Many of the slides through slide 21 are figures from Coulouris, Dollimore, Kindberg, and Blair.
Transactions

- “ACID” property:
  - atomic
    - all or nothing
  - consistent
    - take system from one consistent state to another
  - isolated
    - have no effect on other transactions until committed
  - durable
    - persists
**Coordination**

Coordinator

- Begin Transaction:
  - a.withdraw(100);
  - b.deposit(50);
  - c.deposit(50);
- End Transaction:

a. withdraw(100);

b. deposit(50);

c. deposit(50);
Atomic Commit

- AC1: All participants that reach a (commit/abort) decision reach the same one
- AC2: A participant cannot reverse its decision
- AC3: The commit decision can be reached only if all participants agree
- AC4: If there are no failures and all participants vote yes, then decision will be commit
- AC5: For any execution, if all failures are repaired and no new failures occur for a sufficiently long interval, then all participants will reach a (commit/abort) decision

This is adapted from Bernstein et al.
Two-Phase Commit

- **Phase 1**
  - coordinator prepares to commit:
    - asks participants to vote either “commit” or “abort”
    - participants respond appropriately
- **Phase 2**
  - coordinator decides outcome:
    - if all participants vote commit, outcome is commit, otherwise outcome is abort
    - outcome sent to all participants
  - participants do what they’re told
The labeling of the arcs (A/B) means that if “A” occurs, then perform action “B” and follow the arc to the next state.
Failures

- Coordinator or participants could crash
  - assume “fail-stop”
    - crash detected by time-out
    - no byzantine failures
  - crashed machines restart
    - recover their state
Dealing with Timeouts (1)

- Coordinator times out in *Wait* state
  - waiting for a participant to vote
  - takes no response to mean “abort”
  - sends abort to all other participants
- Participant times out in *Uncertain* state
  - waiting for coordinator to say “commit” or “abort”
  - can’t assume either outcome
  - waits for coordinator to restart
  - contacts coordinator for final outcome
This implies that the participants know one another’s identities. They could be supplied by the coordinator in the initial vote request.
Improving on Two-Phase Commit

- It works fine in practice!
- But …
  - all participants could conceivably be in uncertain state and coordinator is down (for a long time)
- Can we make it so such blocking can’t happen?
What Causes Blocking?

- Coordinator is down
- If all operational (not-failed) participants are in uncertain state, they are blocked
- If all participants are operational, they can elect new coordinator
- If any participant has crashed, the others don’t know if it crashed before or after voting (to commit or abort)
Guaranteeing Non-Blocking

- Non-blocking property (NB):
  - if any operational process is in the Uncertain state, then no process (operational or failed) can have decided to commit
- If NB holds, then operational processes may elect new coordinator and decide to commit or abort

Note that NB does not hold for two-phase commit!
For our upcoming discussion of three-phase commit, we assume that the only sort of failure is that of a machine crashing (then recovering). In particular, communication failures do not happen.
Three-Phase Commit

- Phase 1
  - coordinator prepares to commit:
    - asks participants to vote either “commit” or “abort”
    - participants respond appropriately
- Phase 2
  - coordinator counts votes:
    - if all participants vote commit, outcome is pre-commit, otherwise outcome is abort
    - outcome sent to all participants
  - participants ack and either abort or wait for commit
- Phase 3
  - coordinator waits for all acks
  - if committing, sends final commit to all participants
  - participants commit
Revised State Diagrams

Init -> app commit/vote req -> Wait
Wait -> any abort/abort
Abort
Abort -> all commit/precommit
Pre Commit
Commit

Init -> vote req/abort
Uncertain
Uncertain -> abort/ack
Commit
Commit
Commit
Commit

Abort

Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Commit
Comm
If a participant times out in its init state while waiting for a vote req from the coordinator, it may safely unilaterally abort.
If the coordinator times out in its wait state while waiting to receive votes from participants, it should send aborts to all operational participants.
If the coordinator times out while in its precommit state, waiting to receive acks from the participants, it may safely commit, since it had received commit votes from all. The failed participants will learn about the commit when they reboot.
If a participant times out in its uncertain state waiting to hear from the coordinator, it must communicate with the other operational participants to determine if it should commit or abort. In particular, if any other participant has aborted, it should abort. But what if this is not the case? (Go on to the next slides ...)
If a participant times out in its precommit state, waiting to hear from the coordinator, shouldn't it simply assume it may commit? The answer is no, because that might violate NB: there may be some other participant that's still in the uncertain state.

The situation we're concerned about is that, after committing, the participant might fail, while some other participant (perhaps the only other participant) remains operational, but in the uncertain state. That participant, now not knowing anything about the states of the others, should be allowed to abort by virtue of NB.
Details (1)

- If original coordinator remains operational
  - participant crashes handled as in two-phase commit

- If participant times out in Uncertain or PreCommit states
  - if any other participant has aborted, it aborts (it must have been in Uncertain state)
  - otherwise, it starts an election for a new coordinator
Note that the newly elected coordinator could fail. If so, a new one is elected. (Participants will time-out waiting for a message.)
Details (3)

- When failed participant comes back up
  - if it failed in \textit{Init} state
    - it aborts
  - Otherwise it asks other participants for outcome
    - will eventually get either commit or abort
      - (could get abort even if it was in the \textit{PreCommit} state when it crashed)
Correctness (1)

- Lemma 1: After a new coordinator starts up, exactly one of TR1 – TR4 will hold
- Theorem 1: In the absence of total failures, participants will never block
  - they clearly won't block if the coordinator never fails
  - if the coordinator fails, a new one is elected
  - one of TR1-TR4 will hold and a decision will be reached
  - if the new coordinator fails, a new one is elected; if it fails another is elected, etc. until there are no more participants

For details, see chapter 7 from Bernstein et al.
Correctness (2)

- Lemma 2: All participants that reach a decision on the same invocation of the termination protocol reach the same one
- Lemma 3: If NB holds before the termination protocol starts, it holds through the execution of the protocol
- Theorem 2: All operational participants reach the same decision
  – proof by induction on the invocations of the termination protocol
Total Failure

• What if coordinator and all participants fail?
• When they come back up, how do they decide?
  – if resurrected participant either didn’t vote or voted abort, it may unilaterally abort
  – otherwise, must run termination protocol
  – but works only if last participant to fail has come back up

See Bernstein et al. for details.
Communication Failures

- Network could partition into multiple pieces
- Not sufficient to get agreement in a piece containing a quorum
  - consensus is required for commit!
- Scenario
  - all participants vote
  - coordinator collects results
  - network partitions before or after all results collected
  - if network reconnects: easy
  - network never fully reconnects, but each participant eventually can communicate (perhaps briefly) with all others

See Bernstein et al. for details.