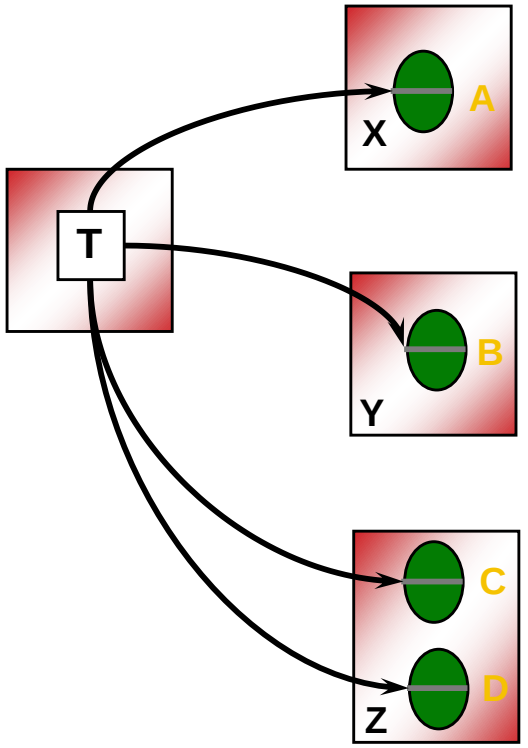
Commit Protocols

Edited slides of
Pallabh Dasgupta IITKgp
D Goswami IIT Guwahati

Distributed Transactions

- A transaction that invokes operations at several servers.



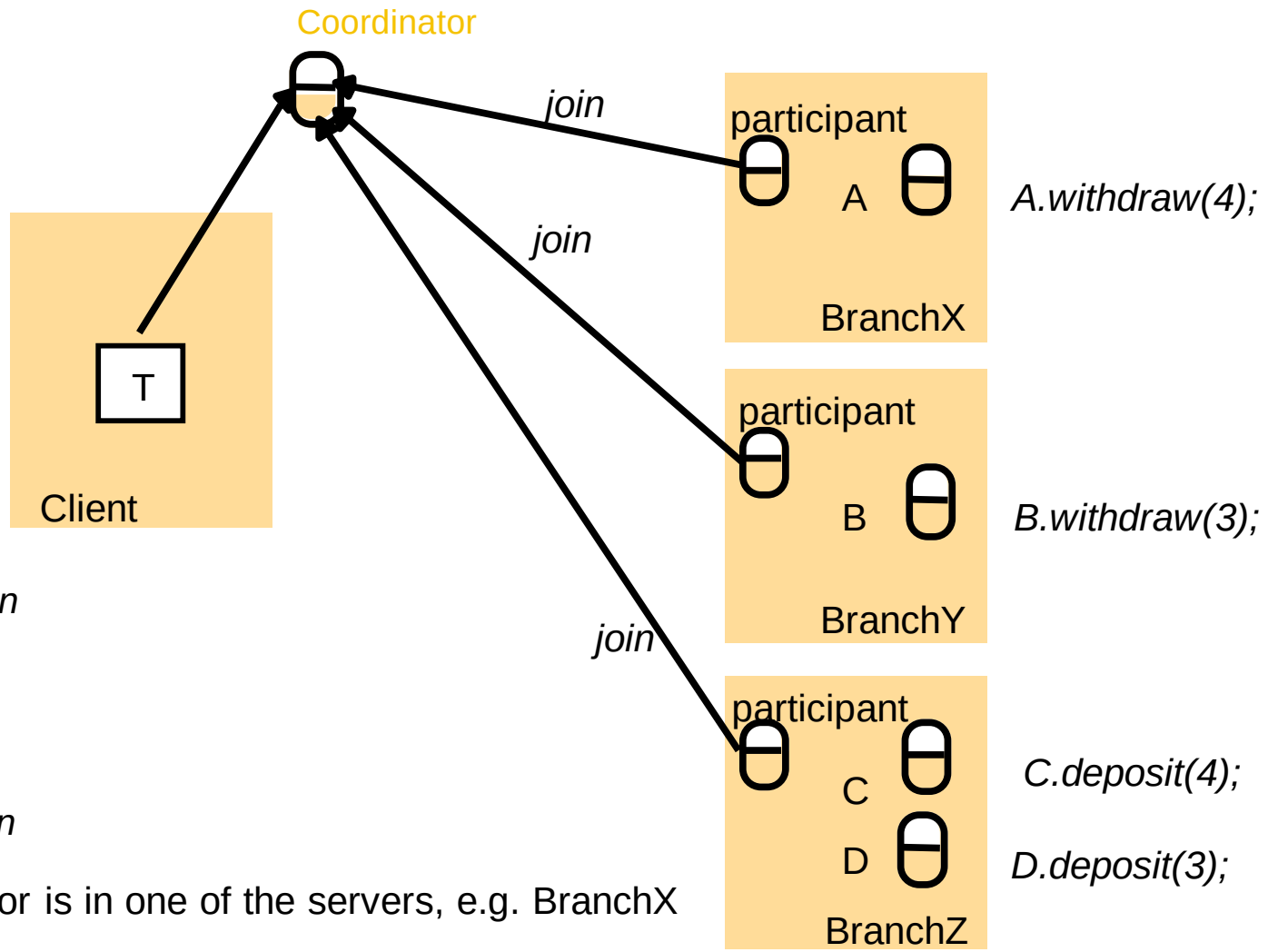
Coordinator of a Distributed Transaction

In a distributed environment, a coordinator is needed

Client sends an *openTransaction* to the coordinator

- **Other servers that manage the objects accessed by the transaction become *participants*.**

Distributed banking transaction



T = openTransaction
A.withdraw(4);
C.deposit(4);
B.withdraw(3);
D.deposit(3);
closeTransaction

Note: the coordinator is in one of the servers, e.g. BranchX

Transaction needs to follow the ACID properties

***Atomicity*—all or none—Transaction is an atomic unit. It is performed either to entirety or not performed at all.**

***Consistency*—don't violate DB integrity constraints: execution of the operation should be correct taking from one consistent state to another.**
Example: Amount cant be negative, total amount in the bank remains the same after within bank transfer, etc.

***Isolation* (Atomicity)—partial results are hidden. That is, execution of a transaction should not be interfered with by other transactions executing concurrently.**

***Durability*—effects (of transactions that "happened" or committed) are forever. They should not be lost due to failure**

What do we want to do?

Consider T1 and T2

T1
ADD(X,100)
REMOVE(Y,100)

T2
Get(X + Y)

Ensure isolation, atomicity, durability and consistency to these transactions given X and Y are on different locations.

In 2PC, each sub transaction locks the data before doing any work on it. The locks are released only after the transaction is completed.
This ensures isolation.

Can this lead to a deadlock?

What is an ABORT ?

Sub Transactions on certain sites may want to decide to fail(abort) in certain situations

Some Examples
- if in a deadlock, the process may need to abort to break the deadlock and release the resource
- Abort might be needed in case the transaction goes into an error like account does not exist or no money in account.
- Issues like divide by 0 situation encountered.
- Node failure so unable to decide.

Concurrency Control

A. Lock based which is a very careful approach.
However, might be slower as you wait for locks to be released but needed if too many conflicts

B. Don't bother about concurrent transactions. If you lucky then no conflicts. That saves you waiting on locks. If not lucky, then abort and retry.
If conflicts not frequent then this can be used.

Two Phase Commit Protocol (2PC)

- Acquires locks before accessing any record
- Lock released only after transaction is either committed or aborted
- This can land in deadlock situations
- Why do we need to keep the locks?

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.
- Site A commits as it completes its work but site B realizes there is an error so has to abort – violates atomicity

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

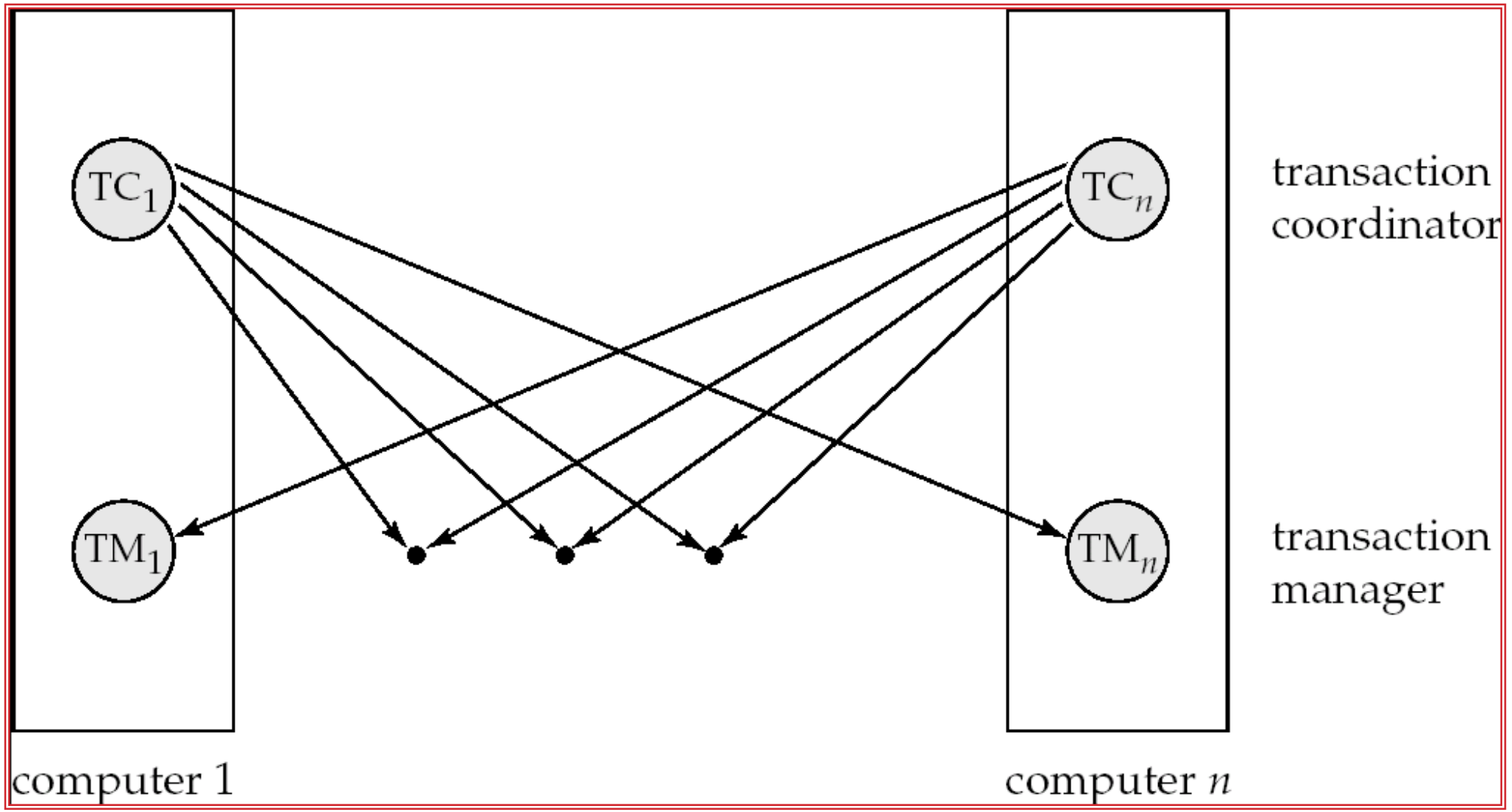
- The *two-phase commit* (2PC) protocol is widely used

- The *three-phase commit* (3PC) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol. This protocol is not used in practice.

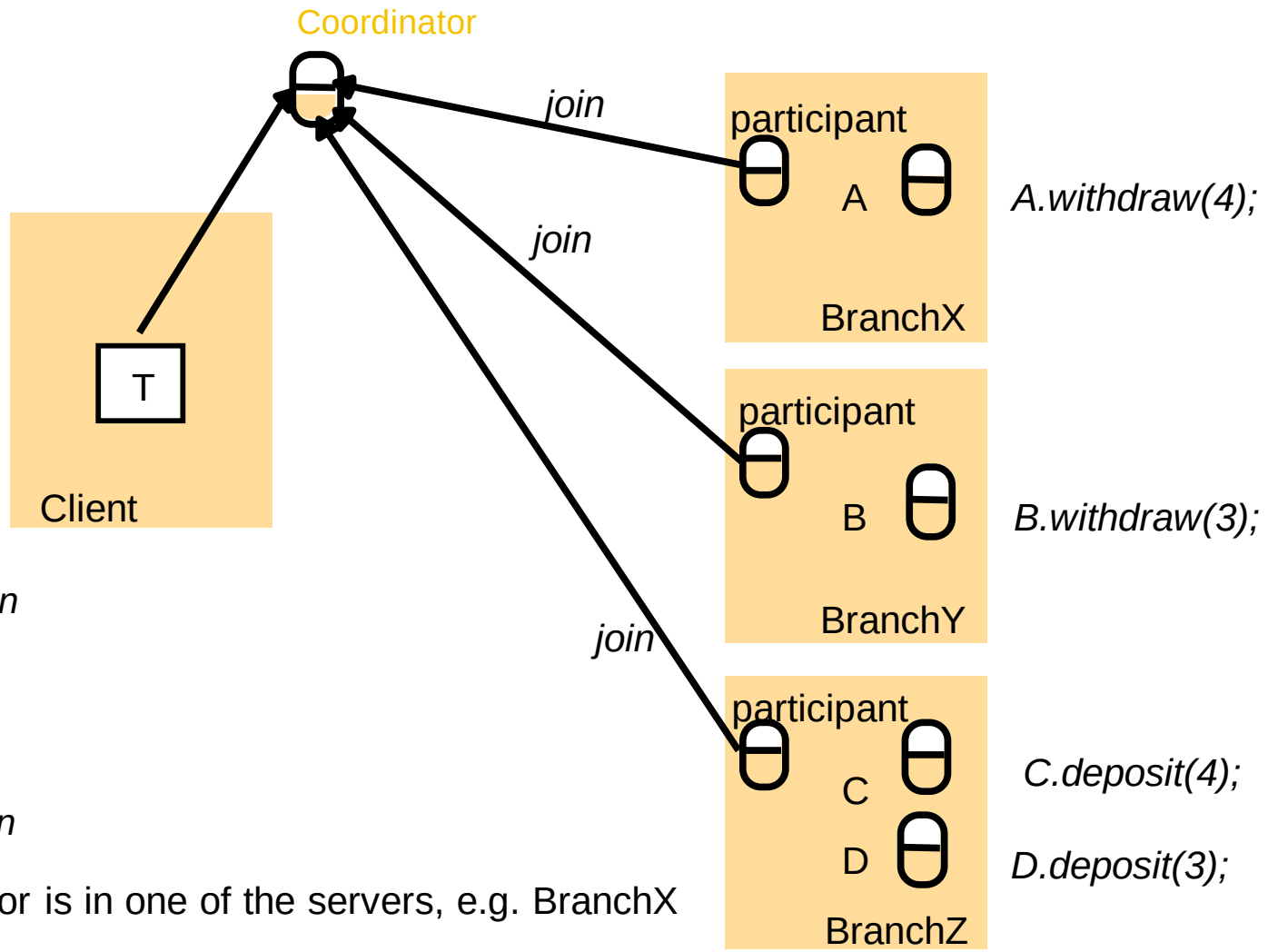
Distributed Transactions

- Transaction may access data at several sites.
- 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



Distributed banking transaction



T = openTransaction
A.withdraw(4);
C.deposit(4);
B.withdraw(3);
D.deposit(3);
closeTransaction

Note: the coordinator is in one of the servers, e.g. BranchX

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. They can come up later

- 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

** Hence prepare<T> and <no T> or <ready T> are now stored in log which is now in stable storage (database untouched)*

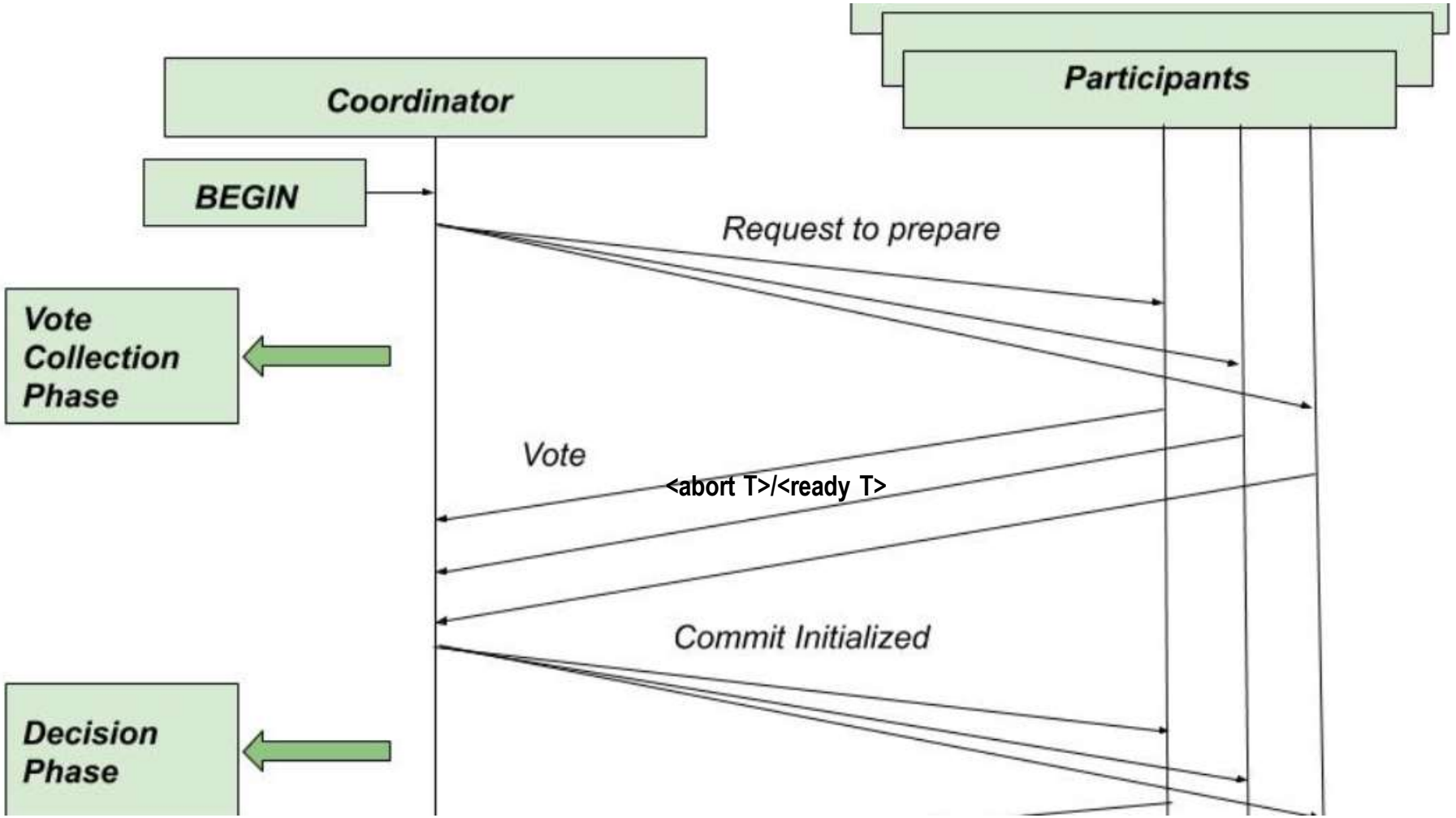
Phase 2: Recording the Decision

- ***T*** **can be committed of 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 <**abort ***T***>, to the log and forces record onto stable storage. Once the record 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.**


Coordinator
Participants
BEGIN
Request to prepare
Vote Collection Phase
Vote
<abort T>/<ready T>
Commit Initialized
Decision Phase

Handling of Failures - Site Failure

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- **Log contain <**commit *T*> **record: site executes** redo (*T*) ***copies from log to db***
- **Log contains <**abort *T*> **record: site executes** undo (*T*) ***remove from log***
- **Log contains <**ready *T*> **record: site must consult C_i to determine the fate of *T*.**
  - **If *T* committed, redo (*T*)**
  - **If *T* aborted, undo (*T*)**
- **The log contains no control records concerning *T* implies that 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_1 must abort *T***
  - **S_k must execute** undo (*T*)

Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for *T* is executing 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*. Can therefore abort *T*.
  4. If none of the above cases hold, 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.

     ** we don't know Ci's local decision as a participating site. Also we don't know if all ready T reached. If has not reached then Ci has not written in db so we don't need to commit.

- Blocking problem : active sites may have to wait for failed coordinator to recover (hence resources locked and held up). Again Coordinator failure very uncommon

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 which 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

Recovery and Concurrency Control

- **In-doubt transactions have a <ready *T*>, but neither a**

  <commit *T*>, nor an <abort *T*> log record.

- **The recovering site must determine the commit-abort status of such transactions by contacting other sites; this can slow and potentially block recovery.**
- **Recovery algorithms can note lock information in the log.**
  - **Instead of <ready *T*>, write out <ready *T*, *L*> *L* = list of locks held by *T* when the log is written (read locks can be omitted).**
  - **For every in-doubt transaction *T*, all the locks noted in the**

    <ready *T*, *L*> log record are reacquired.
- **After lock reacquisition, transaction processing can resume; the commit or rollback of in-doubt transactions is performed concurrently with the execution of new transactions.**

Three Phase Commit (3PC)

- Assumptions:
  - No network partitioning
  - At any point, at least one site must be up.
  - At most K sites (participants as well as coordinator) can fail

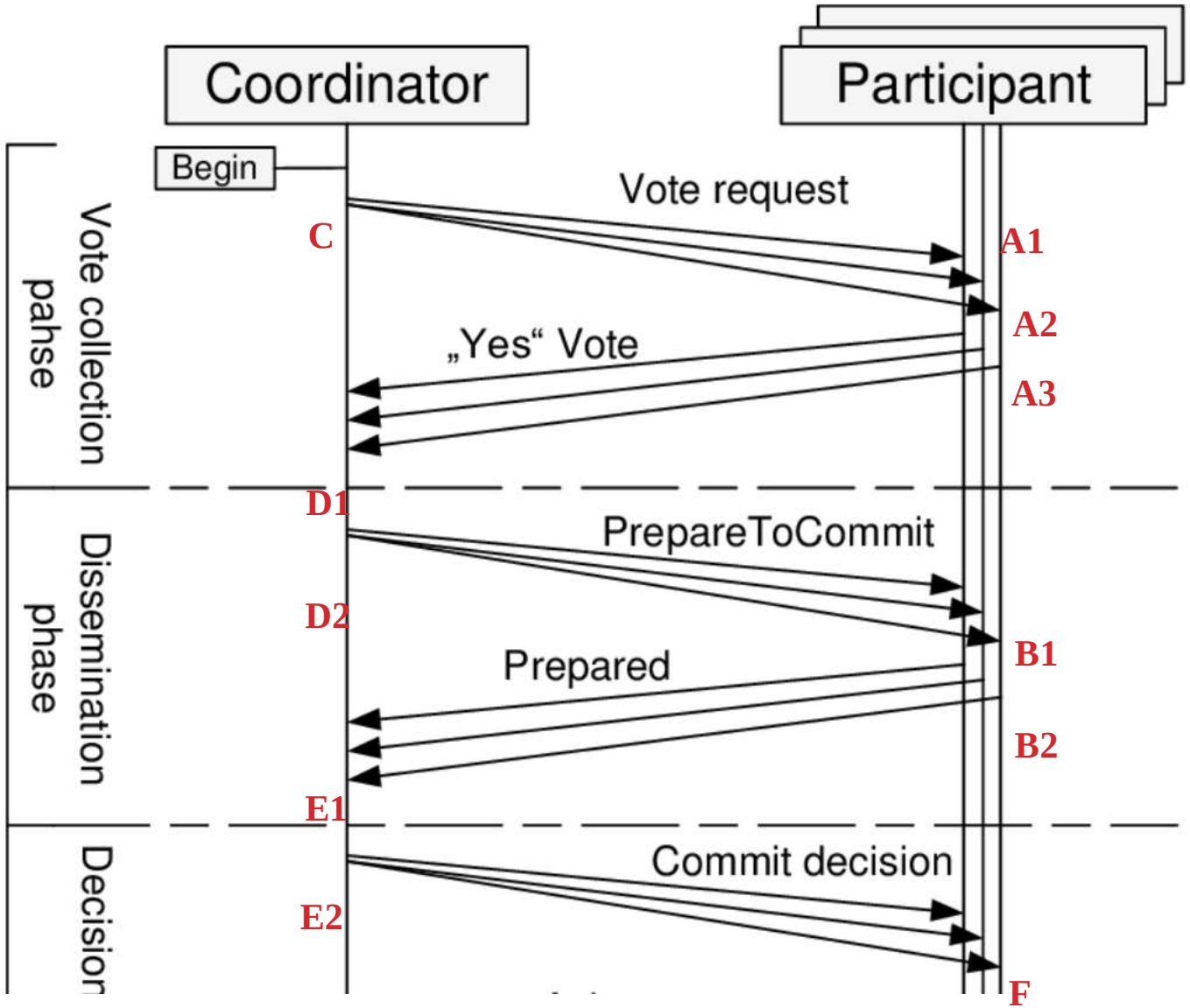
- Phase 1: Obtaining Preliminary Decision: Identical to 2PC Phase 1.
  - Every site is ready to commit if instructed to do so

Three Phase Commit (3PC)

- 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 sends pre-commit msg.
    When he receives at least k replies (ack) then he starts sending commit to those sites. As he receives more acks he keeps sending commits. Hence its decision to commit is recorded in multiple (at least K) sites before final commit goes out.
  - In phase 3, coordinator sends commit/abort message to all participating sites,

- Under 3PC, knowledge of pre-commit decision can be used to commit despite coordinator failure
  - Avoids blocking problem as long as upto K sites fail. If beyond k fail then blocking could happen

- Drawbacks:
  - higher overheads
  - assumptions may not be satisfied in practice


Coordinator
Participant
Begin
Vote request
C
A1
A2
„Yes“ Vote
A3
Vote collection pahse
D1
PrepareToCommit
D2
B1
Prepared
B2
Dissemination phase
E1
Commit decision
E2
F
Decision

Handling of Failures - Site Failure

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- **F:Log contain <**commit *T*> **record: site executes** redo (*T*) ***copies from log to db***
- **F: Log contains <**abort *T*> **record: site executes** undo (*T*) ***remove from log***
- **A3: Log contains <**ready *T*> **record: site must consult C_i to determine the fate of *T*.**
  - **If *T* committed, redo (*T*)**
  - **If *T* aborted, undo (*T*)**
- **A1 A2: (includes (a) After <prepare T> before <ready t> and (b) Before <prepare T>)**

  The log contains no control records concerning *T* replies that 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_1 must abort *T***
  - **S_k must execute** undo (*T*)

Handling of Failures - Site Failure

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- **B1:Log contain <pre-commit *T*> record: site needs to check with coordinator since on getting k acks the coordinator could have committed.**

  The transaction can also be in an abort situation if the coordinator failed after sending precommit to site Si(Si also had failed). So remaining sites seeing no precommit msgs among them would abort.

- **B2: Log contains <ack *T*> record: same as B1**

Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for *T* is executing then participating sites must decide on *T*'s fate:

  1. E2:If an active site contains a <commit *T*> record in its log, then *T* must be committed.
  2. E2:If an active site contains an <abort *T*> record in its log, then *T* must be aborted.
  3. C: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*. Can therefore abort *T*.
  4. all active sites have a <ready *T*> record in their logs, but no additional control records (such as <abort *T*> of <commit *T*>).

     D2,E1: If there is <precommit T> in someone's log then vote for new coordinator. New coordinator behaves as if it received <ready T> from everyone. It sends <precommit T> to make sure k sites have the info.

     D1: Abort

     .