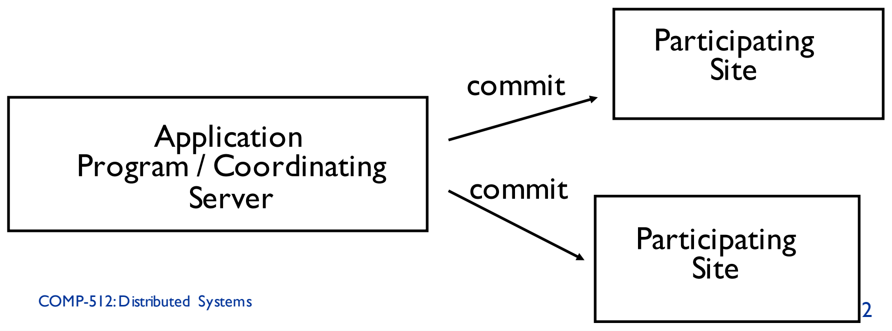
Atomic Commit Protocols

Distributed Commit

- ❑ A transaction is distributed across n processes.
- ❑ Each process can decide to commit or abort the transaction
- ❑ A transaction must commit on all sites or abort on all sites
- ❑ One-phase-commit:
  - ☆ Problems?



Consensus Problem: The classical agreement problem

- ❑ Model:
  - ☆ set of processes P_1, …. P_N
  - ☆ communication reliable but asynchronous
    - each message sent is eventually delivered to all correct recipients
      - ▲ (retransmission, network partitions eventually heal, ….)
  - ☆ processes only fail by crashing and then stop executing
    - correct process: exhibits no failures at any point in the execution under consideration
    - faulty process: opposite

Consensus II

- ❑ Problem Definition:
  - ☆ Start
    - every process P_i begins in undecided state and proposes a value v_i.
  - ☆ Protocol:
    - processes communicate with each other exchanging values.
  - ☆ End
    - Each process then sets a decision variable d_i and enters decided state, and does not change d_i anymore.
  - ☆ Conditions:
    - *Termination*: Eventually each correct process sets its decision variable
    - *Agreement*: The decision variable of all correct processes is the same: if p_i and p_j are correct and have entered the decided state, then $d_i=d_j$
    - *Integrity*: If the correct processes all proposed the same value, then every correct process in the decided state has chosen that value.
- ❑ Solution: non-existent [Fischer et al., 1985] even with one process crash failure

Atomic Commit in failure cases

- ❑ The impossibility result (consensus cannot be reached in an asynchronous system if processes sometime fail)
  - ☆ In general NO
- ❑ BUT: we will use a different failure model
  - ☆ Collection of processes p_i, i=1, … N
  - ☆ Communication reliable but asynchronous
  - ☆ Processes can crash
  - ☆ Processes recover from failure
  - ☆ Processes can log their state in stable storage
    - • Stable storage survives crashes and is accessible upon restart

Components of Atomic Commit

- ❑ Normal execution
  - ☆ The steps executed while no failures occur
- ❑ Termination protocol
  - ☆ When a site fails, the correct sites should still be able to decide on the outcome of pending transactions.
  - ☆ They run a termination protocol to decide on all pending transactions.
- ❑ Recovery
  - ☆ When a site fails and then restarts it has to perform recovery for all transactions that it has not yet committed appropriately
  - ☆ Single site recovery: abort all transactions that were active at the time of the failure
  - ☆ Distributed system: might have to ask around; maybe an active transaction was committed in the rest of the system
    - • Independent recovery: a site does not need to communicate with other sites at restart

Atomic Commitment

Properties to enforce:

- ❑ **Agreement**: All processors that reach a decision reach the same one.
- ❑ **Integrity**: Commit can only be decided if all processors vote YES (no imposed decisions).
- ❑ **Non-triviality**: If there are no failures and all processors voted YES, the decision will be to commit.
- ❑ **Termination**: Consider an execution with normal failures. If all failures are repaired and no more failures occur for sufficiently long, then all processors will eventually reach a decision.

2PC Protocol without failures

- ❑ Coordinator:
  - ☆ Start: Coordinator sends VOTE-REQ to all participants (also called PREPARE-TO-COMMIT; book notation is *canCommit?*).
  - ☆ Upon receiving votes from all participants
    - If all vote YES:
      - ▲ send COMMIT decision to all participants (book notation *doCommit*)
      - ▲ submit transaction commit locally (if coordinator also executed part of the txn)
    - If one or more vote NO:
      - ▲ send ABORT decision to all participants that voted YES (book notation *doAbort*)
      - ▲ submit transaction abort locally (if coordinator also executed part of the txn)
- ❑ Participant:
  - ☆ Upon receiving a VOTE-REQ?,
    - send a message with YES or NO
    - If vote is NO, submit transaction abort locally
  - ☆ Upon receiving decision (only if votes YES)
    - If decision is COMMIT, then submit transaction commit locally
    - Else submit transaction abort locally

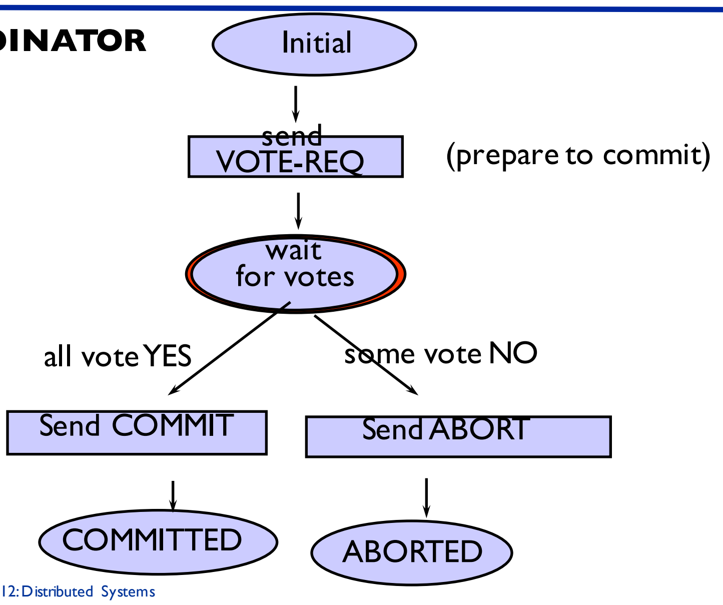
2PC Correctness

The protocol discussed meets the AC conditions

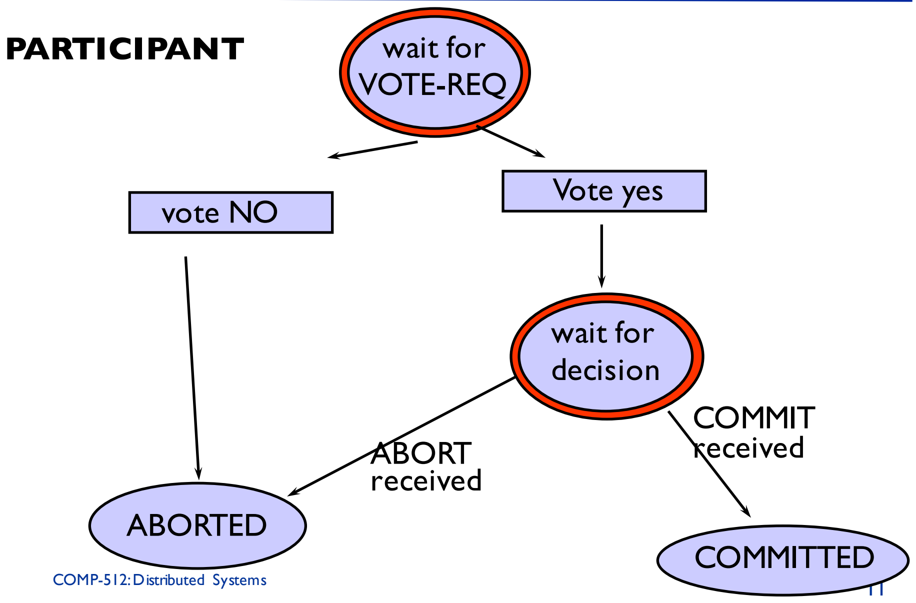
- ❑ Agreement:
  - ☆ every processor decides what the coordinator decides (if one decides to abort, the coordinator will decide to abort).
- ❑ Integrity:
  - ☆ the coordinator will decide to commit if all decide to commit (all vote YES).
- ❑ Non-triviality:
  - ☆ if there are no failures and everybody votes YES, the decision will be to commit.
- ❑ Termination:
  - ☆ the protocol needs to be extended in case of failures (in case of timeout, a site may need to "ask around").

Timeout Possibilities

COORDINATOR



Timeout Possibilities



Timeout Possibilities

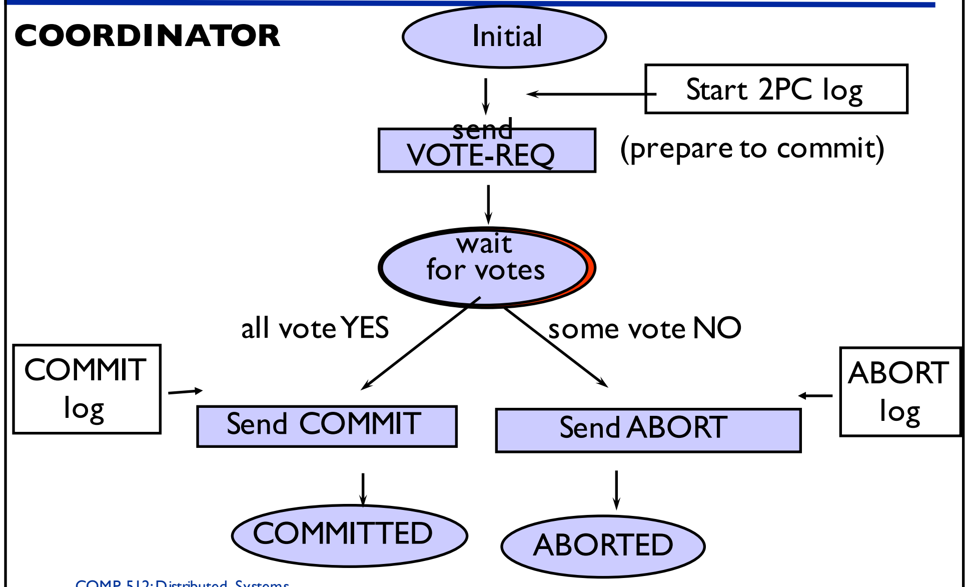
In those three waiting periods:

- Coordinator times out waiting for votes:
  - ☆ decide to abort (nobody has decided anything yet, or if they have, it has been to abort)
- Participant times out waiting for VOTE-REQ:
  - ☆ decide to abort (nobody has decided anything yet, or if they have, it has been to abort)
- Participant times out waiting for a decision:
  - ☆ cannot decide anything unilaterally
    - block or
    - run a Cooperative Termination Protocol.
  - ☆ A cooperative termination protocol can only be run, if a participant knows other participants
    - if at least one available participant has decision, commit/abort accordingly
    - if everybody is waiting, protocol blocks until coordinator is up again
      - ▲ thus protocol only correct because we assume that coordinator recovers!
  - ☆ Time from sending vote to getting decision is called uncertainty period
    - note that coordinator does not have uncertainty period

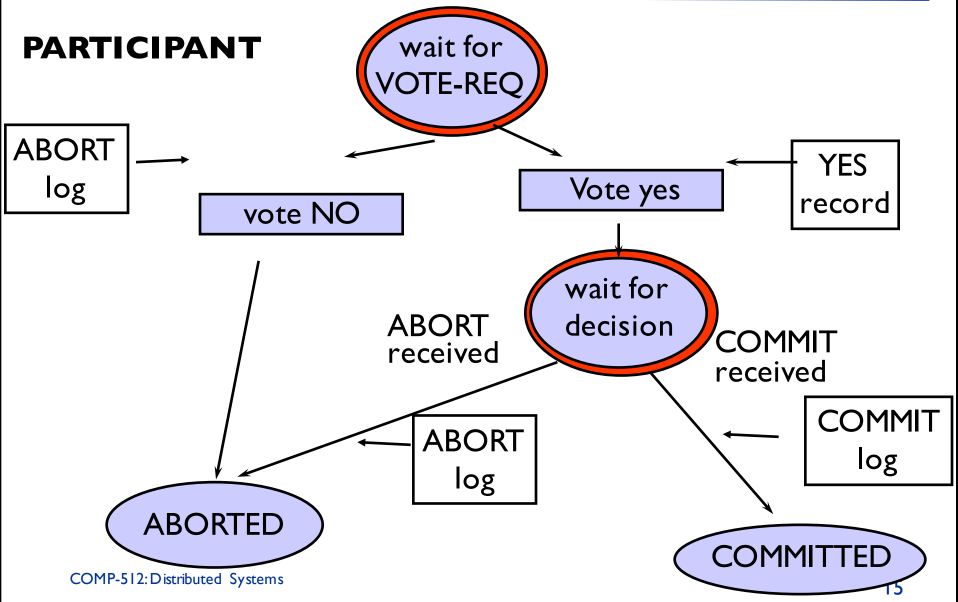
Recovery

- ❑ Processors must know their state to be able to tell others whether they have reached a decision
- ❑ The state must be persistent
  - ☆ must be available after failure and restart
- ❑ Persistence is achieved by writing a log record. This requires flushing the log buffer to disk (I/O).
  - ☆ This is done for every state change in the protocol.
  - ☆ This is done for every distributed transaction.
  - ☆ This is expensive!

Log Entries I



Log Entries II



Protocol with failures

(Parenthesis () mean that action is executed by different module)

```
Coordinator:
    Write start-2PC record in log
    Send VOTE-REQ to all participants
    Wait for vote messages (YES/NO) from all participants
      On timeout
        (abort transaction)
        Write ABORT record in log
        Send ABORT to all processes from which YES was received;
        Return;
    if all votes were YES and coordinator votes YES then
      (commit transaction)
      Write COMMIT record to log
      Send COMMIT decision to all participants
    Else
      (abort transaction)
      Write ABORT record in log
      Send ABORT decision to all processes from which YES was
      received
```

Protocol with failures

```
Participant:
    Wait for VOTE-REQ from coordinator
        On timeout
            (abort transaction)
            Write ABORT record in log
            Return;
    If participant votes NO
        (Abort transaction)
        Write ABORT record in log
        Send NO to coordinator
    Else /* participant votes no */
        (write undo/redo information in log)
        Write a YES record in log
        Send YES to coordinator
        Wait for decision message from coordinator
            On timeout (initiate termination protocol)
        If decision message is COMMIT,
            (commit transaction)
            write COMMIT record in log
        Else
            (abort transaction)
            write ABORT record in log
```

Comments

- ❑ Protocol does not cover all cases
  - ☆ (e.g., : Participant might abort before receiving VOTE-REQ)
- ❑ Coordination with local transaction manager:
  - ☆ before writing abort record:
    - undo transaction and write enough information to stable storage that undo is persistent
  - ☆ before writing YES record:
    - write enough information into stable storage so that changes of transaction are persistent AND the transaction can still be undone.
  - ☆ before writing COMMIT record:
    - coordinator:
      - ▲ write enough information into stable storage that changes are persistent
    - participant: ?

Restart Coordinator

- ❑ Scan forward through log
- ❑ For transaction T executing before coordinator fails
  - ☆ Does not find start 2PC record (but coordinator knows about transaction because it executed part of it…)
    - ● Possible states of participants:
      - ▲ timed-out or still waiting for vote request
    - ● Action:
      - ▲ Abort and tell participants (or participants will finally time out)
  - ☆ Finds start 2PC record but no commit/abort record
    - ● Possible states of participants:
      - ▲ timed-out waiting for vote request, waiting for vote request, or waiting for decision
    - ● Action:
      - ▲ send abort or resend vote request (in latter case, if coordinator has executed part of the transaction, then this only works if coordinator has enough information on stable storage itself to commit transactions)
  - ☆ Finds commit/abort record
    - ● Possible states of participants:
      - ▲ committed/aborted, waiting for decision
    - ● Actions:
      - ▲ Resend commit/abort record

Restart Participant

- ❑ Finds commit/abort record
  - ☆ Do nothing
- ❑ No yes/commit/abort record of transaction
  - ☆ Abort and (send vote-abort to coordinator)
- ❑ Finds yes record but no commit/abort
  - ☆ Ask the coordinator
  - ☆ No independent recovery possible!!

Garbage Collection

- ❑ At restart of coordinator, coordinator resends commit/abort decision of all transactions it has ever terminated
- ❑ Limit recovery overhead of coordinator
  - ☆ Must know which transactions are terminated at all sites
  - ☆ Solution during normal processing
    - Participants send ack to coordinator after abort/commit of transaction (notation in book: *haveCommitted)*
    - Once coordinator has acks from all participating sites it writes a END-OF-TRANSACTION record
  - ☆ Upon restart of coordinator
    - Finds commit/abort but no END-OF-TRANSACTION: resend decision
    - Finds END-OF-TRANSACTION: do nothing
  - ☆ On a regular basis: remove all records from txn with END-OF-TRANSACTION record

Guarantees

The impossibility result implies that there is always a chance to remain uncertain (unable to make a decision), hence:

- ❑ If failures may occur, then all entirely asynchronous commit protocols may block.
  - ☆ (will block for ever if failed site does not recover)
- ❑ No commit protocol can guarantee independent recovery (if a site fails when being uncertain, upon recovery it will have to find out from others what the decision was).

This is a very strong result with important implications in any distributed system.

Overhead

- ❑ Number of message rounds:
  - ☆ Determines the delay introduced by the protocol
  - ☆ 4 rounds (vote-request, vote, decision, ack)
  - ☆ Three of them are within transaction boundaries
- ❑ Number of messages with n+1 processes
  - ☆ Determines the bandwidth and processing power required by the protocol
  - ☆ Point-to-point:
    - n + n + n + (n)
  - ☆ Broadcast medium:
    - 1 + n + 1 + (n)
- ❑ Number of log rounds:
  - ☆ start 2PC, vote, commit/abort at coordinator, commit/abort at participant