COEN6731 Distributed Software Systems

Week 3: RPCs, Raft

Gengrui (Edward) Zhang, PhD
Web: gengruizhang.com

Today's outline

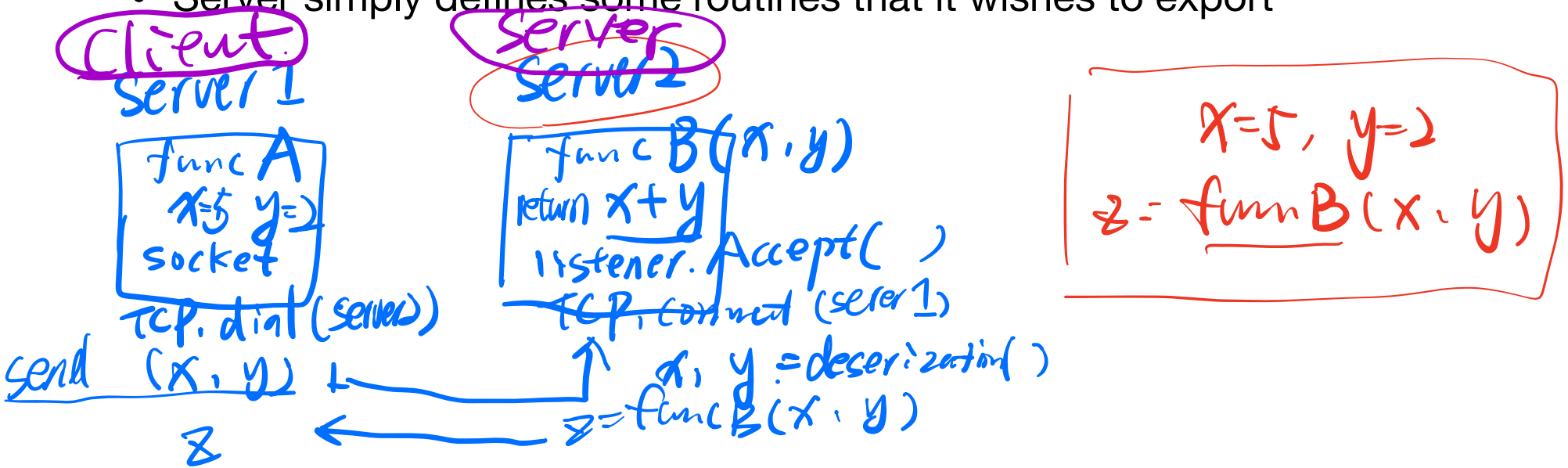
Remote procedure calls (RPCs)

Raft

- Log replication
- Leader election

What is remote procedure calls (RPCs)?

- Goal: make the process of executing code on a remote machine as simple and straight-forward as calling a local function
  - Client issues a procedure call and wait for results to be returned
  - Server simply defines some routines that it wishes to export

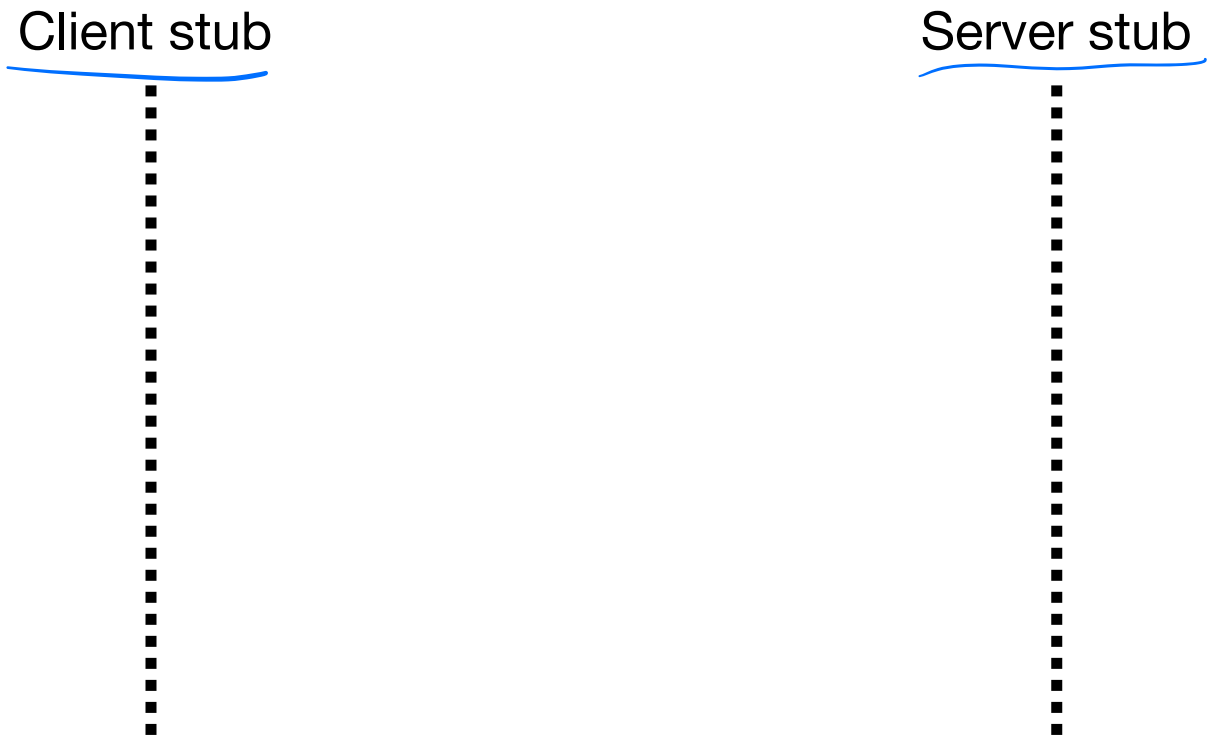


What is remote procedure calls (RPCs)?

- Goal: make the process of executing code on a remote machine as simple and straight-forward as calling a local function
  - Client issues a procedure call and wait for results to be returned
  - Server simply defines some routines that it wishes to export
- RPC has two important components:
  - **Stub generator** (aka protocol compiler)
  - **Run-time library**

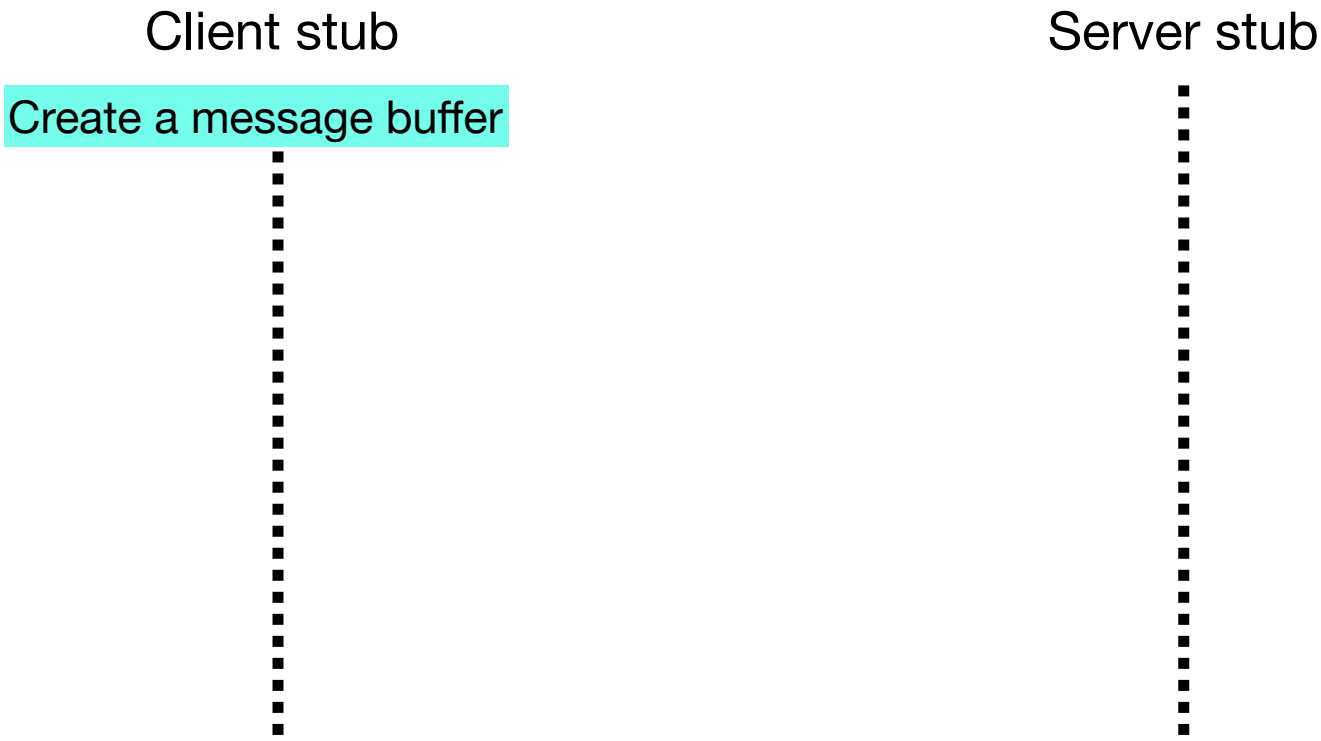
Stub generator

- Automate packing function arguments and results into messages



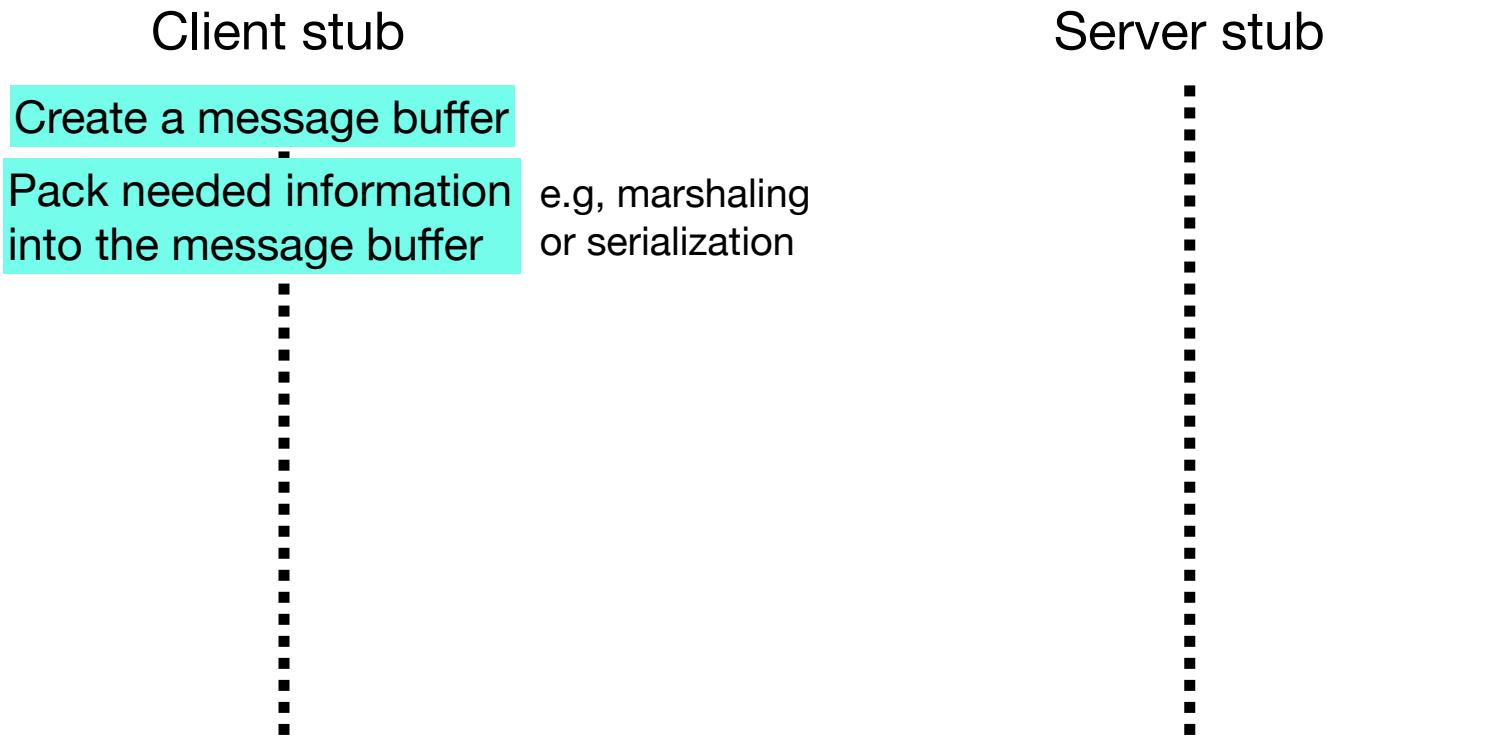
Stub generator

- Automate packing function arguments and results into messages



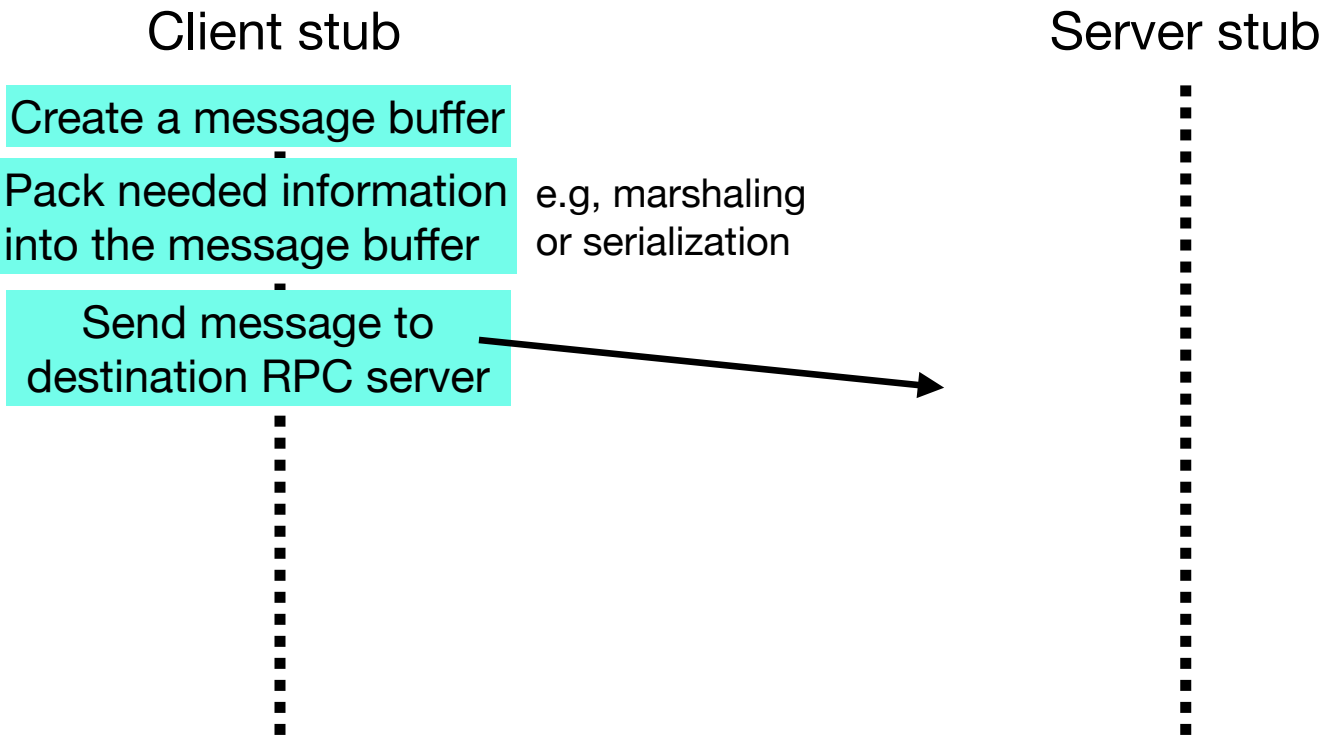
Stub generator

- Automate packing function arguments and results into messages



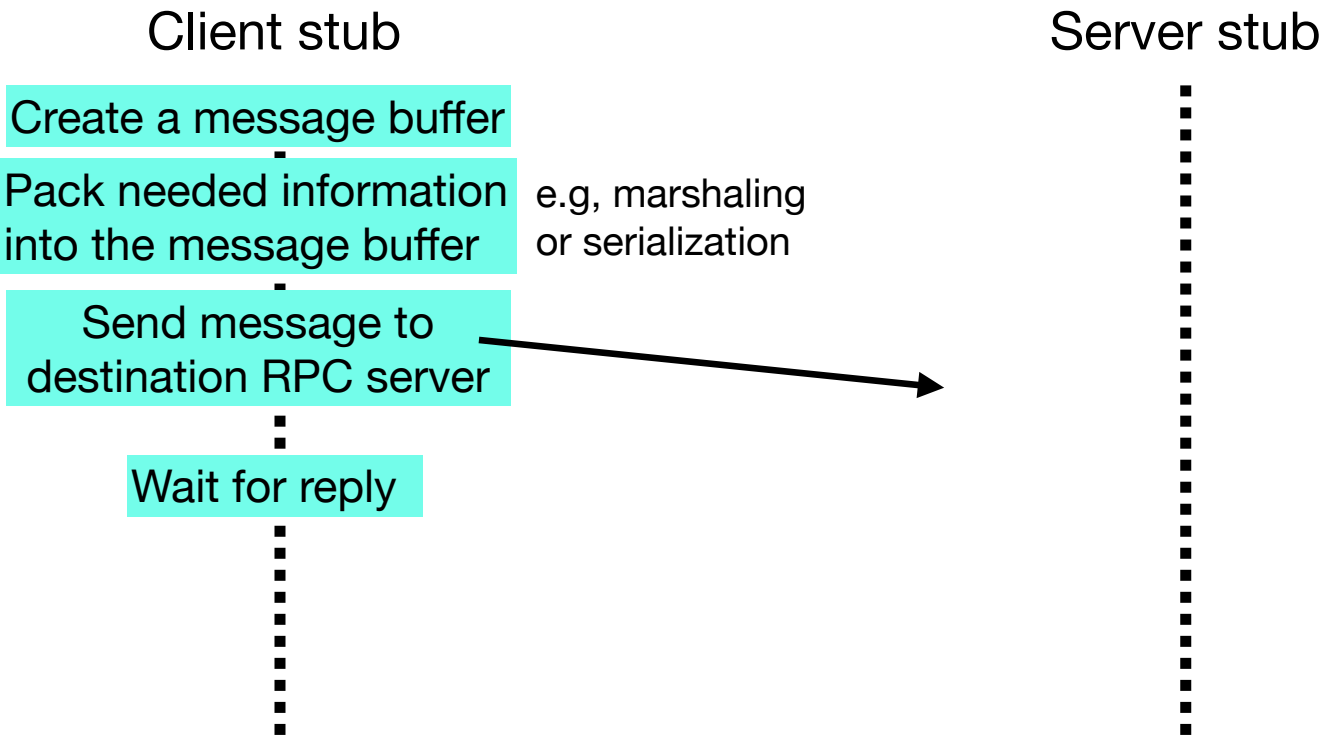
Stub generator

- Automate packing function arguments and results into messages



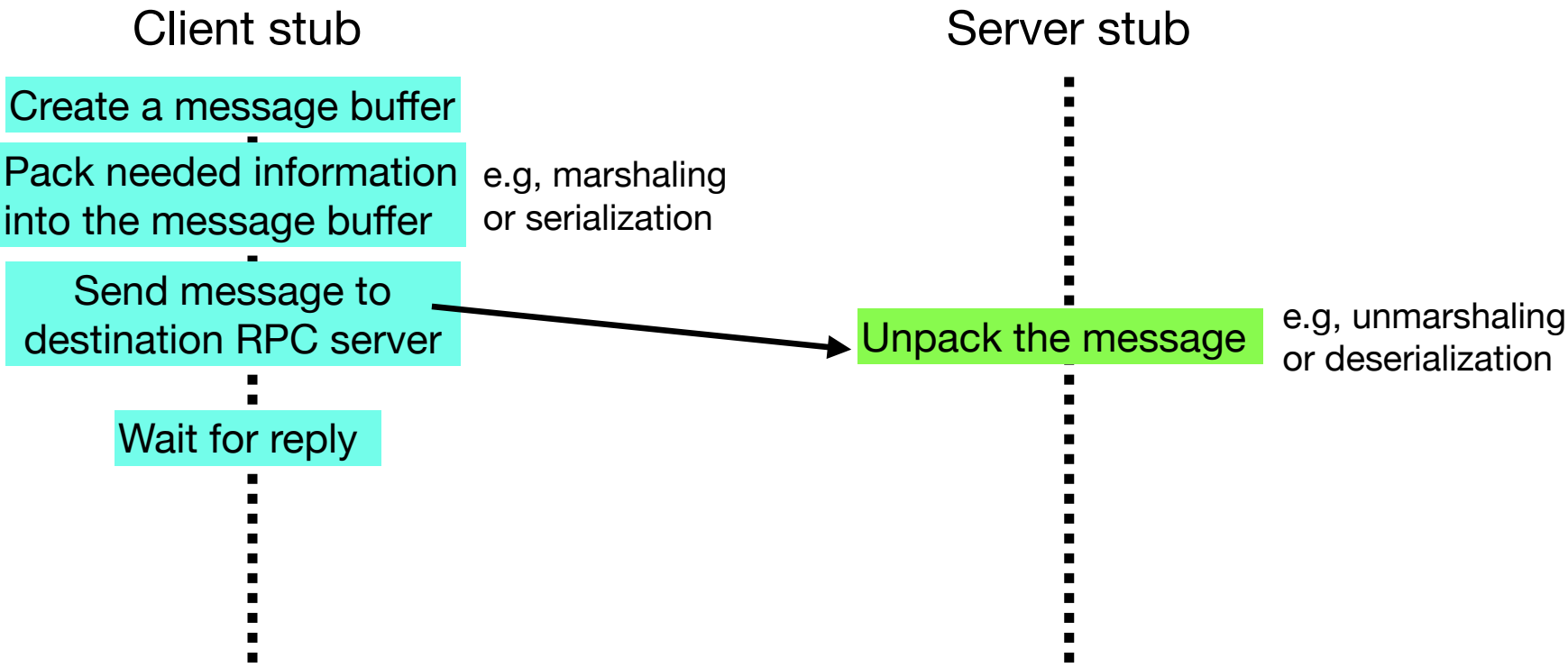
Stub generator

- Automate packing function arguments and results into messages



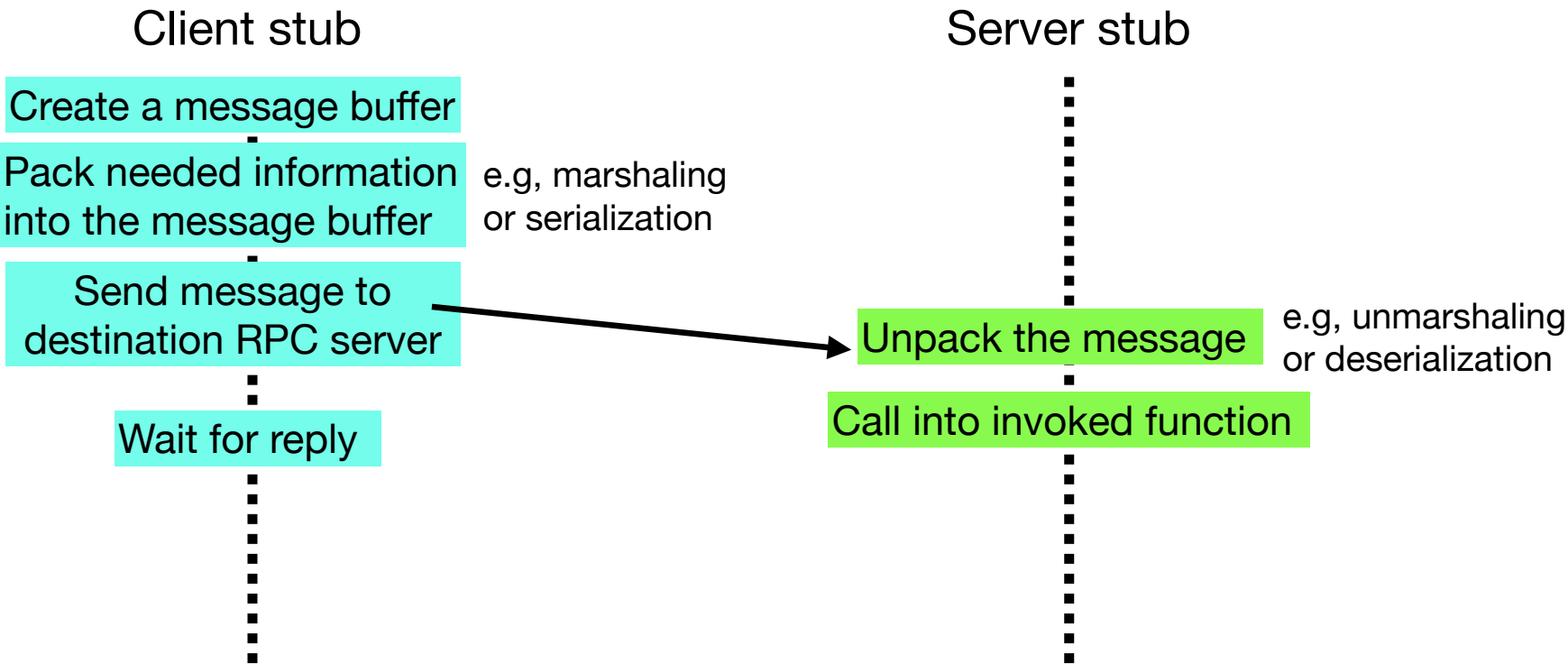
Stub generator

- Automate packing function arguments and results into messages



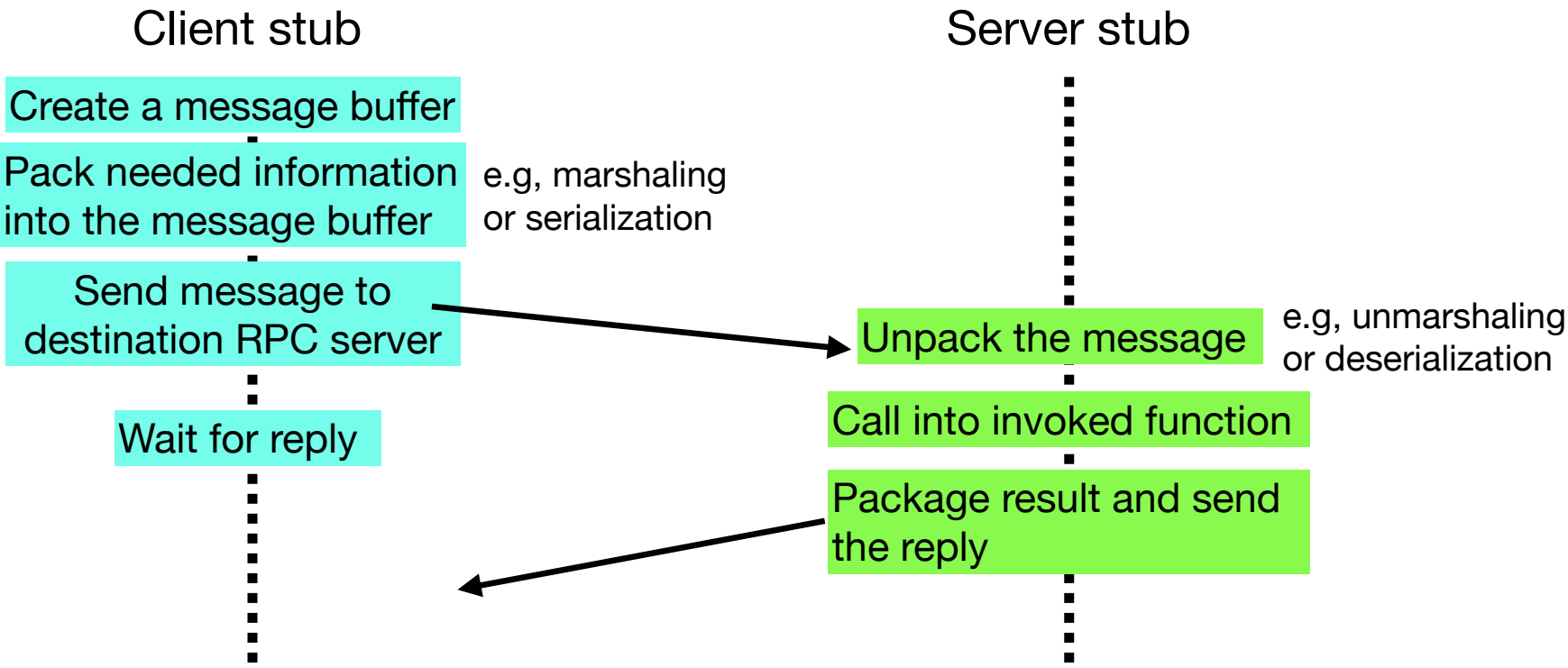
Stub generator

- Automate packing function arguments and results into messages



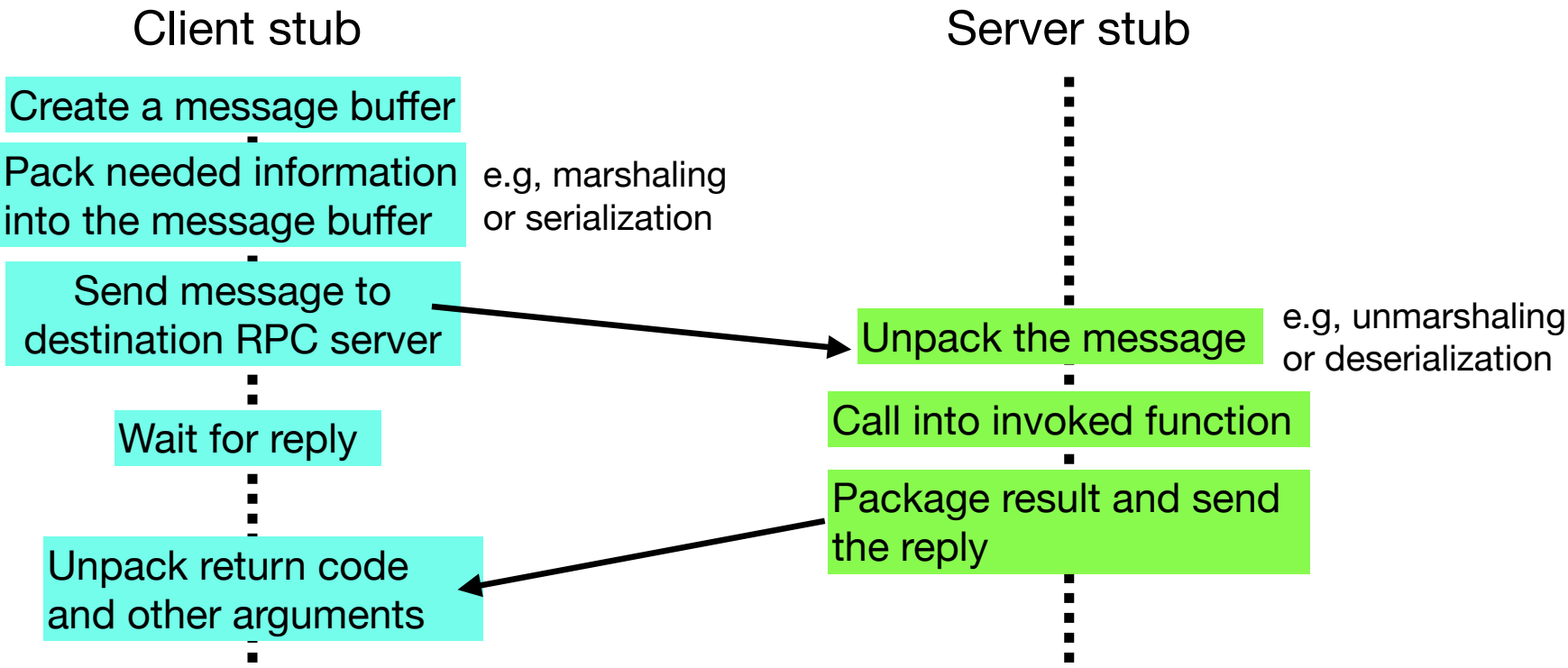
Stub generator

- Automate packing function arguments and results into messages



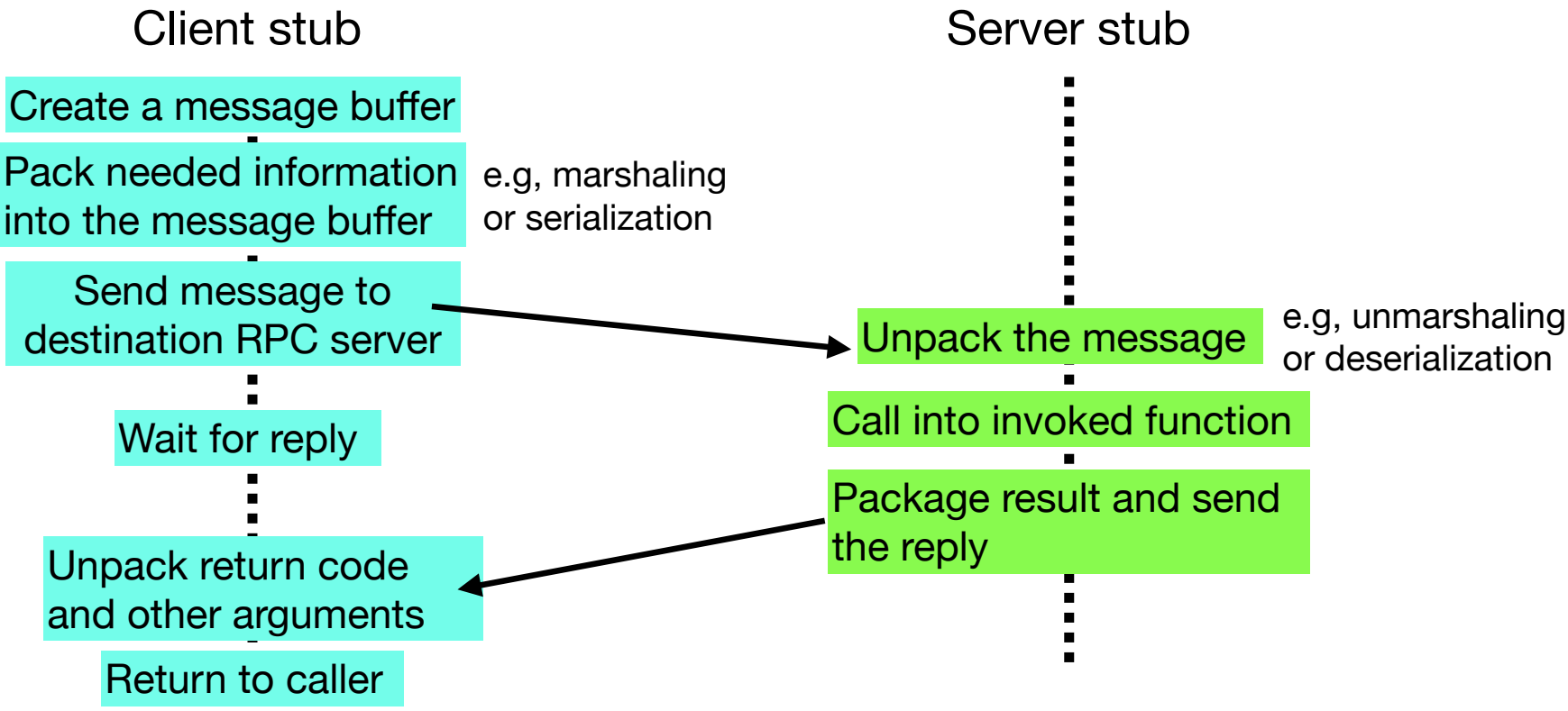
Stub generator

- Automate packing function arguments and results into messages



Stub generator

- Automate packing function arguments and results into messages



Run-time library (RTL)

RTL handles communication between client and server. E.g.,

Run-time library (RTL)

RTL handles communication between client and server. E.g.,

- Naming
  - E.g., hostnames and port numbers provided by internet protocols

Run-time library (RTL)

RTL handles communication between client and server. E.g.,

- Naming
  - E.g., hostnames and port numbers provided by internet protocols
- Transport-level protocol
  - E.g., TCP (e.g., gRPC) and UDP

Run-time library (RTL)

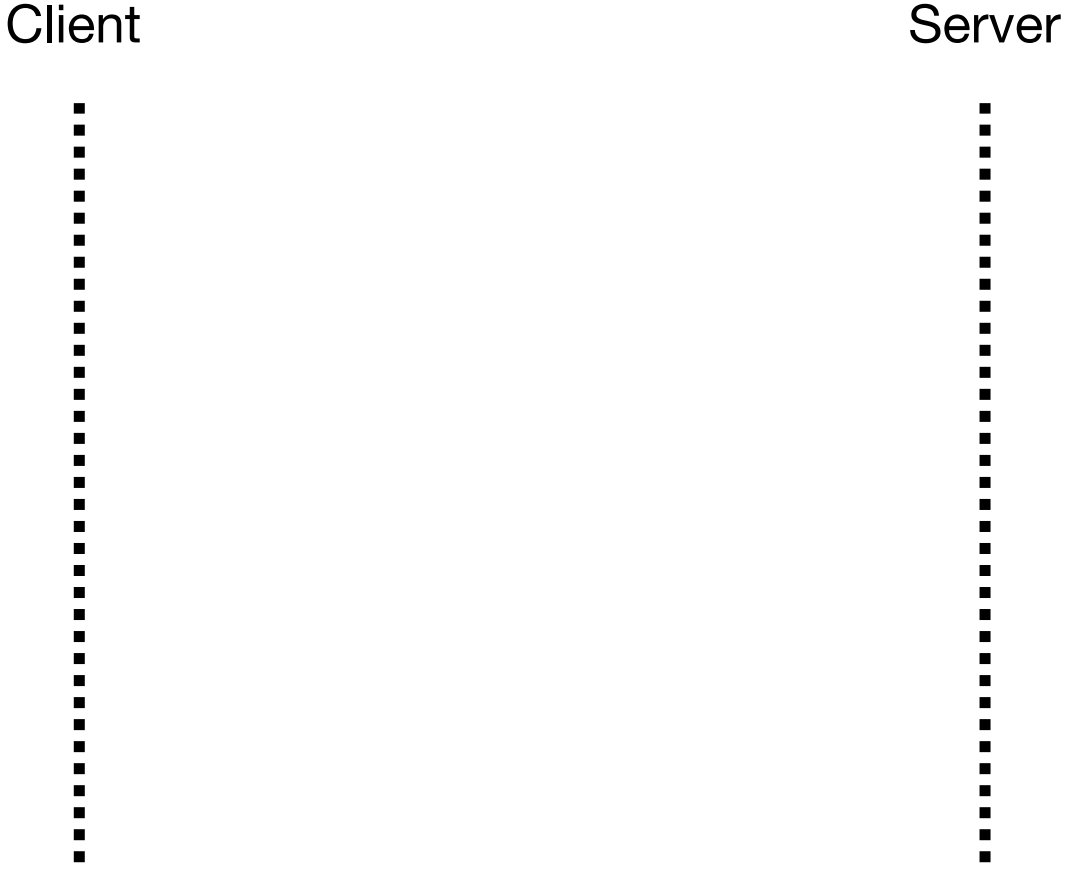
RTL handles communication between client and server. E.g.,

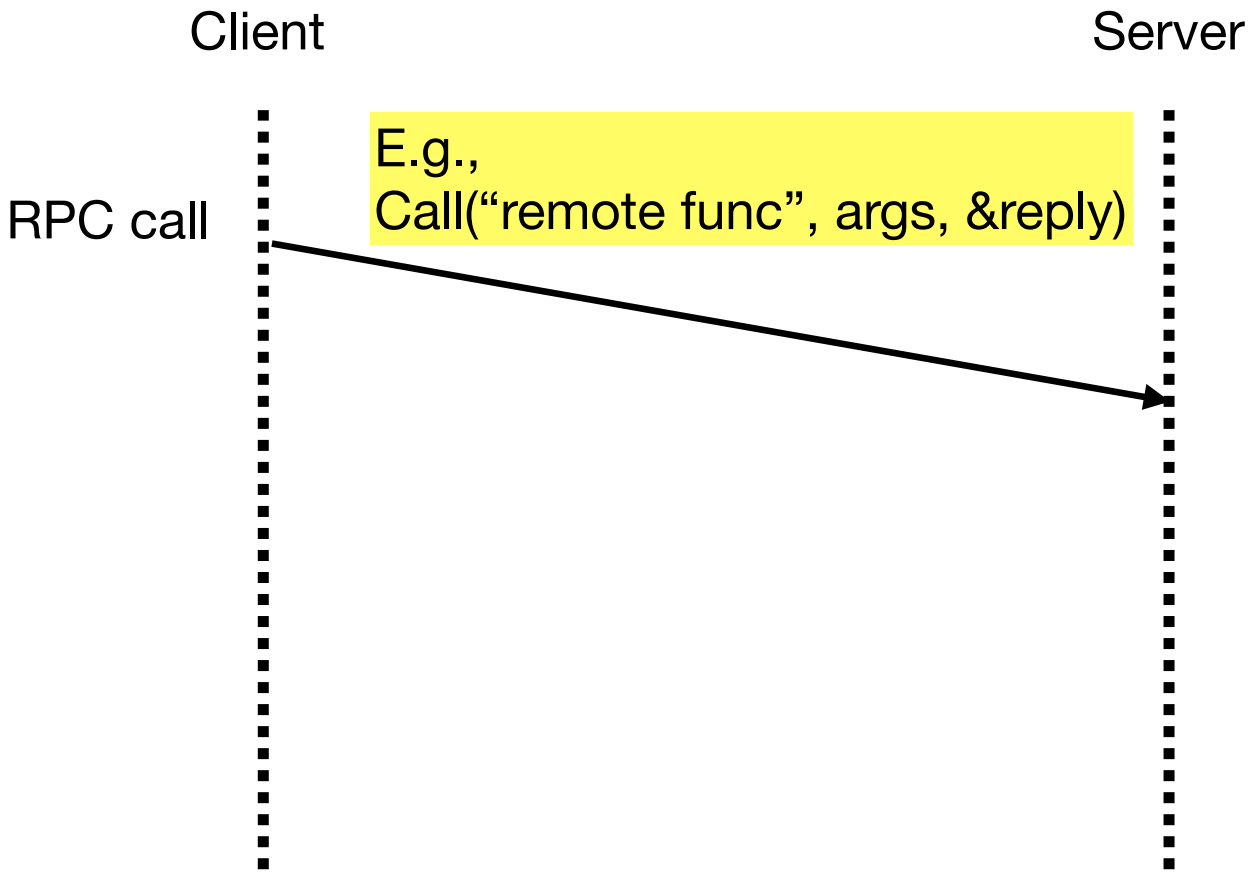
- Naming
  - E.g., hostnames and port numbers provided by internet protocols
- Transport-level protocol
  - E.g., TCP (e.g., gRPC) and UDP
- Timing

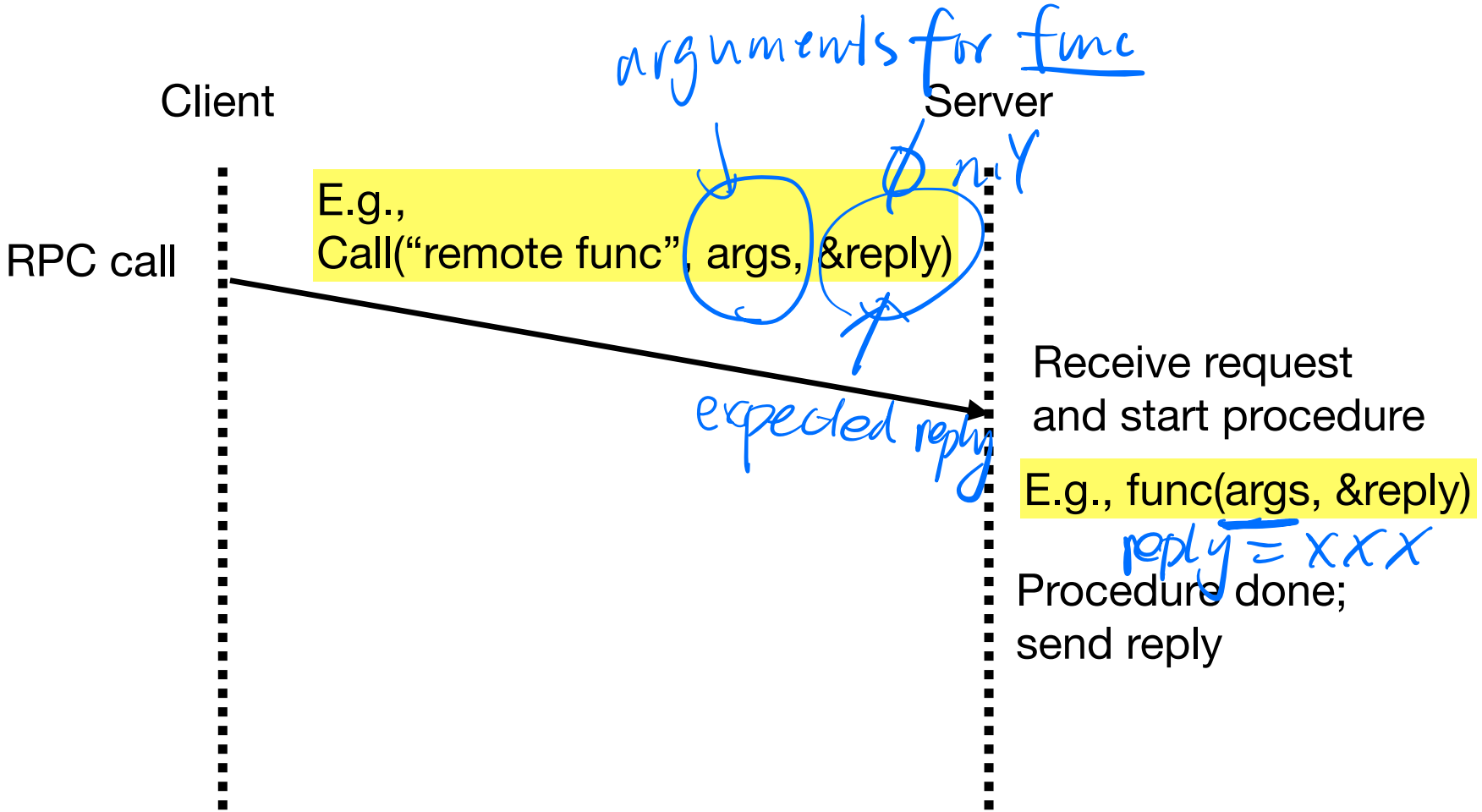
Run-time library (RTL)

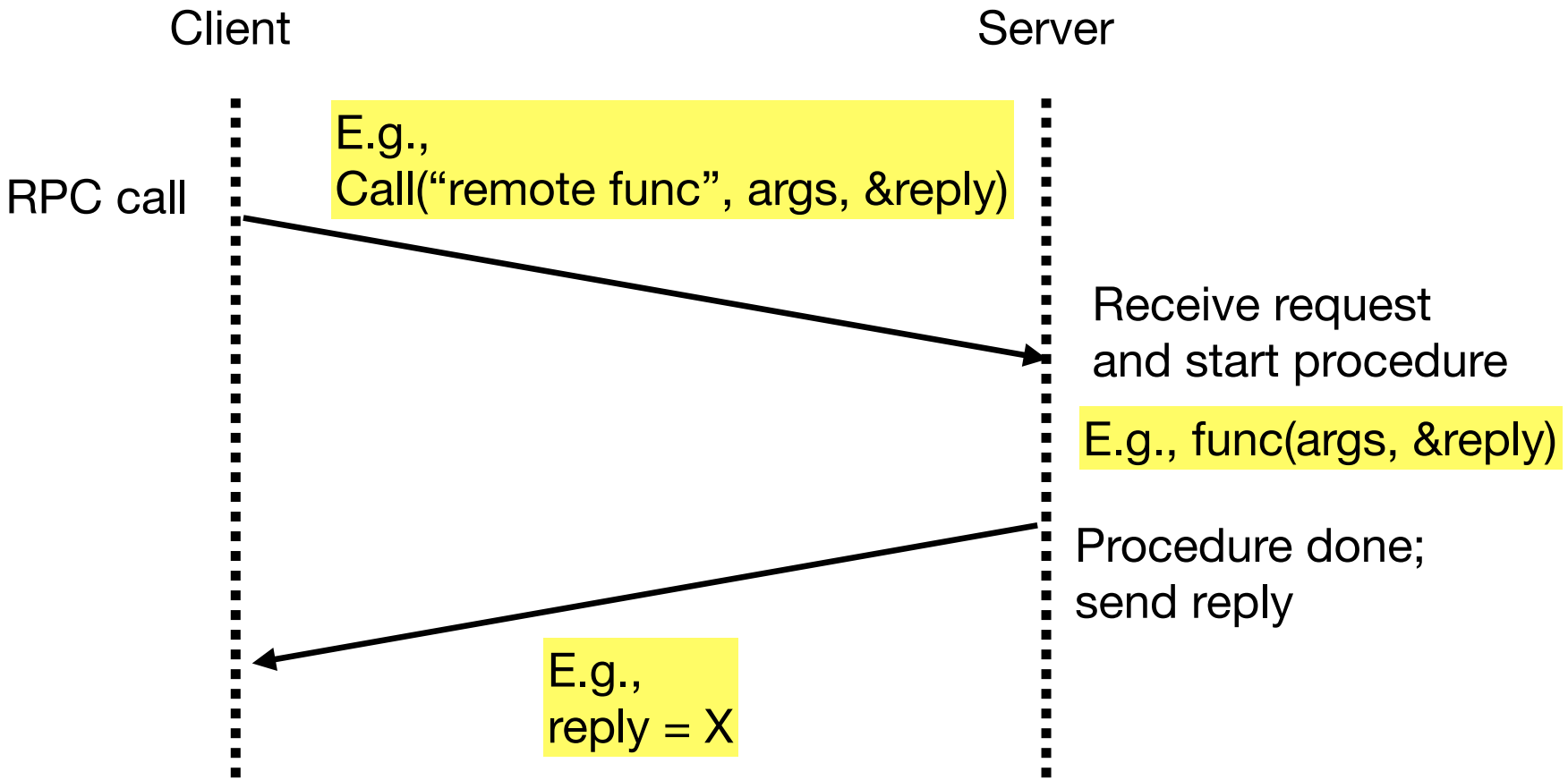
RTL handles communication between client and server. E.g.,

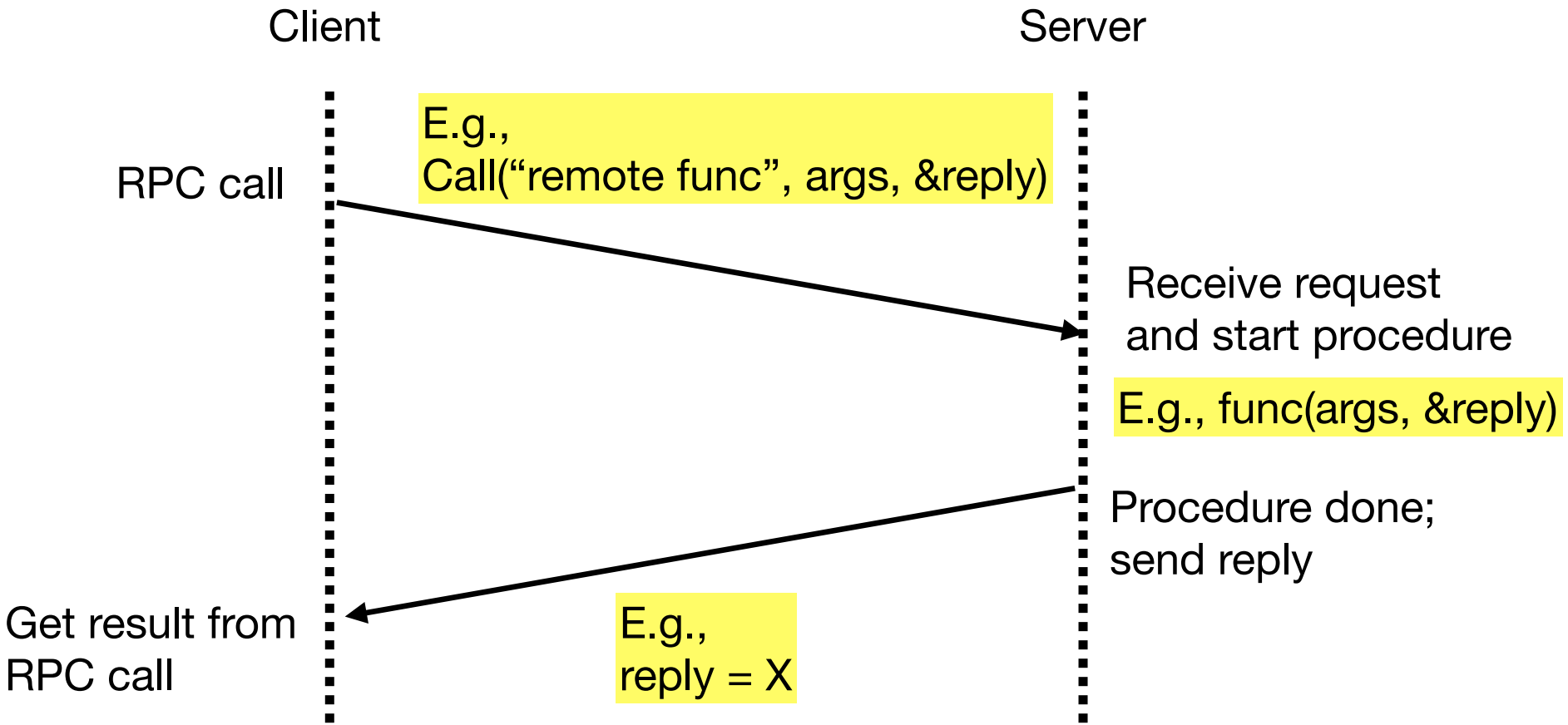
- Naming
  - E.g., hostnames and port numbers provided by internet protocols
- Transport-level protocol
  - E.g., TCP (e.g., gRPC) and UDP
- Timing
- Large arguments (larger than a single packet)
  - E.g., fragmentation (sender) and reassembly (receiver)











Demo

client.go

server.go

go run client.go

go run server.go

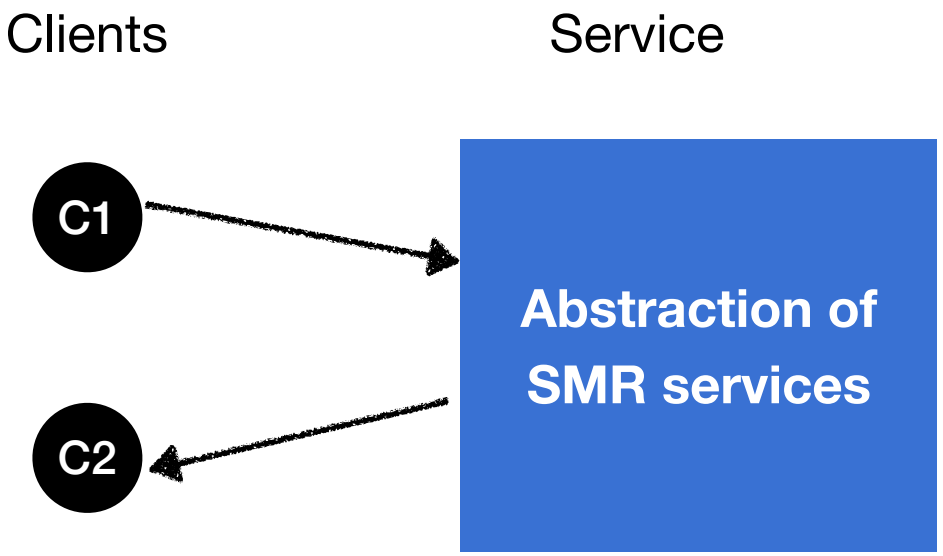
Today's outline

Remote procedure calls (RPCs)

Raft

- **Log replication**
- **Leader election**

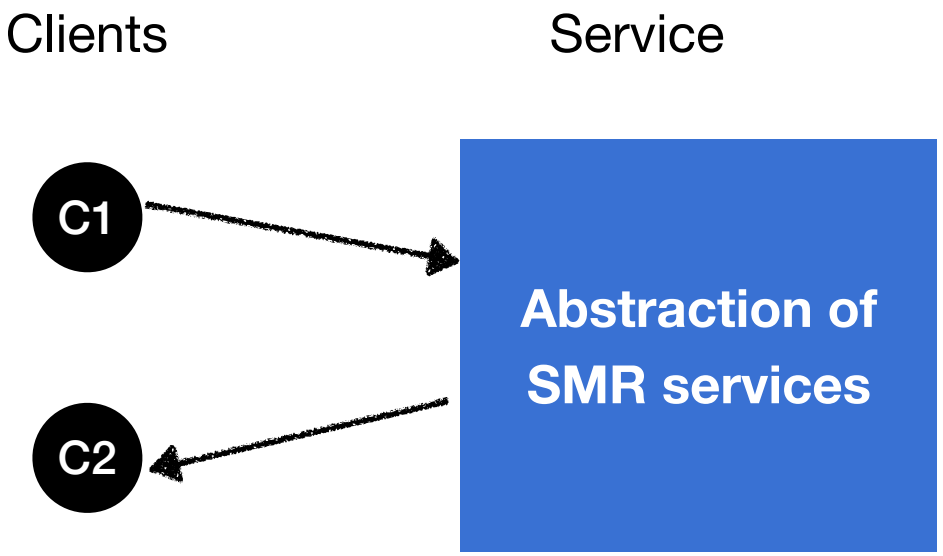
State machine replication (SMR)



SMR is a **replication service** where a set of servers compute **identical copies** of the **same state**

Recall properties of consensus

State machine replication (SMR)



SMR is a **replication service** where a set of servers compute **identical copies** of the **same state**

Recall properties of consensus

Safety: No two correct nodes decide differently

Liveness: Nodes eventually decide

Raft

- Published by Diego Ongaro et al. (from Stanford) and received Best Paper Award at 2014 USENIX Annual Technical Conference

Raft

- Published by Diego Ongaro et al. (from Stanford) and received Best Paper Award at 2014 USENIX Annual Technical Conference
- Raft is a **strong leader-based** consensus algorithm
  - More understandable than Paxos (alleged)
  - Only one leader at any time
  - Tolerates non-Byzantine failures
    - E.g., server crash, packet loss, duplication, and reordering
  - Numerous applications
    - File systems, databases, cloud computing

Raft basics #1: server roles/states

- Recall server roles in Paxos
  - Proposer, acceptor, and learner
  - A server can have multiple roles at the same time

Raft basics #1: server roles/states

- Recall server roles in Paxos
  - Proposer, acceptor, and learner
  - A server can have multiple roles at the same time
- In Raft, servers may have three roles:
  - **Leader**, **follower**, and **candidate**
  - A server can operate as only **one role** at any given time
  - Under normal operation, there is **one leader** and other servers operate as followers

Raft basics #2: timers and heartbeats

- How often do you check on your family or friends?

Raft basics #2: timers and heartbeats

- How often do you check on your family or friends?

- How do followers know if the “**leader is doing well**”
  - Leader must check on follower in a given period

Raft basics #2: timers and heartbeats

- How often do you check on your family or friends?

- How do followers know if the “**leader is doing well**”
  - Leader must check on follower in a given period
- Each follower uses a timer to monitor the “health” of the leader
  - **Timer keeps counting down until follower receives a heartbeat**
  - Otherwise, timer expires, follower starts leader election

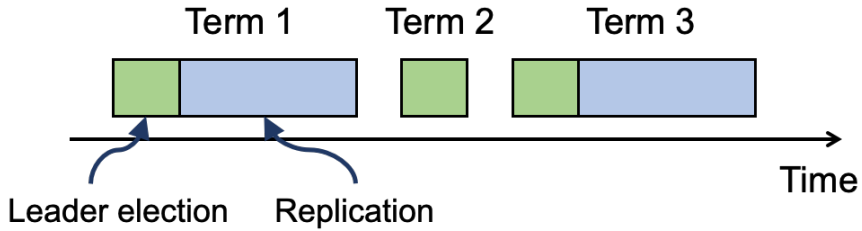
Raft basics #2: timers and heartbeats

- How often do you check on your family or friends?

- How do followers know if the “**leader is doing well**”
  - Leader must check on follower in a given period
- Each follower uses a timer to monitor the “health” of the leader
  - **Timer keeps counting down until follower receives a heartbeat**
  - Otherwise, timer expires, follower starts leader election
- Leader sends periodic heartbeats to reset followers’ timers
  - Heartbeat intervals (e.g., 50ms) << timer timeouts (e.g., 1-2s)

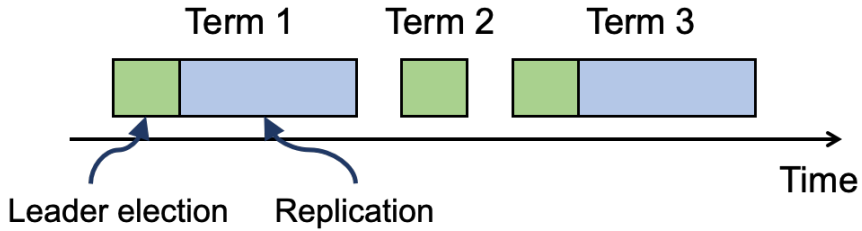
Raft basics #3: terms

- Time is divided into terms, which increase monotonically
  - Recall Lamport/logical clocks



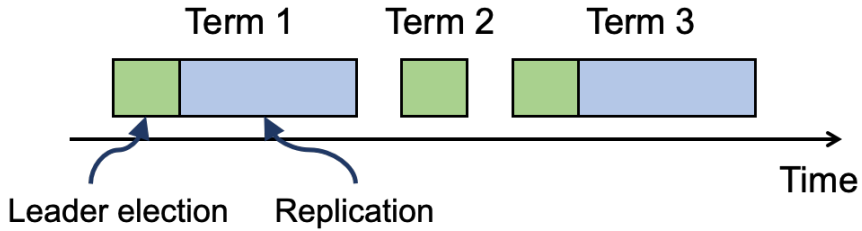
Raft basics #3: terms

- Time is divided into terms, which increase monotonically
  - Recall Lamport/logical clocks
- Terms are a local variable and act as logical clocks
  - It does not increase for all events
  - Primarily used for leader election



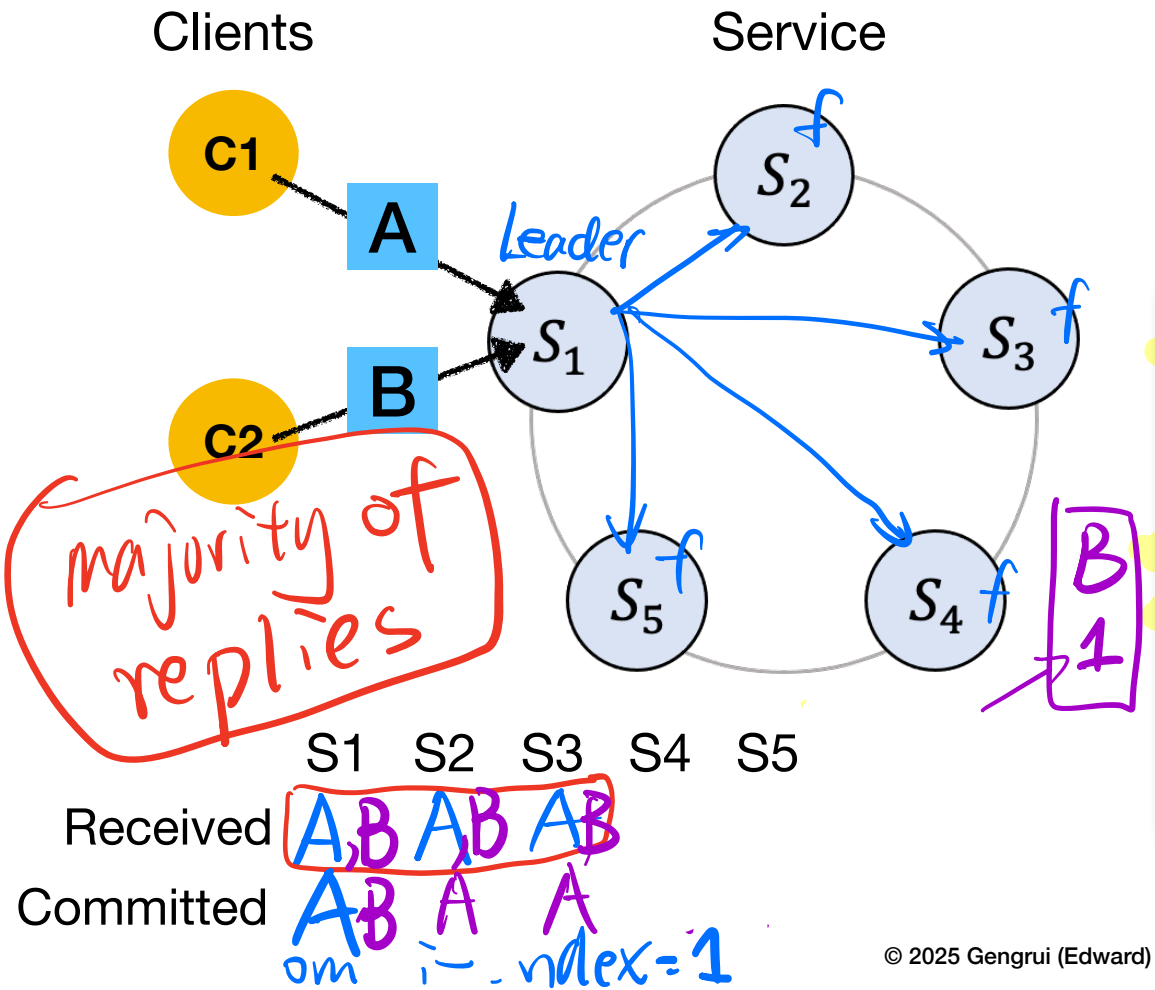
Raft basics #3: terms

- Time is divided into terms, which increase monotonically
  - Recall Lamport/logical clocks
- Terms are a local variable and act as logical clocks
  - It does not increase for all events
  - Primarily used for leader election
- A server, regardless of its operating role, always sync up to a higher term



Example 1

Raft: log replication



- S1 is leader; others are followers
- Leader issues AppendEntriesRPC

AppendEntries RPC

Arguments:

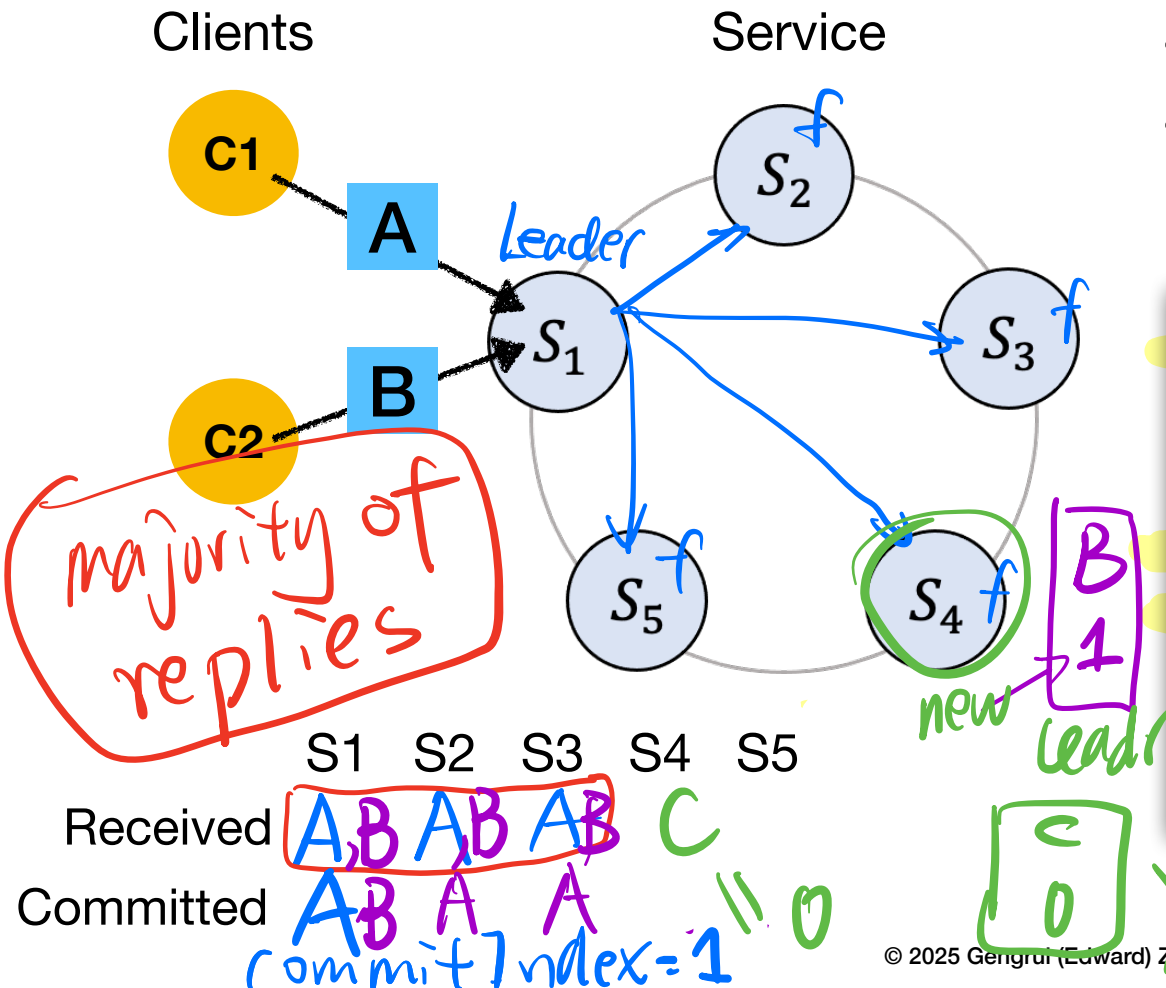
term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Example 1

Raft: log replication



- S1 is leader; others are followers
- Leader issues AppendEntriesRPC

AppendEntries RPC

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Raft: log replication

- Strong leadership
  - Log entries flow only from the leader to followers
  - Followers must synchronize its log according to leader's log

- Quorum replication
  - In a system of $n = 2f + 1$ servers, consensus is reached when $f + 1$ servers commit
  - A minority of slow servers ($\leq f$) do not impact overall replication performance

Failures in Raft

- We’ve seen how Raft efficiently replicate log entries under normal case
- Now let’s discuss what will happen under failures

Failures in Raft

- We've seen how Raft efficiently replicate log entries under normal case
- Now let's discuss what will happen under failures
- Followers fail
  - How many followers are allowed to fail?

f # of failures

$n = 2f + 1$

$f = 2 \rightarrow n = 5$

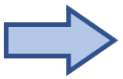
Failures in Raft

- We’ve seen how Raft efficiently replicate log entries under normal case
- Now let’s discuss what will happen under failures
- Followers fail
    - How many followers are allowed to fail?
- Leader fails
    - What do we need from a new leader?

Failures in Raft

- We've seen how Raft efficiently replicate log entries under normal case
- Now let's discuss what will happen under failures
- Followers fail
  - How many followers are allowed to fail?
- Leader fails
  - What do we need from a new leader?

A new leader should have
- **the highest term value**
- **the most up-to-date log**



Make sure the system **never falls back** to a previous state; i.e., not loosing log entries when leadership changes

"view stamped replication" "PBFT" "HotStuff"

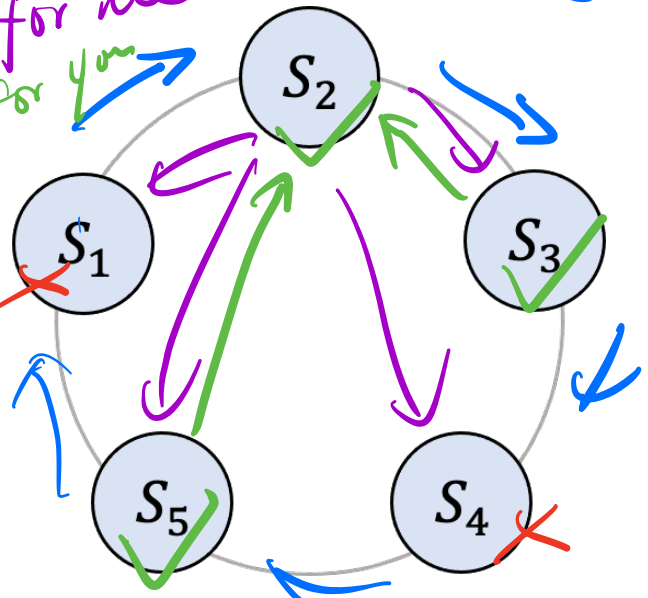
Solution 1: Passive leadership rotation

- Pre-define a leadership schedule
  - $leaderID = term \mod n$
- Write an algorithm

1 mod 5 = 1
2 mod 5 = 2

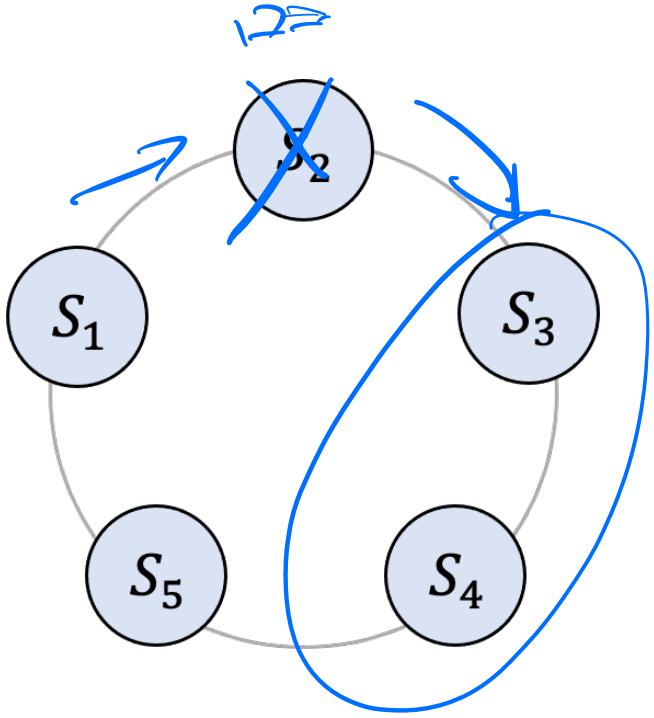
upon timeout (whole process)!
term = term + 1
leaderID = term mod n
SEND "your turn" to leaderID

SEND "vote for me" with it's logIndex
when receive "vote for me" from leaderID
SEND "I vote for you"
receive a majority replies → become the new leader

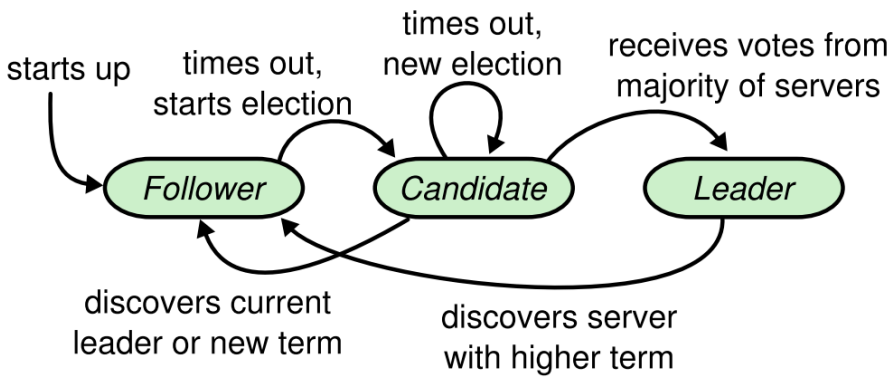


Solution 1: Passive leadership rotation

- Pre-define a leadership schedule
  - $leaderID = term \mod n$
- Pros:
  - Simple; easy to implement
- Cons:
  - Cannot avoid already crashed servers
  - Cannot avoid slow servers

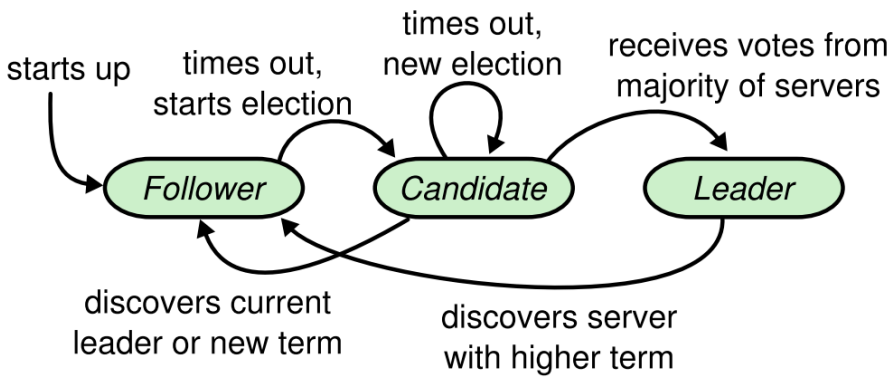


Raft's leader election



- Instead of passively rotate leadership, Raft enables followers who detect a leader's failure to actively **campaign for leadership**
  - Crashed servers will not start a campaign
  - Slow servers will not win

Raft's leader election



- Instead of passively rotate leadership, Raft enables followers who detect a leader's failure to actively **campaign for leadership**
  - Crashed servers will not start a campaign
  - Slow servers will not win
- Properties we need to guarantee:
  - **At most one leader** is elected in a given term
  - Elected leader must have **most up-to-date log**
  - Elected leader must be in the **highest term**

Raft's leader election

Upon a timeout // timer resets and keeps going

1. Transition from follower to candidate
2. Increment term
3. Issue RequestVote RPCs
4. Vote for itself

   // wait for a majority of votes
5. Majority of votes received before timeout?
   become new leader : go back to 1. and repeat

RequestVote RPC

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Raft's leader election

Upon a timeout // timer resets and keeps going

1. Transition from follower to candidate
2. Increment term
3. Issue RequestVote RPCs
4. Vote for itself

   // wait for a majority of votes
5. Majority of votes received before timeout?
   become new leader : go back to 1. and repeat

Discovers current leader or higher term?

Go back to follower

RequestVote RPC

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Voters: how should I vote for a candidate?

A server votes for the candidate if

1. Candidate's term $\geq$ its own term
2. It has not voted yet in this term
3. Candidate's log is at least as up-to-date as its log

A server ONLY votes ONCE in a given term

⇓

at most one leader can be elected in a given term

Example 1

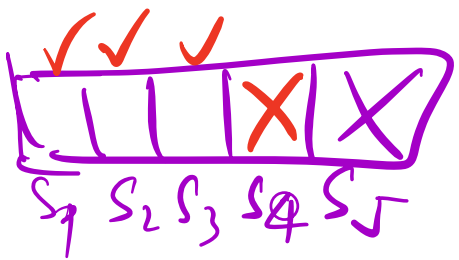
Upon a timeout // timer resets and keeps going

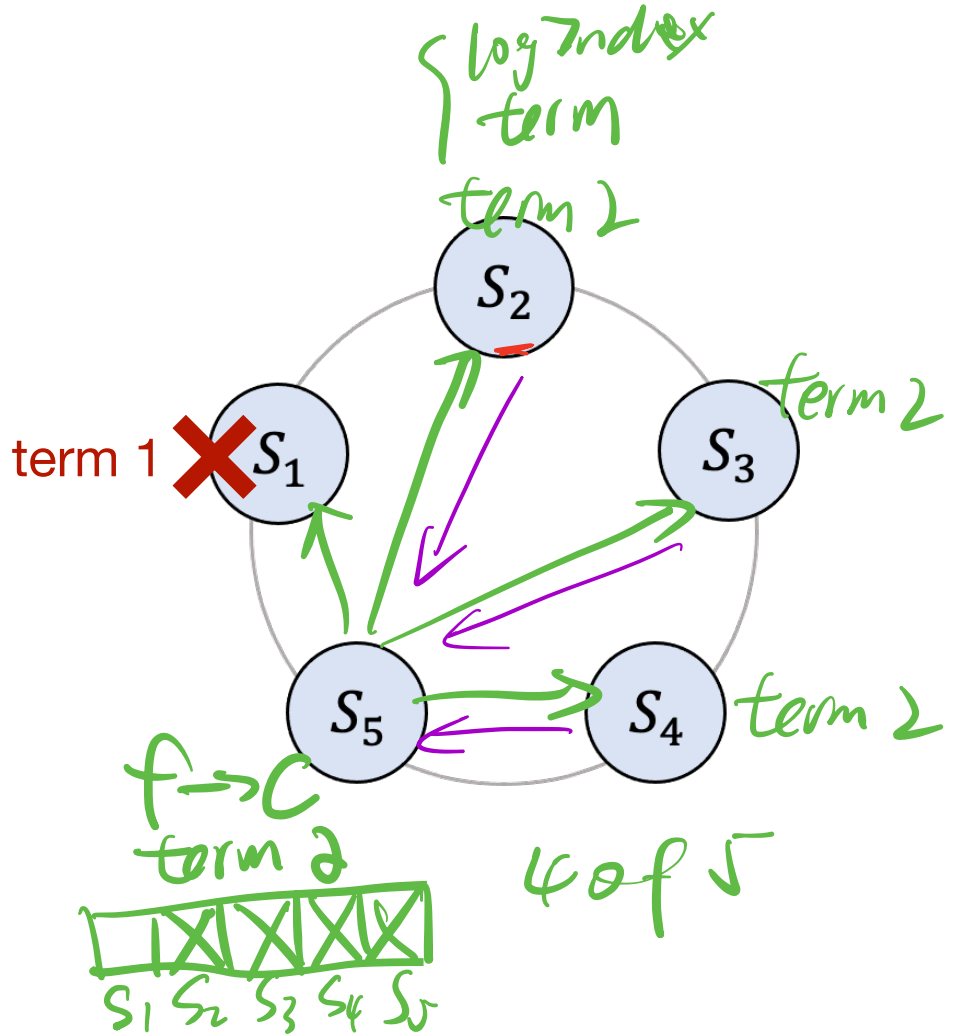
1. Transition from follower to candidate
2. Increment term
3. Issue RequestVote RPCs
4. Vote for itself

   // wait for a majority of votes
5. Majority of votes received before timeout?
   become new leader : go back to 1. and repeat

A server votes for the candidate if

1. Candidate's term ≥ its own term
2. It has not voted yet in this term
3. Candidate's log is at least as up-to-date as its log





Example 2

Split votes

Upon a timeout // timer resets and keeps going

1. Transition from follower to candidate
2. Increment term
3. Issue RequestVote RPCs
4. Vote for itself

   // wait for a majority of votes

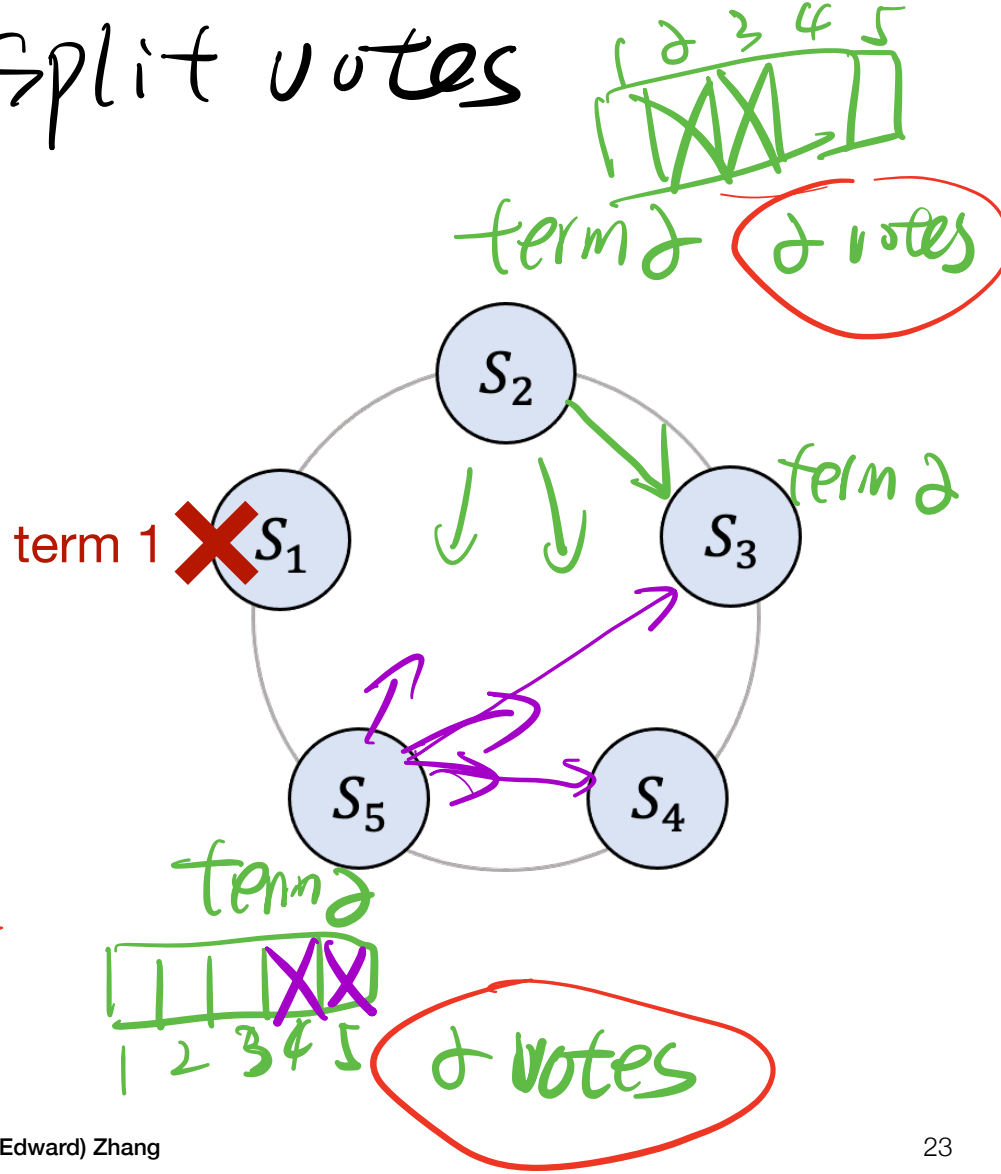
5. Majority of votes received before timeout?
   become new leader : go back to 1. and repeat

A server votes for the candidate if

1. Candidate's term ≥ its own term
2. It has not voted yet in this term
3. Candidate's log is at least as up-to-date as its log

① randomized timeouts

② wait (hope)



Summary

- Raft operates in a succession of terms
  - Leader election
  - Replication

- Raft is fast and efficient
  - Under normal operation, consensus is achieved by one round of RPCs
  - Strong leadership: followers synchronize to leader
  - Leader election mechanism allows servers to proactively campaign for leadership

Worksheet