Consensus is possible !
Paxos

Etienne Renault

2 octobre 2020

https://www.lrde.epita.fr/~renault/teaching/algorep/

A Word on Paxos

- Is Paxos hard ?
  ⇒ Not overly complex
- A troubled history
  ⇒ L. Lamport waited 10 years before paper accepted for publication (1998)
  ⇒ Build on work by Lynch and Liskov
  ⇒ Proved accidentally by Lamport
- **Altruism** : goal is to reach consensus, not "win"

Intuition

- You are with a group of friends and decide to go diner
- **Constraint** :
  - The whole group has to agree “gladines” or “pizza”
  - No leader in the group
  - Everybody is hungry : you have to terminate
  - Use person-to-person communication (yelling is useless)

Single Acceptor (bad solution)

A single person (acceptor) choses the value.

What if this person leaves the group (the acceptor crashes) ?

Single Acceptor (bad solution)

A single person (acceptor) choses the value.

What if this person leaves the group (the acceptor crashes) ?

Solution

Quorum-based solution ⇒ the value is chosen among a majority of person (acceptors)

Problem : Split votes

What if acceptors accept only the first received values ?

Problem : Split votes

What if acceptors accept only the first received values ?

Lets us consider 5 processes :

- P1, P2 accepts red
- P3, P4 accepts blue
- P5 accepts green

Problem : Split votes

What if acceptors accept only the first received values ?

Lets us consider 5 processes :

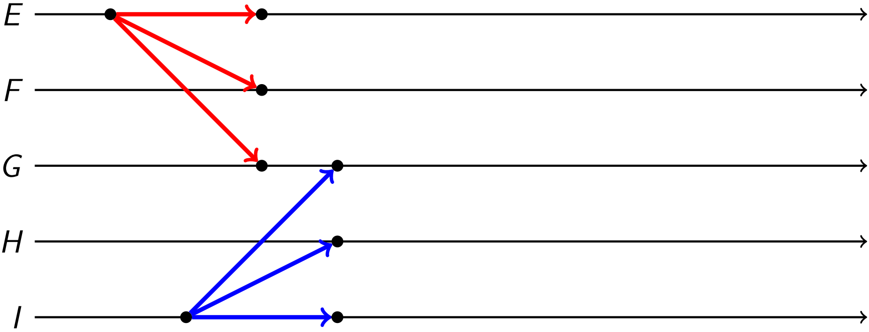
- P1, P2 accepts red
- P3, P4 accepts blue
- P5 accepts green

Acceptor might sometime change their mind in order to reach majority.

Toward a solution

Acceptors must sometitme accept multiple (different) values

Problems : Conflicting choices



If an acceptor accept all values it receives, multiple majorities can emerge !

Multiple Phases requirement

Multiple Phases (two) are mandatory

- What's happening ? *Need to ask the majority*
- Let's go for pizza ! *The majority wins*

Multiple Phases requirement

Multiple Phases (two) are mandatory

- What's happening ? *Need to ask the majority*
- Let's go for pizza ! *The majority wins*

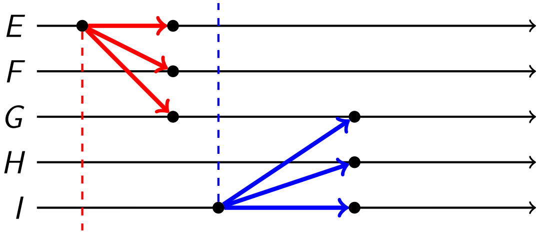
Remark

You cannot have two overlapping majority sets in a group of objects
$\Rightarrow 2m + 1$ processes required to tolerate m faults

2-phases protocol

Solution

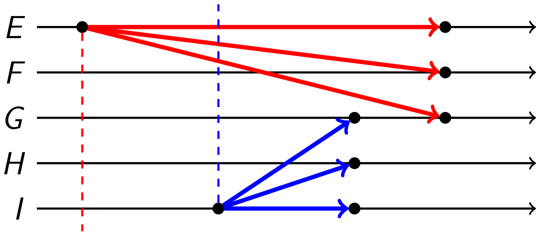
A process must check proposed values before submitting a new one !



Limitations

Limitations

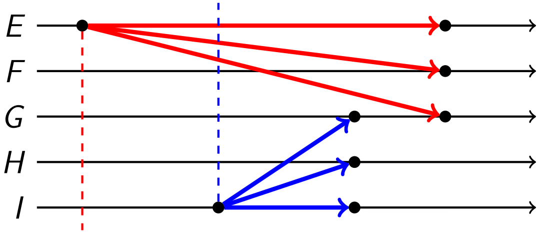
When checking proposed values, some may be in transit !



Limitations

Limitations

When checking proposed values, some may be in transit !



Proposals must be ordered, and old one must be rejected

Need for proposal numbers

Each proposal must be identified uniquely

- Maintain a “round number” (the largest round number seen so far)
- Generate a new proposal number by
  (1) Incrementing the round number
  (2) concatenate with server ID (lower bits so it’s unique)

round number must be stored on disk in case of crash/recovery

Basic Paxos

Two phases approach :

- Phase 1 : Broadcast **Prepare**
  - Find about any chosen values
  - Block older proposal that have not yet been completed
- Phase 2 : Broadcast **Accept**
  - Ask acceptors to accept a specific value

Conceptual Roles in Paxos

A process can have three conceptual roles

- **Proposers** : propose values
  Job : try convince the other nodes to accept proposed values
- **Acceptors** : accept values, where a value is chosen if a majority accept
  Job : remember values proposed by proposers
- **Learners** : learn the outcome (chosen value)

Conceptual Roles in Paxos

A process can have three conceptual roles

- **Proposers** : propose values
  Job : try convince the other nodes to accept proposed values
- **Acceptors** : accept values, where a value is chosen if a majority accept
  Job : remember values proposed by proposers
- **Learners** : learn the outcome (chosen value)

In practice, a process can play any/all roles

Paxos Protocol Overview 1/2

Phase 1 :

1. **[Proposer]** Choose a proposal number *n*
2. **[Proposer]** Broadcast **prepare(n)** to all servers
3. **[Acceptor]** Response to **prepare(n)**
    - if n $>$ minProposal then minProposal $= n$
    - Return (accepted_proposal, accepted_value)
4. **[Proposer]** When responses received from majority
    - if any accepted_value returned, replace value by accepted_value for highest accepted_proposal

Paxos Protocol Overview 2/2

Phase 2 :

5. **[Proposer]** Broadcast **accept(n, value)** to all servers
6. **[Acceptor]** Response to **accept(n, value)**
   - if n >= minProposal then
     accepted_proposal = min_proposal = *n*
     accepted_value = value
   - Return (min_proposal)
7. **[Proposer]** When responses received from majority
   - Any objection (result > n) ? restart
   - Otherwise, value is chosen

Paxos Protocol Overview 2/2

Phase 2 :

5. **[Proposer]** Broadcast **accept(n, value)** to all servers
6. **[Acceptor]** Response to **accept(n, value)**
   - if n >= minProposal then
     accepted_proposal = min_proposal = *n*
     accepted_value = value
   - Return (min_proposal)
7. **[Proposer]** When responses received from majority
   - Any objection (result > n) ? restart
   - Otherwise, value is chosen

Important : Stability Remark

accepted_proposal, min_proposal and accepted_value must be stored on disk.

Example 1

Later proposal

Previous value already chosen ⇒ new proposer will find and use it.

Example 1

Later proposal

Previous value already chosen $\Rightarrow$ new proposer will find and use it.

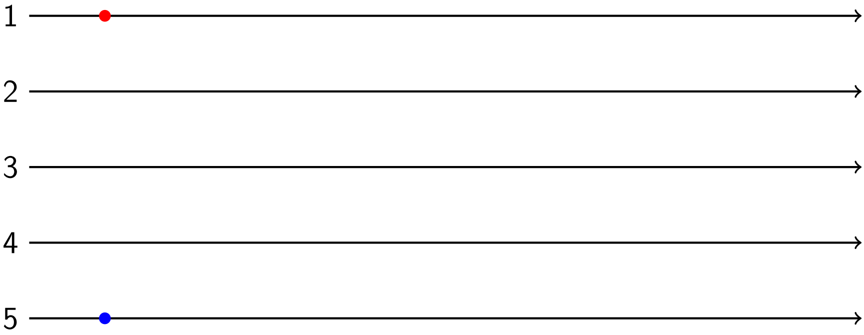
P_1 suggests X and P_5 suggests Y

Example 1

Later proposal

Previous value already chosen ⇒ new proposer will find and use it.

P_1 suggests X and P_5 suggests Y

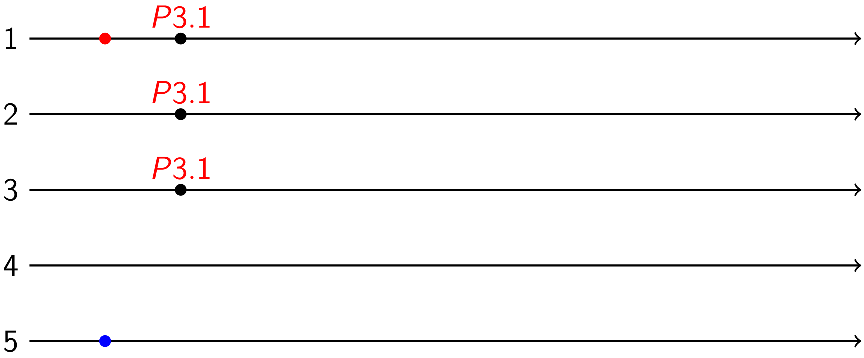


Example 1

Later proposal

Previous value already chosen ⇒ new proposer will find and use it.

P_1 suggests X and P_5 suggests Y

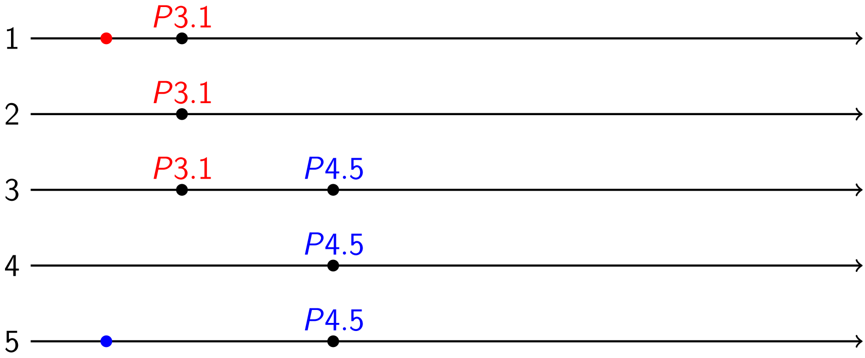


Example 1

Later proposal

Previous value already chosen ⇒ new proposer will find and use it.

P_1 suggests X and P_5 suggests Y

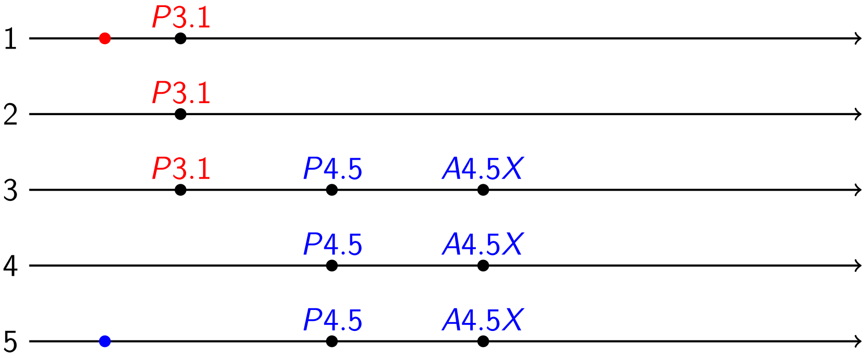


Example 1

Later proposal

Previous value already chosen ⇒ new proposer will find and use it.

P_1 suggests X and P_5 suggests Y



Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

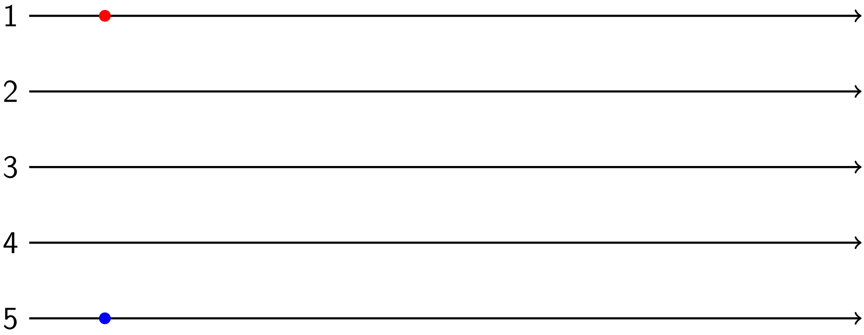
P_1 suggests X and P_5 suggests Y

Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

P_1 suggests X and P_5 suggests Y

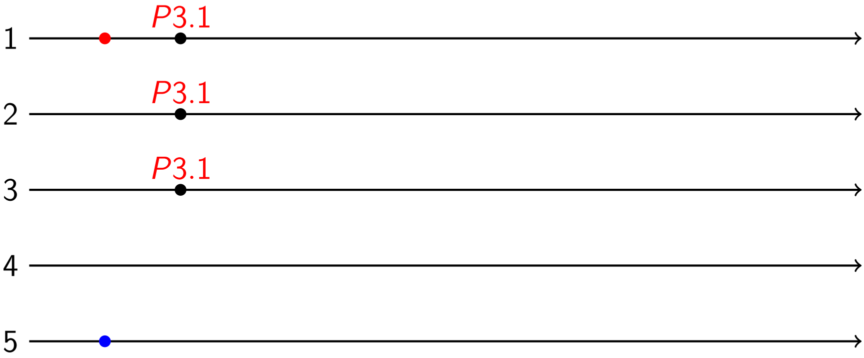


Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

P_1 suggests X and P_5 suggests Y

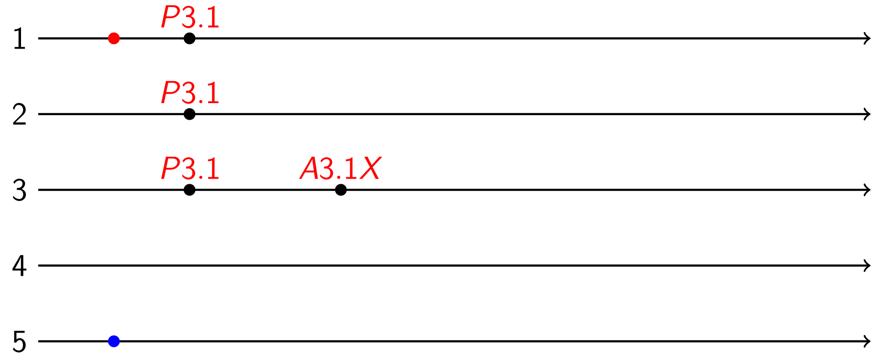


Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

P_1 suggests X and P_5 suggests Y

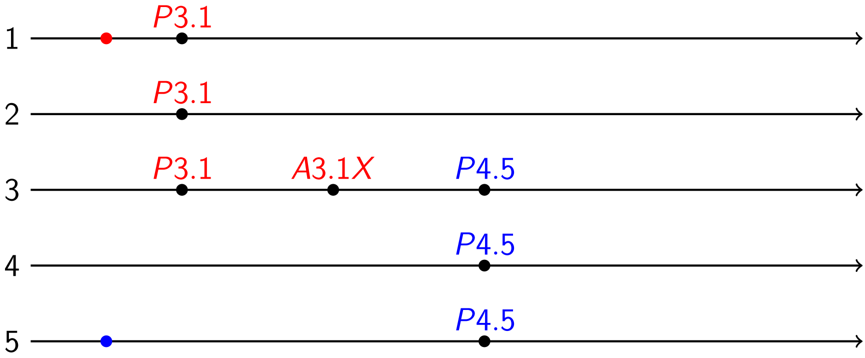


Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

P_1 suggests X and P_5 suggests Y

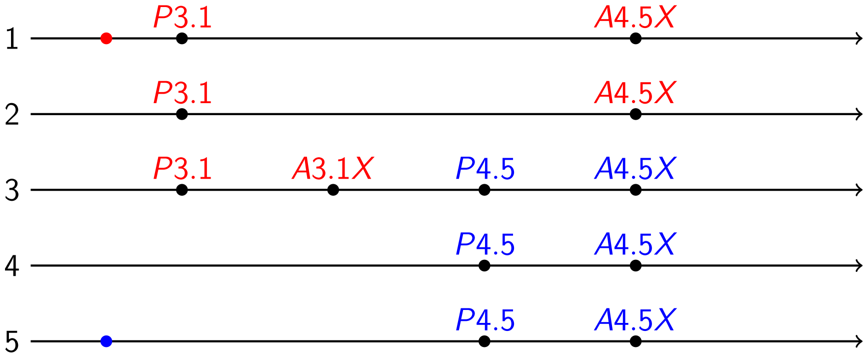


Example 2

Later proposal

Previous value not already chosen BUT new proposer see it

P_1 suggests X and P_5 suggests Y



Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
$\Rightarrow$ Block older proposal and new phase 1 relaunched

Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
$\Rightarrow$ Block older proposal and new phase 1 relaunched

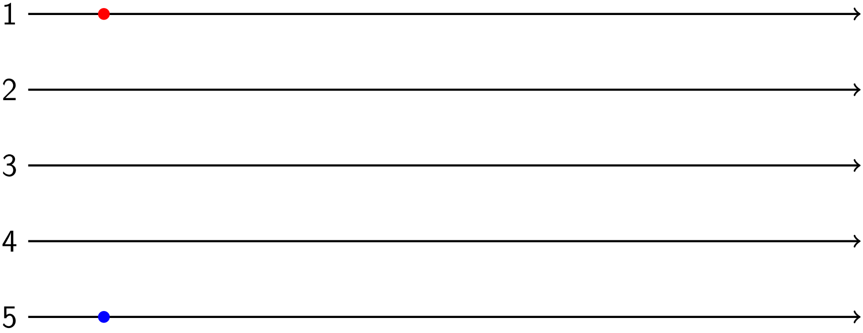
P_1 suggests X and P_5 suggests Y

Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
$\Rightarrow$ Block older proposal and new phase 1 relaunched

P_1 suggests X and P_5 suggests Y

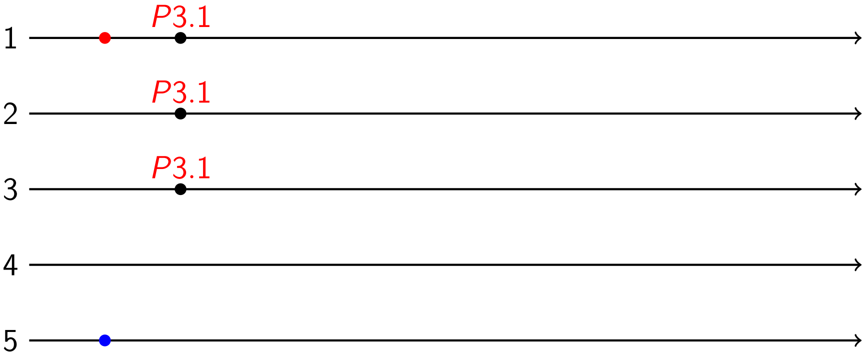


Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
⇒ Block older proposal and new phase 1 relaunched

P_1 suggests X and P_5 suggests Y

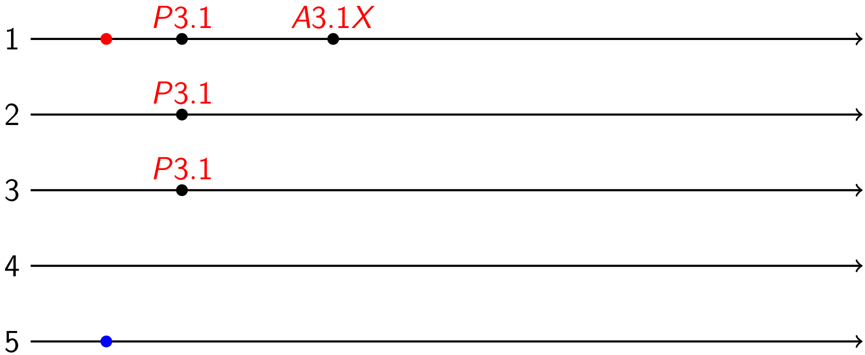


Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
$\Rightarrow$ Block older proposal and new phase 1 relaunched

P_1 suggests X and P_5 suggests Y

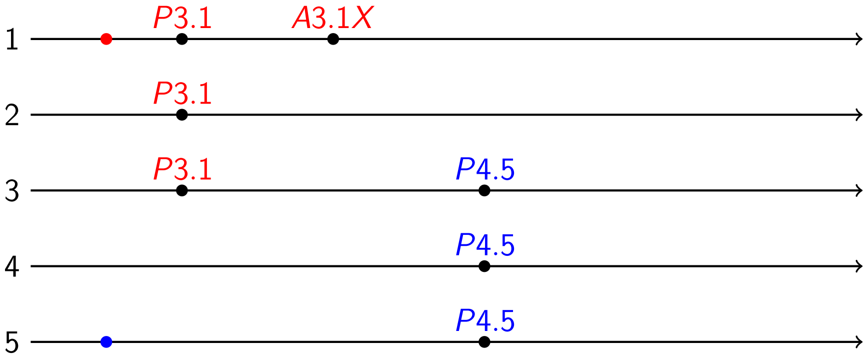


Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
⇒ Block older proposal and new phase 1 relaunched

P_1 suggests X and P_5 suggests Y

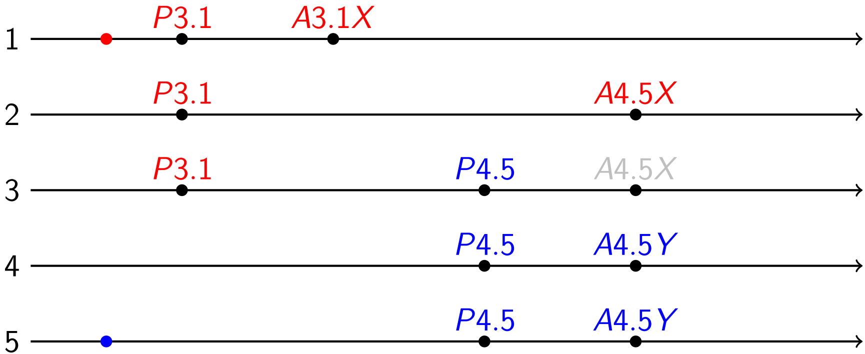


Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
⇒ Block older proposal and new phase 1 relaunched

P_1 suggests X and P_5 suggests Y

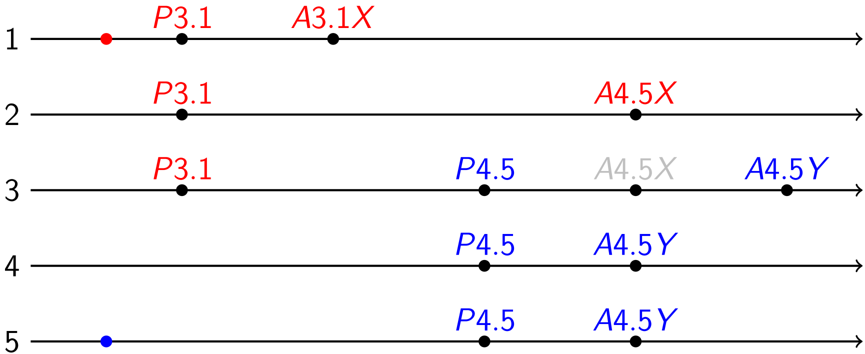


Example 3

Later proposal

Previous value not already chosen new proposer doesn't see it
⇒ Block older proposal and new phase 1 relaunched

P_1 suggests X and P_5 suggests Y



Liveness

Competing processes can livelock !

Liveness

Competing processes can livelock !

P_1 suggests X and P_5 suggests Y

Liveness

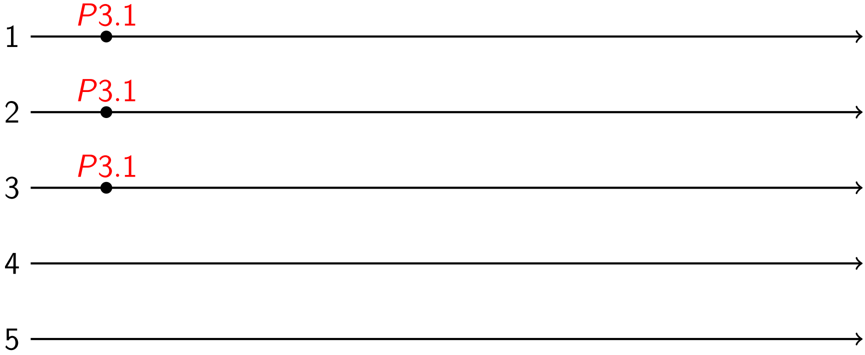
Competing processes can livelock !

P_1 suggests X and P_5 suggests Y

Liveness

Competing processes can livelock !

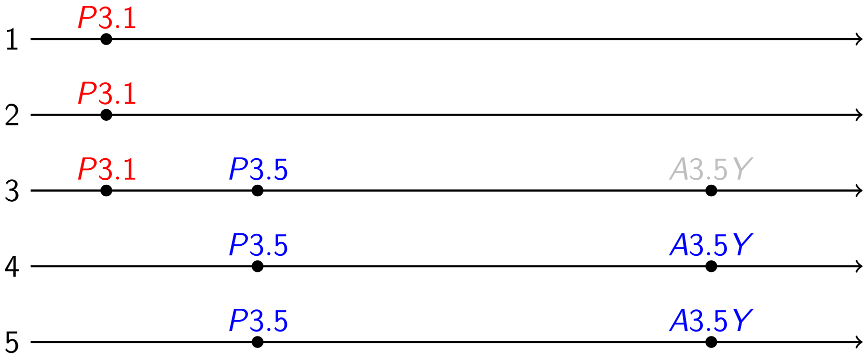
P_1 suggests X and P_5 suggests Y



Liveness

Competing processes can livelock !

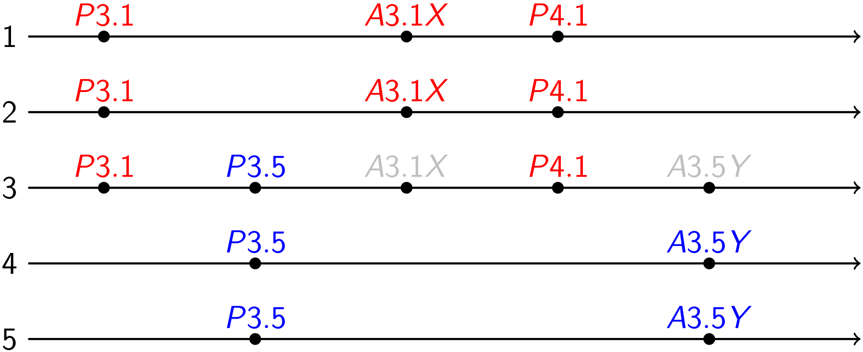
P_1 suggests X and P_5 suggests Y



Liveness

Competing processes can livelock !

P_1 suggests X and P_5 suggests Y



Solutions

Randomized delays before restarting
$\Rightarrow$ Give a chance to a process to finish !

Multi-paxos will use leader election instead

Multi-Paxos

Goal

Create a replicated log.

Main idea :

- Use a collection of Paxos algorithms
- Add index to Prepare and Accept
  This index selects entry in log

Example

1. Client send command to a server
2. Server uses Paxos to choose command as value for log entry
3. Server waits for previous entries to be applied then applied command
4. Server returns result to client

Example

1. Client send command to a server
2. Server uses Paxos to choose command as value for log entry
3. Server waits for previous entries to be applied then applied command
4. Server returns result to client

Multi-paxos not specified precisely in litterature !