Distributed Systems

Lec 17: Agreement in Distributed Systems: Three-phase Commit, Paxos

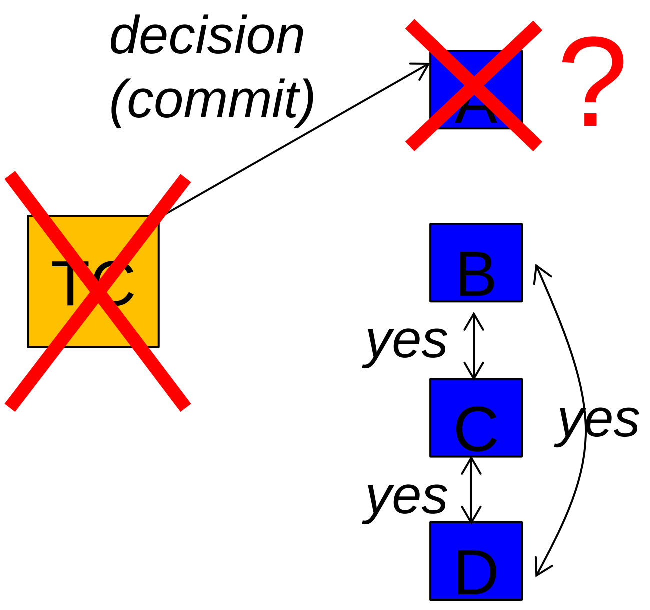
Slide acks: Jinyang Li, “The Paper Trail”

(http://news.cs.nyu.edu/~jinyang/fa10/notes/ds-paxos.ppt, http://the-paper-trail.org/blog/)

Example Blocking Failure for 2PC

- Scenario:
  - TC sends commit decision to A, A gets it and commits, and then both TC and A crash
  - B, C, D, who voted Yes, now need to wait for TC or A to reappear (w/ mutexes locked)
    - They can't commit or abort, as they don't know what A responded
  - If that takes a long time (e.g., a human must replace hardware), then availability suffers
  - If TC is also participant, as it typically is, then this protocol is vulnerable to a single-node failure (the TC's failure)!

Phase 2: Decision



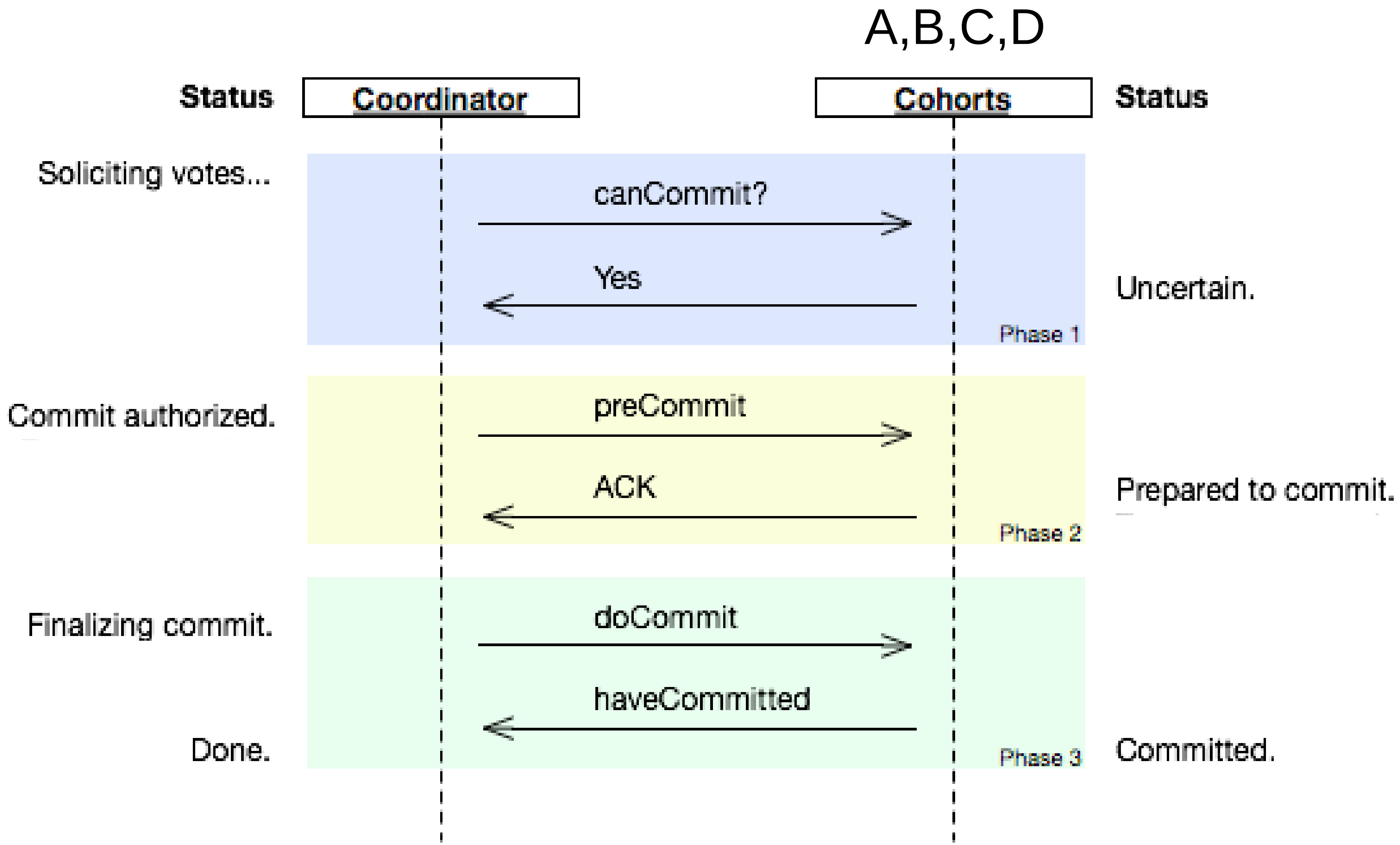
- This is why 2 phase commit is called a blocking protocol
- In context of consensus requirements: 2PC is safe, but not live

Three-Phase Commit
(the original protocol)

3PC: Goal and Idea

- Goal: Turn 2PC into a live (non-blocking) protocol
  - 3PC should never block on node failures as 2PC did
- Insight: 2PC suffers from allowing nodes to irreversibly commit an outcome before ensuring that the others know the outcome, too
- Idea in 3PC: split "commit/abort" phase into two phases
  - First communicate the outcome to everyone
  - Let them commit only after everyone knows the outcome

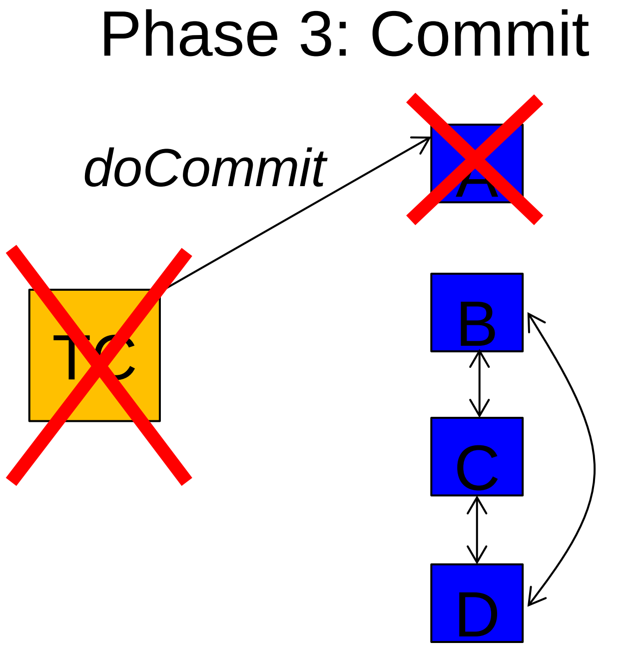
3PC



Img source: wikipedia

Can 3PC Solve Blocking 2PC Ex.?

- Assuming same scenario as before (TC, A crash), can B/C/D reach a safe decision when they time out?
  1. If one of them has received preCommit, …
  2. If none of them has received preCommit, …

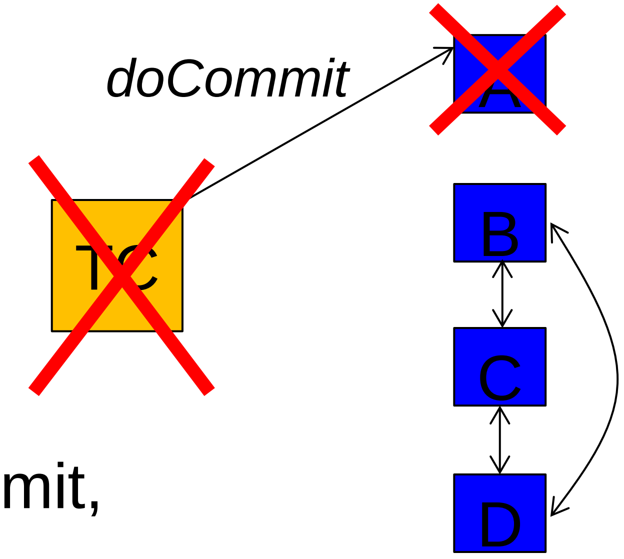


Can 3PC Solve Blocking 2PC Ex.?

- Assuming same scenario as before (TC, A crash), can B/C/D reach a safe decision when they time out?

1. If one of them has received preCommit, they can all commit
   - This is safe if we assume that A is DEAD and after coming back it runs a recovery protocol in which it requires input from B/C/D to complete an uncommitted transaction
   - This conclusion was impossible to reach for 2PC b/c A might have already committed and exposed outcome of transaction to world

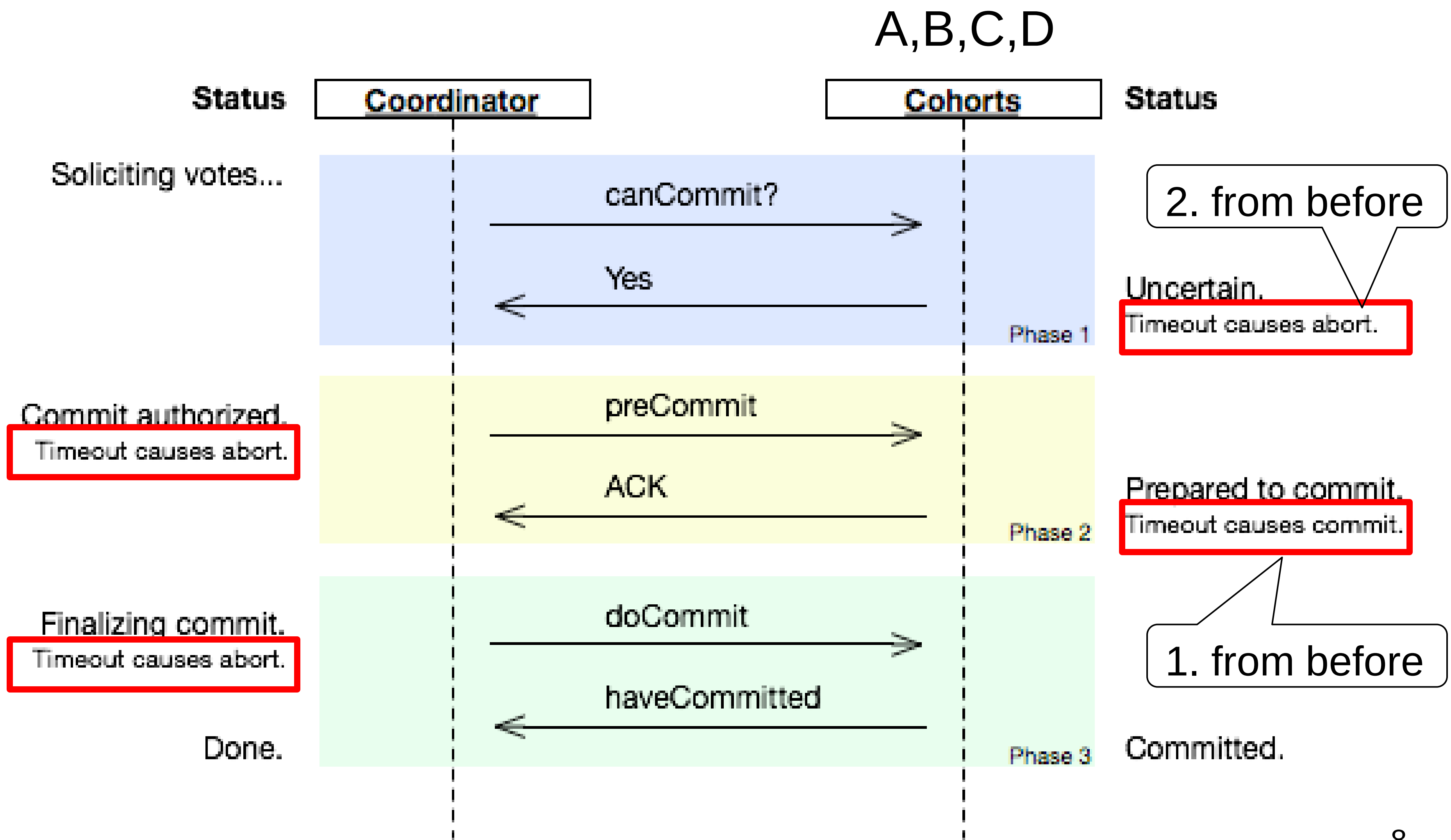
Phase 3: Commit



2. If none of them has received preCommit, they can all abort
   - This is safe, b/c we know A couldn't have received a doCommit, so it couldn't have committed

3PC is safe for node crashes (including TC+participant)

3PC: Timeout Handling Specs (trouble begins)



Img source: wikipedia

But Does 3PC Achieve Consensus?

- Liveness (availability): Yep
  - Doesn’t block, it always makes progress by timing out

- Safety (correctness): Nope
  - Can you think of scenarios in which original 3PC would result in inconsistent states between the replicas?

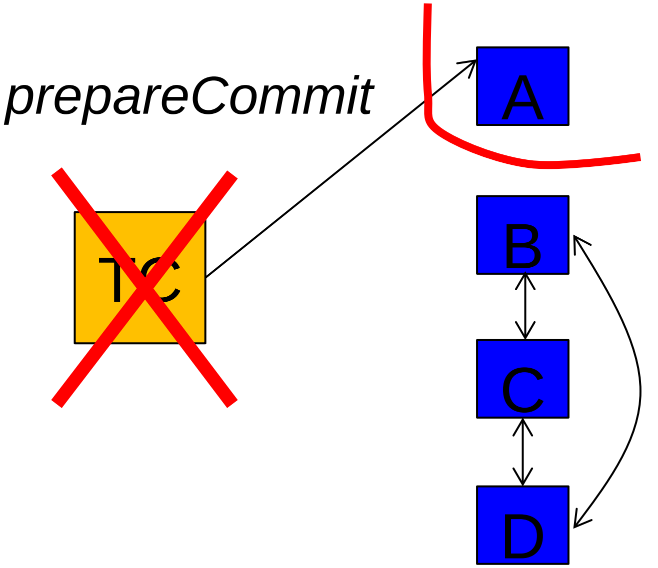
- Two examples of unsafety in 3PC:
  - A hasn’t crashed, it’s just offline
  - TC hasn’t crashed, it’s just offline

Network partitions

3PC with Network Partitions

- One example scenario:
  - A receives prepareCommit from TC
  - Then, A gets partitioned from B/C/D and TC crashes
  - None of B/C/D have received prepareCommit, hence they all abort upon timeout
  - A is prepared to commit, hence, according to protocol, after it times out, it unilaterally decides to commit

Phase 2: Prepare



- Similar scenario with partitioned, not crashed, TC

Safety vs. Liveness

- So, 3PC is doomed for network partitions
  - The way to think about it is that this protocol's design trades safety for liveness
- Remember that 2PC traded liveness for safety

- Can we design a protocol that's both safe and live?

- Well, it turns out that it's impossible in the most general case!

Fischer-Lynch-Paterson [FLP'85] Impossibility Result

- It is impossible for a set of processors in an asynchronous system to agree on a binary value, even if only a single process is subject to an unannounced failure
  - We won't show any proof here – it's too complicated
- The core of the problem is asynchrony
  - It makes it impossible to tell whether or not a machine has crashed (and therefore it will launch recovery and coordinate with you safely) or you just can't reach it now (and therefore it's running separately from you, potentially doing stuff in disagreement with you)
- For synchronous systems, 3PC can be made to guarantee both safety and liveness!
  - When you know the upper bound of message delays, you can infer when something has crashed with certainty

FLP – Translation

- What FLP says: you can't guarantee both safety and progress when there is even a single fault at an inopportune moment
- What FLP doesn't say: in practice, how close can you get to the ideal (always safe and live)?
- Next: Paxos algorithm, which in practice gets close

Paxos

Paxos

- The only known completely-safe and largely-live agreement protocol

- Lets all nodes agree on the same value despite node failures, network failures, and delays
  – Only blocks in exceptional circumstances that are vanishingly rare in practice

- Extremely useful, e.g.:
  – nodes agree that client X gets a lock
  – nodes agree that Y is the primary
  – nodes agree that Z should be the next operation to be executed

Paxos Is Everywhere

- Widely used in both industry and academia
- Examples:
  - Google: Chubby (Paxos-based distributed lock service)
    - Most Google services use Chubby directly or indirectly
  - Yahoo: Zookeeper (Paxos-based distributed lock service)
  - MSR: Frangipani (Paxos-based distributed lock service)
    - The YFS labs contain a Paxos assignment, hopefully we'll get to it
  - UW: Scatter (Paxos-based consistent DHT)
  - Open source:
    - libpaxos (Paxos-based atomic broadcast)
    - Zookeeper is open-source and integrates with Hadoop

Paxos Properties

- Safety
  - If agreement is reached, everyone agrees on the same value
  - The value agreed upon was proposed by some node
- Fault tolerance (i.e., as-good-as-it-gets liveness)
  - If less than half the nodes fail, the rest nodes reach agreement *eventually*
- No guaranteed termination (i.e., imperfect liveness)
  - Paxos may not always converge on a value, but only in very degenerate cases that are improbable in the real world
- Ah, and lots of awesomeness ☺
  - Basic idea seems natural in retrospect, but why it works in any detail is incredibly complex!

Outline of Paxos Presentation

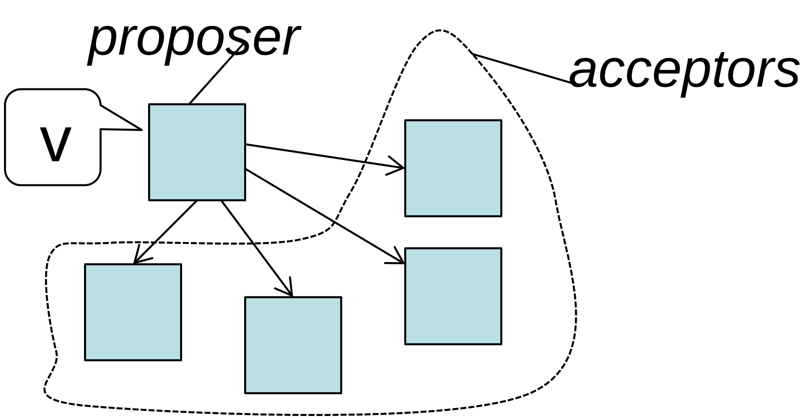
- High-level overview
- Detailed operation

Outline of Paxos Presentation

- High-level overview
- Detailed operation

The Basic Idea

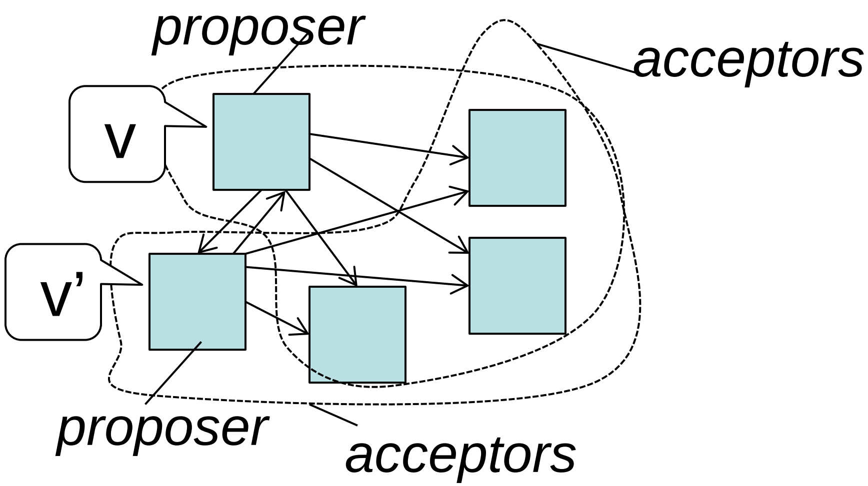
- Paxos is similar to 2PC, but with some twists
- One (or more) node decides to be coordinator (*proposer*)
- Proposer proposes a value and solicits acceptance from others (*acceptors*)
- Proposer announces the chosen value or tries again if it's failed to converge on a value



- Values to agree on:
  - Whether to commit/abort a transaction
  - Which client should get the next lock
  - Which write we perform next
  - What time to meet (rmb. party example)

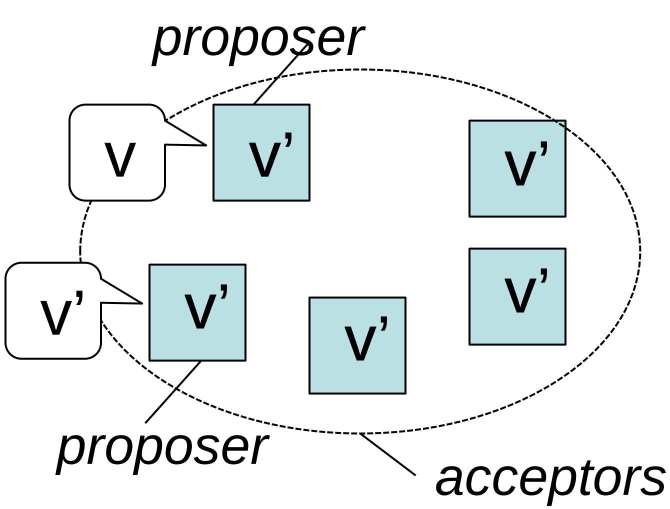
The Basic Idea

- Paxos is similar to 2PC, but with some twists
- One (or more) node decides to be coordinator (*proposer*)
- Proposer proposes a value and solicits acceptance from others (*acceptors*)
- Proposer announces the chosen value or tries again if it's failed to converge on a value



The Basic Idea

- Paxos is similar to 2PC, but with some twists
- One (or more) node decides to be coordinator (*proposer*)
- Proposer proposes a value and solicits acceptance from others (*acceptors*)
- Proposer announces the chosen value or tries again if it's failed to converge on a value



- Hence, Paxos is egalitarian: any node can propose/accept, no one has special powers
- Just like real world, e.g., group of friends organize a party – anyone can take the lead (including >1 @once)

Challenges Addressed in Paxos

- What if multiple nodes become proposers simultaneously?
- What if the new proposer proposes different values than an already decided value?
- What if there is a network partition?
- What if a proposer crashes in the middle of solicitation?
- What if a proposer crashes after deciding but before announcing results?
- …

- Similar concerns occur in the party example – go over scenario

Core Differentiating Mechanisms

1. Proposal ordering
   – Lets nodes decide which of several concurrent proposals to accept and which to reject

2. Majority voting
   – 2PC needs all nodes to vote Yes before committing
     • As a result, 2PC may block when a single node fails
   – Paxos requires only a majority of the acceptors (half+1) to accept a proposal
     • As a result, in Paxos nearly half the nodes can fail to reply and the protocol continues to work correctly
     • Moreover, since no two majorities can exist simultaneously, network partitions do not cause problems (as they did for 3PC)

Strawman 1

- Strawman:
  - Each proposer proposes to all acceptors
  - Each acceptor accepts the first proposal it receives and rejects rest
  - If the proposer receives positive replies from a majority of acceptors, it chooses its own value
    - There is at most 1 majority, hence at most a single value is chosen, even if there are partitions
  - Proposer sends chosen value to everyone

- Problems?
  - What if multiple proposers propose simultaneously, so there is no majority accepting?
  - What if the proposer dies?
  - (Give examples from group party.)

Strawman 2

- Strawman:
  - Enforce a global ordering of all proposals
  - Let acceptors recant their older proposals and accept newer ones
  - This will allow consistent progress for both simultaneous proposers and dead proposers
- Problem?
  - What if old proposer isn't dead, but rather just slow?
  - It may think that its proposed value has won, whereas a newer proposer's value has won -- getting back on your word creates problems
  - (Give example with printed invitations for group party.)

Paxos’ Solution (P.1): Proposal Ordering

- Each acceptor must be able to accept multiple proposals
- Order proposals globally by a proposal number, which acceptors use to select which proposals to accept/reject
  - “node-address:node-local-sequence-number,” e.g.: N1:7

- Two rules for accepting proposals:
  1. When acceptor receives a new proposal with # n, it looks at the highest-number proposal it has already accepted, # m, and accepts the new proposal only if n>m and rejects it otherwise
  2. If the acceptor decides to accept new proposal # n, then it will ask the new proposer to use the same value as # m’s (the old one he had already agreed to)

Paxos' Solution (P.2): Majorities

- Paxos requires half+1 of the nodes to agree on a value before it can be chosen

- Properties of majorities:
  - No two separate majorities can exist simultaneously, not even in the case of network partitions
  - If two majorities successively agree on two distinct proposals, #n and #m, respectively, then there is at least one node who's been in both majorities and has seen both proposals
  - If another majority agrees then on a third proposal, #p, then there are a set of nodes that collectively will have seen all three proposals, #m, #n, #p!

- Thus, if a proposal with value v is chosen, all higher proposals will preserve value v (so nodes need not recant)

Outline of Paxos Presentation

- High-level overview
- Detailed operation

Detailed Operation: Node State

- Each node maintains:
  - n_a, v_a: highest proposal # accepted and its corresponding accepted value
  - n_h: highest proposal # seen
  - my_n: my proposal # in the current Paxos round

Detailed Operation

- Phase 1 (Propose)
  - A node decides to be proposer (a.k.a. leader)
  - Leader chooses $my_n > n_h$
  - Leader sends <propose, my_n> to all nodes
  - Upon receiving <propose, n>

    If $n < n_h$

        reply <propose-reject>

    Else

        $n_h = n$

        reply <propose-ok, n_a,v_a>

This node promises to not accept any future proposals lower than n

Detailed Operation

- Phase 2 (Accept):
  - If leader gets propose-ok from a majority

    V = value corresponding to the highest n_a received

    If V= null, then leader can pick any V

    Send <accept, my$n$, V> to all nodes (includes himself)
  - If leader fails to get prepare-ok from a majority
    - Delay and restart Paxos
  - Upon receiving <accept, n, V>

    If $n < n_h$

    reply with <accept-reject>

    else

    $n_a = n$; $v_a = V$; $n_h = n$

    reply with <accept-ok>

Detailed Operation

- Phase 2 (Accept):
  - If leader gets propose-ok from a majority

    V = value corresponding to the highest n_a received

    If V= null, then leader can pick any V

    Send <accept, my$_n$, V> to all nodes (includes himself)
  - If leader fails to get prepare-ok from a majority
    - Delay and restart Paxos
  - Upon receiving <accept, n, V>

    If $n < n_h$

        reply with <accept-reject>

    else

        $n_a = n$; $v_a = V$; $n_h = n$

        reply with <accept-ok>

Detailed Operation

- Phase 2 (Accept)
  - If leader gets propose-ok from a majority

    V = value corresponding to the highest n_a received

    If V= null, then leader can pick any V

    Send <accept, my$_n$, V> to all nodes (includes himself)
  - If leader fails to get prepare-ok from a majority
    - Delay and restart Paxos
  - Upon receiving <accept, n, V>

    If $n < n_h$

    reply with <accept-reject>

    else

    $n_a = n$; $v_a = V$; $n_h = n$

    reply with <accept-ok>

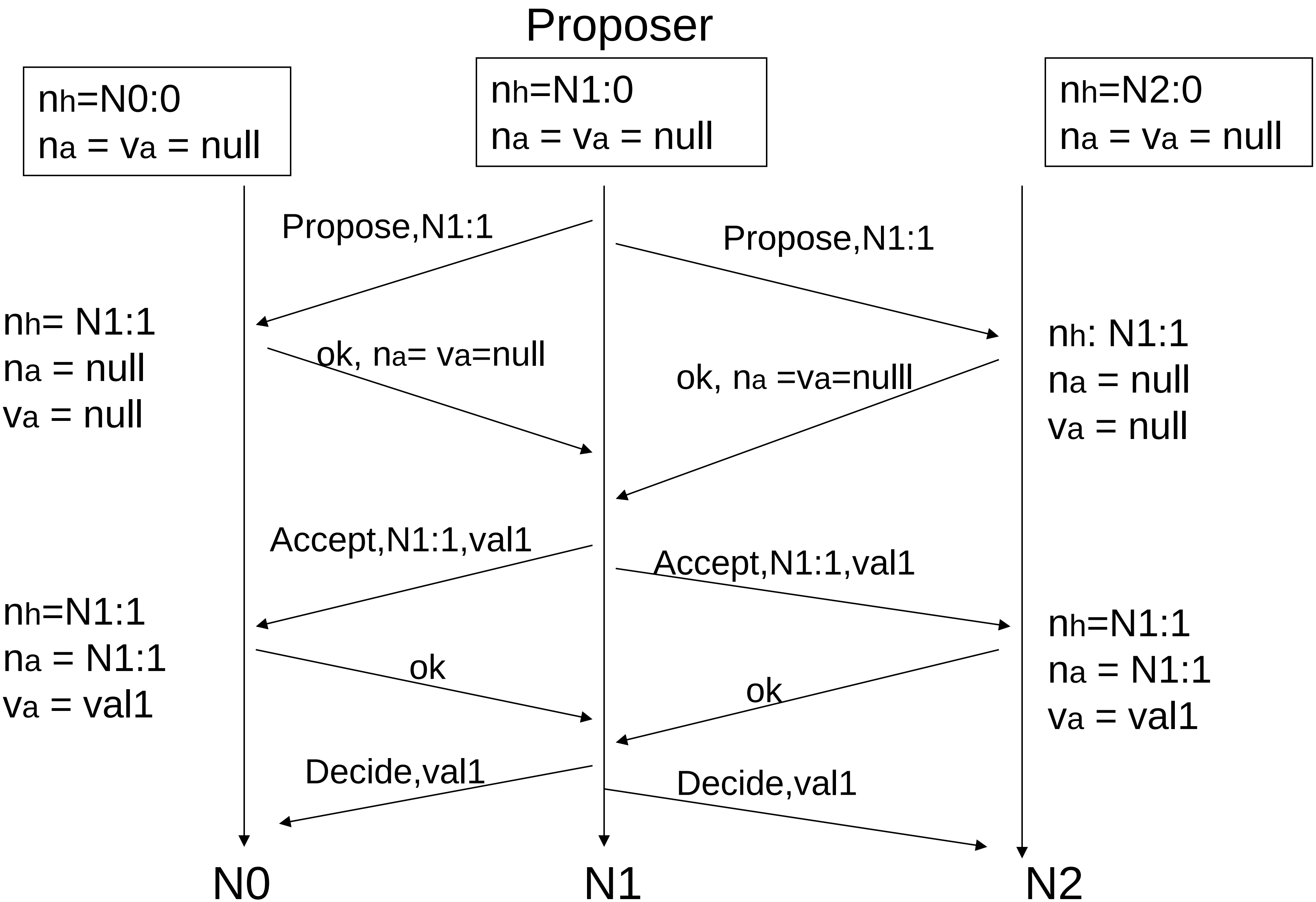
If a value has been chosen at some point in the past, but its proposer didn't quite finish his job, then that value will remain in perpetuity

So: newer proposers win the rounds, but with old proposers' values!!!

Detailed Operation

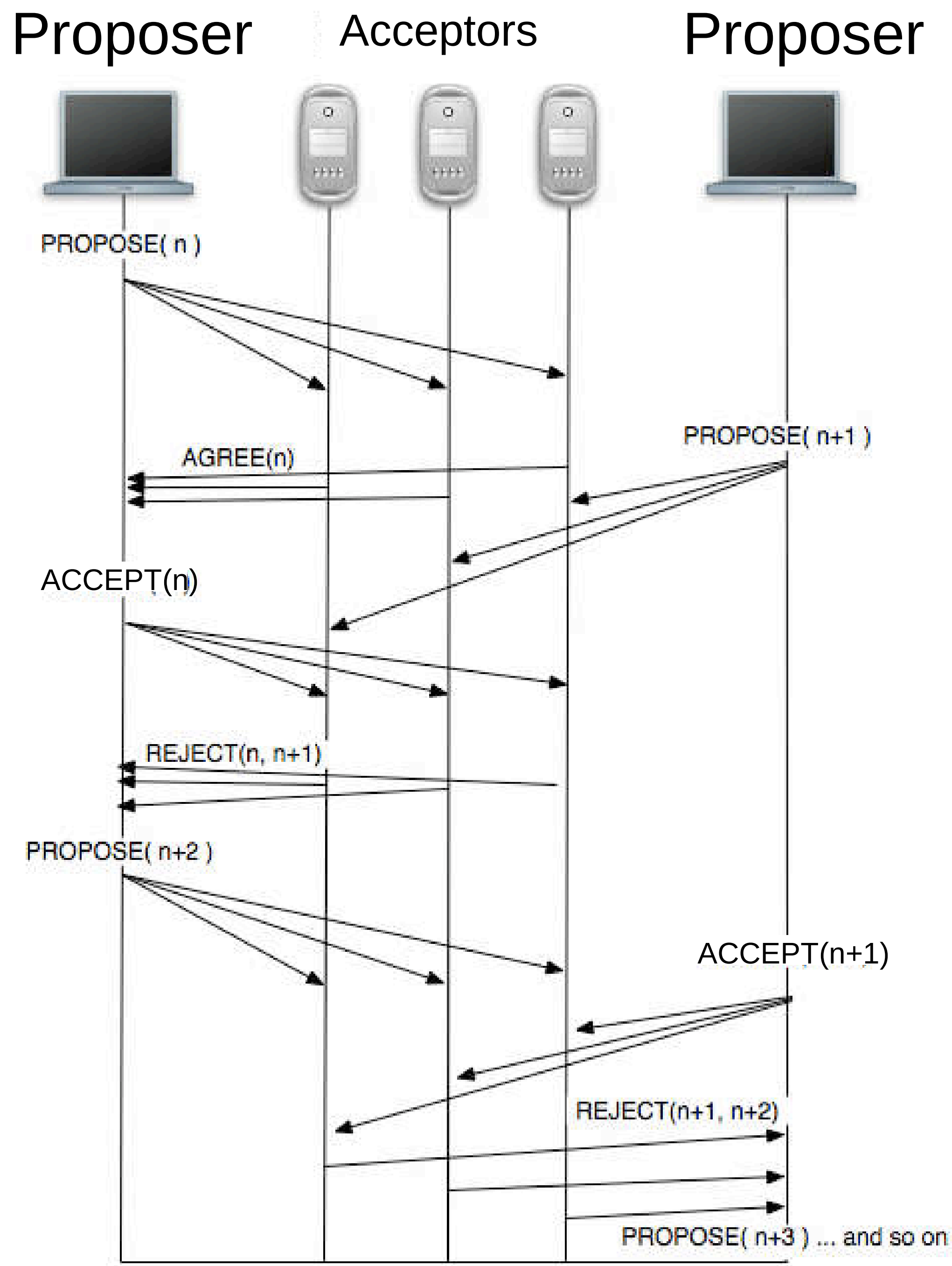
- Phase 3 (Decide)
  - If leader gets accept-ok from a majority
    - Send <decide, va> to all nodes
  - If leader fails to get accept-ok from a majority
    - Delay and restart Paxos
  - This phase is so all folks can learn the value chosen previously and the protocol can close

Detailed Operation: An Example



Paxos May Not Terminate

- Dueling proposers:
  - If two or more proposers race to propose new values, they might step on each other toes all the time
- With randomness, this occurs exceedingly rarely



Understanding Paxos
(For you to think about)

- When is the value V chosen?
  1. When leader receives a majority prepare-ok and proposes V
  2. When a majority nodes accept V
  3. When the leader receives a majority accept-ok for value V

Understanding Paxos
(For you to think about)

- What if more than one leader is active?
- Suppose two leaders use different proposal number, N0:10, N1:11
- Can both leaders see a majority of prepare-ok?

Understanding Paxos
(For you to think about)

- What if leader fails while sending accept?
- What if a node fails after receiving accept?
  - If it doesn’t restart …
  - If it reboots …
- What if a node fails after sending prepare-ok?
  - If it reboots …

Paxos vs. 2/3PC

- Paxos is similar to 2PC/3PC (but not really)
- Remember:
  - 2PC was vulnerable to 1-node failures, especially coordinator failures
  - 3PC was vulnerable to network partitions

- Paxos deals with these issues using two mechanisms:
  - Egalitarian consensus: no node is special, anyone can take over as coordinator at any time
    - Hence, if one coordinator fails, another one will time out and take over
    - But that requires special ordering and acceptance protocols for proposals
  - Safe majorities: instead of requiring all participants to answer Yes, Paxos requires only half + 1 of the nodes
    - Because you cannot have two simultaneous majorities, which avoids partitions

Readings Useful for Final Exam

- Two-phase commit:
  - http://the-paper-trail.org/blog/consensus-protocols-two-phase-commit/
- Three-phase commit:
  - http://the-paper-trail.org/blog/consensus-protocols-three-phase-commit
- Paxos:
  - http://the-paper-trail.org/blog/consensus-protocols-paxos/
- FLP impossibility result in distributed systems:
  - http://betathoughts.blogspot.com/2007/06/brief-history-of-consensus-2pc-and.html

Next Time

- A system that uses Paxos
  - Bigtable on top of Chubby