学んで動かす！Sparkのキホン

NTTデータ 技術統括本部
OSSプロフェッショナルサービス

土橋 昌

NTT DATA

自己紹介

NTT DATA

土橋 昌 - Masaru Dobashi

OSSを徹底活用したシステム開発やR&Dに従事。エンジニア。
7、8年前にHadoopに出会い、1000台超えのHadoopのシステムの開発・運用などを担う。当時の課題感からStorm、Sparkの取り組みをはじめ現在に至る。
技術コンサルから現場開発、インフラからデータ処理、ゲテモノから定番まで、捻じ伏せてどうにかするのがお仕事です。

hadoop Spark STORM kafka 等々

Spark Summit

Strata Hadoop World

本日のおはなし

NTT DATA

Sparkのキホンを知って…

Sparkアプリの書き方を知って…

Sparkの中身を少しだけ知って…

大規模データ処理を少しでも身近に
感じて楽しんでいただければ嬉しいです

ちなみに…翔泳社「Apache Spark入門」

NTT DATA

ここで紹介する内容の**もっと詳しい版**が掲載されています。
Sparkの**動作原理**から標準ライブラリを使った**具体的なプログラム例**まで。

第1章：Apache Sparkとは
第2章：Sparkの処理モデル
第3章：Sparkの導入
第4章：Sparkアプリケーションの開発と実行
第5章：基本的なAPIを用いたプログラミング
第6章：構造化データセットを処理する - Spark SQL -
第7章：ストリームデータを処理する - Spark Streaming -
第8章：機械学習を行う - MLlib -
Appendix
A. GraphXによるグラフ処理
B. SparkRを使ってみる
C. 機械学習とストリーム処理の連携
D. Web UIの活用

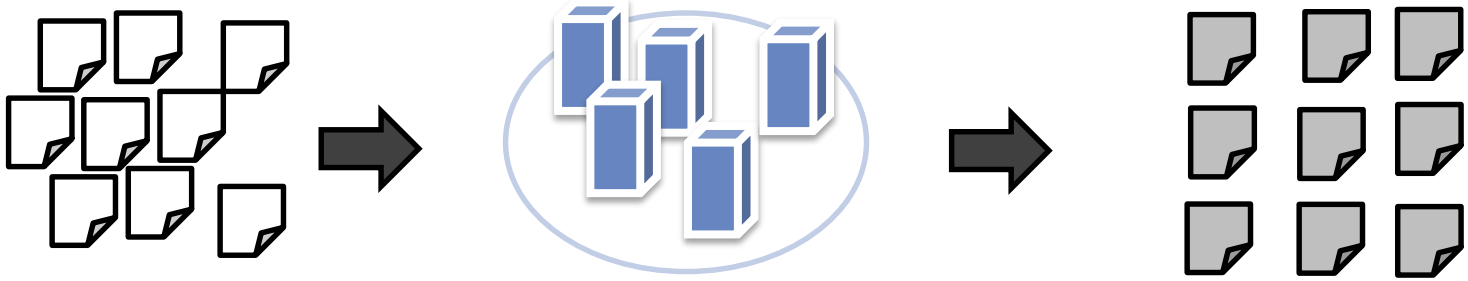
Spark1.5系
対応

まず初めにApache Sparkとは？

NTT DATA

ひとことで言うと…オープンソースの並列分散処理系

データ管理には向きませんが、**データ処理**には向いています。



大量のデータを

たくさんのサーバを並べて
並列分散処理し、

現実的な時間（数分～数時間）で
目的の処理結果を得る

並列分散処理の面倒な部分は処理系が解決してくれる！

- 障害時のリカバリ
- タスクの分割やスケジューリング
- etc

守備範囲の広い優等生ですが、あくまで分散処理系

NTT DATA

いろいろ出来るので、ついうっかり**分散処理でなくても良い処理**も
実装しようとしがちですが、それは**あまり筋が良くありません。**
（そういう場合は分散のための仕組みは不要なものとなります）

とはいえ、もちろん規模の小さなデータに対しても**動きます**。

現実的には実現したいことの全体の傾向によって
Sparkを使うか、他の手段を組み合わせるのかを
判断して用います。

では、うちではSpark使うべき？使わないべき？

NTT DATA

できれば

自分にとって嬉しいかどうかは、
PoCなどを通じてちゃんと確かめましょう。

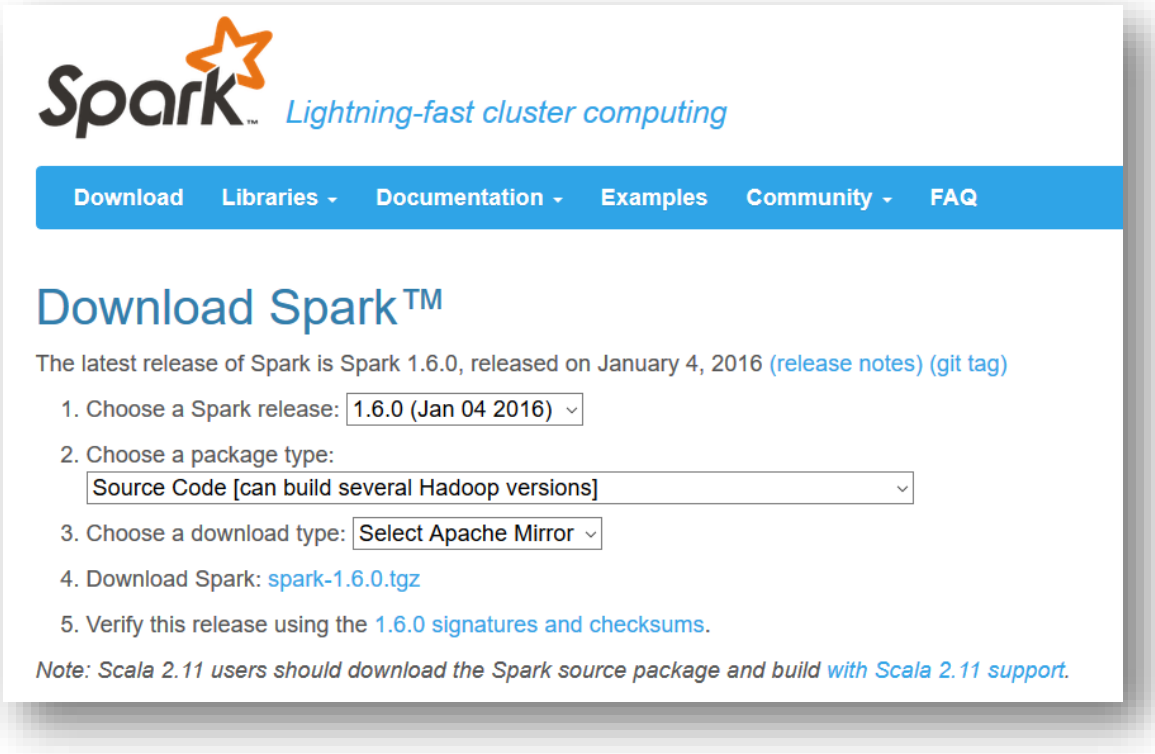
Sparkがあると嬉しい典型的なケース例

すでにHadoopを利用している
1台のマシンでは収まらない量のデータがある
データ件数が多くて既存の仕組みだと一括処理が辛い
SQLもいいが、他の言語の内部DSLとして実装したい
集計を中心としたストリーム処理をやりたい
大規模データに対して定番の機械学習アルゴリズムを適用したい

とはいえ、試すのであれば簡単！

NTT DATA

Demo



JDKのインストールされた環境でパッケージを解凍して動かす
Spark体験の始まり

http://spark.apache.org/downloads.html

さて、ここでひとつ質問

NTT DATA

なぜSparkが生まれたのか？

さて、ここでひとつ質問

NTT DATA

オープンソースの並列分散処理系としては、
既に Hadoop が普及しているけど？

さて、ここでひとつ質問

NTT DATA

ものすごく簡単にHadoopの特徴をおさらいし、

Sparkが生まれた背景 を紐解きます

使ったことのない／使い始めのプロダクトに触れるときは重要

NTT DATA

Hadoop：コモディティなサーバを複数並べて分散処理

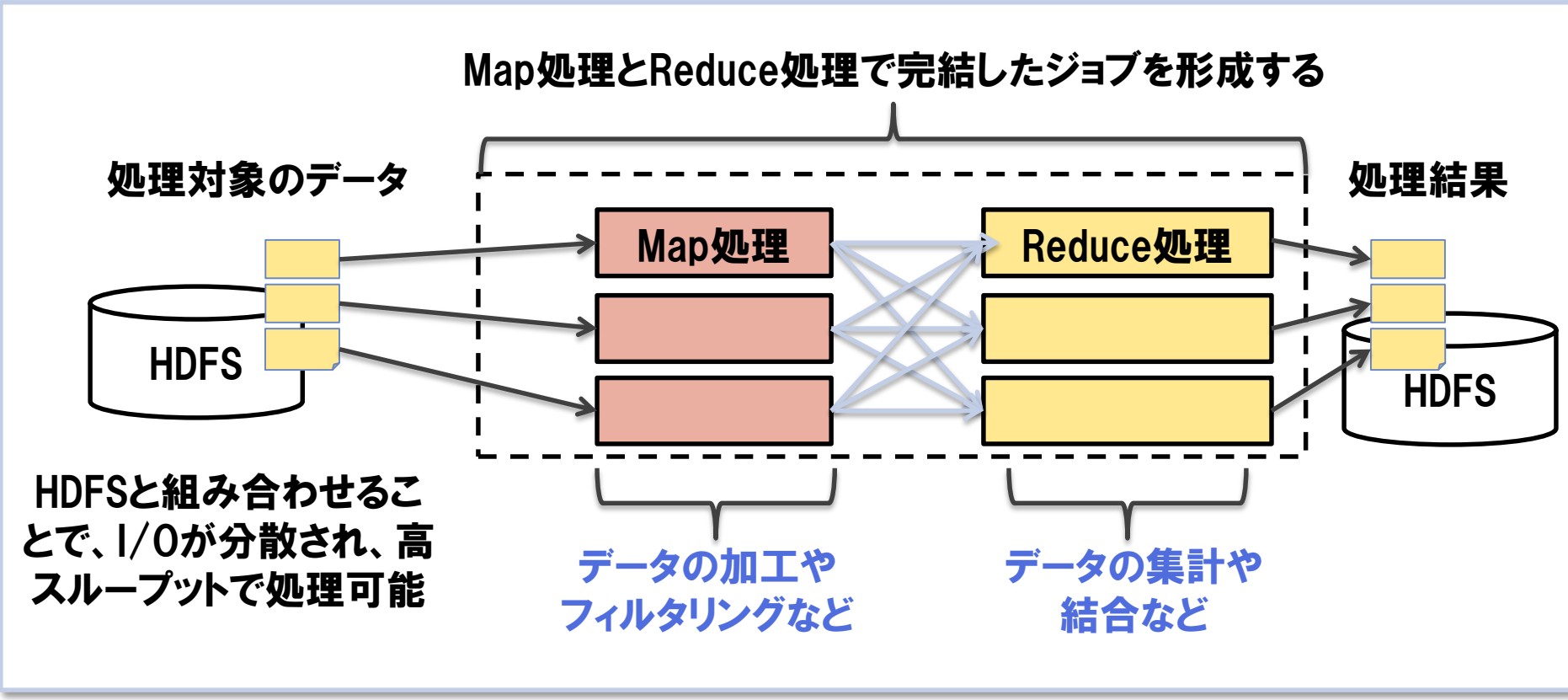
1. データを貯める
   HDFS
2. データ処理のリソースを管理する
   YARN
3. 処理する
   MapReduceフレームワーク

NTT DATA

Hadoop：コモディティなサーバを複数並べて分散処理

1. データを貯める
   HDFS
2. データ処理のリソースを管理する
   YARN
3. 処理する
   MapReduceフレームワーク

（補足）MapReduceフレームワークとは

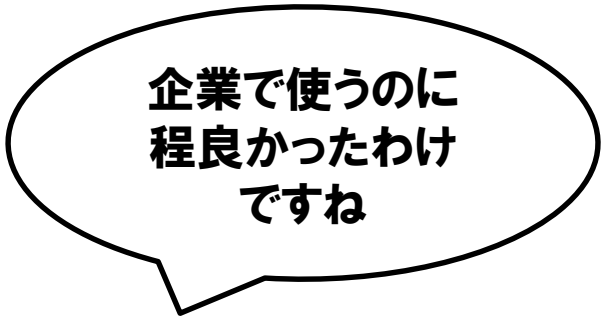


- アプリ開発者はMap処理とReduce処理を実装する（原則Java）
- 上記を元にフレームワークが分散処理する。障害発生時もリトライで処理が継続する。
- 基本的にサーバを増やしてスケールアウトすることで性能等を担保

注）増やせば増やすほど速くなる、というわけではない

MapReduceフレームワークの嬉しい点をざっくりと

NTT DATA



シンプルな処理モデル

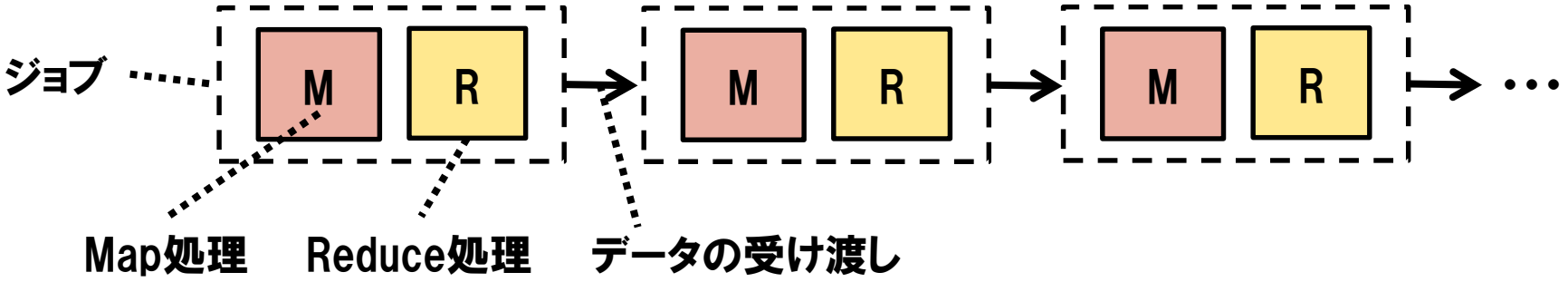
大量データでも動き切ってくれること

MapReduceフレームワークの課題

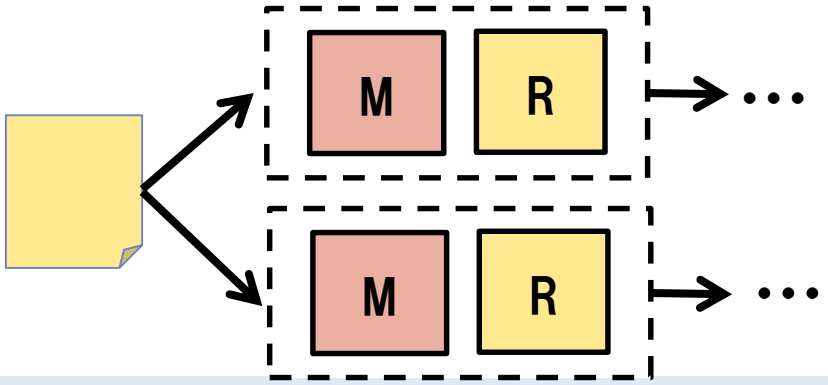
NTT DATA

とはいえ、色々と使っていくと…**処理効率が気になってきます**

- **ジョブが多段に構成される場合**



- **複数のジョブで何度も同じデータを利用する場合**



処理効率が気になる場面の例

NTT DATA

ジョブが多段に構成される場合

反復処理

- 機械学習
- グラフ処理

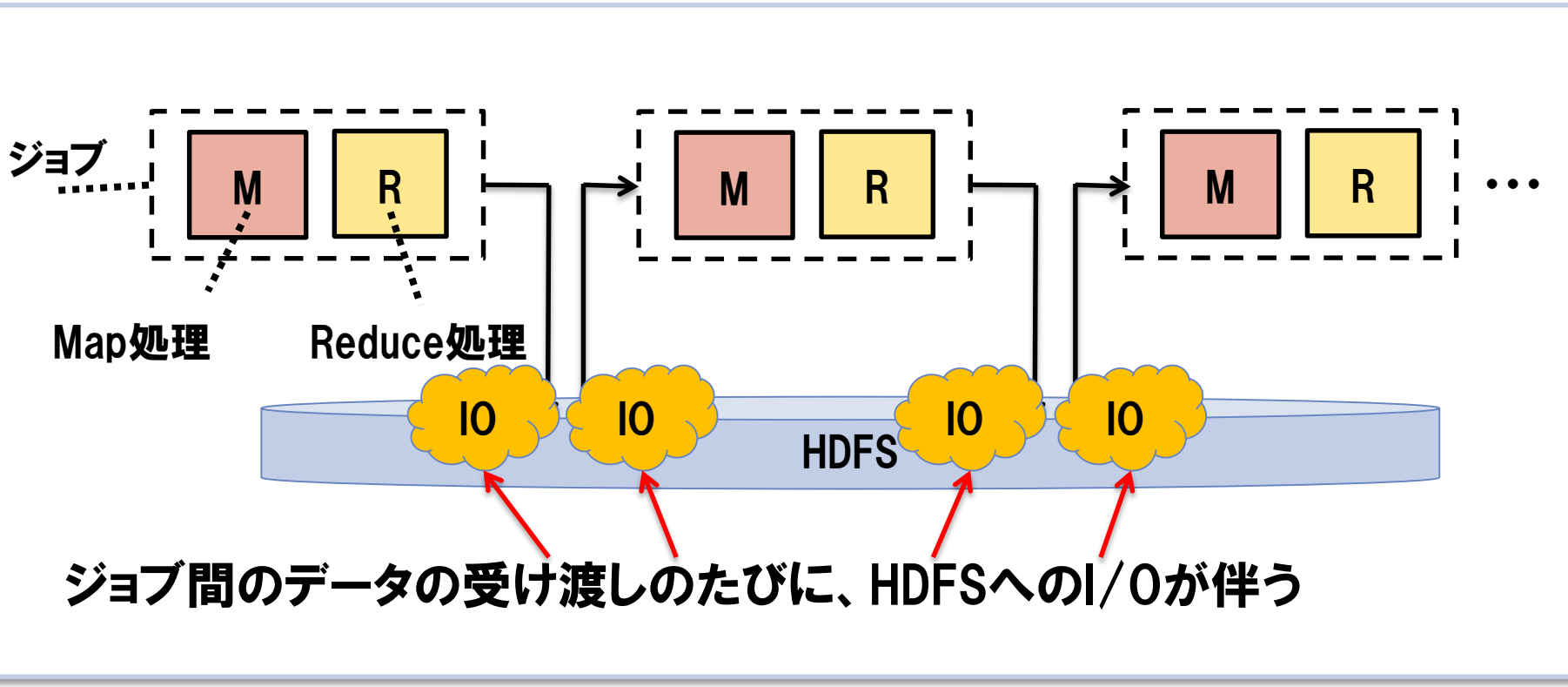
複雑な業務処理

複数のジョブで何度も同じデータを利用する場合

アドホックな分析処理

複雑な業務処理

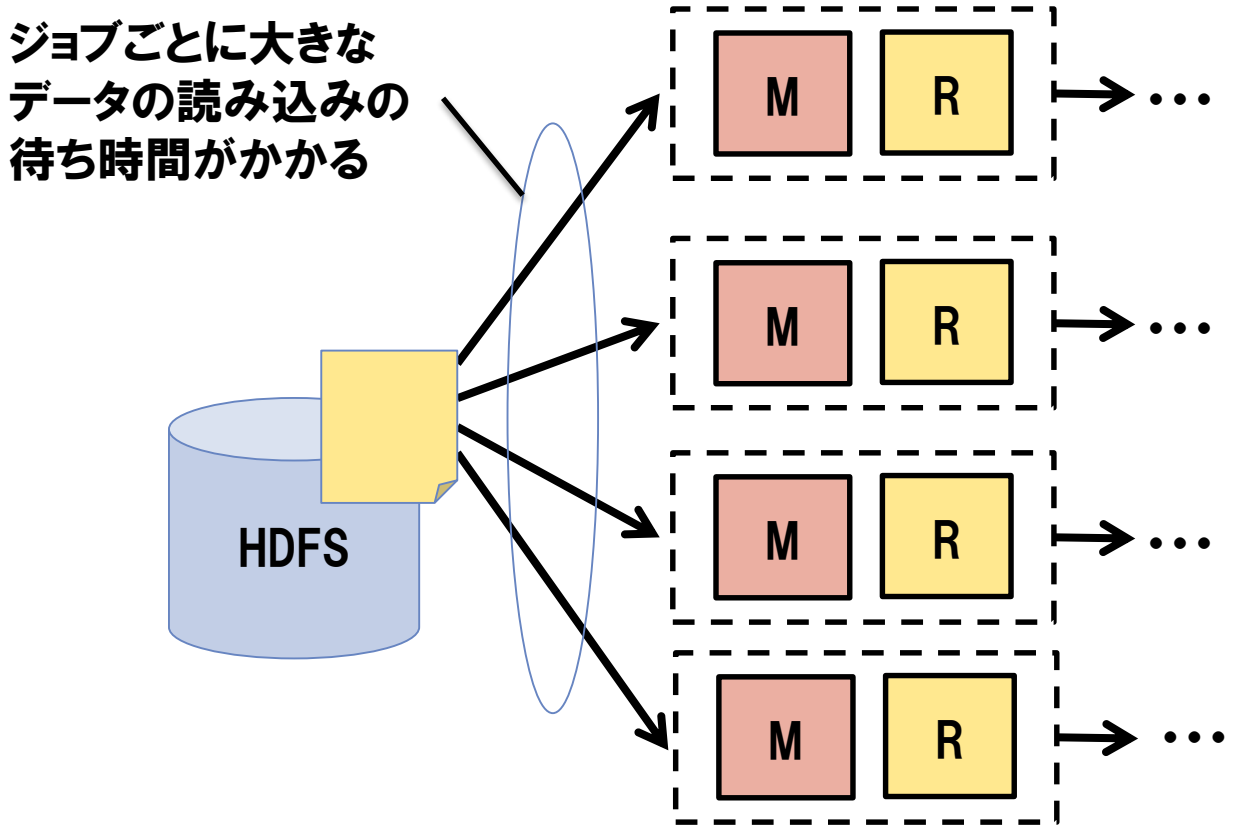
ジョブが多段に構成される場合の課題



- ジョブ間でのデータの受け渡しのために、都度HDFSへのI/Oが発生
- HDFSへの都度のI/Oのレイテンシが、処理時間の大部分を占めることにつながる

複数のジョブで何度も同じデータを利用する場合の課題

NTT DATA



何度も利用するデータを効率的に扱う仕組みがないため、同じデータを利用する場合に都度巨大なデータの読み出しのための待ち時間が発生する

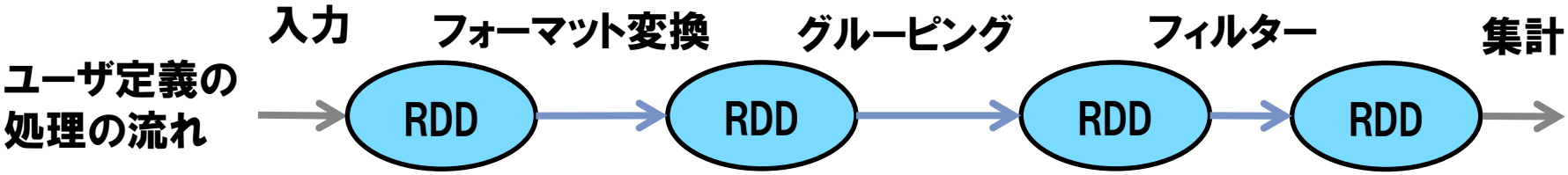
そこで生まれたSparkとは？

NTT DATA

抽象化を進めて、**スループットとレイテンシのバランス**を追求し、**使いやすいAPI**をユーザに提供。

処理モデルの基礎

RDDと呼ばれる**部分故障への耐性**を考慮した**分散コレクション**に対し、**典型的なデータ処理を繰り返して**目的の結果を得る



最新安定バージョンは1.6.1。2.0.0はプレビュー版が公開されました

（補足）Spark＝インメモリ？

「in-memory computing」というキーワードが主要開発者Mateiの論文で登場

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica

University of California, Berkeley

Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory

Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica

University of California, Berkeley

Abstract

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applica-

MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty.

- **Interactive analytics**: Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL interfaces such as Pig [21] and Hive [1]. Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with Hadoop, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk.

This paper presents a new cluster computing frame-

Spark: Cluster Computing with Working Sets
Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

元々は、ふたつのユースケース＝繰り返し計算、インタラクティブ計算を対象
そのために使いやすいRDD（Resilient Distributed Datasets）を提案

大規模データのための抽象概念

2016年現在、インメモリというのはひとつの要素。
本質はありとあらゆる最適化＋柔軟性。

Sparkのプログラミングモデルの基礎

NTT DATA

RDDに対する処理は、コレクション操作のように記述

```
rdd.filter(...).map(...).reduceByKey(...).saveAsText(...)
```

フィルタして　加工して　集計して　結果を保存

Scala / Java / Python向けのAPIが提供されている

インタラクティブシェルが「試行錯誤」を加速する

都度のビルドが不要なため、ロジックの試作から効果の確認のサイクルを高速化できる

【DEMO】WordCountしてみよう

NTT DATA

適当なテキストファイルを読みこんで
適当に単語に区切って「単語数」をカウントします

【DEMO】WordCountしてみよう

NTT DATA

spark-shellを起動し、Hadoop界隈のHello WorldであるWordCount実行

まずは簡単に、tgzを展開した直下にあるCHANGES.txtで試してみる

```
val sparkHome = sys.env("SPARK_HOME")
val textFile = sc.textFile(sparkHome + "/CHANGES.txt")

val wordCounts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b)
=> a + b)
```

ちゃんとカウントできてるかな？数の多い上位10件を出力してみよう

```
wordCounts.sortBy(_._2,false).take(10)
res0: Array[(String, Int)] = Array(("",61509), (Commit:,6830), (-0700,3672), (-0800,2162),
(in,1766), (to,1417), (for,1298), ([SQL],1277), (the,777), (and,663))
```

ノイズが多いな…英数字のみ含まれる文字列のみフィルタしてみよう

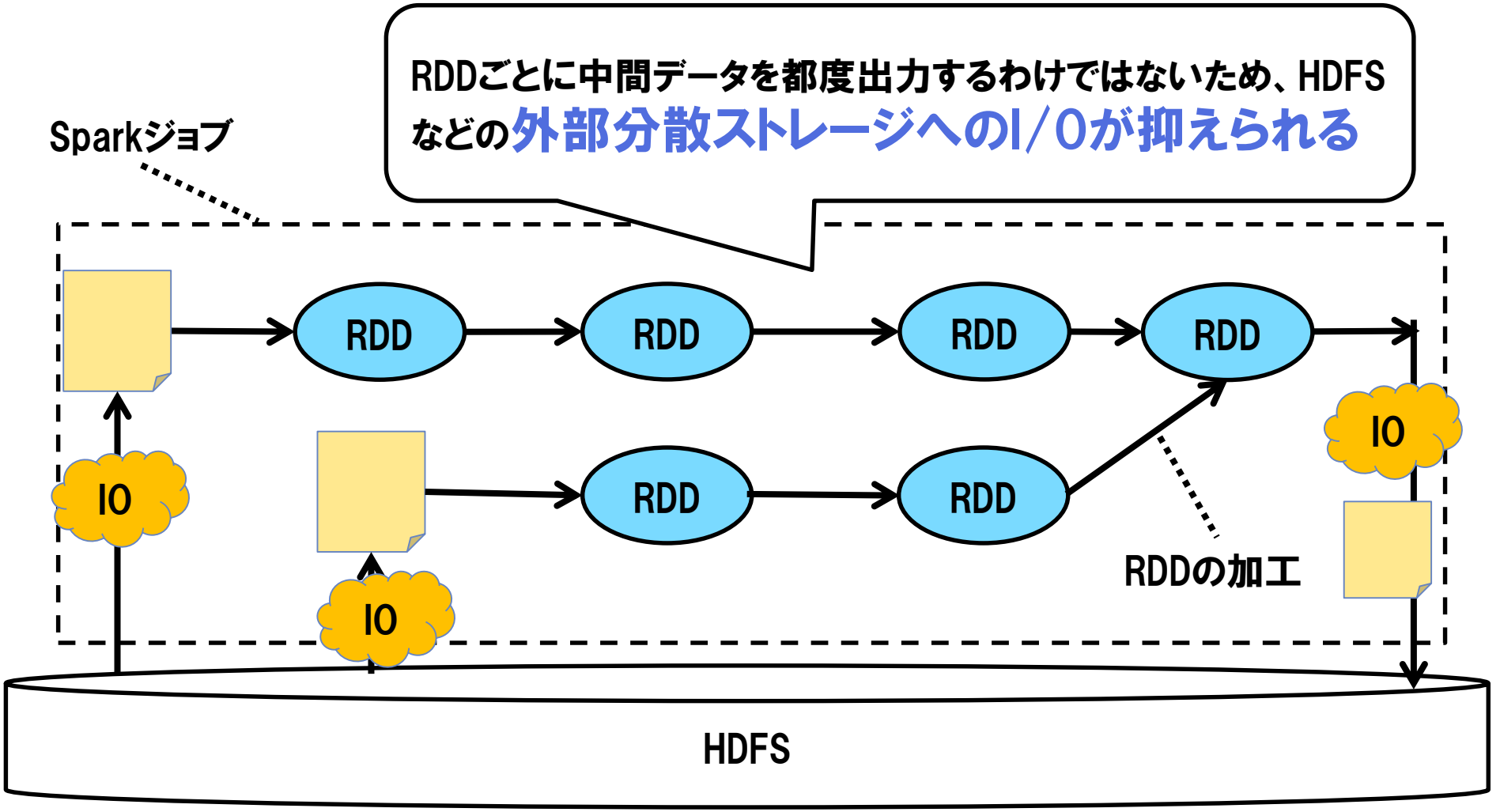
```
wordCounts.filter(_._1.matches("[a-zA-Z0-9]+")).sortBy(_._2,false).take(10)
res1: Array[(String, Int)] = Array((in,1766), (to,1417), (for,1298), (the,777), (and,663), (Add,631),
(of,630), (Fix,547), (Xin,491), (Reynold,490))
```

試行錯誤できる

高スループットと低レイテンシを両立する工夫

NTT DATA

複雑な処理を少ないジョブ数で実現できる



【DEMO】ウェブUIからジョブの構成を見てみます

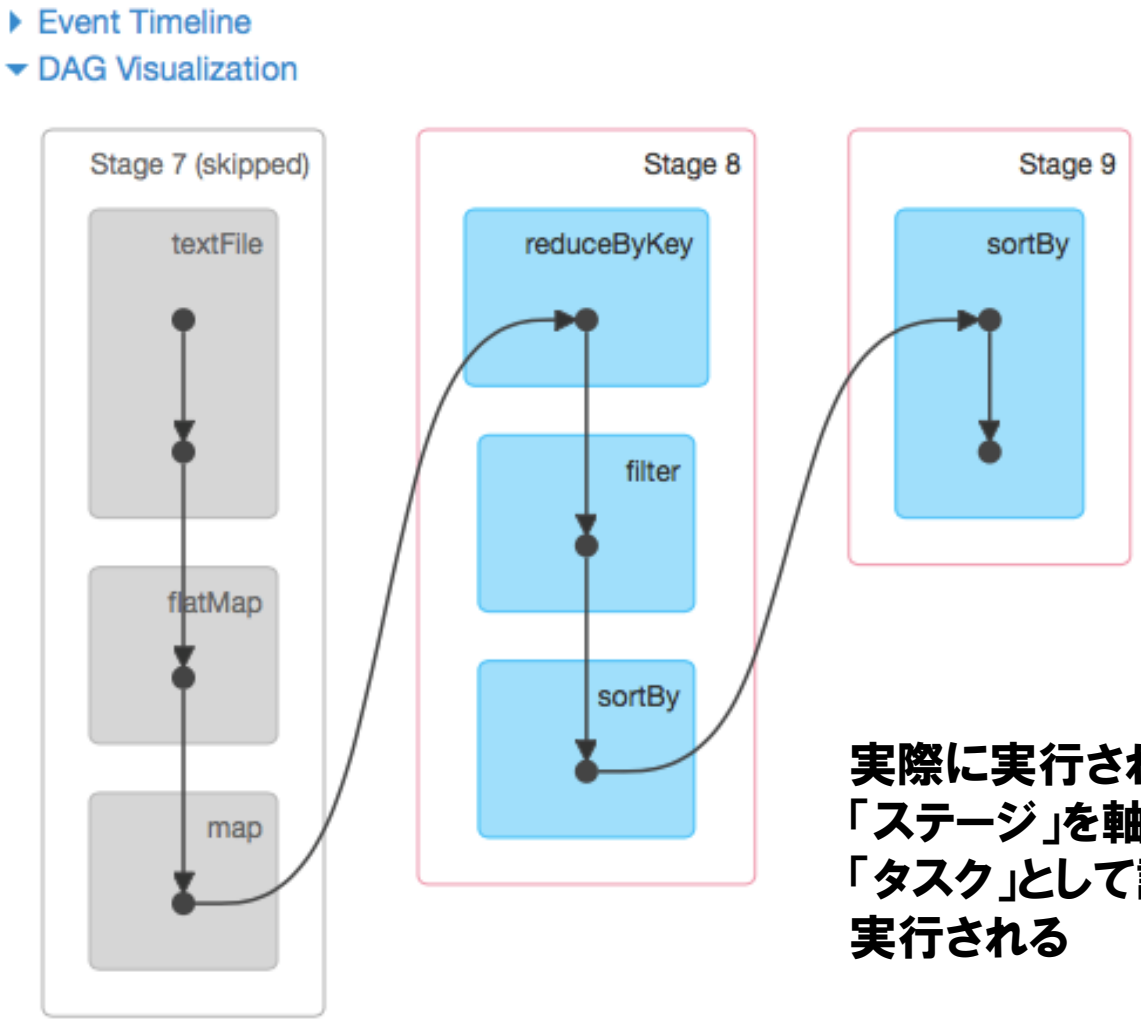
NTT DATA

ワードカウントで上位10件確認するときの
ジョブの構成(DAG)を確認しています。

【DEMO】ウェブUIからジョブの構成を確認

NTT DATA

WordCount (takeで上位10件確認まで) を実行するときのDAG

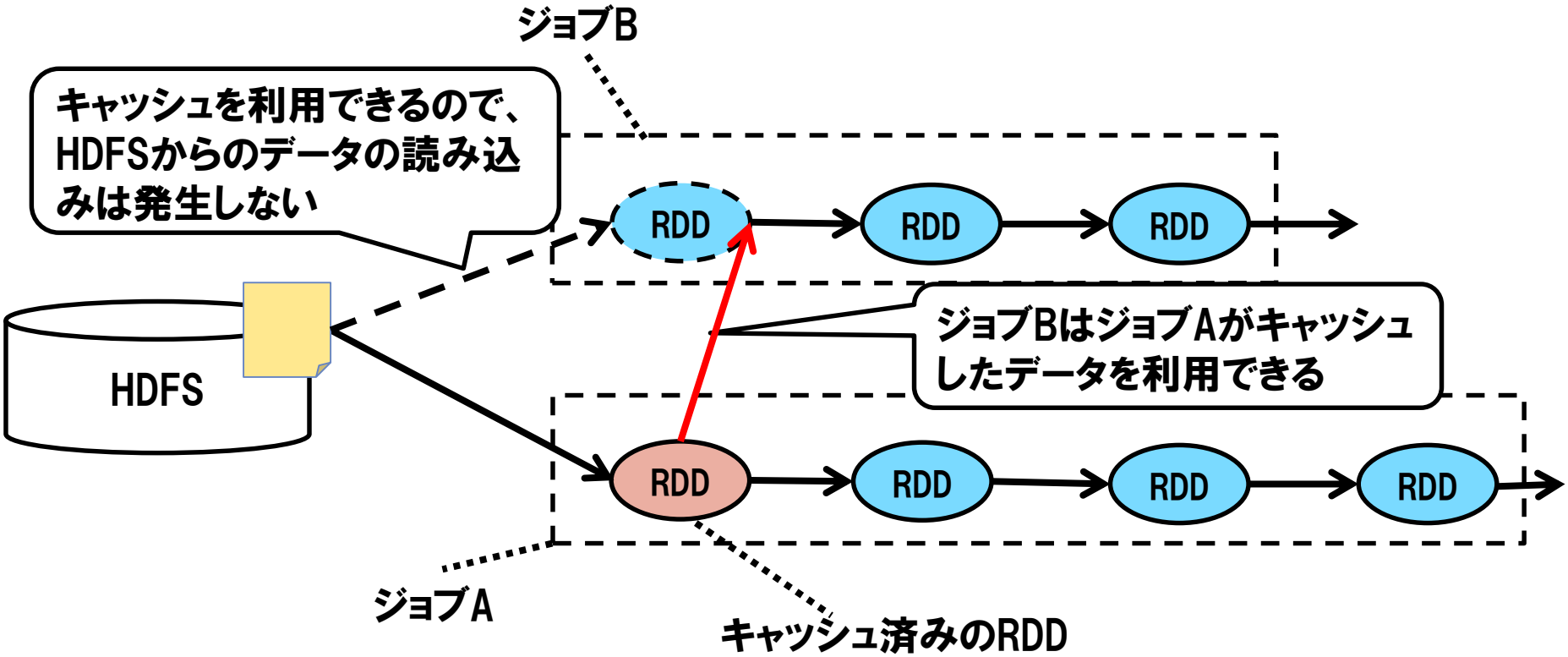


実際に実行される処理は
「ステージ」を軸に構成され、
「タスク」として計算マシンで
実行される

高スループットと低レイテンシを両立する工夫

NTT DATA

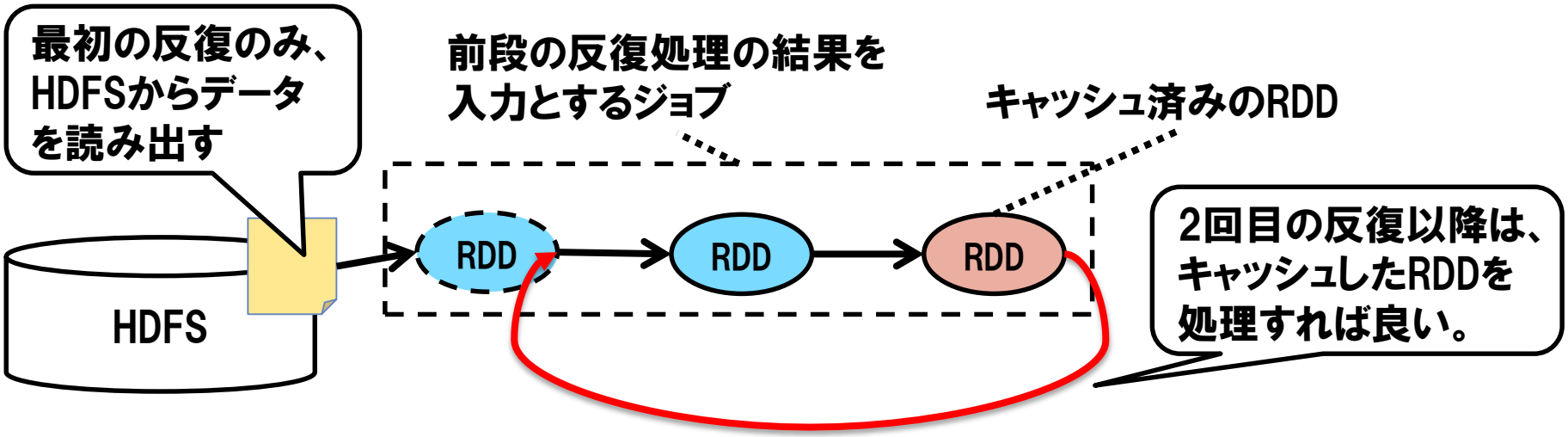
何度も利用するRDDは、複数のサーバのメモリに分割して
キャッシュしてI/Oや計算を減らせる



キャッシュを活用することで、同じデータを利用する場合でも、
都度データを読み込む必要がない

NTT DATA

キャッシュは反復処理にも有効



【DEMO】キャッシュ機能を使ってみよう

NTT DATA

先ほどのワードカウントの例で
キャッシュを使ってみましょう。

当日は省略

【DEMO】キャッシュされたのをUIから確認

NTT DATA

SparkのUIでキャッシュされたことを
確認してみましょう。

当日は省略

【DEMO】キャッシュ機能を使ってみよう

NTT DATA

WordCountの続き。先ほどのwordCountsを明示的にキャッシュしてみる

.cache () でキャッシュ機能を有効化。（ここではキャッシュされない）

```
wordCounts.cache ()
```

アクション契機に処理が実行され、データの実体がキャッシュに残る

```
wordCounts.filter (_._1.matches ("[a-zA-Z0-9]+")).sortBy (_._2,false).take (10)
res1: Array[(String, Int)] = Array((in,1766), (to,1417), (for,1298), (the,777), (and,663), (Add,631), (of,630), (Fix,547), (Xin,491), (Reynold,490))
```

```
wordCounts.filter (_._1.matches ("[a-zA-Z0-9]+")).sortBy (_._2,false).take (10)
res1: Array[(String, Int)] = Array((in,1766), (to,1417), (for,1298), (the,777), (and,663), (Add,631), (of,630), (Fix,547), (Xin,491), (Reynold,490))
```

次にキャッシュされたRDDを対象とした処理を実行すると、キャッシュから優先的にデータが読み込まれる。（キャッシュし切れなかったデータは、通常通り計算されて求められる）

【DEMO】キャッシュされたデータの様子

NTT DATA

キャッシュの状況は、WebUIのStorageタブから確認できる

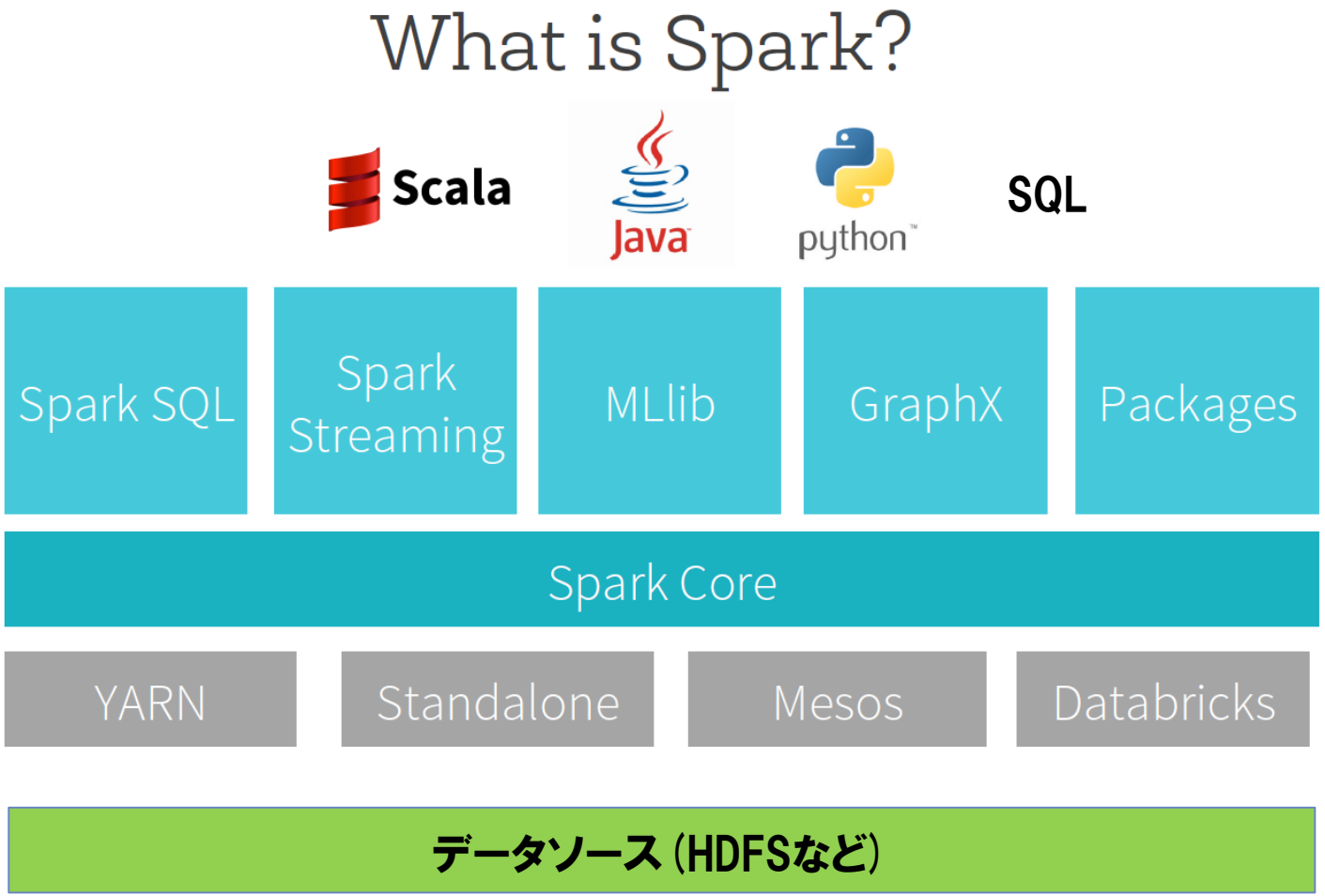
Storage

RDDs

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory
ShuffledRDD	Memory Deserialized 1x Replicated	2	100%	3.4 MB

速いだけじゃないSparkの豊富な機能

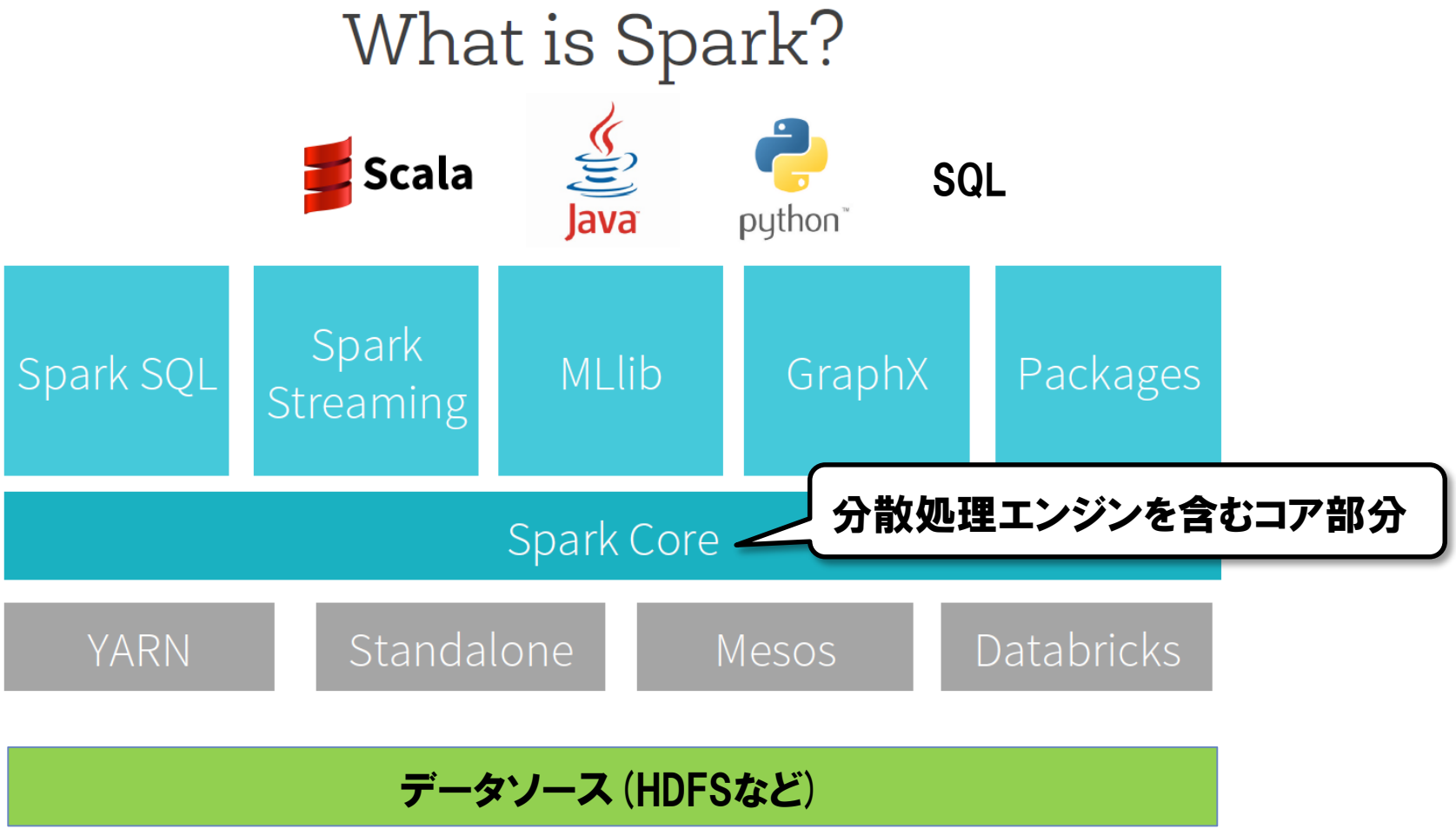
NTT DATA



http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

速いだけじゃないSparkの豊富な機能

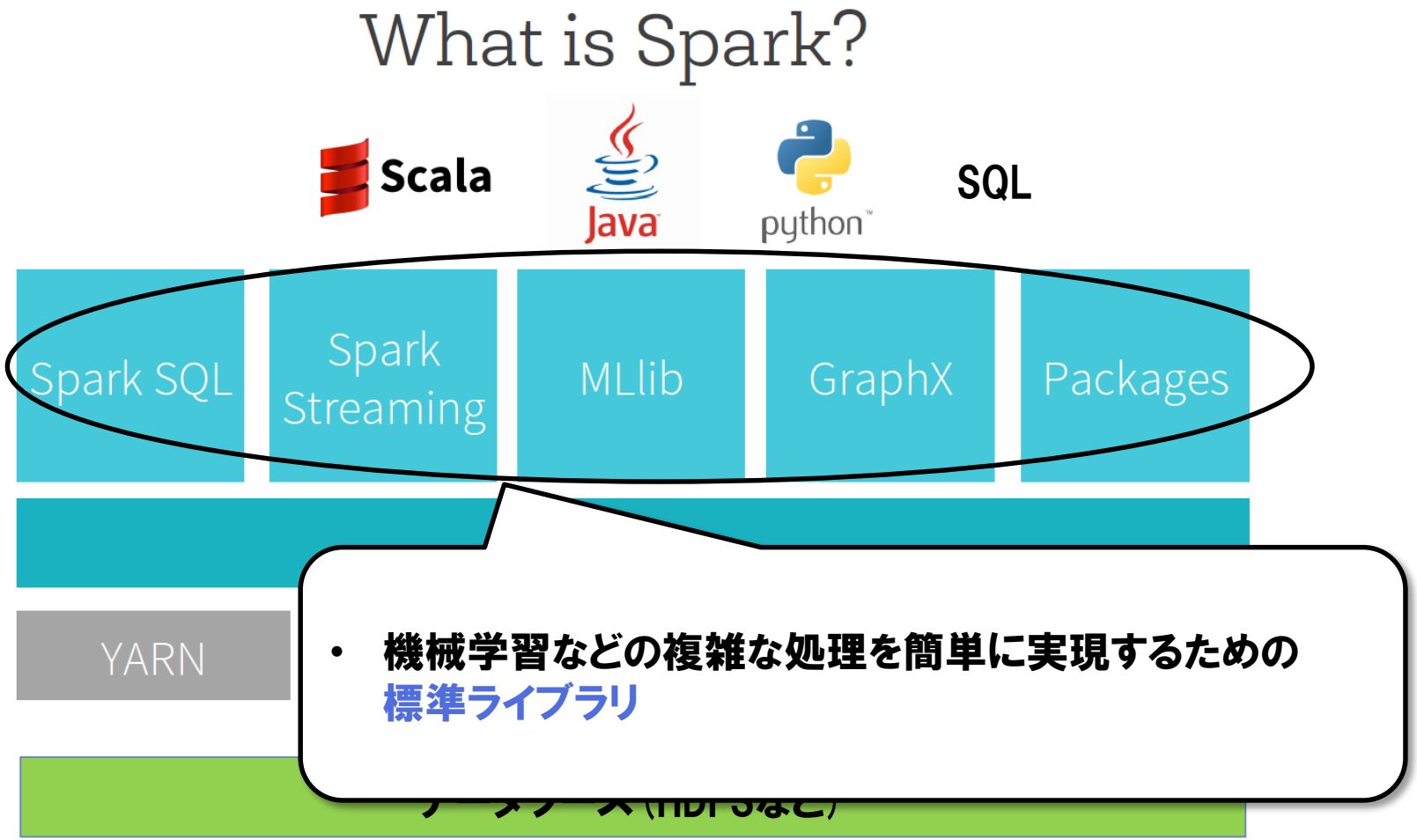
NTT DATA



http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

速いだけじゃないSparkの豊富な機能

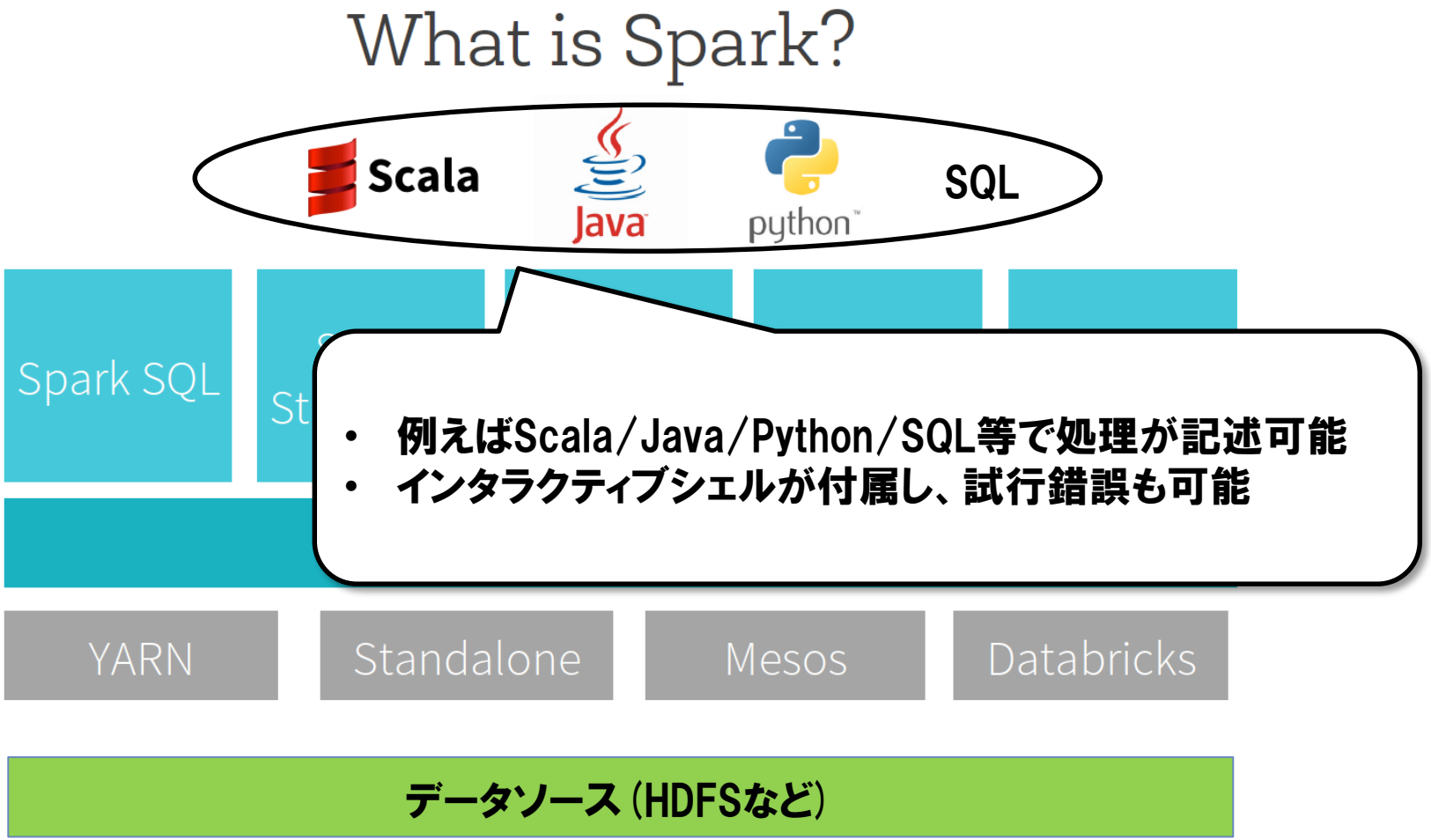
NTT DATA



http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

速いだけじゃないSparkの豊富な機能

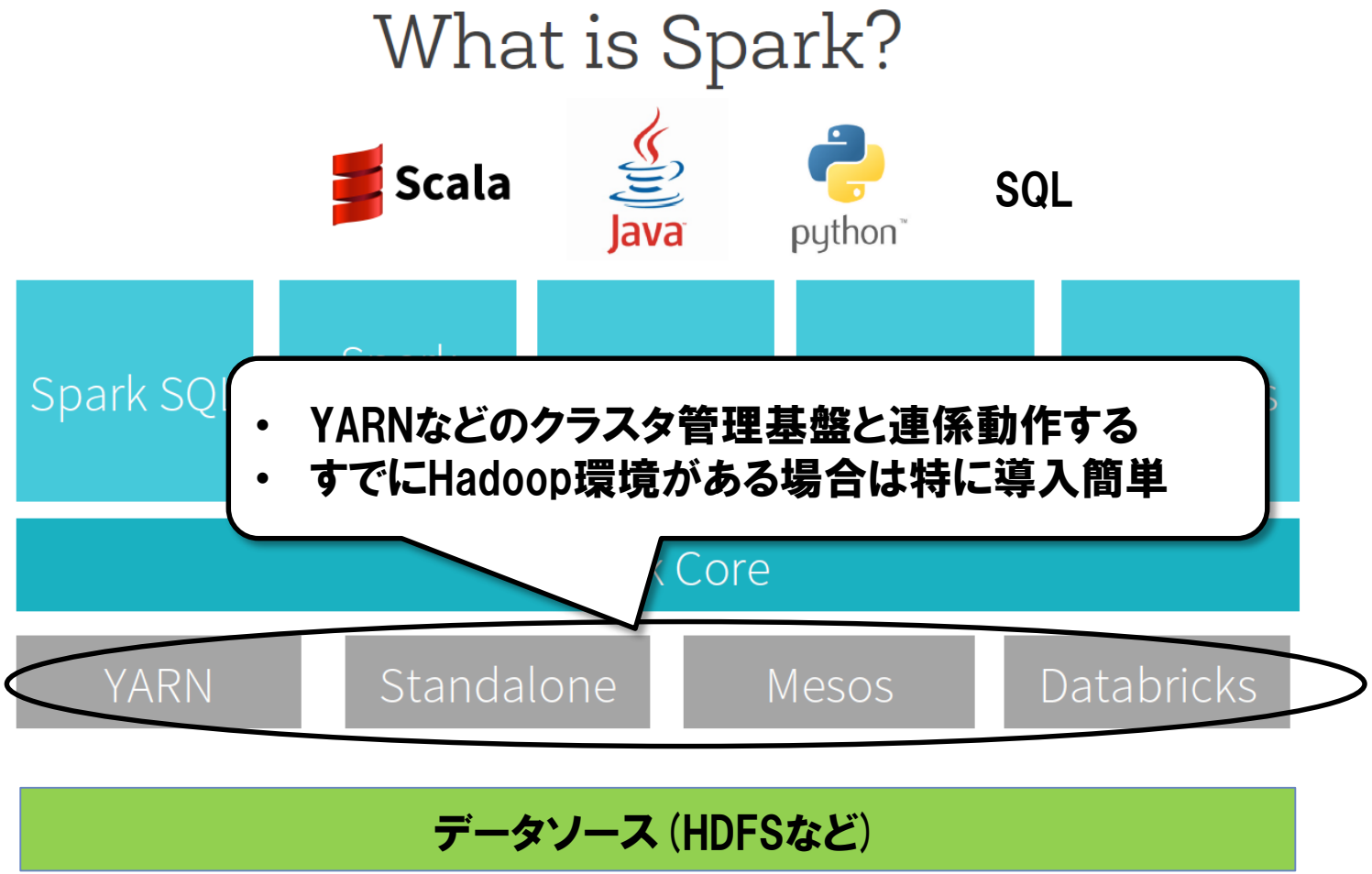
NTT DATA



- 例えばScala/Java/Python/SQL等で処理が記述可能
- インタラクティブシェルが付属し、試行錯誤も可能

http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

速いだけじゃないSparkの豊富な機能

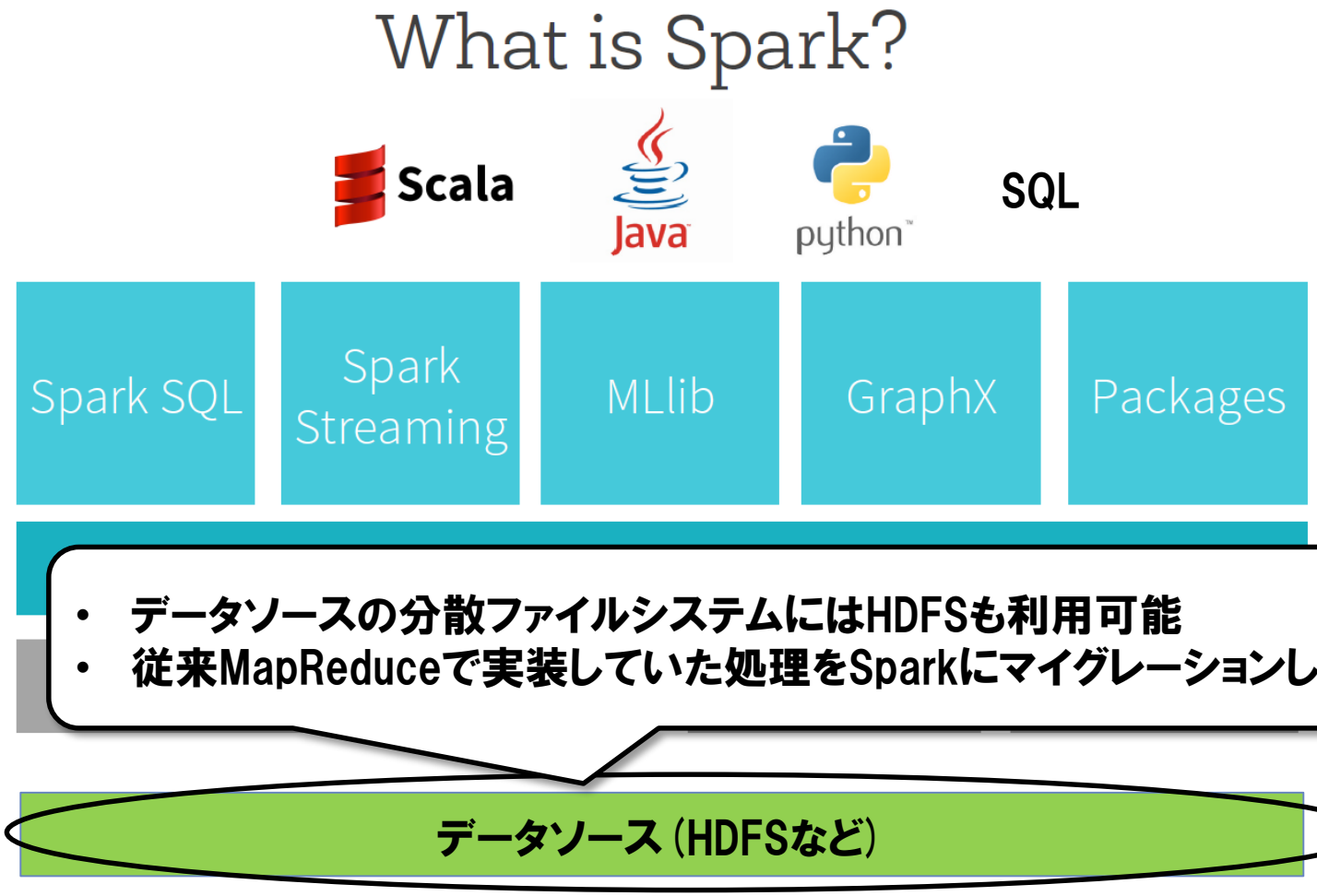


- YARNなどのクラスタ管理基盤と連係動作する
- すでにHadoop環境がある場合は特に導入簡単

http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

速いだけじゃないSparkの豊富な機能

NTT DATA



http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

便利なライブラリ: Spark SQL

NTT DATA

DataFrameに対してSQL/HiveQLを発行して分散処理する

- RDDの上に成り立つスキーマ付きのテーブル状のデータ構造

SQLを使い慣れたデータ分析者担当者が分散処理の恩恵を受けられる

```
// ScalaでSQLを使用する例
Val teenager = sqlContext.sql(“SELECT name FROM people WHERE
age >= 13 AND age <= 19”)
```

例えばPythonによるデータ分析アプリケーションを実装していて、
「ここはSQLで書きたいな」って箇所で有用。

便利なライブラリ: Spark SQL

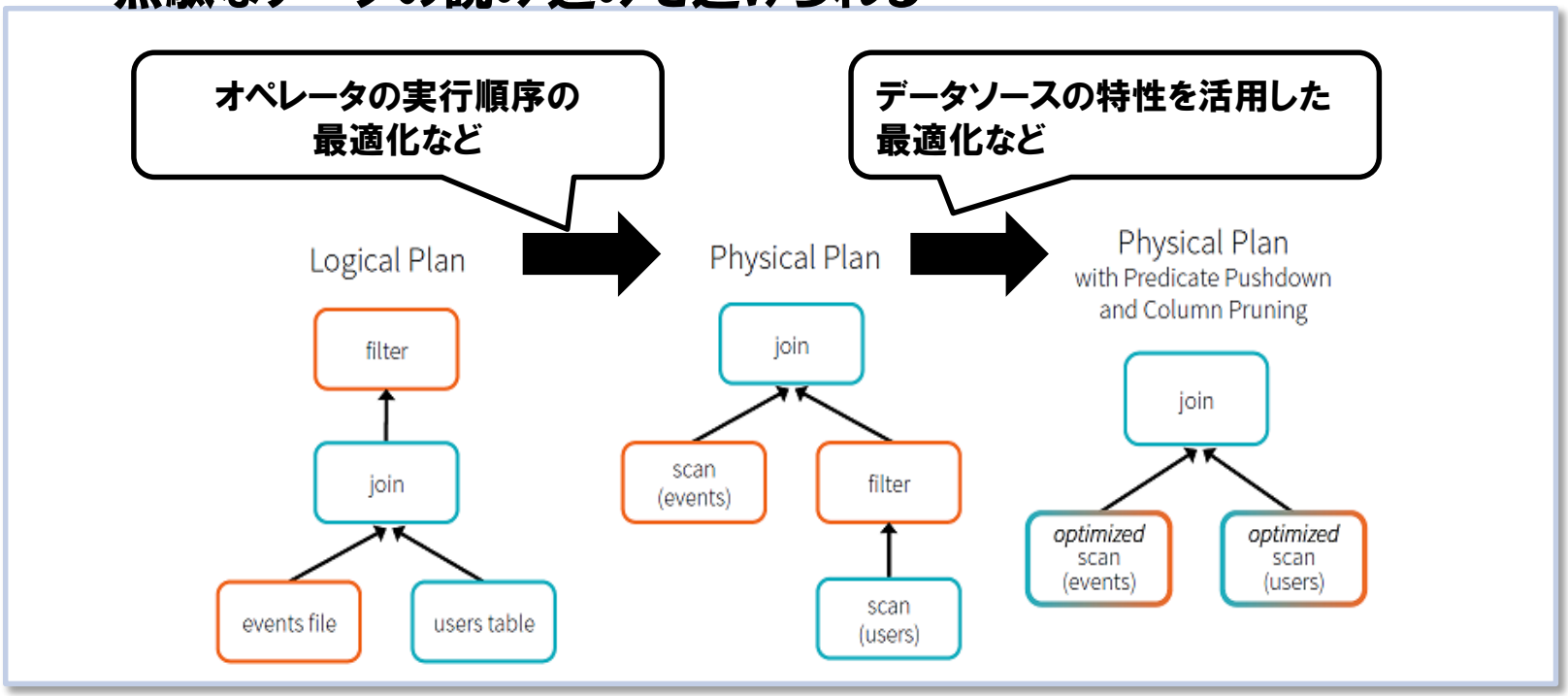
NTT DATA

オプティマイザ付き

RDDベースの処理の物理プランが生成される

構造化データを取り扱うための仕組みが付属

- Parquet / ORC / CSV / JSON / テキスト / JDBC ...
- データソースによってはフィルタ処理をデータソース側に移譲することで、無駄なデータの読み込みを避けられる



http://cdn.oreillystatic.com/en/assets/1/event/126/Apache%20Spark_%20What_s%20new_%20what_s%20coming%20Presentation.pdf

【DEMO】SQLを使って処理を記述してみましょう

NTT DATA

Sparkのソースコード群に含まれている
サンプルデータを読みこんで処理してみます。

【DEMO】SQLによる処理を実行してみる

NTT DATA

サンプルに含まれるデータで、SparkSQLを試してみる

参考：http://spark.apache.org/docs/latest/sql-programming-guide.html

```
val sparkHome = sys.env("SPARK_HOME")
val path = s"${sparkHome}/examples/src/main/resources/users.parquet"
val users = sqlContext.read.parquet(path)

users.registerTempTable("users")

val redLover = sqlContext.sql("SELECT name, favorite_color FROM users WHERE favorite_color = 'red'").show
```

便利なライブラリ: MLlib / ML Pipelines

NTT DATA

統計処理、機械学習を分散処理するためのライブラリ

レコメンデーション / 分類 / 予測など

分散処理に向くポピュラーなアルゴリズムを提供

2016/5現在、spark.mllib と spark.ml の2種類のパッケージが存在

RDD利用　　DataFrame利用

Algorithms (一例)

MLlib contains the following algorithms and utilities:

- logistic regression and linear support vector machine (SVM)
- classification and regression tree
- random forest and gradient-boosted trees
- recommendation via alternating least squares (ALS)
- clustering via k-means, bisecting k-means, Gaussian mixtures (GMM), and power iteration clustering
- topic modeling via latent Dirichlet allocation (LDA)
- survival analysis via accelerated failure time model
- singular value decomposition (SVD) and QR decomposition
- principal component analysis (PCA)
- linear regression with L_1, L_2, and elastic-net regularization
- isotonic regression
- multinomial/binomial naive Bayes
- frequent itemset mining via FP-growth and association rules
- sequential pattern mining via PrefixSpan
- summary statistics and hypothesis testing
- feature transformations
- model evaluation and hyper-parameter tuning

便利なライブラリ: MLlib / ML Pipelines

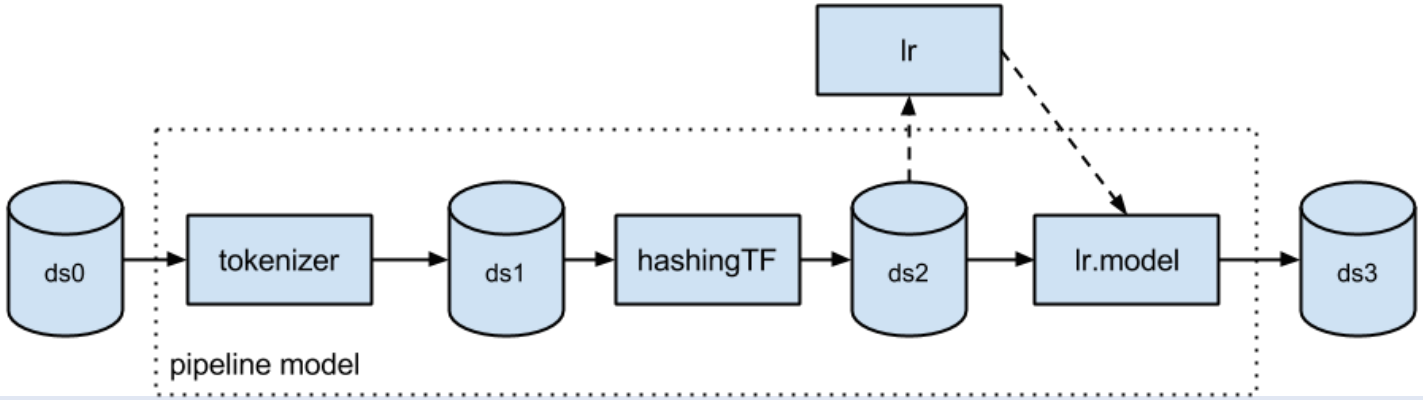
NTT DATA

昨今はspark.mlのML Pipelinesの開発が活発

scikit-learnのように機械学習を含む処理全体をパイプラインとして扱うAPI

処理全体のポータビリティ向上！
見通しの良くなって開発効率の向上！

```
val tokenizer = new Tokenizer()
                      .setInputCol("text")
                      .setOutputCol("words")
val hashingTF = new HashingTF()
                       .setNumFeatures(1000)
                       .setInputCol(tokenizer.getOutputCol)
                       .setOutputCol("features")
val lr = new LogisticRegression()
            .setMaxIter(10)
            .setRegParam(0.01)
// パイプラインにトークン分割、ハッシュ化、処理とロジスティック回帰を設定
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))
val model = pipeline.fit(trainingDataset) // モデルの当てはめ
```



【DEMO】K-meansを使ってクラスタリング

NTT DATA

DataFrameで保持されたデータを
クラスタリングしてみます。

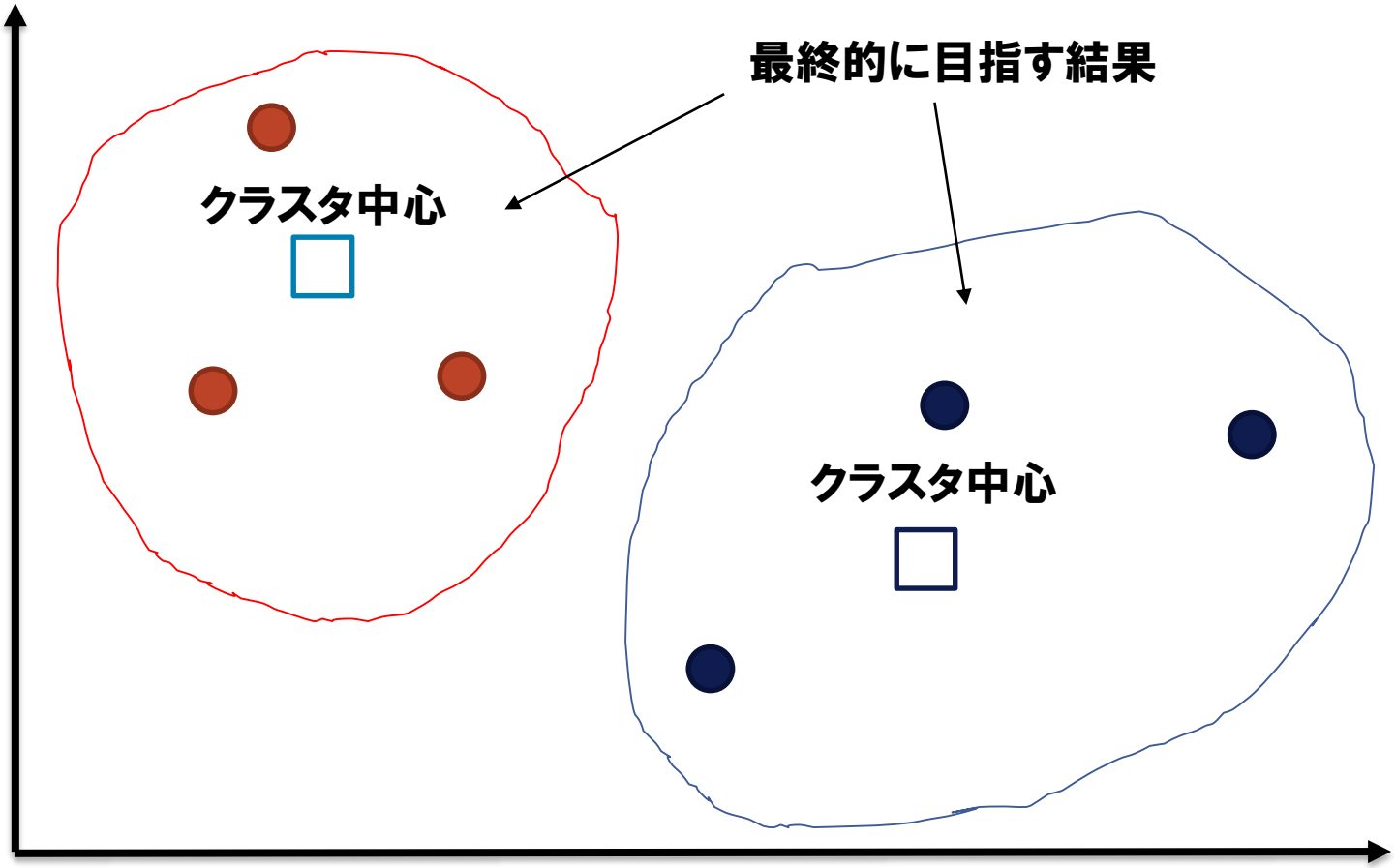
LTで紹介

（補足）K-Meansの直感的なイメージ

NTT DATA

シンプルで分散処理にも実装しやすい 手堅い クラスタリング・アルゴリズム

主観的印象



【DEMO】K-meansによるクラスタリング(１)

NTT DATA

あやめデータでクラスタリングを試してみる。
http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
まずはモデル生成まで。

```
// あらかじめダウンロード
// wget http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data -O /tmp/iris.data

// ライブラリを読み込みながらspark-shellを起動
// spark-shell --packages com.databricks:spark-csv_2.10:1.4.0

import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

// 入力データの定義とテーブル登録
val data = sqlContext.read.format("com.databricks.spark.csv").option("inferSchema",
"true").load("/tmp/iris.data")
sqlContext.udf.register("toVector", (a: Double, b: Double, c:Double,  d:Double) =>
Vectors.dense(a, b, c, d))

// モデル生成
val features = data.selectExpr("toVector(C0, C1, C2, C3) as feature", "C4 as name")
val kmeans = new
KMeans().setK(3).setFeaturesCol("feature").setPredictionCol("prediction")
val model = kmeans.fit(features)
```

【DEMO】K-meansによるクラスタリング（２）

NTT DATA

次にモデルを使ってクラスタ判定結果を取得

```
// 各ベクトルのクラスタを判定
val predicted = model.transform(features)
predicted.show
predicted.registerTempTable("predicted")

import org.apache.spark.sql.expressions.Window

// 各グループから適当に3個取りだしてみる。
val top3 = sqlContext.sql("SELECT * FROM (SELECT *, ROW_NUMBER() OVER (PARTITION BY
name) AS rn FROM predicted) x WHERE rn <= 3")
top3.show()
```

（補足）Sparkは何が嬉しいのか？

NTT DATA

例えば、これが大きなデータだったら…？

小さなデータと同様に**試行錯誤**しながら、大きなデータを扱えるのは強力

標準機能＋ライブラリの**プロゴラマブルなデータ処理**

SQLも利用できるし、Scala、Pythonなどで実装可能

さらに、自前でアルゴリズムを実装する人たちにとって、

RDDやDataFrameなどのAPIは柔軟で便利

（分散処理の仕組みとしては…）

突き詰めると、**やりたいことに素早く近づけていくこと**

（補足）Sparkでは何に気を付けるべき？

NTT DATA

本格的に使おうとしたら、それなりに気を付けるべき

便利な分散処理基盤とはいえ、「分散処理を忘れられる」ほどではない。
例えば処理コストの高いシャッフルを減らす工夫、非正規化などのコツ。

要件や実行環境に応じたチューニングも必要
OS、JVM、Hadoop、Sparkなどを対象とし、必要に応じて。

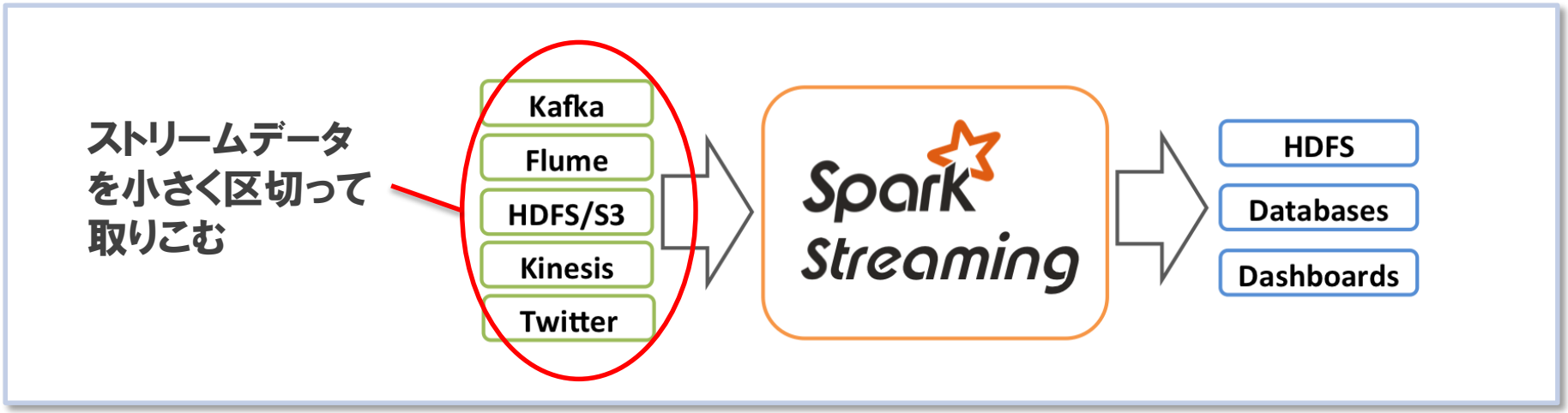
Hadoopと同様の運用設計は必要
分散処理に慣れていない方からすると、マシン1台で動作するような処理基盤とは印象が異なるはず
とはいえ基本は、メトリクス観測、監視、故障対応、などの作りこみ

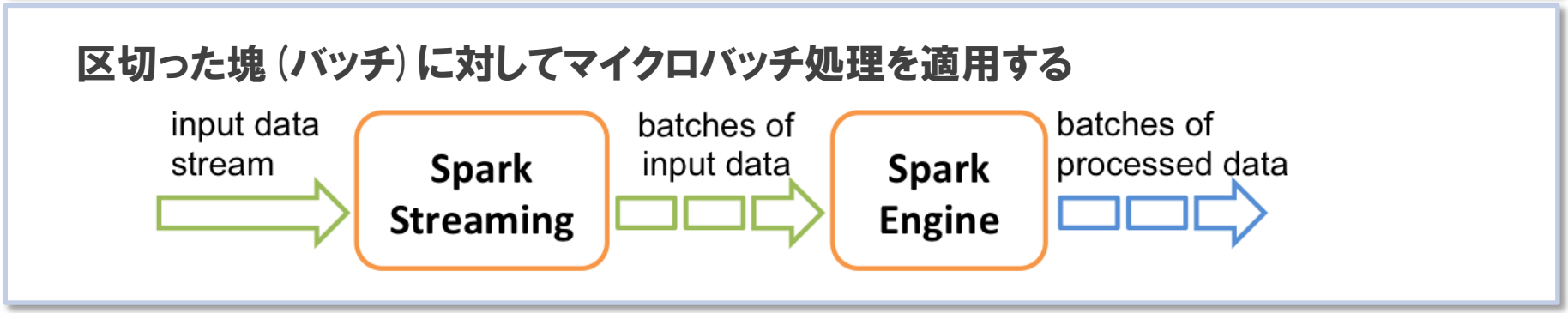
便利なライブラリ: Spark Streaming

NTT DATA

小さなバッチ処理を繰り返して**ストリーミング処理**を実現

ストリーム処理用のスケジューラが以下の流れを繰り返し実行。実際の**データ処理はSparkの機能で実現**。





http://spark.incubator.apache.org/docs/latest/streaming-programming-guide.html

便利なライブラリ: Spark Streaming

NTT DATA

特に向いているケース

集計を中心とした処理
バッチ処理と共通の処理
ある程度のウィンドウ幅でスループットも重視する処理

【DEMO】Twitterのハッシュタグフィルタリング

NTT DATA

Spark Streamingでお手軽に
Twitterデータを読みこんで
フィルタリング処理してみます

当日は省略

翔泳社「Apache Spark入門」について

NTT DATA

ここで紹介した内容の**もっと詳しい版**が掲載されています。
Sparkの**動作原理**から標準ライブラリを使った**具体的なプログラム例**まで。

第1章：Apache Sparkとは
第2章：Sparkの処理モデル
第3章：Sparkの導入
第4章：Sparkアプリケーションの開発と実行
第5章：基本的なAPIを用いたプログラミング
第6章：構造化データセットを処理する - Spark SQL -
第7章：ストリームデータを処理する - Spark Streaming -
第8章：機械学習を行う - MLlib -
Appendix
A. GraphXによるグラフ処理
B. SparkRを使ってみる
C. 機械学習とストリーム処理の連携
D. Web UIの活用

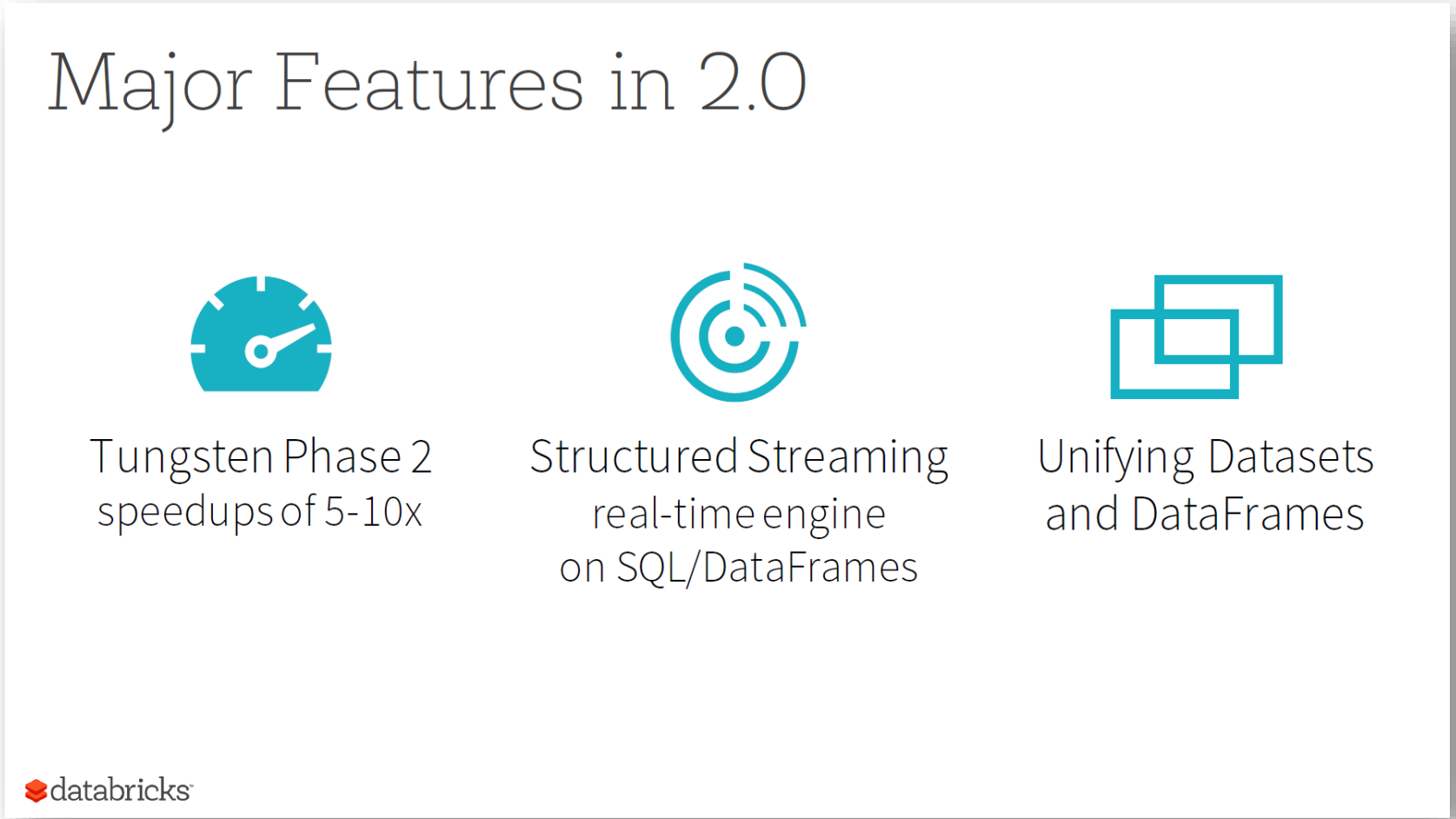
Spark1.5系
対応

ほんの少しだけ「Spark2.0」

Spark 2.0がもうすぐやってくる

NTT DATA

Spark2.0のプレビュー版が出ました。
以下、Mateiの講演より。



http://www.slideshare.net/databricks/2016-spark-summit-east-keynote-matei-zaharia

NTT DATA

Global IT Innovator

お問い合わせ先：
株式会社NTTデータ 基盤システム事業本部
OSSプロフェッショナルサービス
URL: http://oss.nttdata.co.jp/hadoop
メール： hadoop@kits.nttdata.co.jp TEL： 050-5546-9000