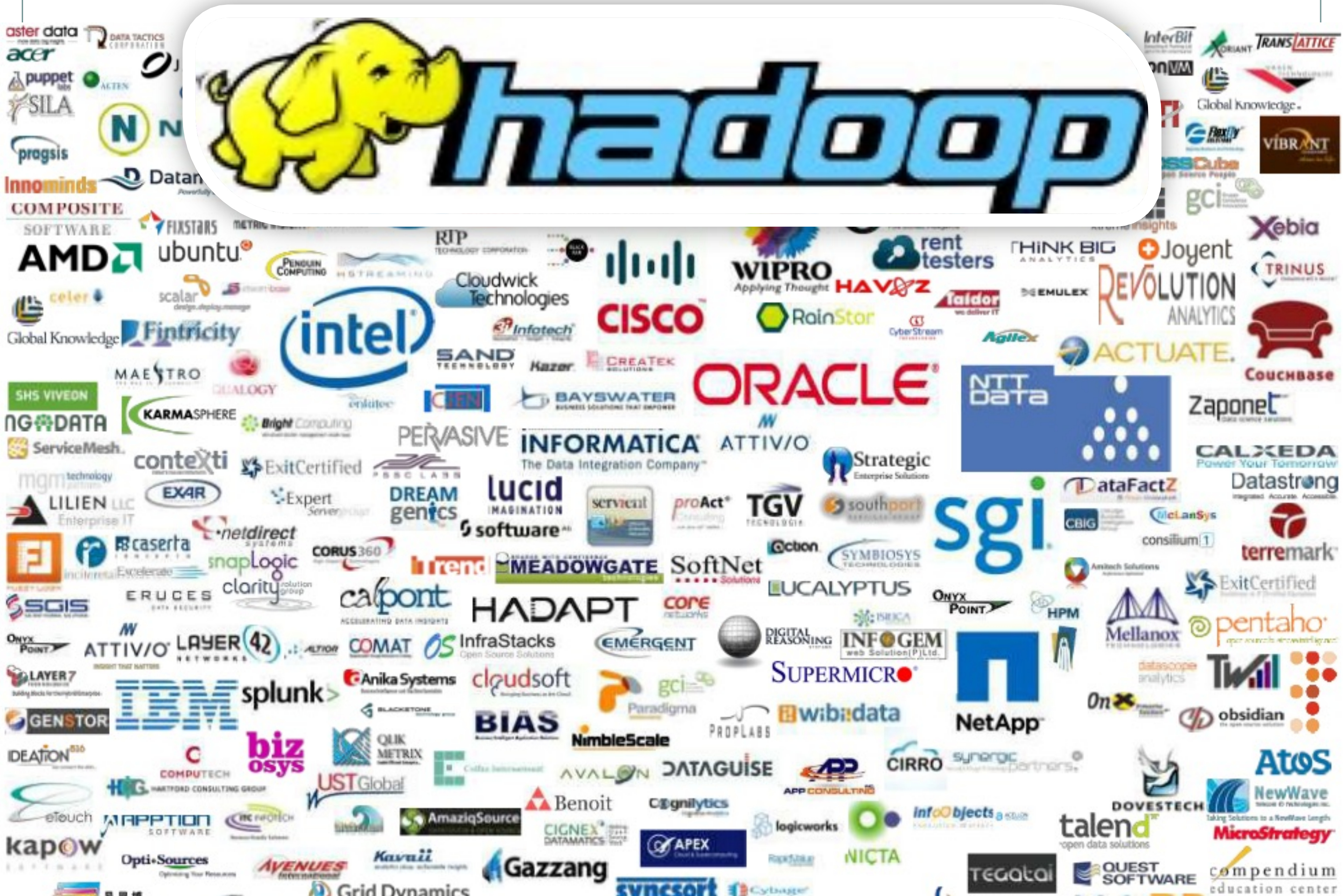
Hadoop File System

SLIDES MODIFIED FROM PRESENTATION BY B. RAMAMURTHY

Moving Computation is Cheaper than Moving Data



Motivation: Big Data!

- What is BigData ?
  - Google (over 20~ PB/day)
- Where does it come from ?
- Why to take so much of pain ?
  - Information everywhere, but where is the knowledge?
- Existing systems (vertical scalibility)
- Why Hadoop (horizontal scalibility)?

Origin of Hadoop

- Google File System 2003
- MapReduce paper in OSDI 2004 (Google)
  - new programming paradigm to handle data at internet scale
  - Google developed GFS
- Hadoop/HDFS open source version of MapReduce
  - Hadoop started as a part of the Nutch project.
  - In Jan 2006 Doug Cutting started working on Hadoop at Yahoo
  - Yahoo gave it to Apache
- First release of Apache Hadoop in September 2007
  - Jan 2008 - Hadoop became a top level Apache project

What is Hadoop ?

- Flexible infrastructure for large scale computation & data processing on a network of commodity hardware
- Completely written in java
- Open source & distributed under Apache license
- Hadoop Common, HDFS & MapReduce

Hadoop distributions

- Amazon
- Cloudera
- MapR
- HortonWorks
- Microsoft Windows Azure.
- IBM InfoSphere Biginsights
- Datameer
- EMC Greenplum HD Hadoop distribution
- Hadapt

Google/GFS Assumptions

- Based on Google workloads
  - Also the assumptions driving HDFS

1. Hardware failure common
2. Files are large, and numbers are limited (millions not billions)
3. Two main types of reads: large streaming reads and small random reads
4. Sequential writes that append to files
5. Files rarely modified (simplifies coherence)
6. High sustained bandwidth preferred to low latency

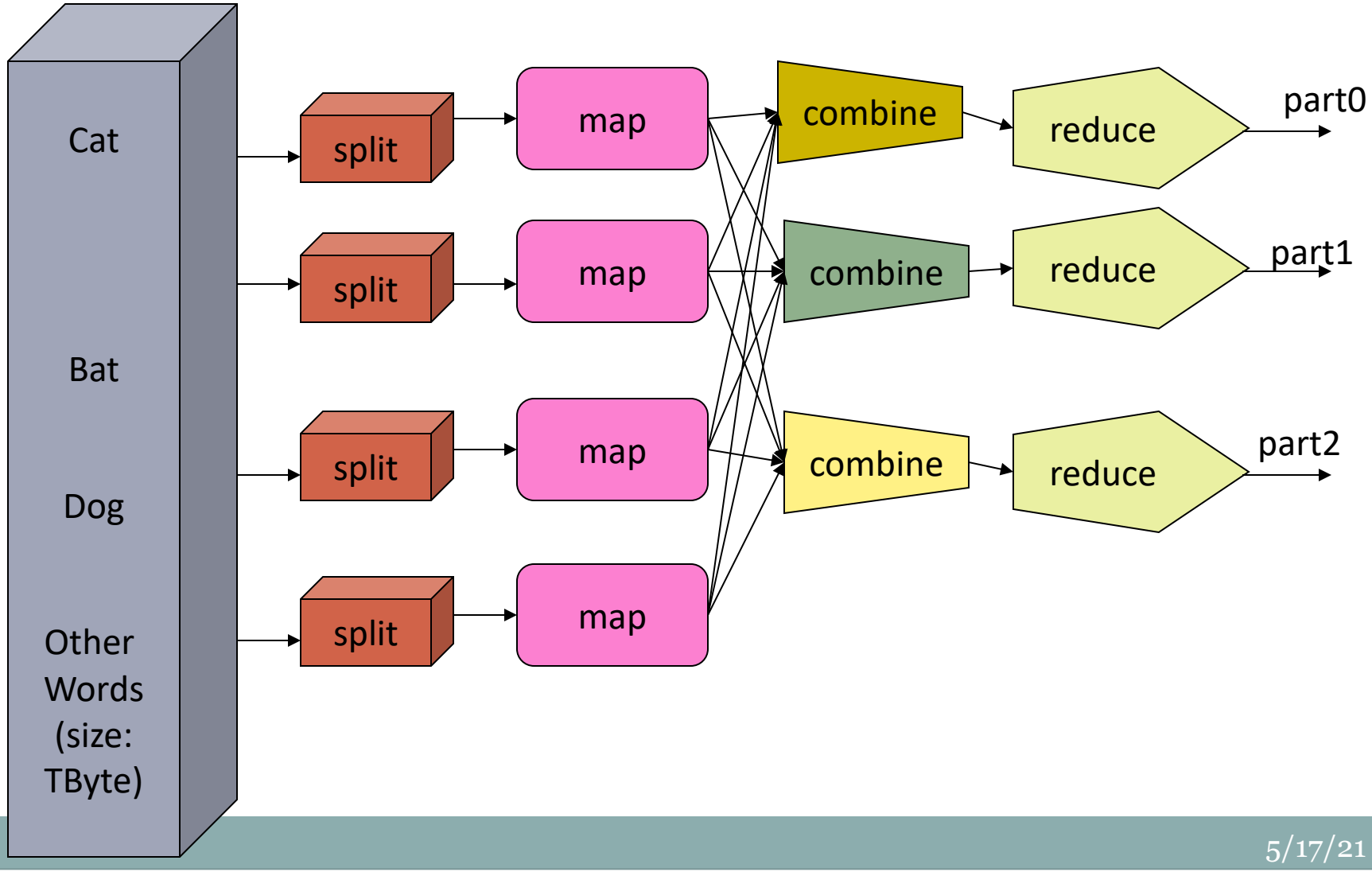
Basic Features: HDFS

- Highly fault-tolerant
  - Thousands of server machines
  - Failure norm rather than exception
- High throughput
  - Internet scale workloads
  - Move compute to data (e.g., MapReduce)
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware
  - Compare to RAID

Data Characteristics

- Large data sets and files: gigabytes to terabytes size
  - Tens of millions of files in a single instance
- Applications need streaming access to data
  - Batch processing rather than interactive user access.
- Scale to thousands of nodes in a cluster
- Write-once-read-many: a file once created rarely needs to changed
  - Assumption simplifies coherence
  - Internet workloads (e.g., map-reduce or web-crawler) fits model

MapReduce


Cat
Bat
Dog
Other Words (size: TByte)
split
split
split
split
map
map
map
map
combine
combine
combine
reduce
reduce
reduce
part0
part1
part2

Architecture

Whats new for us?

- Why not just run NFS?
  - Scalability
  - Performance
  - Elasticity
  - Reliability
- Is it a different form of RAID?

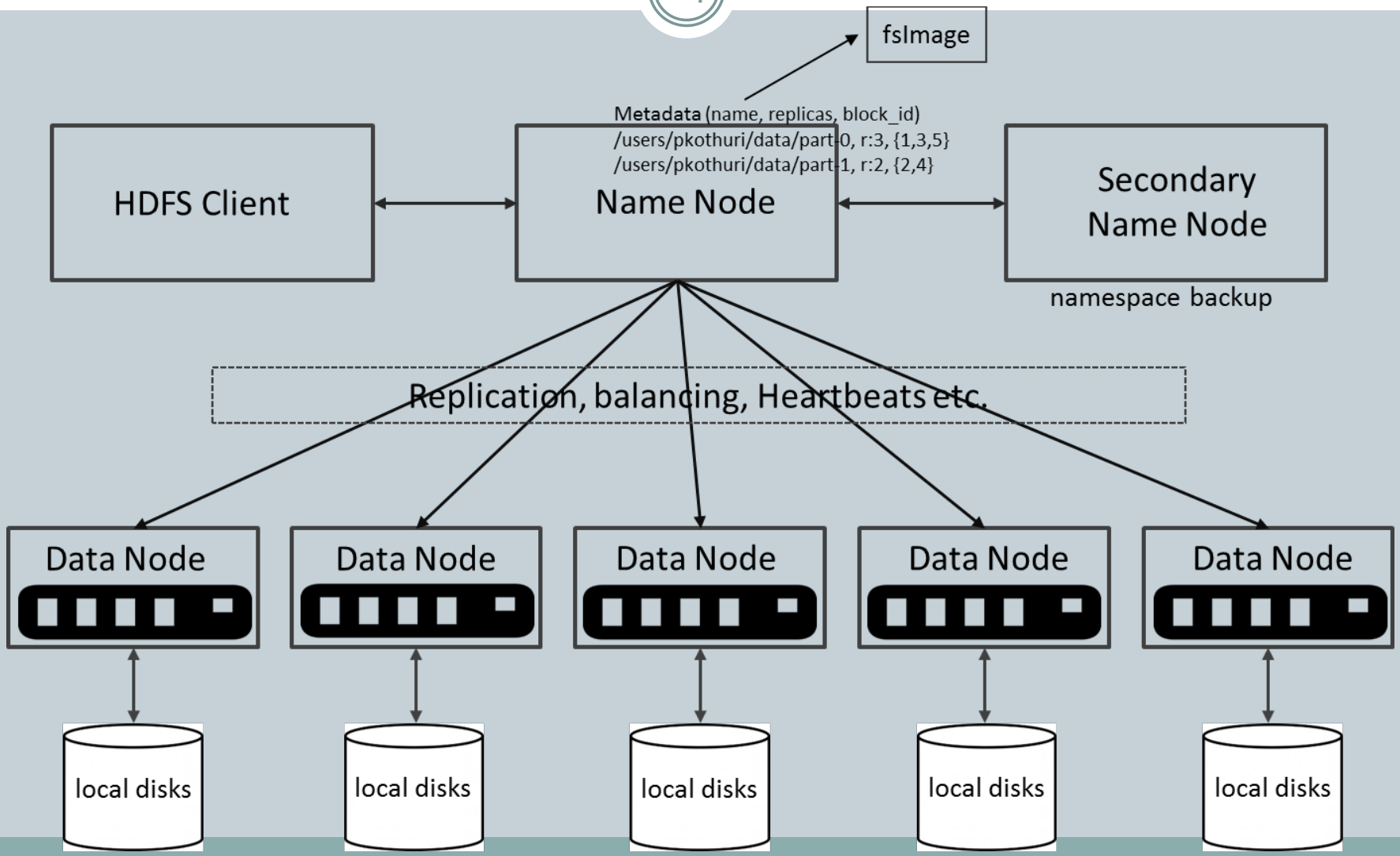
Namenode and Datanodes

- HDFS exposes a file system
- Each file is split into one or more

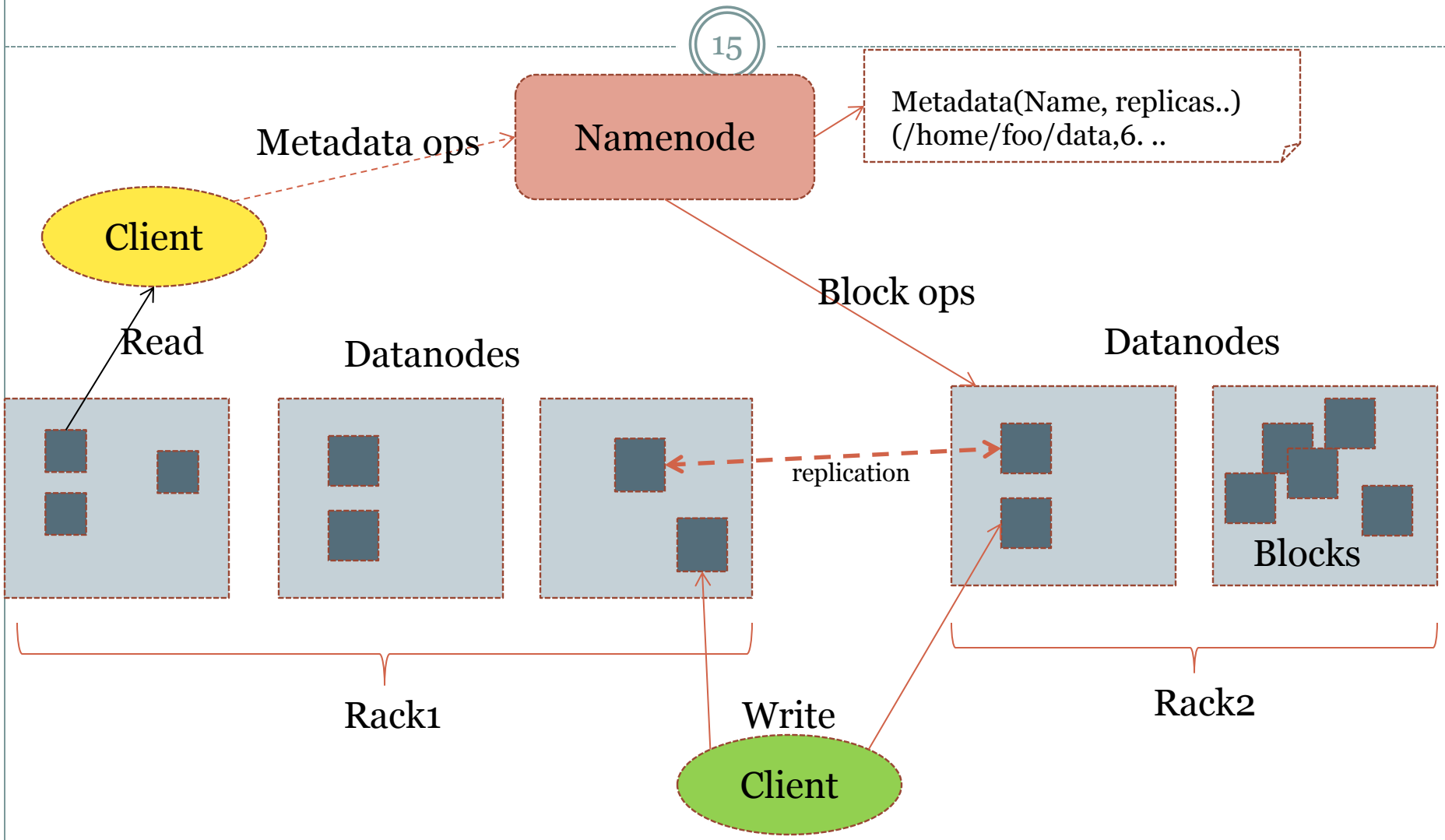
- A single **Namenode**
  - Maintains metadata and name space
  - Regulates access to files by clients
  - Carries out rebalancing and fault recovery

- Many **DataNodes,** usually one per node in a cluster
  - DataNodes manage storage attached to the nodes that they run on
  - Serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode
  - Communicates to Namenode via heartbeats
    - Namenode communicates back through replies; otherwise clients talk directly to datanodes

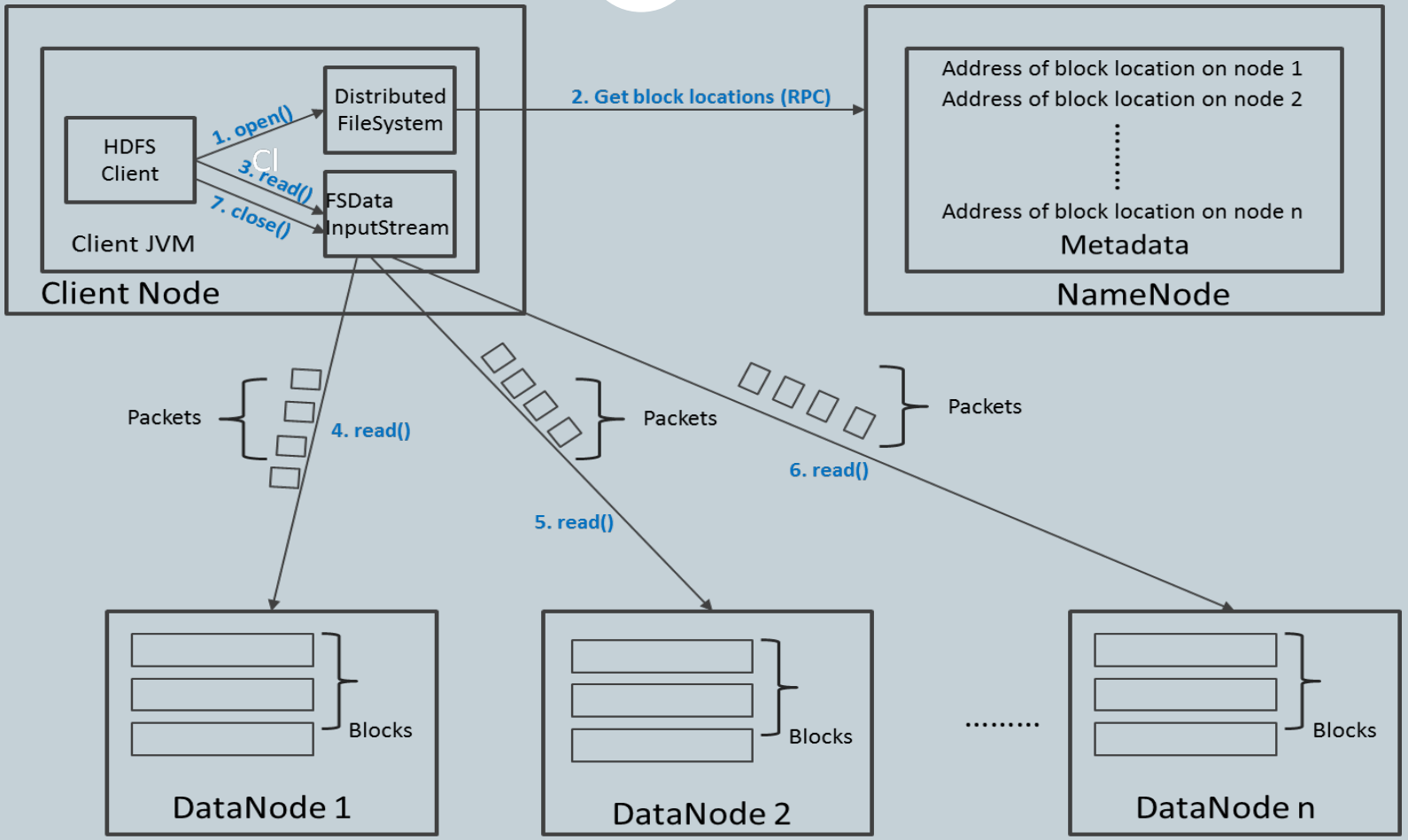
HDFS Architecture



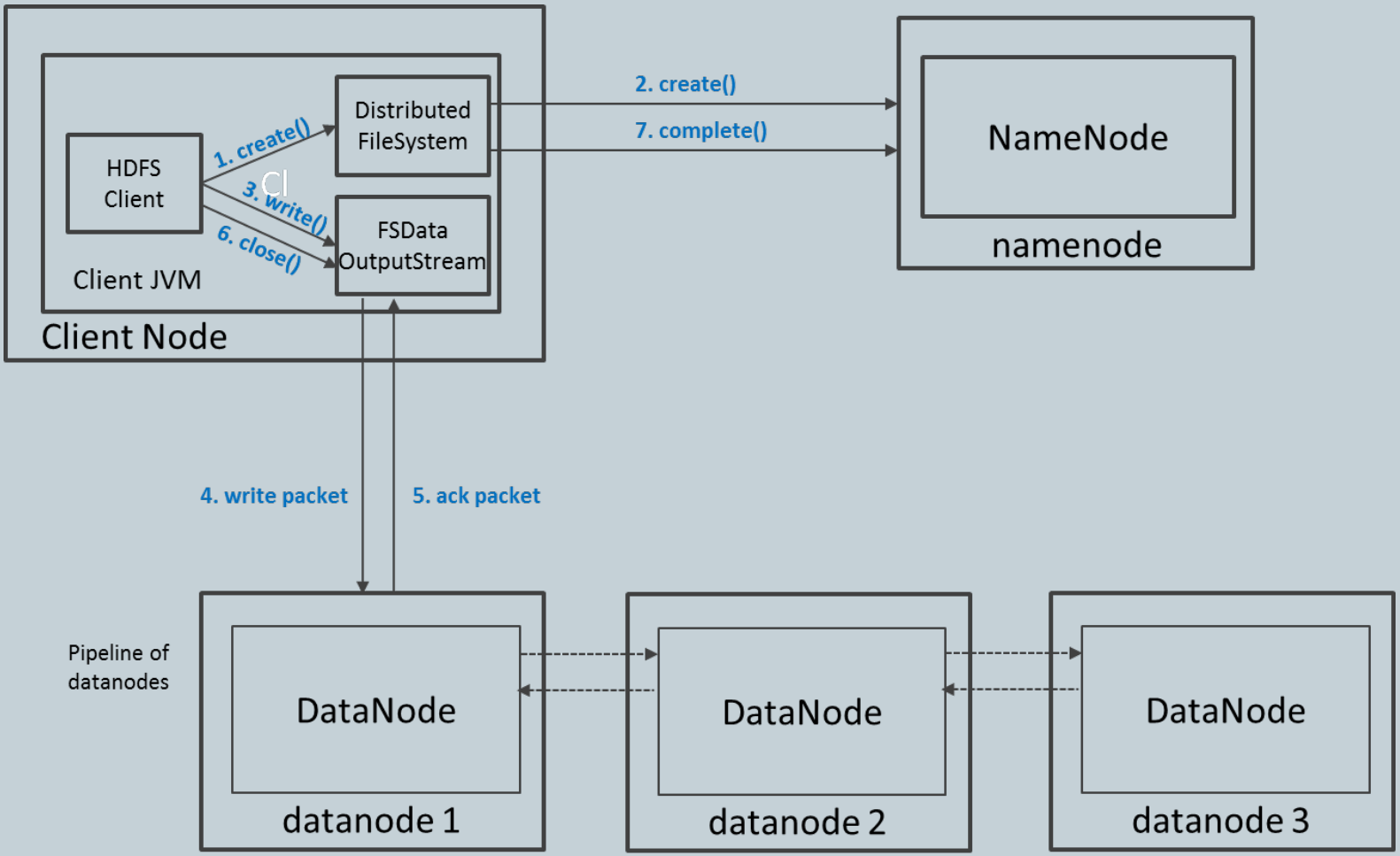
HDFS Architecture



Read Operation in HDFS



Write Operation in HDFS



File system Namespace

- Hierarchical file system with directories and files
  - Create, remove, move, rename etc.
- File system API started as unix compatible
  - “faithfulness to standards was sacrificed in favor of improved performance …”
- Any meta information changes to the file system recorded by the Namenode.

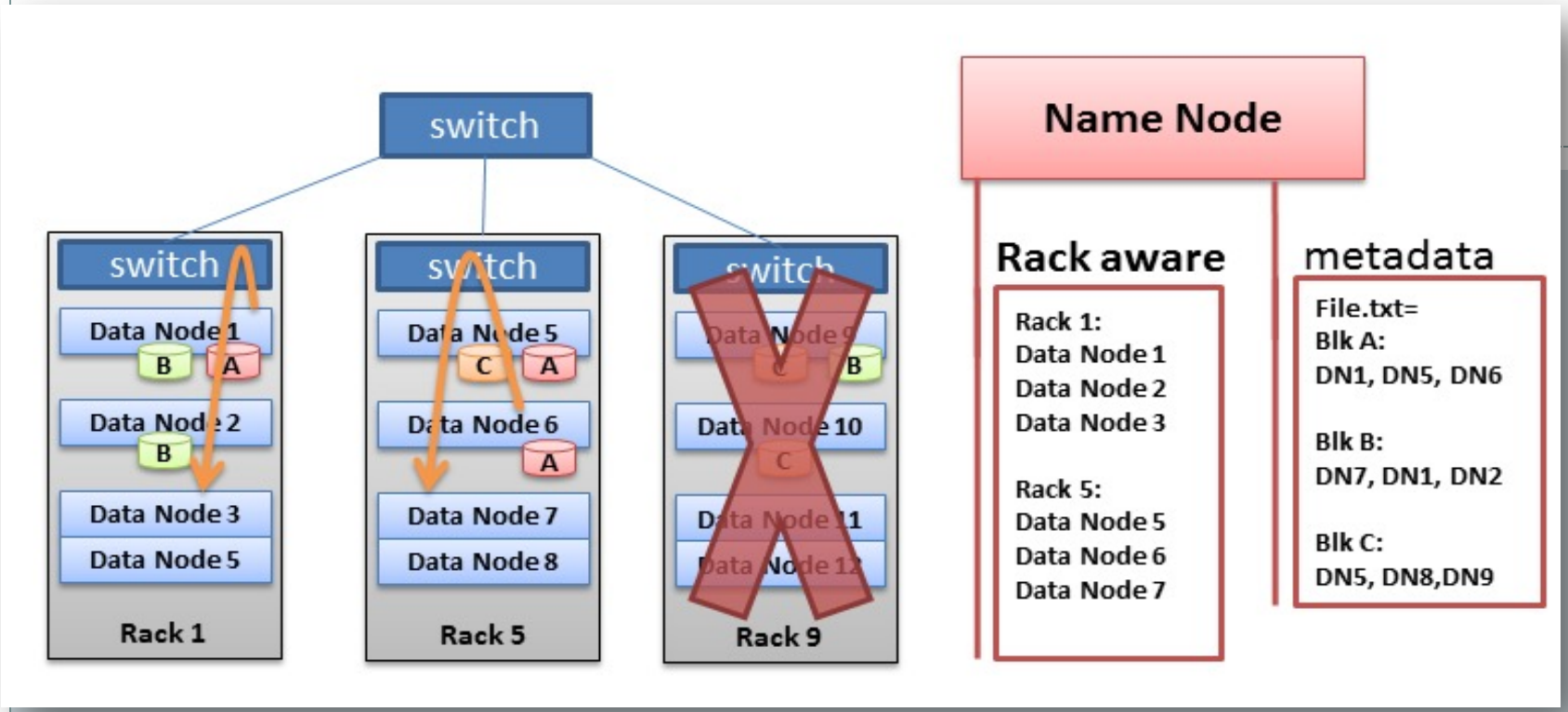
Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
  - Each file is a sequence of blocks.
  - All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
  - Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
  - BlockReport contains all the blocks on a Datanode.

Replica Placement

- The placement of the replicas is critical to HDFS reliability and performance.
- Rack-aware replica placement:
  - Goal: improve reliability, availability and network bandwidth utilization
  - Research topic
- Many racks, communication between racks are through switches.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - Writes are expensive
  - Replication factor is 3
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack and one on a node in a different rack.

Rack awareness



Typically large Hadoop clusters are arranged in racks and network traffic between different nodes with in the same rack is much more desirable than network traffic across the racks. In addition Namenode tries to place replicas of block on multiple racks for improved fault tolerance. A default installation assumes all the nodes belong to the same rack.

Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode's local filesystem
  - Metadata only
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode's local filesystem.

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
  - 4GB of local RAM is sufficient
- When Namenode starts up it gets the FsImage and Editlog from its local file system
  - update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

Datanode

- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- It stores each block of HDFS data in a separate file.
- Datanode does not create all files in the same directory.
- It uses heuristics to determine optimal number of files per directory and creates directories appropriately:
  - Research issue?
- When the filesystem starts up it generates a list of all HDFS blocks and send this report to Namenode: Blockreport.

Protocol

The Communication Protocol

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

Robustness

Objectives

- Primary objective of HDFS is to store data reliably in the presence of failures.
- Three common failures are: Namenode failure, Datanode failure and network partition.

DataNode failure and heartbeat

- A network partition can cause a subset of Datanodes to lose connectivity with the Namenode.
- Namenode detects this condition by the absence of a Heartbeat message.
- Namenode marks Datanodes without Hearbeat and does not send any IO requests to them.
- Any data registered to the failed Datanode is not available to the HDFS.
- Also the death of a Datanode may cause replication factor of some of the blocks to fall below their specified value.

Re-replication

- Re-replication (creating more replicas) is sometimes needed
- Re-replication could be invoked due to:
  - A Datanode is unavailable,
  - A replica is corrupted,
  - A hard disk on a Datanode fails, or
  - The replication factor on the block may be increased.

Metadata Disk Failure

- FsImage and EditLog are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- Meta-data is not data-intensive.
- The Namenode could be single point failure: automatic failover originally NOT supported!

Discussion

- Single metadata server - why?
- Can we replicate?
  - Consistency issues!
- Can we parallelize?
  - Yes, GFS does it, as do several research DFS
  - But by partitioning directory space to different servers - no replication

Data Organization

Data Blocks

- HDFS support write-once-read-many with reads at streaming speeds.
- A typical block size is 64MB (or even 128 MB).
- A file is chopped into 64MB chunks and stored.

Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.

Staging (contd.)

- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion; also it has precedence is AFS (Andrew file system).

Replication Pipelining

- When the client receives response from Namenode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on.
- Thus data is pipelined from Datanode to the next.

API (Accessibility)

Application Programming Interface

- HDFS provides Java API for application to use.
- Python access is also used in many applications.
- A C language wrapper for Java API is also available.
- A HTTP browser can be used to browse the files of a HDFS instance.

FS Shell, Admin and Browser Interface

- HDFS organizes its data in files and directories.
- It provides a command line interface called the FS shell that lets the user interact with data in the HDFS.
- The syntax of the commands is similar to bash and csh.
- Example: to create a directory /foodir

/bin/hadoop dfs –mkdir /foodir

- There is also DFSAdmin interface available
- Browser interface is also available to view the namespace.

Space Reclamation

- When a file is deleted by a client, HDFS renames file to a file in be the /trash directory for a configurable amount of time.
- A client can request for an undelete in this allowed time.
- After the specified time the file is deleted and the space is reclaimed.
- When the replication factor is reduced, the Namenode selects excess replicas that can be deleted.
- Next heartbeat(?) transfers this information to the Datanode that clears the blocks for use.

Summary

- We discussed the features of the Hadoop File System, a peta-scale file system to handle big-data sets.
- What discussed: Architecture, Protocol, API, etc.
- Missing element: Implementation
  - The Hadoop file system (internals)
  - An implementation of an instance of the HDFS (for use by applications such as web crawlers).