Industrial Machine Learning on Hadoop and Spark

Maks Nakhodnov, Bremen 2025

Distributed computing. HDFS

Plan:

How to store big data?

- Structure of the Linux filesystem.
- Fault tolerance.
- HDFS: its structure and main properties.


hadoop

The "The Apache™ Hadoop™" project develops open-source software for fault-tolerant, scalable, and distributed computing.

Features:

- Works with Big Data on standard servers
- Strong open-source community
- Many different products and tools use Hadoop

- Started as a subproject in Apache Nutch
  – Nutch is an open-source web search engine
  – An open-source alternative to Google
  – Started by Doug Cutting

- In 2004, Google published papers on GFS and MapReduce

- Doug Cutting and the Nutch team implemented their framework based on these papers

- In 2006, Yahoo! hired Doug Cutting to work on Hadoop in their team

- In 2008, Hadoop became an Apache Top Level Project
  – http://hadoop.apache.org

Hadoop users

Data storage

Disk capacity has grown exponentially, unlike read speed:

— 1990

- Capacity: 1,400 MB
- Read speed: 4.5 MB/s
- Reading the entire disk takes ~5 minutes

— 2010

- Capacity: 1 TB
- Read speed: 100 MB/s
- Reading the entire disk takes ~3 hours

Hadoop

— 100 HDDs can simultaneously read 1 TB of data in 2 minutes

Hadoop Cluster

- “Cheap” standard hardware:
  – Not supercomputers
  – Not desktops

- Connected via a network

- Located in one place:
  – Servers in racks within a data center

Hadoop System Principles

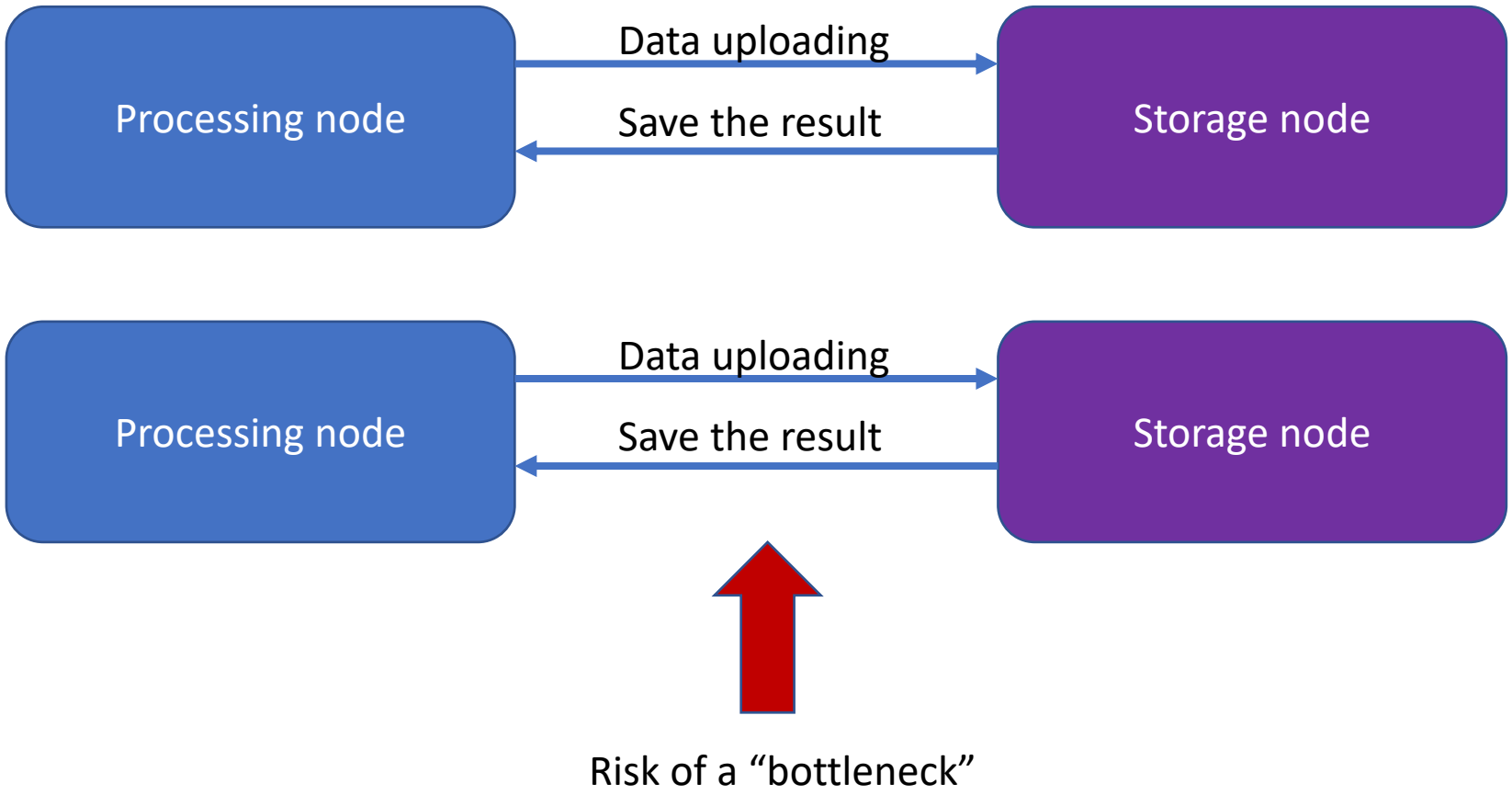
- Horizontal (Scale-Out) scaling instead of vertical (Scale-Up)
- Move code to the data
- Handle node failures and hardware faults
- Encapsulate the complexity of distributed and multithreaded applications

Scaling

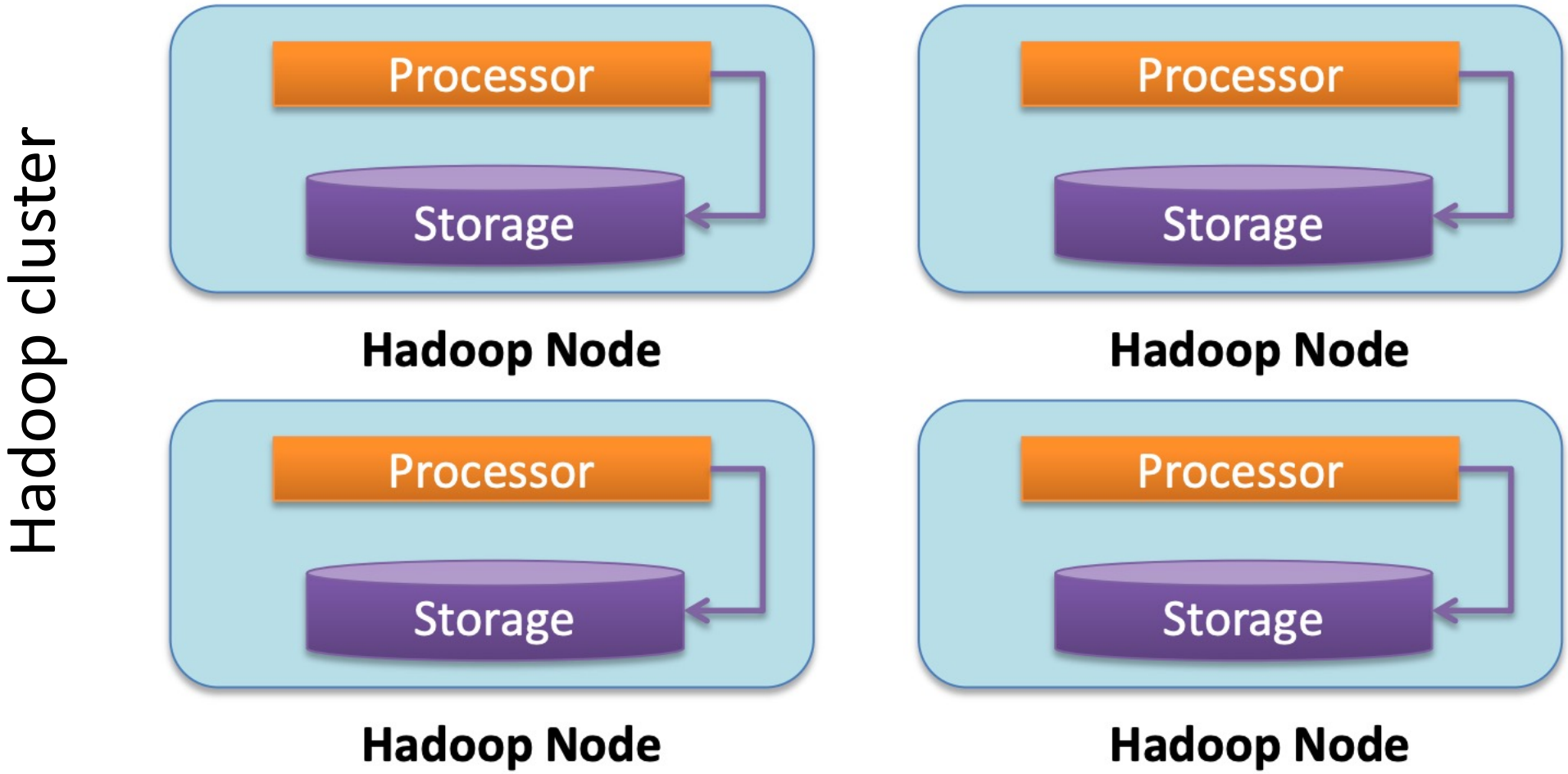
- Vertical scaling:
  – Add additional resources to existing hardware (CPU, RAM)
  – If hardware cannot be upgraded, buy a more powerful new one
  – Moore’s Law can’t keep up with data growth

- Horizontal scaling:
  – Add more machines to the existing cluster
  – Applications support adding/removing servers
  – Easy to scale “down” as well

Data to code


Processing node
Data uploading
Save the result
Storage node
Processing node
Data uploading
Save the result
Storage node
Risk of a “bottleneck”

Code to data


Hadoop cluster
Processor
Storage
Hadoop Node
Processor
Storage
Hadoop Node
Processor
Storage
Hadoop Node
Processor
Storage
Hadoop Node

Hardware Failures

- The more machines, the more frequent hardware failures will occur

- Hadoop is designed with hardware failures in mind:
  – Data replication
  – Task restart

Encapsulation of Implementation Complexity

- Hadoop hides many complexities of distributed and multithreaded systems
- Frees developers from worrying about system-level issues:
  – Race conditions, data waiting
  – Organizing data transfer, data distribution, code delivery, etc.
- Allows developers to focus on application development and implementing business logic

Hadoop Ecosystem

- Main Hadoop components:
  – HDFS: Hadoop Distributed FileSystem
  – MapReduce: Framework for distributed data processing

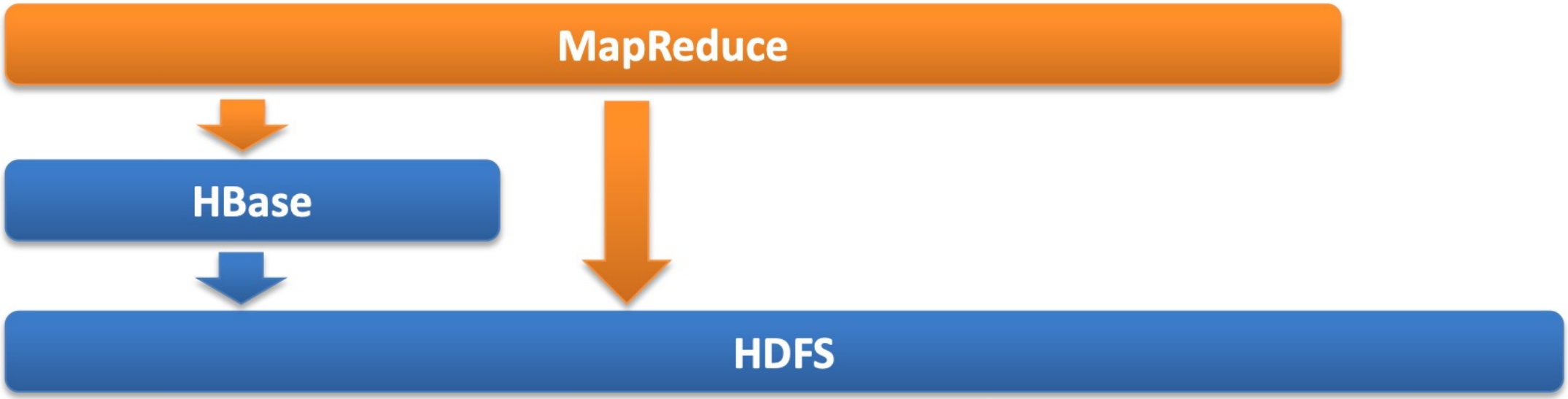
- Other components:
  – HBase: Column-oriented DB, supports sequential and random reads, and simple queries
  – Zookeeper: Highly-available coordination service
  – Oozie: Workflow scheduler for Hadoop
  – Pig: Data processing language and runtime
  – Hive: Data warehouse with a SQL interface

Hadoop Ecosystem

- To develop an application, a filesystem is needed:
  – In Linux: ext3 and ext4
  – In the Hadoop world: usually Hadoop Distributed File System (**HDFS**)
- A convenient interface for working with data is also needed:
  – a Relational DBMS on top of the local filesystem
  – **HBase:** key/value store implemented on top of HDFS

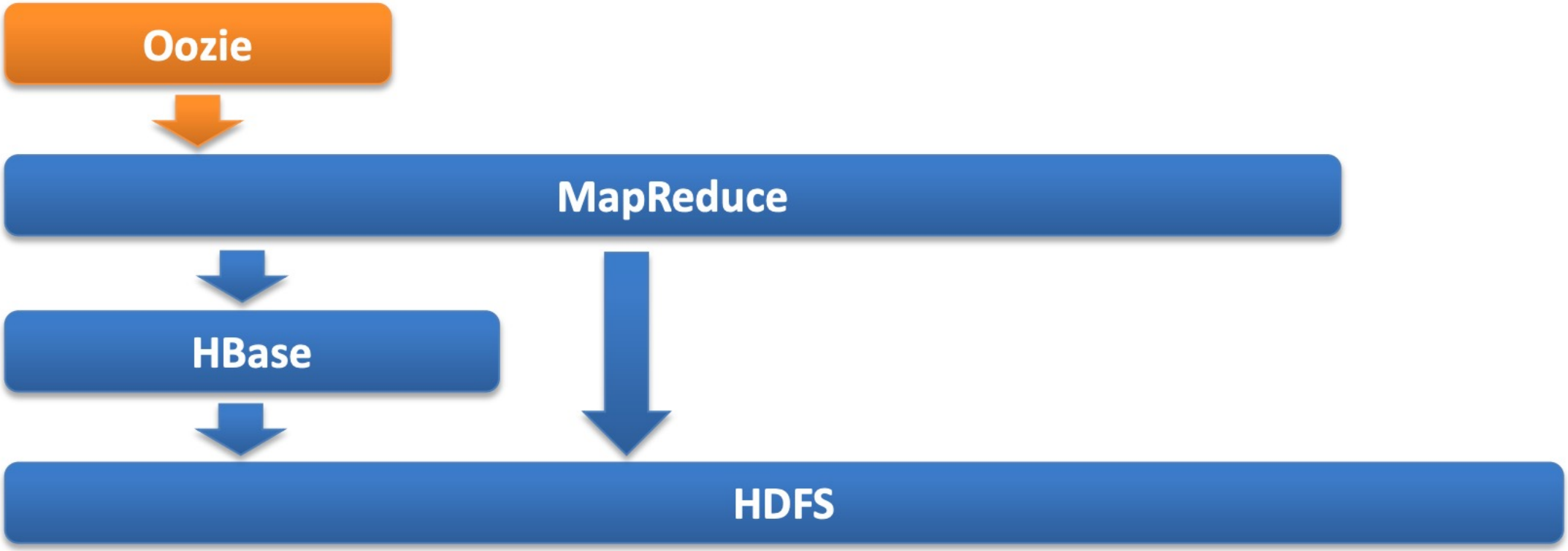
Hadoop Ecosystem

- Framework for running MapReduce tasks



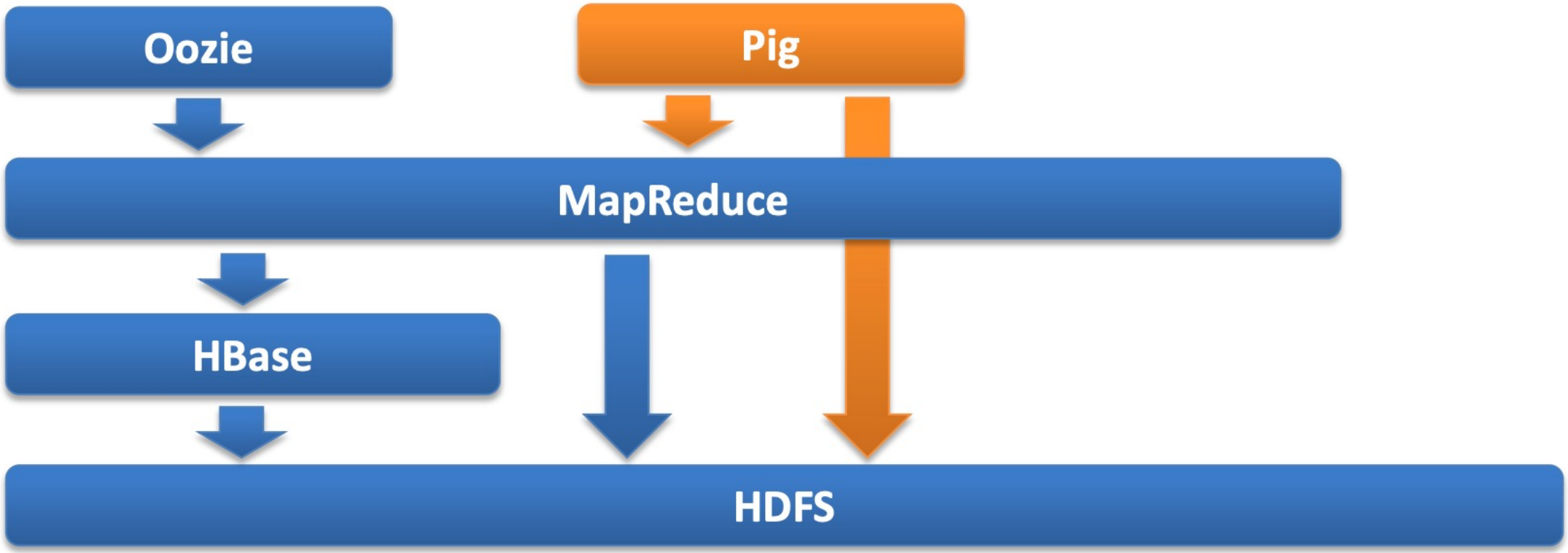
Hadoop Ecosystem

Apache Oozie: a popular tool for coordinating workflows of MapReduce tasks



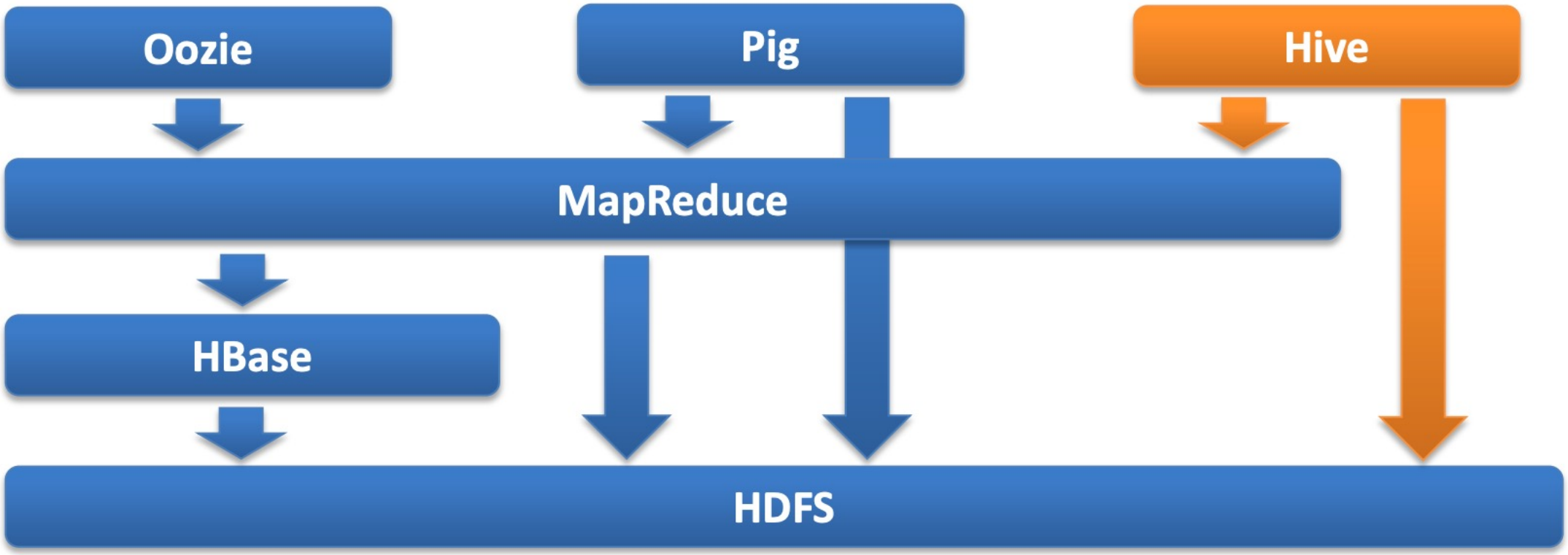
Hadoop Ecosystem

Apache Pig: a tool for data processing using high-level commands



Hadoop Ecosystem

Apache Hive: data processing using SQL-like queries



Hadoop installation

Quite difficult to install from scratch. Using distributives is highly advised:

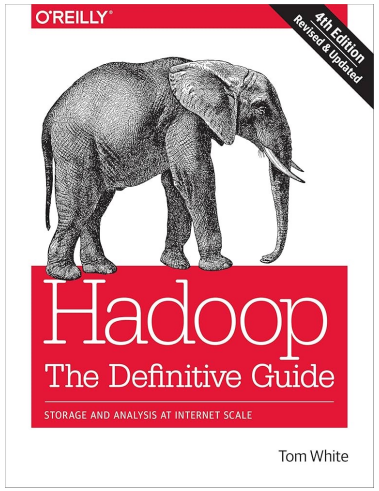
- Cloudera Distribution for Hadoop (CDH)
- MapR Distribution
- Hortonworks Data Platform (HDP)
- Apache BigTop Distribution
- Greenplum HD Data Computing Appliance

Resources

- **Apache Hadoop Documentation**
  – http://hadoop.apache.org

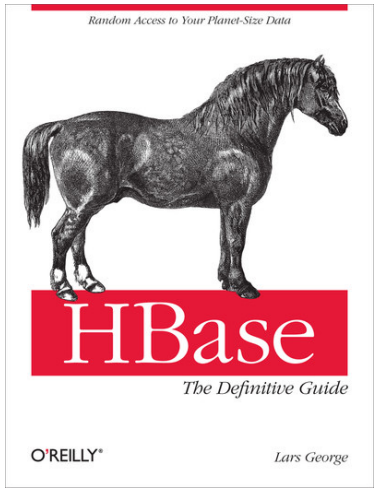
- Each individual product has its own documentation

- Each Hadoop vendor provides its own documentation
  – https://ccp.cloudera.com/display/DOC/Documentation

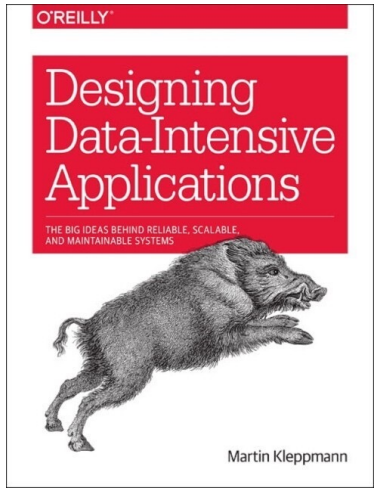


Hadoop: The Definitive Guide Tom White (Author) O'Reilly Media; 3rd Edition

MapReduce Design Patterns Donald Miner, Adam Shook (Authors) O'Reilly Media



HBase: The Definitive Guide Lars George (Author) O'Reilly Media; 1 edition



Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Martin Kleppmann

HDFS: its architecture and key properties

Google's Filesystem GFS

research.google.com/archive/gfs-sosp2003.pdf

Main tasks of HDFS

1. Fault tolerance in conditions of frequent failures and breakdowns
2. Parallel processing of data segments

Goals:

- Hardware failure is more the norm than the exception.
- Applications running on HDFS deal with large datasets. A typical HDFS file ranges from gigabytes to terabytes.
- Write once, read many (WORM)
- "Moving computation is cheaper than moving data" (Data locality)
- HDFS is designed to be easily portable across different platforms.

Main Principles: Fault Tolerant

- Runs on a server cluster
- Appears to the user as “one large disk”
- Operates on top of standard filesystems (Ext3, Ext4, XFS)
- Data is not lost if disks or servers fail

Main Principles: Uses standard “hardware”

“Cheap” server equipment:

- **Not** supercomputers!
- **Not** desktops!
- **Yes**, ordinary (unreliable) servers!

Main Principles: HDFS is good for…

Storing large files

- Terabytes, petabytes…
- Millions (but not billions) of files
- Files sized from 100 MB and up

Data streaming

- “Write once / read-many times” pattern
- Optimized for sequential reading: no random read operations

Standard servers

- Less reliable than supercomputers

Main Principles: HDFS is bad for…

Low-latency reads

- High throughput rather than fast data access
- HBase helps address this requirement

A large number of small files

- Better to have a million large files than a billion tiny ones
- For example, 1,000 files of 1 GB each is better than 100,000 files of 10 MB each

Multithreaded writing

- Only one process can write to a file
- Data is appended to the end of the file
- No support for writing at a specific offset

Useful definitions

Cluster – a collection of computers connected by a network and performing tasks sent by clients.

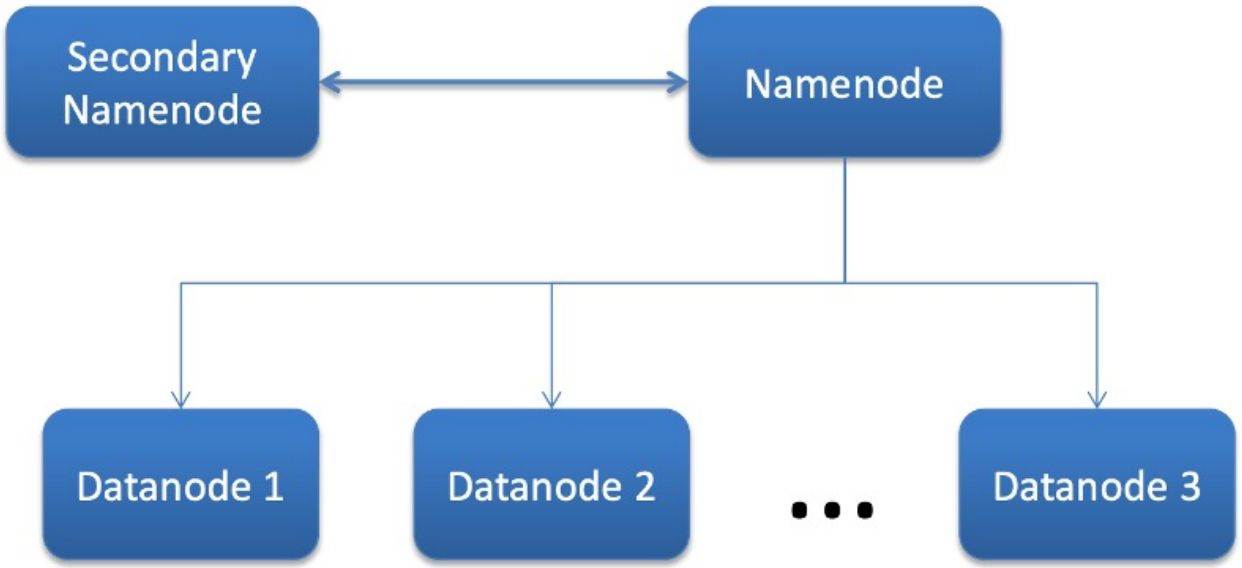
Node – one of the computers connected to the cluster.

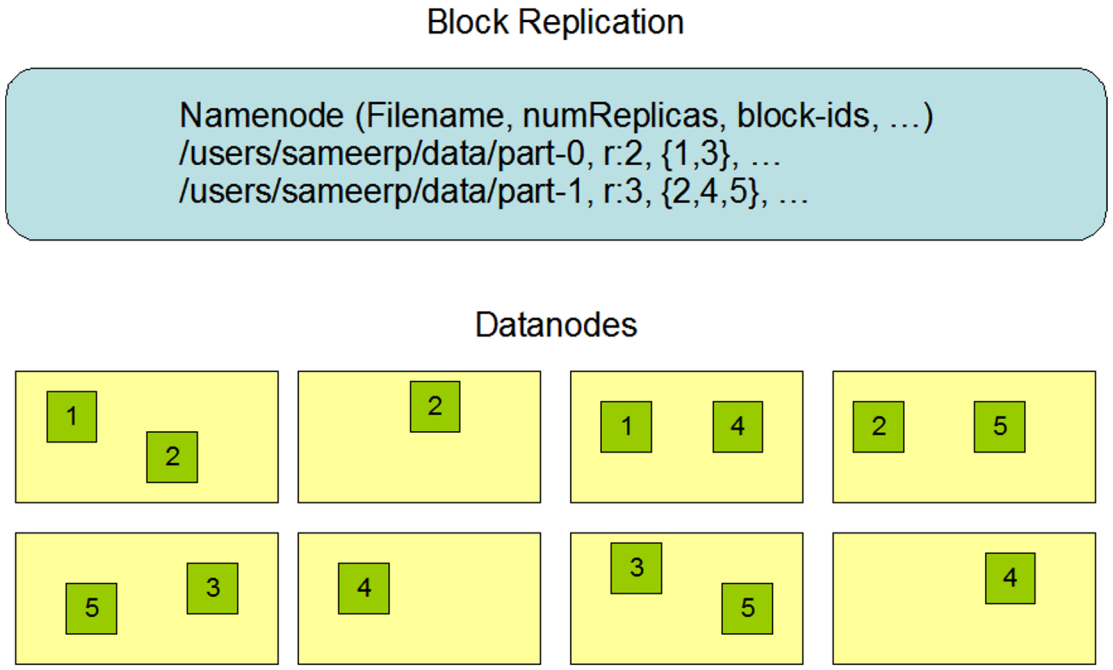
Rack – a group of several nodes connected by a network.

Daemon – a program in UNIX-like systems that is started by the system and runs in the background without direct user interaction.

Client – a hardware or software component of a computing system that sends requests to a server.

HDFS Daemons





NameNode

Responsible for:

- File namespace
- Metadata
- File block locations
- Runs on a single (dedicated) machine

NameNode

FSImage:

- Snapshot of the entire HDFS filesystem at a point in time
- Stored on the NameNode; contains metadata: files, directories, access permissions

Edit Logs:

- Log of filesystem changes (adding, deleting, renaming files)
- Applied to FSImage at NameNode startup to restore the current state

Under the hood:

- At NameNode startup: FSImage + all Edit Logs → builds the current filesystem tree
- Secondary NameNode / Checkpoint Node: periodically merges FSImage + Edit Logs → creates a new FSImage, reducing log size

DataNode

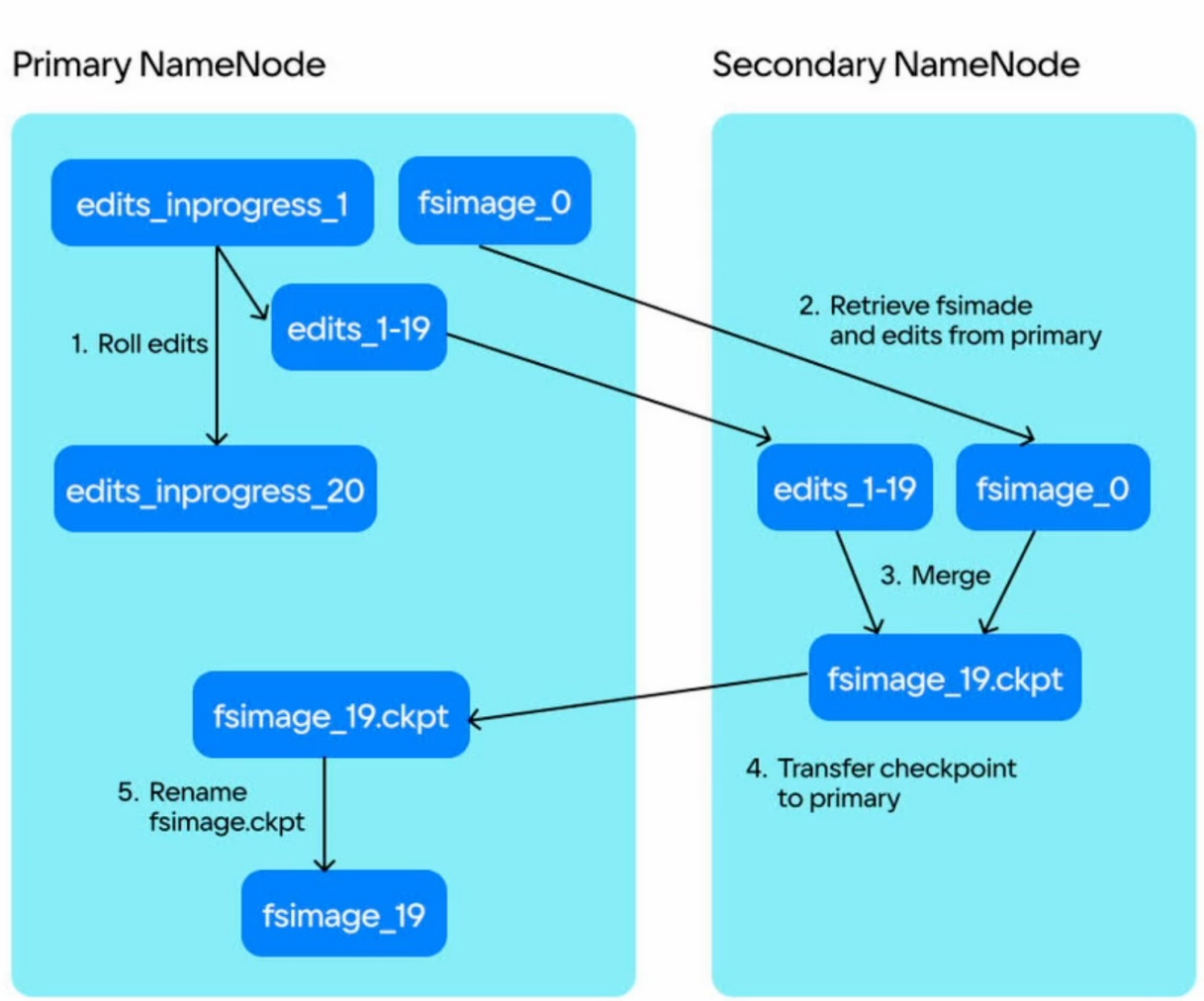
Stores and serves data blocks

- Sends status reports to the NameNode
- Runs on every machine in the cluster

Secondary NameNode

Periodically updates the fsimage

- Requires the same hardware as the NameNode
- (!) Not used for high availability; i.e., it is not a backup for the NameNode



Files and blocks

- Files in HDFS consist of blocks
  - A block is a unit of data storage

- Managed by the NameNode
- Stored on DataNodes

- Blocks are replicated across machines during writing
  - The same block is stored on multiple DataNodes
  - Default replication factor is 3
  - Ensures fault tolerance and simplifies data access

HDFS Blocks

- Standard block size: 64 MB or 128 MB
- The main reason is to reduce the cost of seek time compared to data transfer speed: *Time to transfer > Time to seek*

Let

- seek time = 10 ms
- transfer rate = 100 MB/s

What should be the block size so that the seek time is only 1% of the transfer rate?

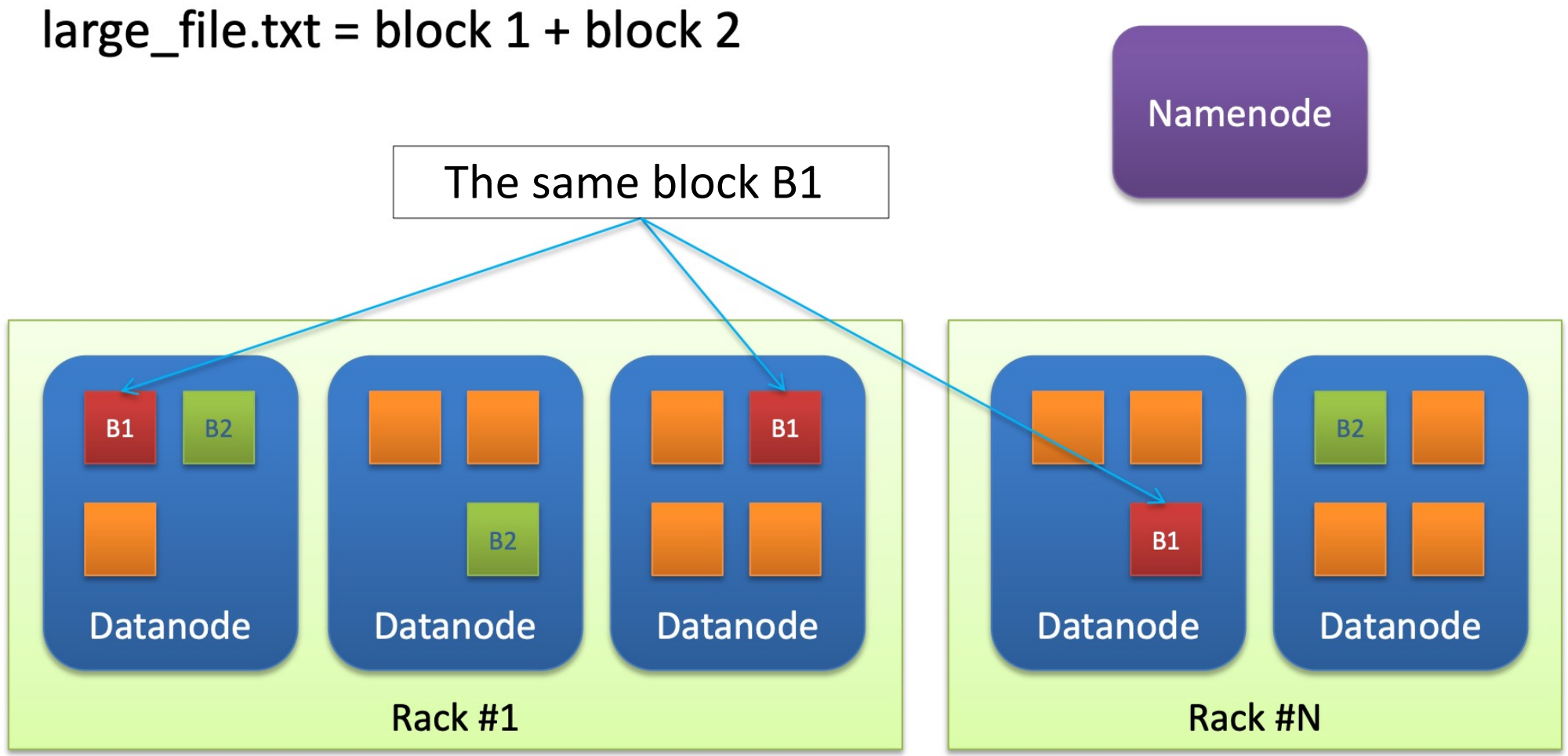
Placing & Replication

- The NameNode determines where to place blocks
- Balances reliability and performance
  – Attempts to reduce network load (bandwidth)
  – Attempts to improve reliability by placing replicas on different racks

Default replication factor is 3:

- 1st replica on the local machine
- 2nd replica on another machine in the same rack
- 3rd replica on a machine in a different rack

Replication



Replication. Erasure coding (EC)

- Alternative to replication for data protection
- Data is split into **k data blocks + m parity blocks**
- Can recover data if up to m blocks are lost
- Saves space compared to 3x replication
- Suitable for **cold storage** of large data volumes

	Data Durability	Storage Efficiency
Single replica	0	100%
Three-way replication	2	33%
XOR with six data cells	1	86%
RS(6,3)	3	67%
RS(10,4)	4	71%

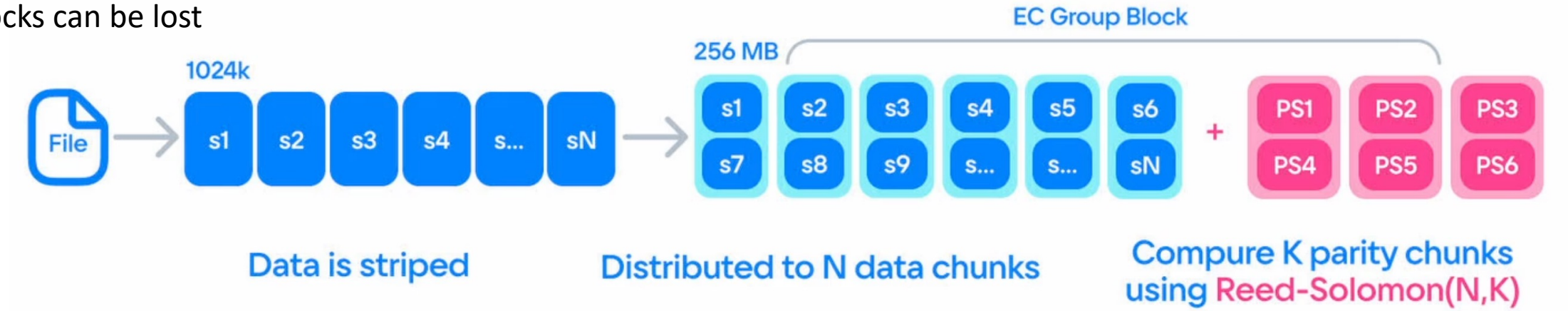
Example: 600MB file, block size 100MB

3x replication:

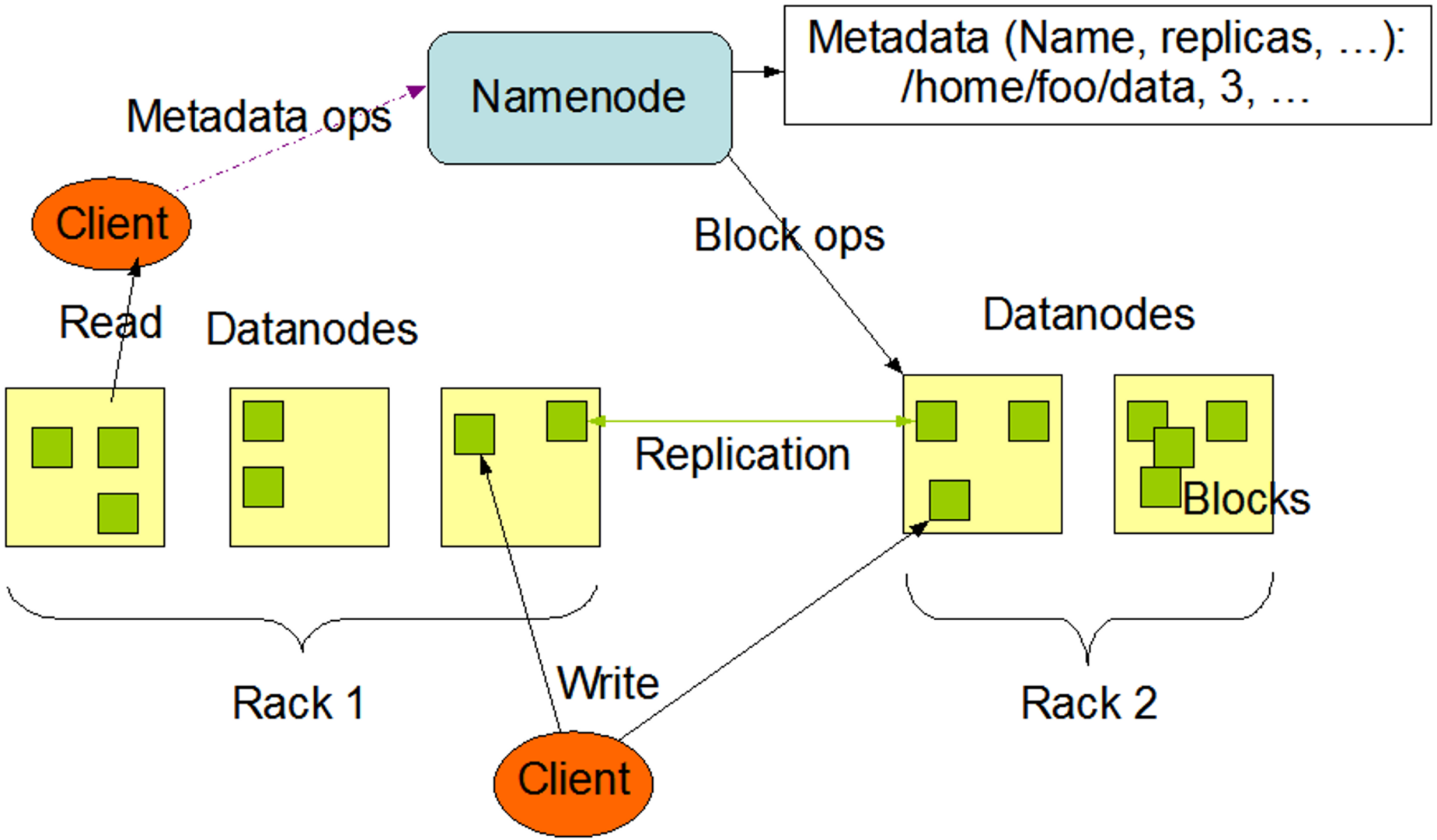
6 blocks × 3 = 18 blocks (1800 MB)

EC(6,3):

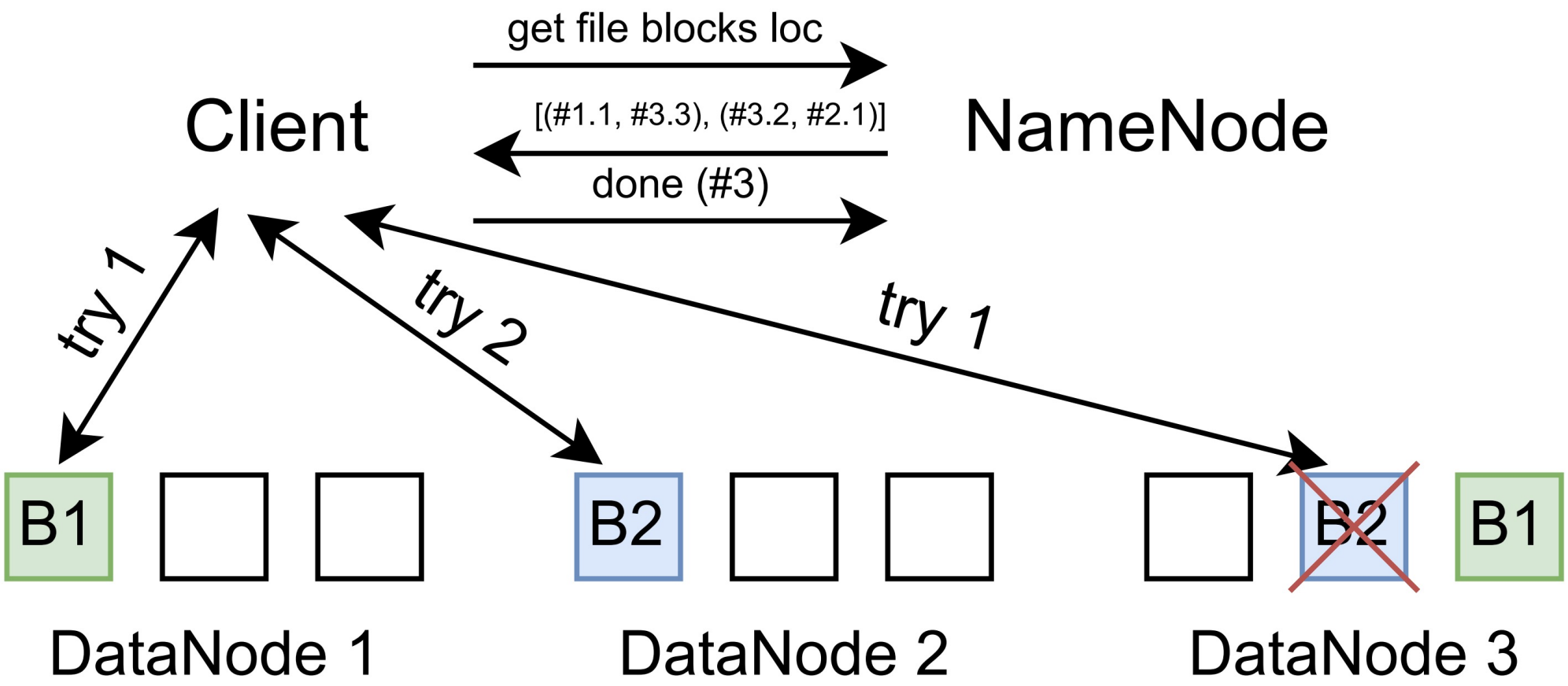
6 blocks + 3 parity = 9 blocks (900 MB); up to 3 blocks can be lost



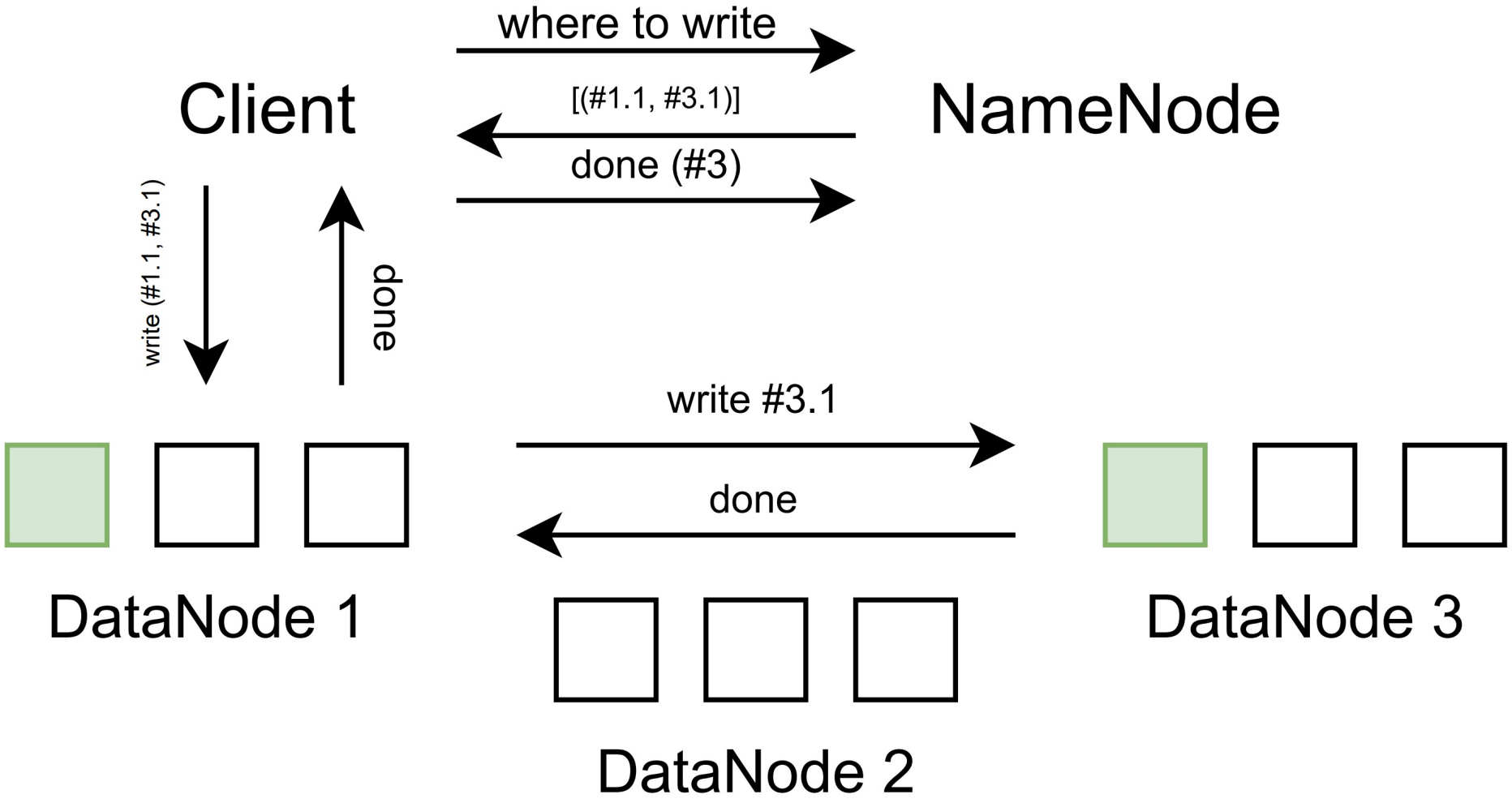
Client-server interaction



Client-server interaction. Read



Client-server interaction. Write



NameNode: memory utilization

For fast access, all block metadata is stored in the NameNode's RAM:

- The larger the cluster, the more RAM is required
- Better to have millions of large files than billions of small ones
- Operates on clusters with hundreds of machines

NameNode Federation (*Hadoop 2+*):

- Each NameNode manages a portion of the blocks
- Horizontal scaling of NameNodes
- Supports clusters with thousands of machines
- Details: http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/Federation.html

NameNode: memory utilization

How does block size affect the maximum filesystem size?

Larger block size → fewer blocks
Fewer blocks → more files in the filesystem
More blocks → more RAM (~1 GB heap for 1M blocks)

- Let's say we have 200 TB = 209,715,200 MB
- With a block size of 64 MB:
  209,715,200 MB / 64 MB = 3,276,800 blocks

- With a block size of 128 MB:
  209,715,200 MB / 128 MB = 1,638,400 blocks

NameNode: Fault tolerance

If the NameNode fails, HDFS does not work:

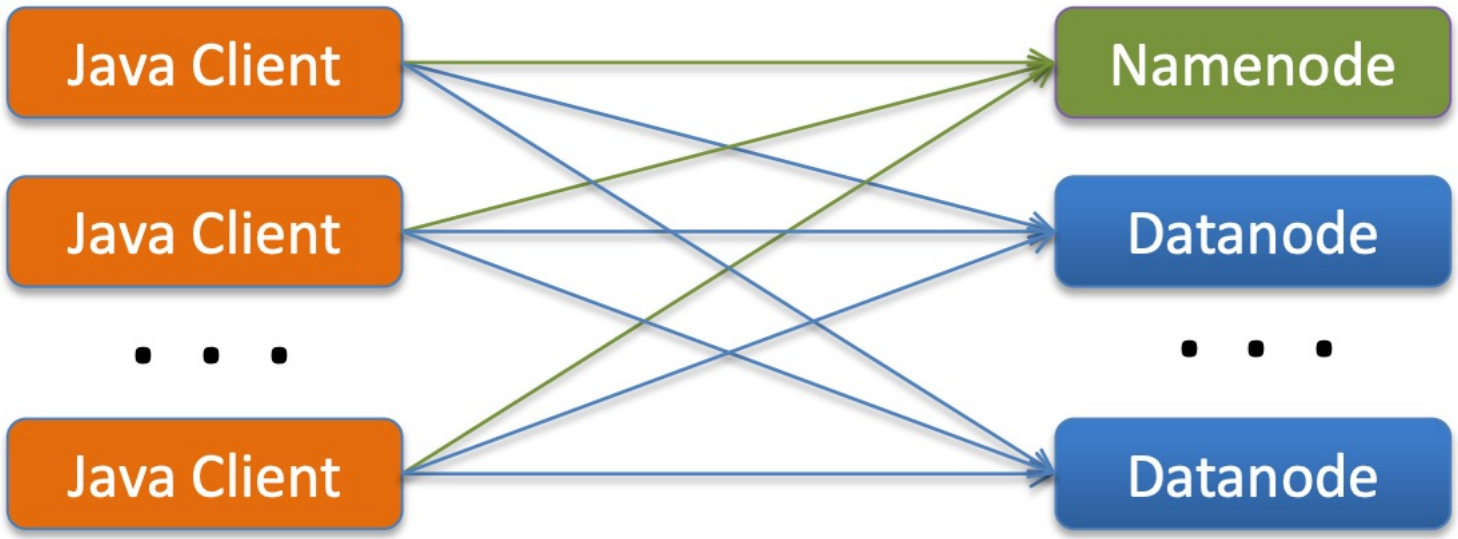
- NameNode is a single point of failure
- Should run on a separate, reliable machine
- Usually, this is not a problem

High Availability NameNode (*Hadoop 2+*):

- An Active-Standby process is always running and takes over management if the NameNode fails
- More details here: http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailability.html

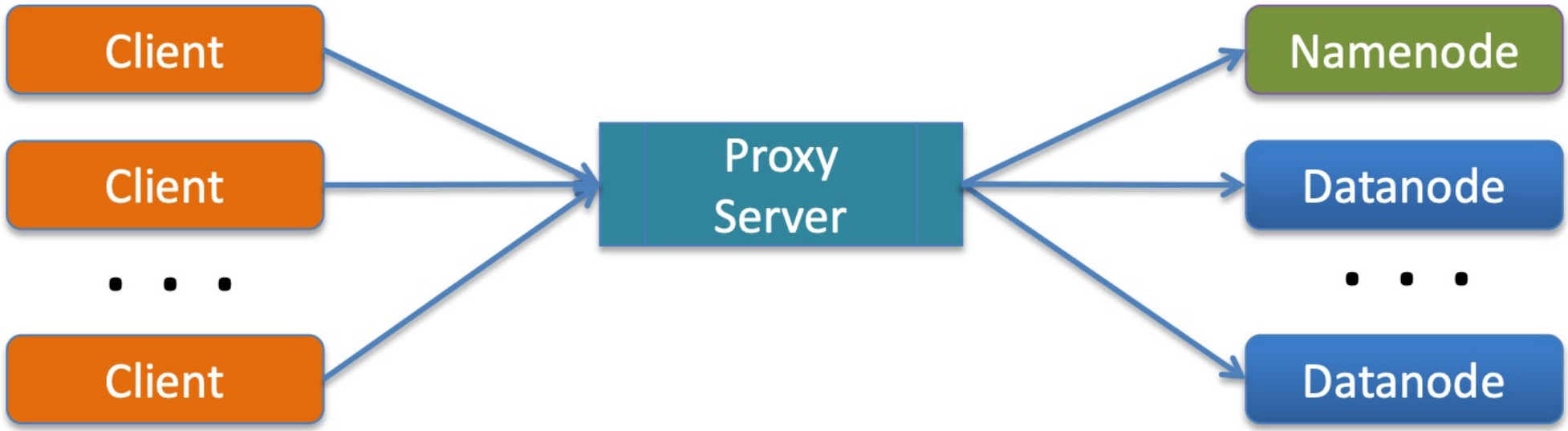
HDFS access — Direct access

- API for Java and C++
- Client requests metadata from the NameNode
- Client directly requests data from the DataNode
- Used for MapReduce



HDFS access — Proxy Server

- Client works through an external Proxy Server
- Several servers are included with Hadoop:
  - Thrift – interface definition language
  - WebHDFS REST – responses in JSON, XML, or ProtoBuf
  - Avro – serialization framework
- Python bindings: !pip install hdfs



Key features

- Large block size
- Designed for inexpensive, and therefore not the most reliable, servers
- Cluster-level replication
- Replication occurs asynchronously
- Clients can read and write HDFS files directly via the Java API
- Files are written once
- WORM principle (Write-once, read-many)
- Data compression and efficient use of disk space
- Self-diagnosis
- All NameNode metadata is stored in RAM

HDFS vs S3

Criterion	HDFS	S3
Architecture	Filesystem, NameNode + DataNode	Object storage
Data locality	Data locality for MapReduce/Spark	No data locality, compute is separate
Scaling	Limited by NameNode metadata	Practically unlimited
Cost	Requires its own clusters	Pay for storage and requests
Speed	Low latency inside the cluster	Higher latency, depends on the network

HDFS Shell

HDFS Shell

Basic command structure:

$hdfs dfs -<command> -<option> <URI>

Ex.: $hdfs dfs –ls /

HDFS Shell. URI

hdfs://namenode:9000/user/home

Scheme Authority HDFS path

- Local:

  $hdfs dfs -ls file:///to/path/file3

- HDFS:

  $hdfs dfs -ls hdfs://localhost/to/path/dir

- fs.default.name=hdfs://localhost

  $hdfs dfs -ls /to/path/dir

HDFS Shell. Commands

Bash-like commands:

cat, rm, ls, du, mkdir, cat, tail, …

Specific commands:

setrep — to set replication factor:

hdfs dfs -setrep <rep_factor> <path>

help — get reference:

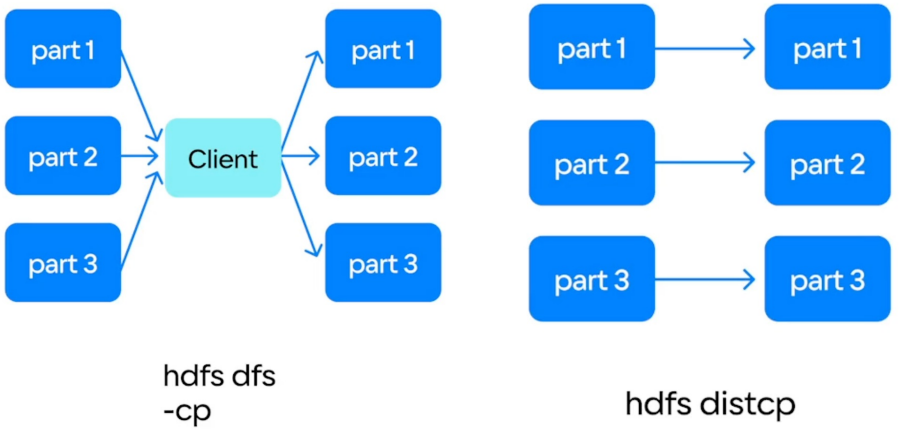
hdfs dfs -help

hdfs dfs -help <command_name>

text — like cat but uncompress archives

cp — to copy small files

distcp — to copy large files or many files



HDFS Shell. Data transfer

- mv – move a file from one place to another:
  hdfs dfs -mv /dir/file1 /dir2

- put (copyFromLocal) – copy a local file into HDFS:
  hdfs dfs -put localfile /dir/file

- get (copyToLocal) – copy a file from HDFS into the local FS:
  hdfs dfs -get /dir/file localfile

HDFS Shell. Permissions in HDFS

- Restrictions at the file/directory level
  – Similar to the POSIX permission model
  – Read (r), Write (w), and Execute (x)
  – Divided into: user, group, and others

- User rights are determined based on the rights of the OS where the client application is running

- Authorization is performed via Kerberos

```
root@namenode:/# hdfs dfs -ls /
Found 6 items
drwxrwxrwt   - root root                0 2025-09-07 14:17 /app-logs
drwxr-xr-x   - root supergroup          0 2025-09-07 14:14 /data
drwxr-xr-x   - root supergroup          0 2025-09-07 14:11 /rmstate
drwxrwxr-x   - root supergroup          0 2025-09-07 14:17 /tmp
drwxr-xr-x   - root supergroup          0 2025-09-07 14:11 /user
drwxr-xr-x   - root supergroup          0 2025-09-07 14:17 /wordcount_streaming
```

HDFS Shell. fsck

- Checks the filesystem for inconsistencies
- Shows issues such as:
  - Missing blocks
  - Under-replicated blocks
- Does **not** fix problems, only reports them
- NameNode will try to repair issues automatically
- Examples:

  Check a specific path $hdfs fsck <path>

  Check the entire filesystem $hdfs fsck /

HDFS Shell. dfsadmin

- Commands for **HDFS administration**
  – $hdfs dfsadmin -<command>
  – Example: $hdfs dfsadmin -report

- **report** – displays HDFS statistics

- **safemode** – enables safe mode for administrative tasks
  – Used for upgrade, backup, etc.

HDFS Shell. fsimage, edits & EC

Get a current state of the FC:

```
$hdfs dfsadmin -fetchImage /tmp/fsimage
```

Convert it to the readable xml format:

```
$hdfs oiv -i fsimage -o xml -p Delimited
```

Manage EC policies for different directories:

```
$hdfs ec –listPolicies
$hdfs ec -setPolicy -path <dir_path> -policy <policy>
```

HDFS Shell. Balancer

- Blocks in HDFS may be unevenly distributed across the cluster’s DataNodes
- Balancer – a utility that automatically analyzes block placement in HDFS and attempts to rebalance them
- Command: $hdfs balancer

HDFS Shell. Snapshots

- Allow creating a point-in-time copy of a directory without duplicating data (copy-on-write)
- Useful for protecting against accidental file deletion/modification
- Create: hdfs dfs -createSnapshot /dir snap1
- Delete: hdfs dfs -deleteSnapshot /dir snap1
- List: hdfs dfs -listSnapshots /dir