Consensus I

FLP Impossibility, Paxos

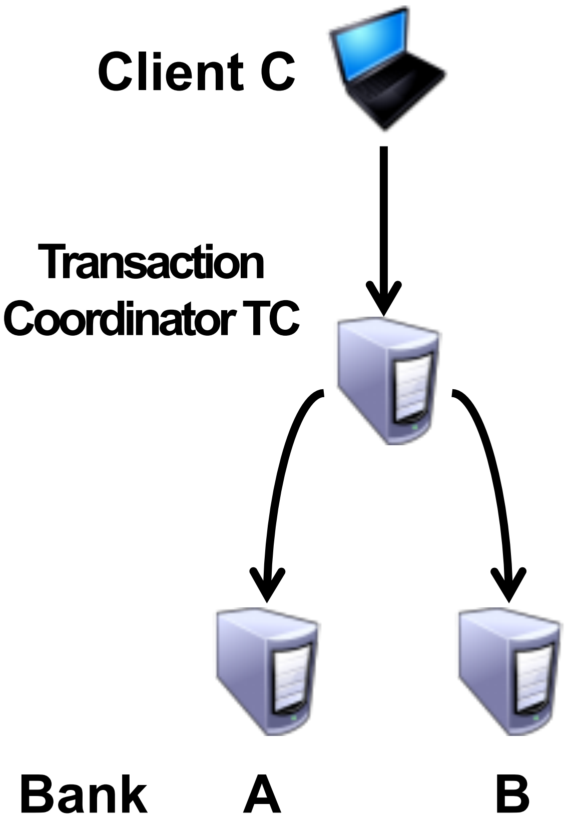
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS 240: Computing Systems and Concurrency
Lecture 8

Marco Canini

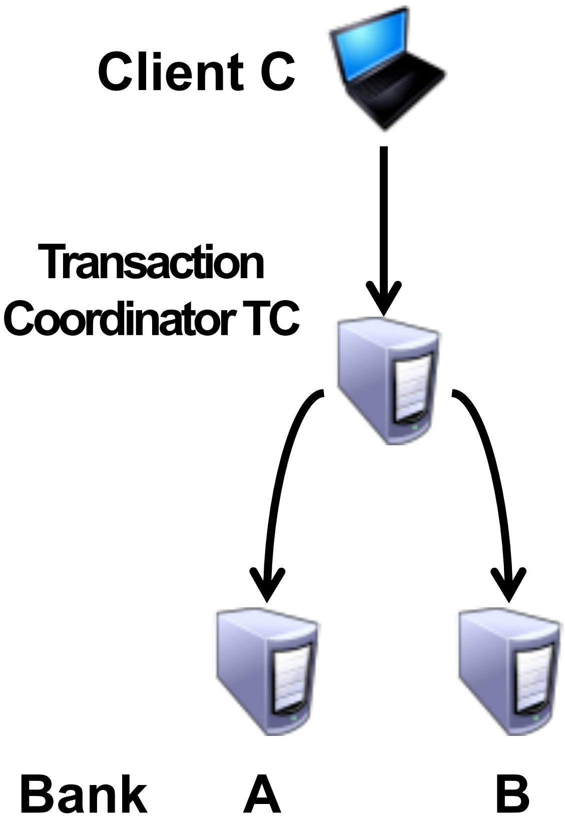
Credits: Michael Freedman and Kyle Jamieson developed much of the original material.

Recall our 2PC commit problem



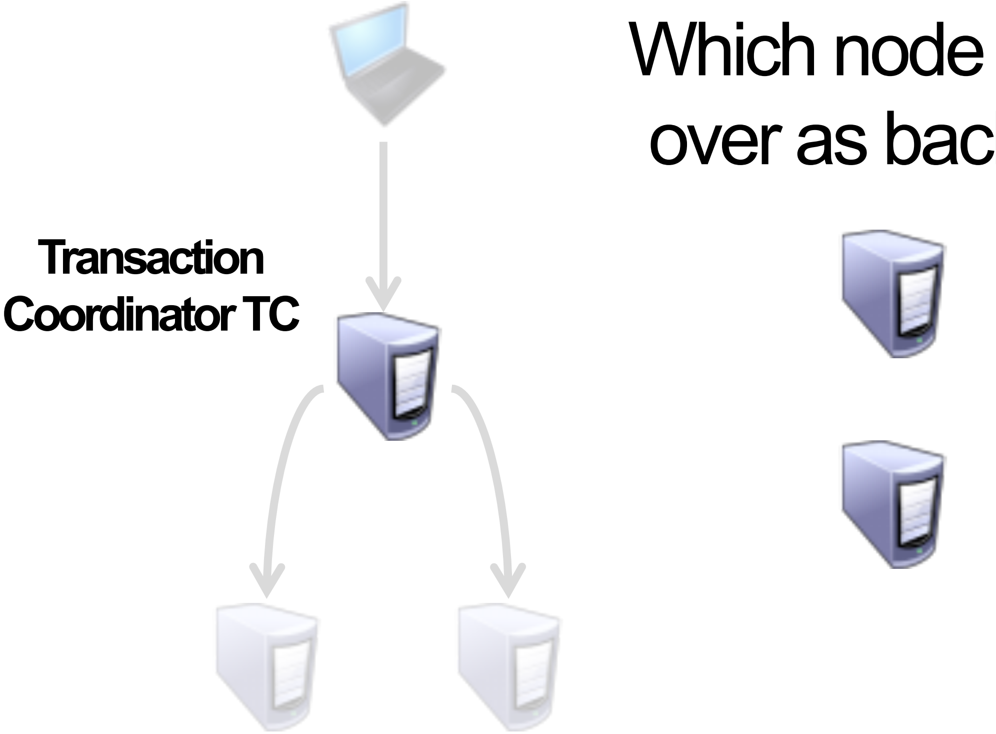
1. C → TC: *"go!"*
2. TC → A, B: *"prepare!"*
3. A, B → P: *"yes"* or *"no"*
4. TC → A, B: *"commit!"* or *"abort!"*

Recall our 2PC commit problem



- Who acts as TC?
- Which server(s) own the account of A? B?
- Who takes over if TC fails? What about if A or B fail?

Doing failover "correctly" isn't easy

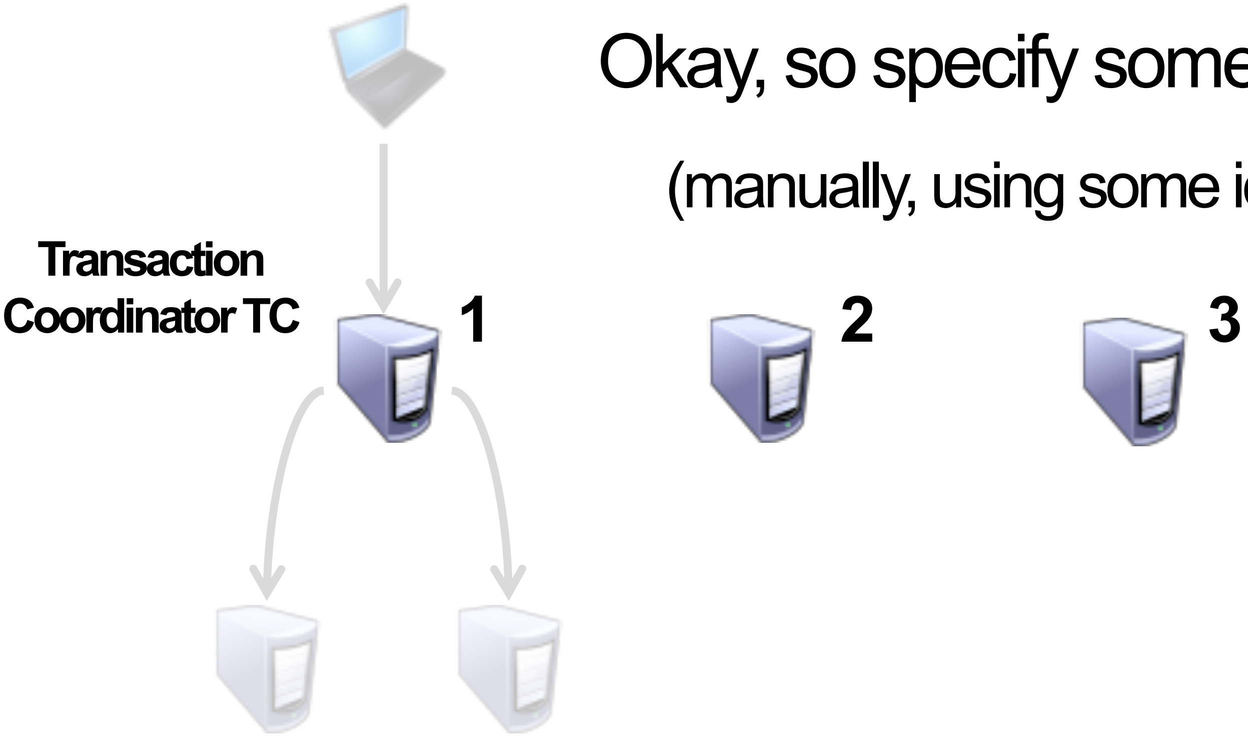


Which node takes over as backup?

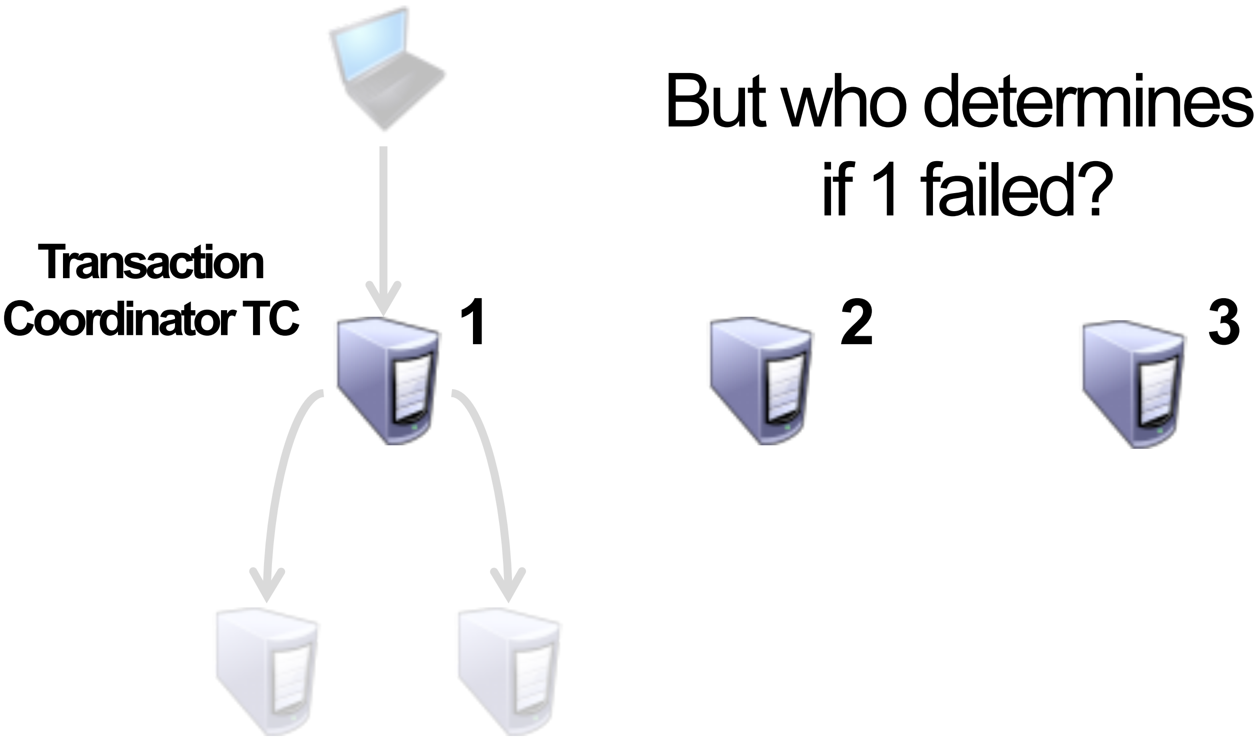
Doing failover “correctly” isn’t easy

Okay, so specify some ordering

(manually, using some identifier)



Doing failover “correctly” isn’t easy



But who determines if 1 failed?

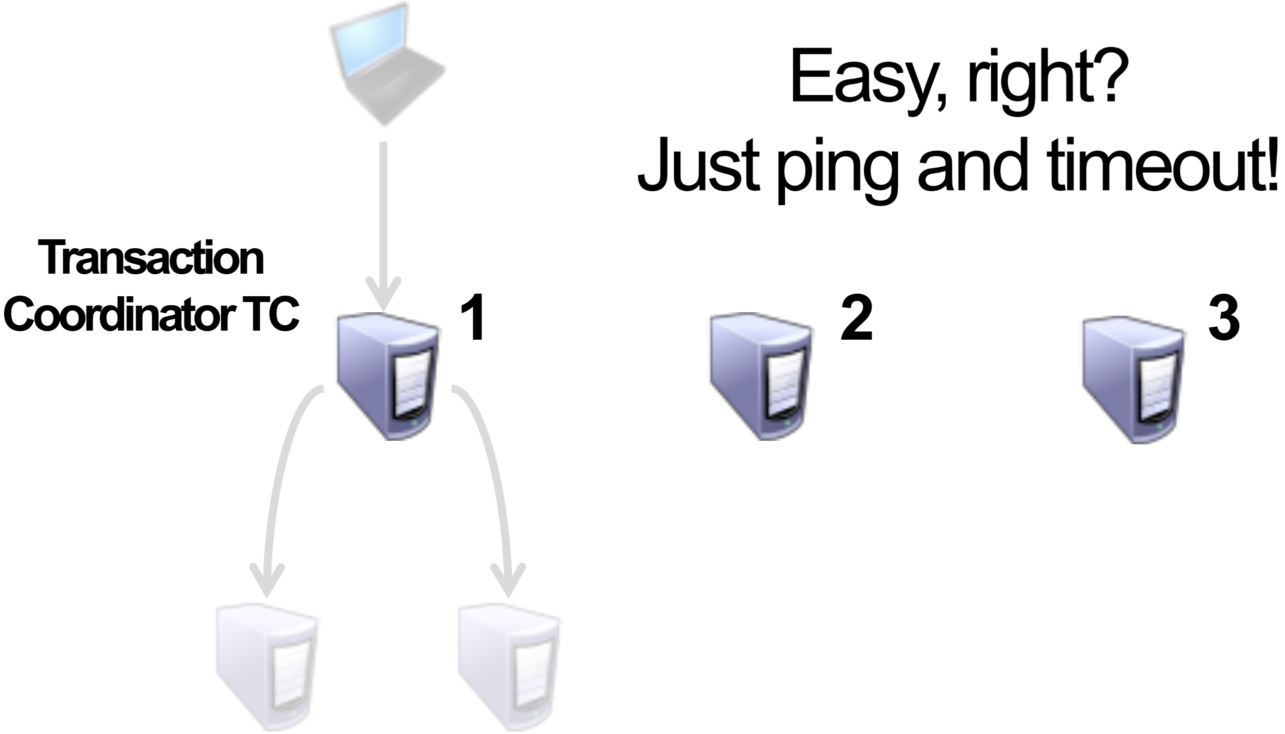
Transaction Coordinator TC

1

2

3

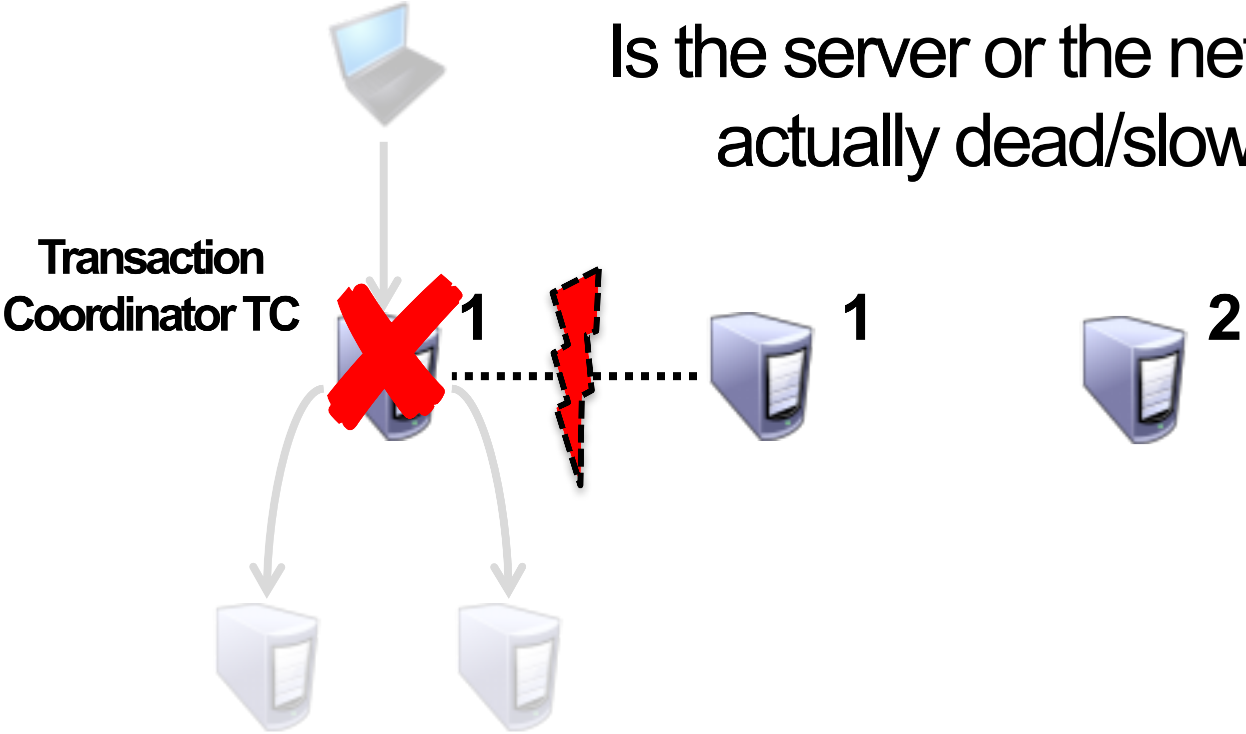
Doing failover “correctly” isn’t easy



Easy, right?
Just ping and timeout!

Doing failover "correctly" isn't easy

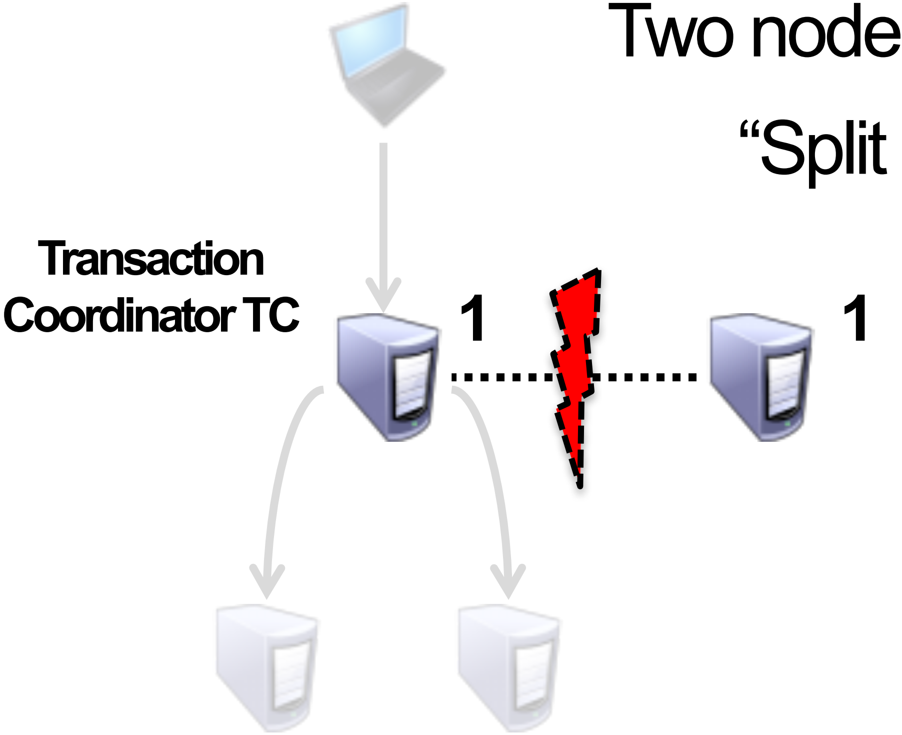
Is the server or the network actually dead/slow?



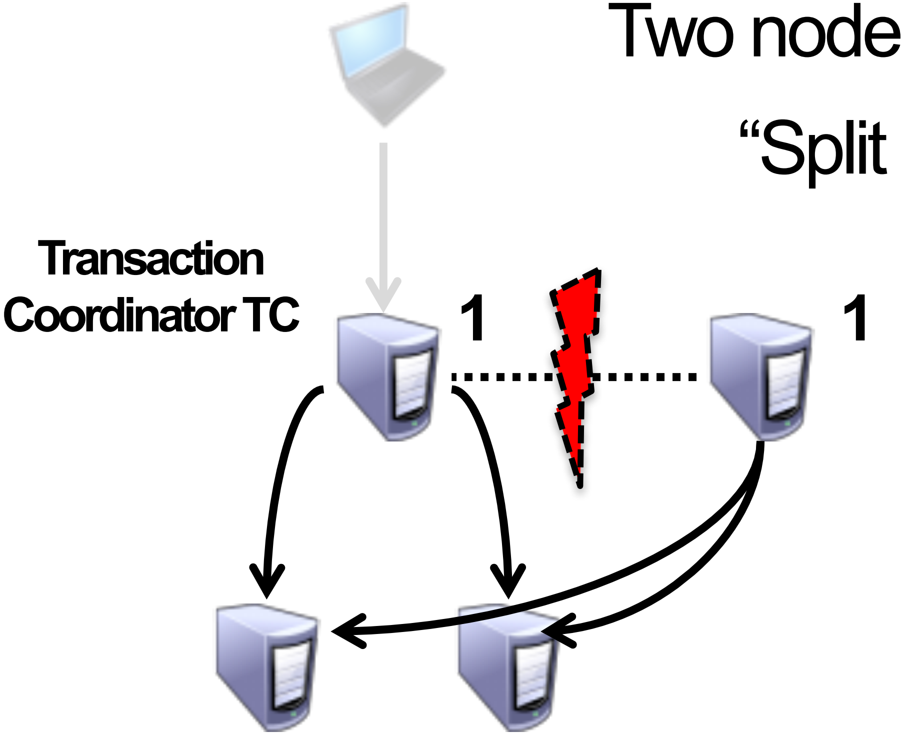
What can go wrong?

Two nodes think they are TC:

“Split brain” scenario



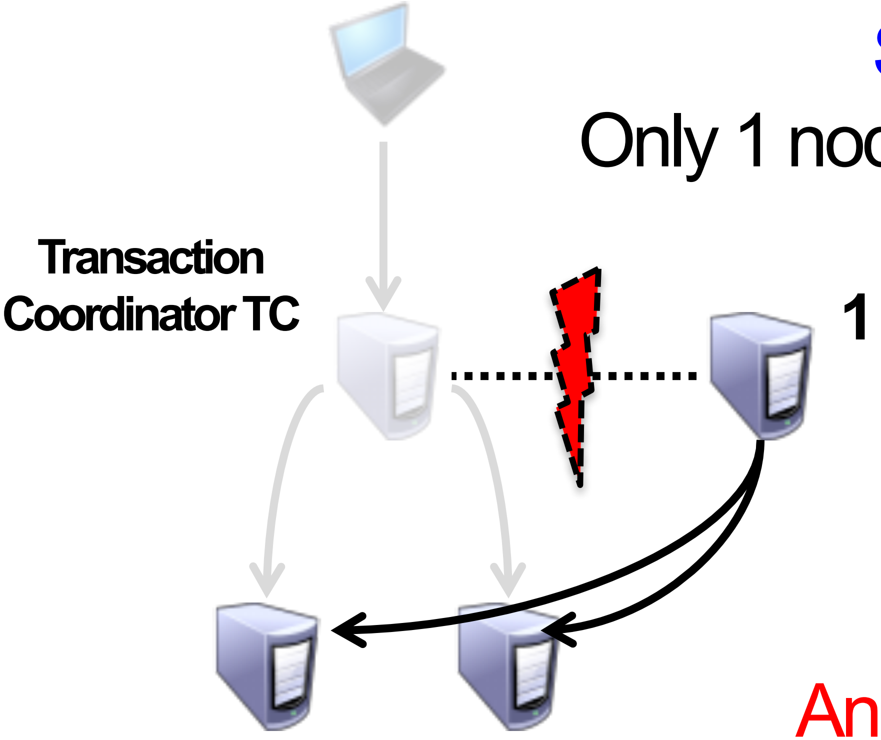
What can go wrong?



Two nodes think they are TC:

“Split brain” scenario

What can go wrong?



Safety invariant:

Only 1 node is TC at any single time

Another problem:

A and B need to know (and agree upon) who the TC is…

Consensus

Definition:

1. A general agreement about something
2. An idea or opinion that is shared by all the people in a group

Origin: Latin, from *consentire*

Consensus

Given a set of processors, 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 proposed by some process

Consensus used in systems

Group of servers attempting:

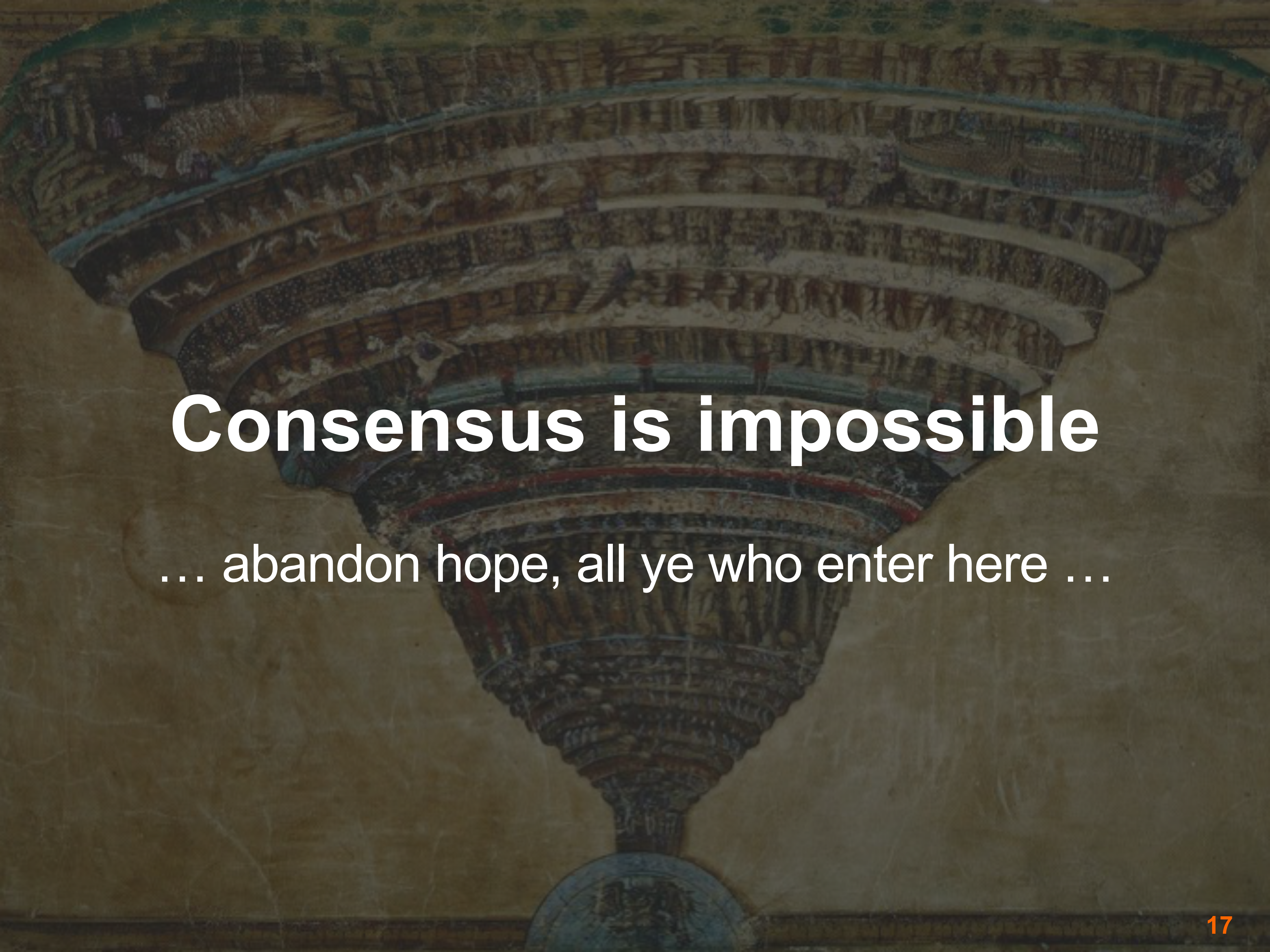
- Make sure all servers in group receive the same updates in the same order as each other
- Maintain own lists (views) on who is a current member of the group, and update lists when somebody leaves/fails
- Elect a leader in group, and inform everybody
- Ensure mutually exclusive (one process at a time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

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:
  - Fail-stop (correct/dead) or Byzantine (arbitrary)

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:
  - Fail-stop (correct/dead) or Byzantine (arbitrary)



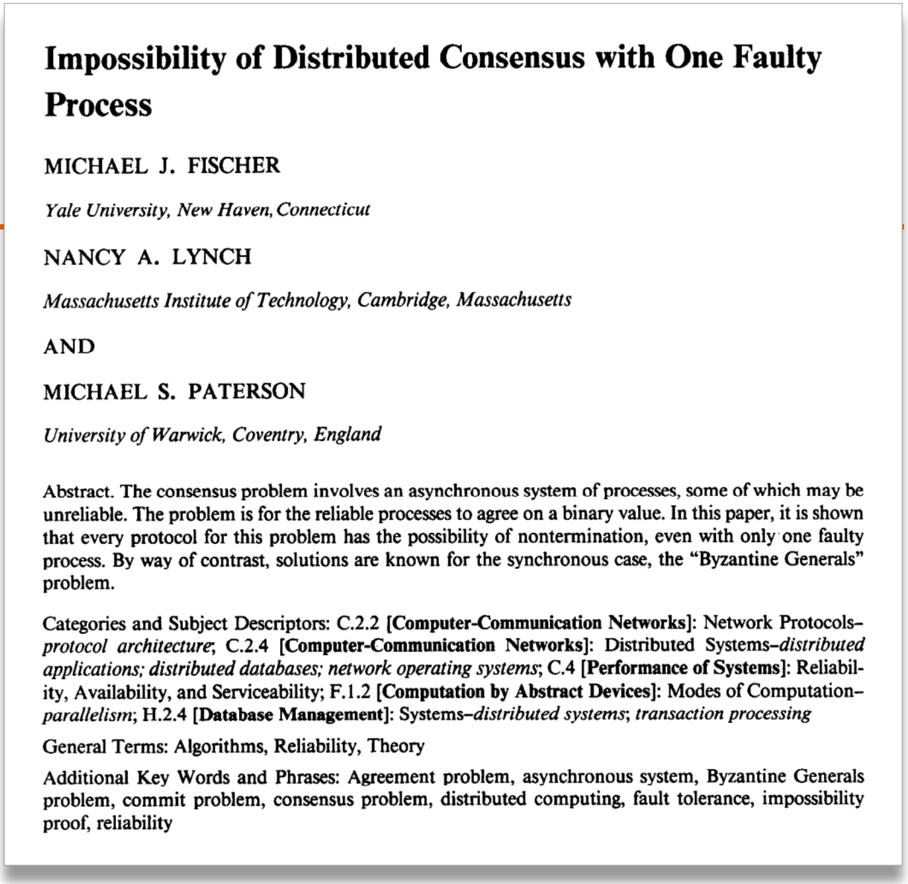
Consensus is impossible

… abandon hope, all ye who enter here …

“FLP” result

1985

- No deterministic 1-crash-robust consensus algorithm exists for asynchronous model



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

- Holds even for “weak” consensus (i.e., only *some* process needs to decide, not *all*)

- Holds even for only two states: 0 and 1

Main technical approach

- Initial state of system can end in decision “0” or “1”
- Consider 5 processes, each in some initial state

[1,1,0,1,1] → 1

[1,1,0,1,0] → ?

[1,1,0,0,0] → ?

[1,1,1,0,0] → ?

[1,0,1,0,0] → 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

[1,1,0,1,1] → 1

[1,1,0,1,0] → 1

[1,1,0 0,0] → 1

[1,1,1 0,0] → 0

[1,0,1,0,0] → 0

Assume decision differs between these two processes

Main technical approach

- Goal: Consensus holds in face of 1 failure

One of these configs must be “bi-valent”:
Both futures possible

[1,1 🟥 0,0] → 1 | 0

[1,1 🟥 0,0] → 0

Main technical approach

- Goal: Consensus holds in face of 1 failure

One of these configs must be "bi-valent":
Both futures possible

[1,1 🟥 0,0] → 1
[1,1 🟥 0,0] → 0 | 1

- Key result: All bi-valent states can remain in bi-valent states after performing some work

You won’t believe this one trick!

1. System thinks process *p* crashes, adapts to it…

2. But then *p* recovers and *q* crashes…

3. Needs to wait for *p* to rejoin, because can only handle 1 failure, which takes time for system to adapt …

4. *… repeat ad infinitum …*

All is not lost…

- But remember
  - “Impossible” in the formal sense, i.e., “there does not exist”
  - Even though such situations are extremely unlikely …
- Circumventing FLP Impossibility
  - Probabilistically
  - Randomization
  - Partial Synchrony (e.g., “failure detectors”)

Why should you care?

Werner Vogels, Amazon CTO

Job openings in my group

What kind of things am I looking for in you?

"You know your distributed systems theory: You know about logical time, snapshots, stability, message ordering, but also acid and multi-level transactions. **You have heard about the FLP impossibility argument.** You know why failure detectors can solve it (but you do not have to remember which one diamond-w was). **You have at least once tried to understand Paxos by reading the original paper."**

Paxos

- Safety
  - Only a single value is chosen
  - Only a proposed value can be chosen
  - Only chosen values are learned by processes
- Liveness ***
  - Some proposed value eventually chosen if fewer than half of processes fail
  - If value is chosen, a process eventually learns it

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”

- Accepters 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 (n, v), if $n \geq n_h$,
    - Accept proposal and notify learner(s)

      $n_a = n_h = n$

      $v_a = v$

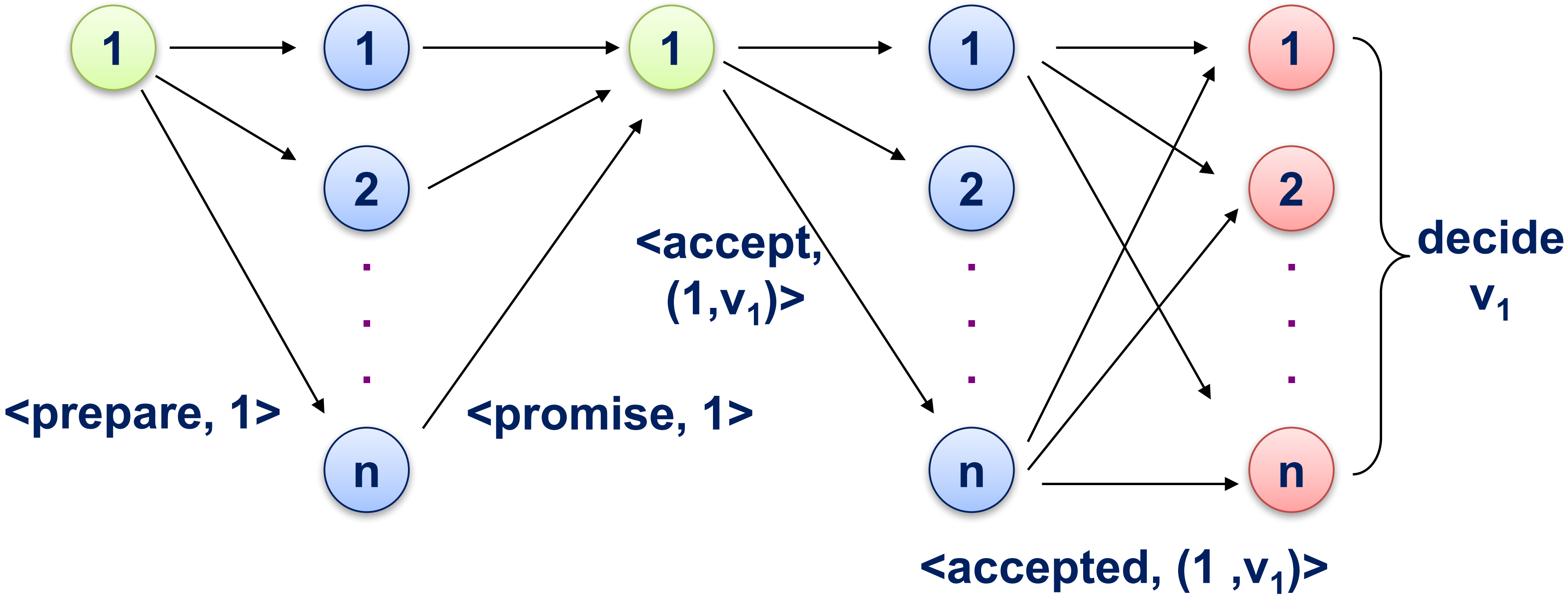
Paxos Phase 3

- Learners need to know which value chosen

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

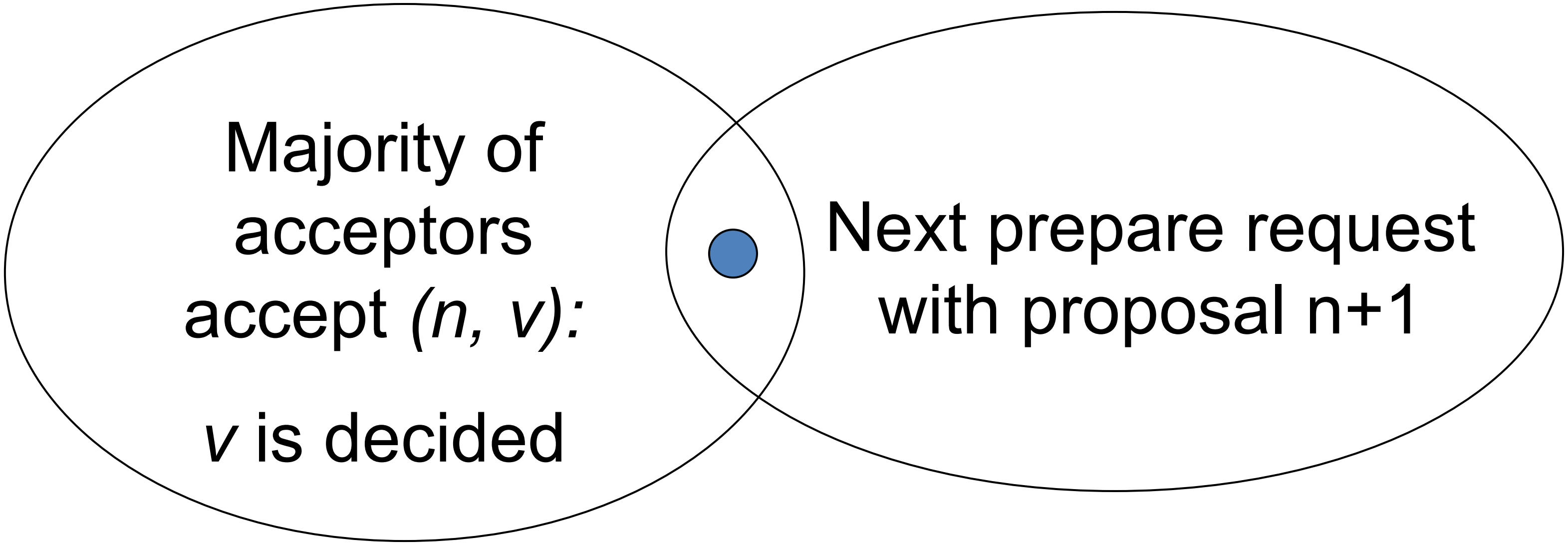
- Approach #2
  - Elect a “distinguished learner”
  - Acceptors notify elected learner, which informs others
  - Failure-prone

Paxos: Well-behaved Run

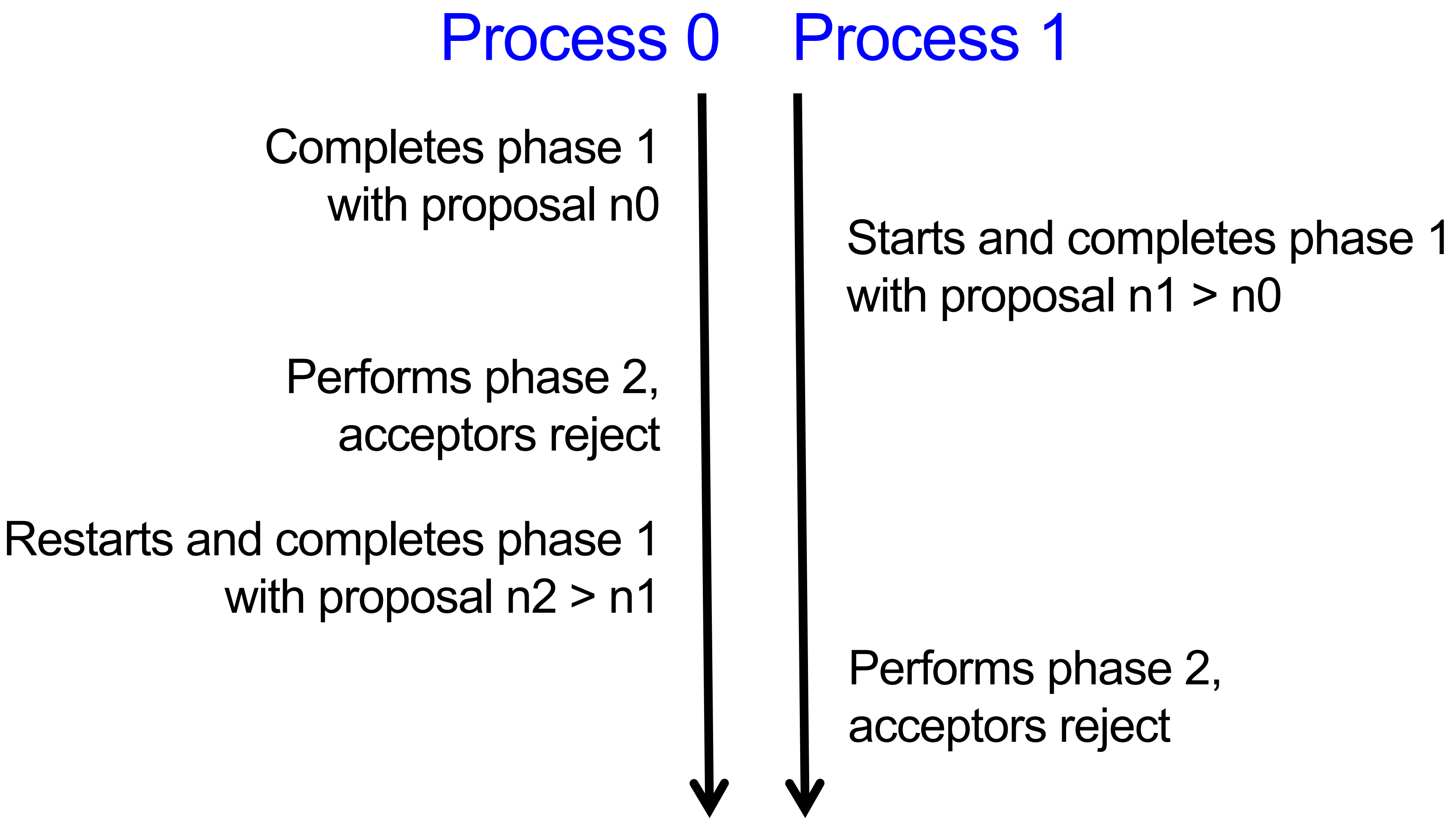


Paxos is safe

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


Majority of acceptors accept *(n, v):*
v is decided
Next prepare request with proposal n+1

Race condition leads to liveness problem



Process 0

Process 1

Completes phase 1
with proposal n0

Starts and completes phase 1
with proposal n1 > n0

Performs phase 2,
acceptors reject

Restarts and completes phase 1
with proposal n2 > n1

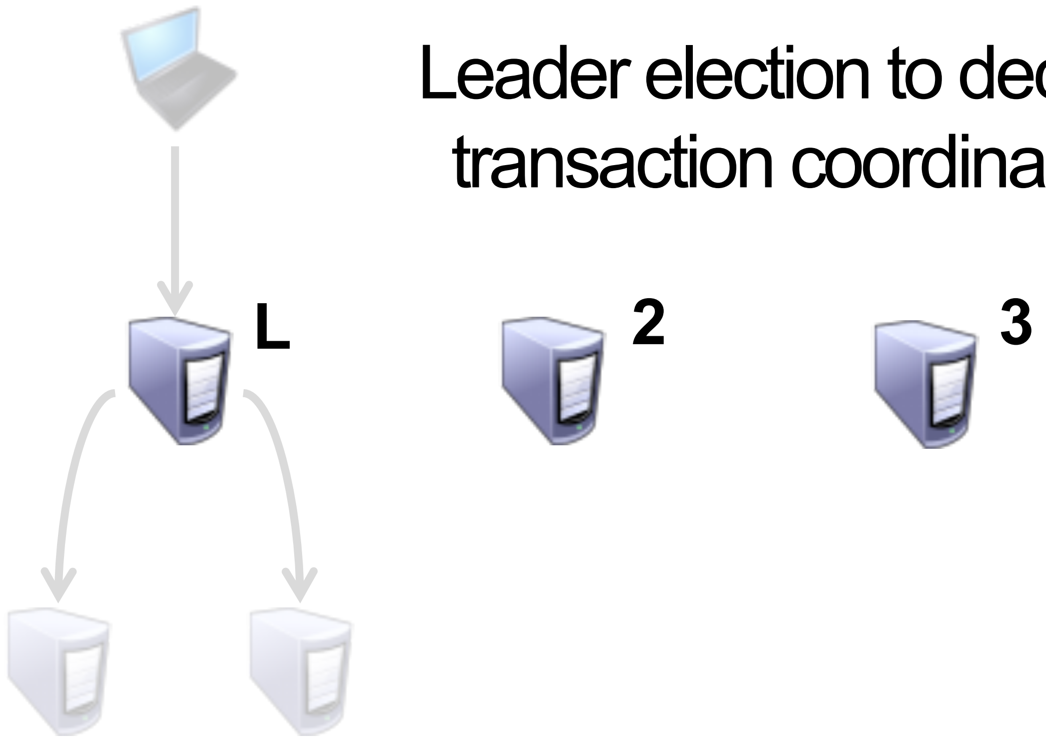
Performs phase 2,
acceptors reject

… can go on indefinitely …

Paxos with leader election

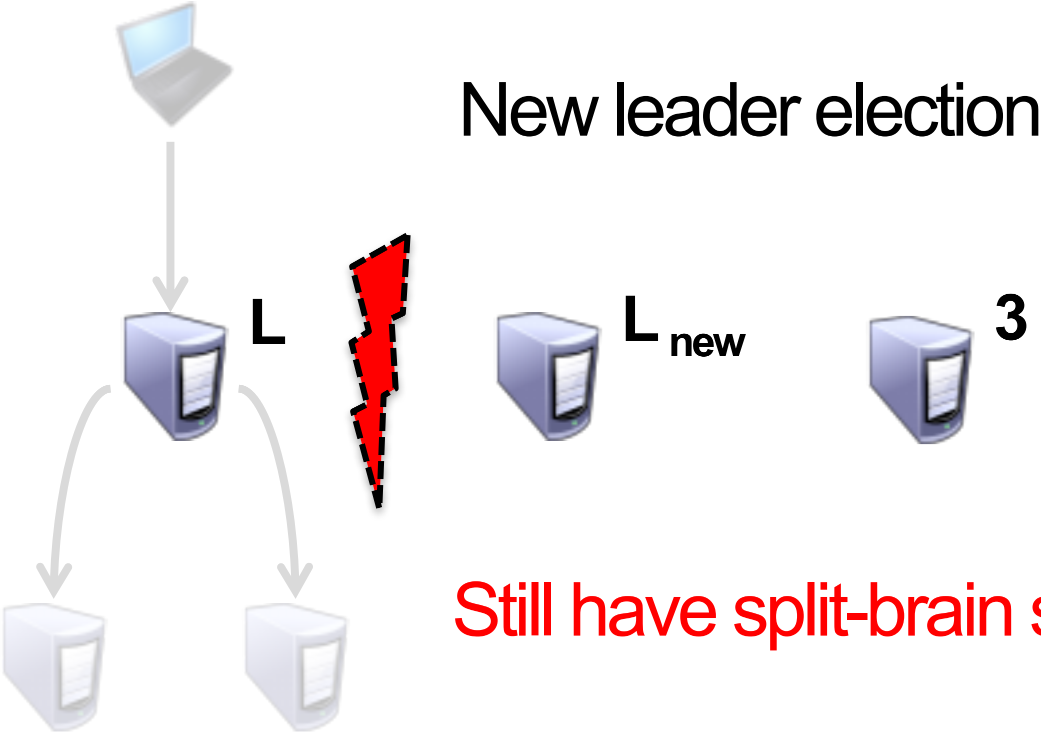
- Simplify model with each process playing all three roles
- If elected proposer can communicate with a majority, protocol guarantees liveness
- Paxos can tolerate failures f < N / 2

Using Paxos in system



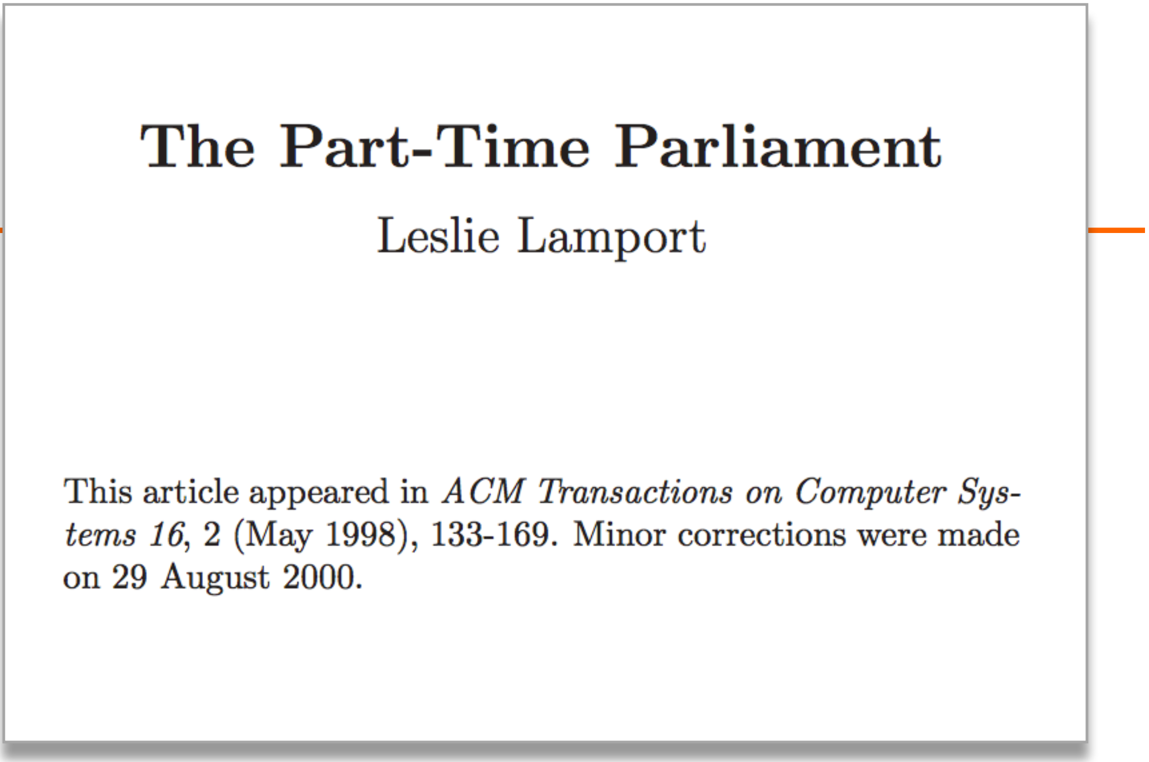
Leader election to decide transaction coordinator

Using Paxos in system



New leader election protocol

Still have split-brain scenario!



The Part-Time Parliament

Leslie Lamport

This article appeared in *ACM Transactions on Computer Systems 16*, 2 (May 1998), 133-169. Minor corrections were made on 29 August 2000.

- Tells mythical story of Greek island of Paxos with “legislators” and “current law” passed through parliamentary voting protocol
- Misunderstood paper: submitted 1990, published 1998
- Lamport won the Turing Award in 2013

The Paxos story…

As Paxos prospered, legislators became very busy.

Parliament could no longer handle all details of government, so a bureaucracy was established.

Instead of passing a decree to declare whether each lot of cheese was fit for sale, Parliament passed a decree appointing a cheese inspector to make those decisions.

Cheese inspector ≈ leader

using quorum-based voting protocol

The Paxos story…

Parliament passed a decree making Δῐκστρα the first cheese inspector. After some months, merchants complained that Δῐκστρα was too strict and was rejecting perfectly good cheese.

Parliament then replaced him by passing the decree

> 1375: Γωυδα is the new cheese inspector

But Δῐκστρα did not pay close attention to what Parliament did, so he did not learn of this decree right away.

There was a period of confusion in the cheese market when both Δῐκστρα and Γωυδα were inspecting cheese and making conflicting decisions.

Split-brain!

The Paxos story…

To prevent such confusion, the Paxons had to guarantee that a position could be held by at most one bureaucrat at any time.

To do this, a president included as part of each decree the time and date when it was proposed.

A decree making Δĭκστρα the cheese inspector might read

> 2716: 8:30 15 Jan 72 – Δĭκστρα is cheese inspector for 3 months.

Leader gets a lease!

The Paxos story…

A bureaucrat needed to tell time to determine if he currently held a post. Mechanical clocks were unknown on Paxos, but Paxons could tell time accurately to within 15 minutes by the position of the sun or the stars.

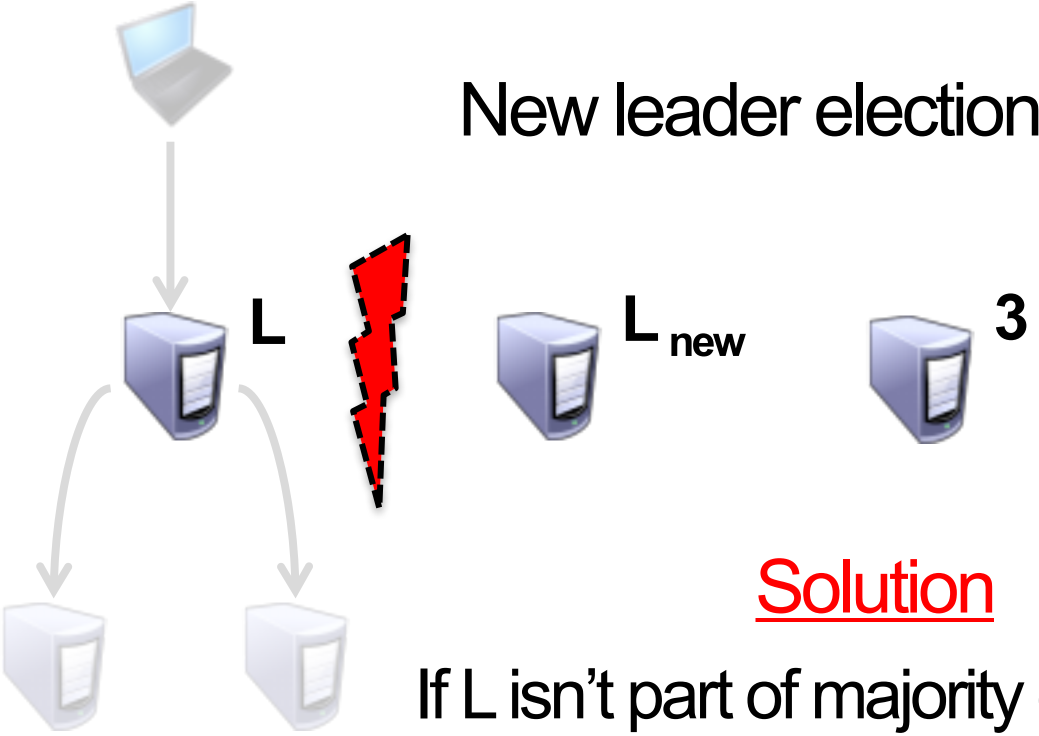
If Δῐκστρα’s term began at 8:30, he would not start inspecting cheese until his celestial observations indicated that it was 8:45.

Handle clock skew:

Lease doesn’t end until expiry + max skew

Solving Split Brain

New leader election protocol



Solution

If L isn’t part of majority electing L_{new}

L_{new} waits until L’s lease expires before accepting new ops

Next lecture: Sunday

Other consensus protocols with group membership + leader election at core

- Viewstamped Replication
- RAFT (assignment 3)