The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google∗

Presenter: Xinyi Ye

November 17, 2021

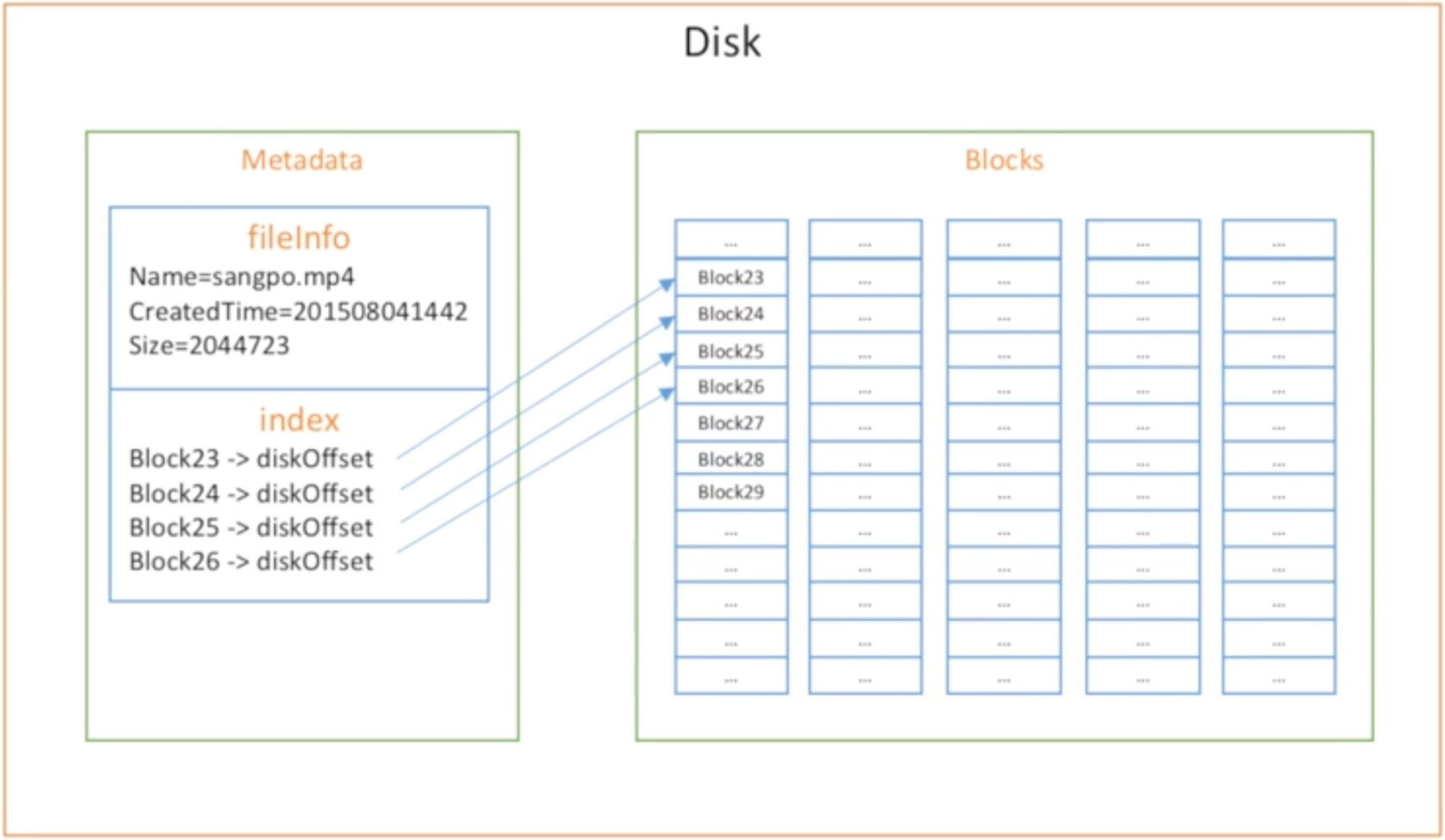
What is GFS?

- A scalable, fault-tolerant distributed file system

Before designing…

- Assumptions
  - The system is built from many inexpensive commodity components (component failures are the norm)
  - Files are huge
  - Files are write-once, mostly appended to
  - Large streaming reads
  - High sustained bandwidth is more important than low latency

Regular file system



Disk

Metadata

fileInfo

Name=sangpo.mp4

CreatedTime=201508041442

Size=2044723

index

Block23 -> diskOffset

Block24 -> diskOffset

Block25 -> diskOffset

Block26 -> diskOffset

Blocks

Block23

Block24

Block25

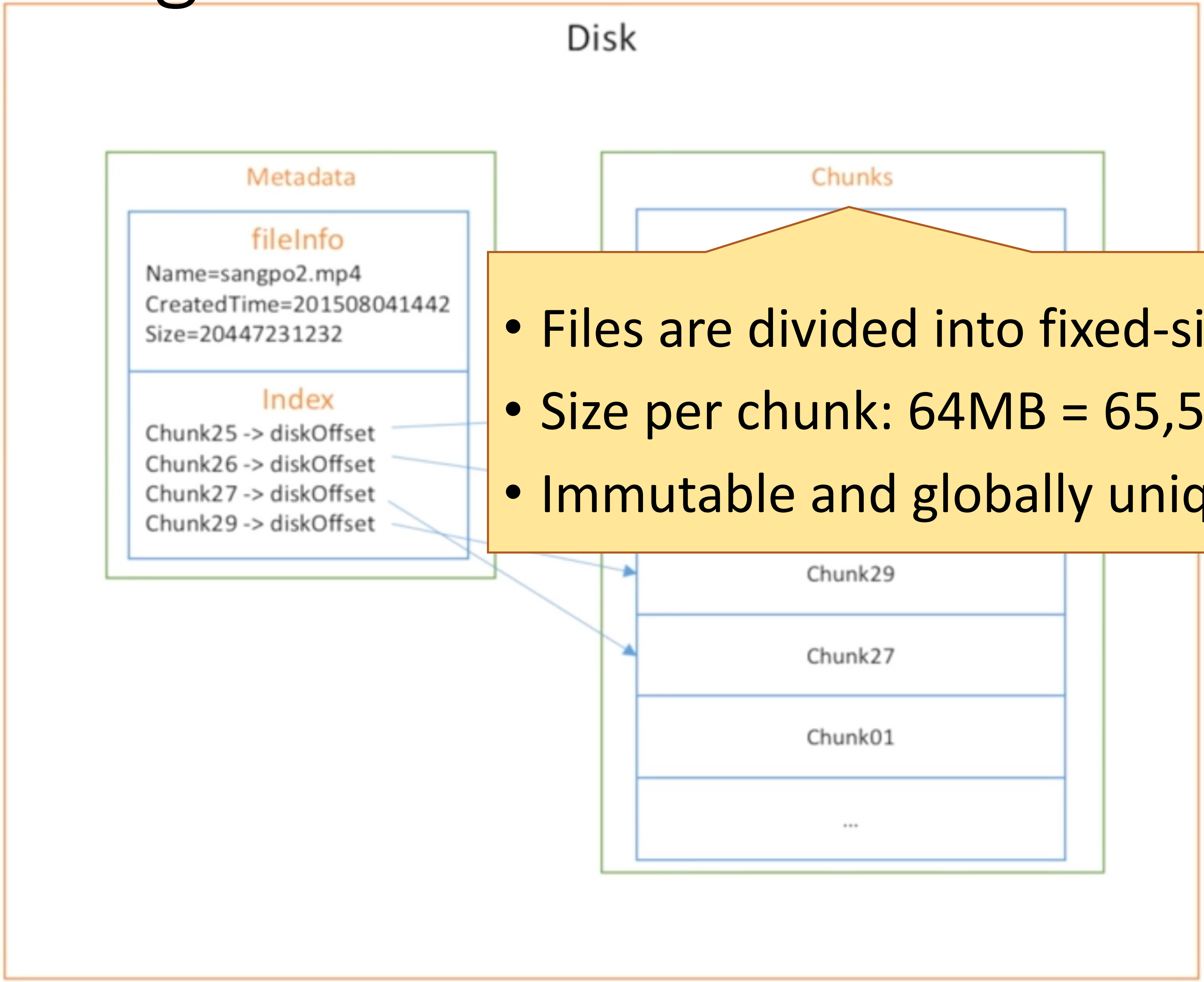
Block26

Block27

Block28

Block29

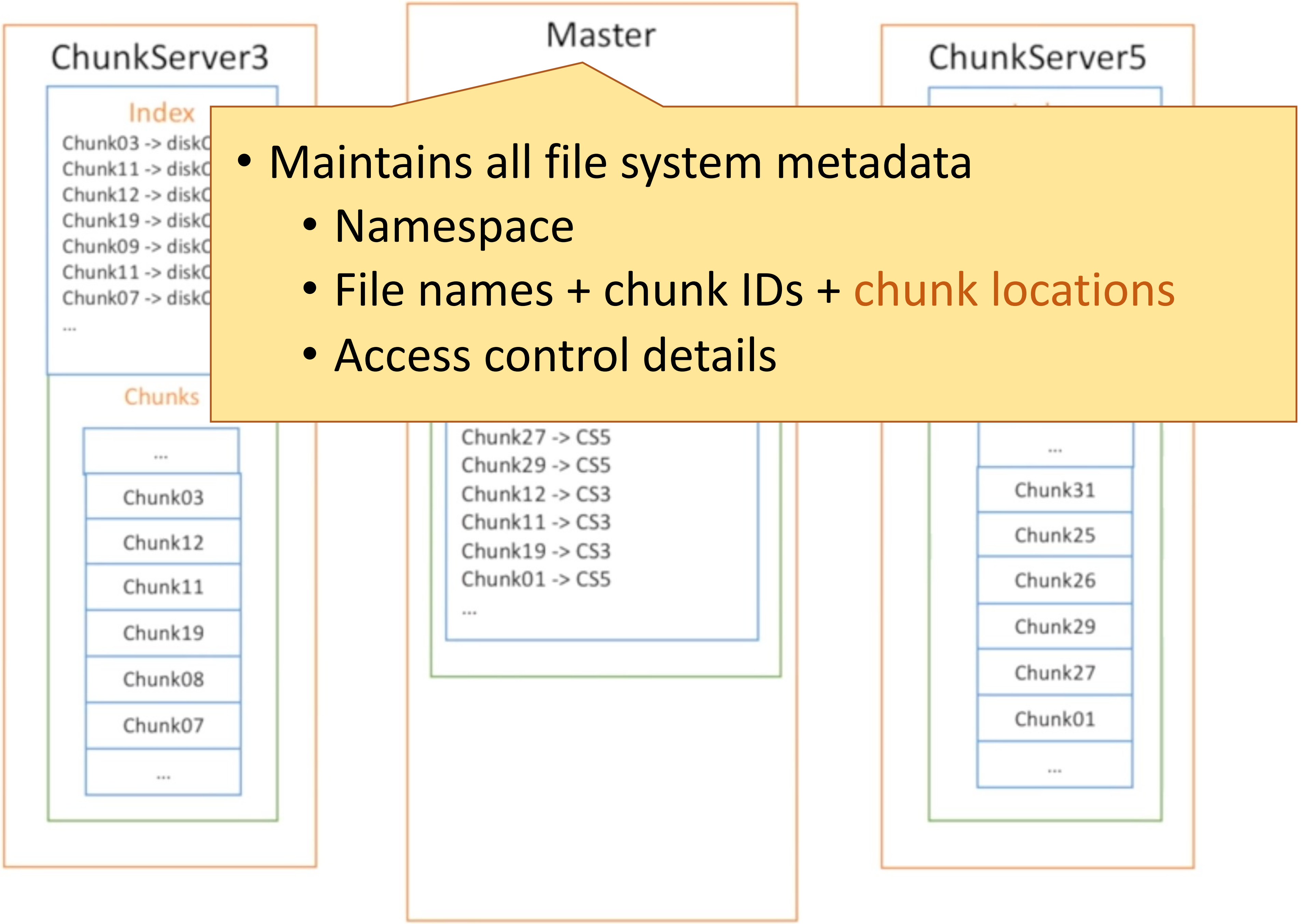
Larger files



- Files are divided into fixed-size chunks
- Size per chunk: 64MB = 65,536 blocks
- Immutable and globally unique 64-bit *chunk handle*

- Advantage
  - Reduce clients' interaction with
  - Reduce size of the metadata stored on the master

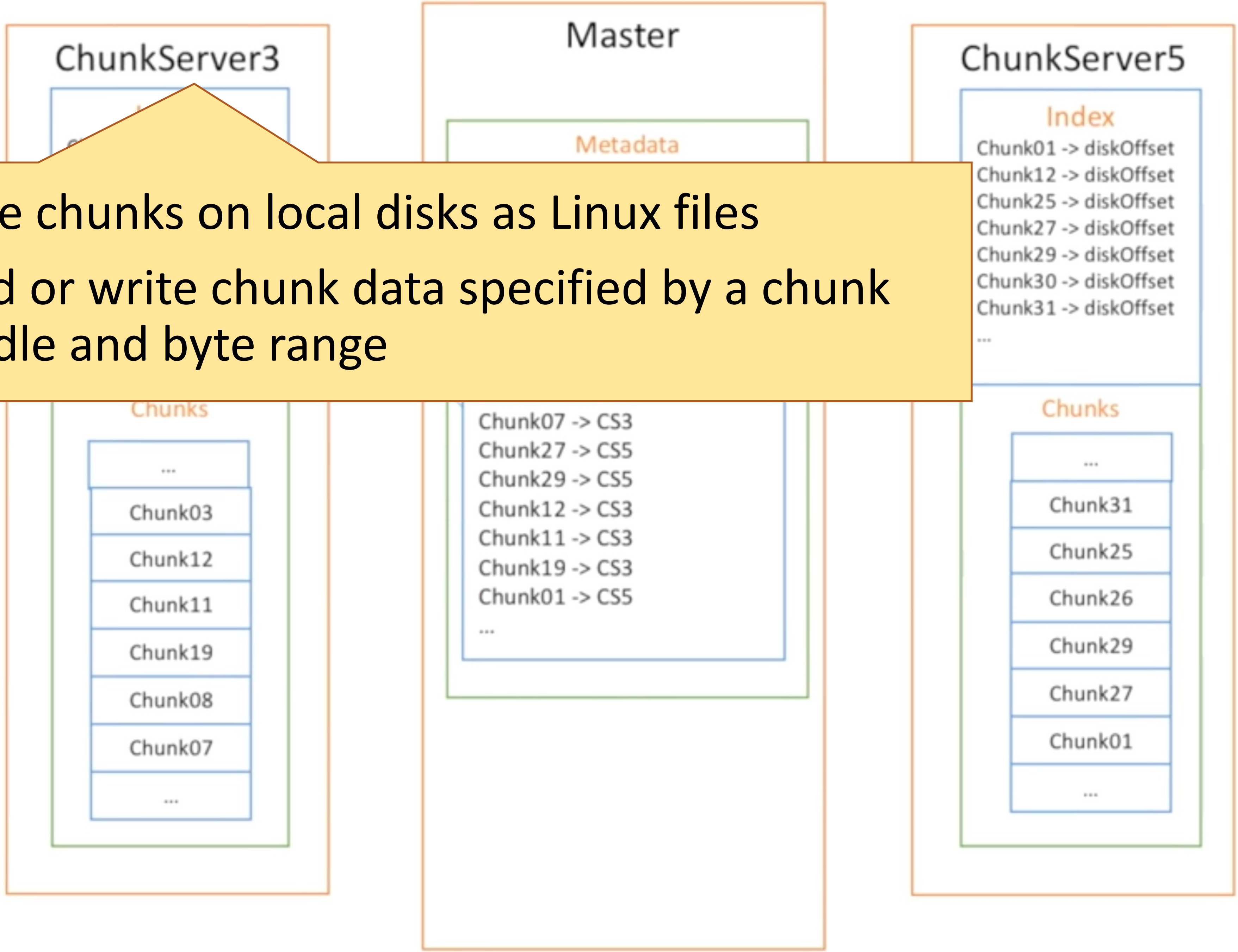
GFS



- Maintains all file system metadata
  - Namespace
  - File names + chunk IDs + chunk locations
  - Access control details

GFS

- Store chunks on local disks as Linux files
- Read or write chunk data specified by a chunk handle and byte range



ChunkServer3

Chunks

...
Chunk03
Chunk12
Chunk11
Chunk19
Chunk08
Chunk07
...

Master

Metadata

Chunk07 -> CS3
Chunk27 -> CS5
Chunk29 -> CS5
Chunk12 -> CS3
Chunk11 -> CS3
Chunk19 -> CS3
Chunk01 -> CS5
...

ChunkServer5

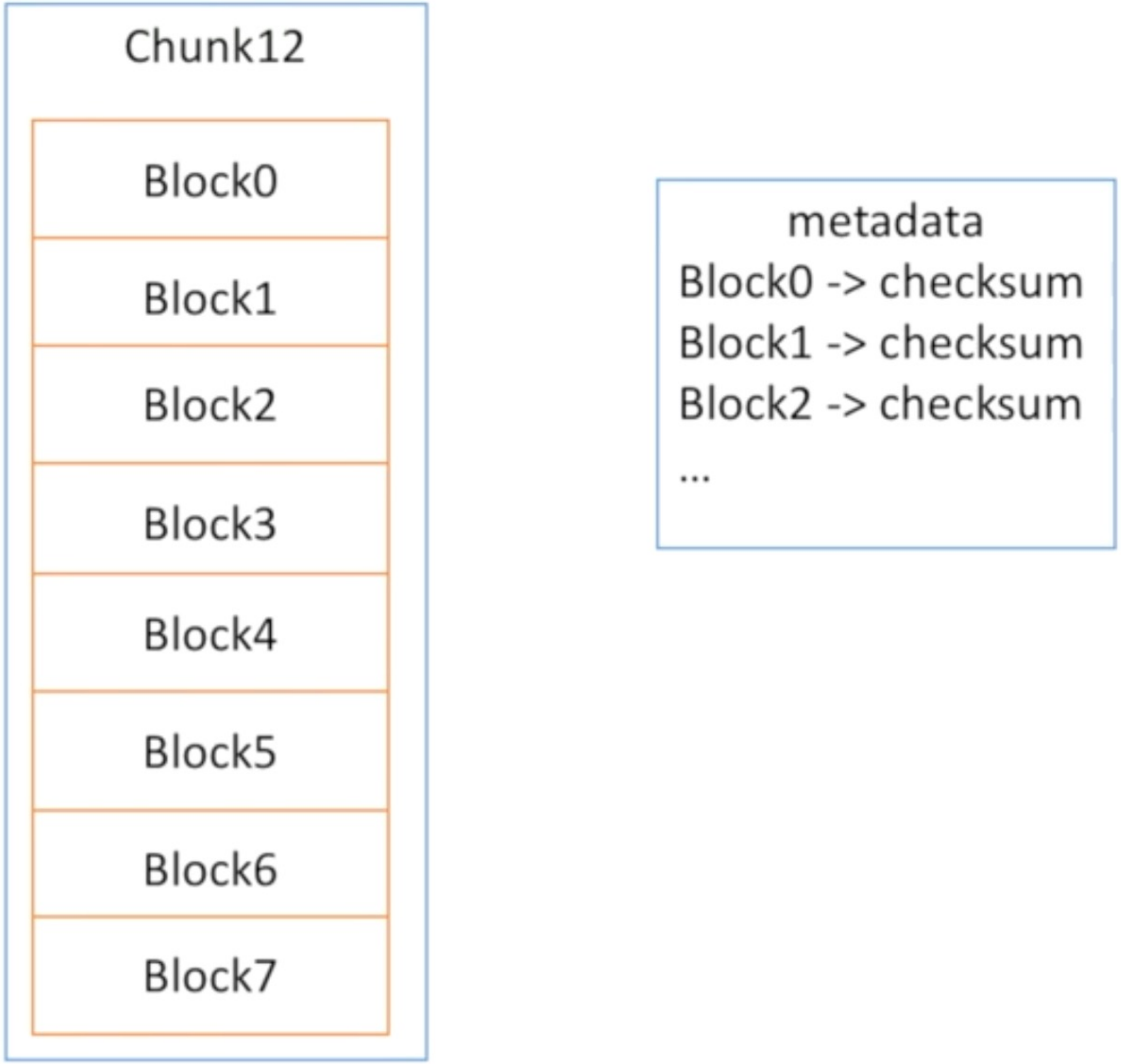
Index

Chunk01 -> diskOffset
Chunk12 -> diskOffset
Chunk25 -> diskOffset
Chunk27 -> diskOffset
Chunk29 -> diskOffset
Chunk30 -> diskOffset
Chunk31 -> diskOffset
...

Chunks

...
Chunk31
Chunk25
Chunk26
Chunk29
Chunk27
Chunk01
...

Checksum



Chunk12

Block0

Block1

Block2

Block3

Block4

Block5

Block6

Block7

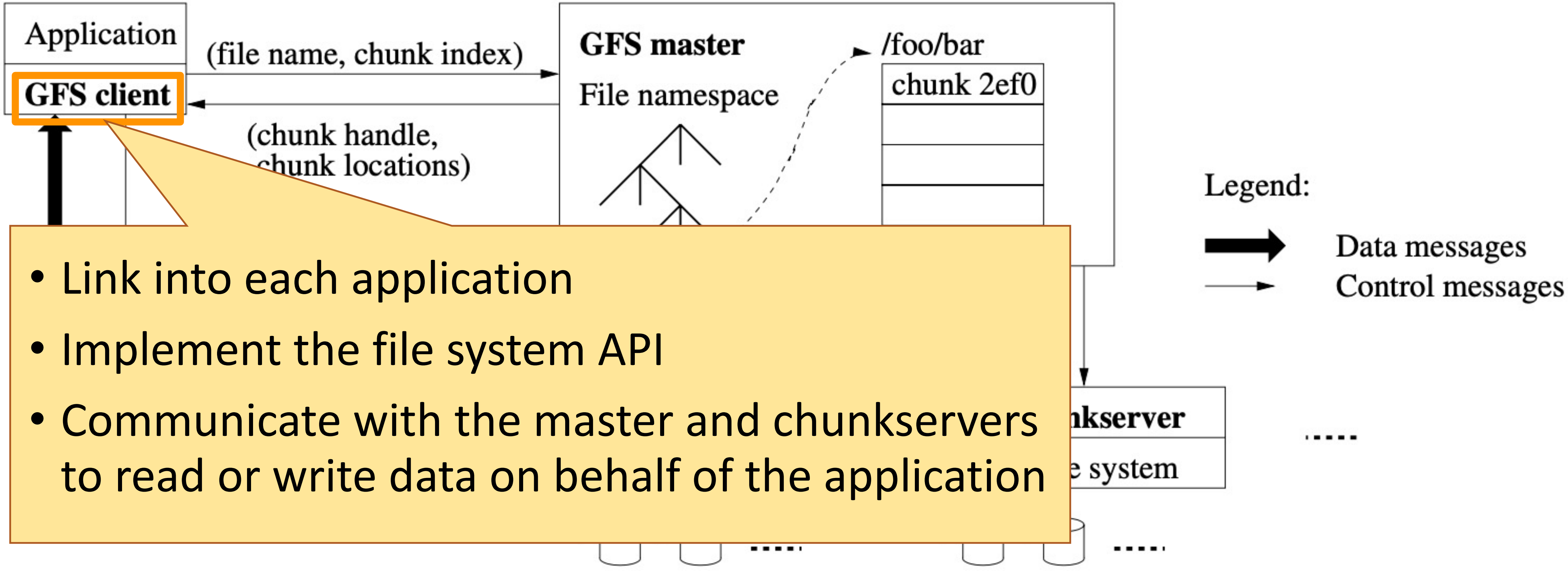
metadata
Block0 -> checksum
Block1 -> checksum
Block2 -> checksum
...

- 1 block = 64KB
- 1 checksum = 32bit
- Check checksum when reading

Operation log

- Used if master crashes
- Checkpointed regularly
- Rebooted master replays log

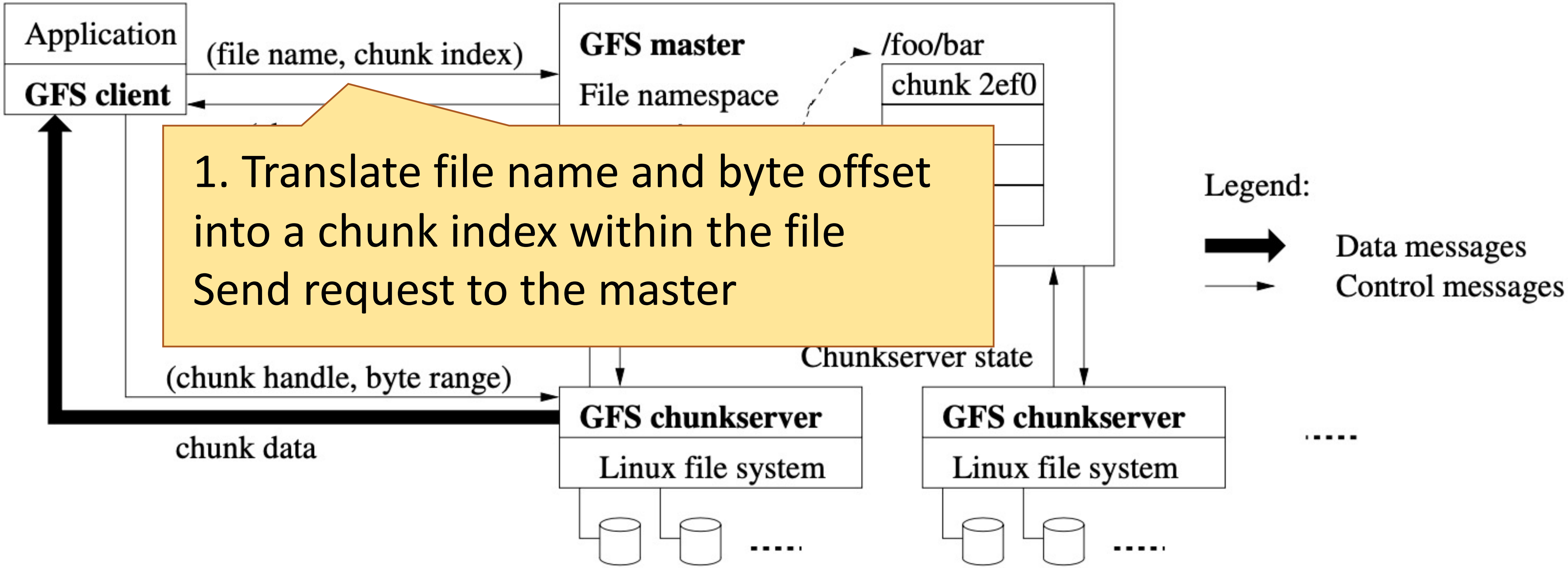
Architecture



- Link into each application
- Implement the file system API
- Communicate with the master and chunkservers to read or write data on behalf of the application

Figure 1: GFS Architecture

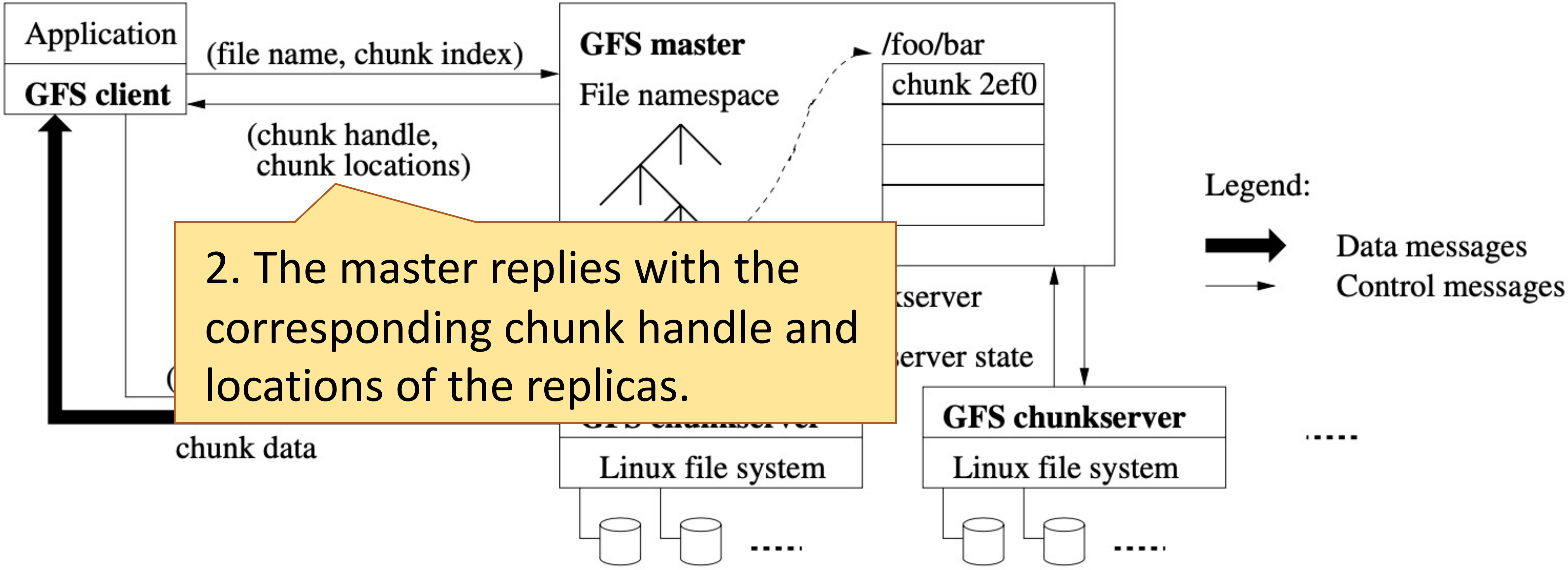
Interactions for a simple read



1. Translate file name and byte offset into a chunk index within the file
Send request to the master

Figure 1: GFS Architecture

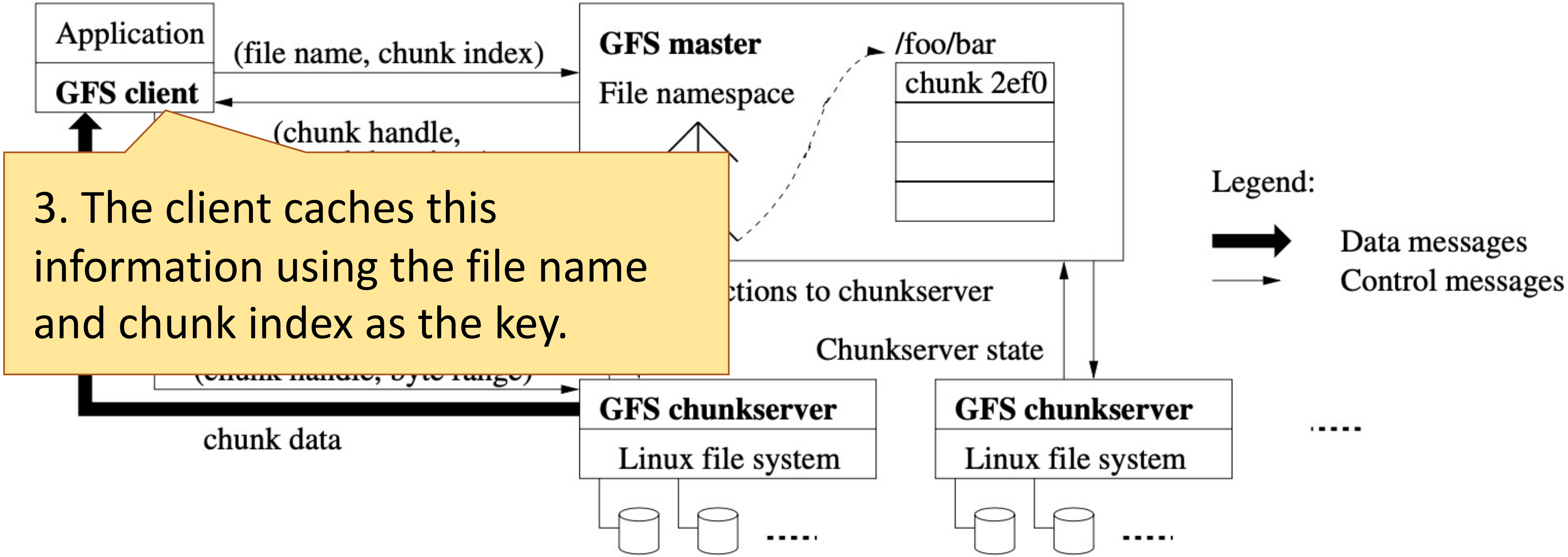
Interactions for a simple read



2. The master replies with the corresponding chunk handle and locations of the replicas.

Figure 1: GFS Architecture

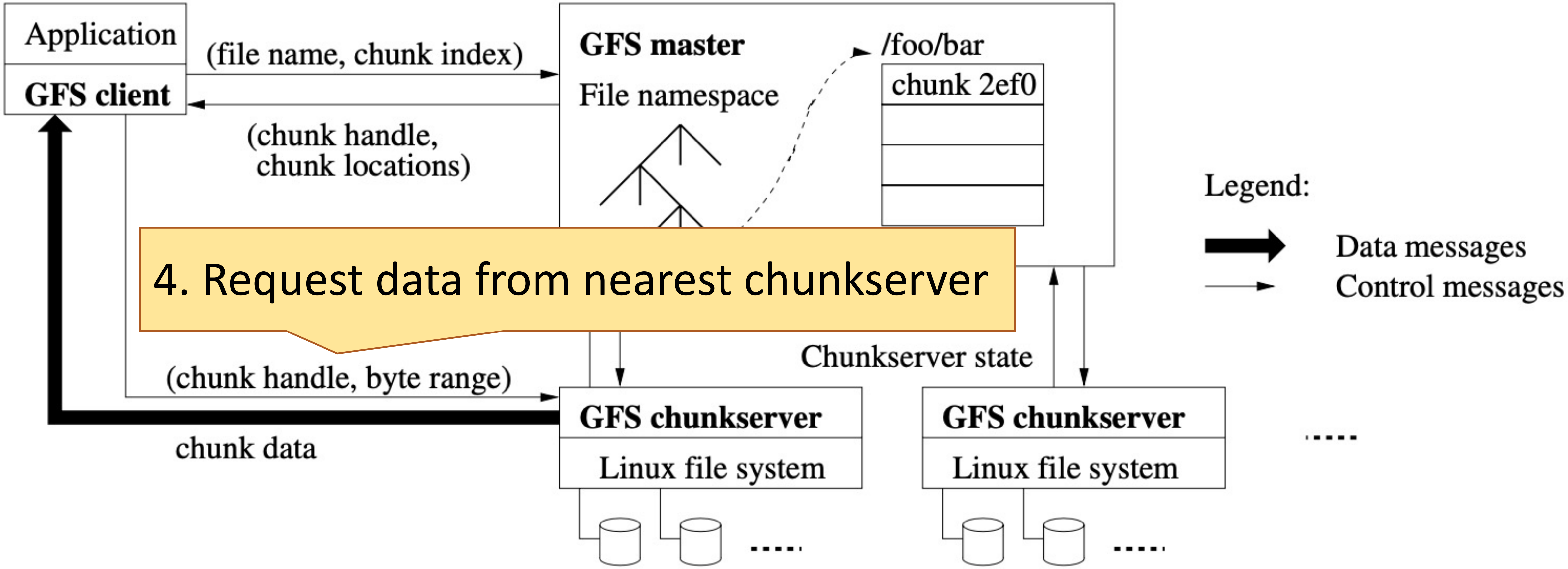
Interactions for a simple read



3. The client caches this information using the file name and chunk index as the key.

Figure 1: GFS Architecture

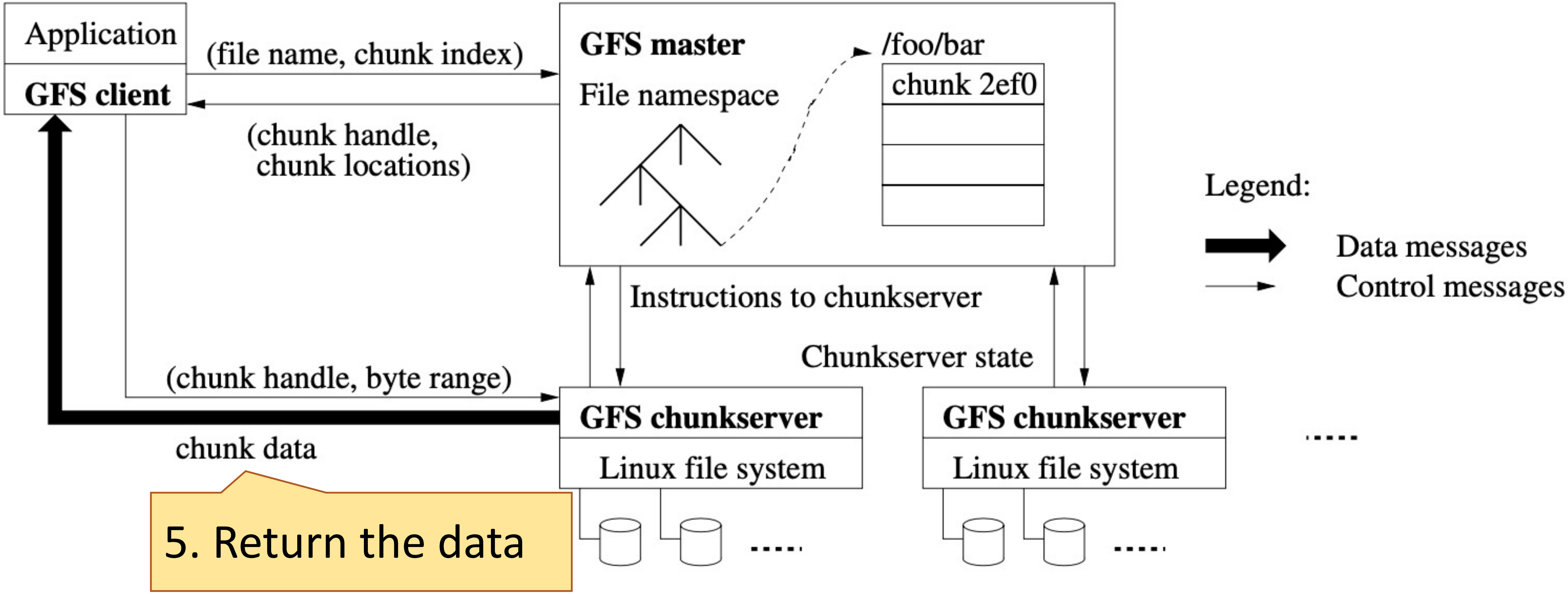
Interactions for a simple read



4. Request data from nearest chunkserver

Figure 1: GFS Architecture

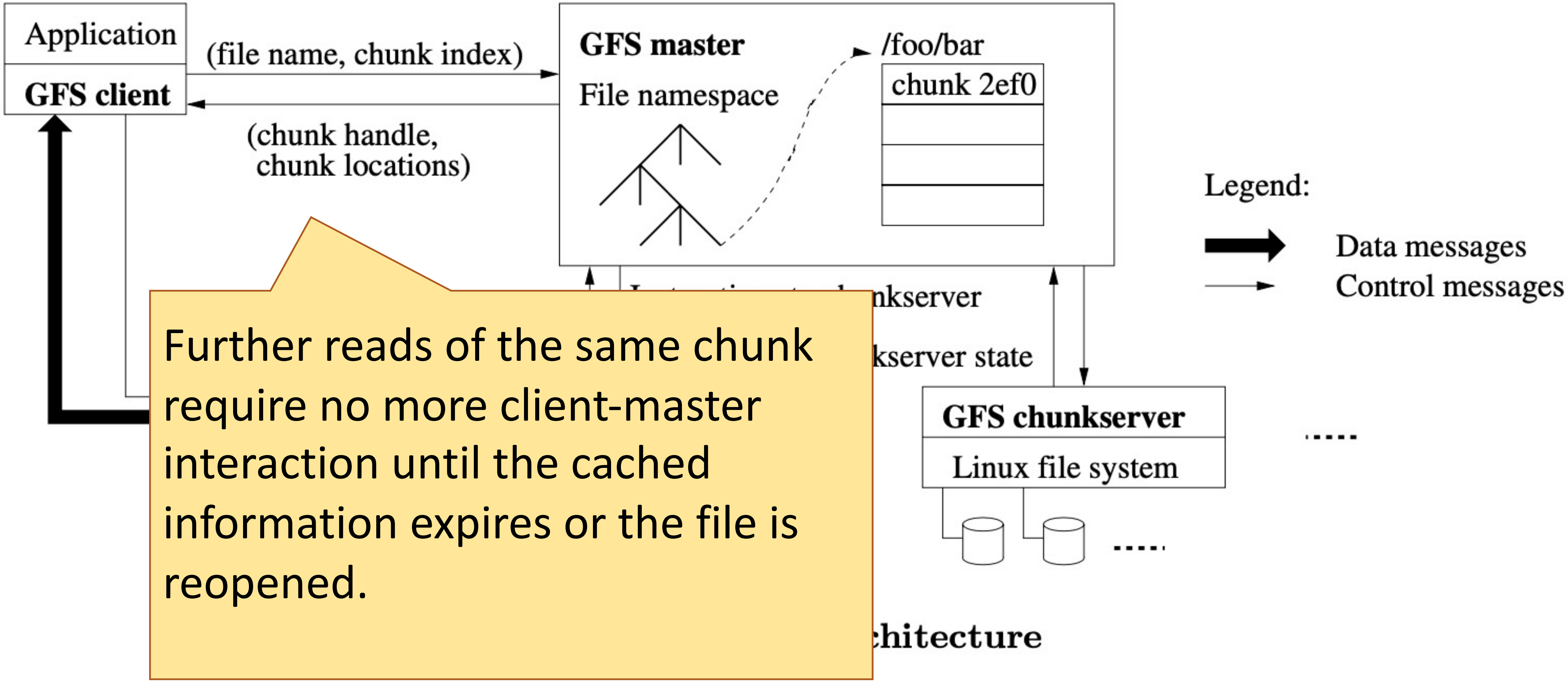
Interactions for a simple read



5. Return the data

Figure 1: GFS Architecture

Interactions for a simple read



Further reads of the same chunk require no more client-master interaction until the cached information expires or the file is reopened.

Interactions for a simple read

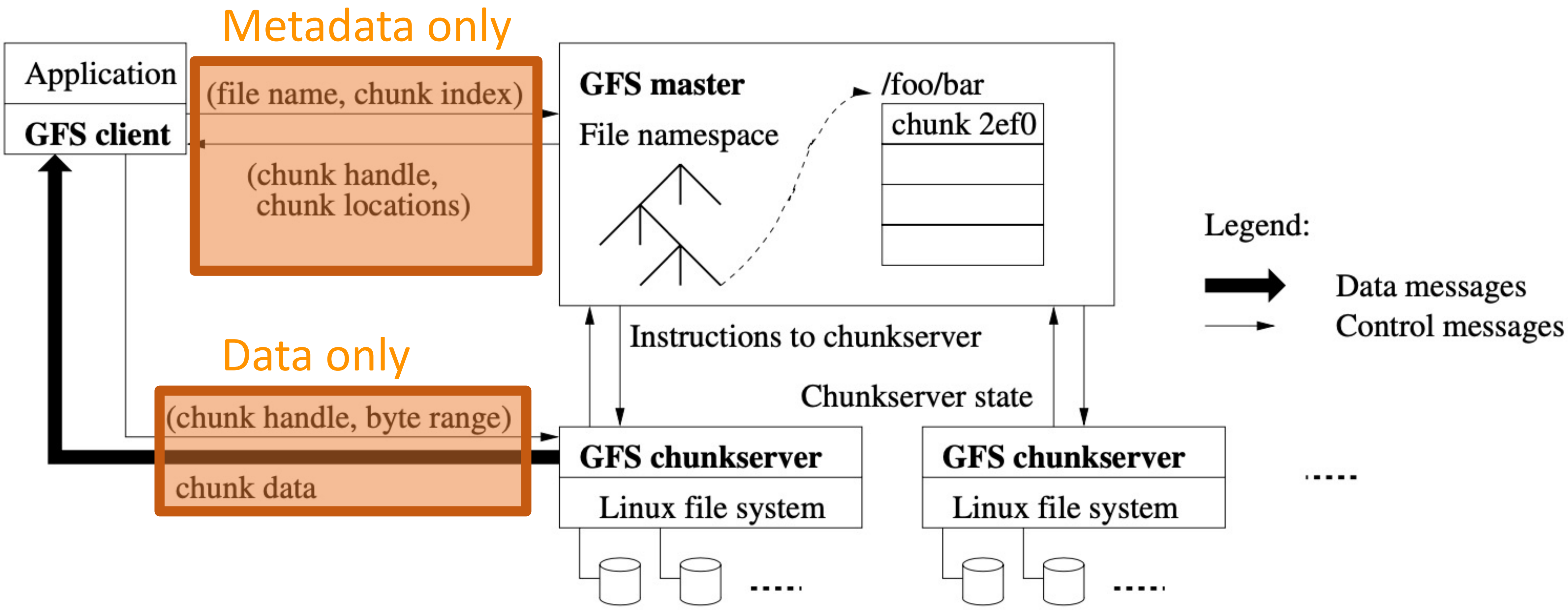
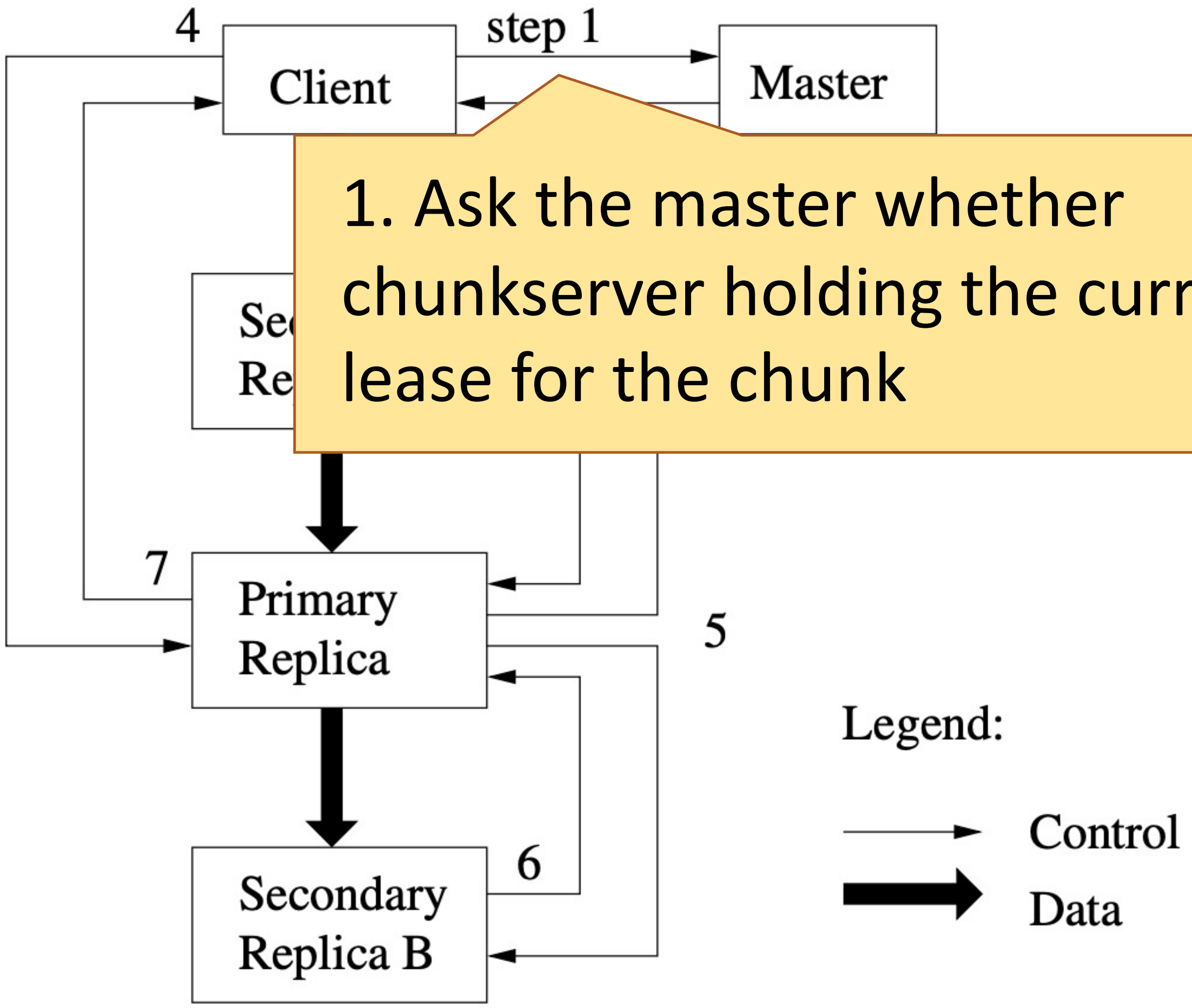


Figure 1: GFS Architecture

Leases and Mutation Order

- Mutation: An operation that changes the contents or metadata of a chunk (e.g., write or an append operation)
- Master grants a chunk lease to one replica (called *primary*)
- Primary picks a serial order for all mutations to the chunk

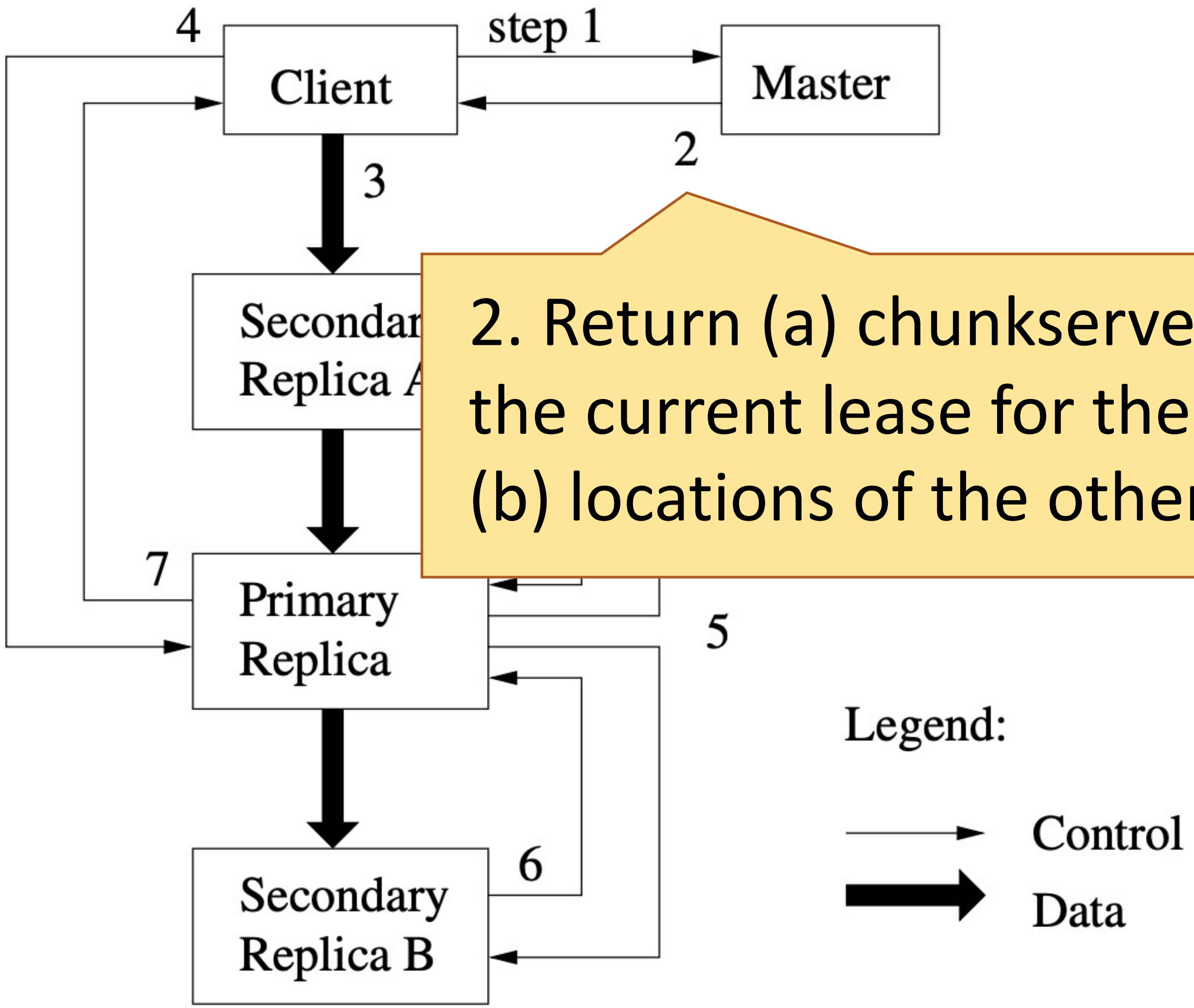
Interactions for a simple write



1. Ask the master whether chunkserver holding the current lease for the chunk

Figure 2: Write Control and Data Flow

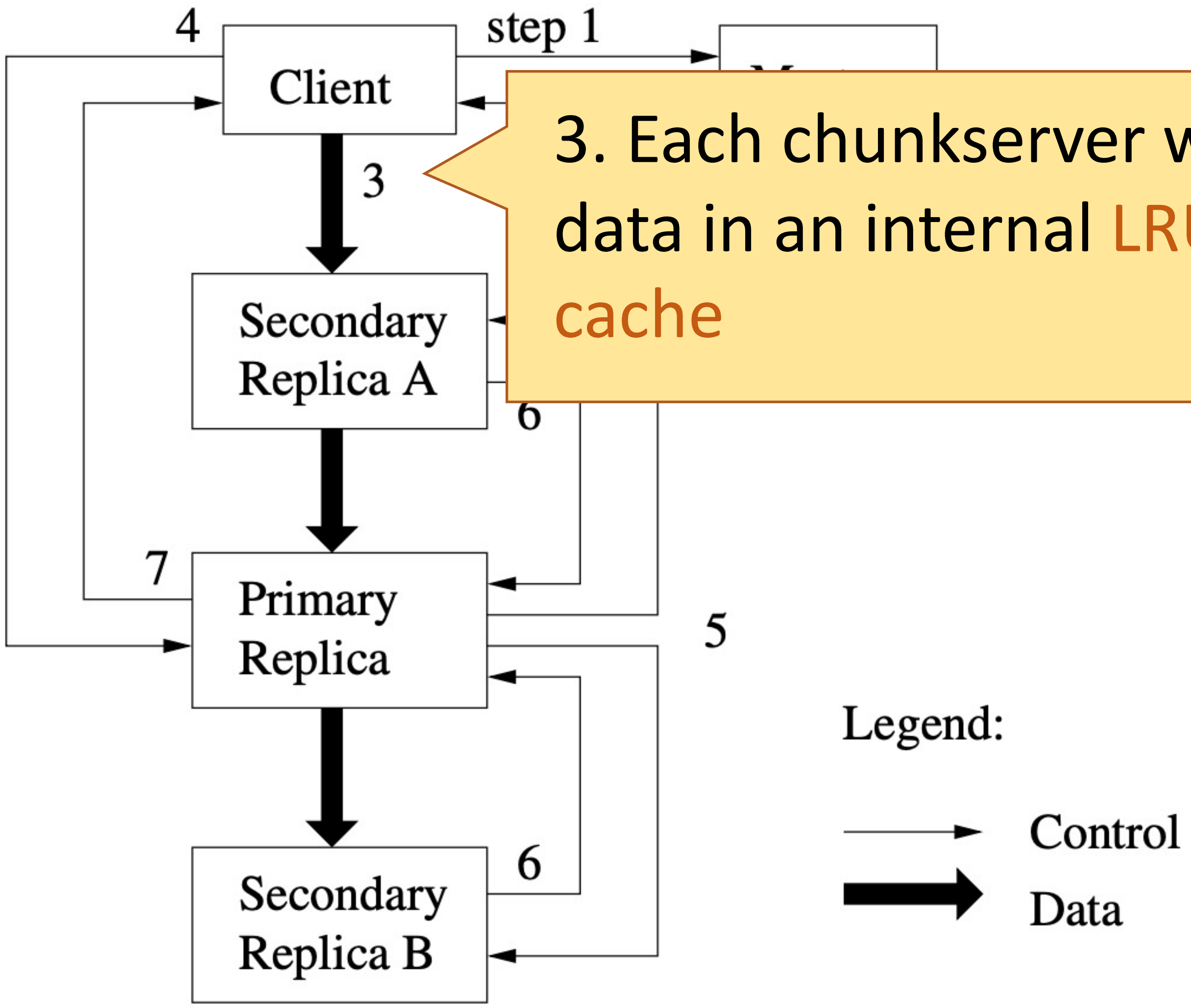
Interactions for a simple write



2. Return (a) chunkserver holding the current lease for the chunk (b) locations of the other replicas

Figure 2: Write Control and Data Flow

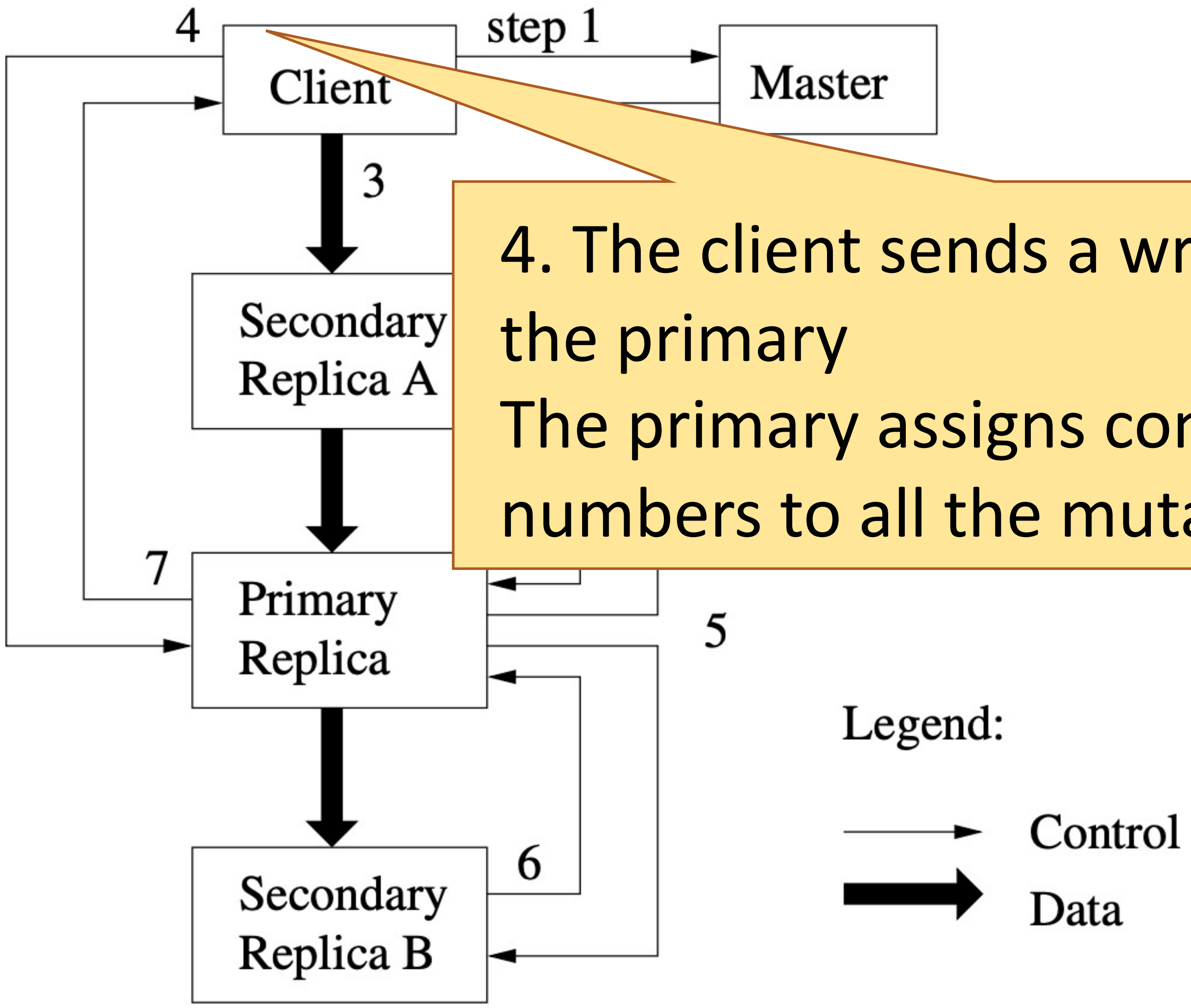
Interactions for a simple write



3. Each chunkserver will store the data in an internal LRU buffer cache

Figure 2: Write Control and Data Flow

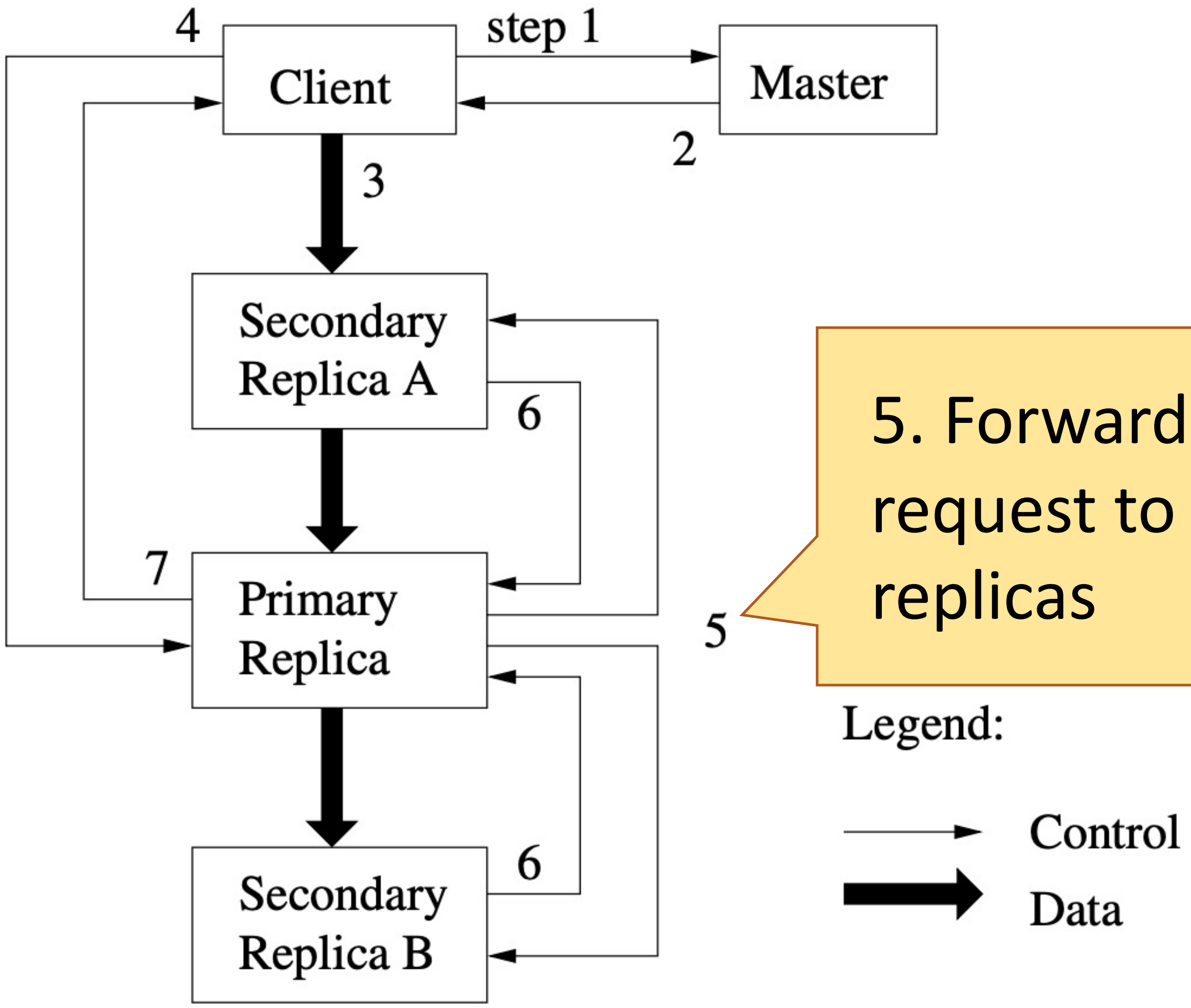
Interactions for a simple write



4. The client sends a write request to the primary
The primary assigns consecutive serial numbers to all the mutations

Figure 2: Write Control and Data Flow

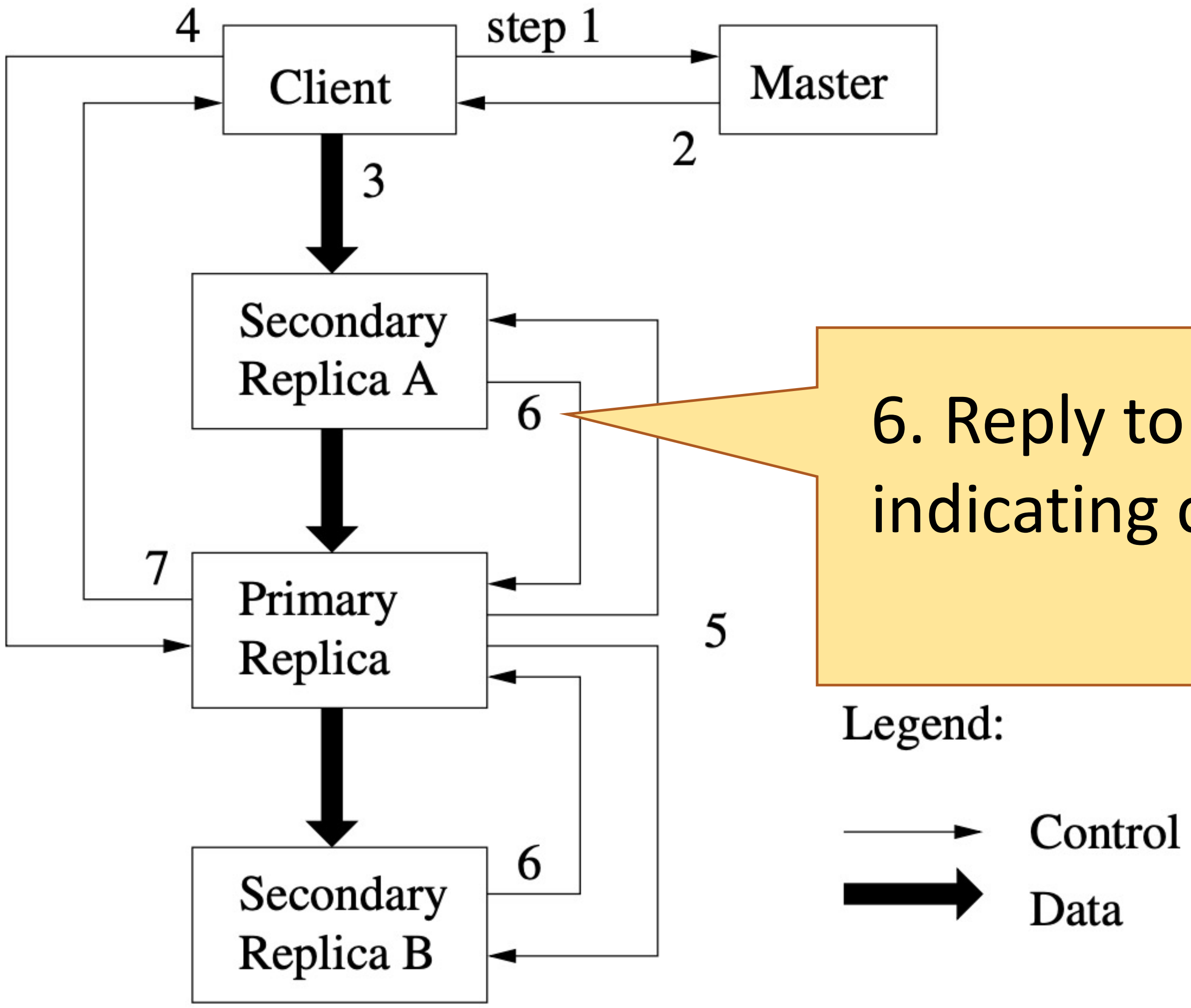
Interactions for a simple write



5. Forward the write request to all secondary replicas

Figure 2: Write Control and Data Flow

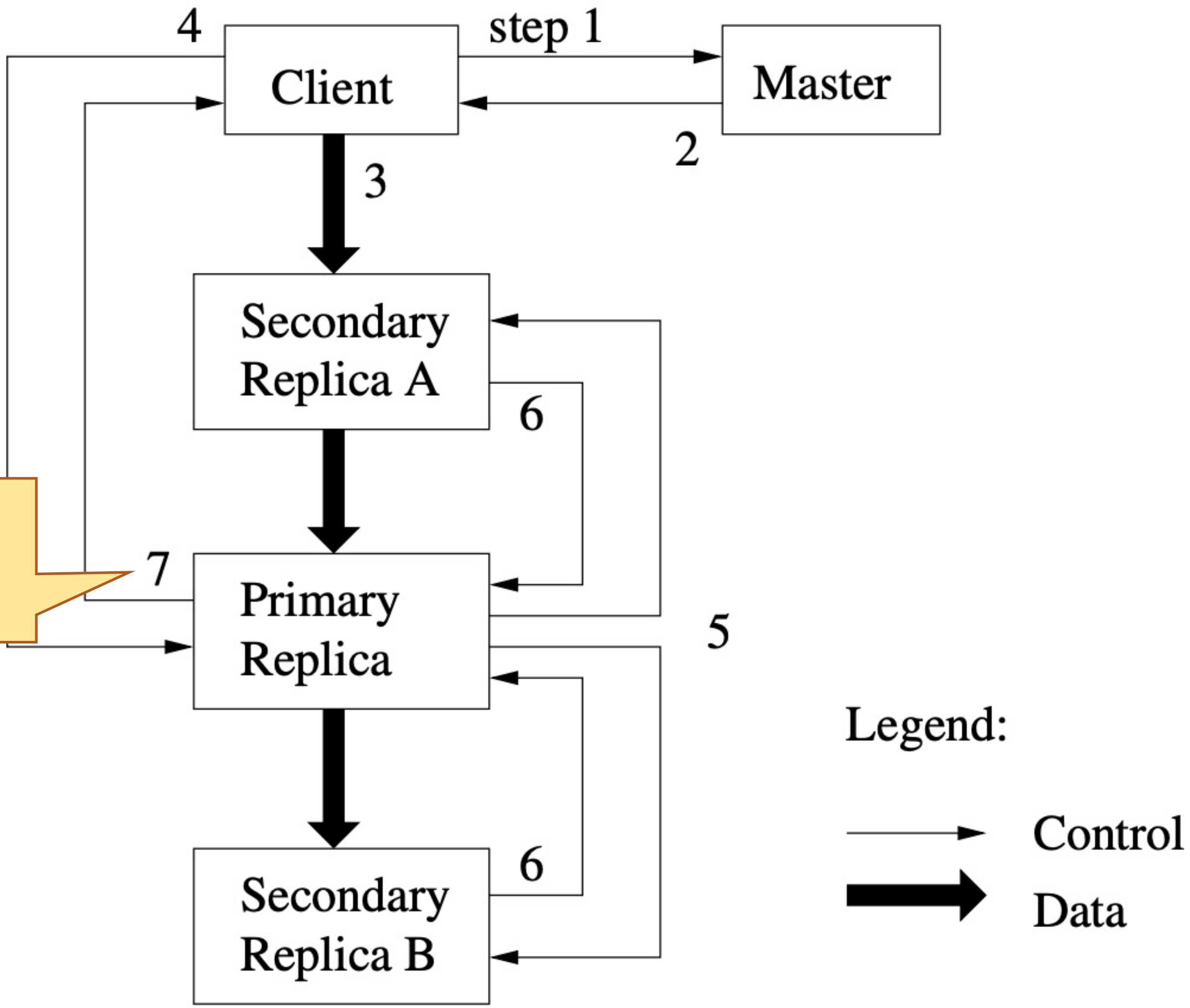
Interactions for a simple write



6. Reply to the primary indicating completing

Figure 2: Write Control and Data Flow

Interactions for a simple write



7. Reply to the client

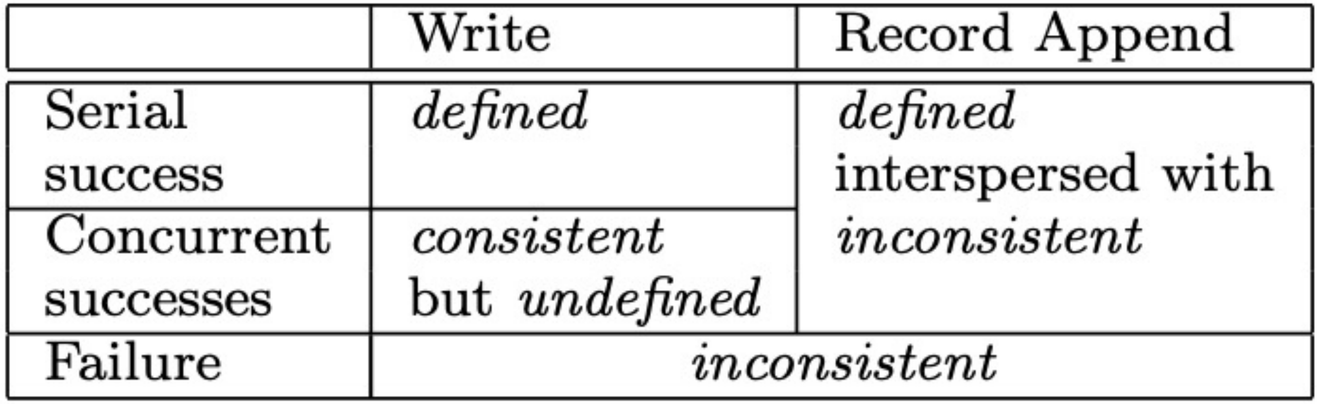
Figure 2: Write Control and Data Flow

Atomic Record Appends

- The client specifies only the data
- Similar to writes
- GFS appends data to the file at least once atomically

Consistency model

- Consistent: If all clients will always see the same data, regardless of which replicas they read from.
- Defined after a file data mutation: If it is consistent and clients will see what the mutation writes in its entirety.



	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

Table 1: File Region State After Mutation

Snapshot

Objective: To quickly create branch copies of huge data sets

Process

- Revoke all leases on the chunks in the files
- Duplicate the metadata pointing to the same chunks as the source files
- A new chunk is created due to the modification of either files
- Modify the metadata

Master’s Responsibilities

- Metadata storage
- Namespace management/locking

Namespace Management and Locking

- A lookup table mapping full pathnames to metadata
- Use locks over regions of the namespace to ensure proper serialization
- Each master operation acquires a set of locks before it runs

- How this locking mechanism can prevent a file /home/user/foo from being created while /home/user is being snap shotted to /save/user

	Read locks	**Write locks**
Snapshot operation	/home	/home/user
	/save	/save/user
Creation operation	/home	/home/user/foo
	/home/user	

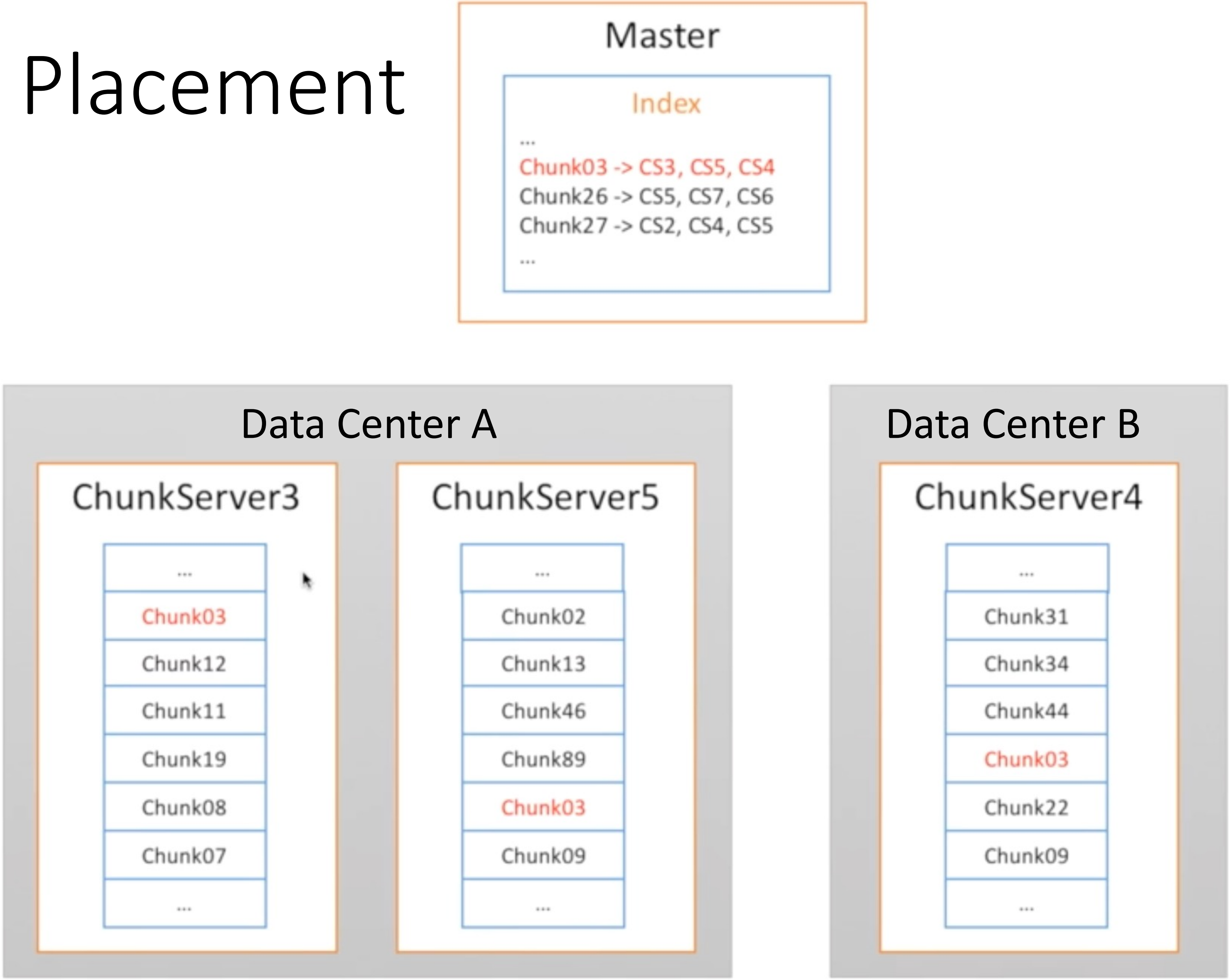
Master’s Responsibilities

- Metadata storage
- Namespace management/locking
- Heartbeat with chunkservers
- Chunk creation
  - Chunkservers with below-average disk space utilization
  - Limit the number of “recent” creations on each chunkserver
  - Spread replicas of a chunk across racks

Replica Placement

- Maximize data reliability and availability
- Maximize network bandwidth utilization
- Default: 3 replicas (2+1)

Replica Placement



Master

Index

...
Chunk03 -> CS3, CS5, CS4
Chunk26 -> CS5, CS7, CS6
Chunk27 -> CS2, CS4, CS5
...

Data Center A

ChunkServer3

...
Chunk03
Chunk12
Chunk11
Chunk19
Chunk08
Chunk07
...

ChunkServer5

...
Chunk02
Chunk13
Chunk46
Chunk89
Chunk03
Chunk09
...

Data Center B

ChunkServer4

...
Chunk31
Chunk34
Chunk44
Chunk03
Chunk22
Chunk09
...

Master’s Responsibilities

- Metadata storage
- Namespace management/locking
- Heartbeat with chunkservers
- Chunk creation
  - Chunkservers with below-average disk space utilization
  - Limit the number of “recent” creations on each chunkserver
  - Spread replicas of a chunk across racks
- Re-replication
- Rebalancing

Fault Tolerance

- High availability
  - Fast recovery
    - Master and chunks server can restart in a few seconds
  - Chunk replication
  - Shadow master
    - Provide read-only access to the file system even when the primary master is down
- Data Integrity
  - Checksum

Conclusion

- Fault tolerance + High aggregate throughput
- Widely used
  - HDFS - corresponding open-source classic implementation of GFS

Thanks!