CS 582: Distributed Systems

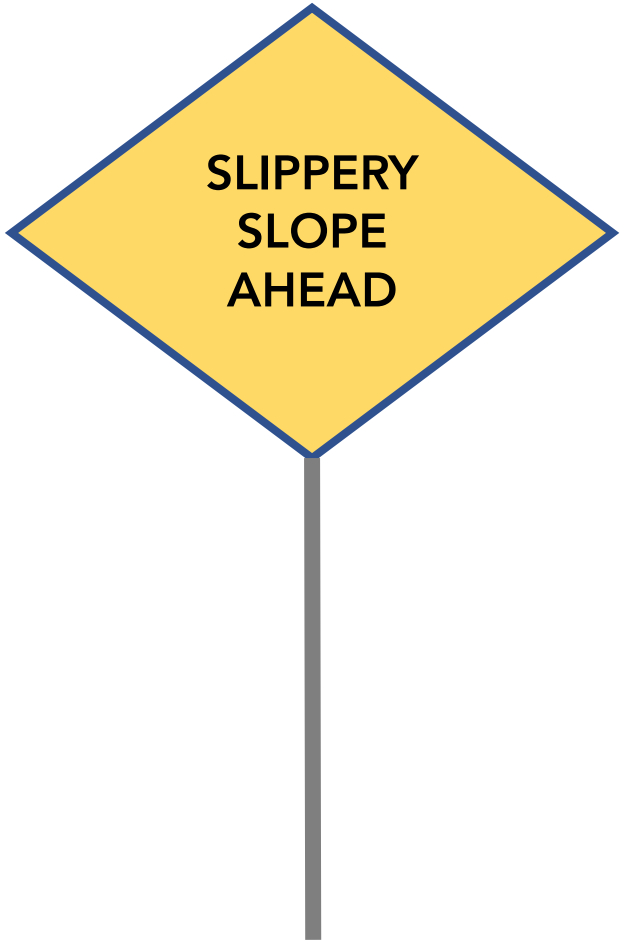
Vector Clocks & Leader Elections

Dr. Zafar Ayyub Qazi

Fall 2024

Acknowledgements: Slides use some material from Ryan Stutsman (University of Utah), Wyatt Lloyd (Princeton University) and Martin Kleppmann (University of Cambridge)

This lecture will have a few slippery slopes



Today's Agenda

- Application of Vector Clocks
  - Causally-ordered Slack-like application
- Leader Elections
- Leader Elections in Raft
  - Quick overview of Raft
  - Leader election algorithm

Specific learning outcomes

By the end of today's lecture, you should be able to:

- ❑ Apply Vector clocks to provide causally ordered communication
- ❑ Describe the leader election problem
- ❑ Explain how leader elections are conducted in Raft
- ❑ Analyze leader election algorithm in Raft for safety and liveness

Recap: Motivation for Vector Clocks

- How can we design logical clocks that can allow us to infer potential causality?
    - More precisely, whether a → b?

Recap: Vector Clocks

- A Vector Clock is a vector of integers, one entry for each process in the system

- Label each event e with a vector V(e)= <$c_{1,}c_{2,}$,…, c_n>
  - Where c_i is the count of events in process i that "happen-before" e
  - And we have n processes in the system

- Initially, all vectors are <0, 0, …, 0>

Recap: Rules for updating Vector Clocks

- Local update rule:
  - For each local event on process i, increment local entry c_i (by 1)
- Message rule:
  - If process j receives message with vector <d_1, d_2, …, d_n>:
    - Set each local entry $c_k = \max\{c_k, d_k\}$
    - Increment local entry c_j (by 1)

Vector Clock Orderings

- Rules for comparing vector timestamps:
  - $V(a) = V(b)$ iff $a_k = b_k$ for all k
  - $V(a) < V(b)$ iff $a_k \leq b_k$ for all k and $V(a) \neq V(b)$
  - $V(a) \,||\, V(b)$ iff $a_i < b_i$ and $a_j > b_j$, for some i, j
- Properties of this order
  - $(V(a) < V(b)) \Leftrightarrow (a \rightarrow b)$
  - $(V(a) = V(b)) \Leftrightarrow (a = b)$
  - $(V(a) \,||\, V(b)) \Leftrightarrow (a \,||\, b)$

Vector Clocks Capture Potential Causality

- V(w) < V(z) then there is a chain of events linked by

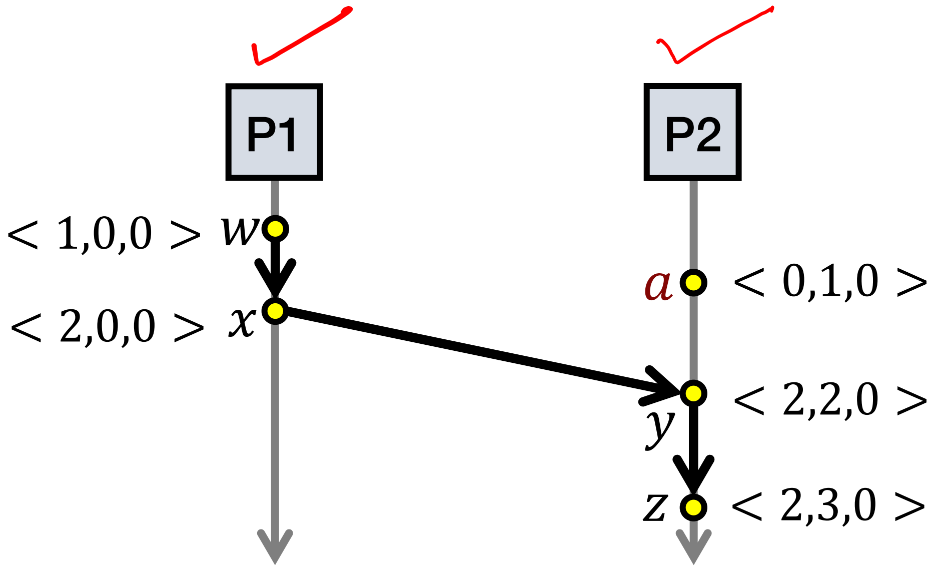
  happens-before (→) between w and z

Vector Clocks Capture Potential Causality

- V(w) < V(z) then there is a chain of events linked by

  happens-before (→) between w and z

- V(a) || V(w) then there is no such chain of events between a and w



$V(a) = \langle 0, 1, 0 \rangle$

$V(w) = \langle 1, 0, 0 \rangle$

$V(w)[1] > V[a][1]$

$V(a)[2] > V(w)[2]$

Two events **a, z**

Lamport Clocks: **C(a) < C(z)**

Conclusion: **None**

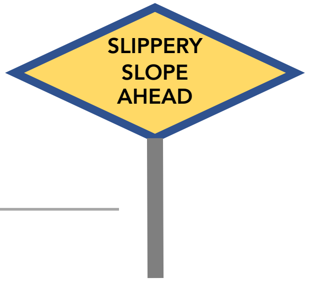
a → b or a || z

Vector Clocks: **V(a) < V(z)**

Conclusion: **a→ …→ z**

Vector clock timestamps precisely capture the happens-before relation

Slack-Like Application

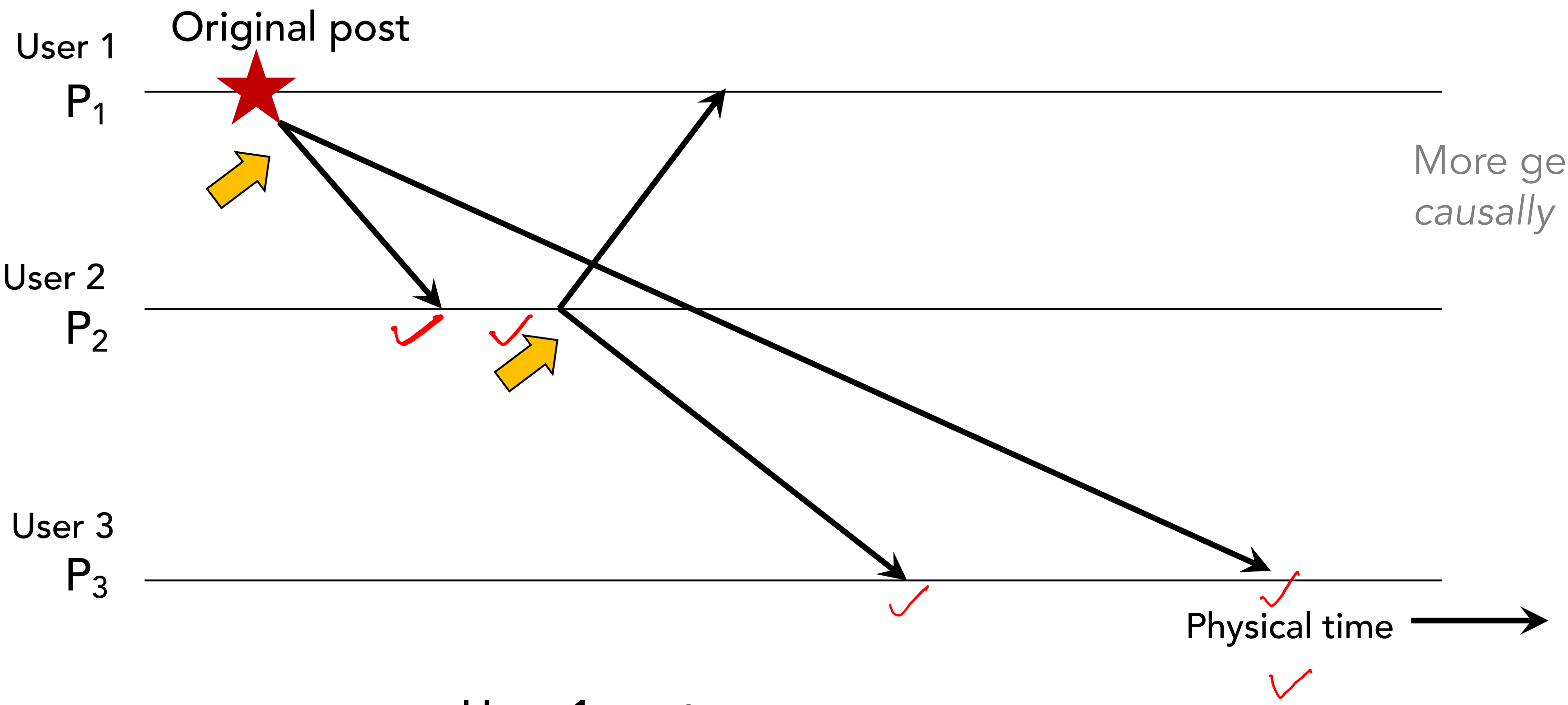


- Slack-like application
  - Broadcast posts to all other users

- **Goal:** No user should *display* a response post before the corresponding original message post

- Deliver a message only after all messages that might causally precede it have been delivered
  - Otherwise, the user would see a reply to a message they could not find

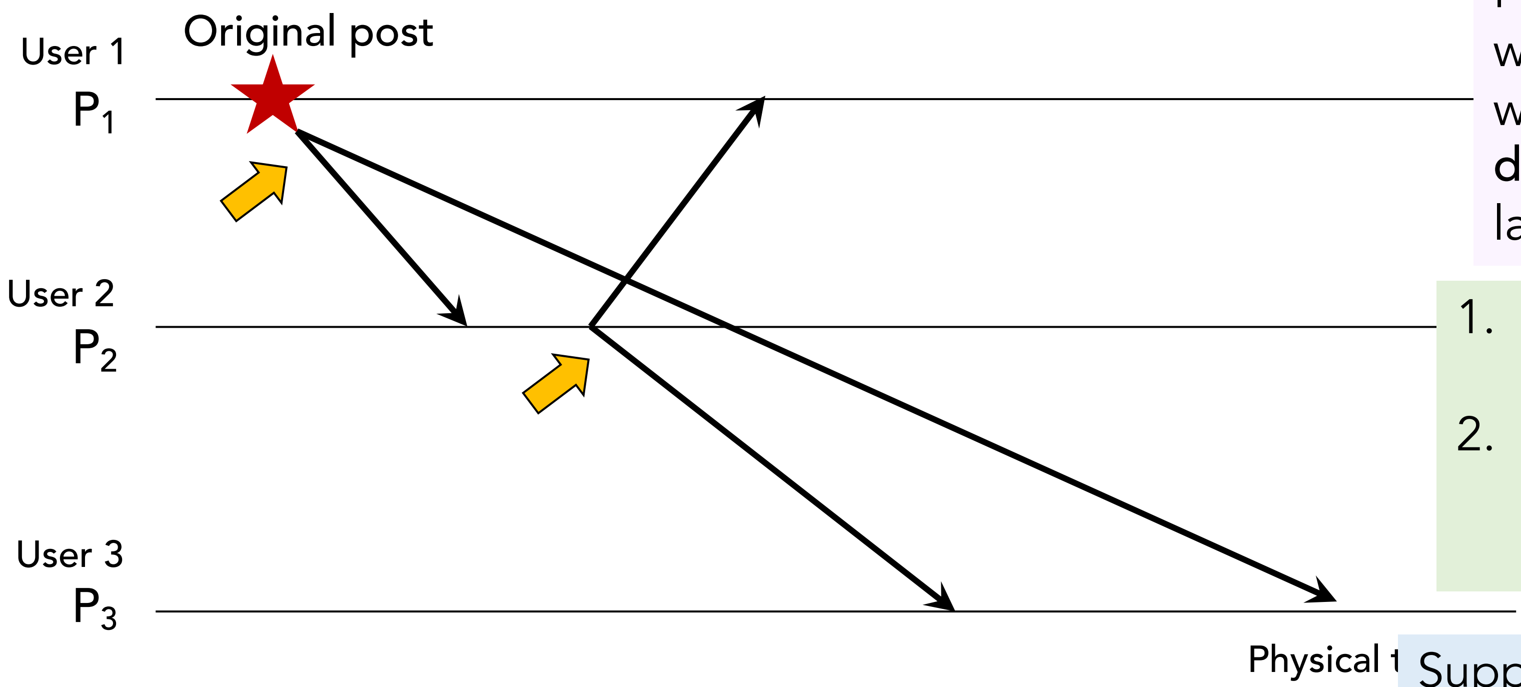
Causally-ordered Slack-like App



More generally this is referred to as providing *causally ordered communication*

- User 1 posts
- User 2 replies to User 1's post
- User 3 observes

Causally-ordered Slack-like App



For enforcing causal message delivery, we assume clocks are adjusted only when sending and **delivering** messages **delivering** messages to application layer

1. Upon sending a message, process P_i will increment $\mathbf{V_i[i]}$ by 1
2. When it delivers a message m with timestamp **ts(m),** it only adjusts $\mathbf{V_i[k]}$ **to** $\mathbf{\max\{V_i[k], ts(m)[k]\}}$ for each k

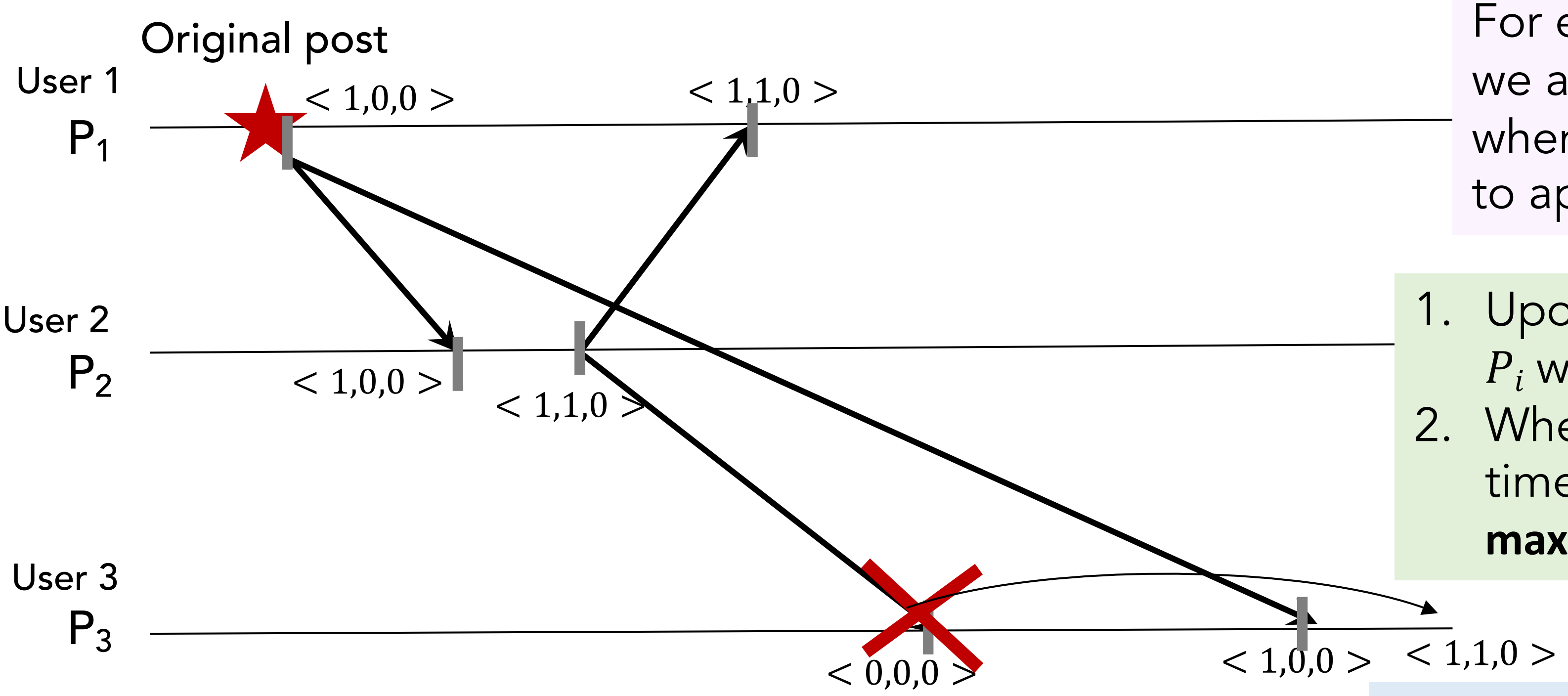
- User 1 posts
- User 2 replies to User 1's post
- User 3 observes

Suppose that P_j receives a message m from P_i. Delivery of the message to the application layer of a process P_j is delayed until the following two conditions are met:

1. $\mathbf{ts(m)\ [i] = V_j[i] + 1}$
2. $\mathbf{ts(m)\ [k] <= V_j[k]}$ **for all** $\mathbf{k \neq i}$

**Read note 6.4 (page 320) of the course textbook (by Tanenbaum et al.) for a detailed explanation of this example.*

Causally-ordered Slack-like App



- User 1 posts
- User 2 replies to User 1's post
- User 3 observes

For enforcing causal message delivery, we assume clocks are adjusted only when sending and **delivering** messages to application layer

1. Upon sending a message, process P_i will increment **V$_i$[i]** by 1
2. When it delivers a message m with timestamp **ts(m),** it only adjusts **V$_i$[k] to max{V$_i$[k], ts(m)[k]}** for each k

Suppose that P_j receives a message m from P_i. Delivery of the message to the application layer of a process P_j is delayed until the following two conditions are met:

1. **ts(m) [i] = V$_j$[i] + 1**
2. **ts(m) [k] <= V$_j$[k] for all k ≠ i**

**Read note 6.4 (page 320) of the course textbook (by Tanenbaum et al.) for a detailed explanation of this example.*

Leader Elections

Why Election?

- Often need one process to play a special role
  - Coordinator, Initiator, etc.
  - Useful for coordination among distributed servers
  - E.g., pick a server node in Berkeley algorithm
- How to select a leader if you have multiple candidates?
- Leader Election widely used in industry
  - Apache Zookeeper, Google's Chubby, Google Spanner database, Network Time Protocol, Berkeley Algorithm, Raft, etc.

Leader Election Problem

- In a group of processes, elect a Leader to undertake special tasks
    - And let everyone in the group know about this Leader

- What happens when a leader fails (crashes)?
    - Some process detects this (e.g., using timeouts)
    - Then what?

- Need to start elections to elect a leader
    - An election algorithm defines how elections are conducted

Goals of an Election Algorithm

1. Elect only one leader among the non-faulty processes

2. All non-faulty processes agree on who the leader is

Requirements

- Any process can call for an election
- A process can call for at most one election at a time
- Multiple processes are allowed to call an election simultaneously
    - All of them together must yield only a single leader

Leader Elections in Raft

Short overview of Raft

- We will come back and study Raft in detail a couple of weeks later

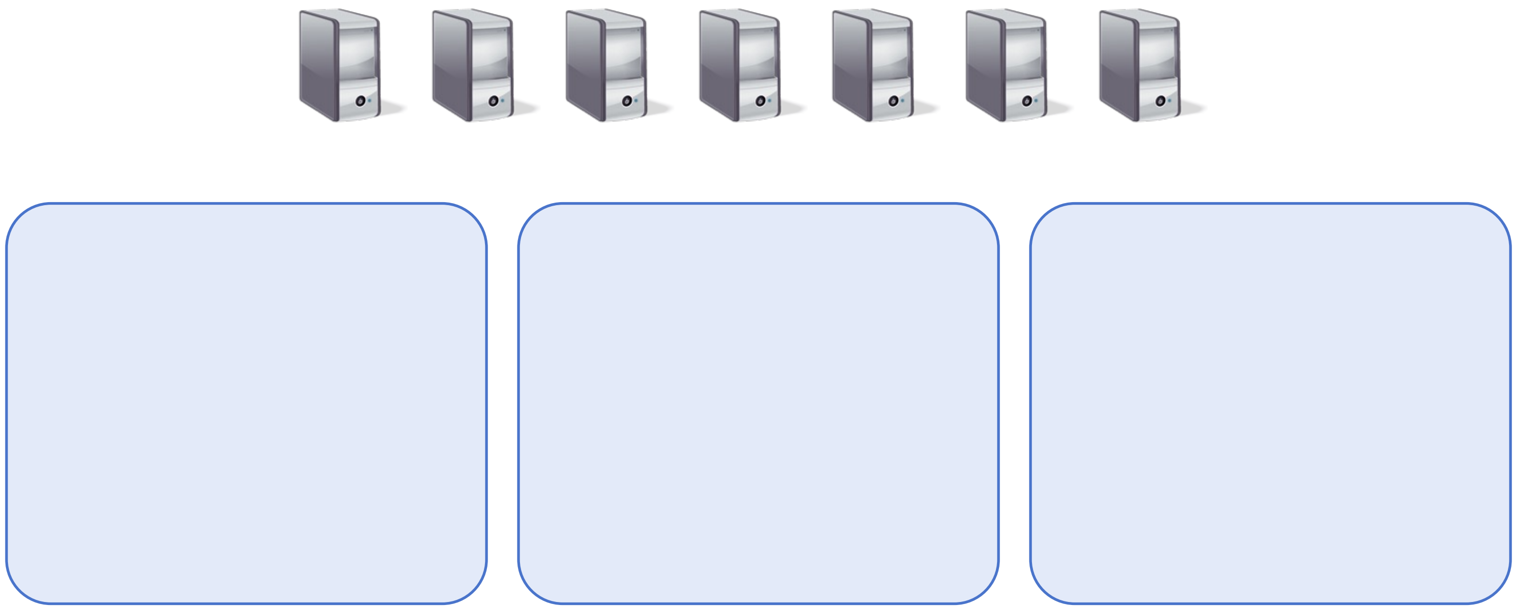
What is Raft?

- Raft is a distributed consensus algorithm
- Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members
  - Play a key role in building large-scale distributed software systems
- Raft is deployed by several companies
- You will implement it in assignments 2-4

Replicated State Machines

- Consensus algorithms typically arise in the context of replicated state machines

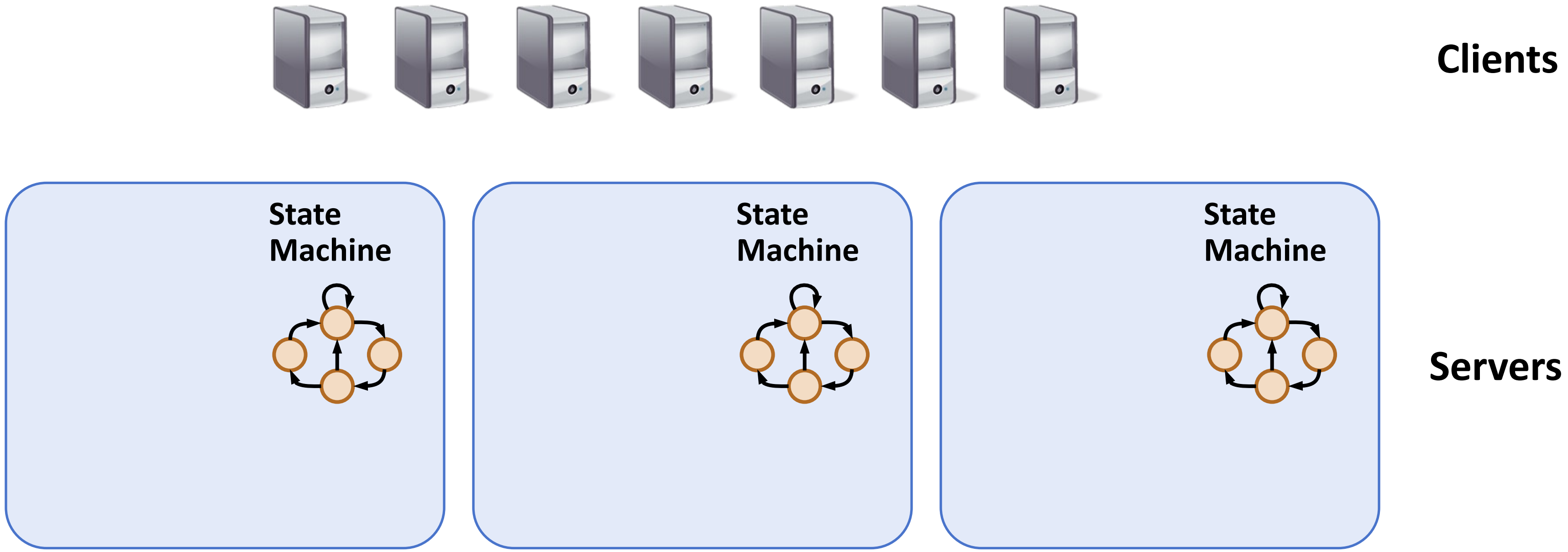
Replicated State Machines



Clients

Servers

Replicated State Machines



Clients

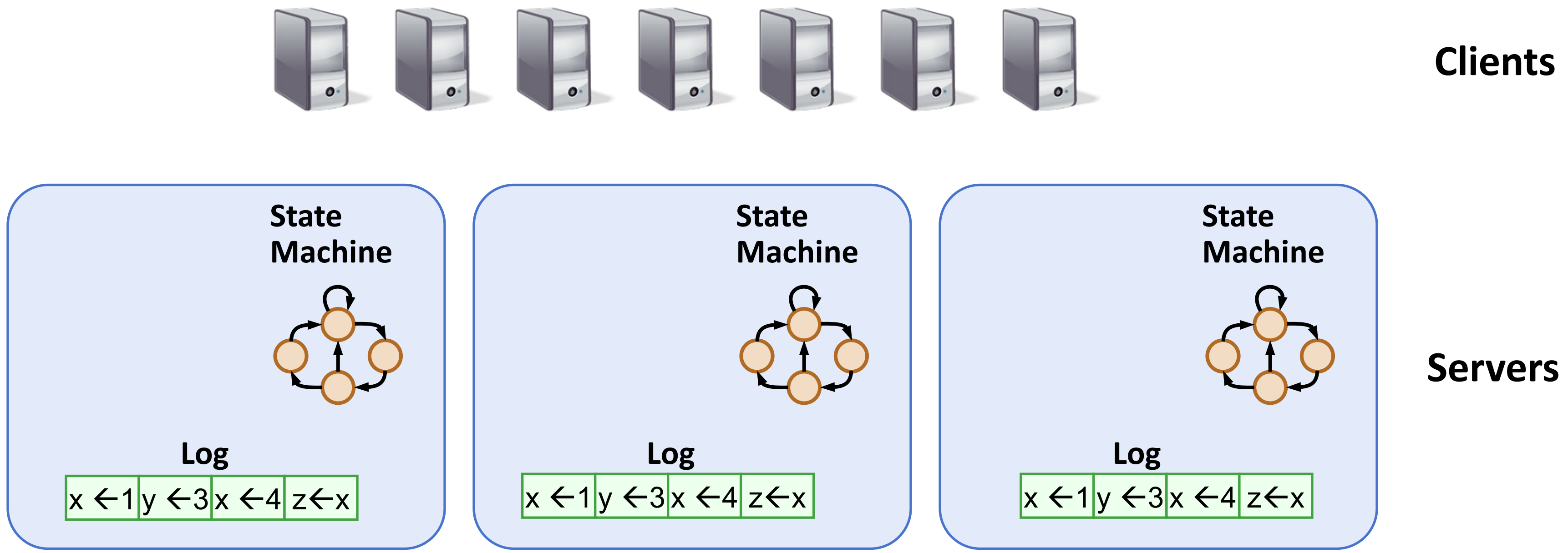
State Machine

State Machine

State Machine

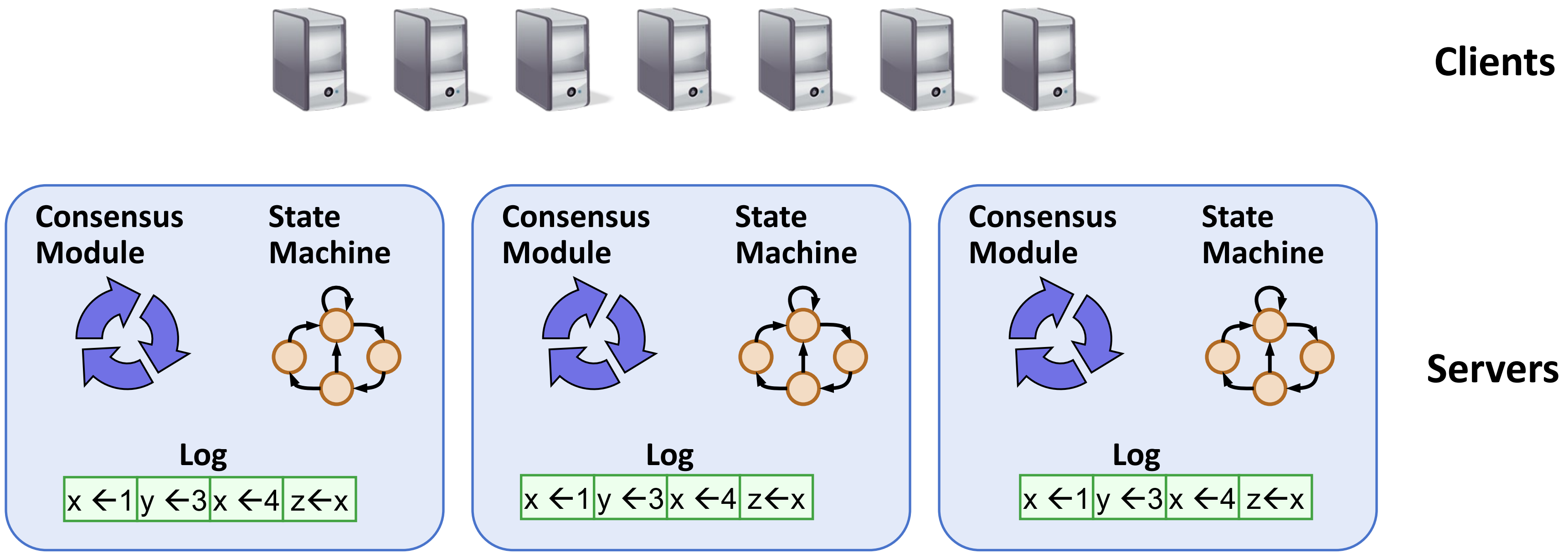
Servers

Raft → Replicated Log



- Replicated log => replicated state machine
  - All servers execute same commands in same order

Raft → Replicated Log



- Replicated log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System should make progress as long as any majority of servers are up
- Failure model: fail-stop (not Byzantine), delayed/lost messages

Raft has multiple components

- Leader Election
- Log Replication
- Safety
- Membership change

Raft servers can be in three states

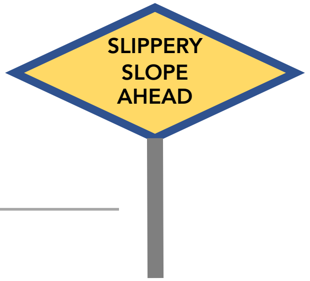
Follower	Only responds to incoming requests
Candidate	Used to elect a new leader
Leader	Handles all client interactions, log replication

Servers use RPCs

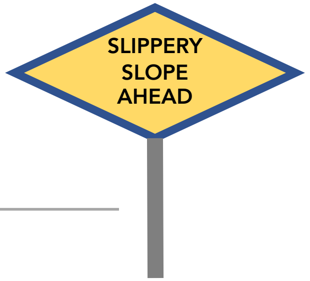
- RequestVote RPCs
  - Sent when a candidate is requesting votes to become a leader

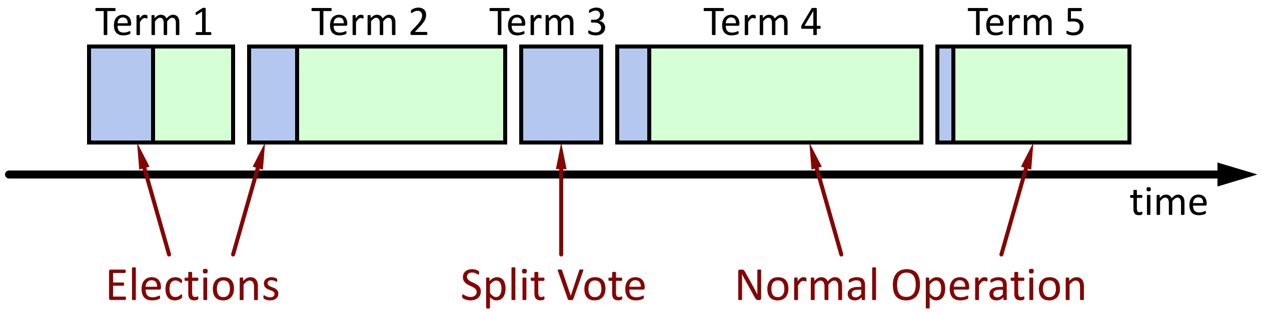
- AppendEntries RPCs
  - To send heartbeat messages: to let other servers know "I am up"
  - Another purpose is log replication, which we will not discuss today

Time is divided into Terms



Time is divided into Terms





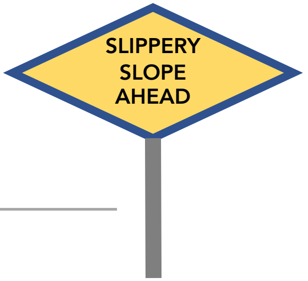
- Time divided into terms:
  - Election
  - Normal operation under a single leader
- At most 1 leader per term
- Some terms have no leader (failed election)
- Each server maintains current term value (no global view)
  - Exchanged in every RPC
  - Peer has later term? Update term, revert to follower
  - Incoming RPC has obsolete term? Reply with error

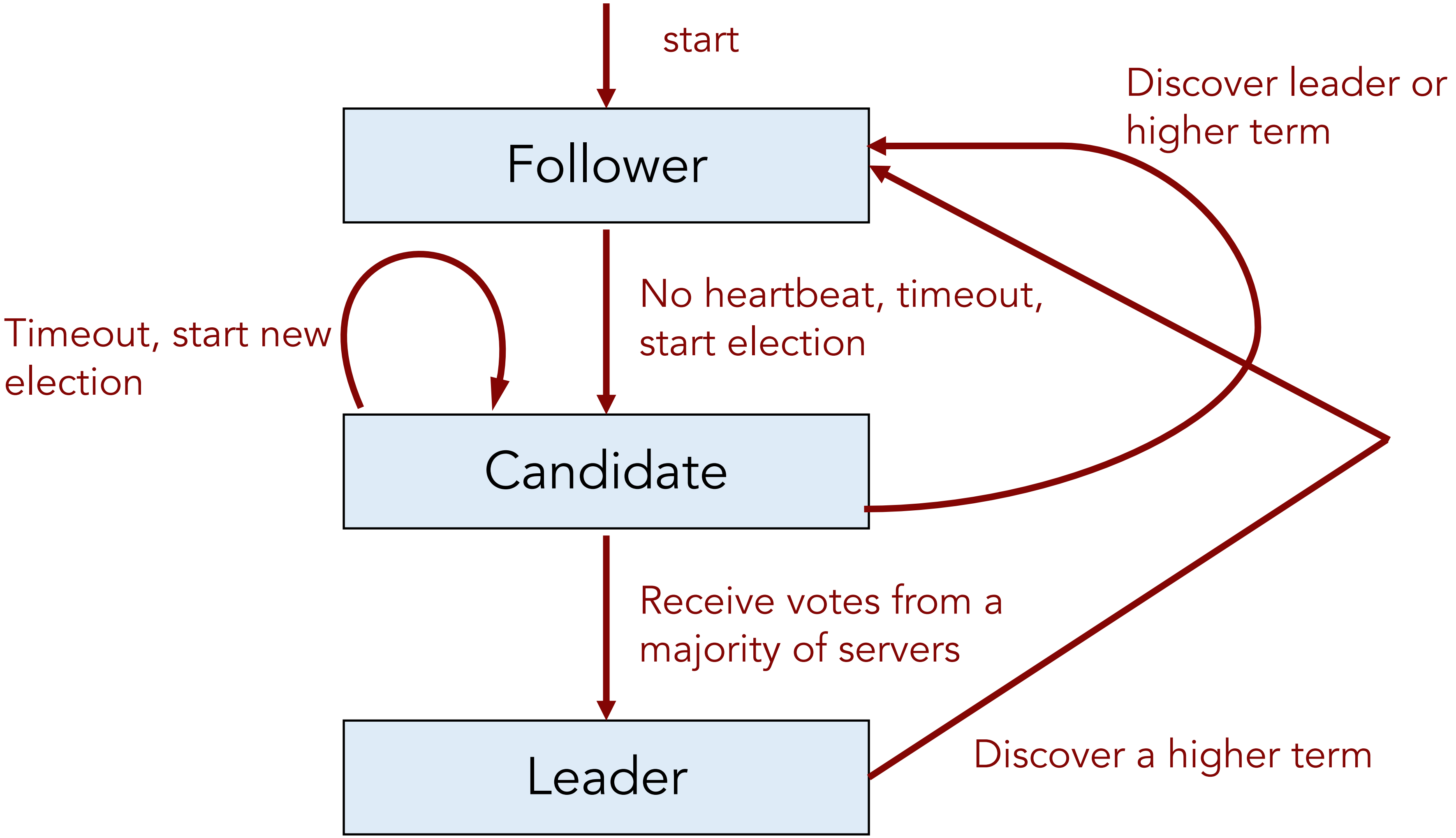
How are elections conducted in Raft?

Election Basics

- If a server suspects a leader has failed, increment current term
- Change to Candidate state
- Vote for self
- Send RequestVote RPCs to all other servers, retry until either:
  1. Receive votes from majority of servers:
     - Become leader
     - Send AppendEntries heartbeats to all other servers
  2. Receive RPC from a valid leader:
     - Return to follower state
  3. No-one wins election (election timeout elapses):
     - Increment term, start new election

Server State Transitions





start

Follower

No heartbeat, timeout, start election

Timeout, start new election

Candidate

Discover leader or higher term

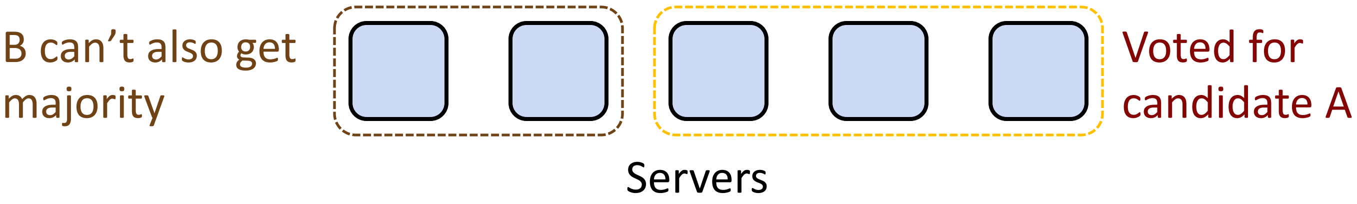
Receive votes from a majority of servers

Leader

Discover a higher term

Election Correctness

- Safety: allow <u>at most one winner per term</u>
  - Each server gives out only one vote per term (persist on disk)
  - Two different candidates can't accumulate majorities in same term



- Liveness: some candidate must eventually win
  - Choose election timeouts randomly in [T, 2T](e.g., 150-300 ms)
  - One server usually times out and wins election before others wake up
  - Works well if T >> broadcast time

Summary: Leader Election in Raft

- Three possible server states: follower, candidate, leader
- All servers startup as followers
- If a follower detects a leader has failed
    - Follower can start new election
    - Increments term, and transiton to Candidate state
    - Candidate becomes a leader if a majority of processes vote for it
    - Possible no one wins election: split votes

For more on leader elections in Raft …

- Attend the next tutorial
- Check out Raft's extended paper (will be shared with the assignment 2 handout)
- Checkout the following excellent visualization of elections in Raft:
    - http://thesecretlivesofdata.com/raft/

Next Lecture

- Replication and Consistency
  - Consistency Models
  - CAP