The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

Transactions

Database System Concepts, 6th Ed.

Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
- E.g., transaction to transfer $50 from account A to account B:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

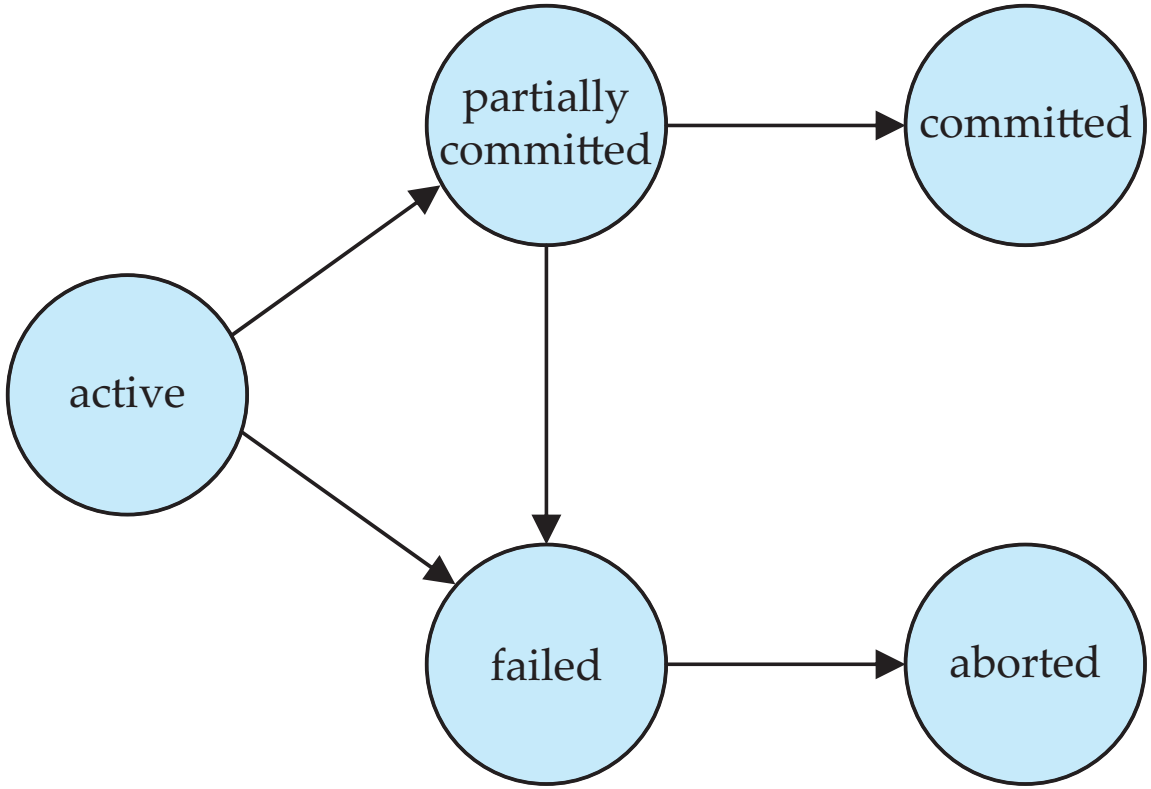
Required Properties of a Transaction (Cont.)

- **Consistency requirement** in above example:
  - The sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
    - Explicitly specified integrity constraints such as primary keys and foreign keys
    - Implicit integrity constraints
      - e.g., sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction, when starting to execute, must see a consistent database.
- During transaction execution the database may be temporarily inconsistent.
- When the transaction completes successfully the database must be consistent
  - Erroneous transaction logic can lead to inconsistency

Transaction State

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - Restart the transaction
    - can be done only if no internal logical error
  - Kill the transaction
- **Committed** – after successful completion.

Transaction State (Cont.)



Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - **Increased processor and disk utilization**, leading to better transaction *throughput*
    - E.g. one transaction can be using the CPU while another is reading from or writing to the disk
  - **Reduced average response time** for transactions: short transactions need not wait behind long ones.
- **Concurrency control schemes** – mechanisms to achieve isolation
  - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a **commit** instruction as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an **abort** instruction as the last statement

Schedule 1

- Let T_1 transfer \$50 from *A* to *B*, and T_2 transfer 10% of the balance from *A* to *B.*
- An example of a **serial** schedule in which T_1 is followed by T_2 :

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 2

- A **serial** schedule in which T_2 is followed by T_1 :

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Schedule 3

- Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is **equivalent** to Schedule 1.

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

Note -- In schedules 1, 2 and 3, the sum "A + B" is preserved.

Crash Recovery

Failure Classification

- Transaction failure :
  - Logical errors: transaction cannot complete due to some internal error condition
  - System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)
- System crash: a power failure or other hardware or software failure causes the system to crash. It is assumed that non-volatile storage contents are not corrupted.
- Disk failure: a head crash or similar failure destroys all or part of disk storage

Storage Structure

- Volatile storage:
  - does not survive system crashes
  - examples: main memory, cache memory
- Nonvolatile storage:
  - survives system crashes
  - examples: disk, tape
- Stable storage:
  - a mythical form of storage that survives all failures
  - approximated by maintaining multiple copies on distinct nonvolatile media

Data Access

- *Physical blocks* are those blocks residing on the disk. *Buffer blocks* are the blocks residing temporarily in main memory.
- Two operations:
  - **input**(*B*) transfers the physical block *B* to main memory.
  - **output**(*B*) transfers the buffer block *B* to the disk, and replaces the appropriate physical block there.
- Each transaction T_i has its private work-area in which local copies of all data items accessed and updated by it are kept. T_i's local copy of a data item *X* is called x_i.

Data Access (Cont.)

- Transaction transfers data items between system buffer blocks and its private work-area using the following operations :
  - **read**(X) assigns the value of data item X to the local variable x_i.
  - **write**(X) assigns the value of local variable x_i to data item {X} in the buffer block.
  - both these commands may necessitate the issue of an **input**(B_X) instruction before the assignment, if the block B_X in which X resides is not already in memory.
- Transactions perform **read**(X) while accessing X for the first time; all subsequent accesses are to the local copy. After last access, transaction executes **write**(X).
- **output**(B_X) need not immediately follow **write**(X). System can perform the **output** operation when it deems fit.

Fault-Tolerance: Log-Based Recovery

- A *log* is kept on stable storage. The log is a sequence of *log records*, and maintains a record of update activities on the database.
- When transaction T_i starts, it registers itself by writing a $<T_i$ **start**>log record
- *Before* T_i executes **write**(X), a log record $<T_i, X, V_1, V_2>$ is written, where V_1 is the value of X before the write, and V_2 is the value to be written to X.
- It means that T_i has performed a write on data item X_j. X_j had value V_1 before the write, and will have value V_2 after the write.
- When T_i finishes it last statement, the log record $<T_i$ **commi**t> is written.
- We assume for now that log records are written directly to stable storage (that is, they are not buffered)

Distributed Transactions

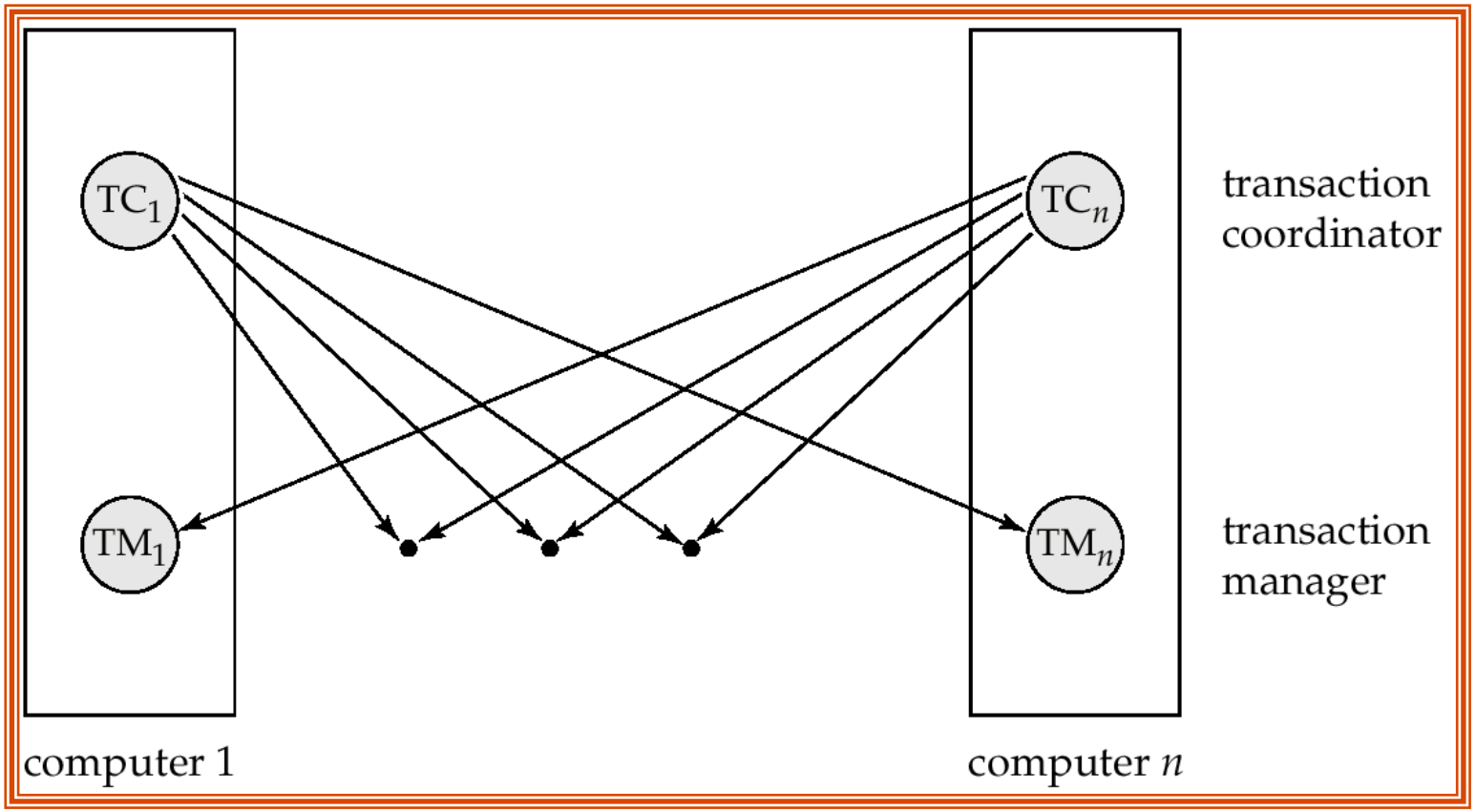
Distributed Database System

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites

Distributed Transactions

- Each site has a local transaction manager responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a transaction coordinator, which is responsible for:
  - Starting the execution of transactions that originate at the site.
  - Distributing subtransactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Transaction System Architecture



System Failure Modes

- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of messages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - **Network partition**
    - A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them
      - Note: a subsystem may consist of a single node
- Network partitioning and site failures are generally indistinguishable.

Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another
- The *two-phase commit* (2 *PC*) protocol is widely used
- The *three-phase commit* (3 *PC*) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol.

Two Phase Commit Protocol (2PC)

- Assumes **fail-stop** model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.
- The protocol involves all the local sites at which the transaction executed
- Let T be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

Phase 1: Obtaining a Decision

- Coordinator asks all participants to *prepare* to commit transaction T_i.
  - C_i adds the records <**prepare** *T*> to the log and forces log to stable storage
  - sends **prepare** *T* messages to all sites at which *T* executed
- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not, add a record <**no** *T*> to the log and send **abort** *T* message to C_i
  - if the transaction can be committed, then:
    - add the record <**ready** *T*> to the log
    - force *all records* for *T* to stable storage
    - send **ready** *T* message to C_i

Phase 2: Recording the Decision

- *T* can be committed if C_i received a **ready** *T* message from all the participating sites: otherwise *T* must be aborted.
- Coordinator adds a decision record, <**commit** *T*> or <a**bort** *T*>, to the log and forces record onto stable storage. Once the record is forced in stable storage it is irrevocable (even if failures occur)
- Coordinator sends a message to each participant informing it of the decision (commit or abort)
- Participants take appropriate action locally.

Handling of Failures - Site Failure

When a participating site S_k recovers from a failure, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contain <**commit** *T*> record: commit *T*
- Log contains <**abort** *T*> record: abort *T*
- Log contains <**ready** *T*> record: site must consult C_i to determine the fate of *T*.
  - If *T* committed, **commit** *T*
  - If *T* aborted, **abort** *T*
- The log contains no control records (abort, commit, ready) concerning *T.*
- S_k failed before responding to the **prepare** *T* message from C_i
  - since the failure of S_k precludes the sending of such a response, C_i must abort *T*
  - S_k aborts *T*

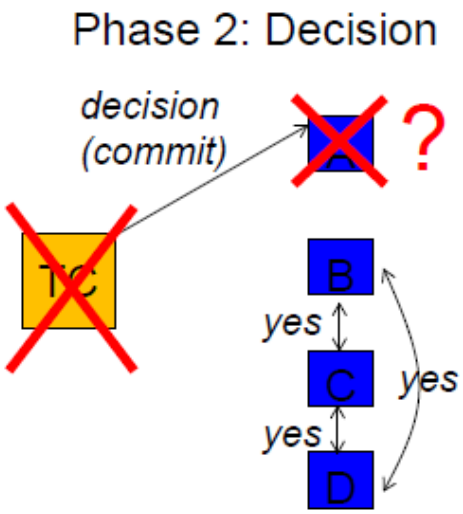
Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for T is executed then participating sites must decide on T's fate:
  1. If an active site contains a <**commit** T> record in its log, then T must be committed.
  2. If an active site contains an <**abort** T> record in its log, then T must be aborted.
  3. If some active participating site does not contain a <**ready** T> record in its log, then the failed coordinator C_i cannot have decided to commit T. Therefore abort T.
  4. If none of the above cases holds, then all active sites must have a <**ready** T> record in their logs, but no additional control records (such as <**abort** T> of <**commit** T>). In this case active sites must wait for C_i to recover, to find decision.
- **Blocking problem** : active sites may have to wait for failed coordinator to recover.

Two Phase Commit (2PC)
(slide by: Jinyang Li, "The Paper Trail")

Example Blocking Failure for 2PC

- Scenario:
  - TC sends commit decision to A, A gets it and commits, and then both TC and A crash
  - B, C, D, who voted Yes, now need to wait for TC or A to reappear (w/ mutexes locked)
    - They can't commit or abort, as they don't know what A responded
  - If that takes a long time (e.g., a human must replace hardware), then availability suffers
  - If TC is also participant, as it typically is, then this protocol is vulnerable to a single-node failure (the TC's failure)!



- This is why 2 phase commit is called a blocking protocol
- In context of consensus requirements: 2PC is safe, but not live

Failure of a Link

- When a link fails, all the messages that are in the process of being routed through the link do not arrive at their destination intact.
- From the viewpoint of the sites connected through that link, it appears that the other sites have failed.
- The previous scheme applies here.

Handling of Failures - Network Partition

- If the coordinator and all its participants remain in one partition, the failure has no effect on the commit protocol.
- If the coordinator and its participants belong to several partitions:
  - Sites that are not in the partition containing the coordinator think the coordinator has failed, and execute the protocol to deal with failure of the coordinator.
    - No harm results, but sites may still have to wait for decision from coordinator.
- The coordinator and the sites are in the same partition as the coordinator think that the sites in the other partition have failed, and follow the usual commit protocol.
    - Again, no harm results

Three Phase Commit (3PC)

- Assumptions:
  - No network partitioning and no link failures
  - At most K sites (participants as well as coordinator) can fail
  - At any point, at least one site must be up.
- Phase 1: Obtaining Preliminary Decision: Identical to 2PC Phase 1.
  - Every site is ready to commit if instructed to do so
- Phase 2 of 2PC is split into 2 phases, Phase 2 and Phase 3 of 3PC
  - In phase 2 coordinator makes a decision as in 2PC (called the pre-commit decision) and records it in multiple (at least K) sites
  - In phase 3, coordinator sends commit/abort message to all participating sites

Three Phase Commit (3PC)

Phase 2

- If C_i receives an abort T message from a participating site, or if C_i receives no response within a prespecified interval from a participating site, then C_i decides to abort T.
- The abort decision is implemented in the same way as in the 2PC protocol.
- If C_i receives a ready T message from every participating site, C_i makes the preliminary decision to precommit T.
  - T may still be aborted later.
- C_i adds a <pre-commit T> record to the log and forces it into stable storage.
- Then, C_i sends a precommit T message to all participating sites.
- When a site receives a message from the coordinator (either abort T or precommit T), it records it in its log, forces this information to stable storage, and sends a message ack T to C_i.

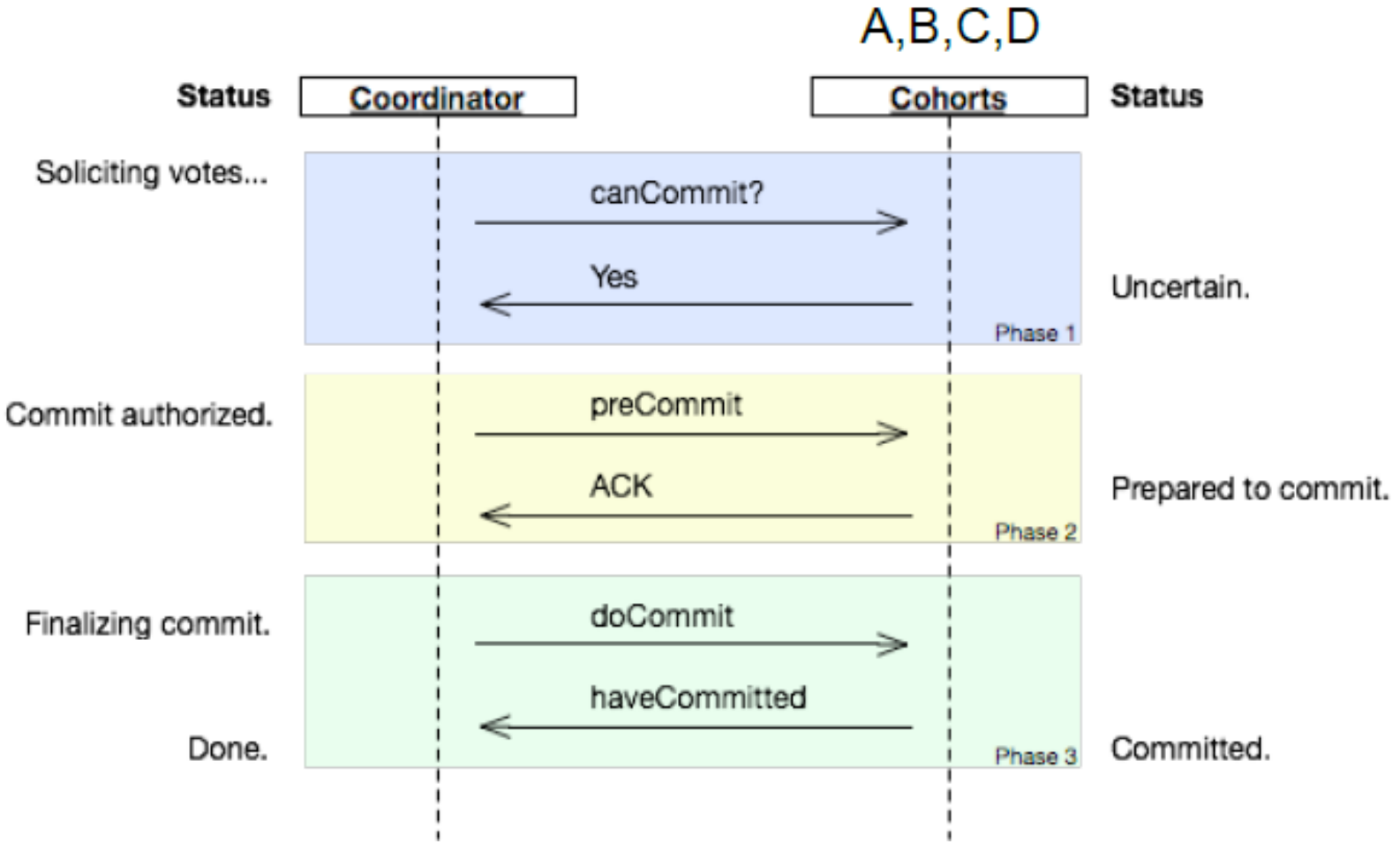
Three Phase Commit (3PC)

Phase 3

- This phase is executed only if the decision in phase 2 was to precommit.
- After <precommit T> are sent to all participating sites, C_i must wait until it receives at least K <ack T> messages.
- Then, C_i adds <commit T> to its log and forces it to stable storage, and
- C_i sends a <commit T> message to all participating sites.
- When a site receives that message, it records it in its log.

Three Phase Commit (3PC)

(slide by: Jinyang Li, "The Paper Trail")



Handling of Failures (3PC)

Failure of a participating site S_k

- S_k's log contains a <commit T> record ⇒ commit T
- S_k's log contains an <abort T> record ⇒ abort T
- S_k's log contains a <ready T> record but no <abort T> or <precommit T> ⇒ consult C_i to determine state of T
  - If C_i responds with <abort T> ⇒ abort T
  - If C_i responds with <commit T> ⇒ commit T
  - If C_i responds with <precommit T> ⇒ record this in log and resume protocol by sending an <ack T> to C_i
  - If C_i fails, execute coordinator failure protocol.

Failure of the coordinator

- When a participating site fails to receive a response from the coordinator, it executes the coordinator failure protocol.

3PC: Coordinator Failure Protocol

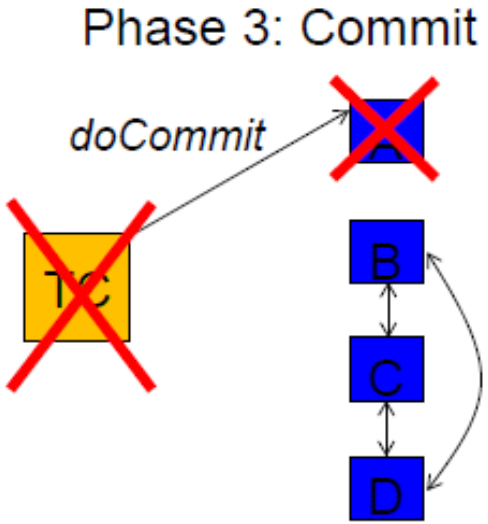
- The active participating sites select a new coordinator using an election protocol.
- The new coordinator, C_{new}, sends a message to each paticipating site requesting the local status of T.
- Each participating site, including C_{new}, determines the local status of T:
  - committed: the log contains a <commit T> record
  - aborted: the log contains an <abort T> record
  - ready: the log ctonais a <ready T> record but no <abort T> or <precommit T> record
  - precommited: the log contains a <precommit T> record but no <abort T> or <commit T>
  - not-ready: the log contains neither a <ready T> nor an <abort T> record.
- Each participating site sends its local status to C_{new}.

3PC: Coordinator Failure Protocol

- Depending upon the responses received, C_{new}, decides either to commit or abort T, or to restart a part of the three-phase commit protocol.
  - If at least one site has local status = committed, then commit T.
  - If at least one site has local status = aborted, then abort T.
  - If no site has local status = aborted, and no site has local status = committed, but at least one site has local status = precommitted, then C_{new} resumes the three-phase commit protocol by sending new precommit messages.
  - Otherwise, abort T.

3PC: Coordinator Failure Protocol
(slide by: Jinyang Li, "The Paper Trail")

- Assuming same scenario as before (TC, A crash), can B/C/D reach a safe decision when they time out?
  1. If one of them has received preCommit, …
  2. If none of them has received preCommit, …



Slide acks: