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 (again!)

Transactions - Definition

- A transaction is a sequence of data operations with the following properties:
  - **A** Atomic
    - All or nothing
  - **C** Consistent
    - Consistent state in => consistent state out
  - **I** Independent (Isolated)
    - Partial results are not visible to concurrent transactions
  - **D** Durable
    - Once completed, new state survives crashes

Summary

Isolation and serializability

- Definitions
  - isolation
    - no transaction can see incomplete results of another
  - serializability
    - actual execution same as some serial order
- Algorithms (based on locks)
  - two-phase locking
    - serializability
  - strict two-phase locking
    - isolation and serializability

© 2015 Donald Acton et al

Recoverability (Atomicity)

- Problem
  - ensure atomic update in face of failure
- If no failure, it's easy
  - just do the updates
- If failure occurs while updates are performed
  - Roll back to remove updates or
  - Roll forward to complete updates
  - What we need to do and when will depend on just when we crash

Logging

- **Persistent (on disk)** log
  - records information to support recovery and abort
- Types of logging
  - redo logging --- roll forward (log contains new values)
  - undo logging--- roll back (and abort) (log contains old values)
  - Write-ahead logging --- roll forward and back
- Types of log records
  - *begin*, *update*, *abort*, *commit*, and *truncate*
- Atomic update
  - atomic operation is write of *commit* record to disk
  - transaction committed iff *commit* record in log

Write-ahead logging

- Idea
  - combine undo and redo logging
- How
  - write old values to log
  - modify data
  - write new values to log anytime before commit
  - write commit record to log
  - write data back to disk at anytime, when done write truncate record to log (truncate is the indicator that data is on disk!)

Failure Recovery

- Commit but no truncate
  - Use roll forward based on new values (i.e., commit flushes all of the new values from the log)
- No commit
  - Use old value to roll back (i.e., stored old values, so perform undo of all modifications)

Shrinking the Log File (Truncation)

- Truncation is the process of
  - removing unneeded records from transaction log
- For redo logging
  - remove transactions with <u>truncate</u> or <u>abort</u>
- For undo logging
  - remove transactions with <u>commit</u> or <u>abort</u>

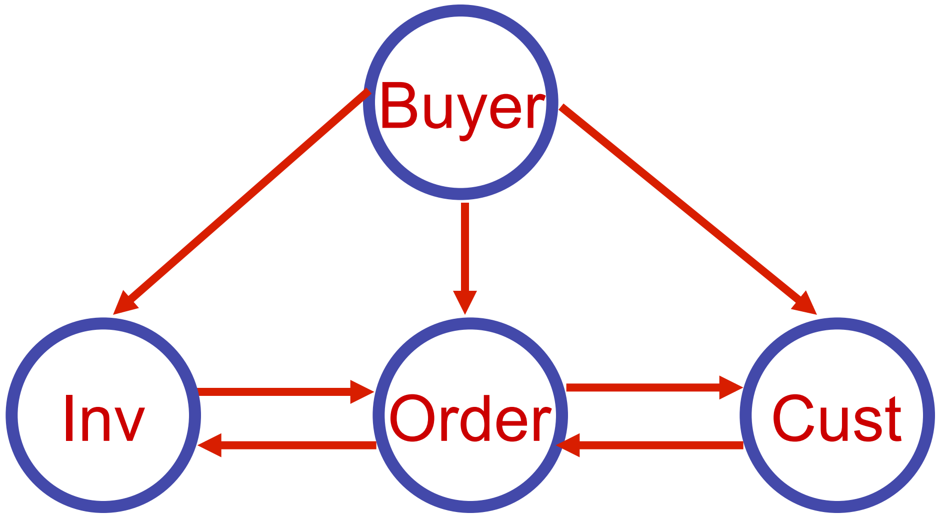
Transactions summary

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

© 2015 Donald Acton et al

Trans in Distributed Systems

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



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 (In fact it will be the same decision)

Distributed transactions

- Easy part
  - make transaction monitor a distributed service
- Hard part
  - 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?

Atomic commit using coordinator

- Transaction coordinator
  - issues TID to clients (called workers)
  - 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

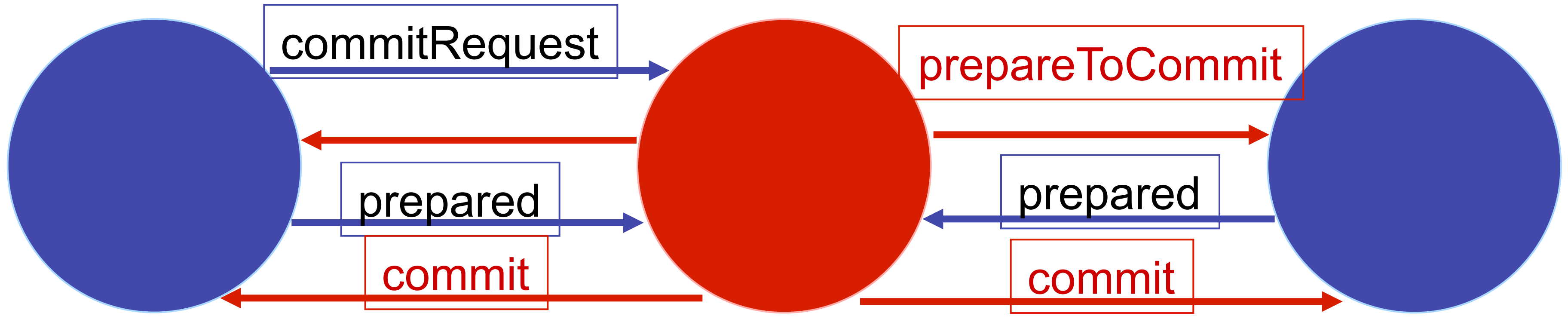
Two phase commit

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

Two phase commit

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

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)

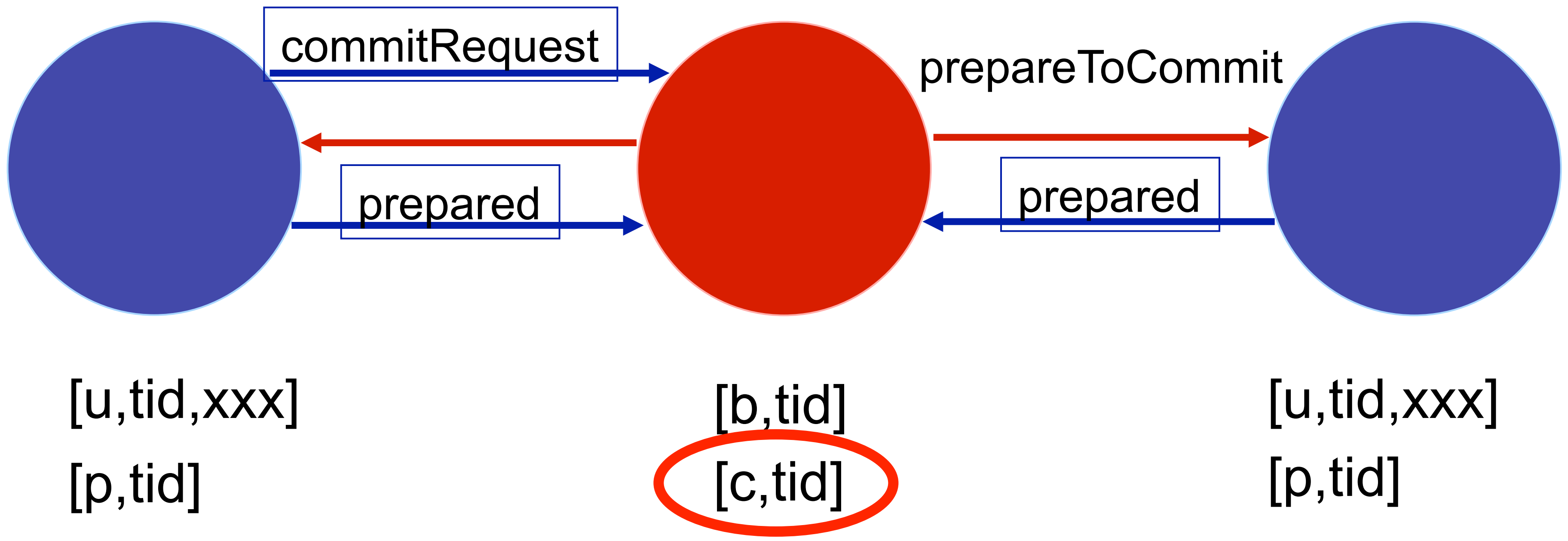
- Coordinator action
  - Coordinator’s prepareToCommit has a corresponding timeout and it aborts transaction if worker fails to reply
- Worker recovery
  - Looks for log records with no decision (commit or abort) and no preparedToCommit record
    - Locally abort transaction

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.

© 2015 Donald Acton et al

Two phase commit in action (2)



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

Failure recovery

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?

© 2015 Donald Acton et al

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?)