HBase技术原理

www.huawei.com

HUAWEI

目标

- 学完本课程后，您将能够：
  - 掌握**HBase**的系统架构
  - 掌握**HBase**的关键特性
  - 熟悉**HBase**的基本功能
  - 熟悉**HBase**华为增强特性

目录

HBase基本定义

HBase是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统。

- 适合于存储大表数据（表的规模可以达到数十亿行以及数百万列），并且对大表数据的读、写访问可以达到实时级别；
- 利用**Hadoop HDFS**（**Hadoop Distributed File System**）作为其文件存储系统，提供高可靠性、高性能、列存储、可伸缩、实时读写的数据库系统；
- 利用**ZooKeeper**作为协同服务。

与RMDB比较

HBase

分布式存储，面向列。

动态扩展列。

普通商用硬件支持，扩容成本低。

RMDB

数据结构固定。

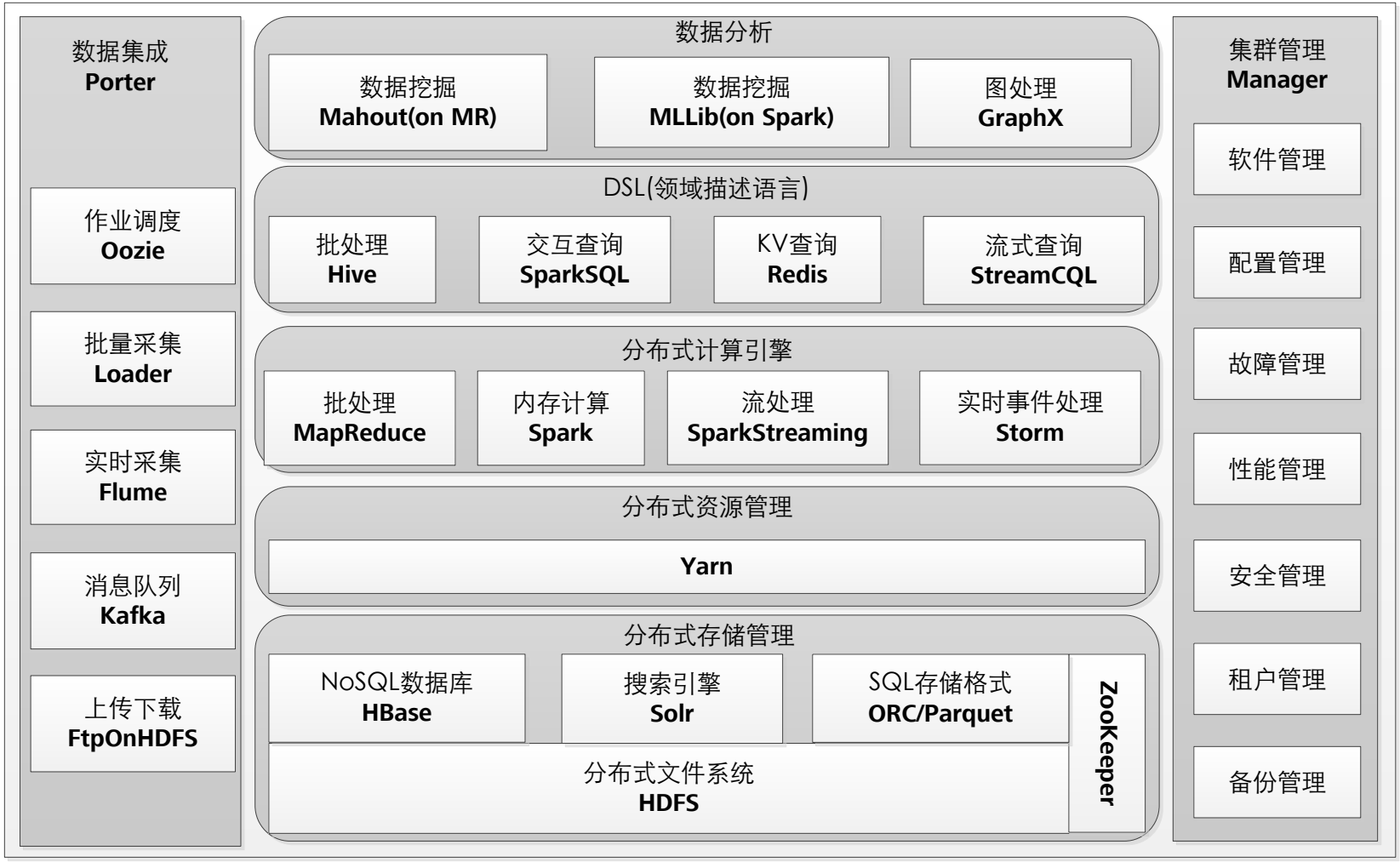
需要预先定义好数据结构。

需要大量IO，扩展成本大。

HBase应用场景

- **HBase适合具有如下需求的应用：**
  - 海量数据（**TB**、**PB**）
  - 高吞吐量
  - 需要在海量数据中实现高效的随机读取
  - 需要很好的性能伸缩能力
  - 能够同时处理结构化和非结构化的数据
  - 不需要完全拥有传统关系型数据库所具备的**ACID**特性

HBase产品定位



数据结构介绍

结构化数据

- **具有固定的结构，属性划分，以及类型等信息。**我们通常所理解的关系型数据库中所存储的数据信息，大多是结构化数据， 如职工信息表，拥有**ID**、**Name**、**Phone**、**Address**等属性信息。
- 通常直接存放在数据库表中。数据记录的每一个属性对应数据表的一个字段。

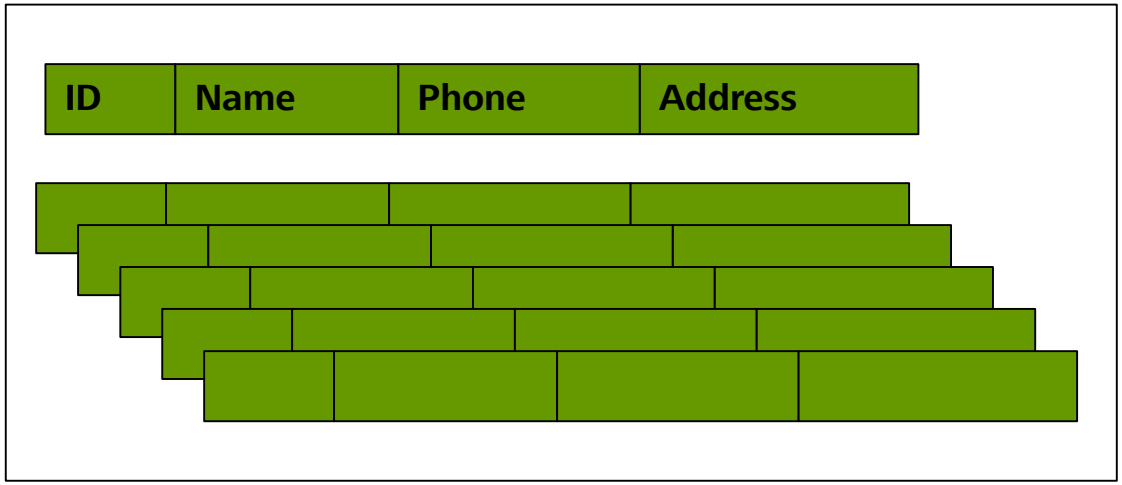
非结构化数据

- **无法用统一的结构来表示。**如文本文件、图像、视频、声音、网页等信息。
- 数据记录较小时**(**如**KB**级别**)**，可考虑直接存放到数据库表中（整条记录映射到某一个列中），这样也有利于整条记录的快速检索。
- 数据较大时，通常考虑直接存放在文件系统中。数据库可用来存放相关数据的索引信息。

半结构化数据

- **具有一定的结构，但又有一定的灵活可变性。**典型如**XML**、**HTML**等数据。其实也是非结构化数据的一种。
- 可以考虑直接转换成结构化数据进行存储。
- 根据数据记录的大小和特点，选择合适的存储方式。这一点与非结构化数据的存储类似。

按行存储

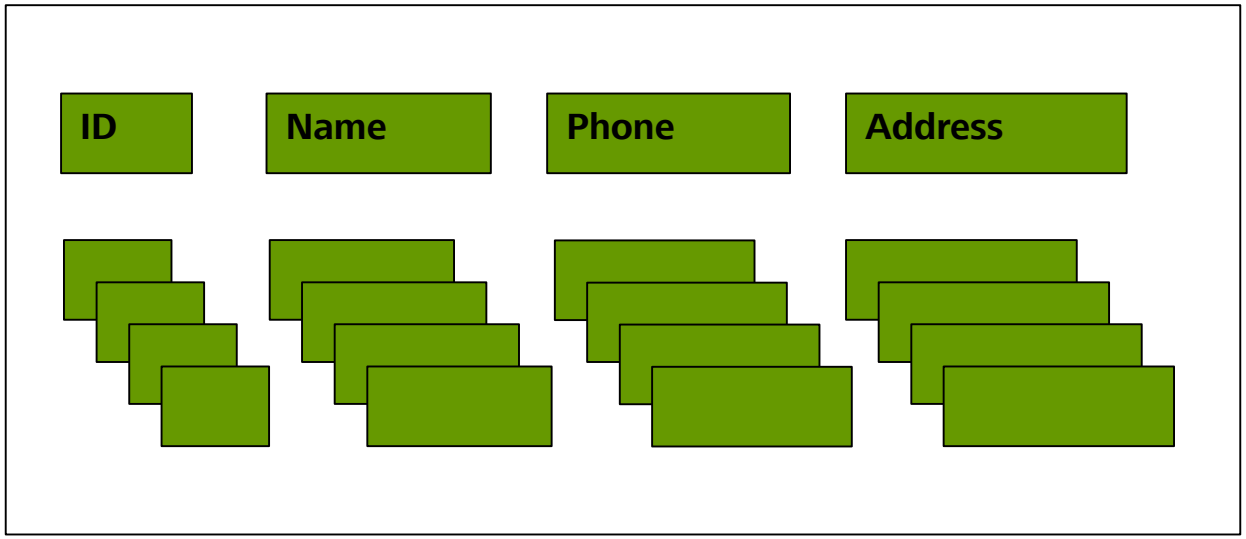


- 按行存储：

数据按行存储在底层文件系统中。通常，每一行会被分配固定的空间。

- 优点：有利于增加/修改整行记录等操作；有利于整行数据的读取操作；
- 缺点：单列查询时，会读取一些不必要的数据。

按列存储



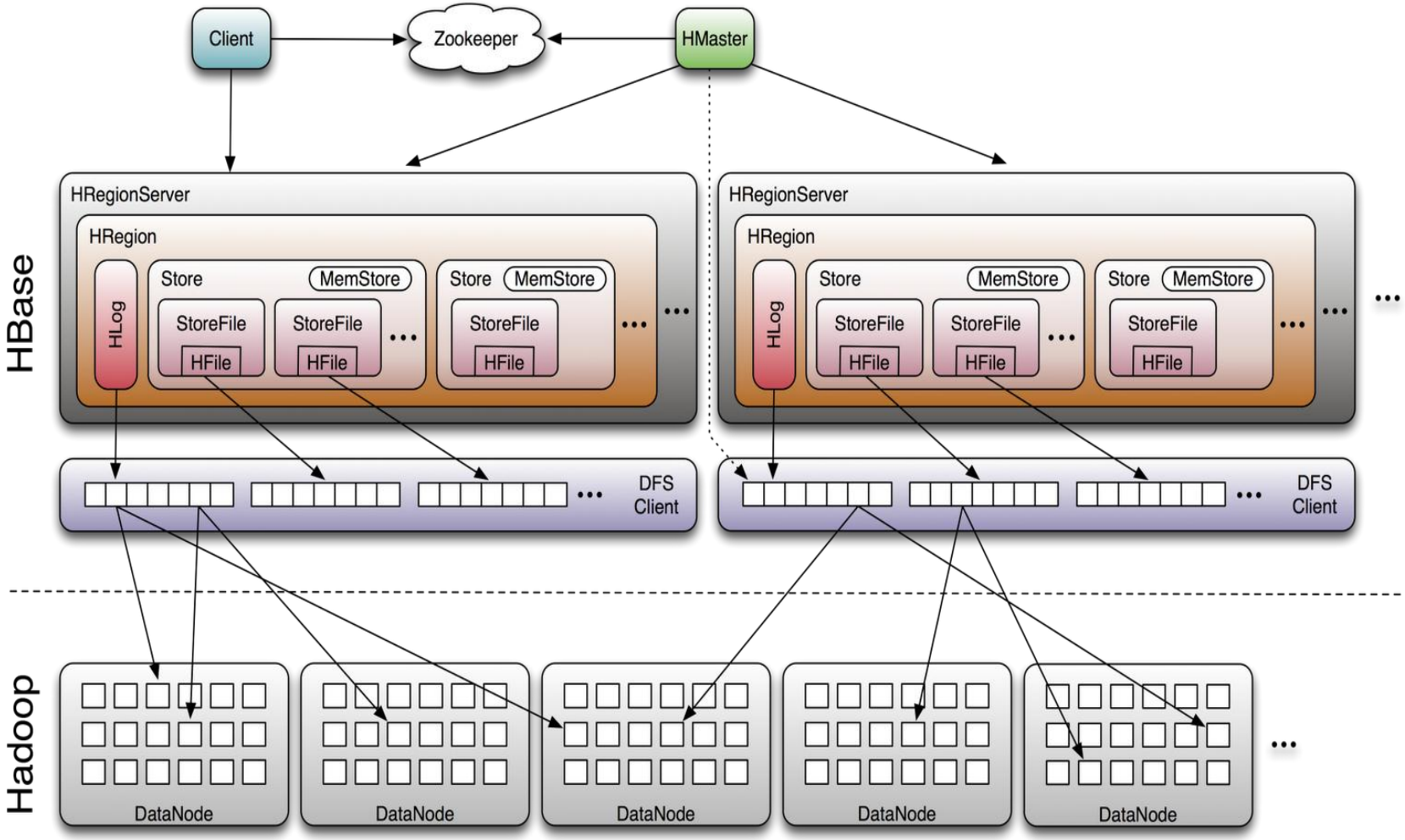
- 按列存储：

数据以列为单位，存储在底层文件系统中。

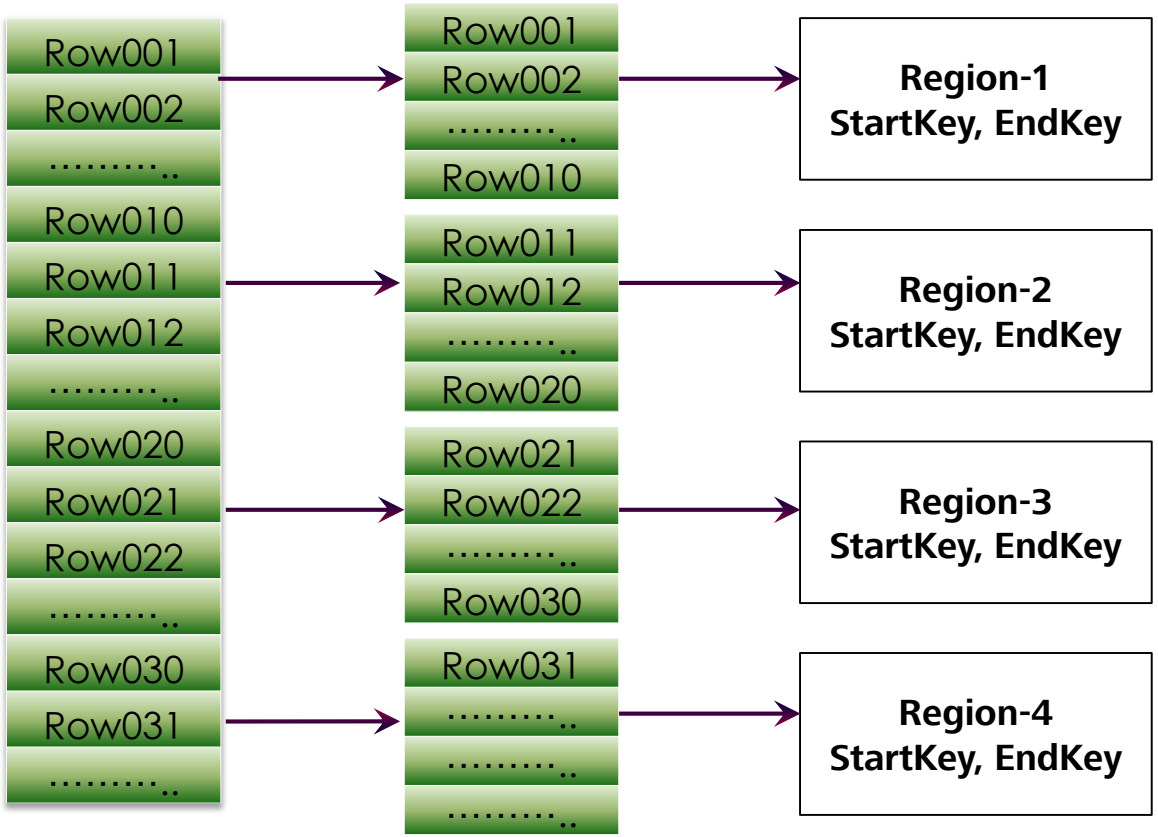
  - 优点：有利于面向单列数据的读取/统计等操作。
  - 缺点：整行读取时，可能需要多次I/O操作。

目录

HBase架构介绍



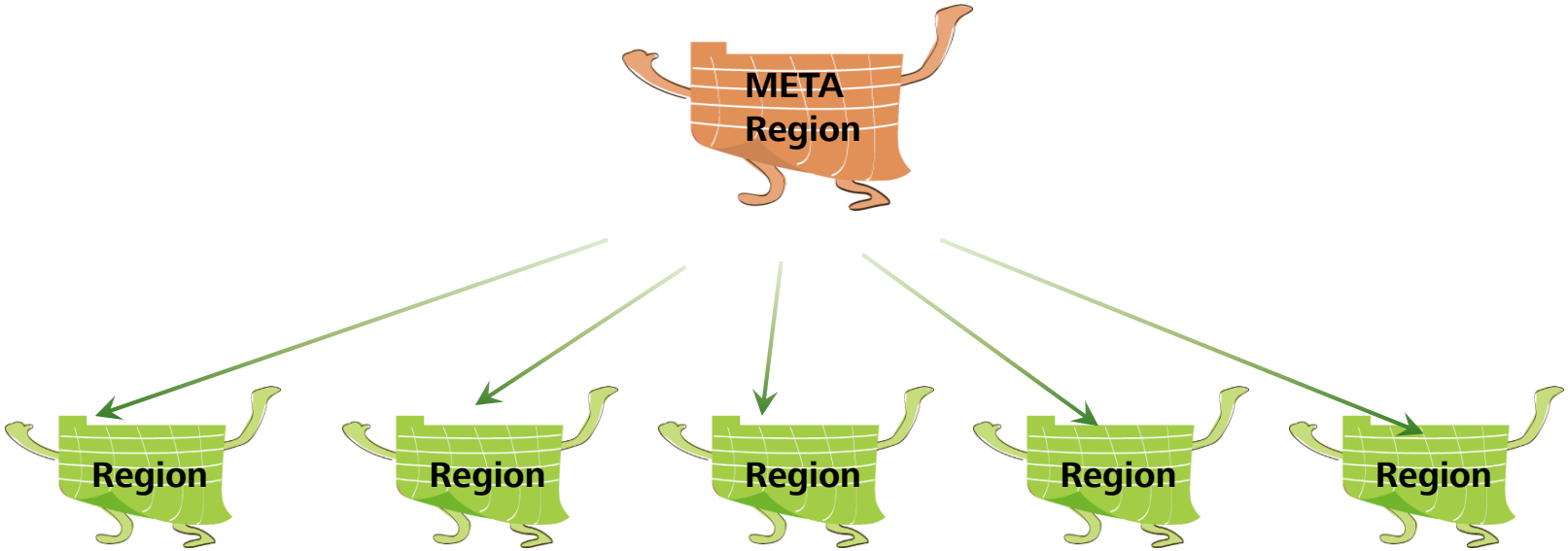
基本概念 —— Region(1)



基本概念 —— Region(2)

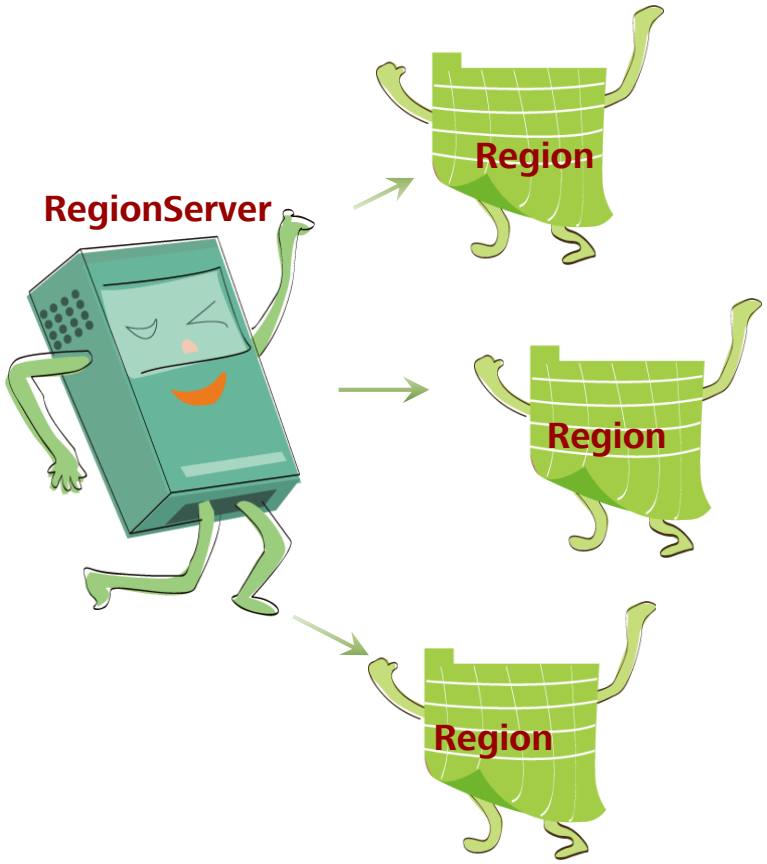
- 将一个数据表按**Key**值范围横向划分为一个个的子表，实现分布式存储。
- 这个子表，在**HBase**中被称作“**Region**”。
- 每一个**Region**都关联一个**Key**值范围，即一个使用**StartKey**和**EndKey**描述的区间。事实上，每一个**Region**仅仅记录**StartKey**就可以了，因为它的**EndKey**就是下一个**Region**的**StartKey**。
- **Region**是**HBase**分布式存储的最基本单元。

基本概念 —— Region(3)



- **Region**分为元数据**Region**以及用户**Region**两类。
- **Meta Region**记录了每一个**User Region**的路由信息。
- 读写**Region**数据的路由，包括如下几步：
  - 找寻**Meta Region**地址。
  - 再由**Meta Region**找寻**User Region**地址。

角色介绍 —— RegionServer

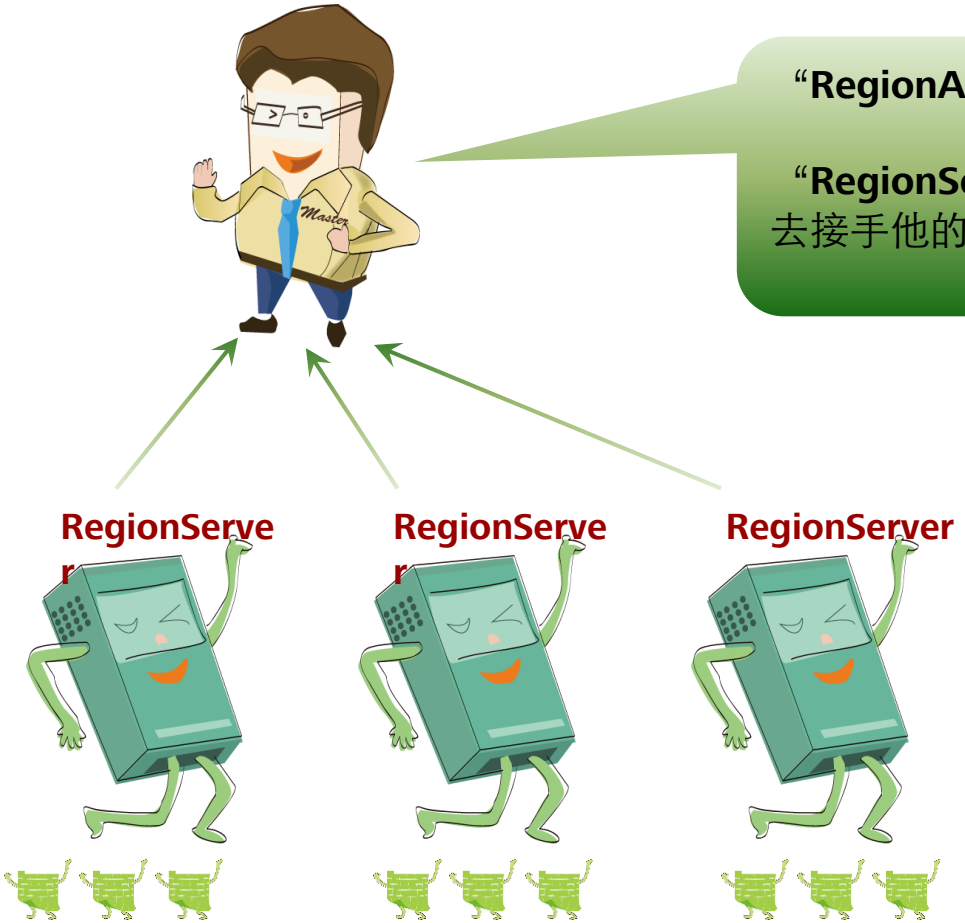


- **RegionServer**是**HBase**的数据服务进程。负责处理用户数据的读写请求。
- **Region**被交由**RegionServer**管理。实际上，所有用户数据的读写请求，都是和**RegionServer**上的**Region**进行交互。
- **Region**可以在**RegionServer**之间发生转移。

思考：

- 一条用户数据**KeyValue**必然属于一个唯一的**Region**；**Region**由**RegionServer**来管理，那么，这个路由信息保存在哪里呢?
- **Region**如何才可以转移？由谁负责转移?

角色介绍 —— HMaster(1)



角色介绍 —— HMaster(2)

- **HMaster**进程负责管理所有的**RegionServer**。
  - 新**RegionServer**的注册。
  - **Regio`nServer Failover**处理。
- 负责建表/修改表/删除表以及一些集群操作。
- **HMaster**进程负责所有**Region**的转移操作。
  - 新表创建时的**Region**分配。
  - 运行期间的负载均衡保障。
  - **RegionServer Failover**后的**Region**接管。
- **HMaster**进程有主备角色。集群可以配置两个**HMaster**角色，集群启动时，这些**HMaster**角色通过竞争获得主**HMaster**角色。主**HMaster**只能有一个，备**HMaster**进程在集群运行期间处于休眠状态，不干涉任何集群事务。

疑问：
主备HMaster进程角色是如何进行裁决的?

ZooKeeper

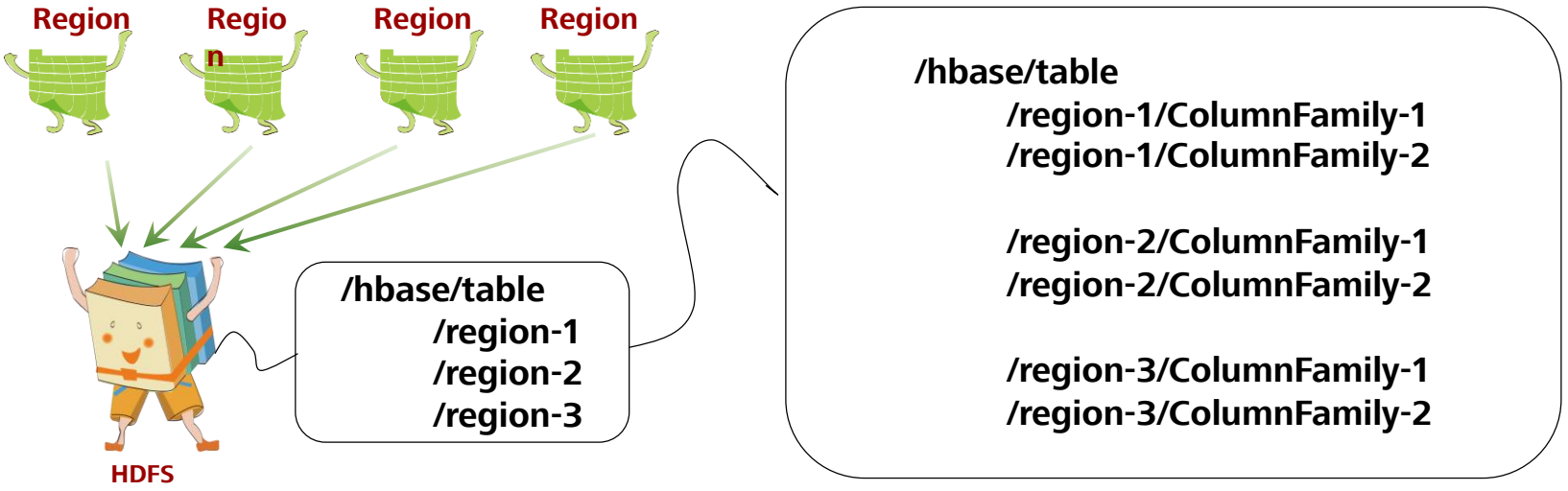
- **ZooKeeper为HBase 提供:**
  - 分布式锁的服务

    多个**HMaster**进程竞争主**HMaster**角色时，怎么样保证仅有一个**Active**角色存在？这就需要一个分布式的锁机制来保证。多个**HMaster**进程都尝试着去**ZooKeeper**中写入一个对应的节点，该节点只能被一个**HMaster**进程创建成功，创建成功的**HMaster**进程就是主角色。
  - 事件监听机制

    主**HMaster**进程宕掉之后，其它的备**HMaster**如何能够快速的接管？这个过程中，备**HMaster**在监听那个对应的**ZooKeeper**节点。主**HMaster**进程宕掉之后，该节点会被删除，那么，其它的备**HMaster**就可以收到相应的消息。
  - 微型数据库角色

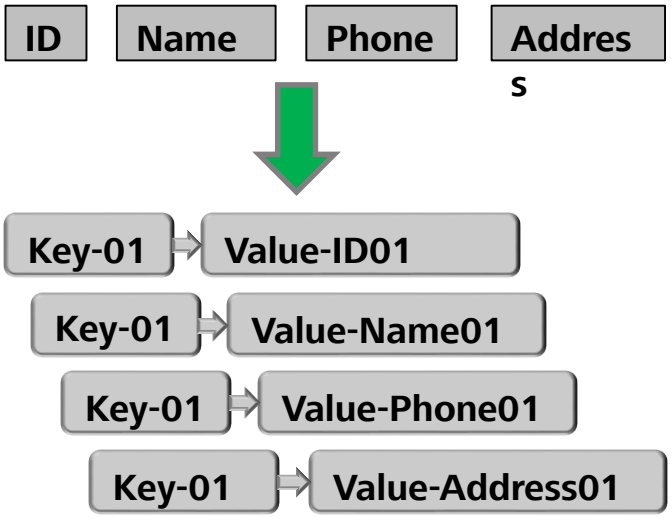
    ZooKeeper中存放了**Region Server**的地址，此时，可以将它理解成一个微型数据库。

HBase数据模型 - Column Family



- **ColumnFamily**是**Region**的一个物理存储单元。同一个**Region**下面的多个**ColumnFamily**，位于不同的路径下面。
- **ColumnFamily**信息是表级别的配置。也就是说，同一个表的多个**Region**，都拥有相同的**ColumnFamily**信息（例如，都有两个**ColumnFamily**，且不同**Region**的同一个**ColumnFamily**配置信息相同）。

HBase数据模型 - KeyValue(1)

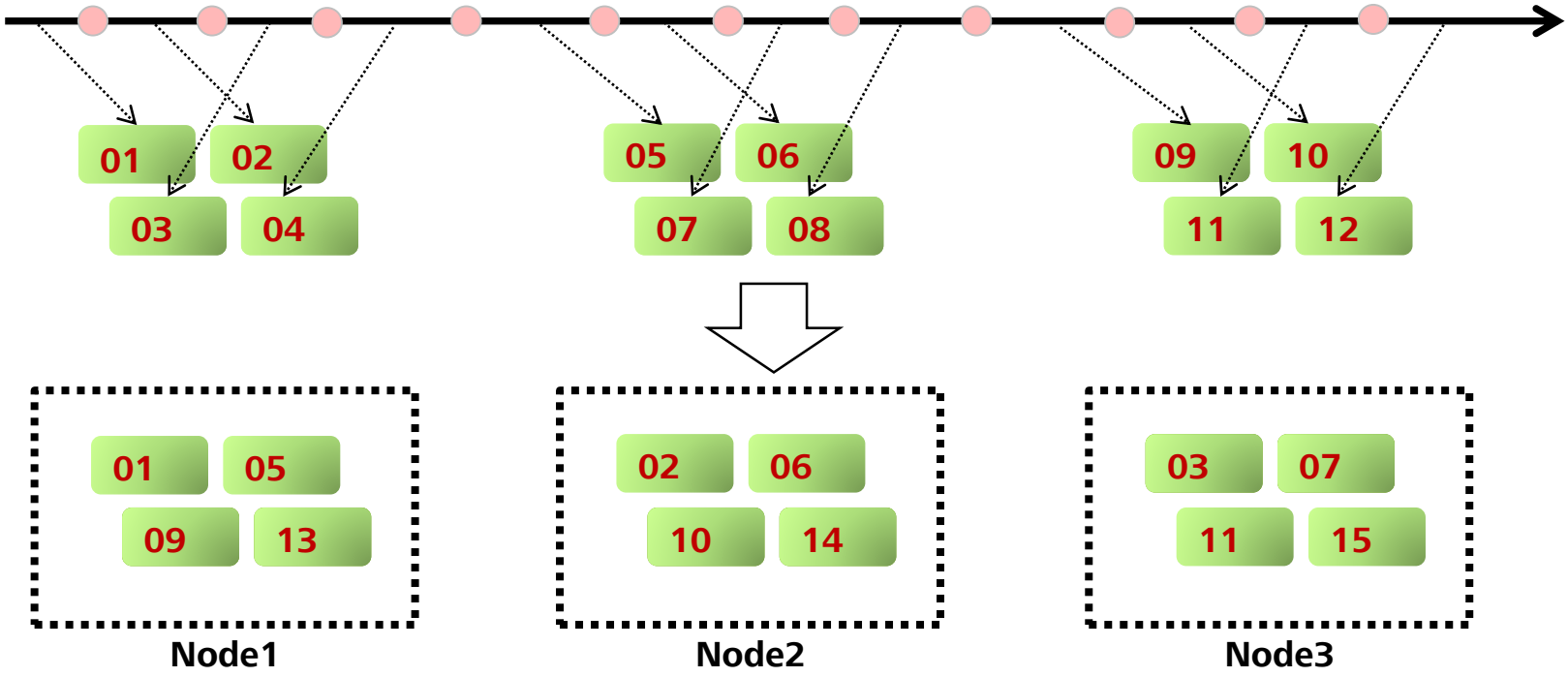


关系型数据库	KeyValue数据库
• 关系型数据库是基于表的数据库，每行皆由固定列数的列组成。 • 关系型数据库的数据结构必须事先定义好。 • 基于现实含义来建立数据模型，而非基于应用程序功能。 • 数据模型标准化，表间可以彼此关联协作存储。	• 数据基于动态结构，容易适应数据类型和结构的变化。 • 以块为单元操作数据。 • 列间、表间并无关联关系。

- **KeyValue**具有特定的结构。**Key**部分被用来快速的检索一条数据记录，**Value**部分用来存储实际的用户数据信息。
- **KeyValue**作为承载用户数据的基本单元，需要保存一些对自身的描述信息，例如，时间戳，类型等等。那么，势必会有一定的结构化空间开销。

HBase数据模型 - KeyValue(2)

- **KeyValue**型数据库数据分区方式--按**Key**值连续范围分区



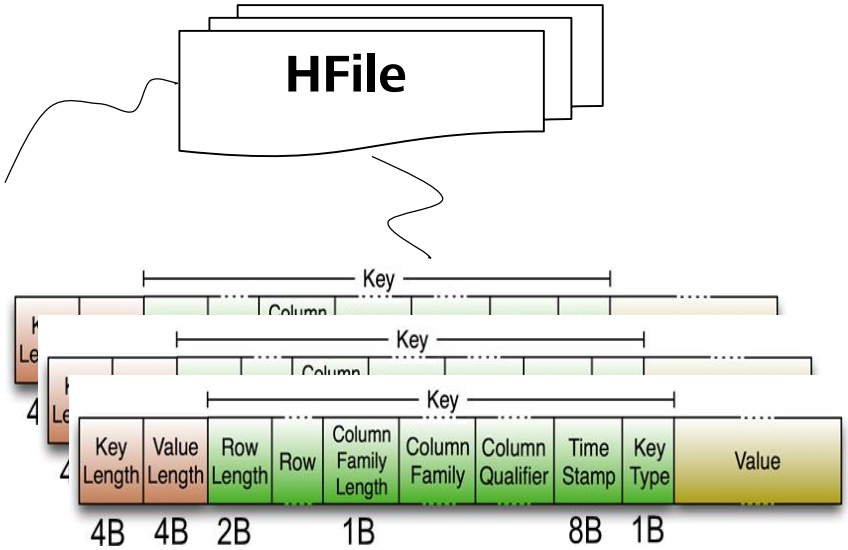
数据按照**RowKey**的范围(按一定的排序算法排序的结果，如按**RowKey**的字典顺序)，划分为一个个的子区间。每一个子区间都是一个分布式存储的基本单元。

HBase数据模型 - KeyValue(3)

/hbase/table
/region-1/ColumnFamily-1
/region-1/ColumnFamily-2

/region-2/ColumnFamily-1
/region-2/ColumnFamily-2

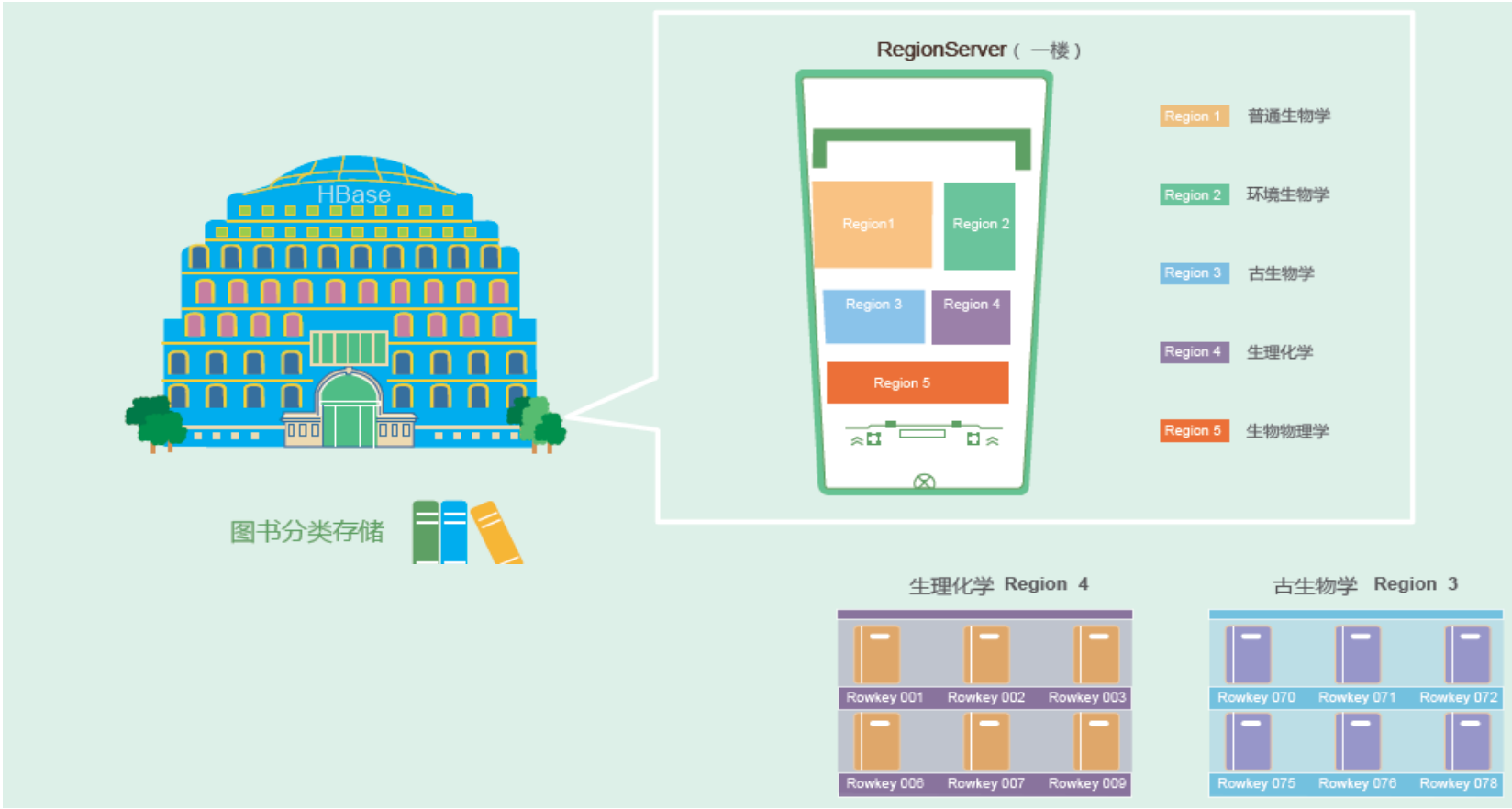
/region-3/ColumnFamily-1
/region-3/ColumnFamily-2



- **HBase**的底层数据都是以**KeyValue**的形式存在的。**KeyValue**具有特定的格式。
- **KeyValue**中拥有时间戳、类型等关键信息。
- 同一个**Key**值可以关联多个**KeyValue**，每一个**KeyValue**都拥有一个**Qualifier**标识。
- 即使是**Key**值相同，**Qualifier**也相同的多个**KeyValue**，也可能有多个，此时使用时间戳来区分，这就是同一条数据记录的多版本。

目录

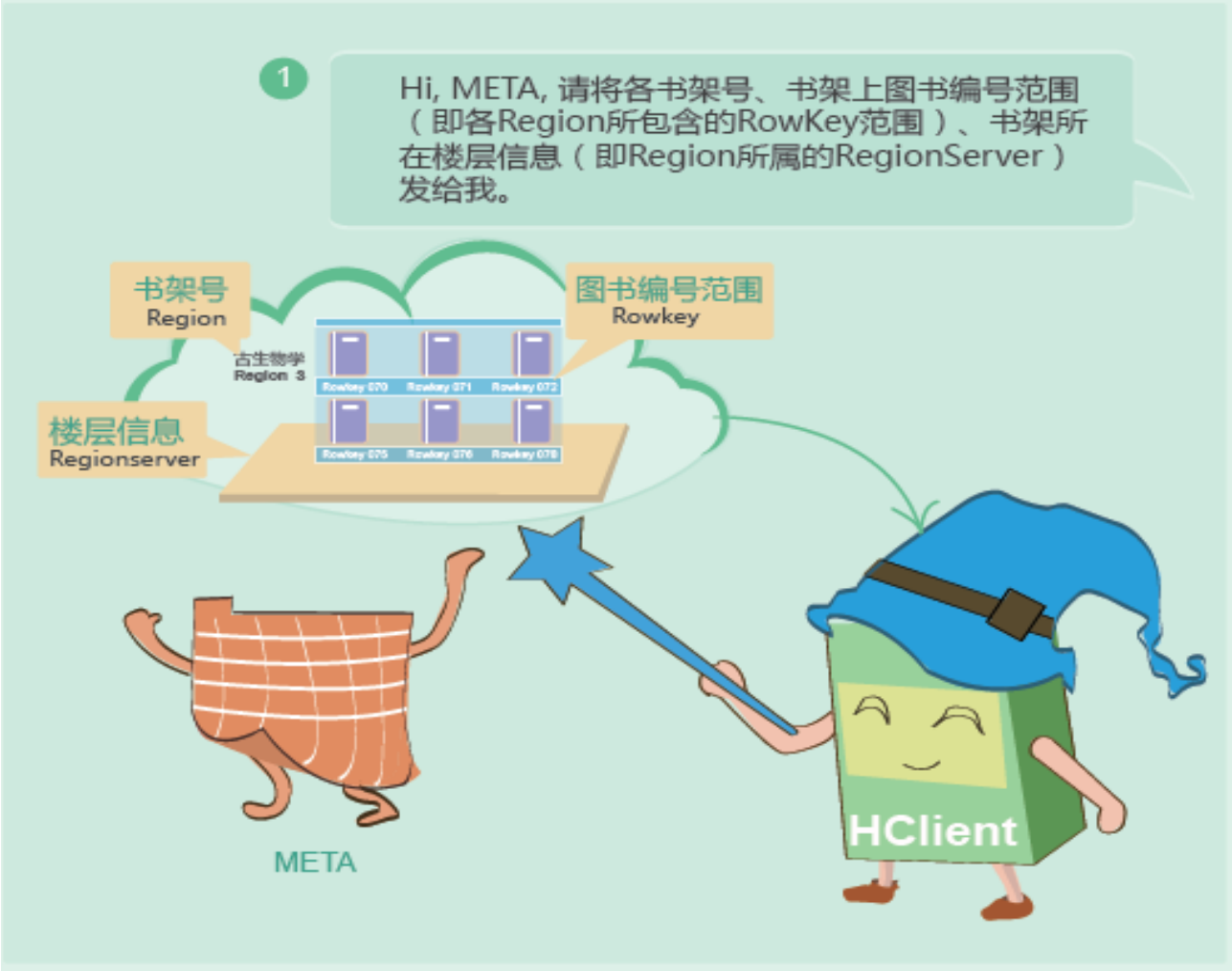
写流程



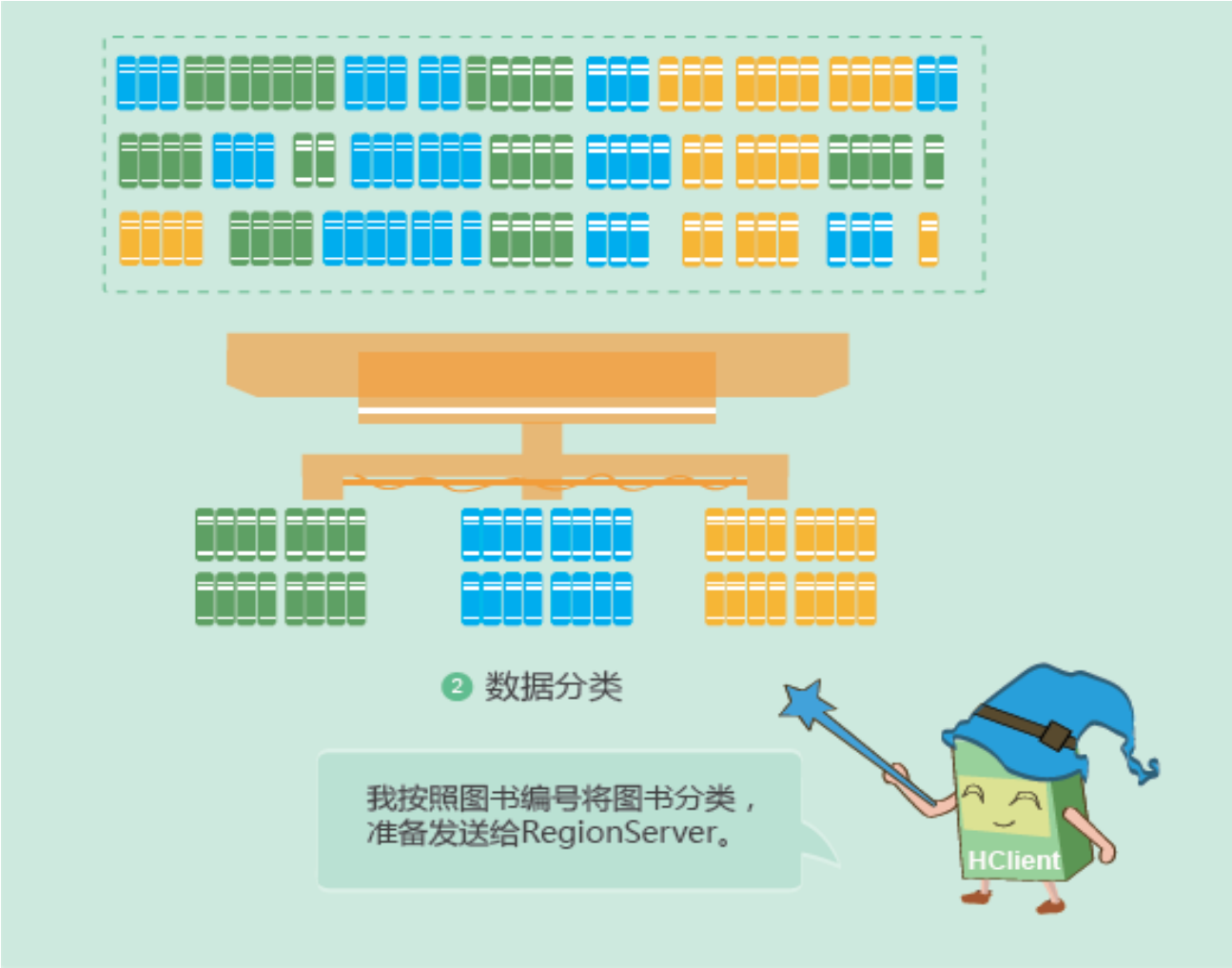
A. 客户端发起写数据请求

客户端发起写请求，相当于图书供应商需要把图书发往到图书馆，但是这时候需要定位到哪些图书该发往到哪栋楼哪一层，也就是下一页描述的定位**region**

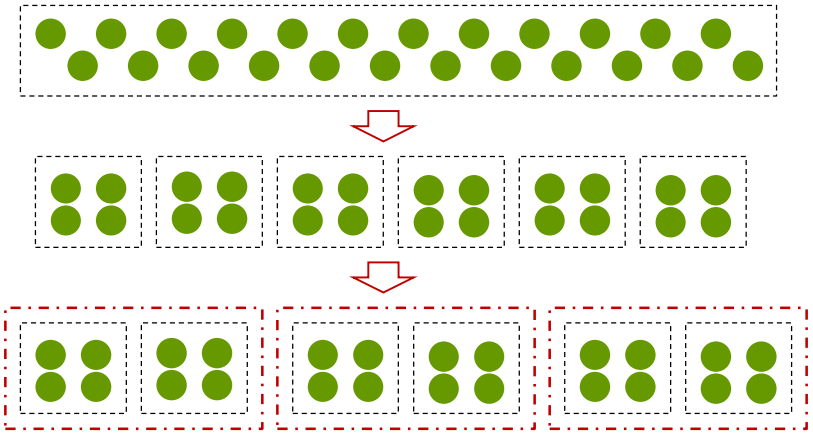
B. 写流程 - 定位Region



C. 写流程 - 数据分组

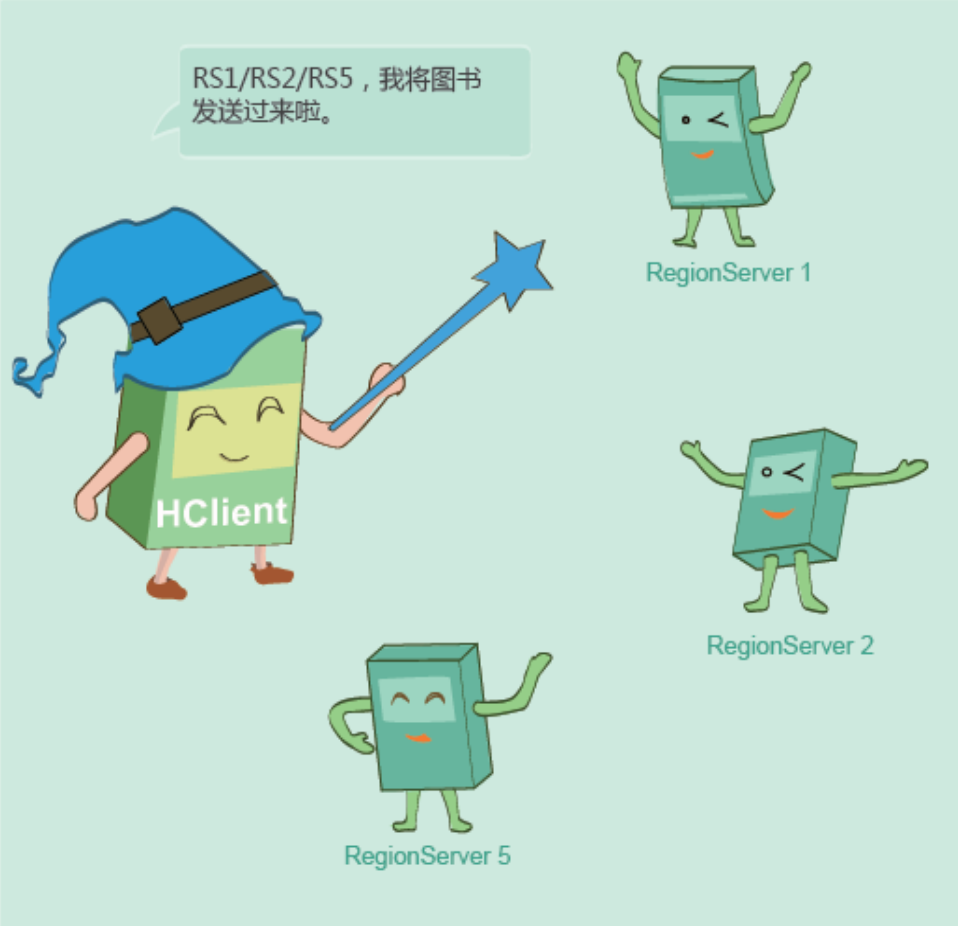


C. 写流程 – 数据分组



- 整个数据分组，涉及到两步“分篮子”操作：
  - 将所有的记录按**Region**划分。
  - 将所有的记录按**RegionServer**划分。
- 每个**RegionServer**上的数据会一起发送，这样，发送的数据中，都是已经按照**Region**分好组了。

C. 写流程－往RegionServer发送请求



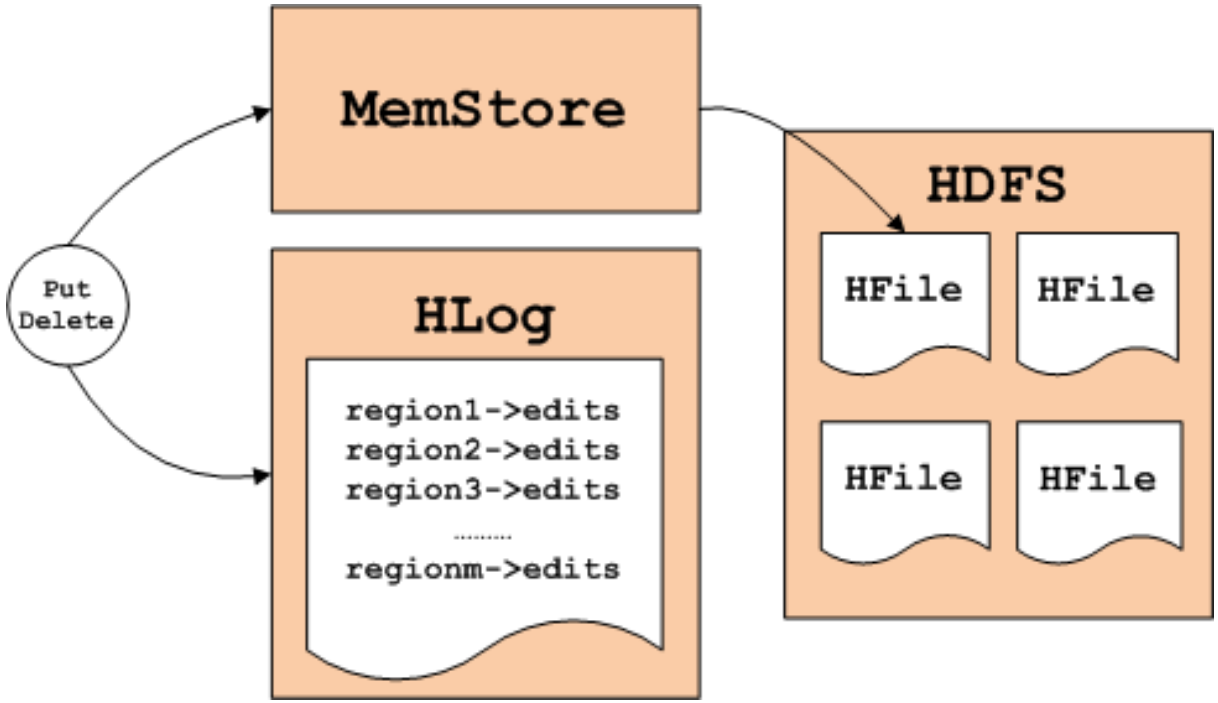
- 利用**HBase**自身封装的**RPC**框架，来完成数据发送操作。
- 往多个**RegionServer**发送请求是并行操作。
- 客户端发送完写数据请求后，会自动等待请求处理结果。
- 如果客户端没有捕获到任何的异常，则认为所有数据都已经被写入成功。如果全部写入失败，或者部分写入失败，客户端能够获知详细的失败**Key**值列表。

D. 写流程 - Region写数据流程

D. 写流程 - Region写数据流程

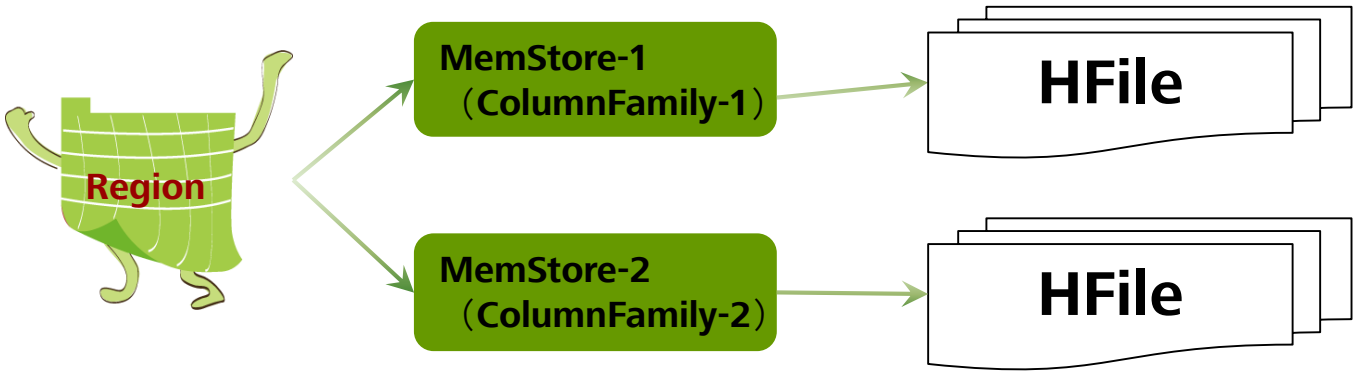
先写内存的原因：**HBase**提供了一个**MVCC**机制，来保障写数据阶段的数据可见性。先写**MemStore**再写**WAL**，是为了一些特殊场景下，内存中的数据能够更及时的可见。如果写**WAL**失败的话，**MemStore**中的数据会被回滚。

E. 写流程



- 将需要写入或删除的数据暂时保存在每个**Region**的内存中，即**MemStore**中。
- **写内存，避免多Region情形下带来的过多的分散IO操作。**
- 数据在写入到**MemStore**之后，也会**顺序写**入到**HLog**中，以保证数据的安全。

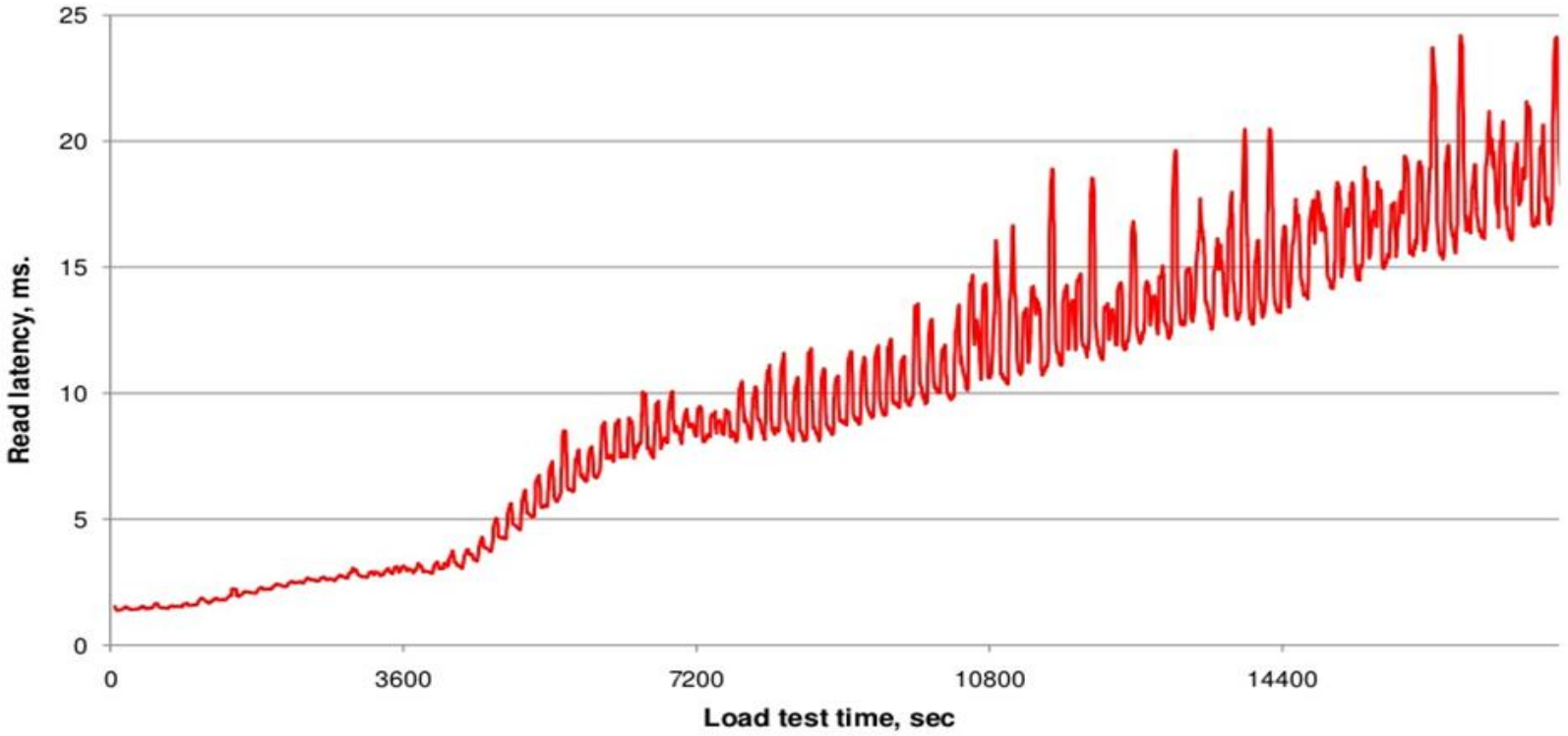
F. 写流程 - Flush



如下三种场景，会触发一个**Region**的**Flush**操作：

- 该**Region**的**MemStore**的总大小，达到了预设的**Flush Size**阈值。这种场景下的**Flush**操作，通常仅瞬间堵塞用户的写操作。但如果超出预设**Flush Size**阈值过多的话，也可能会引起一小段时间的堵塞。
- **RegionServer**的总内存大小超出了预设的阈值大小。这种场景下，在总内存没有降低到预设的阈值以下之前，可能会较长时间堵塞。
- 当**WALs**中文件数量达到阈值时。

写流程 - 多HFile的影响

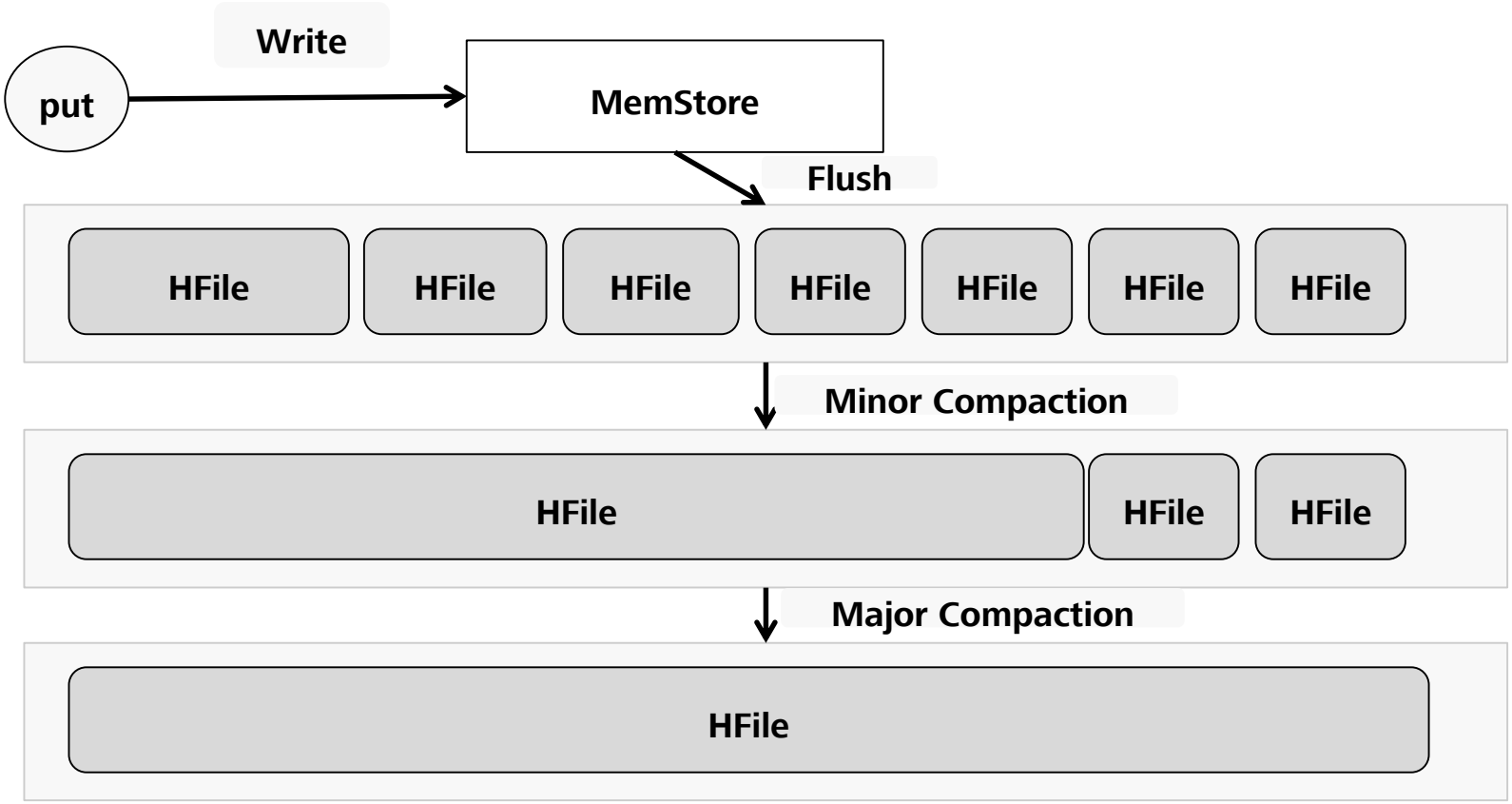


随着时间的不断迁移，**HFile**文件数目越来越多，读取时延也越来越大!

写流程 - Compaction

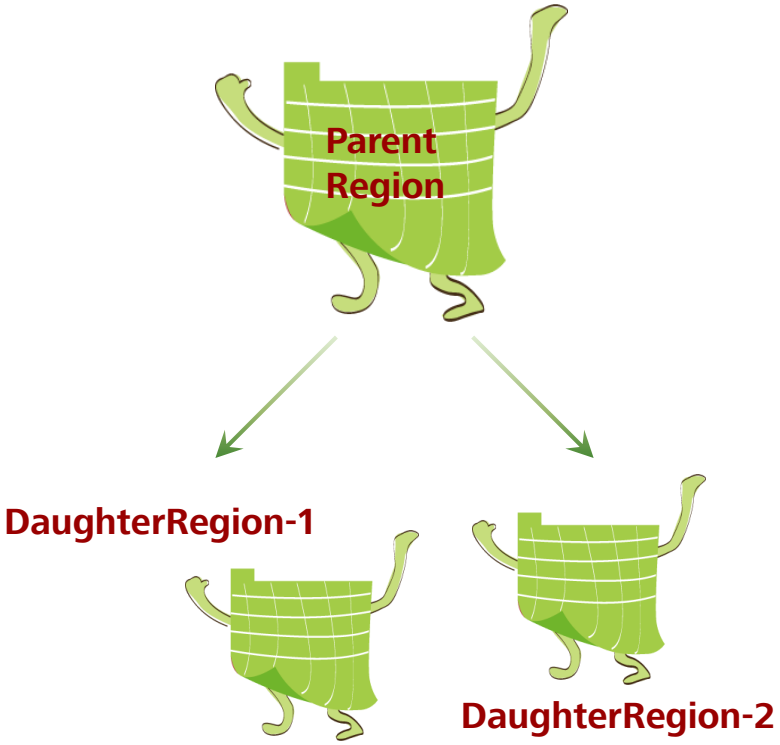
- **Compaction**的主要目的，是为了减少同一个**Region**同一个**ColumnFamily**下面的小文件数目，从而提升读取的性能。
- **Compaction**分为**Minor**、**Major**两类：
  - **Minor:**小范围的**Compaction**。有最少和最大文件数目限制。通常会选择一些连续时间范围的小文件进行合并。
  - **Major:**涉及该**Region**该**ColumnFamily**下面的所有的**HFile**文件。**Major Compaction**过程中，会清理被删除的数据。
- **Minor Compaction**选取文件时，遵循一定的算法。

写流程 - Compaction

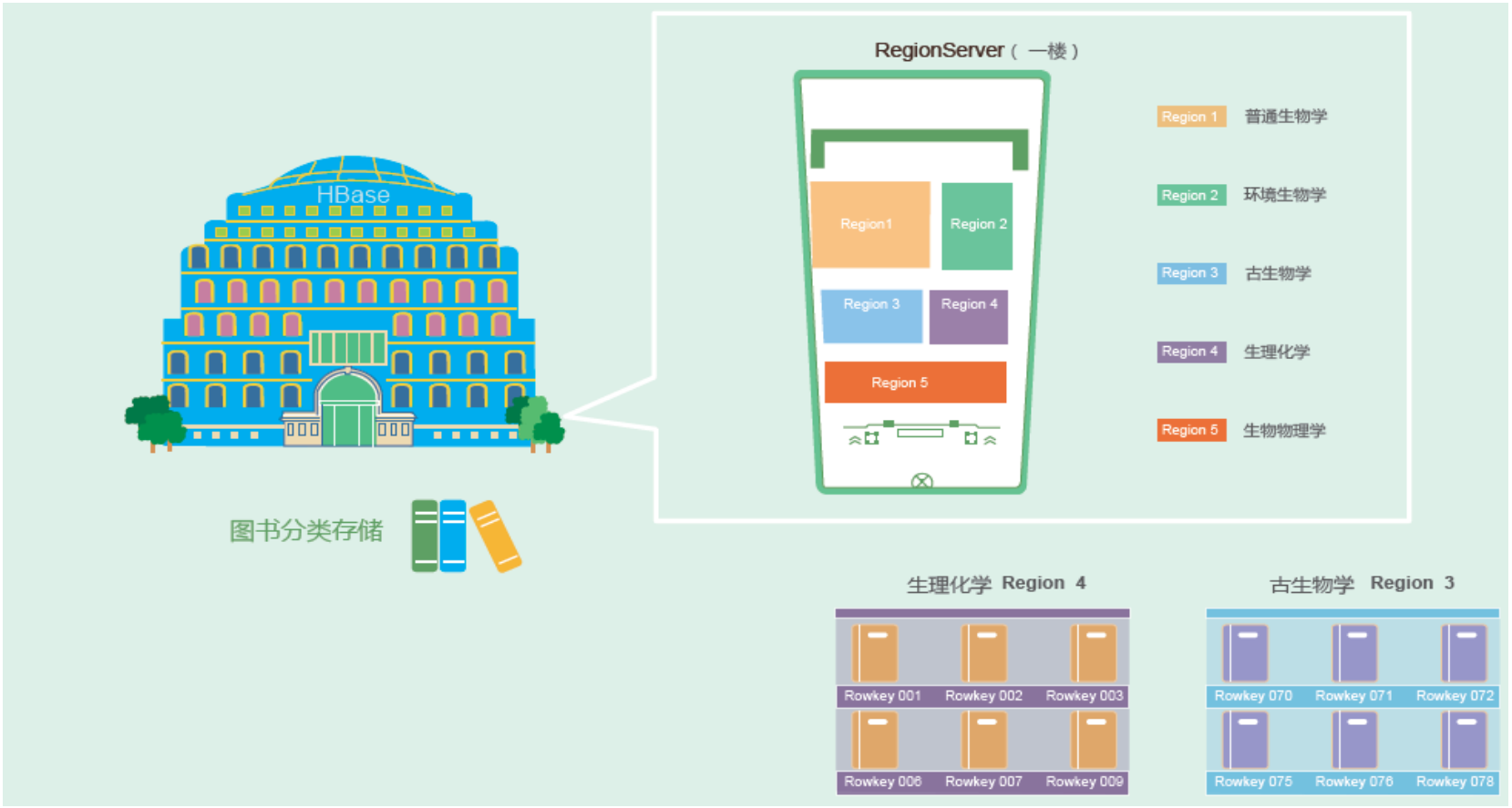


写流程 - Split

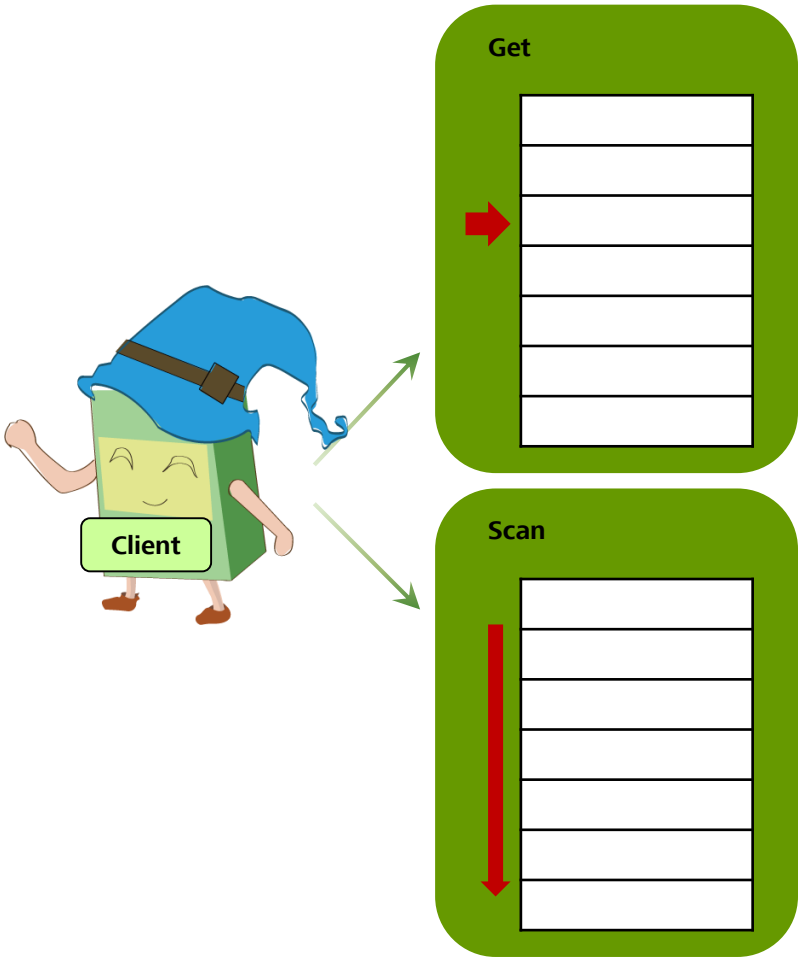
- 普通的**Region Split**操作，是指集群运行期间，某一个**Region**的数据大小超出了预设的阈值，则需要将该**Region**自动分裂成为两个子**Region**。
- **分裂过程中，被分裂的Region会暂停读写服务。**由于分裂过程中，父**Region**的数据文件并不会真正的分裂并重写到两个子**Region**中，而是仅仅通过在新**Region**中创建引用文件的方式，来实现快速的分裂。因此，**Region**暂停服务的时间会比较短暂。
- 客户端侧所缓存的父**Region**的路由信息需要被更新。



读流程

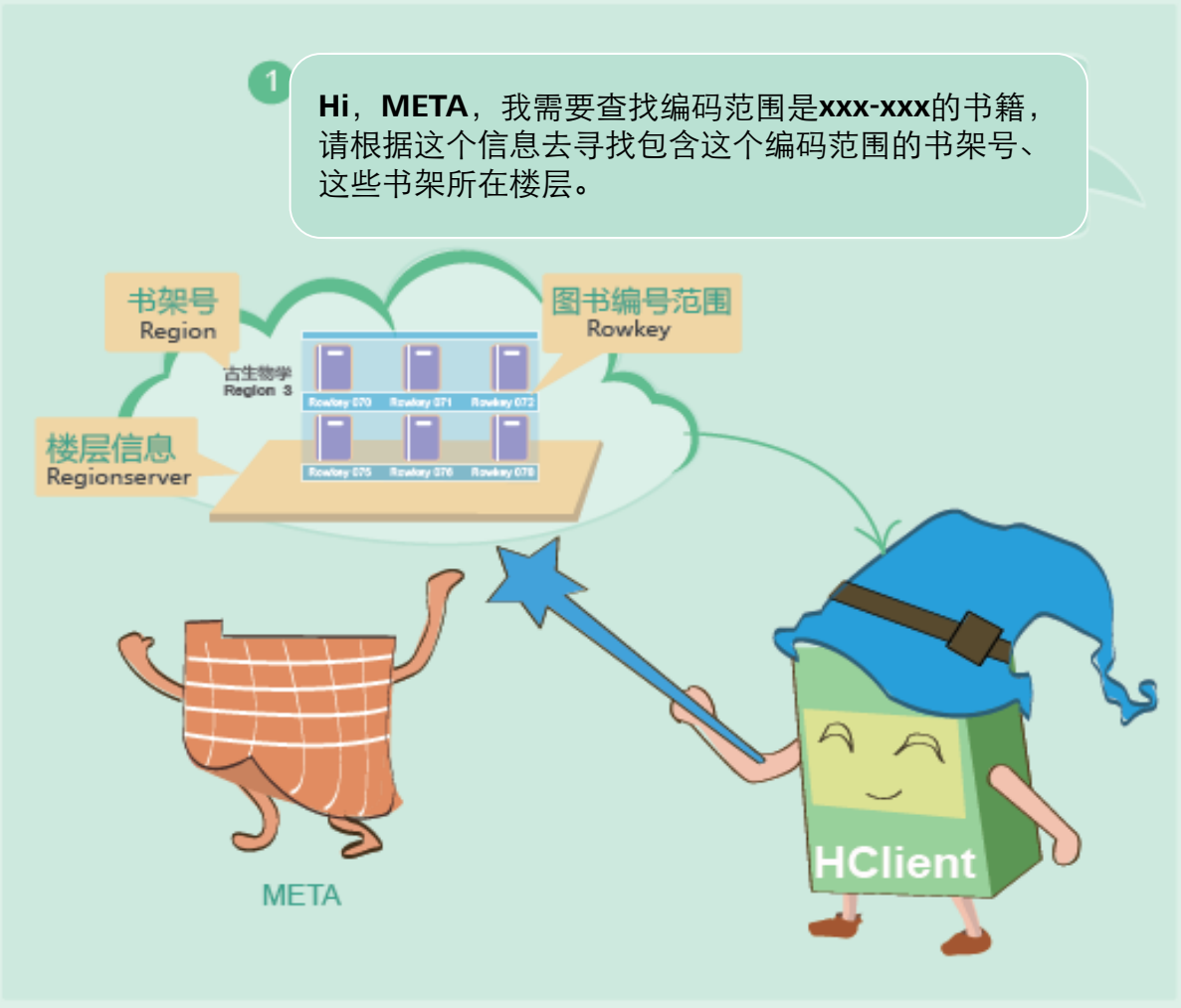


A. 客户端发起读数据请求

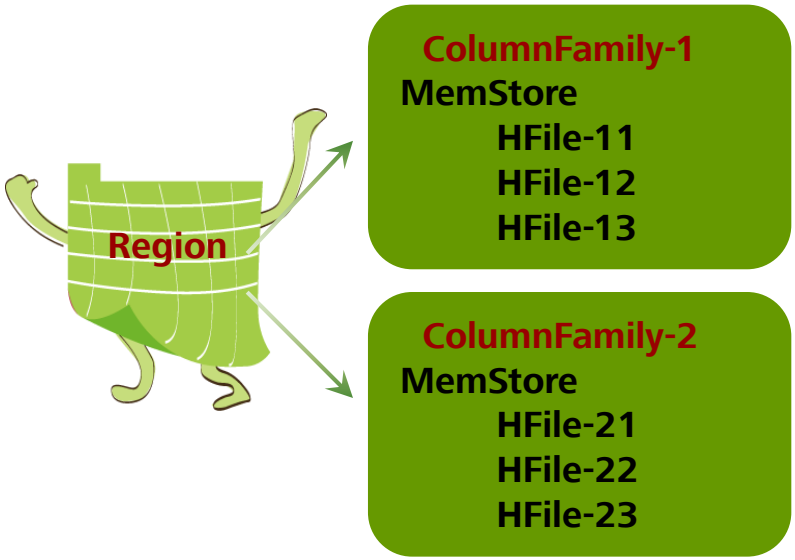


- **Get**操作在提供精确的**Key**值的情形下，读取单行用户数据。
- **Scan**操作是为了批量扫描限定**Key**值范围内的用户数据。

B. 读流程 - 定位Region



C. 读流程 - OpenScanner



OpenScanner的过程，会为**MemStore**，以及各个**HFile**创建所对应的**Scanner**：

- **MemStore**对应的**Scanner**为**MemStoreScanner**。
- **HFile**对应的**Scanner**为**StoreFileScanner**。

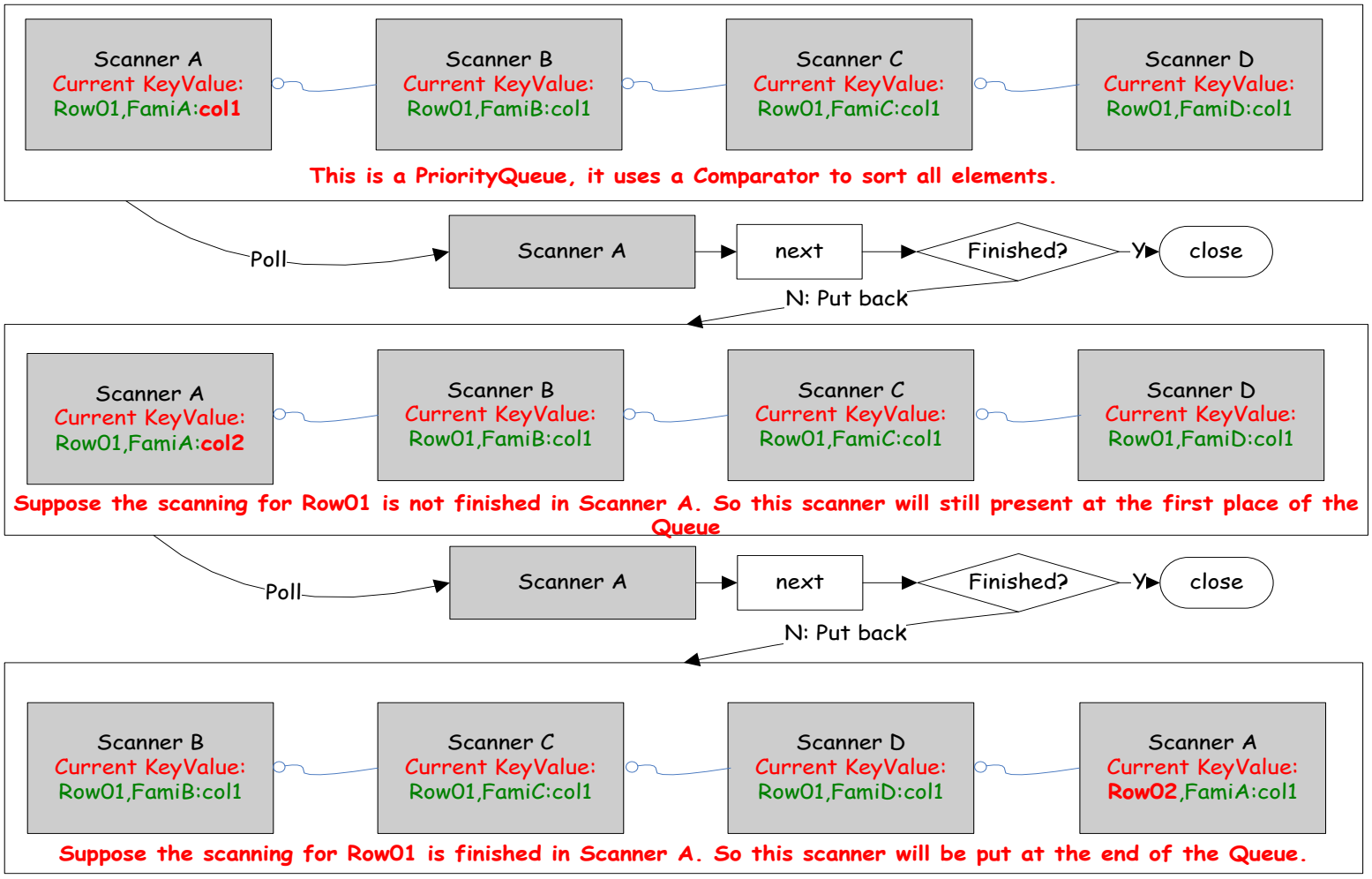
思考：

一个**Region**可能有多个列族…

一个列族，可能包含有多个**HFile**文件，同时，还有部分数据存在于**MemStore**中，尚未固化…

如何读取，才可以读到想要的用户数据?

D. 读流程 - Next

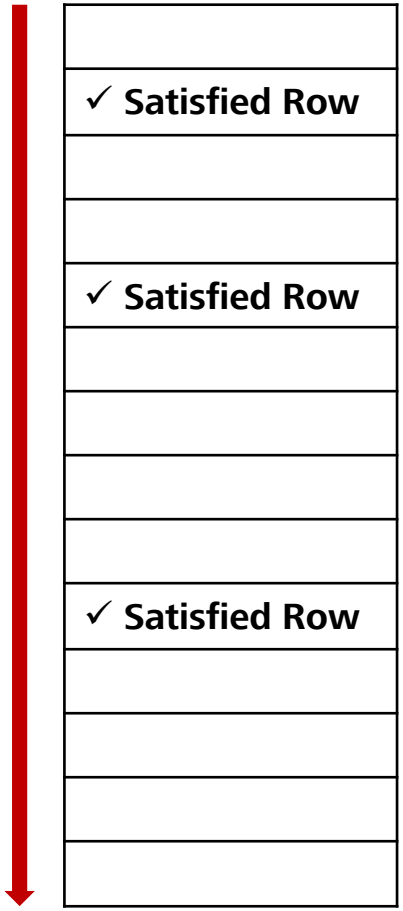


D. 读流程 - Next

- 每一个**Scanner**中，都有一个指针，指向接下来要读取的用户数据**KeyValue**是哪一个。
- 同一级的**Scanner**，被放在同一个优先级队列中。通过不断的对比每一个**Scanner**的指针所指向的**KeyValue**，将这些**Scanner**进行排序。
- 每一次**next**请求，都是从该优先级队列中，**Poll**出一个**Scanner**，然后，读取该**Scanner**的当前指针所指向的**KeyValue**即可。
- 每读一个**Scanner**，指针都会往下移一个**KeyValue**。而后，该**Scanner**被返还到队列中。如果已经读完，则直接关闭。

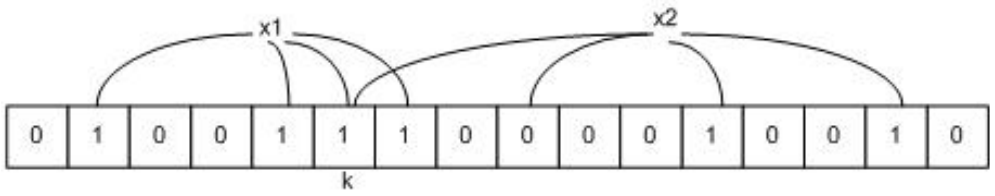
读流程 - Filter

- **Filter**允许在**Scan**过程中，设定一定的过滤条件。符合条件的用户数据才返回。
- 当前包含的一些典型的**Filter**有：
  - **RowFilter**
  - **SingleColumnValueFilter**
  - **KeyOnlyFilter**
  - **FilterList**
  - **……**
- 使用**Filter**时，可能会扫描大量的用户数据，才可以找到所期望的满足条件的数据。因此，一些场景下的性能是不可预估的。

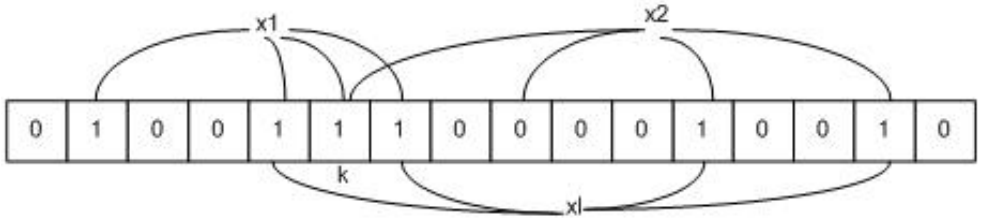


读流程 - BloomFilter

- **BloomFilter**被用来优化一些随机读取的场景，即**Get**场景。 它可以被用来快速的判断一条用户数据在一个大的数据集合（该数据集合的大部分数据都没法被加载到内存中）中是否存在。
- **BloomFilter**在判断一个数据是否存在时，拥有一定的误判率。但对于“用户数据 **XXXX**不存在” 的判断结果是可信的。
- **HBase**的**BloomFilter**的相关数据，被保存在**HFile**中。



数据写入阶段，对用户数据进行**N**次**Hash**，并且将映射到的对应的位改为**1**



数据读取阶段，对用户数据进行**N**次**Hash**，判断对应的位是否为**1**

目录

支持二级索引（1）

- **HBase**是一个**Key-Value**类型的分布式存储数据库。每张表的数据，是按照**RowKey**的字典顺序排序的，因此，如果按照某个指定的**RowKey**去查询数据，或者指定某一个**RowKey**范围去扫描数据时，**HBase**可以快速定位到需要读取的数据位置，从而可以高效地获取到所需要的数据。
- 在实际应用中，很多场景是查询某一个列值为**XXX**的数据。**HBase**提供了**Filter**特性去支持这样的查询，它的原理是：按照**RowKey**的顺序，去遍历所有可能的数据，再依次去匹配那一列的值，直到获取到所需要的数据。可以看出，可能仅仅为了获取一行数据，它却扫描了很多不必要的数据。因此，如果对于这样的查询请求非常频繁并且对查询性能要求较高，使用**Filter**无法满足这个需求。

这就是**HBase**二级索引产生的背景。二级索引为**HBase**提供了按照某些列的值进行索引的能力。

支持二级索引（2）

- 没有二级索引时，查找手机号=“186×××”的记录，必须按RowKey做全表扫描，逐行匹配“mobile”字段，时延大；

- 有二级索引时，先查索引表，再定位到数据表中的位置，不用全表扫描，时延小；

	ColumnFamily A			ColumnFamily B	
RowKey	A: Name	A: Addr	A: Age	B: Mobile	B: Email
001	张三	北京	35	18623532	……
002	李四	上海	27	18623542	……
003	王五	深圳	29	18635355	……
004	赵六	佛山	30	18623497	……
005	……	武汉	28	18623414	……
006	……	西安	32	18612234	……
007	……	广州	23	1861234	……
008	……	成都	……	……	……
009					

数据表

Mobile	RowKey
18623532	……
18623542	……
18635355	……
18623497	……
18623414	……
18612234	……
1861234	……
……	……

二级索引表

	ColumnFamily A			ColumnFamily B	
RowKey	A: Name	A: Addr	A: Age	B: Mobile	B: Email
001	张三	北京	35	18623532	……
002	李四	上海	27	18623542	……
003	王五	深圳	29	18635355	……
004	赵六	佛山	30	18623497	……
005	……	武汉	28	18623414	……
006	……	西安	32	18612234	……
007	……	广州	23	1861234	……
008	……	成都	……	……	……
009					

容灾增强

- 主备集群之间的容灾能力可以增强**HBase**数据的高可用性，主集群提供数据服务，备用集群提供数据备份，当主集群出现故障时，备集群可以提供数据服务。
- **HBase**集群容灾作为提高**HBase**集群系统高可用性的一个关键特性，为**HBase**提供了实时的异地数据容灾功能。它对外提供了基础的运维工具，包含灾备关系维护，重建，数据校验，数据同步进展查看等功能。为了实现数据的实时容灾，可以把本**HBase**集群中的数据备份到另一个集群。

容灾增强

- 相比开源**Replication**功能，做了如下增强：
  - 备集群白名单功能，只接受指定集群**ip**的数据推送。
  - 开源版本中**replication**是基于**WAL**同步，在备集群回放**WAL**实现数据备份的。对于**BulkLoad**，由于没有**WAL**产生，**BulkLoad**的数据不会**replicate**到备集群。通过将**BulkLoad**操作记录在**WAL**上，同步至备集群，备集群通过**WAL**读取**BukLoad**操作记录，将对应的主集群的**HFile**加载到备集群，完成数据的备份。
  - 开源版本中**HBase**对于系统表**ACL**做了过滤，**ACL**信息不会同步至备集群，通过新加一个过滤器***org.apache.hadoop.hbase.replication.SystemTableWALEntryFilterAllowACL***，允许**ACL**信息同步至备集群，用户可以通过配置***hbase.replication.filter.sytemWALEntryFilter***使用该过滤其实现**ACL**同步。
  - 备集群只读限制，备集群只接受备集群节点内的超级用户对备集群的**HBase**进行修改操作，即备集群节点之外的**HBase**客户端只能对备集群的**HBase**进行读操作。

思考题

1. 客户端采用批量写入接口写入10条数据，某个RegionServer节点上包含该表的2个Region，分别A和B，写入的10条数据中有2条属于A，4条属于B，请问需要写入这10条数据需要向该RegionServer发送几次RPC请求?
2. 一张表的包含以下几个分区[10,20),[20,30),[30,+∞),分别编号为①,②,③，那么11，20，222分别属于哪个Region?

习题

1. **HBase的物理存储单元是什么？（　　）**

   A. **Region**

   B. **Column Family**

   C. **Column**

   D. 行

2. **Compaction的目的是什么？**

   A. 减少同一个**Region**同一**Column Family**下的文件数目

   B. 提升数据读取性能

   C. 减少同一个**Column Family**的文件数量

   D. 减少同一个**Region**的文件数量

习题

1. **HBase的物理存储单元是什么？（　　）**

   A. **Region**

   B. **Column Family**

   C. **Column**

   D. 行

2. **Compaction的目的是什么？**

   A. 减少同一个**Region**同一**Column Family**下的文件数目

   B. 提升数据读取性能

   C. 减少同一个**Column Family**的文件数量

   D. 减少同一个**Region**的文件数量

本章总结

学习完本章，应该做到：

1、认识了解**KeyValue**存储；

2、清楚**HBase**的基本架构；

3、了解**HBase**读写流程；

4、了解**FusionInsight HBase**的增强特性。

学习推荐

- 华为**Learning**网站

  http://support.huawei.com/learning/Index!toTrainIndex

- 华为**Support**案例库

  http://support.huawei.com/enterprise/servicecenter?lang=zh

Thank you

www.huawei.com