File Systems: GFS vs. HDFS

Ciprian Lucaci

Daniel Straub

Distributed File Systems - Motivation

The problem!!! >>= TB/PB of data

The Requirements

- Store very large data sets reliably
- High bandwidth streaming
- Distribute storage
- Distribute computation
- Analysis and transformation of data

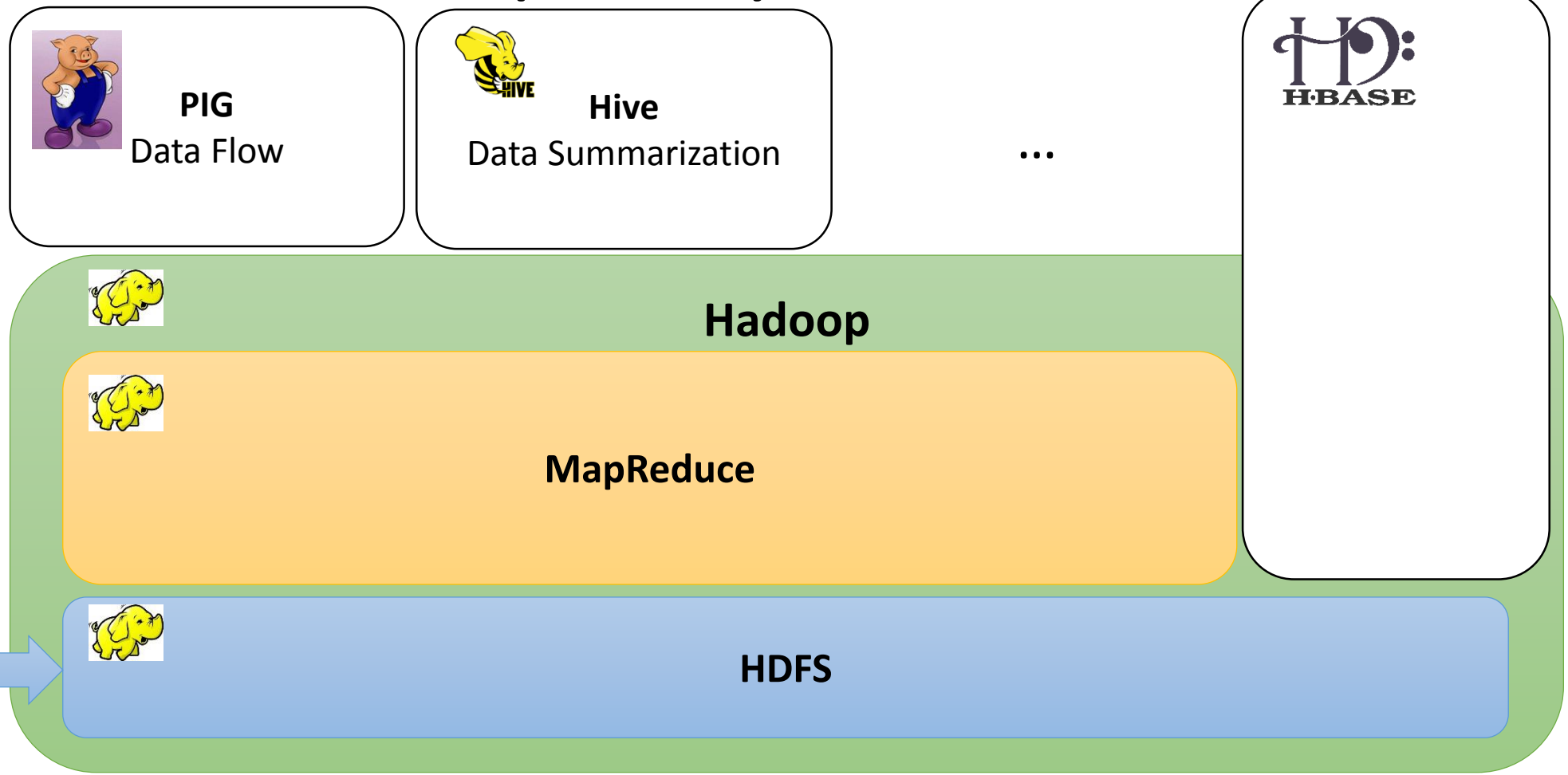
Distributed File Systems - Motivation

The solution

Google File System & Hadoop Distributed File System

- Moving computation where data resides
- Low costs – commodity machines
- Vertical and Horizontal scalability

HDFS – Hadoop Ecosystem



HDFS - Architecture

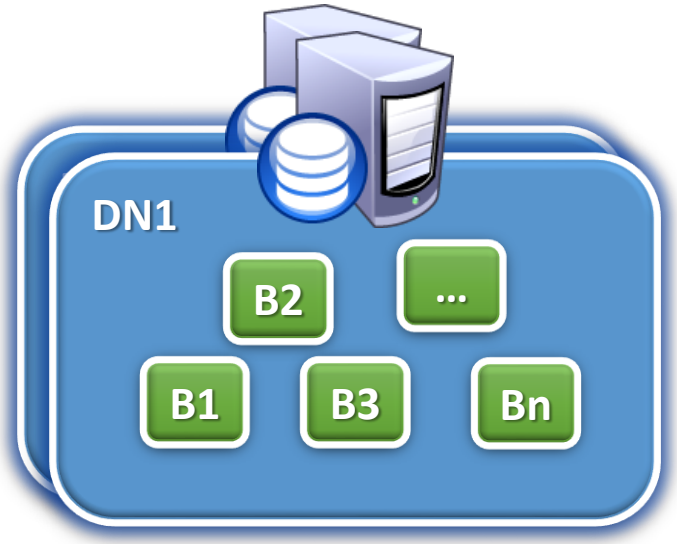
- **Blocks**
  - 64 MB / block (default), 128 MB, 256 MB
  - Split large files into blocks
  - Large file > x100 MB/GB
  - Data in any format

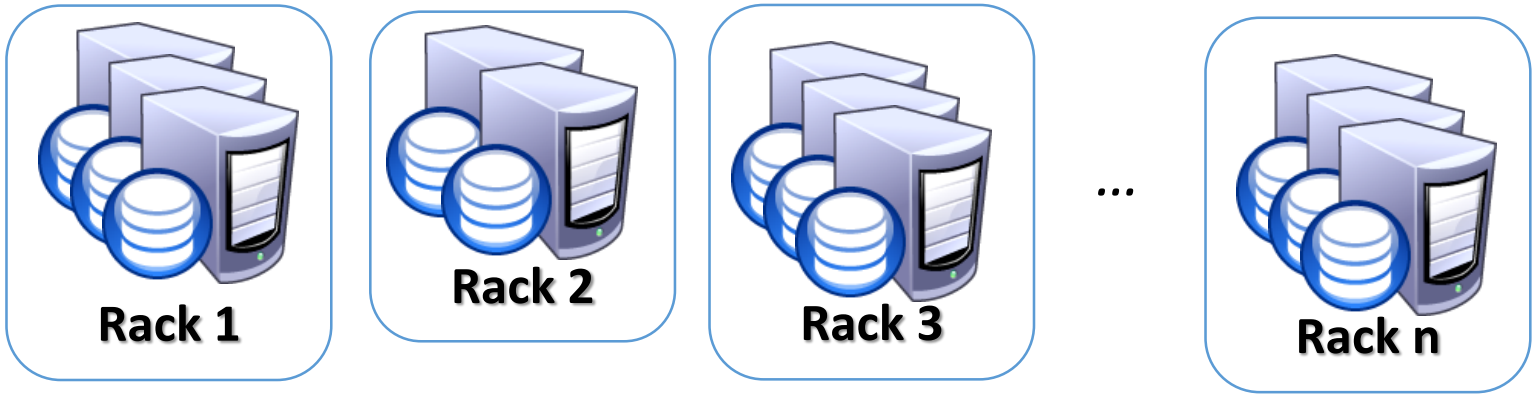
  - Data
  - Metadata: checksum + generation timestamp

HDFS - Architecture

- Blocks
- **Data Nodes**

DataNode1: **B1**, B2, B3, …, B10
DataNode2: **B1**, B10, B11, …, B2
DataNode3: B3, B10, B11, …, **B1**
…
Replication factor – 3 (default)



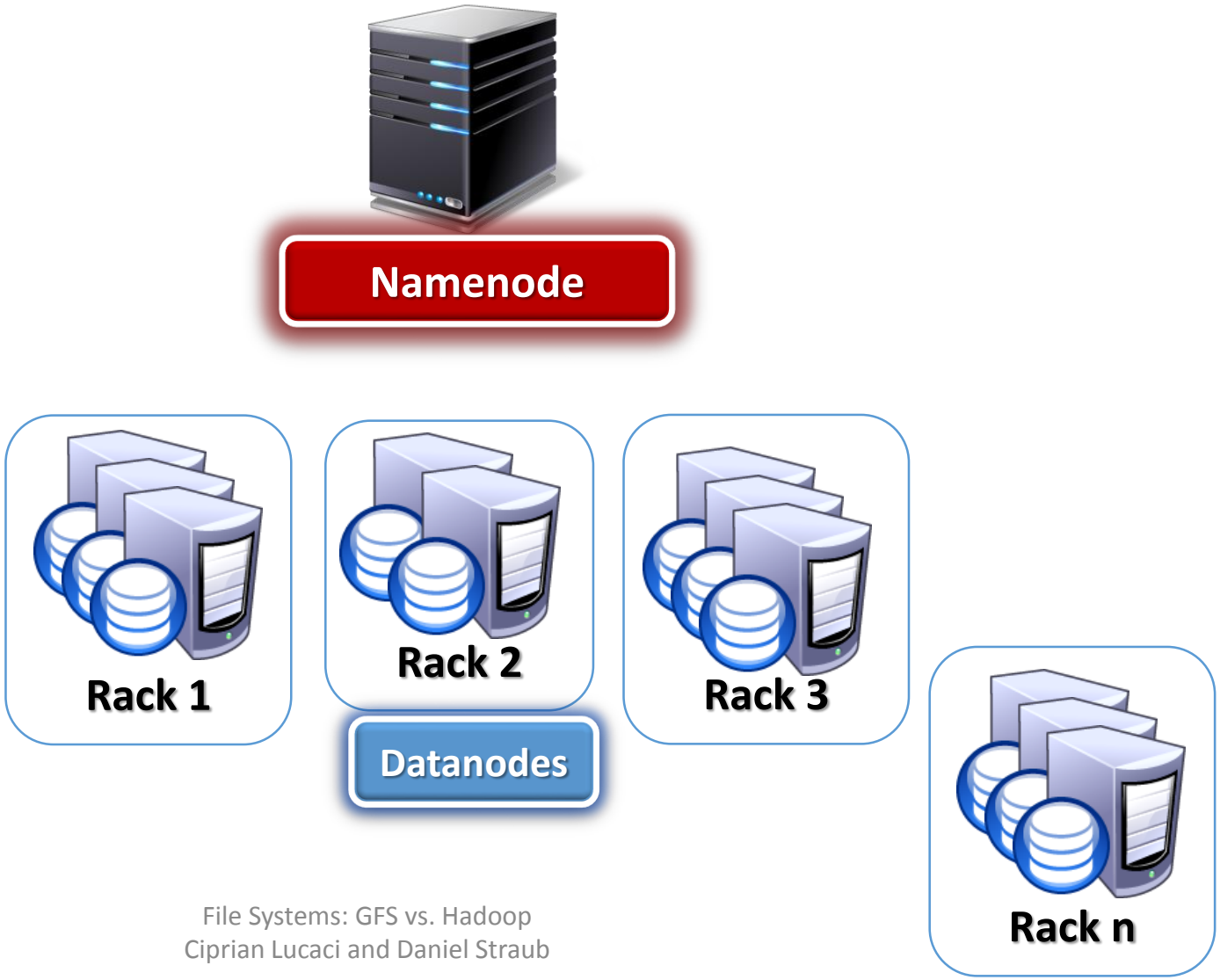


X1000s Data nodes

- Heartbeat
- Report blocks

HDFS - Architecture

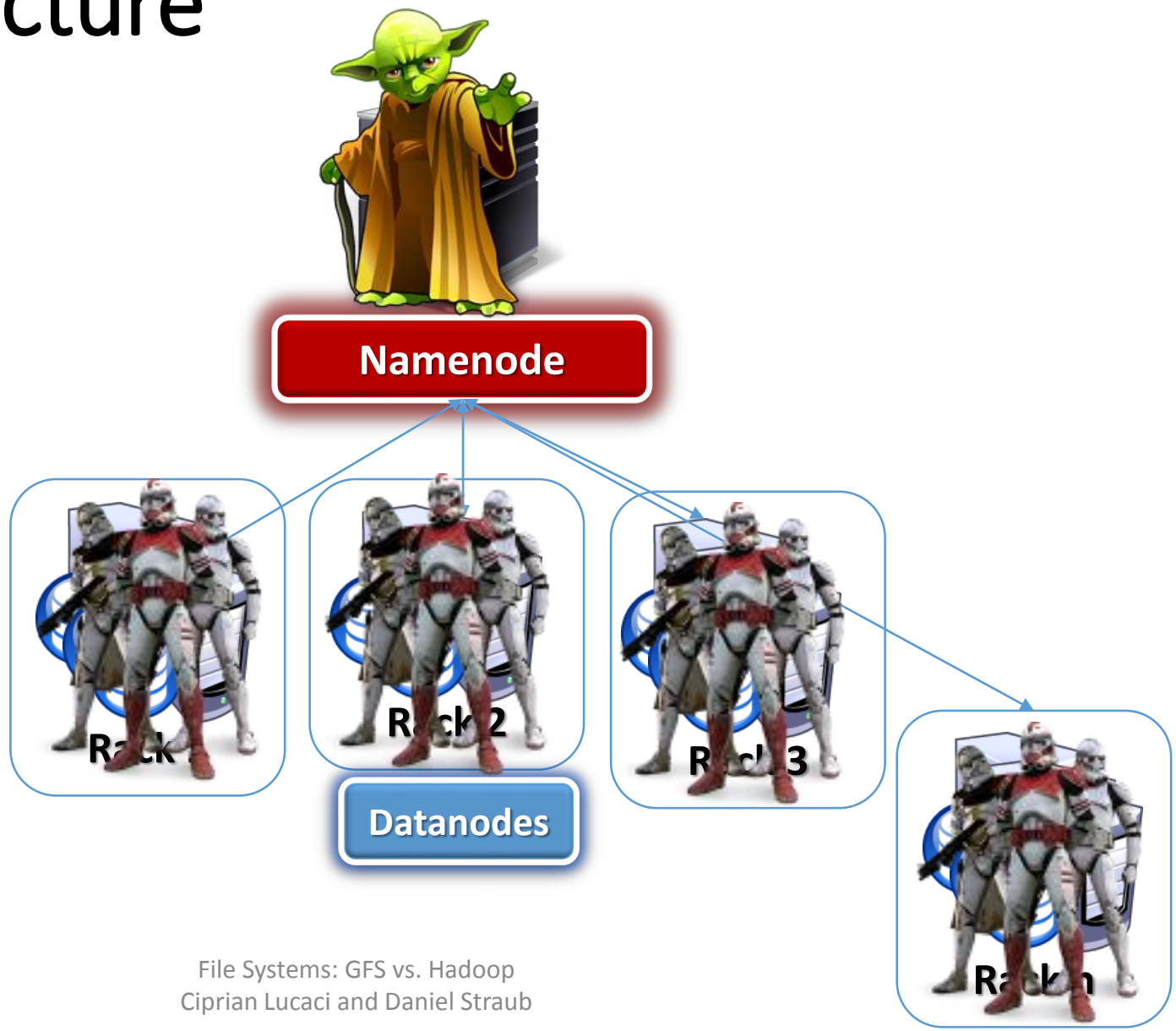
- Blocks
- Data Nodes
- **Name Node**



HDFS - Architecture

- Blocks
- Data Nodes
- **Name Node**

Master-worker pattern



HDFS - Architecture

Where is the namespace located?

- **Name Node**

Metadata
- Edit log
- System image
- Block checksum
- Block location
- Namespace tree

Namespace

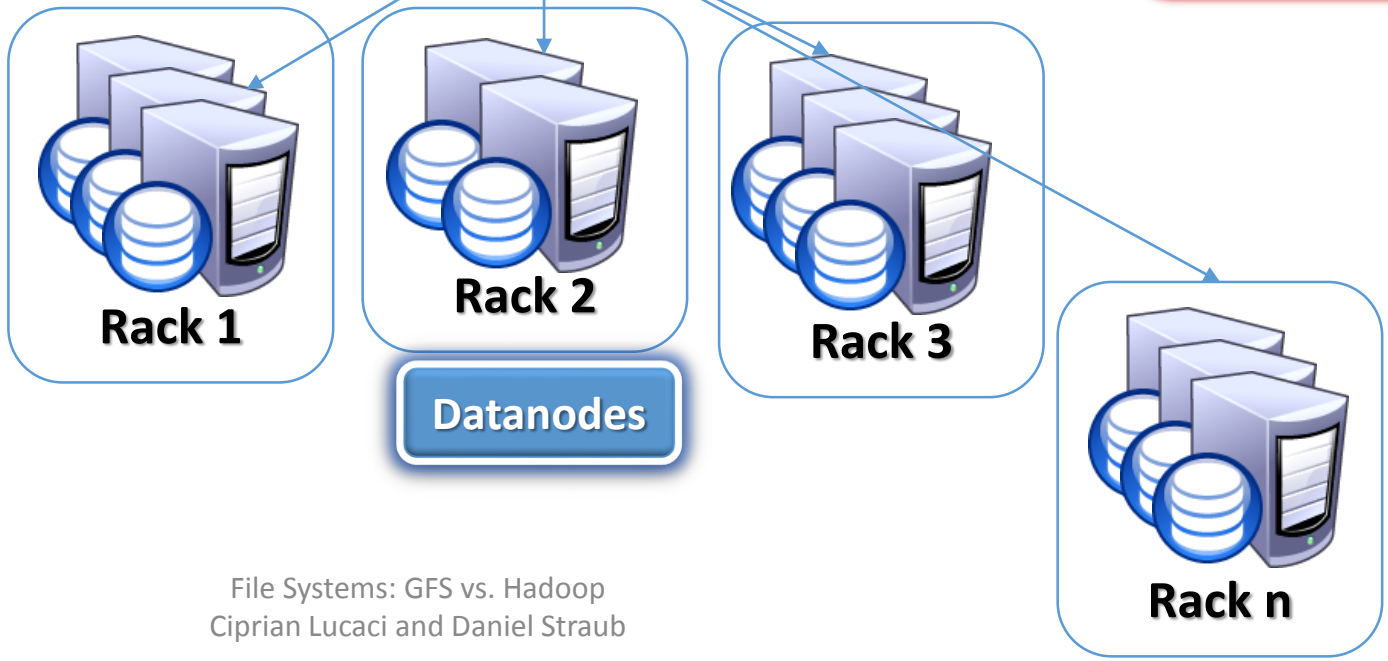
Filename: block-ids (node#block#)

/user/dir1/file1: n1b1, n1b2, n3b1, b4b3

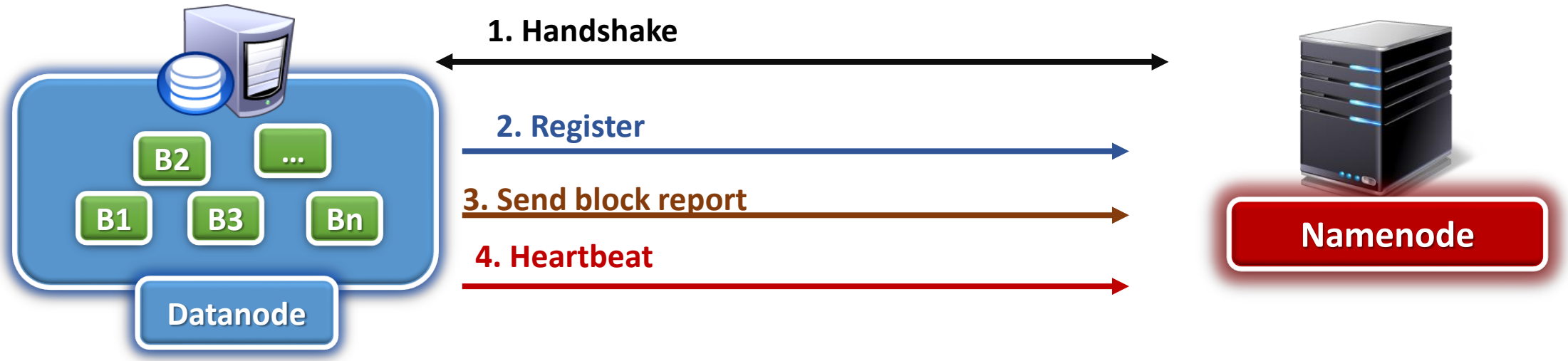
/user/dir2/file2: n3b7, n4b8, n1b6, b2b5

…

Inode
- Permissions
- Modification, access times



HDFS – Workflow: Startup



1. **Handshake**
- Namespace id
- Software version

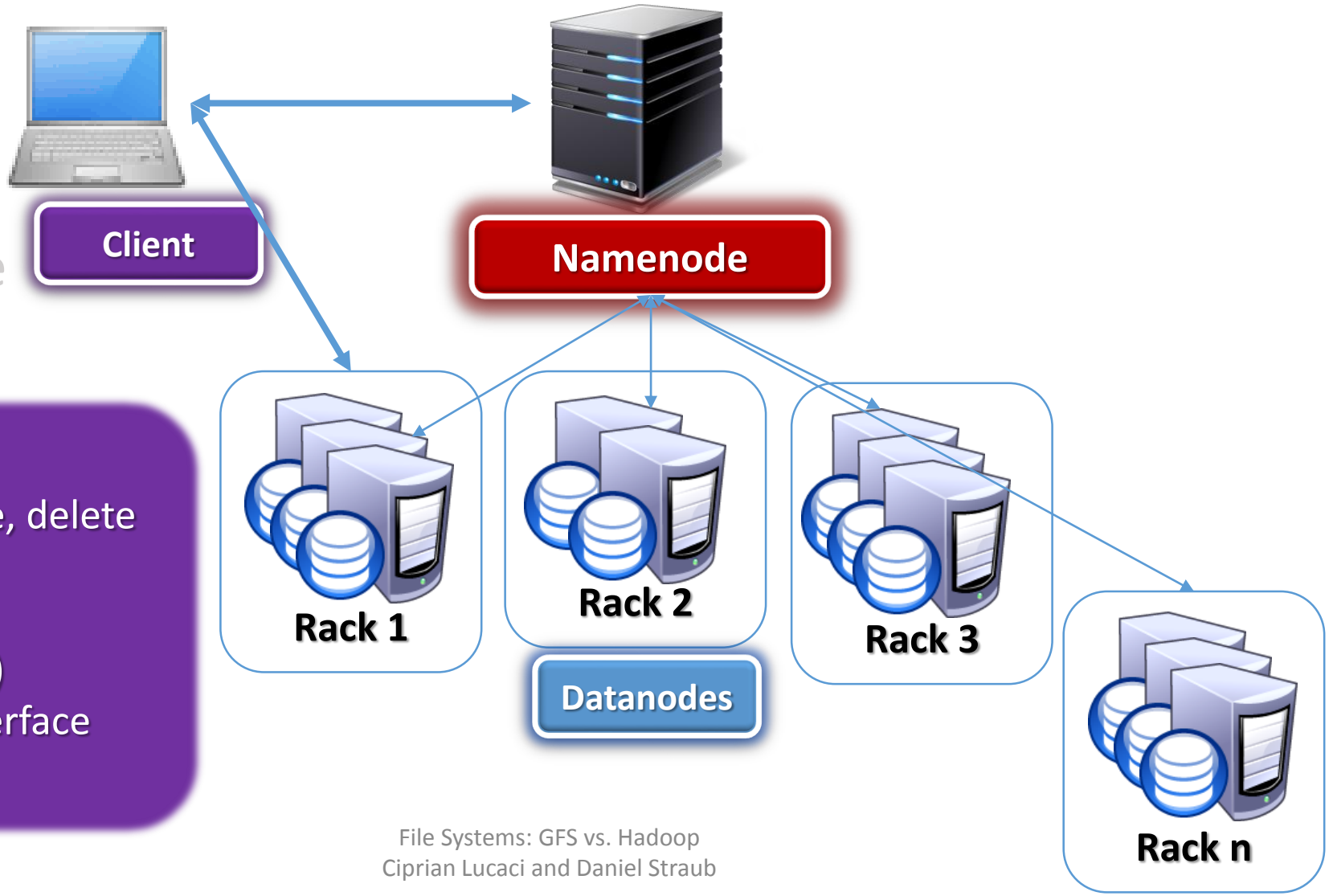
2. **Register**
- storage id

3. **Block Report**
- Block id
- Generation stamp
- Block length
- 1 / hour (default)

4. **Heartbeat**
- Every 3 seconds
- 10 min timeout
- Storage capacity
- Storage usage
- # current transfers

HDFS - Architecture

- Blocks
- Data Nodes
- Name Node
- **Client**

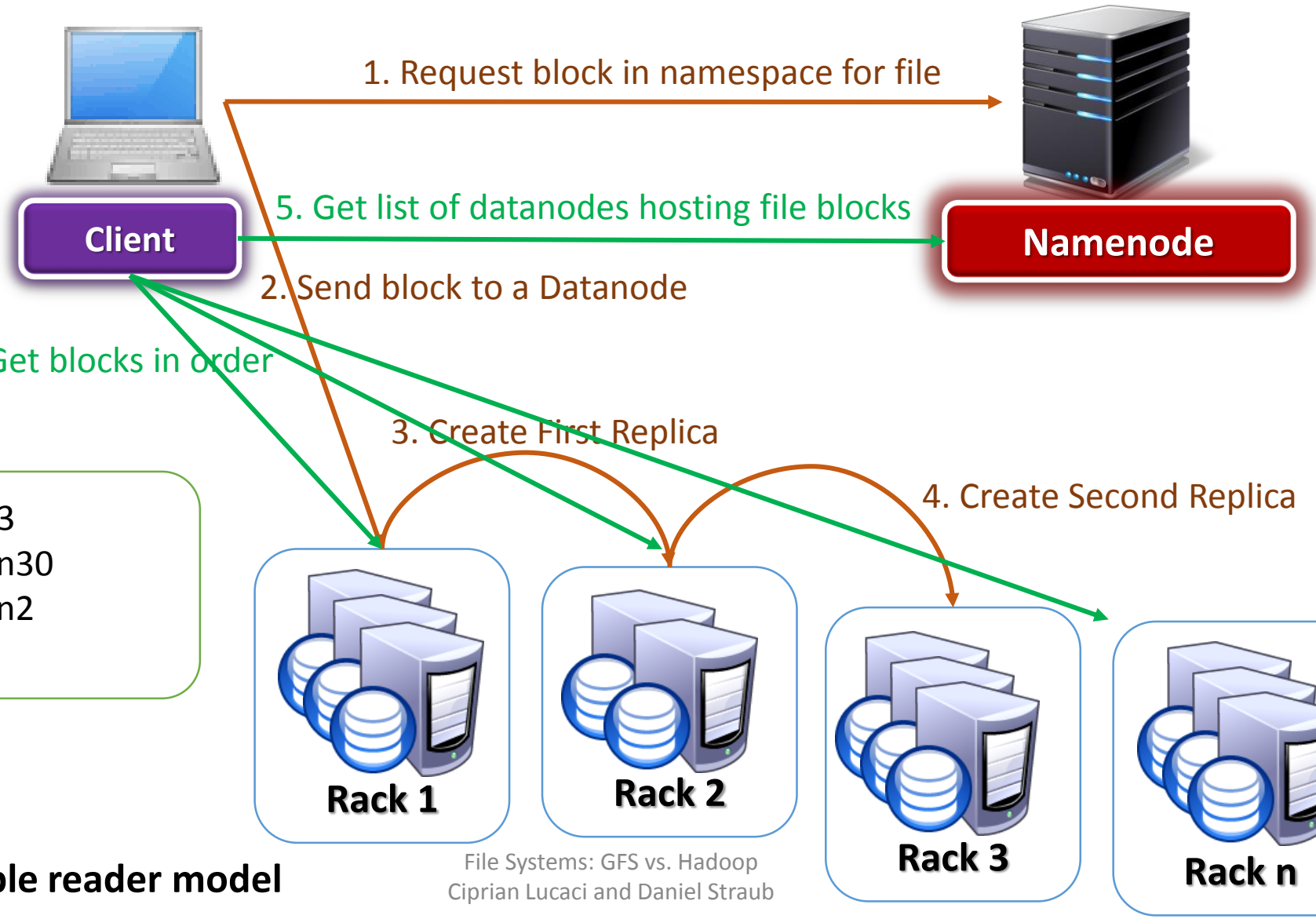


- read, write, create, delete
- files, directories

- Code library (Java)
- Exposes HDFS interface

HDFS - Workflow

- **Write**
- **Read**



Single-writer, multiple reader model

HDFS - Workflow

1: Request block in namespace for file

Read

```
Configuration configuration = new Configuration();
FileSystem hdfs = FileSystem.get( new URI( "hdfs://localhost:54310" ), configuration );
Path newFilePath = new Path("hdfs://localhost:54310/s2013/batch/table.html");

InputStreamReader isr = new InputStreamReader(hdfs .open(newFilePath ))
BufferedReader br=new BufferedReader(isr);
String line=br.readLine();
br.close();
```

Rack 1

Rack 3

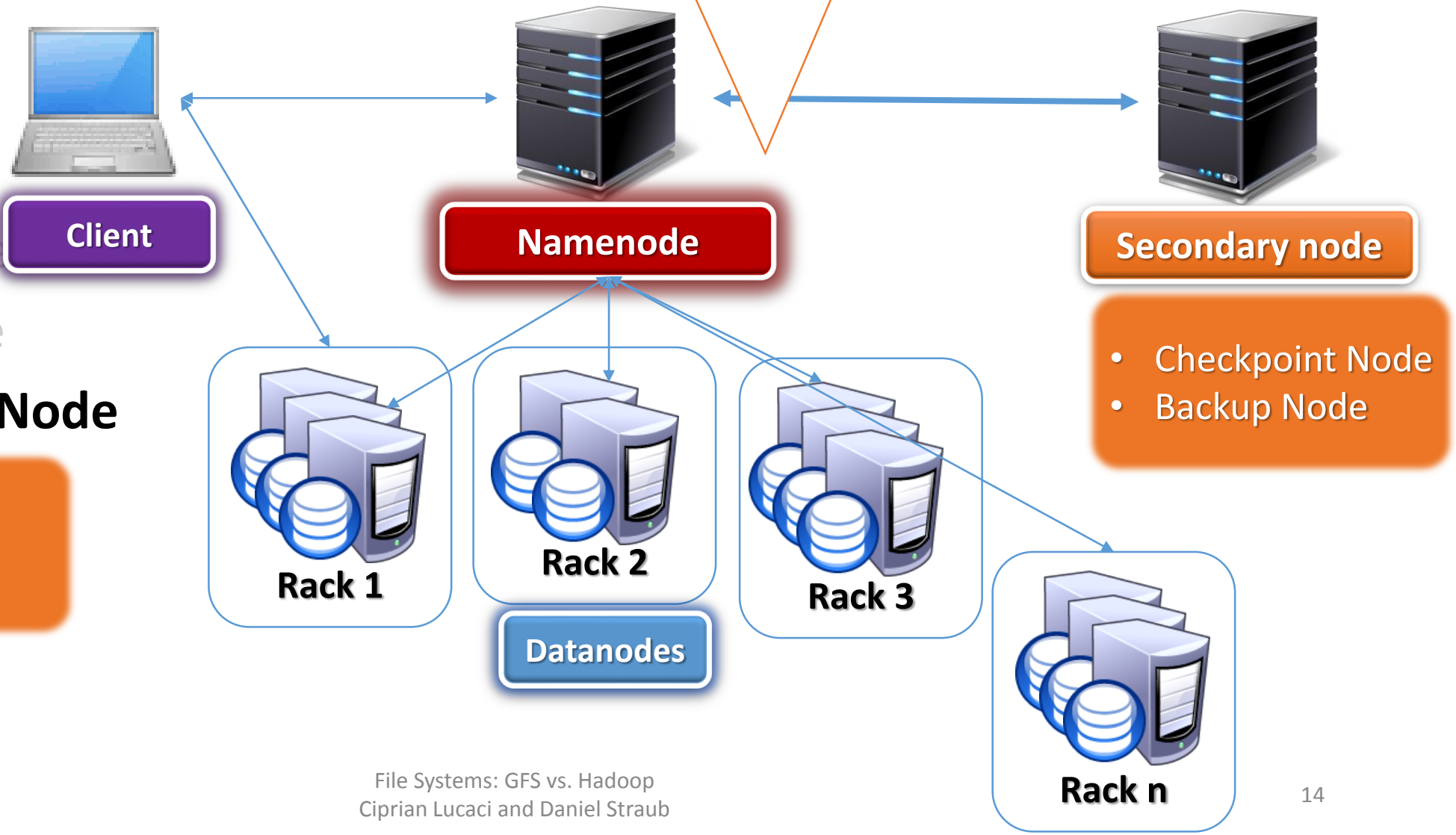
Rack n

Single-writer, multiple reader model

HDFS - Architecture

- Blocks
- Data Nodes
- Name Node
- Client Node
- **Secondary Node**

  - Edit log
  - System image



HDFS - Features

How would you place the replicas?

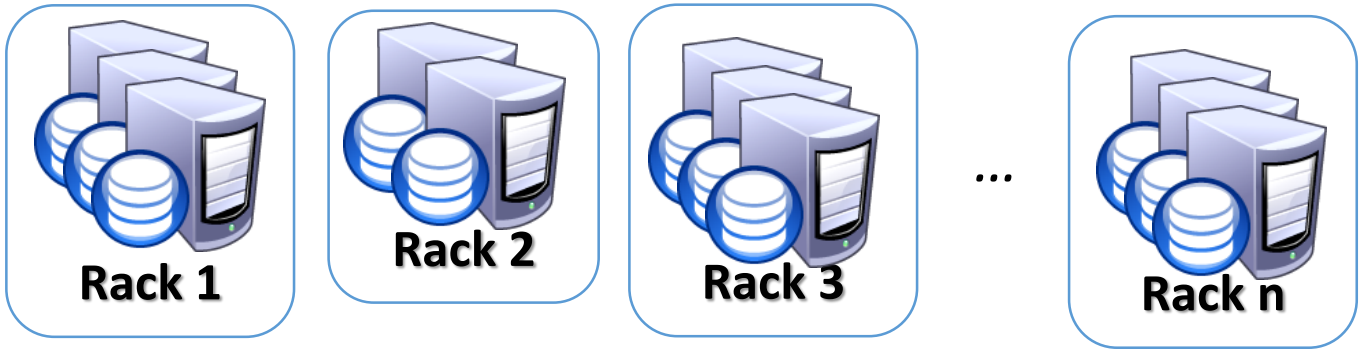
- **Block placement policy**

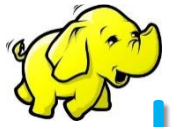
1. **No Datanode** contains **more than one replica** of any block.
2. **No rack** contains **more than two replicas** of the same block (if sufficient racks on the cluster)

- **Replica management** (@NameNode)
  - Optimize disk usage
  - Replication factor
  - Rack awareness

- **Balancer** (@NameNode)
  - Disk space usage
  - Moves replicas

- **Block scanner** (@DataNode)





HDFS - Purpose

- Optimized
  - Large files
  - Commodity hardware
  - Enable streaming
  - Batch processing
  - Multiple reads
- Not Optimized
  - Big amount of small files
  - Concurrent modification
  - Arbitrary modification - appends only
  - General purpose applications
- Cross platform: Java, Thrift, Rest
  - Web access, console …
  - opensource

amazon

LinkedIn

The New York Times

twitter

ebay

facebook

IBM

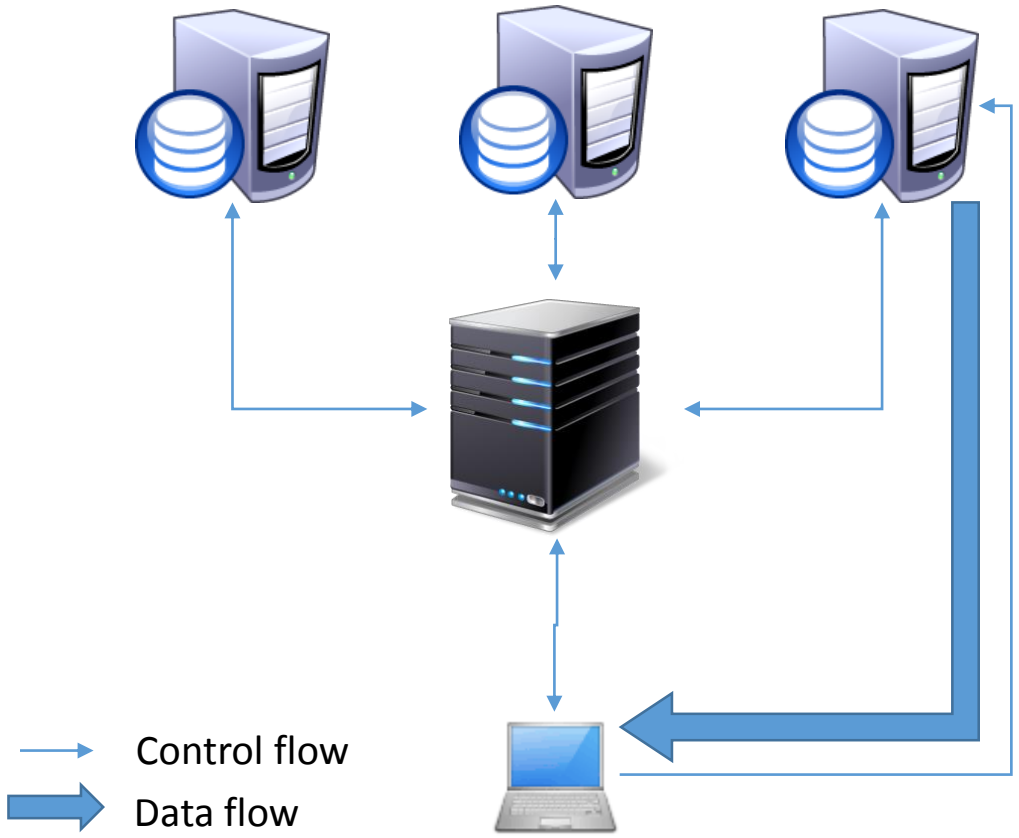
Spotify

YAHOO!

GFS - Purpose

- Large files (100MB and more)
- Commodity hardware (failures are the norm)
- (Concurrently) appending files
- Sequentially reading files
- High data throughput

GFS - Architecture



Chunkserver 1
Chunk a
Chunk d
Chunk k
…

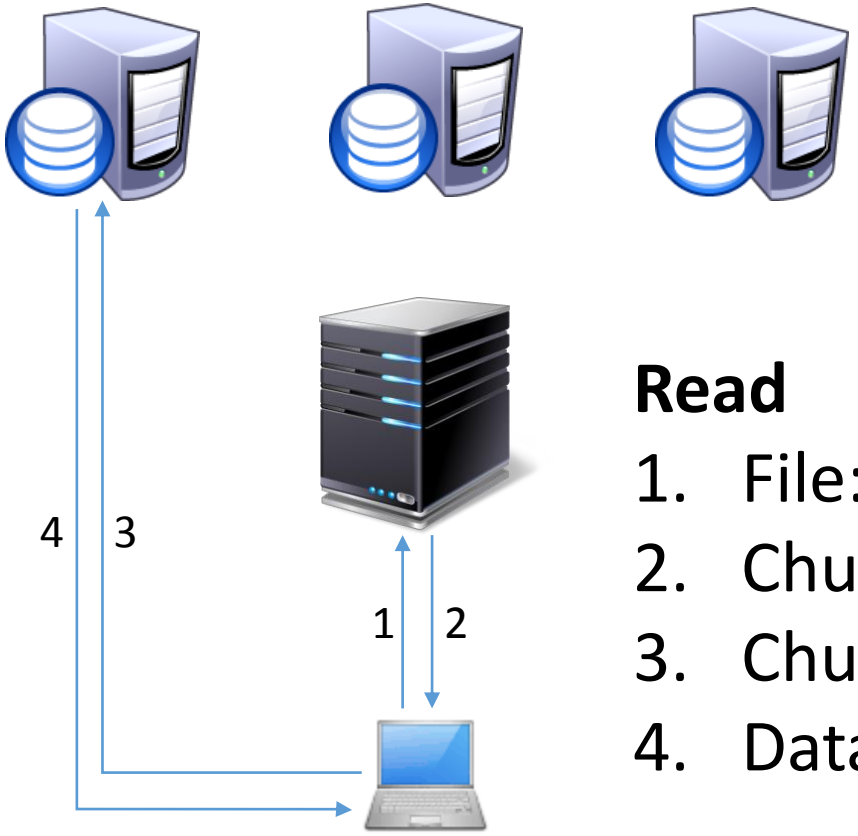
Chunkserver 2
Chunk b
Chunk d
Chunk e
…

Chunk: 64MB of data

Master
File1: chunk a, chunk c, chunk d, …
File2: chunk b, chunk x
…

Client

GFS – Workflow (Read)



CS 1	CS 2
Chunk a	Chunk b
Chunk d	**Chunk d**
Chunk k	Chunk e
…	…

Master

File1: chunk a, chunk c, chunk d, …

File2: chunk b, chunk x

…

Read

1. File: File1, chunk index: 3
2. Chunk d, location: CS 1, CS 2
3. Chunk d, byte range: 1-1000
4. Data: byte 1-1000

GFS – Leases

- CS may have leases for chunks
- Only CS with the lease for a chunk can modify it
- The master grants a chunk lease to one of the chunk‘s replicas

Does this affect CAP?
And if, how?

GFS – Workflow (Write)

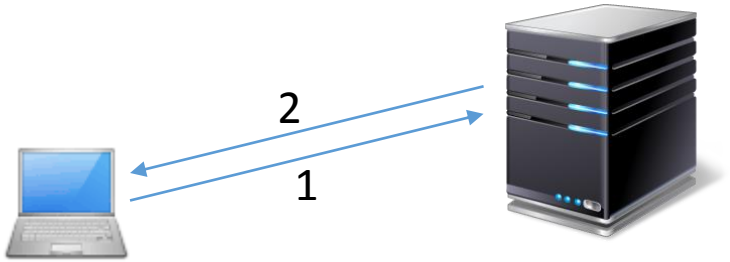
CS 1
Chunk c

CS 2
Chunk c

CS 3
Chunk c

1. Client:
   Where is chunk c, who has the lease

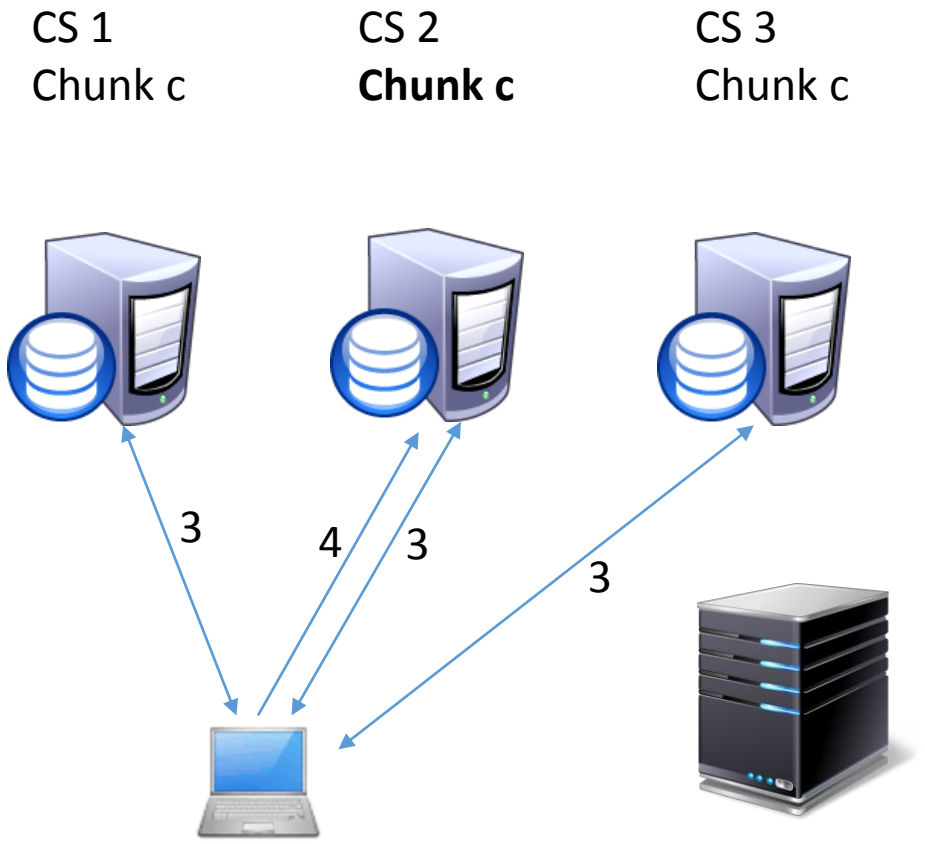
2. Master:
   CS2, CS1, CS3



Master
File1: chunk a, chunk c, …

Chunk d: CS1, **CS2**, CS3

GFS – Workflow (Write)



3. Client:
   Push data to CSs, stored in buffer.
   CSs ackknowledge reveiving data.

4. Client:
   Write request to primary.
   Primary applies write to itself.

Master
File1: chunk a, chunk c, …

Chunk d: CS1, **CS2**, CS3

GFS – Workflow (Write)

CS 1
Chunk c

CS 2
Chunk c

CS 3
Chunk c

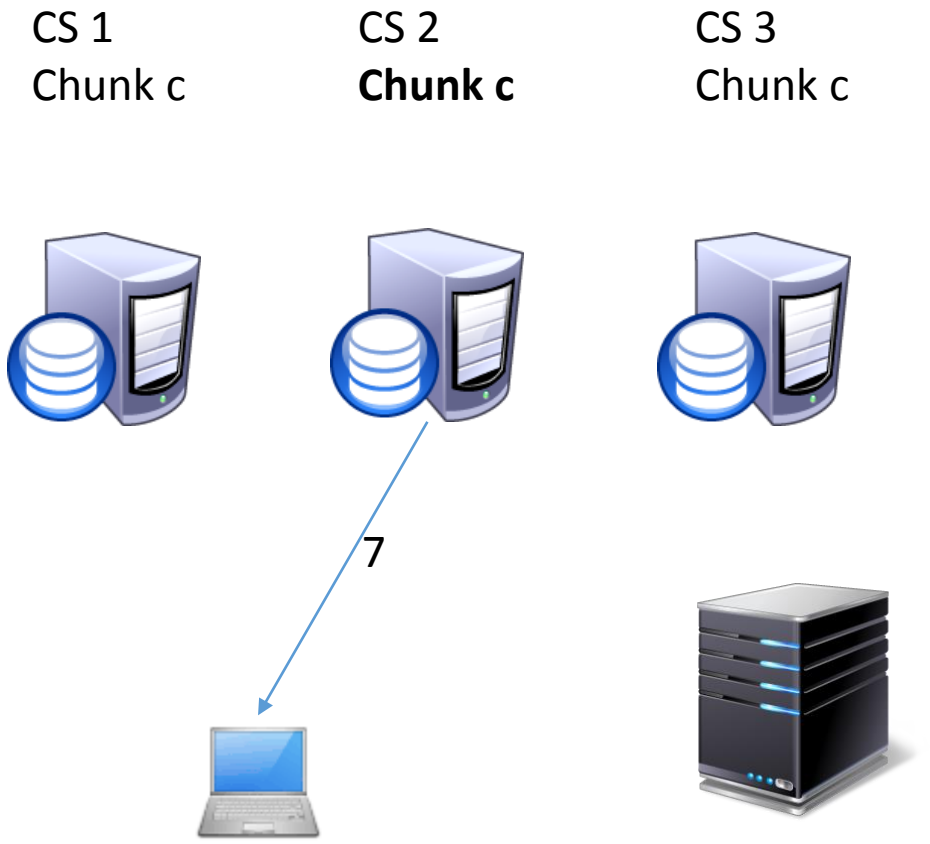
5. Primary:
   Forward write request to secondaries.
   Secondaries apply changes in the same order.

6. Secondaries:
   Indicate completion of write

Master
File1: chunk a, chunk c, …

Chunk d: CS1, **CS2**, CS3

GFS – Workflow (Write)



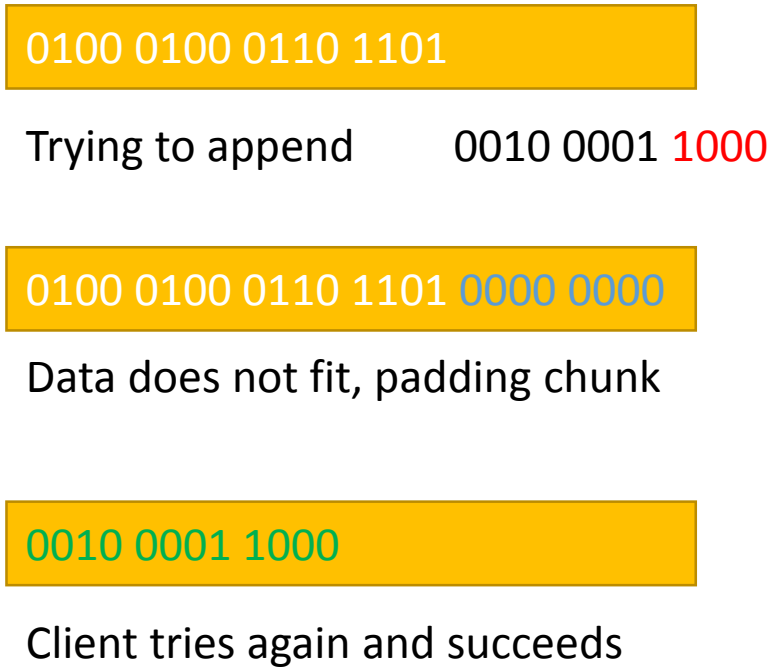
7. Primary:
   Returns result of operation to client.

   If there is an error, the client will retry writing from step 3.

   If error persists, client starts with step 1.

GFS – Workflow (Atomic Record Append)

- Differences to Write:
  3. Client pushes data to last chunk of file
  4. Client requests: Record append
  5. Primary checks if data fits in chunk



GFS – (In)consistency

- Inconsistency can be result of any write request.
- “GFS does not guarantee that all replicas are bytewise identical“

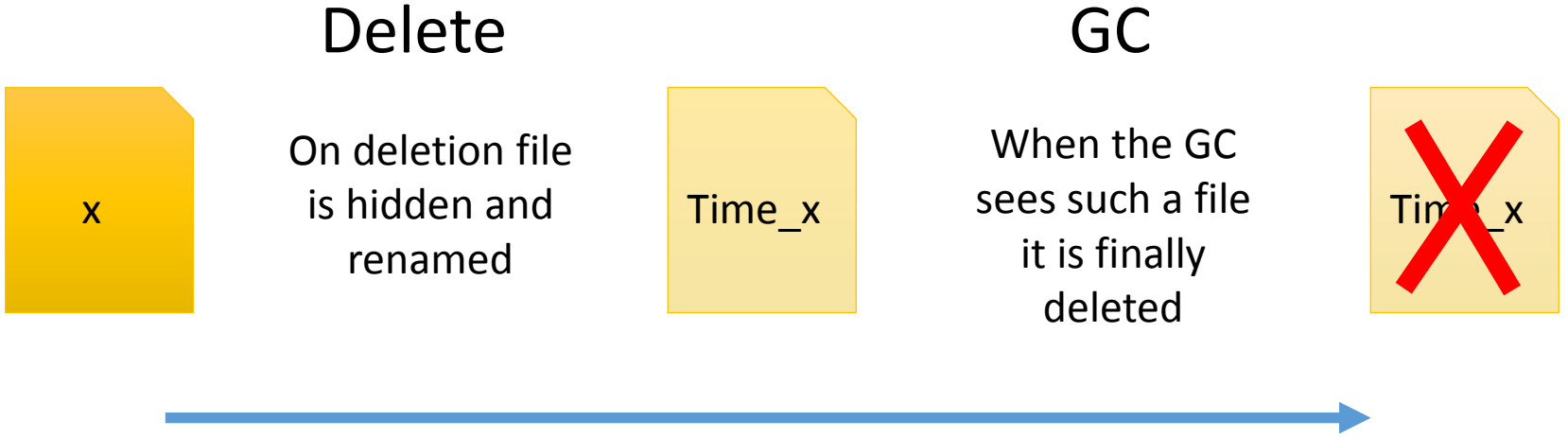
	Write	Record Append
Serial	Defined	Defined and partly inconsistent
Concurrent	Consistent	
Failure	Inconsistent	

- Inconsistency is handled by checksums

GFS – Garbage Collection

What are the advantages of GC ?

- Deleting files/chunks does not directly free memory:



- GC also removes non reachable chunks

GFS – Replicas

Creation

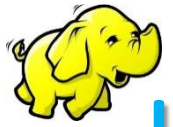
- Even disk space usage on CS
- Limited # of recent creations per CS

Re-replication

- If # of replicas < threshold, master creates new replicas

Rebalancing

- Improve disk space and load balancing



HDFS vs. GFS

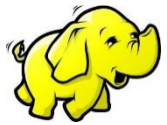
Similarities

- Large files
- Write once, read multiple times
- Appending of files
- Streaming of files
- Focus on AP (of CAP)

Differences

- Block id vs. Chunk index
- Concurrently appending to files
- Open source vs. Closed source
- Namenode vs. Master fail
- Permissions

References



- HDFS
  - K. Shvachko, H. Kuang, S. Radia, R. Chansler. The Hadoop Distributed File System. 2010
  - Thomas Kiencke. Hadoop Distributed File System (HDFS)
  - http://www.sas.com/en_us/insights/big-data/hadoop.html

- GFS
  - S. Ghemawat, H. Gobioff, S. Leung. The Google File System. 2003

- Comparison
  - R.Vijayakumari, R.Kirankumar, K.Gangadhara Rao. Comparative analysis of Google File System and Hadoop Distributed File System. 2014