



NOSQLの基礎知識

河野 達也 / Tatsuya Kawano

R&D ソフトウェアエンジニア

クラウディアン株式会社

2013年7月30日

日本データマネジメント・コンソーシアム (JDMC)

研究会テーマ2 第2回オープン研究会



Cloudian Inc. & KK – Except where otherwise noted, content on this document is licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License.

本日の内容

- NOSQLの特徴
- データモデルによる分類
 - MongoDB、Cassandra、VoltDBの実演
- アーキテクチャによる分類（簡単に）
- 主要な製品（7+1製品）の紹介
- 活用事例（Cloudian）の紹介

この勉強会を終えると

- NOSQL製品を分類するためのキーワードが理解できる
- 主要な製品の特徴が説明できる
- NOSQLの使いどころが説明できる

駆け足です

- 本当は2日くらいかけて学ぶ内容を
3時間でやります
- このスライドの置き場所
- <http://bit.ly/jdmc-nosql>
<https://github.com/tatsuya6502/nosql-ja/tree/jdmc-jul-2013/slides>
- 推薦図書（4冊）も紹介します

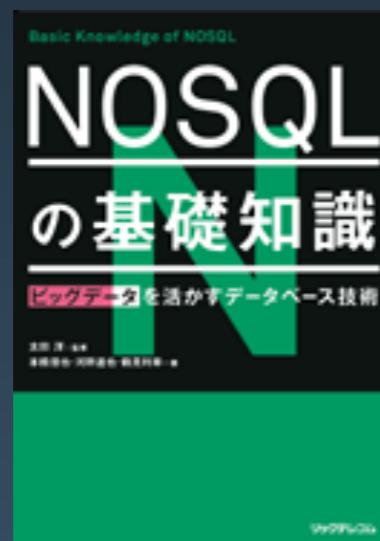
河野 達也 / Tatsuya Kawano

APACHE
HBASE

エバンジェリスト



開発者



著者の一人

クラウディアン株式会社



- **Cloudian**

- S3完全準拠オブジェクトストレージ

- **Hibari DB**

- GBクラスのウェブメールに採用

- NOSQL afternoon in Japanを主催 (2010年)

- グローバル開発リーダー Gary Ogasawara

- Inktomiで開発を経験 (CAP定理のE. Brewer氏らが立ち上げた会社)

本セミナー資料のライセンス（使用条件）

- スライド
クリエイティブコモンズ
表示-非営利 3.0 非移植 (CC BY-NC 3.0)
- 原著者のクレジットを表示し、
- かつ、営利目的で利用しなければ、
- 本作品を複製、頒布、展示、実演することができる。
- 二次的著作物を作成することができる。
- ソースコード
MITライセンス



NOSQLの 特徴

スケーラビリティー
半構造のデータ

NOT Only SQL

*It's about recognizing
that for some problems
other storage solutions
are better suited!*

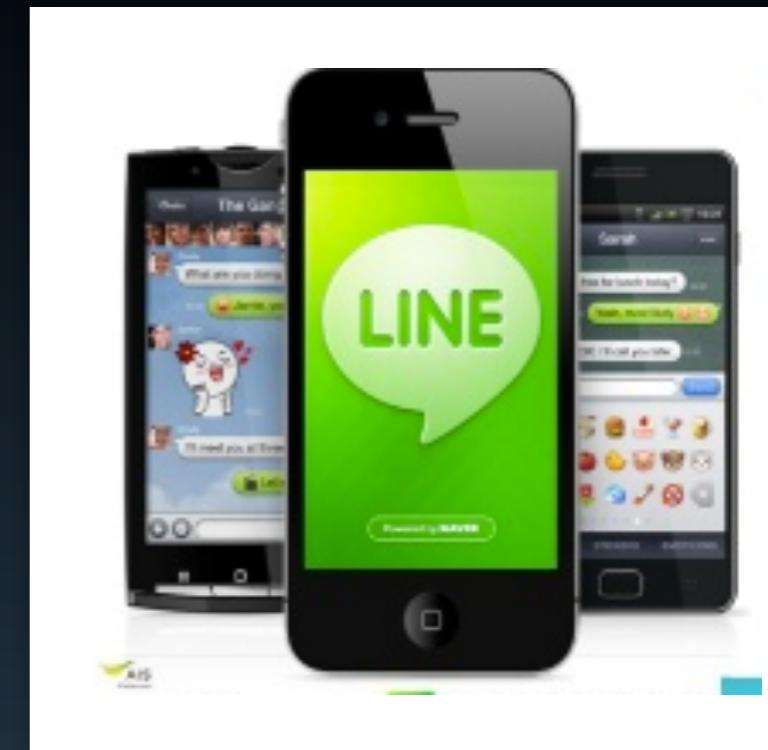
NOSQL: Not Only SQL

- ウィキペディア「NoSQL」のページより <http://ja.wikipedia.org/wiki/NoSQL>
- NoSQL（一般に“Not only SQL”と解釈される）は、リレーショナルデータベース管理システム（RDBMS）以外のデータベース管理システムを指す、おおまかな分類語
- RDBをやみくもに使用してきた長い歴史を打破し、それ以外のデータベースの利用・発展を促進させようとする運動の標語となっている
- このようなデータベースの傾向として以下が挙げられる
 - 固定されたスキーマに縛られない
 - 関係モデルの結合操作を利用しないこと（場合によっては単にそのような機能が欠落しているだけ）
 - 水平スケーラビリティが確保しやすい事が多いこと
 - トランザクションを利用できないものが多いこと

なぜ NOSQL なのか？

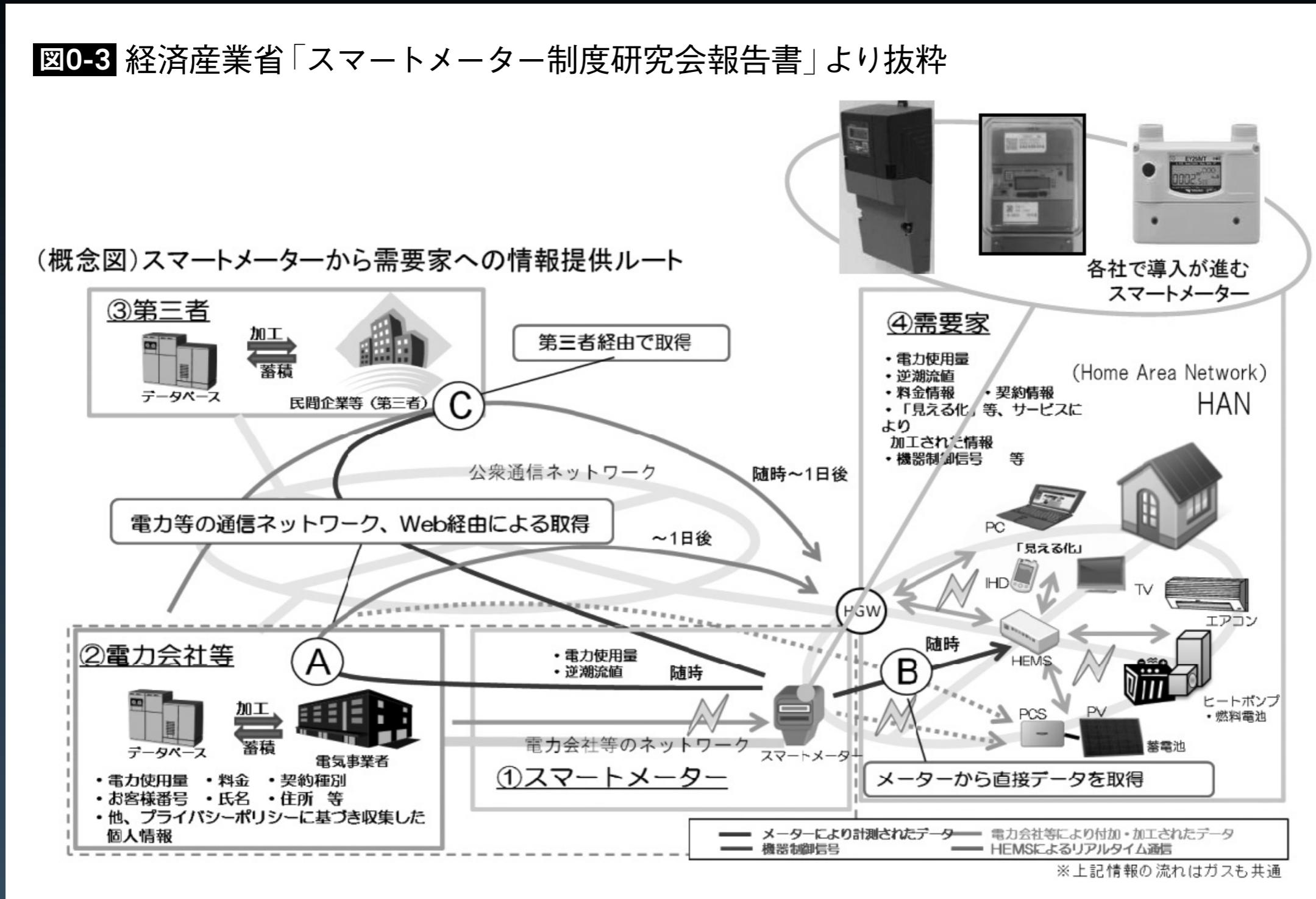


<http://www.slideshare.net/sunsuk7tp/hbase-at-line>



M2M センサーデータ

図0-3 経済産業省「スマートメーター制度研究会報告書」より抜粋



従来のリレーショナルDBでは対応が難しい

- 膨大な量
 - 12TB/日
- 速く処理する
 - $12\text{TB} \div 80\text{MB/秒} \div 42\text{時間}$
- 半構造データ
 - 全てのデータを均一に整えるのは難しい

ビッグデータに対応するための新技術

図0-2 Google BigtableとAmazon Dynamoの発表論文(表紙の一部)

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that spans many machines across a wide area. It stores petabytes of data across thousands of servers. Many programs run on Bigtable, including web search, machine learning, and data warehousing. These applications have different requirements than traditional relational databases. They need to support fast, low-latency access to large amounts of data, and they often need to process data in parallel across many servers. Bigtable achieves scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it supports a simple key-value interface that is well-suited for structured data. This paper describes the design and implementation of Bigtable, and shows how it has been used to build a variety of applications.

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components. One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture to manage application state. This paper describes the design and implementation of the system, and shows how it has been used to build a variety of applications.

大手Webサービスが利用を開始

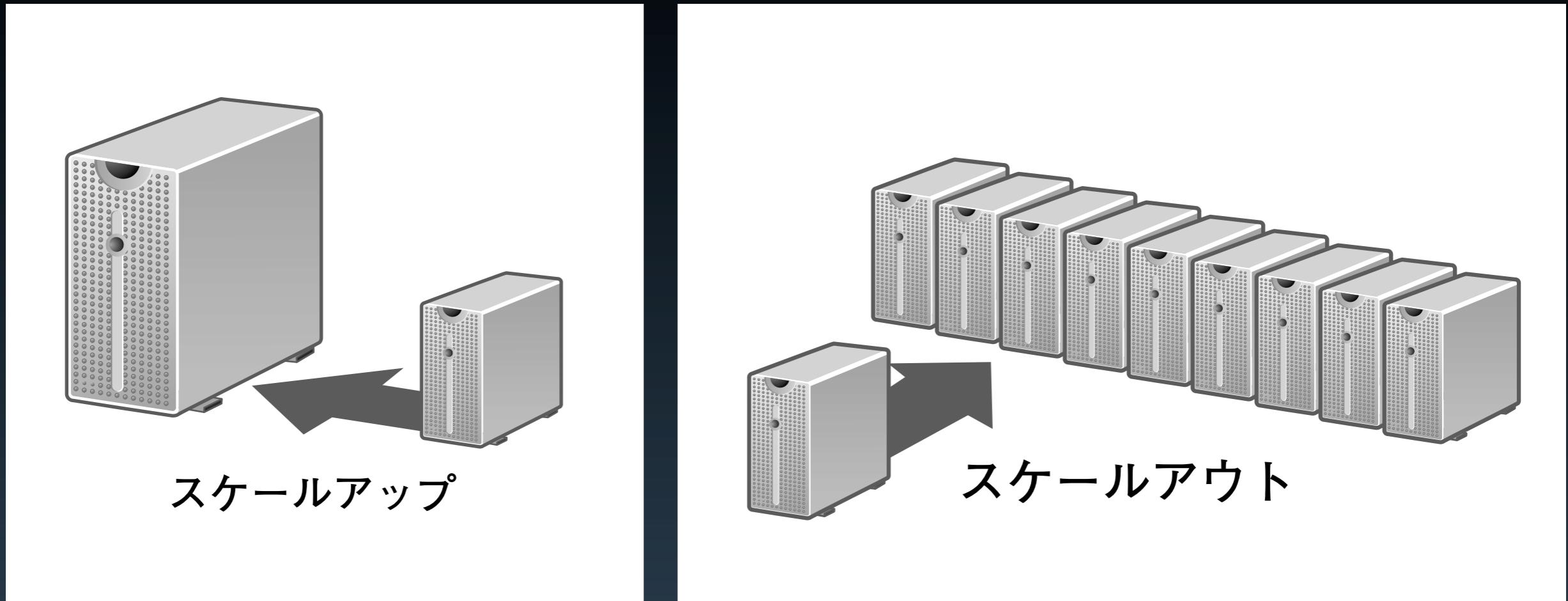
図1-2 代表的な NOSQL の利用企業の例



SQL以外のデータベースを総称

- NOSQLは世界に100種類以上
 - “One size does not fit all”
- 知名度だけで選ぶと大失敗する
- 用途に合った製品を選ぶためには、NOSQLに使われる技術を理解し、個々の製品の特徴を把握する必要がある

一般的なNOSQL製品の特徴



- スケールアウト性（水平スケーラビリティー）に優れるものが多い
- 汎用的で価格のこなれたハードウェアを多数並べることで、処理能力、データの格納容量、故障への耐性などを高める

一般的なNOSQL製品の特徴（続き）

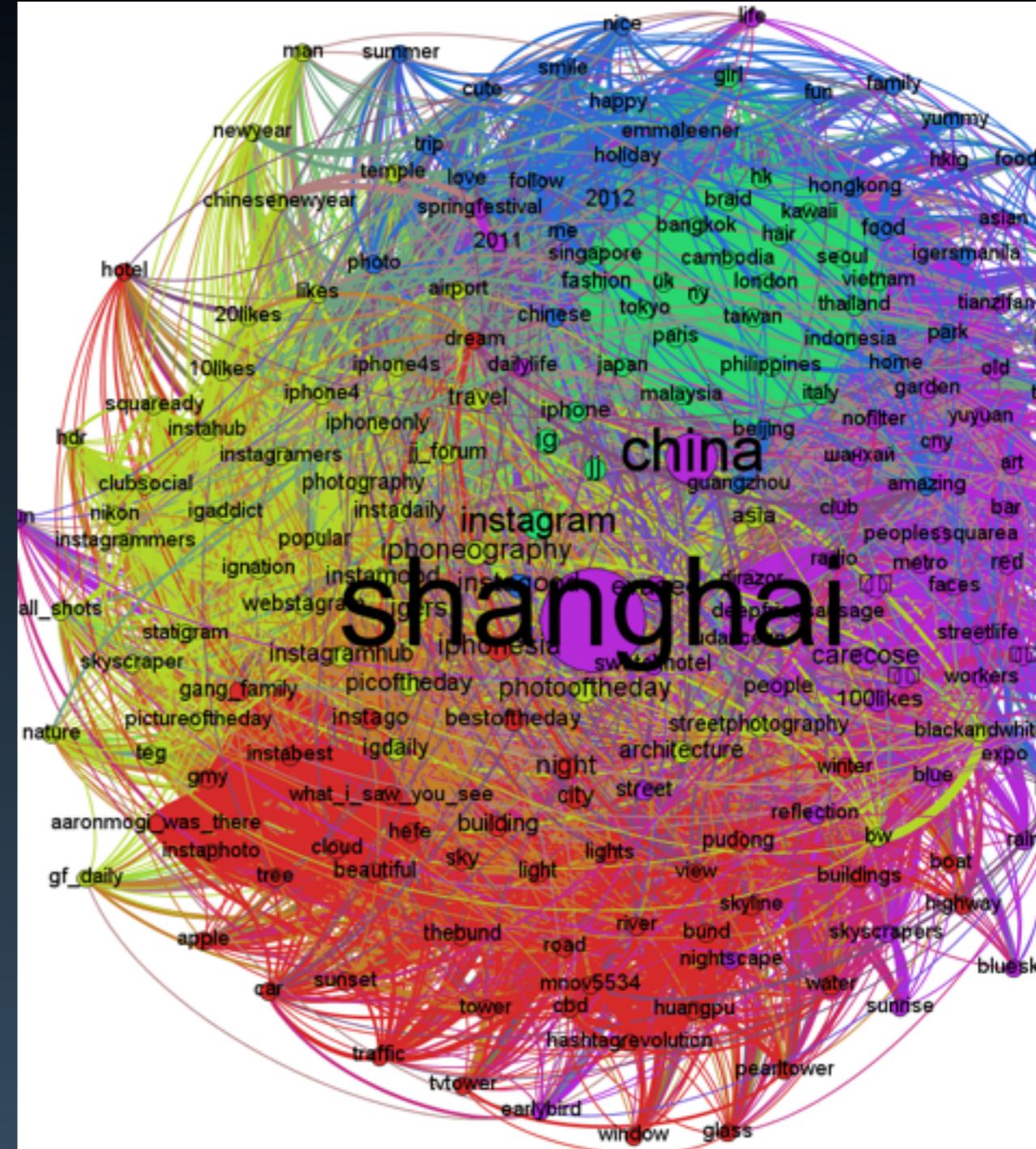
- 固定されたスキーマに縛られない
- 複雑で多様に変化するデータ構造に対応
- 高可用性と高信頼性
- 耐障害性
- サービス無停止でノード追加

NOSQLは用途を絞り込んだデータベース

- 機能が少ない
- リレーショナルモデルの結合操作ができない
- トランザクションが使えないものが多い
- 逆に、ある用途のために機能を特化・強化したものもある（例：グラフ型NOSQL）
- データの整合性が緩い（結果整合性など）
- 製品の成熟度？
- 主要な製品は成熟期にさしかかりつつある

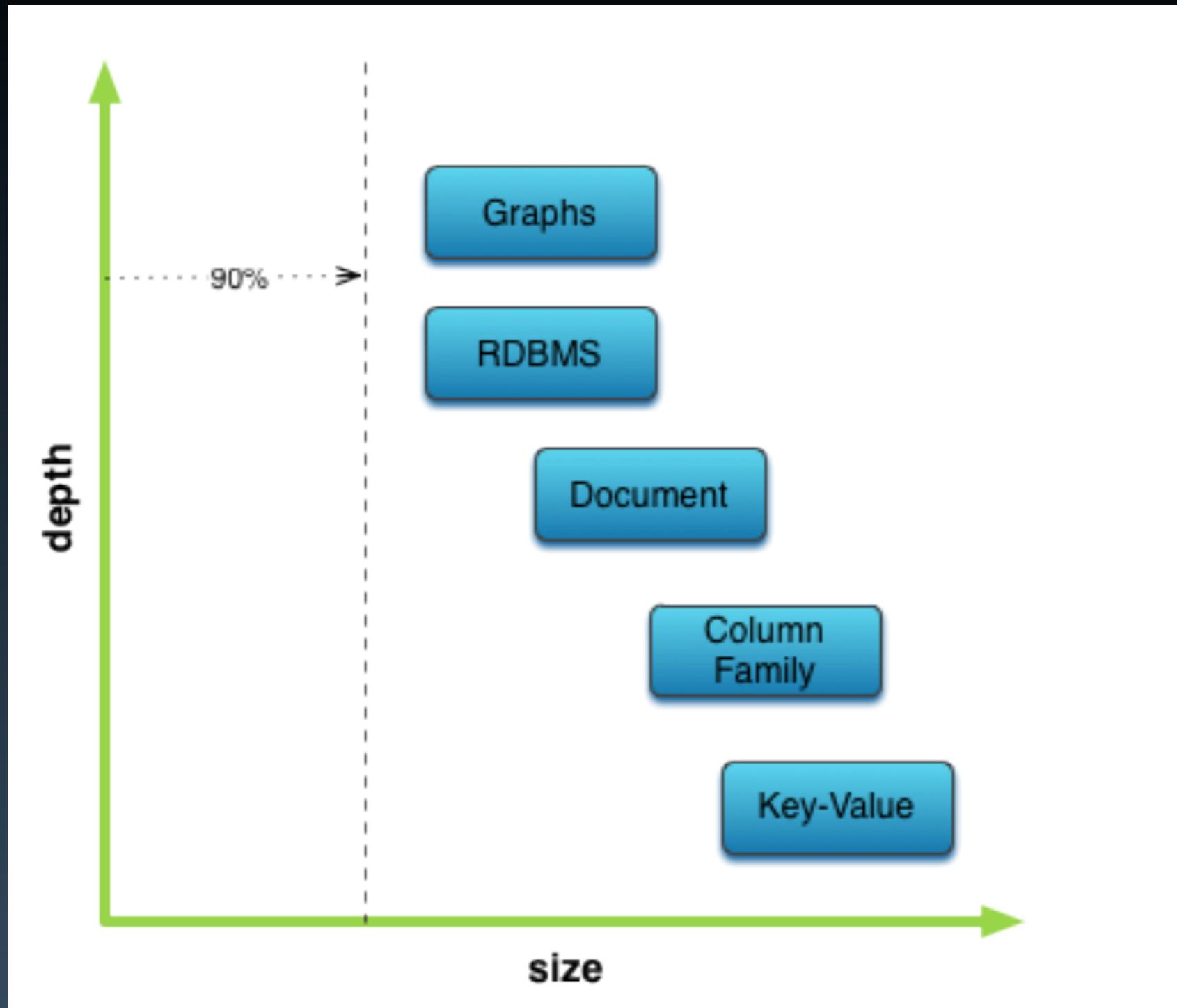
データモデルによる分類

キー・バリュー型
ドキュメント指向型
カラムファミリー型
グラフ型



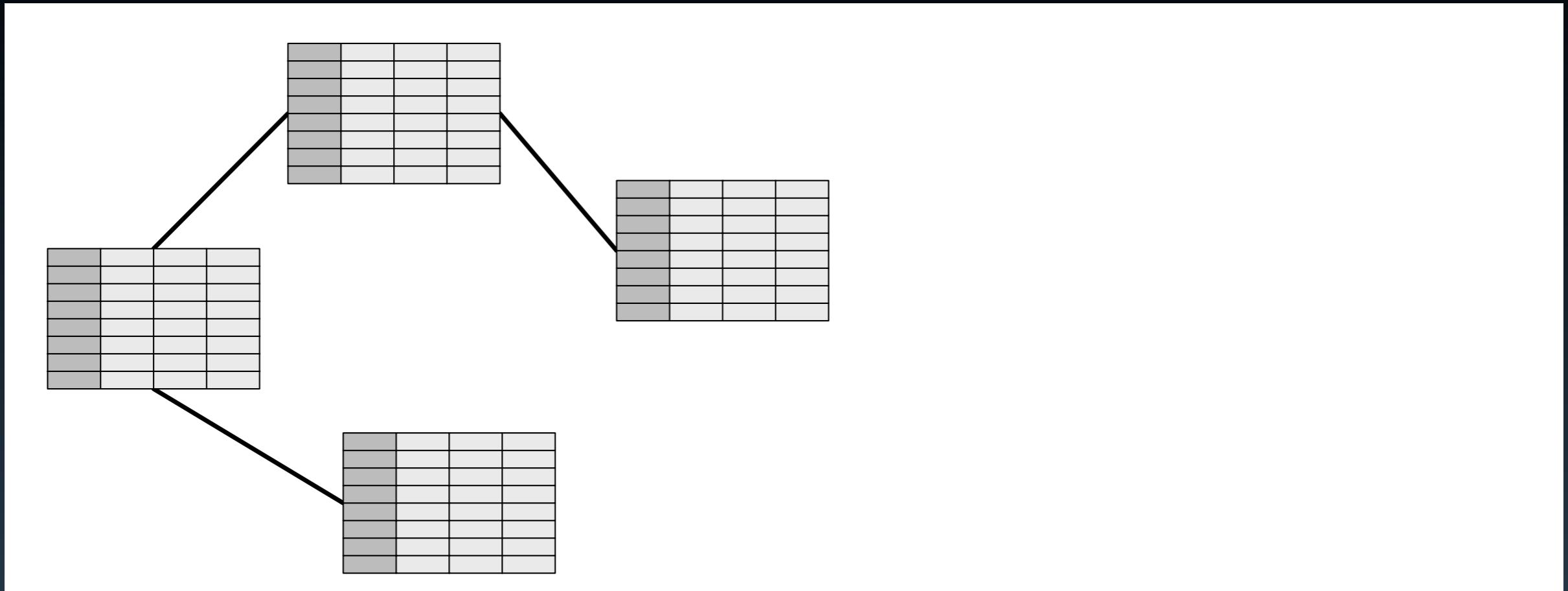
<http://beautifuldata.net/2012/01/telling-stories-with-network-data-instagram-in-china/>

NOSQLのデータモデル



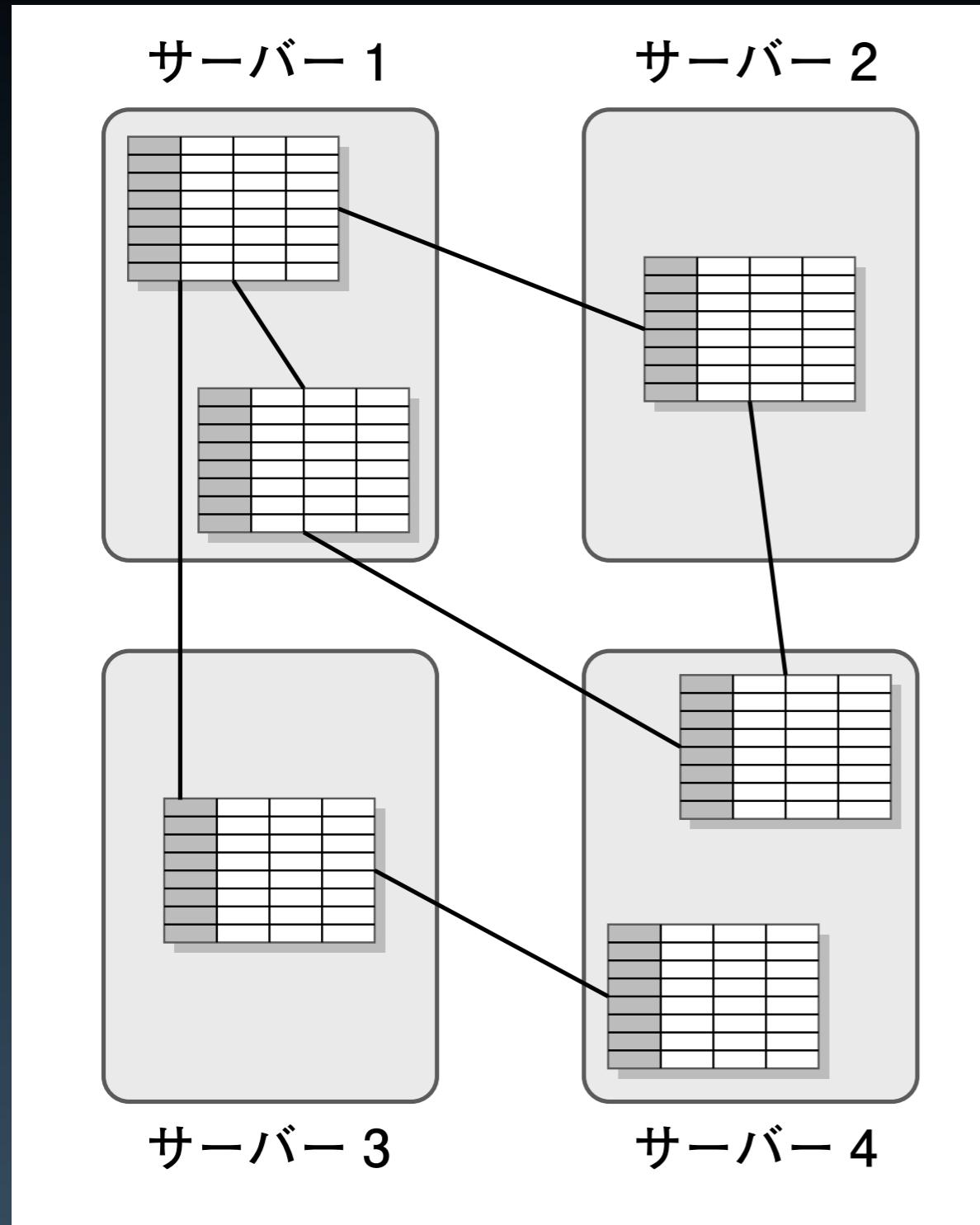
<http://www.neo4j.org/learn/nosql>

リレーショナルDB (RDB) のデータモデル



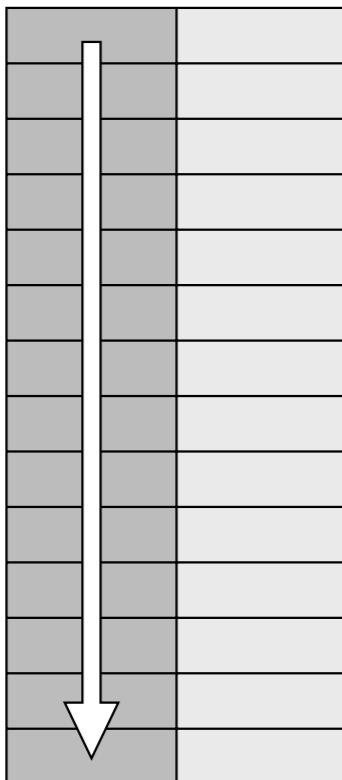
- 行と列で構成されるテーブルを定義
- テーブル間の関係性を定義
- 正規化により冗長生と不整合を排除

RDBにおけるデータ分散



分散環境では性能が
スケールしづらい

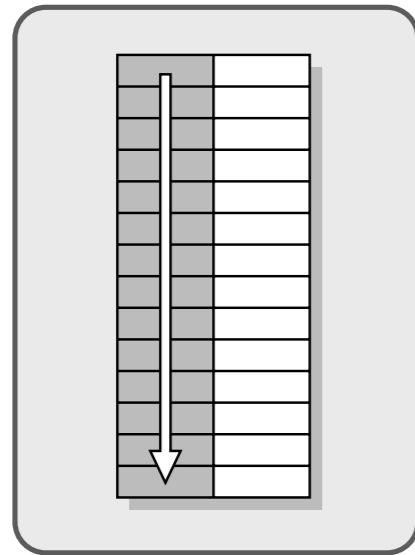
キー・バリュー型



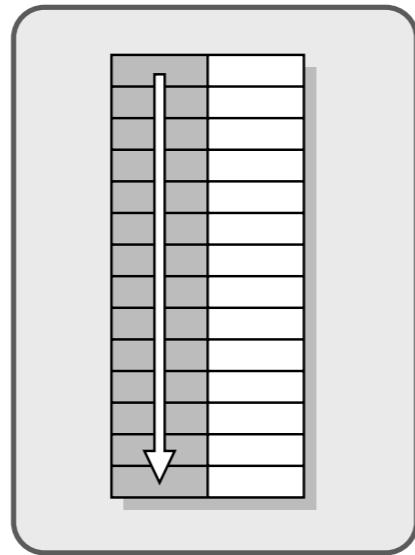
- 辞書のようなデータ形式

キー・バリュー型におけるデータ分散

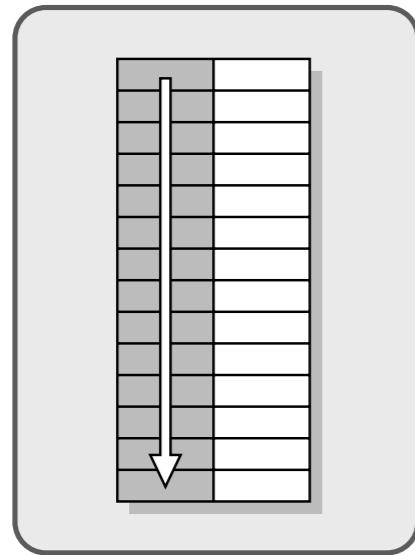
サーバー 1



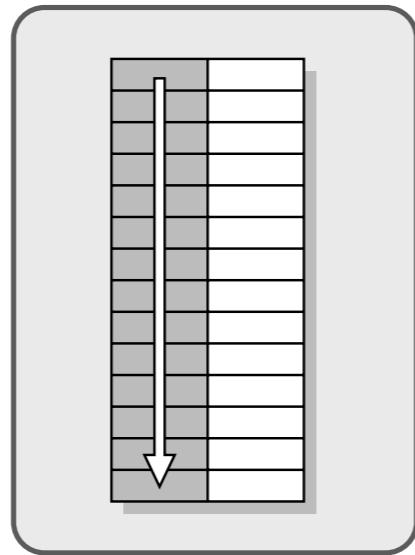
サーバー 2



サーバー 3

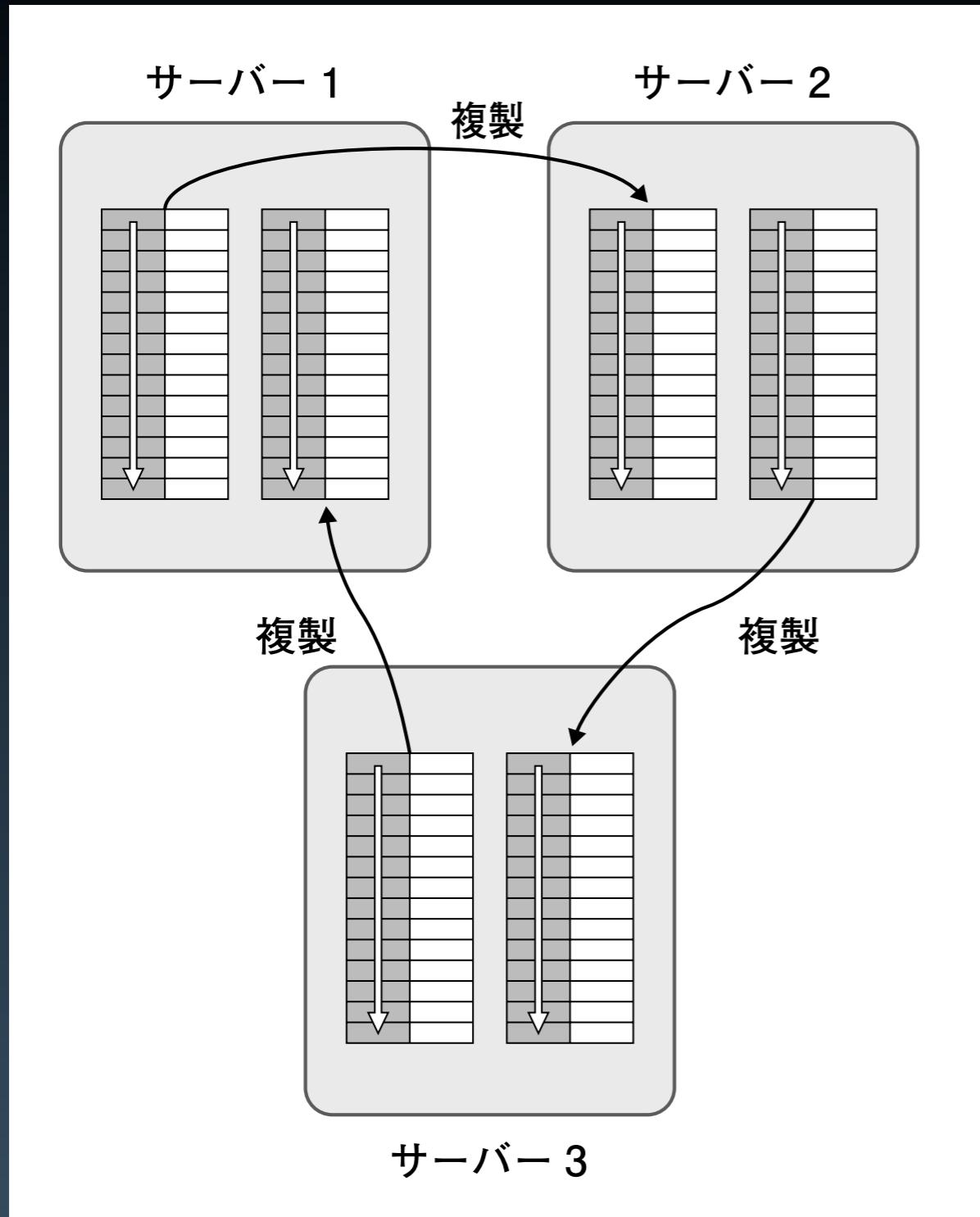


サーバー 4



テーブル間のリレーション
やトランザクションがない
ため、容易にスケールする

キー・バリュー型におけるデータの複製



ドキュメント指向型

- バリューに構造を持ったデータを格納
- 各項目にインデックスが付けられる

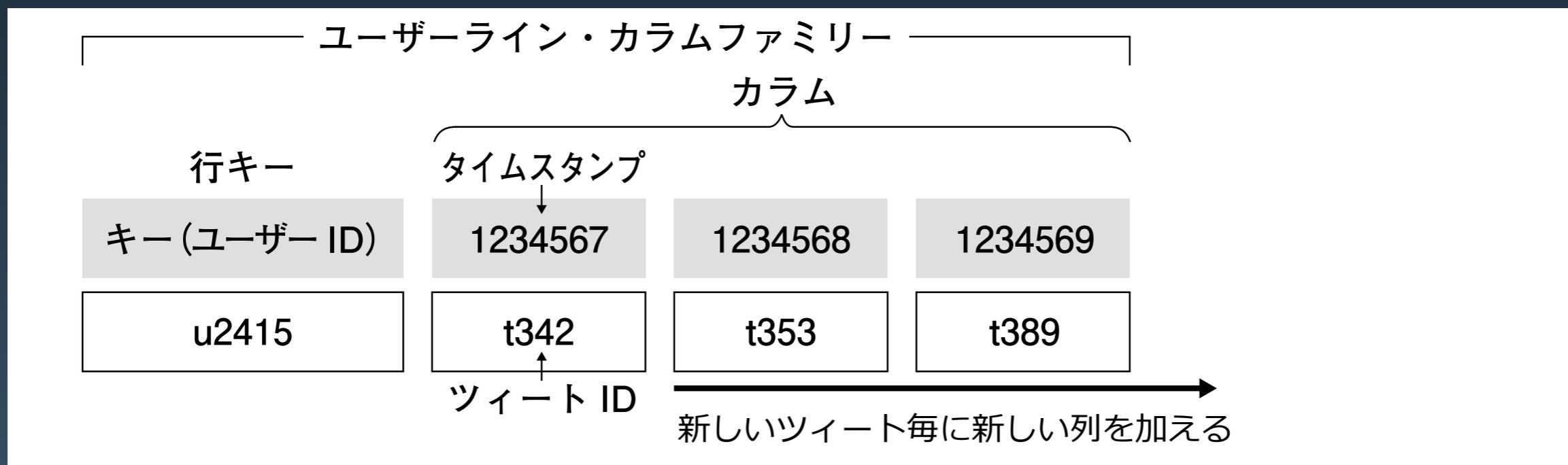
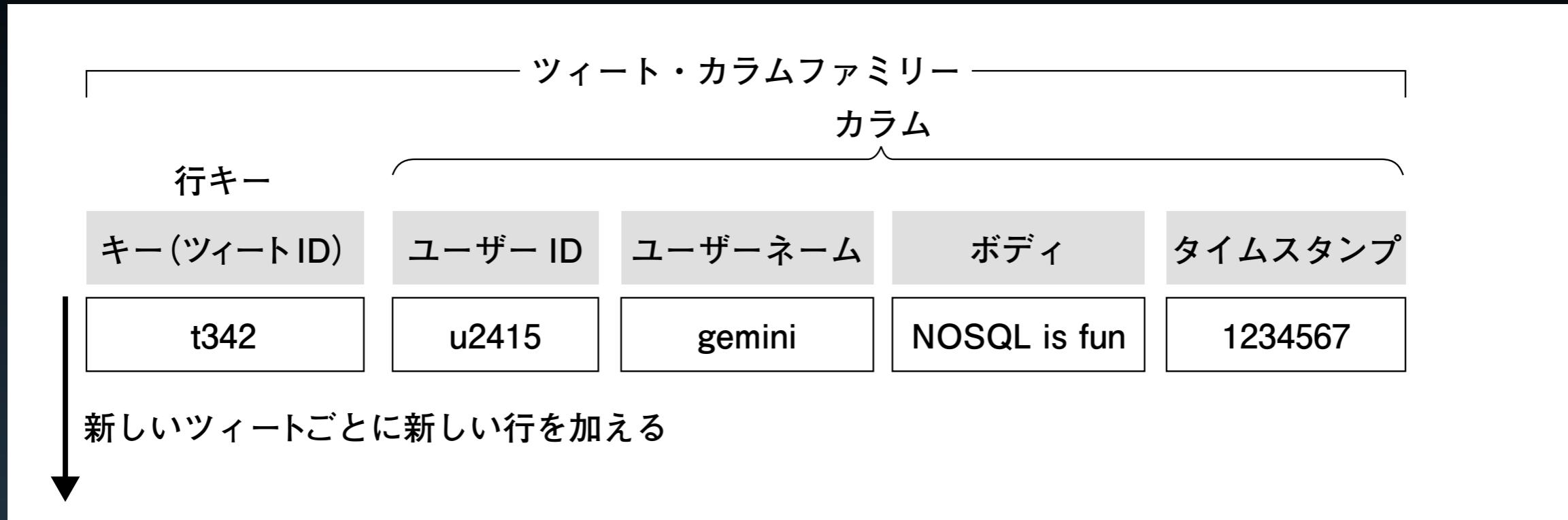
```
{ author:      'joe',
  created:     new Date('03/28/2009'),
  title:       'Yet another blog post',
  text:        'Here is the text...',
  tags:        [ 'example', 'joe' ],
  comments :  [ { author: 'jim',   comment: 'I disagree' },
    { author: 'nancy', comment: 'Good post' } ]
}
```

カラムファミリー型

http://www.datastax.com/docs/1.1/ddl/column_family

jbillis	dhutch	egilmore	datastax	mzcassie
dhutch				
egilmore				
datastax				
mzcassie				

カラムファミリー型（続き）



グラフ型

<http://www.neo4j.org/learn/try>

Graph Setup:

```
START root=node(0)
CREATE (Neo { name: 'Neo' }), (Morpheus { name: 'Morpheus' }), (Trinity { name: 'Trinity' }), (Cypher { name: 'Cypher' }), (Smith { name: 'Agent Smith' }), (Architect { name: 'The Architect' })
root-[:ROOT]->Neo, Neo-[:KNOWS]->Morpheus, Neo-[:LOVES]->Trinity,
Morpheus-[:KNOWS]->Trinity, Morpheus-[:KNOWS]->Cypher, Cypher-[:KNOWS]->Smith, Smith-[:CODED_BY]->Architect
```

Help Share Toggle Viz Options

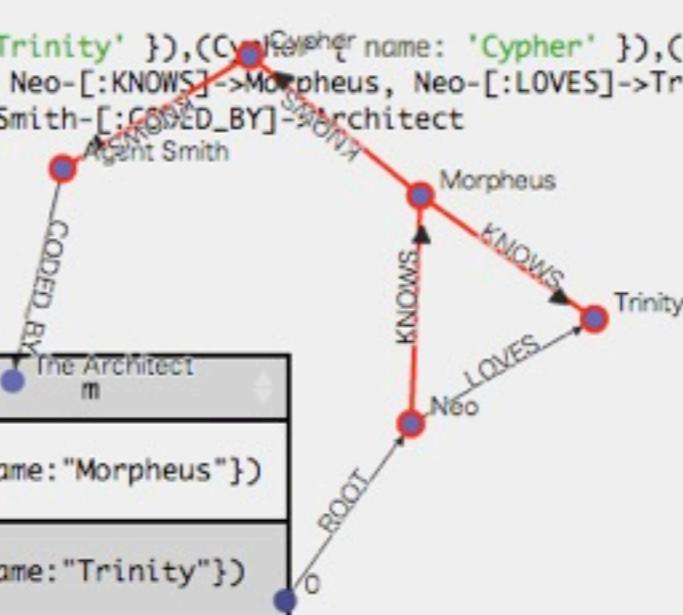
Query:

```
START n=node:node_auto_index(name='Neo')
MATCH n-[r:KNOWS*]-m
RETURN n AS Neo, r, m
```

Neo	r	m
(6 {name:"Neo"})	[(6)-[1:KNOWS]->(5)]	(5 {name:"Morpheus"})
(6 {name:"Neo"})	[(6)-[1:KNOWS]->(5), (5)-[3:KNOWS]->(4)]	(4 {name:"Trinity"})
(6 {name:"Neo"})	[(6)-[1:KNOWS]->(5), (5)-[4:KNOWS]->(3)]	(3 {name:"Cypher"})

```
START n=node:node_auto_index(name='Neo')
MATCH n-[r:KNOWS*]-m
RETURN n AS Neo, r, m
```

Run



- グラフ構造の扱いに特化
- データ同士の繋がりを可視化、独自のクエリ言語で分析
 - 最短経路の検索
 - ソーシャルグラフ、ジオメトリックグラフとリコメンド計算
- データ構造上、スケールアウトは困難

Demo

ドキュメント指向型NOSQL

このデモでは

- MongoDB を用いてドキュメント指向型 NOSQLのデータモデルとクエリを確認します。
- 比較のために、インメモリ型リレーショナルDBMSの VoltDB も紹介します。

デモで使用するコード：

<https://github.com/tatsuya6502/nosql-ja/tree/jdmc-jul-2013/mongodb-ruby>

デモの流れ

- リレーショナルDBMS (VoltDB) で、MovieLensのデータ構造を確認
- VoltDBでSQLを使ったクエリ
- MongoDBでデータ構造と、その柔軟性を確認
- MongoDBでSQLと同等のクエリを実行

デモのまとめ

- ドキュメント内のフィールドはドキュメント毎に自由に持てる。データ構造に対する柔軟性が高い
- MongoDBでは様々なクエリがサポートされている。集計はMap Reduceで行う

デモのまとめ（続き）

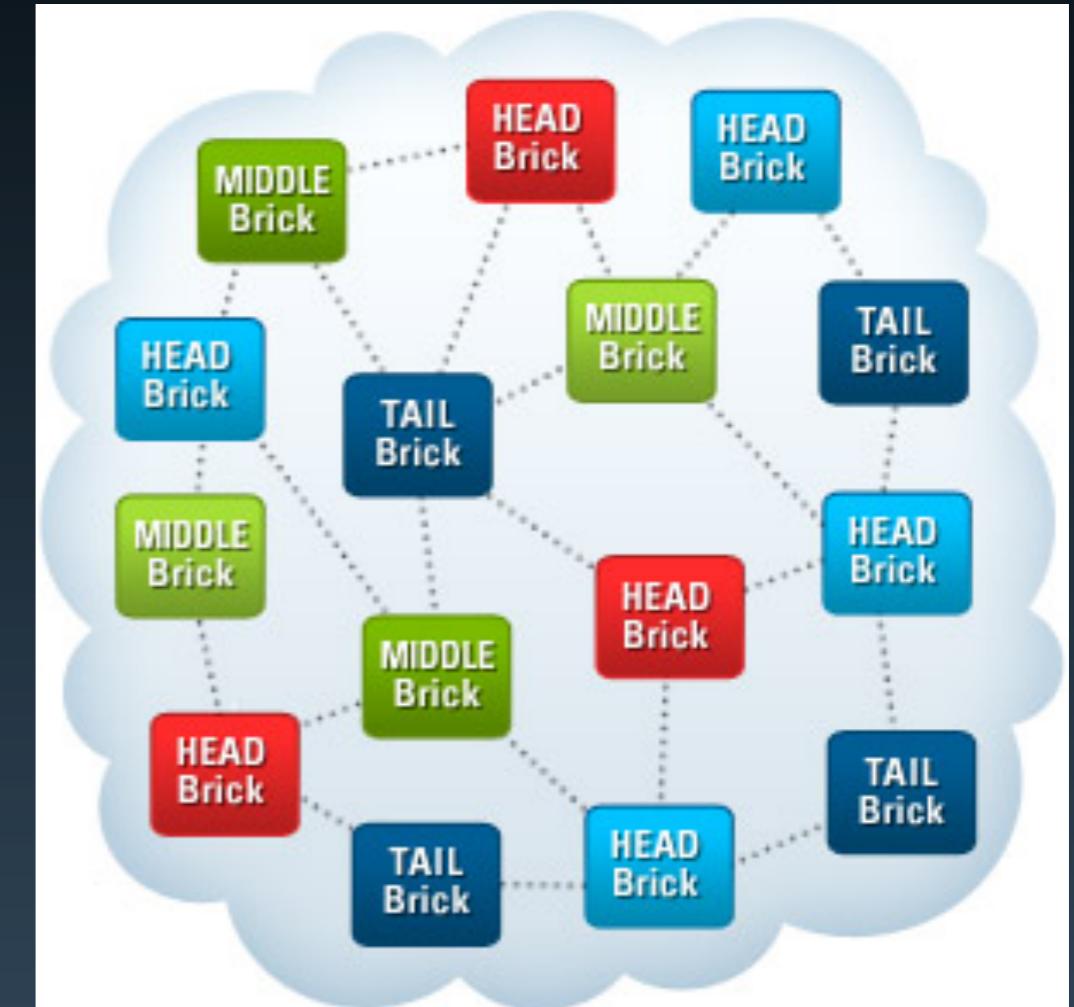
- MongoDBだけで十分？
- 大半の用途に十分な性能を持つ
- 以下のようなケースでは、カラムファミリー型やキー・バリュー型も検討する
 - さらに大規模なデータを扱いたい
 - さらに低いレイテンシー、または、高いスループットが求められる
- ACIDなトランザクションは実現できない
- リレーショナルモデルが最適な用途なら、VoltDBのようなNewSQLも選択肢に

デモのまとめ（続き）

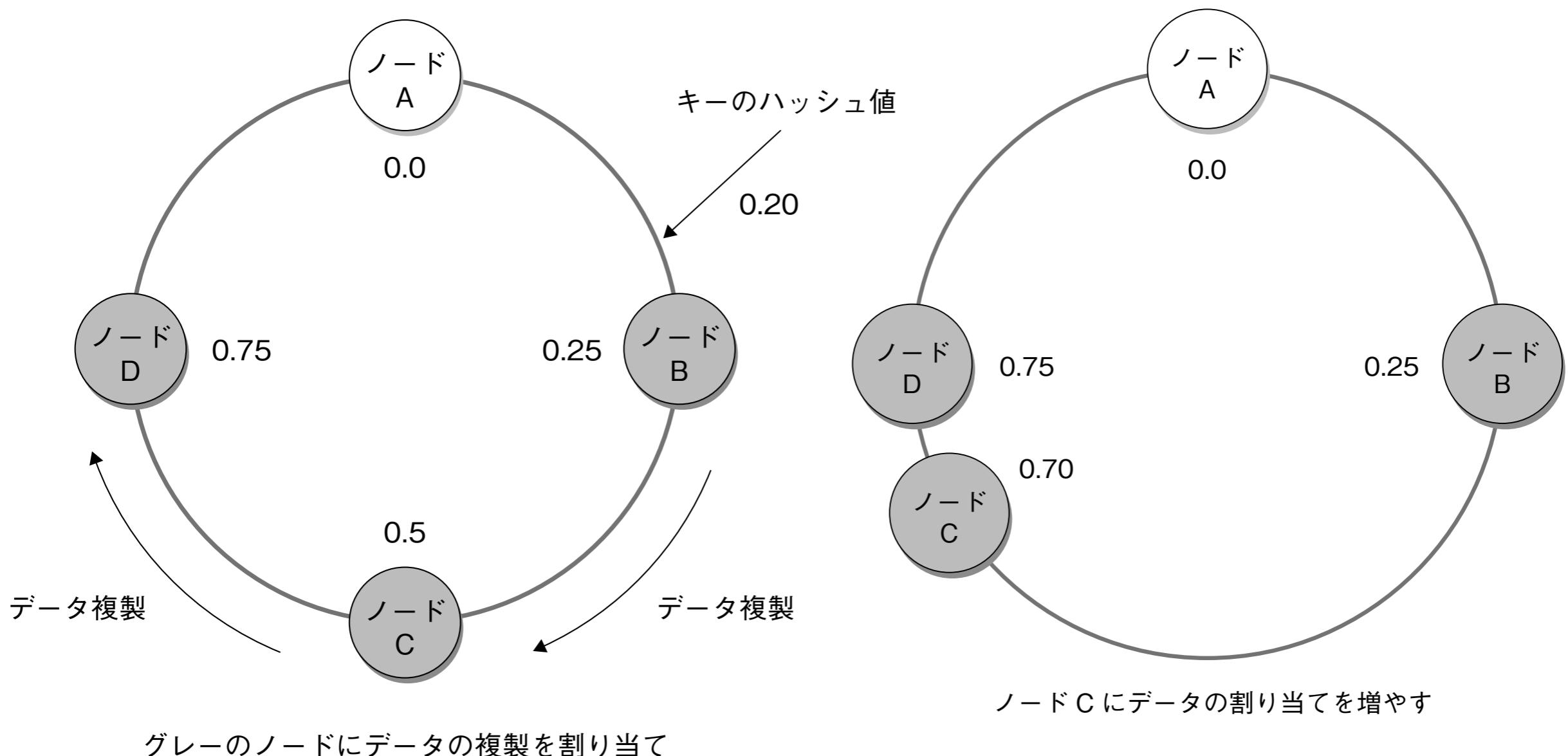
- VoltDB は NewSQL と呼ばれ、NOSQL に匹敵するスケーラビリティーを持ちながら、リレーショナルモデルと ACID なトランザクションを提供する
- 一方で柔軟性やエラスティック性（弾力性）を犠牲にしている
 - サービス中のスキーマ変更ができない
 - サービス中のノード追加に制限がある

アーキテクチャ による分類

データ分割
CAP定理

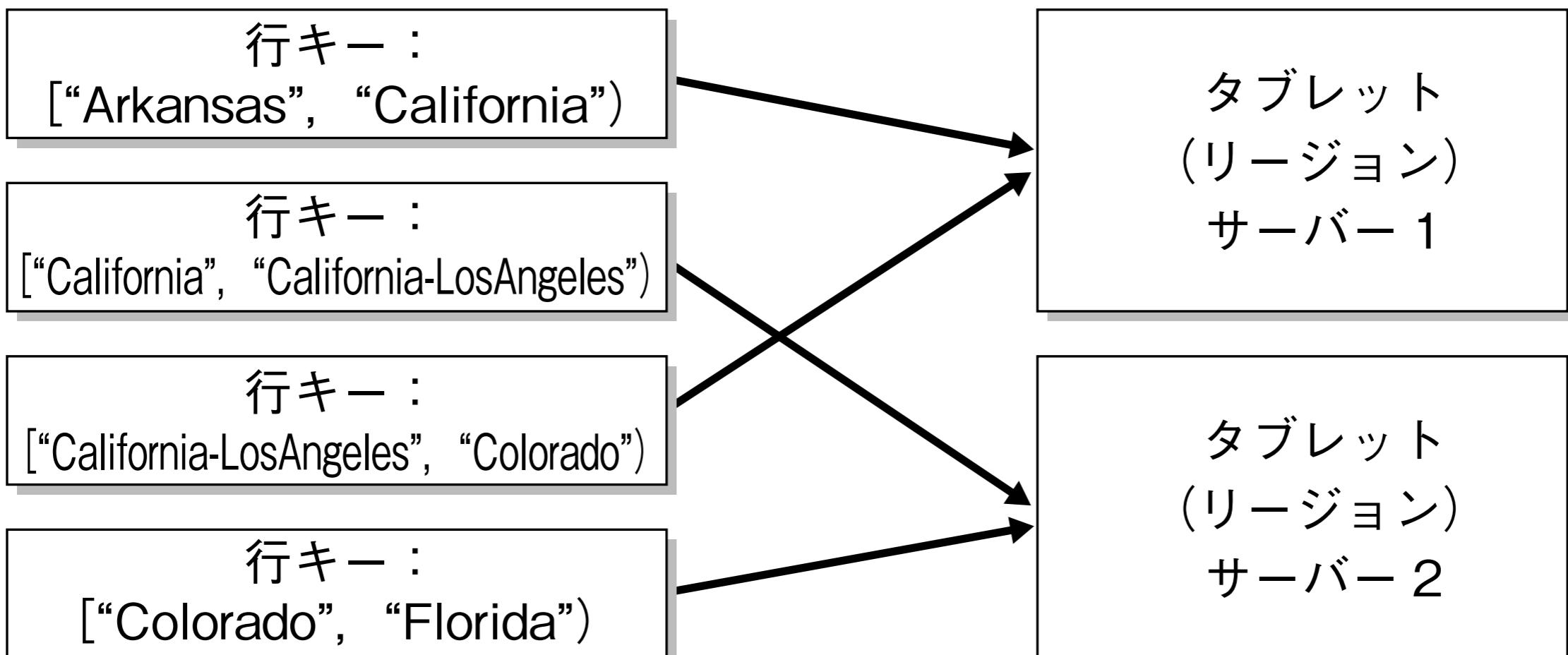


データ分割 コンシスティント・ハッシング

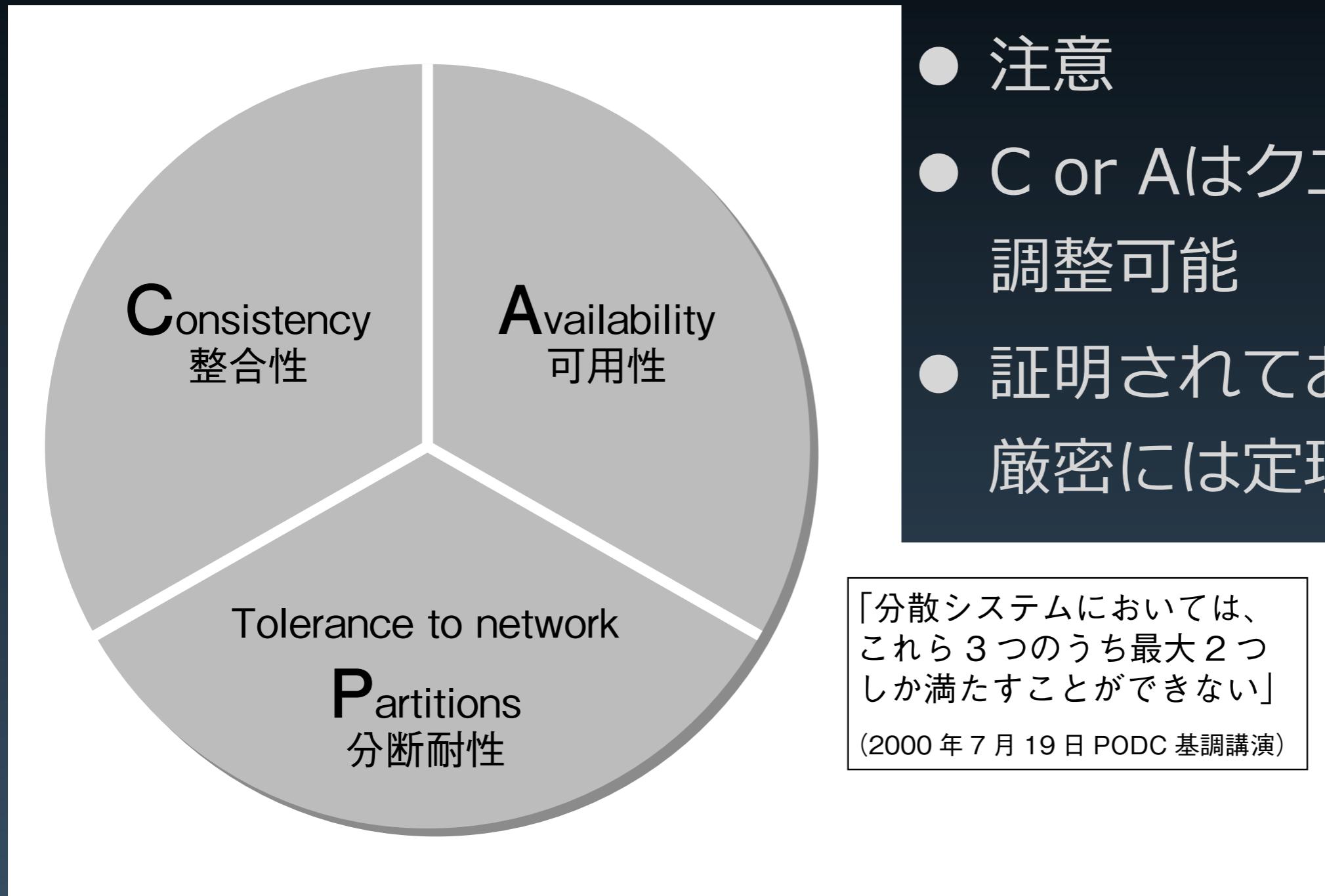


データ分割 シャーディング

タブレット(リージョン)



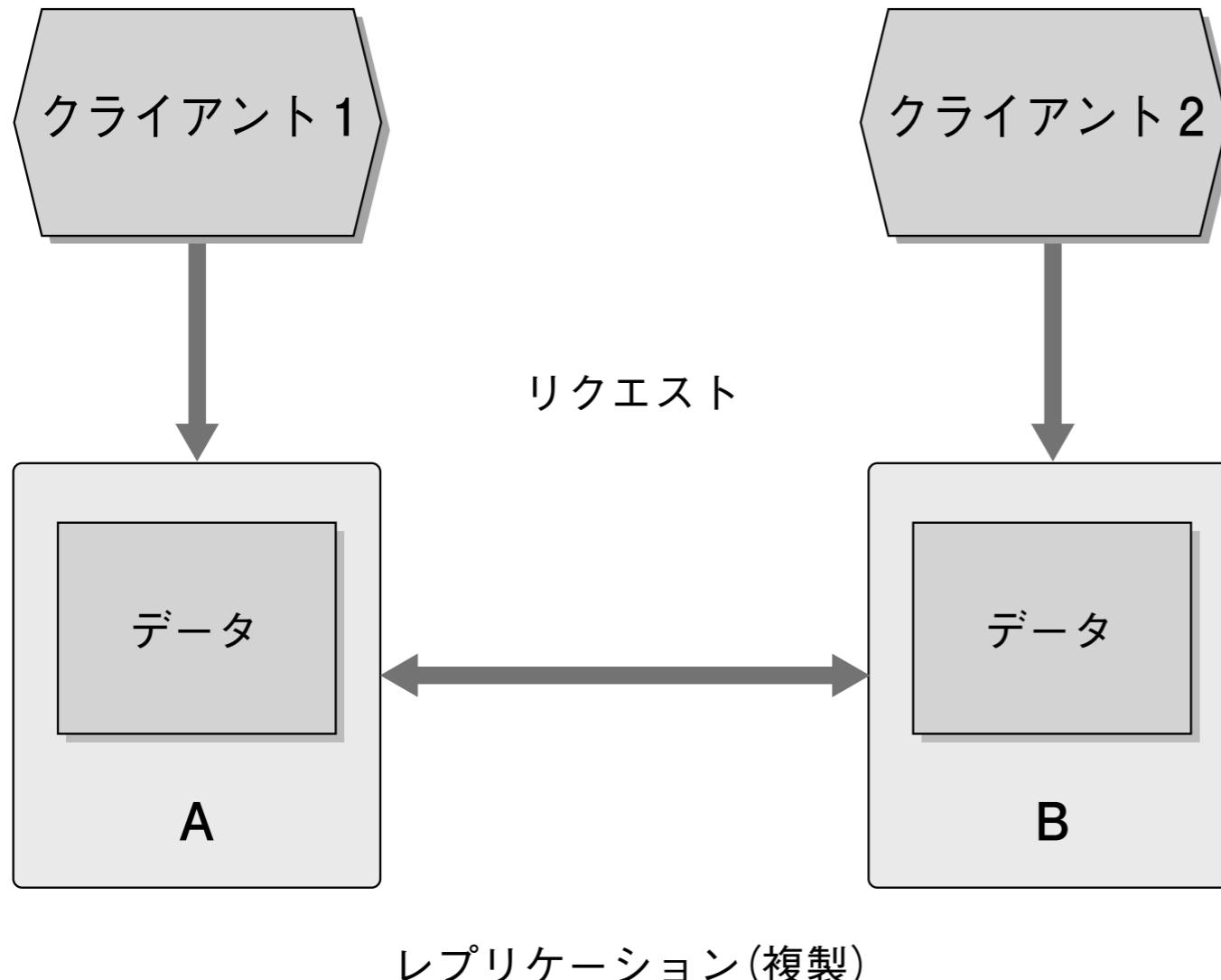
Eric BrewerのCAP定理



- 注意
- C or Aはクエリの都度
調整可能
- 証明されておらず
厳密には定理ではない

「分散システムにおいては、
これら 3 つのうち最大 2 つ
しか満たすことができない」
(2000 年 7 月 19 日 PODC 基調講演)

データ複製時の整合性

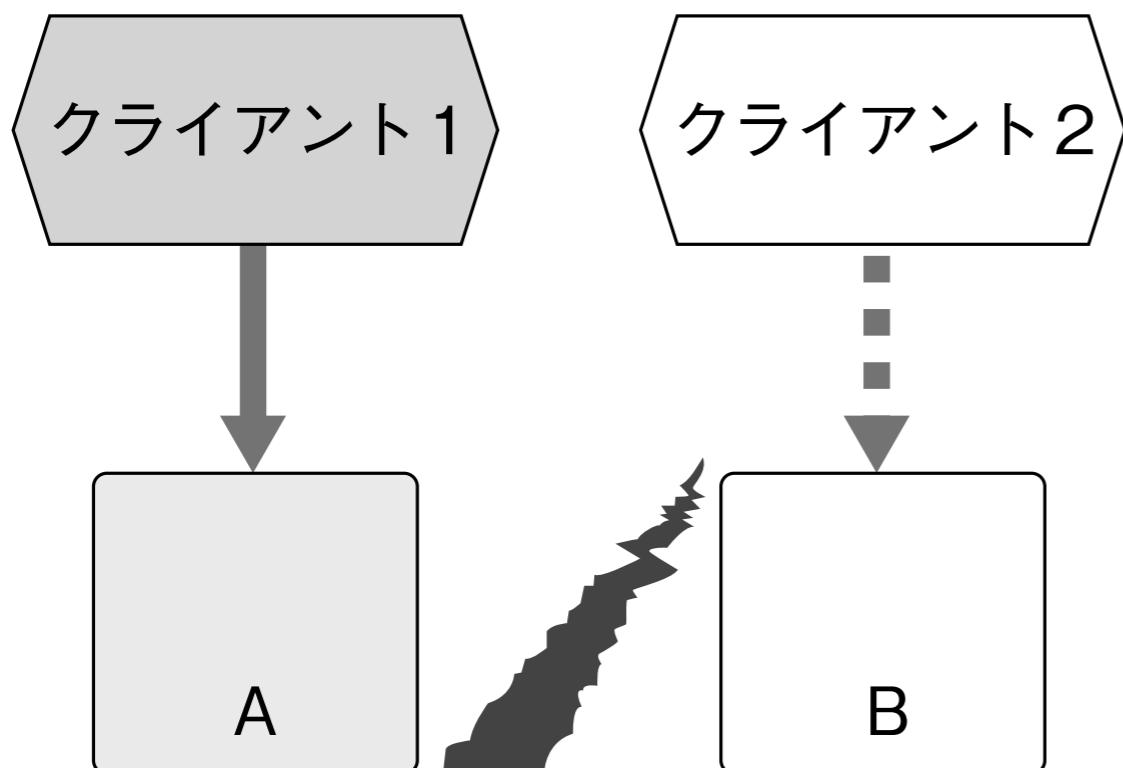


分割耐性

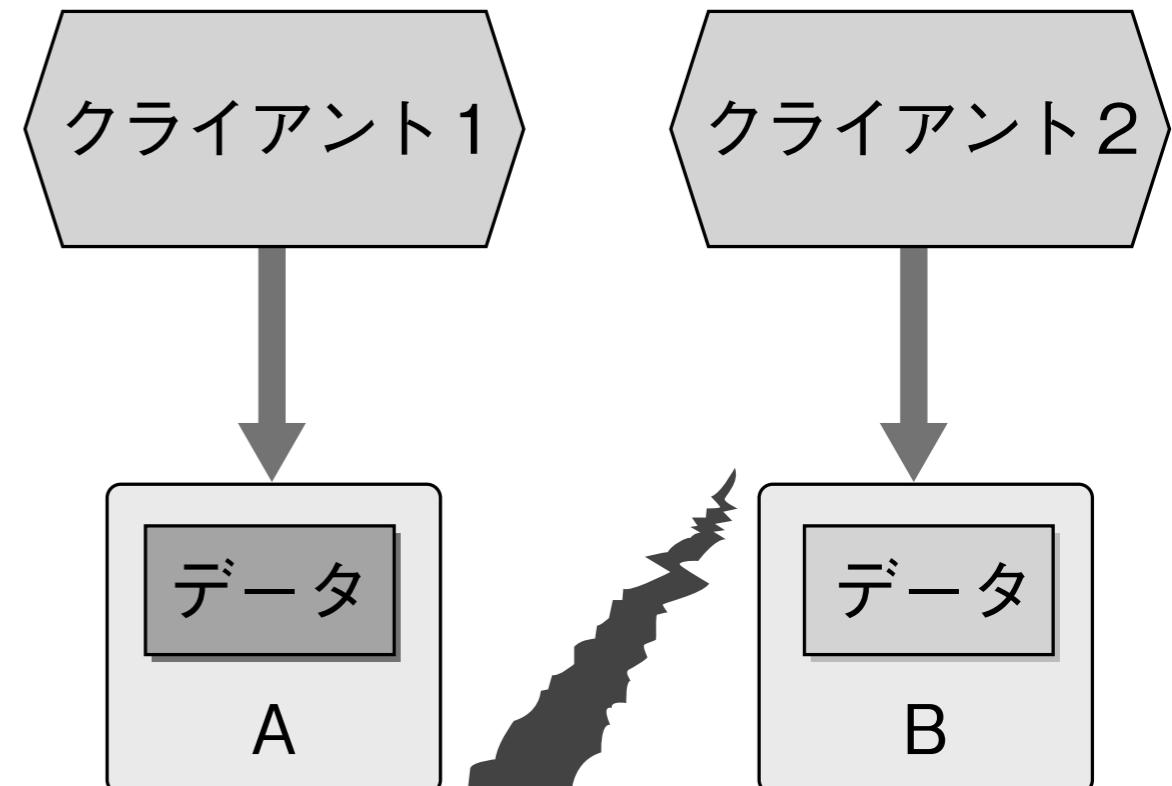
- ネットワークが分断されても
その特性 (CP、または、AP) を維持する

CPとAP ネットワーク分割が発生

CP(整合性と分断耐性)の場合：
リクエストは片方のグループ(仮に A)
だけが受け付け、他のグループは
自主的に停止する。

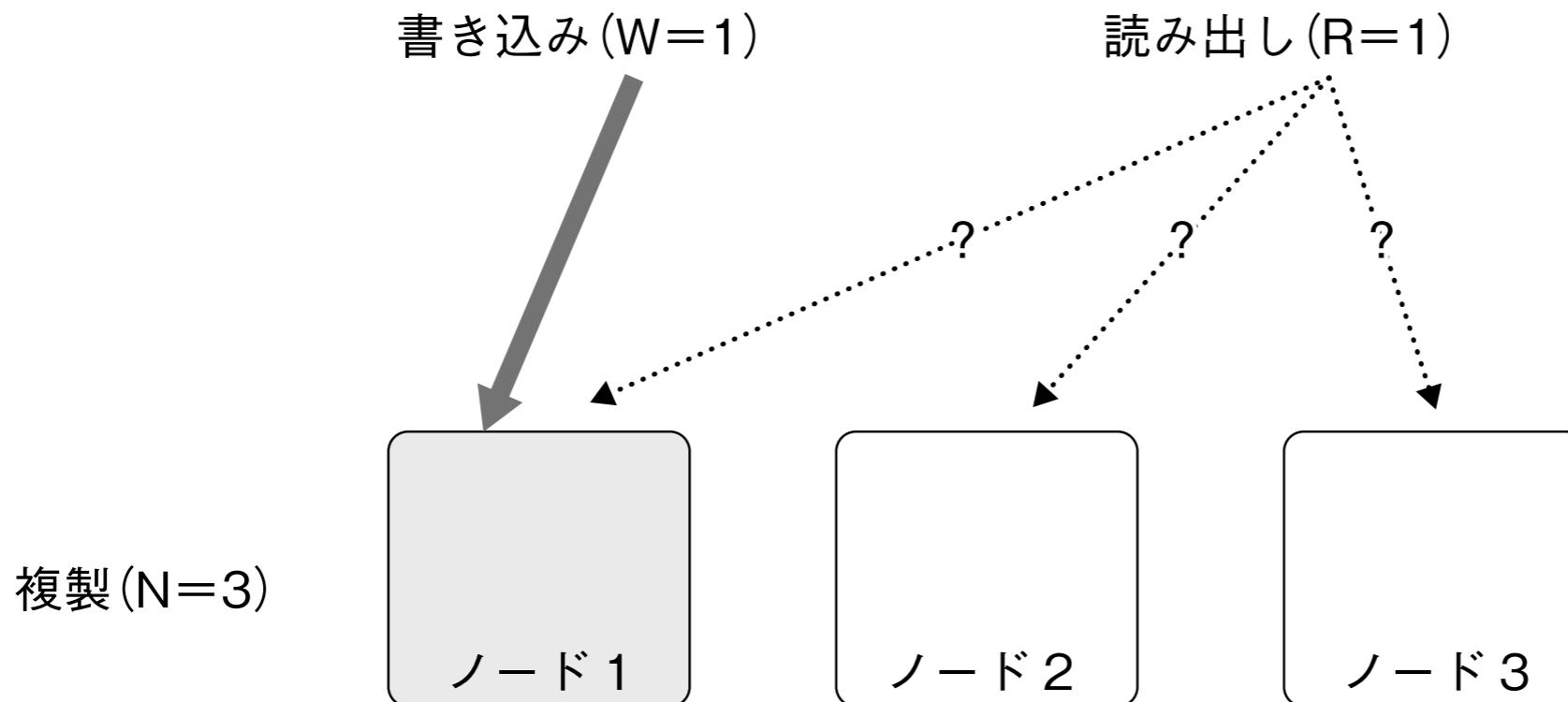


AP(可用性と分断耐性)の場合：
リクエストは全てのグループが受け
付けるが、A と B のデータは不整合
となる。



整合性と性能のトレードオフ

R+W< N の場合

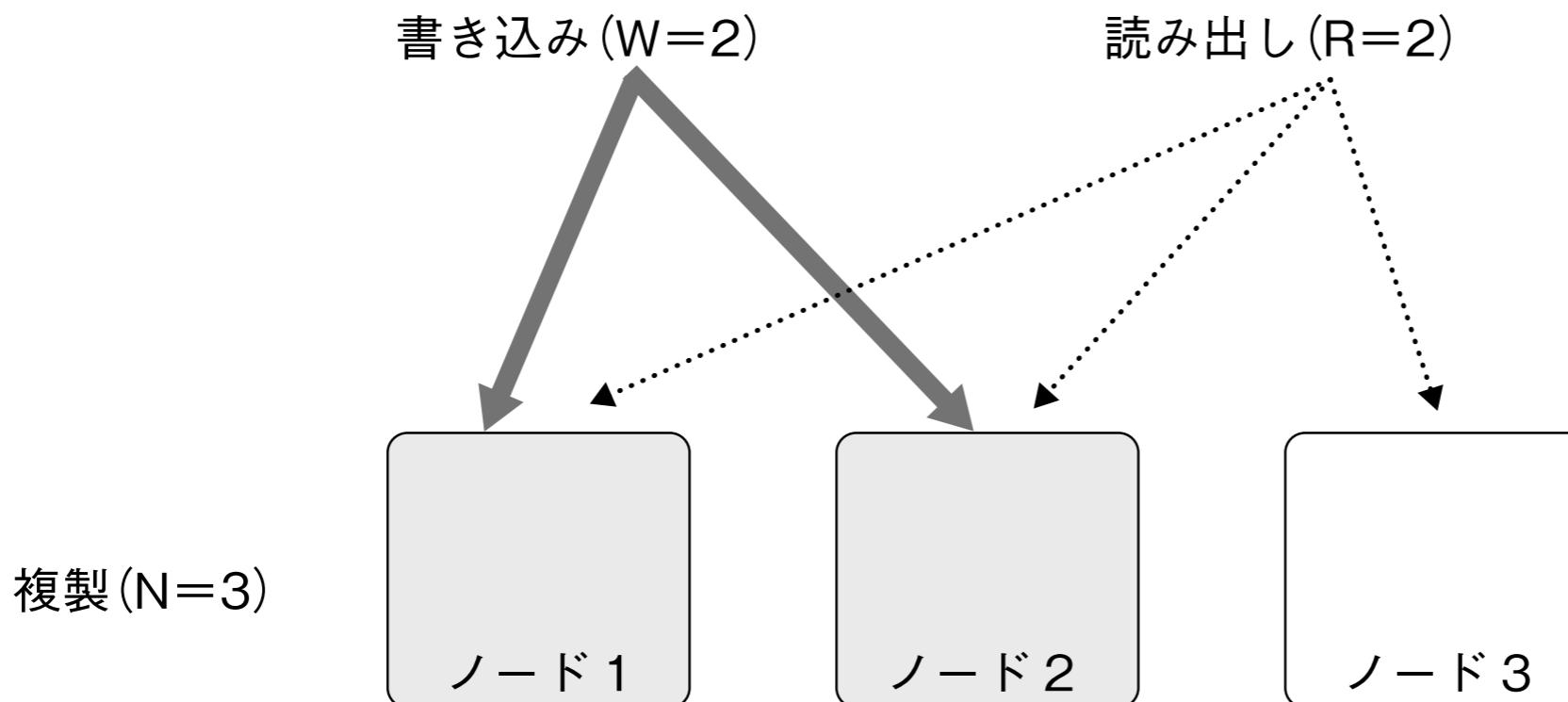


書き込みが 1 つのノードに行われているが、ノード 1.2.3 のどのノードから 1 つだけデータを読み出すかはわからない。

● 結果整合性 (Eventual Consistency)

整合性と性能のトレードオフ (続き)

R+W>N の場合



書き込みが 2 つのノードに行われ、読み出しが 2 つのノードに行われば、必ず書き込みが行われた 2 つのノードのうちの 1 つに行きつく。

● 強い整合性 (Strong Consistency)

広域複製の課題

- ネットワークの遅延が大きい環境
- Strong Consistencyでは性能が大きく劣化
- Eventual Consistencyではアプリケーション・ロジックが複雑になる
- イベントの因果関係に着目した Causal Consistencyという手法もある

Causal Consistency

- Aさんが、写真をアップロード
 - Aさんが、写真をアルバムに登録
 - B君が、Aさんのアルバムを開く
-
- Eventual Consistencyではビューに不整合が発生するかもしれない
 - 写真がまだ登録されていないのに、更新されたアルバムが見える
 - Causal Consistencyではこのようなことは避けられる

Causal Consistency

- Strong Consistencyより緩く、以下のようなことは実現できない
 - トランザクショナルな操作
 - ユニーク性などのintegrity制約
- Causal Consistencyは、今後、広域複製で注目されそうな手法

Demo

カラムファミリー型NOSQL

このデモでは

- Cassandra に MovieLens を格納します。
- CQL3のテーブル定義を確認
- CQL3でクエリ
- moviesカラムファミリーとratingsカラムファミリーの内部構成を確認

デモで使用するコード：

<https://github.com/tatsuya6502/nosql-ja/tree/jdmc-jul-2013/cassandra-cql3>

CQL3層での MovieLens データ構造

movies テーブル

movie_id	title	genres	...
1	Toy Story	{Anime, Comedy, For-Children}	...
788	Relative Fear	{Horror, Thriller}	...

ratings テーブル

movie_id	user_id	timestamp	rating
1	1	...	5
1	2	...	3
788	1	...	3
788	4	...	3

Cassandraストレージ層でのデータ構造

行キー	movies カラムファミリー				ratings カラムファミリー							
1	title	genres:Anime	...	1:	1:timestamp	1:rating	2:	2:timestamp	2:rating	...		
	Toy Story	空欄		空	...	5	空	...	3			
788	title	genres:Horror	...	1:	1:timestamp	1:rating	4:	4:timestamp	4:rating	...		
	Relative Fear	空欄		空	...	3	空	...	3			

- ratingsカラムファミリーは横（列方向）に成長する
- このような数千・数万の列を持つ行を「wide row」と呼ぶ
- moviesとratingsは、リレーションナルモデルで2つのテーブルを結合した時とそっくりの構造
- このように、テーブルを結合した状態でデータを格納することをリレーションナルモデルでは、非正規化と呼ぶ。管理はしづらくなるが、性能を大きく改善できるテクニックのひとつ

デモのまとめ

- カラムファミリー型 NOSQL では、カラムを自由に追加できる性質を利用して、リレーショナルDBにおけるテーブルの非正規化と同じことを実現できる
- 非正規化により結合操作を代用できるだけでなく、Readを高速化できる

デモのまとめ（続き）

- CassandraではSQL風のクエリ言語とスキーマを導入し、モデリングの難しさを排除しようとしている。wide rowもCQL3で複合キーを持つテーブルを定義することで、内部的に作成される
- CQL3はスケーラビリティーを保つために、リレーショナルモデルの結合操作はサポートしない（代わりに wide row、つまり、複合キーを持つテーブルやコレクション型を使う）
- 一方で、VoltDBでは柔軟性や弾力性を犠牲にした

主な製品

Cassandra

HBase

Riak

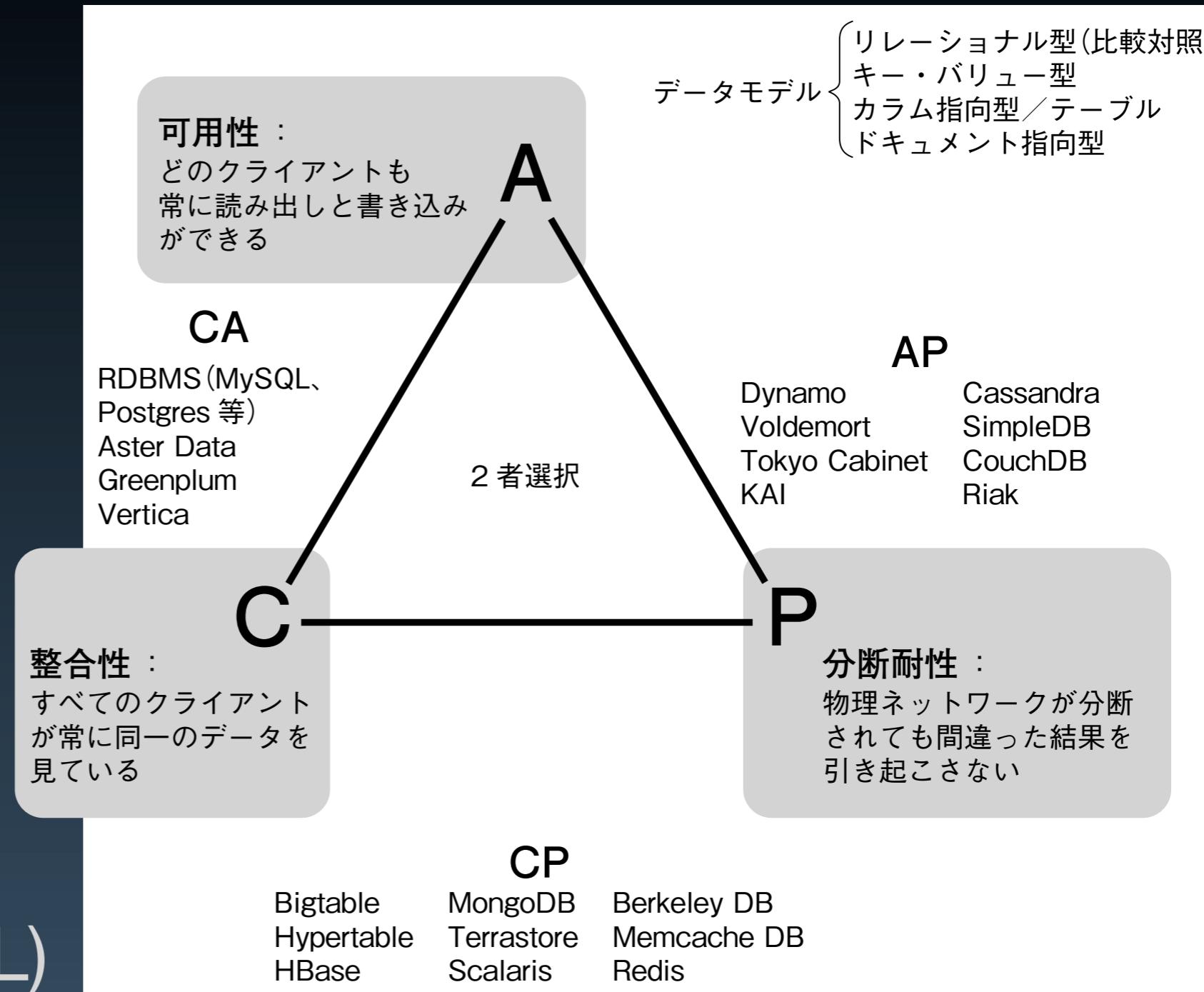
MongoDB

Redis

Infinispan

Neo4J

VoltDB (NewSQL)





Cassandra

Apache Cassandra

- Amazon DynamoとGoogle Bigtableを参考に設計された
- カラムファミリー型
- 緩い整合性（調整可能AP→CP）
- コンシスティント・ハッシングを採用し、スケールアウト性に優れる
- 大規模データに強く、100台程度のクラスターの運用実績が豊富。Writeが高速なことが特徴
- 最新のバージョンでは、CQL3というSQL風のクエリ言語を備える

- Google Bigtableを参考に設計
- カラムファミリー型
- 強い整合性 (CP)
- 自動シャーディングを採用し、スケールアウト性に優れる
- 大規模データに強く、100台程度のクラスターの運用実績が豊富。 Hadoop環境と相性がよい

- Amazon Dynamoを参考に設計
- キー・バリュー型
- JSONの扱いに優れ、使い勝手はドキュメント指向型に近い
- 緩い整合性（調整可能AP→CP）
- コンシスティント・ハッシングを採用し、スケールアウト性に優れる
- 大規模データに強く、50台程度のクラスターの運用実績が豊富。安定性が高く、扱いやすいことが特徴



{name: "mongo", type:"DB"}

- ドキュメント指向型
- 強い整合性 (CP)
- 分散環境では 緩い整合性 (AP) になる
- データモデルの扱いやすさと、強力なクエリが特徴。ウェブアプリ開発で特に人気が高い
- 分散構成では、自動シャーディングとレプリケーションクラスターを組み合わせることになり、リソースの使用効率が悪い



Redis

- キーバリュー型
- リストやハッシュなど、データ構造をネイティブでサポート
- 強い整合性 (CP)
- インメモリ、高速
- リッチなデータ構造と高速性から、ランキング情報、ユーザーのアクティビティ統計情報などのデータ格納先として人気がある
- 高可用性構成に対応しているが、現バージョンでは、スケールアウトはできない

Infinispan

- JBoss Cacheから派生した、データグリッド製品
- JavaのキャッシュAPIに準拠
- キーバリュー型、インメモリ、高速
- Java以外のクライアントもサポート
- 強い整合性（CP）だが、非同期操作などのスループットを高める仕組みが豊富
- ACIDなトランザクションをサポート
- コンシスティント・ハッシングを採用し、スケールアウト性に優れる
- キャッシュローダーにより、既存のデータストアとバックグラウンドでデータを同期できる
- RDBMS、Amazon S3、Cassandraなどに対応

Infinispan

続き

- 主な用途
- キャッシュとして
- Hibernateの2次キャッシュ
- JBoss AS (Application Server) のセッションキャッシュ
- Luceneインデックスの格納
- データグリッドとして
- 株式のトレーディングシステム
- ニア・リアルタイムのデータ分析、リスク計算

- グラフ型
- グラフエンジンのデファクトスタンダード
- 深いグラフのトラバースでは、RDBMSの1000倍以上の性能を発揮
- ACIDなトランザクションをサポート
- 専用のクエリ言語を持ち、グラフの可視化ができる（対話型の分析ツールにもなる）

- オンライントランザクション処理に特化したインメモリ型のリレーショナルデータベース
- Postgres や Ingres などのデータベース開発を手がけた Michael Stonebraker氏が設計
- NewSQLと呼ばれる
- SQLとACIDなトランザクションが使える
- 高速でスケールアウト性に優れる

- 柔軟性や弾力性（エラスティック性）が犠牲に
- 事前にストアドプロシージャを書いてコンパイルする。
アドホックなクエリでは性能を発揮できない
- 結合できるテーブル数に限りがある
 - 分析用途には不向き。RDBMSやHadoopなどへの差分データ連携機能を備える
- サービス稼働中にスキーマの変更はできない
- サービス稼働中のノード追加に制限がある
 - 現状、エンタープライズ版のみノード追加が可能。さらに、複製機能をオフにしたテーブルのみが対象となる

オープンソースソフトウェアの主なNOSQL製品

2013年7月現在

	Apache Cassandra	Basho Riak	Apache HBase	MongoDB	Redis	Infinispan	Neo4J	VoltDB
バージョン	1.2	1.3	0.94	2.4	2.6	5.3	1.9	3.4
開発者 括弧内はスポンサー	ASF	Basho Technologies	ASF	10gen	S. Sanfilippo + (Pivotal)	JBoss + (Red Hat)	Neo Technology	VoltDB
データモデル	CF	KV	CF	ドキュメント	KV (データ構造)	KV	グラフ	リレーションナル
メインストレージ	HDD/SSD	HDD/SSD	HDD/SSD	HDD/SSD	インメモリ	インメモリ	HDD/SSD	インメモリ
整合性モデル	AP	AP	CP	非分散HA : CP 分散HA : AP	CP	CP	CP	CP
ACID Transaction	×	×	×	×	×	○	○	○
サーバーサイド・ プロシージャ	×	JavaScript	Java	JavaScript	Lua	Java	×	Java
SQL風クエリ	○ (CQL)	×	×	×	×	×	×	○
高可用性(HA)構成	○	○	○	○	○	○	○	○
スケールアウト性	○	○	○	○	×	○	×	○
エラスティック性	○	○	○	○	×	○	×	× 有償版は△
永続性	○	○	○	△	△	△	○	○
遠隔地への データ複製	○ 非同期/同期	×(有償版は○) 非同期	○ 非同期	○ 非同期	○ 非同期	○ 非同期	×	○ 非同期

スケールアウト性：ノードを増やすことで性能がリニアにスケールするか？ エラスティック性：サービス無停止でノードの追加・停止できるか？

永続性：停電などの全体障害でもデータロストしないか？ Clodian Inc. & KK – Except where otherwise noted, content on this document is licensed

事例

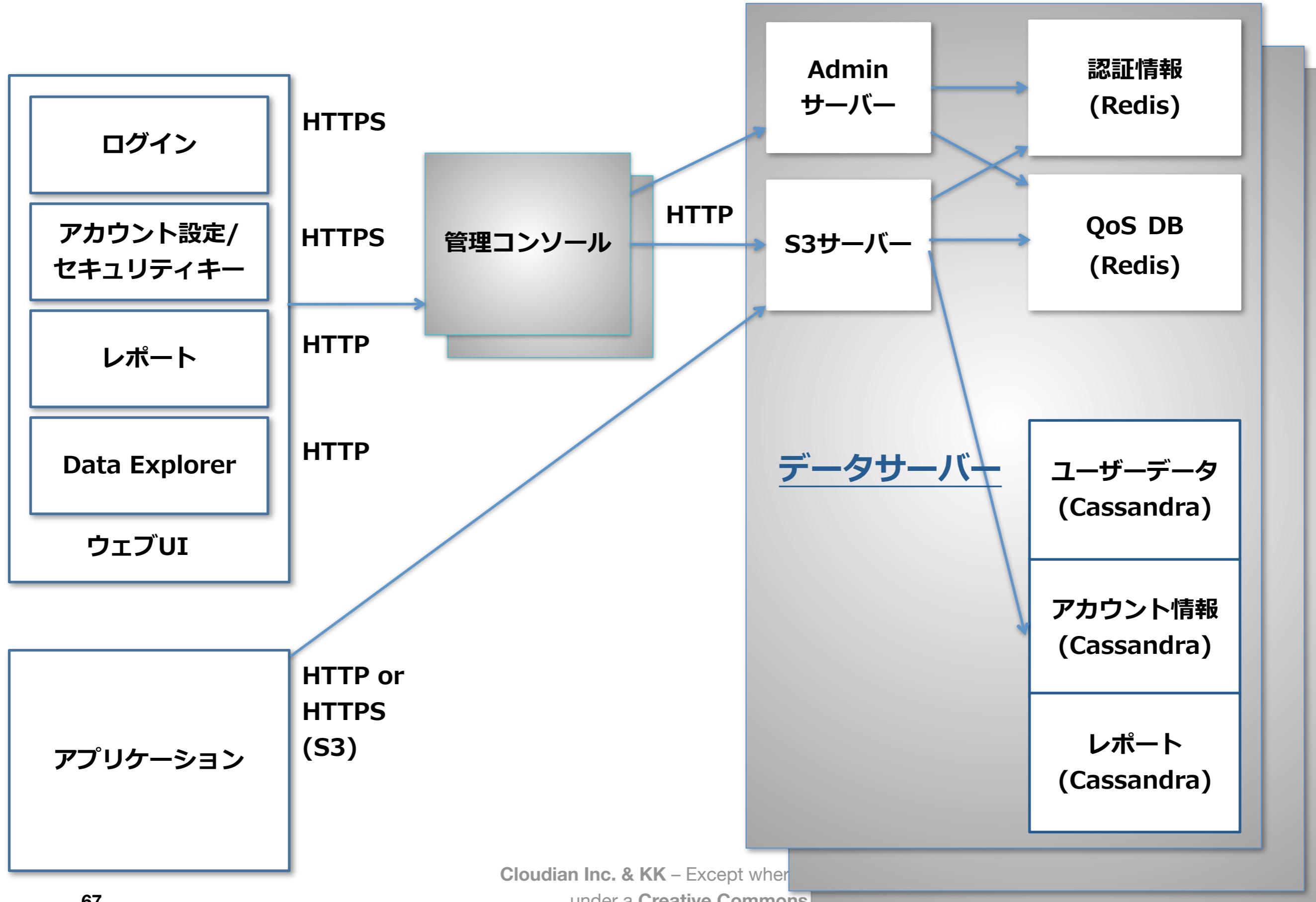
Cloudian

Redis
Cassandra
HyperStore



<http://cloudian.jp/cloud-storage-products/cloudian.html>

Cloudian™の論理アーキテクチャ



オブジェクトストアとしてのCassandra

• BLOBストレージ

- グループごとに column family を作成
- 行キー <バケットID>/<オブジェクトID>

• オブジェクトのメタデータ

- ACL (アクセスコントロールリスト)、オブジェクトのサイズ・・・
- Cassandra 行キャッシュを活用

• 巨大なS3オブジェクトのサポート

- **マルチパート** Amazon S3 multi-part APIを使ってアップロード
- **チャンкиング** 大きなオブジェクトを小さなチャンク（例 10MB）に分割して保存
- **HTTPレンジヘッダー** ダウンロード時は HEAD リクエストでオブジェクトのサイズを取得してから、スタートのバイト位置と長さを指定してダウンロード
- **HyperStore™** 巨大なオブジェクトをネイティブなファイルシステムに保存

大きなデータの扱いで性能が劣化

バリューサイズ 1KB

	遅延時間の中央値 (ミリ秒)		遅延時間の 95パーセンタイル値 (ミリ秒)		スループット (件/秒)	30分間で 処理した 件数
	Get	Put	Get	Put		
Cassandra 1.0	1.016	0.949	2.476	4.789	8,748	15,755,306
Cassandra 0.8.6	1.282	0.948	5.729	2.243	8,700	15,668,017

バリューサイズ 100KB

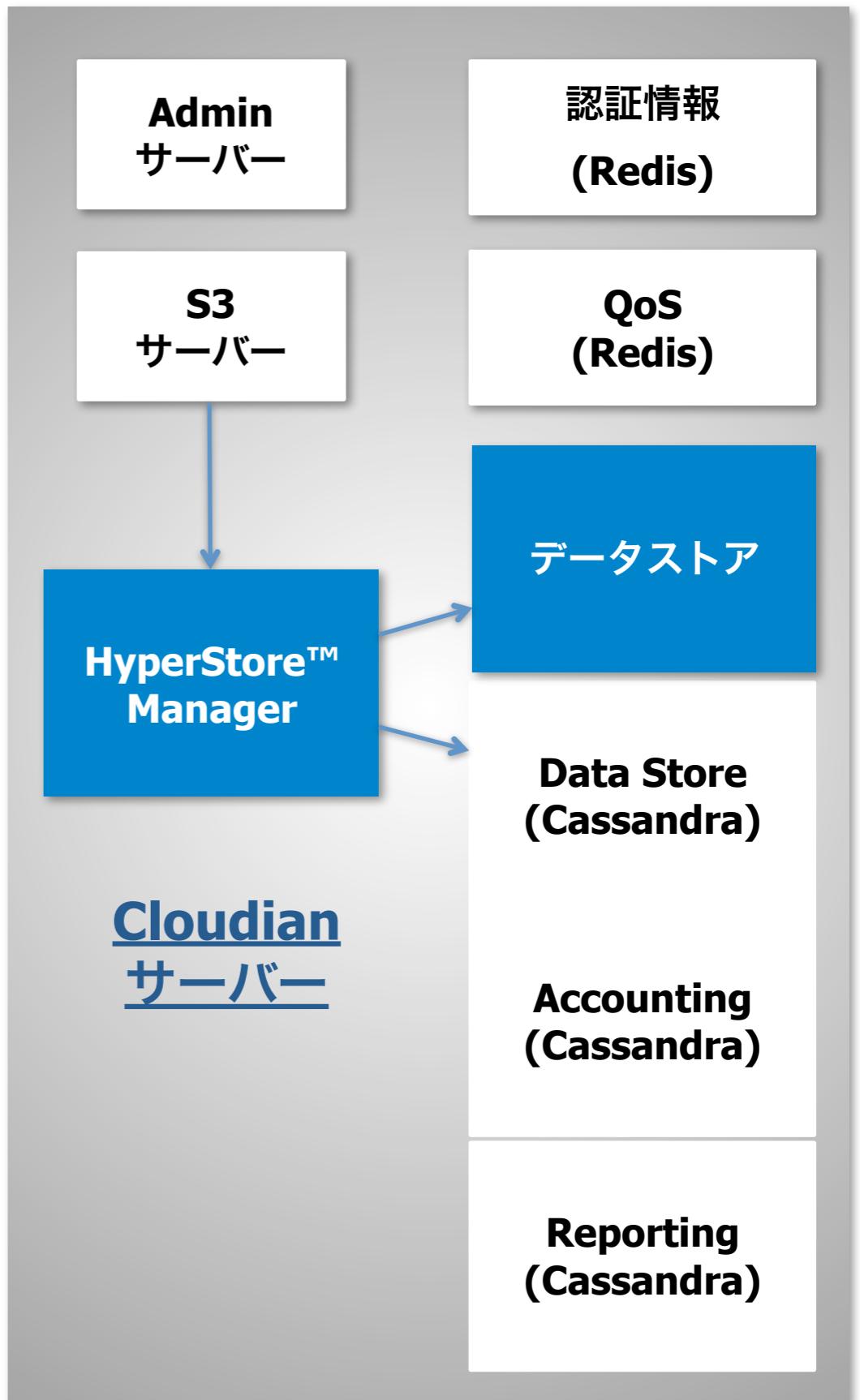
Cassandra 1.0	28.551	9.992	379.745	155.680	1,699	3,060,198
Cassandra 0.8.6	34.099	8.402	1,015.888	333.048	1,336	2,406,446

出典:「NOSQLの基礎知識」 リックテレコム、2012年4月出版

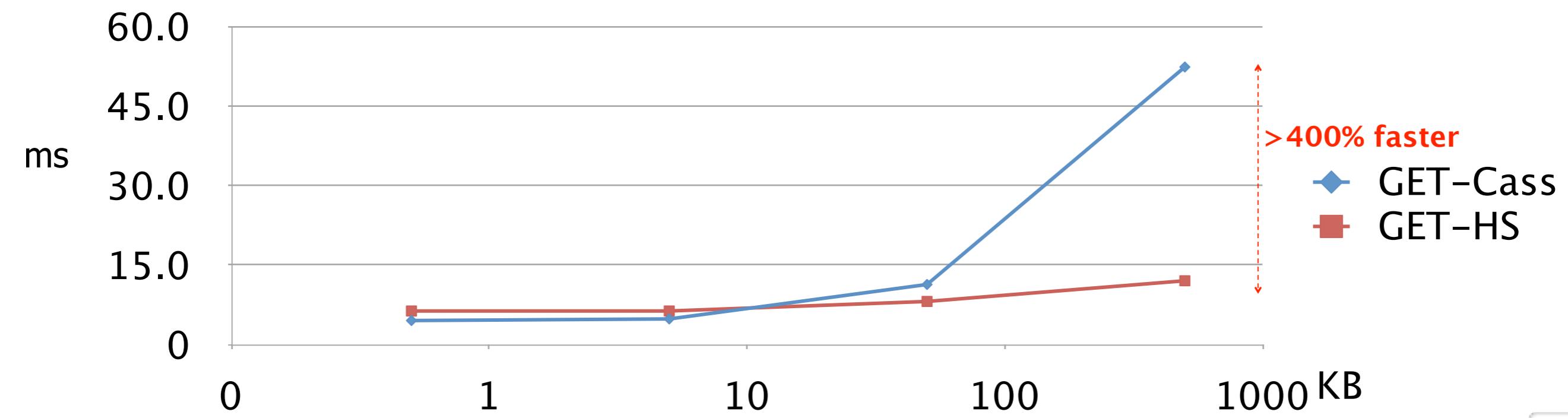
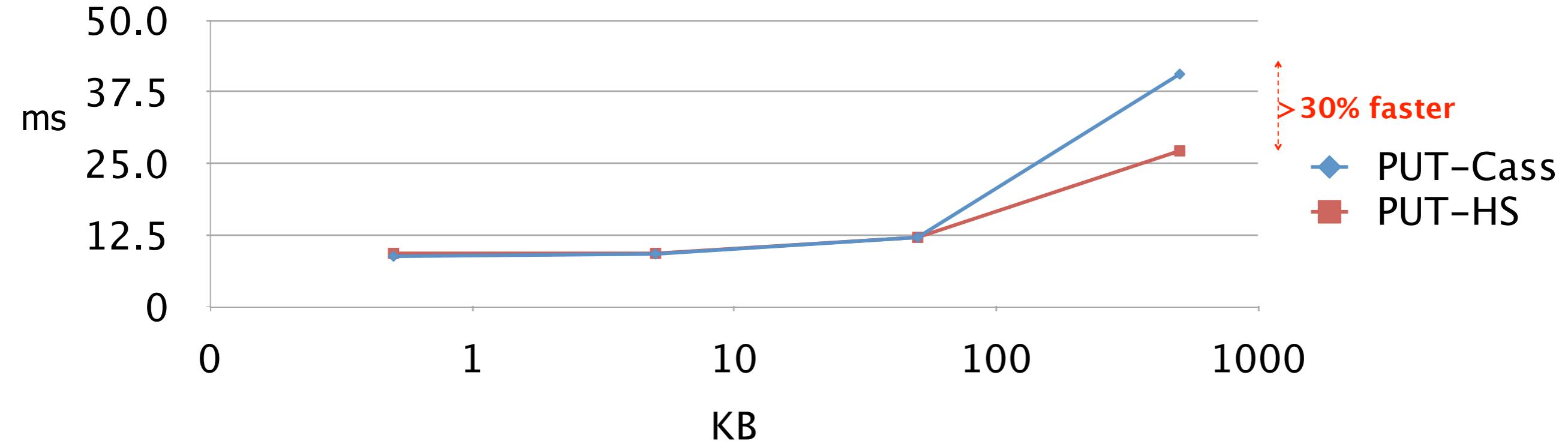
HyperStore™ (特許出願中)

HyperStore

- ・ストレージのハイブリッド化により
処理性能とディスク利用効率の向上を実現
- ・オブジェクトの大きさに応じて
最適なストレージを自動選択
- ・メタデータは引き続きCassandraに格納
- ・パーティショニング、レプリケーション、ノード
の死活監視は、Cassandraの分散機能を使用



HyperStore: レイテンシーの測定



推薦図書



NOSQLの基礎知識

ビッグデータを活かす
データベース技術

本橋信也 (著)

河野達也 (著)

鶴見利章 (著)

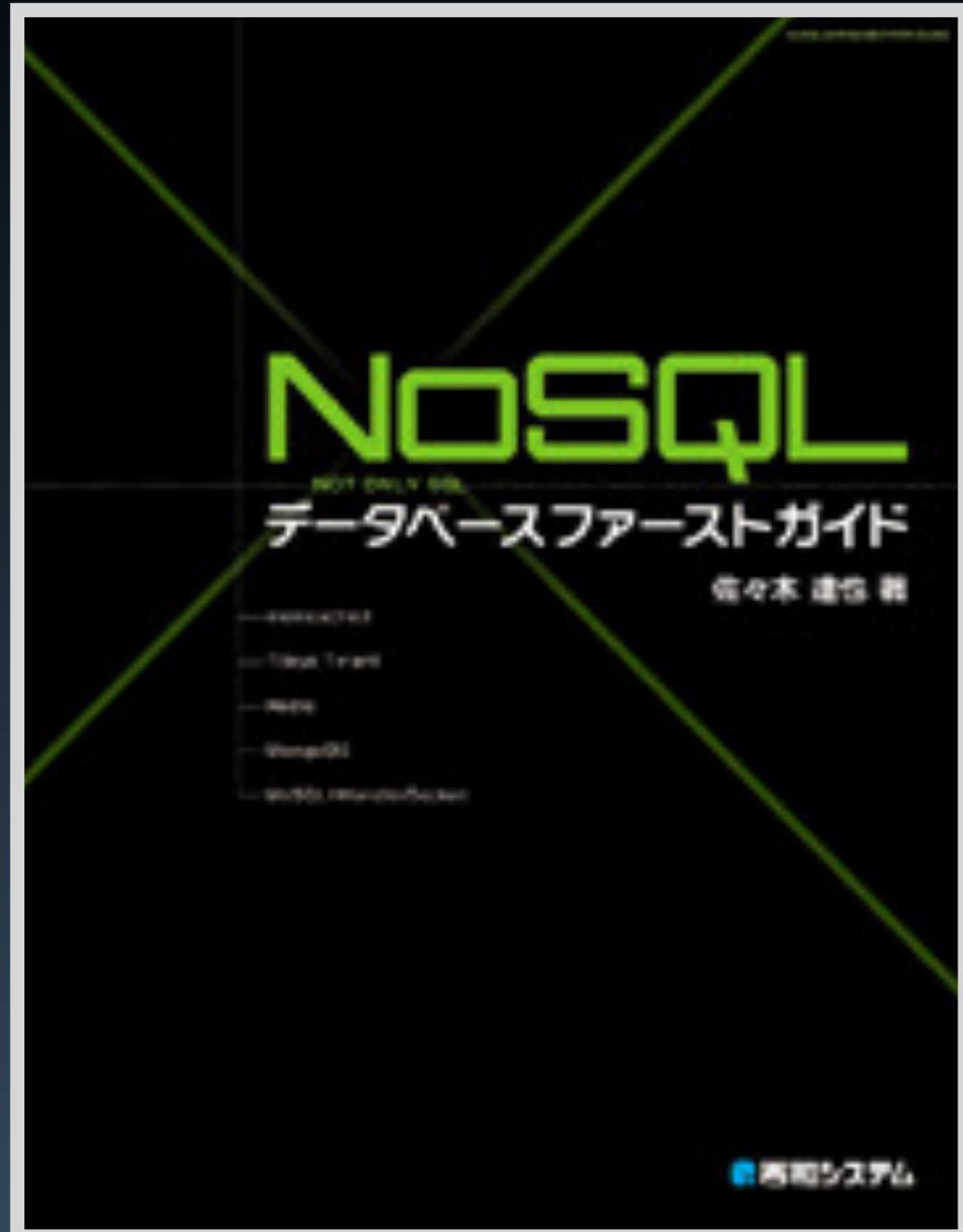
太田 洋 (監修)

256ページ

出版社: リックテレコム (2012/4/25)

ISBN-10: 4897978874

ISBN-13: 978-4897978871



NoSQLデータベースファーストガイド

佐々木 達也 (著)

MongoDB、Redis

232ページ

出版社: 秀和システム (2011/04)

ISBN-10: 4798029599

ISBN-13: 978-4798029597



NoSQLプログラミング 実践活用技法

Shashank Tiwari (著)
中村 泰久 (監修)
長尾 高弘 (翻訳)

HBase、Cassandra、
MongoDB、Redis

407ページ
出版社: 翔泳社 (2012/5/18)
ISBN-10: 4798126055
ISBN-13: 978-4798126050



まつもとゆきひろ コードの未来

まつもと ゆきひろ (著)

MongoDB、VoltDB

360ページ

出版社: 日経BP社 (2012/5/17)

ISBN-10: 4822234630

ISBN-13: 978-4822234638

この勉強会を終えると（再）

- NOSQL製品を分類するためのキーワードが理解できる
- 主要な製品の特徴が説明できる
- NOSQLの使いどころが説明できる

Questions?



<http://gplus.to/tatsuya6502>



<http://twitter.com/tatsuya6502>



Slides: <http://bit.ly/jdmc-nosql>

<https://github.com/tatsuya6502/nosql-ja/tree/jdmc-jul-2013/slides>

クリエイティブコモンズ

表示 - 非営利 3.0 非移植 ライセンス

CC Attribution-NonCommercial 3.0 Unported License

- この文書は **クリエイティブ・コモンズ 表示-非営利 3.0 非移植 (CC BY-NC 3.0) ライセンス** の下に提供されています。使用許諾条件を見るには、<http://creativecommons.org/licenses/by-nc/3.0/> をチェックしてください。



- あなたは以下の条件に従う場合に限り、自由に
 - 本作品を複製、頒布、展示、実演することができます。
 - 二次的著作物を作成することができます。
- あなたの従うべき条件は以下の通りです。
 - 表示 – あなたは原著者のクレジットを表示しなければなりません。
 - 非営利 – あなたはこの作品を営利目的で利用してはなりません。