

# Distributed Systems

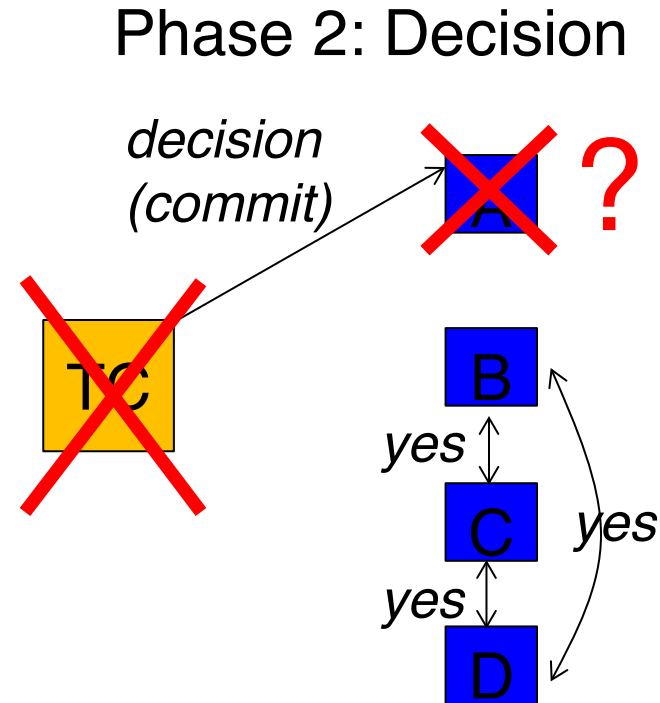
## Lec 17: Agreement in Distributed Systems: Three-phase Commit, Paxos

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)!



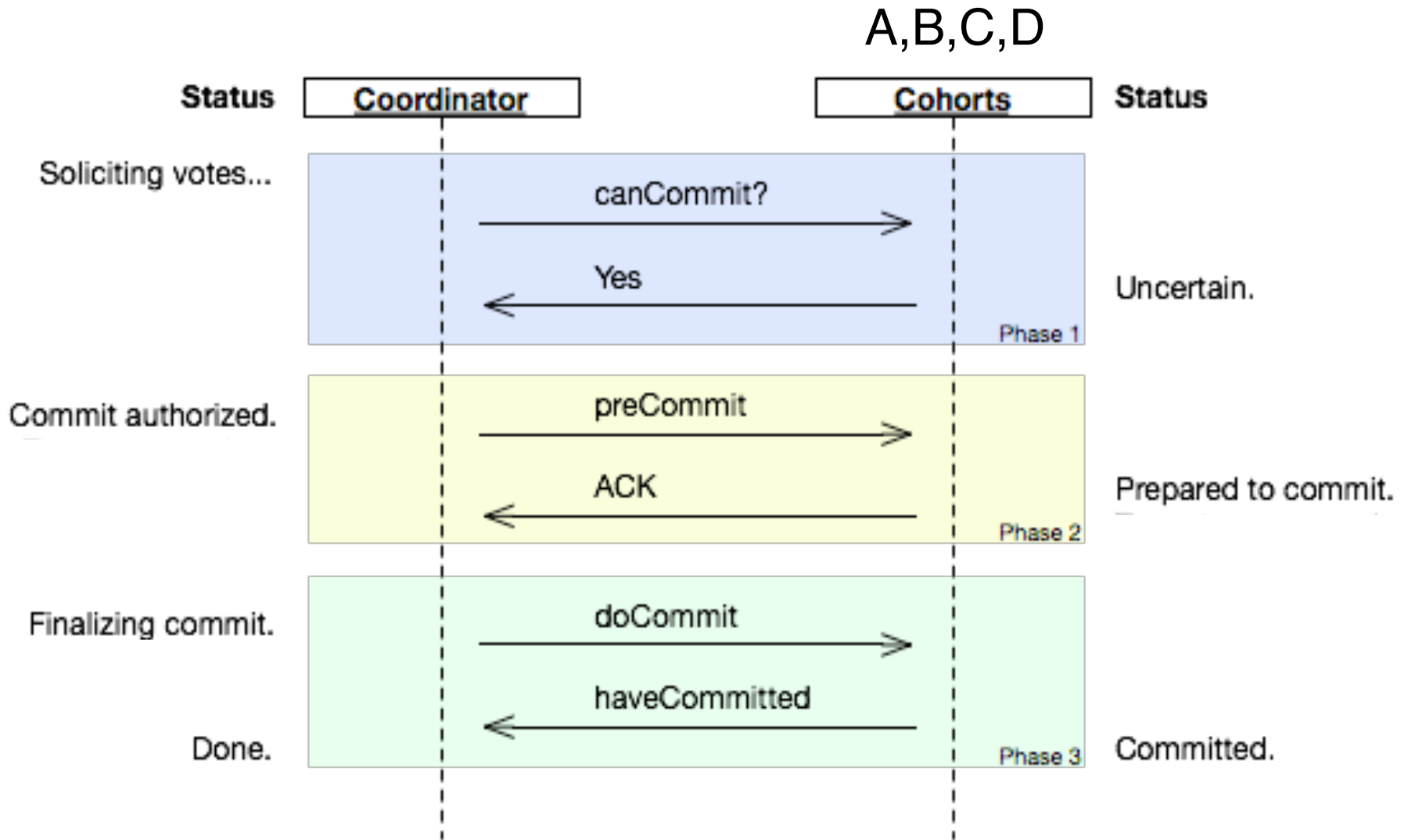
- This is why 2 phase commit is called a **blocking protocol**
- In context of consensus requirements: **2PC is safe, but not live**

# Three-Phase Commit (the original protocol)

# 3PC: Goal and Idea

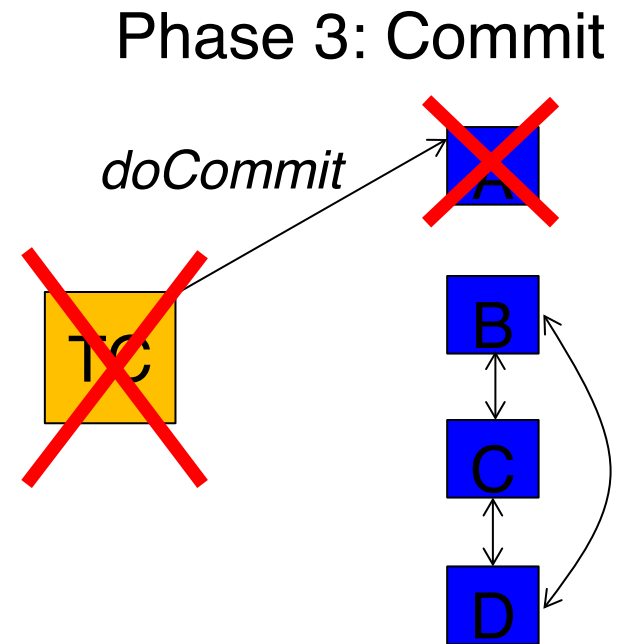
- **Goal:** Turn 2PC into a **live (non-blocking) protocol**
  - 3PC should never block on node failures as 2PC did
- **Insight:** 2PC suffers from allowing nodes to irreversibly commit an outcome before ensuring that the others know the outcome, too
- **Idea in 3PC:** split “commit/abort” phase into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

# 3PC



# 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?
  - If one of them has received preCommit, ...
  - 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?

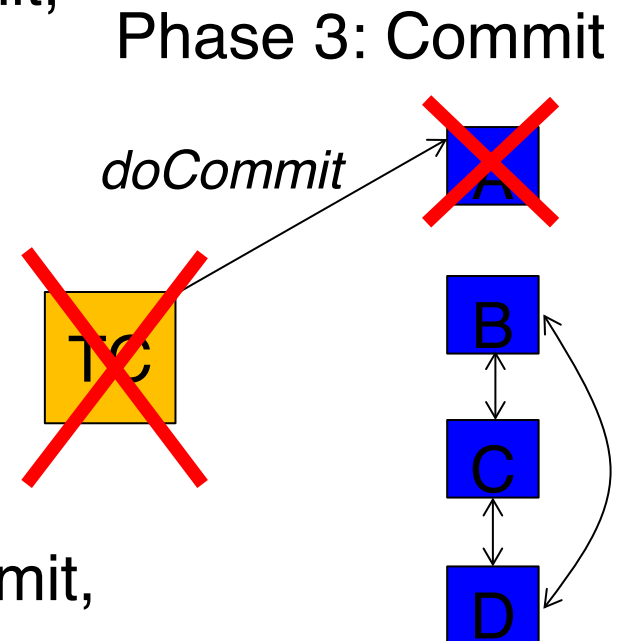
- If one of them has received preCommit, they can all **commit**

- 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

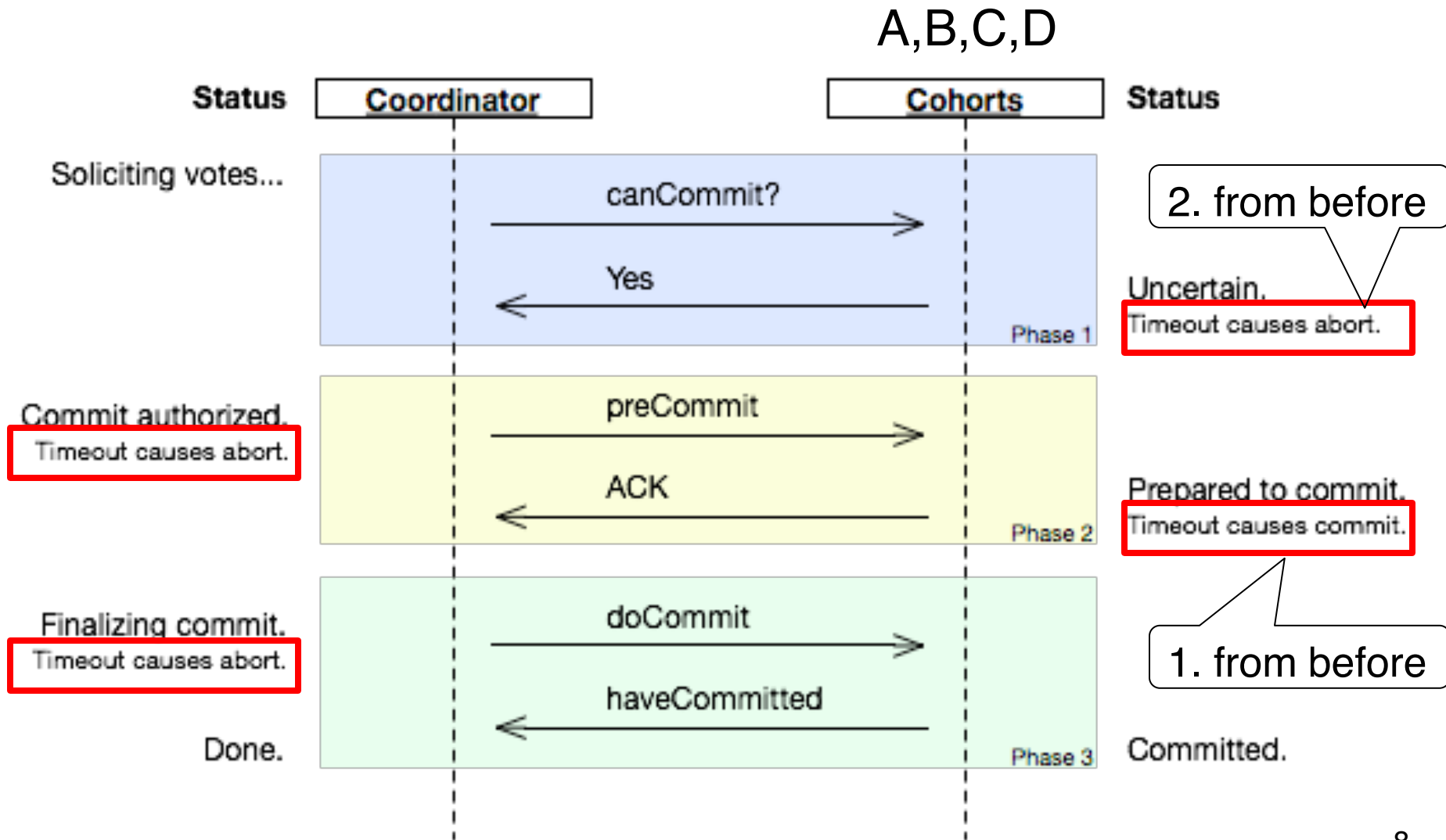
- If none of them has received preCommit, they can all **abort**

- 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)





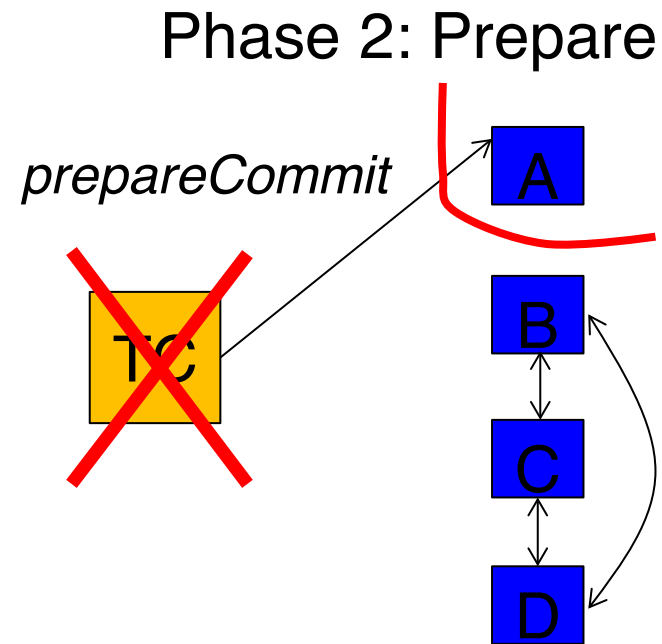
# But Does 3PC Achieve Consensus?

- **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



- 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