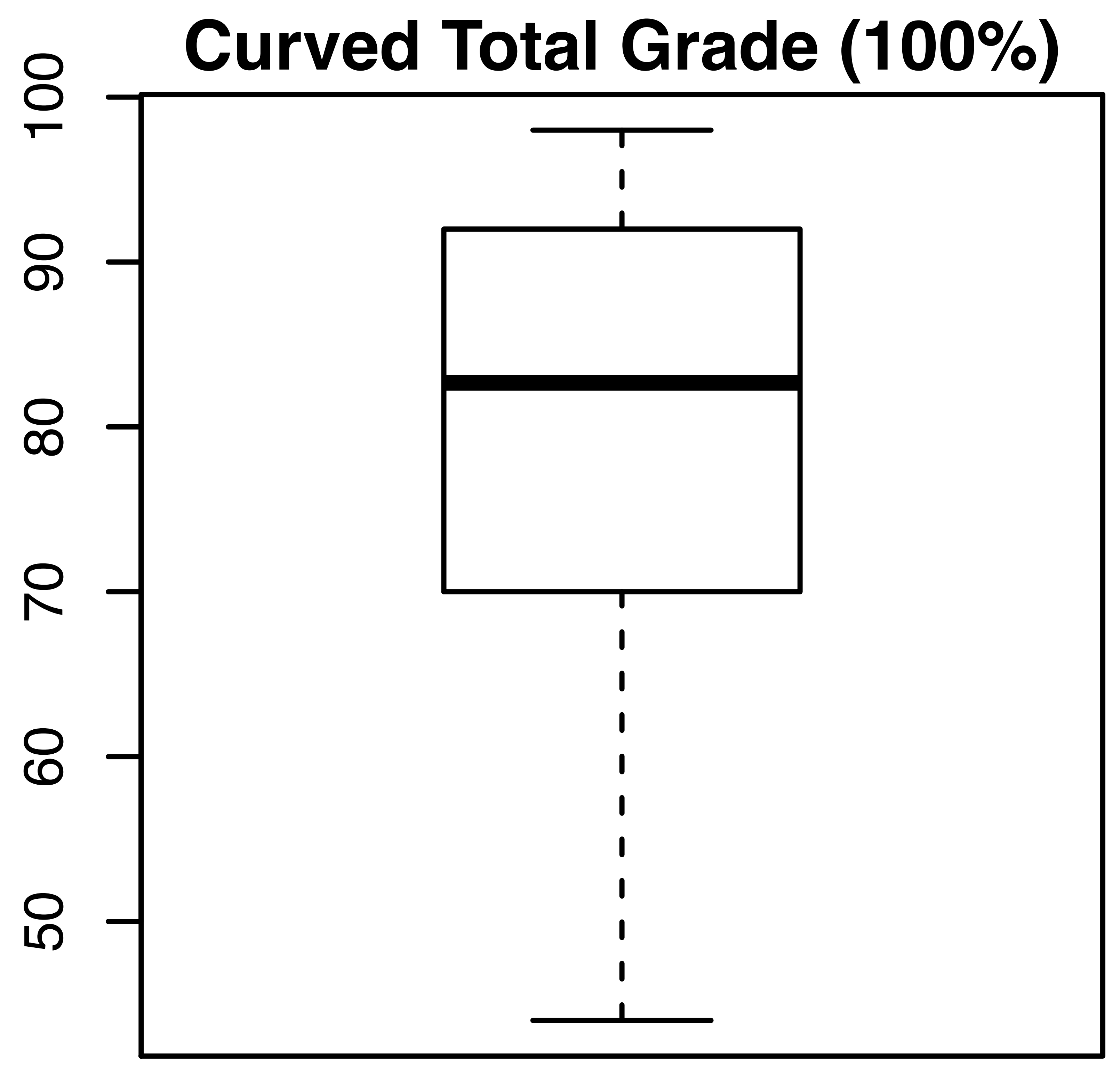
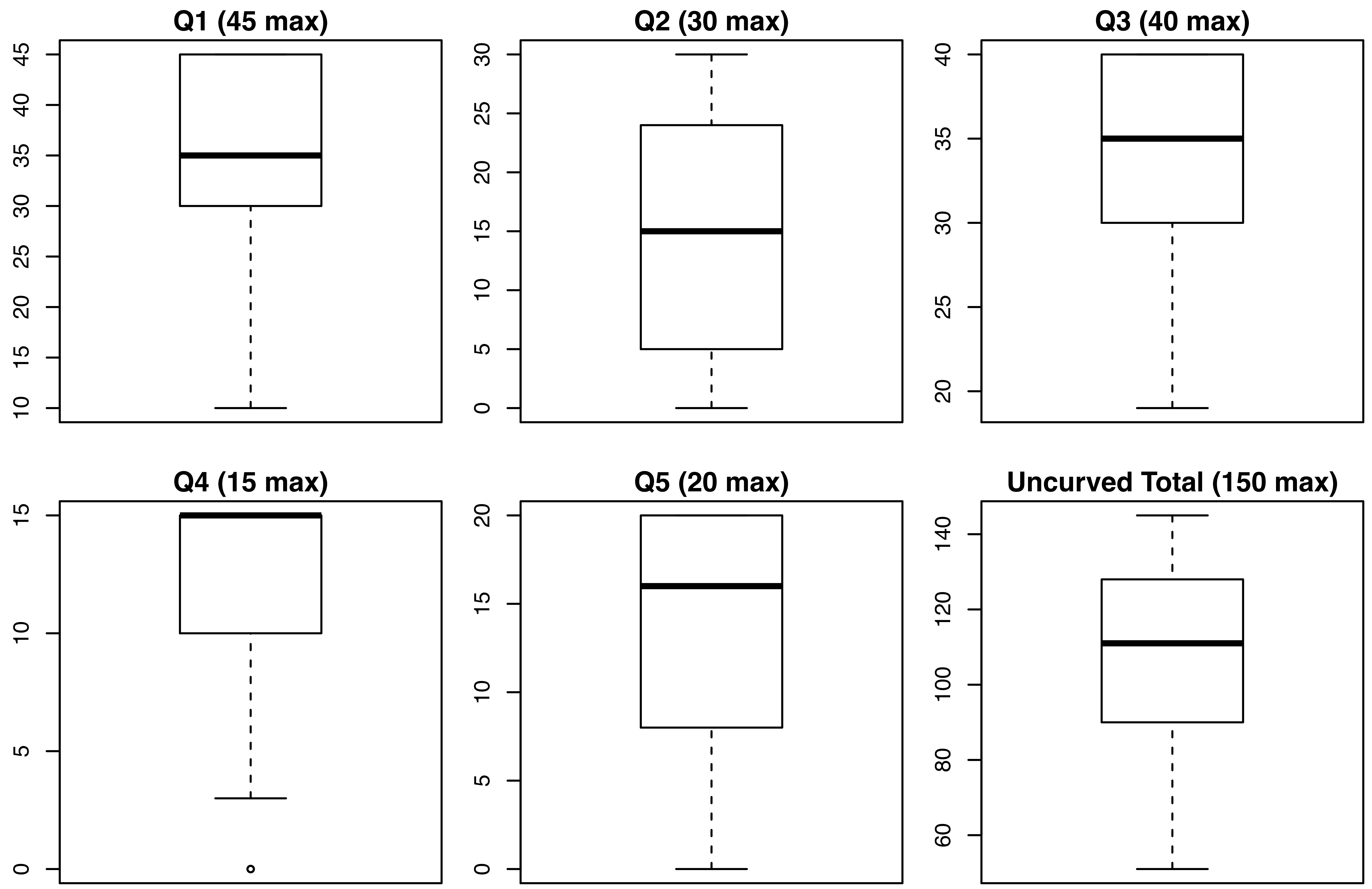
Midterm Review & RMI

CS 475, Spring 2019
Concurrent & Distributed Systems

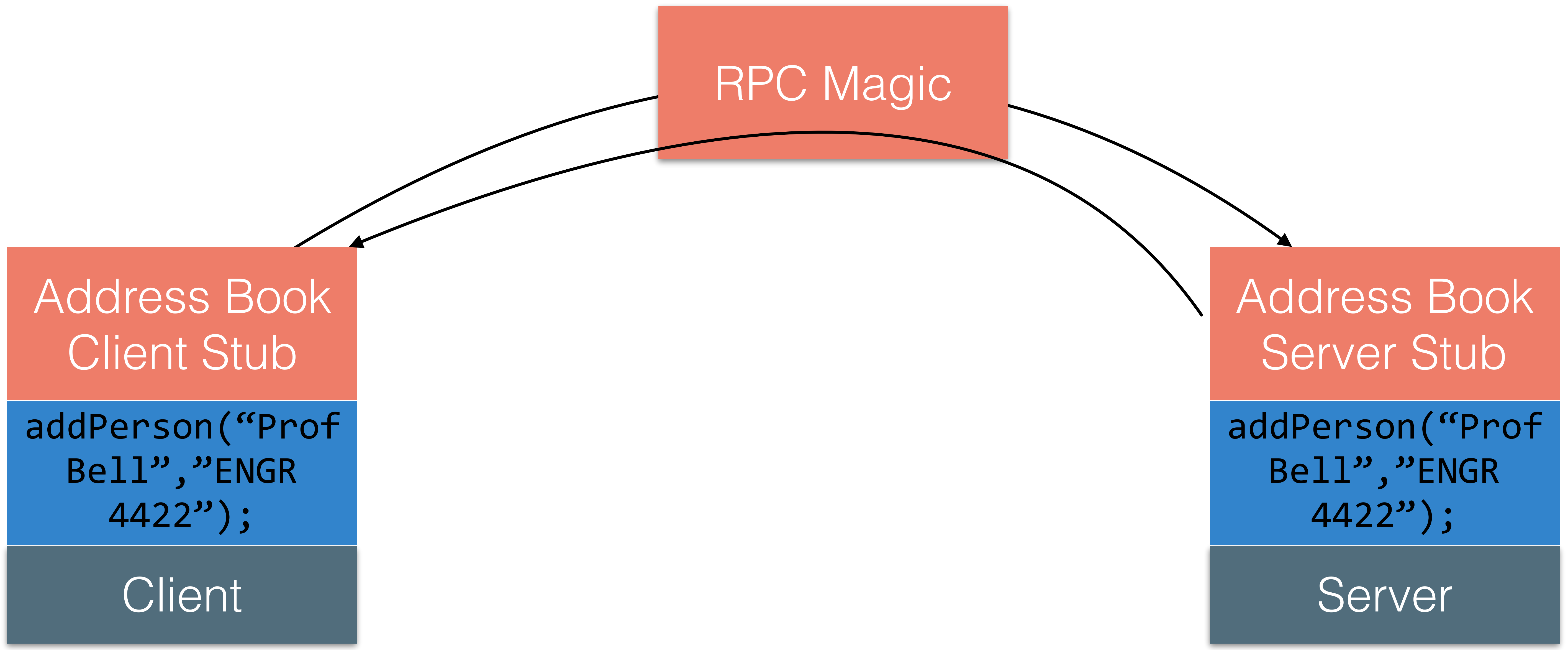
Midterm Discussion





Reminder: Solutions are posted on Piazza

Review: Remote Procedure Calls


RPC Magic
Address Book Client Stub
addPerson(“Prof Bell”,”ENGR 4422”);
Client
Address Book Server Stub
addPerson(“Prof Bell”,”ENGR 4422”);
Server

Review: RPC: The Good Stuff

- Hides networking details
- Hides marshaling details
- Lets you evolve the protocol separately from the communication path
- Easy to add extra shared features
  - Authentication
  - Naming/location

Today

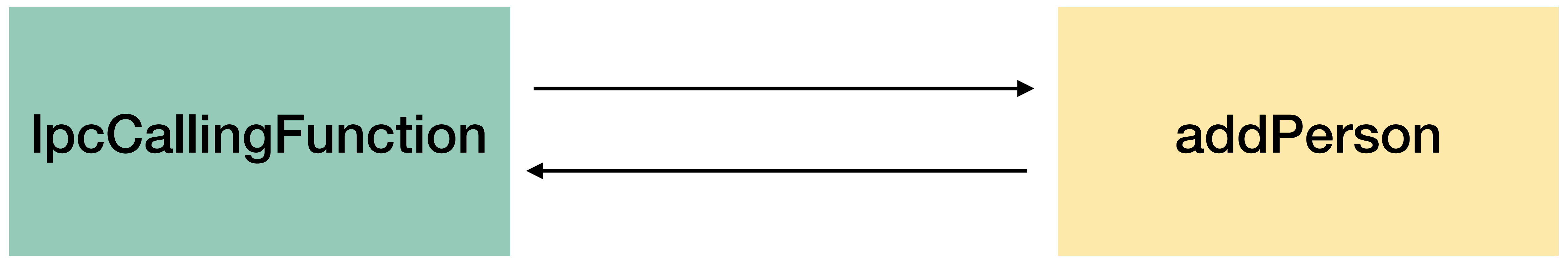
- Midterm Discussion
- Continue discussion of network abstractions: RMI, web services
- Reminders:
  - HW3 posted

RPC: The Bad Stuff

- Latency
- Pointer transfers
- Failures

Local Procedure Call - Failures

- Local code calls `addPerson`
- What can happen?
  - LPC returns (OK, can see return value/error code)
  - LPC doesn't return (OK, know it's still running)

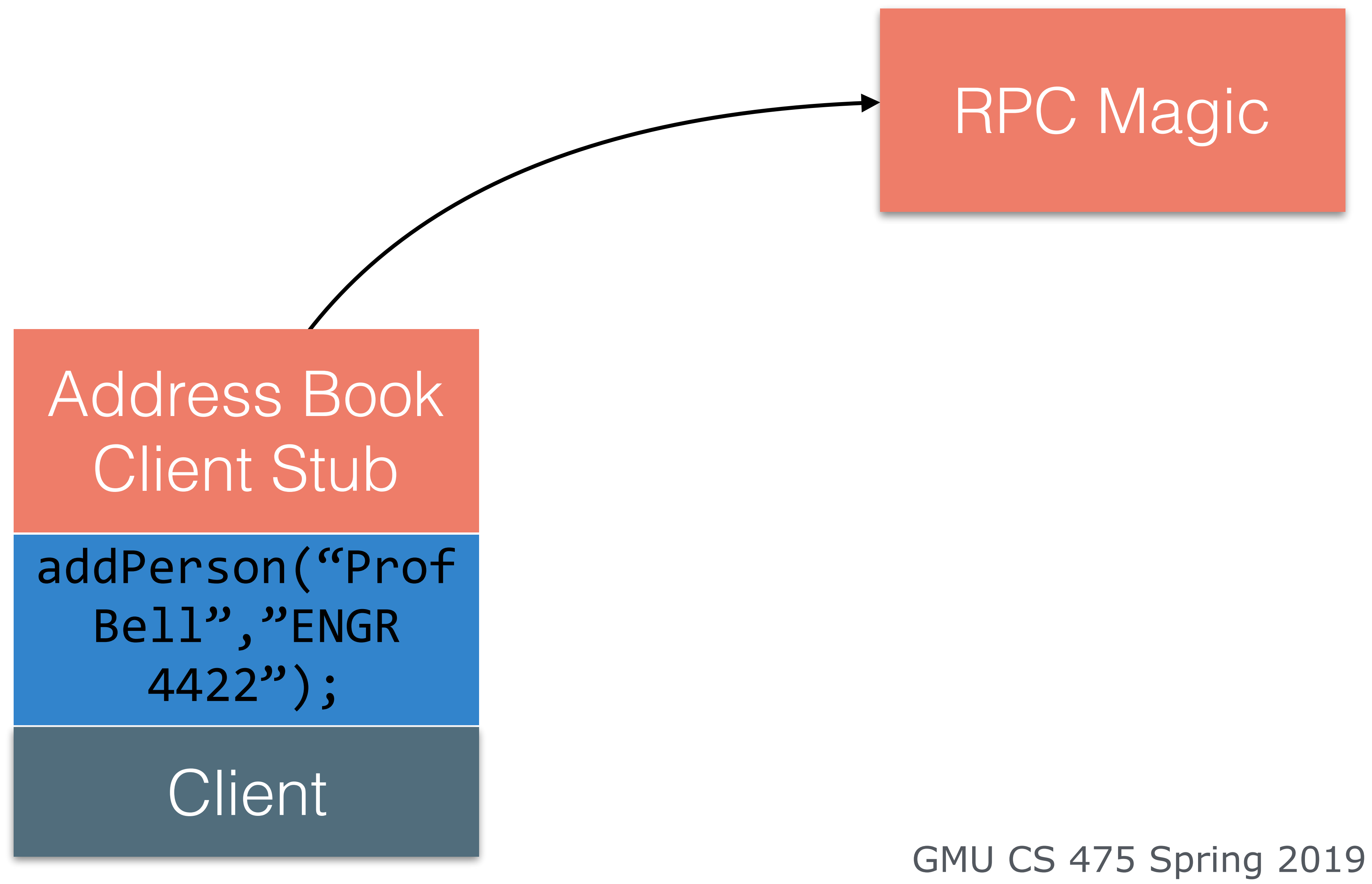


Shared Fate

- Two methods/threads/processes running on the same computer generally have **shared fate**
- They will either both crash, or neither will crash

RPC Failures

- If our client makes an RPC request, but hasn't heard back yet, how do we know what happened?



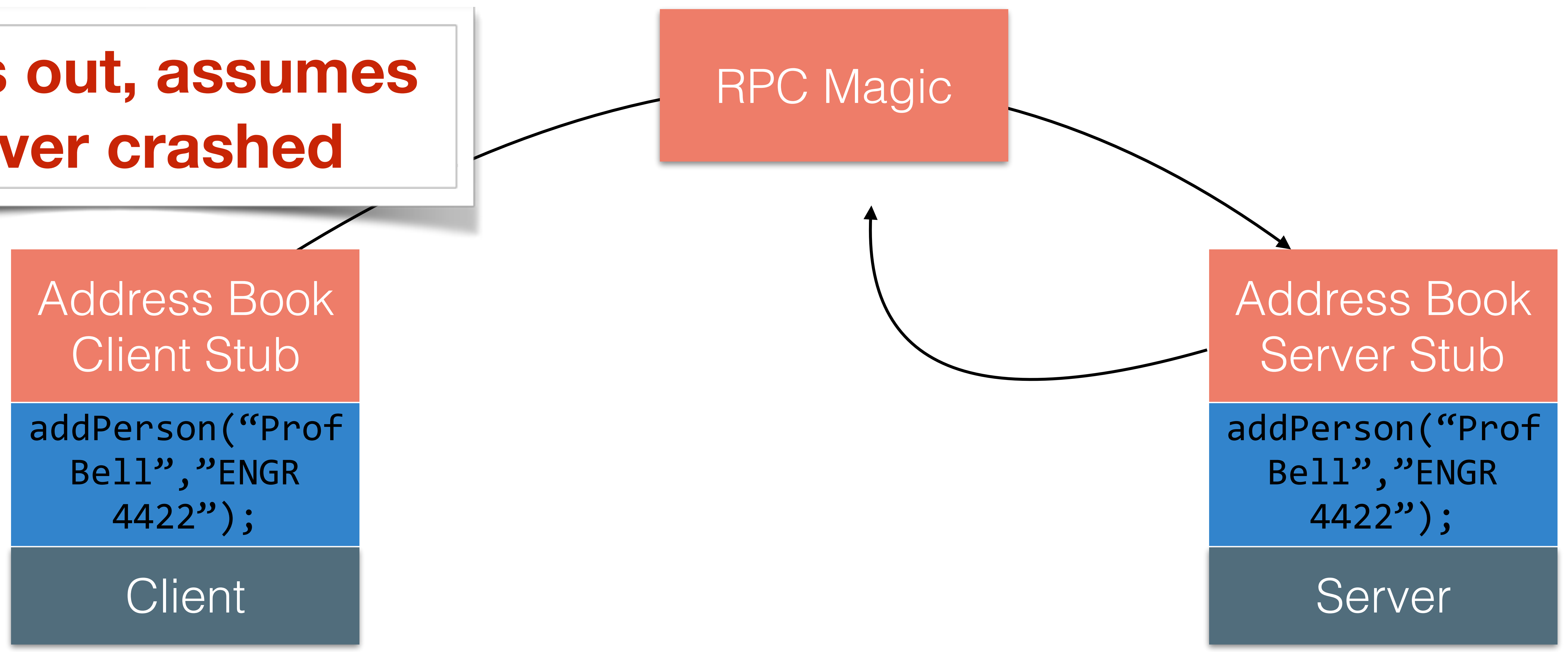
RPC Failures

- If we haven't heard back from the server yet, possibilities are:
  - Server never received request
  - Server received request and crashed
  - Server received request, processed it, crashed
  - Server received requested, processed it, sent response but never received it

Split Brain

- When two machines in a distributed system can't talk to each other, they might start believing different things
- Two sides can not reconcile view of world because they can't talk to each other
- We call this a **split brain** problem

Split Brain in RPC

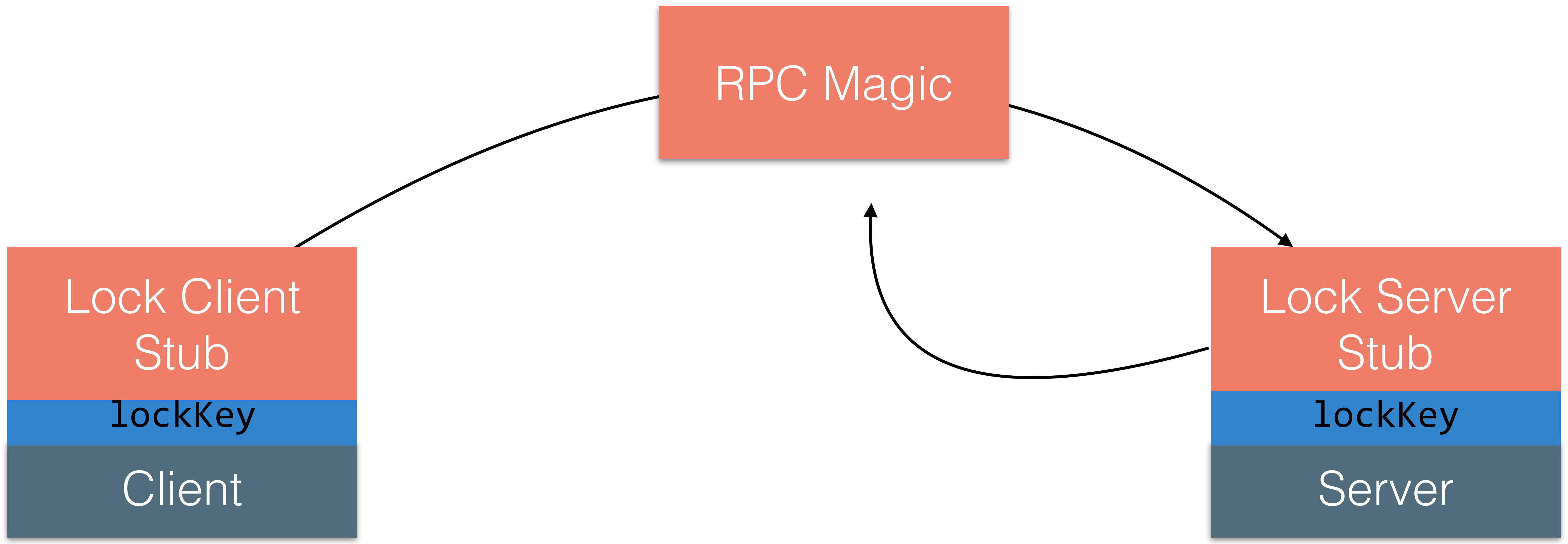

RPC Magic
Address Book Client Stub
addPerson("Prof Bell","ENGR 4422");
Client
Address Book Server Stub
addPerson("Prof Bell","ENGR 4422");
Server

Times out, assumes server crashed

Split brain: Client thinks addPerson didn't succeed, server did complete it though!

Split Brain in RPC

This gets even worse when you consider more complicated semantics



Who has the lock?

RPC Semantics

- No matter what we do, if we want RPC, we have networks, networks might have timeouts/failures
- How do we handle the potential for split brain?
  - If we don't hear a response, just freeze?
- What can the abstraction guarantee?
  - Leak some of this complexity through

RPC Semantics: Exactly once delivery

- Can our RPC abstraction guarantee call is sent and processed exactly once?
- In general, not possible:
  - Server can crash at **any** point
  - Never will know exactly what happened

RPC Semantics: At least once

- RPC system might guarantee **at least once** delivery
- Client library keeps track of unconfirmed messages
- If message is not confirmed, keep re-sending
- Works fine for **idempotent** requests (requests that can be repeated with no side effect)

RPC Semantics: At most once

- RPC system might guarantee at most once delivery
- Client library re-sends if it doesn’t hear an ACK
- Client library adds message IDs
- Server library keeps track of received message IDs
- Problems?
  - Server needs to perpetually track received IDs

RPC - Implementation Issues

- Performance is the usual issue:
    - When do you generate the stubs?
    - How do you patch in the stubs?
    - What data is copied from client to server?
- Environmental concerns:
    - What if client/server are very different (OS/language)?

Java RMI

- Synchronous (client method doesn’t return until server completes)
- At most once delivery
- Hence, in the event of a communication failure, an exception is thrown on your client
- Implications:
  - Client code needs to be aware that failures might happen (and exception might be thrown)
  - Client code needs to have some plan to handle when a message fails to get through (application specific)

Java RMI

- Threading model:
  - What happens when there are multiple simultaneous RMI requests to the same server?
- RMI creates a *thread pool*, a set of threads ready to handle each request
  - Subsequent calls from the same client might or might not use the same thread
  - Subsequent calls from other clients might use the same thread as others
- Implications:
  - Can process multiple requests simultaneously
  - Need to be cognizant of thread safety

Java RMI

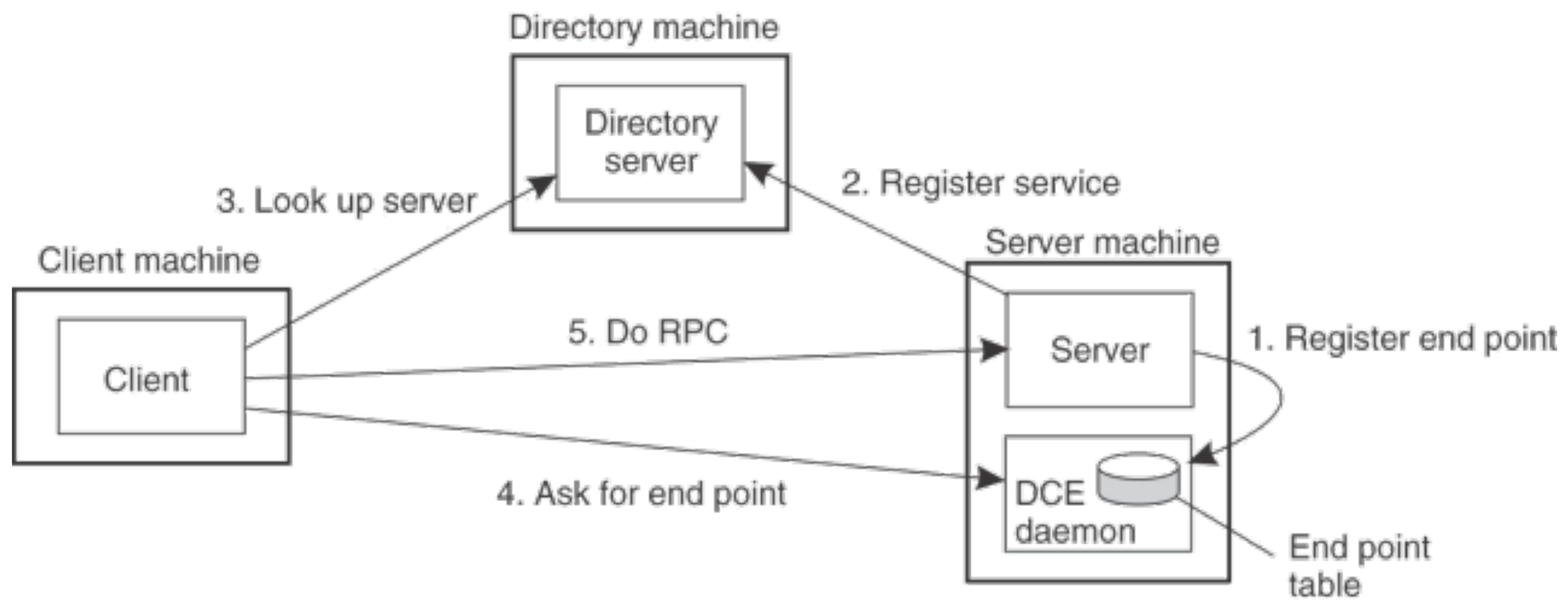
```java
public interface AddressBook extends Remote {
   public LinkedList<Person> getAddressBook() throws RemoteException;

   public void addPerson(Person p) throws RemoteException;
}
```

```java
AddressBook book = new AddressBookServer();
AddressBook stub = (AddressBook) UnicastRemoteObject.exportObject(book, 0);
Registry registry = LocateRegistry.createRegistry(port);
registry.rebind("AddressBook", stub);
```

```java
Registry registry = LocateRegistry.getRegistry("localhost", 9000);
AddressBook addressBook = (AddressBook) registry.lookup("AddressBook");
```

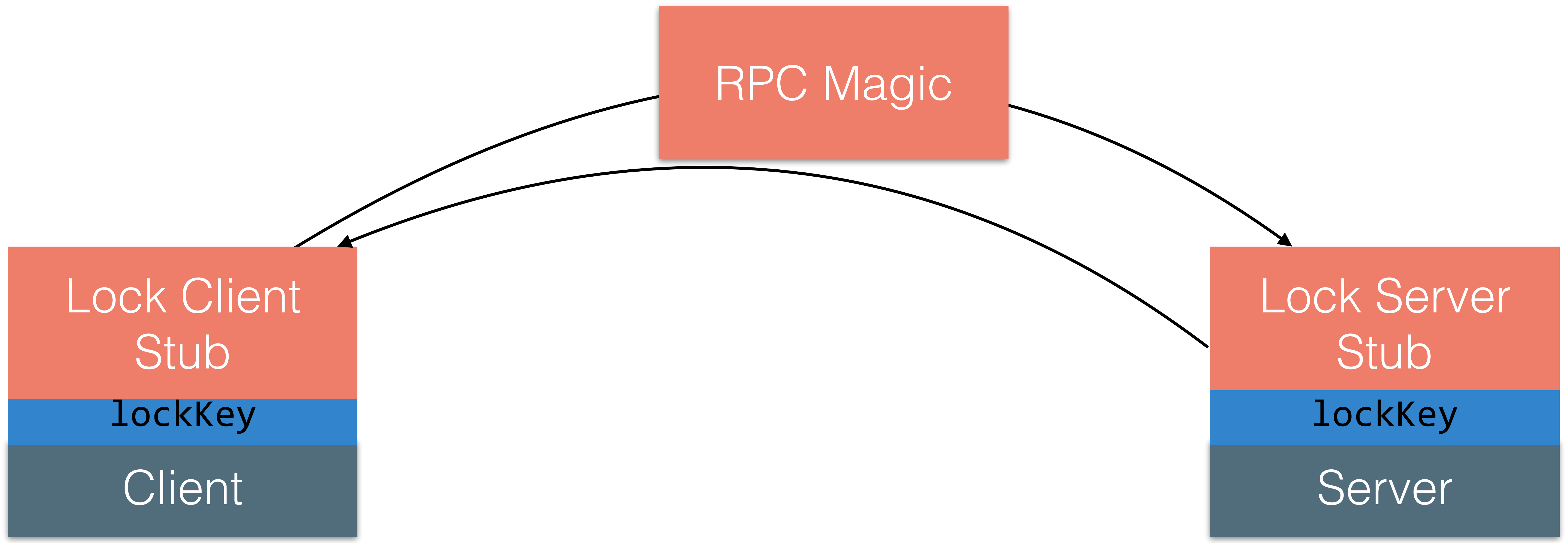
Java RMI

- Registration of a server makes it possible for a client to locate the server and bind to it
- Server location is done in two steps:
  - Locate the server's machine.
  - Locate the server on that machine.



Split Brain in RPC

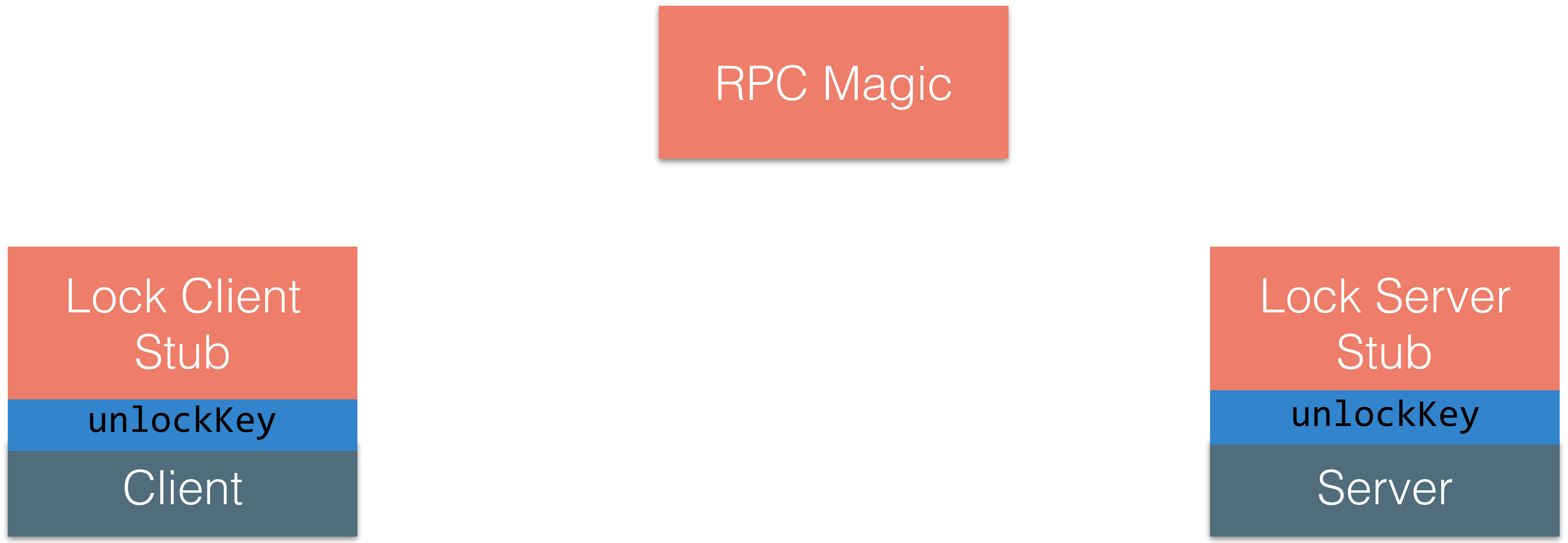
This gets even worse when you consider more complicated semantics



Who has the lock? How do we handle this?

Split Brain in RPC

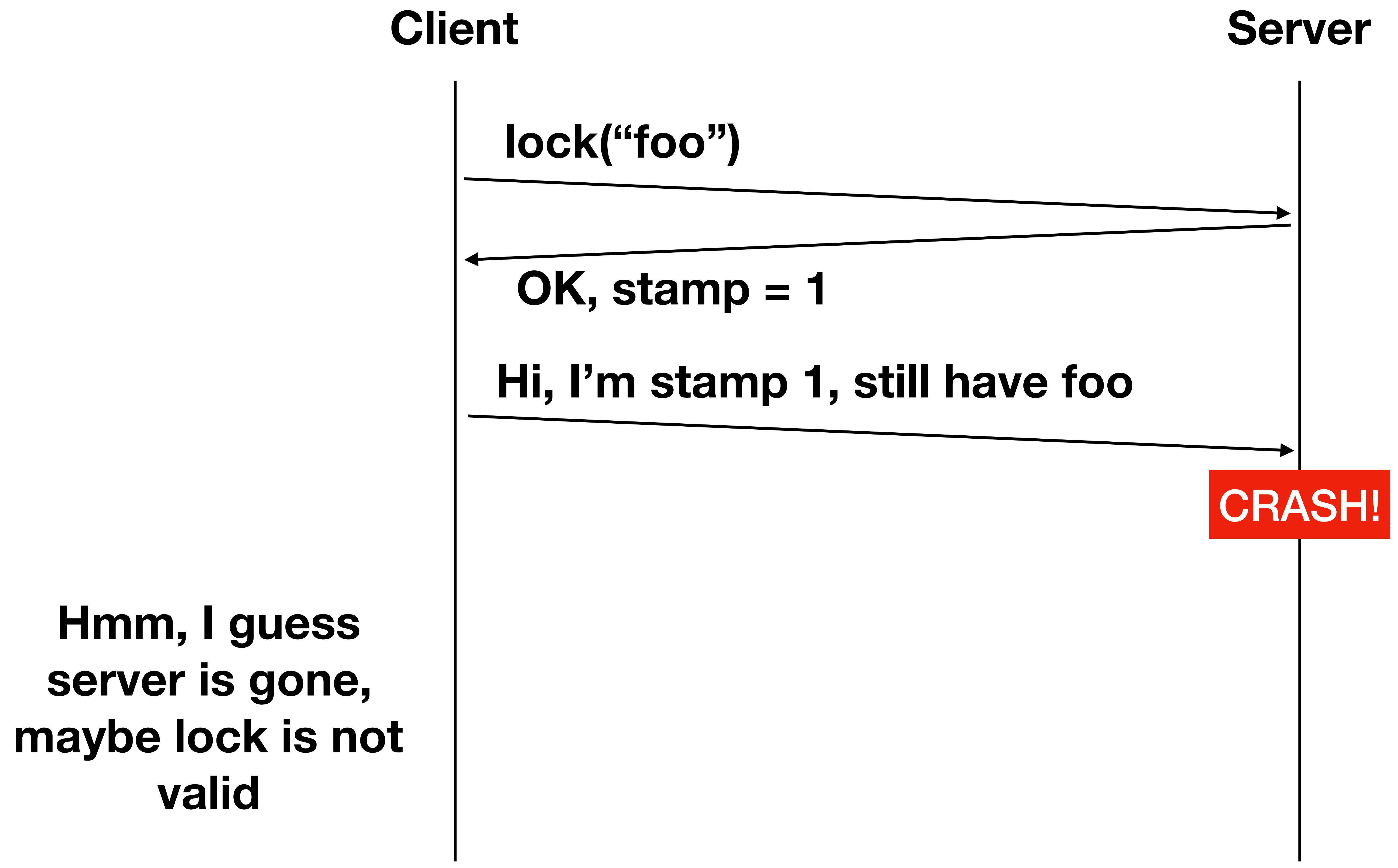
This gets even worse when you consider more complicated semantics



Who has the lock? How do we handle this?

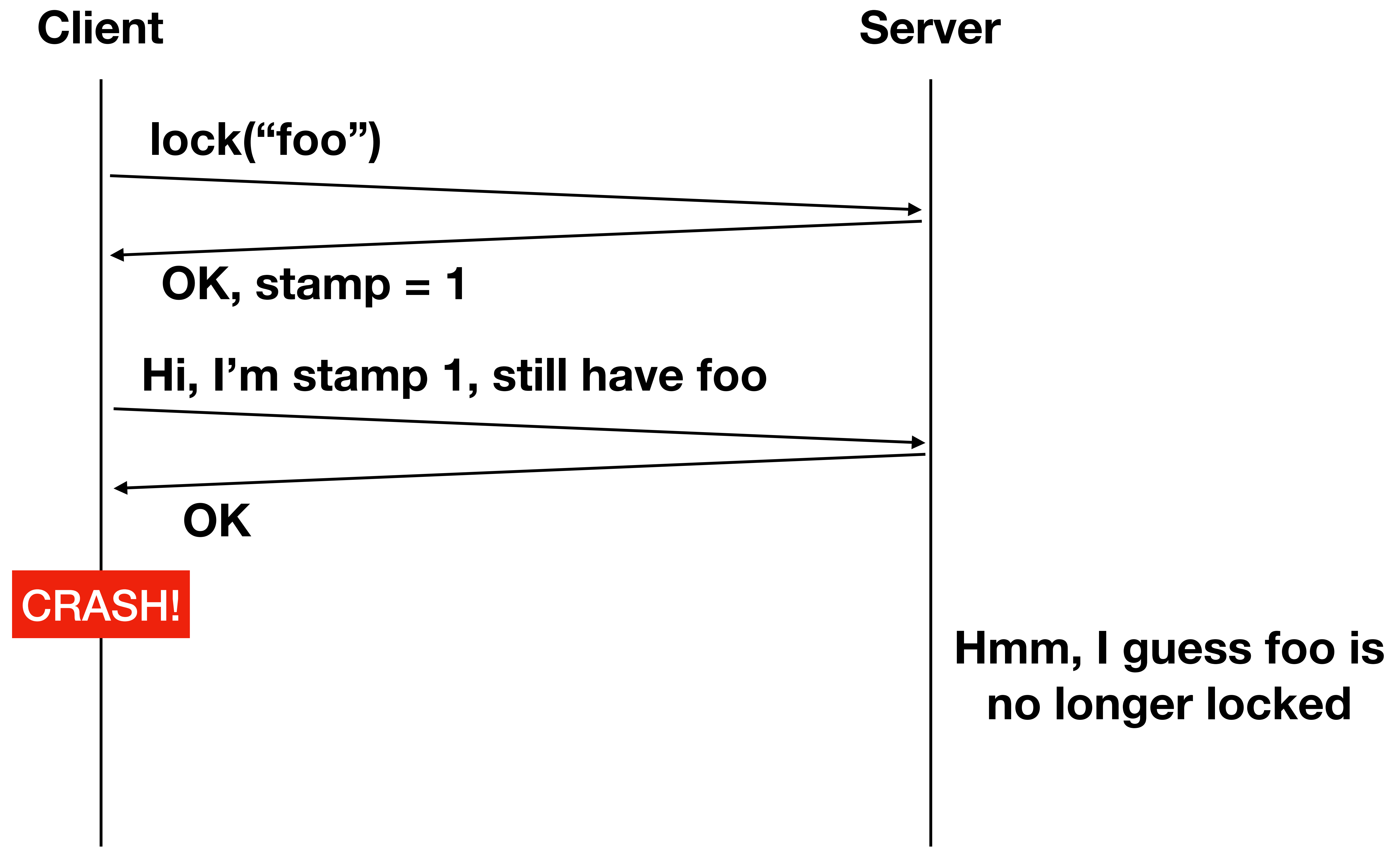
Sidebar: Heartbeat Protocols

- Allow client/server to remain aware of each other's status
- For HW3: does client still have locks (client checking server, server checking client)


Client
Server
lock("foo")
OK, stamp = 1
Hi, I'm stamp 1, still have foo
CRASH!
Hmm, I guess server is gone, maybe lock is not valid

Sidebar: Heartbeat Protocols

- Allow client/server to remain aware of each other’s status
- For HW3: does client still have locks (client checking server, server checking client)


Client
Server
lock(“foo”)
OK, stamp = 1
Hi, I’m stamp 1, still have foo
OK
CRASH!
Hmm, I guess foo is no longer locked

Sidebar: Heartbeat Protocols

- We call these time-limited locks **leases**
- What does a lease guarantee?
  - If no network failures
    - Locks that are relinquished when client crashes
  - If network failures/delays:
    - Nothing

RPC Summary

- Expose RPC properties to client, since you cannot hide them
  - Application writers have to decide how to deal with partial failures
  - Consider: E-commerce application vs. game

Summary

- Procedure calls
  - Simple way to pass control and data
  - Elegant transparent way to distribute application
  - Not only way…
- Hard to provide true transparency
  - Failures
  - Performance
  - Memory access
  - Etc.
- How to deal with hard problem: give up and let programmer deal with it

This work is licensed under a Creative Commons Attribution-ShareAlike license