CS 795 Distributed Systems & Cloud Computing

Fall 2018

Lec 5: Big data systems

Yue Cheng

Agenda

- GFS/MapReduce primer
- 5 min break
- Spark discussion
- 5 min break
- Bigtable discussion
- Project discussion + Q&A

Google File System
MapReduce

Google File System

MapReduce

Google File System (GFS) Overview

- Motivation
- Architecture

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

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

Example Workloads

- Read entire dataset, do computation over it
- 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

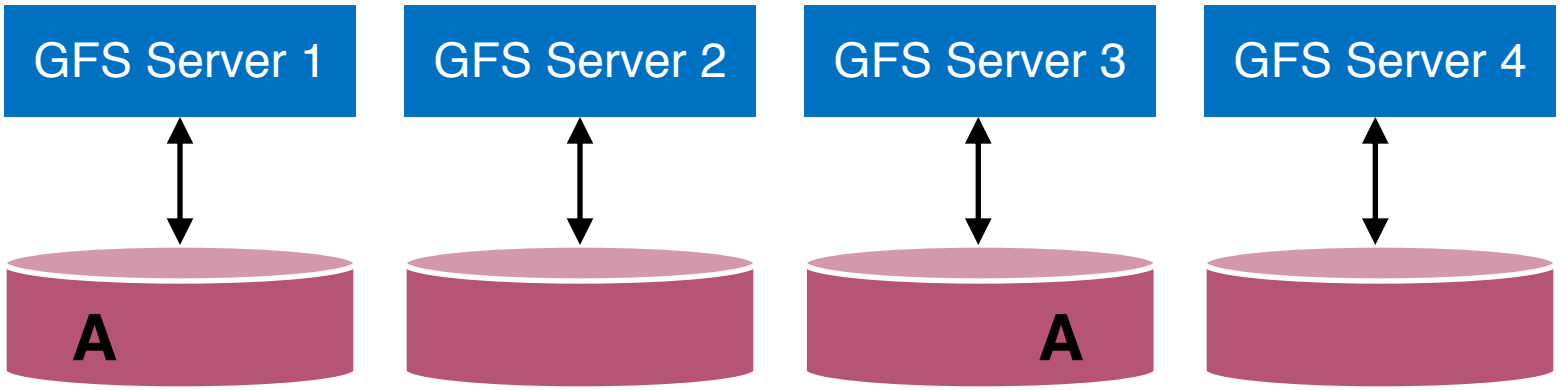
Real-world use cases of HDFS

- NetApp provides storage solution to businesses/companies
- Large financial firm: 60 PB of raw data
- Requires **1200** HDFS storage nodes organized as a data lake
  - Includes 3x replicas + overprovisioned space for failures etc.
  - 288 TB disk capacity per HDFS node (storage dense configuration)

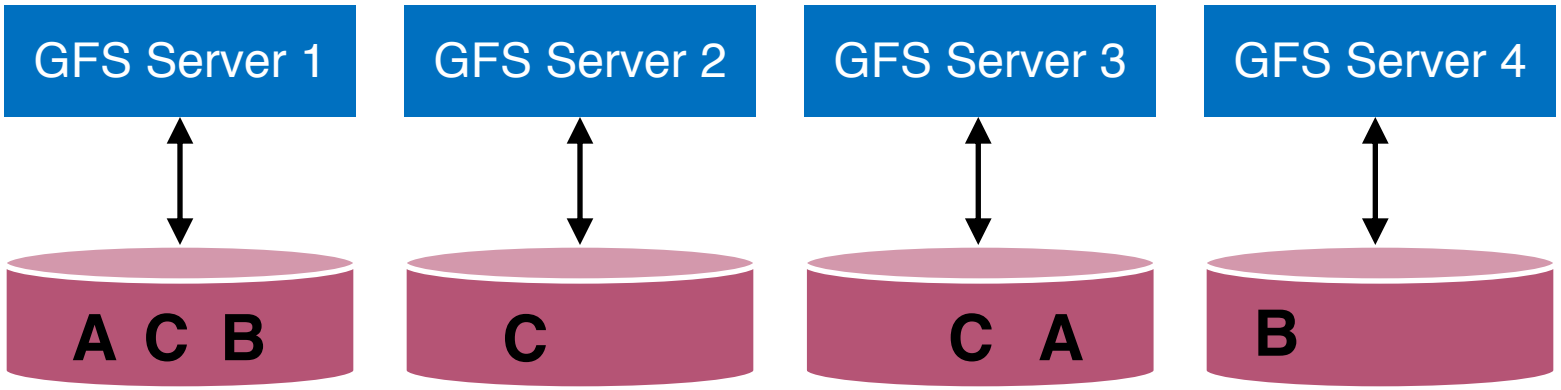
Google File System (GFS) Overview

- Motivation
- Architecture

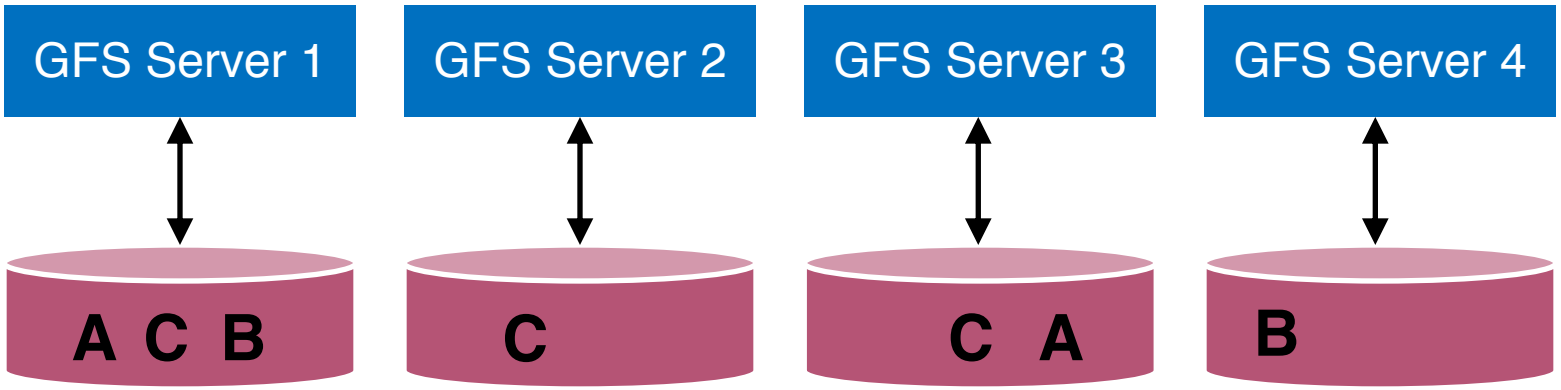
Replication



Replication



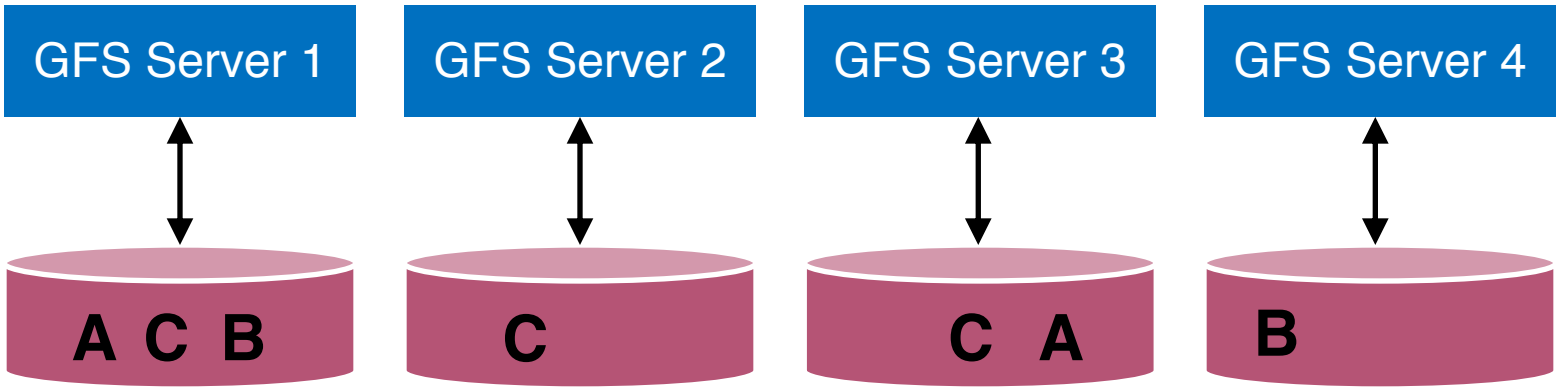
Replication



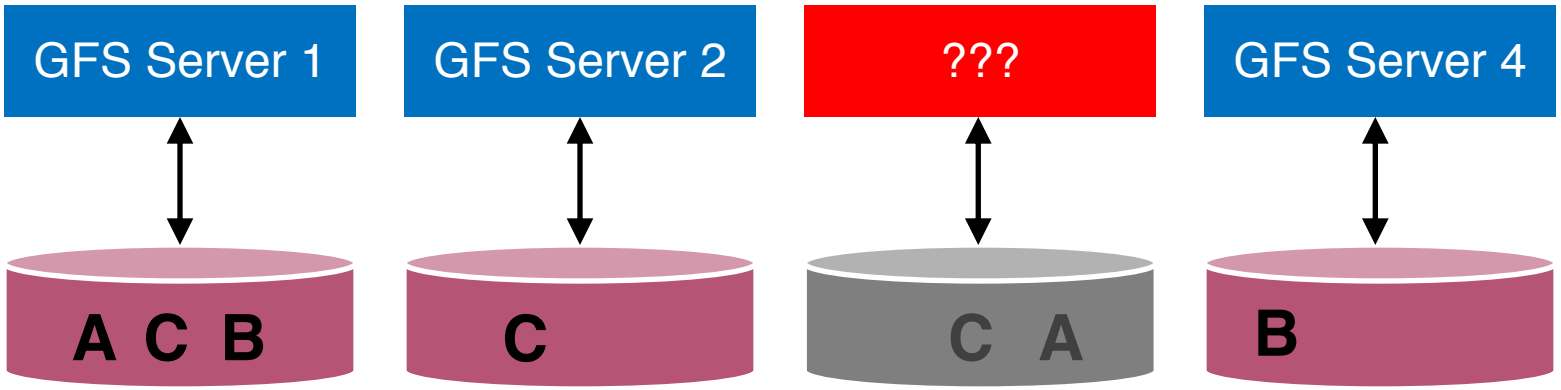
Similar to RAID, but less orderly than RAID

- Machines' capacity may vary
- Different data may have different replication factors

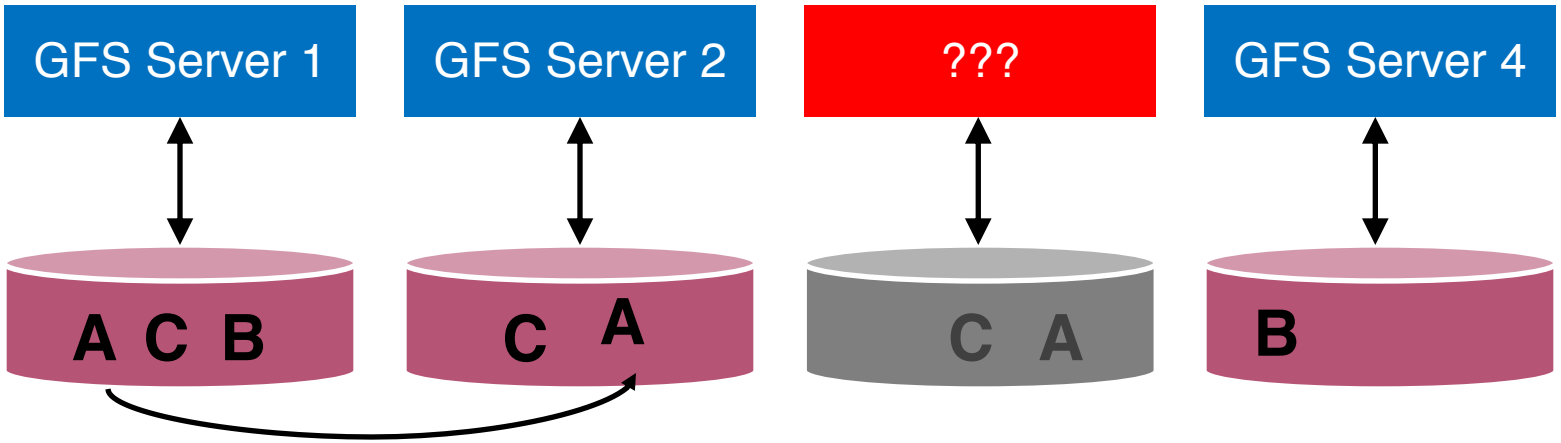
Data Recovery



Data Recovery

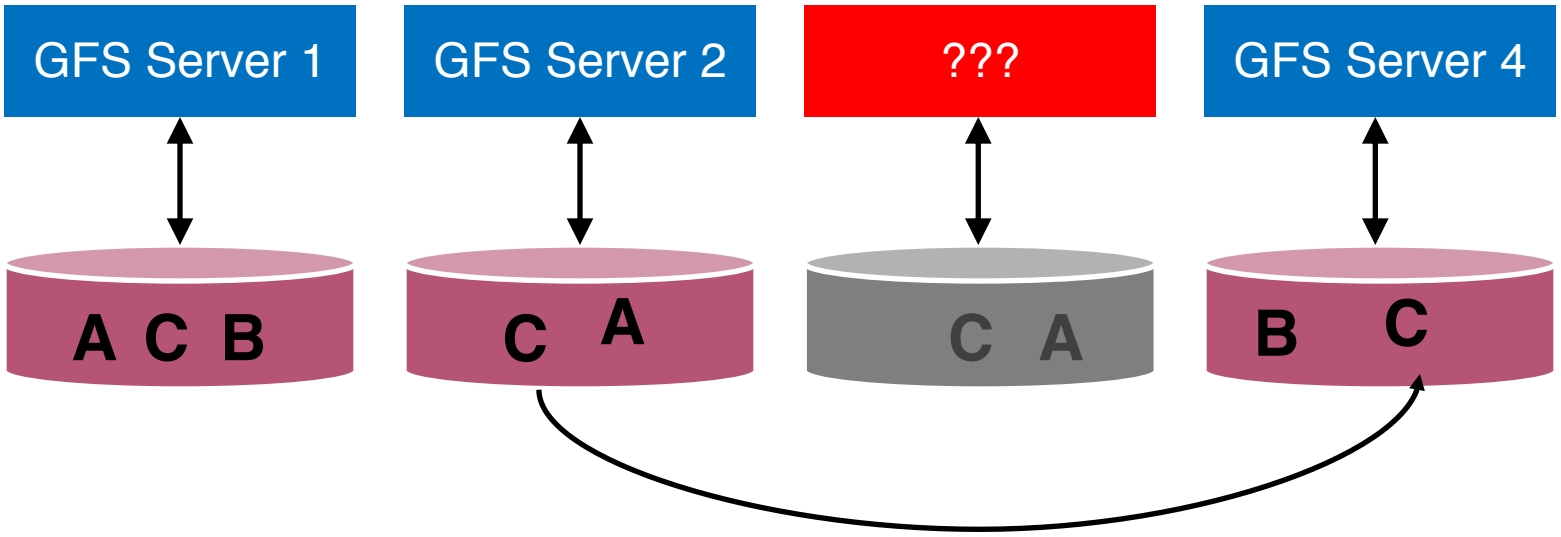


Data Recovery



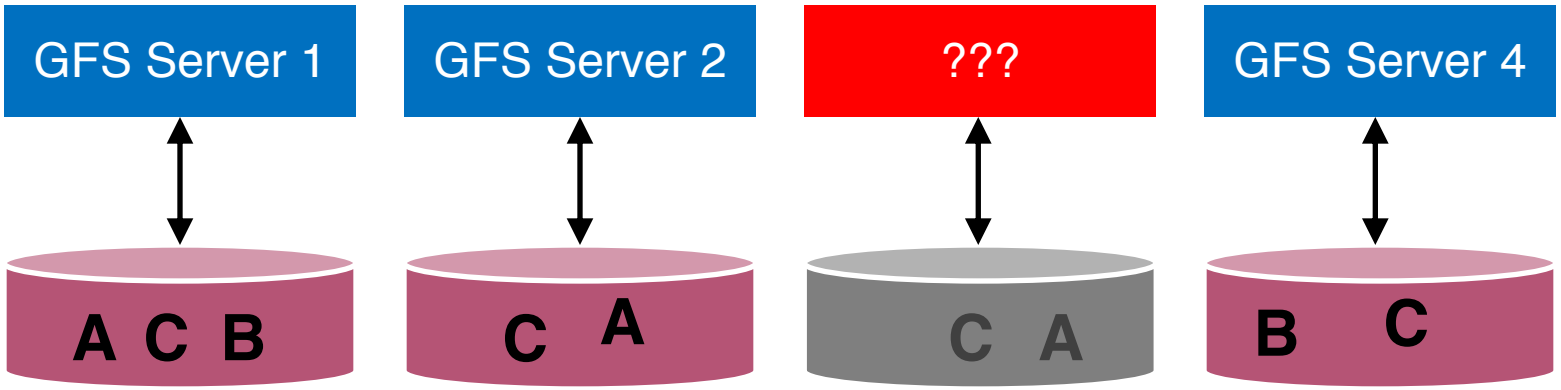
Replicating A to maintain a replication factor of 2

Data Recovery



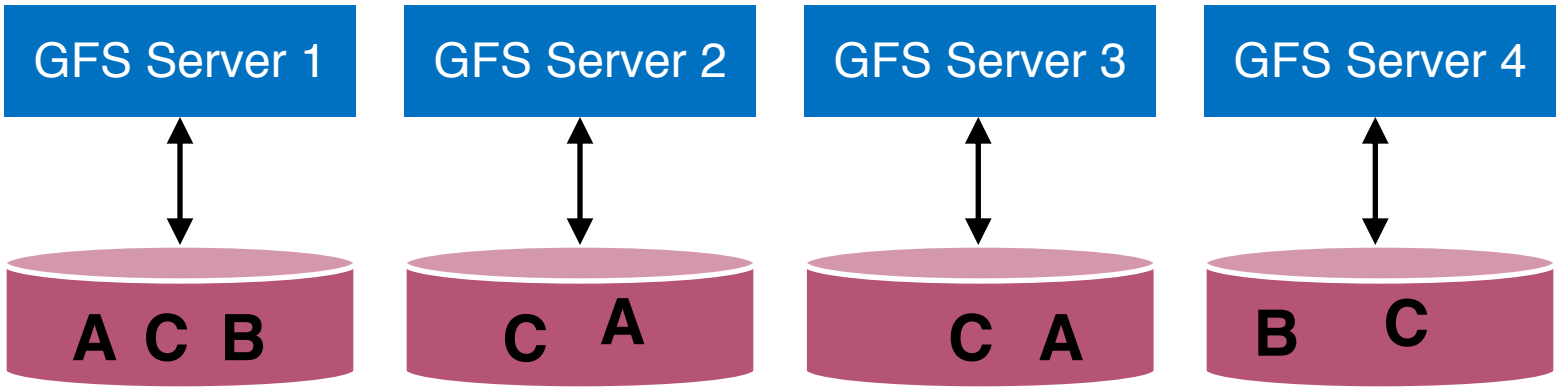
Replicating C to maintain a replication factor of 3

Data Recovery



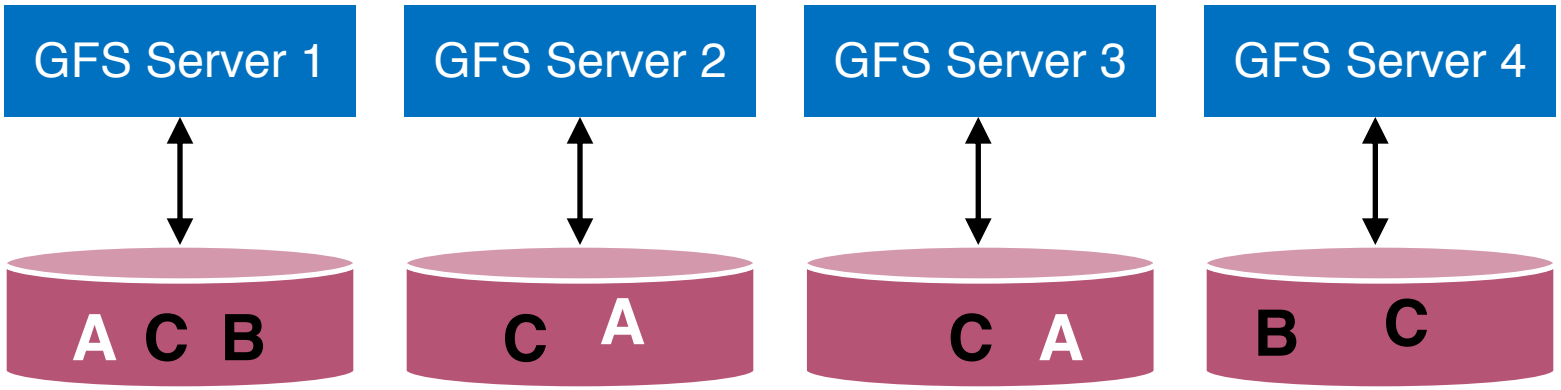
Machine may be dead forever, or it may come back

Data Recovery

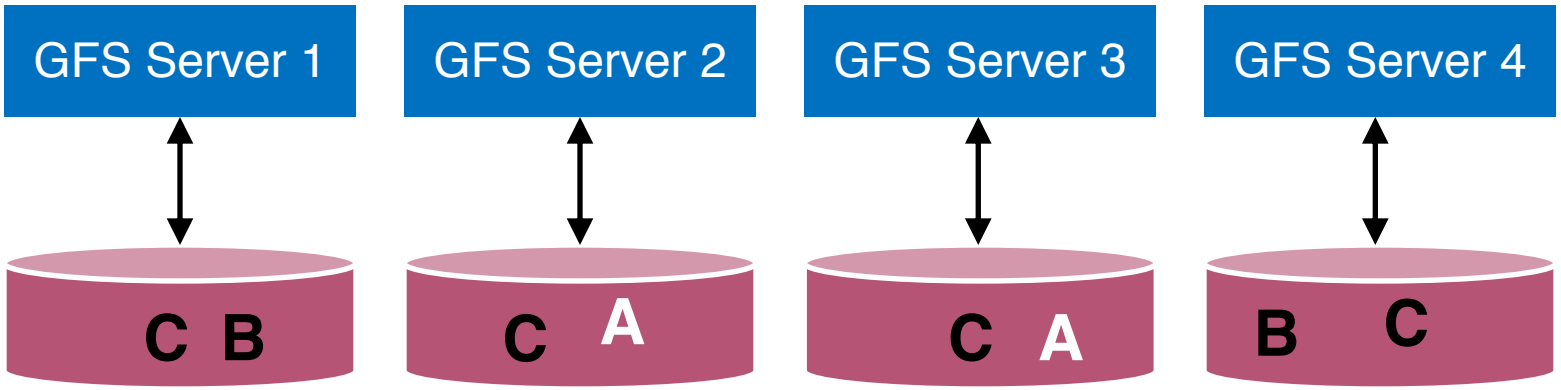


Machine may be dead forever, or it may come back

Data Recovery



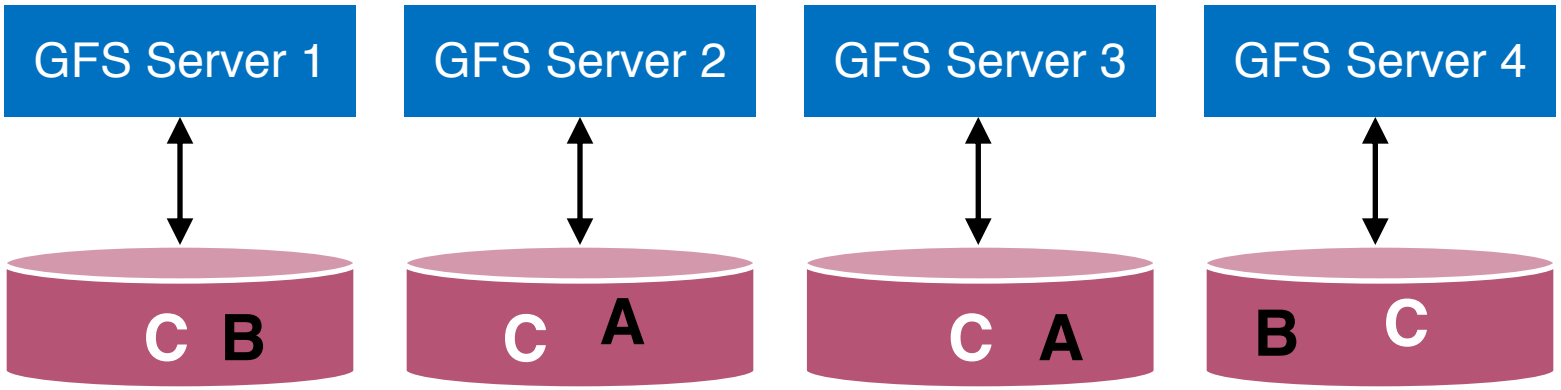
Data Recovery



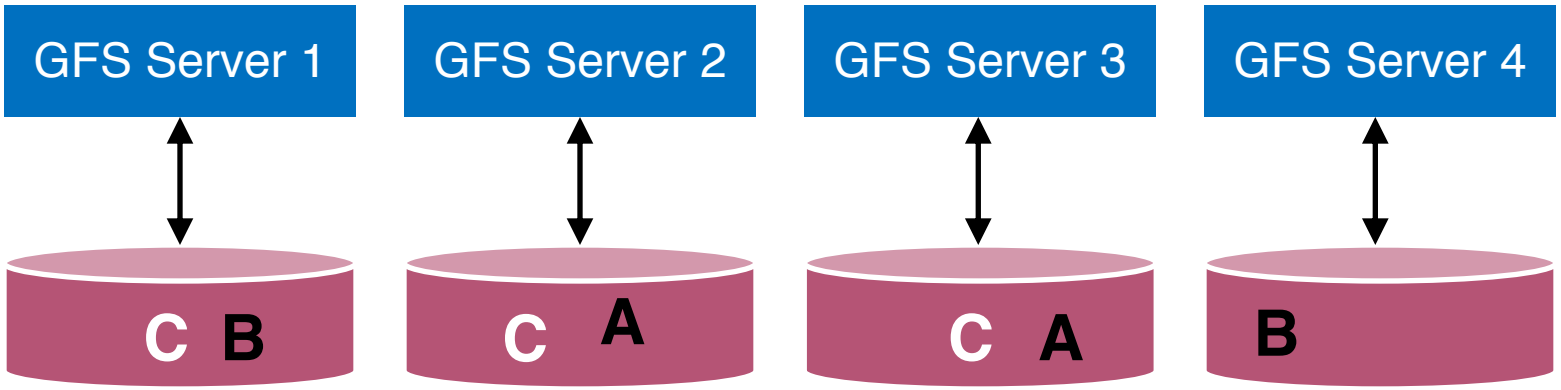
Data Rebalancing

Deleting one A to maintain a replication factor of 2

Data Recovery



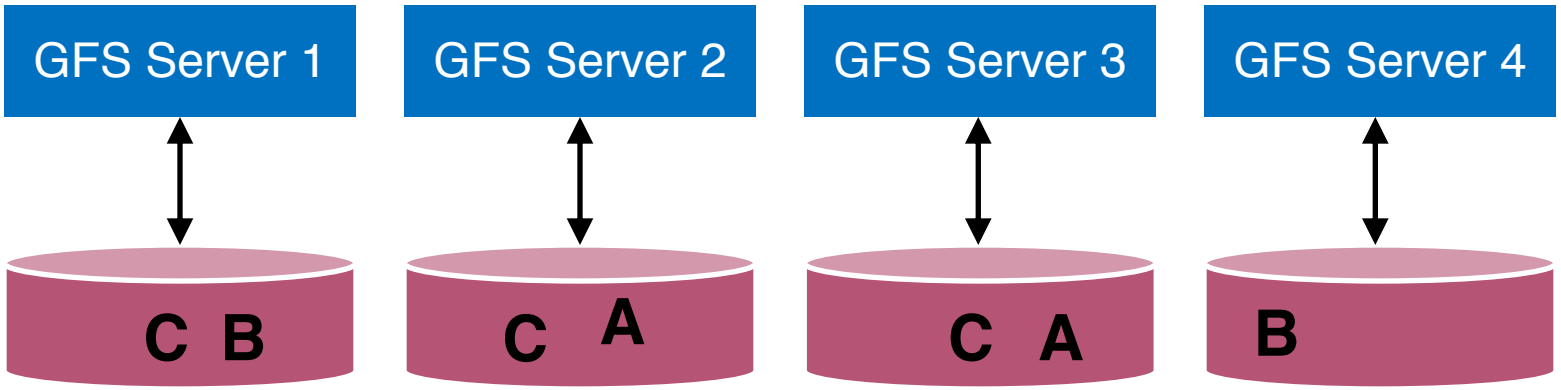
Data Recovery



Data Rebalancing

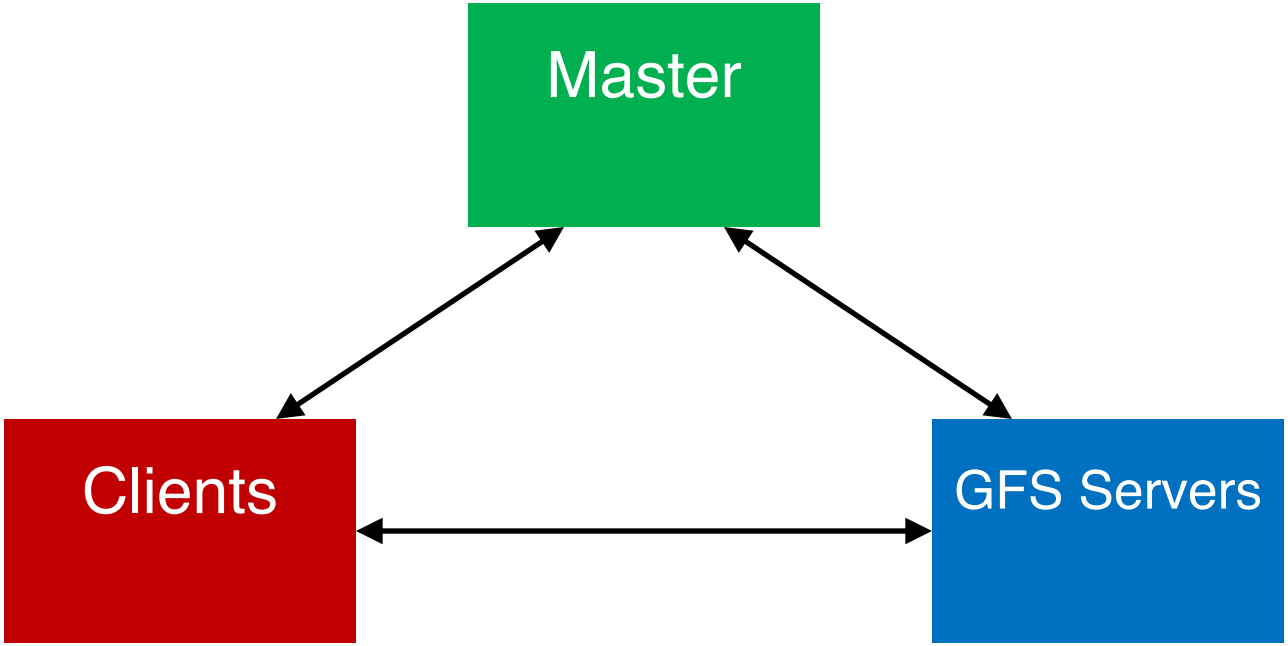
Deleting one C to maintain a replication factor of 3

Data Recovery

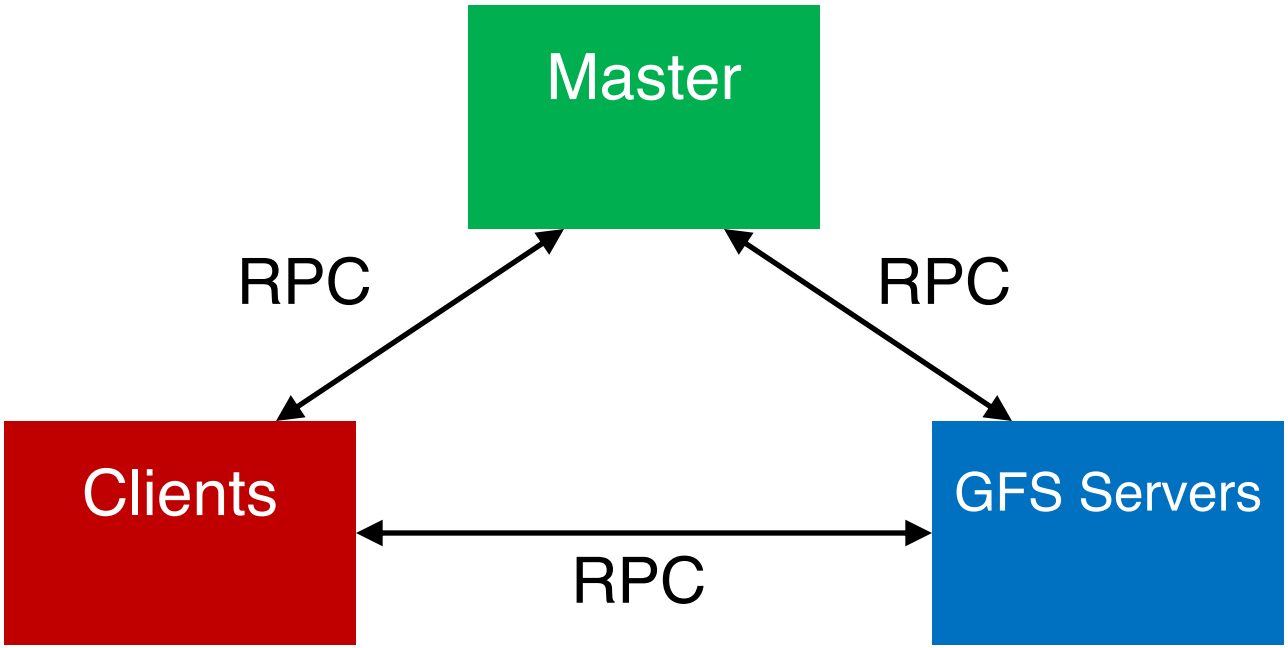


Question: how to maintain a global view of all data distributed across machines?

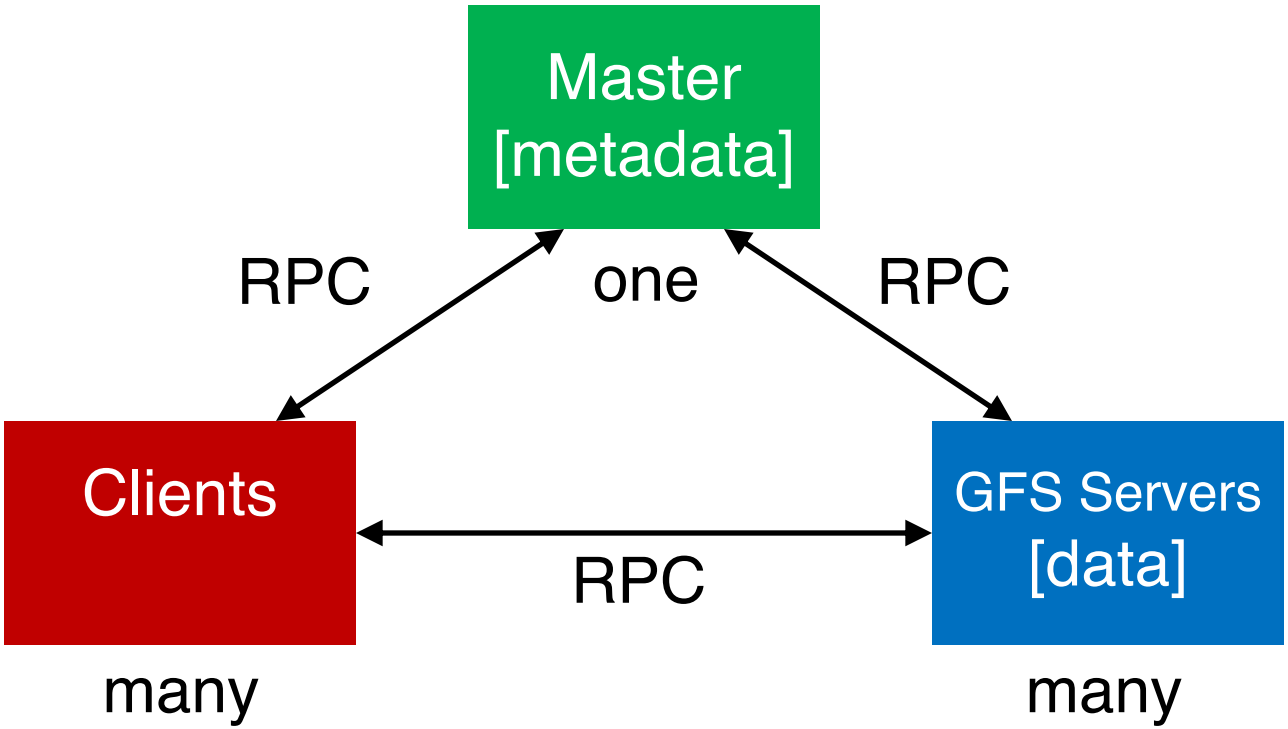
GFS Architecture



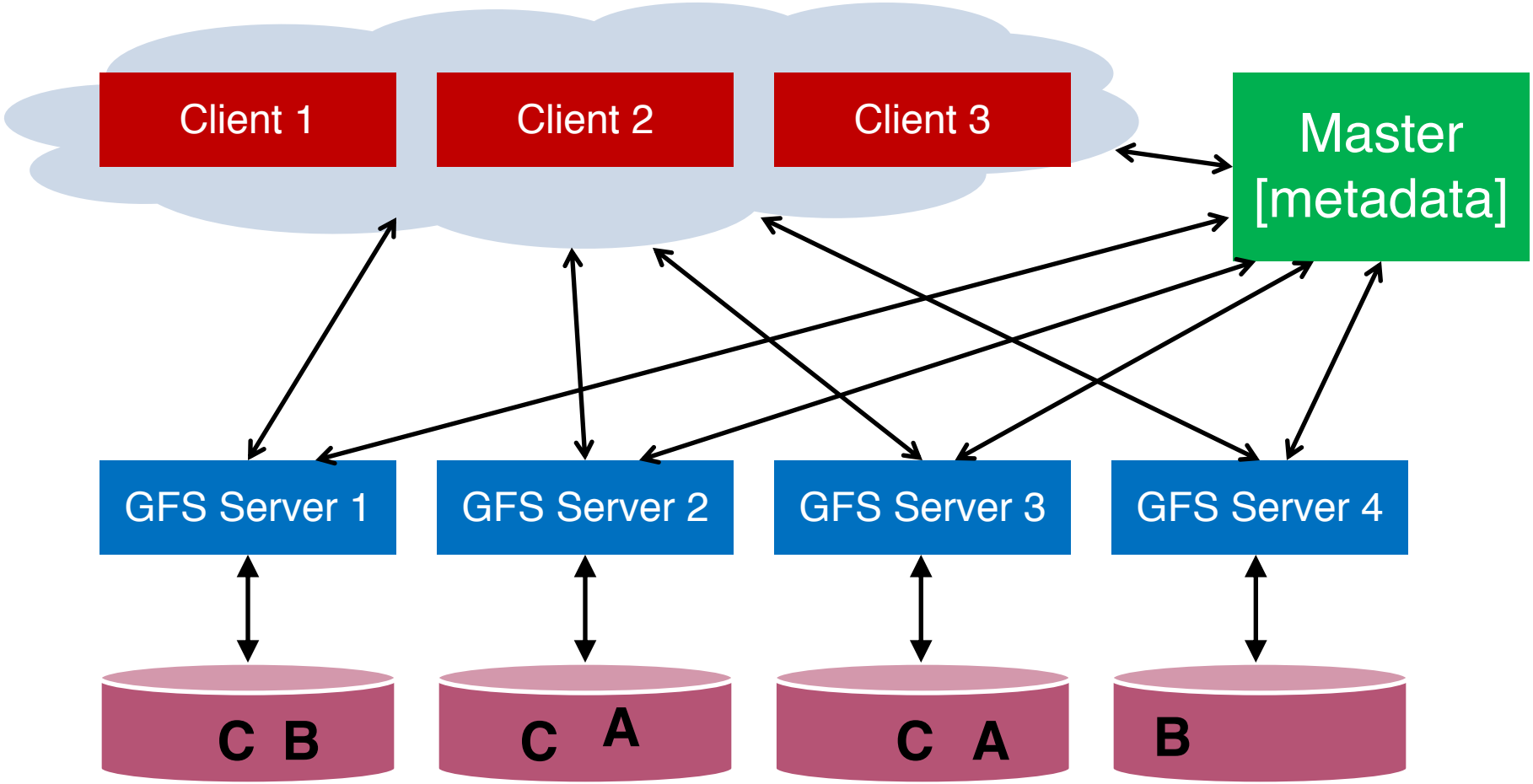
GFS Architecture



GFS Architecture



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

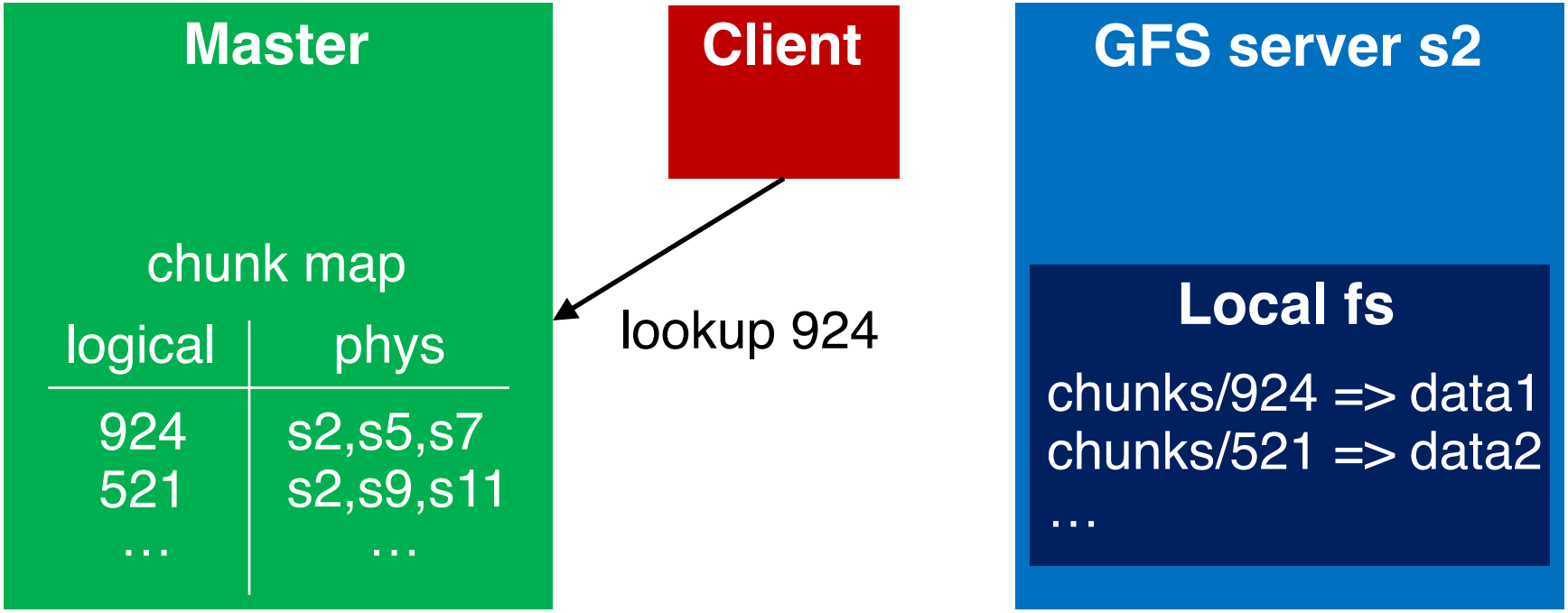
logical	phys
924	s2,s5,s7
521	s2,s9,s11
…	…

GFS server s2

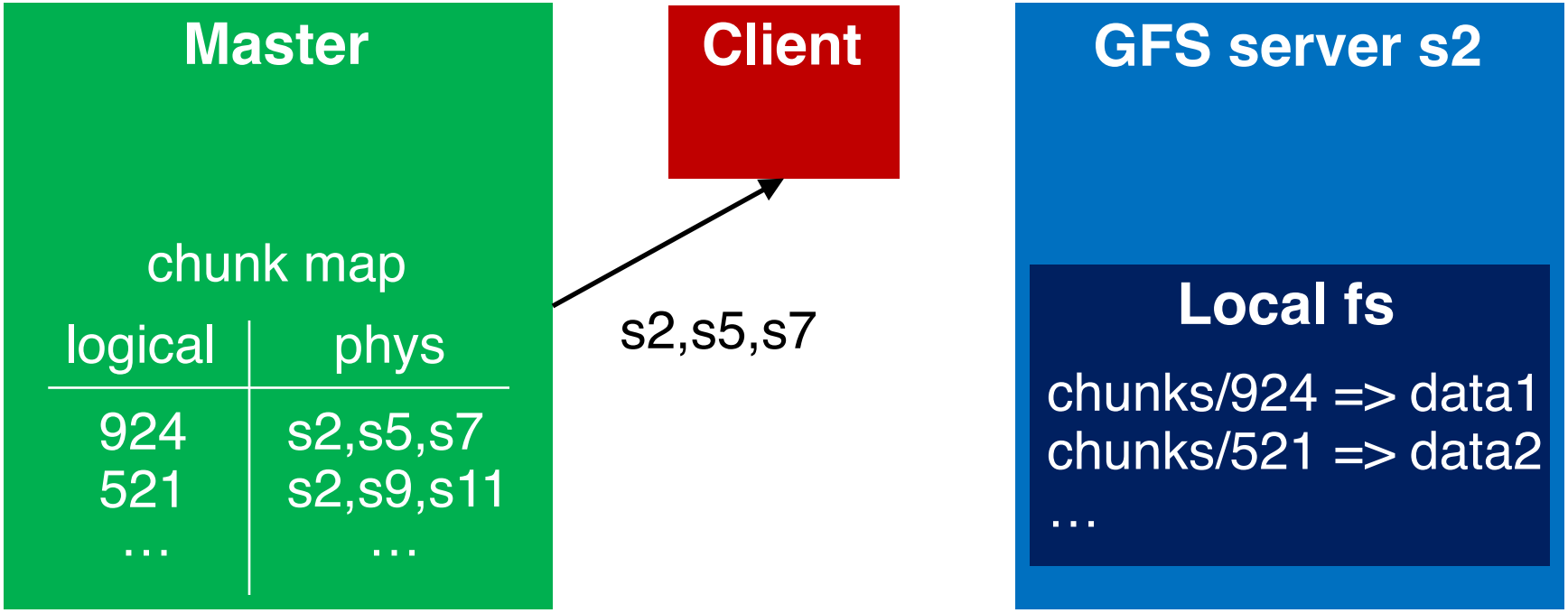
Local fs

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

Client Reads a Chunk

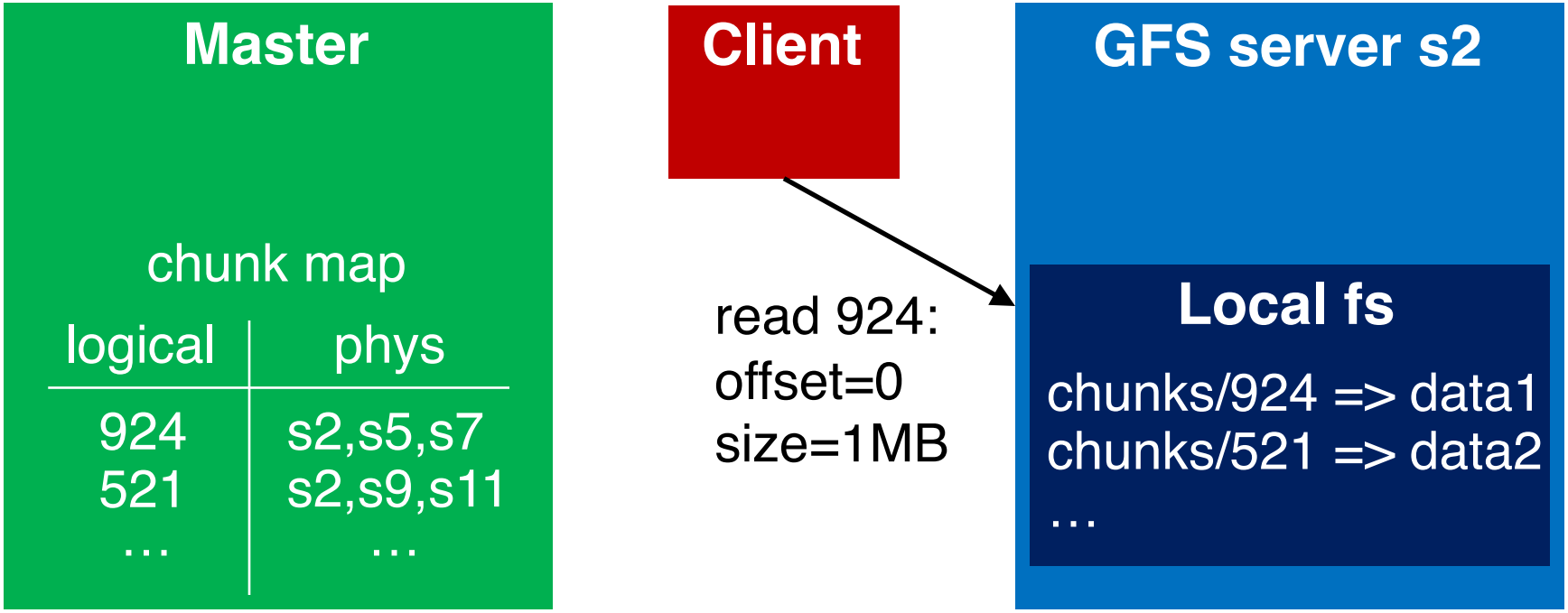


Client Reads a Chunk

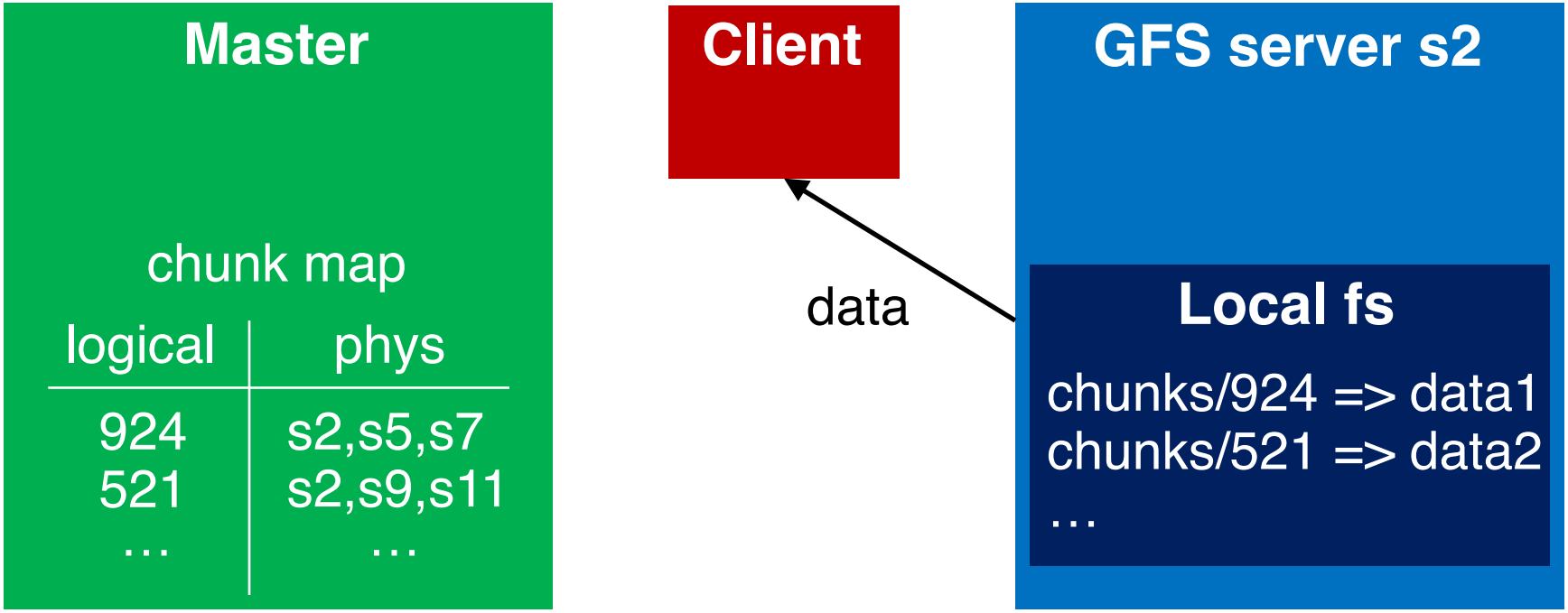


Client Reads a Chunk

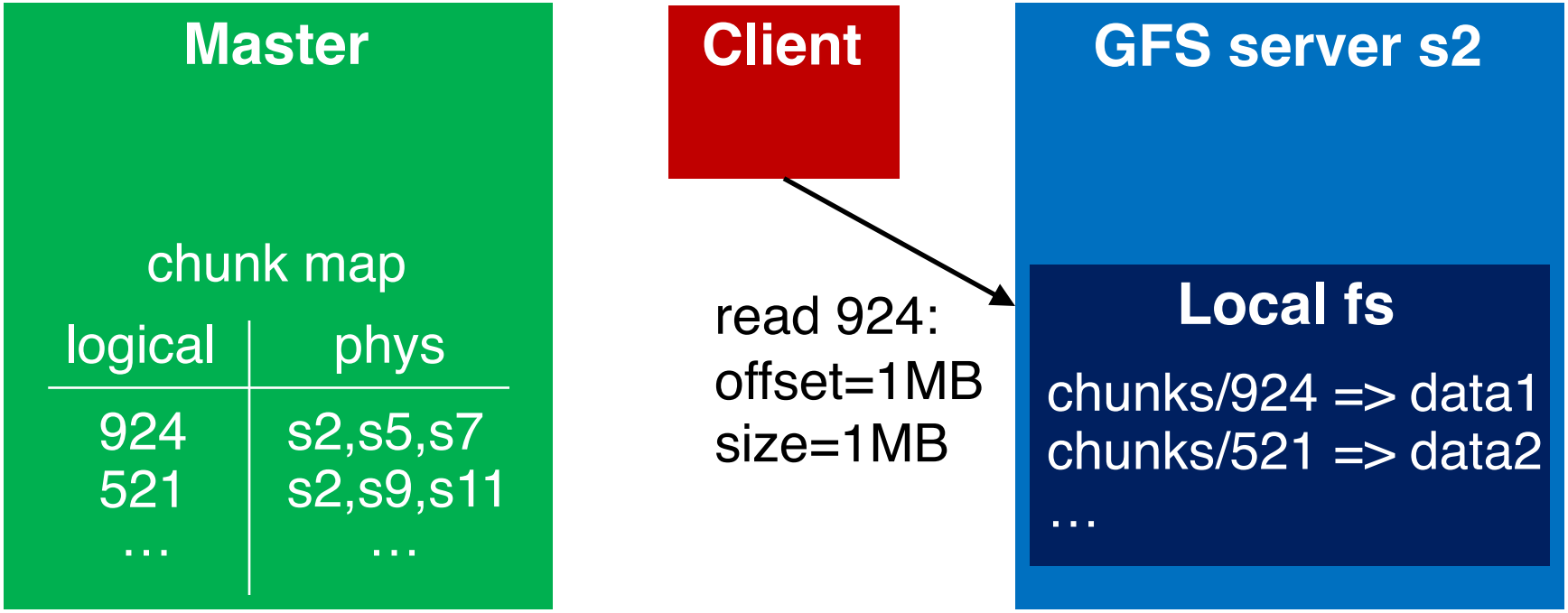
Client Reads a Chunk



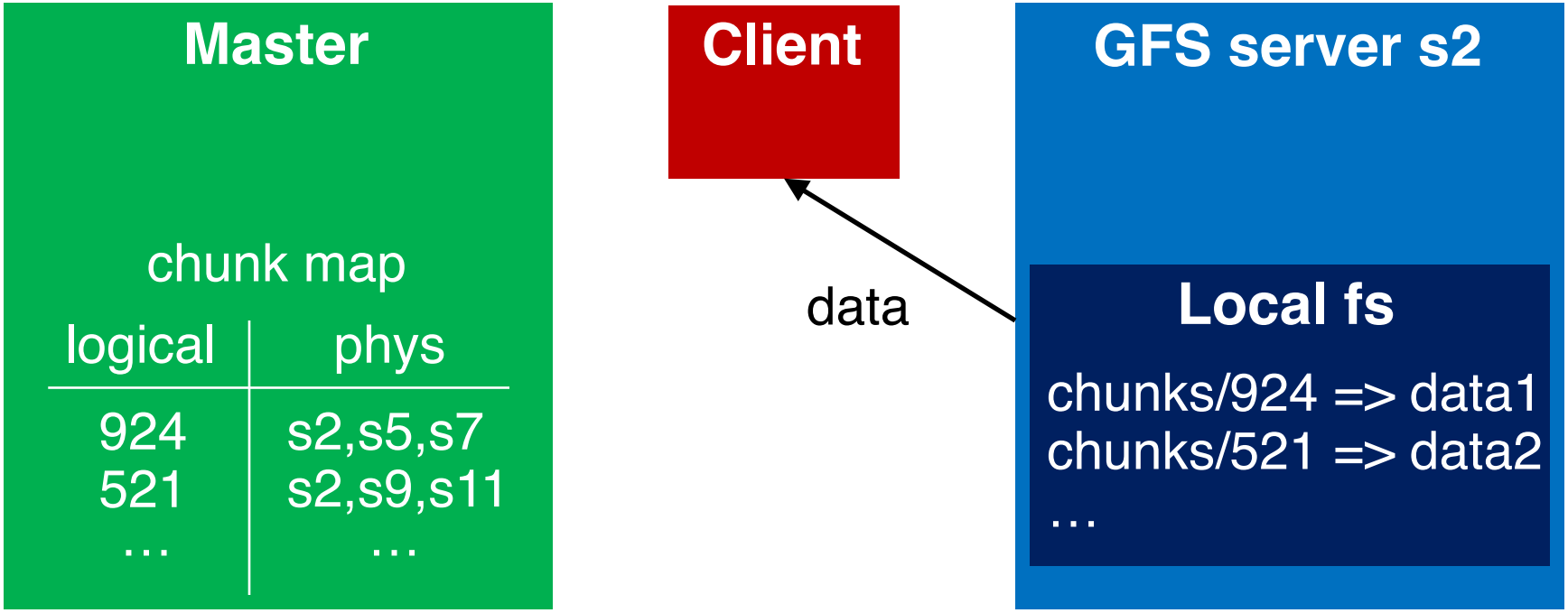
Client Reads a Chunk



Client Reads a Chunk



Client Reads a Chunk



File Namespace

path names mapped to logical names

Google File System
MapReduce

MapReduce Overview

- Motivation
- Architecture
- Programming Model

Problem

- Datasets are too big to process using single machine
- Good concurrent processing engines are rare
- Want a concurrent processing framework that is:
  - easy to use (no locks, CVs, race conditions)
  - general (works for many problems)

MapReduce

- Strategy: break data into buckets, do computation over each bucket
- Google published details in 2004
- Open source implementation: Hadoop

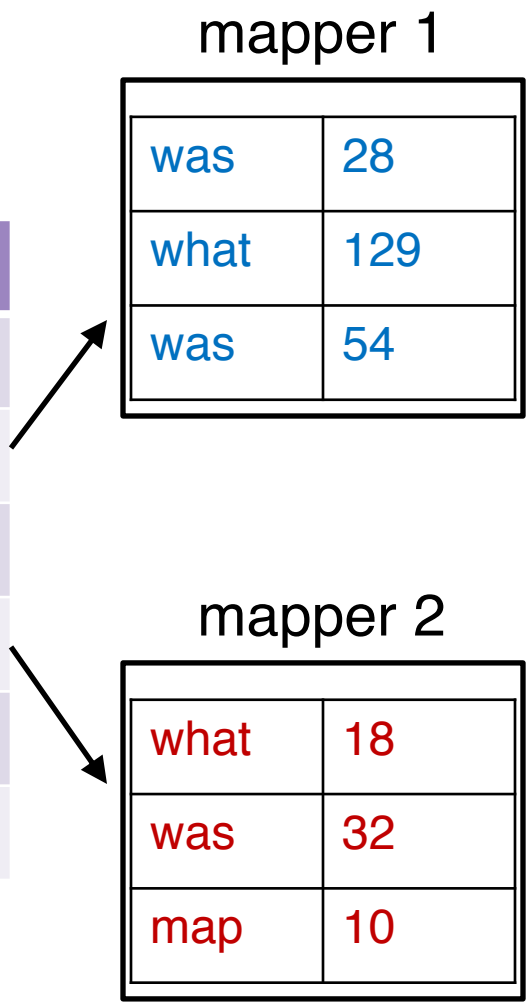
Example: Word Count

Word	Count
was	28
what	129
was	54
what	18
was	32
map	10

How to quickly sum word counts with multiple machines concurrently?

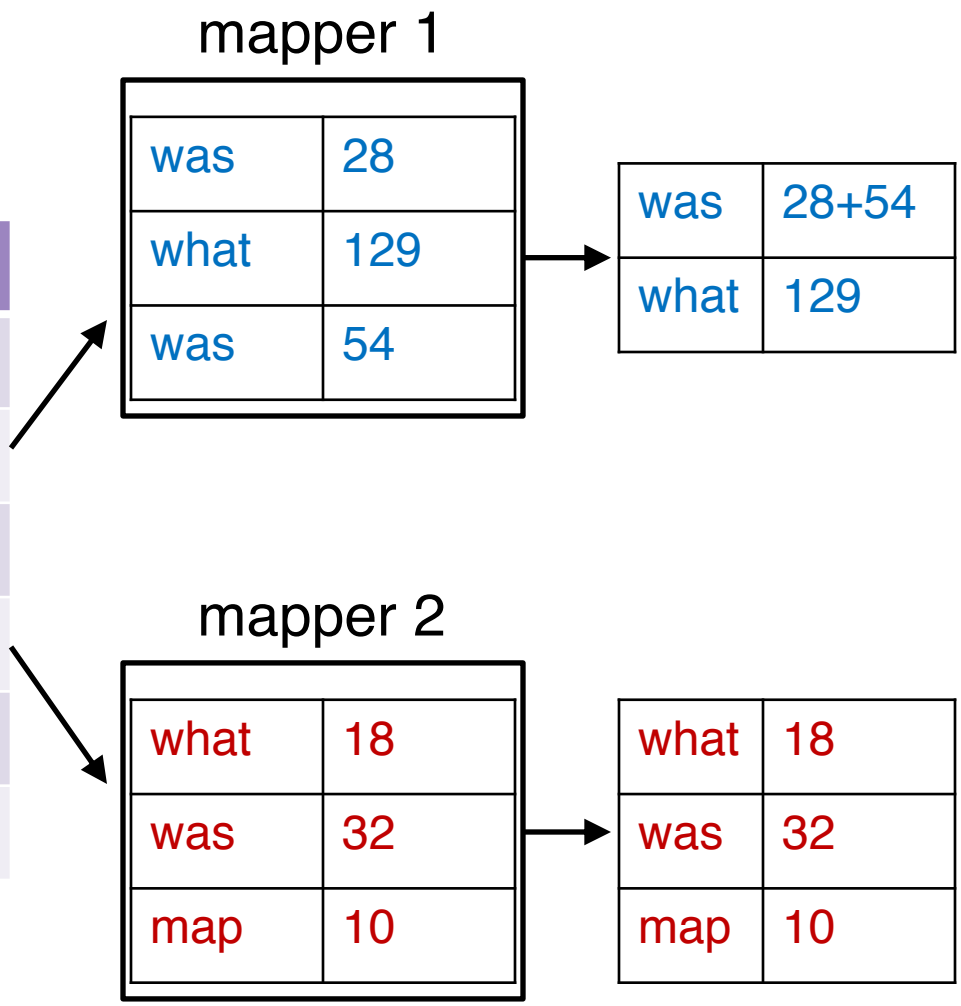
Example: Word Count

Word	Count
was	28
what	129
was	54
what	18
was	32
map	10



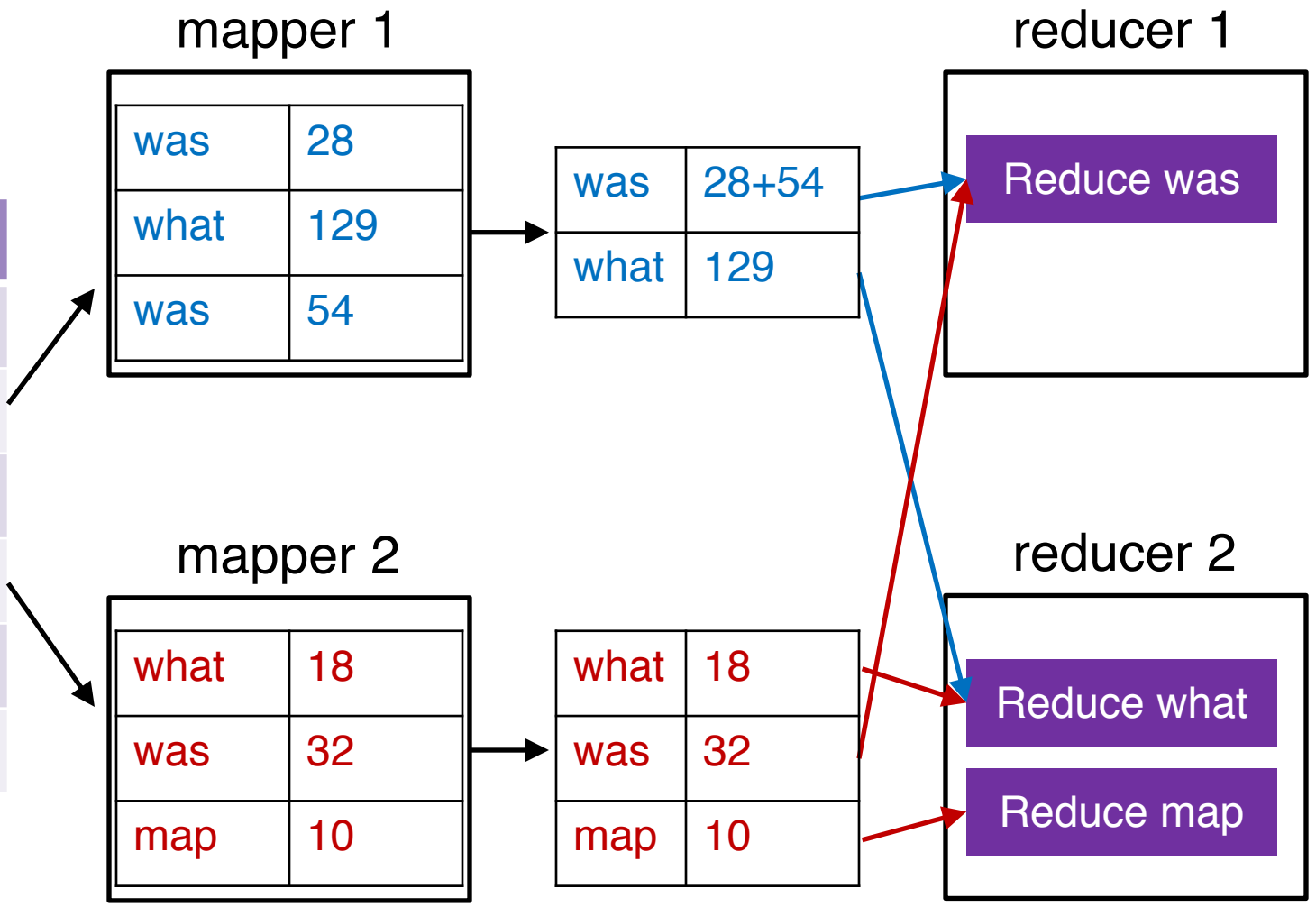
Example: Word Count

Word	Count
was	28
what	129
was	54
what	18
was	32
map	10



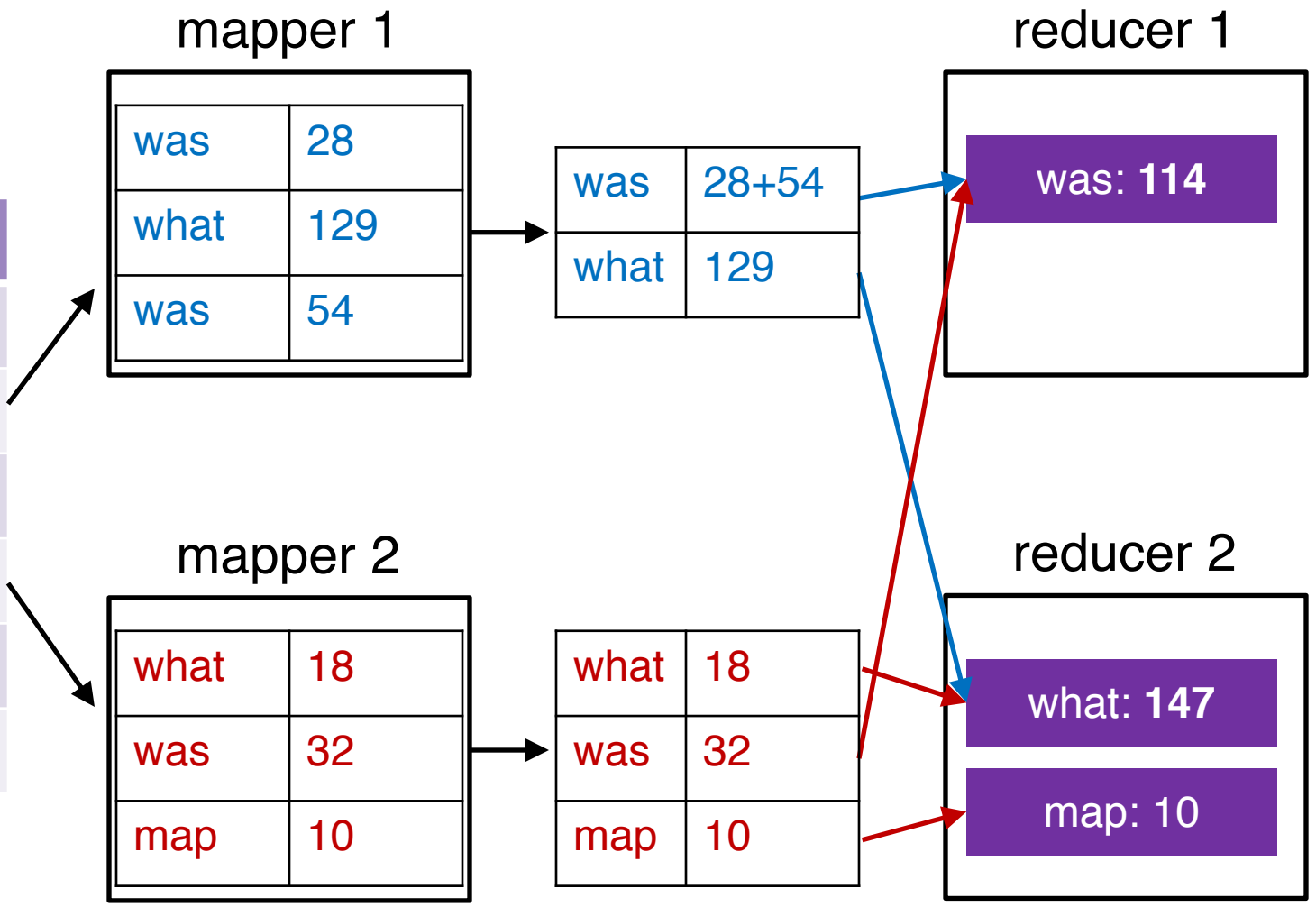
Example: Word Count

Word	Count
was	28
what	129
was	54
what	18
was	32
map	10



Example: Word Count

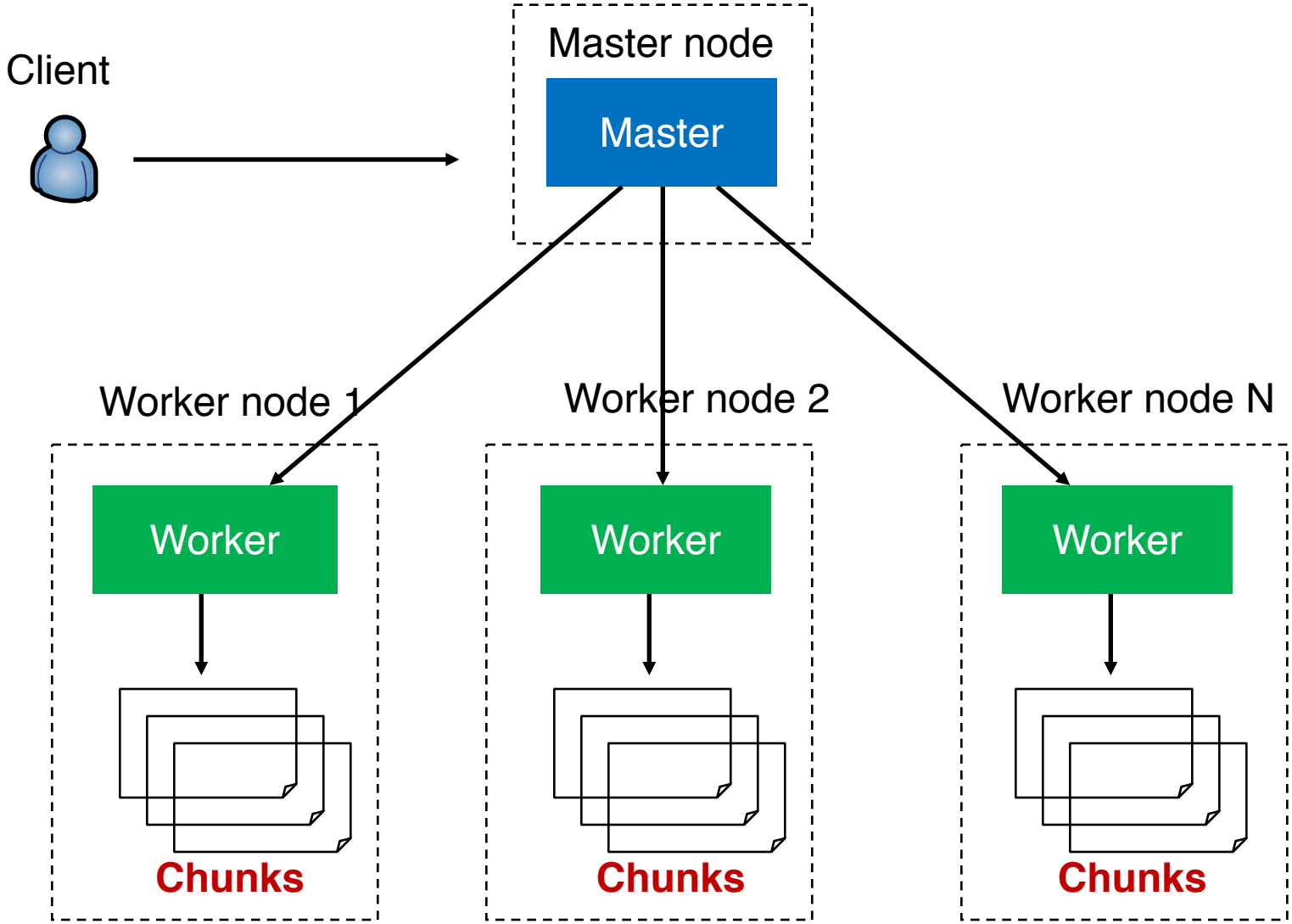
Word	Count
was	28
what	129
was	54
what	18
was	32
map	10



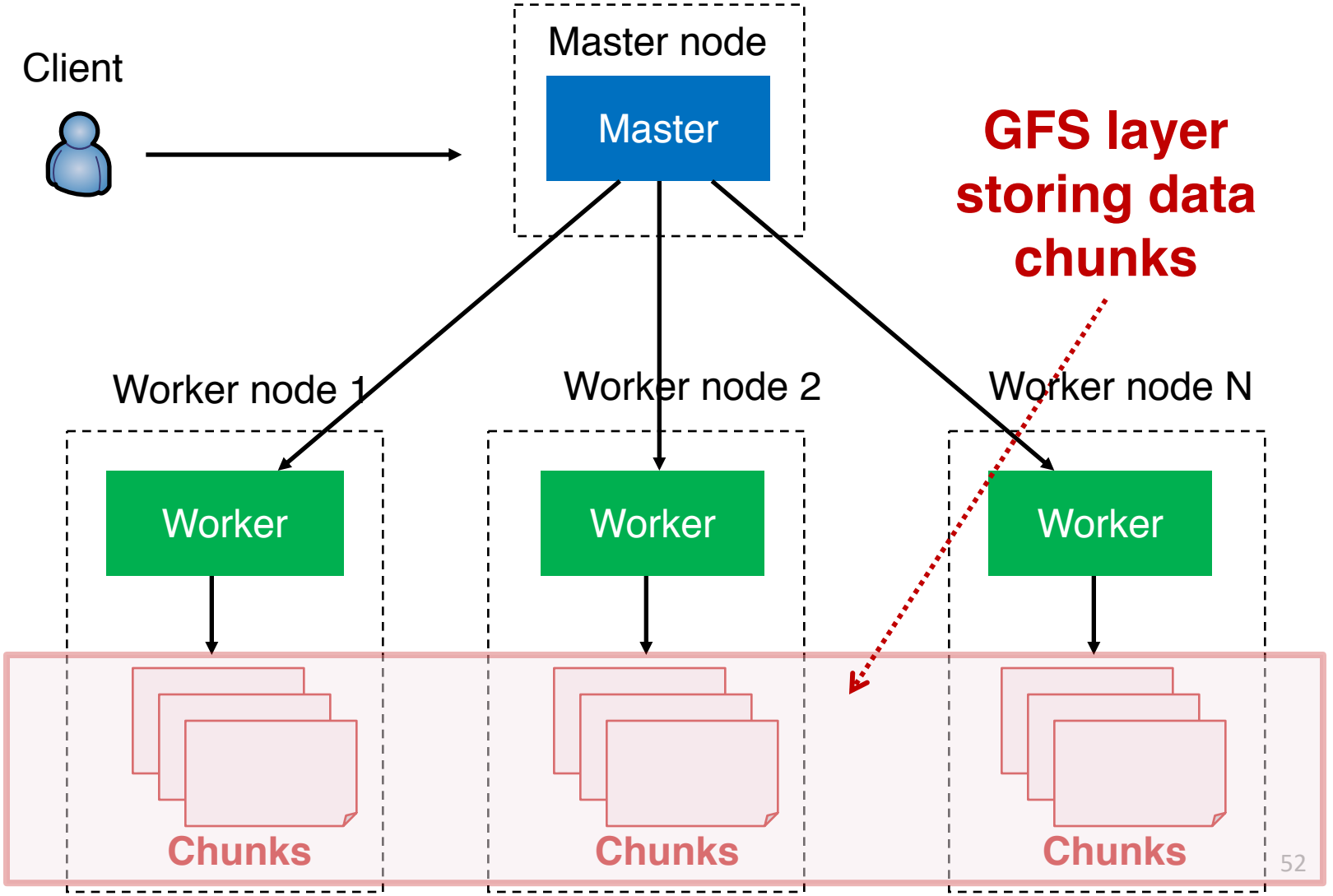
MapReduce Overview

- Motivation
- Architecture
- Programming Model

MapReduce Architecture



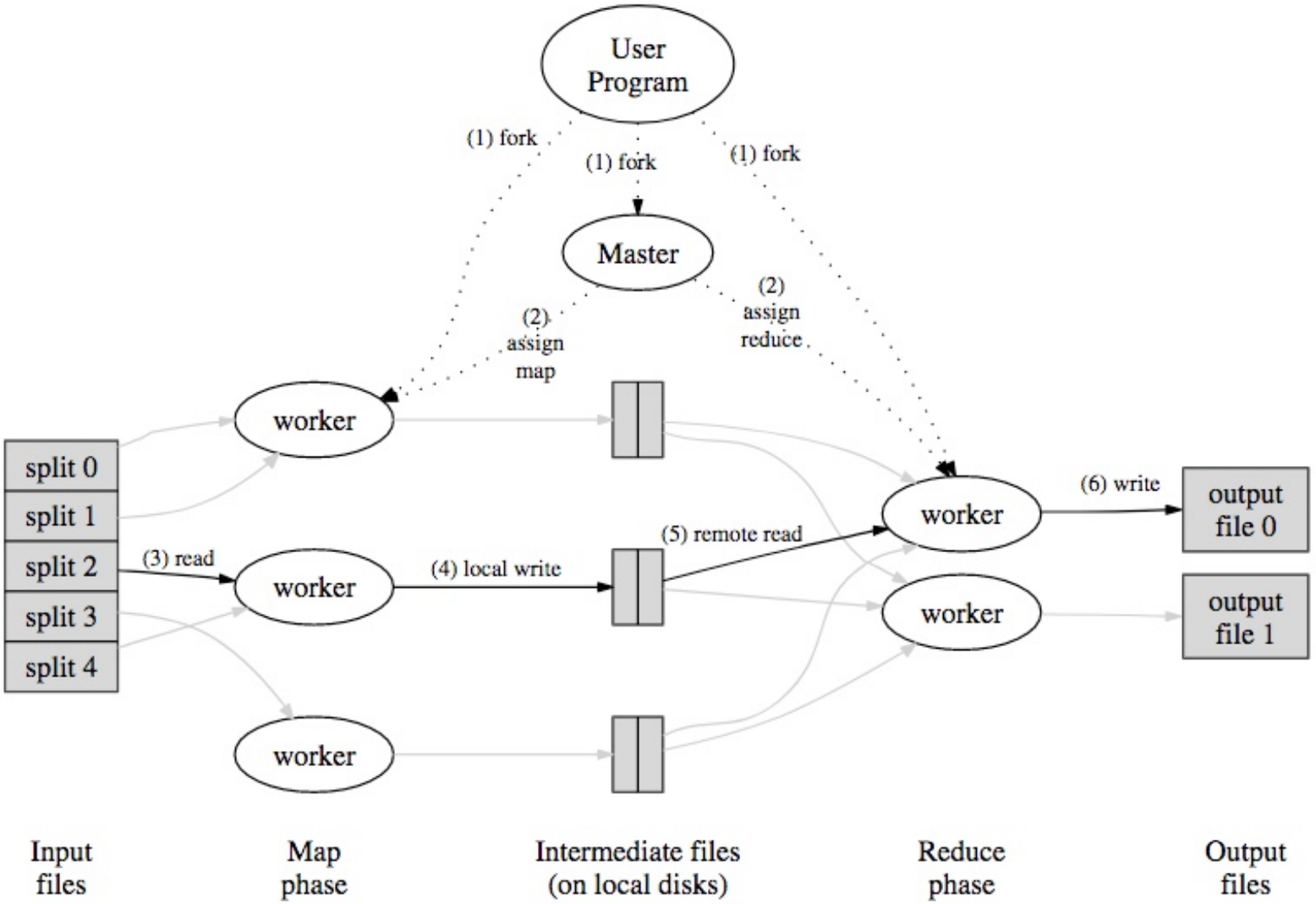
MapReduce Architecture

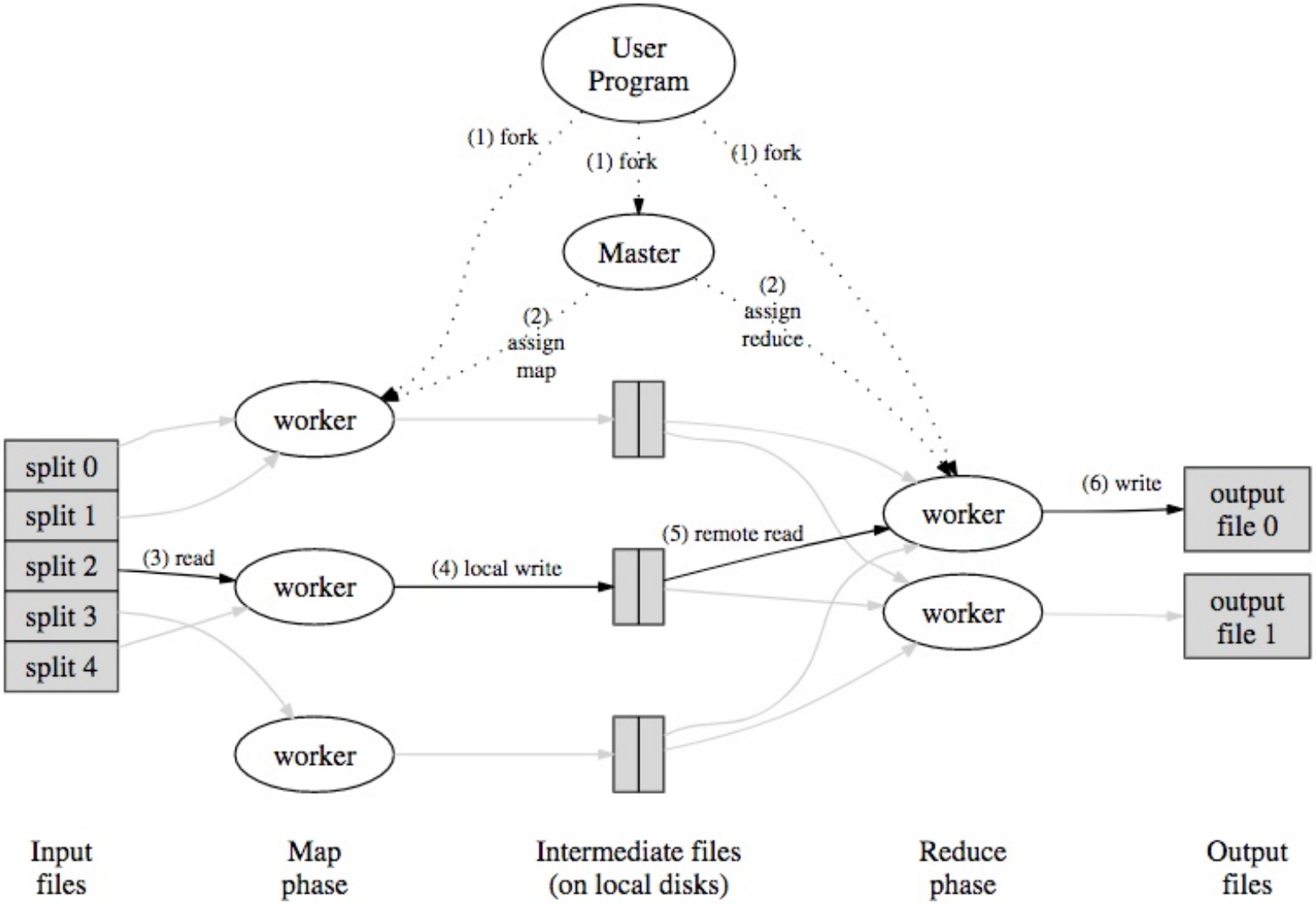


MapReduce over GFS

- MapReduce writes and reads data to/from GFS
- MapReduce workers run on same machines as GFS server daemons

MapReduce Data Flows & Executions





GFS files → Mappers → Intermediate local files → Reducers → GFS files

MapReduce Overview

- Motivation
- Architecture
- Programming Model

Map/Reduce Function Types

- map(k1, v1) → list(k2, v2)
- reduce(k2, list(v2)) → list(k3, v3)

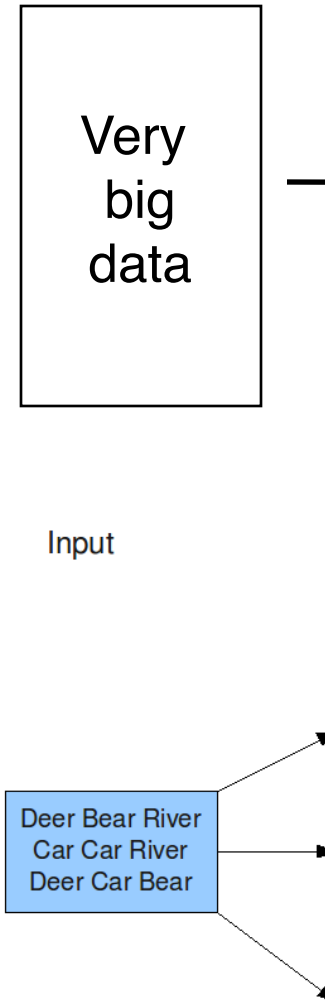
Hadoop API

```
public void map(LongWritable key, Text value) {
    // WRITE CODE HERE
}

public void reduce(Text key, Iterator<IntWritable> values)
{
    // WRITE CODE HERE
}
```

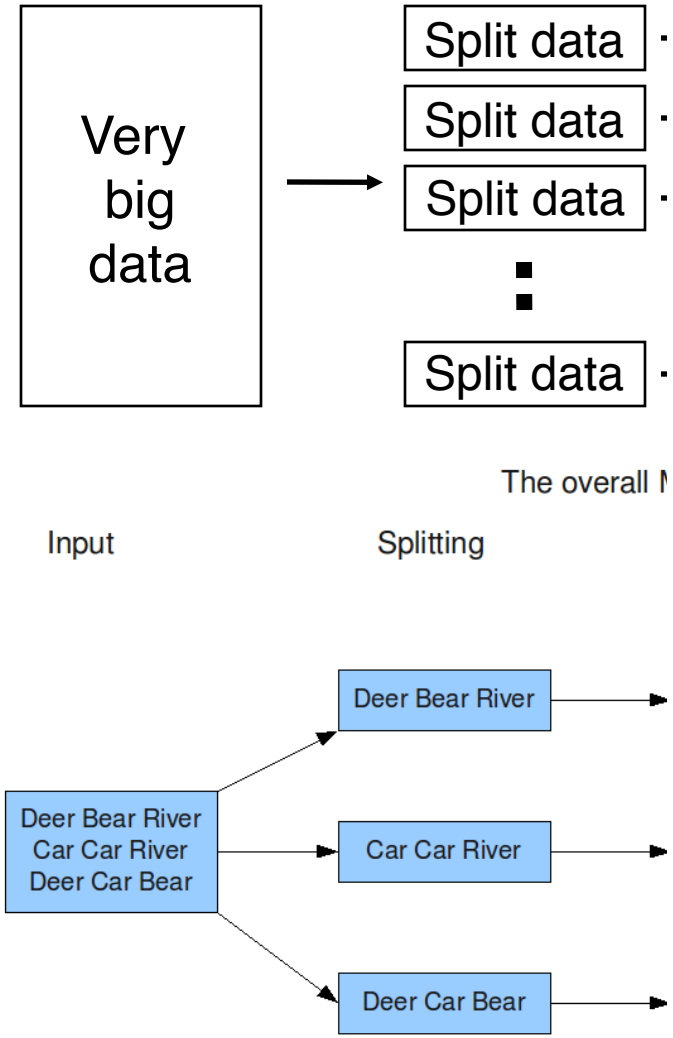
MapReduce Word Count Pseudo Code

```
func mapper(key, line) {
    for word in line.split()
        yield word, 1
}

func reducer(word, occurrences) {
    yield word, sum(occurrences)
}
```

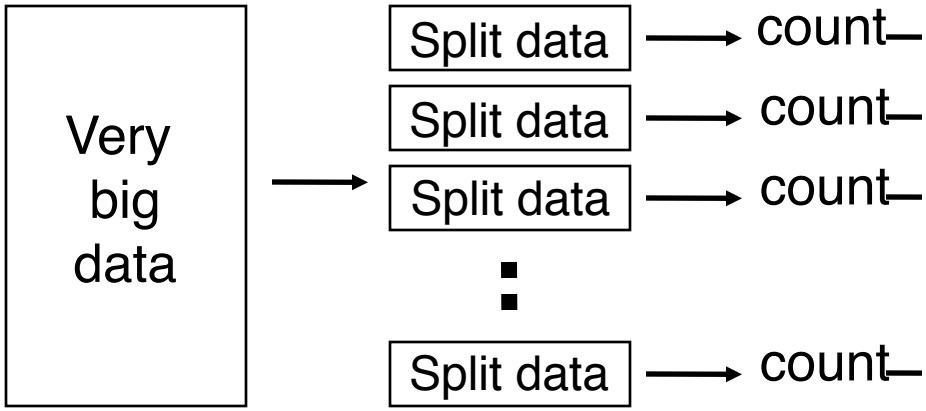
MapReduce Word Count



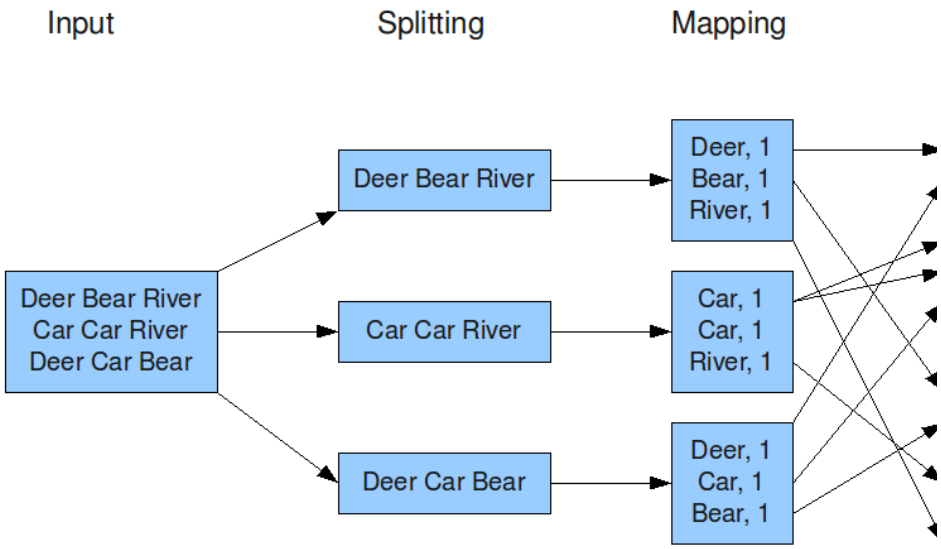
MapReduce Word Count



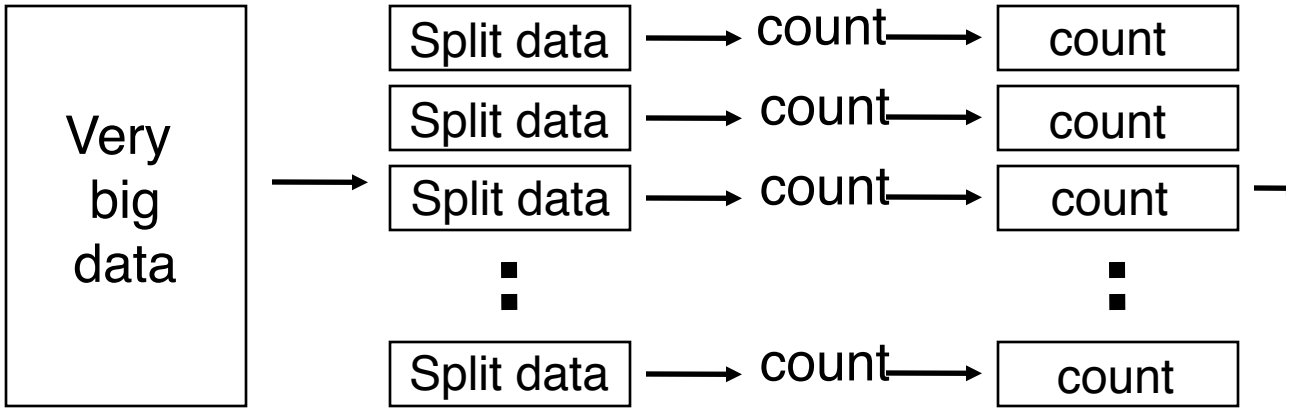
MapReduce Word Count

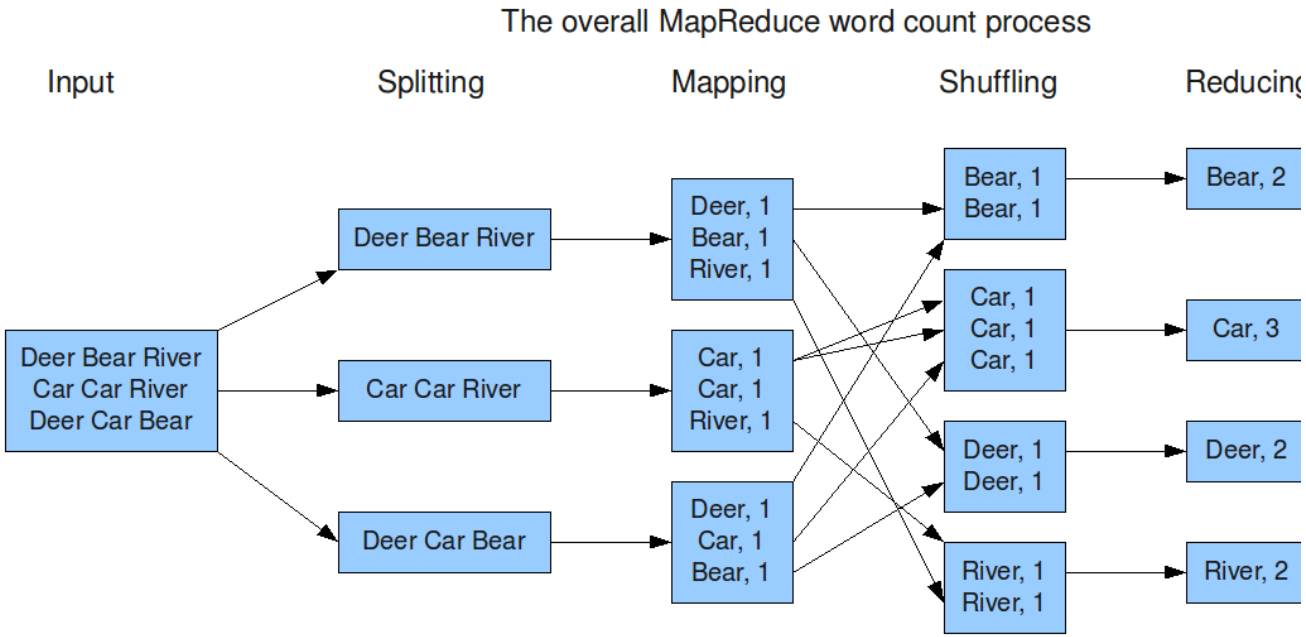


The overall MapReduce word co

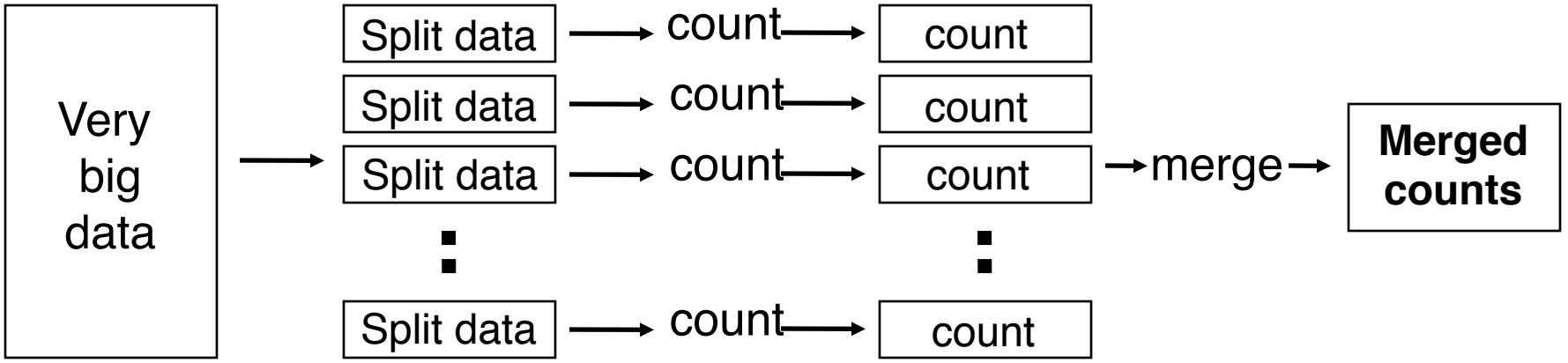


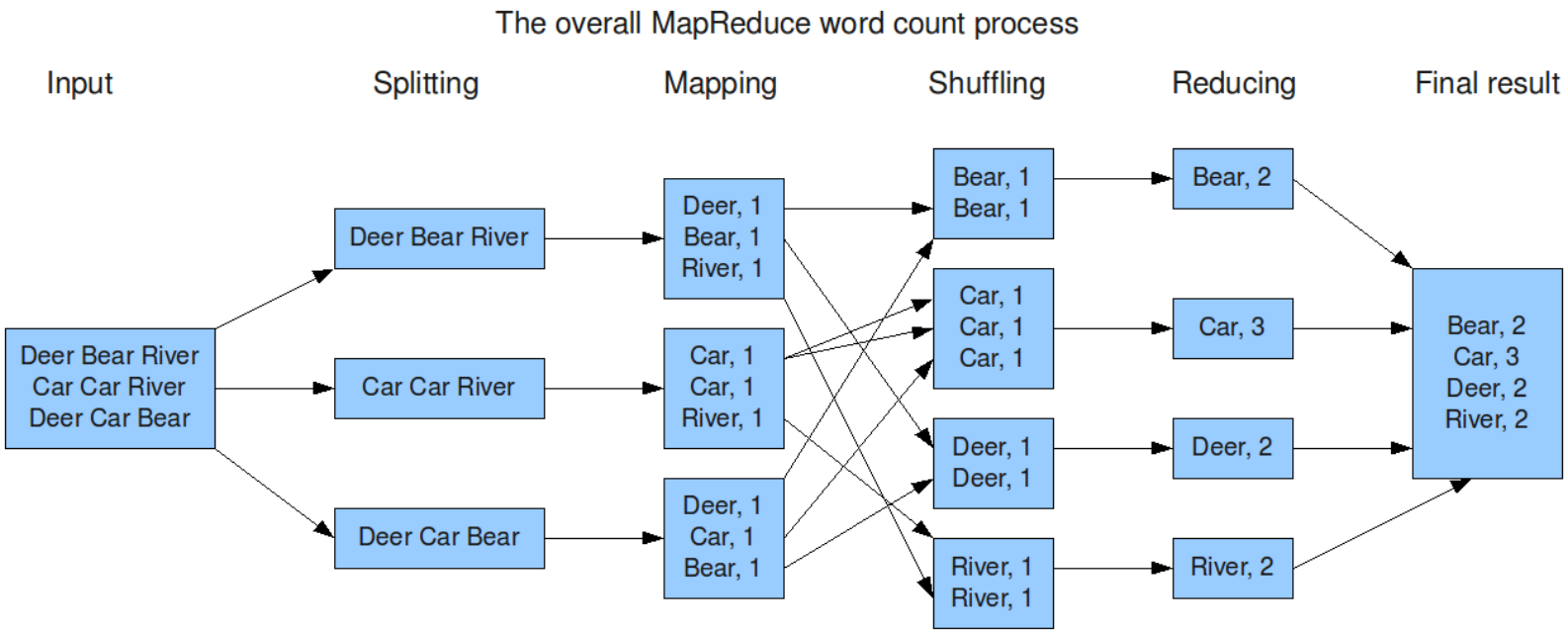
MapReduce Word Count



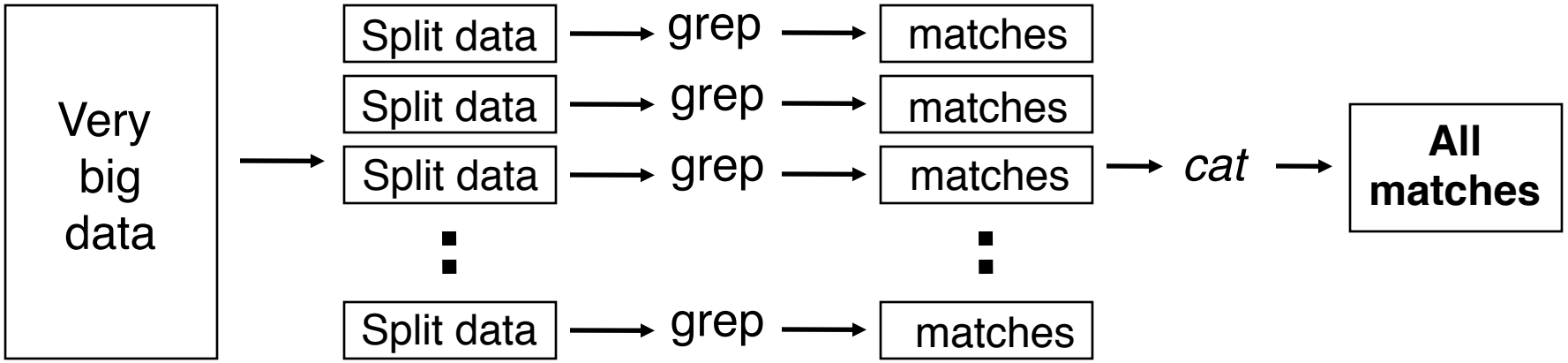


MapReduce Word Count





MapReduce Grep



Announcements

- Homework assignment 2 due mid-night this Friday (11:59pm Oct 5)

- Project milestone 1: mid-term proposal presentation
  - Oct 17
  - Proposal report due Oct 26