Consensus and Paxos

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS 240: Computing Systems and Concurrency
Lecture 12

Marco Canini

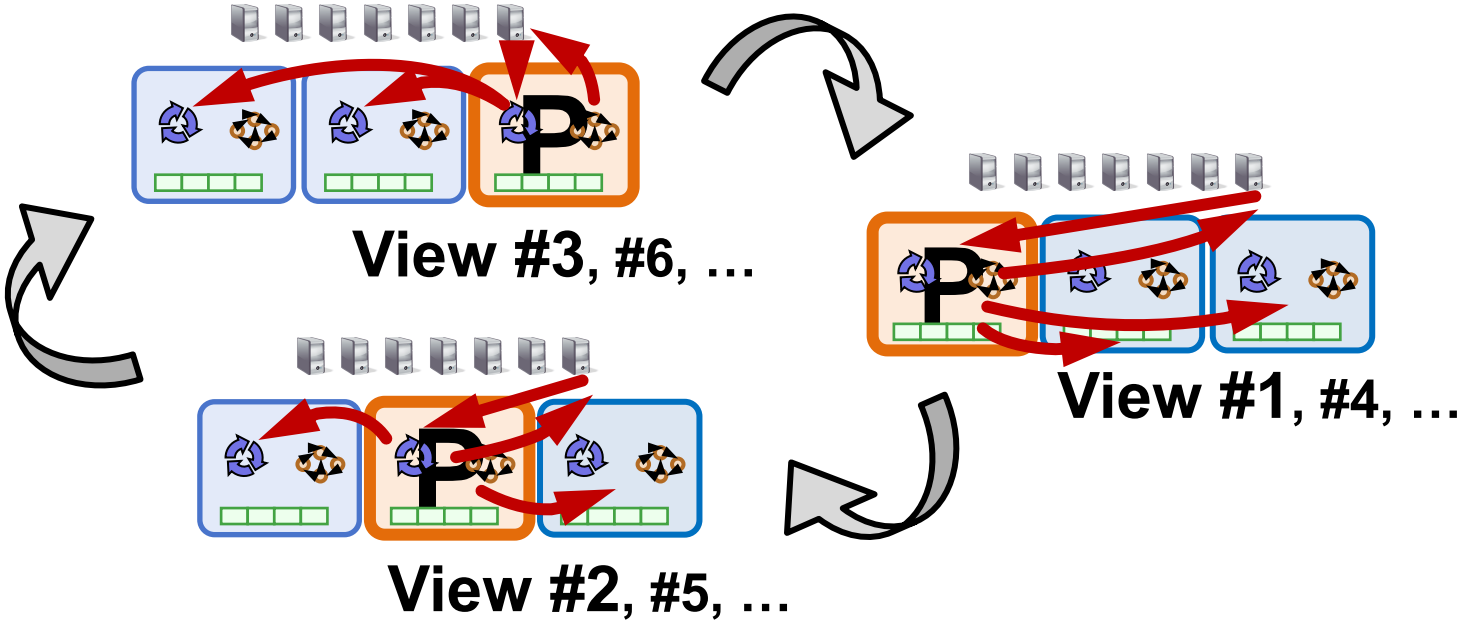
Credits: Michael Freedman and Kyle Jamieson developed much of the original material.

Today

1. **Consensus in distributed systems**
2. FLP impossibility
3. Paxos

Recall the use of Views

- Let different replicas assume role of primary over time
- System moves through a sequence of views
- **How do the nodes agree on view / primary?**



Consensus

Definition:

1. A general agreement about something

2. An idea or opinion that is shared by all the people in a group

Consensus

Given a set of processes, each with an initial value:

- **Termination:** All non-faulty processes eventually decide on a value
- **Agreement:** All processes that decide do so on the same value
- **Validity:** The value that has been decided must have been proposed by some process

Consensus used in systems

Group of servers attempting:

- Make sure all servers in group receive the same updates in the same order as each other
- Maintain own lists (views) on who is a current member of the group, and update lists when somebody leaves/fails
- Elect a leader in group, and inform everybody
- Ensure mutually exclusive (one process at a time only) access to a critical resource like a file

Can we achieve consensus?

Step one: Define your system model

- Network model:
  - Synchronous (time-bounded delay) or asynchronous (arbitrary delay)
  - Reliable or unreliable communication
  - Unicast or multicast communication
- Node failures:
  - Crash (correct/dead) or Byzantine (arbitrary)

(Left options indicate an “easier” setting.)

Step one: Define your system model

- Network model:
  - Synchronous (time-bounded delay) or asynchronous (arbitrary delay)
  - Reliable or unreliable communication
  - Unicast or multicast communication
- Node failures:
  - Crash (correct/dead) or Byzantine (arbitrary)

(Left options indicate an “easier” setting.)

Consensus is impossible

Today

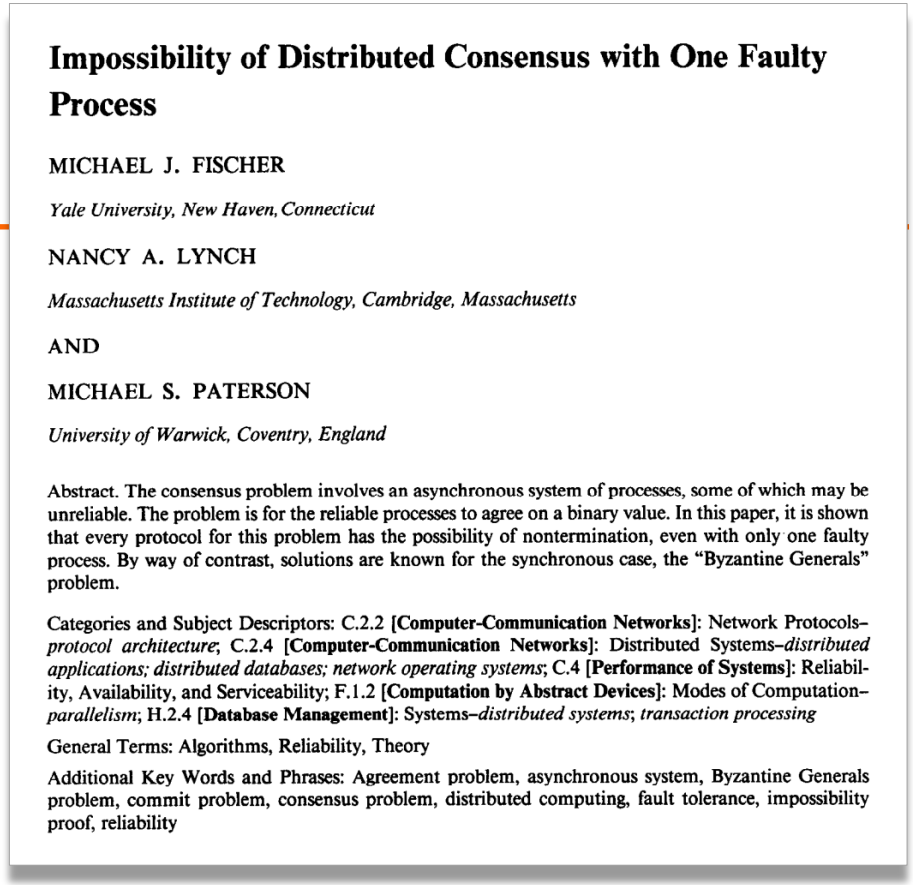
1. Consensus in distributed systems

2. **FLP impossibility**

3. Paxos

"FLP" result

- No deterministic 1-crash-robust consensus algorithm exists with asynchronous communication



Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

Categories and Subject Descriptors: C.2.2 **[Computer-Communication Networks]**: Network Protocols–*protocol architecture*; C.2.4 **[Computer-Communication Networks]**: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 **[Performance of Systems]**: Reliability, Availability, and Serviceability; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation–*parallelism*; H.2.4 **[Database Management]**: Systems–*distributed systems*; *transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

FLP's weak assumptions

- Only 1 failure
  - Also impossible for more failures
- For "weak" consensus (only some process needs to decide)
  - Also impossible for real consensus
- For reliable communication
  - Also impossible for unreliable communication
- For only two states: 0 and 1
  - Also impossible for more failures
- For crash failures
  - Also impossible for Byzantine failures

FLP's strong assumptions

- Deterministic actions at each node
- Asynchronous network communication
- All "runs" must eventually achieve consensus

Main technical approach

- Initial state of system can end in decision “0” or “1”

- Consider 5 processes, each in some initial state

[1,1,1,1,1] → 1
[1,1,1,1,0] → ?
[1,1,1,0,0] → ?
[1,1,0,0,0] → ?
[1,0,0,0,0] → 0

Must exist two configurations here which differ in decision

Main technical approach

- Initial state of system can end in decision “0” or “1”

- Consider 5 processes, each in some initial state

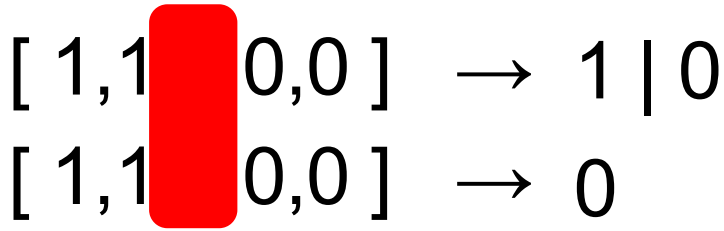
[1,1,1,1,1] → 1
[1,1,1,1,0] → 1
[1,1,1 0,0] → 1
[1,1,0 0,0] → 0
[1,0,0,0,0] → 0

Assume decision differs between these two processes

Main technical approach

- Goal: Consensus holds in face of 1 failure

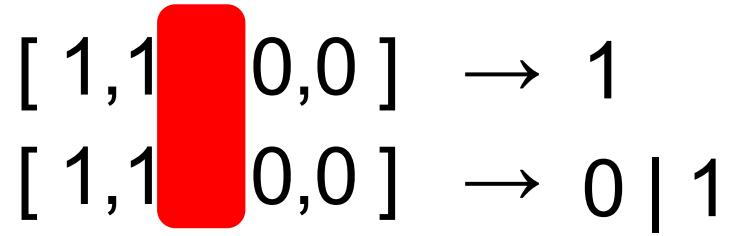
One of these configurations must be "bi-valent" (i.e., undecided): Both futures possible



Main technical approach

- Goal: Consensus holds in face of 1 failure

One of these configurations must be "bi-valent" (i.e., undecided): Both futures possible



- Inherent non-determinism from asynchronous network
- Key result: All bi-valent states can remain in bi-valent states after performing some work

Staying bi-valent forever

1. System thinks process *p* failed, adapts to it…
2. But no, *p* was merely slow, not failed…
   (Can't tell the difference between slow and failed.)
3. System think process *q* failed, adapts to it…
4. But no, *q* was merely slow, not failed…
5. Repeat ad infinitum …

Consensus is impossible

But, we achieve consensus all the time…

FLP's strong assumptions

- Deterministic actions at each node
  - Randomized algorithms can achieve consensus
- Asynchronous network communication
  - Synchronous or even partial synchrony is sufficient
- All "runs" must eventually achieve consensus
  - In practice, many "runs" achieve consensus quickly
  - In practice, "runs" that never achieve consensus happen vanishingly rarely
    - Both are true with good system designs

Consensus is possible

With Paxos!

Today

1. Consensus in distributed systems
2. FLP impossibility
3. **Paxos**

Consensus

Given a set of processes, each with an initial value:

- **Termination:** All non-faulty processes eventually decide on a value ← Good thing that eventually should happen
- **Agreement:** All processes that decide do so on the same value ← Bad thing that should never happen
- **Validity:** The value that has been decided must have been proposed by some process ← Bad thing that should never happen

Recall: Safety vs liveness properties

Safety (bad things never happen)

Liveness (good things eventually happen)

Paxos properties

Safety

- Only a single value is chosen ← **agreement**
- Only chosen values are learned by processes
- Only a proposed value can be chosen ← **validity**

Liveness

- Some proposed value eventually chosen if fewer than half of processes fail
- If value is chosen, a process eventually learns it ← **termination**

Paxos' safety and liveness

- Paxos is always safe

- Paxos is very often live
  - But not always live

Roles of a process

- Three conceptual roles
  - Proposers propose values
  - Acceptors accept values, where chosen if majority accept
  - Learners learn the outcome (chosen value)

- In reality, a process can play any/all roles

Strawman

- 3 proposers, 1 acceptor
  - Acceptor accepts first value received
  - No liveness on failure
- 3 proposals, 3 acceptors
  - Accept first value received, acceptors choose common value known by majority
  - But no such majority is guaranteed

Paxos

- Each acceptor accepts multiple proposals
  - Hopefully one of multiple accepted proposals will have a majority vote (and we determine that)
  - If not, rinse and repeat (more on this)
- How do we select among multiple proposals?
  - Ordering: proposal is tuple (proposal #, value) = (n, v)
  - Proposal # strictly increasing, globally unique
  - Globally unique?
    - Trick: set low-order bits to proposer's ID

Paxos Protocol Overview

- Proposers:
  1. Choose a proposal number n
  2. Ask acceptors if any accepted proposals with $n_a < n$
  3. If existing proposal v_a returned, propose same value (n, v_a)
  4. Otherwise, propose own value (n, v)

  Note altruism: goal is to reach consensus, not "win"

- Acceptors try to accept value with highest proposal n

- Learners are passive and wait for the outcome

Paxos Phase 1

- Proposer:
  - Choose proposal number n, send <prepare, n> to acceptors
- Acceptors:
  - If $n > n_h$
    - $n_h = n$ ← promise not to accept any new proposals n' < n
    - If no prior proposal accepted
      - Reply < promise, n, Ø >
    - Else
      - Reply < promise, n, (n_a, v_a) >
  - Else
    - Reply < prepare-failed >

Paxos Phase 2

- Proposer:
  - If receive promise from majority of acceptors,
    - Determine v_a returned with highest n_a, if exists
    - Send <accept, (n, v_a || v)> to acceptors
- Acceptors:
  - Upon receiving <accept, (n, v)>, if $n \geq n_h$,
    - Accept proposal and notify learner(s)

      $n_a = n_h = n$

      $v_a = v$

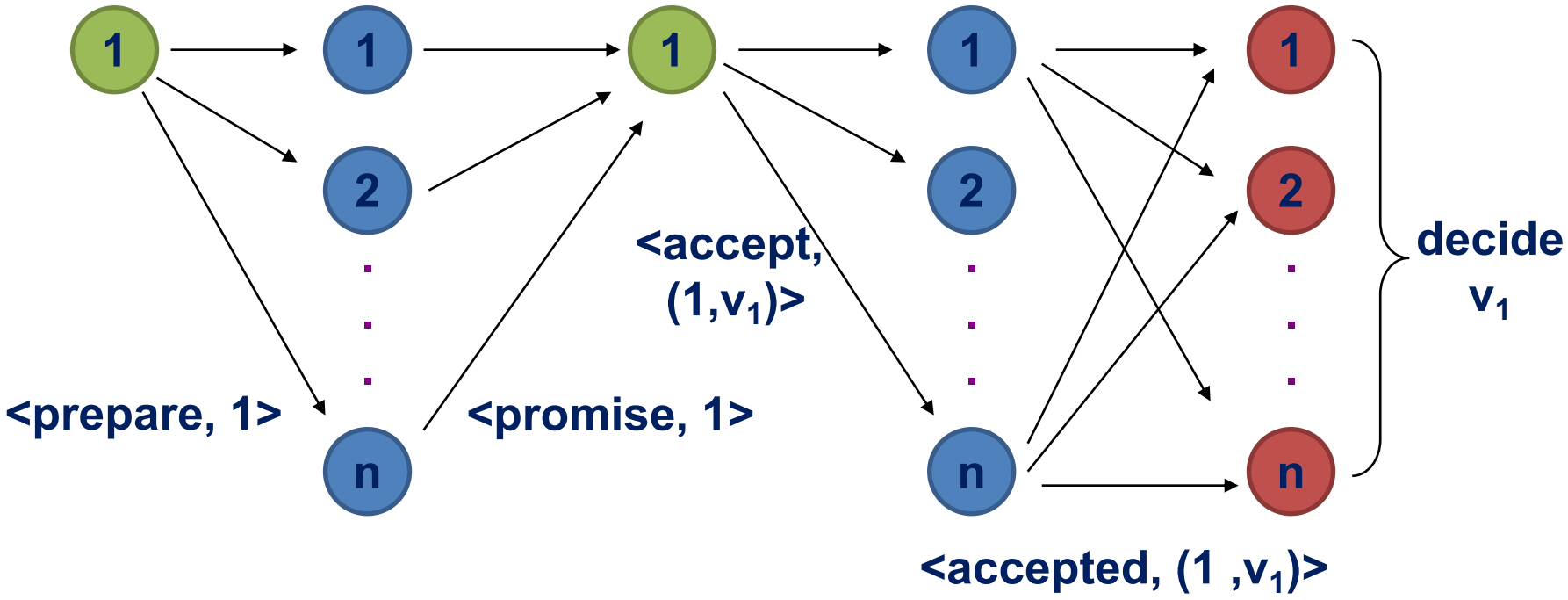
Paxos Phase 3

- Learners need to know which value chosen

- Approach #1
  - Each acceptor notifies all learners
  - More expensive

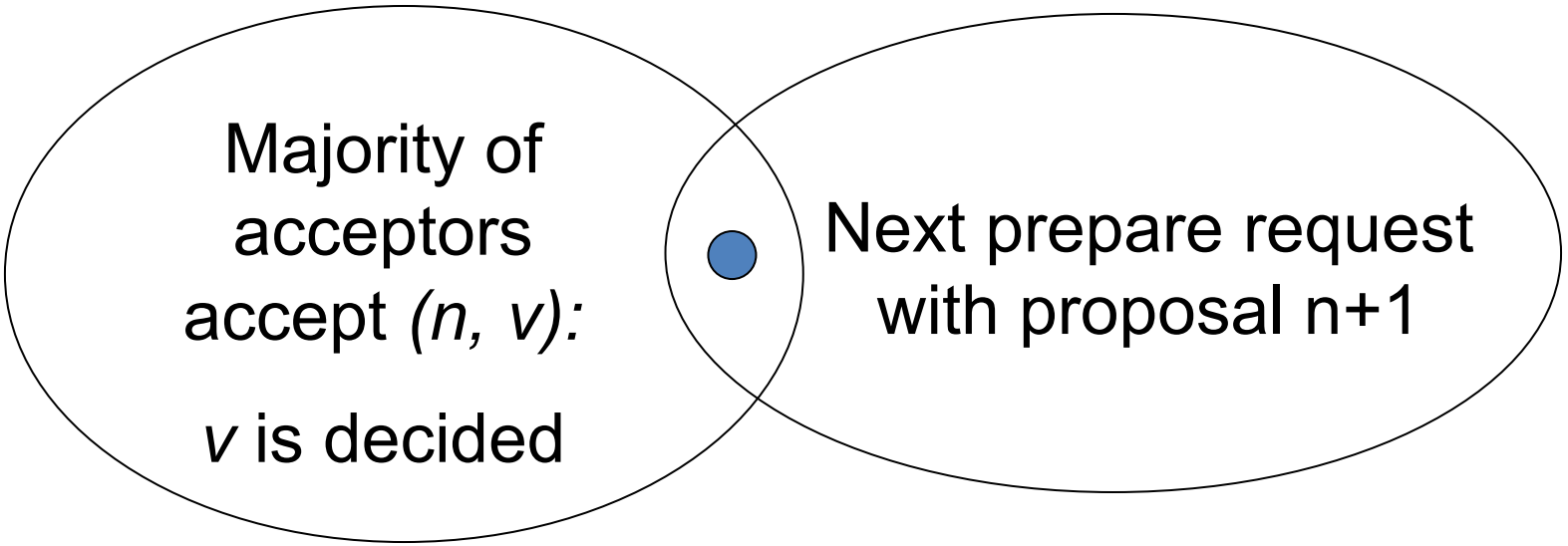
- Approach #2
  - Elect a “distinguished learner”
  - Acceptors notify elected learner, which informs others
  - Failure-prone

Paxos: Well-behaved Run

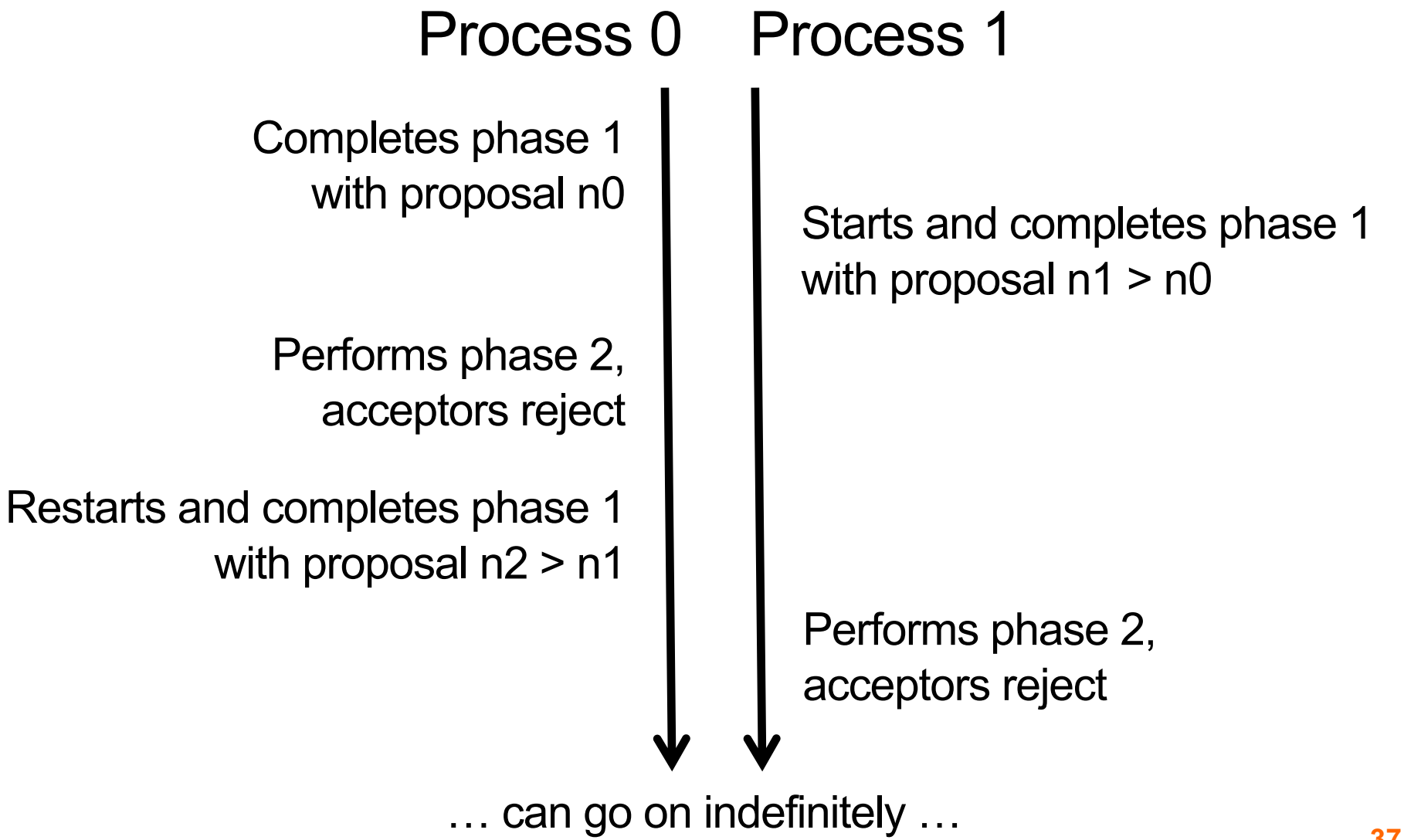


Paxos is safe

- Intuition: if proposal with value v decided, then every higher-numbered proposal issued by any proposer has value v.



Race condition leads to liveness problem



Paxos summary

- Described for a single round of consensus
- Often implemented with nodes playing all roles
- Always safe
  - Quorum intersection
- Often live
  - “FLP Scenario” prevents it from always being live
- Acceptors accept multiple values
  - But only one value is ultimately chosen
- Once a value is accepted by a majority it is chosen
- Can tolerate failures f < N / 2 (aka, 2f+1 nodes)

Flavors of Paxos

- Terminology is a mess
- Paxos loosely, and confusingly defined…
- We'll stick with
  - Basic Paxos
  - Multi-Paxos

Flavors of Paxos: Basic Paxos

- Run the full protocol each time
  - e.g., for each slot in the command log
- Takes 2 rounds until a value is chosen
- “FLP Scenario” is dueling proposers

Flavors of Paxos: Multi-Paxos

- Elect a leader and have it run the 2nd phase directly
  - e.g., for each slot in the command log
  - Leader election uses Basic Paxos
- Takes 1 round until a value is chosen
  - Faster than Basic Paxos
- "FLP Scenario" is dueling proposers during leader election
  - Rarer than Basic Paxos
- Used extensively in practice!

Consensus takeaways

- Consensus: Terminating agreement on a valid proposal
- Consensus is impossible to always achieve
  - FLP result
- Consensus is possible to achieve in practice
  - With Multi-Paxos
    - Mostly happens in a single round to the nearest quorum
    - Sometimes takes a single round to a further quorum
    - Rarely takes multiple rounds to elect a new leader and for that node to get the request accepted
    - Runs exist where no new leader is ever elected

Next topic:

Consensus protocol with group membership + leader election at core

RAFT (assignment 3)