Hadoop Distributed File System

Prof. Dr. Stephan Trahasch
Hochschule Offenburg

Gliederung

- HDFS
- Practice
- Data Integrity
- Deployment & Sizing

Google File System

Reasons for developing your own file system at Google

- Very large number of read operations
- Reliability
- High througput: 40 GB/s read/write throughput
- Data is deleted, overwritten, compressed in extremely rare cases
- Data are usually appended or read out
- Petabyte of data

More than 200 clusters, each cluster with more than 5,000 servers (estimated in 2008)

Jeff Dean: Software Engineering Advice from Building Large-Scale Distributed Systems

The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures

slow disks, bad memory, misconfigured machines, flaky machines, etc.

Google

Design Features

- Memory and processing scale to petabyte size.
- Storage and processing of very large files can be several hundred gigabytes in size.
- Efficient data processing as batch processing once, read-many-times
- Import of "arbitrary" data without a schema definition (schema on read vs. schema on write).
- Optimized for streaming reads of files Optional access complex.
- commodity hardware
- HDFS continues to work even during a node failure without noticeable interruption

Hadoop is less suitable

- for many small, structured data sets
- Latency times in the millisecond range
- when transactions are required
- Data must be changed frequently

Seek vs. Scan

Szenario: Datenbank mit 1 TB (10^{12}) an Daten. Ein Record besteht aus 100 Bytes.

Es sollen 1% der Records aktualisiert werden. 1% → 10^8 Records

Scenario 1: Random Access

- Jedes Update benötigt ~30 ms (seek, read, write)
- $30ms * 10^8 \approx 35$ Tage für Updates

Scenario 2: Rewrite alle Records

- Annahme 100 MB/s throughput
- Zeit = 5.6 Stunden

→ Avoid Random Seeks!

Numbers Everyone Should Know*

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA → Netherlands → CA	150,000,000 ns

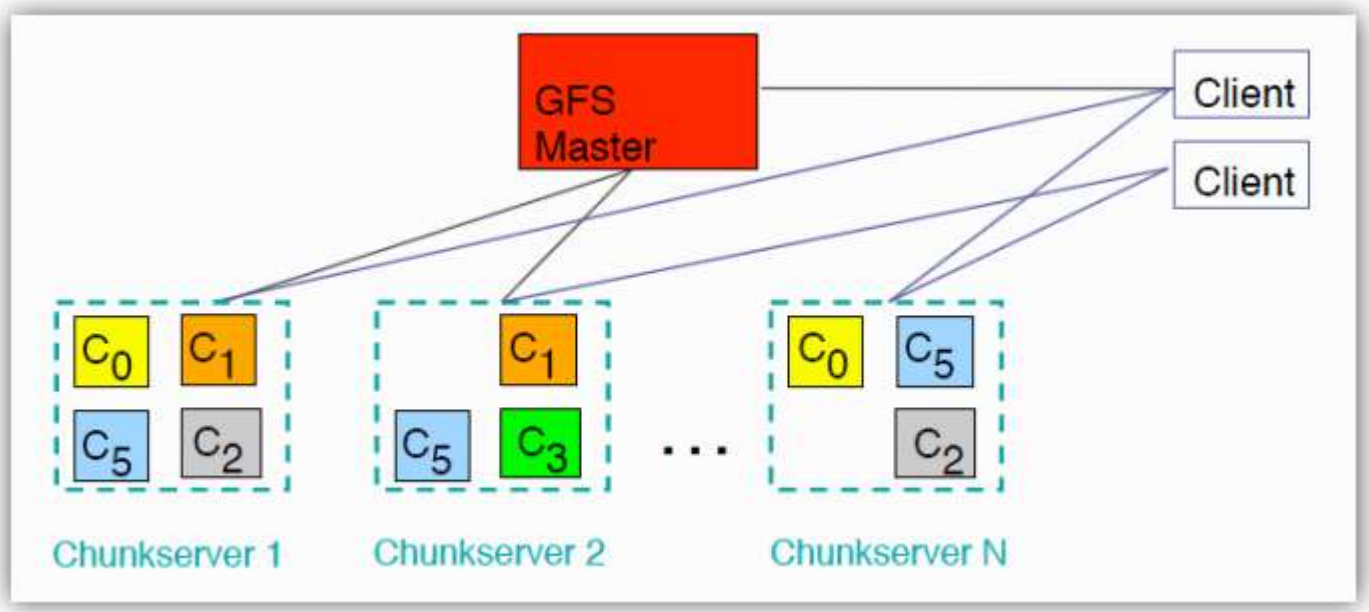
* According to Jeff Dean (LADIS 2009 keynote)

Hadoop Filesystem is based on the Google Filesystem

Apache HDFS is an open source implementation of the Google Files system.

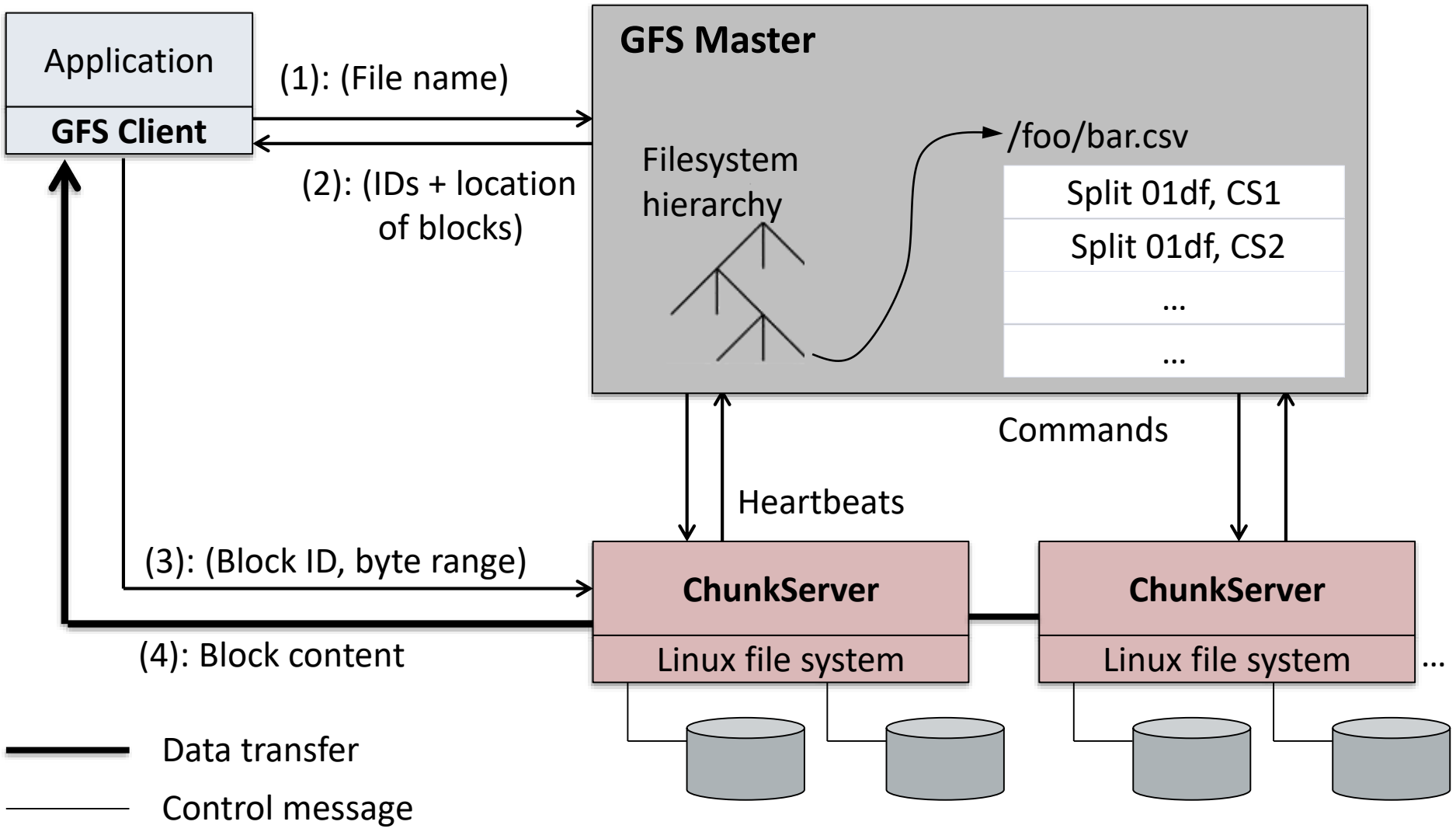
http://hadoop.apache.org/

Google File System



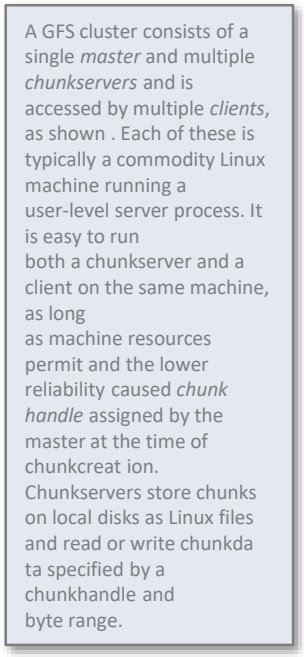
- Master responsible for metadata
- Data transfer between Client and Chunkserver
- Data stored as Chunks/Blocks (64 MB, today 128 MB).
- Replication of each block, 3 times

Master is central access point

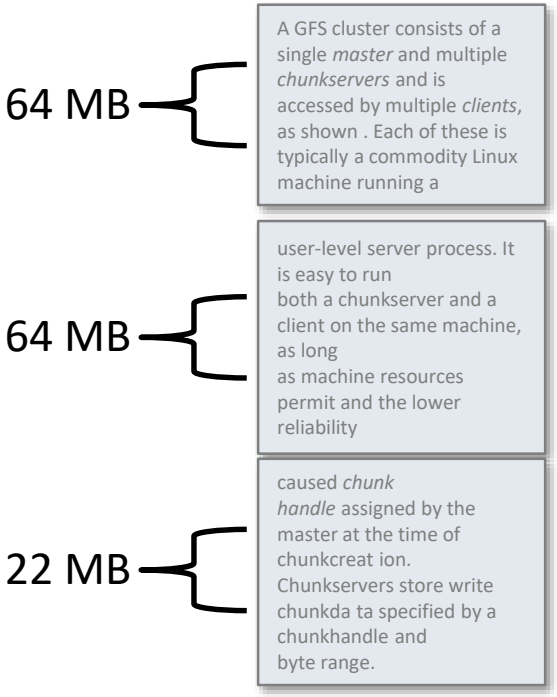


(Source [GFS03])

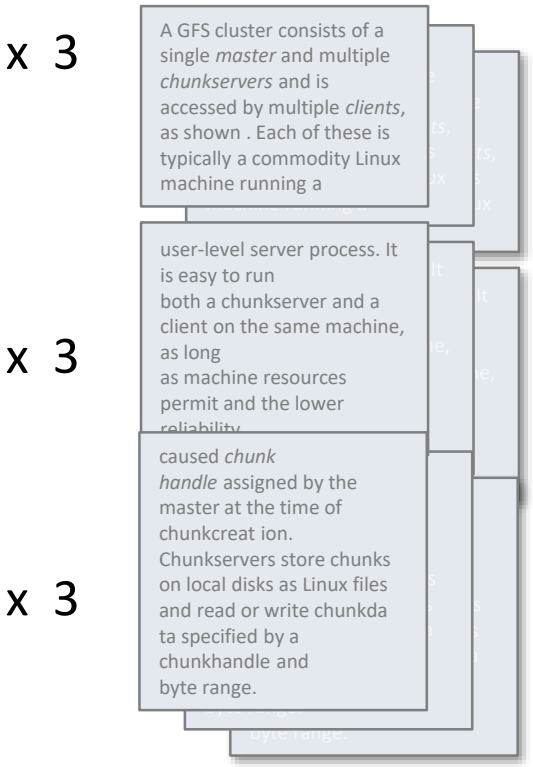
GFS repliziert jede Datei mindestens 3 mal



150 MB

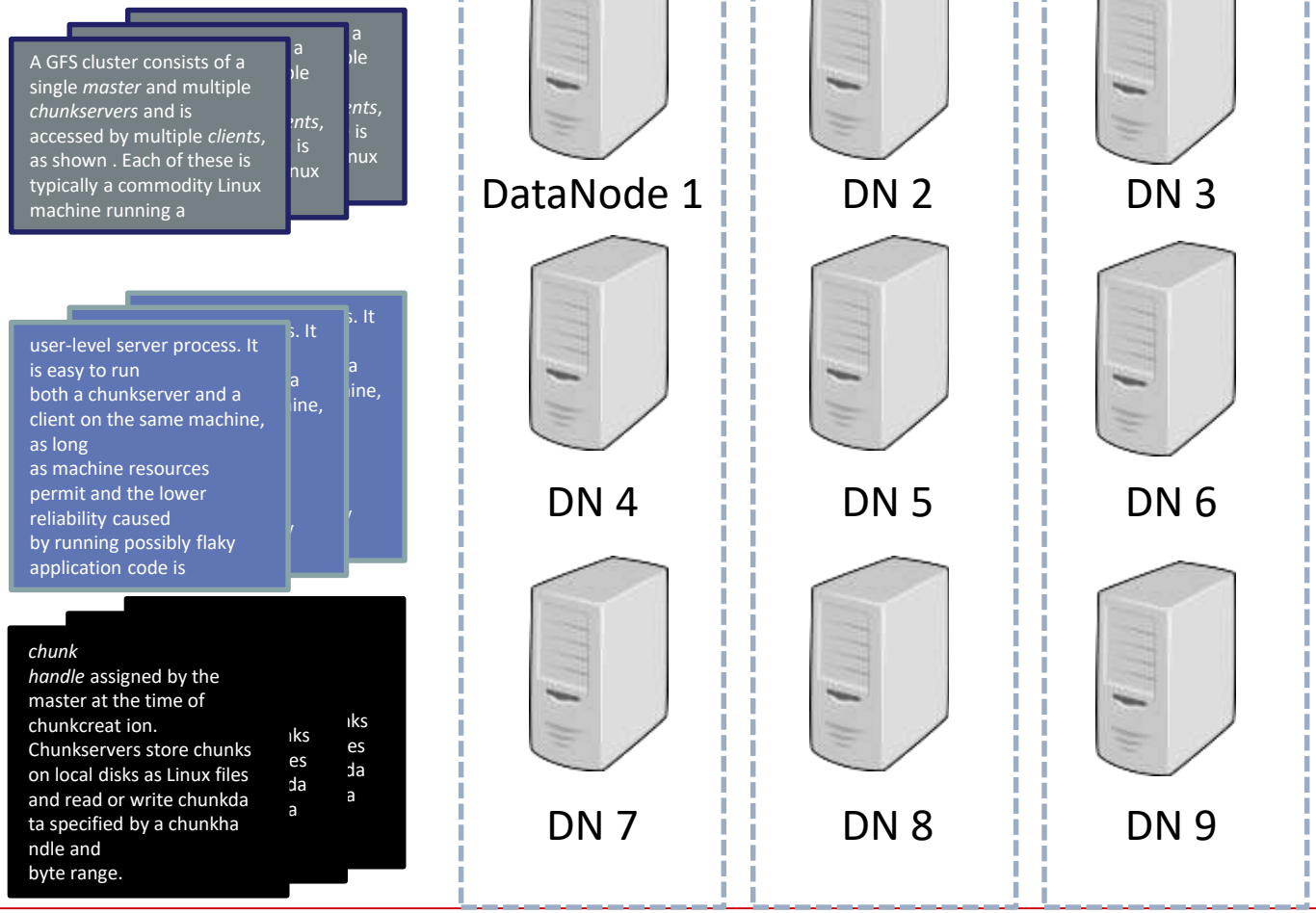


Blockbildung
(früher 64 MB;
heute 128 MB)



Replikation
(einstellbar pro Datei,
Default 3)

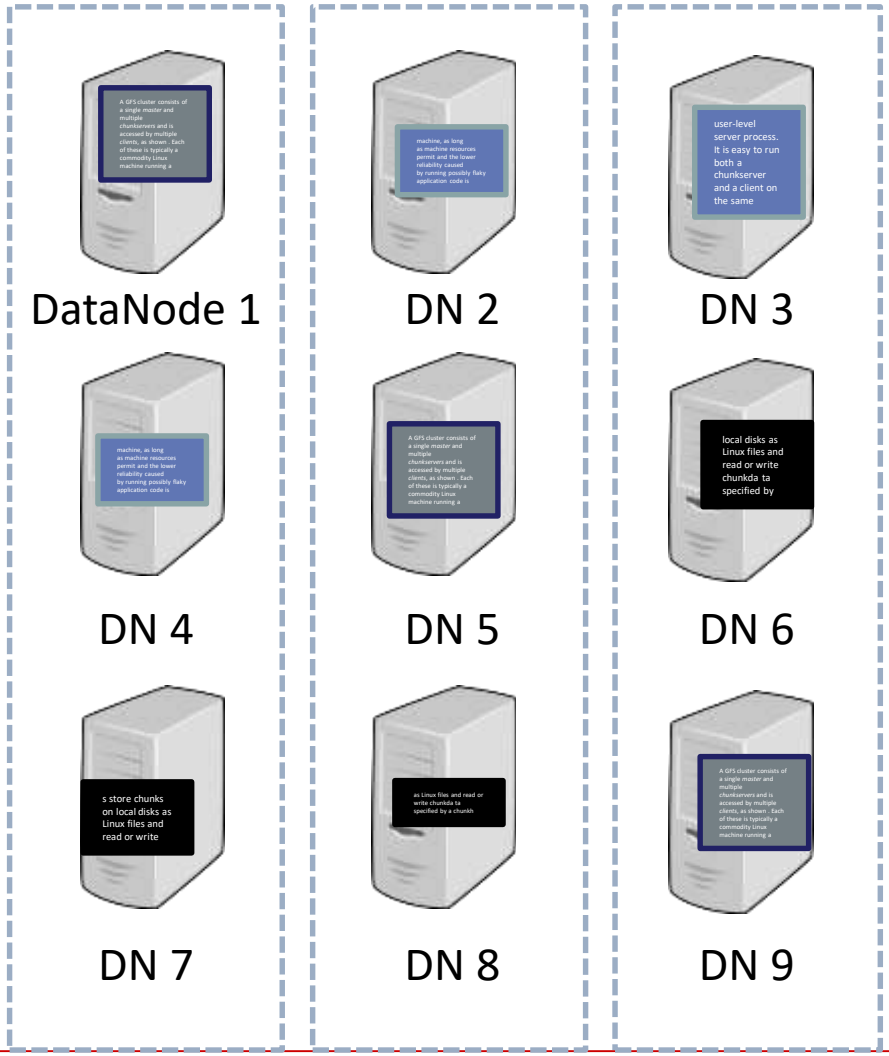
Replikation



Replikation

Blöcke werden auf DataNodes verteilt

Möglichst kein Block zweimal im selben Rack

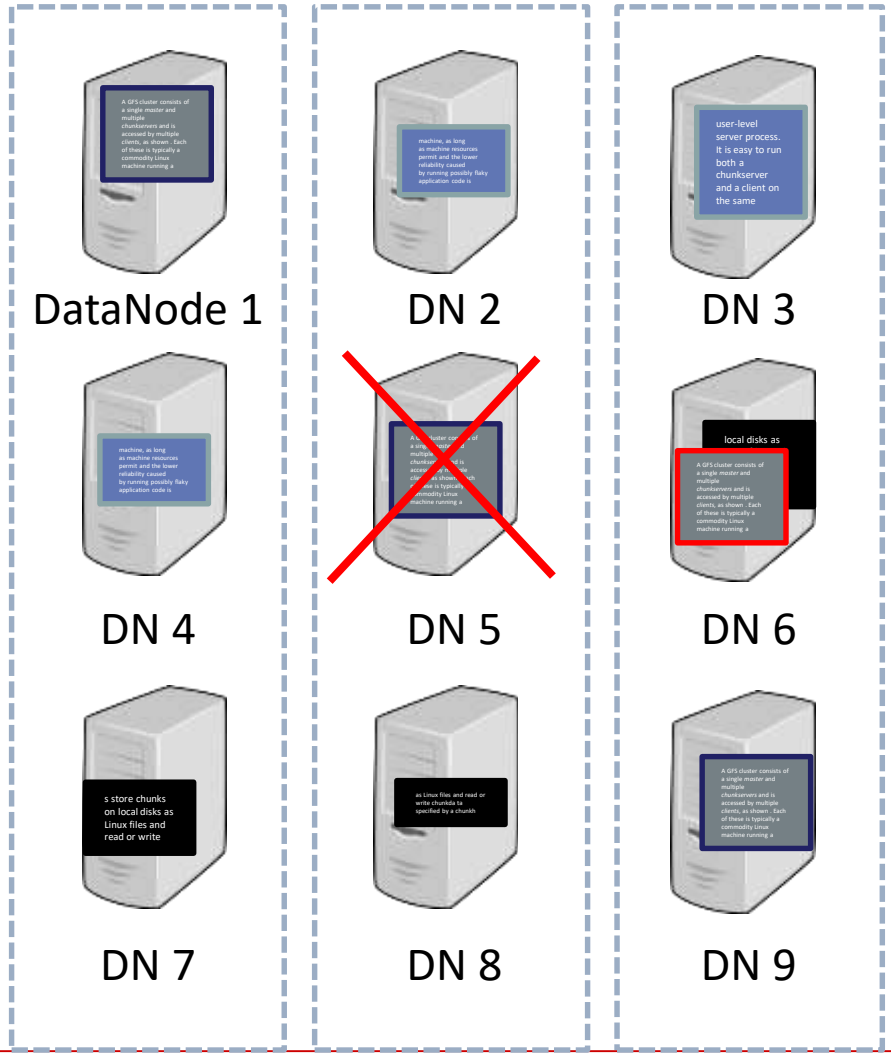


Replikation

Sehr viele Nodes

⇒ Häufig Ausfälle

NameNode veranlasst Replikation fehlender Blöcke



Default Replication Policy

Goal of first replica: speed

- Goes to data node closest to the client

Goal of second replica: availability

- If rack awareness if configured, each DataNode will have a rack id associated with it
- Second replica goes to a different rack in the cluster

Goal of third replica: minimizing inter-rack network traffic

- Goes to a different node on the same rack as second replica

Modifying the Replication Factor

- Syntax:
  `hdfs fs -setrep [-w] <rep> <path/file>`
  `[-w]` instructs NameNode to block and wait for the task to complete

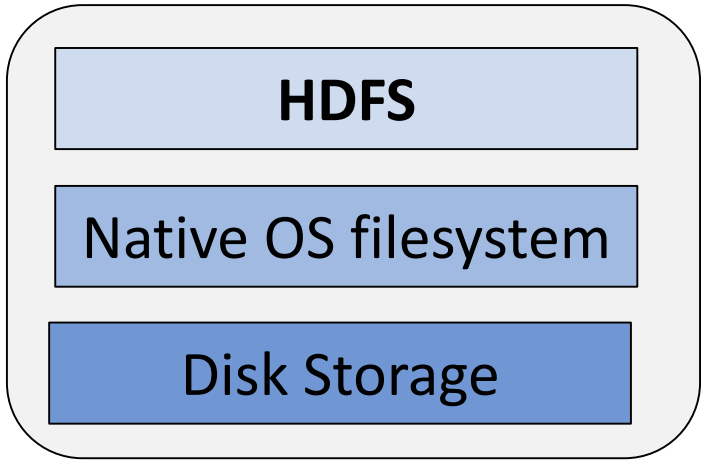
- To change a directory replication factor to 4:
  ```
  $ hdfs fs -setrep 4 /user/hdfs
  ```

- To change a file replication factor to 2:
  ```
  $ hdfs fs -setrep –w 2 /directorypath/file
  ```

- To change the replication factor of the entire HDFS namespace hierarchy to 4:
  ```
  $ hdfs fs -setrep –w 4   /
  ```

Hadoop File System (HDFS)

- Basiert auf GFS
- HDFS ist ein Filesystem, programmiert in Java
- Setzt auf einem vorhanden Filesystem auf
- Block Size 128MB
- Blocks werden 3x über DataNodes repliziert
- MasterNode (master) und DataNode (worker)



Why HDFS Blocks are Large in Size?

HDFS blocks are large in size → reduce cost of seek time.
Seek time is 10ms and disk transfer rate is 100MB/s. To make the seek time 1% of the disk transfer rate, block size should be 100MB. Default size HDFS block is 128MB.

Advantages of HDFS Block

- fixed size, it is very easy to calculate the number of blocks that can be stored on a disk.
- If size of a file is less than HDFS block size, then the file does not occupy the complete block storage.
- It is easy to store a file that is larger than disk size. Data blocks are distributed and stored on multiple nodes in a cluster.
- Blocks are easy to replicate between the datanodes and thus provide fault tolerance and high availability.

Hadoop setzt sich aus zwei unterschiedlichen Nodes

1. **NameNode**
   Verwaltet die Metadaten für HDFS.
   Optional: Secondary oder Standby NameNode
   - Secondary NameNode übernimmt Hintergrundarbeiten für NameNode, ist aber kein Backup oder Hot-Standby
   - Standby NameNode für HA Modus
     Im Hochverfügbarkeitsmodus ist dieser NameNode gleichzeitig Hot-Standby als auch Hintergrundarbeiter.

2. **DataNode**
   Speichert die eigentlichen HDFS-Datenblöcke.

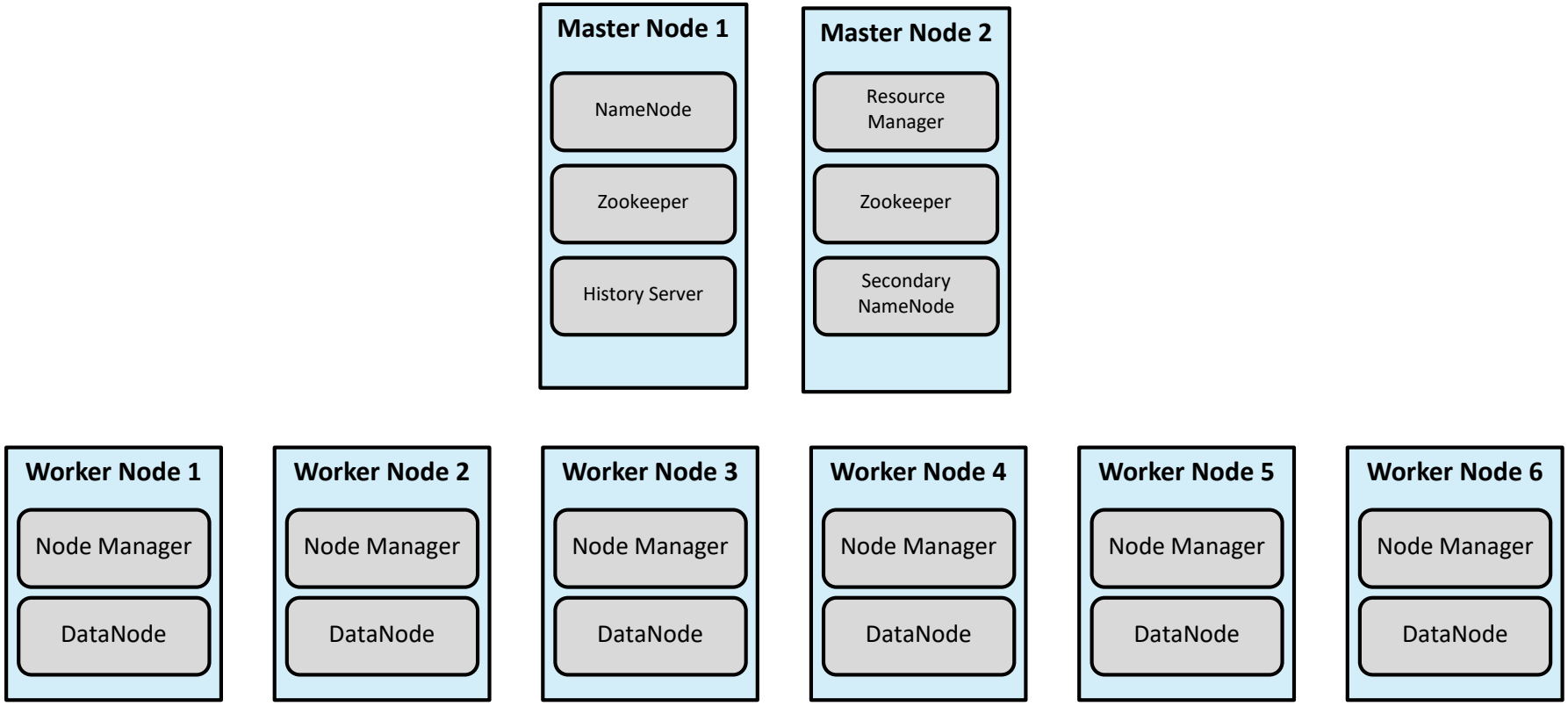
Auf den Nodes laufen mehrere Prozesse.

Hadoop cluster is made up of master and worker (data) nodes

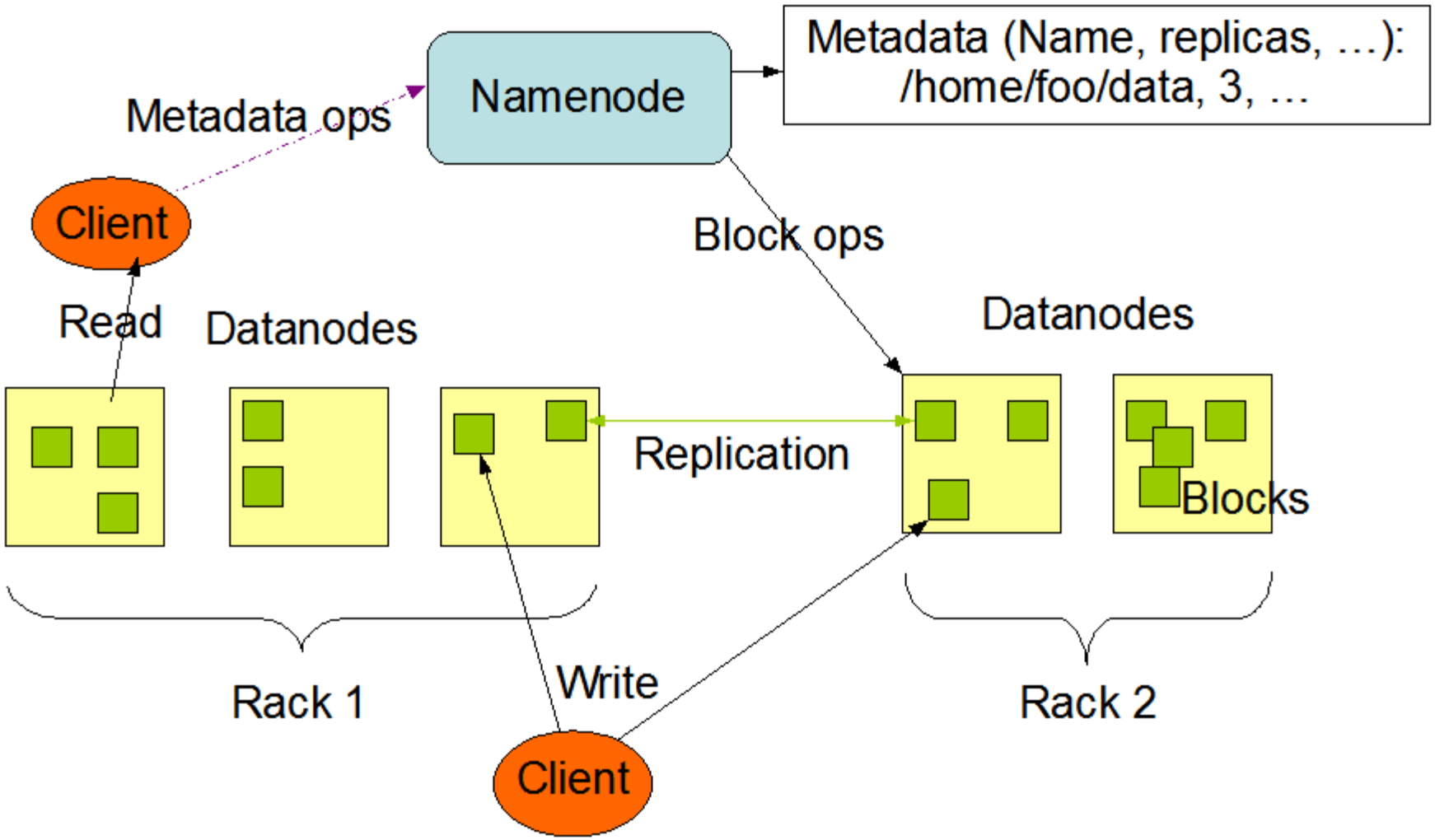
Master nodes manage (NameNode) the infrastructure

Without metadata on NameNode, there is no way to access files

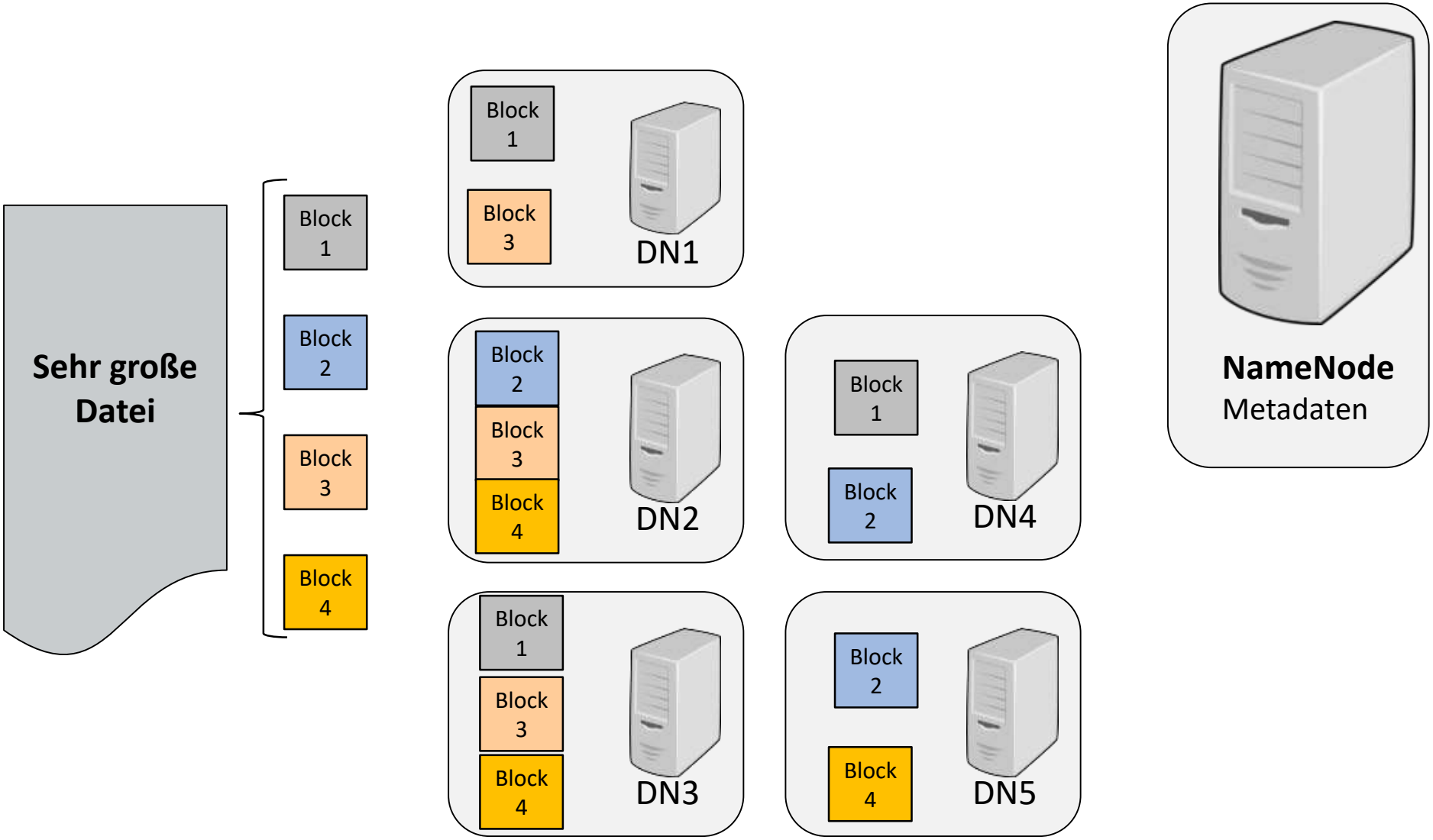
Worker nodes contain the distributed data and perform processing



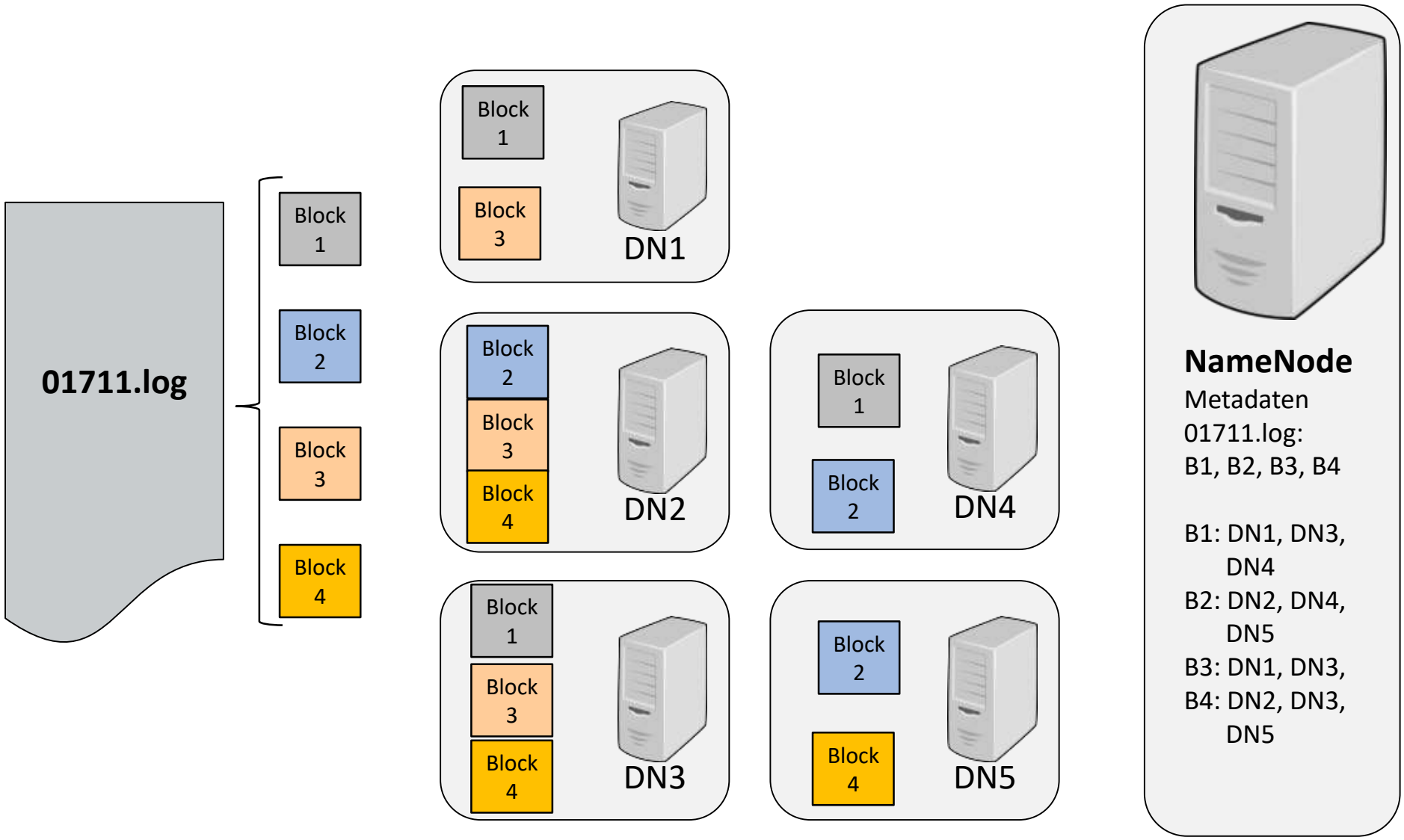
HDFS-Architektur



Speicherung und Replikation

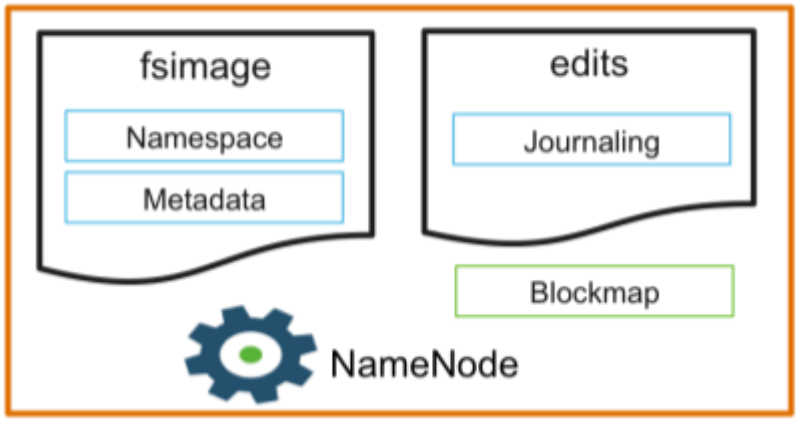


Speichern


01711.log
Block 1
Block 2
Block 3
Block 4
Block 1
Block 3
DN1
Block 2
Block 3
Block 4
DN2
Block 1
Block 2
DN4
Block 1
Block 3
Block 4
DN3
Block 2
Block 4
DN5
NameNode
Metadaten
01711.log:
B1, B2, B3, B4
B1: DN1, DN3, DN4
B2: DN2, DN4, DN5
B3: DN1, DN3,
B4: DN2, DN3, DN5

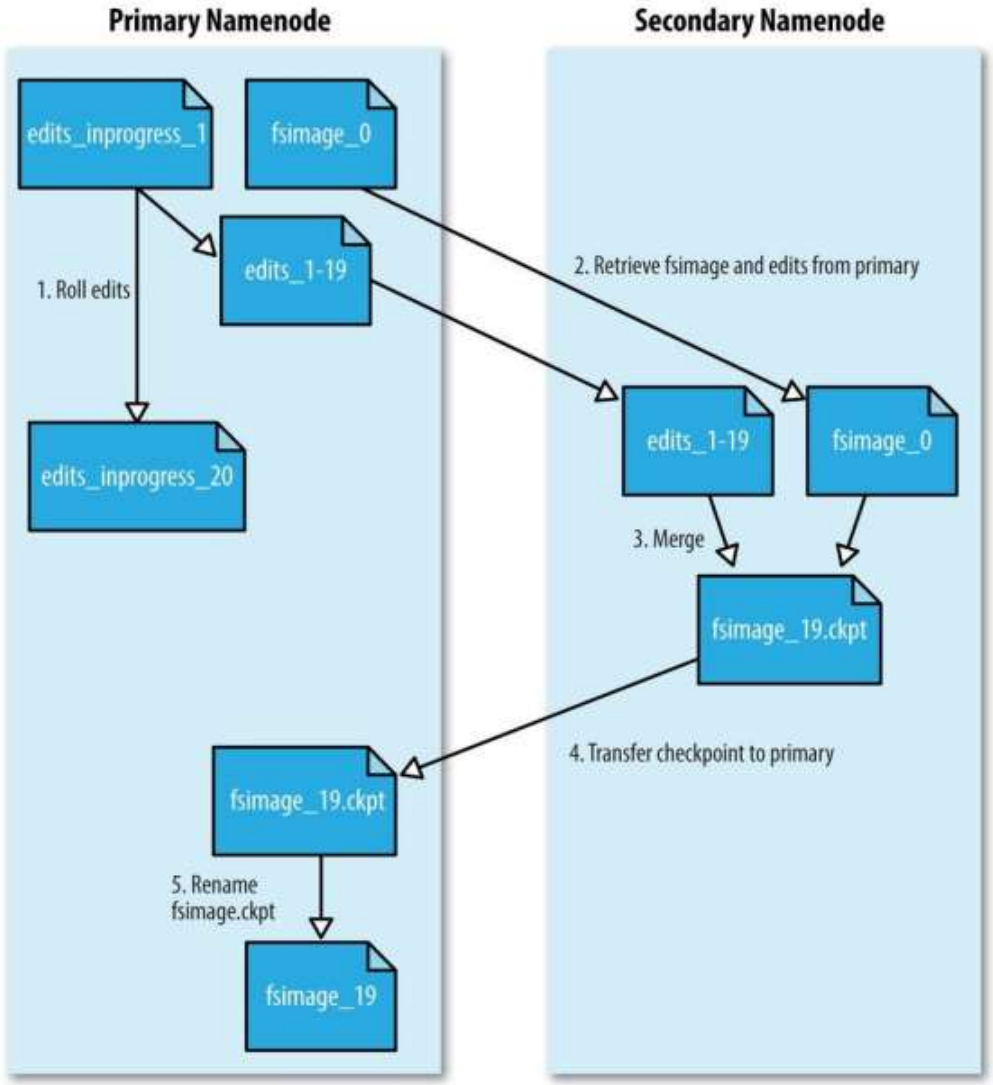
NameNode tracks changes with edit-log and fsimage

```
${dfs.namenode.name.dir}/
├──current
│   ├── VERSION
│   ├── edits_0000000001-0000000019
│   ├── edits_inprogress_0000000000020
│   ├── fsimage_0000000000000000
│   ├── fsimage_0000000000000000.md5
│   ├── fsimage_0000000000000019
│   ├── fsimage_0000000000000019.md5
│   └── seen_txid
└── in_use.lock
```



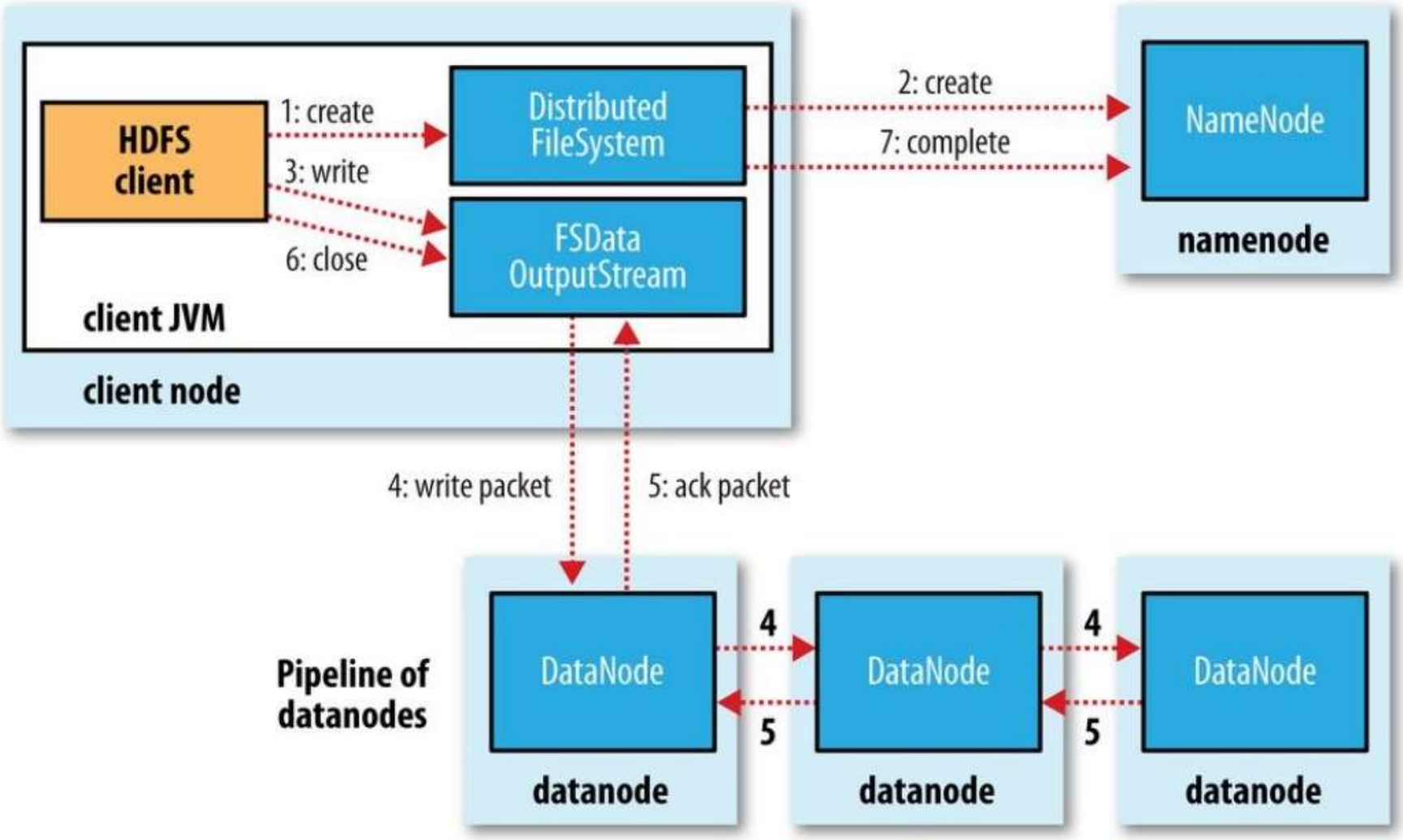
1. NameNode starts: reads **fsimage_N** and **edits_N**
2. Transactions in **edits_N** merged with **fsimage_N**
3. **fsimage_N+1** is written, **edits_N+1** is created

Checkpoint Process with Secondary NameNode



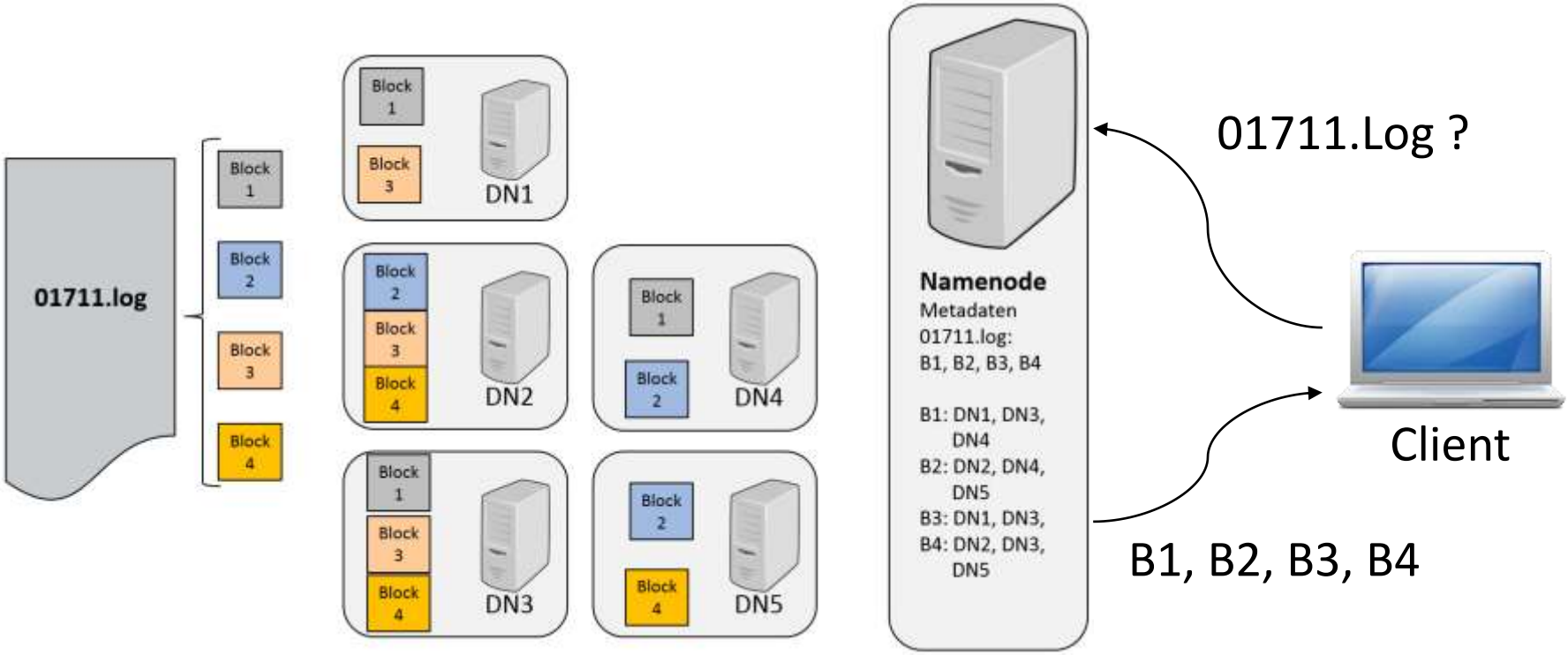
Source: Hadoop – The Definitive Guide

Writing data

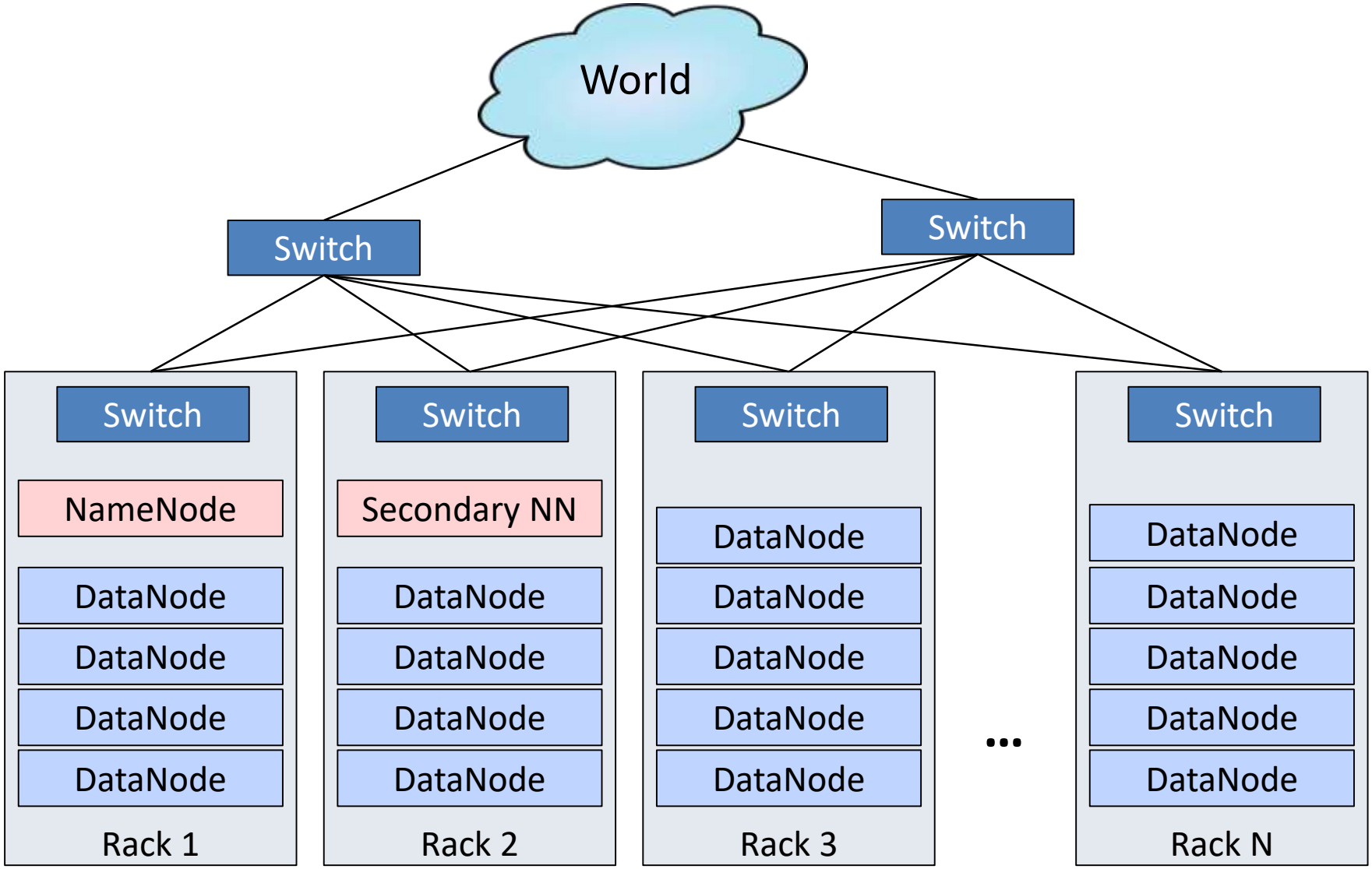


Source: Hadoop – The Definitive Guide

Reading


01711.log
Block 1
Block 2
Block 3
Block 4
DN1
DN2
DN3
DN4
DN5
Namenode
Metadaten
01711.log:
B1, B2, B3, B4
B1: DN1, DN3, DN4
B2: DN2, DN4, DN5
B3: DN1, DN3,
B4: DN2, DN3, DN5
01711.Log ?
Client
B1, B2, B3, B4

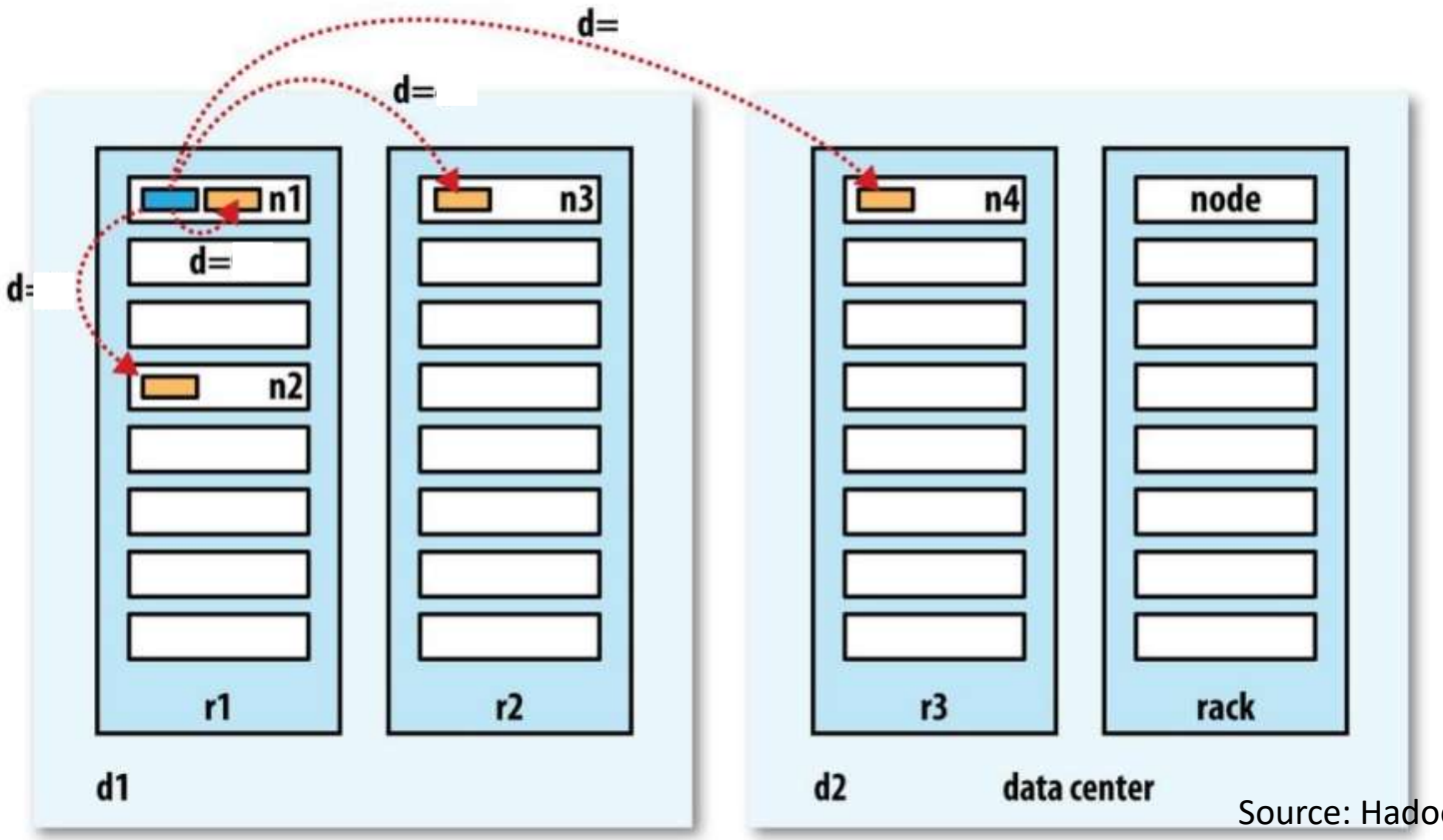
Server are Rack-aware



Network Distance

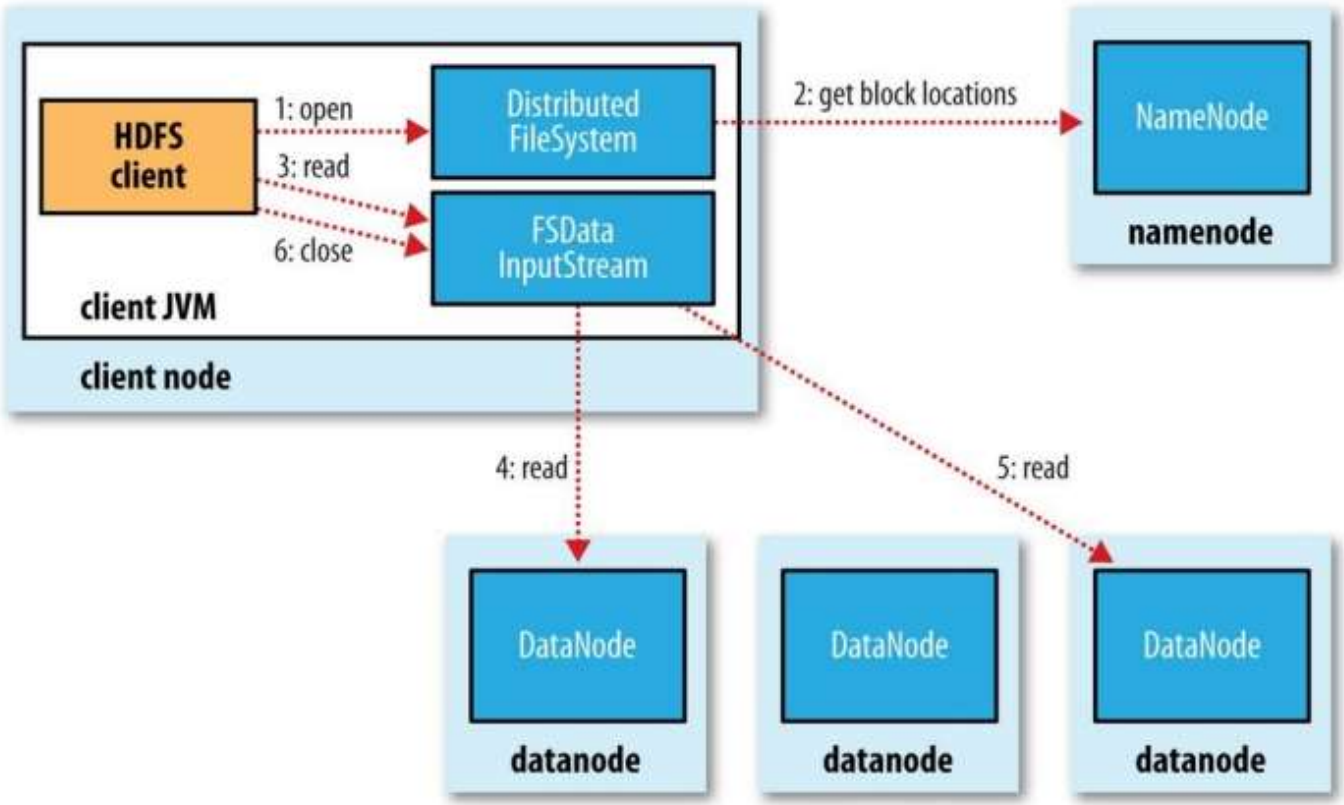
Network wird als Baum repräsentiert.

Distanz zwischen Knoten ist Summe der Distanz zum ersten gemeinsamen Elternknoten.



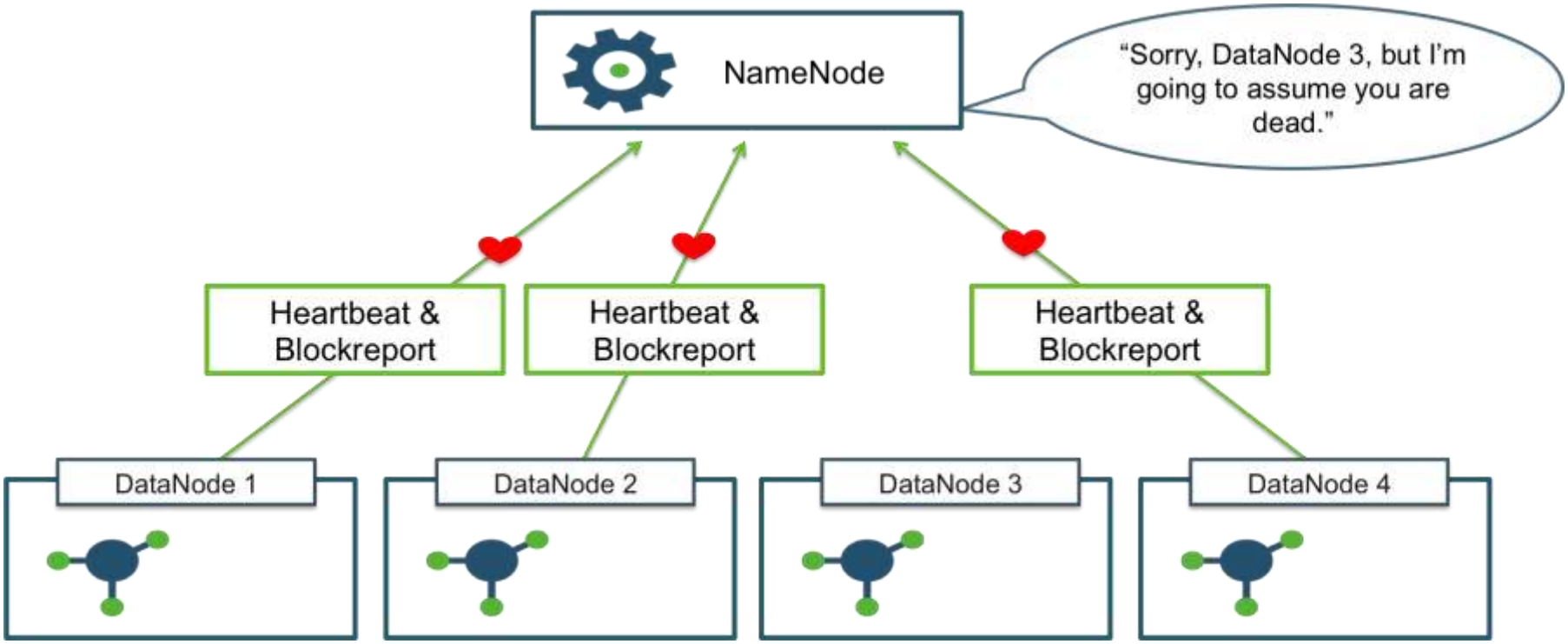
Source: Hadoop – The Definitive Guide

Client reading data



Source: Hadoop – The Definitive Guide

DataNodes



Hortonworks

Gliederung

- HDFS
- Practice
- Data Integrity
- Deployment & Sizing

HDFS Commands

```
hdfs dfs -command [args]
```

Here are a few (of the almost 30) HDFS commands:

-cat: display file content (uncompressed)

-text: just like cat but works on compressed files

-chgrp,-chmod,-chown: changes file permissions

-put,-get,-copyFromLocal,-copyToLocal:
copies files from the local file system to the HDFS and vice versa.

-ls, -ls -R: list files/directories

-mv,-moveFromLocal,-moveToLocal: moves files

-stat: statistical info for any given file (block size, number of blocks, file type, etc.)

HDFS Command Line Examples

Copy file foo.txt from local disk to the user's directory in HDFS

```
$ hdfs dfs -put foo.txt foo.txt
```

This will copy the file to /user/username/foo.txt

Get a directory lis-ng of the user's home directory in HDFS

```
$ hdfs dfs -ls
```

Get a directory listing of the HDFS root directory

```
$ hdfs fs –ls/
```

Delete the directory input_old and all its contents

```
$ hdfs fs –rm -r input_old
```

HDFS Command Line Examples (2)

Display the contents of the HDFS file /user/fred/bar.txt

```
$ hdfs dfs -cat /user/fred/bar.txt
```

Copy that file to the local disk, named as baz.txt

```
$ hdfs dfs -get /user/fred/bar.txt baz.txt
```

Create a directory called input under the user's home directory

```
$ hdfs fs –mkdir input
```

HDFS File Permissions

Files and directories have owners and groups

r = read

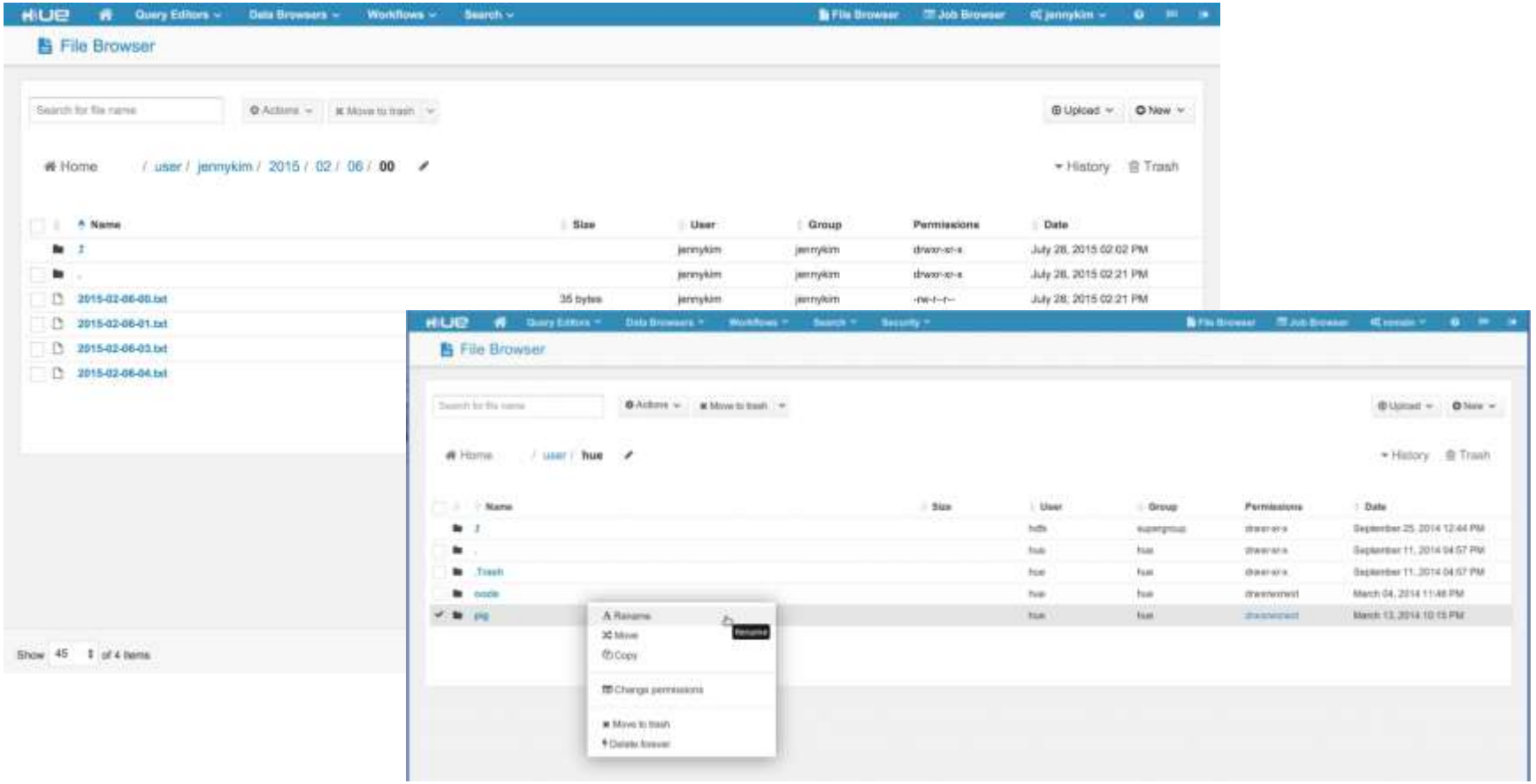
w = write

x = permission to access the contents of a directory

```
drwxr-xr-x   - hue       hue            0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/shell
-rwxr-xr-x   3 hue       hue           77 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/shell/hello.py
drwxr-xr-x   - hue       hue            0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sleep
-rwxr-xr-x   3 hue       hue            0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sleep/empty
drwxr-xr-x   - hue       hue            0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop
-rwxr-xr-x   3 hue       hue         7175 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop/TT.java
-rwxr-xr-x   3 hue       hue          420 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop/db.hsqldb.properties
-rwxr-xr-x   3 hue       hue          276 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop/db.hsqldb.script
drwxr-xr-x   - hue       hue            0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/ssh
-rwxr-xr-x   3 hue       hue            0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/ssh/empty
drwxr-xr-x   - root      root           0 2013-08-29 03:22 /user/root
drwxr-xr-x   - root      root           0 2013-08-29 03:23 /user/root/mydata
-rw-r--r--   3 root      root        2549 2013-08-29 03:23 /user/root/mydata/numbers.txt
-rw-r--r--   3 root      root     3613198 2013-08-28 21:55 /user/root/stocks.csv
[root@sandbox demos]#
```

HUE Filebrowser

Create,move, rename, modify, upload, download and delete directories and files. Also view file contents.



Empfehlungen für Aufbau des HDFS (Quelle Cloudera)

HDFS is a repository for all your data.
→ Structure and organize carefully!

→ Define a standard directory structure!

Example organization

- /user/… – data and configuration belonging only to a single user
- /etl – Work in progress in Extract/Transform/Load stage
- /tmp – Temporary generated data shared between users
- /data – Data sets that are processed and available across the organization for analysis
- /app – Non-data files such as configuraIon, JAR files, SQL files, etc.

Gliederung

- HDFS
- Practice
- Data Integrity
- Deployment & Sizing

Ensuring Data Integrity

Hadoop provides a number of features and products that ensure data is available and robust:

NameNodes

Maintain the defined number of block replicas assuring availability

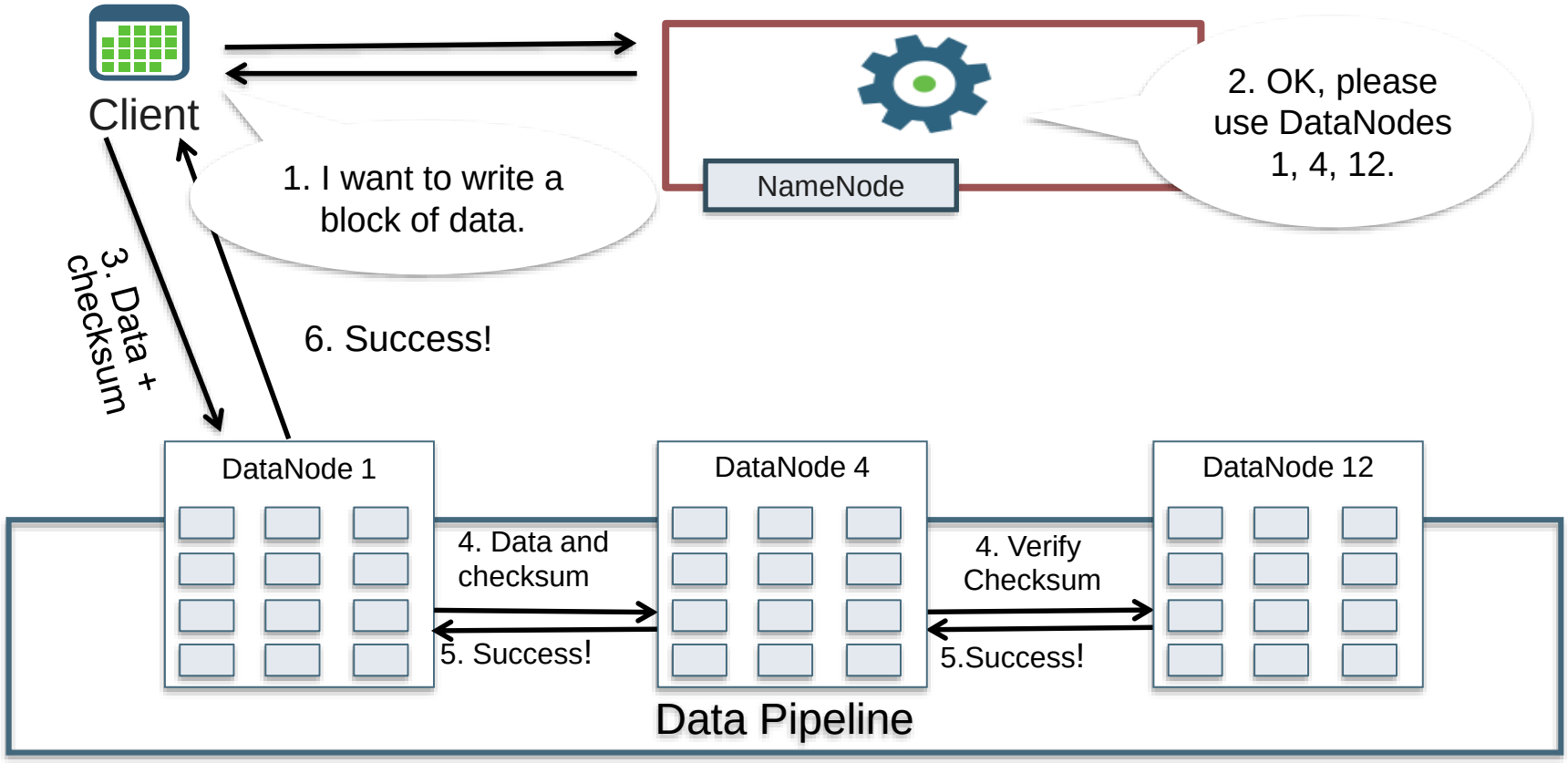
Block scanning

Automated scanning all blocks in a Hadoop cluster over a defined time period

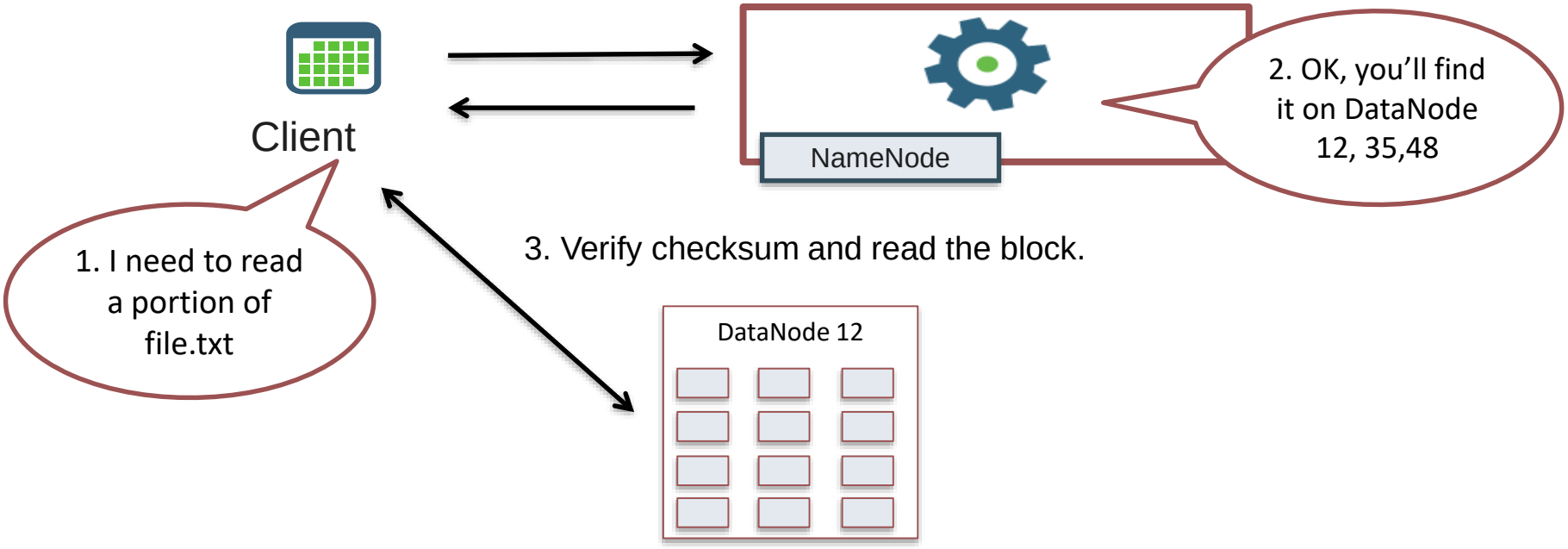
Filesystem checks

Manually be performed on the entire file system or on individual directory structures

Writing Data

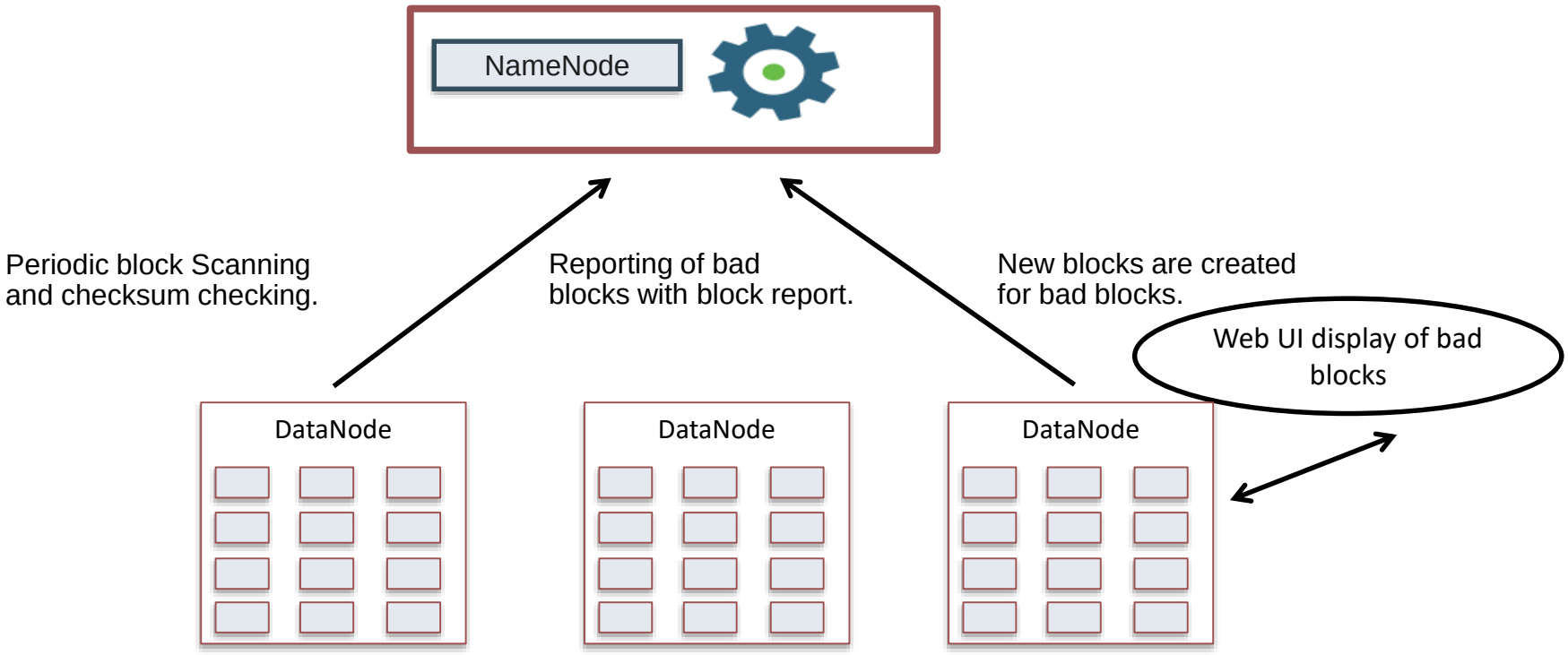


Reading Data



Block Scanning

Deep inspection every 3 weeks



Running a File System Check on HDFS

A file system check is run when:

- There may be block corruption
- After a system malfunction or failure to verify no blocks have been corrupted
- Before performing an upgrade

A file system check on HDFS is performed manually

- Block scanning is configured to run automatically
- HDFS file system check does not repair any blocks
- Reports on block and replication inconsistencies
- NameNode will perform corrections

What Does File System Check Look For?

- Missing replicas
  - Missing from expected replication factor
- Under-replicated blocks
  - Places these files on a list for replication
  - Review the list by entering: `hdfs dfsadmin –metasave`
- Over-replicated blocks
  - Extra blocks will be deleted
- Incorrectly replicated blocks
  - All replicas on a single rack, out of sync with the rack topology configuration
  - Fixed automatically
- Corrupt blocks
  - Reported only when ALL replicas are corrupt

hdfs fsck Syntax

```
hdfs fsck [OPTIONS] <path> [-move | -delete | -openforwrite] [-files
[-blocks [-locations | -racks]]]
```

Options	Description
path	Start checking from this path
-move	Move corrupted files to /lost+found
-delete	Delete corrupted files
-openforwrite	Print out files opened for write
-files	Print out information on files being checked
-blocks	Print out block information
-locations	Print out DataNode address of blocks
-racks	Print out rack information for data-nodes

File System Check

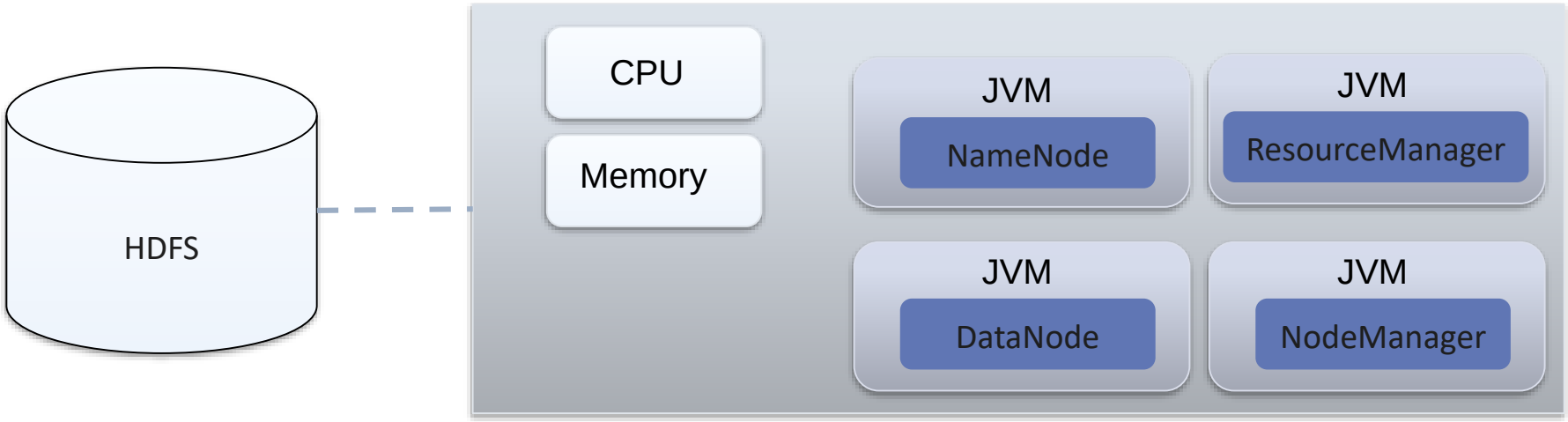

```
$ hdfs fsck /
.....................................................................
Status: HEALTHY
 Total size:        128847681 B
 Total dirs:        144
 Total files:       200 (Files currently being written: 3)
 Total blocks (validated):    198 (avg. block size 650745 B) (Total open file blocks (not validated): 3)
 Minimally replicated blocks: 198 (100.0 %)
 Over-replicated blocks:      0 (0.0 %)
 Under-replicated blocks:     1 (0.5050505 %)
 Mis-replicated blocks:               0 (0.0 %)
 Default replication factor:  3
 Average block replication:   2.989899
 Corrupt blocks:              0
 Missing replicas:            7 (1.1824324 %)
 Number of data-nodes:                3
 Number of racks:             1
FSCK ended at Wed Oct 03 12:05:58 EDT 2012 in 44 milliseconds
```

Gliederung

- HDFS
- Practice
- Data Integrity
- Deployment & Sizing

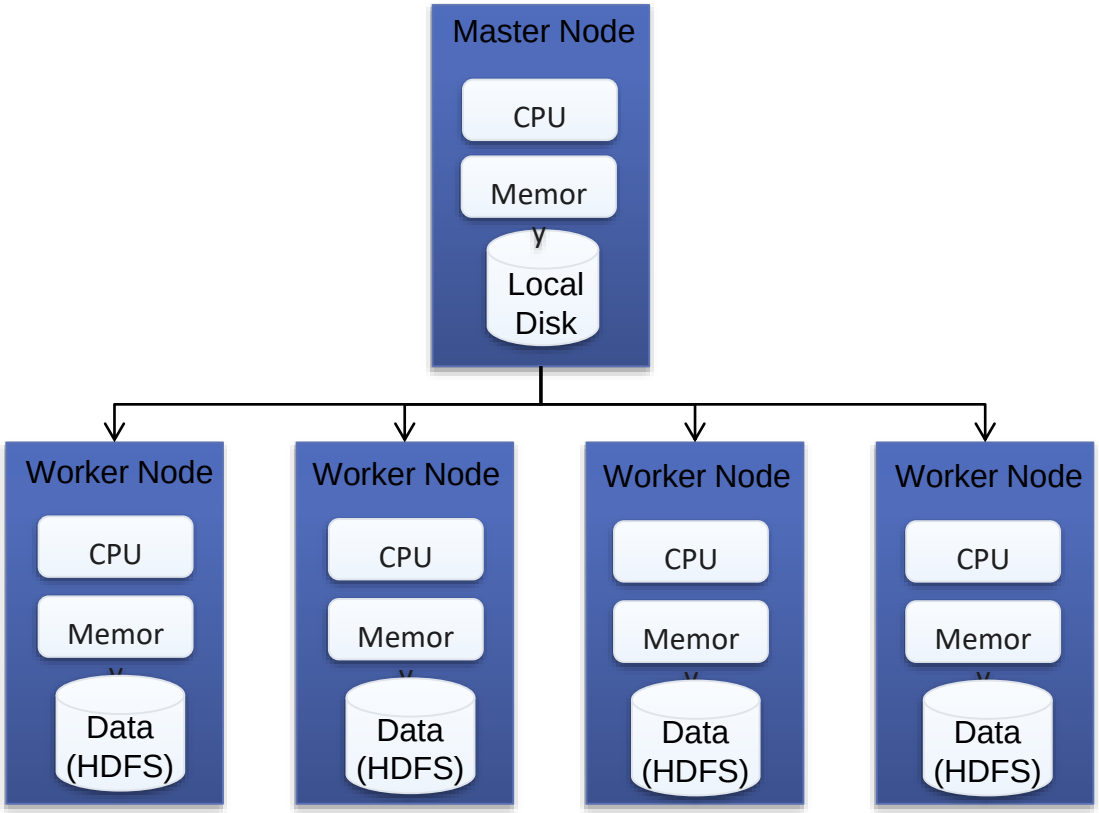
Pseudo-Distributed Deployment Mode

- Each daemon runs its own JVM
- Hortonworks Sandbox runs in Pseudo-Distributed mode
- They all run on a single machine using HDFS
- Appropriate for QA and development environments

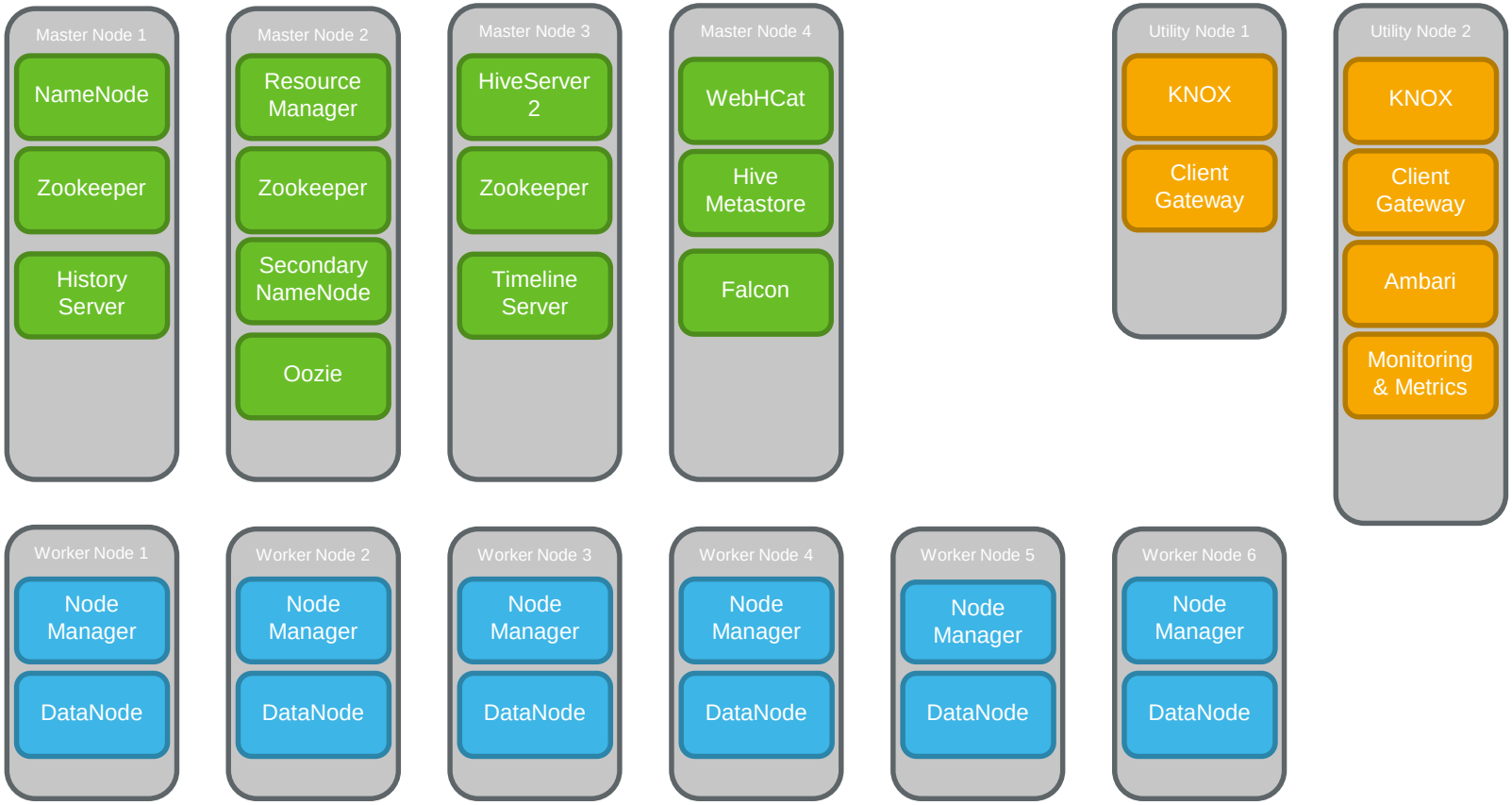


Distributed Deployment Mode

- Each daemon runs on its own machine
- Data is maintained with local HDFS
- Best for production environments



Small Cluster Without High Availability



External Database Layout

- Several components require external databases:
  - Ambari: PostgreSQL
  - Hive: MySQL
  - Oozie: Derby
  - Ranger: MySQL

- Supported: Oracle, MySQL, PostgreSQL
  - Consider using same technology for ease of management
  - Use the same servers where possible
  - Let your DBAs manage so backups, HA, … are taken care of

Heavy usage of other tools like Falcon+Oozie or Ambari may require dedicated instances

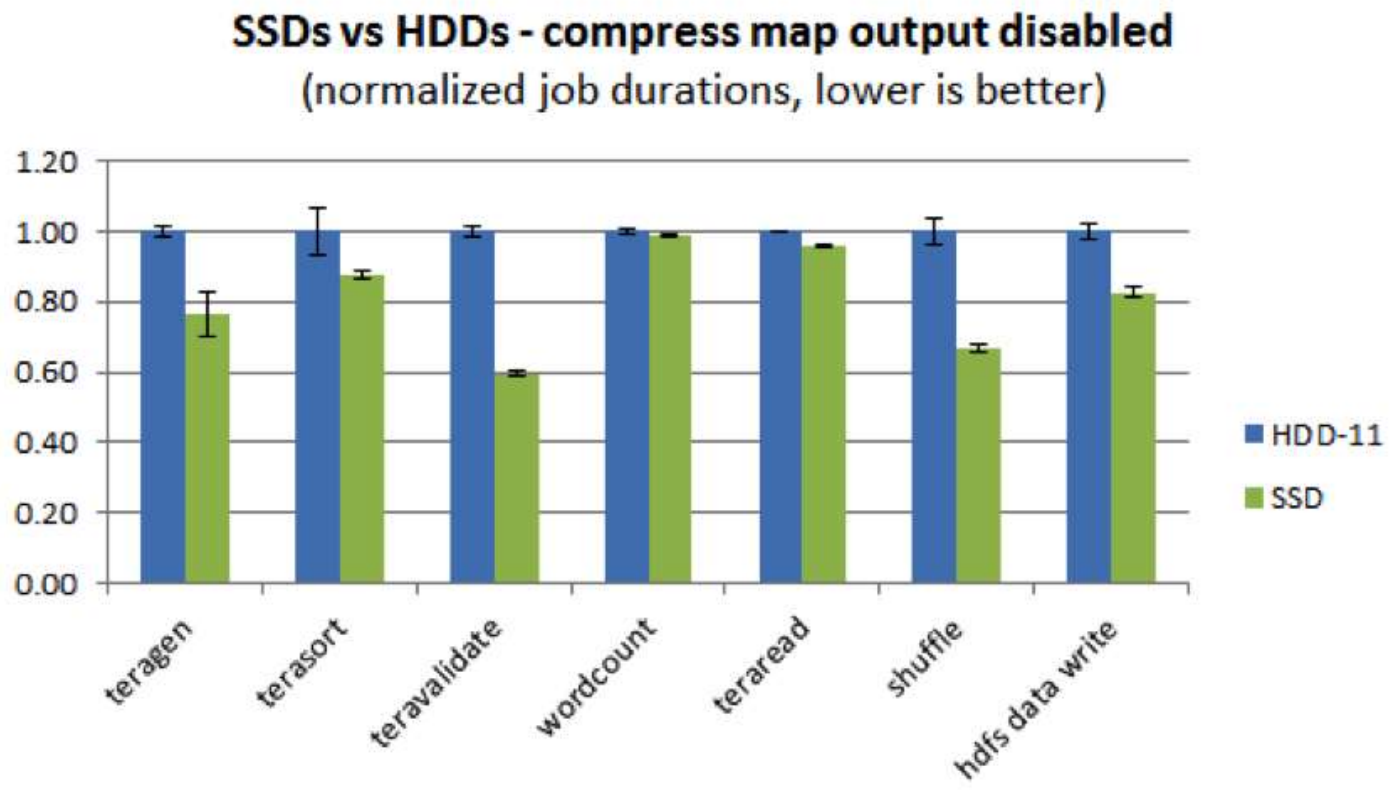
Typical Worker

- CPU
  - Dual socket / 8-12 core each
- RAM
  - Typical: 128GB
  - Not uncommon: 256GB
- HDFS Storage
  - 8-12 x 2-3TB (NL-SAS/SATA)
  - 1TB for performance focus
  - 4TB+ for storage archive focus

Storage Configuration

- Master Nodes
  - RAID-10
  - O/S + Data disks
  - NFS for shared directory in the case of HA
- Worker Nodes
  - RAID-0 per disk or JBOD
    - e.g.12 disks = 12 x 1 disk arrays
    - Ensure they are mounted separately
- Data nodes survive disk loss:
  - `dfs.datanode.failed.volumes.tolerated`

SSD vs. HDD



http://blog.cloudera.com/blog/2014/03/the-truth-about-mapreduce-performance-on-ssds/

Hardware Sizing

- What is the workload?
  - Balanced: most common
  - Memory: such as Spark
  - Compute: such as Storm
  - Storage: such as archive

- Mixed hardware in a cluster
  - Ambari Configuration groups
  - YARN Labels: Pin processing to subset of hardware
  - Heterogeneous Storage

Storage Calculator

Total Storage Required

$$\frac{(\text{Initial Size} + \text{YOY Growth} + \text{Intermediate Data Size}) \times \text{Replication Count} \times 1.2}{\text{Compression Ratio}}$$

Good Rule of Thumb

Replication Count = 3

Compression Ratio = 3-4

Intermediate Data Size = 30%-50% of Raw Data Size

Note

1.2 factor is included in the sizing estimator to account for the temp space

Network Design

Be prepared for overhead from node failure

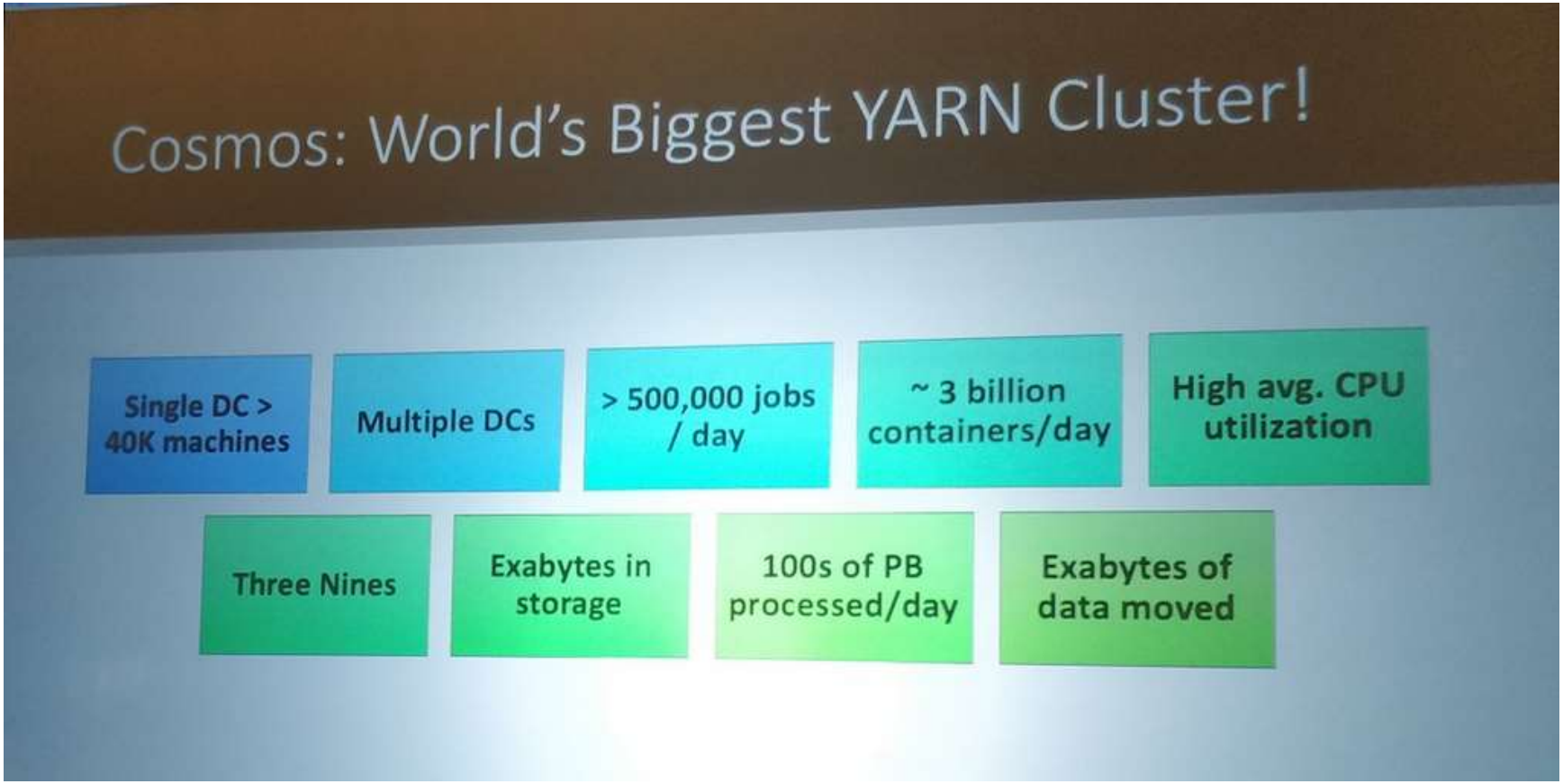
Example: a single data node with 10 TB fails

The cluster will produce ~10 TB of network traffic to recover

Typically

- Data nodes: Bonded 1 GB or Single 10GB
  Might want to go with 10GB to future proof
- Switches dedicated to the cluster only
- Rack Interconnect: 2 x 10 GB
- 2 TOR (top of rack) switches, or multiple spines so there is no Single Point of Failure

Hadoop @ Microsoft Data Center



Summary

- Apache Hadoop
  framework for processing large amounts of data
- Redundant data storage = HDFS
- Error-tolerant, distributed, parallel processing = MapReduce
- Linear scalability of memory, computing power and costs
- MapReduce
  Design principle in Hadoop SW ecosystem for processing data

References

[1] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in Proceedings of the 4th Annual Symposium on Cloud Computing, New York, NY, USA: ACM, 2013, pp. 5:1-5:16.