分散処理アプリ演習　第13回
HBase概要

（株）NTTデータ

最終日の演習全体スケジュール

- 1コマ目 HBase概要
  - HBase概要、データモデル、アーキテクチャ
- 2コマ目 HBaseスキーマ設計
  - スキーマ設計のポイント、TwitterログをHBaseに格納（演習）
- 3コマ目 HBase演習
  - Twitterログを用いた演習

講義内容

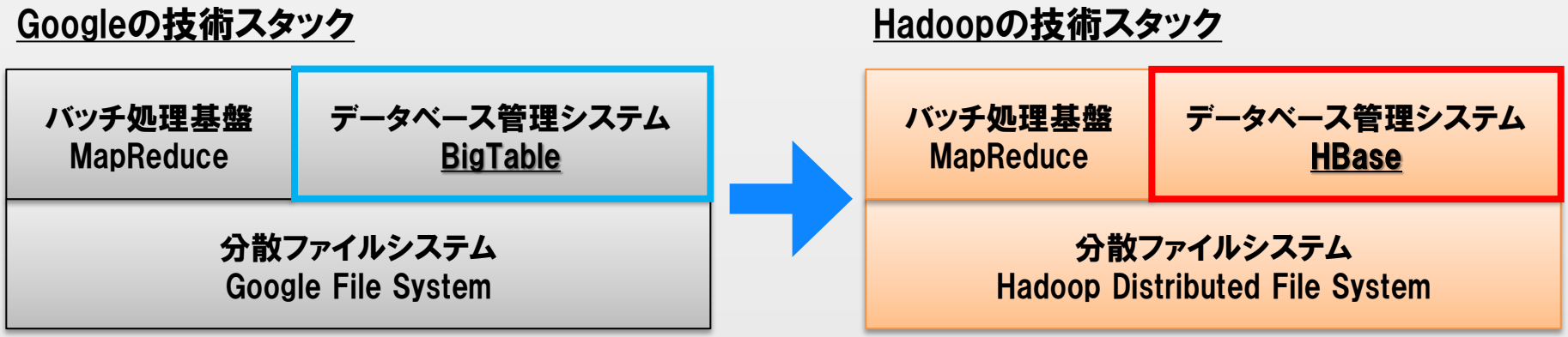
1. HBaseとは
   - 背景、RDBMSとの比較、Hadoopとの関係
2. HBase データモデル
   - データモデル、基本操作
3. HBase アーキテクチャ
   - アーキテクチャ、アクセスフロー、クラスタ構成、負荷分散

1. HBaseとは

HBaseとは

- HBaseは、分散型データベース管理システムである
- TBクラスのデータを扱うことができる

- Googleが開発した分散データベース管理システムであるBigTableのクローン
- HBaseはHDFSの上に構築されている



HBaseが開発された背景

- 従来より、業務データの格納先としてRDB(リレーショナルデータベース)が用いられてきた。
- RDBはデータの一貫性、可用性を重視した設計になっている。
  - サーバ台数を増やすことでスケールさせることは難しい。
  - スケールアップにも限界がある。
- クライアント数が多いWeb系システム等では、SQLの問い合わせ結果をキャッシュする目的でmemcachedをはじめとしたKeyValueストアが使われはじめる。
- Keyの値ごとにサーバを振り分けることで、データ(Value)を複数のサーバに分散して保存することが可能なことから、KeyValueストアを分散させるアプローチがでてきた。

- ✓ Google社が、BigTableと呼ばれる独自データストアを使用していることを論文で公表。
- ✓ その後、BigTableのクローンを目指して
  Apache OpenSource ProjectでHBaseが開発。

HBaseとRDBMSの比較・HBaseの特徴

- HBaseとRDBMSを比較する

	RDBMS	HBase
データモデル	行指向	列指向
トランザクション	可能	行単位のみ可能
データ操作	SQLを用いた複雑な処理が可能	単純なクエリのみ （put/get/scan/delete）
インデックス	任意のカラム	行キーのみ
スケーラビリティ	なし （基本的には単一ノード）	あり （スレーブノードの追加）
対象データサイズ	数百GBまで	数百TBも可能

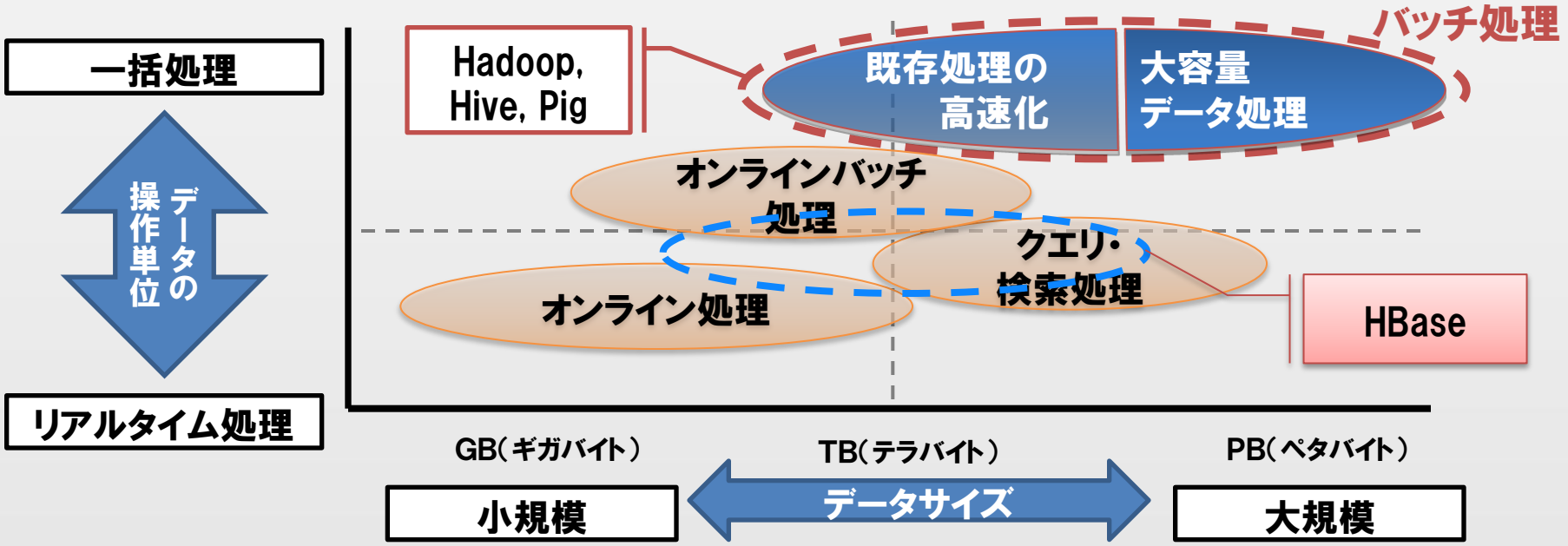
- また、HBaseには以下の特徴もある
  - 自動的にデータが分散配置され、負荷分散が行われる
  - テーブル構造が柔軟

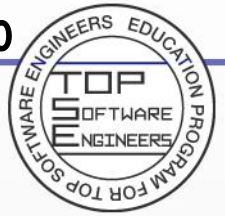
Hadoopとの関係

- HBaseはHDFS上に構築される
  - HDFSの特徴である「可用性」、「大容量」の利益を得ている
  - HDFSのランダム読込・書込ができない欠点を補っている
    - ただし、全データをスキャンするような場合は、HDFSの方が効率が良い
    - HBaseはランダムアクセス、ショートレンジスキャンが得意
    - HDFS：シーケンシャルアクセスが得意→バッチ処理向き
    - HBase：ランダム・アクセスが可能→オンライン処理向き
      - HBaseは、特に「大量の細かい書き込み」が得意
- MapReduceとの依存関係はない
  - map、reduceの入出力はKeyValueデータであるため、入出力データや中間データの保存先としての相性はよい
  - HadoopのMapReduceジョブのデータ入出力先として、HBaseを利用するAPIが提供されている

HadoopエコシステムでのHBaseの位置づけ

- HBaseは、Hadoop (MapReduce, Hive, Pig) とは以下の点で異なる
  - 扱うデータ規模は、数十GB～TBオーダである
    - 数十GB以下も扱えるが、RDBMSの方が利便性が良い
    - 現在のHBase実装では、データの管理方法やHBaseクラスタ内通信の影響により Hadoop (HDFS) と比べると扱えるデータ規模は小さい
  - 小さいデータ操作単位を扱う
    - Hadoop自体は、TB以上の大規模データを一括して処理することが出来るのがポイントであるが、HBaseは一括処理には向いていない





2. HBase データモデル

HBaseのデータモデル（クラスタ～テーブル）

- 1つのHBaseクラスタには単一のデータベースが存在する
- データベースには複数のテーブルが保存できる
  - 索引、制約、ビューなど、テーブル以外のオブジェクトは存在しない
- テーブルの各行は行キーで識別され、また行キーで昇順にソートされる
  - 行キーは、データ書き込み時にユーザが指定する任意のバイト列

HBaseクラスタ=データベース

tweet テーブル

行キー	tweet	user_id
1234	ほげほげ #hoge	24
1235	テストです #test	1256
1236	サンプル文字	12
1237	なにかつぶやき #test	862
1238	あああああああ	516

行キー

user テーブル

行キー	name	followers_count
24	A	10
28	B	2314
295	C	18
31	D	241
67	E	3

HBaseのデータモデル（テーブル～セル）

- テーブルは複数の列ファミリで構成される。（例：tweet,user_id）
- 列ファミリは複数の列で構成される。各列は修飾子で分類され、修飾子はデータの書き込み時に自由に追加することができる（例：text,hashtag）
- 列は（列ファミリ名、修飾子）の組み合わせで指定する（例： tweet:text->サンプル文字）
- 行と列の交差する所をセルと呼ぶ
- 行キー、列ファミリ名、修飾子、セルの値とも、任意長のバイト列が格納できる

tweet テーブル

行キー	tweet		user_id
	text	hashtag	
1234	ほげほげ #hoge	#hoge	24
1235	テストです #test	#test	1256
1236	サンプル文字		12
1237	なにかつぶやき #test	#test	862
1238	あああああああ		516

HBaseのデータモデル（セル～値）

- セルの値は、処理時間のタイムスタンプによって、セル単位のバージョンが付けられる
- デフォルトでは最新の値が取得されるが、タイムスタンプを指定して、古いバージョンのデータを読み込むことができる
- テーブル作成時に指定した、列ファミリごとの最大バージョン数（デフォルト=3）よりも古いバージョンは消滅する
  - 各データは複数のノードに分散して配置されるが、HBaseではデータをタイムスタンプで管理するため、各ノードの時間があっていることが重要。

user テーブル

行キー	タイムスタンプ		name	followers_count
28	17:21:50,593	新		<write> 2
	14:39:18:037		<write>hoogee	
	09:52:28:707			<write> 1
	05:09:44:081			<write> 0
	02:03:24:141	古	<write>hogehoge	

HBaseのデータモデル（データファイル）

- HDFS上に作成されるデータファイルについて
  - <Key、Value>で保存されている
    - 【Key】: *行キー*+ *列ファミリ名:修飾子*+ *バージョン(タイムスタンプ)*+ *操作名*+ *valueのbyte配列サイズ*
    - 【Value】: *セルの値*
    - name ファミリのデータファイルの内容

Key　Value

```
K: 28/name:/1326814999738/Put/vlen=6 V: hoogee
K: 28/name:/1326814921689/Put/vlen=8 V: hogehoge
```

    - followers_count ファミリのデータファイルの内容

```
K: 28/followers_count:/1326815003716/Put/vlen=1 V: 2
K: 28/followers_count:/1326814983535/Put/vlen=1 V: 1
K: 28/followers_count:/1326814977298/Put/vlen=1 V: 0
```

- 列ファミリごとに別のファイルで管理される
- 列ファミリはスキーマに保存されるが、修飾子は行のデータとして保存される

HBase の基本操作（事前準備）

- 事前準備
  - hdclient01サーバにVNCでログイン
  - コンソールの起動

```
[root@hdclient01 ~]#
```

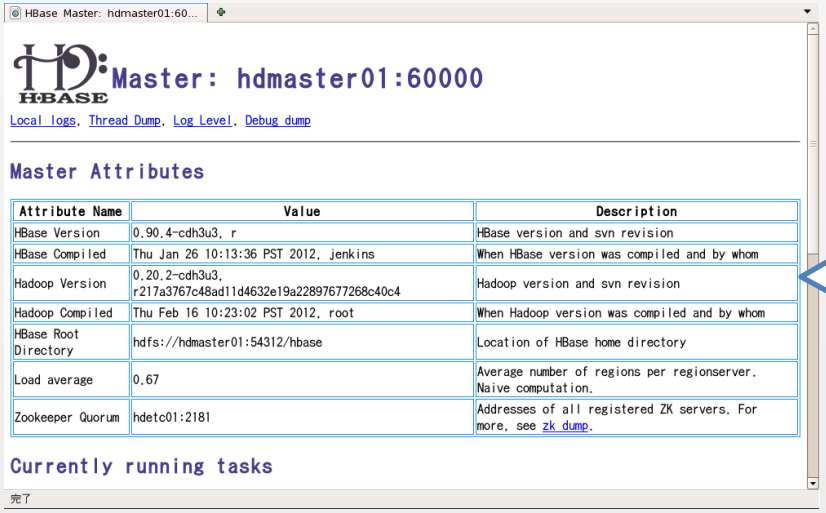
  - HBaseの起動

```
[root@hdclient01 ~]# sh /root/shell/hbase_operation/hbase_start.sh
```

【参考】HBase WebUI

- Hadoop同様、HBaseにもWebUIがあります。
  - アクセスURI

```
http://hdmaster01:60010/
```



- リージョンサーバの状態
- テーブルの状態
  →コンパクション、スプリットの実行
- ログやスレッドダンプの確認

Region Servers

	Address	Start Code	Load
	hdslave01:60030	1334032842961 hdslave01,60020,1334032842961	requests=0, regions=8, usedHeap=23, maxHeap=998
	hdslave02:60030	1334032842961 hdslave02,60020,1334032842961	requests=0, regions=8, usedHeap=23, maxHeap=998
	hdslave03:60030	1334032842961 hdslave03,60020,1334032842961	requests=0, regions=8, usedHeap=23, maxHeap=998
Total:	servers: 3		requests=0, regions=8

Region Serversが
3台あることを確認

HBase の基本操作（HBase shell起動）

- HBase shellの実行

```
$ /usr/bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.4-cdh3u3, r, Thu Jan 26 10:13:36 PST 2012

hbase(main):001:0>
```

- コマンドリスト
  - help
    - shellコマンドのリストアップ
  - status
    - クラスタについての基本状況を表示
  - list
    - テーブルをリストアップ
  - describe '<table name>'
    - テーブル構造を表示
  - exit
    - HBase shellを終了

［補足］
hbase shellはRubyのirbを改造したものなので、
$ hbase shell foobar.rb
のように、rubyスクリプトとして操作をまとめて記述しておいて実行することも可能

HBase の基本操作（create）

- テーブルを作成
  - create '<テーブル名>',
    {NAME => '<列ファミリ名>'} [, <option>] } [, {...}]
- 例

```
hbase(main):001:0> create 'table1', {NAME => 'colfamily1'}, {NAME =>
'colfamily2'}
```

※簡略化した指定も可能

```
hbase(main):002:0> create 'table1', 'colfamily1', 'colfamily2'
```

- また、列ファミリの定義には、主に以下のパラメータが指定可能

パラメータ	説明
NAME	列ファミリの名前
VERSIONS	保持するバージョン数、デフォルト=3
BLOCKSIZE	HFileのブロックのサイズ、デフォルト=64KB
BLOCKCACHE	HFileのブロックキャッシュを使うか否か、デフォルト=true
COMPRESSION	HFileの圧縮形式、LZO, GZ, NONEから選択、デフォルト=NONE

```
hbase(main):003:0> create 'table1', {NAME => 'colfamily1', VERSIONS
=> 1}
```

HBase の基本操作演習（create）

- テーブルを作成

テーブル名：testTable

行キー	colfamily1

```
hbase(main):001:0> create 'testTable', 'colfamily1'
0 row(s) in 4.4530 seconds

hbase(main):002:0> list
TABLE
testTable

1 row(s) in 0.0360 seconds

hbase(main):003:0> describe 'testTable'
```

HBase の基本操作（put/get）

- データの格納/取得
  - put '<テーブル名>', '<行キー>',
    '<列ファミリー名>:<修飾子>', '<値>' [,タイムスタンプ]
    - column
  - get '<テーブル名>', '<行キー>' [, options]

パラメータ	説明
COLUMN	列を指定(列ファミリ:修飾子)
TIMESTAMP	バージョンを指定
TIMERANGE	バージョン範囲を指定
VERSIONS	表示するバージョン数、デフォルト=1(最新のみ)

- 例

```
hbase(main):004:0> put 'table1', 'row1', 'colfamily1:qualifier1', 'value1'

hbase(main):005:0> get 'table1', 'row1'
hbase(main):006:0> get 'table1', 'row1', { COLUMN =>
'colfamily1:qualifier1' ,TIMERANGE => [timestamp1, timestamp2] }
```

HBase の基本操作（scan）

データの検索

- scan '<テーブル名>' [, options]

パラメータ	説明
COLUMNS	列を指定（列ファミリ:修飾子） ※列ファミリ全てを取得する場合は、列ファミリ:（空欄）にする。
TIMESTAMP	バージョンを指定
TIMERANGE	バージョン範囲を指定
LIMIT	検索件数を指定
STARTROW	指定した行キー以上を読み込み
STOPROW	指定した行キー未満を読み込み

例

```
hbase(main):007:0> scan 'table1'
hbase(main):008:0> scan 'table1', { COLUMNS =>
'colfamily1:qualifier1', LIMIT => 1,  STARTROW => 'row1' }
```

HBase の基本操作演習（put/get/scan）

- **データ追加、確認**

テーブル名：testTable

行キー	colfamily1	
		qua1
a1	msg	hoge

```
hbase(main):004:0> put 'testTable','a1','colfamily1:','msg'
hbase(main):005:0> put 'testTable','a1','colfamily1:qua1','hoge'
hbase(main):006:0> get 'testTable','a1'
COLUMN                                CELL
 colfamily1:                          timestamp=1327058699768, value=msg
 colfamily1:qua1                      timestamp=1327059422927, value=hoge
1 row(s) in 0.0190 seconds

hbase(main):007:0> scan 'testTable'
ROW                                   COLUMN+CELL
 a1                                   column=colfamily1:,
                                        timestamp=1327058699768, value=msg
 a1                                   column=colfamily1:qua1,
                                        timestamp=1327058888155, value=hoge
1row(s) in 0.0780 seconds
```

HBase の基本操作（delete）

- データの削除
  - delete '<テーブル名>', '<行キー>',
    '<列ファミリー名>:<修飾子>' [,タイムスタンプ]

特定セルの削除

column

  - deleteall '<テーブル名>', '<行キー>'
    [,'<列ファミリー名>:<修飾子>',タイムスタンプ]

特定行の全セルの削除

- 例

```
hbase(main):009:0> delete 'table1', 'row1', 'colfamily1:qualifier1'

hbase(main):010:0> deleteall 'table1', 'row1'
```

HBase の基本操作（disable/enable,drop,truncate）

- テーブルの使用可否の切り替え
  - disable '<テーブル名>'
  - enable '<テーブル名>'
- 例

```
hbase(main):011:0> disable 'table1'

hbase(main):012:0> enable 'table1'
```

- テーブルの削除
  - drop '<テーブル名>'
    - dropするには、事前にdisable（使用不可）にしておく必要がある。
- テーブルの再構成
  - truncate '<テーブル名>'
    - 内部的には、disable -> drop -> createが行われる
- 例

```
hbase(main):013:0> drop 'table1'

hbase(main):014:0> truncate 'table1'
```

HBase の基本操作（alter）

- テーブル定義を変更
  - 列ファミリの追加
    - alter '<テーブル名>' , { NAME => '<列ファミリ名>' [, options] }
  - 列ファミリの削除
    - alter '<テーブル名>' , { NAME => '<列ファミリ名>', METHOD => 'delete' }
    - alterの実行には、事前にdisable(使用不可)にしておく必要がある。
- 例

```
hbase(main):015:0> alter 'table1', { NAME => 'colfamily2' }

hbase(main):016:0> alter 'table1', { NAME =>'colfamily2', METHOD =>
'delete' }
```

HBase の基本操作演習（alter）

- **列ファミリ追加**

  テーブル名：testTable

行キー	colfamily1		colfamily1
		qua1	
a1	msg	hoge	

```
hbase(main):008:0> disable 'testTable'
hbase(main):009:0> alter 'testTable',{NAME=>'colfamily2'}
hbase(main):010:0> enable 'testTable'
hbase(main):011:0> describe 'testTable'

  {NAME => 'colfamily2',…が追加されていること
```

- **修飾子の追加は、HBaseの停止なしに行えるが、列ファミリの追加は、テーブルの停止を伴う。**

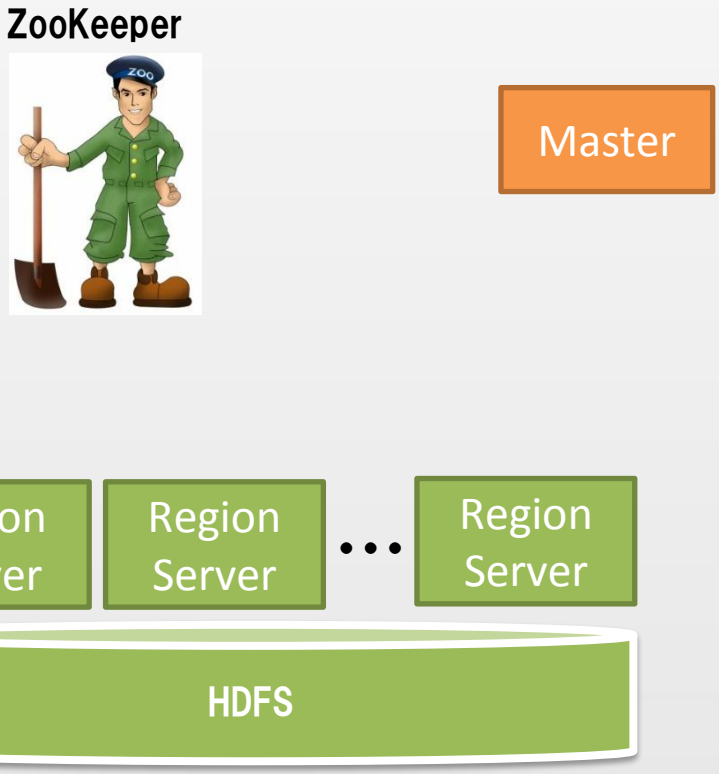
HBase の基本操作（その他）

- その他のコマンド
  - flush　：テーブルのフラッシュを行う
  - major_compact　：テーブル・リージョンのMajorコンパクションを実行する
  - split　：テーブル・リージョンのスプリットを行う

3. HBase アーキテクチャ

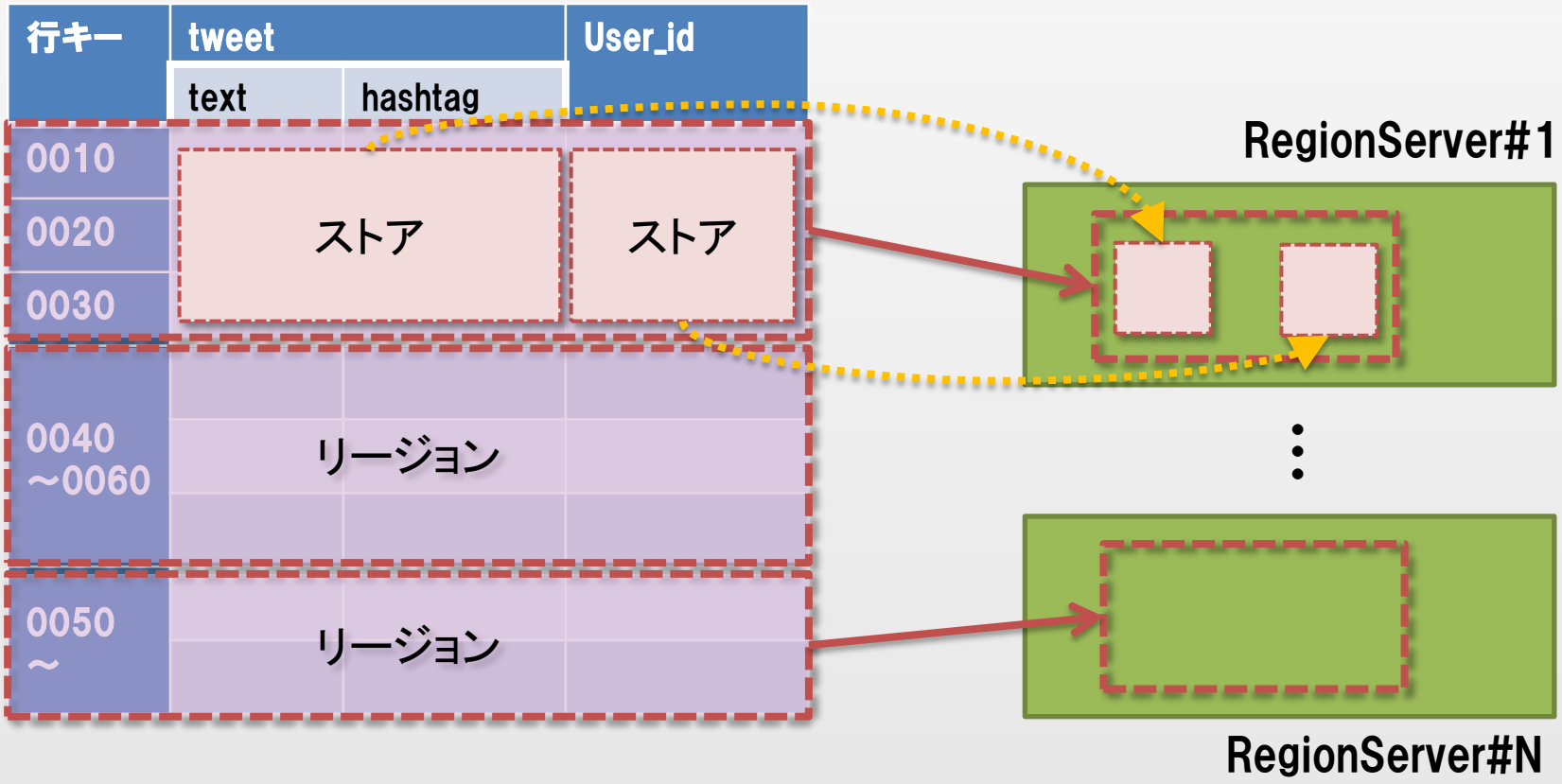
HBase クラスタ構成

- HBaseクラスタは、次のノードで構成される
  - RegionServer
  - ZooKeeper
  - Master



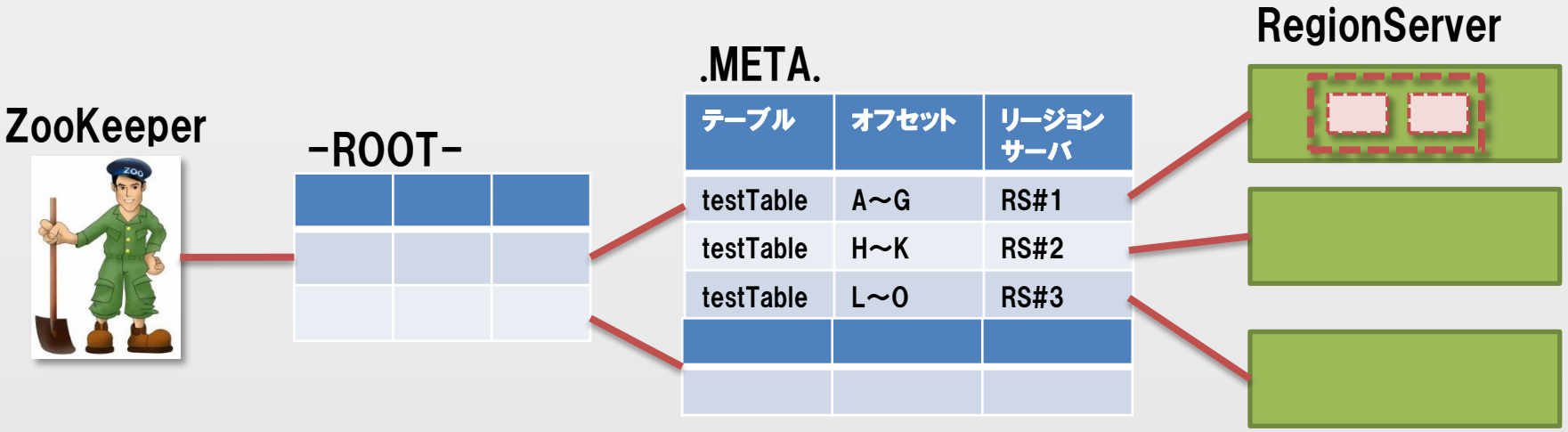
HBaseのデータ配置（RegionServerの役割）

- テーブルは、テーブルを構成する行キーにより、リージョンに分割される
- RegionServerには複数のリージョンが割り当てられ、データの読み書きを担当する
- リージョン中のデータは列ファミリごとに、まとめて保存される。この単位をストアと呼ぶ



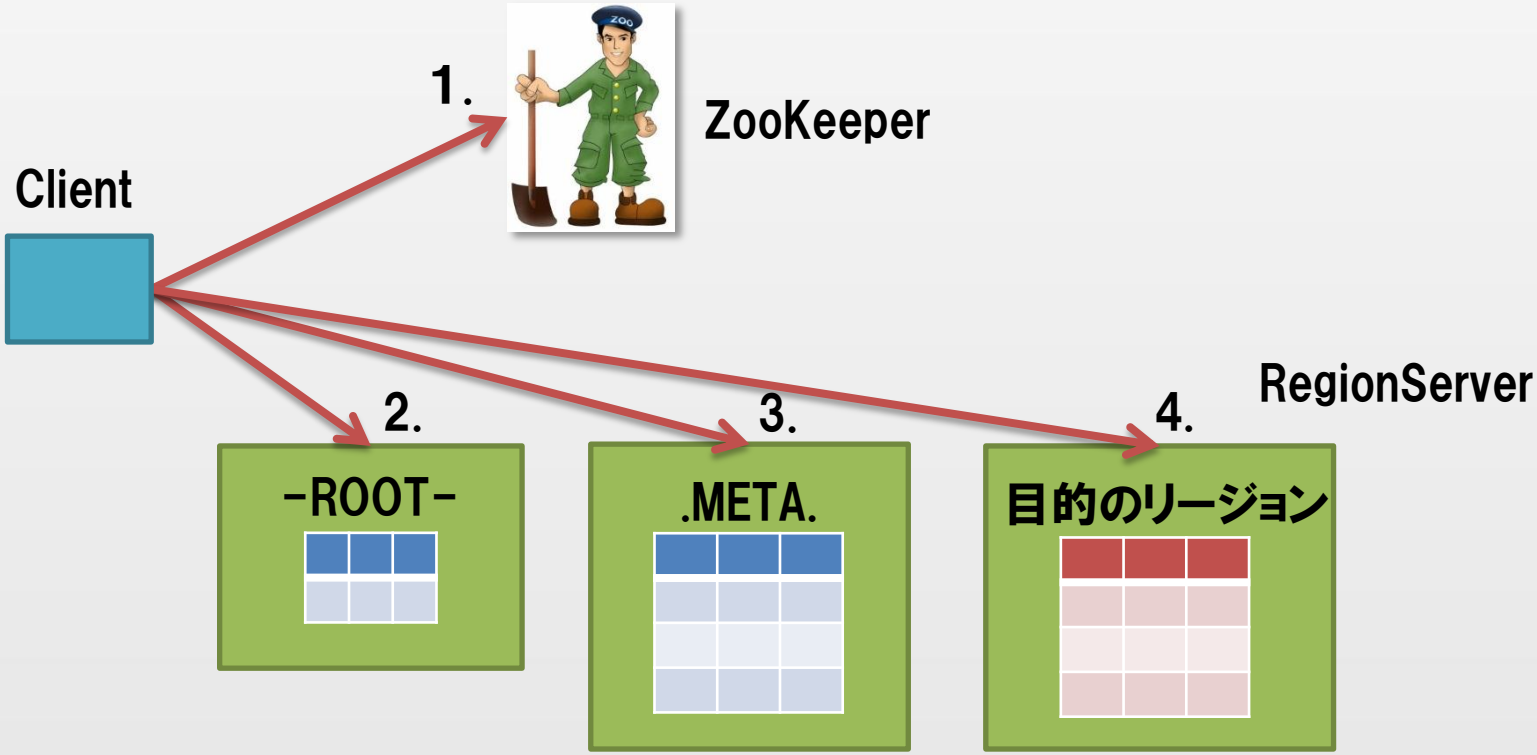
HBase のデータ管理（ZooKeeperの役割）

- RegionServer上に分散されたデータへのアクセスを単純化するため、メタデータはツリー構造になっている。
- Clientは、メタデータツリーのルートであるZooKeeperにアクセスし、ツリーをたどることで、目的のデータにアクセスすることができる。
  - ZooKeeper
    - -ROOT-テーブルを担当しているRegionServerを保存
  - -ROOT-テーブル
    - .META.テーブルを担当しているRegionServerを保存
  - .META.テーブル
    - どのテーブルの、どのデータを、どのRegionServerが担当しているかを保存



クライアントからのアクセスフロー

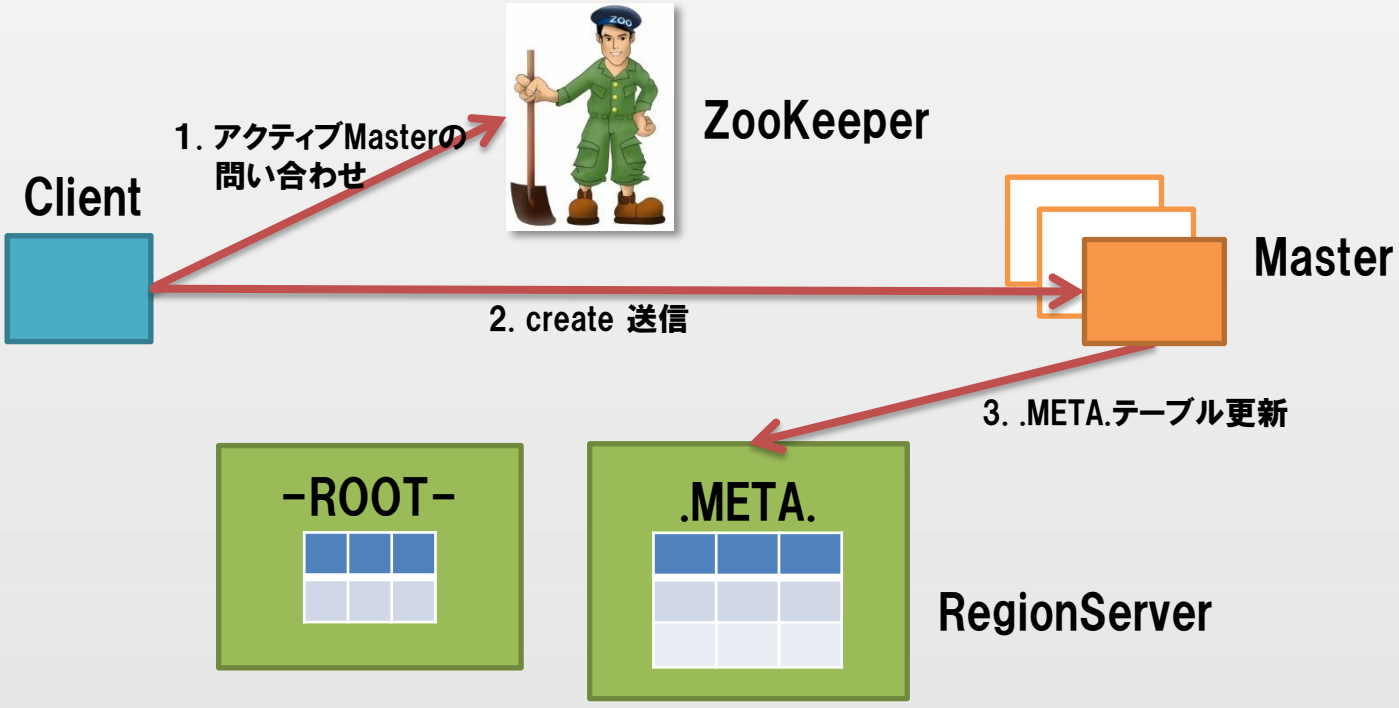
- クライアントが、リージョンにアクセスするまでのフローは次の通り
  1. ZooKeeperから、-ROOT-テーブルを保持するRegionServerを取得
  2. -ROOT-テーブルにアクセスし、.META.テーブルを保持するRegionServerを取得
  3. .META.テーブルにアクセスし、目的のデータを保持するRegionServerを取得
  4. 目的のリージョンにアクセス



HBase のRegion管理（Masterの役割）

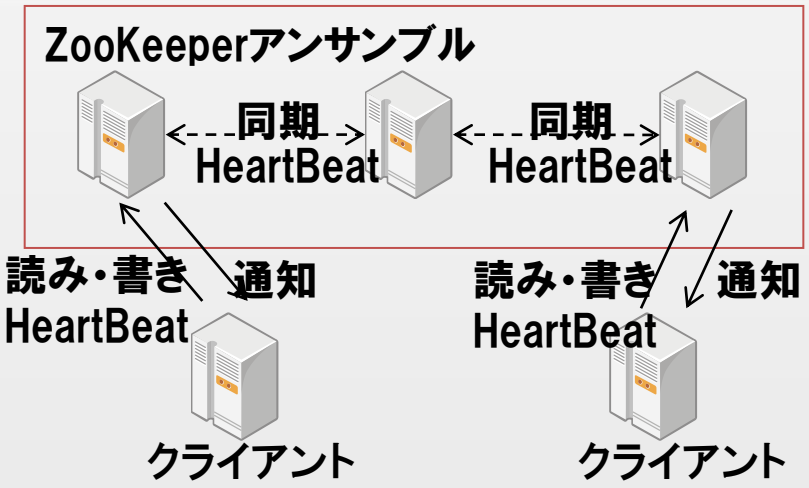
- Master ノードによって、リージョンが調整・管理される
  - 新規リージョンのRegionServerへの振り分け
  - 特定のRegionServerにリージョンが偏らないようにバランスする
- また、Masterノードは複数台でホットスタンバイ構成にできる
  - ZooKeeperにより、アクティブなMasterノードが管理される

- テーブル追加時の処理フロー



【参考】ZooKeeperアンサンブル

- ZooKeeperクラスタは複数のZooKeeperサーバで構成される。これをアンサンブルと呼ぶ
- アンサンブル内で、データがレプリケートされる
- ZooKeeperは、アンサンブルの半数未満のZooKeeperサーバが故障しても、データが失われず、サービスが停止しないことを保証する
- クライアントはアンサンブル中のサーバのリストを保持し、いずれかのサーバに接続する

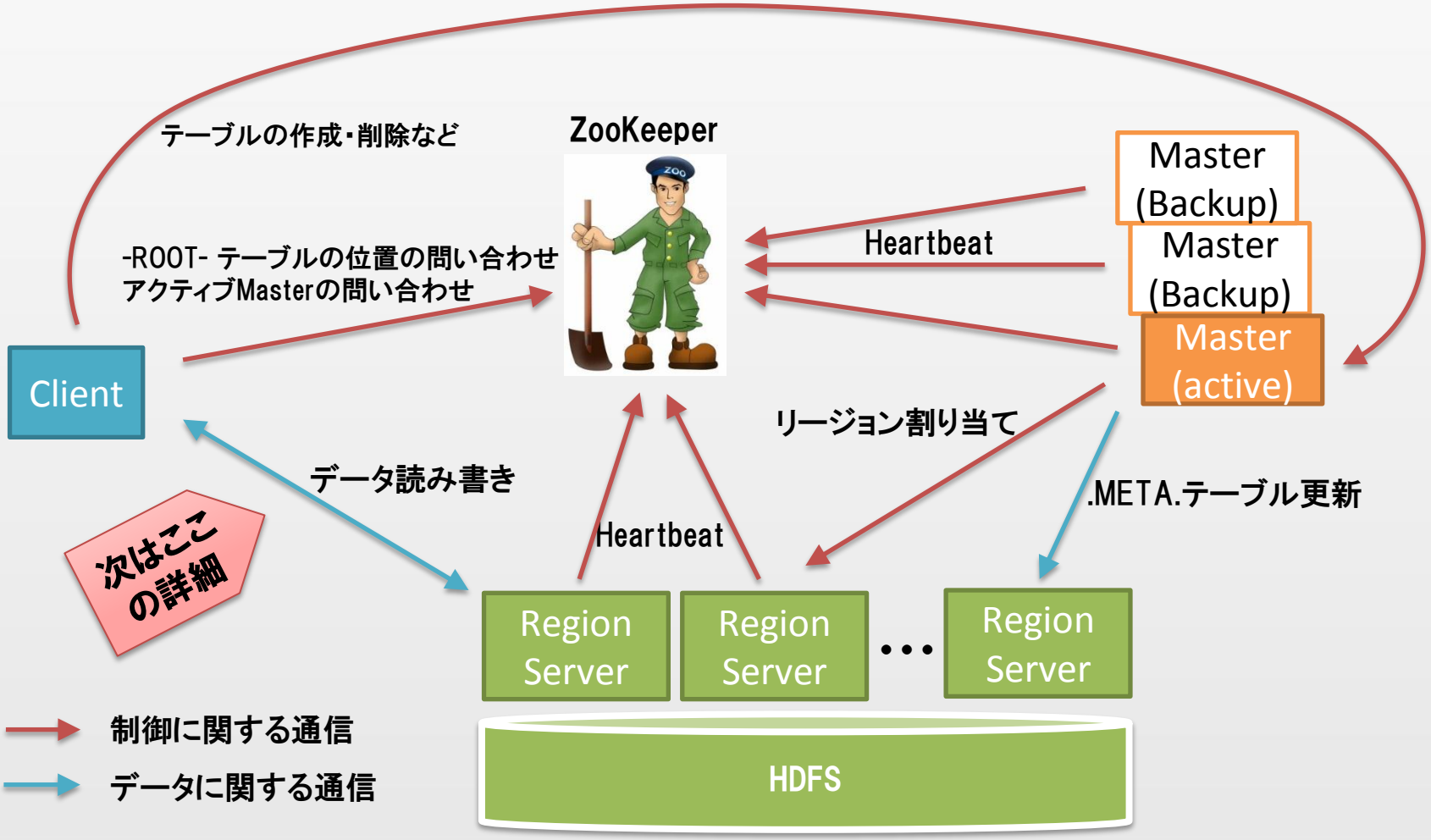


HBase クラスタ構成における役割

HBaseクラスタを構成するノード・クラスタ	
Masterノード	■ HBaseのマスターノード ■ データの管理単位であるリージョンをRegionServerに割り当てる ■ 複数台でホットスタンバイ構成にできる
RegionServerノード	■ HBaseのスレーブノード ■ データの読み書きを管理する
ZooKeeperクラスタ	■ 複数サーバの協調動作支援システム ■ アクティブなMasterノードの位置の保持、ROOTリージョンの位置の保持、RegionServerノードの死活監視などを行う

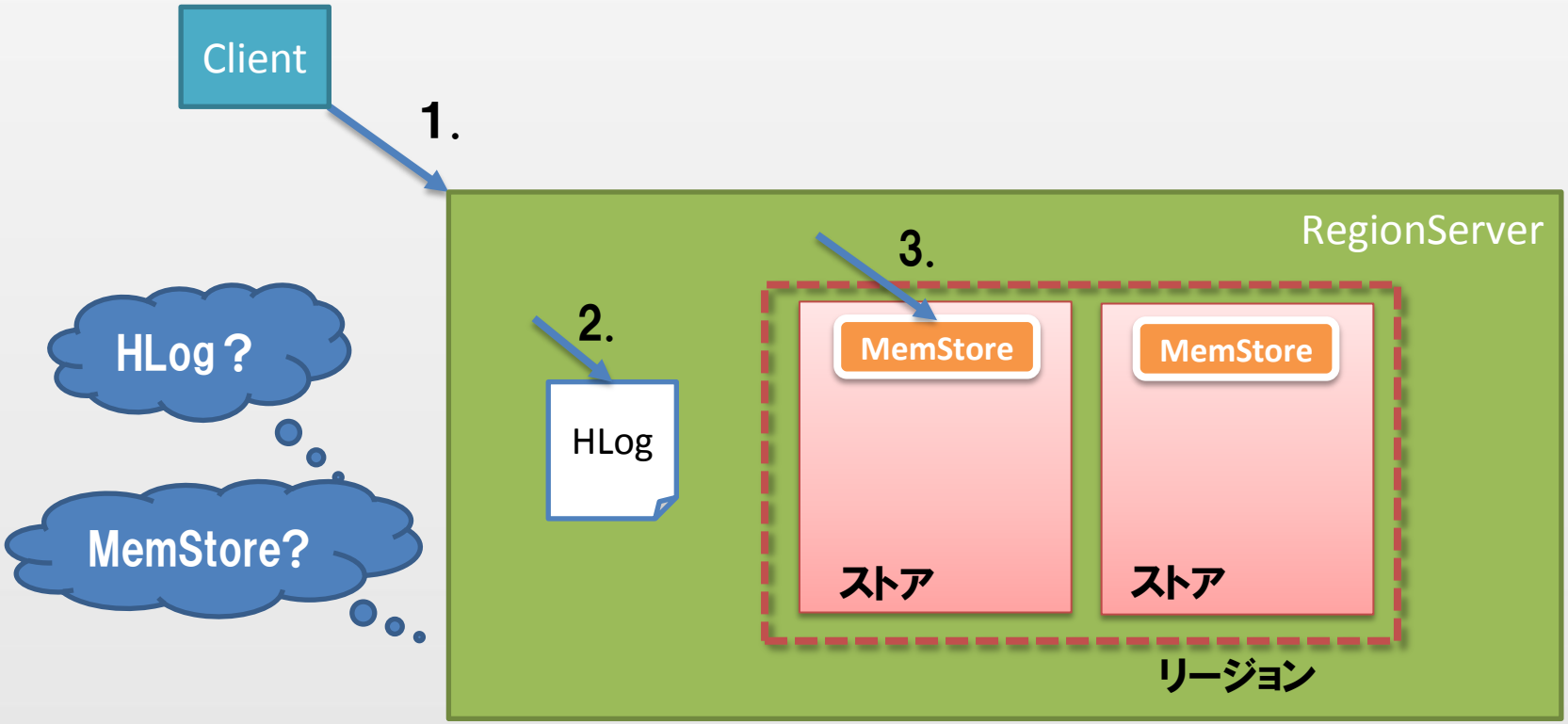
HBaseクラスタ間の通信

- HBaseクラスタでの各ノード間の主な通信は、以下の通りである。



データアクセス詳細（書込）

- データの書き込みフローは以下のようになる
  - クライアントで書き込み処理（put/delete）を実行
    1. RegionServerに書き込みデータ（Key-Value）が送られる
    2. HLogに更新情報を追記する
    3. Key-Valueに対応したMemStoreに書きこむ



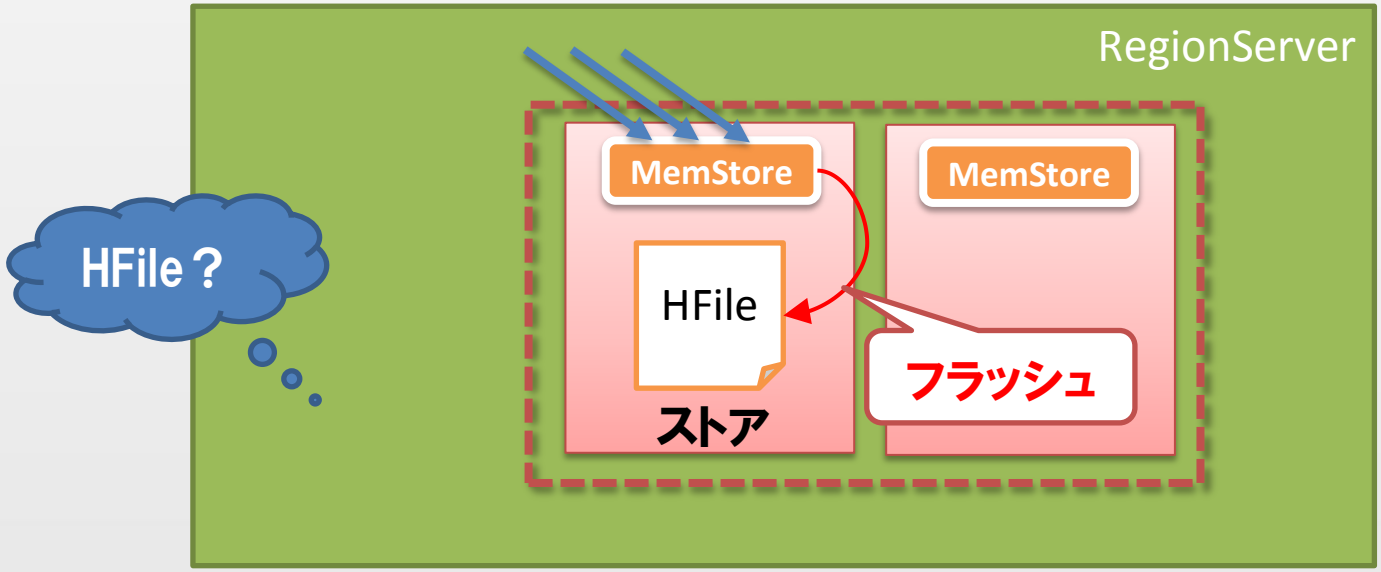
HLogの振る舞い

- HLogとは
  - WAL(Write-Ahead-Log)
  - データベースの更新情報のログ
  - サーバクラッシュ時に、一貫性を確保するのに利用
    - 直近データは、MemStore上に保持されているため、RegionServerがクラッシュした場合、HLogからデータを復旧する

MemStoreの振る舞い（フラッシュ）

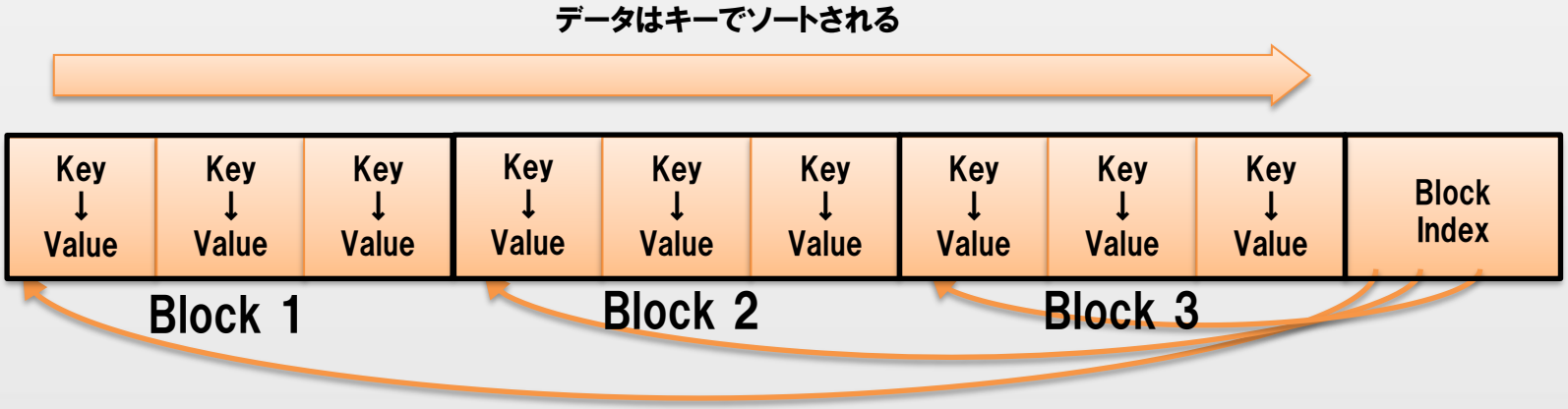
- MemStoreとは
  - 新規データの書き込みがあると、MemStoreにデータが書き込まれる
    - データの追加、修正、削除も同様に書きこまれます
  - MemStoreのサイズがしきい値を超えると、HFileに全てを出力する（フラッシュ）

プロパティ: hbase.hregion.memstore.flush.size
デフォルト値：67108864（64MB）



HFile の振る舞い

- HFileとは
  - HBaseのストアデータを保管するために特化したファイルフォーマット
  - キーの昇順にデータが格納されている
  - データは一定サイズごとのブロックに区切られ、ブロック単位で読み込まれる
    - デフォルト値：64KB。テーブル作成時に指定。
  - ブロックのインデックスがファイルの末尾に付与される

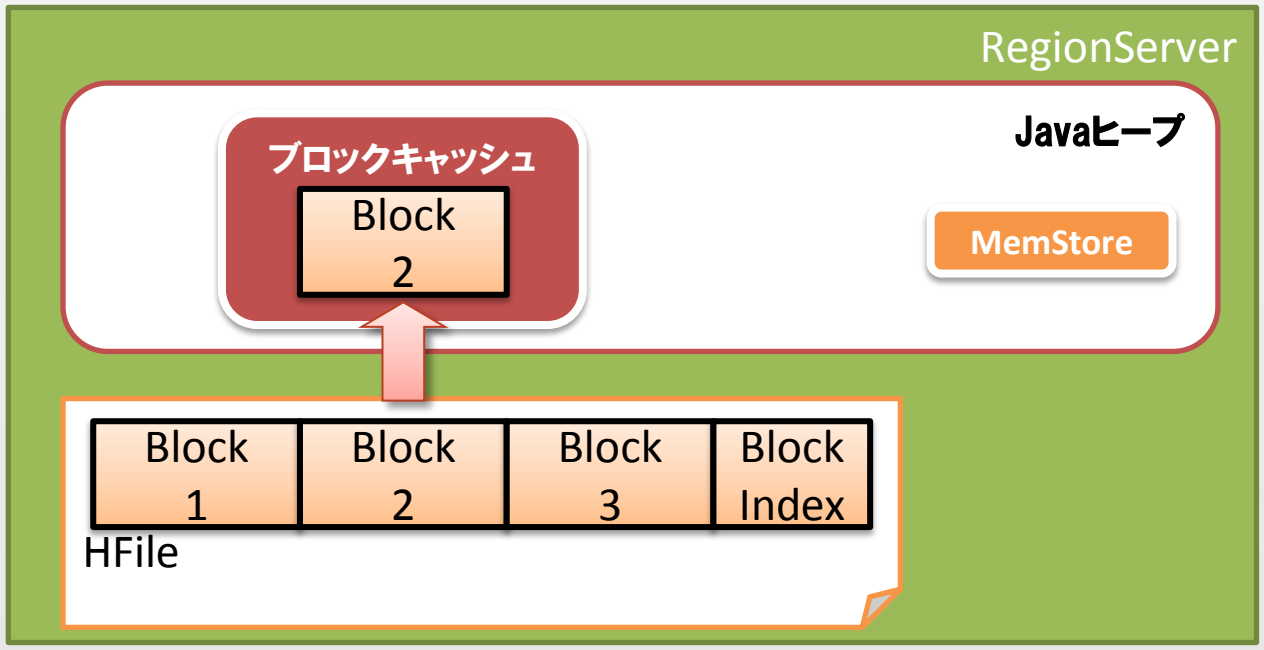


データアクセス詳細（読込）

- データ読み込み時は、ストア内のHFileから読み込みを行う。
- HFileの読み込みを効率化するため、RegionServerはブロックキャッシュを行う
  - ブロックのデータは、RegionServerのメモリ上でキャッシュされる（ブロックキャッシュ）

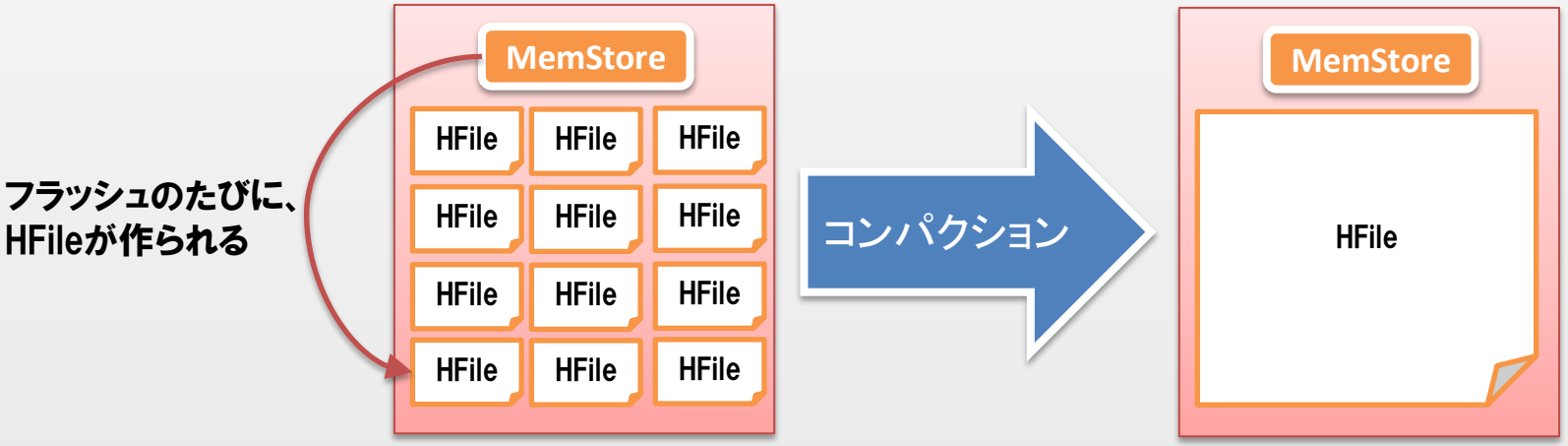
    デフォルト値：true。テーブル作成時に指定。
  - RegionSeverあたりのブロックキャッシュのサイズは、Javaヒープ最大サイズに対する割合で指定できる。

    プロパティ: hfile.block.cache.size
    デフォルト値：0.2（20%）



HFileの振る舞い（コンパクション）

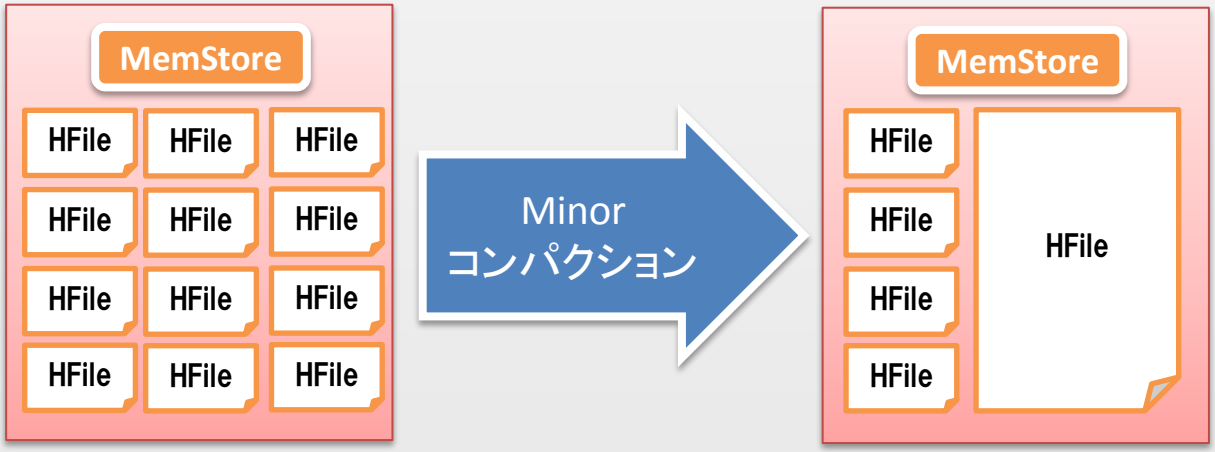
- フラッシュのたびに新たなHFileが作られる
- ストア内のHFile数が増えた場合、読み込み時のオーバヘッドが大きくなる
  - コンパクションによりHFileを統合することにより、Read性能を改善



HFileの振る舞い（Minorコンパクション）

- Minorコンパクション
  - 処理契機
    - フラッシュ時にHFileが一定数以上存在する場合に実行
    - 実行後に、スプリットも行う
  - 処理内容
    - 小さいHFileを1つに結合する。

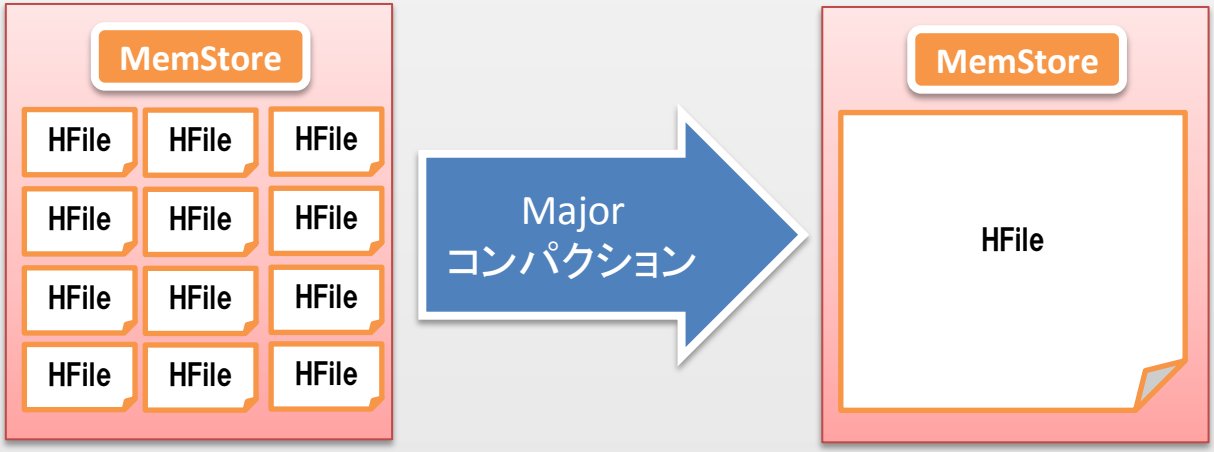
プロパティ：hbase.hstore.compactionThreshold
デフォルト値：3



HFileの振る舞い（Majorコンパクション）

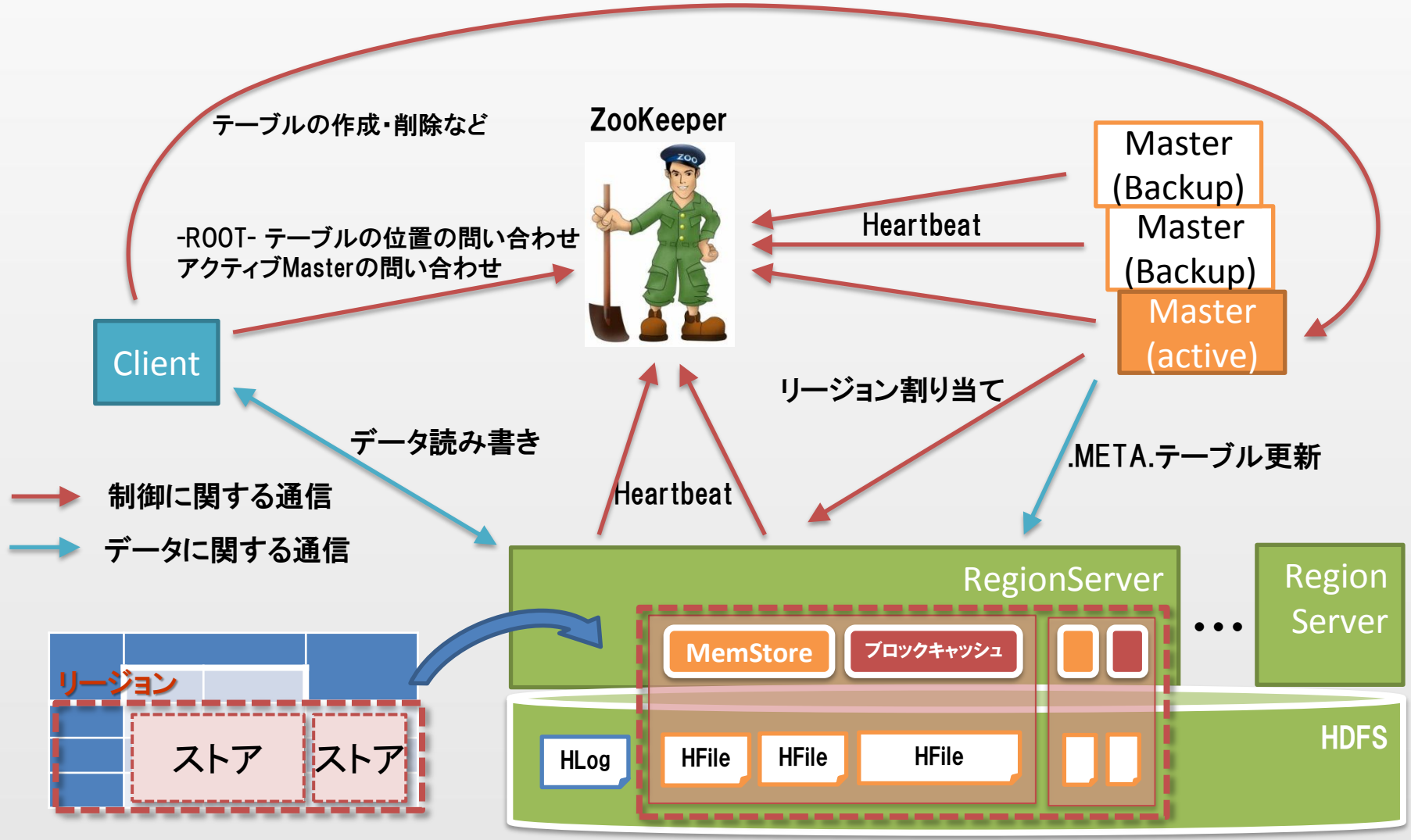
- Majorコンパクション
  - 処理契機
    - Majorコンパクションは、前回のMajorコンパクションまたは、ストア生成時より一定時間経過した場合に実行
    - 実行後にスプリットも行う
  - 処理内容
    - ストア内の全てのHFileを結合する

プロパティ：hbase.hregion.majorcompaction
デフォルト値：24時間

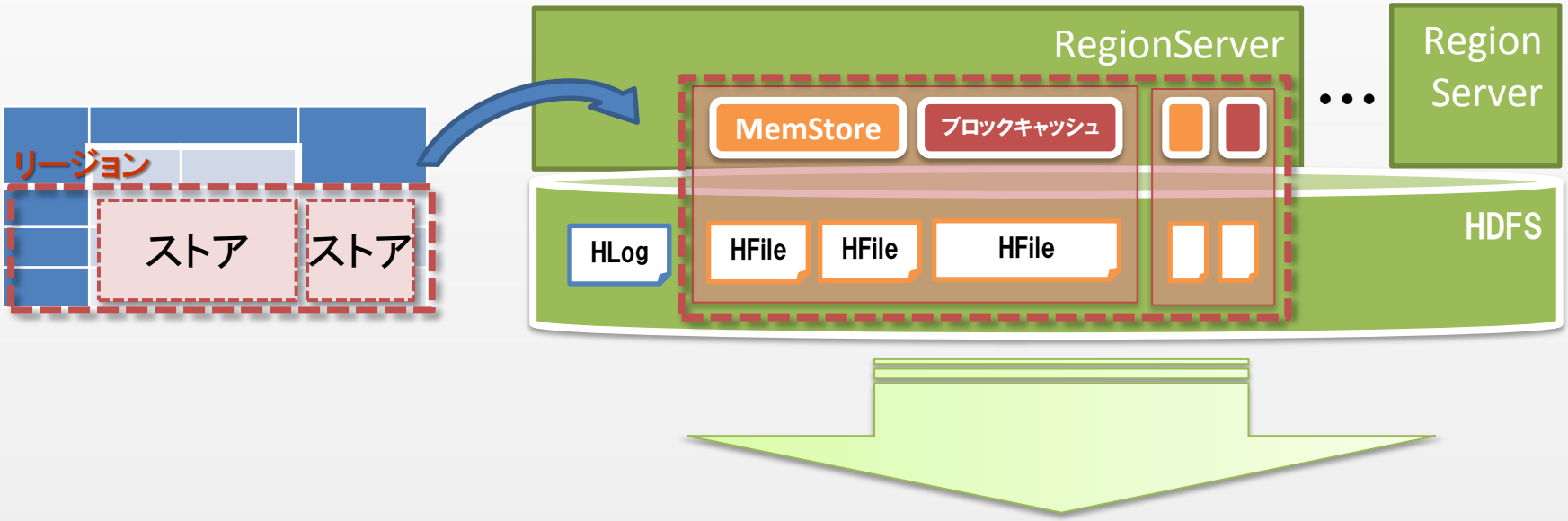


【再掲】HBaseクラスタ間の通信

- HBaseクラスタでの各ノード間の主な通信は、以下の通りである。



HBaseとHDFS



HDFS

/hbase/.logs/<region-server>/ HLog

/hbase/<tablename>/<region>/<column-family-name>/ HFile

HLogは、リージョンサーバごとに作られる

HFileは、テーブルごと、リージョンごと、カラムファミリーごとに作られる

実ファイルの確認（ディレクトリ構成）

1. HDFS上のファイル構成を確認

- 実行コマンド

```
$ hadoop fs -lsr /hbase
```

- 出力例

```
/hbase/-ROOT-/70236052/.oldlogs/hlog.1326706859115
/hbase/-ROOT-/70236052/.regioninfo
/hbase/-ROOT-/70236052/info/1804592931694086901

/hbase/.META./1028785192/.oldlogs/hlog.1326706859221
/hbase/.META./1028785192/.regioninfo
/hbase/.META./1028785192/info/5309202955053870004

/hbase/.logs/hdslave01,60020,1326793698011/hdslave01%3A60020.1326793698484

/hbase/testTable/<region>/colfamily1/8678027681811920936
/hbase/testTable/<region>/colfamily2/8678027681811920936
```

-ROOT-テーブルのデータ

.META.テーブルのデータ

HLog

HFile

実ファイルの確認（HLog）

2. HLogの中身の確認

- 実行コマンド

```
$ hbase org.apache.hadoop.hbase.regionserver.wal.HLog --dump
                    'hdfs://hdmaster01:54312/＜HLogのフルパス＞'
```

- 出力例

```
Sequence 104559 from region 35981c3bce0f484f67deca55beee68fc in table
testTable
  Action:
    row: a1
    column: colfamily1:
    at time: Fri Jan 20 23:15:55 JST 2012
Sequence 104560 from region 35981c3bce0f484f67deca55beee68fc in table
testTable
  Action:
    row: a1
    column: colfamily1:qua1
    at time: Fri Jan 20 23:16:02 JST 2012
```

実ファイルの確認（HFile）

3. HFileの中身の確認

- 実行コマンド

```
$ hbase org.apache.hadoop.hbase.io.hfile.HFile -p -f
            'hdfs://hdmaster01:54312/<HFileのフルパス>'
```

- 出力例

```
K: a1/colfamily1:/1327068955005/Put/vlen=3 V: msg
K: a1/colfamily1:qua1/1327068962044/Put/vlen=4 V: hoge
Scanned kv count -> 2
```

HBase の性能特性

- 書き込み時の動作
  - HLogへの書き込み
    - HDFSの機能により、3ノードに書き込まれる
    - HDFSの各ノードのメモリバッファリングするため、書き込みは高速。
      （ディスクへの書き込みまでは保証しない）
  - MemStoreへの書き込み
    - メモリにバッファリングするため、書き込みは高速。
- 読み込み時の動作
  - メモリ上(ブロックキャッシュ)に無いデータは、HFileから読み込む
  - HFileの各世代ごとにランダムアクセスするため、低速

- したがって、HBaseは以下の性能特性を持つ
  1. 読み込み処理よりも、書き込み処理を得意とする
  2. ランダムな読み込み処理よりも、シーケンシャルな読み込み処理を得意とする

負荷分散

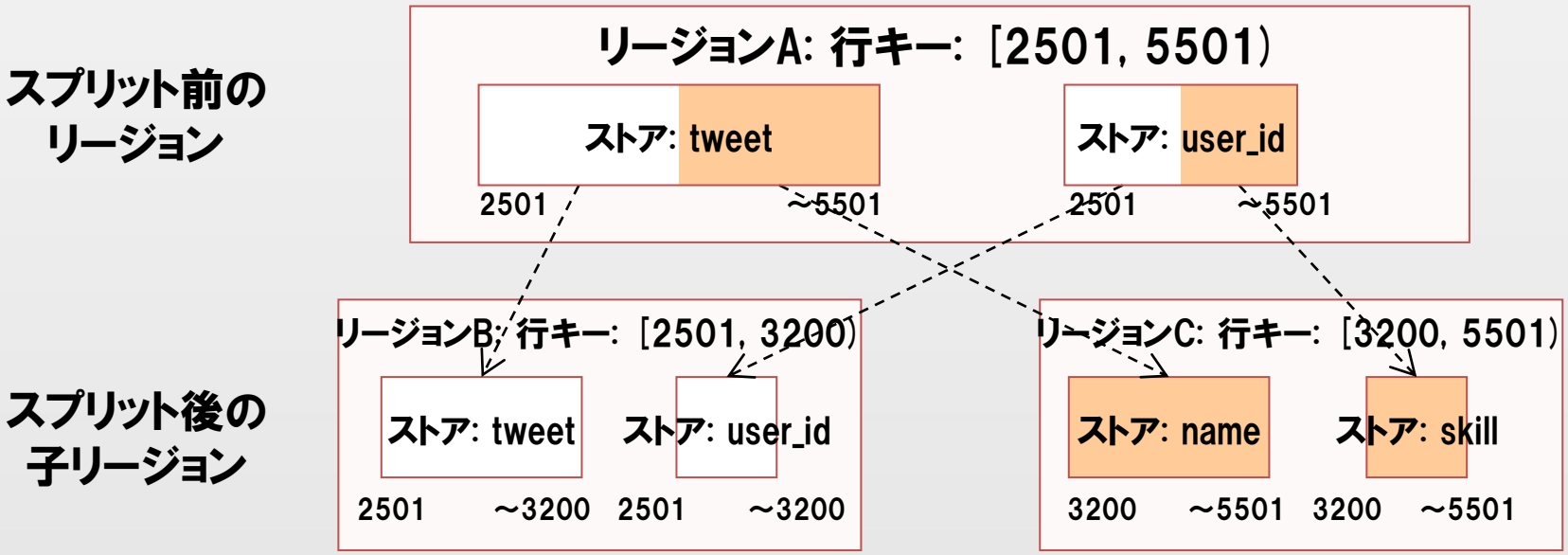
- HBaseは、RegionServerへの負荷分散を行うために、以下の機能がある
  - スプリット
    - ストアのサイズが一定のバイト数に達したリージョンを2つのリージョンに分割する
  - バランス
    - RegionServerが担当しているリージョンの数が、ほぼ同数になるように、担当を変更する
- スプリットとバランスを組み合わせて行うことにより、RegionServerが担当するリージョンの数およびバイト数は平準化される
- ただし、スプリットとバランスのいずれにおいても、アクセス数の平準化は考慮されない。このため、一部のテーブルの限られた範囲の行にアクセスが集中する場合には、一部のRegionServerに対して負荷が集中してしまうということに注意が必要

スプリット

- スプリットは、テーブル内のリージョンのサイズを平準化するために、RegionServerが行う
- 処理契機
  - コンパクションを実行した直後。
  - コンパクション後のストアが持つHFileの最大サイズが閾値を超える時、続けてスプリットを実行する

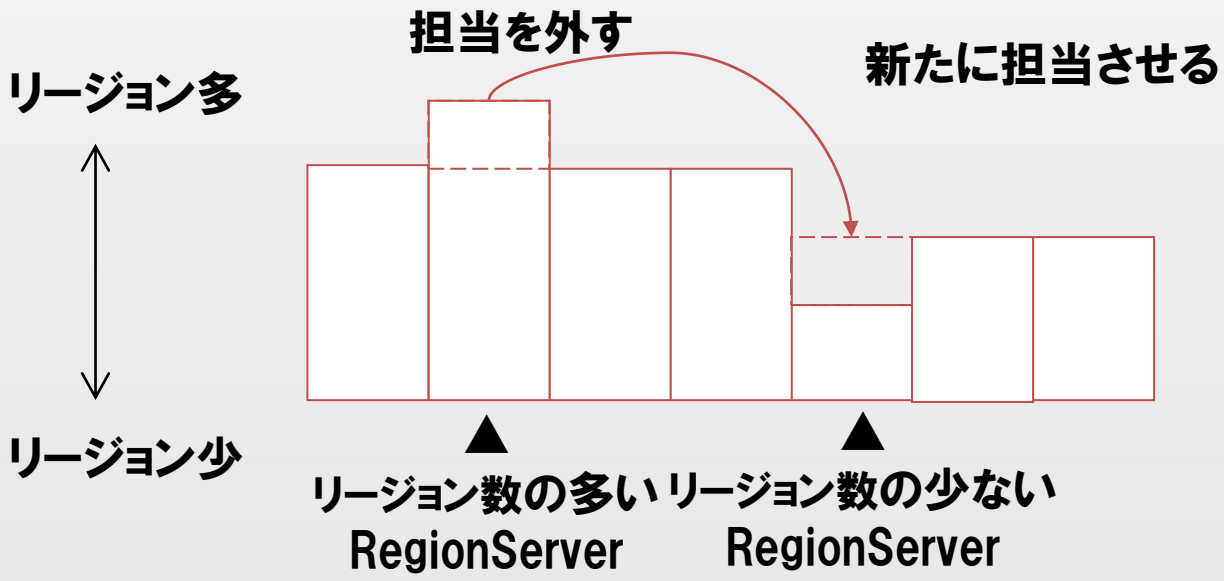
プロパティ：hbase.hregion.max.filesize
デフォルト値：256MB

- 処理内容
  - リージョン中で最大のストア、最大のHFileが二等分されるキーを分割キーとして、2つのリージョンを作成する



バランス

- バランスは、RegionServerが担当するリージョン数を平準化するために、Masterが行う
- 処理契機
  - 5分ごと（hbase.balancer.periodで変更可能）にMasterのバックグラウンドスレッドBalancerChoreが実行する
- 処理内容
  - 平均よりも多くのリージョンを担当しているRegionServerから担当を外し、平均よりも少ないリージョンを担当しているRegionServerに、新たに担当させる



まとめ

本講義で学んだ内容

- HBase概要
  - 列指向データベース
  - トランザクションは行単位のみ可能
  - 単純なクエリのみ可能
  - インデックスは行キーのみ
  - 柔軟なテーブル構造
- HBaseデータモデル
  - 行キー、列ファミリ、修飾子、タイムスタンプ、
  - create/drop、put/get/scan/delete
- HBaseアーキテクチャ
  - RDBMSとは異なる性能特性（read ＜ write）
  - 性能向上のための仕組み(フラッシュ/コンパクション/WALログ）
  - 自動的に負荷分散(スプリット/バランス）