15-440
Distributed Systems
Recitation 6

Ammar Karkour

Adopted from: Previous TAs

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

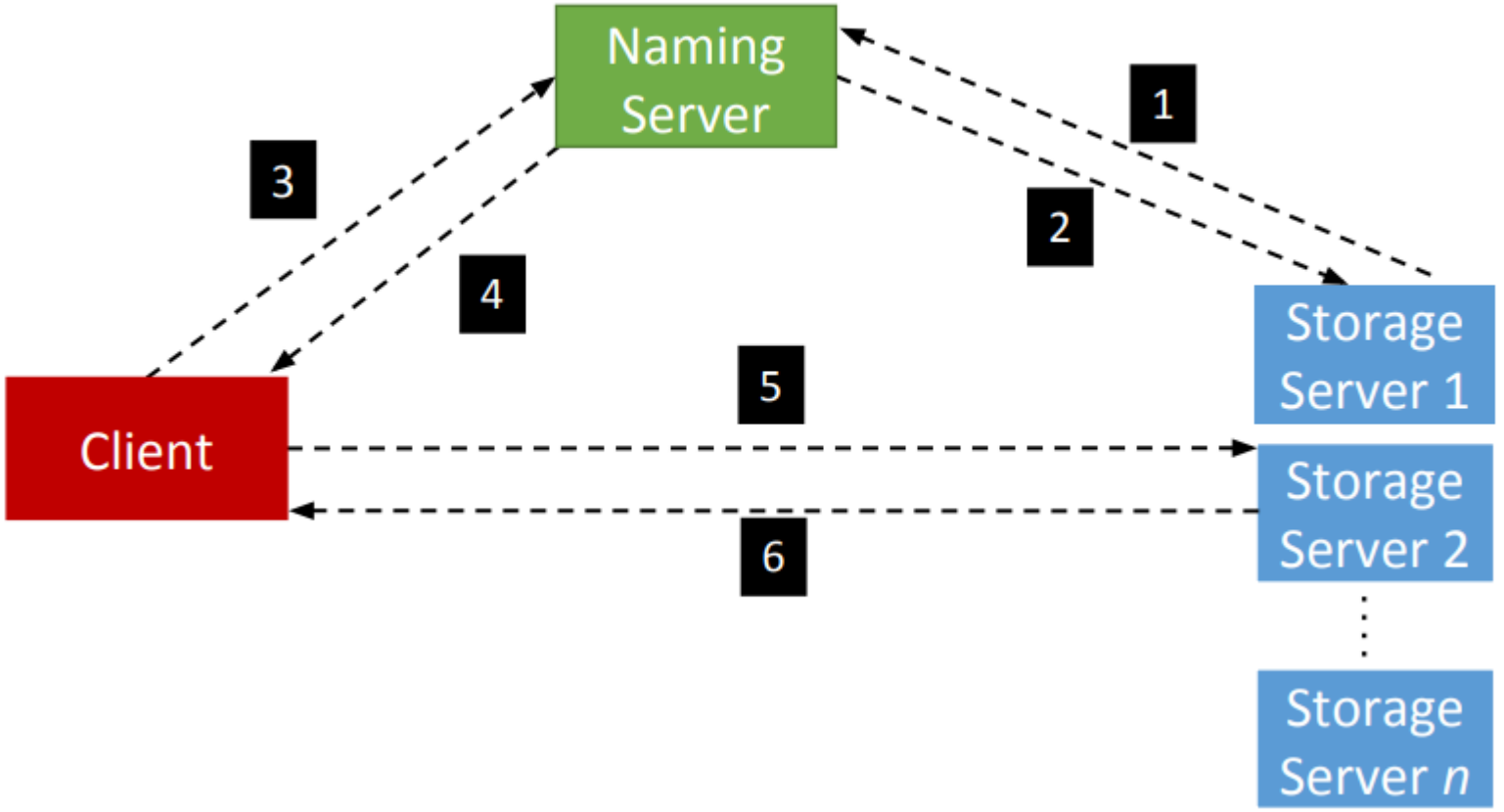
Logistics

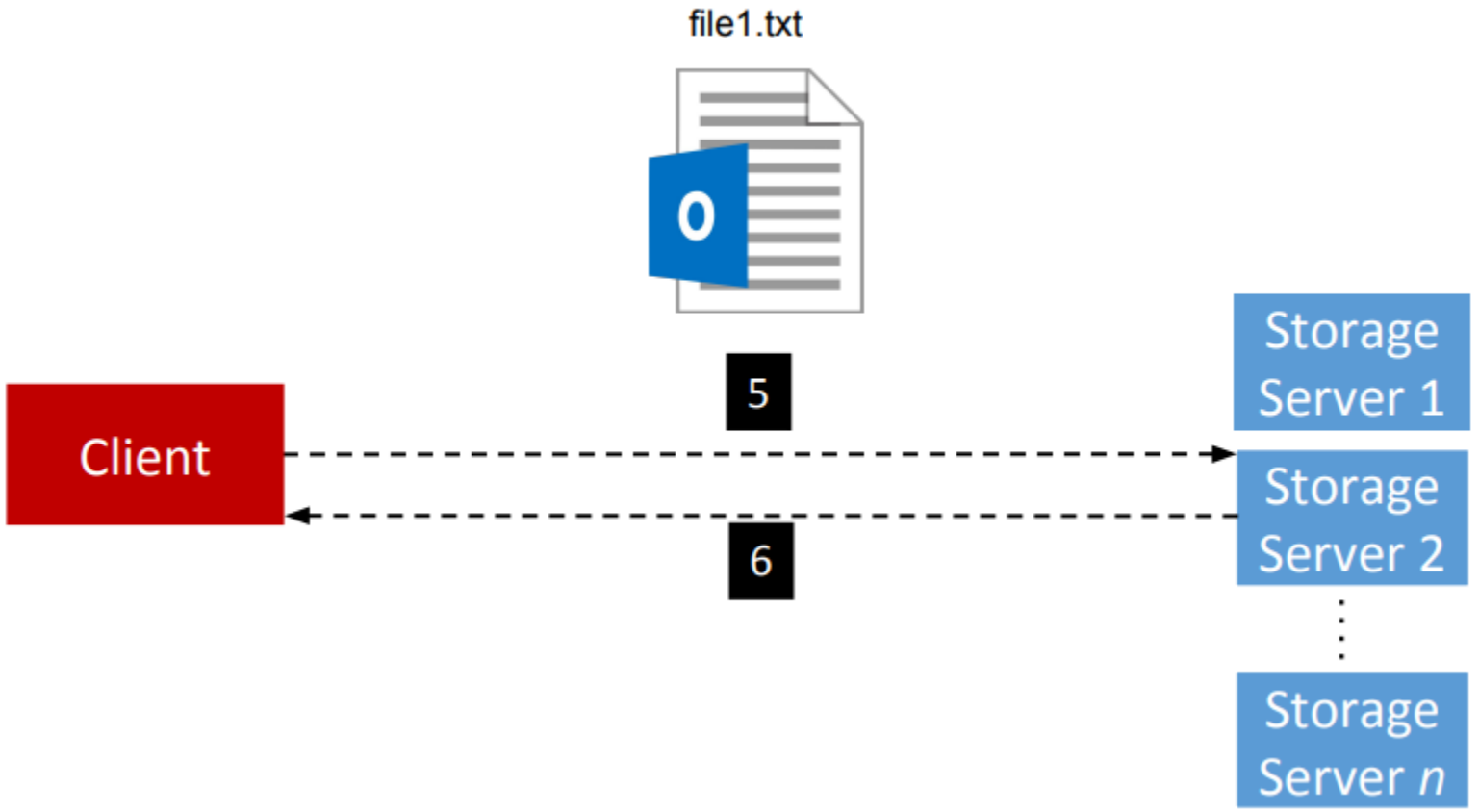
- P1 Done!
- P2 Out (due October 4)
- Midterm (September 25)
- PS3 (due September 29)

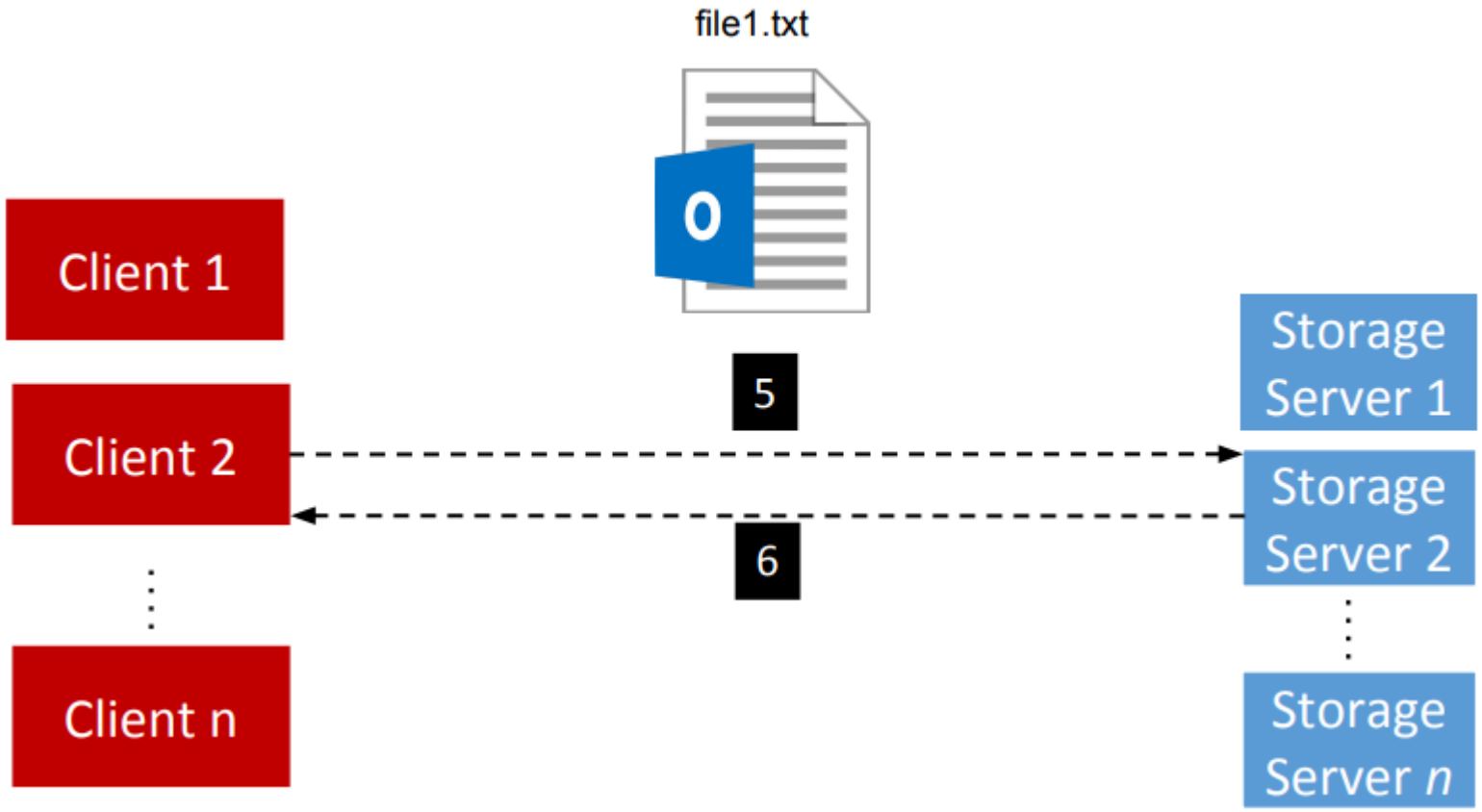
Project 2

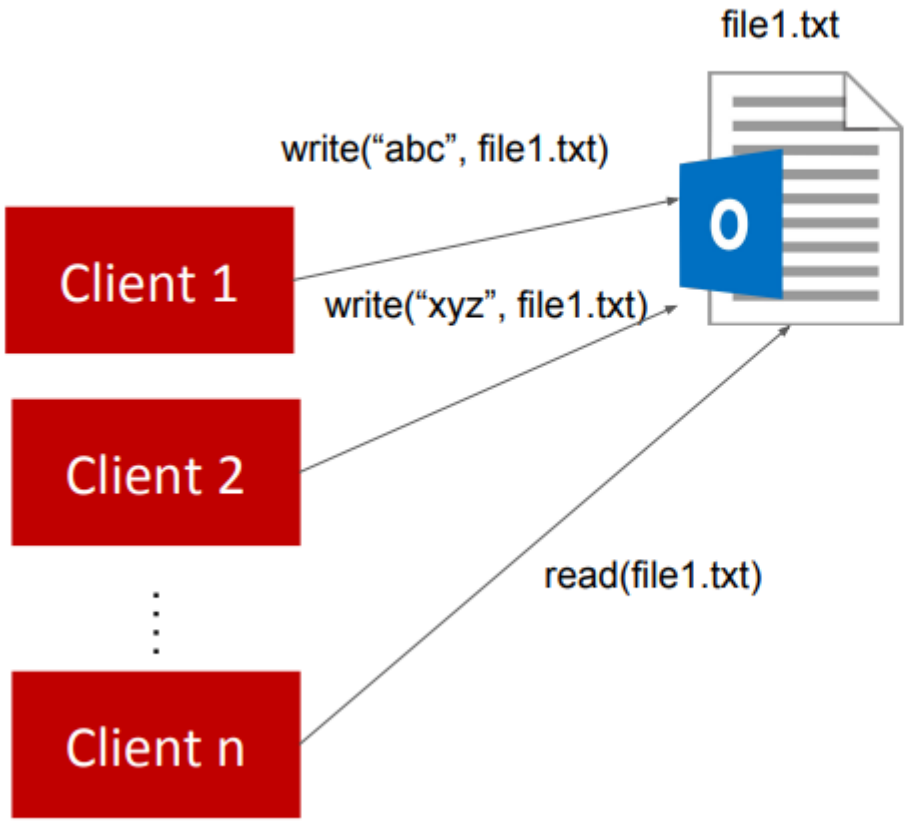
- Involves *building on your Project 1 Distributed File System* (**DFS**): *FileStack*
- P2_StarterCode: Copy files into your P1 folder
- Release Date: **September 15th**
- Due date: **October 4th**

FileStack Architecture



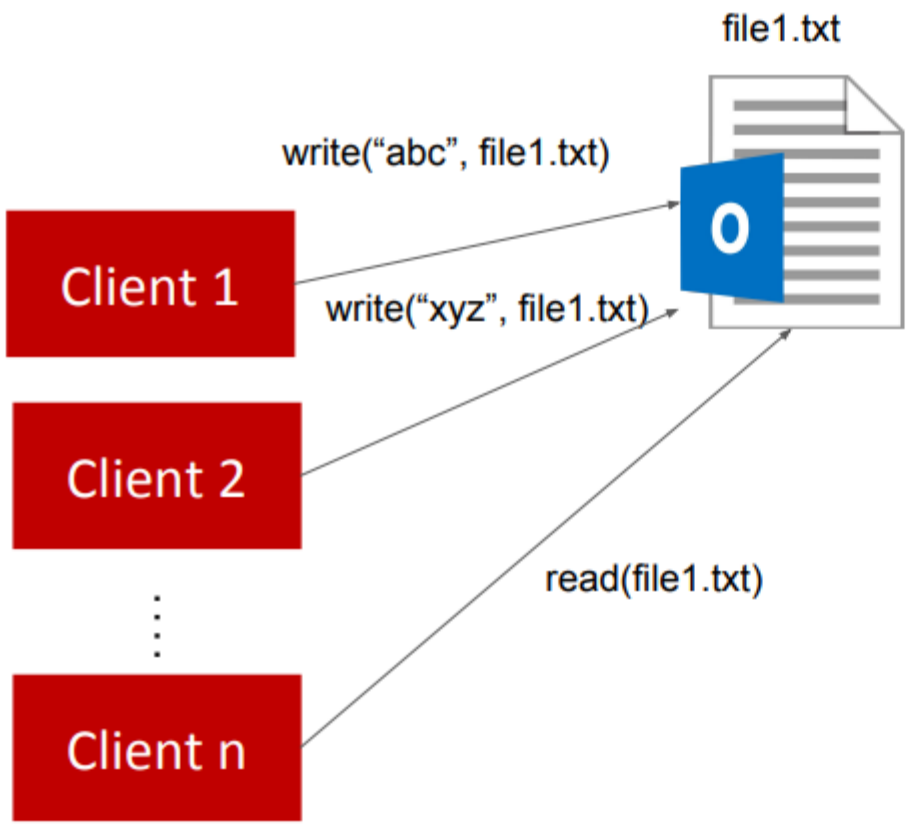

file1.txt
Client
5
6
Storage Server 1
Storage Server 2
Storage Server n


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



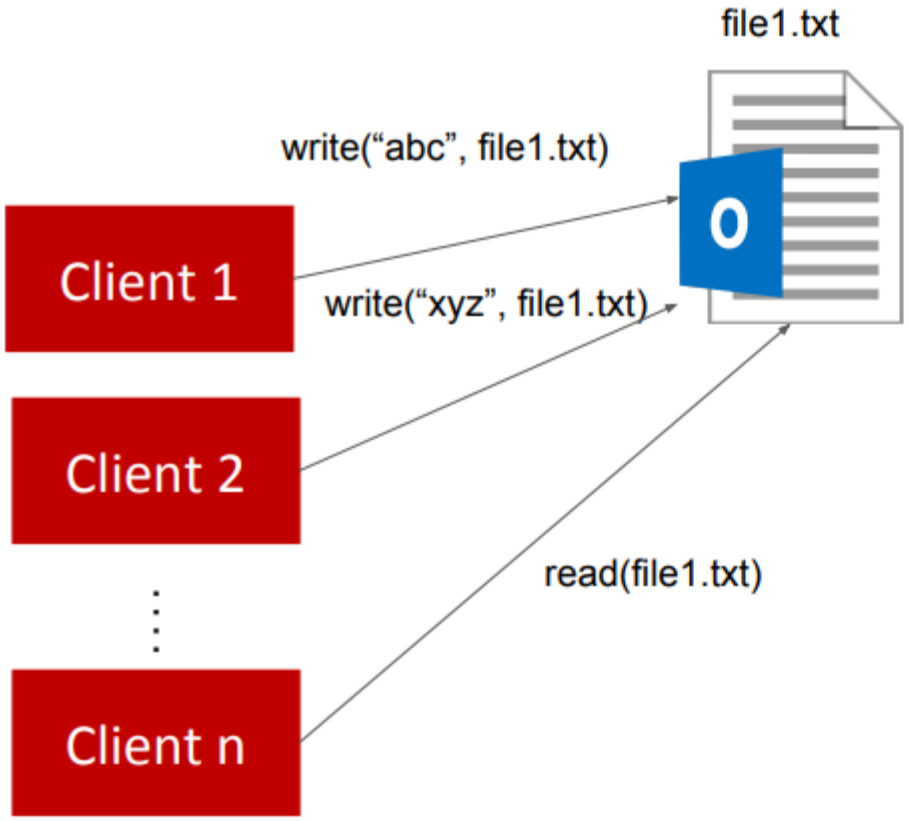
What might go wrong?

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar



What might go wrong?

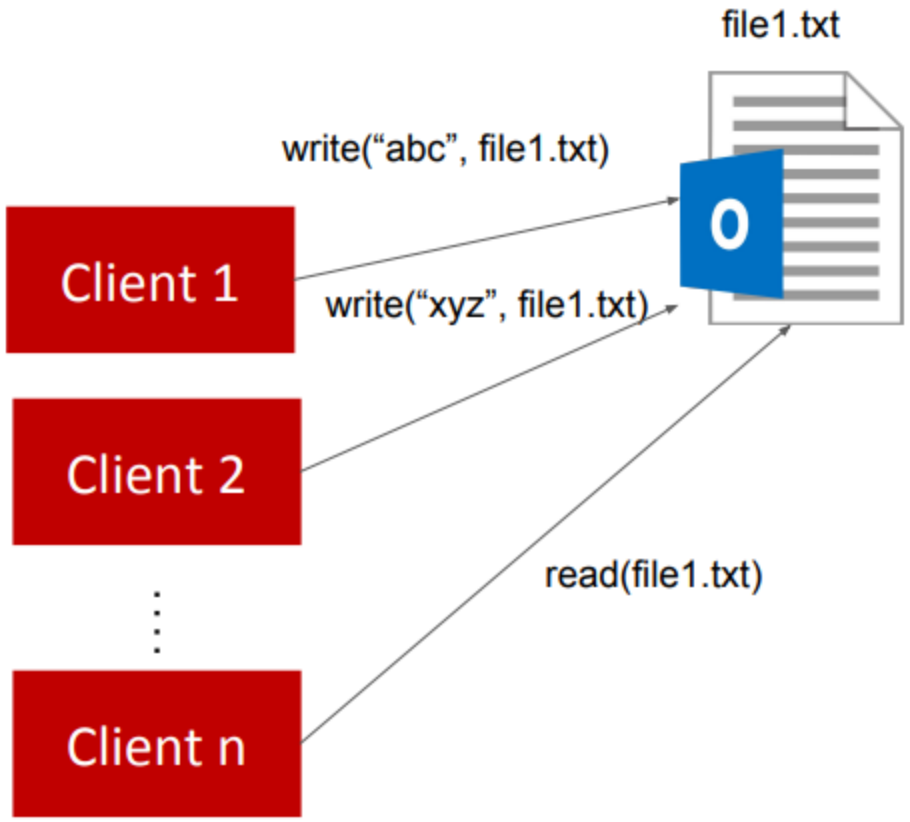
- Synchronization



What might go wrong?

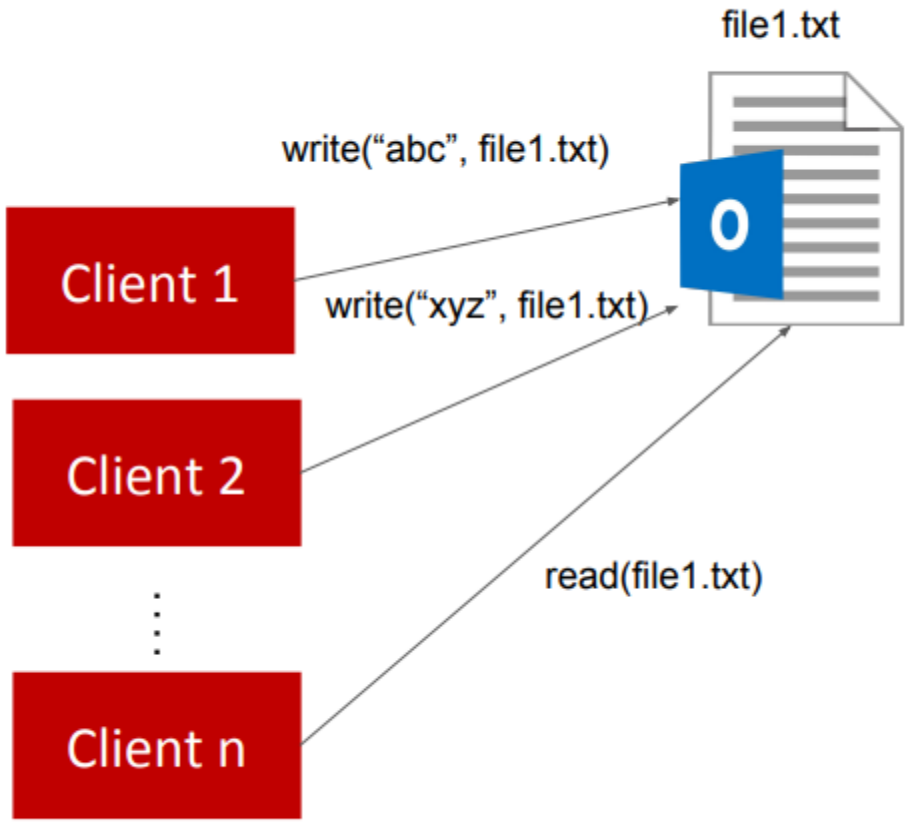
- Synchronization

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

جامعة كارنيجي ميلون في قطر
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.

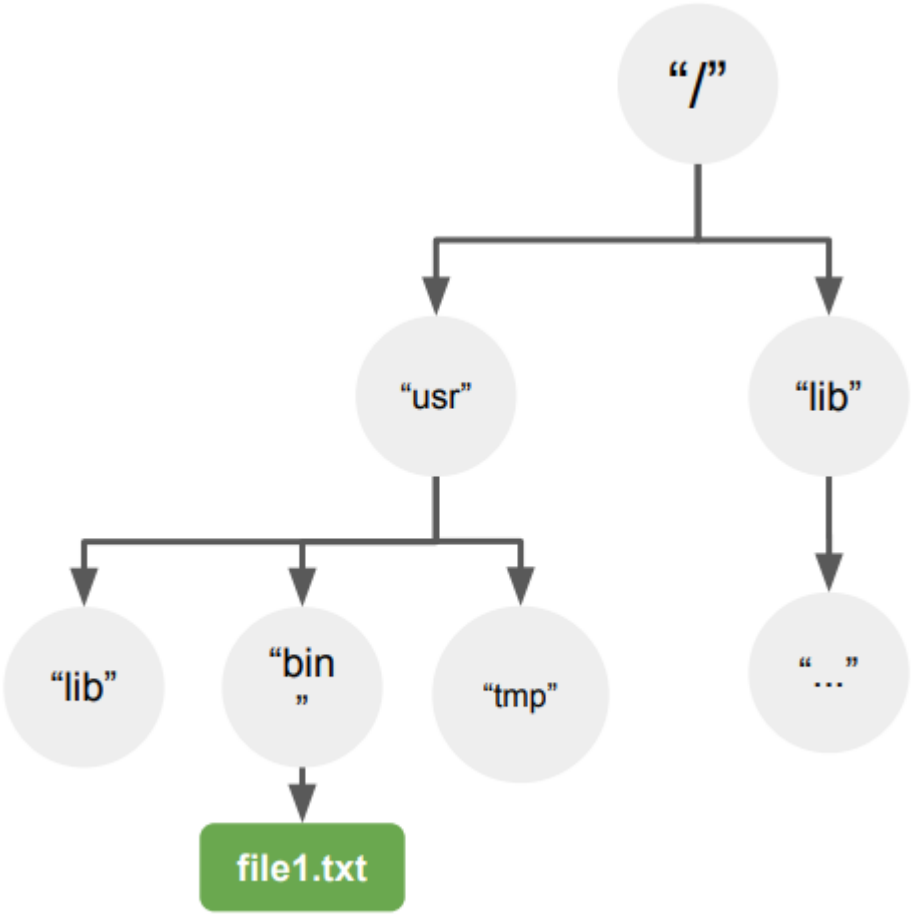
2. **Devise and apply a replication algorithm that:**
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

Project 2 Objectives

1. Logical Synchronization of Readers and Writers

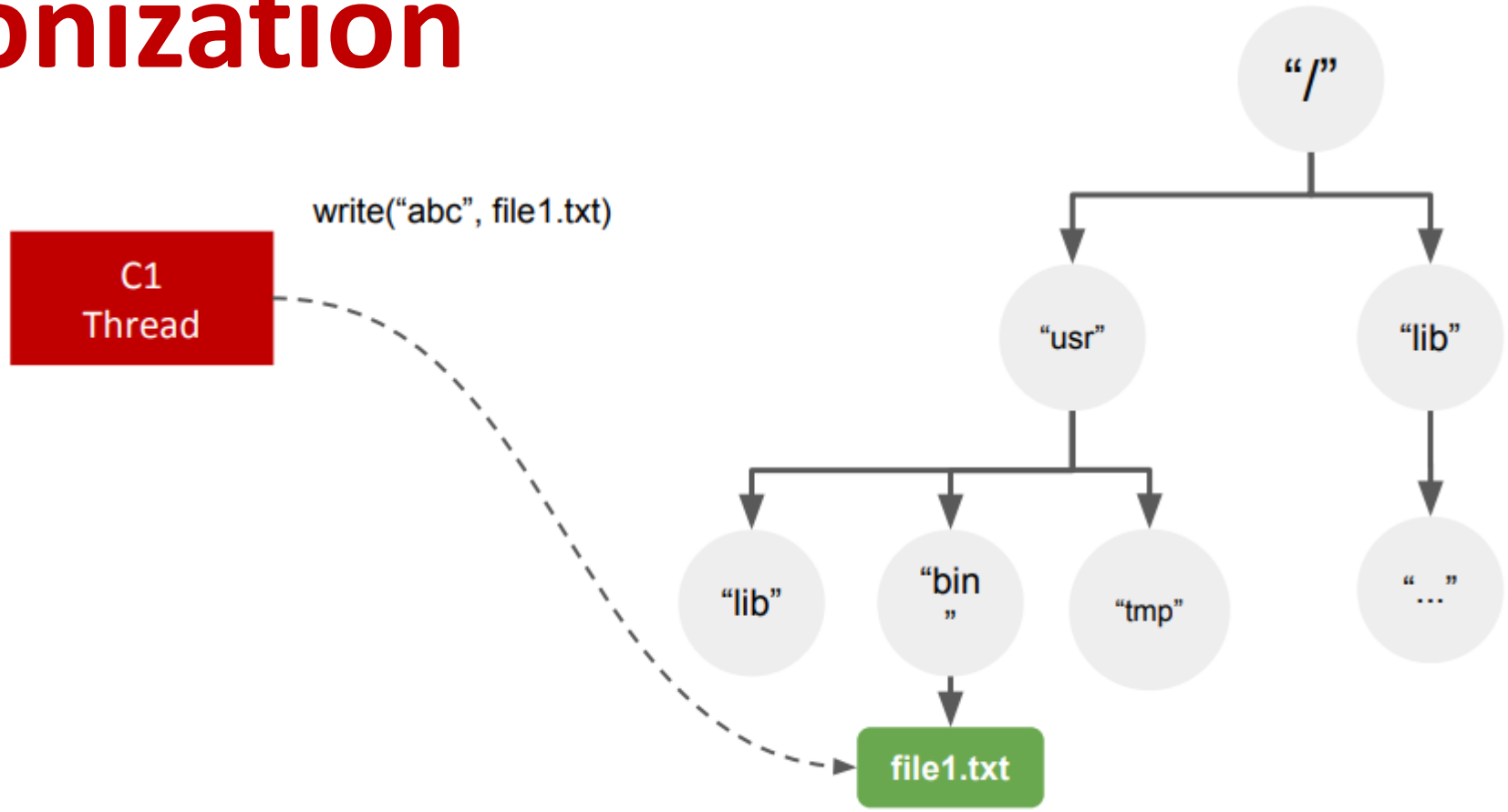
2. **Devise and apply a replication algorithm that:**
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

Synchronization



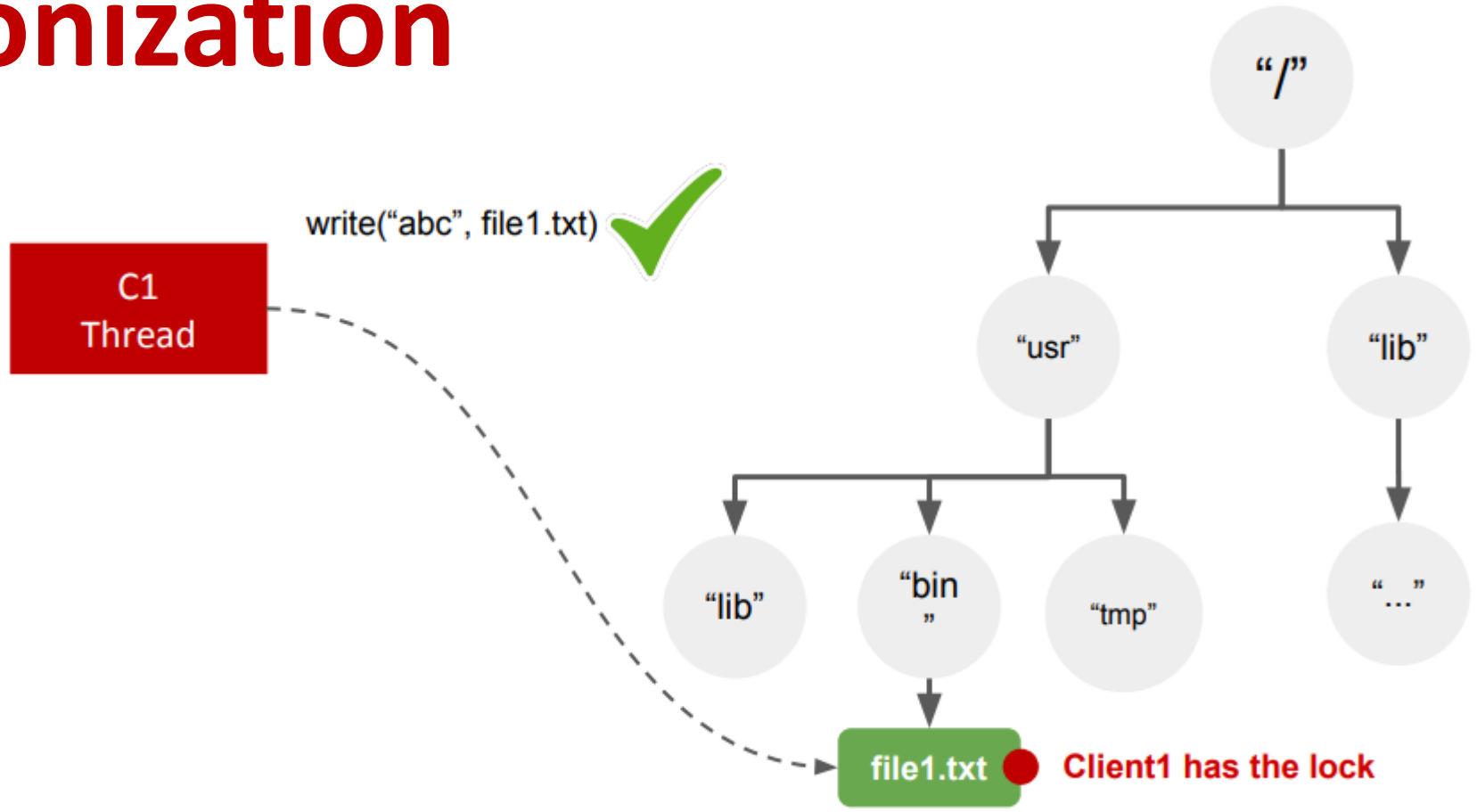
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Synchronization

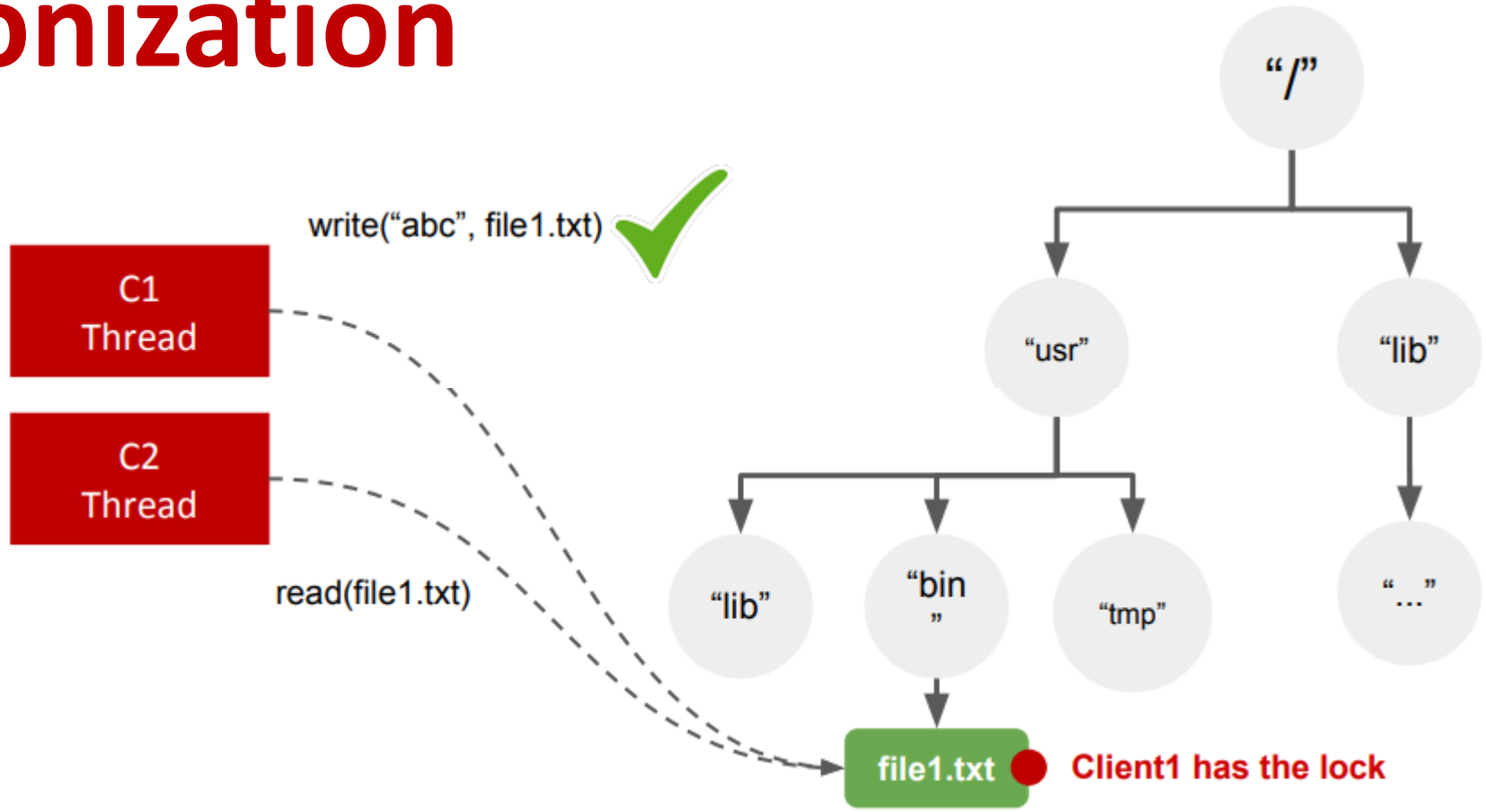


جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

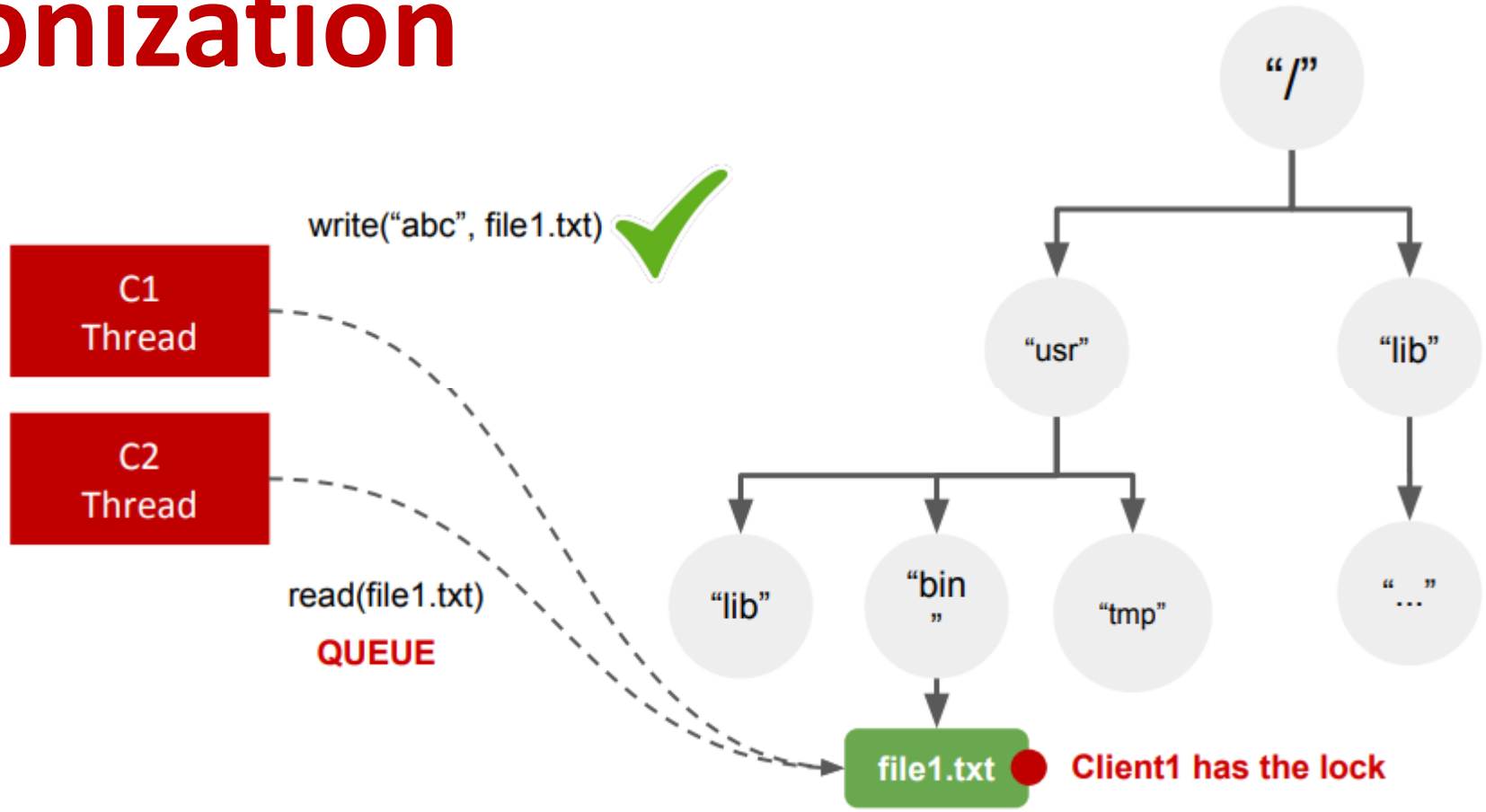
Synchronization



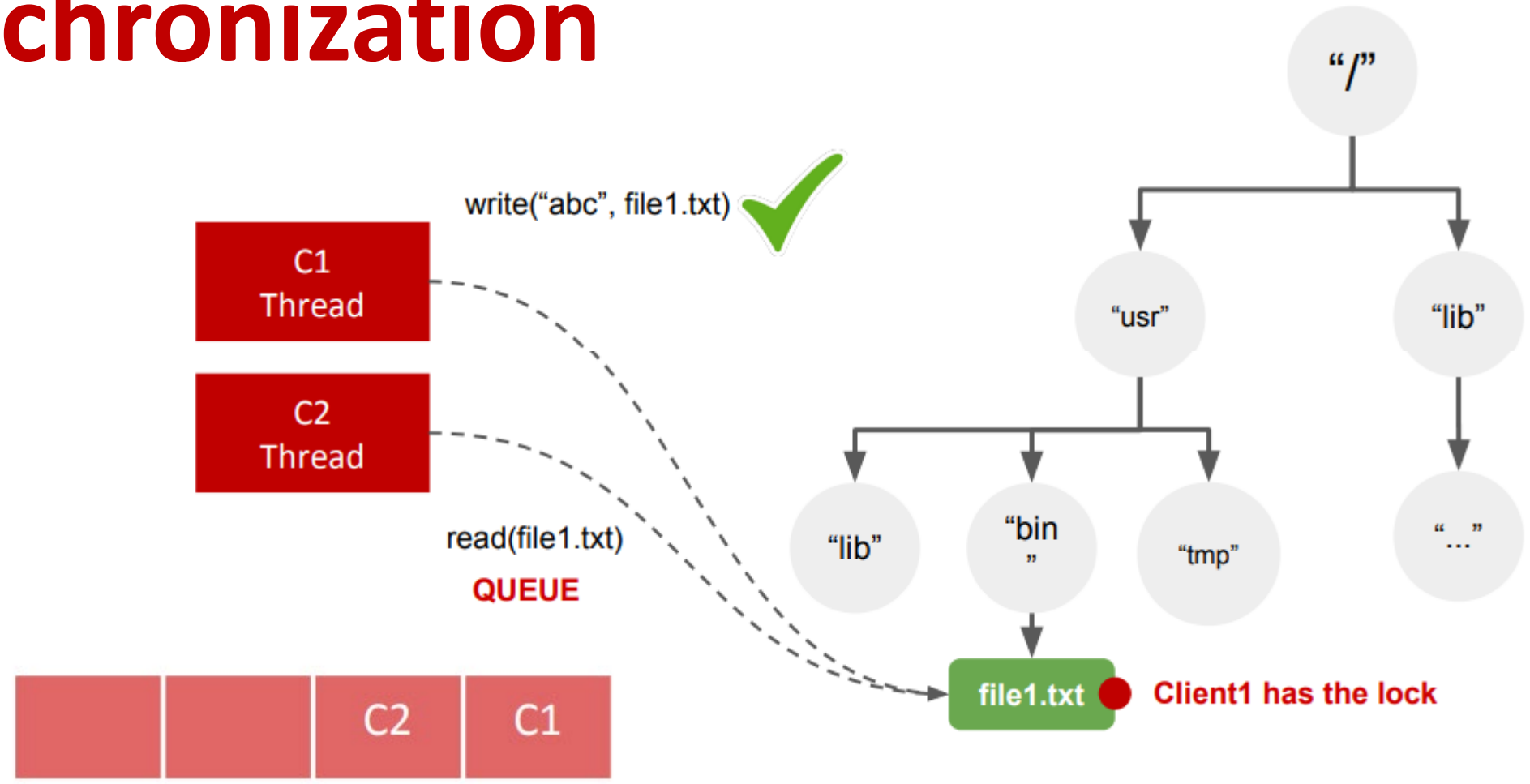
Synchronization



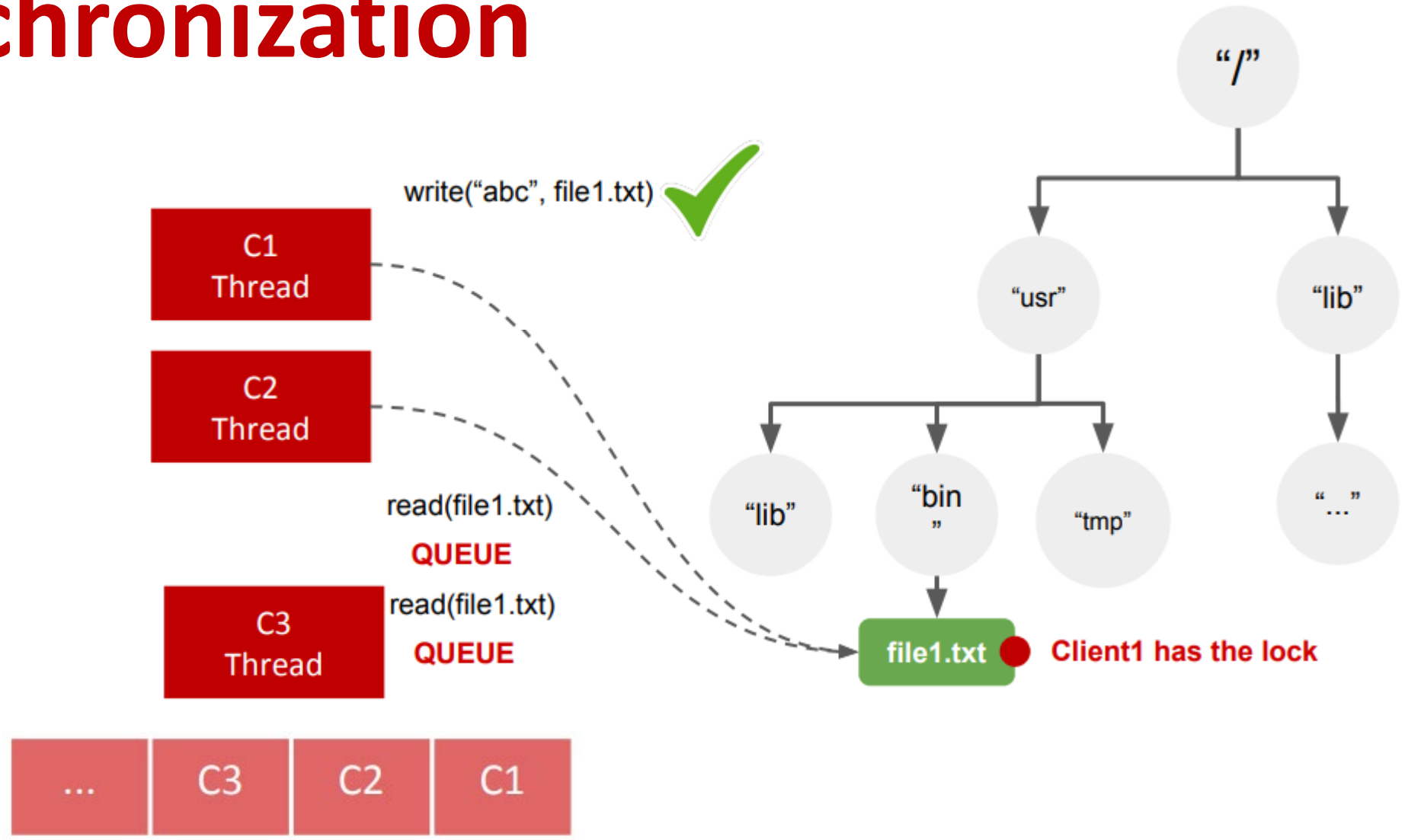
Synchronization



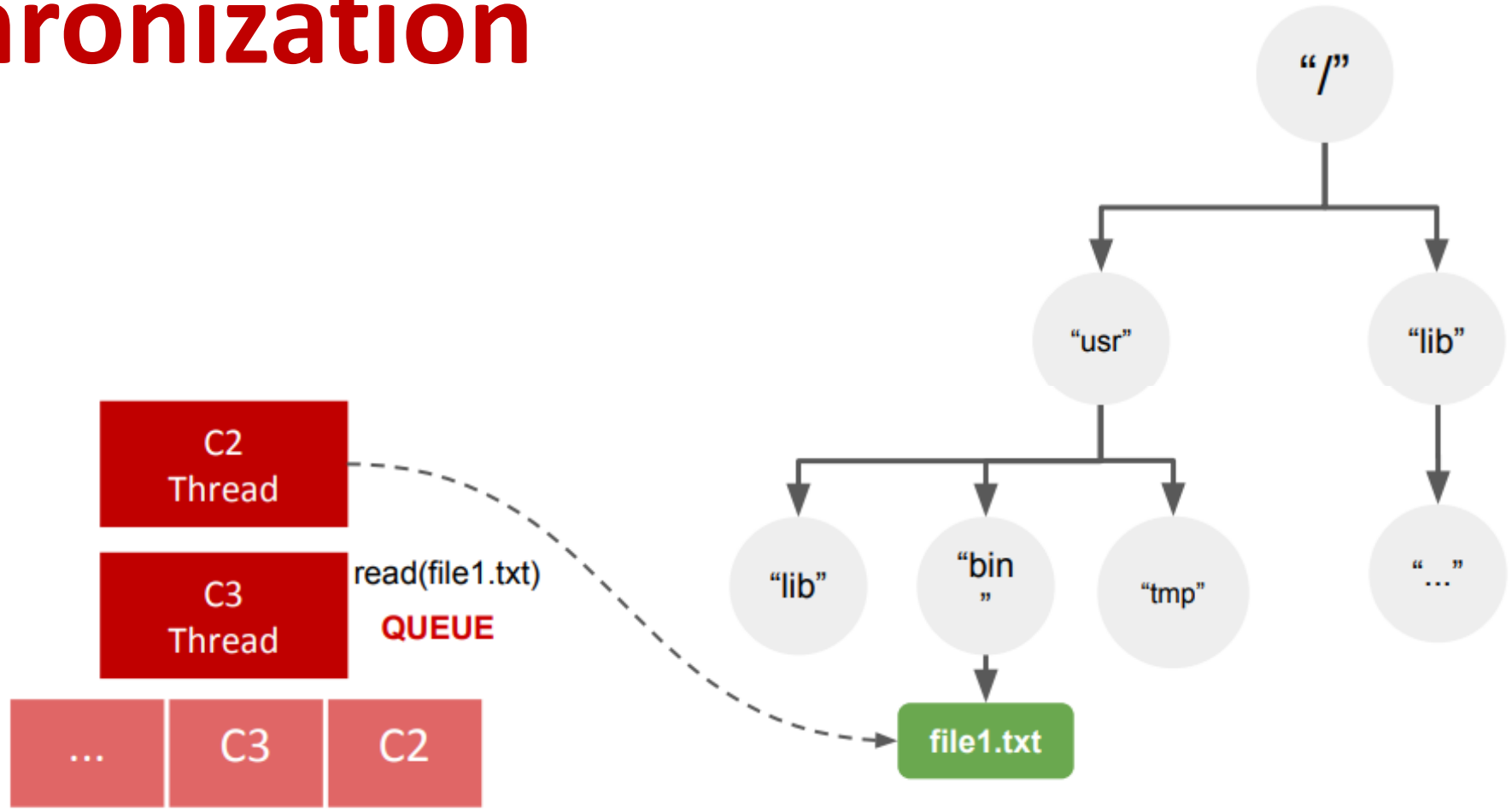
Synchronization



Synchronization



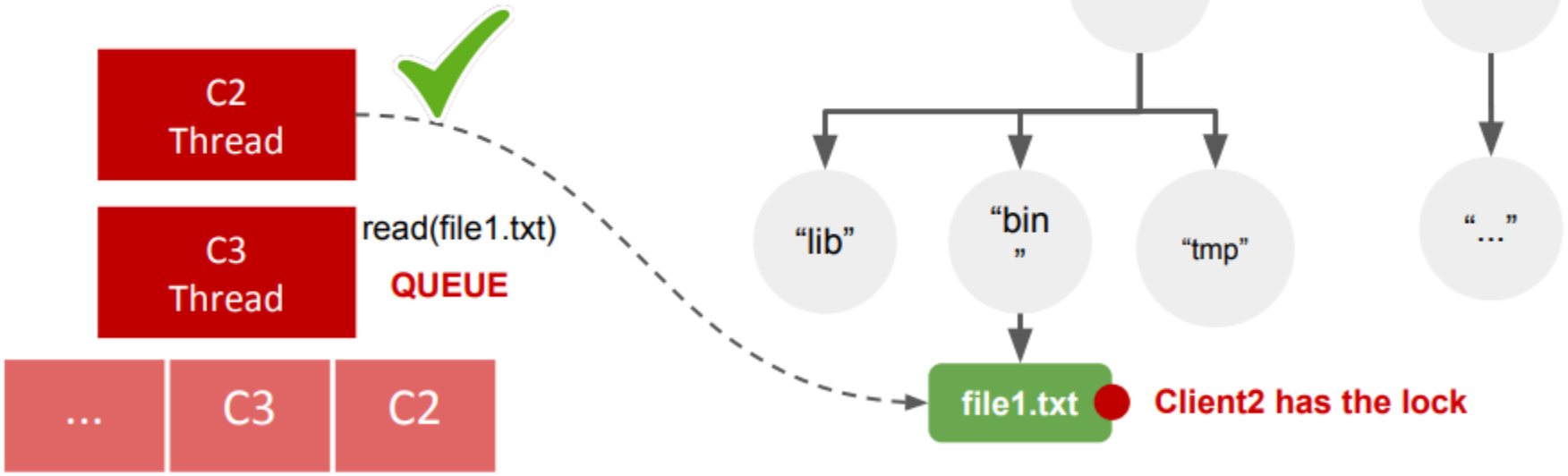
Synchronization



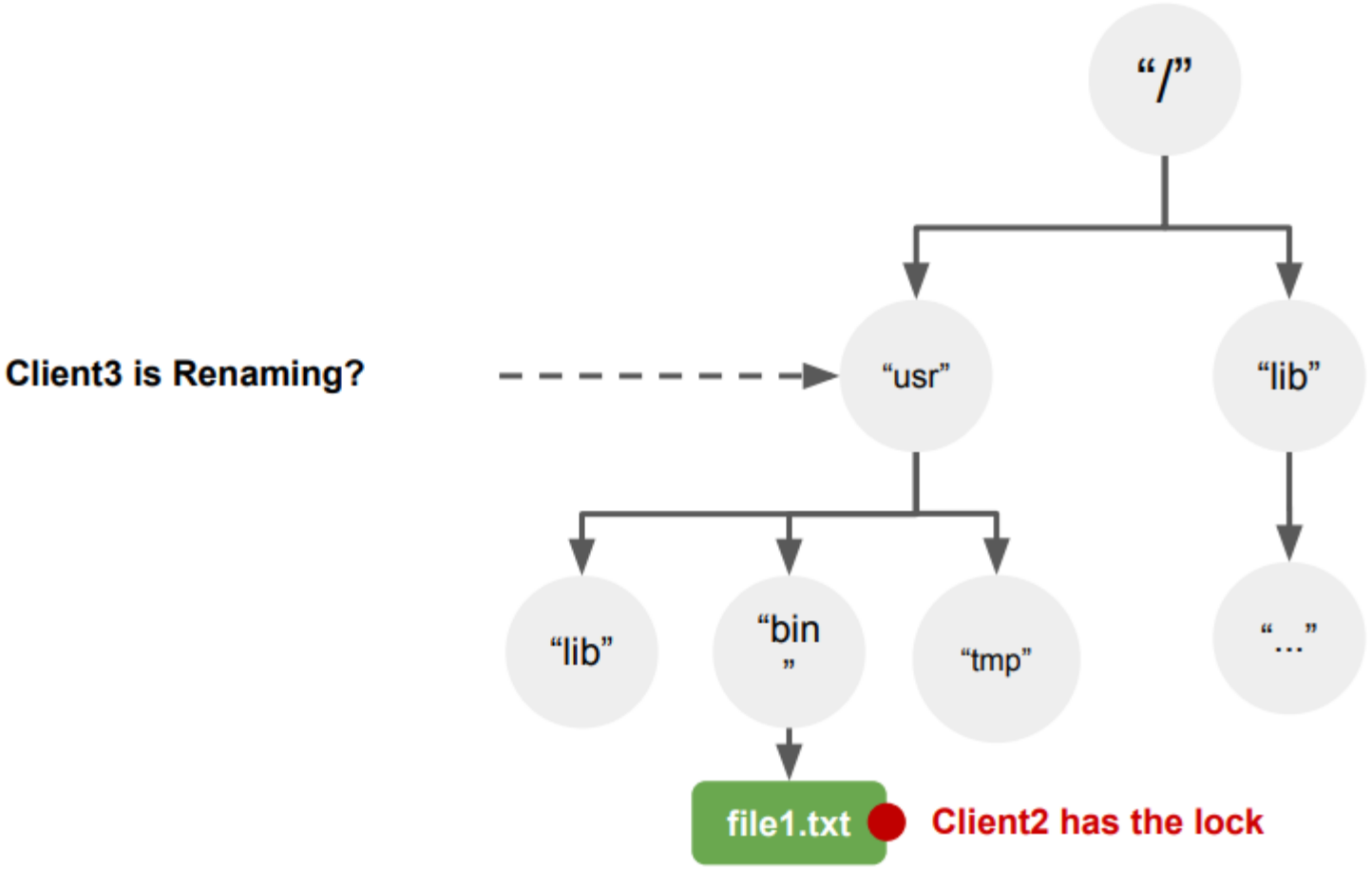
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Synchronization

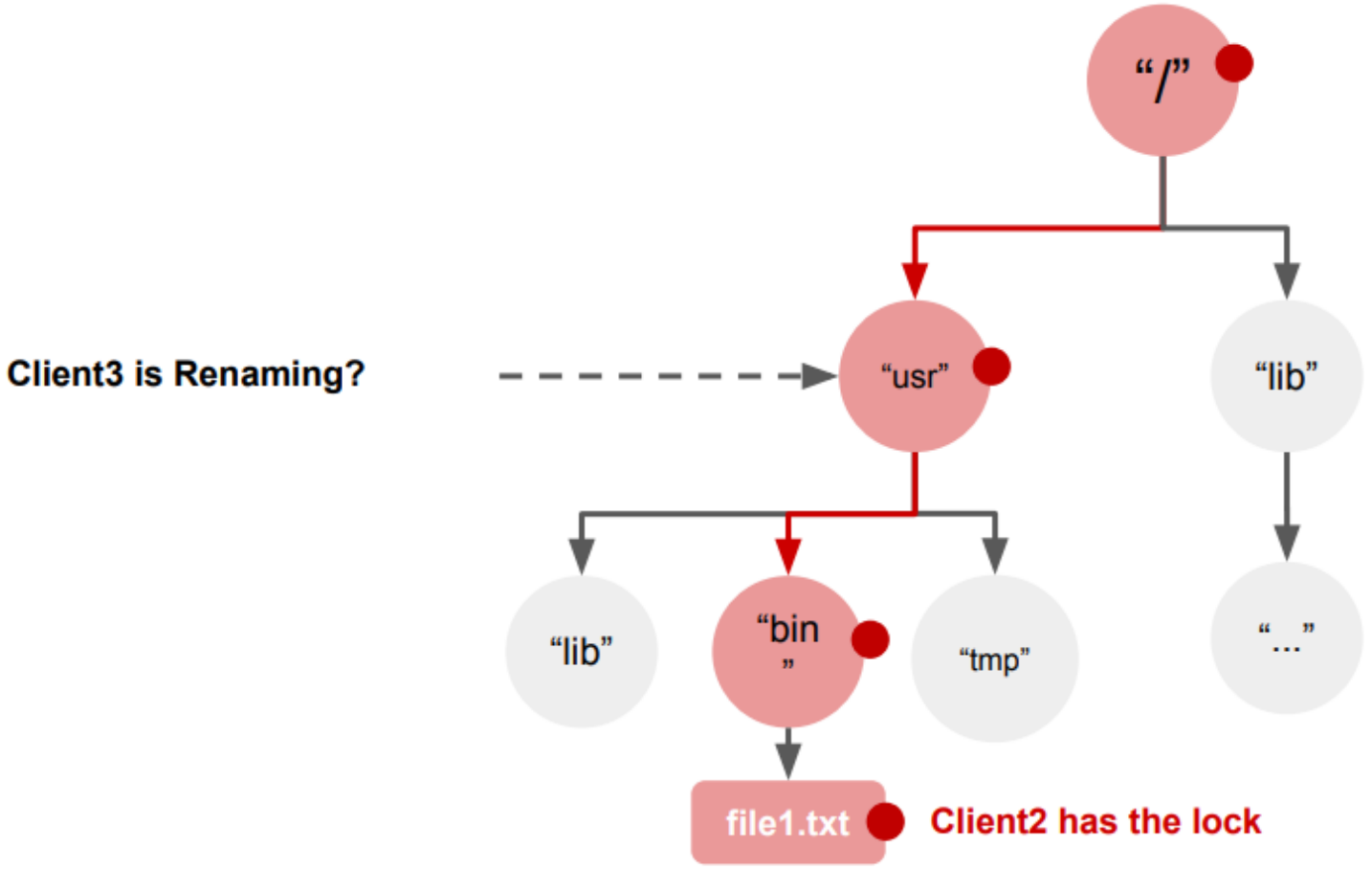
Is this good enough?



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 the NS for read locks** 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 the NS for write locks** 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

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Service Interface

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

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.

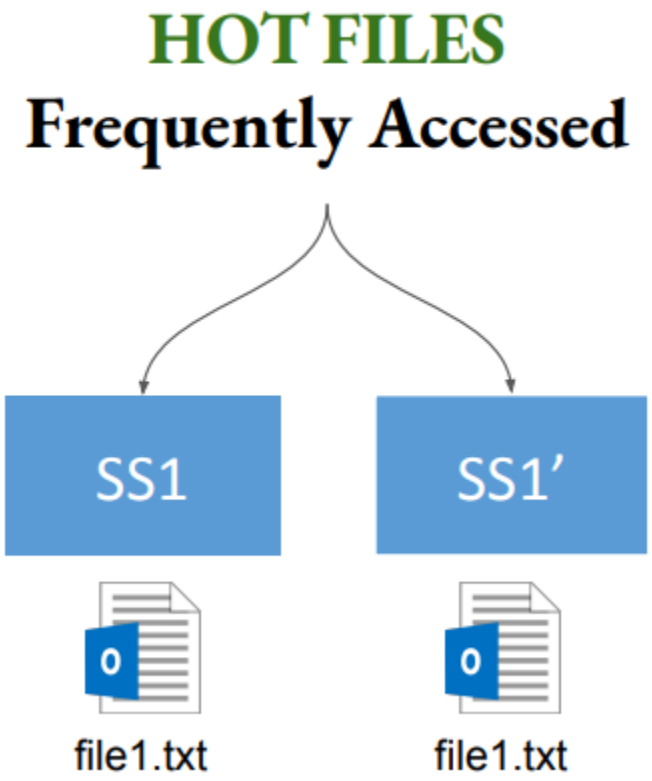
2. Dynamic Replication of Files

Why 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

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

How many replicas?



num_replicas = ALPHA * num_requesters

num_replicas = min (ALPHA * num_requesters , REPLICA_UPPER_BOUND)

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)

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**

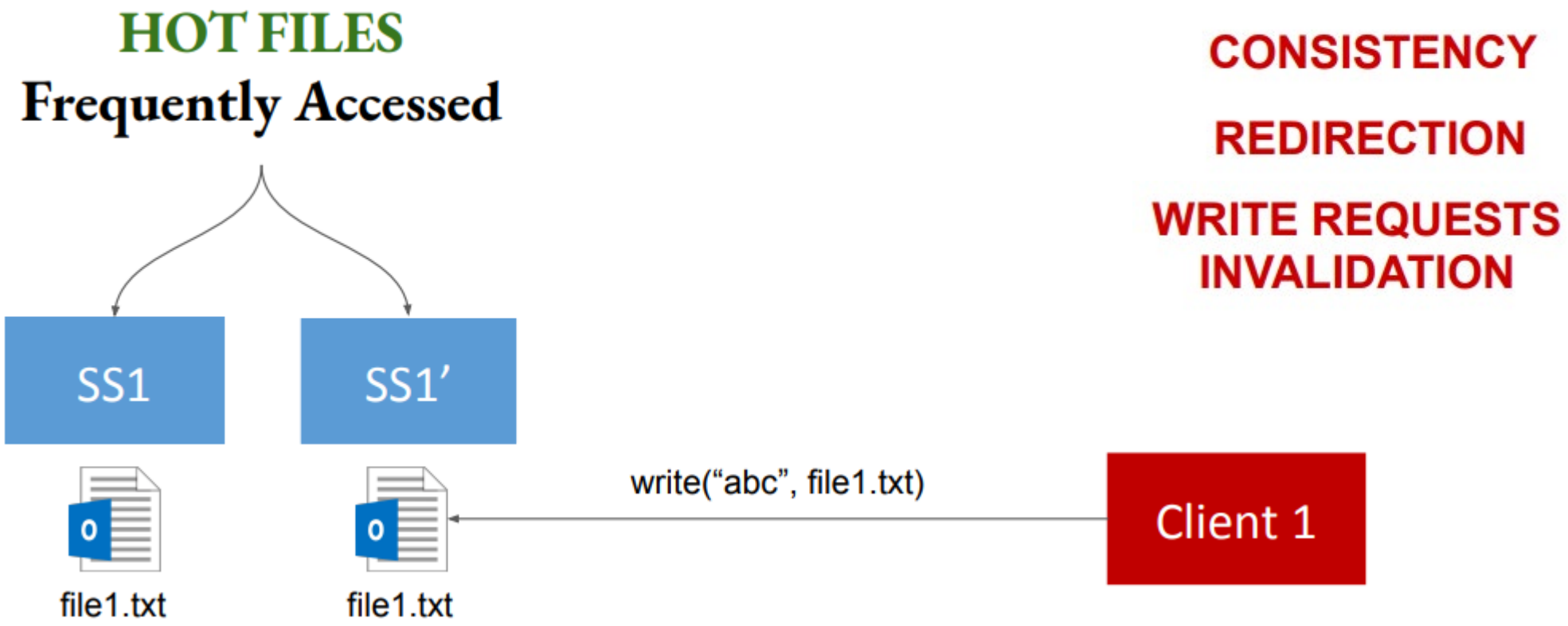
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

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

Load Balancing

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

Implementation Tips: Synchronization

- Consider a **Lock** object that:
  - Stores a list of "**Request**s" (represents a read/write Request)
  - Is assigned to each **Node** in your tree
- In the new LOCK/UNLOCK method:
  - Traverse your tree
  - Obtain/Release locks as necessary

Implementation Tips: Replication

- Keep track of the number of reads for files:
  - You need to modify your Tree data structure
- Create a formula for calculating the number of replicas given the number of reads
  - Similar to the one shown earlier
- After each read/write:
  - Update the number of replicas