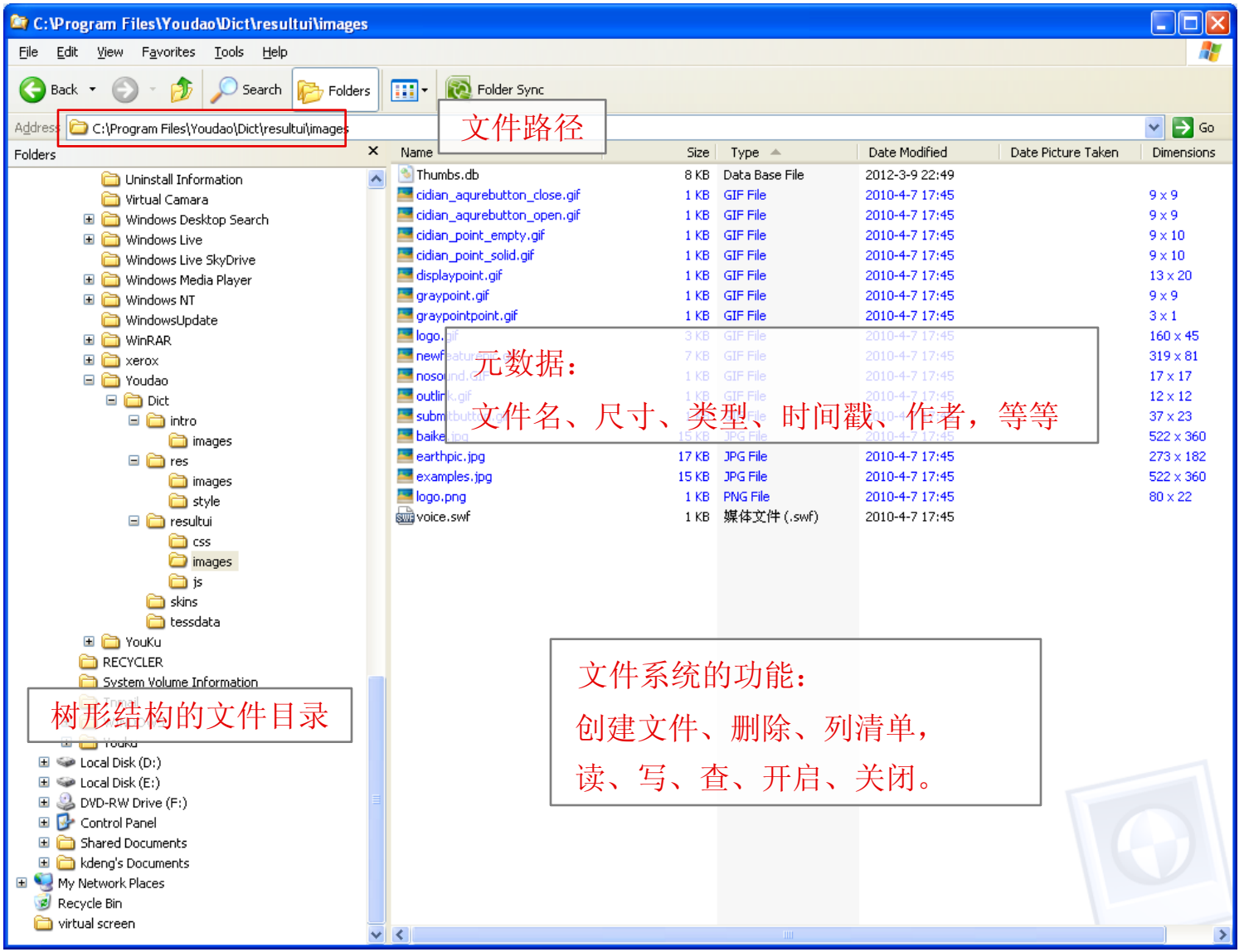
The 4th China Cloud Computing Conference
第四届中国云计算大会
5月23-25日 北京·国家会议中心

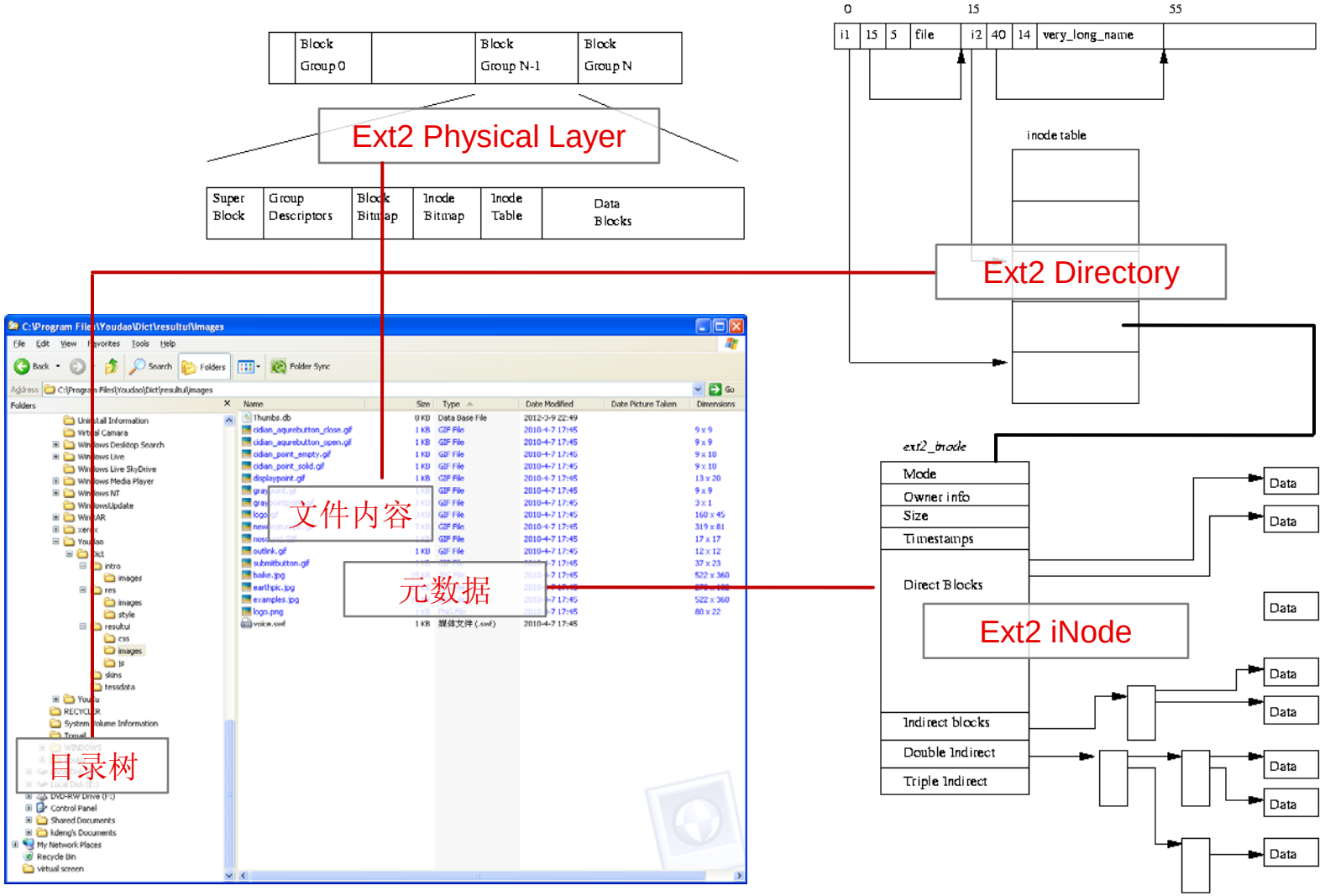
云存储系统设计

邓侃博士 dengkan@smartclouder.com

单机文件系统面临的问题


C:\Program Files\Youdao\Dict\resultui\images
File Edit View Favorites Tools Help
Back Search Folders Folder Sync
Address C:\Program Files\Youdao\Dict\resultui\images
Go
文件路径
Folders
Name Size Type Date Modified Date Picture Taken Dimensions
Thumbs.db 8 KB Data Base File 2012-3-9 22:49
cidian_aqurebutton_close.gif 1 KB GIF File 2010-4-7 17:45 9 x 9
cidian_aqurebutton_open.gif 1 KB GIF File 2010-4-7 17:45 9 x 9
cidian_point_empty.gif 1 KB GIF File 2010-4-7 17:45 9 x 10
cidian_point_solid.gif 1 KB GIF File 2010-4-7 17:45 9 x 10
displaypoint.gif 1 KB GIF File 2010-4-7 17:45 13 x 20
graypoint.gif 1 KB GIF File 2010-4-7 17:45 9 x 9
graypointpoint.gif 1 KB GIF File 2010-4-7 17:45 3 x 1
earthpic.jpg 17 KB JPG File 2010-4-7 17:45 273 x 182
examples.jpg 15 KB JPG File 2010-4-7 17:45 522 x 360
logo.png 1 KB PNG File 2010-4-7 17:45 80 x 22
voice.swf 1 KB 媒体文件 (.swf) 2010-4-7 17:45
元数据：
文件名、尺寸、类型、时间戳、作者，等等
Uninstall Information
Virtual Camara
Windows Desktop Search
Windows Live
Windows Live SkyDrive
Windows Media Player
Windows NT
WindowsUpdate
WinRAR
xerox
Youdao
Dict
intro
images
res
images
style
resultui
css
images
js
skins
tessdata
YouKu
RECYCLER
System Volume Information
树形结构的文件目录
Local Disk (D:)
Local Disk (E:)
DVD-RW Drive (F:)
Control Panel
Shared Documents
kdeng's Documents
My Network Places
Recycle Bin
virtual screen
文件系统的功能：
创建文件、删除、列清单，
读、写、查、开启、关闭。

Linux Ext2 文件系统数据结构

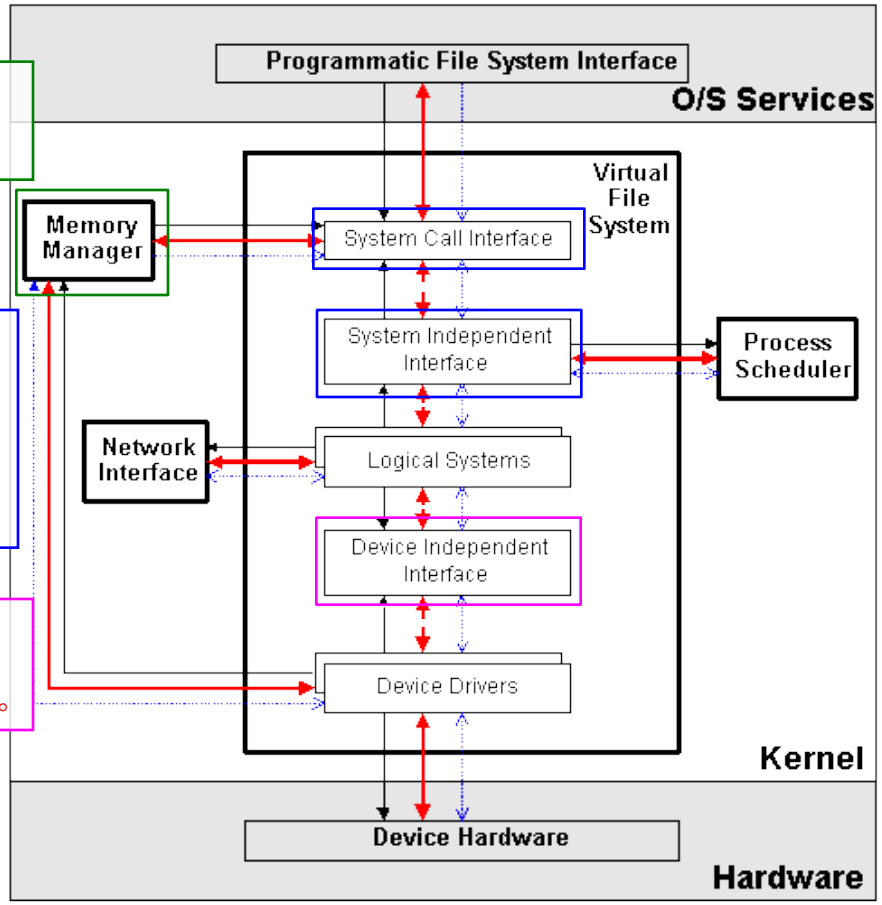


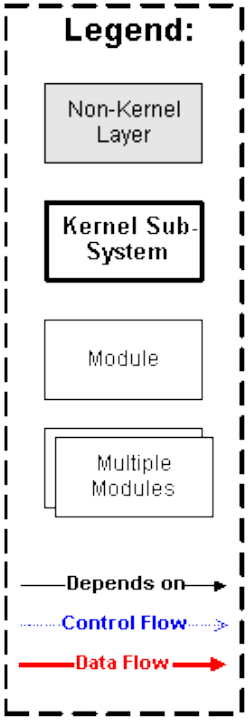
Linux Ext2 文件系统功能模块

在读写过程中，
文件内容暂存在内存。

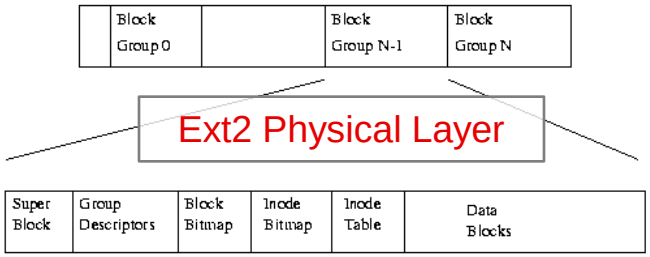
Virtual File System
提供统一的 APIs，
不依赖于文件系统的
具体实现。

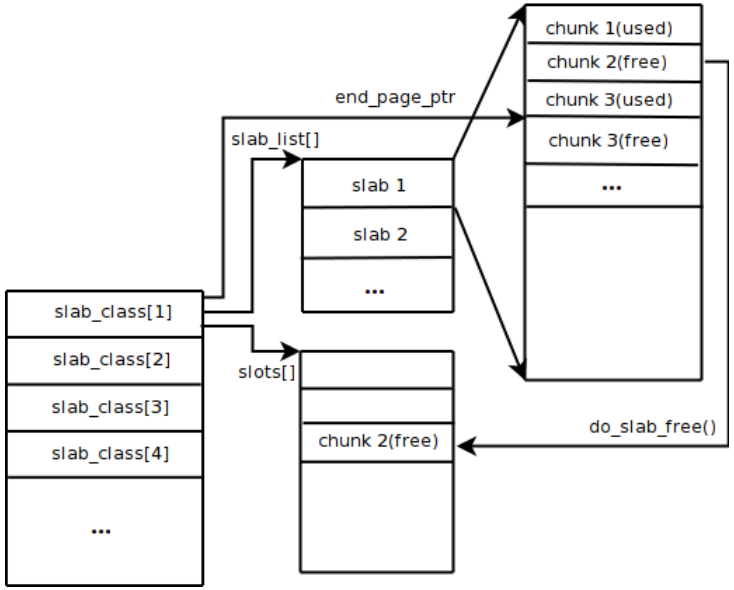
统一的驱动接口，
不依赖于具体的设备。



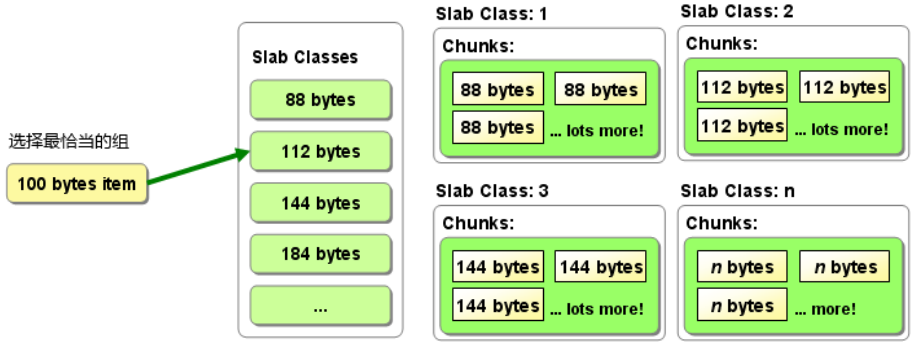


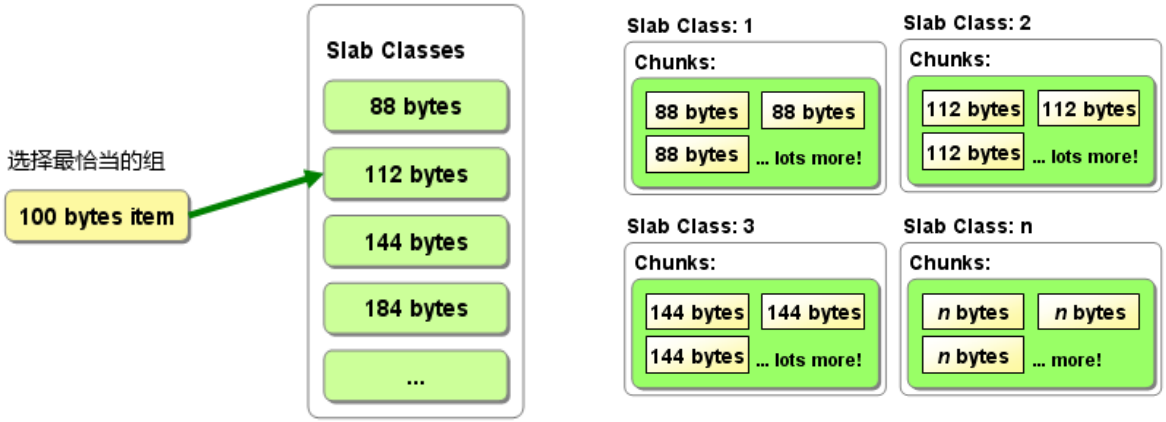
控制信号流
与数据流分离。

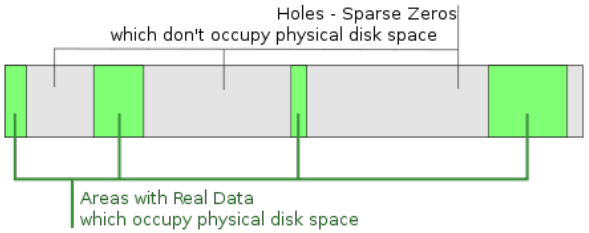




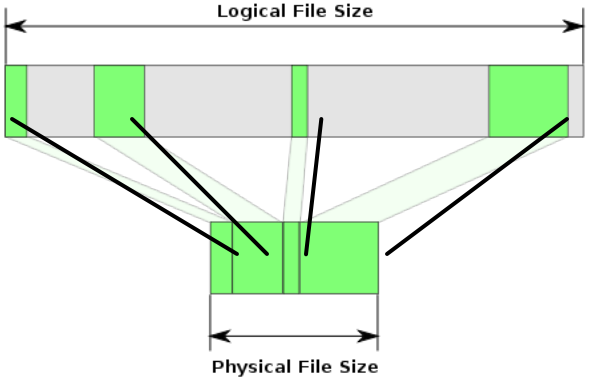
- 通常硬盘空间被划分为若干组，每一组又细分为若干块。
- 每一组中的所有块，尺寸都相同。
  但是不同组，下属的块的尺寸可以不同。
- 在写入某尺寸的数据前，
  先找到可写空间最适宜的块，所谓最适宜，
  通常指可写空间比写入内容的尺寸相同或略大。
- 把硬盘划分为组与块的目的，
  为了便于增删改，便于重复使用。
  如同停车场划分停车位一样。
- 当删除文件时，需要记住被释放的块。
- 问题：1. 有空间被浪费。
  2. 文件内容不一定被存储在连续空间，降低读写速度。







- 往现有文件里，写入更多内容时，
  硬盘空间的分配，是根据内容占用的空间而定，
  所以，无法保证整个文件被存放在连续的硬盘空间。
- 频繁增删改文件内容，会导致文件的内容，
  被放置在离散的硬盘空间里，这就是 Fragmentation 问题。
- Fragmentation 导致硬盘读写效率下降，
  因为在不连续的硬盘区域读写，需要磁头切换硬盘磁道。
- 解决办法：隔一段时间，进行硬盘清理，合并离散存储空间。



- 问题：硬盘“写”操作速度较慢。
  尤其是在指定位置的“插入”（Random Access），速度最慢。
  而在文件末尾，追加内容（Append）， 速度比插入快得多。
- 解决办法：先写入内存，然后写入硬盘。
  写入硬盘时，先在日记中，记录操作步骤（Append Log），以后再修改文件内容。

Linux Ext3 = Linux Ext2 + Journaling File System

Time: T1

Commit Time

Committed File							Log (Journal)			
Position	#1	#2	#3	#4	#5	#6	@1	@2	@3	@4
Data	10	20	30	40	50	-	Add 2 to #2	Append 90	Del #4	Minus 2 to #2

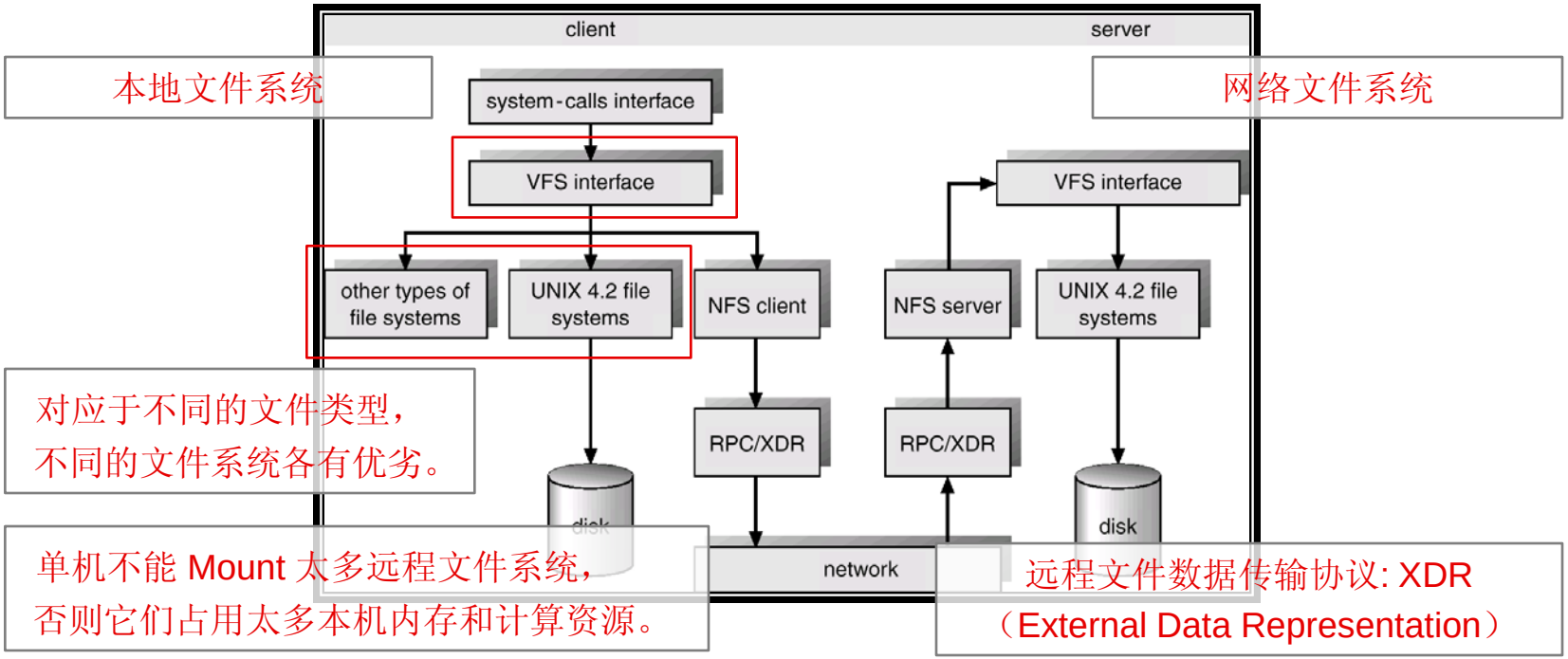
Time: T2

Committed File							Log (Journal)			
Position	#1	#2	#3	#4	#5	#6	@4	@5	@6	@7
Data	10	22	30	-	50	90	Minus 2 to #2	Add 2 to #5	Set #2 80	

Data(#2) = ? Data(#2) = File(#2) + Log(@1) + Log(@2) + Log(@3) = 80

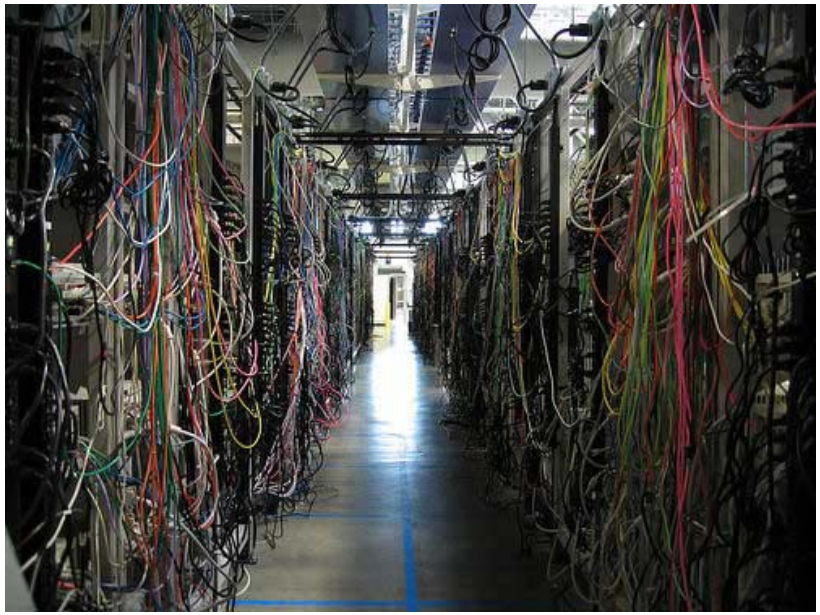
- 问题：单台计算机的硬盘空间，可能不够用，
- 解决办法：把其它机器上的文件系统，mount 到本机上，让用户以为都是本地文件系统。
- VFS：让不同来源不同类型的文件系统，保持统一的 APIs。

Distributed Linux Ext3 with NFS



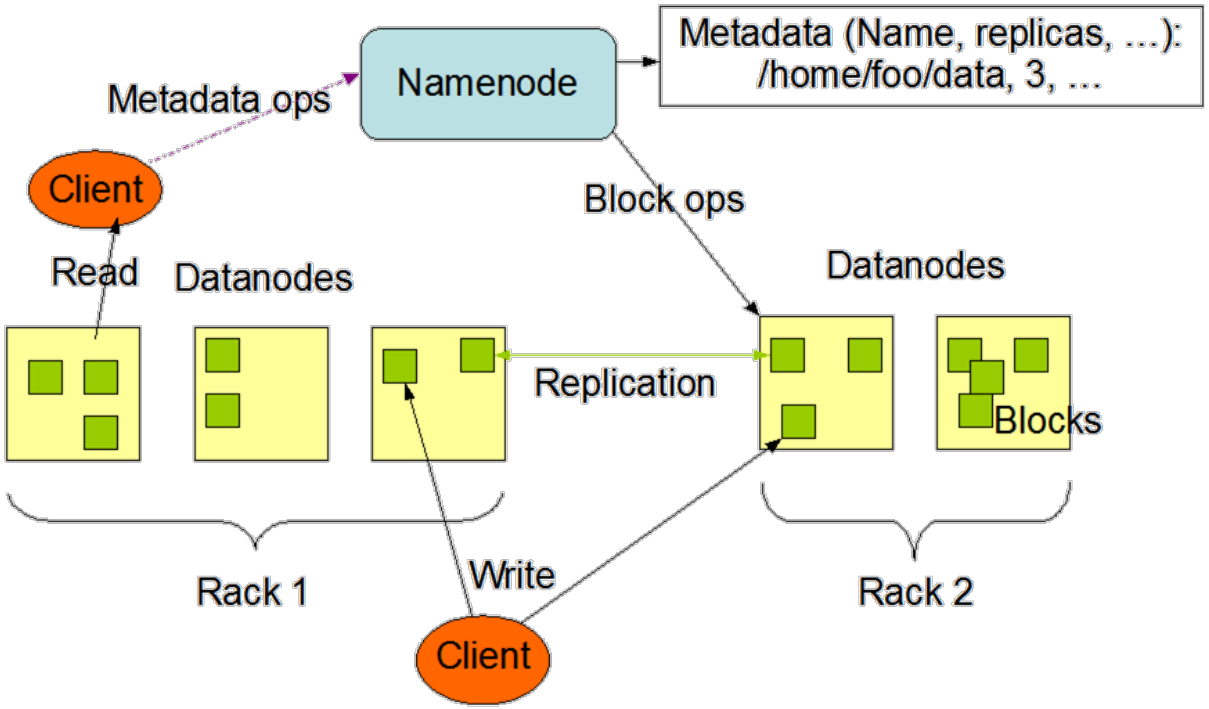
- 文件系统的三个主要功能：目录树，元数据，文件内容。
- Linux Ext2 的数据结构，用于存储以上三种信息。
- Linux Ext2 的功能模块，两层抽象。
  System Interface，兼容不同的文件系统实现。Device Interface，兼容不同的存储设备。
- Linux Ext2 在硬盘上的存储单元，是定长的，有利于增删改，有利于重复利用。
- 问题一，定长的存储单元，导致 Fragmentation，文件碎片化。
  解决办法：定期 Merge 文件，去除碎片。
- 问题二，“写”速度太慢。
  解决办法：Append Log, then Merge (Journaling)。Linux Ext3 = Ext2 + Journaling
- 问题三，磁盘空间太小。
  解决办法：Mount 网络文件系统。例如 NFS。
- 问题四，目录树占用太多空间。
  解决办法：把目录树外移到专门的服务器上，即，完全的分布式文件系统。

分布式文件系统面临的问题

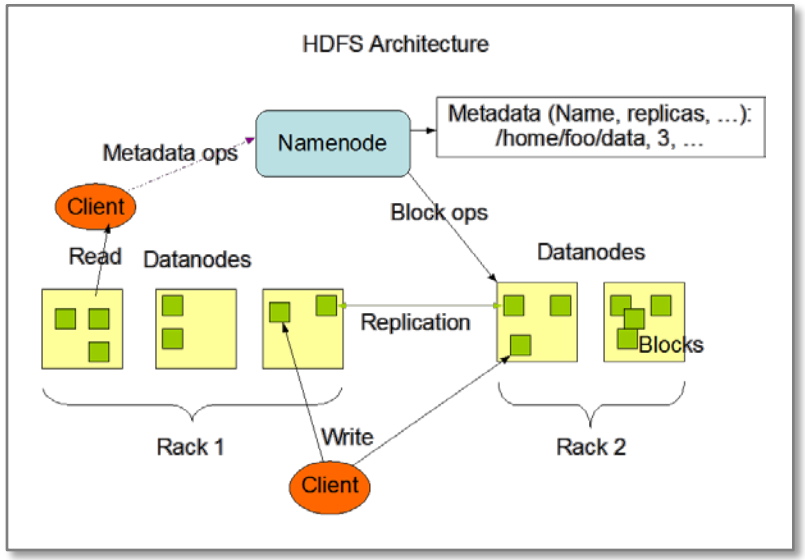


- HDFS 分布式文件系统，
  每个文件被分拆成若干块，每块尺寸均等，例如 256MB。
  这些块被存放到不同的存储服务器上。
- 可靠性，
  每一块，被复制成多个备份 (Replica).
- Master-Slave 体系结构.
  Master: Namenode，
  存放目录树等等。
  Slave: Datanode.
  存放内容数据。

Hadoop HDFS 体系结构



- Google File System 与 HDFS 非常相似。
- 一个主力 Master 服务器，配几个备份服务器。
  多个存储服务器 chunk servers，
  存放着定长尺寸的数据块。
- Chunk Servers 与主力服务器之间，
  定期发送心跳消息 HeartBeat message，
  通报彼此的健康情况。



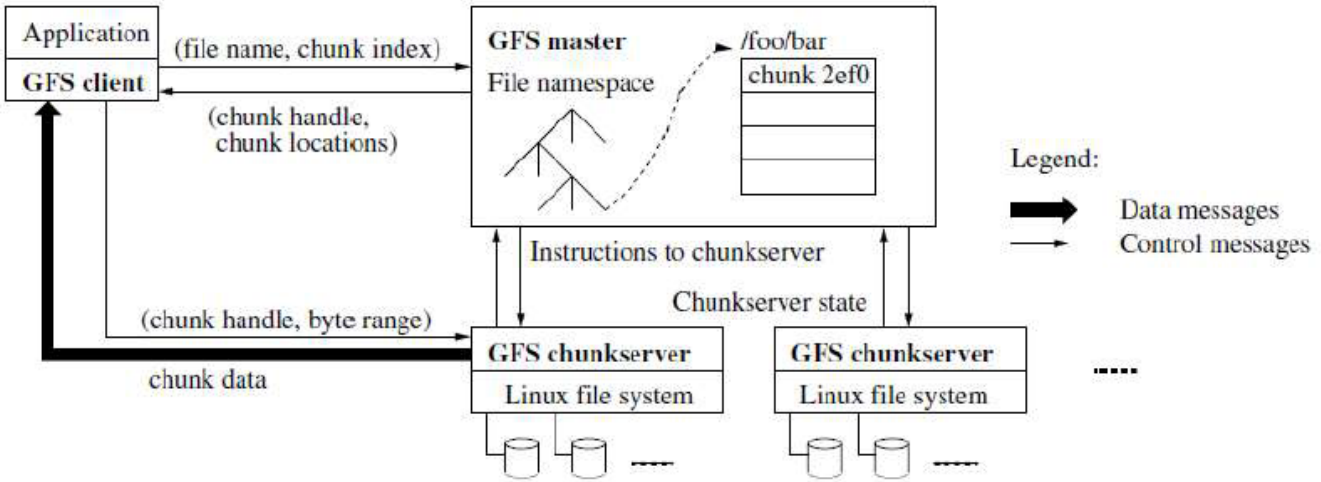
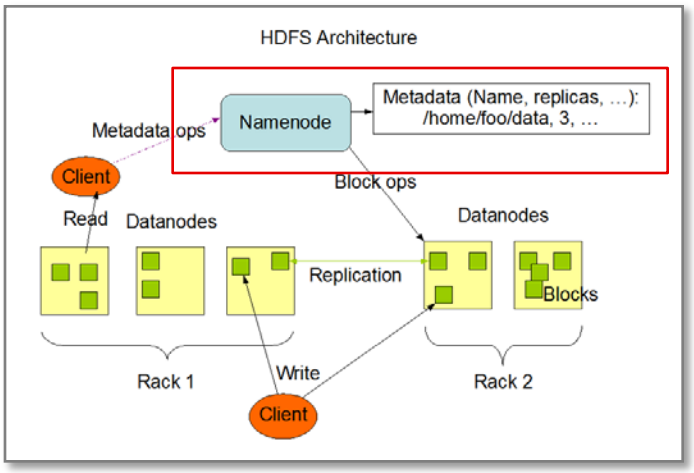
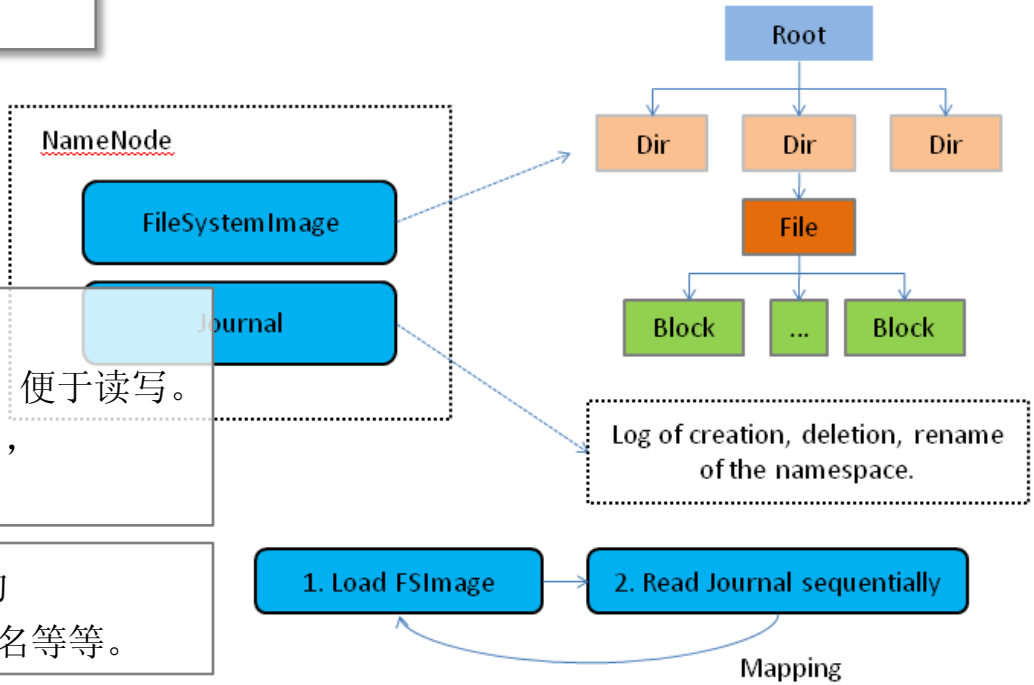


Figure 1: GFS Architecture



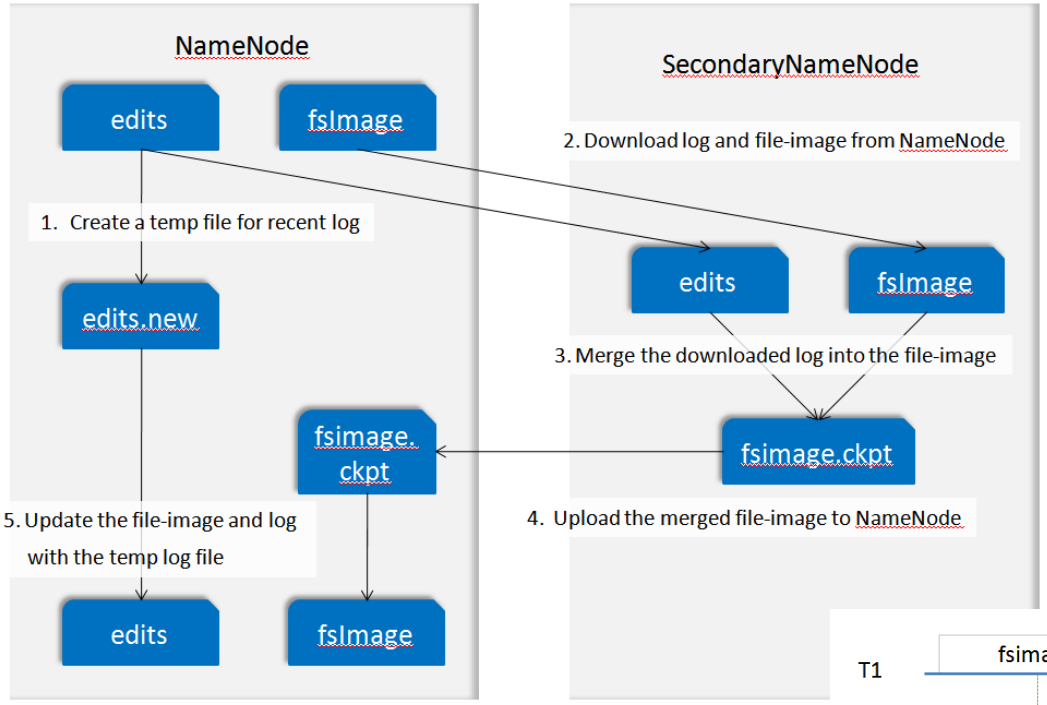
Namenode 负责文件系统目录树管理，
同时负责文件的开启、关闭，
以及文件和文件夹的命名。

Namenode 还负责监管数据块的
创建、删除、和复制。



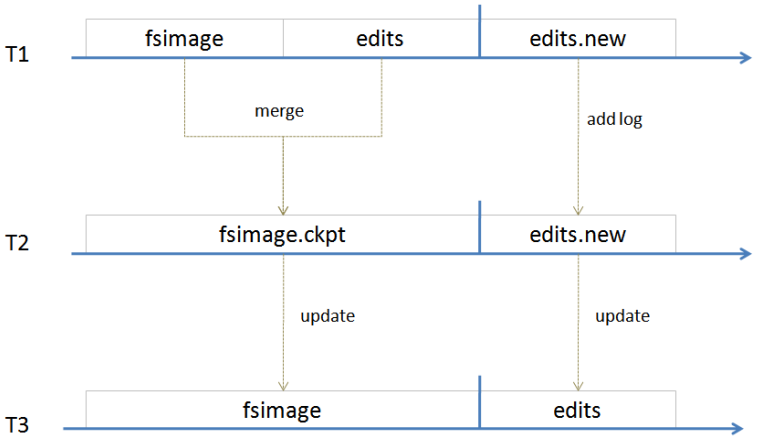
Namespace 所在服务器，
把文件系统目录树缓存在内存中，便于读写。
同时永久记录在 FSImage 文件中，
并存放在硬盘中。

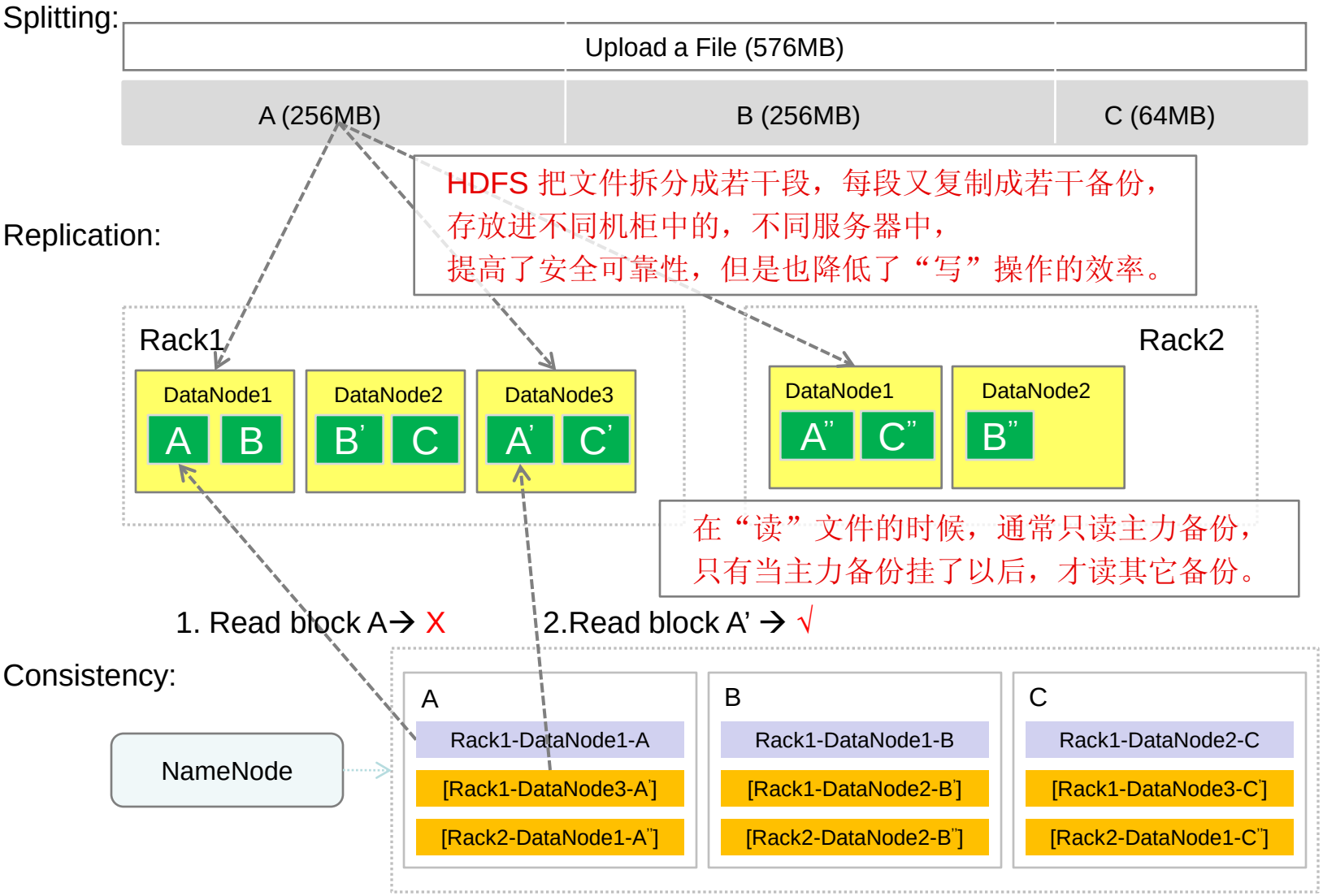
Journal 日志记录文件和文件夹的
所有操作，包括开启、关闭、命名等等。



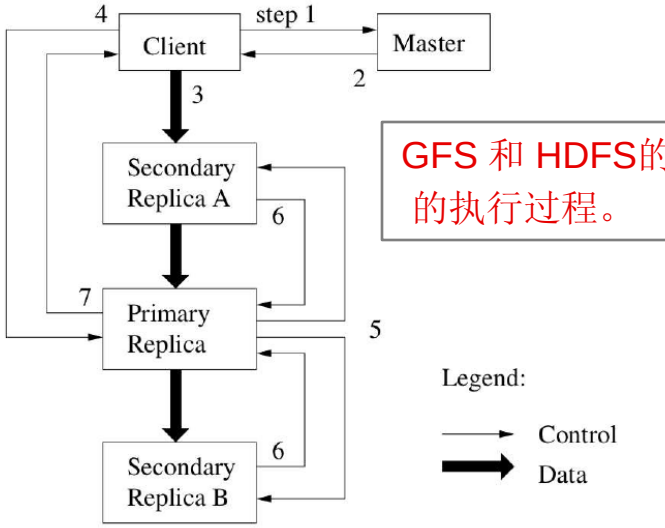
SecondaryNameNode
是另一台服务器，它负责合并文件目录树 FSImage 和操作日志 Edits。

用户对于文件和文件夹的增删改等等操作，
先写进日志 (edits)，
然后在合并进目录树文件(fsImage)。
这样有利于缩短对用户操作的响应时间。



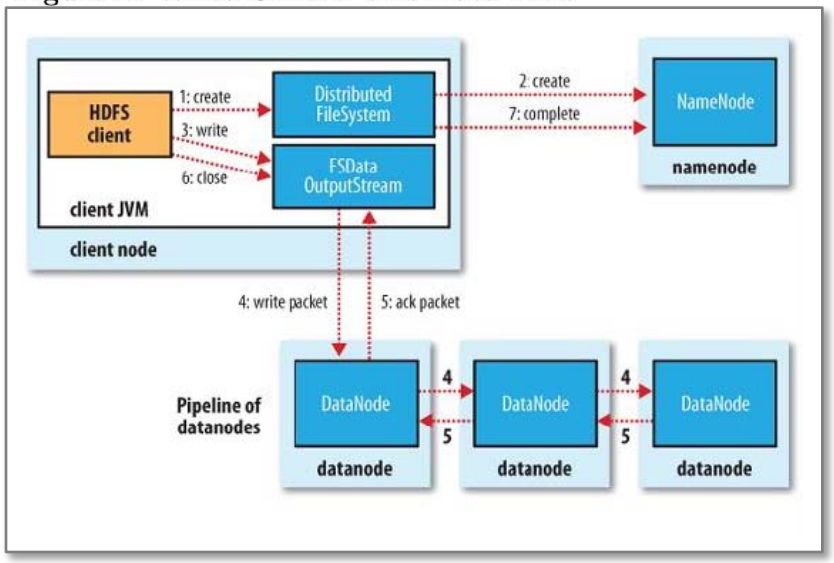

Splitting:
Upload a File (576MB)
A (256MB)
B (256MB)
C (64MB)
Replication:
HDFS 把文件拆分成若干段，每段又复制成若干备份，
存放进不同机柜中的，不同服务器中，
提高了安全可靠性，但是也降低了“写”操作的效率。
Rack1
Rack2
DataNode1
A
B
DataNode2
B'
C
DataNode3
A'
C'
DataNode1
A''
C''
DataNode2
B''
在“读”文件的时候，通常只读主力备份，
只有当主力备份挂了以后，才读其它备份。
1. Read block A→ X
2.Read block A' → √
Consistency:
NameNode
A
Rack1-DataNode1-A
[Rack1-DataNode3-A']
[Rack2-DataNode1-A'']
B
Rack1-DataNode1-B
[Rack1-DataNode2-B']
[Rack2-DataNode2-B'']
C
Rack1-DataNode2-C
[Rack1-DataNode3-C']
[Rack2-DataNode1-C'']

- 1,2: 用户向 Master 询问
  存放文件及备份的所有服务器的地址。
- 3: 用户把文件内容数据，
  发送到最近的存储服务器的内存里。
  该服务器把数据依次发送到其它服务器中。
- 4: 用户向主力服务器，
  发送“写”请求。
- 5: 主力服务器决定在各个服务器硬盘中，
  文件数据的存放地点，
  并指令各个服务器把数据从内存写入硬盘。
- 6: 各个服务器向主力服务器
  汇报“写”操作完成情况。
- 7: 主力服务器回复用户。



GFS 和 HDFS的“写”操作的执行过程。

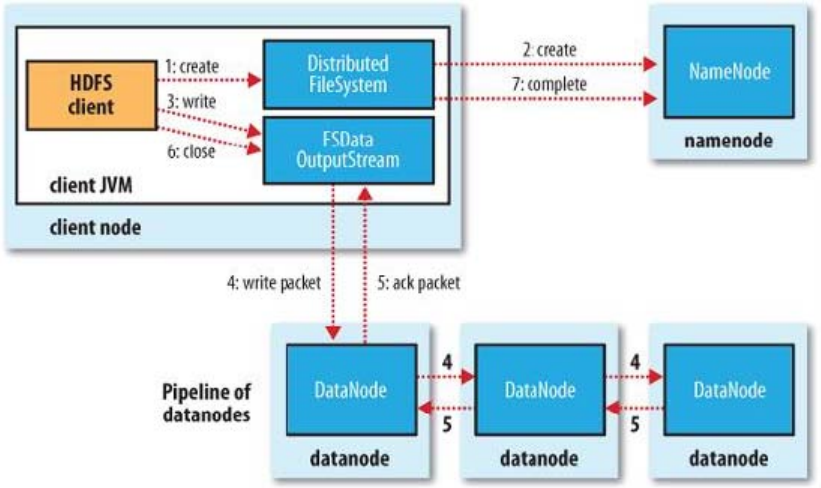
Figure 2: Write Control and Data Flow



HDFS 创建文件过程

DistributedFileSystem 咨询 NameNode 后，
生成 FSDataOutputStream 数据通道。

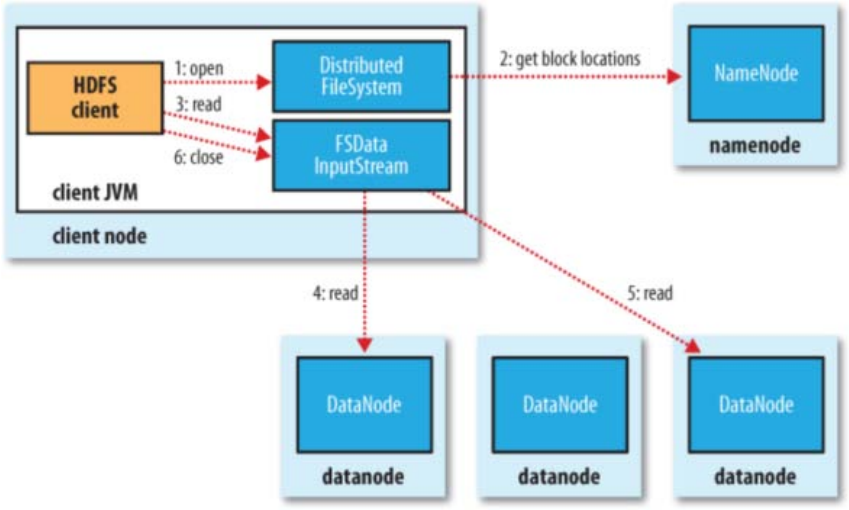
上传文件被拆分成若干段，
只有当第一段被所有服务器都写入硬盘以后，
第二段数据才开始写操作。



HDFS 读文件操作过程

DistributedFileSystem 咨询 NameNode，
文件数据的存放地址，
然后生成 FSDataInputStream 数据通道。

FSDataInputStream 从主力服务器中，
读取文件数据。
如果主力服务器无响应，
则从其它服务器中读取文件数据。



- 把文件系统目录树，与文件数据，分别存放在不同的服务器上。
  HDFS: NameNode vs. DataNode，
  GFS： Master vs. Chunk Server。

- HDFS NameNode 内存中存放着整个文件系统的目录树，同时存入硬盘中的文件 FSImage。
  文件的创建、删除、和命名等等，这些目录树的操作，
  被先记录在日志 Journal 中，然后合并入 FSImage。

- 每个文件，被拆分成若干段，每一段又别复制成若干备份，存放在不同的服务器中，提高可靠性。
  但是文件备份导致“写”操作更加复杂，更加耗时。

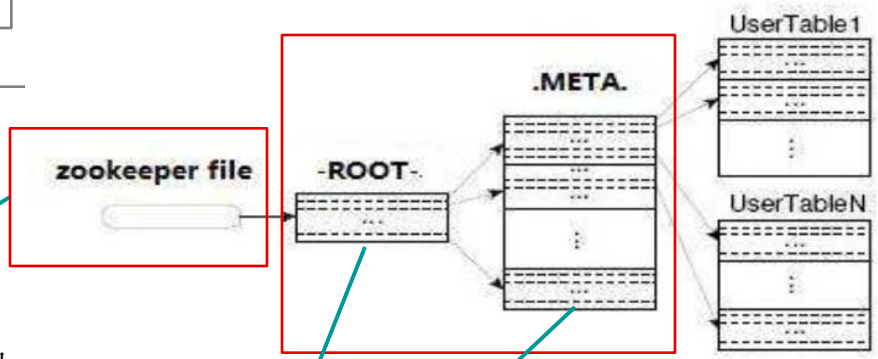
- 问题一：整个文件系统的目录树，不可过大，否则无法存放在 NameNode、Master 的内存中。
- 问题二：适用于大文件，“写”方面，经常 append，但是很少 random access modify。
  “读”方面，经常从头到尾 sequential read，但是很少 random read。
- 解决办法：目录树存放在多个 NameNode 服务器上，细分目录树的颗粒度。
  或者，完全绕开目录树！

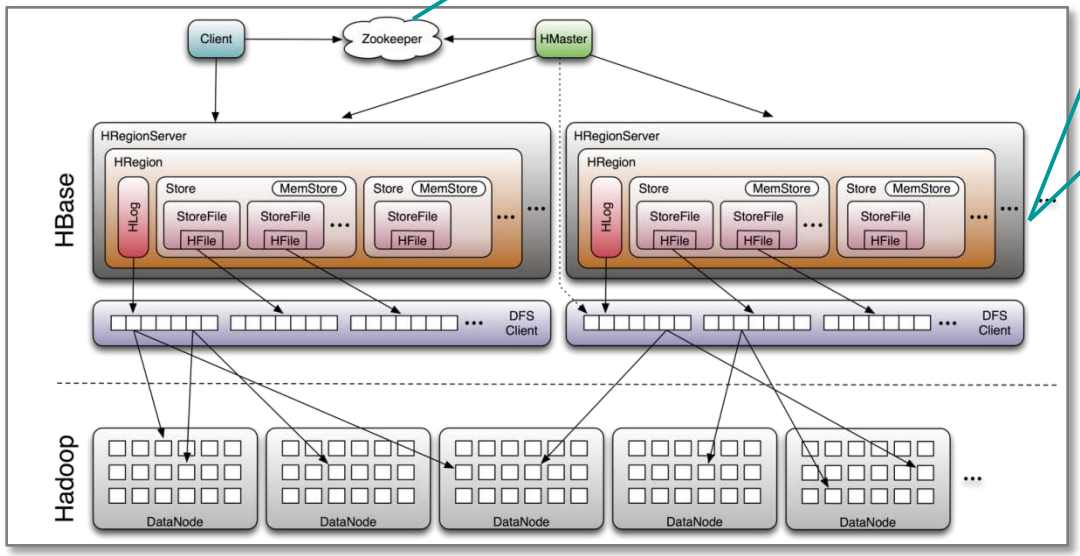
分布式文件系统问题
可能的解决办法

Zookeeper:
保存目录树的入口地址 -ROOT- 。

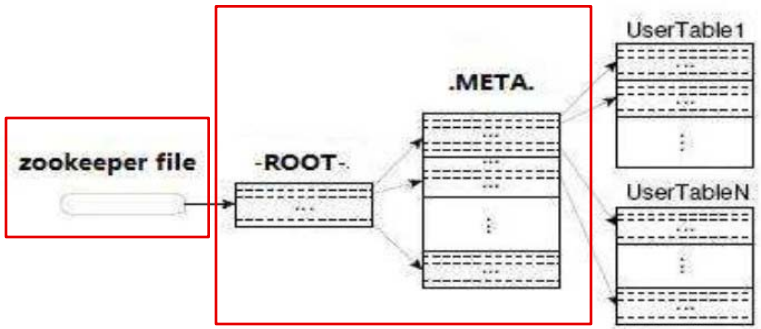
-ROOT- 和 .META.

- 整个分布式文件系统的目录树，被拆分分成两级，每级若干段。
- 每一段都是一个独立文件，分别存放在某个 RegionServer 的内存和硬盘中。





Hadoop HBase 体系结构

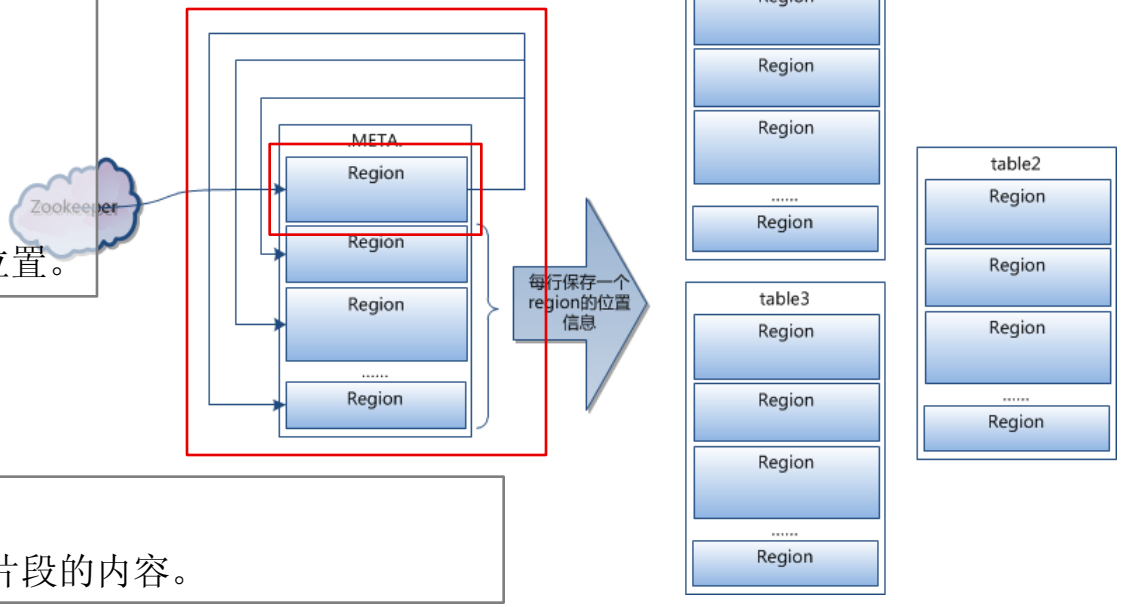


借用 **Hadoop HBase** 体系
解决分布式目录树数据结构

Zookeeper
保存目录树的入口地址 -ROOT- 。

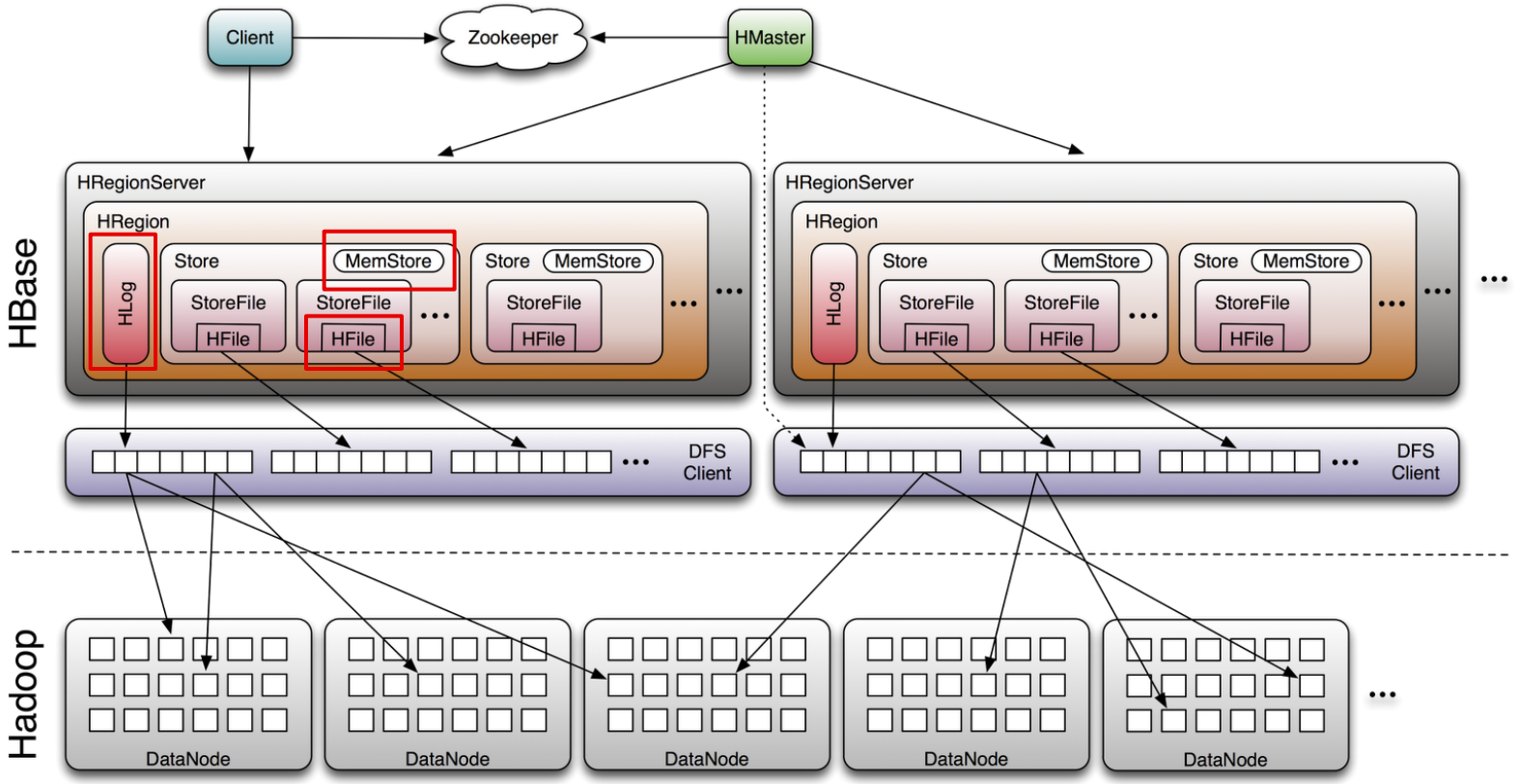
-ROOT- 和 .META.
-ROOT- 与其它目录树数据，
都存放在 .META. 中。

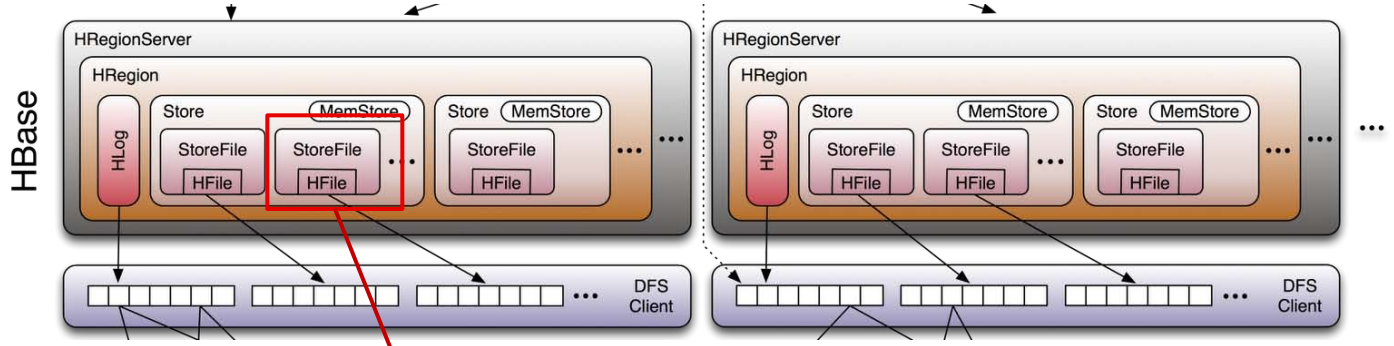
.META. 中第一个 Region，
存放 -ROOT- 内容，
记录着其它 .META. Regions 的位置。



.META. Region:
记录整个文件系统目录树的一个片段的内容。

- 无论是目录树，还是内容文件，都被无差别地当成文件来处理。
- HRegionServer 负责所有文件的读写操作，而内容数据最终被写入 HDFS DataNode 中。
- 为了提高读写效率，文件内容被缓存在内存中的 MemStore，操作步骤被记录在 HLog 中。
- 满足预定触发条件时，MemStore 内存中的数据，被写入硬盘文件 HFile，并与 HLog 合并。



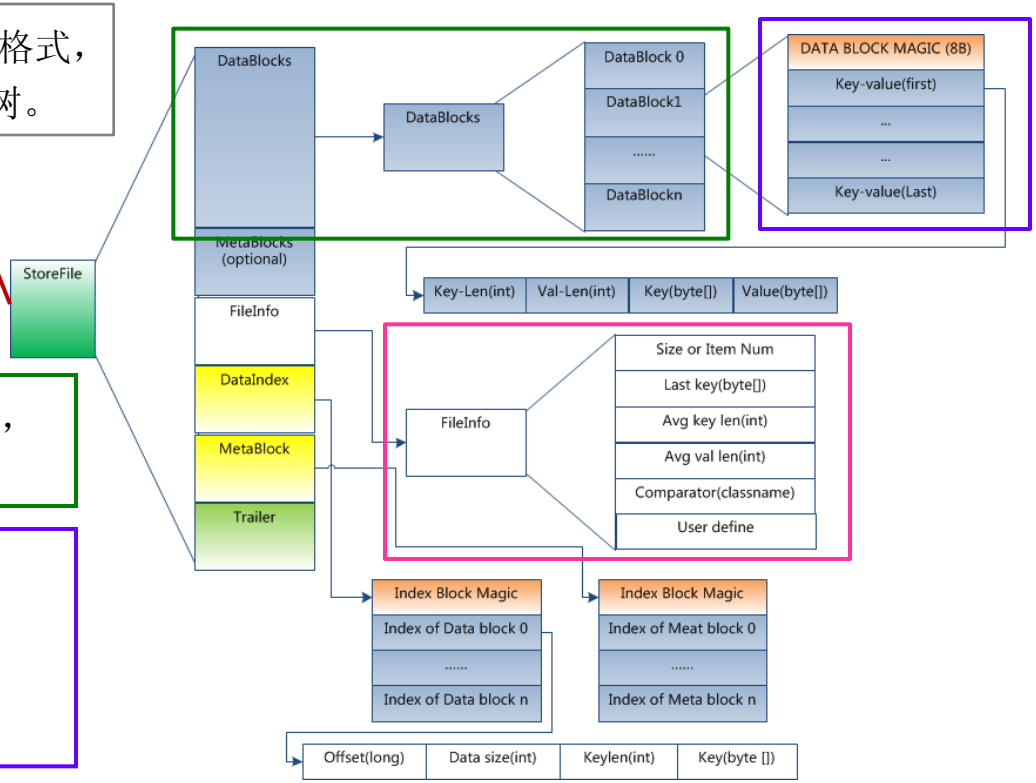


StoreFile 数据格式，可用做通用的文件格式，
即可用于存储文件内容，也可用于目录树。

文件属性等元数据，
也被存放在 StoreFile 中。

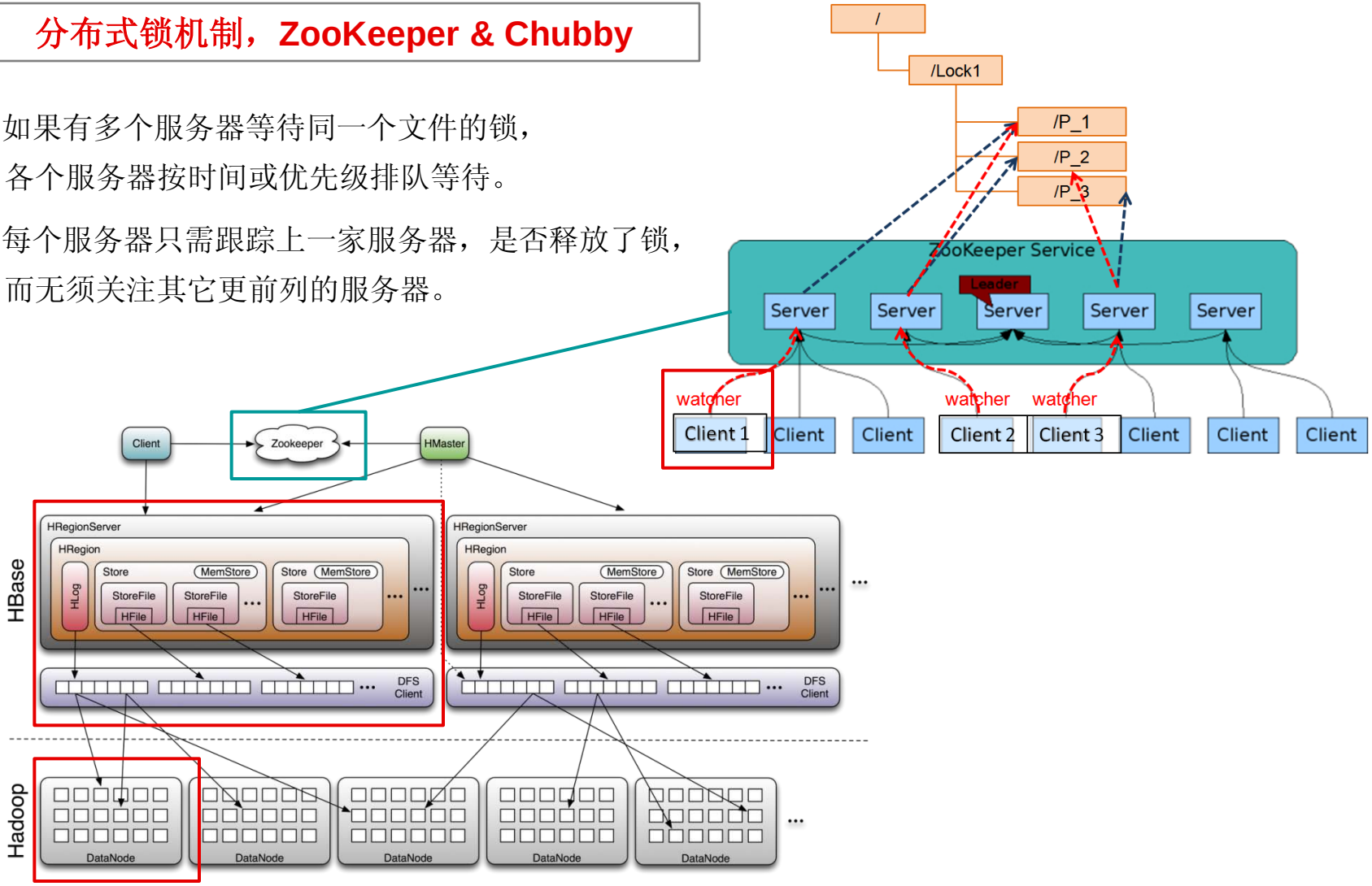
用硬盘空间被分割成等长的 DataBlocks，
便于增删改，重复使用。

必要时，可以进一步细分 DataBlocks，
提供颗粒度更细的内容存储，
例如 Key-Value 对偶，
或者多媒体文件内部的各个数据段。



分布式锁机制，ZooKeeper & Chubby

- 如果有多个服务器等待同一个文件的锁，各个服务器按时间或优先级排队等待。
- 每个服务器只需跟踪上一家服务器，是否释放了锁，而无须关注其它更前列的服务器。

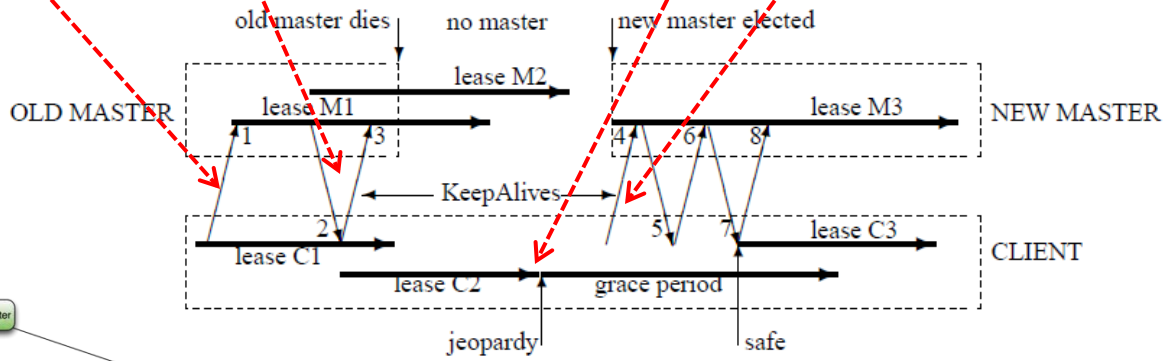


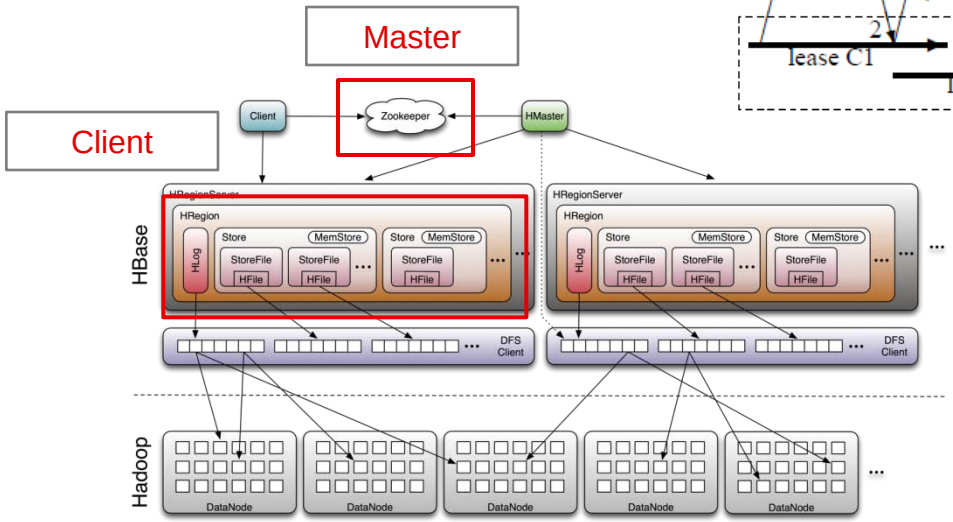
用户申请锁。

负责该文件的锁的 ZooKeeper 服务器，
核实并发放使用许可证，
用户收到许可证后，回复。

当旧的 ZooKeeper 服务器挂掉以后，
用户无法验证锁使用许可证是否仍然有效，
用户向 ZooKeeper 发送危险信号 “jeopardy”。

发现新的负责该锁的 ZooKeeper 服务器，
并续约锁使用许可证。





- 把整个分布式文件系统的目录树，拆分分成两级，每级若干段。
  每一段都作为一个独立文件，分别存放在一个服务器的内存和硬盘中。

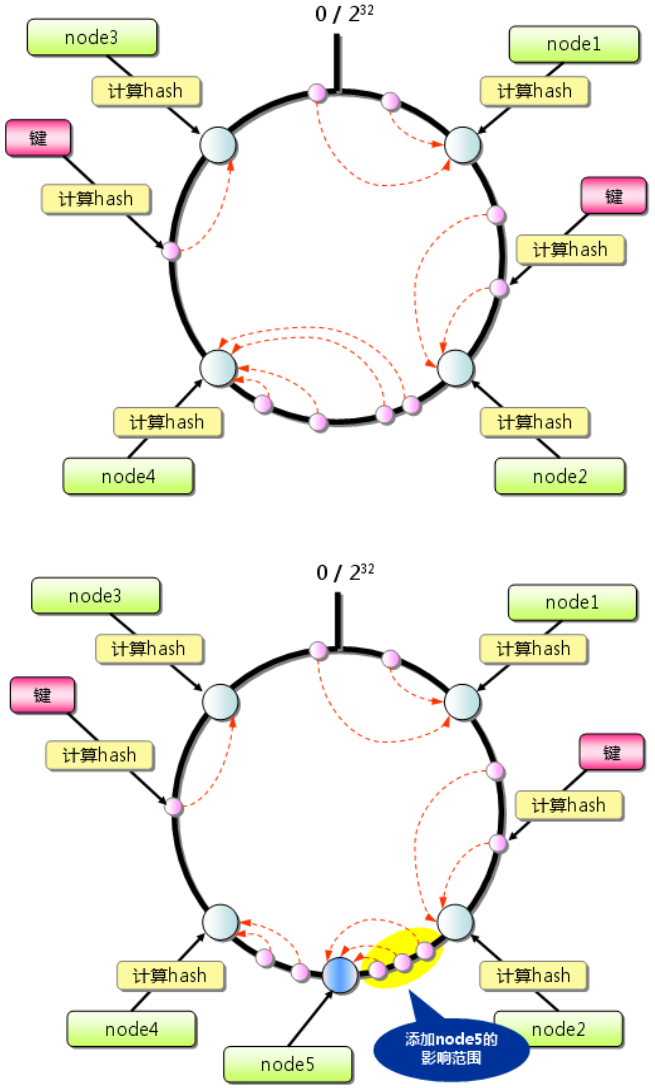
- 使用统一的文件格式，既可用于存放目录树，也可存放普通的文件内容。
  文件被拆分为若干段，分别存入等长的数据块中。
  数据块可以再度被细分，用于存放颗粒度更细的文件数据段。

- ZooKeeper 使用队列来安排锁的使用，每位申请使用锁的服务器，只需要跟踪前一家是否释放了锁。
  当维护锁队列的 ZooKeeper 服务器挂掉以后，
  使用锁的服务器，向 ZooKeeper 总管汇报，并使用新的锁服务器。

- 能否彻底取消目录树？
  用算法取代目录树，确定某一文件存放在哪些服务器上。
- 如何解决云存储中的存储碎片问题？
  把数据按冷热程度，分成几代 (generation)，新生代预留的修改空间，比中古代、远古代的大。
  需要建设完善的 Bookkeeping 系统。
- 能否文件系统与数据库合二为一？
  文件夹作为文件的关键词标签的一种。
  把目录树树形结构，改变成表结构。
- 基于搜索的数据库索引？
  取消 B-Tree，代之以 Inverted Index。
- 面向数据挖掘的云存储系统？
  扩大 Inverted Index，使之能够方便柔性地插入解析过的 Log，以及 Version Control。

有没有可能彻底回避目录树？

- 借用 Consistent-Hash 算法，
  用算法确定文件存放位置，取代目录树。
- 把所有服务器的 IP 地址，根据哈希算法，
  映射到有 2^32 个节点的环上去。
- 每当创建一个文件时，或者读写既有文件时，
  根据预定算法，把文件名映射到环上某节点。
- 从这个文件的节点出发，
  顺时针寻找，离它最近的服务器的节点，
  把该文件存入这个服务器。
- 当读写文件时，根据相同的映射算法，
  找到存放该文件的服务器。
- 当增加或删减服务器时，
  只有少数文件需要搬家，
  以维护映射算法的有效性。



第四届中国云计算大会

THANKS