

# Google File System

*CS 475: Concurrent & Distributed Systems (Fall 2021)*

Lecture 5-2

Yue Cheng

Some material taken/derived from:

- Princeton COS-418 materials created by Michael Freedman and Wyatt Lloyd.
- MIT 6.824 by Robert Morris, Frans Kaashoek, and Nickolai Zeldovich.

Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

# Review: MapReduce assumptions

- Commodity hardware
  - Economies of scale!
  - Commodity networking with less bisection bandwidth
  - Commodity storage (hard disks) is cheap
- Failures are common
- Replicated, distributed file system for data storage

# Review: Fault tolerance

- If a task crashes:
  - Retry on another node
    - Why this is okay?
  - If the same task repeatedly fails, end the job

# Review: Fault tolerance

- If a task crashes:
  - Retry on another node
    - Why this is okay?
  - If the same task repeatedly fails, end the job
- If a node crashes:
  - Relaunch its current tasks on another node
    - What about task inputs?

# Google file system (GFS)

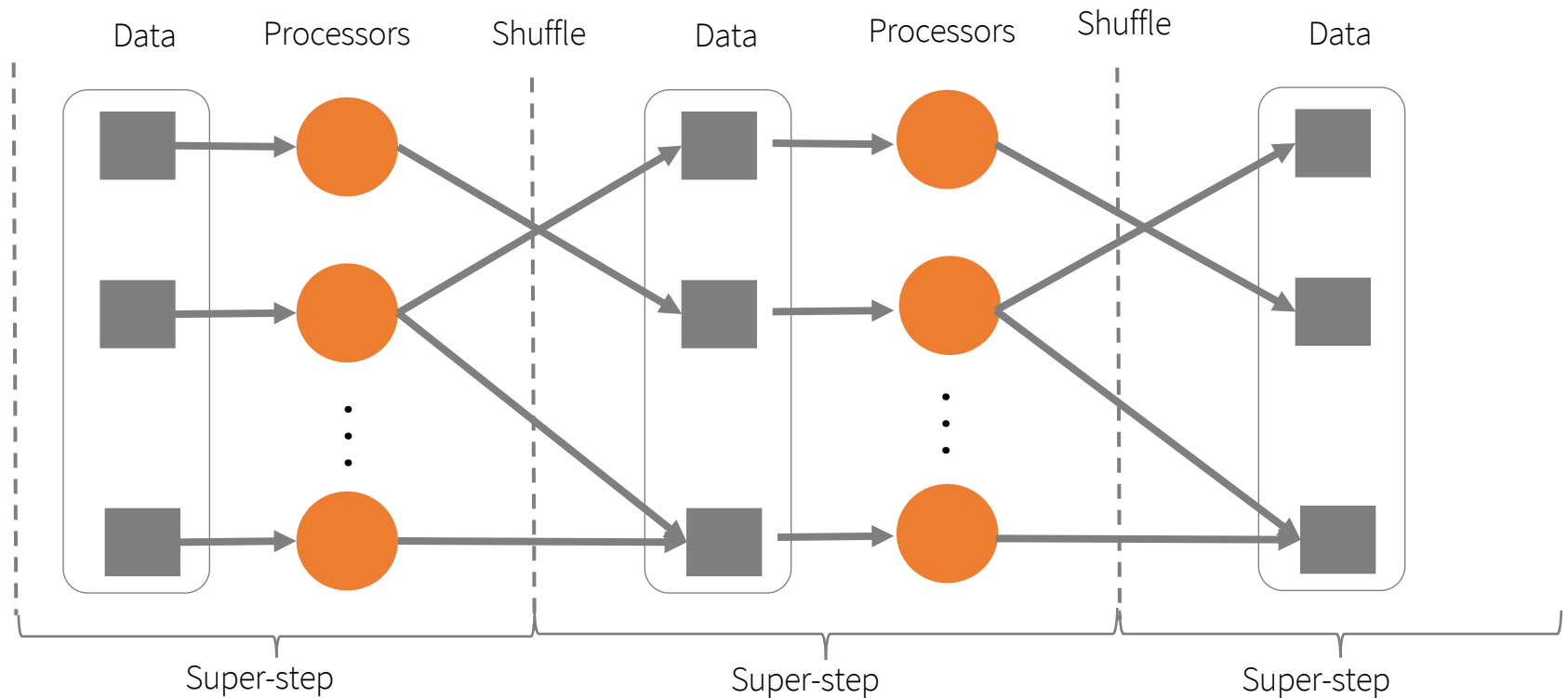
- Goal: a global (distributed) file system that stores data across many machines
  - Need to handle 100's TBs
- Google published details in 2003
- Open source implementation:
  - Hadoop Distributed File System (HDFS)



# Workload-driven design

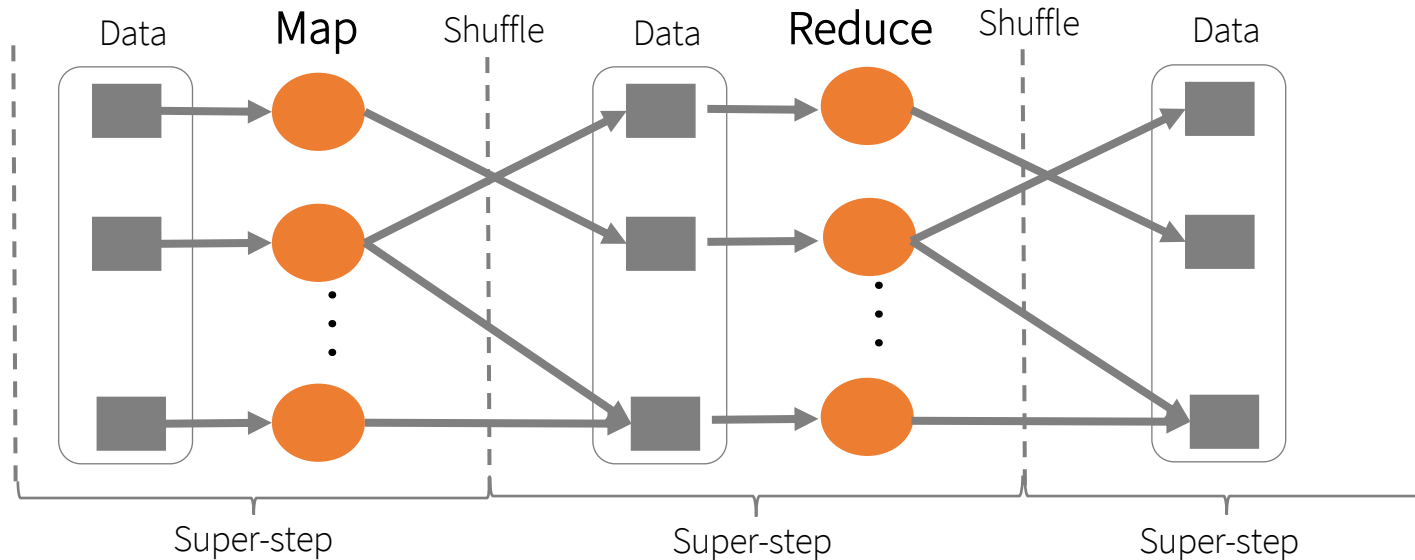
- MapReduce workload characteristics
  - Huge files (GBs)
  - Almost all writes are appends
  - Concurrent appends common
  - High throughput is valuable
  - Low latency is not

# Example workloads: Bulk Synchronous Processing (BSP)



\*Leslie G. Valiant, A bridging model for parallel computation, Communications of the ACM, Volume 33 Issue 8, Aug. 1990

# MapReduce as a BSP system



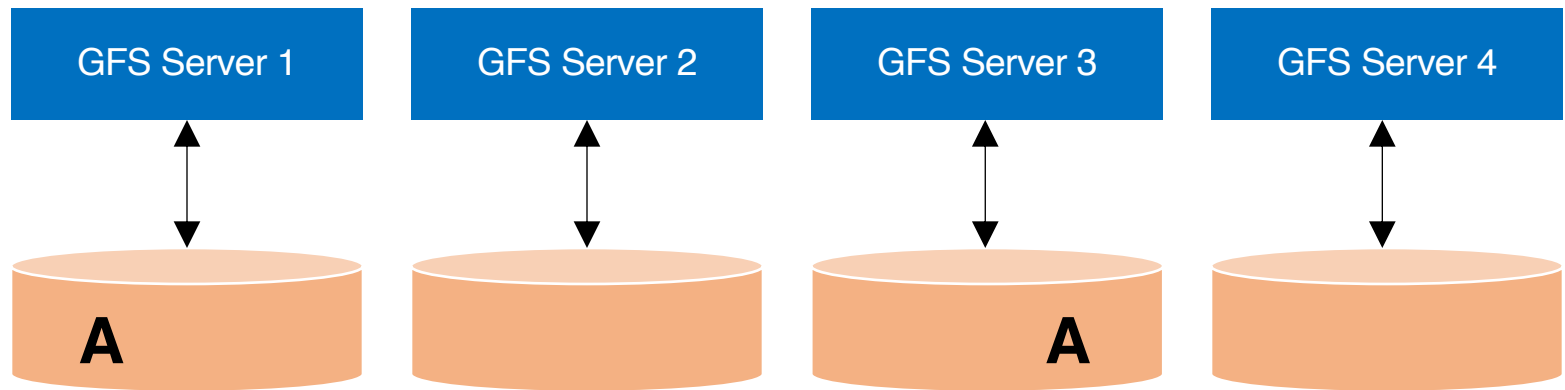
- Read entire dataset, do computation over it
  - Batch processing
- Producer/consumer: many producers append work to file concurrently; one consumer reads and does work



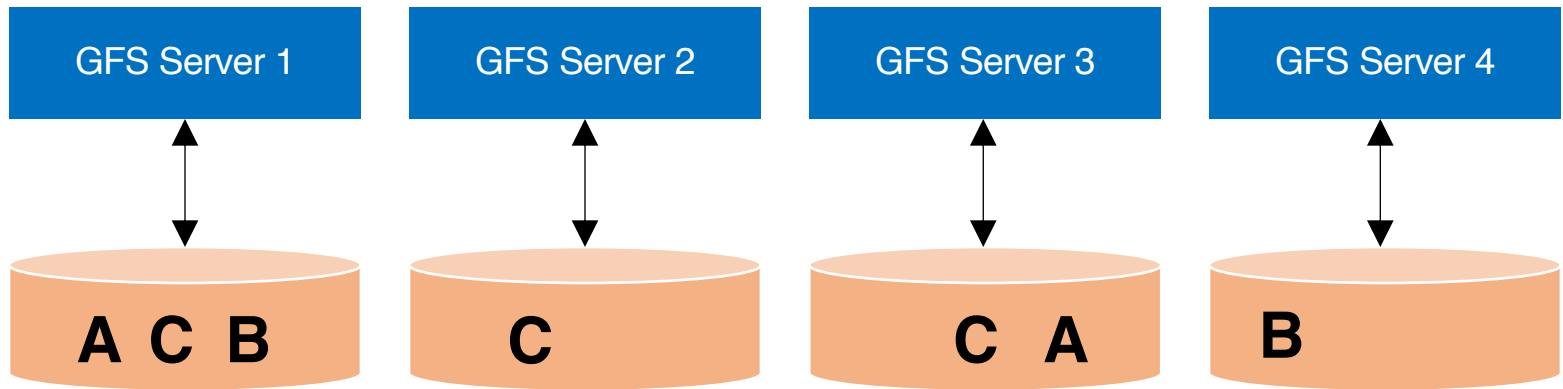
# Workload-driven design

- Build a global (distributed) file system that incorporates all these application properties
- Only supports **features required by applications**
- Avoid difficult local file system features, e.g.:
  - rename dir
  - links

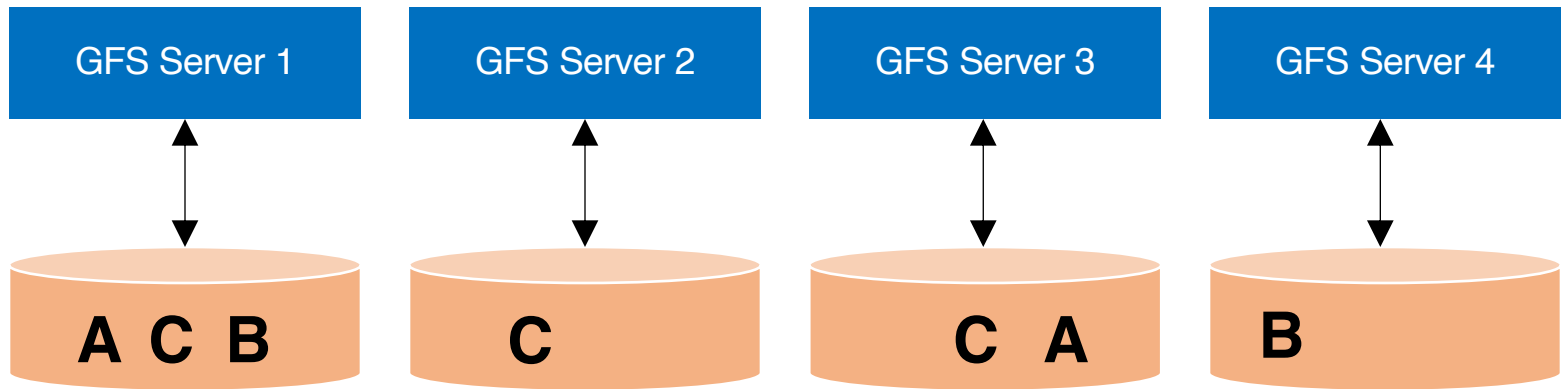
# Replication



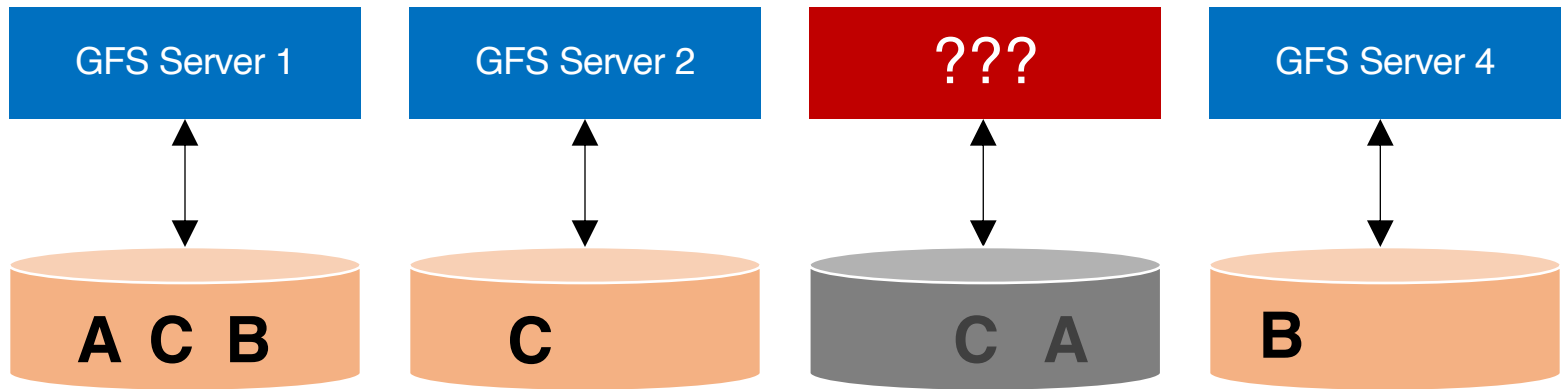
# Replication



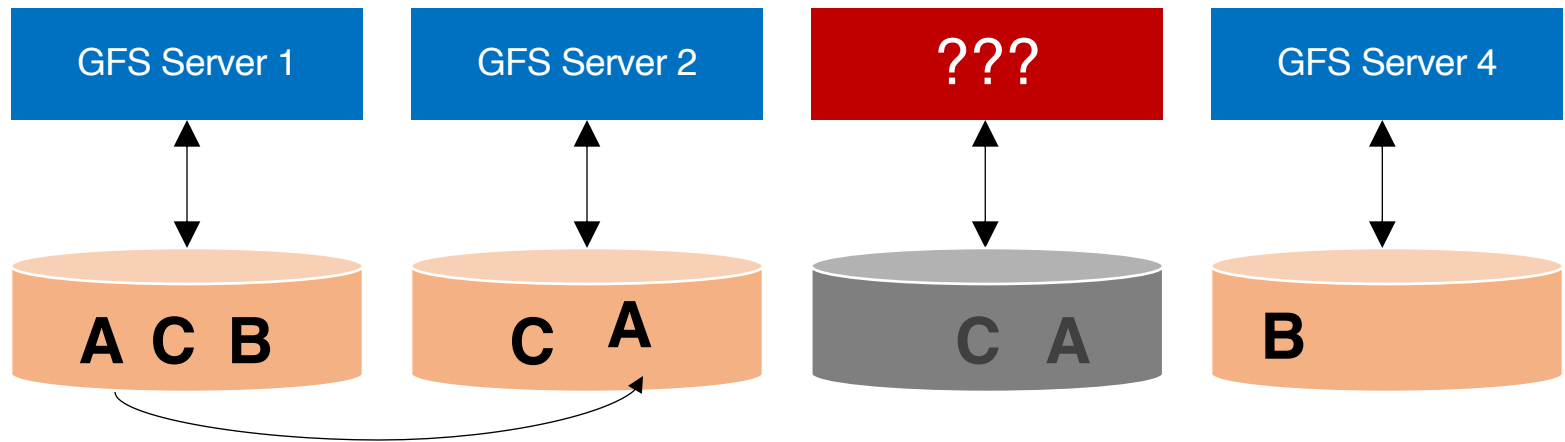
# Resilience against failures



# Resilience against failures

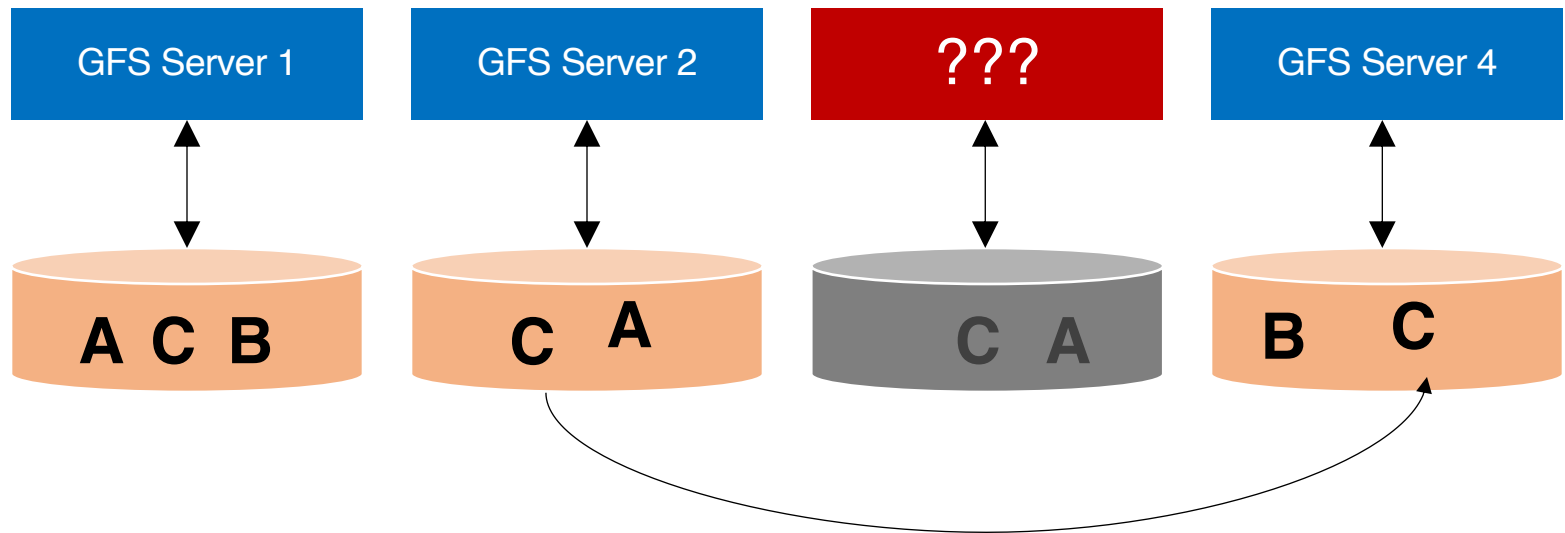


# Data recovery



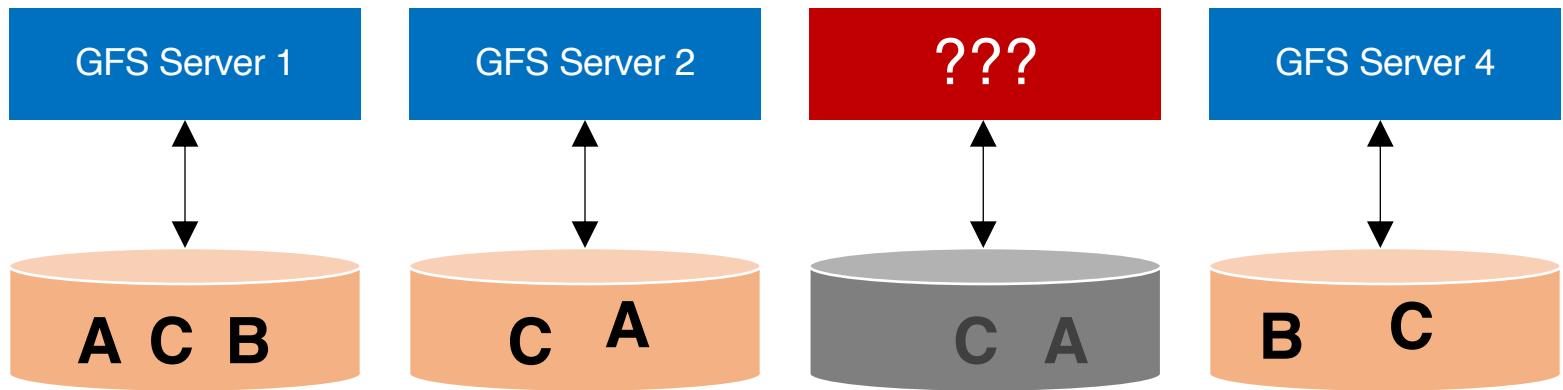
Replicating A to maintain a replication factor of 2

# Data recovery



Replicating C to maintain a replication factor of 3

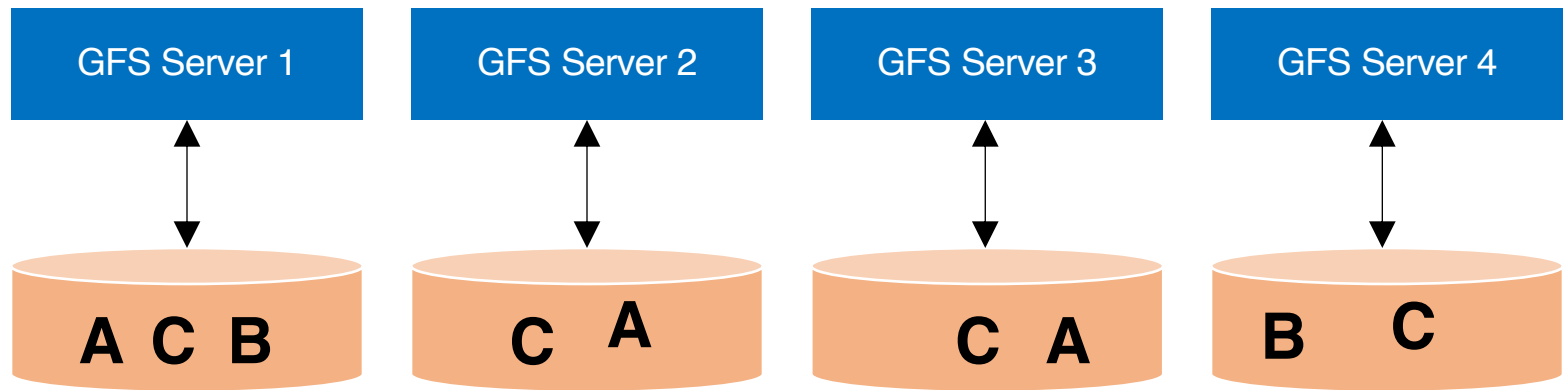
# Data recovery



Machine may be dead forever, or it may come back

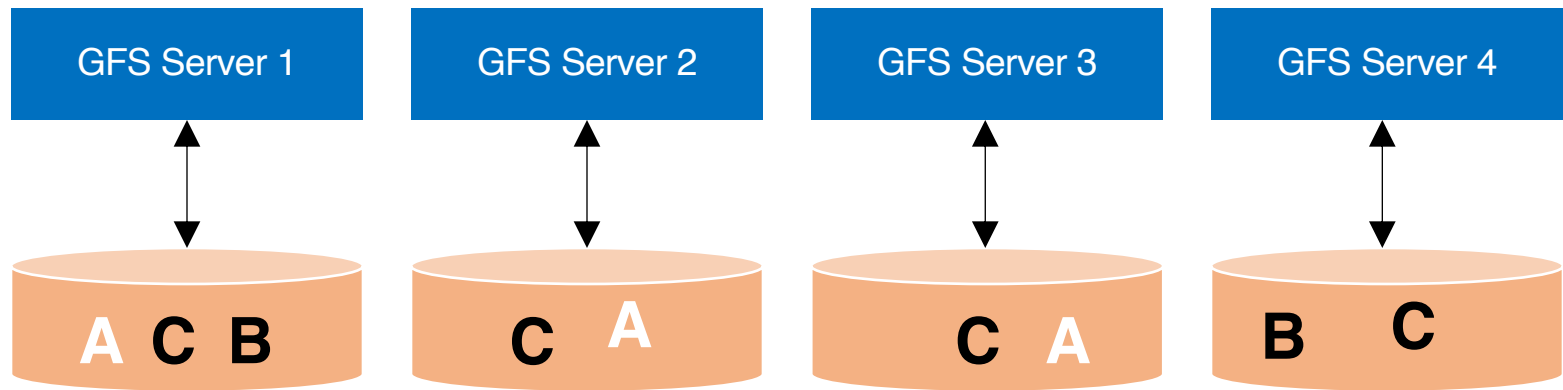


# Data recovery

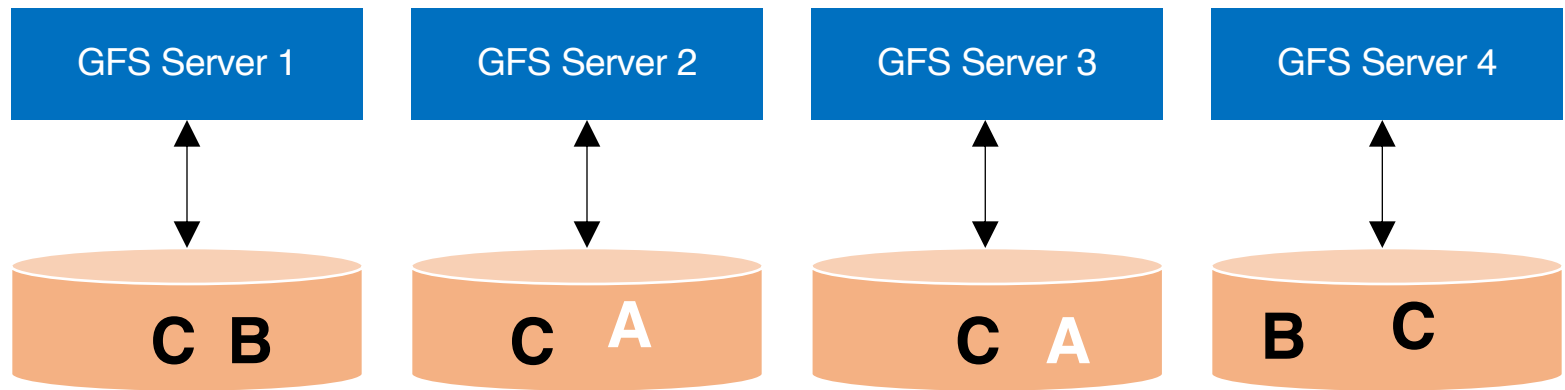


Machine may be dead forever, or it may come back

# Data recovery



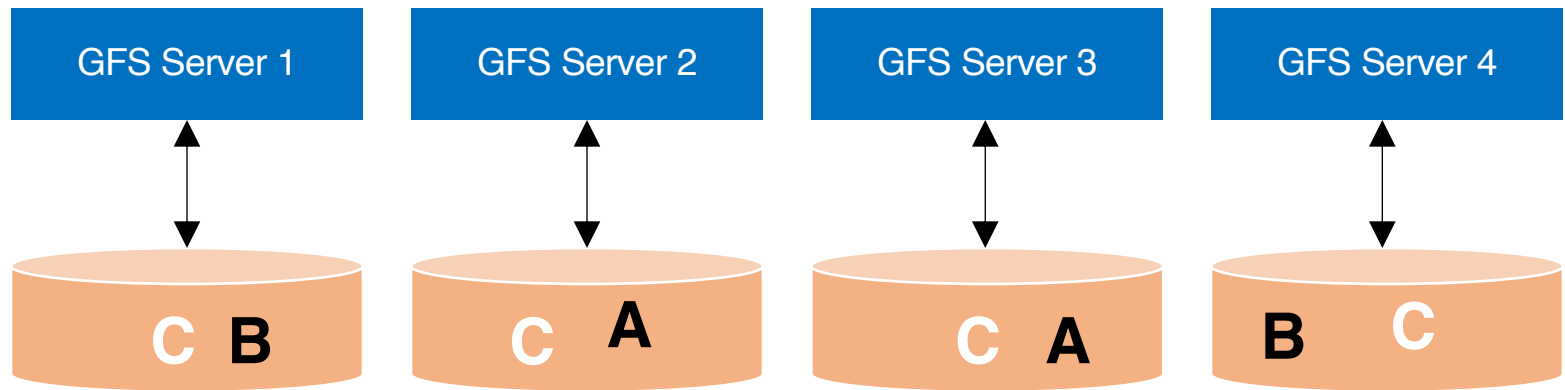
# Data recovery



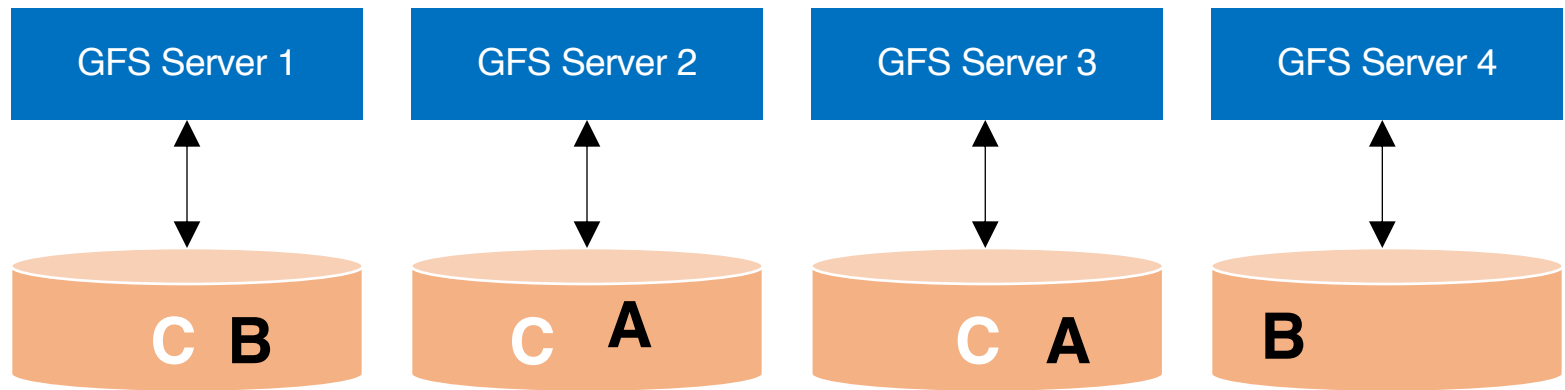
## Data Rebalancing

Deleting one A to maintain a replication factor of 2

# Data recovery



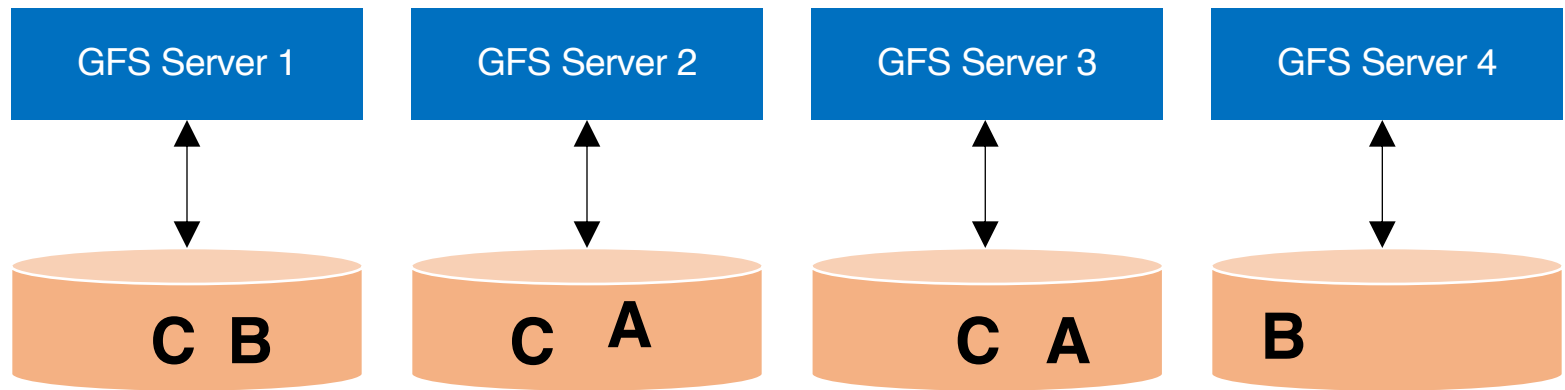
# Data recovery



## Data Rebalancing

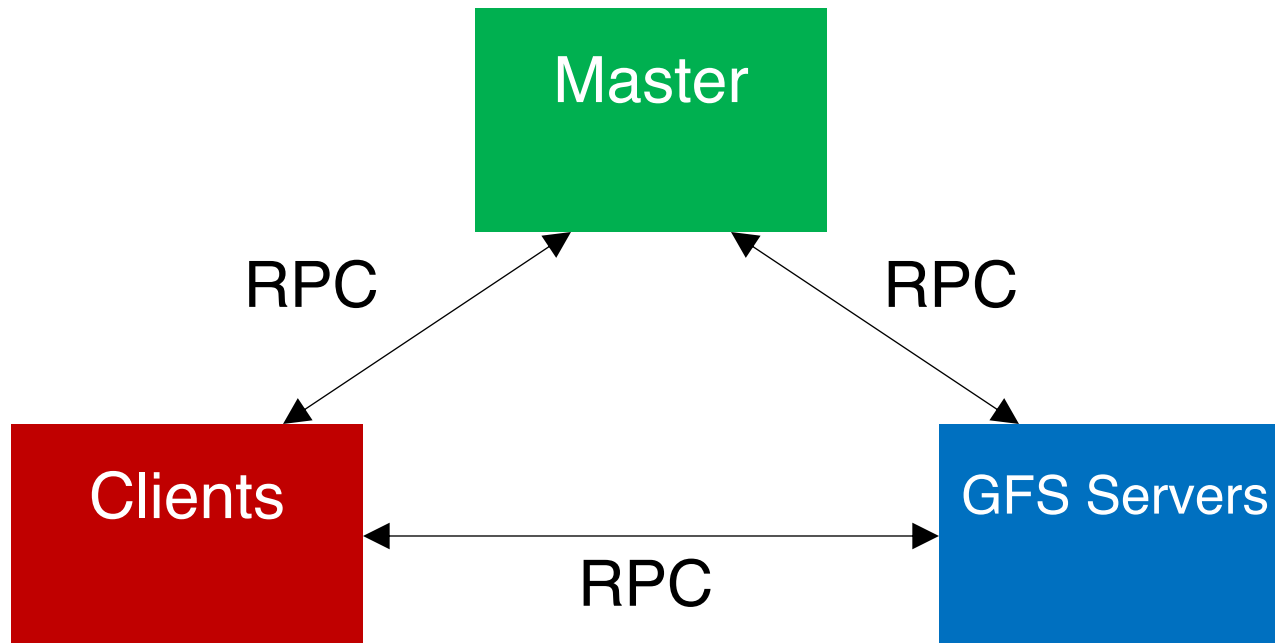
Deleting one C to maintain a replication factor of 3

# Data recovery

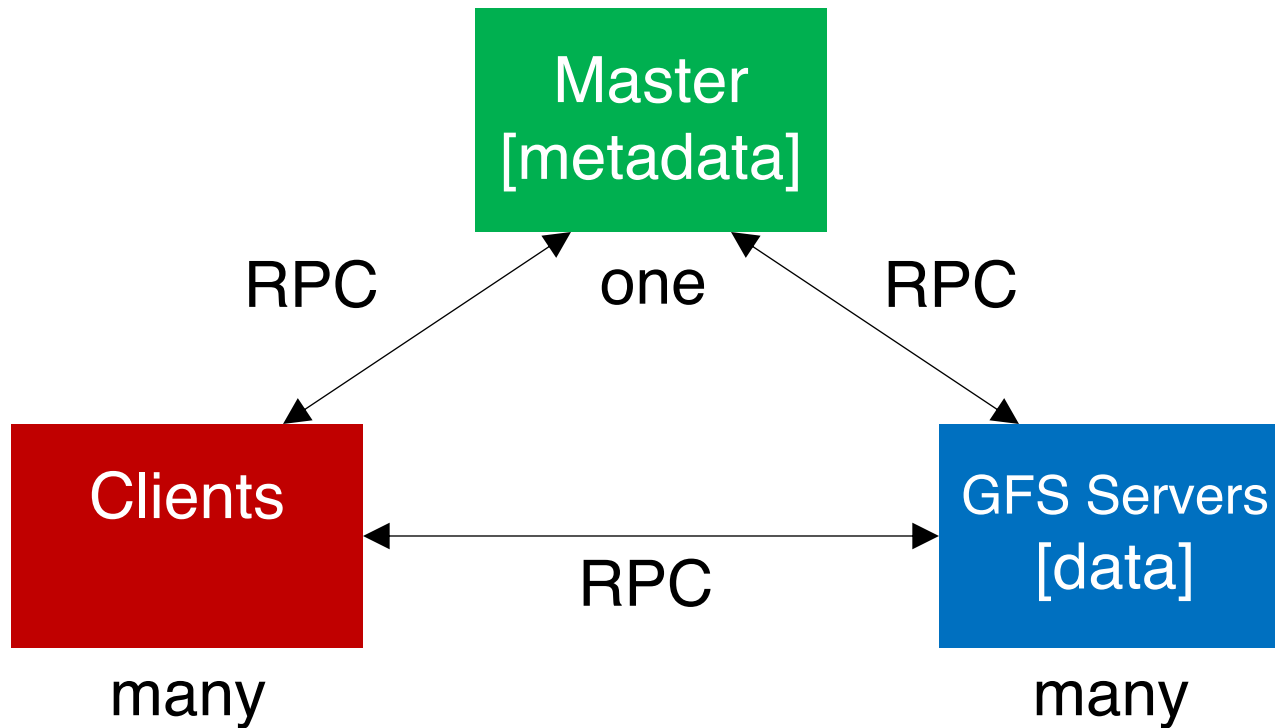


**Question:** how to maintain a global view of all data distributed across machines?

# GFS architecture

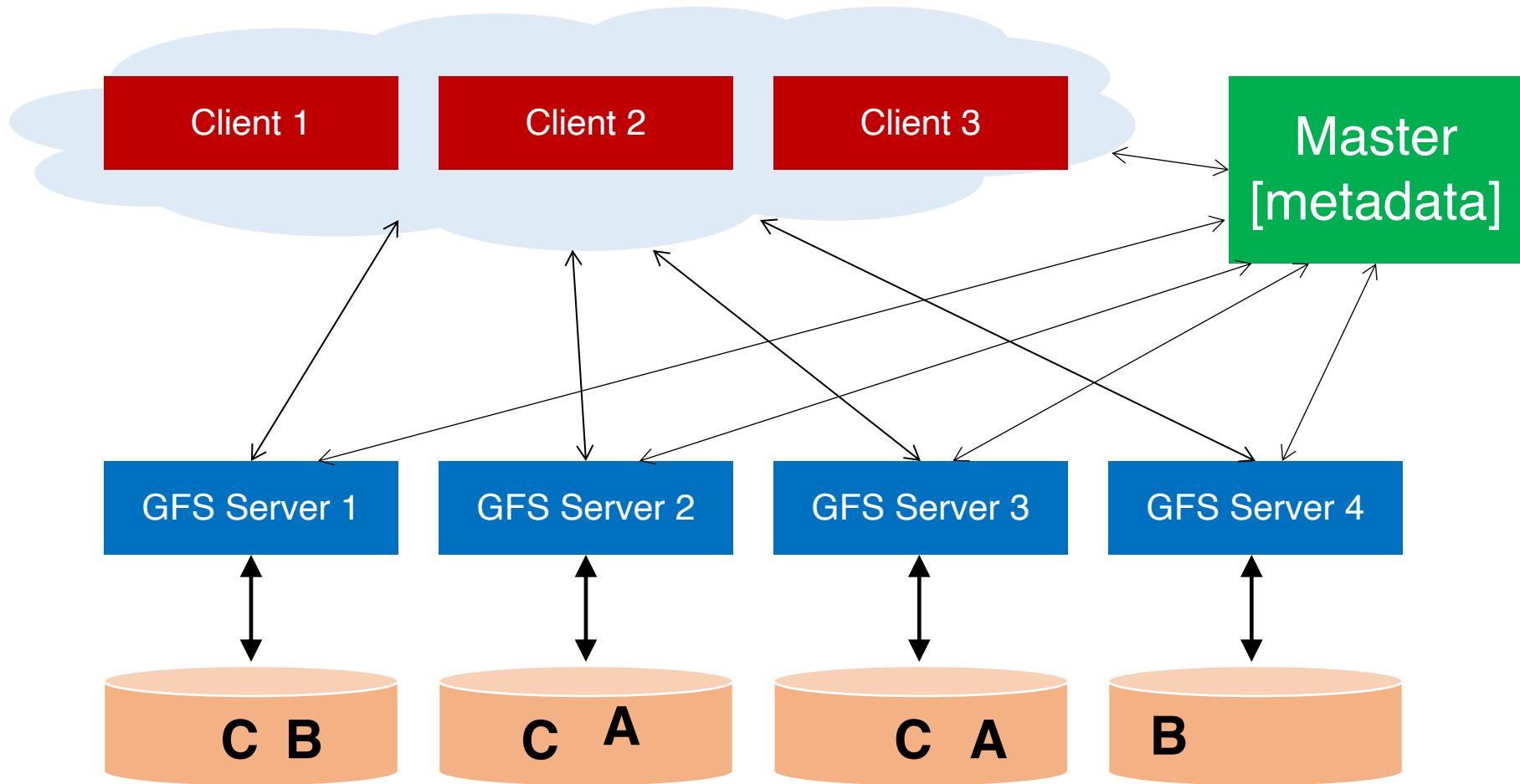


# GFS architecture





# GFS architecture



# Data chunks

- Break large GFS files into **coarse-grained** data chunks (e.g., 64MB)
- GFS servers store physical data chunks in **local Linux file system**
- **Centralized** master keeps track of mapping between logical and physical chunks

# Chunk map

## Master

chunk map

logical	phys
924	s2,s5,s7
521	s2,s9,s11
...	...

# GFS server s2

## Master

chunk map

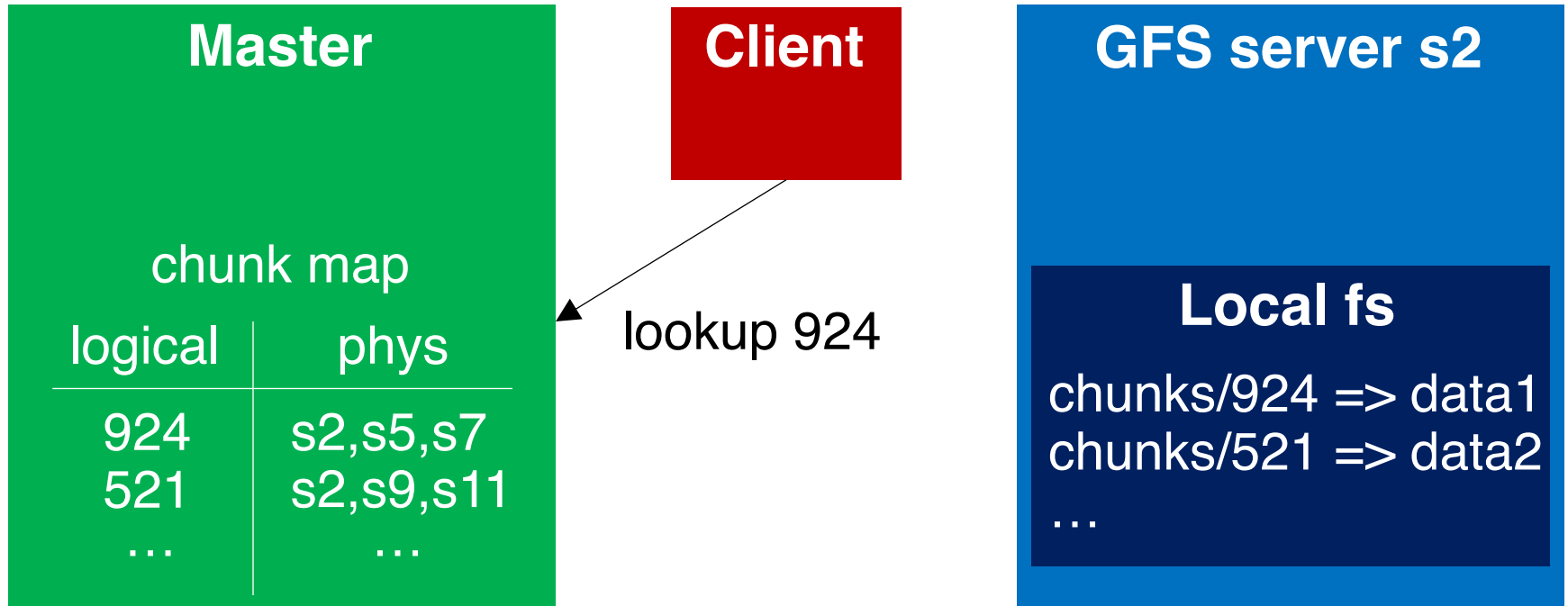
logical	phys
924	s2,s5,s7
521	s2,s9,s11
...	...

## GFS server s2

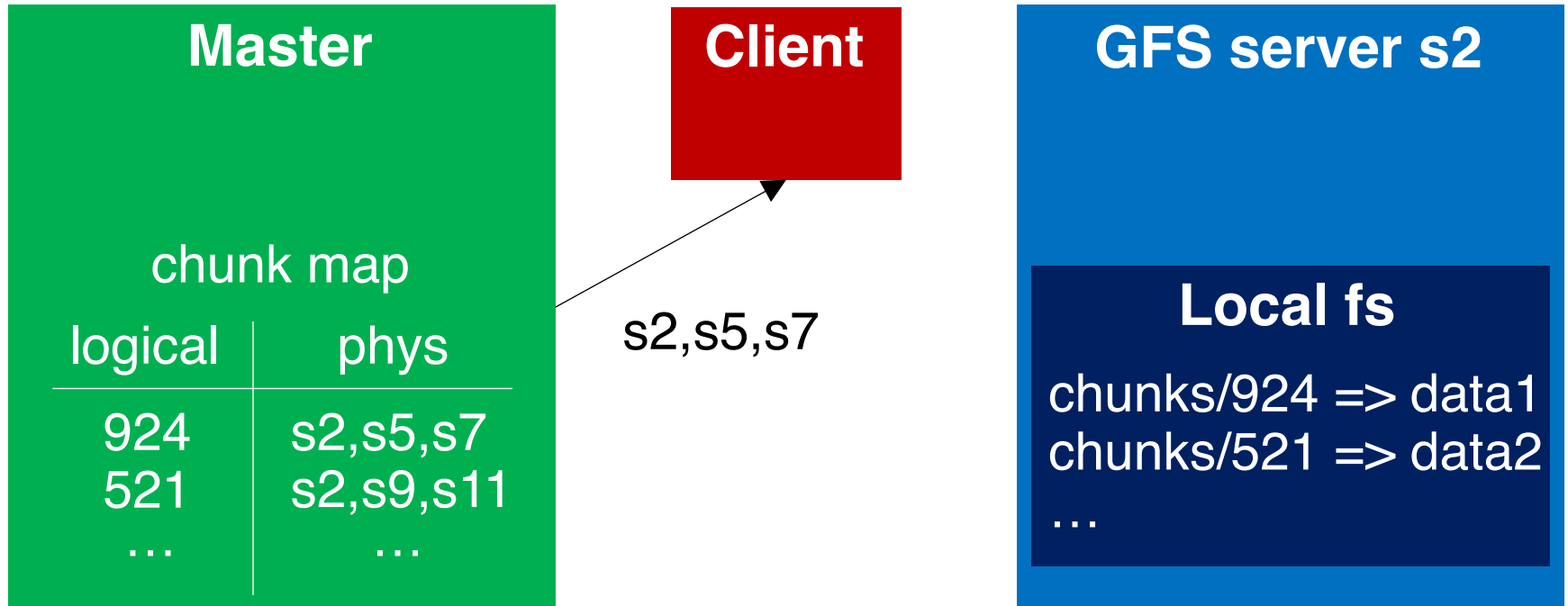
### Local fs

chunks/924 => data1  
chunks/521 => data2  
...

# Client reads a chunk



# Client reads a chunk



# Client reads a chunk

## Master

chunk map

logical	phys
924	s2,s5,s7
521	s2,s9,s11
...	...

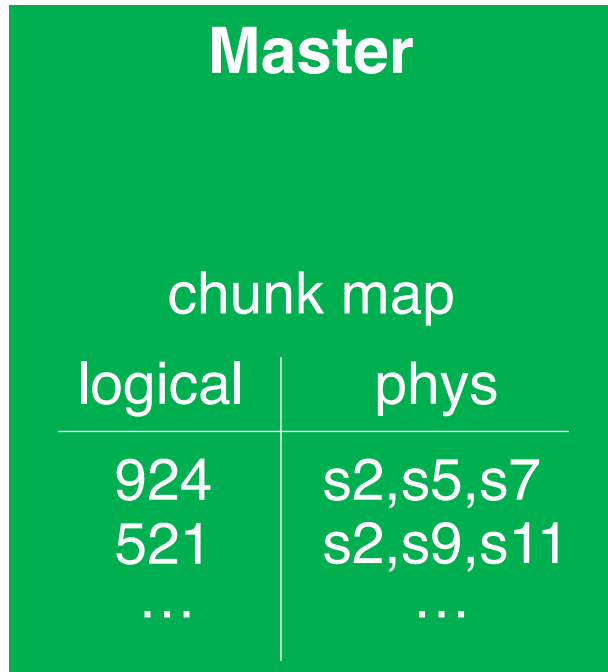
## Client

## GFS server s2

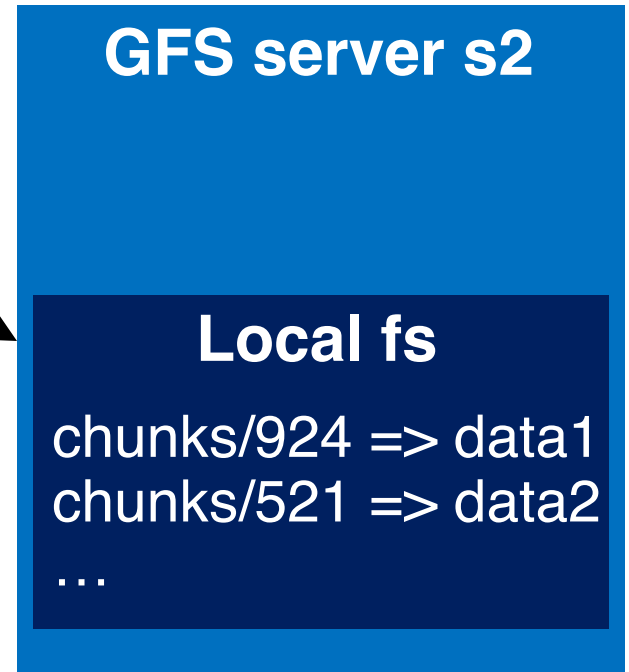
### Local fs

chunks/924 => data1  
chunks/521 => data2  
...

# Client reads a chunk

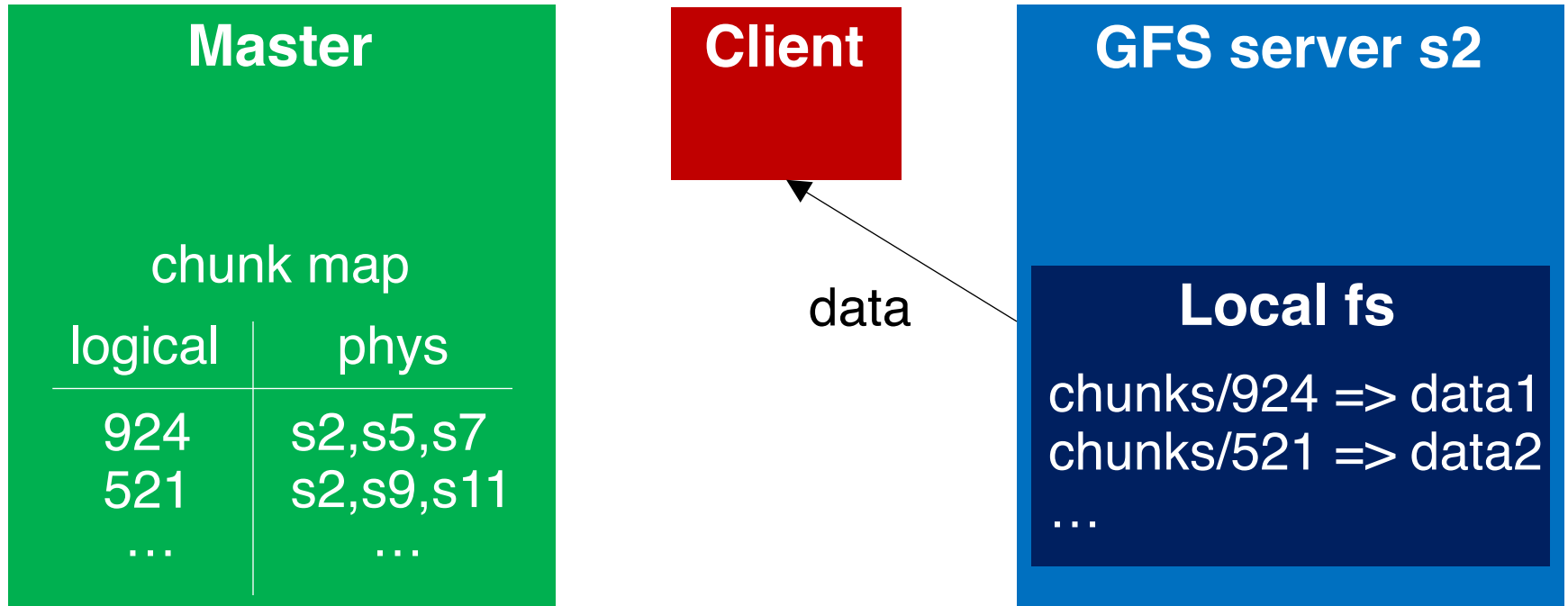


read 924:  
offset=0  
size=1MB

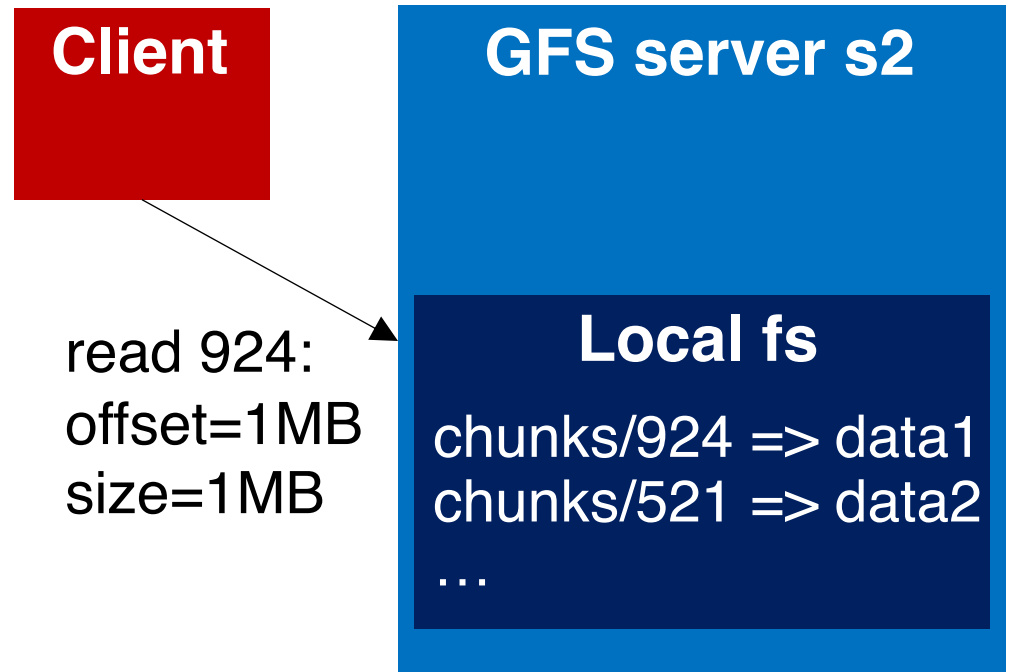
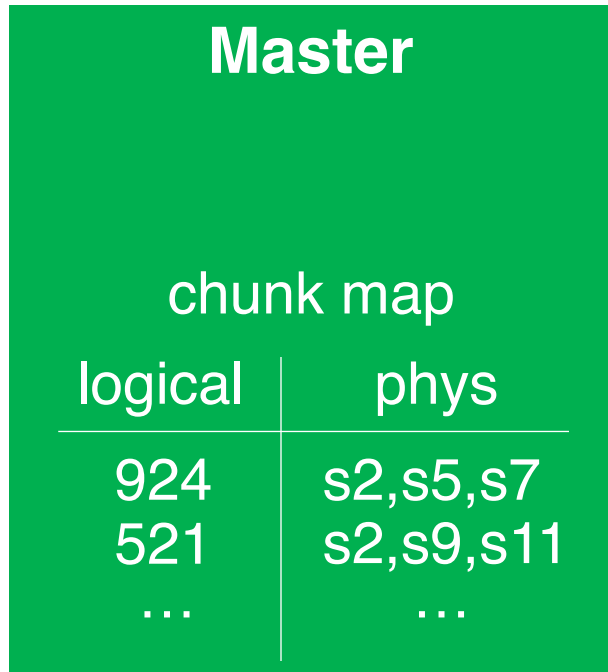




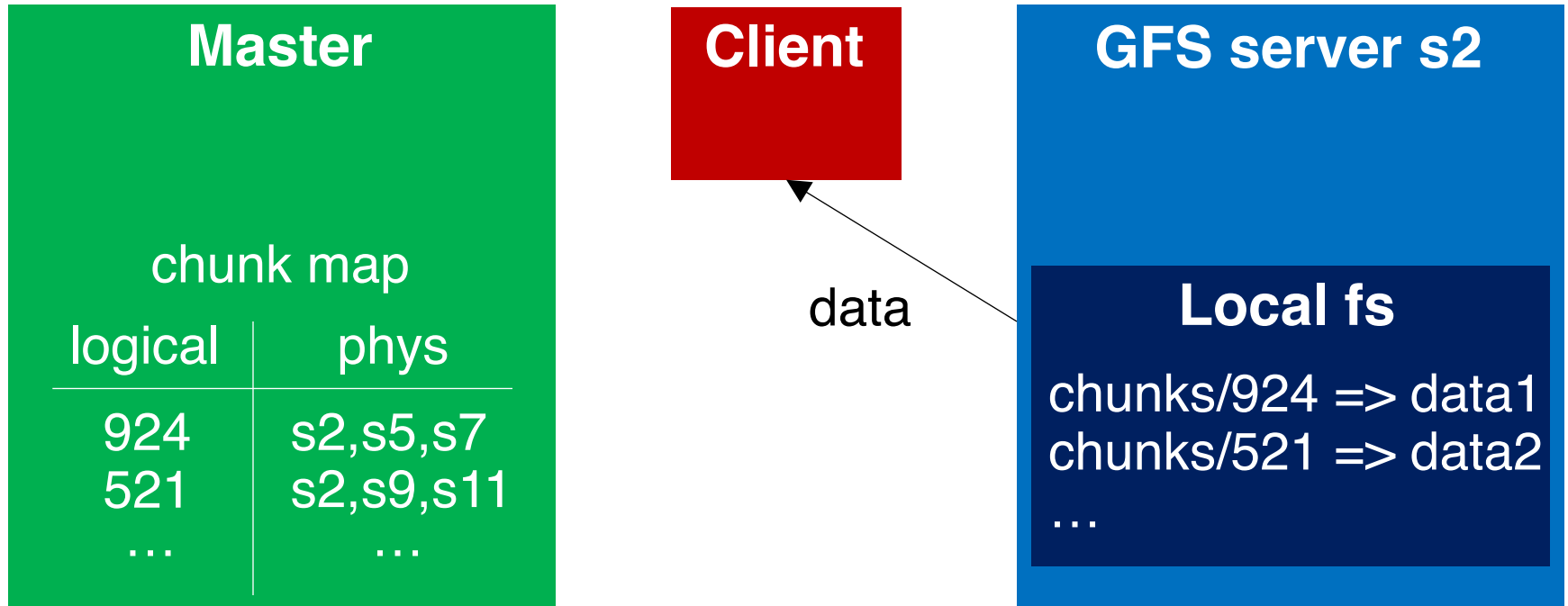
# Client reads a chunk



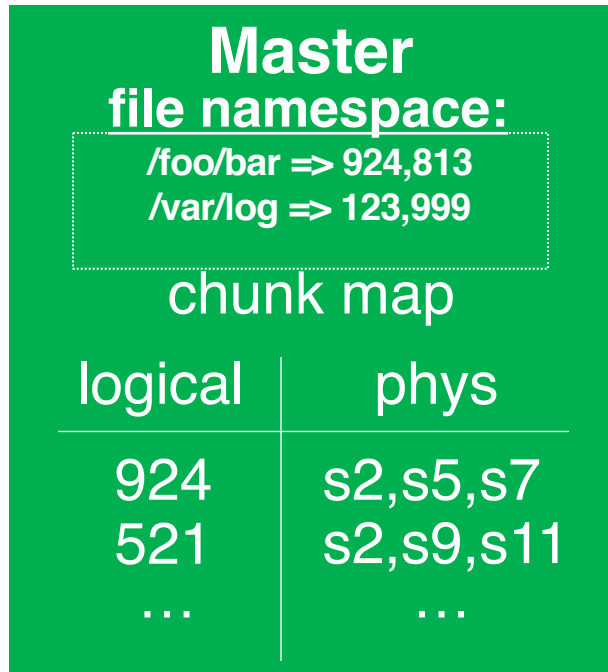
# Client reads a chunk



# Client reads a chunk



# File namespace



**Client**

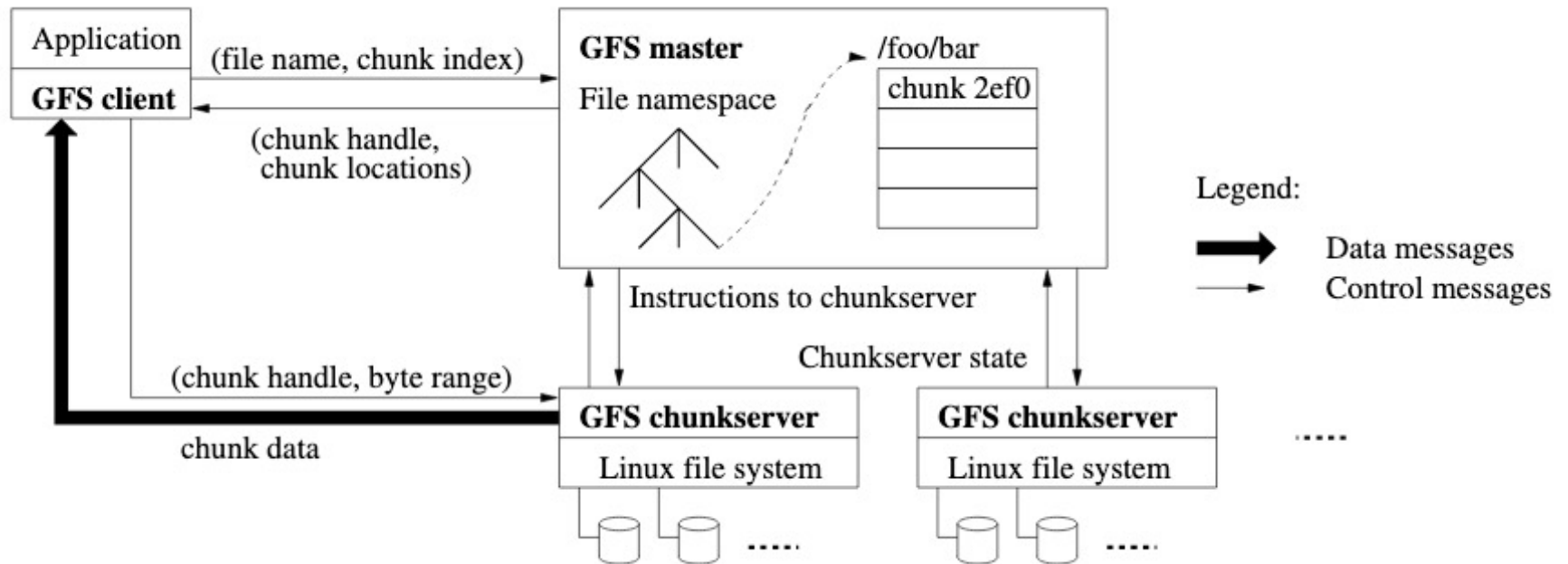
**GFS server s2**

**Local fs**

`chunks/924 => data1`  
`chunks/521 => data2`  
...

path names mapped to logical names

# GFS architecture (original paper)



# MapReduce+GFS: Put everything together

