



Apache CassandraとApache Sparkで 作るデータ解析プラットフォーム

株式会社INTHEFOREST

村岡 志保



Agenda

- NoSQLとは
- CassandraとはどのようなDB?
- Cassandraクラスタ構築
- IoTデータとは
- Cassandraの普遍性
- Sparkの力





Agenda

- NoSQLとは
- CassandraとはどのようなDB?
- Cassandraクラスタ構築
- IoTデータとは
- Cassandraの普遍性
- Sparkの力





NoSQLとは

NoSQLとはRDBMS以外のデータベース管理システムの一般的総称です。

非関係型の分散データストアなどが多いですが
厳密な関係性を省き

柔軟なデータ構造の格納や分散型システムの構築など
それぞれ特徴的な機能を持っています。

主なNoSQLとしてHBaseやCassandra、MongoDB、Redisなどがあります。

なぜ、このようなデータストアが生まれてきたのでしょうか？



NoSQL



NoSQLの存在意義（なぜNoSQLを選ぶのか？）

データベースで実際におこっていたこと。

- ・ 帳票からオンライントランザクションへ。
 - RowKey数の増大
 - 同時読込み・書込みの増大
- ・ 定時Batchから24時間運用へ
 - ショートトランザクションの増大
 - 負荷の読めない使用方法に
 - クライアント数増大



なぜRDBMSではダメなのか

多数のクライアントからの
同時読込み・書込みに対する
負荷分散が難しい

この一点に尽きます。





Agenda

- NoSQLとは
- **CassandraとはどのようなDB?**
- Cassandraクラスタ構築
- IoTデータとは
- Cassandraの普遍性
- Sparkの力



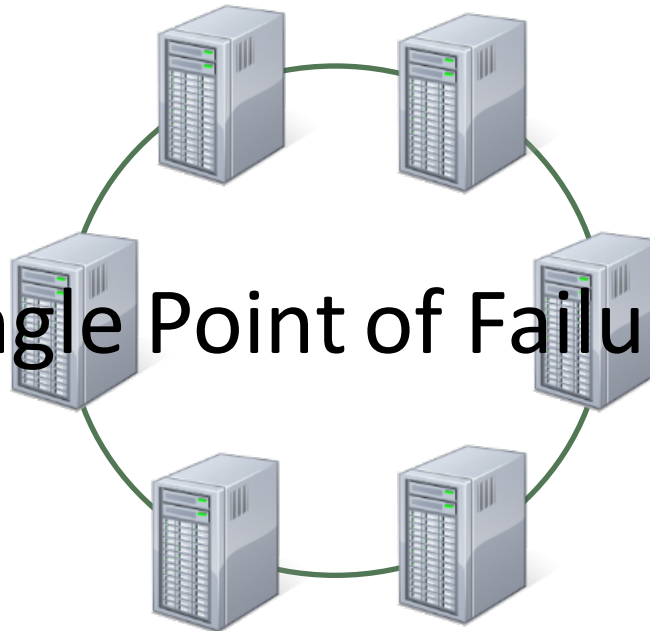
Cassandra ってどんなDB



大規模かつ線形にスケーラブルなNoSQLデータベース

- ・完全に分散され、単一障害点がない
複数のノードで分散ができ、ノードが1つ死んでも問題ないことを前提に設計されている

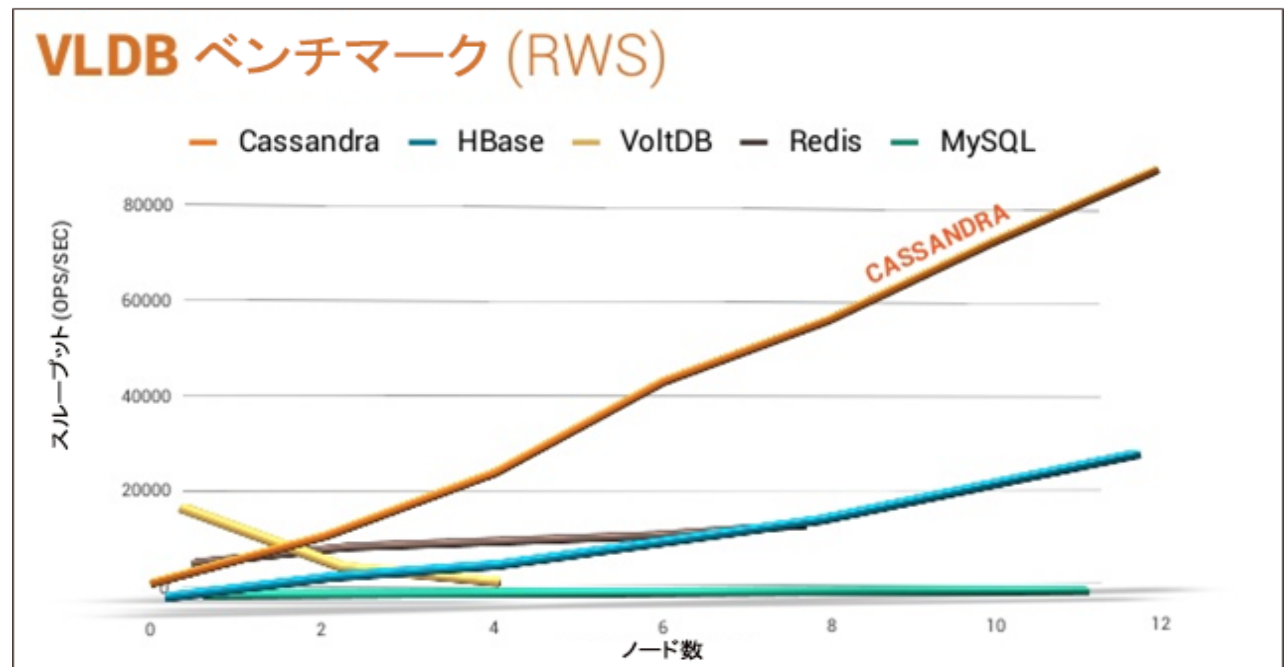
SPOF (Single Point of Failure) **が無い**



Cassandra ってどんなDB



- 無償かつオープンソースであり、開発者が強力にサポート
- 非常にパフォーマンスが高く、適切なユースケースでは、ほぼ線形で水平方向にスケーリング



Cassandraがどのように進化したのか



中核となるテクノロジー

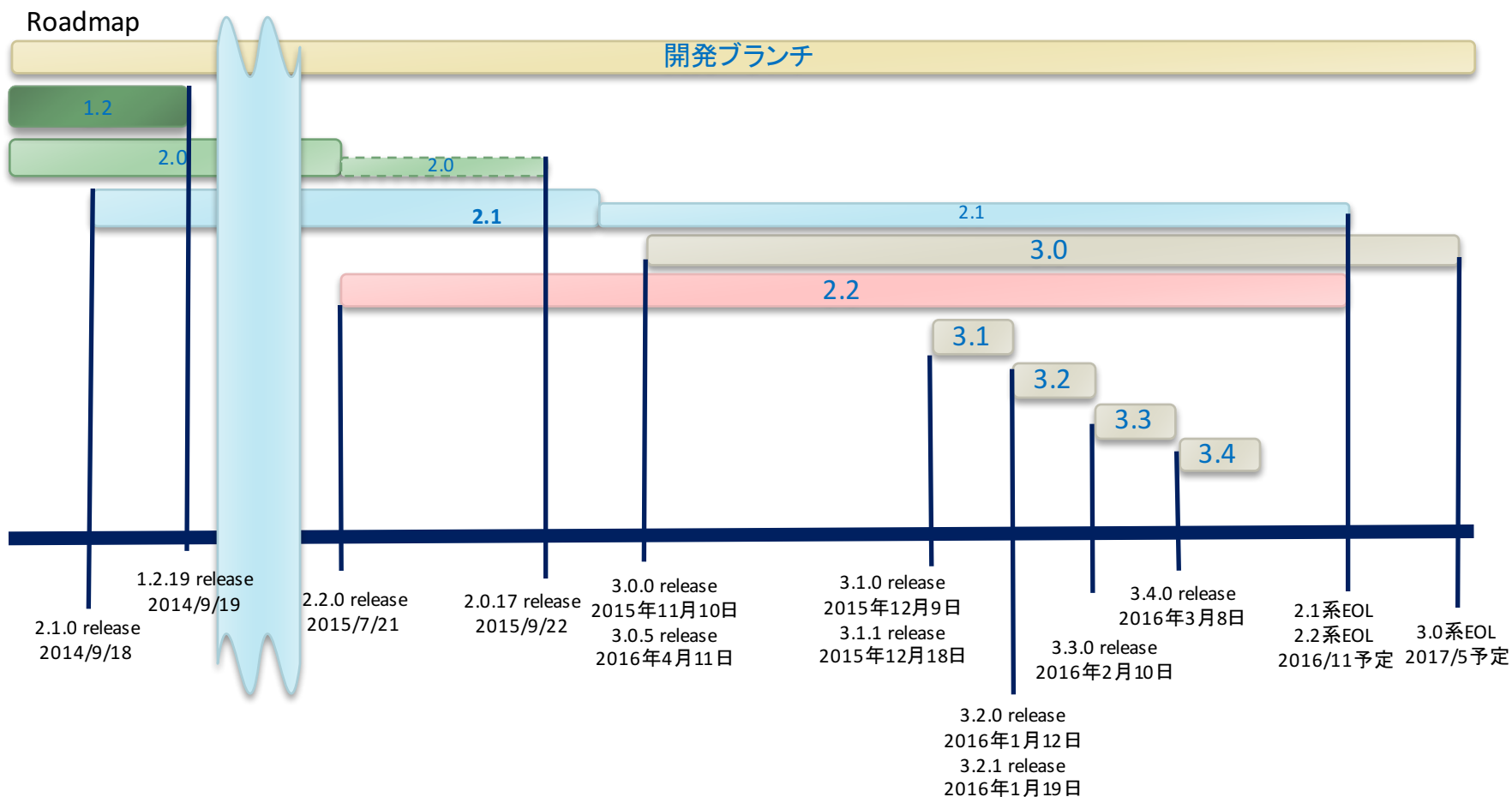
- Google BigTable : ストレージ・モデルの基盤
- Amazon Dynamo : 分散バックボーンの基盤
- Facebook : BigTableとDynamoの統合後に、Cassandraとしてリリース

急速な進化

- | | |
|------------------|------------------|
| • 0.6 - 2010年4月 | • 2.2 - 2015年5月 |
| • 0.7 - 2011年1月 | • 3.0 - 2015年11月 |
| • 0.8 - 2011年6月 | • 3.1 - 2015年12月 |
| • 1.0 - 2011年10月 | • 3.2 - 2016年1月 |
| • 1.1 - 2012年4月 | • 3.3 - 2016年2月 |
| • 1.2 - 2013年1月 | • 3.4 - 2016年3月 |
| • 2.0 - 2013年9月 | |
| • 2.1 - 2014年9月 | |



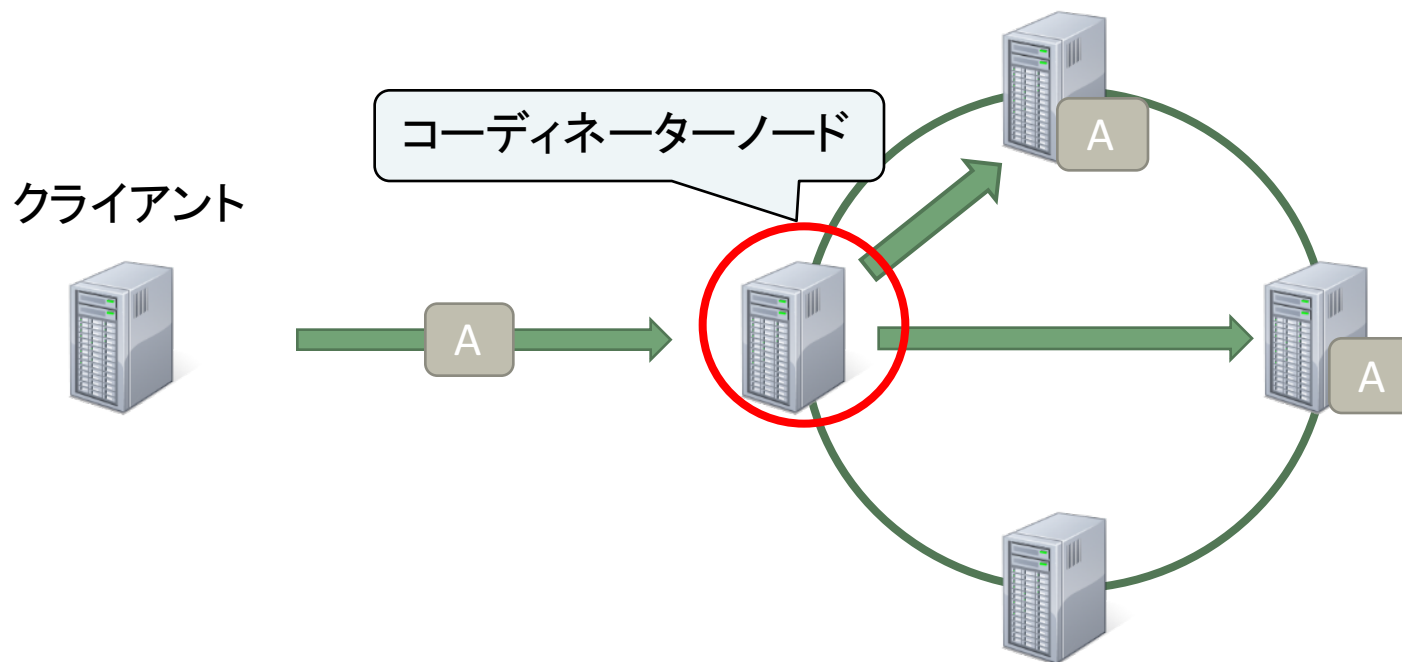
Cassandra Roadmap



データを入れる時の流れ



Cassandra4ノードでレプリケーション2つの場合

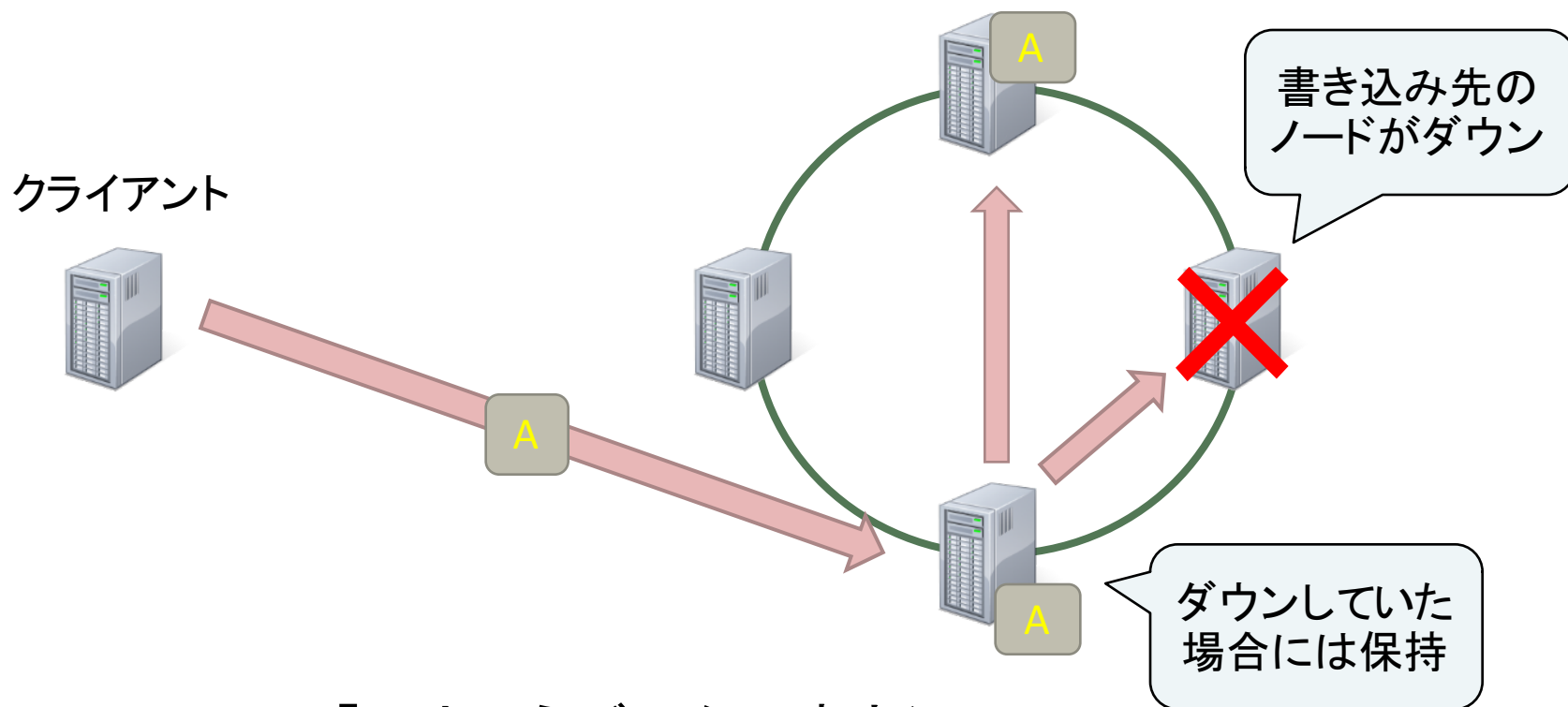


「A」というデータの書き込み

データを入れる時の流れ(ノードダウン)



Cassandra4ノードでレプリケーション2つの場合

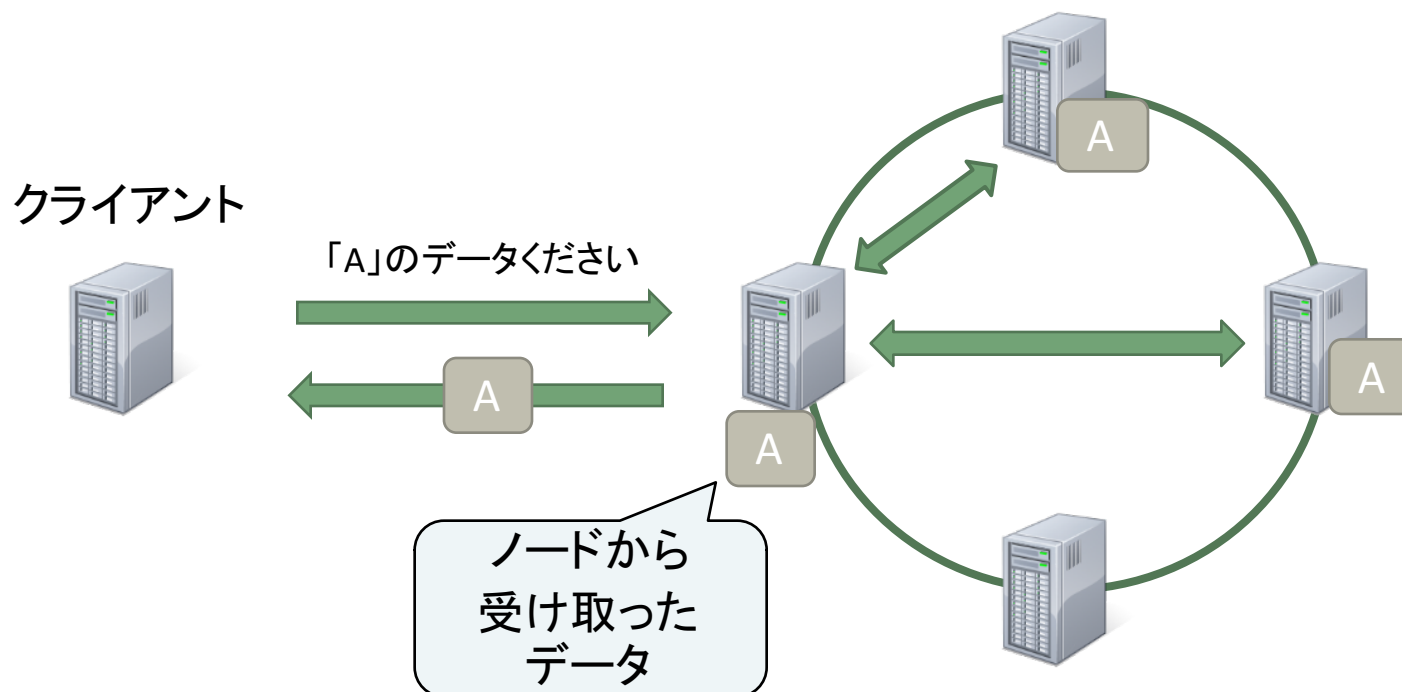


「A」というデータの書き込み

データを取得する時の流れ



Cassandra 4ノード構成でレプリケーションは2つの場合

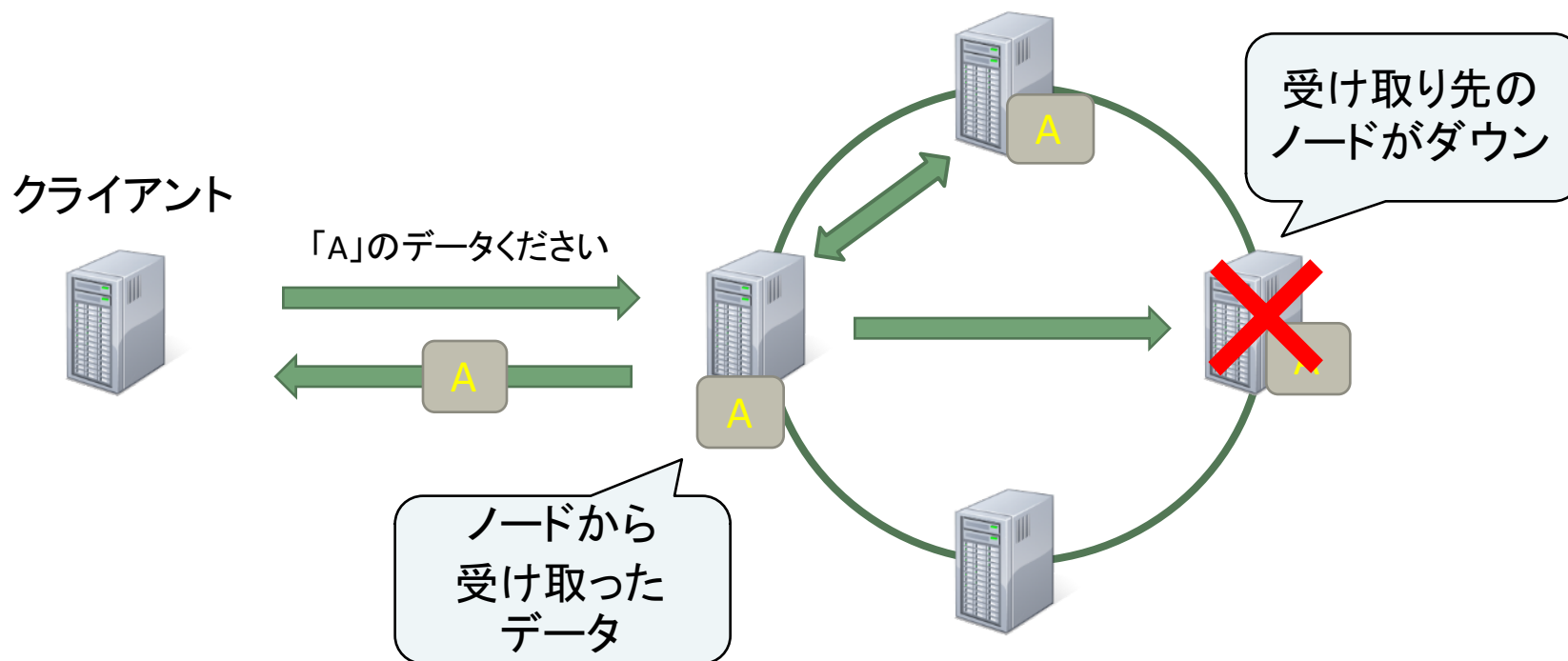


「A」というデータの読み込み

データを取得する時の流れ(ノードダウン)



Cassandra 4ノード構成でレプリケーションは2つの場合

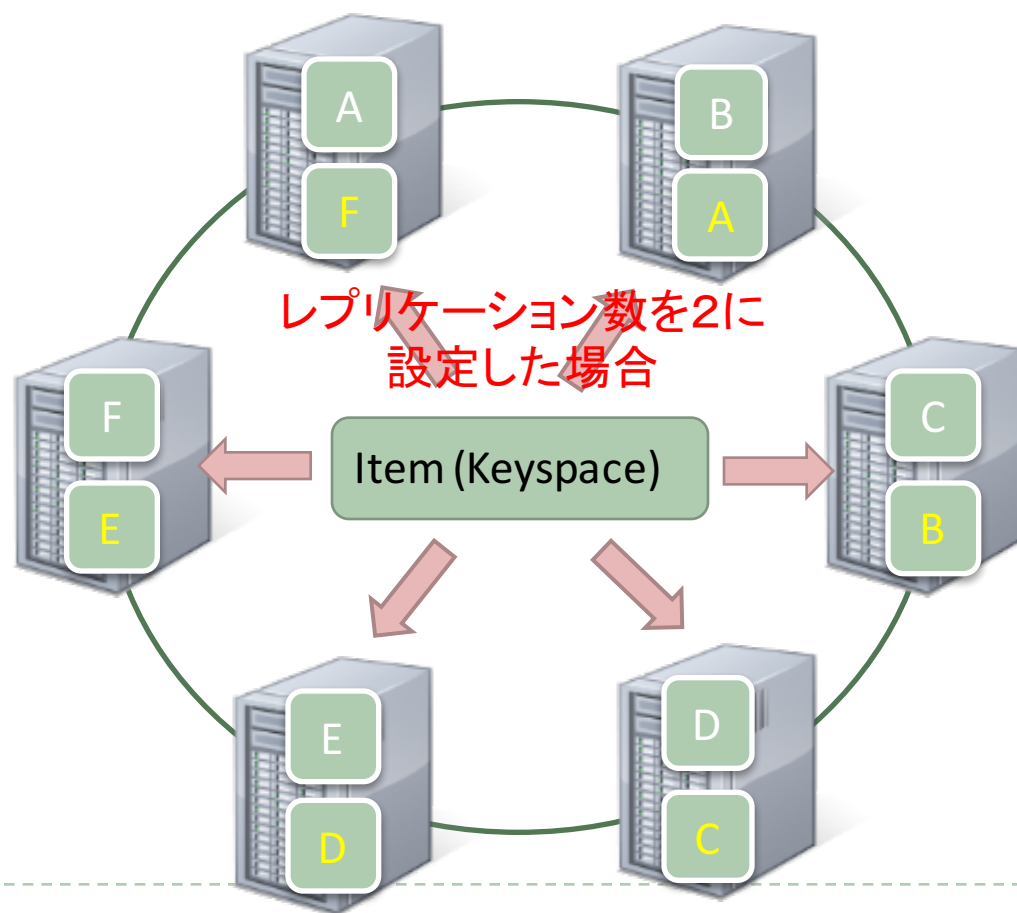


「A」というデータの読み込み



Cassandraのレプリケーション

SPOFを無くす為にレプリカを作成する。



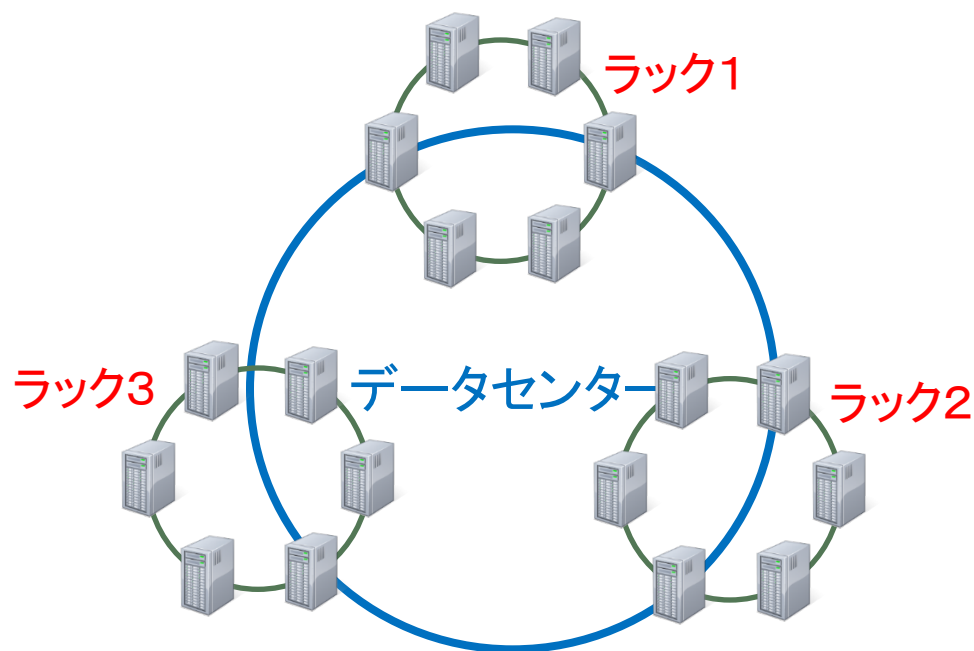


ラック、データセンターという概念

ノード: 1つのCassandraのインスタンスの仮想または物理ホスト

ラック: 物理的に関連する複数のノードの論理的なグループ

データセンター: ラックの集合の論理的なグループ



Cassandraの良い所



- ノードが死んでも緊急対応しなくて良い(ポリシーにもよるが)
- 分散して読書きするのでパフォーマンスも落ちにくい
- 100ノードなど多くなった時、管理が現実的





Agenda

- ◆ NoSQL
- ◆ CassandraってどんなDB?
- ◆ **Cassandra クラスタ構築**
- IoTデータとは
- Cassandraの普遍性
- Sparkの力





Cassandraクラスタ構築の基礎知識

必要環境(最低必要な環境)

OSは Linux、Unix 又はWindows 7以上

Cassandra 3系以降はJava 8 必須

Python 2.7 必須

必要データ領域

/var/lib/Cassandra/hints	(ヒント)
/data	(データ)
/commitlog	(コミットログ)
/saved_caches	(キャッシュ)

(設定ファイルで保存先は変更可能)

ネットワークの以下2つのPortが開いている事

9042 クライアント用Port

7000 ノード間のコミュニケーション用Port

(ポート番号は変更可能)



環境構築



Disable Swap の挙動の変更(推奨)

OOM Killer によって JVM を落とさないように設定する

```
$ sudo /sbin/swapoff --all
```

NTPで時間同期をする(推奨)

書き込まれるデータの優先度はタイムスタンプなので
ノード間同じ時間設定にする事は重要

```
$ sudo yum install ntp
```



環境構築



cassandra.yaml で行う設定

クラスタの形を決める設定

• cluster_name	クラスタの名前
• Partitioner	パーティショナーの指定
• hints_directory	ヒントを入れるディレクトリの設定
• data_file_directories	SSTableを入れるディレクトリの指定
• commitlog_directory	コミットログを入れるディレクトリの指定
• save_caches_directory	キャッシュを入れるディレクトリの指定
• seed_provider	シードノードの指定
• listen_address	どのアドレスで受け取るかの指定
• rpc_address	どのアドレスで受け付けるかの指定
• endpoint_snitch	スニッチの指定

多ノードでクラスタを組む場合確実に必要な設定は赤い部分
(他はデフォルトがすでに指定されている)



環境構築



seed_provider

立ち上げた3ノードのうち、一つをシードノードと決めて
そのアドレスを全てのノードのcassandra.yamlの以下の部分に書き込みます

```
# any class that implements the SeedProvider interface and has a
# constructor that takes a Map<String, String> of parameters will do.
seed_provider:
  # Addresses of hosts that are deemed contact points.
  # Cassandra nodes use this list of hosts to find each other and learn
  # the topology of the ring. You must change this if you are running
  # multiple nodes!
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      # seeds is actually a comma-delimited list of addresses.
      # Ex: "<ip1>,<ip2>,<ip3>"
      - seeds: "192.168.0.154"
```

環境構築



listen_address

そのノードが持っているNICのどれで受け付けるかの設定
(基本そのノードのアドレス)

listen_address: 自分のアドレス

rpc_address

そのノードが何処からのノードに対して受付を許可するかの設定

rpc_address: 自分のアドレス



環境構築



- スニッチ
ノード配置設定

設定場所: `cassandra.yaml`
どのようにリングを作るかを設定します。

```
endpoint_snitch: SimpleSnitch  
endpoint_snitch: PropertyFileSnitch  
endpoint_snitch: GossipingPropertyFileSnitch
```

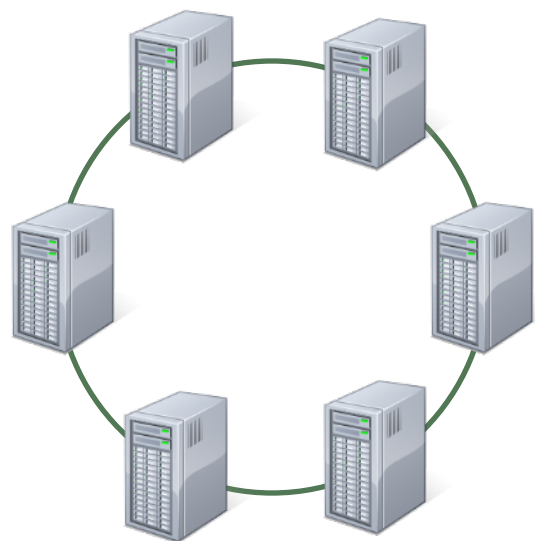




SimpleSnitch

単純なリング構成

単純な1リングのみで
全体を構成する



環境構築



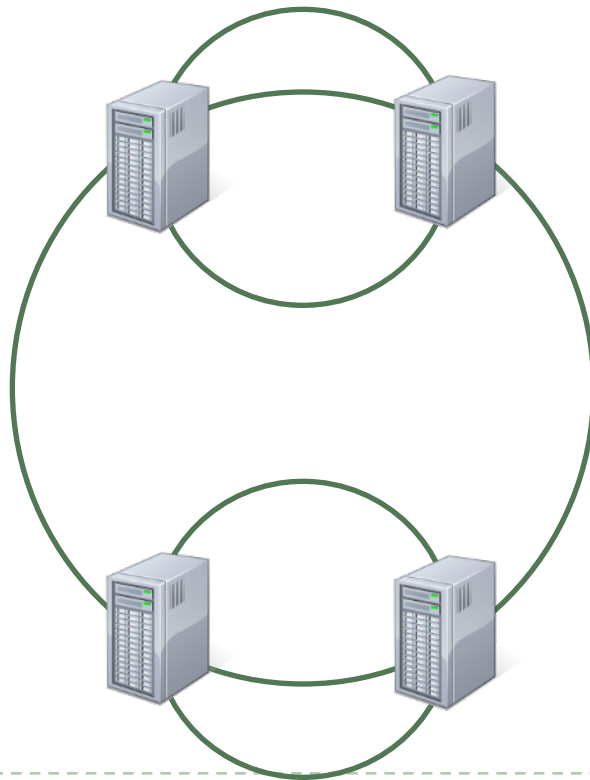
PropertyFileSnitch

設定ファイルでDCを分ける

/etc/cassandra/cassandra-topology.properties

DC1:RAC1

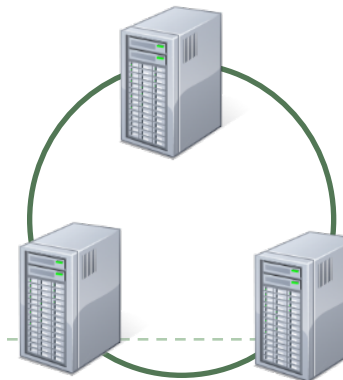
DC1:RAC2



Cassandra Node IP=Data Center:Rack

```
192.168.1.6=DC1:RAC1
172.16.2.1=DC1:RAC1
192.168.1.8=DC1:RAC2
172.16.2.2=DC1:RAC2
192.168.1.7=DC2:RAC1
172.16.2.5=DC2:RAC1
192.168.1.9=DC2:RAC1
```

default for unknown nodes
default=DC1:RAC1



DC2:RAC1

環境構築

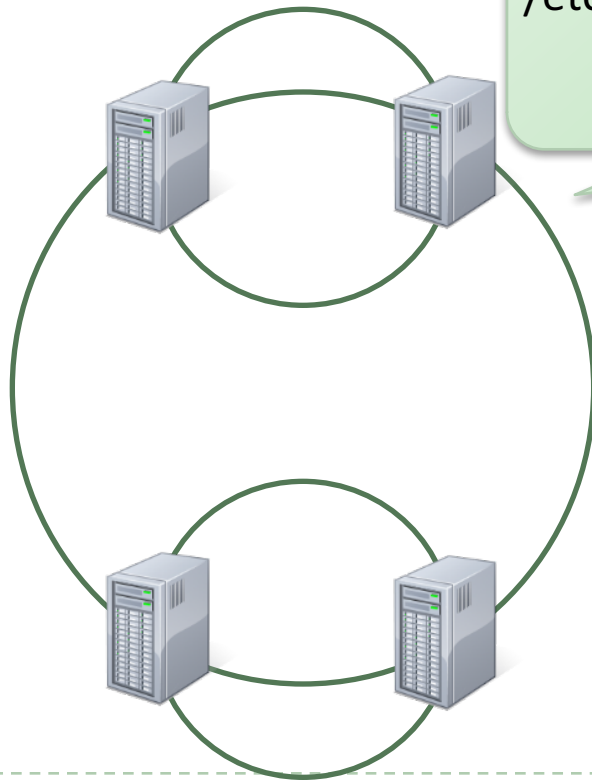


GossipPropertyFileSnitch

設定ファイルで指定した情報をGossipでばらまく

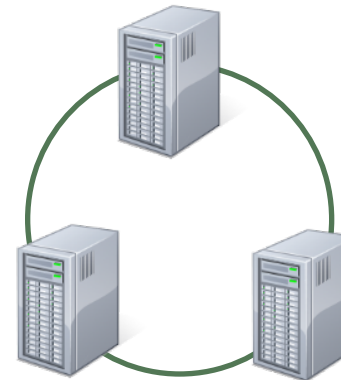
DC1:RAC1

DC1:RAC2



/etc/cassandra/default/cassandra-rackdc.properties

dc=DC1
rack=RAC1



DC2:RAC1

環境構築



- ストラテジ
データ配置設定

設定場所: KeySpaceMeta情報
データをどのリングに配置するかを設定します。

```
placement_strategy = 'SimpleStrategy'  
placement_strategy = 'NetworkTopologyStrategy'
```

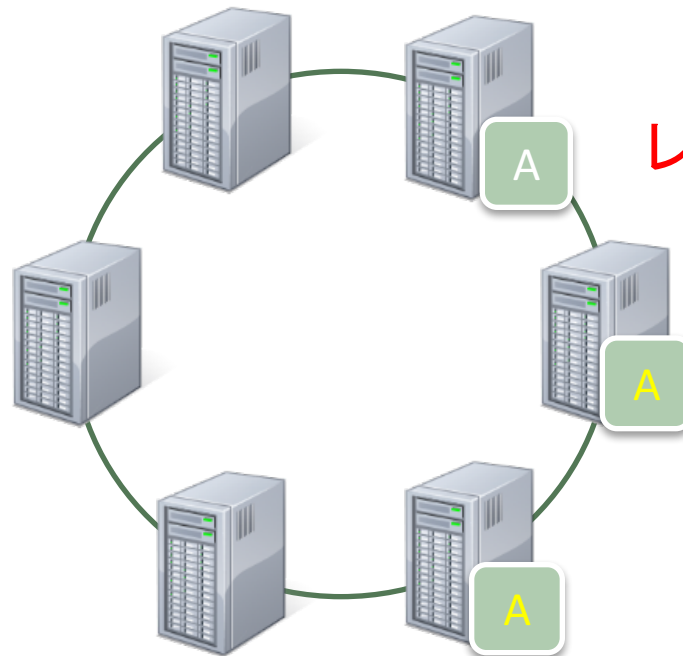


環境構築



SimpleStrategy

replication_factorの設定(レプリカの数)
に沿って右隣へレプリカを設定



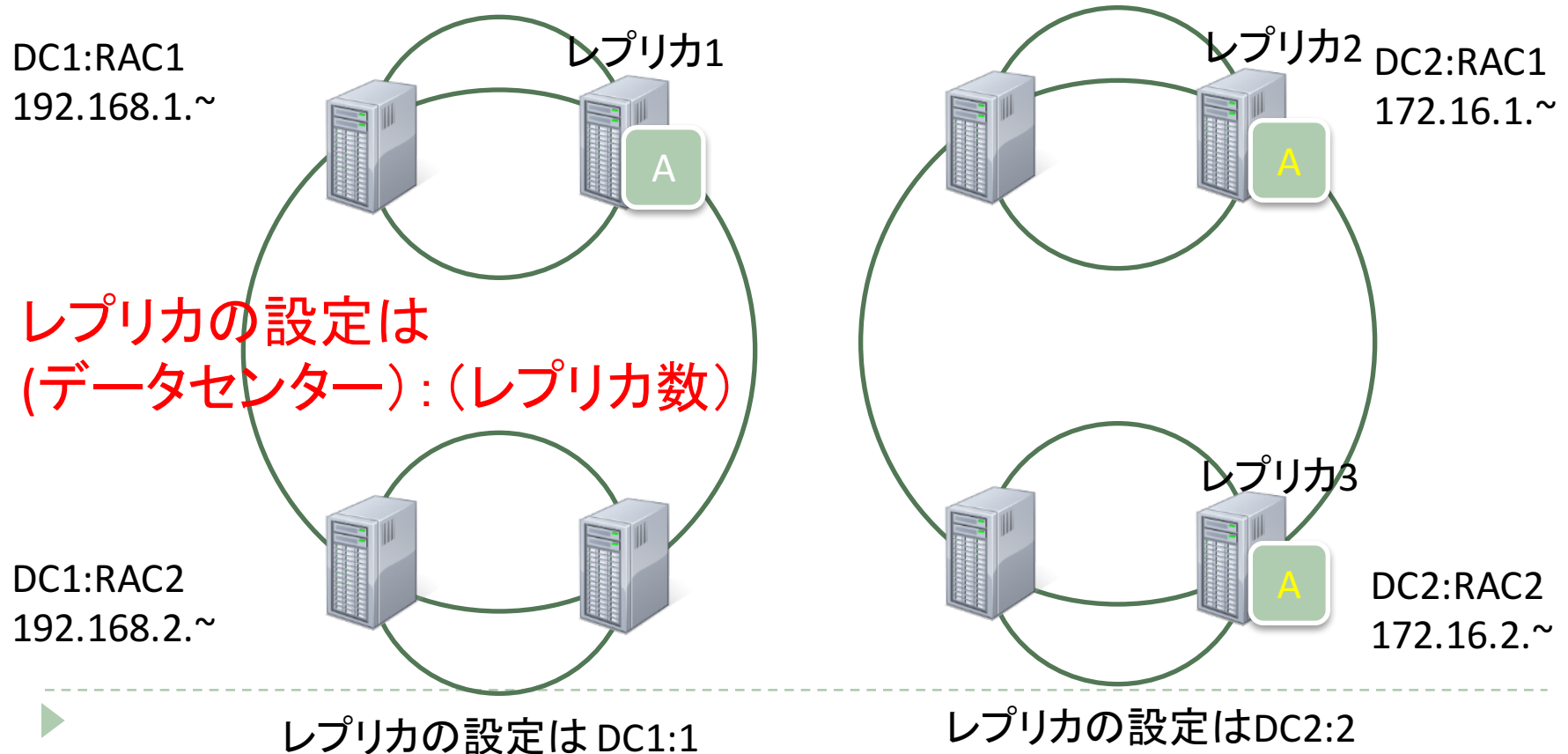
レプリカの数3の場合

環境構築



NetworkTopologyStrategy

KeySpaceMETA情報指定通り配置



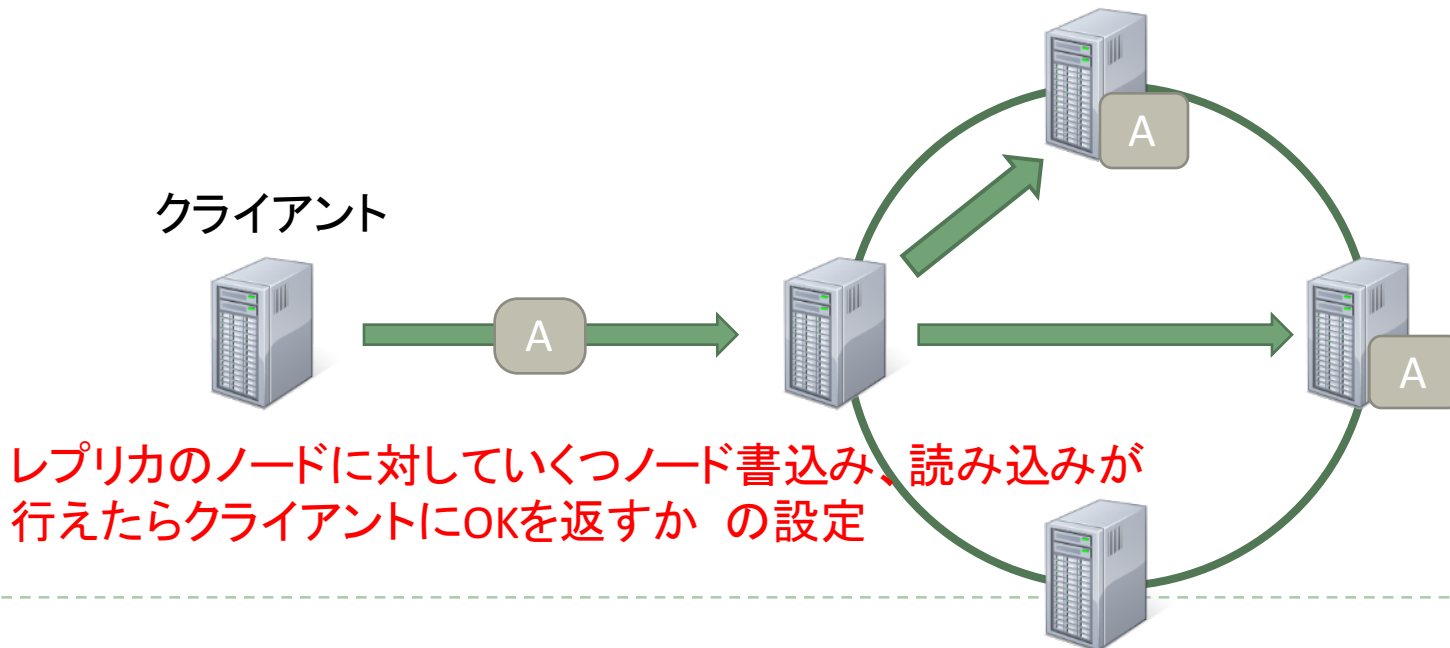
環境構築



Cassandraの分散構造

- ConsistencyLevel

設定場所: 問い合わせ時
どのようにデータの保全性を確保するか設定する



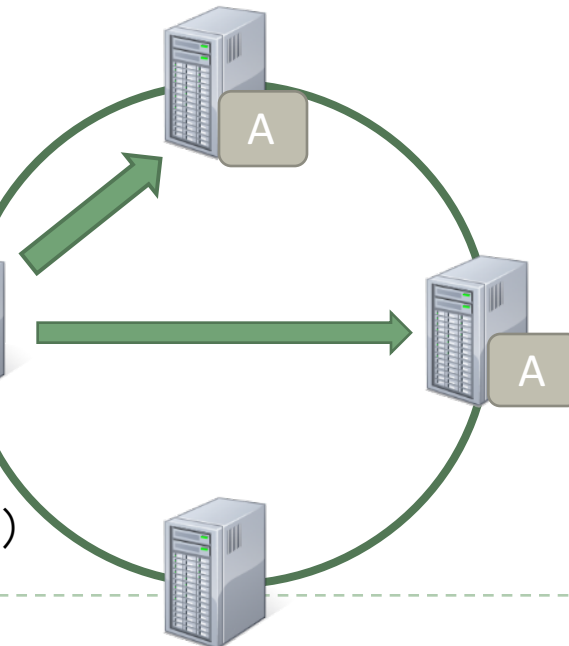
環境構築



Cassandraはロールバックの概念が無い

とにかく書き込んで！
失敗しても戻さなくていいから
でも正常に書き込めた数は
カウントするよ

クライアント



この**カウント数**でOKかNGか
判断する基準がCL(コンステンシレベル)

環境構築



ConsistencyLevel

Write

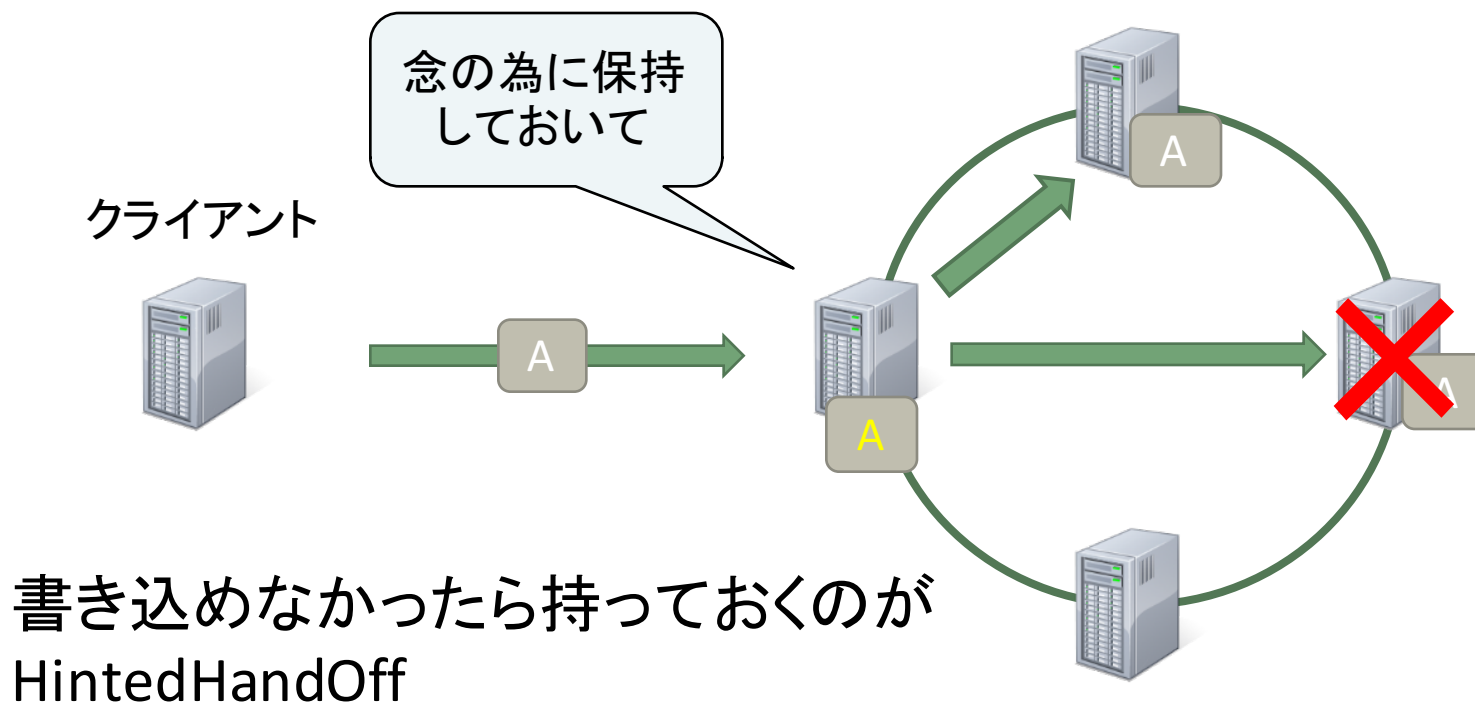
ANY	書き込みがHintedHandoffを含む少なくとも1ノードに書かれたことを保証。
ONE	書き込みが少なくとも1つのノードに書かれたことを保証。
TWO	書き込みが少なくとも2つのノードに書かれたことを保証。
THREE	書き込みが少なくとも3つのノードに書かれたことを保証。
QUORUM	書き込みが少なくとも $N/2+1$ 個(過半数)のノードに書かれたことを保証。
LOCAL_QUORUM	ローカルのデータセンター内の $N/2+1$ 個のノードに書かれたことを保証
EACH_QUORUM	リモートのデータセンター内の $N/2+1$ 個のノードに書かれたことを保証
ALL	書き込みがすべてのノードに書かれたことを保証。



HintedHandOff



Cassandra 4ノード構成でレプリケーションは2つの場合



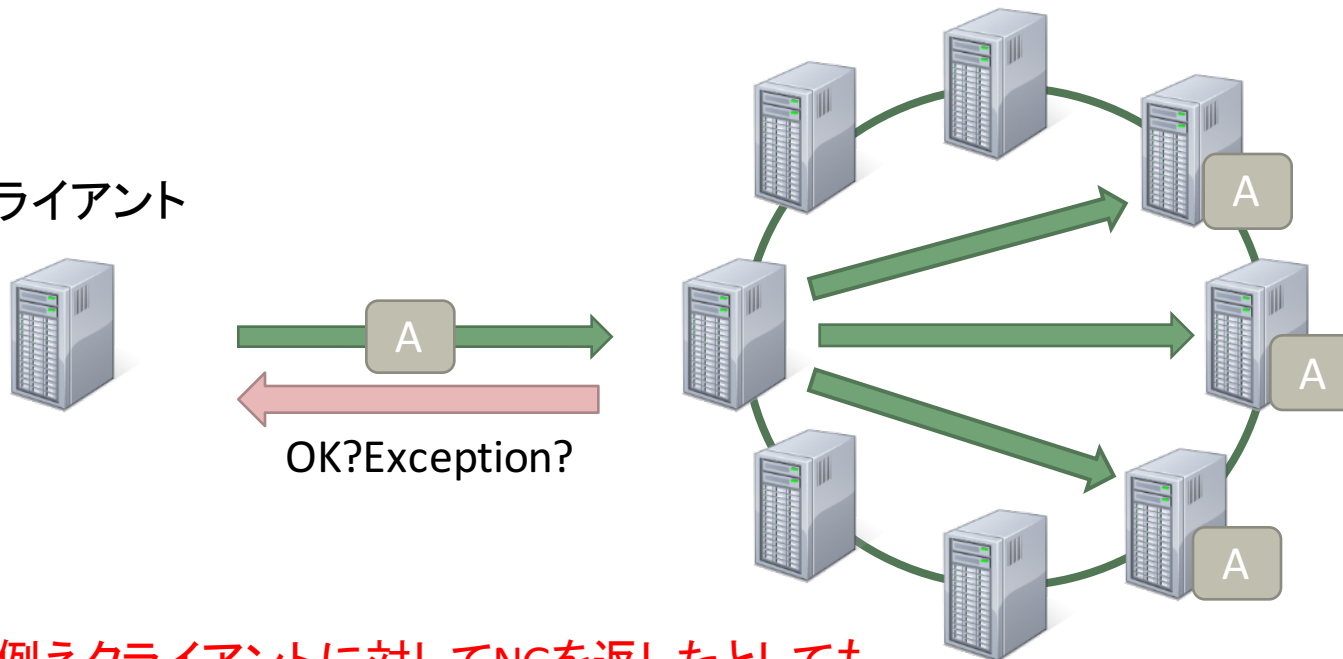
ノードが復旧した際にHintedHandOffをしたデータが
復旧したノードへ送られ、更新される

Write ONE, TWO, THREE



単純にONEは1、TWOは2、THREEは3台
書き込めた時点で、クライアントに正常に書き込めた事を通知します

クライアント



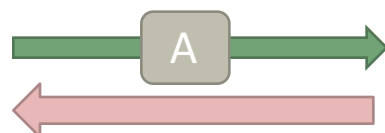
例えばクライアントに対してNGを返したとしても
書き込めたノードに対してはそのままなので
データの更新は行われている



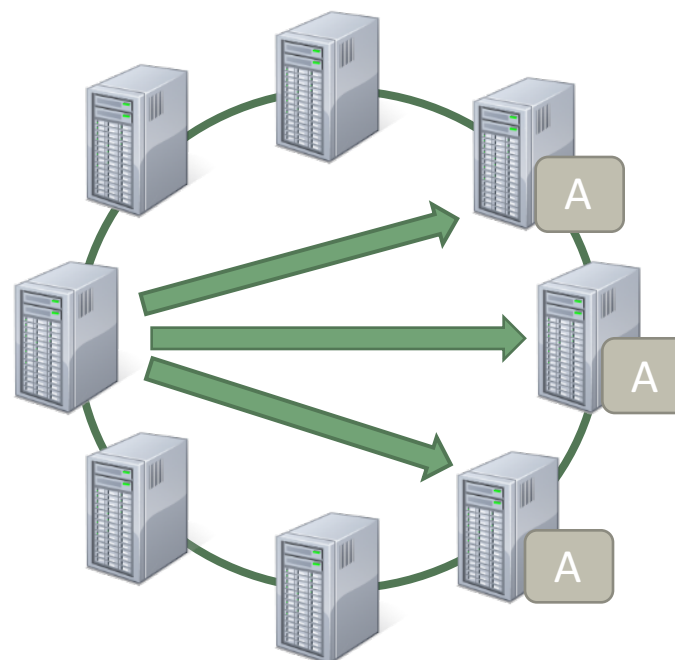
Write QUORUM

その書き込み先のレプリケーション数/ $2 + 1$ に書き込めた時点で
正常に書き込めた事を通知します

クライアント



OK?Exception?

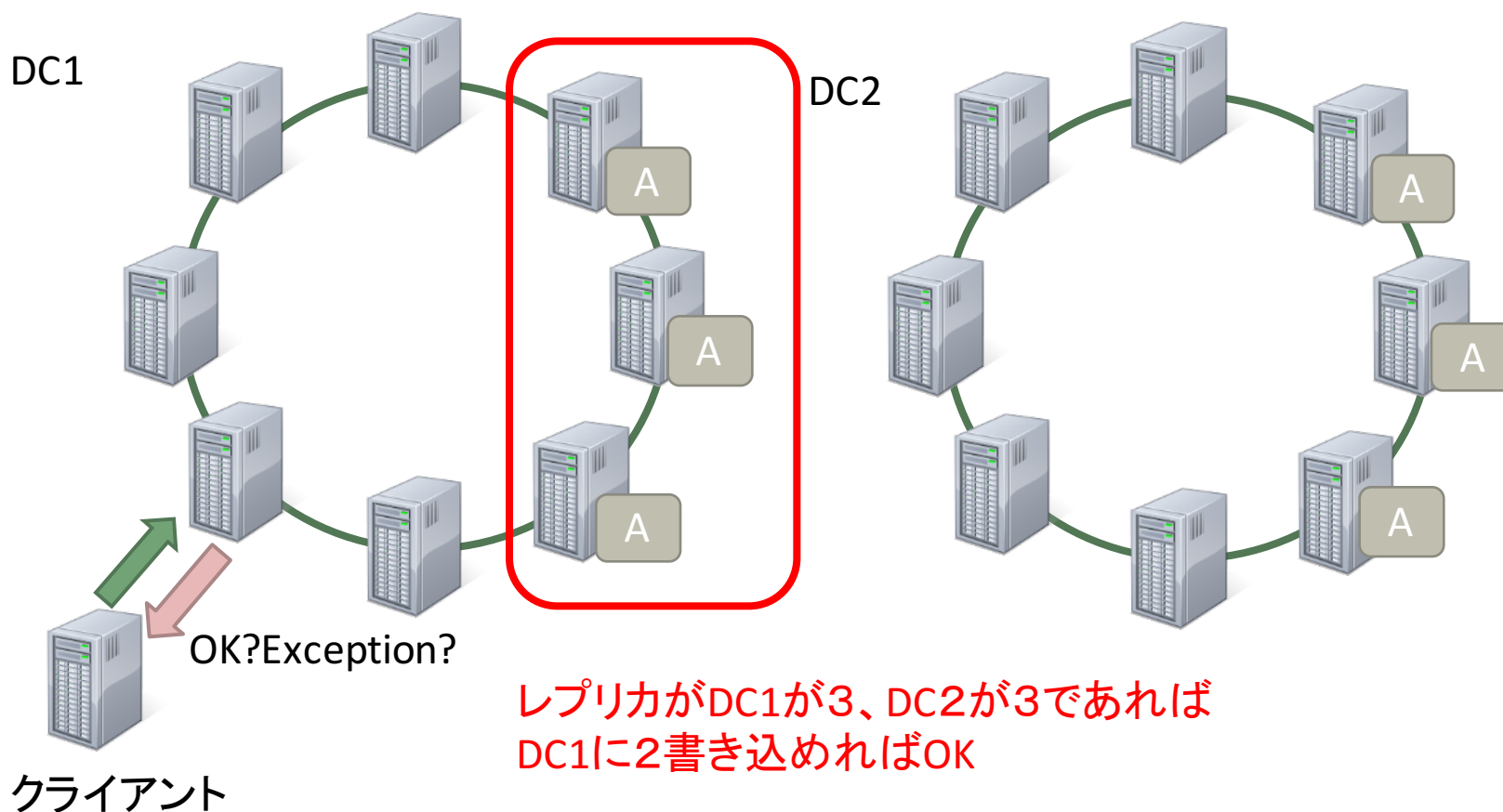


レプリカが3であれば2書き込めればOK



Write LOCAL_QUORUM

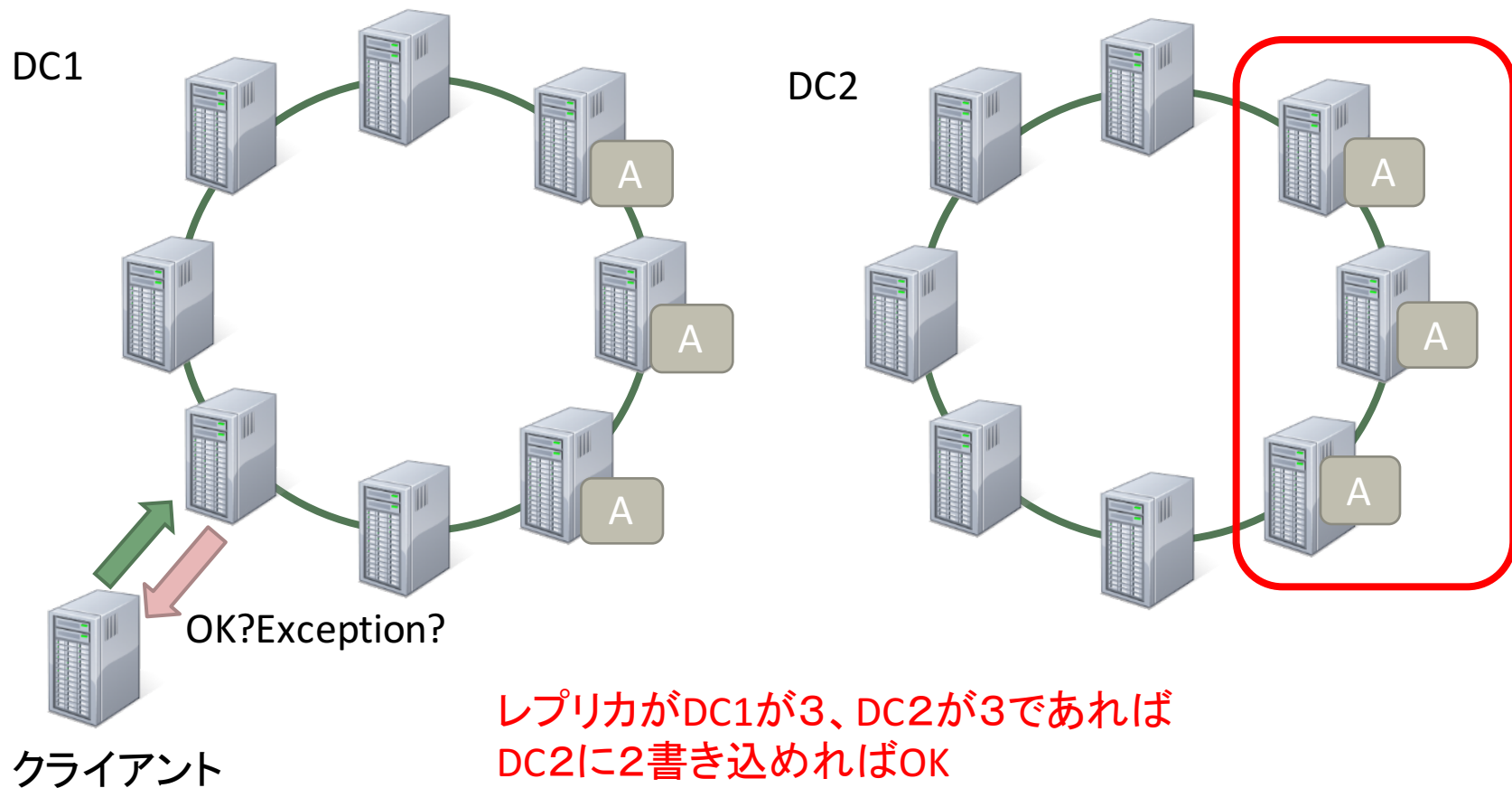
その書込み先のDCのレプリケーション数/2 + 1 に書き込めた時点で
正常に書き込めた事を通知します



Write EACH_QUORUM



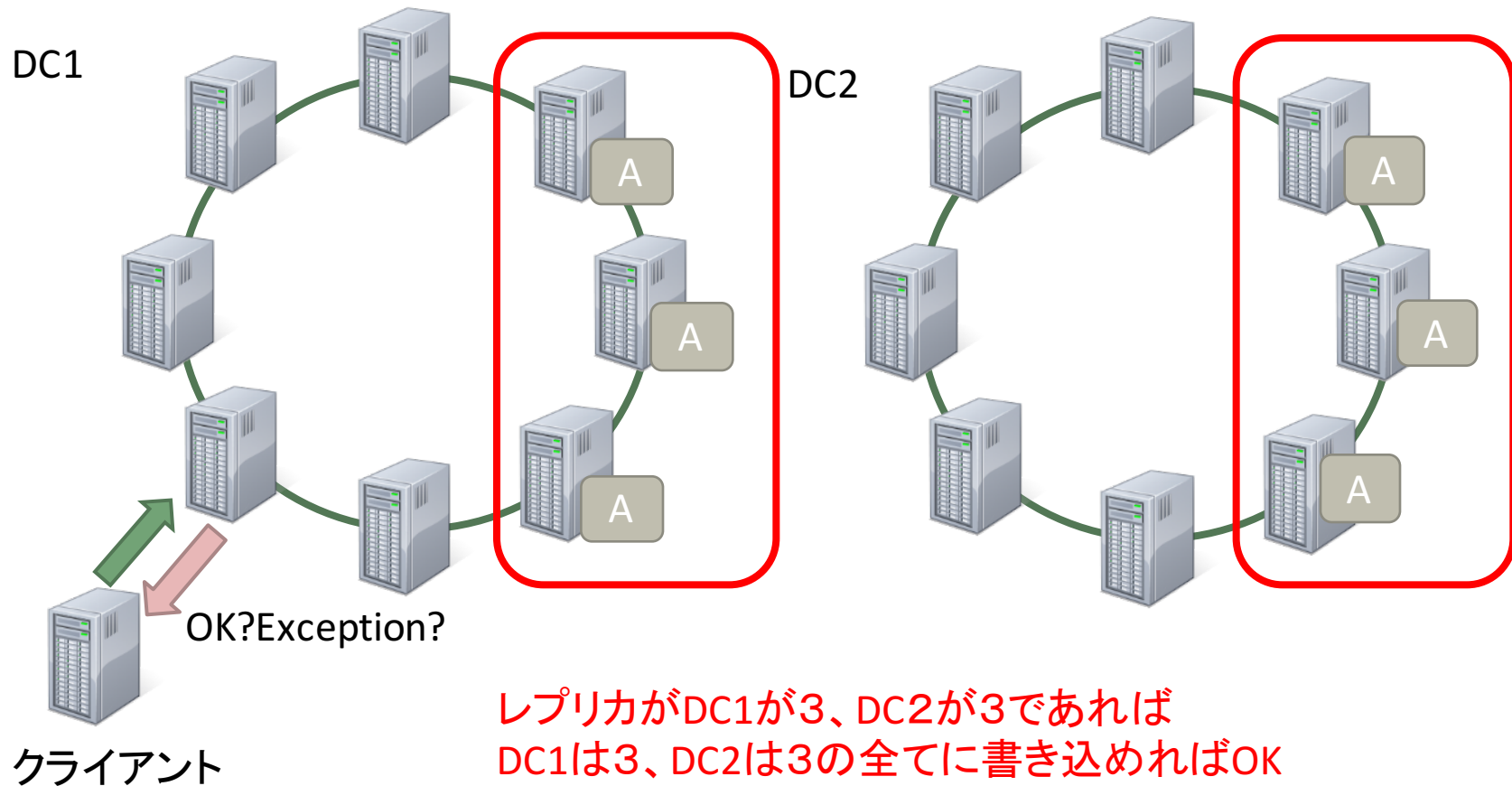
その書き込み先のDCではないレプリケーション数/2 + 1 に書き込めた時点で
正常に書き込めた事を通知します



Write ALL



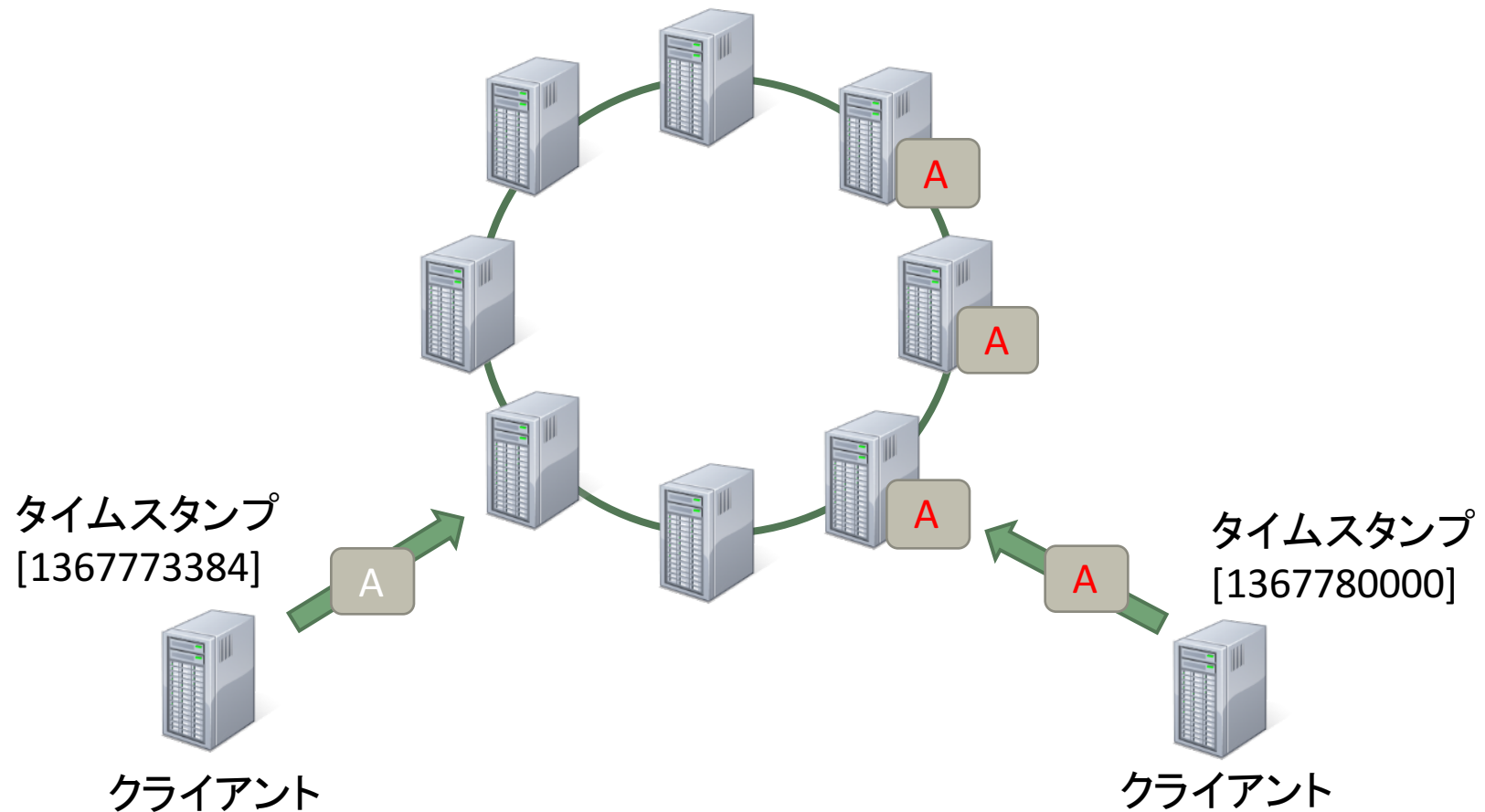
全てのレプリケーションに対して書き込みが行えた場合
正常に書き込めた事を通知する



データの優位性



Cassandraのデータの優位性はタイムスタンプを基準とします
タイムスタンプが新しいデータが残るようになります



Cassandraのシステム構造



ConsistencyLevel

Read

レプリカの数え方はWriteのConsistencyLevelと同じ、ANYは無い

ONE	最初にリターンがあったレコードをそのまま返答
TWO	最初にリターンがあった2つのレコードのタイムスタンプを比較しもっとも新しいレコードを返答
THREE	最初にリターンがあった3つのレコードのタイムスタンプを比較しもっとも新しいレコードを返答
QUORUM	最初にリターンがあった $N/2+1$ (過半数) つのレコードのタイムスタンプを比較しもっとも新しいレコードを返答
LOCAL_QUORUM	ローカルのデータセンター内の $N/2+1$ 個のノードから返答があったレコードのタイムスタンプを比較しもっとも新しいレコードを返答
EACH_QUORUM	リモートのデータセンター内の $N/2+1$ 個のノードから返答があったレコードのタイムスタンプを比較しもっとも新しいレコードを返答
ALL	すべてのノードからリターンを待ちレコードのタイムスタンプを比較しもっとも新しいレコードを返答



Read ONE

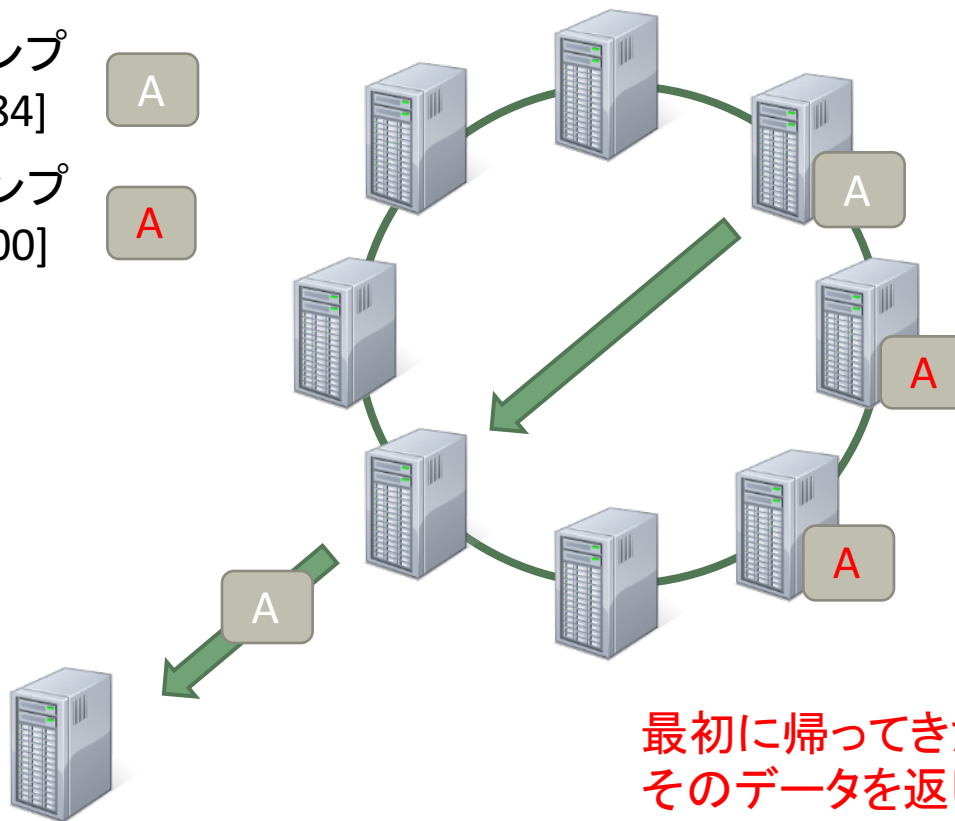


ONEの場合は全てのレプリカに問い合わせた後
最初に返答が帰ってきたノードのデータを返す

タイムスタンプ
[1367773384]



タイムスタンプ
[1367780000]



クライアント

最初に帰ってきたデータが古い場合
そのデータを返してしまう場合が存在する

Read TWO やその他

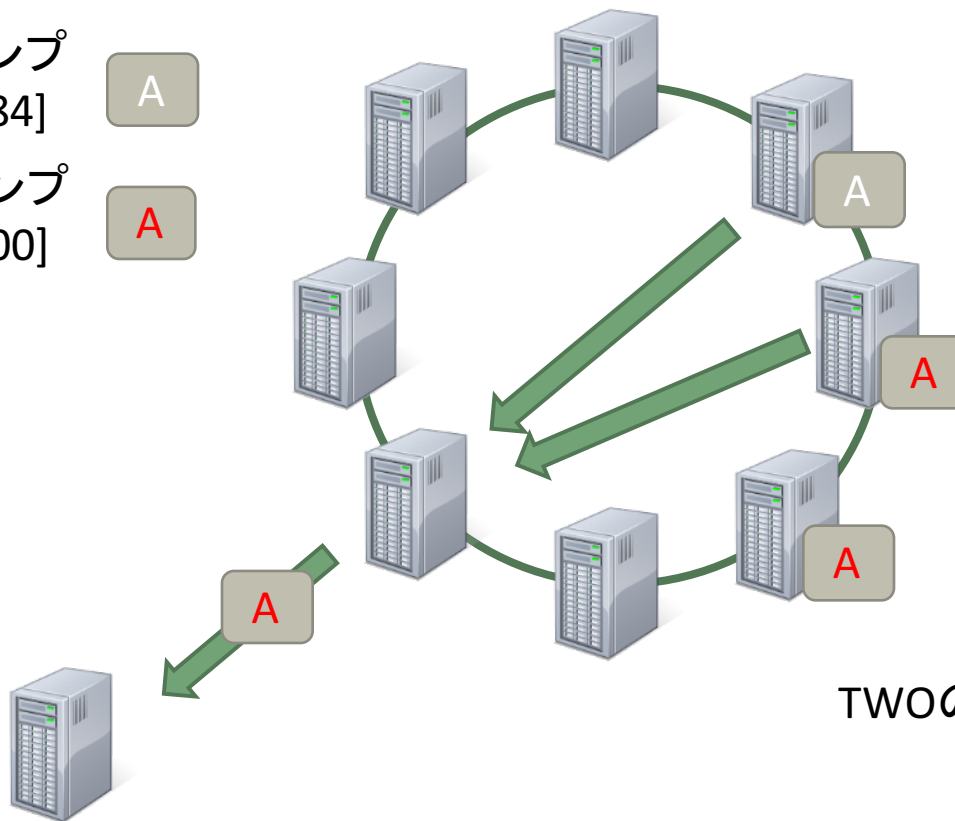


TWOやその他に関しては、対象のノード数(TWOであれば2ノード)
返答があれば、その中から新しいタイムスタンプのデータを返す

タイムスタンプ
[1367773384]



タイムスタンプ
[1367780000]



TWOの場合

クライアント



Readの場合ReadRepairが行われる

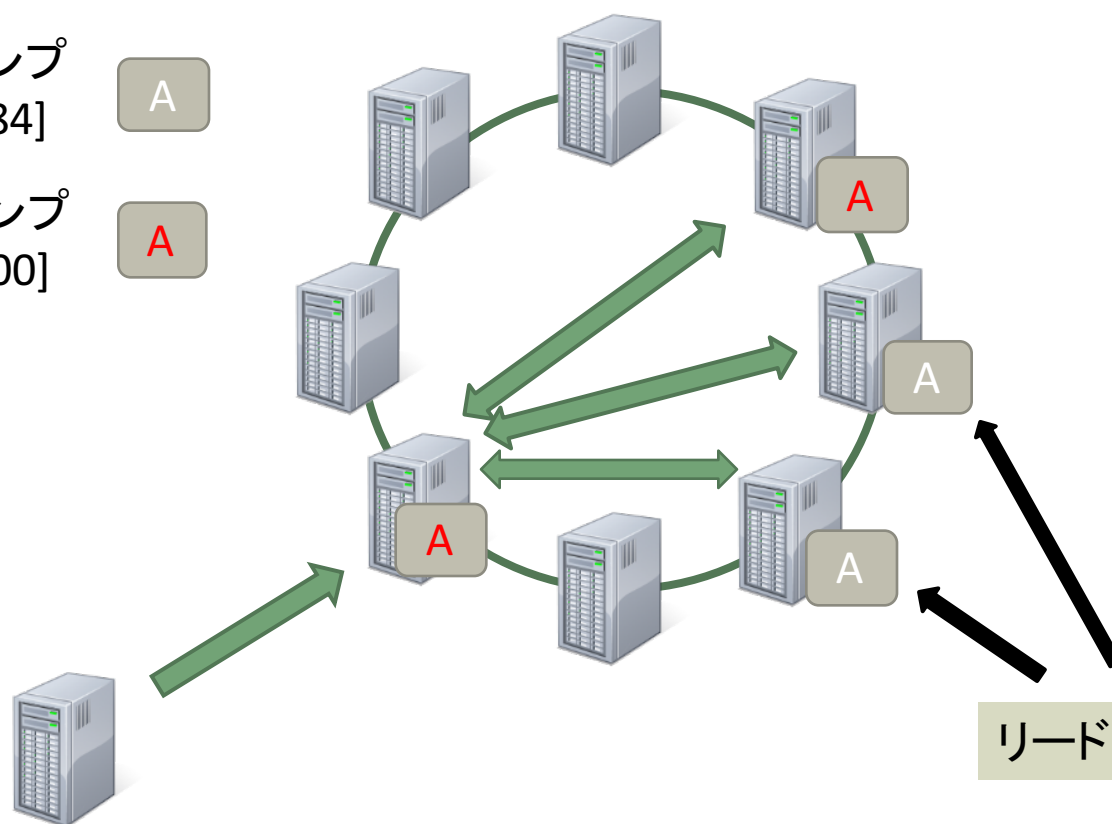
一度問い合わせたデータに古いデータが入っているレプリカがあれば
新しいタイムスタンプのデータに更新される

タイムスタンプ
[1367773384]

A

タイムスタンプ
[1367780000]

A



クライアント

リード後に A になる

環境構築



リングの確認

3つのノードでCassandraを無事起動できた後はringの状態を確認します
確認には bin/nodetool を使用します

```
$ nodetool status
```





Agenda

- NoSQLとは
- CassandraとはどのようなDB?
- Cassandraクラスタ構築
- IoTデータとは
- Cassandraの普遍性
- Sparkの力



IoTデータとは



IoTとM2M

Internet of Things

一意に識別可能な「もの」がインターネット/クラウドに接続され、情報交換することにより相互に制御する仕組みである

Machine to Machine

マシンツーマシン (Machine-to-Machine) とは、コンピュータネットワークに繋がれた機械同士が人間を介在せずに相互に情報交換し、自動的に最適な制御が行われるシステムを指す。



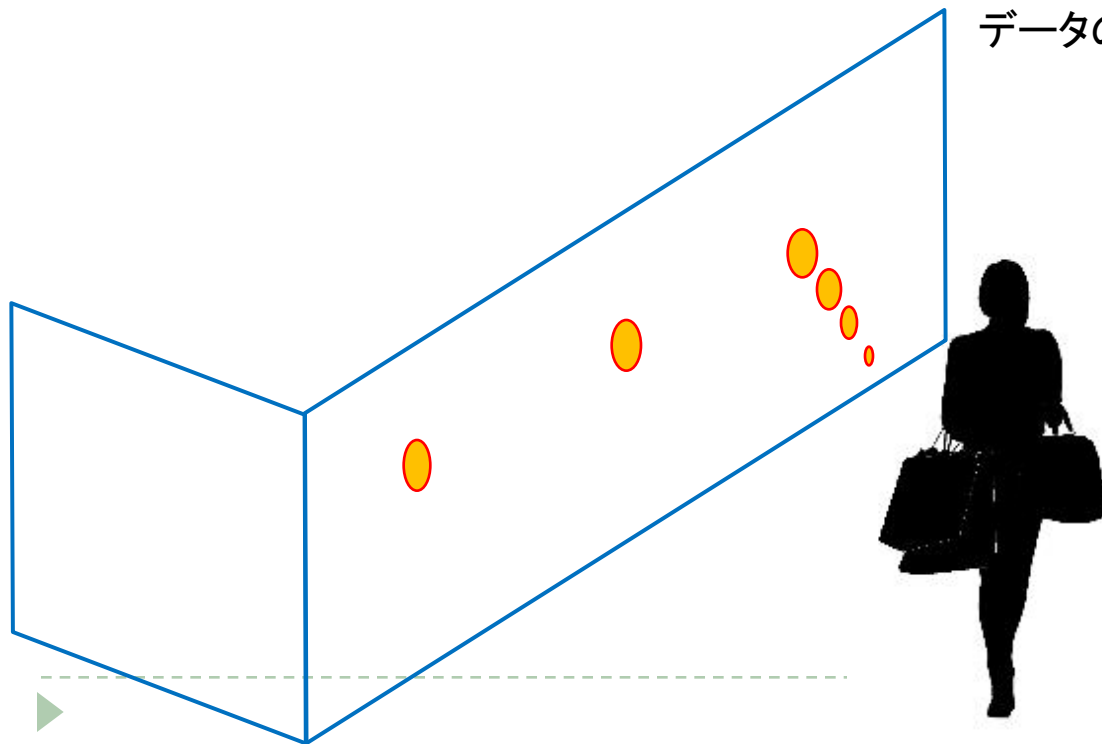
IoTデータとは



例1)

iBeacon

BLE(Bluetooth low Energy)を使える端末(スマートフォンなど)を発信器として、その端末の持つエリアへの入出検知や距離計測を行える仕組み(東京駅構内ナビなど)



データの性質: 人数 × 回数 × 拠点数 × 時間

1時間に1000人の人が2回通過
ID等: 100b
拠点数: 50箇所

増加率 1mb/h
24mb/d
720mb/m
4. 32gb/hm

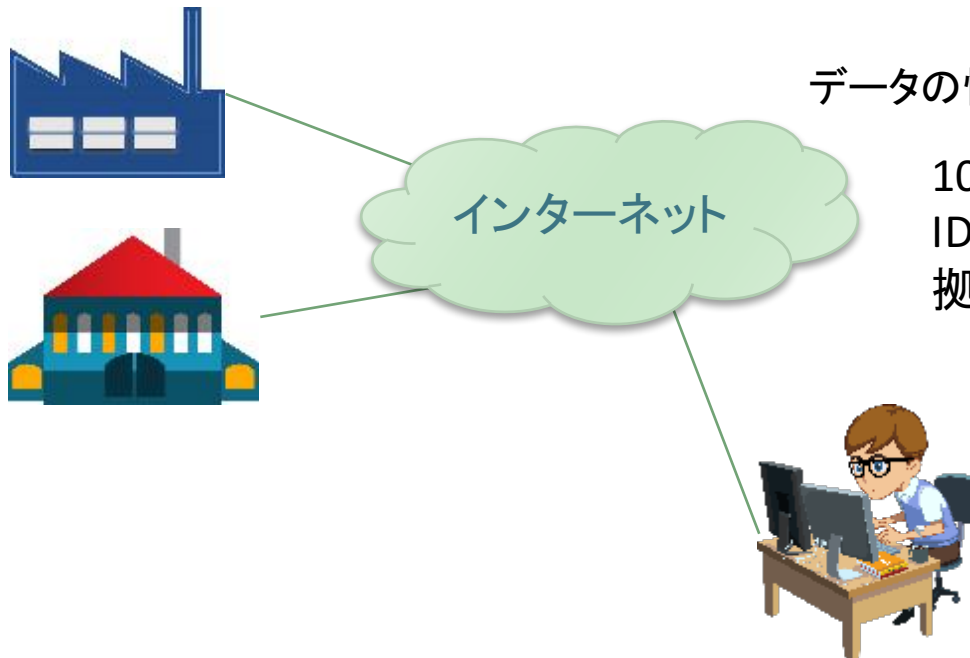
IoTデータとは



例2)

Industrial Internet

産業用機器とITの融合に関するコンセプト。
高機能の機器、低コストのセンサー、インターネット、ビッグデータ収集・分析技術などを組み合わせ故障率検地など機器の効率的な運用を行う。



データの性質: センサー数 × 回数 × 拠点数 × 時間

10秒間に10000個のセンサーが1回通信

ID等: 100b

拠点数: 3箇所

増加率 6mb/h

144mb/d

864mb/m

5. 184gb/hm



Agenda

- NoSQLとは
- CassandraとはどのようなDB?
- Cassandraクラスタ構築
- IoTデータとは
- **Cassandraの普遍性**
- Sparkの力

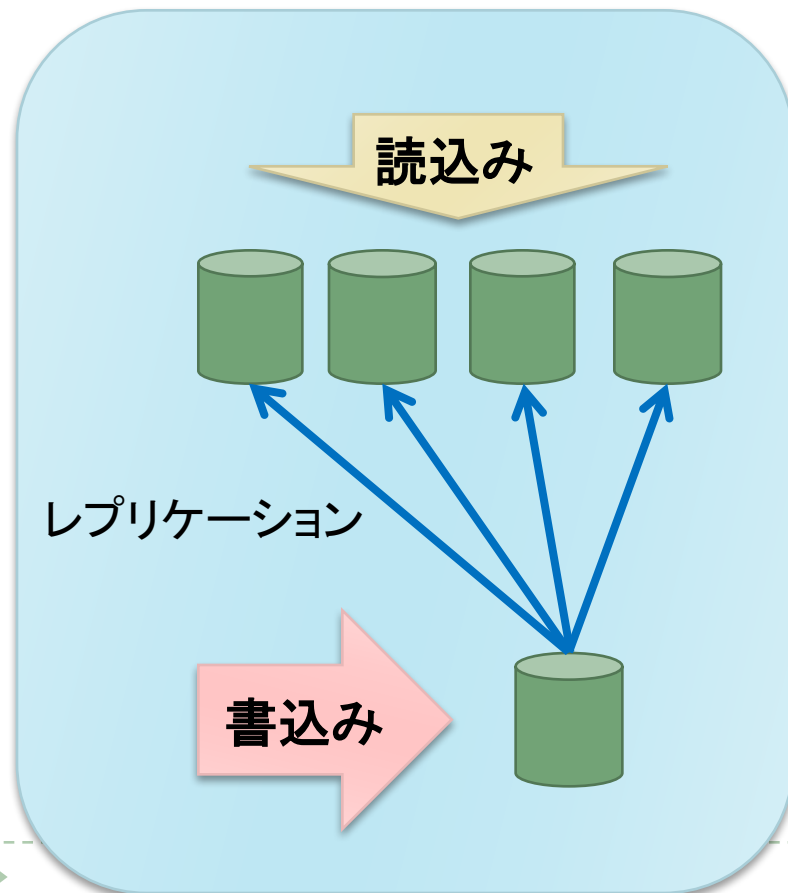


Cassandraの普遍性



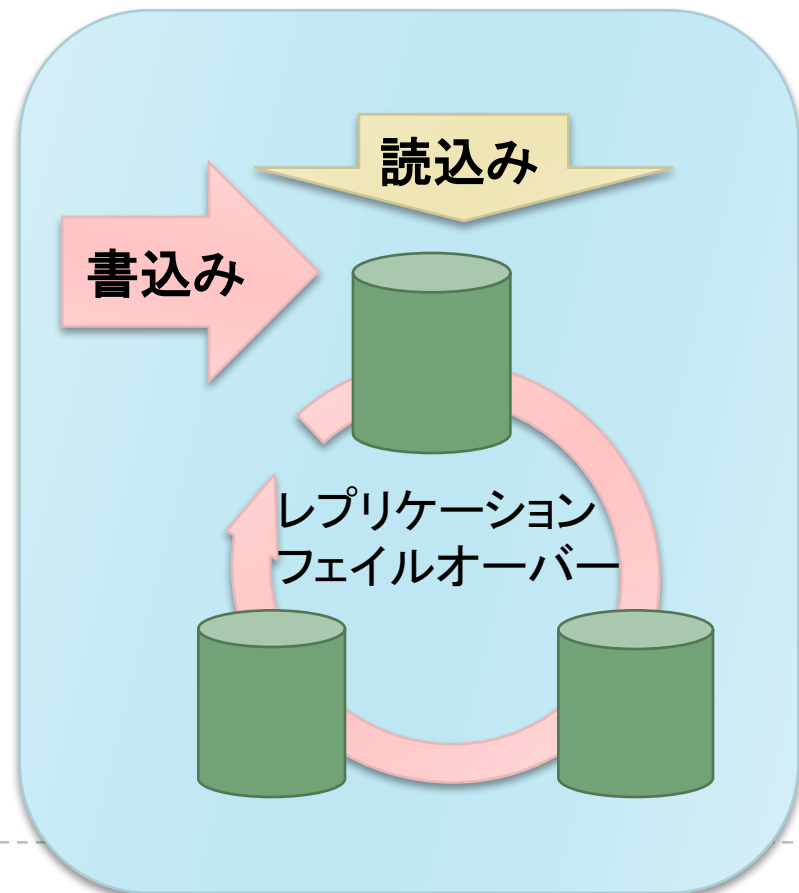
一般的なMySQLレプリケーションの構造

単一書込み・複数読込み



一般的なSQL Serverクラスタリングの構造

単一書込み・単一読込み

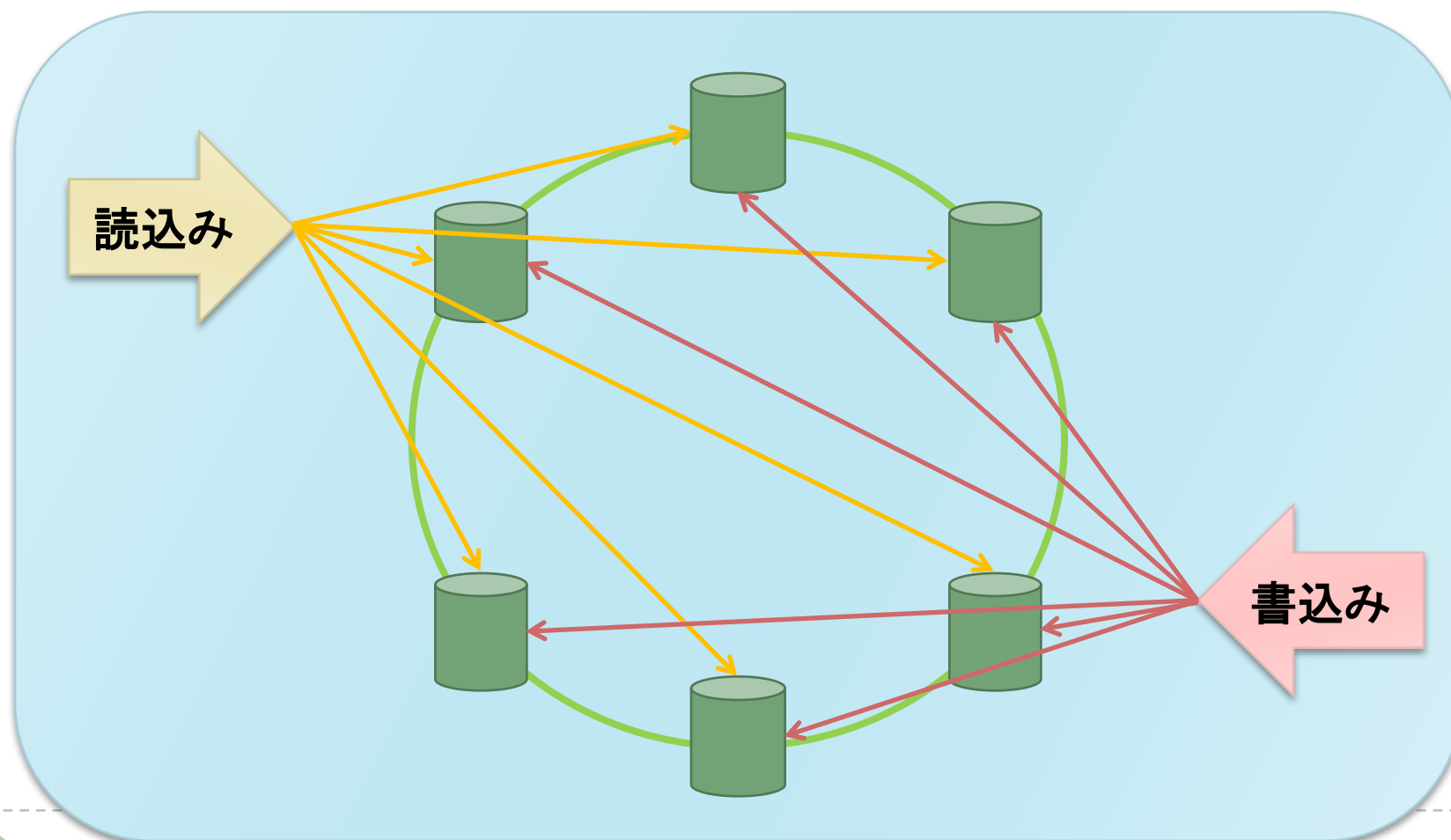




Cassandraの普遍性

Cassandraの構造

複数書込み・複数読込み



Cassandraの普遍性



IoT向けデータベースとしてのCassandraの特徴

- 書込みに強い。
 - 書込み先が分散化されているので同時多数書込みに強い（秒間100万書込み等）
 - 結果整合性による柔軟な書込み精度を選択可能
- 解析ツールとの親和性
 - 多彩なドライバ（ODBC、JDBC、PHP、Ruby、Perl等）
 - Apache Hadoop、Apache Spark、Presto等の多彩な解析ツールを利用可能
- マルチベンダー
 - Windows、Linux、各種クラウド、JVMが稼働すれば使用する事が出来ます。Windowsでの採用実績もあります。





Agenda

- NoSQLとは
- CassandraとはどのようなDB?
- Cassandraクラスタ構築
- IoTデータとは
- Cassandraの普遍性
- **Sparkの力**

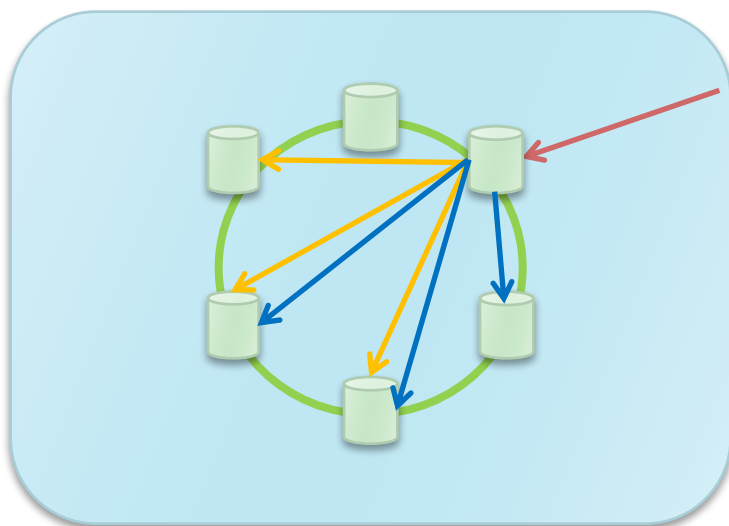


Sparkの力



Cassandraは横断検索が苦手

RowKeyベースのConsistent Hashingの為、連続したKeyが同じノードに存在するとは限らないので連続した領域のデータ取得はCassandra単体では苦手



データ解析は総当たりの逐次処理

Sparkの力



Sparkとは

高速なデータ分析のための新たな手段

Sparkの特徴

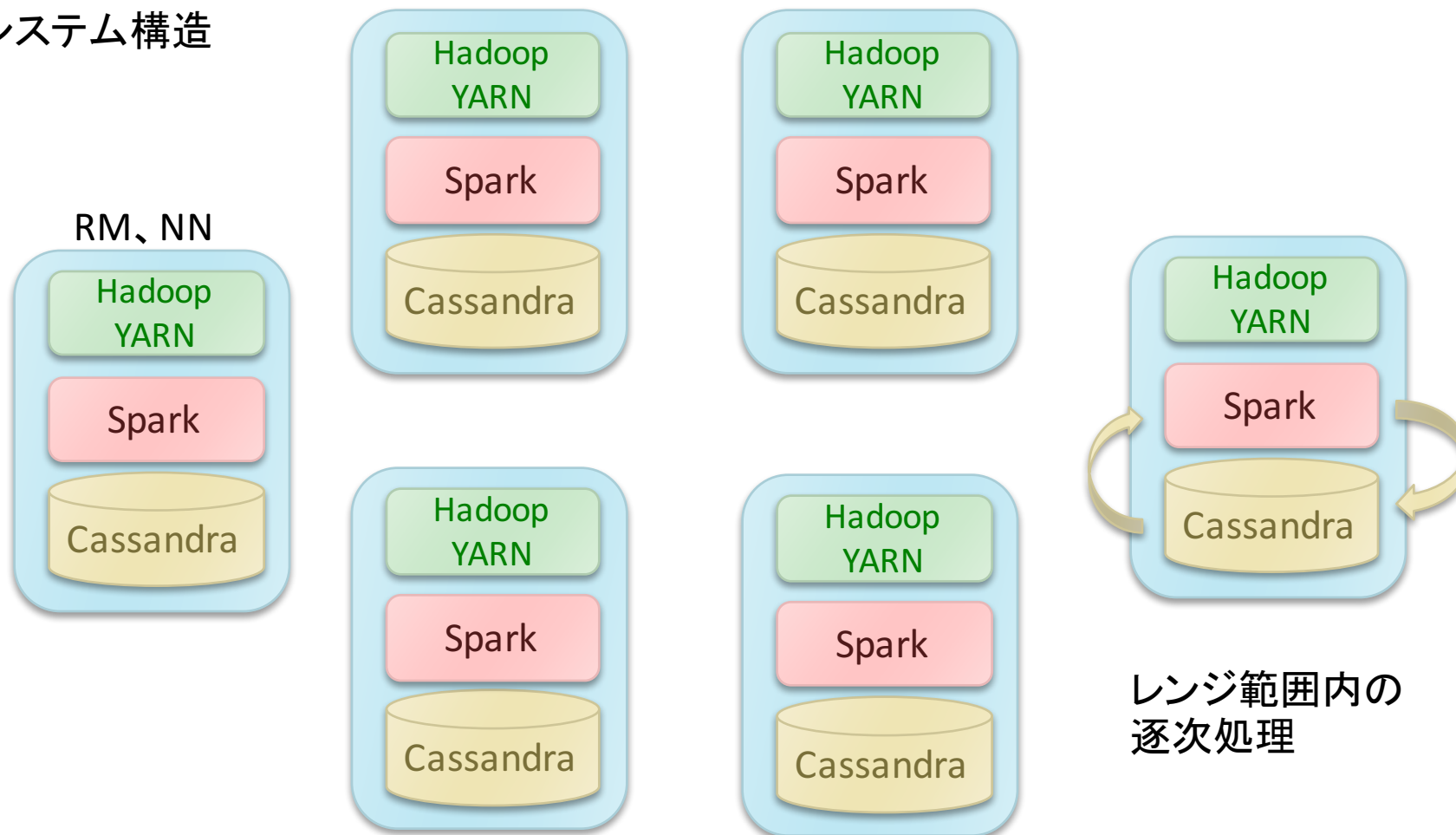
- ◇ RDD (Resilient Distributed Dataset・弾性分散データセット)
 - ◇ 不変 (イミュータブル)
 - ◇ 分割・分散配置
 - ◇ インメモリー
 - ◇ 遅延評価
- ◇ Hadoop連携
 - ◇ HDFS自動連携
 - ◇ YARN連携



Sparkの力



システム構造



Sparkの力



アプリケーション

Scala、Java (7,8)、Pythonでアプリケーションを作成可能。
データ解析に適したLibraryを適時利用することにより迅速に業務アプリケーションを作成可能。



Sparkの力



Sparkのライブラリ

- **GraphX**

- グラフとグラフ並列計算API

- **MLlib**

機械学習アルゴリズムAPI

Spark Streaming

データの逐次時系列処理

- **Spark SQL**

- SQLライクなQuery言語

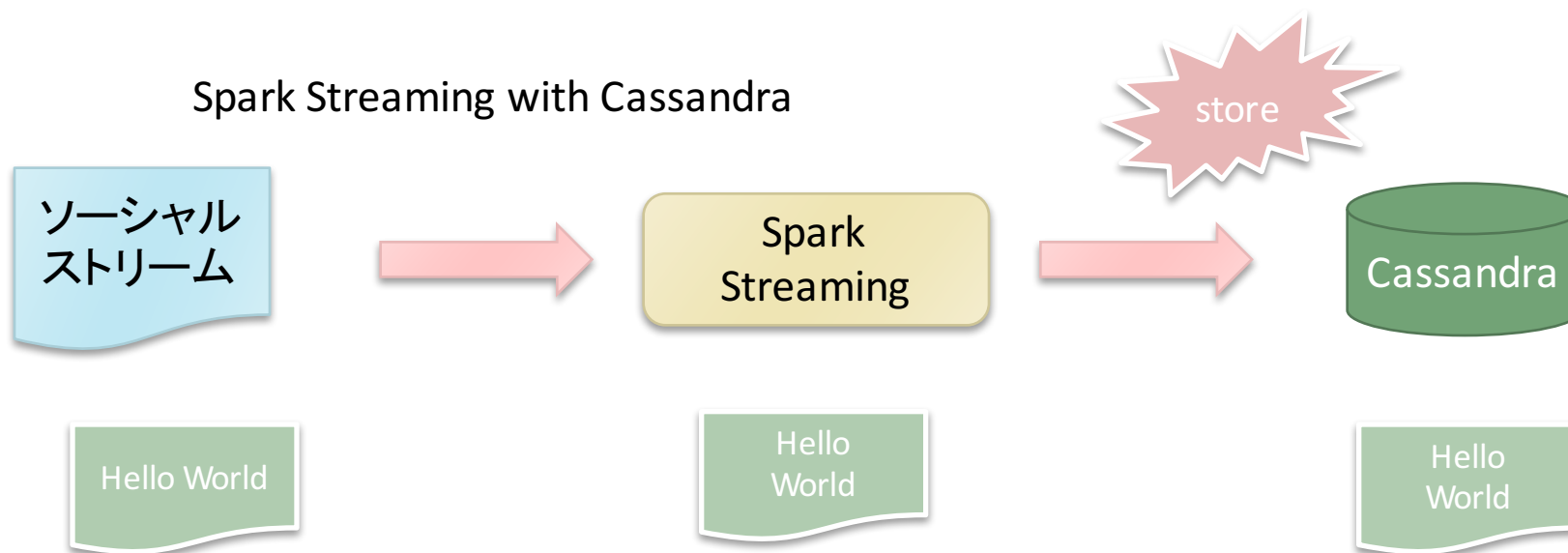


Sparkの力



- Spark Streaming
データの逐次時系列処理

Spark Streaming with Cassandra



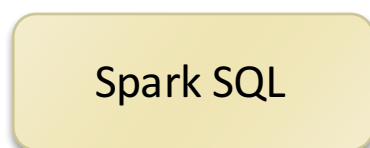
※短時間のShortBatchを逐次実行可能。

Sparkの力



- Spark SQL
SQLライクなDSL言語

Spark SQL with Cassandra



```
var rdd = cc.sql("SELECT * from test2.words a join  
test2.phrase b on a.word = b.phrase")
```



Sparkの力



■インタラクティブ

spark-shell

Spark向けScala用のインタラクティブシェル。Scalaでその場でロジックを実行可能
SparkSQLも実行可能、インタラクティブなデータ問い合わせが可能。

```
kazutaka ~ tomila@macosx: ~ - ssh - 104x37
ter reached minRegisteredResourcesRatio: 0.8
15/03/13 14:43:30 INFO SparkILoop: Created spark context..
Spark context available as sc.

scala> 15/03/13 14:43:31 INFO BlockManagerMasterActor: Registering block manager parphylo.intneforest.c
o.jp:58115 with 530.3 MB RAM, BlockManagerId(1, parphylo.intneforest.co.jp, 58115)
exit
warning: there were 1 deprecation warning(s); re-run with -deprecation for details
hadoopuser@p:~$ /opt/spark/bin/spark-shell --master yarn-client
Spark assembly has been built with Hive, including Hadoop jars on classpath
15/03/13 14:43:40 INFO SecurityManager: Changing view acls to: hadoopuser
15/03/13 14:43:40 INFO SecurityManager: Changing modify acls to: hadoopuser
15/03/13 14:43:40 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; use
rs with view permissions: Set(hadoopuser); users with modify permissions: Set(hadoopuser)
15/03/13 14:43:40 INFO HttpServer: Starting HTTP Server
15/03/13 14:43:41 INFO Utils: Successfully started service 'HTTP class server' on port 56380.
Welcome to

      ____
     /___ \
    /___ \
   /___ \
  /___ \
 /___ \
/_/___ \
      \_/_/

version 1.2.1

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_75)
Type in expressions to have them evaluated.
Type :help for more information.
15/03/13 14:43:46 WARN SparkConf:
SPARK_CLASSPATH was detected (set to /home/works/build/protobuf-2.5.0/java/target/protobuf-java-2.5.0.jar
or /home/hadoop/bin/spark-cassandra-connector-java-assembly-1.2.0-SNAPSHOT.jar).
This is deprecated in Spark 1.2+.

Please instead use:
- ./spark-submit with --driver-class-path to augment the driver classpath
- spark.executor.extraClassPath to augment the executor classpath

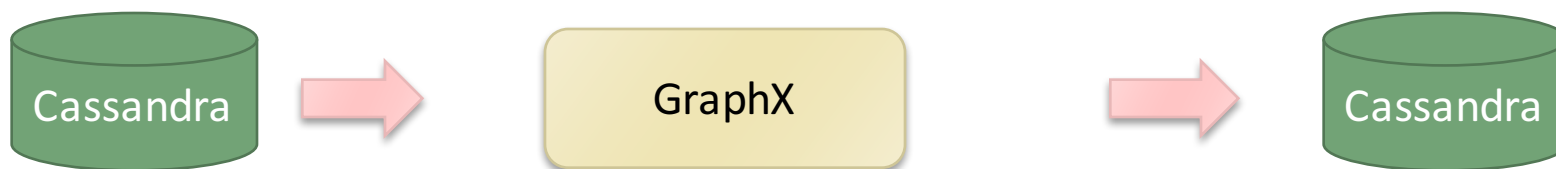
15/03/13 14:43:46 WARN SparkConf: Setting 'spark.executor.extraClassPath' to /home/works/build/protobuf-2.5.0/java/target/protobuf-java-2.5.0.jar:/home/hadoop/bin/spark-cassandra-connector-java-assembly-1.2.
```

Sparkの力



- GraphX
グラフとグラフ並列計算API

ソーシャルグラフ解析・テキスト解析など



GraphX with Cassandra



Sparkの力



- MLLib
 - 機械学習アルゴリズムAPI

最新ではアルゴリズムの数が格段に増えました。

MLLib with Cassandra

大量にためる→機械学習

この組合せに最適

Tweet 3000万件のClusteringなど

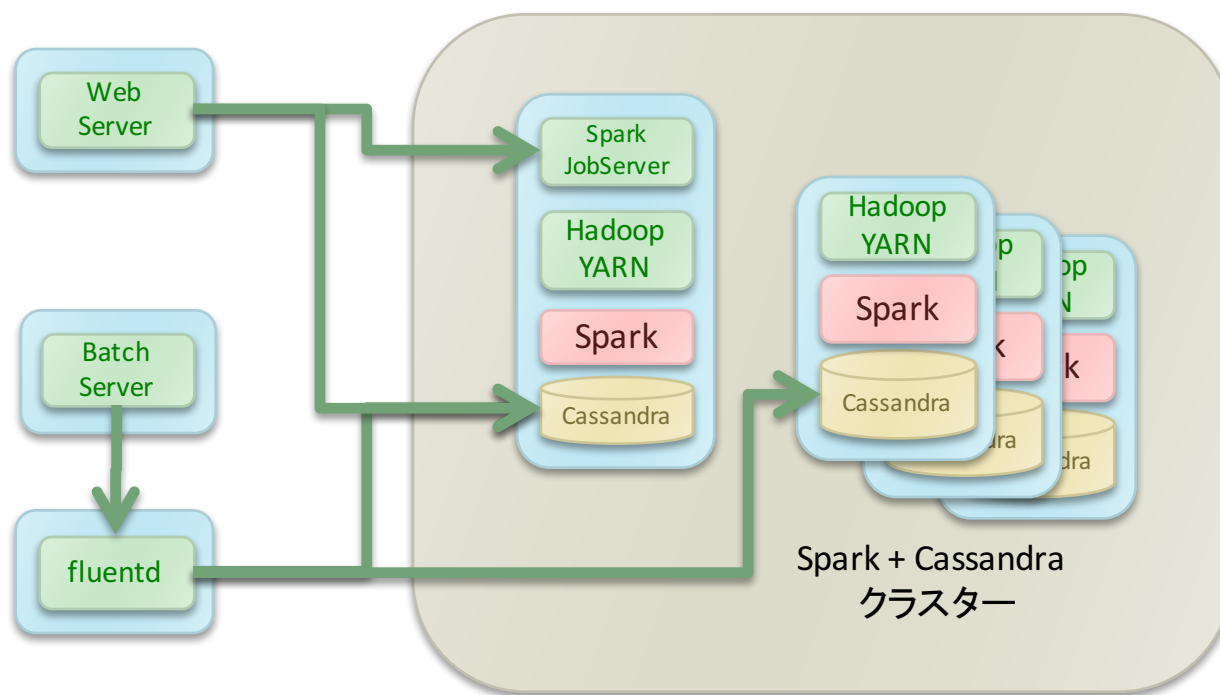
Data types
Basic statistics
 summary statistics
 correlations
 stratified sampling
 hypothesis testing
 random data generation
Classification and regression
 linear models (SVMs, logistic regression, linear regression)
 naive Bayes
 decision trees
 ensembles of trees (Random Forests and Gradient-Boosted Trees)
 isotonic regression
Collaborative filtering
 alternating least squares (ALS)
Clustering
 k-means
 Gaussian mixture
 power iteration clustering (PIC)
 latent Dirichlet allocation (LDA)
 streaming k-means
Dimensionality reduction
 singular value decomposition (SVD)
 principal component analysis (PCA)
Feature extraction and transformation
Frequent pattern mining
 FP-growth
Optimization (developer)
 stochastic gradient descent
 limited-memory BFGS (L-BFGS)



Sparkの力



実際のシステム構成



まとめ



➤ NoSQLとは

NoSQLとはRDBMS以外のデータベース管理システムの総称
柔軟なデータ構造の格納や分散型システムの構築など
それぞれ特徴的な機能を持つ

➤ CassandraとはどのようなDB?

オープンソースでありスケーラブルなNoSQLデータベース
完全に分散され、単一障害点がない



まとめ



➤ Cassandra クラスタ構築

大量データの負荷分散や冗長化を実現するため、
クラスタ構成を構築することが必要。

➤ IoT データとは

➤ 一意に識別可能な「もの」がインターネット/クラウドに接続され、情報交換することにより相互に制御する仕組みである

よって、IoT データはデータ量が多くなる



まとめ



➤ Cassandraの普遍性

Cassandraは大規模データの管理に親和性がある

Cassandraはデータストレージなので機能そのものはそれほど多くない

➤ Sparkの力

➤ SparkはCassandraの足りないところを上手に補ってくれる。

