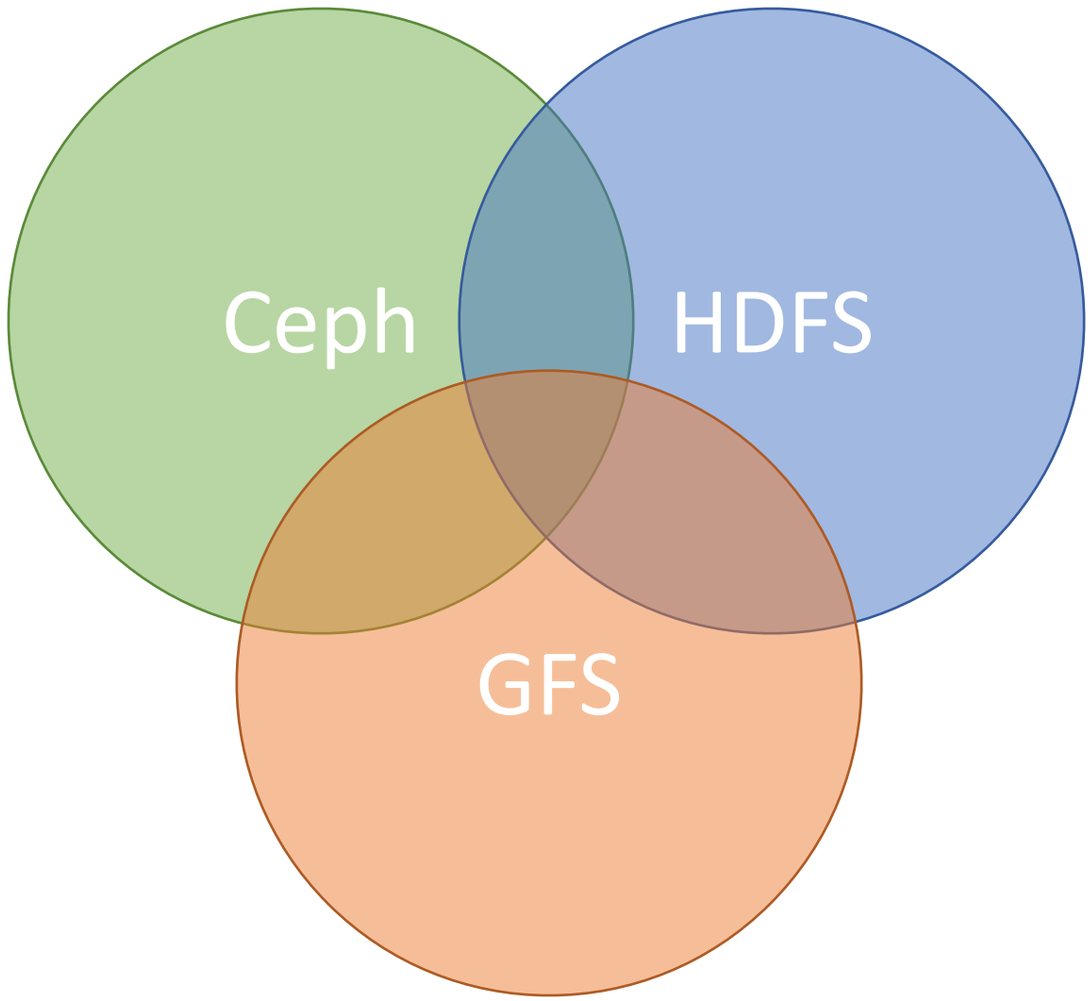
Distributed File Systems

GFS, HDSF, Ceph

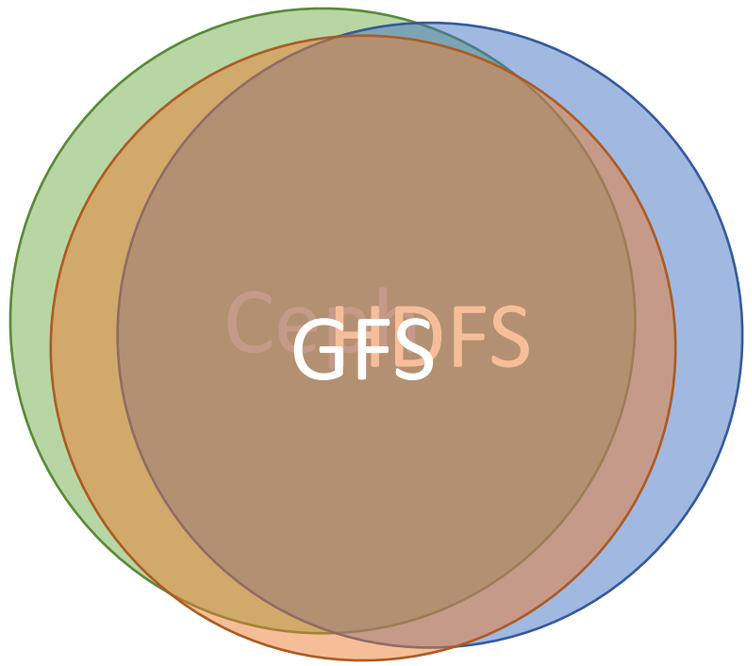
Content

- What is a distributed file system?
- GFS – Google File System
- HDFS – Hadoop Distributed File System
- Ceph – RedHat Distributed File System



Content

- What is a distributed file system?
- GFS – Google File System
- HDFS – Hadoop Distributed File System
- Ceph – RedHat Distributed File System



What is a distributed file system?

The difference between a distributed file system and a distributed data store is that a distributed file system allows files to be accessed using the same interfaces and semantics as local files - e.g. mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model. Distributed data stores, by contrast, require using a different API or library and have different semantics (most often those of a database).

GFS – Google File System

- Centrally managed distributed file system for Google's internal use
- Developed from BigFiles by Larry and Sergey
- Not implemented in kernel, but rather a user space library
- Intra data centre file system
- New version in the works …

GFS – Motivation

- Fault tolerance
  - Google is using commodity parts with relatively high failure rate
  - Software quality cannot be guaranteed - SW errors
  - High degree of shared resources: power supplies, switches, etc.
- Client guarantees
  - High availability - scale to infinity
  - Consistency
  - Concurrent write and read from multiple clients
- Infrastructure management
  - High sustained bandwidth is more important than latency.

GFS – Dimensions

- Multiple GFS instances in a data centre
- Millions of files
  - 100 MB is the norm
  - Multi GB files are common
  - Small files not common
- Aggregately storing terabytes of data
- 100’s of storage nodes
- Concurrently accessed by 100’s of clients

GFS – Assumptions

- High HW and SW failure rate
- Modest number of large files
- Large streaming reads – 1 MB per operation
- Few and small random reads – KB per operation
- Large sequential append writes
- Append to the end of a chunk is far more likely than writes that overwrite existing data

GFS – Guarantees

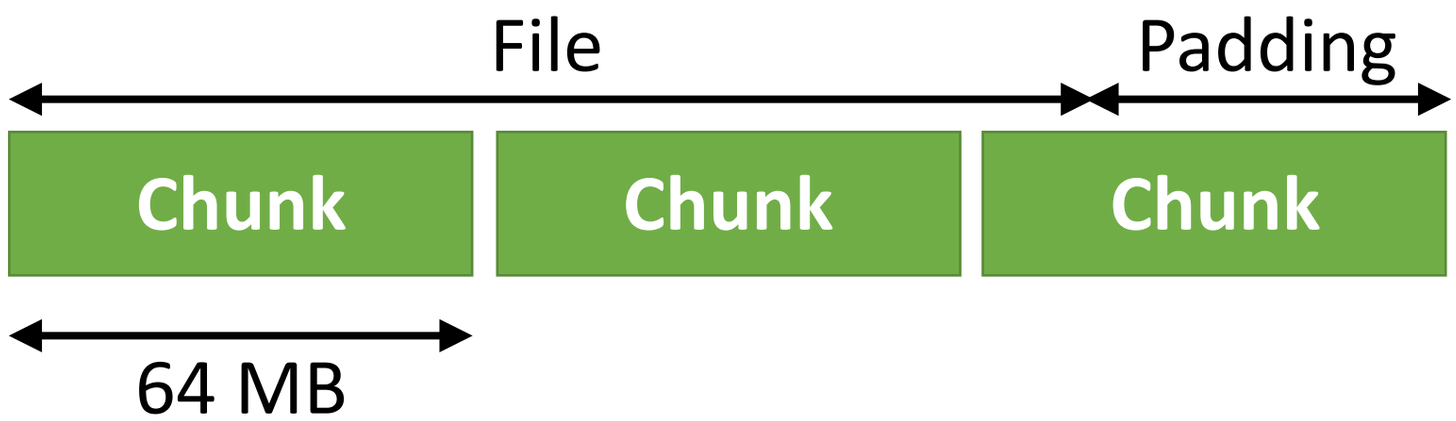
- Consistent
  All clients will see the same data,
  regardless of which replicas they read from
- High availability
- Atomic write

GFS – Entities

- Chunk
- Chunk server
- Master
- Client

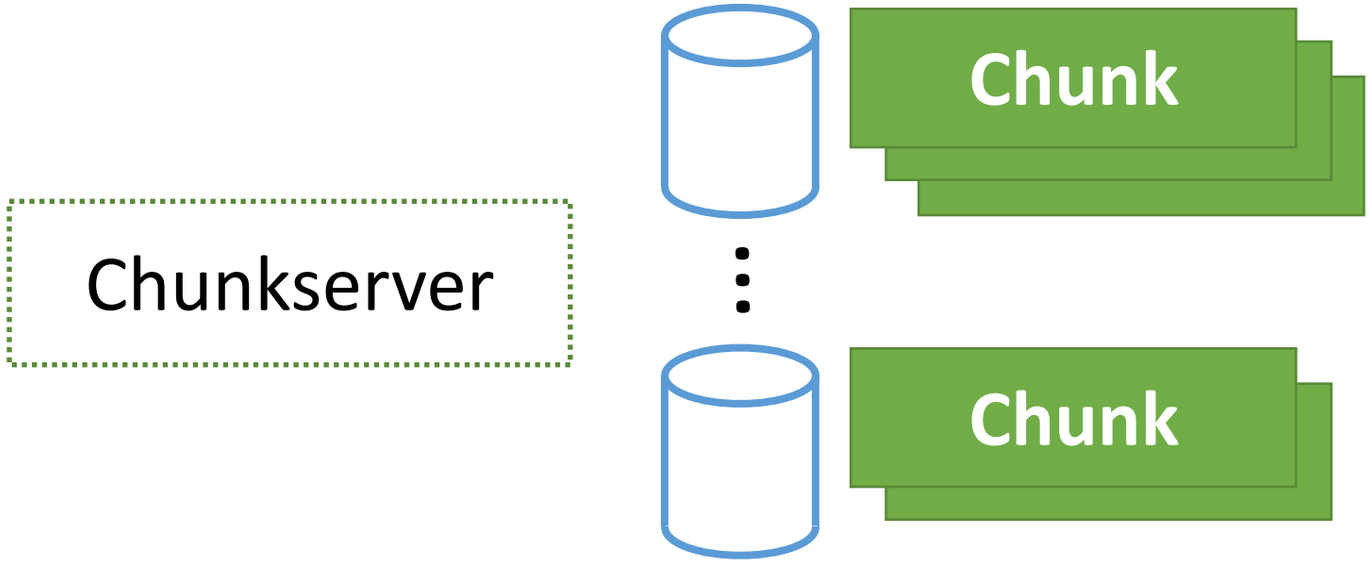
GFS – Entities – Chunk

- 64 MB linux file – Most files are larger than 100 MB
- Requires 64 Bytes of metadata in the master
- Stored and replicated on multiple chunk servers



GFS – Entities – Chunkserver

- Stores chunks on local disk(s)
- Any machine in the data centre, heterogenous
- Performs checksum on each block
- Checksum and chunk map is stored in memory
- Communicates with master

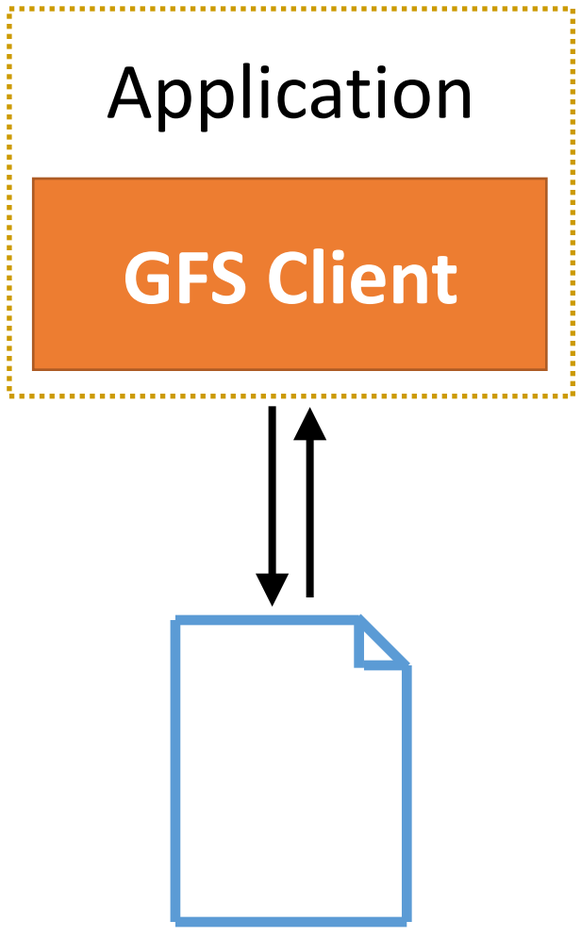


GFS – Entities – Master

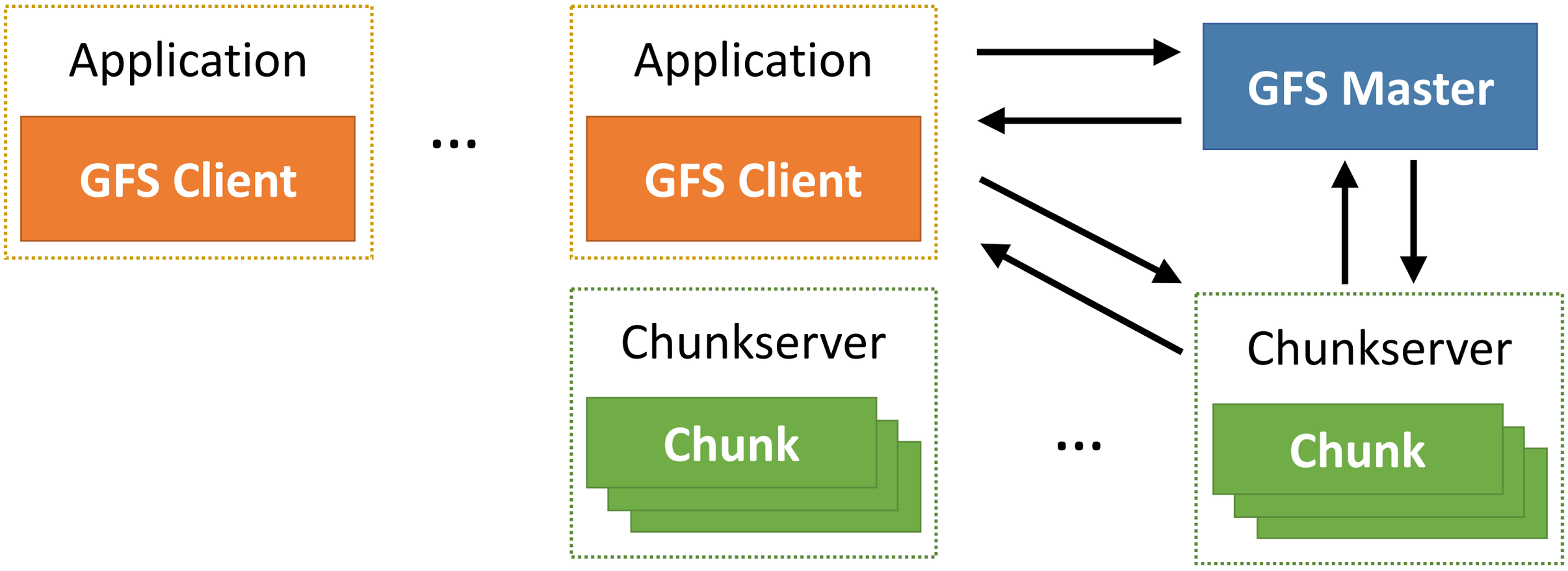
- File system metadata – Stored in memory
  - Namespace
  - Access control information
  - Imposes read and write locks
  - File to chunk mapping
  - Chunk location
- Lookup table mapping full pathnames with metadata. Stored in memory
- No per-directory data structure or aliasing
- Handles file permutations
- Periodically scans namespace for changes,
  loss of chunkservers, deleted files, etc. using heartbeat messages
- Ensures fault tolerance
- There can be only one, for now

GFS – Entities – Client

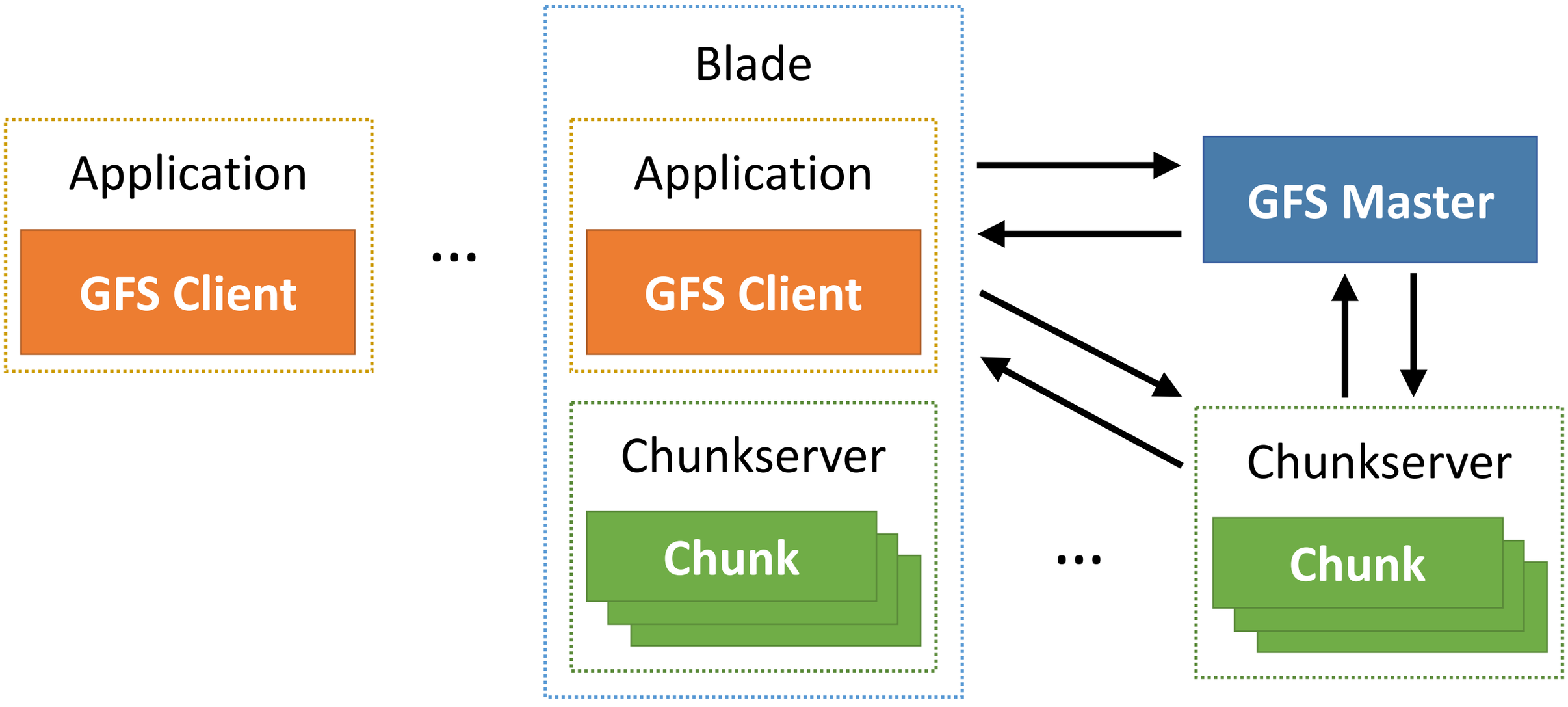
- Application, value adder, revenue generator
- Map Reduce, Google Services
- Consumes and produces data
  - i.e. reads and writes to the file system
- Own checksum
- Has no ability to report error in file system
- Caches metadata from master



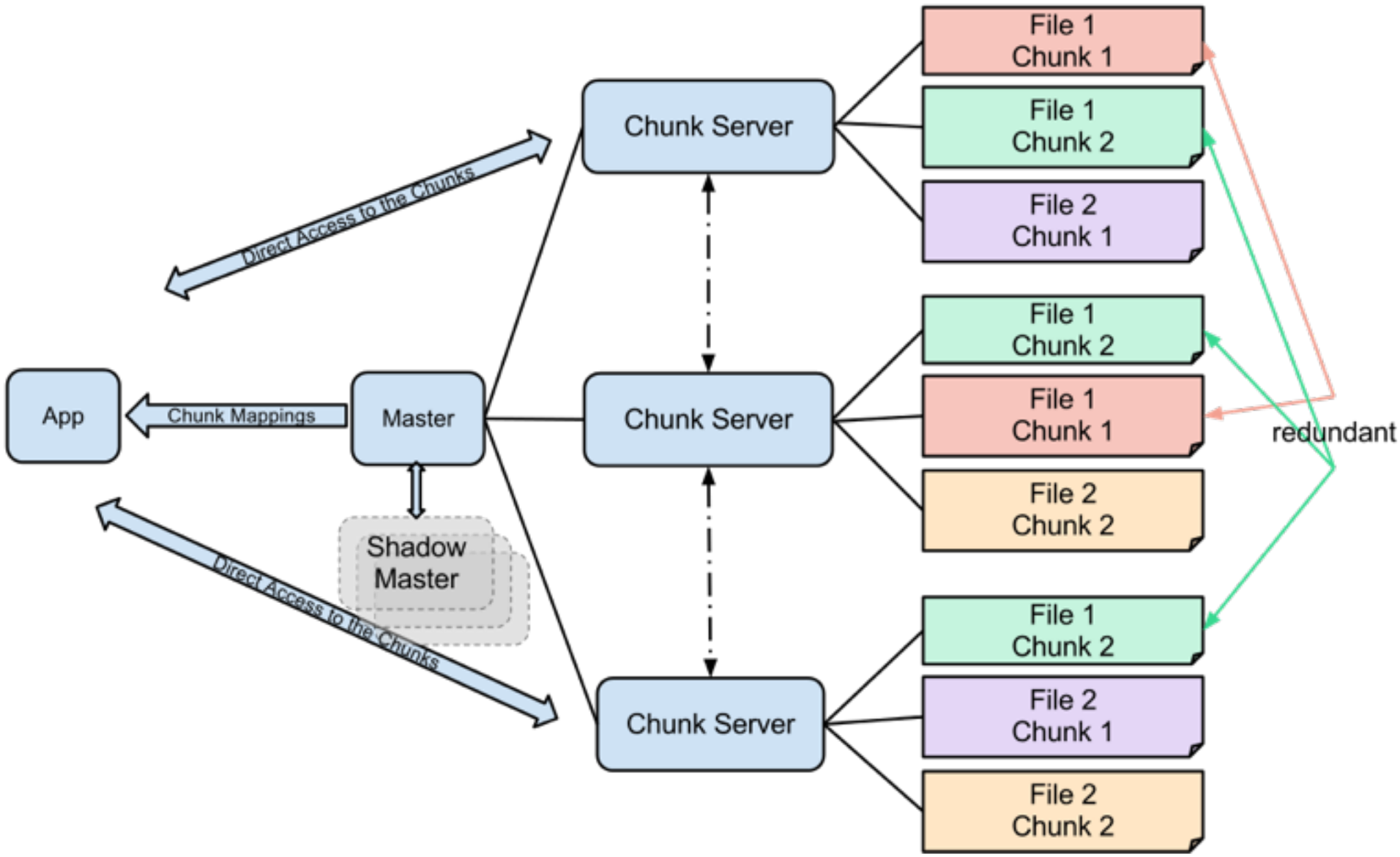
GFS – Architecture



GFS – Architecture

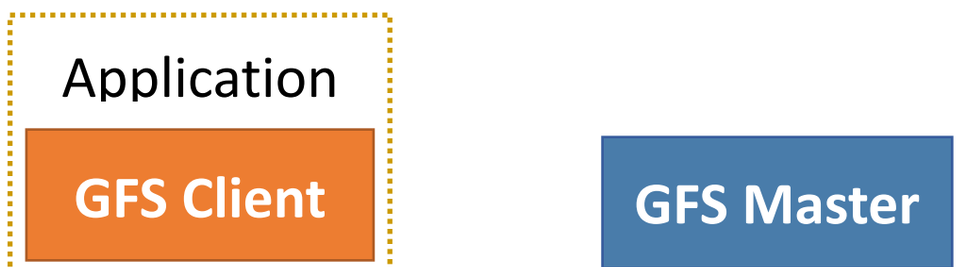


GFS – Redundancy



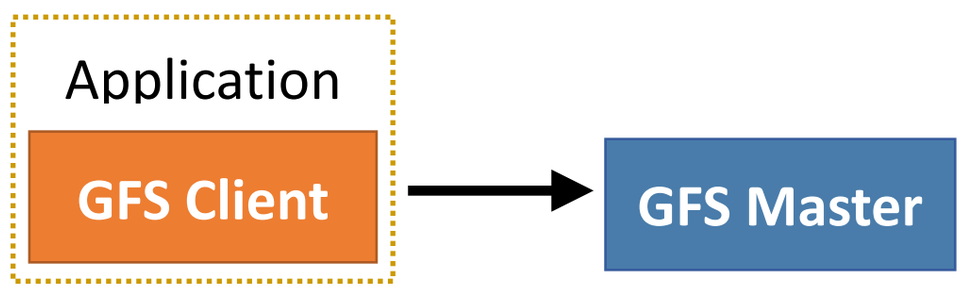
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



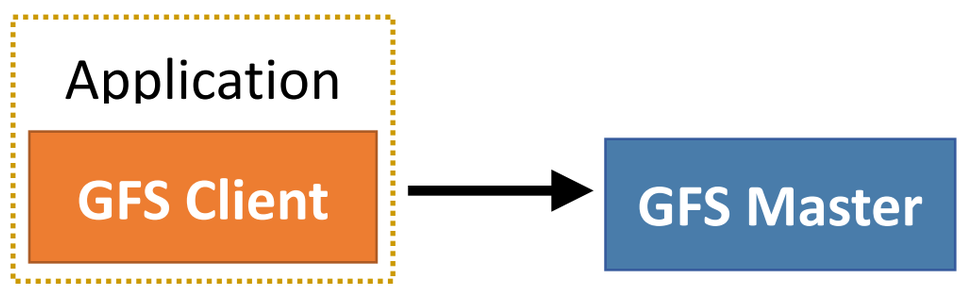
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



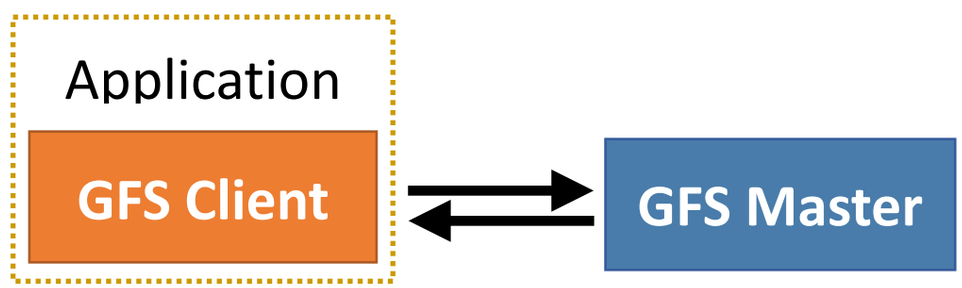
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



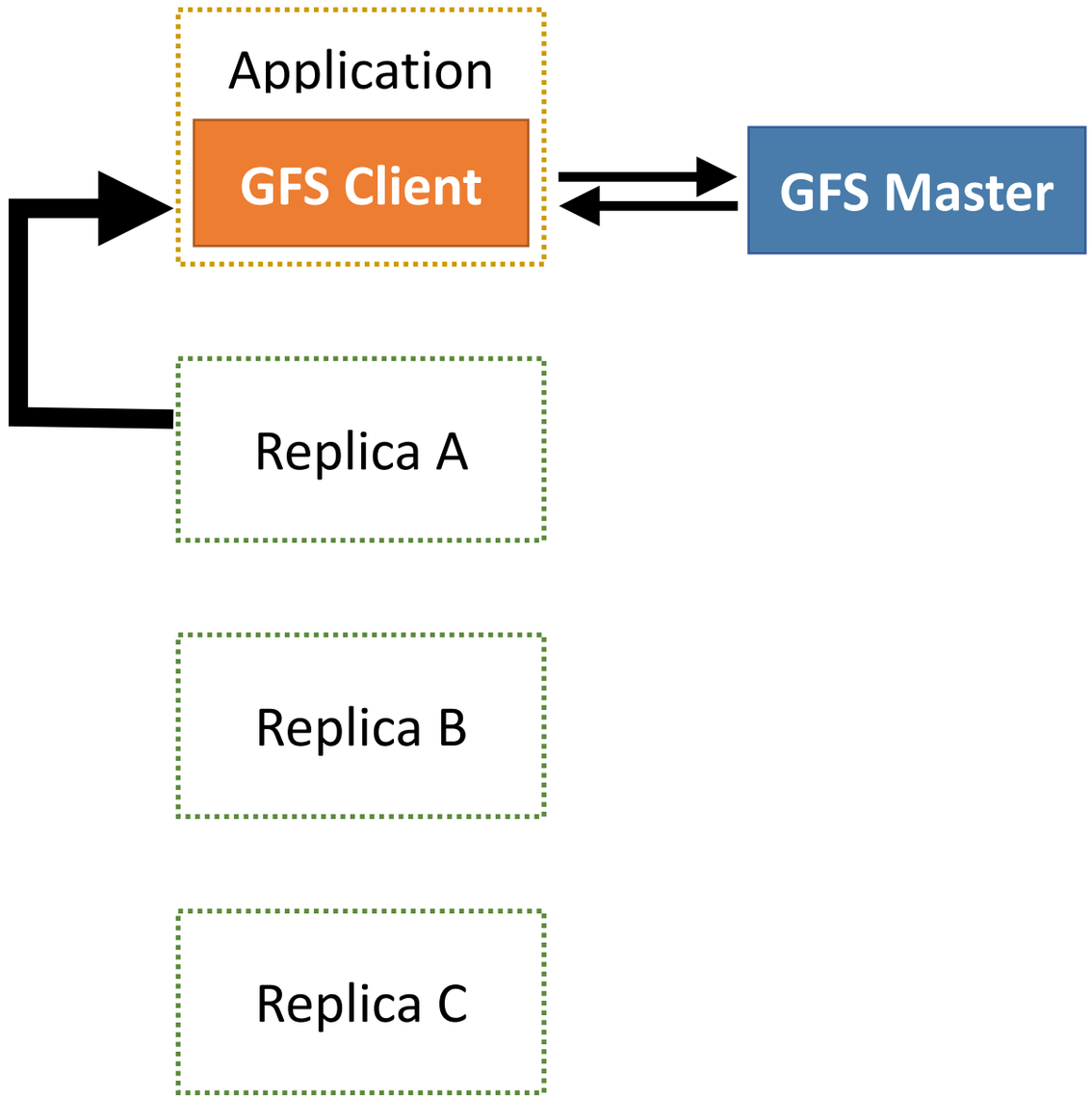
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



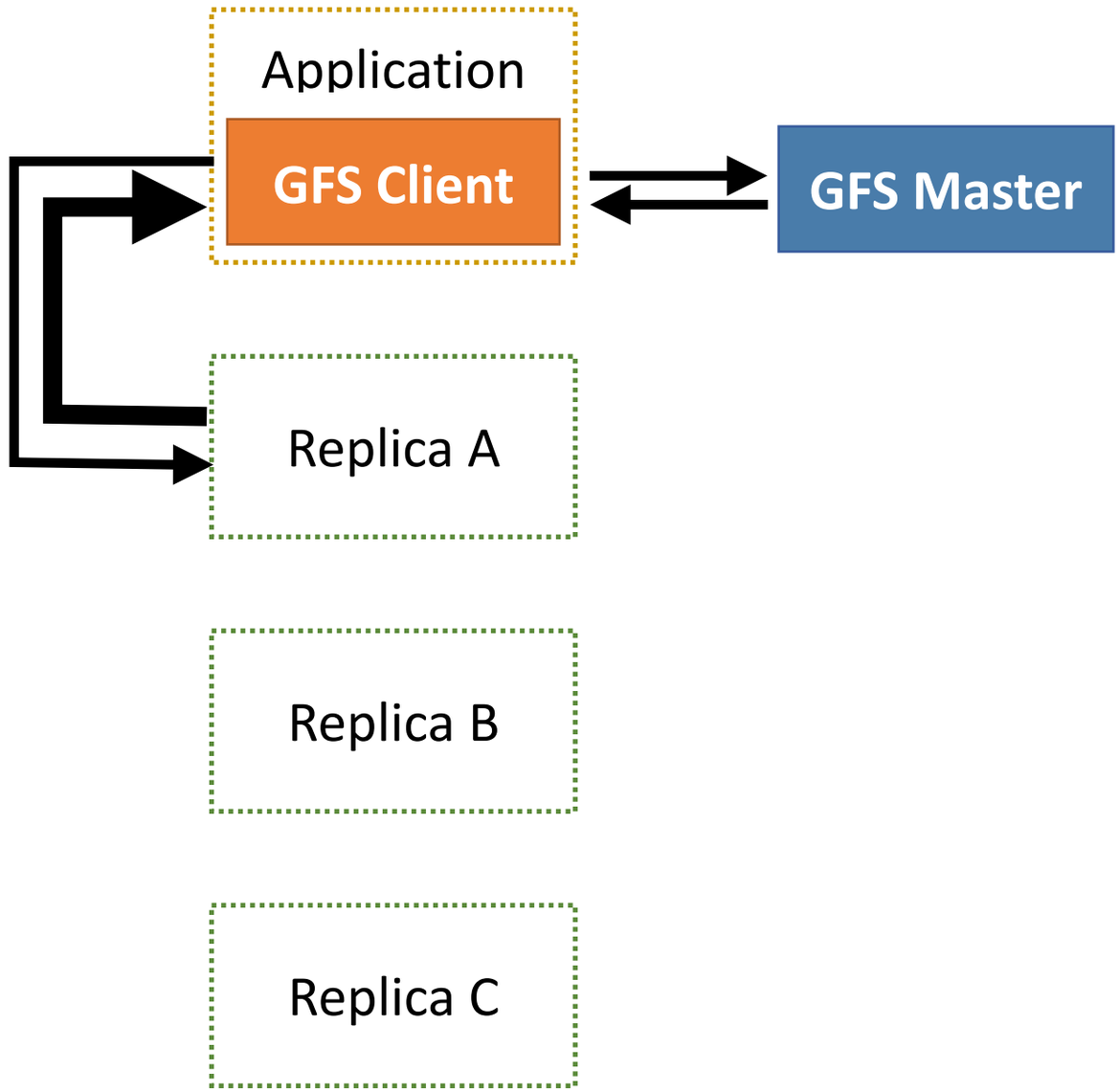
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



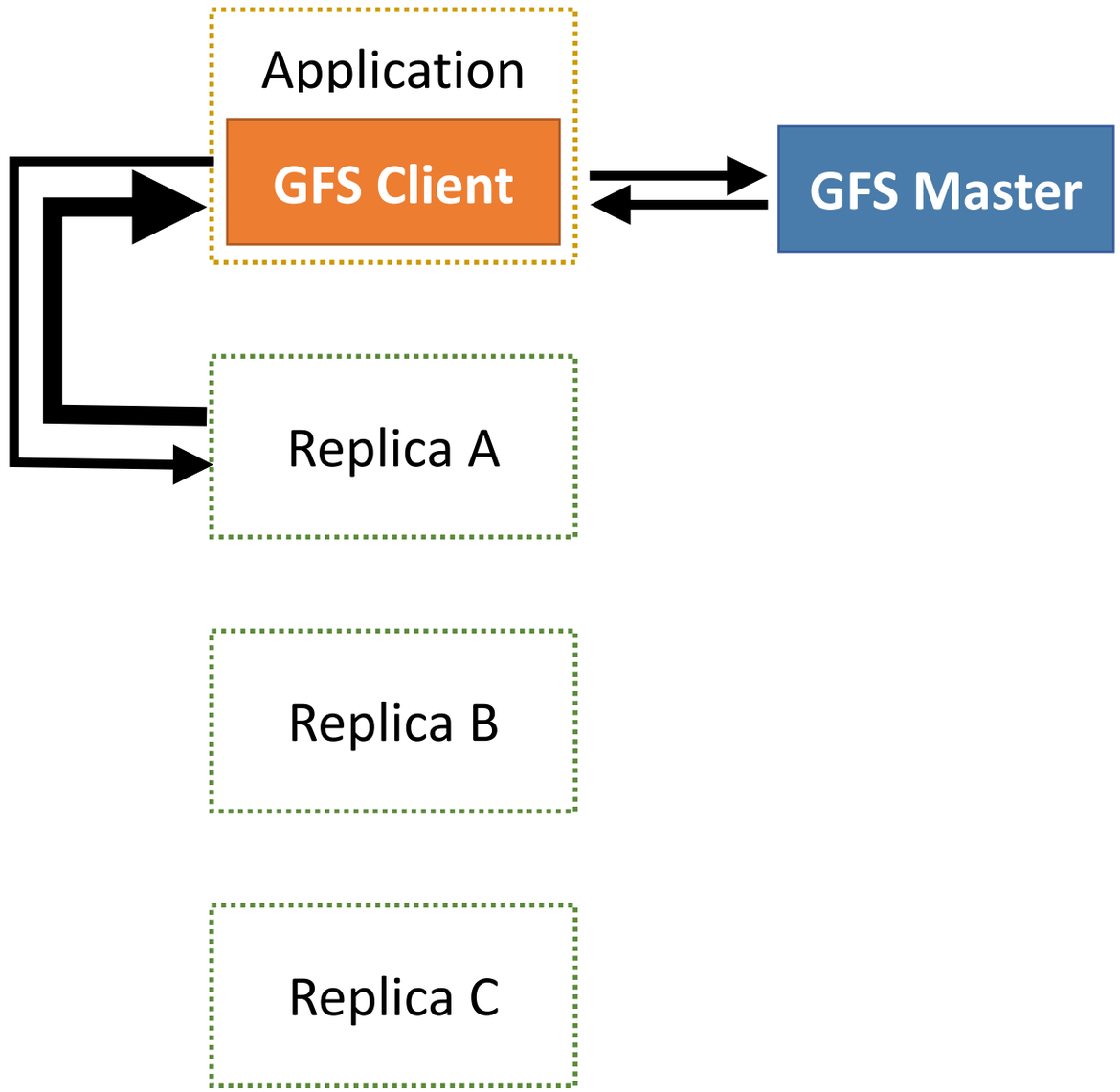
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



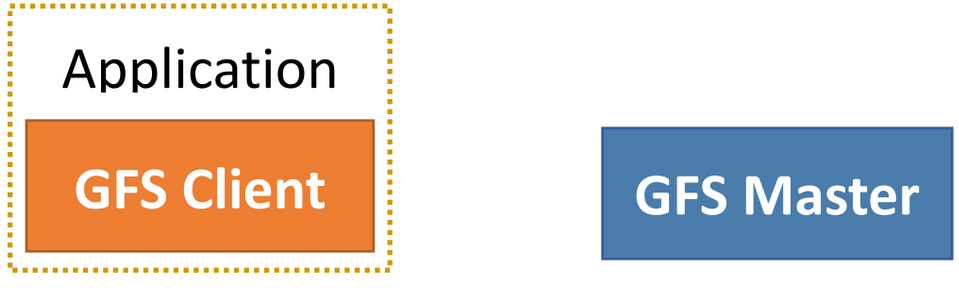
GFS – Read

1. Client asks master for replicas (Namespace)
2. Master replies with replicas (IP addresses and offsets)
3. Client reads from random replica
4. If it fails to read, it retires from a different replica



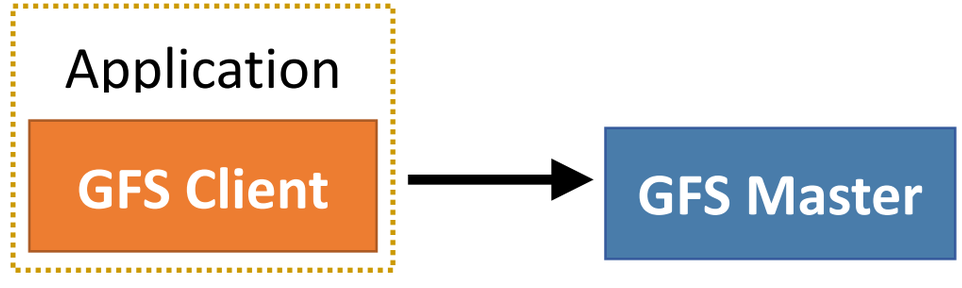
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



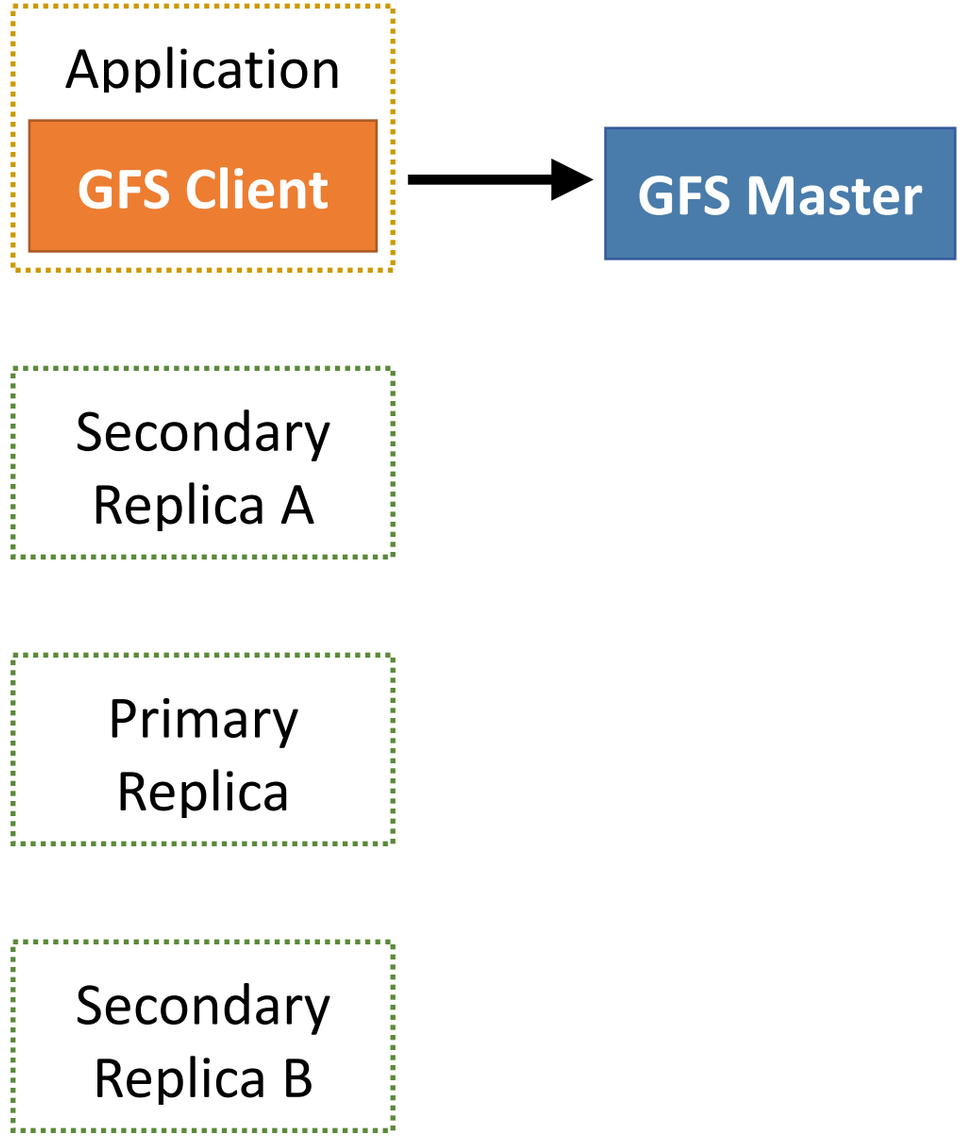
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



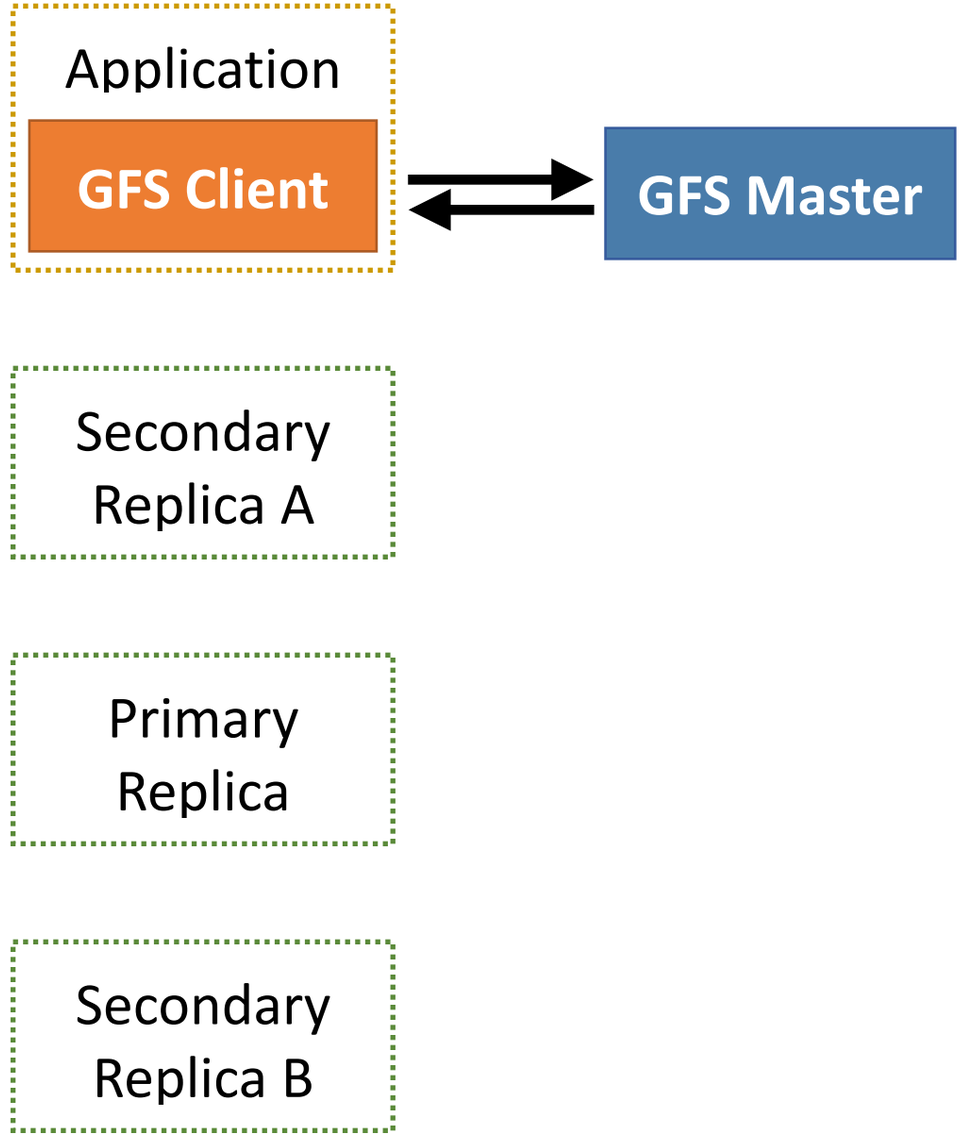
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



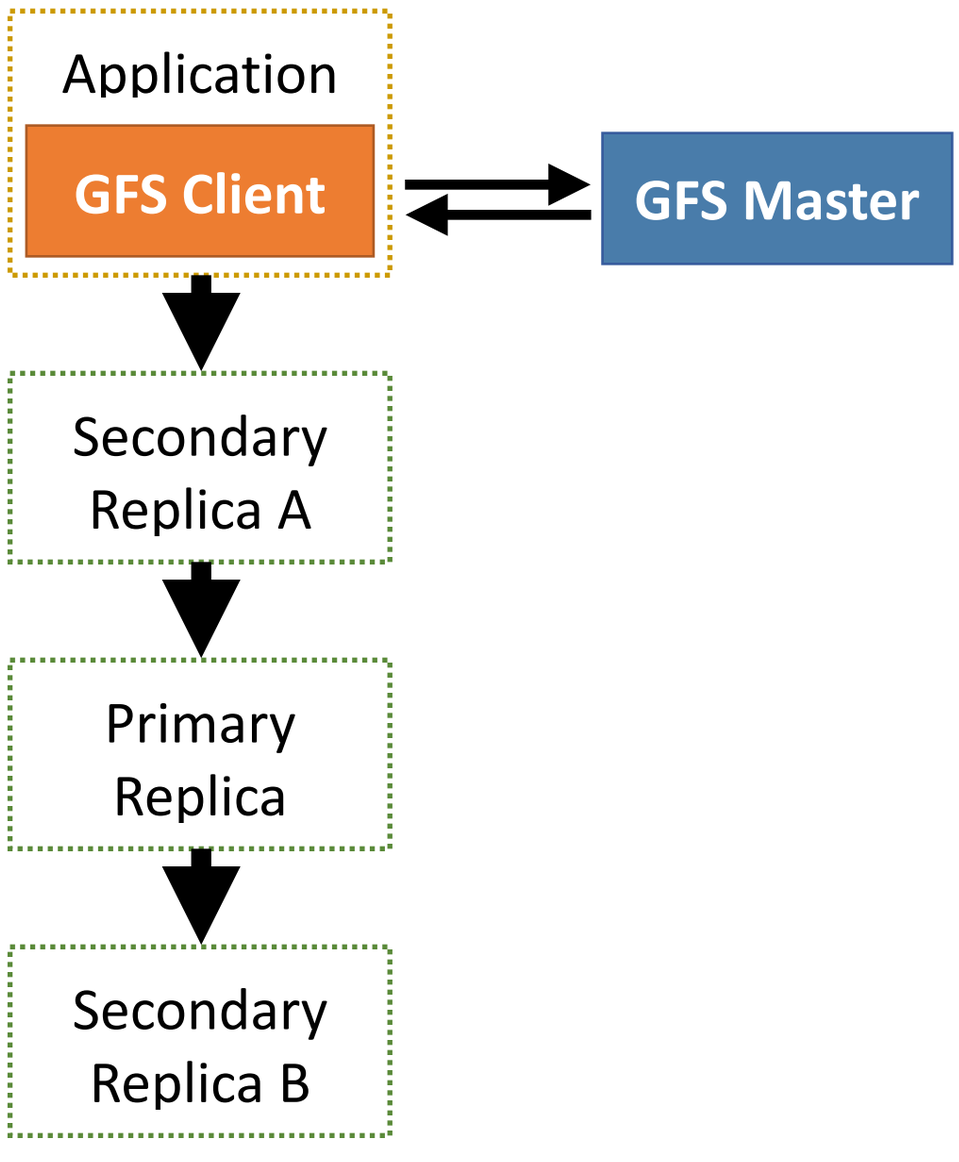
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



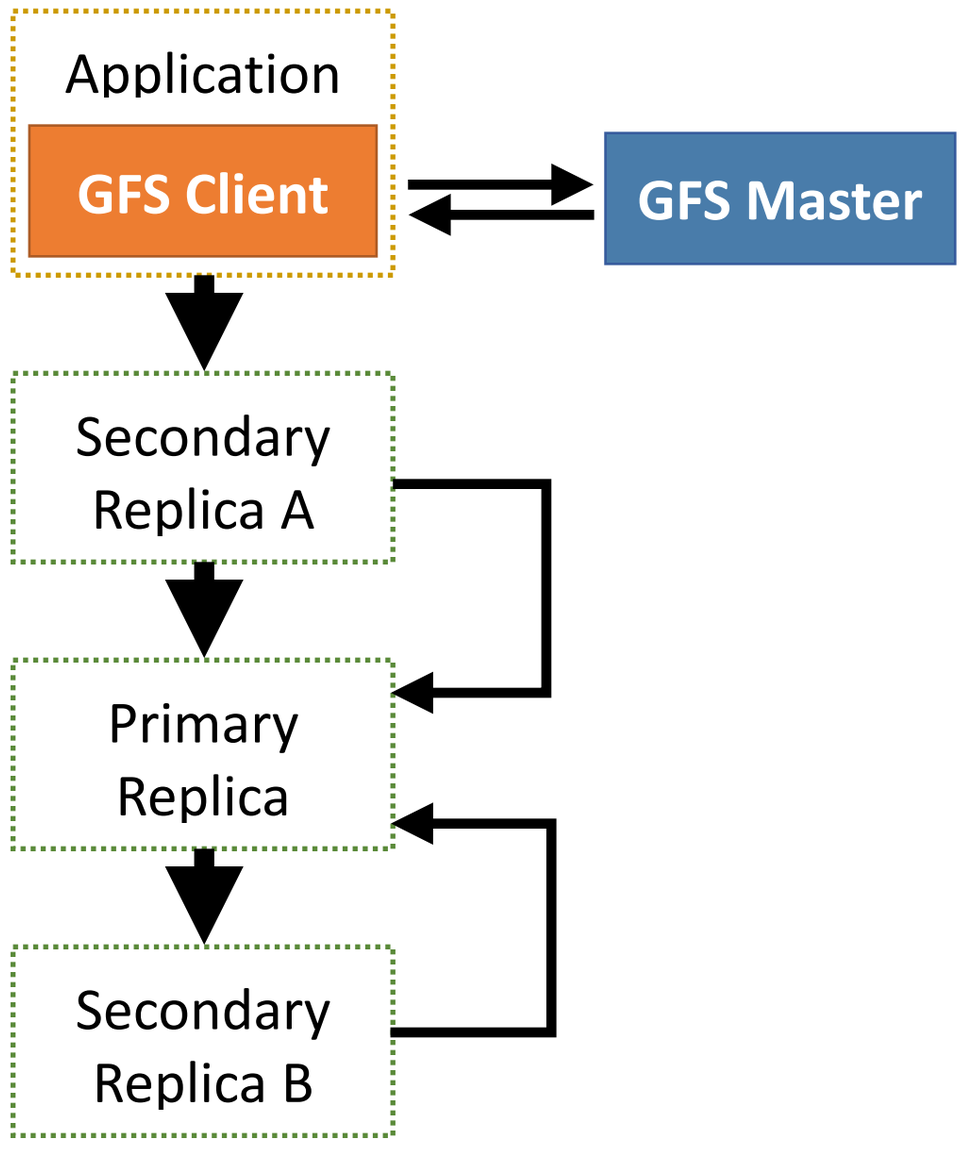
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



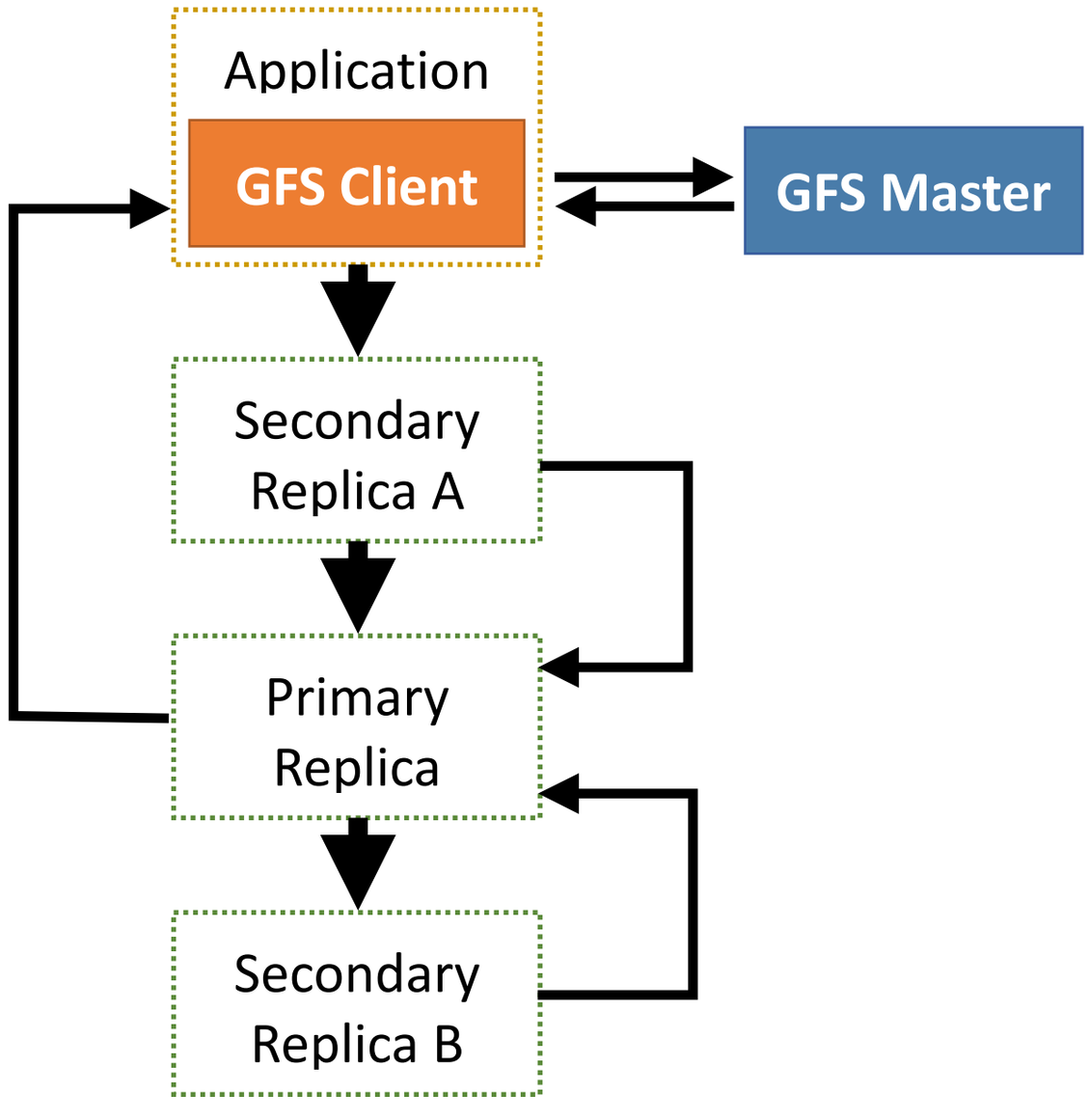
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



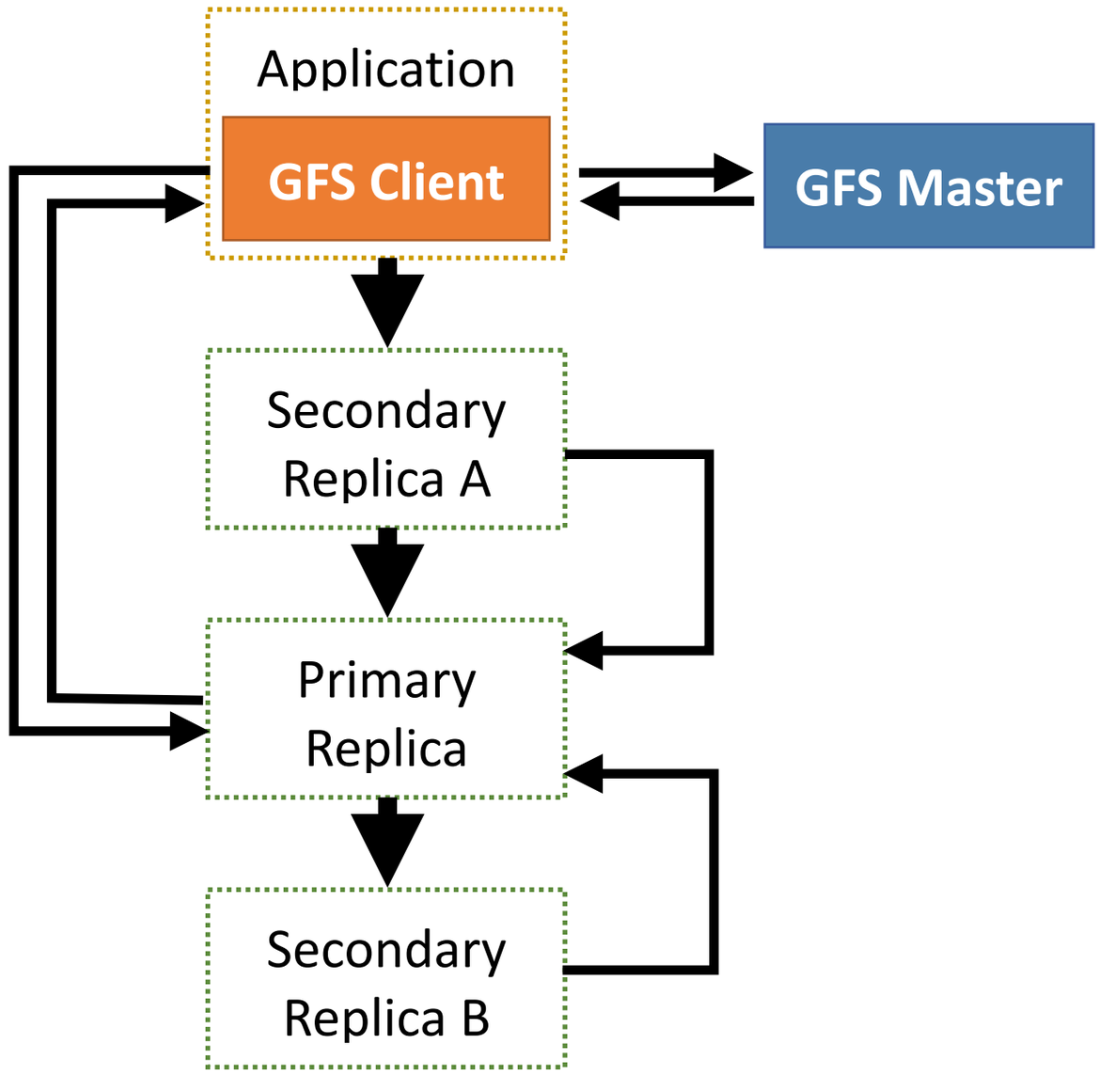
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



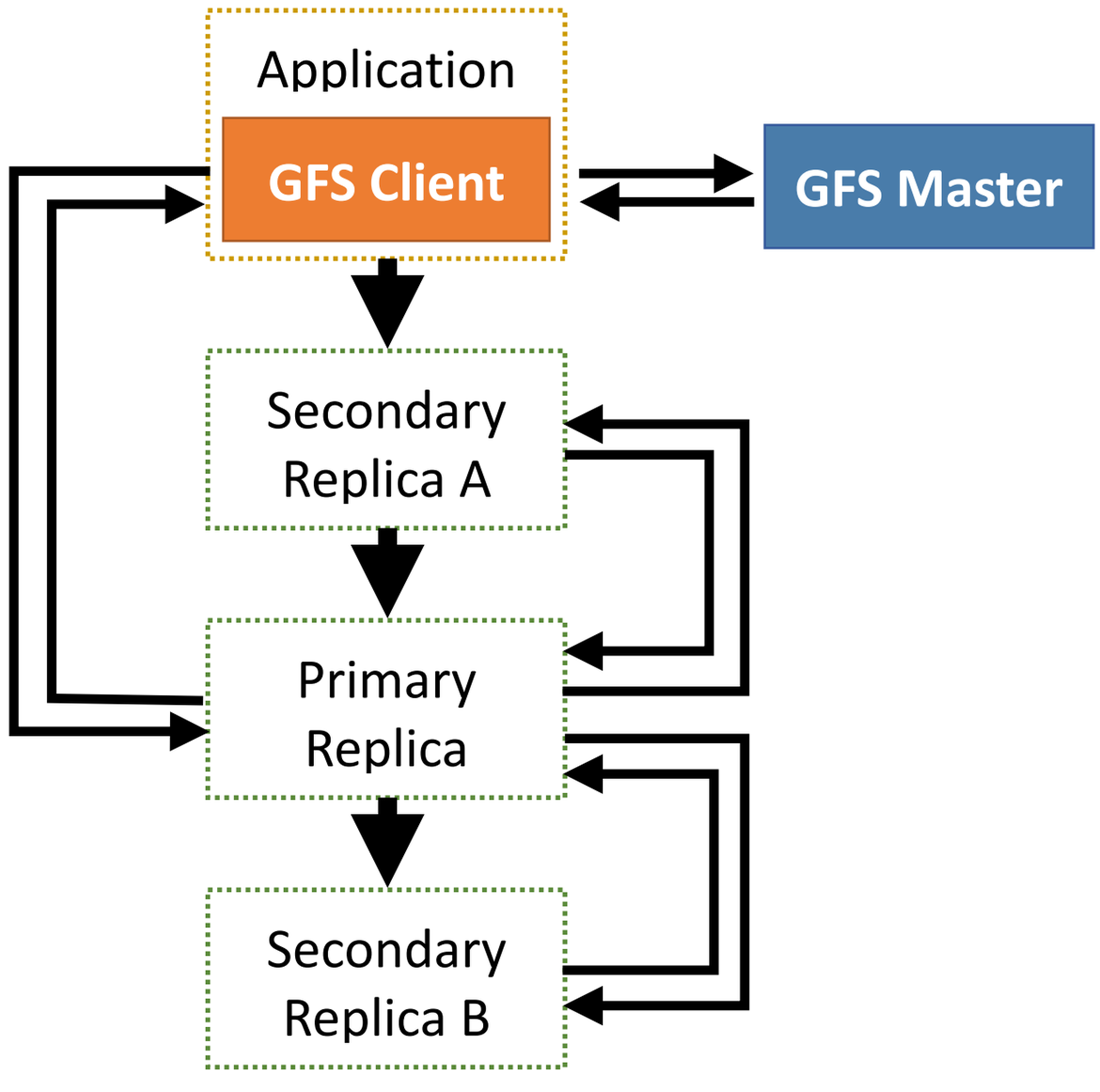
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



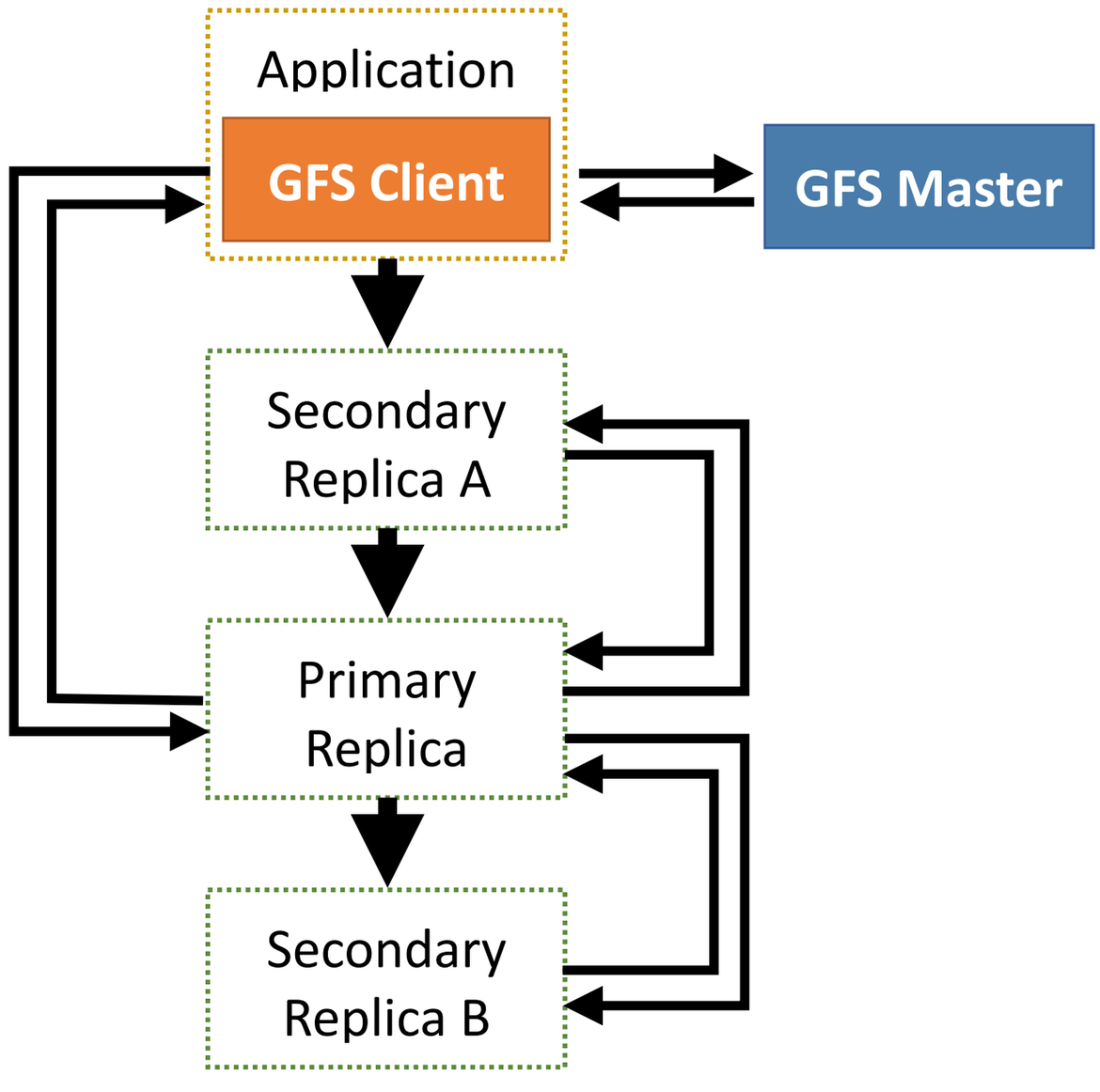
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



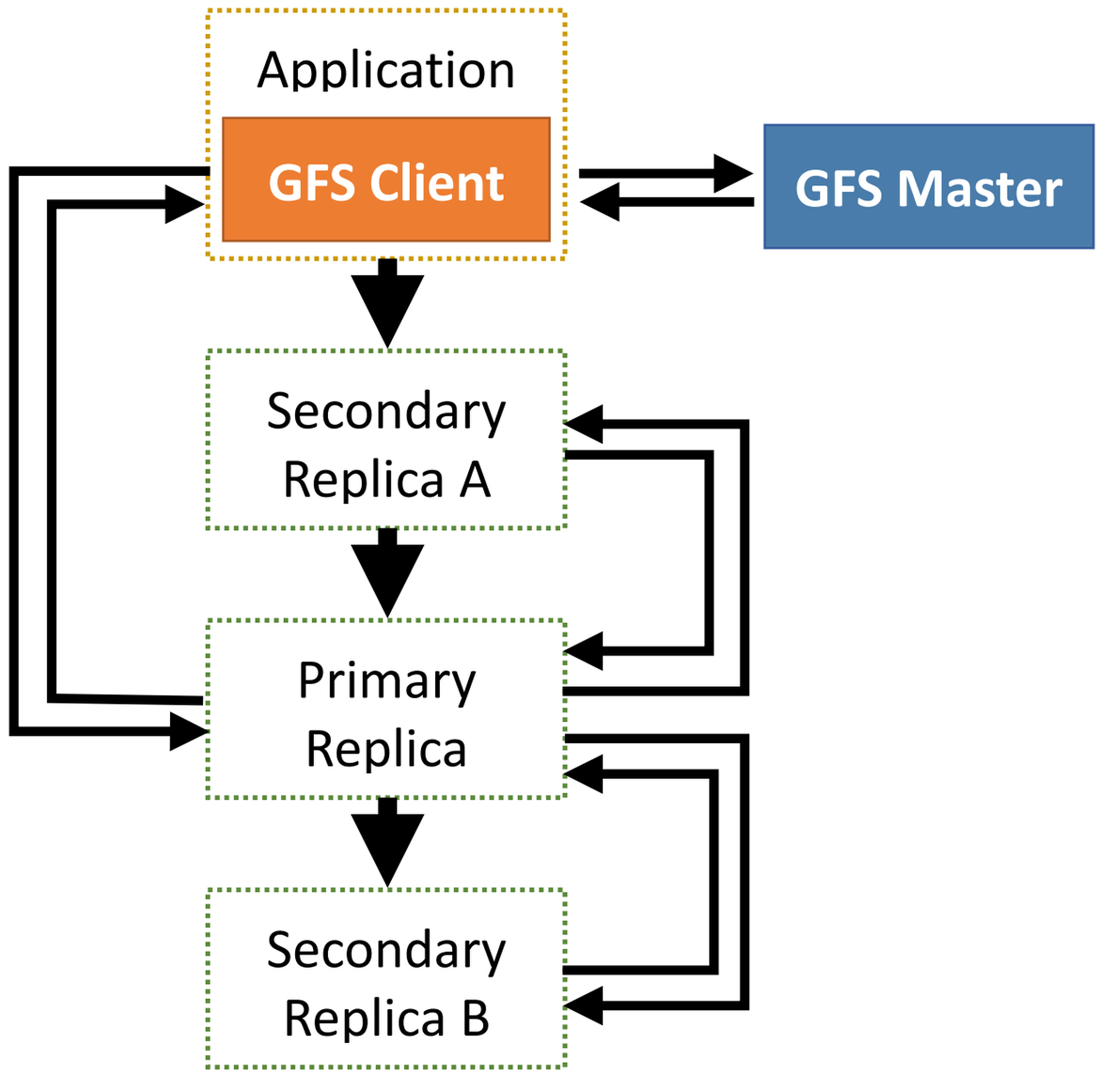
GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



GFS – Write (per chunk)

1. Client asks master for primary chunkserver (Namespace)
2. Master evaluated lease validity
3. Master replies with identity of the primary (IP address)
   Client caches data
4. Client pushed data to closest chunkserver
5. Client forwards write requests



GFS – Fault tolerance – Master

- Does not distinguish between normal and abnormal termination.
- Accessed using it's canonical name
- Log file is key
  - Replicated to multiple machines
  - Defines the order of concurrent operations
  - Log file only only persistent record of meta data
  - Fast recovery by replaying log file from checkpoint

GFS – Fault tolerance – Chunkserver failure

- Chunk are replicated across multiple chunkservers, not RAID
- Chunks are split into 64KB blocks with 32bit checksum stored in memory
- Not discovered until when master probes chunkserver
  - New replica created when not responding

GFS – Maintenance (I)

- **Garbage collection**
  - Performed during heartbeat
  - File renamed when deleted, up to client to know the new name
    - Action can be undone
- Resources are reclaimed lazily, with a user configurable interval
  - Google uses a 3-day grace period
- Orphan chunks are removed
- Anything not know to the master is “garbage”

GFS – Maintenance (II)

- **Replica placement**
  - Simple default policy, replicas across multiple racks
  - Equalise disk utilisation across chunk servers
- **Creating, re-replication, and rebalancing**
  - Limit number of recent creations on each chunkserver
  - Cloning done to closest suitable chunkserver
  - Periodically, graceful rebalancing

GFS – User tuneable parameters

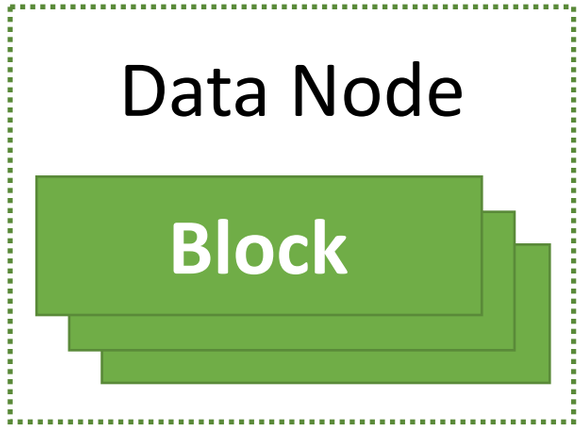
- Number of replicas
- Chunksever placement policy
- Client and chunkservers co-locality policy
- Lease timeout
- Rebalancing periodicity
- Garbage collection periodicity

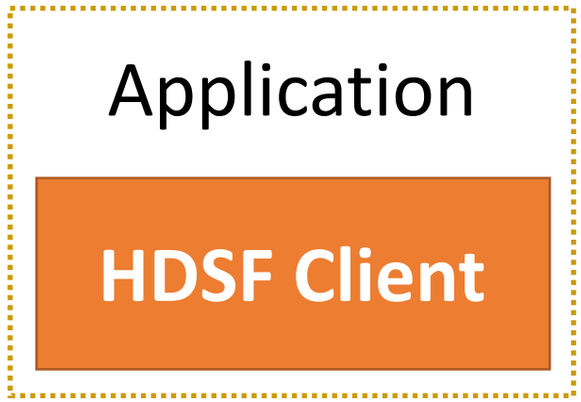
HDFS – Hadoop Distributed File System

- Decoupled metadata and data store
- One metadata store
- Locality though the client and replica placement policies
- More signalling: storage capacity, status HB, etc.
- Greater configurability, block sizes, fault tolerance
- Fault tolerance through multiple replicas
- Obviously designed for Hadoop workloads

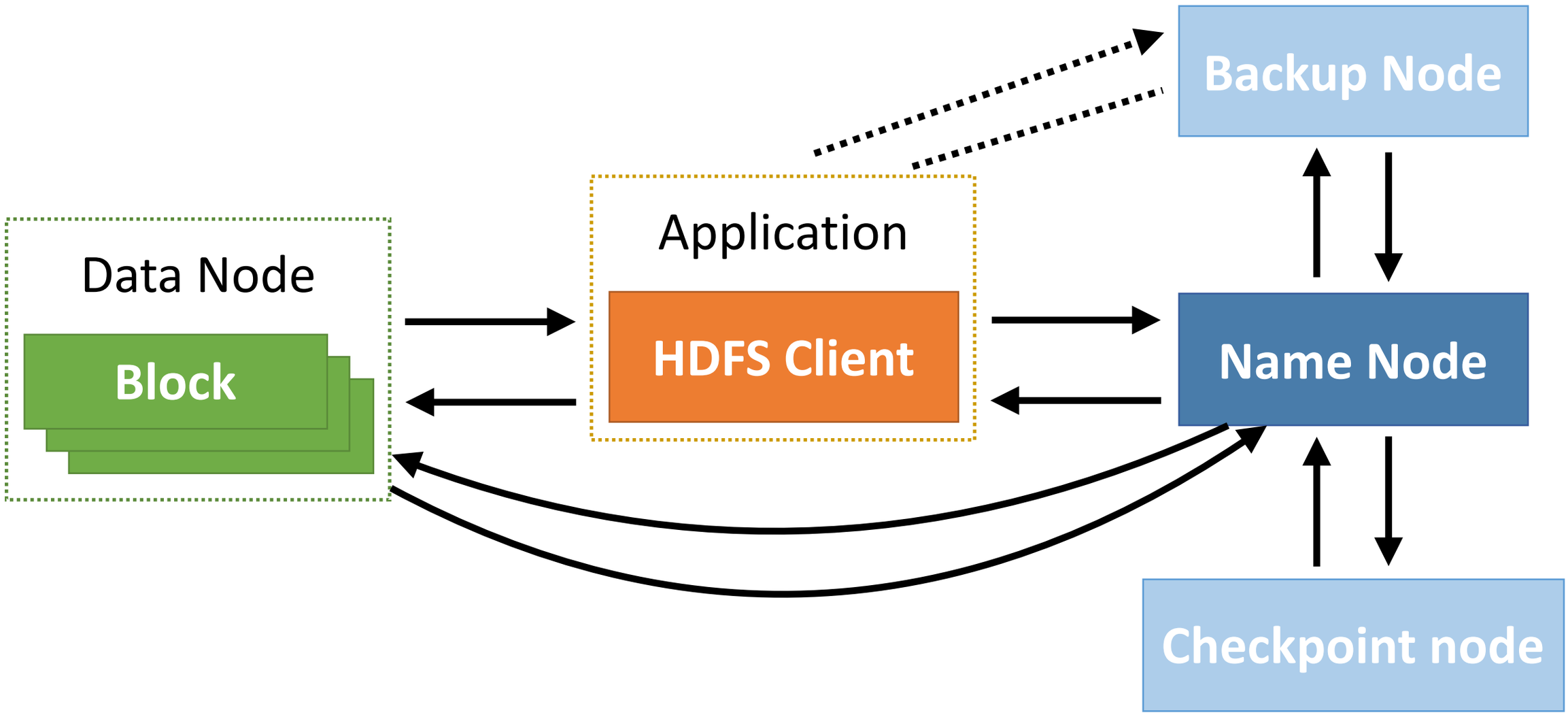
HDFS – Entities

GFS	HDFS
Chunk	Block
Chunkserver	Datanode
Master	Namenode
Client	Client
-	CheckpointNode
-	BackupNode





HDFS – Architecture



HDFS – Fault tolerance - NameNode

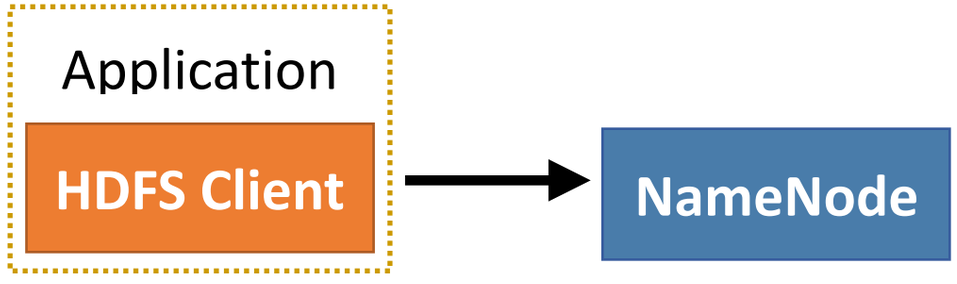
- CheckpointNode
  - Keeps a persistent replica of the journal
  - Truncates journal for faster recovery
- BackupNode
  - Same as CheckpointNode
  - Can act as a read only NameNode, requires a load balancer

HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends

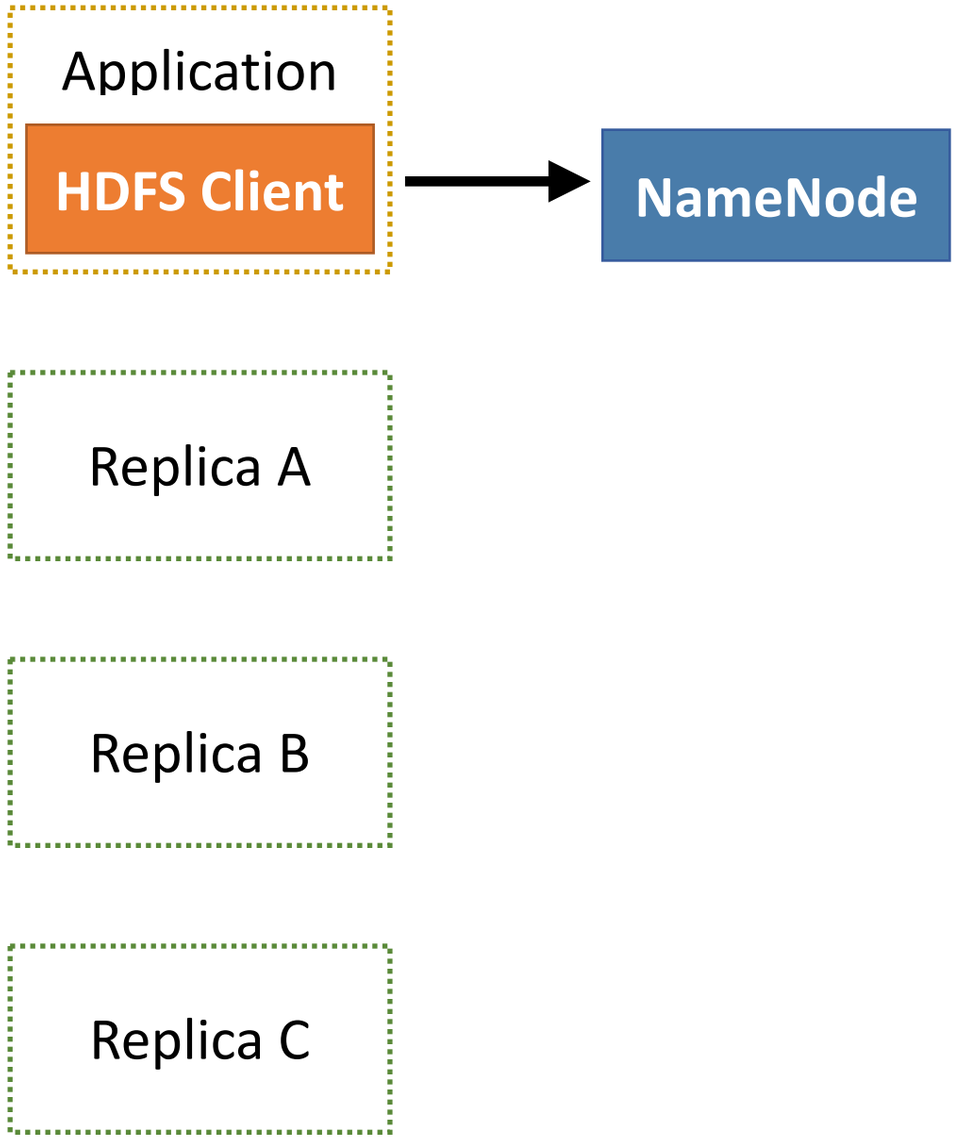
HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



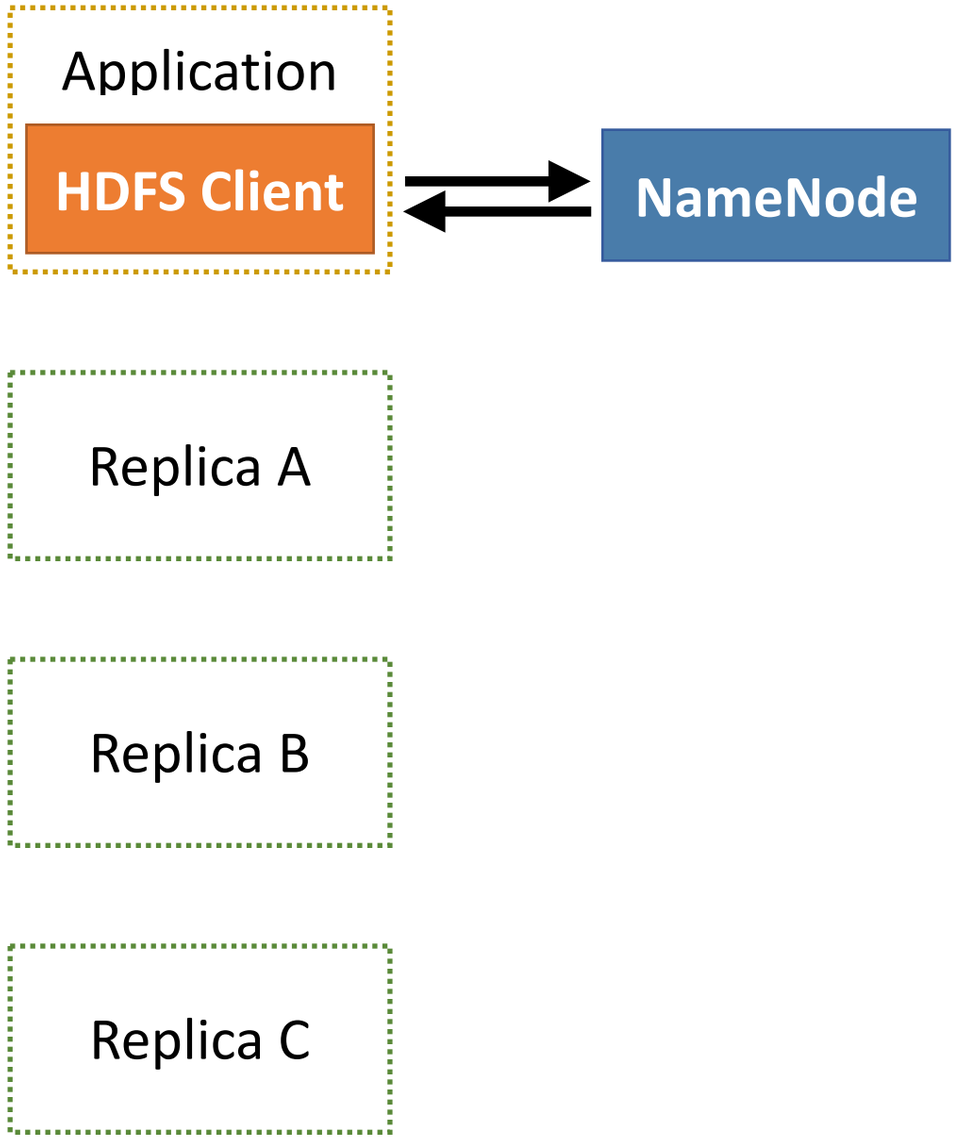
HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



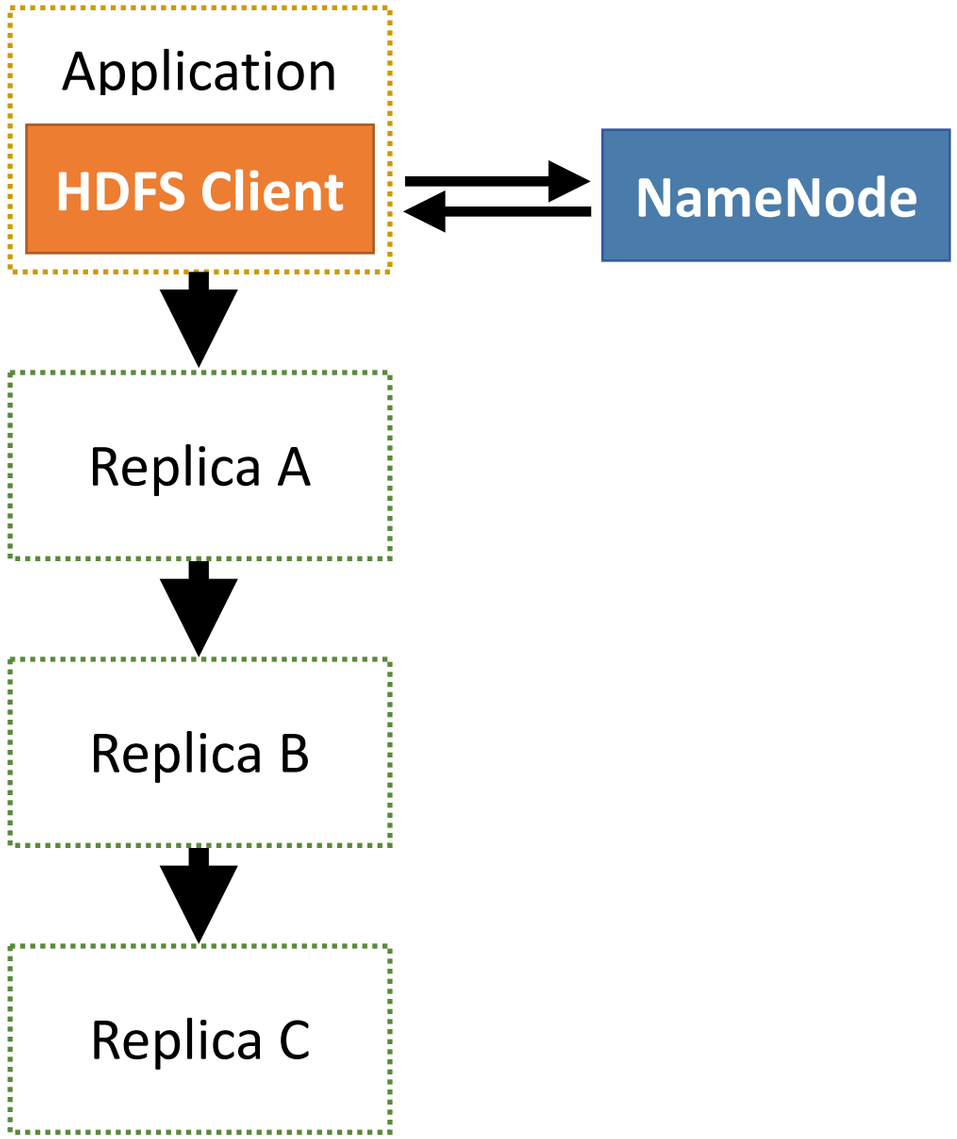
HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



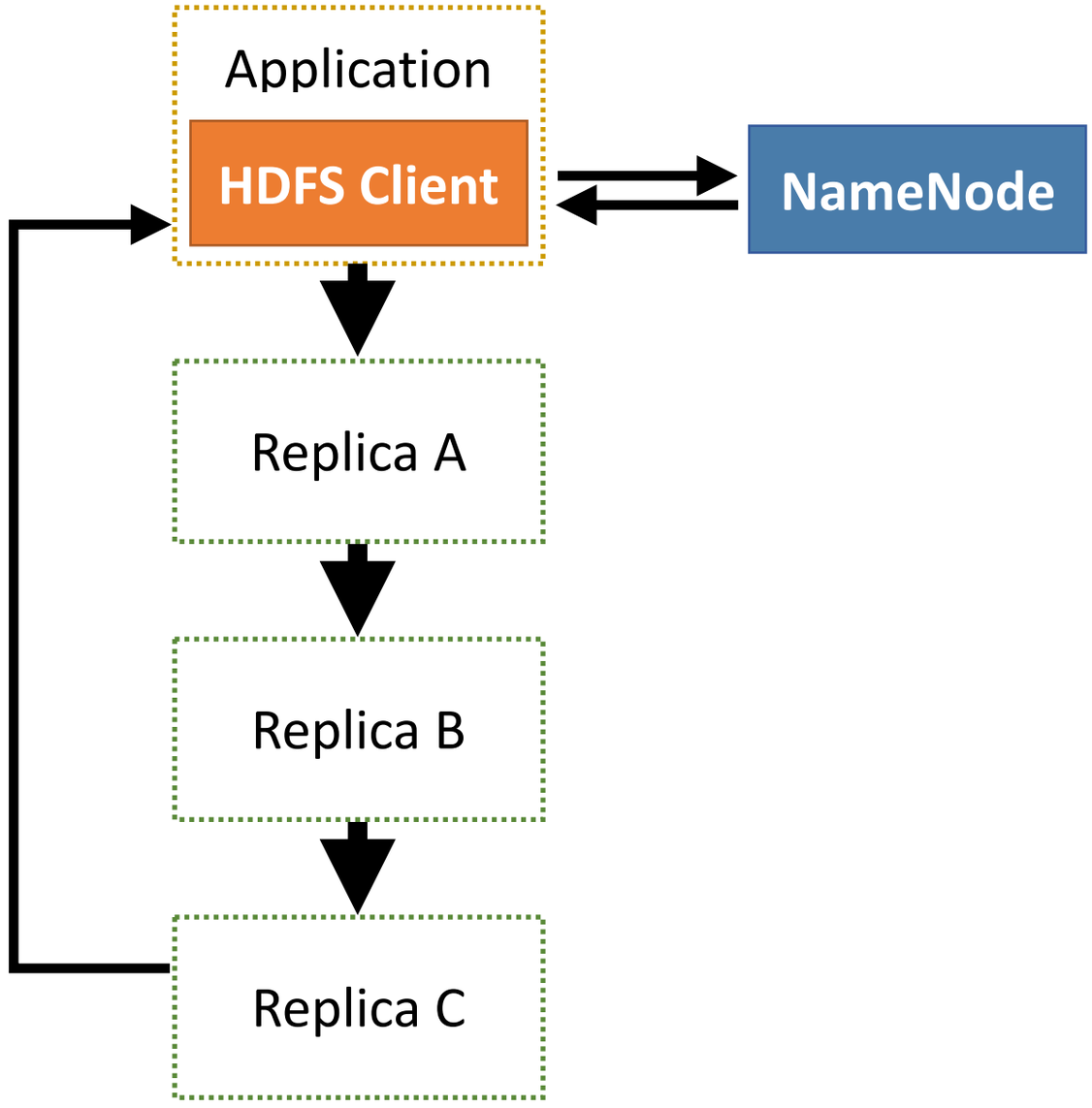
HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



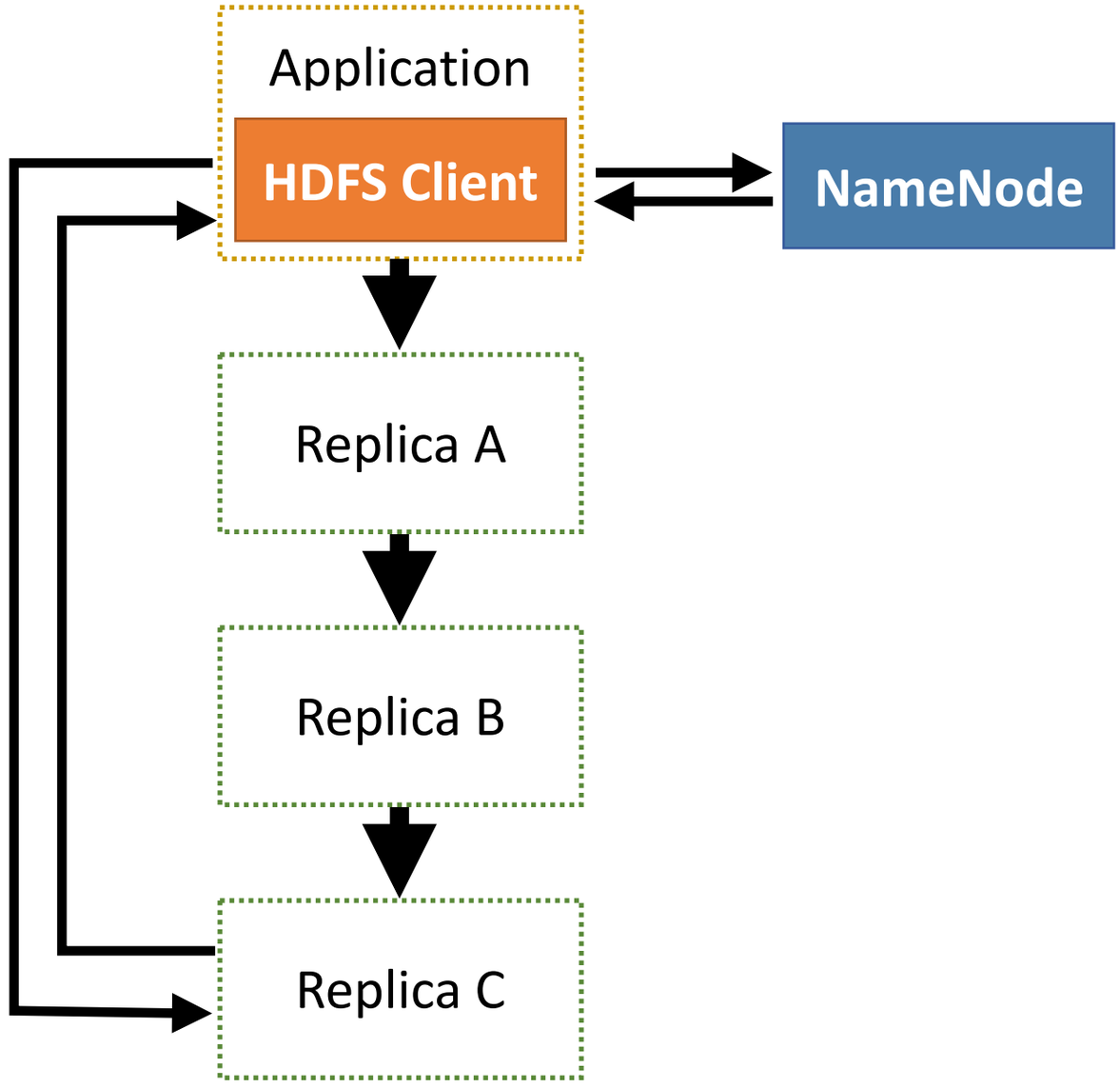
HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



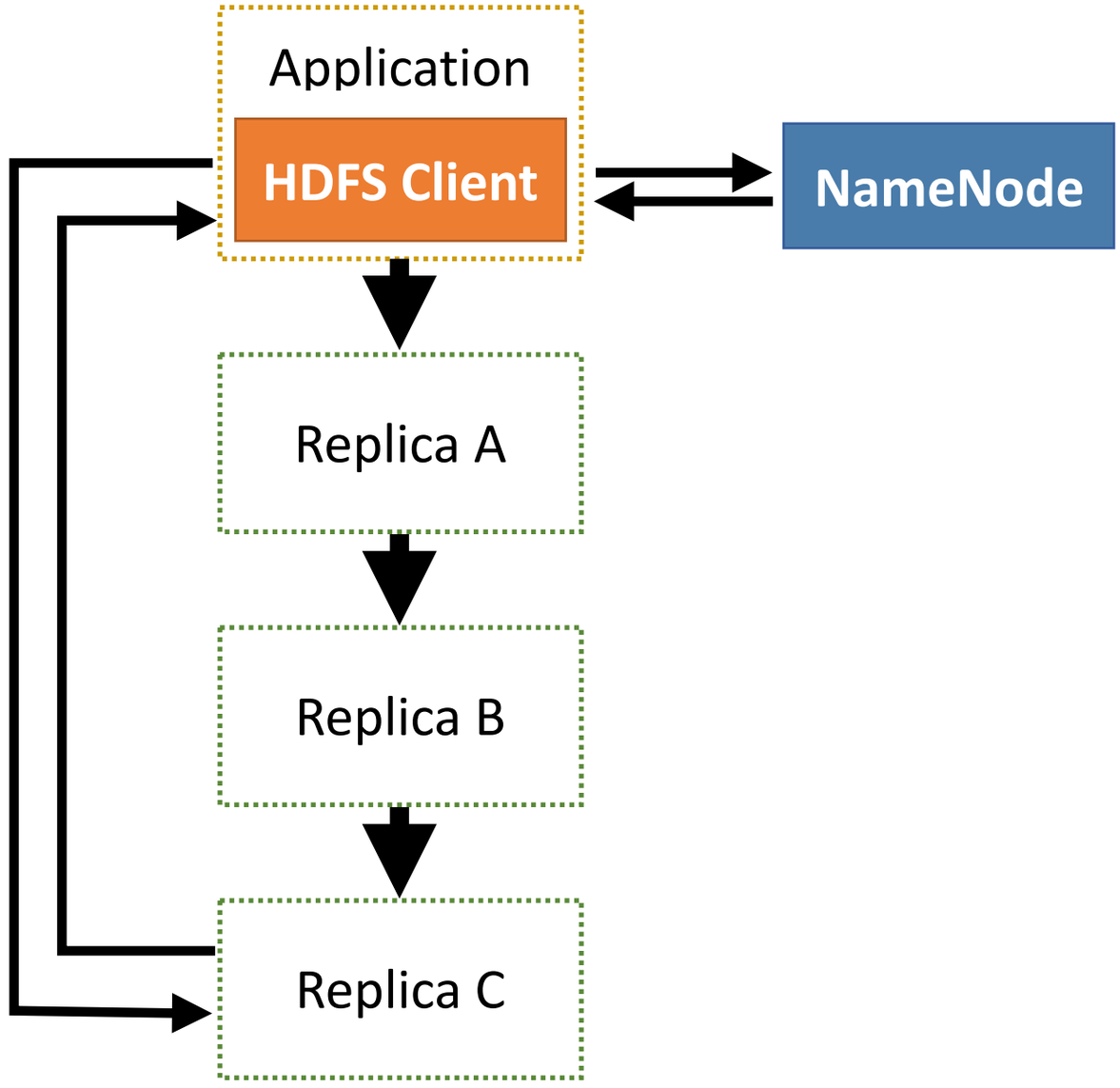
HDFS – Write

1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



HDFS – Write

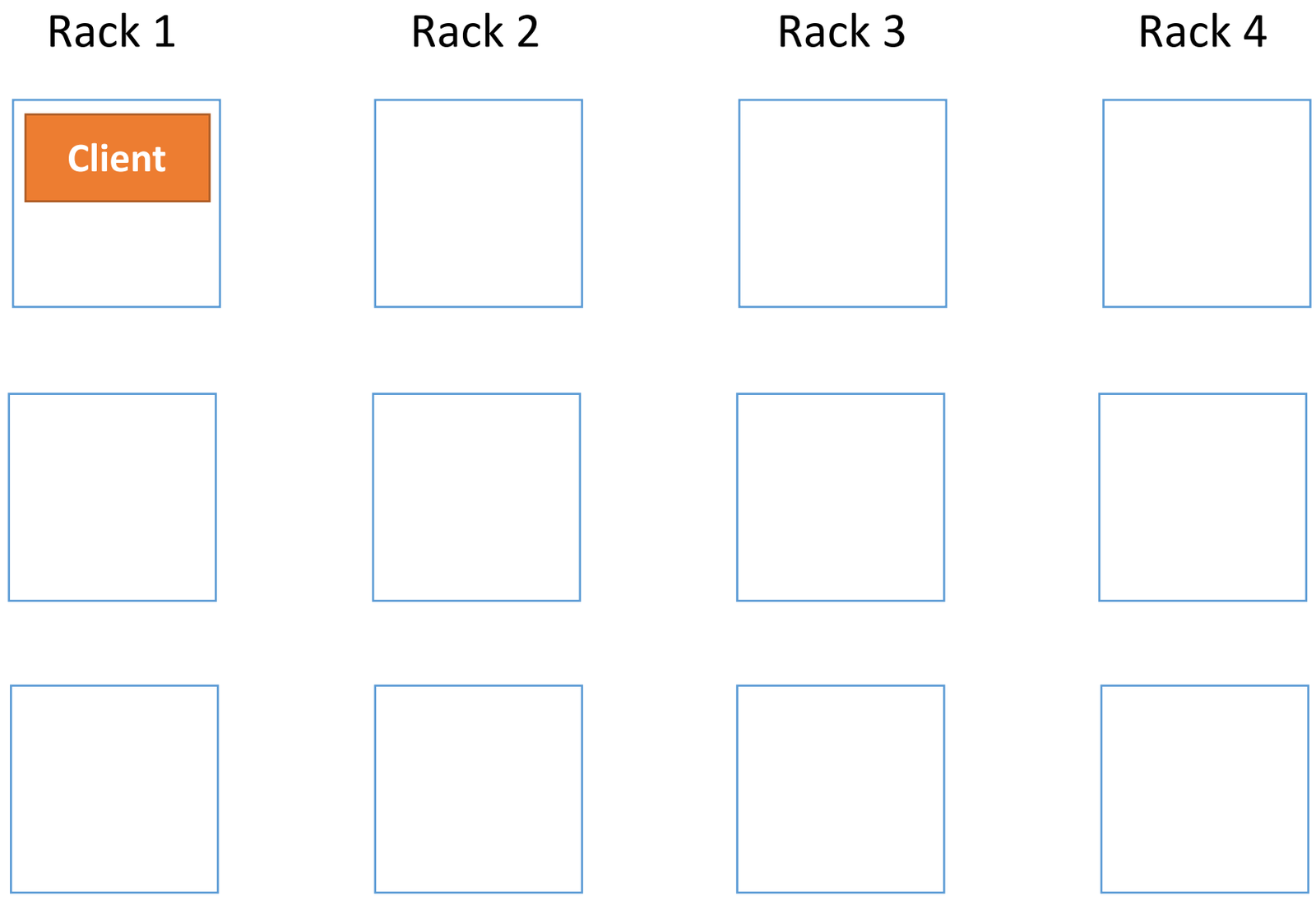
1. Client asks NameNode for replicas
2. NameNode evaluated lease validity
3. NameNode replies with identity of the Replicas (IP address)
4. Client pipelines data to replicas from the closest
5. Client send “close file”
6. Last replica responds
7. Lease ends



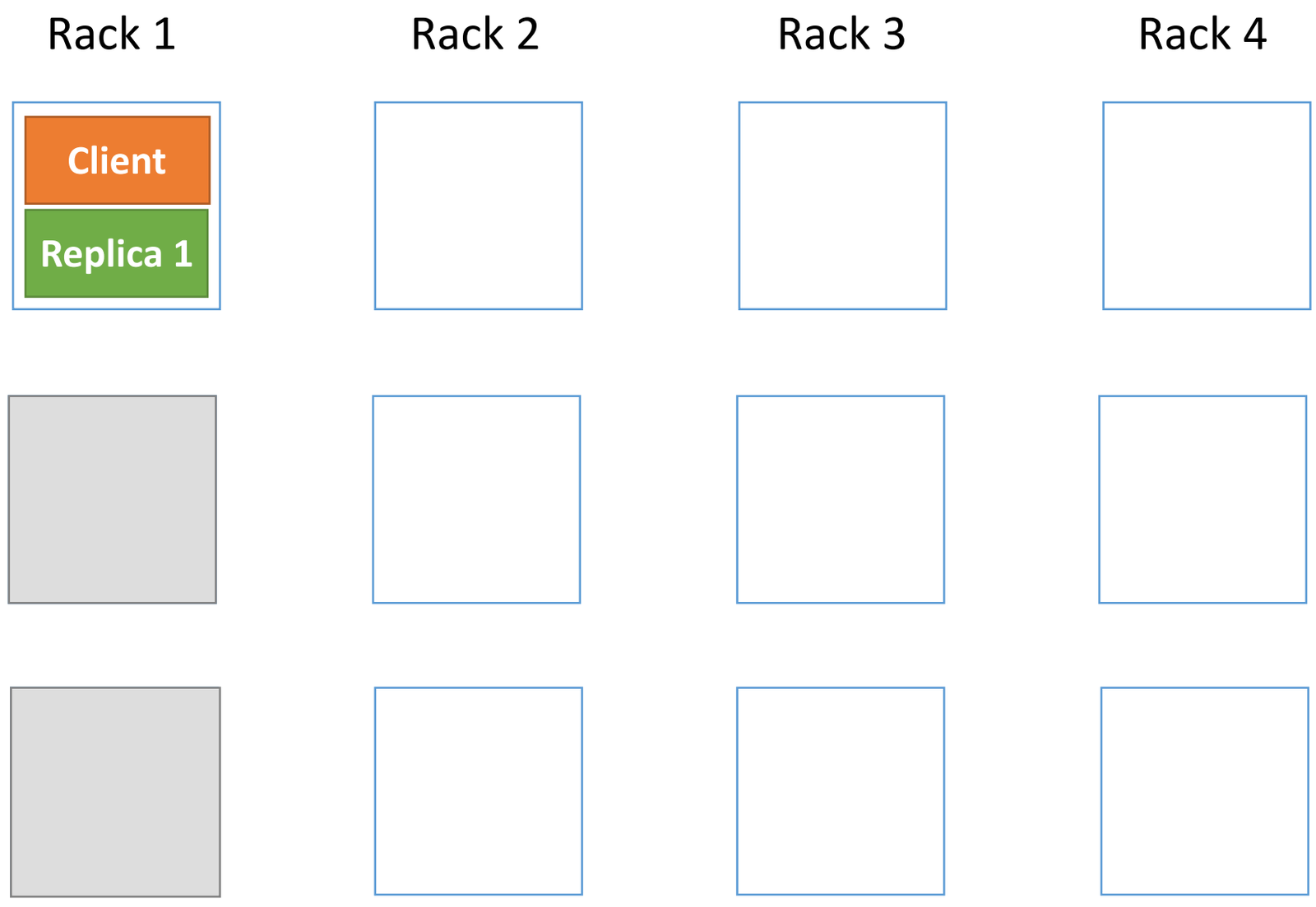
HDFS – Block placement

- Client always tries to read from the DataNode with least separation. Based on IP addresses.
- Does not take into account DataNode disk utilisation
- When creating a new file
  - 1st replica is placed on the same node as the client
  - 2nd and 3rd replicas are placed on different racks
  - No rack contains more than two replicas
  - No DataNode contains more than one replica

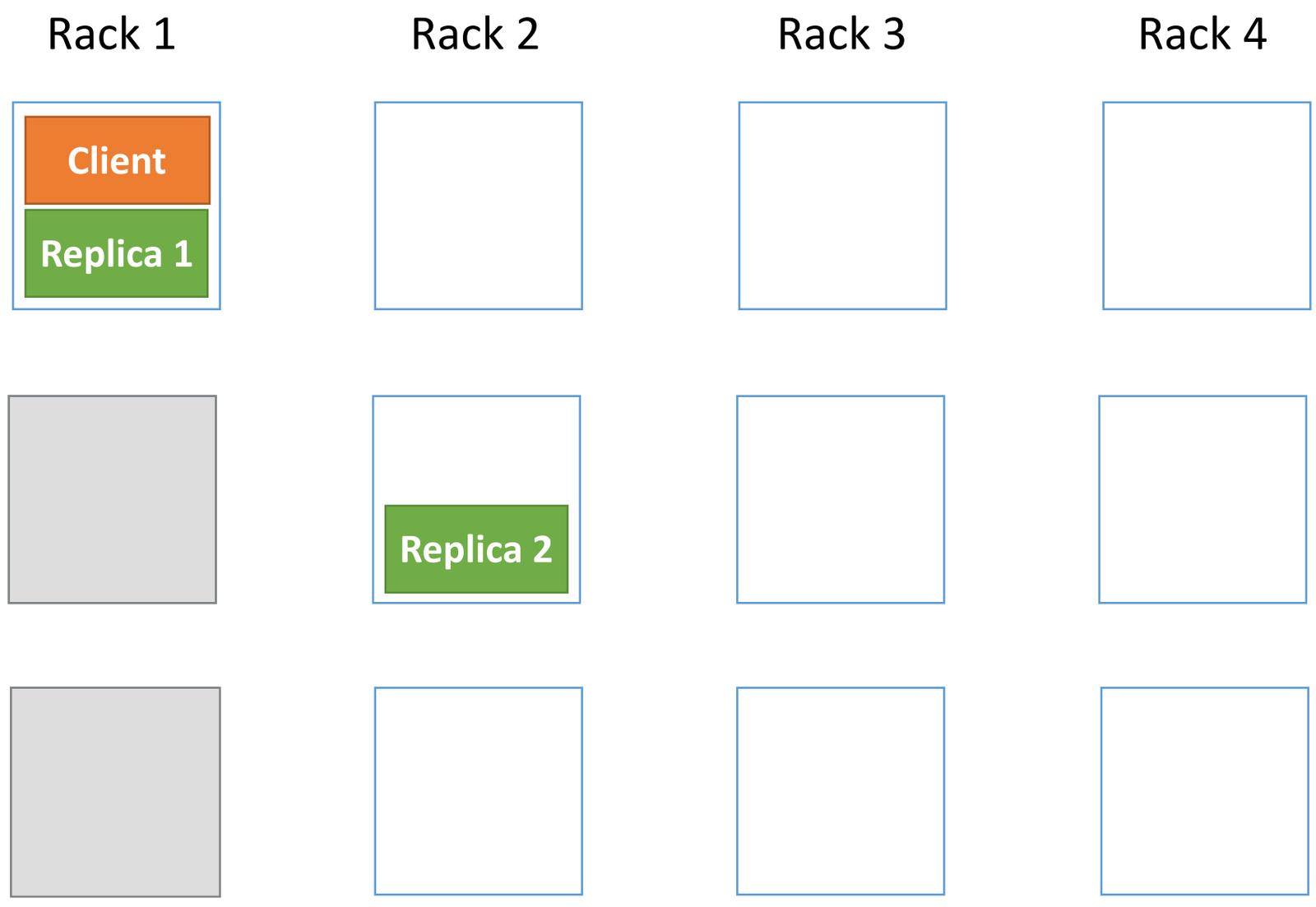
HDFS – Block placement



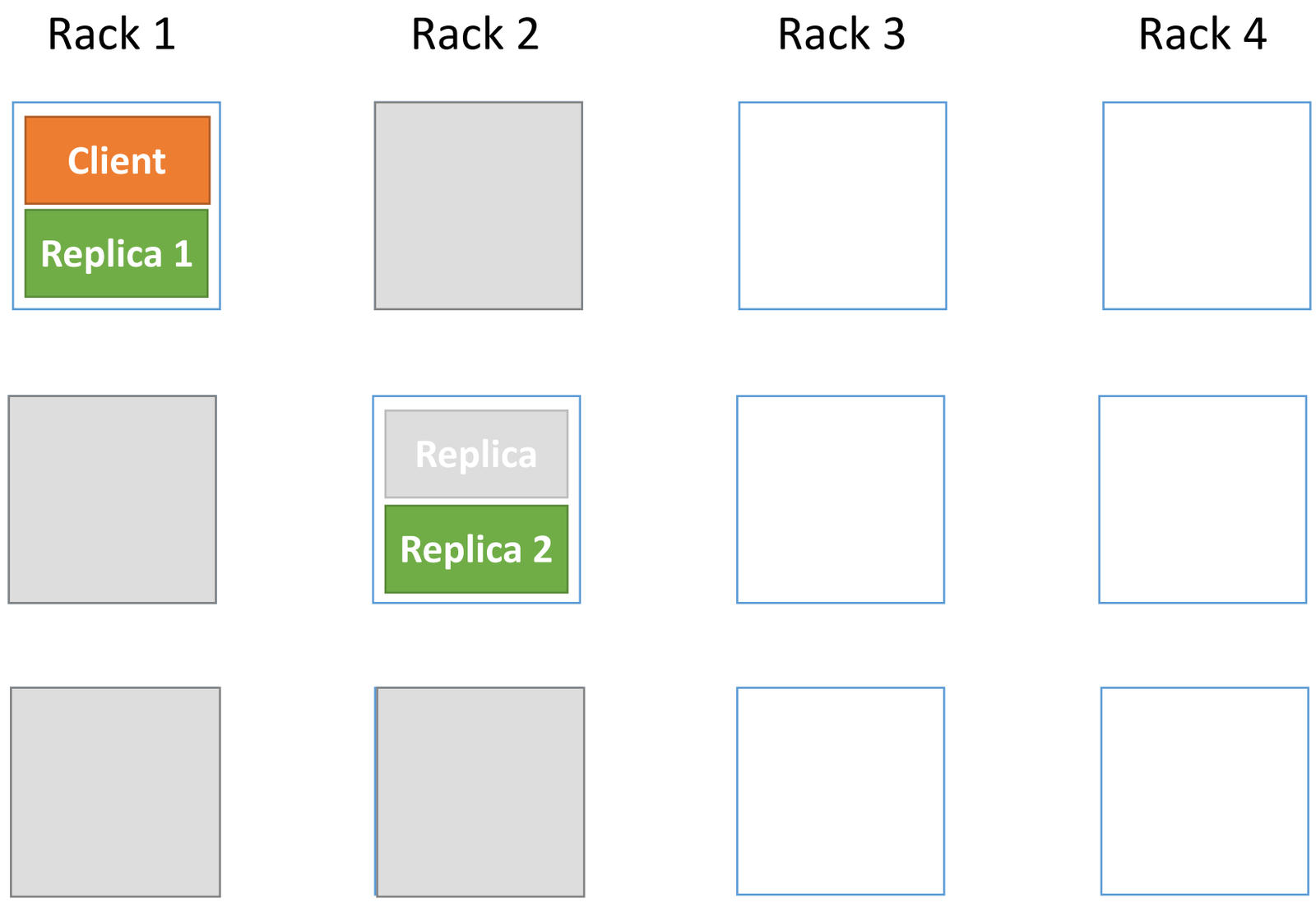
HDFS – Block placement



HDFS – Block placement

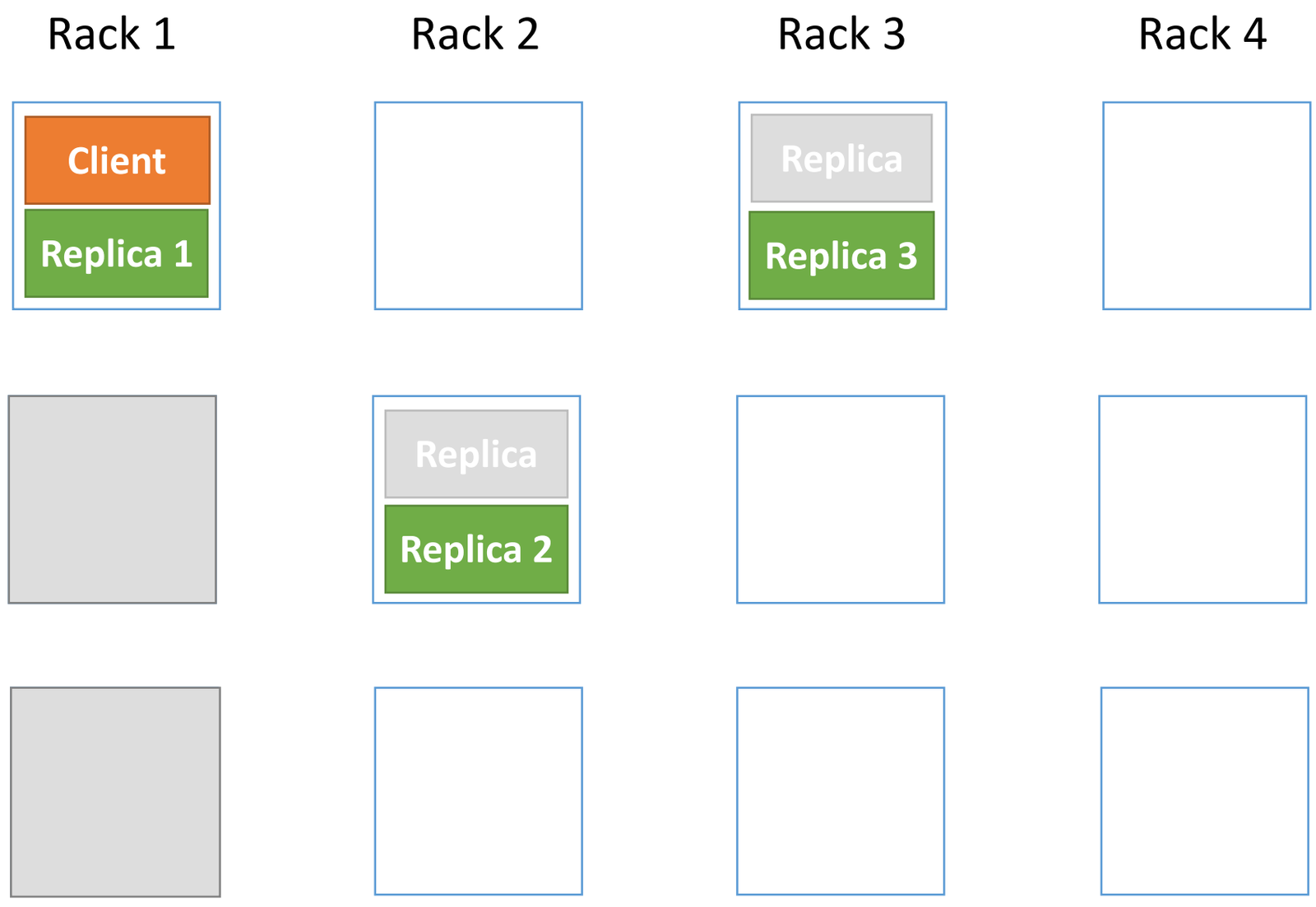


HDFS – Block placement

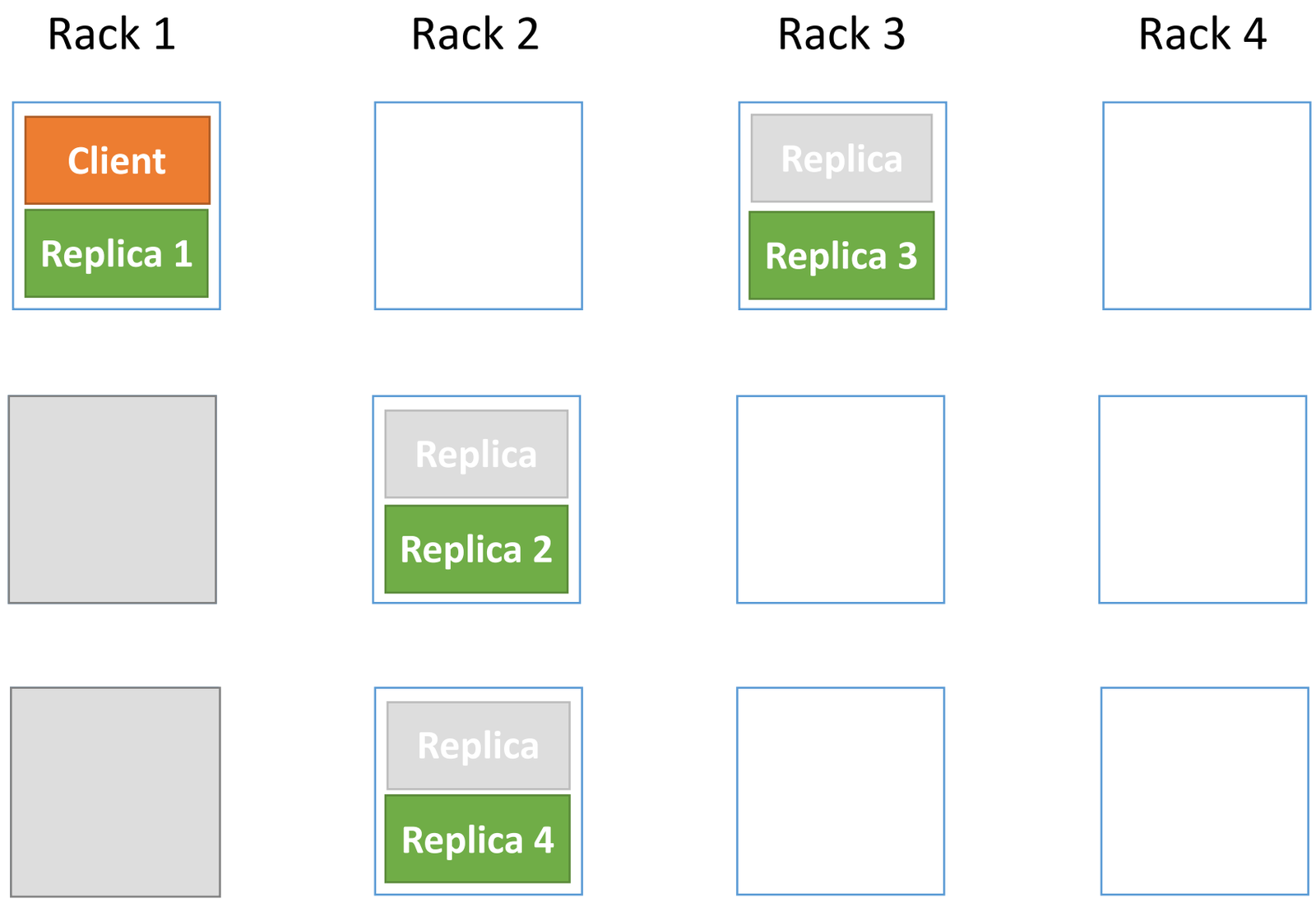


HDFS – Block placement

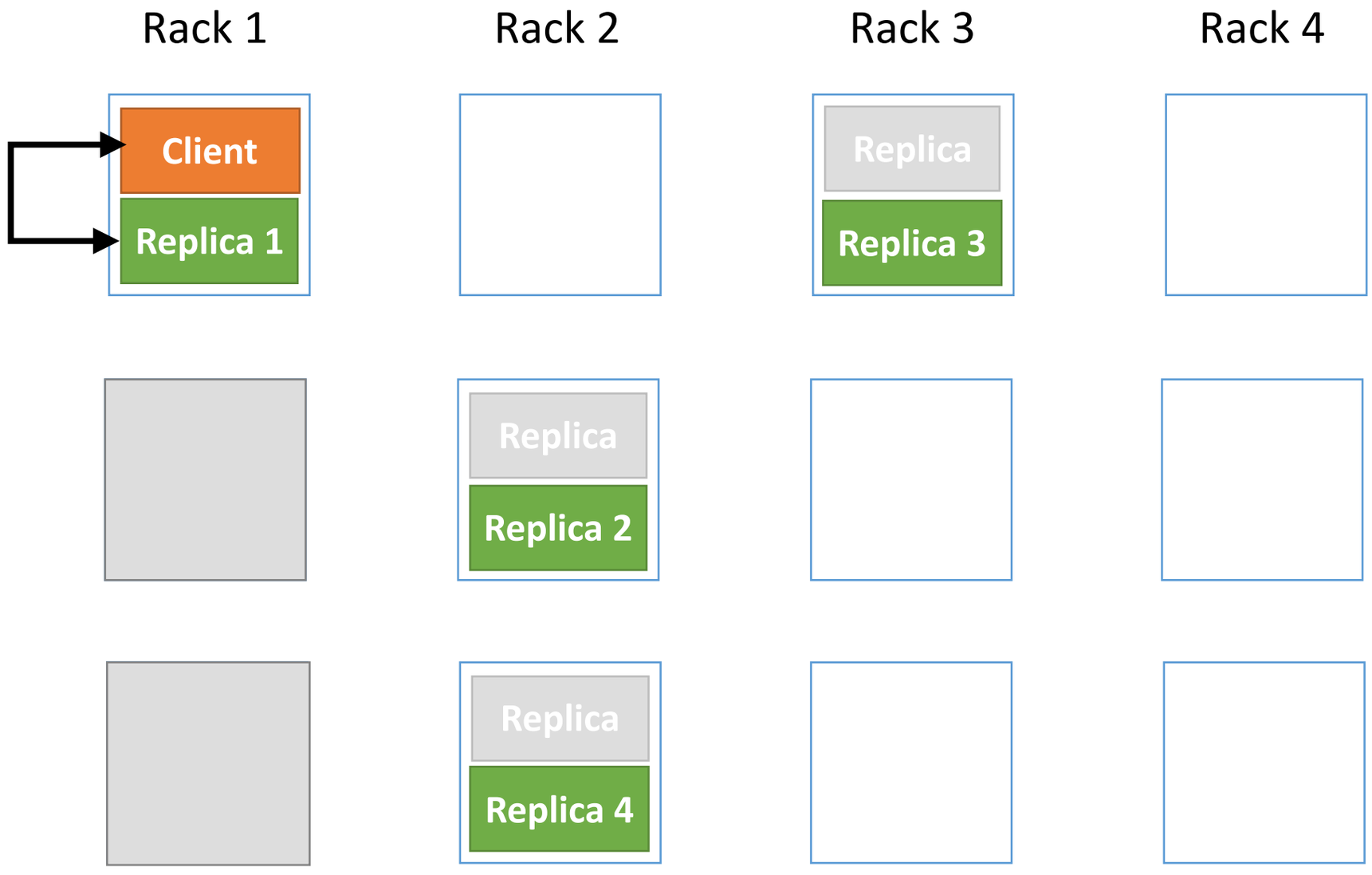
HDFS – Block placement



HDFS – Block placement



HDFS – Block placement

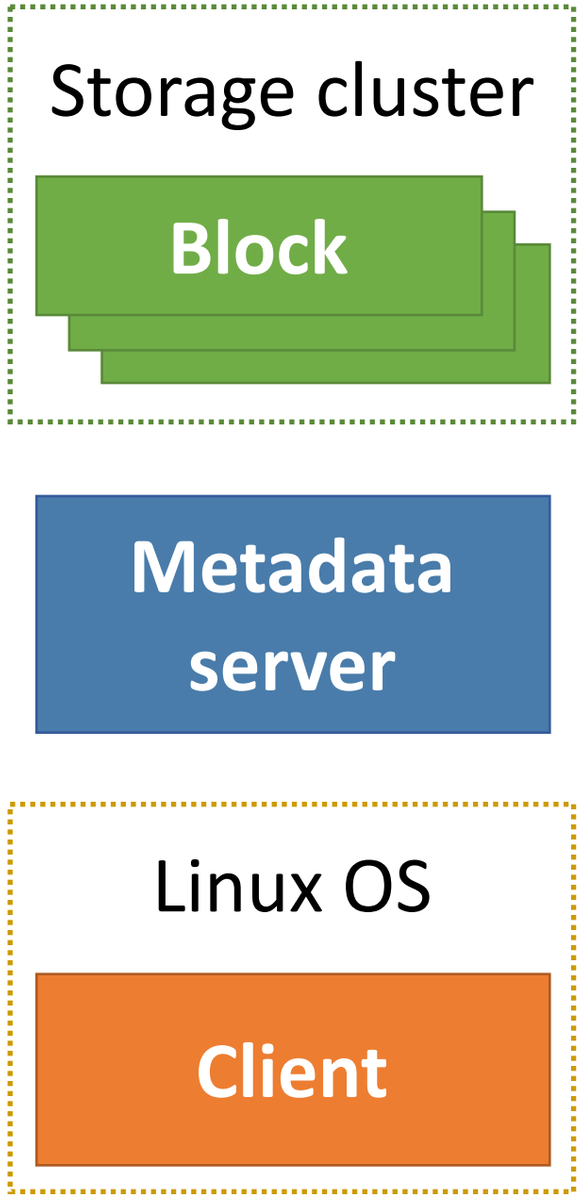


Ceph

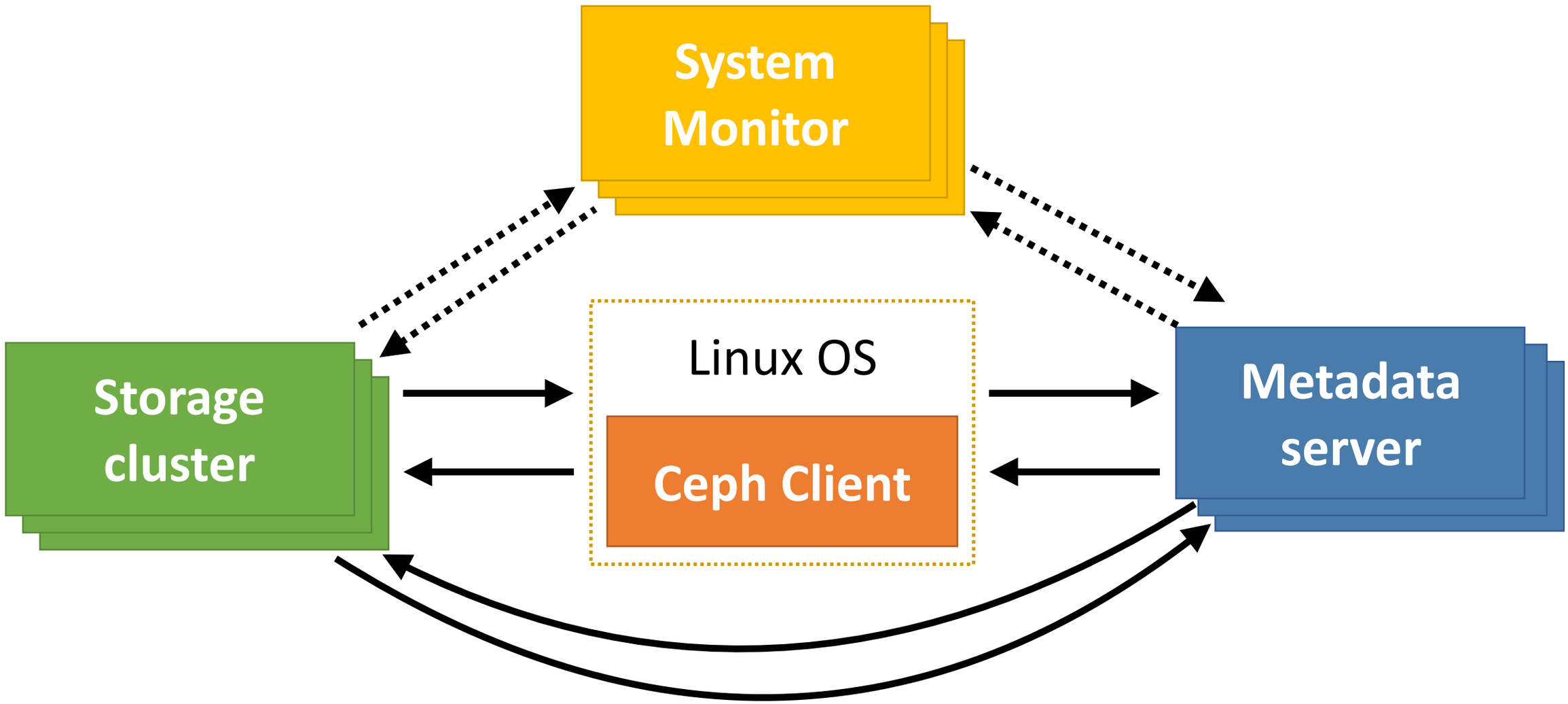
- Integrated into Linux kernel, transparent to application
- Maintained by RedHat
- Found in Fedora, Ubuntu, Debian, etc.
- More general than the other two
- Scalable through higher degree of distribution
  - Multiple metadata nodes, partitioned namespace
  - Multiple data/storage nodes
- Pseudo-random block placement
- Trusting client capability implementation

Ceph – Entities

GFS	HDFS	Ceph
Chunk	Block	Block
Chunkserver	Datanode	Storage cluster
Master	Namenode	Metadata server
Client	Client	Client
-	CheckpointNode	-
-	BackupNode	-
-	-	System monitor

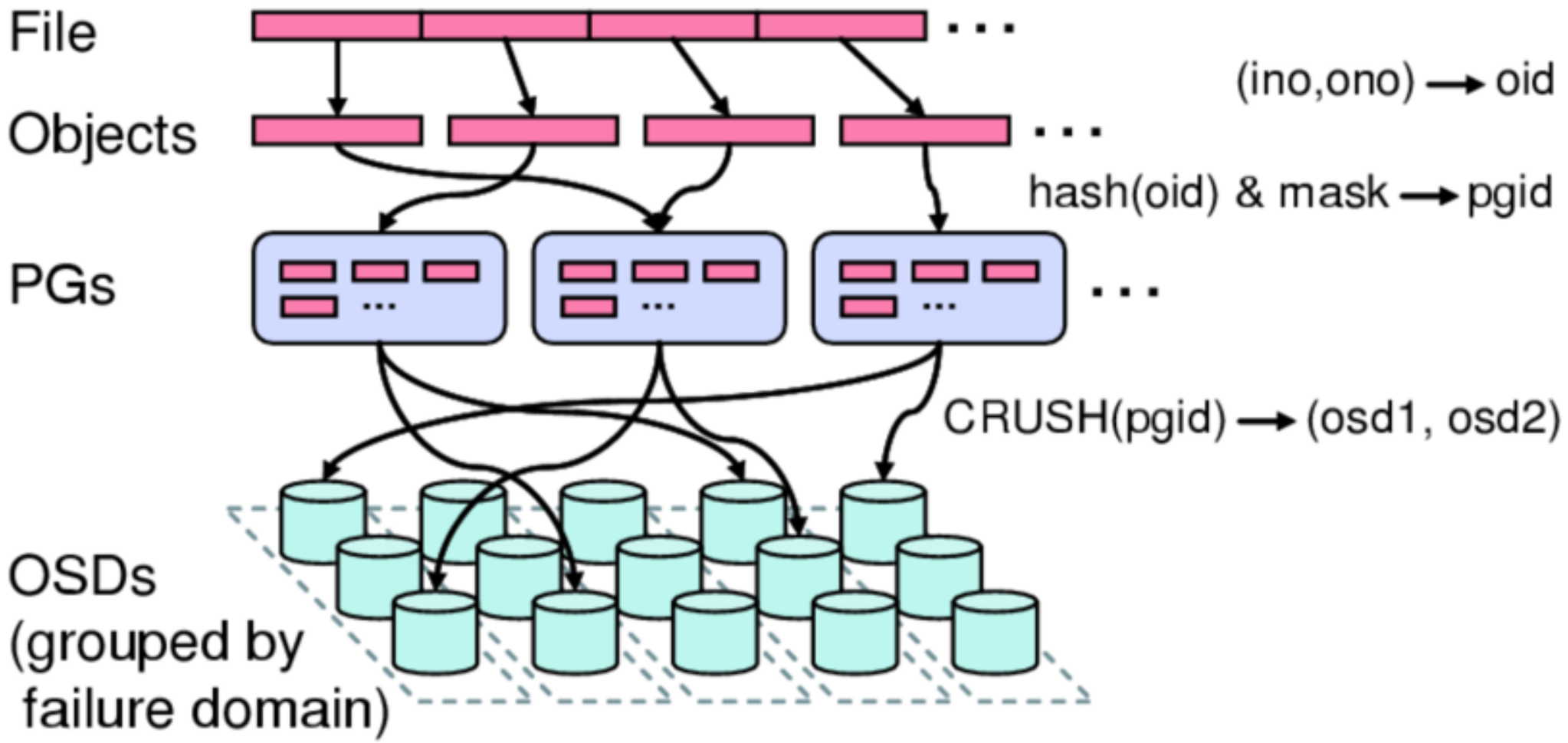


Ceph – Architecture

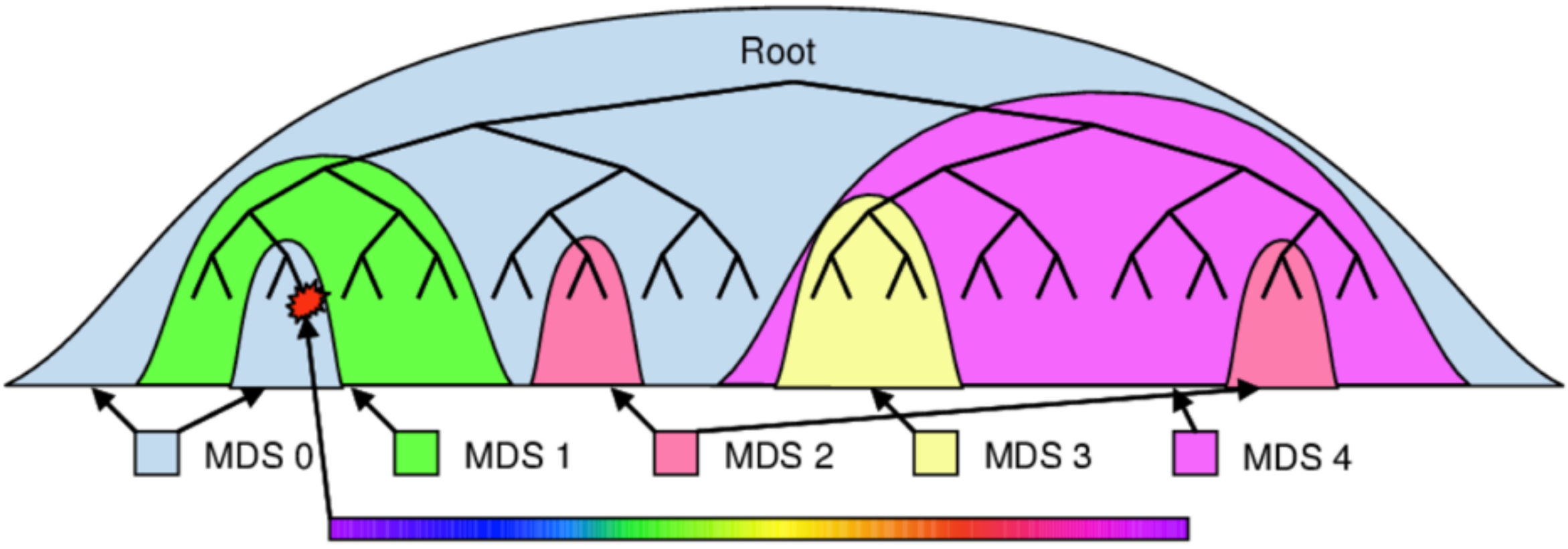


Ceph – Metadata (CRUSH)

Controlled Replication Under Scalable Hashing



Ceph – Namespace partitioning



Busy directory hashed across many MDS's

Conclusion

- GFS
  - Bespoke, Google centric
  - Stereotypically google-simple
  - Not very adaptive
- HDFS
  - More control signalling
  - More tuning parameters
- Ceph
  - Most general
  - Higher degree of distribution
  - Most configurable but also most adaptive