分散処理アプリ演習　第3回
MapReduceプログラミング基礎

（株）NTTデータ

講義内容

1. 導入
   - Hadoop MapReduceの構成要素、処理単位、クラスタとジョブ/タスクの関係
2. Hadoop MapReduceプログラミング
   - MapReduceジョブの構成要素、各構成要素に対応するAPI

1. 導入

Hadoop MapReduceとは

- 大規模データをMapReduceで分散処理するためのソフトウェアフレームワーク
- MapReduceを実行するための基盤としてのミドルウェアと、処理およびミドルウェアの制御を行うAPIから構成される
- Hadoop MapReduceでは処理を「ジョブ」や「タスク」という単位で管理する
- MapReduceジョブはミドルウェアが動作する複数のサーバから構成されるクラスタ上で動作する

※本講義内では混乱を避けるため、処理としての「MapReduce」とフレームワークとしての「Hadoop MapReduce」を区別する

Hadoop MapReduceの構成要素

- ミドルウェア
  - JobTracker
    - ジョブやタスクの進捗と成否およびTaskTrackerの死活確認などを行う、クラスタの管理ノード
  - TaskTracker
    - タスク（後述）の処理やJobTrackerへの進捗報告、死活確認のためのハートビートの送信を行う、クラスタの計算ノード

- API
  - MapReduceジョブを起動するAPI
  - MapReduceアプリケーションの開発に関するAPI
  - ミドルウェアの制御を行うAPI
  - Etc...

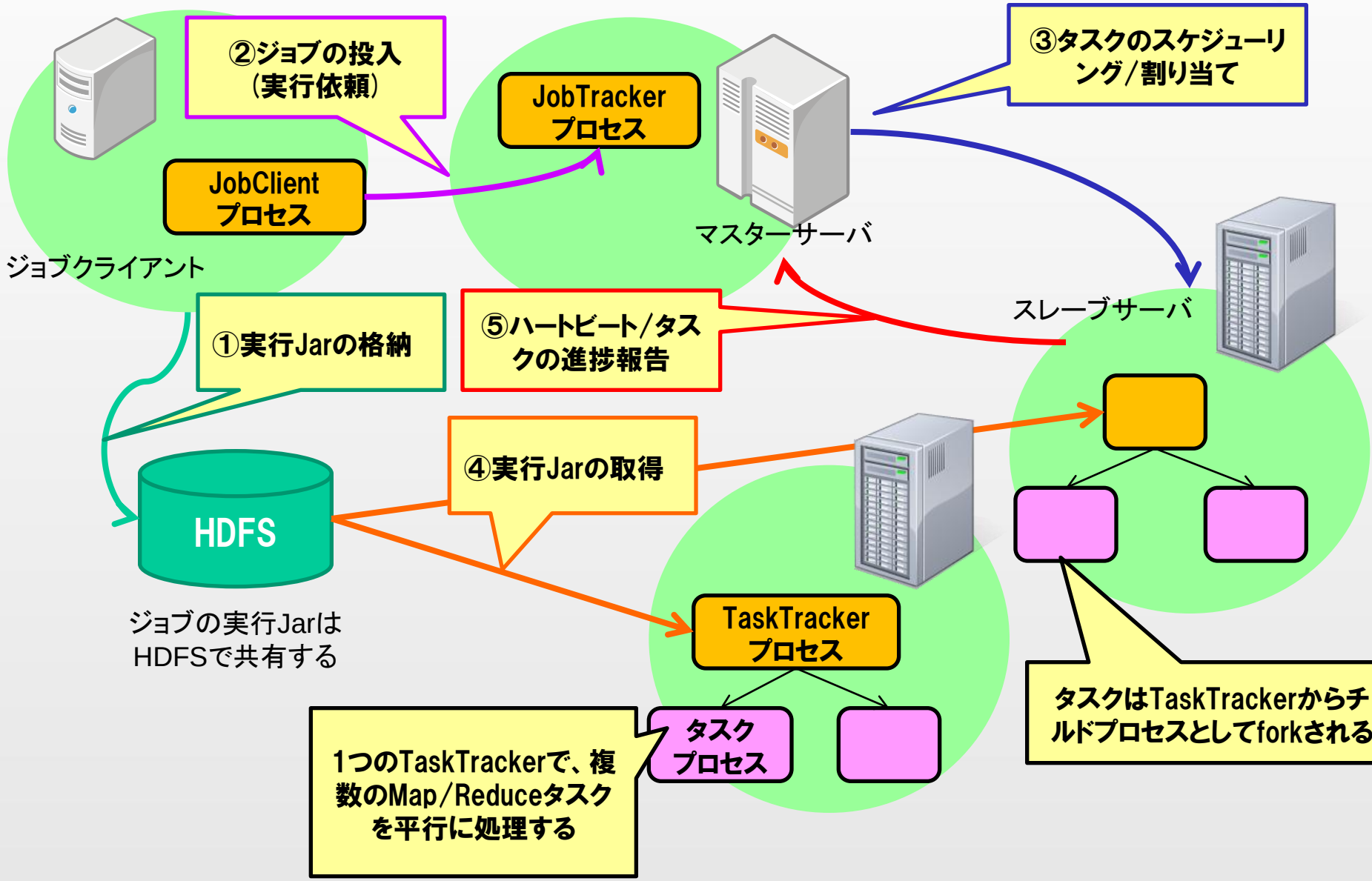
  アプリケーションを開発する際には、Hadoop MapReduce APIのほかに、Hadoop Commons APIやHadoop HDFS APIを組み合わせて使う

Hadoop MapReduceの処理単位

- ジョブ
  - 1回のMapReduceの処理。Mapフェーズ/Shuffleフェーズ/Reduceフェーズの大きく3つのフェーズで構成される。ジョブはさらに、Mapフェーズで実行されるMapタスクとReduceフェーズで実行されるReduceタスクに分割される

- タスク
  - Mapタスク
    - MapReduceのmap処理を行う。Hadoop MapReduceでは、一度に複数のMapタスクが同時に実行される
    - 主としてフィルタリングや抽出、データの加工を行う
  - Reduceタスク
    - MapReduceのreduce処理を行う。Hadoop MapReduceでは、すべてのMapタスクが終了した後にReduceフェーズが実行され、その中で一度に複数のReduceタスクが実行される
    - 主としてMapタスクで処理した結果のデータに対する集約処理

Hadoopクラスタとジョブ/タスクの関係

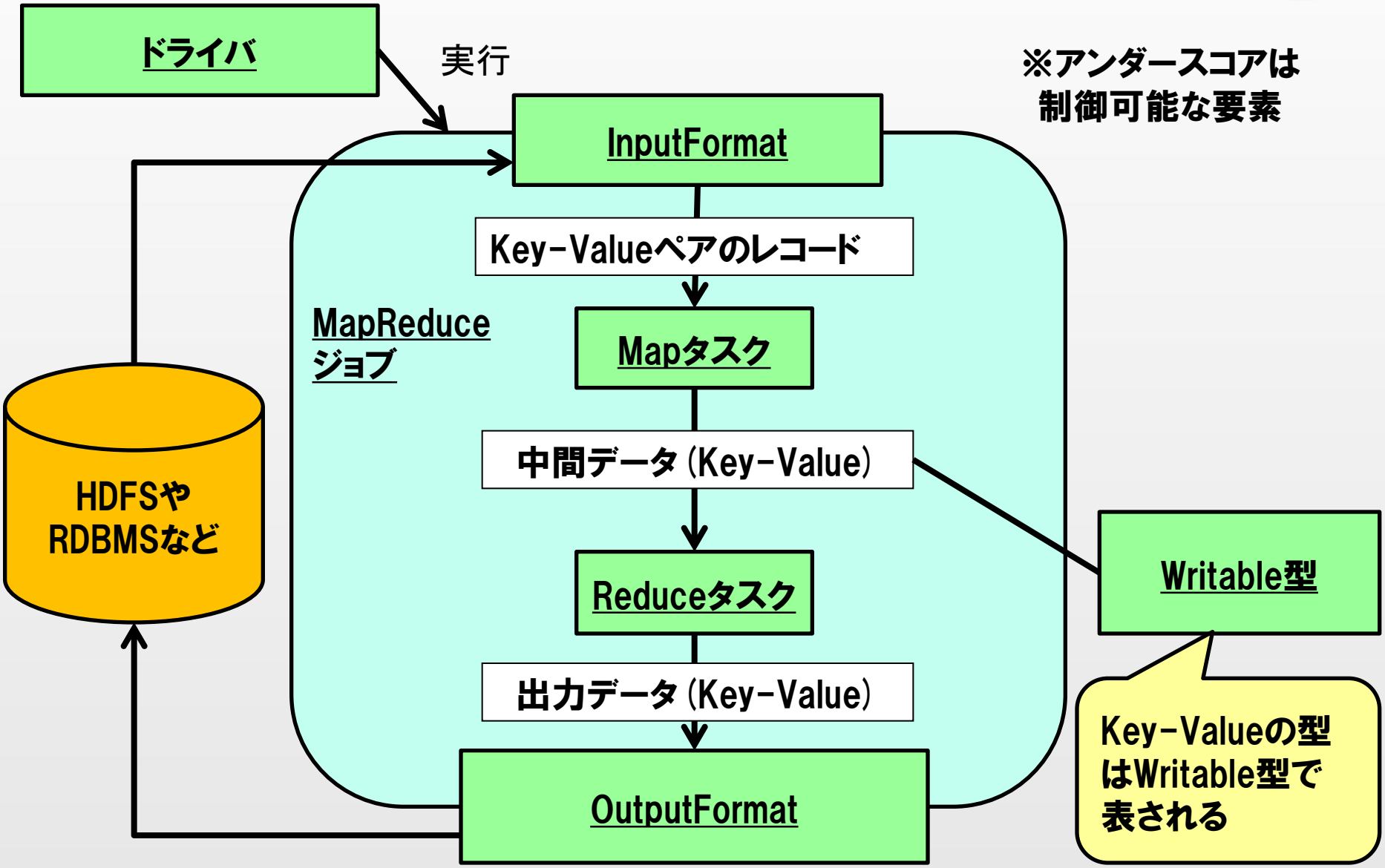


2. Hadoop MapReduceプログラミング

MapReduceジョブの基本的な構成要素

- タスク
  - Mapタスク
  - Reduceタスク
- InputFormat（入力データの読み込み方）
  - データソース（HDFSやRDBMS）
  - レコードの定義
  - Key-Valueの定義
- OutputFormat（出力データの書き出し方）
  - データストア（HDFSやRDBMS）
  - レコードの定義
  - Key-Valueの定義
- Writable型
  - 入出力データ、中間データのKey-Valueの型
- Etc...

MapReduceジョブの基本的な構成要素（図解）



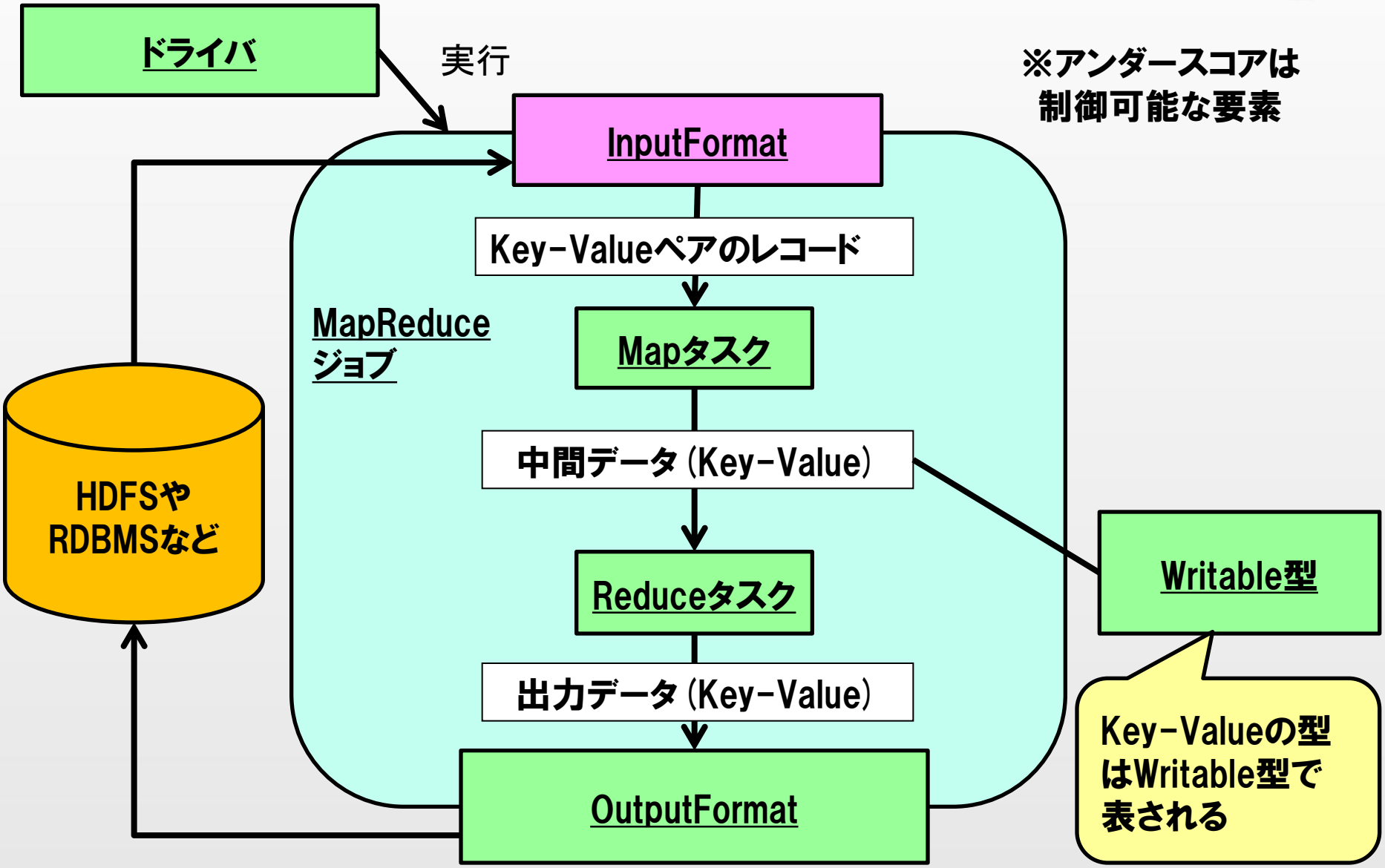
ここからの内容

① それぞれの構成要素をAPIでどのように扱うのか順に追って理解する

② 続いて個々の構成要素を組み合立ててMapReduceジョブを構成する方法を理解する

③ 最後に組み立てたMapReduceジョブを動作させるドライバクラスの作り方を理解する

InputFormat



InputFormatとは

- 入力データの各種属性を決定する

- データソース (HDFSやRDBMS)
  - 入力元はどこか (HDFS? RDBMS?)
- データ形式の定義
  - 何を読み込むか (テキストファイル? バイナリファイル? DBのテーブル?)
- レコードの定義
  - 読み込んだデータ中の何をレコードとみなすか (行? 固定バイト?)
- レコード中のKey-Valueの定義
  - レコード中の何をKey-Valueとみなすか
- Key-Valueのデータ型
  - Key-Valueの型は何を選ぶか
- スプリットの定義
  - どのように分割するか
- Etc...

スプリットの分割の仕方が決まるので、Mapタスクの数が決まる

InputFormatを表すAPI（InputFormatクラス）

- InputFormatを表すAPI（InputFormatクラス）には様々な派生クラスがある

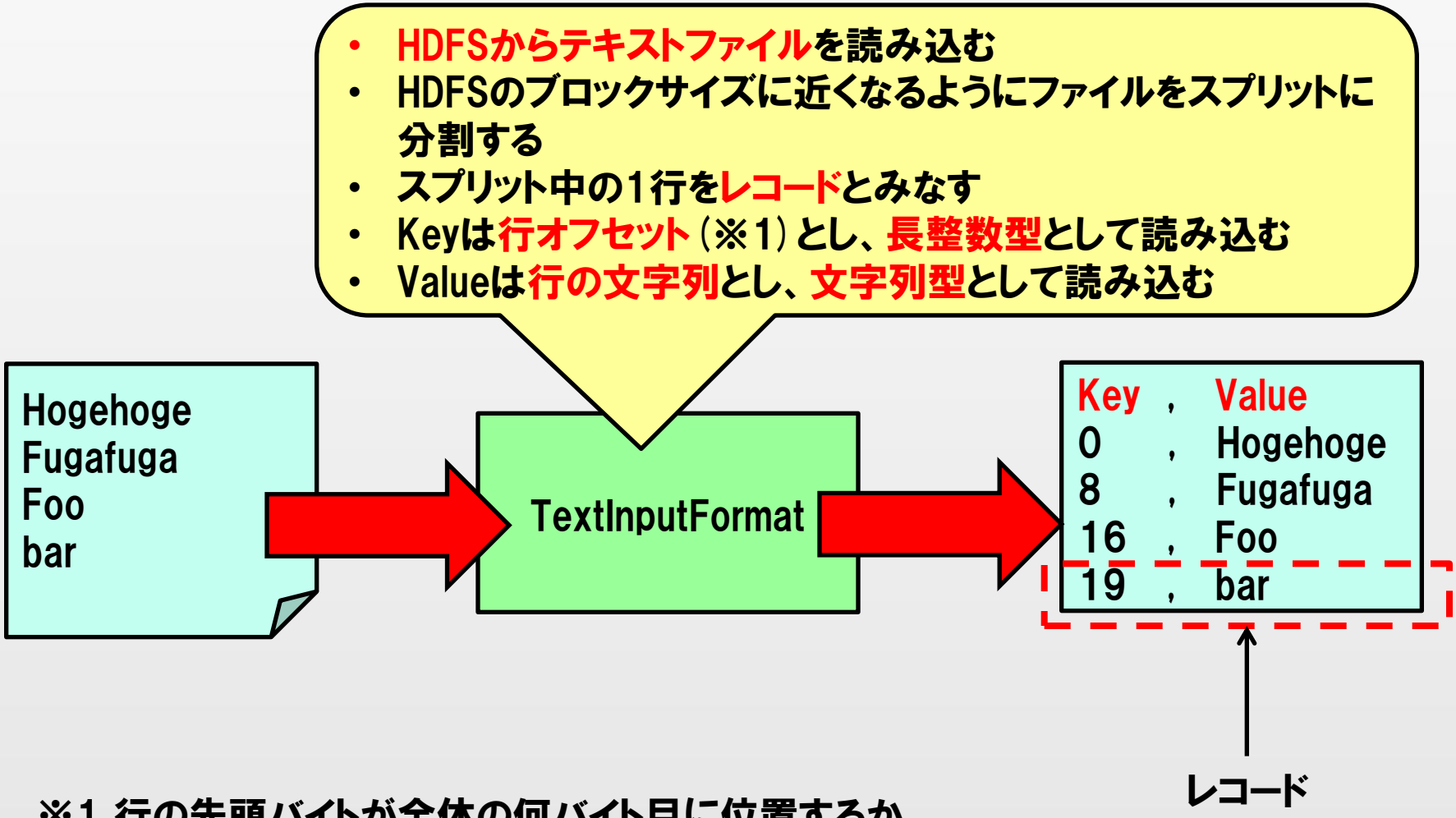
- TextInputFormat
  - テキストファイルを読み込むためのInputFormat
- DBInputFormat
  - RDBMSからテーブルを読み込むためのInputFormat
- Etc...

利用するInputFormatを選ぶ際のポイント

① どこから読み込むか（データソースは何か）
② 何を読み込むか
③ 何をレコードとみなすか
④ レコード中の何をKey/Valueとみなし、どのような型で読むか
⑤ どのようにスプリットを分割するか

InputFormatを表すAPI（InputFormatクラス）

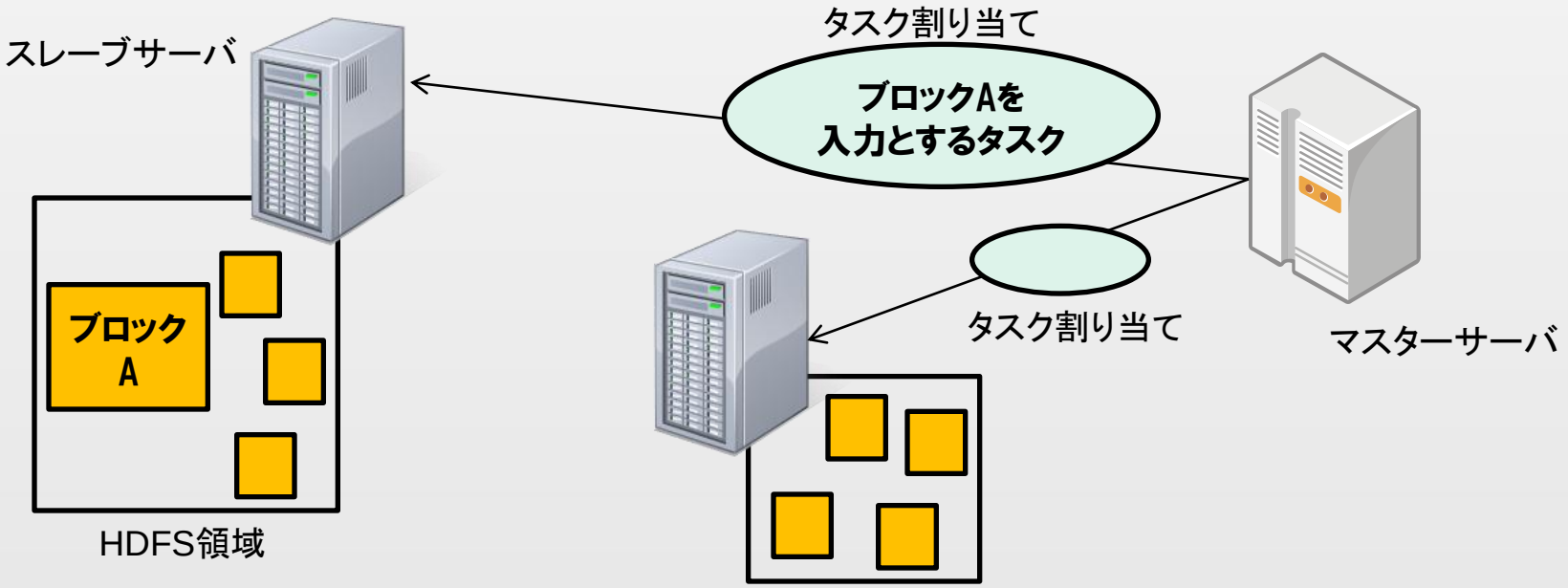
- データソース/レコード/Key/Valueの解釈（TextInputFormatの場合）



※1 行の先頭バイトが全体の何バイト目に位置するか

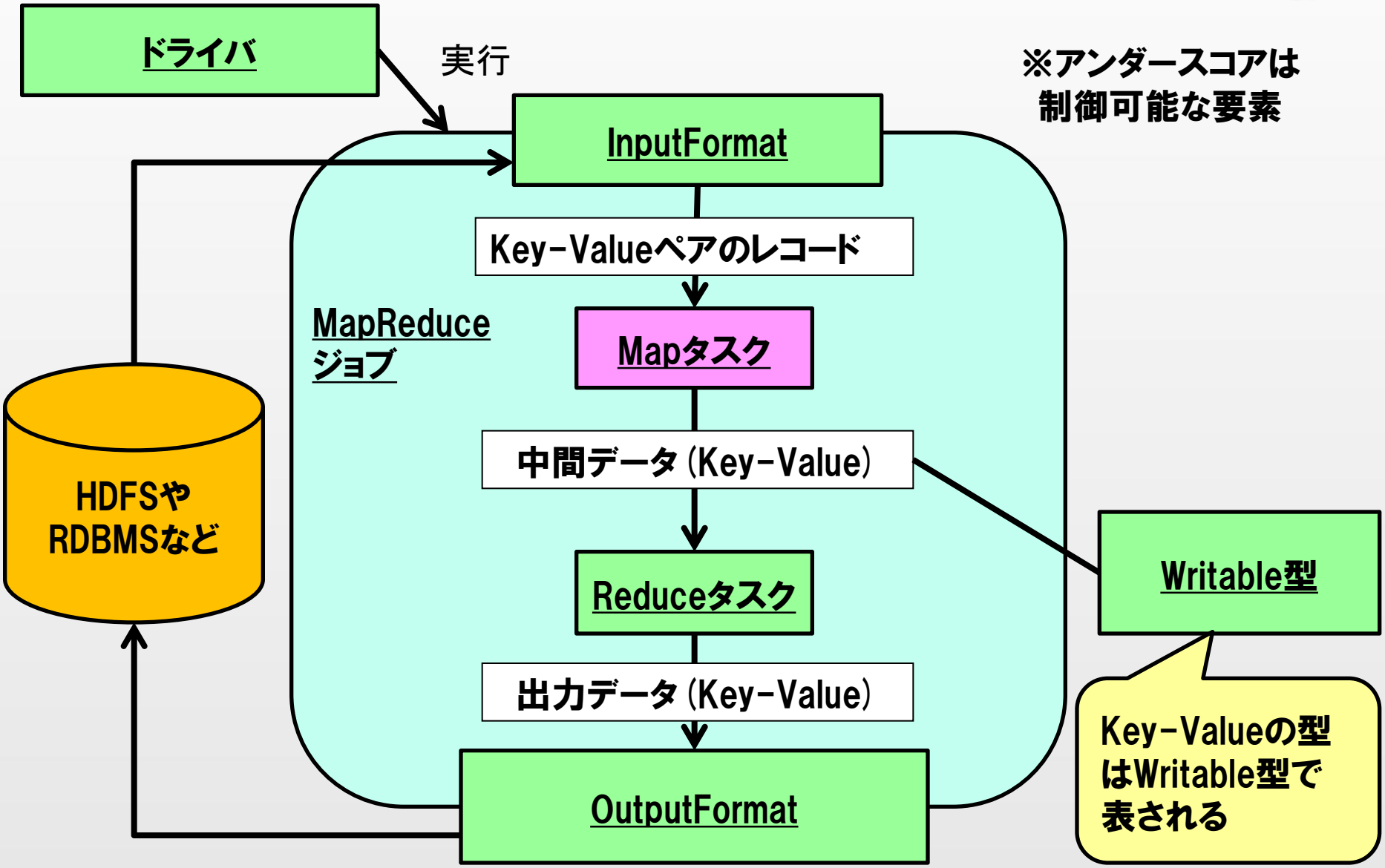
HDFSとの連携

- Hadoop MapReduceは、HDFSとの連係動作が最適化されている
- FileInputFormatから派生するAPI（TextInputFormatなど）でHDFSと連携可能
  - スプリットのサイズがHDFSのブロックサイズ（デフォルト64MB）に近くなるように入力ファイルが分割されるため、入力データが大きい場合スループットが出やすい
  - データローカリティを考慮してタスクのアサインがスケジューリングされる。通常、TaskTrackerとDataNodeは同じスレーブサーバに同居させるため、手元にあるブロックを優先的に処理するように最適化される。これにより通信コストを軽減できる

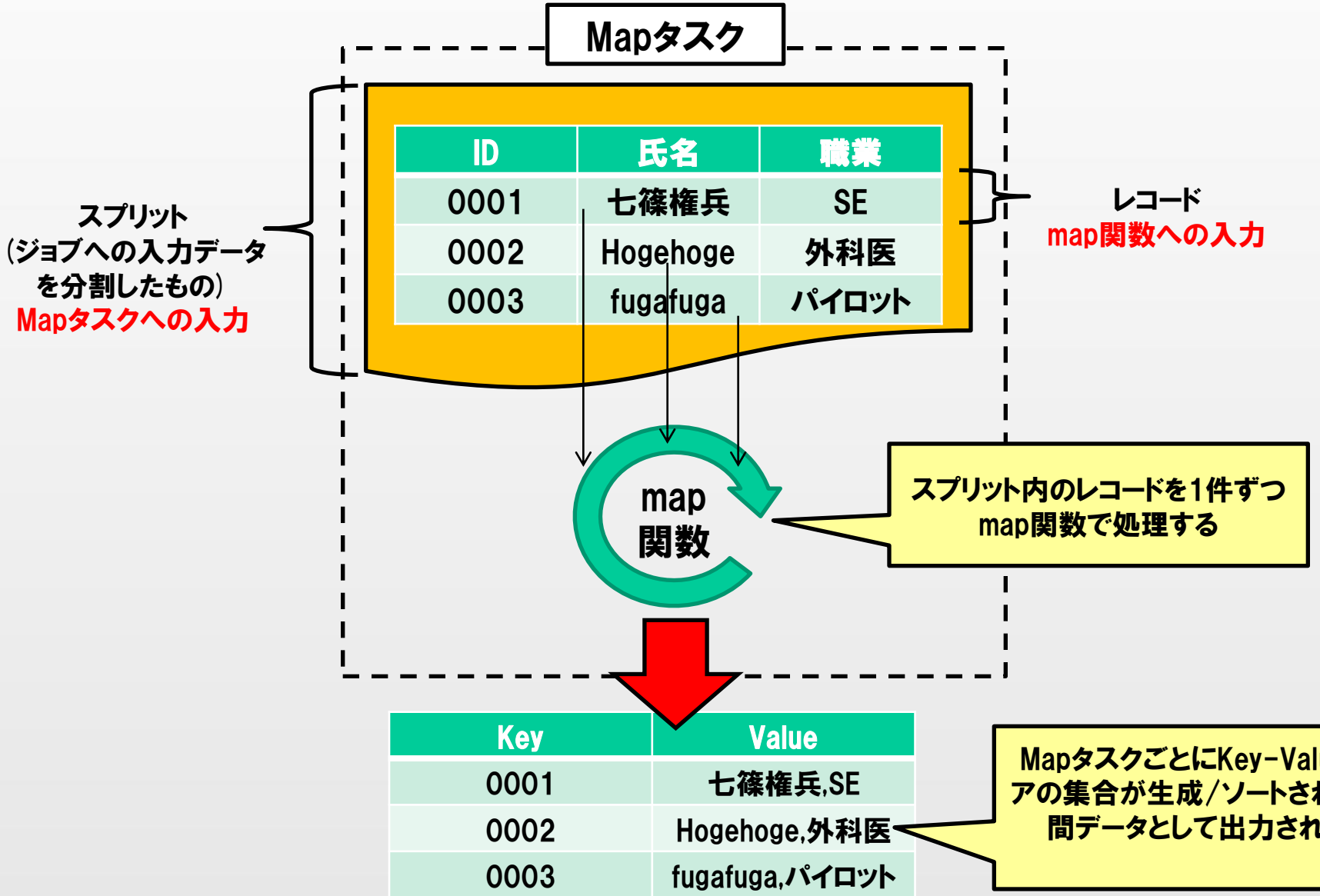


データローカリティのイメージ

Mapタスク


ドライバ
実行
※アンダースコアは
制御可能な要素
InputFormat
Key-Valueペアのレコード
MapReduce
ジョブ
Mapタスク
中間データ (Key-Value)
Reduceタスク
出力データ (Key-Value)
OutputFormat
HDFSや
RDBMSなど
Writable型
Key-Valueの型
はWritable型で
表される

Mapタスクでの処理


Mapタスク
ID
氏名
職業
0001
七篠権兵
SE
0002
Hogehoge
外科医
0003
fugafuga
パイロット
スプリット
(ジョブへの入力データを分割したもの)
Mapタスクへの入力
レコード
map関数への入力
map
関数
スプリット内のレコードを1件ずつmap関数で処理する
Key
Value
0001
七篠権兵,SE
0002
Hogehoge,外科医
0003
fugafuga,パイロット
MapタスクごとにKey-Valueペアの集合が生成/ソートされ、中間データとして出力される

Mapタスクを表すAPI（Mapperクラス）

- Mapタスクでの処理とAPIの対応は次の通り

MapReduce	Hadoop MapReduce API
Mapタスク	Mapperクラス
map関数	Mapper#mapメソッド

Mapタスクを表すAPI（Mapperクラス）

- MapタスクはMapperクラスで表される。Mapperクラス自体はジェネリッククラスとして定義されている。独自のMapperクラスは、これを継承して実装する

入力データのKey-Value、中間データのKey-Valueの順に、それぞれの型を型パラメータとして指定する。中間データのKey-Valueや出力データのKey－Valueの型の設定方法は後述

```
public class MyMapper extends Mapper <LongWritable , Text , Text , IntWritable> {
        ・・・
}
```

Mapperクラスを継承し、独自のMapperクラスを作る

mapメソッドの実装

- map関数はMapperクラスのmapメソッドをオーバーライドして実装する
- フレームワークは、Mapタスクに割り当てたスプリットから1レコードずつ取り出し、mapメソッドを呼び出して渡す

入力データのKeyが渡される。
型には入力データのKeyの型を指定する（入力データのKeyの型はInputFormatに依存する）

入力データのKeyに紐づいたValueが渡される。
型には入力データのValueの型を指定する（入力データのValueの型はInputFormatに依存する）

```
@Override
public void map ( LongWritable key , Text value , Context context)
        throws IOException , InterruptedException {

    ・・・処理・・・

    context.write ( outKey , outValue) ;
```

Contextオブジェクトのwriteメソッドで中間データを書き出す。引数はKey、Valueの順に指定する。出力するKeyとValueの型はクラスの型パラメータで指定した型パラメータと一致させる必要があることに注意

初期化/終了時処理の実装

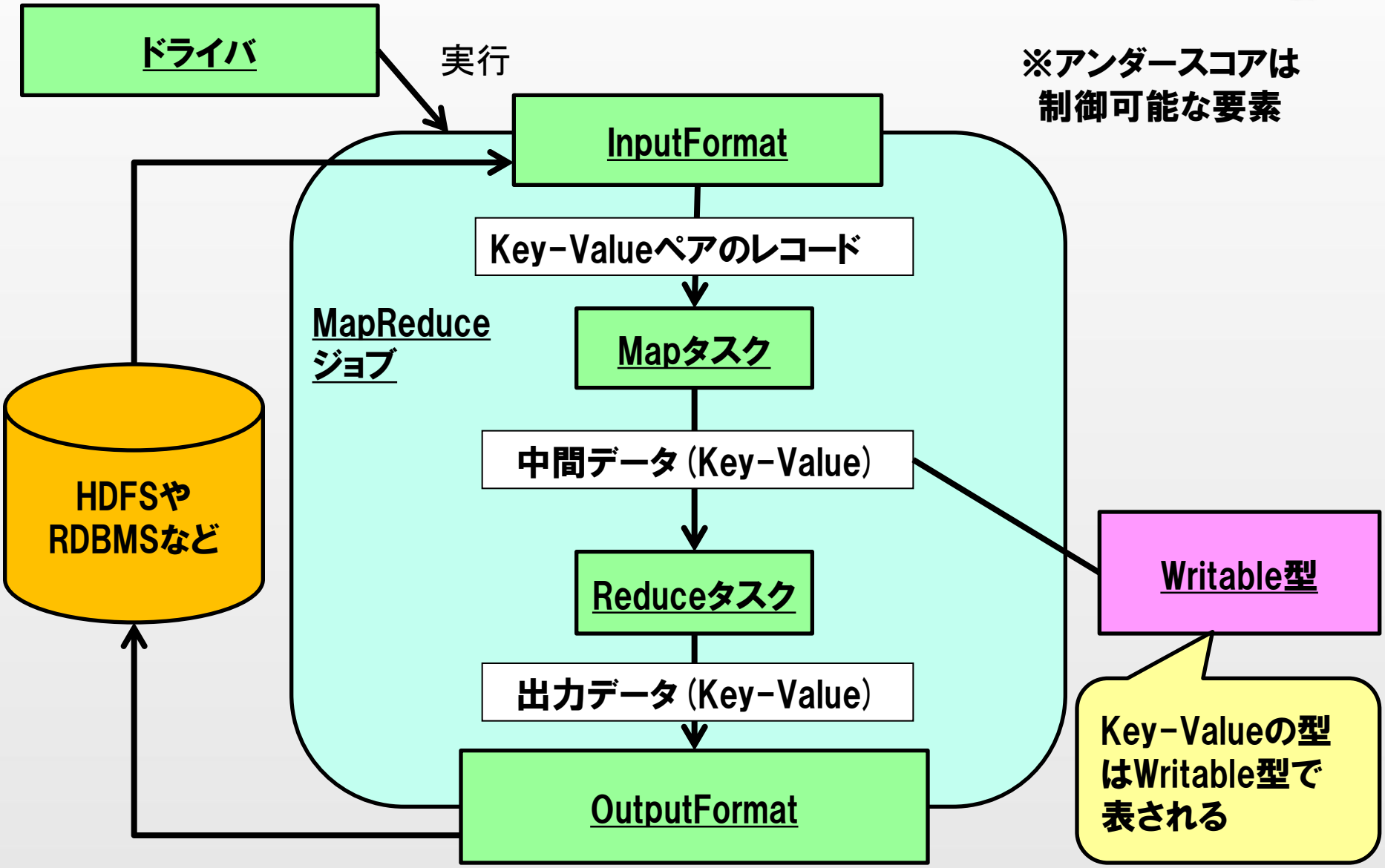
- setupメソッド
  - タスク開始時に1度だけフレームワークから呼び出される。主に初期化処理を実装するのに利用する

```
public void setup (Context context) throws IOException , InterruptedExcdeption
```

- cleanupメソッド
  - タスク終了時に1度だけフレームワークから呼び出される。主に中間生成物の削除などの清掃や、バッファのフラッシュなどに利用する

```
public void cleanup (Context context) throws IOException , InterruptedExcdeption
```

Writable型


ドライバ
実行
※アンダースコアは
制御可能な要素
InputFormat
Key-Valueペアのレコード
MapReduce
ジョブ
Mapタスク
中間データ (Key-Value)
HDFSや
RDBMSなど
Reduceタスク
Writable型
出力データ (Key-Value)
Key-Valueの型
はWritable型で
表される
OutputFormat

Writable型とは？

- Hadoop独自の型システム
- 独自のシリアライズ方式を実装可能
- ShuffleフェーズではMapタスクから出力された中間データはネットワーク経由でReduceタスクに渡されるため、**シリアライズ可能なデータ型**が必要

Q. Javaには標準でシリアライズの仕組みが用意されているが、改めてHadoopでWritable型を用意する理由は？

A. Javaで標準で提供されているシリアライズの仕組みは効率が悪い。例えばシリアライズ後のデータにクラス名が含まれているなど

※シリアライズ
オブジェクトをファイルやネットワークなどに書き出すことができるよう、構造を整列すること

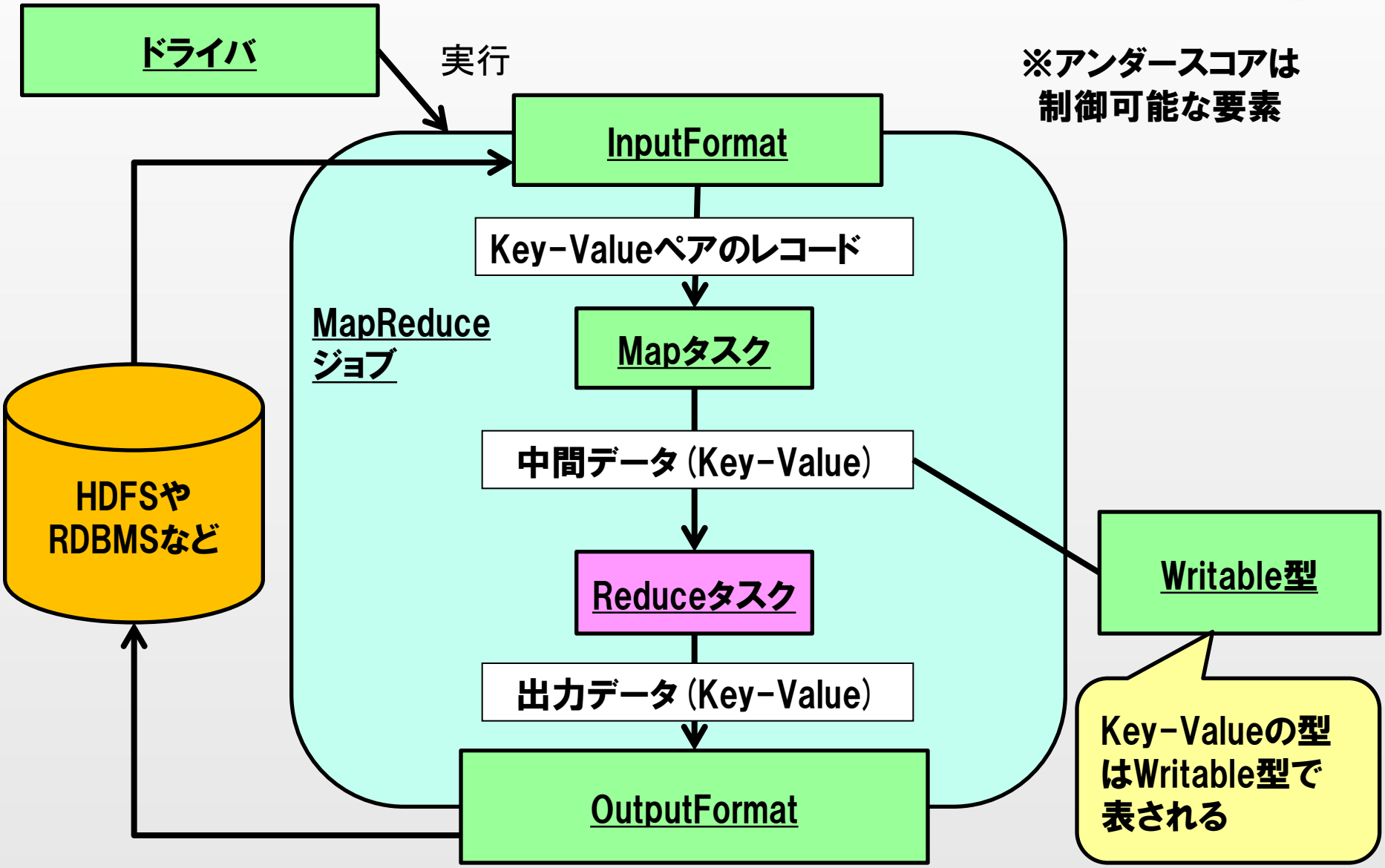
Writable型を表すAPI（Writable/WritableComparableクラス）

- Writable型は、WritableまたはWritableComparableクラスのサブクラスとして定義する。中間データのKeyの型に利用する場合は、Writableクラスを比較可能なように拡張したWritableComparableクラスのサブクラスとして定義する
- Hadoopでは標準で基本的なデータ型のWritable型を提供している。WritableComparableクラスのサブクラスのため、中間データのKeyにも利用可能である（中間データはフレームワークによりソートされるため、比較が必要）

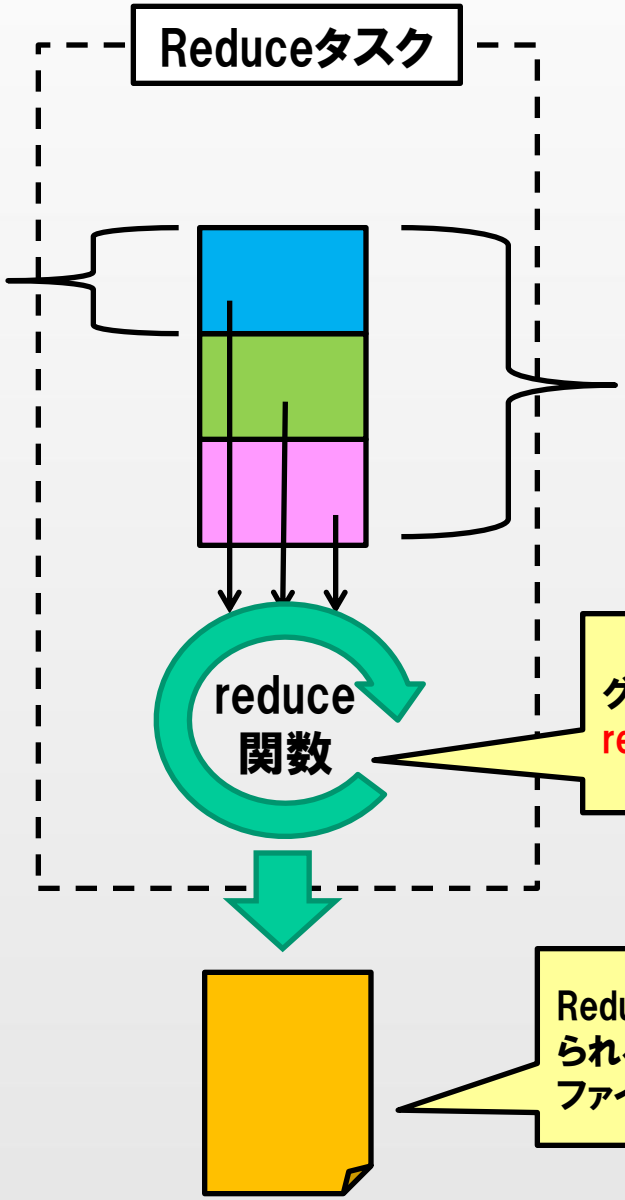
型	クラス
文字列	Text
バイト列	BytesWritable
整数	IntWritable
超整数	LongWritable
単精度浮動小数点数	FloatWritable
倍精度浮動小数点数	DoubleWritable
空（何も表示しない）	NullWritable

Hadoopが標準で提供するWritable型の一例

Reduceタスク


ドライバ
実行
※アンダースコアは
制御可能な要素
InputFormat
Key-Valueペアのレコード
MapReduce
ジョブ
Mapタスク
中間データ (Key-Value)
HDFSや
RDBMSなど
Reduceタスク
Writable型
出力データ (Key-Value)
Key-Valueの型
はWritable型で
表される
OutputFormat

Reduceタスクでの処理



グループ
（パーティションを特定の単位で区切ったKey-Valueペア）

パーティション
Shuffleフェーズから渡された中間データのKey-Valueペアの集合。
Reduceタスクへの入力

グループごとにreduce関数で処理する。
reduce関数への入力

Reduceタスクごとにアウトプットが得られる。例えばReduceタスクごとにファイルが生成される）

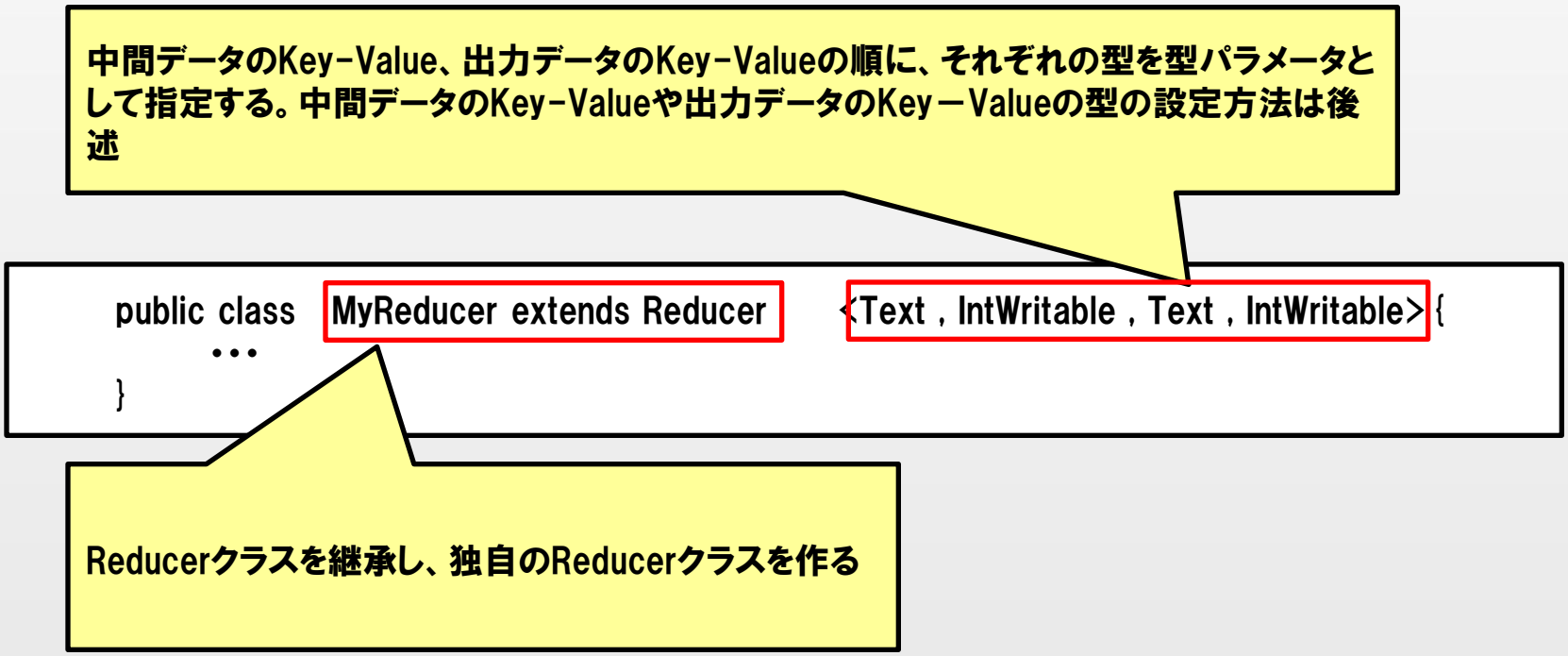
Reduceタスクを表すAPI（Reducerクラス）

- Reduceタスクでの処理とAPIの対応は次の通り

MapReduce	Hadoop MapReduce API
Reduceタスク	Reducerクラス
reduce関数	Reducer#reduceメソッド

Reduceタスクを表すAPI（Reducerクラス）

- ReduceタスクはReducerクラスで表される。Reducerクラスはジェネリッククラスとして定義されている。独自のReducerクラスは、これを継承して実装する



reduceメソッドの実装

- reduce関数はReducerクラスのreduceメソッドをオーバーライドして実装する
- フレームワークは、Reduceタスクに割り当てたパーティションから1グループずつ取り出し、reduceメソッドを呼び出して渡す

パーティションのKeyが渡される。
型には中間データのKeyの型を指定する（中間データのKeyの型は、Mapperクラスで設定した中間データのKeyの型と一致させる）

パーティションのKeyに紐づいたValueの反復可能なコレクションが渡される。
Iterableの型パラメータには中間データのValueの型を指定する（中間データのValueの型は、Mapperクラスで設定した中間データのValueの型と一致させる）

```
@Override
public void reduce ( Text key     , Iterable<IntWritable> values , Context context)
        throws IOException , InterruptedException {

        Iterator< iterator = values.Iterator ();
        ・・・処理・・・

        context.write ( outKey       ,     outValue);
```

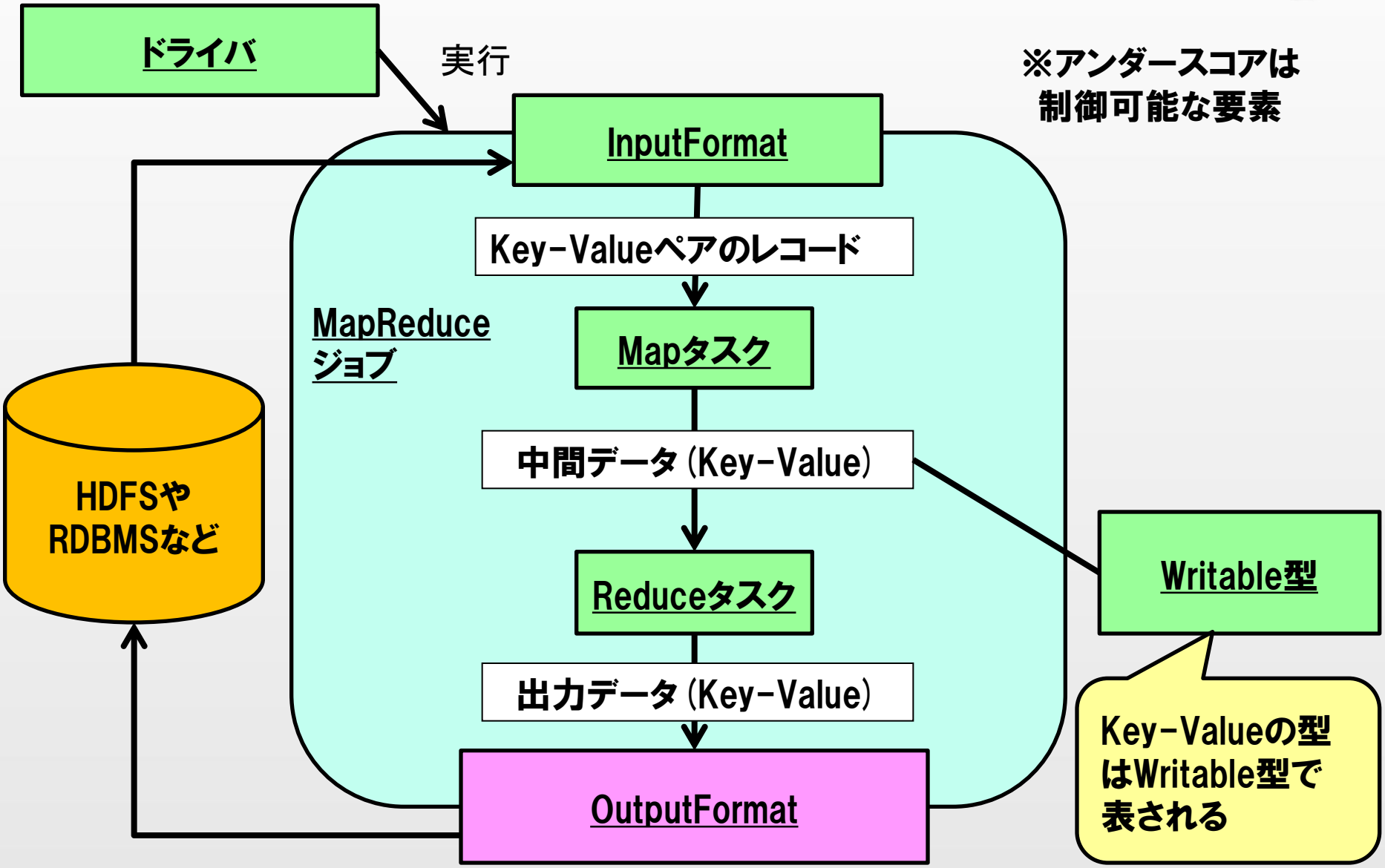
Contextオブジェクトのwriteメソッドで出力レコードを書き出す。引数はKey、Valueの順に指定する。出力するKeyとValueの型はクラスの型パラメータで指定した型パラメータと一致させる必要があることに注意

初期化/終了時処理の実装

- Mapperクラス同様、setupメソッドとcleanupメソッドをオーバーライドすることで、初期化/終了時処理を実装することができる

実装方法はMapperクラスの場合と同様なので、そちらを参照すること

OutputFormat



OutputFormatとは

- 出力データの各種属性を決定する

- データストア (HDFSやRDBMS)
  - 出力先はどこか
- データ形式の定義
  - 何に書き込むか (ファイル？ テーブル?)
  - 識別子 (ファイル名やテーブル名)
- Key-Valueの型
- Key-Valueの出力方法
  - KeyとValueがどのように書き出されるか (カンマ区切り、タブ区切りなど)
- Etc...

OutputFormatを表すAPI（OutputFormatクラス）

- 出力データフォーマットを表現するにはOutputFormatという種類のクラスを利用する。OutputFormatには様々な種類がある。

- TextOutputFormat
  - テキストファイルに書きだすためのOutputFormat
- DBOutputFormat
  - RDBMSのテーブルにレコードを挿入するためのOutputFormat
- Etc...

利用するOutputFormatを選ぶ際のポイント

① どこに書き込むか
② 何を書き込むか
③ Key-Valueの型は何か
④ Key-Valueをどのように書き出すか

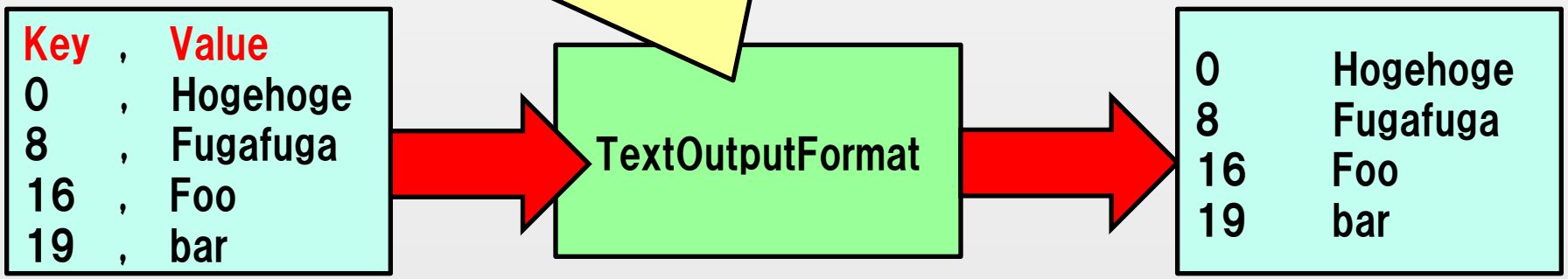
※InputFormatとは異なり、Key-Valueの型はユーザが決める。また書き出したKey-Valueのペアが出力レコードとなる

OutputFormatを表すAPI（OutputFormatクラス）

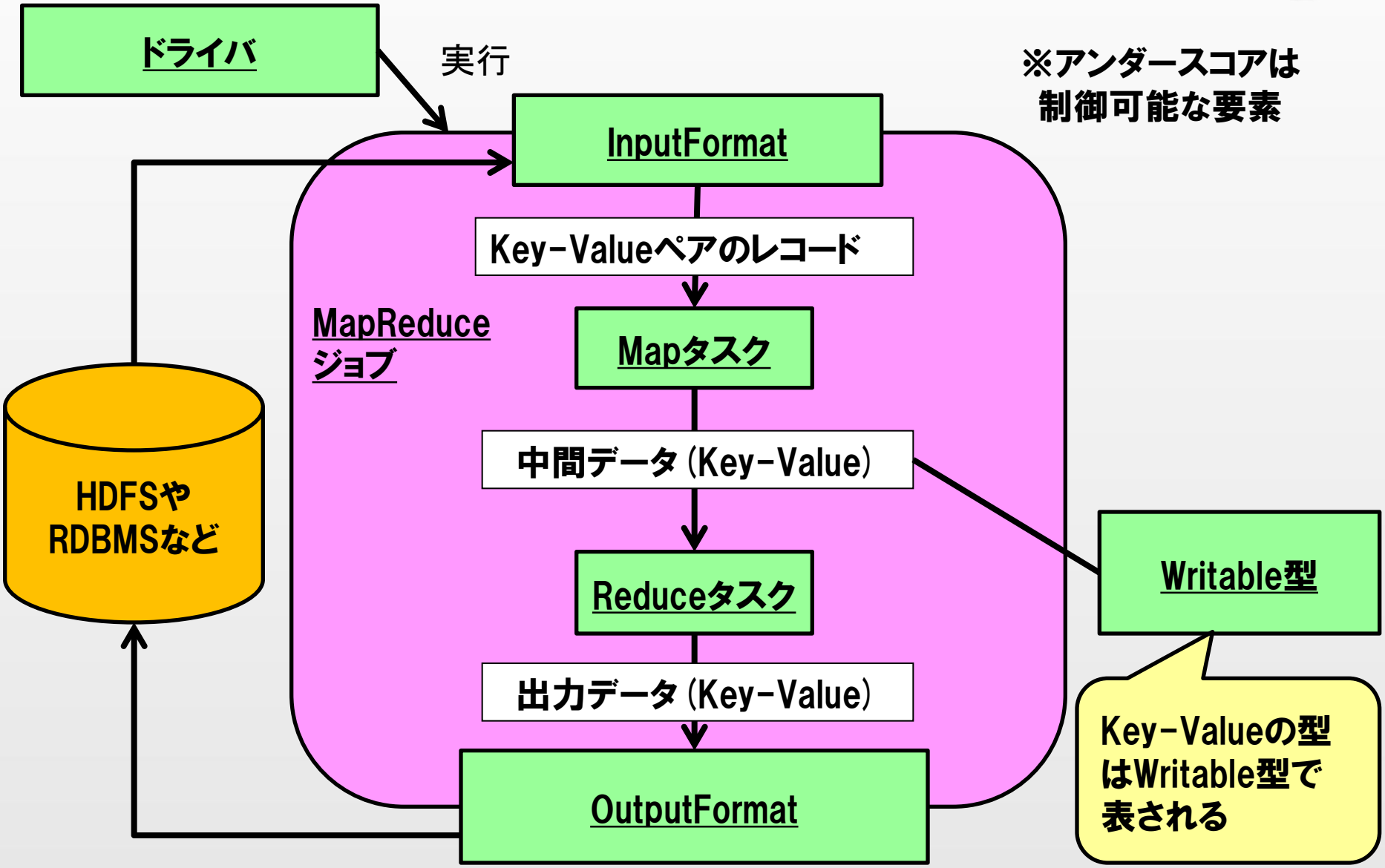
- データソース/Key-Valueの書き出し方（TextOutputFormatの場合）

- HDFSへ書き込む
- ファイルを作成する
- Key-Valueの型はユーザが定義する
- KeyとValueは「mapred.textoutputformat.separator」プロパティで表される文字で区切られ（デフォルトはタブ）、1レコードにつき1行に書き出される

※書き出す際のKey-Valueの型はJobオブジェクトに設定する（後述）



MapReduceジョブ



MapReduceジョブを表すAPI（Jobクラス）

- Jobクラスでジョブ全体の設定を行う
  - ジョブの名前
  - 実行Jarファイルの配布設定
  - 中間データのKey-Valueのデータ型
  - 出力データのKey-Valueのデータ型
  - ジョブが利用するMapper/Reducer
  - ジョブが利用するInputFormat/OutputFormat
  - Reduceタスクの数
  - その他設定
    - 利用するクラスに応じた個別の設定
      - Combinerの設定
      - Partitionerの設定
      - 独自SortComparatorの設定
      - 独自GroupingComparatorの設定
      - 独自プロパティの設定
      - データの圧縮に関する設定
      - etc

Mapタスクの数はInputFormatで決まったが、Reduceタスクの数は開発者がJobクラスに設定する

後の講義で扱います

Jobクラスの各種設定用メソッド

- Jobクラスには、各要素を設定するメソッドが用意されている

設定項目	設定メソッド
ジョブ名	setJobName ()
実行Jarファイルの配布方法	setJarByClass ()
Mapper	setMapperClass ()
Reducer	setReducerClass ()
中間データのKeyの型	setMapOutputKeyClass ()
中間データのValueの型	setMapOutputValueClass ()
出力データのKeyの型	setOutputKeyClasss ()
出力データのValueの型	setOutputValueClass ()
Reduceタスクの数	setNumReduceTasks ()

使用するクラスに応じた個別の設定

- 使用するクラスによっては個別に設定が必要な場合もある

例）FileInputFormatのサブクラスを利用する場合

- TextInputFomatやKeyValueTextInputFormatなどのFileInputFormat系のクラスは、FileInputFormat.addInputPathメソッドでHDFS上の入力パスを設定する必要がある

- 同様に、TextOutputFormatなども、FileOutputFormat.setOutputPathメソッドで出力パスを設定する必要がある

MapReduceジョブはどこで組み立てる？

- ジョブの設定方法を理解し、MapReduceジョブを組み立てる準備は整った
- どこでMapReduceジョブを組み立てるか？

① ドライバ（後述）内で組み立てる
  - 簡易的。単一のジョブのみ実行する場合など

② Jobクラスを継承した独自のJobクラスの内部で組み立てる
  - 複数のジョブを実行する場合や、ジョブをモジュール化したい場合

本講義では②の方法でMapReduceジョブを組み立てる

独自Jobクラスの実装

- Jobクラスを拡張した独自Jobクラスの例

```
public class MyJob extends Job {
    public MyJob (String jobName , String inputPath , String outputPath) {
        setJobName (jobName) ;

        setJarByClass (MyJob.class) ;

        setInputFormatClass (TextInputFormat.class) ;
        setOutputFormatClass (TextOutputFormat.class) ;

        setMapperClass (MyMapper.class) ;
        setReducerClass (MyReducer.class) ;

        setNumReduceTasks (50) ;

        setMapOutputKeyClass (Text.class) ;
        setMapOutputValueClass (IntWritable.class) ;

        setOutputKeyClass (Text.class) ;
        setOutputValueClass (IntWritable.class) ;

        FileInputFormat.addInputPath (this , new Path (inputPath) ) ;
        FileOutputFormat.setOutputPath (this , new Path (outputPath) ) ;
    }
}
```

ジョブ名の設定

Jarファイル配布の設定

InputFormat/OutputFormatの設定

Mapper/Reducerの設定

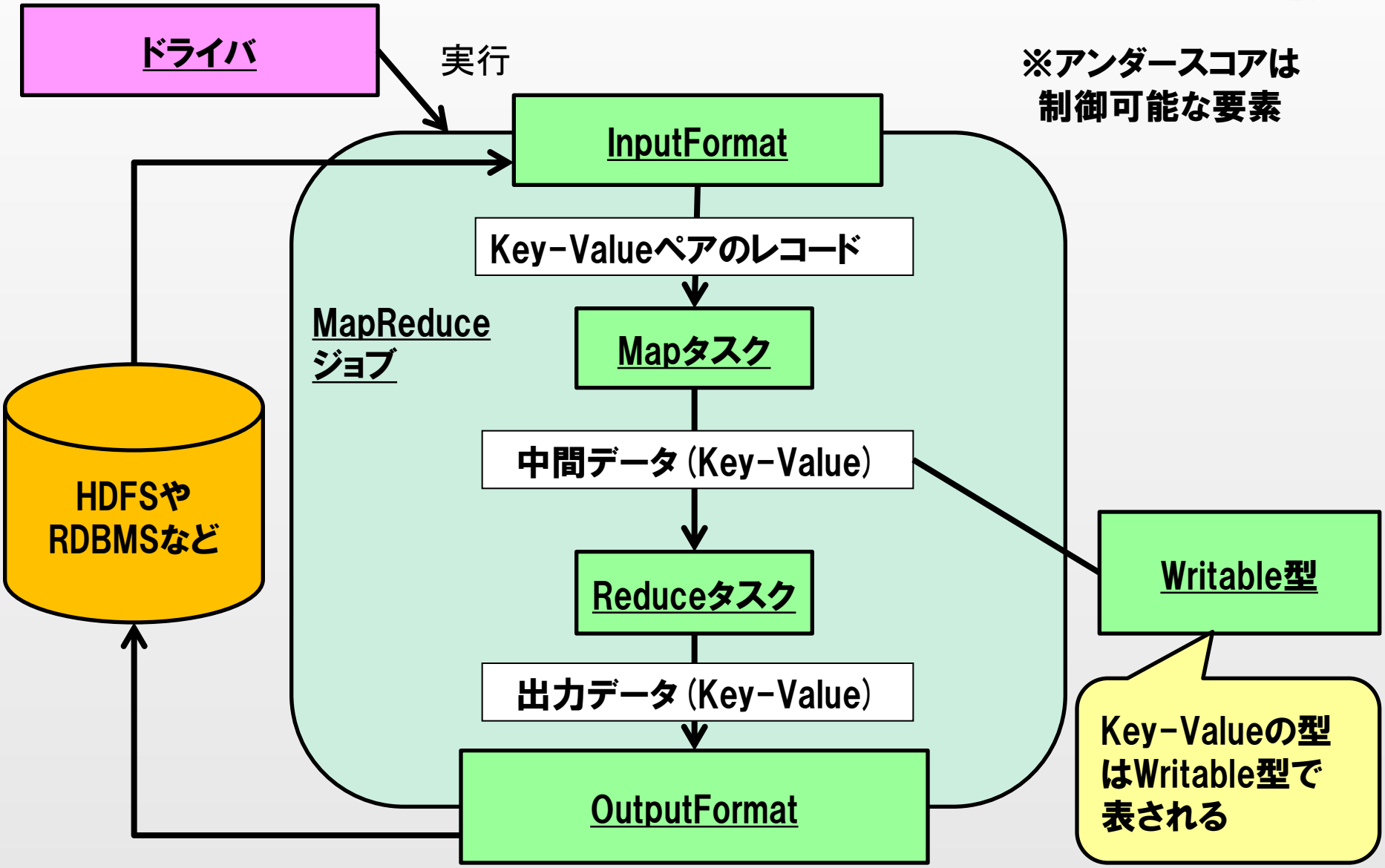
Reduceタスク数の設定

中間データのKey-Valueの型の設定

出力データのKey-Valueの型の設定

ファイルの入力/出力パスの設定

ドライバ


ドライバ
実行
※アンダースコアは
制御可能な要素
InputFormat
Key-Valueペアのレコード
MapReduce
ジョブ
Mapタスク
中間データ(Key-Value)
HDFSや
RDBMSなど
Reduceタスク
Writable型
出力データ(Key-Value)
OutputFormat
Key-Valueの型
はWritable型で
表される

ドライバの役割

- コマンドラインから与えられた引数をジョブに渡す
- コマンドラインから与えられたオプションを解釈する
- ジョブを起動する
- ジョブの起動順序を制御する

ドライバの実装

- ToolRunnerというクラスを利用して起動する
  - -Dプロパティを一時的に設定するためのオプション）や-conf（設定ファイルを一時的に変更する際のオプション） など、コマンドラインから与えられたオプションを解釈する
- ドライバクラスはToolRunnerが内部的に必要とするTool、Configuredというインターフェイス、クラスを実装する
- 独自JobオブジェクトのwaitForCompletionメソッドでジョブを起動する
- waitForCompletionからの戻り値などをもとに、後続ジョブの実行を制御する

```
public class MyJobDriver extends Configured implement Tool {

    @Override
    public int run (String [] args) throws Exception {
        Job myJob = new MyJob (“MyJob” , args [0] , args [1] ) ;
        return myJob.waitForCompletion (true) ? 0 : 1;
    }

    public static void main (String [] args) throws Exception {
        int returnCode = ToolRunner.run (new MyJobDriver , args) ;
        if (returnCode == 0) {
            •••
        }
    }
}
```

まとめ

本講義で学んだ内容

- Hadoop Mapreduceの構成要素
- MapReduceジョブの構成要素
- MapReduceジョブの構成要素を制御する基本的なAPI
  - ① InputFormat
  - ② Mapper
  - ③ Writable
  - ④ Reducer
  - ⑤ OutputFormat
  - ⑥ Job
  - ⑦ ドライバ