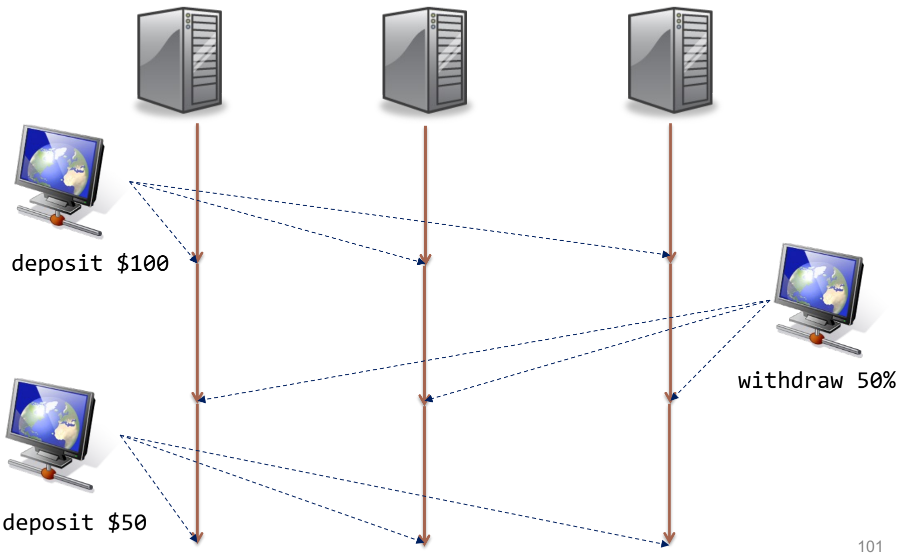
STATE MACHINE: ORDERING UPDATES

State Machine Repication

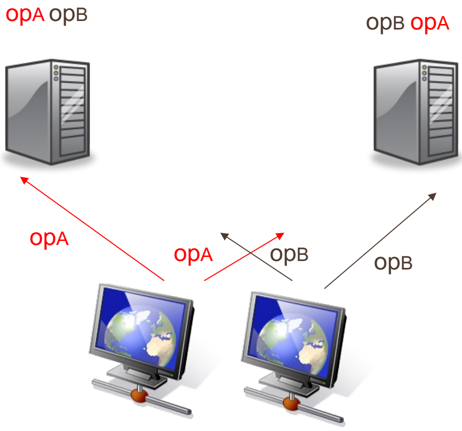
- Active replication (State machine)
  - Peer-to-peer replicas
  - Each replica is **deterministic** state machine
  - **Operations** executed in same order on all replicas
  - All updates are totally ordered

Consistent Ordering of Updates



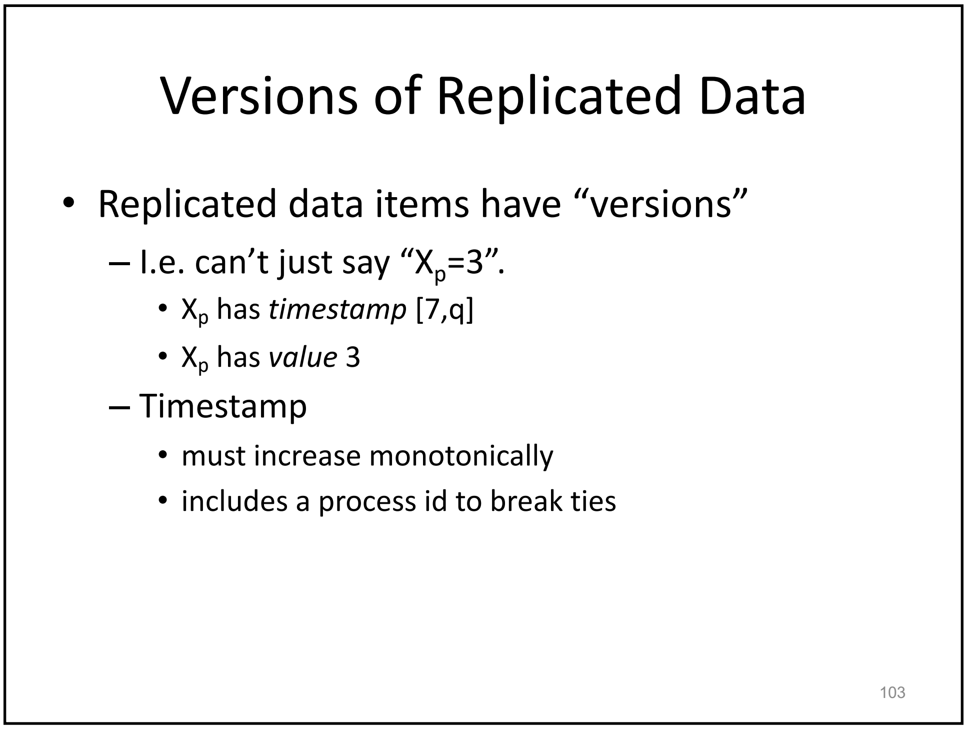
101

State Machine Replication



- How to maintain a single order in the face of concurrent client requests?

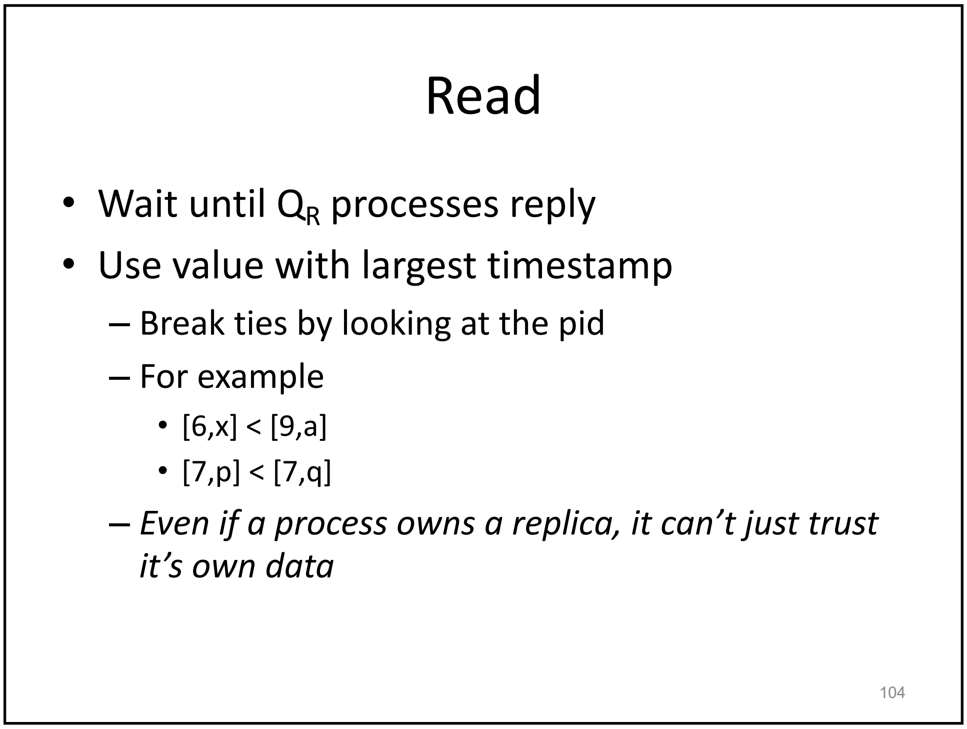
102



Versions of Replicated Data

- Replicated data items have “versions”
  - I.e. can’t just say “X_p=3”.
    - X_p has *timestamp* [7,q]
    - X_p has *value* 3
  - Timestamp
    - must increase monotonically
    - includes a process id to break ties

103



Read

- Wait until Q_R processes reply
- Use value with largest timestamp
  - Break ties by looking at the pid
  - For example
    - [6,x] < [9,a]
    - [7,p] < [7,q]
  - *Even if a process owns a replica, it can’t just trust it’s own data*

104

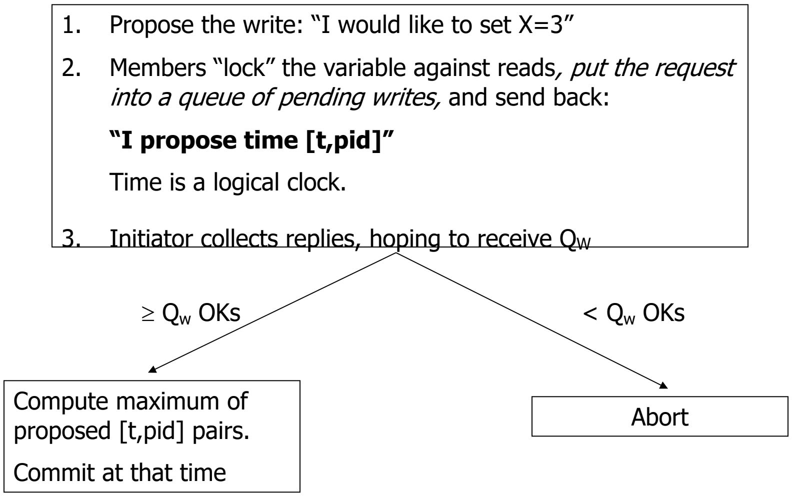
Write

- Can’t support incremental updates
  - x=x+1
  - Insert into a queue
- Quorum
  - Use a commit protocol
- How to determine the version number
  - Voting protocol

105

105

Protocol



1. Propose the write: “I would like to set X=3”
2. Members “lock” the variable against reads*, put the request into a queue of pending writes,* and send back:

   “I propose time [t,pid]”

   Time is a logical clock.
3. Initiator collects replies, hoping to receive Q_W

$\geq Q_w$ OKs

Compute maximum of proposed [t,pid] pairs.

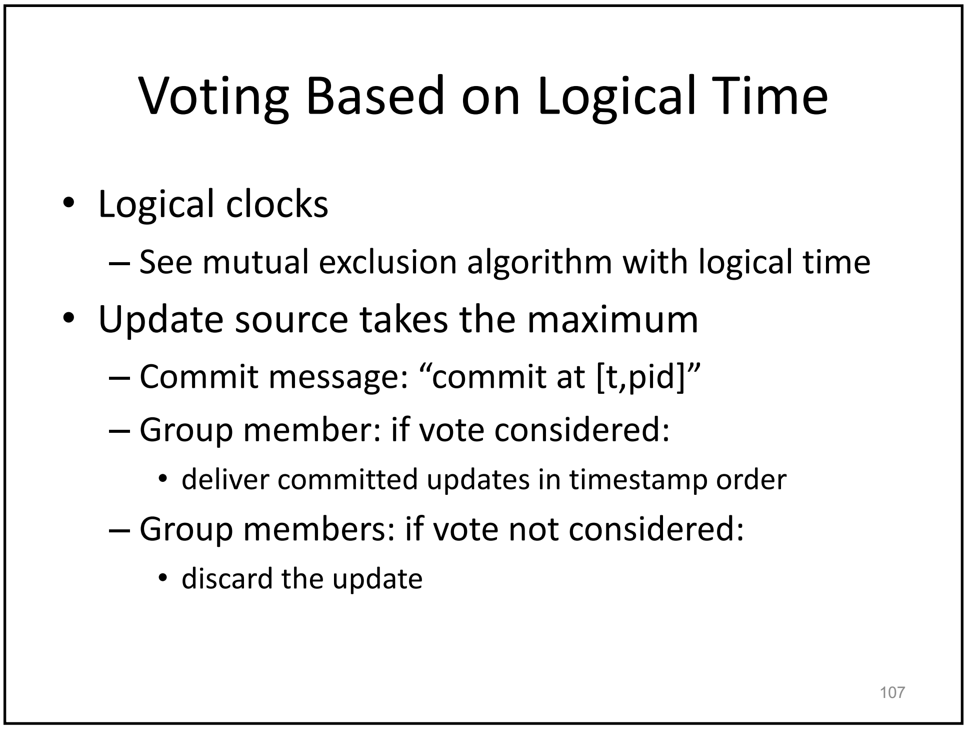
Commit at that time

$< Q_w$ OKs

Abort

106

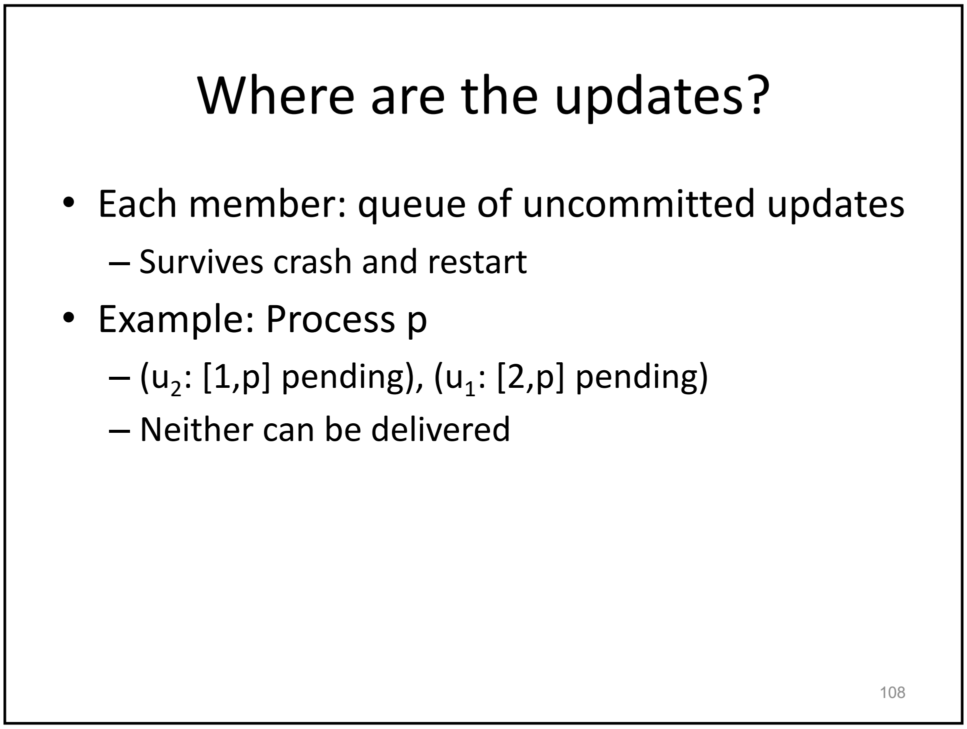
106



Voting Based on Logical Time

- Logical clocks
  - See mutual exclusion algorithm with logical time
- Update source takes the maximum
  - Commit message: “commit at [t,pid]”
  - Group member: if vote considered:
    - deliver committed updates in timestamp order
  - Group members: if vote not considered:
    - discard the update

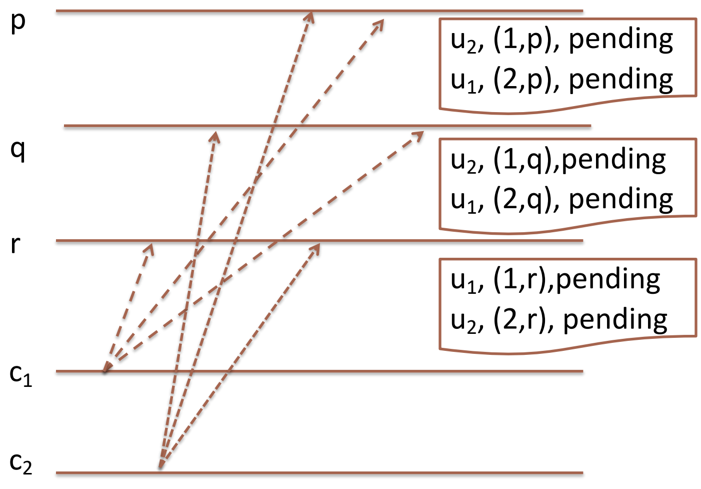
107



Where are the updates?

- Each member: queue of uncommitted updates
  - Survives crash and restart
- Example: Process p
  - (u_2: [1,p] pending), (u_1: [2,p] pending)
  - Neither can be delivered

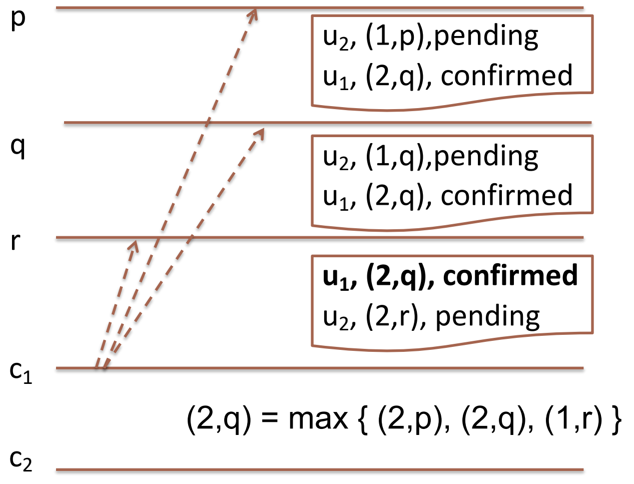
108



u_2, (1,p), pending
u_1, (2,p), pending

u_2, (1,q),pending
u_1, (2,q), pending

u_1, (1,r),pending
u_2, (2,r), pending

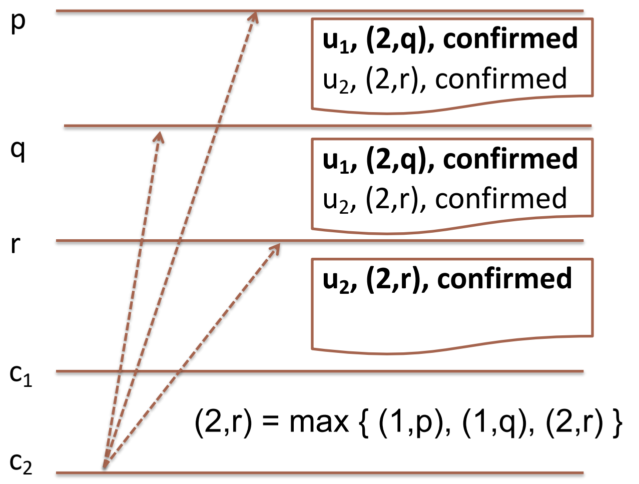


u_2, (1,p),pending
u_1, (2,q), confirmed

u_2, (1,q),pending
u_1, (2,q), confirmed

u_1, (2,q), confirmed
u_2, (2,r), pending

(2,q) = max { (2,p), (2,q), (1,r) }



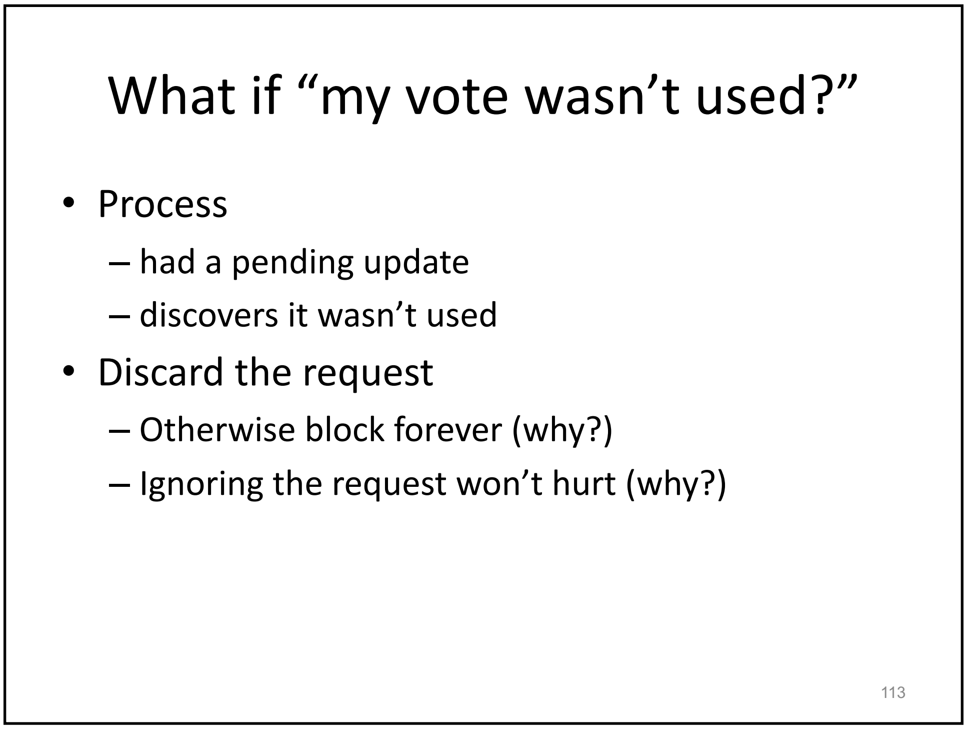
u_1, (2,q), confirmed
u_2, (2,r), confirmed

u_1, (2,q), confirmed
u_2, (2,r), confirmed

u_2, (2,r), confirmed

(2,r) = max { (1,p), (1,q), (2,r) }

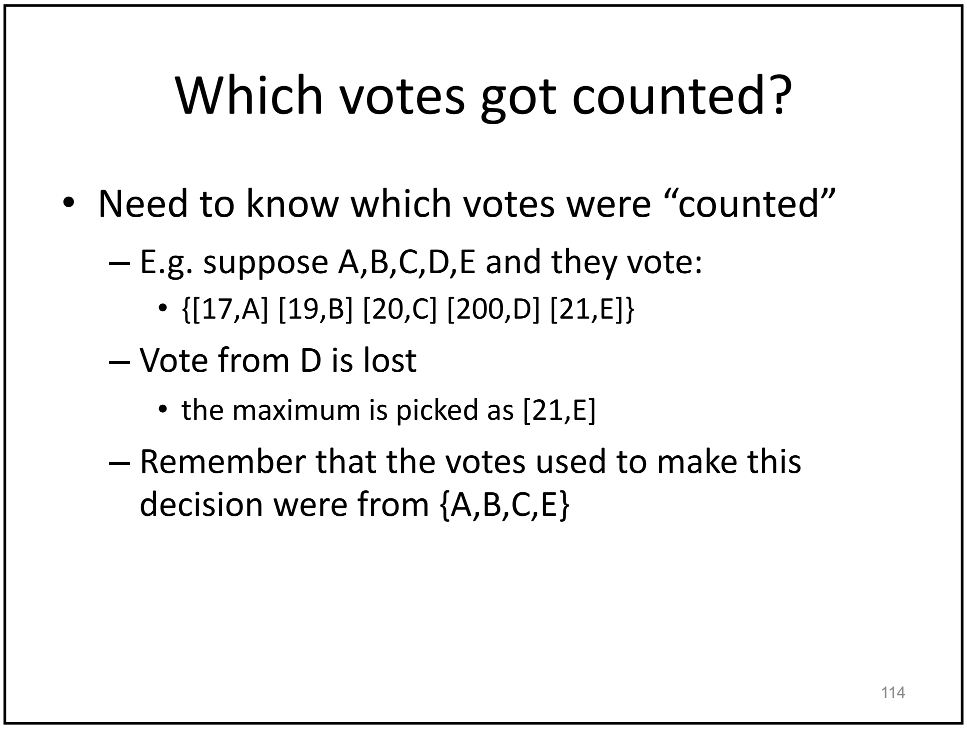
STATE MACHINE: PROTOCOL ANALYSIS



What if “my vote wasn’t used?”

- Process
  - had a pending update
  - discovers it wasn’t used
- Discard the request
  - Otherwise block forever (why?)
  - Ignoring the request won’t hurt (why?)

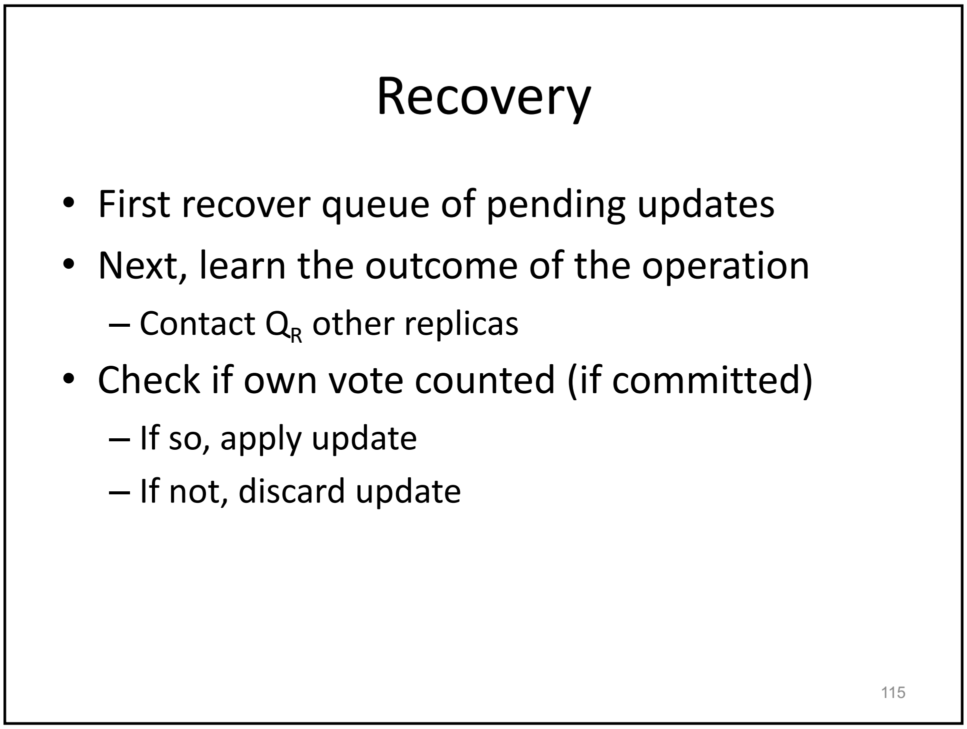
113



Which votes got counted?

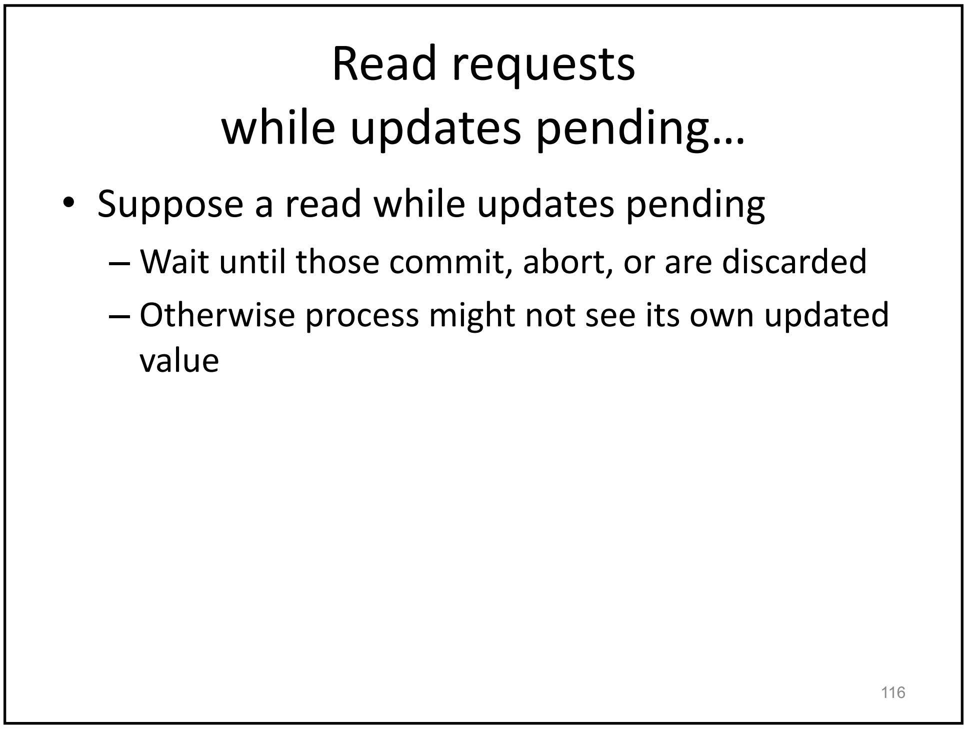
- Need to know which votes were “counted”
  - E.g. suppose A,B,C,D,E and they vote:
    - {[17,A] [19,B] [20,C] [200,D] [21,E]}
  - Vote from D is lost
    - the maximum is picked as [21,E]
  - Remember that the votes used to make this decision were from {A,B,C,E}

114



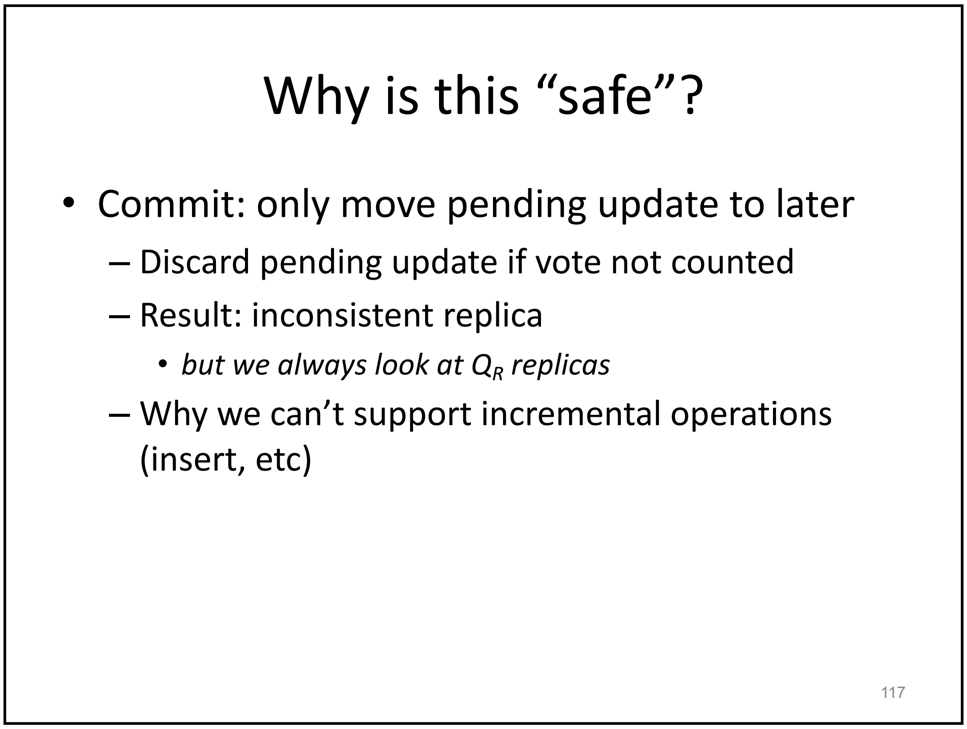
Recovery

- First recover queue of pending updates
- Next, learn the outcome of the operation
  - Contact Q_R other replicas
- Check if own vote counted (if committed)
  - If so, apply update
  - If not, discard update



Read requests while updates pending…

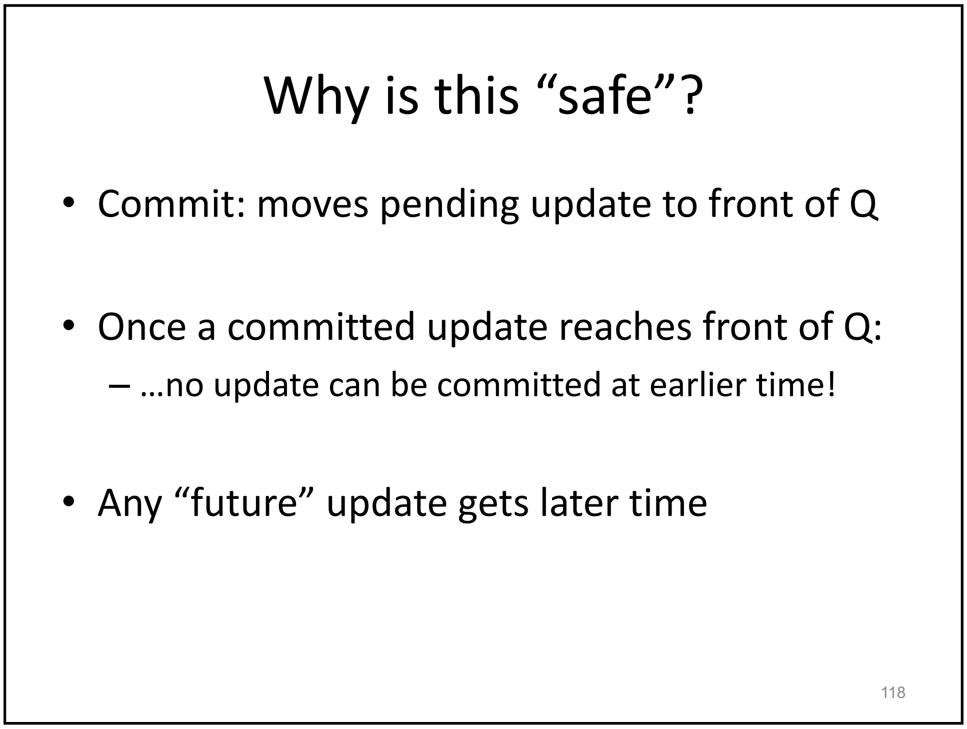
- Suppose a read while updates pending
  - Wait until those commit, abort, or are discarded
  - Otherwise process might not see its own updated value



Why is this “safe”?

- Commit: only move pending update to later
  - – Discard pending update if vote not counted
  - – Result: inconsistent replica
    - *but we always look at Q_R replicas*
  - – Why we can’t support incremental operations (insert, etc)

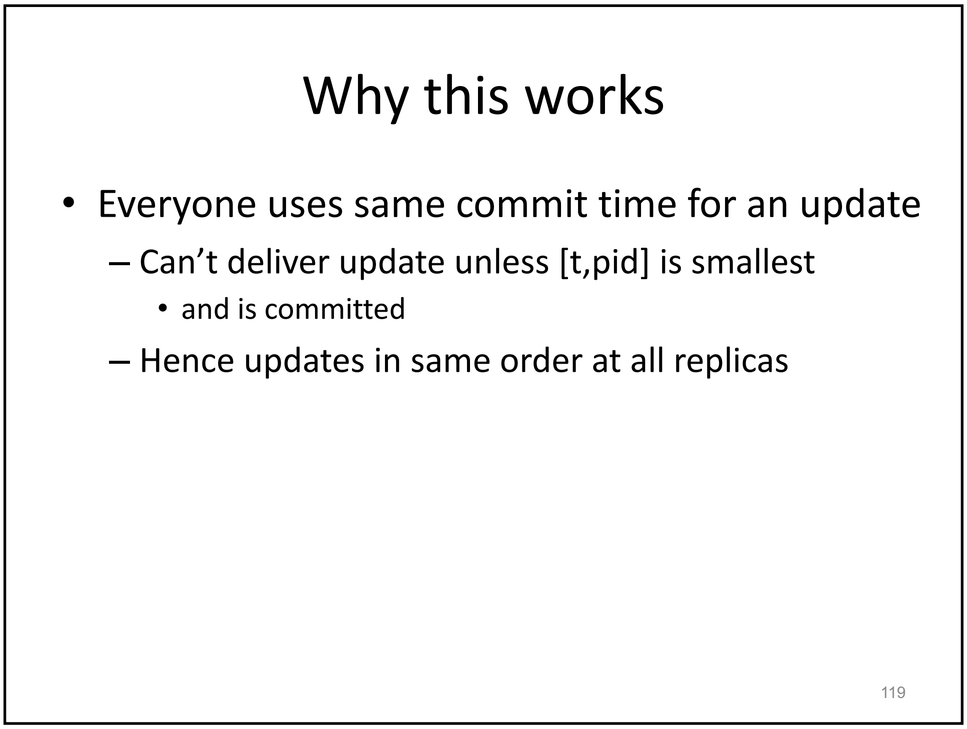
117



Why is this “safe”?

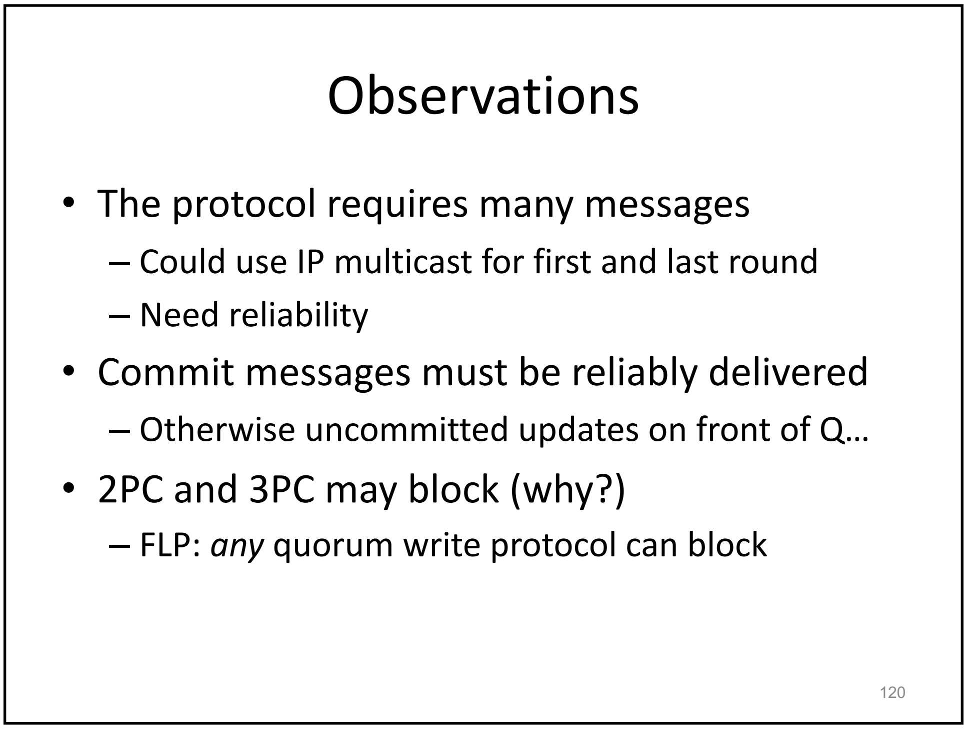
- Commit: moves pending update to front of Q
- Once a committed update reaches front of Q:
  - – …no update can be committed at earlier time!
- Any “future” update gets later time

118



Why this works

- Everyone uses same commit time for an update
  - Can’t deliver update unless [t,pid] is smallest
    - and is committed
  - Hence updates in same order at all replicas



Observations

- The protocol requires many messages
  - Could use IP multicast for first and last round
  - Need reliability
- Commit messages must be reliably delivered
  - Otherwise uncommitted updates on front of Q…
- 2PC and 3PC may block (why?)
  - FLP: *any* quorum write protocol can block