KOÇ UNIVERSITY

Distributed File System(DFS)

Google and Hadoop File Systems

Didem Unat

COMP304 - Operating Systems (OS)

Basic Features: Distributed File Systems (DFS)

- 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

Data Characteristics

- Applications need streaming access to data
- Batch processing rather than interactive user access.
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
  - Tens of millions of files in a single instance
  - Large data sets and files: gigabytes to terabytes size

- Write-once-read-many: a file once created, written and closed need not be changed – this assumption simplifies coherency
- A map-reduce application or web-crawler application fits perfectly to this model.

Example: Distributed File Systems

- Hadoop is an open-source implementation based on **Google File System** (GFS) and **MapReduce** from Google
- Developed at Yahoo
- Hadoop was donated to **Apache** in 2006

Motivation Questions

- **Problem 1:** Data is too big to store on one machine.

- **HDFS:** Store the data on multiple machines!

Motivation Questions

- **Problem 2:** Very high-end machines are too expensive

- **HDFS:** Run on commodity hardware!

Motivation Questions

- **Problem 3:** Commodity hardware will fail!

- **HDFS:** Software is intelligent enough to handle hardware failure!

Fault tolerance

- Failure is the norm rather than exception
- An DFS 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 DFS.

Motivation Questions

- **Problem 4:** What happens to the data if the machine stores the data fails?

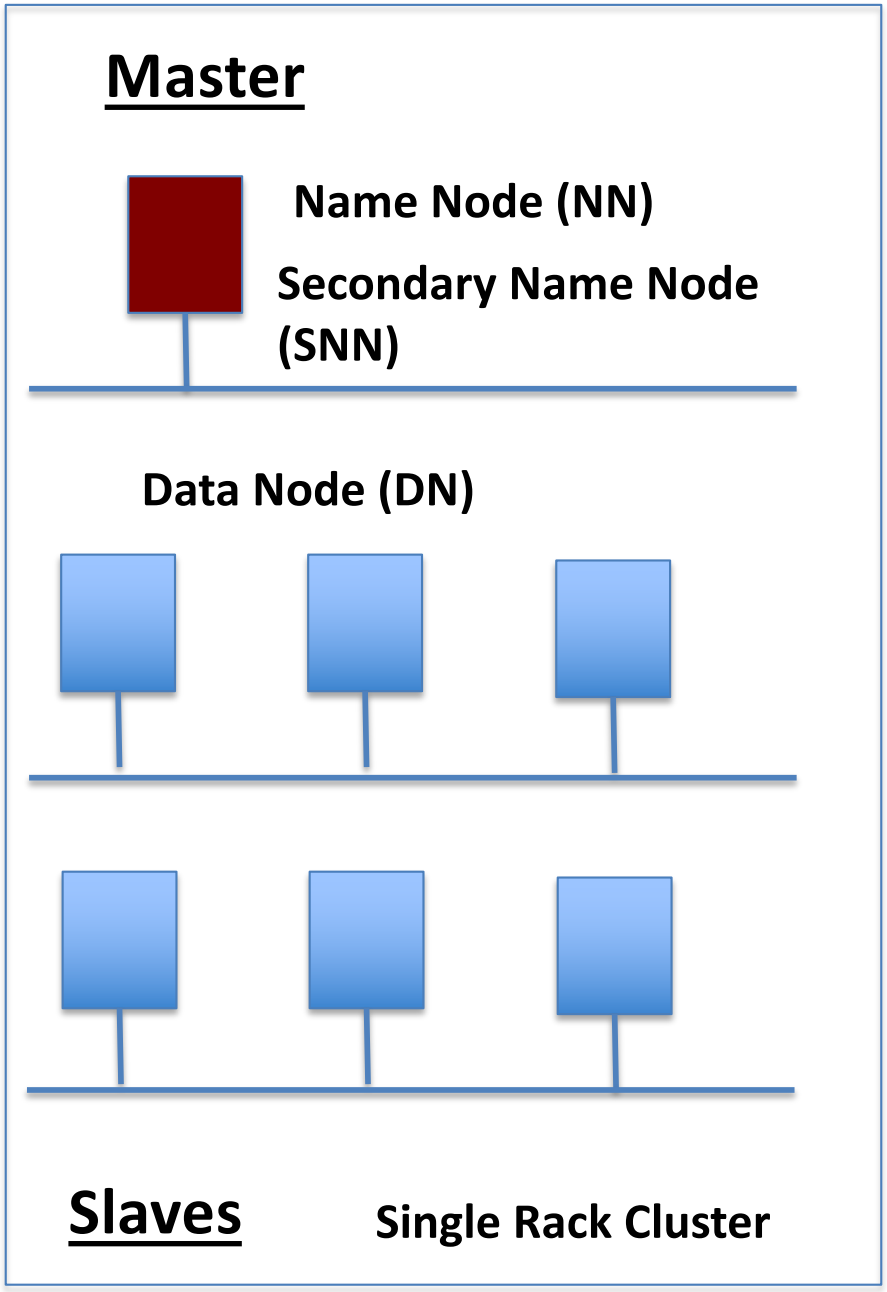
- **HDFS:** Replicate the data!

Motivation Questions

- **Problem 5:** How can distributed machines organize the data in a coordinated way?

- **HDFS:** Master-Slave Architecture!

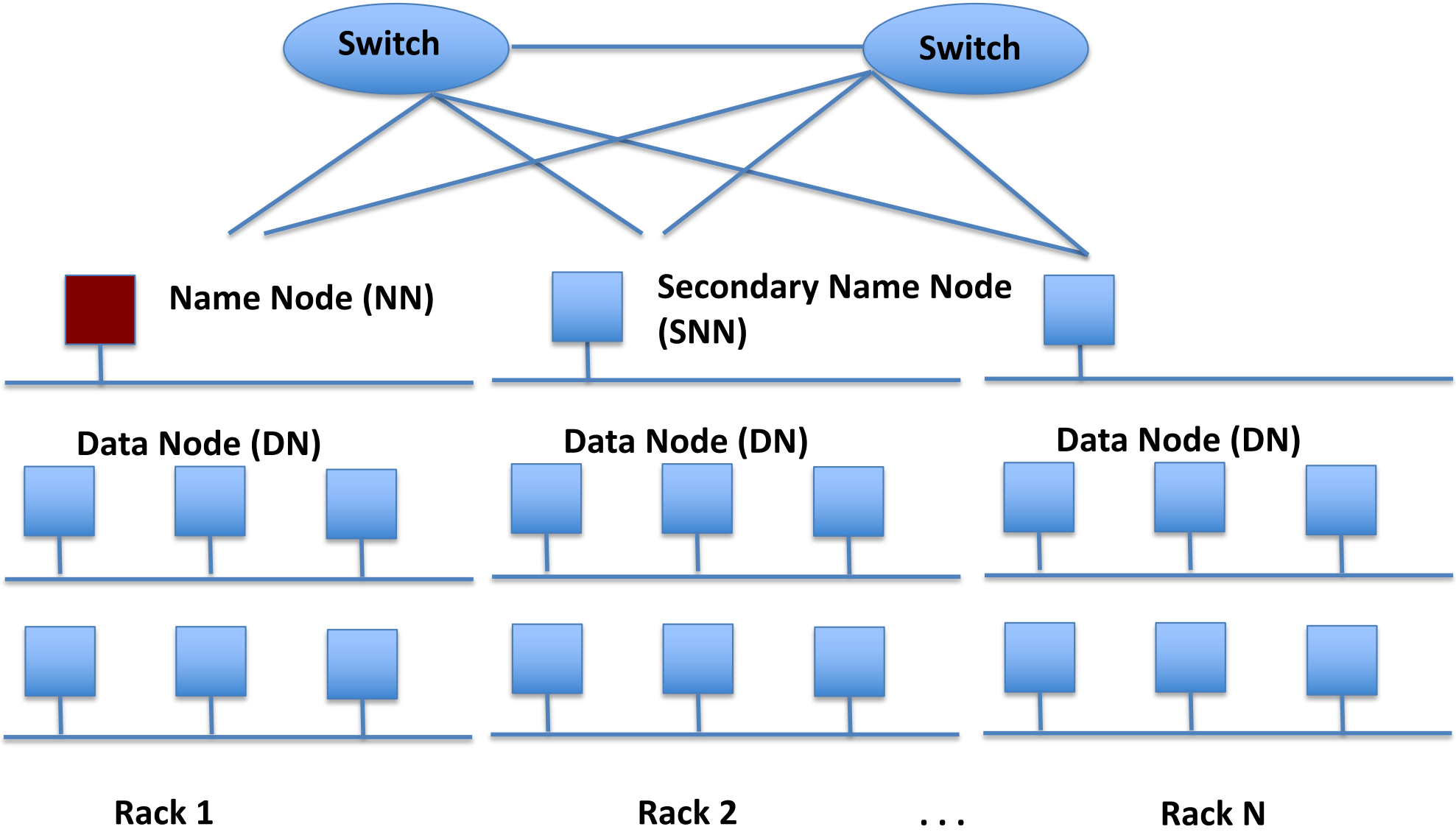
HDFS Architecture: Master-Slave



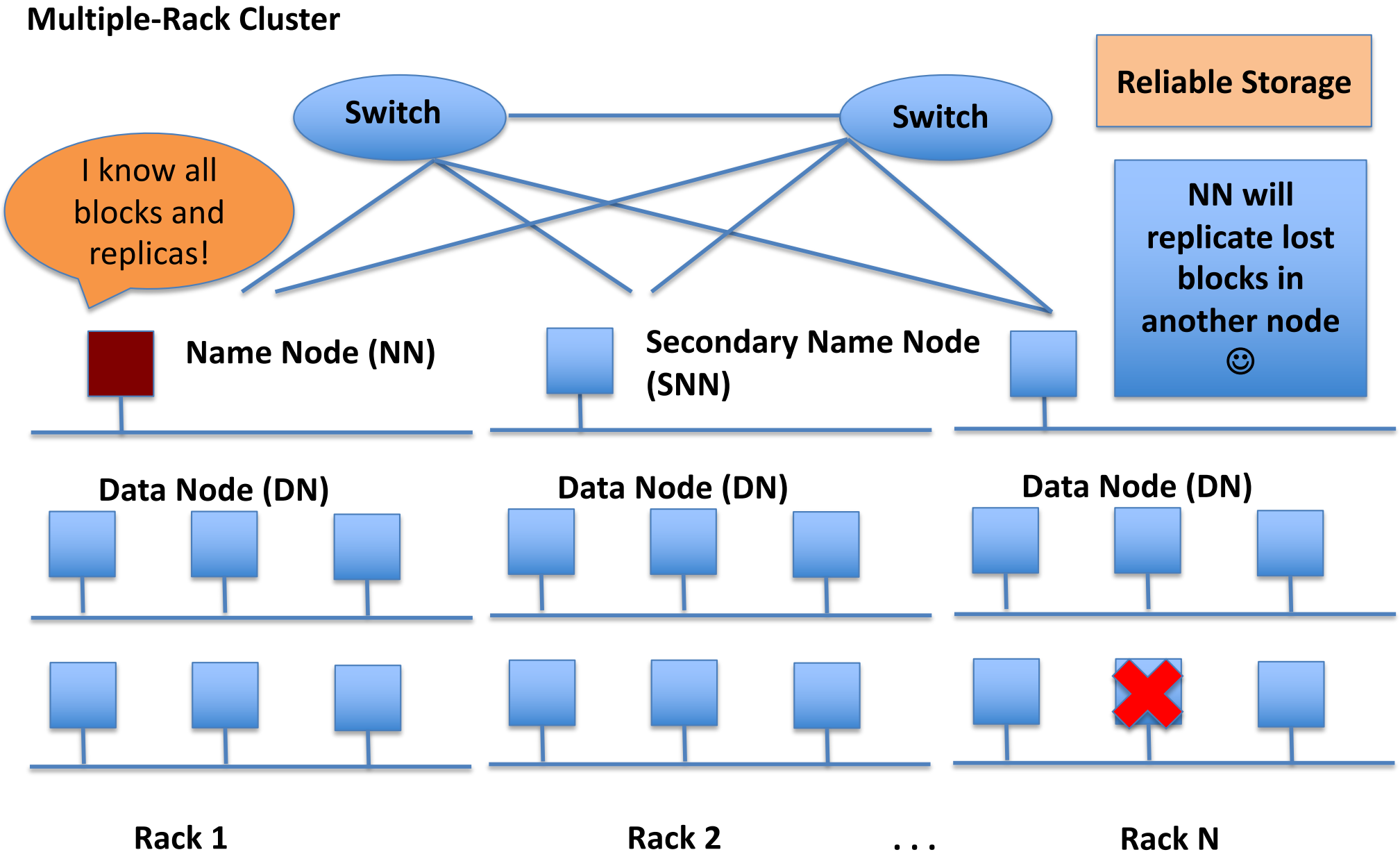
- Name Node: Controller
  - File System Name Space Management
  - Block Mappings
- Data Node: Work Horses
  - Block Operations
  - Replication
- Secondary Name Node:
  - Checkpoint node

HDFS Architecture: Master-Slave

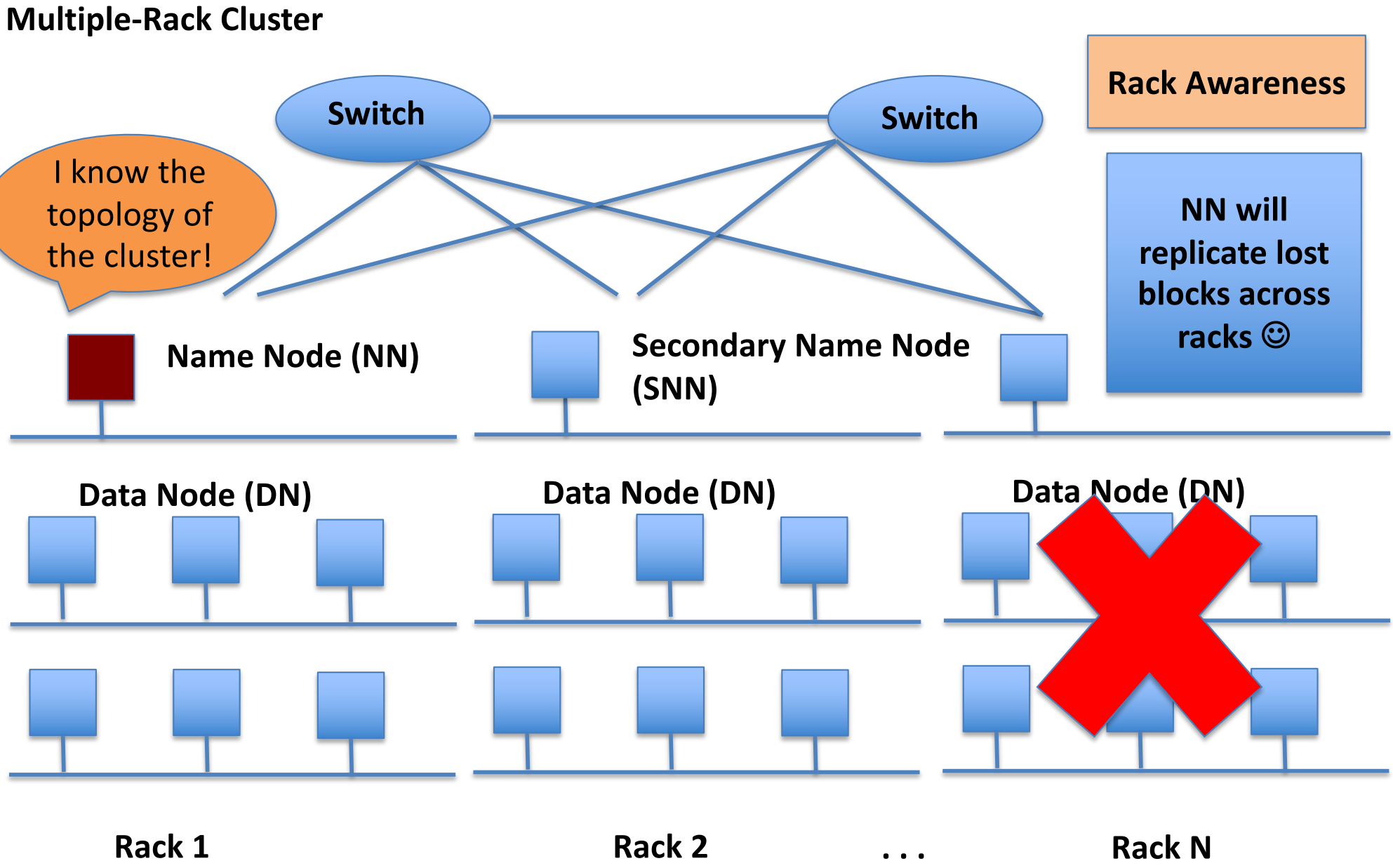
Multiple-Rack Cluster



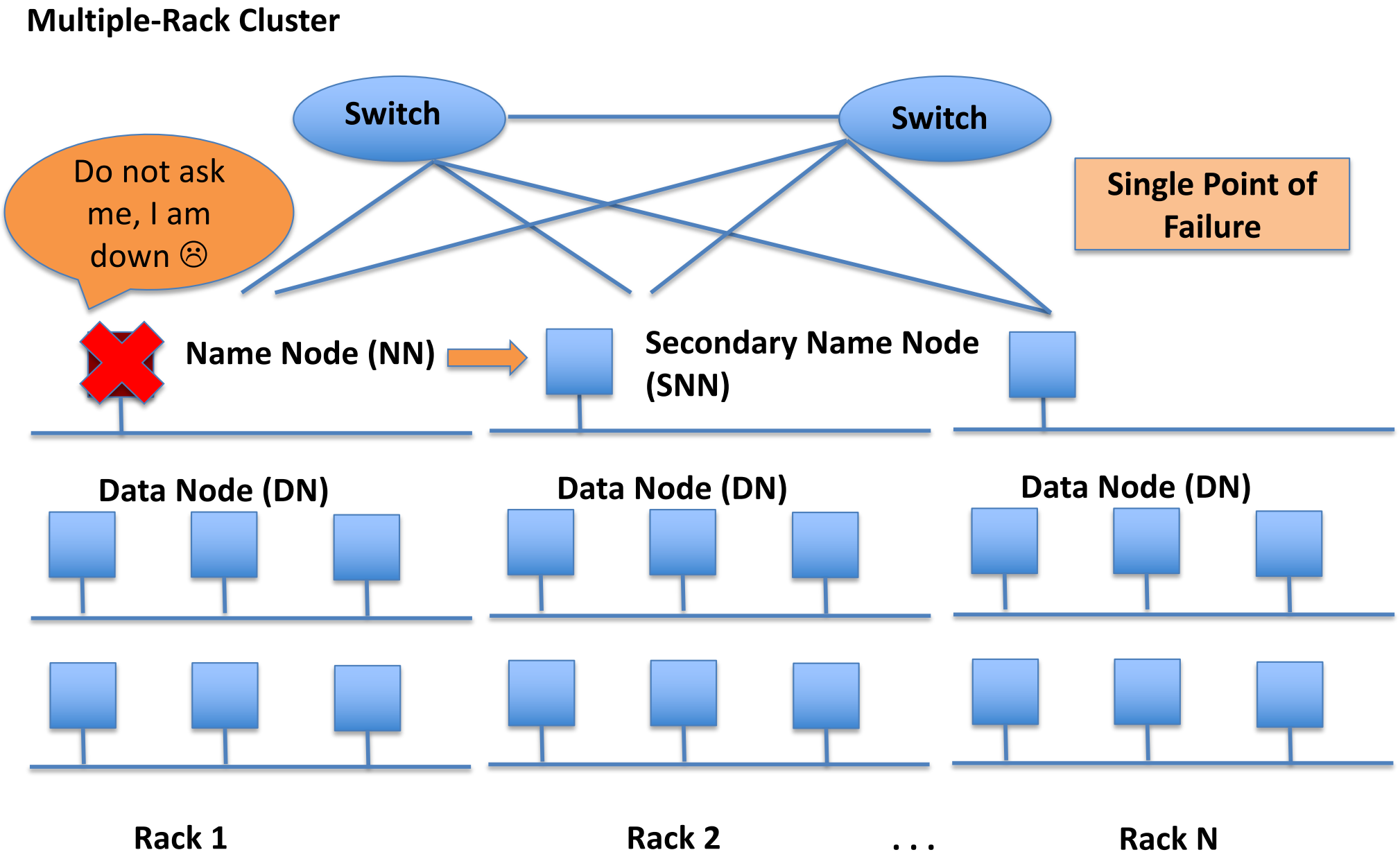
HDFS Architecture: Master-Slave



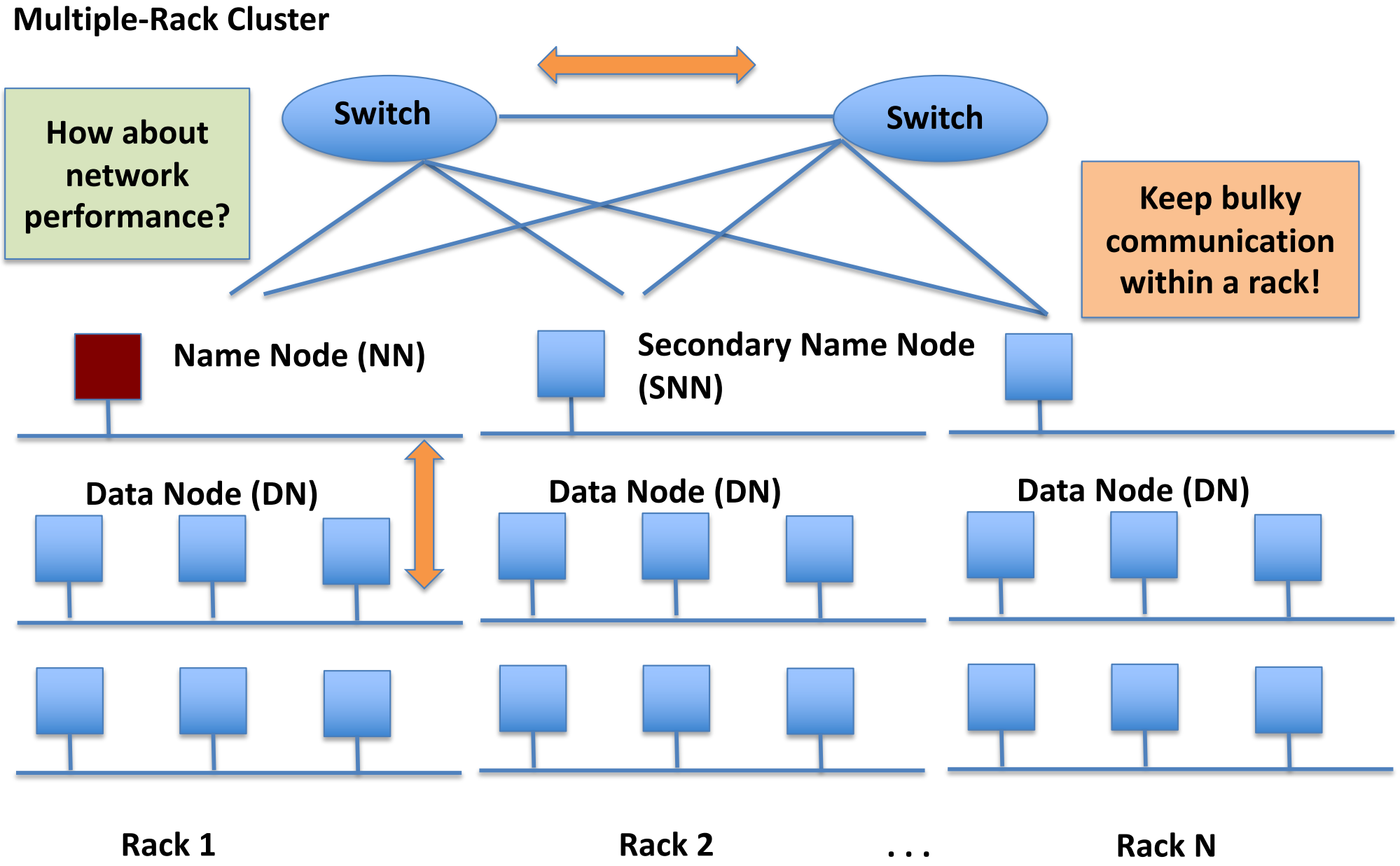
HDFS Architecture: Master-Slave



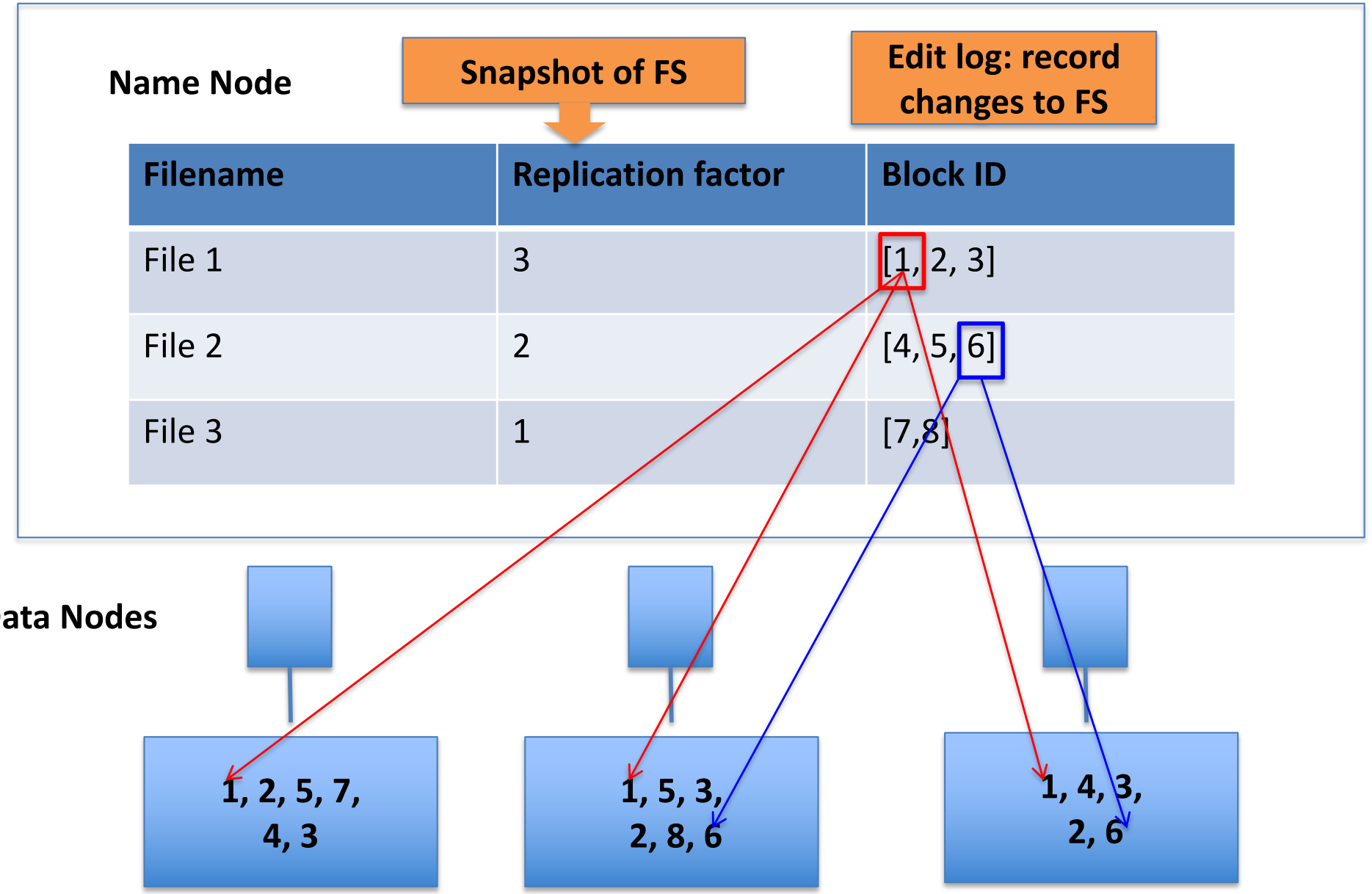
HDFS Architecture: Master-Slave



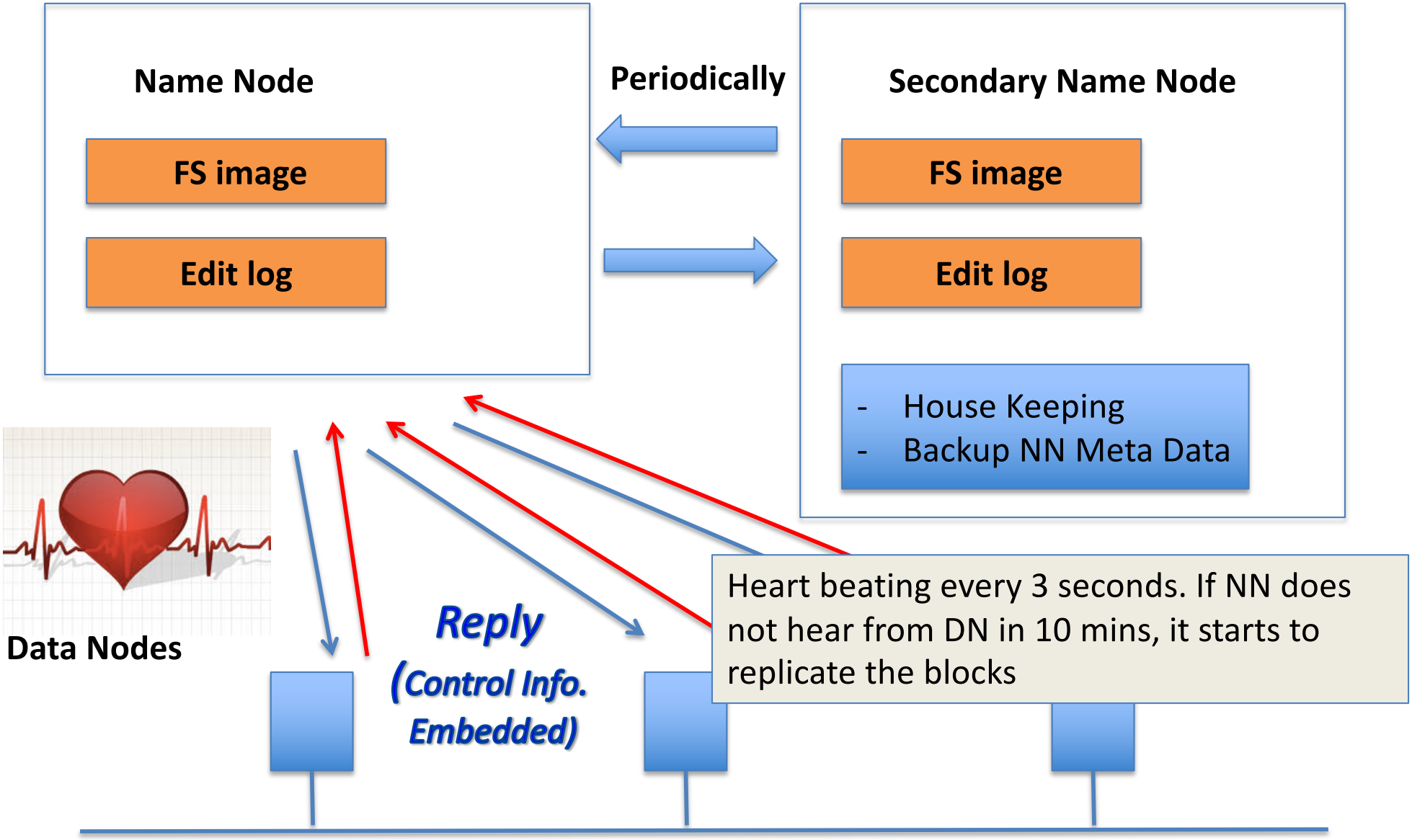
HDFS Architecture: Master-Slave



HDFS Architecture: Master-Slave



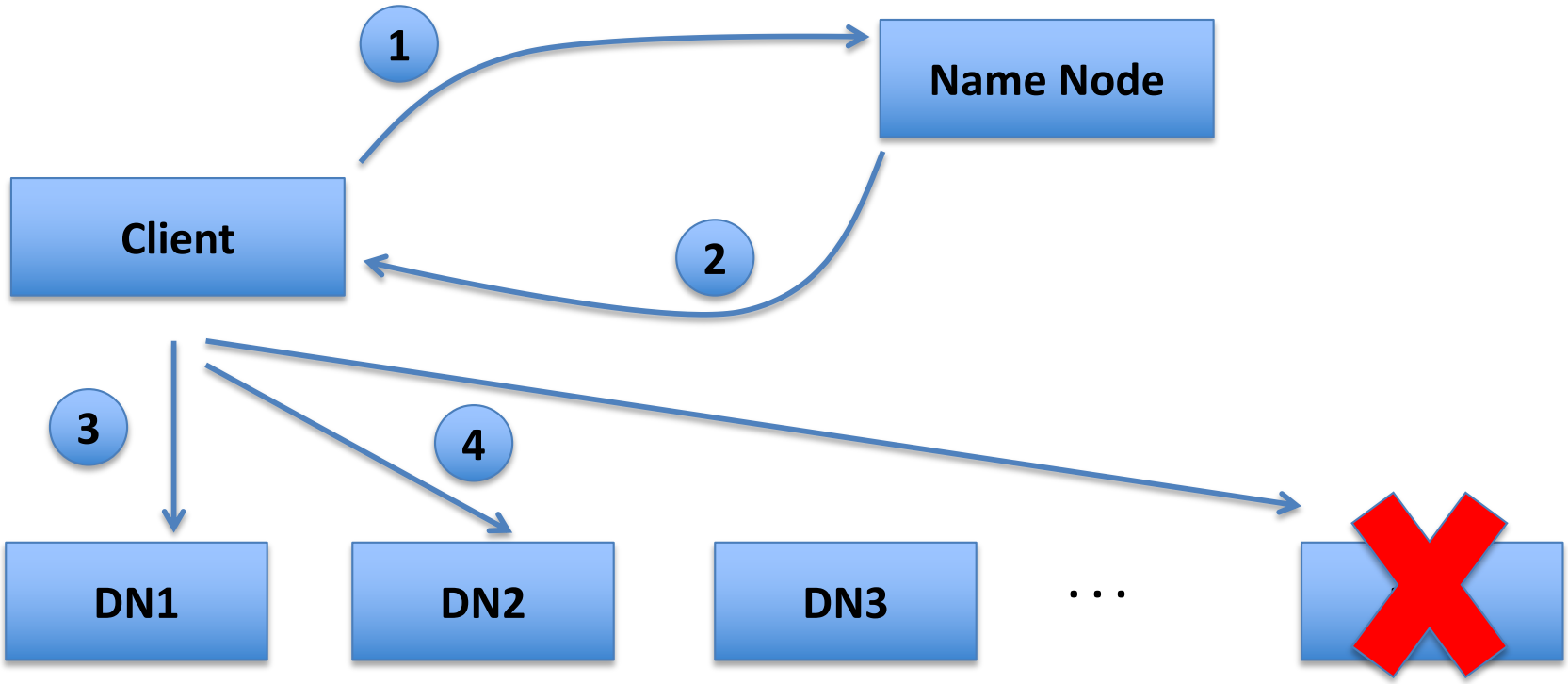
HDFS Architecture: Master-Slave



HDFS Inside: Blocks

- Q: Why do we need the abstraction “Blocks” in addition to “Files”?
- Reasons:
  - Files can be larger than a single disk
  - Block is of fixed size, easy to manage and manipulate
  - Easy to replicate and do more fine-grained load balancing

HDFS Inside: Read



1. Client connects to NN to read data
2. NN tells client where to find the data blocks
3. Client reads blocks directly from data nodes (without going through NN)
4. In case of node failures, client connects to another node that serves the missing block

HDFS Inside: Read

- Q: Why does HDFS choose such a design for read? Why not ask client to read blocks through NN?
- Reasons:
  - Prevent NN from being the bottleneck of the cluster
  - Allow HDFS to scale to large number of concurrent clients
  - Spread the data traffic across the cluster

HDFS Inside: Read

- Q: Given multiple replicas of the same block, how does NN decide which replica the client should read?
- HDFS Solution:
  - Rack awareness based on network topology

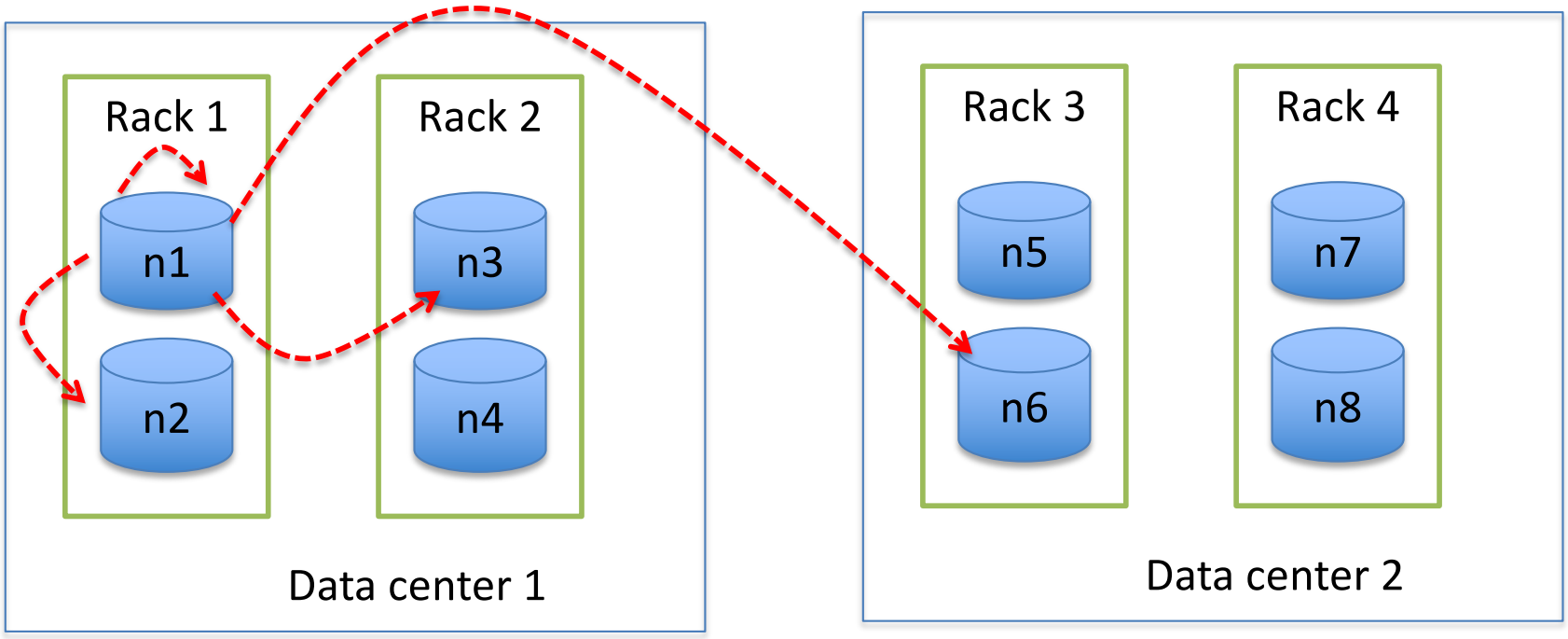
HDFS Inside: Network Topology

- 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

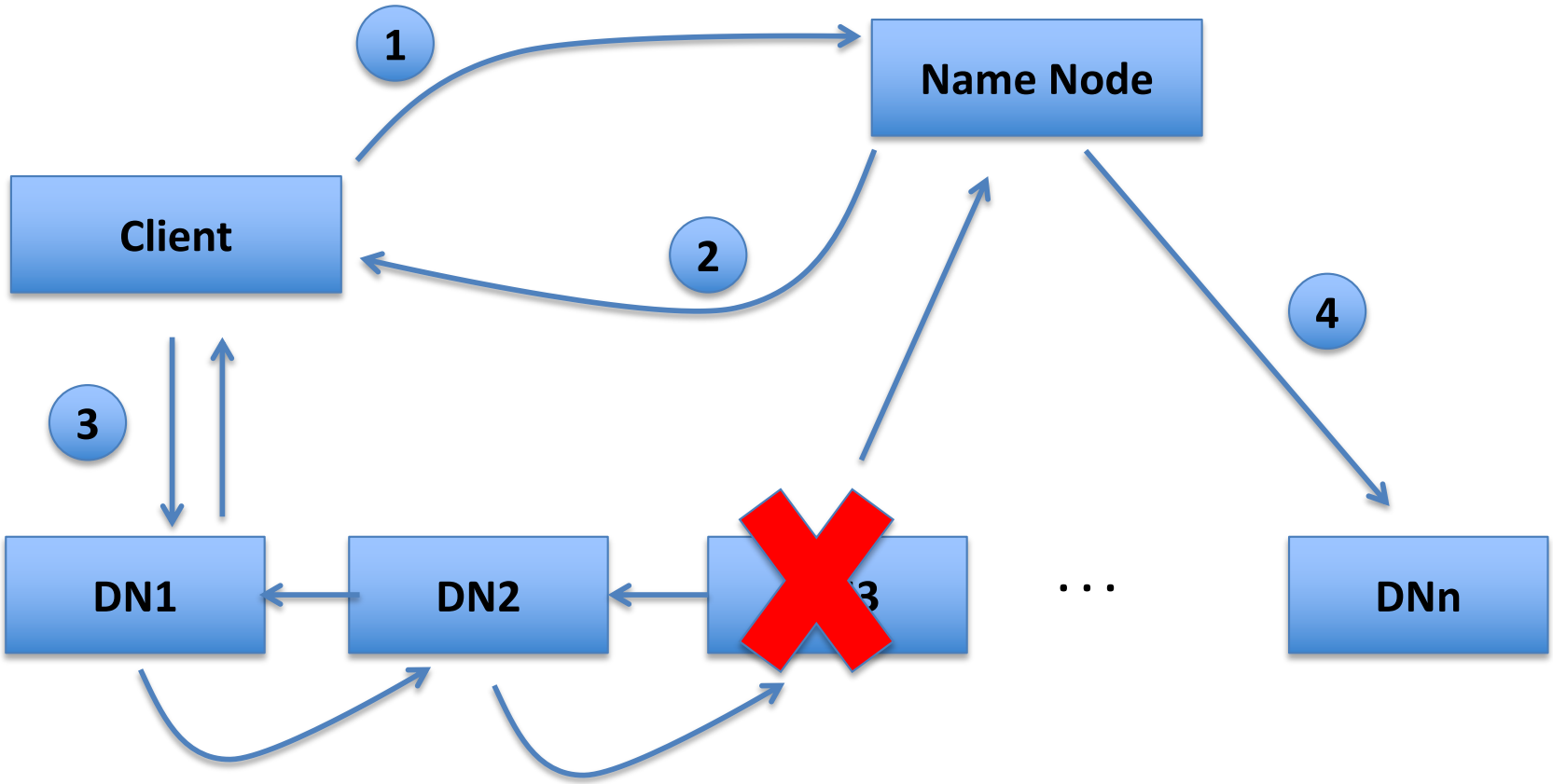
Bandwidth becomes less

HDFS Inside: 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 Inside: Write



1. Client connects to NN to write data
2. NN tells client write these data nodes
3. Client writes blocks directly to data nodes with desired replication factor
4. In case of node failures, NN will figure it out and replicate the missing blocks

HDFS Inside: Write

- Q: Where should HDFS put the three replicas of a block? What tradeoffs we need to consider?
- Tradeoffs:
  - Reliability
  - Write Bandwidth
  - Read Bandwidth

Q: What are some possible strategies?

HDFS Inside: Write

- Replication Strategy vs Tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			

HDFS Inside: Write

- Replication Strategy vs Tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			
HDFS: 1-> same node as client 2-> a node on different rack 3-> a different node on the same rack as 2			

HDFS Inside: Blocks

- HDFS Block size is by default **64 MB**, why it is much larger than regular file system block?
  - HDFS block is 32K times larger than regular file system
- Reasons:
  - Minimize overhead: disk seek time is almost constant

HDFS Interface

- Web Based Interface
  - http://ccl.cse.nd.edu/operations/hadoop/
- Command Line: Hadoop FS Shell
  - https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html
- These slides are adapted from
  - Prof. Dong Wang

I am recruiting

- Multiple graduate student positions are available
- Visit parcorelab.com/jobs