

HBase原理和架构

主讲人：杨老师

01 HBase概述

02 HBase数据模型

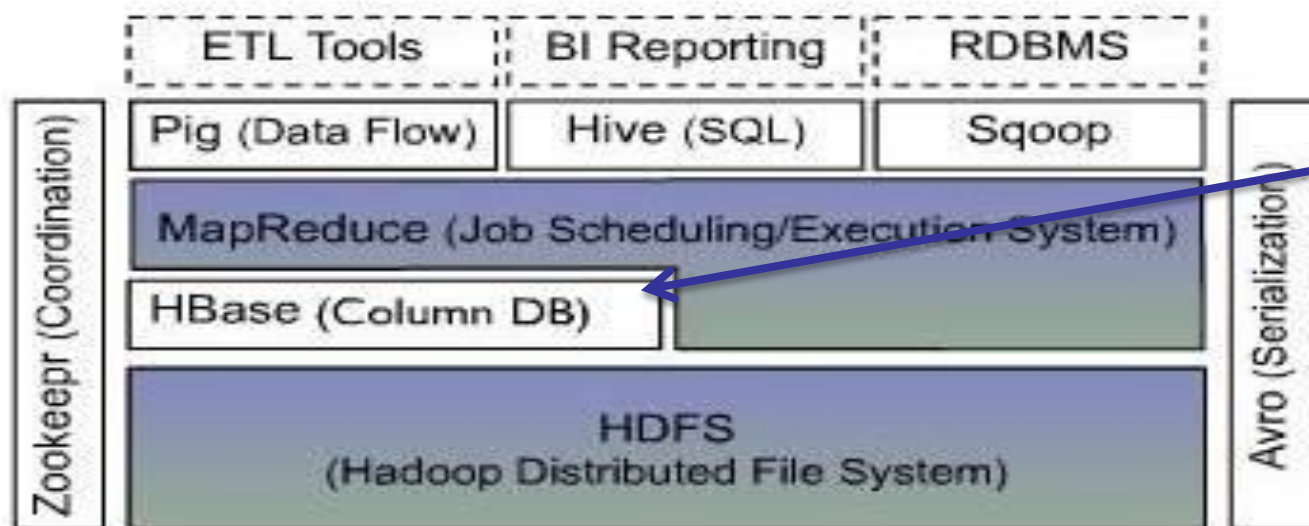
03 HBase物理模型

04 HBase系统架构

01 HBase概述

- HBase是构建在HDFS之上的分布式列存储数据库，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可以在廉价PC Server上搭建起大规模结构化存储集群。
- HBase 是Google Bigtable的开源实现，类似Google Bigtable利用GFS作为其文件存储系统，Google运行MapReduce来处理Bigtable中的海量数据，HBase同样利用Hadoop MapReduce来处理HBase中的海量数据；Google Bigtable利用 Chubby作为协同服务，HBase利用Zookeeper作为对应。

The Hadoop Ecosystem



HBase构建在HDFS之上



HBase内部的文件
全部存储在HDFS上

- 上图描述了Hadoop EcoSystem中的各层系统，其中HBase位于结构化存储层，Hadoop HDFS为HBase提供了高可靠性的底层存储支持，Hadoop MapReduce为HBase提供了高性能的计算能力，Zookeeper为HBase提供了稳定服务和failover（故障切换）机制。
- 此外，Pig和Hive还为HBase提供了高层语言支持，使得在HBase上进行数据统计处理变的非常简单。Sqoop则为HBase提供了方便的RDBMS数据导入功能，使得传统数据库数据向HBase中迁移变的非常方便

- 两者都具有良好的容错性和扩展性，都可以扩展到成百上千个节点
- HDFS适合批处理场景
 - ✓ 不支持随机随机查找（全文件扫描）
 - ✓ 不适合增量数据处理（处理整个文件，不能处理其中一部分）
 - ✓ 不支持数据更新（一旦存储，无法修改）
- HBase是对HDFS很好的补充
 - ✓ 可以插入一条数据
 - ✓ 可以删除一条数据
 - ✓ 可以随机查询、过滤数据

- 大：单表可以数十亿行，数百万列
- 无模式：同一个表的不同行可以有截然不同的列
- 面向列：存储、权限控制、检索均面向列
- 稀疏：空列不占用存储，表是稀疏的
- 多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元格插入时的时间戳
- 数据类型单一：数据都是字符串，没有类型

contract	client	date	name	price	city	product
12302346	10042334		Eno		Redmond	Car
37611373	10007007		Gotz		Redmond	House
51213123	10032423		Jones		Washington	Travel
54535545	10087023		Smith		New York	House
45447004	10013232		Doe		Boston	Car
95371001	10032112		Chen		Seattle	House

old	contract
1000	12302346
1001	37611373
1002	51213123
1003	54535545
1004	45447004
1005	95371001

old	client
1000	10042334
1001	10007007
1002	10032423
1003	10087023
1004	10013232
1005	10032112

old	name
1000	Eno
1001	Gotz
1002	Jones
1003	Smith
1004	Doe
1005	Chen

old	city
1000	Redmond
1001	Redmond
1002	Washington
1003	New York
1004	Boston
1005	Seattle

old	product
1000	Car
1001	House
1002	Travel
1003	House
1004	Car
1005	House

- 数据是按行存储的
 - 没有索引的查询使用大量I/O
 - 建立索引和物化视图需要花费大量时间和资源
 - 面向查询的需求，数据库必须被大量膨胀才能满足性能要求
-
- 数据是按列存储-每一列单独存放
 - 数据即是索引
 - 只访问查询涉及的列-大量降低系统I/O
 - 每一列由一个线索来处理-查询的并发处理
 - 数据类型一致，数据特征相似-高效压缩

02 HBase数据模型

- Table & Column Family

Row Key	version	Column Family	
		URI	Parser
r1	t2	url=http://www.taobao.com	title=天天特价
	t1	host=taobao.com	
r2	t5	url=http://www.alibaba.com	content=每天...
	t4	host=alibaba.com	

- Ø Row Key: 行键，Table的主键，Table中的记录按照Row Key排序
- Ø version: 版本，每次数据操作对应的时间戳，可以看作是数据的version number
- Ø Column Family: 列族，Table在水平方向有一个或者多个Column Family组成，一个Column Family中可以由任意多个Column组成，即Column Family支持动态扩展，无需预先定义Column的数量以及类型，所有Column均以二进制格式存储，用户需要自行进行类型转换。

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Different types of data separated into different "column families"

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Useful for *-To-Many mappings

<http://blog.csdn.net/woshiwanxin102213>

每条记录被划分到若干个 Column Family 中

每一行与一个 Rowkey

PERSON TABLE					
row key	personal_data		demographic		...
PersonID	Name	Address	BirthDate	Gender	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2. Census Data in Column Families

每个column family 由一个或者多个 Column 组成


- HBase schema可以有多个 *Table*
- 每个表可由多个 *Column Family*组成
- HBase 可以有 *Dynamic Column*
 - ✓ 列名称是编码在cell中的
 - ✓ 不同的cell可以拥有不同的列

“Roles” column family has
different columns in
different cells



Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

- *version number* 可由用户提供
 - ✓ 无需以递增的顺序插入
 - ✓ 每一行的rowkey必须是唯一的
- **Table** 可能非常稀疏
 - ✓ 很多 cell 可以是空的
- *Row Key* 是主键



Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

- 所有操作均是基于rowkey的；
- 支持CRUD（Create、Read、Update和Delete）和Scan；
- 单行操作
 - ✓ Put
 - ✓ Get
 - ✓ Scan—含头不含尾
- 多行操作
 - ✓ Scan
 - ✓ MultiPut
- 没有内置join操作，可使用MapReduce解决。

03 HBase物理模型

- 每个column family存储在HDFS上的一个单独的文件里
- Rowkey和version在每个column family里均有一份
- 空值不保存，占位符都没有

HBase 为每个值维护了多级索引，即：
<key, column family, column name, timestamp>

Table 5.3. ColumnFamily contents

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Table 5.2. ColumnFamily anchor

Row Key	Time Stamp	Column Family anchor
"com.cnn.www"	t9	anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

info Column Family

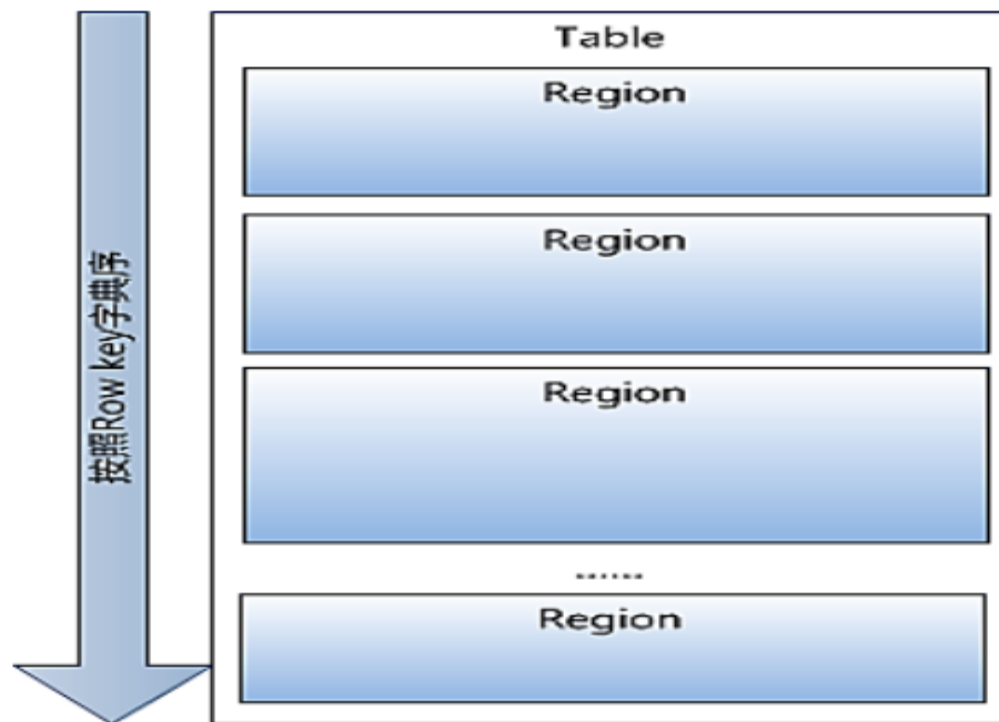
Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

roles Column Family

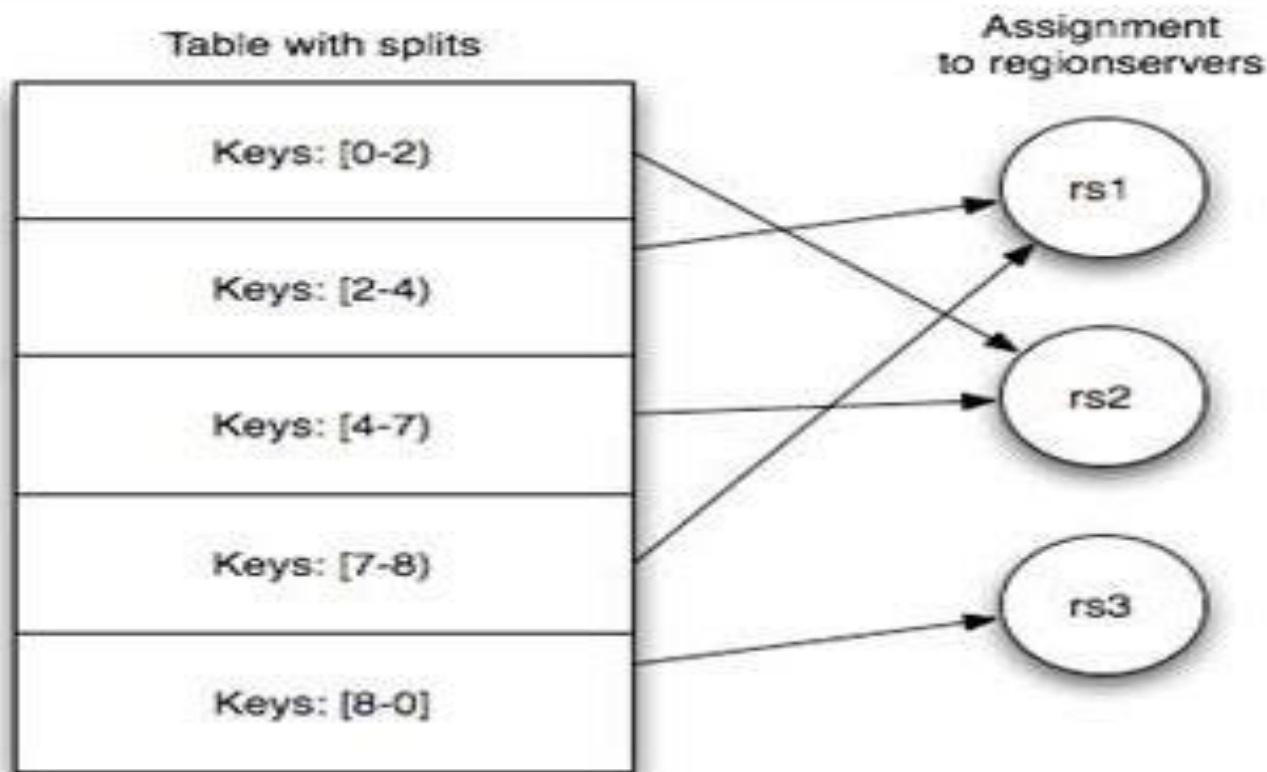
Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

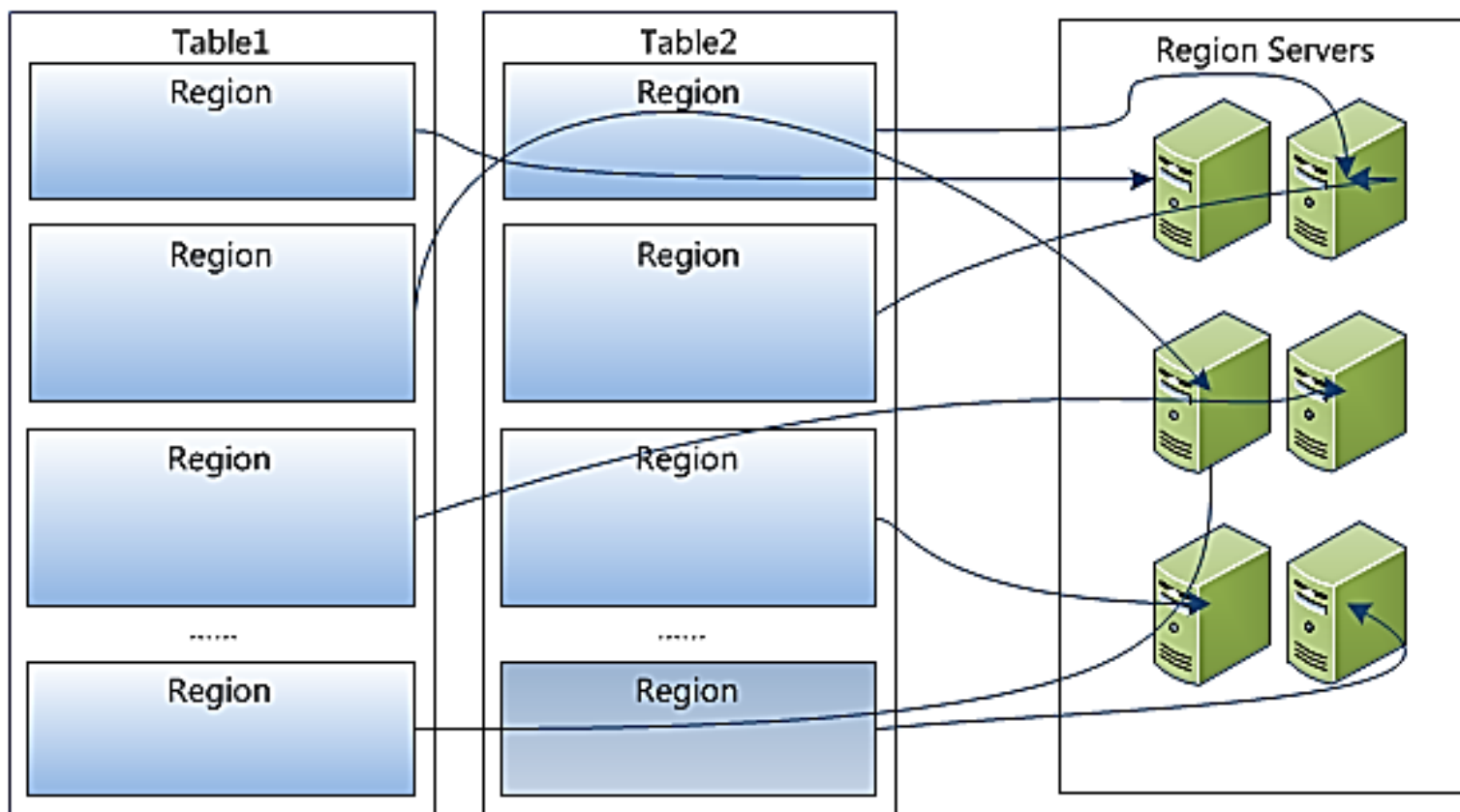
- Table中的所有行都按照row key的字典序排列；
- Table 在行的方向上分割为多个Region；



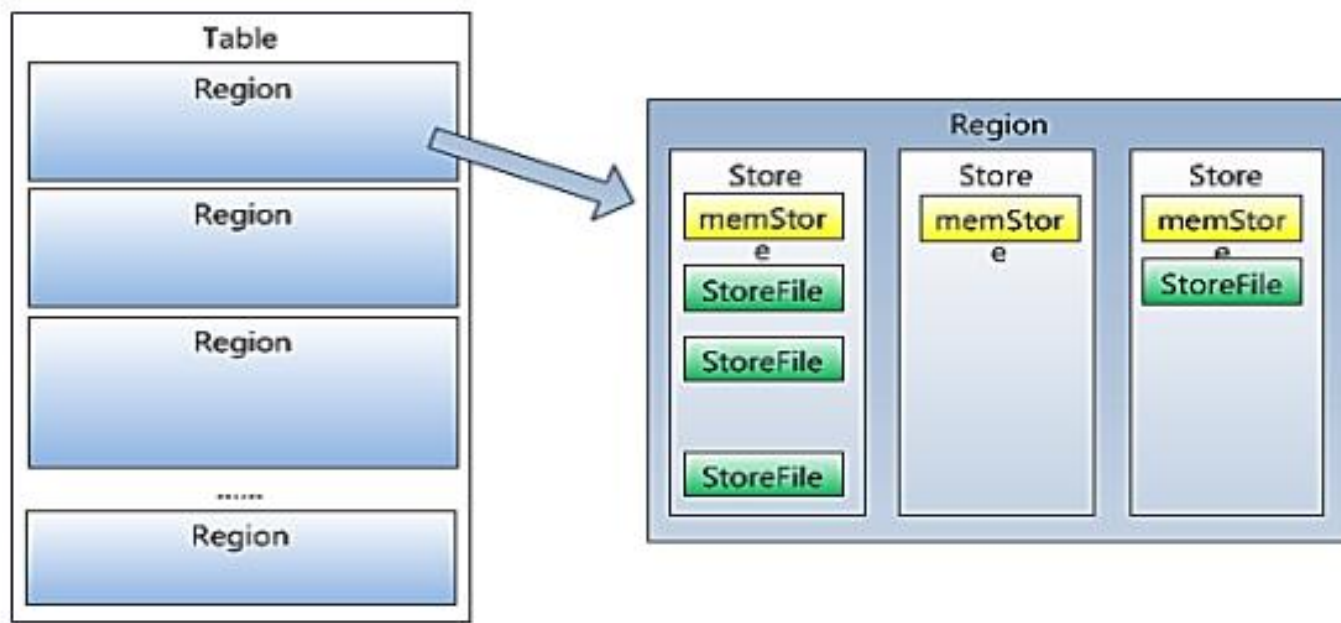
- **Table**默认最初只有一个**Region**，随着记录数不断增加而变大后，会逐渐分裂成多个**region**，一个**region**由[startkey,endkey]表示，不同的**region**会被**Master**分配给相应的**RegionServer**进行管理



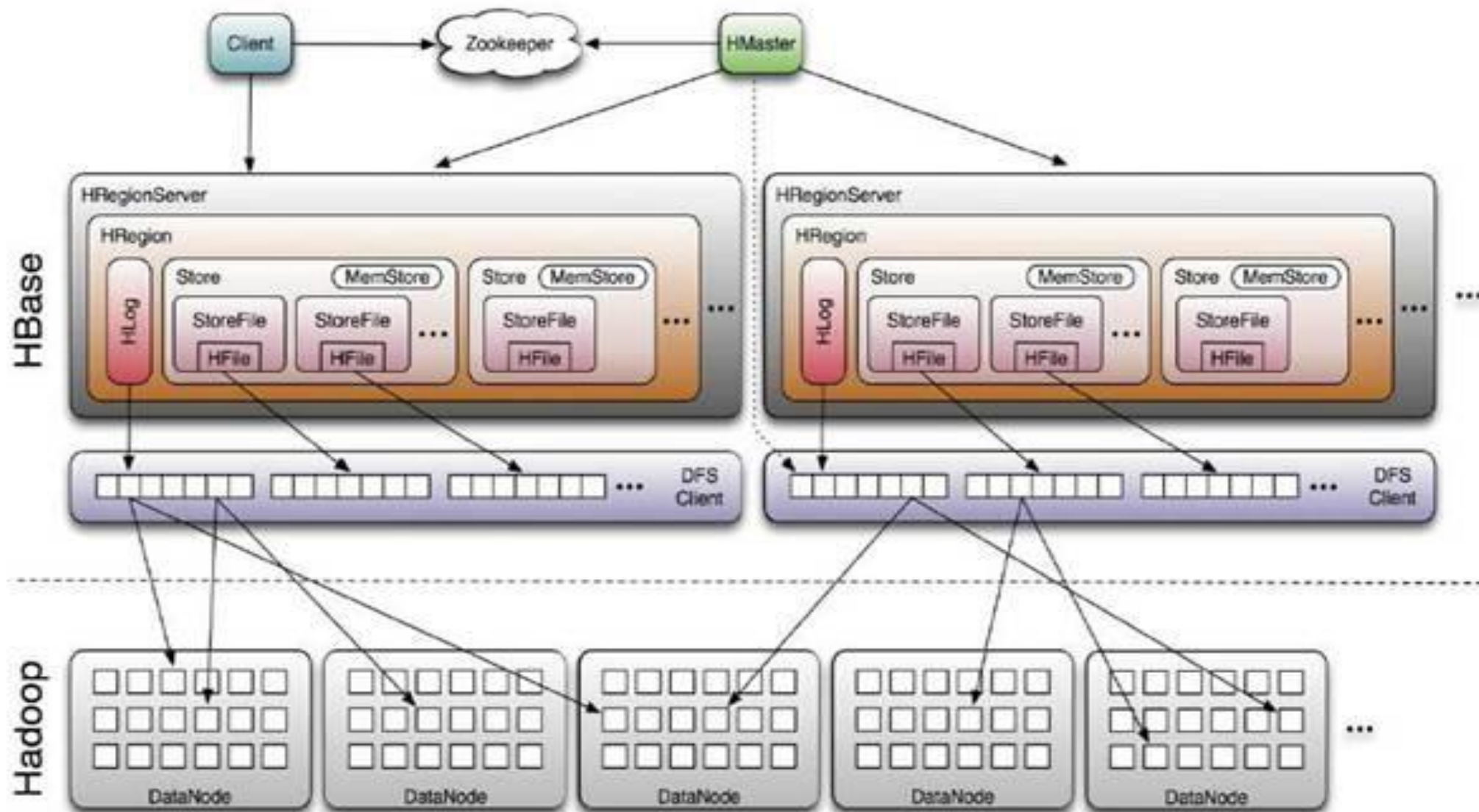
- Region是HBase中分布式存储和负载均衡的最小单元。不同Region分布到不同RegionServer上



- Region虽然是分布式存储的最小单元，但并不是存储的最小单元。
- Region由一个或者多个Store组成，每个store保存一个columns family
- 每个Store又由一个memStore和0至多个StoreFile组成；
- memStore存储在内存中，StoreFile存储在HDFS上。



04 HBase系统架构



- Client
 - ✓ 包含访问HBase的接口，并维护cache来加快对HBase的访问
- Zookeeper
 - ✓ 保证任何时候，集群中只有一个active master
 - ✓ 存贮所有Region的寻址入口
 - ✓ 实时监控Region server的上线和下线信息。并实时通知给Master
 - ✓ 存储HBase的schema和table元数据
- Master
 - ✓ 为Region server分配region
 - ✓ 负责Region server的负载均衡
 - ✓ 发现失效的Region server并重新分配其上的region
 - ✓ HDFS上的垃圾文件回收
 - ✓ 处理schema更新请求
- Region Server
 - ✓ Region server维护region，处理对这些region的IO请求
 - ✓ Region server负责切分在运行过程中变得过大的region

Client :

- HBase Client使用HBase的RPC机制与HMaster和HRegionServer进行通信。对于管理类操作（通过Java api 建表），Client与HMaster进行RPC；对于数据读写类操作，Client与HRegionServer进行RPC

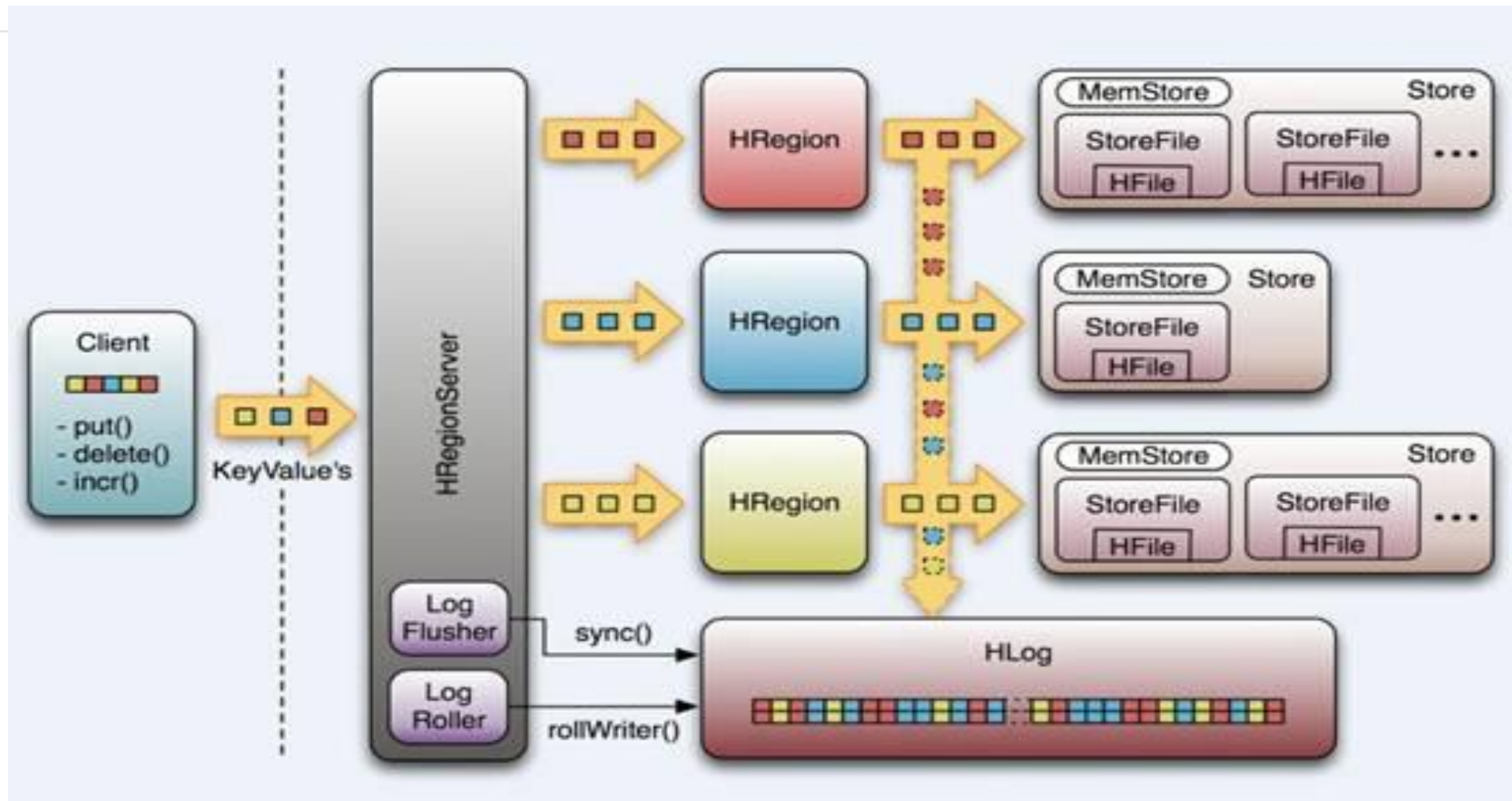
Zookeeper :

- Zookeeper Quorum中除了存储了-ROOT-表的地址和HMaster的地址，HRegionServer也会把自己以Ephemeral方式注册到Zookeeper中，使得HMaster可以随时感知到各个HRegionServer的健康状态。此外，Zookeeper也避免了HMaster的单点问题，见下文描述

Hmaster :

- HMaster没有单点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master运行，HMaster在功能上主要负责Table和Region的管理工作：
 1. 管理用户对Table的增、删、改、查操作
 2. 管理HRegionServer的负载均衡，调整Region分布
 3. 在Region Split后，负责新Region的分配
 4. 在HRegionServer停机后，负责失效HRegionServer 上的Regions迁移

讲 Hbase Write-Ahead-Log (预先写日志)



- HRegionServer内部管理了一系列HRegion对象，每个HRegion对应了Table中的一个Region，HRegion中由多个HStore组成。每个HStore对应了Table中的一个Column Family的存储，可以看出每个Column Family其实就是一个集中的存储单元，因此最好将具备共同IO特性的column放在一个Column Family中，这样最高效。
- HStore存储是HBase存储的核心了，其中由两部分组成，一部分是MemStore，一部分是StoreFiles。MemStore是Sorted Memory Buffer，用户写入的数据首先会放入MemStore，当MemStore满了以后会Flush成一个StoreFile（底层实现是HFile），当StoreFile文件数量增长到一定阈值，会触发Compact合并操作，将多个StoreFiles合并成一个StoreFile，合并过程中会进行版本合并和数据删除，因此可以看出HBase其实只有增加数据，所有的更新和删除操作都是在后续的compact过程中进行的，这使得用户的写操作只要进入内存中就可以立即返回，保证了HBase I/O的高性能。

HBase Compact & Split

当StoreFiles Compact后，会逐步形成越来越大的StoreFile，当单个StoreFile大小超过一定阈值后，会触发Split操作，同时把当前Region Split成2个Region，父Region会下线，新Split出的2个孩子Region会被HMaster分配到相应的HRegionServer上，使得原先1个Region的压力得以分流到2个Region上。下图描述了Compaction和Split的过程：



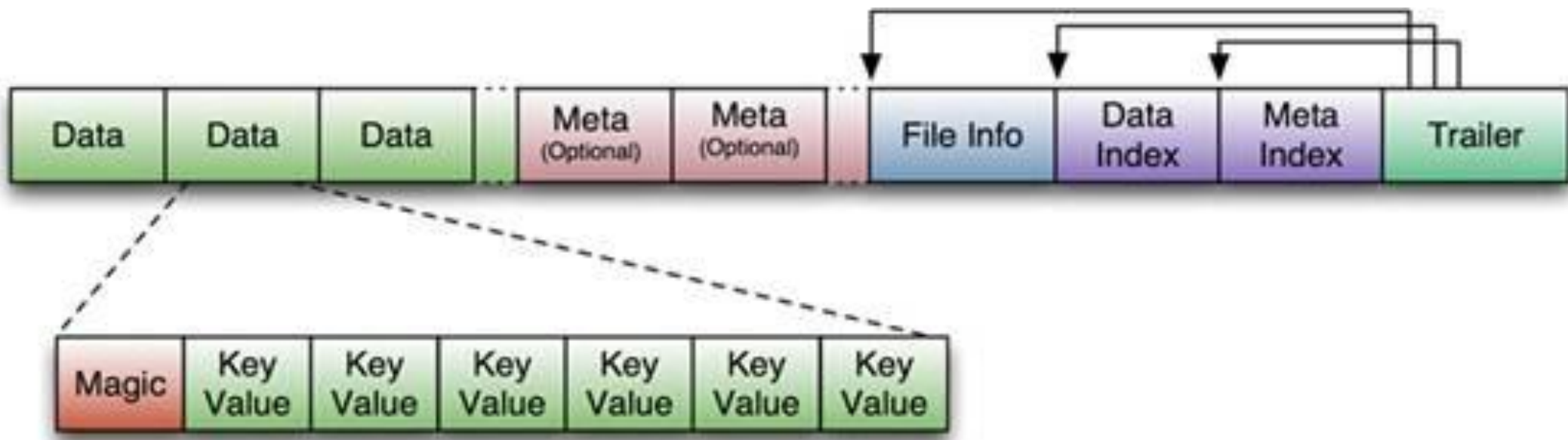
- 在理解了上述HStore的基本原理后，还必须了解一下HLog的功能，因为上述的HStore在系统正常工作的前提下是没有问题的，但是在分布式系统环境中，无法避免系统出错或者宕机，因此一旦HRegionServer意外退出，MemStore中的内存数据将会丢失，这就需要引入HLog了。每个HRegionServer中都有一个HLog对象，HLog是一个实现Write Ahead Log的类，在每次用户操作写入MemStore的同时，也会写一份数据到HLog文件中（HLog文件格式见后续），HLog文件定期会滚动出新的，并删除旧的文件（已持久化到StoreFile中的数据）。当HRegionServer意外终止后，HMaster会通过Zookeeper感知到，HMaster首先会处理遗留的HLog文件，将其中不同Region的Log数据进行拆分，分别放到相应region的目录下，然后再将失效的region重新分配，领取到这些region的HRegionServer在Load Region的过程中，会发现历史HLog需要处理，因此会Replay HLog中的数据到MemStore中，然后flush到StoreFiles，完成数据恢复。

讲 Hfile存储格式

HBase中的所有数据文件都存储在Hadoop HDFS文件系统上，主要包括上述提出的两种文件类型：

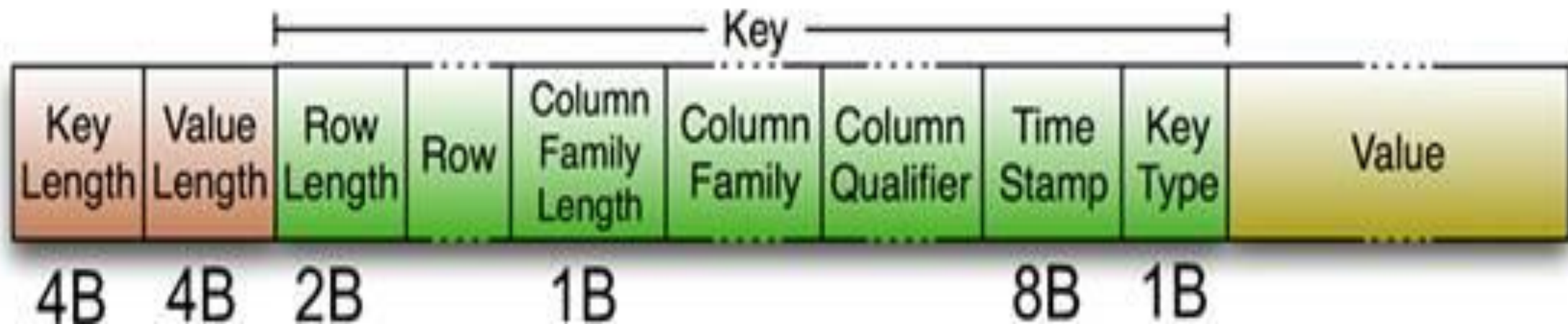
- **HFile**：HBase中KeyValue数据的存储格式，HFile是Hadoop的二进制格式文件，实际上StoreFile就是对HFile做了轻量级包装，即StoreFile底层就是HFile
- **HLog File**：HBase中WAL（Write Ahead Log）的存储格式，物理上是Hadoop的Sequence File

下图是HFile的存储格式：



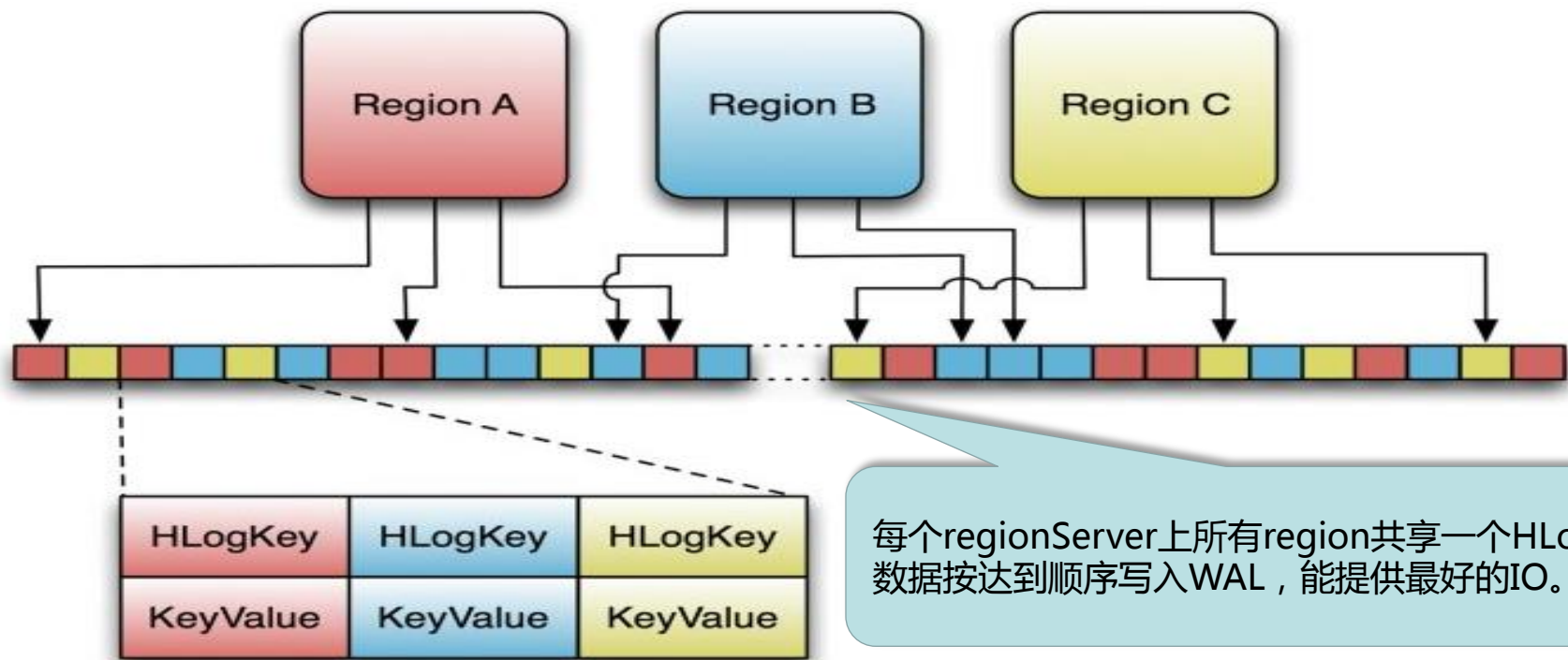
- 首先HFile文件是不定长的，长度固定的只有其中的两块：Trailer和FileInfo。正如图所示的，Trailer中有指针指向其他数据块的起始点。File Info中记录了文件的一些Meta信息，例如：AVG_KEY_LEN, AVG_VALUE_LEN, LAST_KEY, COMPARATOR, MAX_SEQ_ID_KEY等。Data Index和Meta Index块记录了每个Data块和Meta块的起始点。
- Data Block是HBase I/O的基本单元，为了提高效率，HRegionServer中有基于LRU的Block Cache机制。每个Data块的大小可以在创建一个Table的时候通过参数指定，大号的Block有利于顺序Scan，小号Block利于随机查询。每个Data块除了开头的Magic以外就是一个个KeyValue对拼接而成，Magic内容就是一些随机数字，目的是防止数据损坏。后面会详细介绍每个KeyValue对的内部构造。

- HFile里面的每个KeyValue对就是一个简单的byte数组。但是这个byte数组里面包含了很多项，并且有固定的结构。我们来看看里面的具体结构：

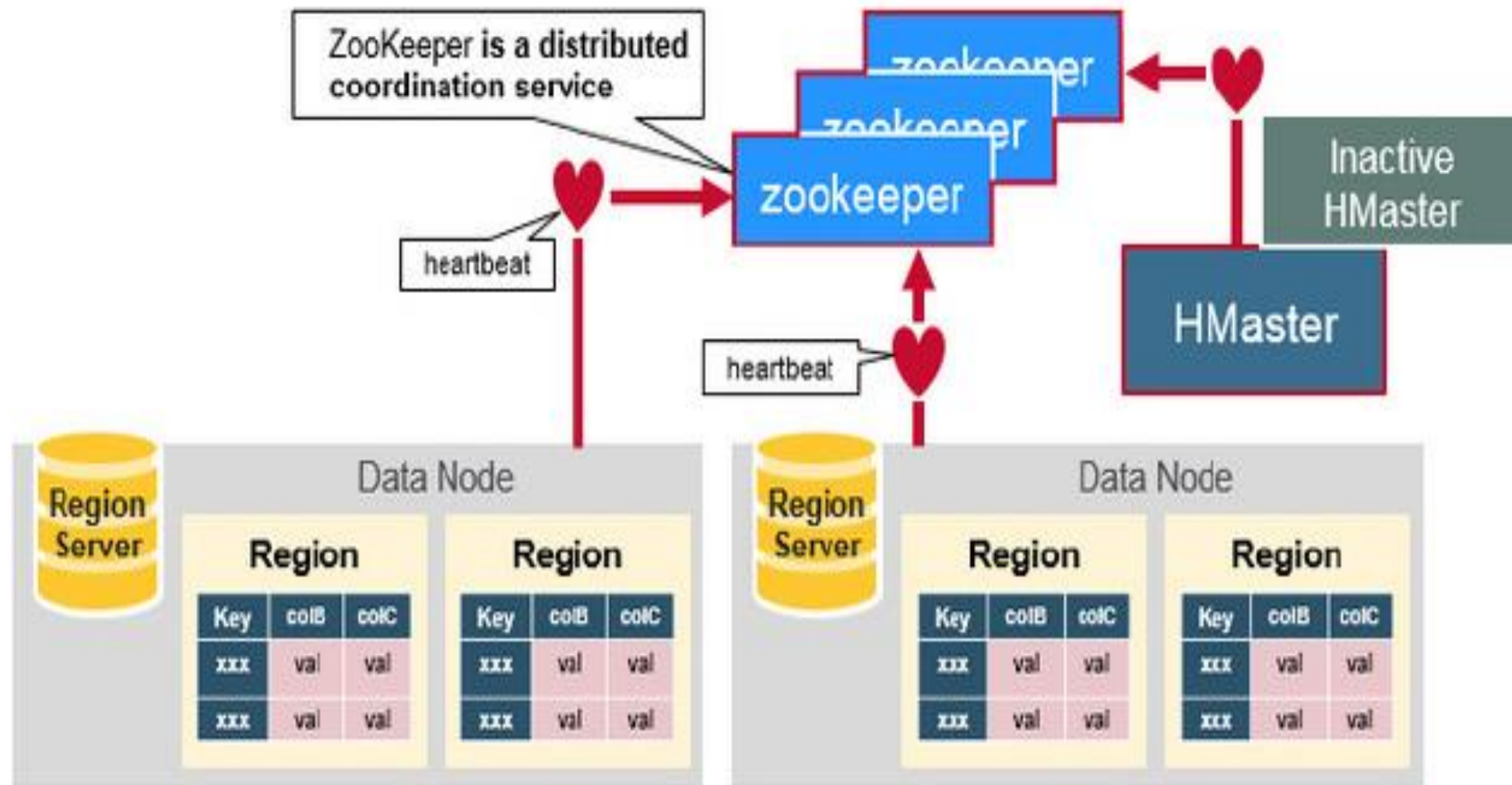


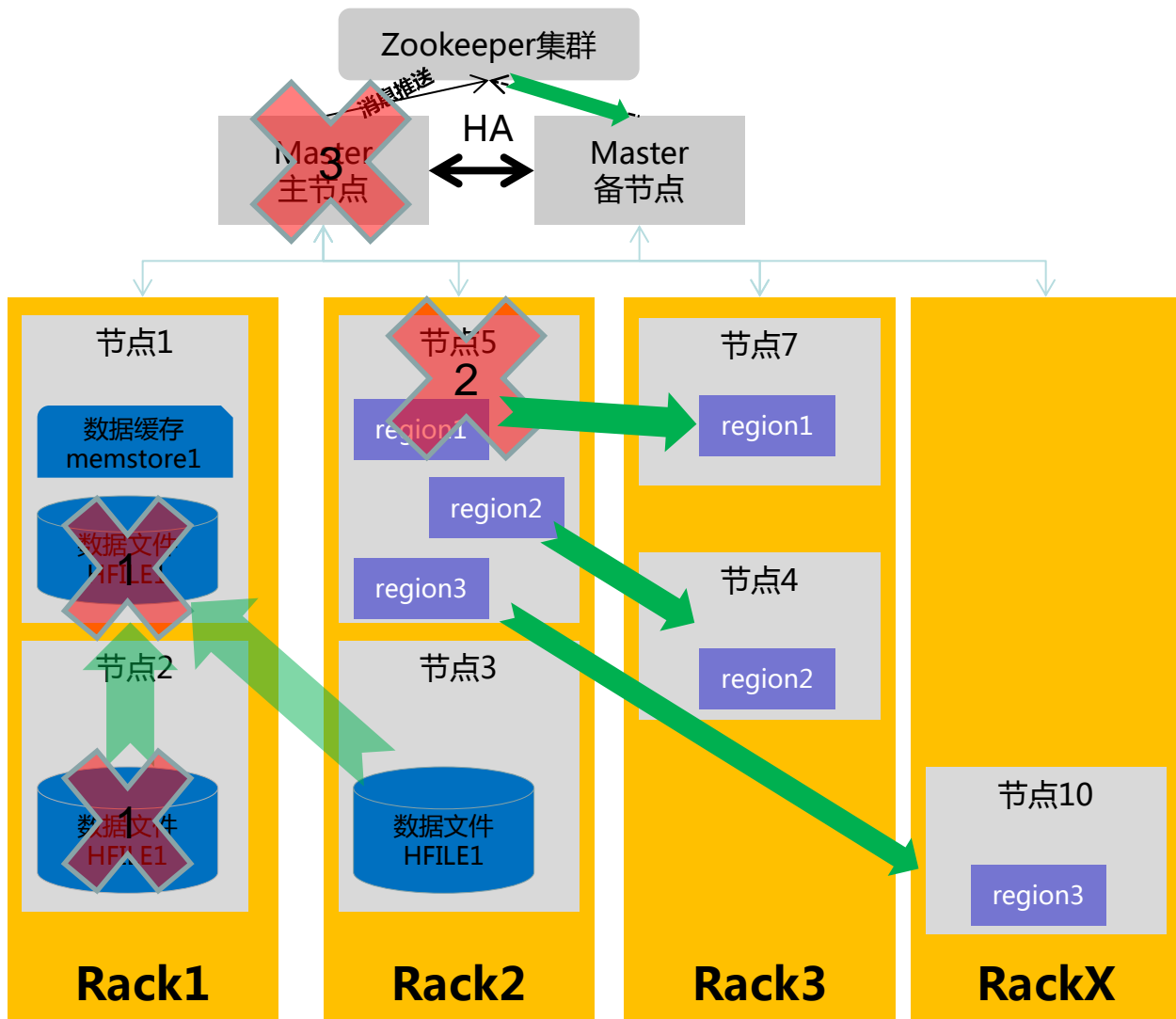
- 开始是两个固定长度的数值，分别表示Key的长度和Value的长度。紧接着是Key，开始是固定长度的数值，表示RowKey的长度，紧接着是RowKey，然后是固定长度的数值，表示Family的长度，然后是Family，接着是Qualifier，然后是两个固定长度的数值，表示Time Stamp和Key Type (Put/Delete)。Value部分没有这么复杂的结构，就是纯粹的二进制数据了。

WAL(Write-Ahead-Log): regionserver在处理插入和删除过程中用来记录操作内容的日志，只有日志写入成功，才会通知客户端操作成功。



- 上图中示意了HLog文件的结构，其实HLog文件就是一个普通的Hadoop Sequence File，Sequence File 的Key是HLogKey对象，HLogKey中记录了写入数据的归属信息，除了table和region名字外，同时还包括 sequence number和timestamp，timestamp是“写入时间”，sequence number的起始值为0，或者是最近一次存入文件系统中sequence number。
- HLog Sequence File的Value是HBase的KeyValue对象，即对应HFile中的KeyValue，可参见上文描述。





1 HDFS机架识别策略

✓当数据文件损坏时，会找相同机架上备份的数据文件，如果相同机架上的数据文件也损坏会找不同机架备份数据文件。

2 HBASE的REGION快速恢复

✓当节点损坏时，节点上的丢失的region，会在其他节点上均匀快速恢复。

3 Master节点的HA机制

✓Master为一主多备，当Master主节点宕机后，剩下的备节点通过选举，产生主节点。

➤ **Master容错：Zookeeper重新选择一个新的Master**

无Master过程中，数据读取仍照常进行；

无master过程中，region切分、负载均衡等无法进行；

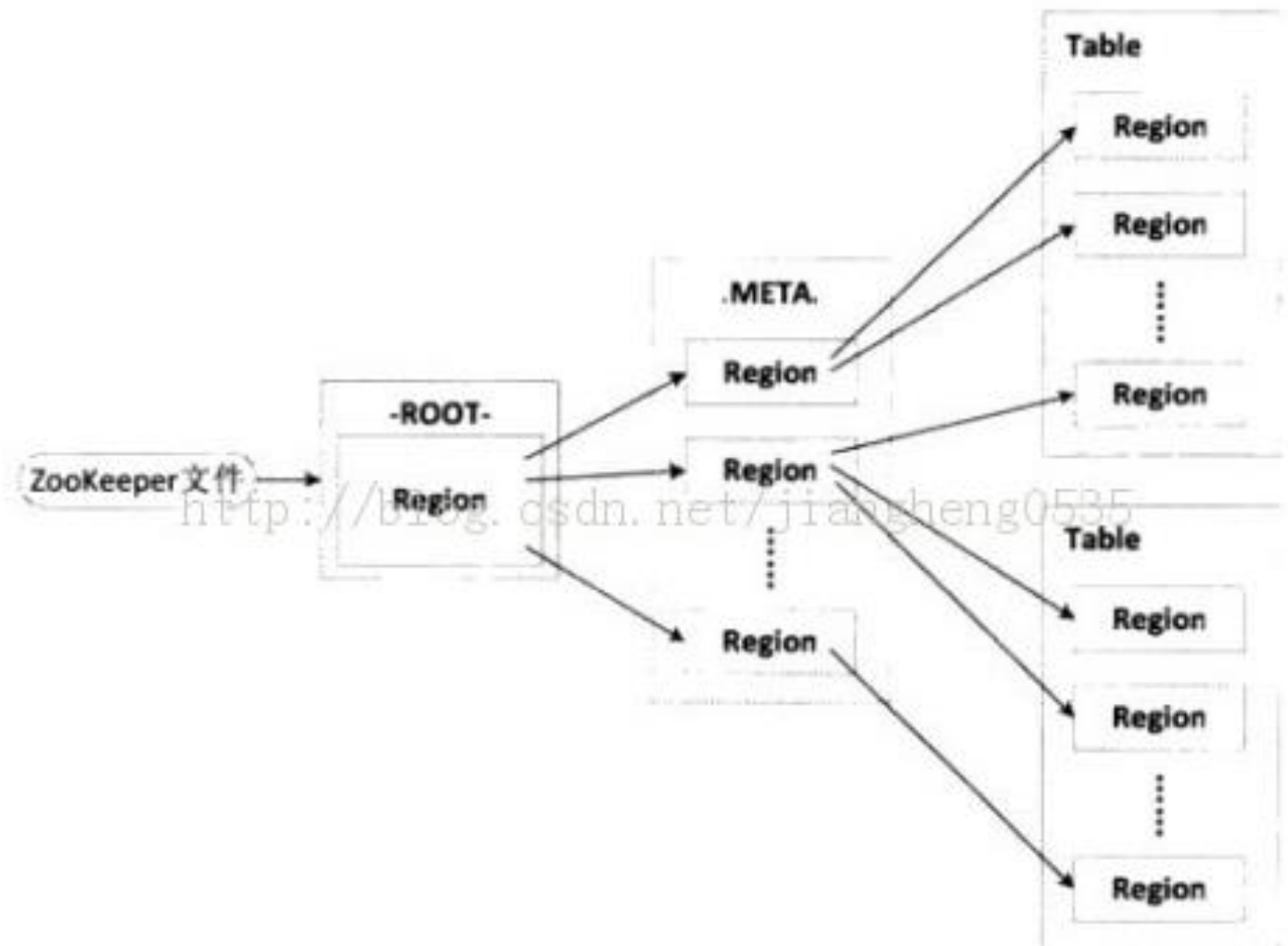
➤ **RegionServer容错：定时向Zookeeper汇报心跳，如果一旦时间内未出现心跳**

Master将该RegionServer上的Region重新分配到其他RegionServer上；

失效服务器上“预写”日志由主服务器进行分割并派送给新的RegionServer

➤ **Zookeeper容错：Zookeeper是一个可靠地服务**

一般配置3或5个Zookeeper实例



➤ 寻找RegionServer

✓ ZooKeeper

✓ -ROOT-(单Region)

✓ .META.

✓ 用户表



-ROOT-和. META. 表结构

RowKey	info			historian
	regioninfo	server	serverstartcode	
TableName, StartKey, Timestamp	StartKey, EndKey, Family List { Family, BloomFilter, Compress, TTL, InMemory, BlockSize, BlockCache }	address		

➤ -ROOT-

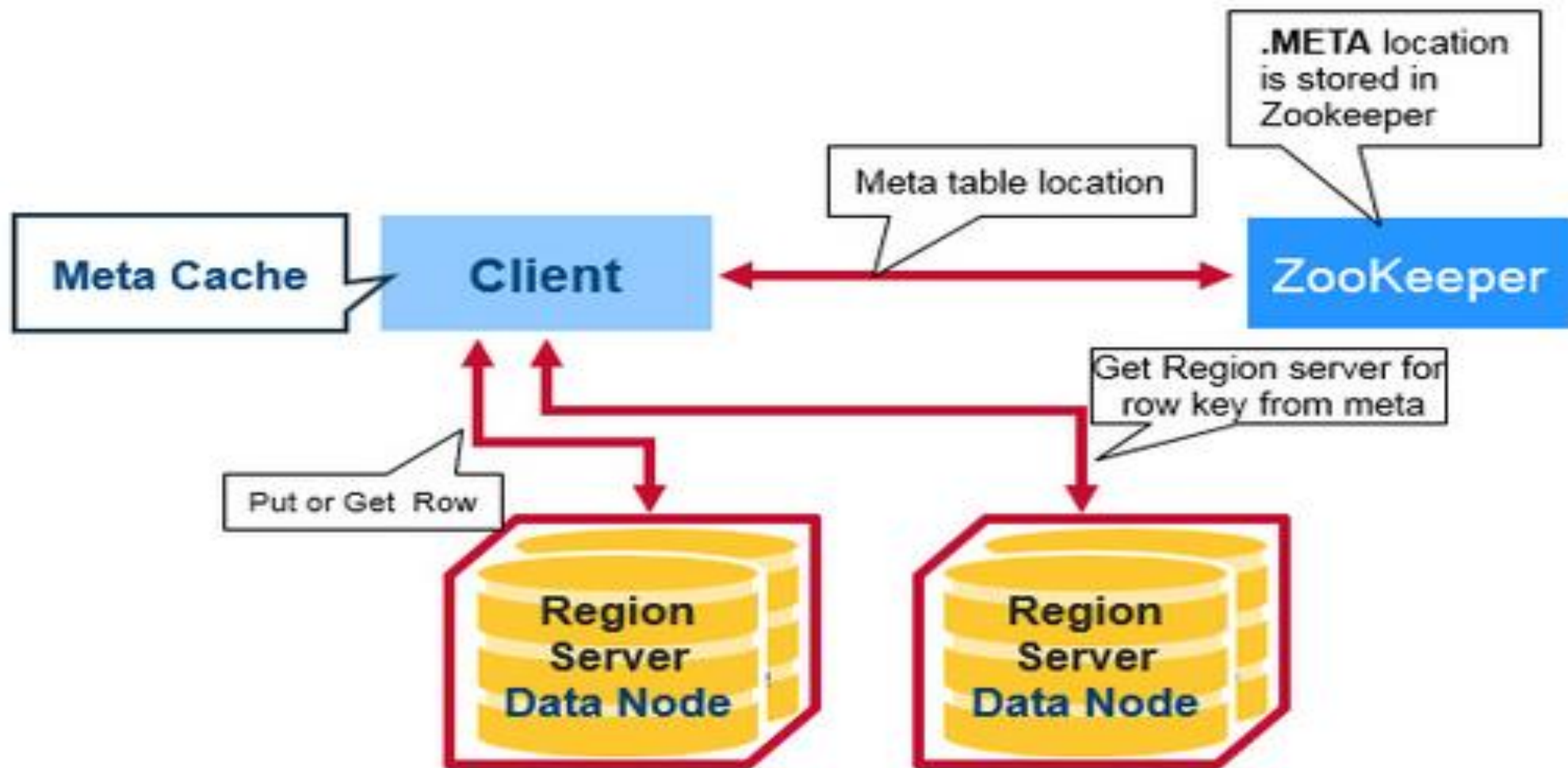
- ✓ 表包含.META.表所在的region列表，该表只会有一个Region；
- ✓ Zookeeper中记录了-ROOT-表的location。

➤ .META.

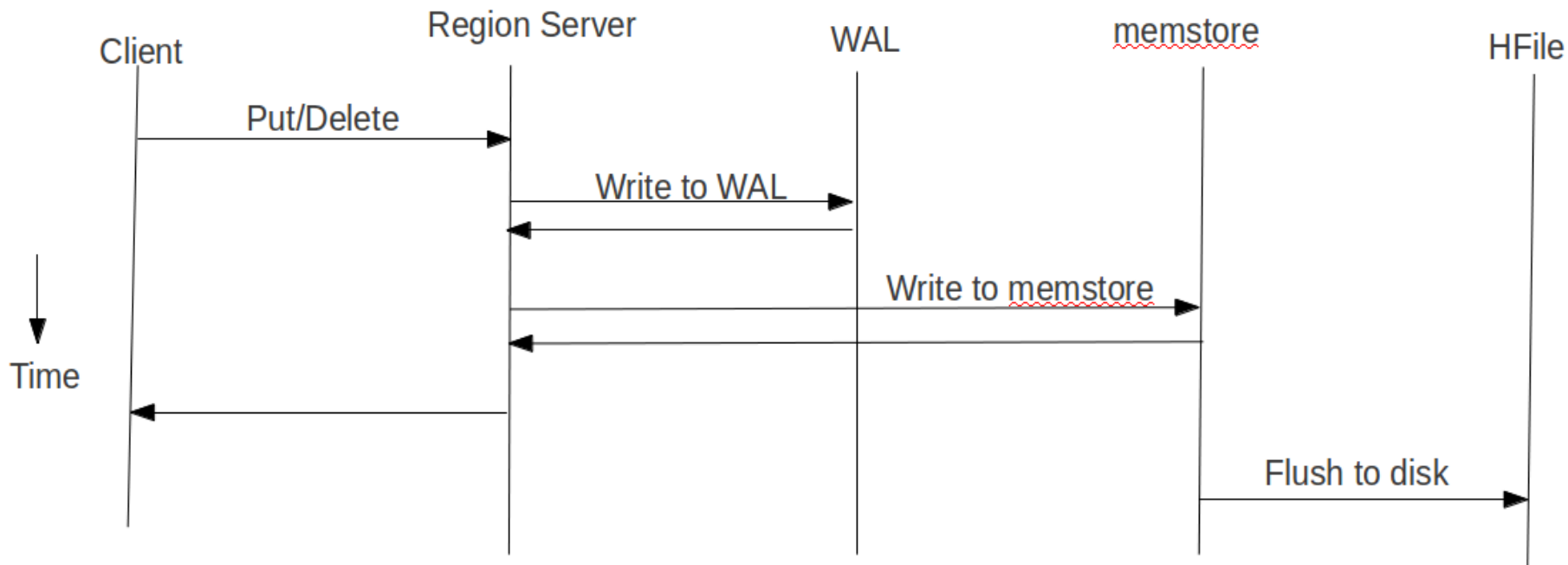
- ✓ 表包含所有的用户空间region列表，以及RegionServer的服务器地址。

➤ Region定位优化改进

- ✓ 在hbase 0.96 之前的版本中，hbase catalog包括-ROOT-和.META.两个表，但是在0.96以及之后的hbase版本中，-ROOT-表被废弃，只有hbase:meta（即.META.）。HBase的meta信息存储在表hbase:meta中，该表也是一个hbase表，但是在hbase shell下执行list命令时，会将该表过滤掉，不会显示。
- ✓ 在hbase:meta中，存储着所有regions的信息，且该表的位置直接记录在zookeeper中，而无需-ROOT-表。



讲 HBase put写流程



客户端：

1. 客户端发起Put写请求，将put写入writeBuffer，如果是批量提交，写满缓存后自动提交
2. 根据rowkey将put分发给不同regionserver

服务端：

1. Regionserver将put按rowkey分给不同的region
2. Region首先把数据写入wal
3. wal写入成功后，把数据写入memstore，此时写成功，并返回通知客户端
4. Memstore写完后，检查memstore大小是否达到flush阈值
5. 如果达到flush阈值，将memstore写入到HDFS，生成HFile文件

Hbase VS 关系型数据库

	关系型数据库	HBase
数据存储	面向行	面向列
事务	支持多行事务	仅支持单行事务
查询语言	Sql	get、put、scan
安全	比较完善	欠缺
索引	对任意列建立索引	只支持row key
数据大小	TB	PB级别以上数据
读写吞吐率	适合查询少量数据	适合批量数据查询

	Hive	HBase
联系	hbase与hive都是架构在hadoop之上的。都是用hadoop作为底层存储	
区别	<p>1、Hive中的表是纯逻辑表，只是表的定义等，即表的元数据。Hive本身不存储数据，它完全依赖HDFS和MapReduce。这样就可以将结构化的数据文件映射为为一张数据库表，并提供完整的SQL查询功能，并将SQL语句最终转换为MapReduce任务。</p> <p>2、Hive是基于MapReduce来处理数据,而MapReduce处理数据是基于行的模式</p> <p>3、Hive表是稠密型，即定义多少列，每一行有存储固定列数的数据。</p> <p>4、Hive使用Hadoop来分析处理数据，而Hadoop系统是批处理系统，因此不能保证处理的低延迟问题</p> <p>5、Hive不提供row-level的更新，它适用于大量append-only数据集（如日志）的批任务处理。</p> <p>6、Hive提供完整的SQL实现，通常被用来做一些基于历史数据的挖掘、分析。</p>	<p>1、HBase表是物理表，适合存放非结构化的数据。</p> <p>2、HBase处理数据是基于列的而不是基于行的模式，适合海量数据的随机访问。</p> <p>3、HBase的表是疏松的存储的，因此用户可以给行定义各种不同的列</p> <p>4、HBase是近实时系统，支持实时查询。</p> <p>5、HBase的查询，支持和row-level的更新。</p> <p>6、HBase不适用与有join，多级索引，表关系复杂的应用场景。</p>



THANKS