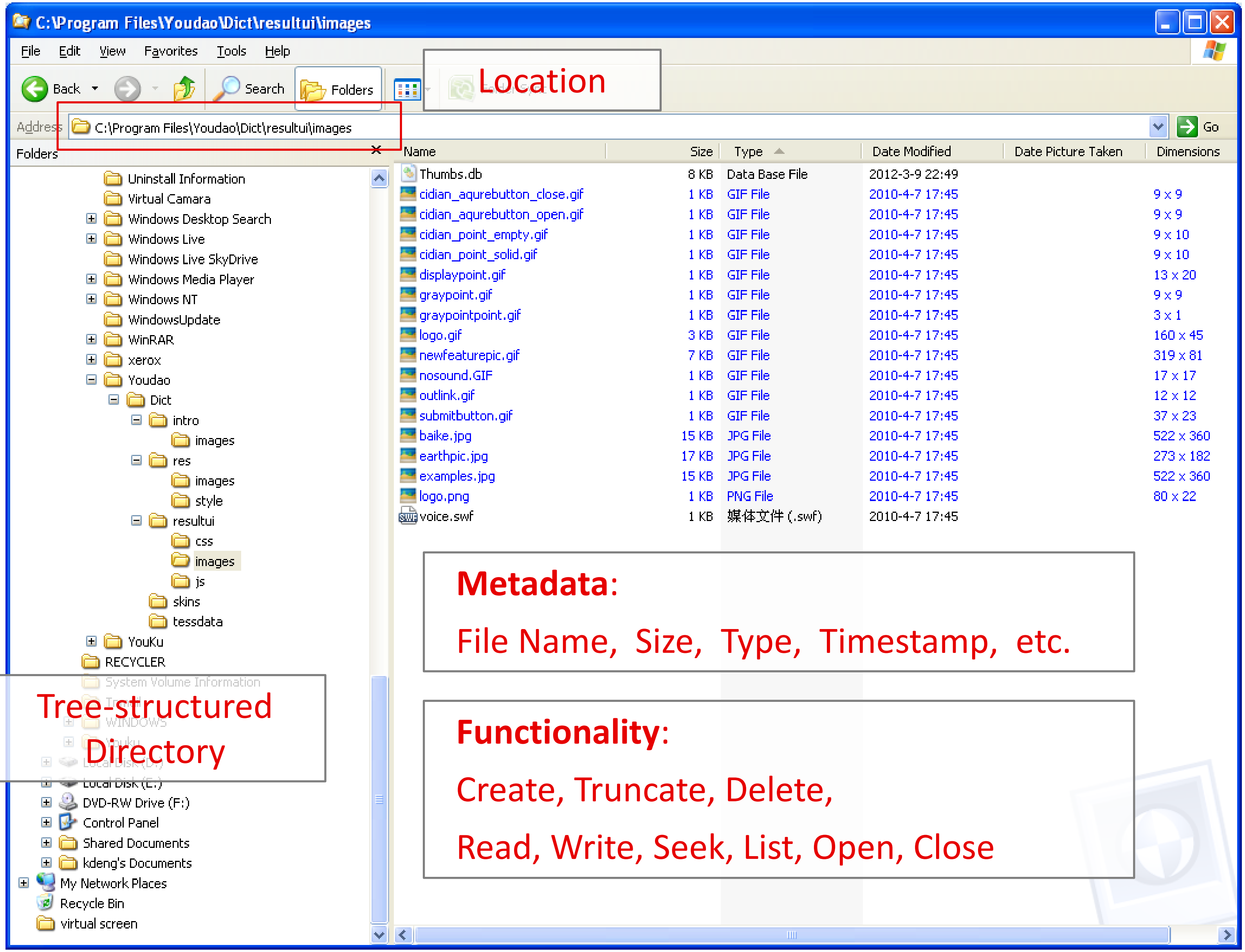
Google File System

@ 北京航天航空大学软件学院

18:00 03/13/2012 周二晚 北航主校区 主M201

By 邓侃 Ph.D, 朱小杰, 李小吉
SmartClouder.com

File System Recapitulation



Metadata:

File Name, Size, Type, Timestamp, etc.

Functionality:

Create, Truncate, Delete,

Read, Write, Seek, List, Open, Close

Directory ID	Subdir IDs & File IDs
1	21， 22，
21	31， 32， 33， …
22	34， 35， …

Node ID	Metadata	Storage Chunk IDs
22	theNameOfADir|20120306|24(Number of subdirs and files）	NIL
32	theNameOfAFile|20120306|1024 (File Size)	3A28C329, …

How to allocate storage space for directory and file?

- System architect's 3 routines:

  Define the data structure first,

  Decide the workflow,

  Design the modules and where to deploy.

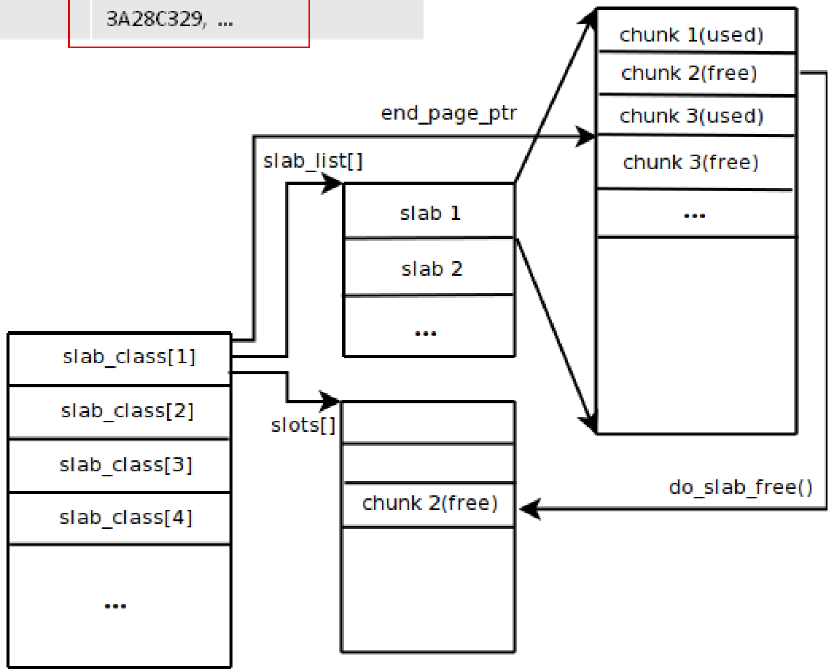
- File system's fundamental data structure:

  Tree structure of directories and files,

  Metadata, physical storage address.

A homebrew data structure of File System

Node ID	Metadata	Storage Chunk IDs
22	theNameOfADir|20120306|24(Number of subdirs and files)	NIL
32	theNameOfAFile|20120306|1024 (File Size)	3A28C329, ...



- Slab: fixed-size storage space.
- SlabClass:
  A group of slabs of the same size.
- Chunk:
  Each slab splits into many chunks,
  the chunks are of the same size.
- Slots:
  An address list pointing to the re-usable chunks.
- Why split the storage into fixed-size slabs and chunks?
  Easy to re-use, but may waste storage space.
- Before storing a data, find the slab with the appropriate size,
  equal or a little bigger than the data.

Learn from MemCached

Directory ID	Subdir IDs & File IDs
1	21，22，
21	31，32，33，…
22	34，35，…

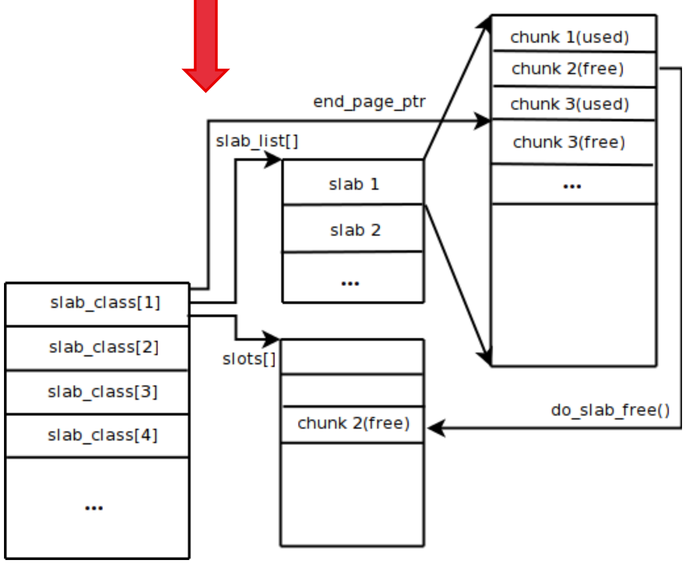
Node ID	Metadata	Storage Chunk ID
22	theNameOfADir|20120306|24(Number of subdirs and files）	NIL
32	theNameOfAFile|20120306|1024 (File Size)	3A28C329, …

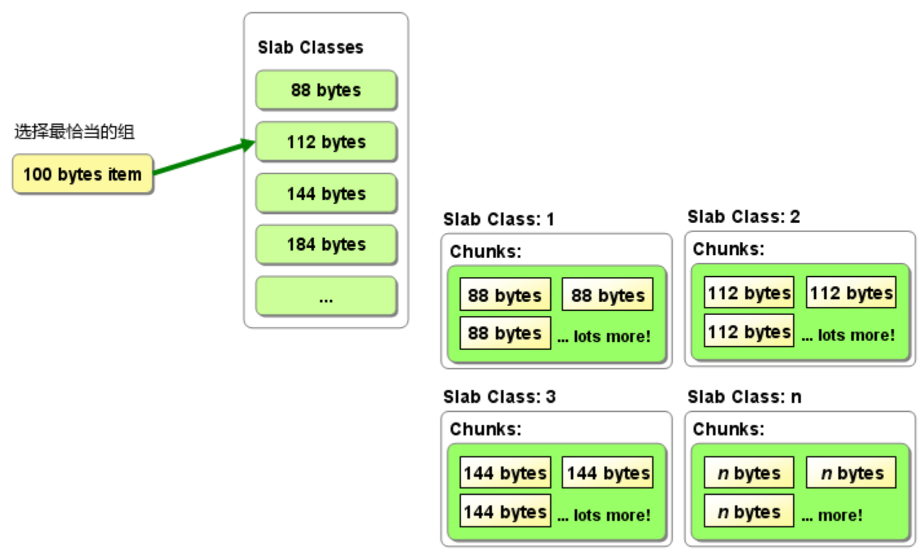
% fappend

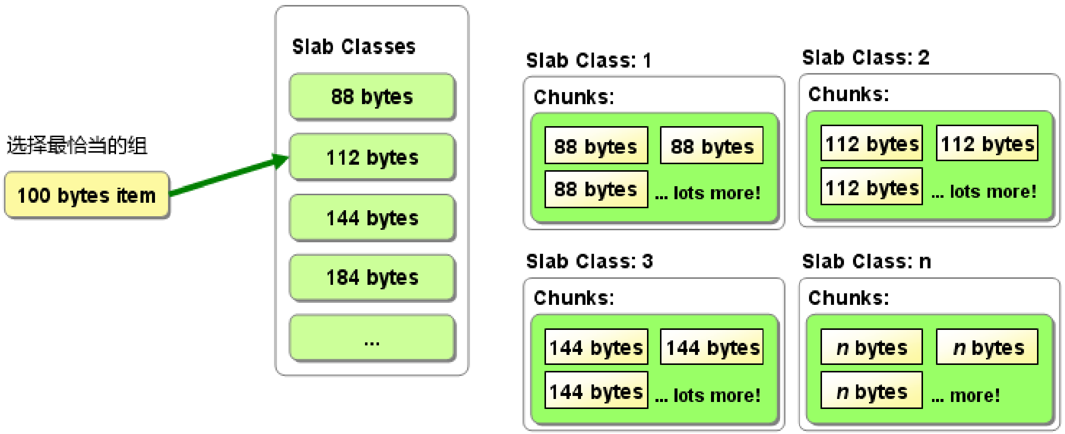
/home/user/test.txt

“content added to the tail …”

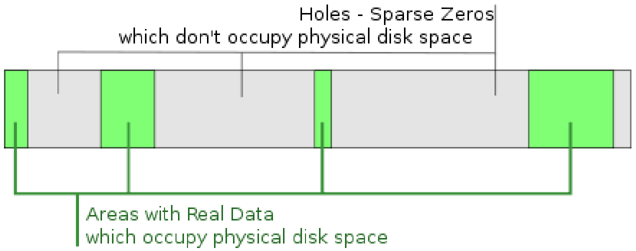
Workflow

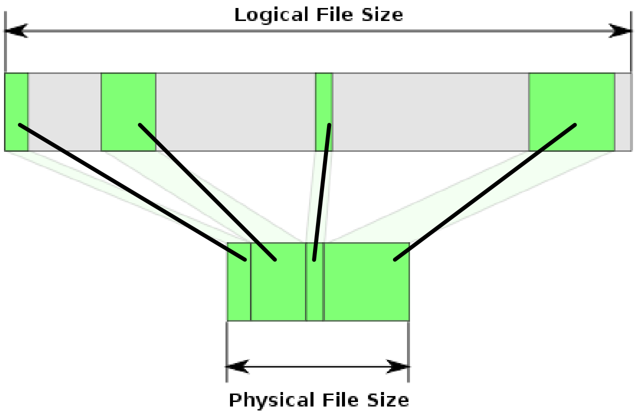




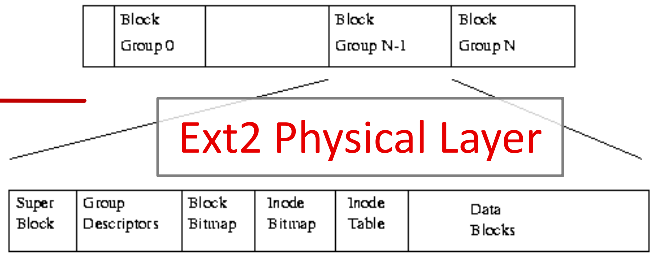


- When appending new data,
  the storage location is chosen by the size.
  No guarantee to allocate the data
  of the same file, in continuous blocks.
- Frequently modifying file, induce fragmentation.
- Fragmentation decrease disk IO efficiency,
  because disk IO involves in mechanical movement.
- From time to time, do defragmentation.
- Flash storage may not care about fragmentation.

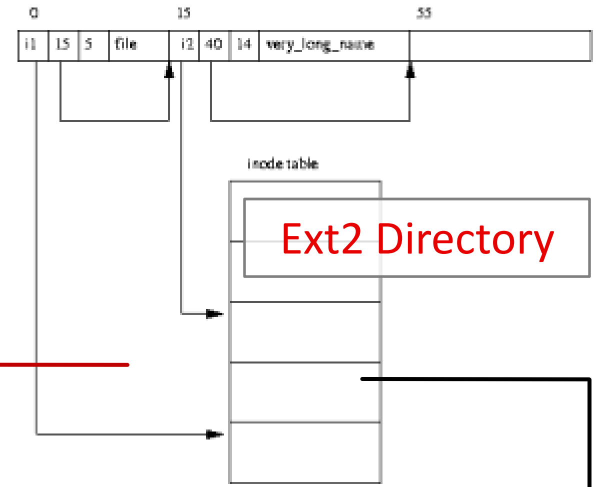




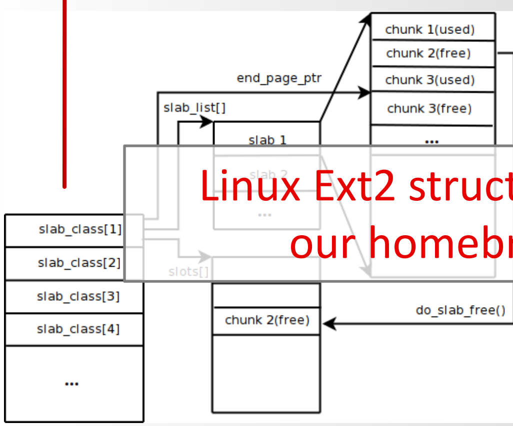
Linux Ext2 Layers



Ext2 Physical Layer



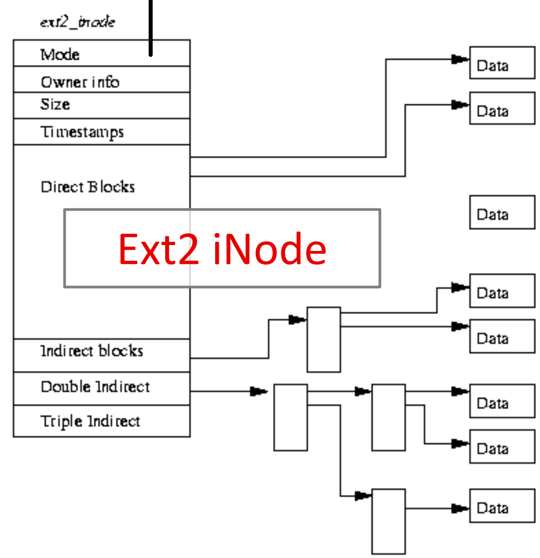
Ext2 Directory



Linux Ext2 structures are similar to our homebrew structure.

Directory ID	Subdir IDs & File IDs
1	21，22，
21	31，32，33，…
22	34，35，…

Node ID	Metadata	Storage Chunk ID
22	theNameOfADir|20120306|24(Number of subdirs and files）	NIL
32	theNameOfAFile|20120306|1024 (File Size)	3A28C329, …



Ext2 iNode

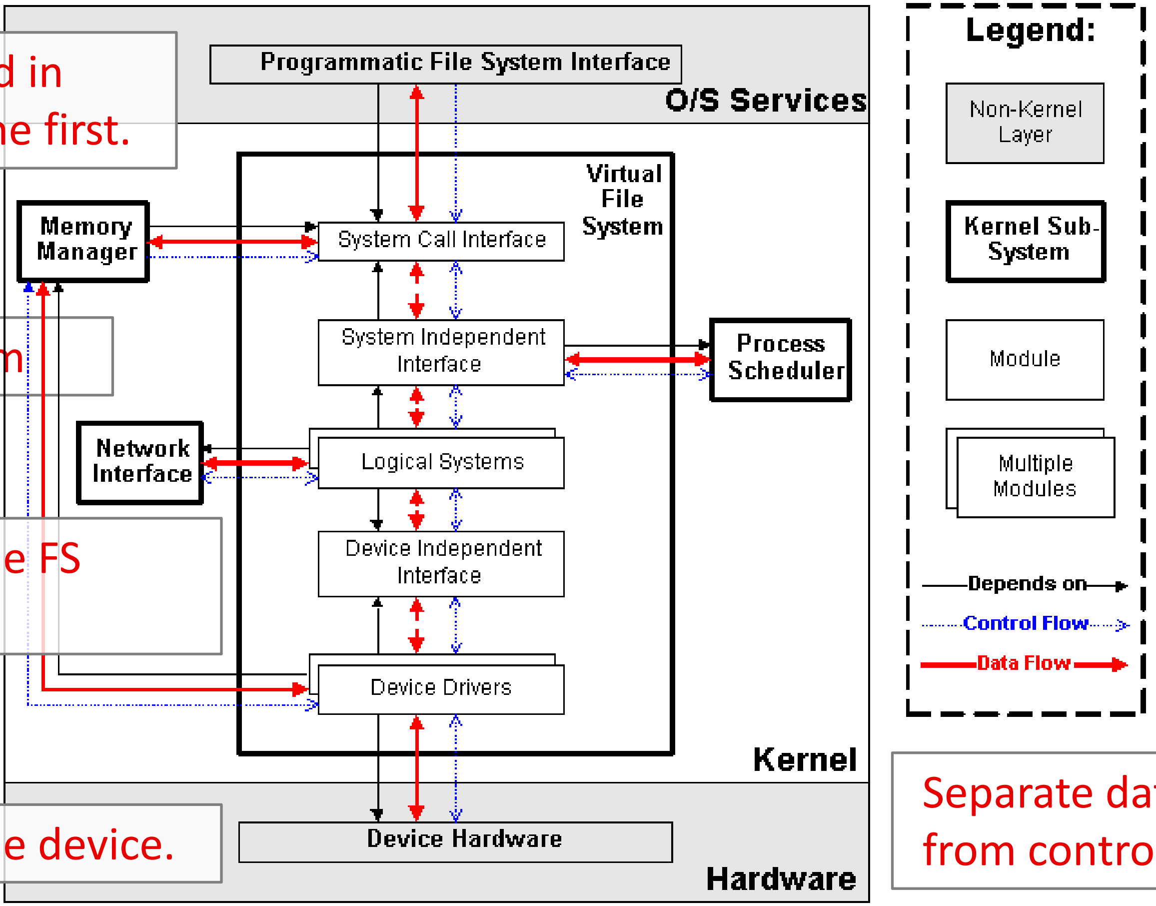
Linux Ext2 System Architecture

Data stored in buffer cache first.

Virtual File System

Transparent to the FS implementation.

Transparent to the device.



Separate data flow from control flow.

- Writing to disk is slow,
  So, appending is slow, but still much faster than random accessing.
- Store in buffer cache first, then write to disk.
  Store in buffer cache first, then write to disk as log, then merger (commit) into files.

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

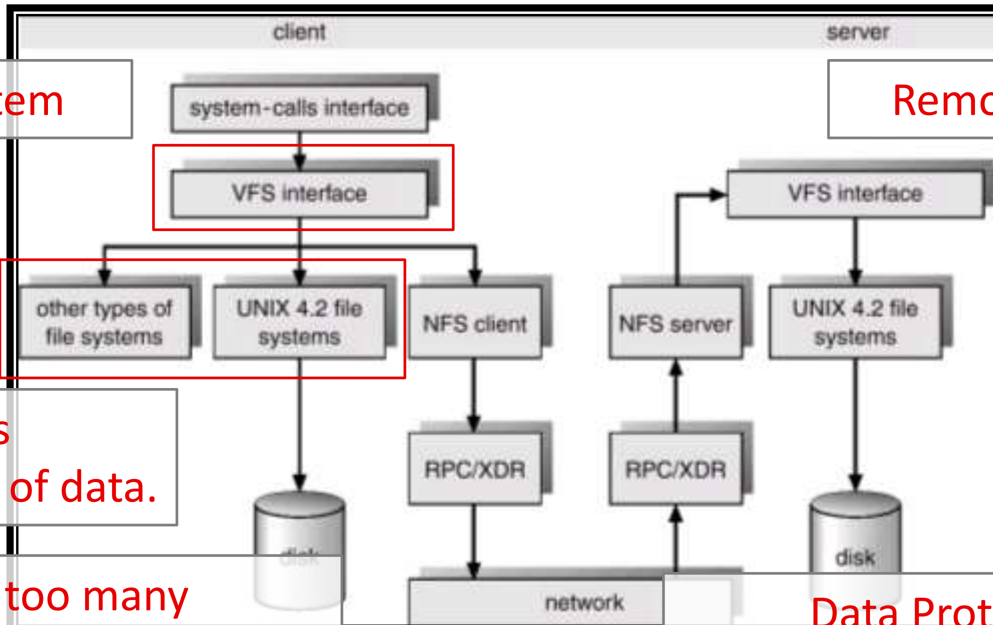
Time: T2

Committed File							Log (Journal)			
Position	#1	#2	#3	#4	#5	#6	@1	@2	@3	@4
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

- When a single machine's local disk space is not sufficient, expand the storage by mounting remote file system to the local one.
- VFS makes the File System interface consistent, the variety of the FS implementations is transparent to the client.

Distributed Linux Ext3 with NFS



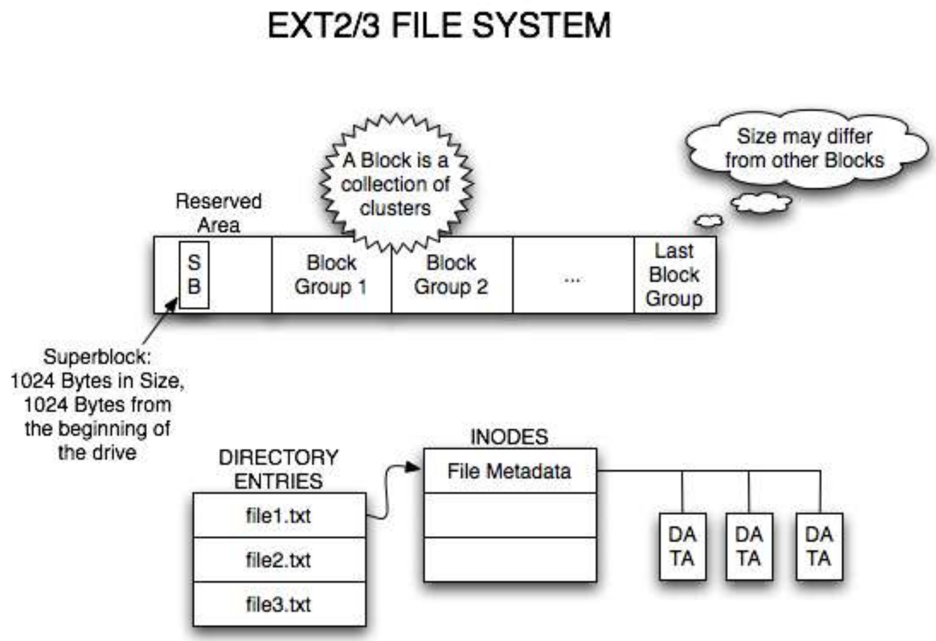
Local file system

Remote file system

Different FS fits different types of data.

Cannot mount too many remote FS's, consuming too much local resource.

Data Protocol: XDR
External Data Representation



- Data structure:
  1. tree structure of directory and file,
  2. metadata (inode).

- Physical layer:
  1. fixed-size blocks,
  2. block groups of different sizes.

- Problems to solve:

  Fragmentation,

  Disk IO is slow, but append is faster than random access.

  Local disk space is not sufficient.

- Concepts to remember:

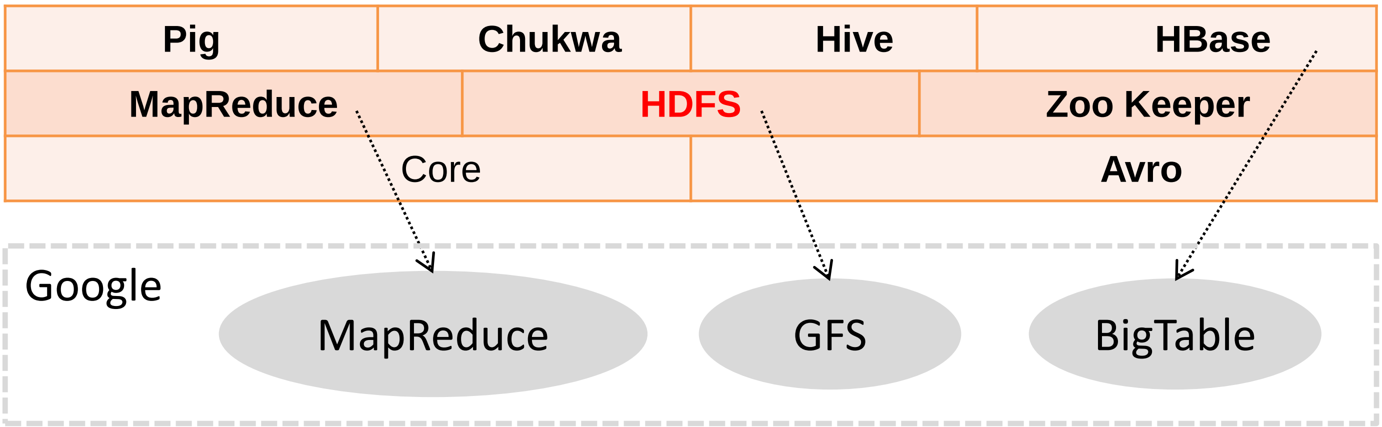
  iNode: metadata of directory and file in Unix/Linux.

  Virtual File System: an abstract layer to unify the APIs of different FS implementations.

  Journaling File System: append to log first, then merge into file.

Hadoop HDFS

- Hadoop is an open source project, supervised by Apache org. Implemented in Java.
- Hadoop is a distributed system, for large scale storage and paralleled computing. A mimic of Google system.



Hadoop Common: The common utilities that support the other Hadoop subprojects.
Avro: A data serialization system that provides dynamic integration with scripting languages.
Chukwa: A data collection system for managing large distributed systems.
HBase: A scalable, distributed database that supports structured data storage for large tables.
HDFS: A distributed file system that provides high throughput access to application data.
Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying.
MapReduce: A software framework for distributed processing of large data sets on compute clusters.
Pig: A high-level data-flow language and execution framework for parallel computation.
ZooKeeper: A high-performance coordination service for distributed applications.

- Hadoop is popular.

  Adopted by many companies.

- Yahoo:

  More than 100,000 CPUs in more than 25,000 computers running it.

  The biggest cluster: 4000 nodes,

  2 x 4CPU boxes, with 4 x 1 TB disk, and 16 GB RAM.

- Amazon :

  Process millions of sessions daily for analytics.

  Using both Java and streaming APIs.

- Facebook:

  Use it to store copies of internal log and dimension data sources.

  a source for reporting and analytics, with machine learning algorithms.

- Facebook:

  Use it to analyze the log of search, data mining on web page database.

  Process 3000 TB data per week.

- Distributed File system,

  Every long file is split into blocks of 256 MB,

  each block is allocated to different storage node.

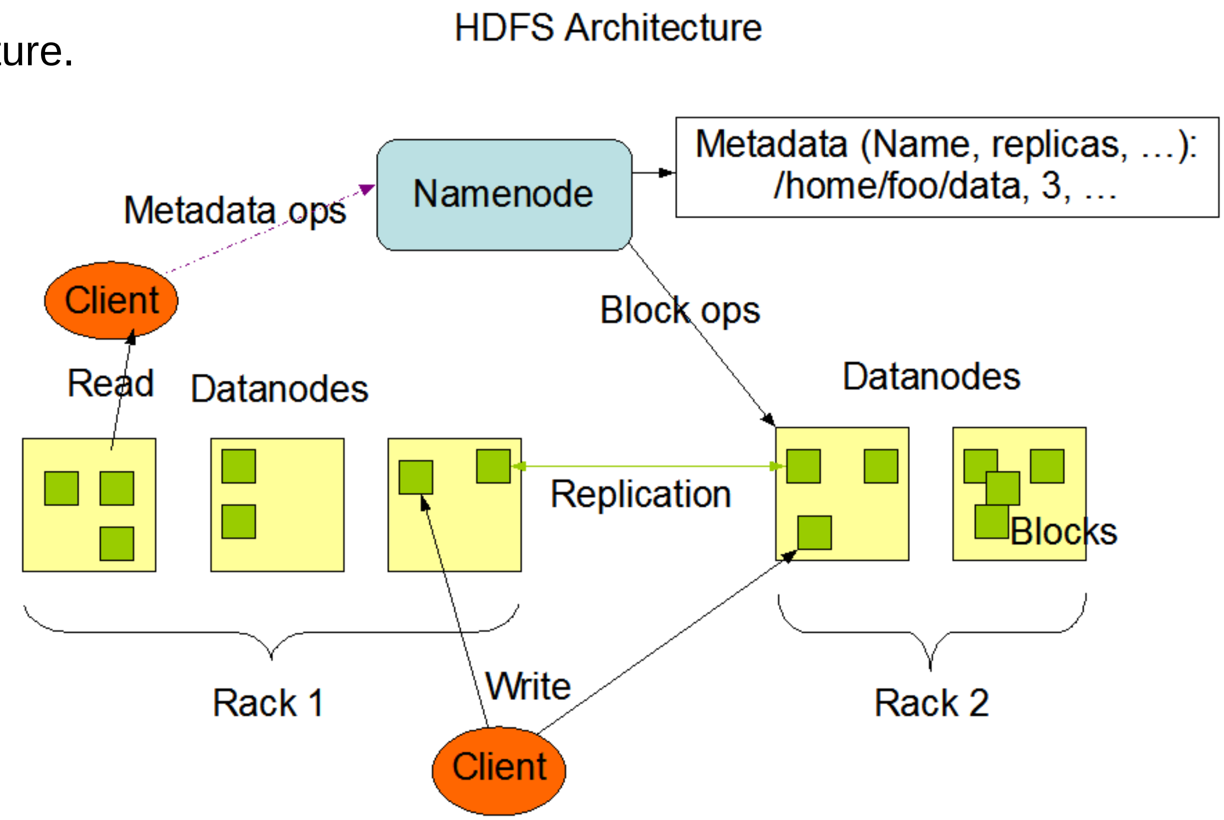
- Reliability,

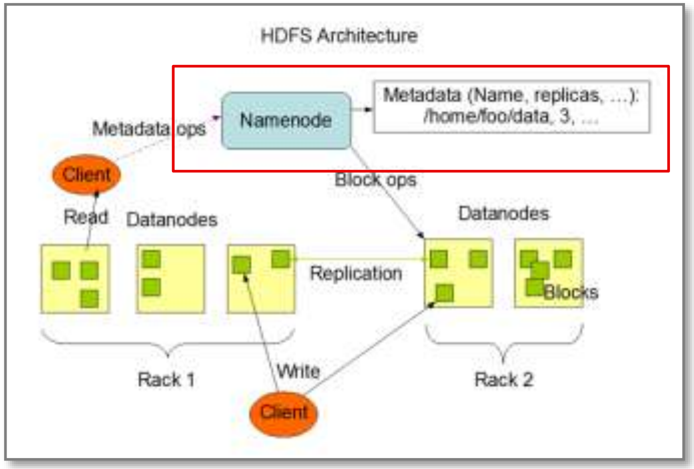
  Each block has multiple replica.

- Master-Slave architecture.

  Master: Namenode,

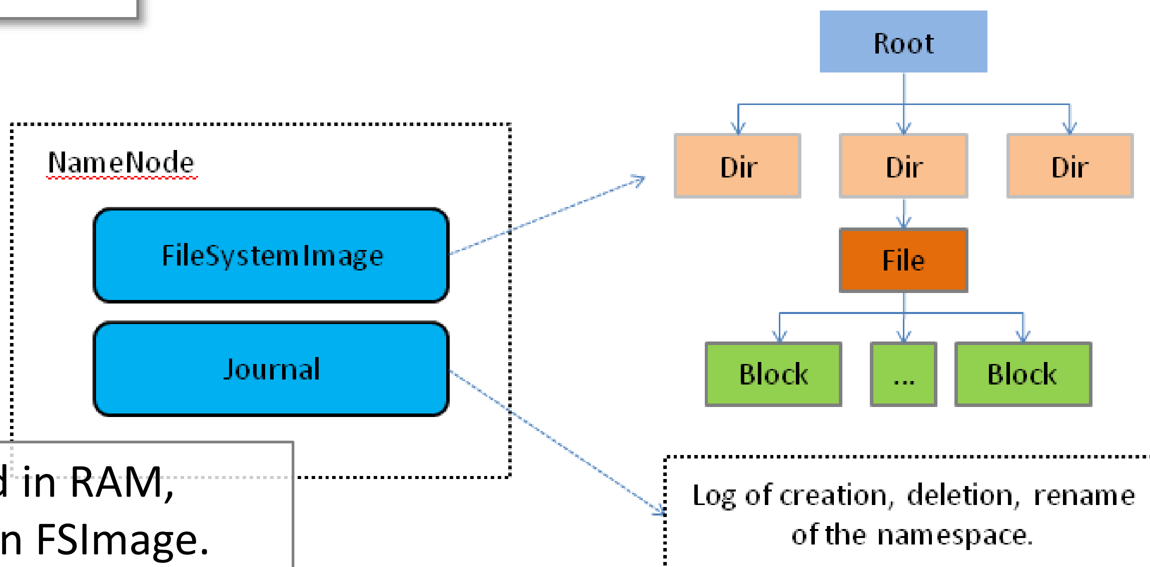
  Slave: Datanode.





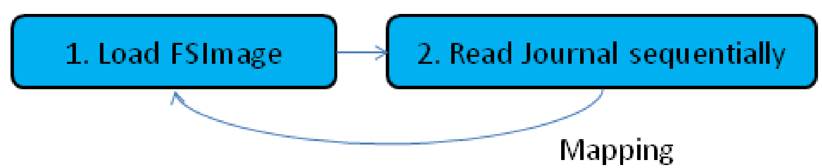
Namenode handles the FS namespace, including open, close, rename file or directory.

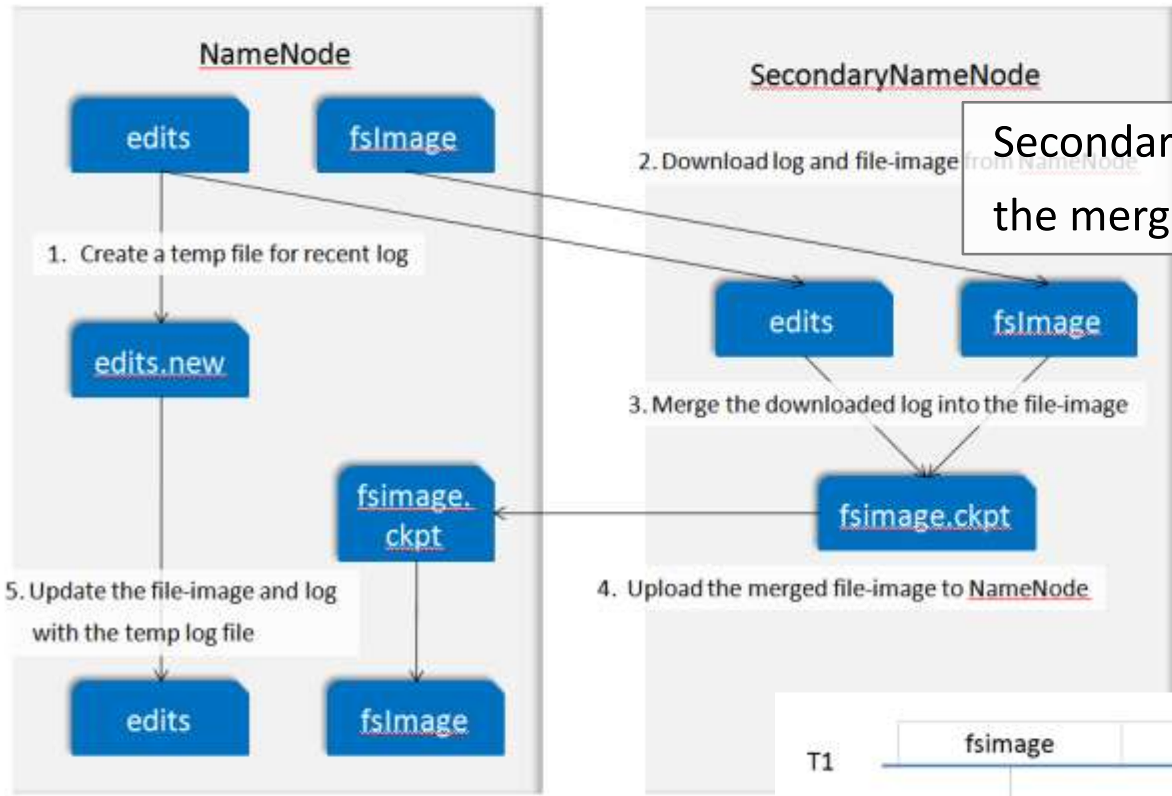
Namenode supervises the creation, deletion and copy of blocks.



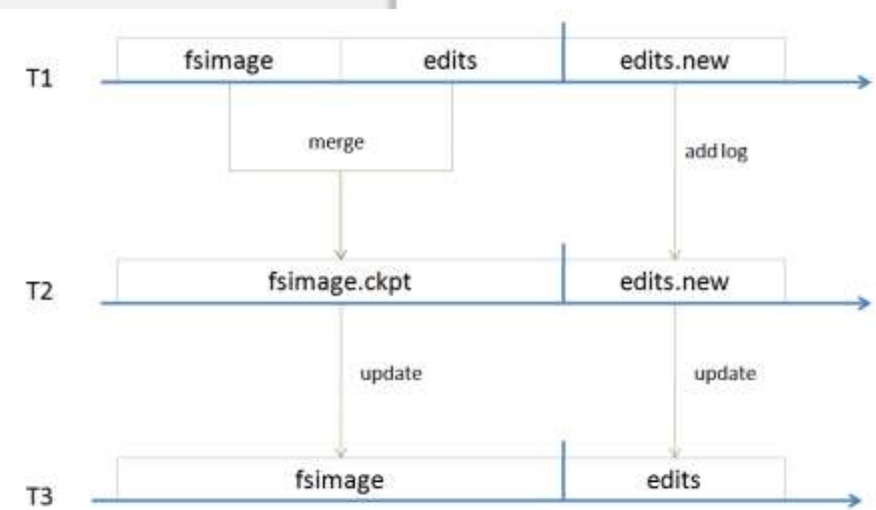
Namespace tree is cached in RAM, also stored permanently in FSImage.

Log takes record of open, close, rename file or directory, etc.





SecondaryNameNode executes the merge of fsimage and edits.



NameNode is of Journaling file system.

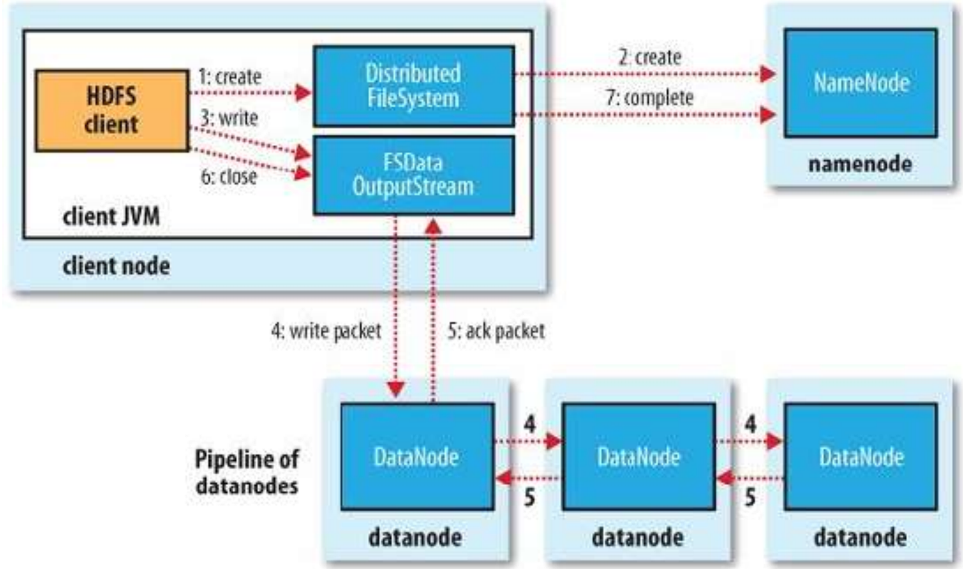
Client's request to create, delete and rename directory and file,

is written into "edits" first,
then merged into "fsimage".

HDFS Create

FSDataOutputStream is returned by DistributedFileSystem after contacting NameNode.

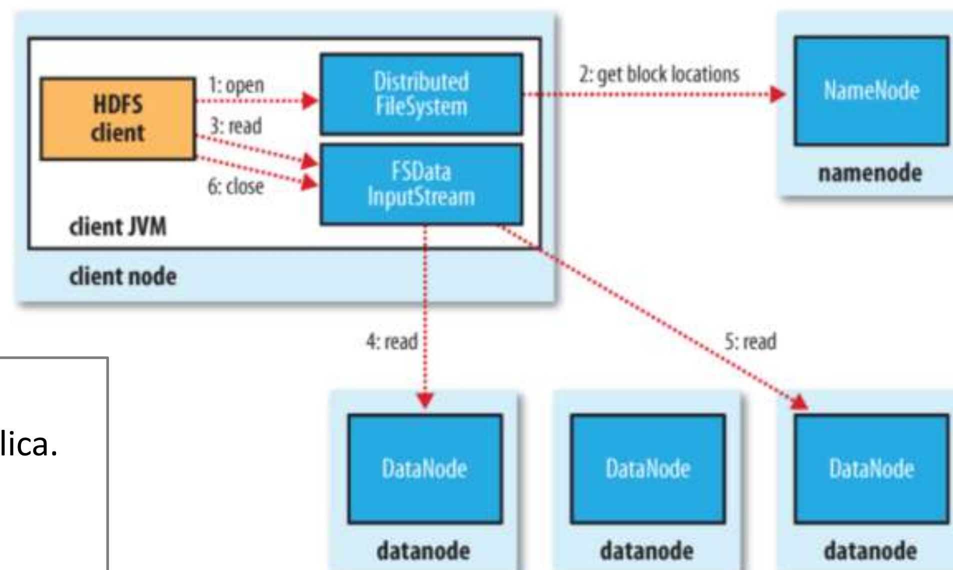
Upload file is split into multiple packets.
Only after the first packet has been stored into all the replica,
the second packet begins to upload.

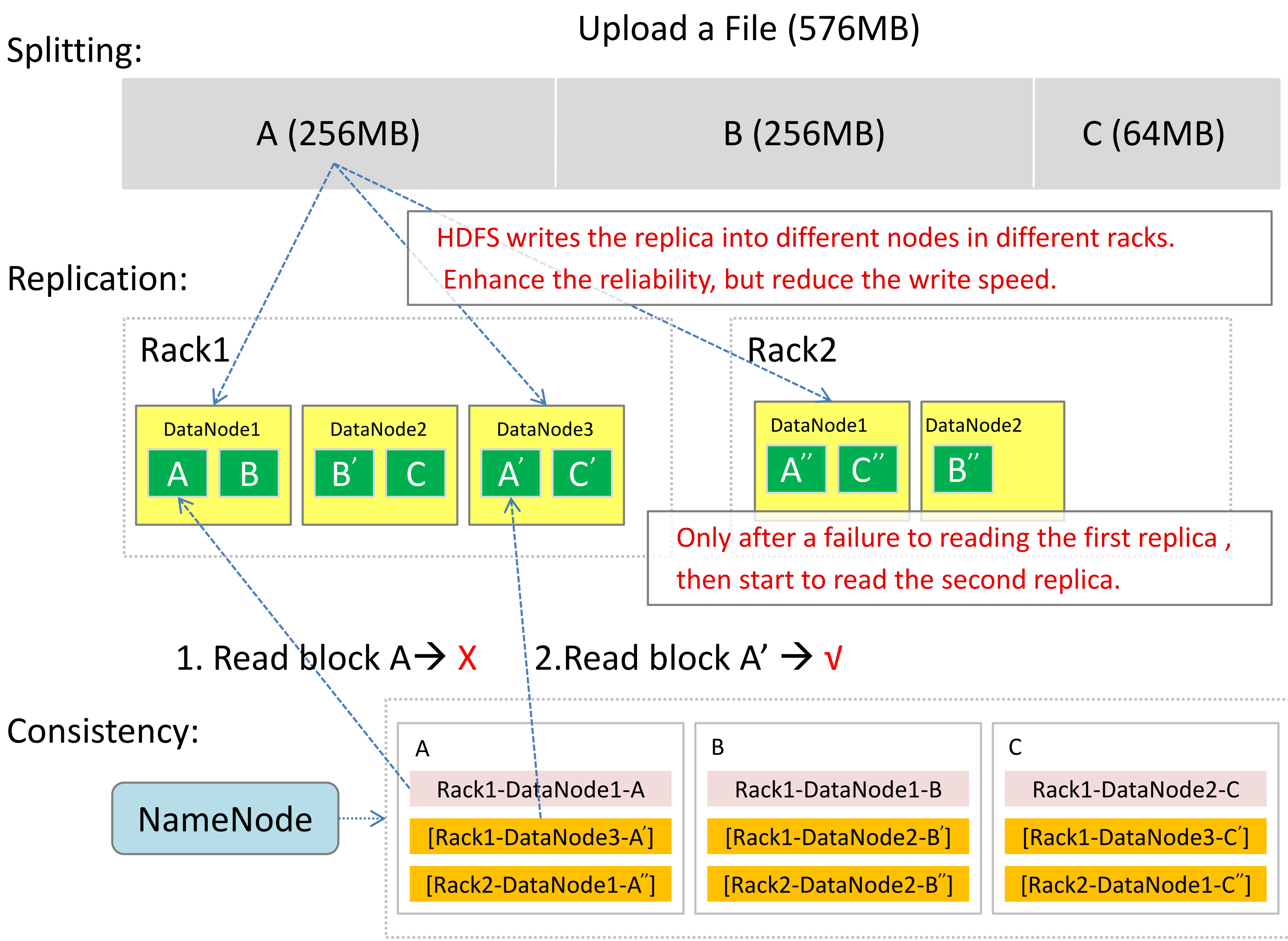


HDFS Read

FSDataOutputStream is returned by DistributedFileSystem after contacting NameNode to find the block address.

FSDataOutputStream read the block
from the datanode, if fails it goes to the replica.
Only after finishing reading the first block,
it goes to the second.




Splitting:
Upload a File (576MB)
A (256MB)
B (256MB)
C (64MB)
HDFS writes the replica into different nodes in different racks.
Enhance the reliability, but reduce the write speed.
Replication:
Rack1
DataNode1
A
B
DataNode2
B′
C
DataNode3
A′
C′
Rack2
DataNode1
A″
C″
DataNode2
B″
Only after a failure to reading the first replica ,
then start to read the second replica.
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″]

- Hadoop HDFS:

  An open source implementation of Google File System.

- Master-Slave Architecture:

  Namenode, the master for namespace, i.e. directory and file.

  Datanode, the blocks for data storage.

- Namenode is of Journaling File System:

  Creation, deletion, rename of the namespace is written into “edits” first,

  Then merge into the FSImage file, by the SecondaryNamenode.

- HDFS write,

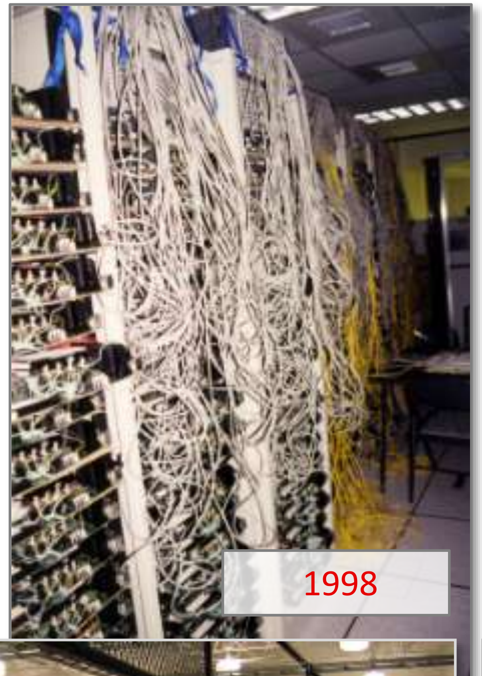
  A file splits into packets,

  After the first packet is written into every replica,

  then the second packet starts to upload.

Google File System


1997

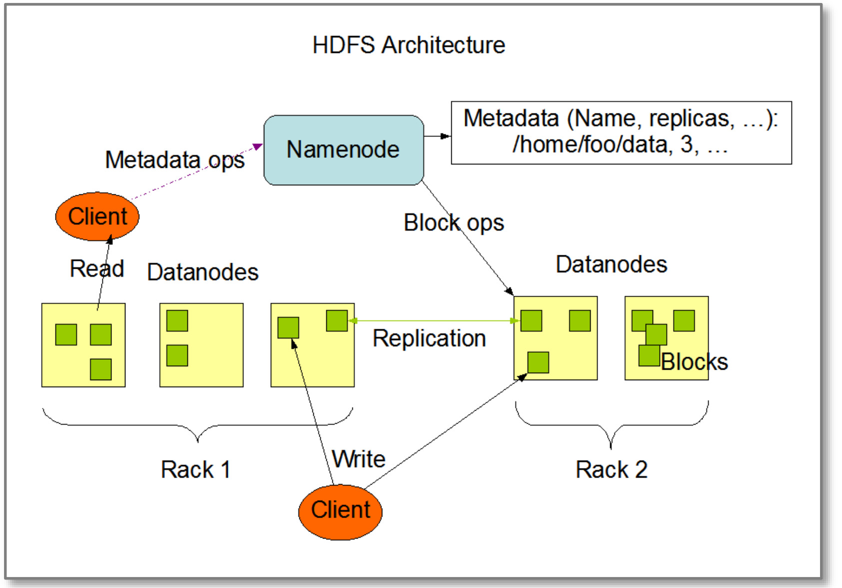

1998


2000


2001


2009

- HDFS and Google File System, are very similar to each other.
- A single master with shadow masters. Multiple chunk servers, containing fixed size data blocks.
- HeartBeat message between the master and chunkservers to ensure each other's aliveness.



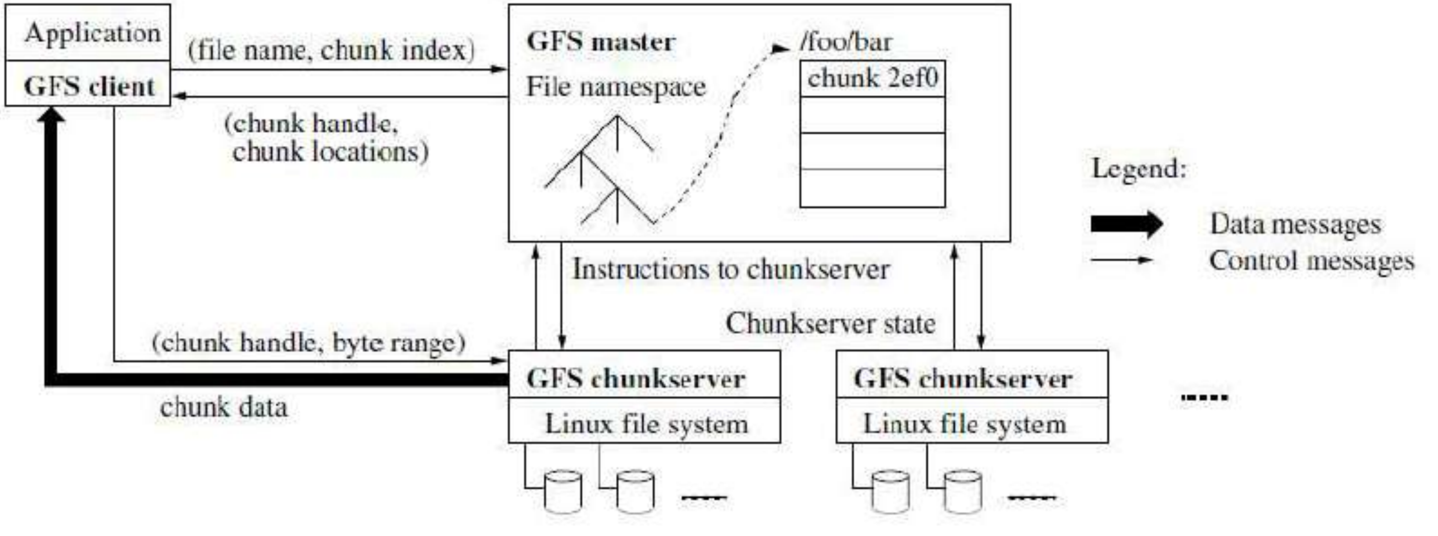
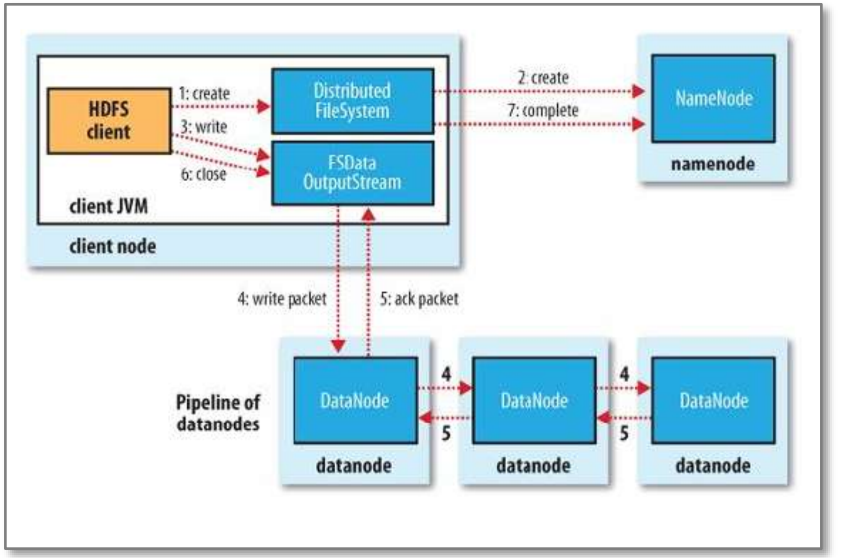


Figure 1: GFS Architecture

- 1,2: A client asks the master for all the replica addresses.
- 3: The client sends data to the nearest chunk server, the other chunk servers are in pipeline, the received data is buffered in cache.
- 4: The client sends a write request to the primary.
- 5: The primary replica decides the offset of the received data in the chunk.
- 6: Completion messages from secondary replicas.
- 7: The primary replies to the client.



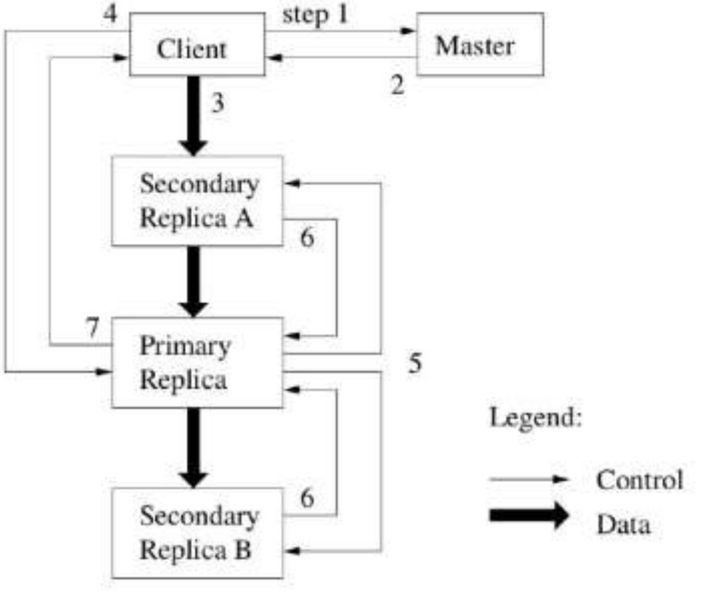


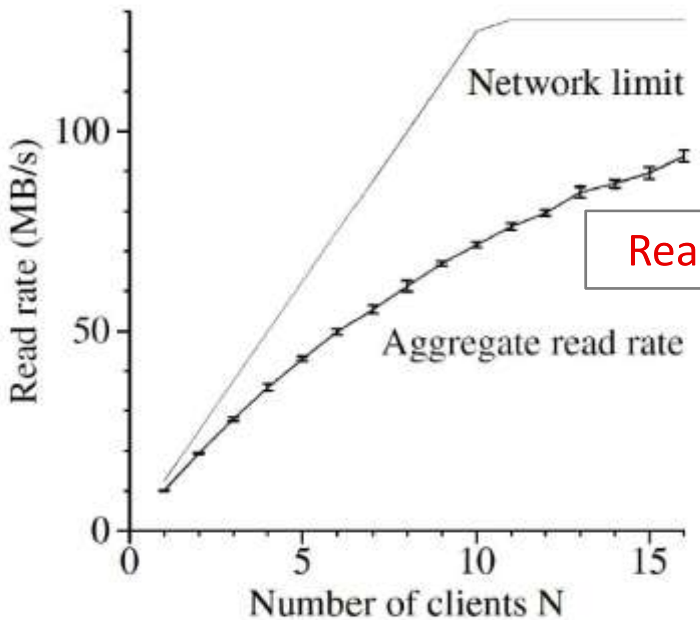
Figure 2: Write Control and Data Flow

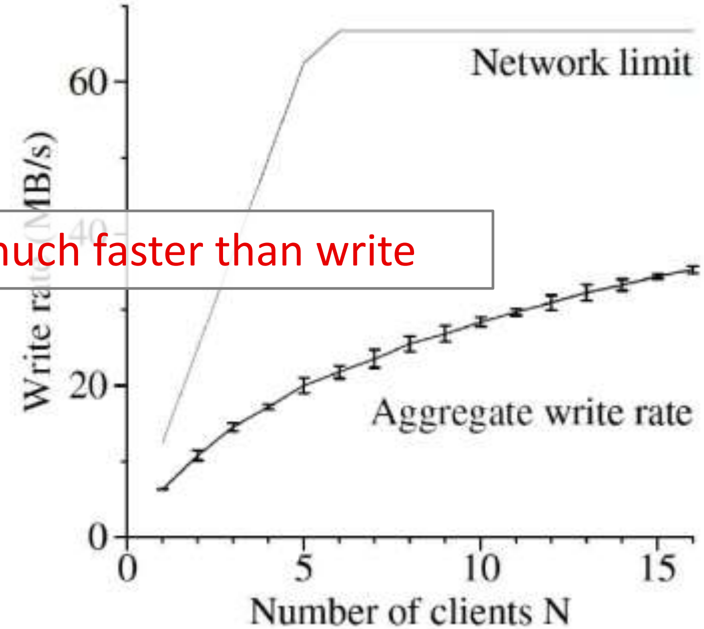
Experimental Environment

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

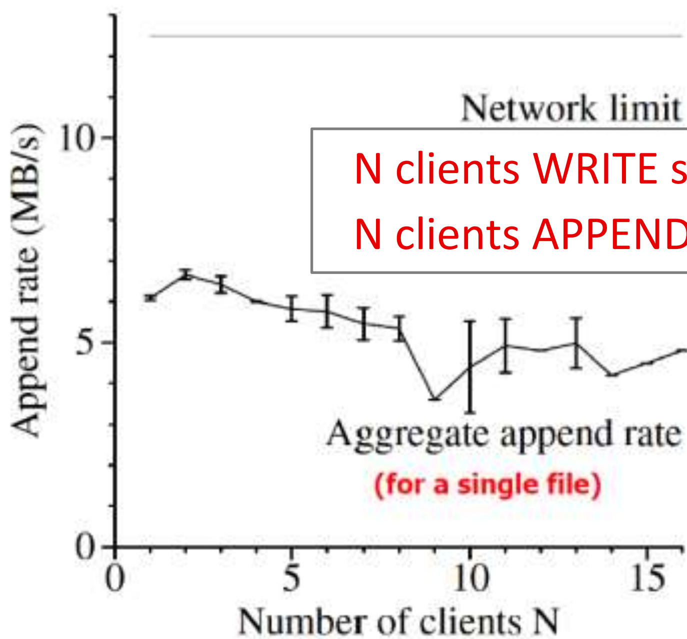
Aggregate Rates

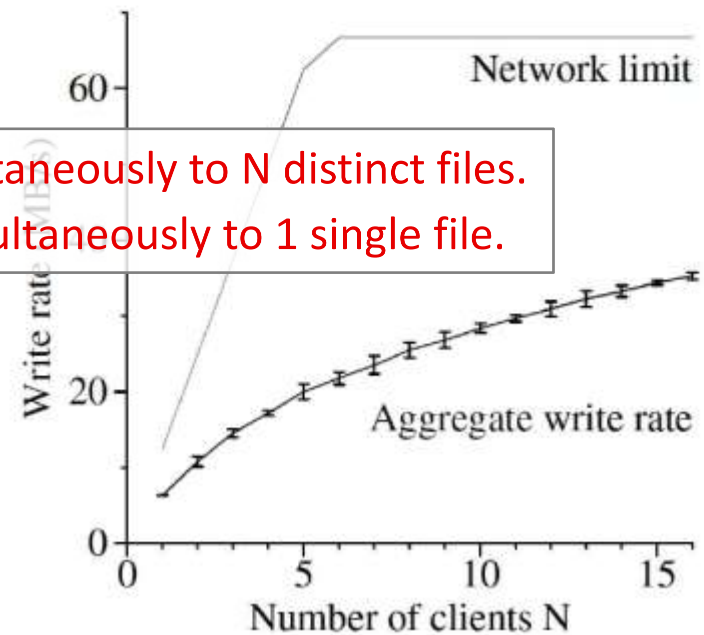
Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s





Read is much faster than write





N clients WRITE simultaneously to N distinct files.

N clients APPEND simultaneously to 1 single file.

- GFS is for large files, not optimized for small files:

  Millions of files, each >100 MB or >1 GB,

  from BigTable, MapReduce records, etc.

GFS is for big files

- Workload: streaming

  Large streaming reads (> 1MB), small random reads (a few KBs).

  Sequentially append, and seldom modified again.

  Response time for read and write is not critical.

- GFS has no directory (inode) structure:

  Simply uses directory-like filenames, e.g. /foo/bar

  Thus listing files in a directory is slow.

- Re-replication:

  When the number of replicas falls below predefined threshold,

  the master assigns the highest priority to clone such chunks.

- Recovery:

  The master and the chunk servers restore their states within a few seconds.

  The shadow master provides read-only accesses.

- GFS is NOT for all kinds of files:

  Migrate files to other FS's which fit better,

  Picasa, Gmail, Project codes.

GFS is not a panacea

- Highly relaxed consistency:

  3 replica, but okay for "at-least-once", sometimes too risky.

  No transaction guarantee, "all succeed or roll back".

- Master is the bottleneck:

  All client has to contact the master first.

  The namespace of all directories and files, cannot be too big.

Q&A

- No stupid questions, but it is stupid if not ask.
- When sleepy, the best trick to wake up is to ask questions.