Distributed Storage Systems

Anthony Kougkas

akougkas@.iit.edu

ILLINOIS INSTITUTE OF TECHNOLOGY

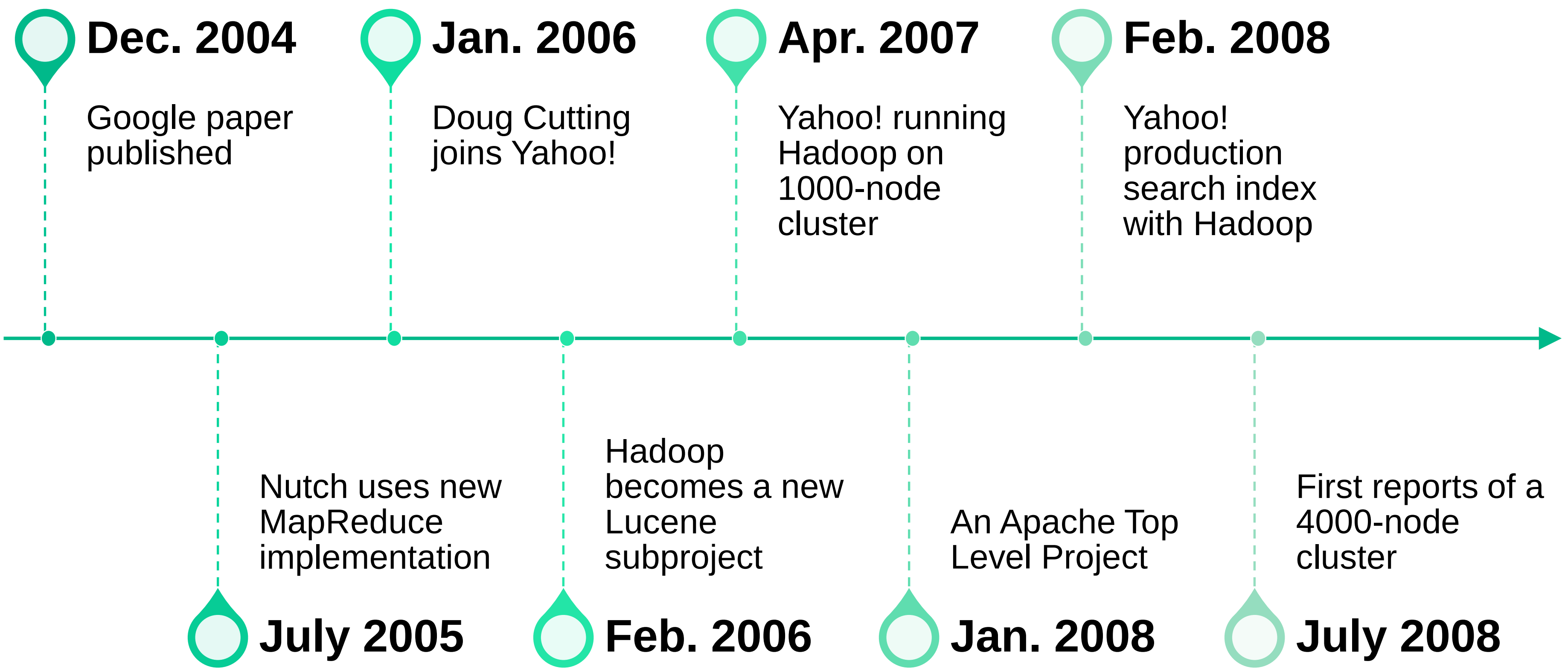
Outline

- Apache Hadoop
- Overview of MapReduce framework
- Distributed file systems (HDFS)
- Parallel file systems (vs Distributed)

What is Apache Hadoop?

- Open source software framework designed for storage and processing of large scale data on clusters of commodity hardware
- Created by Doug Cutting and Mike Carafella in 2005.
- Based on work done by Google in the early 2000s
  - “The Google File System” in 2003
  - “MapReduce: Simplified Data Processing on Large Clusters” in 2004

Apache Hadoop History



Why Apache Hadoop?

- Need to process huge datasets on large clusters of computers
- Nodes fail every day
  - Failure is expected, rather than exceptional.
  - The number of nodes in a cluster is not constant.
- Very expensive to build reliability into each application.
- Need common infrastructure
  - Efficient, reliable, easy to use
  - Open Source, Apache License

Core Hadoop concepts

- The core idea was to distribute the data as it is initially stored
  - Each node can then perform computation on the data it stores without moving the data for the initial processing
- Applications are written in a high-level programming language
- Nodes should communicate as little as possible (share-nothing architecture)
- Data is spread among the machines in advance

Who Uses Hadoop?

facebook

The New York Times

ebay

hp

IBM

JPMorganChase

eHarmony

twitter

intel

NETFLIX

rackspace HOSTING

amazon.com

VISA

NING

SAMSUNG

YAHOO!

What is Hadoop used for?

- Search
  - Yahoo, Amazon, Zvents
- Log processing
  - Facebook, Yahoo, ContextWeb. Joost, Last.fm
- Recommendation Systems
  - Facebook
- Data Warehouse
  - Facebook, AOL
- Video and Image Analysis
  - New York Times, Eyealike

Public Hadoop Clouds

- Hadoop Map-reduce on Amazon EC2 instances
  - http://wiki.apache.org/hadoop/AmazonEC2
- IBM Blue Cloud
  - Partnering with Google to offer web-scale infrastructure
- Global Cloud Computing Testbed
  - Joint effort by Yahoo, HP and Intel

Outline

- Apache Hadoop
- Overview of MapReduce framework
- Distributed file systems (HDFS)
- Parallel file systems (vs Distributed)

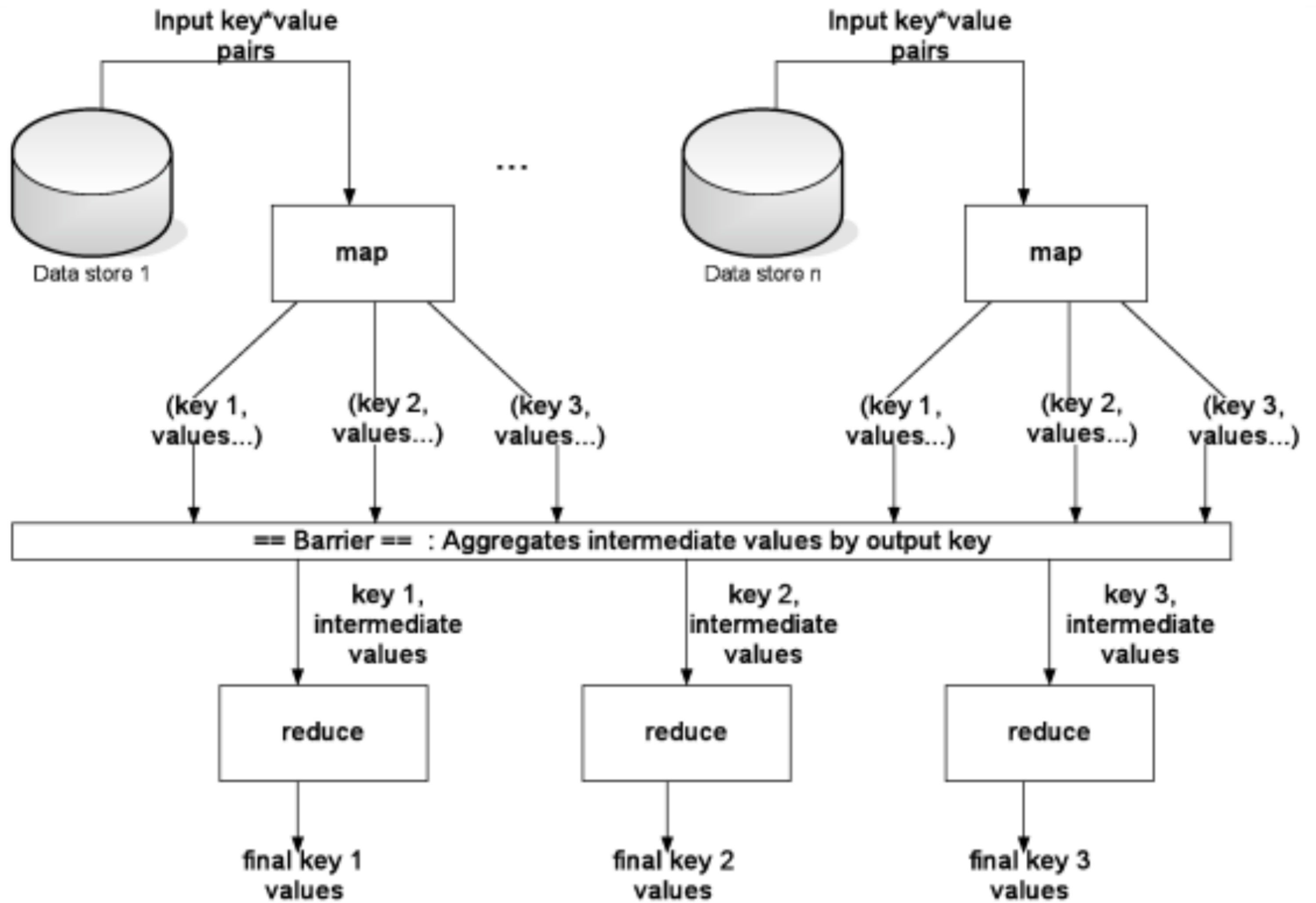
What is MapReduce?

- MapReduce is a method for distributing a task across multiple nodes
- Each node processes data stored on that node
- Consists of two phases:
  - Map
  - Reduce
- Map: (K1, V1) -> (K2, V2)
- Reduce: (K2, list(V2)) -> list(K3, V3)

Why MapReduce is so popular?

- Automatic parallelization and distribution (The biggest advantage!!!)
- Fault-tolerance (individual tasks can be retried)
- Hadoop comes with standard status and monitoring tools
- A clean abstraction for developers
- MapReduce programs are usually written in Java (possibly in other languages using streaming)

MapReduce: High-level



The Mapper

Reads data as key/value pairs

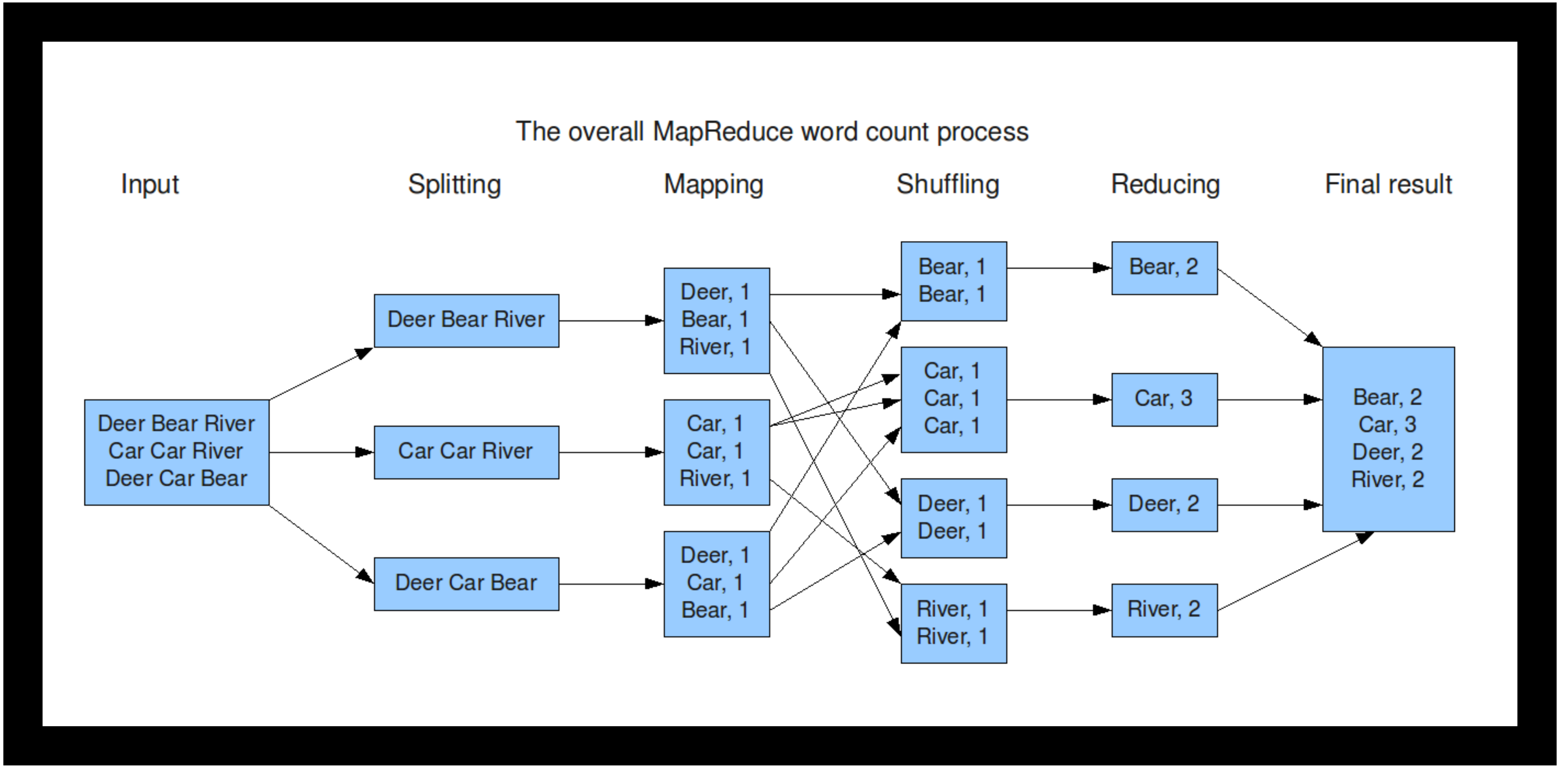
- The key is often discarded

Outputs zero or more key/value pairs

Shuffle and Sort

- Output from the mapper is sorted by key
- All values with the same key are guaranteed to go to the same machine

Example: Word Count



Outline

- Apache Hadoop
- Overview of MapReduce framework
- Distributed file systems (HDFS)
- Parallel file systems (vs Distributed)

HDFS Overview

- Based on Google’s GFS (Google File System)
- Provides redundant storage of massive amounts of data
  - Using commodity hardware
- Data is distributed across all nodes at load time
  - Provides for efficient Map Reduce processing

HDFS Design

- Runs on commodity hardware
  - Assumes high failure rates of the components
- Works well with lots of large files
  - Hundreds of Gigabytes or terabytes in size
- Built around the idea of “write-once, read-many-times”
- Large streaming reads
  - Not random access
- High throughput is more important than low latency

Goals of HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

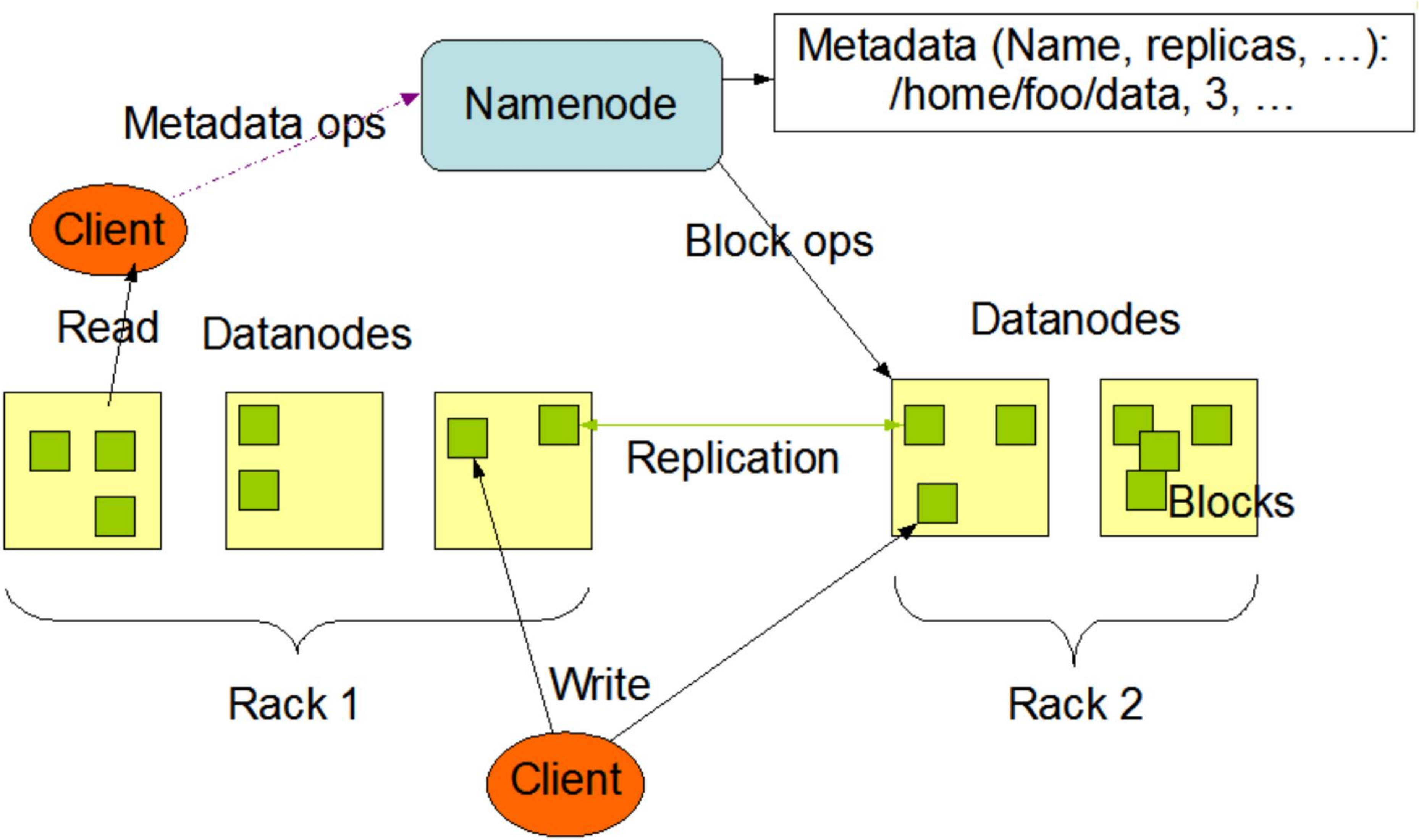
Goals of HDFS(cont.)

- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

HDFS Architecture

- Operates on top of an existing filesystem
- Files are stored as ‘Blocks’
  - Block’s default size – 64MB
- Provides reliability through replication
  - Each Block is replicated across several Data Nodes
- NameNode stores metadata and manages access
- No data caching due to large datasets

Architecture Diagram



Functions of a NameNode

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

NameNode Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas
- Would like to make this policy pluggable

Fault tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system's data.
- Since we have huge number of components and that each component has non-trivial probability of failure means that there is always some component that is non-functional.
- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Heartbeats

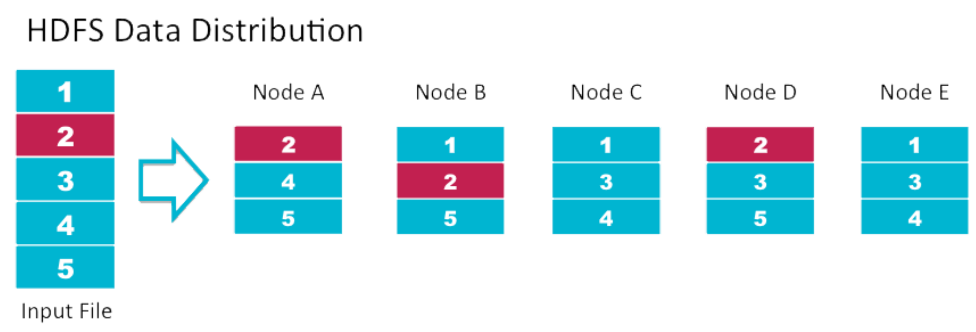
- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

Data replication

- Default replication is 3-fold



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.
- Optimizing replica placement distinguishes HDFS from other distributed file systems.
- Rack-aware replica placement:
  - Goal: improve reliability, availability and network bandwidth utilization
  - Research topic
- Many racks, communication between racks are through switches.
- Network bandwidth between machines on the same rack is greater than those in different racks.
- Namenode determines the rack id for each DataNode.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - Writes are expensive
  - Replication factor is 3
  - Another research topic?
- 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.
- 1/3 of the replica on a node, 2/3 on a rack and 1/3 distributed evenly across remaining racks.

Replica Selection

- Replica selection for READ operation: HDFS tries to minimize the bandwidth consumption and latency.
- If there is a replica on the Reader node then that is preferred.
- HDFS cluster may span multiple data centers: replica in the local data center is preferred over the remote one.

Data Correctness

- Use Checksums to validate data
  - Use CRC32
- File Creation
  - Client computes checksum per 512 bytes
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas

Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next node in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

Data retrieval

- When a client wants to retrieve data
  - Communicates with the NameNode to determine which blocks make up a file and on which data nodes those blocks are stored
  - Then communicated directly with the data nodes to read the data

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Need to develop a real HA solution

Rebalancer

- Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool

Secondary NameNode

- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode
  - Transaction Log on NameNode is purged

Filesystem Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

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
- 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.

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 is NOT supported! Another research topic.

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.

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.

Application Programming Interface(API)

- 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.

User Interface

- Commads for HDFS User:
  - hadoop dfs -mkdir /foodir
  - hadoop dfs -cat /foodir/myfile.txt
  - hadoop dfs -rm /foodir/myfile.txt
- Commands for HDFS Administrator
  - hadoop dfsadmin -report
  - hadoop dfsadmin -decommision datanodename
- Web Interface
  - http://host:port/dfshealth.jsp

Summary of HDFS

- Highly fault-tolerant
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware

Outline

- Apache Hadoop
- Overview of MapReduce framework
- Distributed file systems (HDFS)
- Parallel file systems (vs Distributed)

Parallel File Systems

- Store application data persistently
  - Usually extremely large datasets that can’t fit in memory
- Provide global shared namespace (files, directories)
- Designed for parallelism
  - Concurrent (often coordinated) access from many clients
- Designed for high-performance
  - Operate over high-speed networks (IB,Myrinet,Portals)
  - Optimized I/O path for maximum bandwidth

Parallel vs. Distributed

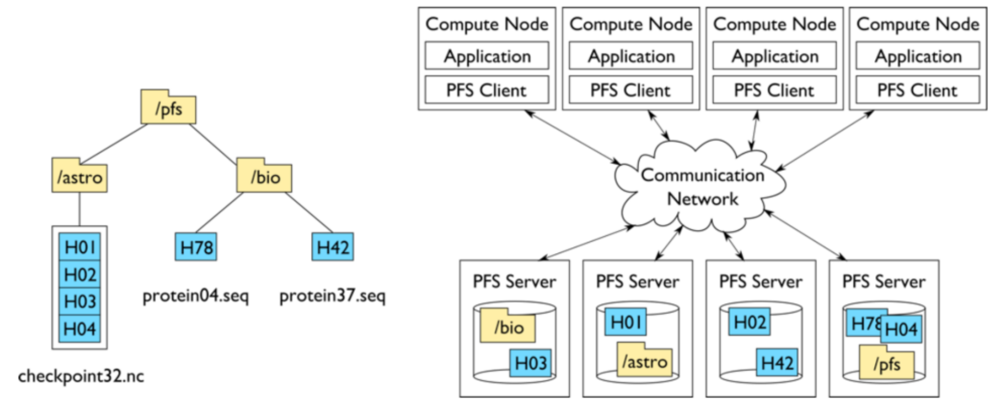
- Data distribution
  - DFSs often store entire objects (files) on a single storage node
  - PFSs distribute data across multiple storage nodes
- Symmetry
  - DFSs often run on architectures where the storage is co-located with the application (but not always, e.g., GoogleFS, Ceph)
  - PFSs are ofren run on architecures where storage is ohysically separated from the compute system (again not always true)
- Fault-tolerance
  - DFSs take on fault tolerance
  - PFSs run on enterprise shared storage

Parallel vs. Distributed(cont.)

- Workloads
  - DFSs are geared for loosely coupled, distributed applications (think data-intensive)
  - PFSs target HPC applications, which tend to perform highly coordinated I/O accesses, and have massive bandwidth requirements
- Overloaded terms:
  - GlusterFS, CephFS claim to be both
  - PVFS is often run in symmetric environments

Parallel File Systems

- Provide a directory tree all nodes can see (global name space)
- Map data across many servers and drives (parallelism of access)
- Coordinate access to data so certain access rules are followed (useful semantics)



When and why a PFS?

Main PFS advantages

- Throughput performance
- Scalability: Usable by 1000s of clients
- Lower management costs for huge capacity

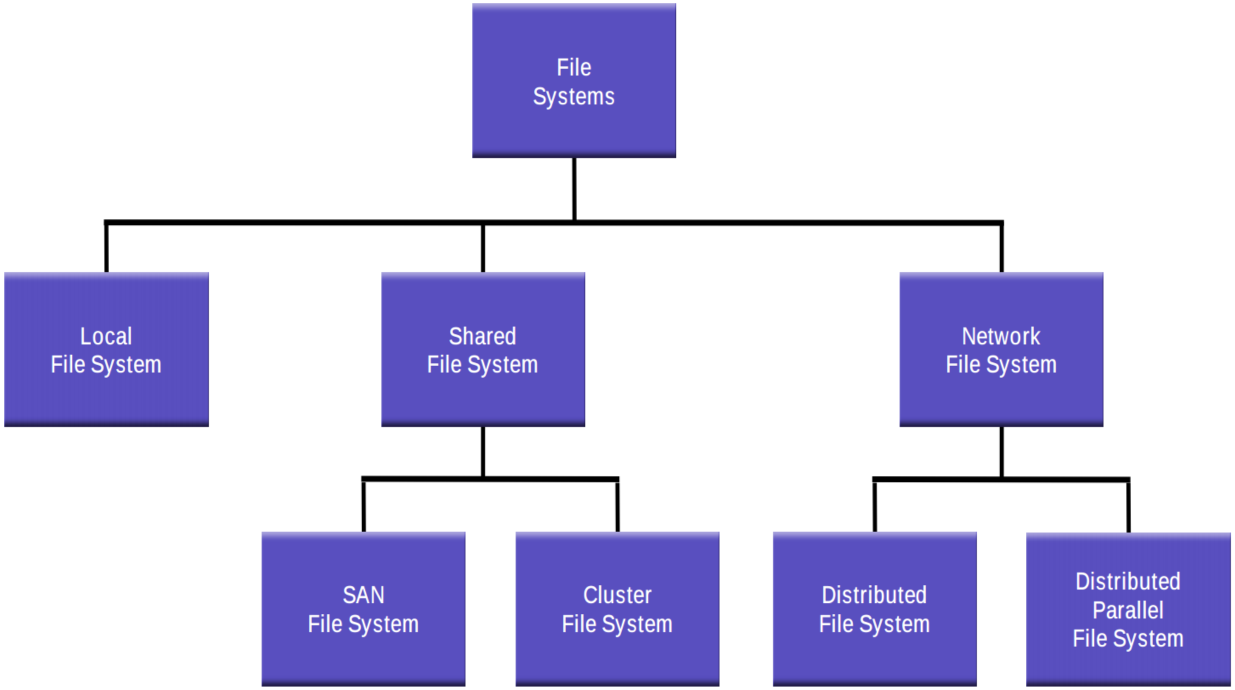
Main PFS disadvantages

- Metadata performance low compared to many separate file servers
- Complexity: Management requires skilled administrators
- Most PFS require adaption of clients for new Linux kernel versions

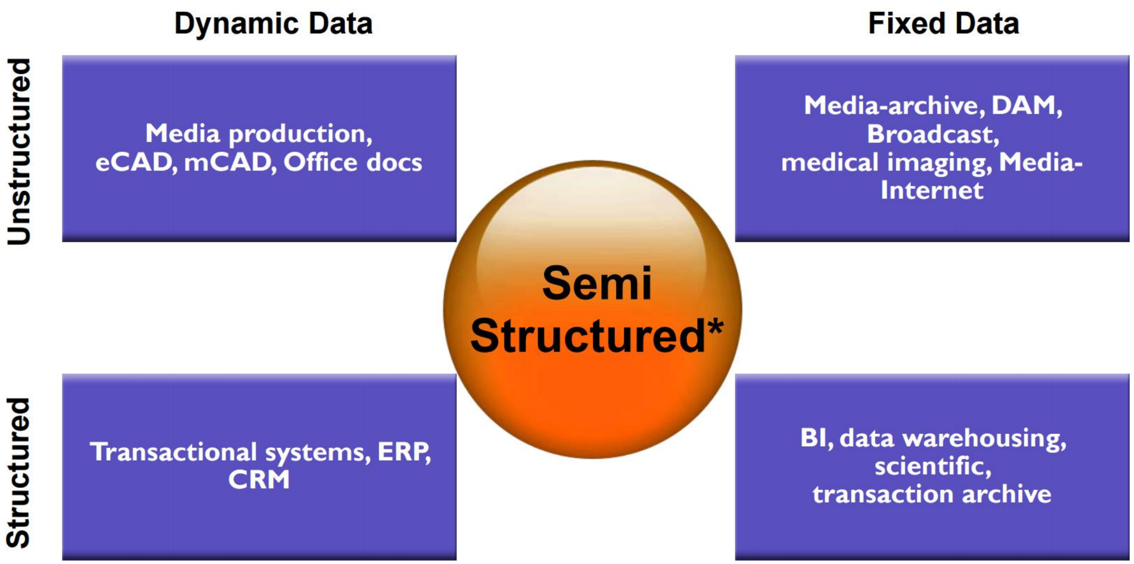
Which solution is better?

- This depends on the applications and on the system environment
- Price also depends on the quality and is hard to compare
  - e.g. huge price differences of NFS products
- If PFS is not required, distributed file system is much easier

File System Taxonomy



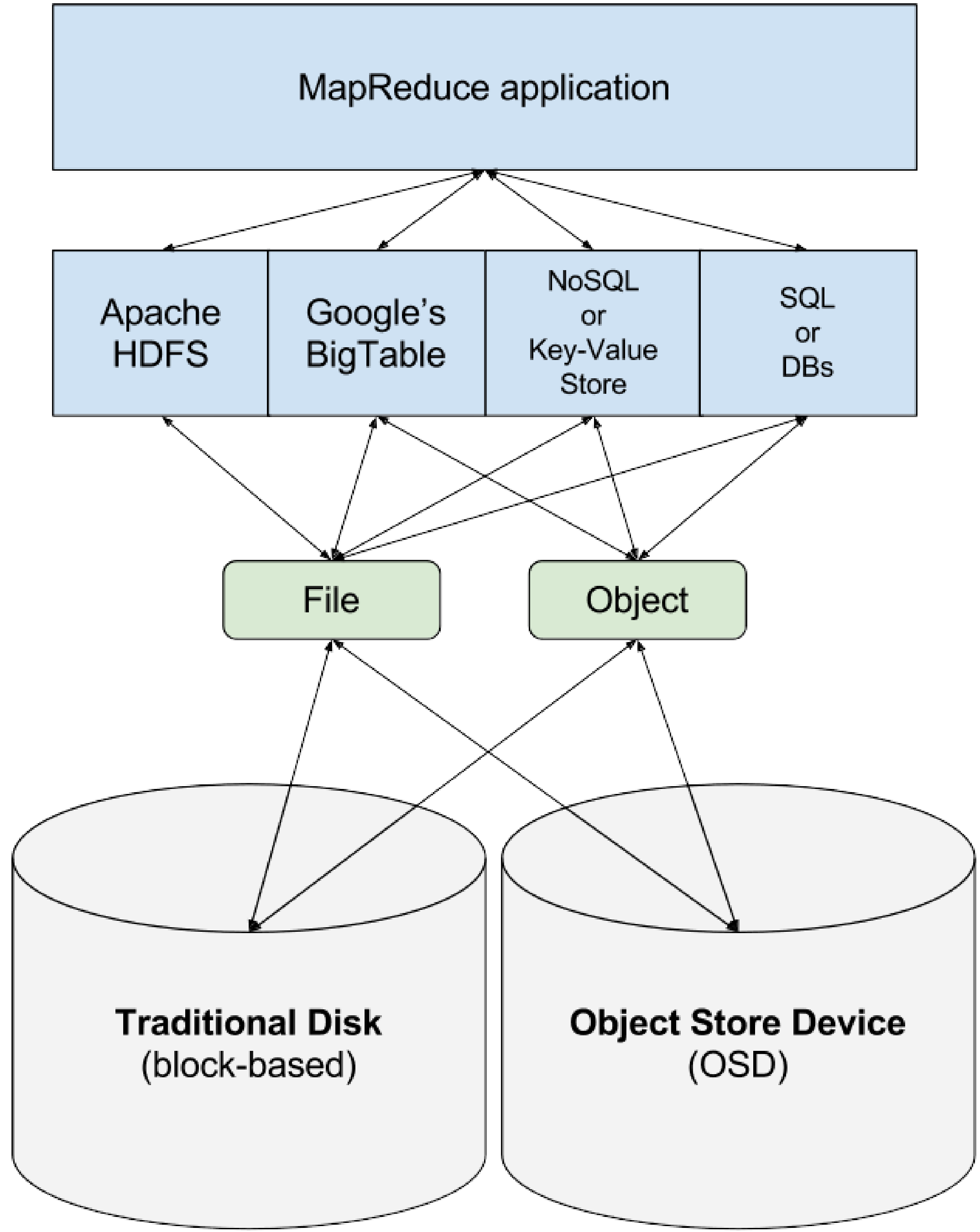
New reality of data segmentation



*Semi-Structured Data contains dynamic meta-data defined by users and/or applications

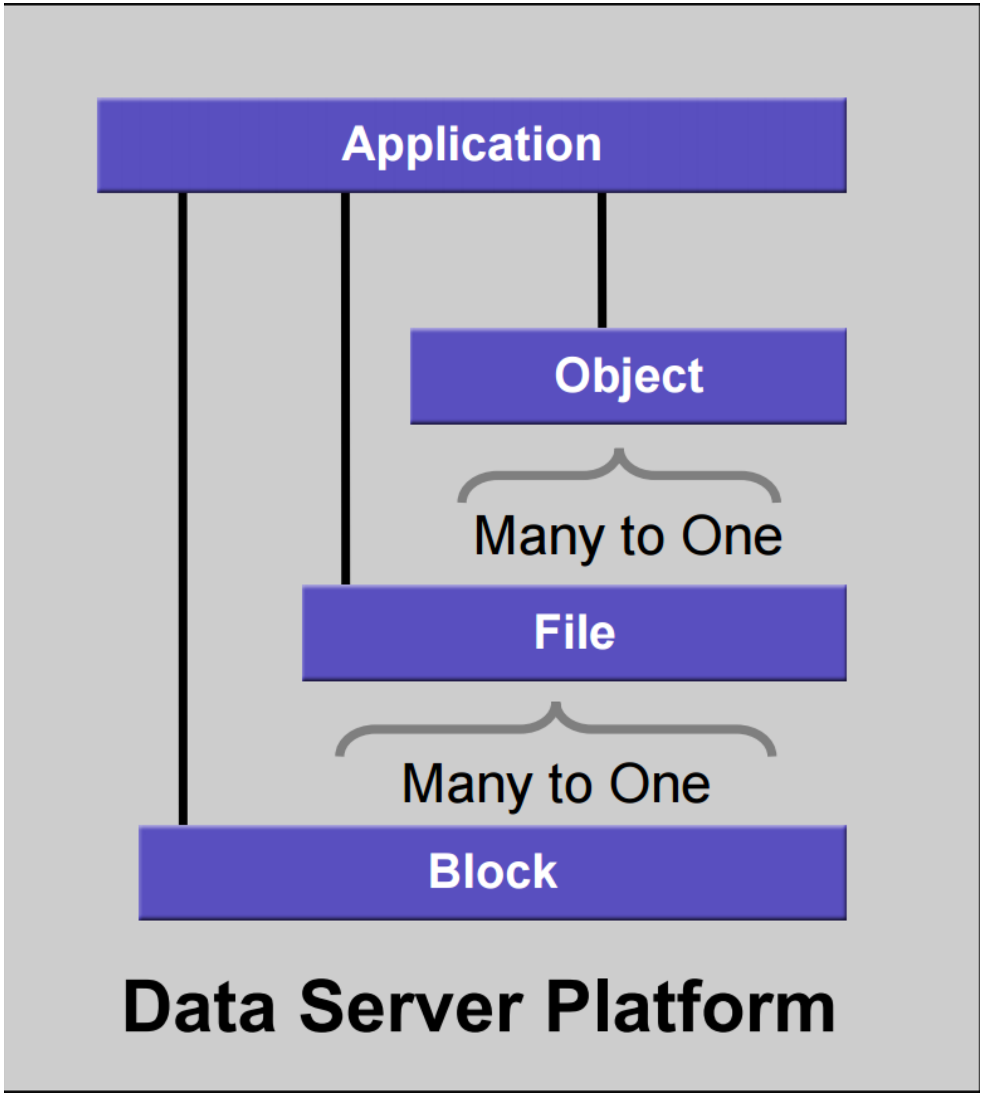
Data access taxonomy

11/5/2019



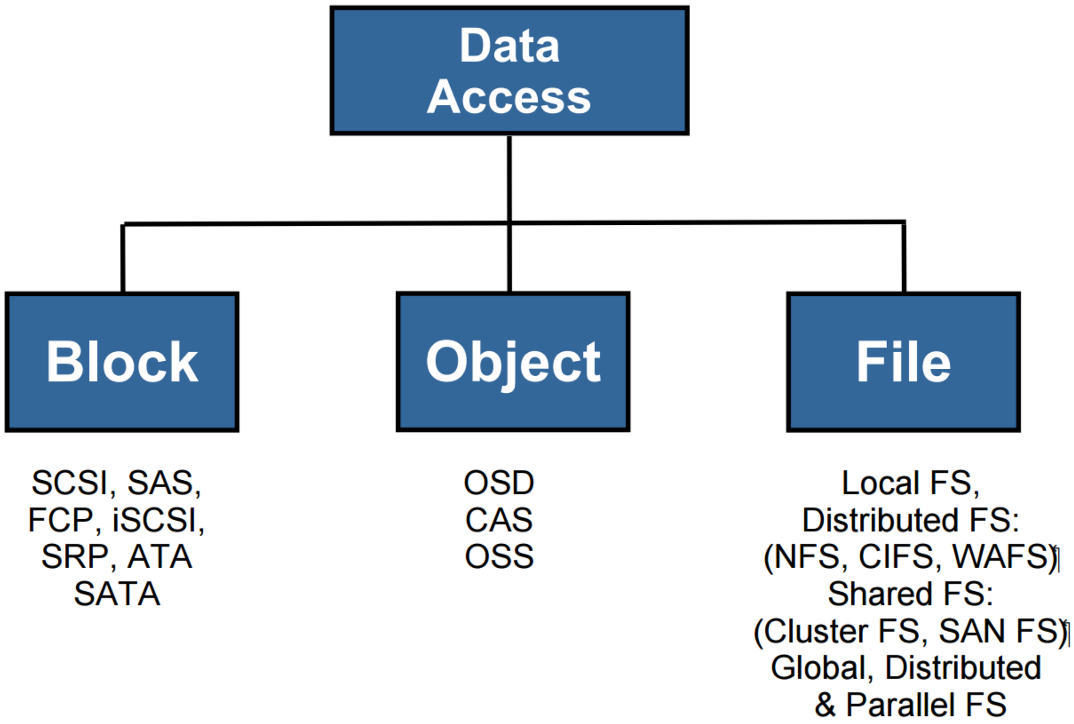
Data access taxonomy

- Application may interface with the storage subsystem in any of three layers:
  - **Block**: highest performance and very little metadata
  - **File**: high performance and some metadata
  - **Object**: medium performance and rich metadata



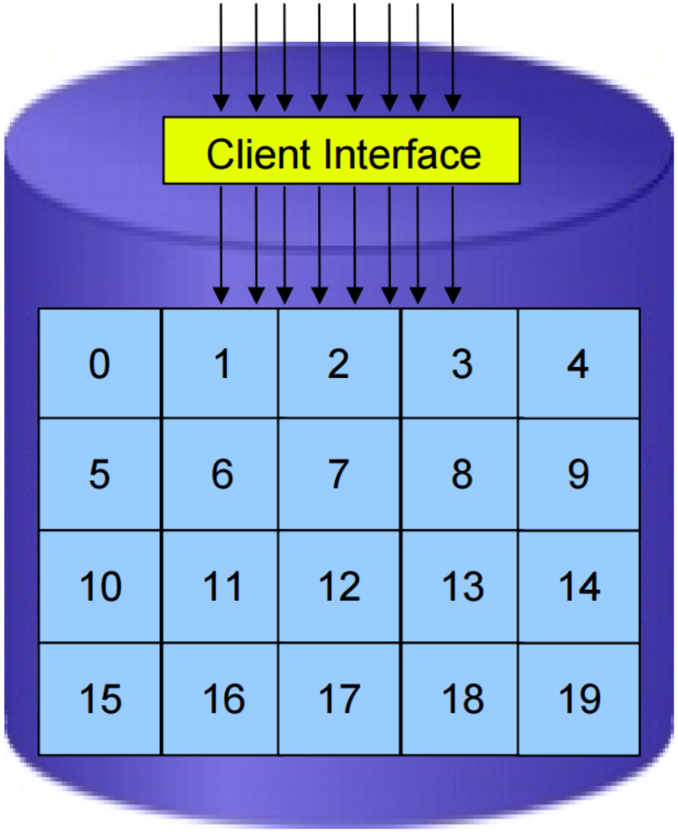
Data access taxonomy

11/5/2019



The Block Paradigm

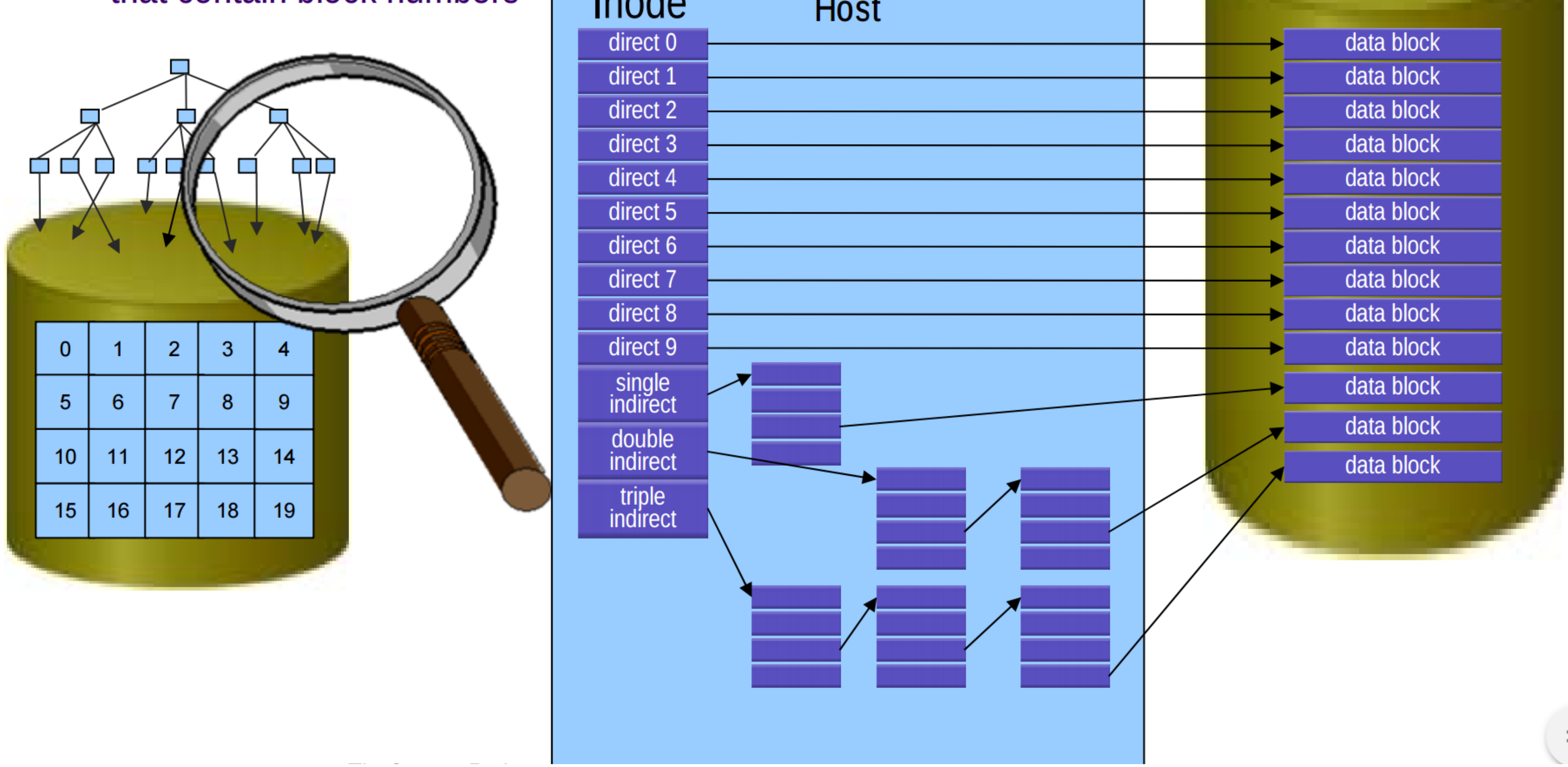
SCSI, SAS, FCP, SRP, iSCSI, ATA, SATA



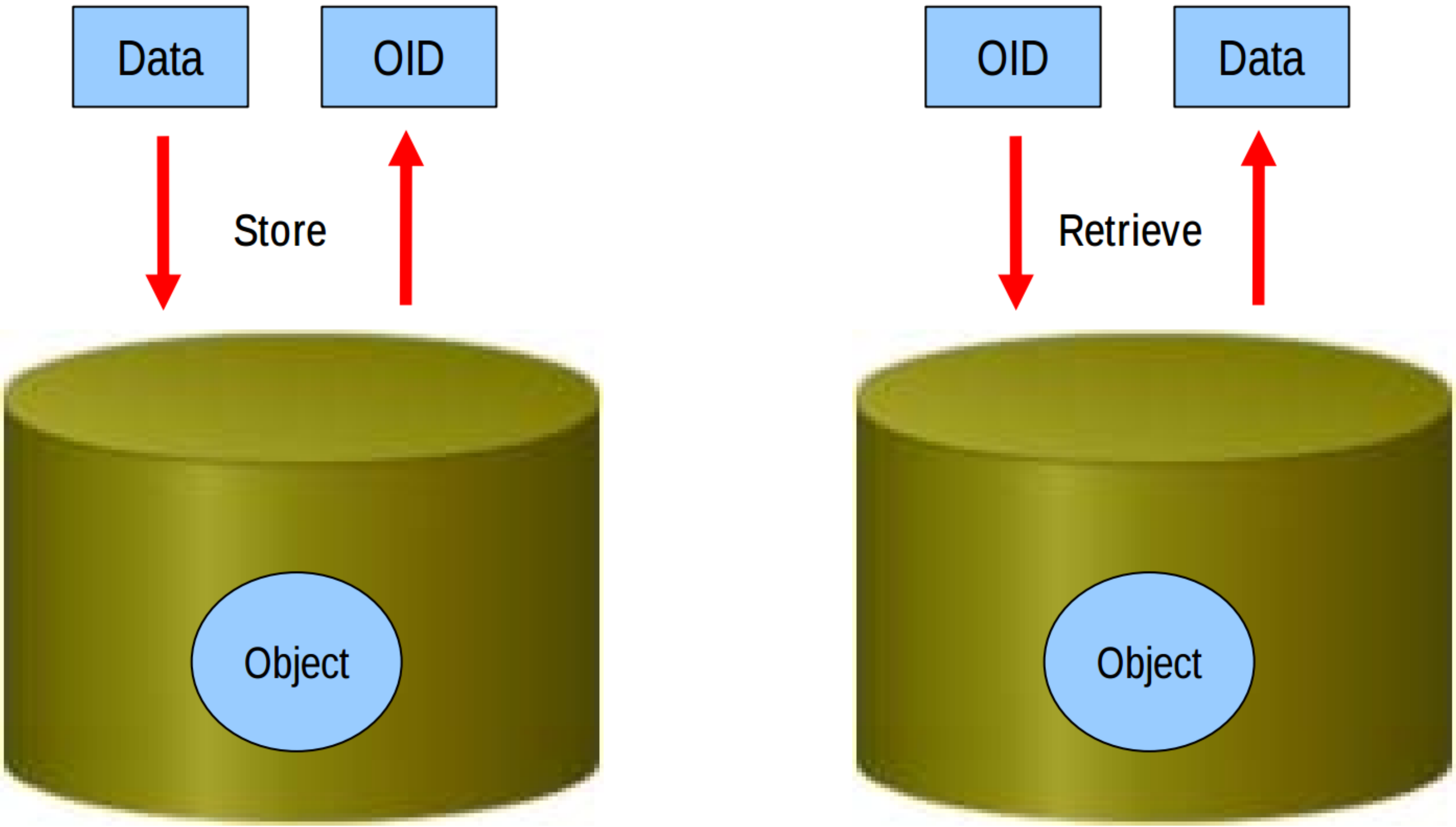
Physical Blocks:
e.g. 512 bytes

The File Paradigm

- The inode contains a few block numbers to ensure efficient access to small files. Access to larger files is provided via indirect blocks that contain block numbers



The Object Paradigm



Flat Namespace