SIC

Serviços e Infraestruturas de Computação

Hadoop, HDFS & MapReduce

A distributed framework for Big Data

Credits

Several slides and figures in this presentation are based on the following materials:

- The Adam Shook's UMBC materials available at https://redirect.cs.umbc.edu/~shadam1/491s16/lectures/02-History_HDFS_MR.pptx
- The Prof. Don Wang's materials available at https://www3.nd.edu/~dthain/courses/cse40822/fall2014
- The Apache Hadoop materials available at https://hadoop.apache.org

Lecture Outline

- An overview of Hadoop Basics
- HDFS (Hadoop Distributed Filesystem)
- MapReduce

Apache Hadoop

*"The Apache Hadoop software library is a framework that allows for the **distributed processing of large data sets across clusters of computers** using **simple programming models**.*

*It is designed to **scale up from single servers to thousands of machines**, each offering local computation and storage.*

*Rather than rely on hardware to deliver high-availability, the library itself is **designed to detect and handle failures at the application layer**, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures."*

[from: https://hadoop.apache.org]

Who uses Hadoop?

- A9/Amazon
- Adobe
- Alibaba
- Cloudspace
- Ebay
- Facebook
- Google
- …

An incomplete list is available at:
https://cwiki.apache.org/confluence/display/hadoop2/PoweredBy

Why Should I Care?
The short answer is…

The V's of Big Data!

An all-encompassing term for any collection of data sets so ***large*** *and* ***complex*** *that it becomes* ***difficult*** *to process using on-hand data management tools or traditional data processing applications*.

- Volume (petabyte scale)
- Velocity (social media, sensor, throughput)
- Variety (structured, semi-structured, unstructured)
- Veracity (unclean, imprecise, unclear)
- Value

Data Sources

- Social Media
- Web Logs
- Video Networks
- Sensors
- Transactions (banking, etc.)
- E-mail, Text Messaging
- Paper Documents
- …

Value in all of this!

- Fraud Detection
- Predictive Models
- Recommendations
- Analyzing Threats
- Market Analysis
- Others!

How do we extract value?
The classic approach

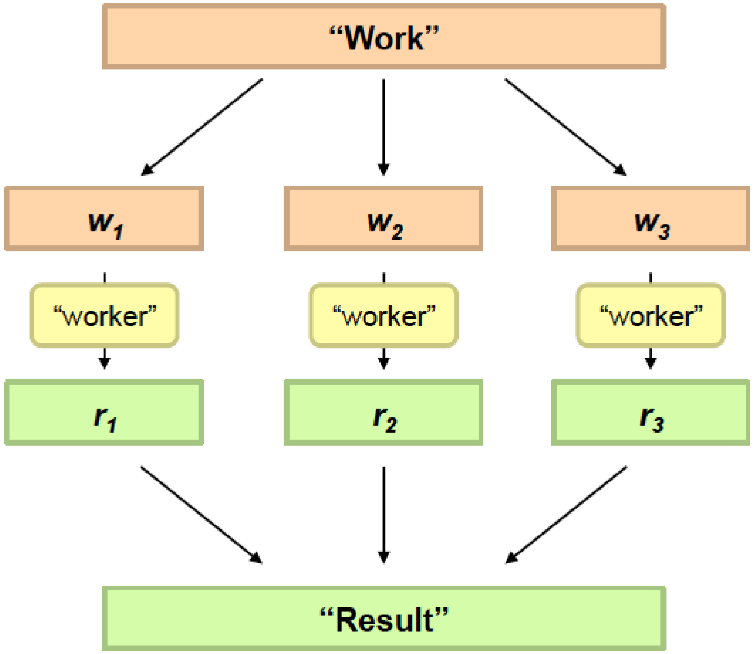
- Monolithic Computing
- Keep building bigger and faster computers

Limited solution

- Expensive
- Does not scale as data volume increases

Enter Distributed Processing

- Processing is distributed across many computers
- Distribute the workload to powerful compute nodes with some separate storage



Divide work, combine results

Enter Distributed Processing

- Processing is distributed across many computers
- Distribute the workload to powerful compute nodes with some separate storage

But also some new associated challenges:

- May fail to scale gracefully
- Hardware failure becomes more common
(due to the number of hardware components)
- Sorting, combining, and analyzing data spread across thousands of machines is not easy

Distributed processing is non-trivial

- How to assign tasks to different workers in an efficient way?
- What happens if tasks fail?
- How do workers exchange results?
- How to synchronize distributed tasks allocated to different workers?

Big data storage is challenging

- Data Volumes are massive
- Reliability of storing PBytes of data is challenging
- All kinds of failures: disk/hardware/network failures
- Probability of failures simply increases with the number of machines…

RDBMS is still alive!

- SQL is still very relevant, with many complex business rules mapping to a relational model

- But there is much more than can be done by leaving relational behind and looking at all the data

- NoSQL/non-relational databases can expand what we have, but have their own disadvantages:
  - Increased middle-tier complexity
  - Constraints on query capability
  - No standard semantics for query
  - Complex to setup and maintain

An Ideal Cluster…

- Linear horizontal scalability
- Analytics run in isolation
- Simple API with multiple language support
- Robust to hardware failures

An Ideal Cluster…

- Linear horizontal scalability
- Analytics run in isolation
- Simple API with multiple language support
- Robust to hardware failures

Enter Hadoop:

- Hits these major requirements
- Two core pieces
  - Distributed File System (HDFS)
  - Flexible analytic framework (MapReduce)
- Many ecosystem components to expand on what core Hadoop offers

Scalability

- Near-linear horizontal scalability
- Clusters can be built on commodity hardware
- Component failure is an expectation and is handled by design

Data Access

- Moving data from storage to a processor is expensive
- Store data and process the data on the same machines
- Process data intelligently by being local

Disk Performance

- Disk technology has made significant advancements

- Take advantage of multiple disks in parallel
  - 1 disk, 3TB of data, 300MB/s, ~2.5 hours to read
  - 1,000 disks, same data, ~10 seconds to read

- Distribution of data and co-location of processing makes this a reality

Complex Processing Code

- The Hadoop framework abstracts the complex distributed computing environment:
  - No synchronization code
  - No networking code
  - No I/O code

- MapReduce developer focuses on the analysis
  - Job runs the same on one node or on thousands of nodes

Fault Tolerance

- Component failure is inevitable, therefore it should be "planned for"

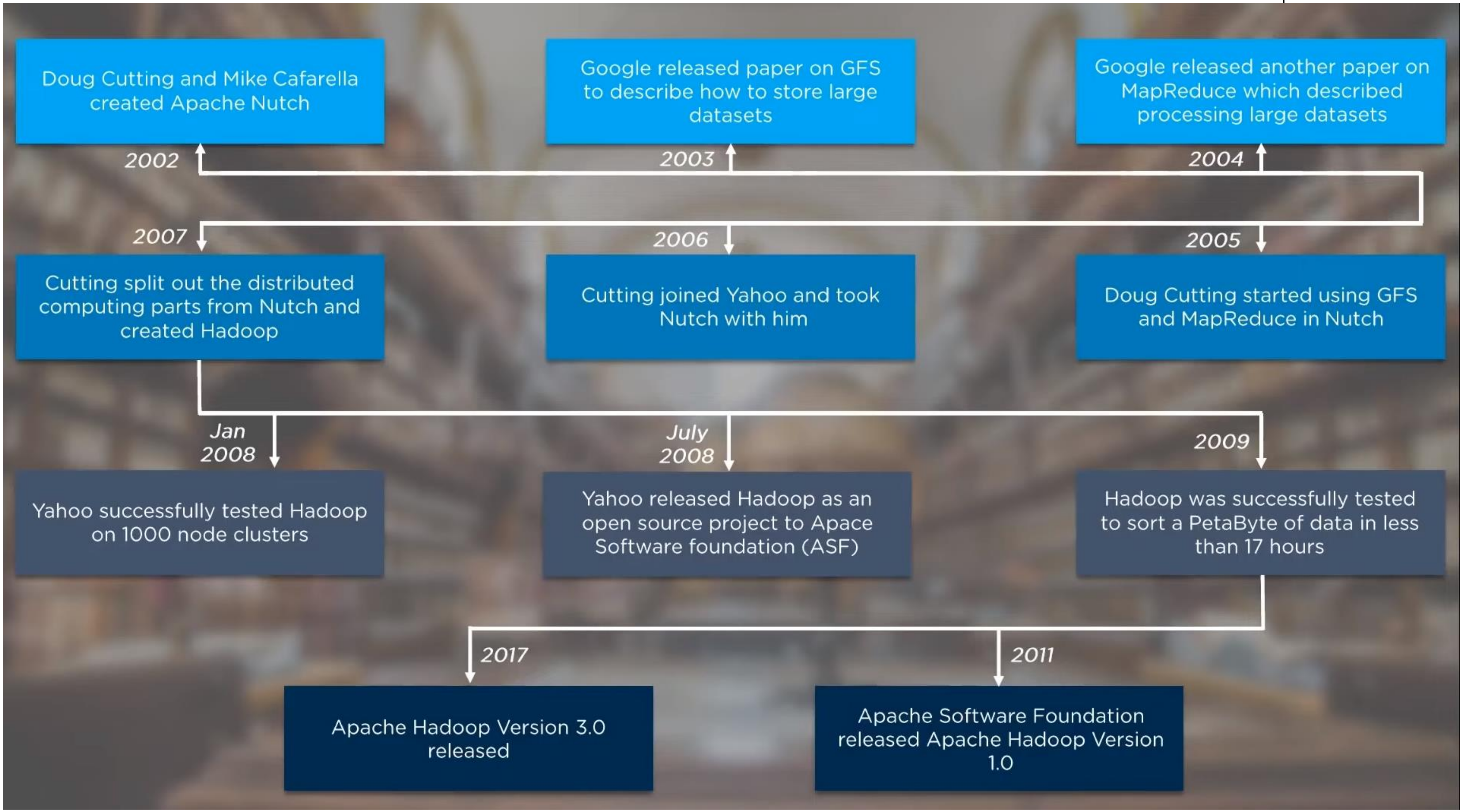
- Component failure is automatically detected and handled

- System continues to operate as expected with minimal degradation

Hadoop History (1/2)

- Spin-off from *Nutch*, an opensource web search engine
- Based on two Google Whitepapers
  - GFS (Google File System)
  - MapReduce
- *Nutch* re-architected lead to the birth of Hadoop
- Meanwhile, Hadoop became very mainstream

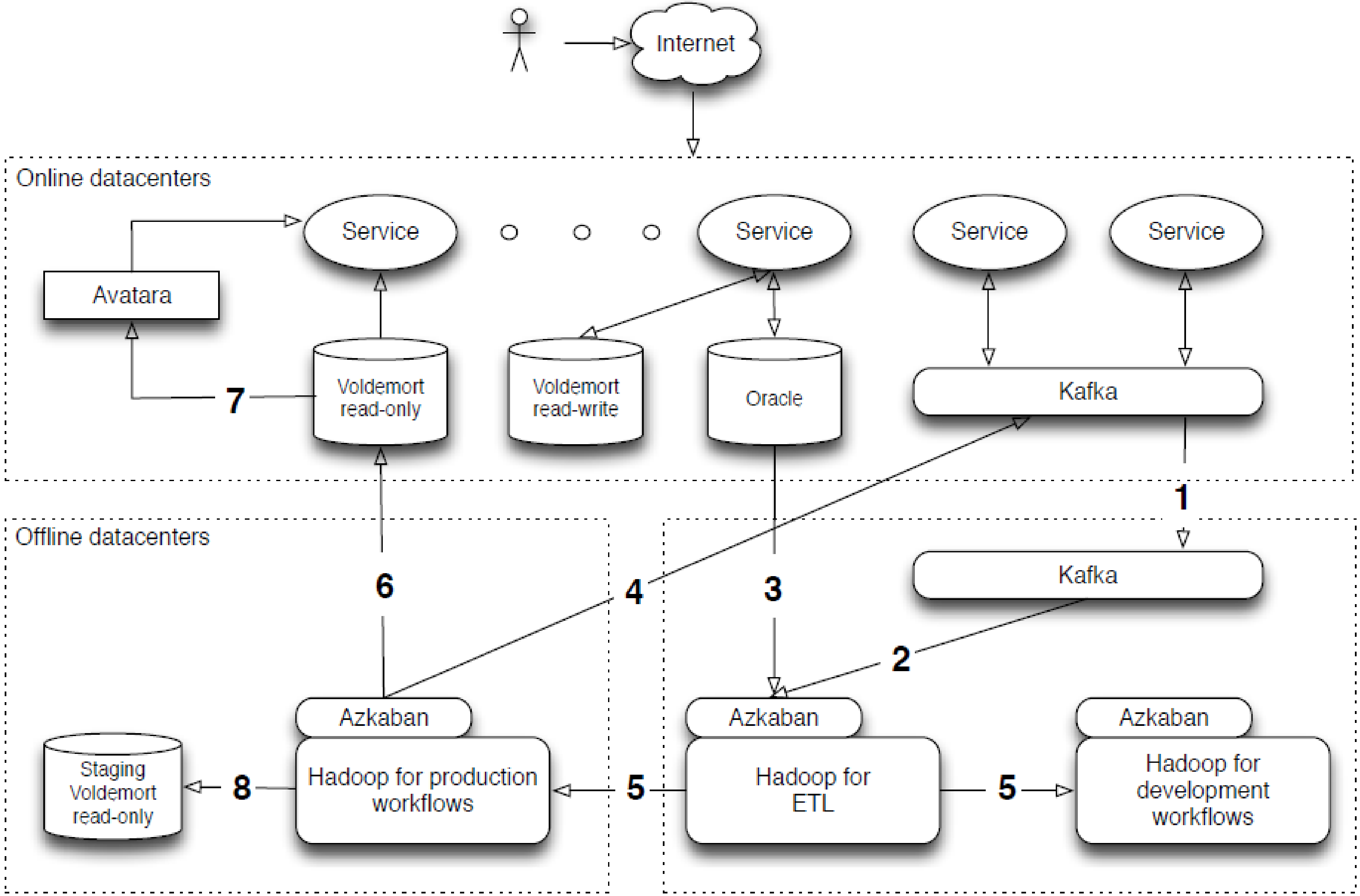
Hadoop History (2/2)



Common Use Cases

- Log Processing
- Image Identification
- Extract Transform Load (ETL)
- Recommendation Engines
- Time-Series Storage and Processing
- Building Search Indexes
- Long-Term Archive
- Audit Logging
- …

A Use Case Example
The LinkedIn Architecture



A Use Case Example
The LinkedIn Architecture

1. Two Kafka clusters kept in sync via mirroring supported in Kafka. Second cluster is for offline prototyping and data loading. Over 100 TB of compressed data for 300 topics, 15 billion message writes each day, 200 thousand messages / second. Deliver 55 billion messages each day.

2. Activity data ingests into Hadoop via Azkaban every 10 minutes, which is an open-source workflow scheduler. Event data consists of a stream of immutable activities or occurrences. Examples of event data include logs of page views being served, search queries, and clicks. Uses a schema registry to validate and reject data coming into Kafka. Older data schemas are automatically updated to new versions. Avro is the format

3. Core database snapshots are stored in Hadoop. Database data includes information about users, companies, connections, and other primary site data

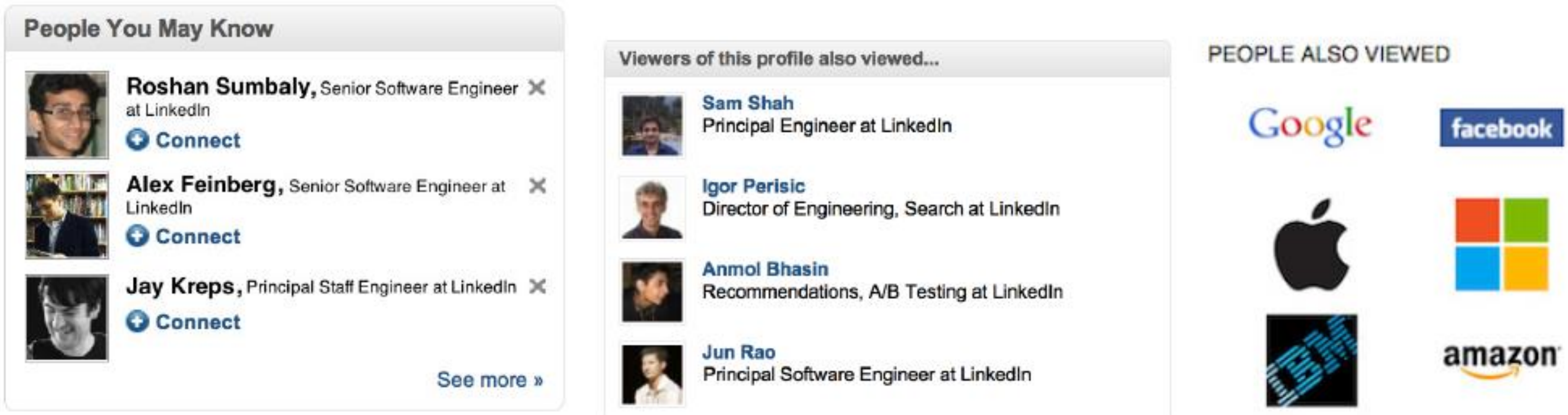
A Use Case Example
The LinkedIn Architecture

5. ETL jobs copy data to production and development systems for various workflows and perform extraction and transformation once. Daily job to combine and dedupe data throughout the day into another HDFS cluster, removing many small files

4/6/7. Production output can write to Kafka, Voldemort, or OLAP cubes to feed services These workflows are native MapReduce, Hive, or Pig jobs. Wrapper support for partition pruning. Takes one line of code to push output to these systems

8. Can push to staging clusters for debugging prior to going to the online systems

A Use Case Example
LinkedIn Applications



(a) "People You May Know"

(b) Collaborative Filtering

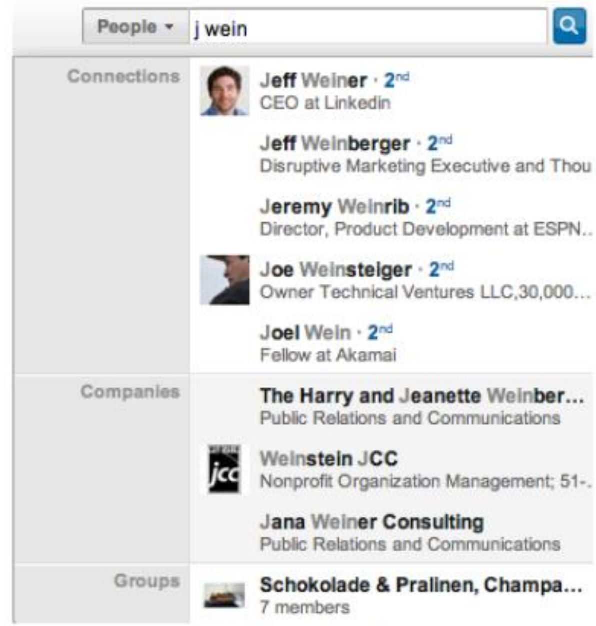
(c) Skill Endorsements

(d) Related Searches

A Use Case Example
LinkedIn Applications

(a) News Feed Updates

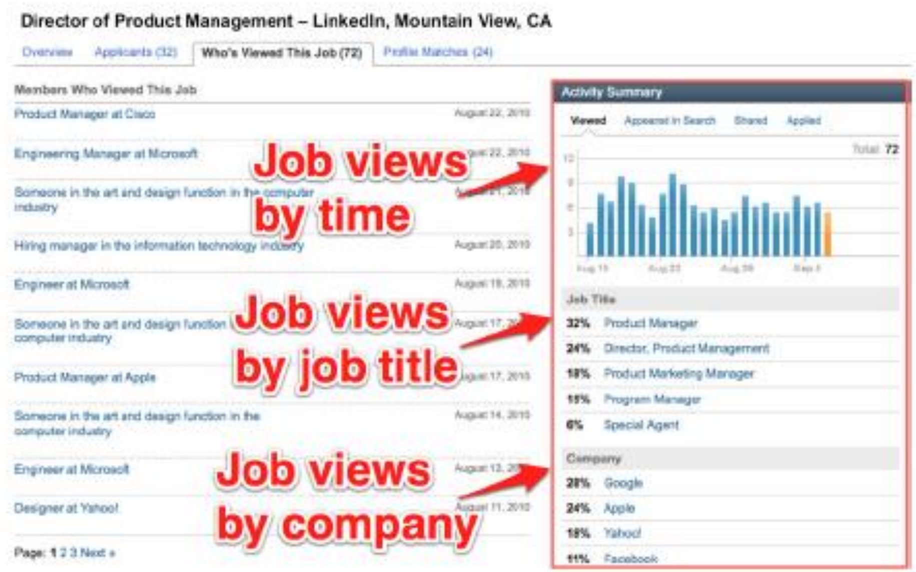
(b) Email



(c) Typeahead

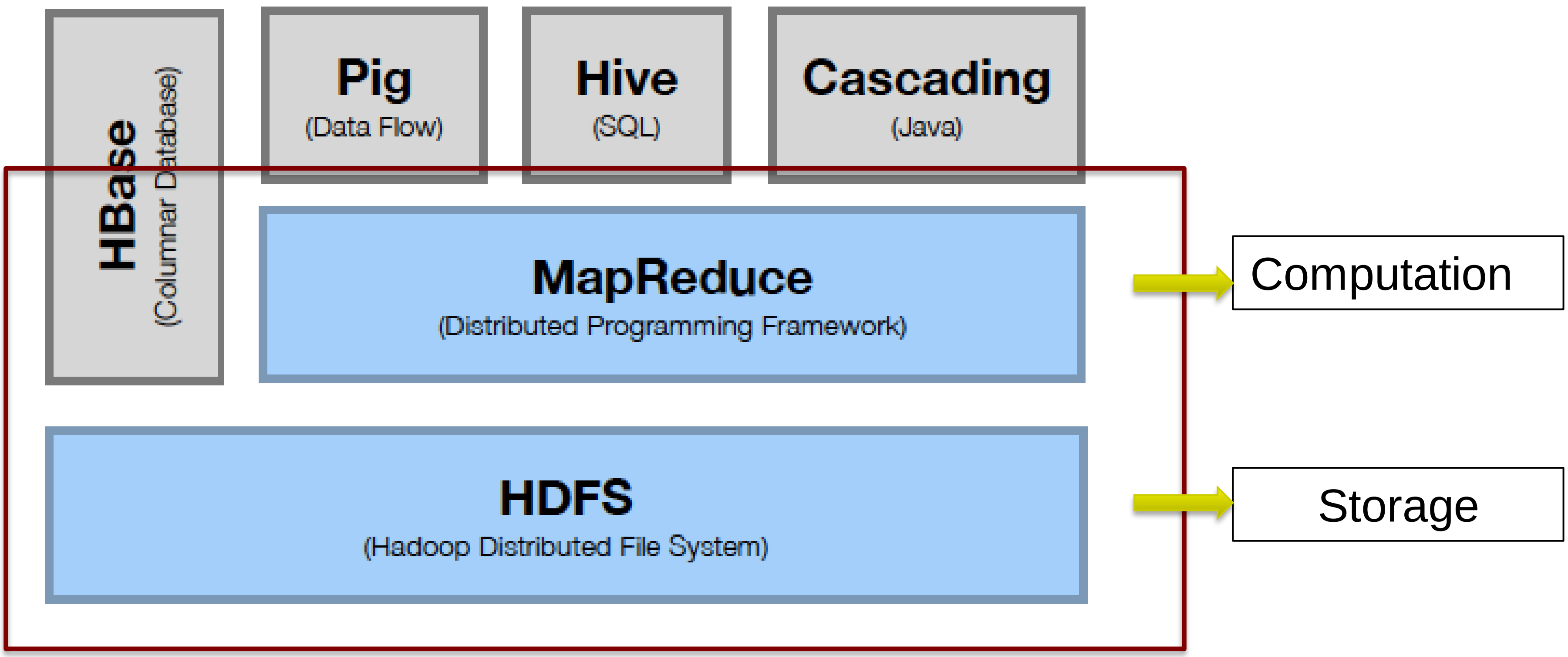
A Use Case Example
LinkedIn Applications

(a) "Who's Viewed My Profile?"



(b) "Who's Viewed This Job?"

Hadoop Stack



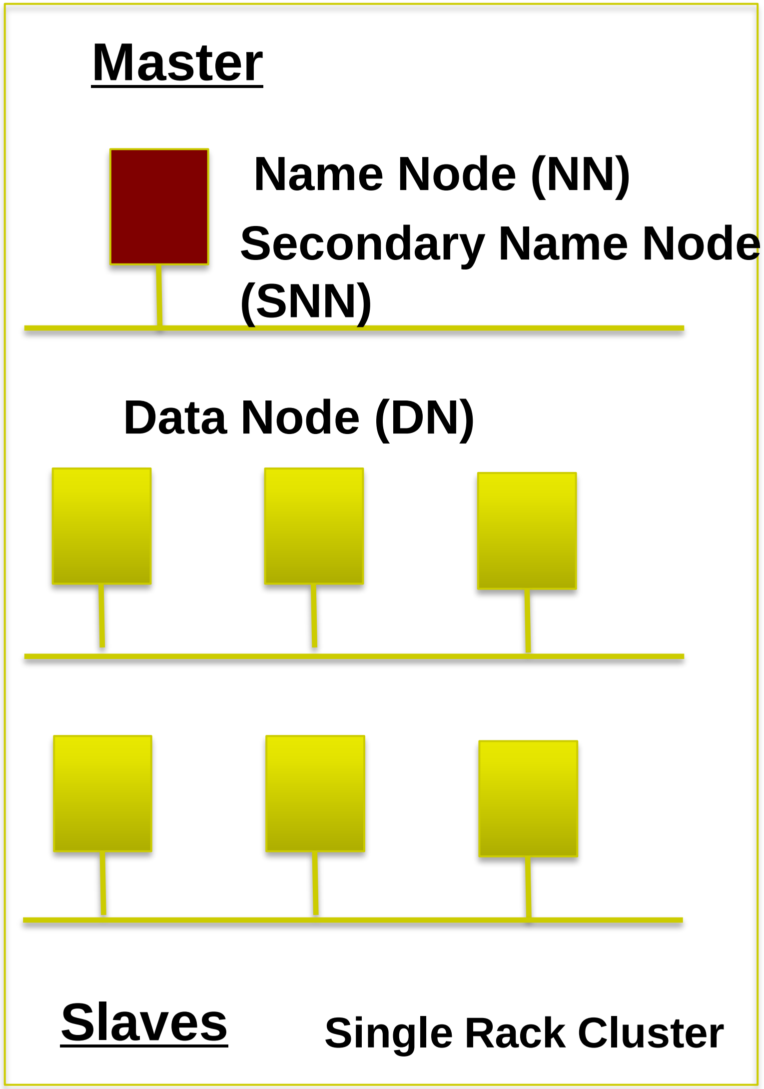
HDFS
Hadoop Distributed File System

- Inspired by Google File System (GFS)
- High performance file system for storing data
- Relatively simple centralized management
- Fault tolerance through data replication
- Optimized for MapReduce processing *(exposing data locality)*
- Linearly scalable
- Written in Java, APIs in all the useful languages
- Use of commodity hardware
- Files are “write once, read many”
- Leverages large streaming reads vs random
- Favors high throughput vs low latency
- Modest number of huge files

HDFS Architecture

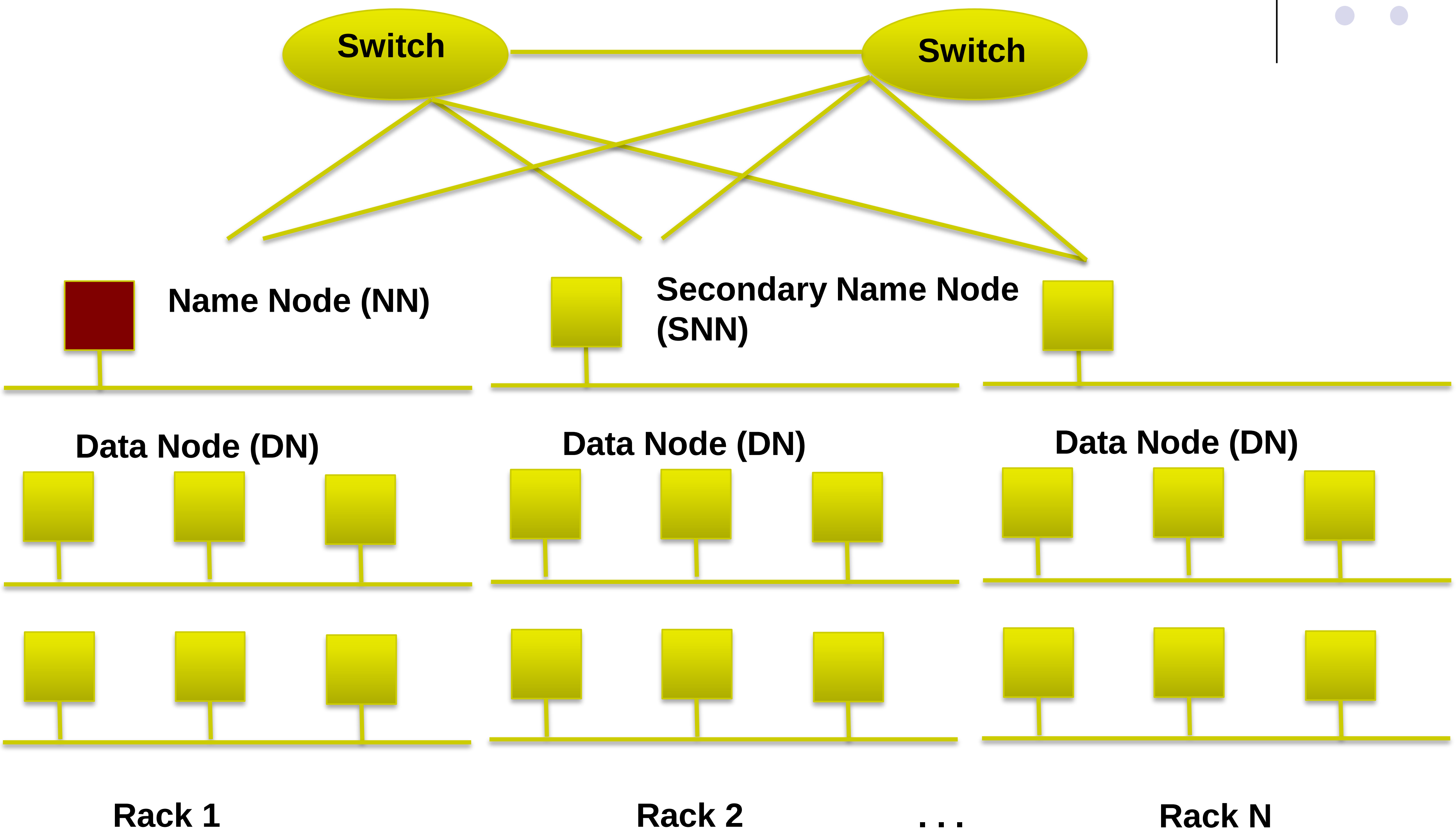
- Split large files into blocks
- Distribute and replicate blocks to nodes
- Two key services:
  - Master → NameNode
  - Many → DataNodes
- Backup/Checkpoint NameNode for HA

HDFS arch. (single rack cluster)

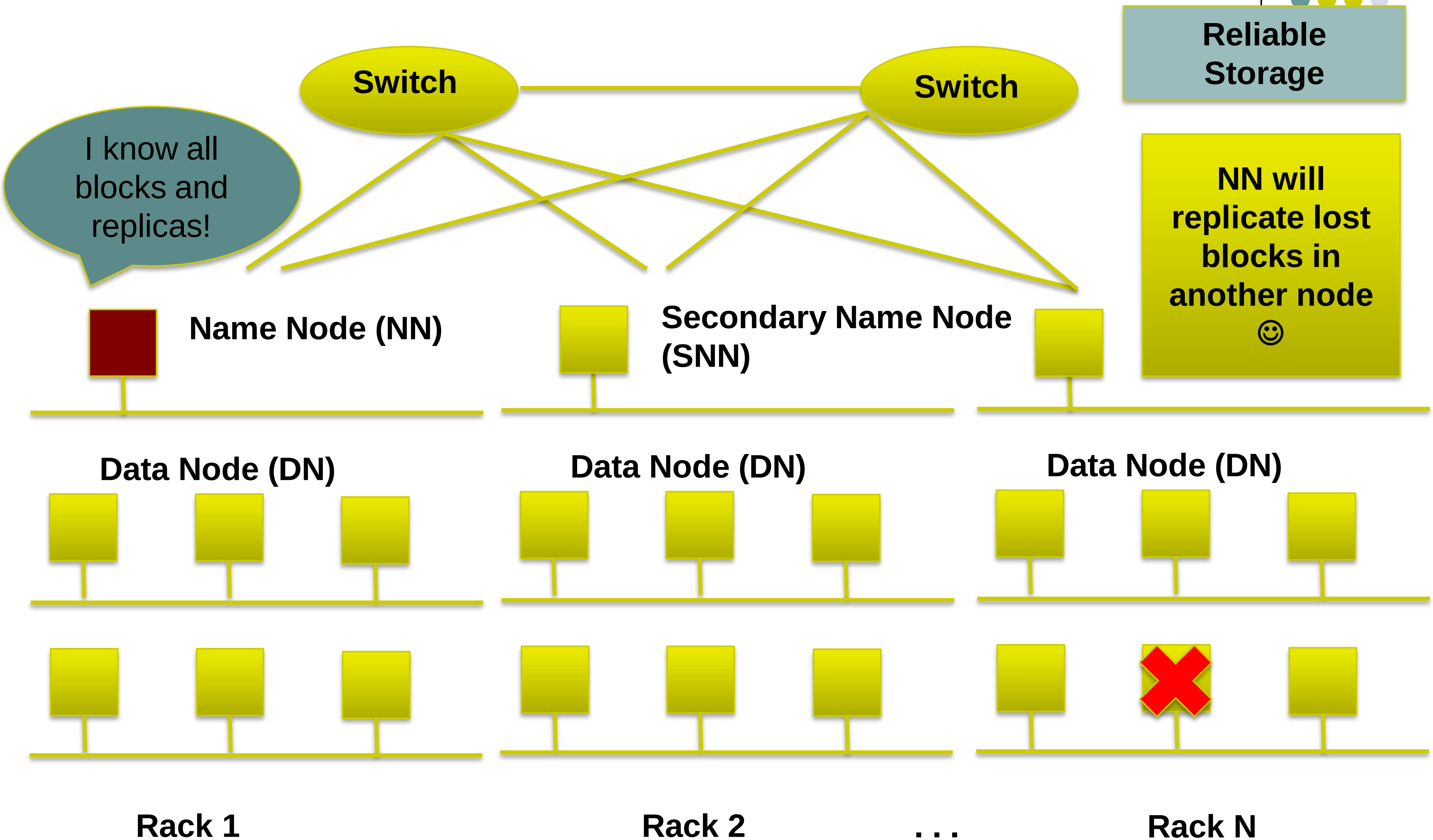


- Name Node: Controller
  - File System Name Space Management
  - Block Mappings
- Data Nodes: Work Horses
  - Block Operations
  - Replication
- Secondary Name Node:
  - Checkpoint node

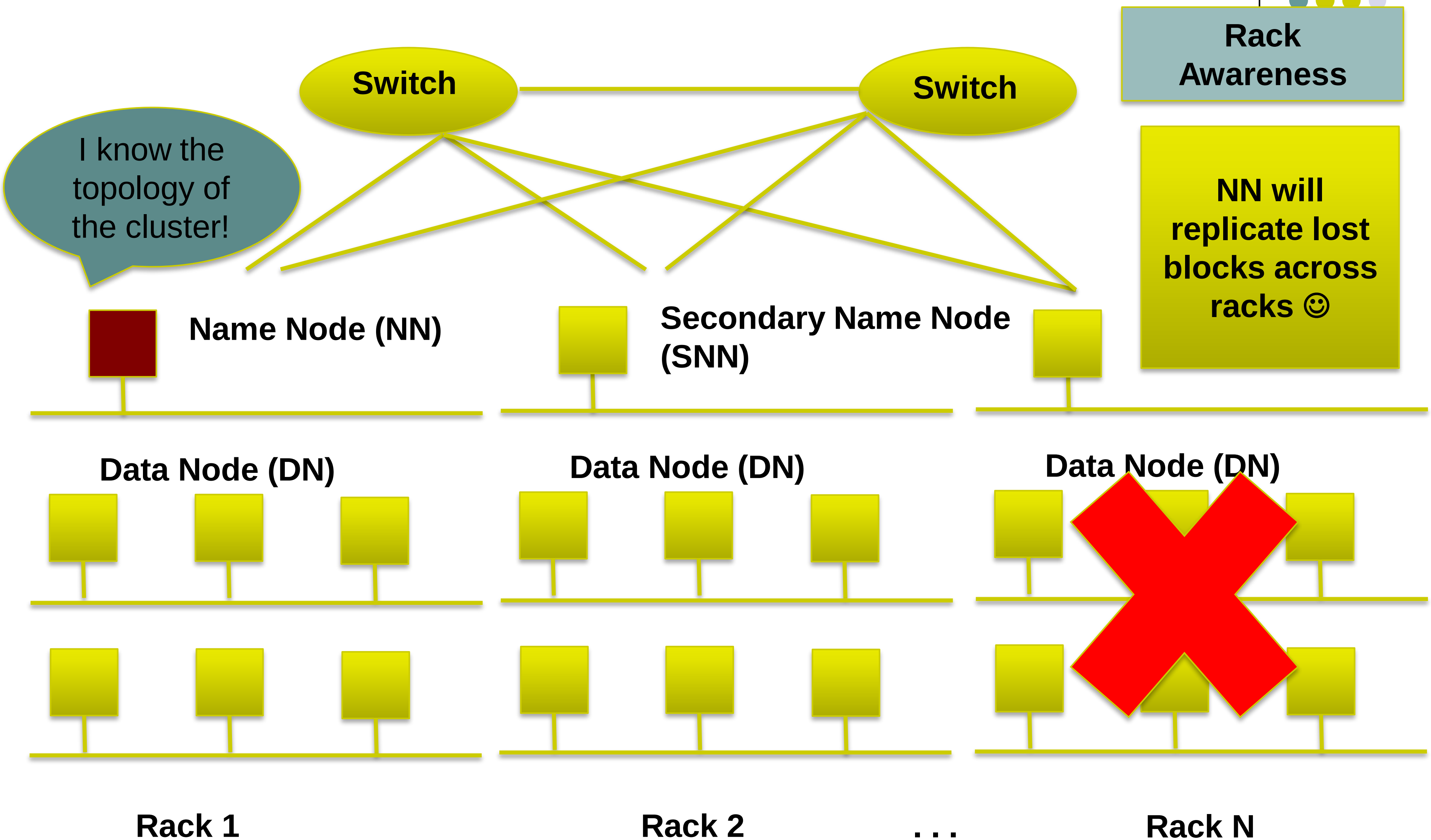
HDFS arch. (multiple rack cluster)



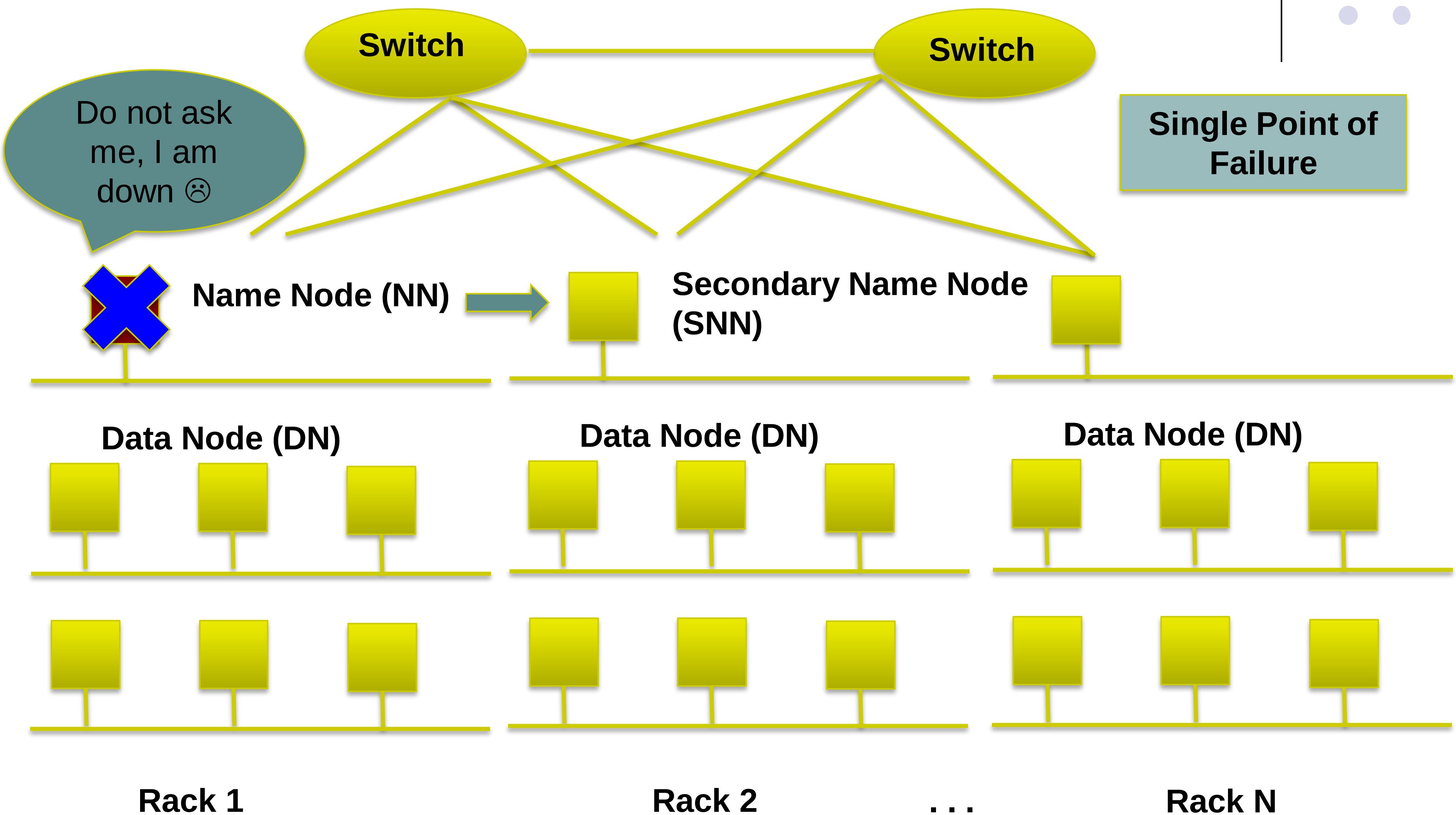
HDFS arch. (multiple rack cluster)



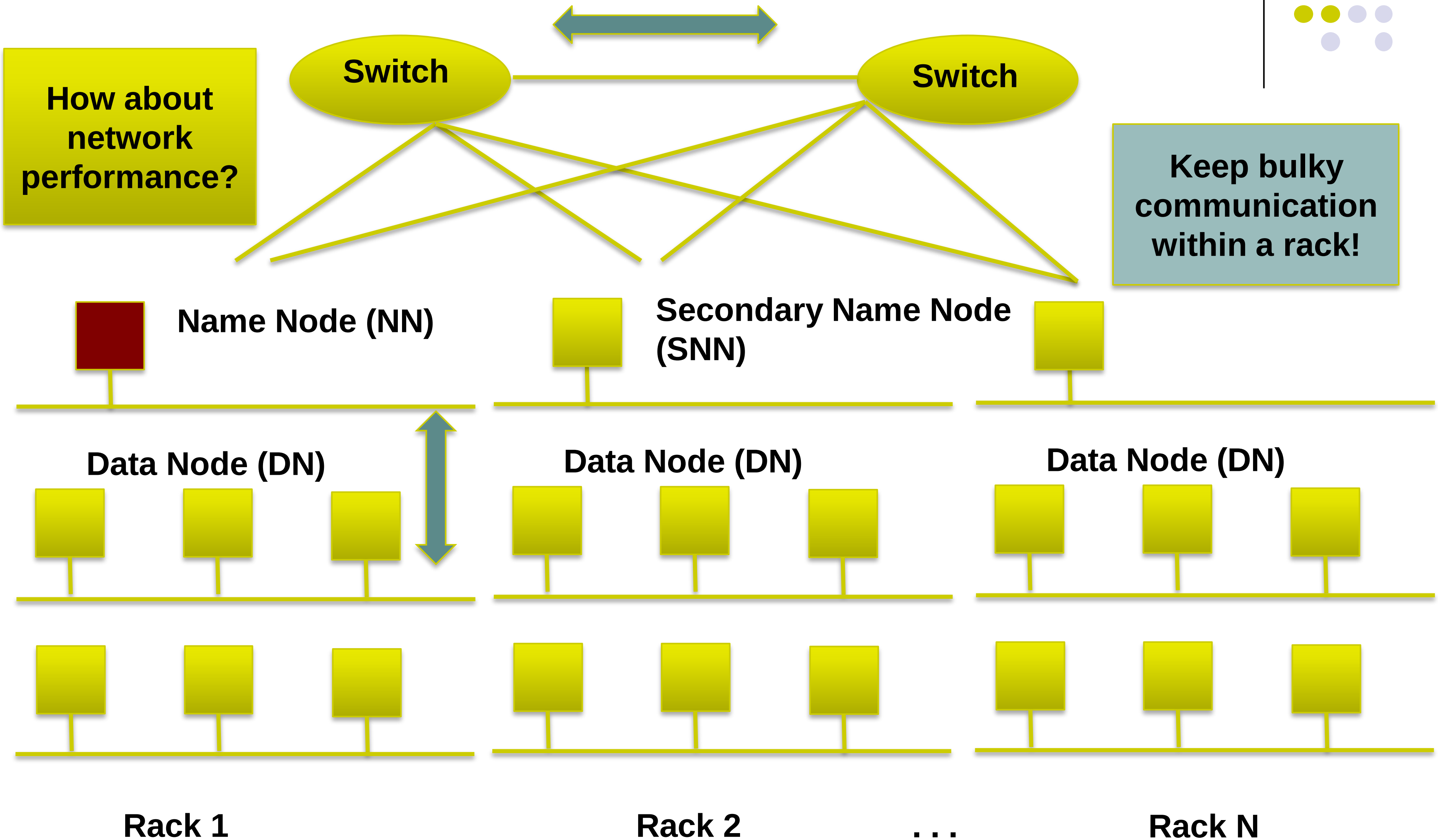
HDFS arch. (multiple rack cluster)



HDFS arch. (multiple rack cluster)



HDFS arch. (multiple rack cluster)



To be continued!

NameNode

- Single master service for HDFS
- Was a single point of failure
- Stores file to block location mappings in a *namespace*
- All transactions are *logged* to disk
- Can recover based on checkpoints of the namespace and transaction logs

Checkpoint Node (Secondary NN)

- Performs checkpoints of the *namespace* and *logs*
- Not (just) a hot backup!

- HDFS 2.0 introduced NameNode HA
  - Active and a Standby NameNode service coordinated via ZooKeeper

HDFS Inside: Name Node

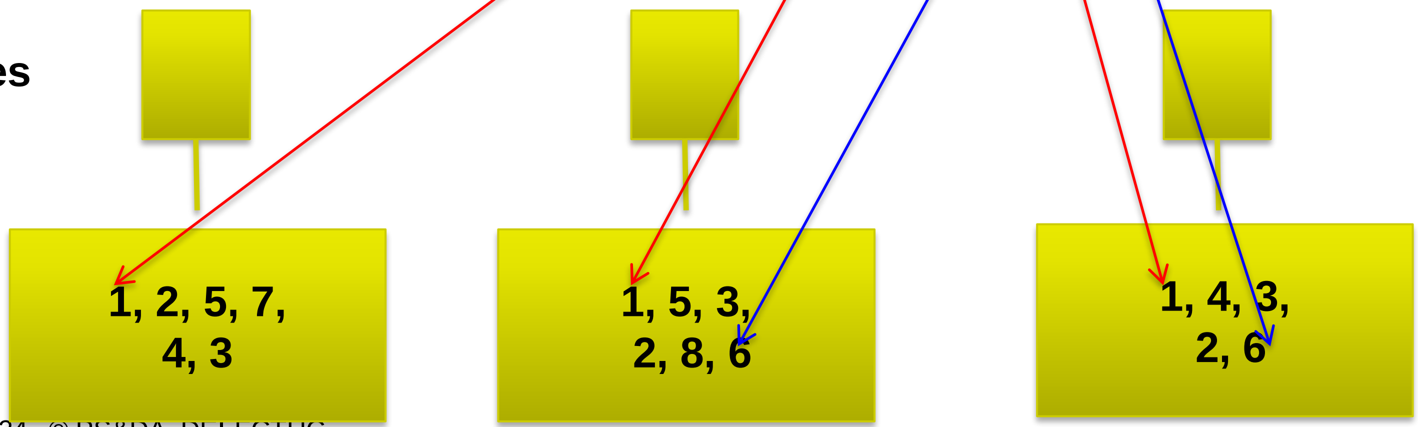
Name Node

Snapshot of FS

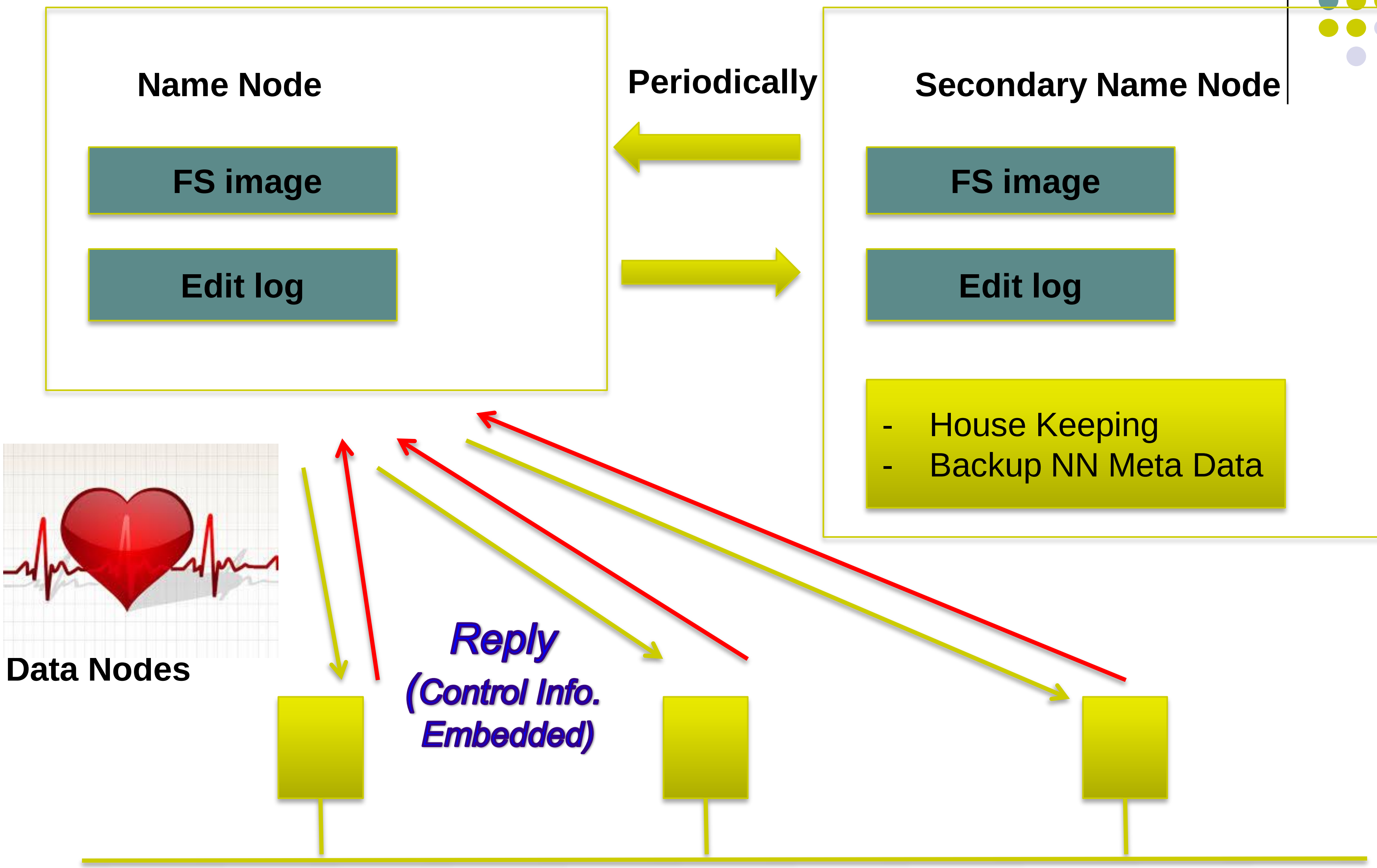
Edit log: record changes to FS

Filename	Replication factor	Block ID
File 1	3	[1, 2, 3]
File 2	2	[4, 5, 6]
File 3	1	[7,8]

Data Nodes



HDFS Inside: Name Node



DataNode

- Stores “blocks” on local disk
- Sends frequent heartbeats to NameNode
- Sends “block” reports to NameNode
- Clients connect to DataNodes for I/O

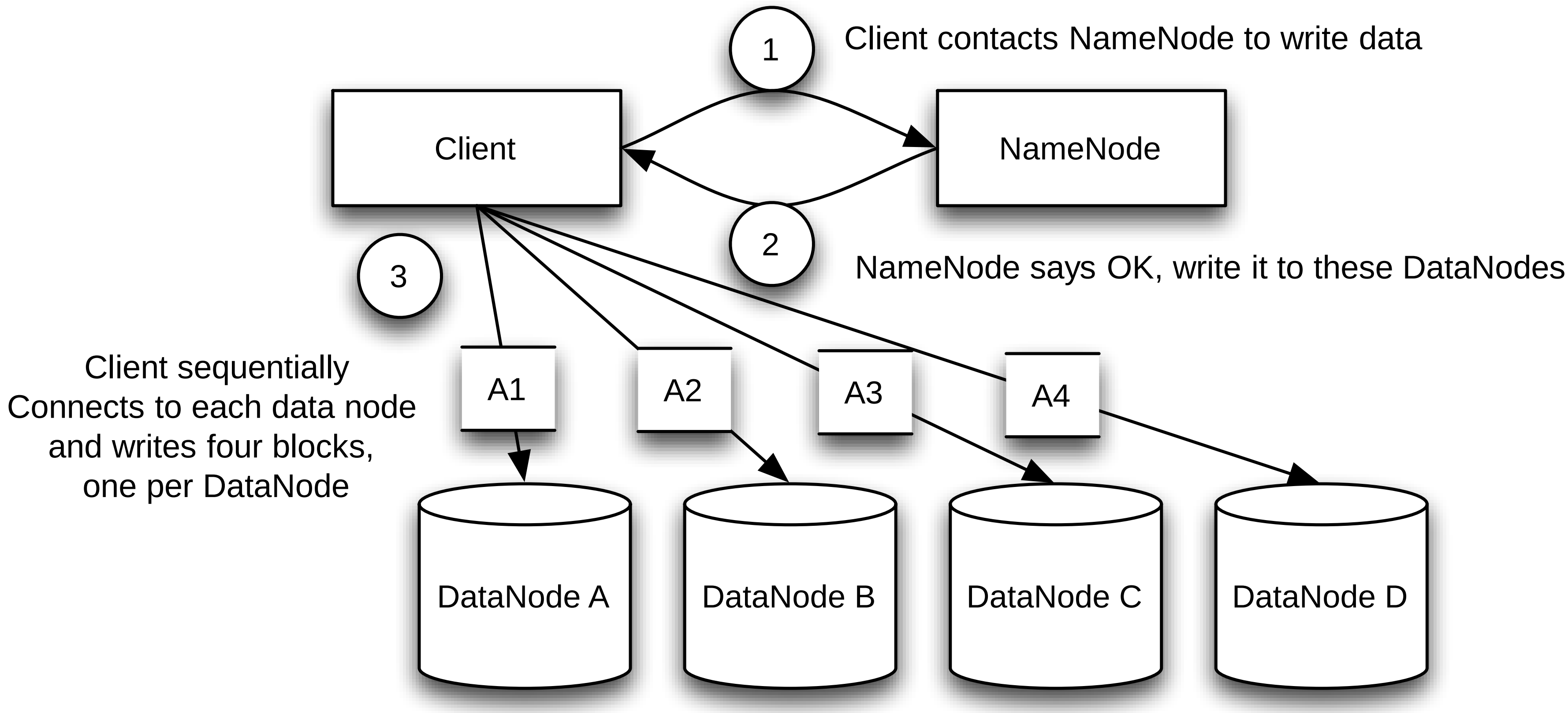
The role of “blocks” in HDFS

- Why do we need the abstraction “blocks”, in addition to “files”?
  - File can be larger than a single disk
  - Block is of fixed size, easy to manage and manipulate
  - Easier replication and fine-grained load balancing
- HDFS Block size is by default **64 MB**.
- Why is it larger than regular file system blocks?
  - To minimize overhead
    (disk seek time is almost constant)

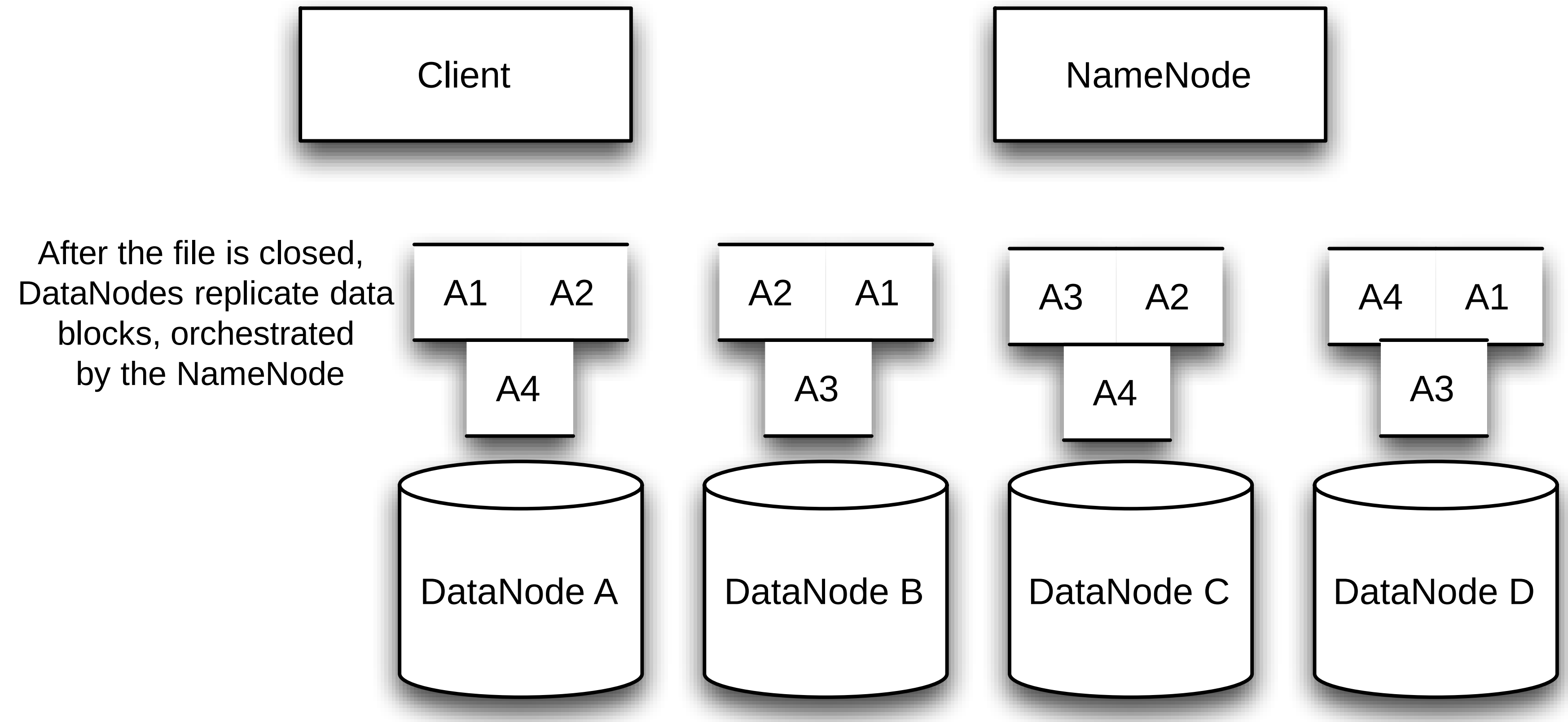
HDFS Blocks

- Default block size was 64MB, now 128 MB
  - Configurable
  - Larger sizes are common, say 256 MB or 512 MB
- Default replication is threefold
  - Also configurable
- Stored as files on the DataNode’s local fs
  - Cannot associate any block with its true file

How HDFS Works - Writes



How HDFS Works - Writes

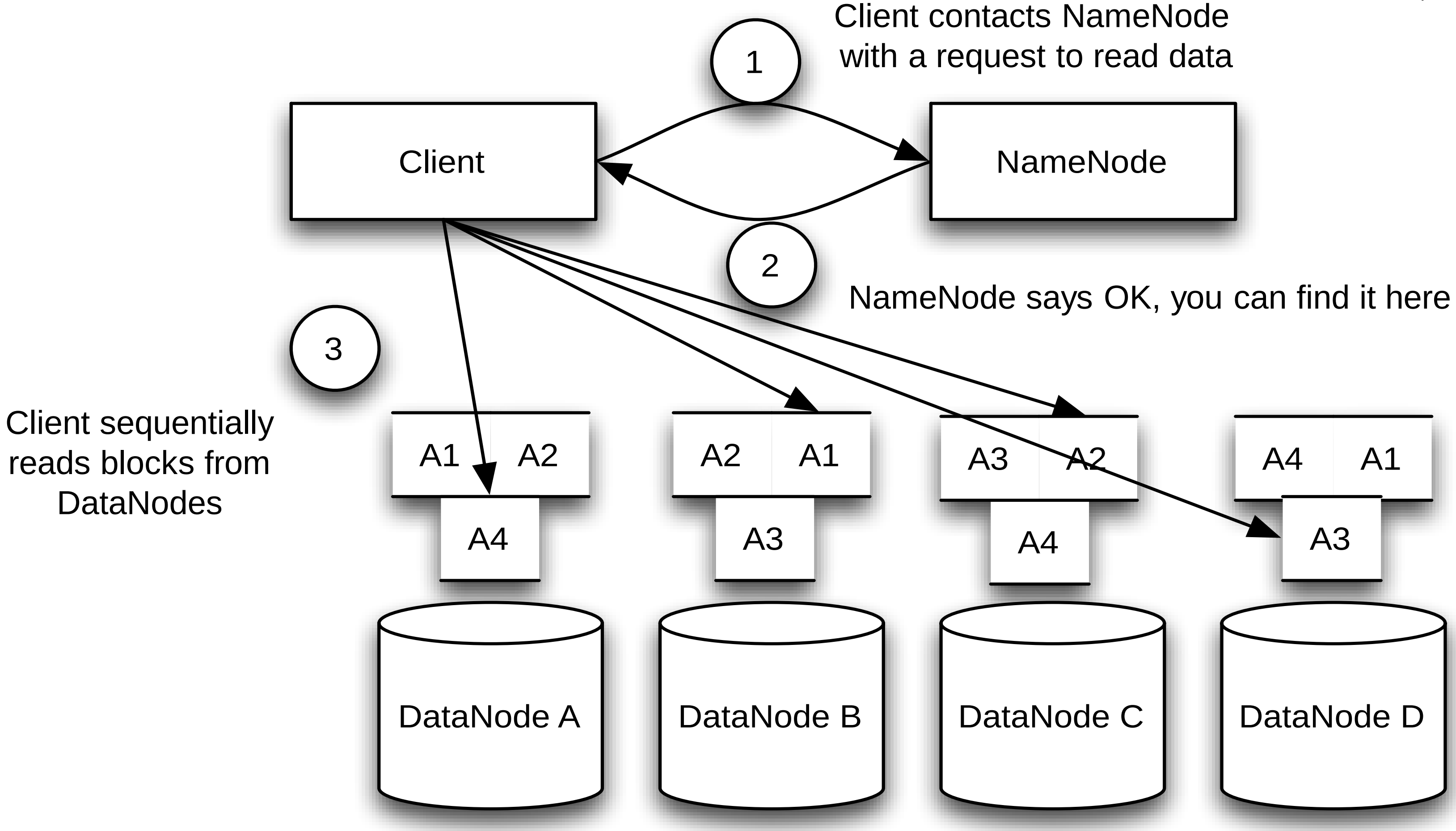


In the event of a node failure, data can be accessed on other nodes and the NameNode will move data blocks to other nodes

Replication Strategy

- 1st copy is written to the same node as the client
  - If the client is not part of the cluster, first block goes to a random node
- 2nd copy is written to a node on a different datacenter rack
- 3rd copy is written to a different node on the same rack as the second copy

How HDFS Works - Reads



How HDFS Works – Reads

- Why does HDFS choose such a design for read? *(instead of asking client to read blocks through NN)*
  - Prevent NN from being the bottleneck of the cluster
  - Allows scaling to large number of concurrent clients
  - Spreads the data traffic across the cluster
- Given multiple replicas of the same block, how does NN decide which replica the client should read?
  - Rack awareness based on network topology

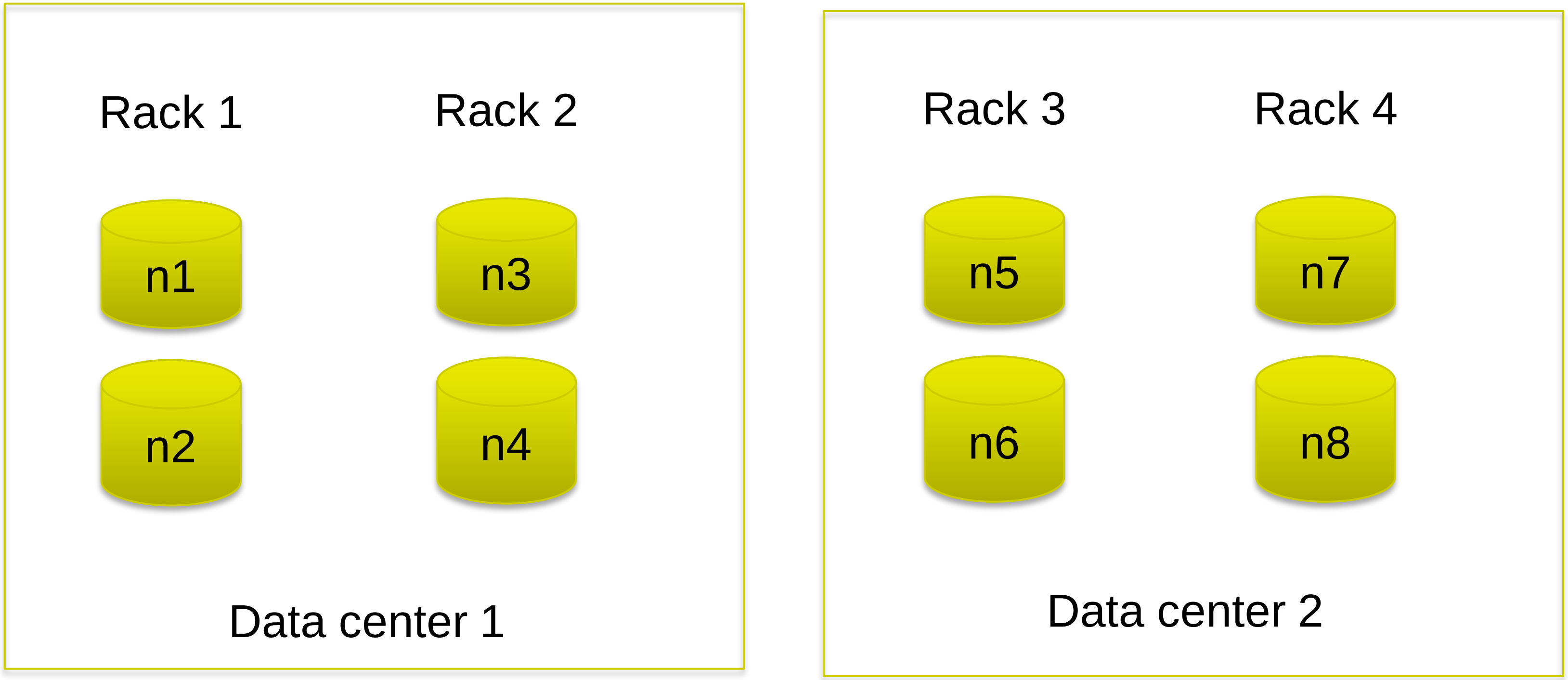
HDFS Network Topology

- The critical resource in HDFS is **bandwidth**, “distance” is defined based on that
- Measuring bandwidths between any pair of nodes is too complex and **does not scale**
- **Basic Idea:**
  - Processes on the same node
  - Different nodes on the same rack
  - Nodes on different racks in the same data center (cluster)
  - Nodes in different data centers

Bandwidth becomes smaller

HDFS Network Topology

- HDFS takes a simple approach:
  - See the network as a tree
  - **Distance between two nodes is the sum of their distances to their closest common ancestor**



HDFS Network Topology

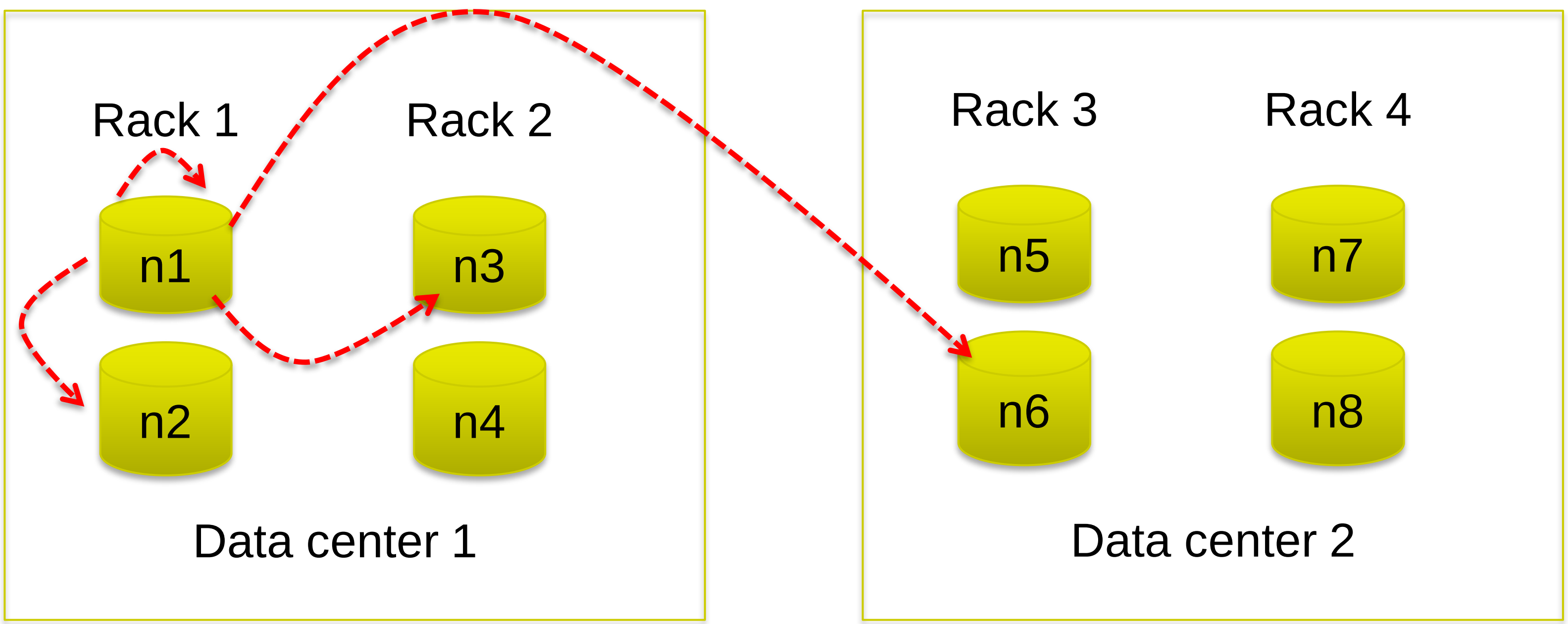
What are the distance of the following pairs:

Dist (d1/r1/n1, d1/r1/n1)= **0**

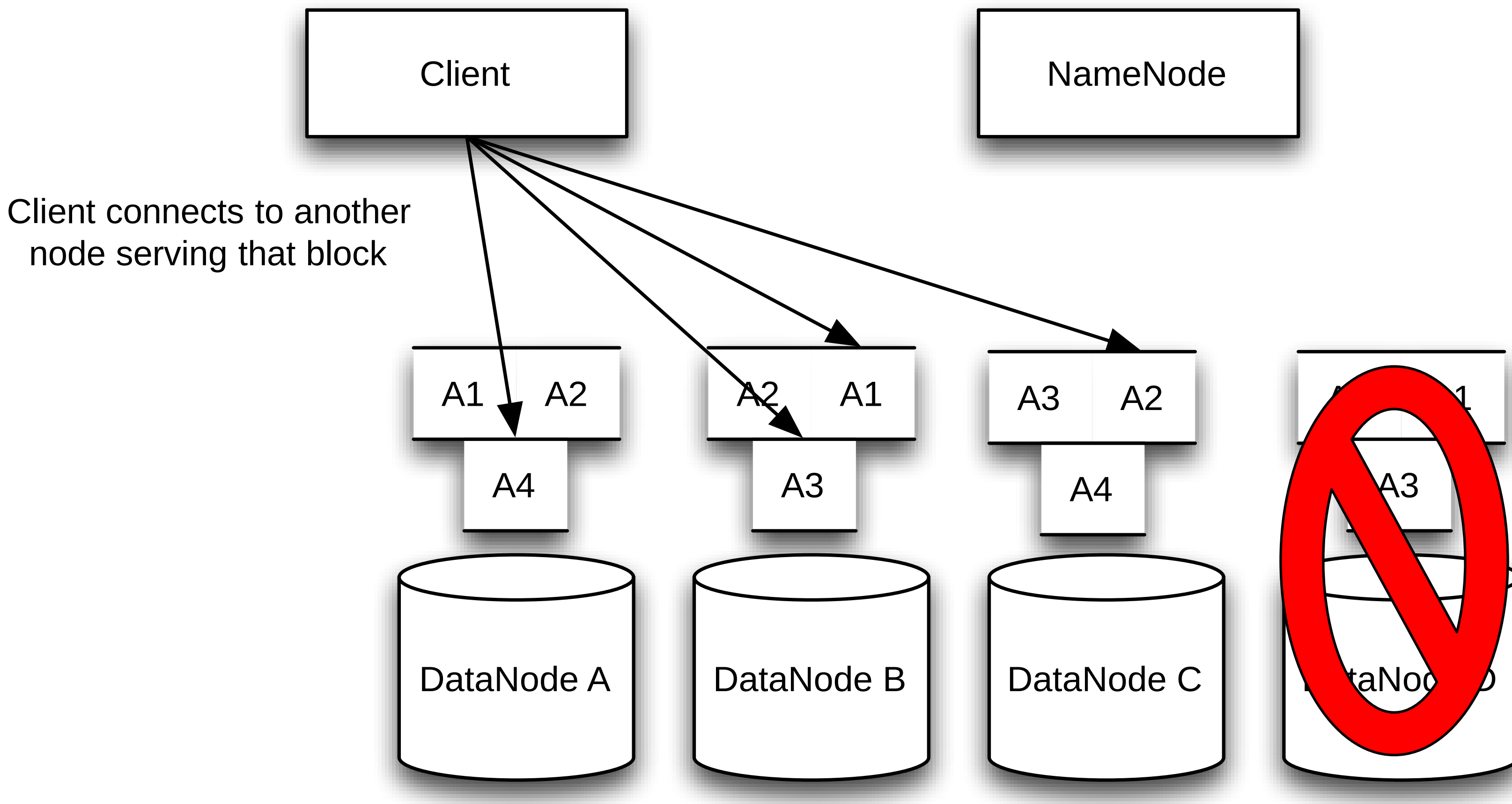
Dist(d1/r1/n1, d1/r1/n2)= **2**

Dist(d1/r1/n1, d1/r2/n3)= **4**

Dist(d1/r1/n1, d2/r3/n6)= **6**



How HDFS Works – Failure



Data Locality

- Key in achieving performance
- MapReduce tasks run as close to data as possible
- Some terms
  - Local
  - On-Rack
  - Off-Rack

Data Corruption

- Use of checksums to ensure block integrity
- Checksum is calculated on reading and compared against that when it was written
  - Fast to calculate and space-efficient
- If the checksums differ, the client reads the block from another DataNode
  - A corrupted block will be deleted and replicated by a non-corrupt block
- DataNode periodically runs a background thread to do this checksum process
  - This is important for files that are not read very often, otherwise data corruption might be discovered too late

Fault Tolerance

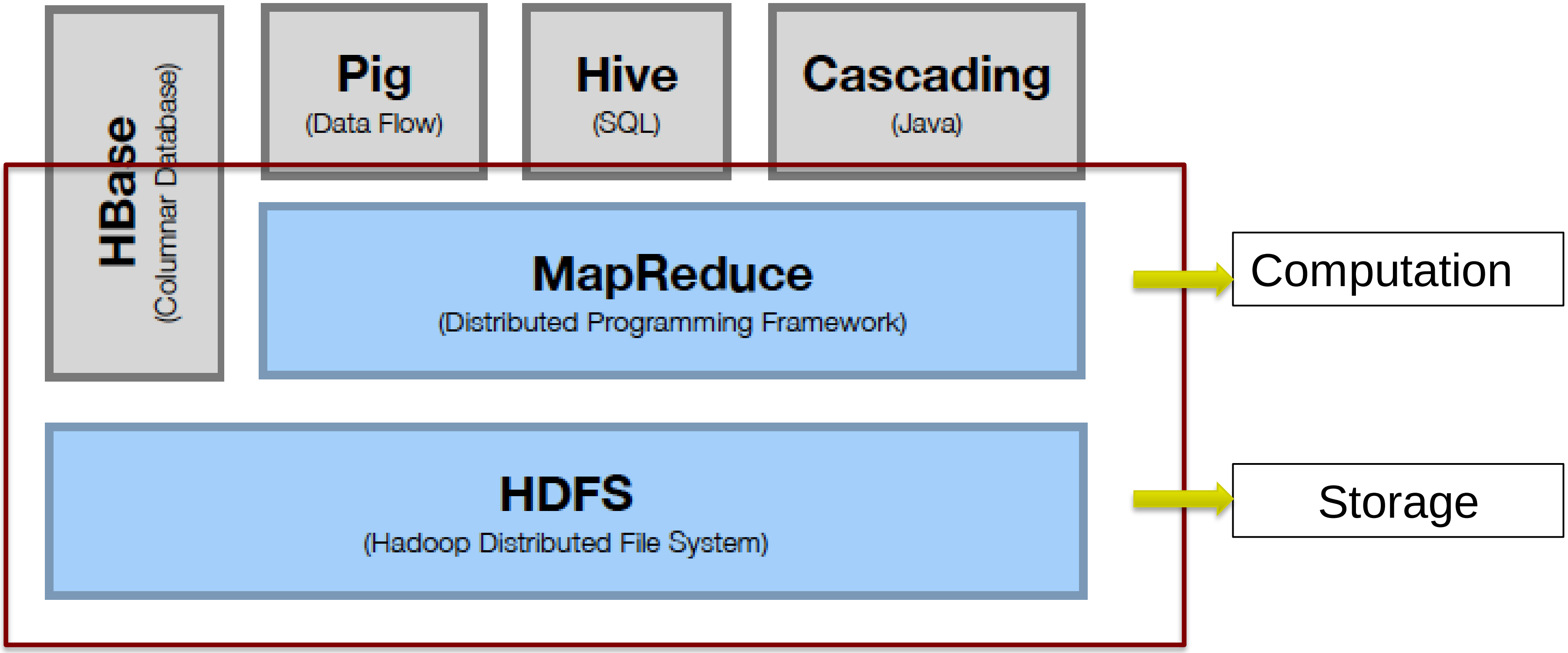
- If no heartbeat is received from a DataNode within a (configurable) time window, it is considered lost
  - Default time window is 10 minutes
- The NameNode will:
  - Determine which blocks were on the lost node
  - Locate other DataNodes with valid copies
  - Instruct DataNodes with copies to replicate blocks to other DataNodes

Interacting with HDFS

Several alternatives:

- Primary interfaces are CLI and Java API
- Web UI for read-only access
- WebHDFS provides RESTful read/write
- HttpFS also exists
- snakebite is a Python library from Spotify
- The FUSE tool allows HDFS to be mounted on standard file systems, so legacy apps can use HDFS data

Hadoop Stack (revisited)



Hadoop MapReduce (MR)

- A programming model for processing data
- Contains two phases:
  - Map (perform a *map* function on key/value pairs)
  - Reduce (perform a *reduce* function on key/value groups)

- Groups are created by sorting map output

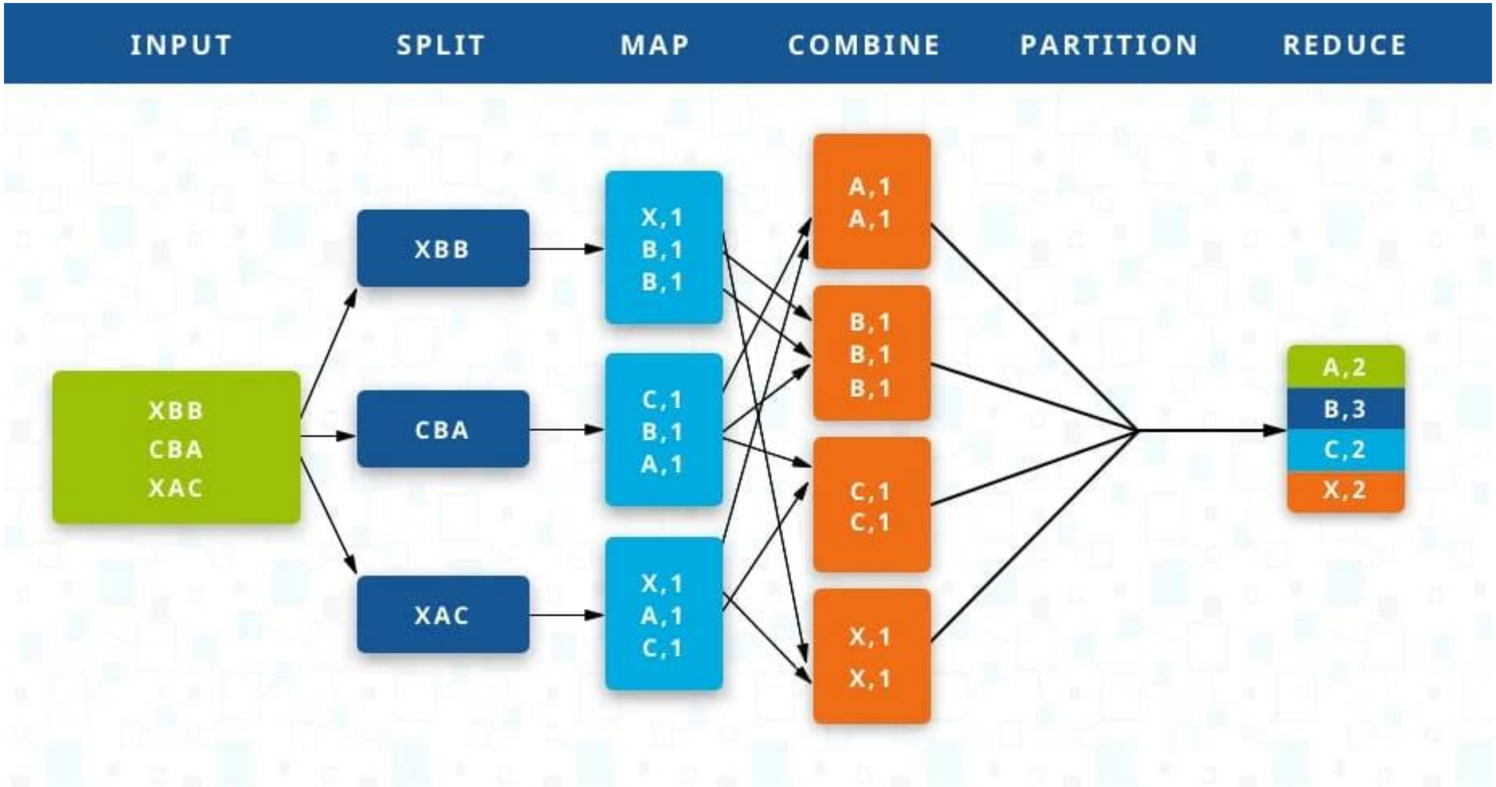
- Operations on key/value pairs open the door for highly parallel algorithms

Hadoop MapReduce (cont’d)

- Automatic parallelization and distribution of tasks
- Framework handles scheduling tasks and repeating failed tasks
- Developer can code many pieces of the puzzle
- The framework handles a “Shuffle and Sort” phase between map tasks and reduce tasks

- Developers need to focus only on the task at hand, rather than how to manage where data comes from and where it goes to

Hadoop MapReduce (cont'd)



https://www.talend.com/resources/what-is-mapreduce/

MRv1 and MRv2

- Both manage compute resources, jobs, and tasks
- Job API is the same

- MRv1 is proven in production
  - JobTracker / TaskTrackers

- MRv2 is a more recent application on YARN
  - YARN is a generic platform for developing distributed applications

MRv1

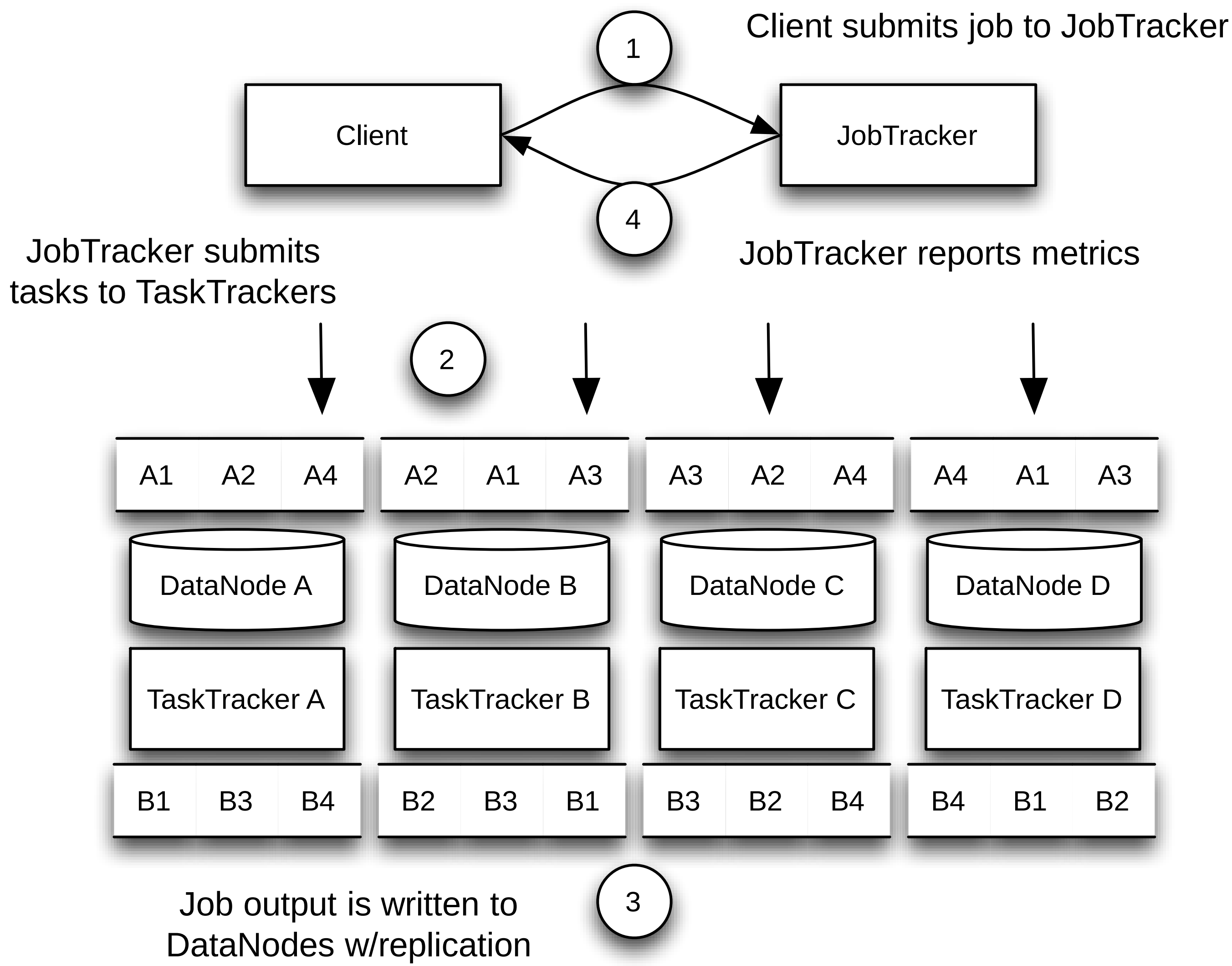
JobTrackers:

- Monitor *job* and *task* progress
- Issue *task attempts* to TaskTrackers
- Re-try failed task attempts
- Four failed attempts of same task = one failed job

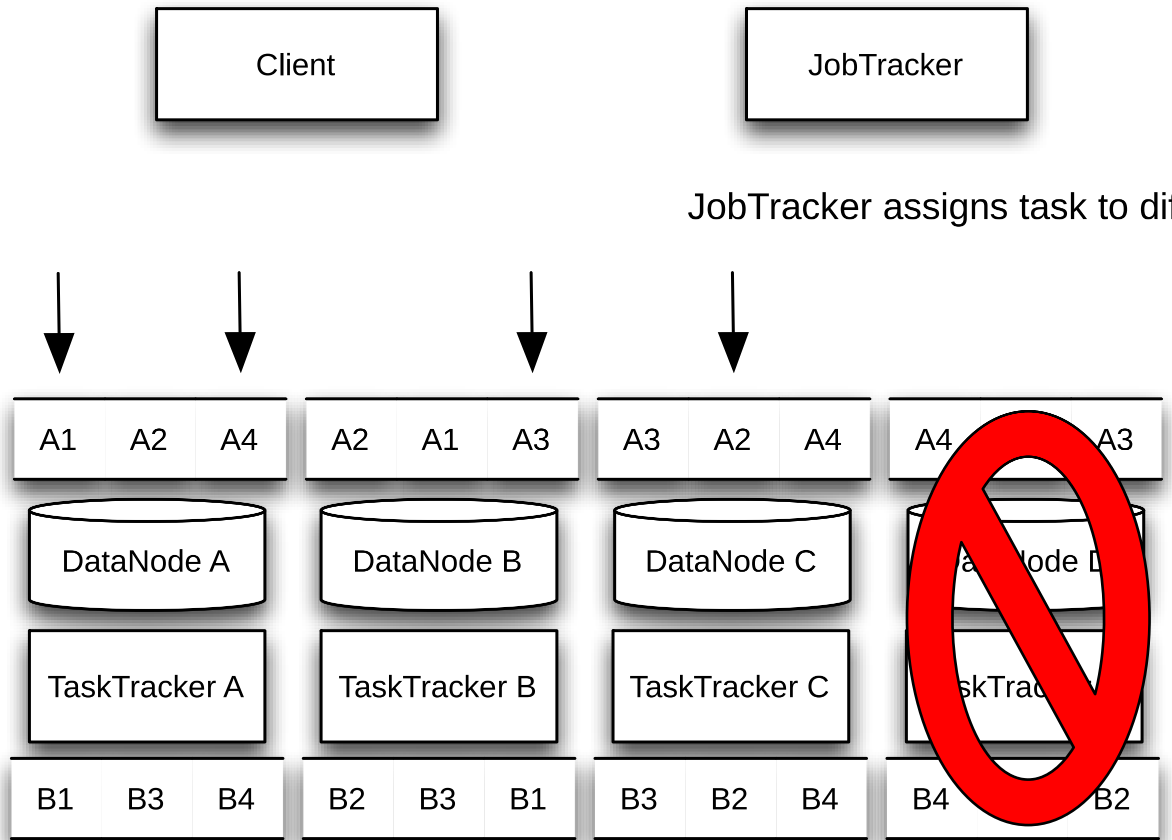
Tasktrackers

- Run on the same node as DataNodes
- Send heartbeats and task reports to JobTrackers
- Configurable number of map and reduce slots
- Run map and reduce *task attempts* in a separate JVM

How MapReduce Works



How MapReduce Works – Failure

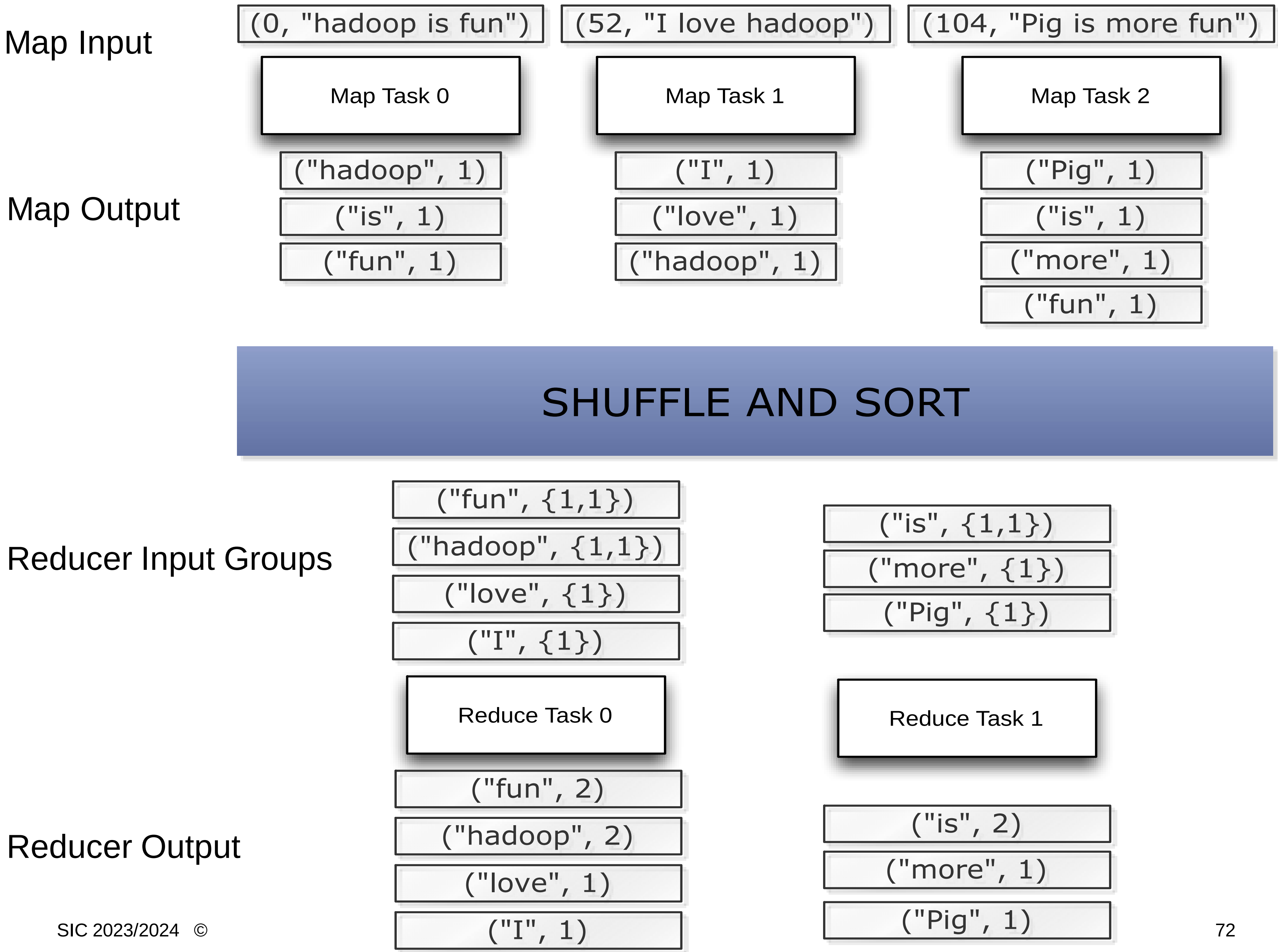


Example – Word Count

- Count the number of times each word is used in a body of text
- Uses TextInputFormat and TextOutputFormat

```
map(byte_offset, line)
    foreach word in line
        emit(word, 1)

reduce(word, counts)
    sum = 0
    foreach count in counts
        sum += count
    emit(word, sum)
```



Map Input
(0, "hadoop is fun")
(52, "I love hadoop")
(104, "Pig is more fun")
Map Task 0
Map Task 1
Map Task 2
Map Output
("hadoop", 1)
("is", 1)
("fun", 1)
("I", 1)
("love", 1)
("hadoop", 1)
("Pig", 1)
("is", 1)
("more", 1)
("fun", 1)
SHUFFLE AND SORT
Reducer Input Groups
("fun", {1,1})
("hadoop", {1,1})
("love", {1})
("I", {1})
("is", {1,1})
("more", {1})
("Pig", {1})
Reduce Task 0
Reduce Task 1
Reducer Output
("fun", 2)
("hadoop", 2)
("love", 1)
("I", 1)
("is", 2)
("more", 1)
("Pig", 1)

Mapper Code

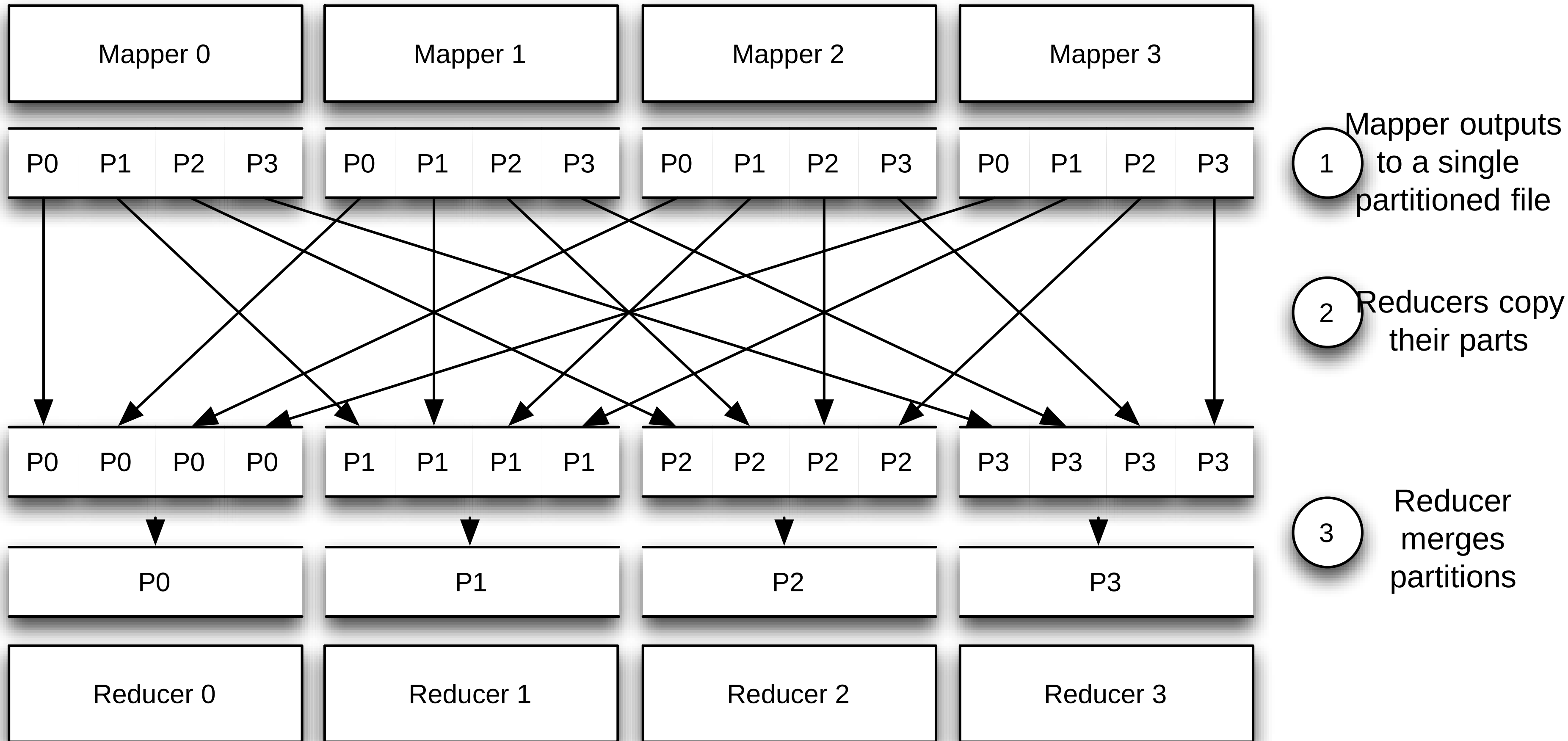
```
public class WordMapper
              extends Mapper<LongWritable, Text, Text, IntWritable>
{

    private final static IntWritable ONE = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, ONE);
        }
    }
}
```

Shuffle and Sort



Reducer Code

```
public class IntSumReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
                                                Context context) {
        int sum = 0;

        for (IntWritable val : values) {
            sum += val.get();
        }

        context.write(key, new IntWritable(sum));
    }
}
```

Hadoop ecosystem

Resources and Wrap-up

- Take a look at these videos:
  - Hadoop explained
    - https://www.youtube.com/watch?v=hLnB0uzGvDI
  - Hadoop ecosystem
    - https://www.youtube.com/watch?v=p0TdBqIt3fg
  - Hadoop vs. Spark
    - https://www.youtube.com/watch?v=2PVzOHA3ktE

Resources and Wrap-up

- http://hadoop.apache.org
- Supportive community
  - Hadoop-DC
  - Data Science MD
  - Baltimore Hadoop Users Group
- Plenty of resources available to learn more
  - Books
  - Email lists
  - Blogs