Google File System

CS 4740: Cloud Computing

Fall 2024

Lecture 7

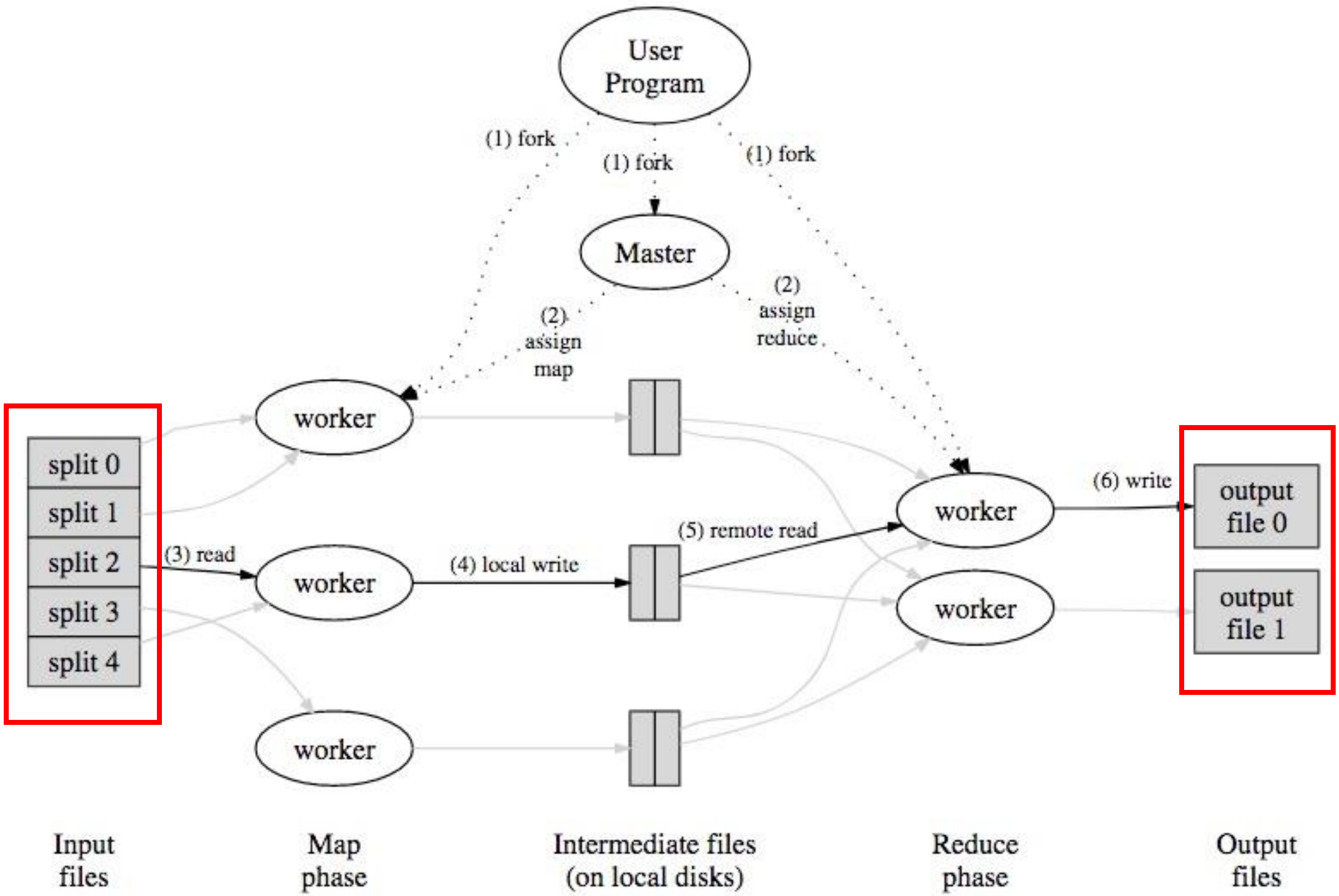
Yue Cheng

UNIVERSITY of VIRGINIA

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.

Recap: MapReduce



Recap: 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 ← Today

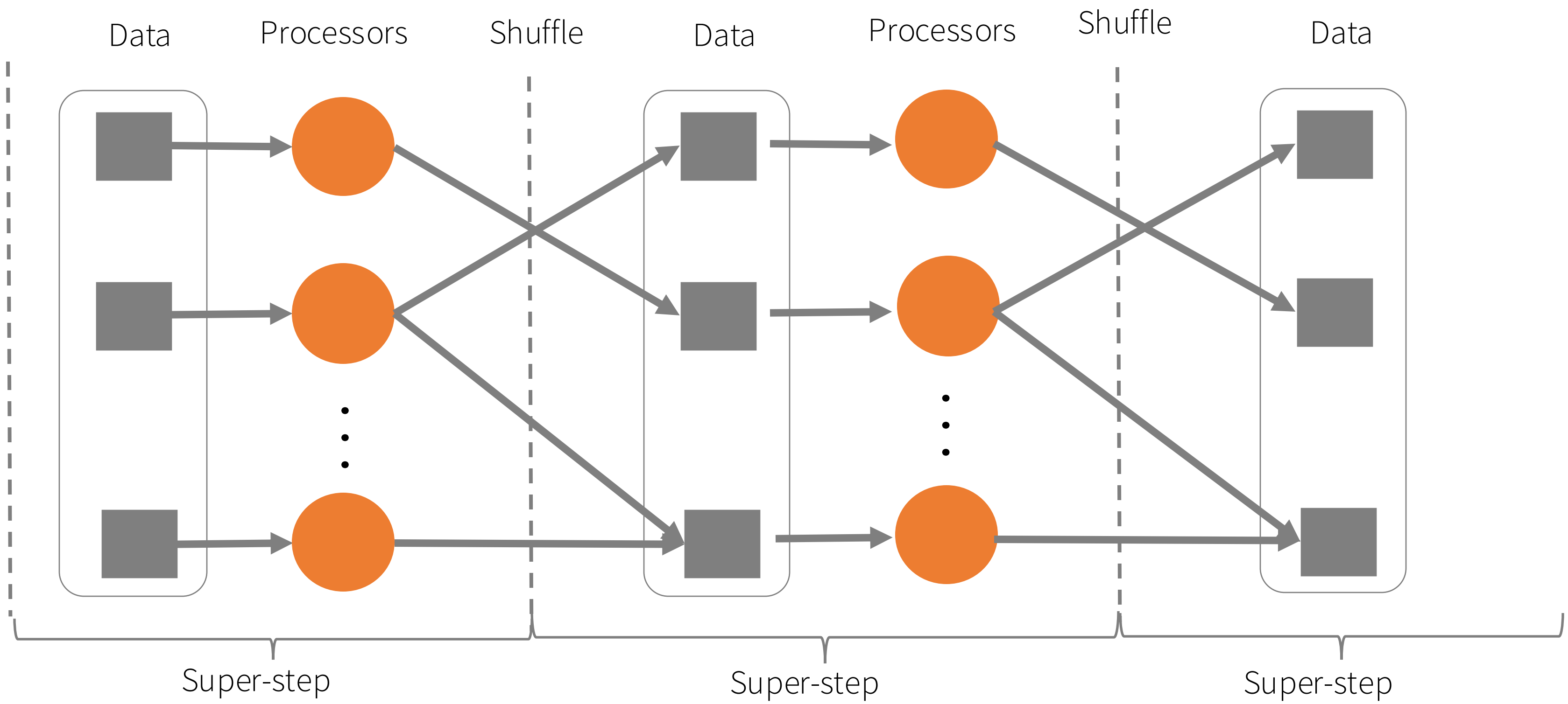
Google file system (GFS)

- 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

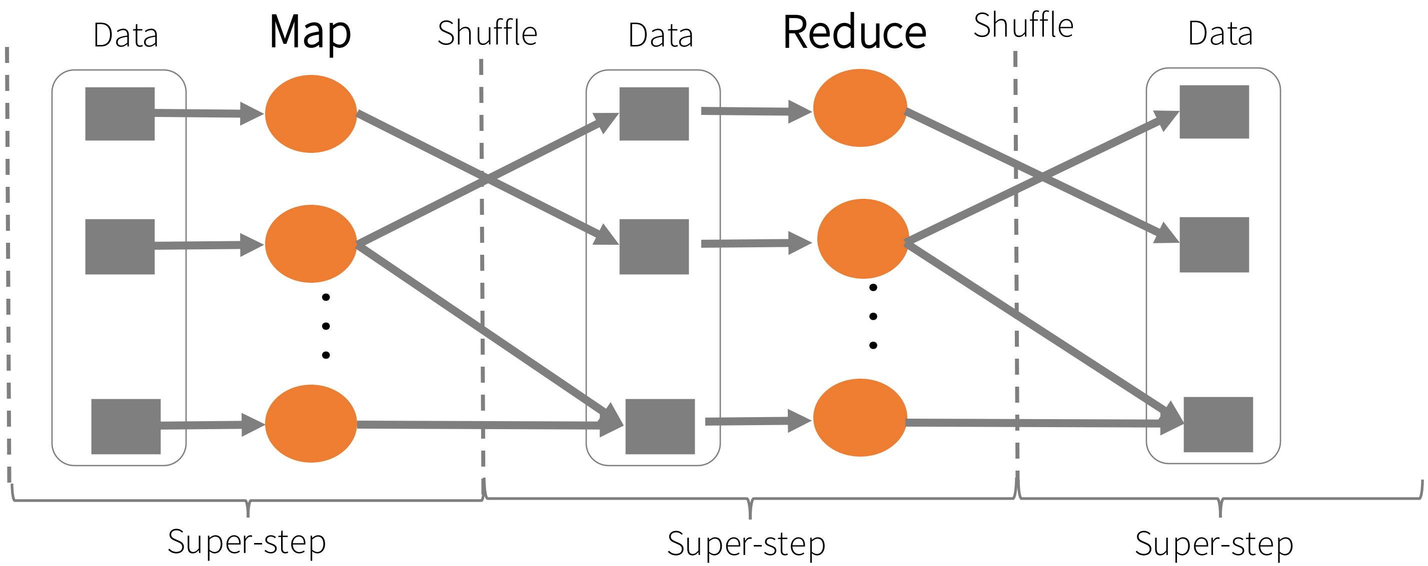
- MapReduce workload characteristics
  - Huge files (GBs)
  - Almost all writes are appends
  - Parallel appends common
  - High throughput is valuable
  - Low latency is not

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

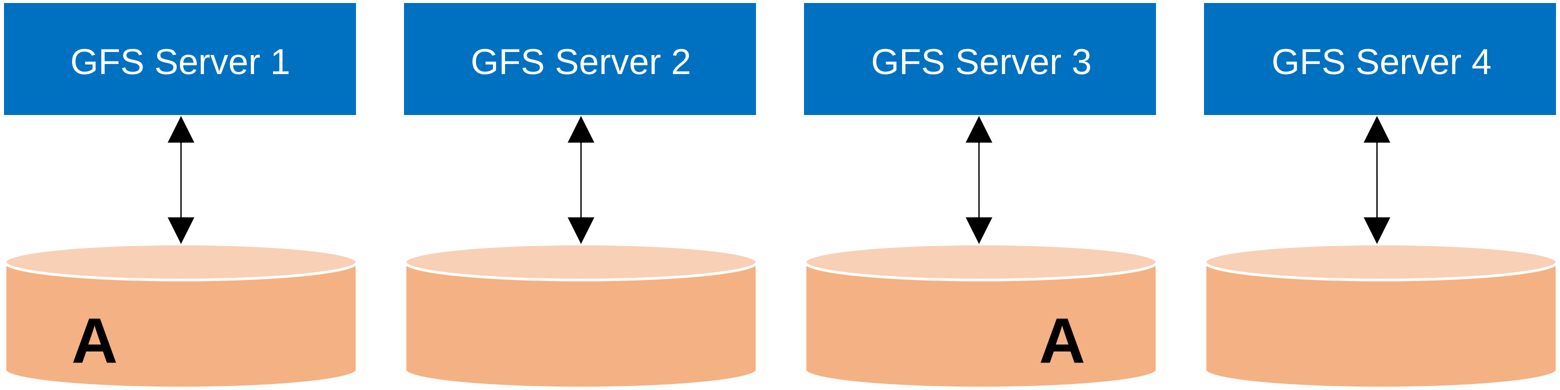
Workload-driven design

- Build a global (distributed) file system that incorporates all these application properties

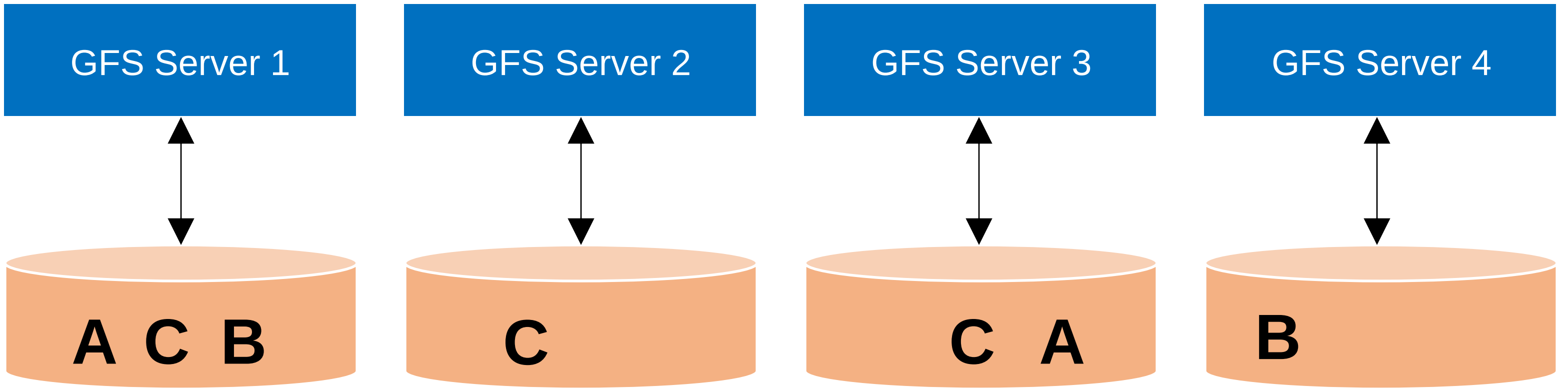
- Only supports features required by applications

- Avoid difficult local file system (POSIX) features, e.g.:
  - atomic rename
  - (symbolic or hard) links
  - limited file permissions and ownership
  - …

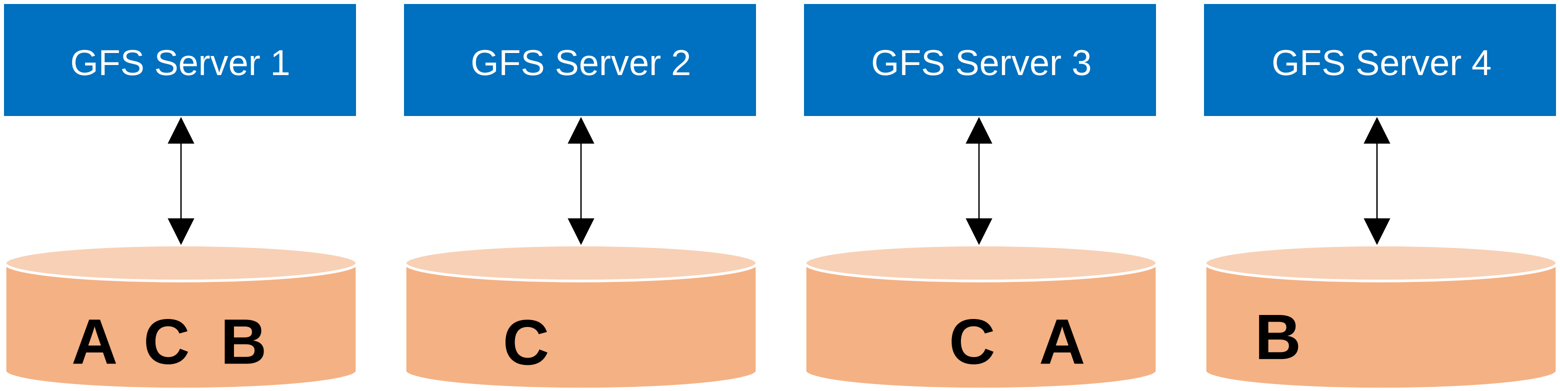
Replication



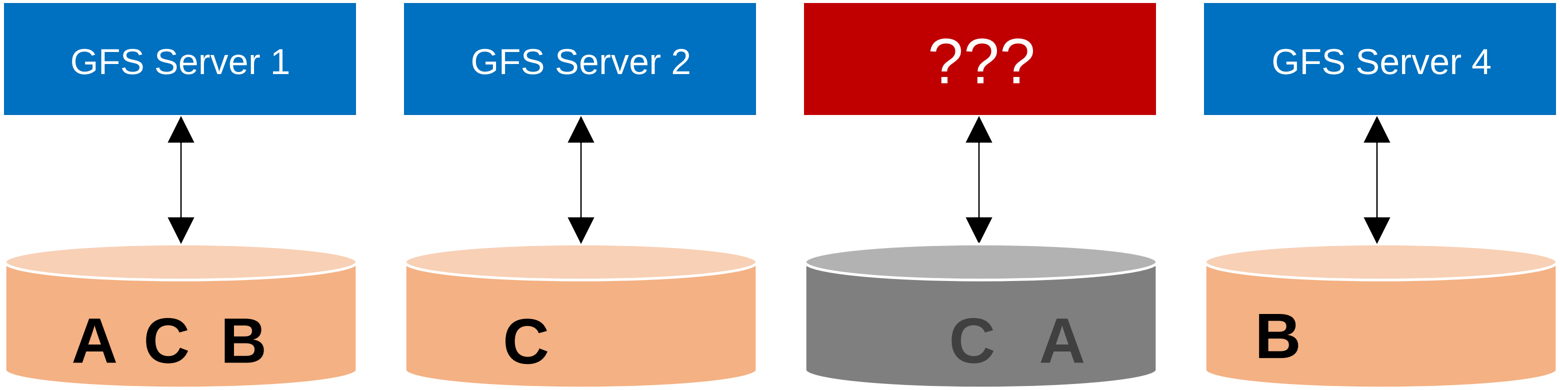
Replication



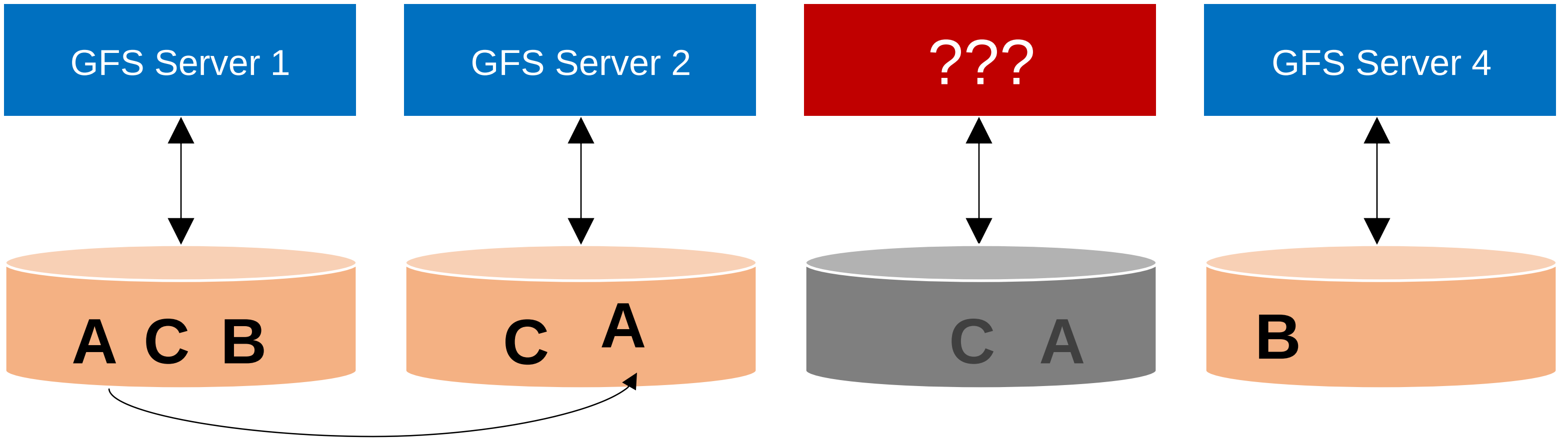
Resilience against failures



Resilience against failures

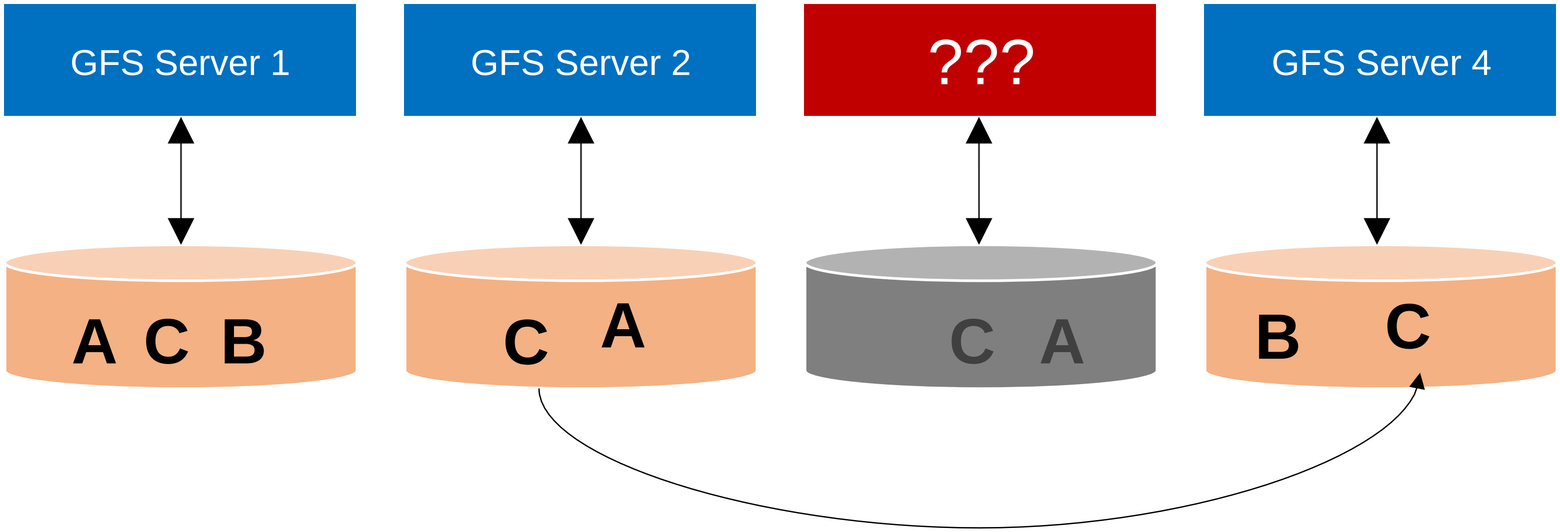


Data recovery



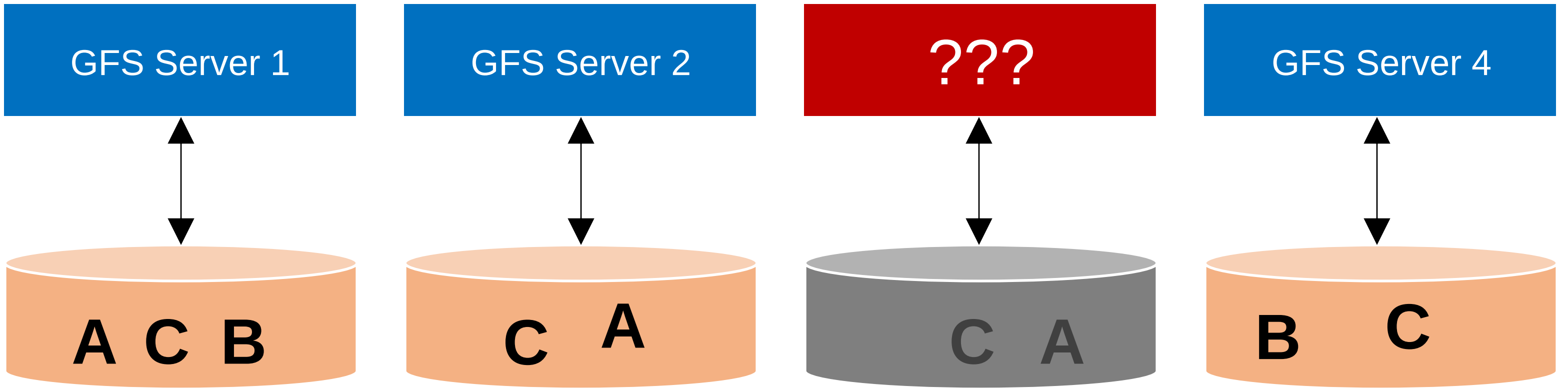
Replicating A to maintain a replication factor of 2

Data recovery



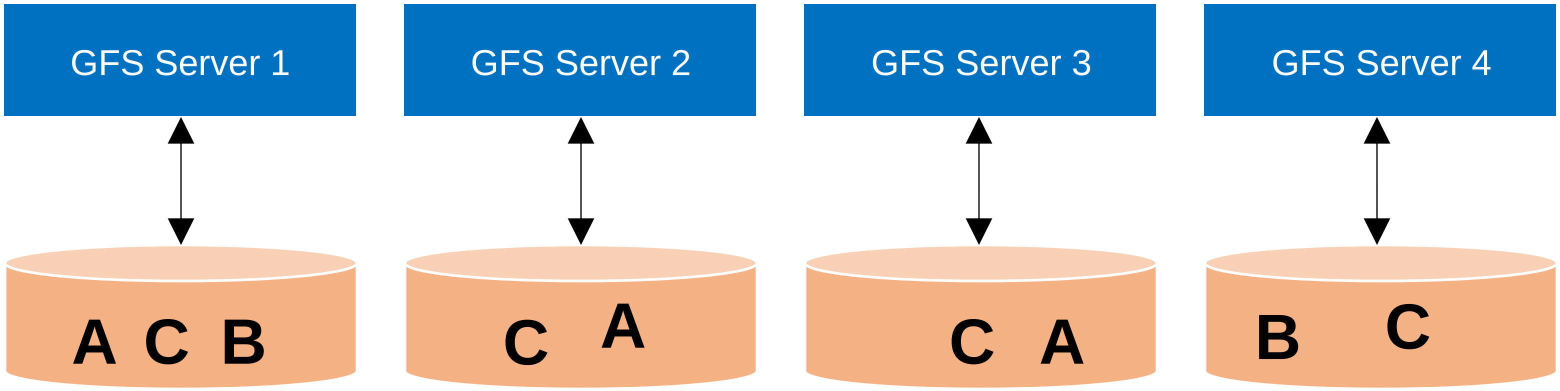
Replicating C to maintain a replication factor of 3

Data recovery



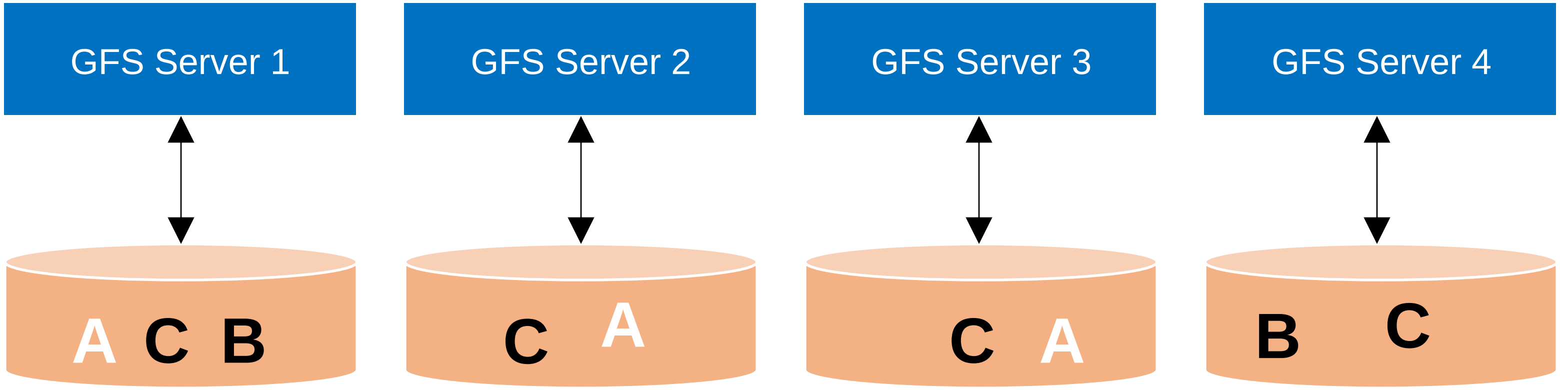
Machine may be dead forever, or it may come back

Data recovery

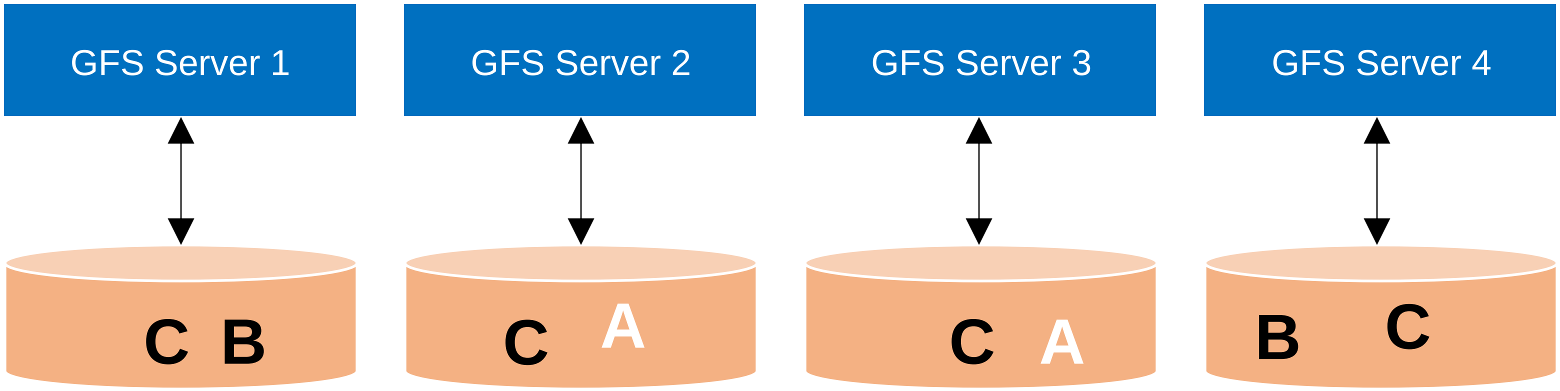


Machine may be dead forever, or it may come back

Data recovery



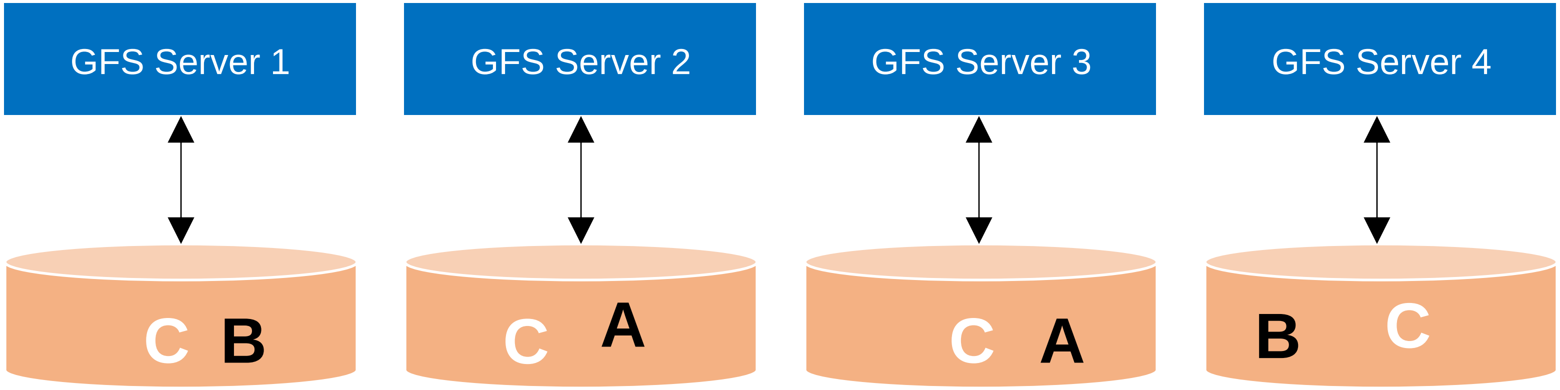
Data recovery



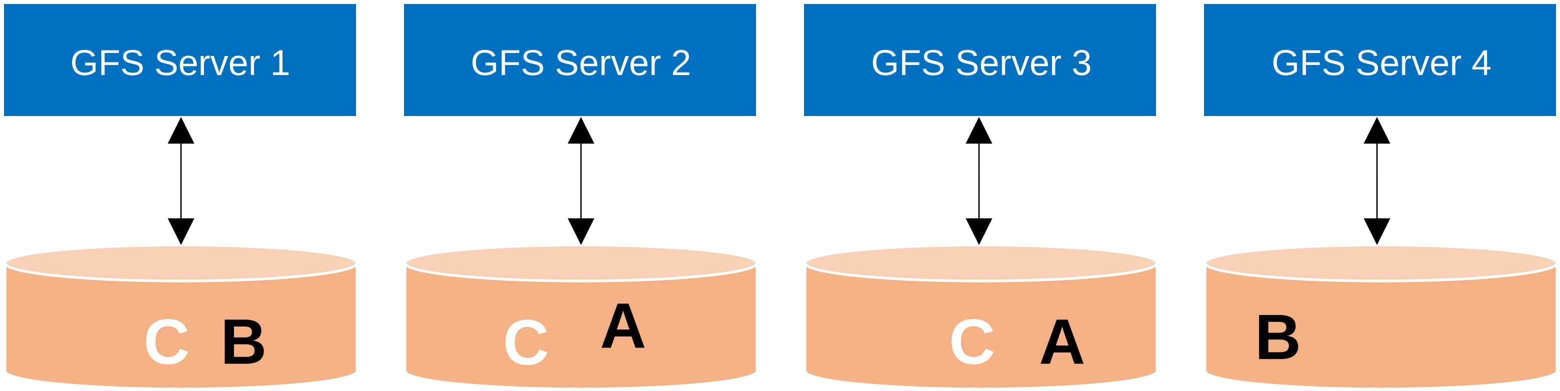
Data Rebalancing

Deleting one A to maintain a replication factor of 2

Data recovery



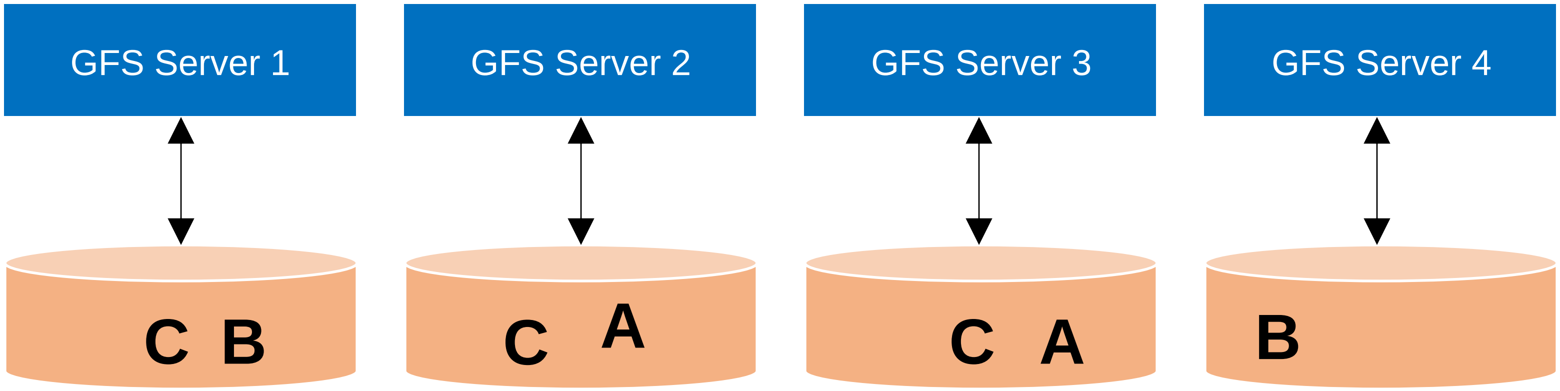
Data recovery



Data Rebalancing

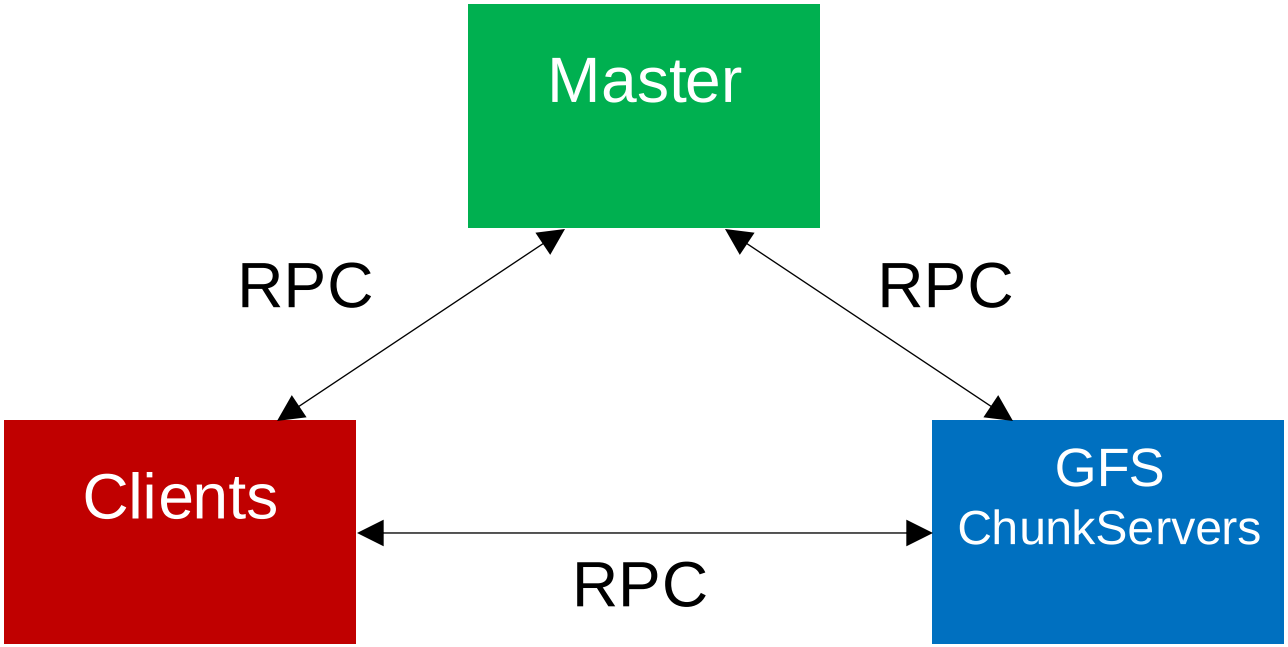
Deleting one C to maintain a replication factor of 3

Data recovery

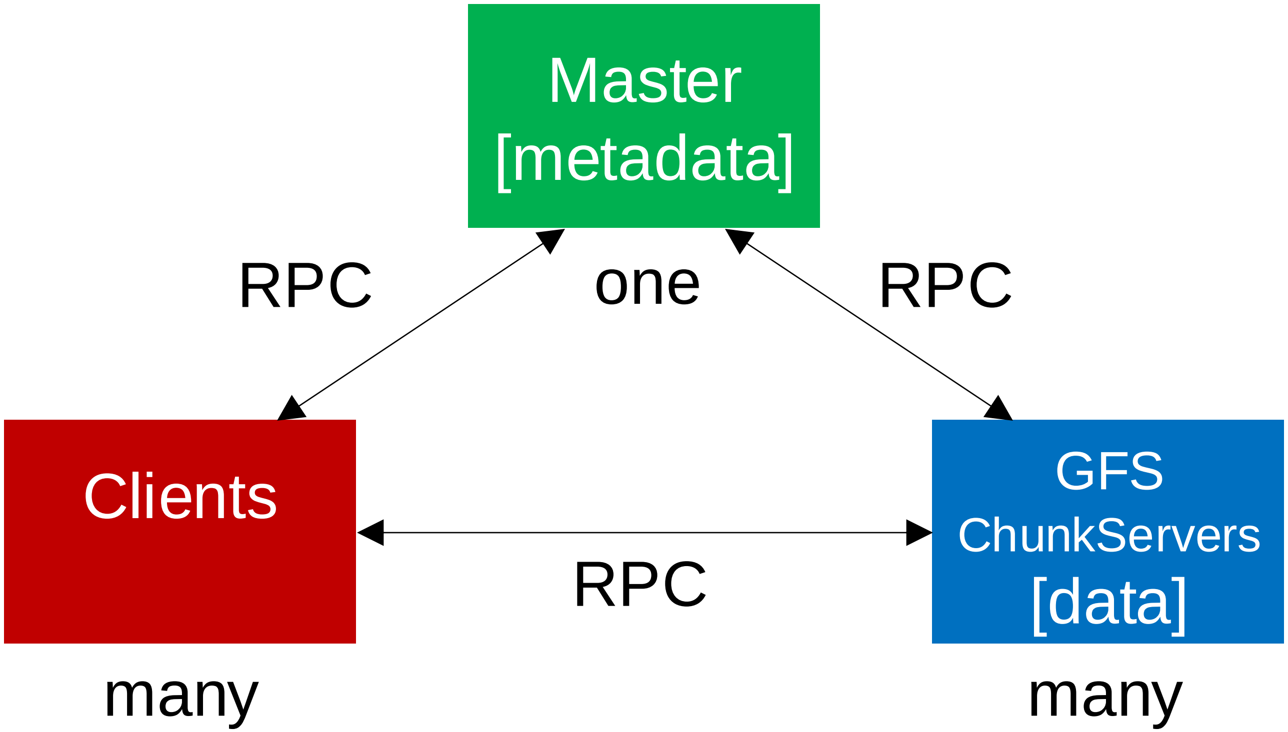


Question: how to maintain a global view of all data distributed across machines?

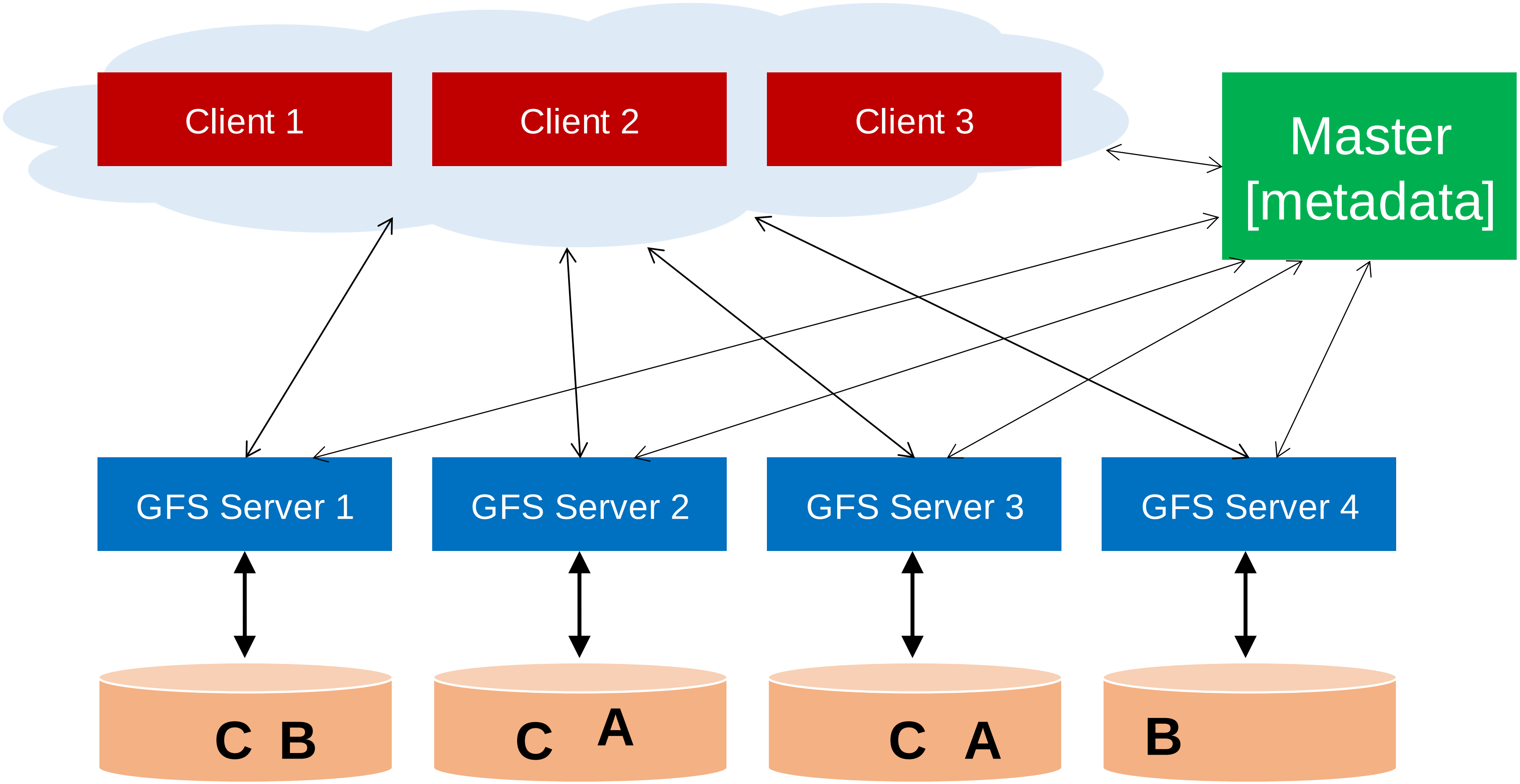
GFS architecture



GFS architecture



GFS architecture



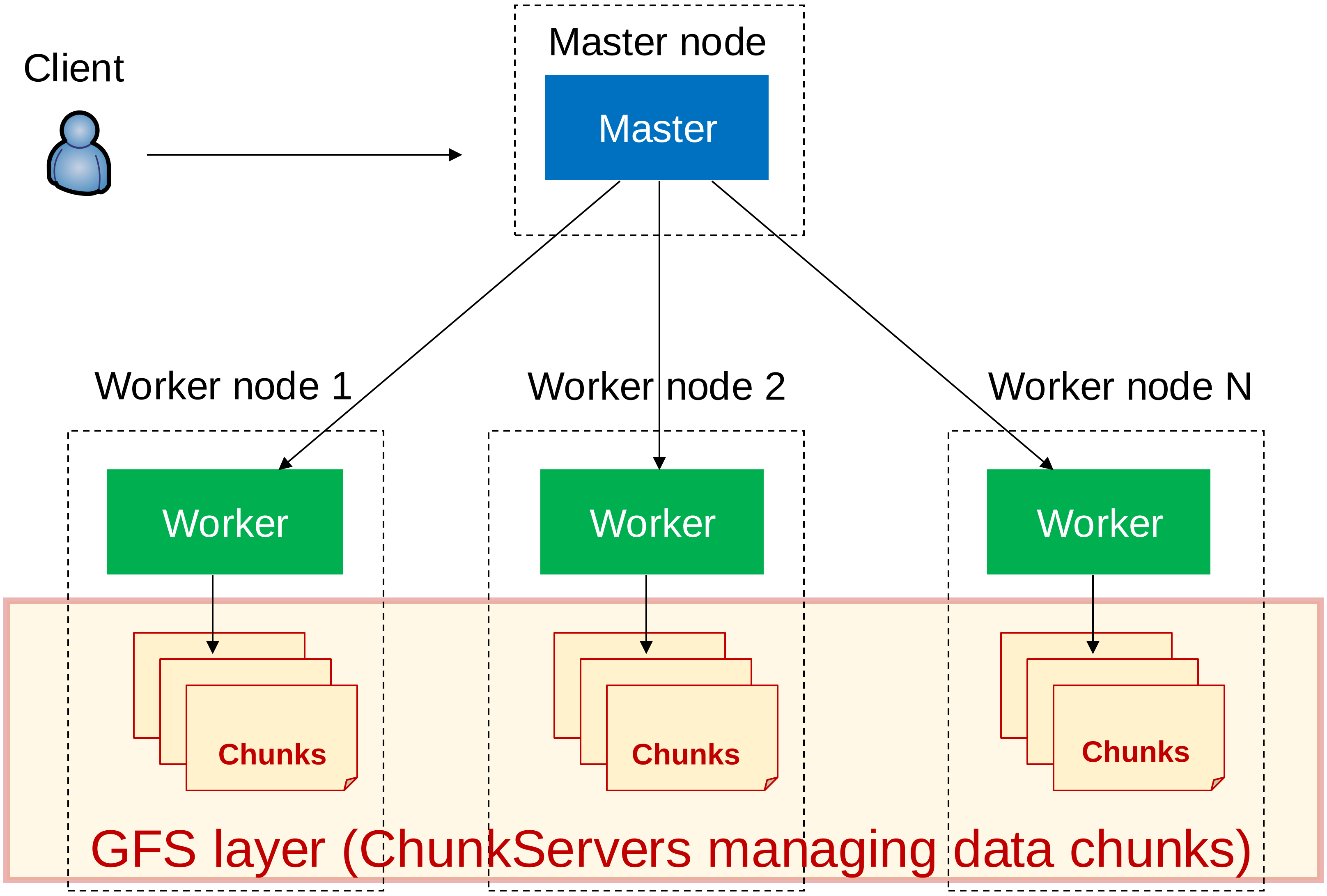
File chunks

- Break large GFS files into coarse-grained data chunks (e.g., 64MB)

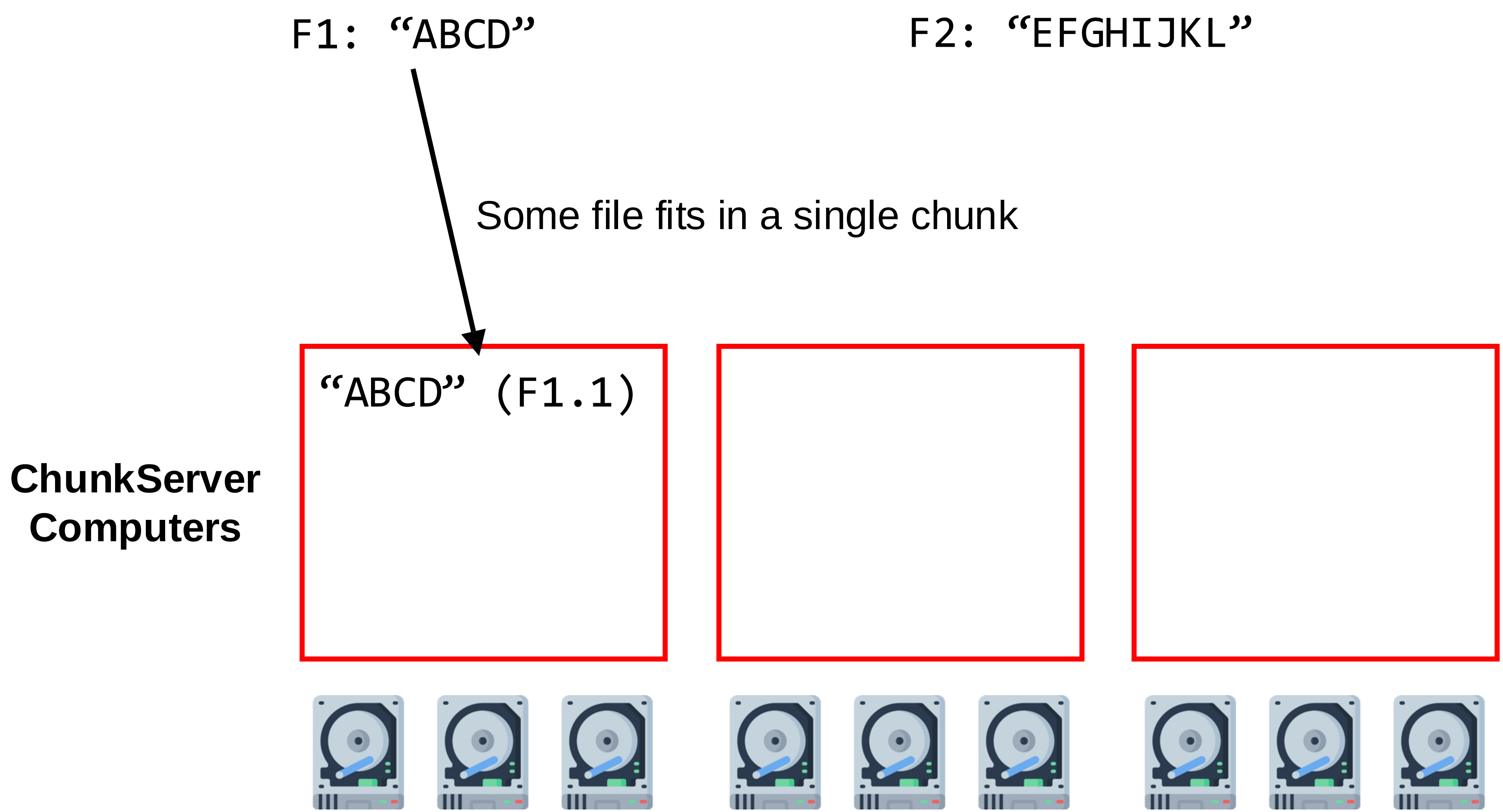
- GFS servers store physical data chunks in local Linux file system

- Centralized master keeps track of mapping between logical and physical chunks

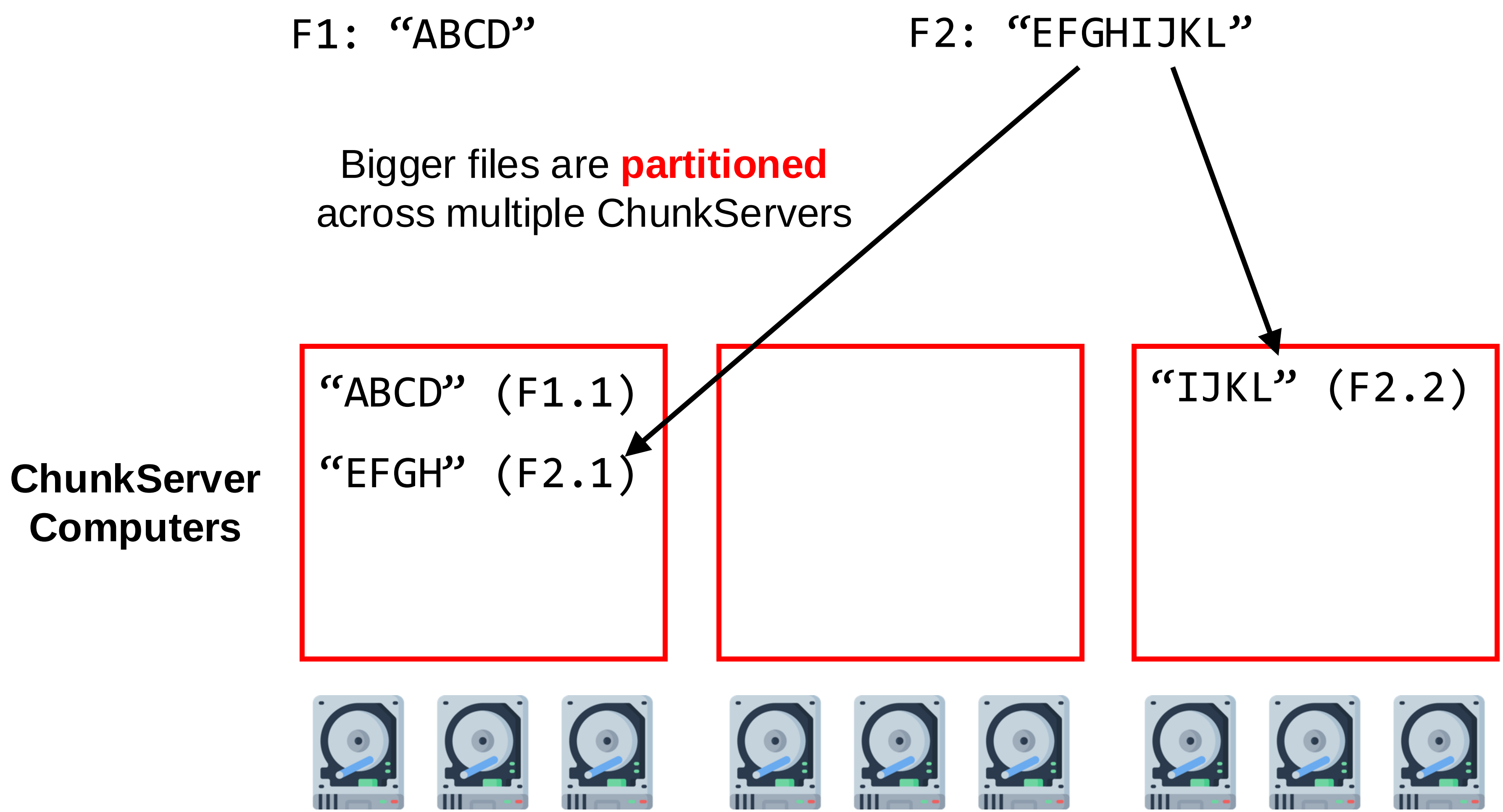
MapReduce+GFS: Layered design



GFS: ChunkServers store file chunks



GFS: Partitioning across ChunkServers



GFS: Replication across ChunkServers

F1: “ABCD”

3x replication

F2: “EFGHIJKL”

2x replication

ChunkServer Computers

“ABCD” (F1.1)
“EFGH” (F2.1)

“ABCD” (F1.1)
“IJKL” (F2.2)

“IJKL” (F2.2)
“ABCD” (F1.1)
“EFGH” (F2.1)

GFS: Replication across ChunkServers

F1: “ABCD”	F2: “EFGHIJKL”
3x replication	2x replication

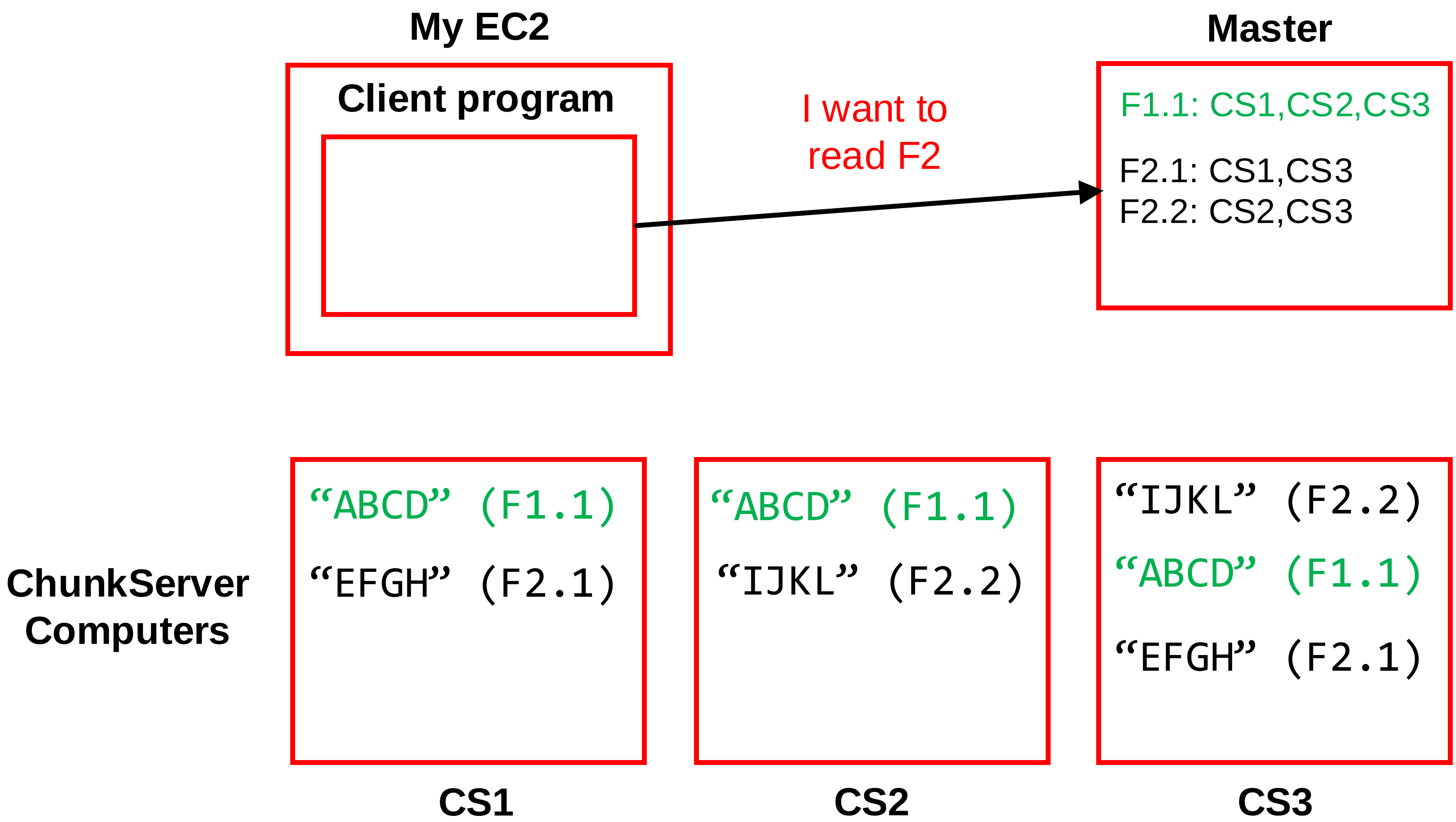
If a ChunkServer dies, we still have all the data.

Which file is safer in general? F1 or F2?

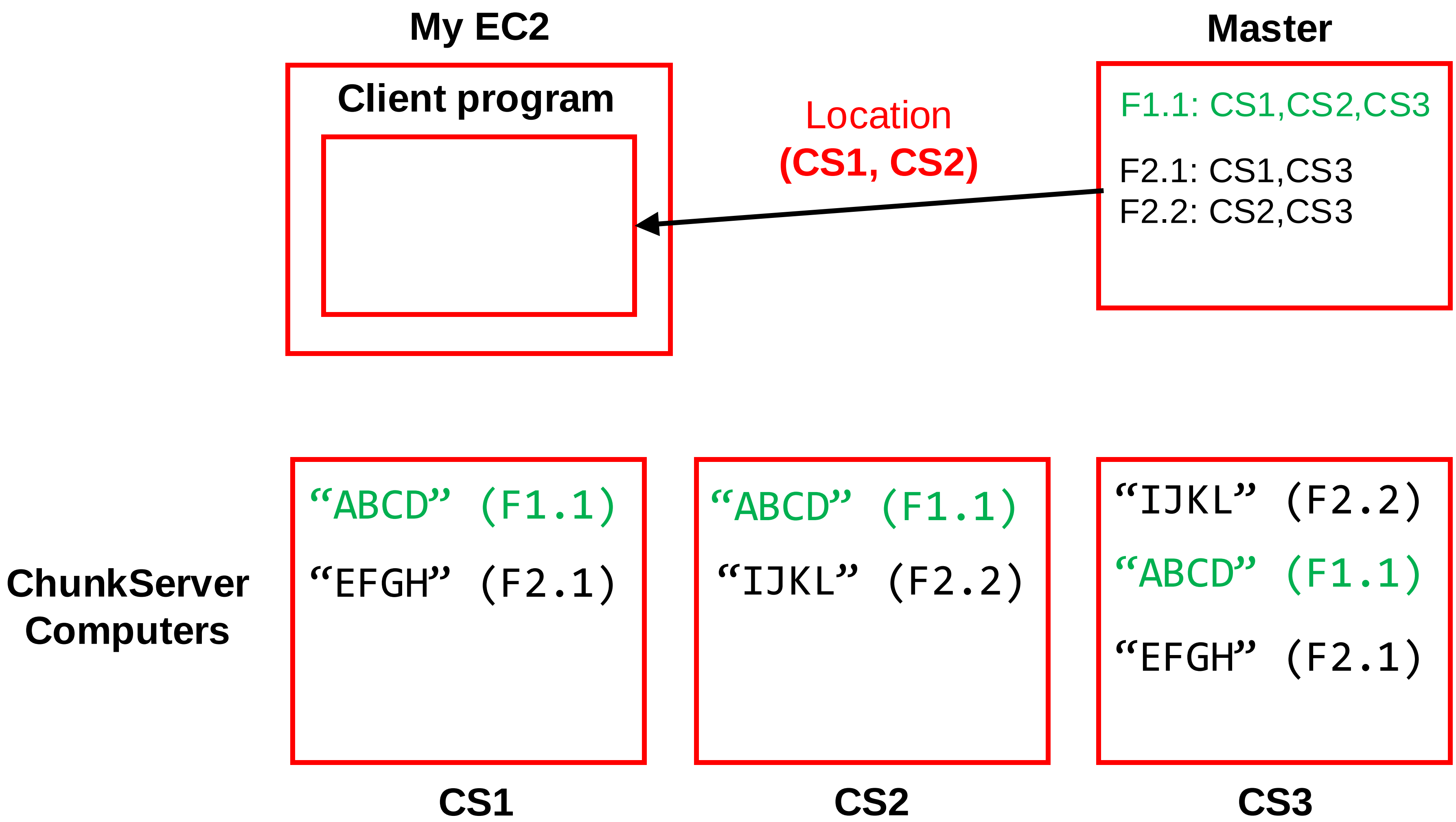
ChunkServer Computers

“ABCD” (F1.1)	“ABCD” (F1.1)	“IJKL” (F2.2)
“EFGH” (F2.1)	“IJKL” (F2.2)	“ABCD” (F1.1)
		“EFGH” (F2.1)

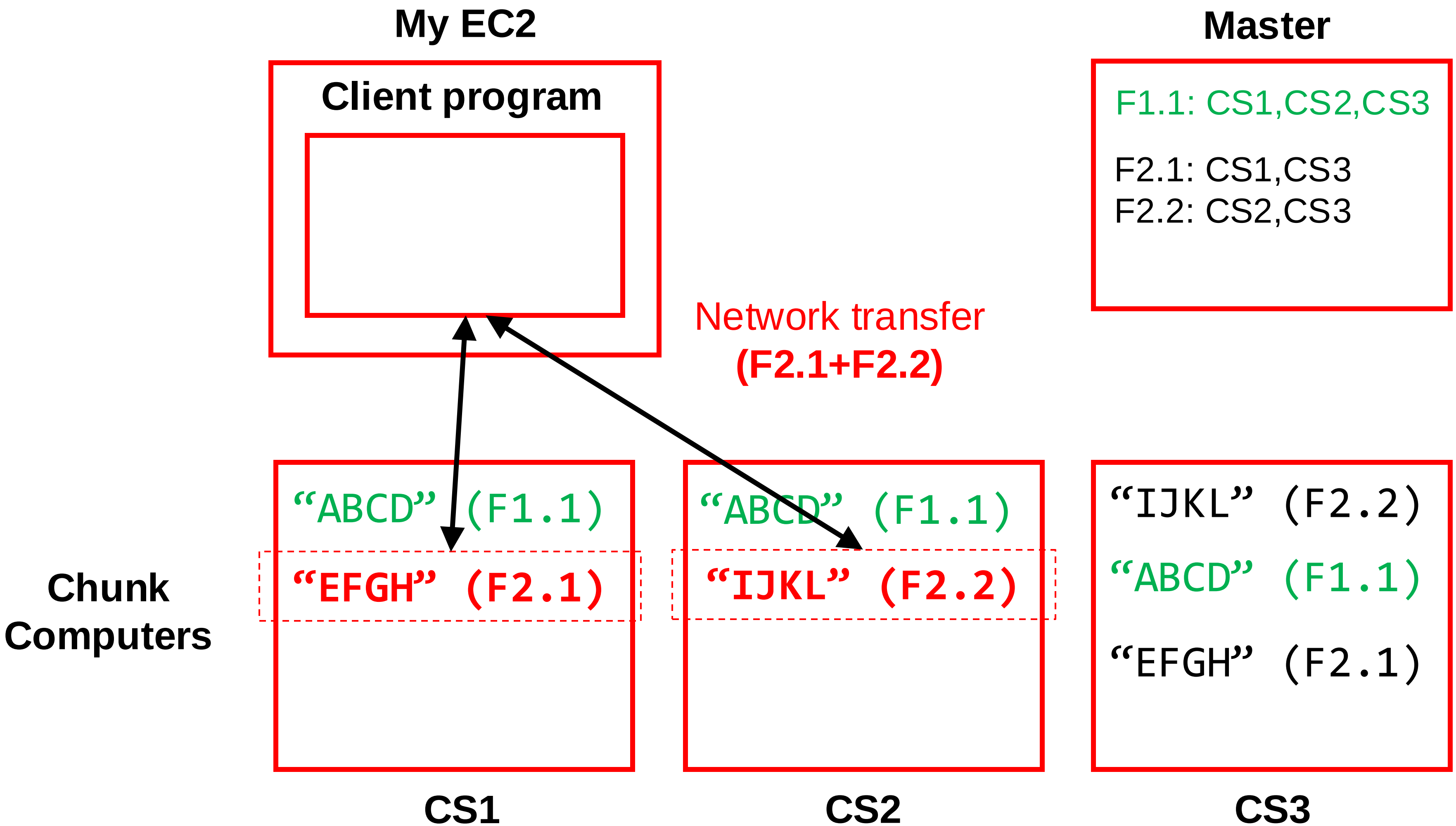
Master/Worker architecture



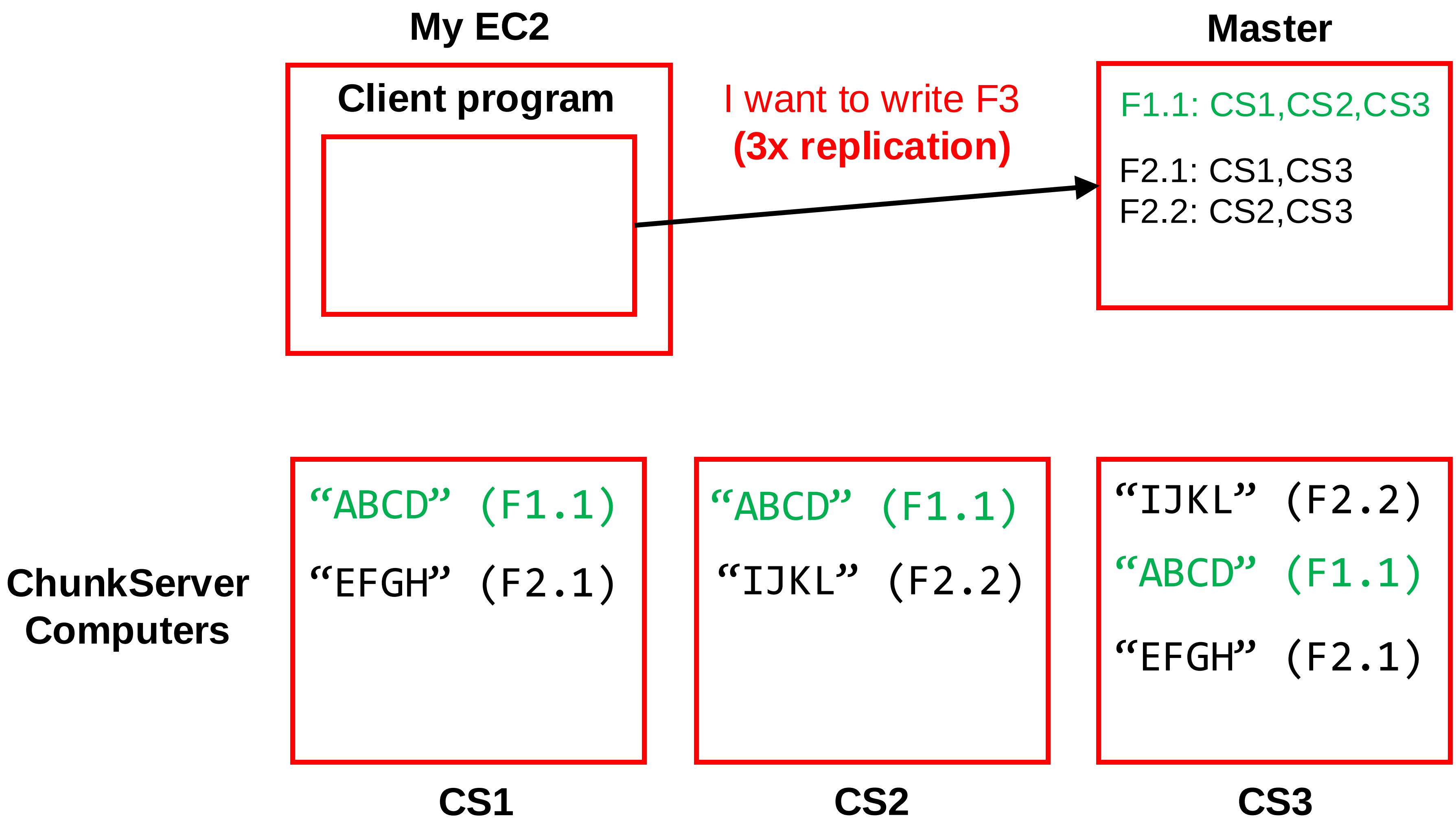
Master/Worker architecture



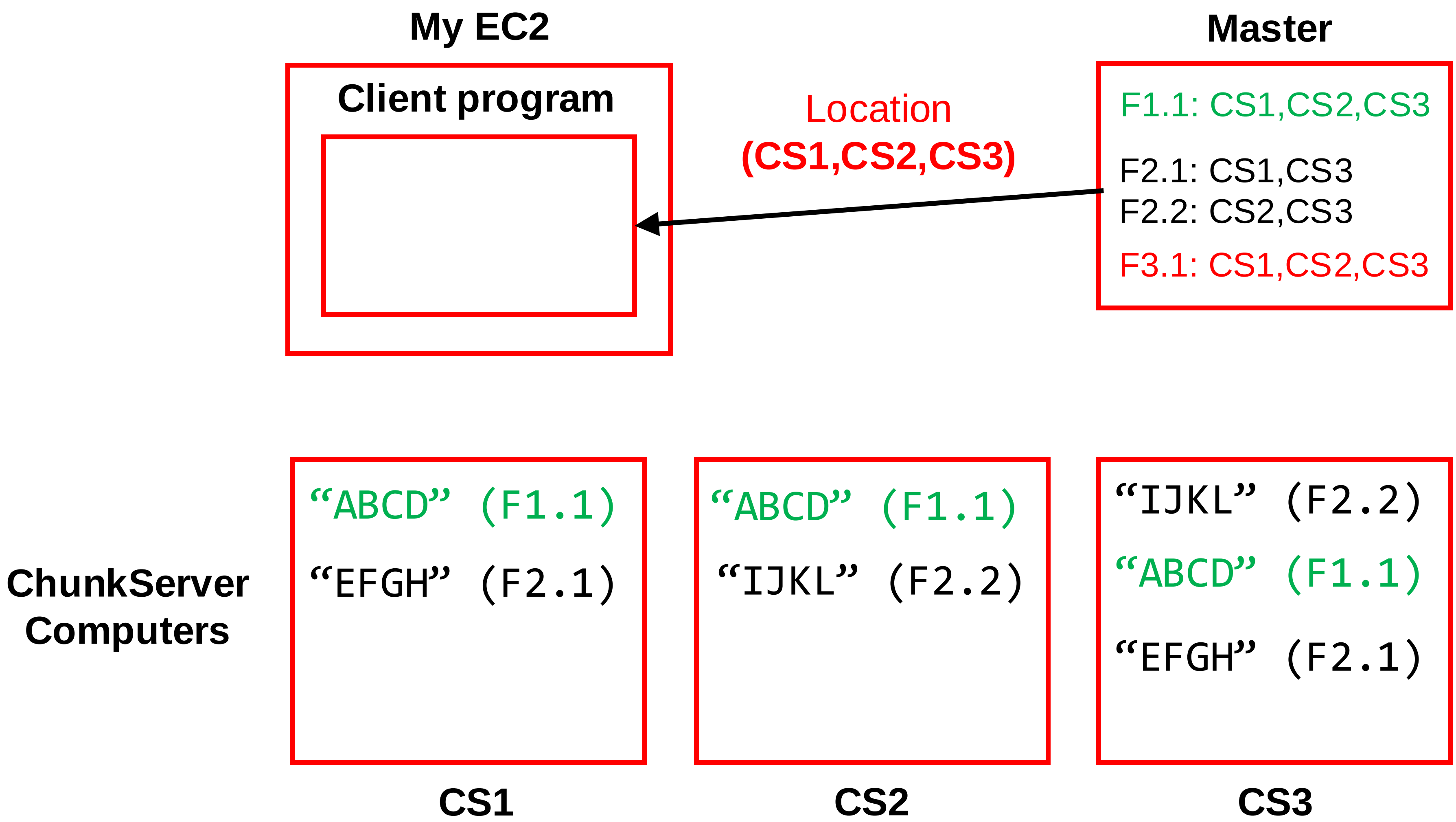
Master/Worker architecture



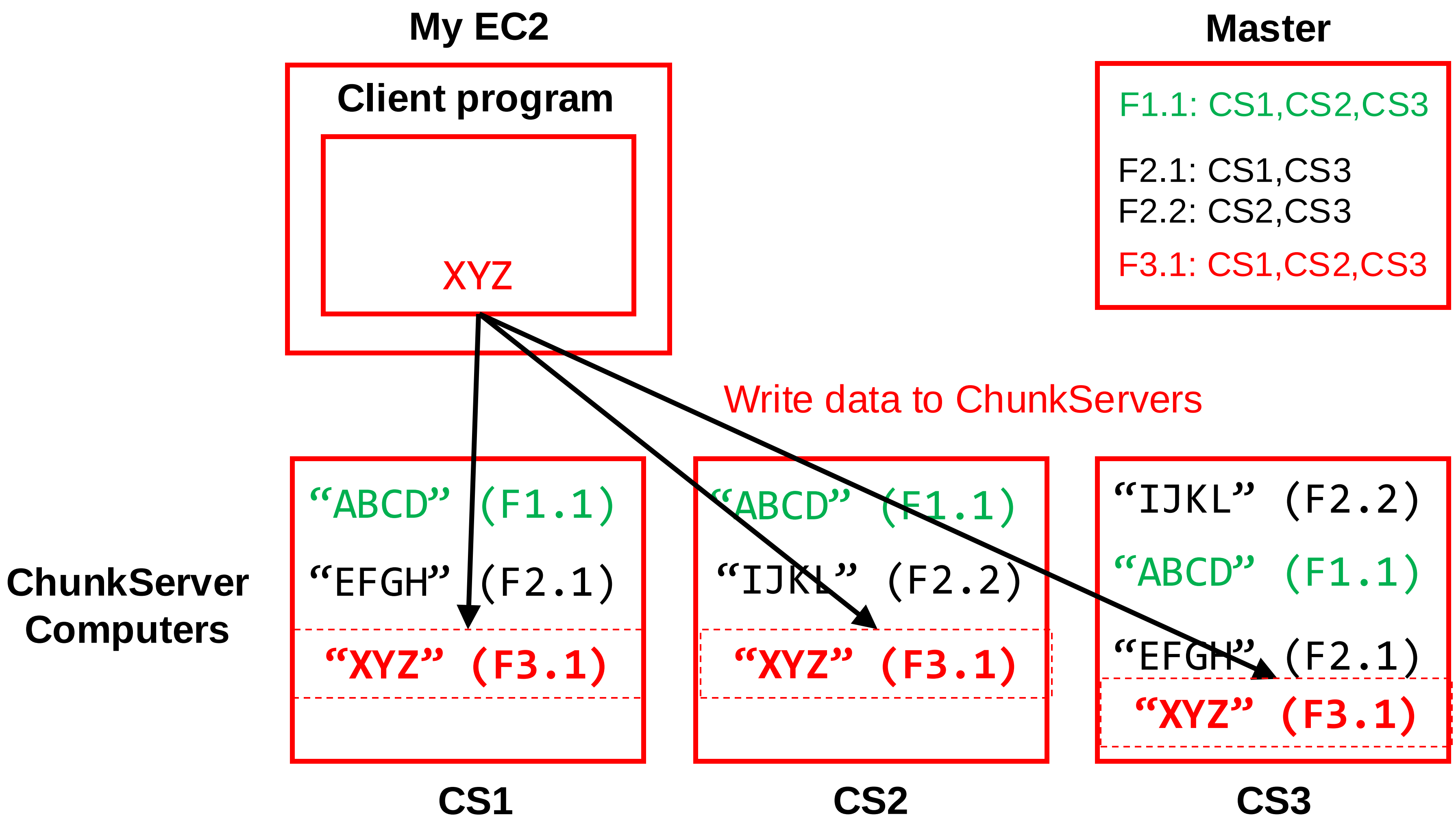
Master/Worker architecture



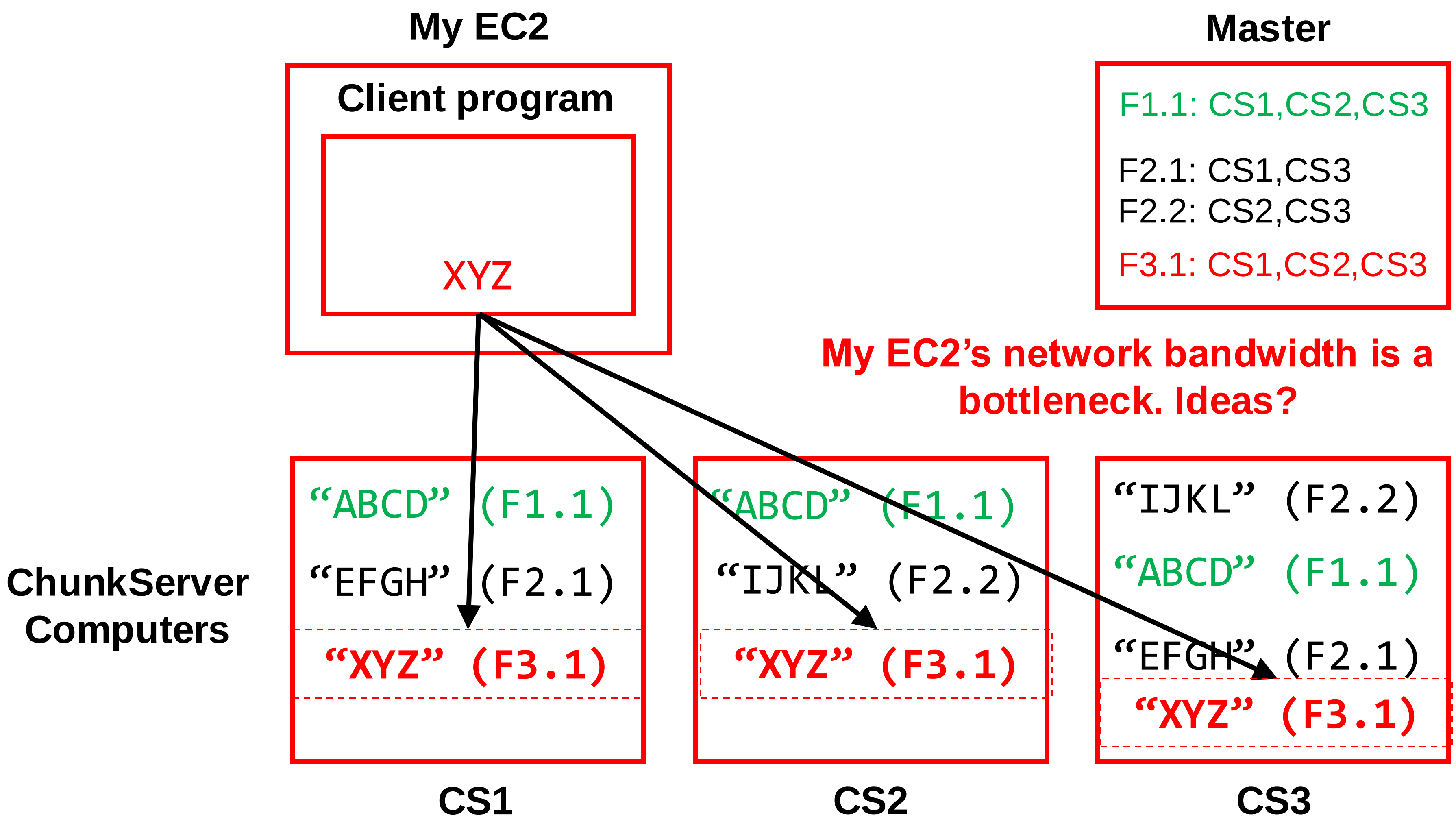
Master/Worker architecture



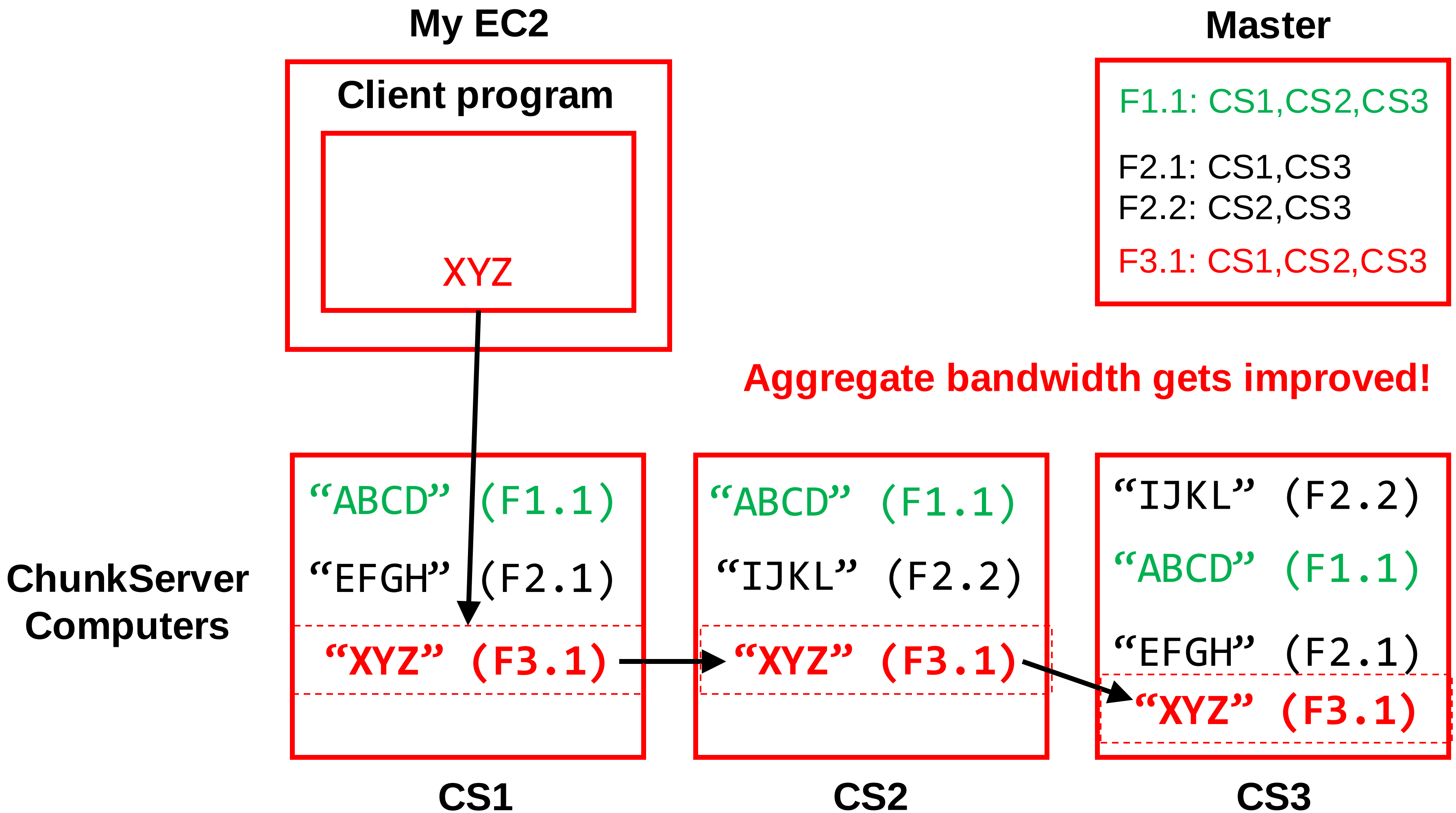
Master/Worker architecture



Master/Worker architecture



Pipelined writes



How are reads/writes amplified at disk level?

Q1: If a client **writes** 4MB to a 2x replicated file, how much data does GFS **write** to disks?

Q2: If a client **reads** 2MB from a 3x replicated file, how much data do we **read** from disks?

Master

F1.1: CS1,CS2,CS3

F2.1: CS1,CS3
F2.2: CS2,CS3

F3.1: CS1,CS2,CS3

ChunkServer Computers

CS1	CS2	CS3
"ABCD" (F1.1)	"ABCD" (F1.1)	"IJKL" (F2.2)
"EFGH" (F2.1)	"IJKL" (F2.2)	"ABCD" (F1.1)
"XYZ" (F3.1)	"XYZ" (F3.1)	"EFGH" (F2.1)
		"XYZ" (F3.1)

What are the tradeoffs of replication factor and chunk size?

Benefit of high replication?

Benefit of low replication?

Benefit of large block size?

Benefit of small block size?

Master

F1.1: CS1,CS2,CS3

F2.1: CS1,CS3
F2.2: CS2,CS3

F3.1: CS1,CS2,CS3

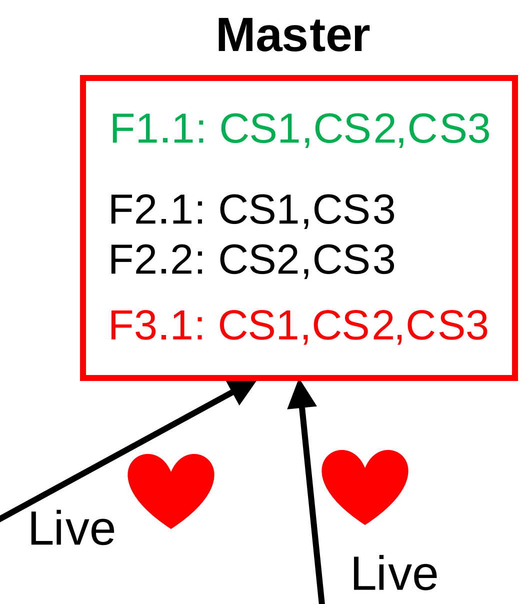
ChunkServer Computers

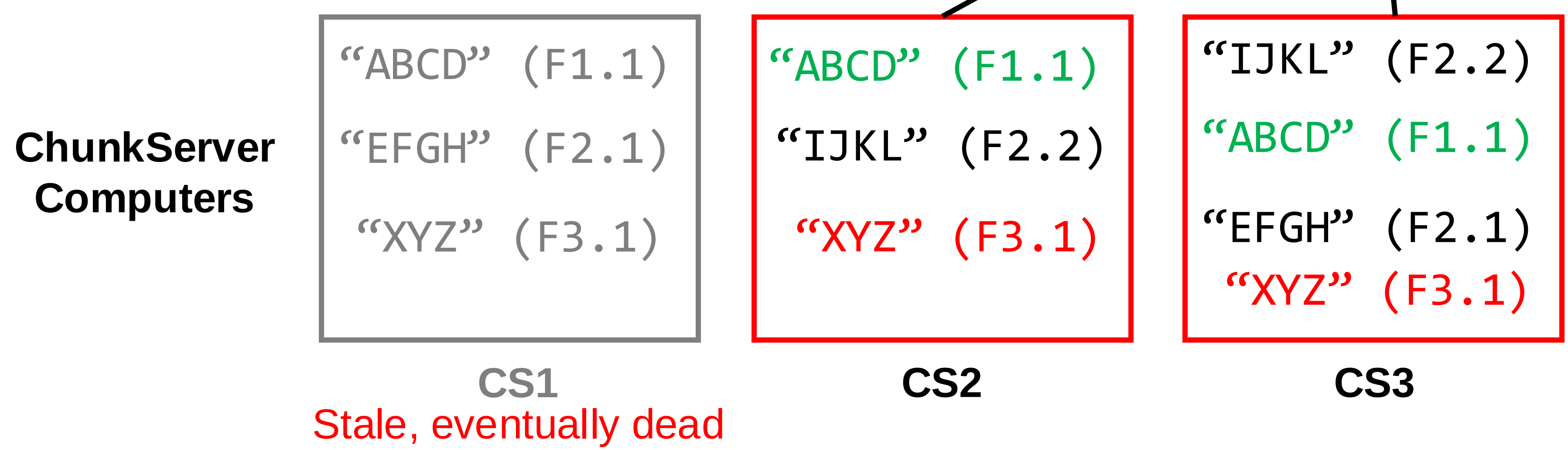
CS1	CS2	CS3
"ABCD" (F1.1)	"ABCD" (F1.1)	"IJKL" (F2.2)
"EFGH" (F2.1)	"IJKL" (F2.2)	"ABCD" (F1.1)
"XYZ" (F3.1)	"XYZ" (F3.1)	"EFGH" (F2.1)
		"XYZ" (F3.1)

How do we know when a ChunkServer fails?

Heartbeat message

- ChunkServers to Master
- Every N seconds (e.g., 3)
- Threshold for no message
  - Stale (> M seconds)
  - Dead (> N seconds)
- When dead, chunks might be underreplicated and need new replicas





Summary: Key ideas applied to GFS

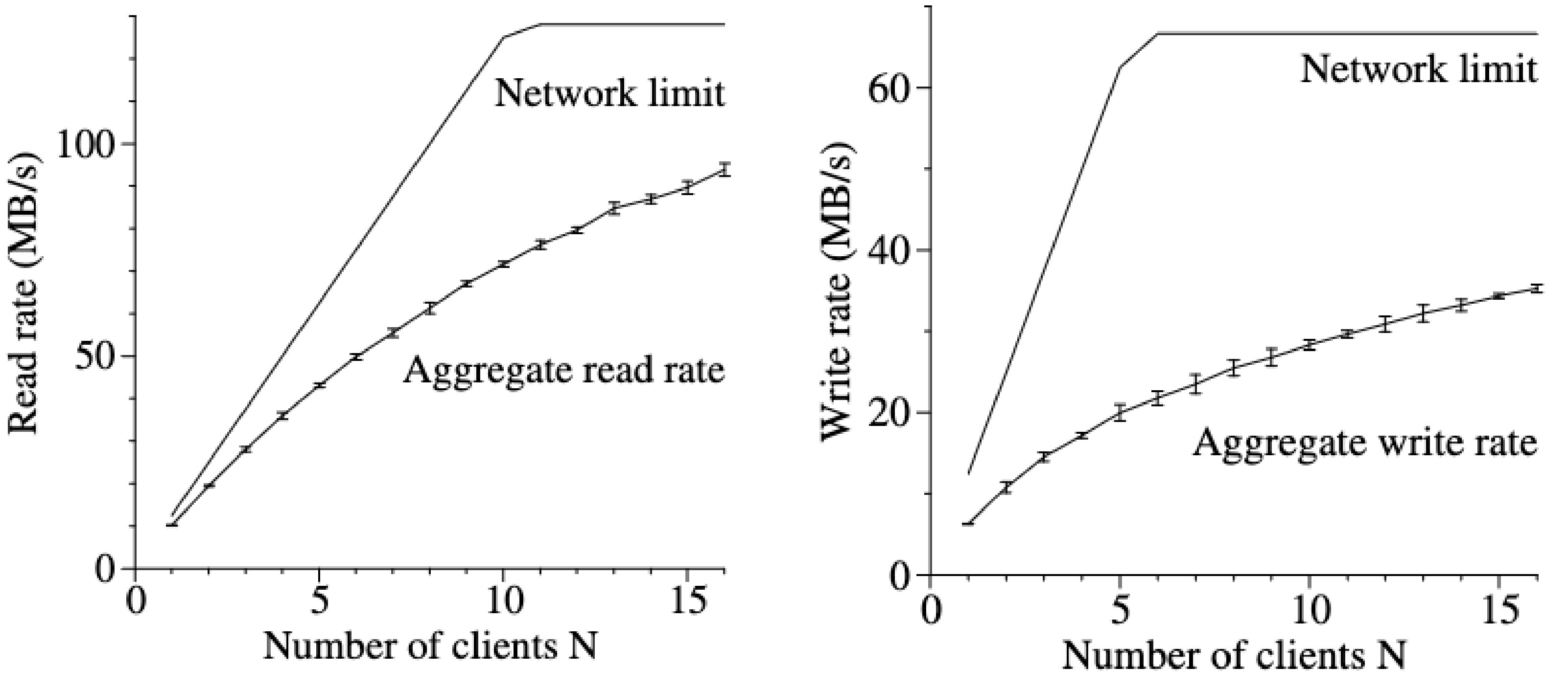
- To scale out…
- To handle faults…
- To detect faults…
- To optimize I/O…

Summary: Key ideas applied to GFS

- To scale out…
  - Partition your data
- To handle faults…
  - Replicate your data
- To detect faults…
  - Sand heartbeats
- To optimize I/O…
  - Pipeline writes

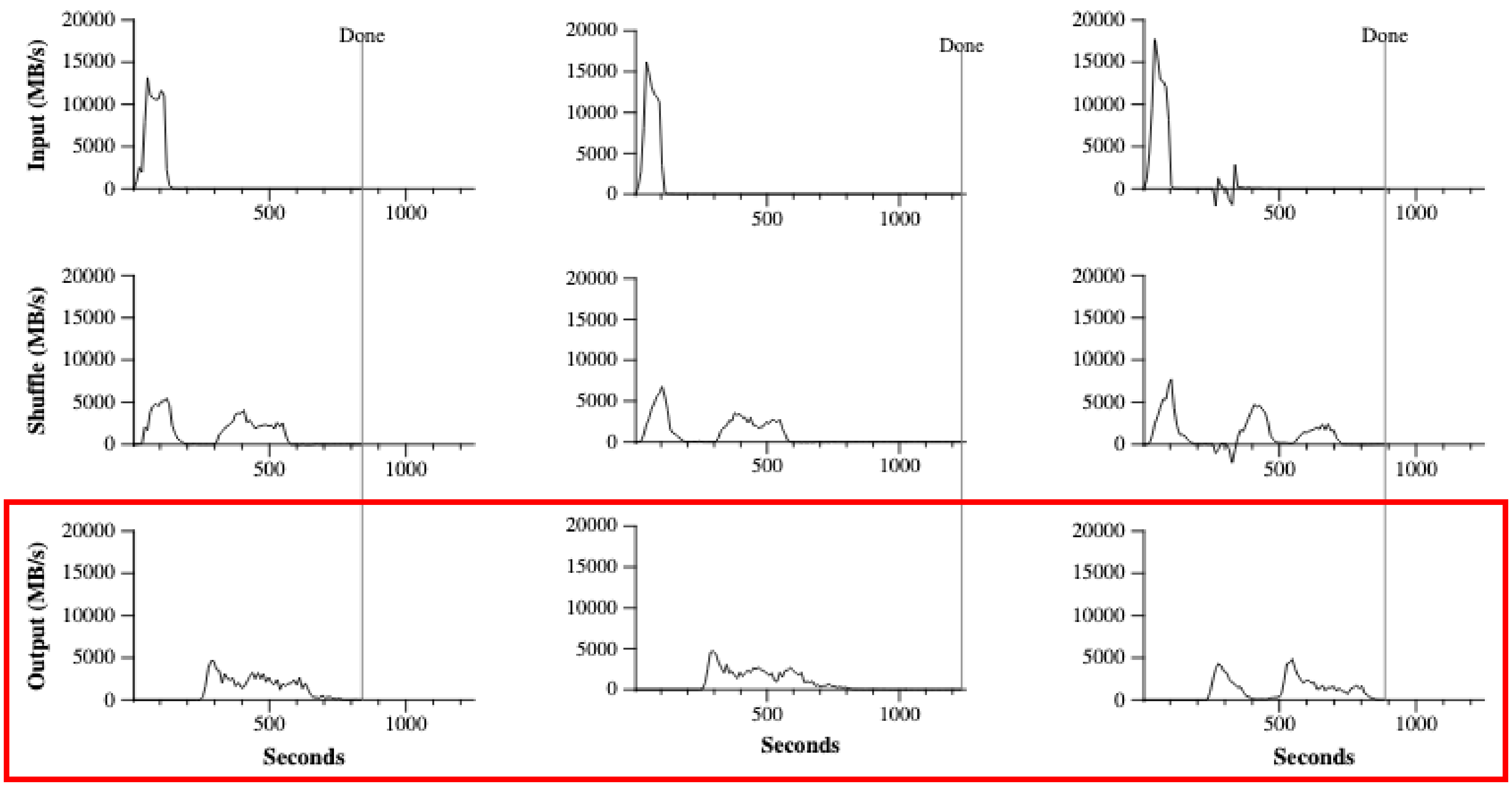
Discussion: GFS eval (GFS paper)

Describe your observations from “Figure 3”



(a) Reads (b) Writes

Discussion: GFS affects MapReduce perf



(a) Normal execution (b) No backup tasks (c) 200 tasks killed

Hadoop ecosystem

Yahoo, Facebook, Cloudera, and others developed open-source Hadoop ecosystem, mirroring Google's big data systems

	Google (paper only)	Hadoop (open source)	Modern Hadoop
Distributed File System	GFS	HDFS	
Distributed Processing & Analytics	MapReduce	Hadoop MapReduce	Spark
Distributed Database	BigTable	HBase	MongoDB

https://hadoop.apache.org/