Research Group
Distributed Systems

Christian-Albrechts-Universität zu Kiel
Technische Fakultät

Fault Tolerance IV: Paxos

Olaf Landsiedel

Presentations Timeline

- Project Due: Thursday, 23.1., noon!
- Presentations: 2 slots
  - Thursday 23.1. afternoon (lab slot)
  - Tuesday 28.1. afternoon (lecture slot)
  - As before: just get up and present
    - Let us know if you have availability constraints

- Best Project Voting and Award
  - Tuesday 28.1. afternoon (lecture slot)
  - Via “one minute madness” (see next slides)

Presentations

- Presentation
  - 10 minutes
  - hard 10 minutes limit
- Q&A
  - 5 minutes
- Total time
  - 15 minutes

Presentations

- Present
  - Motivation
  - Background
  - Restate goals (from project idea talk)
  - Results (goals achieved)
  - Approach
  - Demo / Video
  - Evaluation
  - Lessons learned
  - Summary
  - …

Presentations - Background

- Do not forget the background part
  - If your algorithm, technology was part of the lectures
    - Restate it in the background to make sure everybody has their caches fresh
      - Also good for exam ;-)
  - If it was not part of the lectures
    - Provide compact background (watch time)
    - But so that everybody understands it
      - Have fun with this part, I struggle with it everyday
  - Background total time:
    - Not more than 2 to 3 minutes

Presentations - Demo / Video

- Consider showing a short demo or video
  - To highlight the best features of your system

- This way you can easily convince every one that you reached your goals

Presentations - Evaluation

- Do not forget to evaluate your approach
  - Performance
  - Overhead
  - Complexity
  - Resource consumption
  - …
- **Add nice graphs and plots**

Presentations

- I think you got some training until now

- Try to make the best out of your 10 minutes
- Think
  - What do you want to focus on?
    - You cannot show everything…
  - What is the most impressive part of your work?
  - How do you demonstrate your results?

Grading Criteria

- Project
  - Quality
  - Presentation
  - Reaching of goals
    - For 5 points:
    - For 10 points:
    - For 15 points:
  - Make sure you present a list of the goals achieved
    - Make clear to how many points this maps

Best Project Voting

- At the end: One-minute madness
  - Everyone presents a one minute summary
    - Submit a single PowerPoint slide, no animations etc.
    - We will add a project number we use for voting
  - We will prepare an animation, that shows
    - One slide per team for one minute, then it switches
    - Use this one minute to convince the other students
      - That your project is best
  - Voting: after the one minute madness session
    - At the end, we hand out the awards

One minute madness

- Why?
  - Fairness to recap all projects
  - Why one minute?
    - So called elevator pitch
    - One minute time to convince your boss/partner/friend of an idea
      - Usual time an elevator ride takes
      - Or: if you cannot convince someone in one minute that something is interesting, you most likely cannot do it all...
    - Good thing to train…

Submissions

- As in all the other labs
  - Your code
  - Your presentations
  - Anything else we would need to know
- Plus (own item in iLearn)
  - The single, one minute madness slide
    - No animations etc.
    - As PowerPoint
    - Submit under “Project: One minute madness slide”
  - Own deadline

Last Time

- Applications Part 2
  - Google file system
  - Map Reduce
  - Amazon Dynamo

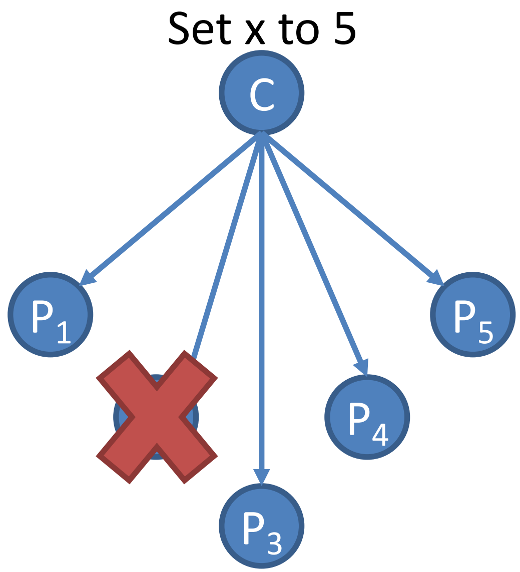
  - Research group:
    - Teaching and research

Today

- Paxos
  - Consensus Protocol
  - Begin with recap on consensus

Consensus

- Scenario assume
  - Write values to a replicated data store
  - Machines can fail
    - And restart
- We want
  - A system that keeps data on the (working) machines in sync
    - And not be blocked by failures



Consensus

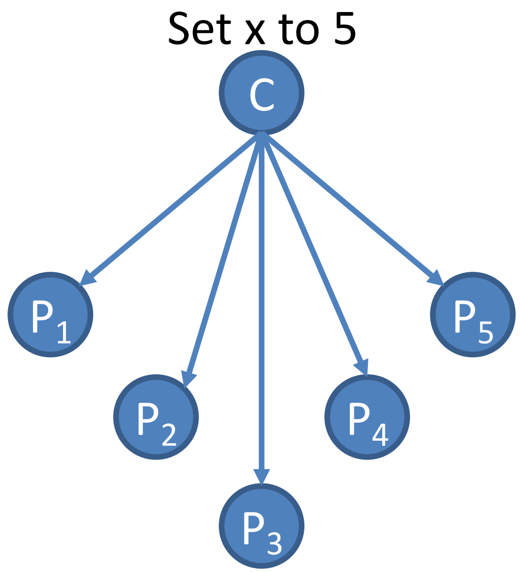
- A fundamental problem in Distributed Computing
  - achieve overall system reliability in the presence of a number of faulty processes.
- Approach
  - requires processes to agree on some data value that is needed during computation (=Consensus)

Consensus

- Examples of applications of **consensus** include
  - whether to commit a transaction to a replicated database
  - agreeing on the identity of a leader
  - state machine replication
  - atomic broadcasts, …
- The real world applications include
  - clock synchronization, PageRank, opinion formation, smart power grids, state estimation, control of UAVs (and multiple robots/agents in general), load balancing and others.

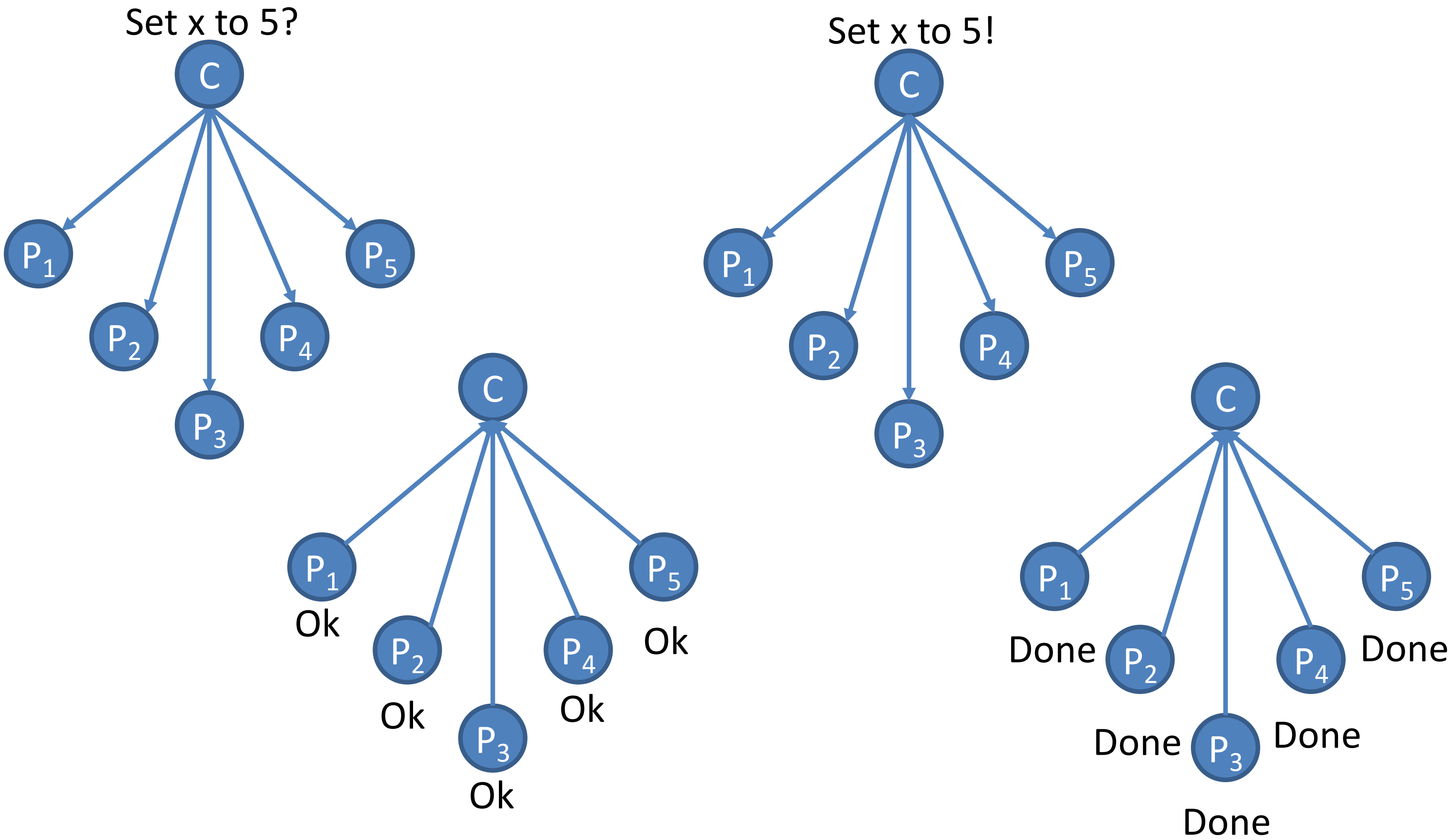
Recap: Two-Phase Commit Protocol

- Common Setting
  - Coordinator (C)
  - Participants (P_1 to P_n)

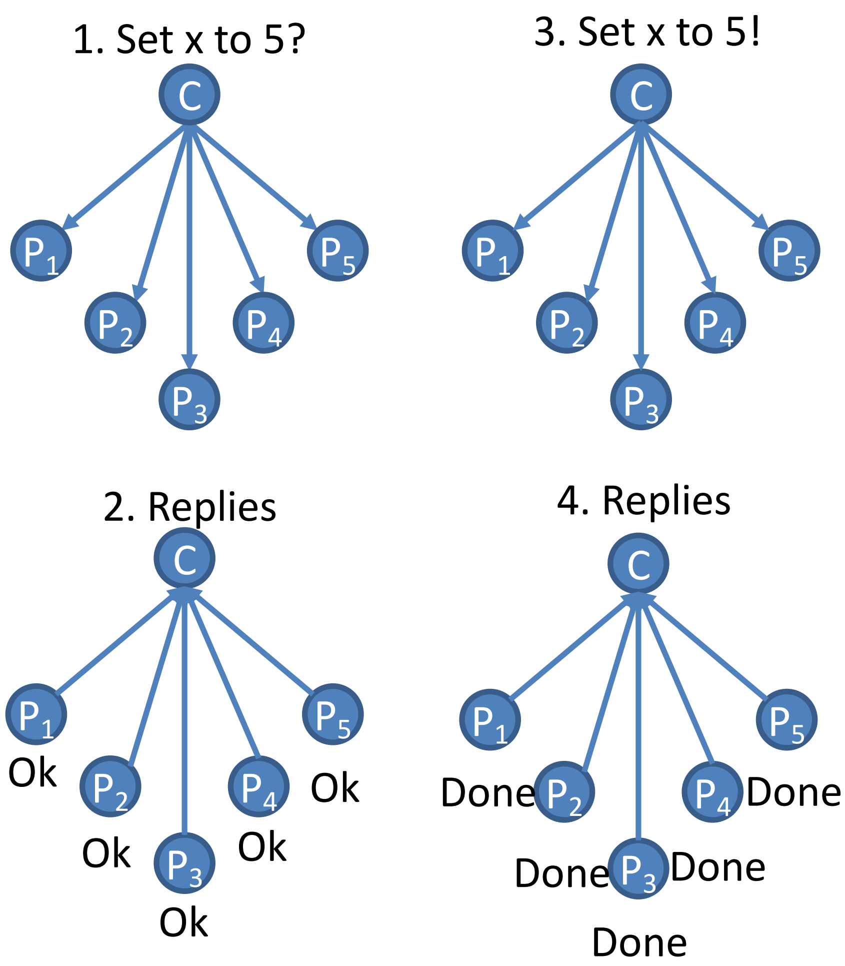


- What does the protocol do?
- How does it work?
  - See lectures on fault tolerance

Two-Phase Commit Protocol



Two-Phase Commit Protocol



- If a process P fails?
  - Will halt for recovery
  - Check all nodes: what did we agree on
  - -> Inefficient
  - Same problem for 3PC
  - Can we do better?
    - Majority should be sufficient….
    - -> Paxos…

Summary

- 2PC is not consensus
  - Must wait for all sites and Coordinator to be up
  - Must know if each site voted yes or no
  - C must be up to decide
  - Doesn’t tolerate faults well; must wait for repair
- 3PC
  - Can lead to inconsistencies in case of failures
- -> Paxos: fix these issues

Today

- Paxos: A consensus algorithm
  - Known as one of the most efficient & elegant consensus algorithms
- Plan
  - Brief history (with a lot of quotes)
  - The protocol itself

Reminder : Agreement in Faulty Systems

- Reaching a distributed agreement (=consensus) is only possible in the following circumstances:

		Message Ordering				
		Unordered		Ordered		
Process Behavior	Synchronous	✓	✓	✓	✓	Bounded
				✓	✓	Unbounded
	Asynchronous				✓	Bounded
					✓	Unbounded
		Unicast	Multicast	Unicast	Multicast	Communication Delay
		Message Transmission				

- Known as FLP result: Fischer, Lynch and Patterson, ‘Impossibility of Distributed Consensus with One Faulty Process’, 1985

Failure Detector

- Failure detector
  - distinguish between a slow connection and a failed connection / node
- How
  - Example: We know the typical round trip time (RTT) of a connection
    - If a reply is not received after X RTT, we transmit
    - For example: After three retransmissions
    - -> consider a node not available

Basic Idea

- X nodes
  - All keep a copy of the data
    - And record each change (history)
  - Also known as replicated state machine

Paxos: Brief History

- Developed by Leslie Lamport
  - Logical clocks, and many others things…
- Why? Leslie Lamport:
  "A fault-tolerant file system called Echo was built at SRC in the late 80s. The builders claimed that it would maintain consistency despite any number of non-Byzantine faults, and would make progress if any majority of the processors were working."

Paxos: Brief History

- Leslie Lamport:
  “I decided that what they were trying to do was impossible, and set out to prove it. Instead, I discovered the Paxos algorithm.”

- *“I decided to cast the algorithm in terms of a parliament on an ancient Greek island (Paxos).”*
  - *As the algorithm works like a paralment*

Paxos: Brief History

- The paper abstract of the Paxos paper:
  - Have a look and tell me your thoughts…

  "Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems."

Brief History

- Paper abstract
  - This is not how research papers are written
- Leslie Lamport:
  - *"I gave a few lectures in the persona of an Indiana-Jones-style archaeologist."*
  - *"My attempt at inserting some humor into the subject was a dismal failure. People who attended my lecture remembered Indiana Jones, but not the algorithm."*

Summary

- People thought that Paxos was a joke
- Lamport finally published the paper 8 years later in 1998 after it was written in 1990
  - Title: “The Part-Time Parliament”
- People did not understand the paper
  - Probably the paper was ahead of its time
  - The paper is not easy to read, even today
  - But, it is a very important algorithm today
    - Used in many real world systems

Summary

- Lamport gave up and wrote another paper that explains Paxos in simple English.
  - Title: “Paxos Made Simple”
  - Abstract: “The Paxos algorithm, when presented in plain English, is very simple.”
- Still, it’s not the easiest algorithm to understand
- So people started to write papers and lecture notes to explain “Paxos Made Simple.” (e.g., “Paxos Made Moderately Complex”, “Paxos Made Practical”, etc.)

Consensus

- Today, many real-world systems implement it
  - Google Chubby, Google Spanner and Google Megastore
  - IBM SAN Volume Controller
  - MS AutoPilot cluster management
  - Heroku for its consistent distributed data store
  - Ceph uses Paxos
  - Clustrix distributed SQL database
  - Neo4j HA graph
  - Apache Cassandra NoSQL database
  - …

Amazon CTO Werner Vogels

- *"What kind of things am I looking for in you?"*
  - *"You know your distributed systems knowledge: You know about …*
    - *logical time,* ✓
    - *snapshots,* ✓
    - *stability,* ✓
    - *message ordering* ✓
    - *acid and multi-level transactions.* ***X Database course***
    - *You have heard about the FLP impossibility argument. You know why failure detectors can solve it …* ✓
    - *You have at least once tried to understand Paxos by reading the original paper* **Today** ✓

Simple Pseudocode

outcome[*p*]	The decree written in *p*'s ledger, or BLANK if there is nothing written there yet.
lastTried[*p*]	The number of the last ballot that *p* tried to begin, or $-\infty$ if there was none.
prevBal[*p*]	The number of the last ballot in which *p* voted, or $-\infty$ if he never voted.
prevDec[*p*]	The decree for which *p* last voted, or BLANK if *p* never voted.
nextBal[*p*]	The number of the last ballot in which *p* agreed to participate, or $-\infty$ if he has never agreed to participate in a ballot.

Next come variables representing information that priest *p* could keep on a slip of paper:

status[*p*]	One of the following values: *idle* Not conducting or trying to begin a ballot *trying* Trying to begin ballot number *lastTried*[*p*] *polling* Now conducting ballot number *lastTried*[*p*] If *p* has lost his slip of paper, then *status*[*p*] is assumed to equal *idle* and the values of the following four variables are irrelevant.
prevVotes[*p*]	The set of votes received in *LastVote* messages for the current ballot (the one with ballot number *lastTried*[*p*]).
quorum[*p*]	If *status*[*p*] = *polling*, then the set of priests forming the quorum of the current ballot; otherwise, meaningless.
voters[*p*]	If *status*[*p*] = *polling*, then the set of quorum members from whom *p* has received *Voted* messages in the current ballot; otherwise, meaningless.
decree[*p*]	If *status*[*p*] = *polling*, then the decree of the current ballot; otherwise, meaningless.

Try New Ballot
Always enabled.
- Set *lastTried*[*p*] to any ballot number *b*, greater than its previous value, such that *owner*(*b*) = *p*.
- Set *status*[*p*] to *trying*.
- Set *prevVotes*[*p*] to $\emptyset$.

Send *NextBallot* Message
Enabled whenever *status*[*p*]=*trying*.
- Send a *NextBallot*(*lastTried*[*p*]) message to any priest.

Receive *NextBallot*(*b*) Message
If $b \geq$ *nextBal*[*p*] then
- Set *nextBal*[*p*] to *b*.

Send *LastVote* Message
Enabled whenever *nextBal*[*p*] > *prevBal*[*p*].
- Send a *LastVote*(*nextBal*[*p*], *v*) message to priest *owner*(*nextBal*[*p*]), where $v_{pst} = p$, $v_{bal} = prevBal[p]$, and $v_{dec} = prevDec[p]$.

Receive *LastVote*(*b*, *v*) Message
If *b* = *lastTried*[*p*] and *status*[*p*] = *trying*, then
- Set *prevVotes*[*p*] to the union of its original value and $\{v\}$.

Start Polling Majority Set *Q*
Enabled when *status*[*p*] = *trying* and $Q \subseteq \{v_{pst} : v \in prevVotes[p]\}$, where *Q* is a majority set.
- Set *status*[*p*] to *polling*.
- Set *quorum*[*p*] to *Q*.
- Set *voters*[*p*] to $\emptyset$.
- Set *decree*[*p*] to a decree *d* chosen as follows: Let *v* be the maximum element of *prevVotes*[*p*]. If $v_{bal} \neq -\infty$ then $d = v_{dec}$, else *d* can equal any decree.
- Set $\mathcal{B}$ to the union of its former value and $\{B\}$, where $B_{dec} = d$, $B_{qrm} = Q$, $B_{vot} = \emptyset$, and $B_{bal} = lastTried[p]$.

Send *BeginBallot* Message
Enabled when *status*[*p*] = *polling*.
- Send a *BeginBallot*(*lastTried*[*p*], *decree*[*p*]) message to any priest in *quorum*[*p*].

Receive *BeginBallot*(*b*, *d*) Message
If *b* = *nextBal*[*p*] > *prevBal*[*p*] then
- Set *prevBal*[*p*] to *b*.
- Set *prevDec*[*p*] to *d*.
- If there is a ballot *B* in $\mathcal{B}$ with $B_{bal} = b$ [there will be], then choose any such *B* [there will be only one] and let the new value of $\mathcal{B}$ be obtained from its old value by setting B_{vot} equal to the union of its old value and $\{p\}$.

Send *Voted* Message
Enabled whenever *prevBal*[*p*] $\neq -\infty$.
- Send a *Voted*(*prevBal*[*p*], *p*) message to *owner*(*prevBal*[*p*]).

Receive *Voted*(*b*, *q*) Message
If *b* = *lastTried*[*p*] and *status*[*p*] = *polling*, then
- Set *voters*[*p*] to the union of its old value and $\{q\}$

Succeed
Enabled whenever *status*[*p*] = *polling*, *quorum*[*p*] $\subseteq$ *voters*[*p*], and *outcome*[*p*] = BLANK.
- Set *outcome*[*p*] to *decree*[*p*].

Send *Success* Message
Enabled whenever *outcome*[*p*] $\neq$ BLANK.
- Send a *Success*(*outcome*[*p*]) message to any priest.

Receive *Success*(*d*) Message
If *outcome*[*p*] = BLANK, then
- Set *outcome*[*p*] to *d*.

Paxos Assumptions & Goals

- The network is *asynchronous* with message delays
- The network can *lose or duplicate* messages, but *cannot corrupt* them
- Processes can *crash*
- Processes are *non-Byzantine* (only crash-stop)
- Processes have *permanent storage*
- Processes can *propose* values

- The goal: every (working) process agrees on a value out of the proposed values.

Note on Goals

- Paxos
  - In the basic version we discuss now
  - **Can only achieve consensus once**
  - Ensures that the consensus does not change anymore
    - Once reached

- MultiPaxos and others
  - Extend on this
  - To have multiple consensus rounds

Desired Properties

- Safety
  - Only a value that has been proposed can be chosen
  - Only a single value is chosen
  - A process never learns that a value has been chosen unless it has been
- Liveness
  - Some proposed value is eventually chosen
  - If a value is chosen, a process eventually learns it

Roles of a Process I

Three roles

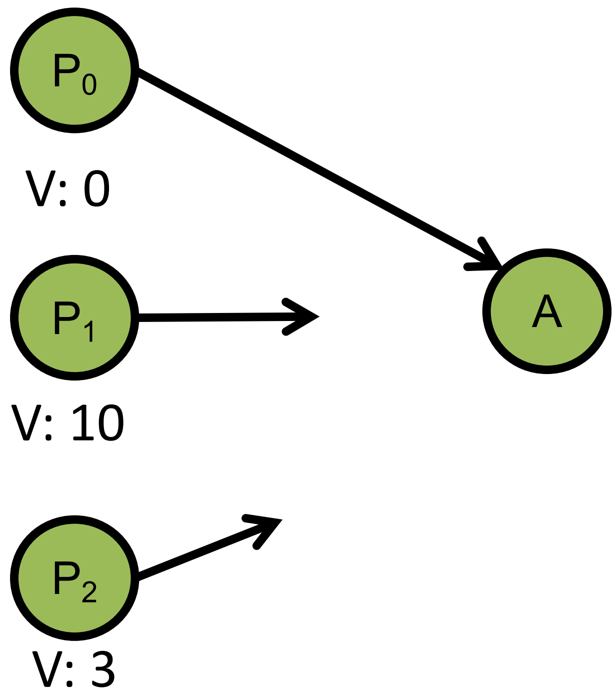
- Proposers: processes that propose values
- Acceptors: processes that accept (i.e., consider) values
  - "Considering a value": the value is a candidate for consensus.
  - Majority acceptance → choosing the value
    - They do the majority decision (=voting)
- Learners: processes that learn the outcome
  - i.e., chosen value
  - These do not participate in the majority decision

Roles of a Process II

- In reality, a process can be
  - any one, two, or all three.
- Important requirements
  - The protocol should work under process failures and with delayed and lost messages.
  - The consensus is reached via a majority (> ½).
- Example: a replicated state machine
  - All replicas agree on the order of execution for concurrent transactions
  - All replica assume all roles, i.e., they can each propose, accept, and learn

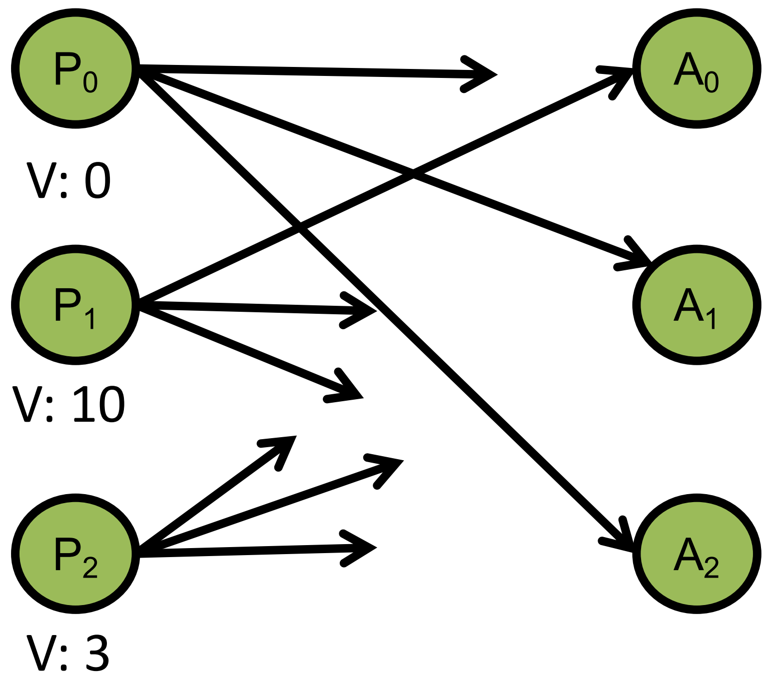
First Attempt

- Processes P
  - Each proposes a value V
- Let's just have one acceptor A
  - choose the first one that arrives
  - A tells the proposers about the outcome
- No learners involved in this example
- What's wrong?
  - Single point of failure!



Second Attempt

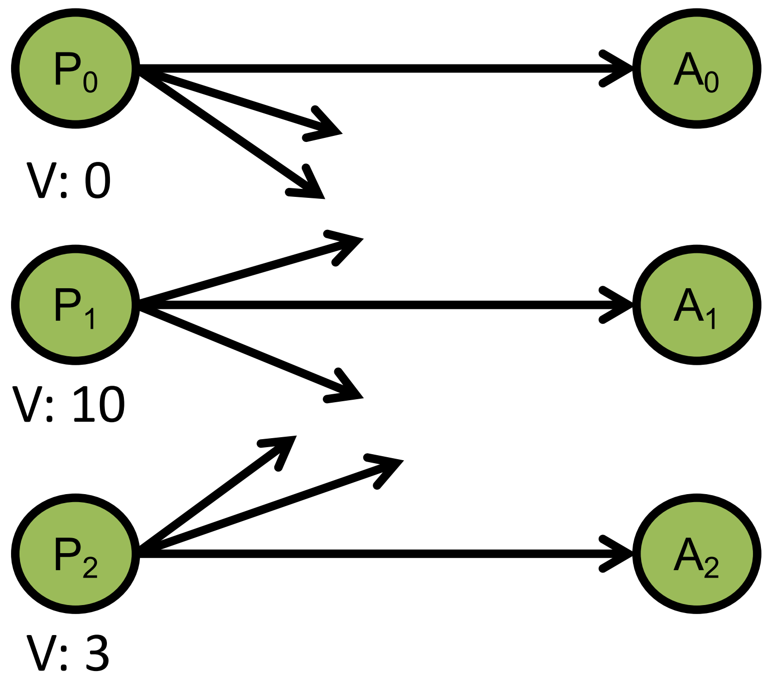
- Let's have multiple acceptors
  - each accepts the first proposal it receives
  - then all choose the majority
  - and tell the proposers about the outcome



- What's wrong? (next slide)

Second Attempt

- What if each acceptor receives a different message?
- One example
  - – many other possibilities



Let's fix this

- With Paxos…

Simple Implementation

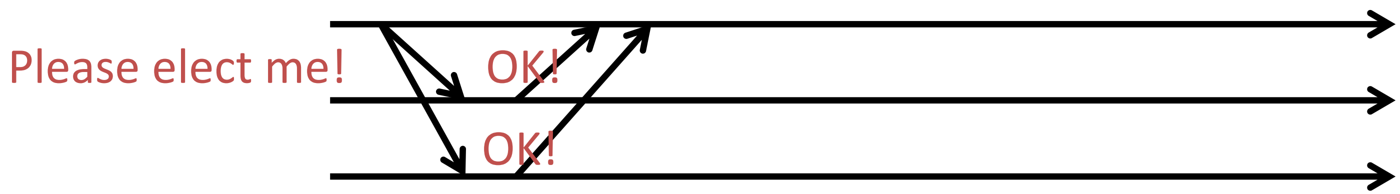
- Typically, every process is acceptor, proposer, and learner
- A leader is elected to be the distinguished proposer and learner
  - Distinguished proposer to guarantee progress
    - Avoid dueling proposers
  - Extension
    - Distinguished learner to reduce too many broadcast messages

Political Analogy

- Paxos has rounds: each round has a unique ballot ID
  - New rounds are started until majority is reached and known by all -> Paxos has completed
- Rounds are asynchronous
  - Time synchronization not required
  - If you are in round j and hear a message from round j+1, abort everything and move to round j+1
- Each round consists of three phases
  - Phase 1: A leader is elected (Election)
  - Phase 2: Leader proposes a value, processes acks (Bill)
  - Phase 3: Leader multicasts final value (Law)
  - Note:
    - Phases maybe interrupted and nodes move to a new round: see above
    - Ends when all nodes know the result, see above

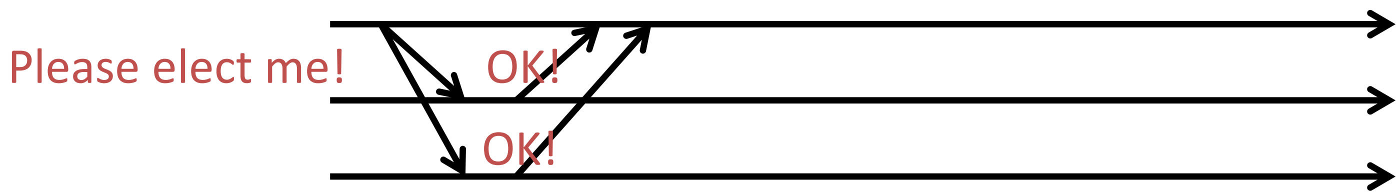
Phase 1 – Election

- Potential leader chooses a unique ballot ID, higher than anything it has seen so far
- Sends ballot ID to all processes
- Processes respond to highest ballot id
  - If potential leader sees a higher ballot id, it can't be a leader
  - Processes log received ballot ID on disk: Failure persistent



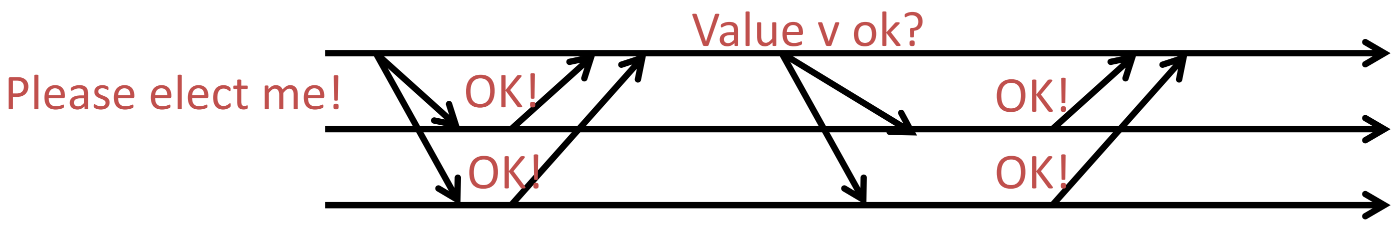
Phase 1 – Election

- If a process has in a previous round decided on a value v’, it includes value v’ in its response
- If majority (i.e., quorum) responded OK then you are the leader
  - If no one has majority, start new round
- A round cannot have two leaders (why?)
  - Need majority to be the leader



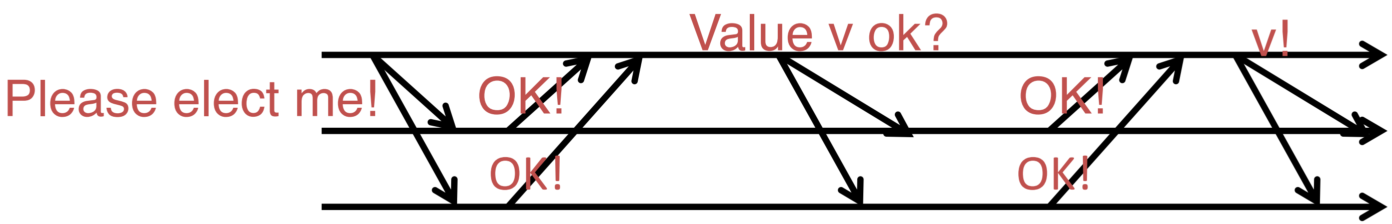
Phase 2 – Proposal (Bill)

- Leader sends proposal value v to all
  - If some process already decided value v’ in a previous round
  - It told leader about this during election
    - See previous slide
  - Leader sends v = v’
- Recipient log on disk, and responds OK



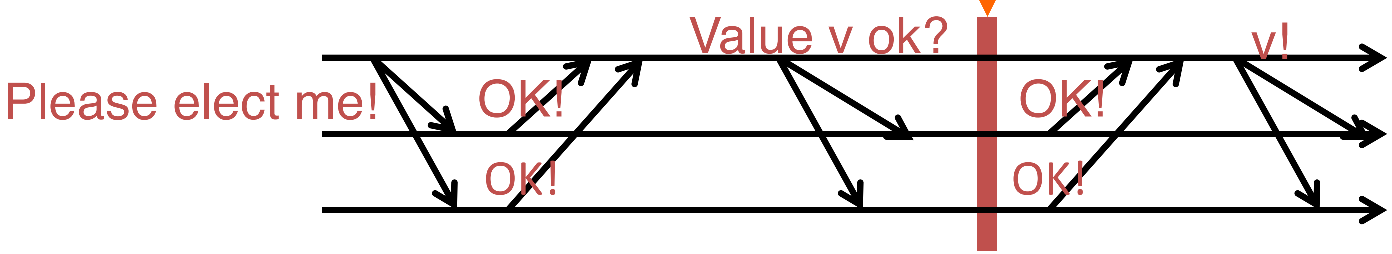
Phase 3 – Decision (Law)

- If leader hears OKs from majority, it lets everyone know of the decision
- Recipients receive decisions, log it on disk



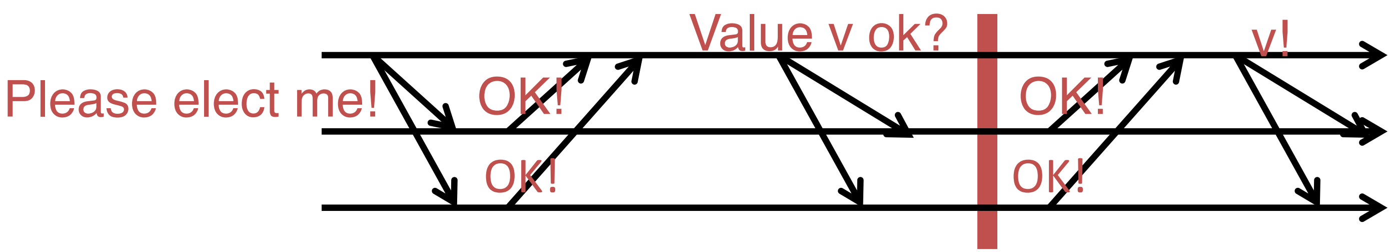
When is Consensus Achieved?

- When is Consensus Achieved?
- When a majority of processes hear proposed value and accept it:
  - Are about to respond (or have responded) with OK!
- At this point decision has been made even though
  - Processes or even leader may not know!
- What if leader fails after that?
  - Keep having rounds until some round complete
  - With a new leader, but same value v (see previous slide)



Safety

- Assume a round with a majority hearing proposed value v' and accepting it (mid of Phase 2). Then subsequently at each round either:
  - The round chooses v' as decision
  - The round fails
- "Proof":
  - Potential leader waits for majority of OKs in Phase 1
  - At least one will contain v' (because two majority sets intersect)
  - It will choose to send out v' in Phase 2
- Success requires a majority, and two majority sets intersects



More Paxos in more detail…

Basic Paxos Protocol

Phase 1a: **"Prepare"**
Select proposal number* *N* and send a ***prepare(N)*** request to a quorum of acceptors.

Proposer

Phase 1b: **"Promise"**
If N > *number of any previous promises or acceptances*,
* promise to never accept any future proposal less than *N*,
- send a ***promise(N, U)*** response
(where *U* is the highest-numbered proposal accepted so far (if any))

Phase 2a: **"Accept!"**
If proposer received promise responses from a quorum,
- send an ***accept(N, W)*** request to those acceptors
(where ***W*** is the value of the highest-numbered proposal among the ***promise*** responses, or any value if no ***promise*** contained a proposal)

Acceptor

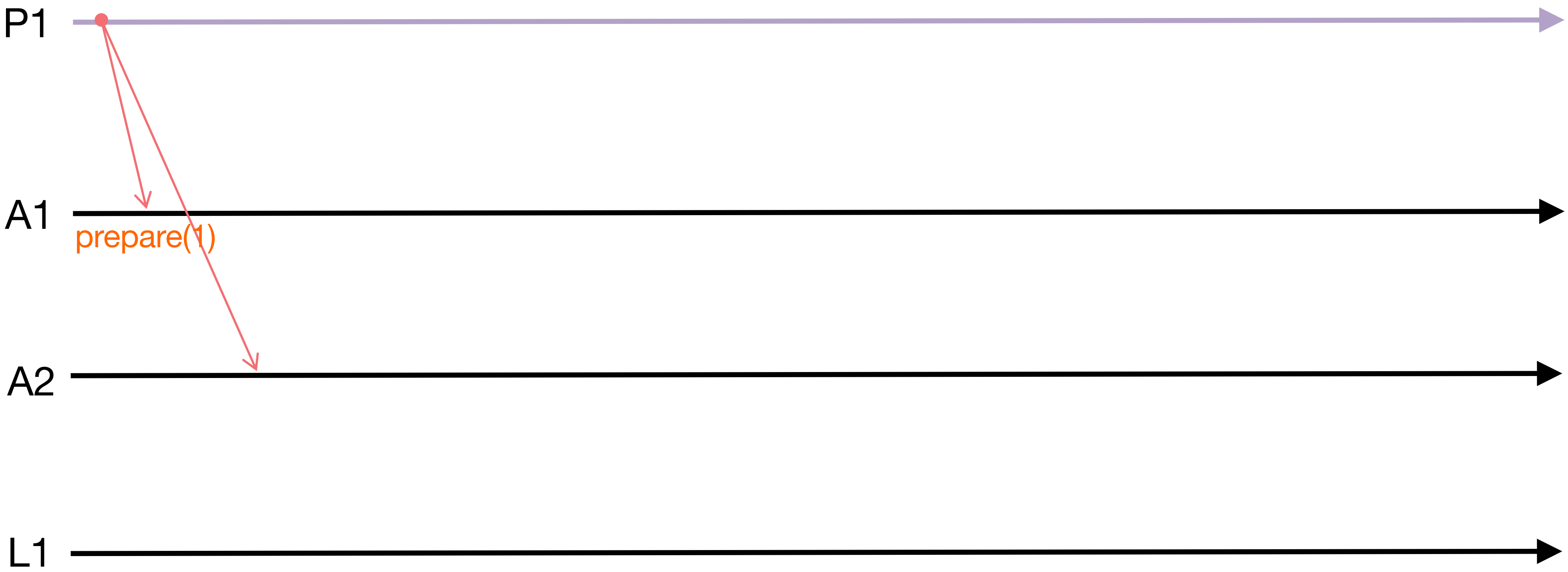
Phase 2b: **"Accepted"**
If N >= *number of any previous promise*,
* accept the proposal
- send an ***accepted*** notification to the learner

* = record to stable storage

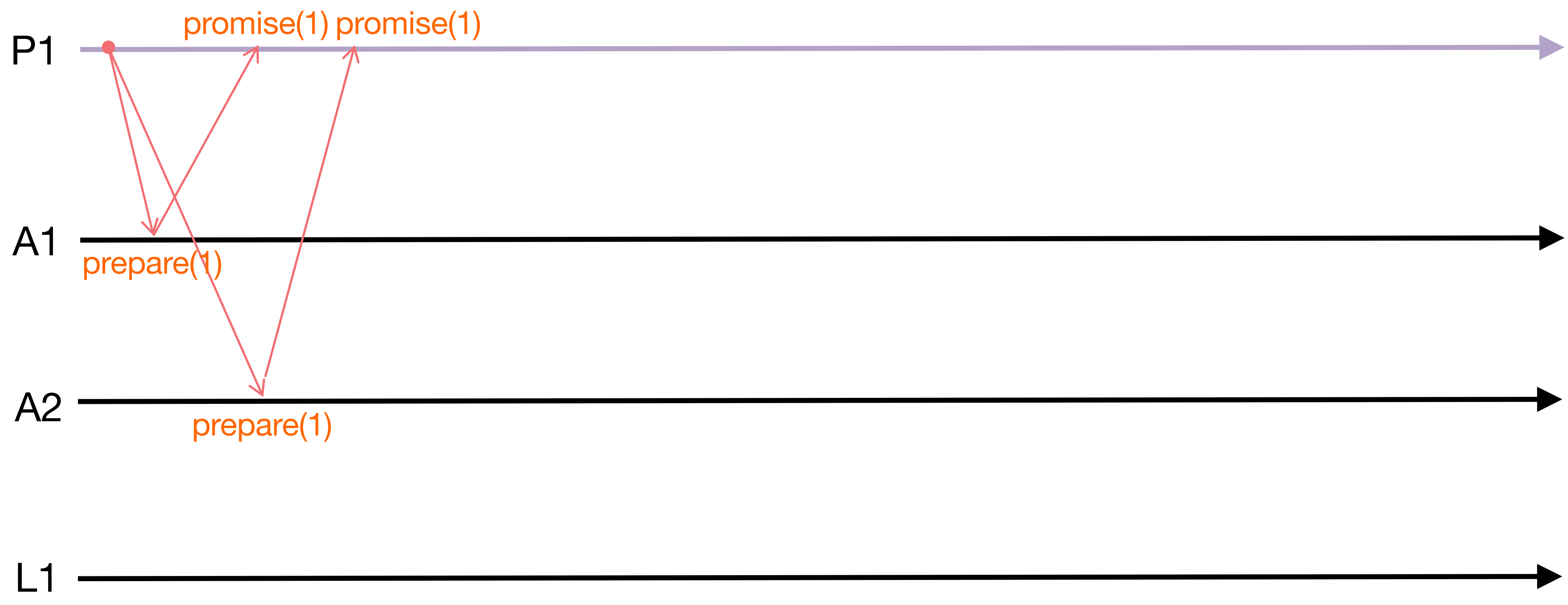
Trivial Example: P1 wants to propose “A”

- To warm up
- 4 nodes
  - One proposer
    - Proposes the value “A”
  - Two acceptors
    - Need majority of the nodes, so both nodes to accept
  - One learner
- In this example
  - Ballot ID used by the proposer: 1

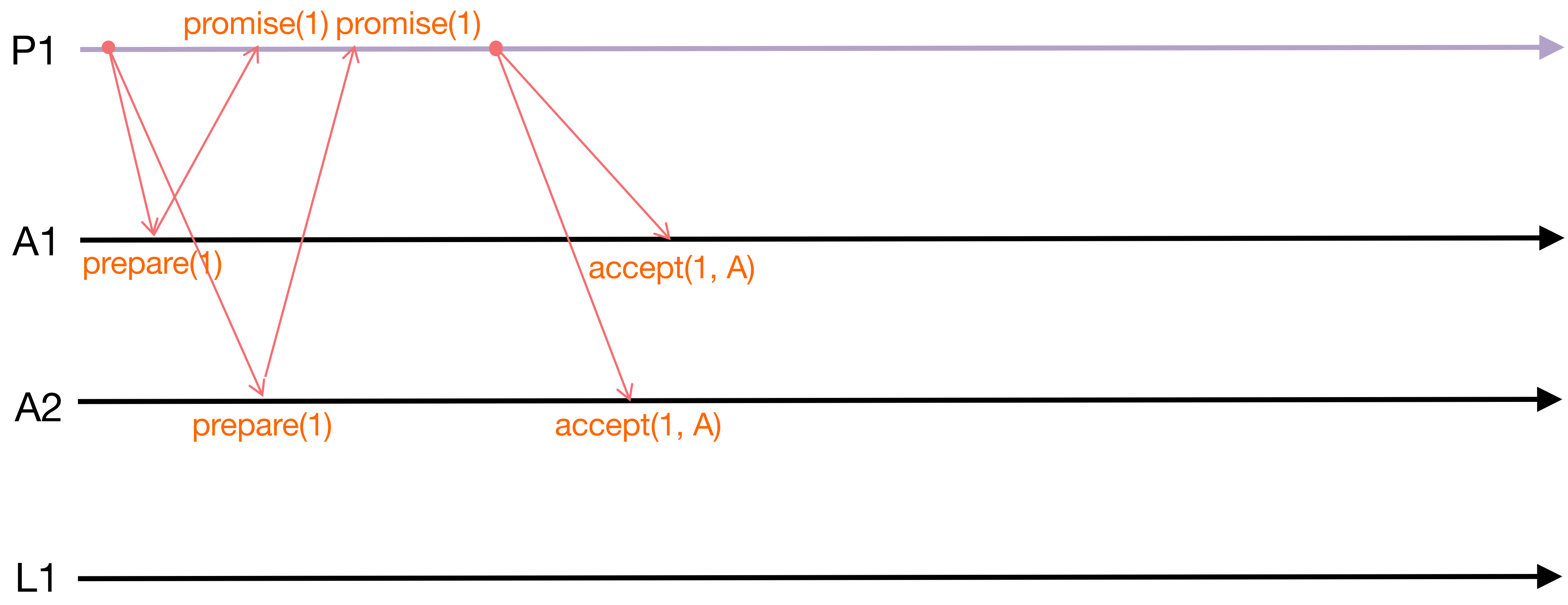
Trivial Example: P1 wants to propose "A"



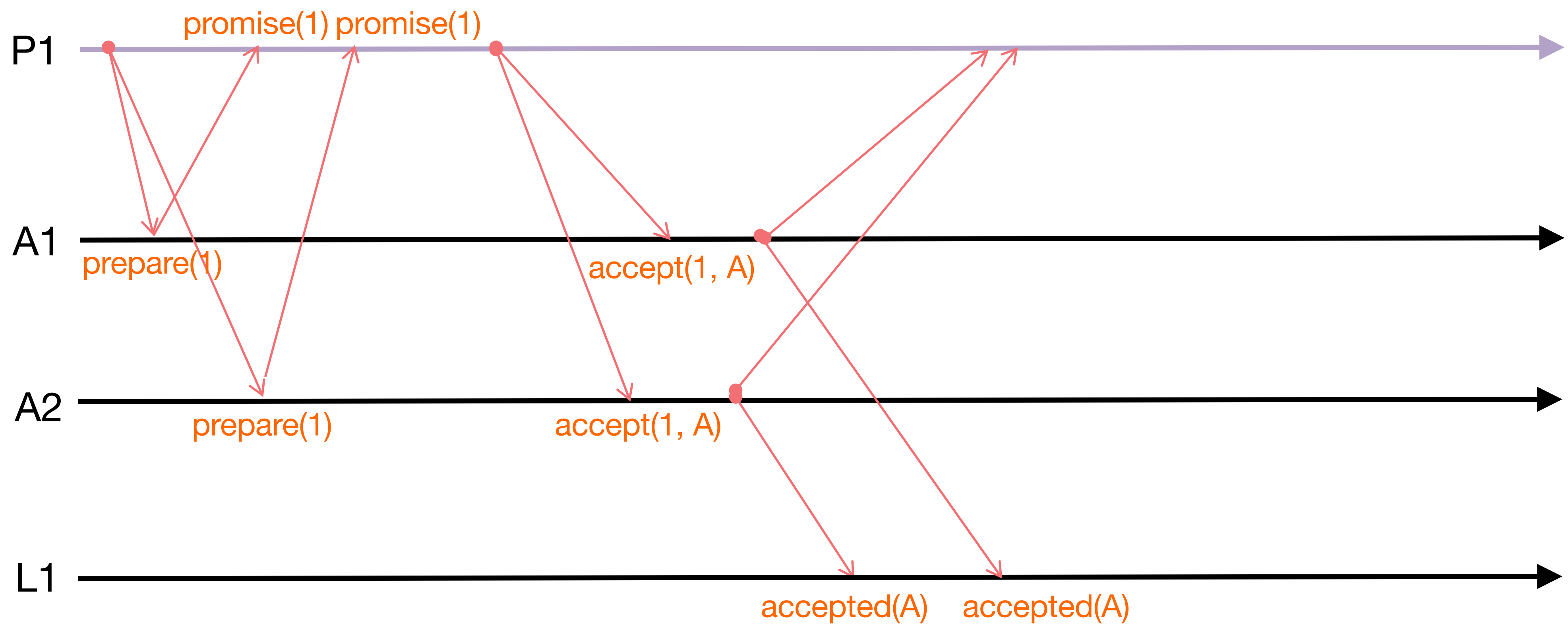
Trivial Example: P1 wants to propose "A"



Trivial Example: P1 wants to propose "A"



Trivial Example: P1 wants to propose "A"

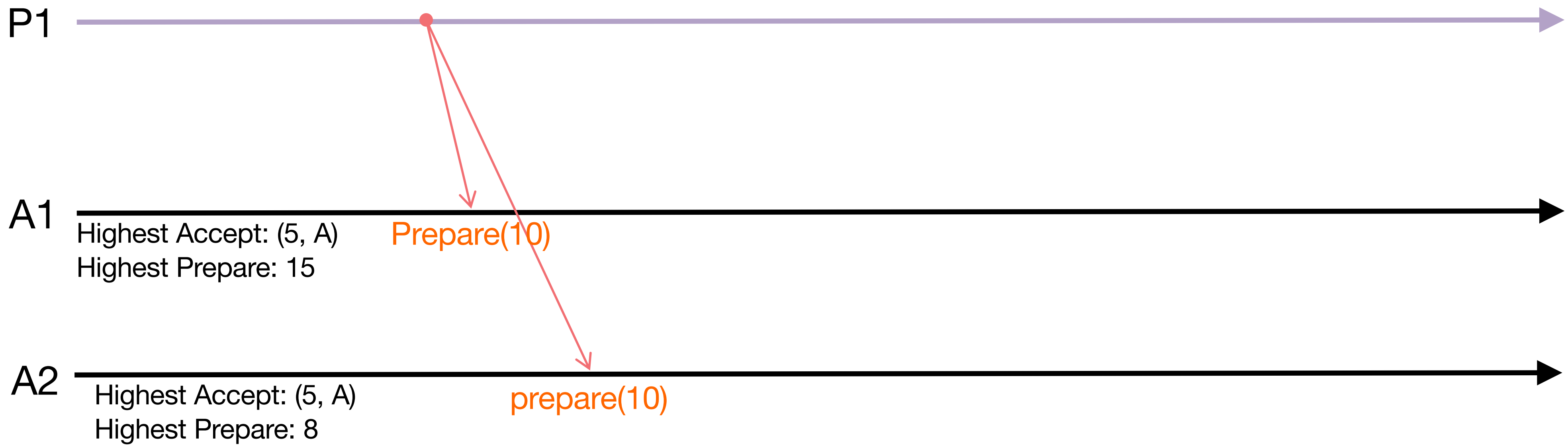


Prepare example

- Scenario
  - Different ballot IDs
  - Proposer chooses a ballot ID that is lower than what other nodes have seen before

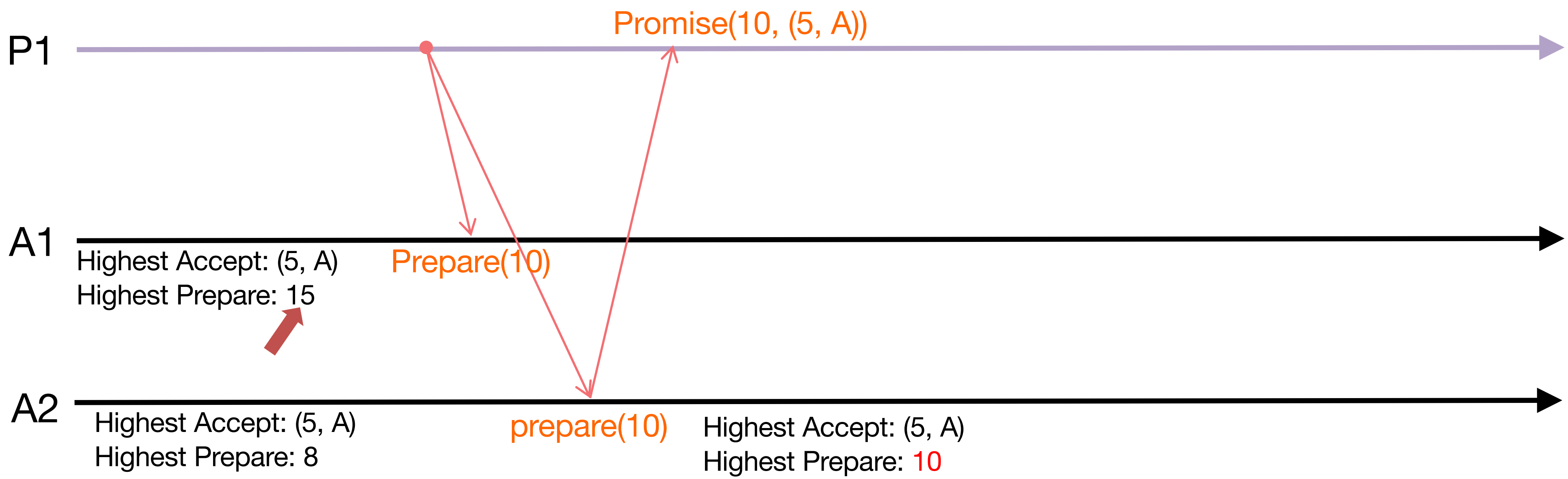
- One proposer
- Two acceptors
- We omit the learners here

Prepare Example



Proposer P1 chooses a ballot ID of 10, but Acceptor A1 has already received a ballot ID of 15 previously -> A1 cannot accept P1 as new leader

Prepare Example

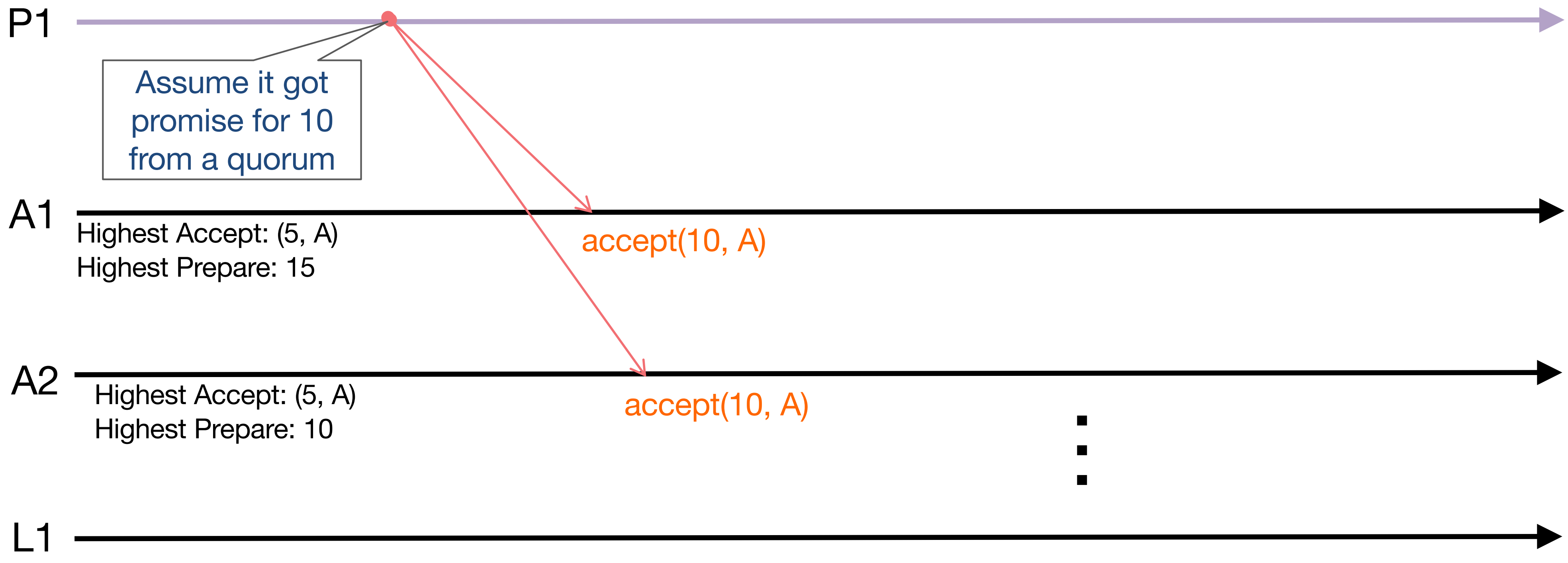


Proposer P1 chooses a ballot ID of 10, but Acceptor A1 has already received a ballot ID of 15 previously -> A1 cannot accept P1 as new leader

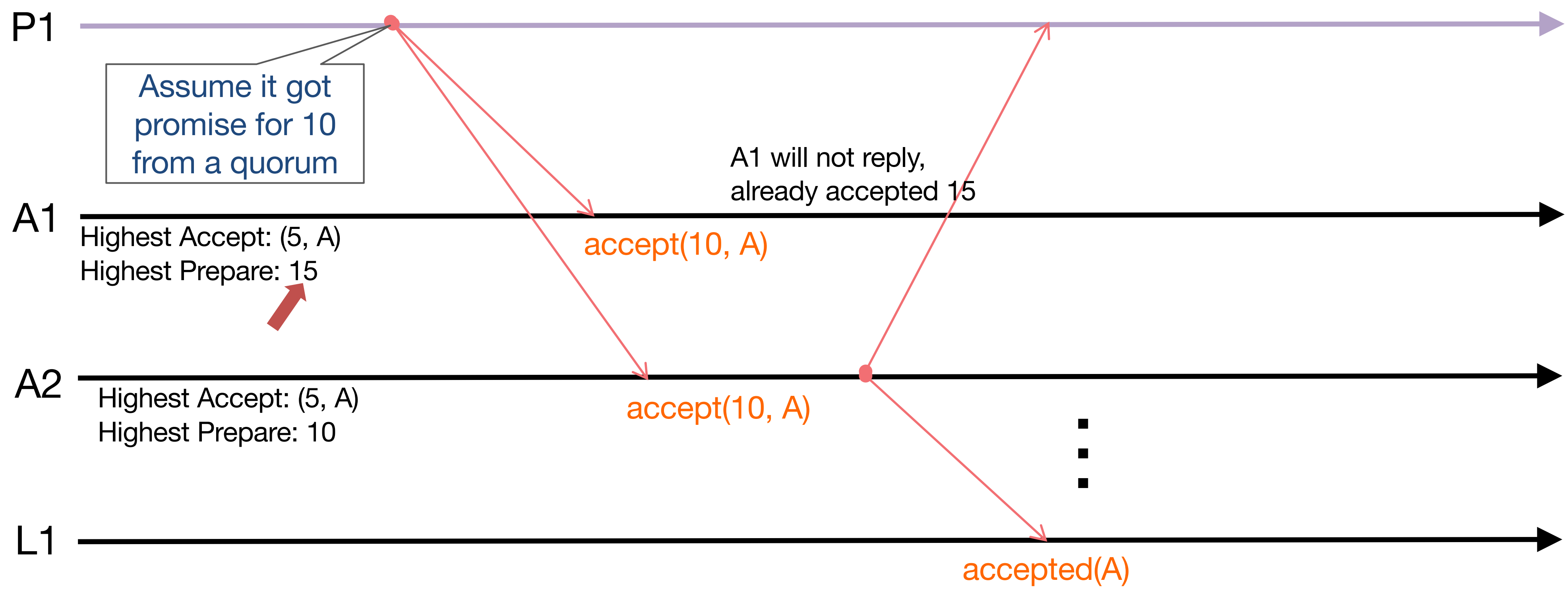
Simple accept

- Now: focus on second part
  - Accept messages
- Two or more proposers
  - But only 1 is shown
- Three or more acceptors
  - But only 2 are shown
- One learner

Simple Accept Example

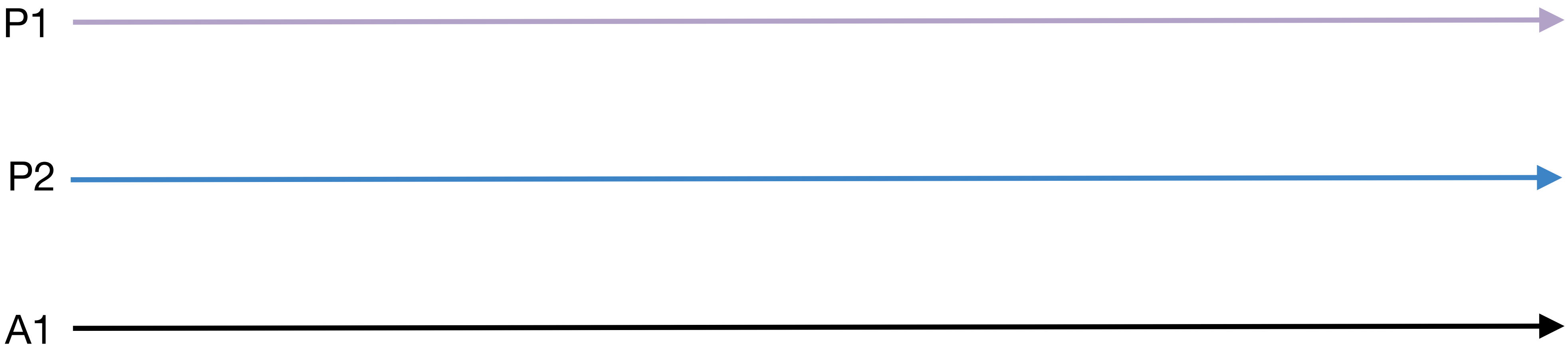


Accept Example



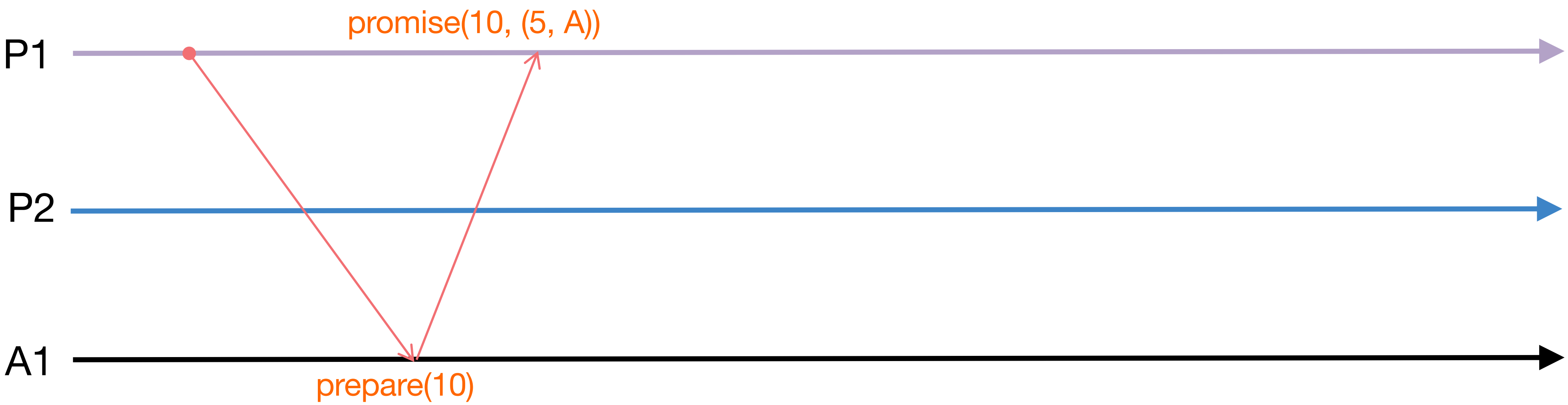
This can lead to funny things…

Example: Livelock, Two Proposers



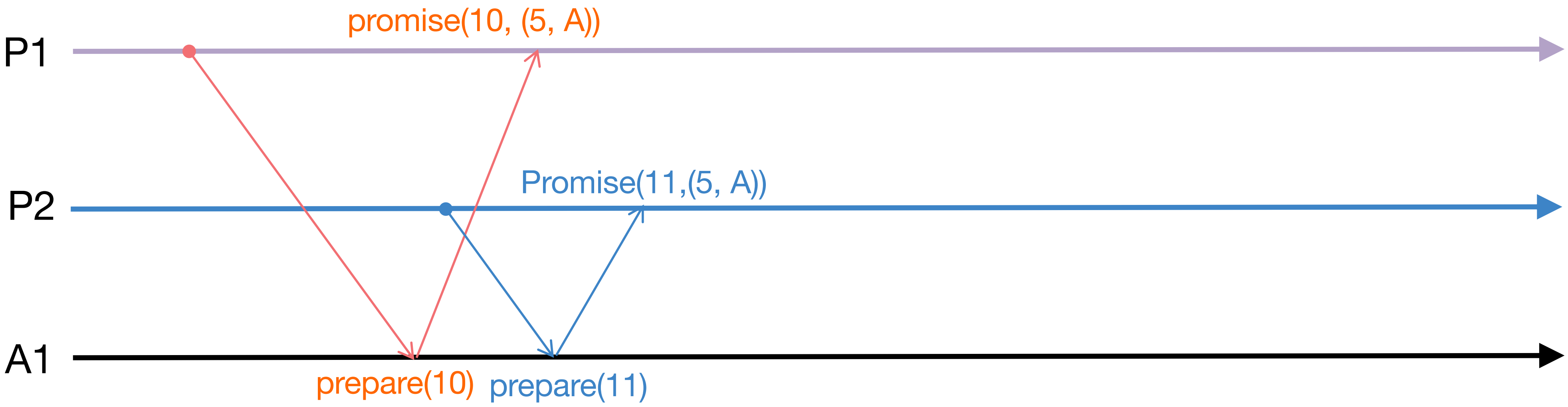
A1: Highest accept; (5, A)
Highest prepare: 8

Example: Livelock



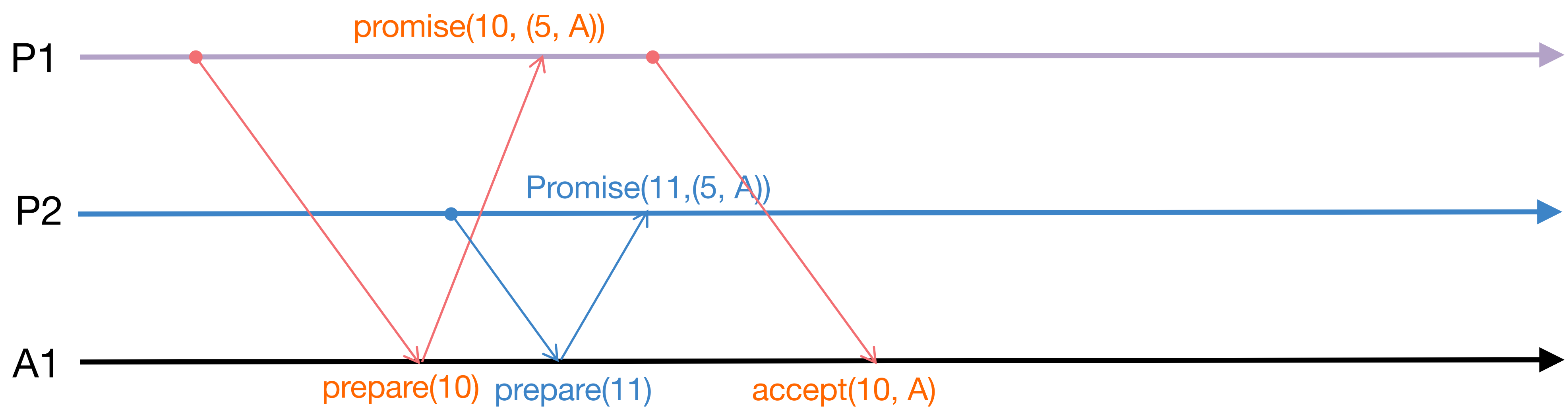
A1: Highest accept; (5, A)
Highest prepare: 10

Example: Livelock



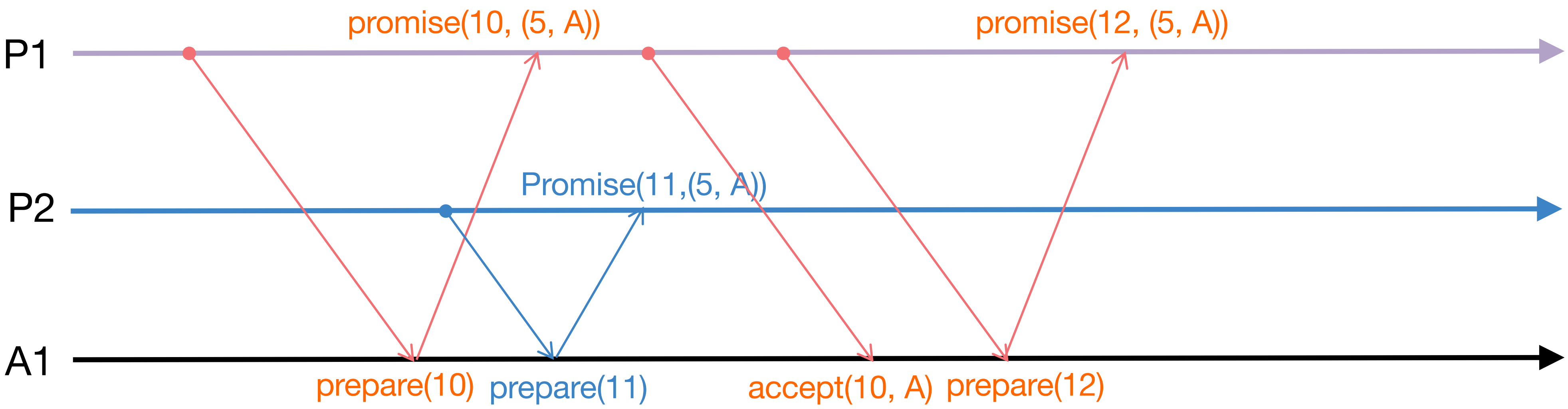
A1: Highest accept; (5, A)
Highest prepare: 11

Example: Livelock



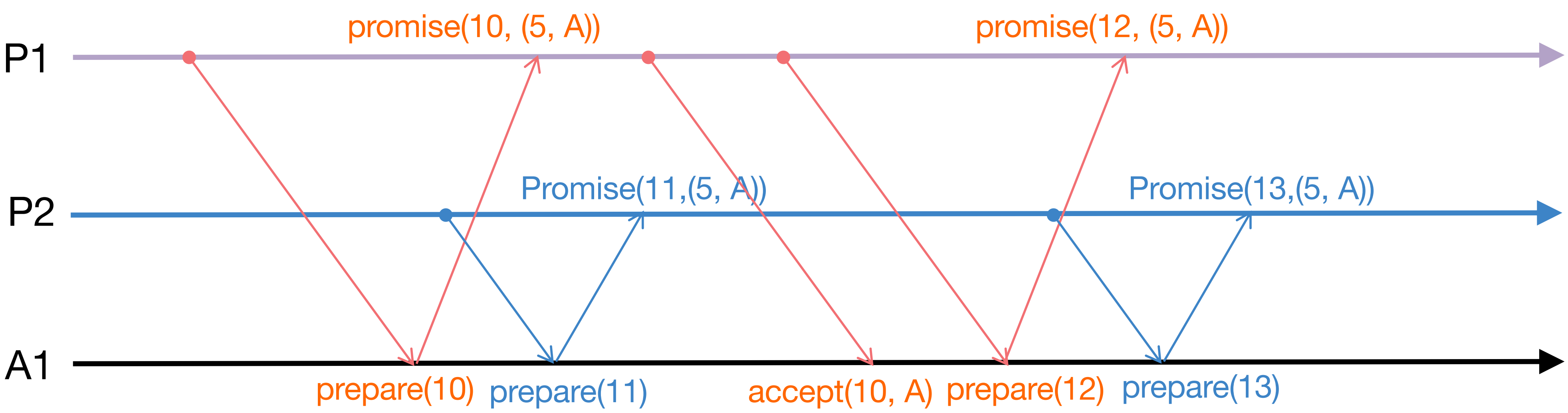
A1: Highest accept; (5, A)
Highest prepare: 11

Example: Livelock



A1: Highest accept; (5, A)
Highest prepare: 12

Example: Livelock

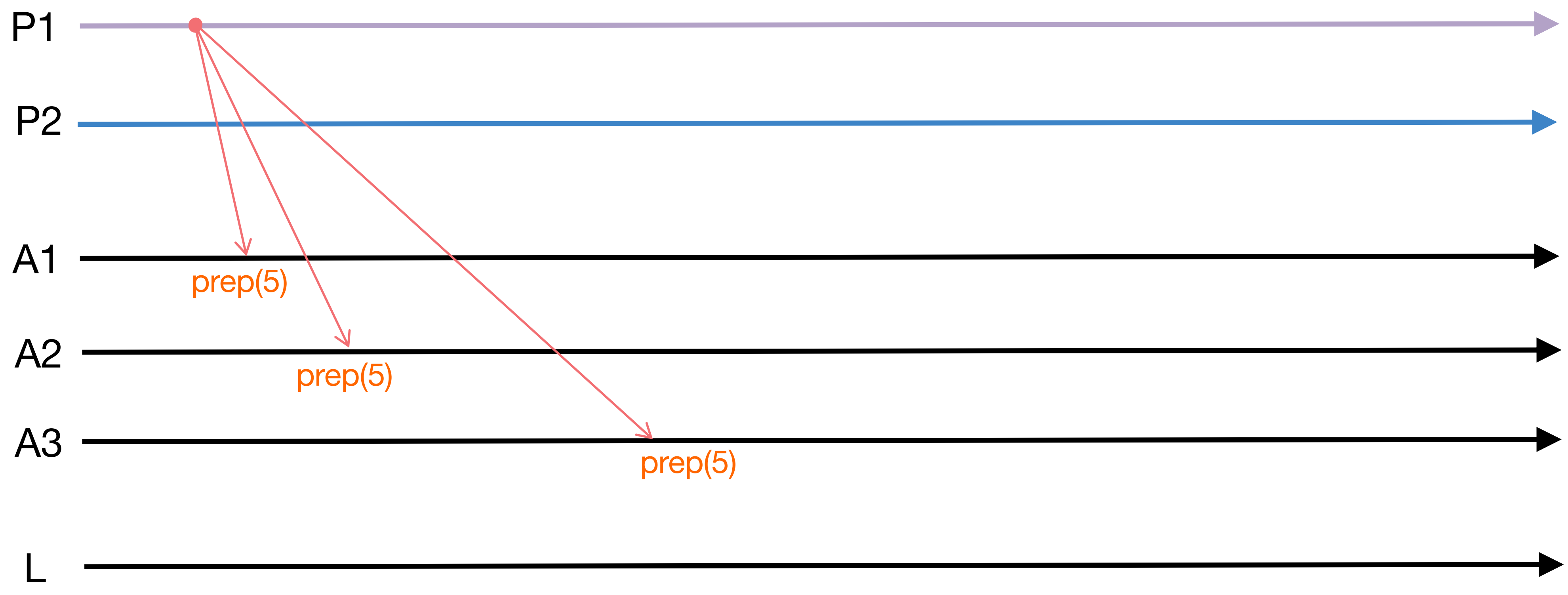


A1: Highest accept; (5, A)
Highest prepare: 13

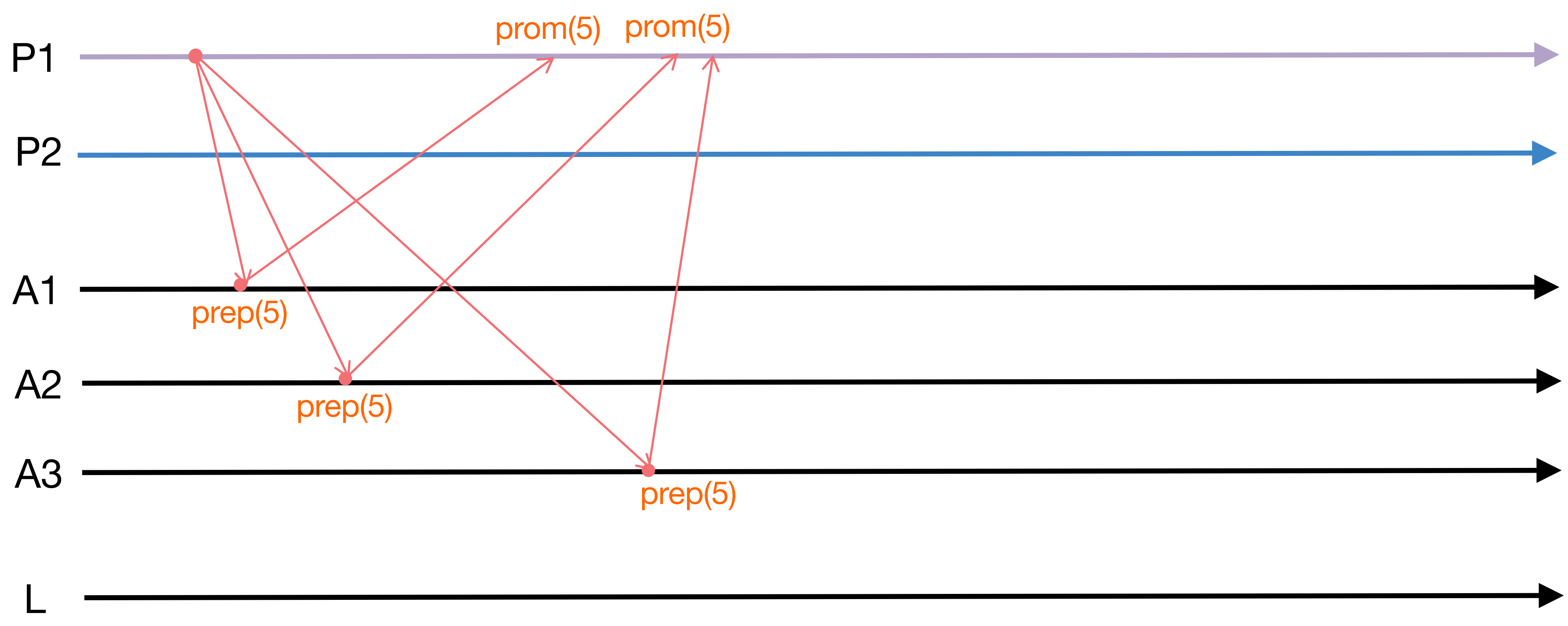
More examples

- Two proposers
- Three acceptors
- One Learner

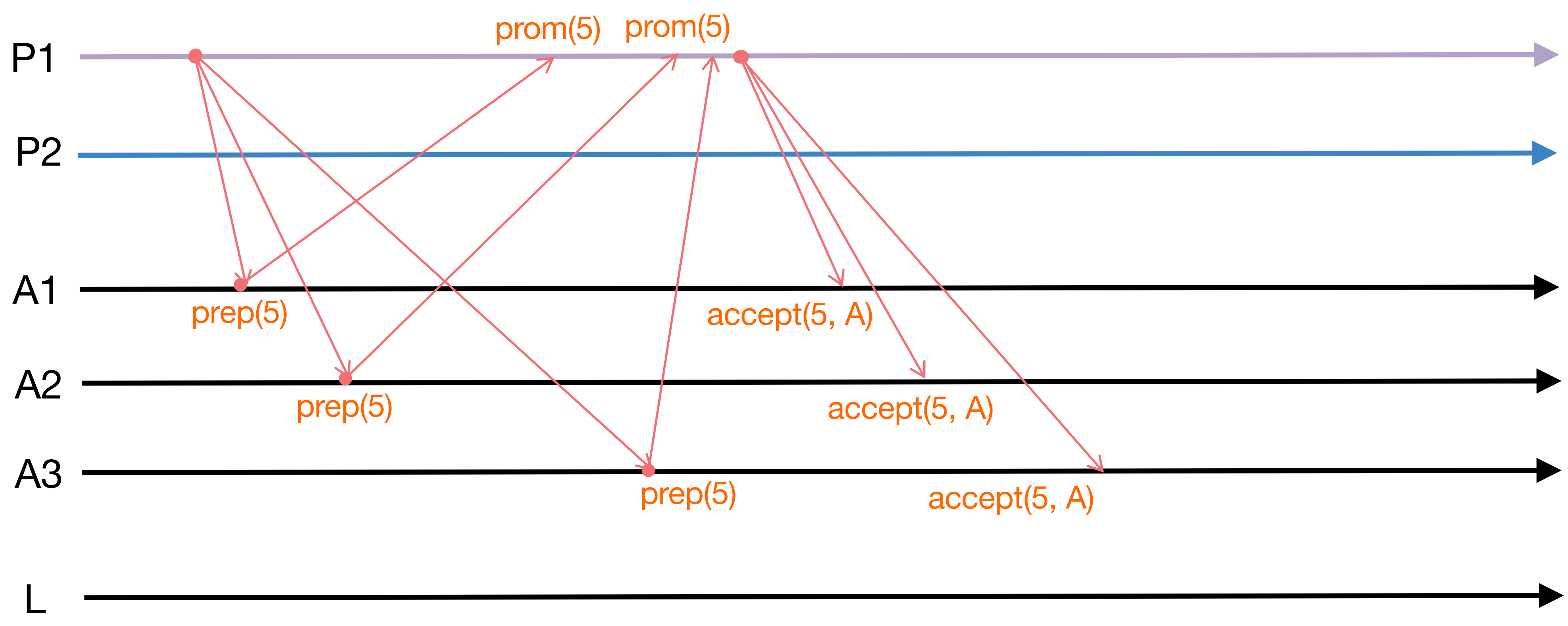
Example: P1 want to propose value A



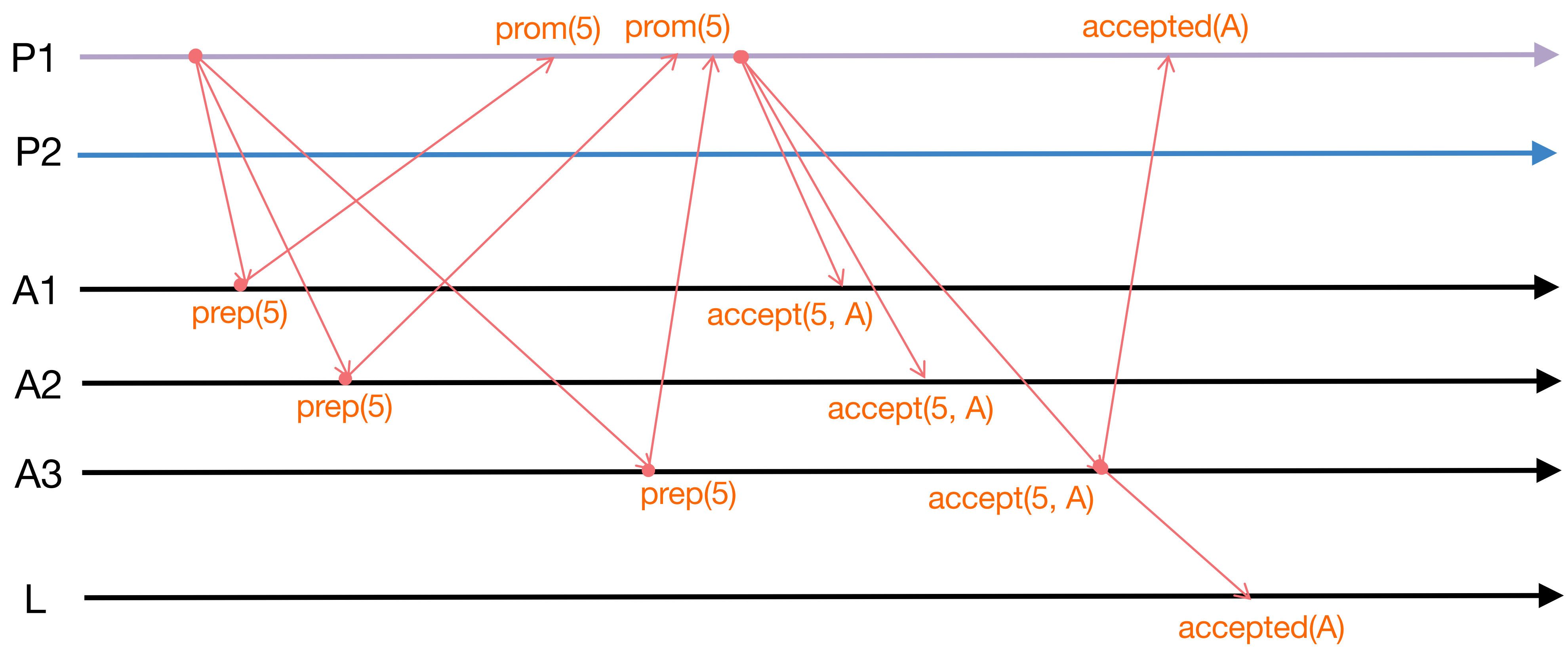
Example: P1 want to propose value A



Example: P1 want to propose value A



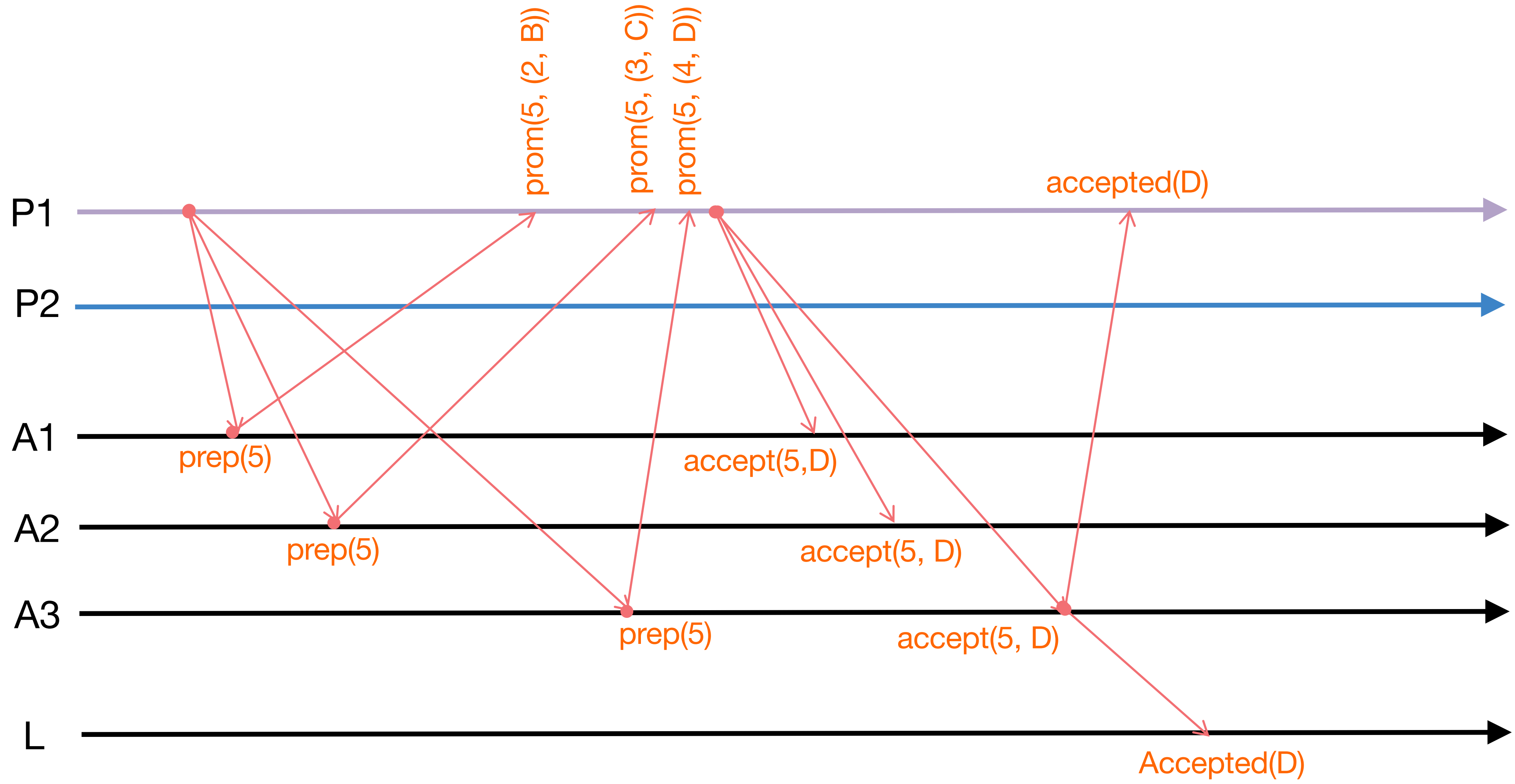
Example: P1 want to propose value A



Example

- Same scenario
  - But notes already made promises

Example

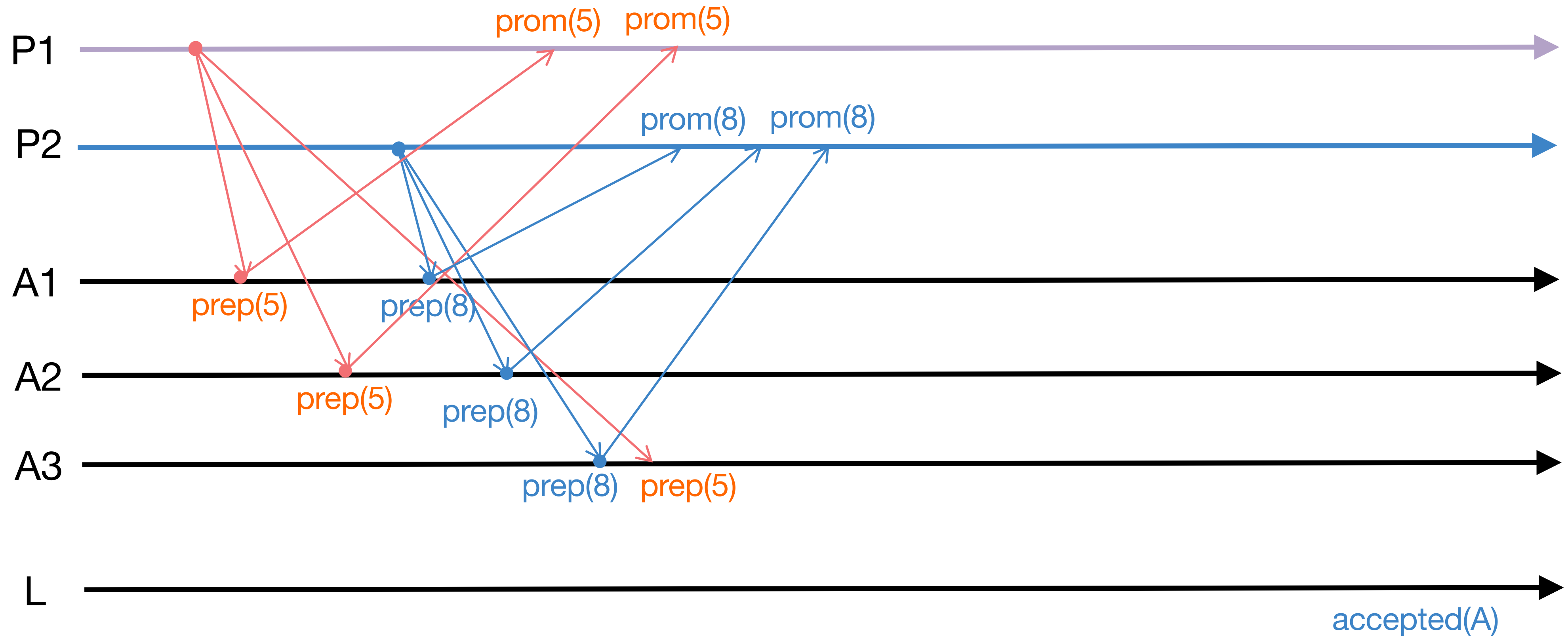


prom(5, (2, B))
prom(5, (3, C))
prom(5, (4, D))
accepted(D)
P1
P2
A1
prep(5)
accept(5,D)
A2
prep(5)
accept(5, D)
A3
prep(5)
accept(5, D)
L
Accepted(D)

Example

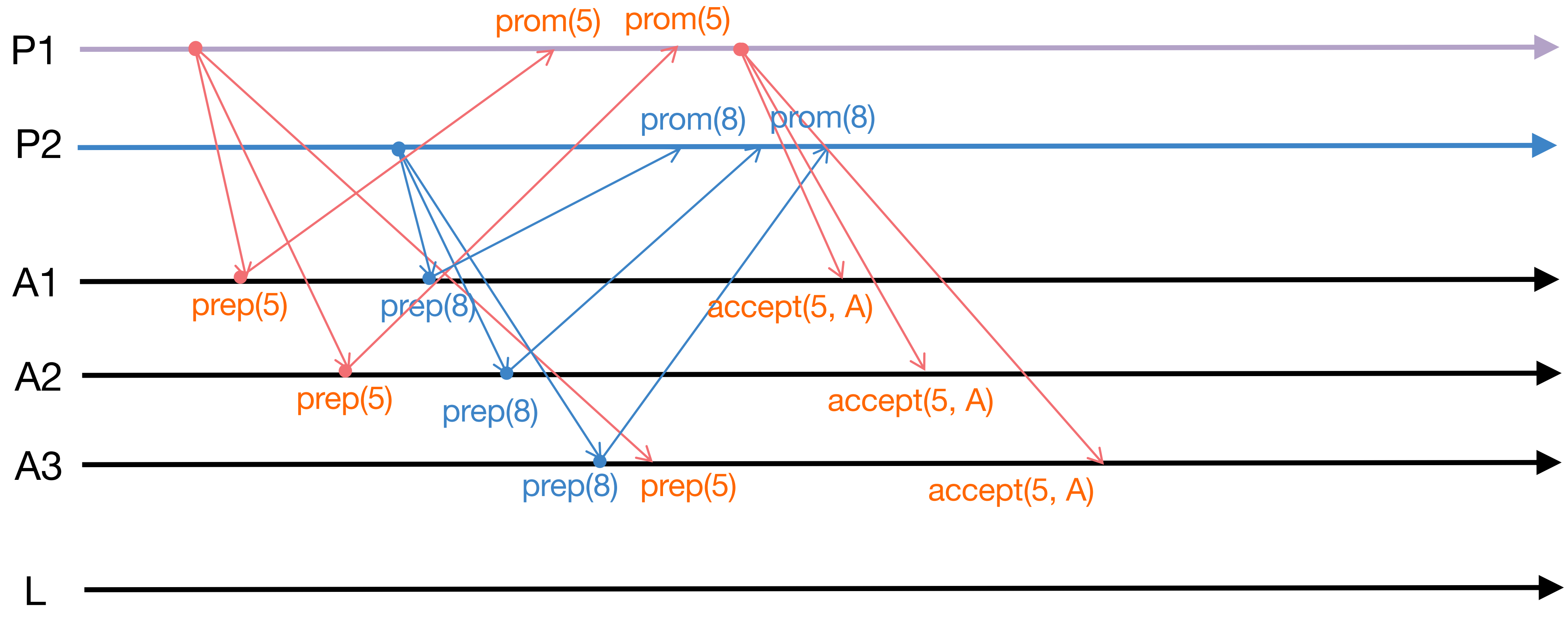
- Two propsers
  - P1 wants A, and P2 wants B

Example: P1 wants A, and P2 wants B



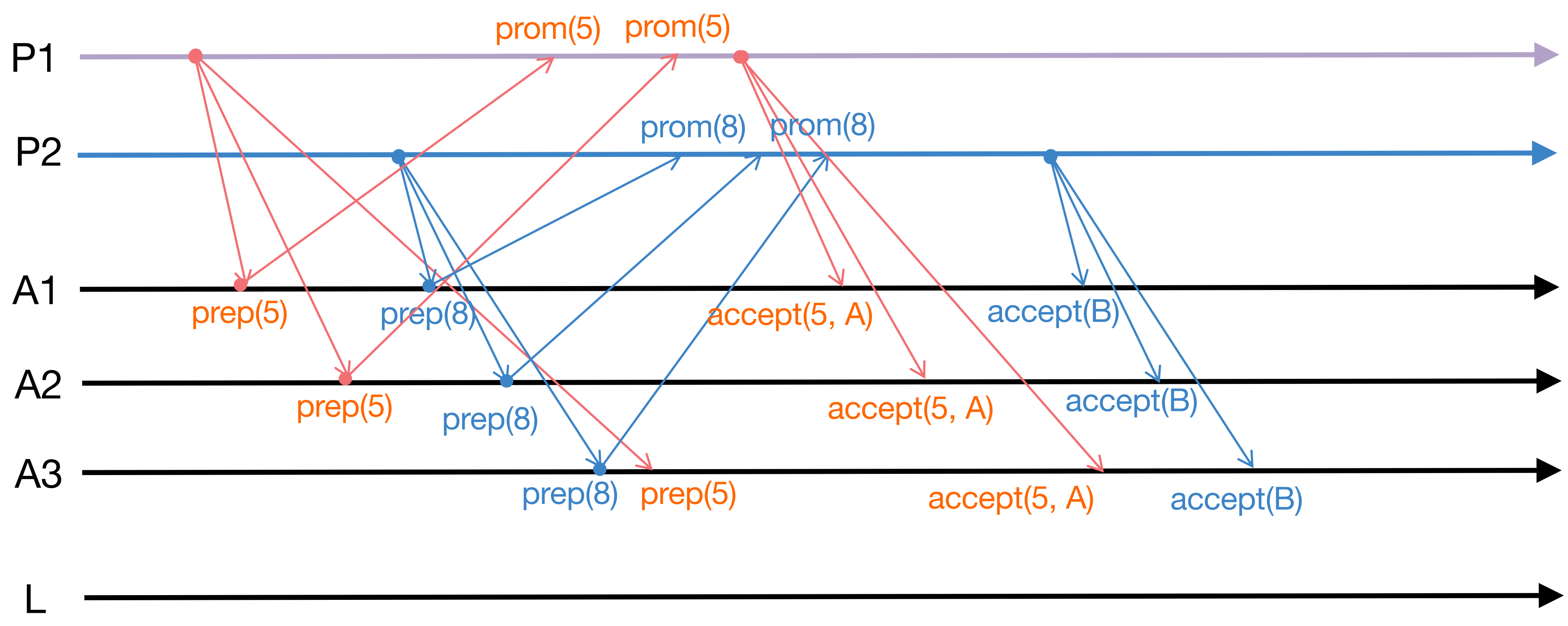
Nodes move to ballot ID 8, as it is higher than 5. P1 does not know yet…

Example: P1 wants A, and P2 wants B

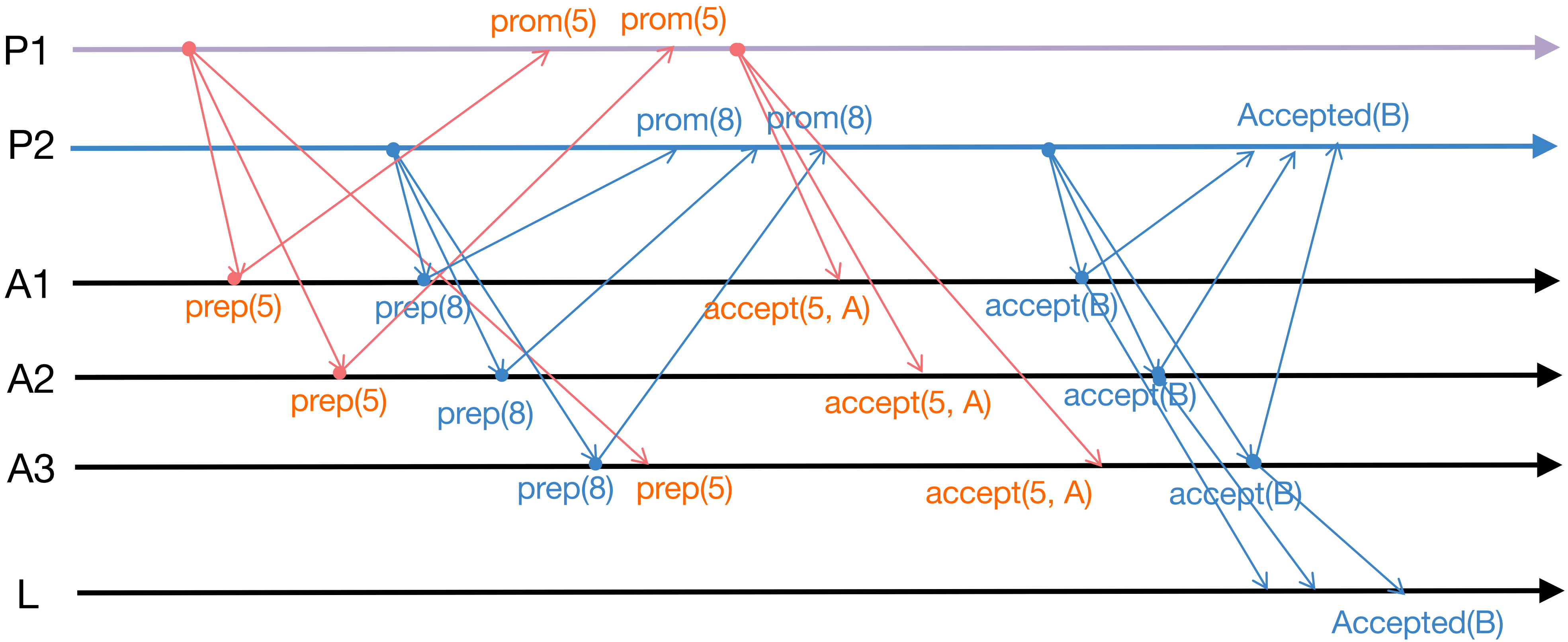


Nodes move to ballot ID 8, as it is higher than 5.
P1 does not know yet, so it will send out accepts.
But Acceptors will not reply…

Example: P1 wants A, and P2 wants B



Example: P1 wants A, and P2 wants B



Others

- In practice
  - send NACKs if not accepting a promise
  - To avoid timeouts

- Promise IDs should increase slowly
  - Otherwise too much too converge
  - Solution: different ID spaces for proposers

Next Time

- Recap (lecture slot)
- Project Presentations (lab slot)

- Questions?

- Inspired from / based on slides from
  - Jason Madden, Mehmet H. Gunes
  - Johannes Kofler
  - Terence Spies
  - And many others