



ECAP470: CLOUD COMPUTING

Dr. Tarandeep Kaur
Assistant Professor

Learning Outcomes



After this lecture, you will be able to,

- learn about Google File System (GFS).
- Know the architecture and operations of GFS.

Google File System (GFS)

- Scalable distributed file system for large distributed data-intensive applications.
- Provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.
- To meet the rapidly growing demands of Google's data processing needs.

Google File System (GFS)

- Scalable distributed file system for large distributed data-intensive applications.
- **Provides fault tolerance** while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.
- To meet the rapidly growing demands of Google's data processing needs.

Google File System (GFS)

- Scalable distributed file system for large distributed data-intensive applications.
- Provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.
- To **meet the rapidly growing demands** of Google's data processing needs.

Google File System (GFS)

Initial exploration on design space before implementing GFS-

1. Component failures are the norm rather than the exception.
2. Files are huge by traditional standards.
3. Most files are mutated by appending new data rather than overwriting existing data.

Google File System (GFS)

4. Co-designing the applications and the file system API benefits the overall system by increasing the flexibility.
5. Multiple GFS clusters are currently deployed for different purposes.

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

GFS Assumptions

Hardware failures are common (commodity hardware)

Files are large (GB/TB) and their number is limited (millions, not billions)

Two main types of reads: large streaming reads and small random reads!

Workloads with sequential writes that append data to files

Once written, files are seldom modified. (!=append) again

Random modification in files possible, but not efficient in GFS

High sustained bandwidth trumps low latency

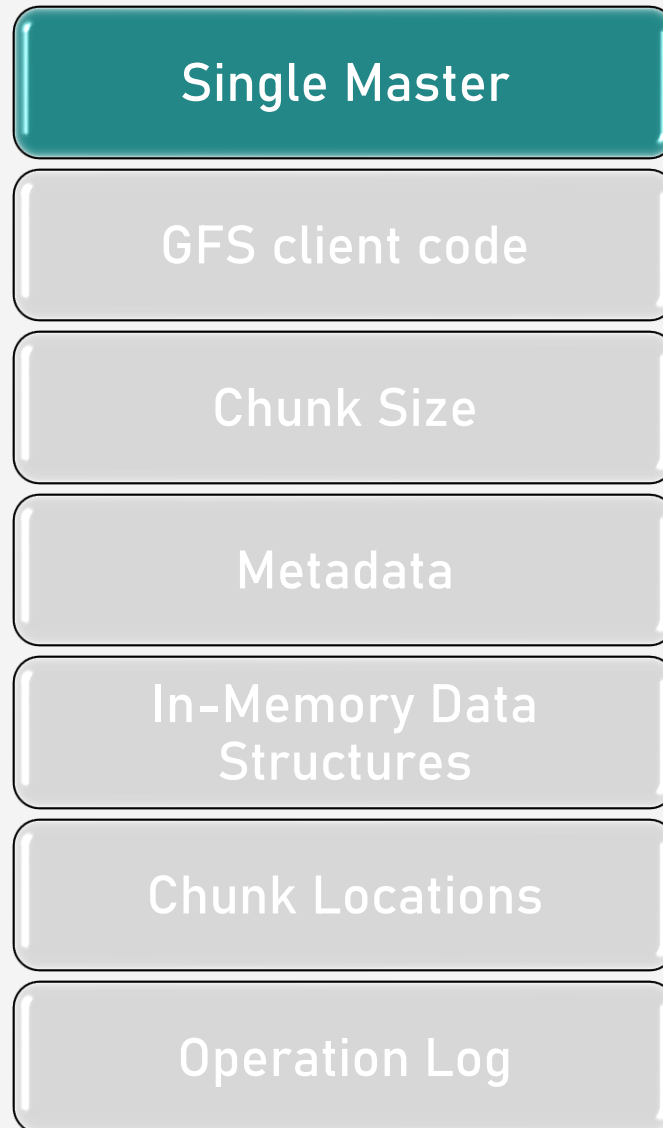
GFS Interface

- GFS provides a familiar file system interface, though it does not implement a standard API such as POSIX. Files are organized hierarchically in directories and identified by pathnames.
- Supports usual operations to create, delete, open, close, read, and write files.
- GFS has snapshot and record append operations.

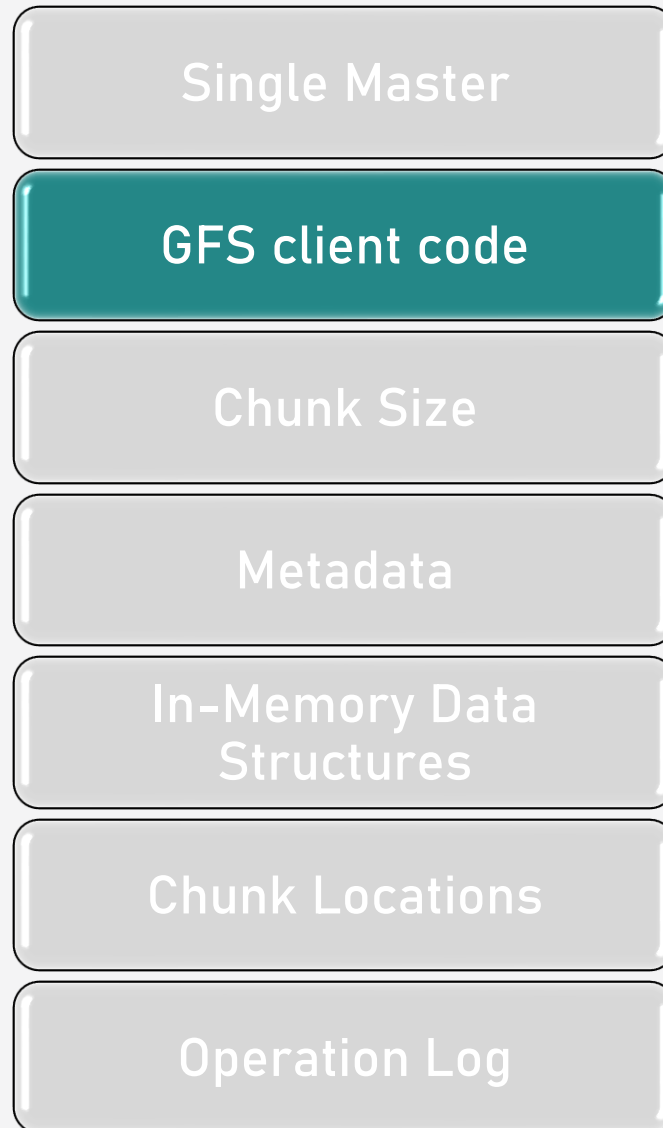
GFS Architecture

- GFS cluster **consists of a single master and multiple chunkservers and is accessed by multiple clients.**
- Files are divided into fixed-size chunks.
- Chunkservers

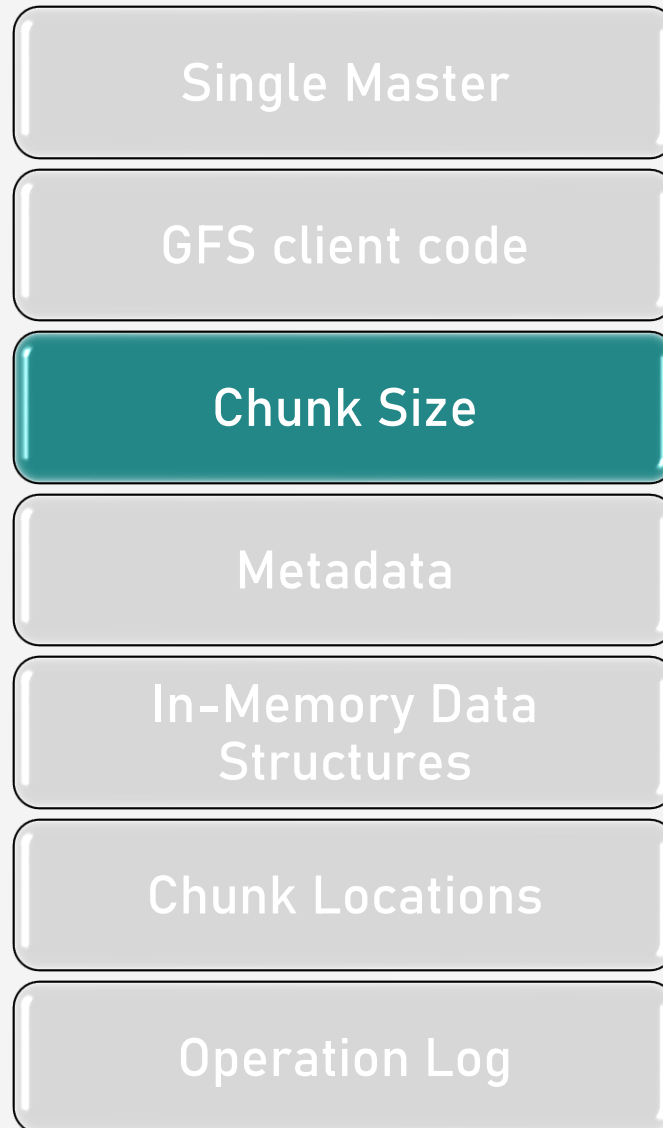
GFS Architecture



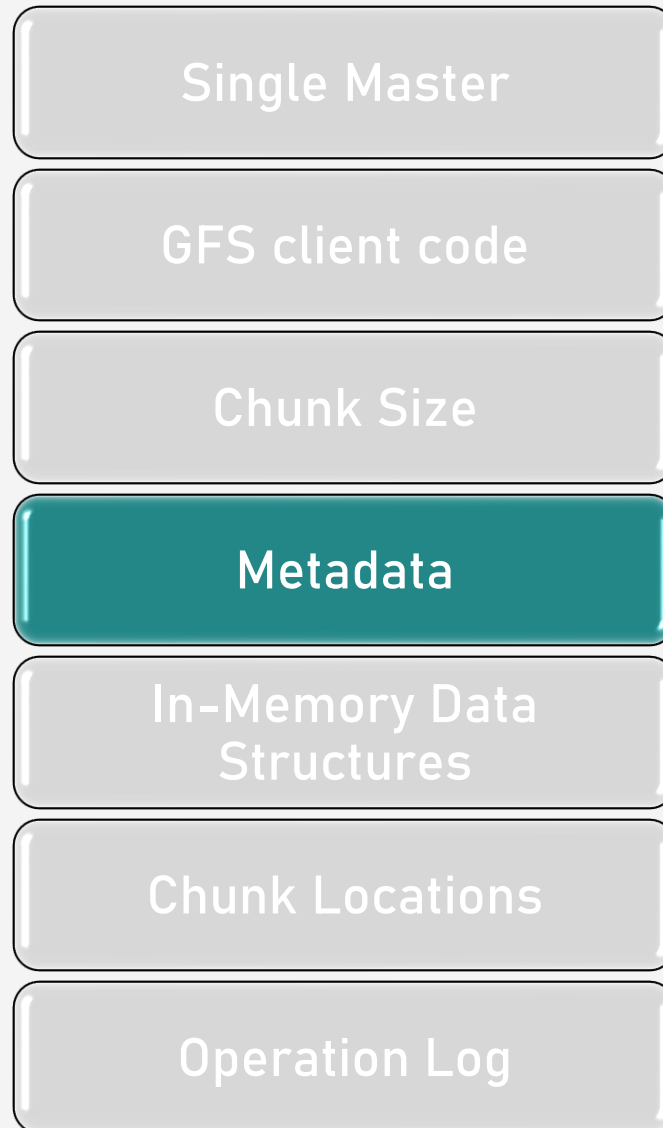
GFS Architecture



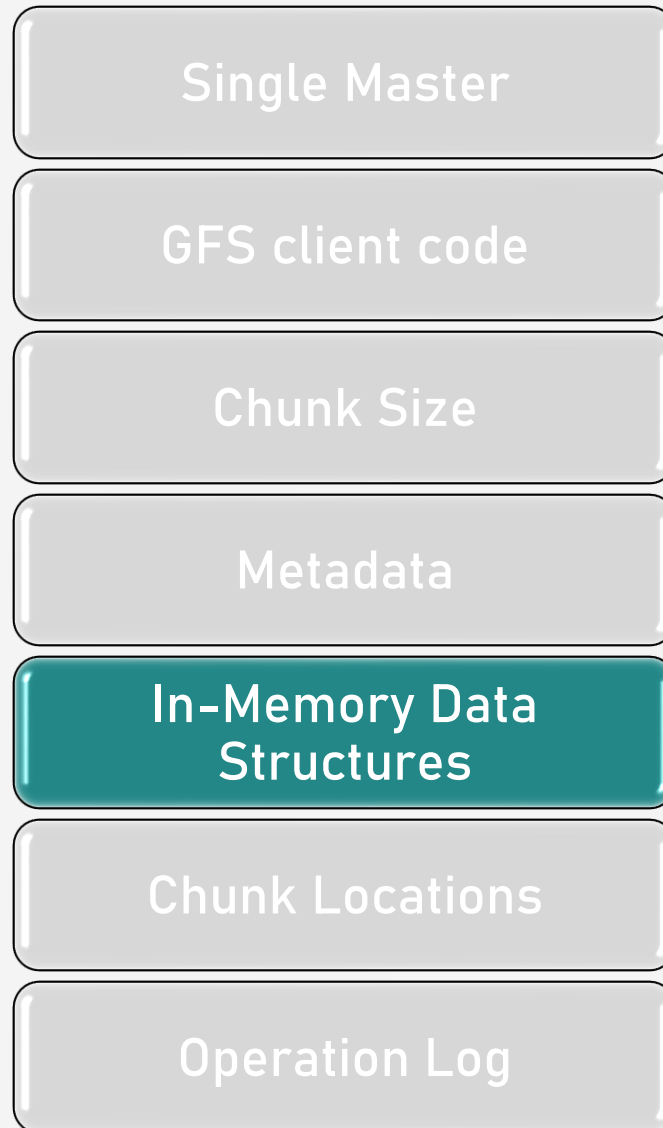
GFS Architecture



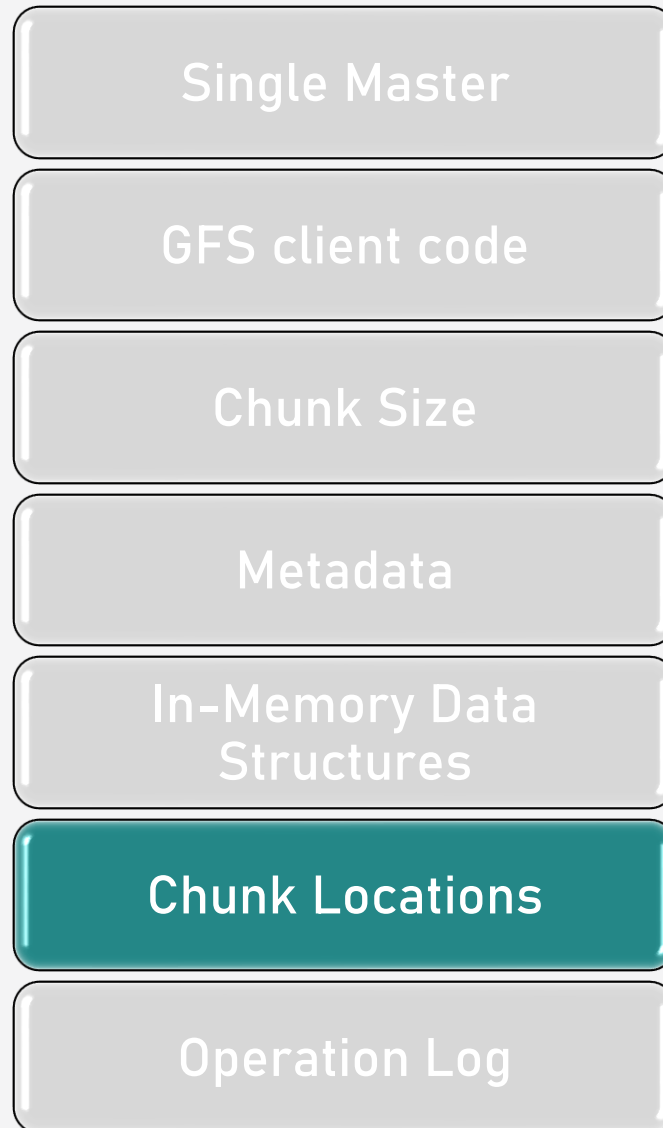
GFS Architecture



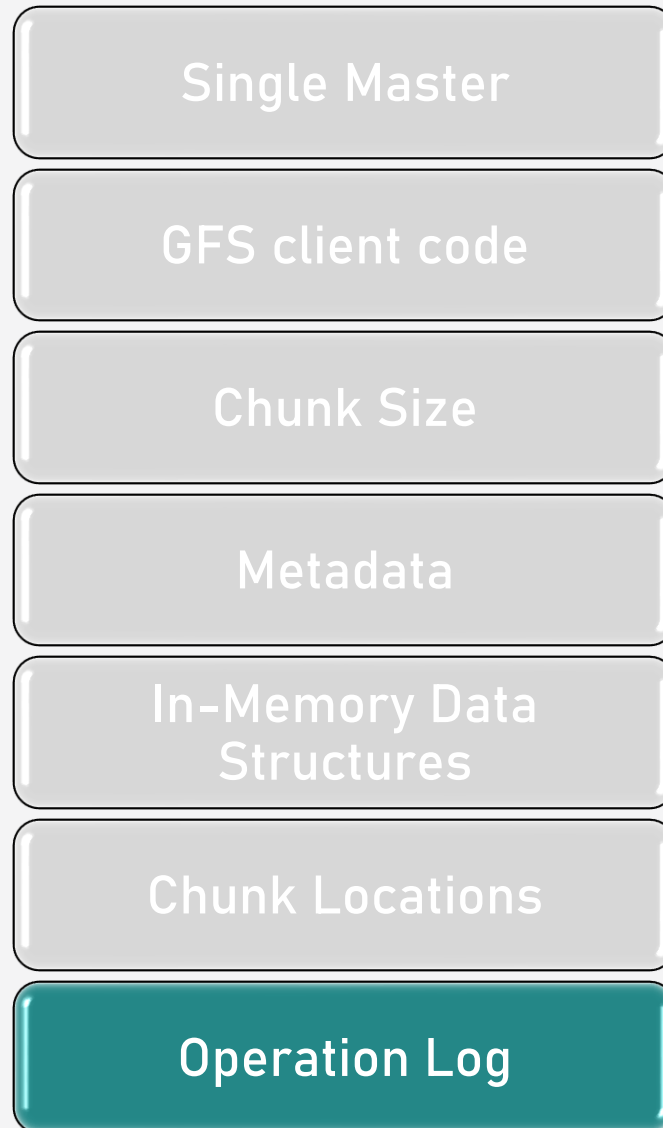
GFS Architecture



GFS Architecture

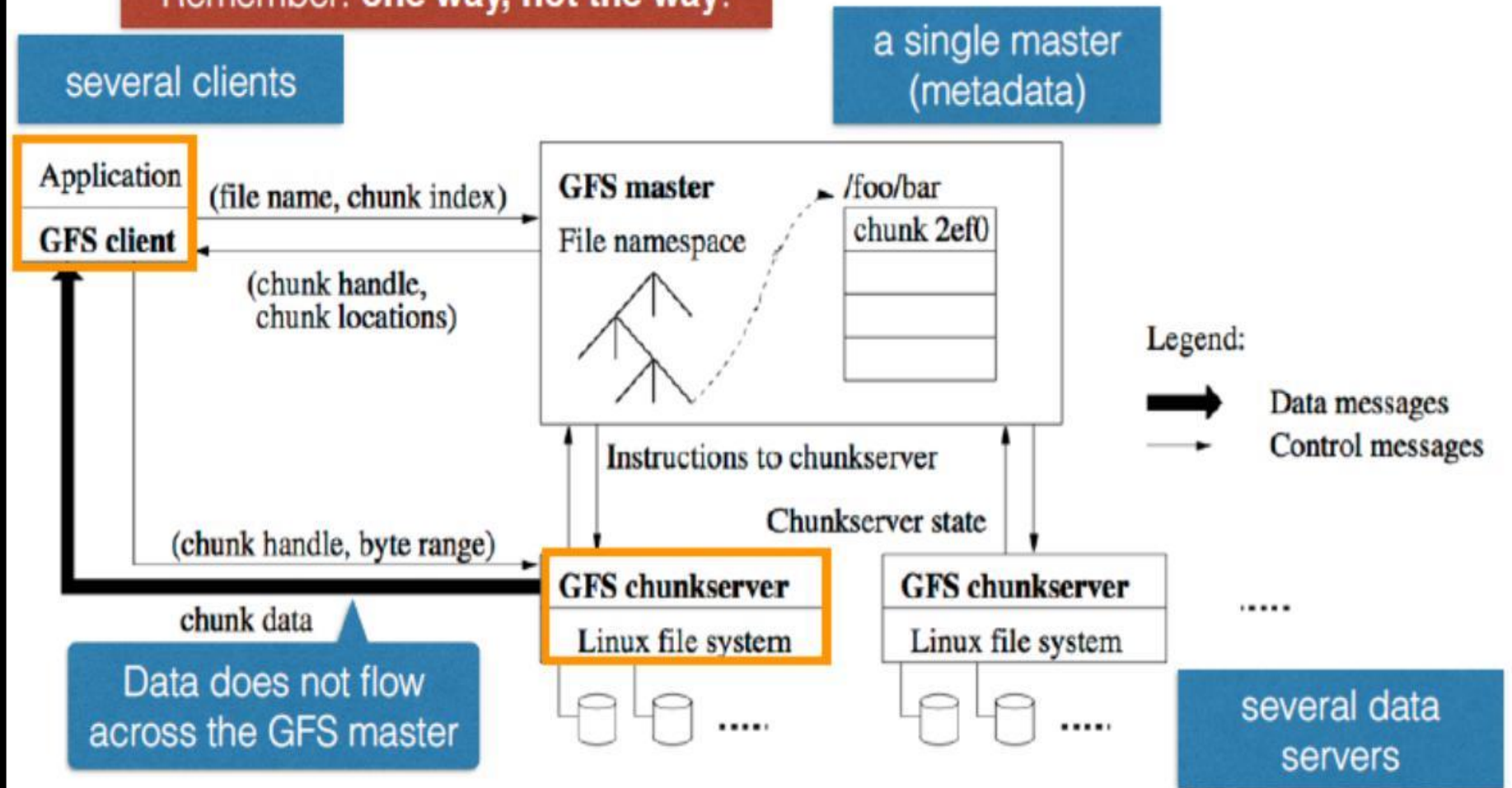


GFS Architecture



GFS Architecture

Remember: **one way, not the way.**



GFS Operation

- First, using the fixed chunk size, the client translates the **file name and byte offset** specified by the application into **a chunk index** within the file.
- Then, it sends the master a request containing the file name and chunk index.

GFS Consistency Model

GFS has a relaxed consistency model that supports the highly distributed applications well but remains relatively simple and efficient to implement.

Detailed GFS Master Operation

The master executes all namespace operations. In addition, it manages chunk replicas throughout the system: it makes placement decisions, creates new chunks and hence replicas, and coordinates various system-wide activities to keep chunks fully replicated, to balance load across all the chunkservers, and to reclaim unused storage.

Detailed GFS Master Operation

1. Namespace Management and Locking
2. Replica Placement
3. Creation, Re-replication, Rebalancing

Conclusion

GFS demonstrates the qualities essential for supporting large-scale data processing workloads on commodity hardware. While some design decisions are specific to our unique setting, many may apply to data processing tasks of a similar magnitude and cost consciousness.

Conclusion

- GFS provides fault tolerance.
- GFS delivers high aggregate throughput to many concurrent readers and writers performing a variety of tasks.
- GFS has successfully met the storage needs and is widely used within Google.

A teal-colored oval with a white-to-teal gradient fill and a subtle drop shadow, centered on a teal background.

That's all for now...