

CS244B Project 2: Distributed Replicated Files

May 6th, 2011

Overview

- Implement a file server
- Implement a client library that supports atomic transactions on updates to files stored on servers
- Specify the protocol in a write-up
- Evaluate this replication system in write-up
 - Compare with other designs using reliable unicast connections, e.g. TCP
- Discuss future directions in write-up
 - What would need to change in the protocol and implementation for a real-world deployment?

Client API

```
int InitReplFs(unsigned short portNum,  
               int packetLoss);
```

```
int AddServer(char *id);
```

```
int OpenFile(char *name);
```

```
int WriteBlock(int fd, char *buffer,  
               int byteOffset,  
               int blockSize);
```

```
int Commit(int fd);
```

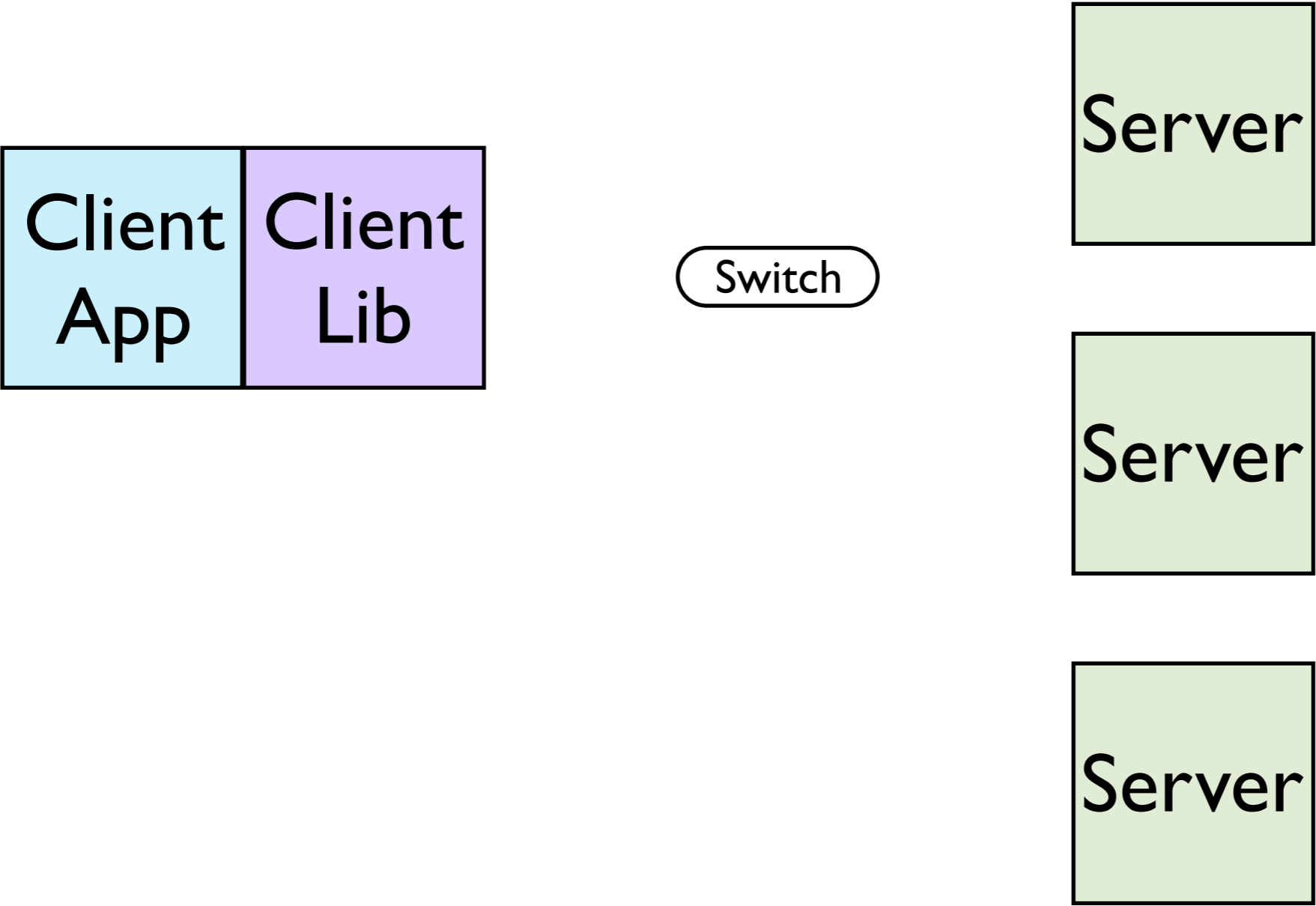
```
int Abort(int fd);
```

```
int CloseFile(int fd);
```

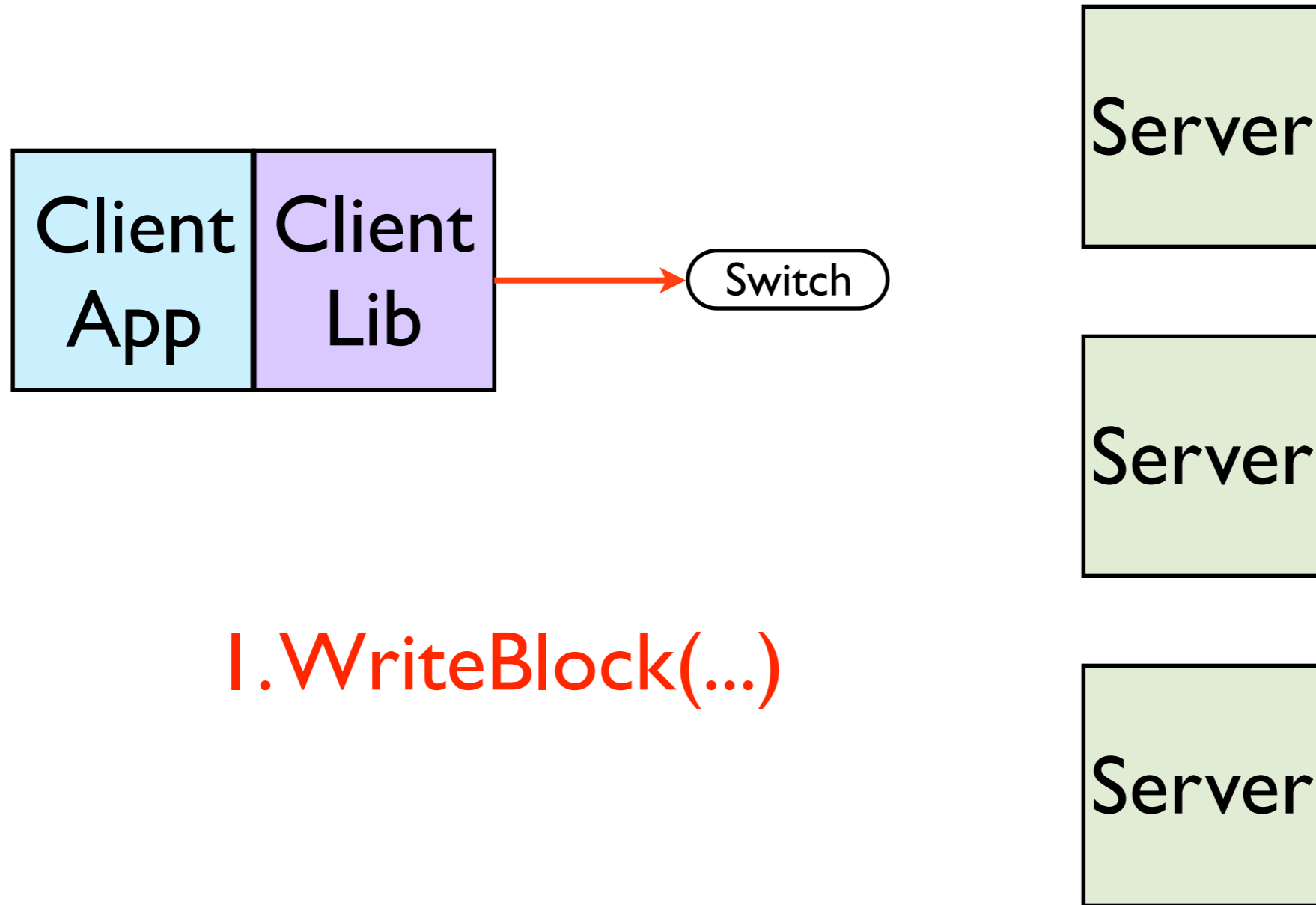
No ReadBlock() call

(We'll test by looking at the files your servers write)

Normal Operation: Multicast Writes, No Ack

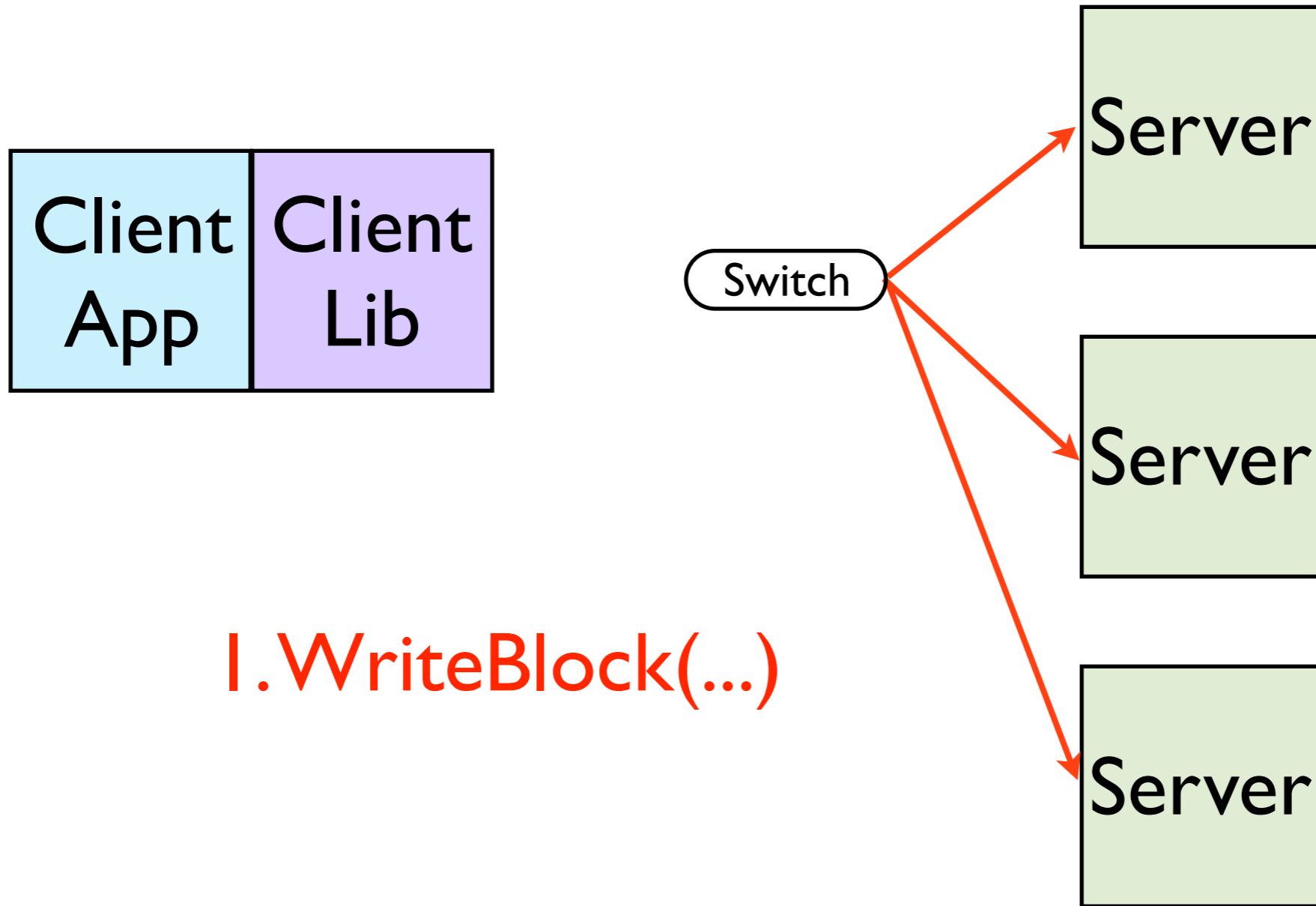


Normal Operation: Multicast Writes, No Ack

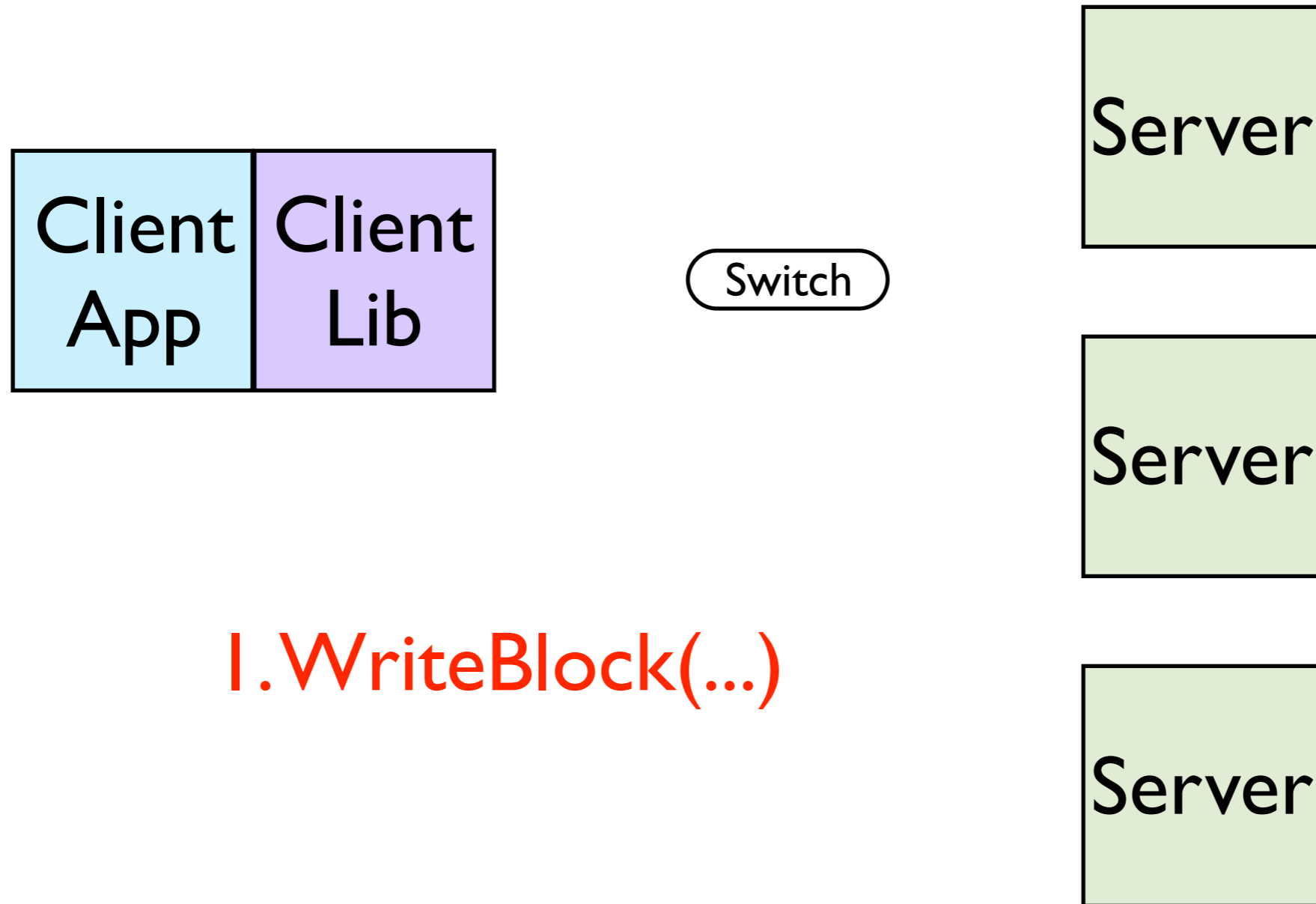


I. WriteBlock(...)

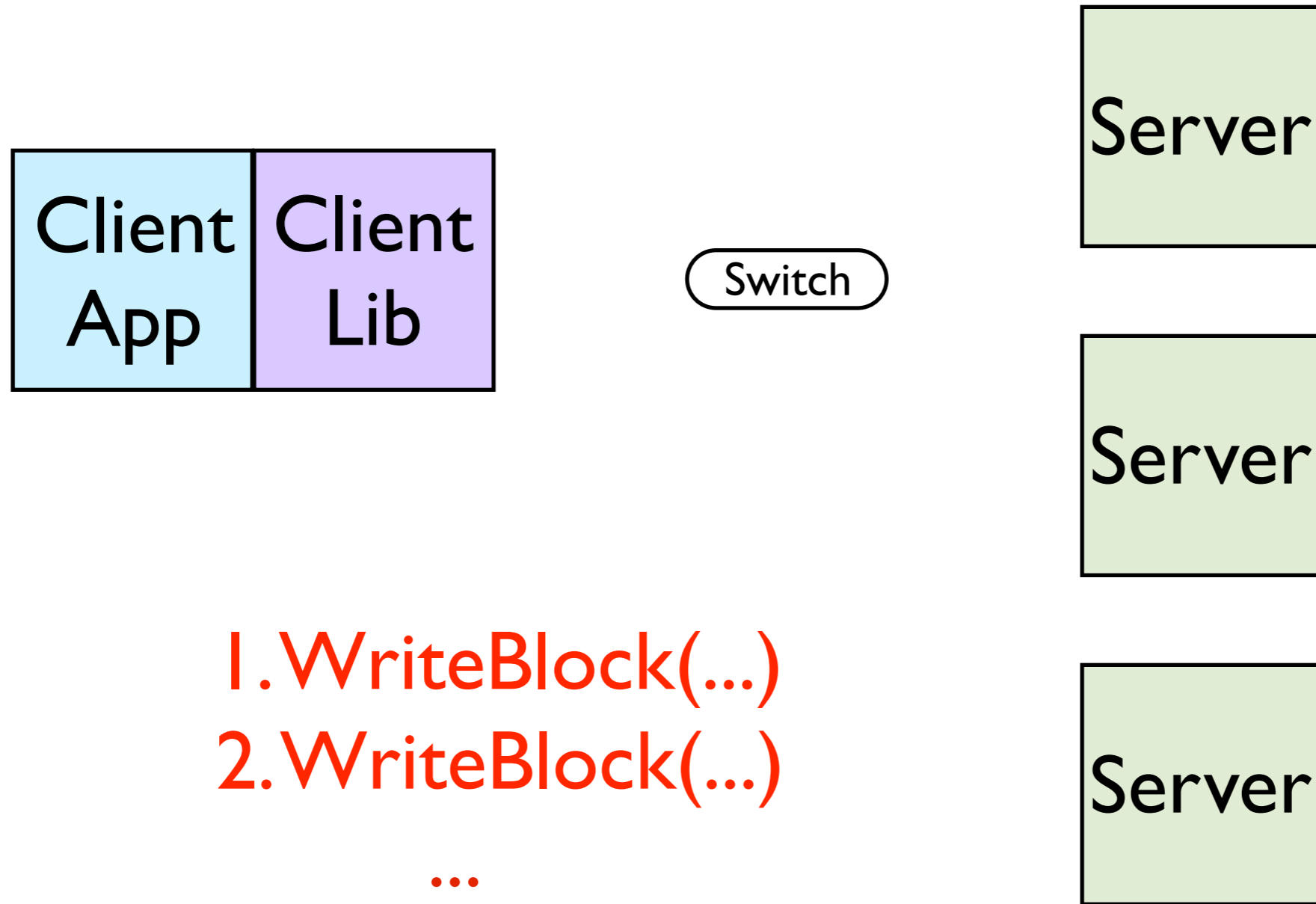
Normal Operation: Multicast Writes, No Ack



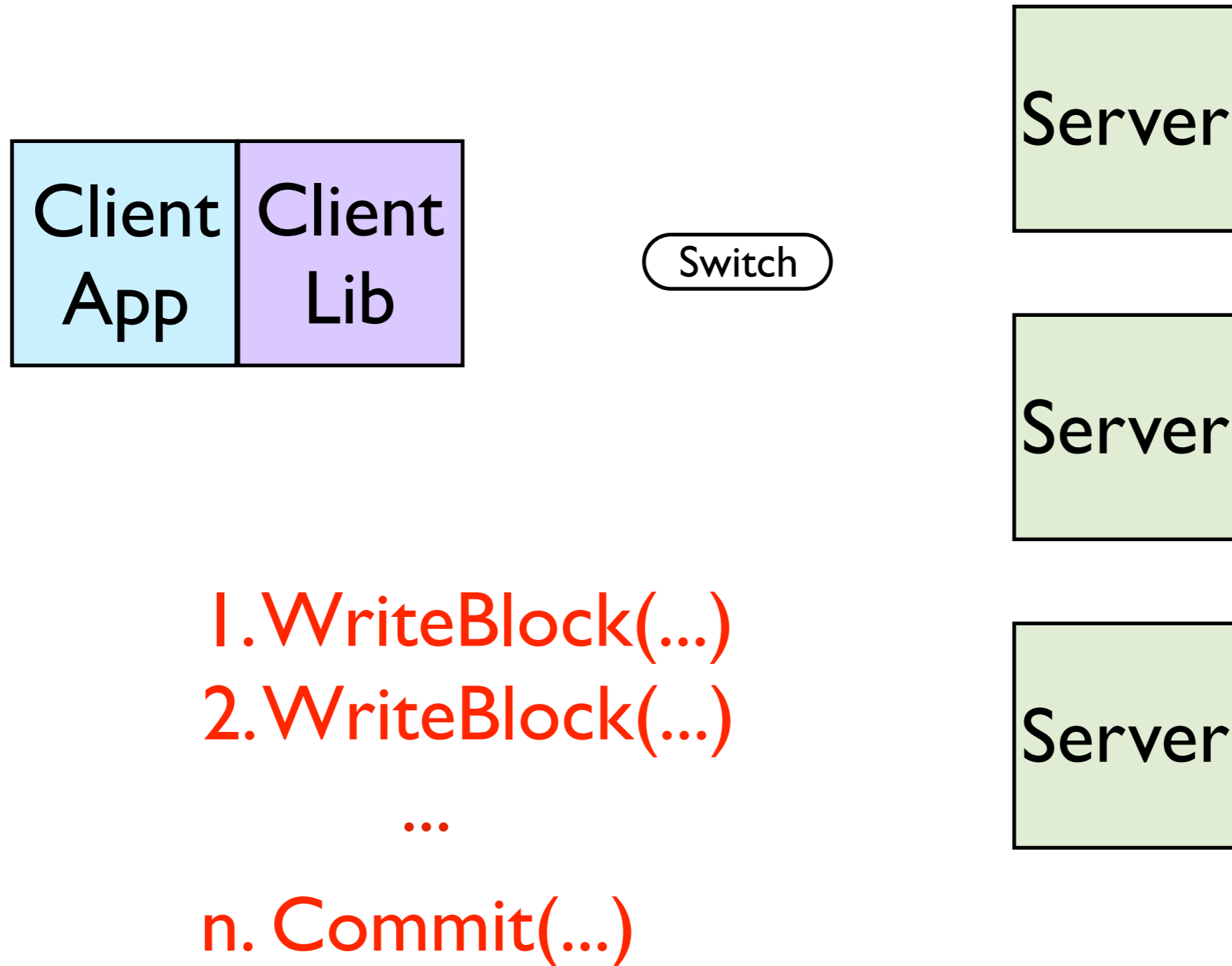
Normal Operation: Multicast Writes, No Ack



Normal Operation: Multicast Writes, No Ack



Normal Operation: Multicast Writes, No Ack



Commit

- On Commit(), all file servers apply the updates
- Issues:
 - Need to ensure consistency between files
 - All servers must commit, or none
 - WriteBlock()s were unreliable
 - Servers may not have all updates
 - Servers could have crashed
 - Commit protocol messages may be unreliable

Two-Phase Commit Protocol

Request Phase

1. Client sends request to commit to all servers
2. Servers each decide if they can commit
3. Each server replies with yes or no

Commit Phase

On success (all servers voted yes):

1. Client sends request to commit message to all servers
2. Each server completes the operation, returns ack to client
3. Client completes when all acks received

On failure (server voted no):

1. Client sends rollback to all servers
2. Servers undo the transaction, send ack to client
3. Client considers transaction undone when all acks received

Failures

- Main complexity in 2PC is handling failure cases
- Solution is for nodes to log where they are in the protocol
 - E.g.:
When a server votes “yes”, they log that response.
If the server crashes and restarts, it notices that “yes” was issued, but no “commit” and enters a recovery protocol.
 - E.g.:
The client (coordinator) may crash after issuing “commit” to a subset of the servers. It must log intent to commit so it can recover on failure and restart.
- Good news:
 - You don't need to implement server & client restarts/recovery
 - But your client shouldn't hang if a server fails

Simplifying Assumptions

- `InitReplFs()` always called first
- `AddServer()` never occurs while a file is open or after it has been modified
- `WriteBlock()` never takes buffer larger than 512 bytes
 - Should keep you from having to fragment packets (Ethernet MTU is ~1500 bytes)
- No more than 128 `WriteBlock()` calls before a `Commit()` or `Abort()`
- No files larger than 1MB (2^{20} , not 10^6)
- No server recovery
 - If a server crashes, your client library must not hang
 - However, we don't require your servers to be restartable

Grading

- 70% Implementation
 - We will run a suite of unit/stress tests using your client library
- 30% Writeup
 - Protocol Spec
 - Evaluation
 - Future Directions

2PC References

- 244B Reader, Chp. 7
- Wikipedia “Two-Phase Commit Protocol”
- <http://www.scs.stanford.edu/07wi-cs244b/notes/l3d.txt>
- <http://www.news.cs.nyu.edu/~jinyang/fa08/notes/ds-lec7.ppt>

Due Date

Friday, May 27th
11:59pm PST