15-440 Distributed Systems

Recitation 7

Slides By: Hend Gedawy

& Previous TAs

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Announcements

- **P1** Done!
- **P2 Out** (due October 24)
- **Midterm** (October 15)
- **PS3** (due October 19)

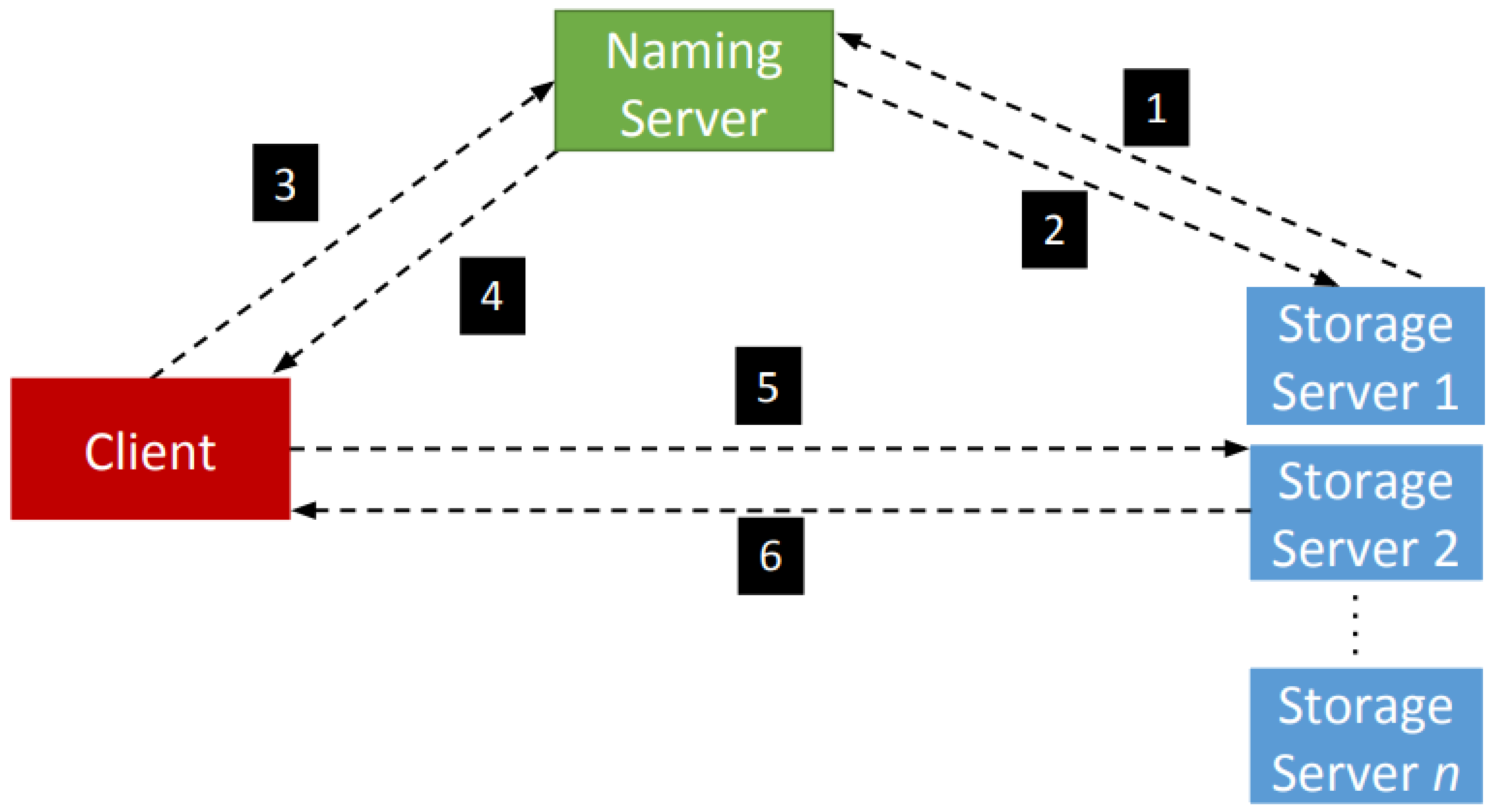
Outline

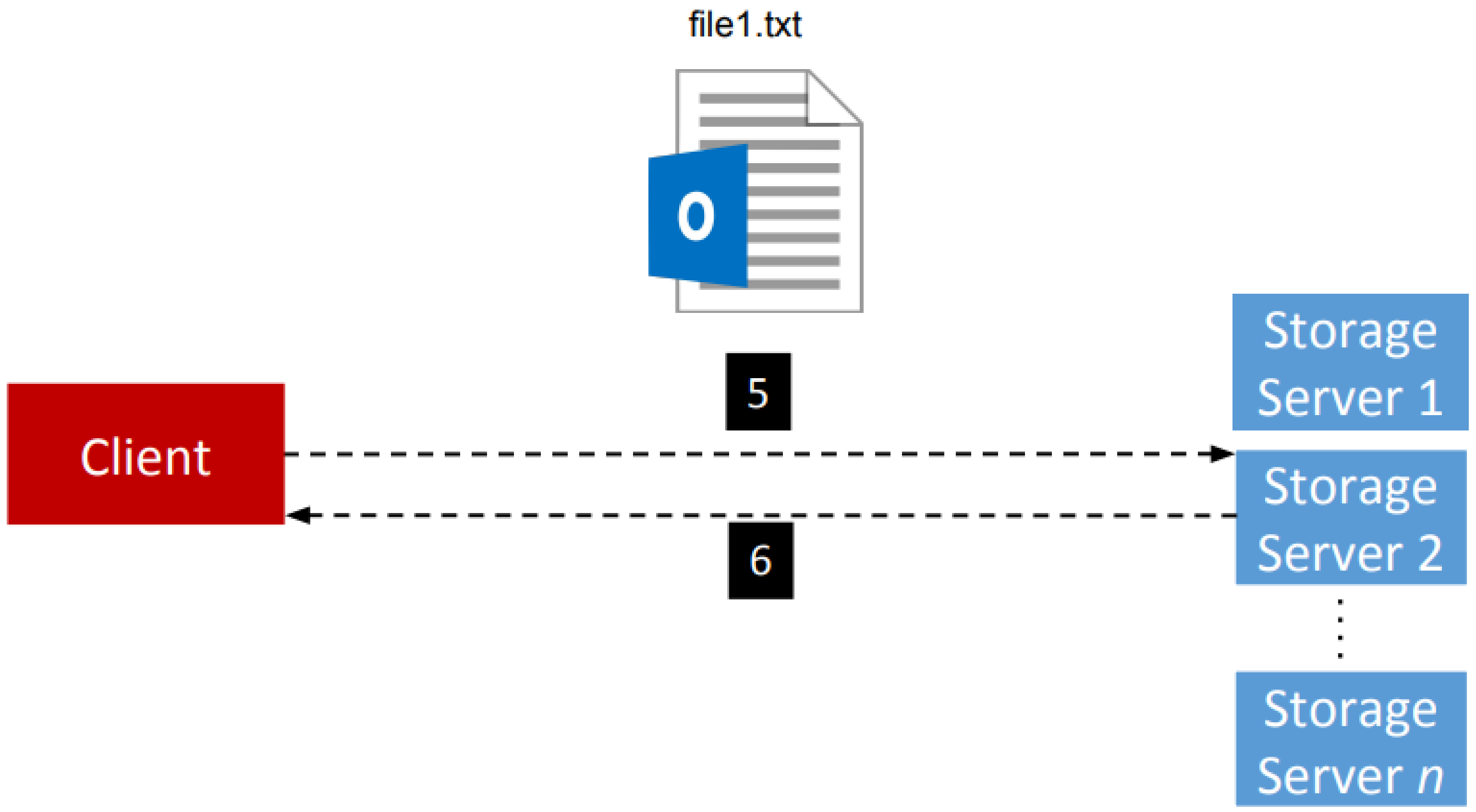
- Project 2 Overview & Objective
- Synchronization of File Readers & Writers
- Dynamic Replication of Files
- Implementation Tips

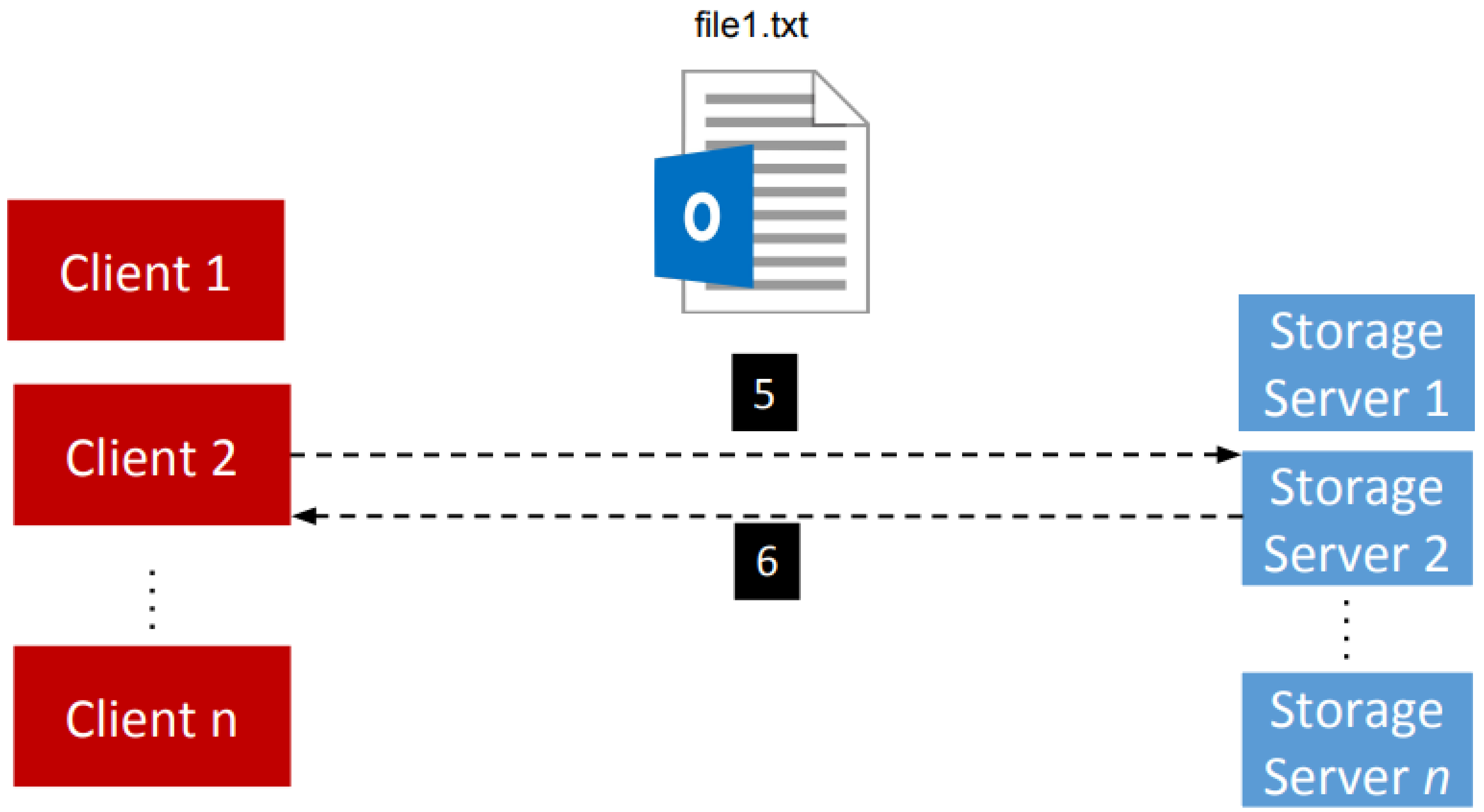
Project 2

- Involves *building on your Project 1 Distributed File System* (**DFS**): *FileStack*
- P2_StarterCode:
  - Follow the Handout on what files you will copy from project 1
- Release Date: **October 5th**
- Due date: **October 24th**

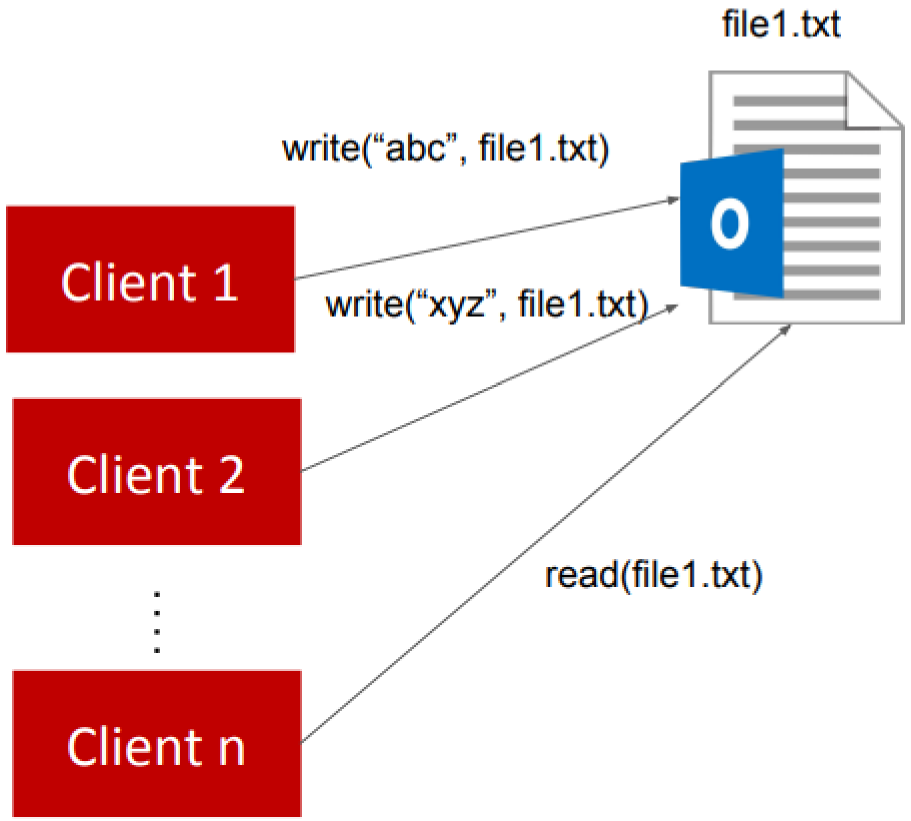
FileStack Architecture

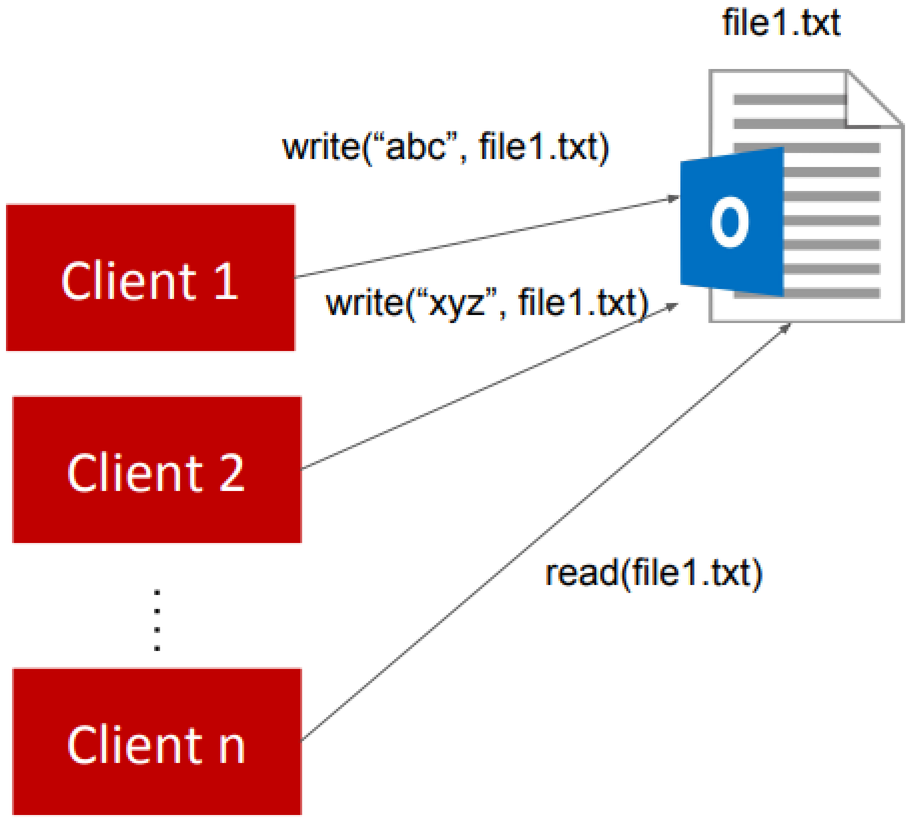



file1.txt
5
Client
6
Storage Server 1
Storage Server 2
Storage Server n


file1.txt
Client 1
Client 2
Client n
5
6
Storage Server 1
Storage Server 2
Storage Server *n*

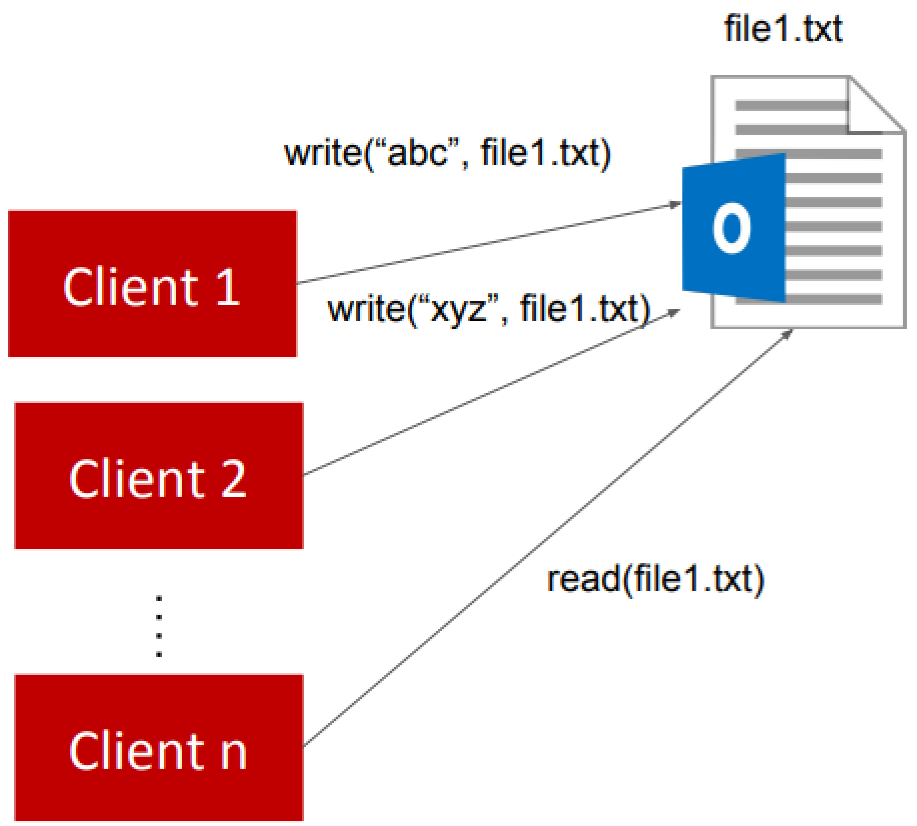
What might go wrong?





What might go wrong?

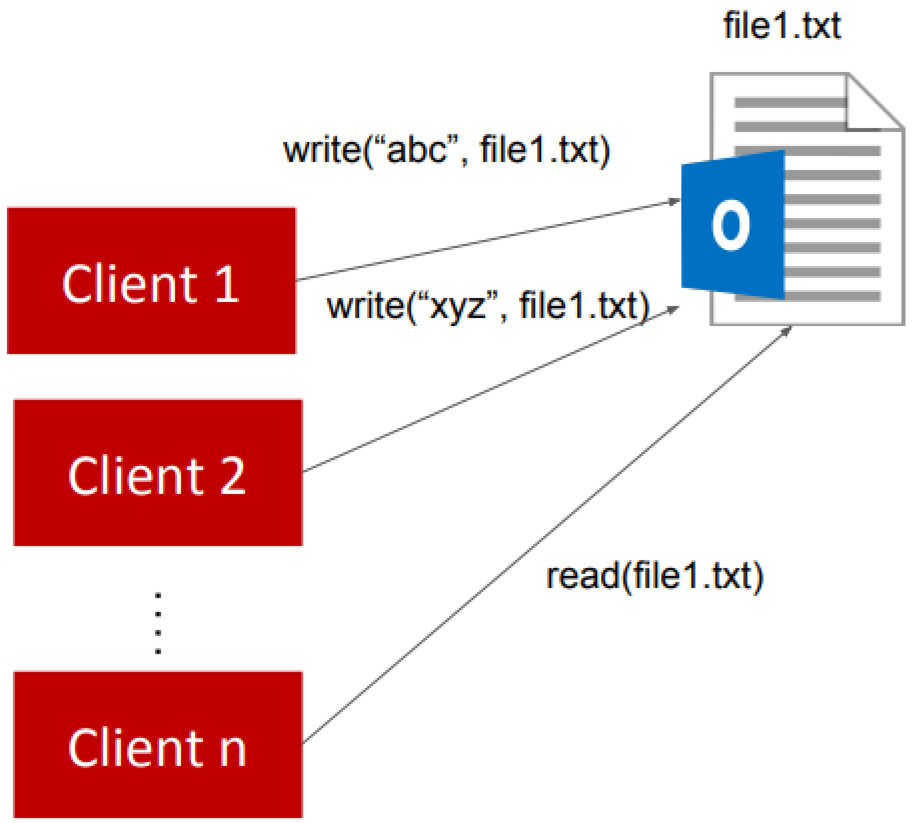
- Synchronization



What might go wrong?

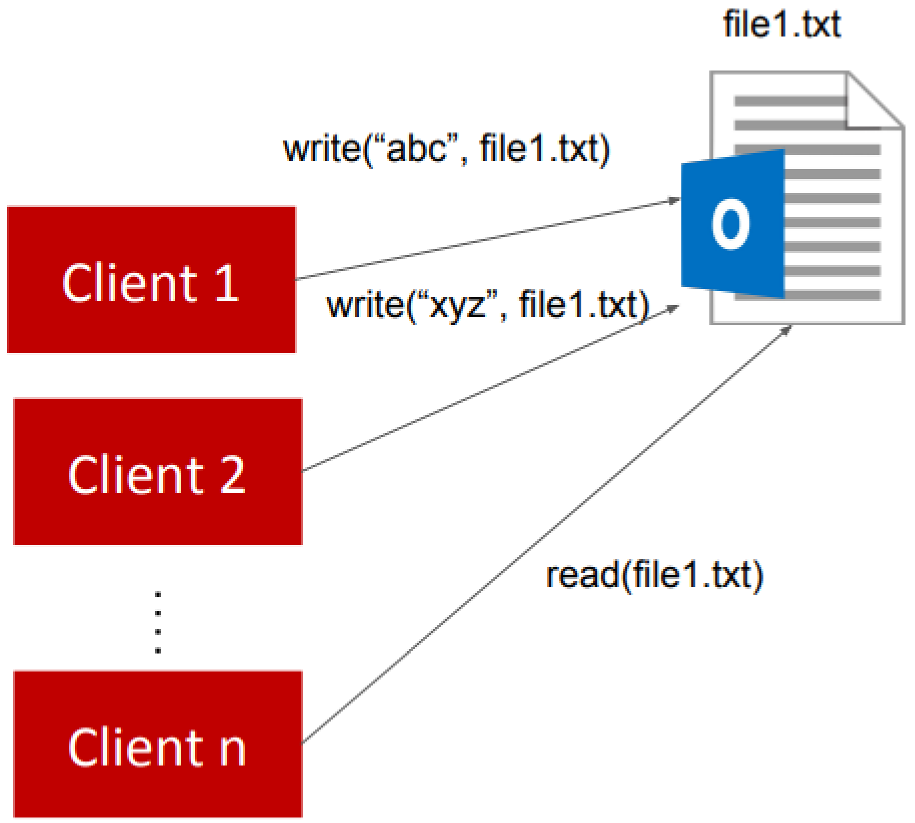
- Synchronization

file1.txt is hosted on SS9, and it's gets 5000 reqs/ sec. As opposed to file2.txt which gets 1000 reqs / month on SS3



What might go wrong?

- Synchronization
- Load-balancing



What might go wrong?

- Synchronization
- Load-balancing

Replicate file1.txt on multiple Storage Servers

Carnegie Mellon University Qatar

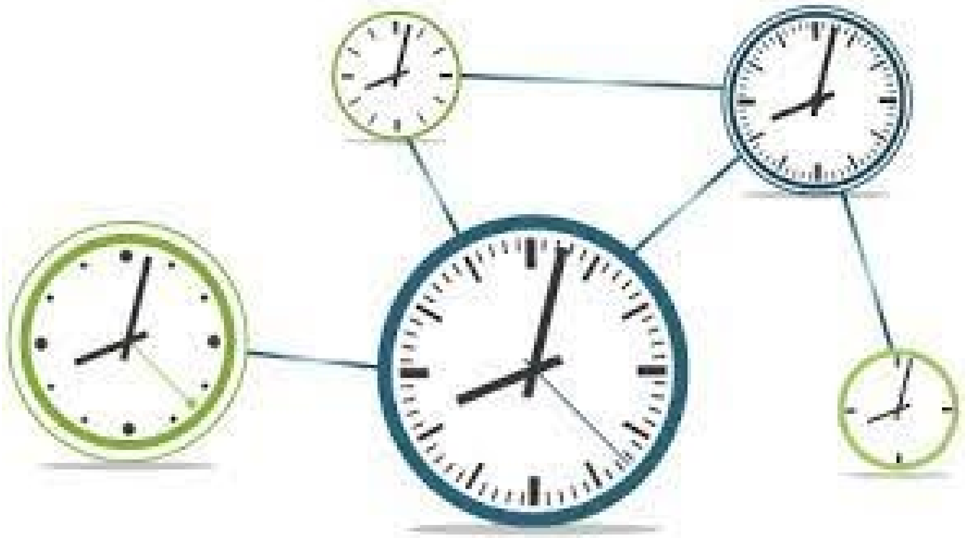
Project 2 Objectives

1. **Devise and apply a synchronization algorithm that:**
   - achieves *correctness* while sharing files
   - and ensures *fairness* to clients.

2. **Devise and apply a replication algorithm that:**
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

Outline

- Project 2 Overview
- Synchronization of File Readers & Writers
- Dynamic Replication of Files
- Implementation Tips



Project 2 Objectives

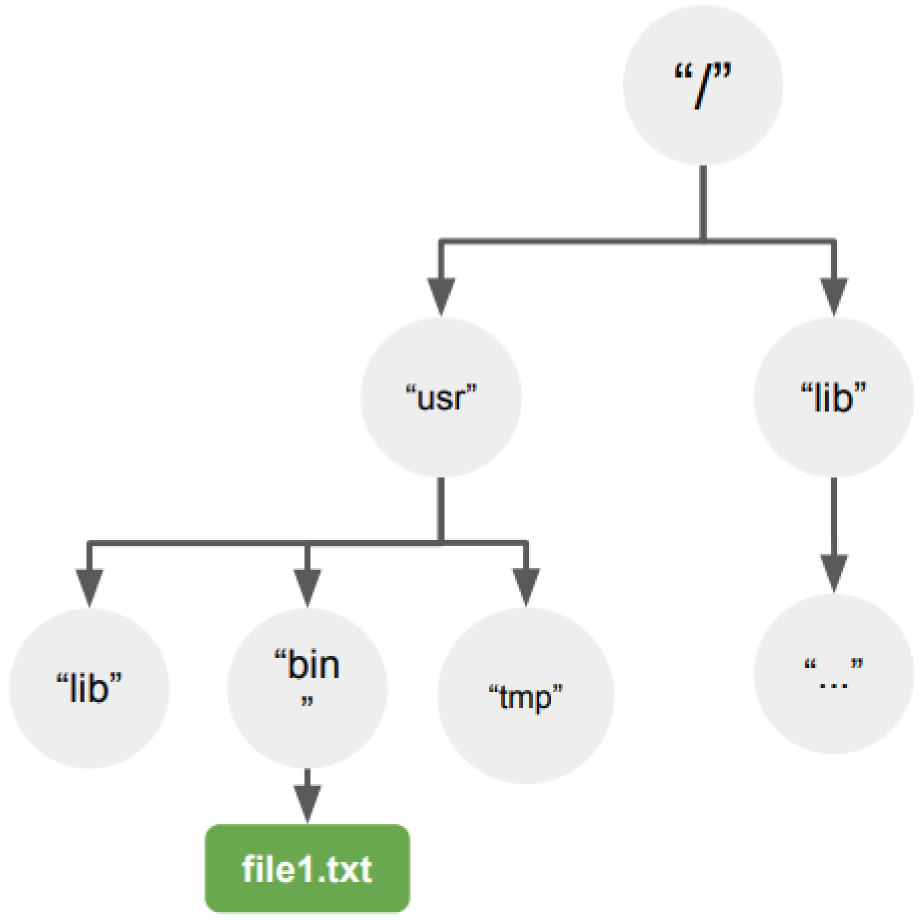
1. Logical Synchronization of Readers and Writers

2. **Devise and apply a replication algorithm that:**
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

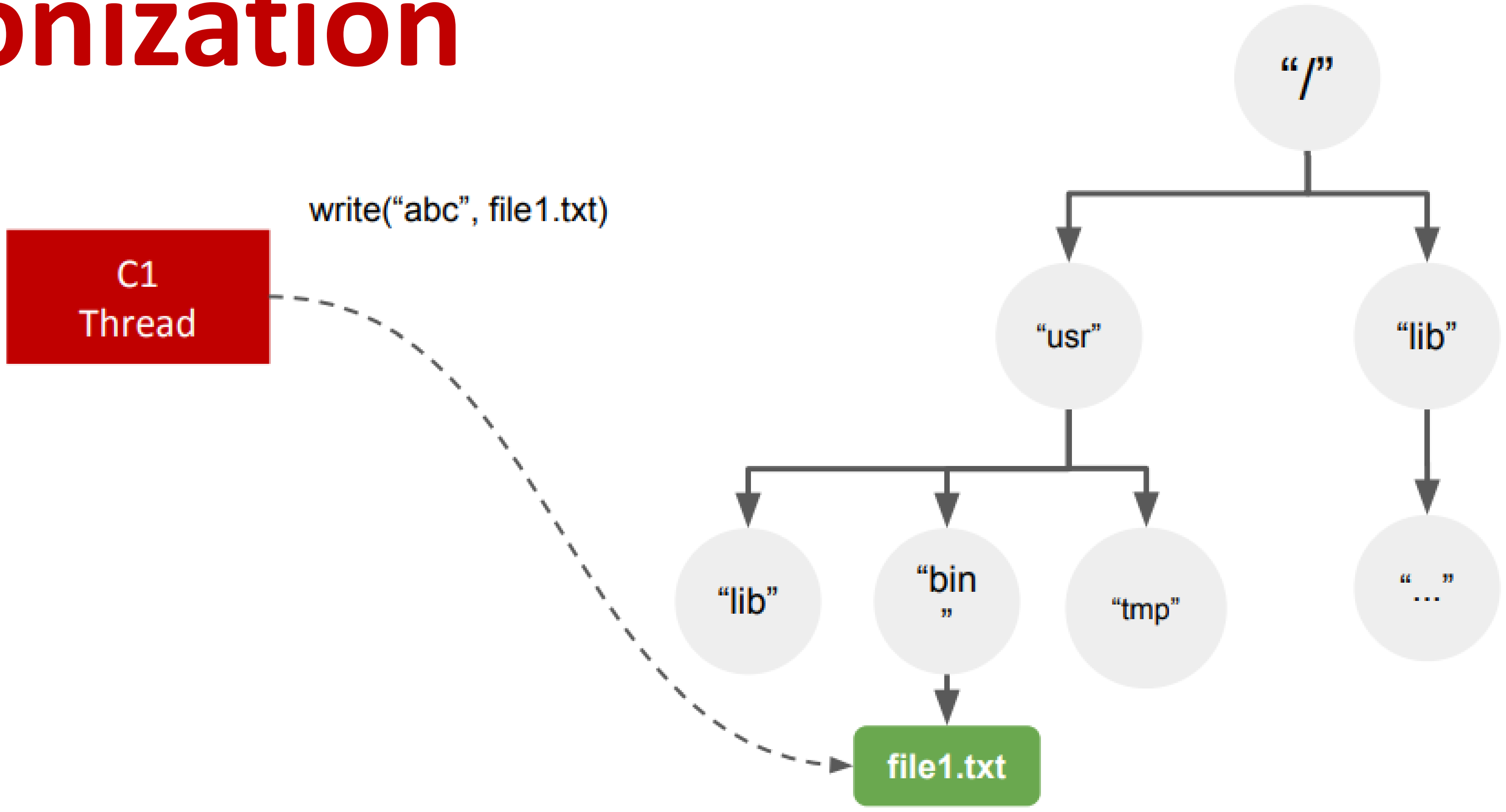
Synchronization - Questions

- What to lock?
- How to handle read lock requests?
- How to handle write lock requests?
- How to ensure fair access?

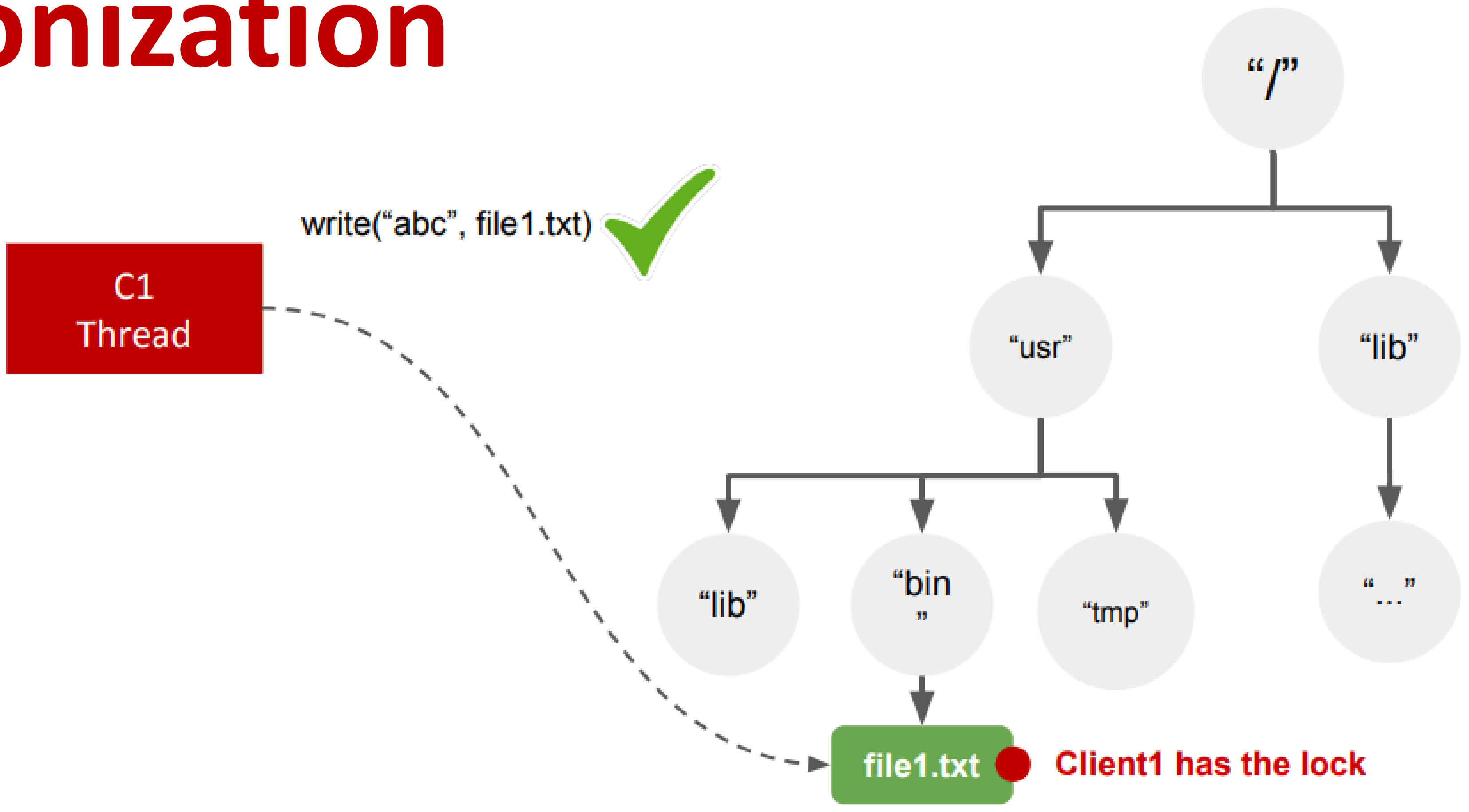
Synchronization



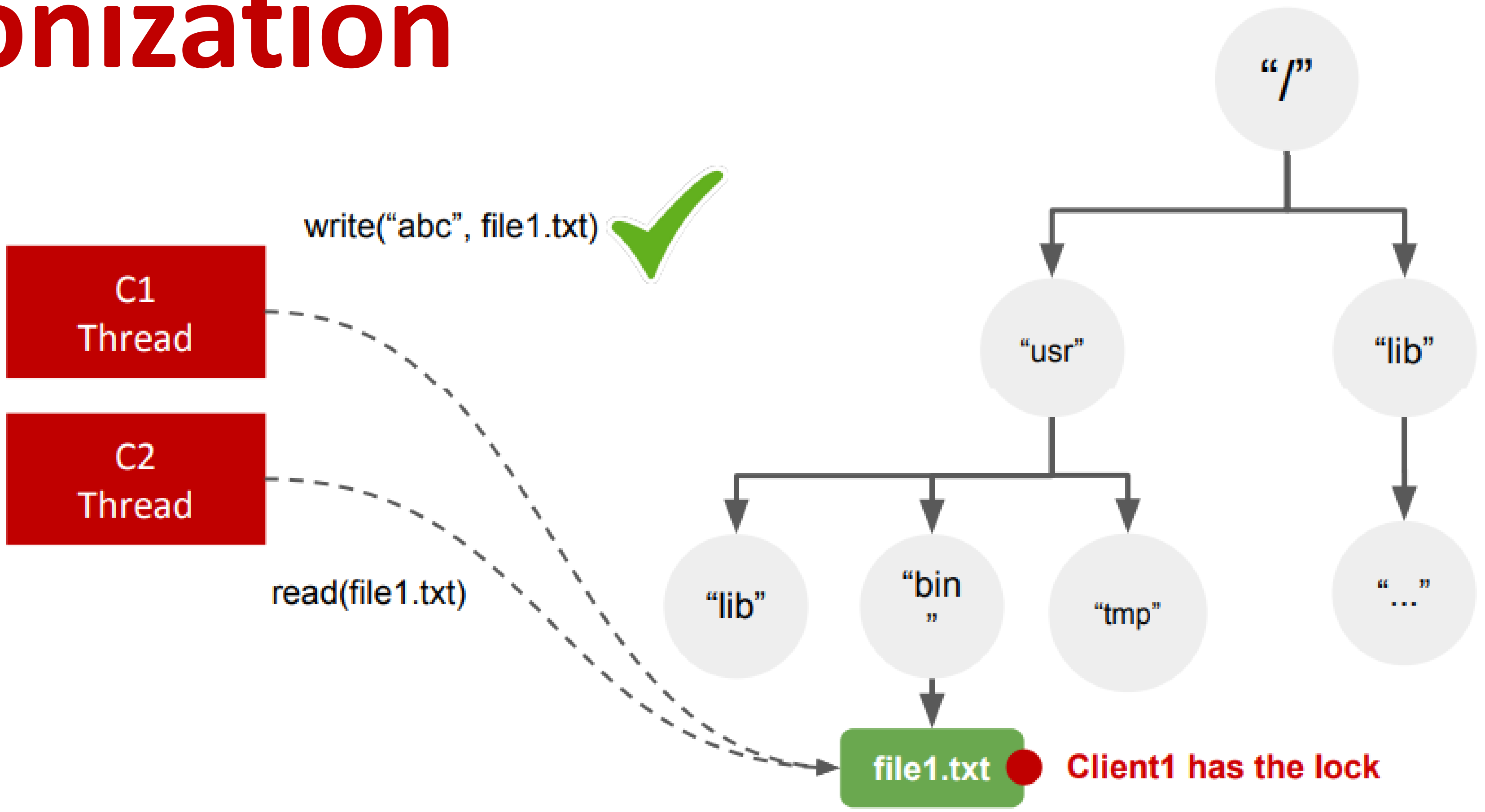
Synchronization



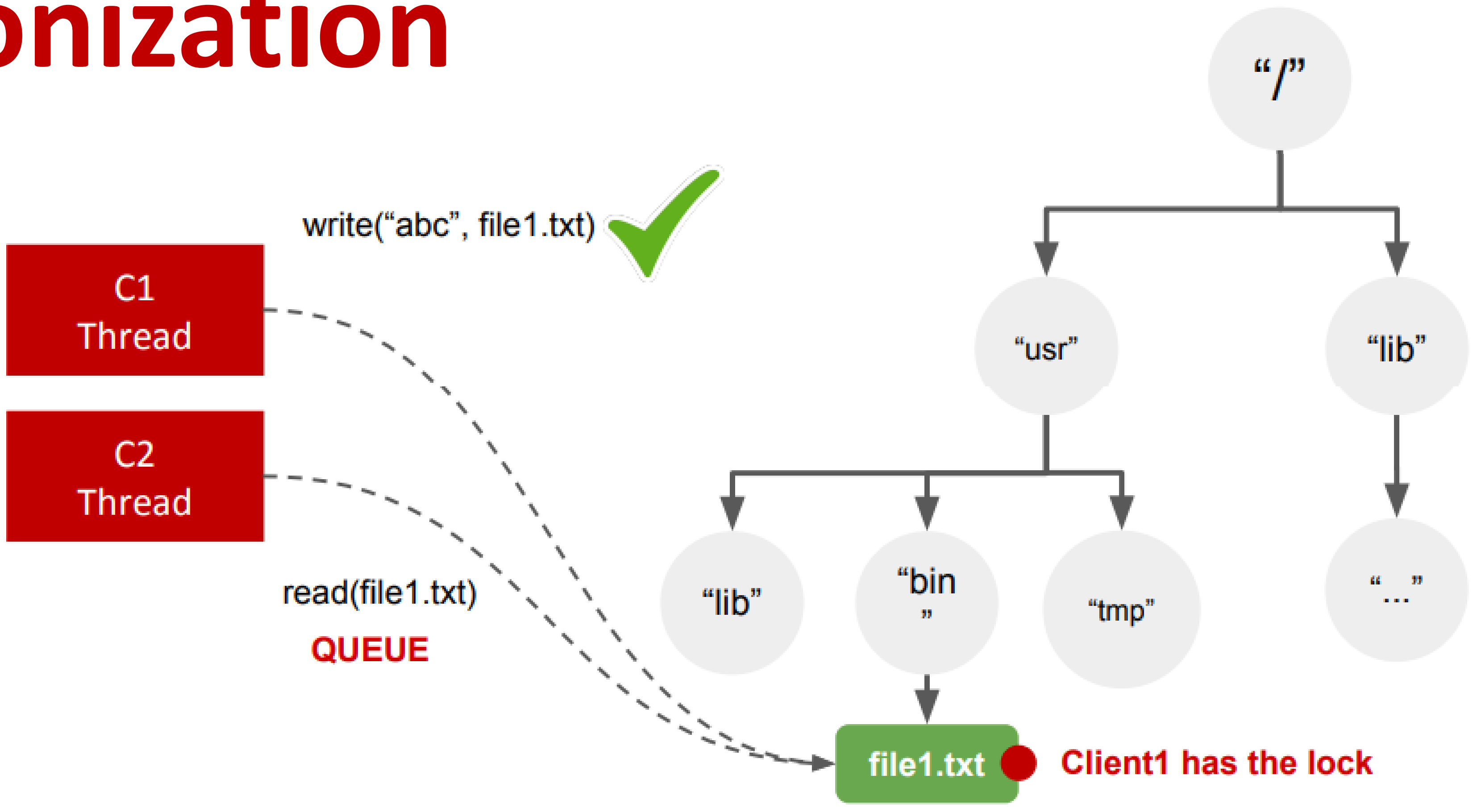
Synchronization



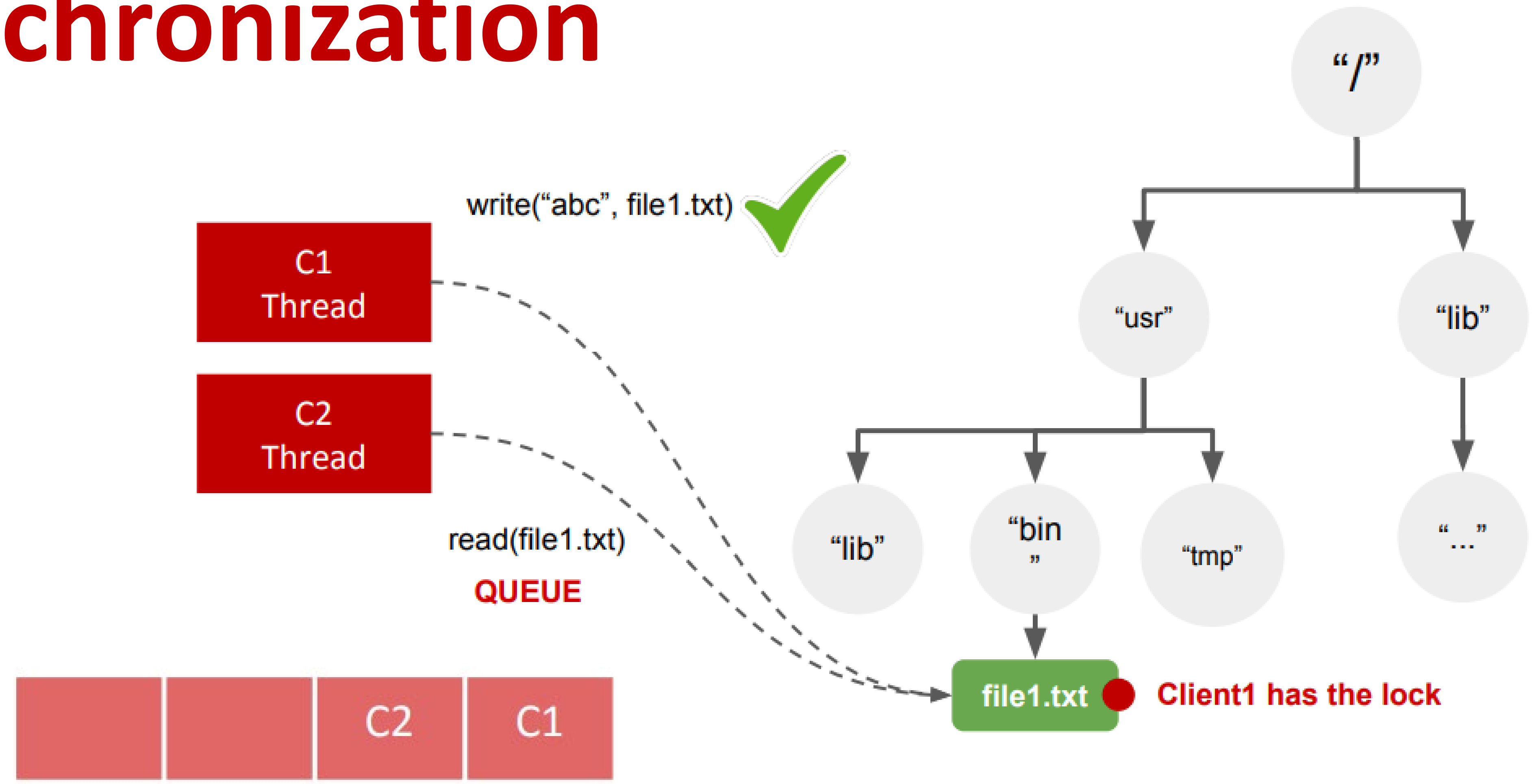
Synchronization



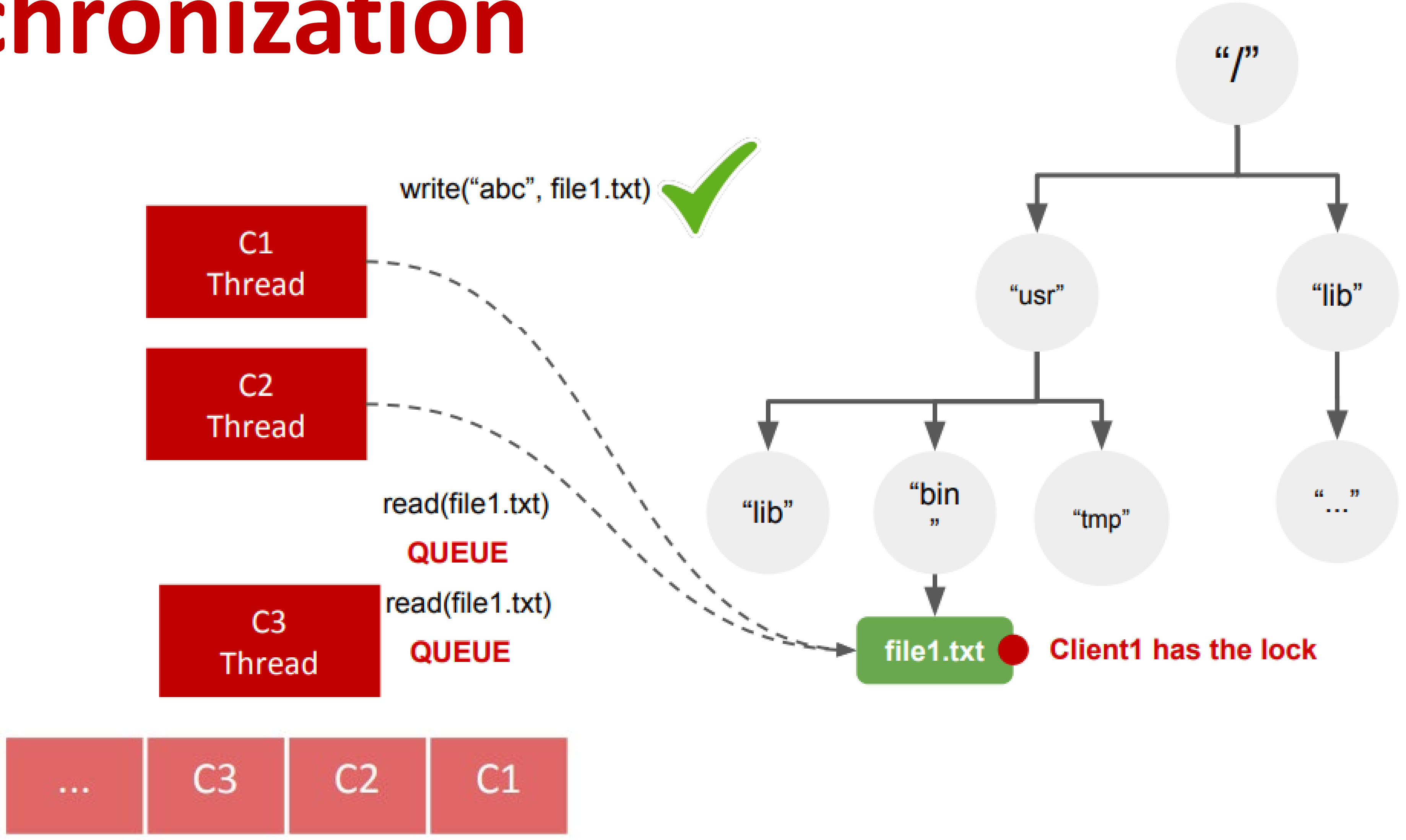
Synchronization



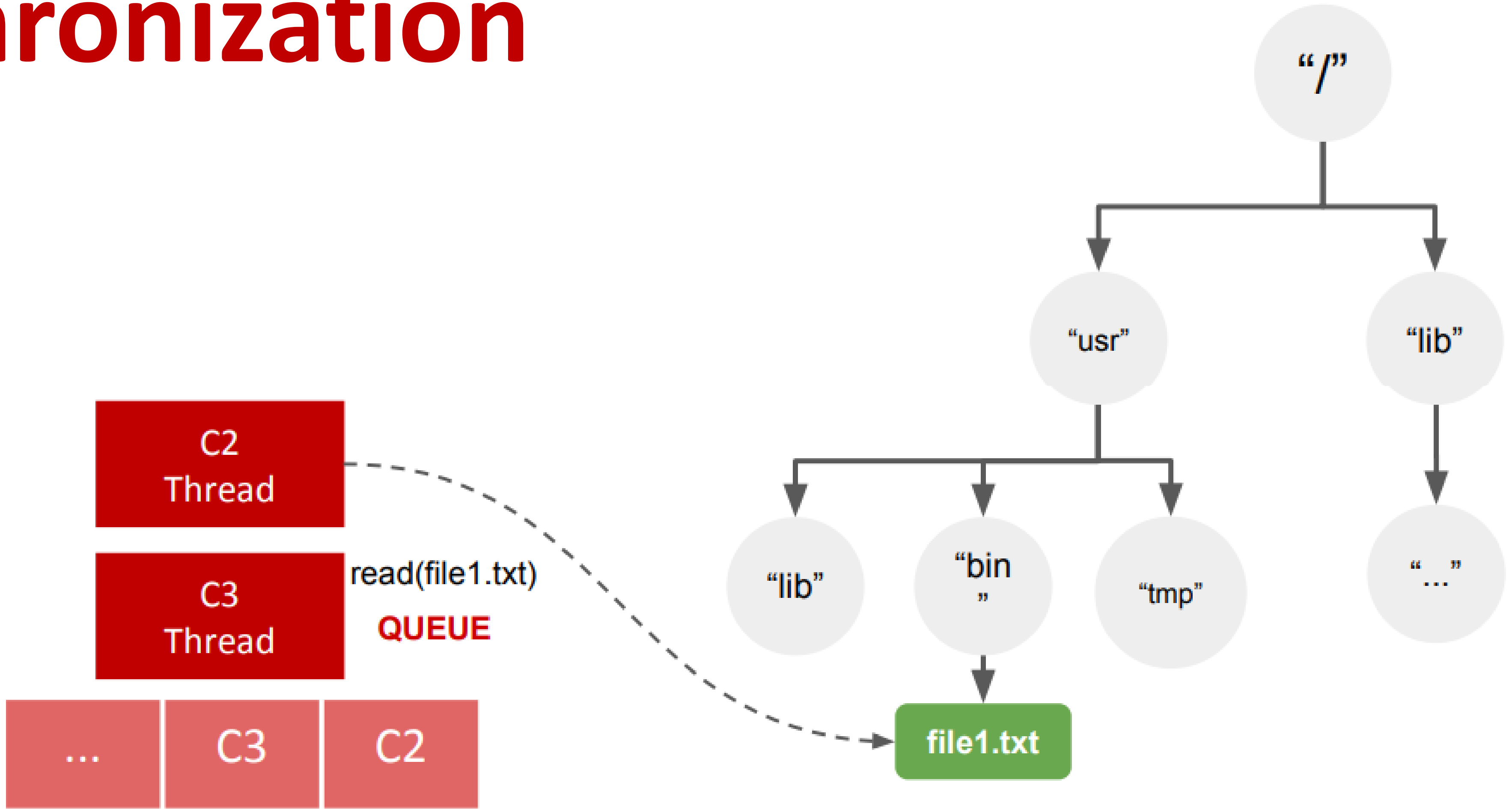
Synchronization



Synchronization

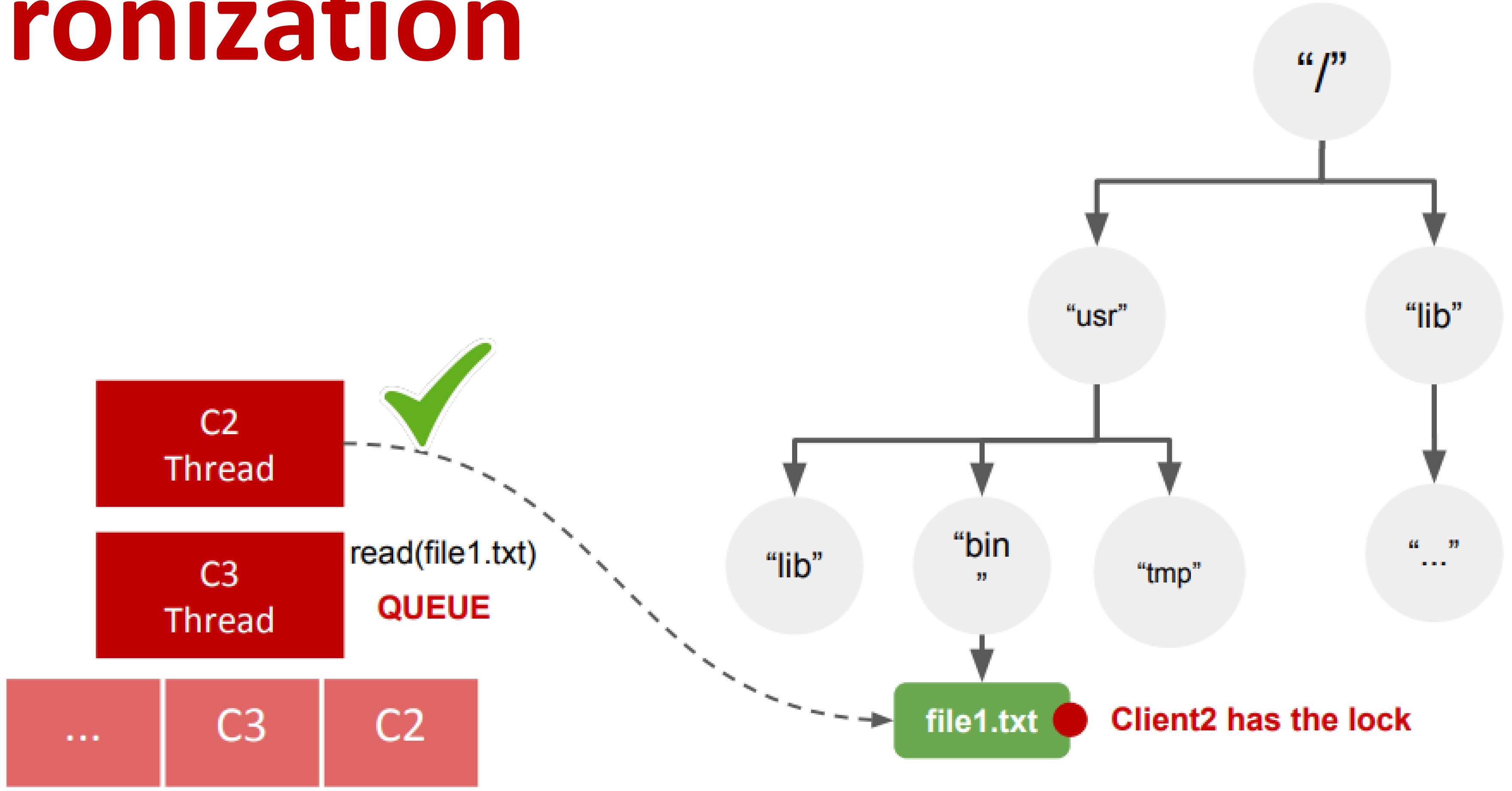


Synchronization

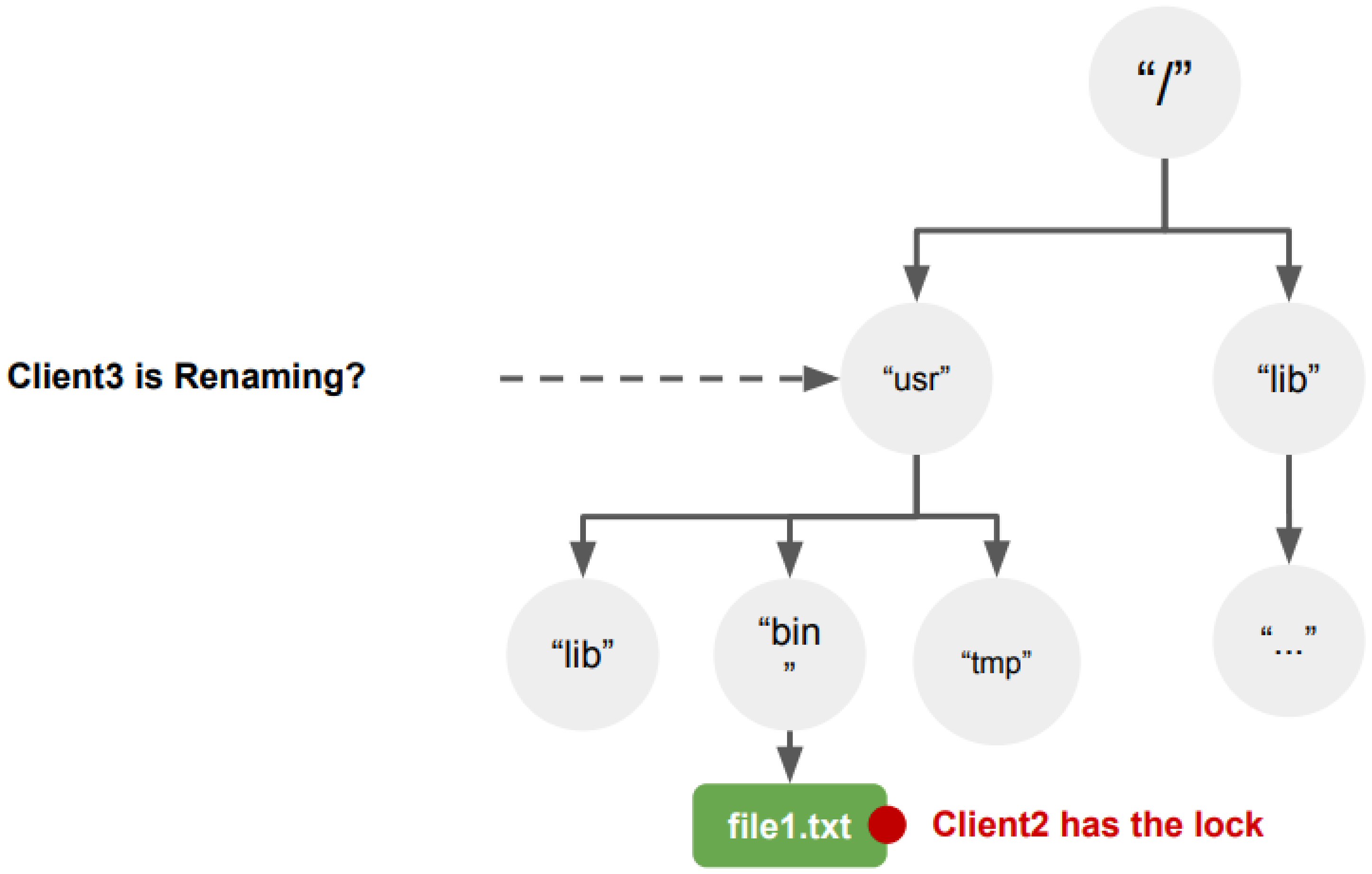


Synchronization

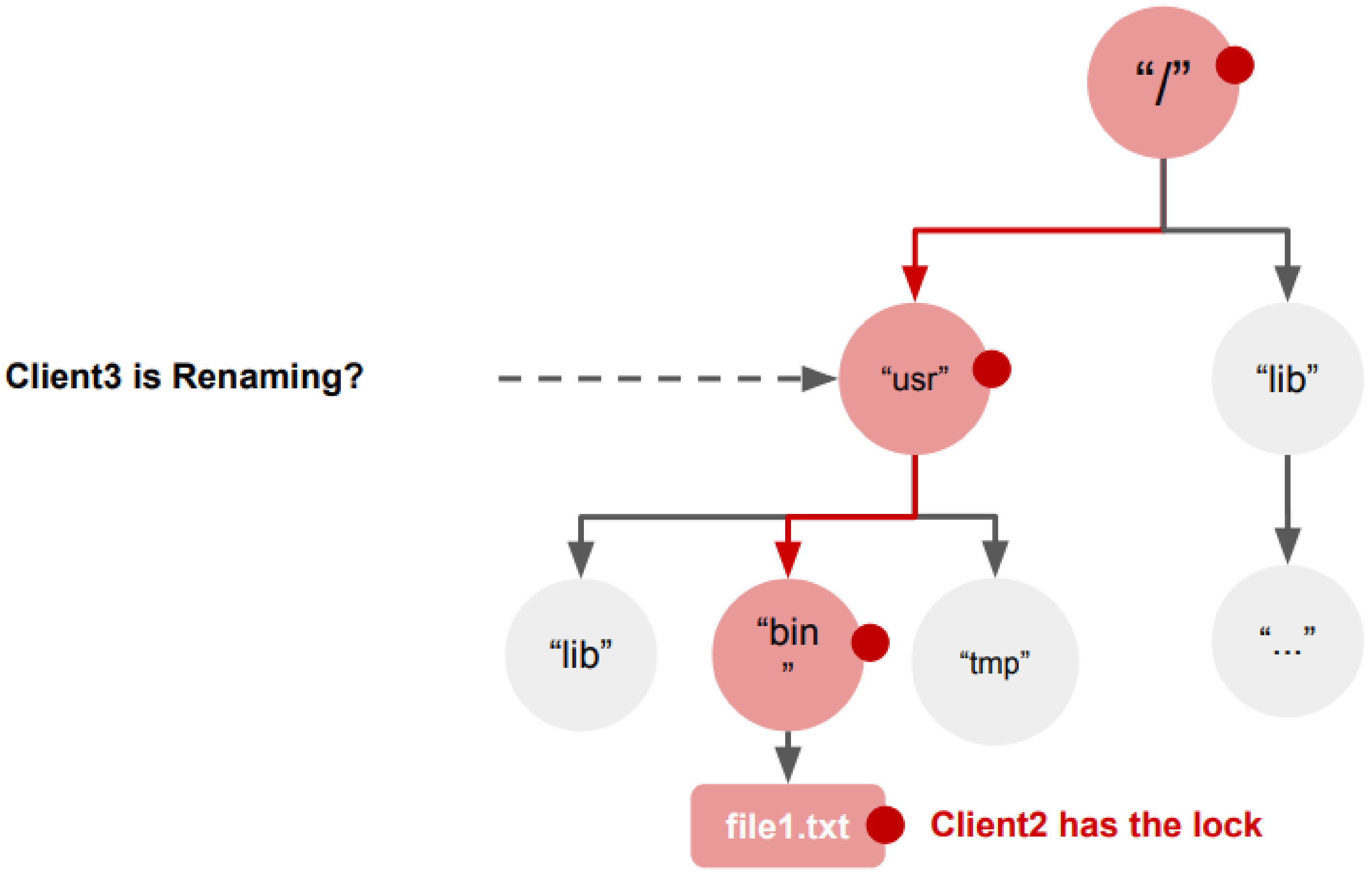
Is this good enough?



Synchronization



Synchronization



Mutual Exclusion Recap

1. **Reader:**
   - Reader is a Client who wishes to read a file at a SS
   - Reader first requests a **read/non-exclusive/shared lock**
2. **Writer:**
   - Writer is a Client who wishes to write to a file at a SS
   - Writer first requests a **write/exclusive lock**
3. **Order:**
   - Readers and writers are queued and served in the **FIFO** order

Read Locks

- Readers **request read locks from the NS** before reading files
- Readers **do not modify** contents of a file/directory
- **Multiple readers can acquire a read lock** simultaneously
- Readers **unlock files once done**

Write Locks

- Writers **request write locks from the NS** before reading/writing to files
- Writers **can modify contents** of files/directories
- Only **one writer can acquire a write lock** at a time
- Writers **unlock files once done**

Write Locks

- NS grants a write lock on a file if:
  - No reader is currently reading the file
  - No writer is currently writing to the file

- Assume a writer requests a write lock for project2.txt:

  /FileStack/users/student1/work/project2.txt

- NS applies read locks on all the directories in the path to prevent modifications

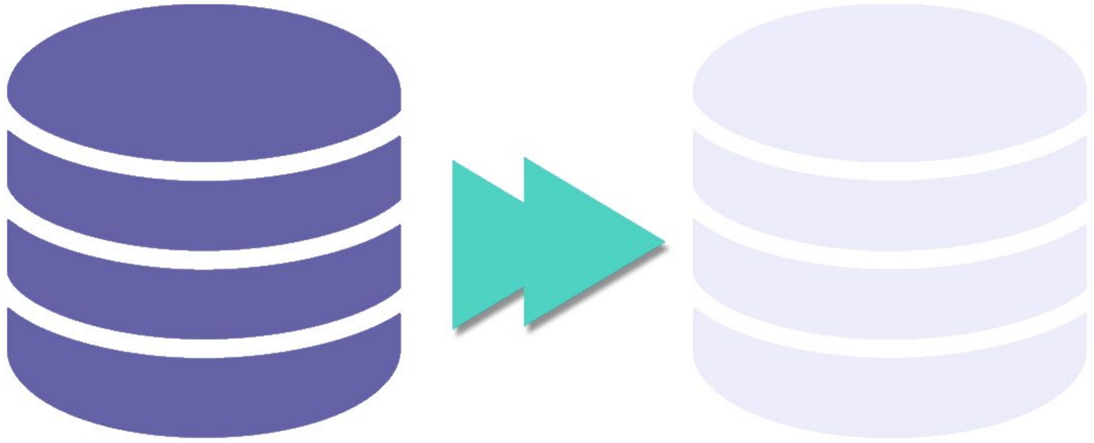
- NS then grants a write lock to the requestor of project2.txt

Service Interface

- Two new operations available to Clients:
  - LOCK(path, read/write)
  - UNLOCK(path, read/write)

Outline

- Project 2 Overview
- Synchronization of File Readers & Writers
- Dynamic Replication of Files
- Implementation Tips



Project 2 Objectives

1. Logical Synchronization of Readers and Writers

2. **Devise and apply a replication algorithm that:**
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

Project 2 Objectives

1. **Devise and apply a synchronization algorithm that:**
   - achieves *correctness* while sharing files
   - and ensures *fairness* to clients.

2. Dynamic Replication of Files

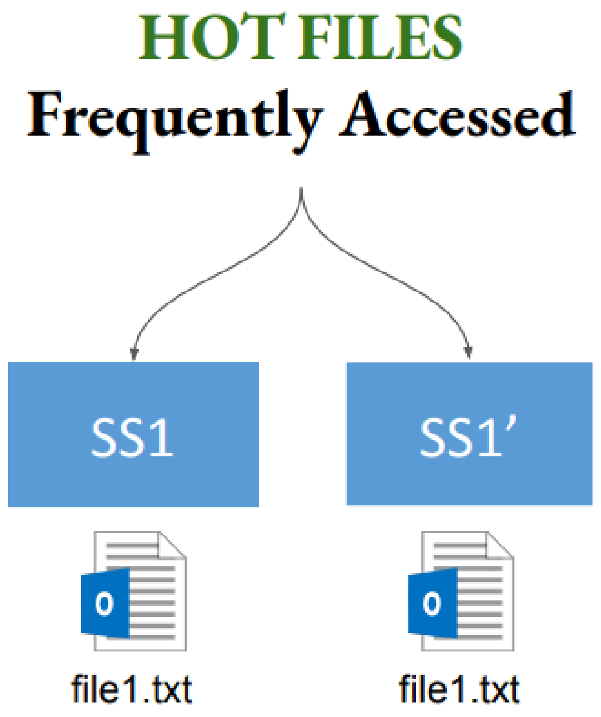
Replication - Questions

- Which files to replicate?
- How many Replicas?
- When to replicate?
- How to Replicate?
- How to achieve consistency of replicas?

Which files to replicate?

- In our DFS, we'll have two kinds of Files:
  - Files that have a lot of requests
    - These are denoted as "***hot-files***"
  - Files that are very rarely accessed
    - These are denoted as "***cold-files***"
- To achieve load-balancing, we can replicate "***hot-files***" onto other SSs

How many replicas?



Fine-grained Approach:

num_replicas = ALPHA * num_requesters

num_replicas = min (ALPHA * num_requesters , REPLICA_UPPER_BOUND)

Coarse-grained Approach:

num_requesters_coarse = {N | N >= num_requesters & a m ul ti pl e o f 20}

num_replicas = min (ALPHA * num_requesters_coarse , REPLICA_UPPER_BOUND)

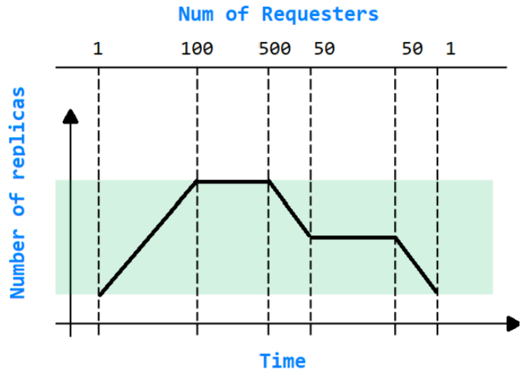


Figure 2: Linear Replication Policy with an upper-bound

When to Replicate?

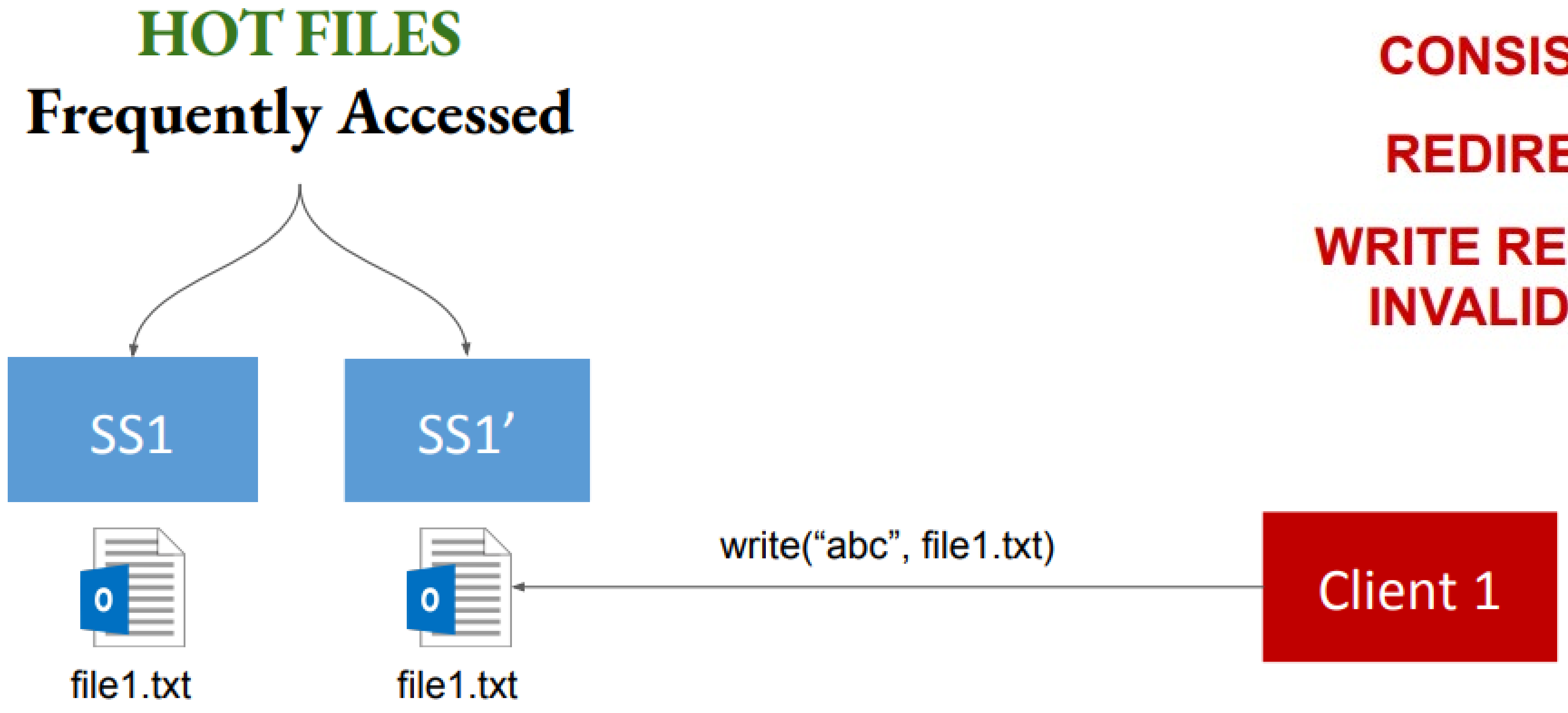
- NS would want to store ***num_requests*** as file metadata
- However, how can we determine and in turn update ***num_requests*** over time?
  - We know that Clients invoke read operations on storage servers
  - Therefore, every "read" lock request from a client is deemed as a read operation
  - Afterward, NS increments ***num_requests***
  - Reavaluate **num_replicas**

How can we Replicate?

- NS first elects one or many SSs to store the replicas
- NS commands each elected SS to copy the file from the original SS
- Therefore, the metadata of a file now includes ***a set of SSs*** instead of a single SS

Replication Challenges

What are the challenges?



CONSISTENCY

REDIRECTION

WRITE REQUESTS INVALIDATION

How to Update Replicas?

- **When a Client requests a write lock on a file:**
  - It causes the NS to *invalidate* all the replicas except the locked one

- Invalidation is achieved by **commanding those SSs hosting replicas to delete the file**

- When the Client unlocks the file, the NS commands SSs to copy the modified file

The Command Interface

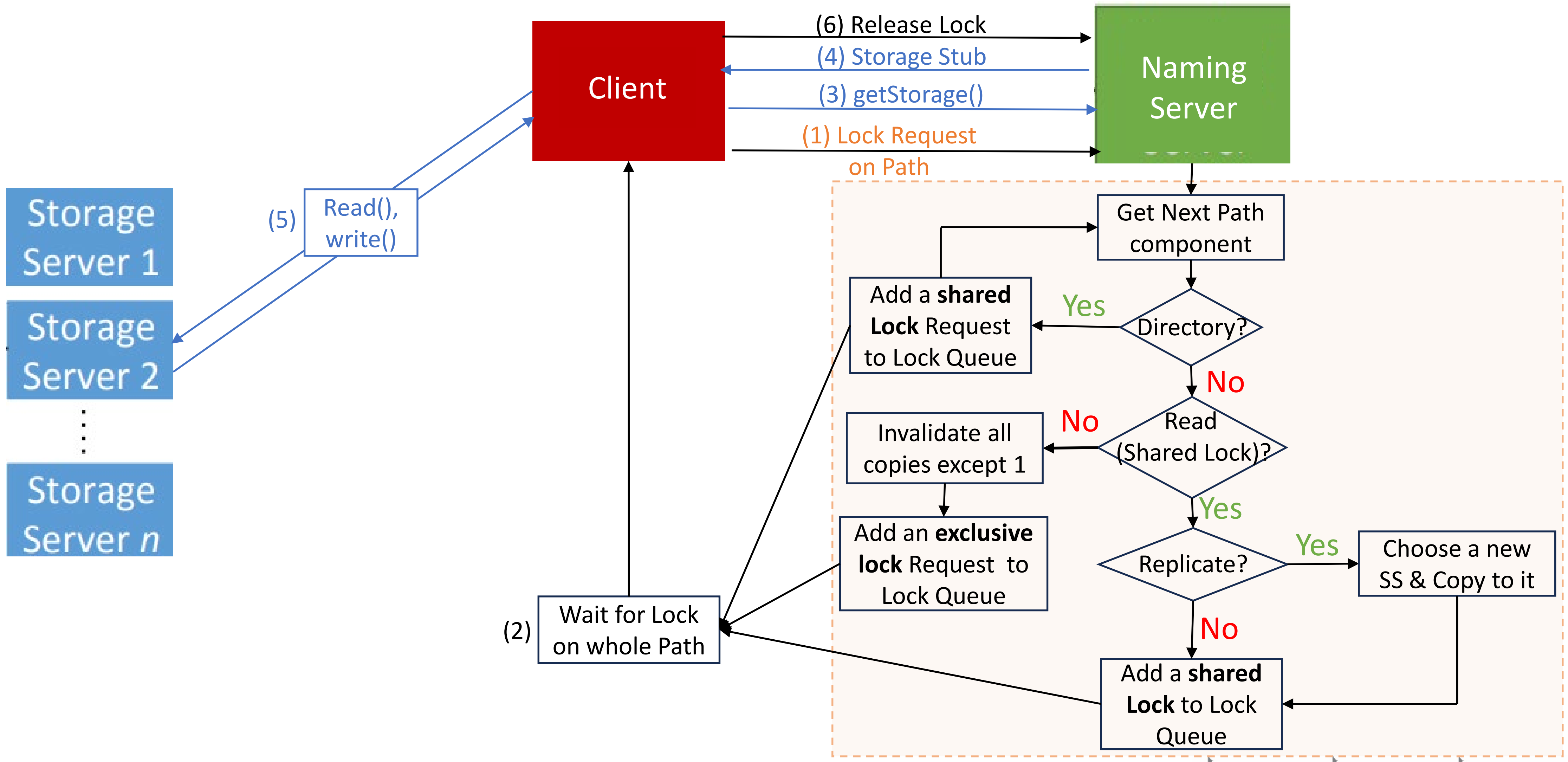
- **One new operation** available to the NS:
    - COPY (path P, StorageStub S)

      copies file with path P from StorageStub S

Outline

- Project 2 Overview
- Synchronization of File Readers & Writers
- Dynamic Replication of Files
- Implementation Tips

Process Flow w/ Locking & Replication



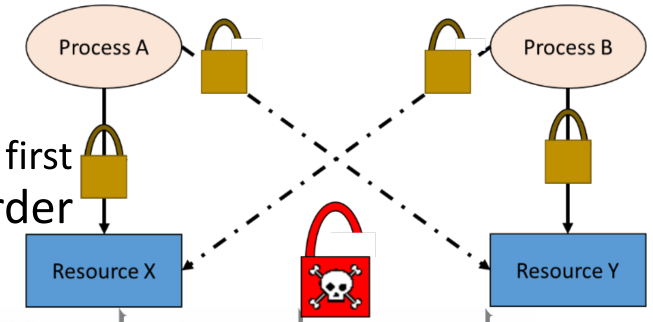
Implementation Tips: Read/Write Locks

- Tracking Lock requests on Nodes
  - Each tree Node at the Naming Server should have an Object to manage locks on that node
  - A queue of read/write lock requests should be maintained

- Granting Locks
  - `lock(Path path, boolean exclusive)` method: defined in the service interface & should be implemented in the Naming Server

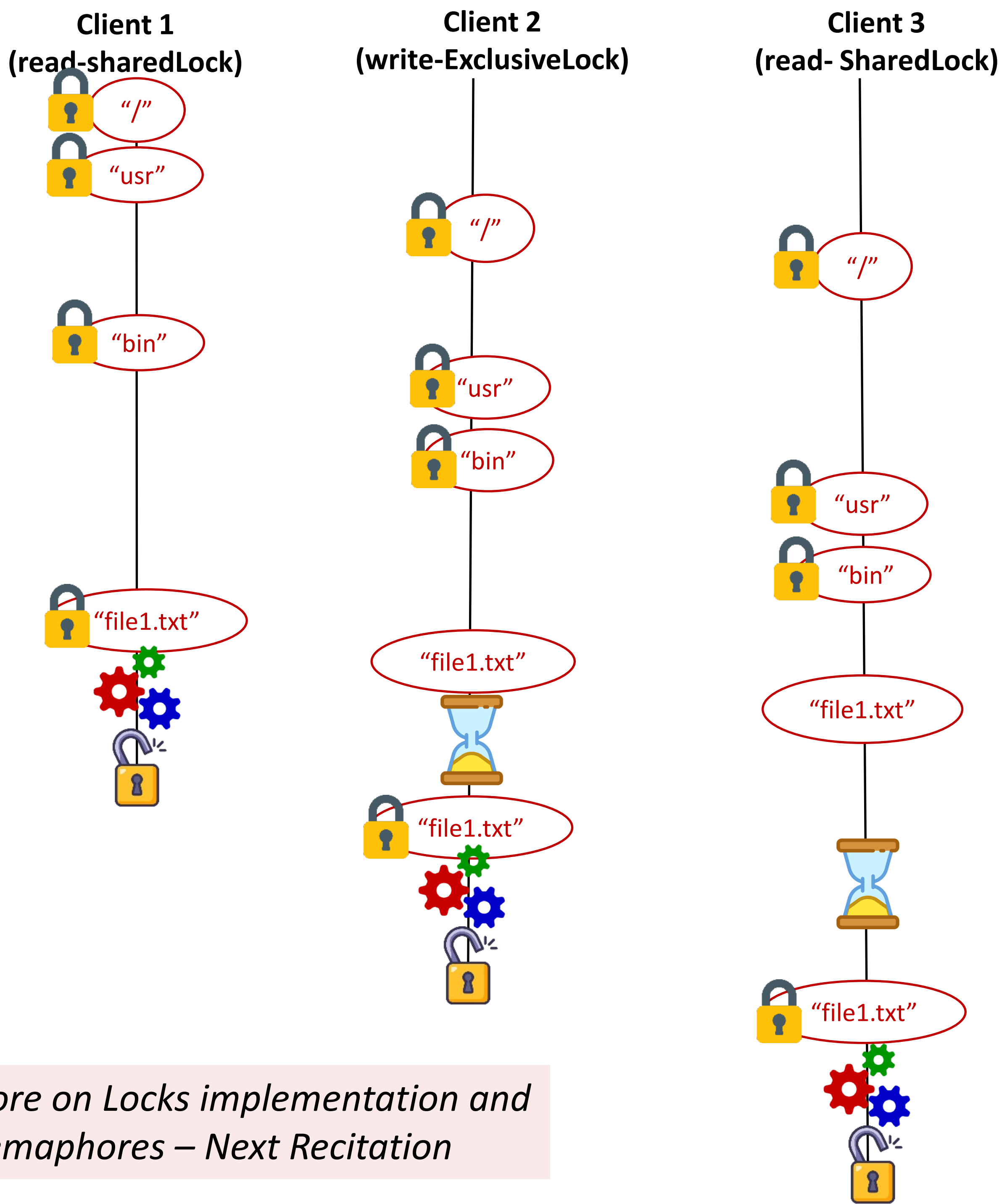
- Releasing Locks
  - `unlock(Path path, boolean exclusive)` method: defined in the service interface & should be implemented in the Naming Server

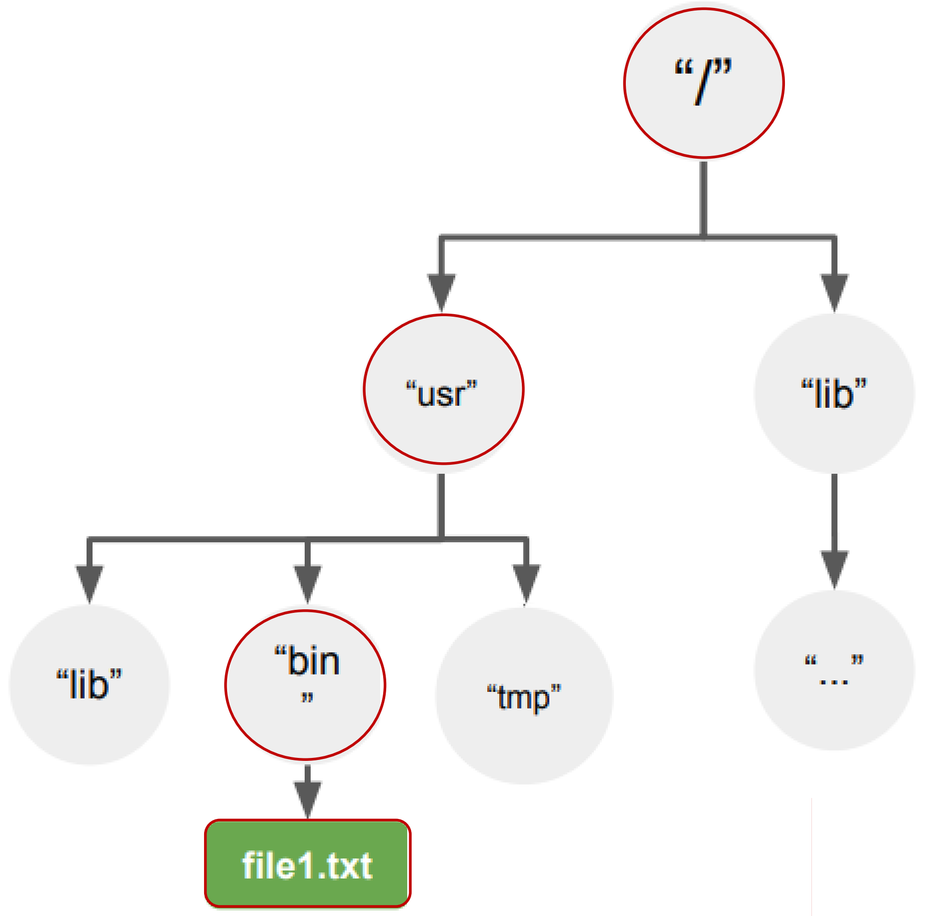
- Avoiding Deadlocks by adding Ranks to Paths
  - Path implements Comparable:
    - - CompareTo() method: compares two paths to determine which one to be locked first
  - Paths that need to be locked simultaneously are locked in an increasing order



Carnegie Mellon University Qatar

Locking/Unlocking a Path





More on Locks implementation and Semaphores – Next Recitation

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Implementation Tips: Replication

- Replication Decision: deciding whether a file should be replicated
  - Track number of reads to a file (update tree metadata)
- Create a Formula to define the number of replicas; given the number of reads
- Replication Process
  - Selecting a new host storage server
  - Copying the file to a new server
    - `copy(Path file, Storage server)` method: defined in the Command interface and implemented in the Storage Server
  - Updating tree metadata (list of storage servers hosting the file)
- Ensuring Consistency
  - Given a shared Lock request for read:
    - Assess number of reads and replicate if needed
  - Given an exclusive lock request for Write
    - Invalidate all copies except 1

Code Overview

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar