MySQL

Transactions

Intel (TX memory):
Transactional
Synchronization
Extensions (TSX)

PostgreSQL

Goal – A Distributed Transaction

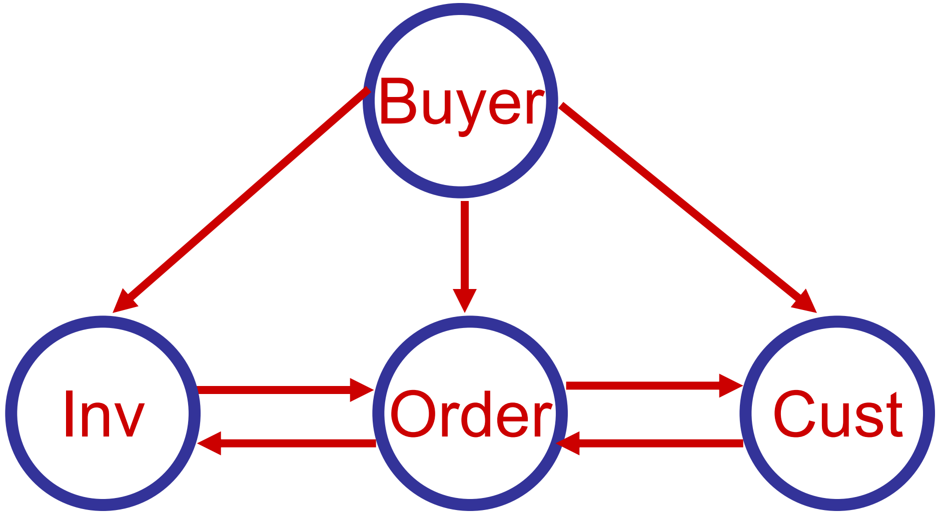
- We want a transaction that involves multiple nodes
- Review of transactions and their properties
- Things we need to implement transactions
  - Locks
  - Achieving atomicity through logging
    - Roll ahead, roll back, write ahead logging
- Finally, 2 Phase Commit (aka 2PC) and 3PC
- Lead into Paxos

Transactions summary

- Key properties
  - ACID
- Serializability and Independence
  - two phase locking
    - serializability
  - strict two phase locking
    - Serializability and Independence
- Recovery
  - redo and/or undo logging

Trans in Distributed Systems

- A distributed transaction involves
  - updates at multiple nodes
  - and the messages between those nodes
- For example, buying widgets



Distributed transactions

- Easy part
  - Make the transaction manager a distributed service
- Hard part
  - Distributed atomic commit
  - right now we get it with atomic disk write of *commit* record to transaction log.
  - how do we get it if there are multiple nodes involved in the transaction? (and how do we even define it?)

Distributed Atomic Commit Requirements

1. All workers that reach a decision reach the same one
2. Workers cannot change their decisions on commit or abort once a decision is made
3. To commit all workers must vote commit
4. If all workers vote commit and there are no failures the transaction will commit
5. If all failures are repaired and there are no more failures each worker will eventually reach a decision (all will reach the same decision)

Atomic commit using coordinator

- Transaction coordinator
  - issues TID to workers (that submit it transactions)
  - knows about all workers
  - provides atomic commit
  - maintains a log of decisions/progress
- Workers
  - contact coordinator to begin and commit trans, to respond to votes and to determine outcome when uncertain
  - maintain local log of updates (reminder: each worker has its own DB)

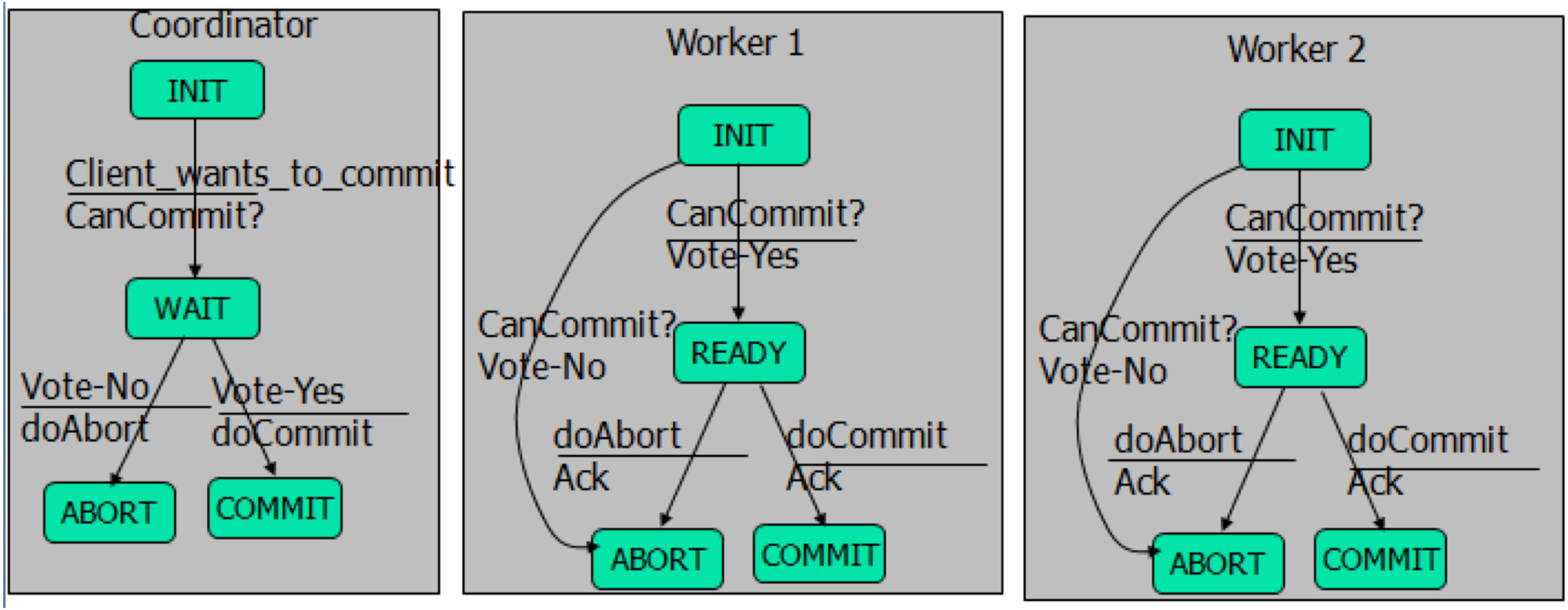
Two phase commit

- Transaction logs
  - coordinator – *begin*, *commit*, and *abort*
  - worker – *b*, *c*, *a*, *update*, and *prepared*
- Messages
  - w ➔ c: commitRequest
  - c ➔ w: prepareToCommit
    - End of phase 1
  - w ➔ c: prepared or abort
  - c ➔ w: committed or abort
    - End of phase 2

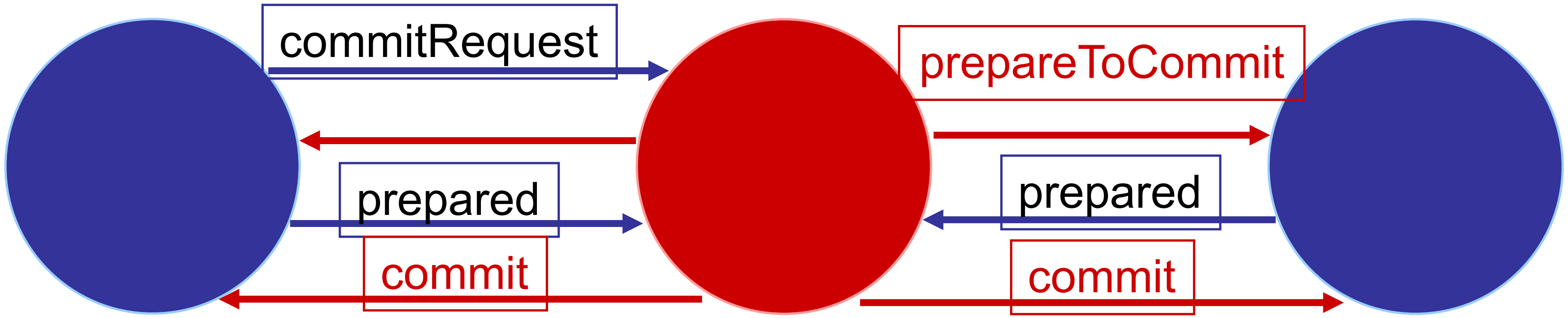
Two phase commit

- Phase 1 (voting)
  - worker – sends commitRequest to coord
  - cord – sends prepareToCommit to all workers
  - worker – writes *prepared* to its log and sends prepared to coord, then waits
- Phase 2 (completing the transaction)
  - coord – waits for prepared from all workers
    - a no from any worker aborts the transaction
  - coord – writes *commit* to its transaction log
    - transaction is now committed
  - coord – sends committed to workers
  - worker – write *commit* to log when committed recvd

2PC state machines



Two phase commit in action



[u,tid,xxx]

[p,tid]

[c,tid]

[b,tid]

[c,tid]

[u,tid,xxx]

[p,tid]

[c,tid]

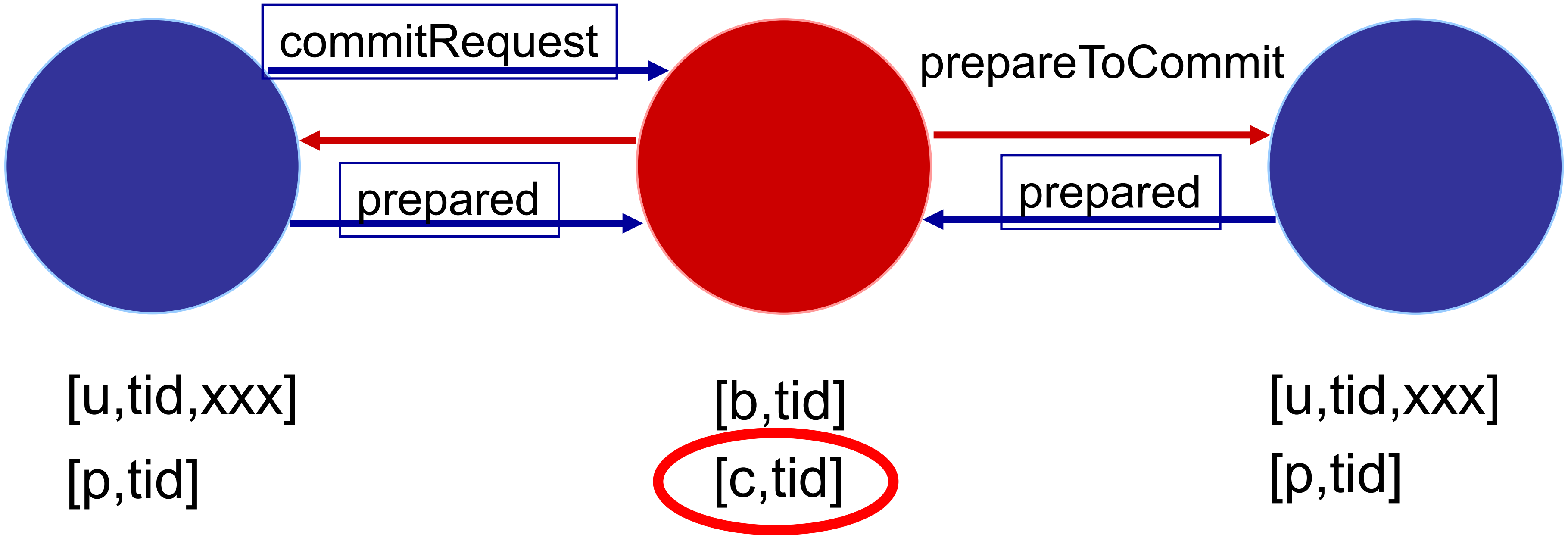
Failure of worker (after prepareToCommit sent)

- Coordinator action
  - Coordinator's prepareToCommit has a corresponding timeout and it aborts transaction if worker fails to reply
- Worker recovery
  - Looks for log records with no decision (commit or abort) and no preparedToCommit record
    - Locally abort transaction

Failure of worker (after replied with prepared)

- Any transaction with a P and no C (or A) in the worker's log
  - worker does not know if transaction committed
  - must send message to coordinator to find out
    - If coordinator is down could it send a message to another worker?
- Key observation:
  - once worker has sent prepared, the transaction, from a high level, could commit at any time, even if the worker has not received the commit message.

Two phase commit in action (2)



commitRequest

prepareToCommit

prepared

prepared

[u,tid,xxx]

[p,tid]

[b,tid]

[c,tid]

[u,tid,xxx]

[p,tid]

This transaction has committed, but workers don't know yet

Failure of coordinator

- worker sends commitRequest
  - timeout if no prepareToCommit received
  - abort transaction locally

- worker sends prepared (or aborted)
  - timeout if no committed (or aborted) received
  - worker does not know if transaction has committed
    - must check with someone

Determining transaction decision

- need to ask someone else when
  - coordinator fails with incomplete prepareToCommit
  - worker fails with P, but not C in its log
- ask coordinator
  - worker sends *decisionRequest(tid)* to coord
  - coord scans log for this tid
    - sends committed or aborted back to worker
- problem?

Coordinator Unavailable

- Worker checks with other workers (got list of workers with the prepareToCommit)
  - Some worker has commit – then commit the transaction
  - Some worker has abort – then abort the transaction
  - Some worker has no prepared – it can abort
  - All workers have prepared – block indefinitely (in some cases may be OK to select a new coordinator – when?)

Coordinator Unavailable

- Worker checks with other workers (got list of workers with the prepareToCommit)
  - Some worker has commit – then commit the transaction
  - Some worker has abort – then abort the transaction
  - Some worker has no prepared – it can abort
  - All workers have prepared – block indefinitely (in some cases may be OK to select a new coordinator – when?)
    - When you know that the old coordinator won't come back up (otherwise could reach inconsistent decision)

Coordinator Unavailable

- Worker checks with other workers (got list of workers with the prepareToCommit)
  - Some worker has commit – then commit the transaction
  - Some worker has abort – then abort the transaction
  - Some worker has no prepared – it can abort
  - All workers have prepared – block indefinitely (in some cases may be OK to select a new coordinator – when?)
    - When you know that the old coordinator won't come back up (otherwise could reach inconsistent decision)

In general, this is summarized as:

Two phase commit is always safe, but is not always live.