Distributed Storage

EE379K - Architectures for Big Data Sciences
Dr. Sriram Vishwanath
Department of Electrical and Computer Engineering
The University of Texas at Austin
Spring 2016

Why Distributed Storage?

- Hard to store petabytes on a single machine
- Need to distribute data over many different machines
- This introduces some challenges

Real systems that use distributed storage codes

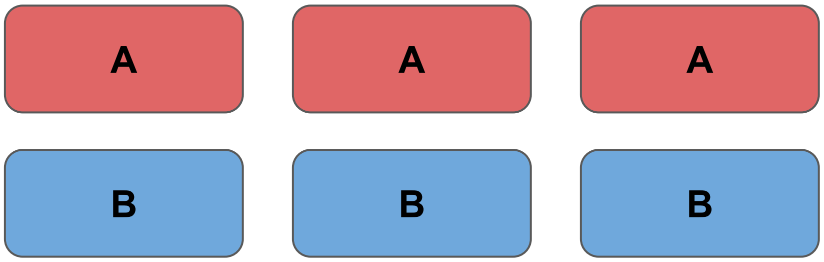
- Windows Azure, (Cheng et al. USENIX 2012) (LRC Code)
- Used in production in Microsoft
- CORE (Li et al. MSST 2013) (Regenerating EMSR Code)
- NCCloud (Hu et al. USENIX FAST 2012) (Regenerating Functional MSR)
- ClusterDFS (Pamies Juarez et al.) (SelfRepairing Codes)
- StorageCore (Esmaili et al.) (over Hadoop HDFS)
- HDFS Xorbas (Sathiamoorthy et al. VLDB 2013) (over Hadoop HDFS) (LRC code)
- Testing in Facebook clusters

Fault Tolerance

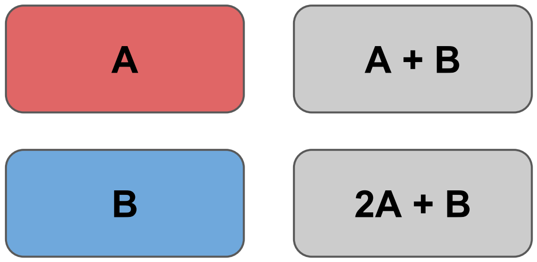
- Drives are not immune to failure
- High MTBF counteracted by large number of drives
    - Google datacenters house more than 1 million servers
    - Drive failures occur on the order of **minutes**
- How to prevent data loss when a drive fails?

Fault Tolerance: Solution

- Add redundancy to the system



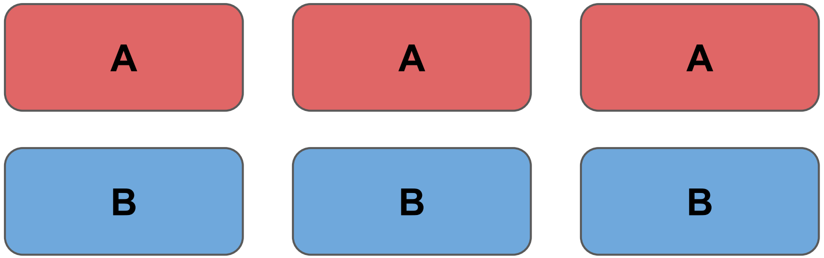
Triple Replication



(4, 2) Erasure Coding

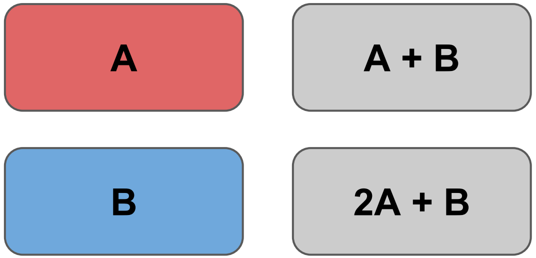
Fault Tolerance: Solution

- Add redundancy to the system



Triple Replication

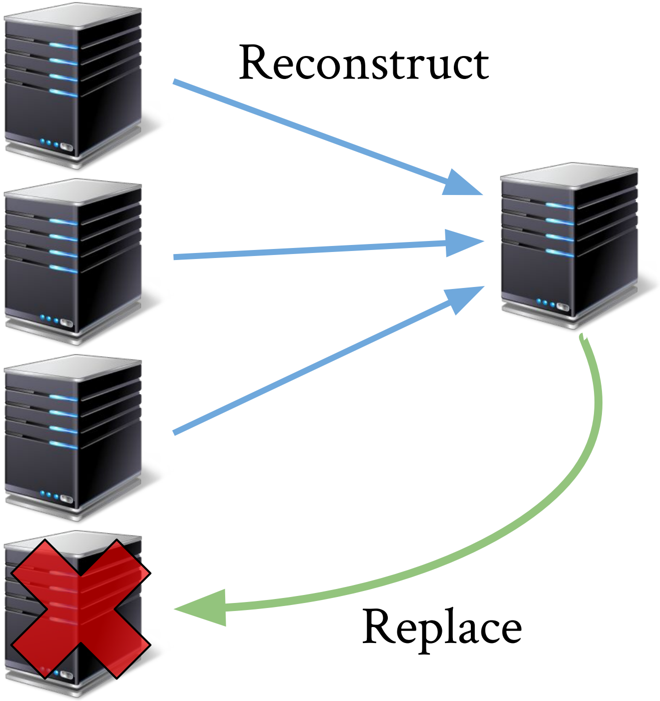
UNSUSTAINABLE



(4, 2) Erasure Coding

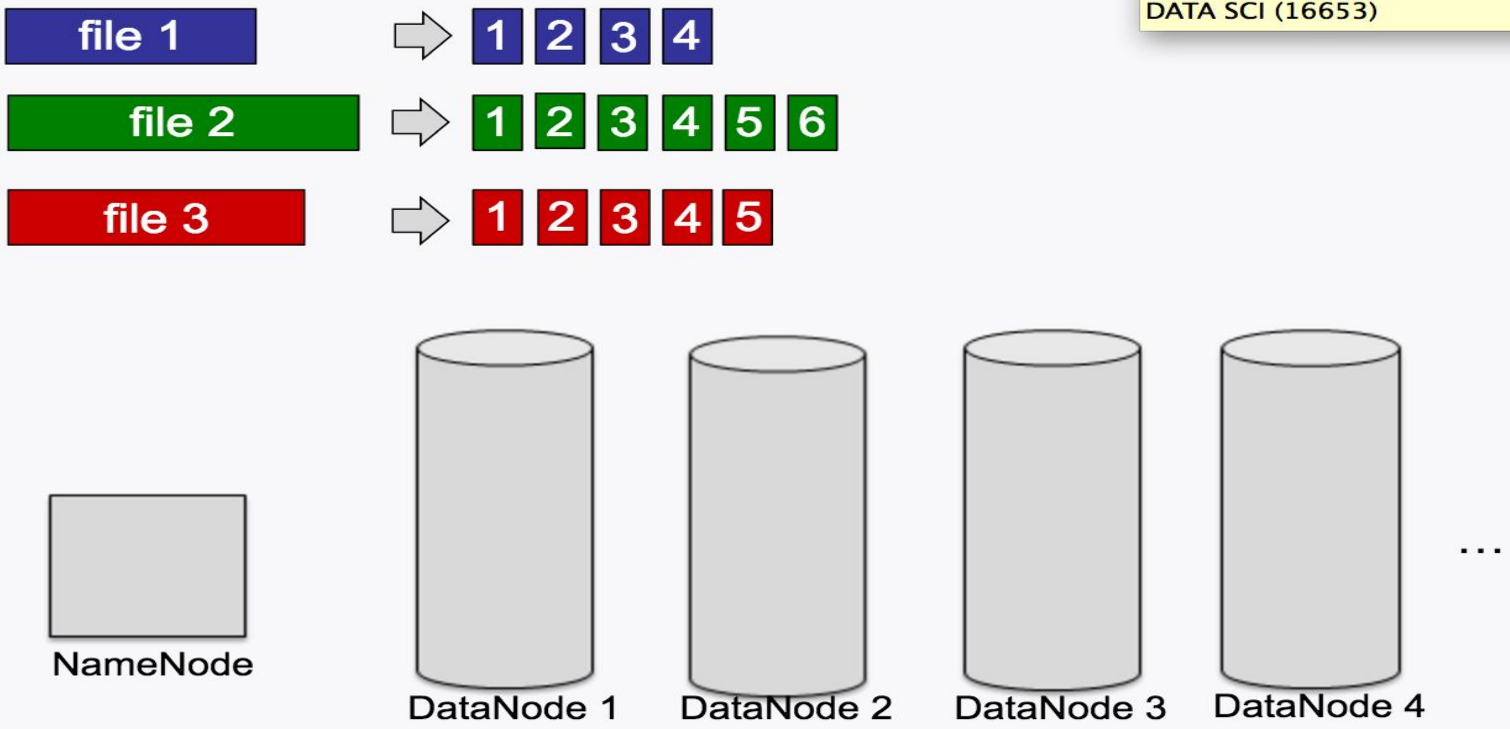
Node Repair

- On drive failure, node needs to be repaired

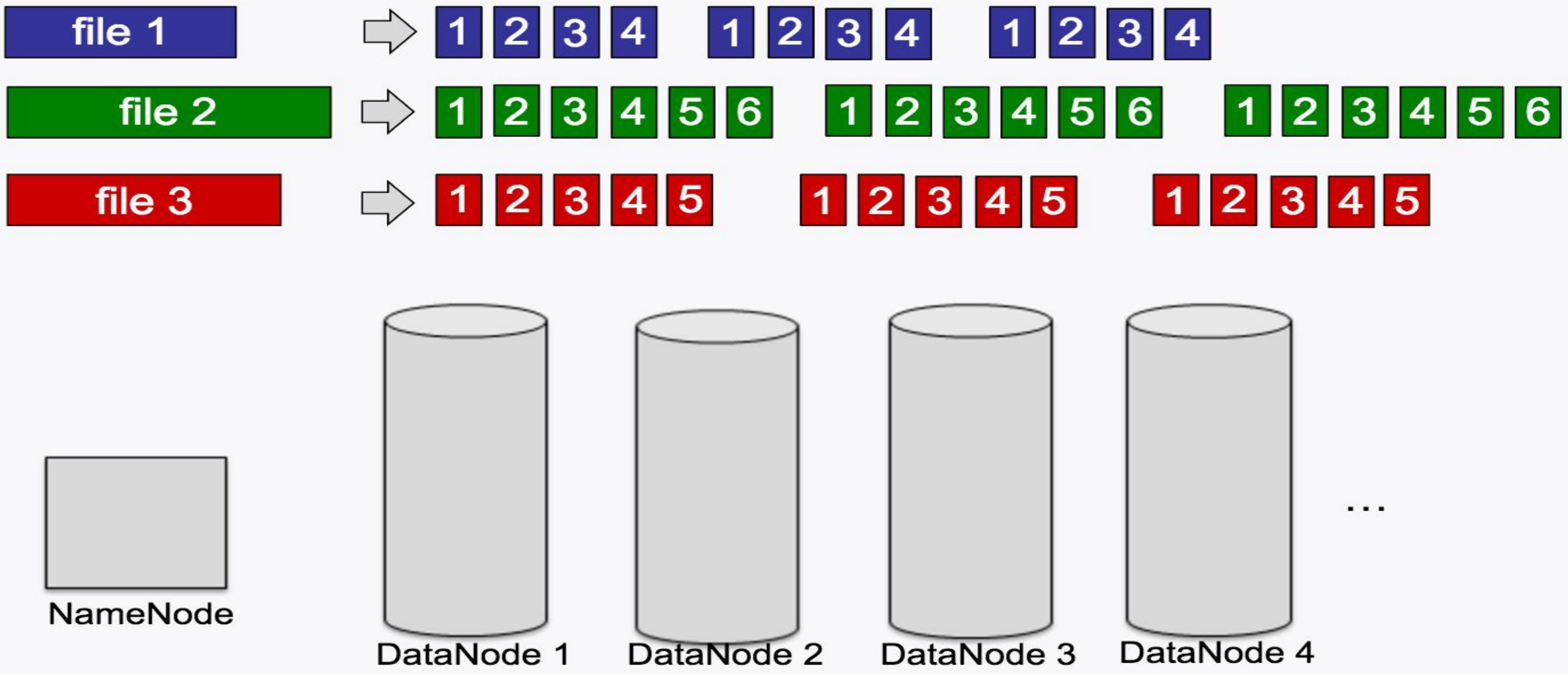


current hadoop architecture

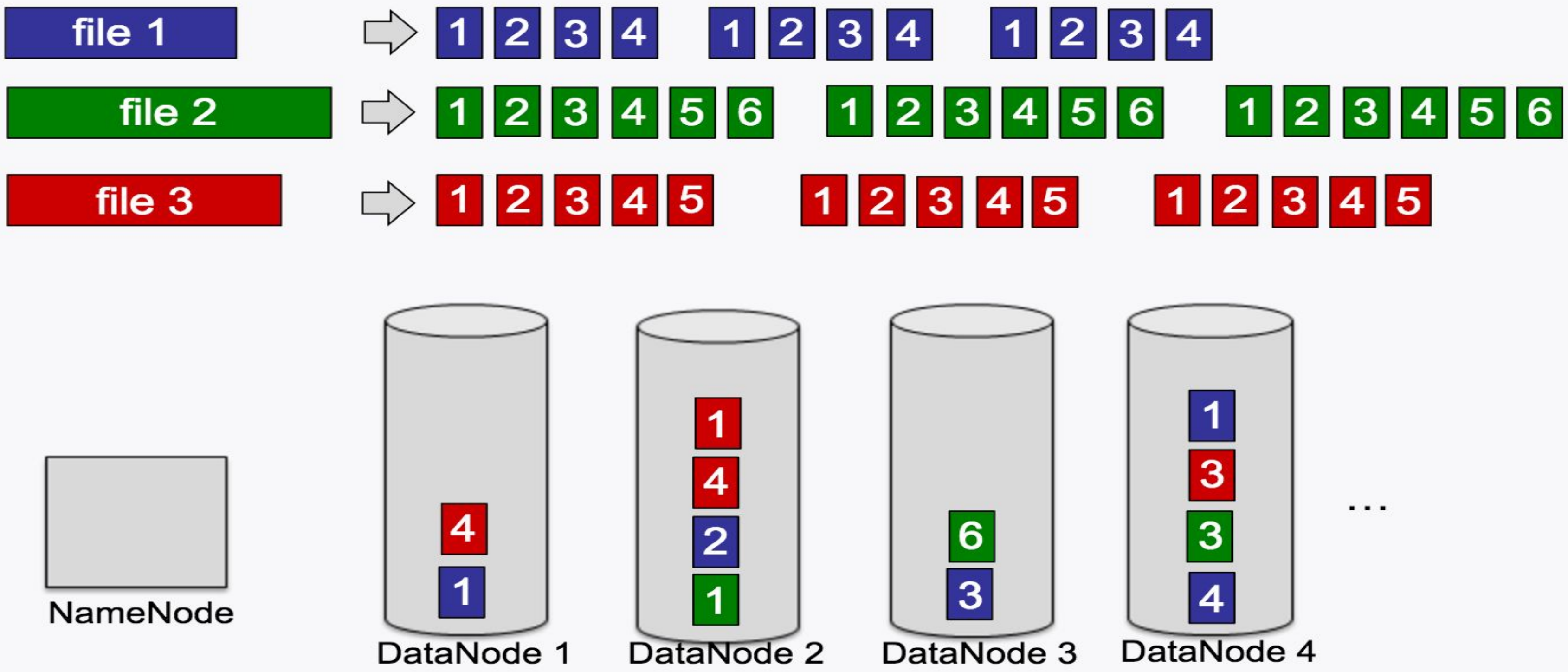
Course Roster: Sp16 – ARCHITECTURE FOR BI
DATA SCI (16653)



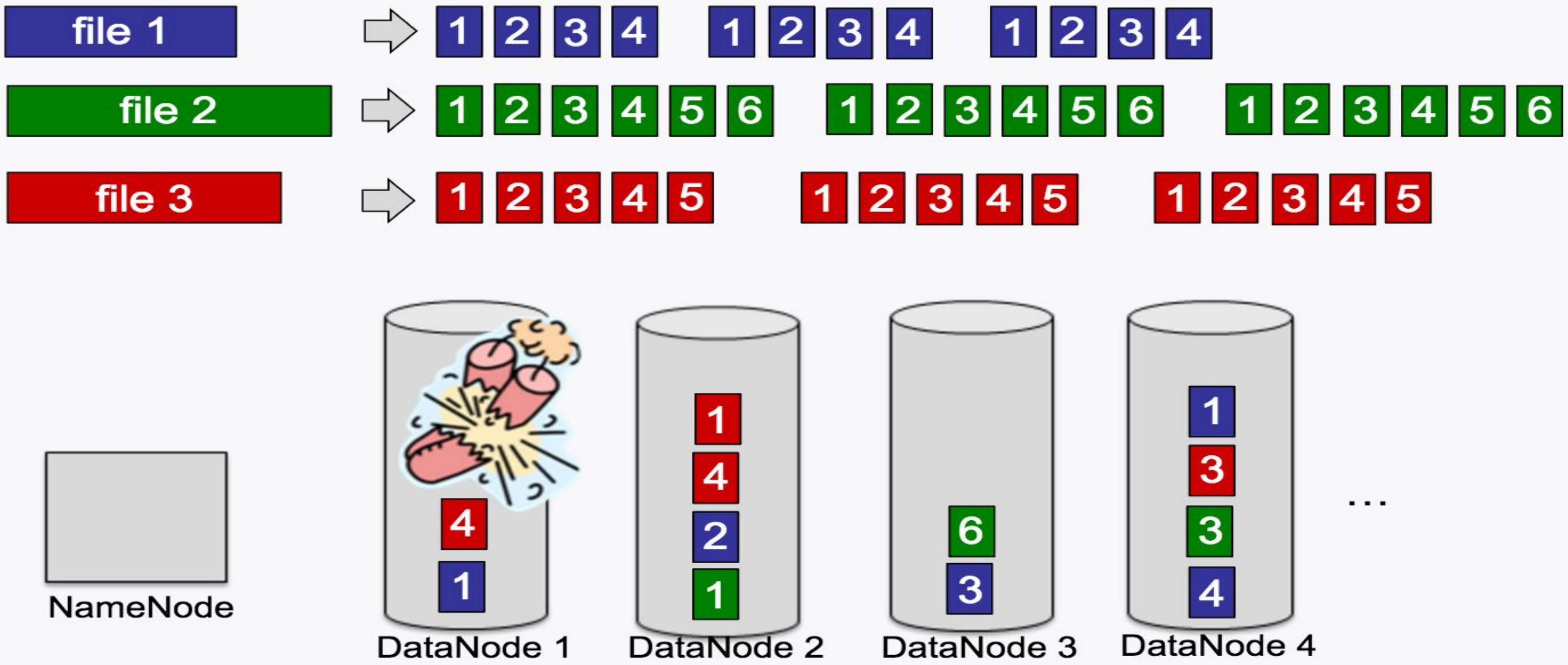
current hadoop architecture



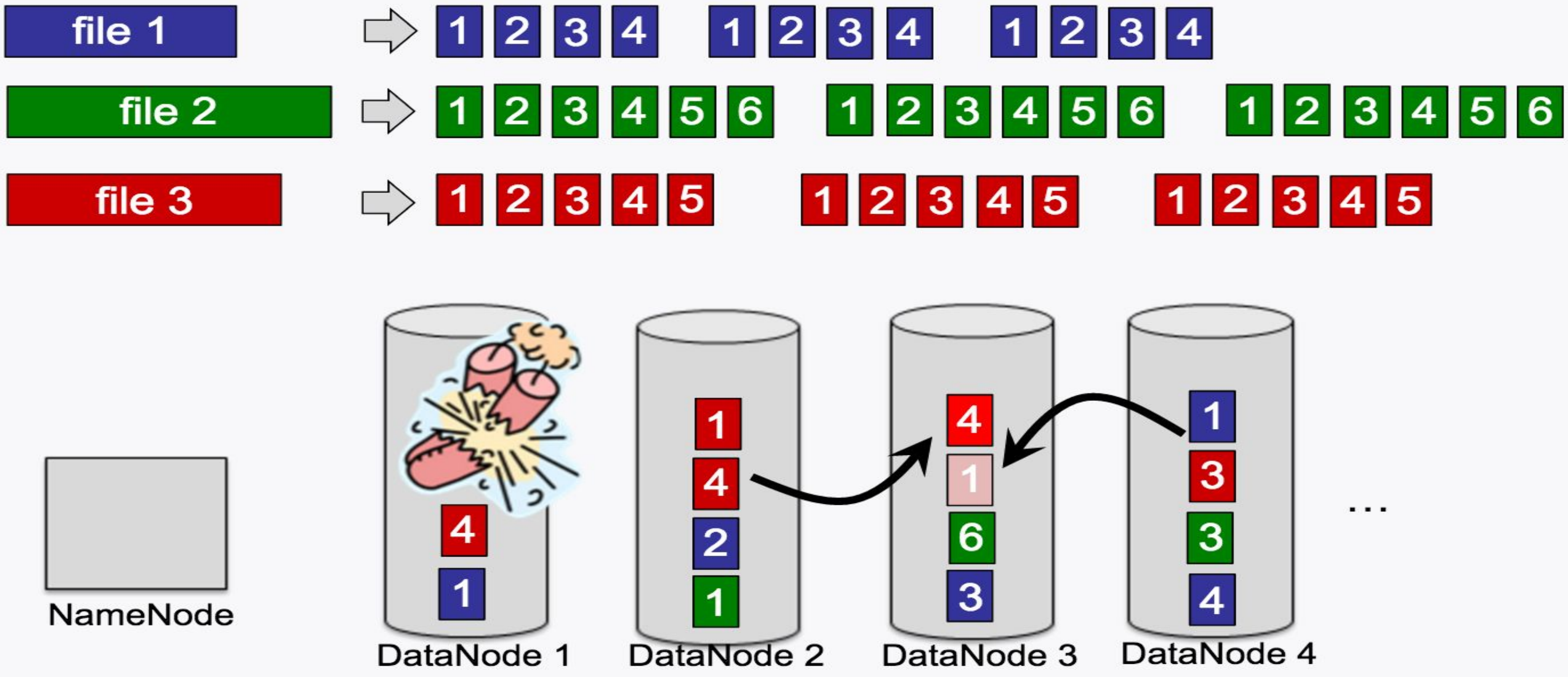
current hadoop architecture



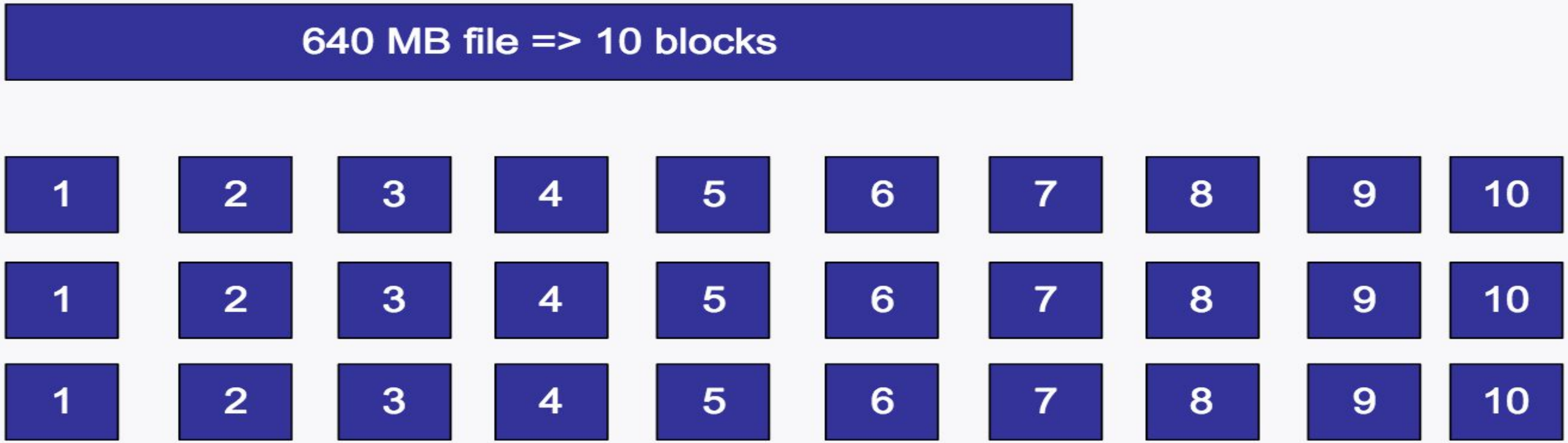
current hadoop architecture



current hadoop architecture

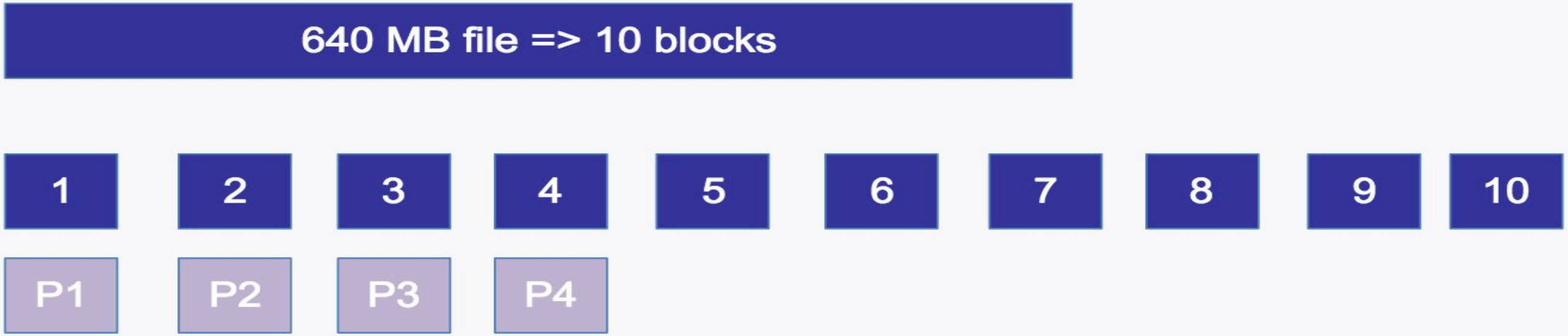


current default hadoop architecture

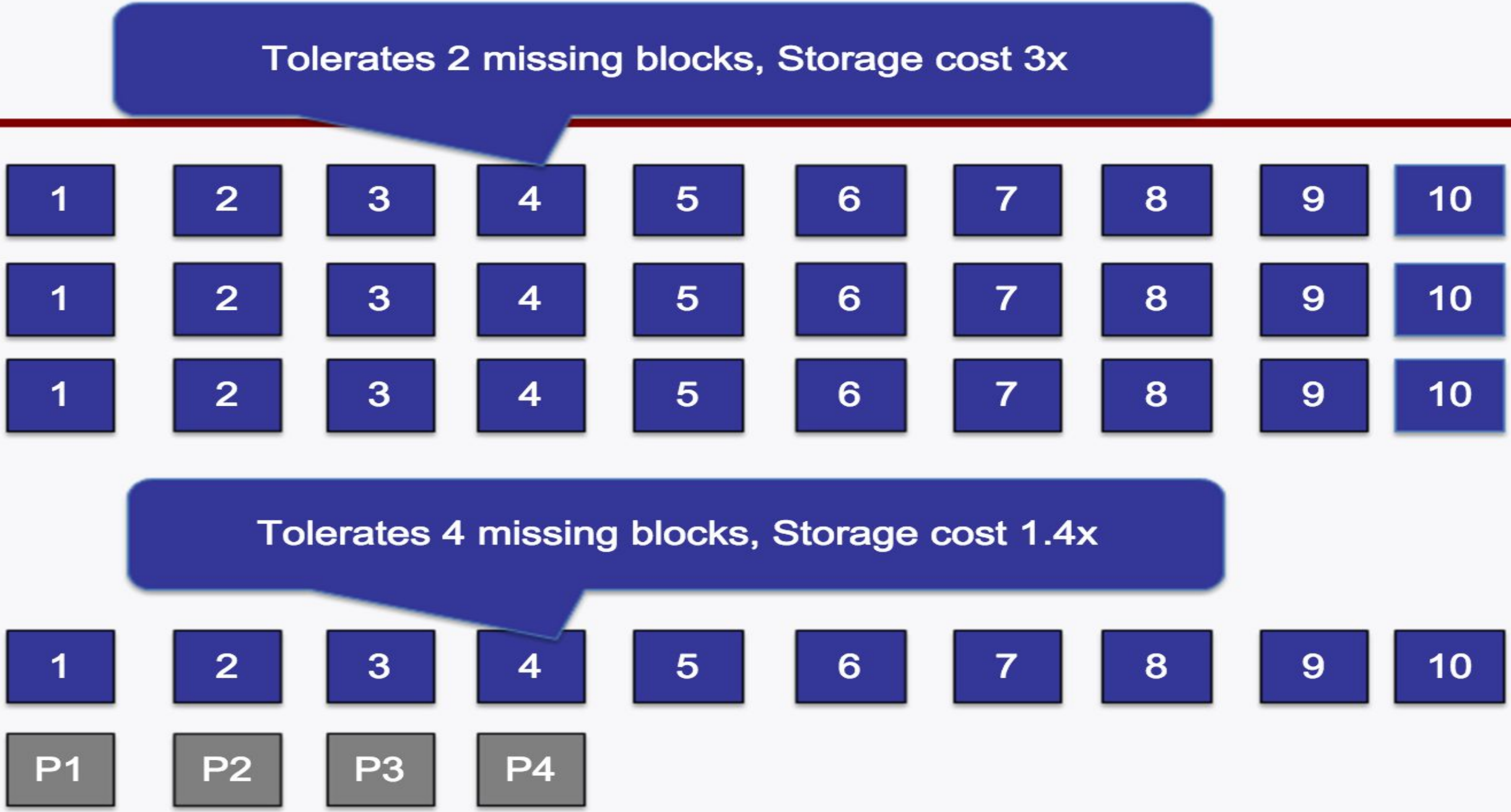


3x replication is HDFS current default.
Very large storage overhead.
Very costly for BIG data

facebook introduced Reed-Solomon (HDFS RAID)



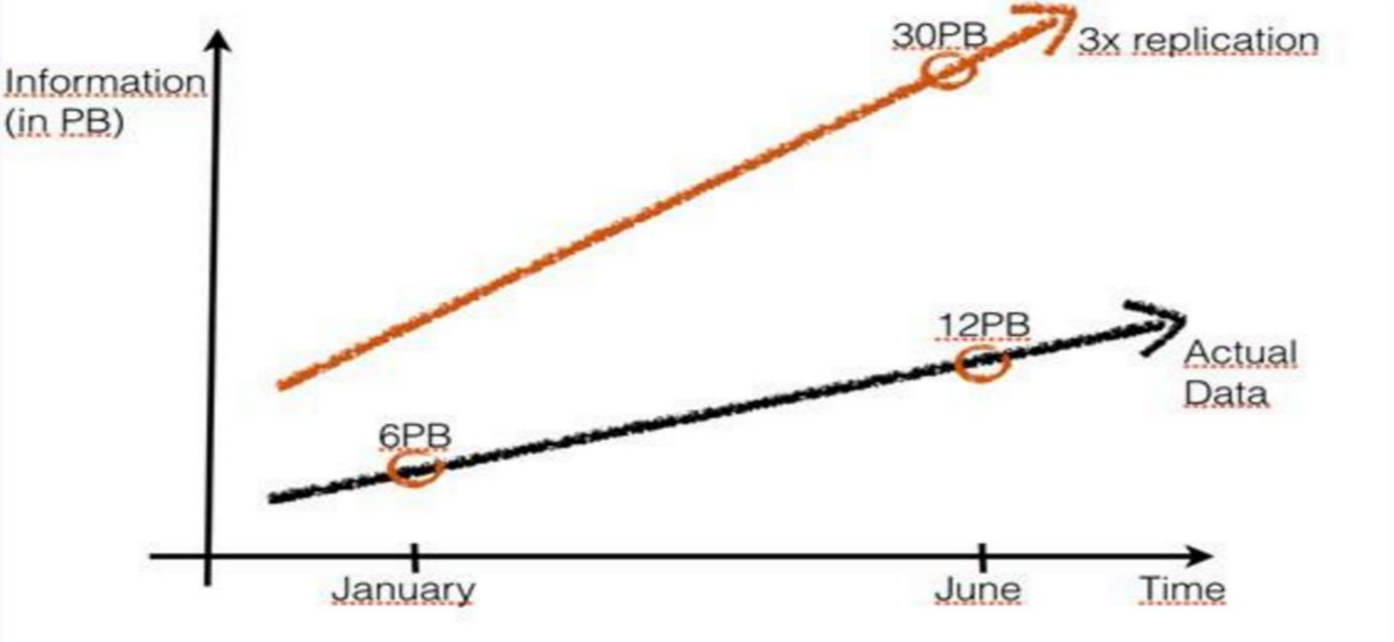
Older files are switched from 3-replication to (14,10) Reed Solomon.



HDFS RAID. Uses Reed-Solomon Erasure Codes

Diskreduce (B. Fan, W. Tantisiriroj, L. Xiao, G. Gibson)

erasure codes save space



Three repair metrics of interest

1. Number of bits **communicated** in the network during single node failures (Repair Bandwidth)

   Capacity known for two points only. My 3-year old conjecture for intermediate points was just disproved. [ISIT13]

2. The number of bits **read** from disks during single node repairs (Disk IO)

   Capacity unknown.
   Only known technique is bounding by Repair Bandwidth

3. The **number of nodes accessed** to repair a single node failure (Locality)

   Capacity known for some cases.
   Practical LRC codes known for some cases. [ISIT12, Usenix12, VLDB13]
   General constructions open

CAP Theorem

CAP Theorem: Introduction

- Three attributes used when describing distributed storage
- Consistency
- Availability
- Partition tolerance

CAP Theorem: Consistency

- A read is guaranteed to return the most recent write
- If data is replicated, any write **must** update all copies
- More copies implies more indirect writes
    - Requires communication bandwidth and time
    - Degrades performance

CAP Theorem: Availability

- A functioning node **eventually** will respond to your request

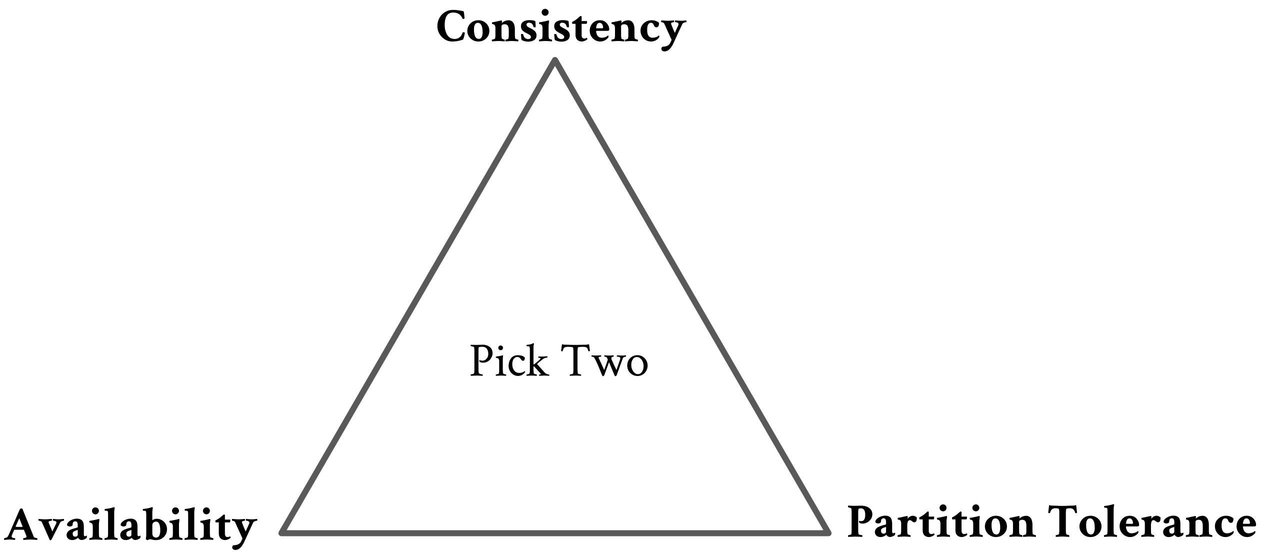
- Functioning nodes must not return errors or timeouts
  - Why would this happen?

CAP Theorem: Partition Tolerance

- System will function even if network becomes partitioned

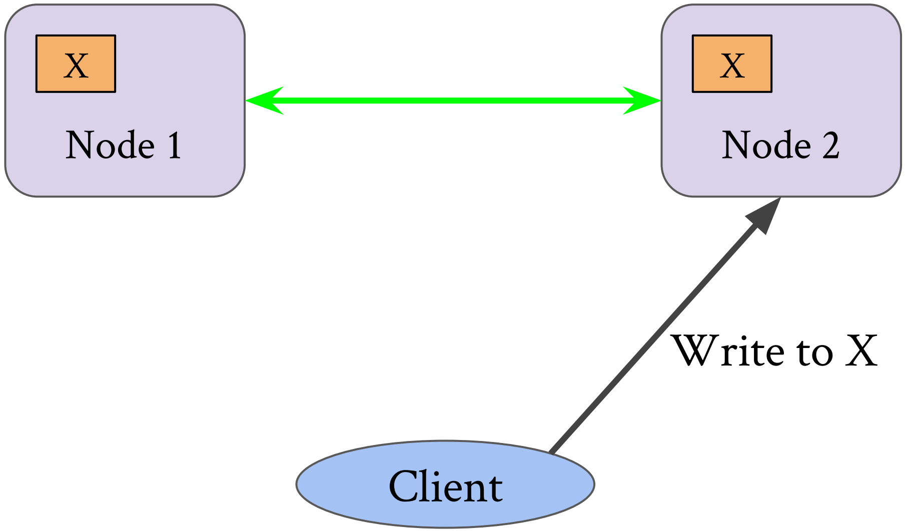
CAP Theorem

- CAP theorem asserts that you **cannot** have all three
- At best you can have two

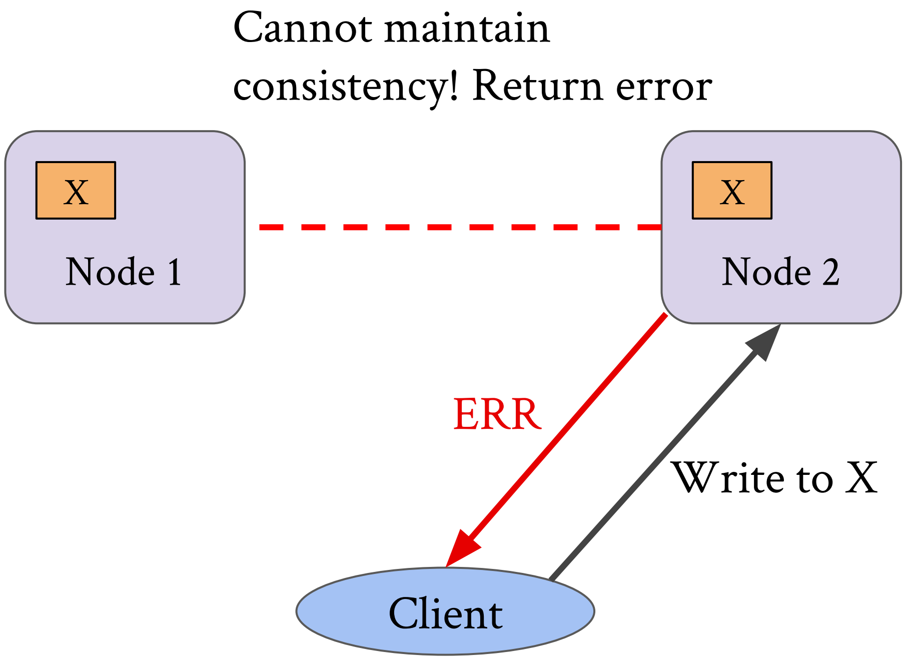


Consistency and Partition Tolerance

- System is consistent and partition tolerant, but not available

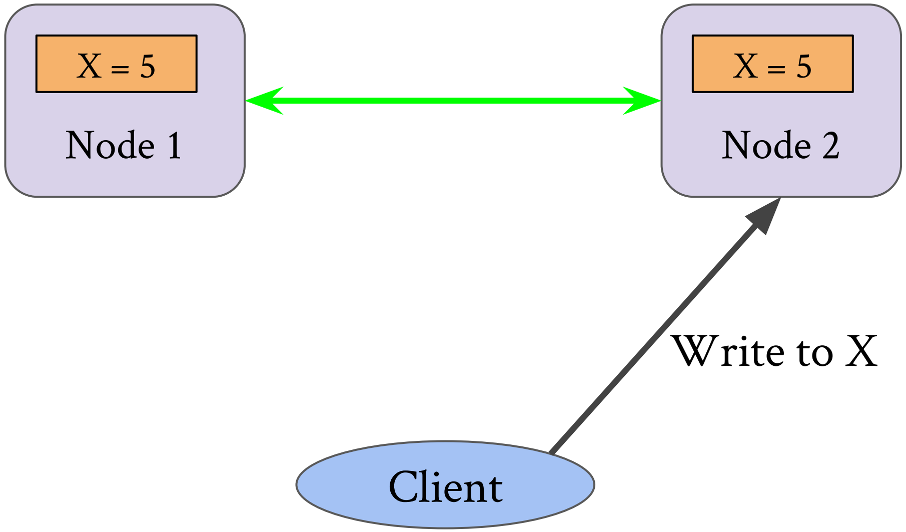


Consistency and Partition Tolerance

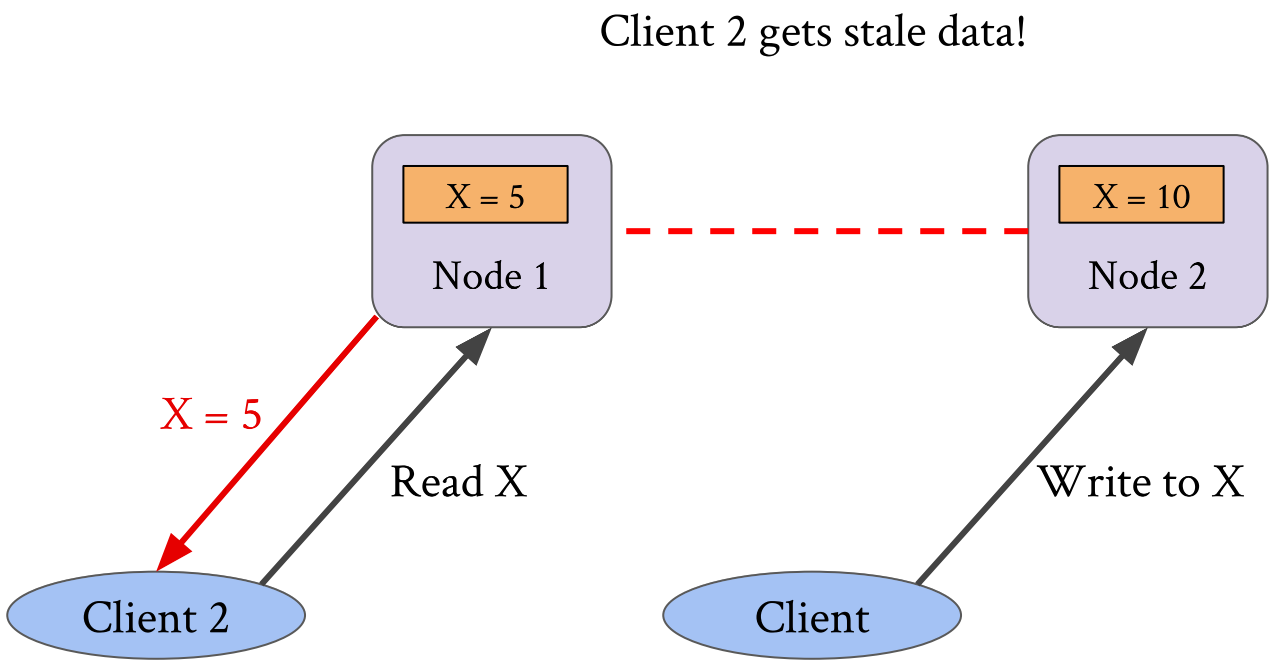


Availability and Partition Tolerance

- System is available and partition tolerant, but not consistent



Availability and Partition Tolerance



CAP Theorem: On Partition Tolerance

- In reality, distributed systems **must** be partition tolerant
- In any reasonably sized system, network failures **will** happen
- Your only choice is between consistency and availability
- Choose consistency when you require atomic operations
- Choose availability when system response is important

Replication/Coding and Consistency

Replication/Coding Dilemma

- All replicas must be kept consistent
  - Otherwise, what’s the point?
- Paradox:
  - Replicate data for better performance
  - Modifying one copy triggers a write to every other copy
  - Volume of writes uses bandwidth and reduces performance

Consistency Models

- A **consistency model** specifies when and how modifications are made to replicas
- Many different types:
  - Strict consistency
  - Sequential consistency
  - Causal consistency
  - Eventual consistency

Strict Consistency

- Strongest form of consistency
- Any execution is the same as if all reads and writes were executed in the wall-clock order that they were issued
- Pros: Reads always return the most recent value
- Cons: Poor performance. Why?

Eventual Consistency

- Weakest form of consistency
- Any read will **eventually** get the most recent value
    - “Eventually” could be a long time
- Pros: High performance. Why?
- Cons: Reads can get stale values

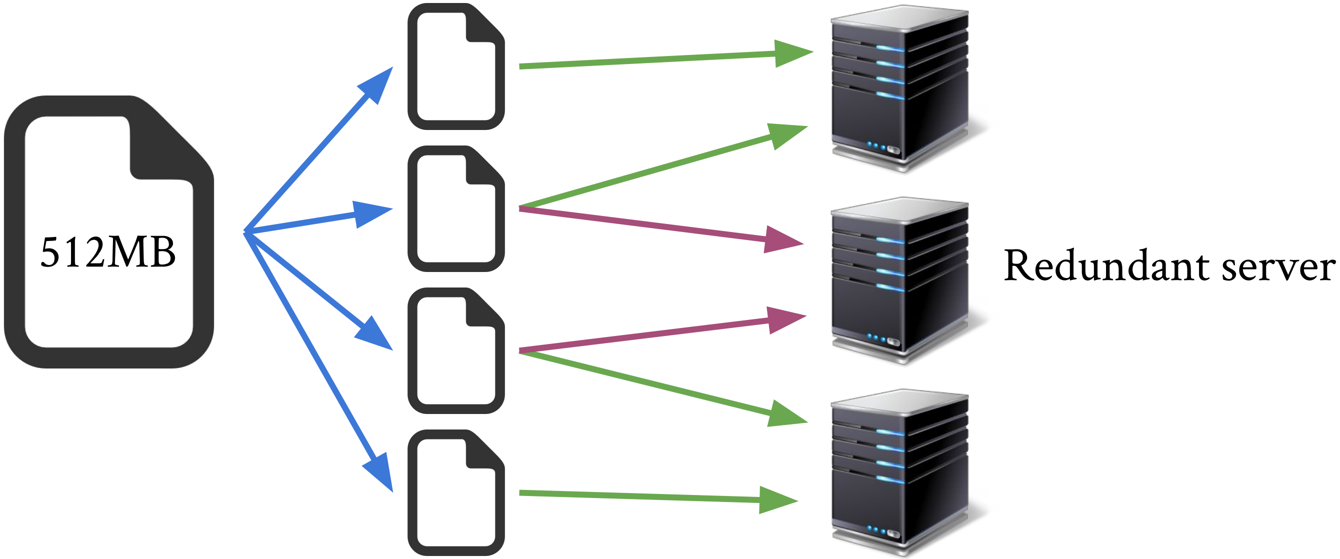
Hadoop Distributed File System (HDFS)

HDFS: Intro

- Hadoop is an open-source MapReduce framework by Apache
  - We will be using it for lab1
- HDFS is the file system used by Hadoop
- Designed to:
  - Store large data sets
  - Provide high reliability
  - Provide high bandwidth to user applications

HDFS Architecture: Blocks

- HDFS is a **block-structured** file system
  - Files are divided into 128MB blocks by default
  - Blocks are spread out and replicated across a cluster



HDFS Architecture: NameNodes and DataNodes

- Two node types: NameNode and DataNode
- NameNode - stores file **metadata**
  - Permissions, sizes, list of blocks
  - Considered the master
- DataNode - stores actual application data
  - Slave nodes
- All communication over TCP

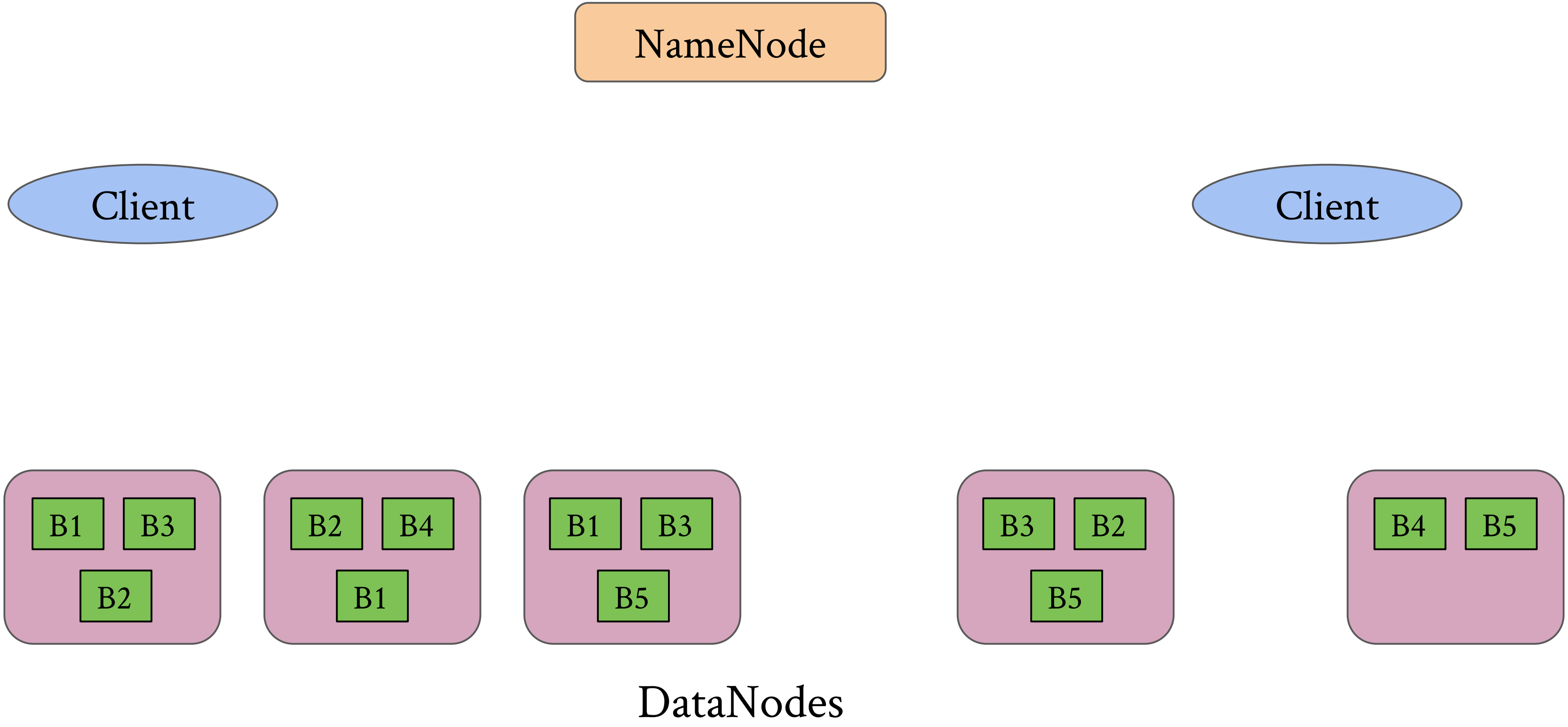
HDFS Architecture: NameNode Duties

- Only one NameNode
- Maintains the directory hierarchy
  - Handles open/close/rename requests
- Handles authentication
- Collects block reports from DataNodes
- Issues block commands to DataNodes

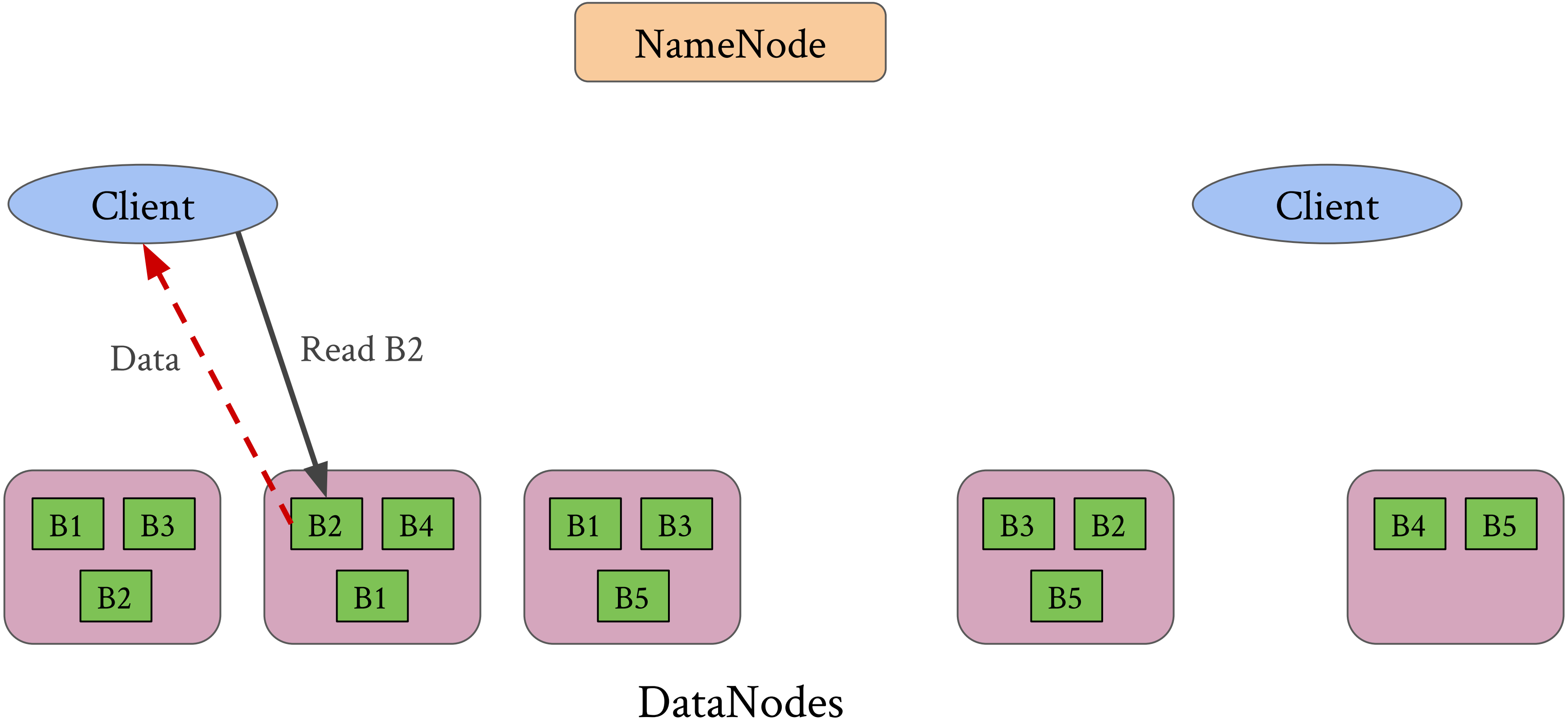
HDFS Architecture: DataNode Duties

- Many DataNodes
- Handle read/write requests from clients
- Perform block creation, deletion, and replication
  - Follow NameNodes orders
- Send block reports to NameNode

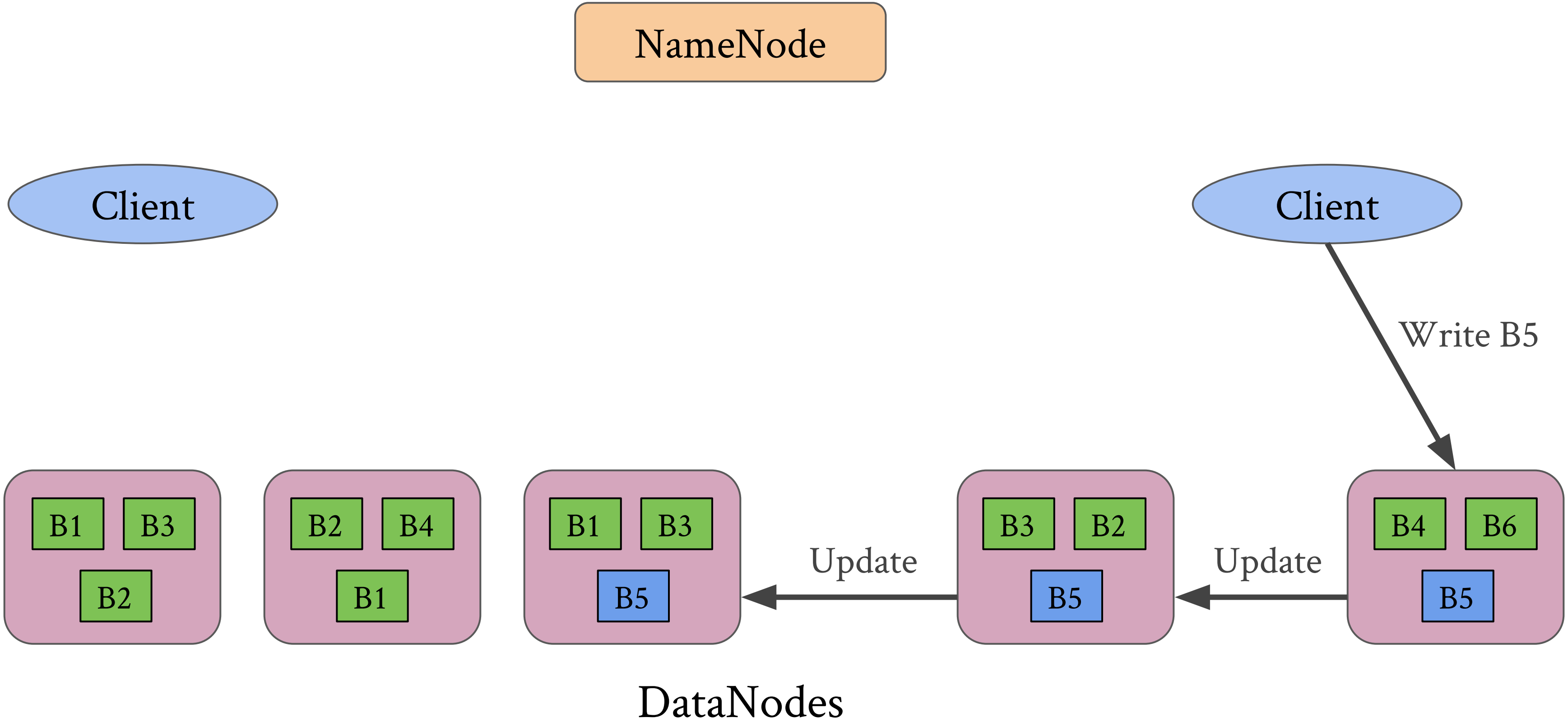
HDFS Architecture with replication



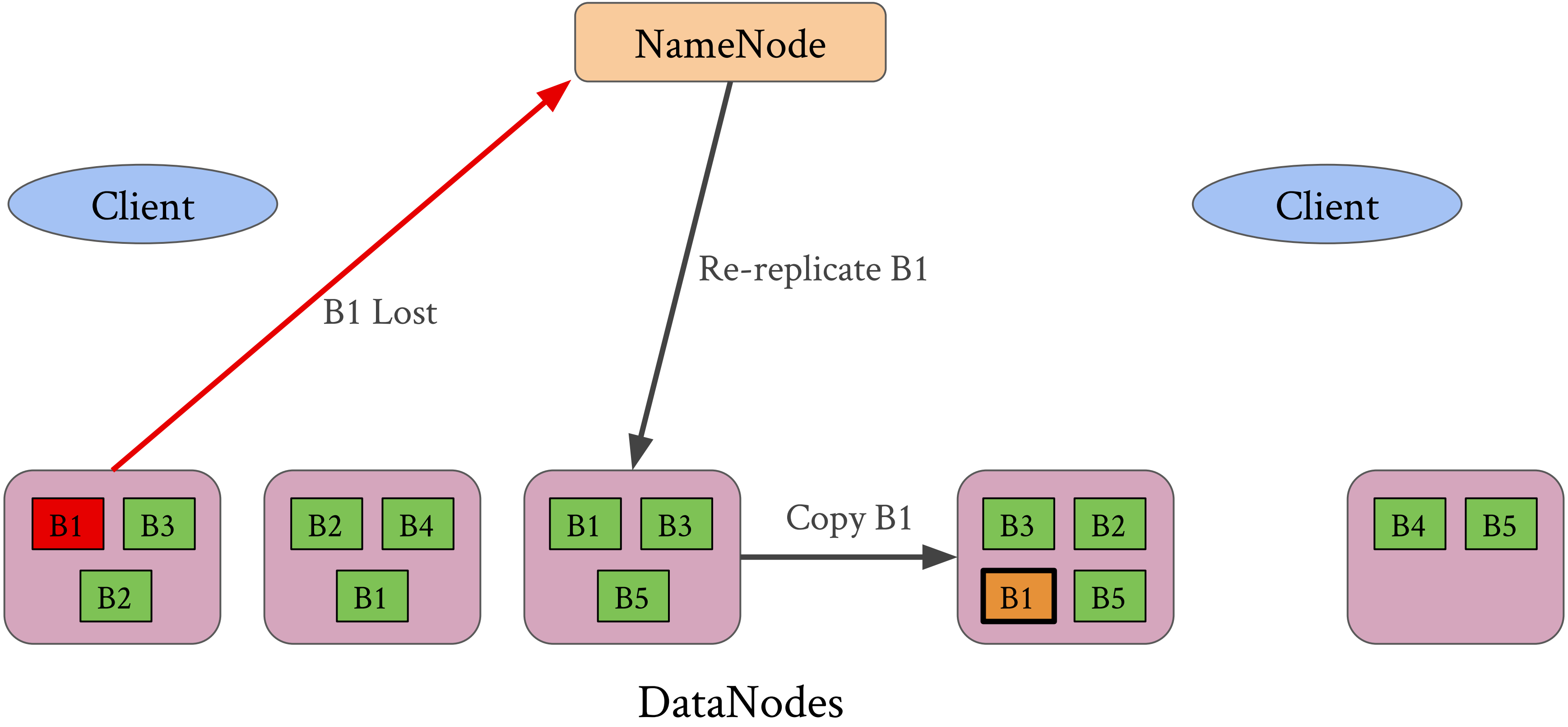
HDFS Architecture: Block Read



HDFS Architecture: Block Write



HDFS Architecture: Block Data Loss



Conclusion

Distributed data storage a challenge

- Fundamental limits in design: CAP theorem

Efficiency in storage through coding

Particular implementations:

HDFS, GFS and many others: HDFS used in assignment

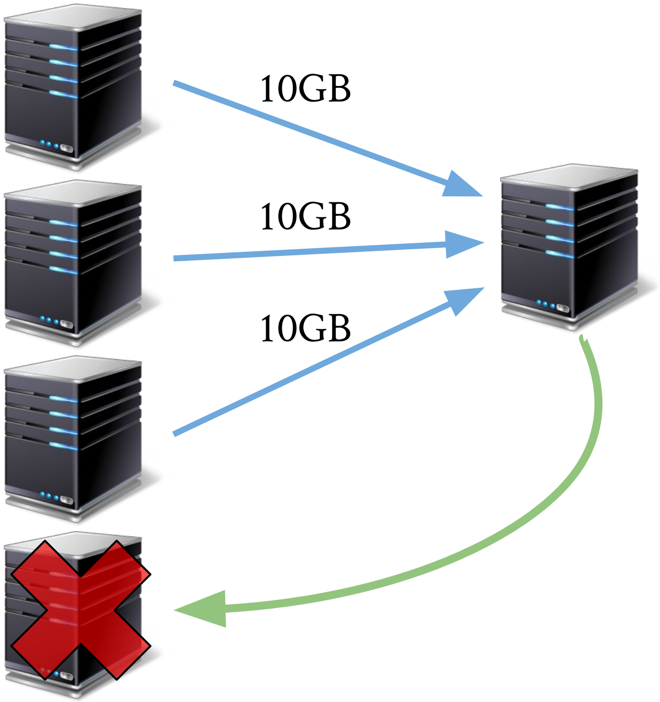
HDFS Architecture: Heartbeats

- Method of communication between NameNode and DataNodes

- DataNodes send heartbeats to NameNode
    - Signifies that DataNode is working correctly
    - Block replicas on the node are available

Node Repair: Repair Bandwidth

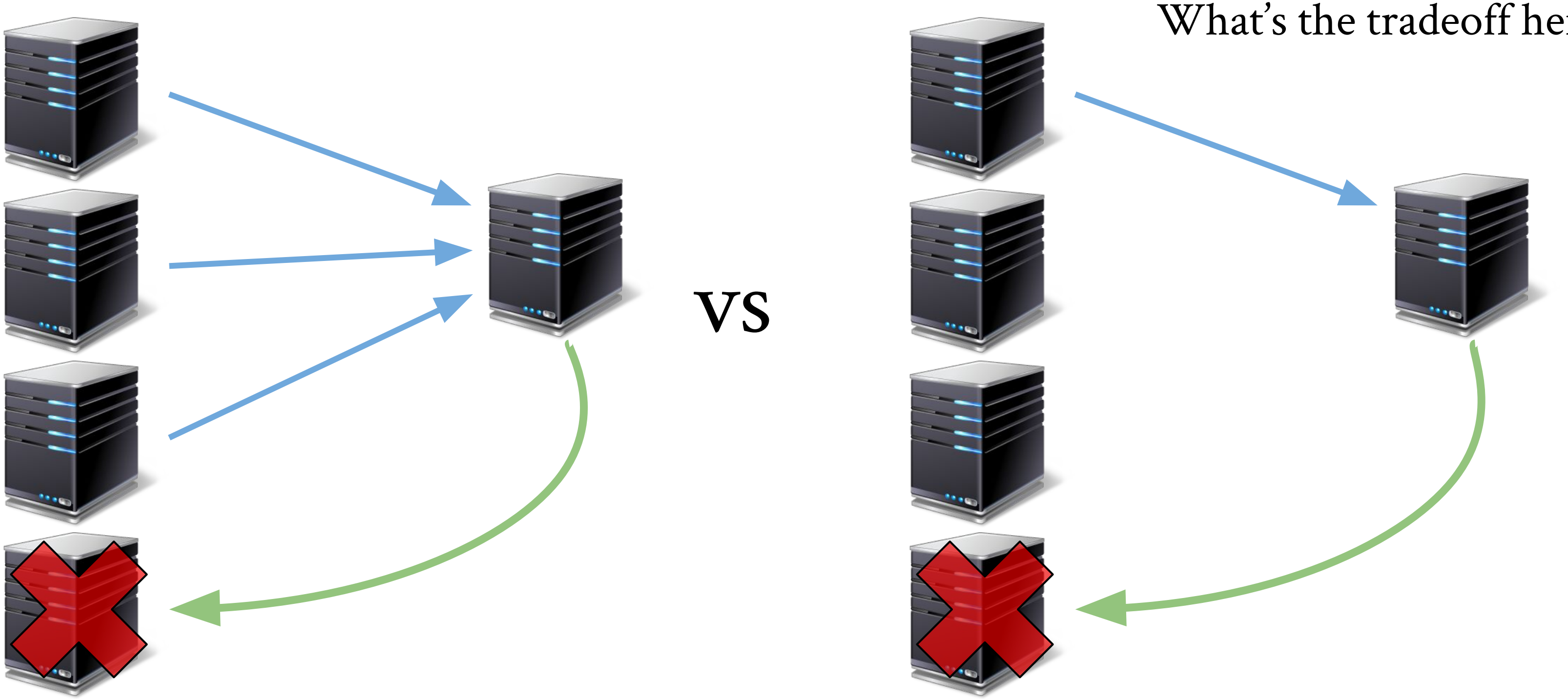
- How much data needs to be downloaded to reconstruct?



Node Repair: Locality

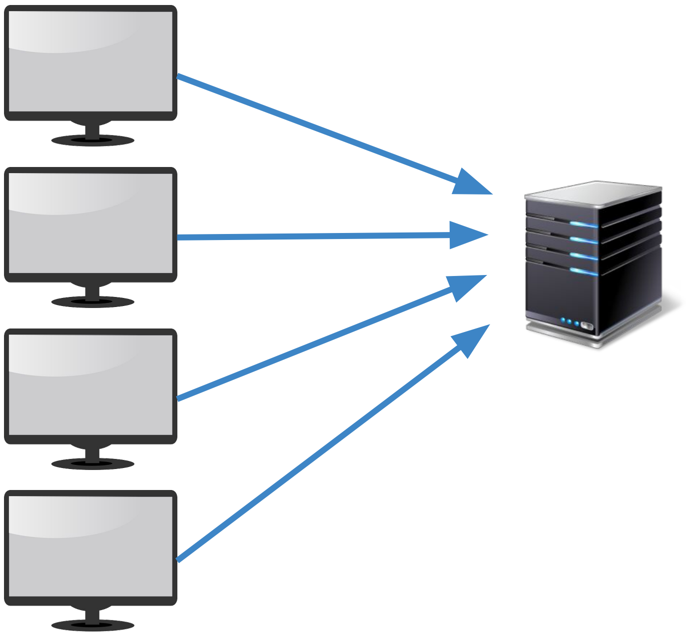
- Number of nodes accessed for a repair

What's the tradeoff here?



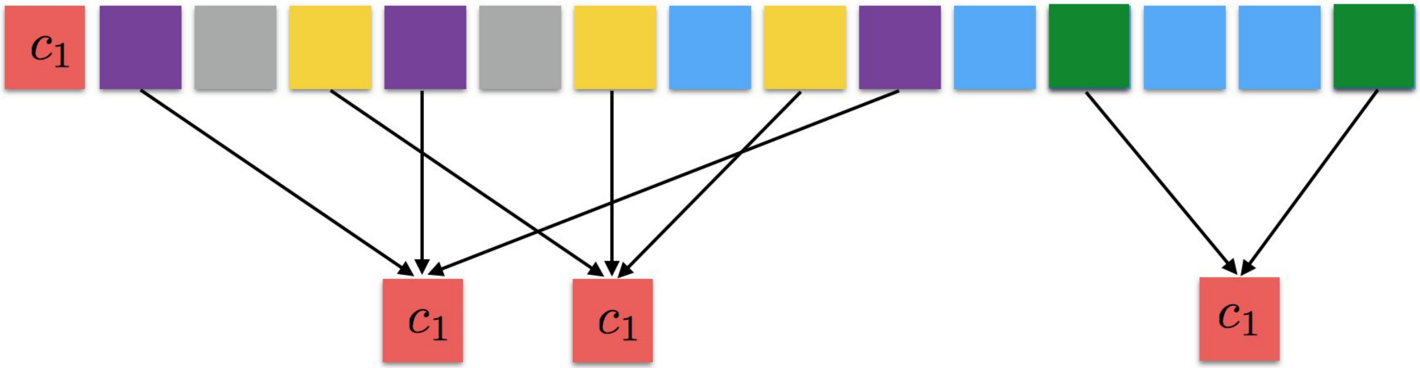
Parallel Access

- Support multiple users accessing the same data



Parallel Access

- 4 parallel read groups, locality less than 3



Parallel Access: Hot Data

- Hot data requires fast access by multiple concurrent users
  - Facebook photos
  - Google Maps
- CERN labels 13PB out of its 100PB dataset as hot data
- Replicate hot data on more machines
  - More machines $\Rightarrow$ more bandwidth

Security

- Prevent data snooping during regular operation and node repairs