Workflow of Write Operations in GFS

Learn how GFS performs write operations.

We'll cover the following

We've discussed how GFS carries out file operations, and we have seen the two of them (create and read files) in the previous lesson. In this lesson, we will look into what kind of write operations GFS supports and the workflow of these operations.

Writing data to the file

Data can be written at a random position in the file or appended to the file in case of sequential writes. GFS offers two operations–a random write and an append operation. In a **random write operation**, the client provides the offset at which the data should be written. In an **append operation**, the data is written at the end of the file at the GFS chosen offset.

For write operations, the GFS client needs to know which chunk they will be writing the data to. We've already seen that the manager keeps three replicas for each chunk by default. Among these three replicas, the manager gives one the lease. The replica that has the current lease acts

as the primary node, while others act as
secondary replicas.
All write operations from the clients are carried out by the replica that has the current lease, called the **primary replica**. The primary replica then forwards the request to secondary replicas and replies to the client once the request has been processed. Since it is the manager who decides which replica will take the lease, therefore, replicas don't need any election algorithms to choose a primary between them.

> **Note: Leases** are a fault-tolerance mechanism that enable the manager to provide good availability under different kinds of failures. Lease value should not be either too large (in which case if a server takes a lease and dies, manager might need to wait out for the lease to expire before it can give it to some other server) or too small (in which case manager might get excessive traffic for lease renews). A good lease value usually depends on specific use cases and evolves with the operational experience with the system.

Question

How are leases easing the availability of the system?

Hide Answer

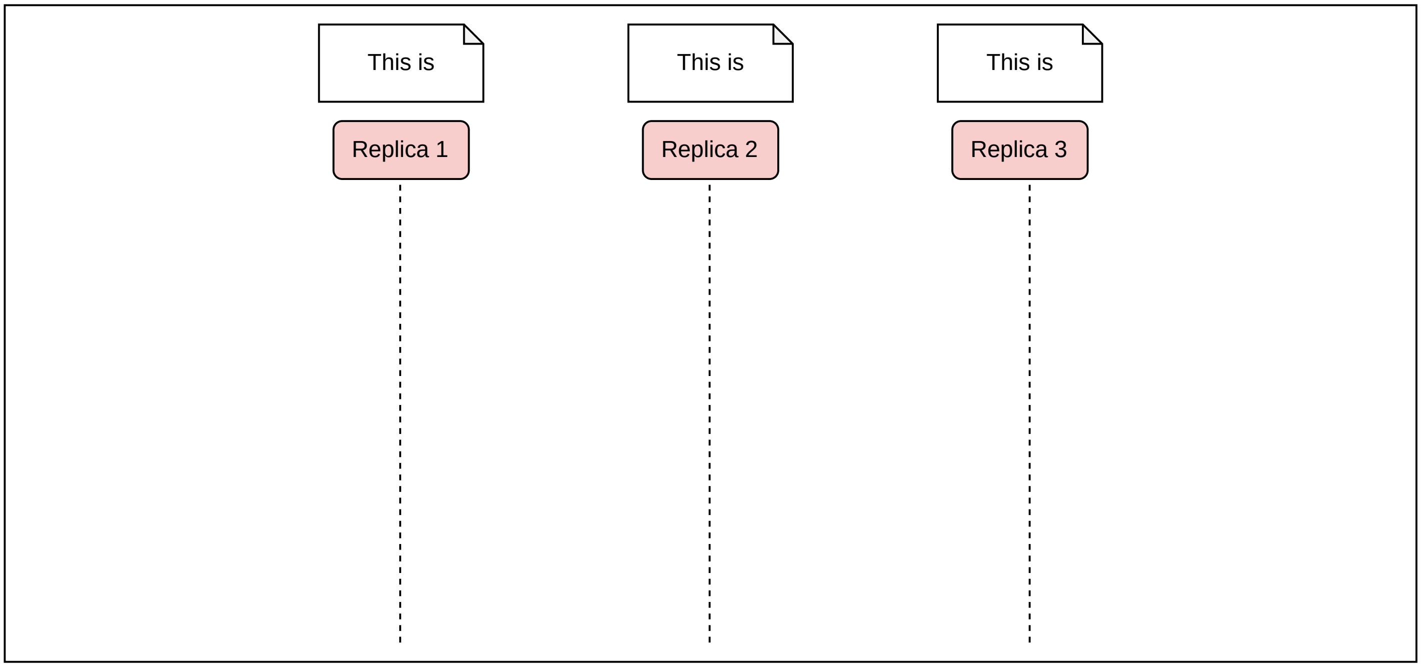
Since the manager has the authority to choose the primary replica for the clients, it grants a lease to a healthy replica, not a failed replica. Still, it is possible that everything was okay when the manager handed out the lease, and a client was about to contact the primary (the leaseholder) when the primary died. The client will go back to manager because the connection/request will fail/time-out.

> Lease values are usually small, like a few seconds to a few minutes. If the manager doesn’t do anything proactively, the lease will expire, a different replica will be given the lease, and a new node will be found to reach the replication factor of 3. This helps GFS to be available for the client requests with some small windows of unavailability (during lease reassignment and finding a new node to have enough replication factor) for writing to a specific chunk (not reading because reading can happen from other replicas).

In the next section, we will discuss what can go wrong if one of the replicas was not designated as a primary replica.

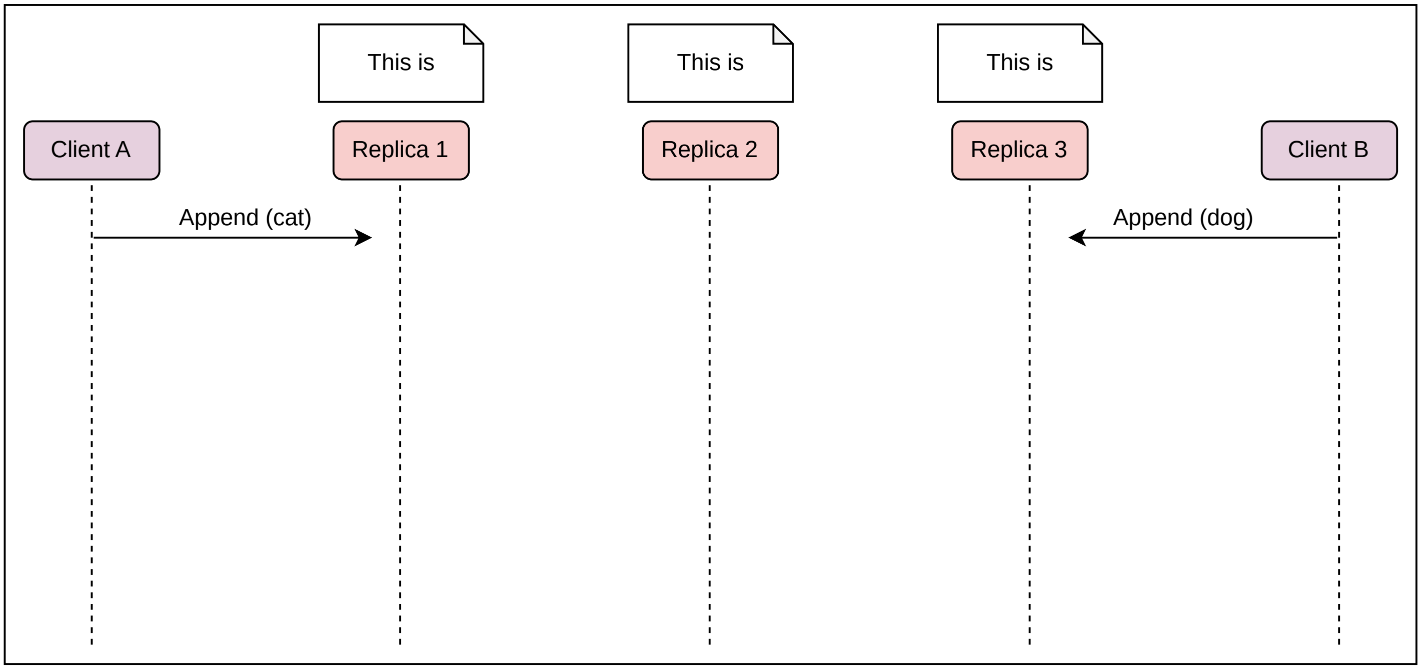
Problems with not designating a primary replica

Suppose the manager allows the clients to perform a write operation on any random replica. In that case, multiple write operations on the same chunk will be performed by different replicas simultaneously. Each replica that receives the write request from the client also has to propagate the write to the other replicas. These replicas can get the propagated write operations out of order. Each replica executes the same set of operations in a different order. Thus, the replicas will contain different data. However, the replicas should actually be identical. The following illustration depicts this issue through an example.



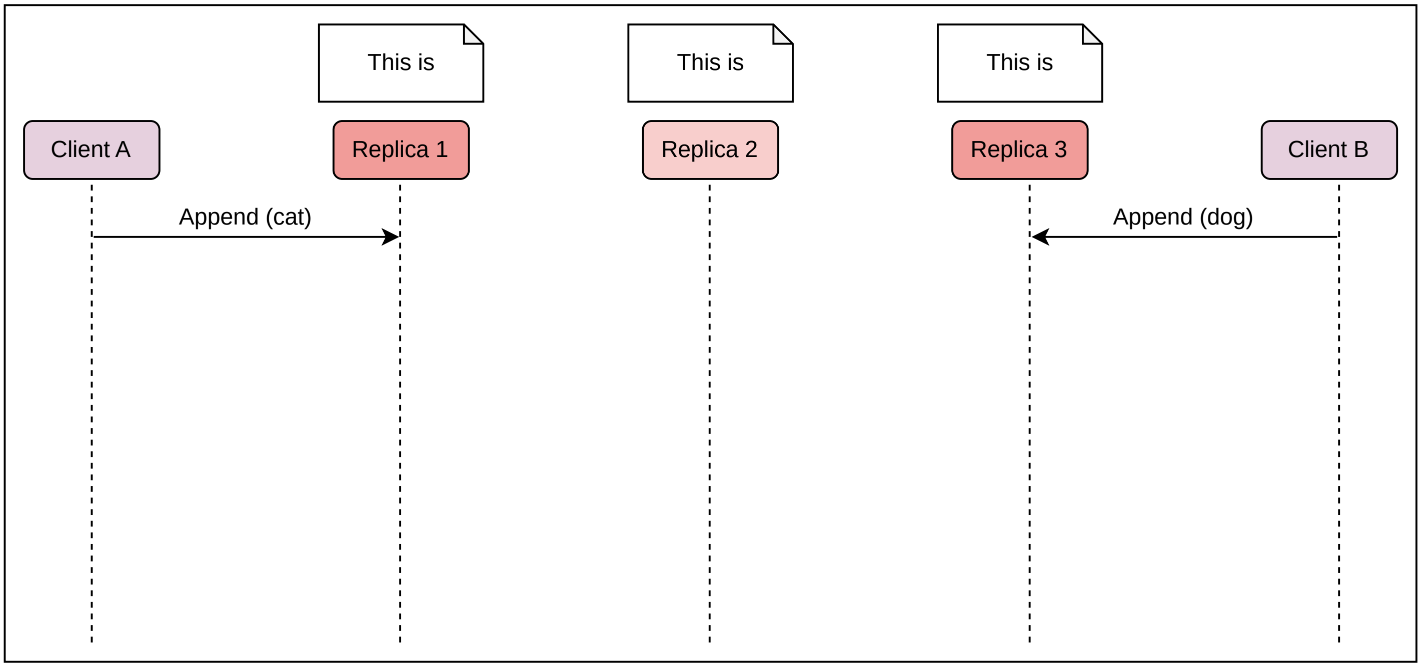
There are three replicas hosting the same data

1 of 13



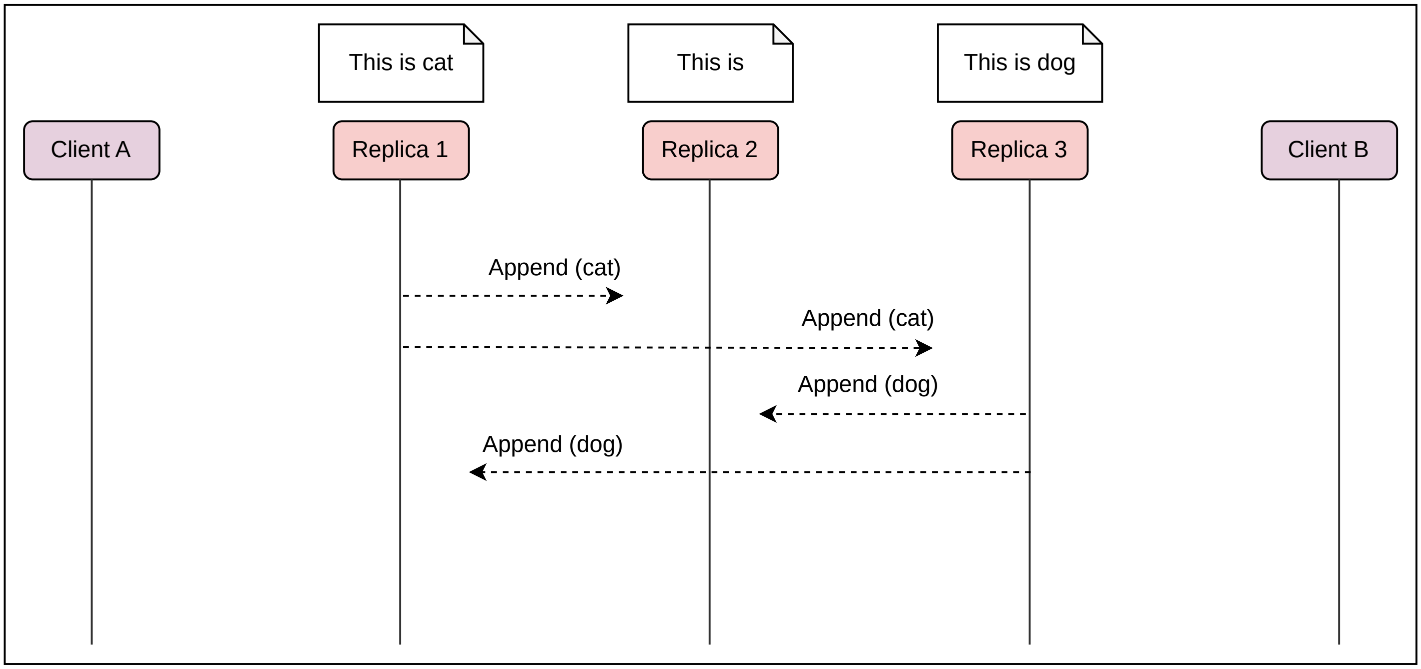
Two clients concurrently perform append operation on the same chunk

2 of 13



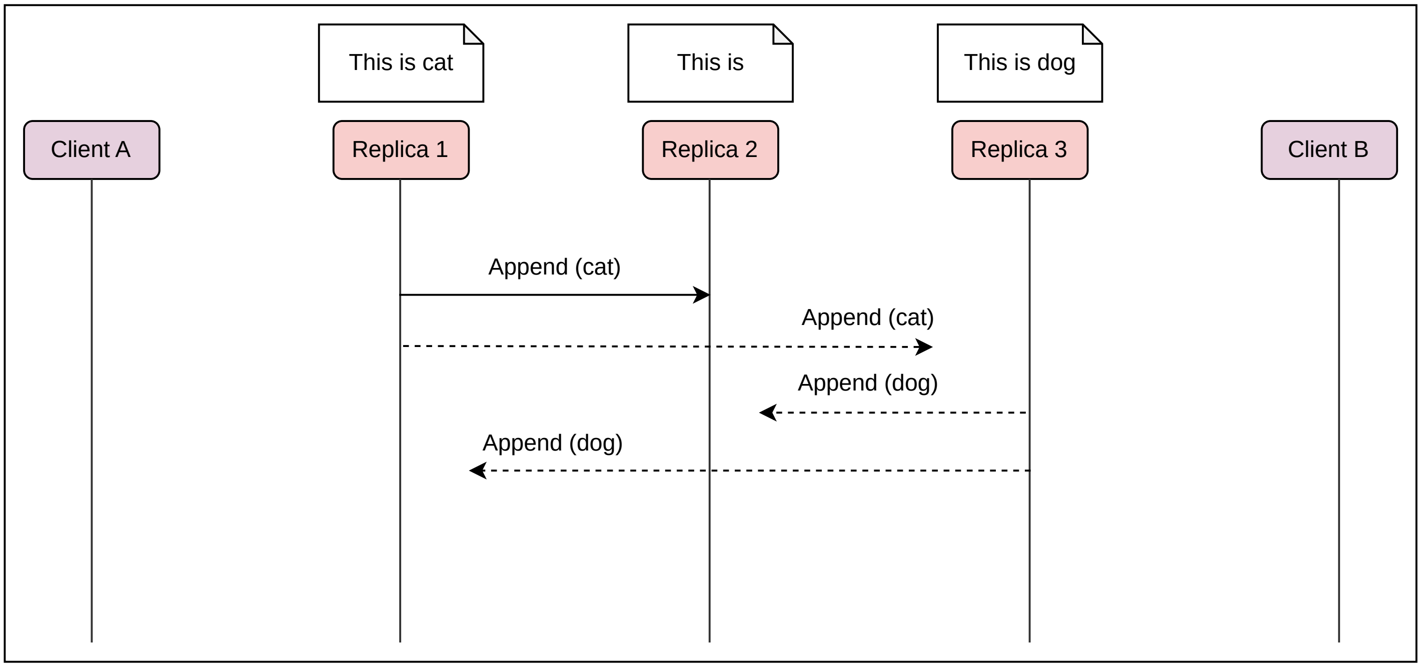
Client A's request is processed at Replica 1 while Client B's request is processed at Replica 3

3 of 13



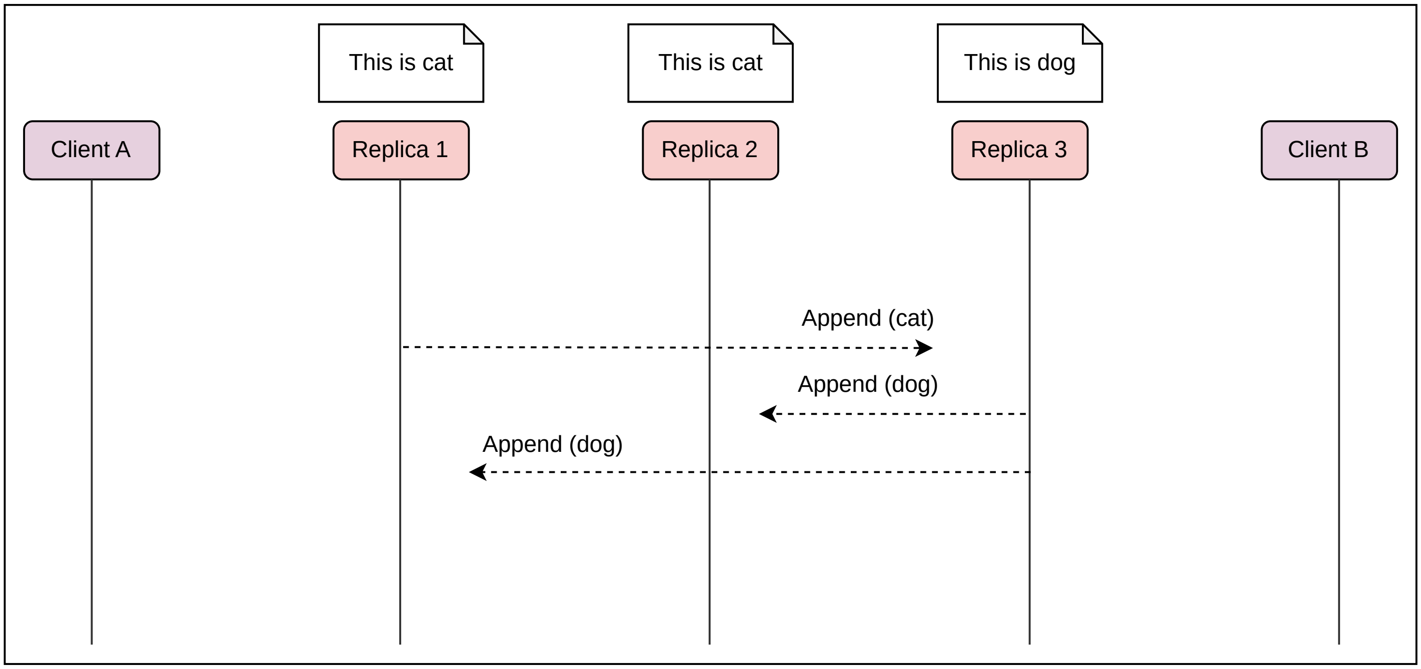
Replica 1 and Replica 3 update the data and propagate the append request to other replicas

4 of 13



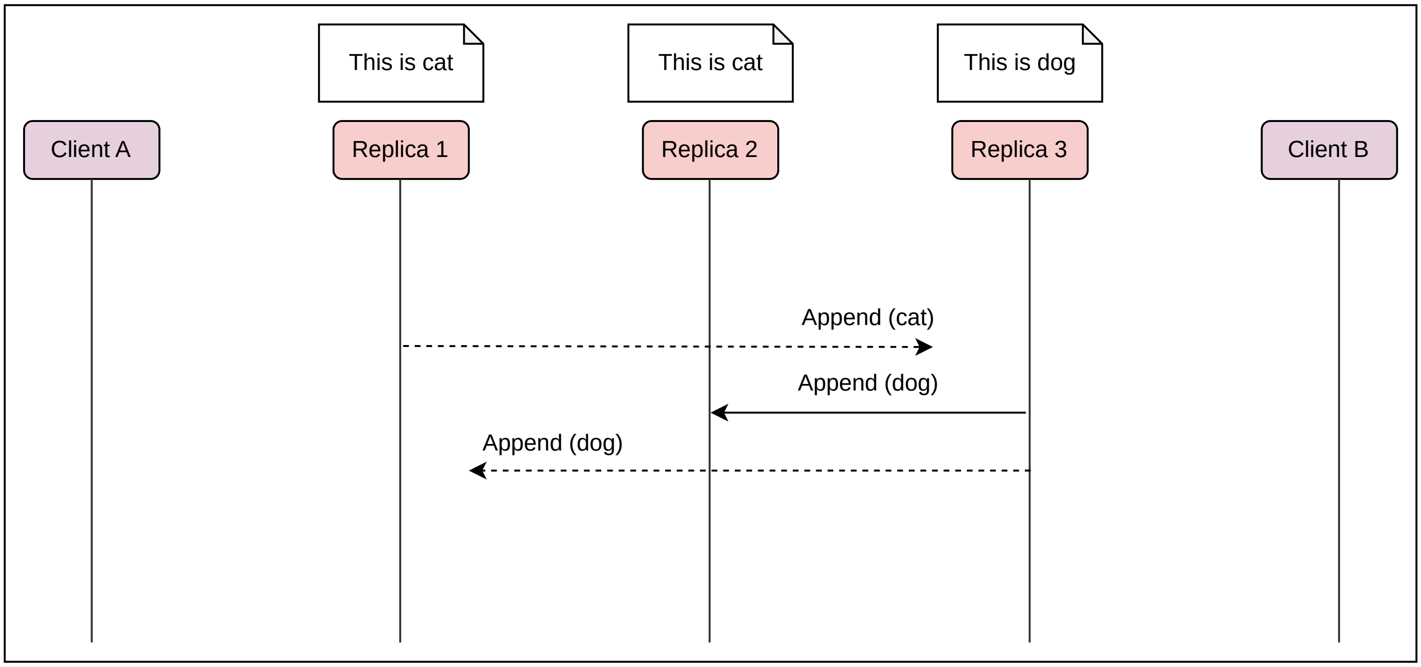
Replica 2 receives Client A's append operation propagated by Replica 1

5 of 13



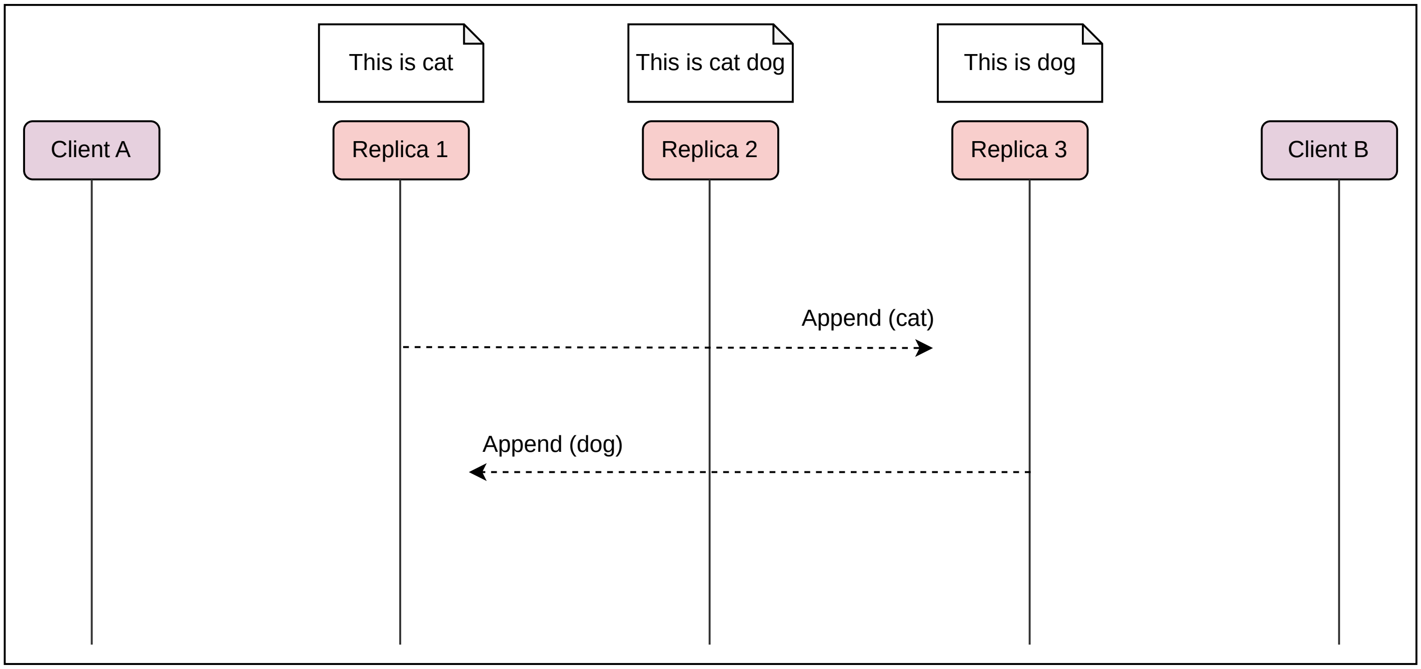
Replica 2 performs Client A's append operation propagated by Replica 1

6 of 13



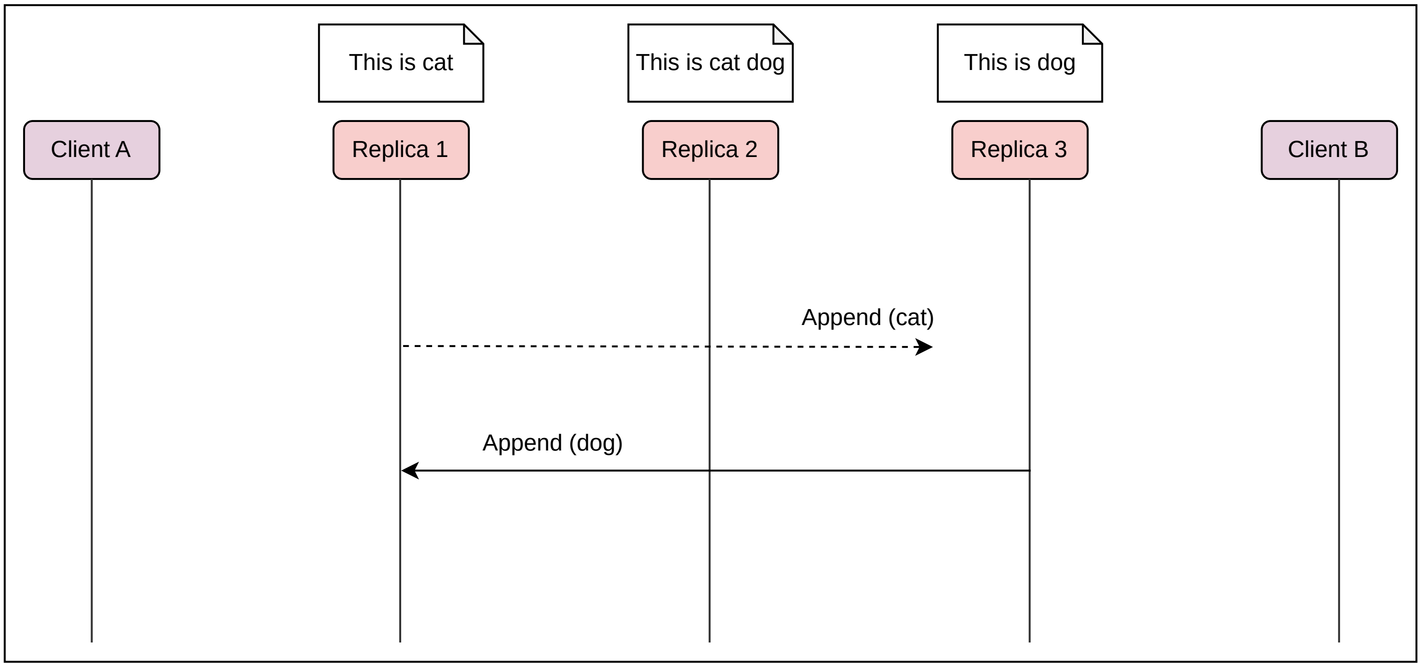
Replica 2 receives Client B's append operation propagated by Replica 3

7 of 13



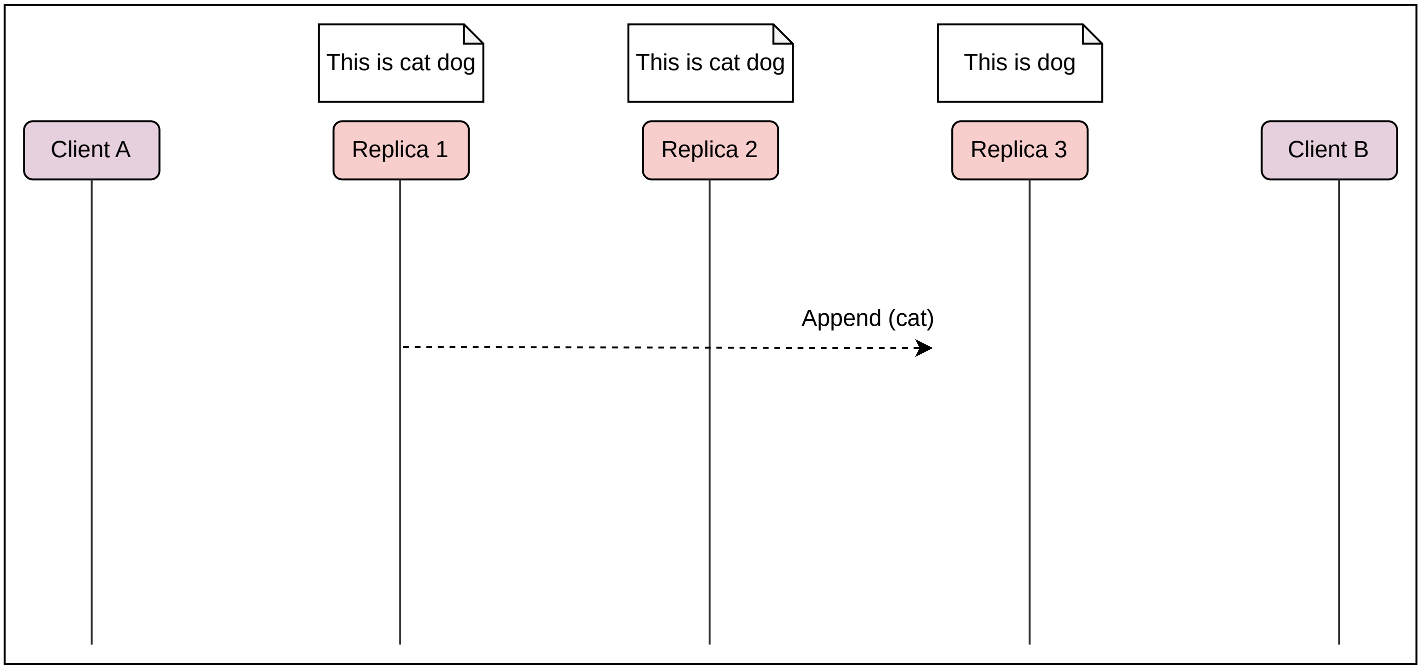
Replica 2 performs Client B's append operation propagated by Replica 3

8 of 13



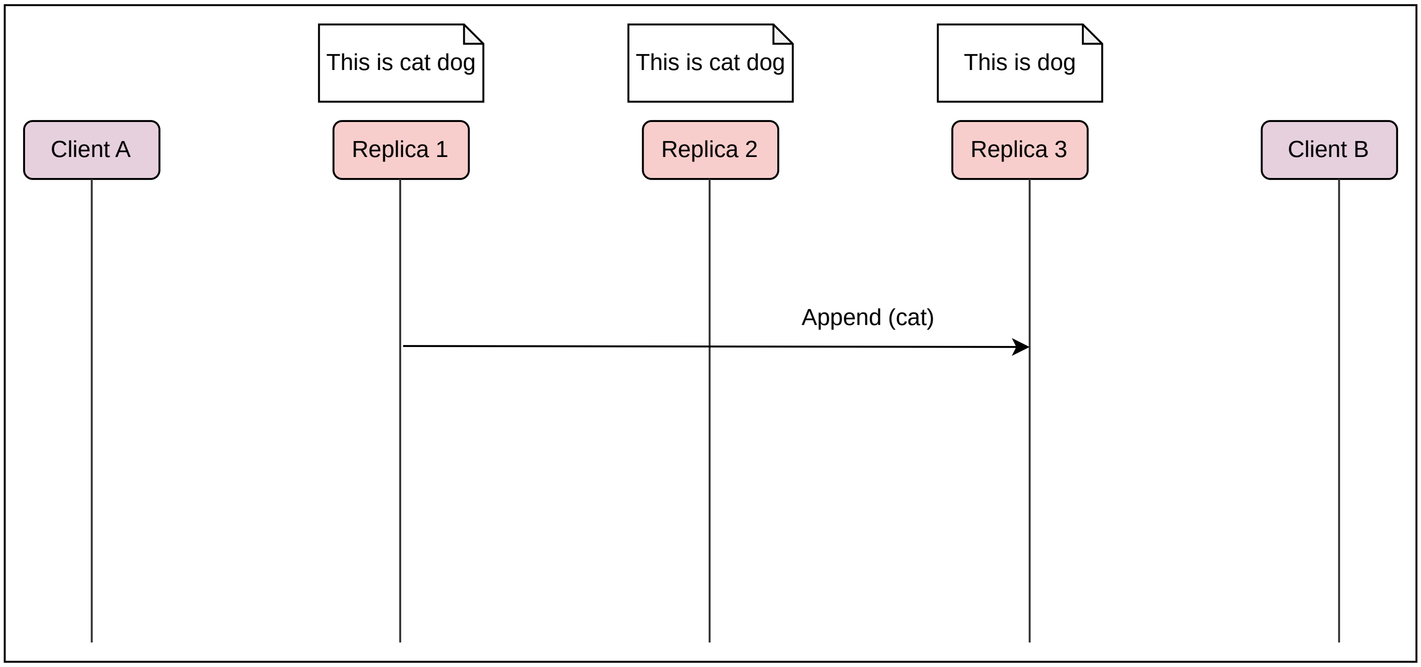
Replica 1 receives Client B's append operation propagated by Replica 3

9 of 13



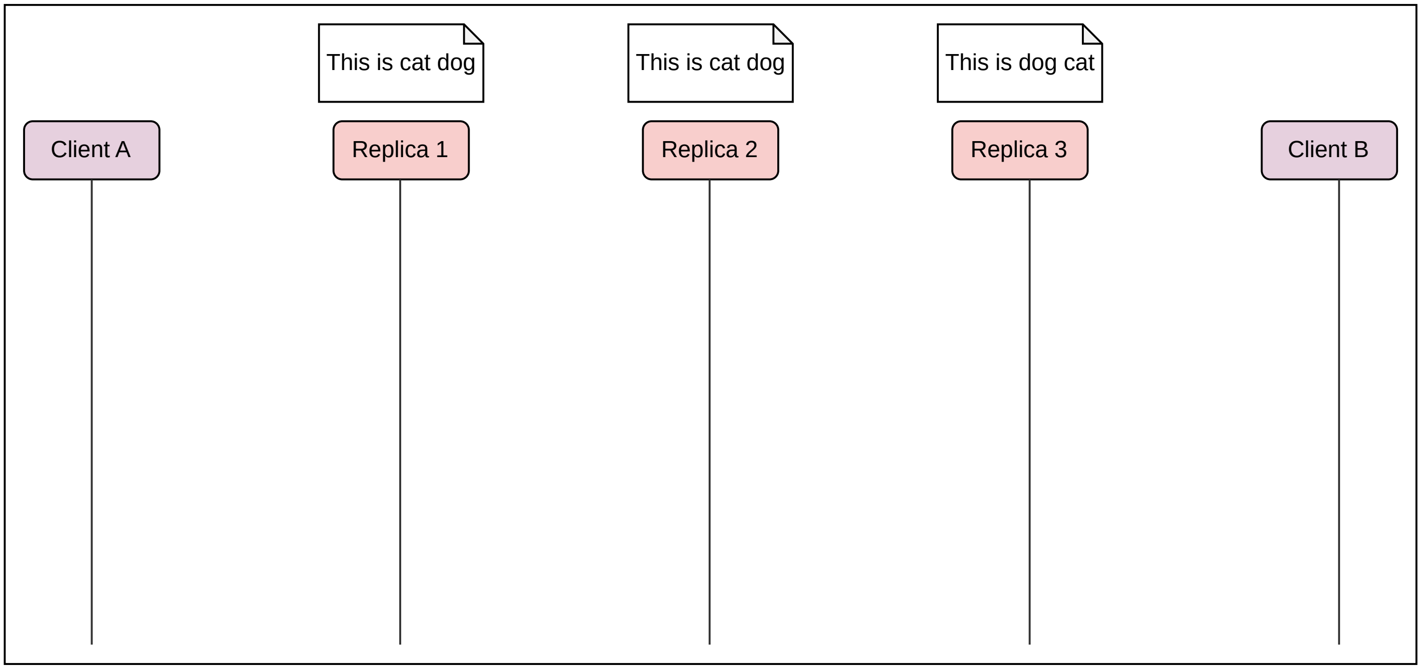
Replica 1 performs Client B's append operation propagated by Replica 3

10 of 13



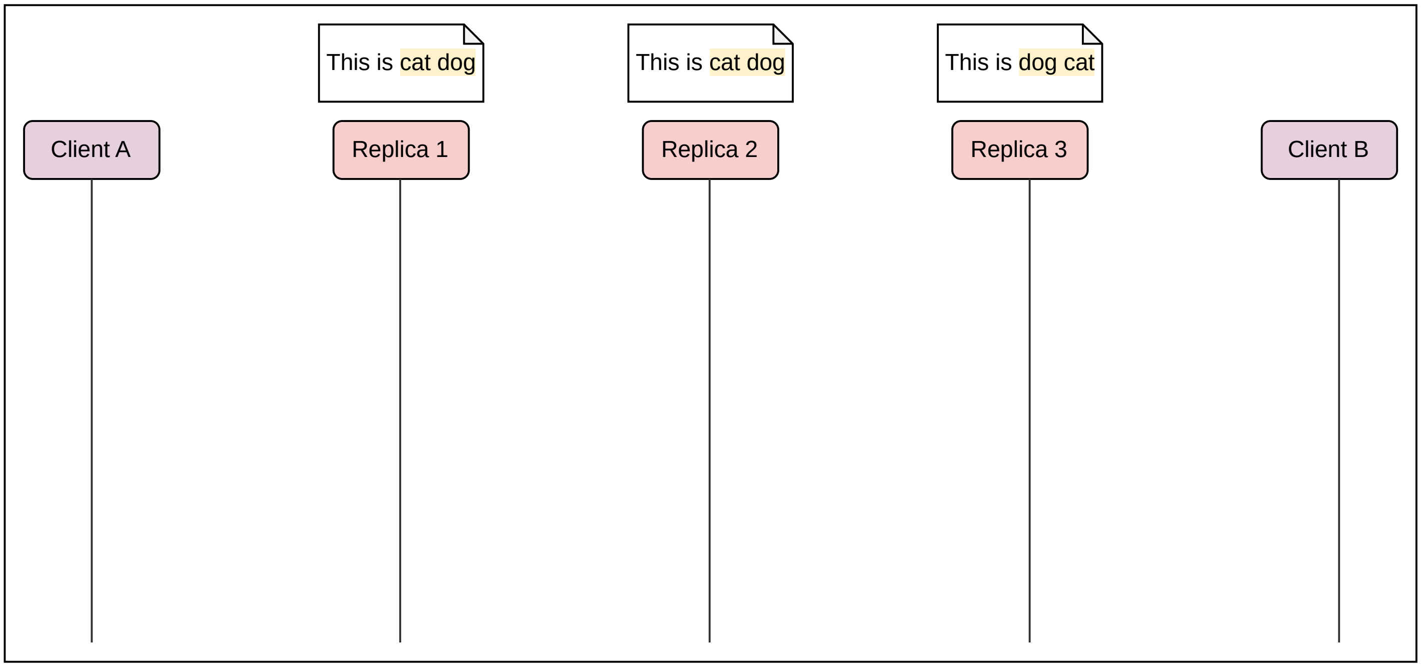
Replica 3 receives Client A's append operation propagated by Replica 1

11 of 13



Replica 3 performs Client A's append operation propagated by Replica 1

12 of 13



The data is inconsistent between two replicas, and if there were more than two operations, it would have been inconsistent among all replicas

13 of 13

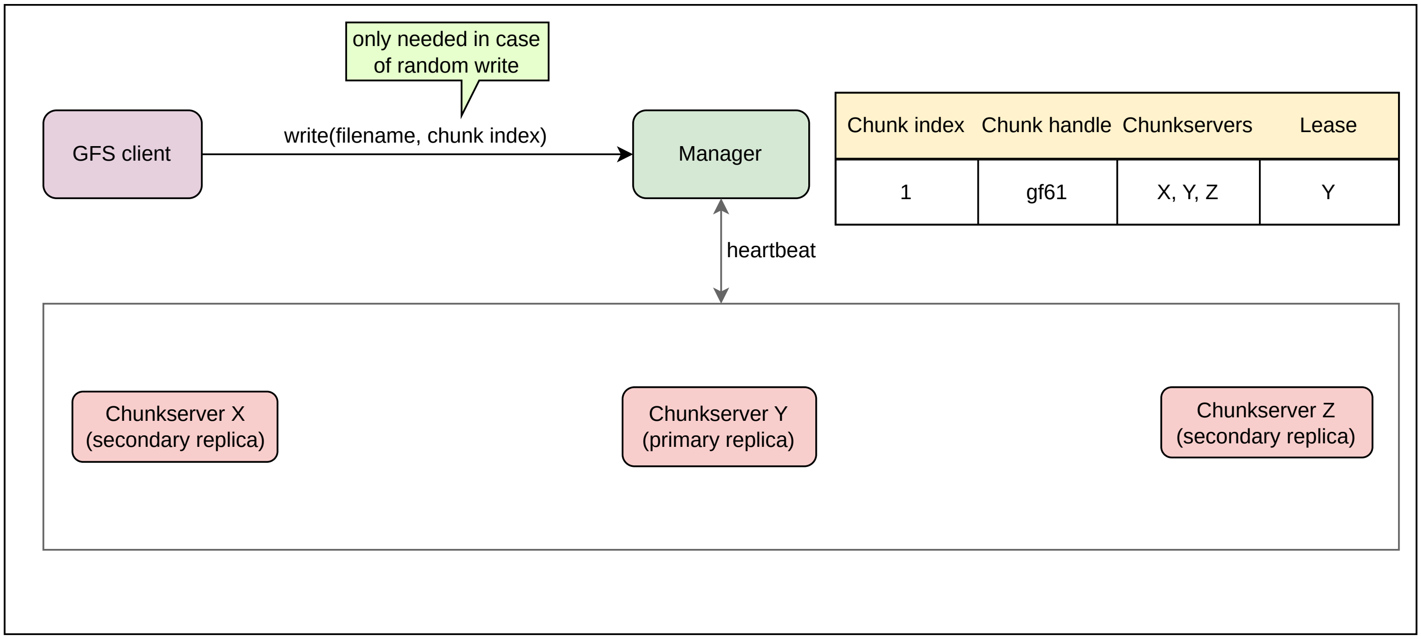
Serialization is required to cope with the data inconsistency issue among the replicas shown above. It is challenging to serialize write operations across multiple replicas. The manager has to do a lot of work to manage all this. So, the manager uses the lease mechanism to reduce this management overhead. With a lease, all the write operations are carried out on a single replica at a time, which makes it easy to serialize operations on a single chunkserver. Let's see how it works by looking into the workflow of a write operation.

Workflow of write operations

The workflow for both the random write and the append operation is almost the same and is illustrated below. The only difference is the chunk to which the data is written.

- In random writes, the clients provide the offset at which the data is to be written. The client can map the offset to a chunk index if each chunk contains all 64 MB (the maximum data a chunk holds/the size of the chunk) and there is no padding. This is internal to GFS. It is also possible that it uses the byte range instead of an index to deal with the padding. The chunk is found using the client-provided offset.

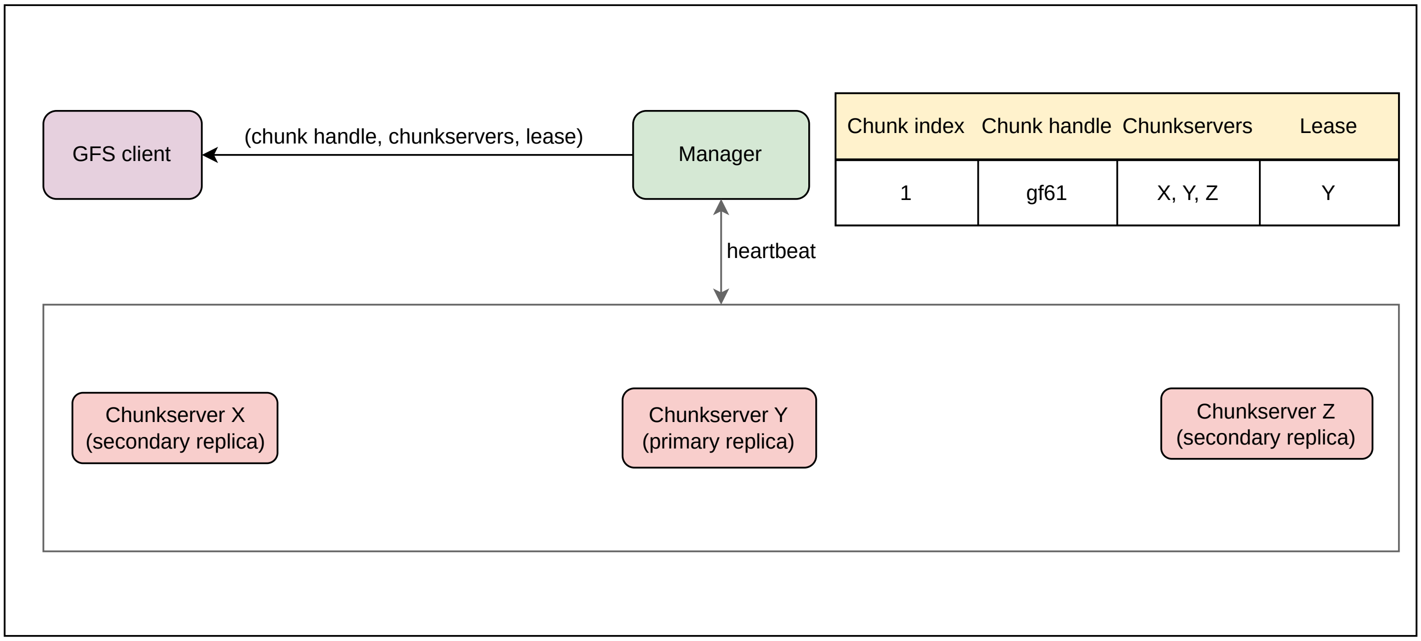
- In append operations, the client doesn't need to provide the offset. The data is pushed to the last chunk.


only needed in case of random write
GFS client
write(filename, chunk index)
Manager
heartbeat
Chunkserver X (secondary replica)
Chunkserver Y (primary replica)
Chunkserver Z (secondary replica)

Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

The client asks the manager for the chunk to which the data is to be written, the replicas holding that chunk, and the current lease for that chunk

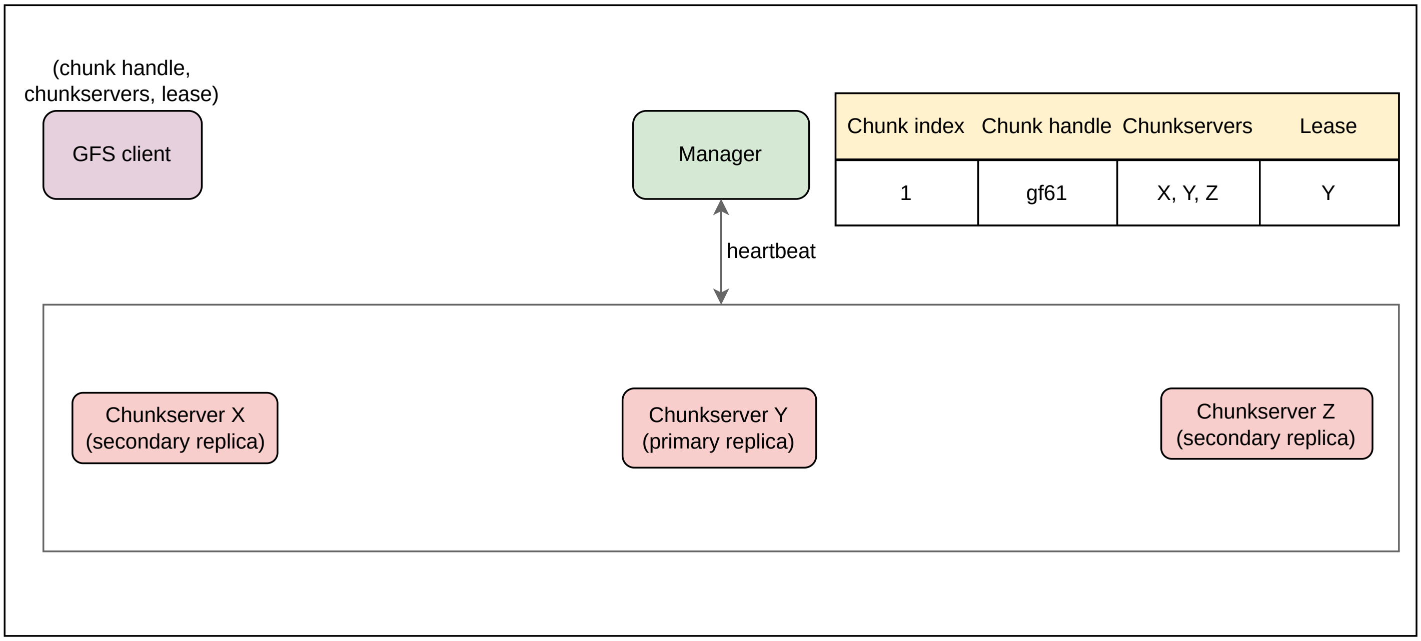
1 of 10


GFS client
(chunk handle, chunkservers, lease)
Manager
heartbeat
Chunkserver X (secondary replica)
Chunkserver Y (primary replica)
Chunkserver Z (secondary replica)

Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

The manager looks for the required information in the metadata and responds the client

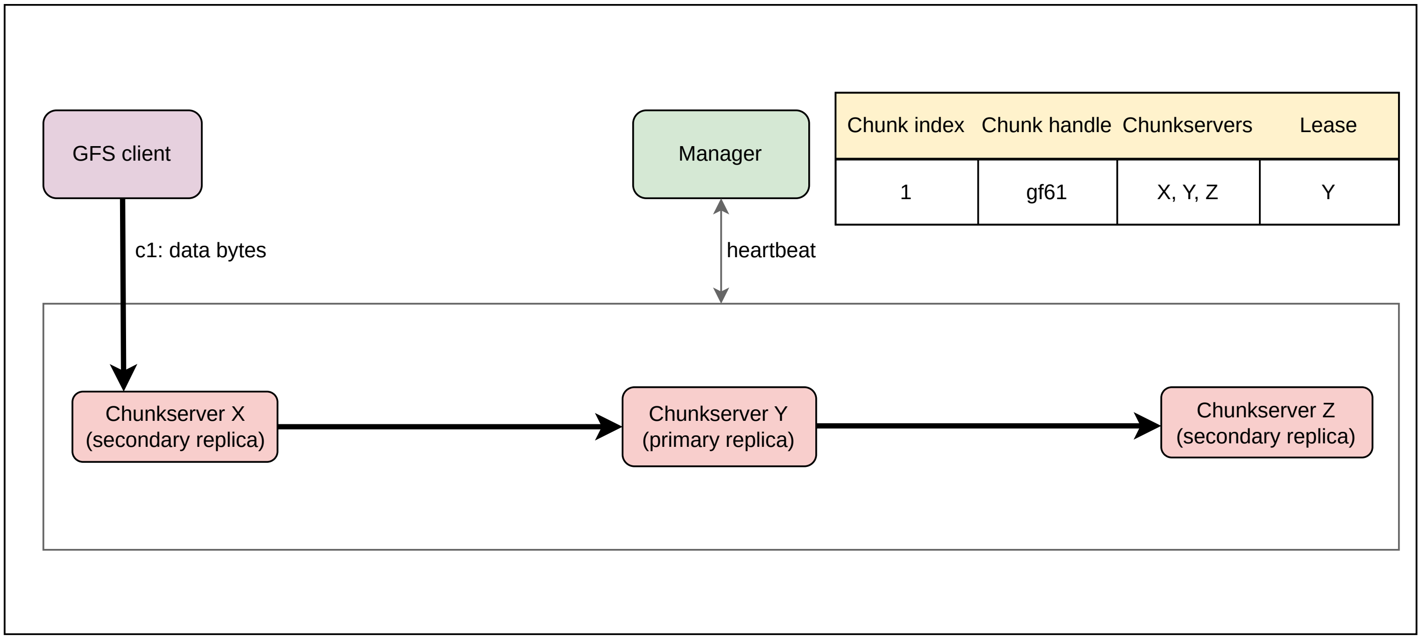
2 of 10


(chunk handle, chunkservers, lease)
GFS client
Manager
heartbeat
Chunkserver X (secondary replica)
Chunkserver Y (primary replica)
Chunkserver Z (secondary replica)

Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

Client caches the metadata for future mutations

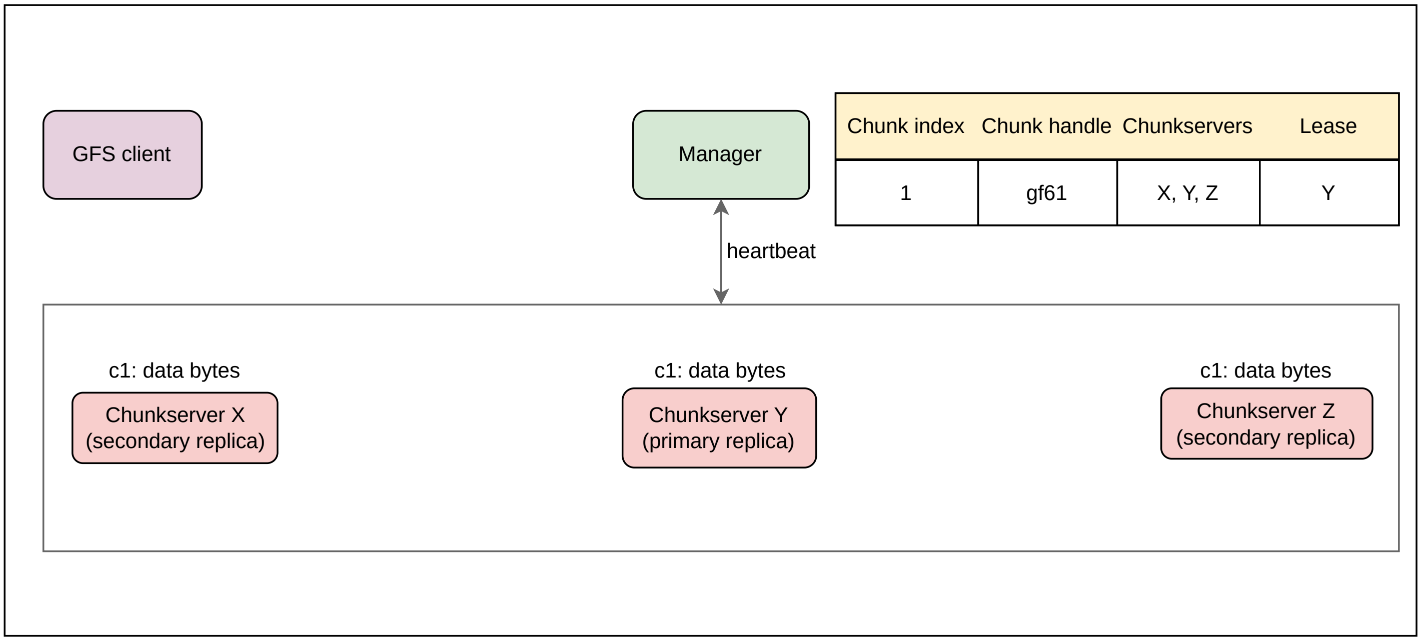
3 of 10


GFS client
c1: data bytes
Manager
heartbeat
Chunkserver X (secondary replica)
Chunkserver Y (primary replica)
Chunkserver Z (secondary replica)

Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

The client pushes the data linearly via a properly chosen chain of chunk replicas to make the most of the network

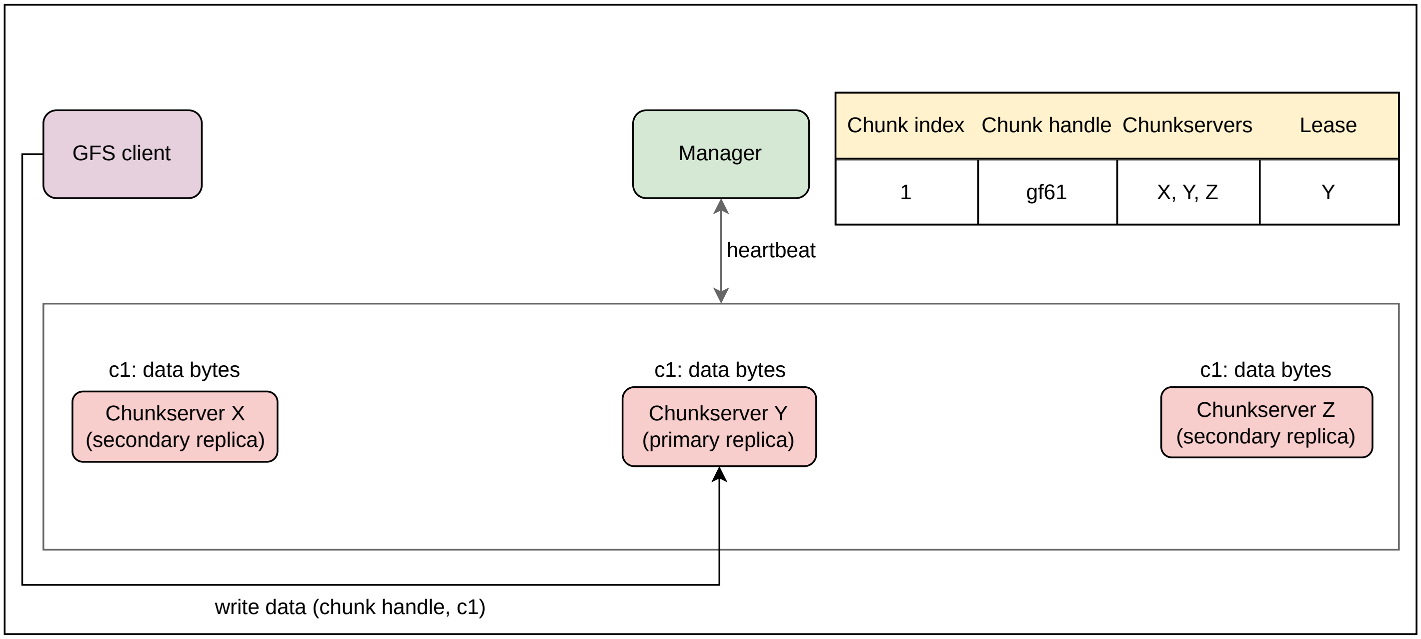
4 of 10



Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

Each chunkserver keeps the data in an internal LRU buffer cache until it is used or has reached its age limit

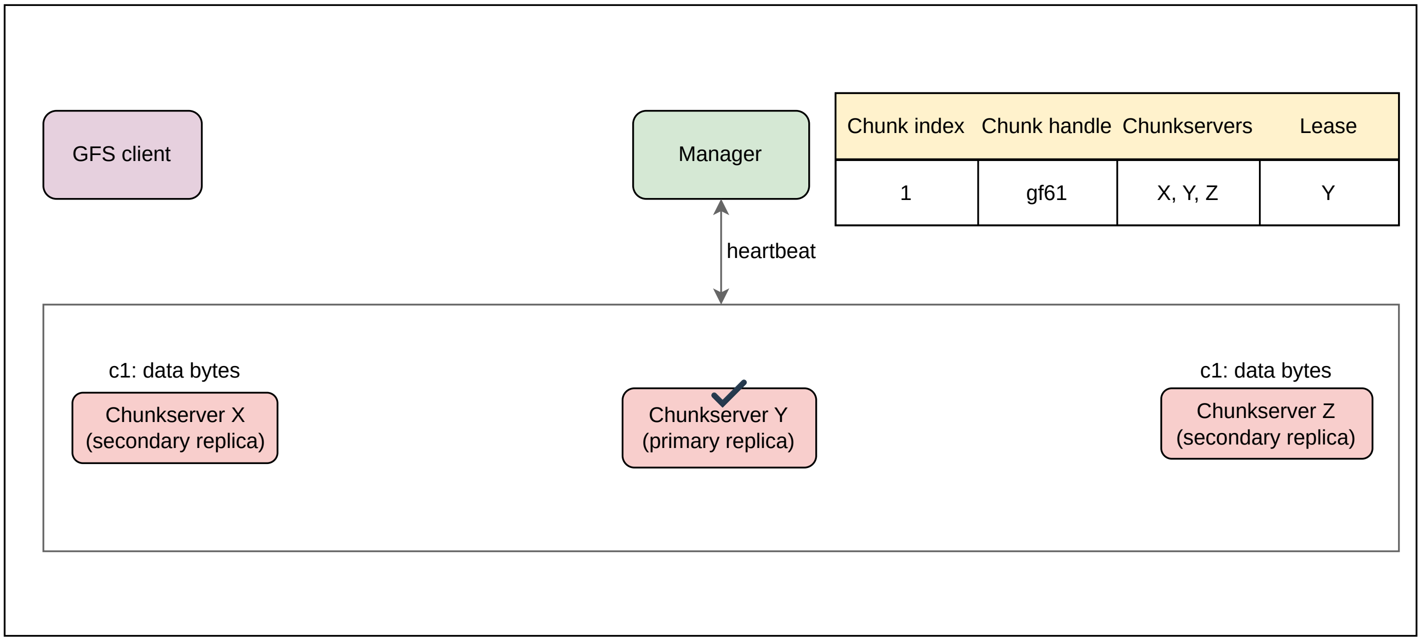
5 of 10



Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

The client makes a write request to the leased chunkserver after each replica has acknowledged receiving the data

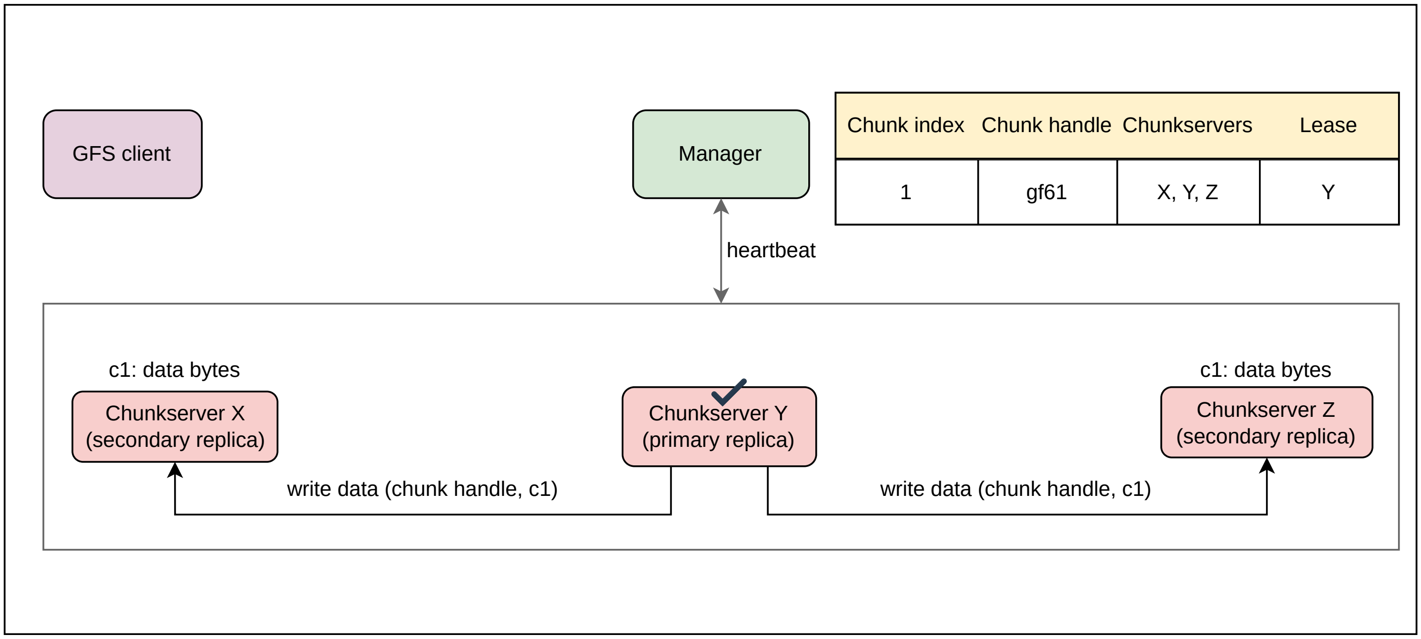
6 of 10



Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

The primary replica identifies the data pushed earlier on which this write request has to be performed and then applies the write request on it

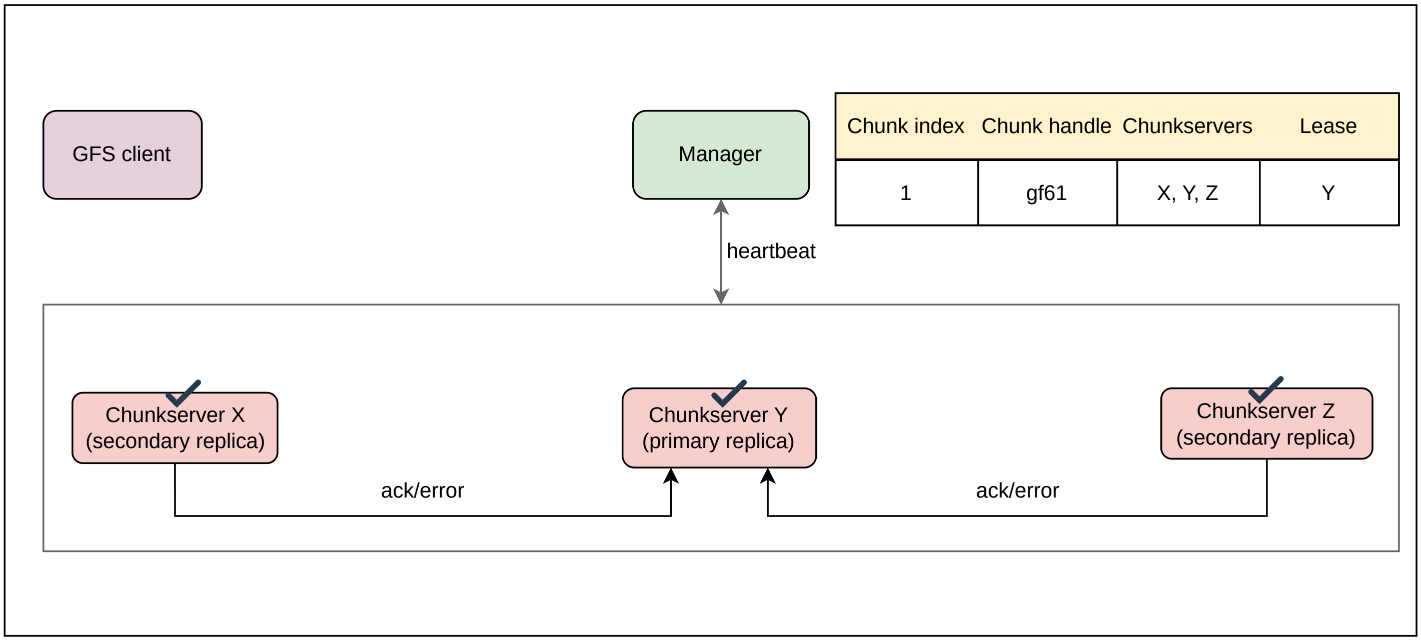
7 of 10



Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

Primary replica after performing the write request on itself forwards the same to secondary replicas

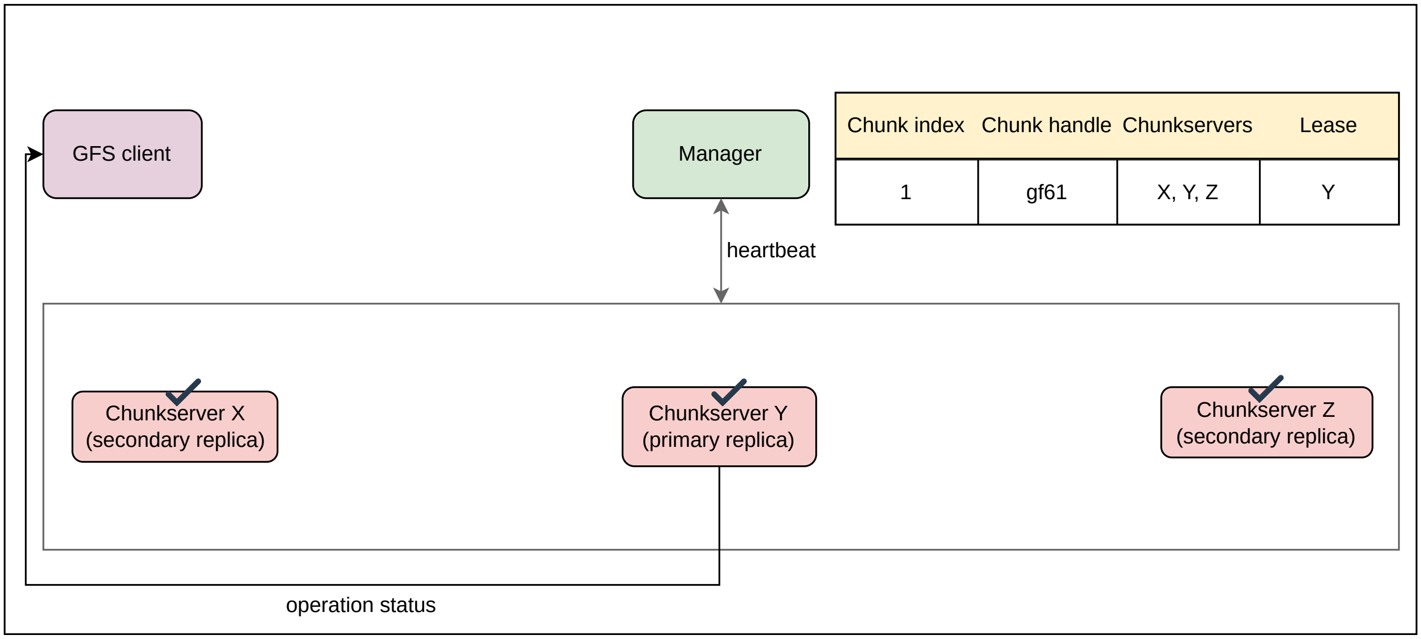
8 of 10


GFS client
Manager
heartbeat
Chunkserver X (secondary replica)
Chunkserver Y (primary replica)
Chunkserver Z (secondary replica)
ack/error
ack/error

Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

Secondary replicas identify the data pushed earlier for the write request, perform the write operation, and respond the primary replica

9 of 10


GFS client
Manager
heartbeat
Chunkserver X (secondary replica)
Chunkserver Y (primary replica)
Chunkserver Z (secondary replica)
operation status

Chunk index	Chunk handle	Chunkservers	Lease
1	gf61	X, Y, Z	Y

The primary replica replies to the client with the operation status, success or failure

10 of 10

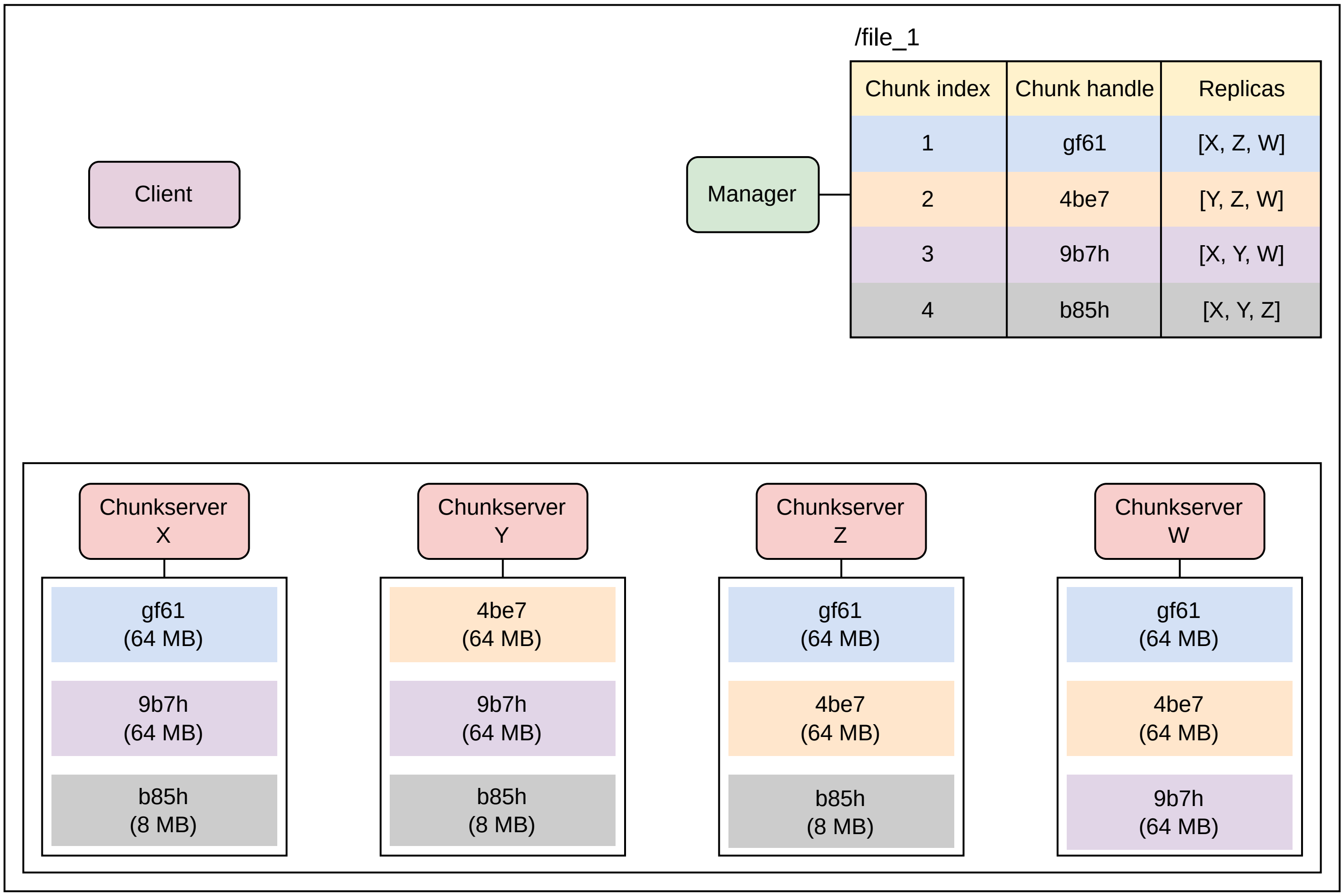
Now, all of the write operations from multiple clients will be directed to the replica that has the current lease for the chunk. The replica holding the lease is the primary replica. The primary replica will serialize all of the write operations that are to be performed on the chunk. It then performs the operation in the order of serial numbers assigned to each operation. All of the secondary replicas execute the write operations in the same order.

The successful serialized execution of write operations on the replicas makes sure that all the replicas contain the same data. If any of the replicas fail to perform the write operation on it, then that replica will lag behind. If the clients read data from that replica, they won’t get the updated data. We will see how GFS copes with this issue, as well as others, in the data consistency model lesson.

The edge cases

The append operation writes data to the end of the file. To perform the operation, the client asks the manager for the last chunk's metadata (chunk handle, and chunkservers). The manager node looks into the metadata for the last chunk of the file and responds to the GFS client with the required metadata. The client then sends data to the chunkserver holding the last chunk. There are three different scenarios the client has to deal with depending on the available space on the last chunk and the total size of the data to be written. Let's look at each scenario.

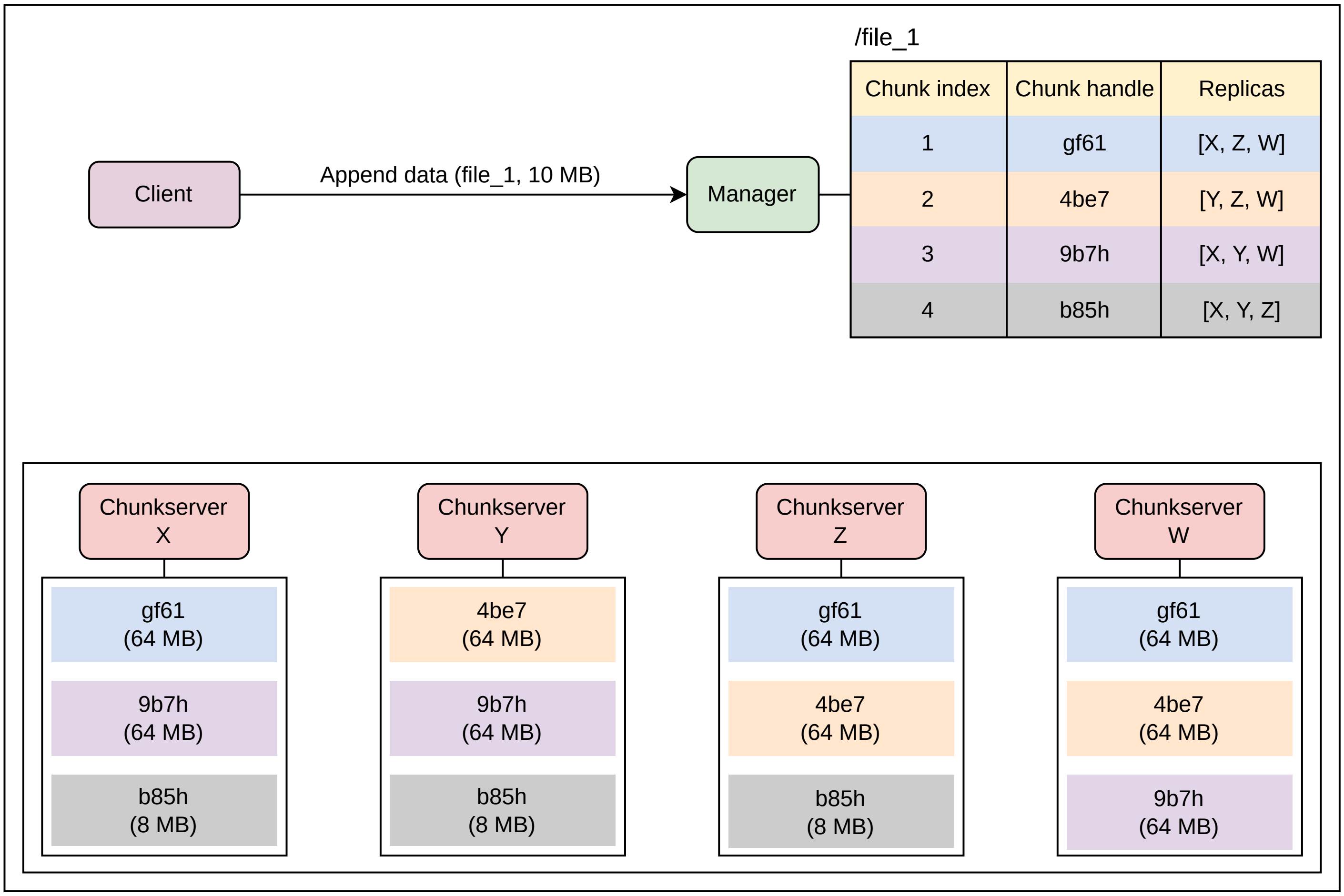
1. Suppose the last chunk has available space for appending new data. In that case, the chunkservers write that data in the specific chunk returned by the manager and respond to the client with a success message, as shown in the following illustration.



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The system consists of a client, a manager node, and the chunkservers



Client

Append data (file_1, 10 MB)

Manager

/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

Chunkserver X

gf61 (64 MB)

9b7h (64 MB)

b85h (8 MB)

Chunkserver Y

4be7 (64 MB)

9b7h (64 MB)

b85h (8 MB)

Chunkserver Z

gf61 (64 MB)

4be7 (64 MB)

b85h (8 MB)

Chunkserver W

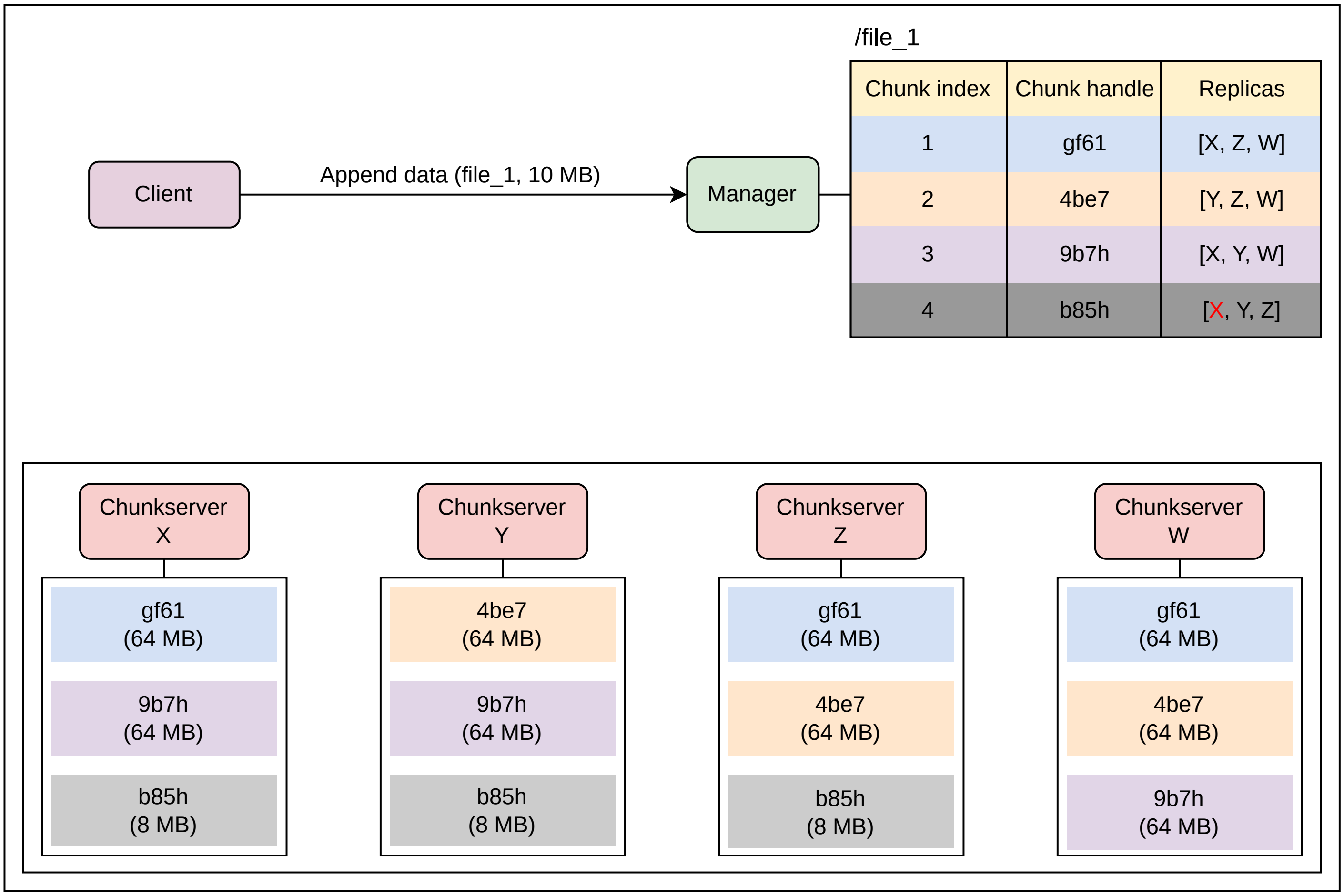
gf61 (64 MB)

4be7 (64 MB)

9b7h (64 MB)

The client puts an append request that is received at the manager

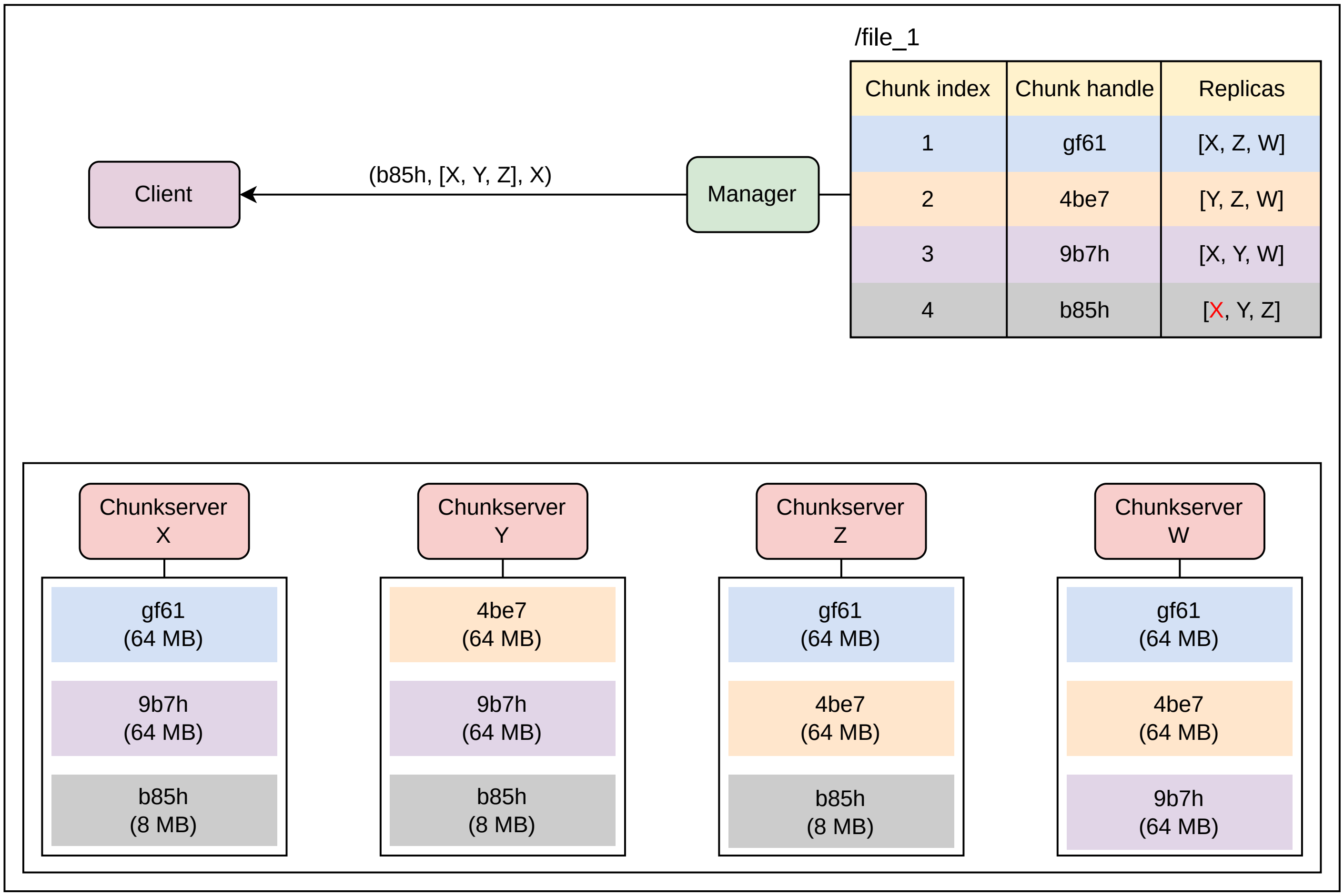
2 of 7



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

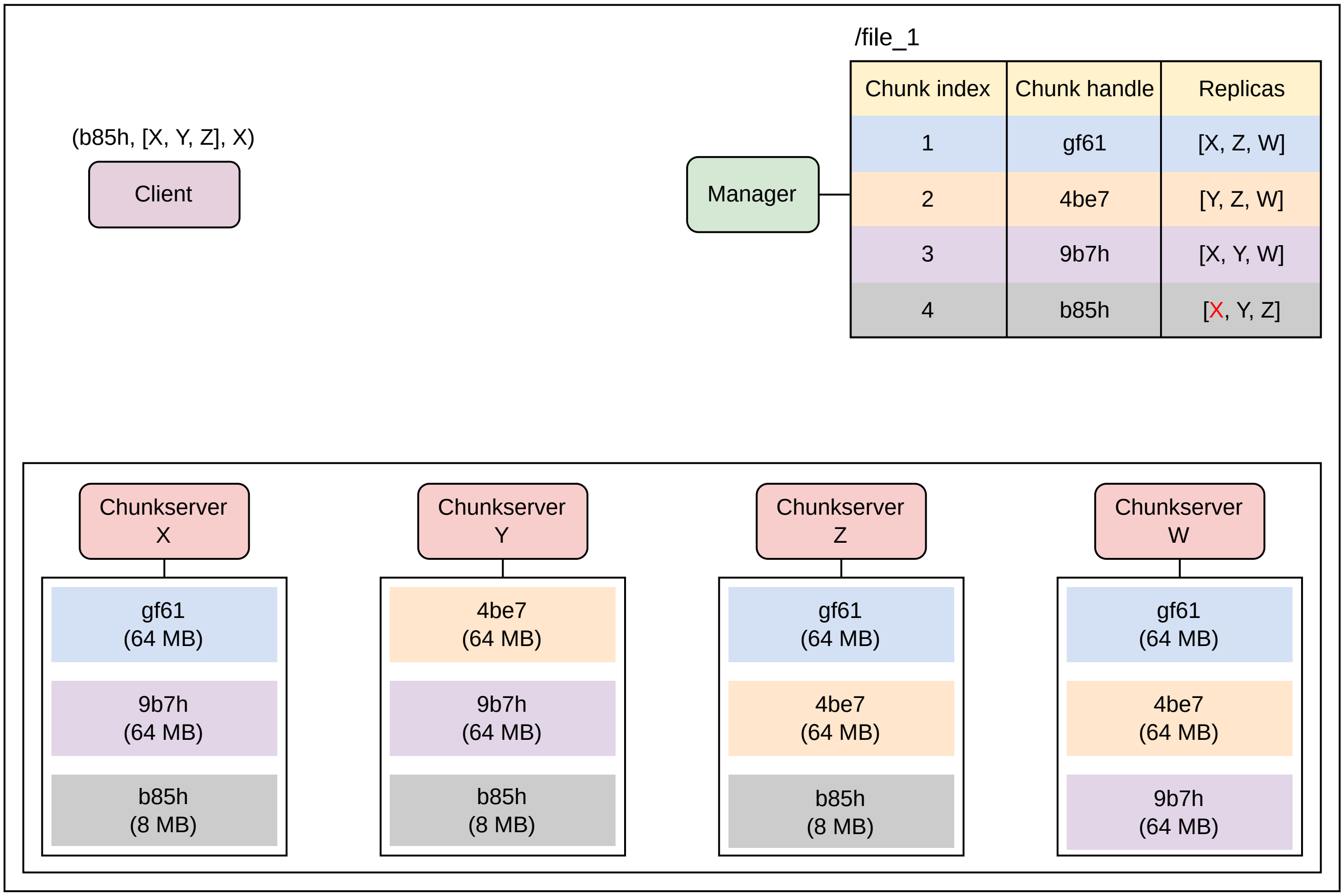
The manager looks for the last chunk of the file in the metadata, and grants lease to a replica (shown in red) if no one has a lease already



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The manager replies the client with the metadata consisting of chunk handle, list of replicas, and the leased replica



(b85h, [X, Y, Z], X)

Client

Manager

/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

Chunkserver X

gf61 (64 MB)

9b7h (64 MB)

b85h (8 MB)

Chunkserver Y

4be7 (64 MB)

9b7h (64 MB)

b85h (8 MB)

Chunkserver Z

gf61 (64 MB)

4be7 (64 MB)

b85h (8 MB)

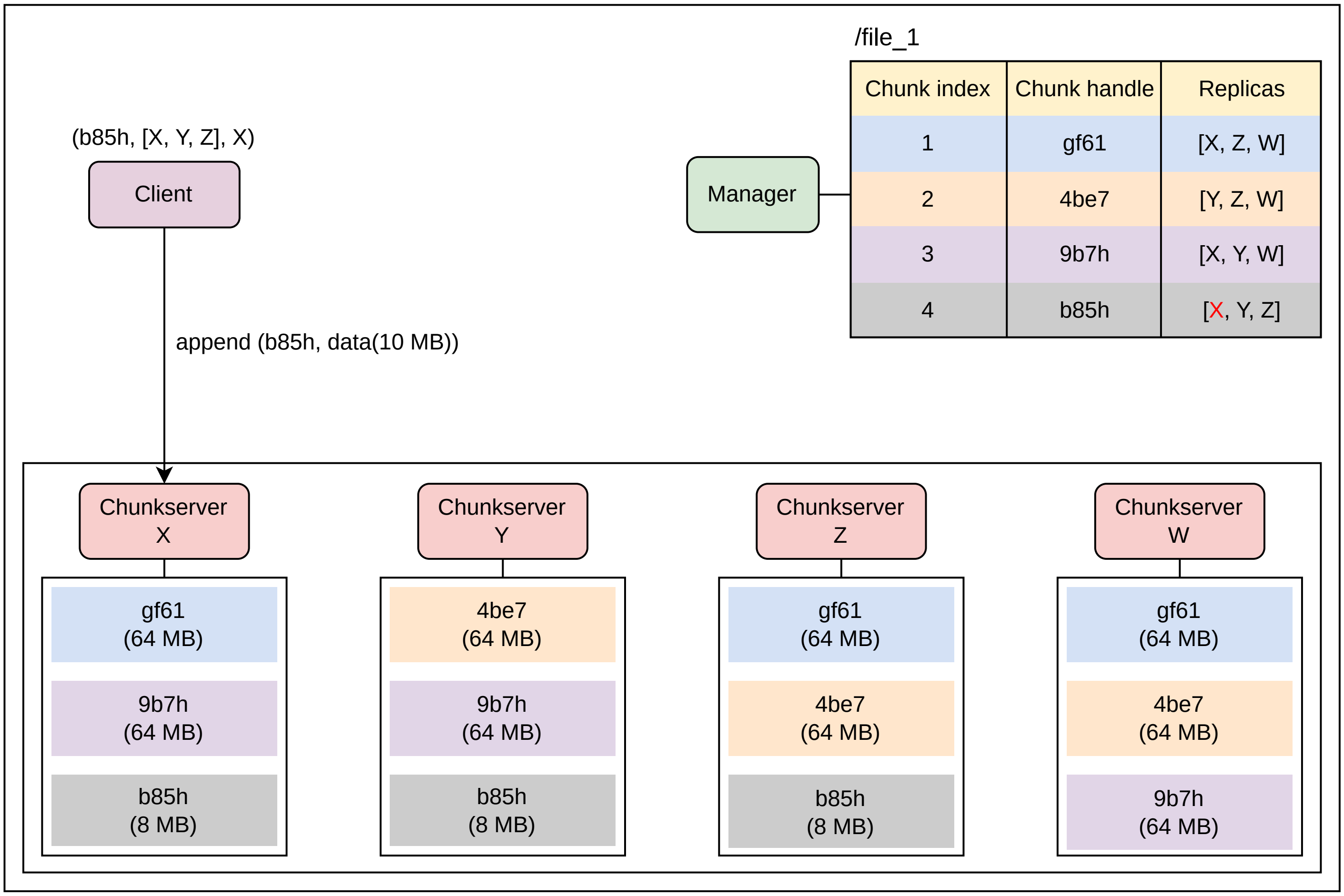
Chunkserver W

gf61 (64 MB)

4be7 (64 MB)

9b7h (64 MB)

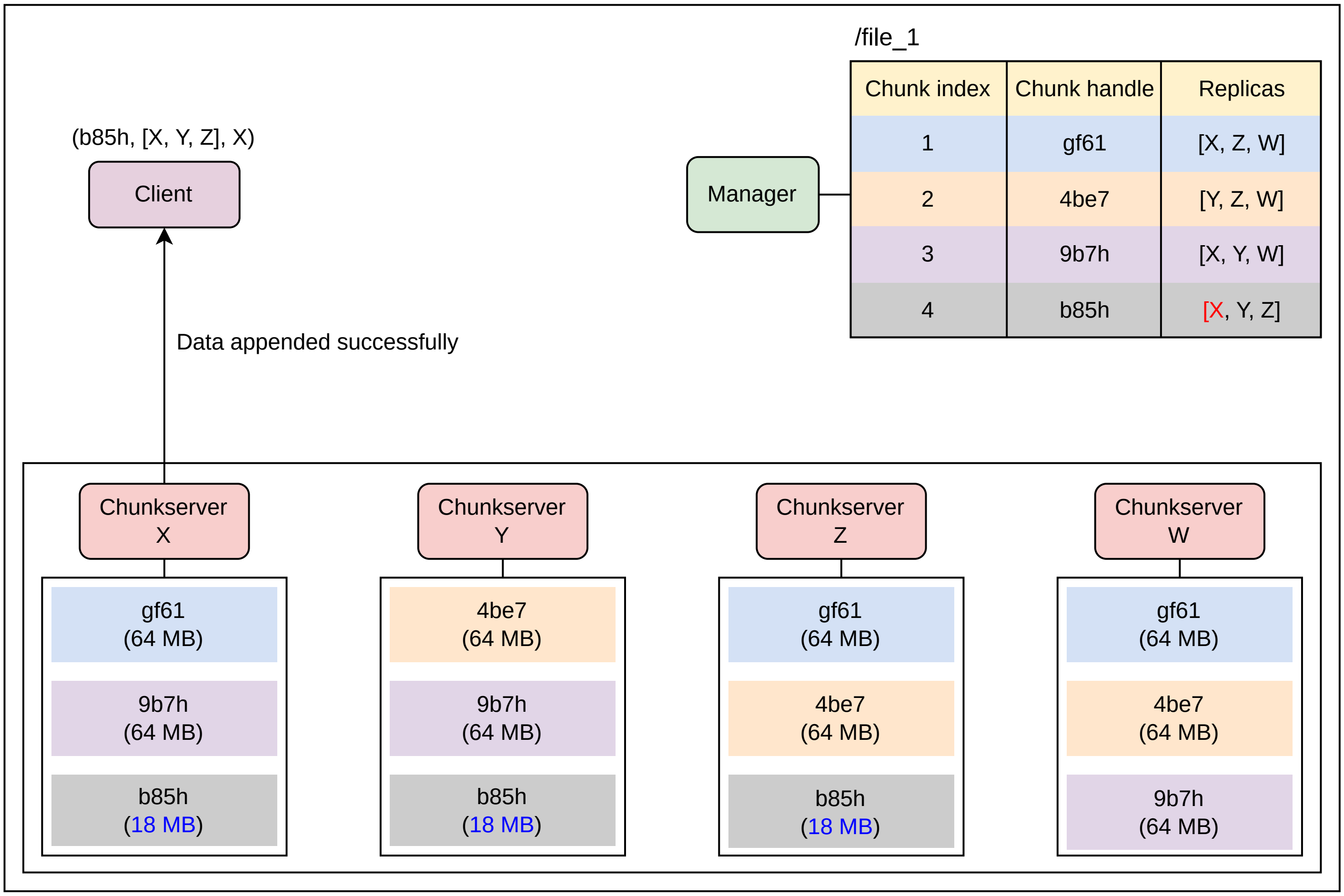
The client caches the metadata to perform further mutations



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The client pushes the data to all the replicas (not shown) and asks the leased replica (X) to carry out the append operation



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

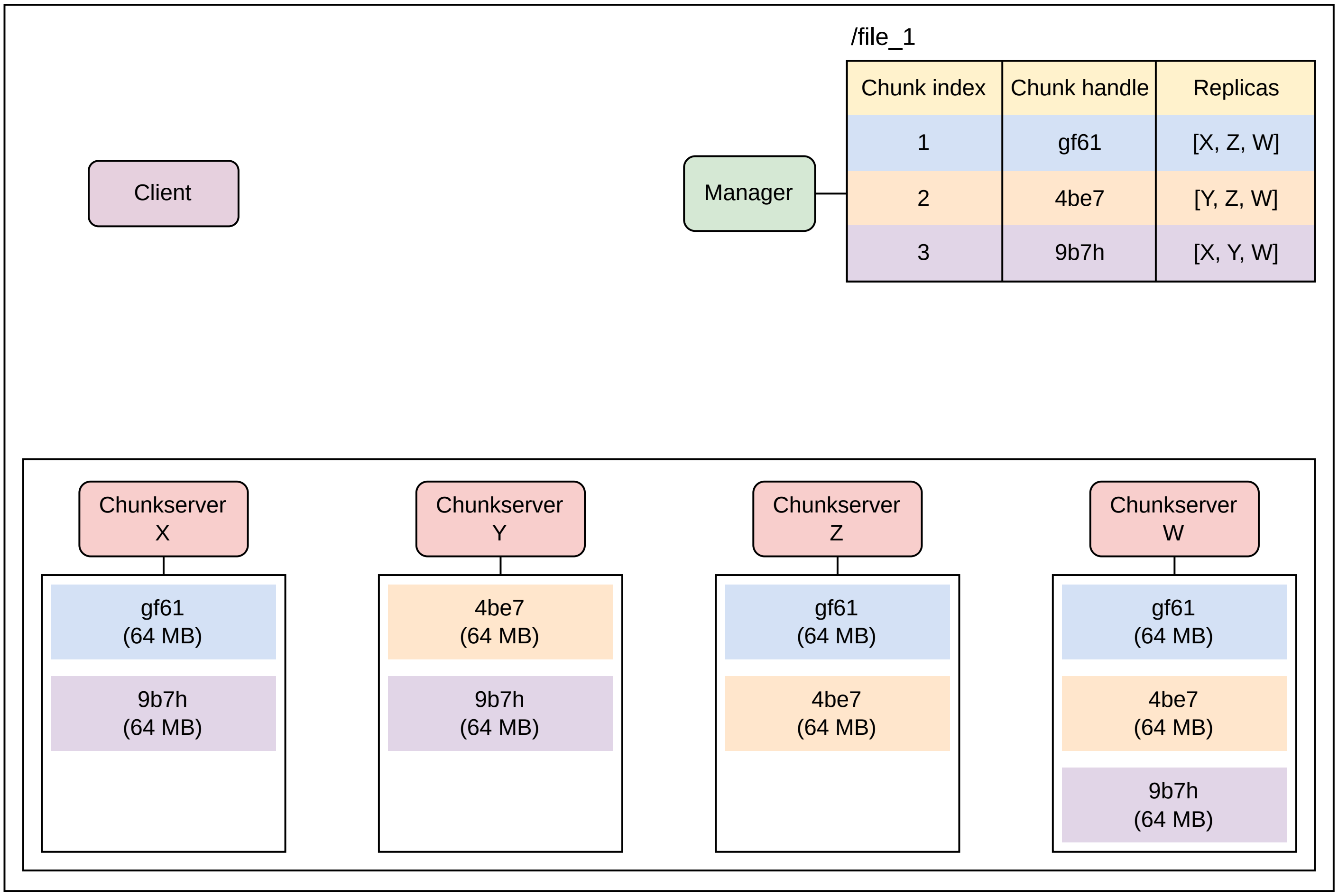
Chunk b85h was only filled with 8 MBs of data and the rest (64-8=56 MB) space is available, the chunkserver writes the data to the chunk

7 of 7

> Since each chunk is replicated on multiple chunkservers for availability, the primary replica has to ensure that the write operation is performed on all replicas.

2. If the last chunk already has 64 MB of data, then the chunkserver responds to the client that it is already full. The client then asks the manager to create a new chunk. The manager generates a new chunk ID, allocates the chunkservers where this chunk's data would be placed, and responds to the client with the new chunk's metadata. The client will then write the data to the new chunk. An example is shown in the following illustration.

> One approach is for the chunkserves to inform the manager about the full chunk in the last heartbeat message to save the client a round-trip to the manager later on. On the other hand, delaying such a declaration to the manager might reduce the immediate load on the manager.

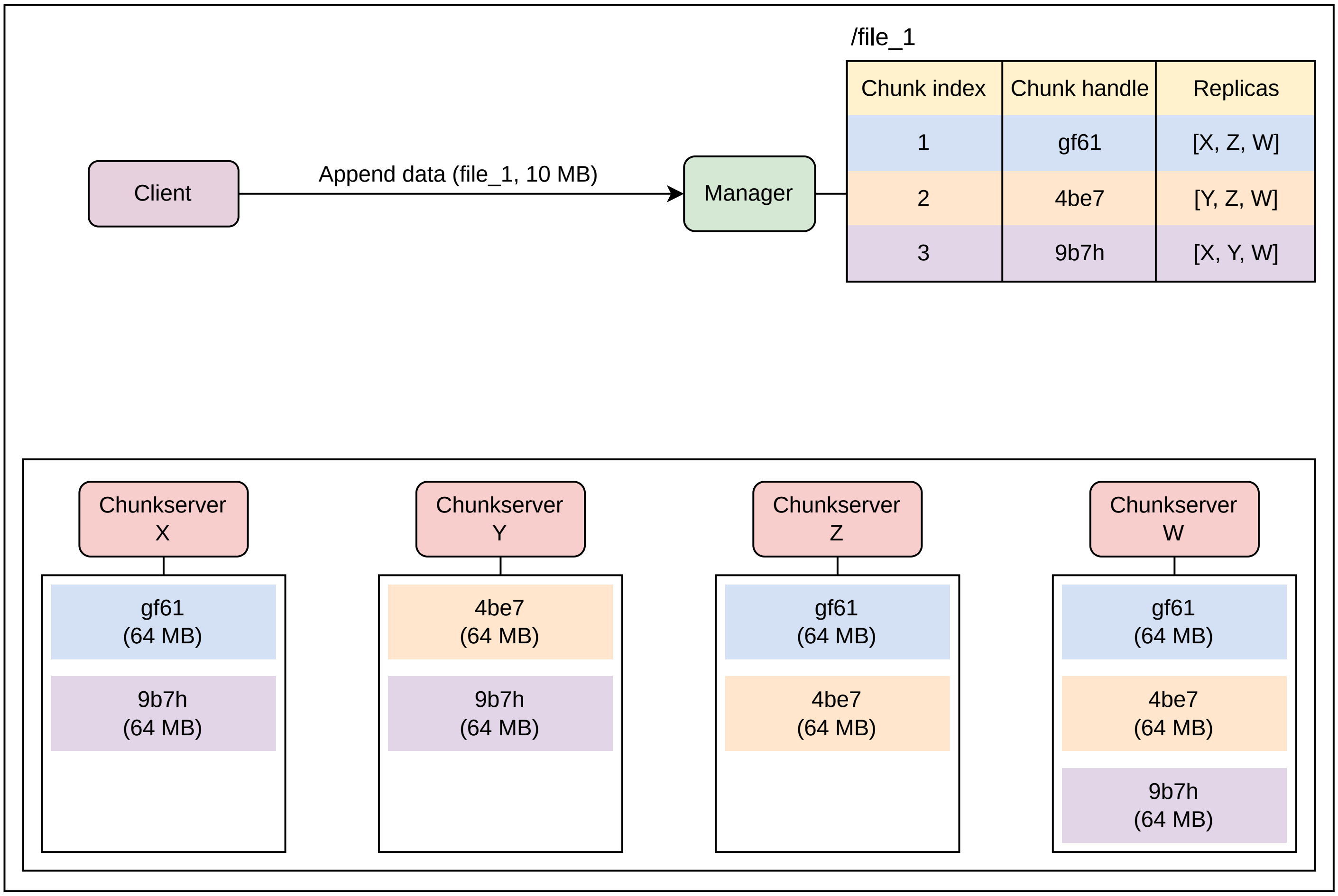


/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

The system consists of a client, a manager node, and the chunkservers

1 of 14

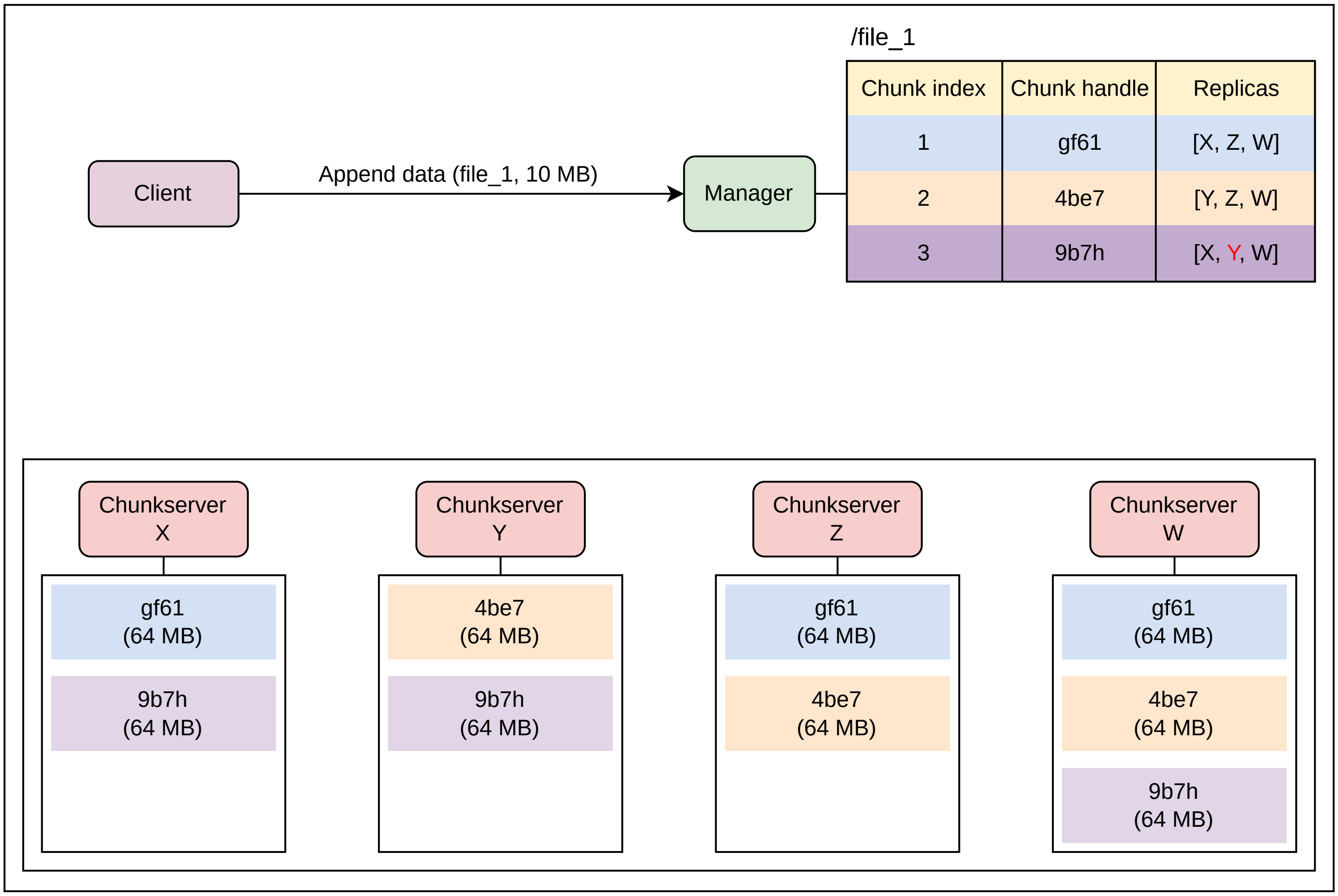


/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

The client puts an append request that is received at the manager node

2 of 14



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

Client

Append data (file_1, 10 MB)

Manager

Chunkserver X

gf61 (64 MB)

9b7h (64 MB)

Chunkserver Y

4be7 (64 MB)

9b7h (64 MB)

Chunkserver Z

gf61 (64 MB)

4be7 (64 MB)

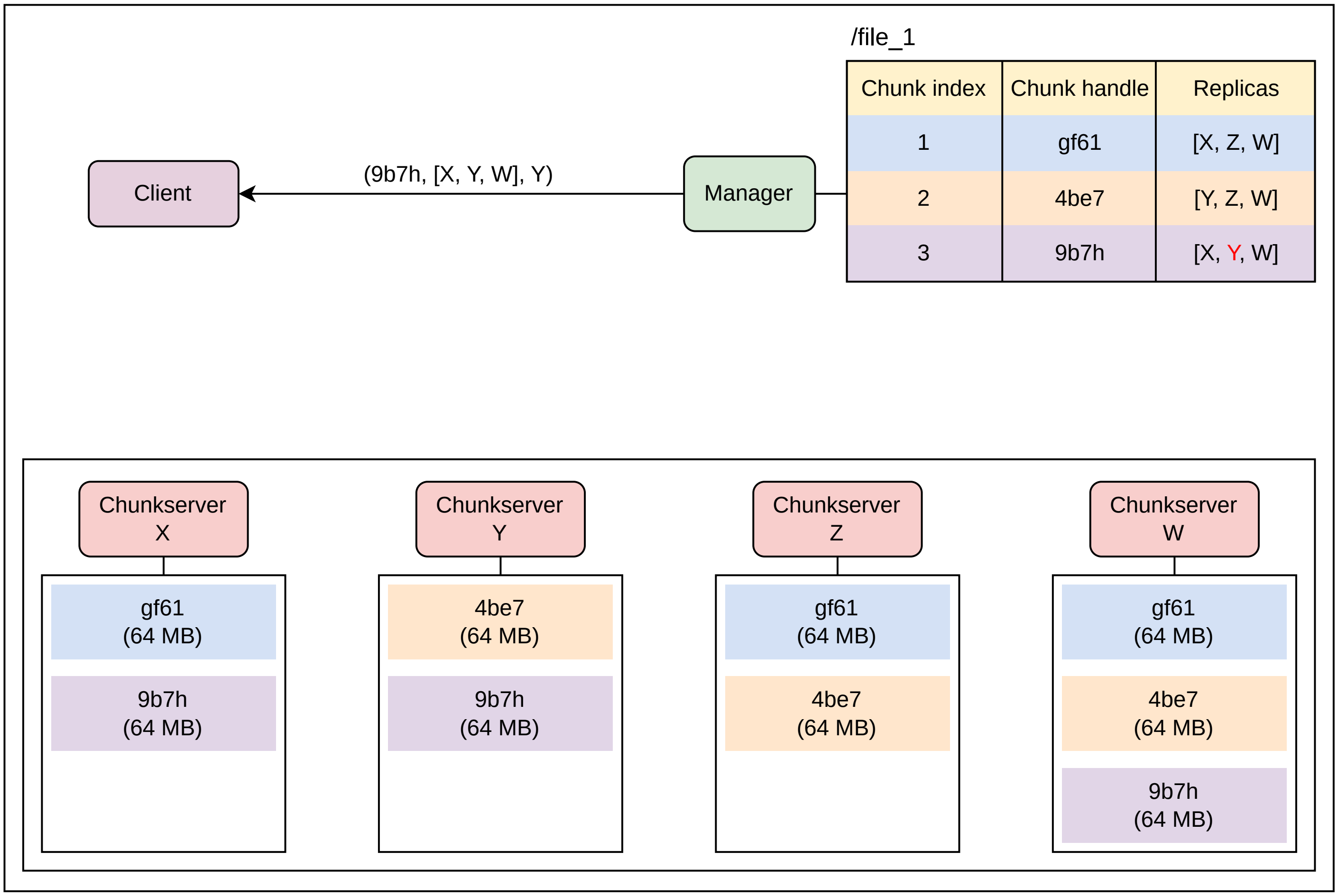
Chunkserver W

gf61 (64 MB)

4be7 (64 MB)

9b7h (64 MB)

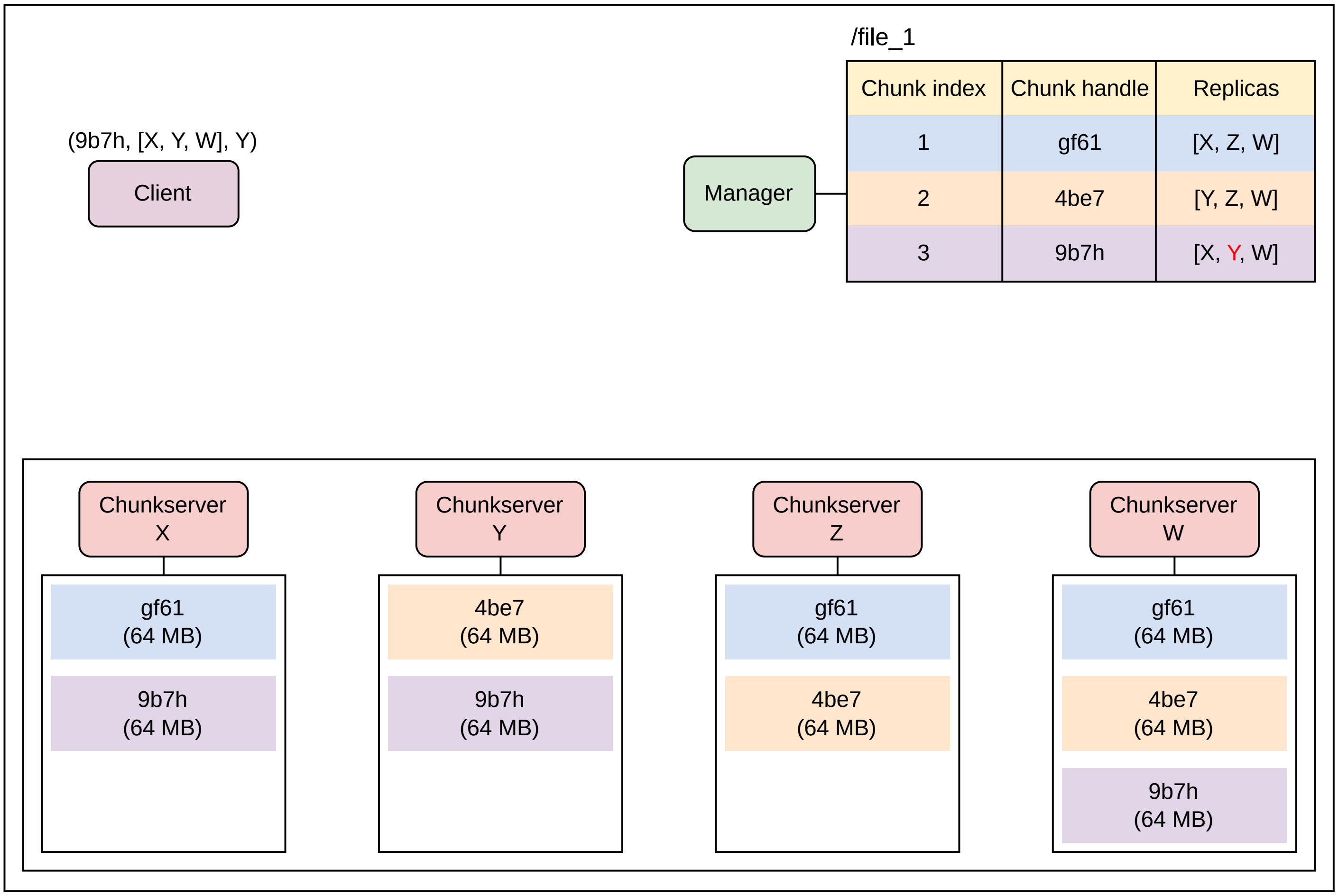
The manager looks for the last chunk of the file in the metadata, and grants lease to a replica (shown in red) if no one has a lease already



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

The manager replies to the client with the metadata consisting of chunk handle, a list of replicas, and the leased replica



(9b7h, [X, Y, W], Y)

Client

Manager

/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

Chunkserver X

gf61 (64 MB)

9b7h (64 MB)

Chunkserver Y

4be7 (64 MB)

9b7h (64 MB)

Chunkserver Z

gf61 (64 MB)

4be7 (64 MB)

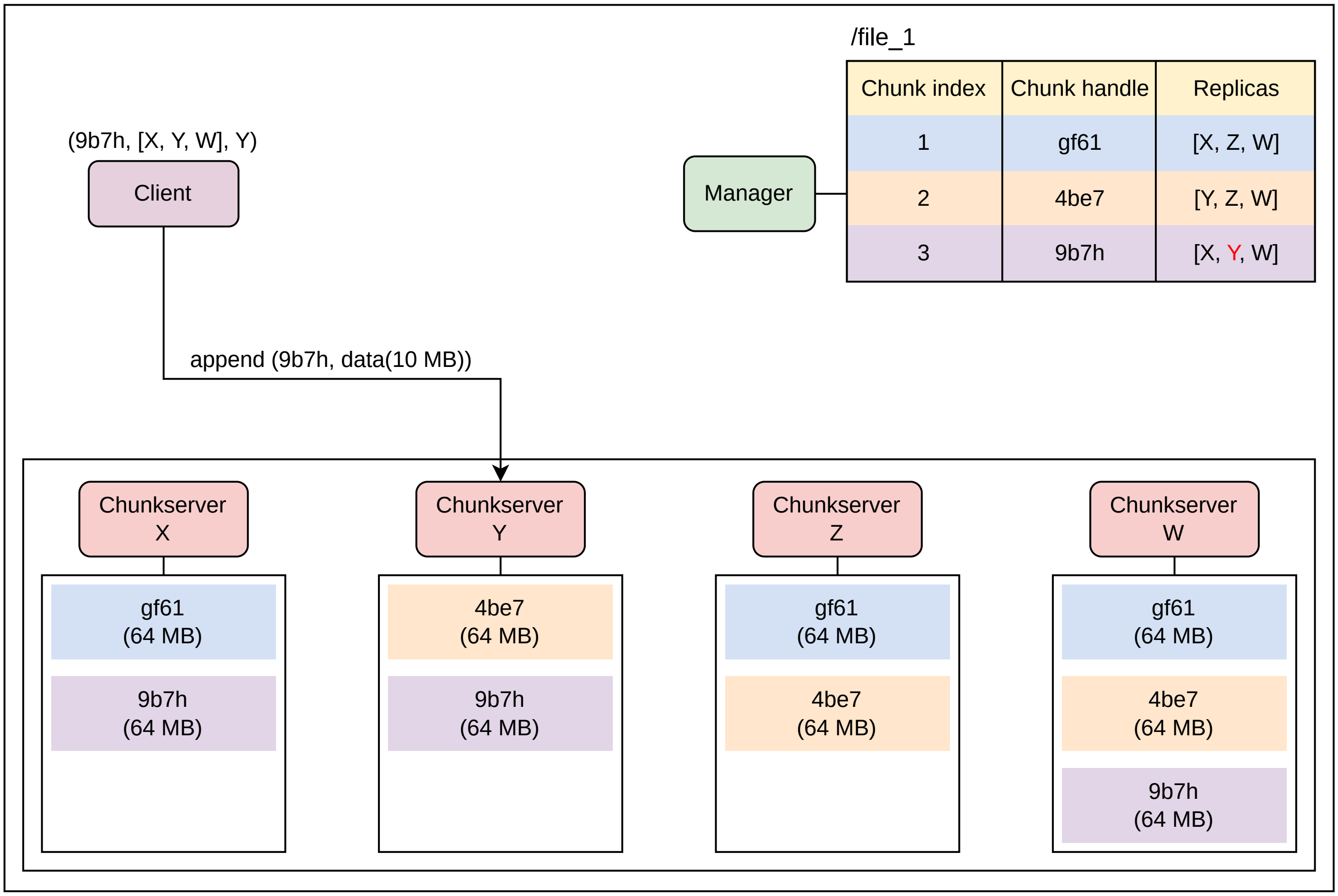
Chunkserver W

gf61 (64 MB)

4be7 (64 MB)

9b7h (64 MB)

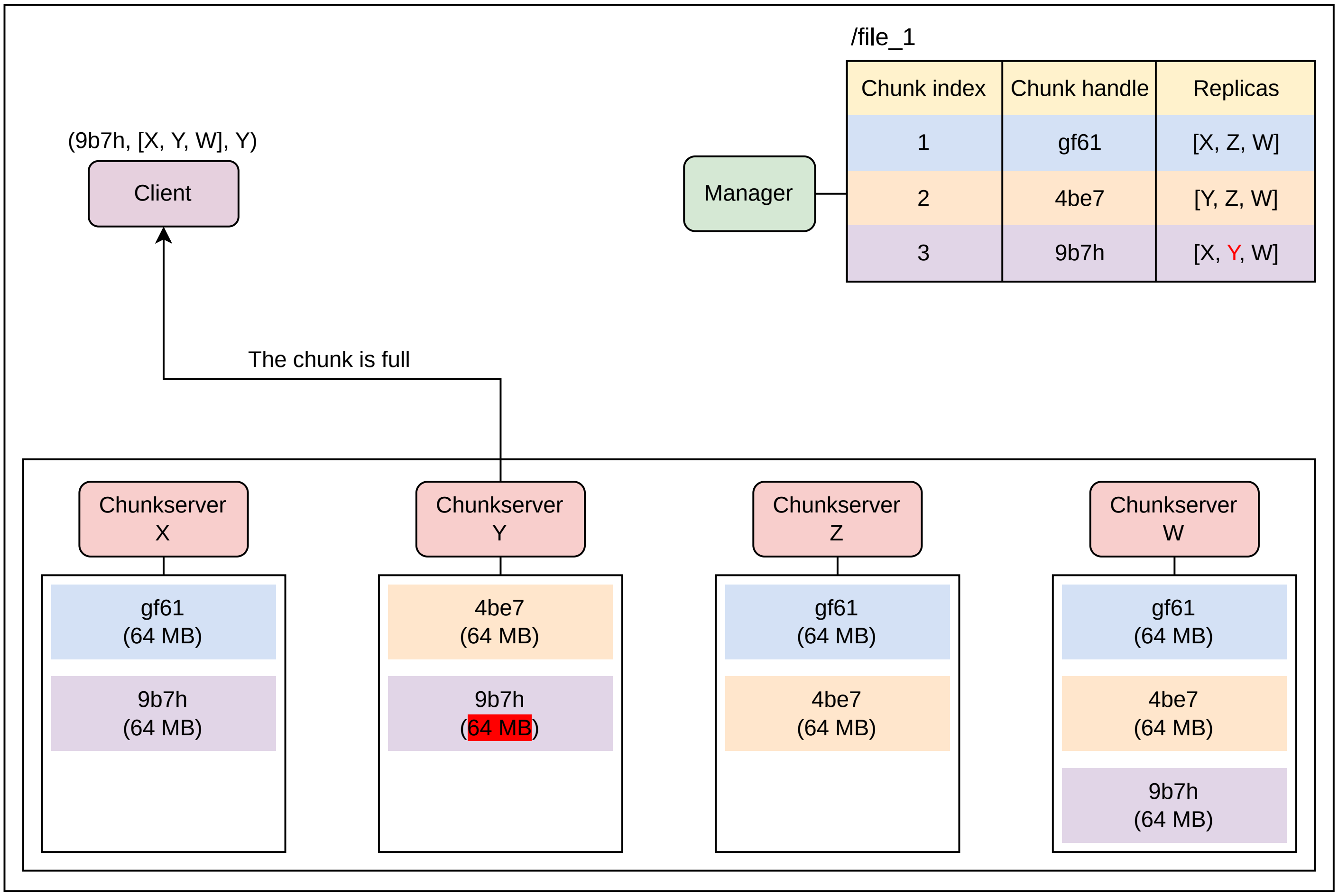
The client caches the metadata to perform further mutations



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

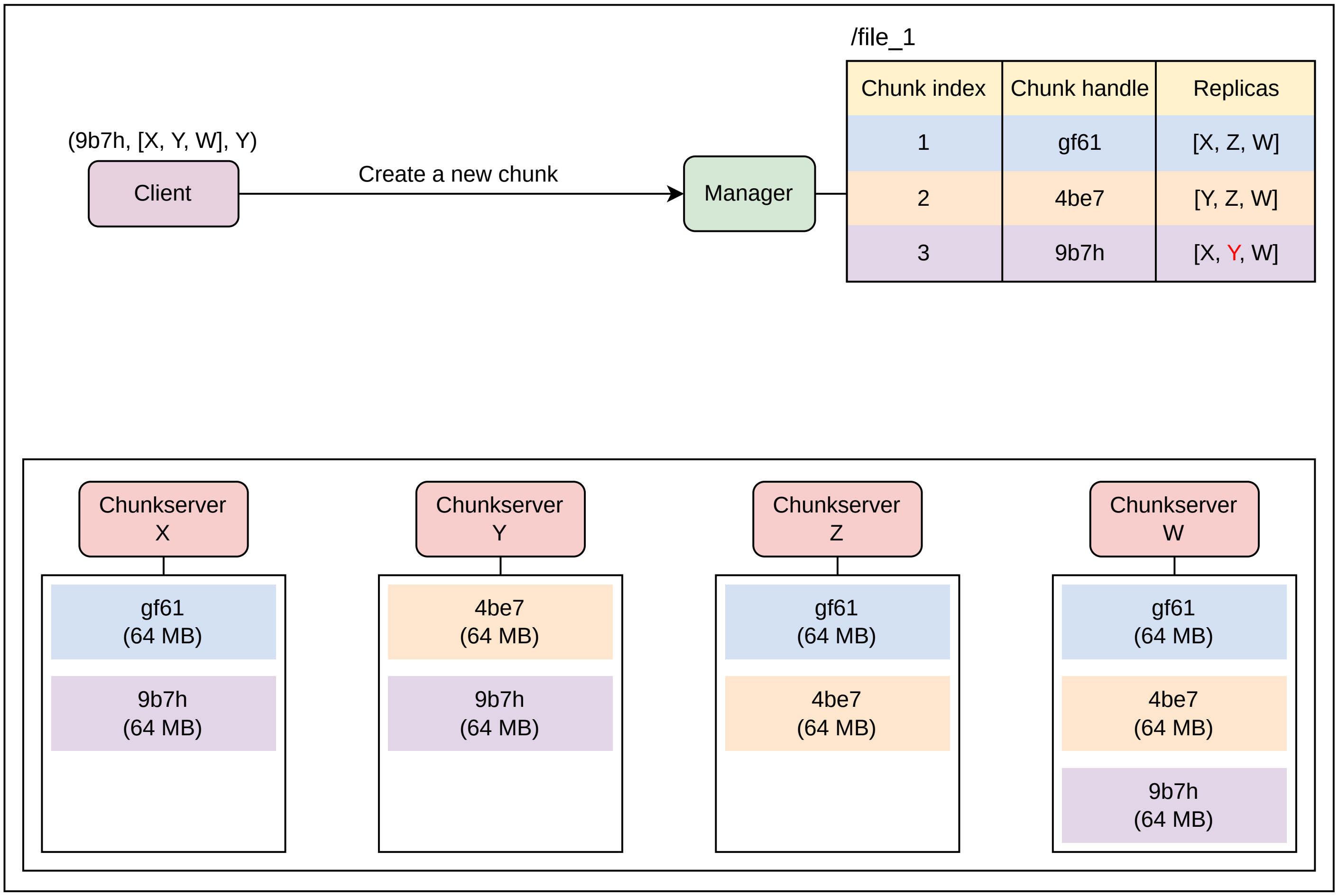
The client pushes the data to all the replicas (not shown) and asks the leased replica (Y) to carry out the append operation



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

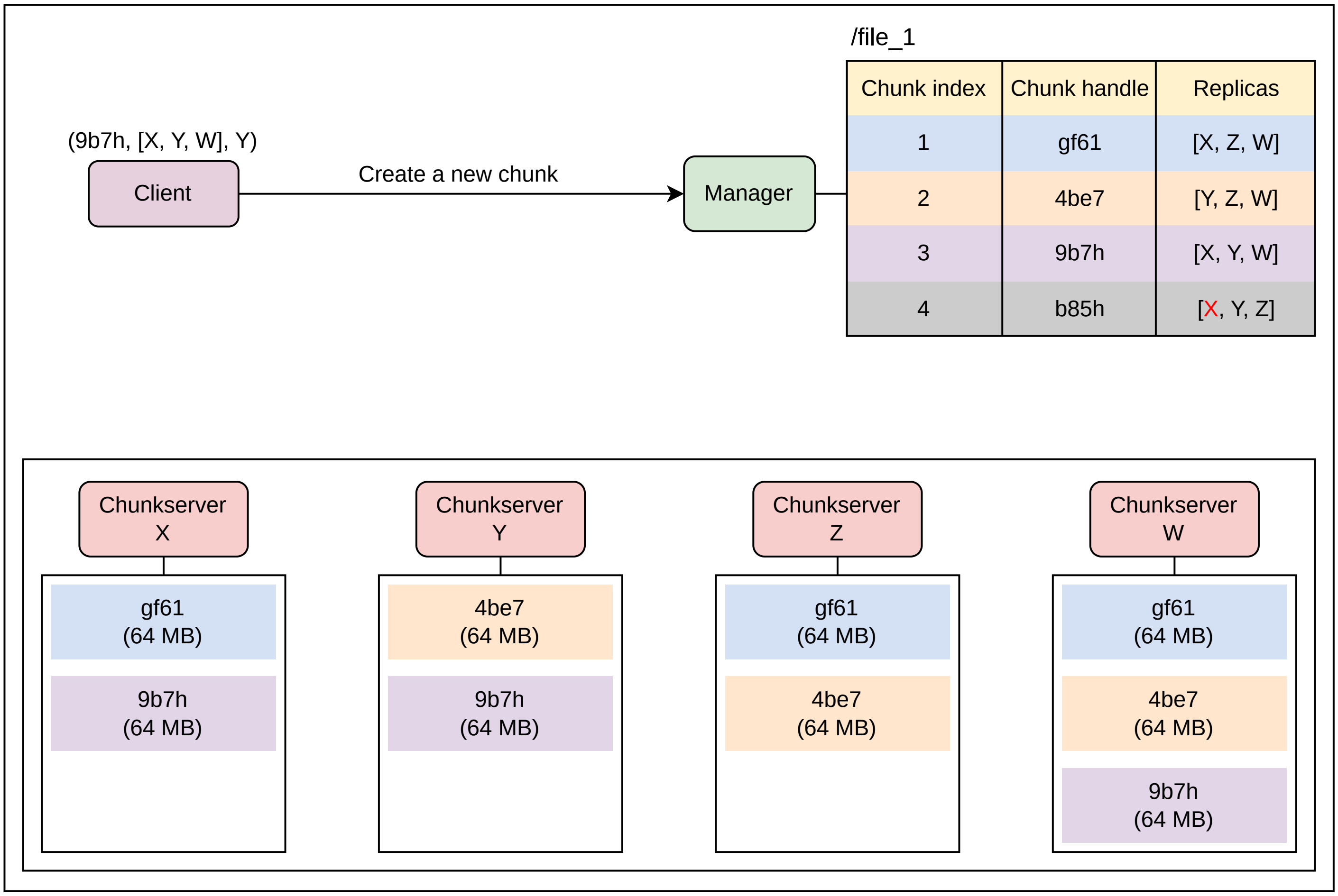
The leased replica finds that the chunk is already full and has no capacity to accommodate a single byte, so it replies to the client that the chunk is full



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

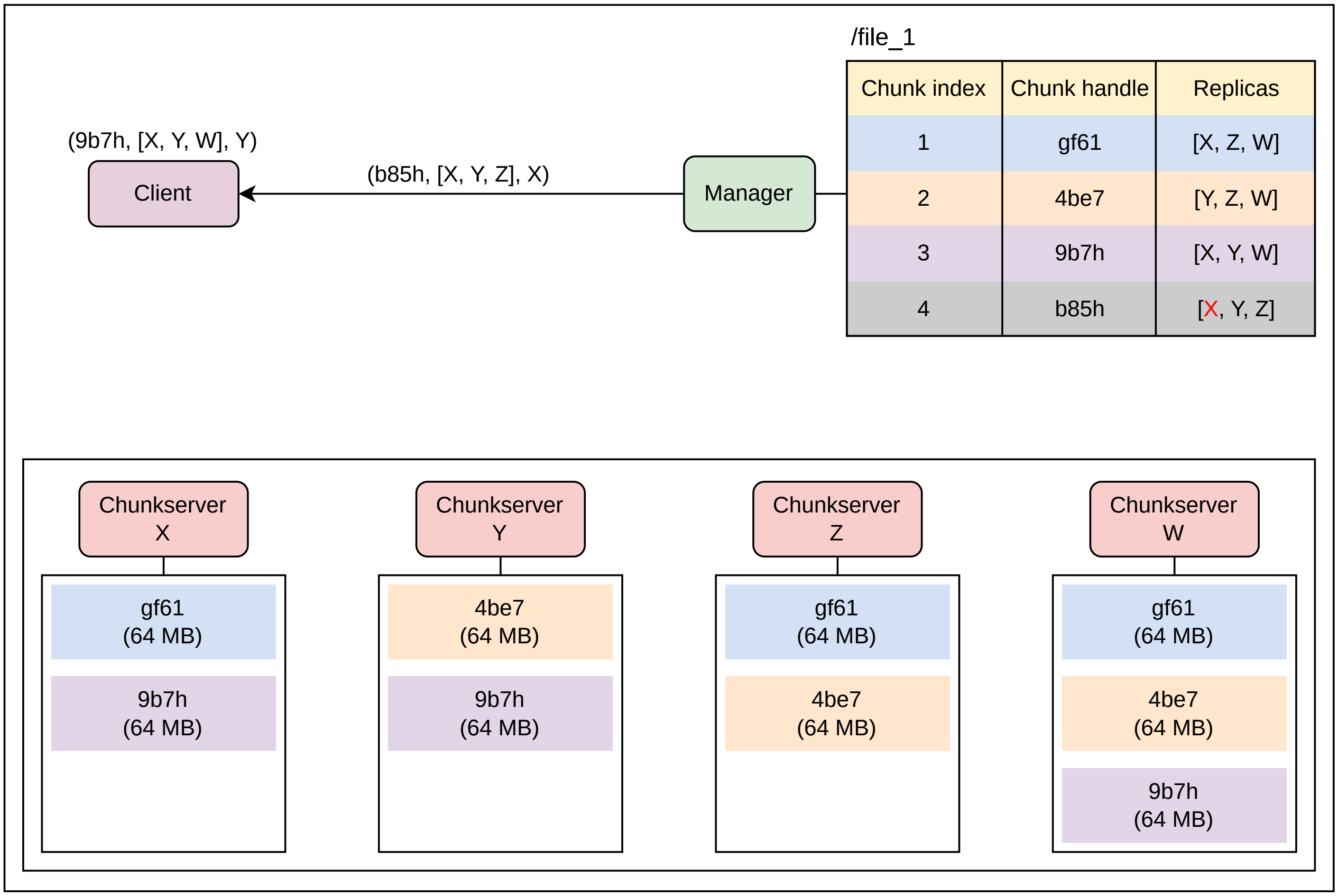
The client asks the manager to create a new chunk, since the last chunk is already full



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The manager generates a new chunk ID (chunk handle), adds an entry in the metadata for the new chunk, and allocates chunkserver to hold the new chunk's data



(9b7h, [X, Y, W], Y)

Client

(b85h, [X, Y, Z], X)

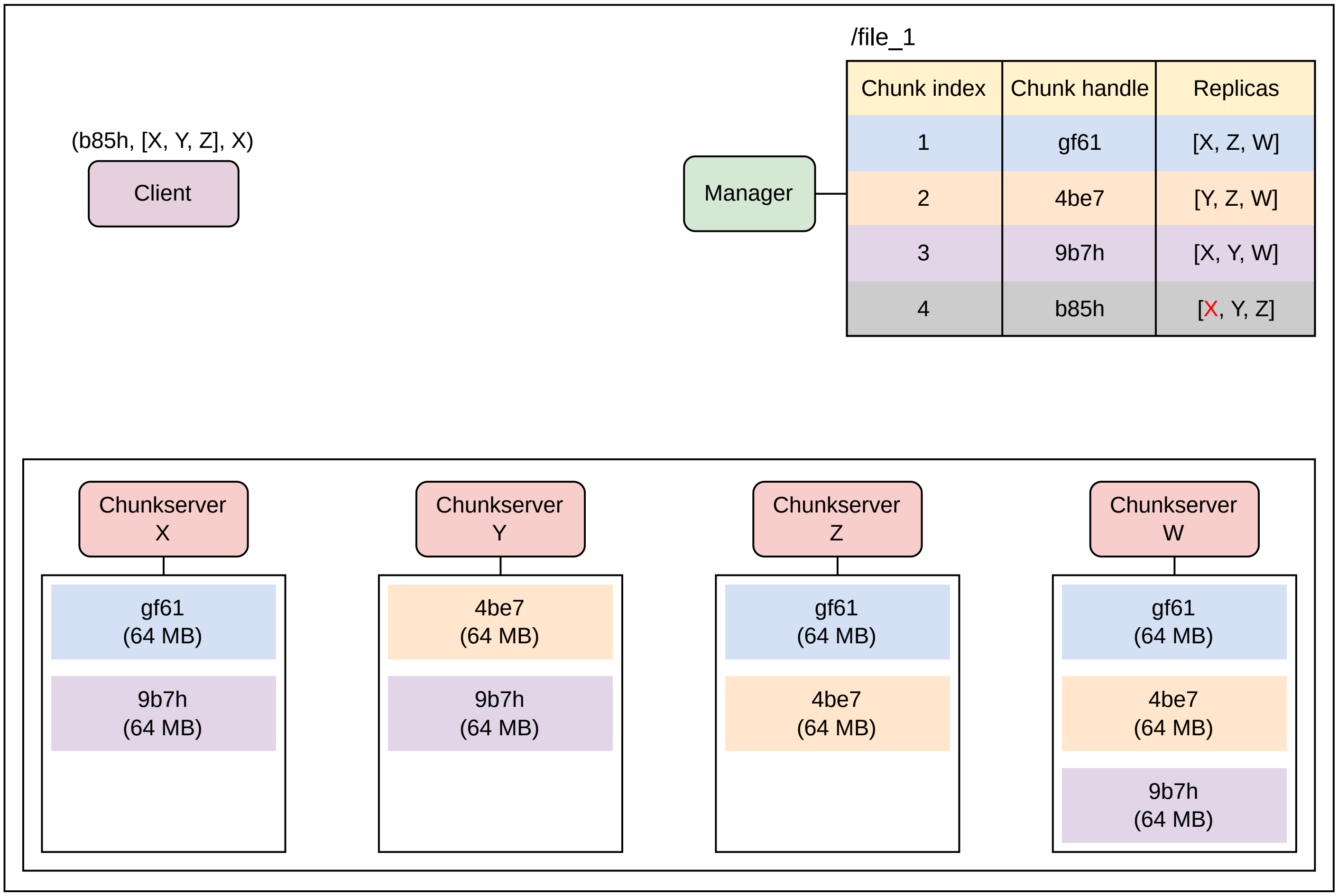
Manager

/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

Chunkserver X	Chunkserver Y	Chunkserver Z	Chunkserver W
gf61 (64 MB)	4be7 (64 MB)	gf61 (64 MB)	gf61 (64 MB)
9b7h (64 MB)	9b7h (64 MB)	4be7 (64 MB)	4be7 (64 MB)
			9b7h (64 MB)

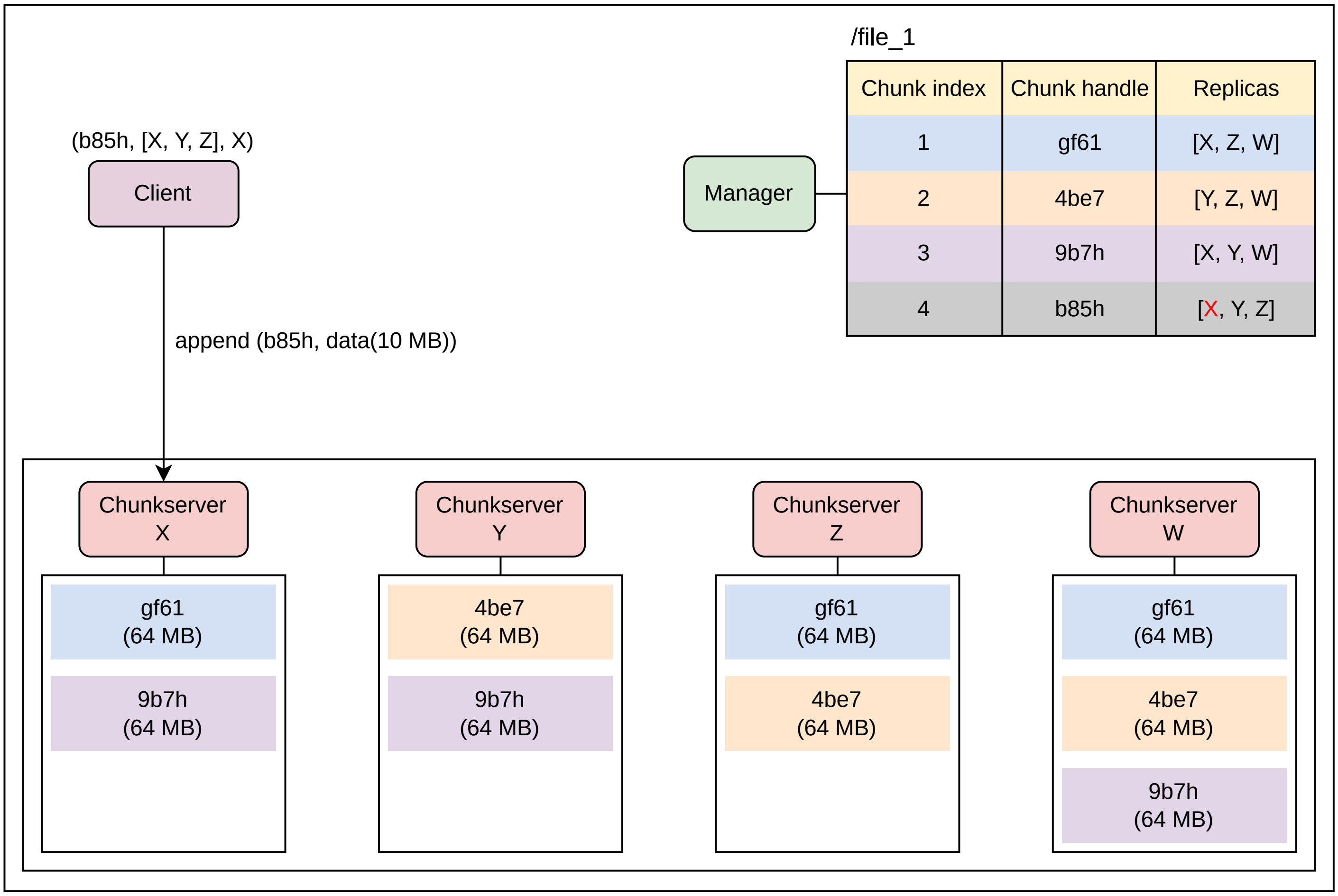
The manager replies to the client with the metadata of the newly created chunk, which consists of a chunk handle, a list of replicas, and the leased replica



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The client caches the metadata to perform further mutations



(b85h, [X, Y, Z], X)

Client

append (b85h, data(10 MB))

Manager

/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

Chunkserver X

gf61 (64 MB)

9b7h (64 MB)

Chunkserver Y

4be7 (64 MB)

9b7h (64 MB)

Chunkserver Z

gf61 (64 MB)

4be7 (64 MB)

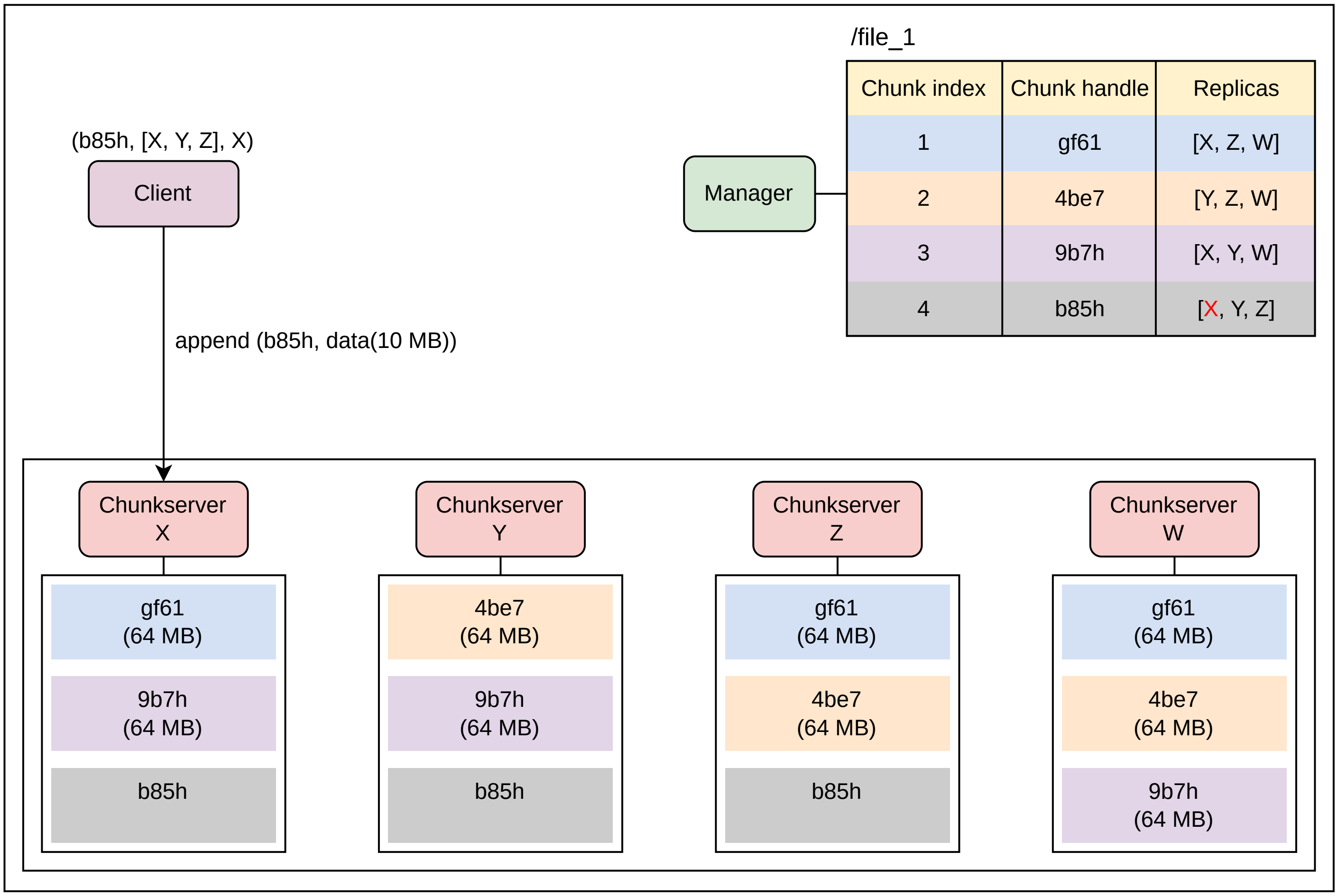
Chunkserver W

gf61 (64 MB)

4be7 (64 MB)

9b7h (64 MB)

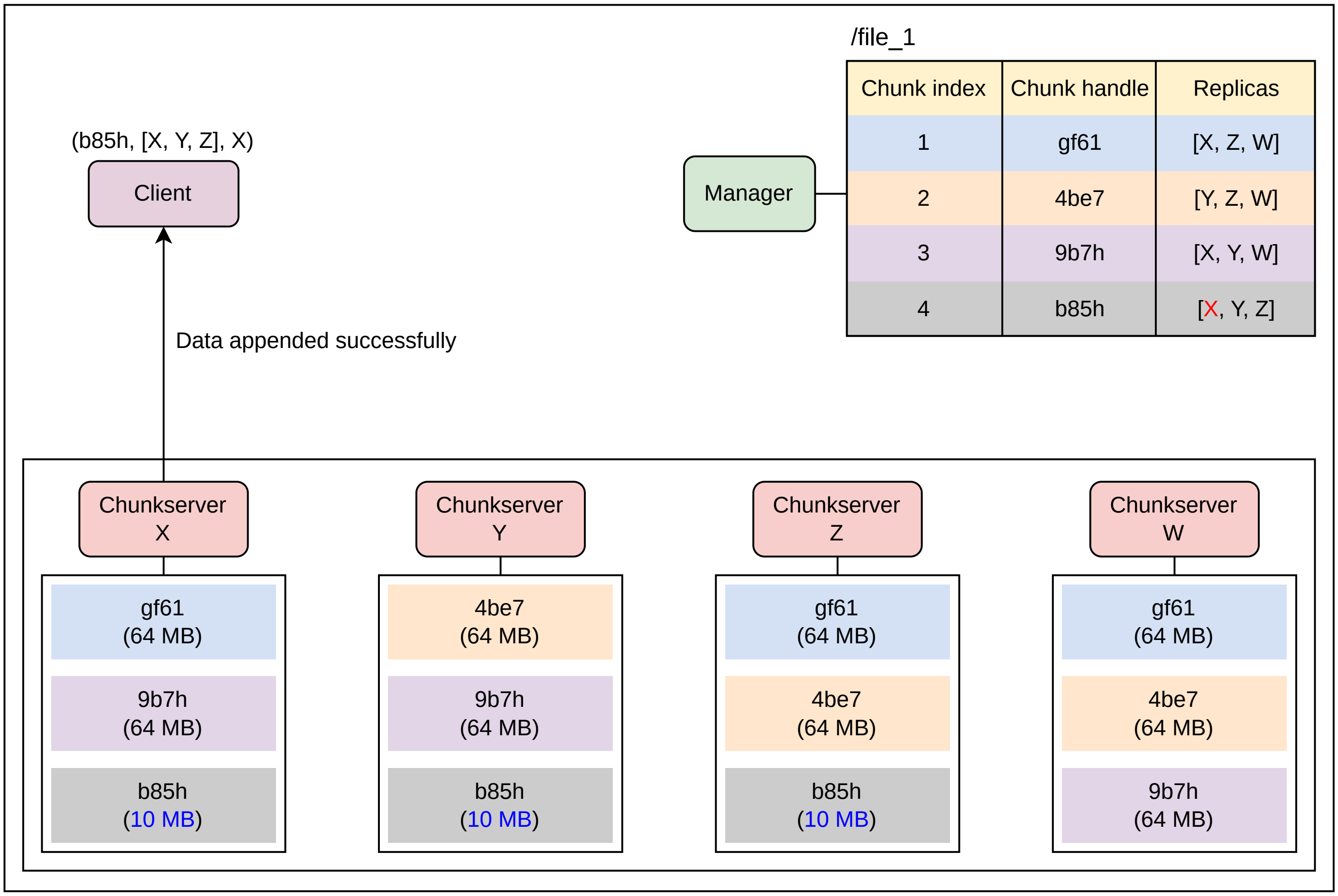
The client pushes the data to all the replicas (not shown) and asks the leased replica (X) to carry out the append operation



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The chunkservers, after receiving the append request on a chunk that they don't know about, confirm the chunk with the manager via a heartbeat message and add it to their list of chunks



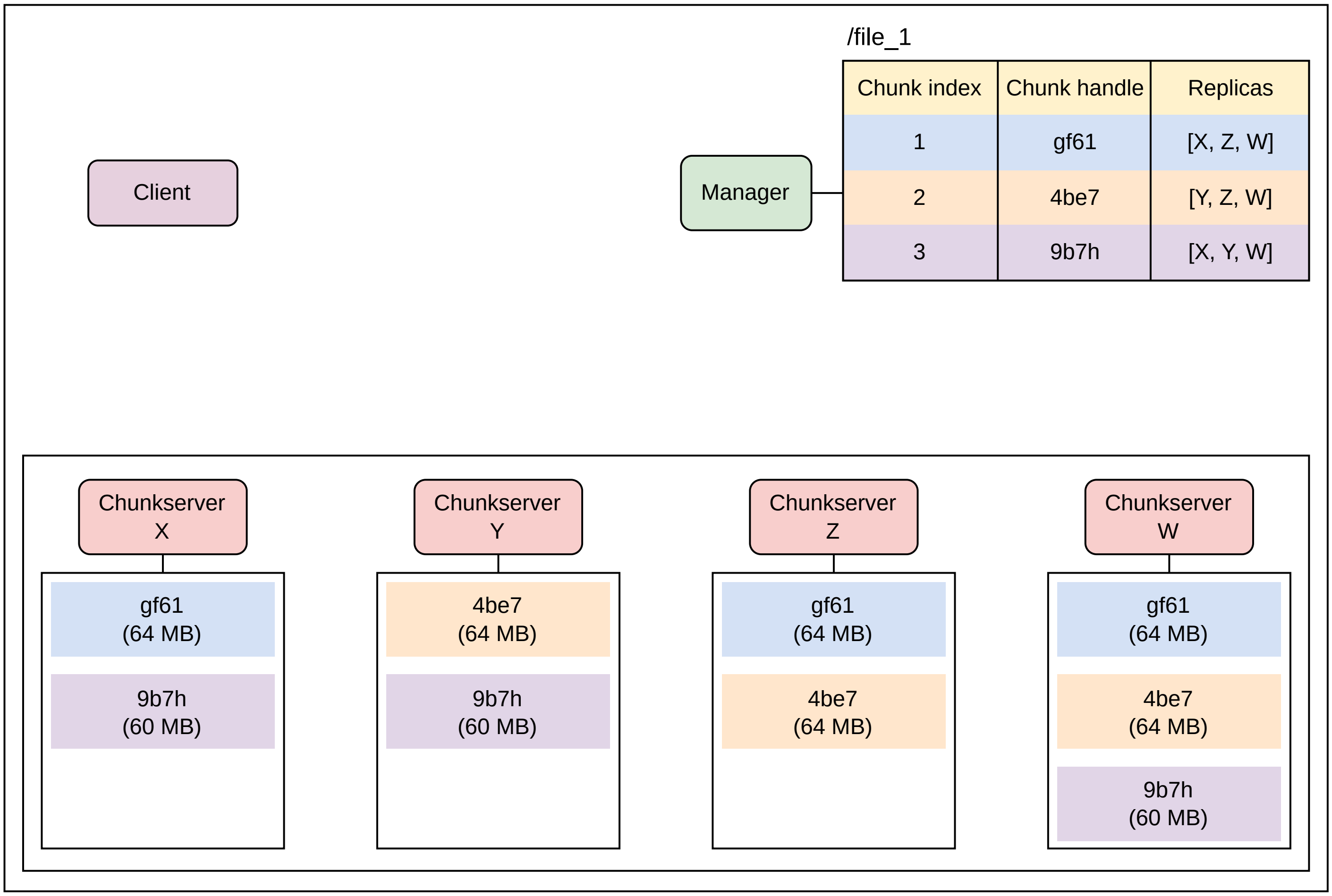
/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]
4	b85h	[X, Y, Z]

The replicas append the data to the newly created chunk and the leased replica successfully responds to the client

14 of 14

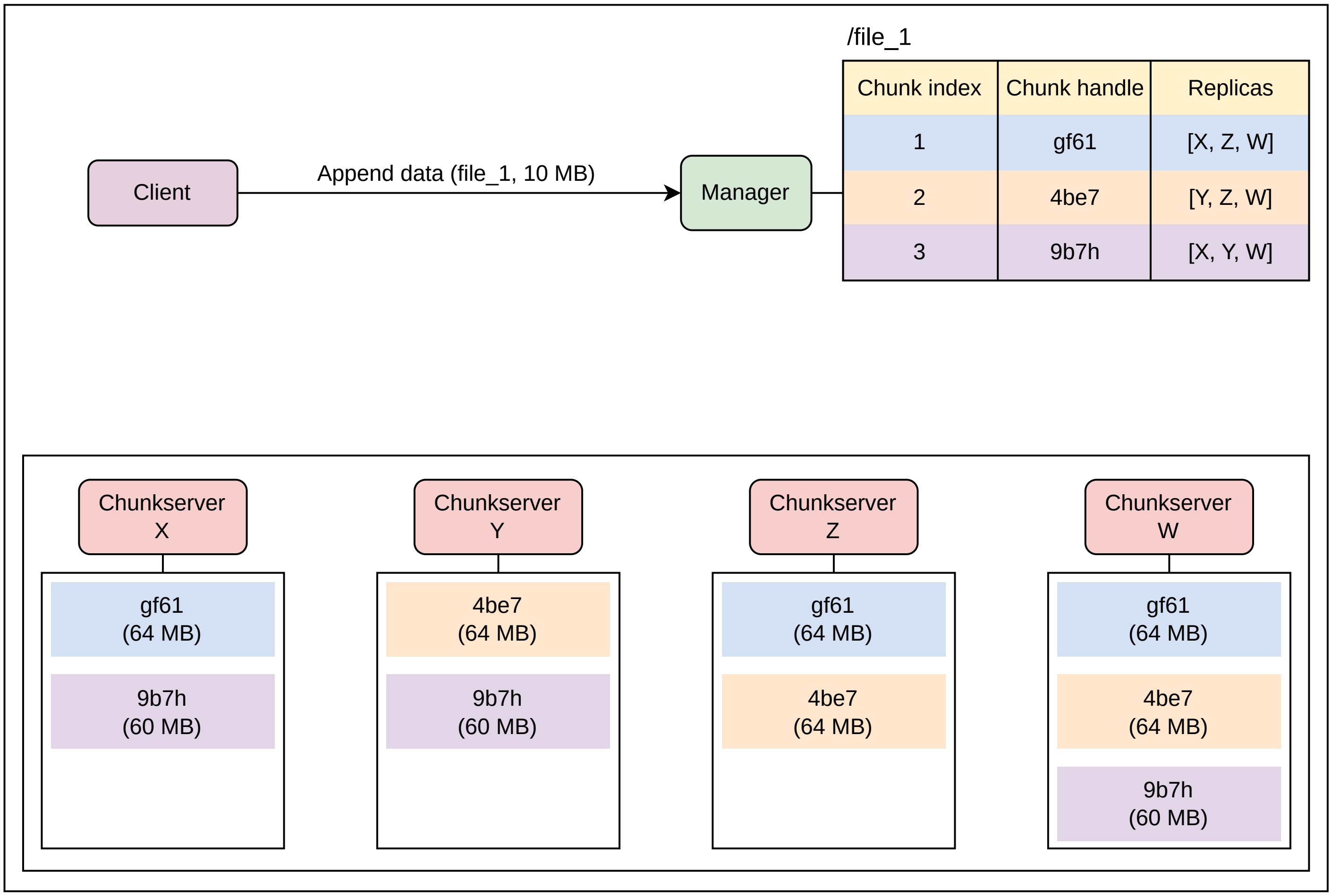
3. It is possible that the last chunk is not full, but doesn't have the capacity to accommodate all data bytes in the append request. In this case, the chunkserver holding the last chunk will respond to the client with a message that the available space in that chunk is less than the size of the write. The chunkserver likely adds information about how much available space it has on the last chunk, based on which the client will split the writing data into two. The first part of the split will be written to the end of the last chunk, and for the second part, the client asks the manager to generate a new chunk. The remaining data will be written at the start of the newly created chunk.



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

The system consists of a client, a manager node, and the chunkservers



/file_1

Chunk index	Chunk handle	Replicas
1	gf61	[X, Z, W]
2	4be7	[Y, Z, W]
3	9b7h	[X, Y, W]

The client puts an append request that is received at the manager node