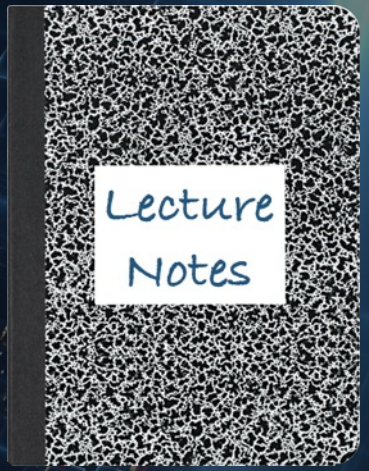
CS 417 – DISTRIBUTED SYSTEMS

Week 5: Consensus
Part 1: Theoretical framework



Paul Krzyzanowski

Last Week

Mutual Exclusion | Leader Election | Virtual Synchrony

Coordinate behavior:

- Who gets to access a resource
- Who's in charge
- What messages get delivered to a group

Do not guarantee "one truth" under partitions

Today:

Partitions | Split-Brain | Quorum | Consensus

Mutual exclusion algorithms: what they assume

Central coordinator

Lamport timestamps

Ricart-Agrawala

- **Goal:** at most one process enters the critical section
- **Preserve safety** by waiting for messages that might never arrive

Failure or disruption can stall progress

Leader election algorithms: what they assume

Bully

Ring

- **Goal**: choose one leader to coordinate decisions
- **Timeouts**: guesses, not proofs

Disruption can lead to multiple leaders or leader churn

Virtual synchrony : what It gives you, and what it does not

Virtual Synchrony

- **Goal**: consistent message delivery within a membership view
- Provides clean semantics for view changes and group messaging

Does not guarantee one system-wide sequence of decisions when views diverge – assumes networks never get partitioned

The common failure pattern

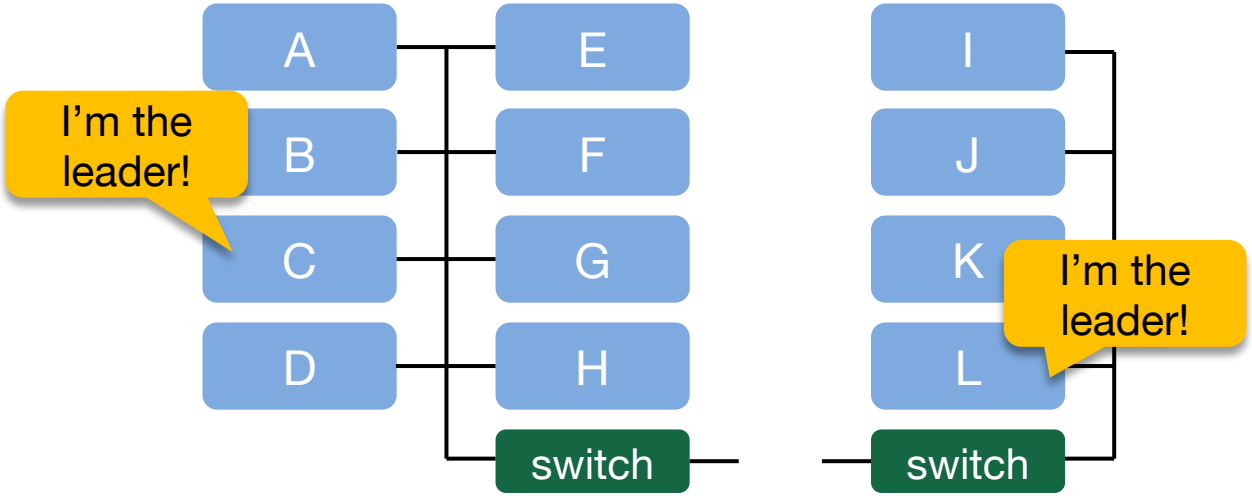
- Communication disruption creates incomplete information
- Different groups can make progress based on different observations

Result: conflicting decisions that cannot both be correct

Network partitions and split brain

- **Partition**: cluster splits into groups that cannot communicate
- **Split brain**: more than one group acts as if it is authoritative

Cost: divergent histories, difficult recovery, possible data loss



Common scenario:
Broken ethernet link between racks of computers – splits the network into two groups that cannot communicate with each other.

An election algorithm chooses two leaders.

Why timeouts do not solve this

Timeouts: detect missing messages, not failures

Slow and crashed look identical without timing bounds

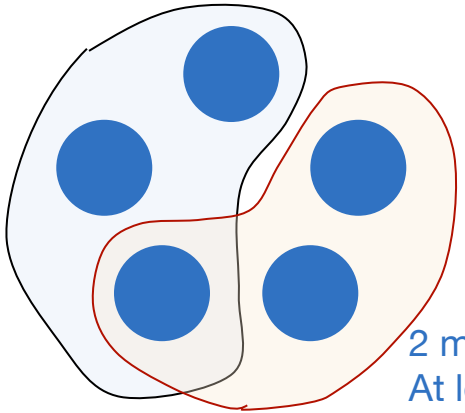
In a partition, both sides can time out:
They don’t know a network link was broken

Quorum

Quorum: threshold required to make a binding decision

Majority quorum: $\lceil (n+1)/2 \rceil$

- Any two majorities must overlap
- At most one partition can have a majority

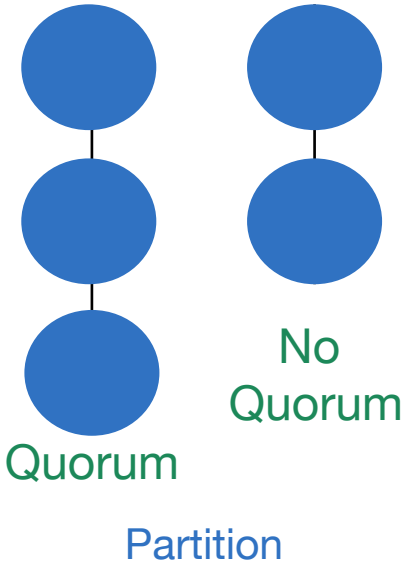


2 majorities:
At least one common member

Note:
There are other types of quorum, like ownership of a shared disk, but those usually don't scale and are rarely used.

Quorum example

- Partition 3-2: only the 3-side can commit decisions
- Minority side must reject updates (usually reject any requests)



Quorum math and tradeoff

Progress requires $\geq \lceil (n+1)/2 \rceil$ live, connected nodes: *majority*

Tolerates up to $\lfloor (n-1)/2 \rfloor$ **failures**: *group size – minimum majority*

Tradeoff: consistency vs. availability under partitions

Consistency	Availability
Only the majority can process requests.	Anyone can process requests.
If there are multiple majorities, overlap is guaranteed, so everyone can get the latest updates	If there are partitions, information cannot be shared across them, leading to inconsistencies. Everyone won't get all the updates.

The consensus problem

Consensus: get a set of nodes to decide on one value

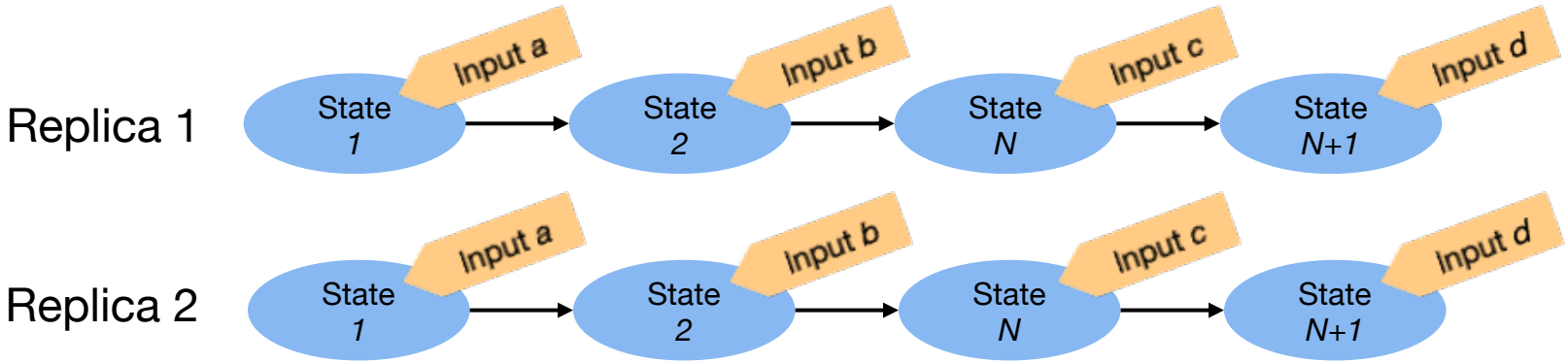
- Used for leader choice and for ordering updates
- In most environments, consensus is run repeatedly
  - We need to maintain consensus for a sequence of events

Examples of "values" nodes may need to agree on

- The next entry in the replicated log is "SET account_balance = 500"
- "This transaction is committed"
- "Server 3 is the new leader for this epoch"
- "The new cluster configuration includes servers 1, 2, and 4"

Replicated state machines

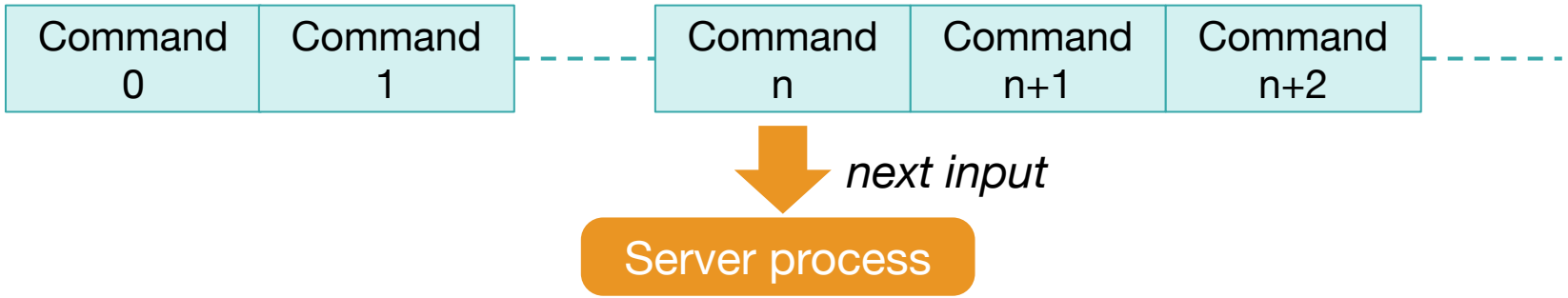
- Services can be viewed as a sequence of **states**
  - An input produces a deterministic output and a transition to a new state
  - "State" represents data storage or computing operations
- Same start state + same ordered commands → same state and outputs
- We want to **replicate** systems & their state for fault tolerance and scale
  - Replication reduces to agreeing on the command sequence



Replicated logs

Log = ordered sequence of commands

- Replicas append the same command at each position in the log
- Committed entries are applied in order
- Consensus chooses the entry for each log slot



Defining consensus

- Agreement

  No two nodes may select different values – you agree with everyone else

- Validity

  Only values proposed by some process may be selected – you can't make stuff up

- Integrity

  – A node can select only a single value – you cannot change your mind

- Termination (Progress)

  Every node will eventually decide on a value – you come to a decision

Why consensus is hard

Missing information is ambiguous

- *Late message* vs. *lost message* vs. *crashed sender*

To preserve safety, you may need to wait

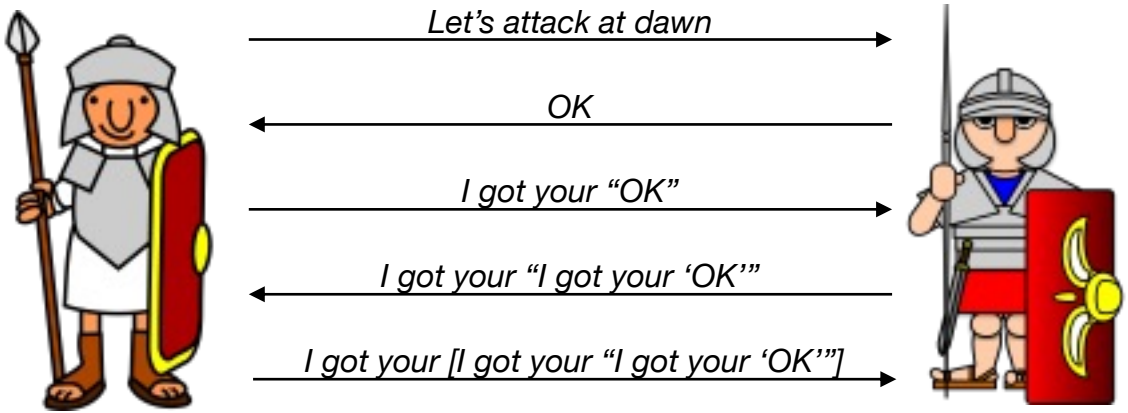
- Waiting can destroy termination

Reaching agreement in faulty systems

Two Armies Problem

- Good processors
- Asynchronous & unreliable communication lines
- Challenge: agree on an attack – be certain both sides have the message

⇒ *Infinite acknowledgment problem*



FLP Impossibility Result

Conditions:

- Pure asynchrony: no bound on message delay
- Crash failure allowed: at least one process may stop
- Processes behave correctly (no Byzantine faults)

FLP Impossibility Result:

- No deterministic protocol guarantees both safety and termination

Two Armies and FLP Impossibility

Shared problem: indistinguishability

- "Missing information" can be late or never arrive

- **Two Armies**: prevents guaranteed coordinated action
- **FLP**: prevents guaranteed termination of consensus

To be continued…