Distributed Transactions

Dominic Duggan

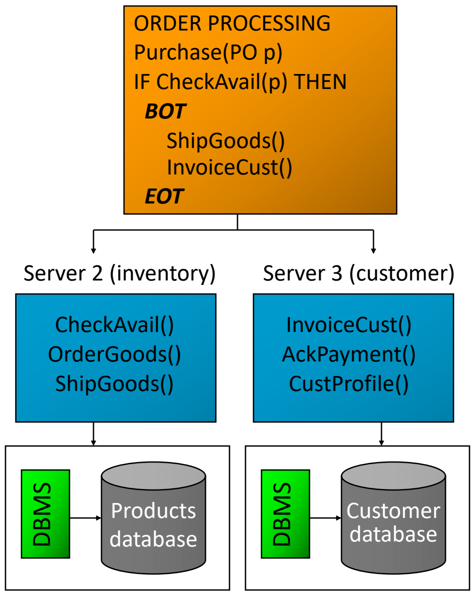
Stevens Institute of Technology

Based in part on materials by K. Birman

1

1

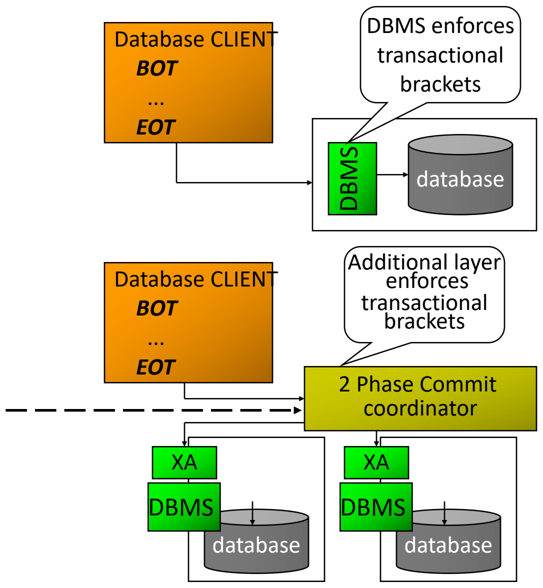
Client, server, and databases



2

2

Multiple Databases

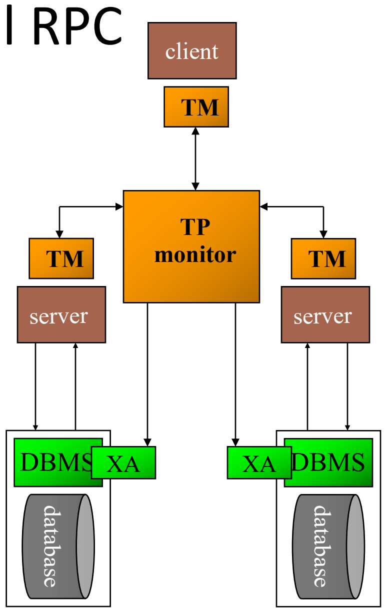

Database CLIENT
BOT
...
EOT
DBMS enforces transactional brackets
DBMS
database
Database CLIENT
BOT
...
EOT
Additional layer enforces transactional brackets
2 Phase Commit coordinator
XA
DBMS
database
XA
DBMS
database

3

3

Transactional RPC

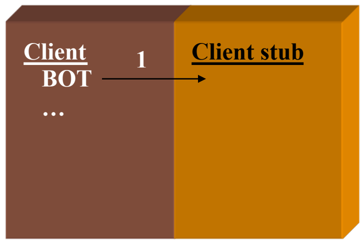
- Transaction Managers
- Transaction Processing Monitor (TPM)
- XA API



4

4

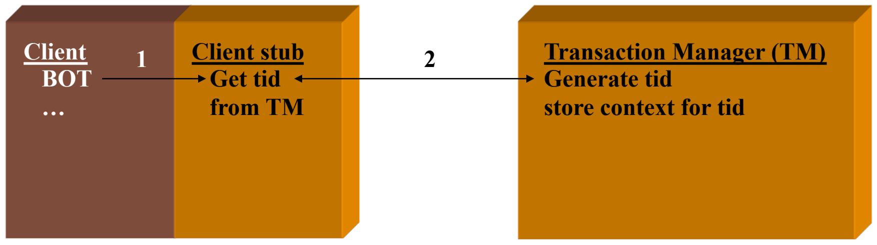
Basic TRPC (making calls)



5

5

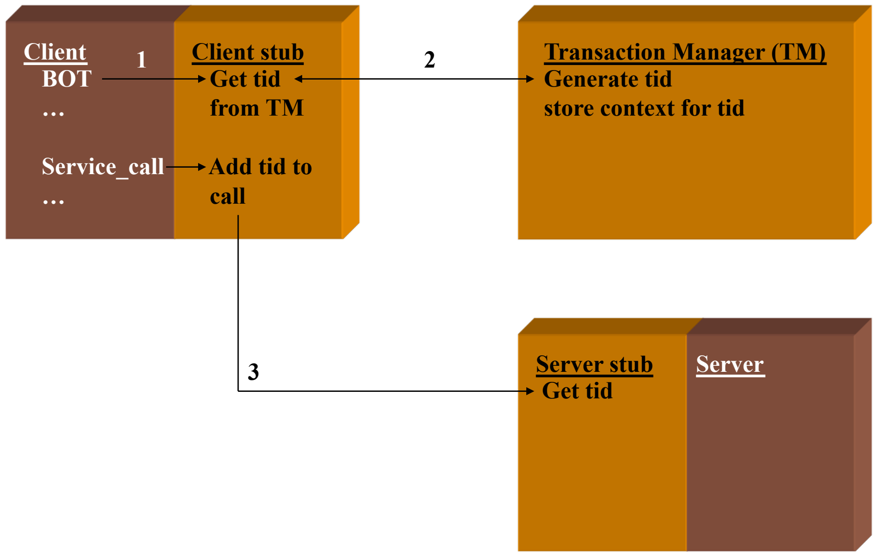
Basic TRPC (making calls)



6

6

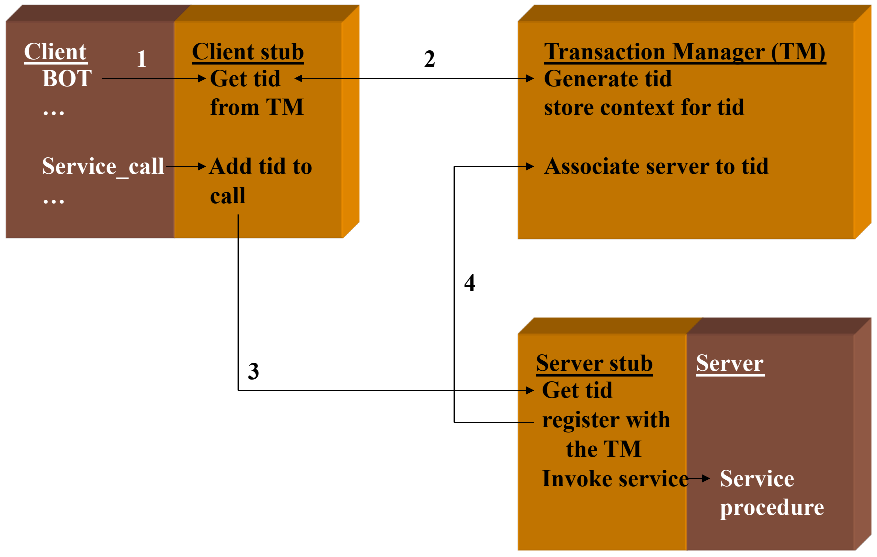
Basic TRPC (making calls)



7

7

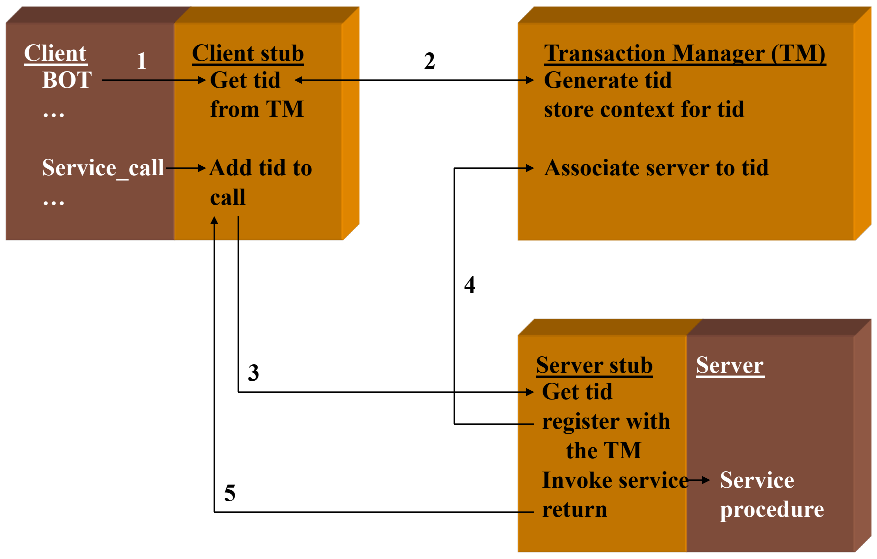
Basic TRPC (making calls)



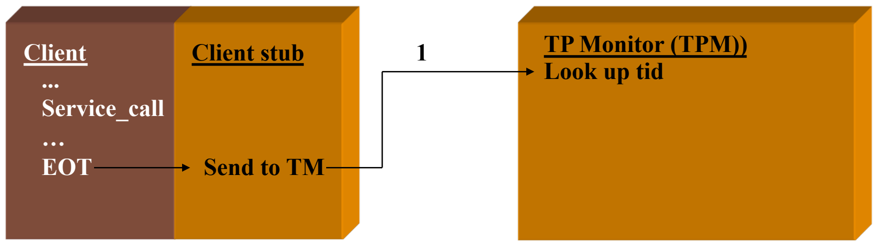
8

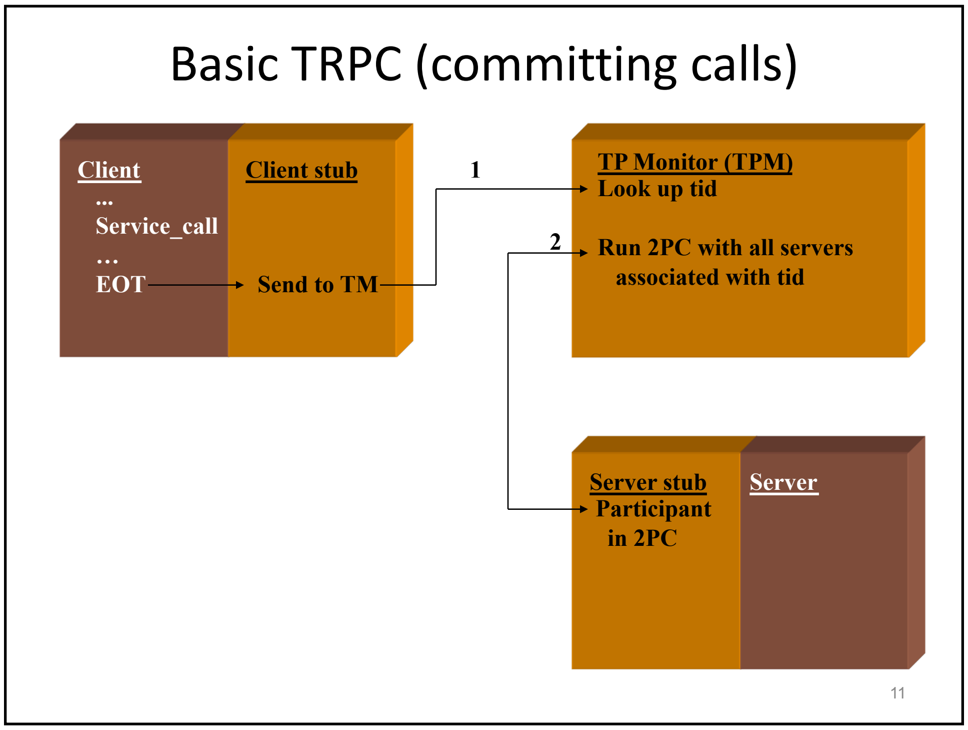
8

Basic TRPC (making calls)



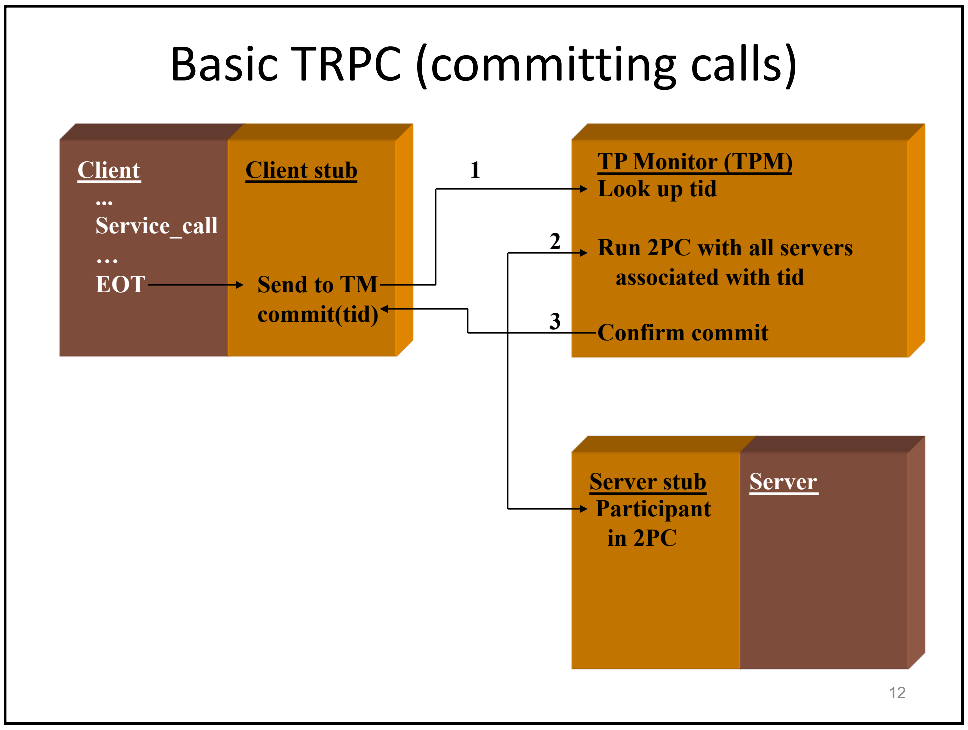
Basic TRPC (committing calls)





Basic TRPC (committing calls)

11



Basic TRPC (committing calls)

12

TWO PHASE COMMIT (2PC)

13

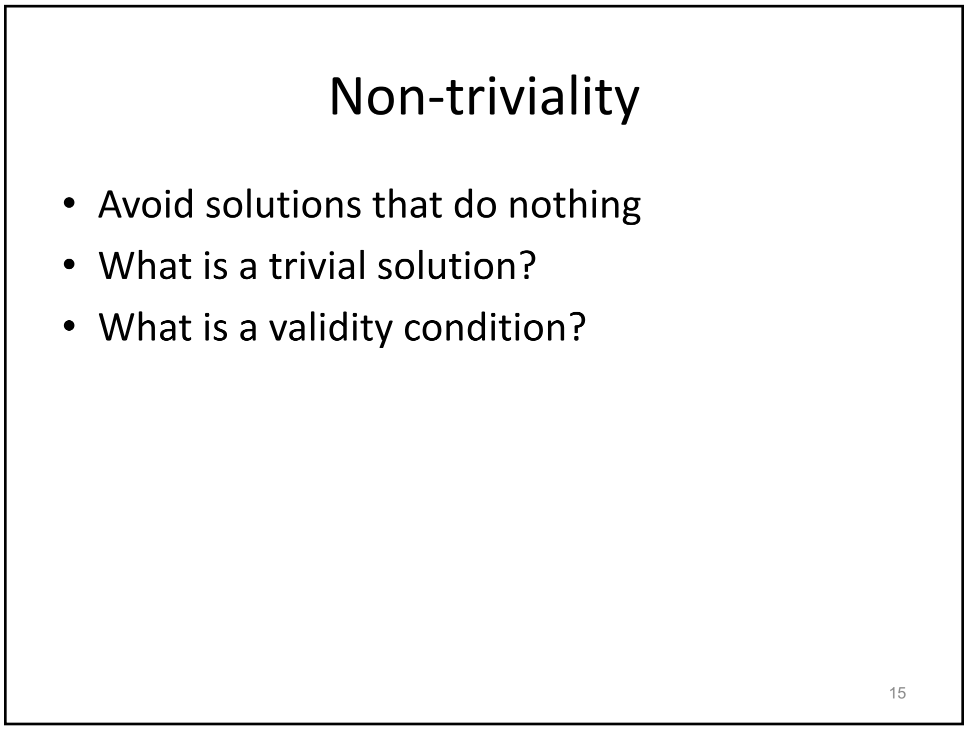
13

Atomic Commitment

- Given a set of processes
- **Coordinator** (i.e. TPM) wants to initiate an action (commit)
- **Participants** may vote for or against the action
- Perform the action only if all vote in favor
- Otherwise abort
- Goal is *all-or-nothing* outcome

14

14

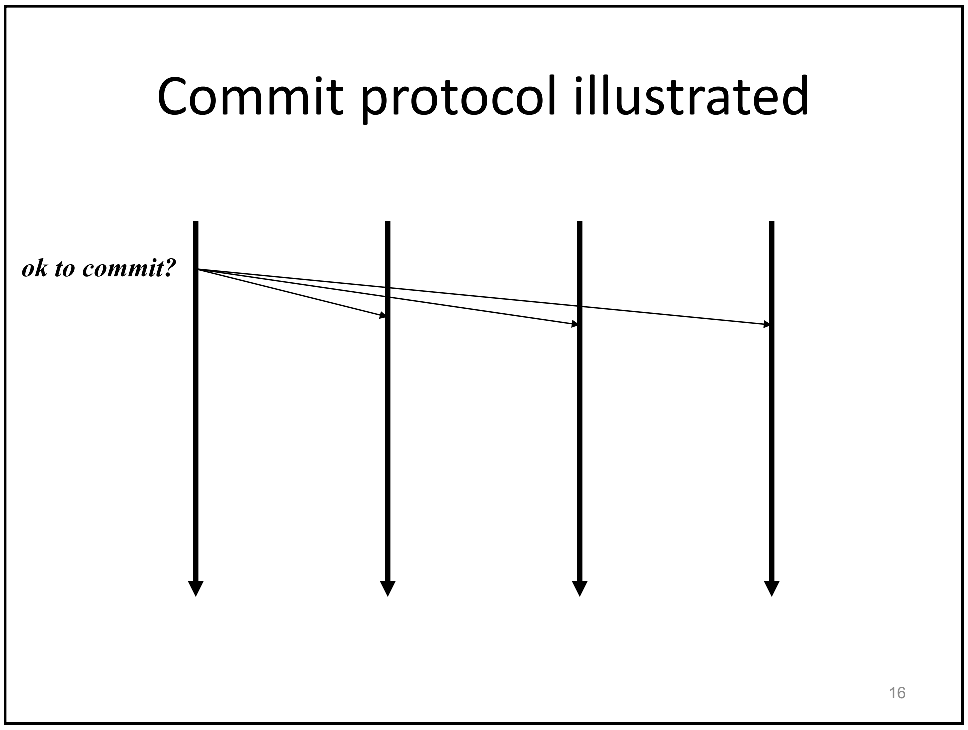


Non-triviality

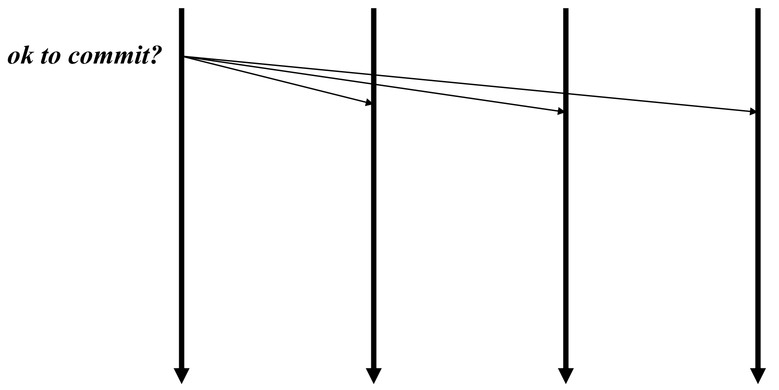
- Avoid solutions that do nothing
- What is a trivial solution?
- What is a validity condition?

15

15



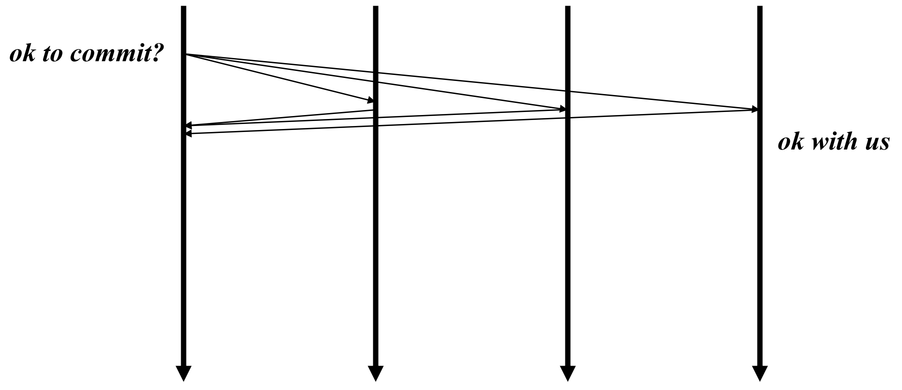
Commit protocol illustrated



16

16

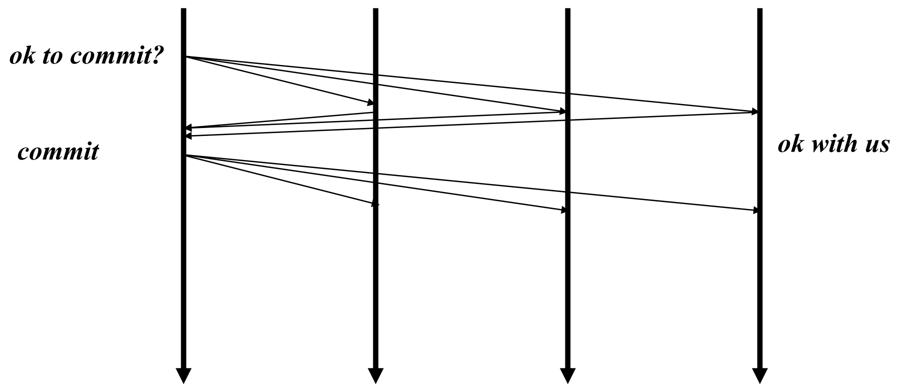
Commit protocol illustrated



17

17

Commit protocol illustrated



Note: garbage collection protocol not shown here

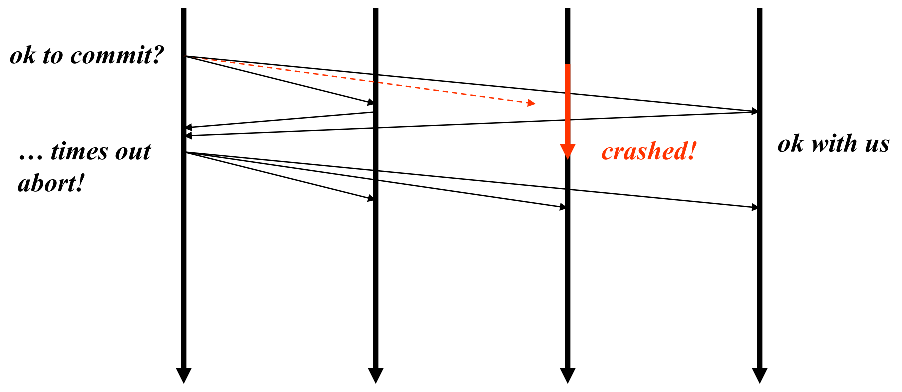
18

18

Two-phase Commit

- Phase 0: Flush caches on Web, app server
- Phase 1: Coordinator asks participants for vote
  - – Data managers force updates to the log
  - – Then say “ok to commit”
- Phase 2: If all are ok to commit, then coordinator tells participants to commit. Otherwise, abort.
- Data managers then make updates permanent or rollback to old values, and release locks

Commit protocol illustrated



Note: garbage collection protocol not shown here

Non-triviality

- Avoid solutions that do nothing
- Commit validity: if all vote for commit, protocol must commit
- …but what if participant vote is lost?
- “Non-triviality” condition hard to capture

21

21

Unilateral abort

- Any data manager can unilaterally abort a transaction until it has said “prepared”
- Implication: even a data manager where only reads were done must participate in 2PC protocol!

22

22

PROBLEMS WITH 2PC

Non-blocking Commit

- Goal: a protocol that allows all operational processes to terminate the protocol even if some subset crash

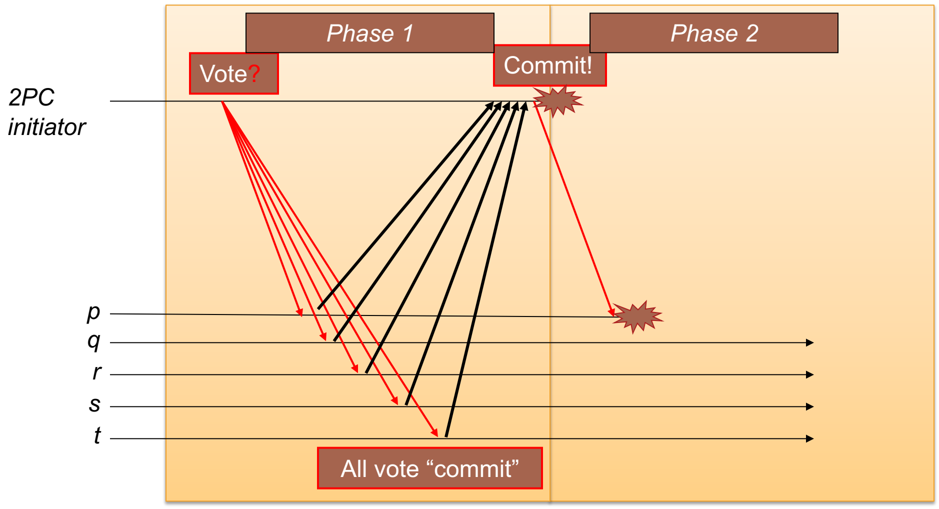
Commit with unreliable failure detectors

- Assume processes fail by crashing
  - No Byzantine failures
- Coordinator detects failures (unreliably) using timouts
- Challenge: terminate the protocol if the coordinator fails

25

25

2 phase commit



26

THREE PHASE COMMIT (3PC)

27

27

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

28

28

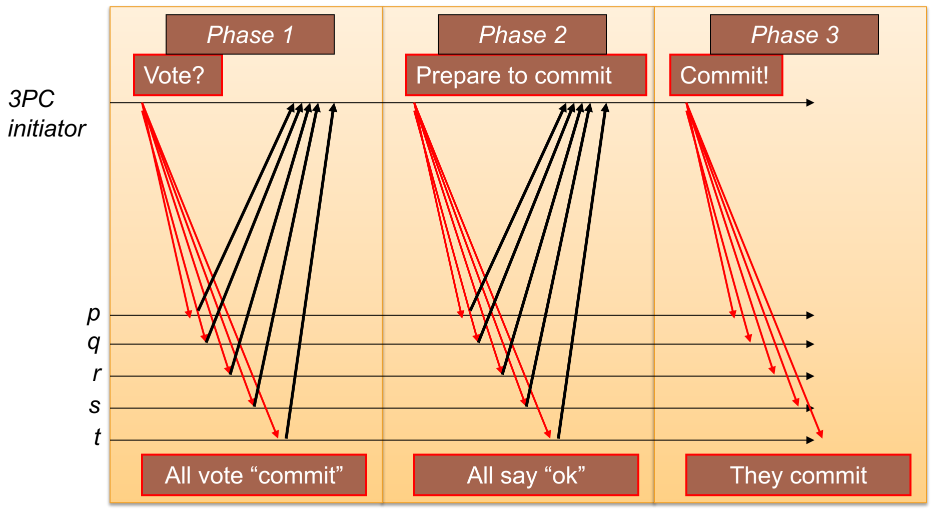
Three-phase commit

- 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

29

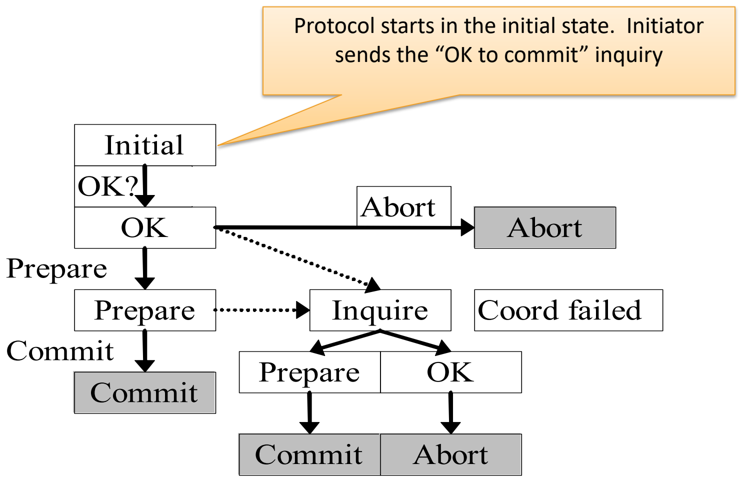
29

Three-phase commit



30

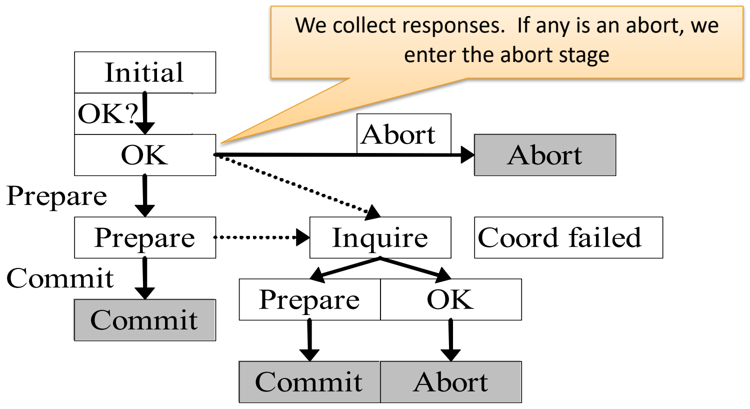
State diagram for non-faulty member



31

31

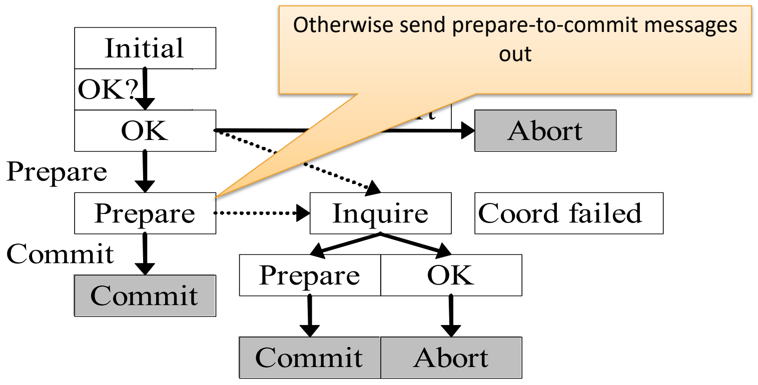
State diagram for non-faulty member



32

32

State diagram for non-faulty member

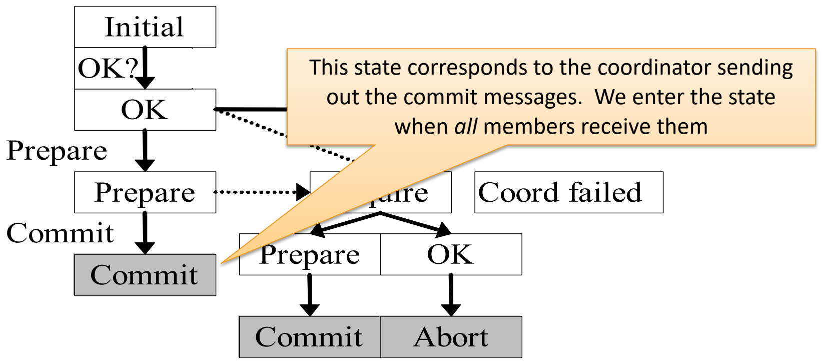


Otherwise send prepare-to-commit messages out

33

33

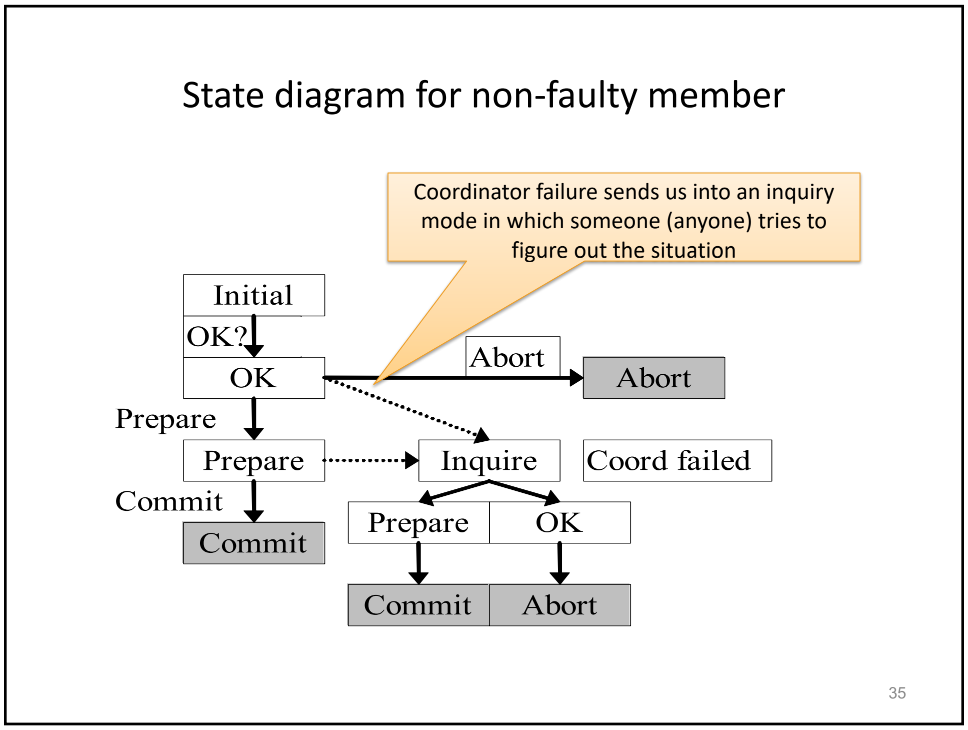
State diagram for non-faulty member



This state corresponds to the coordinator sending out the commit messages. We enter the state when *all* members receive them

34

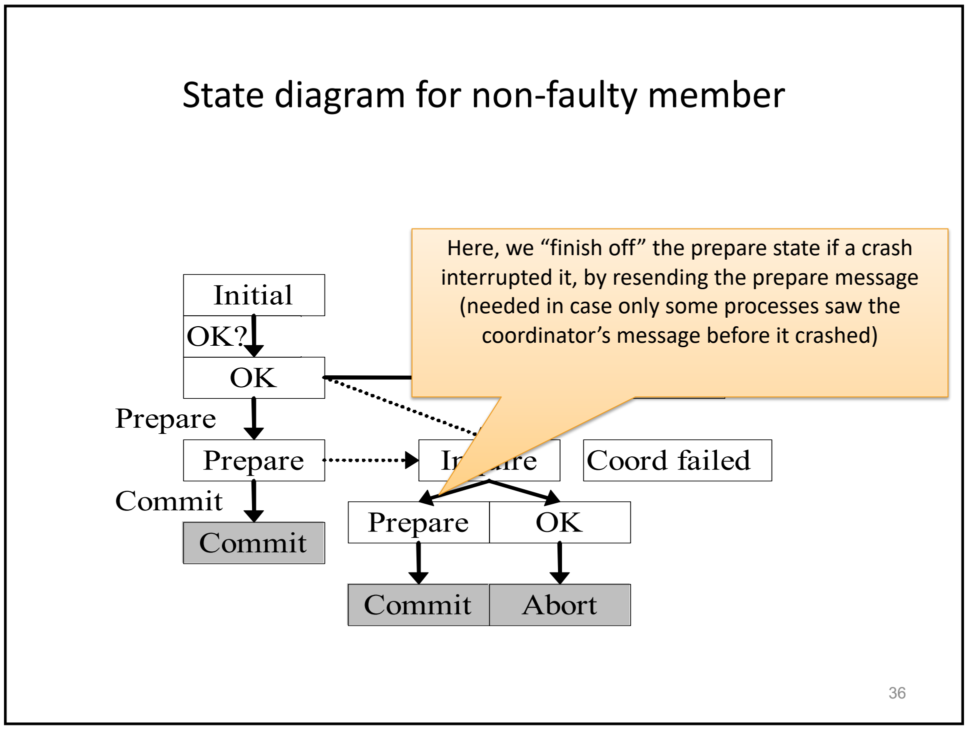
34



State diagram for non-faulty member

Coordinator failure sends us into an inquiry mode in which someone (anyone) tries to figure out the situation

35

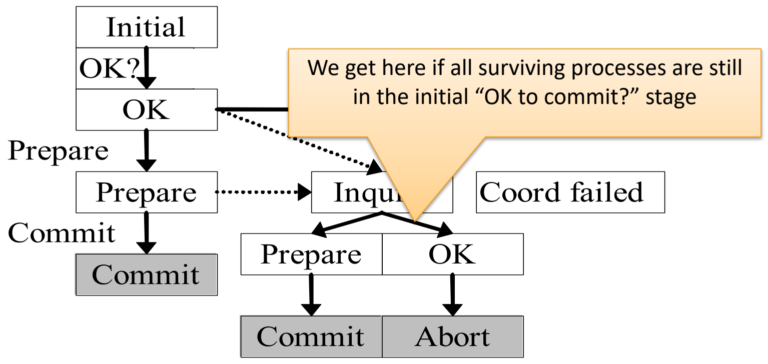


State diagram for non-faulty member

Here, we "finish off" the prepare state if a crash interrupted it, by resending the prepare message (needed in case only some processes saw the coordinator's message before it crashed)

36

State diagram for non-faulty member

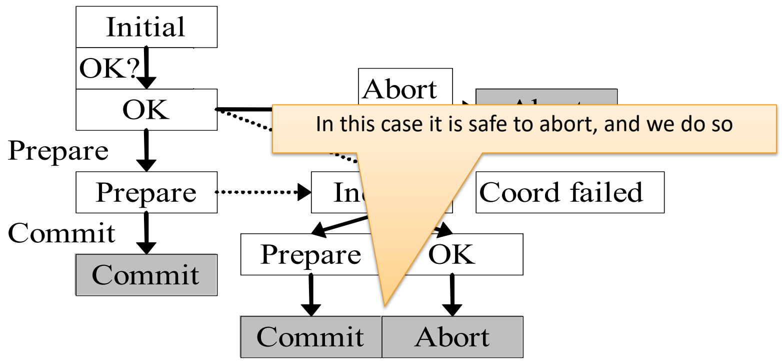


We get here if all surviving processes are still in the initial “OK to commit?” stage

37

37

State diagram for non-faulty member



In this case it is safe to abort, and we do so

38

38

Observations about 3PC

- Key point: Extra buffer state
- What if none of surviving participants have heard from coordinator?
  - After voting phase
  - 2PC: Some crashed processes may have committed
  - 3PC: No crashed process has committed yet (Why?)

39

39

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)

40

40

Problems with 3PC

- Assumes reliable failure detectors
- But even with realistic failure detectors (that can make mistakes), protocol still blocks!
  – "Network partitioning"
- Can prove that this problem is not avoidable

41

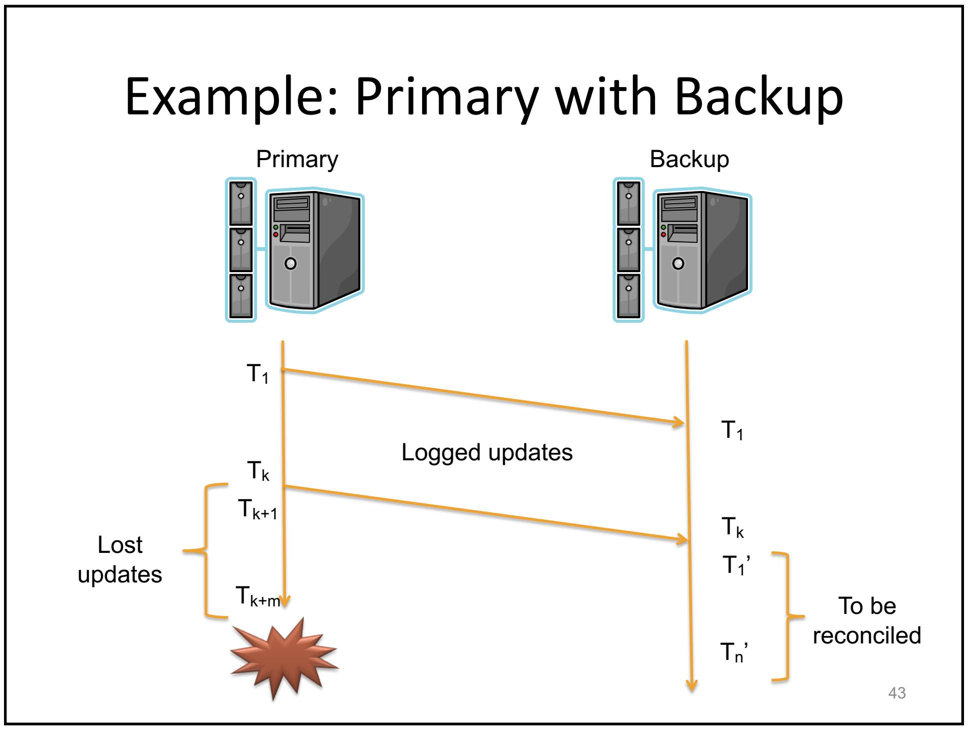
41

Situation in practical systems?

- Most use protocols based on 2PC
- Need to extend garbage collection
  – protocol state information
- Some systems accept the risk of blocking
- Others reduce the consistency property to make progress

42

42



Example: Primary with Backup

43