Multi-phase Commit Protocols

Based on slides by Ken Birman,
Cornell University

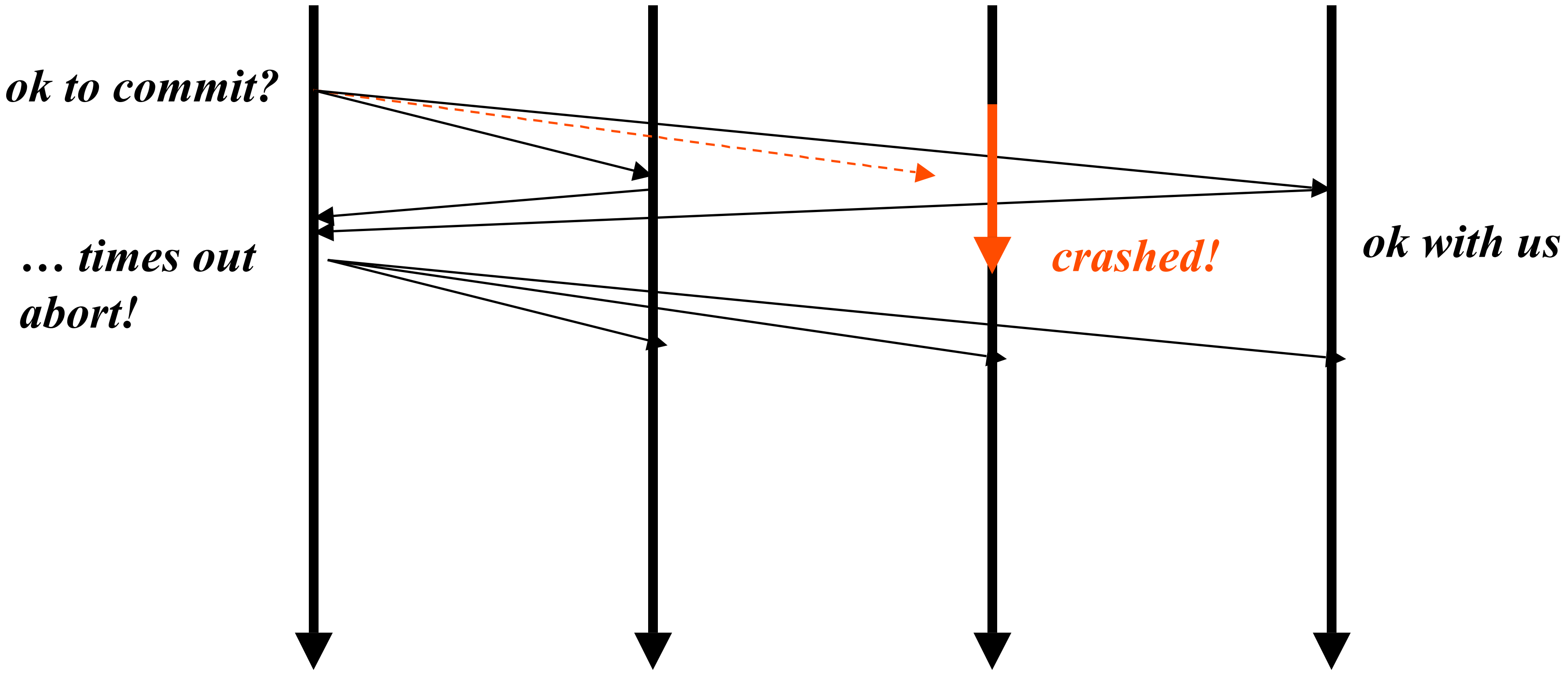
Failure model impacts costs!

- Byzantine model is very costly: 3*f*+1 processes needed to overcome *f* failures, protocol runs in *f*+1 rounds
- This cost is unacceptable for most real systems, hence protocols are rarely used
- Main area of application: hardware fault-tolerance, security systems

Commit with simpler failure model

- Assume processes fail by halting
- Coordinator detects failures (unreliably) using timouts. It can make mistakes!
- Now the challenge is to terminate the protocol if the coordinator fails instead of, or in addition to, a participant!

Commit protocol illustrated



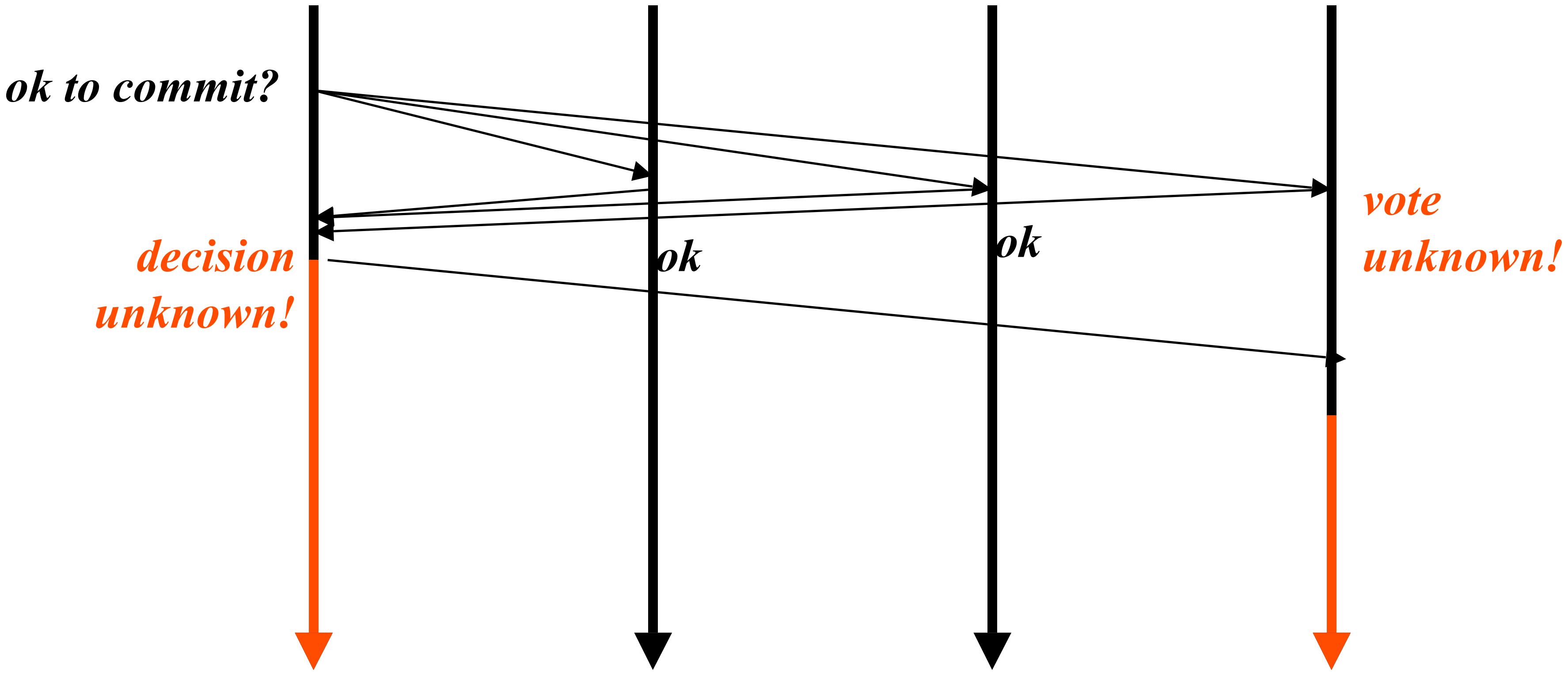
Note: garbage collection protocol not shown here

Example of a hard scenario

- Coordinator starts the protocol
- One participant votes to abort, all others to commit
- Coordinator and one participant now fail

... *we now lack the information to correctly terminate the protocol!*

Commit protocol illustrated



Example of a hard scenario

- Problem is that if coordinator told the failed participant to abort, all must abort
- If it voted for commit and was told to commit, all must commit
- Surviving participants can’t deduce the outcome without knowing how failed participant voted
- Thus protocol “blocks” until recovery occurs

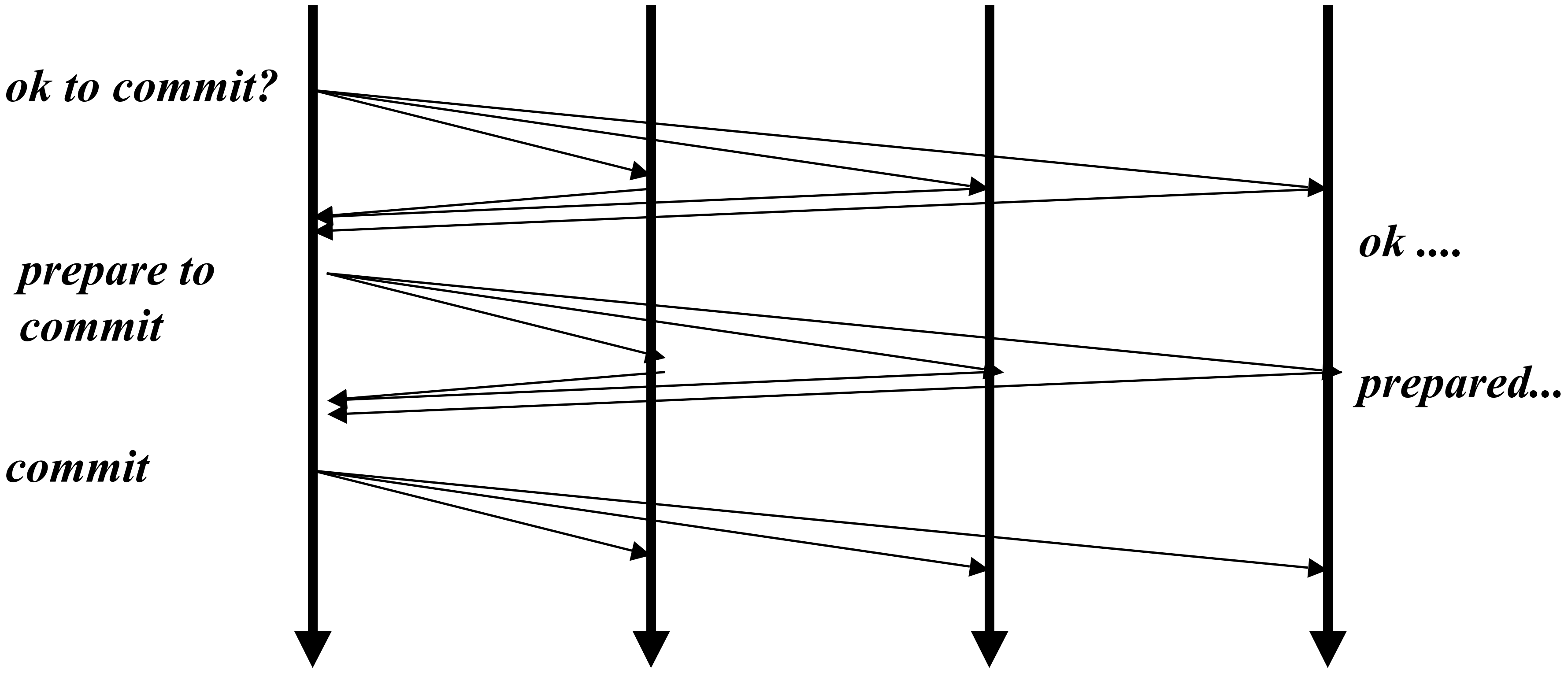
Skeen ('82): Three-phase commit

- Seeks to increase availability
- Makes an unrealistic assumption that failures are accurately detectable
- With this, can terminate the protocol even if a failure does occur

Three-phase commit

- Coordinator starts protocol by sending request
- Participants vote to commit or to abort
- Coordinator collects votes, decides on outcome
- Coordinator can abort immediately
- To commit, coordinator first sends a “prepare to commit” message
- Participants acknowledge, commit occurs during a final round of “commit” messages

Three phase commit protocol illustrated



Note: garbage collection protocol not shown here

Observations about 3PC

- If any process is in “prepare to commit” all voted for commit
- Protocol commits only when all surviving processes have acknowledged prepare to commit
- After coordinator fails, it is easy to run the protocol forward to commit state (or back to abort state)

Assumptions about failure

- If the coordinator suspects a failure, the failure is “real” and the faulty process, if it later recovers, will know it was faulty
- Failures are detectable with bounded delay
- On recovery, process must go through a reconnection protocol to rejoin the system! (Find out status of pending transactions that terminated while it was not operational)

Problems with 3PC

- With realistic failure detectors (that can make mistakes), protocol still blocks!
- Bad case arises during “network partitioning” when the network splits the participating processes into two or more sets of operational processes
- Can prove that this problem is not avoidable: there are no non-blocking commit protocols for asynchronous networks

Situation in practical systems

- Most use protocols based on 2PC: 3PC is more costly and ultimately, still subject to blocking!
- Need to extend with a form of garbage collection to avoid accumulation of protocol state information (can occur in the background)
- Some systems simply accept the risk of blocking when a failure occurs
- Others reduce the consistency property to make progress at risk of inconsistency after failure.