8.ビッグデータ講座　概要

ねらい	IoT、AI に欠かせないビッグデータについて学ぶ。また、ワークショップを通じ、その具体的な活用方法について考える。				
開催日程	15 時間（e-learning　3 時間含む）				
受講条件	IT 技術者としての経験が 3 年以上、ICT の基礎知識を持っていること				
学習目標	・ビッグデータの特徴と、実現のための技術要素を理解する。				
	・具体的なビッグデータの活用方法が考えられるようになる。				
	時間	講義	演習	学習概要	学習詳細
カリキュラム概要	0:30	0:30	0:00	ビッグデータ概要	・ビッグデータとは、量、質、種類
					・.ビッグデータ登場の背景
					・.ビッグデータの特性と所在
					・.オープンデータとは
					・.ビッグデータ活用のイメージ（業種・業態）
				ビッグデータとセキュリティ	・.ビッグデータとセキュリティ（個人情報保護）
	0:40	0:40	0:00	ビッグデータ活用を支える技術 クラウド　IoT	・ビッグデータ周辺技術：クラウド・ソーシャルメディア・IoT・匿名化
	2:20	0:50	1:30	ビッグデータ活用を支える技術 データ収集・加工	・データ解析までの流れ ・データ蓄積までの流れ ・データ収集例
					・同期通信と非同期通信
					・処理と保存
					・データ管理
					・従来型のデータベース
					演習　Spark
	1:00	1:00	0:00	ビッグデータコア技術　NoSQL	・NoSQL とは ・NoSQL の基本概念
	1:10	0:40	0:30	ビッグデータコア技術　分散処理	・分散処理 ・Hadoop
					演習　Redis によるデータ管理
	0:40	0:40	0:00	ビッグデータにおけるデータ解析	・ビッグデータにおけるデータ解析
					・一般的な分析手法
	1:10	0:40	0:30	ビッグデータと AI　機械学習	・機械学習アルゴリズム ・画像分析

	4:30	0:40	3:50	ビッグデータ活用のプランニング （ワークショップ）	・以下課題について、データの活用方法のプランニング、及び運用・バックアップ・セキュリティなど懸念点を整理する
					○課題：
					・社内ファイル共有と活用ができていない
					・売上などの可視化や傾向分析ができていない
					・センサーデータや、オープンデータの活用ができていない
					・製品の評判を知りたい
					○データの定義：
					・エクセル、CSV ファイル
					・売上、予算データ
					・気温、天候、外部イベント、内部キャンペーン
合計時間	12:00	5:40	6:20		

8.ビッグデータ講座　詳細カリキュラム

時間	学習項目	学習項目の狙い	詳細内容
0:30	ビッグデータ概要	目的：ビッグデータの概要と個人情報について学ぶ ゴール：ビッグデータを扱う注意点について理解する	[講義]
			ビッグデータとは
			(1)ビッグデータの量
			・ビッグデータの定義と、実際にはどのくらいの容量のものを示すのか例を交えて説明する。
			(2)ビッグデータの質
			・ビッグデータとして蓄積されているデータにはどのようなものがあるのか説明し、事例につても紹介する。
			(3)ビッグデータの種類
			・ビッグデータは３つに分類できることから、分類の特徴を解説し、どのデータがどの分類に該当するか考えさせる。
			(4)ビッグデータが持つ特性
			・５つの V についてビッグデータの特徴を説明する。
			ビッグデータ登場の背景
			(1)ビッグデータ周辺の環境
			・インターネットが実用化され、IoT や SNS、クラウド等の技術の普及がビッグデータの登場にどのような影響を与えたのか説明する。
			ビッグデータの特性と所在
			(1)特性と所在
			・ビッグデータは膨大な量のデータだが、どこに保管されているのかについて、企業や自治体の例を出しながら解説する。
			オープンデータとは
			(1)オープンデータの特徴
			・誰でも自由に利用できるデータとして、どんな特徴を持っているのか、公開先と具体的なデータ例を示して説明する。
			(2)オープンデータ×ビッグデータ活用例
			・オープンデータとビッグデータが一緒に活用されている例を取り上げ具体的に説明する。
			ビッグデータ活用のイメージ（業種・業態）

			(1)ビッグデータが活用される分野
			・活用される分野は多岐にわたることを説明し、また、分野ごとの事例についても紹介する。
			(2)ビッグデータが活用される業界
			・ここでは主に 4 つの業界に分けて活用の事例を紹介し、活用方法について説明する。
			ビッグデータとセキュリティ(個人情報保護)
	ビッグデータとセキュリティ		(1)個人情報保護法
			・個人情報の定義や、2015 年の改正内容について説明する。
			(2)個人情報保護の情勢
			・世界ではどのような動きがいつ頃からあったのか、アメリカや EU の例を挙げて説明する。
			(3)オプトインとオプトアウト
			・第三者提供等にも関わる内容であり、言葉の意味と具体的な例を説明する。
			(4)匿名加工処理の手法
			・技法例を 2 つの手法に分けられ、それぞれがどのような特徴を持つのか、組み合わせることでどうなるのか説明する。
			・仮名化したテーブルの具体例を示し、識別子の削除（仮名化）について説明する。
0:40	ビッグデータを支える技術 クラウド　IoT	目的：ビッグデータとその周りの技術の関係について学ぶ ゴール：周辺技術ごとにビッグデータとどのように関連しているのか理解する	ビッグデータを取り巻く周辺技術：クラウド・ソーシャルメディア・IoT・匿名化
			(1)クラウドサービス
			・なぜクラウドと呼ばれるのか、特徴の説明を交えて紹介する。
			・クラウドサービスは、SaaS や PaaS 等インターネット経由で提供するものの内容により分類されていることを説明。
			・使用上のメリット・デメリットも踏まえてクラウドとそうでない場合の違いを説明する。
			・クラウドがビッグデータの保管場所として有用であることとその理由について、SNS や IoT を例に挙げ説明する。

			(2)IoT
			・IoT の概要、モノとインターネットがつながり、ユビキタスと M2M を包括する IoT の概念について説明する。
			・IoT を構成する技術要素であるセンサー、デバイス、IoT サービスについて、各々の特徴と具体例を説明する。
			・IoT と IoT により蓄積されるビッグデータの関係に付いても説明する。サーバーとネットワーク間のデータのやり取りの流れ等。
			・IoT に使用されるセンサーについて、得られるデータの種類について、MessagePack や JSON の特徴を例に上げながら説明する。
			・デバイスの持つ 2 つの機能と、IoT サービスにおける役割について説明する。
			・IoT サービスのサーバー構成について、役割別に説明する。
			(1)ビッグデータ解析までの流れ
2:20	ビッグデータを支える技術 データ加工・収集		・収集、蓄積、解析の大まかな流れを説明。各々について以下で詳しく説明するため、ここでは大きな流れの説明に留める。
			(2)ビッグデータ蓄積までの流れ
			・解析可能な状態に処理するまでの蓄積の流れについて、クレンジングの役割と種類を説明する。
			(3)データ収集例
			・データ転送の方法について 3 つの例を挙げ、その特徴について各々説明する。
			・デバイスとの通信方法について、同期通信と非同期通信の違いについて説明する。
			(5)ビッグデータの処理と保存
			・ストリーム処理とパッチ処理の 2 種類に分けて、処理の違いと状況に応じどちらを方法を選択するべきかについて、例を交えて説明する。ストリーム処理の具体的な流れと、使用されるフレームワークについても説明。
			(8)クレンジング

			・データ管理におけるクレンジングについて、クレンジング自体がどのような処理で、なんのために行うのか、クレンジングの対象と具体例を交えて説明する。
			(9)ETL と ELT
			・クレンジングに利用される 2 つの処理の加工のタイミングの違いや、SQL 命令で処理が可能どうかについて説明する。
			(7)データレイク
			・従来型のデータ蓄積と比較しながら解説。非構造化データの扱いに適し、多岐に渡る種類のデータをまとめてためておけることや、データ構造の定義をいつ行うのかについて説明する。
			(8)スループットとレイテンシ
			・ビッグデータの処理性能を測る指標について、二つの指標の違いと、どのような場合に使用されるか解説する。
			(13)従来型データベース
			・行指向 DB、列指向 DB、MPP と 3 種類の従来型 DB について説明し、NoSQL との対比につなげる。
			演習 Spark
1:00	ビッグデータとコア技術 NoSQL	目的：ビッグデータの基盤システムを学ぶ ゴール：基本的なビッグデータの基盤システムを理解する	(1)NoSQL とは
			・NoSQL の概要について、従来型データベースとの違いをまじえながら説明する。NoSQL は非構造化データを扱うのに利用されるが、構造化、準構造化データの種類についても説明する。
			(3)NoSQL のメリット、デメリット
			・保存に適するデータの種類や性能向上のために使える手法や、RDBMS との違い等を説明する。
			(4)NoSQL の代表的な種類
			・代表的な種類として 4 つ挙げ、各々について特徴を説明していく。また、各種類により作られた製品についても具体例を紹介。
			(10)NoSQL の基本的概念と技術
			・ここではアーキテクチャから見る分類のうち、マスタ

			型とP2P型について解説する。代表的なデータベースサービスについても例を紹介する。
			・また、どれだけのデータが収容できるのか等のNoSQLに求められる要件についても解説する。
			・整合性について、読み出すデータの状態やネットワークの分断等、複数の概念が存在することを説明し、それぞれのポイントについて説明していく。
			・データ分割技術には整合性を修復する仕組みやデータの隔たりを防ぐ仕組み等、様々な仕組みが使用されていることを説明する。
			・ストレージアウトにおいて、サーバーメモリとサーバーディスクで行われる処理を、図を用いながら説明する。
			(1)分散処理とは
1:10	ビッグデータとコア技術 分散処理		・複数のサーバーで処理を分散する仕組みと、メリット/デメリットについて説明する。
			(3)Hadoopとは
			・Hadoopの概要について説明。Hadoopはフレームワークであるため、内包する基本構成についても解説する。
			(5)HDFS（Hadoop Distributed File System）
			・マスタ型であることからその特徴を解説し、読み書きに関する処理も説明する。
			(8)MapReduce処理
			・MapReduce処理について、Map処理とReduce処理に分けて説明する。
			・論理構造については図を提示し流れの解説を進める。JobTrackerやMapper、Reducer等の役割を確認する。
			・CPUやメモリの計算はリソースマネジャーにより行うことと、リソースマネージャーであるYARNの説明、流れを確認する。
			・Javaではなく SQLを使用するには Hive、高速化には Tez、対話型クエリ実行には Impara と Prestoのように、各製品を使用する利点と概要を

			説明する。
			演習　Redis によるデータ管理
0:40	ビッグデータにおけるデータ解析	目的：ビッグデータで役に立つ分析手法を学ぶ	一般的な統計分析手法
			・度数分布や標準偏差等の一般的な統計分析手法を 5 つ例として用意し、その各々の手法について解説していくほか、相関関数や回帰分析、クラスタリング、テキストマイニングについても利用例等を提示しながら解説し、次項の AI や機械学習の基礎知識として習得させる。
			アドホック分析ツール
			・アドホック分析ツールは、データを可視化するためのツールの 1 つであり、他にインタラクティブ分析に用いられる Jupiter Notebook や Apache Zeppelin などのツールがあることを説明。
			・アドホック分析とは対象的なダッシュボードツールについても説明し、アドホック分析との違い、使い分けを解説する。
			(1)AI の発展
1:10	ビッグデータと AI、機械学習		・ここでは AI 自体が何を指す言葉なのかということと、結論を得るまでの処理の流れ、結論を得るための推論方法について説明する。
			(2)機械学習
			・機械学習の概要と機械学習の種類、使われる用語について解説する。
			(3)教師あり学習と教師なし学習
			・学習方法には入出力データを予め与える方法と入力データの特徴のみを与える方法に二分できるため、それぞれの特徴や用途について解説する。
			(4)機械学習アルゴリズム
			・機械学習で使用されるアルゴリズムの種類について説明する。(1)で学習した回帰分析等も改めて確認する。
4:30	ビッグデータ活用のプランニング（ワークショップ）	目的：ビッグデータの活用方法やセキュリティを学ぶ	演習課題を実施するための手法や注意点に関して説明を行う。
			[演習]

		ゴール：演習を通して具体的な方法を理解する	以下課題について、データの活用方法のプランニング、及び運用・バックアップ・セキュリティなど懸念点を整理する
			○課題：
			・社内ファイル共有と活用ができていない
			・売上などの可視化や傾向分析ができていない
			・センサーデータや、オープンデータの活用ができていない
			・製品の評判を知りたい
			○データの定義：
			・エクセル、CSV ファイル
			・売上、予算データ
			・気温、天候、外部イベント、内部キャンペーン
			・SNS データ

ビッグデータ講座

ビッグデータ概要

目次

第１章　ビッグデータ概要

目次

第２章 ビッグデータとセキュリティ

第１章

ビッグデータ概要

Computer Software Association of Japan

ビッグデータとは

ビッグデータとは

一般的なデータ管理・処理ソフトウエアで扱うことが困難なほど巨大で複雑なデータの集合のこと。
ビッグデータを取り巻く課題の範囲は、情報の収集、取捨選択、保管、検索、共有、転送、解析、可視化等多岐にわたる。これら課題を克服しビッグデータの傾向をつかむことで「ビジネスに使える発見、疾病予防、犯罪防止、リアルタイムの道路交通状況判断」に繋がる可能性がある.

Wikipediaより

つまり、ビッグデータとは、

・従来のシステムでは処理できないほど巨大なデータ

・定型を持たない複雑なデータ

・発見、予防といった新たな価値をもたらし得る、2次元的情報をもたらすデータ

であることが分かります。

CSAJ

・説明の流れ

ビッグデータとは何か、辞書的な説明ではあいまいとしていてはっきりとしていませんが、注意して読むとエッセンスが見えてきます。

・ポイント（絶対に覚えてほしいこと、など）

普通の処理では対応できない量定型なデータではない本来の目的とは異なる使い方により2次効果を得る。

・質問（問いかけ）

このようなデータは身近にありますか。

・補足説明

・ファシリテーションテクニック
　身近な例を考えさせ、それがビッグデータかどうか、隣同士で指摘してもらう

ビッグデータの量

巨大なデータとはどれくらい？

ビッグデータの例を見てみましょう。

データ量を表す単位は、以下の順に1024倍となります。

キロ(KB)<メガ(MB)<ギガ(GB)<テラ(TB)<ペタ(PB)<エクサ(EB)<ゼタ(ZB)

全世界で生成・消費されるデジタルデータの総量

IDC（International Data Corporation）の発表： 59ゼタバイトを超える

出典：https://www.idc.com/getdoc.jsp?containerId=prUS46286020

・説明の流れ

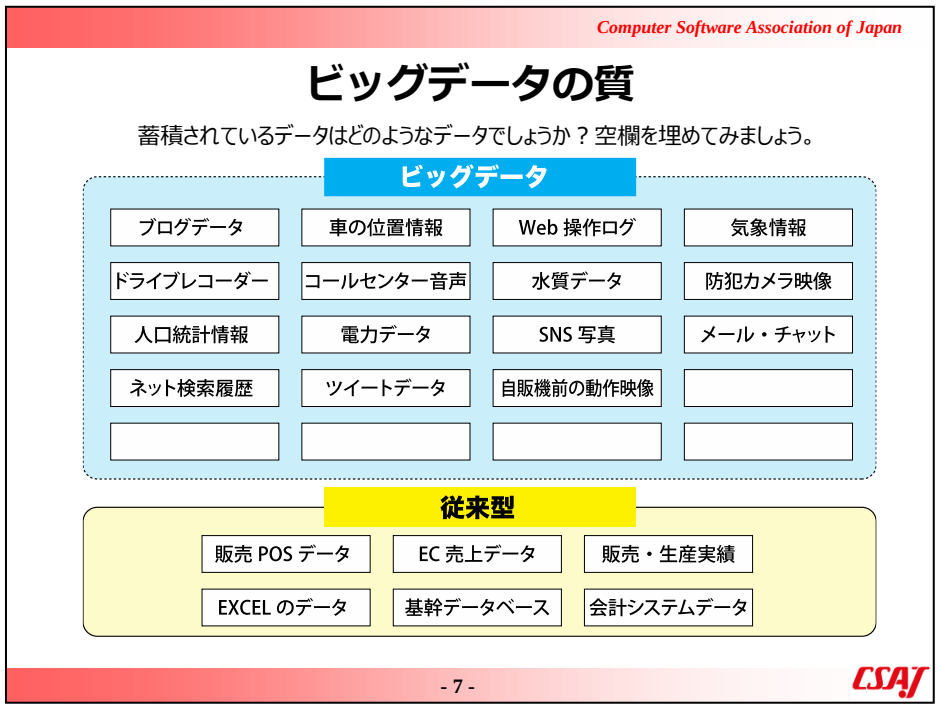
世に言うビッグデータの量はどれくらいでしょうか。

・ポイント（絶対に覚えてほしいこと、など）

ビッグデータの規模はもはや1日TB級のデータを扱うレベル。

・質問（問いかけ）

この規模のデータで思いつくものは？



・説明の流れ

ビッグデータと言えばどのような種類のデータを思い浮かべるでしょうか。

空欄に今まで上げたビッグデータを書いてみましょう。

これと比べて、従来型のデータと決定的に異なる点は何でしょう。

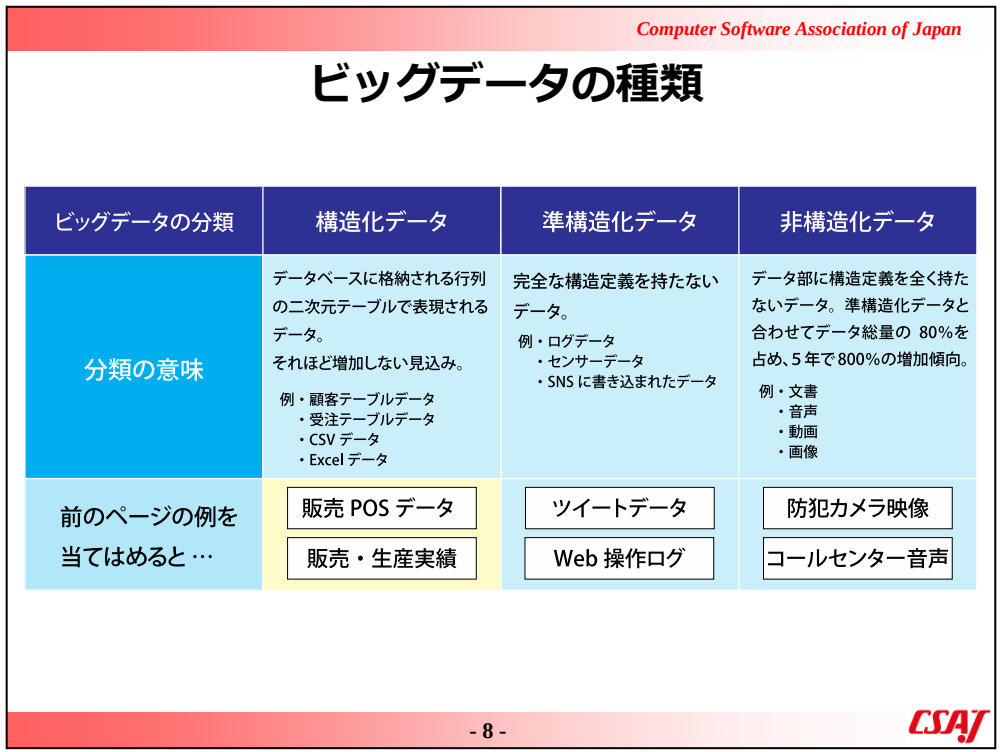
・ポイント（絶対に覚えてほしいこと、など）

ビッグデータでは、定型業務で発生したデータ以外も扱う。

またはそれらを組み合わせて分析する。

・質問（問いかけ）

従来型のデータと決定的に異なる点は何でしょう。



・説明の流れ

ビッグデータには構造化データと準構造化データ、非構造化データがあることを説明します。

特に非構造データの増加率が爆発的であることを強調します。

・ポイント（絶対に覚えてほしいこと、など）

構造化データと準構造化データ、非構造化データの違い

・質問（問いかけ）

前ページのデータを分類してみましょう。

・補足説明
・ファシリテーションテクニック
　分類した内容を隣の方と意見交換する。

・説明の流れ

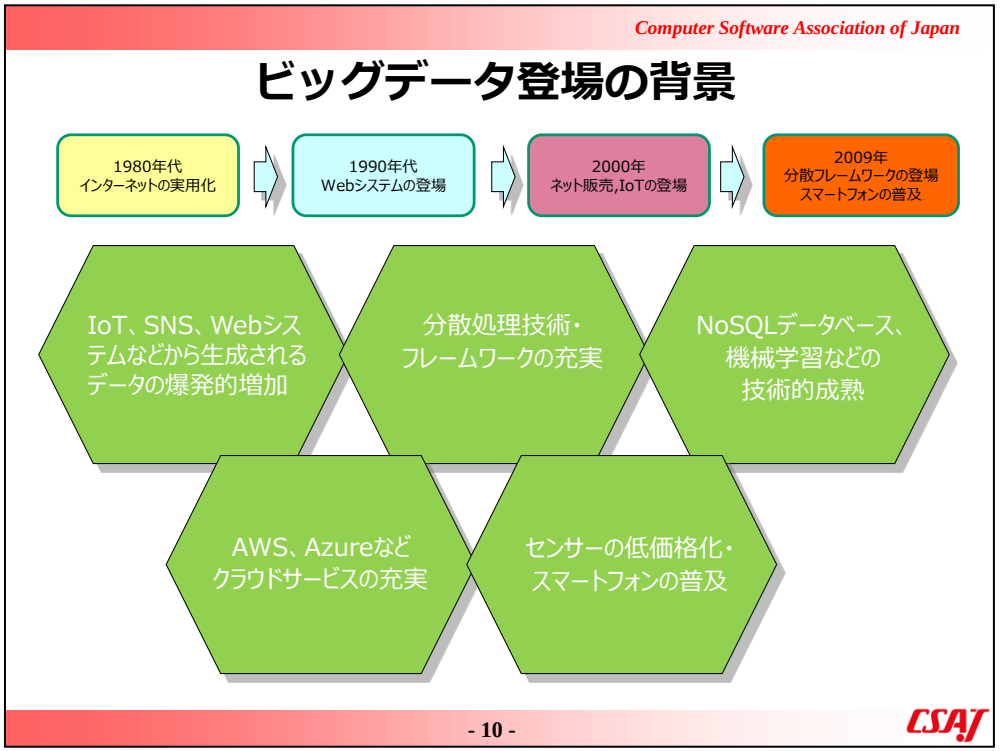
ビッグデータの特性はこれまで述べてきたように、量と多様性（質）があげられますが、もう一つ重要な要素として、リアルタイムにいつでも発生する　データ生成頻度があげられます。

センサーデータやTwitterデータなどは常に発生し、処理をしなければなりません。

これらを合わせビッグデータの３Vと呼びますが、最近では、Veracity、Valueなども合わせて、４V、５Vなどというようになってきています。

・ポイント（絶対に覚えてほしいこと、など）

ビッグデータの３Ｖ



・説明の流れ

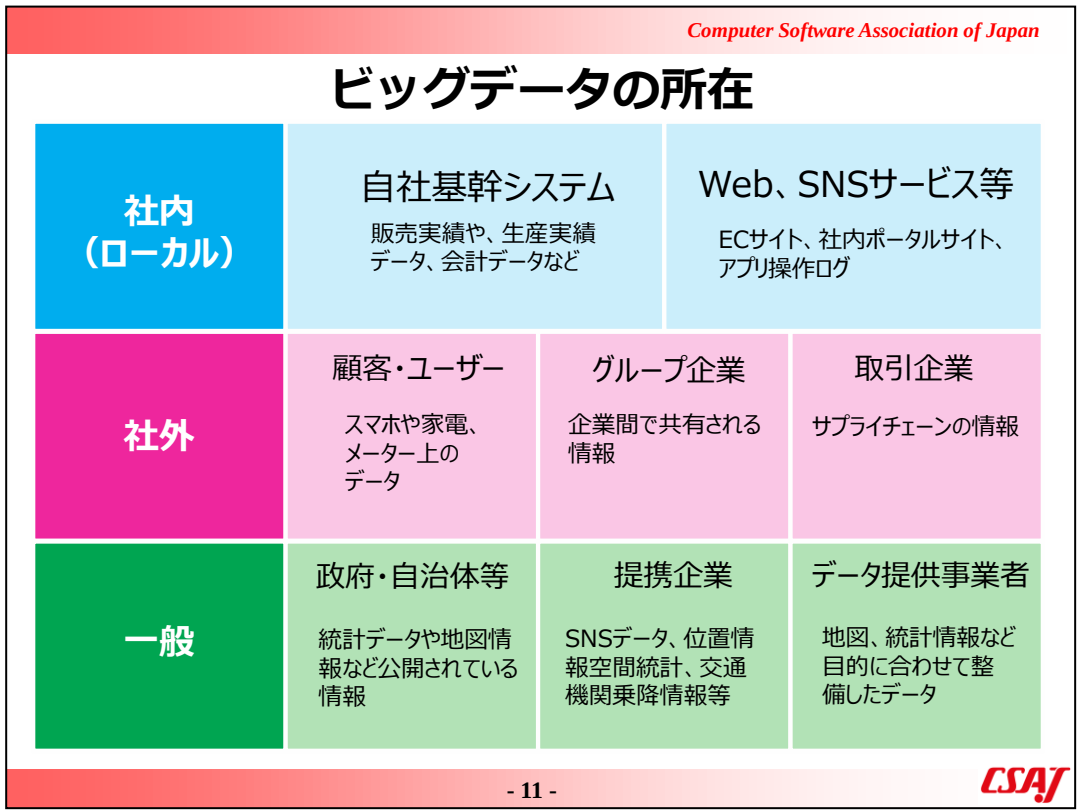
ビッグデータ登場の背景としては、コンピューターの処理能力の向上と、Webシステムの登場が大きいと言えます。

これにより顧客の購買動向や、googleなどの検索内容を補完できるようになり、ビッグデータの道が開けました。

その後の発展は様々な技術により、巨大なデータ処理技術が支えられ続けています。

・ポイント（絶対に覚えてほしいこと、など）

データを保持し続ける状況とそれを分析するニーズ、技術の進歩がマッチして初めて、ビッグデータが登場する契機が生まれた。



・説明の流れ

ビッグデータはどこにあるものでしょうか。取引先から渡ってくるデータや今まで保存しかしていなかったログデータ、社外Webサイトなど、自社内はもちろん自社ローカル以外にも眠っている場合があります。

また公共団体が提供するデータや、事業者が提供する商用データなどもビッグデータである場合もあります。

・ポイント（絶対に覚えてほしいこと、など）

ビッグデータはあらゆるところに散在している。

・質問（問いかけ）

身の回りや企業内で眠っているビッグデータを上げてみましょう。

Computer Software Association of Japan

オープンデータとは

特徴	■ 誰でも入手可能で、自由に利用・再配布できる状態で存在する ■ 特許・著作権に制限がない ■ コンピューターから利用できる状態となっている
公開主体	■ 政府 ■ 地方自治体 ■ 研究機関・大学 ■ 民間企業
具体例	■ 国勢調査データ（政府統計の総合窓口「e-Stat」） ■ 公共施設やAEDの位置データ ■ 気象データ ■ 有志により作られた地図データ（OpenStreetMap など） 「行政と市民によるオープンデータ共創支援プラットフォーム（LinkData）

CSAJ

・説明の流れ

官公庁が公開しているデータなどをオープンデータと呼ぶことがありますが、これもビッグデータの一つです。

具体的な定義は何でしょうか。

・ポイント（絶対に覚えてほしいこと、など）

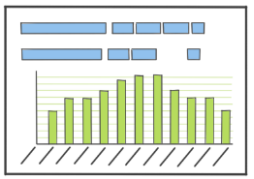
オープンデータは誰でもいつでも利用・再配布できる状態にあるデータ

・質問（問いかけ）

身近なオープンデータをあげてみましょう。

オープンデータ×ビッグデータ活用例

・過去のTwitterなどのSNS上の書き込み + 販売データ
→ 相関を調べ、売上の増減や欠品可能性を予測する。



・自社の販売データ + 気象データ
→ 気象変化と売上推移の相関を見出し、予測を行う。

・医療施設の位置データ + 患者の郵便番号のデータ
→ 来院マップを作成し、診療費ごとの外来状況を分析することで、地域医療に関して重点的な連携、促進を図る。

・国勢調査などの人口統計情報 + 将来の人口推計
+ ターゲット層の世帯が多数存在する地域の売上相関
→ 重点的に販売を行う地域を模索する。

・説明の流れ

オープンデータと、ビッグデータを掛け合わせると、それを単体で使うより大きな価値を生み出す場合があります。

・ポイント（絶対に覚えてほしいこと、など）

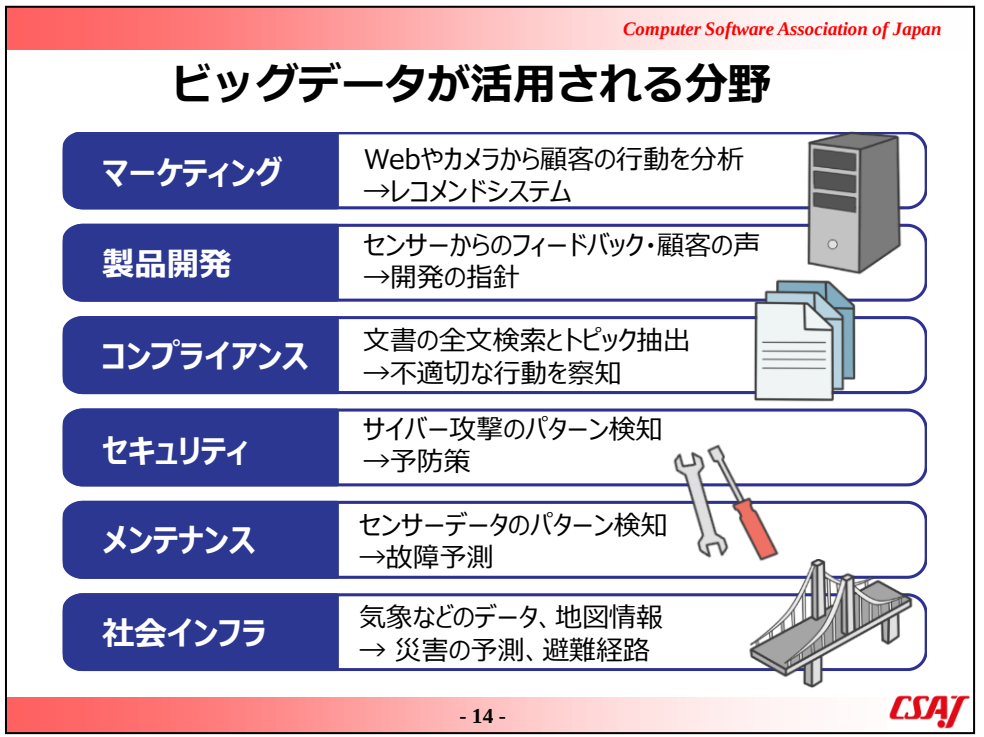
オープンデータ単体では価値を生み出さない場合がある。

・質問（問いかけ）

気象データ単体で利用できるケースとビッグデータを利用した場合の利用ケースをそれぞれ考え、価値に違

いがあるかを考えてみましょう。

・補足説明
公共施設マップなど、オープンデータそのもので価値があるデータも存在します。



・説明の流れ

ビッグデータはどのような分野で活用されているでしょうか。

・ポイント（絶対に覚えてほしいこと、など）

ビッグデータ適用分野は多岐に渡り、これまでのアプリケーションの枠を超え、全く新しいデータ活用をもたらします。

・質問（問いかけ）

今まで耳にしたビッグデータの活用分野を上げてみましょう。

ビッグデータが活用される業界

	運輸	金融	医療・健康	製造
利活用事例	1. 航空機チケットの割引サービスの改善：Web販売サイトでの購入傾向を分析 2. 渋滞予測：スマホGPSの情報を分析 3. トラックの最適な輸送ルート：過去の情報から算出	1. 金融商品の提案・開発：数千万件の顧客情報から、決済や資産運用の動きを分析 2. 保険サービスの開発：車の走行距離の情報を元に保険料を定めるサービスを開始	1. インフルエンザ対策：SNS上のコメントや検索キーワードから、広がりを検知 2. メディケア（アメリカの公的保険）のデータ公開：公的機関が分析して公開	1. 品質改善のための最善策の策定：生産工程に関する多数の情報を収集、分析 2. 製品開発：建機にセンサーを設置し故障時の稼働環境を分析してフィードバック。
効果	販売促進 人流動態分析	新サービス開発	兆候把握・ 情報提供	品質向上

・説明の流れ

ビッグデータ活用の具体的な例です。

新聞などで日々掲載される技術革新が、ビッグデータによって支えられているケースも増えています。

第2章
ビッグデータとセキュリティ

ビッグデータとセキュリティ（個人情報保護）

個人情報とは

個人情報の保護に関する法律 第二条 第１項 第一号において、次のように定義されています。

「生存する個人に関する情報であって、」「当該情報に含まれる氏名、生年月日その他の記述等」「により特定の個人を識別することができるもの（他の情報と容易に照合することができ、それにより特定の個人を識別することができることとなるものを含む。）」

・説明の流れ

ビッグデータを扱う際に、個人情報が含まれているため、利用を断念するケースがあります。

それでは個人情報とはそもそもどのような定義でしょうか。

・ポイント（絶対に覚えてほしいこと、など）

データ単体では個人を特定できない情報であっても、他のデータと組み合わせることによって、個人を特定できる場合がある。

・たとえ話、小ネタ

過疎地の郵便番号や、希少な病名、顧客IDと顧客マスター、写真とGPS情報など

個人情報保護法

個人情報保護法

・成立：2003年（平成15年）5月23日
・施行：即日（但し、一般企業に直接関わり罰則を含む第4～6章を除く）
・全面施行：2005年（平成17年）4月1日 … 成立の2年後

個人情報取扱事業者

・個人情報を個人情報データベース等として所持し事業に用いている事業者のことをいう。

・個人情報保護法および同施行令により、取扱件数に関わらず、個人情報取扱事業者とされるようになった。

・主務大臣への報告や、それに伴う改善措置に従うなどの適切な対処を行わなかった個人情報取扱事業者に対しては、刑事罰が科される。

・説明の流れ

日本では個人情報保護法はいつ、どのような内容で施行されたのでしょうか。

・ポイント（絶対に覚えてほしいこと、など）

個人情報の改善措置に従わない場合は、事業者に対して罰則も科される。

Computer Software Association of Japan

個人情報保護法改正

個人情報保護法　2015年の改正内容

・これまで対象外だった、5,000人分以下の個人情報を取り扱う小規模な事業者に対しても、改正法が適用されるようになった。

・個人情報を取得する場合、予め本人に利用目的を明示することが必要となった。

・個人情報を他企業などの第三者に提供する場合、予め本人から同意を得ることが必要となった。

・オプトアウトには、個人情報保護委員会への届出が必須となった。
更に、第三者提供の事実、その対象項目、提供方法、望まない場合の停止方法などを、全て予め本人に示さなければならなくなった。
※オプトアウト：本人の同意を得ないで個人情報を提供できる特例のこと。

・「人種」、「信条」、「病歴」といった「要配慮個人情報」は、オプトアウトでは提供できないこととされた。

CSAJ

・説明の流れ

個人情報保護法は2015年に改正され、より範囲が明確になり、運用方法も明確に定義されました。

・ポイント（絶対に覚えてほしいこと、など）

個人情報を取り扱う場合には、何らかの形で個人の同意が必要。

個人情報保護の情勢

・1980年　プライバシー保護と個人データの国際流通についてのガイドラインに関するOECD理事会勧告（OECDプライバシーガイドライン）
（OECDの34加盟国）

①収集制限　②データ内容　③目的明確化　④利用制限
⑤安全保護措置　⑥公開　⑦個人参加　⑧責任
の８原則からなる。

・1995年　EUデータ保護指令　（EUの28構成国）

EUおよび英国においては、十分なデータ保護レベルが確保されて
いない第三国への個人データの移動を禁止する。

・2003年　個人情報保護法　（日本）

個人情報を扱う事業者に対し、個人情報の適切な取り扱いを求める。

・2012年　消費者プライバシー権利章典　（アメリカ）

①個人によるコントロール　②透明性　③背景情報の尊重
④セキュリティ　⑤アクセスと正確性　⑥適切な範囲の収集　⑦説明責任
の7つの権利を定める。

・説明の流れ

海外の個人情報に対する規制どのようになっているか見てみましょう。

・ポイント（絶対に覚えてほしいこと、など）

海外では個人の意思表示によりデータの削除もできる仕組みであるなど様々な規制が存在する。

個人情報保護法の情勢

・EU一般データ保護規定（GDPR）が可決（2016年4月 EU）
　データポータビリティ権が提唱される。
　→ 域外適応につき、日本の事業者に影響が出る。

・EU - USプライバシーシールドに米国と欧州委員会が合意（2016年2月 米国）
　スノーデン事件を受けて無効化されていたセーフハーバーの後継。
　商務省とFTCに強い権限が与えられ、企業に対して自主規制を求める機運が高まった。

・APEC 越境プライバシールールシステム（CBPRs）への参加（アジア）
　日本に関しては、2014年にJIPDECがCBPR認証機関に認定された。
　→ 2016年6月1日から申請受付開始。

・個人情報保護委員会が発足（2016年1月 日本）
　個人情報保護法の改正を受け、政府の第三者機関として設立した。

・一般財団法人情報法制研究所（JILIS）が設立（2016年5月 日本）

・説明の流れ

　海外の個人情報保護の情勢をもう少し詳しく見てみましょう。

・ポイント（絶対に覚えてほしいこと、など）

　EU法のGDPR（General Data Protection Regulation：一般データ保護規則）では、IDなども個人情報扱いであったり、EU域外に個人情報を持ち出せない、規定に違反した場合などは制裁金が科せられる場合もあり、注意が必要。

　特にEU国内からアクセスが発生するWebサイトを運営している場合も規定に反する状況が発生している場合も考えられる。

オプトインとオプトアウト

オプトイン（事前承認）

明示的な同意が無い限り、個人情報やプライバシー情報は収集されないような仕組みのことを言います。

例・ショッピングサイトからのセール情報に関するメールの送付を許可する。
・個人情報の収集・利用を含むサービスの利用規約に同意する。

オプトアウト（事後承諾）

オプトインとは反対に、明示的に拒否していない限りは同意したものとみなし、明示的な拒否があった場合に個人情報やプライバシー情報の利用が停止されるような仕組みのことを言います。

例・Webサイトにおけるクッキーを用いた行動追跡
・ショッピングサイトにおける購買履歴の削除

・説明の流れ

個人情報の許諾を個人から得る方法にはどのようなものがあるでしょうか。

・ポイント（絶対に覚えてほしいこと、など）

オプトインは事前承認、オプトアウトは事後承認

匿名加工処理の手法

以下のそれぞれの手法を組み合わせることで、より強固な匿名化が実現されます。

技法大部類	No.	技法例	概要
摂動法	1	K-匿名化	同じグループ内に、同じ属性のユーザが「K人以上いる」状態を作り出す。
	2	L-多様性	漏えいさせたくない属性が同じグループ内で「L種類以上ある」状態を作り出す。
	3	T-近接性	マイナー属性を持つグループが生まれるなど、属性値の分布に偏りが出てしまう場合に、グループの分割や一般化を行う。
	4	差分プライバシー	2006年に提案された新しい手法。元のデータベースにノイズを足した別のデータベースを用意し、守りたいレコードを特定しづらくする。
暗号法	5	質問監査	データベースへのアクセス者に質問を投げかけ、答えられれば、アクセスに対する回答を返す。
	6	秘密計算	関係者全員が、自社データを他人が読めないように変換し、秘密計算のシステムへ投入する。そのシステムの管理者が、秘密計算の結果を求め、関係者に回答する。
	7	準同型性公開鍵暗号を用いた暗号プロトコル	遺伝子データなど、加工してしまうと、そもそも分析できなくなるデータを処理するときに活用。検索者の検索クエリ、データベース、その回答それぞれを暗号化する。分析者が元データにふれずとも、望む解析結果が得られる。

出典：中川裕志『プライバシー保護入門：法制度と数理的基礎』（2015年）

・説明の流れ

個人を特定できないようにデータを加工する事を匿名加工処理と呼びます。

具体的にどのような手法で匿名化を実現するか見てみましょう。

・ポイント（絶対に覚えてほしいこと、など）

K-匿名化や、L-多様化はデータそのものを加工し、もしくはレコードを増やし、個人を特定できないようにする技術である。

K=XXなどの数値を大きくすると安全性は高まるが、大きくしすぎると統計上や機械学習上のノイズになる

場合があり、正しい結果が得られなくなる場合もあるため注意が必要。

識別子の削除（仮名化）

個人の識別・特定に直結するカラムを削除して、仮名化を行います。

No.	ZIPコード	年齢	職業	病状
1	13068	28	ダンサー	心臓病
2	13068	29	技術者	心臓病
3	13053	21	法律家	感染症
4	13053	23	技術者	感染症
5	14853	31	技術者	風邪
6	14853	37	作家	風邪
7	14850	36	法律家	がん
8	14850	35	技術者	がん

準識別子　　漏えいさせたくない属性

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

・説明の流れ

仮名化は個人の氏名や住所などの情報を仮名と置き換えるもしくは削除することによって、データから直接個人を特定することができないようにすることを言います。

K-匿名化したテーブル

次に、再特定・識別につながる「職業」を秘匿した上で、「年齢」、「病状」の列に「同じ値が少なくとも2つ以上は存在する状態」のテーブルを作ります。

No.	ZIPコード	年齢	職業	病状
1	13068	28-29	*	心臓病
2	13068	28-29	*	心臓病
3	13053	21-23	*	感染症
4	13053	21-23	*	感染症
5	14853	31-37	*	風邪
6	14853	31-37	*	風邪
7	14850	35-36	*	がん
8	14850	35-36	*	がん

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

・説明の流れ

K-匿名化は必ずK=数値で表された数以上レコード（行）が存在するようにデータを加工する事です。

同じ保護属性の組み合わせを持つレコードが、少なくともk個存在し、保護属性からの識別がk人未満に絞り込めない状態になります。

L-多様化したテーブル

「ZIPコード」と「年齢」を曖昧にして、「どのレコードを取り出しても、2種類の「病状」が存在する状態」になるようにします。

No.	ZIPコード	年齢	職業	病状
1	130**	21-29	*	心臓病
2	130**	21-29	*	心臓病
3	130**	21-29	*	感染症
4	130**	21-29	*	感染症
5	148**	31-37	*	風邪
6	148**	31-37	*	風邪
7	148**	31-37	*	がん
8	148**	31-37	*	がん

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

・説明の流れ

l-多様性は、同じ保護属性の組み合わせを持つレコードが、少なくともk個存在し、かつ対応する非保護の属性情報の値が少なくともｌ種の”良い”多様性を持つことで、属性推定が起こらない状態です。

具体的には似たようなレコードを追加して、個人を推定できない状況を作り出します。

ビッグデータ

ビッグデータ活用を支える技術

目次

第１章　ビッグデータを支える技術　クラウド、IoT

目次

第２章　ビッグデータを支える技術　データ収集・加工

目次

第３章　ビッグデータコア技術　NoSQL

目次

第４章　ビッグデータコア技術　分散処理

目次

第５章　ビッグデータにおけるデータ解析

目次

第６章　ビッグデータとAI、機械学習

第１章

ビッグデータを支える技術

クラウド、IoT

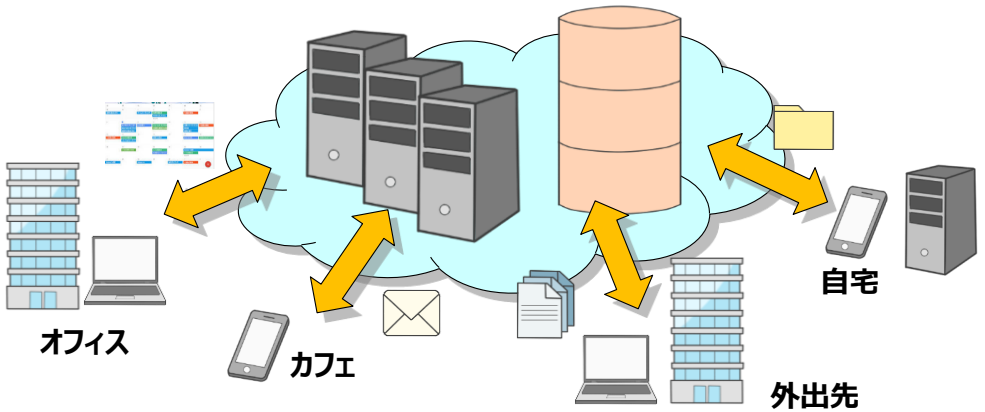
Computer Software Association of Japan

ビッグデータ周辺技術　クラウド

クラウドコンピューティングとは

メールやグループウェア、その他様々なサービスをインターネット上で提供し、インターネット上にデータまでをも保存するようなサービス形態のことをいいます。

アプリケーションの実行場所が、雲（クラウド）のようにどこにあるのかが分からないモヤモヤとした場所にあることから、このように呼ばれています。



CSAJ

・説明の流れ

ビッグデータを支える周辺技術を、学びましょう。

クラウドサービスはビッグデータ登場の背景にも密接にかかわっており、ビッグデータのそもそもの置き場所などでも広く活用されています。

・ポイント（絶対に覚えてほしいこと、など）

クラウドはどこに存在するかわからないデータセンター。

データの共有や業務アプリサービスなど様々なサービスを、資産を持たずに提供できる。

・質問（問いかけ）

そもそもクラウドサービスはどのような用途で使われ、なぜクラウドと呼ばれているのだろうか。

クラウドサービスの種類

クラウドサービスの種類

・SaaS（Software as a Service）
インターネットを経由してソフトウェアパッケージを提供するサービスのこと。
アプリをPCにダウンロードしなくても、Webなどのブラウザ上で利用することができる。
例・メール　・カレンダー　・チャット

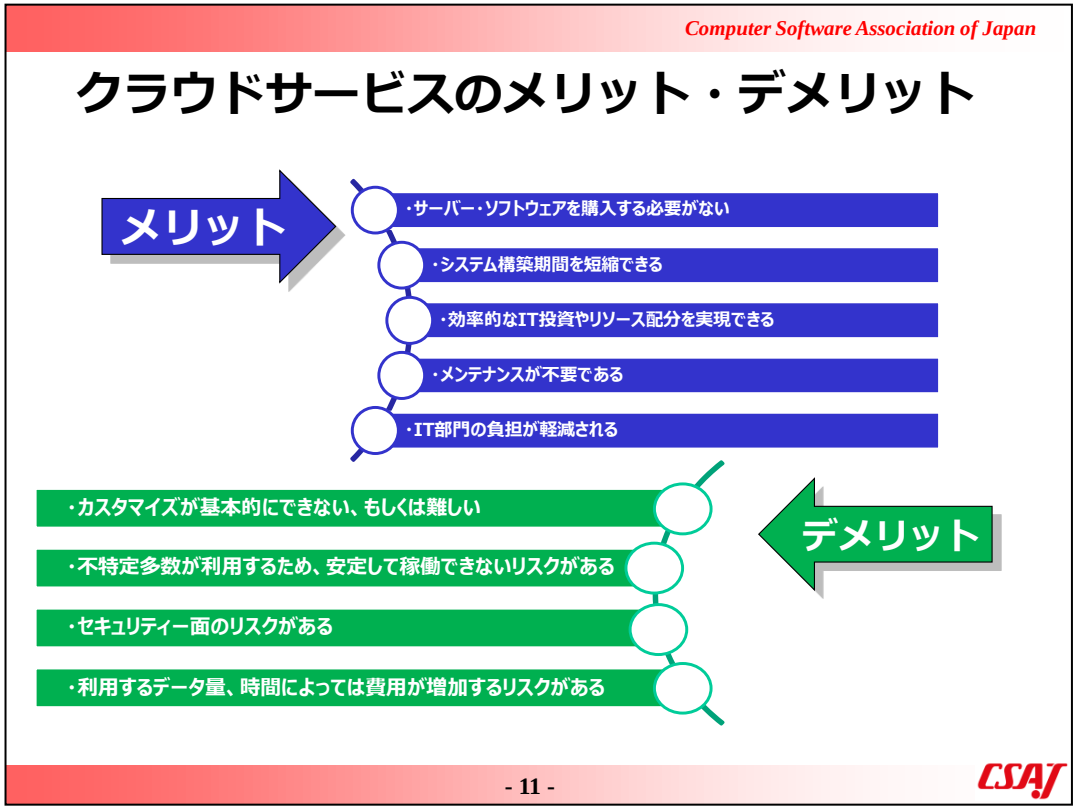
・PaaS（Platform as a Service）
インターネットを経由してアプリの開発・運用環境全体を提供するサービスのこと。システム管理者、開発者向けである。

・HaaS / IaaS（Hardware as a Service / Infrastructure as a Service）
インターネットを経由してハードウェアや回線などのインフラを提供するサービスのこと。ユーザーはハードウェア資産を所有することなく、仮想サーバーやストレージ（外部記憶装置）を利用することができる。

・説明の流れ
クラウドには様々な利用形態の種類があります。
どのようなものがあるのか見てみましょう。

・ポイント（絶対に覚えてほしいこと、など）
SaaSはアプリも提供するソフトウェアサービス。
IaaSは環境を貸し出して好きなようにシステムを構築できるサービス。

・補足説明
パブリッククラウドは、一般、不特定多数のユーザーに公開されるクラウド環境。
プライベートクラウドは1社のみに閉じたクラウド環境。



・説明の流れ

クラウドのメリットとデメリットには、どのようなものがあるでしょうか。

サーバーの調達や、メンテナンスから解放される一方、カスタマイズの難しさや、不特定多数のユーザーの目にさらされる危険もあるため、個人情報の扱いがあるデータの運用には、注意が必要です。

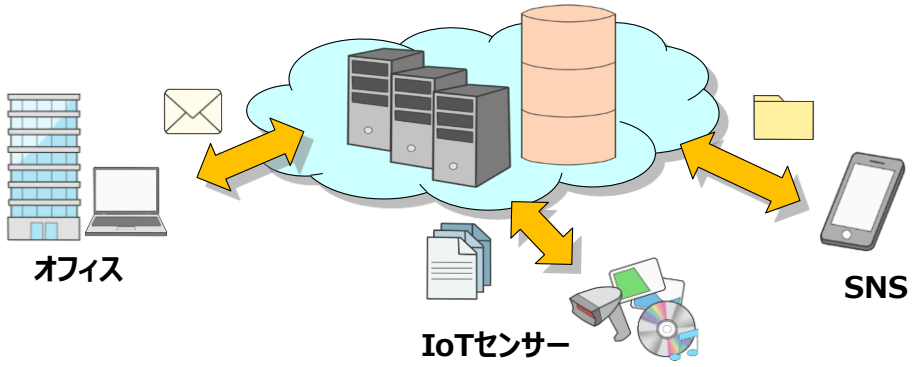
クラウドコンピューティングとビッグデータ

クラウドサービスに蓄積されるビッグデータ

クラウドサービスは、ビッグデータの蓄積場所に有力な選択肢です。

実際、以下はビッグデータがクラウドに保存されています。

・GREE、mixi、Facebook、Twitterなどに代表されるSNSのユーザーデータ
・IoTのようにセンサーなどで発生したデータ



・説明の流れ

ビッグデータの保存先として、サーバーサイジングを自由に変更できるクラウドサービスは有力な選択肢の一つとなります。

IoTセンサーデータのデータ収集もクラウドサービスベンダーが担う状況も整いつつあります。

ビッグデータ周辺技術　IoT

IoTとは

IoT ＝ Internet of Things　「モノのインターネット」

ここで、「モノ」とは、ネットワークに繋がるあらゆる物のことです。
従来はインターネットとは関係のなかったもの、例えば、

・眼鏡　・服　・時計　・冷蔵庫　・電力メーター　・自動車
・太陽光パネル　・家　・スマートフォン

もすべて「モノ」です。

IoTとは、モノがネットワークに繋げられることによって、モノとインターネットが相互に情報交換をできるようになった状態のことをいいます。

・説明の流れ

センサーデータを分析する際に耳にするIoTという言葉は何を意味する言葉でしょうか。

・ポイント（絶対に覚えてほしいこと、など）

モノが双方向でインターネットに繋がりデータのやり取りができる状態をIoTと呼ぶ。

IoTとは

・Internet of Thingsという用語は1999年、ケビン・アシュトン（イギリス）によって初めて提唱されました。

・当初はRFIDによる商品管理システムをインターネットに例えたものでした。

→ 徐々にスマートフォンやクラウドコンピューティングが普及。

→ IoTはモノ自体がインターネットを形作るという環境全体のことを表す概念として捉えられるようになる。

・IDC（ICT市場調査会社）による定義
「IP接続による通信を人の介在なしにローカルまたはグローバルに行うことができる識別可能なエッジデバイスから成るネットワークのネットワーク」

・説明の流れ

IoTという言葉はいつ頃から呼ばれ、定着したのでしょうか。

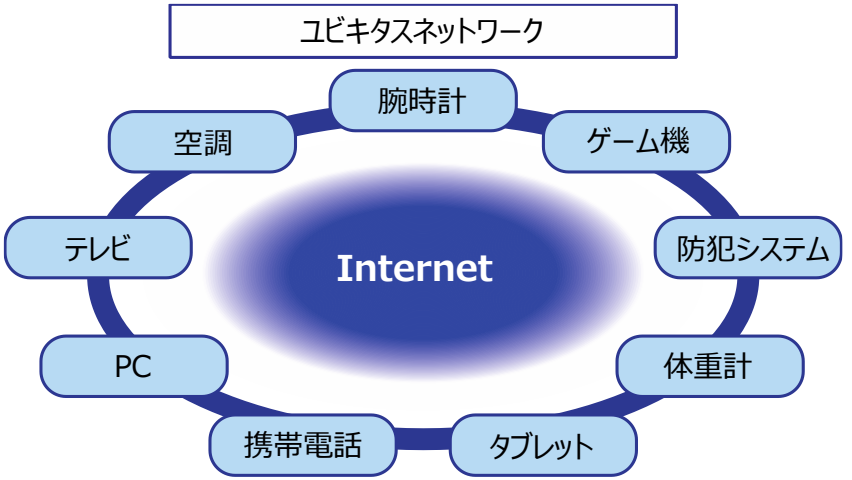
・ポイント（絶対に覚えてほしいこと、など）

IoTは2000年頃から呼ばれ、モノ自体がインターネットを形作る広い概念として捉えられている。

IoTとユビキタス社会

ユビキタスネットワークとは

いつでもどこでもインターネットを利用できるという概念のことをいいます。
1988年ゼロックス社パロアルト研究所のマーク・ワイザーにより提唱されました。



・説明の流れ

2000年初頭から現れたユビキタスネットワークとは何でしょうか。また、IoTとの違いは何でしょうか。

・ポイント（絶対に覚えてほしいこと、など）

ユビキタスネットワークの中心は人であり、IoTのモノとモノとの相互制御とは異なる概念。

IoTとユビキタス社会

IoTとユビキタスの違い

・IoT：モノとモノが相互に制御し合っている状態を表す。
　　…「モノ」を中心とした概念

・ユビキタス：ユーザーが時間や場所にとらわれずインターネットに繋がって様々なサービスを受けられる状態を表す。
　　…ユーザーという「人」を中心とした概念

参考

ユビキタス (ubiquitous) は、遍在（いつでもどこでも存在すること）をあらわす言葉。

パロアルト研究所のマーク・ワイザーが、1991年の論文『The Computer for the 21st Century』にて、コンピュータやネットワークなどの遍在を表す意味合いで用いた。以来、ユビキタスコンピューティングやユビキタスネットワーク、さらにはそれらが当たり前になった社会を指す「ユビキタス社会」の意味で用いられるようになった。

Wikipedia

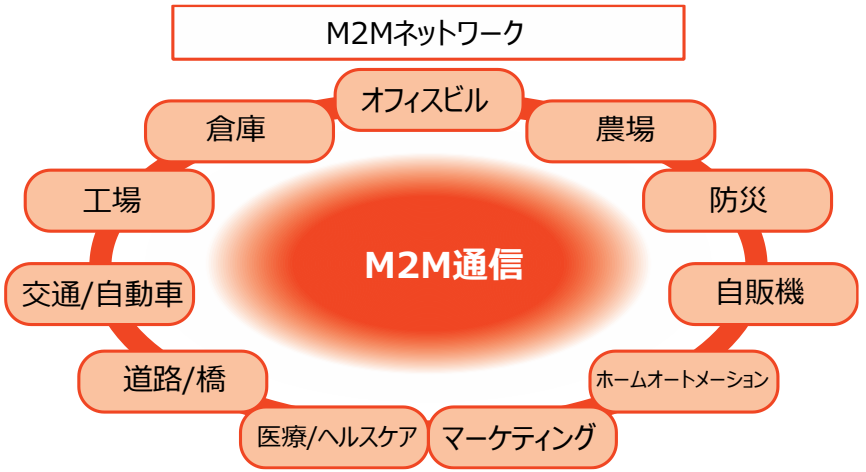
・説明の流れ

　前のスライドを補足する形で、ユビキタスについて説明を加える。

IoTとM2M

M2M = Machine to Machine

機械同士が、人を介在しないで情報をやり取りするシステムのことをいいます。



・説明の流れ

M2MとIoTの違いは何でしょうか。M2Mは人が介在しない機械同士の連携です。

IoTは人も介在するセンサーも対象となるだけでなく、結果を見るのも人である場合もあるため、M2Mの概念だけではIoTを説明することはできません。

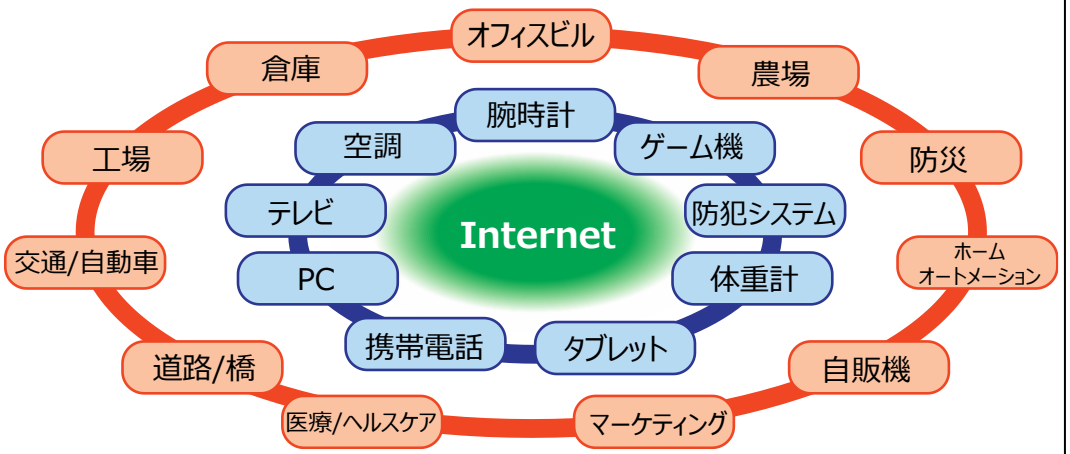
・ポイント（絶対に覚えてほしいこと、など）

M2Mは人が介在しない機械同士の連携

IoTとM2M

IoTはユビキタスとM2Mを包括する概念といえます

IoT社会
Machine to Machine、Human to Machine、Human to Human



・説明の流れ

IoTはユビキタスとIoTを合わせたような概念です。

人とモノ、モノとモノ、人と人を繋ぐ総合的な社会インフラになる可能性を秘めています。

IoTを構成する技術要素

・センサー：物理的な現象を検知し、電気信号として出力する装置。

・デバイス：センサーが組み込まれることによって、ネットワークに接続された装置やモノ。

例　スマホ、時計、メガネ

・ネットワーク：デバイスをIoTサービスに繋ぐ、あるいはデバイス同士を繋ぐことでデータを共有、処理するシステム。

・IoTサービス：①デバイスとのデータの送受信 … IoT
②データの処理と保存 … ビッグデータの技術的守備範囲を行うサービス。

・データ分析：蓄積したデータについて統計分析や機械学習を行う。

→最適な判断や行動方針を導き出す。

・説明の流れ

IoTとは何なのでしょうか。まず、技術要素から切り込んでみましょう。

IoTには、基本的に、センサー、デバイス、ネットワーク、サービス、データ分析がセットになってIoTサービス全体を形作っています。

それぞれの技術要素は独立していることが多く、技術要素の組み合わせによって、IoTを実現しています。

データの処理と保存はまさにビッグデータの技術的守備範囲となり、IoTには欠かせない技術要素です。

・ポイント（絶対に覚えてほしいこと、など）

IoTの実現にはビッグデータ技術が必要。逆にIoTを知ることでビッグデータの現実を知ることになる。

IoTを構成する技術要素　センサー

センサー

物理的な現象を検知し、電気信号として出力する装置のことをいいます。
多くの場合、一つのデバイスに対して複数のセンサーが埋め込まれています。

・画像センサー：光を捉えて処理することで、画像や動画を撮影する。
　赤外線を検知して画像処理するものもある。
・光センサー：光の強度を測定する。
・温度センサー：温度を測定する。
・湿度センサー：湿度を測定する。
・振動/速度/加速度センサー：機器の振動や速度、加速度を測定する。
・地磁気センサー：地磁気を検出することで、方角を計測する。
・ジャイロセンサー：デバイスの傾きを検知する。
・音声マイク：機器が発する音や、人の声などの音声を収集する。

・説明の流れ

IoTのセンサーにはどのようなものがあるか見てみましょう。

データの中には、画像データや、音声データのような非構造データや準構造化データ、バイナリーデータが存在します。

IoTを構成する技術要素　センサー

センサーの代表的なデータフォーマットとしては、
・XML　・JSON　・MessagePack
があります。

XML
```
<xml>
  <info>
    <id>12996</id>
    <name>RoomSensor</name>
    <date>20170123112255</date>
  </info>
  <data>
    <temperature>27.8</temperature>
    <humid>72</humid>
  </data>
</xml>
```

人が読んで分かりやすい
データ量が多い

JSON
```
{"info":{"id":123,
          "name":"RoomSensor",
          "date":20170123112255
         },
 "data"{"temperature": 27.8,
          "humid":72
         }
}
```

データ量が少ない

いずれも各言語のライブラリが充実していますが、文字データであることから、
パース（解析）をしないとプログラムで利用できません。
MessagePackは、バイナリデータをそのまま扱いたい場合、有利です。

・説明の流れ

センサーデータの基本的なフォーマットを見てみましょう。

タグ構造のXMLや、Webサイトでよく使われるJSON形式もありますが、MessagePackのようなバイナリーデータも存在します。

・ポイント（絶対に覚えてほしいこと、など）

センサーデータは、構造化データでないことが多い。

IoTを構成する技術要素　センサー

MessagePack

・センサーの代表的なデータフォーマットの一つ。
・JSONと似た形式だが、値はバイナリのままである。
・軽量でプログラム間処理に向いている。

MessagePackの特徴

・シリアライズ（データの直列化）、デシリアライズ：非常に高速
・シリアライズされたデータのサイズ：小さい
・フォーマット定義：不要
・ストリーム処理：可能

JSONとMessagePackの比較

JSON　{“a”:null,”b”:10,”c”:[20],”d”:”30"}　… 35byte

→ MessagePack に変換すると…

84 a2 61 c0 a2 62 0a a2 63 91 14 a2 64 a2 33 30　… 16byte

・説明の流れ

MesssagePackの形式を見てみましょう。

人が見て理解できる形式ではありませんが、データ量が小さく、変換が高速であることが魅力です。

・ポイント（絶対に覚えてほしいこと、など）

IoTデータにはバイナリーデータも存在する。

IoTを構成する技術要素　デバイス

デバイス

センサーが組み込まれることによって、ネットワークに接続された装置、モノのことをいいます。例えば、スマホ、時計、メガネはいずれもデバイスです。

デバイスの２つの機能

・センシング：センサーを利用して、デバイス自身や周りの環境の状態を収集し、IoTシステムに通知すること。

例・画像センサーによる人の有無の検知
・スマホの位置情報や加速度の計測

・フィードバック：システムからの通知を受け、指示や動作をもとのシステムに返すこと。次のような方法がある。

・可視化：センシング結果表示、デバイスの管理、画面表示
・通知：システムが判断した結果を画面に表示
・制御：デバイス自身や環境の状態そのものを変更

・説明の流れ

IoTにおけるデバイスとは何でしょうか。

スマホやウォッチに搭載されていて、センサーとネットワーク通信機能もデバイスとして捉えられます。

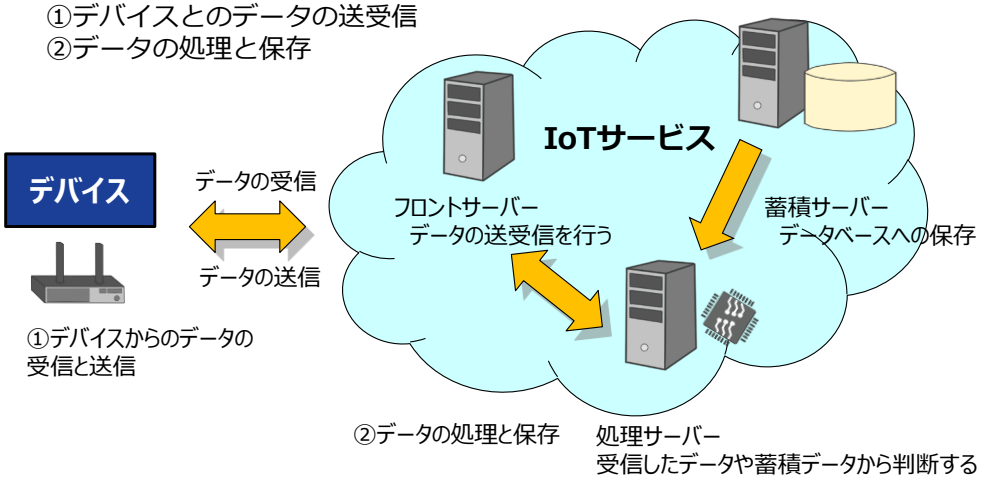
センサーとネットワーク接続を併せ持った機器を指します。

IoTを構成する技術要素　IoTサービス

IoTサービス

以下を行うサービスのことを指します。

①デバイスとのデータの送受信
②データの処理と保存



・説明の流れ

IoTサービスはデバイスから送られてきたデータを受信したり、蓄積、分析したり、サービスを行ううえで必要な処理を行います。

ビッグデータを処理することも役割の一つになります。

・ポイント（絶対に覚えてほしいこと、など）

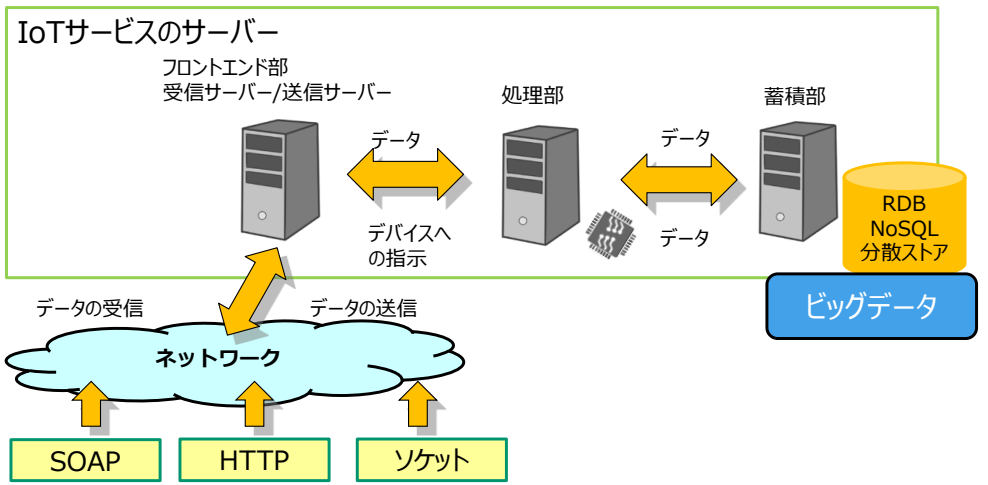
IoTサービスでは、バックエンドでビッグデータも扱う。

IoTを構成する技術要素　IoTサービス

サーバー構成

IoTサービスの役割は、フロントエンド、処理、蓄積の大きく３つに分けられます。

蓄積されるデータは膨大な量となります。



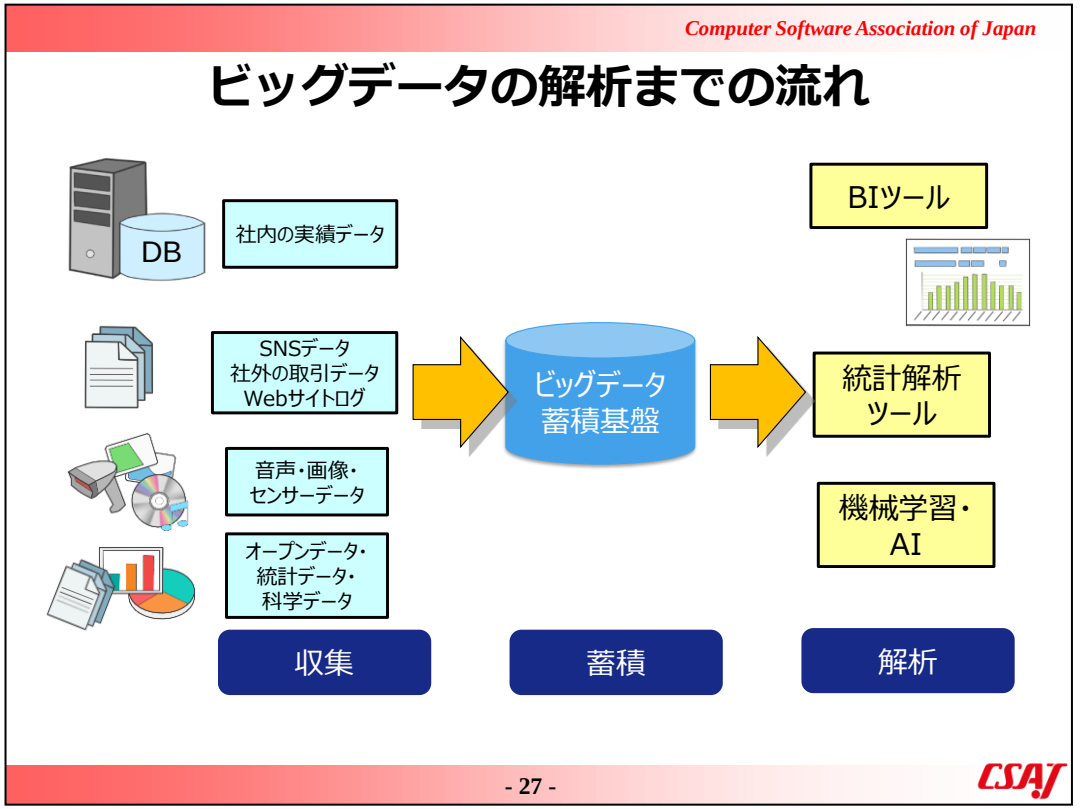
・説明の流れ

IoTサービスをもう少し細かく見てみましょう。

サービスの奥にある蓄積部では、NoSQLや分散処理システムなど、非構造化データを大量に処理できる基盤が利用されていることが分かります。

第２章
ビッグデータを支える技術

データ収集・加工



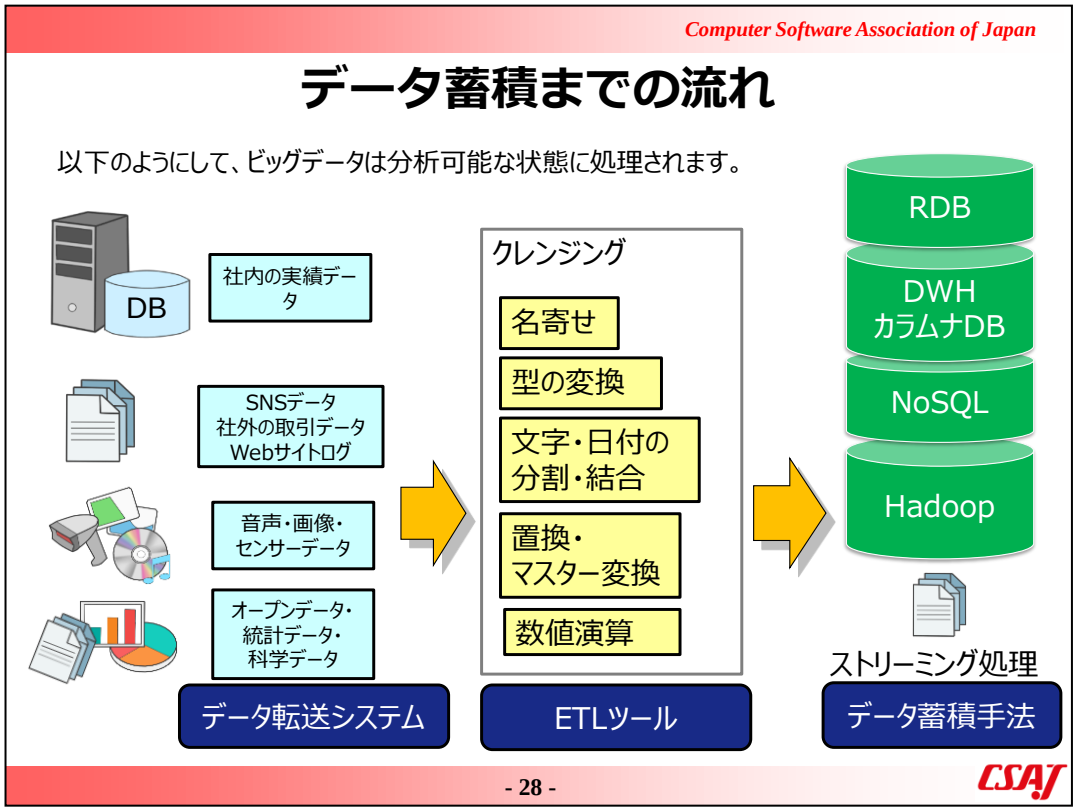
・説明の流れ

ビッグデータの収集から解析までの流れを見てみましょう。

解析部分には、BIツール（ Business Intelligence ） や統計ツール、機械学習、AIなどが利用されています。

・ポイント（絶対に覚えてほしいこと、など）

ビッグデータ利活用には、蓄積だけでなく、前処理とアウトプット部分が必要。



・説明の流れ

データを蓄積する部分までを詳しく見てみましょう。

データを蓄積据える前にデータをIoTサービスまで転送する仕組みが必要になります。

また、データを活用するにあたり、そのままでは使えないデータが混入する場合もあり、これらを正規のデータ、もしくは分析で扱いやすい易い形に加工する処理が必要になります。

この加工処理をクレンジングと呼びます。

データ収集例

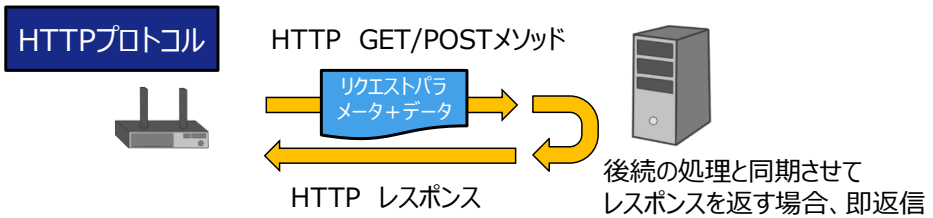
データ収集と通信方法

IoTの受信サーバの通信方法は3種類に分類できます。

・HTTPプロトコル
通常のWebシステムと同様、HTTPプロトコルを利用したWeb APIを利用してデバイスからアクセスを行う。

・WebSocket
音声や動画のリアルタイム通信を行う。

・MQTT
送受信を媒介する第三者の存在により、柔軟な通信を可能にするメッセージ・キュー方式を利用する。



・説明の流れ

データ転送には３つの方法があります。HTTPプロトコルは最も単純ですが、通信は一方的です。

WebSocketは双方向通信が可能ですが、通信を確立するためのアプリが双方に必要になります。

MQTTはメッセージを媒介する、ブローカーが間にあれば、柔軟にデータのやり取りを実現することが可能です。

HTTPプロトコルは通信は、Webシステムと同じHTTPプロトコルを使うため単純ですが、デバイス側からの通信を待つ必要があります。

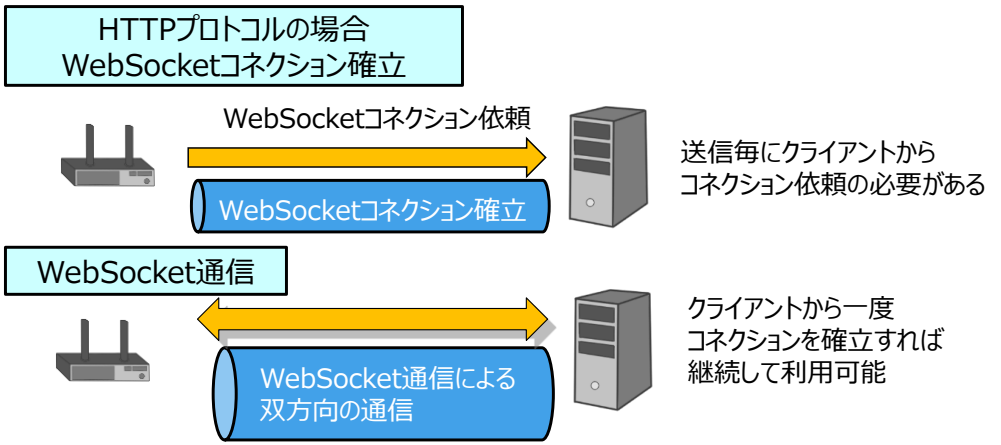
サーバーからの制御も通信があった一回限りになります。

データ収集例

WebSocket

WebブラウザとWebサーバー間でデータを双方向かつ連続的に送受信するための通信プロトコルです。
IoTでWebSocketを用いることで、インターネット上でソケット通信が可能になります。

HTTPプロトコルを用いると送信毎に接続の依頼が必要ですが、WebSocket通信を行うと、接続を継続したままにすることが可能となります。



・説明の流れ

Webソケットの通信イメージを見てみましょう。

・ポイント（絶対に覚えてほしいこと、など）

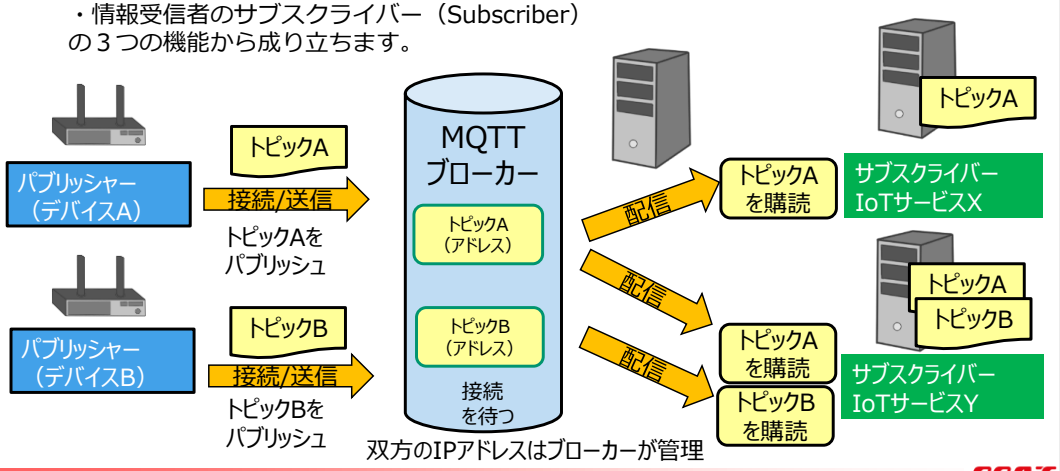
Webソケット通信は双方向の通信に強いため、画像などの大きなデータ通信にも向いている。

データ収集例

MQTT（MQ Telemetry Transport）

IBM社により提唱され、現在はオープンソースとなっているプロトコルです。

・仲介役のブローカー（Broker）
・情報発信者のパブリッシャー（Publisher）
・情報受信者のサブスクライバー（Subscriber）
の３つの機能から成り立ちます。



・説明の流れ

MQTTはIoTに特化したメッセージ・キューシステムです。

メッセージブローカーが、センサーデバイスと、IoTサーバーの間に立ち、データの配信を行っています。

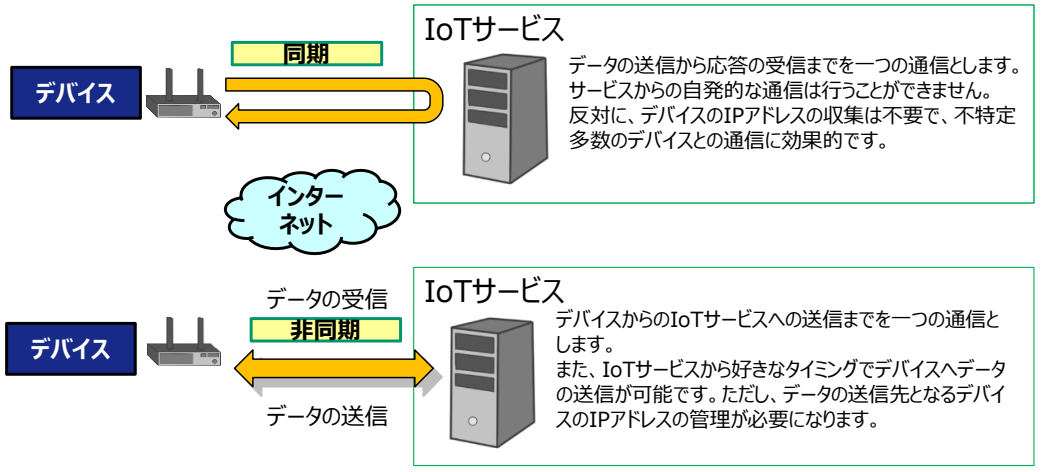
・ポイント（絶対に覚えてほしいこと、など）

MQTTは非同期通信が可能。

障害があった際のメッセージの送り直しなど、多彩な機能が用意されている製品もある。

同期通信と非同期通信

デバイスとの通信には同期通信と非同期通信があります。



・説明の流れ

デバイスとの通信には同期通信と非同期通信があります。

同期通信の場合、デバイスからの通信に対してレスポンスを送るため、デバイスのIPアドレスを知る必要がありません。

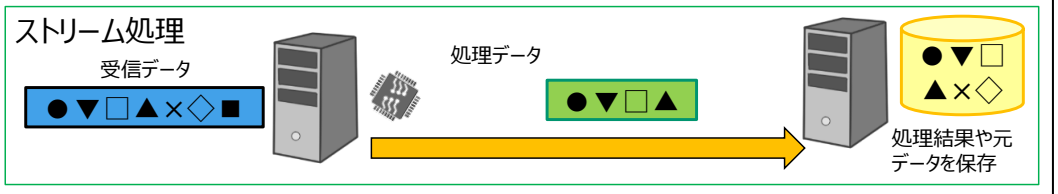
非同期通信の場合は、サーバーから好きなタイミングでデバイス制御を行えますが、デバイスのIPアドレスを管理する必要があります。

ビッグデータの処理と保存

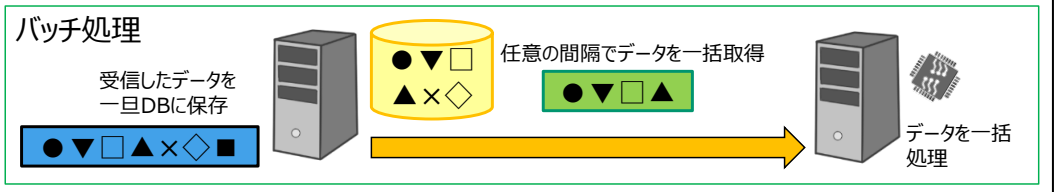
ビッグデータの処理

受信したデータは、データベースや分散ファイルシステムなどに保存されます。
また、受信したデータからデバイス制御の判断を行います。

処理と保存の方法には、ストリーム処理とバッチ処理の2種類があります。



データ内容を常に判断し、即座にデバイスを制御したい場合に利用



記録とデバイス制御にタイムラグがあっても問題ない場合に利用

・説明の流れ

IoTやビッグデータ処理では、データを一括で処理するバッチ処理と流れてくるデータを蓄えず、リアルタイムで処理していくストリーミング処理に分かれます。

・ポイント（絶対に覚えてほしいこと、など）

即時応答が必要かどうか、要件によっていずれかを選択する必要がある。

例えばセンサーデータを監視し続け、故障、危険などにより異常値を検出した場合は、アラートを出したり、関連機器に指令を出して調節などをする必要がある場合は、ストリーム処理が適しています。

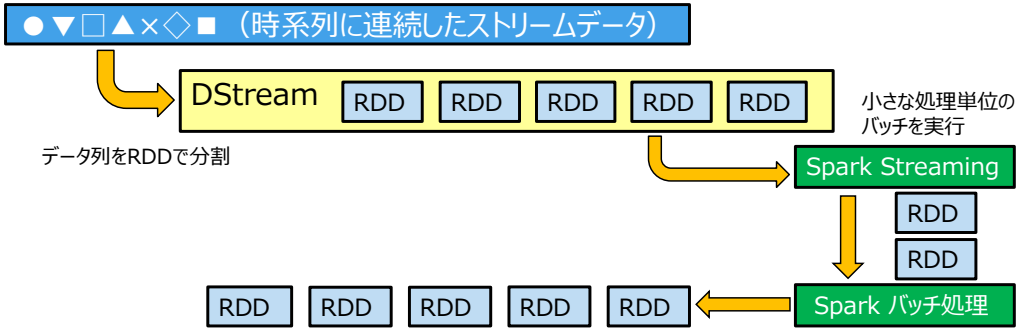
ストリーム処理

ストリーム処理

ストリーム処理はデータを保存せずに、処理サーバーに到着したデータを逐次処理する方法です。与えられたデータにリアルタイムで反応することができます。

Spark Streaming

Spark Streamingは、ストリーム処理を行うためのSparkのライブラリです。
時系列的に連続したデータ列をRDDで分割し、分割されたRDDに対して小さな処理単位のバッチを実行します。



・説明の流れ

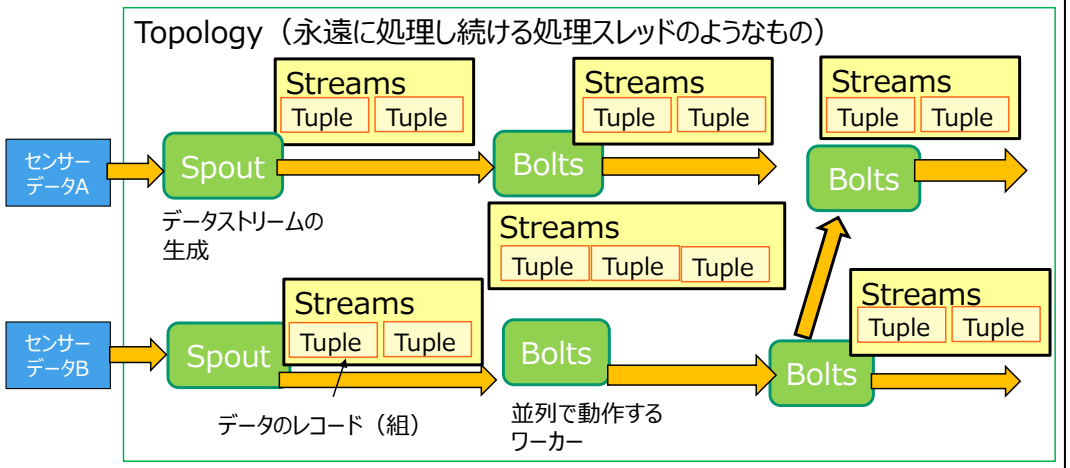
ストリーミング処理の技術にはどのようなものがあるでしょうか。

Spark　StreamingはRDDという小さな処理単位に分けることで常に処理を流し続けることが可能です。

ストリーム処理

Apache Storm

Apache Stormとは、ストリーム処理のためのフレームワークです。



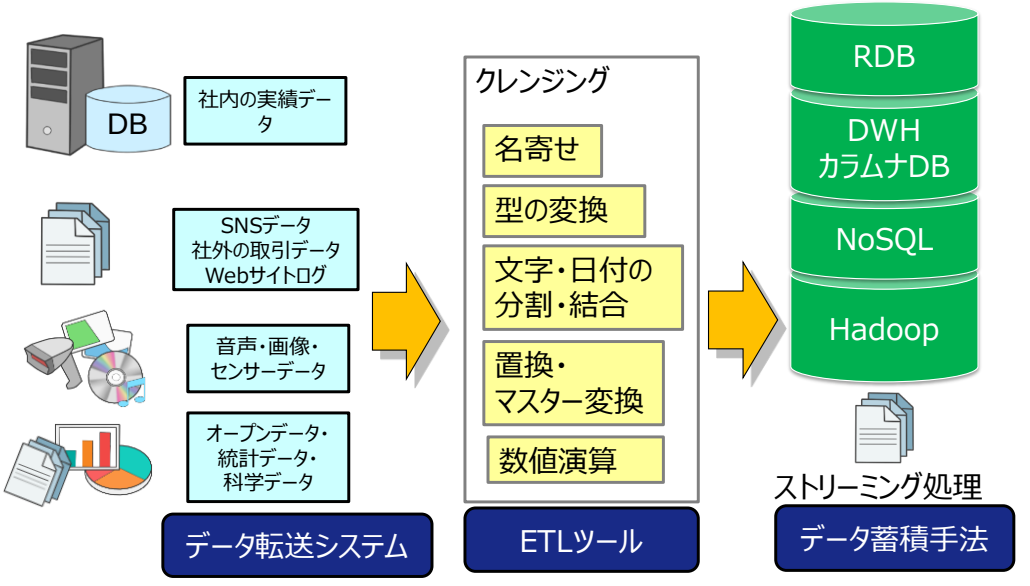
・説明の流れ

ストリーミング処理で有名なフレームワークとしてApache Stormがあります。

Spoutから入ってきたデータをタプルというデータ単位に分けBoltsというワーカーで順次処理をしていきます。

データ蓄積までの流れ

ビッグデータを分析可能な状態に処理する流れは以下のようになっています。



・説明の流れ

次にデータ加工、クレンジングの部分を見ていきましょう。

このページは「データ蓄積までの流れ」の振り返りです。

データ管理：クレンジング

クレンジング

データの整理や加工を行うことで、効率的に分析できるようになります。
これをクレンジングもしくはデータ加工と呼びます。
専用のツールやR、Python、GOなどのスクリプト言語で実施できます。

クレンジング対象	具体例	対処例
型の統一、日付、数値など	数値演算を行いたいデータに文字が入る場合など	基準を用意し、基準に合わないデータ修正
書式の不適合・表記ゆれ	住所、会社名、電話番号、郵便番号などの表記の不統一、通貨や数値の単位、文字コードなど	表記方法や単位の基準を定義し、基準に従うようにデータを修正する
異常データ	ある条件下でのセンサーの誤動作、データの無記入による空データ、データ入力ミスによる意味のないデータ	回帰やクラスタリングにおいて、結果の精度を下げる原因に異常値を検知し、事前に外す
個人情報およびプライバシー情報の保護	氏名、住所、メールアドレスなど個人を直接特定できる情報	個人を直接特定できる情報を削除または匿名化処理を施す

・説明の流れ

データクレンジングでは、日付の形式の統一、データの分割、結合、マスターデータに紐づけなど、要件に応じたあらゆる加工処理を行います。

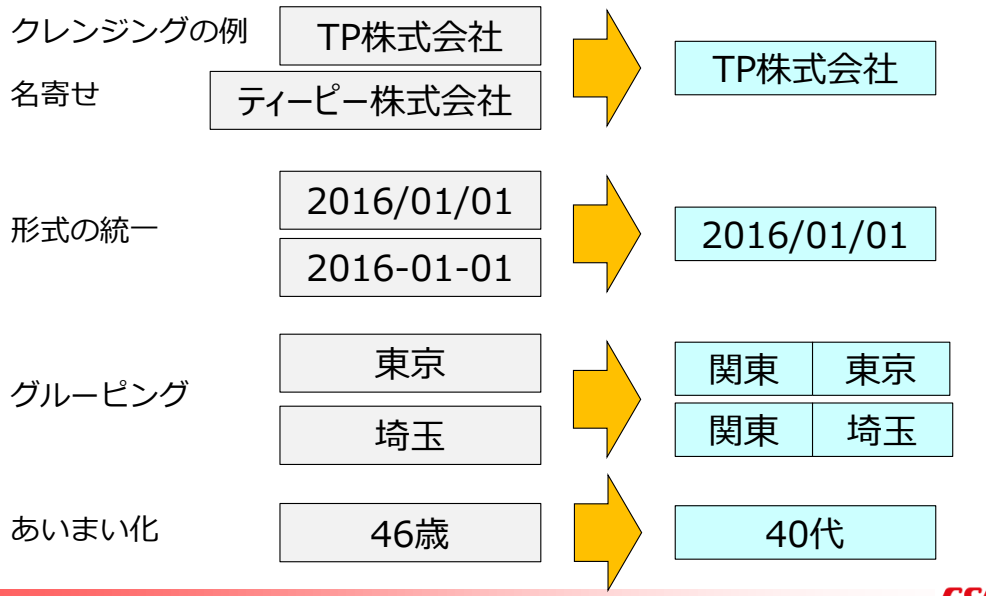
後の工程に回すと処理が複雑になったり、分析の処理速度が低下することが予想される場合は、極力前処理でデータをきれいにする（クレンジングする）事が重要となります。

・ポイント（絶対に覚えてほしいこと、など）

データの欠損、異常値などは分析結果に影響することがあるため、入り口で除去できるのであれば、除去した方が良い場合がある。

ただし、分析の目的が異常値検出などである場合は、除去してはいけない。

データ管理：クレンジング具体例



・説明の流れ

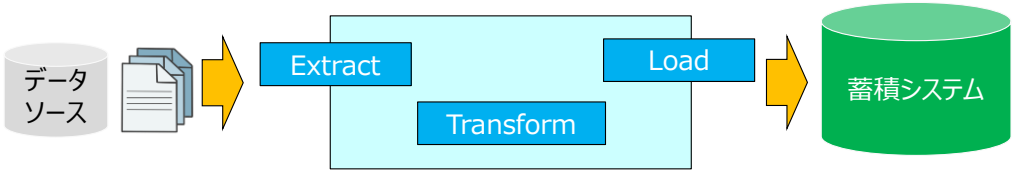
具体的なクレンジングの例を見てみましょう。

名寄せや形式変更、グルーピングといった従来から使われている変換の他、ビッグデータでは匿名化処理もクレンジングでの重要な加工処理となります。

データ管理：ETLとELT

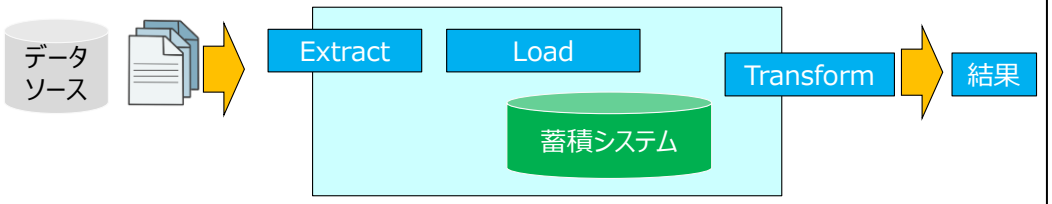
ETL（Extract Transform Load）

収集したデータを抽出(Extract)した後、利用しやすいように加工(Transform)し、DWHなどの蓄積先に書き込む(Load)一連の処理を表します。
大量データをバッチ処理する商用のETLツールなどが存在します。



ELT（Extract Load Transform ）

収集したデータを抽出(Extract)し、DWHなどの蓄積先に書き込み(Load)をした後、利用する時に初めて加工(Transform)を行う方式です。



・説明の流れ

データ加工は、処理の流れからETL（Extract　Transform Load）と呼ばれます。

データを抽出し、加工、DBなどにロードという流れです。

一方でビッグデータの場合は、収集時点では何に使うか決めず、蓄積してから利用する内容を決めることも多いのが現状です。

その場合は、Extractしてから蓄積環境にLoadし、使う際にTransformするという順番になります。

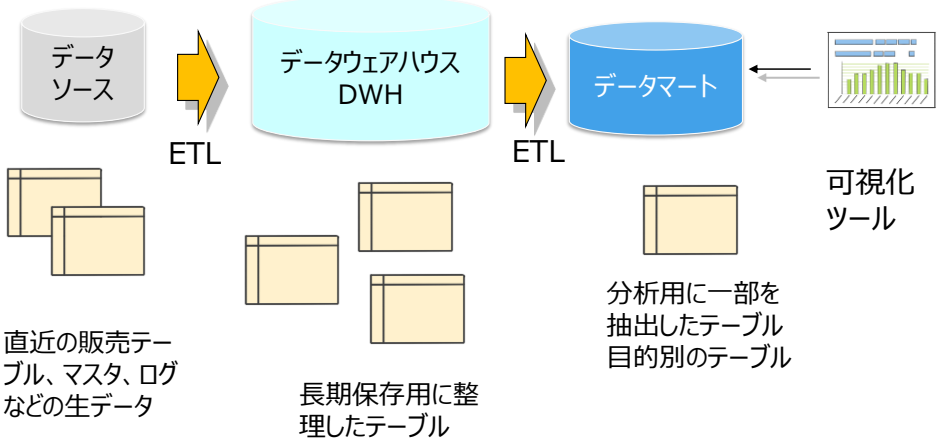
この処理方式をELTと呼びます。

・ポイント（絶対に覚えてほしいこと、など）

ETLとELTの違いを把握しましょう。場合によっては、ETLも使い、ELT処理もするケースもある。

変換するツールはどちらも同じツールになる場合も異なる場合もある。そのため、要件に応じた選定をする必要がある。

データ管理：従来型データ蓄積



・説明の流れ

ビッグデータ以前に一般的に利用されているデータ蓄積方式を見てみましょう。

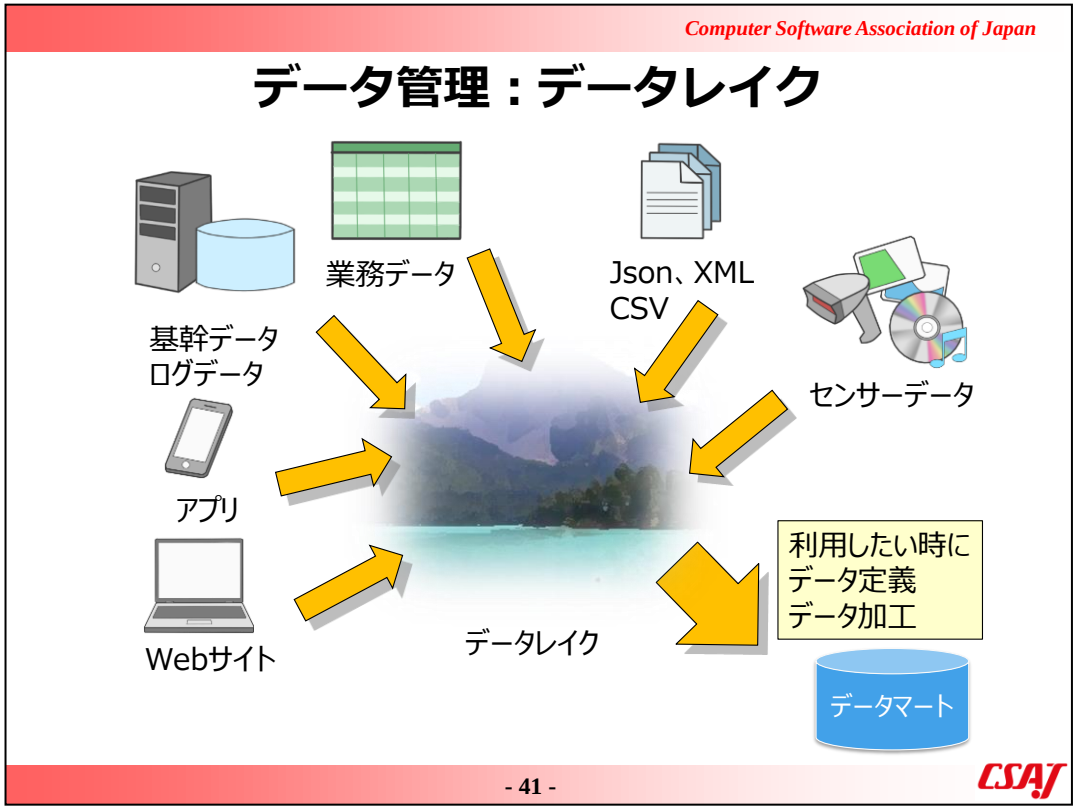
データウェアハウスは、データを長期に渡って蓄積するデータの倉庫として活用されてきました。

構造化DBであるカラムナデータベースや、RDBなどで運用されます。BIツールからのアドホック（随時）集計要求に応えるため、目的別に抽出したデータマートという結合済みテーブルを小さい規模のDBに入れて運用する形式も良く採用されます。

一方で、データ発生源となる基幹データベースは、日中業務で利用されているため、負荷をかけないようDWH、データマート、BIからは直接アクセスさせない運用が一般的です。夜間、DWHにデータを移行します。

・ポイント（絶対に覚えてほしいこと、など）

基幹DBと、DWH、データマートの違いを理解する。



・説明の流れ

DWHが構造化データを扱い、最初から構造を決めないと構築すらできないのに対し、ビッグデータは既にある情報を利用したり、そもそも構造化できないデータの場合が一般的です。

そのため、とりあえずデータを蓄積しておいて、利用する段になったら初めて構造を定義しようという発想が生まれました。

この、データをとりあえず貯めこんで後で利用する形態を、データレイク、データの湖と呼びます。

・ポイント（絶対に覚えてほしいこと、など）

データレイクでは、データ構造の定義をいつ行うかを押さえておく。

スループットとレイテンシ

ビッグデータ処理性能を測る指標

・スループット
一定時間に処理できるデータの総量。
WHやデータレイクなど、データの量が多い処理を行う時に重視。

・レイテンシ
データ処理が終わるまでの待機時間。
アドホック集計、データマート、BIツールなどで重視。

スループットとレイテンシは両立しないことが多く、複数のシステムで役割を分担することがあります。

・説明の流れ

ビッグデータシステムの性能を図る指標は何でしょうか。

それは、処理の早さか、単位時間に処理するデータ量かということです。それらを前者はレイテンシ、後者はスループットと呼びます。

一見似たような指標のようですが、内容が異なりますのでご注意ください。

・ポイント（絶対に覚えてほしいこと、など）

レイテンシはアドホック集計能力、スループットは夜間のバッチ処理能力に関わる。

従来型のデータベース：行指向DB

行指向データベース

テーブルの行を一つのデータとしてディスクに保存する方法です。

追記は末尾に加えるだけなので容易です。
トランザクションが多く発生する業務アプリなどで利用されます。

ディスク格納イメージ

2017-01-01	商品A	5000	10
2017-01-02	商品B	3750	1
2017-01-03	商品C	2160	5

高速化のためにIndexを利用することがありますが、分析対象となる列がわからないため、ディスクI/Oの軽減にならない場合が多いです。

・説明の流れ

RDB、リレーショナルデータベースは、ビッグデータ集計に向くのでしょうか。データ保存の仕組みから紐解いてみましょう。

行指向データベースでは、データの格納を行単位で行うため、列ごとの集計では、必要のない項目（カラム）も読み込んでしまい、高速に集計できない場合が多いです。

行指向データベースは追記や更新（内部的には追記）は高速なため、エントリー業務と参照業務で主に利用されます。

・ポイント（絶対に覚えてほしいこと、など）

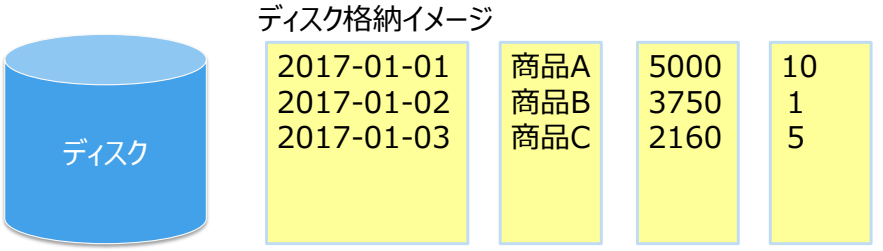
RDBに代表される行カラムDBは、入力業務には強いが、集計業務には向かない。

従来型のデータベース：列指向DB

列指向データベース

テーブルのデータを列単位にまとめて保存する方法です。

集計に必要な項目だけを読み込むことができ、ディスクI/Oの軽減ができます。



また、カラム単位での重複は集約が容易なため、圧縮効率が高いです。

・説明の流れ

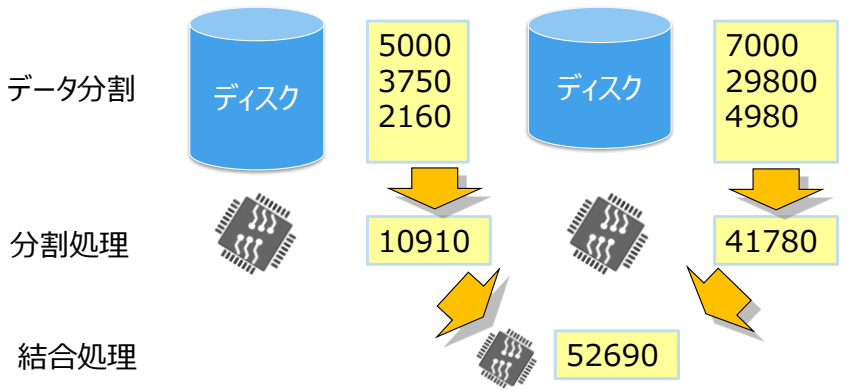
列指向データベースは列単位でデータを保存するため、集計は高速になります。

しかしながら、1台での運用を想定する場合が多く、ビッグデータの規模での運用は難しいことと、仕組み上構造化データしか扱えないため、ビッグデータの分野での活用は進んでいません。

従来型のデータベース：MPP

MPPデータベース（Massively Parallel Processing ）

一つのクエリを多数の小さなタスクに分解し、多くのCPUコアやコンピュータを並列的に稼働することで結果を得る手法です。



・説明の流れ

列指向DBや行指向DBを複数台用意し大量のCPUで並列処理する、MPPデータベースが現れ、より大規模のデータを扱えるようになりました。

しかしながら扱うデータは構造化データであり、ソフトも高額であるため、Hadoopのような安価なサーバーを大量に配置することはできません。

そのため、データレイクのようなとりあえず貯めるという発想での運用は難しいと言えます。

第３章

ビッグデータコア技術

NoSQL

ビッグデータとその周りの技術の関係について学ぶ。

周辺技術ごとにビッグデータとどのように関連しているのか理解する。

NoSQLとは

NoSQLとは

・Not only SQL = RDBMS以外のデータベースの総称
・SQLを否定するものではない。
・1998年にCarlo Strozzi氏が初めて名称を使用した。(NoREL)
・非構造化データの蓄積や管理に利用される。

BASE

・Basically Available Soft-state Eventual consistency
・処理速度を優先するNoSQLで採用されているトランザクションの考え方。
…「稼動が第一で、厳密な整合性は過程ではなく結果でのみ重要視する」

・説明の流れ

NoSQLはビッグデータシステムの話題でよく耳にする言葉です。

NoSQLのNoとは何を指すのでしょうか。

・ポイント（絶対に覚えてほしいこと、など）

NoSQLはSQLを否定する言葉ではない。

むしろNoSQLでもSQLを扱える製品もある。

ビッグデータで扱うデータの種類

ビッグデータの種類	データの例
構造化データ	データベースに格納されたデータ など (顧客テーブルデータ、 受注テーブルデータ など)
準構造化データ	ログデータ、センサーデータ、SNSに書き込まれたデータ など
非構造化データ	文書、音声、動画、画像 など

・説明の流れ
ここでビッグデータの種類をおさらいしましょう。

NoSQLのメリット、デメリット

NoSQLのメリット

・前もったデータの構造の定義が不要で、柔軟な変更が容易である
・特定の形式に固執しないため、複数のサーバにデータの分散ができる
・データ構造が単純で、更新や検索処理の速度が速い
・基本的にPutとGetのみを行えばよく、JOINがないため操作が明快

NoSQLのデメリット

・RDBMSと比較すると、データ間の整合性を維持する機能が弱い

・説明の流れ
NoSQLのメリットとデメリットを整理しましょう。
メリットは、データ構造の変更のしやすさなどです。
デメリットはDB間の整合性の維持の難しさです。

NoSQLのメリット、デメリット

NoSQLとRDBMSとの比較

	RDBMS	NoSQL
保存に適するデータ	構造化データ	非構造化データ
スキーマ定義	事前定義が必要 固定的で変更しにくい	事前定義不要 データ構造を変更しやすい
データの整合性	同時実行制御がある データ整合性を重視	比較的緩い 大容量データの高速処理を優先、結果整合重視
データ操作命令	SQL言語 (SELECT、INSERT、UPDATE、DELETE)	Put(書き込み) Get(読み出し)
拡張方式	スケールアップ	スケールアウト

・説明の流れ

NoSQLとRDBMSを比較した資料です。構造定義とデータの整合性の扱いで大きく特徴が異なります。

NoSQLのメリット、デメリット

データベースの拡張方式の比較

RDBMS ：スケールアップを行う
NoSQL ：スケールアウトを行う

スケールアップ

1台のサーバの処理性能やストレージを拡張します。

スケールアウト

サーバを増設し処理性能を向上させます。

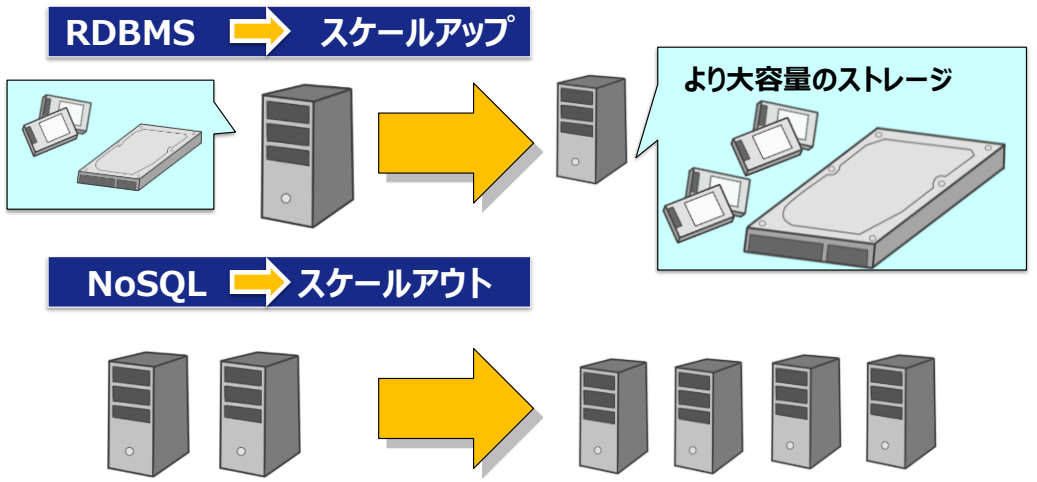
・説明の流れ

データベースの拡張方式の違いをみてみましょう。

スケールアップは、1台のサーバー処理性能のアップで処理能力を向上させるのに対し、スケールアウトでは、サーバー台数を増やすことにより処理性能を向上させます。

NoSQLのメリット、デメリット

データベースの拡張方式の比較



・説明の流れ

データベースの拡張方式の違いをみてみましょう。

スケールアップは、1台のサーバー処理性能のアップで処理能力を向上させるのに対し、スケールアウトでは、サーバー台数を増やすことにより処理性能を向上させます。

NoSQLのメリット、デメリット

スケールアップ

ハードウェアの性能(ディスク、CPU、メモリ)の増強により、特定の1台のサーバの性能を向上させます。

RDBは以下の特徴を持つため、スケールアップが効果的です。

・複数のサーバで運用すると、データ分割や配置作業が必要となる

・サーバ間の大規模トランザクションによりパフォーマンスが低下する

スケールアウト

サーバの台数を増やし、全体での処理能力を向上させます。

高性能がサーバに求められないため、低コストで処理性能を上げることが可能です。

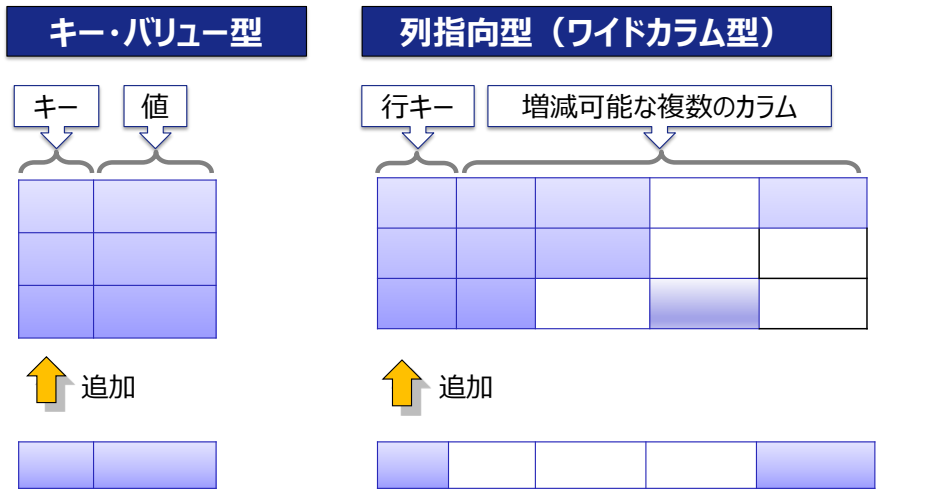
また、1台サーバの停止による影響は少なく、稼働率は向上します。

NoSQLは、データベースで扱うデータが非構造化データなため、複数のサーバへの分割・複製が容易、つまりスケールアウトの手法に向いています。

・説明の流れ

データベースの拡張方式の違いを詳しくみてみましょう。

NoSQLの代表的な種類



列は後から追加も可能。列がない行データも許容。

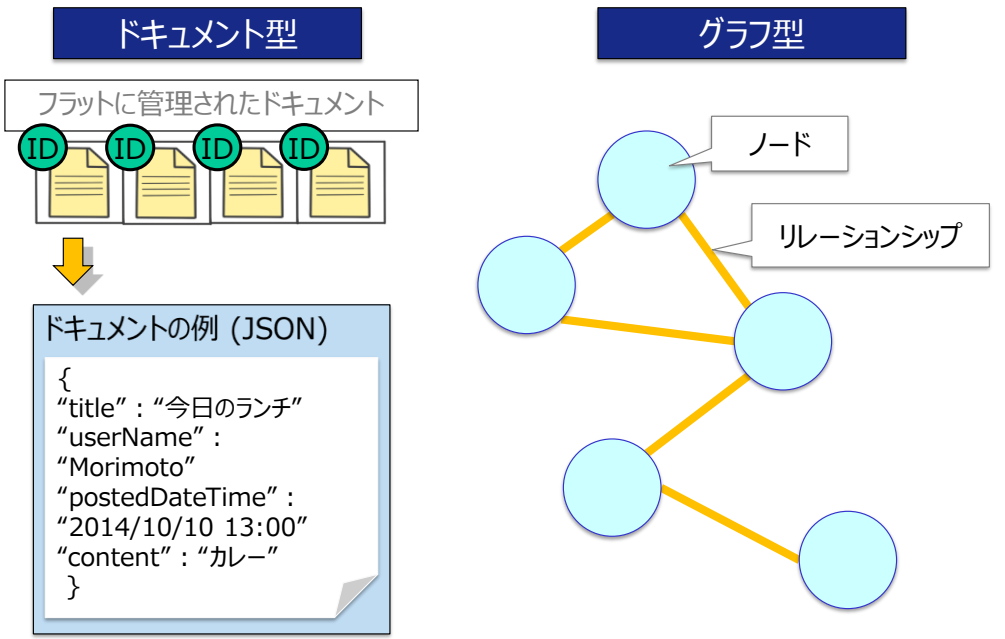
・説明の流れ

NoSQLには様々な種類があり、データの格納方式などで、大きく分けて4種類のNoSQlデータベースに分類されます。

・ポイント（絶対に覚えてほしいこと、など）

4つの種類のNoSQLを押さえる。

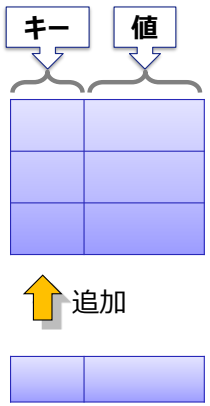
NoSQLの代表的な種類



・説明の流れ
4種類のNoSQLが存在します。

キー・バリュー型の特徴

キー・バリュー型



キー・バリュー型

キーとバリュー(値)のセットでデータを整列します。
キーは、値に対する識別子となるデータです。
値はバイナリデータであれば、BLOB型の画像や音声も格納できます。

キーとキーに紐付けられたデータという単純な構造であるため、スキーマ定義が不要で、検索が高速化でき、またサーバの追加やデータの分割・配置作業が容易となります。つまり、簡単にスケールアウト環境が構築できます。

Dynamo、Voldemort、Riak、Hibari、Redis、Scalaris、TokyoCabinet/Tylantなどで採用されています。

・説明の流れ

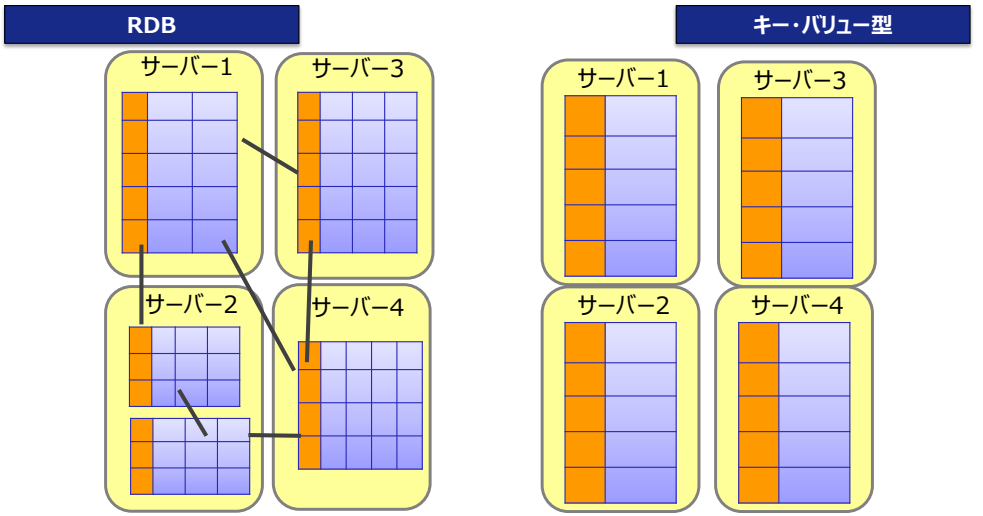
キー・バリュー型のNoSQLDBの特徴を押さえましょう。

画像や音声も格納できること、スキーマ定義が不要なこと、スケールアウトが容易なことが特徴として挙げられます。

キー・バリュー型の特徴

RDBとキーバリュー型NoSQLのスケールアウト比較

キーバリュー型では、整合性を確認する必要性がないので、キーの管理さえしていればサーバを増やすだけで問題はありません。



・説明の流れ

キーバリュー型NoSQLはキーと値という単純構造のため、ただサーバーを増やして続きを格納していきます。

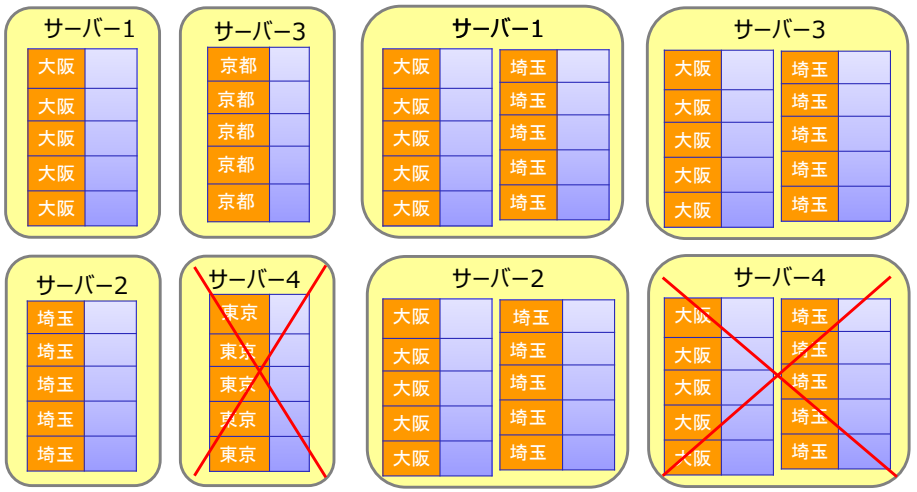
一方でRDBなどの場合は、マスターの配置も重要なため、あらかじめ構成を決めておく必要があります。

Computer Software Association of Japan

キー・バリュー型の特徴

シャーディング / Sharding

あらかじめサーバ毎に保存するデータのキーの範囲を設定しておくことで、検索や管理を容易にする機能です。



CSAJ

・説明の流れ

キーバリュー型では、あらかじめどのサーバーに特定の範囲のデータを入れておくかを決める機能があります。

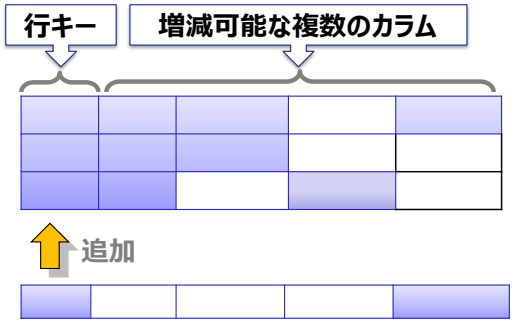
これをシャーディングといい、検索をする際には特定のサーバーを探索すればよい形になるため、処理時間を短縮することができます。

ただしサーバーダウンをした場合、ダウンしたサーバーに格納されているデータは参照できなくなるため、サーバー相互に冗長化してデータを持たせて可用性を担保する方式がとられます。

列指向型の特徴

キー・バリュー型を機能強化したもので、各行のキーが複数の列(カラム)の値を持つことが可能です。

列単位でデータの保存や読み込み処理を行うことができ、列の集計処理などを高速に実行することができます。



一部の列のデータが欠けている行があっても問題ありません。また、行や列の数は自由に好きなだけ拡張可能です。

Bigtable、Cassandra、Hbase、Hypertableが採用

※カラムナデータベースとは別物です。

・説明の流れ

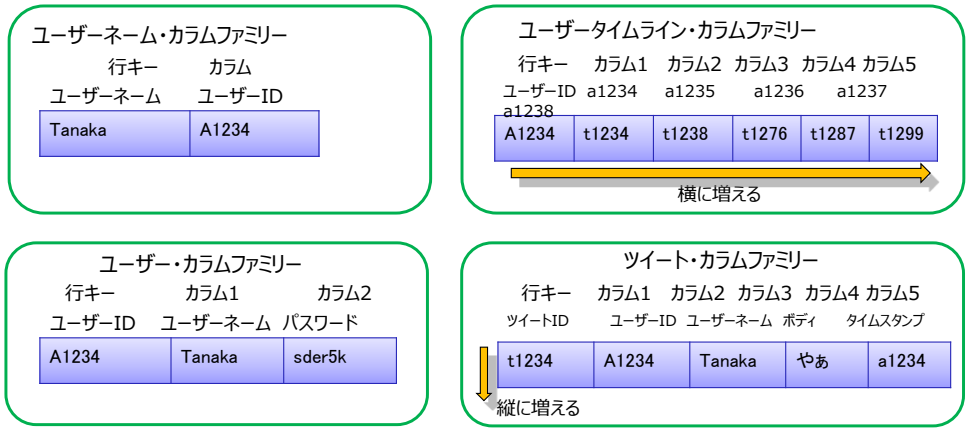
列指向型NoSQLの特徴を見ましょう。列指向型では、キーと複数のカラムを持つことができます。

複数の列項目は、項目数が一致している必要がなく、後から列の削除や追加を自由に行う事が可能です

列指向型の特徴

Twitterの例

行キー：ユーザーネーム
カラムの名前：ユーザーID
カラムファミリー = カラムをまとめる入れ物



・説明の流れ

Twitterの例でどのように柔軟に列指向型NoSQLが働くかを見てみましょう。

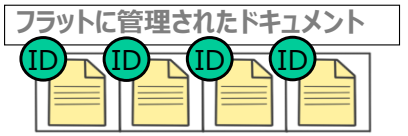
列指向型では用途に応じてカラムファミリーと呼ばれる入れものを定義できます。

ツイート・カラムファミリーでは、行を増やすイメージでツイート情報を管理します。

一方でタイムラインなどは、一人のユーザーのタイムラインで話された内容は横にカラムを増やして対応することができます。

さらに、列指向型は横にも縦にも情報を増やすことが可能です。

ドキュメント型の特徴



ドキュメントの例（JSON）

```
{
“title” : “今日のランチ”
“userName” : “Morimoto”
“postedDateTime” :
“2014/10/10 13:00”
“content” : “カレー”
“count” :2
 }
```

ドキュメント指向型

JSONやXMLなどのデータ記述書式のドキュメントを格納します。

階層構造は持たないフラットなドキュメントの形式でデータを管理します。各ドキュメントには管理のためのユニークIDが割り振られます。

スキーマレスです。(データ設計が不要です。)

ドキュメントの内部構造はDB、アプリケーションからわかり、任意の内容でクエリを作成できます。

MongoDB、CouchDBで採用されています。

・説明の流れ

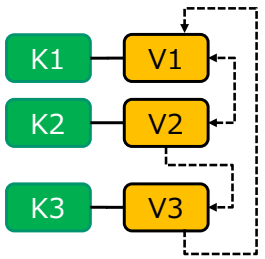
ドキュメント型の特徴は、JSONや、XMLといった準構造化データをドキュメントとしてフラットに格納することができます。

ドキュメント内の特定の情報も検索することが可能です。

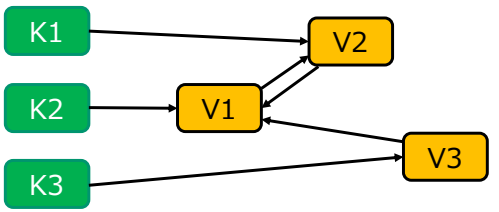
グラフ型の特徴

グラフ型

- キーバリュー型格納の例



グラフ型格納の例



データ同士の入り組んだ関係を独立して表現します。
グラフ型は以下の要素により構成されます。

ノード (node) … 他のノードと関係をもつ要素を表す。
リレーションシップ（RelationShip） … ノード間における関係の有無と方向を矢印で示す。
プロパティ（Property） … ノードとリレーションシップの具体的な属性を示す。

・説明の流れ

　グラフ型はデータ間の複雑な関係を表現しながらデータを格納することができます。

代表的なNoSQL製品

代表的なNoSQL製品

NoSQL製品はさまざまなものが多数存在します。

NoSQL分類	製品	ベンダー
キー・バリュー型	Amazon DynamoDB	Amazon
	Hibari	クラウディアン
	Riak	Basho Technologies
	memchached	(オープンソース)
列指向型	HBase	(オープンソース)
	Apache Cassandra	(オープンソース)
	Bigtable	Google
ドキュメント型	Apache CouchDB	(オープンソース)
	MongoDB	(オープンソース)
グラフ型	InfiniteGraph	Objectivity, Inc.
	Neo4j	オープンソース　/ 商用ライセンス(Neo Technology)

・説明の流れ

　それぞれの型で代表的なNoSQL製品にはどのようなものがあるか見てみましょう。

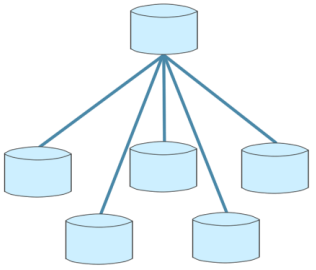
・ポイント（絶対に覚えてほしいこと、など）

　それぞれの製品に特長があるため、試験的に利用しながら用途に応じて最適なものを選ぶ必要があります。

NoSQLの基本的概念と技術

NoSQLのデータベースアーキテクチャ
サーバの配置方法には、マスタ型やP2P型の2つがあります。

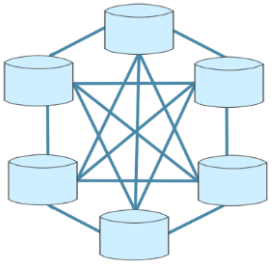
マスタ型



マスターノードはノード管理を役割を持ちます。
単一障害点(SPOF)が存在します。
※マスターノードがSPOFです

例 BigTable、CouchDB、
Hbase、Hibari、MongoDB

ピア・ツゥ・ピア(P2P)型



単一障害点はありません

例 Cassandra、Dynamo
Riak、Voldemart

・説明の流れ

NoSQLのサーバー配置には2通りの種類があります。マスタ型とP2P型です。

マスター型は、処理サーバーを管理するマスターノードが存在します。

マスターノードはダウンした場合に単一障害点になってしまい、システム全体が利用不可能になってしまいます。

一方P2P型は障害が発生しても、サーバーが自動的に連携しながら応答する形をとるため、単一障害点は存在しません。

NoSQL時代の必要要件

必要要件

NoSQLに求められることは様々であり、全てをみたすデータベースは存在しません。

- 膨大なデータ　：複数のサーバを必要とするデータ量を扱う
- 分散配置　：データを複数のサーバに分けて保存する
- コスト　：性能は求めず、安価なハードウェアを多数用意する
- 信頼性　：データの欠損や漏洩を防ぐ
- 可用性　：稼働率を高く維持する
- 耐障害性　：障害による影響を最小限にとどめる
- 性能　：処理や応答をできる限り早く行う

CSAJ

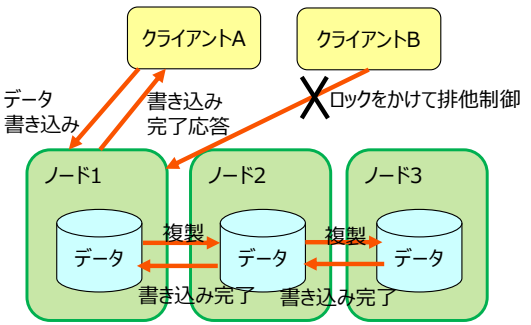
・説明の流れ
NoSQLで要求される必要要件を押さえましょう。
これらすべてを満たすデータベースは存在しません。
必ず何かをトレードオフする必要が発生します。

NoSQLの基本的概念と技術（整合性）

データの整合性（Consistency）とは
　作業の実行後に複製(レプリケーション)されたデータ間に矛盾がないことを示します。
整合性には、「強い整合性」と「緩い整合性」があります。

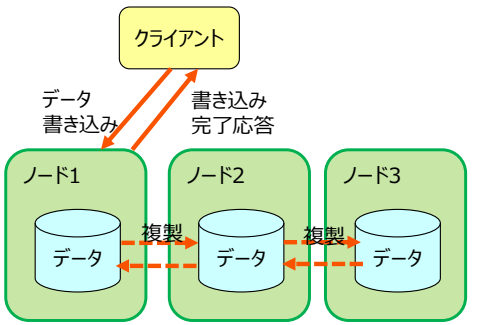
強い整合性

必ず最新の更新データが出力されることを保証します。特に、RDBでは重要視され、保持されます。



緩い整合性

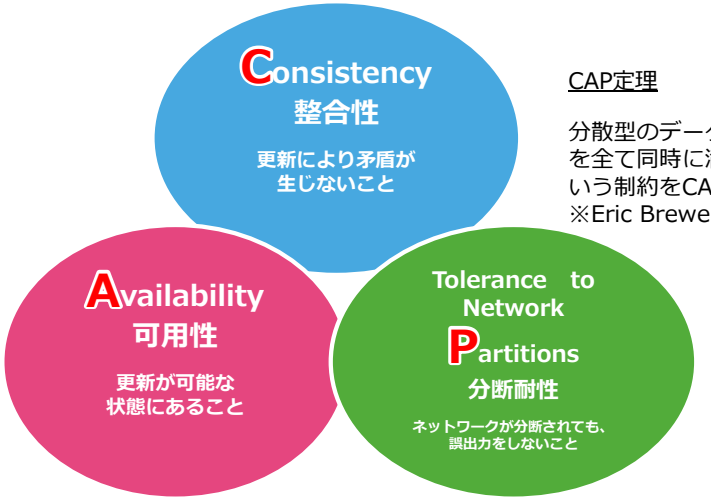
更新から少し時間をおいた後であれば、全てのノードで最新のデータが読み出せればよいとする考え方です。



書き込み完了確認がない状態でクライアントに完了通知を行う

NoSQLの基本的概念と技術（整合性）

データベースの特性でも重要な以下の3つの要件を考えます。



CAP定理

分散型のデータベースでは、3つの要件を全て同時に満たすことはできない、という制約をCAP定理と呼びます。
※Eric Brewer氏により提唱されました。

・説明の流れ

データベース選定の上で、整合性、可用性、分断耐性は重要な選定ポイントです。

しかしながらそのすべてを完全に満たすデータベースは存在せず、3つの中の2つを満たすことが限界です。

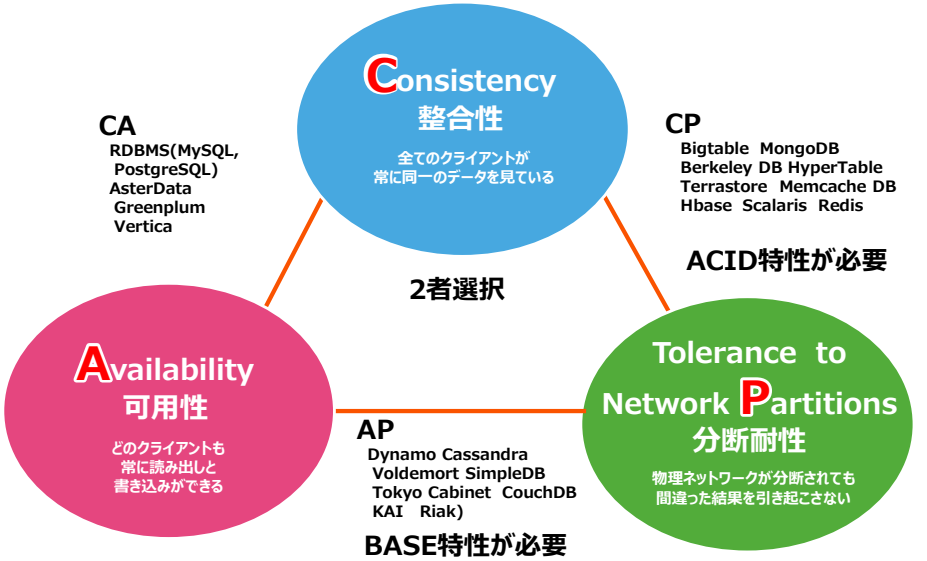
これをそれぞれの頭文字をとって、CAP定理と呼びます。

・ポイント（絶対に覚えてほしいこと、など）

CAP定理の考え方とDB選定

NoSQLの基本的概念と技術（整合性）

NoSQLではCPかAPのいずれかが保証されることが基本です。



・説明の流れ

現存するデータベースで、CAP定理のいずれを重視しているか見てみましょう。

RDBはやはり、CA、整合性と可用性を重視しています。

一方でNoSQLデータベースではCP、APを重視しているケースが多く、瞬間的な整合性を求めずに、とにかく動くことを重視しているケースが多いようです

NoSQLの基本的概念と技術（整合性）

ACIDとBASE

・ACID(Atomicity、Consistentry、Isolation、Durability)

トランザクションは、全ての処理完了または処理以前の状態への復元のいずれかを行う仕組み。

Atomicity 原子性	：手順が全て実行されるか、一つも実行されないか
Consistency 一貫性	：トランザクションの前後で整合性が保たれる
Isolation 独立性	：実行中に他の処理に影響を与えない
Durability 耐久性	：データが正しく記録され、損失の可能性がない

・BASE(Basically Available、Soft-state、Eventual Consistency)

アプリは基本的に常に動作し（Basically Available）、強い整合性は持っていないが（Soft-state）、最終的に整合性をもつ（Eventual Consistency）という特性を備えているべきであると言う考え方。

・説明の流れ

強い整合性か、ネットワーク分断があってもとにかく動くことのどちらを重視するかが大きなポイントとなりますが、それぞれを表す言葉として、ACIDとBASEという言葉があります。

それぞれの中身を見ていきましょう。

・ポイント（絶対に覚えてほしいこと、など）

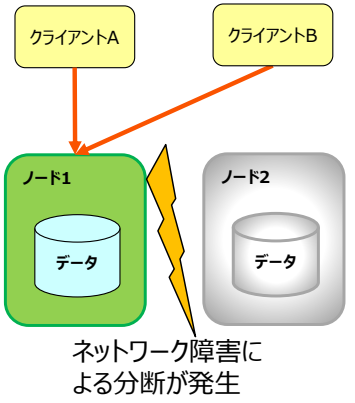
ACIDとBASEの違い、それぞれの特徴

NoSQLの基本的概念と技術（整合性）

ネットワーク障害時のCPとAPの対応

CP（整合性と分断耐性）

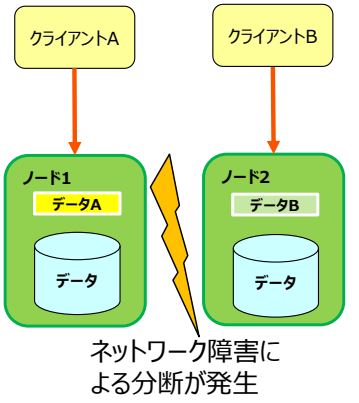
リクエストは片方のグループだけが受け付け、他のグループは自動的に停止します。データを受け付けないグループは更新が起きないため、整合性が乱れる心配はありません。



ネットワーク障害による分断が発生

AP（可用性と分断耐性）

全てのノードで更新を行えるようにするため、リクエストは全てのグループが受け付けます。情報の共有ができないノード間で不整合が発生します。



ネットワーク障害による分断が発生

・説明の流れ

CPとAPでネットワーク分断が起こった際の挙動の違いを見てみましょう。

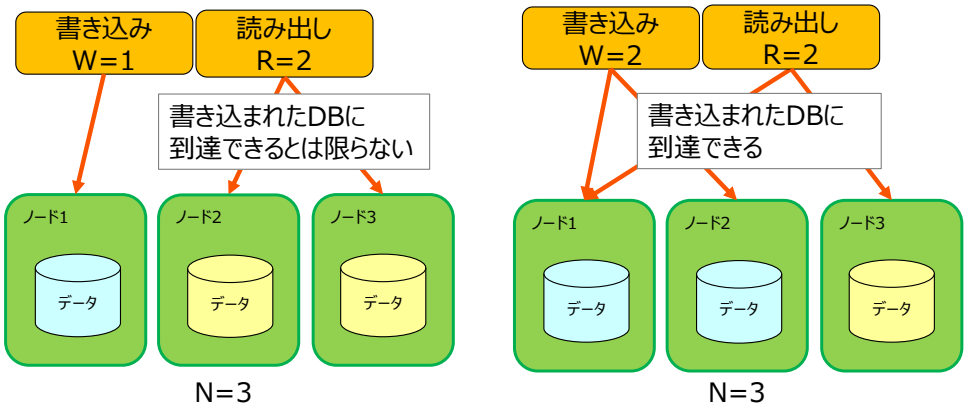
APではネットワーク分断により、ユーザー間で一時的に結果が異なっても結果を返し続ける方法を取ります。

NoSQLの基本的概念と技術（整合性）

整合性と性能のトレードオフ

R(ノードから読み込む数)とW(ノードに書き込む数)の和がN(レプリカの数)より大きければ、整合性が保証することができます。

複数の書き込みと読み出しを行う事は、応答速度とのトレードオフになります。



・説明の流れ

APの場合、整合性をあきらめてしまうしかないかというとそうではない場合もあります。

書き込みと読み出しを行う際に、複数のノードに処理を行う事で、正しい結果に到達できる仕組み（Quorum）を取ることができます。

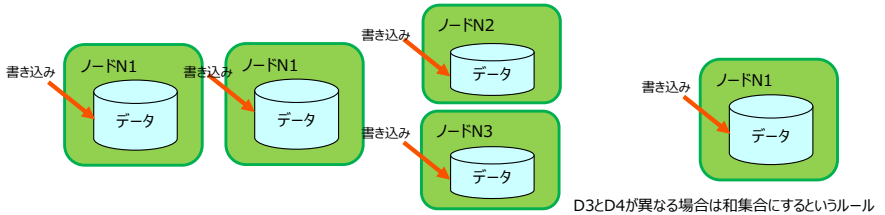
ただし、この場合、書き込み、読み出しを複数行う必要があるため、応答性能は低下します。（トレードオフになります）

NoSQLの基本的概念と技術（整合性）

データのバージョン管理

・タイムスタンプ(TimeStamp)
データの更新時間を記録する。
新しいタイムスタンプを持つものを重要視することでバージョンを管理する。

・ベクタークロック(Vector Cloks)
複数のバージョン間の仲裁をする仕組み。
変更が行われたノードと、そのノードで変更が行われた回数を記録する。



D3とD4が異なる場合は和集合にするというルール

ベクタークロック

D1
([N1 , 1])

D2
([N1 , 2])

D3
([N1 , 2] ,[N2,1])

D4
([N1 , 2],[N3,1])

ルールに基づいてN1に記録

D5
([N1 , 3],[N2,1],
[N3,1])

・説明の流れ

複数ノードで書き込まれたデータが最新化どうかを調べる単純な方法としては２つあり、更新時の更新時間（タイムスタンプ）を見る方法と複数バージョンの記録をとり、仲裁するベクターロックとい仕組みも用意されている方法があります。

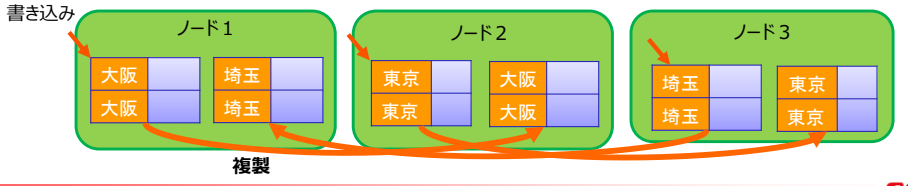
NoSQLの基本的概念と技術（データ分割）

整合性を後から修復する仕組み

・リードリペア（Read Repair）
全てのノードで複製データの読み出し要求を行い、データに矛盾がないかを確認する。発見された矛盾はバックグラウンド処理で修復する。

・ヒンテッド・ハンドオフ（Hinted Handoff）
故障が発生した場合、他の稼働中のノードに、故障の復旧後の更新手続きを指示する仕組み。

・マークル・ツリー（Markle Tree）
Markle Treeという樹形図構造のアルゴリズムによりデータの整合性を確認し、整合性に問題がある場合は手動での修復を行うもの。



・説明の流れ

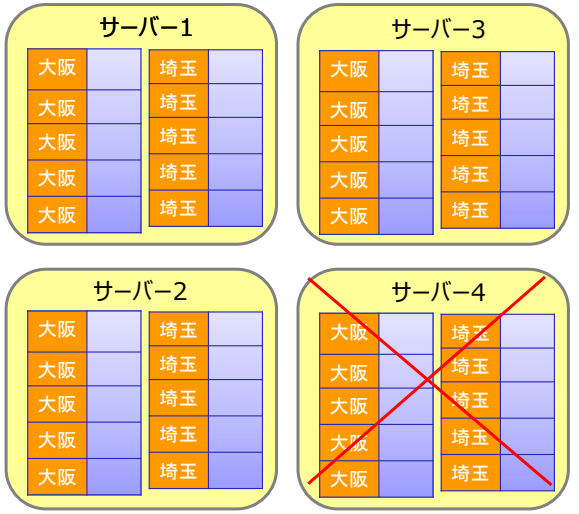
データ分割におけるNoSQLの機能を見ていきましょう。

NoSQLではデータに不整合が発生したり、ノードが故障していたりする場合には、緩やかにバックグラウンドでデータの補修を行う事が機能（リードペア）が搭載されている場合があります。

NoSQLの基本的概念と技術（データ分割）

シャーディング（Sharding）

あらかじめ、サーバ毎に保存するデータのキーの範囲を設定しておくことで、検索や管理を容易にする機能です。



他、アルファベットA～G、H～N、O～Z
ID 1 ～1000、1001～2000…など

データの偏りがあった場合、適切な負荷分散ができません。なので、データ単位を小さくし、ノード数より多くすることなどにより、データの偏りを減らす工夫が必要です。

BigtableやHbaseは再分割、再配置を自動的に行います。

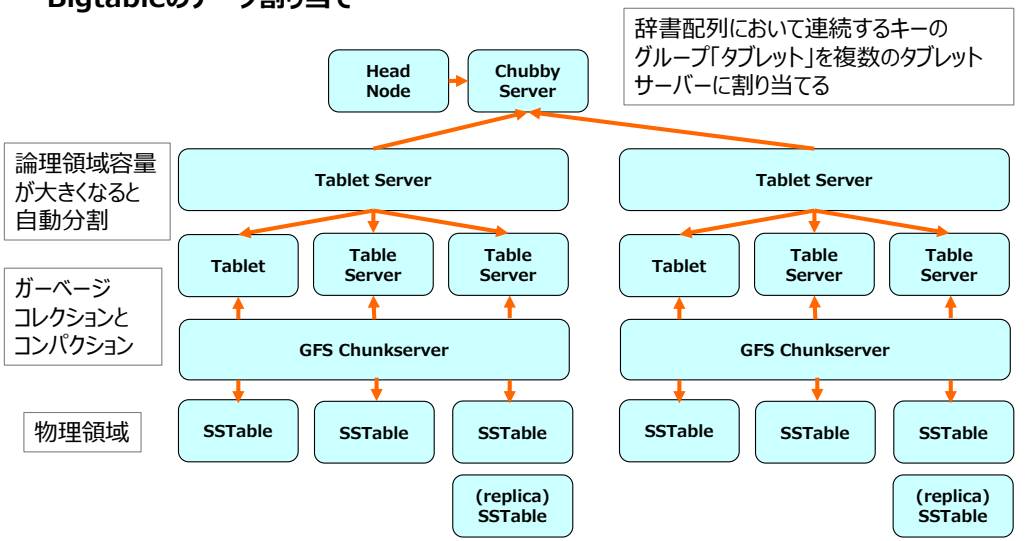
・説明の流れ

すでにご紹介したシャーディングもデータ分割の一手法です。

データの中身によって、偏りが生じる場合もあるため、最初から均等になる設計や再配置、再分割機能があるNoSQLを選定することも重要です。

NoSQLの基本的概念と技術（データ分割）

Bigtableのデータ割り当て



・説明の流れ

ビッグデータのデータ割り当てとして、Bigtableの仕組みを見てみましょう。

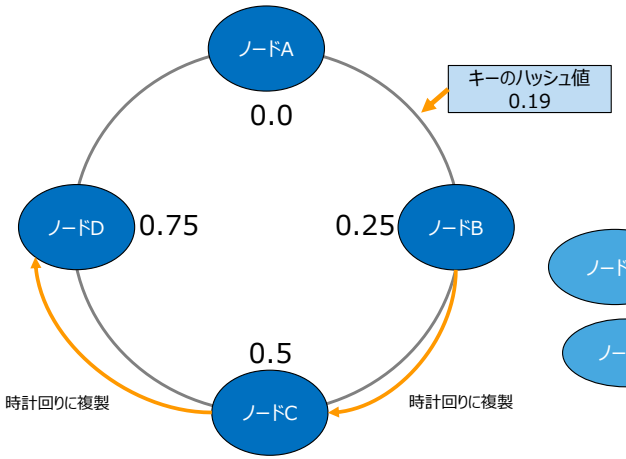
キー文字列の順番が連続するものをタブレットとしてグループ化する機能と不要となったデータを管理する機能が充実しています。

Computer Software Association of Japan

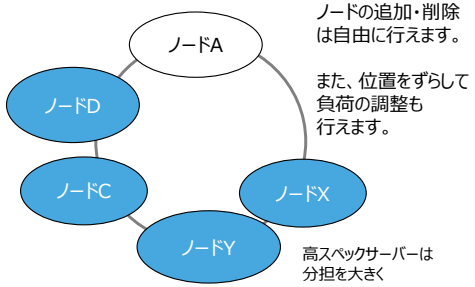
NoSQLの基本的概念と技術（データ分割）

コンシステントハッシング Consistent Hashing

分散型システムを構成するノードを、論理的に輪を形作るよう配置します。データのキーのハッシュ値をハッシュ関数によって求め、不規則な値であるハッシュ値を元に、データを保存するノードを選択します。ノードの位置を変更することで、負荷の代償を変更することも可能です。



例えば、0.0から1.0の間のハッシュ値の場合、一番近い時計回りのノードに割り当てられるようなルールを定めます。



ノードの追加・削除は自由に行えます。

また、位置をずらして負荷の調整も行えます。

高スペックサーバーは分担を大きく

CSAJ

・説明の流れ

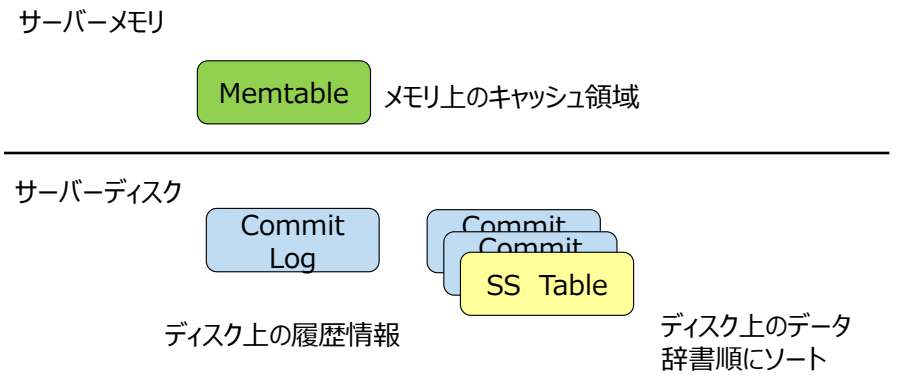
データの分散をデータの内容に依存すると偏りが生じるため、データ内容には依存せず、特別に付与したハッシュコードによって均等に分散させる手法があります。

これをコンシステントハッシングと呼びます。

NoSQLの基本的概念と技術

ストレージレイアウトの技術

・LSM-Tree（Log Structured Merge Tree）
メモリを使って応答性能を高めつつ、ハードディスクにデータを格納することでデータの安全性を保つ機能。



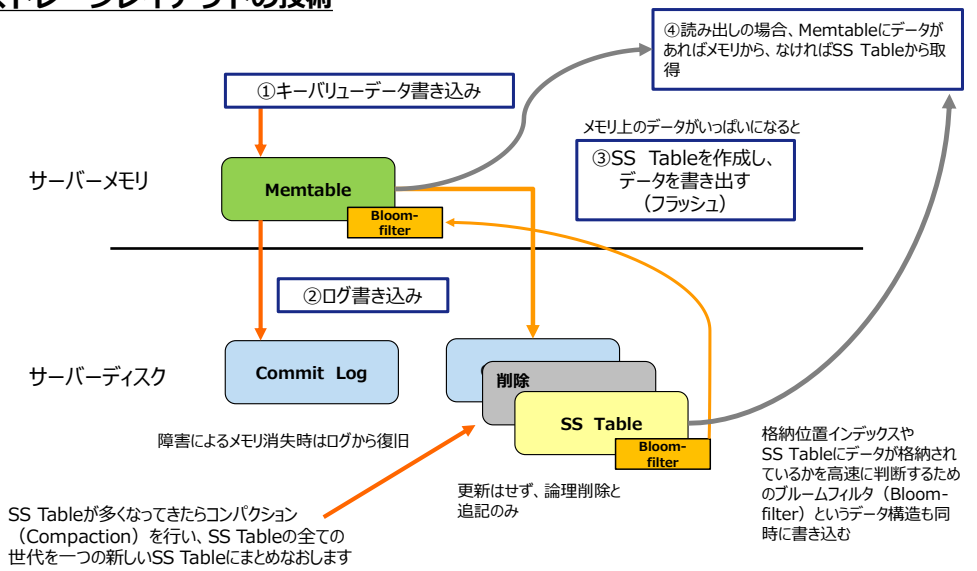
・説明の流れ

NoSQLのストレージはどのような仕組みになっているでしょうか。

LSM－Treeという技術では、メモリを使って応答性能を高めながら、データをSS Tableにというメモリ上の入れ物に蓄積し、一定量たまった段階でディスクに記録する手法を取っています。

NoSQLの基本的概念と技術

ストレージレイアウトの技術



CSAJ

・説明の流れ

LSM－Treeをもう少し詳しくみましょう。

メモリ上のデータ操作時にCommiteログも同時にディスクに記録し、障害時の復旧に利用します。

必要なSS Tableのデータはメモリにあればメモリから、なければディスクから読み込み、 SS Tableの所在を記録したBloom－filterも同時に記録する事で高速な応答性能を実現します。

第４章

ビッグデータコア技術

分散処理

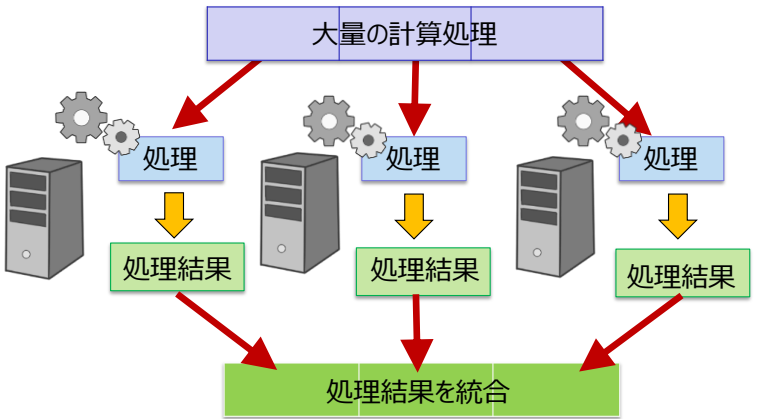
ビッグデータの基盤システムを学ぶ

基本的なビッグデータの基盤システムを理解する

分散処理とは

分散処理とは

ビッグデータを分割し、複数のサーバで同時に並行して処理した後、処理結果を統合することで全体の処理結果を生成する処理方法です。



・説明の流れ

ビッグデータの蓄積技術として、分散処理という技術があります。

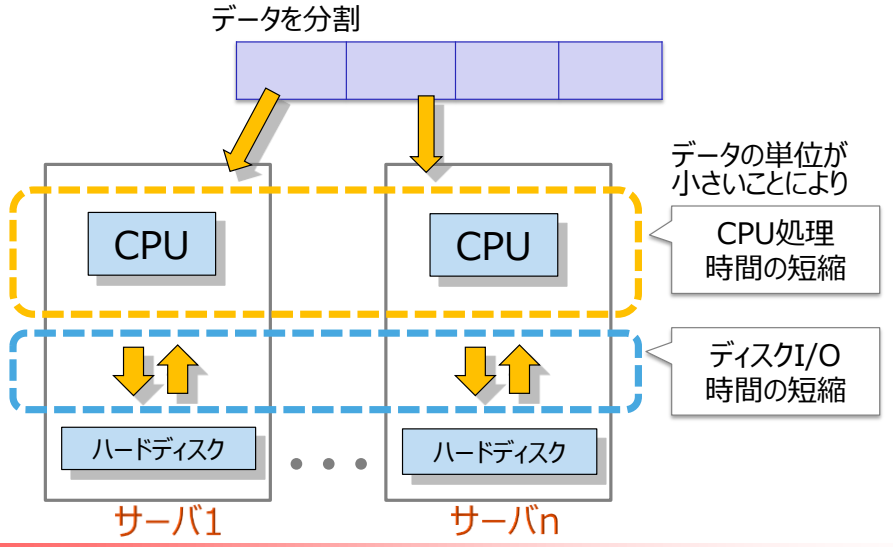
分散処理とはどのようなものでしょうか。複数のサーバーで処理を分担するという仕組みになります。

CPUやメモリ、ディスクを複数サーバーで占有して利用できるため、個々の処理は高速になります。

分散処理とは

分散処理による処理時間の短縮

サーバでは、分割されたデータを処理するので、CPUでの処理時間や、ディスクからの読み込み、ディスクへの書き込みの時間が短縮されます。



・説明の流れ

分散処理を図示するとこのようになります。1台で処理するデータ量が少なくなるため一台当たりのCPU処理時間や、ディスクI/Oは短縮されます。

分散処理のメリット・デメリット

分散処理のメリット

ビッグデータの処理時間を短縮することができます。
スケールアウトによる拡張が可能で、比較的低コストになります。
一台の故障が全体の停止には直結しないので、稼働率が高いです。

分散処理のデメリット

データが複数コンピュータにまたがるので、厳密な整合性が必要なオンライントランザクション処理に向きません。
少量のデータ処理には効果が薄いです。
システム全体の性能は、ネットワークの帯域幅や稼働率から大きな影響を受けます。

・説明の流れ

分散処理のメリットとデメリットを考えましょう。

大量データを扱うことができる、スケールアウトしやすいなどのメリットがある一方で、ノード間の整合性の担保や、ネットワーク通信量の増加、結合処理の負荷などが発生することは念頭に置く必要があります。

・ポイント（絶対に覚えてほしいこと、など）

分散処理も万能ではなく、トレードオフがあることを理解する。

Hadoopとは

Hadoop

Hadoopはデータベースではなく、複数のソフトウェアからなる分散処理フレームワークです。
一定期間のデータを一括処理する、バッチ向けのフレームワークです。

2つの主要プロダクト

・ファイルシステム(HDFS) : Hadoop Distributed File System
バッチ処理向けに高いスループットを分提供する分散ファイルシステム。

・分散処理アルゴリズム(MapReduce):Hadoop MapReduce
複数のサーバでの並列分散処理するプログラミングのためのソフトウェア。
HDFS上のファイルを読み込み、集計結果を別のHDFS上に書き込む。

・説明の流れ

分散処理技術の中核となるHadoopというオープンソースの製品を紹介します。

Hadoopは分散処理を実現するためのフレームワークです。

2つの主要プロダクトを中心として様々な関連ソフトが存在します。

Hadoopとは

Hadoopの特徴

オープンソースソフトウェアフレームワークで、非構造化であるビッグデータの解析や大量のバッチ処理に用いられます。 Googleによる開発の後、Apache Software Foundationが現在は開発、公開を行っています。

HadoopはLinuxで動作します。また、システムはJavaで構築されているため、JVMが必要です。

MapReduceフレームワークやHDFSを操作するためのJavaのAPI（クラスライブラリ）が提供されています。

専門スキルが導入に必要なため、商用のサポートサービスを提供している会社があります。 (クラウデラ、ホートンワークス、MapRなど)

CSAJ

・説明の流れ

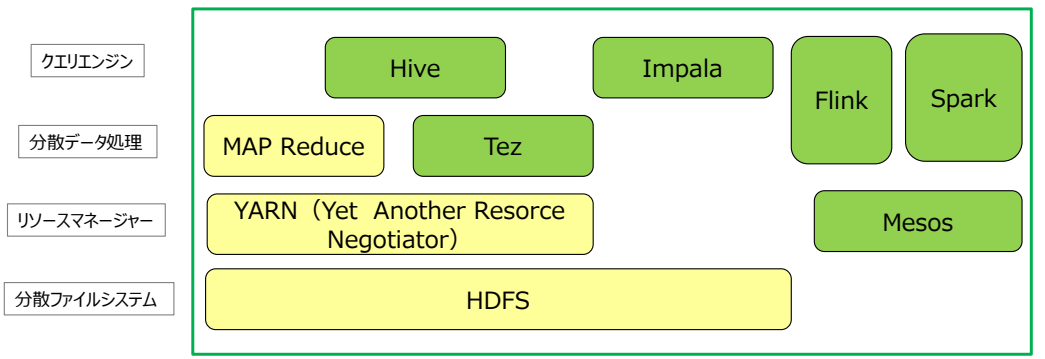
Hadoopの特徴を押さえましょう。

Hadoopの基本構成

Hadoopの基本コンポーネントは、

・分散ファイルシステムのHDFS
・リソースマネージャーのYARN
・分散データ処理を行うMapReduce

です。その他のシステムは、Hadoopから独立して開発されたものです。



・説明の流れ

Hadoopを構成する製品群を確認しましょう。

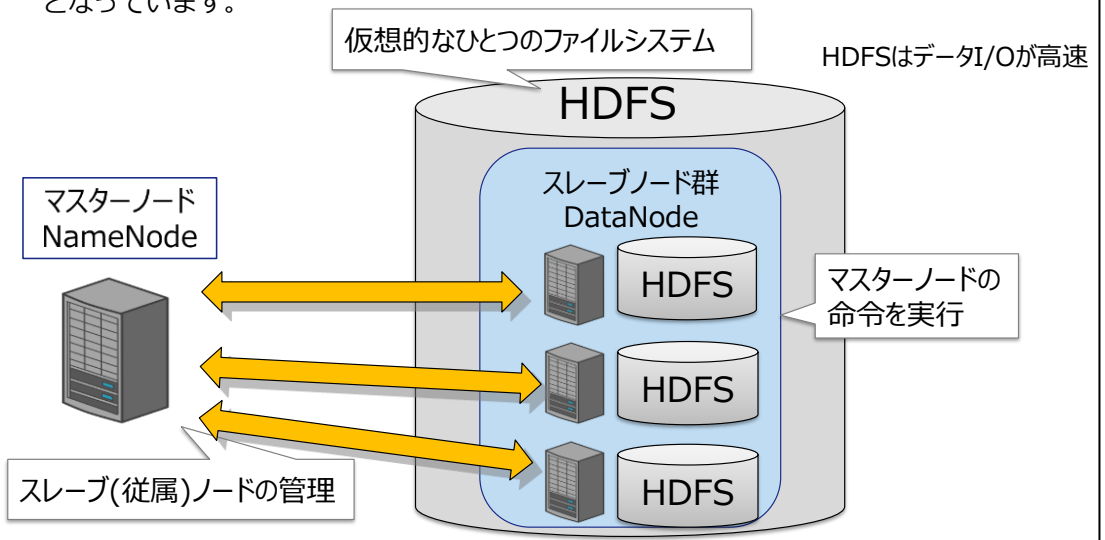
リソース管理のYARNや、高速集計の、Impala、Tez、SQLを扱えるHiveなどもあります。

機械学習のライブラリが充実したSparkもよくHadoopと共に利用されます。

HDFS (Hadoop Distributed File System)

HDFS

HDFSはマスタ型で、データを管理するスレーブノード群と、スレーブノードを管理するマスターノードから構成されています。全体で、仮想的な一つのファイルシステムとなっています。



CSAJ

・説明の流れ

HDFSの仕組みを理解しましょう。

HDFSはデータの位置を管理するマスターノードとデータを格納するスレーブノードで構成されています。

利用する側からは、複数の分散データが一つの巨大ファイルシステムとして見えます。

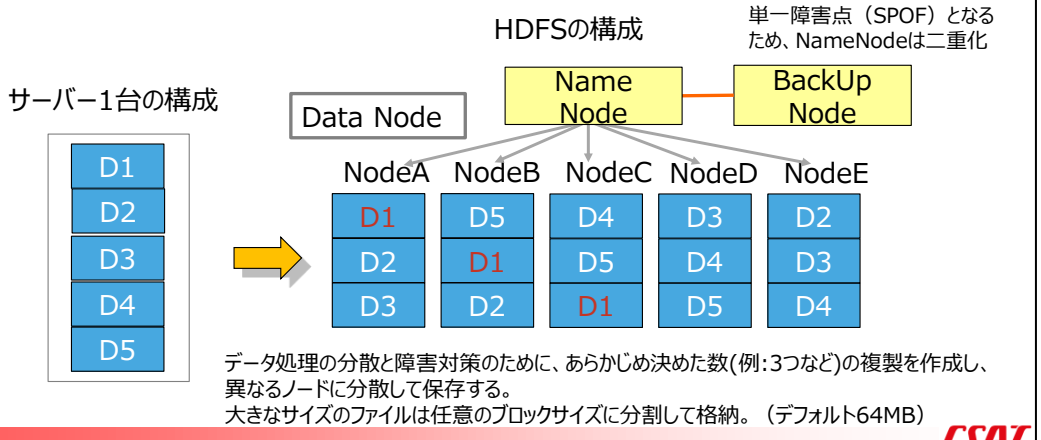
HDFSはマスタ型

マスタースレーブ型

NameNode、DataNodeに役割が分かれます。

NameNodeはメタ情報(他のノードのデータやブロックの対応関係など)を保持するマスターノードです。単一障害点となるため、バックアップノードを作成します。

DataNodeがデータ(ブロック)を保持します。障害対策として、同じデータを複製し、ノードを跨って保存されます。



データ処理の分散と障害対策のために、あらかじめ決めた数(例:3つなど)の複製を作成し、異なるノードに分散して保存する。
大きなサイズのファイルは任意のブロックサイズに分割して格納。（デフォルト64MB）

・説明の流れ

HDFSはノードが破損しても、データ格納時にデータを分割し複数サーバーに配置することで、復元を可能としています。

自動でデータ配置を催事調整し、復元します。

ただし、マスターノードだけは単一障害点となるため、冗長化構成をとる必要があります。

HDFSの読み書き

HDFSへの書き込み・読み出し

データは、クライアント側で任意のサイズのブロックに分割され、NameNodeがどのDataNodeに格納するか指示し、書き込みを行います。
読み出しは、 NameNodeがクライアントに対して目的のデータが保存されているDataNodeを示し、直接ブロックを読み出します。

データの複製と耐障害性

データ処理の分散と障害対策のために、一定数複製を作成し、複数のノードにまたがってデータを保存します。障害時は、障害のあるノードをクラスタから切り離し、切り離されたノードにあった複製の分を、他のノードに複製します。

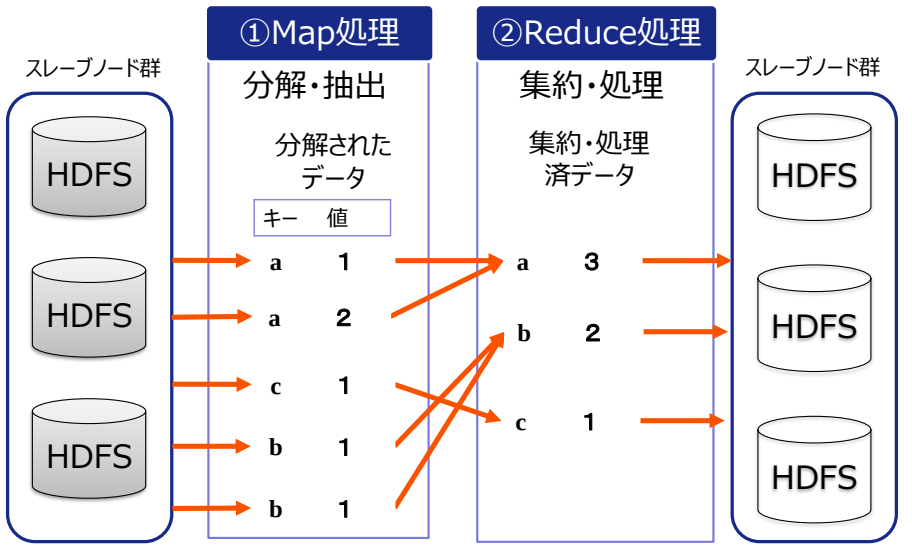
リバランシング

ノードの追加や削除された場合に、データの分散に偏りが生じないようにデータの再配置を行います。

・説明の流れ

HDFSはデータの複製と再配置を自動で行う事ができます。

MapReduce処理とは



・説明の流れ

Hadoopでデータを集計する場合はどのように行うのでしょうか。

Map処理でデータをキーと値に分割、抽出し、Reduce処理で同じキーのデータを集約、集計します。

MapReduce処理とは

MapReduce

大量データを分割し、複数のサーバで並列処理するための方法です。複数のスレーブノードで同一の処理を実行します。次の2段階の処理に分かれます。

①Map処理
データを小さく分割し、必要な情報を選んで出力します。出力データはキーバリュー型の構造となります。

②Reduce処理
キーごとにデータを集約し加工を行なった上で、HDFSに出力します。

Map処理とReduce処理の開発には、Hadoopが提供するJavaのAPI(クラスライブラリ)を利用します。複雑な分散処理アルゴリズムや障害対応処理などの実装は不要で、開発者は業務ロジックの開発に集中できます。

・説明の流れ

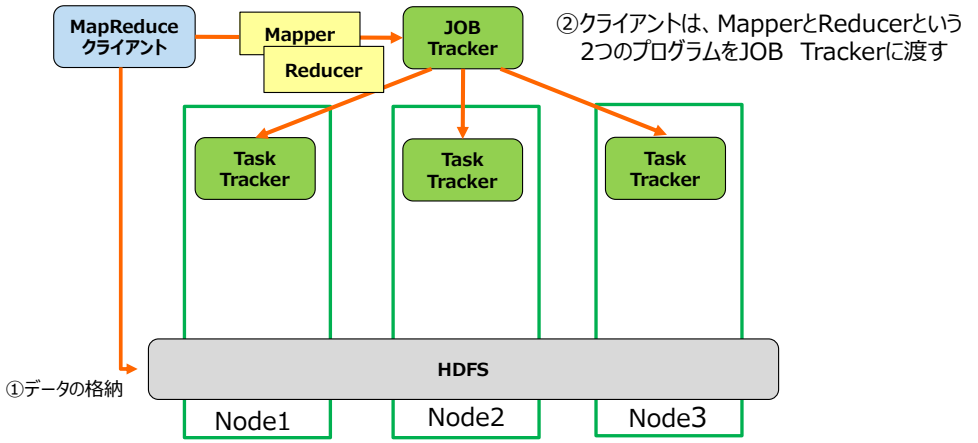
MapReduce処理は、複数サーバー上で動作します。

実際には小さなJavaプログラムが、各サーバーノードで実行されます。

MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

MapReduceのマスターはJobTrackerと呼ばれ、全てのMapReduce処理工程の調整・管理とスレーブとなるTaskTrackerの維持管理を行います。



・説明の流れ

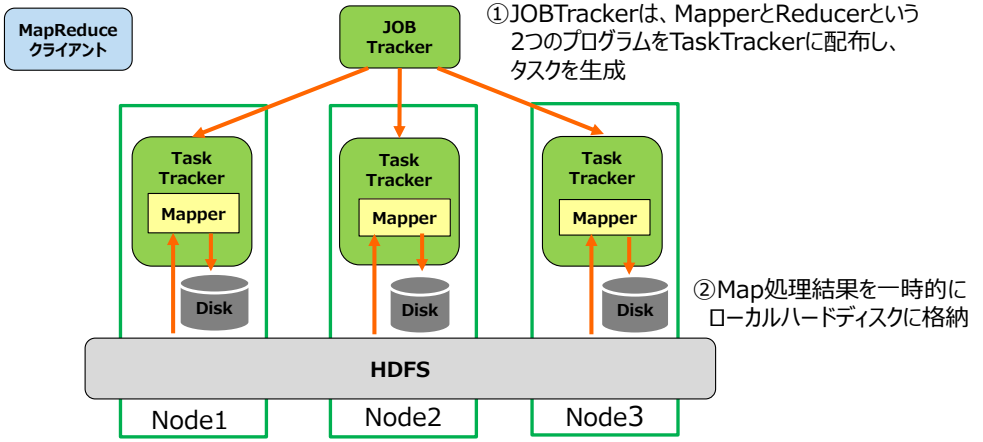
MapReduceの動作の流れを追ってみましょう。

集計要求を発行したいタイミングで、MapReduceクライアントからMapperとReducerという２つのJavaプログラムをJOB Trackerに渡します。

MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

JobTrackerはMapperとReduceプログラムをTaskTrackerに配布し、HDFSブロックと同じ数だけMapタスクを生成します。 Mapタスクは、データが存在するTaskTrackerに割り当て、必要なデータ転送のみを行います。



・説明の流れ

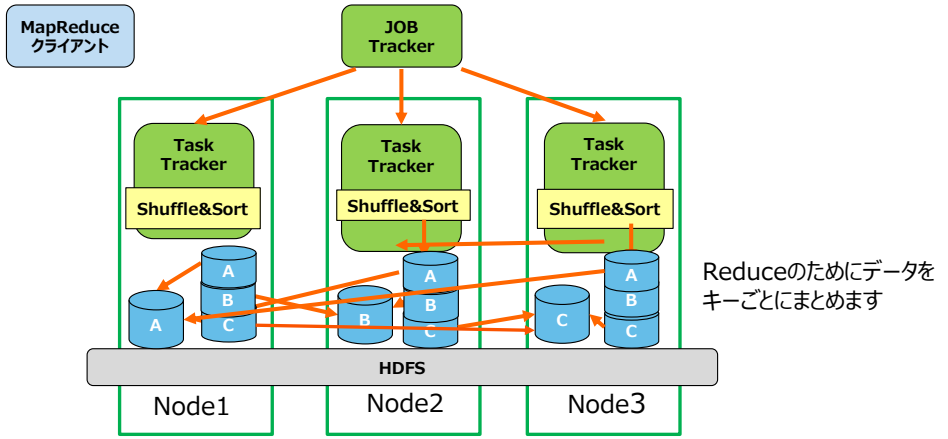
Mapperは各サーバーノードに配布され、HDFSからデータを抽出します。

抽出したデータは一時的にローカルハードディスクに格納されます。

MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

JobTrackerは、ある程度処理が進むとShuffle&Sort処理を起動します。
Mapタスクにより処理された中間データを、キーの値よって、ノードを跨っているものを含めて1箇所にグループ化します。



・説明の流れ

Mapで抽出したデータを各サーバーで処理すると各サーバー間で都度通信が発生してしまいます。

これを避けるため、キーごとに各ノードにデータを集め、同じキーについては、同じサーバーで処理できるように調整します。この作業をシャッフルと呼びます。

・ポイント（絶対に覚えてほしいこと、など）

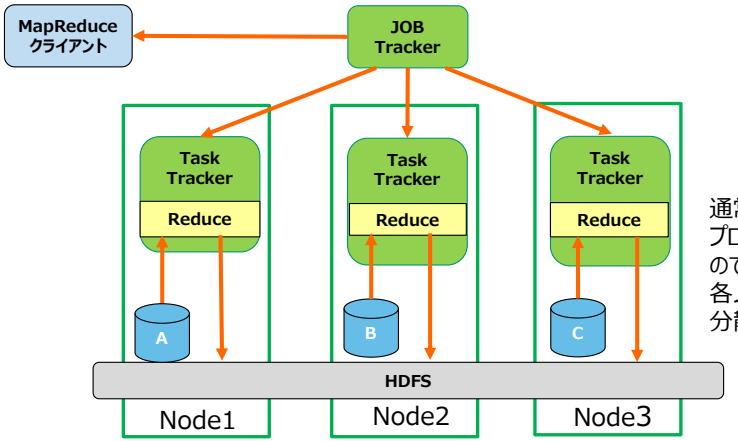
処理によってはシャッフルが必要ない処理と、必ず発生するものが出てくる。

シャッフルが発生するかどうかを常に意識してプログラミングをすれば、思わぬ速度低下を回避できるだろう。

MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

処理がさらに進むとJobTrackerはReduceタスクをTaskTrackerに割り当てます。TaskTrackerはReduceタスクを実行し、処理結果をHDFSに書き込みます。全てのタスクが終了するまで繰り返し行います。



通常のプログラミングでは、プログラムがデータを呼び出すのですが、プログラムの方を各ノードに配布している点が分散処理のポイントです。

・説明の流れ

次に、集計処理をReducerが行います。

一段目の集計で処理が完結しない場合は、トーナメント方式で、集約をかけ続けます。

・ポイント（絶対に覚えてほしいこと、など）

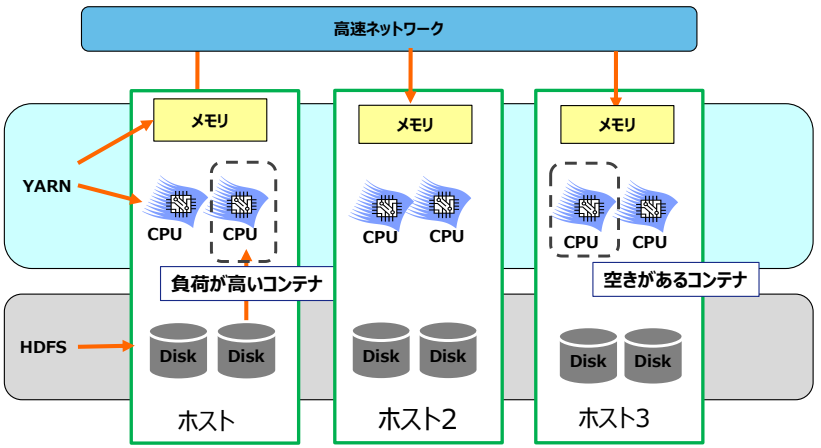
MapReduceの特筆すべきは、プログラムが各ノードからデータだけを収集する形ではなく、プログラムを各ノードに配布して、各ノード上で処理する点である。

つまり、集計処理という業務ロジックまでを分散処理で行っていることになる。

分散ファイルシステムとリソースマネージャー

YARN

CPUやメモリなどの計算リソースはリソースマネージャーであるYARNにより管理されます。
CPUコアやメモリをコンテナと呼ばれる単位で管理します。
分散アプリケーションが実行された時に、クラスタ全体の負荷のバランスを見てコンテナが割り当てます。



・説明の流れ

各ノードに処理を分散させる際に、負荷がかかるノードとそうでないノードはどうしても発生してしまいます。

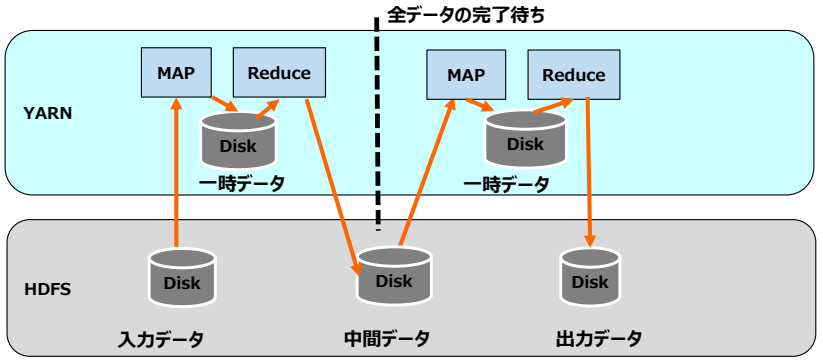
そこで、リソース管理をするYARNが各ノードの負荷状況を常に監視し、効率的に処理を割り振っていきます。

分散データ処理とクエリエンジン

Hive on MapReduce

MapReduceはJavaプログラムをデータに対し実行できるため、加工には便利です。SQLに対応させる場合、専用に設計されたクエリーエンジンApache Hiveを利用します。

Hiveは応答性能が重要視される場合や少量データを扱う場面などではには不適です。



データ処理ステージが変わるタイミングで待ちが発生。
中間データを毎回Diskに保存するため低速

・説明の流れ

MapReduceはJavaプログラムから呼び出しますが、ここがネックとなって使いにくいとの声もありました。

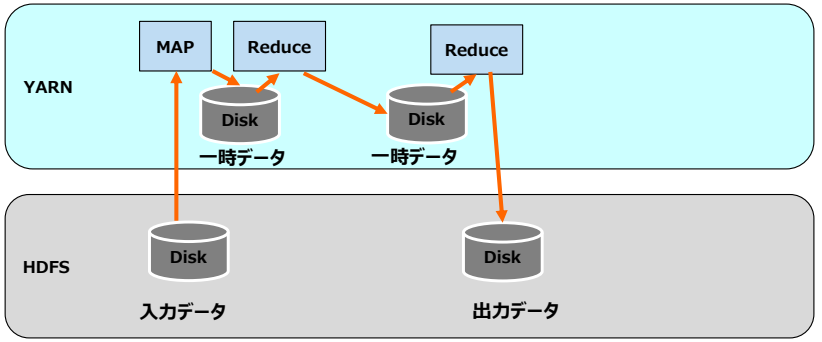
そこでSQL言語で要求を投げられるHiveという仕組みが考案されました。

分散データ処理とクエリエンジン

Hive on Tez

Apache TezはHiveの高速化するために開発されたコンポーネントです。
Mapreduceの仕組みを調整し、処理段階の終わりで他のプログラムの情報を保持せずに後続の処理へと受け渡す仕組みに変更しました。

ステージの合間にあったディスクへの書き込み、待ち、Map処理を省略



現在、Hive on Sparkというものも存在します。RDDがMapreduceに代わる存在として活用されています。

・説明の流れ

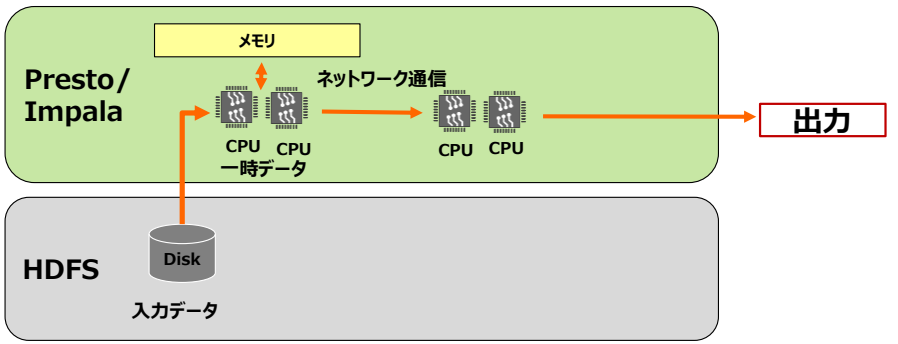
MapReduceはこれまで処理できない量のデータを処理できる点において特筆すべき能力を有していますが、処理の各段階でローカルファイルシステムにデータを書き出すため、レイテンシは下がってしまいます。

これを打破するために余分なMap処理とディスク書き込みを省略することにより、高速化を図っています。

分散データ処理とクエリエンジン

Impala / Presto

ImpalaとPrestoは、Hiveの高速化ではなく、対話型クエリ実行に特化させることで、SQLに対応させた製品です。バッチ処理ではなく、最大瞬間速度を最大にするため、あらゆるリソースを有効活用するよう設計されています。



自前の分散処理機構を実装し、マルチコアを活用しながら並列化処理を実行

・説明の流れ

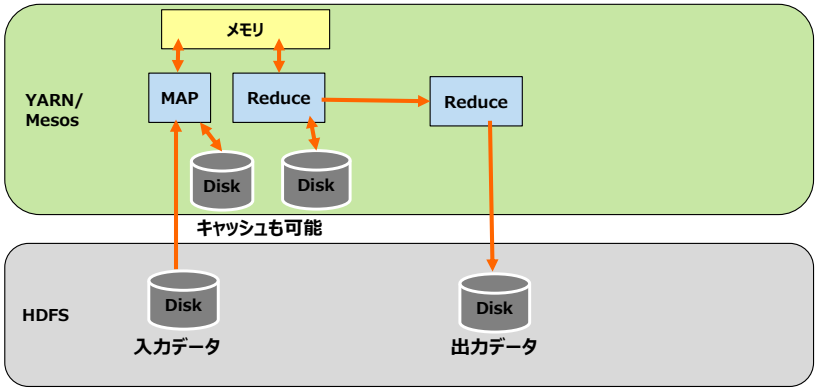
Hiveでも応答性能に応えられない場合もあったため、クエリエンジンとしての製品開発も進みました。

それらが、ImpalaとPrestoです。HDFSから抜き出したデータを一度メモリにあげ、最後までメモリ上で処理を継続します。

分散データ処理とSpark

Spark

Apache SparkもMapReduceより効率良くデータ処理をするために開発されました。メモリの使用量増やすことで高速化を実現します。Sparkの実行にはJavaランタイムが必要ですが、Java、Scala、Python、Rからも呼び出すことができます。



Sparkは中間DBにデータを書き込むことなくメモリ上に保ち続けます。
障害でデータが失われるともう一度最初からやり直します。

・説明の流れ

一方で、分散処理フレームワークに適した、データモデルも考案されました。

分散処理で扱える小さなデータ単位RDDを介して処理を行うSparkという言語です。

SparkではMAP、Reduceという仕組みを保ちながら、処理自体をオンメモリで完結させることが可能です。

・ポイント（絶対に覚えてほしいこと、など）

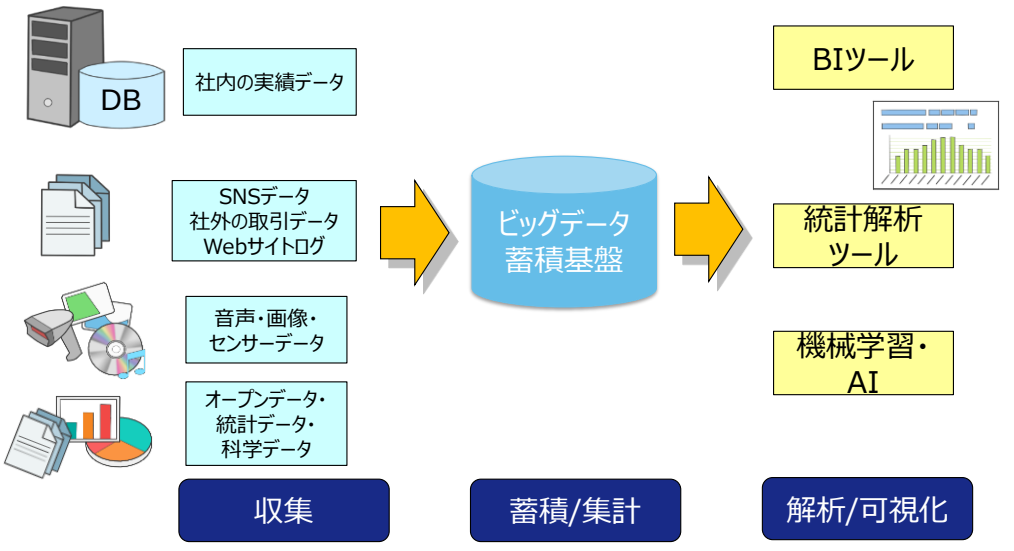
得られた結果明示的にディスクに保存する場合は、キャッシュすることも可能。

第５章

ビッグデータにおける データ解析

ビッグデータで役に立つ分析手法を学ぶ

ビッグデータにおけるデータ解析



・説明の流れ

ビッグデータにおけるデータ解析の流れを復習しましょう。

アドホック分析ツール

データを可視化するためのソフトウェアは多種多様に存在します。

アドホック分析ツール

行錯誤を繰り返しながら、繰り返しデータを見ていく際に利用するものです。
アドホック分析環境には次のようなものがあります。

・Jupyter Notebook

Python、Ruby、Rなどのソースを書き留めながら実行することが可能。
matplotlibライブラリを利用してグラフを作成することもできる。

・Apache Zeppelin

Spark/Hadoopなどの分散処理システムに対してコードを実行し、
実行結果をグラフとして描画することが可能。
シェル／SQL／scala／python言語に対応している。

https://zeppelin.apache.org/

・説明の流れ

アドホック（随時、自由）に集計するアドホック集計ツールなどもオープンソースで存在し、ビッグデータ処理後の可視化に利用されています。

ダッシュボードツール

ダッシュボードツール

傾向の把握が進むと、定期的にクエリを実行し、レポートやグラフなどを用いたダッシュボードを作成するツールです。

対話的なBI商用ツールとしてTableau、ClikViewなどが存在します。
オープンソースのダッシュボードツールも存在します。

・Redash
Python製のダッシュボードツール。
SQLクエリの実行結果を直感的に可視化できる。搭載したDBに結果を記録するため、表示自体は高速だが、クエリ結果が大規模になるとエラーや遅延が生じる。

・Superset
Python製のWebアプリケーションで対話的なダッシュボードを作成する。
集計は外部ストレージ「Druid」で行い、リアルタイムな出力を行える。

・Kibana
JavaScript製の対話的可視化ツール。
Elasticsearchのフロントエンドでよく用いられる。

・説明の流れ

ダッシュボードツールは、分析結果を共有する対話的なBI商用ツールとして、オープンソースの他、商用製品でも存在しています。

データ活用：一般的な統計分析手法

度数分布と ヒストグラム	データを範囲に区切って表したもので、データの分布を 知ることができる
平均と標準偏差	状況を特定することができる
正規分布	統計的なデータを元に、可能な予測を立てることができる
標本調査 （全体像の推測）	標本（一部のデータ）を元に、全体像を把握できる
標本平均	標本の平均を求めることで、全体の平均を推測できる

・説明の流れ

統計学でよく使われる手法を知ることはビッグデータ分析の助けになります。簡単に見ていきましょう。

度数分布とヒストグラム

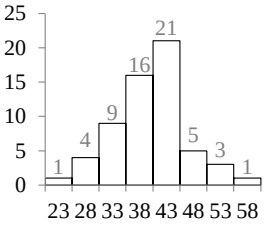
度数分布

度数分布表を用いて表します。
※度数分布表：データをいくつかの範囲に分け、その範囲に入るデータの個数を書き表した表。
データがどのような値を中心にして、どのように広がっているのかを調べることができます。

値）	人数
21～25（23）	1
26～30（28）	4
31～35（33）	9
36～40（38）	16
41～45（43）	21
46～50（48）	5
51～55（53）	3
56～60（58）	1

ヒストグラム

ヒストグラムとは、データの分布状態（度数分布）を表す棒グラフのことです。
度数分布の他の表現方法としては、箱ひげ図があります。



・説明の流れ
度数分布とヒストグラムを紹介します。

平均と標準偏差

・平均：データの代表値としてよく利用される。
　　平均の代わりに、最頻値、中央値を用いることもある。

$$\text{平均値} = \frac{\text{データの値の総計}}{\text{データ数}}$$

・分散：データがどのくらいばらついているかを表す。

$$\text{分散} = \frac{(\text{各データの値} - \text{平均})^2 \text{の総計}}{\text{データ数}}$$

… 平均との差の２乗の平均

・標準偏差：そのデータの傾向や性質を把握するために利用される。
　・異なるデータ間でのデータのばらつきの度合いを比較できる
　・平均値からのばらつきの幅を測定できる。

$$\text{標準偏差} = \sqrt{\frac{(\text{データの値} - \text{平均})^2 \text{の総計}}{\text{データ数}}}$$

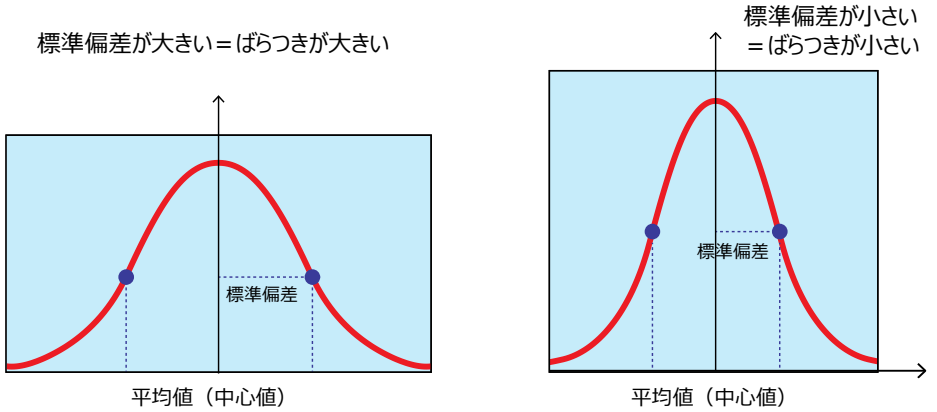
… 分散の平方根

・説明の流れ
　平均と標準偏差を紹介します。

正規分布

正規分布とは

・確率分布の１つ。
・平均値（または中央値）を中心に左右対称な、釣鐘型の分布。
・平均値で最も発生確率は高くなり、平均値から離れるほど低くなる。
・正規分布をしていると想定できるデータに関しては、平均値と標準偏差さえわかれば、どのように分布が広がっているかを推定できる。



・説明の流れ

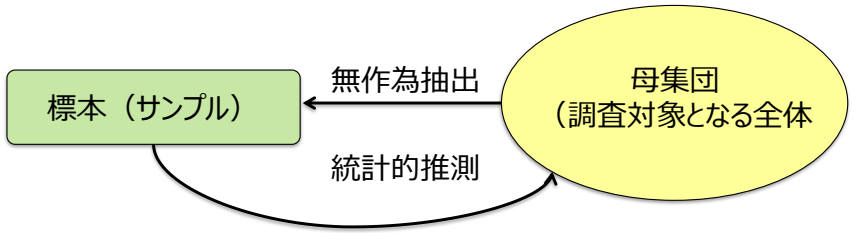
正規分布は、偏りのないバラツキと、どれくらい乖離があるかを調べる際に便利です。

例えば、適正在庫のあり方と、現状の乖離などを見たい場合に使います。

標本調査と標本平均

標本調査とは

標本、つまり大量のデータ（母集団）の一部を入手、観測した際に、その母集団を推測するために仮説検定の考え方を用いて行う統計的推定の手法のことを言います。



標本平均とは

標本データの平均値を利用することで、母集団の平均を推測することができます。母集団の傾向を推測するために利用されます。

…（標本平均）＝（抽出した標本データの合計）÷（抽出した標本データ数）

・説明の流れ

ビッグデータ登場以前は全数分析ができませんでした。

その代わりに標本を抽出して、全体を推測する手法がとられてきました。

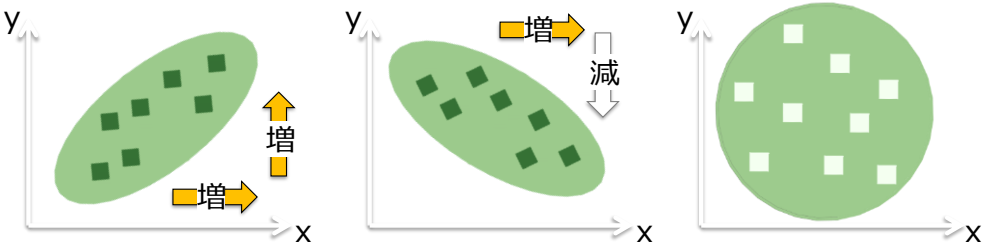
相関関係

相関関係とは

一方の値の変化に伴って、もう一方の値が変化するという、２つの値の関係のことを相関関係と言います。

・正の相関：２つのデータのうち、一方の値が増加すると、もう一方の値も増加するような関係。

・負の相関：２つのデータのうち、一方の値が増加すると、もう一方の値は減少するような関係。



正の相関（右上がり）　負の相関（右下がり）　無相関

・説明の流れ

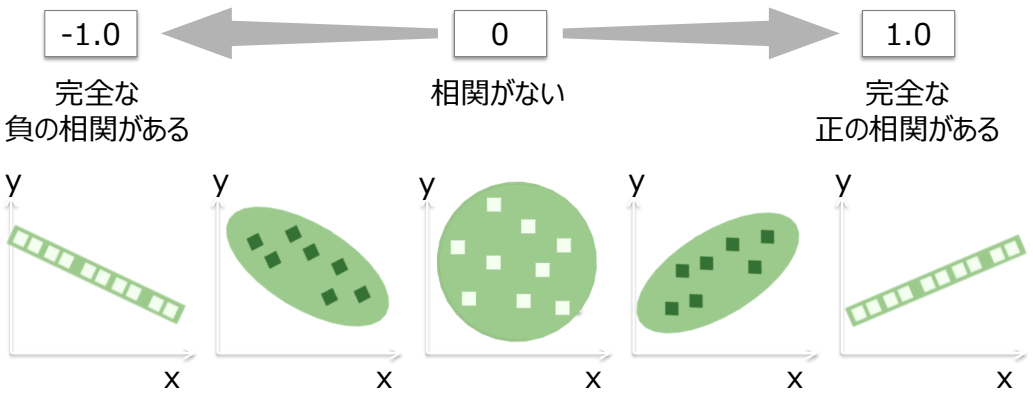
相関関係は、ある数値とある数値の関連を調べることで、一方の増減からもう一方の増減を予測することが可能になる手法です。

相関係数

相関係数とは

２つのデータ間の相関関係の強さを表す値のことで－１から１までの値を取ります。
相関係数が０に近いほど相関は弱く、１に近いほど相関は強くなっています。

…（相関係数）＝（共分散）÷（一方の標準偏差 × もう一方の標準偏差）



・説明の流れ

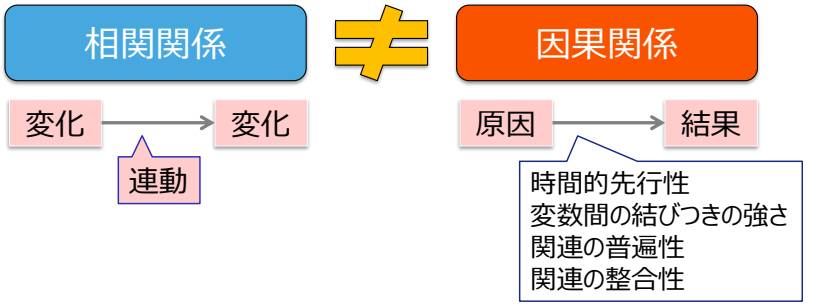
相関係数は相関を数値化した値で、1に近いほど正の相関があると言えます。

相関関係と因果関係

・相関関係：一方の値が変化すると、もう一方の値も変化するという２つの値の関係のこと。

・因果関係：２つ以上のものの間に原因と結果の関係が成り立っていること。つまり、一方によって、もう一方が引き起こされるという関係。

相関関係があっても因果関係があるとは限りません。



・説明の流れ

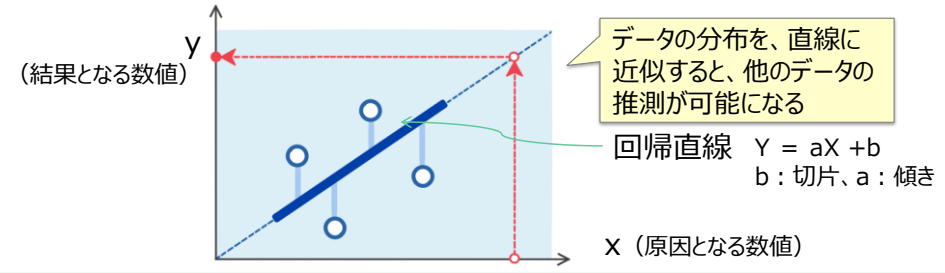
相関関係と因果関係は異なるので注意が必要です。

因果関係は原因に対して必ず結果が発生する場合を指しますが、相関関係では相関があるというだけで、必ずしももう一方の値が連動されることを保証するものではありません。

回帰分析と重回帰分析

回帰分析とは

「原因となる数値」と「結果となる数値」との関連性を、統計的手法によって分析することを言います。
回帰分析を行うことで、一方の数値（説明変数）の変化から、もう一方の数値（被説明変数）の変化を推測することができ、仮説を立てることが可能になります。式にすると、 $Y = aX + b$ のように表されます。



重回帰分析とは

１つの目的変数を複数の説明変数を用いて推測する統計的手法のことを言います。
式にすると、 $Y = a + b_1X + b_2X + b_3X + \ldots + b_nX$ のように表されます。

・説明の流れ

データをグラフ化した場合に、傾向などを見たい場合があります。

数値が直線に集まっている場合は、線形回帰もしくは、一次回帰直線を引くことができます。

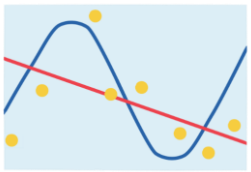
何らかの数式に表せる形なため、線形だけでなく2次曲線や3次曲線、4次曲線などと傾向が一致する場合もあります。

この場合は2次回帰曲線、3次回帰曲線などとなります。

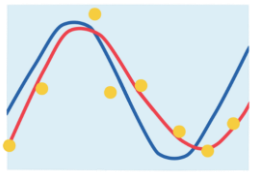
また、重回帰分析では、一つの説明変数ではなく、複数の説明変数で数式を形成するものです。

多項式回帰曲線のイメージ

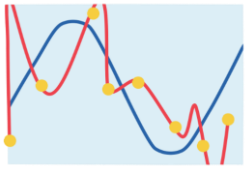
回帰曲線を用いて、データをモデル化することを考えます。



1次回帰線



３次回帰線



９次回帰線

ここでは３次回帰曲線を用いて表すのが適当だと考えられます。
上の図からも分かるように、９次回帰線は全ての点を通っていますが、実態とはかけ離れています。従って、新たな点が現れた時にその点が９次回帰線上にある確率は却って下がってしまい、予測には向いていません。
これを、過学習（オーバーフィッティング）といいます。

・説明の流れ

多項式回帰曲線の例です。

このように次数を増やせば全ての点を通す数式も得られますが、逆に予測精度が落ちてしまいます。

これをオーバーフィッティングと呼びます。

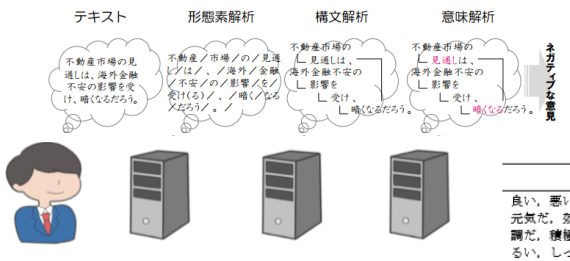
テキストマイニング

テキストマイニングとは

大量のテキストデータから、トピックの傾向や、役立つ知識、情報などを見つけ出す分析技術のことを言います。

まず膨大に蓄積されたテキストデータを単語や文節（形態素）に分解し、それぞれの形態素の出現頻度や相関関係を解析します。
そして、係り受けなどの関係や時系列の変化といった構文解析を行うことで、結果的に客観的な意味内容を掴むことを目的とします。

図表　テキストマイニングの流れ（イメージ）



図表　評判辞書のキーワード例（抜粋）

ポジティブ単語	ネガティブ単語
良い，悪い（否定），良好だ，活発だ，元気だ，効果的だ，最適だ，最高だ，順調だ，積極的だ，調子よく，適切だ，明るい，しっかり，安全だ，快調だ，経済的だ，上々だ，心強い，抜群だ，必要だ，便利だ，満足だ，等	悪い，良い（否定），暗い，異常だ，割高だ，苦しい，最悪だ，最低だ，弱めだ，心配だ，不安だ，不十分だ，不調だ，危険だ，苦しい，酷い，少な目だ，心細い，辛い，大変だ，難しい，悲惨だ，不要だ，無理だ，異様だ，等

出典：「テキストマイニングによる国土政策評価手法の研究」国土交通省

・説明の流れ

ビッグデータで扱うデータに、通常の会話など言語データも含まれてきます。

これらを数値化するにはどのような手法があるでしょうか。

実際には文章を単語や文節などの小さい単位に分割（形態素解析）して語句の出現頻度、語句と語句の関連性から文章内容の傾向を読み取ることが可能です。

例えば「やばい」という言葉が、ポジティブな言葉と一緒に出現するかどうかを測ったり、ある語句グループが出現したりする群をクラスタリングして、対応を打つなどの使い方があります。

テキストマイニング

テキストマイニングをすることで、テキスト情報から何らかの有用な情報を得ることができます。

例 ・自社の評判と他社の評判を分析、比較する。
→トピックの傾向を把握したり、新たな気づきを得たりできる。

・売上やキャンペーンデータについて、時系列で比較する。
→マーケティングなどに有用な情報を得られる。

・説明の流れ

テキストマイニングをすることで、Twitterの発言やサイト上の口コミから、キャンペーンが効果的であったか調べることが可能になるかもしれません。

第６章

ビッグデータとAI、機械学習

クラウド、IoT

AIの発展

AI（artificial intelligence）とは

コンピュータ上などで、人工的に人間と同様の知能を実現させようという試み、あるいはそのための一連の基礎技術のことを言います。

・エキスパートシステムとルールエンジン
エキスパート（専門家）がルールをつくりプログラムに実行させ、推論機能を適用することで結論を得る。
エキスパートシステムは大量の既知情報を処理し、関係性を導ける。

・事例ベース推論（CBR）
類似した過去の事例を基準に、修正をしながら試行を行い、結果と事例を事例ベースに記憶する。

・ベイジアン・ネットワーク
因果関係を確率により記述する。
複雑な因果関係の推論を有向非巡回グラフ構造により表しながら、個々の変数の関係を条件つき確率で表す確率推論のモデルである。

・説明の流れ

ビッグデータ蓄積後の利用の一例として、AIや機械学習の学習データとしてビッグデータを利用する、もしくは学習結果を得るためにビッグデータを蓄積するという利用方法が広く存在します。

当初のAIは人間が知識をコンピューターに入力する方式をとっていましたが、ベイズ推定などに代表されるように推論エンジンも登場し始めました。

機械学習

機械学習と深層学習

2005年以降は機械学習と深層学習による第3次AIブームとなっています。

機械学習

データから反復的に学習し相関を見つけ出すことです。また、その結果を元にして将来を予測することができます。
予測分析におけるモデル構築の自動化ができるため、データサイエンティストの人材不足を補うものとして期待されています。

深層学習

ニューラルネットワークの多層化、1990年代の視覚野の研究や、ブルーノ・オルスホーゼンによるスパース・コーディング理論を基にしたアルゴリズムが実装されたものを指します。
情報が第1層からより深くへ伝達されると、各層で学習が繰り返され、特徴量(問題の解決に必要な本質的な変数であったり、特定の概念を特徴づける変数)が計算されます。

・説明の流れ

第3次AIブーム以降、機械学習が登場しました。

大量のデータを学習し相関値を元にモデルを生成し、同じ傾向の新規データの動きを予測することが可能となりました。

この仕組みはレコメンドエンジンなどで広く使われており、人が介在しないAIの道を切り開きました。

深層学習（ディープラーニング）とは、機械学習の延長でパラメーター（説明変数）を3層同時に適用できるというものです。

画像や音声データの特徴量（特徴的な変数）、例えば画像の輝度の方向などを繰りかえし抽出する事で、画像の一致の判断などをかなりの精度で行う事が可能となります。

・ポイント（絶対に覚えてほしいこと、など）

ディープラーニングを行うためには大量の学習データ

と高速なGPUマシンなどが必要。

教師あり学習と教師なし学習

機械学習の手法として、教師あり学習と教師なし学習があります。

「教師あり学習」のイメージ
入力データに対応する出力を予測

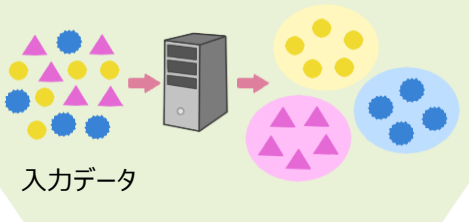
① 入出力のペアで訓練データを用意し、入出力の関係を学習させる

② 未知データを正しく判断できるようにする

→ 迷惑メール判定、株価予測などに使われている

「教師なし学習」のイメージ
入力データの特徴を学習して分類

① 入力データのみを与え、データの特徴をつかみ、分類を行うもの



→ クラスタリングなどを用いて、同じ集合の持つ特性などから結果を予測する

・説明の流れ

機械学習には回答データを必要とする教師あり学習と、必要としない教師なし学習があります。

教師なし学習はクラスタリングのように、集合の集まりがあるかどうかを模索し、集合が取れれば、どの集合に属するかで、新しいデータの特性を類推します。

機械学習アルゴリズム

機械学習アルゴリズムの分類

・Classification/Class probability estimation
既存のデータをクラスに分類し、未知の新規データの分類を予測する。
新規データに対しては、属する特定のクラスを示したり、各クラスに属する期待値も計算したりする。
　例 商品の買い替え時に、もともと使っていた製品から購入を予測する

・Regression（回帰分析）
既存データの関係を示す、数的な「関数」を推測し、定量的な予測をたてる。
例 広告費と売り上げの相関を見つける

・Similarity matching
新規データの、既存データへの類似性を予測する。

・説明の流れ
その他の機械学習アルゴリズムをご紹介します。

機械学習アルゴリズム

機械学習アルゴリズムの分類

・Clustering
具体的な指標を示さず、既存データを自然に分類できるグループを見つける。
分類後、各グループの特性を別のアルゴリズムで分析する。
　例 顧客をグループ化し、それぞれに別のサービスを提供する

・Co-occurrence grouping
既存データから、同時にまたは付帯的に起こる事象を推定する。
　例 Aを買った人は、Bも買っている

・Reinforcement learning(強化学習)
環境との相互作用を元に、データを収集し分析する。
例 囲碁プログラムがコンピュータ同士の対局から有効な手筋を発見する

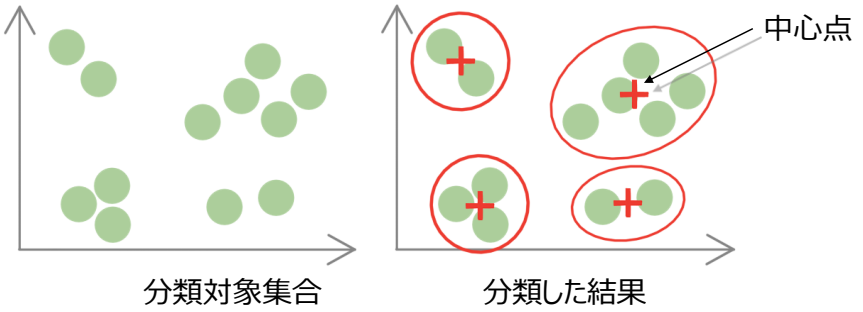
・説明の流れ
その他の機械学習アルゴリズムをご紹介します。

機械学習アルゴリズム

クラスタリング (clustering)

データの集合を、具体的な指標を使わず性質が近い部分集合にすることです。

統計解析や多変量解析の分野ではクラスター分析 (cluster analysis) とも呼ばれ、基本的なデータ解析手法としてデータマイニングでも頻繁に利用されています。



・説明の流れ

クラスタリングは、ランダムに決めた中心点で分類が可能かどうかを繰り返し実施することで、分類を試みる手法です。

分類した内容がどのような意味を持つかは、人が判断するか、別のアルゴリズムで決定します。

機械学習アルゴリズム

協調フィルタリング

商品の閲覧と購入のデータから、人同士の類似性や商品間の共起性をアソシエーション分析（相関分析）で解析し、対象者の行動履歴と関連づけることで、パーソナライズされた商品を提示する手法です。

協調フィルタリングは、商品スペックの関連性や商品閲覧の共起性だけではなく、購買データを基に人と人の類似性も重要視します。それにより、対象者に似ている集団が持つ特徴の中から、意外性のある商品を提示すること（セレンディピティ）もでき、コンバージョンレートのアップが期待できます。

・説明の流れ

協調フィルタリングは、行動履歴の相関値をとり、同じ行動をとるユーザーをモデルとして分類し、新しいユーザーに対して行動を予測するなど広い用途に使われています。

画像分析手法

画像データを解析し、顔認識や、画像の分類などを行う事が可能となります。

・SIFT
特徴点周りの輝度から最も輝度変化が大きいベクトルを調べ分類することにより、細かい特徴を把握することができる。

・Haar-like
形状やパーツが固定的なものに対し、平均的に示される特徴から画像を解析する。カメラの顔認識などに用いられる。

・畳み込み深層学習（CNN）
画像をピースごとに比較して、位置や形状の特徴が類似する箇所を検出する。
その上で、色、明度、エッジなどの多層において学習を繰り返す。
顔認証などに用いられる。

・説明の流れ

画像分析の手法にもいくつかの種類があります。

画像の輝度の方向などの特徴点から画像を分類するSIFTや、カメラなどに搭載されている顔のパーツの特徴から、顔を認識するHaar－like、大量の画像データを学習し、人物を見分ける畳み込み深層学習などがあります。

それぞれに得意、不得意があるため、適切なアルゴリズム選定が重要となります。

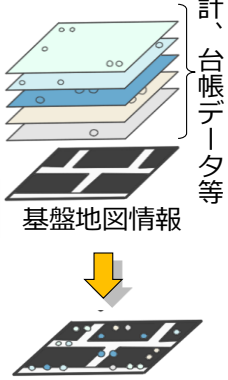
地図情報システムとの連動

GIS

地理空間情報を
データ毎に管理

地図レイヤーと
分析対象データを

位置関係の把握
データ検索、表示、
関連分析



GISの特徴

- 地図データとビッグデータを重ね合わせ、新たな発見を得ることができます。
- 地理的な位置情報との情報の関連性を視覚的に理解することができたり、図面上で情報の集約、整理をしたりできます。

GISの利用シーン

- 災害時の、正確で素早く効果的な行動決定などに利用されています。

・説明の流れ

ビッグデータを地図上にプロットし、知見を得る場合もあります。

地図情報はGISシステムから取得し、緯度経度などを持つビッグデータを地図上で表現すると、災害時の動態分析などが実現可能となります。

ビッグデータ講座

ビッグデータ概要

目次

第１章　ビッグデータ概要

目次

第２章 ビッグデータとセキュリティ

第１章

ビッグデータ概要

ビッグデータとは

ビッグデータとは

一般的なデータ管理・処理ソフトウエアで扱うことが困難なほど巨大で複雑なデータの集合のこと。

ビッグデータを取り巻く課題の範囲は、情報の収集、取捨選択、保管、検索、共有、転送、解析、可視化等多岐にわたる。これら課題を克服しビッグデータの傾向をつかむことで「ビジネスに使える発見、疾病予防、犯罪防止、リアルタイムの道路交通状況判断」に繋がる可能性がある.

Wikipediaより

つまり、ビッグデータとは、

・従来のシステムでは処理できないほど巨大なデータ

・定型を持たない複雑なデータ

・発見、予防といった新たな価値をもたらし得る、2次元的情報をもたらすデータ

であることが分かります。

ビッグデータの量

巨大なデータとはどれくらい？

ビッグデータの例を見てみましょう。

データ量を表す単位は、以下の順に1024倍となります。

キロ(KB)<メガ(MB)<ギガ(GB)<テラ(TB)<ペタ(PB)<エクサ(EB)<ゼタ(ZB)

全世界で生成・消費されるデジタルデータの総量

IDC（International Data Corporation）の発表： 59ゼタバイトを超える

出典：https://www.idc.com/getdoc.jsp?containerId=prUS46286020

ビッグデータの質

蓄積されているデータはどのようなデータでしょうか？空欄を埋めてみましょう。

ビッグデータ

ブログデータ	車の位置情報	Web 操作ログ	気象情報
ドライブレコーダー	コールセンター音声	水質データ	防犯カメラ映像
人口統計情報	電力データ	SNS 写真	メール・チャット
ネット検索履歴	ツイートデータ	自販機前の動作映像	

従来型

販売 POS データ	EC 売上データ	販売・生産実績
EXCEL のデータ	基幹データベース	会計システムデータ

ビッグデータの種類

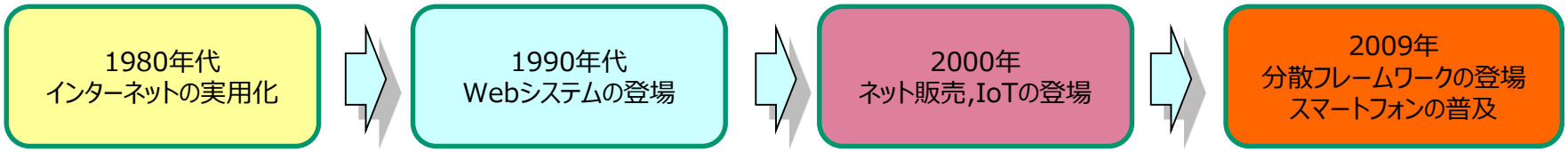
ビッグデータの分類	構造化データ	準構造化データ	非構造化データ
分類の意味	データベースに格納される行列の二次元テーブルで表現されるデータ。 それほど増加しない見込み。 例・顧客テーブルデータ ・受注テーブルデータ ・CSV データ ・Excel データ	完全な構造定義を持たないデータ。 例・ログデータ ・センサーデータ ・SNS に書き込まれたデータ	データ部に構造定義を全く持たないデータ。準構造化データと合わせてデータ総量の 80%を占め、5 年で 800%の増加傾向。 例・文書 ・音声 ・動画 ・画像
前のページの例を当てはめると …	販売 POS データ 販売・生産実績	ツイートデータ Web 操作ログ	防犯カメラ映像 コールセンター音声

ビッグデータが持つ特性

ビッグデータの３V

ビッグデータの5V（3V + Value + Veracity）… 最近提唱されている。

ビッグデータ登場の背景



IoT、SNS、Webシステムなどから生成されるデータの爆発的増加

分散処理技術・フレームワークの充実

NoSQLデータベース、機械学習などの技術的成熟

AWS、Azureなどクラウドサービスの充実

センサーの低価格化・スマートフォンの普及

ビッグデータの所在

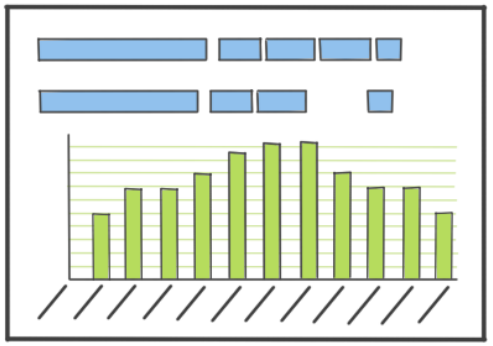
社内（ローカル）	**自社基幹システム** 販売実績や、生産実績データ、会計データなど	**Web、SNSサービス等** ECサイト、社内ポータルサイト、アプリ操作ログ	
社外	**顧客・ユーザー** スマホや家電、メーター上のデータ	**グループ企業** 企業間で共有される情報	**取引企業** サプライチェーンの情報
一般	**政府・自治体等** 統計データや地図情報など公開されている情報	**提携企業** SNSデータ、位置情報空間統計、交通機関乗降情報等	**データ提供事業者** 地図、統計情報など目的に合わせて整備したデータ

オープンデータとは

特徴	■ 誰でも入手可能で、自由に利用・再配布できる状態で存在する ■ 特許・著作権に制限がない ■ コンピューターから利用できる状態となっている
公開主体	■ 政府 ■ 地方自治体 ■ 研究機関・大学 ■ 民間企業
具体例	■ 国勢調査データ（政府統計の総合窓口「e-Stat」） ■ 公共施設やAEDの位置データ ■ 気象データ ■ 有志により作られた地図データ（OpenStreetMap など） 「行政と市民によるオープンデータ共創支援プラットフォーム（LinkData）

オープンデータ×ビッグデータ活用例

・過去のTwitterなどのSNS上の書き込み ＋ 販売データ
→ 相関を調べ、売上の増減や欠品可能性を予測する。



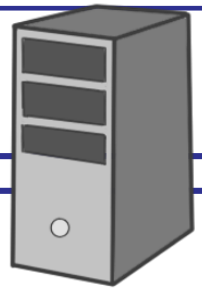
・自社の販売データ ＋ 気象データ
→ 気象変化と売上推移の相関を見出し、予測を行う。

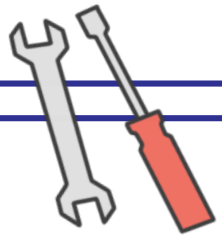
・医療施設の位置データ ＋ 患者の郵便番号のデータ
→ 来院マップを作成し、診療費ごとの外来状況を分析することで、地域医療に関して重点的な連携、促進を図る。

・国勢調査などの人口統計情報 ＋ 将来の人口推計
＋ ターゲット層の世帯が多数存在する地域の売上相関
→ 重点的に販売を行う地域を模索する。

ビッグデータが活用される分野

マーケティング	Webやカメラから顧客の行動を分析 →レコメンドシステム
製品開発	センサーからのフィードバック・顧客の声 →開発の指針
コンプライアンス	文書の全文検索とトピック抽出 →不適切な行動を察知
セキュリティ	サイバー攻撃のパターン検知 →予防策
メンテナンス	センサーデータのパターン検知 →故障予測
社会インフラ	気象などのデータ、地図情報 → 災害の予測、避難経路





ビッグデータが活用される業界

	運輸	金融	医療・健康	製造
利活用事例	1. 航空機チケットの割引サービスの改善：Web販売サイトでの購入傾向を分析 2. 渋滞予測：スマホGPSの情報を分析 3. トラックの最適な輸送ルート：過去の情報から算出	1. 金融商品の提案・開発：数千万件の顧客情報から、決済や資産運用の動きを分析 2. 保険サービスの開発：車の走行距離の情報を元に保険料を定めるサービスを開始	1. インフルエンザ対策：SNS上のコメントや検索キーワードから、広がりを検知 2. メディケア（アメリカの公的保険）のデータ公開：公的機関が分析して公開	1. 品質改善のための最善策の策定：生産工程に関する多数の情報を収集、分析 2. 製品開発：建機にセンサーを設置し故障時の稼働環境を分析してフィードバック。
効果	販売促進 人流動態分析	新サービス開発	兆候把握・ 情報提供	品質向上

第2章
ビッグデータとセキュリティ

ビッグデータとセキュリティ（個人情報保護）

個人情報とは

個人情報の保護に関する法律 第二条 第１項 第一号において、次のように定義されています。

「生存する個人に関する情報であって、」「当該情報に含まれる氏名、生年月日その他の記述等」「により特定の個人を識別することができるもの（他の情報と容易に照合することができ、それにより特定の個人を識別することができることとなるものを含む。）」

個人情報保護法

個人情報保護法

・成立：2003年（平成15年）5月23日
・施行：即日（但し、一般企業に直接関わり罰則を含む第4～6章を除く）
・全面施行：2005年（平成17年）4月1日 … 成立の2年後

個人情報取扱事業者

・個人情報を個人情報データベース等として所持し事業に用いている事業者のことをいう。

・個人情報保護法および同施行令により、取扱件数に関わらず、個人情報取扱事業者とされるようになった。

・主務大臣への報告や、それに伴う改善措置に従うなどの適切な対処を行わなかった個人情報取扱事業者に対しては、刑事罰が科される。

個人情報保護法改正

個人情報保護法　2015年の改正内容

・これまで対象外だった、5,000人分以下の個人情報を取り扱う小規模な事業者に対しても、改正法が適用されるようになった。

・個人情報を取得する場合、予め本人に利用目的を明示することが必要となった。

・個人情報を他企業などの第三者に提供する場合、予め本人から同意を得ることが必要となった。

・オプトアウトには、個人情報保護委員会への届出が必須となった。
更に、第三者提供の事実、その対象項目、提供方法、望まない場合の停止方法などを、全て予め本人に示さなければならなくなった。
※オプトアウト：本人の同意を得ないで個人情報を提供できる特例のこと。

・「人種」、「信条」、「病歴」といった「要配慮個人情報」は、オプトアウトでは提供できないこととされた。

個人情報保護の情勢

・1980年　プライバシー保護と個人データの国際流通についてのガイドラインに関するOECD理事会勧告（OECDプライバシーガイドライン）
（OECDの34加盟国）

①収集制限　②データ内容　③目的明確化　④利用制限
⑤安全保護措置　⑥公開　⑦個人参加　⑧責任
の８原則からなる。

・1995年　EUデータ保護指令　（EUの28構成国）

EUおよび英国においては、十分なデータ保護レベルが確保されて
いない第三国への個人データの移動を禁止する。

・2003年　個人情報保護法　（日本）

個人情報を扱う事業者に対し、個人情報の適切な取り扱いを求める。

・2012年　消費者プライバシー権利章典　（アメリカ）

①個人によるコントロール　②透明性　③背景情報の尊重
④セキュリティ　⑤アクセスと正確性　⑥適切な範囲の収集　⑦説明責任
の7つの権利を定める。

個人情報保護法の情勢

・EU一般データ保護規定（GDPR）が可決（2016年4月 EU）
　データポータビリティ権が提唱される。
　→ 域外適応につき、日本の事業者に影響が出る。

・EU - USプライバシーシールドに米国と欧州委員会が合意（2016年2月 米国）
　スノーデン事件を受けて無効化されていたセーフハーバーの後継。
　商務省とFTCに強い権限が与えられ、企業に対して自主規制を求める機運が高まった。

・APEC 越境プライバシールールシステム（CBPRs）への参加（アジア）
　日本に関しては、2014年にJIPDECがCBPR認証機関に認定された。
　→ 2016年6月1日から申請受付開始。

・個人情報保護委員会が発足（2016年1月 日本）
　個人情報保護法の改正を受け、政府の第三者機関として設立した。

・一般財団法人情報法制研究所（JILIS）が設立（2016年5月 日本）

オプトインとオプトアウト

オプトイン（事前承認）

明示的な同意が無い限り、個人情報やプライバシー情報は収集されないような仕組みのことを言います。

例・ショッピングサイトからのセール情報に関するメールの送付を許可する。
・個人情報の収集・利用を含むサービスの利用規約に同意する。

オプトアウト（事後承諾）

オプトインとは反対に、明示的に拒否していない限りは同意したものとみなし、明示的な拒否があった場合に個人情報やプライバシー情報の利用が停止されるような仕組みのことを言います。

例・Webサイトにおけるクッキーを用いた行動追跡
・ショッピングサイトにおける購買履歴の削除

匿名加工処理の手法

以下のそれぞれの手法を組み合わせることで、より強固な匿名化が実現されます。

技法大部類	No.	技法例	概要
摂動法	1	K-匿名化	同じグループ内に、同じ属性のユーザが「K人以上いる」状態を作り出す。
	2	L-多様性	漏えいさせたくない属性が同じグループ内で「L種類以上ある」状態を作り出す。
	3	T-近接性	マイナー属性を持つグループが生まれるなど、属性値の分布に偏りが出てしまう場合に、グループの分割や一般化を行う。
	4	差分プライバシー	2006年に提案された新しい手法。元のデータベースにノイズを足した別のデータベースを用意し、守りたいレコードを特定しづらくする。
暗号法	5	質問監査	データベースへのアクセス者に質問を投げかけ、答えられれば、アクセスに対する回答を返す。
	6	秘密計算	関係者全員が、自社データを他人が読めないように変換し、秘密計算のシステムへ投入する。そのシステムの管理者が、秘密計算の結果を求め、関係者に回答する。
	7	準同型性公開鍵暗号を用いた暗号プロトコル	遺伝子データなど、加工してしまうと、そもそも分析できなくなるデータを処理するときに活用。検索者の検索クエリ、データベース、その回答それぞれを暗号化する。分析者が元データにふれずとも、望む解析結果が得られる。

出典：中川裕志『プライバシー保護入門：法制度と数理的基礎』（2015年）

識別子の削除（仮名化）

個人の識別・特定に直結するカラムを削除して、仮名化を行います。

No.	ZIPコード	年齢	職業	病状
1	13068	28	ダンサー	心臓病
2	13068	29	技術者	心臓病
3	13053	21	法律家	感染症
4	13053	23	技術者	感染症
5	14853	31	技術者	風邪
6	14853	37	作家	風邪
7	14850	36	法律家	がん
8	14850	35	技術者	がん

準識別子　　漏えいさせたくない属性

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

K-匿名化したテーブル

次に、再特定・識別につながる「職業」を秘匿した上で、「年齢」、「病状」の列に「同じ値が少なくとも2つ以上は存在する状態」のテーブルを作ります。

No.	ZIPコード	年齢	職業	病状
1	13068	28-29	*	心臓病
2	13068	28-29	*	心臓病
3	13053	21-23	*	感染症
4	13053	21-23	*	感染症
5	14853	31-37	*	風邪
6	14853	31-37	*	風邪
7	14850	35-36	*	がん
8	14850	35-36	*	がん

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

L-多様化したテーブル

「ZIPコード」と「年齢」を曖昧にして、「どのレコードを取り出しても、2種類の「病状」が存在する状態」になるようにします。

No.	ZIPコード	年齢	職業	病状
1	130**	21-29	*	心臓病
2	130**	21-29	*	心臓病
3	130**	21-29	*	感染症
4	130**	21-29	*	感染症
5	148**	31-37	*	風邪
6	148**	31-37	*	風邪
7	148**	31-37	*	がん
8	148**	31-37	*	がん

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

ビッグデータ

ビッグデータ活用を支える技術

目次

第１章　ビッグデータを支える技術　クラウド、IoT

目次

第２章　ビッグデータを支える技術　データ収集・加工

目次

第３章　ビッグデータコア技術　NoSQL

目次

第４章　ビッグデータコア技術　分散処理

目次

第５章　ビッグデータにおけるデータ解析

目次

第６章　ビッグデータとAI、機械学習

第１章

ビッグデータを支える技術

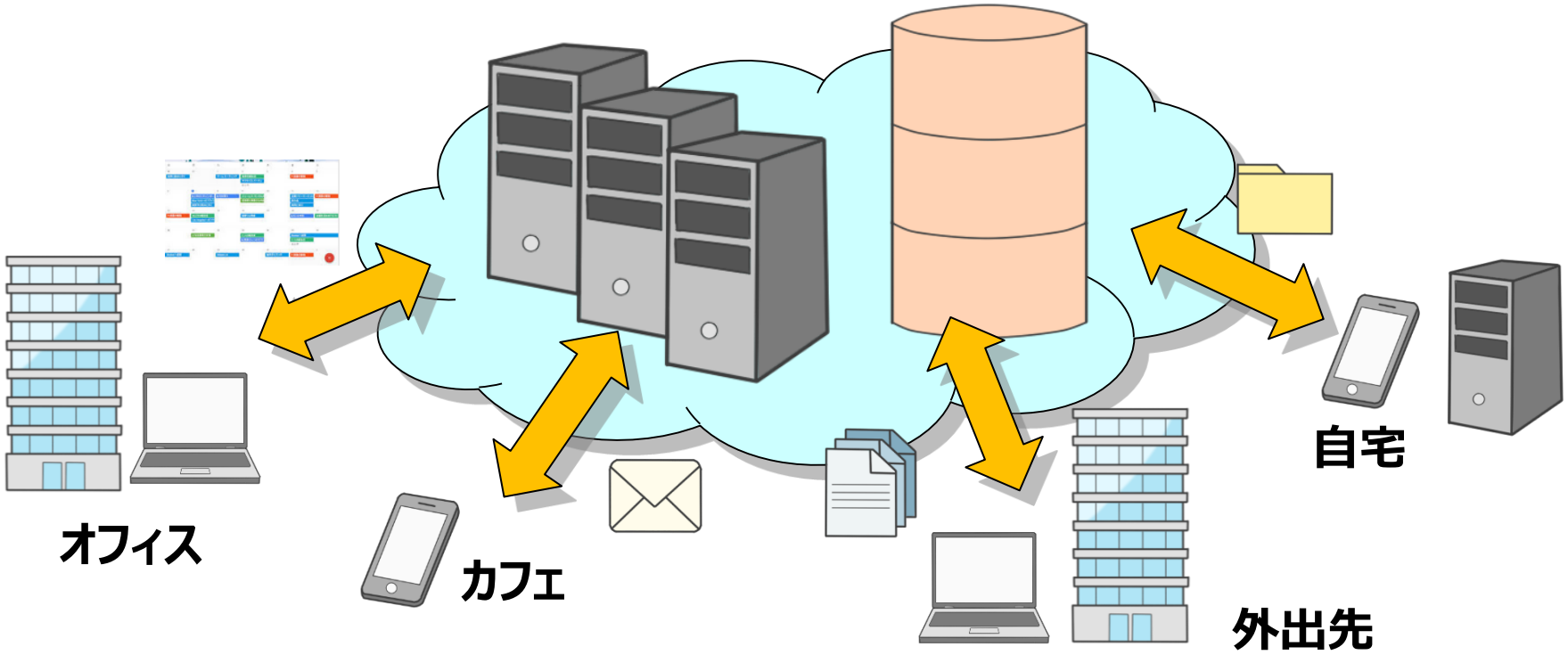
クラウド、IoT

ビッグデータ周辺技術　クラウド

クラウドコンピューティングとは

メールやグループウェア、その他様々なサービスをインターネット上で提供し、インターネット上にデータまでをも保存するようなサービス形態のことをいいます。

アプリケーションの実行場所が、雲（クラウド）のようにどこにあるのかが分からないモヤモヤとした場所にあることから、このように呼ばれています。



クラウドサービスの種類

クラウドサービスの種類

・SaaS（Software as a Service）

インターネットを経由してソフトウェアパッケージを提供するサービスのこと。

アプリをPCにダウンロードしなくても、Webなどのブラウザ上で利用することができる。

例・メール　・カレンダー　・チャット

・PaaS（Platform as a Service）

インターネットを経由してアプリの開発・運用環境全体を提供するサービスのこと。システム管理者、開発者向けである。

・HaaS / IaaS（Hardware as a Service / Infrastructure as a Service）

インターネットを経由してハードウェアや回線などのインフラを提供するサービスのこと。ユーザーはハードウェア資産を所有することなく、仮想サーバーやストレージ（外部記憶装置）を利用することができる。

クラウドサービスのメリット・デメリット

メリット

- ・サーバー・ソフトウェアを購入する必要がない
- ・システム構築期間を短縮できる
- ・効率的なIT投資やリソース配分を実現できる
- ・メンテナンスが不要である
- ・IT部門の負担が軽減される

デメリット

- ・カスタマイズが基本的にできない、もしくは難しい
- ・不特定多数が利用するため、安定して稼働できないリスクがある
- ・セキュリティー面のリスクがある
- ・利用するデータ量、時間によっては費用が増加するリスクがある

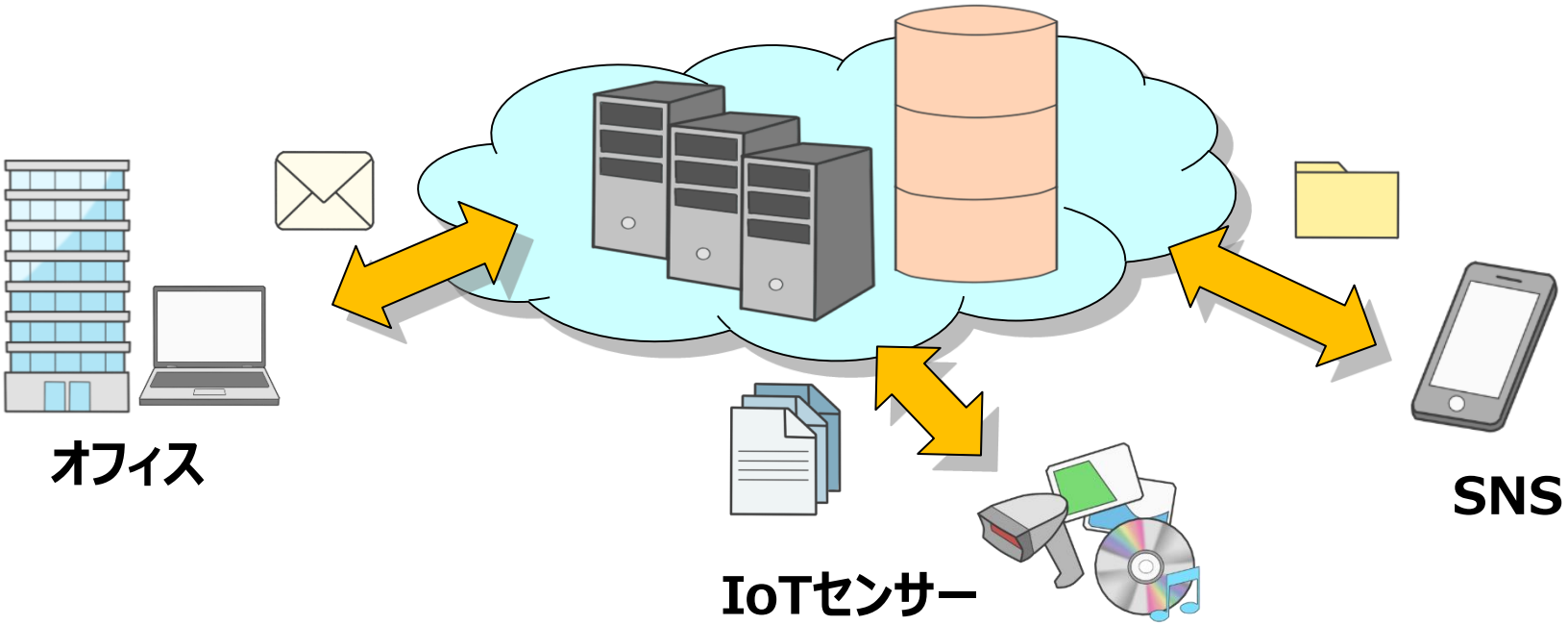
クラウドコンピューティングとビッグデータ

クラウドサービスに蓄積されるビッグデータ

クラウドサービスは、ビッグデータの蓄積場所に有力な選択肢です。

実際、以下はビッグデータがクラウドに保存されています。

- GREE、mixi、Facebook、Twitterなどに代表されるSNSのユーザーデータ
- IoTのようにセンサーなどで発生したデータ



ビッグデータ周辺技術　IoT

IoTとは

IoT ＝ Internet of Things　「モノのインターネット」

ここで、「モノ」とは、ネットワークに繋がるあらゆる物のことです。

従来はインターネットとは関係のなかったもの、例えば、

・眼鏡　・服　・時計　・冷蔵庫　・電力メーター　・自動車
・太陽光パネル　・家　・スマートフォン

もすべて「モノ」です。

IoTとは、モノがネットワークに繋げられることによって、モノとインターネットが相互に情報交換をできるようになった状態のことをいいます。

IoTとは

・Internet of Thingsという用語は1999年、ケビン・アシュトン（イギリス）によって初めて提唱されました。

・当初はRFIDによる商品管理システムをインターネットに例えたものでした。

→ 徐々にスマートフォンやクラウドコンピューティングが普及。

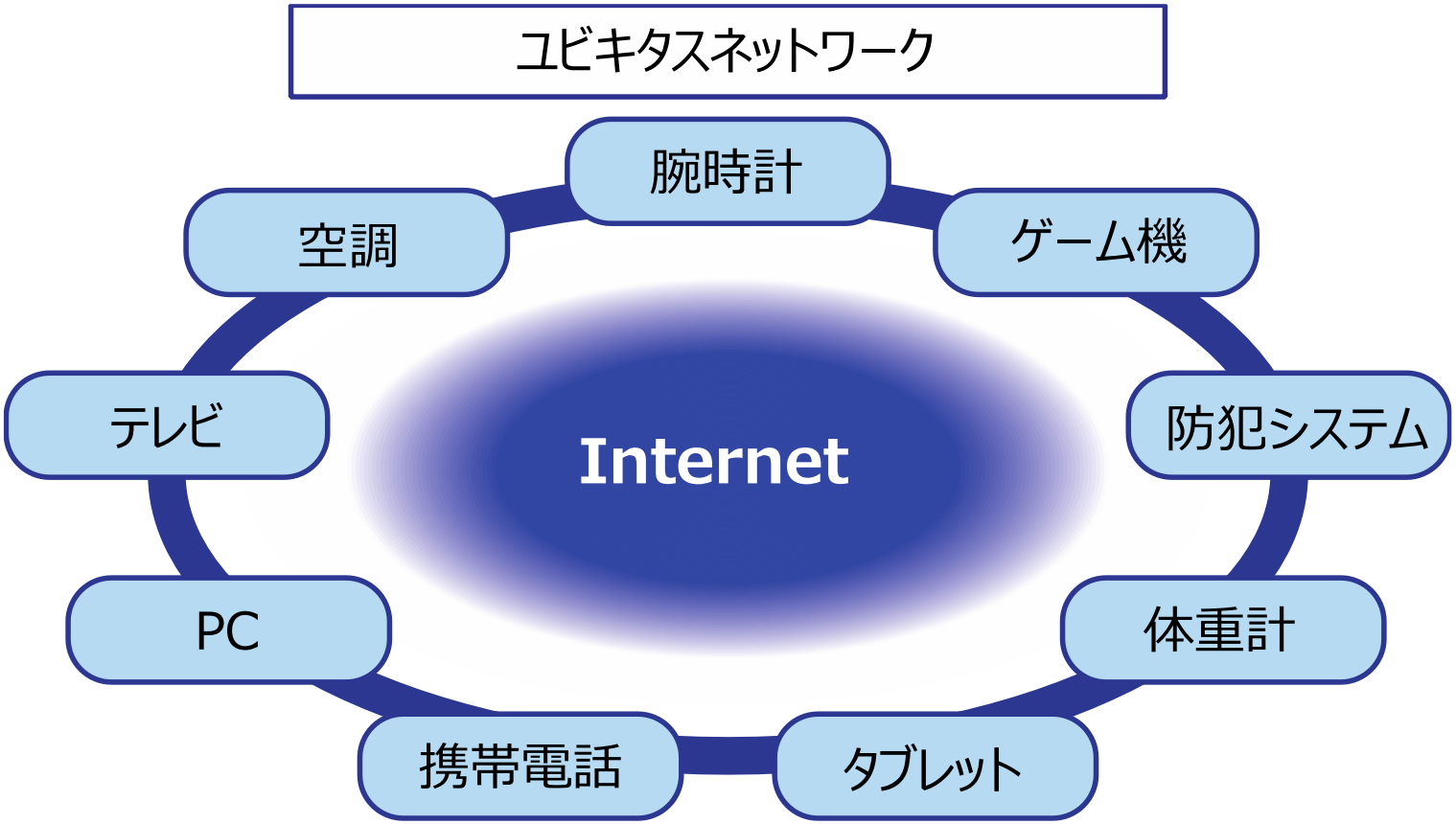
→ IoTはモノ自体がインターネットを形作るという環境全体のことを表す概念として捉えられるようになる。

・IDC（ICT市場調査会社）による定義
「IP接続による通信を人の介在なしにローカルまたはグローバルに行うことができる識別可能なエッジデバイスから成るネットワークのネットワーク」

IoTとユビキタス社会

ユビキタスネットワークとは

いつでもどこでもインターネットを利用できるという概念のことをいいます。
1988年ゼロックス社パロアルト研究所のマーク・ワイザーにより提唱されました。



IoTとユビキタス社会

IoTとユビキタスの違い

・IoT：モノとモノが相互に制御し合っている状態を表す。
　　…「モノ」を中心とした概念

・ユビキタス：ユーザーが時間や場所にとらわれずインターネットに繋がって様々なサービスを受けられる状態を表す。
　　…ユーザーという「人」を中心とした概念

参考

ユビキタス (ubiquitous) は、遍在（いつでもどこでも存在すること）をあらわす言葉。

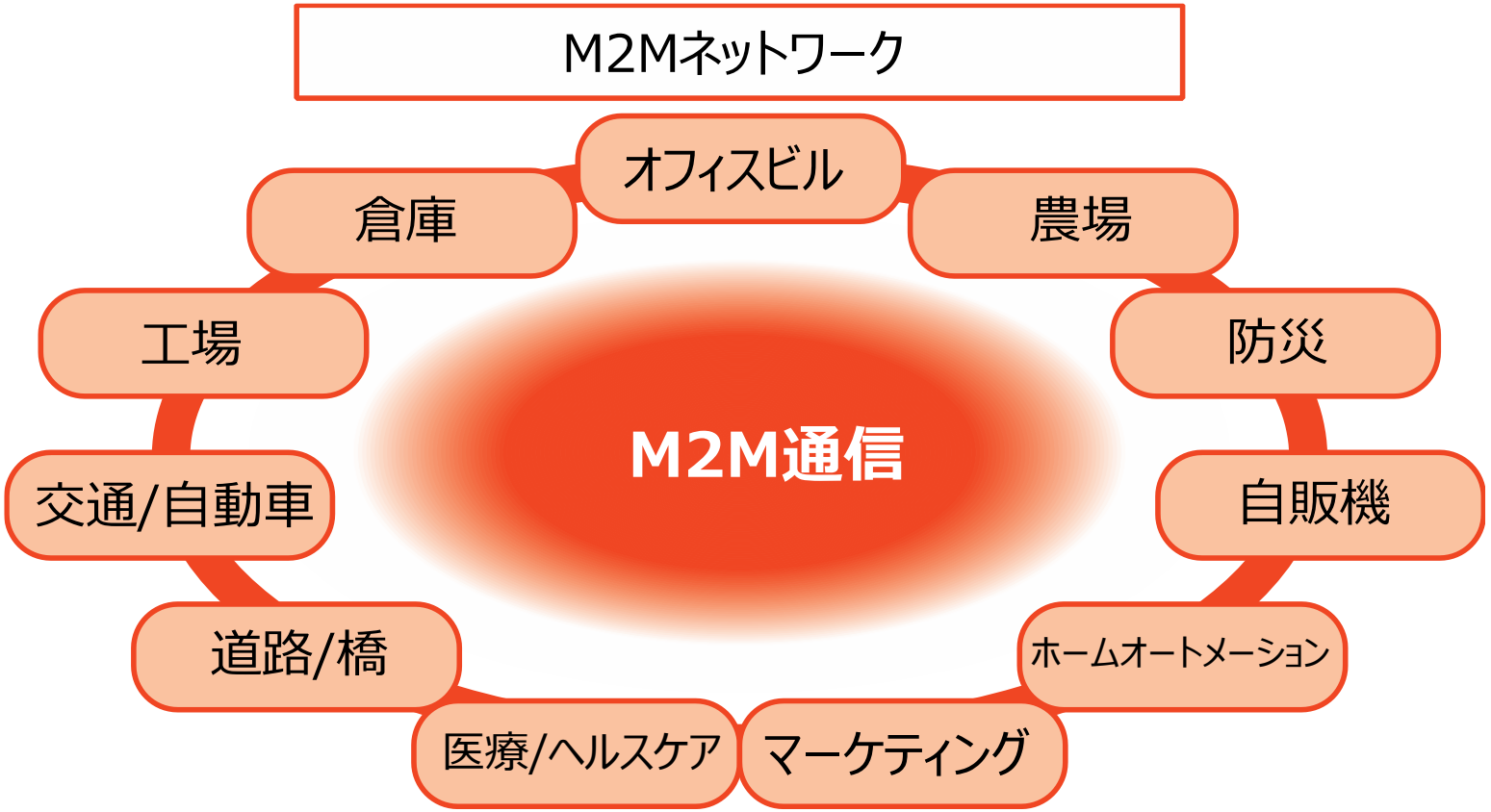
パロアルト研究所のマーク・ワイザーが、1991年の論文『The Computer for the 21st Century』にて、コンピュータやネットワークなどの遍在を表す意味合いで用いた。以来、ユビキタスコンピューティングやユビキタスネットワーク、さらにはそれらが当たり前になった社会を指す「ユビキタス社会」の意味で用いられるようになった。

Wikipedia

IoTとM2M

M2M = Machine to Machine

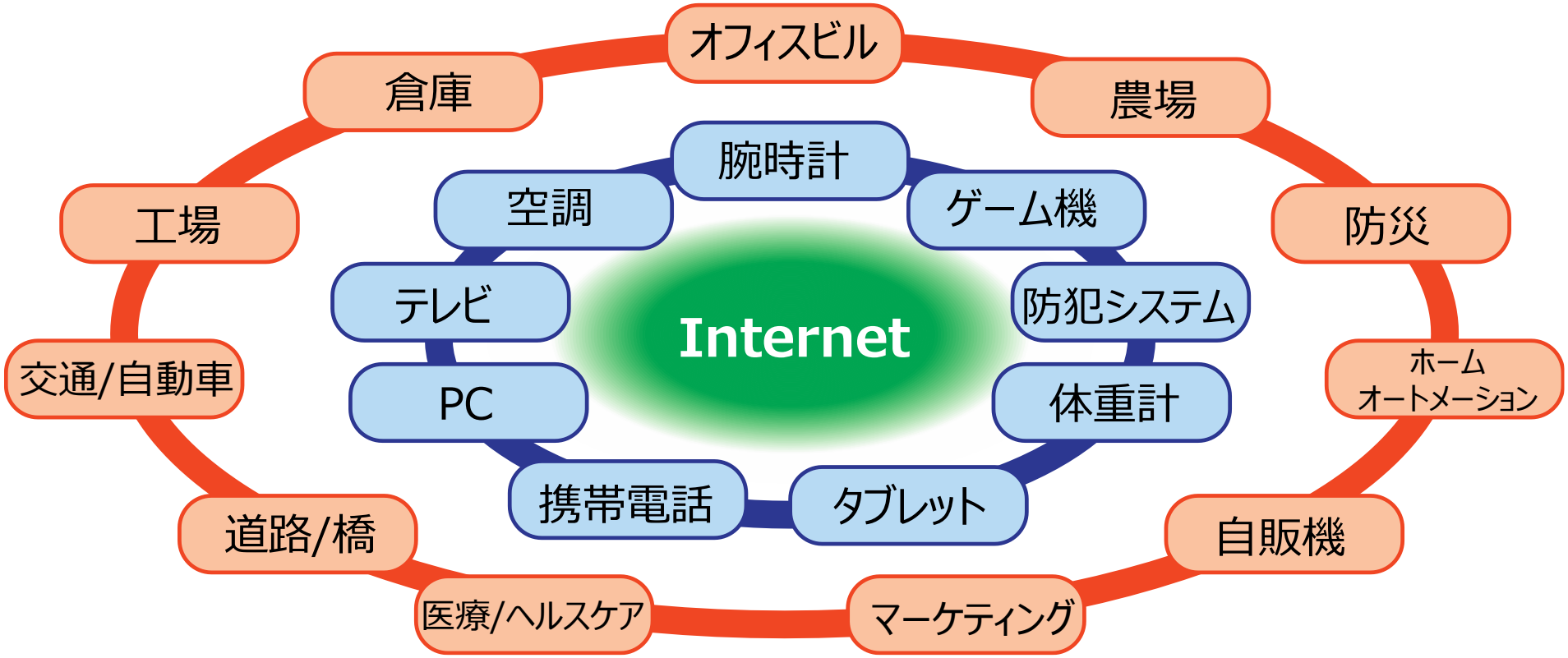
機械同士が、人を介在しないで情報をやり取りするシステムのことをいいます。



IoTとM2M

IoTはユビキタスとM2Mを包括する概念といえます

IoT社会
Machine to Machine、Human to Machine、Human to Human



IoTを構成する技術要素

・センサー：物理的な現象を検知し、電気信号として出力する装置。

・デバイス：センサーが組み込まれることによって、ネットワークに接続された装置やモノ。

例　スマホ、時計、メガネ

・ネットワーク：デバイスをIoTサービスに繋ぐ、あるいはデバイス同士を繋ぐことでデータを共有、処理するシステム。

・IoTサービス：　①デバイスとのデータの送受信 … IoT
②データの処理と保存 … ビッグデータの技術的守備範囲
を行うサービス。

・データ分析：蓄積したデータについて統計分析や機械学習を行う。

→最適な判断や行動方針を導き出す。

IoTを構成する技術要素　センサー

センサー

物理的な現象を検知し、電気信号として出力する装置のことをいいます。
多くの場合、一つのデバイスに対して複数のセンサーが埋め込まれています。

・画像センサー：光を捉えて処理することで、画像や動画を撮影する。
　　赤外線を検知して画像処理するものもある。
・光センサー：光の強度を測定する。
・温度センサー：温度を測定する。
・湿度センサー：湿度を測定する。
・振動/速度/加速度センサー：機器の振動や速度、加速度を測定する。
・地磁気センサー：地磁気を検出することで、方角を計測する。
・ジャイロセンサー：デバイスの傾きを検知する。
・音声マイク：機器が発する音や、人の声などの音声を収集する。

IoTを構成する技術要素　センサー

センサーの代表的なデータフォーマットとしては、
・XML　・JSON　・MessagePack
があります。

XML
```
<xml>
  <info>
    <id>12996</id>
    <name>RoomSensor</name>
    <date>20170123112255</date>
  </info>
  <data>
    <temperature>27.8</temperature>
    <humid>72</humid>
  </data>
</xml>
```

人が読んで分かりやすい
データ量が多い

JSON
```
{"info":{"id":123,
          "name":"RoomSensor",
          "date":20170123112255
         },
 "data"{"temperature": 27.8,
          "humid":72
         }
}
```

データ量が少ない

いずれも各言語のライブラリが充実していますが、文字データであることから、パース（解析）をしないとプログラムで利用できません。
MessagePackは、バイナリデータをそのまま扱いたい場合、有利です。

IoTを構成する技術要素　センサー

MessagePack

・センサーの代表的なデータフォーマットの一つ。
・JSONと似た形式だが、値はバイナリのままである。
・軽量でプログラム間処理に向いている。

MessagePackの特徴

・シリアライズ（データの直列化）、デシリアライズ：非常に高速
・シリアライズされたデータのサイズ：小さい
・フォーマット定義：不要
・ストリーム処理：可能

JSONとMessagePackの比較

JSON　{“a”:null,”b”:10,”c”:[20],”d”:”30"}　… 35byte

→ MessagePack に変換すると…

84 a2 61 c0 a2 62 0a a2 63 91 14 a2 64 a2 33 30　… 16byte

IoTを構成する技術要素　デバイス

デバイス

センサーが組み込まれることによって、ネットワークに接続された装置、モノのことをいいます。例えば、スマホ、時計、メガネはいずれもデバイスです。

デバイスの２つの機能

・センシング：センサーを利用して、デバイス自身や周りの環境の状態を収集し、IoTシステムに通知すること。

例・画像センサーによる人の有無の検知
・スマホの位置情報や加速度の計測

・フィードバック：システムからの通知を受け、指示や動作をもとのシステムに返すこと。次のような方法がある。

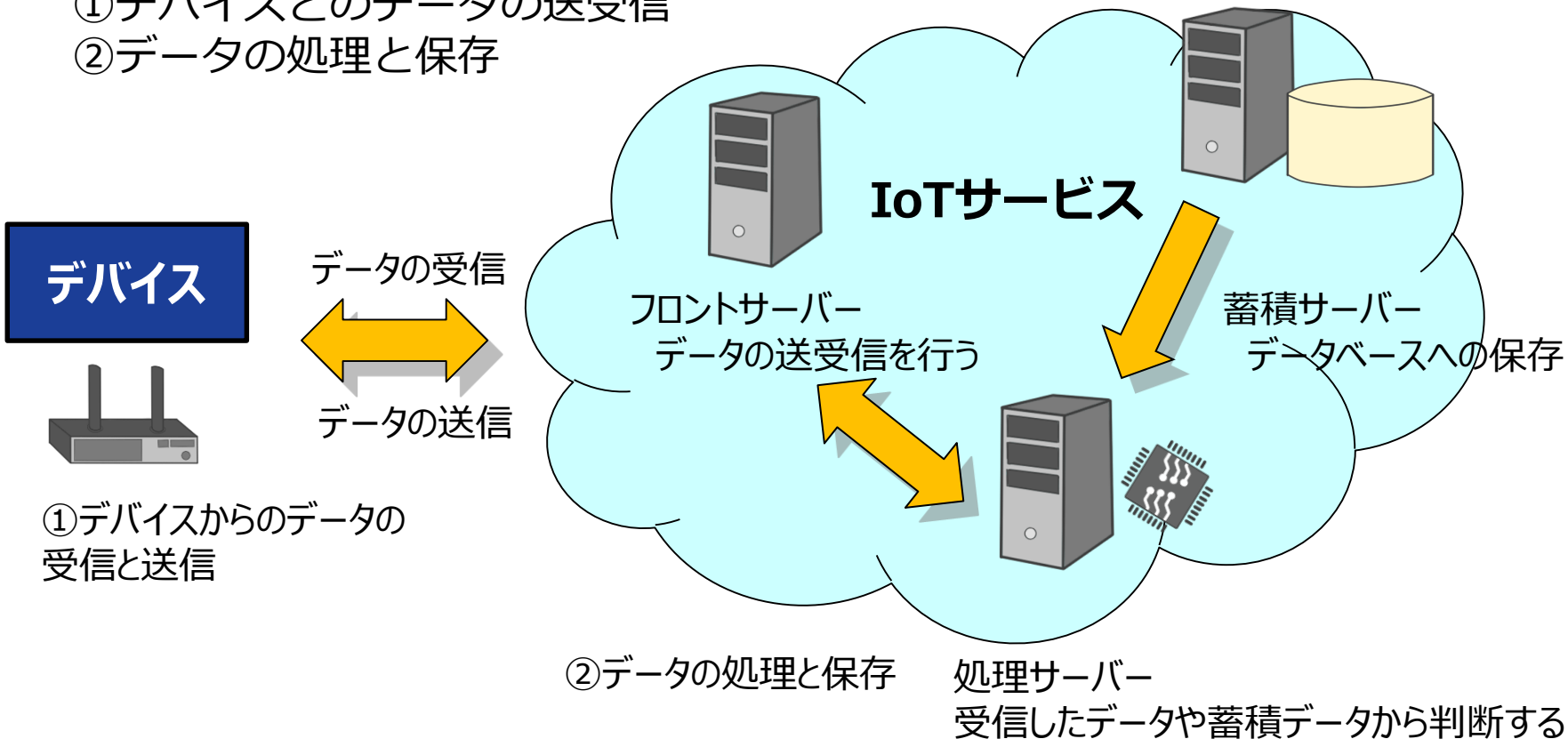
・可視化：センシング結果表示、デバイスの管理、画面表示
・通知：システムが判断した結果を画面に表示
・制御：デバイス自身や環境の状態そのものを変更

IoTを構成する技術要素　IoTサービス

IoTサービス

以下を行うサービスのことを指します。

①デバイスとのデータの送受信
②データの処理と保存

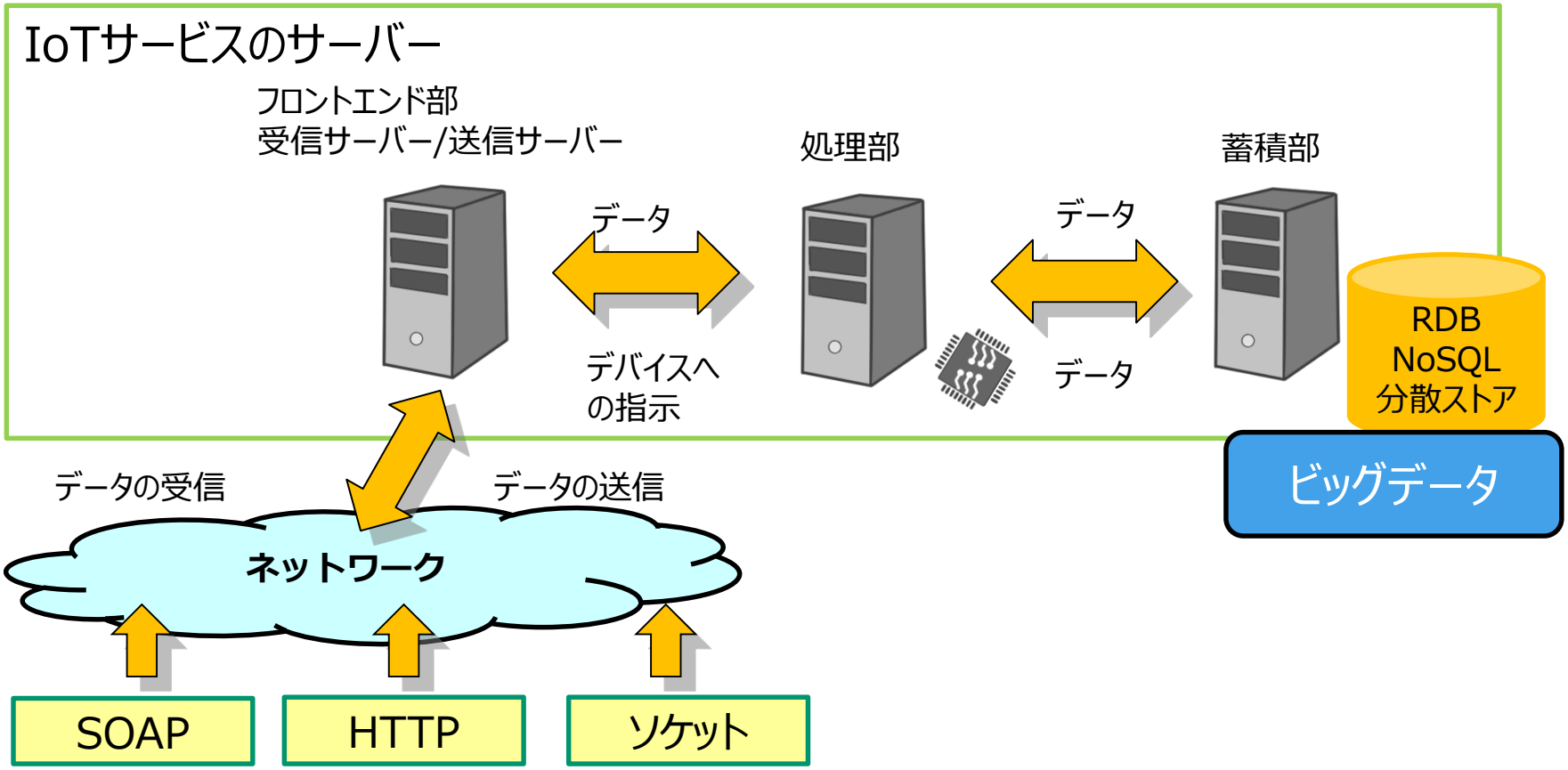


IoTを構成する技術要素　IoTサービス

サーバー構成

IoTサービスの役割は、フロントエンド、処理、蓄積の大きく３つに分けられます。

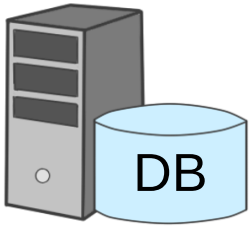
蓄積されるデータは膨大な量となります。



第２章
ビッグデータを支える技術

データ収集・加工

ビッグデータの解析までの流れ



社内の実績データ

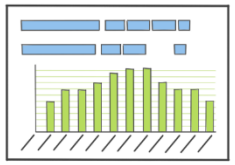
SNSデータ
社外の取引データ
Webサイトログ

音声・画像・
センサーデータ

オープンデータ・
統計データ・
科学データ

ビッグデータ
蓄積基盤

BIツール



統計解析
ツール

機械学習・
AI

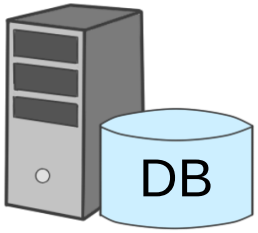
収集

蓄積

解析

データ蓄積までの流れ

以下のようにして、ビッグデータは分析可能な状態に処理されます。



社内の実績データ

SNSデータ
社外の取引データ
Webサイトログ

音声・画像・
センサーデータ

オープンデータ・
統計データ・
科学データ

データ転送システム

クレンジング

- 名寄せ
- 型の変換
- 文字・日付の分割・結合
- 置換・マスター変換
- 数値演算

ETLツール

- RDB
- DWH カラムナDB
- NoSQL
- Hadoop

ストリーミング処理

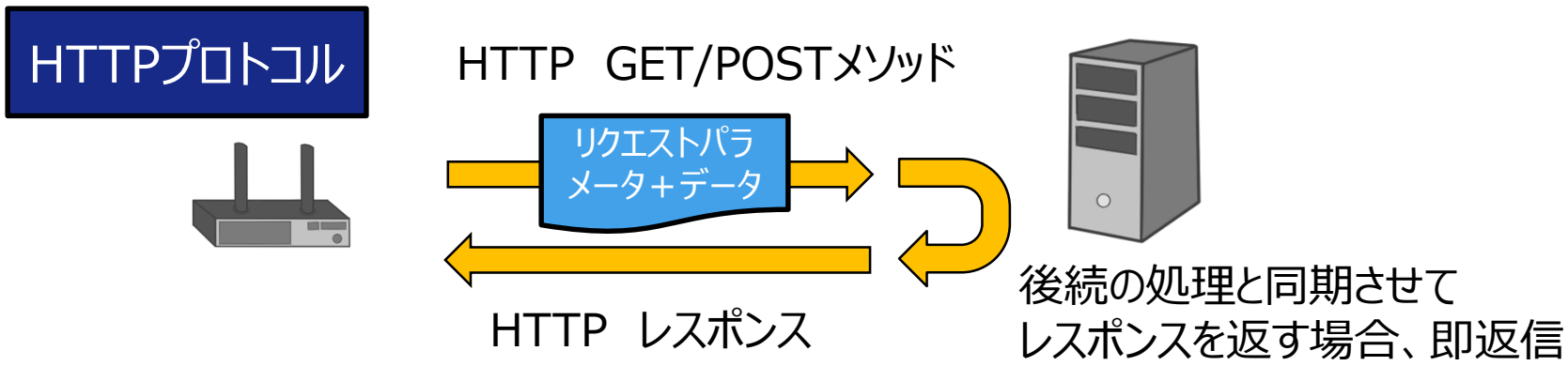
データ蓄積手法

データ収集例

データ収集と通信方法

IoTの受信サーバの通信方法は3種類に分類できます。

- HTTPプロトコル
  通常のWebシステムと同様、HTTPプロトコルを利用したWeb APIを利用してデバイスからアクセスを行う。
- WebSocket
  音声や動画のリアルタイム通信を行う。
- MQTT
  送受信を媒介する第三者の存在により、柔軟な通信を可能にするメッセージ・キュー方式を利用する。



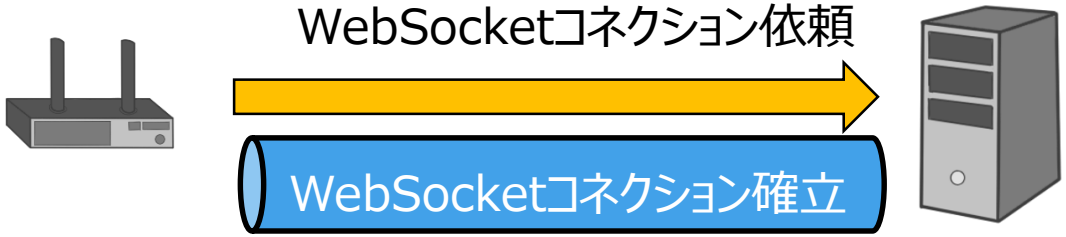
データ収集例

WebSocket

WebブラウザとWebサーバー間でデータを双方向かつ連続的に送受信するための通信プロトコルです。
IoTでWebSocketを用いることで、インターネット上でソケット通信が可能になります。

HTTPプロトコルを用いると送信毎に接続の依頼が必要ですが、WebSocket通信を行うと、接続を継続したままにすることが可能となります。

HTTPプロトコルの場合 WebSocketコネクション確立



送信毎にクライアントから
コネクション依頼の必要がある

WebSocket通信

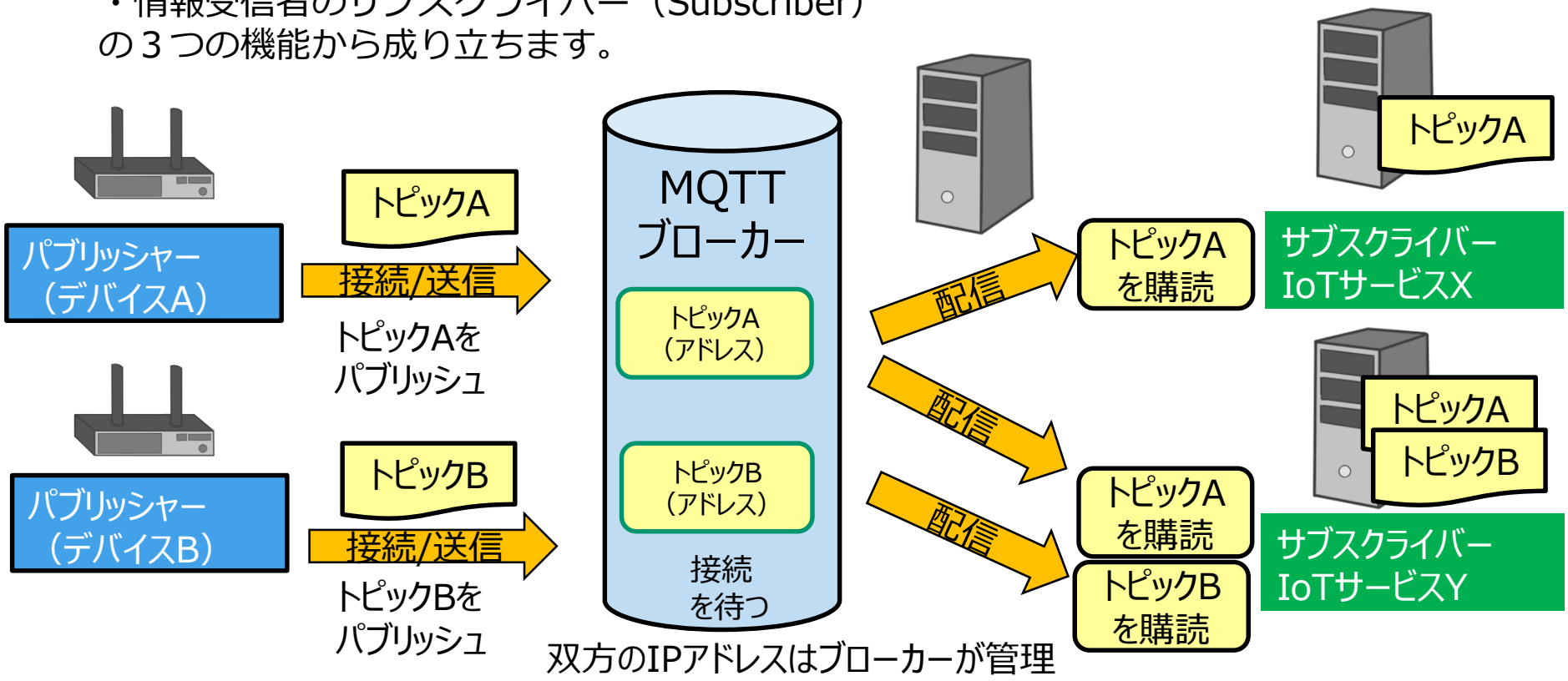
クライアントから一度
コネクションを確立すれば
継続して利用可能

データ収集例

MQTT（MQ Telemetry Transport）

IBM社により提唱され、現在はオープンソースとなっているプロトコルです。

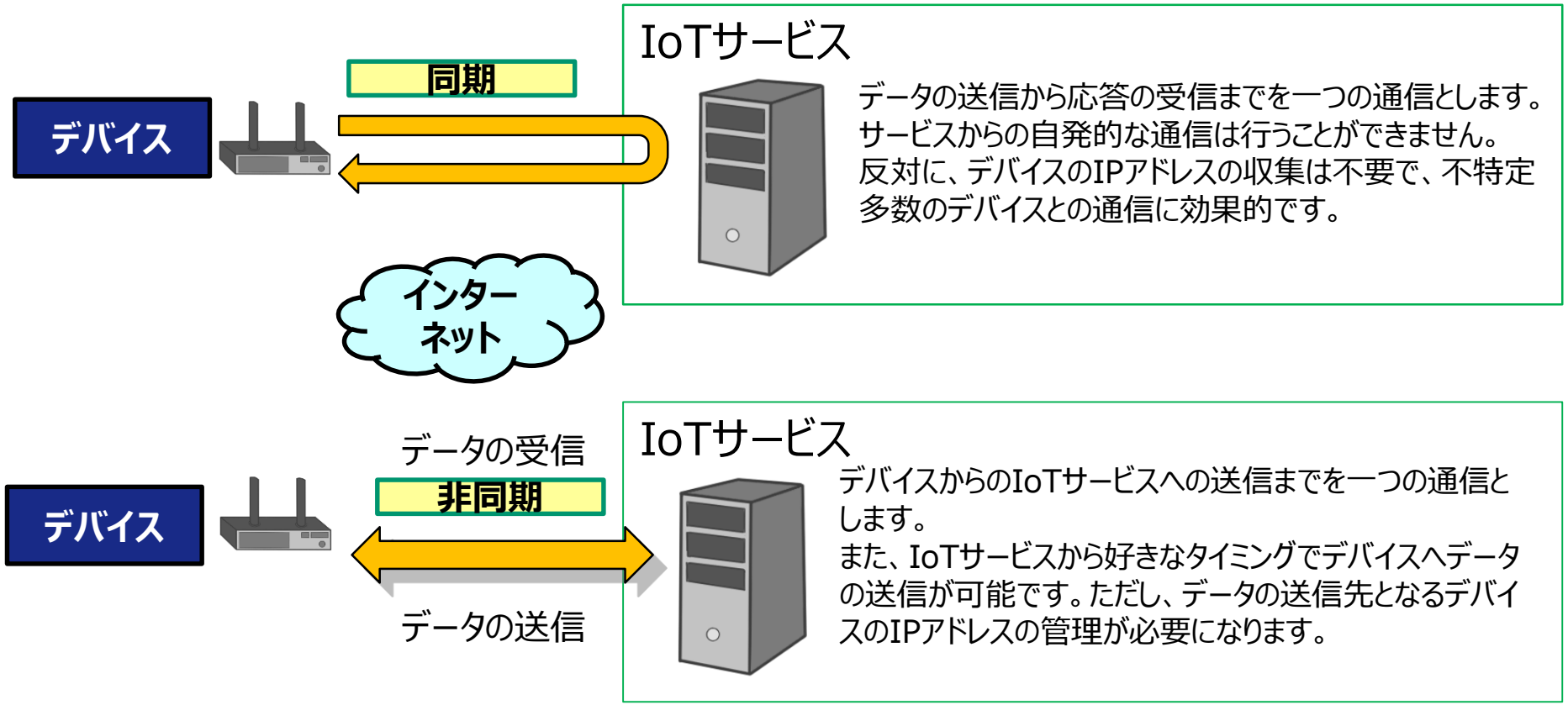
・仲介役のブローカー（Broker）
・情報発信者のパブリッシャー（Publisher）
・情報受信者のサブスクライバー（Subscriber）
の３つの機能から成り立ちます。



双方のIPアドレスはブローカーが管理

同期通信と非同期通信

デバイスとの通信には同期通信と非同期通信があります。

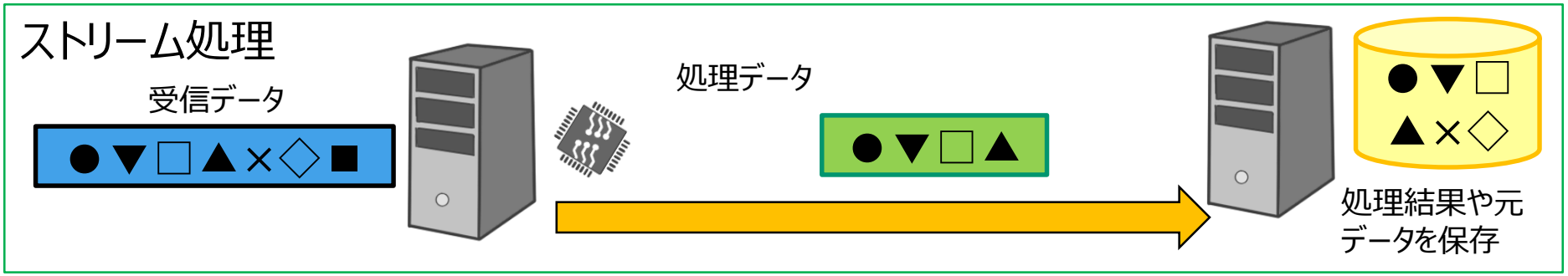


ビッグデータの処理と保存

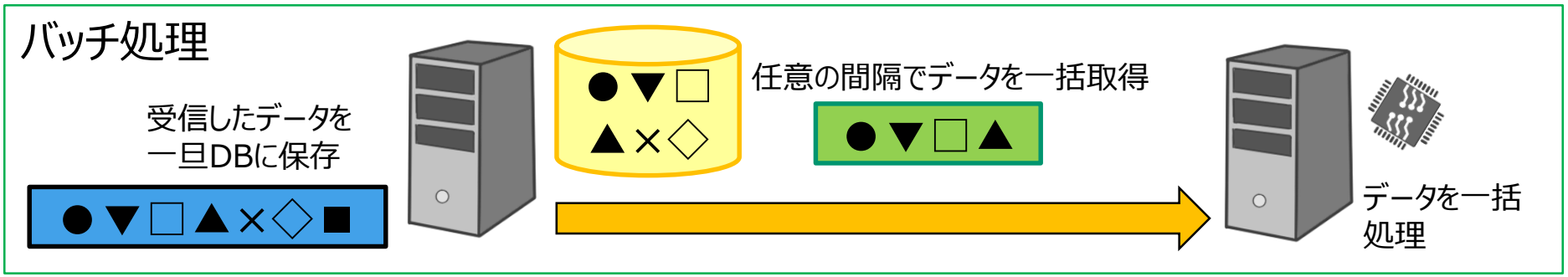
ビッグデータの処理

受信したデータは、データベースや分散ファイルシステムなどに保存されます。
また、受信したデータからデバイス制御の判断を行います。

処理と保存の方法には、ストリーム処理とバッチ処理の2種類があります。



データ内容を常に判断し、即座にデバイスを制御したい場合に利用



記録とデバイス制御にタイムラグがあっても問題ない場合に利用

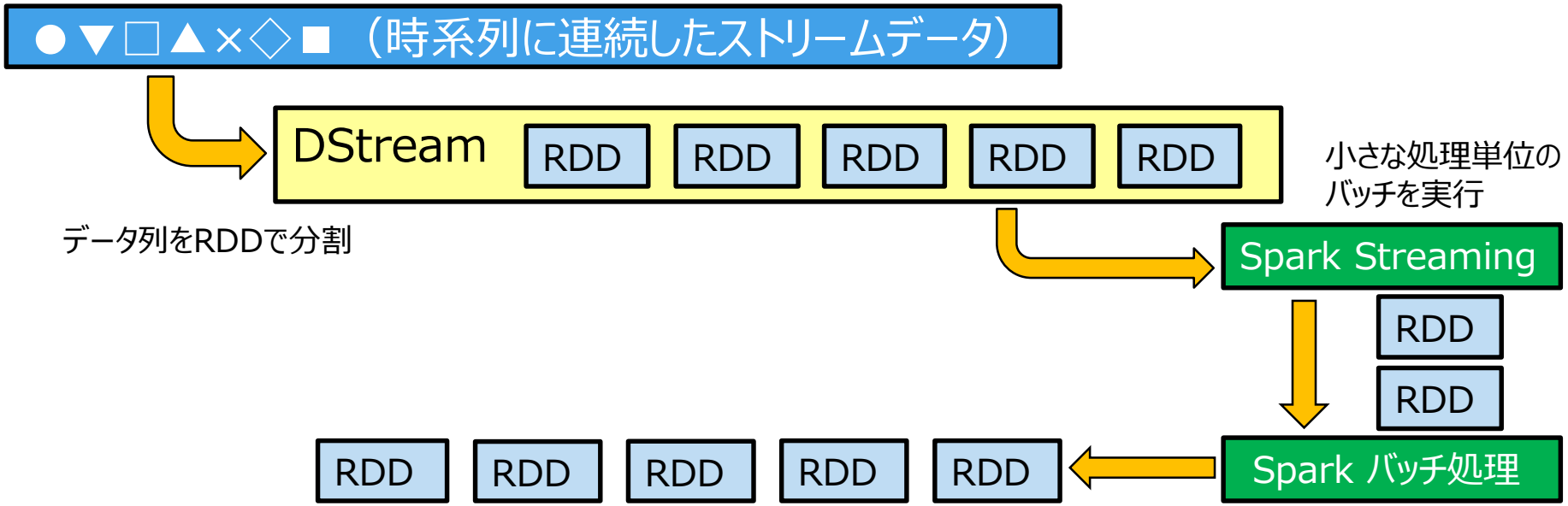
ストリーム処理

ストリーム処理

ストリーム処理はデータを保存せずに、処理サーバーに到着したデータを逐次処理する方法です。与えられたデータにリアルタイムで反応することができます。

Spark Streaming

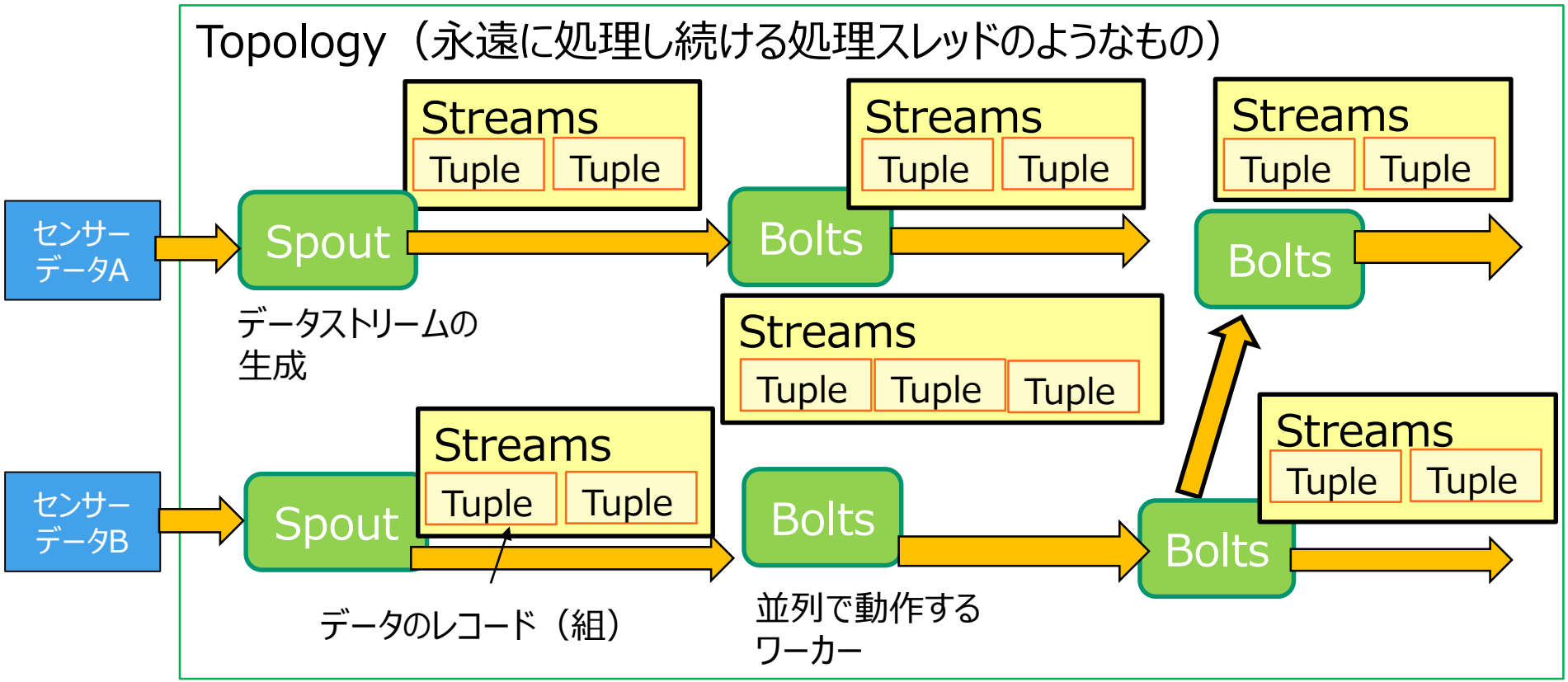
Spark Streamingは、ストリーム処理を行うためのSparkのライブラリです。
時系列的に連続したデータ列をRDDで分割し、分割されたRDDに対して小さな処理単位のバッチを実行します。



ストリーム処理

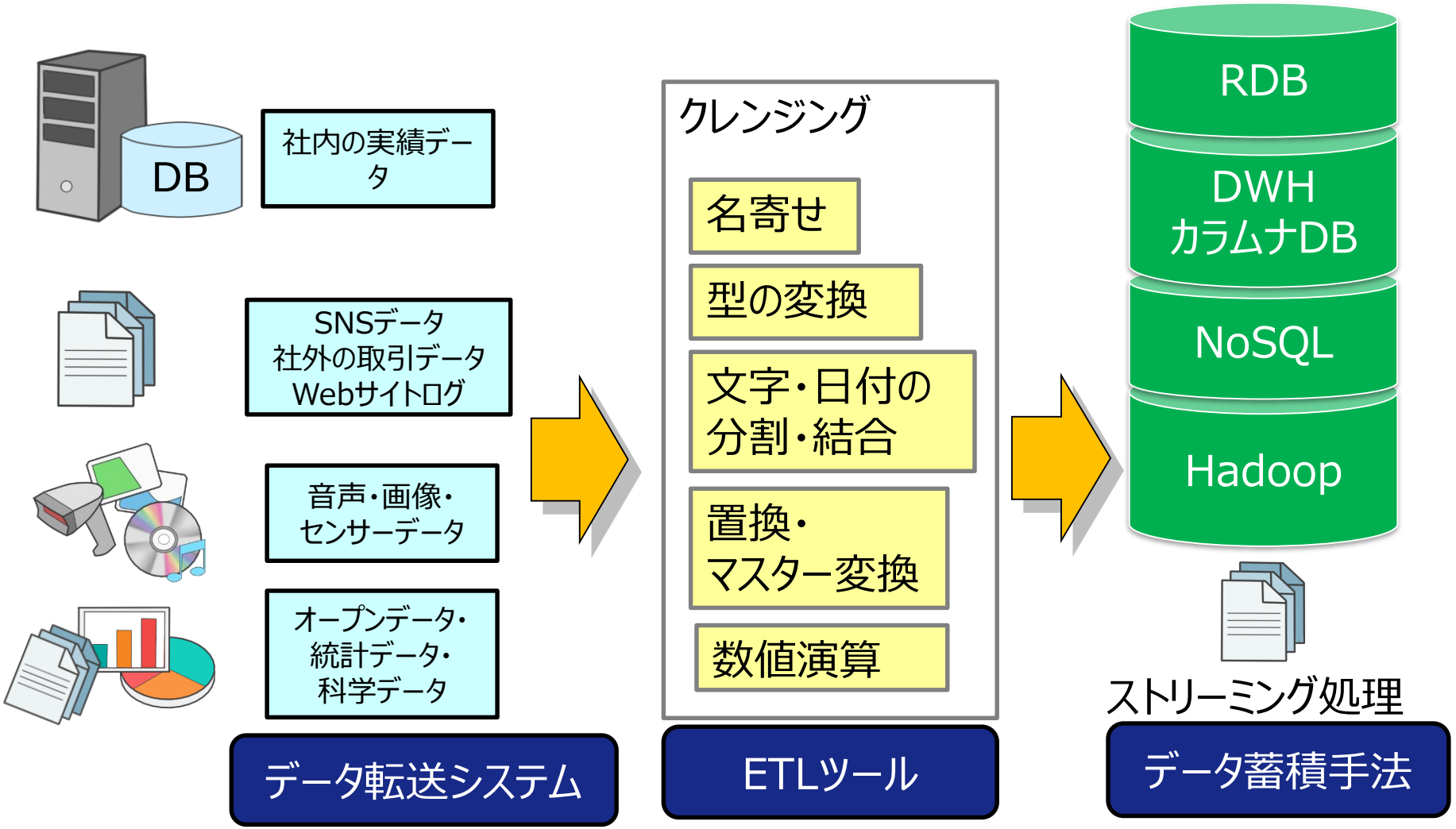
Apache Storm

Apache Stormとは、ストリーム処理のためのフレームワークです。



データ蓄積までの流れ

ビッグデータを分析可能な状態に処理する流れは以下のようになっています。



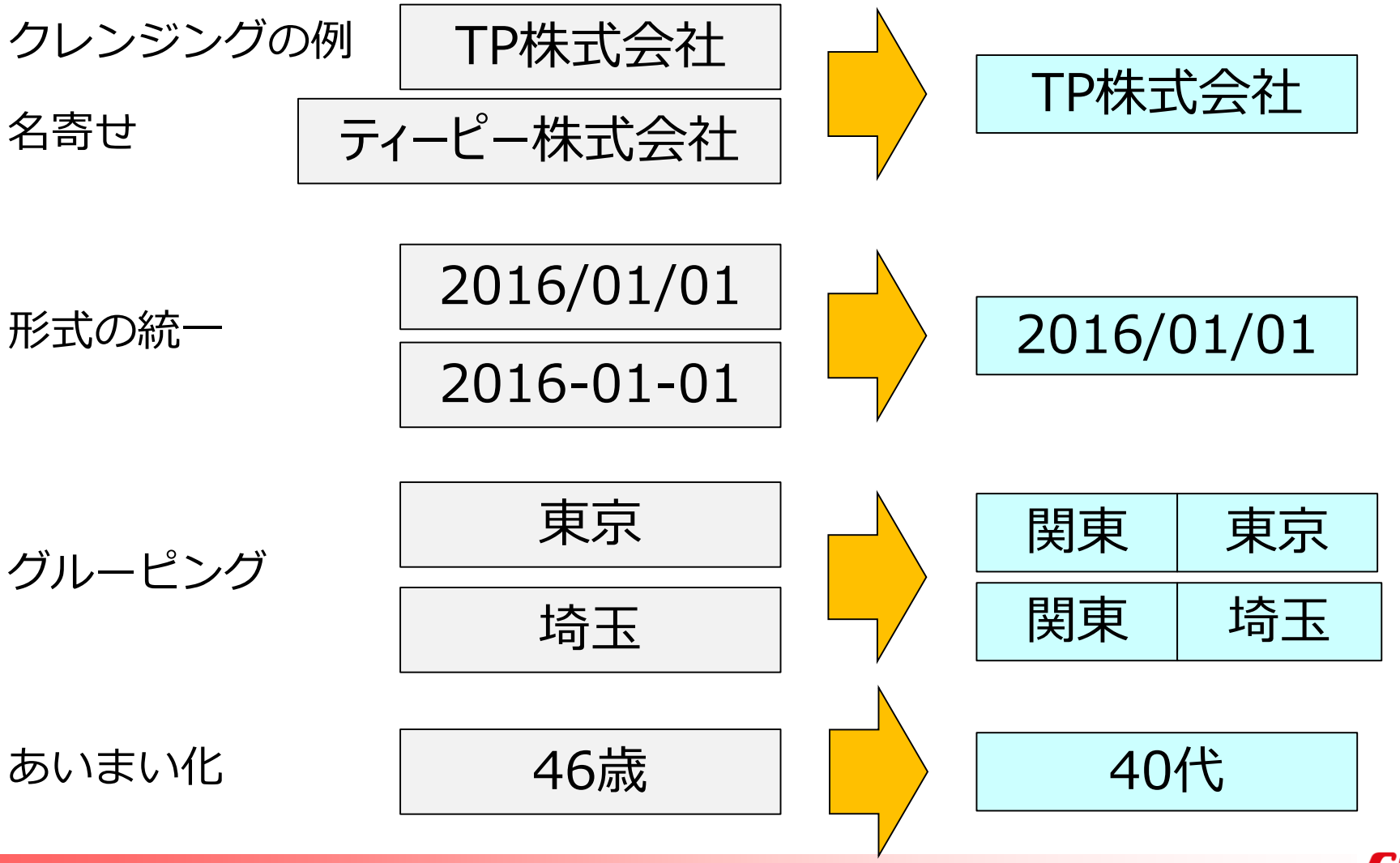
データ管理：クレンジング

クレンジング

データの整理や加工を行うことで、効率的に分析できるようになります。
これをクレンジングもしくはデータ加工と呼びます。
専用のツールやR、Python、GOなどのスクリプト言語で実施できます。

クレンジング対象	具体例	対処例
型の統一、日付、数値など	数値演算を行いたいデータに文字が入る場合など	基準を用意し、基準に合わないデータ修正
書式の不適合・表記ゆれ	住所、会社名、電話番号、郵便番号などの表記の不統一、通貨や数値の単位、文字コードなど	表記方法や単位の基準を定義し、基準に従うようにデータを修正する
異常データ	ある条件下でのセンサーの誤動作、データの無記入による空データ、データ入力ミスによる意味のないデータ	回帰やクラスタリングにおいて、結果の精度を下げる原因に異常値を検知し、事前に外す
個人情報およびプライバシー情報の保護	氏名、住所、メールアドレスなど個人を直接特定できる情報	個人を直接特定できる情報を削除または匿名化処理を施す

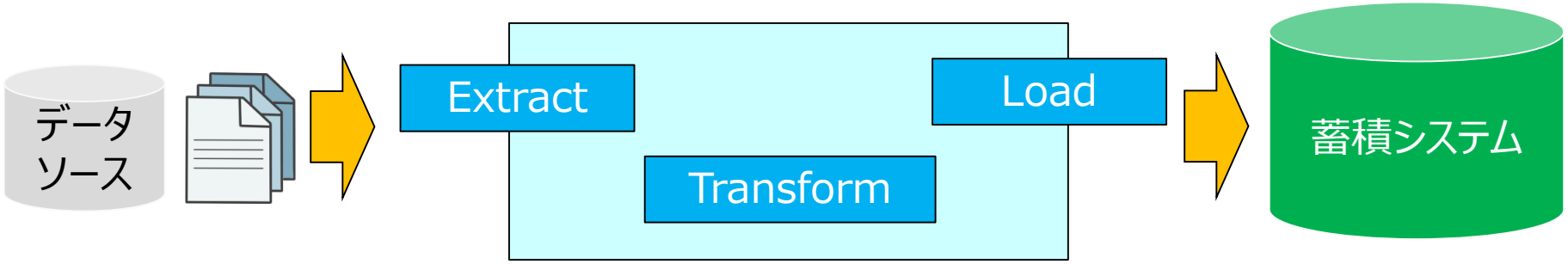
データ管理：クレンジング具体例



データ管理：ETLとELT

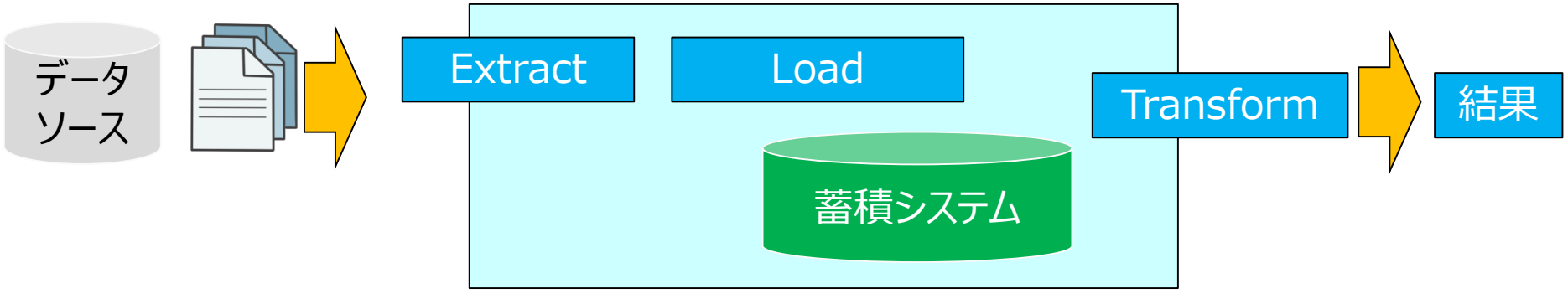
ETL（Extract Transform Load）

収集したデータを抽出(Extract)した後、利用しやすいように加工(Transform)し、DWHなどの蓄積先に書き込む(Load)一連の処理を表します。
大量データをバッチ処理する商用のETLツールなどが存在します。

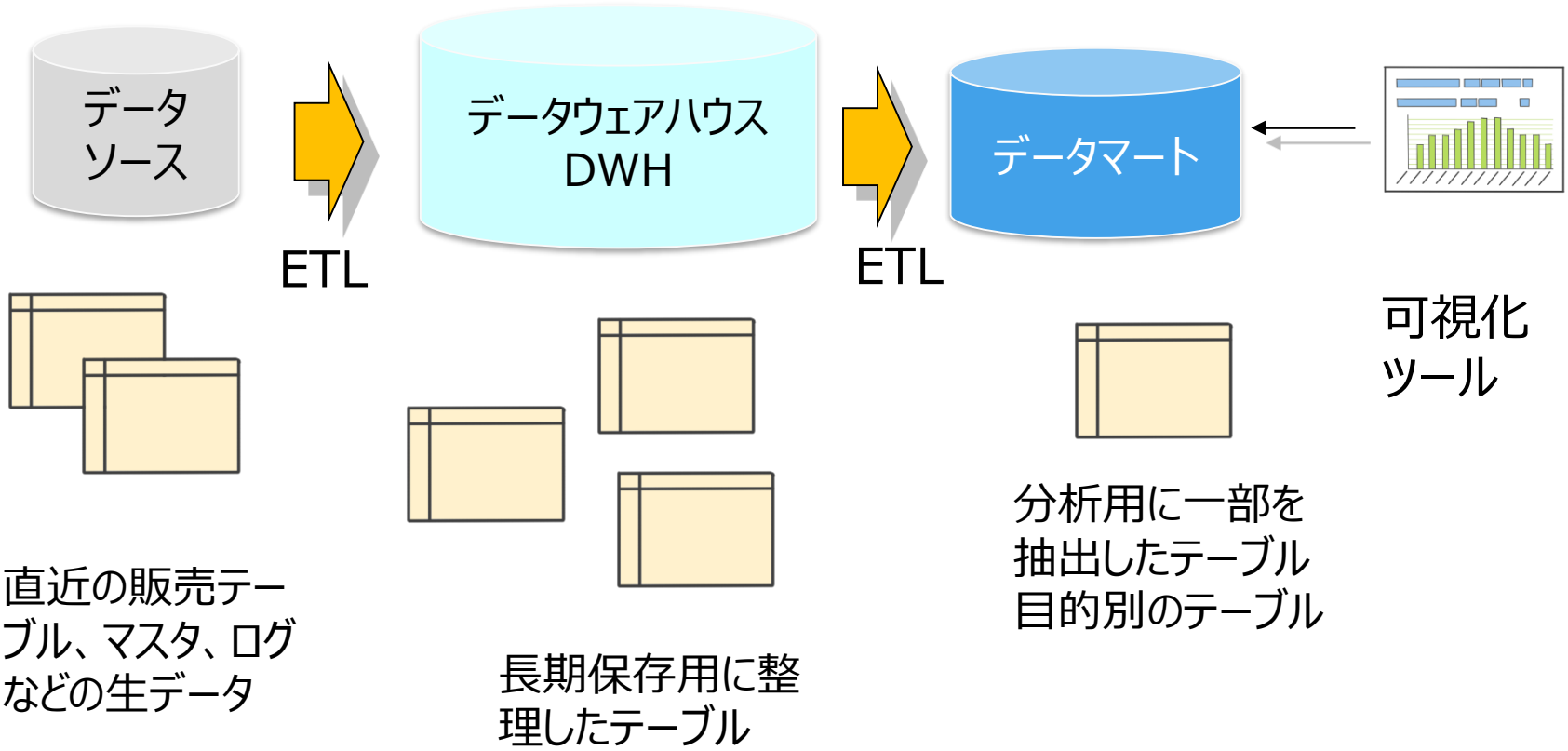


ELT（Extract Load Transform ）

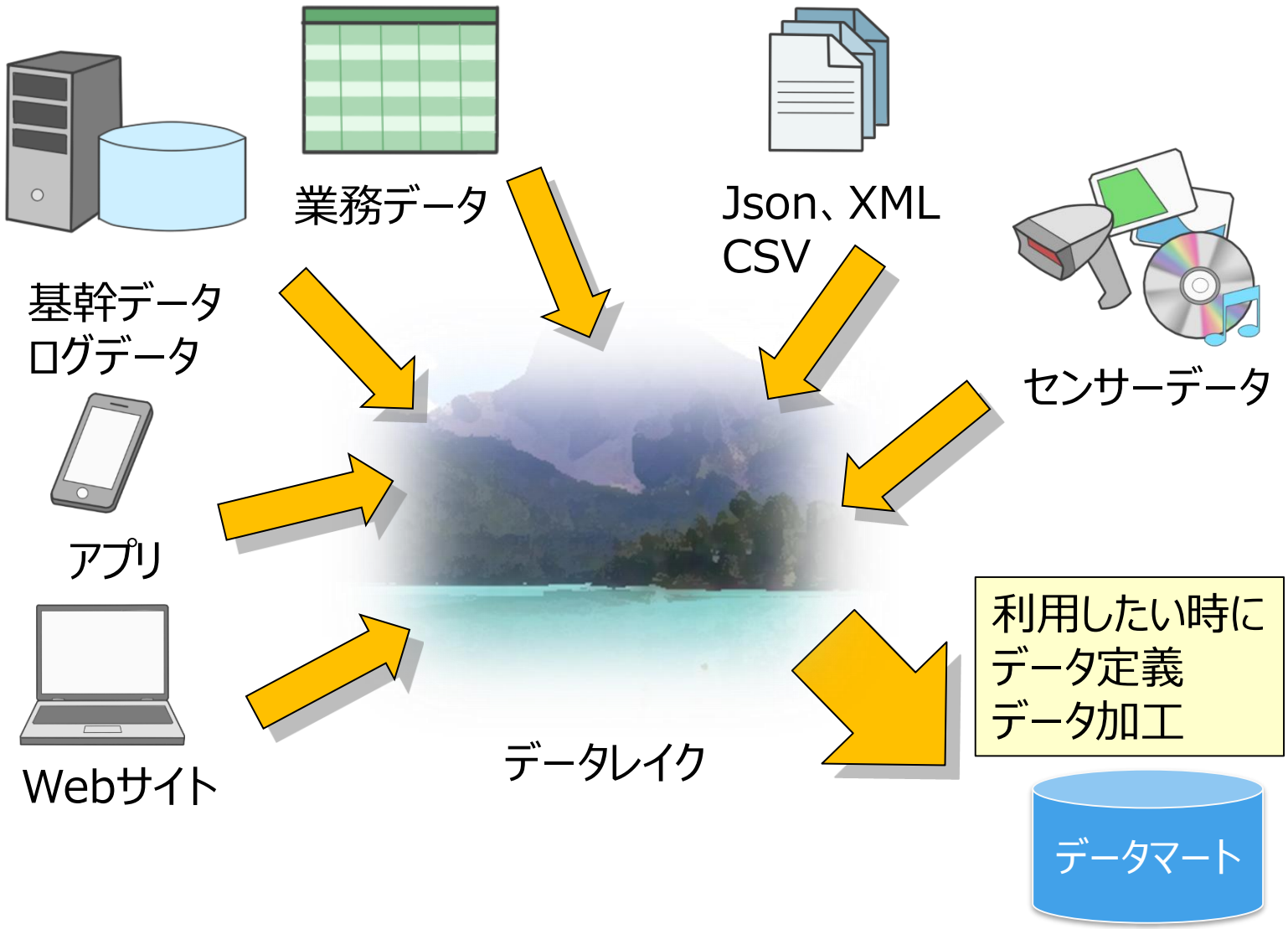
収集したデータを抽出(Extract)し、DWHなどの蓄積先に書き込み(Load)をした後、利用する時に初めて加工(Transform)を行う方式です。



データ管理：従来型データ蓄積



データ管理：データレイク



スループットとレイテンシ

ビッグデータ処理性能を測る指標

・スループット
一定時間に処理できるデータの総量。
WHやデータレイクなど、データの量が多い処理を行う時に重視。

・レイテンシ
データ処理が終わるまでの待機時間。
アドホック集計、データマート、BIツールなどで重視。

スループットとレイテンシは両立しないことが多く、複数のシステムで役割を分担することがあります。

従来型のデータベース：行指向DB

行指向データベース

テーブルの行を一つのデータとしてディスクに保存する方法です。

追記は末尾に加えるだけなので容易です。
トランザクションが多く発生する業務アプリなどで利用されます。

ディスク格納イメージ

2017-01-01	商品A	5000	10
2017-01-02	商品B	3750	1
2017-01-03	商品C	2160	5

高速化のためにIndexを利用することがありますが、分析対象となる列がわからないため、ディスクI/Oの軽減にならない場合が多いです。

従来型のデータベース：列指向DB

列指向データベース

テーブルのデータを列単位にまとめて保存する方法です。

集計に必要な項目だけを読み込むことができ、ディスクI/Oの軽減ができます。

ディスク格納イメージ

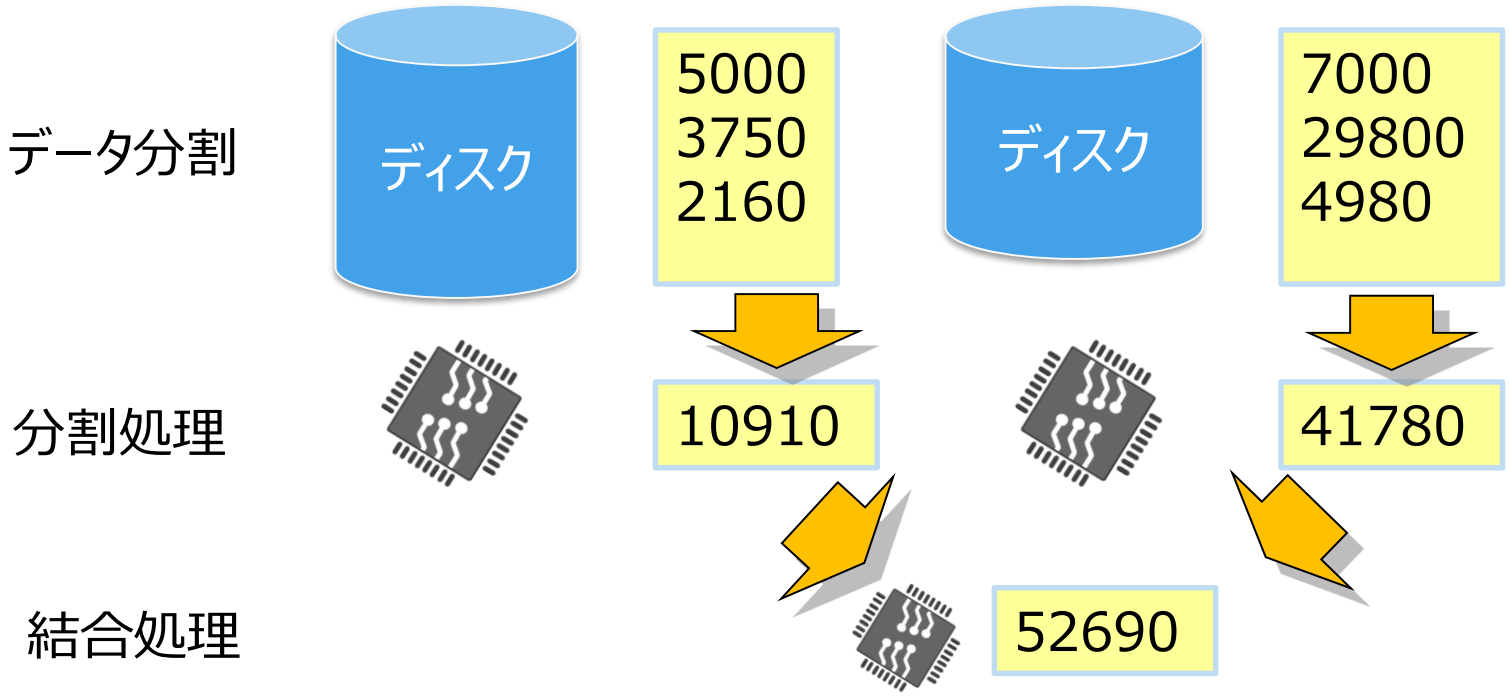
2017-01-01 2017-01-02 2017-01-03	商品A 商品B 商品C	5000 3750 2160	10 1 5

また、カラム単位での重複は集約が容易なため、圧縮効率が高いです。

従来型のデータベース：MPP

MPPデータベース（Massively Parallel Processing ）

一つのクエリを多数の小さなタスクに分解し、多くのCPUコアやコンピュータを並列的に稼働することで結果を得る手法です。



第３章

ビッグデータコア技術

NoSQL

NoSQLとは

NoSQLとは

・Not only SQL = RDBMS以外のデータベースの総称
・SQLを否定するものではない。
・1998年にCarlo Strozzi氏が初めて名称を使用した。(NoREL)
・非構造化データの蓄積や管理に利用される。

BASE

・Basically Available Soft-state Eventual consistency
・処理速度を優先するNoSQLで採用されているトランザクションの考え方。
…「稼動が第一で、厳密な整合性は過程ではなく結果でのみ重要視する」

ビッグデータで扱うデータの種類

ビッグデータの種類	データの例
構造化データ	データベースに格納されたデータ など (顧客テーブルデータ、 受注テーブルデータ など)
準構造化データ	ログデータ、センサーデータ、SNSに書き込まれたデータ など
非構造化データ	文書、音声、動画、画像 など

NoSQLのメリット、デメリット

NoSQLのメリット

・前もったデータの構造の定義が不要で、柔軟な変更が容易である
・特定の形式に固執しないため、複数のサーバにデータの分散ができる
・データ構造が単純で、更新や検索処理の速度が速い
・基本的にPutとGetのみを行えばよく、JOINがないため操作が明快

NoSQLのデメリット

・RDBMSと比較すると、データ間の整合性を維持する機能が弱い

NoSQLのメリット、デメリット

NoSQLとRDBMSとの比較

	RDBMS	NoSQL
保存に適するデータ	構造化データ	非構造化データ
スキーマ定義	事前定義が必要 固定的で変更しにくい	事前定義不要 データ構造を変更しやすい
データの整合性	同時実行制御がある データ整合性を重視	比較的緩い 大容量データの高速処理を優先、結果整合重視
データ操作命令	SQL言語 (SELECT、INSERT、UPDATE、DELETE)	Put(書き込み) Get(読み出し)
拡張方式	スケールアップ	スケールアウト

NoSQLのメリット、デメリット

データベースの拡張方式の比較

RDBMS　：スケールアップを行う
NoSQL　：スケールアウトを行う

スケールアップ

1台のサーバの処理性能やストレージを拡張します。

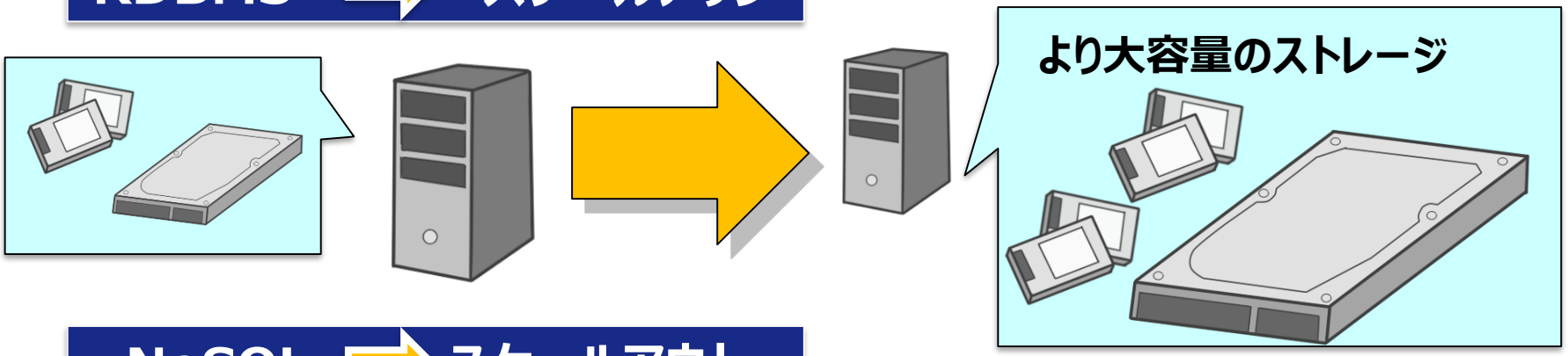
スケールアウト

サーバを増設し処理性能を向上させます。

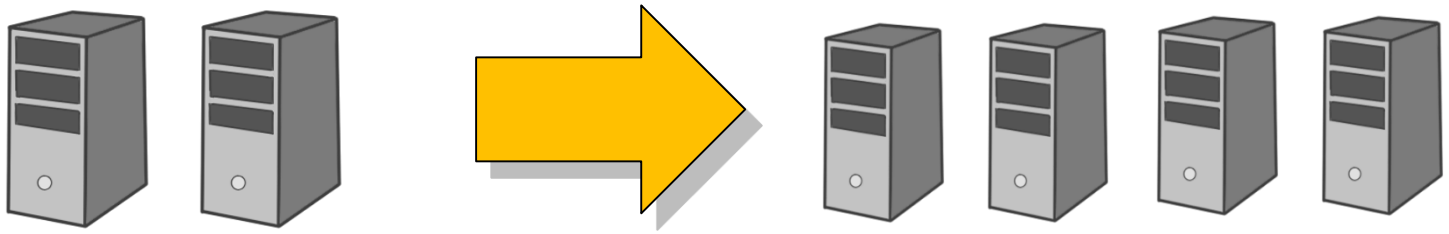
NoSQLのメリット、デメリット

データベースの拡張方式の比較

RDBMS ⇨ スケールアップ



NoSQL ⇨ スケールアウト



NoSQLのメリット、デメリット

スケールアップ

ハードウェアの性能(ディスク、CPU、メモリ)の増強により、特定の1台のサーバの性能を向上させます。

RDBは以下の特徴を持つため、スケールアップが効果的です。

・複数のサーバで運用すると、データ分割や配置作業が必要となる

・サーバ間の大規模トランザクションによりパフォーマンスが低下する

スケールアウト

サーバの台数を増やし、全体での処理能力を向上させます。

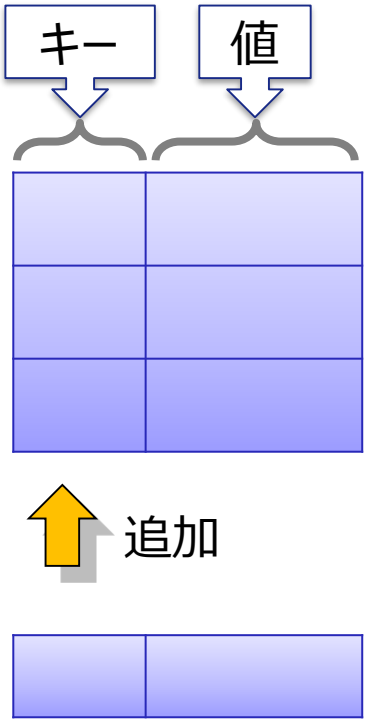
高性能がサーバに求められないため、低コストで処理性能を上げることが可能です。

また、1台サーバの停止による影響は少なく、稼働率は向上します。

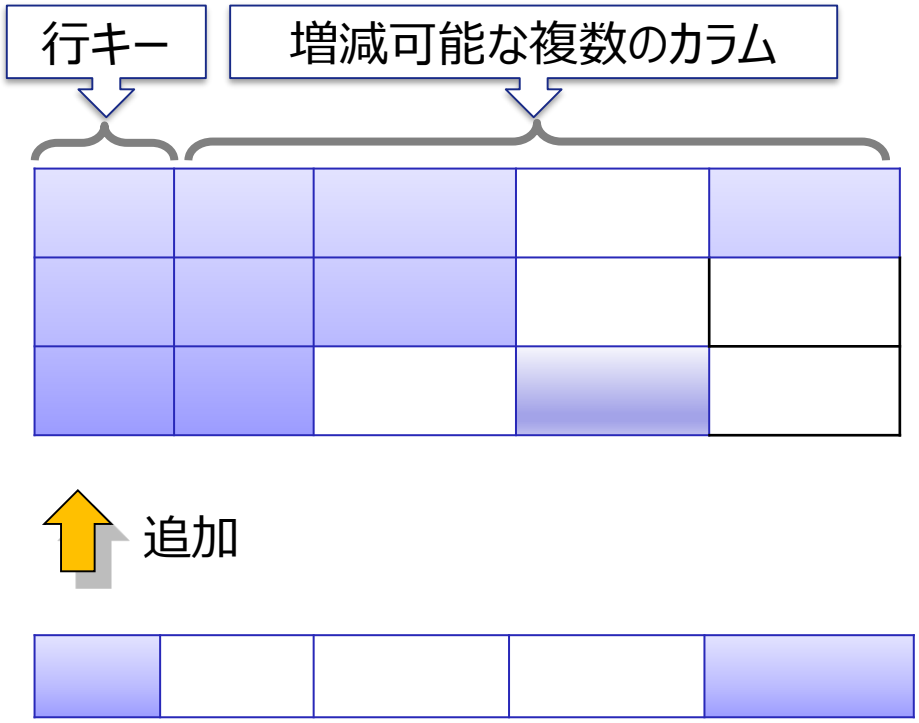
NoSQLは、データベースで扱うデータが非構造化データなため、複数のサーバへの分割・複製が容易、つまりスケールアウトの手法に向いています。

NoSQLの代表的な種類

キー・バリュー型



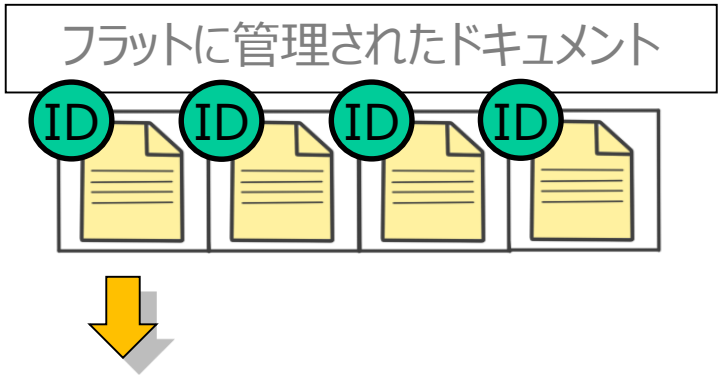
列指向型（ワイドカラム型）



列は後から追加も可能。列がない行データも許容。

NoSQLの代表的な種類

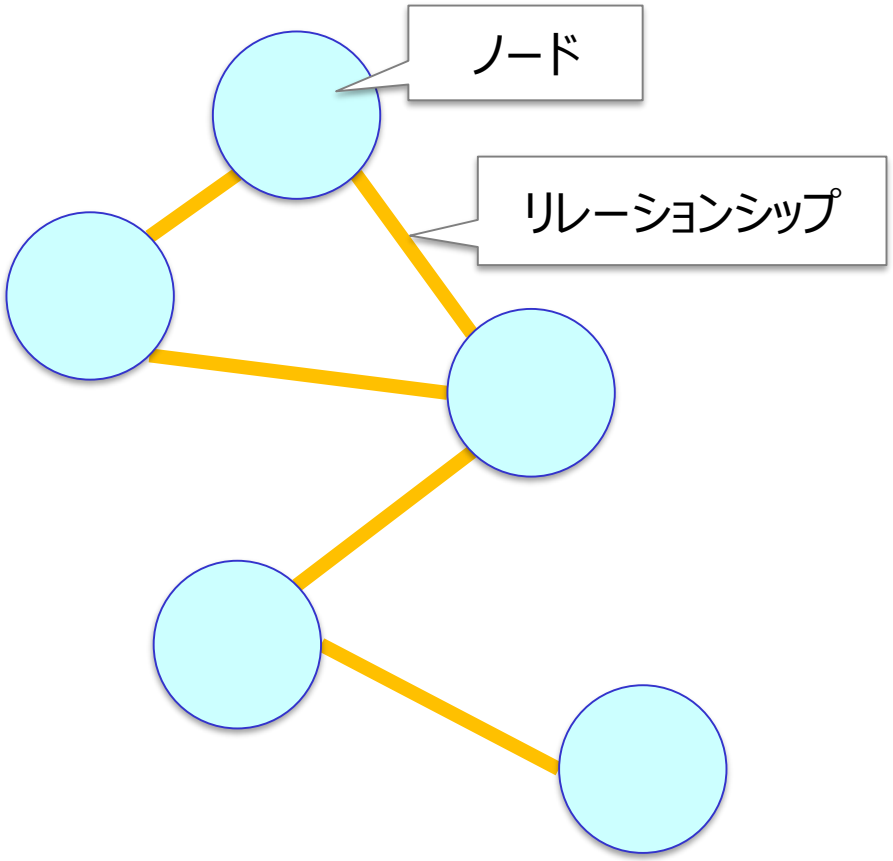
ドキュメント型



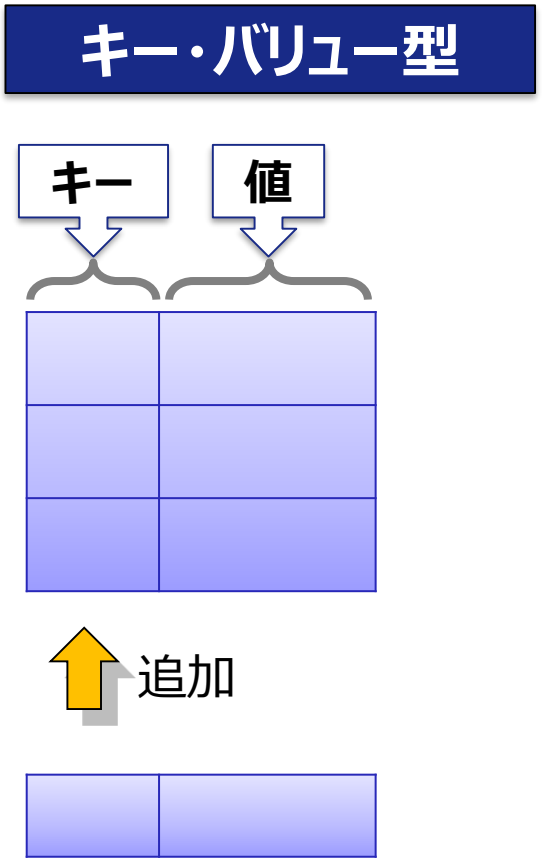
ドキュメントの例 (JSON)

```
{
“title” : “今日のランチ”
“userName” :
“Morimoto”
“postedDateTime” :
“2014/10/10 13:00”
“content” : “カレー”
 }
```

グラフ型



キー・バリュー型の特徴



キー・バリュー型

キーとバリュー(値)のセットでデータを整列します。
キーは、値に対する識別子となるデータです。
値はバイナリデータであれば、BLOB型の画像や音声も格納できます。

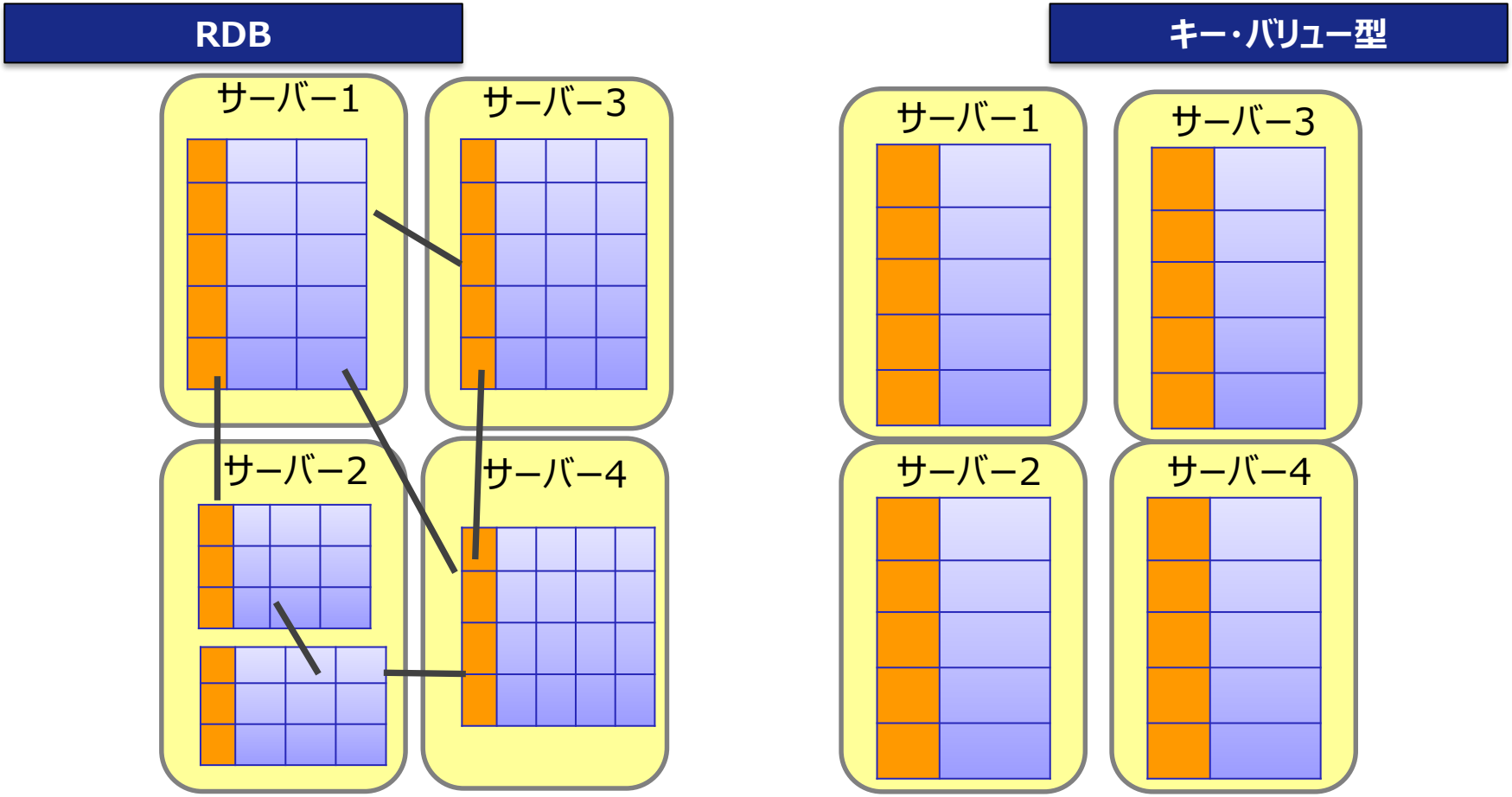
キーとキーに紐付けられたデータという単純な構造であるため、スキーマ定義が不要で、検索が高速化でき、またサーバの追加やデータの分割・配置作業が容易となります。つまり、簡単にスケールアウト環境が構築できます。

Dynamo、Voldemort、Riak、Hibari、Redis、Scalaris、TokyoCabinet/Tylantなどで採用されています。

キー・バリュー型の特徴

RDBとキーバリュー型NoSQLのスケールアウト比較

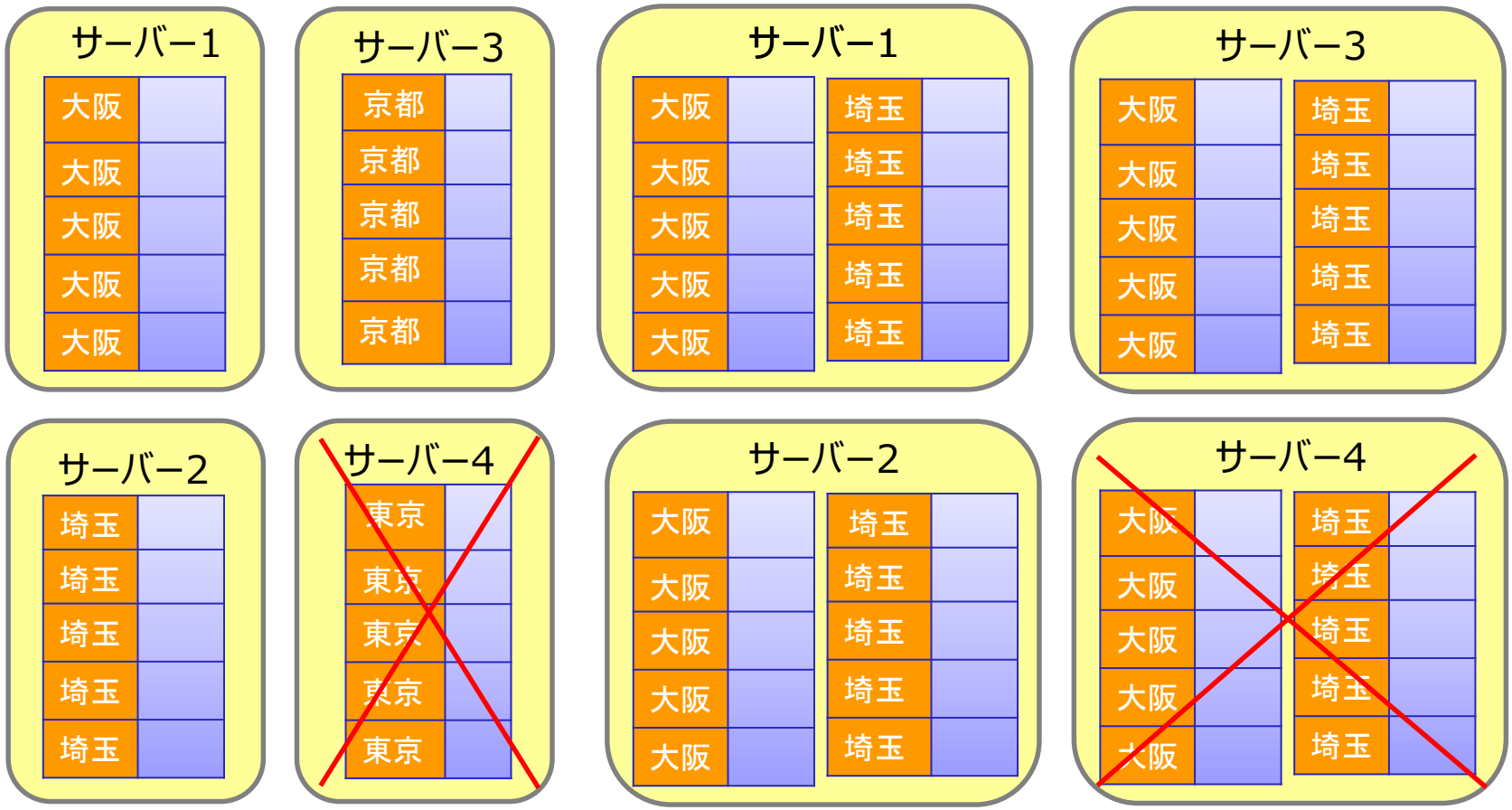
キーバリュー型では、整合性を確認する必要性がないので、キーの管理さえしていればサーバを増やすだけで問題はありません。



キー・バリュー型の特徴

シャーディング / Sharding

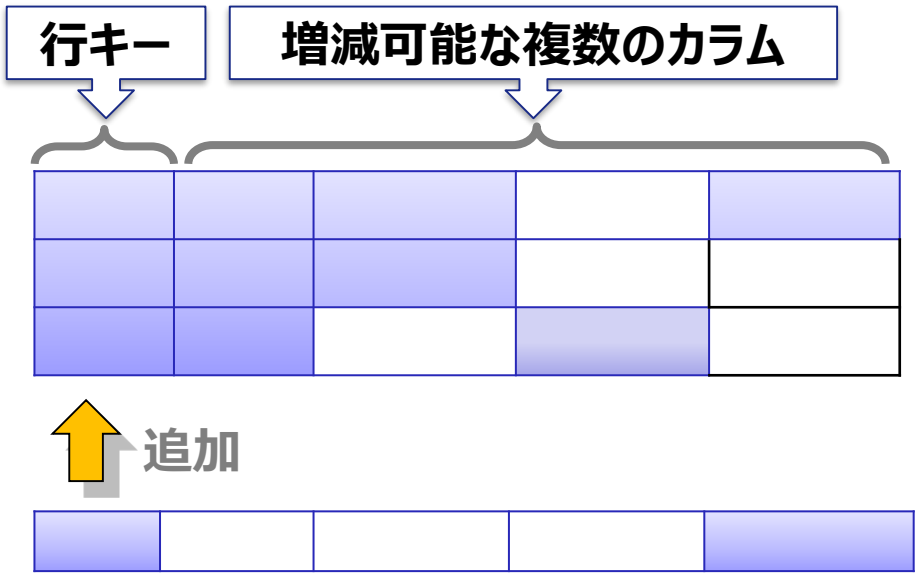
あらかじめサーバ毎に保存するデータのキーの範囲を設定しておくことで、検索や管理を容易にする機能です。



列指向型の特徴

キー・バリュー型を機能強化したもので、各行のキーが複数の列(カラム)の値を持つことが可能です。

列単位でデータの保存や読み込み処理を行うことができ、列の集計処理などを高速に実行することができます。



一部の列のデータが欠けている行があっても問題ありません。また、行や列の数は自由に好きなだけ拡張可能です。

Bigtable、Cassandra、Hbase、Hypertableが採用

※カラムナデータベースとは別物です。

列指向型の特徴

Twitterの例

行キー：ユーザーネーム
カラムの名前：ユーザーID
カラムファミリー ＝ カラムをまとめる入れ物

ユーザーネーム・カラムファミリー

行キー	カラム
ユーザーネーム	ユーザーID
Tanaka	A1234

ユーザータイムライン・カラムファミリー

行キー	カラム1	カラム2	カラム3	カラム4	カラム5
ユーザーID	a1234	a1235	a1236	a1237	a1238
A1234	t1234	t1238	t1276	t1287	t1299

横に増える

ユーザー・カラムファミリー

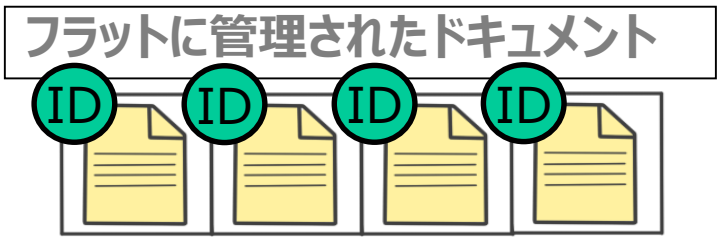
行キー	カラム1	カラム2
ユーザーID	ユーザーネーム	パスワード
A1234	Tanaka	sder5k

ツイート・カラムファミリー

行キー	カラム1	カラム2	カラム3	カラム4	カラム5
ツイートID	ユーザーID	ユーザーネーム	ボディ	タイムスタンプ	
t1234	A1234	Tanaka	やぁ	a1234	

縦に増える

ドキュメント型の特徴



ドキュメントの例 (JSON)

```
{
“title” : “今日のランチ”
“userName” : “Morimoto”
“postedDateTime” :
“2014/10/10 13:00”
“content” : “カレー”
“count” :2
 }
```

ドキュメント指向型

JSONやXMLなどのデータ記述書式のドキュメントを格納します。

階層構造は持たないフラットなドキュメントの形式でデータを管理します。各ドキュメントには管理のためのユニークIDが割り振られます。

スキーマレスです。(データ設計が不要です。)

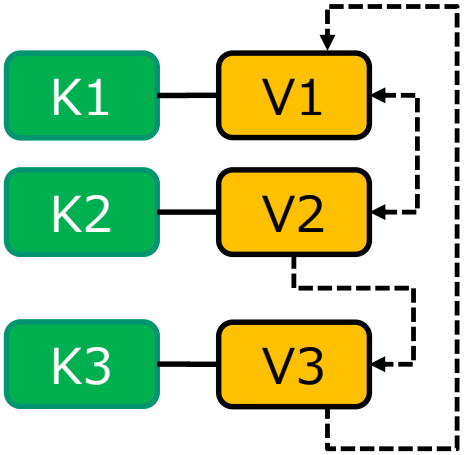
ドキュメントの内部構造はDB、アプリケーションからわかり、任意の内容でクエリを作成できます。

MongoDB、CouchDBで採用されています。

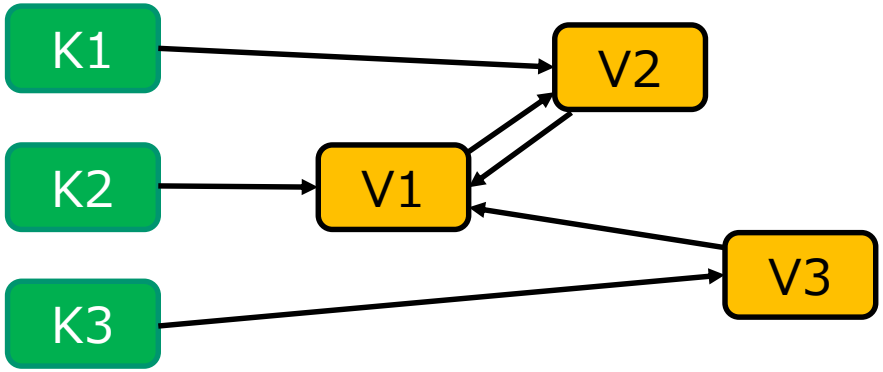
グラフ型の特徴

グラフ型

- キーバリュー型格納の例



グラフ型格納の例



データ同士の入り組んだ関係を独立して表現します。
グラフ型は以下の要素により構成されます。

ノード (node) … 他のノードと関係をもつ要素を表す。
リレーションシップ（RelationShip） … ノード間における関係の有無と方向を矢印で示す。
プロパティ（Property） … ノードとリレーションシップの具体的な属性を示す。

代表的なNoSQL製品

代表的なNoSQL製品

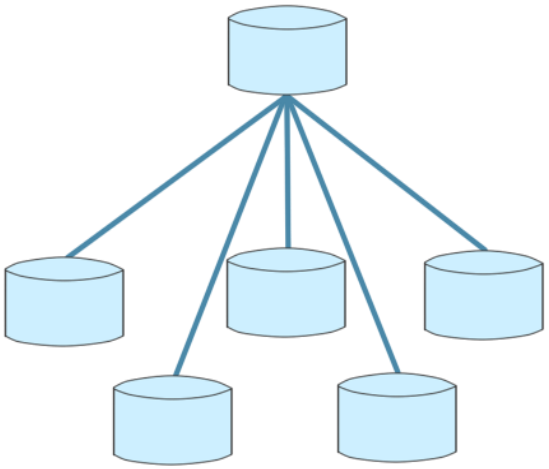
NoSQL製品はさまざまなものが多数存在します。

NoSQL分類	製品	ベンダー
キー・バリュー型	Amazon DynamoDB	Amazon
	Hibari	クラウディアン
	Riak	Basho Technologies
	memchached	(オープンソース)
列指向型	HBase	(オープンソース)
	Apache Cassandra	(オープンソース)
	Bigtable	Google
ドキュメント型	Apache CouchDB	(オープンソース)
	MongoDB	(オープンソース)
グラフ型	InfiniteGraph	Objectivity, Inc.
	Neo4j	オープンソース　/ 商用ライセンス(Neo Technology)

NoSQLの基本的概念と技術

NoSQLのデータベースアーキテクチャ
サーバの配置方法には、マスタ型やP2P型の2つがあります。

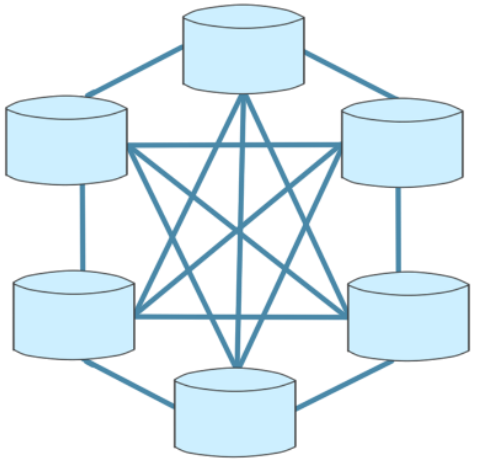
マスタ型



マスターノードはノード管理を役割を持ちます。
単一障害点(SPOF)が存在します。
※マスターノードがSPOFです

例 BigTable、CouchDB、
Hbase、Hibari、MongoDB

ピア・ツゥ・ピア(P2P)型



単一障害点はありません

例 Cassandra、Dynamo
Riak、Voldemart

NoSQL時代の必要要件

必要要件

NoSQLに求められることは様々であり、全てをみたすデータベースは存在しません。

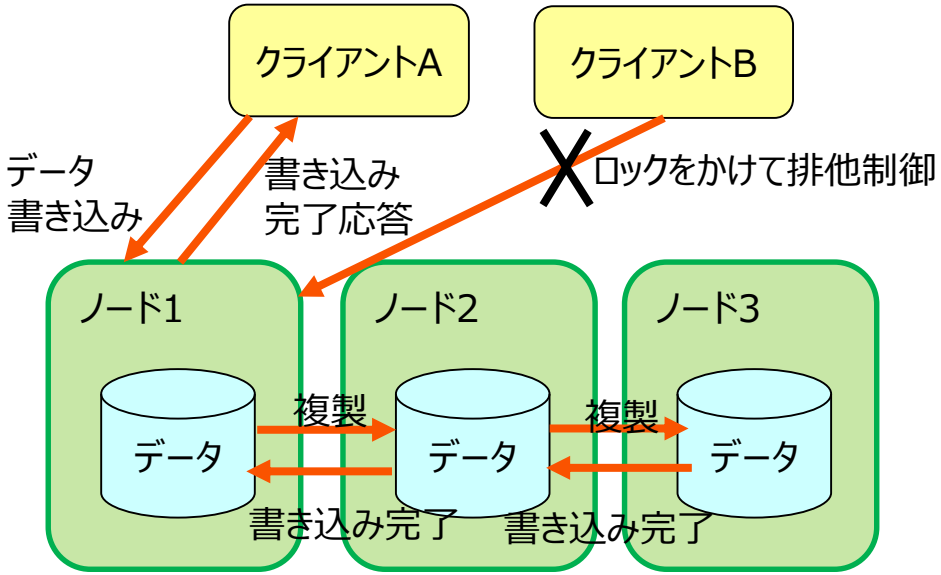
- 膨大なデータ　：複数のサーバを必要とするデータ量を扱う
- 分散配置　：データを複数のサーバに分けて保存する
- コスト　：性能は求めず、安価なハードウェアを多数用意する
- 信頼性　：データの欠損や漏洩を防ぐ
- 可用性　：稼働率を高く維持する
- 耐障害性　：障害による影響を最小限にとどめる
- 性能　：処理や応答をできる限り早く行う

NoSQLの基本的概念と技術（整合性）

データの整合性（Consistency）とは
　作業の実行後に複製(レプリケーション)されたデータ間に矛盾がないことを示します。
整合性には、「強い整合性」と「緩い整合性」があります。

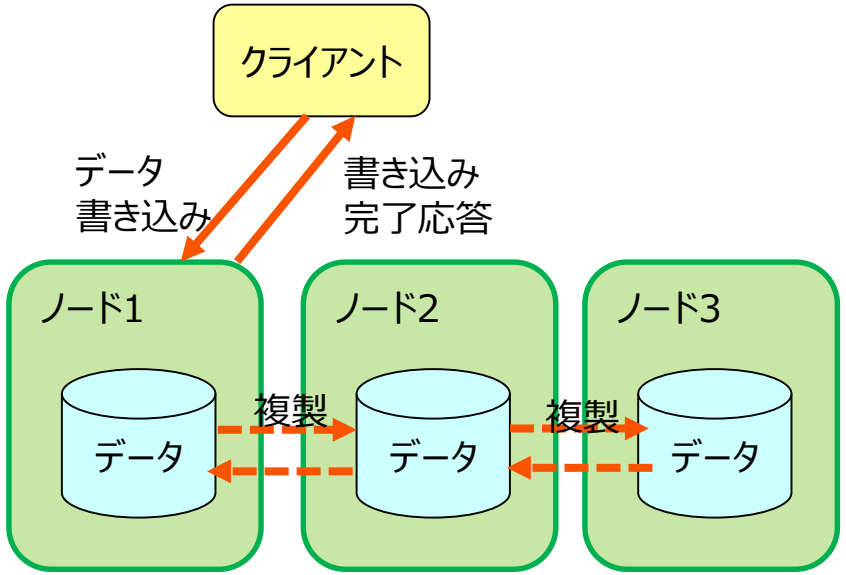
強い整合性

必ず最新の更新データが出力されることを保証します。特に、RDBでは重要視され、保持されます。



緩い整合性

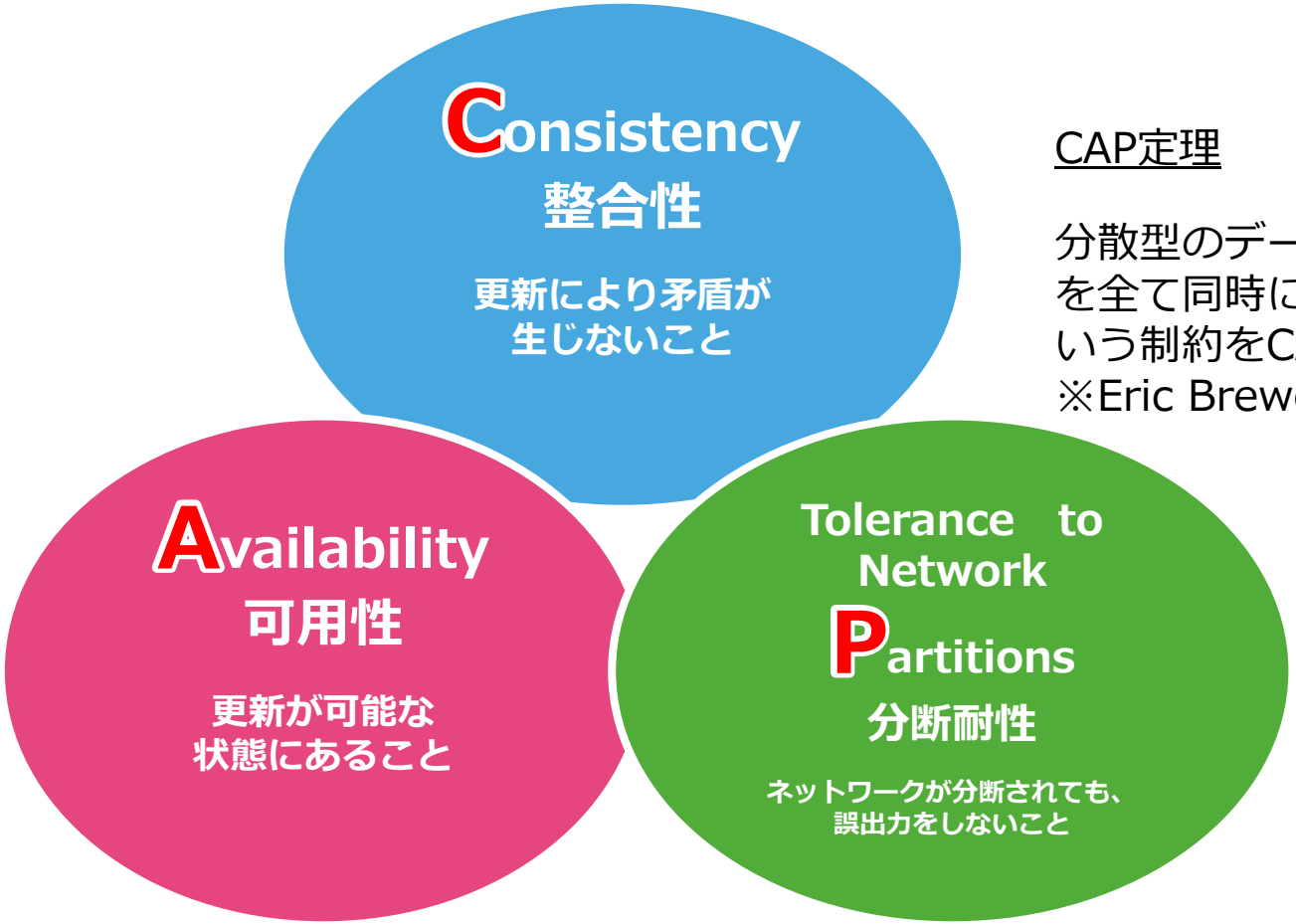
更新から少し時間をおいた後であれば、全てのノードで最新のデータが読み出せればよいとする考え方です。



書き込み完了確認がない状態でクライアントに完了通知を行う

NoSQLの基本的概念と技術（整合性）

データベースの特性でも重要な以下の3つの要件を考えます。

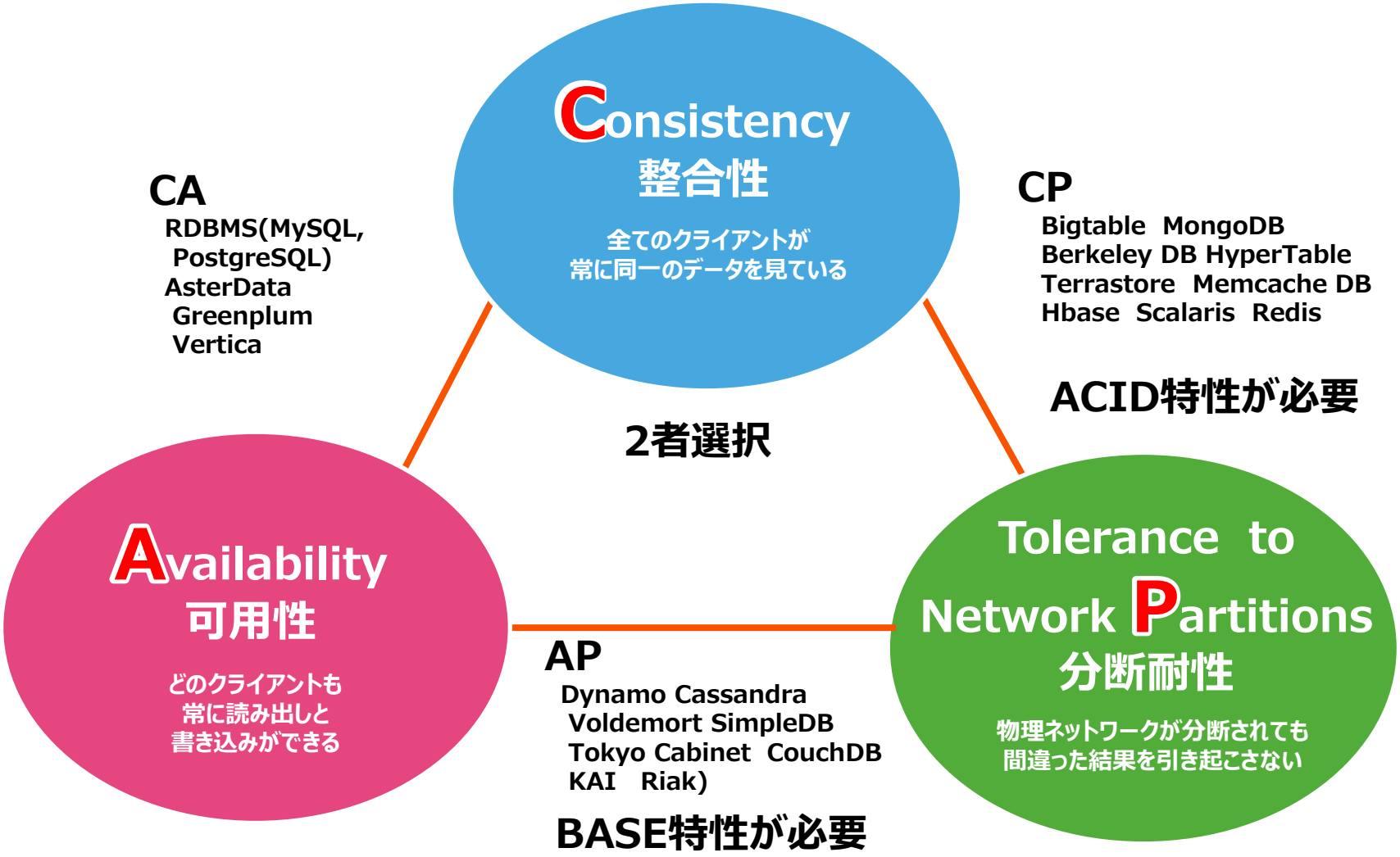


CAP定理

分散型のデータベースでは、3つの要件を全て同時に満たすことはできない、という制約をCAP定理と呼びます。
※Eric Brewer氏により提唱されました。

NoSQLの基本的概念と技術（整合性）

NoSQLではCPかAPのいずれかが保証されることが基本です。



NoSQLの基本的概念と技術（整合性）

ACIDとBASE

・ACID(Atomicity、Consistentry、Isolation、Durability)

トランザクションは、全ての処理完了または処理以前の状態への復元のいずれかを行う仕組み。

Atomicity 原子性	：手順が全て実行されるか、一つも実行されないか
Consistency 一貫性	：トランザクションの前後で整合性が保たれる
Isolation 独立性	：実行中に他の処理に影響を与えない
Durability 耐久性	：データが正しく記録され、損失の可能性がない

・BASE(Basically Available、Soft-state、Eventual Consistency)

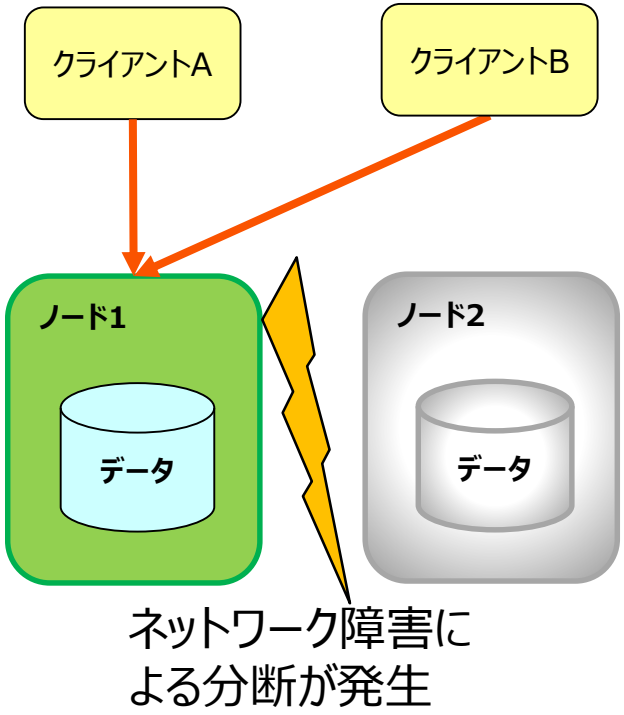
アプリは基本的に常に動作し（Basically Available）、強い整合性は持っていないが（Soft-state）、最終的に整合性をもつ（Eventual Consistency）という特性を備えているべきであると言う考え方。

NoSQLの基本的概念と技術（整合性）

ネットワーク障害時のCPとAPの対応

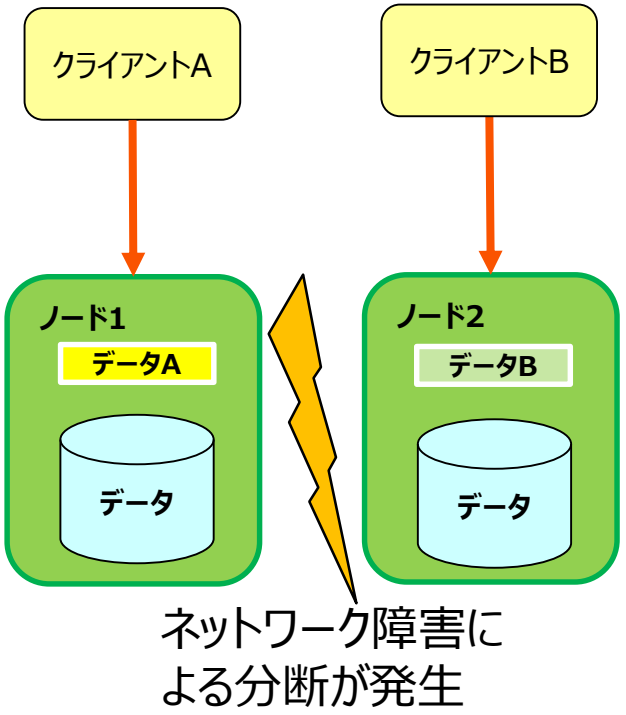
CP（整合性と分断耐性）

リクエストは片方のグループだけが受け付け、他のグループは自動的に停止します。データを受け付けないグループは更新が起きないため、整合性が乱れる心配はありません。



AP（可用性と分断耐性）

全てのノードで更新を行えるようにするため、リクエストは全てのグループが受け付けます。情報の共有ができないノード間で不整合が発生します。

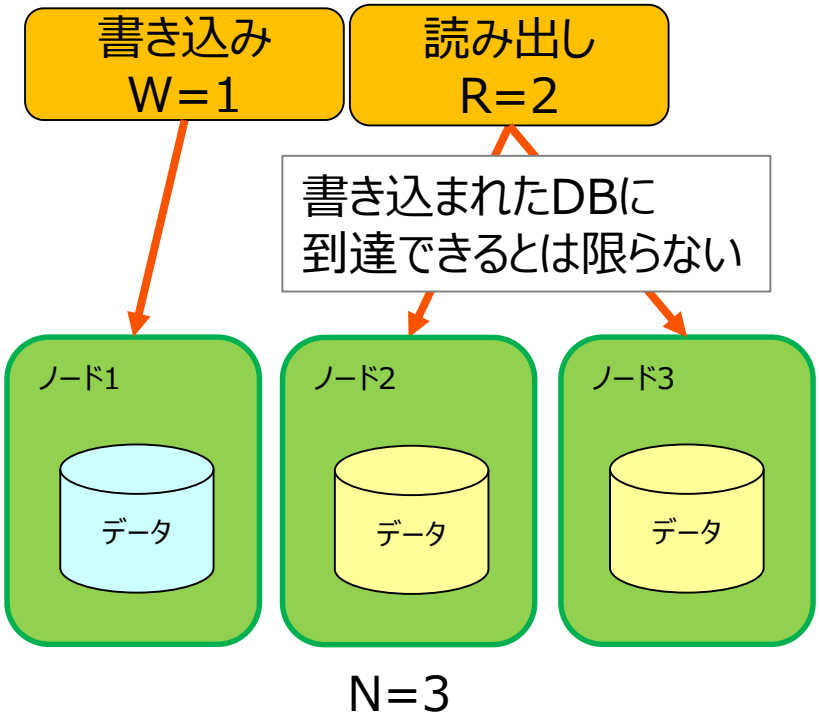


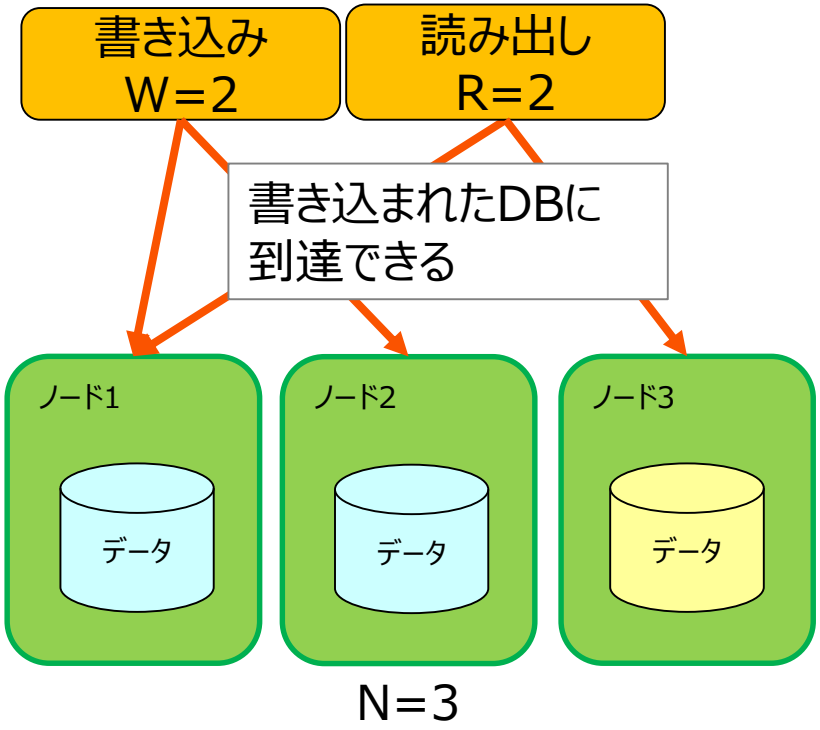
NoSQLの基本的概念と技術（整合性）

整合性と性能のトレードオフ

R(ノードから読み込む数)とW(ノードに書き込む数)の和がN(レプリカの数)より大きければ、整合性が保証することができます。

複数の書き込みと読み出しを行う事は、応答速度とのトレードオフになります。

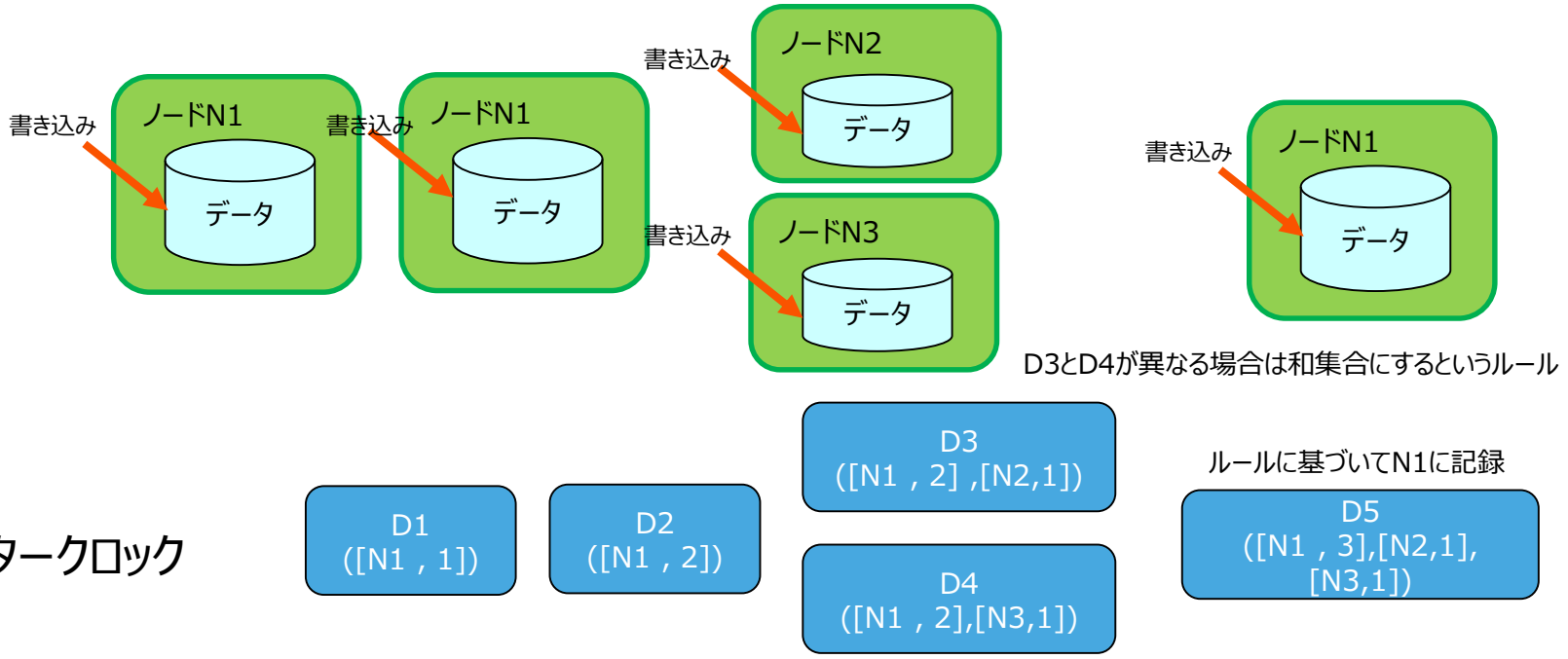




NoSQLの基本的概念と技術（整合性）

データのバージョン管理

- タイムスタンプ(TimeStamp)
  データの更新時間を記録する。
  新しいタイムスタンプを持つものを重要視することでバージョンを管理する。
- ベクタークロック(Vector Cloks)
  複数のバージョン間の仲裁をする仕組み。
  変更が行われたノードと、そのノードで変更が行われた回数を記録する。



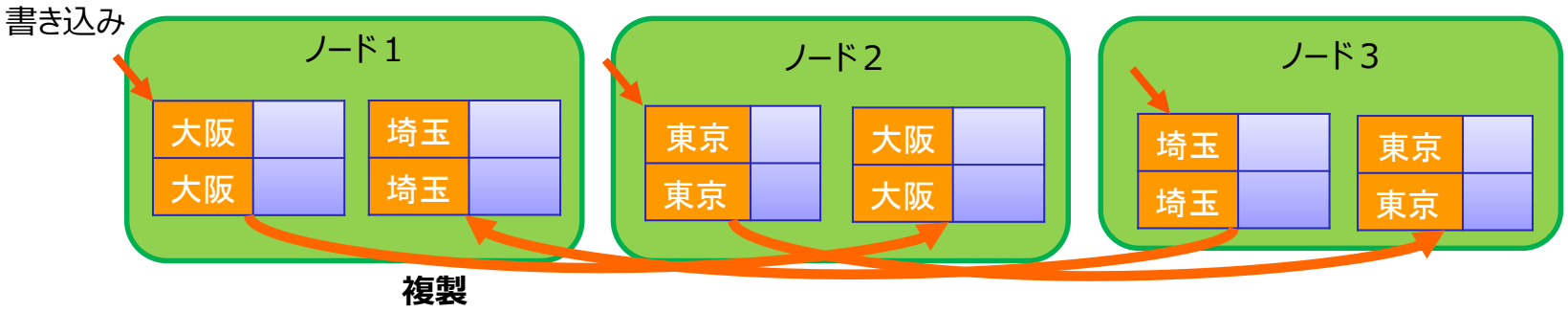
NoSQLの基本的概念と技術（データ分割）

整合性を後から修復する仕組み

・リードリペア（Read Repair）
全てのノードで複製データの読み出し要求を行い、データに矛盾がないかを確認する。発見された矛盾はバックグラウンド処理で修復する。

・ヒンテッド・ハンドオフ（Hinted Handoff）
故障が発生した場合、他の稼働中のノードに、故障の復旧後の更新手続きを指示する仕組み。

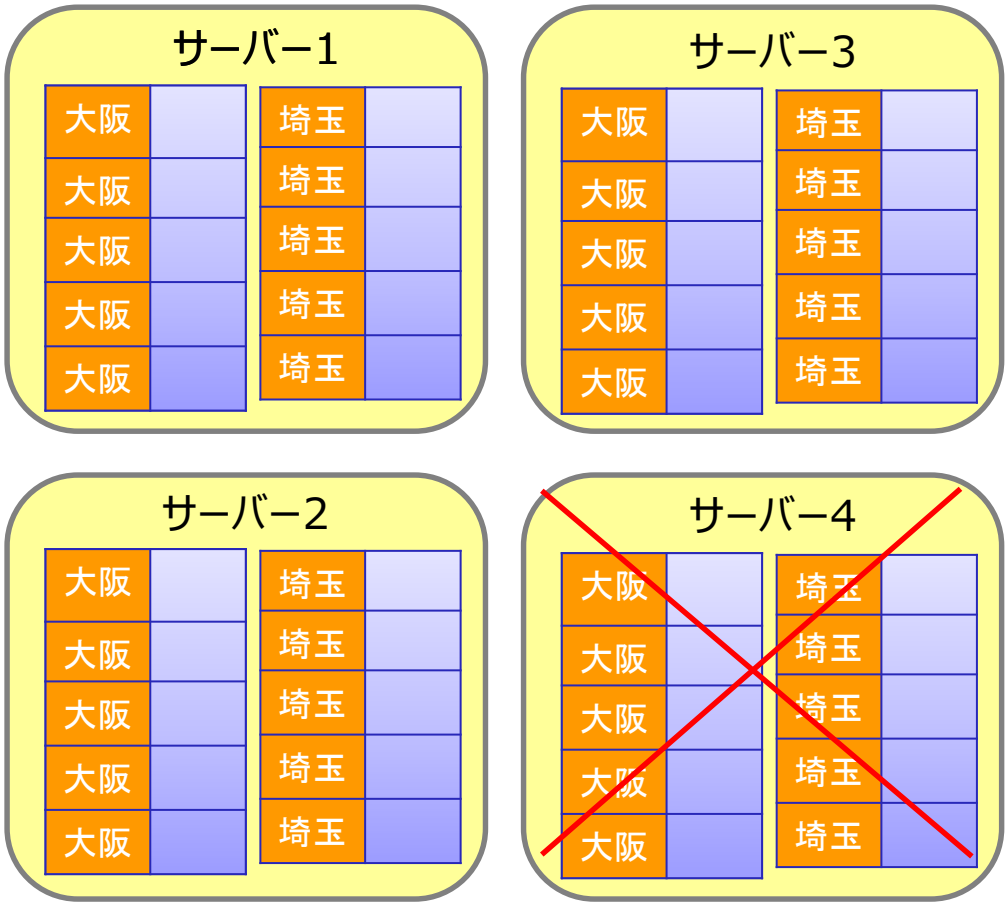
・マークル・ツリー（Markle Tree）
Markle Treeという樹形図構造のアルゴリズムによりデータの整合性を確認し、整合性に問題がある場合は手動での修復を行うもの。



NoSQLの基本的概念と技術（データ分割）

シャーディング（Sharding）

あらかじめ、サーバ毎に保存するデータのキーの範囲を設定しておくことで、検索や管理を容易にする機能です。



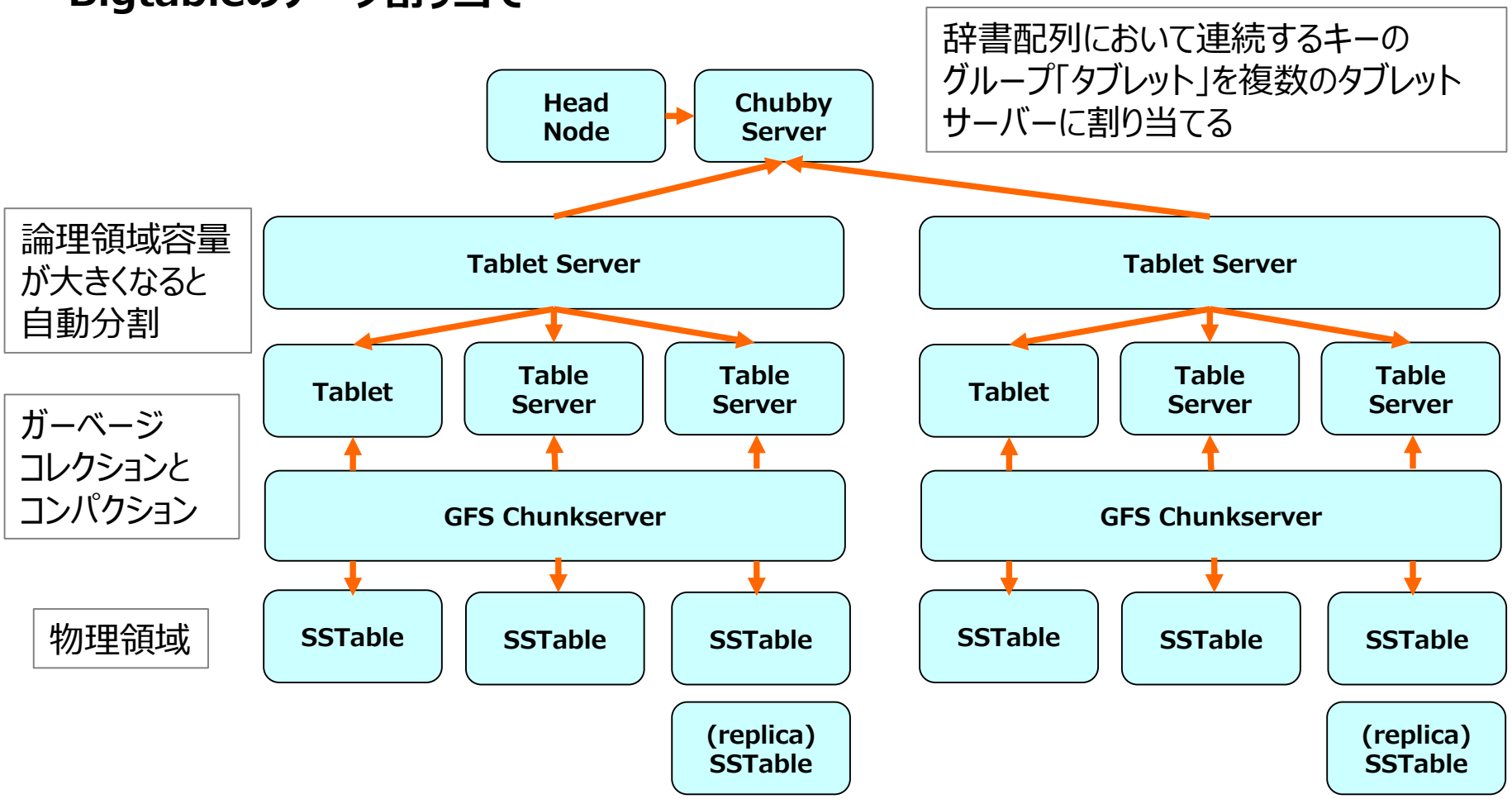
他、アルファベットA～G、H～N、O～Z
ID 1 ～1000、1001～2000…など

データの偏りがあった場合、適切な負荷分散ができません。なので、データ単位を小さくし、ノード数より多くすることなどにより、データの偏りを減らす工夫が必要です。

BigtableやHbaseは再分割、再配置を自動的に行います。

NoSQLの基本的概念と技術（データ分割）

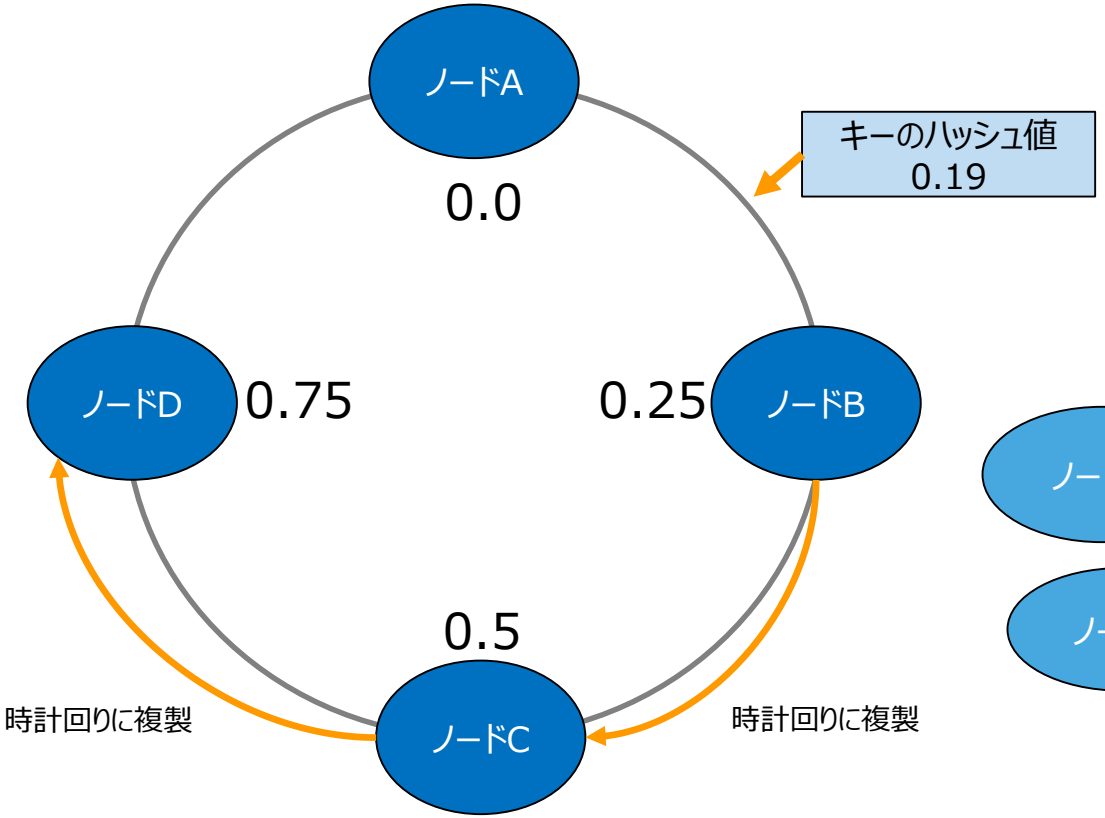
Bigtableのデータ割り当て



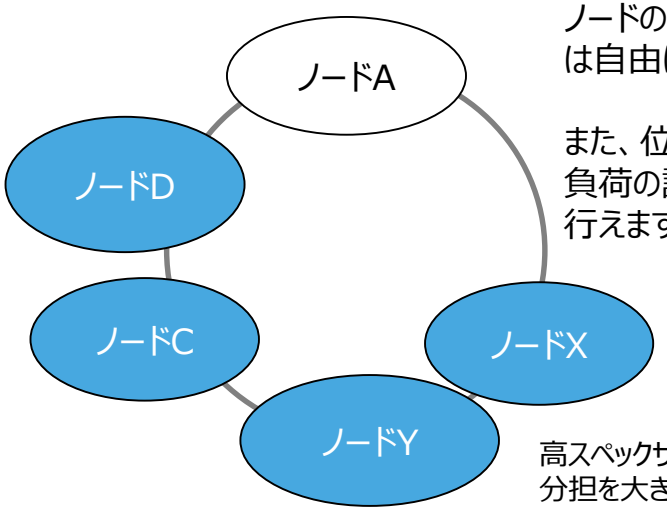
NoSQLの基本的概念と技術（データ分割）

コンシステントハッシング Consistent Hashing

分散型システムを構成するノードを、論理的に輪を形作るよう配置します。データのキーのハッシュ値をハッシュ関数によって求め、不規則な値であるハッシュ値を元に、データを保存するノードを選択します。ノードの位置を変更することで、負荷の代償を変更することも可能です。



例えば、0.0から1.0の間のハッシュ値の場合、一番近い時計回りのノードに割り当てられるようなルールを定めます。



ノードの追加・削除は自由に行えます。

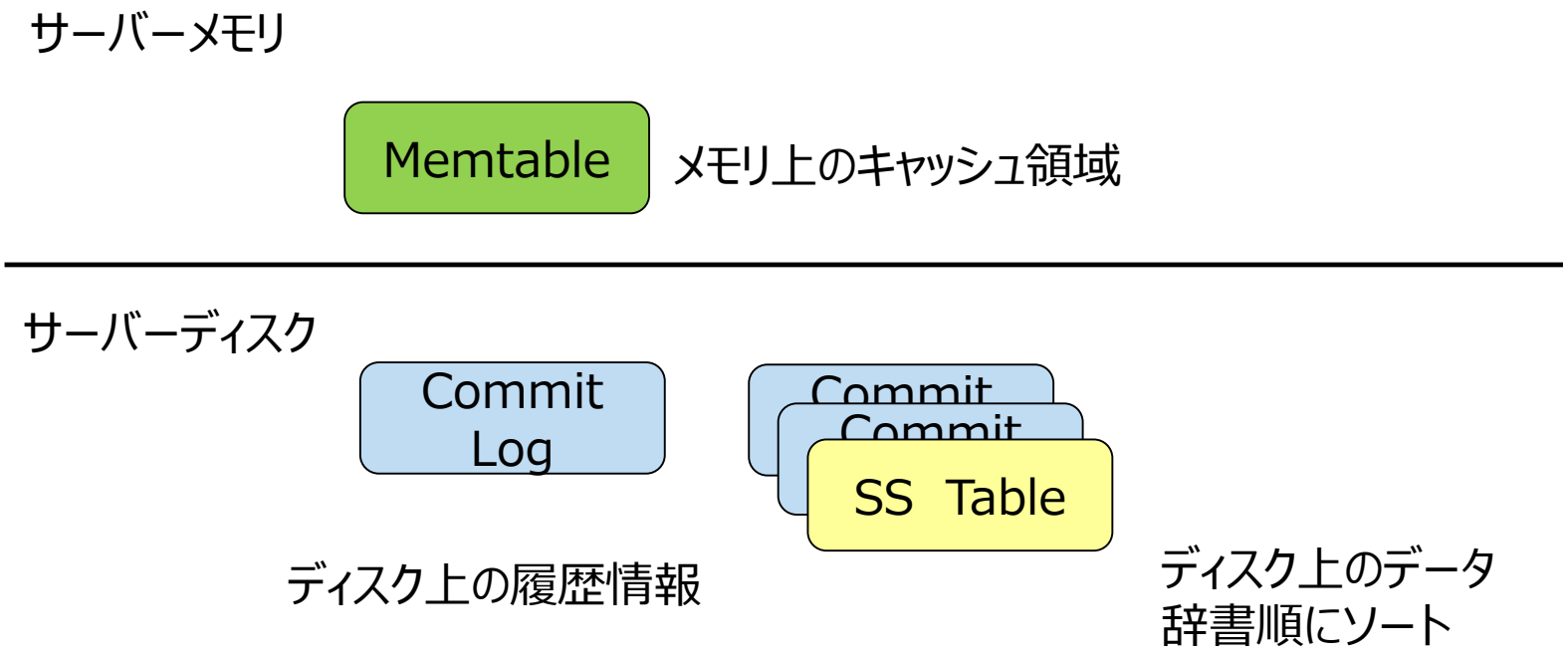
また、位置をずらして負荷の調整も行えます。

高スペックサーバーは分担を大きく

NoSQLの基本的概念と技術

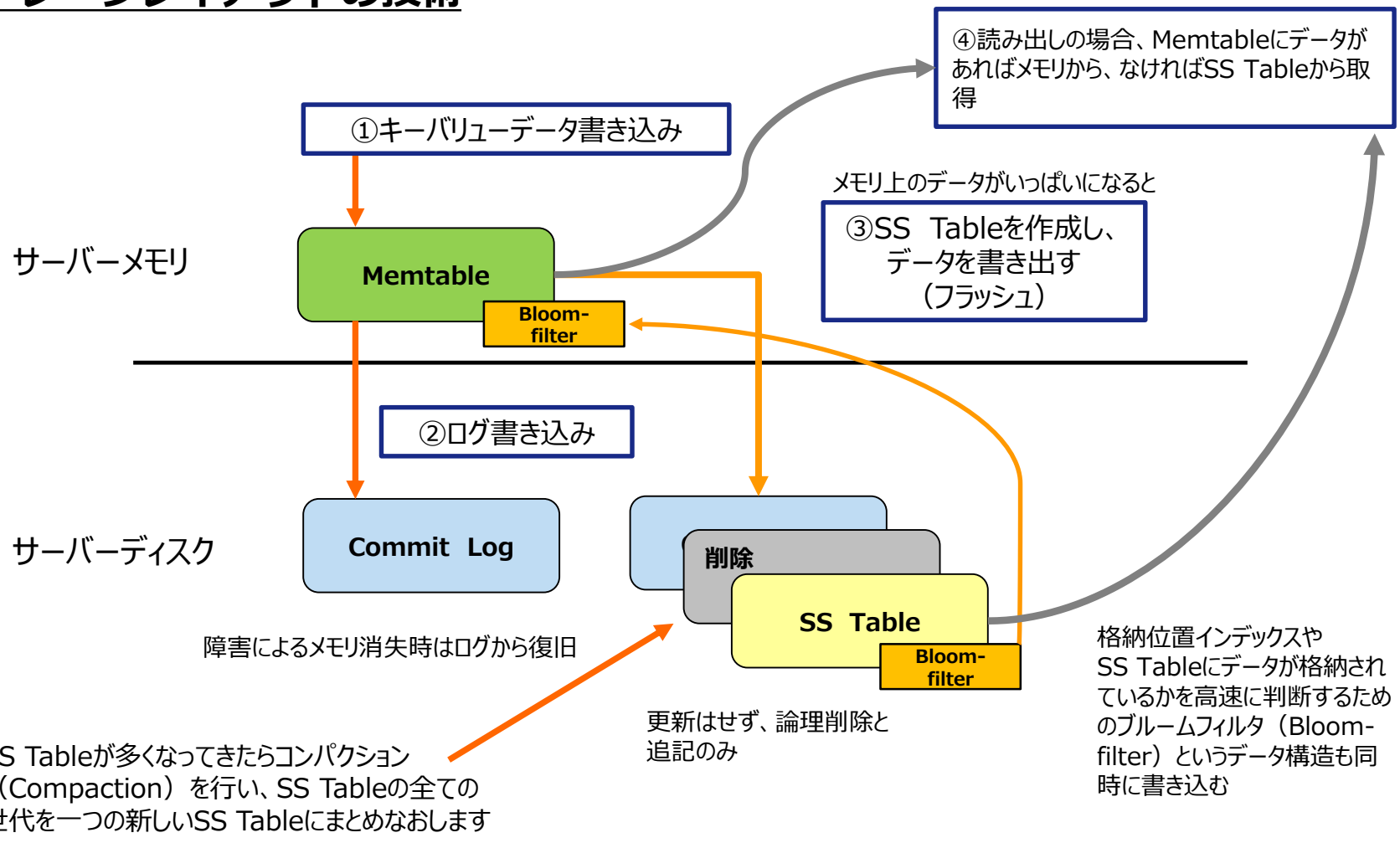
ストレージレイアウトの技術

・LSM-Tree（Log Structured Merge Tree）
メモリを使って応答性能を高めつつ、ハードディスクにデータを格納することでデータの安全性を保つ機能。



NoSQLの基本的概念と技術

ストレージレイアウトの技術



第４章

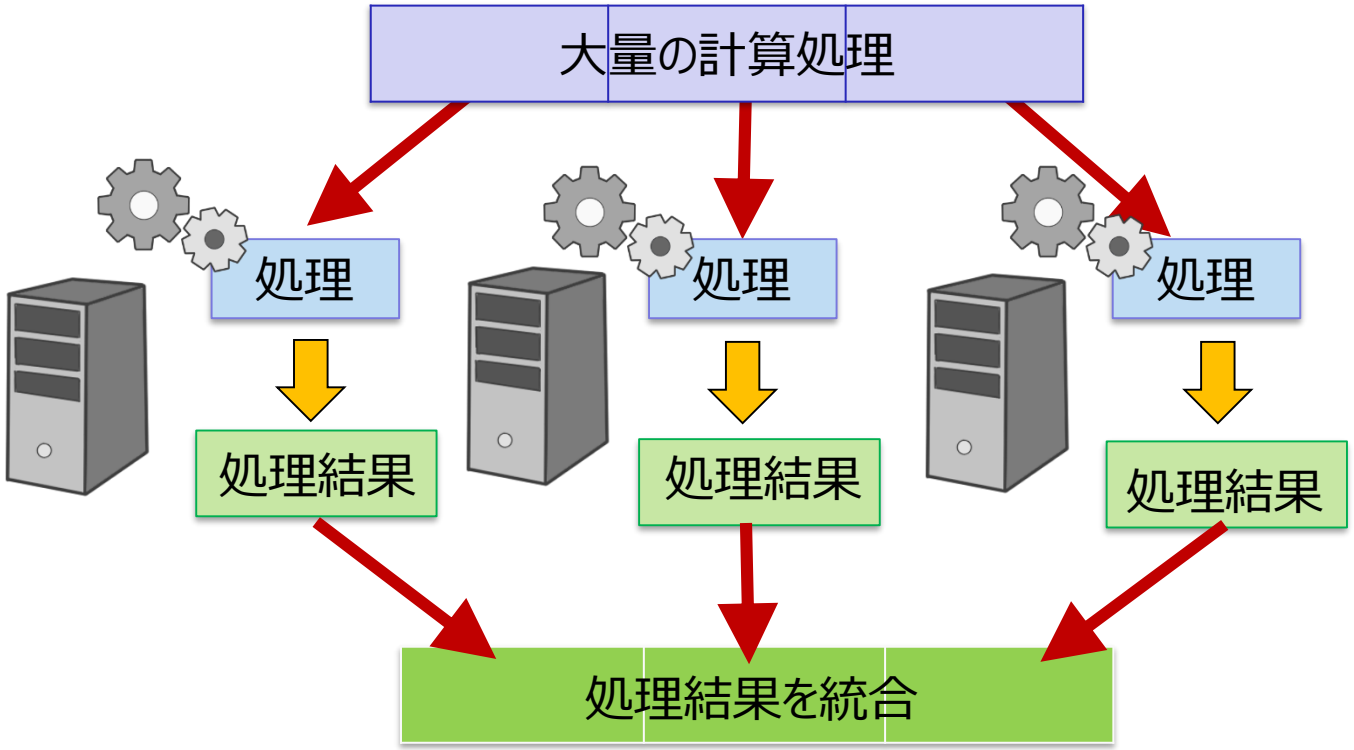
ビッグデータコア技術

分散処理

分散処理とは

分散処理とは

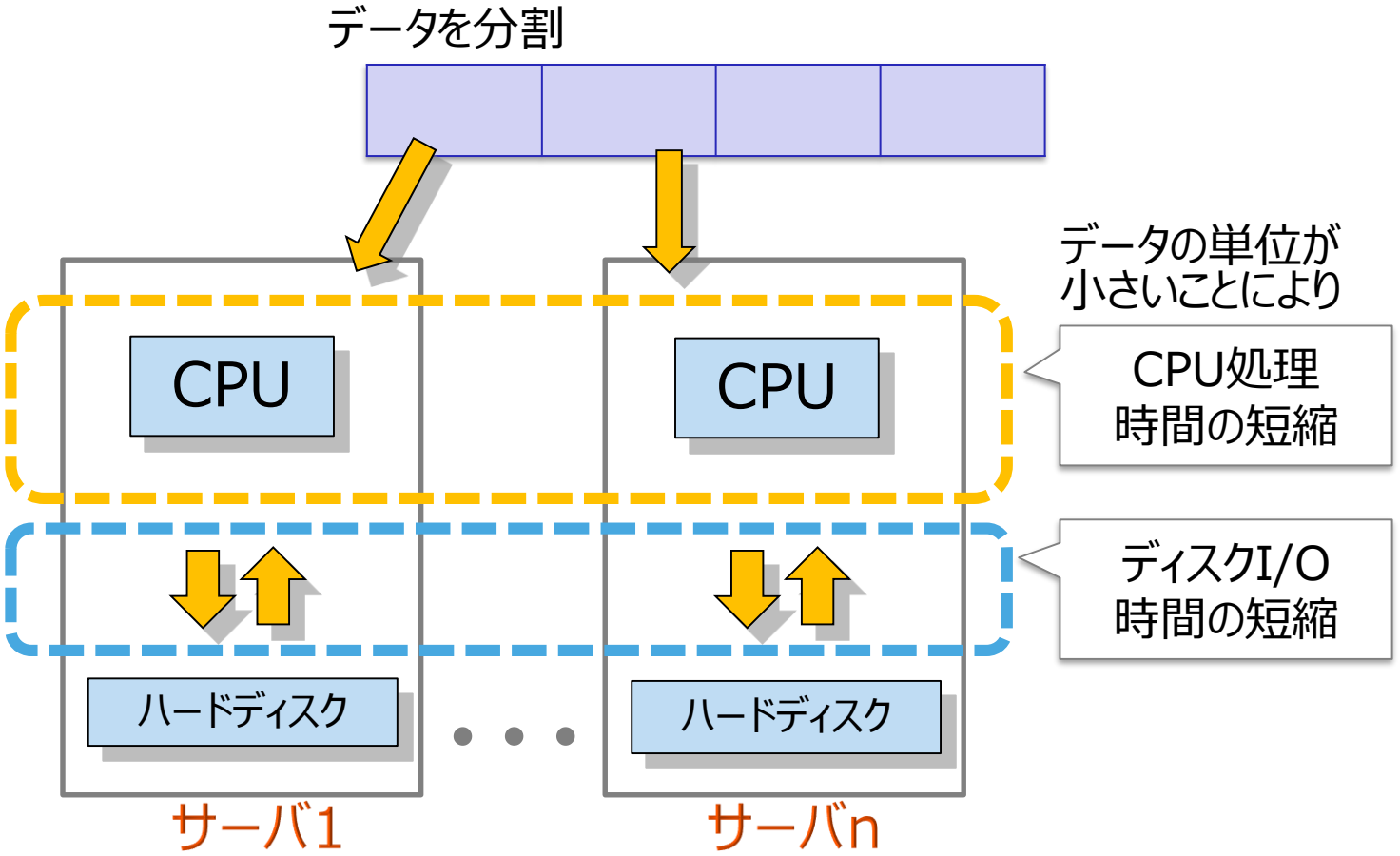
ビッグデータを分割し、複数のサーバで同時に並行して処理した後、処理結果を統合することで全体の処理結果を生成する処理方法です。



分散処理とは

分散処理による処理時間の短縮

サーバでは、分割されたデータを処理するので、CPUでの処理時間や、ディスクからの読み込み、ディスクへの書き込みの時間が短縮されます。



分散処理のメリット・デメリット

分散処理のメリット

ビッグデータの処理時間を短縮することができます。
スケールアウトによる拡張が可能で、比較的低コストになります。
一台の故障が全体の停止には直結しないので、稼働率が高いです。

分散処理のデメリット

データが複数コンピュータにまたがるので、厳密な整合性が必要なオンライントランザクション処理に向きません。
少量のデータ処理には効果が薄いです。
システム全体の性能は、ネットワークの帯域幅や稼働率から大きな影響を受けます。

Hadoopとは

Hadoop

Hadoopはデータベースではなく、複数のソフトウェアからなる分散処理フレームワークです。
一定期間のデータを一括処理する、バッチ向けのフレームワークです。

2つの主要プロダクト

・ファイルシステム(HDFS)：Hadoop Distributed File System
バッチ処理向けに高いスループットを分提供する分散ファイルシステム。

・分散処理アルゴリズム(MapReduce):Hadoop MapReduce
複数のサーバでの並列分散処理するプログラミングのためのソフトウェア。
HDFS上のファイルを読み込み、集計結果を別のHDFS上に書き込む。

Hadoopとは

Hadoopの特徴

オープンソースソフトウェアフレームワークで、非構造化であるビッグデータの解析や大量のバッチ処理に用いられます。 Googleによる開発の後、Apache Software Foundationが現在は開発、公開を行っています。

HadoopはLinuxで動作します。また、システムはJavaで構築されているため、JVMが必要です。

MapReduceフレームワークやHDFSを操作するためのJavaのAPI（クラスライブラリ）が提供されています。

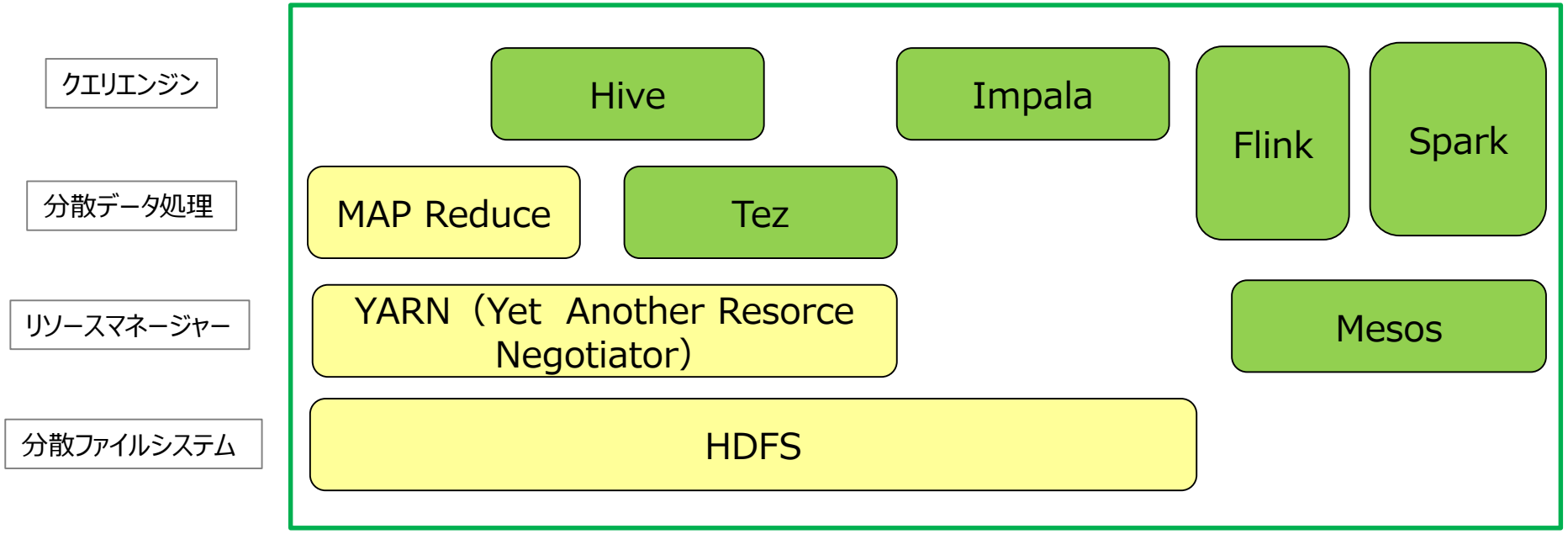
専門スキルが導入に必要なため、商用のサポートサービスを提供している会社があります。 (クラウデラ、ホートンワークス、MapRなど)

Hadoopの基本構成

Hadoopの基本コンポーネントは、

・分散ファイルシステムのHDFS
・リソースマネージャーのYARN
・分散データ処理を行うMapReduce

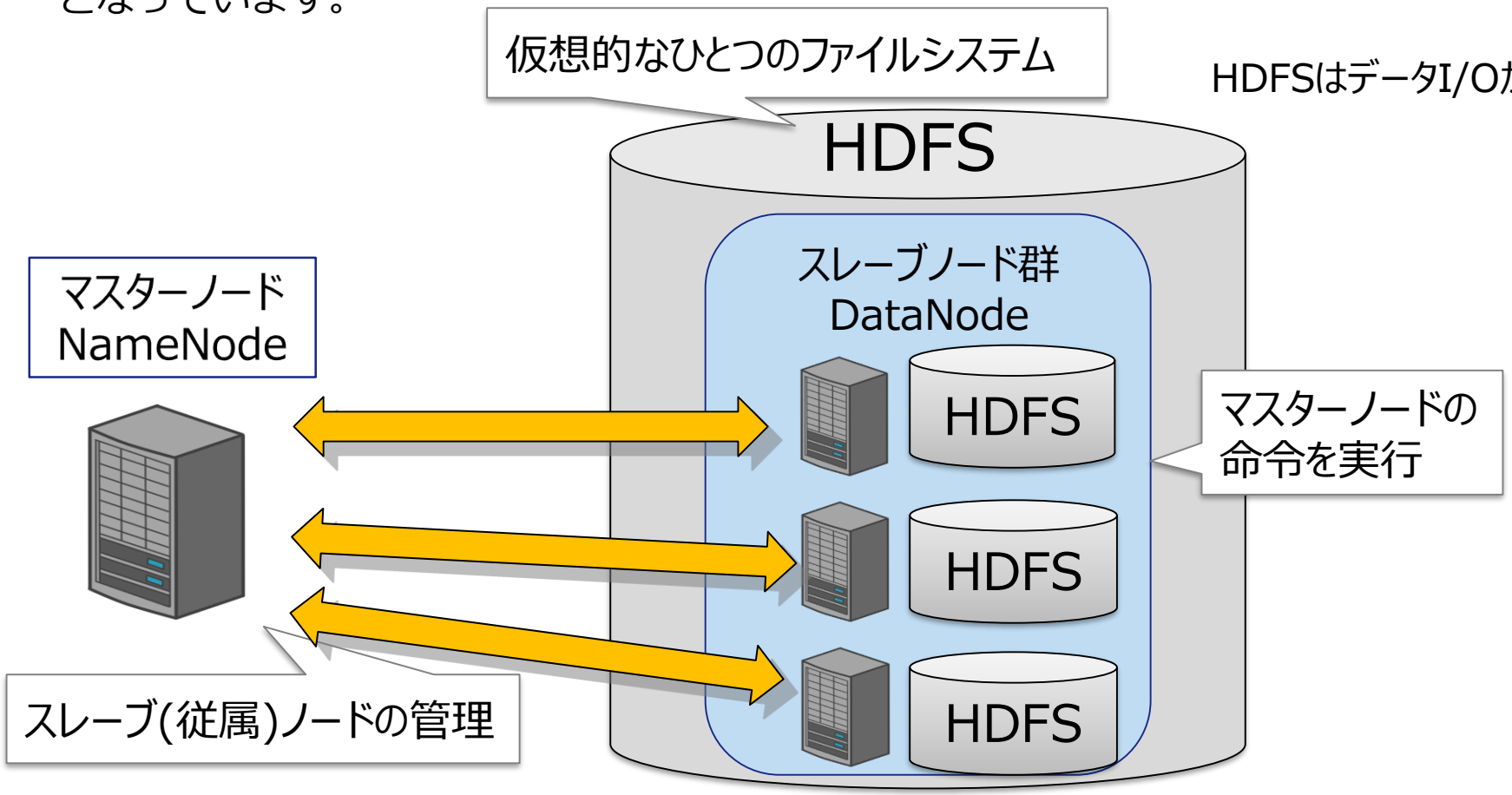
です。その他のシステムは、Hadoopから独立して開発されたものです。



HDFS (Hadoop Distributed File System)

HDFS

HDFSはマスタ型で、データを管理するスレーブノード群と、スレーブノードを管理するマスターノードから構成されています。全体で、仮想的な一つのファイルシステムとなっています。



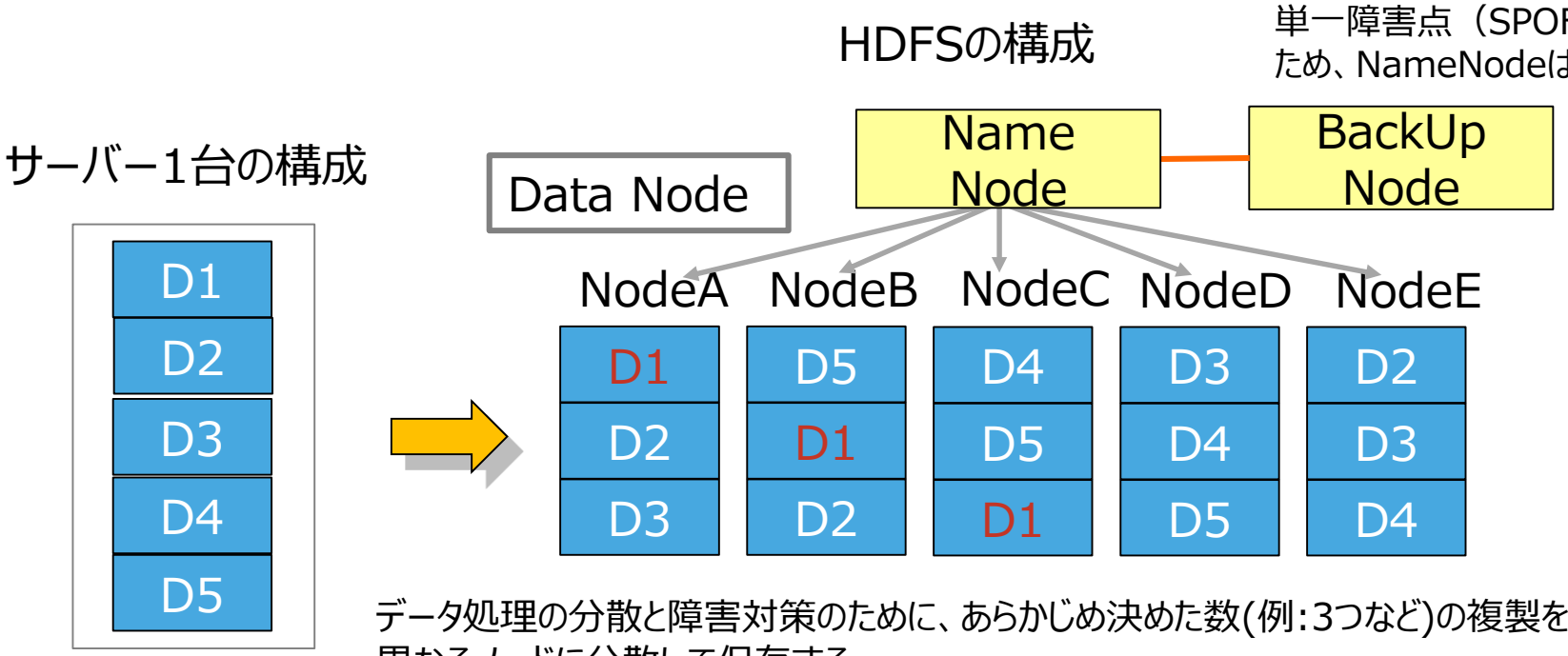
HDFSはマスタ型

マスタースレーブ型

NameNode、DataNodeに役割が分かれます。

NameNodeはメタ情報(他のノードのデータやブロックの対応関係など)を保持するマスターノードです。単一障害点となるため、バックアップノードを作成します。

DataNodeがデータ(ブロック)を保持します。障害対策として、同じデータを複製し、ノードを跨って保存されます。



データ処理の分散と障害対策のために、あらかじめ決めた数(例:3つなど)の複製を作成し、異なるノードに分散して保存する。
大きなサイズのファイルは任意のブロックサイズに分割して格納。（デフォルト64MB）

HDFSの読み書き

HDFSへの書き込み・読み出し

データは、クライアント側で任意のサイズのブロックに分割され、NameNodeがどのDataNodeに格納するか指示し、書き込みを行います。
読み出しは、 NameNodeがクライアントに対して目的のデータが保存されているDataNodeを示し、直接ブロックを読み出します。

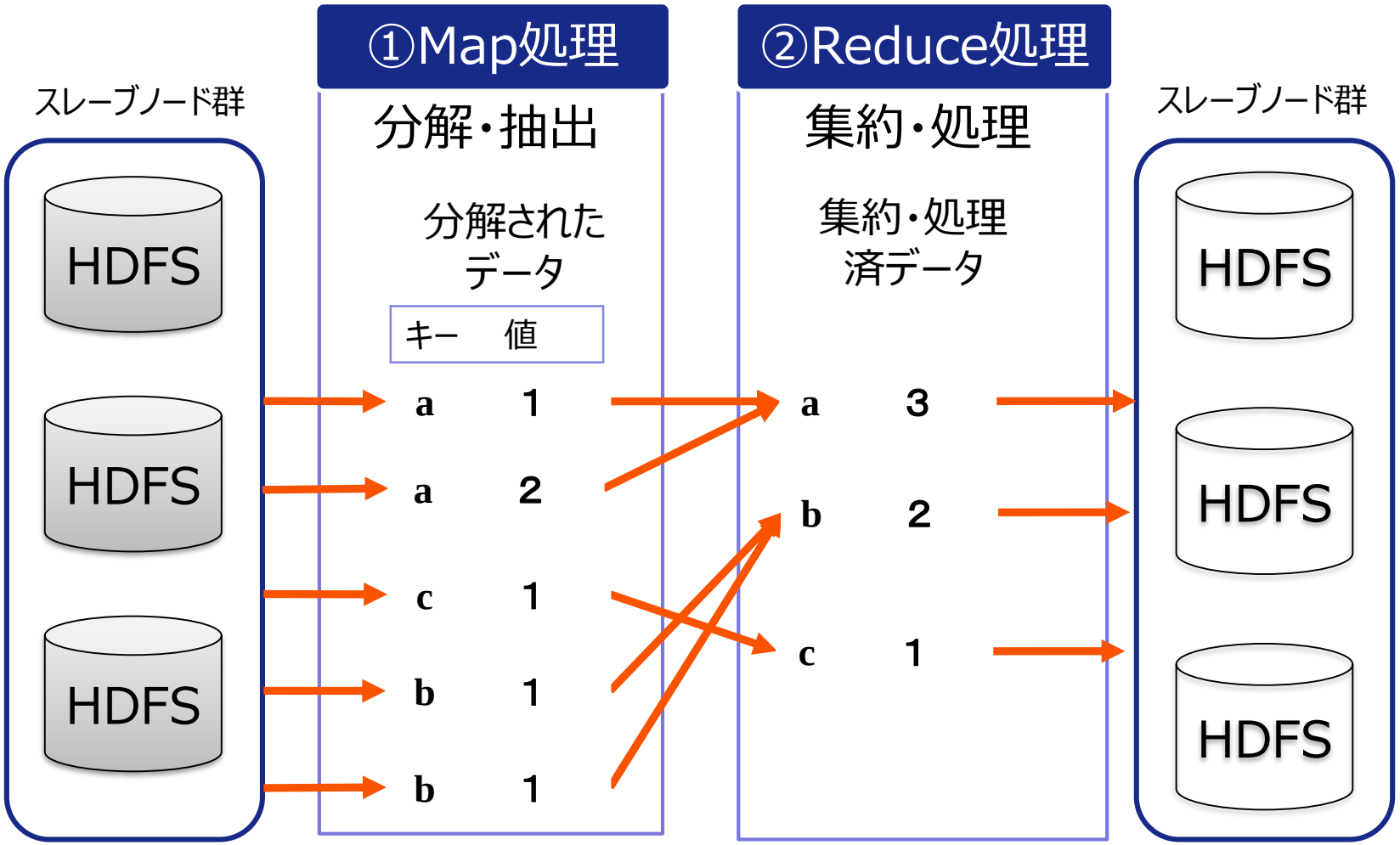
データの複製と耐障害性

データ処理の分散と障害対策のために、一定数複製を作成し、複数のノードにまたがってデータを保存します。障害時は、障害のあるノードをクラスタから切り離し、切り離されたノードにあった複製の分を、他のノードに複製します。

リバランシング

ノードの追加や削除された場合に、データの分散に偏りが生じないようにデータの再配置を行います。

MapReduce処理とは



MapReduce処理とは

MapReduce

大量データを分割し、複数のサーバで並列処理するための方法です。複数のスレーブノードで同一の処理を実行します。次の2段階の処理に分かれます。

①Map処理

データを小さく分割し、必要な情報を選んで出力します。出力データはキーバリュー型の構造となります。

②Reduce処理

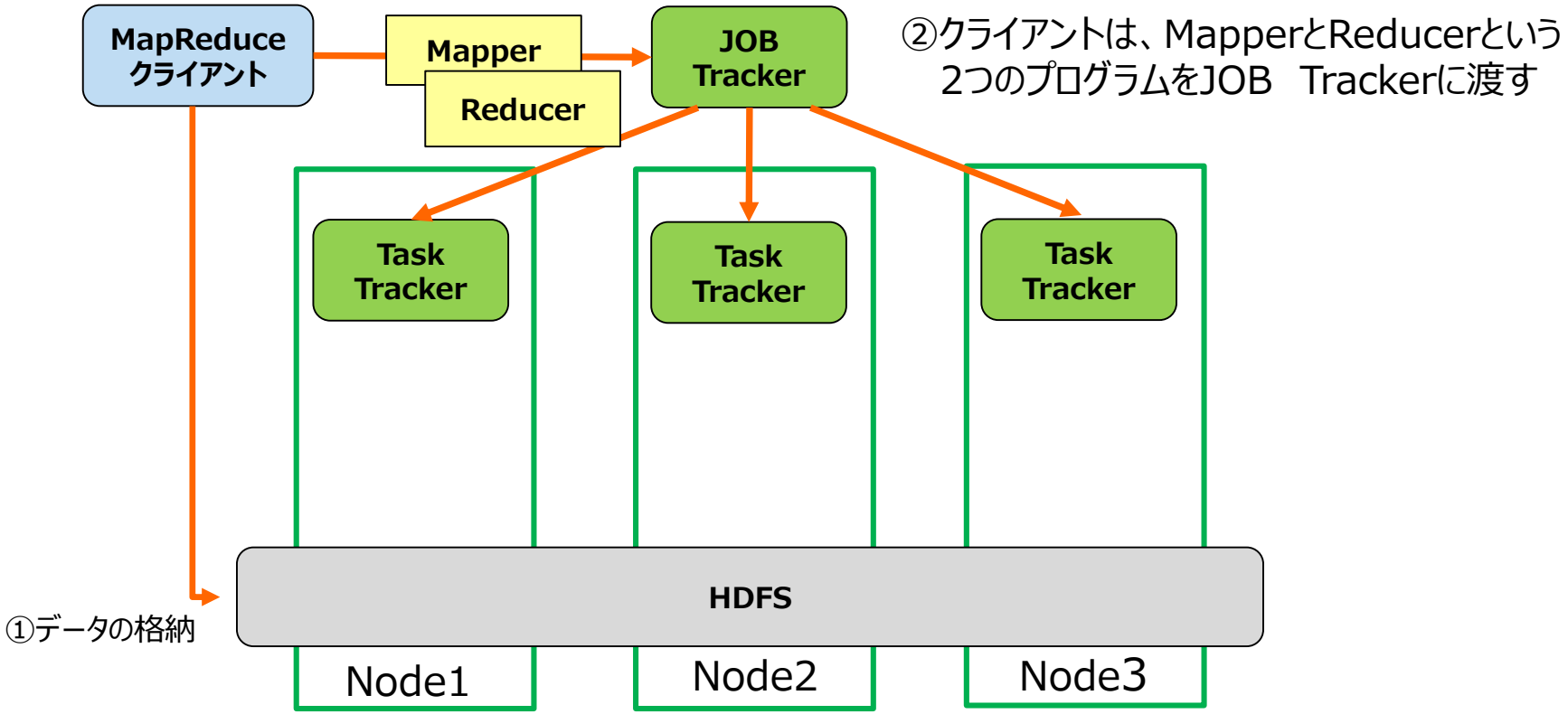
キーごとにデータを集約し加工を行なった上で、HDFSに出力します。

Map処理とReduce処理の開発には、Hadoopが提供するJavaのAPI(クラスライブラリ)を利用します。複雑な分散処理アルゴリズムや障害対応処理などの実装は不要で、開発者は業務ロジックの開発に集中できます。

MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

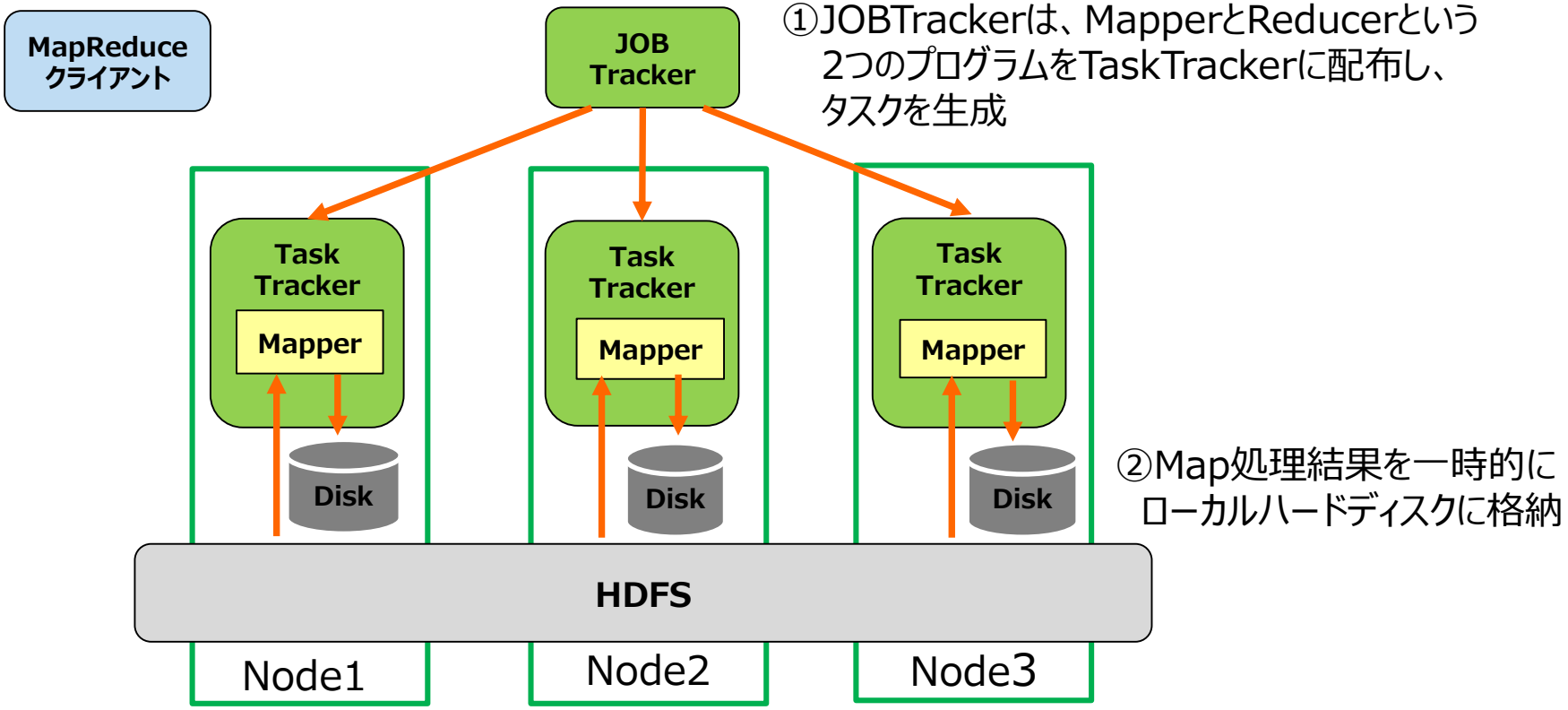
MapReduceのマスターはJobTrackerと呼ばれ、全てのMapReduce処理工程の調整・管理とスレーブとなるTaskTrackerの維持管理を行います。



MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

JobTrackerはMapperとReduceプログラムをTaskTrackerに配布し、HDFSブロックと同じ数だけMapタスクを生成します。 Mapタスクは、データが存在するTaskTrackerに割り当て、必要なデータ転送のみを行います。

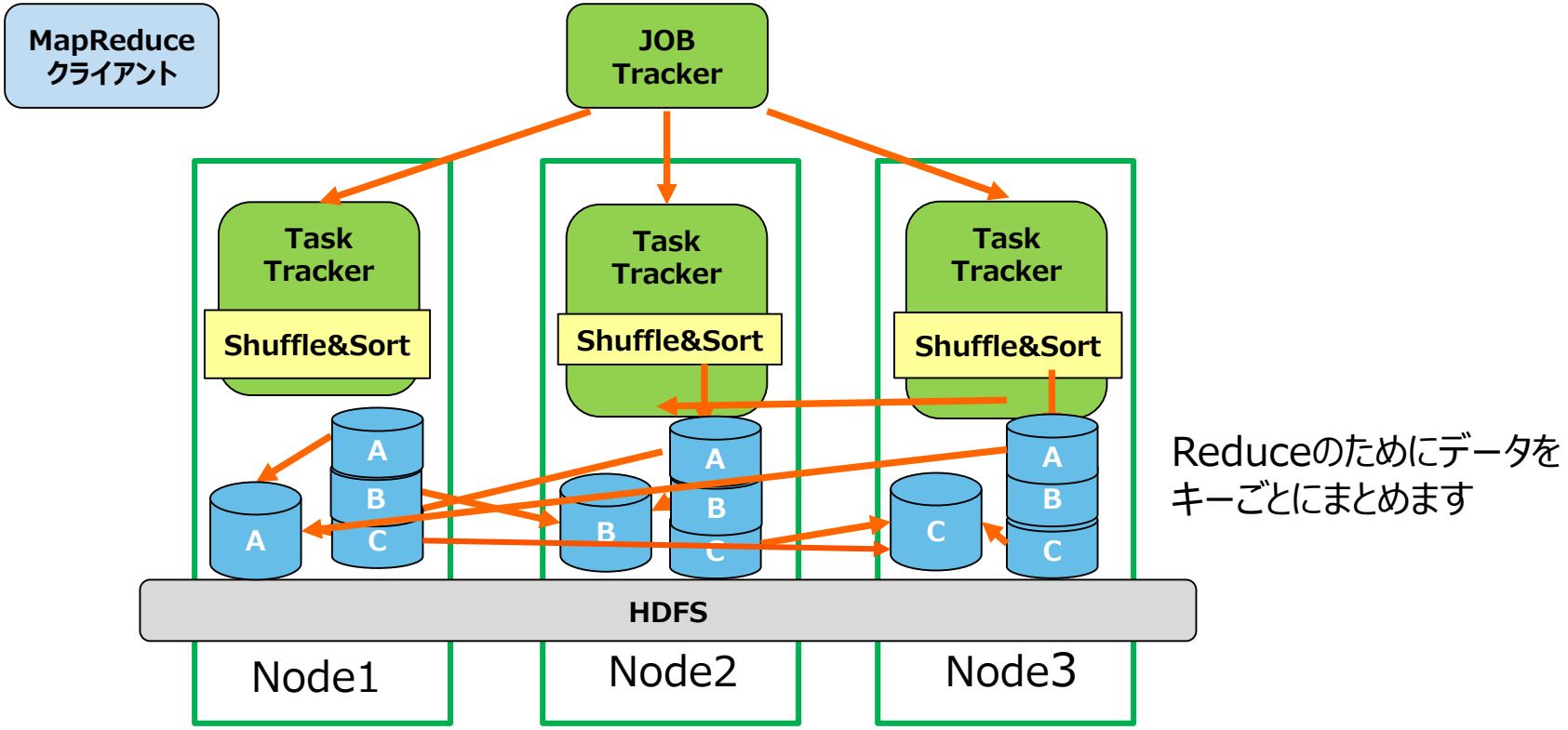


MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

JobTrackerは、ある程度処理が進むとShuffle&Sort処理を起動します。

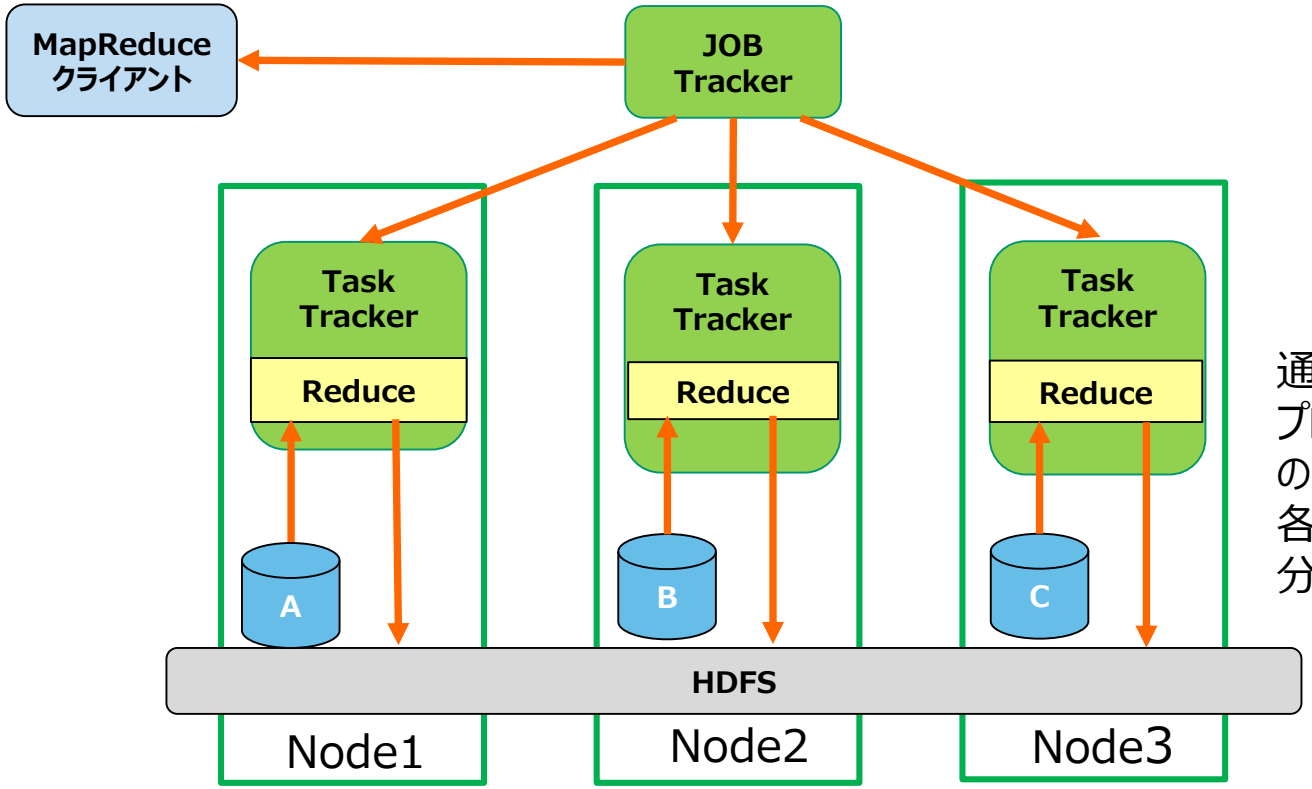
Mapタスクにより処理された中間データを、キーの値よって、ノードを跨っているものを含めて1箇所にグループ化します。



MapReduceのアーキテクチャ

MapReduceのアーキテクチャ

処理がさらに進むとJobTrackerはReduceタスクをTaskTrackerに割り当てます。TaskTrackerはReduceタスクを実行し、処理結果をHDFSに書き込みます。全てのタスクが終了するまで繰り返し行います。



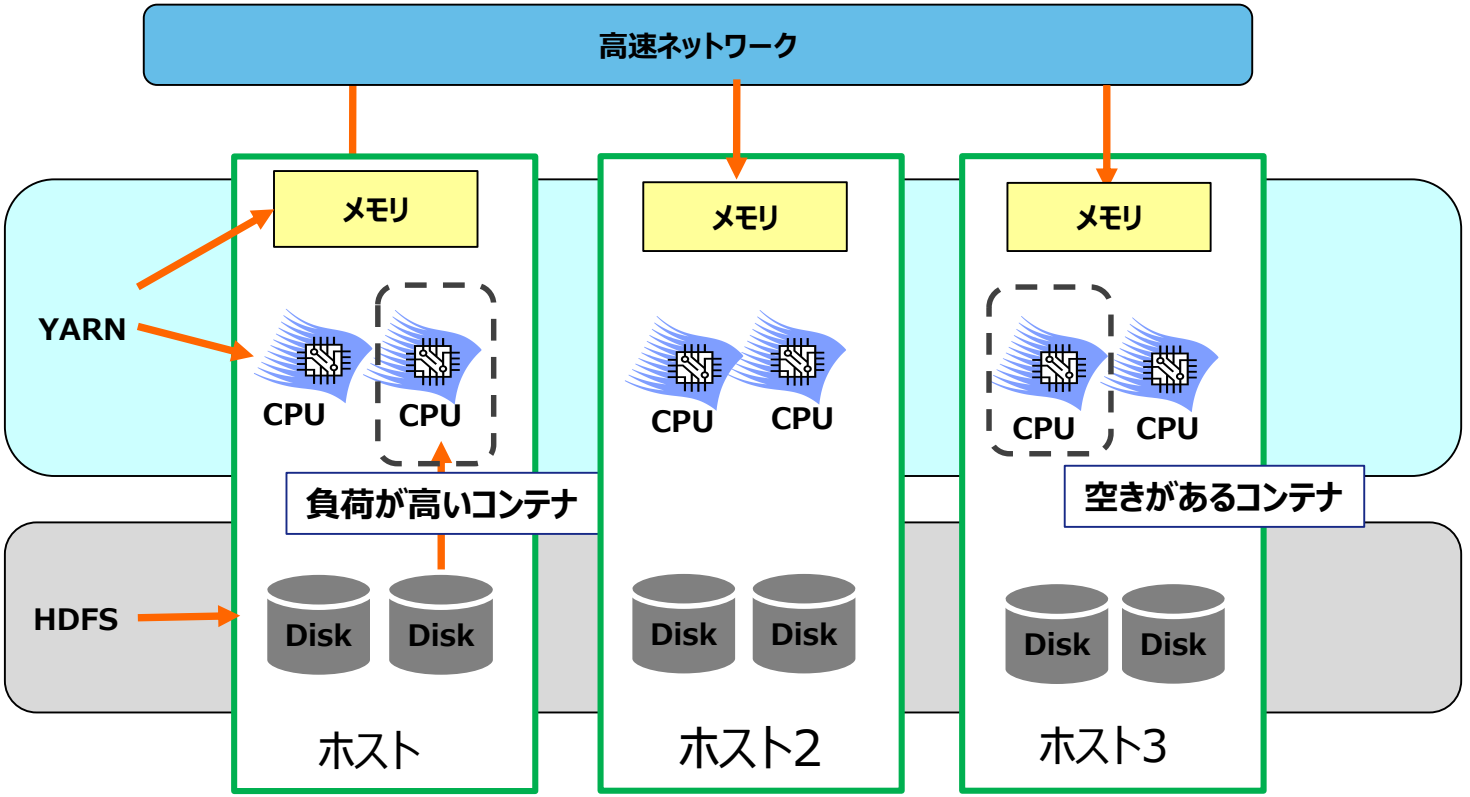
通常のプログラミングでは、プログラムがデータを呼び出すのですが、プログラムの方を各ノードに配布している点が分散処理のポイントです。

分散ファイルシステムとリソースマネージャー

YARN

CPUやメモリなどの計算リソースはリソースマネージャーであるYARNにより管理されます。
CPUコアやメモリをコンテナと呼ばれる単位で管理します。

分散アプリケーションが実行された時に、クラスタ全体の負荷のバランスを見てコンテナが割り当てます。

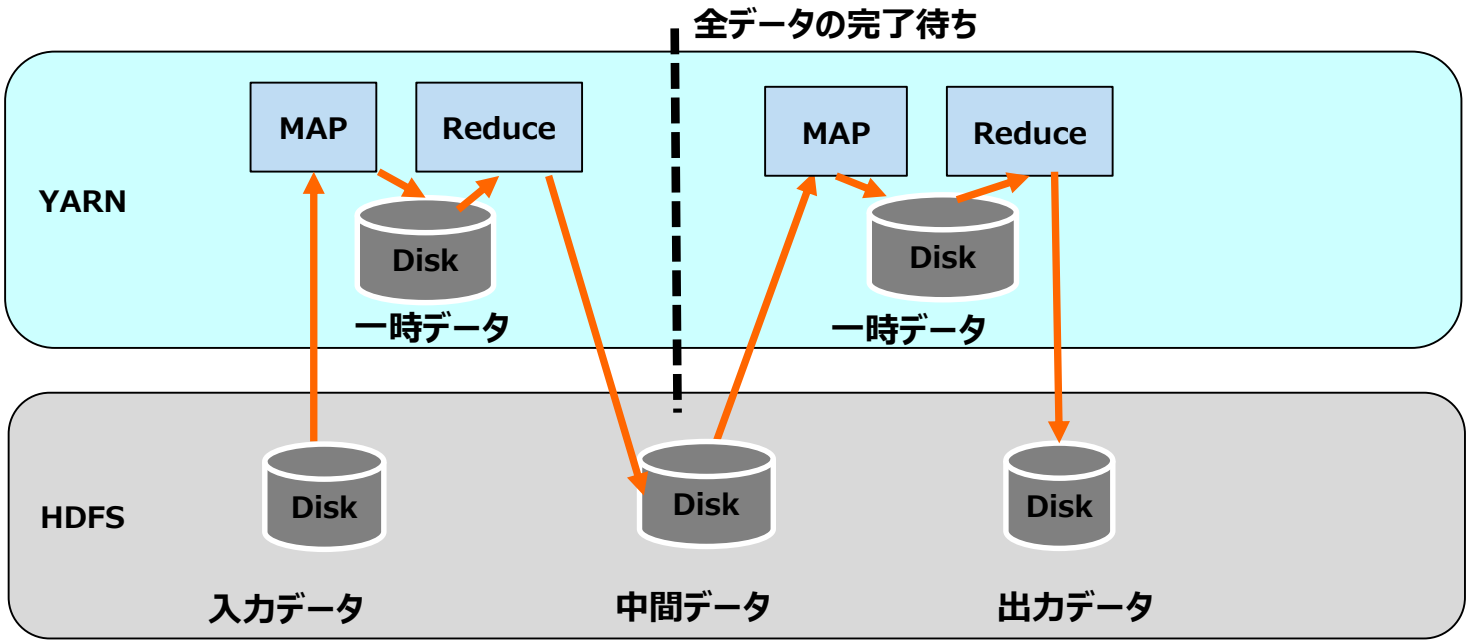


分散データ処理とクエリエンジン

Hive on MapReduce

MapReduceはJavaプログラムをデータに対し実行できるため、加工には便利です。SQLに対応させる場合、専用に設計されたクエリーエンジンApache Hiveを利用します。

Hiveは応答性能が重要視される場合や少量データを扱う場面などではには不適です。



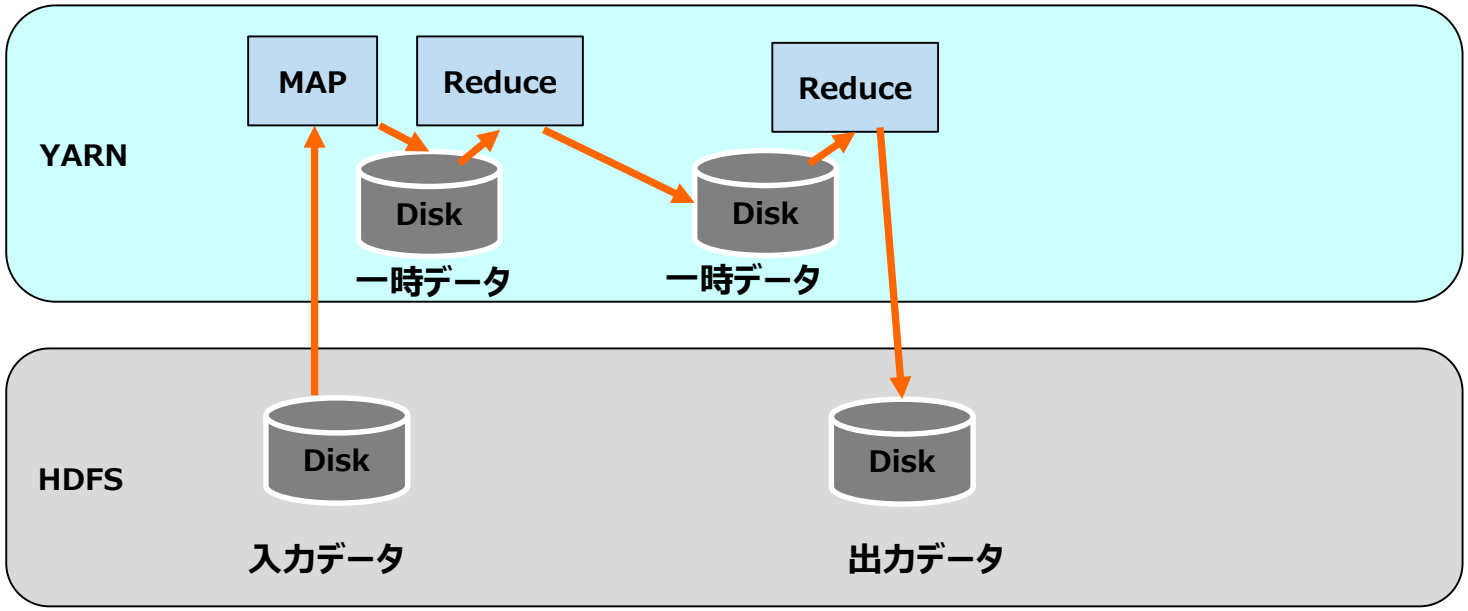
データ処理ステージが変わるタイミングで待ちが発生。
中間データを毎回Diskに保存するため低速

分散データ処理とクエリエンジン

Hive on Tez

Apache TezはHiveの高速化するために開発されたコンポーネントです。
Mapreduceの仕組みを調整し、処理段階の終わりで他のプログラムの情報を保持せずに後続の処理へと受け渡す仕組みに変更しました。

ステージの合間にあったディスクへの書き込み、待ち、Map処理を省略

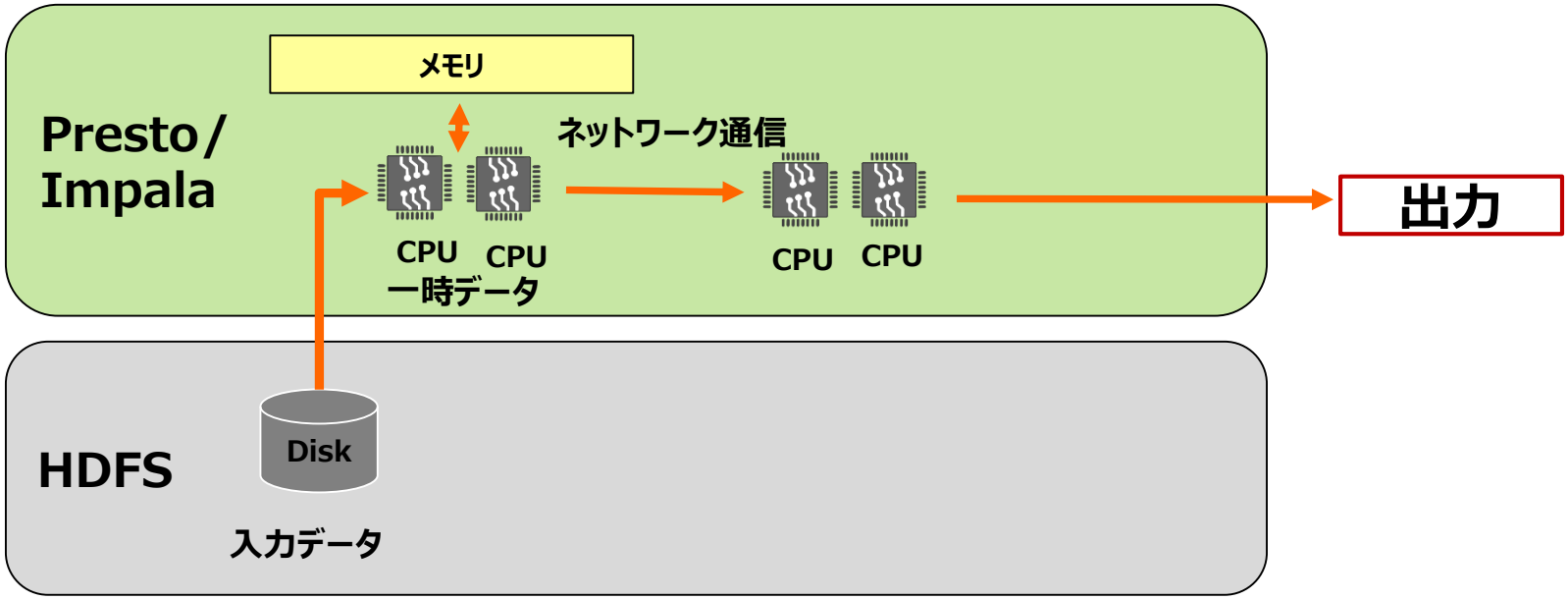


現在、Hive on Sparkというものも存在します。RDDがMapreduceに代わる存在として活用されています。

分散データ処理とクエリエンジン

Impala / Presto

ImpalaとPrestoは、Hiveの高速化ではなく、対話型クエリ実行に特化させることで、SQLに対応させた製品です。バッチ処理ではなく、最大瞬間速度を最大にするため、あらゆるリソースを有効活用するよう設計されています。

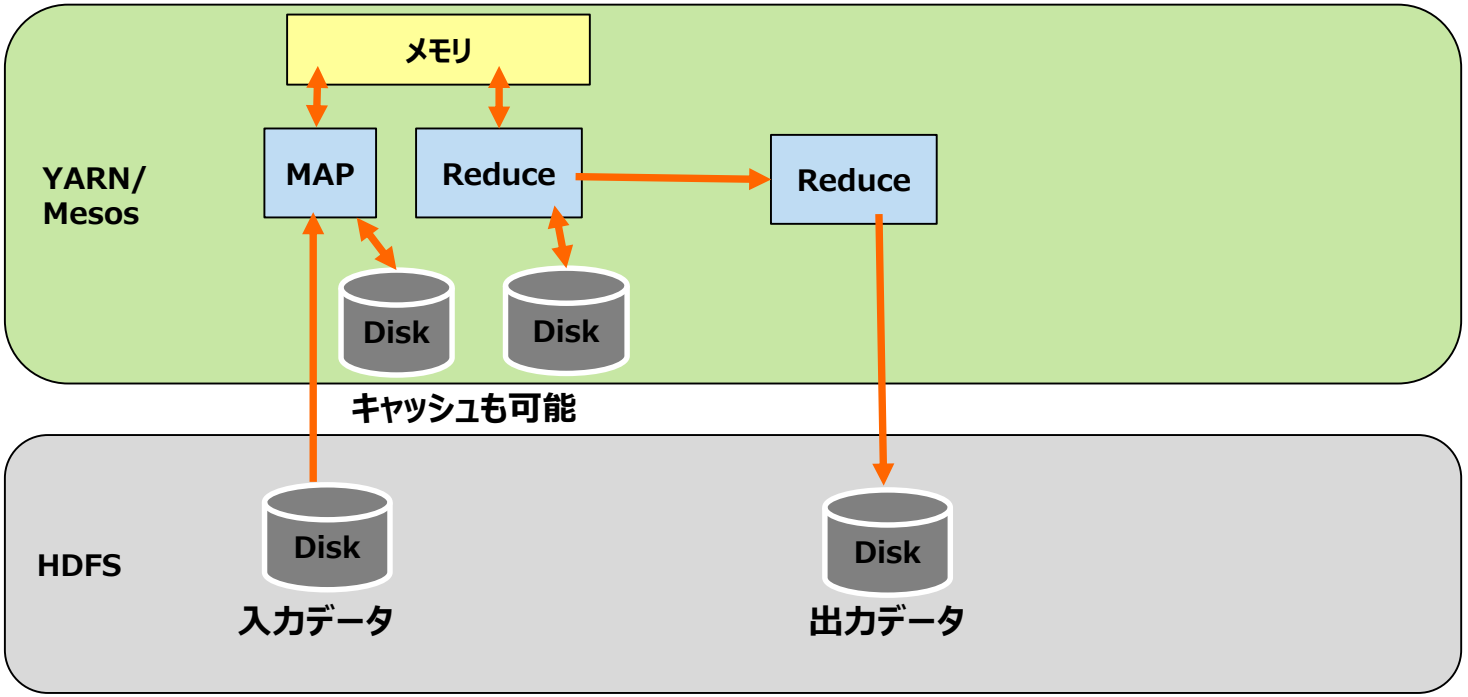


自前の分散処理機構を実装し、マルチコアを活用しながら並列化処理を実行

分散データ処理とSpark

Spark

Apache SparkもMapReduceより効率良くデータ処理をするために開発されました。メモリの使用量増やすことで高速化を実現します。Sparkの実行にはJavaランタイムが必要ですが、Java、Scala、Python、Rからも呼び出すことができます。

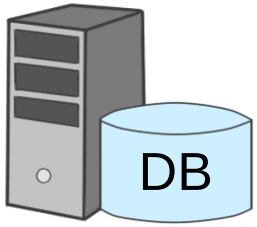


Sparkは中間DBにデータを書き込むことなくメモリ上に保ち続けます。
障害でデータが失われるともう一度最初からやり直します。

第５章

ビッグデータにおける データ解析

ビッグデータにおけるデータ解析



社内の実績データ

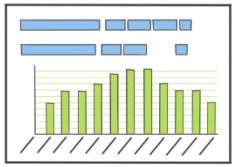
SNSデータ
社外の取引データ
Webサイトログ

音声・画像・
センサーデータ

オープンデータ・
統計データ・
科学データ

ビッグデータ
蓄積基盤

BIツール



統計解析
ツール

機械学習・
AI

収集

蓄積/集計

解析/可視化

アドホック分析ツール

データを可視化するためのソフトウェアは多種多様に存在します。

アドホック分析ツール

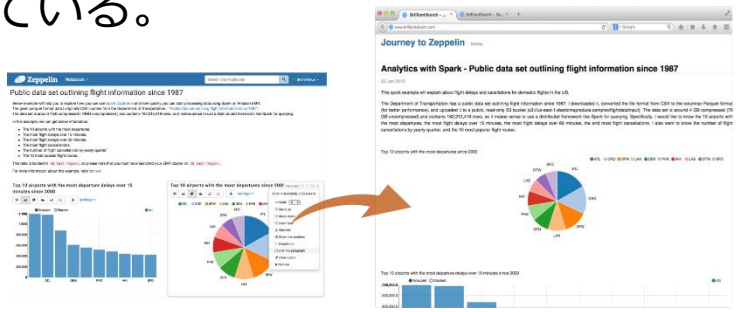
行錯誤を繰り返しながら、繰り返しデータを見ていく際に利用するものです。
アドホック分析環境には次のようなものがあります。

・Jupyter Notebook

Python、Ruby、Rなどのソースを書き留めながら実行することが可能。
matplotlibライブラリを利用してグラフを作成することもできる。

・Apache Zeppelin

Spark/Hadoopなどの分散処理システムに対してコードを実行し、
実行結果をグラフとして描画することが可能。
シェル／SQL／scala／python言語に対応している。



https://zeppelin.apache.org/

ダッシュボードツール

ダッシュボードツール

傾向の把握が進むと、定期的にクエリを実行し、レポートやグラフなどを用いたダッシュボードを作成するツールです。

対話的なBI商用ツールとしてTableau、ClikViewなどが存在します。
オープンソースのダッシュボードツールも存在します。

- Redash
  Python製のダッシュボードツール。
  SQLクエリの実行結果を直感的に可視化できる。搭載したDBに結果を記録するため、表示自体は高速だが、クエリ結果が大規模になるとエラーや遅延が生じる。
- Superset
  Python製のWebアプリケーションで対話的なダッシュボードを作成する。
  集計は外部ストレージ「Druid」で行い、リアルタイムな出力を行える。
- Kibana
  JavaScript製の対話的可視化ツール。
  Elasticsearchのフロントエンドでよく用いられる。

データ活用：一般的な統計分析手法

度数分布とヒストグラム	データを範囲に区切って表したもので、データの分布を知ることができる
平均と標準偏差	状況を特定することができる
正規分布	統計的なデータを元に、可能な予測を立てることができる
標本調査（全体像の推測）	標本（一部のデータ）を元に、全体像を把握できる
標本平均	標本の平均を求めることで、全体の平均を推測できる

度数分布とヒストグラム

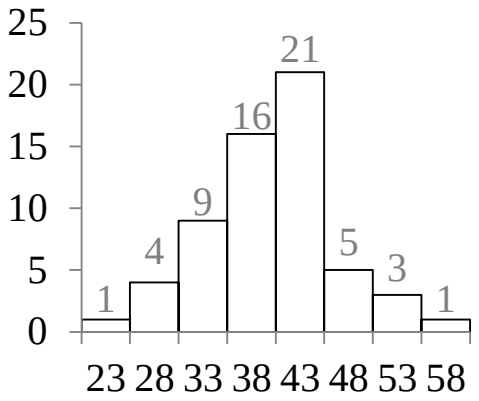
度数分布

度数分布表を用いて表します。
※度数分布表：データをいくつかの範囲に分け、その範囲に入るデータの個数を書き表した表。
データがどのような値を中心にして、どのように広がっているのかを調べることができます。

値）	人数
21～25（23）	1
26～30（28）	4
31～35（33）	9
36～40（38）	16
41～45（43）	21
46～50（48）	5
51～55（53）	3
56～60（58）	1

ヒストグラム

ヒストグラムとは、データの分布状態（度数分布）を表す棒グラフのことです。
度数分布の他の表現方法としては、箱ひげ図があります。



平均と標準偏差

・平均：データの代表値としてよく利用される。
　　平均の代わりに、最頻値、中央値を用いることもある。

$$\text{平均値} = \frac{\text{データの値の総計}}{\text{データ数}}$$

・分散：データがどのくらいばらついているかを表す。

$$\text{分散} = \frac{(\text{各データの値} - \text{平均})^2 \text{の総計}}{\text{データ数}}$$

… 平均との差の２乗の平均

・標準偏差：そのデータの傾向や性質を把握するために利用される。
　・異なるデータ間でのデータのばらつきの度合いを比較できる
　・平均値からのばらつきの幅を測定できる。

$$\text{標準偏差} = \sqrt{\frac{(\text{データの値} - \text{平均})^2 \text{の総計}}{\text{データ数}}}$$

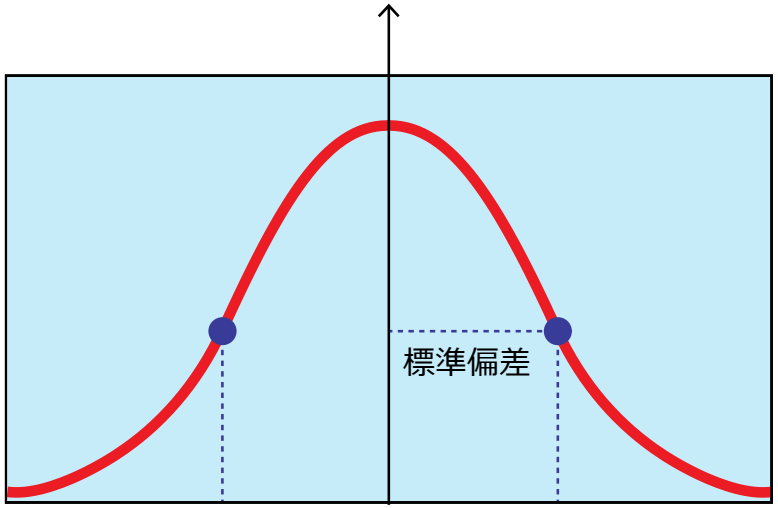
… 分散の平方根

正規分布

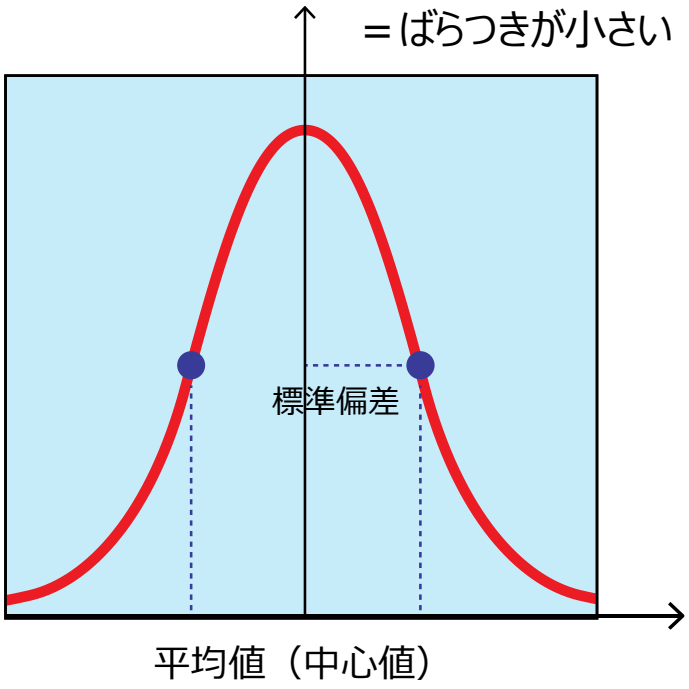
正規分布とは

- 確率分布の１つ。
- 平均値（または中央値）を中心に左右対称な、釣鐘型の分布。
- 平均値で最も発生確率は高くなり、平均値から離れるほど低くなる。
- 正規分布をしていると想定できるデータに関しては、平均値と標準偏差さえわかれば、どのように分布が広がっているかを推定できる。

標準偏差が大きい＝ばらつきが大きい



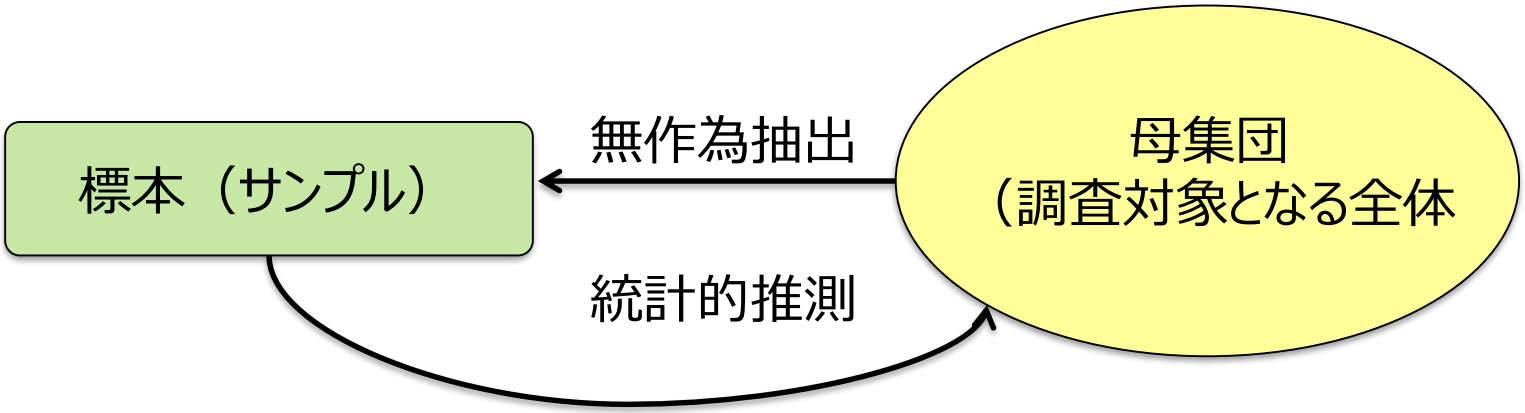
標準偏差が小さい
＝ばらつきが小さい



標本調査と標本平均

標本調査とは

標本、つまり大量のデータ（母集団）の一部を入手、観測した際に、その母集団を推測するために仮説検定の考え方を用いて行う統計的推定の手法のことを言います。



標本平均とは

標本データの平均値を利用することで、母集団の平均を推測することができます。母集団の傾向を推測するために利用されます。

…（標本平均）＝（抽出した標本データの合計）÷（抽出した標本データ数）

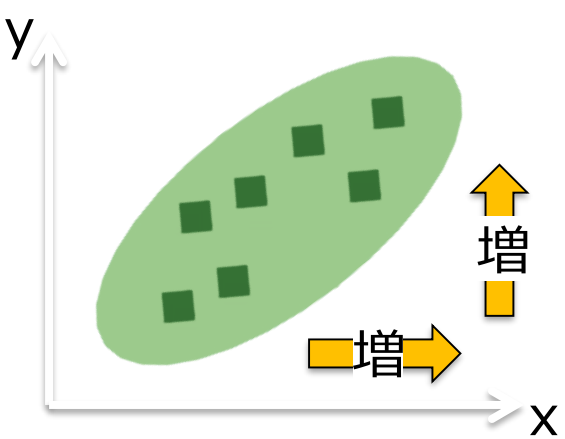
相関関係

相関関係とは

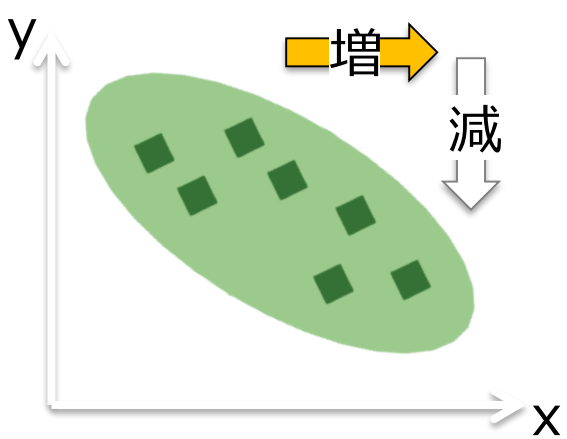
一方の値の変化に伴って、もう一方の値が変化するという、２つの値の関係のことを相関関係と言います。

・正の相関：２つのデータのうち、一方の値が増加すると、もう一方の値も増加するような関係。

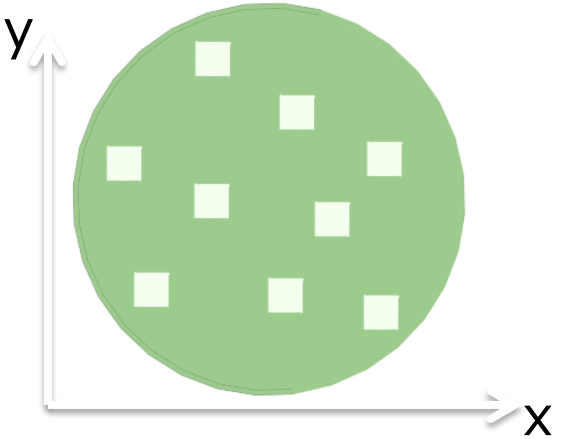
・負の相関：２つのデータのうち、一方の値が増加すると、もう一方の値は減少するような関係。



正の相関（右上がり）



負の相関（右下がり）



無相関

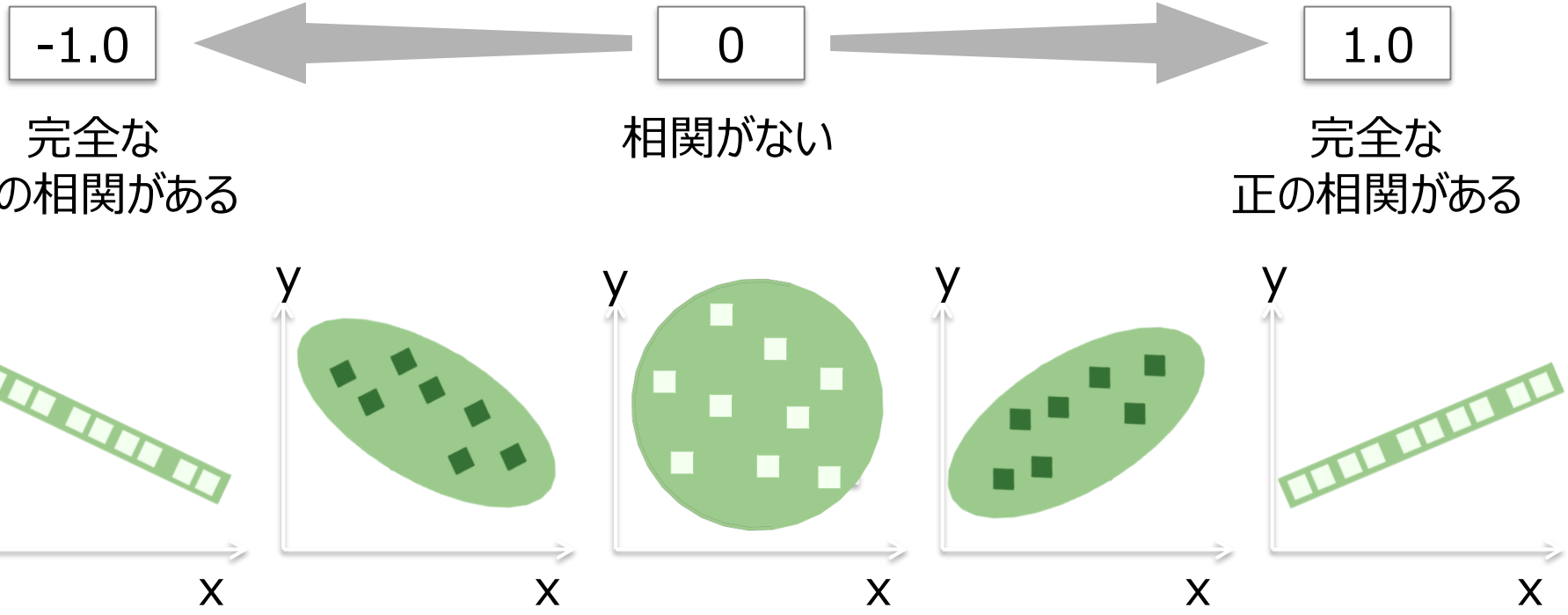
相関係数

相関係数とは

２つのデータ間の相関関係の強さを表す値のことで－１から１までの値を取ります。

相関係数が０に近いほど相関は弱く、１に近いほど相関は強くなっています。

…（相関係数）＝（共分散）÷（一方の標準偏差 × もう一方の標準偏差）

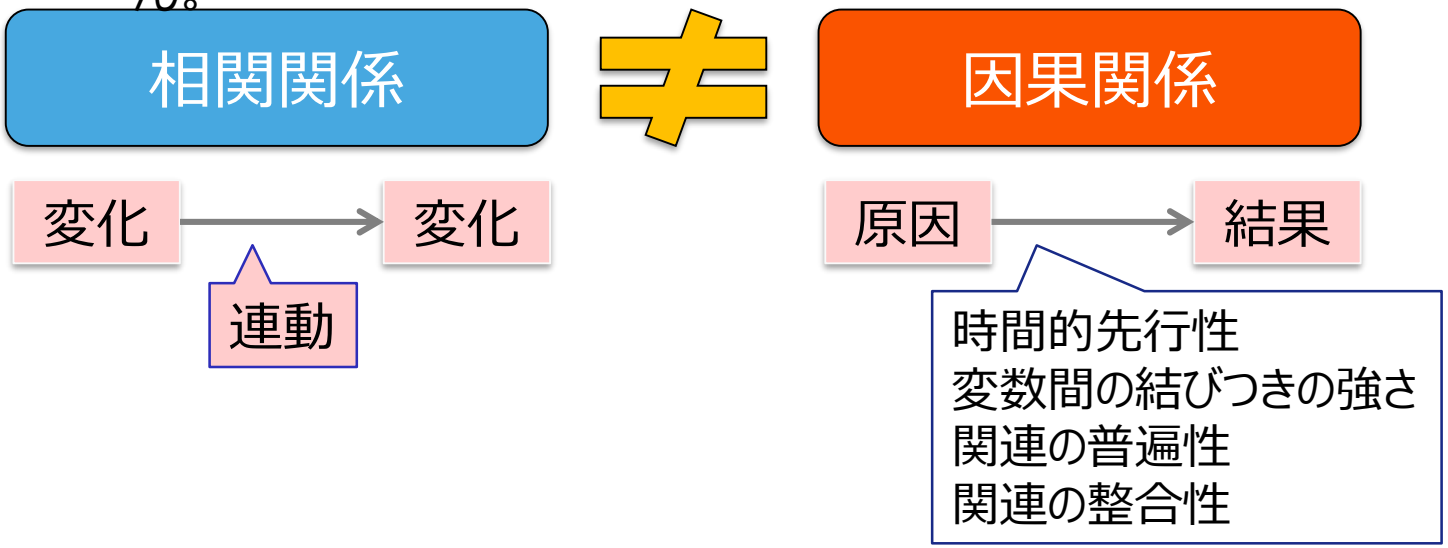


相関関係と因果関係

・相関関係：一方の値が変化すると、もう一方の値も変化するという２つの値の関係のこと。

・因果関係：２つ以上のものの間に原因と結果の関係が成り立っていること。つまり、一方によって、もう一方が引き起こされるという関係。

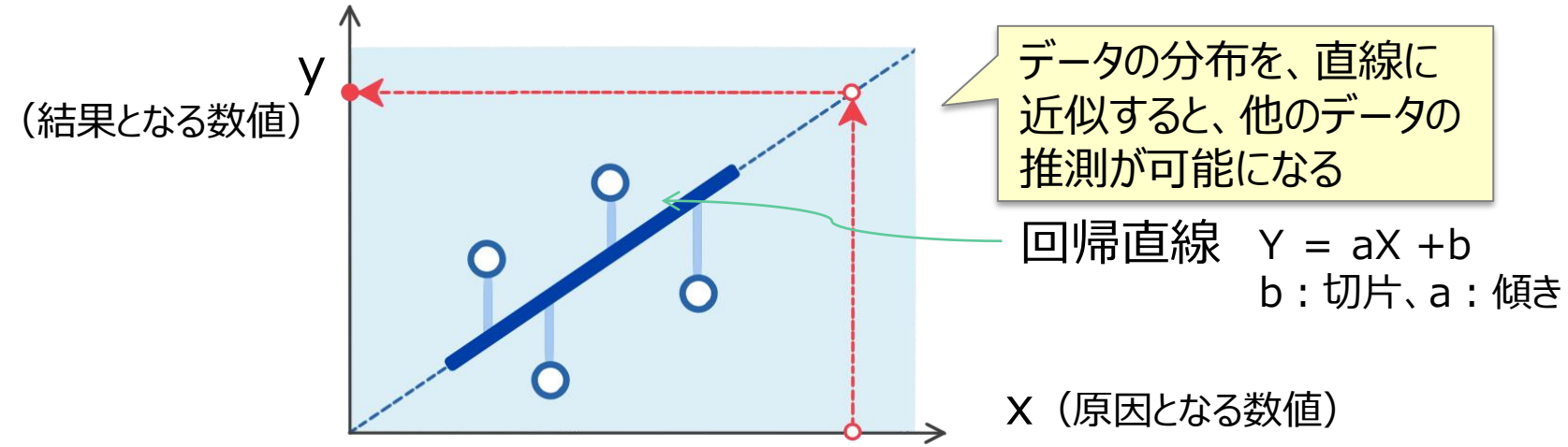
相関関係があっても因果関係があるとは限りません。



回帰分析と重回帰分析

回帰分析とは

「原因となる数値」と「結果となる数値」との関連性を、統計的手法によって分析することを言います。
回帰分析を行うことで、一方の数値（説明変数）の変化から、もう一方の数値（被説明変数）の変化を推測することができ、仮説を立てることが可能になります。式にすると、 $Y = aX + b$ のように表されます。

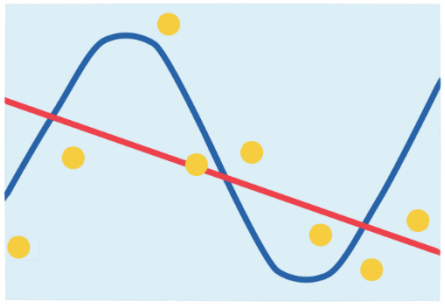


重回帰分析とは

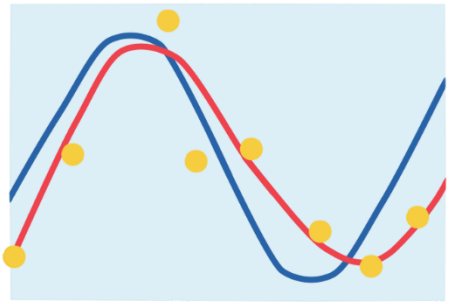
１つの目的変数を複数の説明変数を用いて推測する統計的手法のことを言います。
式にすると、 $Y = a + b_1X + b_2X + b_3X + \ldots + b_nX$ のように表されます。

多項式回帰曲線のイメージ

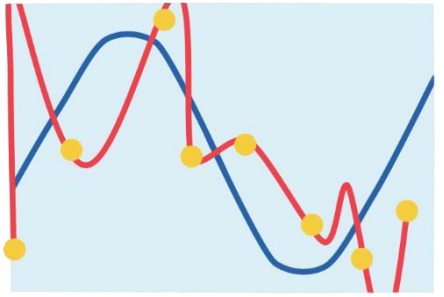
回帰曲線を用いて、データをモデル化することを考えます。



1次回帰線



３次回帰線



９次回帰線

ここでは３次回帰曲線を用いて表すのが適当だと考えられます。
上の図からも分かるように、９次回帰線は全ての点を通っていますが、実態とはかけ離れています。従って、新たな点が現れた時にその点が９次回帰線上にある確率は却って下がってしまい、予測には向いていません。
これを、過学習（オーバーフィッティング）といいます。

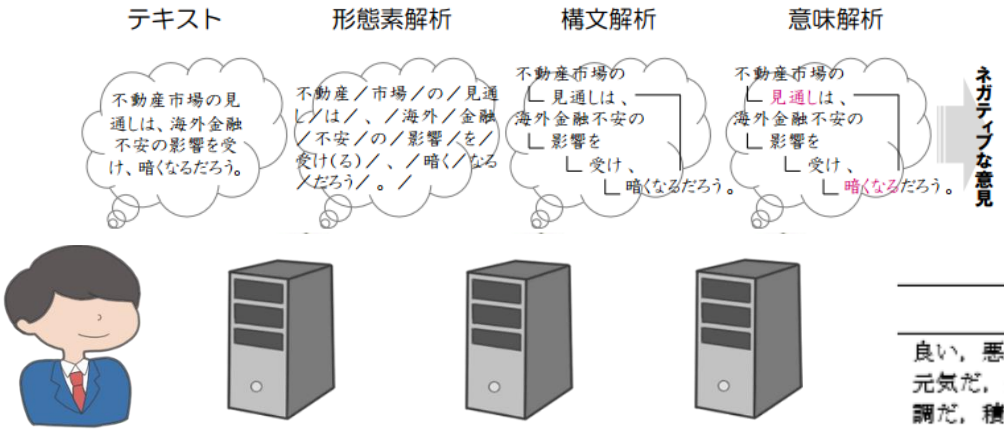
テキストマイニング

テキストマイニングとは

大量のテキストデータから、トピックの傾向や、役立つ知識、情報などを見つけ出す分析技術のことを言います。

まず膨大に蓄積されたテキストデータを単語や文節（形態素）に分解し、それぞれの形態素の出現頻度や相関関係を解析します。
そして、係り受けなどの関係や時系列の変化といった構文解析を行うことで、結果的に客観的な意味内容を掴むことを目的とします。

図表　テキストマイニングの流れ（イメージ）



図表　評判辞書のキーワード例（抜粋）

ポジティブ単語	ネガティブ単語
良い，悪い（否定），良好だ，活発だ，元気だ，効果的だ，最適だ，最高だ，順調だ，積極的だ，調子よく，適切だ，明るい，しっかり，安全だ，快調だ，経済的だ，上々だ，心強い，抜群だ，必要だ，便利だ，満足だ，等	悪い，良い（否定），暗い，異常だ，割高だ，苦しい，最悪だ，最低だ，弱めだ，心配だ，不安だ，不十分だ，不調だ，危険だ，苦しい，酷い，少な目だ，心細い，辛い，大変だ，難しい，悲惨だ，不要だ，無理だ，異様だ，等

出典：「テキストマイニングによる国土政策評価手法の研究」国土交通省

テキストマイニング

テキストマイニングをすることで、テキスト情報から何らかの有用な情報を得ることができます。

例 ・自社の評判と他社の評判を分析、比較する。
→トピックの傾向を把握したり、新たな気づきを得たりできる。

・売上やキャンペーンデータについて、時系列で比較する。
→マーケティングなどに有用な情報を得られる。

第６章

ビッグデータとAI、機械学習

クラウド、IoT

AIの発展

AI（artificial intelligence）とは

コンピュータ上などで、人工的に人間と同様の知能を実現させようという試み、あるいはそのための一連の基礎技術のことを言います。

・エキスパートシステムとルールエンジン
エキスパート（専門家）がルールをつくりプログラムに実行させ、推論機能を適用することで結論を得る。
エキスパートシステムは大量の既知情報を処理し、関係性を導ける。

・事例ベース推論（CBR）
類似した過去の事例を基準に、修正をしながら試行を行い、結果と事例を事例ベースに記憶する。

・ベイジアン・ネットワーク
因果関係を確率により記述する。
複雑な因果関係の推論を有向非巡回グラフ構造により表しながら、個々の変数の関係を条件つき確率で表す確率推論のモデルである。

機械学習

機械学習と深層学習

2005年以降は機械学習と深層学習による第3次AIブームとなっています。

機械学習

データから反復的に学習し相関を見つけ出すことです。また、その結果を元にして将来を予測することができます。
予測分析におけるモデル構築の自動化ができるため、データサイエンティストの人材不足を補うものとして期待されています。

深層学習

ニューラルネットワークの多層化、1990年代の視覚野の研究や、ブルーノ・オルスホーゼンによるスパース・コーディング理論を基にしたアルゴリズムが実装されたものを指します。
情報が第1層からより深くへ伝達されると、各層で学習が繰り返され、特徴量(問題の解決に必要な本質的な変数であったり、特定の概念を特徴づける変数)が計算されます。

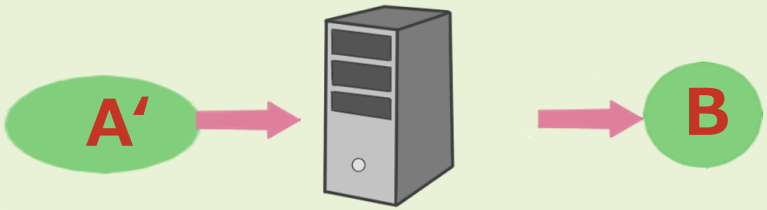
教師あり学習と教師なし学習

機械学習の手法として、教師あり学習と教師なし学習があります。

「教師あり学習」のイメージ
入力データに対応する出力を予測

① 入出力のペアで訓練データを用意し、入出力の関係を学習させる

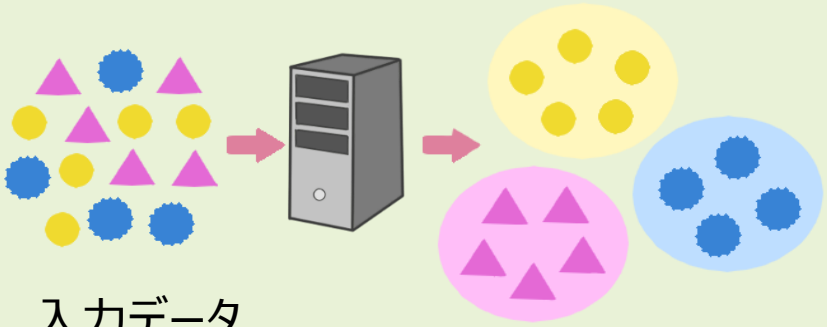
② 未知データを正しく判断できるようにする



→ 迷惑メール判定、株価予測などに使われている

「教師なし学習」のイメージ
入力データの特徴を学習して分類

① 入力データのみを与え、データの特徴をつかみ、分類を行うもの



→ クラスタリングなどを用いて、同じ集合の持つ特性などから結果を予測する

機械学習アルゴリズム

機械学習アルゴリズムの分類

・Classification/Class probability estimation
既存のデータをクラスに分類し、未知の新規データの分類を予測する。
新規データに対しては、属する特定のクラスを示したり、各クラスに属する期待値も計算したりする。
例 商品の買い替え時に、もともと使っていた製品から購入を予測する

・Regression（回帰分析）
既存データの関係を示す、数的な「関数」を推測し、定量的な予測をたてる。
例 広告費と売り上げの相関を見つける

・Similarity matching
新規データの、既存データへの類似性を予測する。

機械学習アルゴリズム

機械学習アルゴリズムの分類

・Clustering
具体的な指標を示さず、既存データを自然に分類できるグループを見つける。
分類後、各グループの特性を別のアルゴリズムで分析する。
　例 顧客をグループ化し、それぞれに別のサービスを提供する

・Co-occurrence grouping
既存データから、同時にまたは付帯的に起こる事象を推定する。
　例 Aを買った人は、Bも買っている

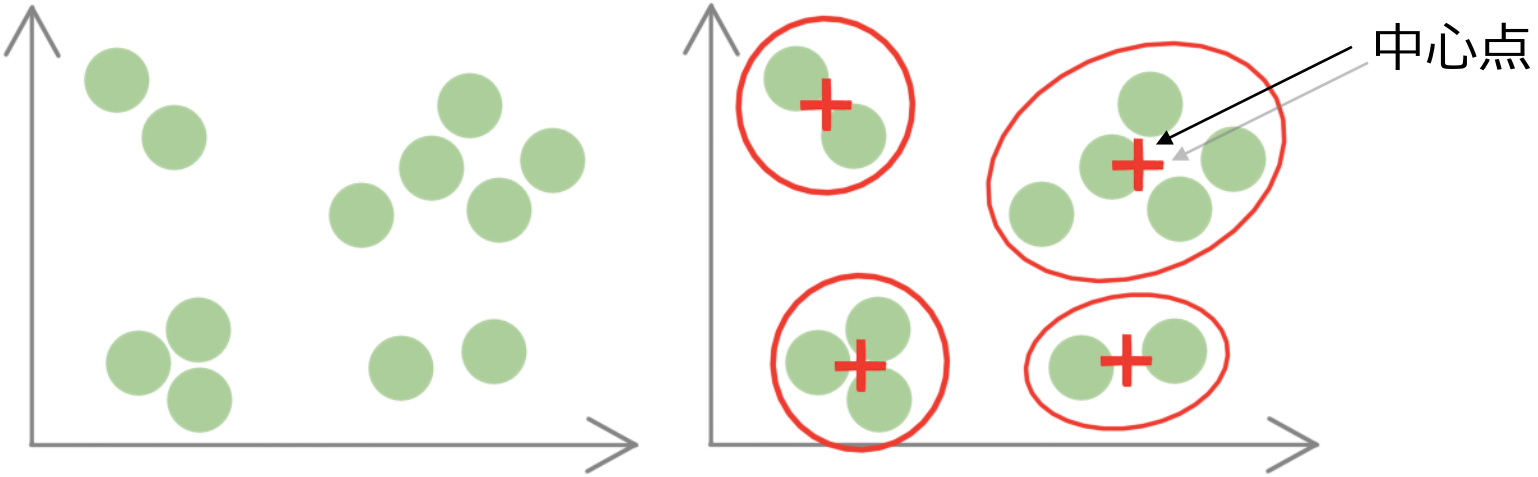
・Reinforcement learning(強化学習)
環境との相互作用を元に、データを収集し分析する。
例 囲碁プログラムがコンピュータ同士の対局から有効な手筋を発見する

機械学習アルゴリズム

クラスタリング (clustering)

データの集合を、具体的な指標を使わず性質が近い部分集合にすることです。

統計解析や多変量解析の分野ではクラスター分析 (cluster analysis) とも呼ばれ、基本的なデータ解析手法としてデータマイニングでも頻繁に利用されています。



機械学習アルゴリズム

協調フィルタリング

商品の閲覧と購入のデータから、人同士の類似性や商品間の共起性をアソシエーション分析（相関分析）で解析し、対象者の行動履歴と関連づけることで、パーソナライズされた商品を提示する手法です。

協調フィルタリングは、商品スペックの関連性や商品閲覧の共起性だけではなく、購買データを基に人と人の類似性も重要視します。それにより、対象者に似ている集団が持つ特徴の中から、意外性のある商品を提示すること（セレンディピティ）もでき、コンバージョンレートのアップが期待できます。

画像分析手法

画像データを解析し、顔認識や、画像の分類などを行う事が可能となります。

・SIFT
特徴点周りの輝度から最も輝度変化が大きいベクトルを調べ分類することにより、細かい特徴を把握することができる。

・Haar-like
形状やパーツが固定的なものに対し、平均的に示される特徴から画像を解析する。カメラの顔認識などに用いられる。

・畳み込み深層学習（CNN）
画像をピースごとに比較して、位置や形状の特徴が類似する箇所を検出する。
その上で、色、明度、エッジなどの多層において学習を繰り返す。
顔認証などに用いられる。

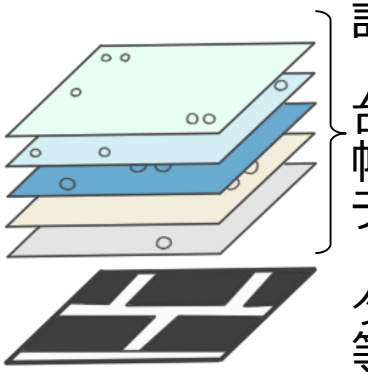
地図情報システムとの連動

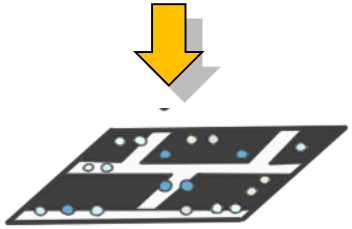
GIS

地理空間情報を
データ毎に管理

地図レイヤーと
分析対象データを

位置関係の把握
データ検索、表示、
関連分析





GISの特徴

- 地図データとビッグデータを重ね合わせ、新たな発見を得ることができます。
- 地理的な位置情報との情報の関連性を視覚的に理解することができたり、図面上で情報の集約、整理をしたりできます。

GISの利用シーン

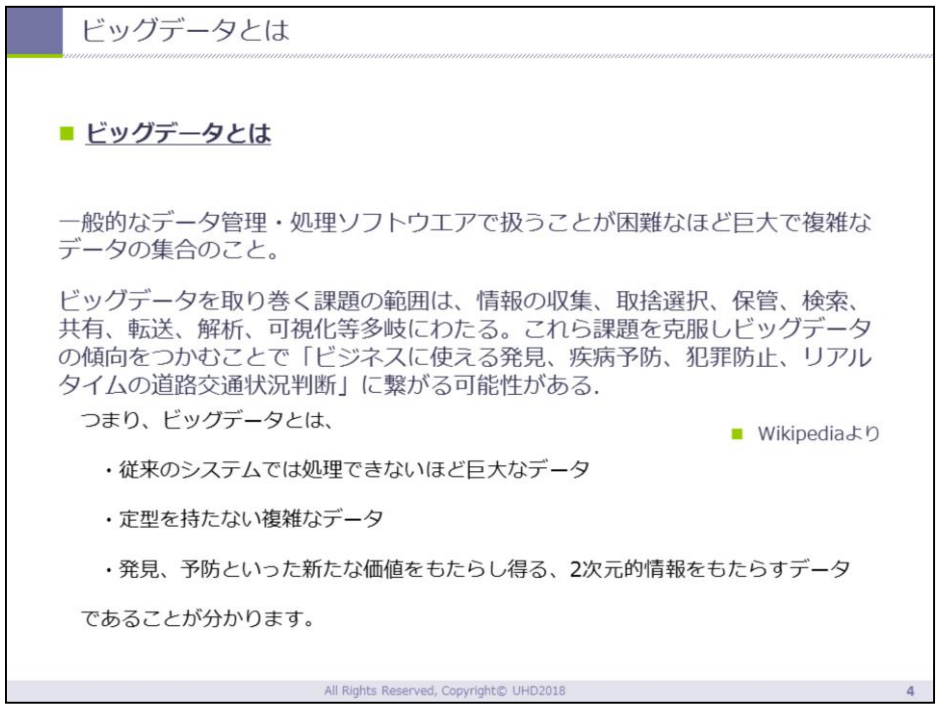
- 災害時の、正確で素早く効果的な行動決定などに利用されています。


ビッグデータ講座

目次

- ビッグデータ概要
- ビッグデータとセキュリティ
- ビッグデータを支える技術　　（クラウド、IoT）

ビッグデータ概要



「ビッグデータ」を調べると、曖昧な説明しか見つかりませんが、それらを注意して読むとエッセンスがみえてきます。以下はビッグデータの特徴です。

・普通の処理では対応できない量
・定型なデータではない
・本来の目的とは異なる使い方により二次効果を得る

このようなデータが身近にあるか、想像してみてください。

ビッグデータの量

- **巨大なデータとはどれくらい？**

ビッグデータの例を見てみましょう。

データ量を表す単位は、以下の順に1024倍となります。

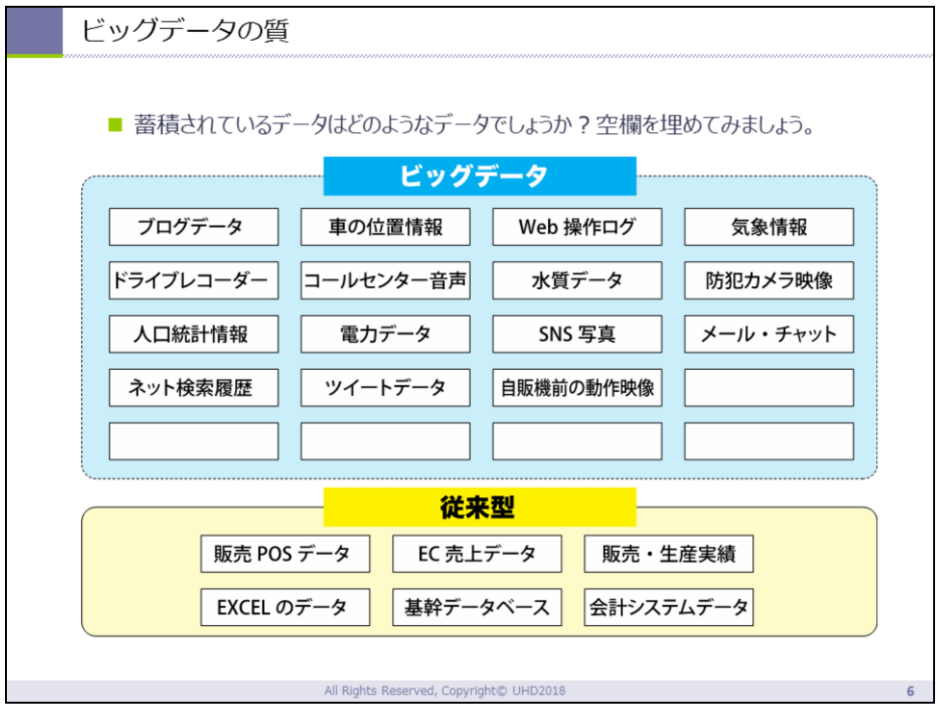
キロ(KB)<メガ(MB)<ギガ(GB)<テラ(TB)<ペタ(PB)<エクサ(EB)<ゼタ(ZB)

全世界で生成・消費されるデジタルデータの総量

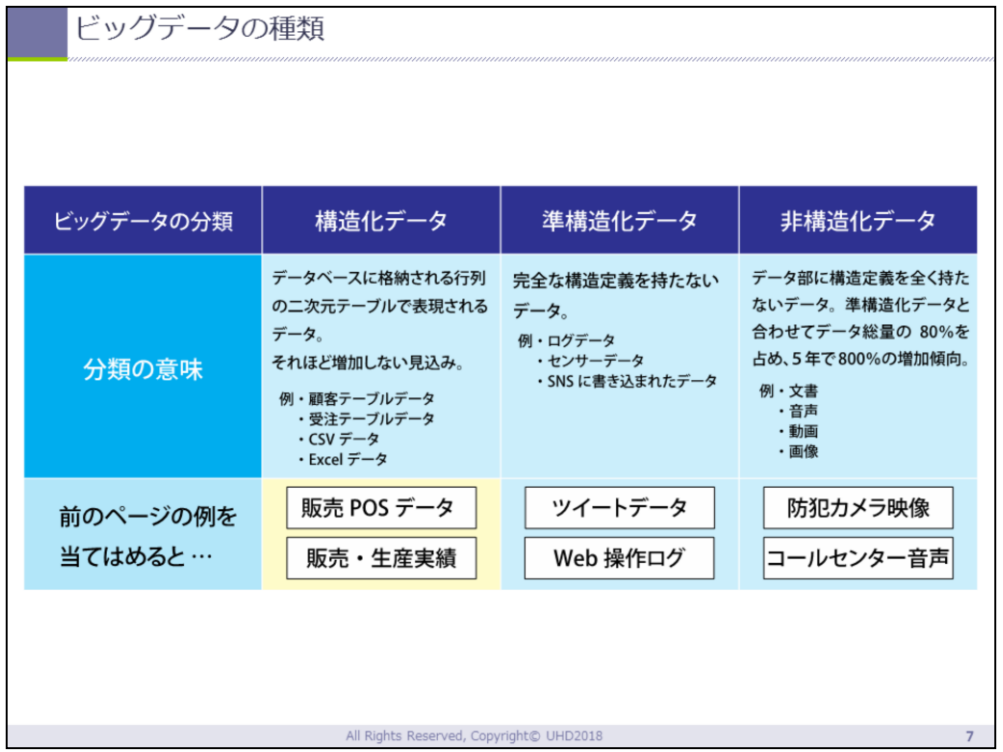
IDC（International Data Corporation）の発表： 59ゼタバイトを超える

出典：https://www.idc.com/getdoc.jsp?containerId=prUS46286020

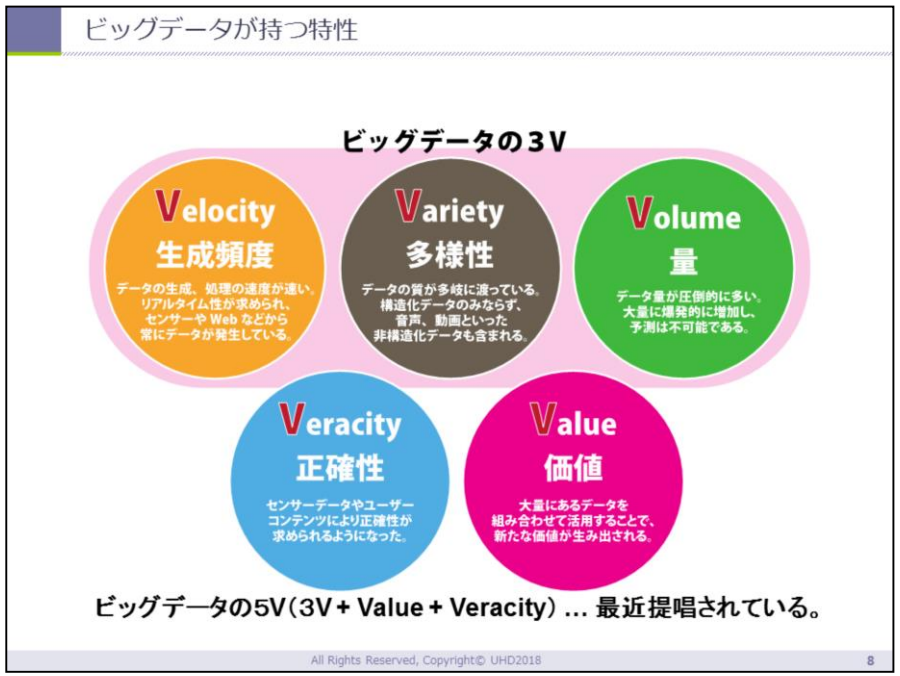
ビッグデータの量はどれくらいでしょうか。その規模は、もはや1日TB級のデータを扱うレベルであるということを覚えておいてください。



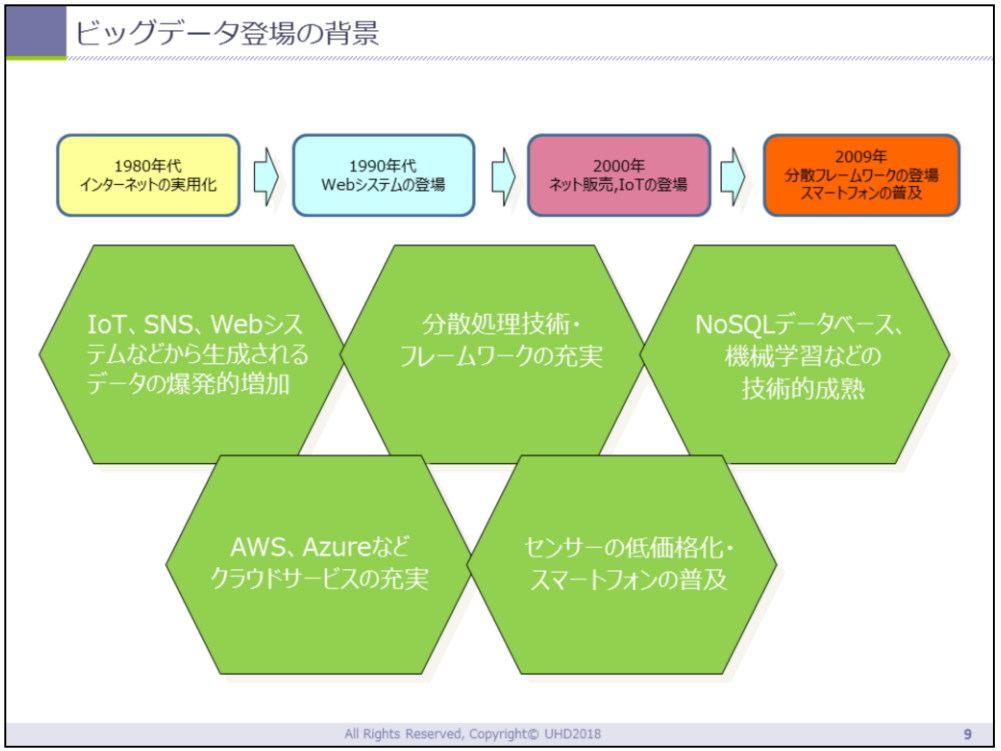
ビッグデータといえばどのような種類のデータを思い浮かべますか？空欄に思い浮かぶものを考えてみましょう。また、それらと比較し、従来型のデータと決定的に異なる点は何でしょう。ビッグデータでは、「定型業務で発生したデータ以外も扱い、それらを組み合わせて分析する」ということを覚えておいてください。



ビッグデータには構造化データ、準構造化データ、非構造化データがあります。特に非構造データの増加率が爆発的です。それぞれの違いをしっかりと覚えておいてください。また、前ページのデータをこのページの定義をもとに分類してみてください。



量と多様性（質）のほかに、ビッグデータが持つ特性として、「リアルタイムにいつでも発生するデータ生成頻度」が挙げられます。センサーデータやTwitterデータなどは常に発生し、処理をしなければなりません。これらを合わせビッグデータの３Vと呼びますが、最近では、Veracity、Valueなども合わせて、４V、５Vといわれることがあります。



　処理能力の向上とWebシステムの登場が大きいといえます。例えば、googleなどの検索内容から顧客の購買動向を把握することにビッグデータが活用されていきました。今日に至るまで、様々な技術により、巨大なデータ処理技術が支えられています。「データを保持し続ける状況とそれを分析するニーズ、技術の進歩がマッチして初めて、ビッグデータが登場する契機が生まれた」ということをしっかりと覚えておいてください。

　ビッグデータはどこに保管されているのでしょうか。取引先から渡ってくるデータや今まで保存していたログデータ、社外Webサイトなど、自社内はもちろん、自社ローカル以外にも眠っている場合があります。また公共団体が提供するデータや、事業者が提供する商用データなどもビッグデータである場合もあり、ビッグデータはあらゆるところに散在しています。

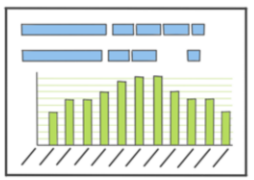
オープンデータとは

特徴	■ 誰でも入手可能で、自由に利用・再配布できる状態で存在する ■ 特許・著作権に制限がない ■ コンピューターから利用できる状態となっている
公開主体	■ 政府 ■ 地方自治体 ■ 研究機関・大学 ■ 民間企業
具体例	■ 国勢調査データ（政府統計の総合窓口「e-Stat」） ■ 公共施設やAEDの位置データ ■ 気象データ ■ 有志により作られた地図データ（OpenStreetMap など） 「行政と市民によるオープンデータ共創支援プラットフォーム（LinkData）

官公庁が公開しているデータなどをオープンデータと呼ぶことがありますが、これもビッグデータの１つです。オープンデータは誰でも、いつでも、利用・再配布できる状態にあるデータです。「身近なオープンデータ」にはどんなものがあるか、考えてみてください。

オープンデータ×ビッグデータ活用例

・過去のTwitterなどのSNS上の書き込み ＋ 販売データ
→ 相関を調べ、売上の増減や欠品可能性を予測する。



・自社の販売データ ＋ 気象データ
→ 気象変化と売上推移の相関を見出し、予測を行う。

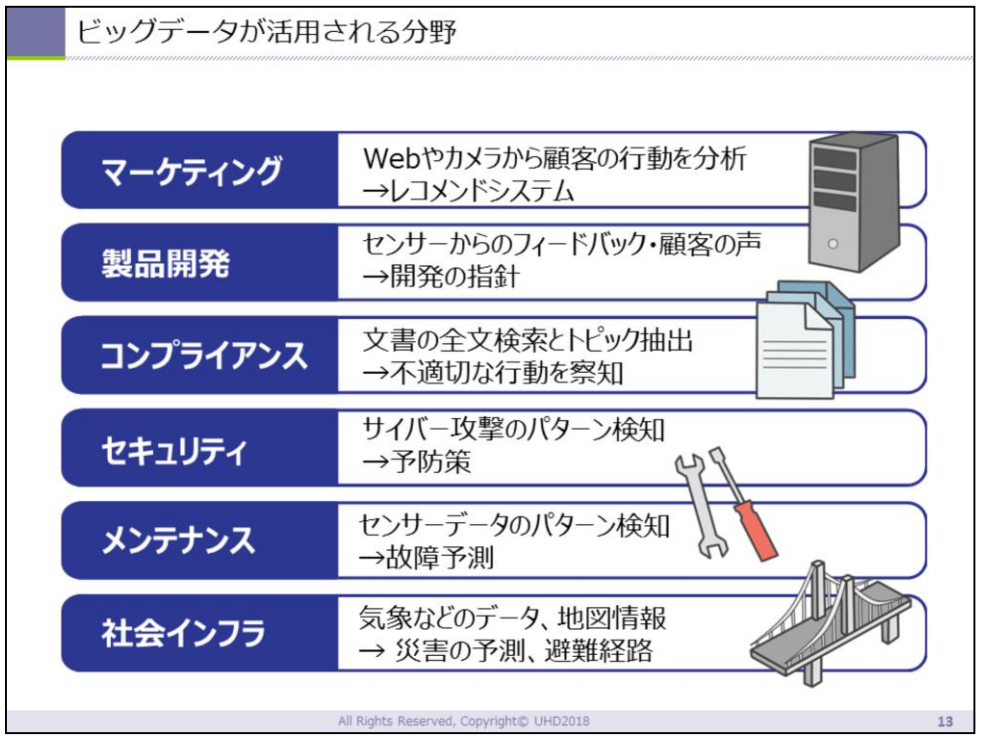
・医療施設の位置データ ＋ 患者の郵便番号のデータ
→ 来院マップを作成し、診療費ごとの外来状況を分析することで、地域医療に関して重点的な連携、促進を図る。

・国勢調査などの人口統計情報 ＋ 将来の人口推計
＋ ターゲット層の世帯が多数存在する地域の売上相関
→ 重点的に販売を行う地域を模索する。

オープンデータ単体では価値を生み出さない場合がありますが、オープンデータとビッグデータを掛け合わせると、それを単体で使うより大きな価値を生み出す場合があります。もちろん、公共施設マップなど、オープンデータそのもので価値があるデータも存在します。

①気象データ単体を利用するケース
②ビッグデータを掛け合わせるケース

を考え、価値に違いがあるかを考えてみましょう。

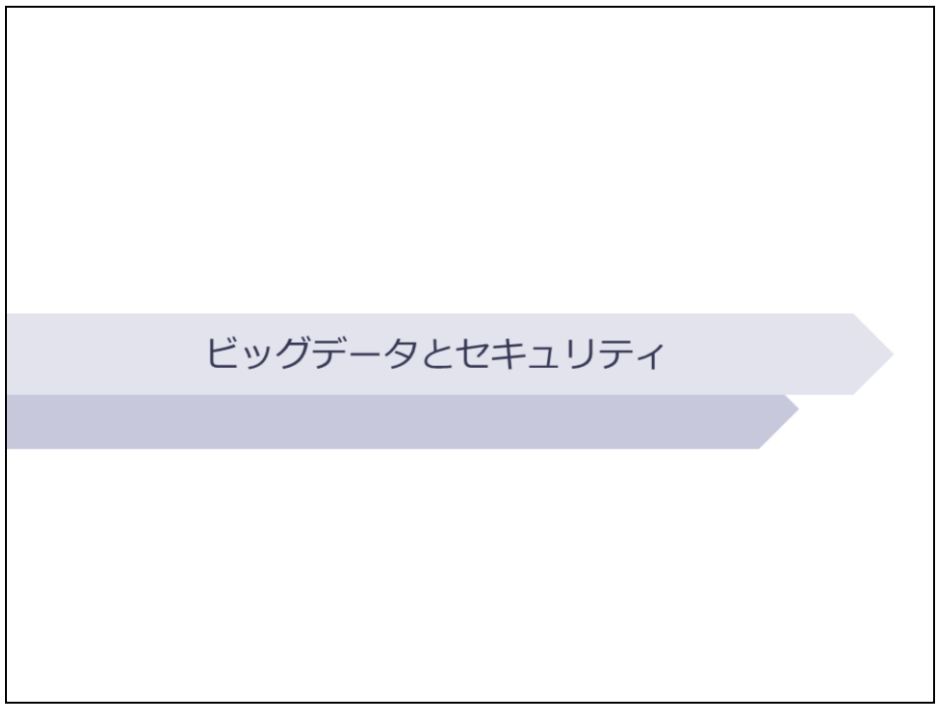


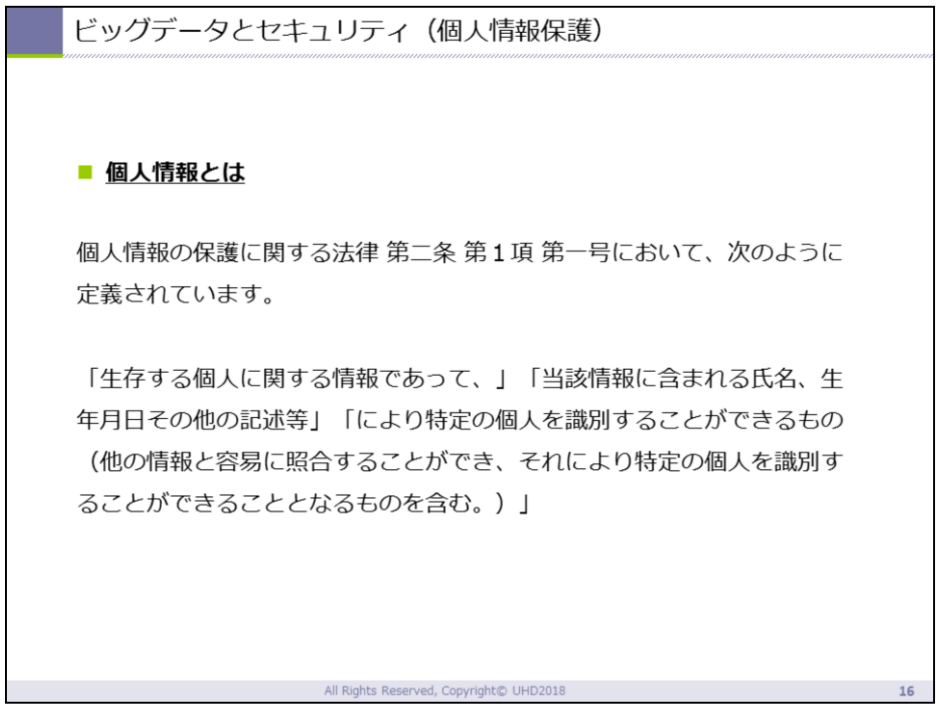
　ビッグデータはどのような分野で活用されているでしょうか。ビッグデータ適用分野は多岐に渡り、これまでのアプリケーションの枠を超え、全く新しいデータ活用をもたらします。今まで耳にしたビッグデータの活用分野を挙げてみてください。

ビッグデータが活用される業界

	運輸	金融	医療・健康	製造
利活用事例	1. 航空機チケットの割引サービスの改善：Web販売サイトでの購入傾向を分析 2. 渋滞予測：スマホGPSの情報を分析 3. トラックの最適な輸送ルート：過去の情報から算出	1. 金融商品の提案・開発：数千万件の顧客情報から、決済や資産運用の動きを分析 2. 保険サービスの開発：車の走行距離の情報を元に保険料を定めるサービスを開始	1. インフルエンザ対策：SNS上のコメントや検索キーワードから、広がりを検知 2. メディケア（アメリカの公的保険）のデータ公開：公的機関が分析して公開	1. 品質改善のための最善策の策定：生産工程に関する多数の情報を収集、分析 2. 製品開発：建機にセンサーを設置し故障時の稼働環境を分析してフィードバック。
効果	販売促進 人流動態分析	新サービス開発	兆候把握・ 情報提供	品質向上

ビッグデータ活用の具体的な例です。新聞などで日々掲載される技術革新が、ビッグデータによって支えられているケースも増えています。

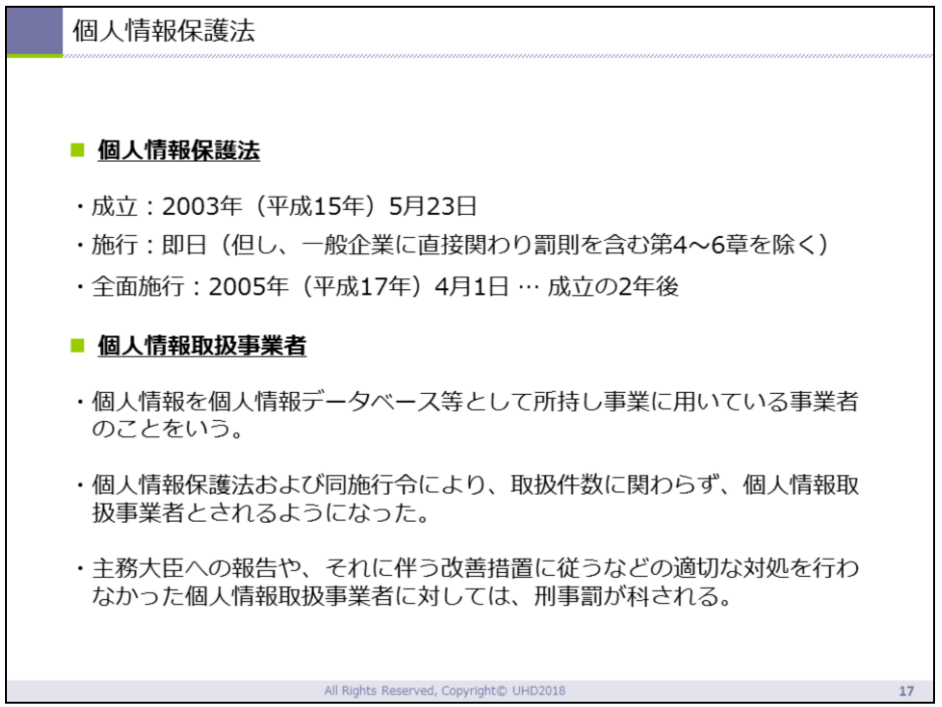

ビッグデータとセキュリティ



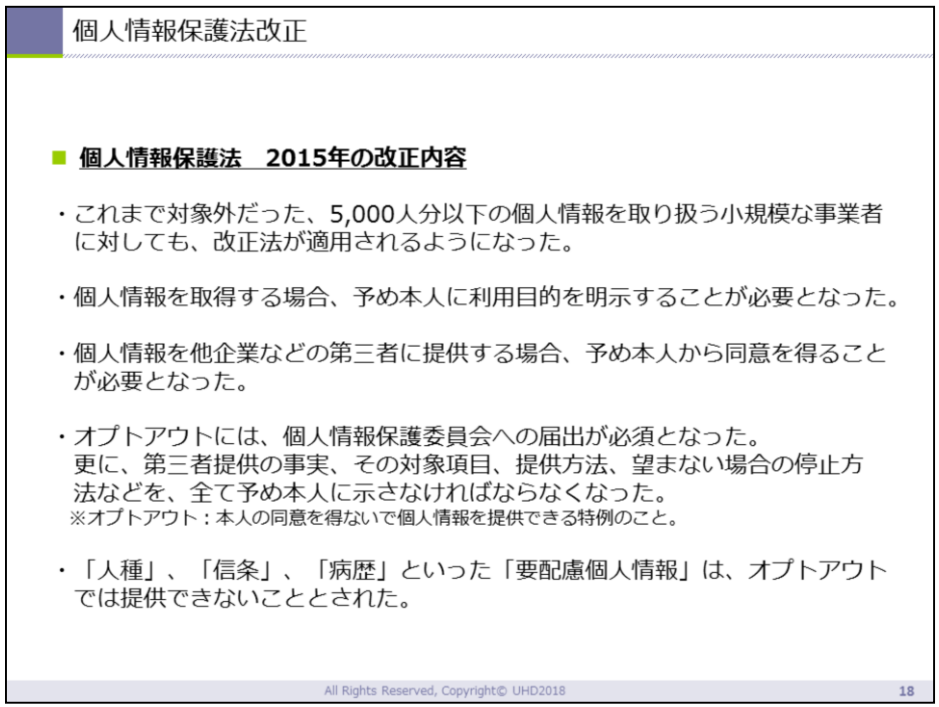
ビッグデータに個人情報が含まれている理由で、活用を断念するケースがあります。ここでは、個人情報を取り上げます。

重要なのは、データ単体では個人を特定できない情報であっても、他のデータと組み合わせることによって、個人を特定できる場合があるということです。「他の情報と容易に照合」することにより、「特定の個人を識別」ができてしまう例として、下記が挙げられます。

・過疎地の郵便番号
・希少な病名
・顧客IDと顧客マスター
・写真とGPS情報　　など



　個人情報保護法は2015年に改正され、より範囲が明確になり、運用方法も明確に定義されました。ここで重要なのは、「個人情報を取り扱う場合には、何らかの形で個人の同意が必要」ということです。



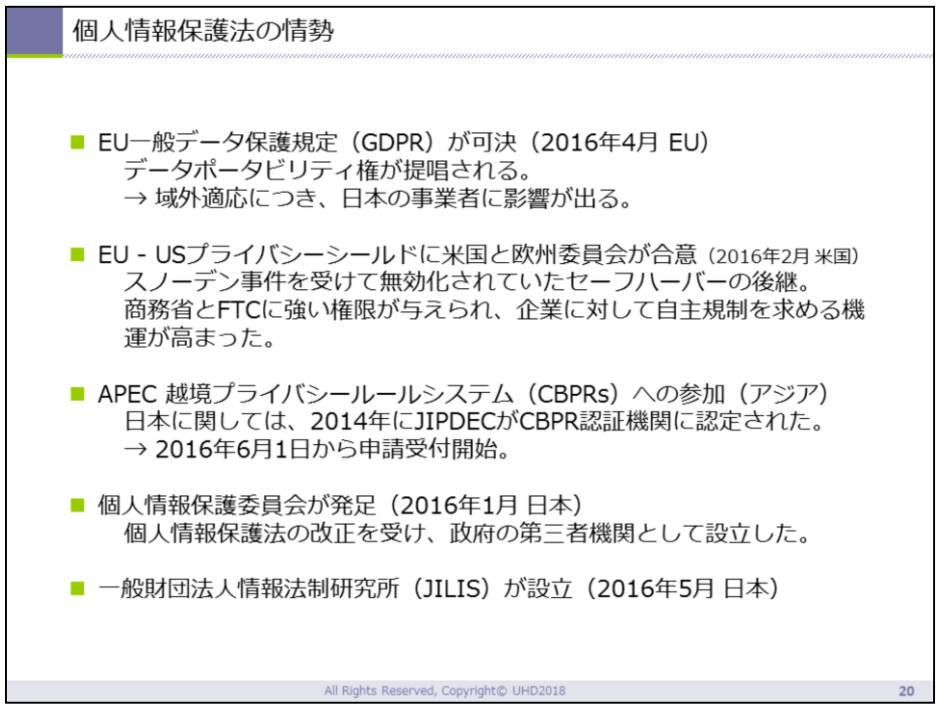
　個人情報保護法は2015年に改正され、より範囲が明確になり、運用方法も明確に定義されました。ここで重要なのは、「個人情報を取り扱う場合には、何らかの形で個人の同意が必要」ということです。

個人情報保護の情勢

- 1980年　プライバシー保護と個人データの国際流通についてのガイドラインに関するOECD理事会勧告（OECDプライバシーガイドライン）
  （OECDの34加盟国）
  ①収集制限　②データ内容　③目的明確化　④利用制限
  ⑤安全保護措置　⑥公開　⑦個人参加　⑧責任
  の８原則からなる。
- 1995年　EUデータ保護指令　（EUの28構成国）
  EUおよび英国においては、十分なデータ保護レベルが確保されていない第三国への個人データの移動を禁止する。
- 2003年　個人情報保護法　（日本）
  個人情報を扱う事業者に対し、個人情報の適切な取り扱いを求める。
- 2012年　消費者プライバシー権利章典　（アメリカ）
  ①個人によるコントロール　②透明性　③背景情報の尊重
  ④セキュリティ　⑤アクセスと正確性　⑥適切な範囲の収集　⑦説明責任
  の7つの権利を定める。

海外の個人情報に対する規制はどのようになっているかみてみましょう。

海外では個人の意思表示により、データの削除ができる仕組みなど様々な規制が存在します。



　海外の個人情報保護の情勢をもう少し詳しくみてみましょう。

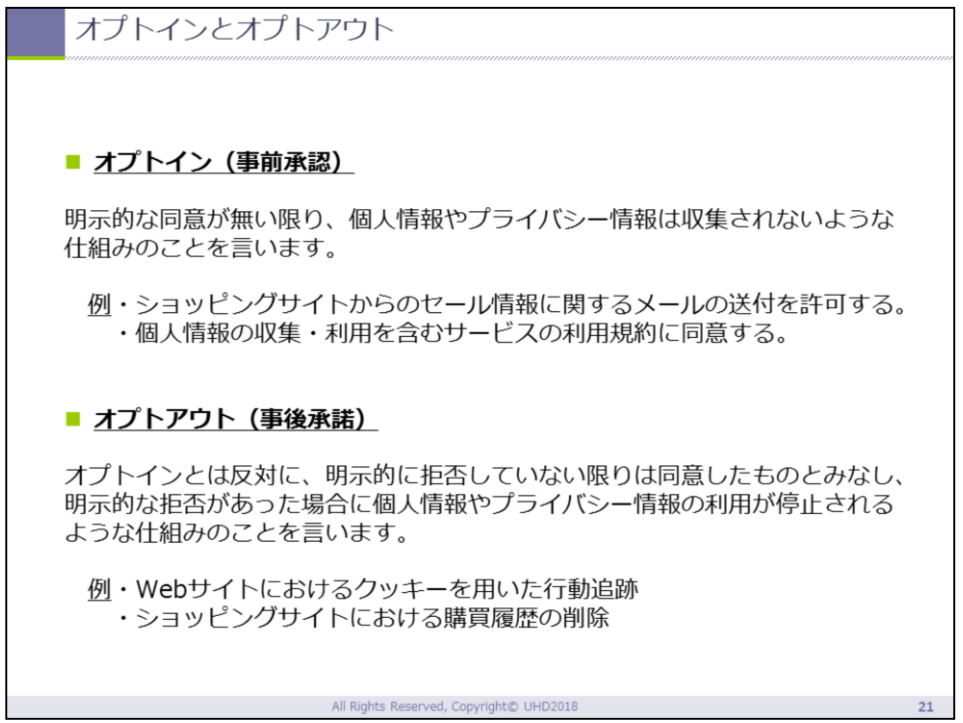
EU法のGDPR（General Data Protection Regulation：一般データ保護規則）では、

①IDなども個人情報扱い

②EU域外に個人情報を持ち出せない

③規定に違反した場合などは制裁金が科せられる場合もある

などの点に注意が必要です。



　個人情報の許諾を個人から得る方法として、オプトインとオプトアウトがあります。オプトインは事前承認、オプトアウトは事後承認と覚えておいてください。

匿名加工処理の手法

以下のそれぞれの手法を組み合わせることで、より強固な匿名化が実現されます。

技法大部類	No.	技法例	概要
摂動法	1	K-匿名化	同じグループ内に、同じ属性のユーザが「K人以上いる」状態を作り出す。
	2	L-多様性	漏えいさせたくない属性が同じグループ内で「L種類以上ある」状態を作り出す。
	3	T-近接性	マイナー属性を持つグループが生まれるなど、属性値の分布に偏りが出てしまう場合に、グループの分割や一般化を行う。
	4	差分プライバシー	2006年に提案された新しい手法。元のデータベースにノイズを足した別のデータベースを用意し、守りたいレコードを特定しづらくする。
暗号法	5	質問監査	データベースへのアクセス者に質問を投げかけ、答えられれば、アクセスに対する回答を返す。
	6	秘密計算	関係者全員が、自社データを他人が読めないように変換し、秘密計算のシステムへ投入する。そのシステムの管理者が、秘密計算の結果を求め、関係者に回答する。
	7	準同型性公開鍵暗号を用いた暗号プロトコル	遺伝子データなど、加工してしまうと、そもそも分析できなくなるデータを処理するときに活用。検索者の検索クエリ、データベース、その回答それぞれを暗号化する。分析者が元データにふれずとも、望む解析結果が得られる。

出典：中川裕志『プライバシー保護入門：法制度と数理的基礎』（2015年）

　個人を特定できないようにデータを加工することを匿名加工処理と呼びます。具体的にどのような手法で匿名化を実現するのかをみてみましょう。

K-匿名化や、L-多様化はデータそのものを加工し、もしくはレコードを増やし、個人を特定できないようにする技術です。あまり、K=XXなどの数値を大きくし過ぎると、安全性は高まりますが、統計上や機械学習上のノイズになる場合があり、正しい結果が得られなくなる場合もあるので注意です。

識別子の削除（仮名化）

個人の識別・特定に直結するカラムを削除して、仮名化を行います。

No.	ZIPコード	年齢	職業	病状
1	13068	28	ダンサー	心臓病
2	13068	29	技術者	心臓病
3	13053	21	法律家	感染症
4	13053	23	技術者	感染症
5	14853	31	技術者	風邪
6	14853	37	作家	風邪
7	14850	36	法律家	がん
8	14850	35	技術者	がん

準識別子　　漏えいさせたくない属性

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

識別子の削除（仮名化）は、個人の氏名や住所などの情報を仮名と置き換える、もしくは削除することによって、データから直接個人を特定できなくすることをいいます。この表の場合は、個人を特定できる氏名を削除し、住所をZIPコードに置き換えています。これによって、個人を特定することが容易ではなくなります。

K-匿名化したテーブル

次に、再特定・識別につながる「職業」を秘匿した上で、「年齢」、「病状」の列に「同じ値が少なくとも2つ以上は存在する状態」のテーブルを作ります。

No.	ZIPコード	年齢	職業	病状
1	13068	28-29	*	心臓病
2	13068	28-29	*	心臓病
3	13053	21-23	*	感染症
4	13053	21-23	*	感染症
5	14853	31-37	*	風邪
6	14853	31-37	*	風邪
7	14850	35-36	*	がん
8	14850	35-36	*	がん

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

K-匿名化は必ず、K=数値で表された数以上レコード（行）が存在するようにデータを加工します。同じ保護属性（漏洩させたくない情報）の組み合わせを持つレコードが、少なくともK個存在し、保護属性からの識別がK人未満に絞り込めない状態になります。この場合はK＝2で、同じ識別子をもったレコードが少なくとも2個以上存在するようにしています。そのために、例えば年齢を28という数値ではなく28-29という幅で表すなどの加工をしています。

L-多様化したテーブル

「ZIPコード」と「年齢」を曖昧にして、「どのレコードを取り出しても、2種類の「病状」が存在する状態」になるようにします。

No.	ZIPコード	年齢	職業	病状
1	130**	21-29	*	心臓病
2	130**	21-29	*	心臓病
3	130**	21-29	*	感染症
4	130**	21-29	*	感染症
5	148**	31-37	*	風邪
6	148**	31-37	*	風邪
7	148**	31-37	*	がん
8	148**	31-37	*	がん

出典：「情報処理学会」（Vol.54 No.11 Nov.2013）より

L-多様性は、同じ保護属性の組み合わせを持つレコードが、複数個存在し、かつ対応する非保護の属性情報の値が少なくともL種の多様性を持つことで、属性推定が起こらない状態です。具体的には似たようなレコードを追加して、個人を推定できない状況を作り出します。この場合は、年齢帯とIPコードの上3桁の組み合わせからレコードを抽出しても、一つの保護属性（病状）には絞れないようになっています。例えば、130**かつ21-29でも、心臓病と感染症の2つの保護属性が抽出されます。

ビッグデータを支える技術

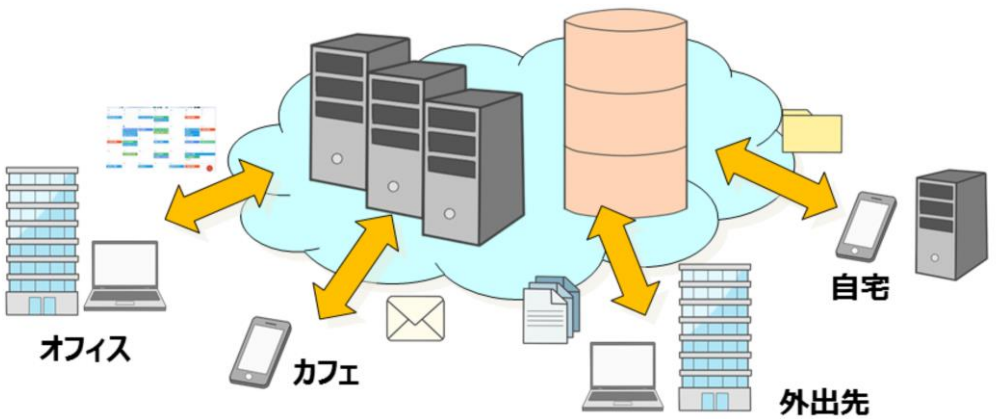
- **クラウド、IoT**

ビッグデータ周辺技術　クラウド

■ **クラウドコンピューティングとは**

メールやグループウェア、その他様々なサービスをインターネット上で提供し、インターネット上にデータまでをも保存するようなサービス形態のことをいいます。

アプリケーションの実行場所が、雲（クラウド）のようにどこにあるのかが分からないモヤモヤとした場所にあることから、このように呼ばれています。



　ここからは、ビッグデータを支える周辺技術を学んでいきます。

クラウドサービスはビッグデータ登場の背景にも密接にかかわっており、ビッグデータの置き場所などに活用されています。クラウドはどこに存在するか分からないデータセンターのことです。データの共有や業務アプリなどの様々なサービスが提供されています。

クラウドサービスの種類

- **クラウドサービスの種類**

- SaaS（Software as a Service）

  インターネットを経由してソフトウェアパッケージを提供するサービスのこと。アプリをPCにダウンロードしなくても、Webなどのブラウザ上で利用することができる。

  例・メール　・カレンダー　・チャット

- PaaS（Platform as a Service）

  インターネットを経由してアプリの開発・運用環境全体を提供するサービスのこと。システム管理者、開発者向けである。

- HaaS / IaaS（Hardware as a Service / Infrastructure as a Service）

  インターネットを経由してハードウェアや回線などのインフラを提供するサービスのこと。ユーザーはハードウェア資産を所有することなく、仮想サーバーやストレージ（外部記憶装置）を利用することができる。

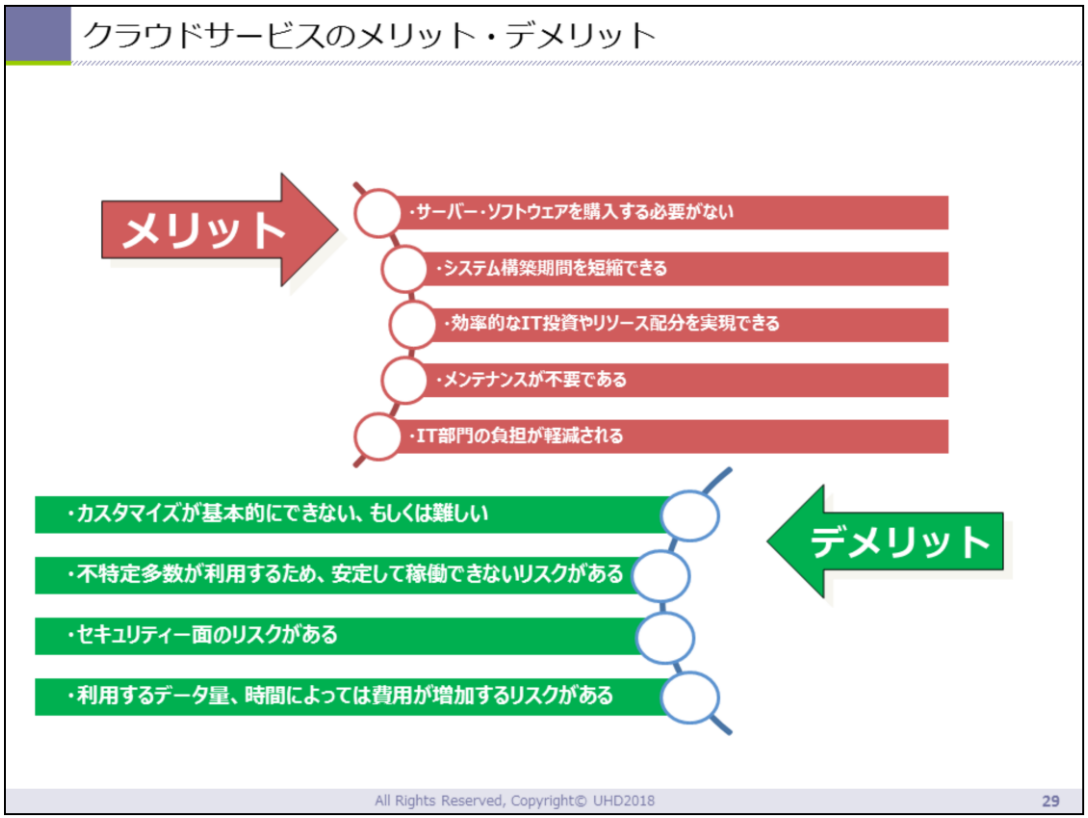
クラウドには様々な利用形態の種類があります。代表的な分け方は上記のようにSaaS（サース）、PaaS（パース）、IaaS（イァース）の分類です。これらは提供するもの（レイヤ）の違いであり、

IaaS（イァース）は、ハードウェアやネットワーク、OSなどの基盤を

PaaS（パース）は、基盤の上で動くミドルウェアなども含む開発・運用環境を

SaaS（サース）は、さらにその上で動くアプリケーションを

提供します。



クラウドサービスのメリットとデメリットには、どのようなものがあるかみていきましょう。

サーバーの調達や、メンテナンスから解放される一方、カスタマイズの難しさや、不特定多数のユーザーの目にさらされる危険もあるため、個人情報の扱いがあるデータの運用には注意が必要です。

クラウドコンピューティングとビッグデータ

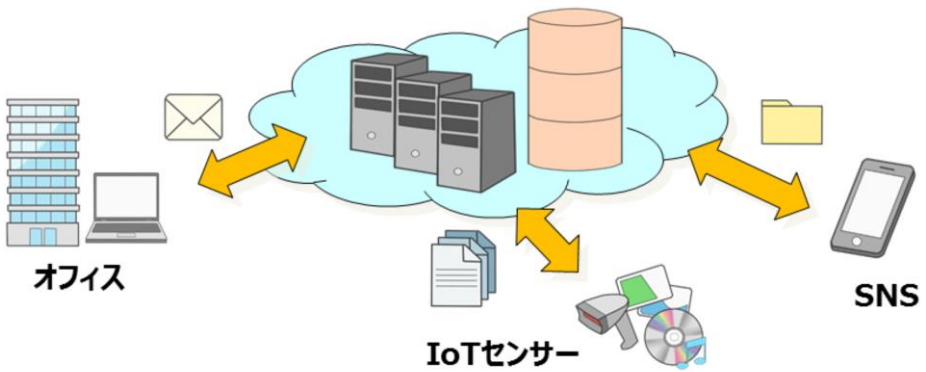
■ **クラウドサービスに蓄積されるビッグデータ**

クラウドサービスは、ビッグデータの蓄積場所に有力な選択肢です。

実際、以下はビッグデータがクラウドに保存されています。

・GREE、mixi、Facebook、Twitterなどに代表されるSNSのユーザーデータ

・IoTのようにセンサーなどで発生したデータ



　ビッグデータの保存先として、サーバーサイジングを自由に変更できるクラウドサービスは有力な選択肢の１つとなります。IoTセンサーデータのデータ収集もクラウドサービスベンダーが担う状況も整いつつあります。

ビッグデータ周辺技術　IoT

- **IoTとは**

IoT = Internet of Things　「モノのインターネット」

ここで、「モノ」とは、ネットワークに繋がるあらゆる物のことです。
従来はインターネットとは関係のなかったもの、例えば、

・眼鏡　・服　・時計　・冷蔵庫　・電力メーター　・自動車
・太陽光パネル　・家　・スマートフォン

もすべて「モノ」です。

IoTとは、モノがネットワークに繋げられることによって、モノとインターネットが相互に情報交換をできるようになった状態のことをいいます。

　センサーデータを分析する際に耳にするIoTという言葉は何を意味するか確認しましょう。モノが双方向でインターネットにつながりデータのやり取りができる状態をIoTと呼びます。

IoTとは

- Internet of Thingsという用語は1999年、ケビン・アシュトン（イギリス）によって初めて提唱されました。
- 当初はRFIDによる商品管理システムをインターネットに例えたものでした。
  → 徐々にスマートフォンやクラウドコンピューティングが普及。
    → IoTはモノ自体がインターネットを形作るという環境全体のことを表す概念として捉えられるようになる。
- IDC（ICT市場調査会社）による定義
  「IP接続による通信を人の介在なしにローカルまたはグローバルに行うことができる識別可能なエッジデバイスから成るネットワークのネットワーク」

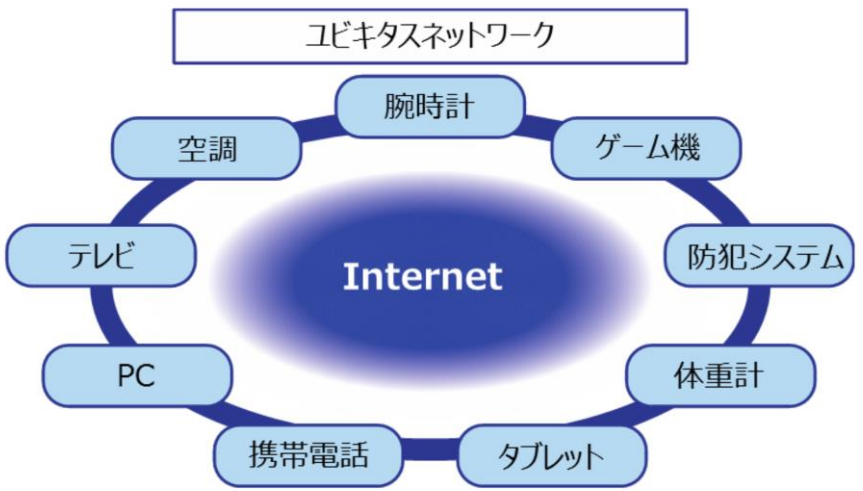
　ここでは、IoTという言葉はいつ頃から使われ始め、定着していったのかをみていきましょう。

IoTは2000年頃から使われ始めましたが、当初は現在よりも狭義な使われ方でした。現在では、モノ自体がインターネットを形作る（モノがインターネットで情報交換する）という広い概念として捉えられています。

IoTとユビキタス社会

■ **ユビキタスネットワークとは**

いつでもどこでもインターネットを利用できるという概念のことをいいます。
1988年ゼロックス社パロアルト研究所のマーク・ワイザーにより提唱されました。



　ここからは、IoTと似た概念であるユビキタスとM2Mについて確認し、その違いをみていきましょう。

まず、ユビキタスについてです。ユビキタスは、2000年頃に登場した概念で、人がどこにいてもネットワークにつなげる（つながる）状態を指します。ユビキタスネットワークの中心は人なので、IoTのモノとモノとの相互制御とはその点で異なります。

IoTとユビキタス社会

IoTとユビキタスの違い

・IoT：モノとモノが相互に制御し合っている状態を表す。
　　…「モノ」を中心とした概念

・ユビキタス：ユーザーが時間や場所にとらわれずインターネットに繋がって様々なサービスを受けられる状態を表す。
　　…ユーザーという「人」を中心とした概念

参考

ユビキタス (ubiquitous) は、遍在（いつでもどこでも存在すること）をあらわす言葉。

パロアルト研究所のマーク・ワイザーが、1991年の論文『The Computer for the 21st Century』にて、コンピュータやネットワークなどの遍在を表す意味合いで用いた。以来、ユビキタスコンピューティングやユビキタスネットワーク、さらにはそれらが当たり前になった社会を指す「ユビキタス社会」の意味で用いられるようになった。

Wikipedia

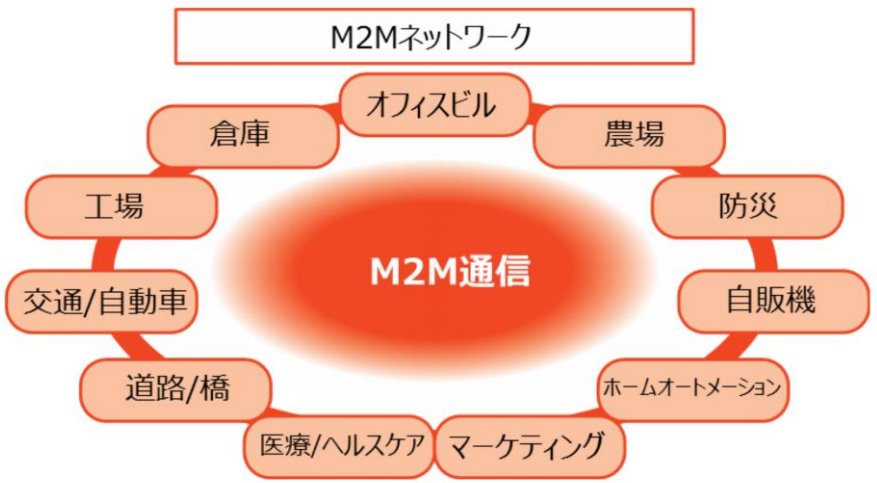
ここからは、IoTと似た概念であるユビキタスとM2Mについて確認し、その違いをみていきましょう。

まず、ユビキタスについてです。ユビキタスは、2000年頃に登場した概念で、人がどこにいてもネットワークにつなげる（つながる）状態を指します。ユビキタスネットワークの中心は人なので、IoTのモノとモノとの相互制御とはその点で異なります。

IoTとM2M

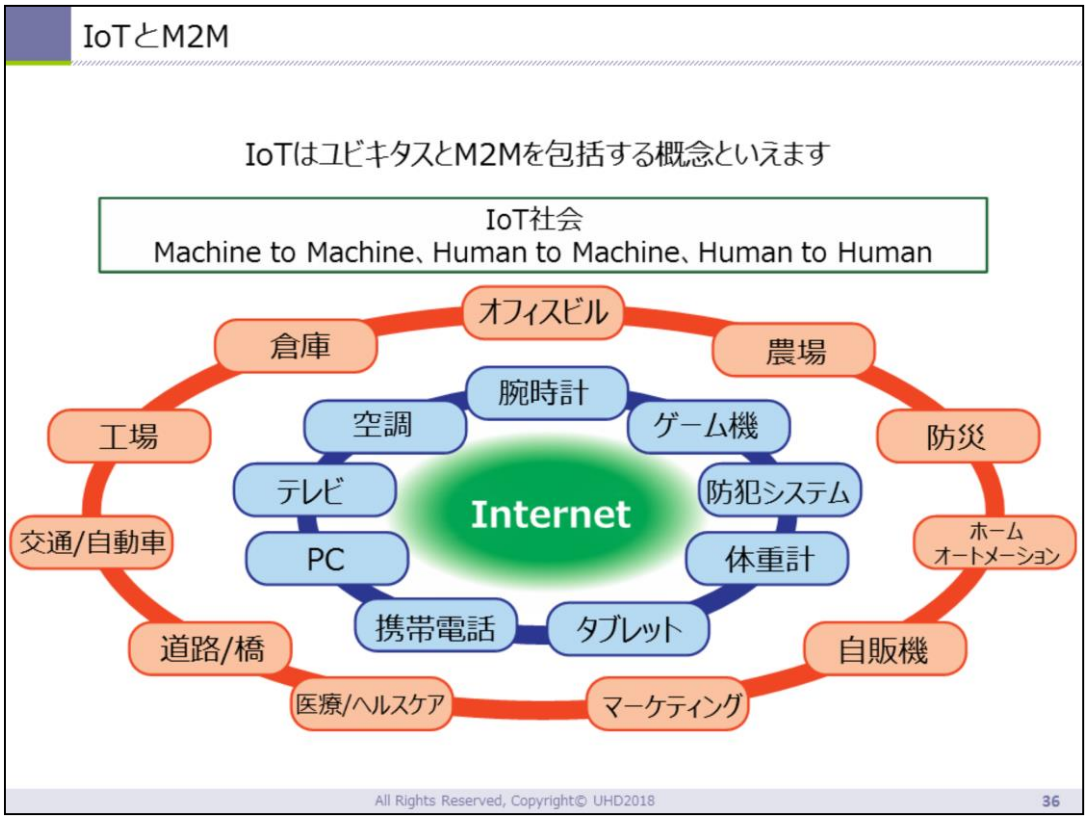
- **M2M = Machine to Machine**

機械同士が、人を介在しないで情報をやり取りするシステムのことをいいます。



次にM2Mをみていきましょう。

M2Mは人が介在しない機械同士の連携です。機会が情報を集め、他の機械にその情報を提供しますが、人が分かる形でデータを表示することはありません。IoTは人が介在するセンサーも対象となりますし、結果をみるのも人である場合もあるため、M2Mの概念だけではIoTを説明することはできません。



　ここまで学んできたように、モノ自体がインターネットを形作るIoTは、ユビキタスとIoTを合わせたような概念といえます。人とモノ、モノとモノ、人と人をつなぐ総合的な社会インフラになる可能性を秘めています。

IoTを構成する技術要素

- センサー：物理的な現象を検知し、電気信号として出力する装置。
- デバイス：センサーが組み込まれることによって、ネットワークに接続された装置やモノ。
  例　スマホ、時計、メガネ
- ネットワーク：デバイスをIoTサービスに繋ぐ、あるいはデバイス同士を繋ぐことでデータを共有、処理するシステム。
- IoTサービス：①デバイスとのデータの送受信 … IoT
  ②データの処理と保存 … ビッグデータの技術的守備範囲
  を行うサービス。
- データ分析：蓄積したデータについて統計分析や機械学習を行う。
  →最適な判断や行動方針を導き出す。

IoTには基本的に、センサー、デバイス、ネットワーク、サービス、データ分析がセットになって、IoTサービス全体を形作っています。それぞれの技術要素は独立していることが多く、技術要素の組み合わせによって、IoTを実現します。
IoT構成要素として欠かせない、データの処理と保存はまさにビッグデータの技術的守備範囲となります。IoTの実現にはビッグデータ技術が必要となります。逆にIoTを知ることでビッグデータの現実を知ることになります。

IoTを構成する技術要素　センサー

■ **センサー**

物理的な現象を検知し、電気信号として出力する装置のことをいいます。
多くの場合、一つのデバイスに対して複数のセンサーが埋め込まれています。

・画像センサー：光を捉えて処理することで、画像や動画を撮影する。
　赤外線を検知して画像処理するものもある。
・光センサー：光の強度を測定する。
・温度センサー：温度を測定する。
・湿度センサー：湿度を測定する。
・振動/速度/加速度センサー：機器の振動や速度、加速度を測定する。
・地磁気センサー：地磁気を検出することで、方角を計測する。
・ジャイロセンサー：デバイスの傾きを検知する。
・音声マイク：機器が発する音や、人の声などの音声を収集する。

　IoTのセンサーにはどのようなものがあるかみてみましょう。データの中には、画像データや、音声データのような非構造データや、準構造化データ、バイナリーデータも存在します。

IoTを構成する技術要素　センサー

センサーの代表的なデータフォーマットとしては、
・XML　・JSON　・MessagePack
があります。

XML
```
<xml>
  <info>
    <id>12996</id>
    <name>RoomSensor</name>
    <date>20170123112255</date>
  </info>
  <data>
    <temperature>27.8</temperature>
    <humid>72</humid>
  </data>
</xml>
```

人が読んで分かりやすい
データ量が多い

JSON
```
{"info":{"id":123,
          "name":"RoomSensor",
          "date":20170123112255
         },
 "data"{"temperature": 27.8,
          "humid":72
         }
}
```

データ量が少ない

いずれも各言語のライブラリが充実していますが、文字データであることから、
パース（解析）をしないとプログラムで利用できません。
MessagePackは、バイナリデータをそのまま扱いたい場合、有利です。

センサーデータの基本的なフォーマットを見てみましょう。タグ構造のXMLや、Webサイトでよく使われるJSON形式もありますが、MessagePackのようなバイナリーデータも存在します。「センサーデータは、構造化データでないことが多い」ということを覚えておいてください。

IoTを構成する技術要素　センサー

MessagePack

・センサーの代表的なデータフォーマットの一つ。
・JSONと似た形式だが、値はバイナリのままである。
・軽量でプログラム間処理に向いている。

MessagePackの特徴

・シリアライズ（データの直列化）、デシリアライズ：非常に高速
・シリアライズされたデータのサイズ：小さい
・フォーマット定義：不要
・ストリーム処理：可能

JSONとMessagePackの比較

JSON　{“a”:null,”b”:10,”c”:[20],”d”:”30"}　… 35byte

→ MessagePack に変換すると…

84 a2 61 c0 a2 62 0a a2 63 91 14 a2 64 a2 33 30　… 16byte

　MesssagePackの形式をみてみましょう。人がみて理解できる形式ではありませんが、データ量が小さく、変換が高速であることが魅力です。「IoTデータにはバイナリーデータも存在する」ということを覚えておいてください。

IoTを構成する技術要素　デバイス

■ **デバイス**

センサーが組み込まれることによって、ネットワークに接続された装置、モノのことをいいます。例えば、スマホ、時計、メガネはいずれもデバイスです。

■ **デバイスの２つの機能**

・センシング：センサーを利用して、デバイス自身や周りの環境の状態を収集し、IoTシステムに通知すること。

例・画像センサーによる人の有無の検知
・スマホの位置情報や加速度の計測

・フィードバック：システムからの通知を受け、指示や動作をもとのシステムに返すこと。次のような方法がある。

・可視化：センシング結果表示、デバイスの管理、画面表示
・通知：システムが判断した結果を画面に表示
・制御：デバイス自身や環境の状態そのものを変更

　ここでは、IoTにおけるデバイスとは何かをみていきましょう。

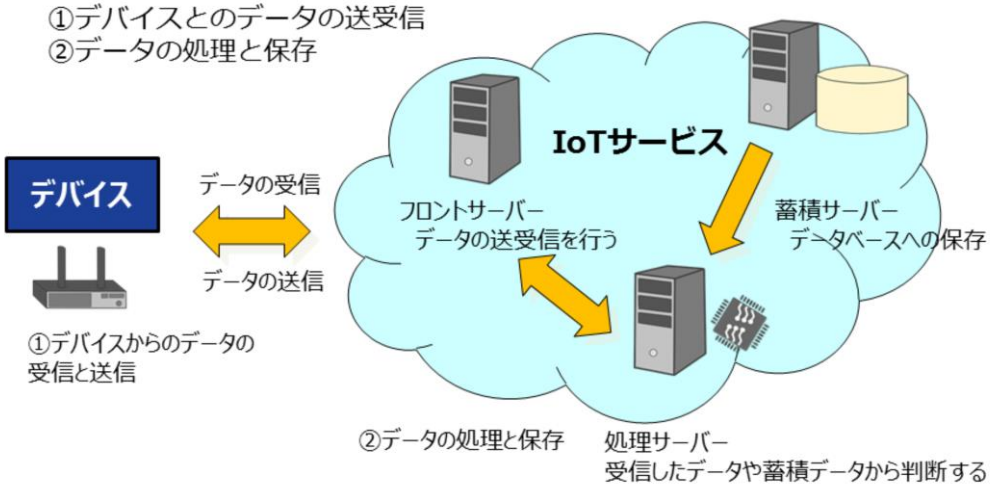
IoTにおいてデバイスとは、センサーとネットワーク接続を併せ持った機器を指します。スマートフォンやスマートウォッチに搭載されるセンサーとネットワーク通信機能もデバイスとして捉えることができます。デバイスには2つの機能が要求されており、1つは周囲の情報をシステムに伝えるセンシング、もうひとつはシステムからの指示によって動作を行うフィードバックです。システムにとっては、デバイスはインプット（センシング）とアウトプット（フィードバック）の両方を担う存在と言えるでしょう。

IoTを構成する技術要素　IoTサービス

- **IoTサービス**

  以下を行うサービスのことを指します。

  ①デバイスとのデータの送受信
  ②データの処理と保存



　IoTサービスはデバイスから送られてきたデータを受信したり、蓄積、分析したり、サービスを行ううえで必要な処理を行うサーバーサイドの処理です。サーバーの主なものは下記です。

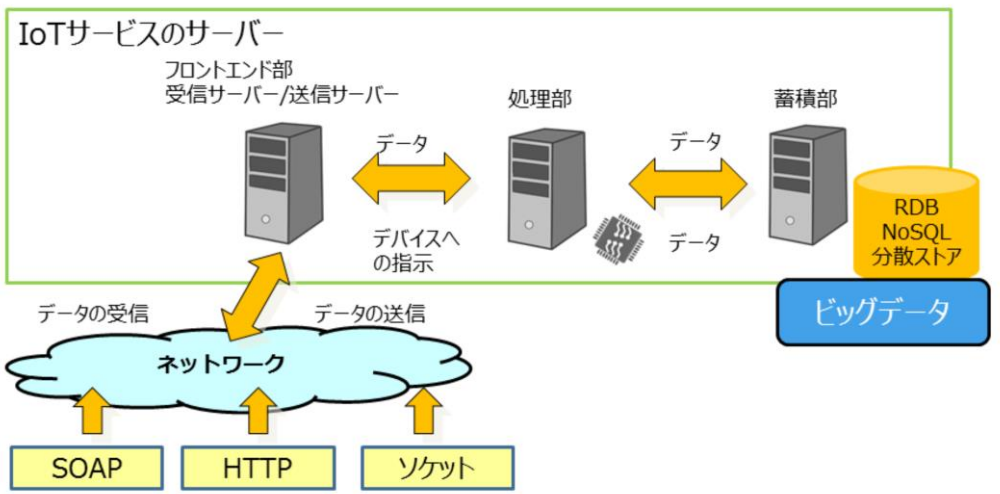
データの送受信を担当するフロントサーバー
データベースへの保存を行う蓄積サーバー
判断のためのデータ処理を行う処理サーバー

IoTで処理されるデータは、膨大であり構造化されていないものも多いため、ここで処理されるデータは、まさにビックデータといえるでしょう。

IoTを構成する技術要素　IoTサービス

■ **サーバー構成**

IoTサービスの役割は、フロントエンド、処理、蓄積の大きく３つに分けられます。
蓄積されるデータは膨大な量となります。



　IoTのサーバー側の役割をもう少し詳しくみてみましょう。

サービスの奥にある蓄積部（蓄積サーバー）では、NoSQLや分散処理システムなど、非構造化データを大量に処理できる基盤が利用されています。大量の非構造化データは、当然ビックデータの定義に当てはまります。つまり、IoTのサーバー構成は、ビックデータを扱える能力が要求されているのです。

ビッグデータ
Redisによるデータ管理
演習資料

演習 1

高速 NoSQL データベース Redis を利用して、ビッグデータ技術に触れましょう。

Redis とは

Redis は 2009 年にイタリアの個人プログラマーSalvatore Sanfilippo 氏によって開発され始めたインメモリによるキーバリュー型 NoSQL データベースです。現在は Redis Lab が開発主体となっています。

特徴

- ・マスター型のキーバリューストア。単一障害点があるものの、マスターがクラッシュした際にスレーブが、マスターに昇格します。（クラッシュしていないスレーブが残っている場合）
- ・マスターは複数のスレーブを持つことができ、非同期でスレーブに複製を作成します。
- ・コンシステント・ハッシングを用いたシャーディングをクライアント側で行います。
- ・オンメモリでデータが処理されるため、書き込み、読み込みが高速。書き込んだデータは一定時間が過ぎると非同期でディスクに保存されます。ディスクに永続化する前に電源が落ちるとデータは消失します。永続化は RDB に書き込む機能と AOF（Append Only File）によるコマンドの記録ファイルに書き込む機能のいずれかが選択できます。
- ・データストラクチャ・ストアとも呼ばれ、画像などのバイナリ、文字列、リスト、セット、ソート済みのセット、ハッシュなどといった多彩なデータを格納することができます。
- ・アトミックな処理（排他制御を行う事）によって更新をかけることができます。
- ・結果整合性　マスターからスレーブに複製するまでの間、システム内には複数のバージョンが存在します。つまり、最終的に結果が合えばよいという結果整合性の考え方を取ります。
- ・ハイパフォーマンス、レプリケーションなどの機能を要求される場合など有利。
- ・スケーラビリティが高いため、プロトタイピングに便利

デメリット

- ・Replace などの部分置換や条件抽出、集計などの複雑な操作は用意されていません。アプリ側か Lua スクリプトで記述する必要があります。
- ・ロールバック機能が存在しません。
- ・厳密な一貫性は担保されません。
- ・セキュリティ機能に乏しい。信頼されたアクセスを前提としているため、必要最低限です。
  アクセス制限、リモート攻撃は別途対策が必要。（コマンドのリネームは可能です。）

Redis の基本的なデータ型は 5 種類です。

型	格納されている情報	例
STRING 型	512 メガバイトまでのデータ(バイナリセーフ) 文字列、整数、浮動小数点数(数値は加減算可能)	”Redis” 10 1.2 など
LIST 型	文字列の連結リスト（42 億個の要素）先頭、末尾の追加は高速	1,2,3,1
SET 型	一意な文字列の順序のないコレクション 重複を許さないため、多彩な集合演算機能が搭載されている	1,5,2
HASH 型	順序のない、キーと値のハッシュテーブル	field1＝1,field2＝199 など項目名と値のペア
ZSET 型 （ソート済み集合）	浮動小数点数のスコア順に並べられた文字列 メンバーからスコアへのマッピング	東京：スコア 1, 大阪：スコア 2, 名古屋：スコア 3 などスコアの順序と共に格納

Redis で利用できるクエリ
先頭に L がついている場合は Lists、S の場合は Sets、Z の場合は Sorted Sets、H の場合は Hashes をそれぞれ操作するコマンドになっています。

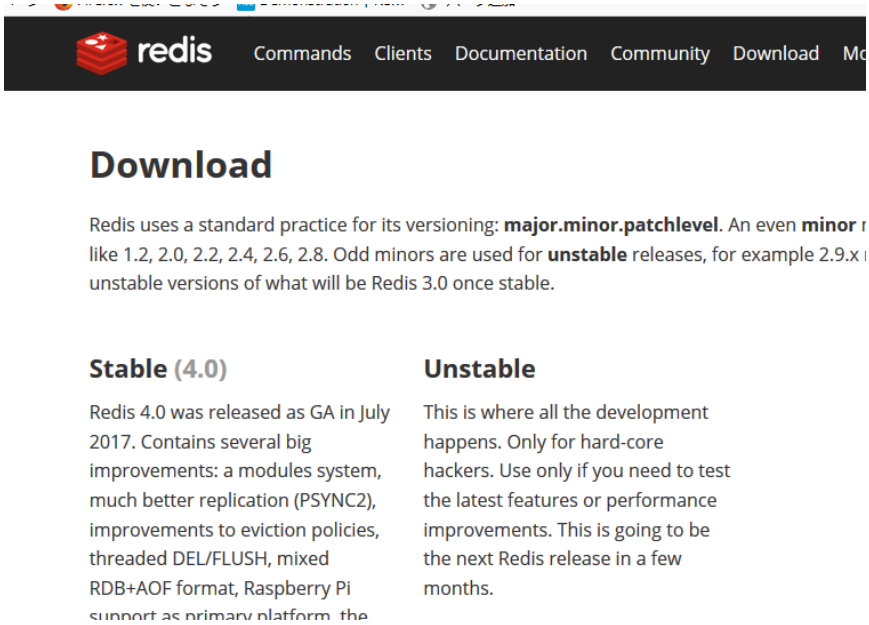
コマンド	説明
SET	キーとそれに対するバリューを指定して、新規に格納するか、既に格納済みの場合は更新する
GET	キーを指定して、バリューを取得する
DEL	キーを指定して、バリューを削除する
INCR	キーを指定して、数値バリューに 1 を足す
MSET	複数キーとそれぞれに対するバリューを指定して、新規に格納するか、既に格納済みの場合は更新する
MGET	複数キーを指定して、それぞれのキーに対するバリューを一括で取得する
LPUSH・RPUSH	キーで指定した List に対して、左端、右端に要素を追加する
LLEN	キーで指定した List の長さを取得する
LRANGE	キーとインデックスを二つ（開始・終了点）指定する事で、キーに対する List の 2 つのインデックス間の要素を取得する
LPOP・RPOP	キーで指定した List の左端、右端からそれぞれ要素を抜き出す。

コマンド	説明
SADD	キーと要素を指定して、指定した Set に対して要素を追加する
SREM	キーと要素を指定して、キーに対する Set から指定した要素を削除する
SISMEMBER	キーと要素を指定して、指定した要素が Set 内に存在するか確認する
SMEMBERS	キーと要素を指定して、Set の要素全てを取得する
SUNION	キーと要素を指定した 2 つ以上の Set を統合した結果を返す。重複は省かれる。
ZADD	キーとバリュー、スコアを指定してキーに対応するソート済み Set にバリューとスコアのエントリを追加する
ZRANGE	キーとインデックスを二つ（開始・終了点）指定することでキーに対するソート済み Set のインデックス間の要素を取得する
HSET	キーとフィールド、バリューを指定して、キーに該当した Hash にフィールドとバリューのエントリを追加する
HGETALL	キーで指定した Hash の要素全てを抜き出す。
HMSET	キーと複数のフィールド、バリューを指定して、キーに対する Hash に指定しフィールドとバリューのエントリをそれぞれ追加する
HGET	キーとフィールドを指定して、キーに対する Hash のフィールドのバリューを取得する
HINCRBY	キーとフィールド、数値を指定すると、キーに対応する Hash の中のフィールドに対応するバリューに指定した数値を加算する
HDEL	キーとフィールドを指定して、キーに対応する Hash の指定したフィールドとバリューを削除する

参考：Redis のインストール

１．インストーラーのダウンロード

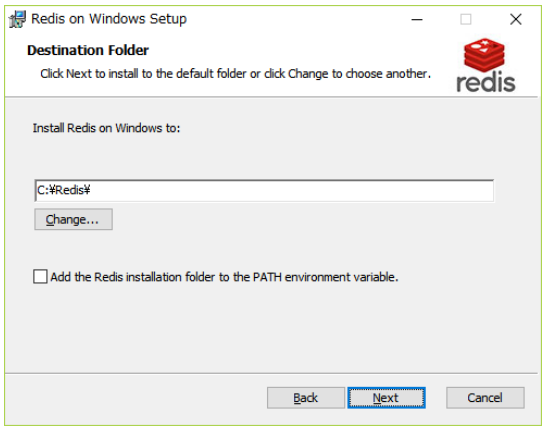
https://redis.io/download

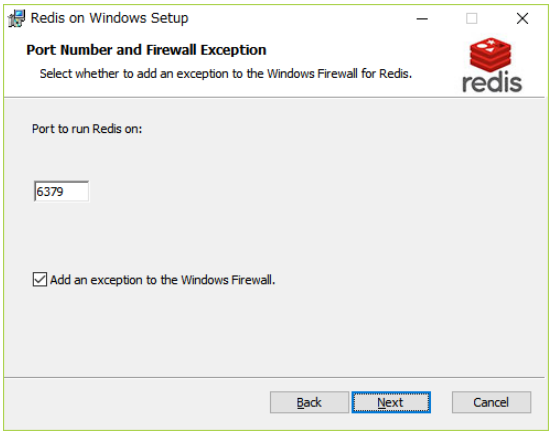


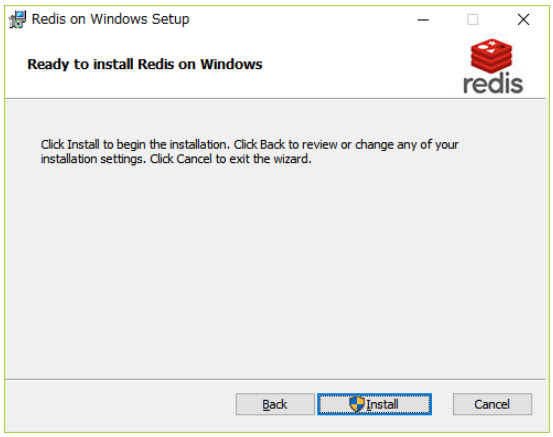
Windows 版は GitHub のリンクから msi ファイルを入手します。

Redis-x64-3.0.500-rc1.msi

２．インストールの実行







３．インストールの確認

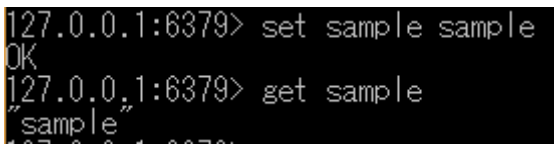
- path の確認
  - > path を実行して環境変数を表示します
  - C:¥develop¥Redis が表示されれば大丈夫です
- Redis の確認　（サービスが起動しているか確認しましょう。）

>redis-cli を実行。

以下のように入力してみましょう

> set sample sample

> get sample "sample"

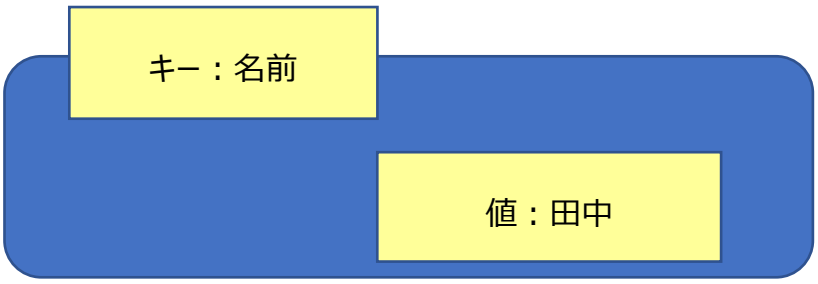


演習　Redis の各型について、データの出し入れを行いましょう。

①String 型の保存イメージ

型	格納されている情報	例
STRING 型	512 メガバイトまでのデータ(バイナリセーフ) 文字列、整数、浮動小数点数(数値は加減算可能)	”Redis” 10 1.2 など

String 型



Redis の型操作（String）

String 型の操作

以下のように入力してみましょう。

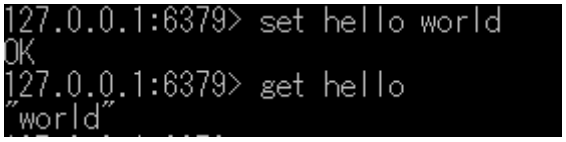
>set hello world

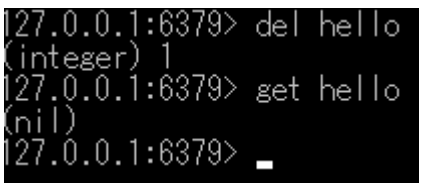
>get hello

>del hello

>get hello

結果



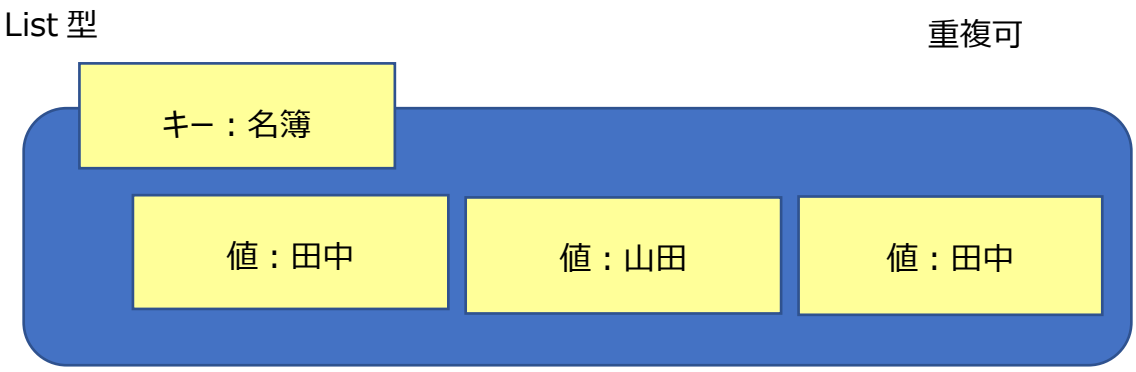


処理成功は 1、処理されない場合は 0 が返ります。

削除されたら値がないため、nil＝空が返る

②List 型の保存イメージ

型	格納されている情報	例
LIST 型	文字列の連結リスト（42 億個の要素）先頭、末尾の追加は高速	1,2,3,1



Redis の型操作（List）

List 型の操作

>rpush list-key item
>rpush list-key item2

>rpush list-key item
>lrange list-key 0 -1
>lindex list-key 1
>lpop list-key
>lrange list-key 0 -1

結果

```
127.0.0.1:6379> rpush list-key item
(integer) 1
127.0.0.1:6379> rpush list-key item2
(integer) 2
127.0.0.1:6379> rpush list-key item
(integer) 3
```

```
127.0.0.1:6379> lrange list-key 0 -1
1) "item"
2) "item2"
3) "item"
```

先頭を 0　末尾を-1 指定すると
全リストを返します。

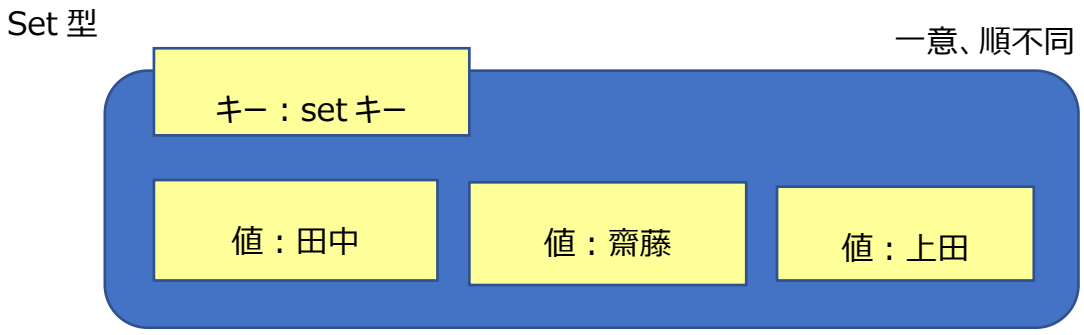
```
127.0.0.1:6379> lindex list-key 1
"item2"
```

```
127.0.0.1:6379> lpop list-key
"item"
127.0.0.1:6379> lrange list-key 0 -1
1) "item2"
2) "item"
```

リストの左端から要素を取りだしたため、リストから削除されます

③Set 型の保存イメージ

型	格納されている情報	例
SET 型	一意な文字列の順序のないコレクション 重複を許さないため、多彩な集合演算機能が搭載されている	1,5,2



Redis の型操作（Set）

Set 型の操作

```
>sadd set-key item
>sadd set-key item2
>sadd set-key item3

>sadd set-key item
>smembers set-key
>sismember set-key item4
>sismember set-key item
>srem set-key item2
```

>srem set-key item2
>smembers set-key

```
127.0.0.1:6379> sadd set-key item
(integer) 1
127.0.0.1:6379> sadd set-key item2
(integer) 1
127.0.0.1:6379> sadd set-key item3
(integer) 1
127.0.0.1:6379> sadd set-key item
(integer) 0
127.0.0.1:6379> smembers set-key
1) "item"
2) "item3"
3) "item2"
```

リストに追加された場合は 1 が返る

重複のためリストに追加されなかった
場合は 0 が返る

```
127.0.0.1:6379> sismember set-key item4
(integer) 0
127.0.0.1:6379> sismember set-key item
(integer) 1
127.0.0.1:6379> srem set-key item2
(integer) 1
127.0.0.1:6379> srem set-key item2
(integer) 0
127.0.0.1:6379> smembers set-key
1) "item3"
2) "item"
```

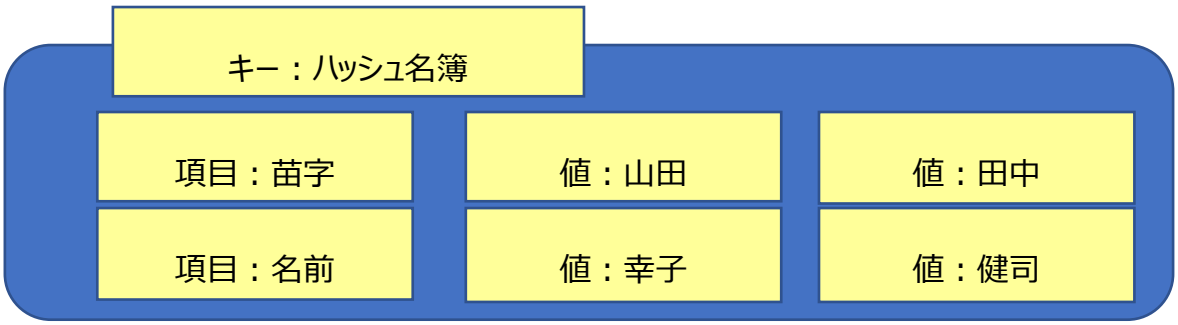
item4 は存在しないため 0 が返る

item2 は存在しないため 0 が返る
Item2 が消えていることを確認

④Hash 型の保存イメージ

型	格納されている情報	例
HASH 型	順序のない、キーと値のハッシュテーブル	field1=1,field2=199 など項目名と値のペア

Hash 型



Redis の型操作（Hash）

Hash 型の操作

>hset hash-key sub-key1 value1

>hset hash-key sub-key2 value2

>hset hash-key sub-key1 value1

>hgetall hash-key

>hdel hash-key sub-key2

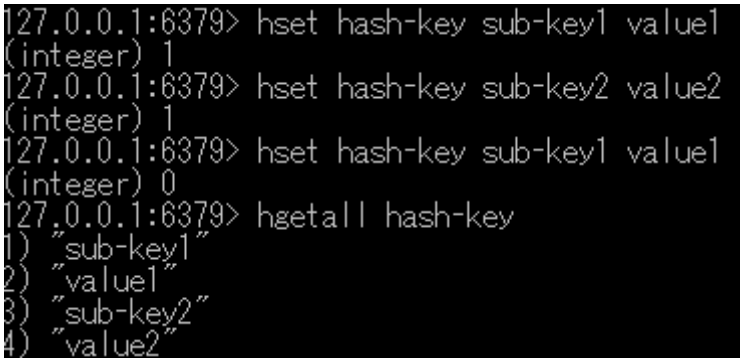
>hdel hash-key sub-key2

>hget hash-key sub-key1

>hget hash-key sub-key2

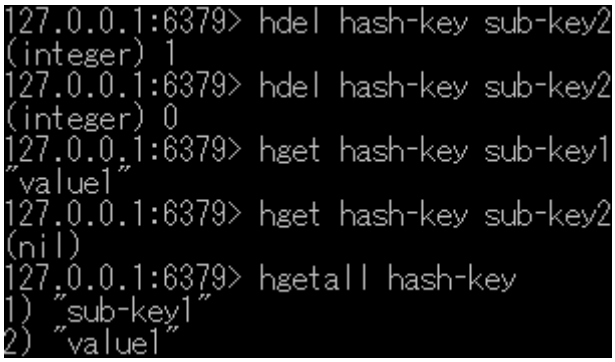
>hgetall hash-key

複数の項目にアクセスできることから、RDB の行に近い構造です



Hash が追加できたら 1、できない場合は 0 が返る

Hash の全ての要素をフェッチ

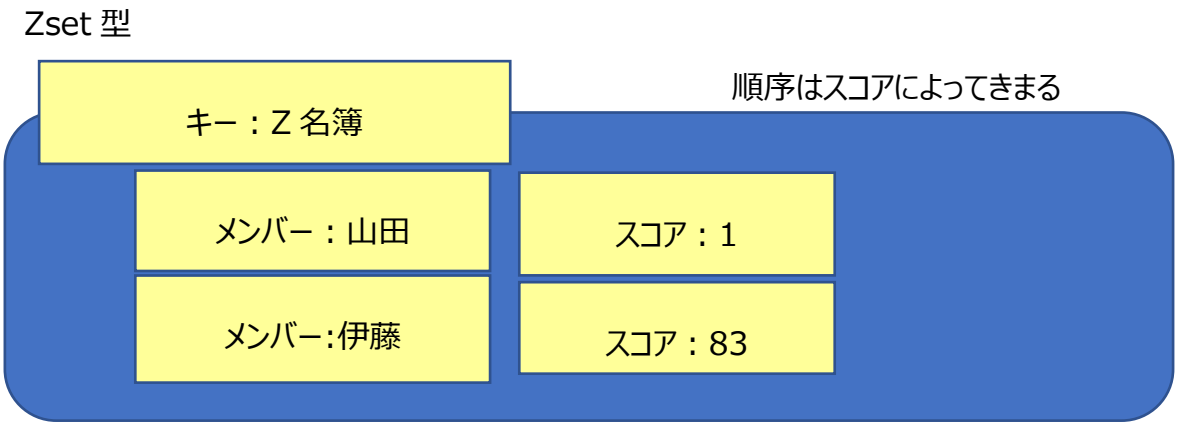


Hash が削除できたら 1、できなかった場合は 0 が返る

削除済みの Hash を取得すると nil が返る

⑤Zset 型の保存イメージ

型	格納されている情報	例
ZSET 型 （ソート済み集合）	浮動小数点数のスコア順に並べられた文字列 メンバーからスコアへのマッピング	東京：スコア 1, 大阪：スコア 2, 名古屋：スコア 3 などスコアの順序と共に格納



Redis の型操作（Zset）

Zset 型の操作

```
>zadd zset-key 887 member1
>zadd zset-key 912 member0
>zadd zset-key 912 member0
>zrange zset-key 0 -1 withscores
 >zrangebyscore zset-key 0 900 withscores
 >zrange zset-key 0 900 withscores
 >zrem zset-key member1
 >zrem zset-key member1
 >zrange zset-key 0 -1 withscores
```

結果

```
127.0.0.1:6379> zadd zset-key 887 member1
(integer) 1
127.0.0.1:6379> zadd zset-key 912 member0
(integer) 1
127.0.0.1:6379> zadd zset-key 912 member0
(integer) 0
```

```
127.0.0.1:6379> zrange zset-key 0 -1 withscores
1) "member1"
2) "887"
3) "member0"
4) "912"
```

```
127.0.0.1:6379> zrangebyscore zset-key 0 900 withscores
1) "member1"
2) "887"
```

スコアの範囲で要素を取得できる

```
127.0.0.1:6379> zrem zset-key member1
(integer) 1
127.0.0.1:6379> zrem zset-key member1
(integer) 0
127.0.0.1:6379> zrange zset-key 0 -1 withscores
1) "member0"
2) "912"
```

参考：Python プログラムから Redis にアクセスする方法

Python から Redis を呼ぶためには Redis クライアントライブラリをインストールする必要があります。
インストールを簡単にするために、easy_install を使用します。

パイソンを起動
c:¥python27¥python

パッケージ ez_setup.py をネットから探し、ダウンロードする
from urllib import urlopen
data = urlopen('http://peak.telecommunity.com/dist/ez_setup.py')
open('ez_setup.py','wb').write(data.read())
パイソンを終了
exit()

ez_setup.py を起動

c:¥python27¥python ez_setup.py

Redis クライアントをインストール

c:¥python27¥python -m easy_install redis

実行画面

プログラムを実行してみましょう。

C:¥Users¥uni03>c:¥python27¥python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

☆Redis をインポート

>>> import redis

☆Redis へのコネクション生成

>>> conn = redis.Redis()

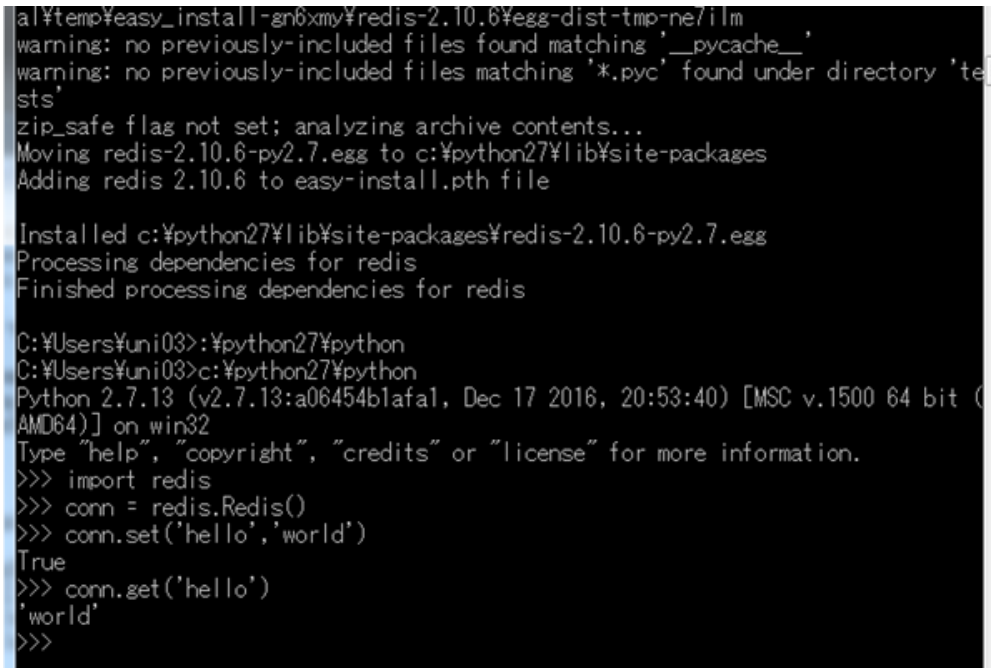
☆コマンドをセットして送信

>>> conn.set('hello','world')
True

☆コマンドをセットして送信

>>> conn.get('hello')
'world'

Python プログラムの中でも同様に記述できますが、操作を簡単にするためにコマンドラインで実行しています。



ビッグデータ
Spark 演習資料

Spark とは

Spark は 2009 年にカリフォルニア大学バークレー校の AMPLab により、大規模データ処理のための分散処理プラットフォームとして開発が開始され、2014 年に Apache Software Foundation に寄贈されました。

Spark は Hadoop にとって代わるものではなく、MapReduce にとって代わる存在です。
Hadoop が Java 言語で作られているのに対して Spark は Java の派生言語である Scala で作られています。

Spark の特徴

インメモリ処理による高速化

「データの格納場所」に関する選択肢の広さ

Hadoop Distributed File System (HDFS)

Cassandra

OpenStack Swift

Amazon S3

プログラム手法に関する選択肢の広さ

・Java

・Python

・R 言語

また、Spark は「Spark SQL」や DataFlow で SQL 言語を扱えます。

Spark と Hadoop の関係は共存関係

・「Hadoop+Spark」の構成も現実的である

(Hadoop 内の Yarn 制御下で Spark を利用する)

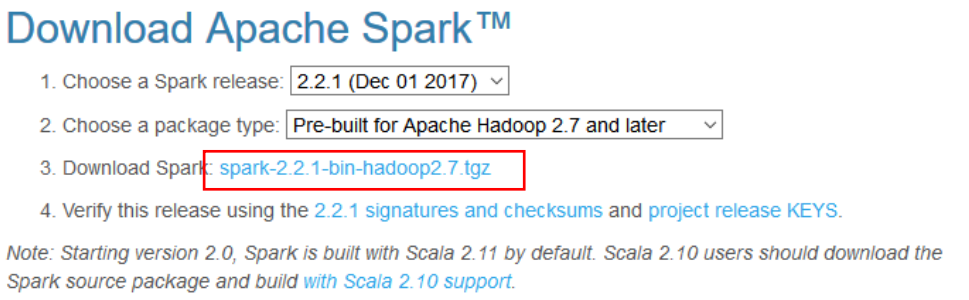
・Spark はデータの入出力場所として HDFS にも対応している

Spark のインストール

1．Java をインストールし、実行できるようにします。

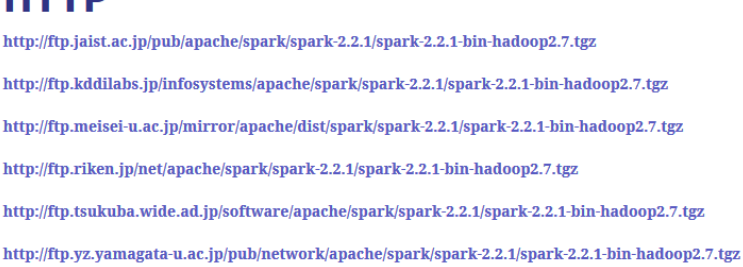
2．Apache の Spark サイトにアクセスします

https://spark.apache.org/downloads.html



3．最新のバージョンの tgz をダウンロードし、

配置したい場所に解凍します。

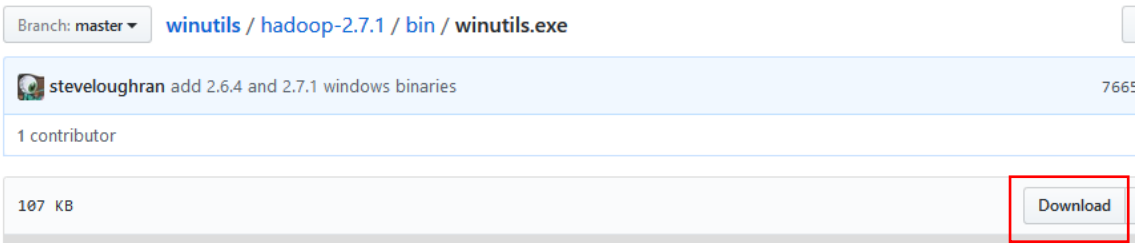


4．Windows で Spark を実行するために、Hadoop の依存関係を解決するための exe を

取得します。

winutils.exe

https://github.com/steveloughran/winutils/blob/master/hadoop-2.7.1/bin/winutils.exe



５．ダウンロードした exe ファイルを、展開後の Spark パッケージの bin の下に配置します。

６．環境変数の設定を行います。

- PATH: Spark インストール先¥bin を追加
- HADOOP_HOME: Spark インストール先

Spark コマンドラインの起動

コマンド・プロンプトを開いて、spark-shell を実行します。

C:¥Users¥XXXX>spark-shell

①簡単なサンプルコードの起動

ReadMe に含まれる英単語をカウントしてみましょう。README.md ファイルは Spark インストールディレクトリの直下にあります。

テキストエディタで開いてみます

コマンドラインから以下を入力してみましょう。

val lines = sc.textFile("C:/spark-2.2.0/README.md")

```
scala> val lines = sc.textFile("C:/spark-2.2.0/README.md")
lines: org.apache.spark.rdd.RDD[String] = C:/spark-2.2.0/README.md MapPartitionsRDD[3] at textFile at <console>:24
```

1 コマンド目の val は RDD 宣言を表します。textFile メソッドで README.md ファイルを読み込み、lines という名前の RDD を宣言します。

lines.count()

```
scala> lines.count()
res0: Long = 103
```

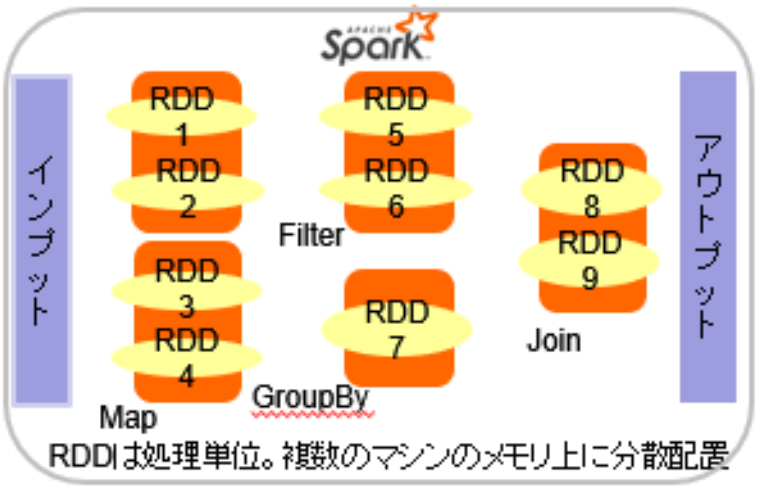
lines.first()

```
scala> lines.first()
res1: String = # Apache Spark
```

Count 関数で行数を、first 関数で最初の行を取得しています。

RDD とは

RDD(Resilient Distributed Dataset) は Spark の主要なデータの処理単位です。これは、クラスタ全体で並列化された要素の分散集合です。RDD は、YARN クラスタの複数の物理ノードのメモリ上に分割、分散された並列操作可能なオブジェクトの集まりです。



RDD を作成する方法は 3 つあります。

- 既存のデータ集合を並列化します。データがすでに Spark 内に存在し、並列で操作できる状態であれば可能です。
- データセットを参照して RDD を作成します。 このデータセットは、HDFS、Cassandra、HBase などの Hadoop でサポートされているすべてのストレージソースから取得できます。
- 既存の RDD を変換して新しい RDD を作成して RDD を作成します。

②簡単なサンプルコードの起動

RDD を加工して、スペース区切りにし、それを word という(キー,1 回)という Map に格納し、reduceByKey メソッドで同じキーの出現回数を集計してみましょう

val wordCounts = lines.flatMap(line => line.split(" ")).map(word => (word,1)).reduceByKey((a,b) => a + b)

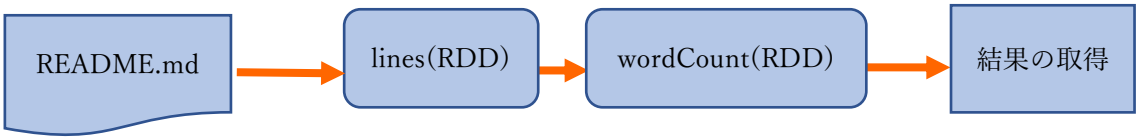
```
scala> val wordCounts = lines.flatMap(line => line.split(" ")).map(word => (word,1)).reduceByKey((a,b) => a + b )
wordCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[9] at reduceByKey at <console>:26
```

結果を collect メソッドで取得してみます。

wordCounts.collect()

```
scala> wordCounts.collect()
res5: Array[(String, Int)] = Array((package,1), (this,1), (Version"](http://spark.apache.org/docs/latest/building-spark.html#specifying-the-hadoop-version),1), (Because,1), (Python,2), (page](http://spark.apache.org/documentation.html).,1), (cluster.,1), (its,1), ([run,1), (general,3), (have,1), (pre-built,1), (YARN,,1), (locally,2), (changed,1), (locally.,1), (sc.parallelize(1,1), (only,1), (several,1), (This,2), (basic,1), (Configuration,1), (learning,,1), (documentation,3), (first,1), (graph,1), (Hive,2), (info,1), ([Specifying,1), ("yarn",1), ([params]`.,1), ([project,1), (prefer,1), (SparkPi,2), (<http://spark.apache.org/>,1), (engine,1), (version,1), (file,1), (documentation,,1), (MASTER,1), (example,3), (["Parallel,1), (are,1), (params,1), (scala>,1), (DataFrames,,1), (provides,...
```

Lines の RDD から、別の RDD を生成して利用しています。



③簡単なサンプルコードの起動

Scala という単語を含む行を探索し、結果を表示してみましょう。

val filterdLines = lines.filter(line => line.contains("Scala"))

```
scala> val filterdLines = lines.filter(line => line.contains("Scala"))
filterdLines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[10] at filter at <console>:26
```

件数を表示します

filterdLines.count()

```
scala> filterdLines.count()
res6: Long = 3
```

filterdLines.first()

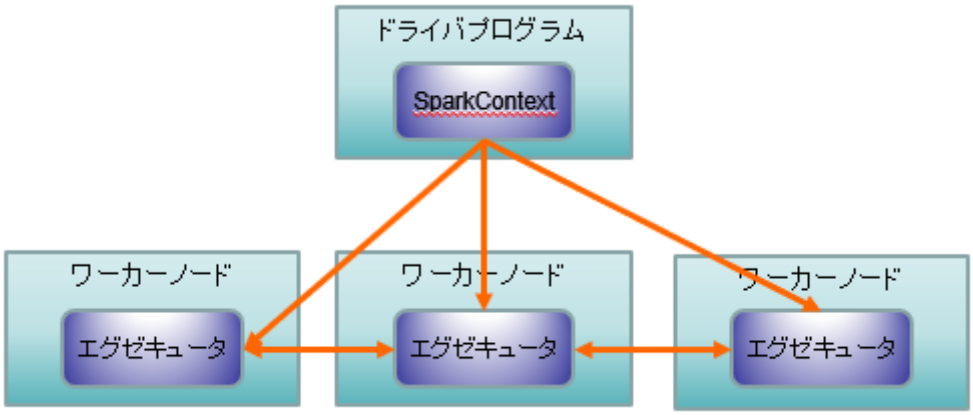
```
scala> filterdLines.first()
res7: String = high-level APIs in Scala, Java, Python, and R, and an optimized engine that
```

filterdLines.collect()

```
scala> filterdLines.collect()
res8: Array[String] = Array(high-level APIs in Scala, Java, Python, and R, and an optimized engine that, ## Interactive
hell, The easiest way to start using Spark is through the Scala shell:)
```

Spark の内部動作（ワーカーノードとエグゼキューター）

Spark アプリケーションは「Driver Program」によって全体の処理を制御します。これまでの RDD 生成や変換の処理を記述したものがドライバプログラムです。ドライバプログラムは実際に分散処理を実行する「Worker Node（ワーカーノード）」上の「Executer（エグゼキューター）」を管理しており、エグゼキューターがプログラム内容を実行します。



Spark の内部動作（SparkContext）

今回の演習では、コマンドラインからローカルマシン 1 台で処理を行っていますが、クラスタ環境で実行した場合は、同じドライバプログラムで複数のワーカーノードに分散されて実行されます。
分散実行を実現するために、SparkContext が用意されています。 SparkContext オブジェクトはそれぞれのマシンが管理する Spark へ接続し、実行します。
今回利用しているインタラクティブシェルでは、起動時に SparkContext が起動されています。

sc と入力して実行してみましょう。

```
scala> sc
res9: org.apache.spark.SparkContext = org.apache.spark.SparkContext@802d25
```

SparkContext が存在することが確認できます。

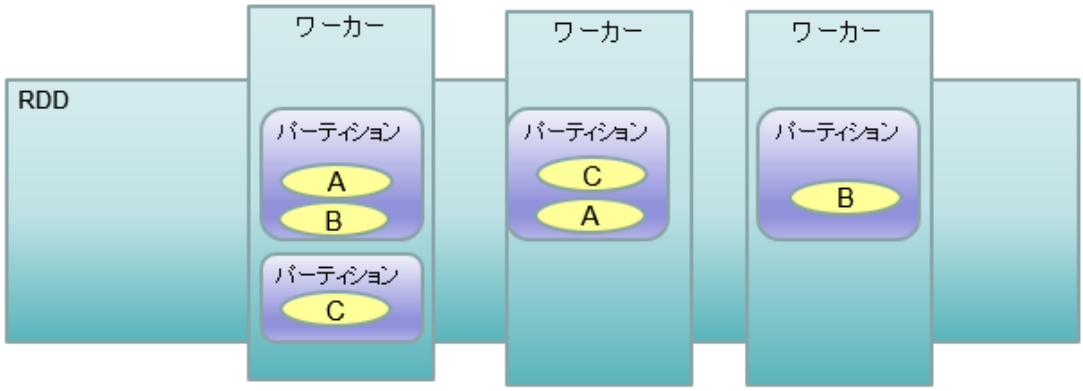
Sc.master でマスタ URL を調べると local[*]が指定されており、ローカルモードであることが分かります。

```
scala> sc.master
res10: String = local[*]
```

RDD の基本と構造

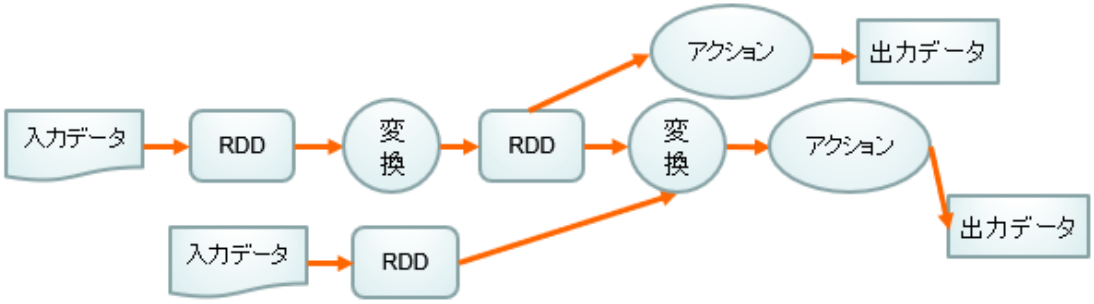
RDD（Resilient Distributed Dataset）
RDD はイミュータブルなデータ要素の分散コレクションです。
イミュータブルとは、一度作成したオブジェクトはその後状態を変えられない事を意味します。分散処理のため、どこかのワーカーで値を変えた場合、後続のワーカーに影響が出ることを避けるための制約です。

RDD はパーティションと呼ばれる単位に分割され、Spark クラスタの複数のノードに分散されて配置されます。パーティション単位で処理が行われるため、大量のデータでも複数のノードで分散して処理できます。



RDD の基本と構造

RDD（Resilient Distributed Dataset）
RDD はイミュータブルなデータ要素です。データの変更は RDD から新たな RDD の作成を繰り返す事で処理を実行します。



RDD の操作

RDD を「新規に」作成する方法は大きく分けて 2 種類あります。

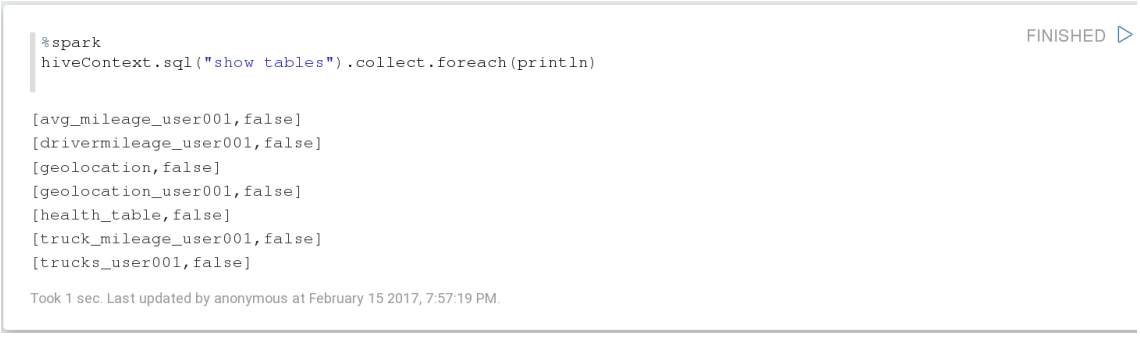
① テキストファイルやデータベースなどの外部データソースからデータをロードする方法
ファイルの例

```
val lines = sc.textFile("C:/spark-2.2.0/README.md")
```

【参考】Hive の例：コンテキストを生成し、sql メソッドで SQL クエリを発行します。

```
val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)
hiveContext.sql("show tables").collect.foreach(println)
```

以下は Zeppelin Notebook 上での実行結果です



② Driver Program で Array などのプログラミング言語が用意する通常のコレクション
から作成する。コレクションから RDD を作成するメソッドとして SparkContext の parallelize メソッドを使用します。

```
val nums = sc.parallelize(Array(3,5,3,2,1))
```

```
scala> val nums = sc.parallelize(Array(3,5,3,2,1))
nums: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[11] at parallelize at <console>:24
```

nums に配列を生成して値を代入した状態です。

parallelize の第 2 引数では、どのパーティションで作成するかを明示的に指定することもできます。ローカルの場合は、CPU のコア数になります。

```
val numsPar2 = sc.parallelize(Array(3,5,3,2,1),2)
```

RDD の操作には、変換（Transformation）とアクション（Action）の 2 種類があります。

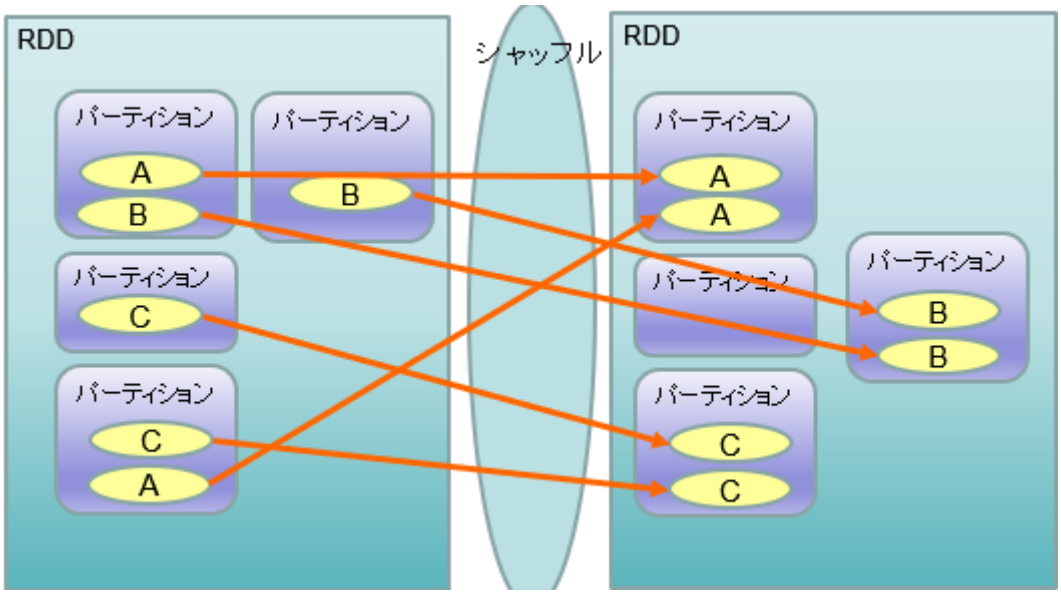
変換：RDD に処理を加えて新しい RDD を作成していく操作。map や filter など、一般的なプログラミング言語におけるコレクションへの操作のようなものです。

アクション：ワーカーでの処理結果をまとめ上げて、Scala での Array など通常のコレクションに変換したり、HDFS や AmazonS3 などの外部ストレージに値を保存したりする操作。

シャッフル

フィルターのようにワーカー内で完結する操作と、集計操作のようにデータを集めなければならない操作があります。その場合は Spark 内でシャッフル（Shuffle）という機能が働き、異なるパーティションに含まれる同一キーの要素を同一のパーティションにまとめる処理を行います

シャッフルのイメージ　Spark 裏側で動きますが、気が付かないところで大量のシャッフルやネットワーク間通信が発生する場合もあります。フィルターをかけてから集計をするなどの配慮が必要です。



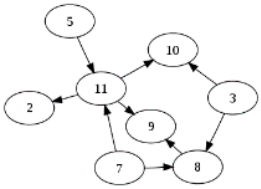
RDD の操作（遅延評価）

遅延評価・・・実際に計算する必要が生じるまで値を評価しない処理方式。

具体的にはアクションが実行されるまでは、変換操作は記録だけ行われるものの、実行自体は何も行われません。
アクションが実行された時点で RDD の変換を逆にたどって、まとめて処理されます。

Spark がメモリ上に大量のデータを乗せる仕組みであるため、不要なデータをメモリに保持したり、不要な処理を行ったりすることを回避することは重要な要素です。本当に必要な処理にリソースを集中させるための仕組みとなります。

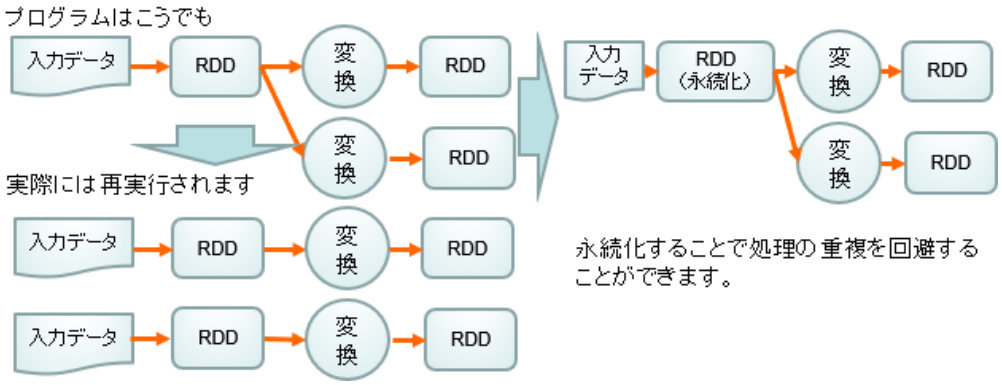
RDD の処理には処理 DAG（ Directed acyclic graph：有向非循環グラフ）を構築します。DAG は循環しない向きのある関係性で、終点から遡ると必ず一本道で始点へと到達できます。



DAG のイメージ　向きはあるが起点に戻らない

RDD の操作（永続化）

永続化　一度作成された RDD を再利用する場合に再度データの読み直しや再処理を
避けるため、RDD を明示的に永続化する事が可能です。
永続化する先はメモリの他、ディスクも選択可能、レプリケーション数や
シリアライズ方法の指定もできます。
シャッフル前の RDD は暗黙的にローカルファイルに永続化されます。



RDD の操作（RDD と DAG の関係）

Spark では RDD へのアクションが実行されると、論理実行プランとして RDD の DAG（有向非循環グラフ）を作成します。次に Spark は DAG を元に物理的な実行プランを構築します。
ノード間通信が発生するシャッフルを境界にして複数のステージに分割します。シャッフルが発生しない処理はなるべく一つのステージにグルーピングします。

グルーピングされたステージから、RDD のデータ配置とパーティションに基づいて構築される処理単位がタスクになります。

実行プランを toDebugString メソッドで確認します。

wordCounts.toDebugString

上記を実行すると以下のように表示されます。

```
scala> wor dCounts.toDebugString
res11: String =
(2) ShuffledRDD[6] at reduceByKey at <console>:26 []
 +-(2) MapPartitionsRDD[5] at map at <console>:26 []
    |  MapPartitionsRDD[4] at flatMap at <console>:26 []
    |  C:/spark-2.2.0/README.md MapPartitionsRDD[3] at textFile at <console>:24 []
    |  C:/spark-2.2.0/README.md HadoopRDD[2] at textFile at <console>:24 []
```

同じインデントが同一のステージです。インデントが変わるとステージが変わる事を
意味します。

基本の変換

map

map メソッドは、RDD の各要素に対して引数で与えられた関数を適用し、新しい RDD を返却します。新しい RDD の要素型は元の RDD のものと同一である必要はありません。結果値は配列やリストになります。単語の RDD から、文字長の RDD を生成してみましょう。

```
val address = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
 address.map(address => address.length).collect()
```

```
scala> val address = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
address: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[16] at parallelize at <console>:24


scala> address.map(address => address.length).collect()
res21: Array[Int] = Array(5, 7, 8, 5)
```

flatMap

結果値をリストではなく、フラットな状態で返します。

```
val lines = sc.parallelize(Array("Apple is red", "PineApple is yellow"))
   lines.flatMap(line => line.split(" ")).collect()
```

```
scala> val lines = sc.parallelize(Array("Apple is red", "PineApple is yellow"))
lines: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[21] at parallelize at <console>:24


scala> lines.flatMap(line => line.split(" ")).collect()
res22: Array[String] = Array(Apple, is, red, PineApple, is, yellow)
```

filter

filter メソッドは、RDD の各要素をフィルタリングし、条件に合致した要素を新しい RDD として返却します。条件は Boolean 型を返却する関数として filter メソッドの引数に渡します。
次の例では先頭の文字が T の値だけを抽出します。

```
val address = sc.parallelize(Array("Tokyo", "saitama", "kanagawa", "chiba"))
address.filter(address => address.startsWith("T")).collect()
```

```
scala> val address = sc.parallelize(Array("Tokyo", "saitama", "kanagawa", "chiba"))
address: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[23] at parallelize at <console>:24

scala> address.filter(address => address.startsWith("T")).collect()
res24: Array[String] = Array(Tokyo)
```

distinct

distinct メソッドは要素の重複を除外した RDD を返します。データが複数ノードに渡る場合はシャッフルが発生します。

```
val address = sc.parallelize(Array("Tokyo", "saitama", "saitama", "Tokyo"))
address.distinct().collect()
```

```
scala> val address = sc.parallelize(Array("Tokyo", "saitama", "saitama", "Tokyo"))
address: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[25] at parallelize at <console>:24

scala> address.distinct().collect()
res25: Array[String] = Array(saitama, Tokyo)
```

zip

zip メソッドは、RDD を引数にとり、それぞれの要素のペア RDD を作成します。

```
val address1 = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
val address2 = sc.parallelize(Array("東京", "埼玉", "神奈川", "千葉"))
address1.zip(address2).collect()
```

```
scala> val address1 = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
address1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[29] at parallelize at <console>:24

scala> val address2 = sc.parallelize(Array("東京", "埼玉", "神奈川", "千葉"))
address2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[30] at parallelize at <console>:24

scala> address1.zip(address2).collect()
res26: Array[(String, String)] = Array((Tokyo,東京), (saitama,埼玉), (kanagawa,神奈川), (chiba,千葉))
```

union

union は要素の型が同じ RDD を結合します。重複の削除や、パーティションの統合などは行われません。

```
val address1 = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
val address2 = sc.parallelize(Array("東京", "埼玉", "神奈川", "千葉"))
address1.union(address2).collect()
```

```
scala> address1.union(address2).collect()
res27: Array[String] = Array(Tokyo, saitama, kanagawa, chiba, 東京, 埼玉, 神奈川, 千葉)
```

collect

collect メソッドは RDD の全ての要素をドライバプログラムに Array 型で返します。全てのパーティションの RDD を返すため、実行には注意が必要です。

```
address.collect()
```

foreach

foreach メソッドは各要素に引数で与えた関数を適用します。戻り値はありません。

```
val address = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
address.foreach(println)
```

```
scala> address.foreach(println)
saitama
saitama
Tokyo
Tokyo
```

count

count メソッドは RDD の要素数を返します。

```
val address = sc.parallelize(Array("Tokyo","saitama","kanagawa","chiba"))
address.count()
```

```
scala> address.count()
res29: Long = 4
```

top , takeOrdered

top メソッドは最も値が大きい順に引数分だけ値を返します（つまり降順）。
takeOrdered は昇順で値を返します。

```
val nums = sc.parallelize(Array(3,2,4,3,2,1))
nums.top(3)
nums.takeOrdered(3)
```

```
scala> val nums = sc.parallelize(Array(3,2,4,3,2,1))
nums: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[36] at parallelize at <console>:24


scala> nums.top(3)
res40: Array[Int] = Array(4, 3, 3)


scala> nums.takeOrdered(3)
res43: Array[Int] = Array(1, 2, 2)
```

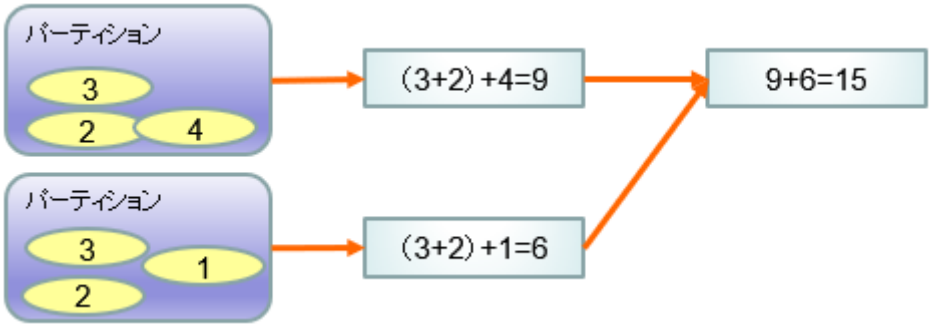
reduce

reduce は RDD の要素を集約します。集約の方法はメソッドの引数に関数を渡します。
数値の合計の例を示します。パーティションは 2 に設定しています。

```
val nums = sc.parallelize(Array(3,2,4,3,2,1),2)
nums.reduce((x,y)=>x+y)
```

```
scala> val nums = sc.parallelize(Array(3,2,4,3,2,1),3)
nums: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[39] at parallelize at <console>:24

scala> nums.reduce((x,y)=>x+y)
res45: Int = 15
```



ビッグデータ
ビッグデータ活用の
プランニング

演習 ビッグデータ活用プランニング

あなたは以下の企業の情報企画室長です。ビックデータを活用した何らかの改善活動、システム構築などを企画・立案ください。

社　　名	株式会社　スポーツワン
本　　社	東京都渋谷区
代　表　者	山田　大輔（52 歳）
設　　立	1973 年 6 月 8 日
資　本　金	100 億円
従 業 員 数	正社員：1000 名 パートタイマー、アルバイト：2032 名
店　舗　数	100 店舗　フラグシップ店舗　10 店舗
売　上　高	3790 億円／年（単体）
主な事業内容	スポーツ用品、ウェア製造・販売

当社を取り巻く状況

創業時期はゴルフ用品、スキー用品など高付加価値の商材を独自開発、北関東を拠点に全国に専門店を出店。現在は総合スポーツ用品店として 100 店舗を展開。業界 5 位の規模。楽天、アマゾンに対抗し EC サイトを立ち上げたが、リアル店舗を補完するほどの売り上げには達しておらず、リアル店舗との連動も進んでいない。スマホアプリも開発済みでリアル店舗から一定量の顧客の利用を実現したが、売上増進の牽引役にはなっていない。

EC、スマホアプリ共に購買履歴や行動ログを取得する準備はできているものの、具体的な施策がないため、活用されていない。

リアル店舗の売り上げは ID - POS によって管理されており、センターにデータが集まってきてはいるものの戦略的活用はされていない。

関東に比べると、関西九州地区への進出は進んでおらず、今後進出する地

域として有力である。創業時に開発した商材のシリーズは今もマニアからは愛好され、静かなブームを呼んでいるとの Twitter 上の噂がある。
スポーツ分野ごとの季節性は、感触として皆持っているが販売店への施策として打ち出されていない。
競合店の位置を加味した出店計画はしておらず、競合の新規キャンペーンによる売上変化も分析されていない。
各支店からのレポートは週次、月次の Excel レポート。

ポイント

- 社内に存在するビッグデータは、現時点で現存するか、新規に取得できるものとして、常識の範囲内で自由に設定してください。
- オープンデータは、現存すると予想されるものを自由に設定してください。
- 社外購入データは、予算の範囲内で調達可能と考えてください。
- 取りえるビッグデータ活用企画を立案し、中期計画の形で立案ください。プロジェクト期間は 2 年以内です。

「企画の背景・目的」、「現状」、「要因分析」、「プラン」、
「スケジュール」の形でまとめてください。

模造紙等にまとめる内容

グループ名	
リーダー	
メンバー	
プロジェクト名	
背景・目的	本企画が立案された課題背景と目指すゴール、目的
現状	背景に記載された課題を記載
要因分析	課題・原因・対策候補を記載
プラン	プロセス・体制・投資対効果・利用データ・利用ツール 開発内容、新規サービス内容など
スケジュール	立案から実施までの大まかなスケジュールを記載