云计算技术与工程

大数据存储技术（二）

徐敏贤

副研究员
云计算研究中心
中国科学院深圳先进技术研究院
http://www.minxianxu.info/vcc

参横斗转欲三更，苦雨终风也解晴。云散月明谁点缀？天容海色本澄清。
——（宋）苏轼

内容回顾

- Q1: Google处理大数据的架构由哪些系统构成?

- **Google：架构提出者**
  - Google文件系统GFS(Google File System)
  - 并行数据处理 MapReduce
  - 结构化数据表 BigTable
  - 分布式锁管理 Chubby

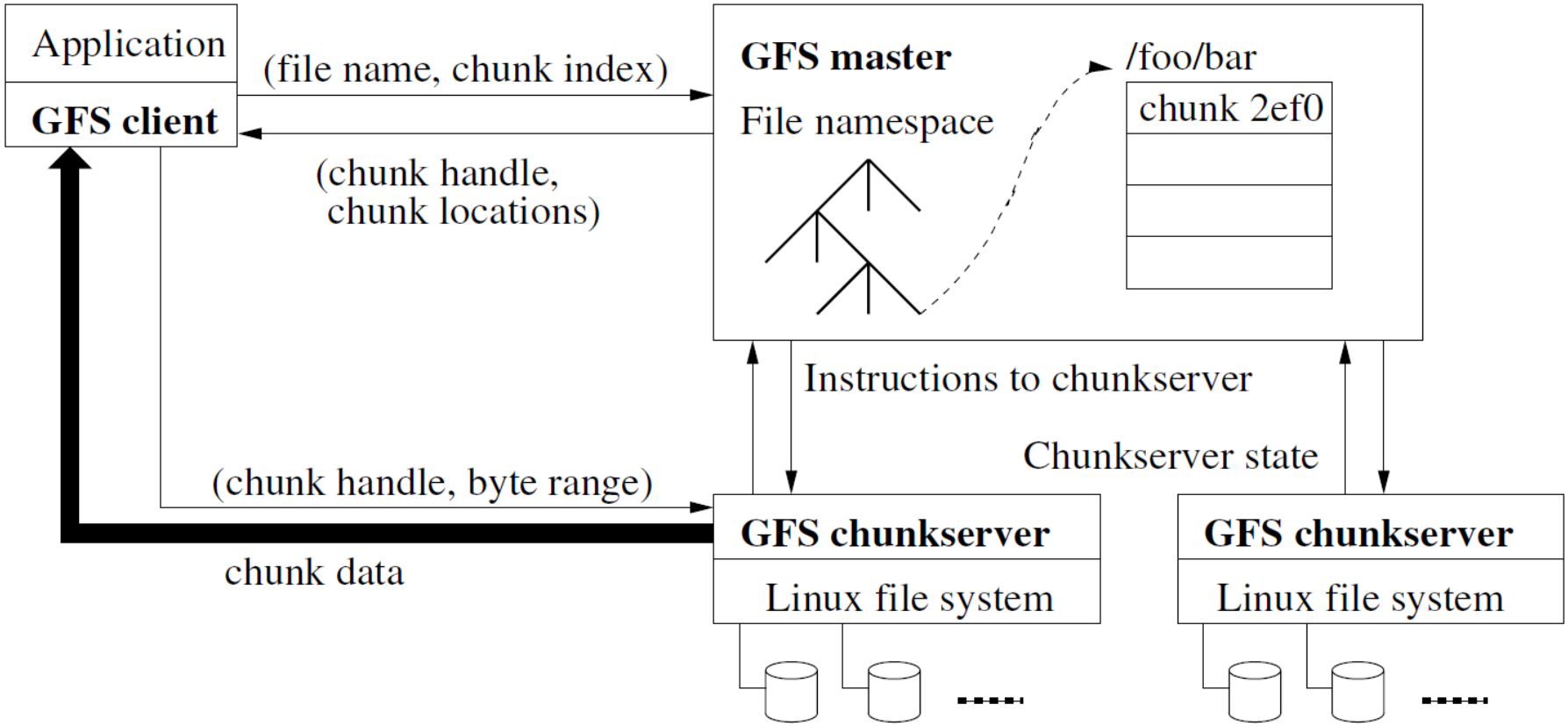
内容回顾

- Q2: Google File System (GFS)由哪些关键的实现技巧?

– 将文件划分为若干块（Chunk）存储
• 每个块固定大小（64M）
– 通过冗余来提高可靠性
• 每个数据块至少在3个数据块服务器上冗余
– 通过单个master来协调数据访问、元数据存储
• 结构简单，容易保持元数据一致性
– 无缓存

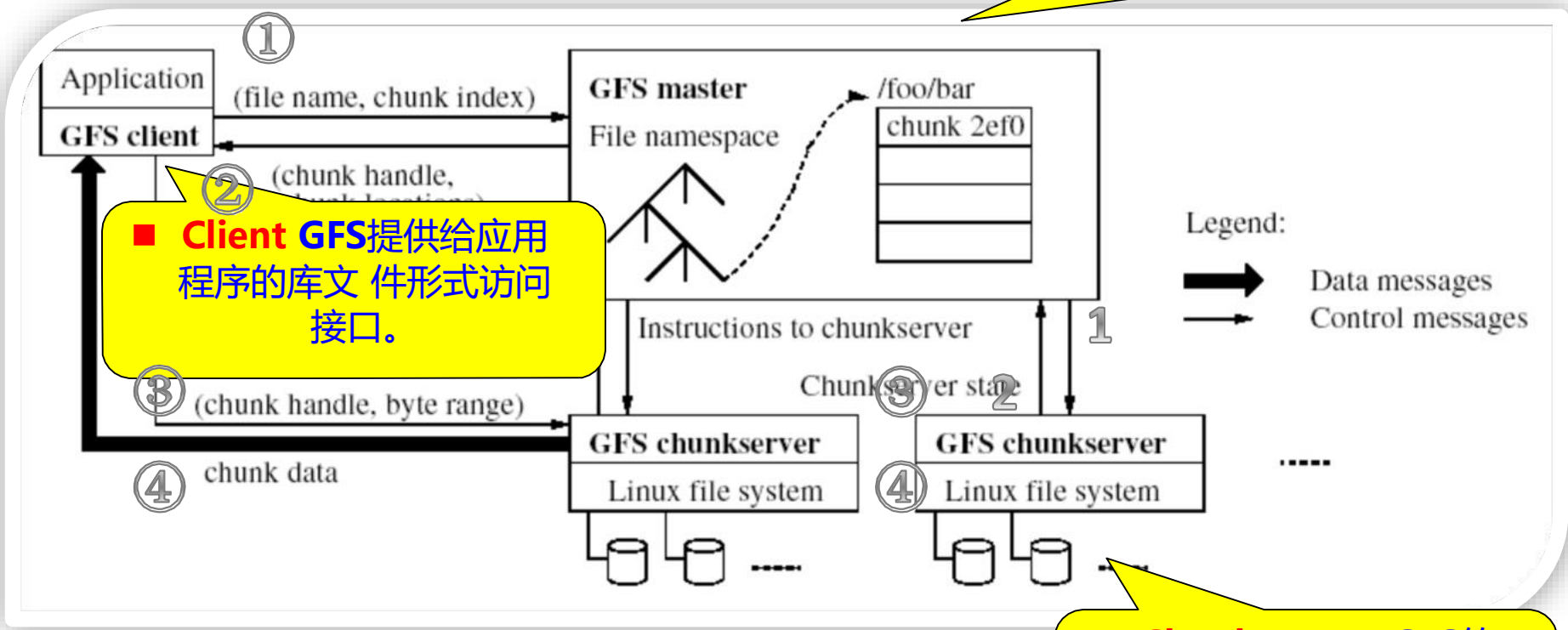
内容回顾

- Q3: 根据下图介绍一下GFS的结构和工作流程



- GFS架构

■ **Master GFS**的管理节点，逻辑上只 有一个，保存系统元数据。



■ **Client GFS**提供给应用程序的库文 件形式访问接口。

■ **Chunk server GFS**的数据存储节点，大小 固定（**64MB**）。

内容回顾

- Q4: GFS Master节点负责哪些主要的任务?

– **存储元数据**

– **文件系统目录管理与加锁**

– **与ChunkServer进行周期性通信**

• 发送指令，搜集状态，跟踪数据块的完好性

– **数据块创建、复制及负载均衡**

• 对ChunkServer的空间使用和访问速度进行负载均衡， 平滑数据存储和访问请求的负载

• 对数据块进行复制、分散到ChunkServer上

• 一旦数据块冗余数小于最低数，就发起复制操作

– **垃圾回收**

• 在日志中记录删除操作，并将文件改名隐藏

• 缓慢地回收隐藏文件

• 与传统文件删除相比更简单、更安全

– **陈旧数据块删除**

• 探测陈旧的数据块，并删除

内容回顾

- Q5: HDFS和GFS对比，有哪些区别?

	GFS	HDFS
实现架构	Master/slave（单一主控机+多台工作机）	Master/slave（单一主控机+多台工作机）
性质	分布式文件系统	分布式文件系统
高性能要求	通过数据分块和复制功能实现	通过数据分块和复制实现
追加操作处理	允许多客户端并发追加同一文件，存在重复追加和乱序问题	只允许一次打开并追加数据，利用临时文件操作解决乱序和重复记录的问题。
数据节点	有主副之分，存在Lease授权，减少追加时master的工作量	无主副之分，不存在Lease授权
Master单点失效处理	对master的数据有备份，master失效时，选举备机接替主master继续对外服务	Master的数据只写入到本机，master失效，需要人工介入
对快照的支持	通过内部的copy-on-write数据结构实现集群快照功能	不提供快照功能

目录 Contents

注：本章部分内容由王洋教授提供

Ceph

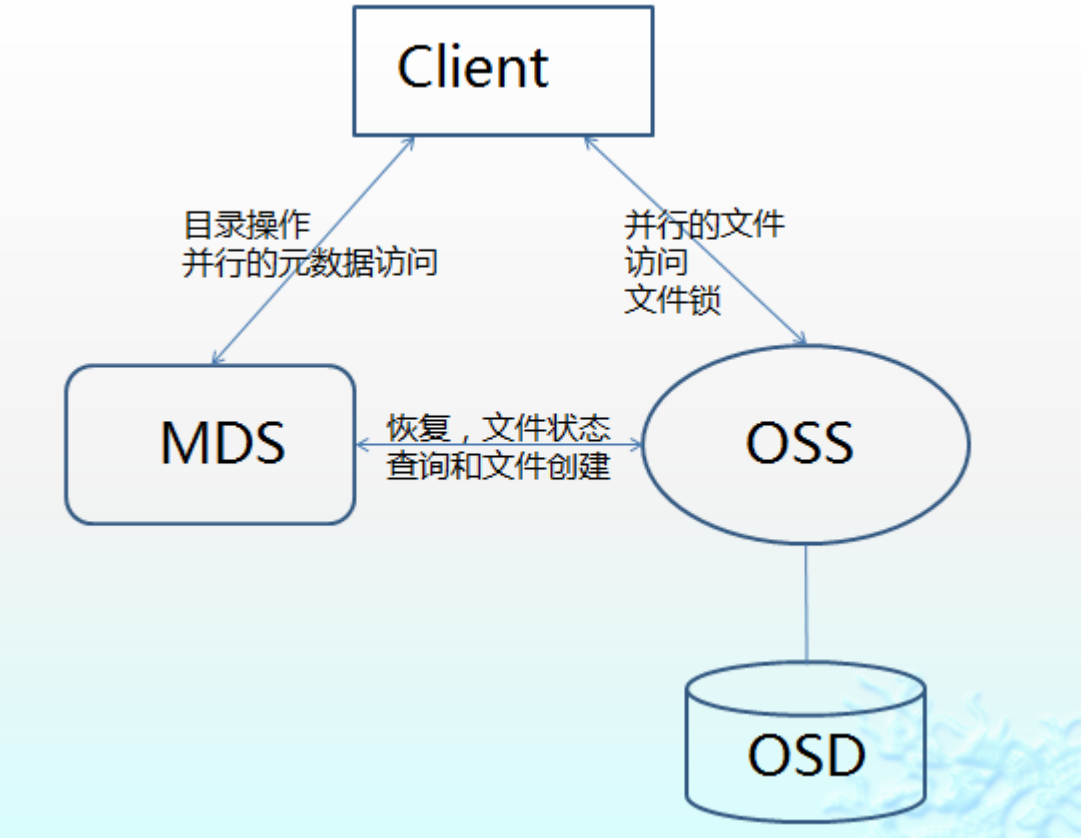
- **Overview**
  - Ceph is a free software distributed file system.
  - Ceph's main goals are to be POSIX-compatible, and completely distributed without a single point of failure.
  - The data is seamlessly replicated, making it fault tolerant.
- **Release**
  - On July 3, 2012, the Ceph development team released Argonaut, the first release of Ceph with long-term support.

Ceph

- Introduction
  - Ceph is a distributed file system that provides excellent performance, reliability and scalability.
  - Objected-based Storage.
  - Ceph separates data and metadata operations by eliminating file allocation tables and replacing them with generating functions.
  - Ceph utilizes a highly adaptive distributed metadata cluster, improving scalability.
  - Using object-based storage device (OSD) to directly access data, high performance.

Ceph

- Objected-Based Storage

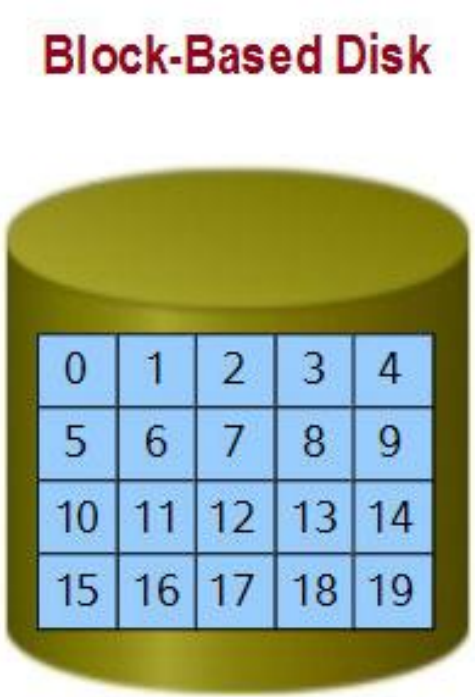


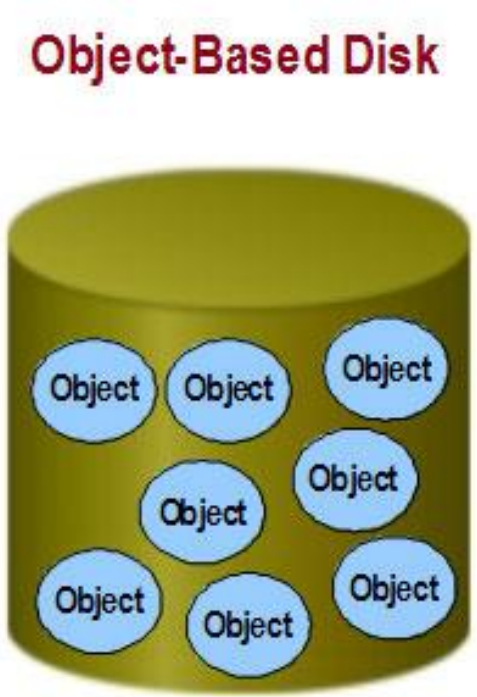
对象存储架构
核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备（Object-based Storage Device，OSD）构建存储系统，**每个对象存储设备具有一定的智能**，能够自动管理其上的数据分布。对象存储结构由对象、对象存储设备、元数据服务器、对象存储系统的客户端四部分组成。

Ceph

- **对象**

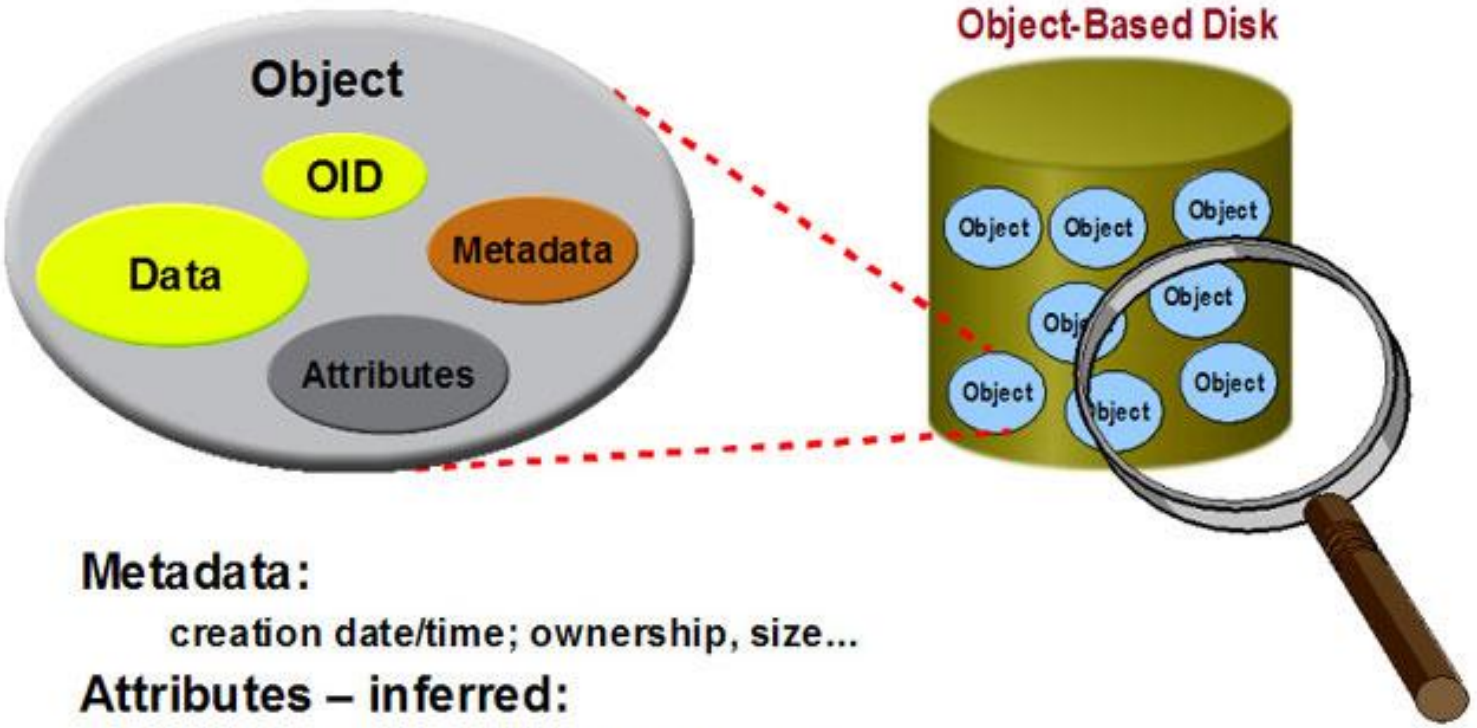
  对象是系统中数据存储的基本单位，每个Object是数据和数据属性集的综合体，数据属性可以根据应用的需求进行设置，包括数据分布、服务质量等。





Ceph

- Objected-Based Storage



Metadata:

creation date/time; ownership, size...

Attributes – inferred:

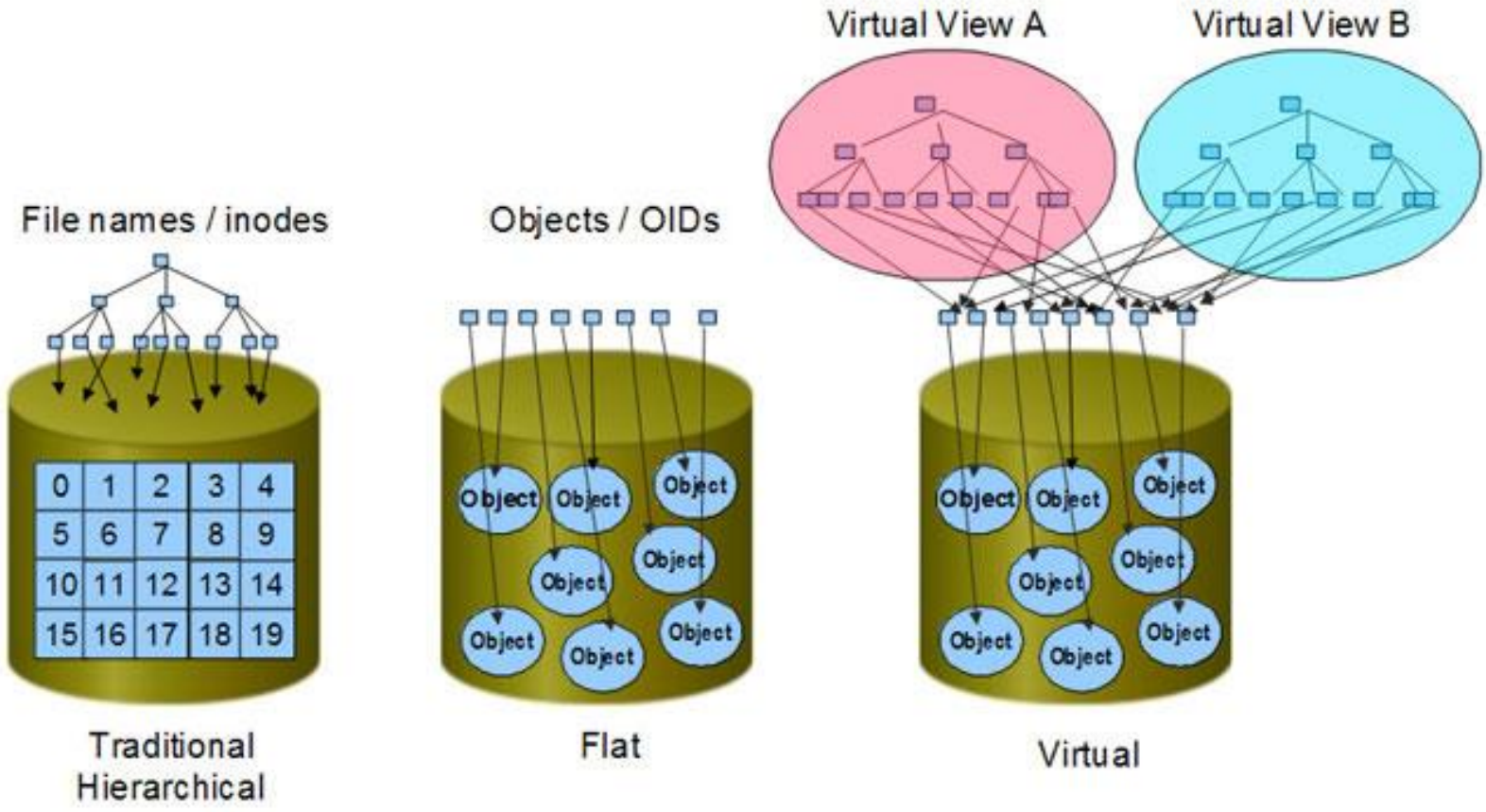
access patterns, content, indexes...

Attributes – user supplied:

retention, QoS...

Ceph

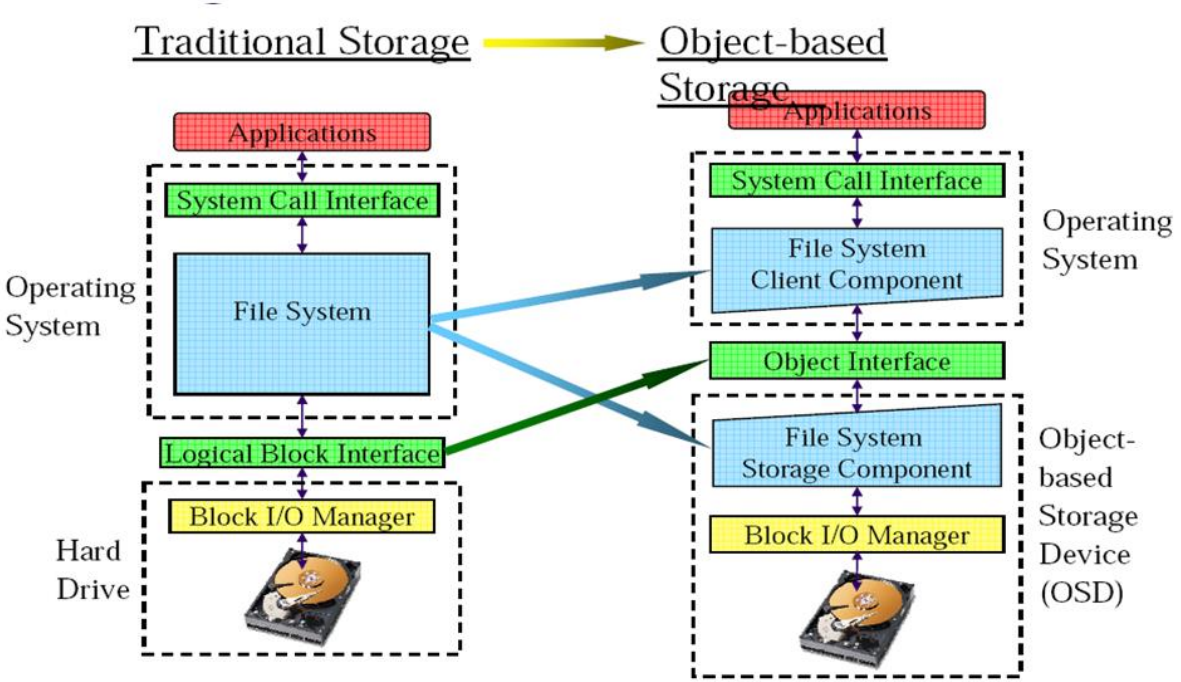
- 对象


Virtual View A
Virtual View B
File names / inodes
Objects / OIDs
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
Object
Traditional
Hierarchical
Flat
Virtual

Ceph

对象存储设备

每个OSD都是一个智能设备，具有自己的存储介质、处理器、内存以及网络系统等，负责管理本地的Object，是对象存储系统的核心



OSD提供三个主要功能：

数据存储：OSD不提供块接口访问方式，Client请求数据时用对象ID、偏移进行数据读写。

智能分布：OSD用其自身的CPU和内存优化数据分布，并支持数据的预取。

每个对象元数据的管理：OSD管理存储在其上**对象的元数据**，该元数据与传统的inode元数据相似，通常包括对象的数据块和对象的长度。对象存储架构将系统中主要的元数据管理工作由OSD来完成，降低了Client的开销。

元数据服务器

MDS控制Client与OSD对象的交互，为客户端提供元数据，主要是文件的逻辑视图，包括文件与目录的组织关系、每个文件所对应的OSD等。主要提供以下几个功能：

•**对象存储访问。**MDS构造、管理描述每个文件分布的视图，允许Client直接访问对象。MDS为Client提供访问该文件所含对象的能力，OSD在接收到每个请求时将先验证该能力，然后才可以访问。

•**文件和目录访问管理。**MDS在存储系统上构建一个文件结构，包括限额控制、目录和文件的创建和删除、访问控制等。

•**Client Cache一致性。**为了提高Client性能，在对象存储系统设计时通常支持Client方的Cache。由于引入Client方的Cache，带来了Cache一致性问题，MDS支持基于Client的文件Cache，当Cache的文件发生改变时，将通知Client刷新Cache，从而防止Cache不一致引发的问题。

对象存储系统的客户端

对象存储系统提供给用户是标准的POSIX文件访问接口。接口具有和通用文件系统相同的访问方式，同时为了提高性能，也具有对数据的Cache功能和文件的条带功能。同时，文件系统必须维护不同客户端上Cache的一致性，保证文件系统的数据一致。文件系统读访问流程：

① 客户端应用发出读请求;
② 文件系统向元数据服务器发送请求，获取要读取的数据所在的OSD;
③ 然后直接向每个OSD发送数据读取请求;
④ OSD得到请求以后，判断要读取的Object，并根据此Object要求的认证方式， 对客户端进行认证，如果此客户端得到授权，则将Object的数据返回给客户端;
⑤ 文件系统收到OSD返回的数据以后，读操作完成。

- # 对象存储文件系统的关键技术

1. 元数据服务器通常提供两个主要功能。
    - 为计算结点提供一个存储数据的逻辑视图（Virtual File System，VFS层），文件名列表及目录结构。
    - 组织物理存储介质的数据分布（inode层）。对象存储结构将存储数据的逻辑视图与物理视图分开，并将负载分布，避免元数据服务器引起的瓶颈（如NAS系统）。

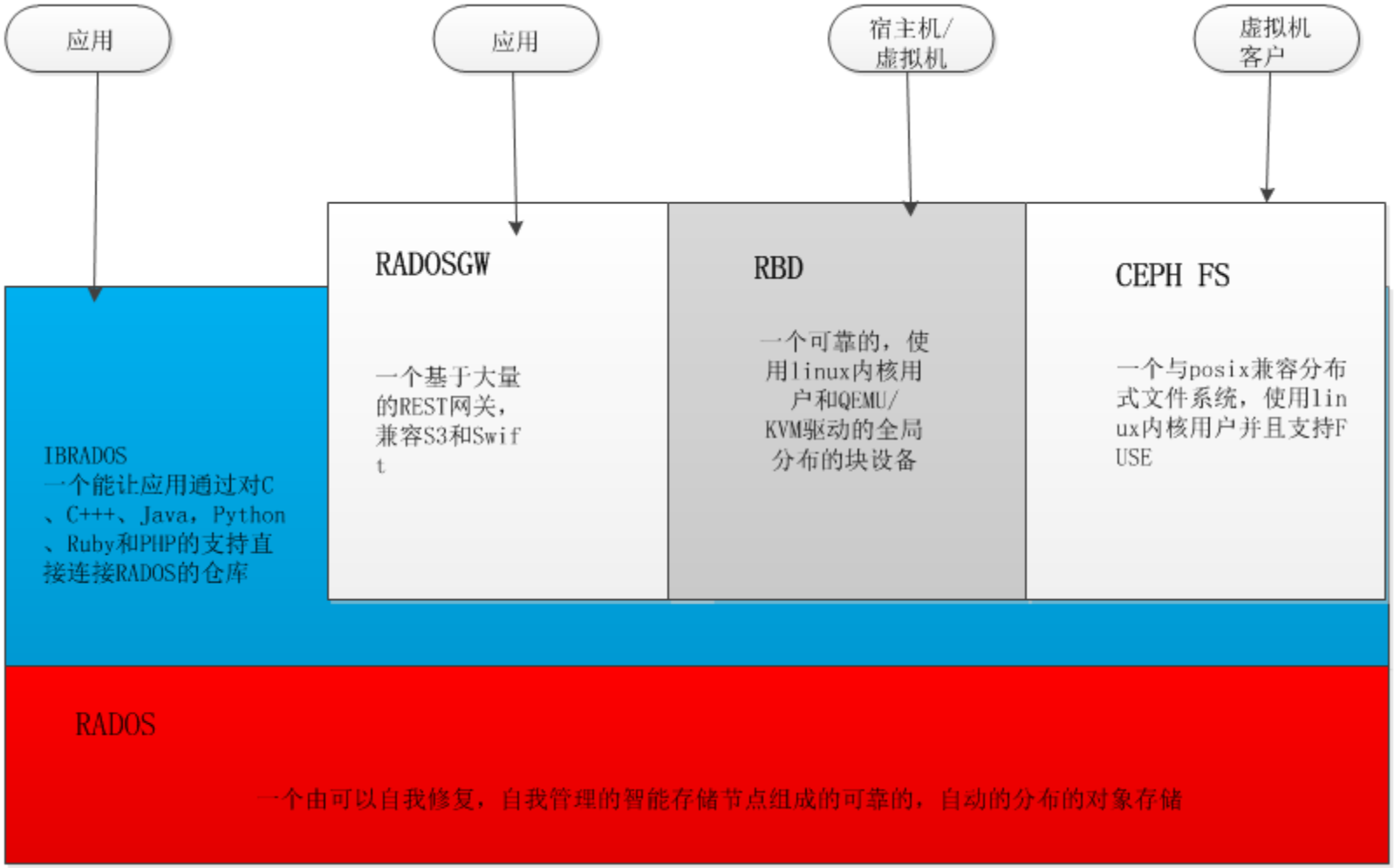
在对象存储结构，inode工作分布到每个智能化的OSD，每个OSD负责管理数据分布和检索，这样90%的元数据管理工作分布到智能的存储设备，从而提高了系统元数据管理的性能。另外，分布的元数据管理，在增加更多的OSD到系统中时，可以同时增加元数据的性能和系统存储容量。

2. 并发数据访问 对象存储体系结构定义了一个新的、更加智能化的磁盘接口OSD。 如果将文件系统的数据分布在多个OSD上，则聚合I/O速率和数据吞吐率将线性增长。对象存储结构提供的性能是目前其它存储结构难以达到的，如ActiveScale对象存储文件系统的带宽可以达到10GB/s。

Ceph

- 目标
  - 可扩展性
    - Storage capacity, throughput, client performance. Emphasis on HPC.
  - 可靠性
    - Failures are the norm rather than the exception, so the system must have fault detection and recovery mechanism.
  - 性能
    - Dynamic workloads
    - Load balance.

Ceph


应用
应用
宿主机/
虚拟机
虚拟机
客户
RADOSGW
一个基于大量的REST网关，兼容S3和Swift
RBD
一个可靠的，使用linux内核用户和QEMU/KVM驱动的全局分布的块设备
CEPH FS
一个与posix兼容分布式文件系统，使用linux内核用户并且支持FUSE
IBRADOS
一个能让应用通过对C、C++、Java，Python、Ruby和PHP的支持直接连接RADOS的仓库
RADOS
一个由可以自我修复，自我管理的智能存储节点组成的可靠的，自动的分布的对象存储

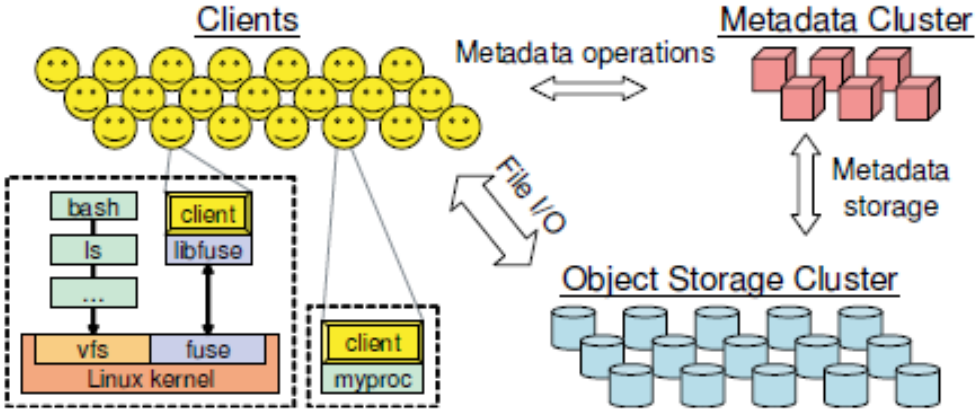
Ceph

RADOS

Ceph中RADOS(Reliable Autonomic Distributed Object Store)存储集群是所有其他客户端接口使用和部署的基础。RADOS由两个组件组成：

① OSD： Object StorageDevice，提供存储资源。

② Monitor：维护整个Ceph集群的全局状态。



OSD集群负责存储所有对象的数据。Monitors集群负责管理Ceph集群中所有成员、关系、属性以及数据分发等信息。

Ceph

- **Ceph客户端接口(Clients)**

  我们将Ceph架构中除了底层基础RADOS之上的LIBRADOS、RADOSGW、RBD以及Ceph FS统一称为Ceph客户端接口。而LIBRADOS又是Ceph其它如RADOSGW、RBD以及Ceph FS的基础。

RADOSGW

RADOSGW(RADOS Gmeway)，又叫Ceph对象存储网关，是一个底层基于librados向客户端提供RESTful接口的对象存储接口。目前Ceph支持两种API接口：

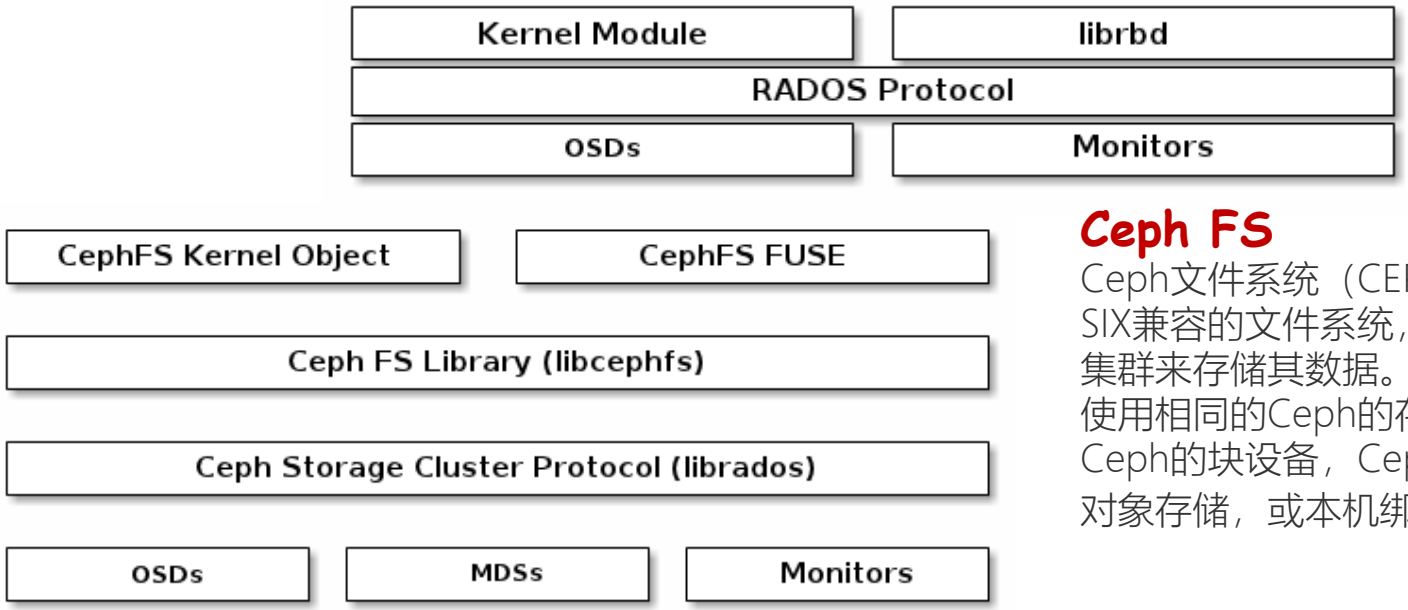
(1) S3.compatible：S3兼容的接口，提供与Amazon S3大部分RESTfuI API接口兼容的API接口。

(2) Swift.compatible：提供与OpenStack Swift大部分接口兼容的API接口。Ceph的对象存储使用网关守护进程（radosgw），radosgw结构图如图所示：

Ceph客户端接口(Clients)

RBD

在一个Ceph的集群中， Ceph的块设备支持自动精简配置，调整大小和存储数据。Ceph的块设备可以充分利用 RADOS功能，实现如快照，复制和数据一致性。Ceph的RADOS块设备（即RBD）通过RADOS协议与内核模块或librbd的库进行交互。RBD的结构如图所示：



Ceph FS

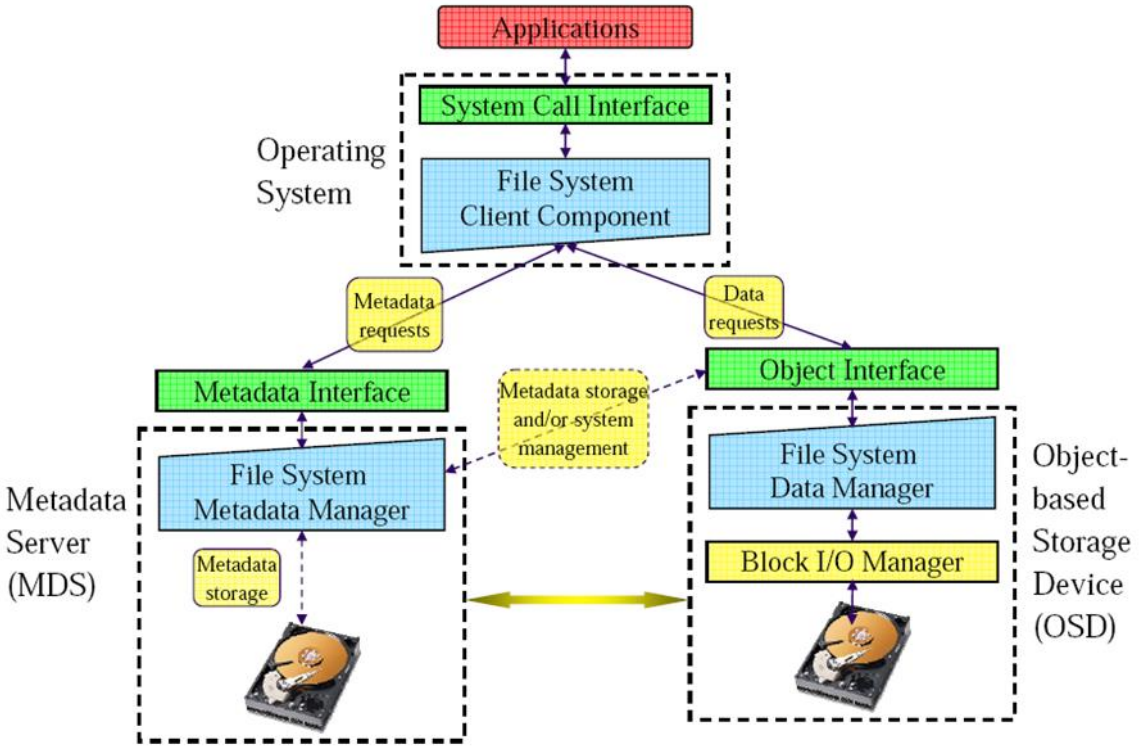
Ceph文件系统（CEPH FS）是一个POSIX兼容的文件系统，使用Ceph的存储集群来存储其数据。Ceph的文件系统使用相同的Ceph的存储集群系统比如Ceph的块设备，Ceph的S3和SwiftAPI对象存储，或本机绑定（librados）。

数据与元数据的分离

- Separation of file metadata management from the storage.
- Metadata operations are collectively managed by a metadata server cluster.
- User can direct access OSDs to get data by metadata.
- Ceph removes data allocation lists entirely.
- CRUSH assigns objects to storage devices.

数据与元数据的分离

- Ceph separates data and metadata operations



数据与元数据的分离

Data Distribution with CRUSH

- In order to avoid imbalance (OSD idle, empty) or load asymmetries (hot data on new device).
  →distributing new data randomly.
- Use a simple hash function, Ceph maps objects to Placement groups (PGs). PGs are assigned to OSDs by CRUSH.

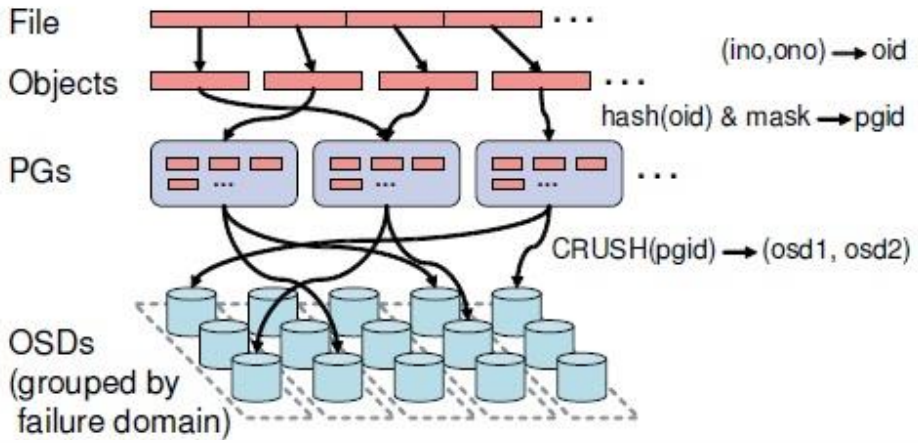
对象的分发需要经过两个阶段的计算，才能得到存储该对象的OSD，然后将对象存储到OSD中对应的位置。

(1) 对象到PG的映射：相同PG里的对象将被分发到相同的OSD节点中。

PG=hash（ObjID）

(2) PG到相应的OSD的映射

CRUSH(x) (osdn1, osdn2, osdn3)
Inputs
  x is the placement group
  Hierarchical cluster map
  Placement rules
Outputs a list of OSDs

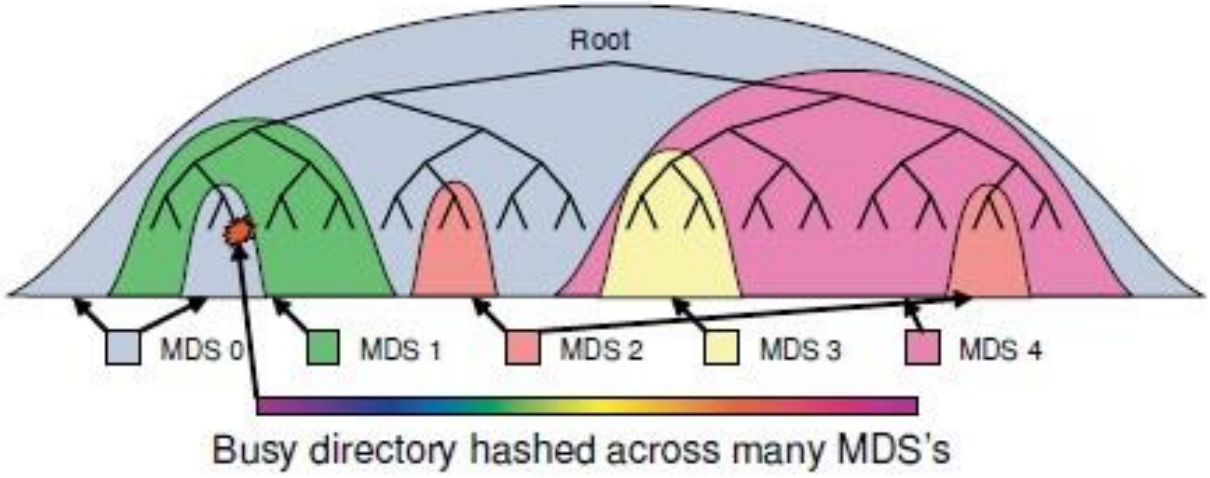


ID	Binary Data	Metadata
1234	01010101010101001101010010 01011000010101001101010010 01011000010101001101010010	name1 value1 name2 value2 nameN valueN

元数据的动态管理

元数据动态分布式管理

- Ceph utilizes a metadata cluster architecture based on Dynamic Subtree Partitioning (workload balance) .
- Dynamic Subtree Partitioning
- Most FS, use static subtree partitioning
  →imbalance workloads.
  →simple hash function can get directory.
- Ceph's MDS cluster is based on a dynamic subtree partitioning →balance workloads



Metadata

- 元数据动态分布式管理
  - MDSs use journaling
    - Repetitive metadata updates handled in memory.
    - Optimizes on-disk layout for read access.
  - Per-MDS has journal (usually a circular log in a dedicated area of the file system), when MDS failure another node can quickly recover with journal.
  - inodes are embedded directly within directories.
  - Each directory's content is written to the OSD cluster using the same striping and distribution strategy as metadata journals and file data.

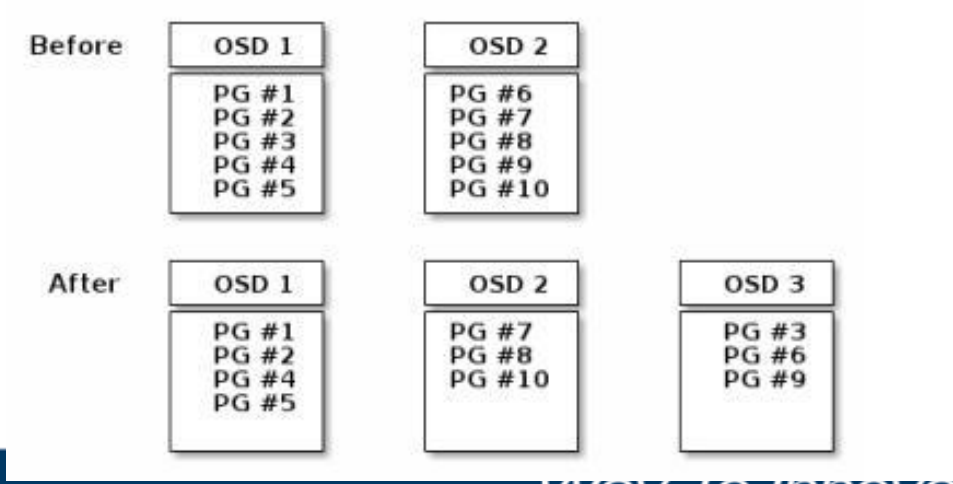
OSD智能负载均衡

当在Ceph集群中增加或减少OSD设备时，集群会执行负载再均衡的过程(rebalancing)。

•集群地图(Cluster Map)会得到更新，PG ID以及OSD集群相关的信息都会得到更新。一些PG从其原来所处的OSD存储设备迁移到了新的OSD存储设备。

•在数据再均衡过程中，CRUSH保持稳定，有许多的PG还是依然保留其原有的配置。并且由于进行了数据的迁出，原有OSD设备中的剩余容量也会相应的有所增加。

•整个数据再均衡过程也是利用的CRUSH算法，数据依然是均衡的分布在新的OSD集群中。



可靠的分布式对象存储

在Ceph中，每个Ceph的客户端和OSD节点都保存有整个系统相关的拓扑信息。这样，客户端就能直接和存储数据的OSD节点进行交互，OSD节点相互之间也能直接进行交互。Ceph中去中心节点的架构能够带来以下一些好处：

(1) OSD节点能直接为客户端提供服务：

Ceph中客户端能与存放数据的OSD节点直接通信，而不用经过任何的中心节点，这样整个系统不仅没有单点问题，而且性能就得到了很大的提升。

(2) OSD节点参与系统的维护：

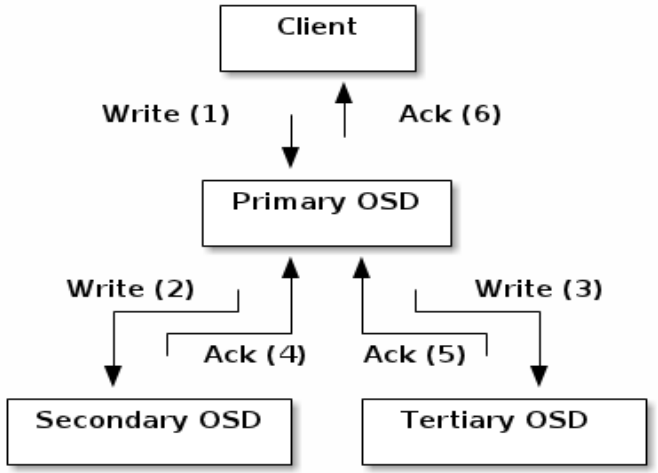
通常一个OSD节点加入到Ceph存储集群中，要向集群中的Monitor节点汇报自己的状态。如果OSD节点宕机，则需要系统能自动检测出来。

(3) OSD节点定期的数据清洁：

数据清洁是指，一个OSD节点中存储的对象与另外一个存储该对象副本的OSD节点之间进行对象的元数据对比，依此来找出文件系统相关的错误。

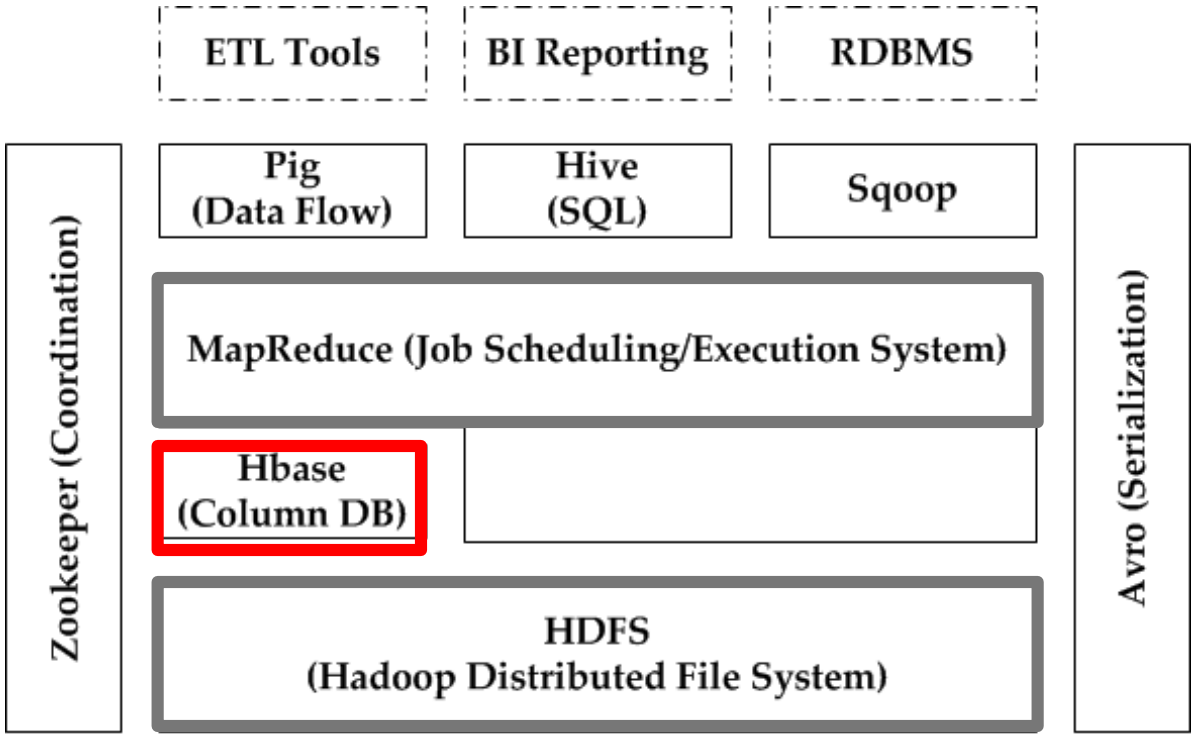
(4) 数据智能备份：

和Ceph客户端一样，Ceph OSD节点也使用CRUSH算法。但是和客户端使用CRUSH算法来查找数据不同，Ceph OSD节点使用该算法来计算对象的备份副本应该被存储在哪个位置。



技术内容

- HDFS — 块存储，分布式架构
- Hbase — 列存储，Key-Value数据库
- MapReduce — 思想及原理



The Hadoop Ecosystem

RDBMS的问题

RDBMS处理海量数据的问题：

- 可扩展性困难
  - Scale up: scale vertically → costly
  - Scale out: scale horizontally → difficult
- 低可用性
  - 索引维护开销巨大
  - “链接 Join”操作开销巨大

36

HBase - What?

HBase – Hadoop Database，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技术可在廉价PC Server上搭建起大规模结构化存储集群

- Modeled on Google's Bigtable
- Row/column store
- Billions of rows/millions on columns
- Column-oriented - nulls are free
- Untyped - stores byte[]

NoSQL

- 不是Anti-SQL和 ‘no’ SQL，而是 (N)ot (O)nly SQL. 准确的含义应该是： Non-Relational DB。
- 不同于传统的 RDBMS. 数据并不是存储在固定的表格内，一般避免Join操作，不试图提供 ACID (atomicity, consistency, isolation, durability) 等属性，支持扩展性.
- NoSQL的特点:
  - 灵活的机制 (降低了关联性)
  - 高可用性
  - 水平扩展

HBase —设计目标

- 为什么需要设计HBase?
  - 如何存储种类众多的数据，如网页，邮件……
  - 如何使用统一的方式存储各类数据?
  - 如何快速地从海量信息中寻找需要的数据?
- 数据存储可靠性
- 存储海量的记录（若干TB），便于压缩
- 可以保存记录的多个版本
- 与写操作相比，数据记录读操作占大多数工作负载
- 单个节点故障损坏是常见的，需要高可用性
- 高可扩展性

海量存储

Hbase适合存储PB级别的海量数据，在PB级别的数据以及采用廉价PC存储的情况下，能在几十到百毫秒内返回数据。

极易扩展

Hbase的扩展性主要体现在两个方面，一个是基于上层处理能力（Region Server）的扩展，一个是基于存储的扩展（HDFS）。

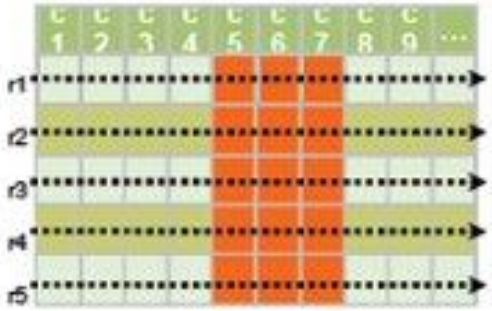
高并发

在并发的情况下，Hbase的单个IO延迟下降并不多。能获得高并发、低延迟的服务。

稀疏

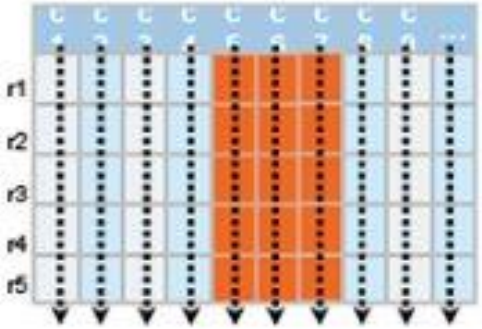
稀疏主要是针对Hbase列的灵活性，在列族中，你可以指定任意多的列，在列数据为空的情况下，是不会占用存储空间的。

传统行式数据库



- 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
- 面对查询的需求，数据库必须被大量膨胀才能满足性能要求

列式数据库

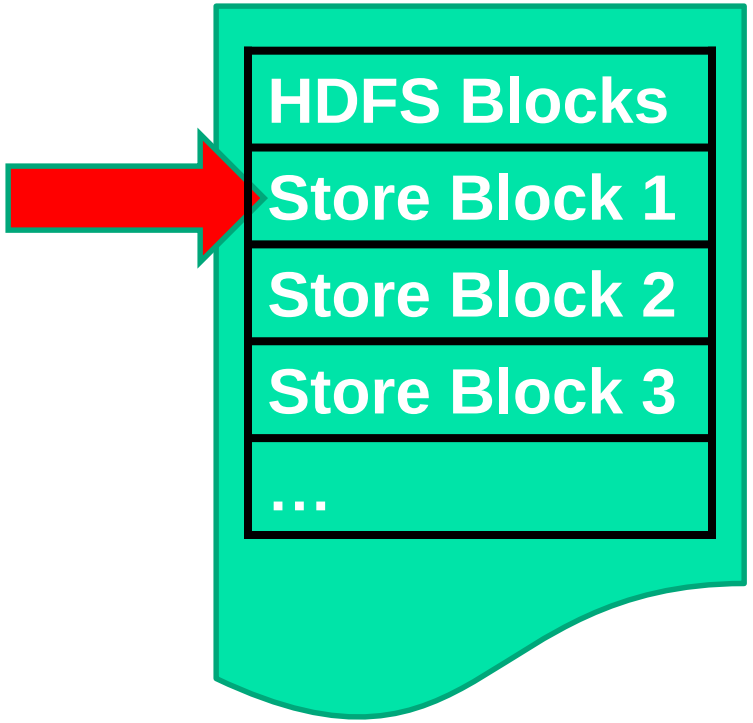


- 数据按列存储 - 每一列单独存放
- 数据即是索引
- 只访问查询涉及的列 - 大量降低系统IO
- 每一列由一个线索来处理 - 查询的并发处理
- 数据类型一致，数据特征相似 - 高效压缩

行存储

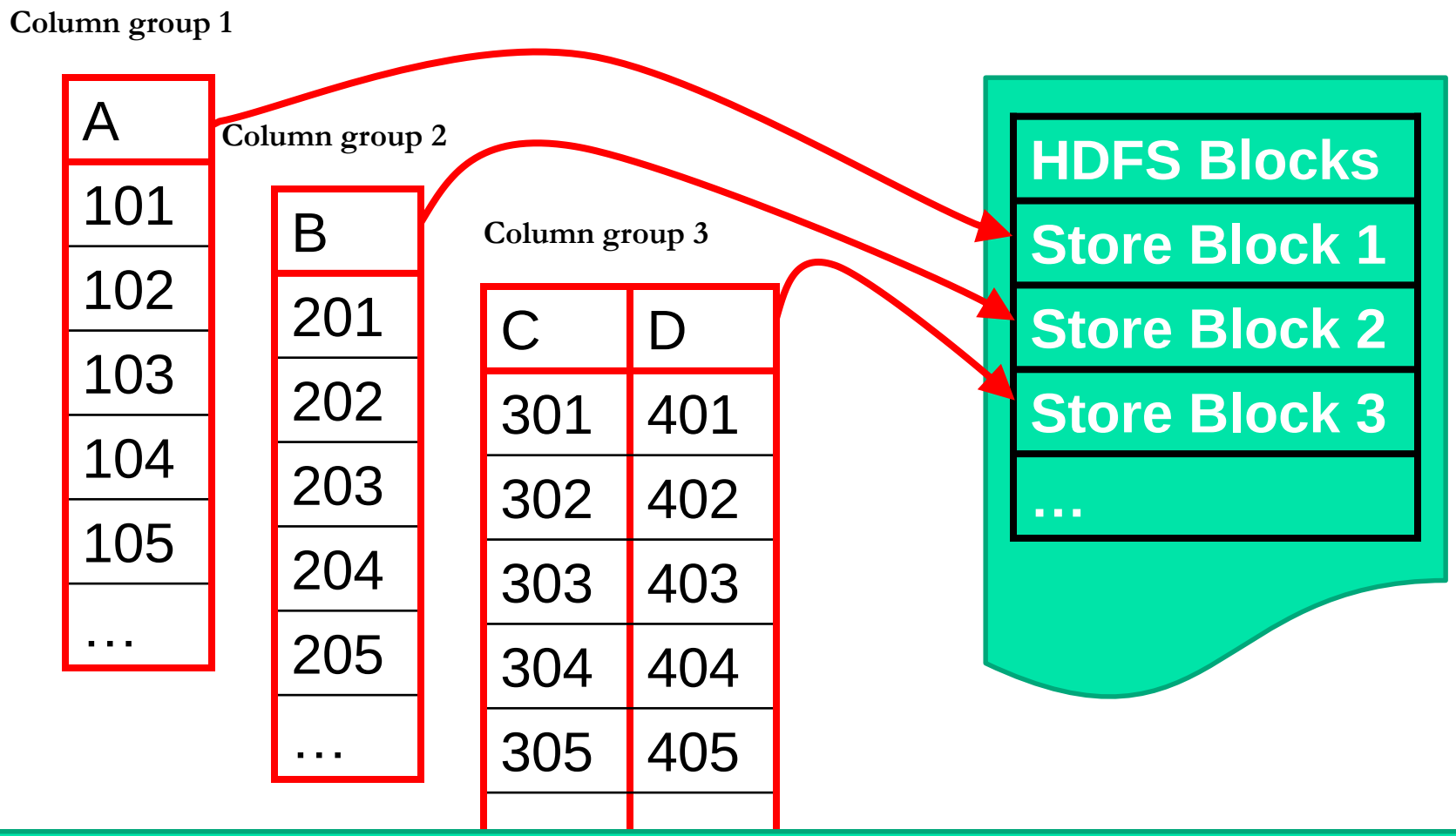
Table

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405



- 数据加载非常迅速
- 一行的所有列被放置在同一个HDFS block
- 并不是所有列被用到，浪费存储带宽
- 难以压缩不同类型的数据，可能增加新的开销

列存储



- 不必要的 I/O 开销被避免，只访问有需要的列，并且易于压缩
- 需要额外的开销来做列的重组

HBase数据模型

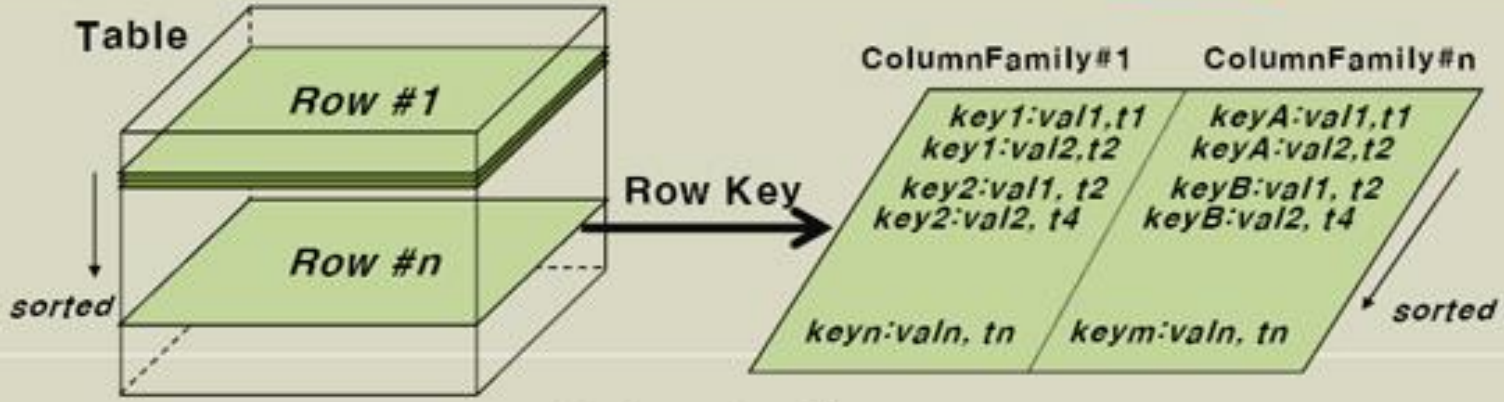
Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	…	…	…	…

Row Key: 行键，Table的主键，Table中的记录按照Row Key排序

Timestamp: 时间戳，每次数据操作对应的时间戳，可以看作是数据的version number

Column Family：列簇，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。

Data Model



Column family

rowkey	contents:	anchor:	language:	...
com.cnn.www	<html>aa<html> (t1) <html>bb<html> (t2)	anchor:cnnsi.com, CNN (t1) anchor:my.look.ca, CNN.com(t1) anchor:jaso.co.kr, jaso.gif(t1)	en	

ColumnFamily:ColumnKey, ColumnValue(Timestamp)

HBase结构

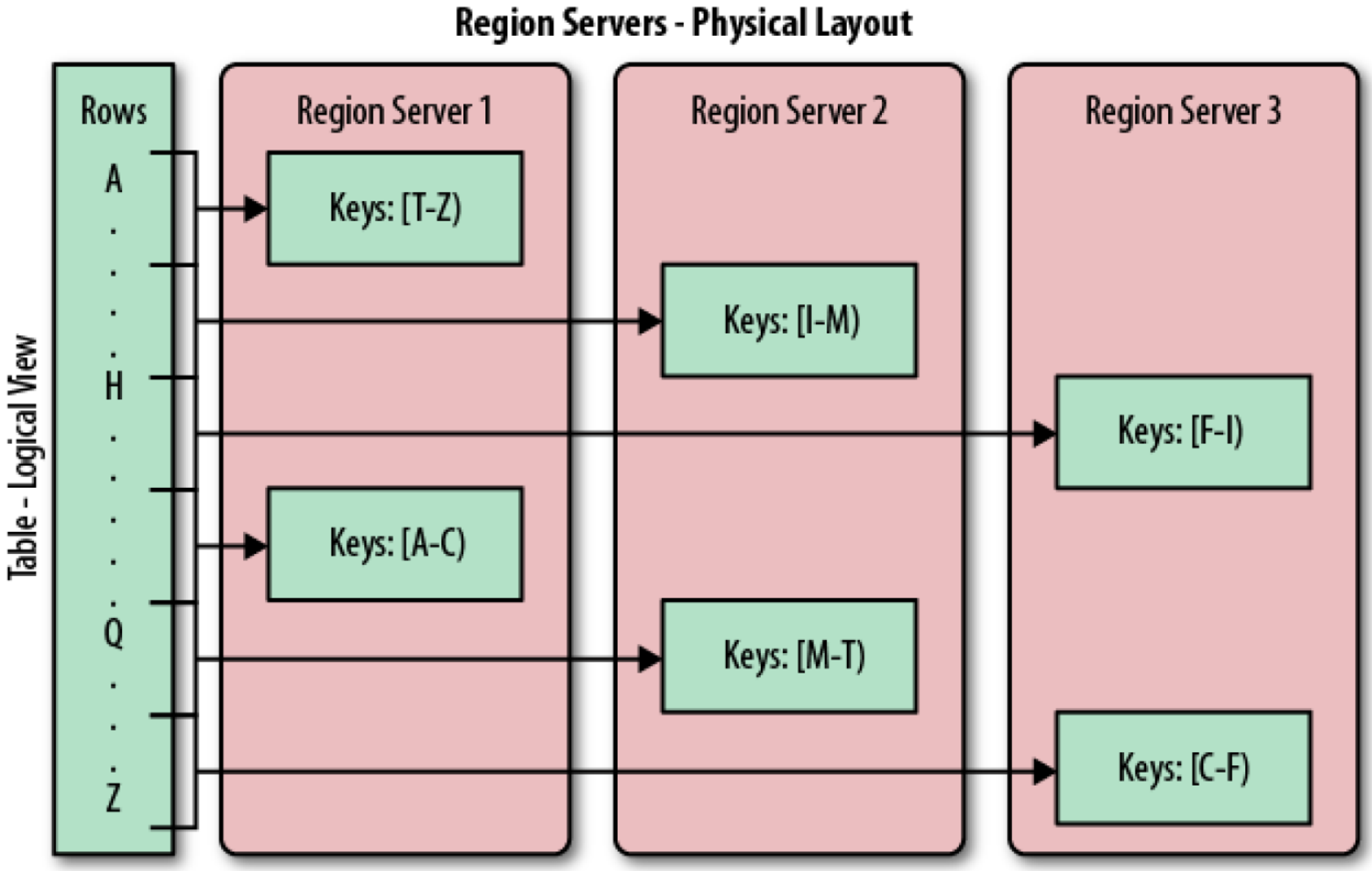

Region Servers - Physical Layout
Table - Logical View
Rows
A
H
Q
Z
Region Server 1
Keys: [T-Z)
Keys: [A-C)
Region Server 2
Keys: [I-M)
Keys: [M-T)
Region Server 3
Keys: [F-I)
Keys: [C-F)

Table & Region

当Table随着记录数不断增加而变大后，会逐渐分裂成多份splits，成为regions，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：

Rowkey	UserInfo Table
0001	name:Suse age:25
0002	name:Simon age:30
0003	name:Fasan age:38
0004	name:Carl age:50
0005	name:Suck age:40
0006	name:Cens age:36
0007	name:Luse age:18

Rowkey	UserInfo Table - Region1
0001	name:Suse age:25
0002	name:Simon age:30

Rowkey	UserInfo Table - Region2
0003	name:Fasan age:38
0004	name:Carl age:50

RS1

Rowkey	UserInfo Table - Region3
0005	name:Suck age:40
0006	name:Cens age:36
0007	name:Luse age:18

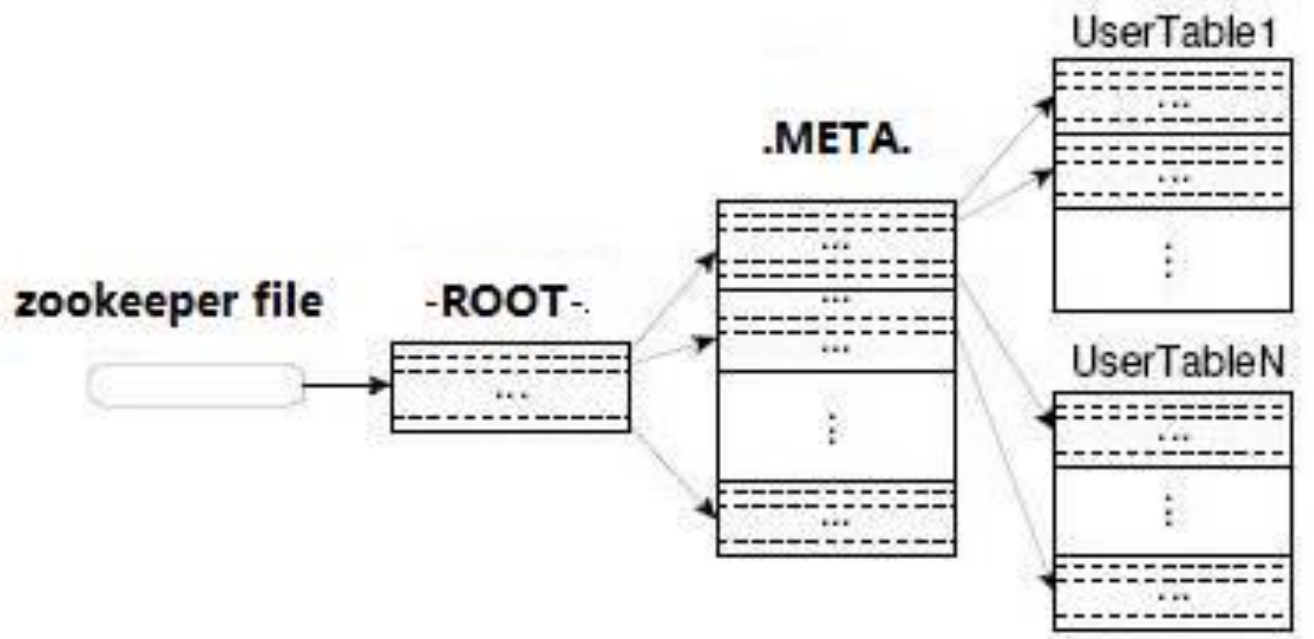
RS2

-ROOT- 和 .META.表

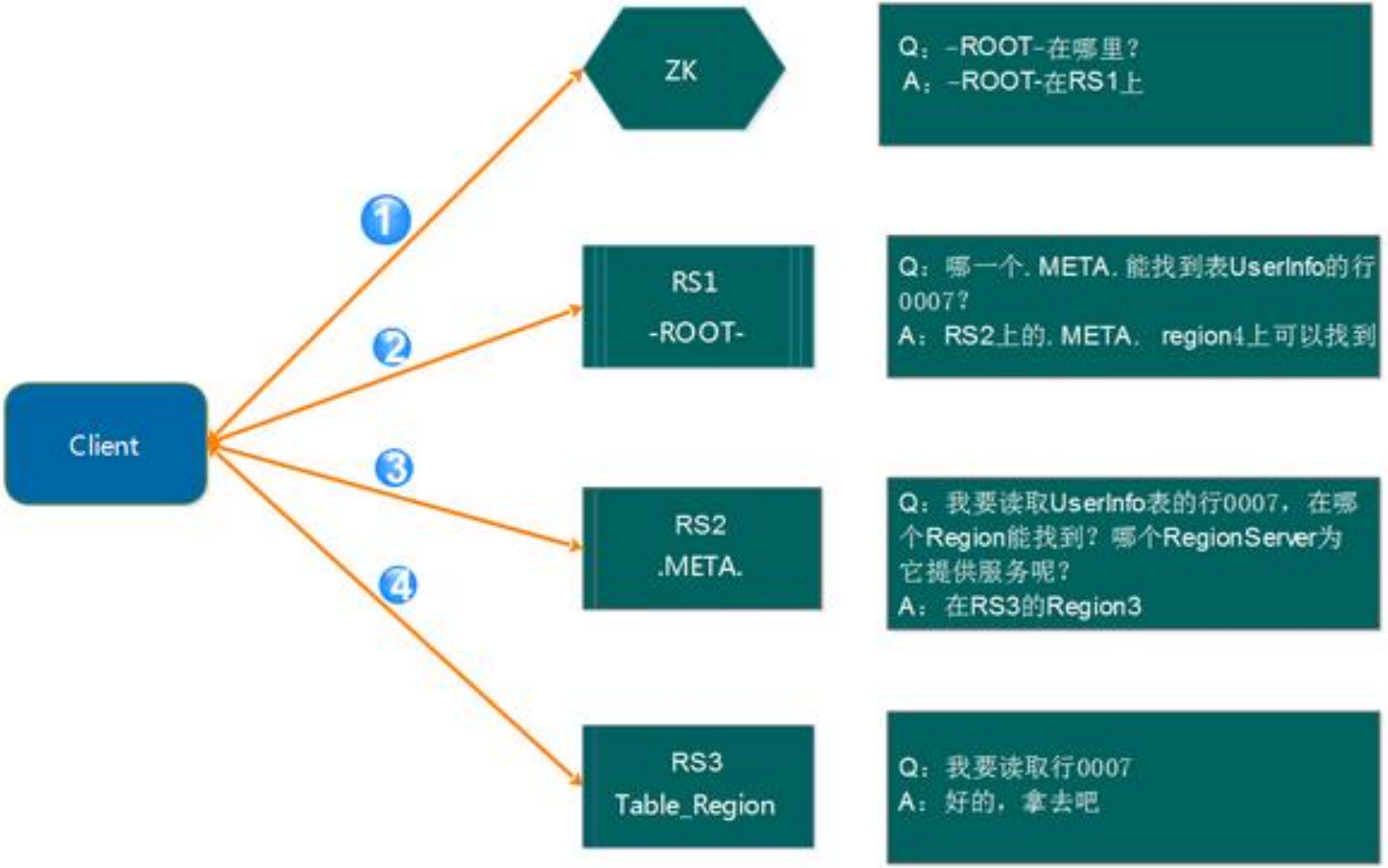
.META.：记录了用户表的Region信息，.META.可以有多个 Regoins

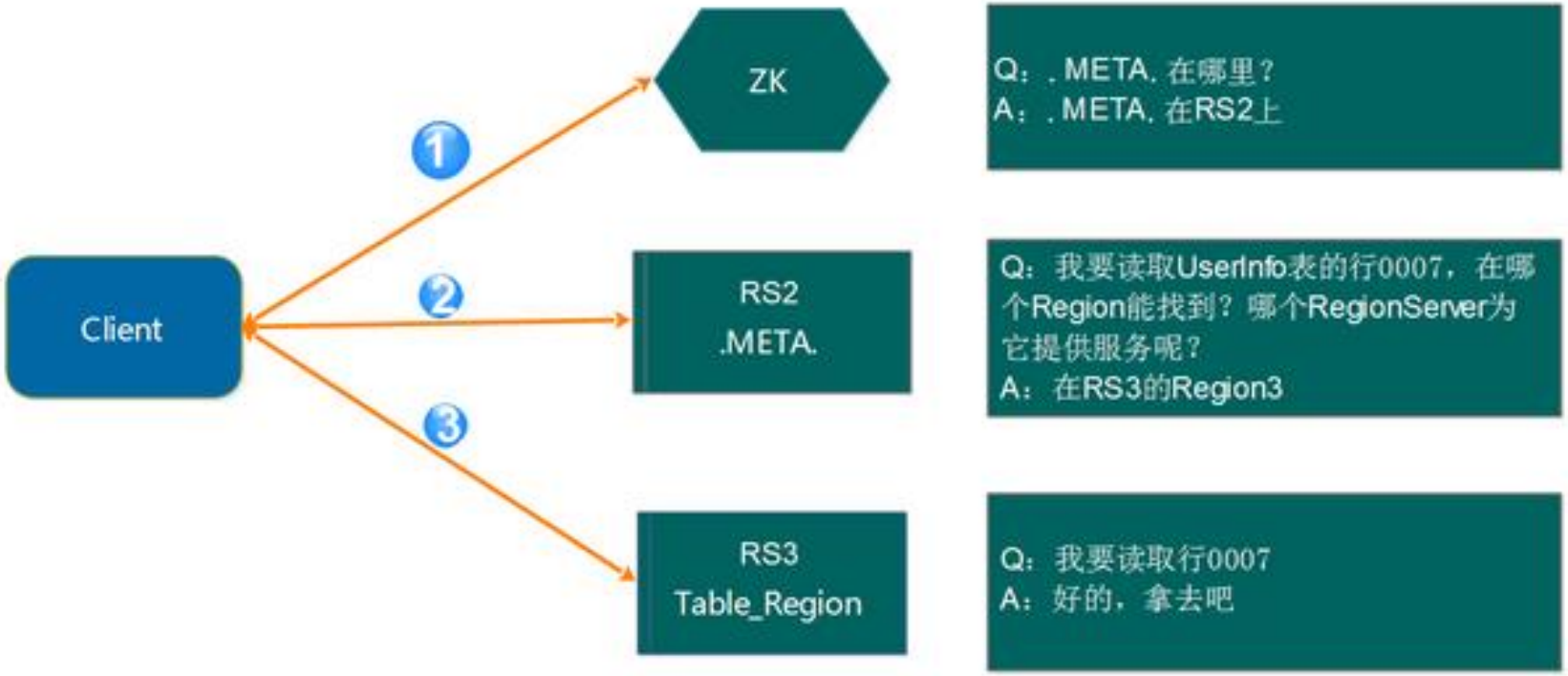
-ROOT-：记录了.META.表的Region信息，-ROOT-只有一个 Region

Zookeeper中记录了-ROOT-表的location

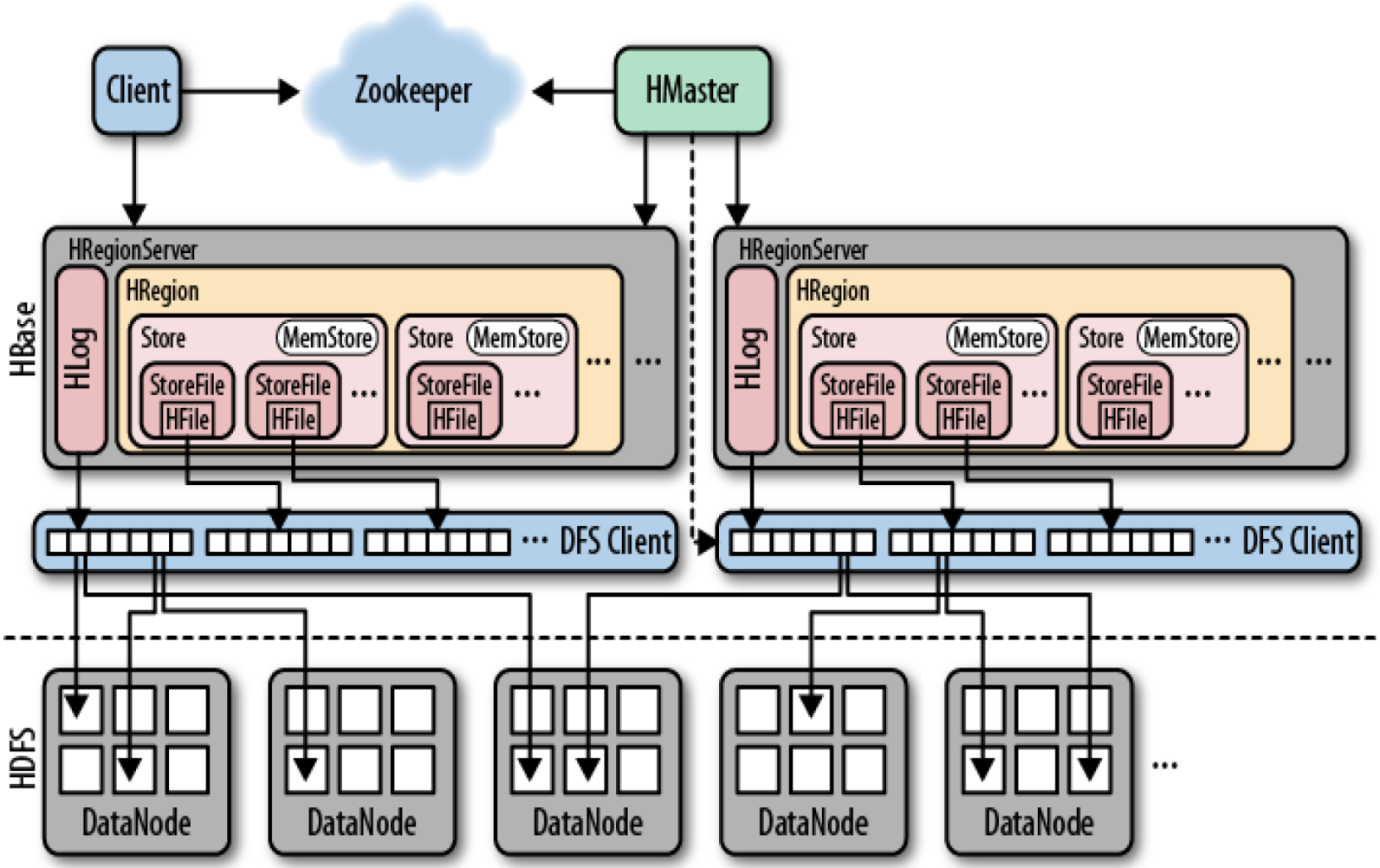


-ROOT- 和 .META.表


ZK
Q：-ROOT-在哪里？
A：-ROOT-在RS1上
1
RS1
-ROOT-
Q：哪一个.META.能找到表UserInfo的行0007？
A：RS2上的.META.，region4上可以找到
2
Client
3
RS2
.META.
Q：我要读取UserInfo表的行0007，在哪个Region能找到？哪个RegionServer为它提供服务呢？
A：在RS3的Region3
4
RS3
Table_Region
Q：我要读取行0007
A：好的，拿去吧


ZK
Client
RS2
.META.
RS3
Table_Region
1
2
3
Q：.META.在哪里？
A：.META.在RS2上
Q：我要读取UserInfo表的行0007，在哪个Region能找到？哪个RegionServer为它提供服务呢？
A：在RS3的Region3
Q：我要读取行0007
A：好的，拿去吧

HBase结构

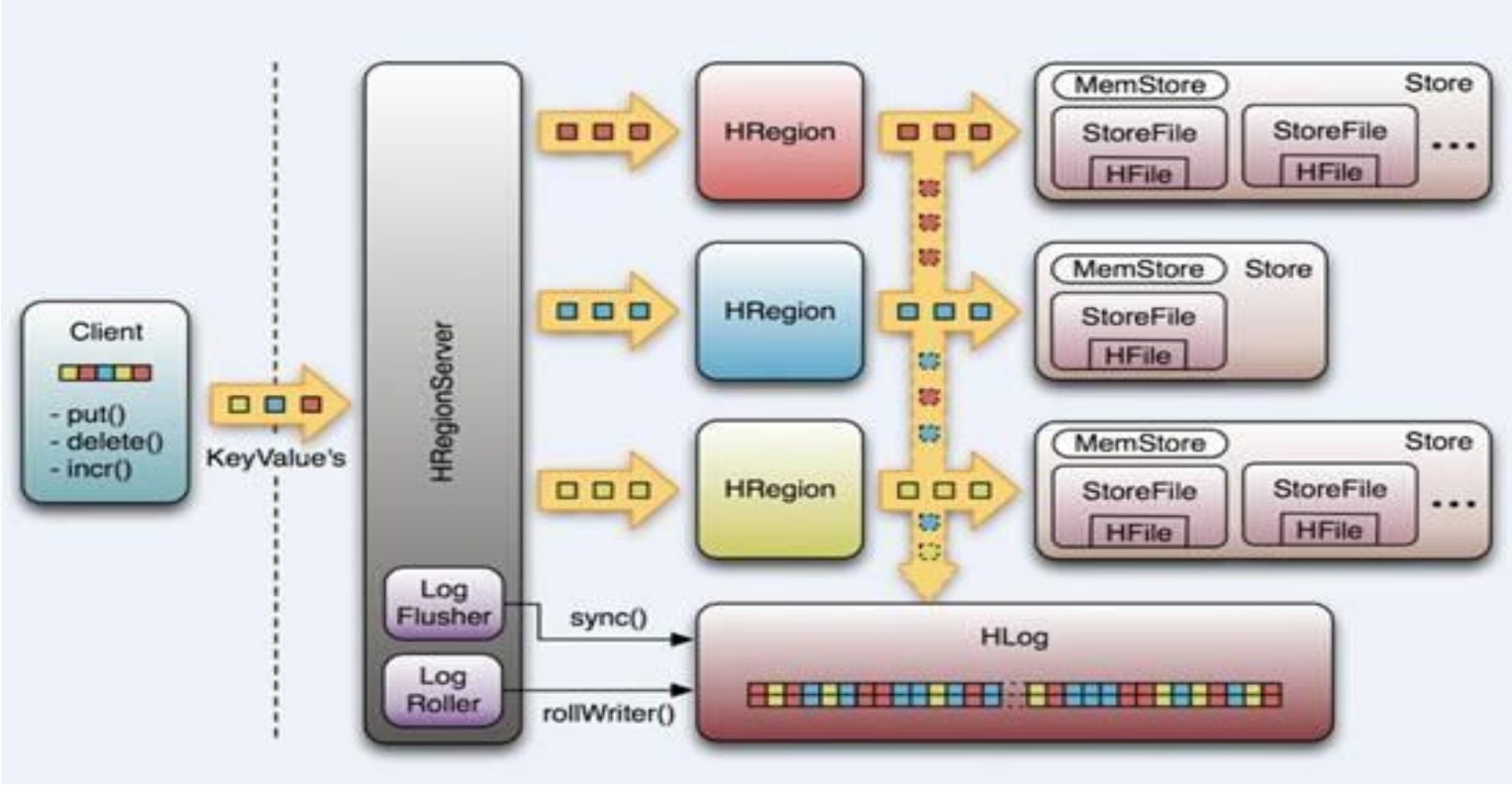

Client
Zookeeper
HMaster
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
HBase
HDFS
DataNode

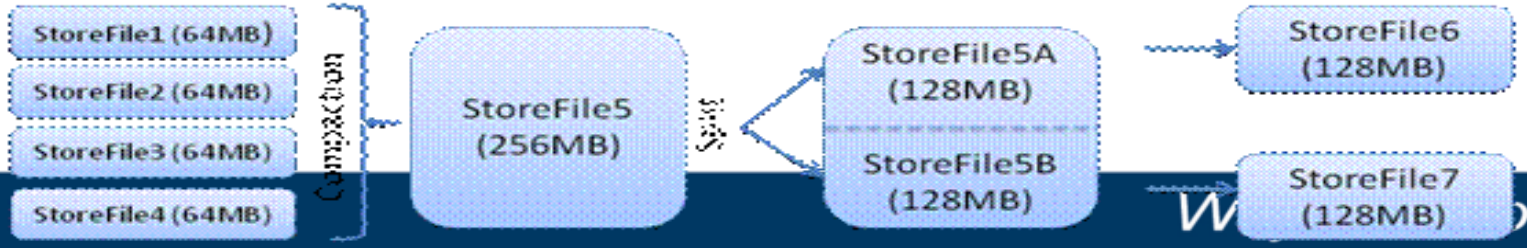
Hbase访问接口

- Native Java API，最常规和高效的访问方式，适合Hadoop MapReduce Job并行批处理HBase表数据
- HBase Shell，HBase的命令行工具，最简单的接口，适合HBase管理使用
- Thrift Gateway，利用Thrift序列化技术，支持C++，PHP，Python等多种语言，适合其他异构系统在线访问HBase表数据
- REST Gateway，支持REST 风格的Http API访问HBase, 解除了语言限制
- Pig，可以使用Pig Latin流式编程语言来操作HBase中的数据，和Hive类似，本质最终也是编译成MapReduce Job来处理HBase表数据，适合做数据统计
- Hive，Hive 0.7.0中支持HBase，可以使用类似SQL语言来访问HBase

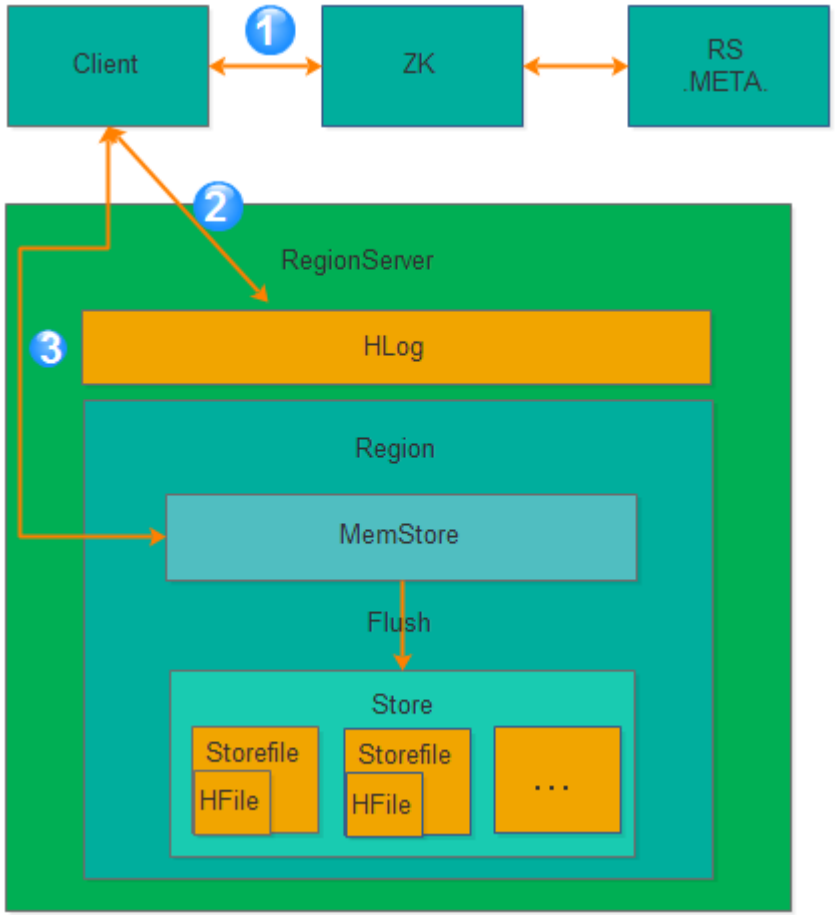
HRegionServer

HRegionServer主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBase中最核心的模块。





Hbase写入逻辑



①Client获取数据写入的Region所在的RegionServer

②请求写Hlog

③请求写MemStore。只有当写Hlog和写MemStore都成功了才算请求写入完成。MemStore后续会逐渐刷到HDFS中。

Hlog存储在HDFS，当RegionServer出现异常，需要使用Hlog来恢复数据。

MemStore刷盘

为了提高Hbase的写入性能，当写请求写入MemStore后，不会立即刷盘。而是会等到一定的时候进行刷盘的操作，总结成如下的几个场景：

- **全局内存控制**

这个全局的参数是控制内存整体的使用情况，当所有memstore占整个heap的最大比例的时候，会触发刷盘的操作

- **MemStore达到上限**

默认128M大小

- **RegionServer的Hlog数量达到上限**

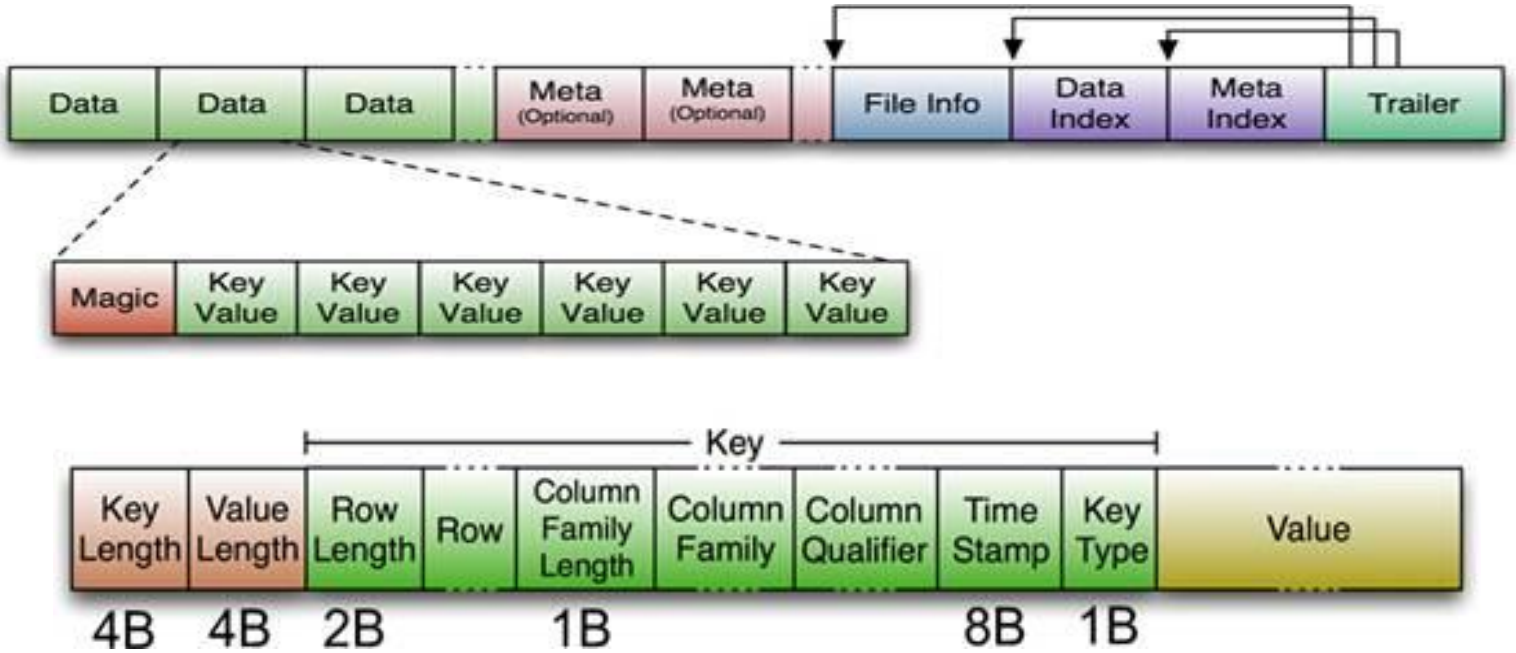
前面说到Hlog为了保证Hbase数据的一致性，那么如果Hlog太多的话，导致故障恢复的时间太长，因此Hbase会对Hlog的最大个数做限制。当达到Hlog的最大个数的时候，会强制刷盘。

- **手工触发**
- **关闭RegionServer触发**
- **Region使用HLOG恢复完数据后触发**

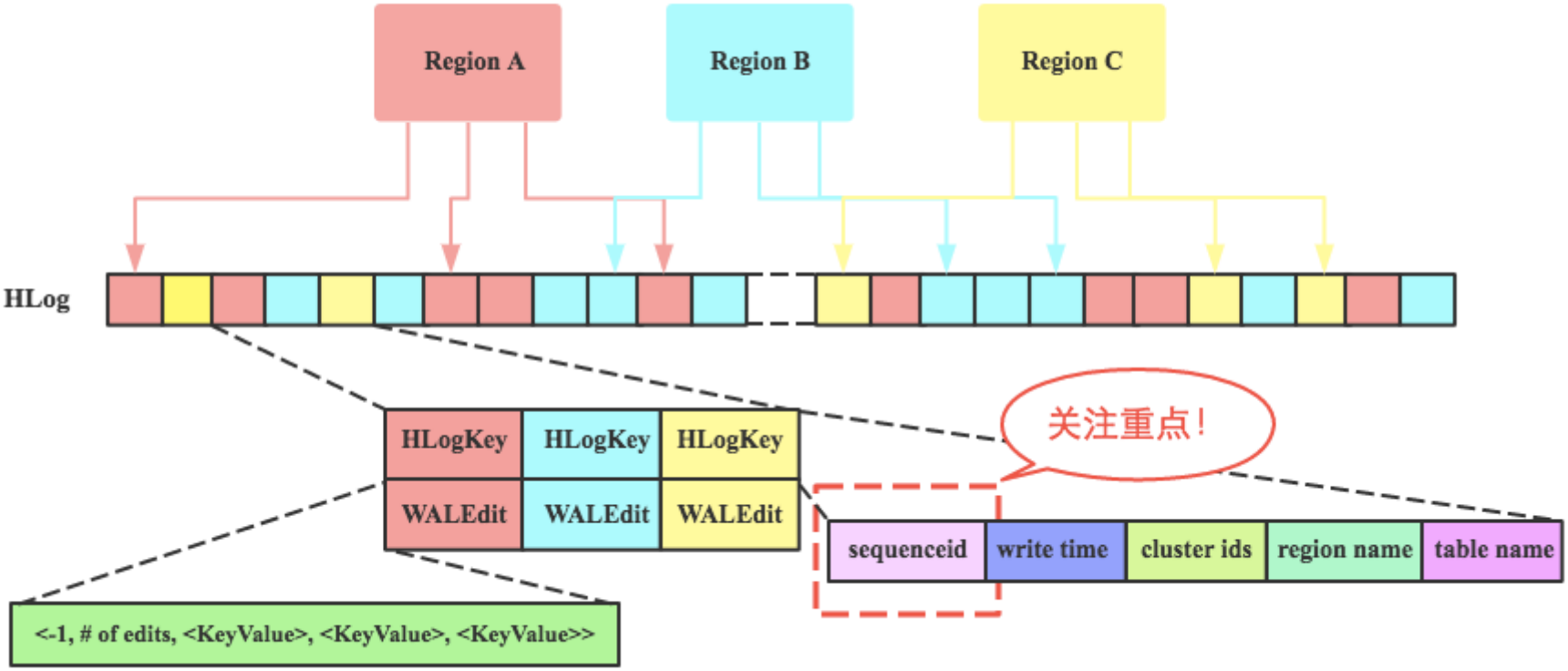
Hbase的存储格式

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

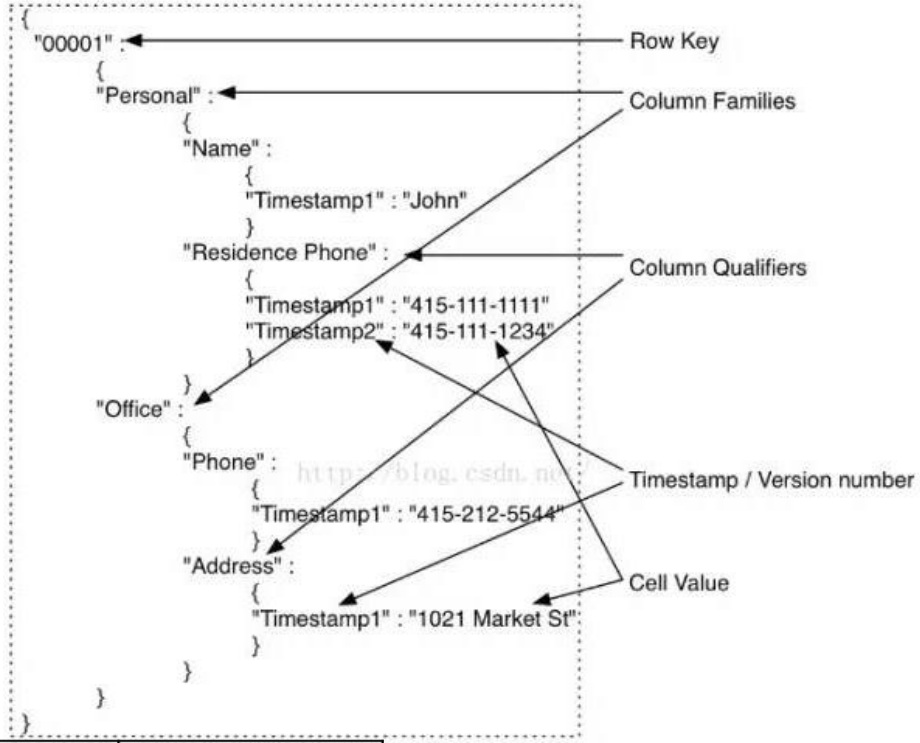


- **HLog File**，HBase中WAL（Write Ahead Log） 的存储格式，物理上是Hadoop的Sequence File



HBase - Code

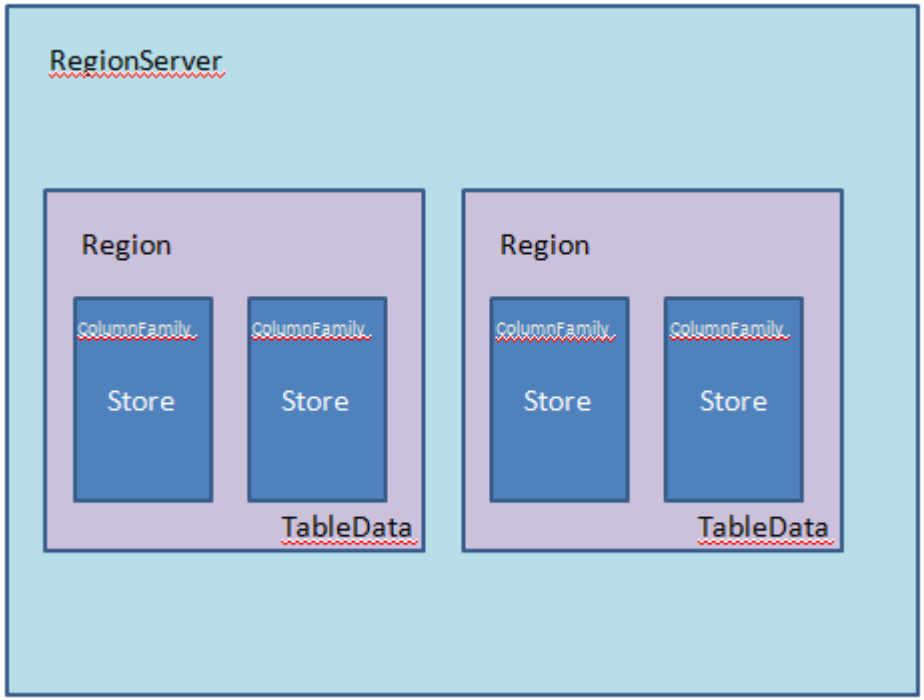
```
HTable table = …
Text row = new Text("enclosure1");
Text col1 = new Text("animal:type");
Text col2 = new Text("animal:size");
BatchUpdate update = new BatchUpdate(row);
update.put(col1, "lion".getBytes("UTF-8"));
update.put(col2, "big".getBytes("UTF-8));
table.commit(update);

update = new BatchUpdate(row);
update.put(col1, "zebra".getBytes("UTF-8"));
table.commit(update);
```

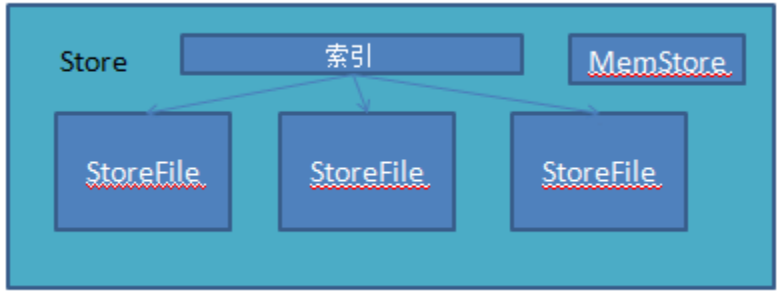


Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	…	…	…	…

HBase - Code

Hbase是面向列存储的数据库，所以在一个Region中，有多个文件来存储这些列。Hbase中数据列是由列簇来组织的，所以每一个列簇都会有对应的一个数据结构，Hbase将列簇的存储数据结构抽象为Store，一个Store代表一个列簇。





Log-Structure-Merge树(1)

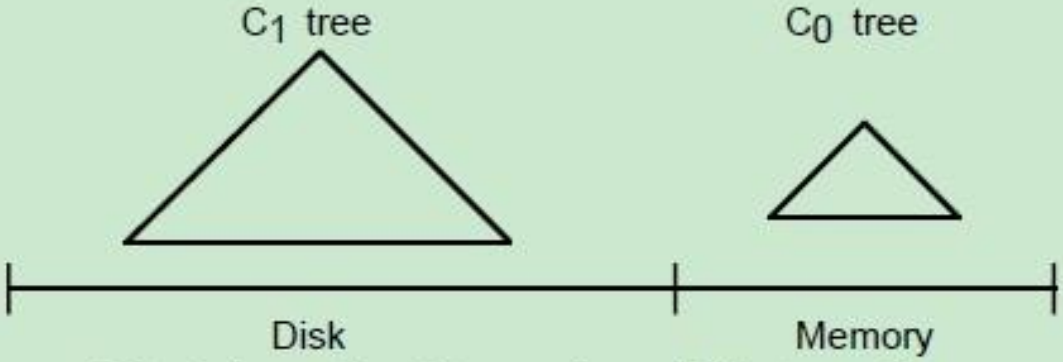


Figure 2.1. Schematic picture of an LSM-tree of two components

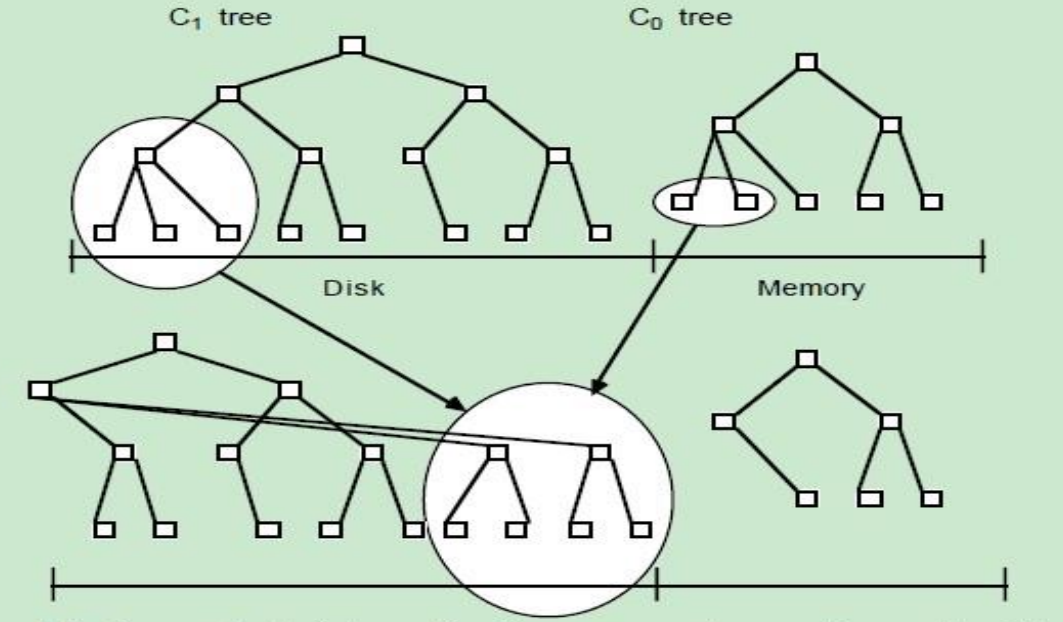
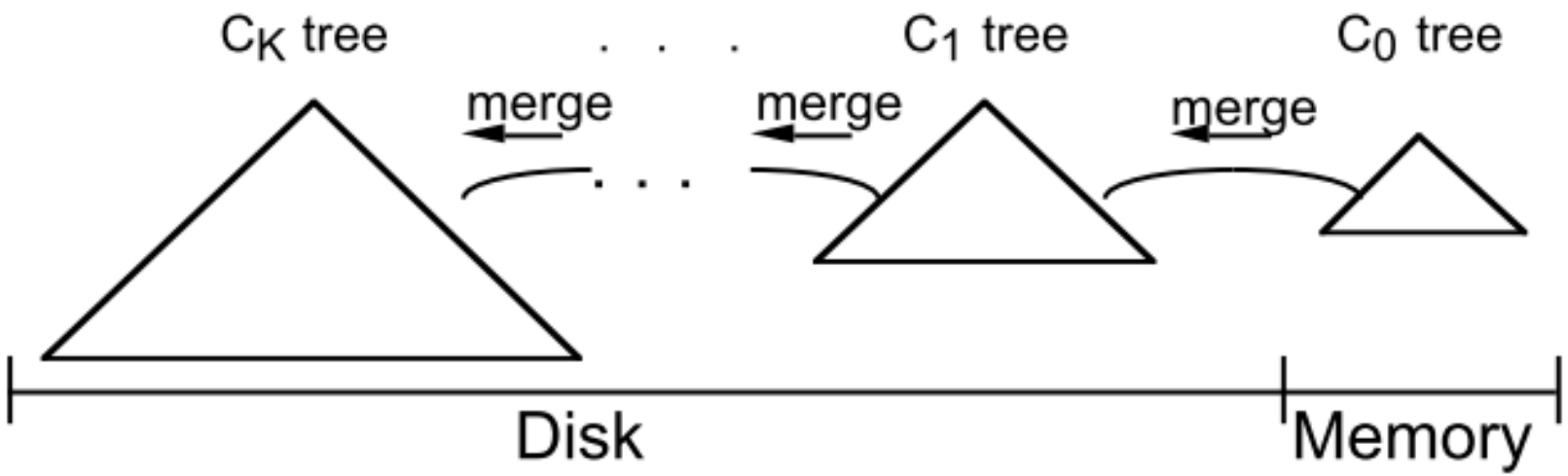


Figure 2.2. Conceptual picture of rolling merge steps, with result written back to disk

Log-Structure-Merge树(2)



HBase – 查询

- Retrieve a cell

  Cell = table.getRow("enclosure1").getColumn("animal:type").getValue();

- Retrieve a row

  RowResult = table.getRow("enclosure1");

- Scan through a range of rows

  Scanner s = table.getScanner(new String[] { "animal:type" });

Hive

- Facebook开发
- 用于大部分Facebook jobs
- 建"关系数据库" 于Hadoop之上
  - Maintains list of table schemas
  - SQL-like query language (HiveQL)
  - 可从HiveQL调用Hadoop Streaming 脚本
  - Supports table partitioning, clustering, complex data types, some optimizations

- 直接使用 MapReduce 所面临的问题：
  1、人员学习成本太高
  2、项目周期要求太短
  3、MapReduce实现复杂查询逻辑开发难度太大
- 为什么要使用 Hive：
  1、更友好的接口：操作接口采用类 SQL 的语法，提供快速开发的能力
  2、更低的学习成本：避免了写 MapReduce，减少开发人员的学习成本
  3、更好的扩展性：可自由扩展集群规模而无需重启服务，还支持用户自定义函数

建立一个Hive Table

```
CREATE TABLE page_views(viewTime INT, userid BIGINT,
                        page_url STRING, referrer_url STRING,
                        ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

- Partitioning breaks table into separate files for each (dt, country) pair

  Ex: /hive/page_view/dt=2008-06-08,country=USA

  /hive/page_view/dt=2008-06-08,country=CA

一个简单查询

- 找出来自于xyz.com的在3月31号的所有page views：

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
AND page_views.referrer_url like '%xyz.com';
```

- Hive只读partition 2008-03-01,* 而不是扫描整个表。

Aggregation and Joins

- Count users who visited each page by gender:

```
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

- Sample output:

page_url	gender	count(userid)
home.php	MALE	12,141,412
home.php	FEMALE	15,431,579
photo.php	MALE	23,941,451
photo.php	FEMALE	21,231,314

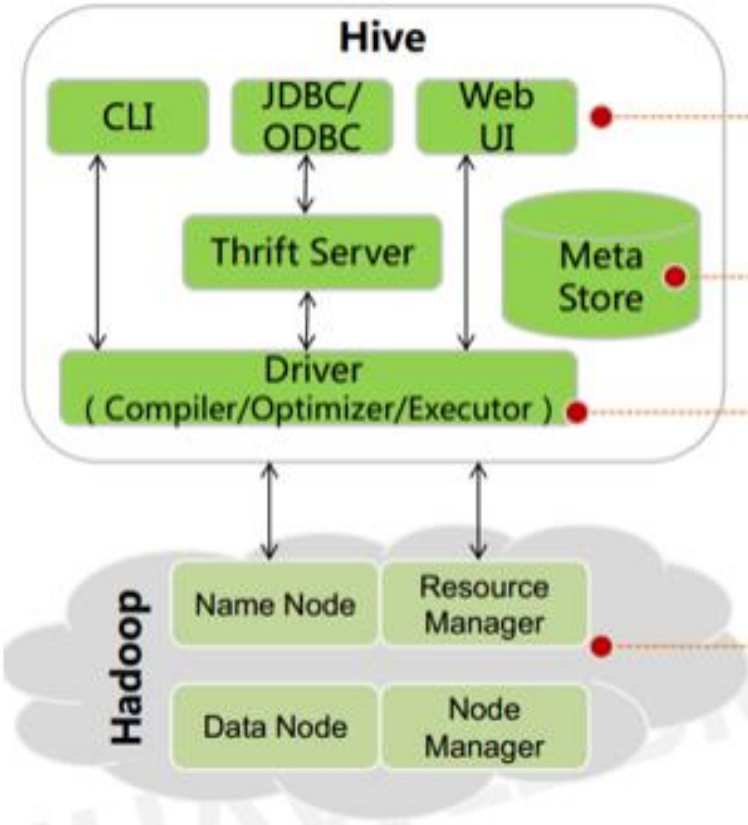
使用Hadoop流Mapper脚本

```
SELECT TRANSFORM(page_views.userid,
                                 page_views.date)
USING 'map_script.py'
AS dt, uid CLUSTER BY dt
FROM page_views;
```

Hive的优缺点

- **可扩展性,横向扩展**，Hive 可以自由的扩展集群的规模，一般情况下不需要重启服务 横向扩展：通过分担压力的方式扩展集群的规模 纵向扩展：一台服务器cpu i7-6700k 4核心8线程，8核心16线程，内存64G => 128G
- **延展性**，Hive 支持自定义函数，用户可以根据自己的需求来实现自己的函数
- **良好的容错性**，可以保障即使有节点出现问题，SQL 语句仍可完成执行
- **Hive 不支持记录级别的增删改操作**，但是用户可以通过查询生成新表或者将查询结 果导入到文件中
- **Hive 的查询延时很严重**，因为 MapReduce Job 的启动过程消耗很长时间，所以不能用在交互查询系统中。
- **Hive 不支持事务**（因为没有增删改，所以主要用来做 OLAP（联机分析处理），而 不是 OLTP（联机事务处理），这就是数据处理的两大级别）。

Hive架构



Hive的用户接口层，CLI即Shell命令行，CLI最常用。

Hive将元数据存储在数据库中，连接到这些数据库(mysql, derby)的模式分三种：单用户模式、多用户模式、远程服务器模式。

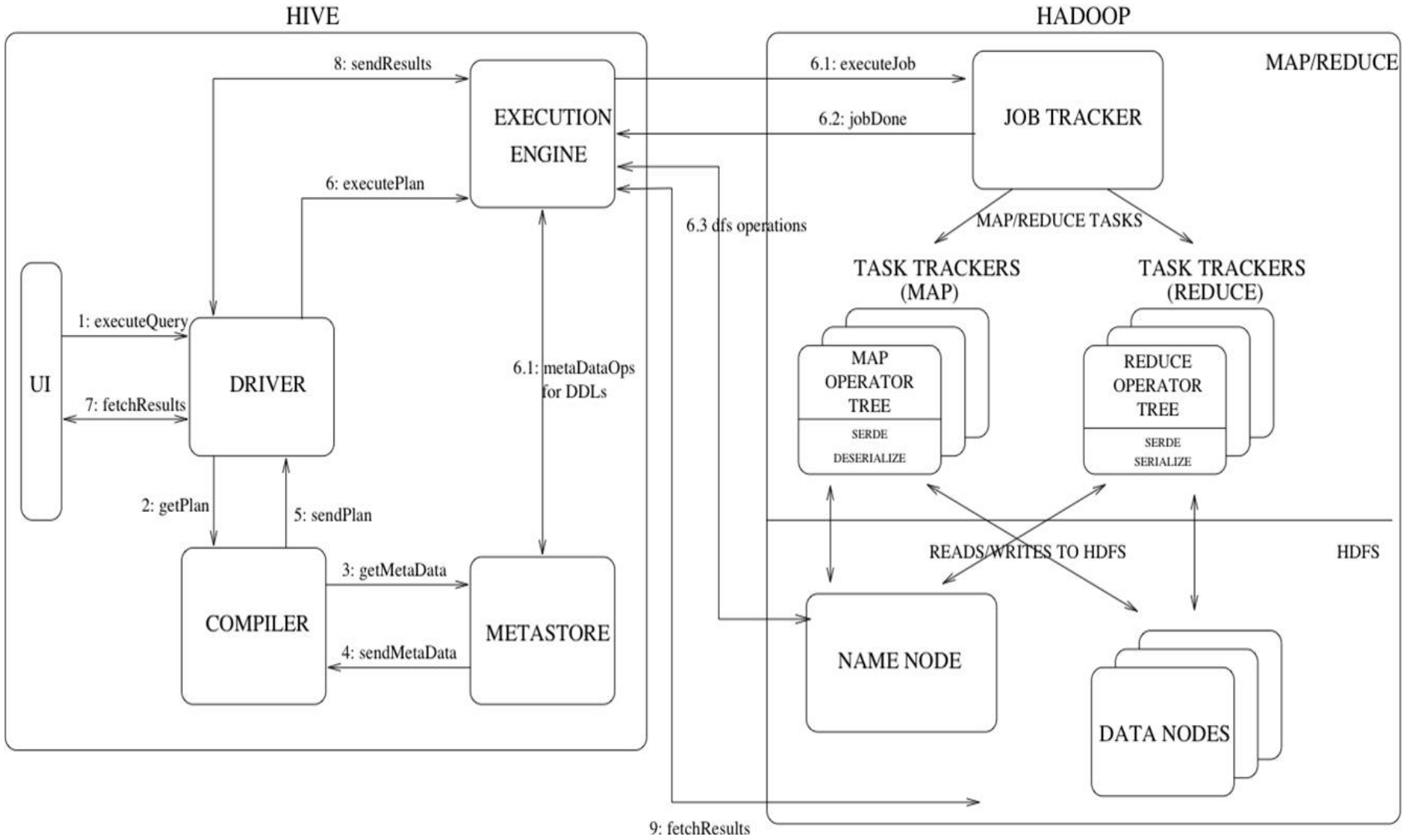
元数据包括Database、表名、表的列及类型、存储空间、分区、表数据所在目录等。

完成HQL的查询语句的词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在HDFS中，并由MapReduce调用执行。

Hive的数据存储在HDFS中，针对大部分的HQL查询请求，Hive内部自动转换为MapReduce任务执行。

70

Hive的执行流程



71

1、用户接口: shell/CLI, jdbc/odbc, webui Command Line Interface

CLI，Shell 终端命令行（Command Line Interface），采用交互形式使用 Hive 命令行与 Hive 进行交互，最常用（学习，调试，生产）

JDBC/ODBC，是 Hive 的基于 JDBC 操作提供的客户端，用户（开发员，运维人员）通过 这连接至 Hive server 服务

Web UI，通过浏览器访问 Hive

2、跨语言服务： thrift server 提供了一种能力，让用户可以使用多种不同的语言来操纵hive

Thrift 是 Facebook 开发的一个软件框架，可以用来进行可扩展且跨语言的服务的开发， Hive 集成了该服务，能让不同的编程语言调用 Hive 的接口

3、底层的Driver： 驱动器Driver，编译器Compiler，优化器Optimizer，执行器Executor

Driver 组件完成 HQL 查询语句从词法分析，语法分析，编译，优化，以及生成逻辑执行 计划的生成。生成的逻辑执行计划存储在 HDFS 中，并随后由 MapReduce 调用执行

Hive 的核心是驱动引擎， 驱动引擎由四部分组成：

(1) 解释器：解释器的作用是将 HiveSQL 语句转换为抽象语法树（AST）

(2) 编译器：编译器是将语法树编译为逻辑执行计划

(3) 优化器：优化器是对逻辑执行计划进行优化

(4) 执行器：执行器是调用底层的运行框架执行逻辑执行计划

4、元数据存储系统： RDBMS MySQL

元数据，通俗的讲，就是存储在 Hive 中的数据的描述信息。

Hive 中的元数据通常包括：表的名字，表的列和分区及其属性，表的属性（内部表和 外部表），表的数据所在目录

Metastore 默认存在自带的 Derby 数据库中。缺点就是不适合多用户操作，并且数据存 储目录不固定。数据库跟着 Hive 走，极度不方便管理

解决方案：通常存我们自己创建的 MySQL 库（本地 或 远程）

Hive 和 MySQL 之间通过 MetaStore 服务交互

Version Control System

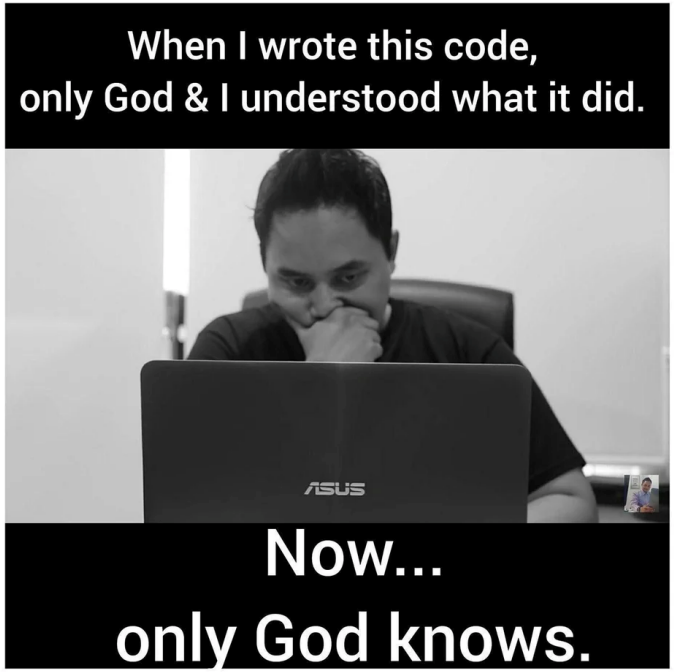
注意：项目作业需使用该工具，并在实验报告中提供Github链接

Version Control System - Git

- What is Git?

  Git is a distributed (decentralized) version-control system for tracking changes in source code during software development.

- Why Git?
  - Keep tracking changes
  - Revert to a specific checking point
  - Work with people



Version Control System - Git

- Git is not the only VCS, but by far the most popular
  - Subversion (SVN) – e.g. Google Code (2006 – 2016)
  - Concurrent Versions System (CVS) – Grandpa of VCS
  - Mercurial – much simpler than Git
- Google Drive / OneDrive is **NOT** VCS!!!!
- Git service providers
  - Bitbucket
  - Github
  - Gitlab

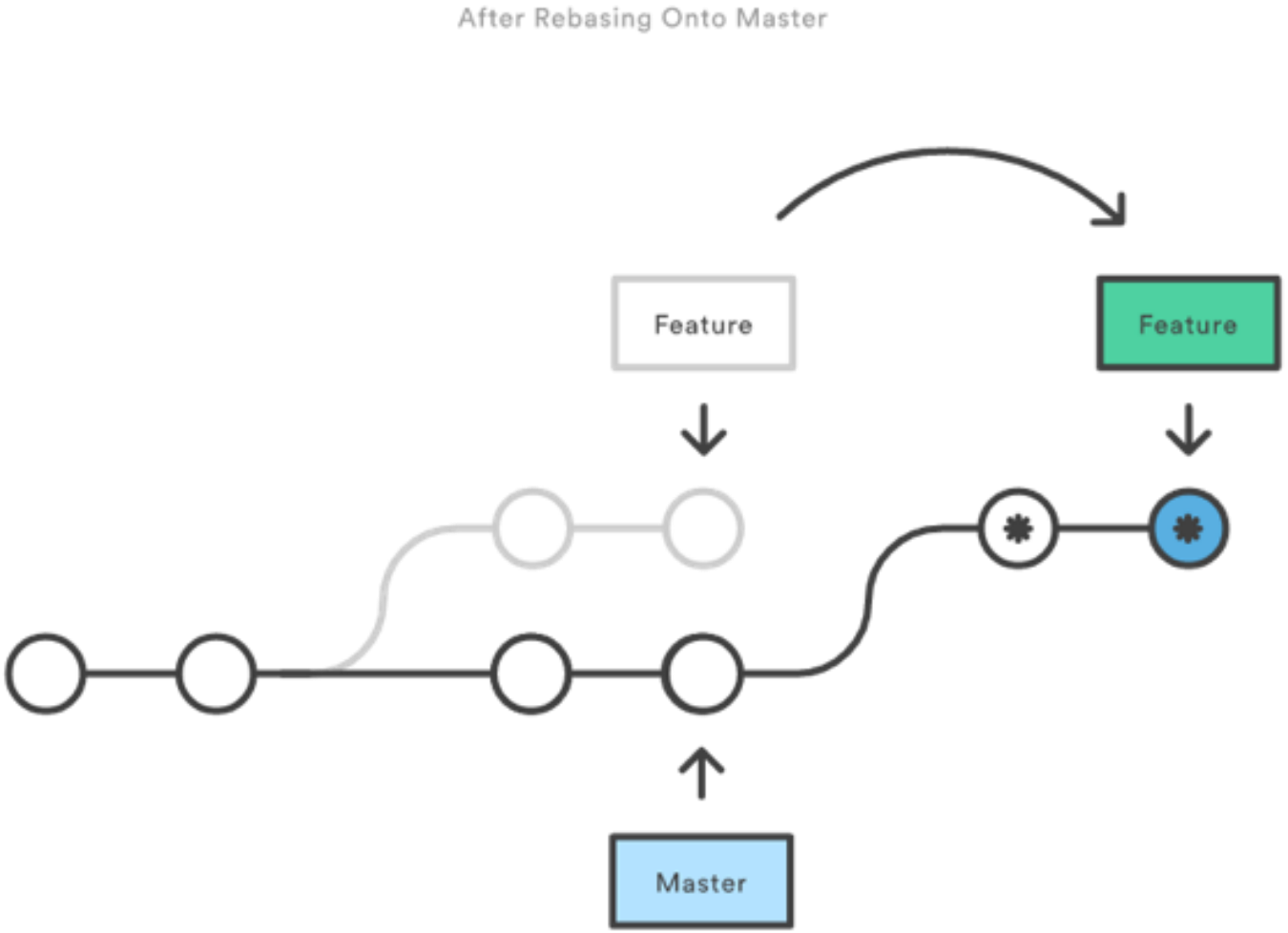
Git: Installation

- macOS
  - *$ brew update*
  - *$ brew install git*
- Linux (Ubuntu)
  - *$ sudo apt-get update*
  - *$ sudo apt-get install git*
- Windows (Git client, WSL, Git GUI client …)

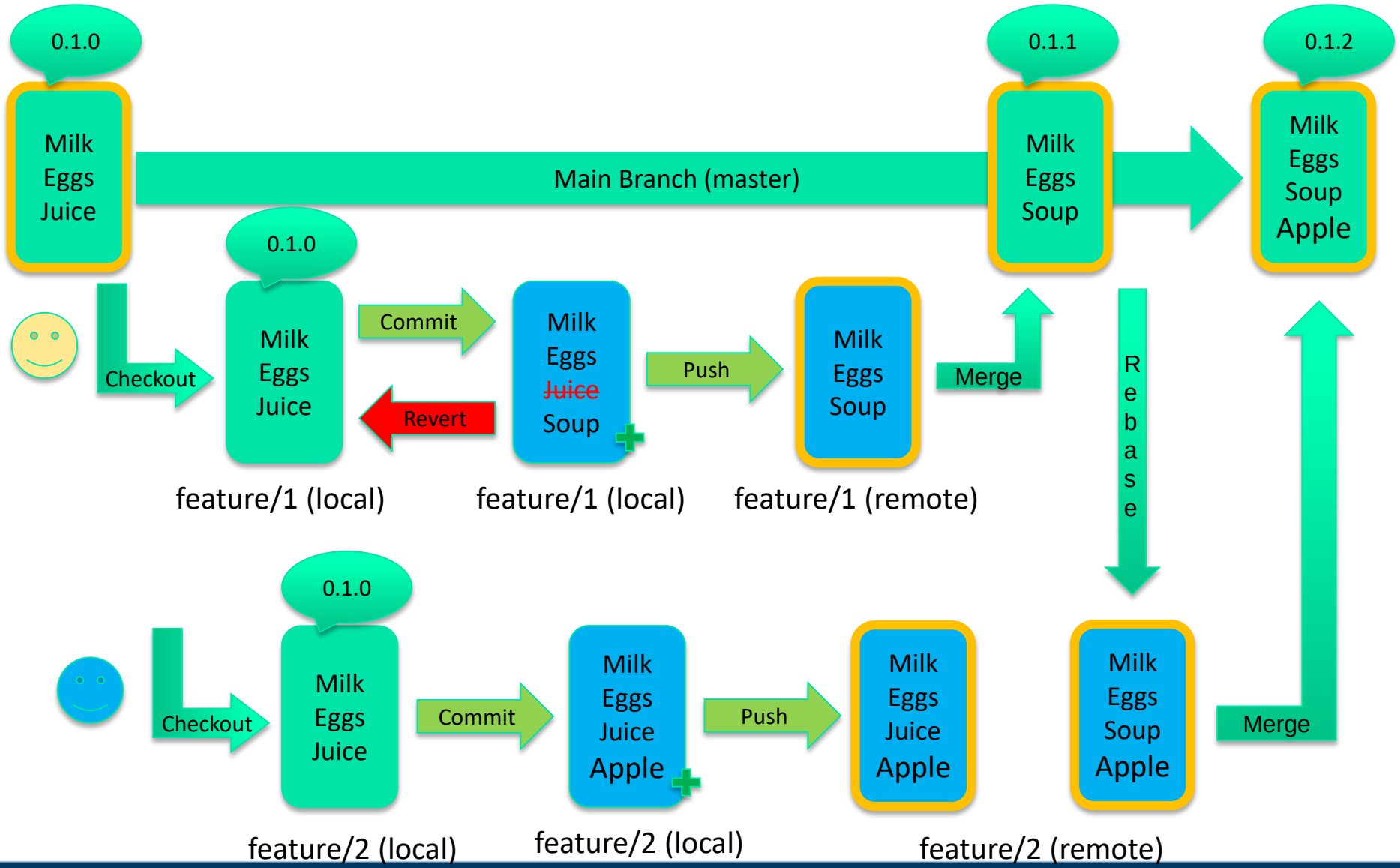
Git Glossary

- **Repository**: a collection of commits, branches and tags.
- **Commit**: an individual change to a file. Like when you save a file, but with a hash that allows you to keep a record of what changes were made and by who.
- **Branch**: represents an independent line of development. It is a parallel version of repository.
- **Tag:** A reference typically used to mark a particular point in the commit chain.
- **Pull**: Fetching changes and merging them.
- **Push:** Sending your changes to a remote repository.
- **Remote:** A version of something that is hosted on a server (Bitbucket, Github, Gitlab).
- **Merge:** take changes from one branch into another.
- **Rebase:** To reapply a series of changes from one branch to another, and reset the head of that branch to the result.

Rebase

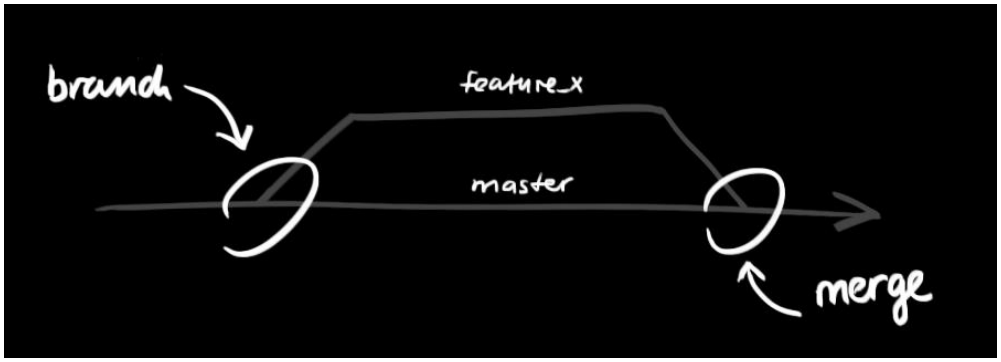


Git at a glance



Git: a simple guide

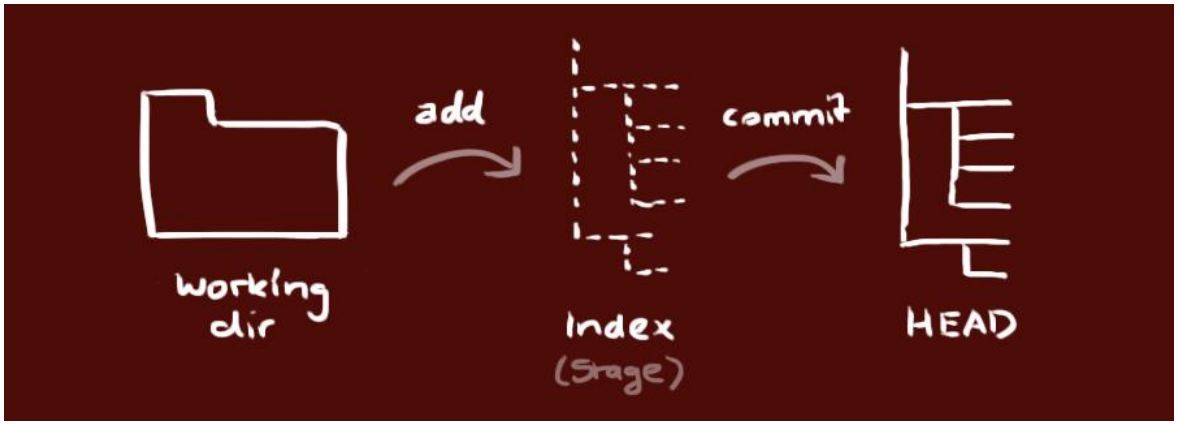
- Branching



  - Diverge from the main line of development and do work without messing with the main line
  - Main branch: e.g. master
  - Development branch: e.g. develop
  - Other branches:
    - **Feature branch**: add a new feature, e.g. feature/add-a-new-feature
    - **Bugfix branch**: fix a bug, e.g. bugfix/fix-a-bug
    - **Release branch**: make a release, e.g. release/1.0
    - **Hotfix branch**: hotfix a bug in a release, e.g. hotfix/fix-another-bug

Git: a simple guide

- Simple Workflow



- Made changes
- Add

  $ git add <filename>
- Commit

  $ git commit -m “commit message”
- Push

  $ git push origin feature/demo-1

  $ git push --set-upstream origin feature/demo-1

Git: a simple guide

- Register an account on Bitbucket, Github or Gitlab
  - Bitbucket - https://bitbucket.org
  - Github - https://github.com/
  - Gitlab - https://gitlab.com
- Create a git repository (Local)

  Create a new directory

  $ git init
- Create a git repository (Bitbucket)
- Checkout a new repository

  $ git clone https://<username>@<hostname>/path/to/repo.git

Git: a simple guide

Branching

- Create a develop branch (from GUI)
- Create two feature branches

  $ git checkout -b feature/demo-1

  $ git checkout -b feature/demo-1-1
- Switch between branches

  $ git checkout feature/demo-1
- List all branches

  $ git branch
- Delete a branch

  $ git branch -d feature/demo-1-1
- Push a local branch to remote

  $ git push origin feature/demo-1

Git: a simple guide

- Pull Request (Merge Request) – Code review
  - Create Pull Request for others to review your code
  - Approve or Decline a Pull Request
  - Merge a Pull Request when all reviewers approved

- Fetch and pull the changes from remote
  - Fetch updates

    $ git fetch -a
  - Pull changes from remote

    $ git pull

Git: a simple guide

Conflicts

- Conflict may happen if users changed the same code block
- Code cannot be merged until conflicts are resolved
- Scenario
  - User A checked out *develop* branch and changed one line in README.md on feature branch *feature/demo-2*

```
10  10
11      - You'll start by editing this README.md file to learn how to edit a file in Bitbucket.
    11  + You'll start by editing this README.txt file to learn how to edit a file in Bitbucket.
12  12
```

  - At the same time User B checked out *develop* branch and changed the same line in README.md on feature branch *feature/demo-3*

```
10  10
11      - You'll start by editing this README.md file to learn how to edit a file in Bitbucket.
    11  + You'll start by editing this README-txt file to learn how to edit a file in Bitbucket.
12  12
```

  - User A merged the change to *develop* branch
  - User B created a Pull Request to merge *feature/demo-3* to *develop* branch and saw conflict

Git: a simple guide

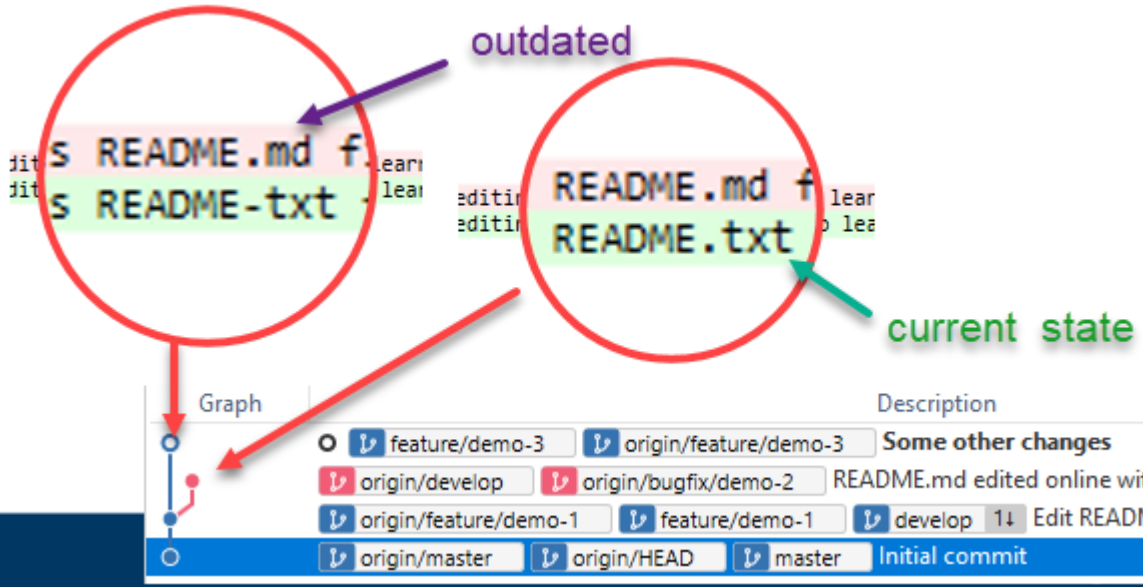
Conflicts

README.md CONFLICT

Side-by-side diff | View file | Comment | •••

Conflict: File modified in both source and destination
To be able to merge you will need to resolve the conflicts manually. Learn how.

```
...
 8   8
 9   9  ## Edit a file
10  10
    11 +<<<<<<< destination:8f0f6df83f494f0803ca8b92425dd9902285b4eb
11  12  You'll start by editing this README.txt file to learn how to edit a file in Bitbucket.
    13 +=======
    14 +You'll start by editing this README-txt file to learn how to edit a file in Bitbucket.
    15 +>>>>>>> source:d7f6d93ff63c16106dce51b645d6054eb30eba0e
12  16
```



- Rebase and resolve the conflict (use tool)

$ git fetch -a

$ git pull --rebase origin develop

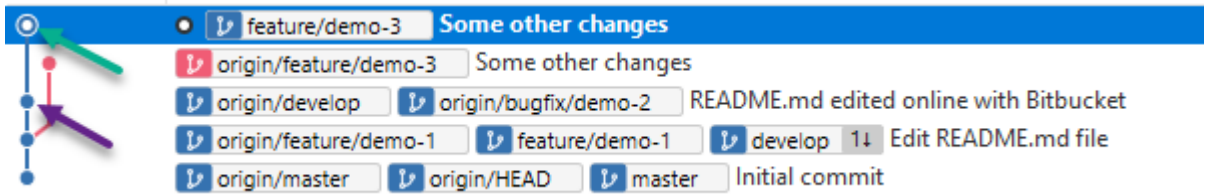
```
## Edit a file
```

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<<< HEAD (Current Change)
You'll start by editing this README.txt file to learn how to edit a file in Bitbucket.
=======
You'll start by editing this README-txt file to learn how to edit a file in Bitbucket.
>>>>>>> Some other changes (Incoming Change)

1. Click **Source** on the left side.
```

$ git add READM.md

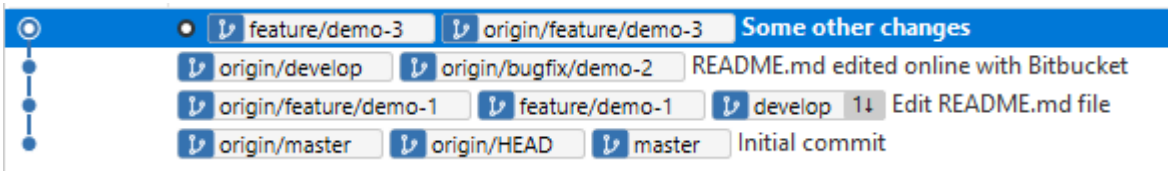
$ git rebase --continue



Git: a simple guide

- Rebase and resolve the conflict (use tool)

$ git push -f origin feature/demo-3



README.md MODIFIED — Side-by-side diff | View file | Comment | •••

```
...
 8   8
 9   9  ## Edit a file
10  10
11      -You'll start by editing this README.txt file to learn how to edit a file in Bitbucket.
    11  +You'll start by editing this README-txt file to learn how to edit a file in Bitbucket.
12  12
13  13  1. Click **Source** on the left side.
14  14  2. Click the README.md link from the list of files.
...
```