Databases 2

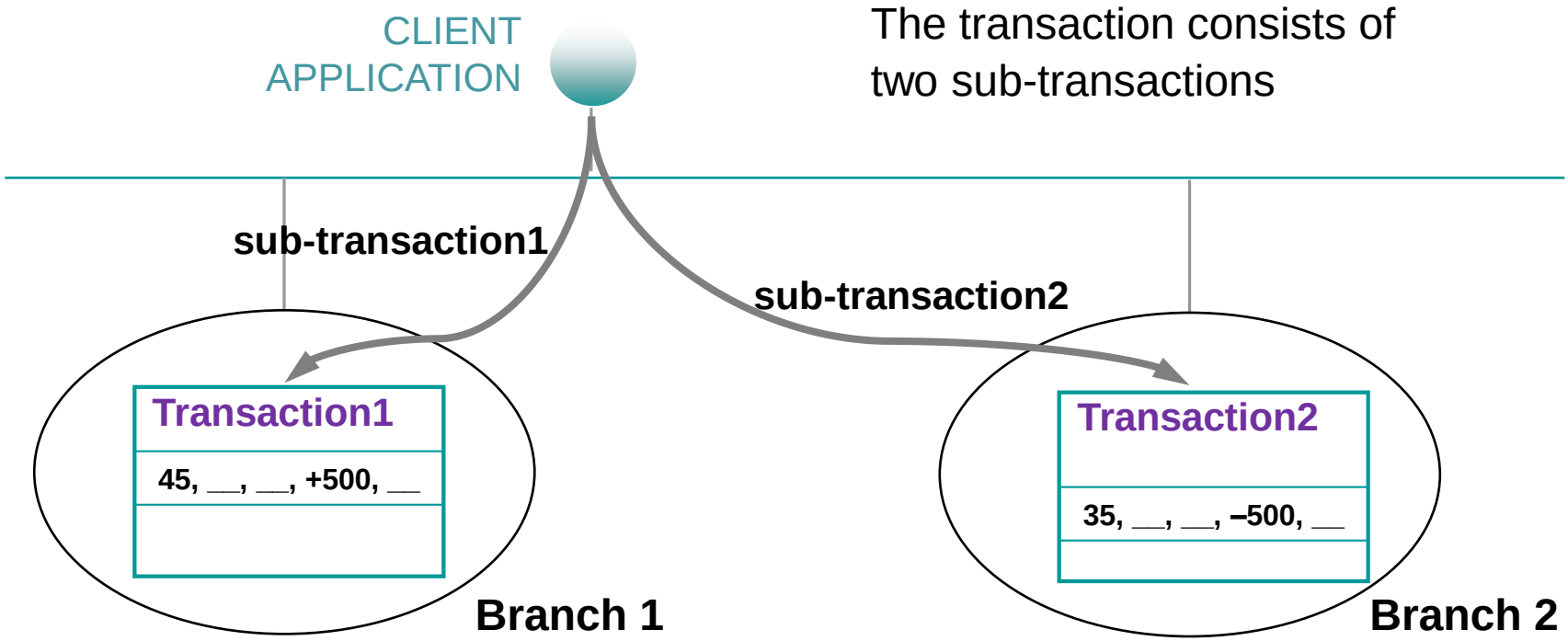
5

Distributed Commit and Recovery Protocols

Distributed Transactions

```
begin transaction;
  update Account1@Branch1
    set Balance = Balance + 500
      where AccNum = 45;
  update Account2@Branch2
    set Balance = Balance - 500
      where AccNum = 35;
commit-work;
end transaction;
```

Distributed Transactions



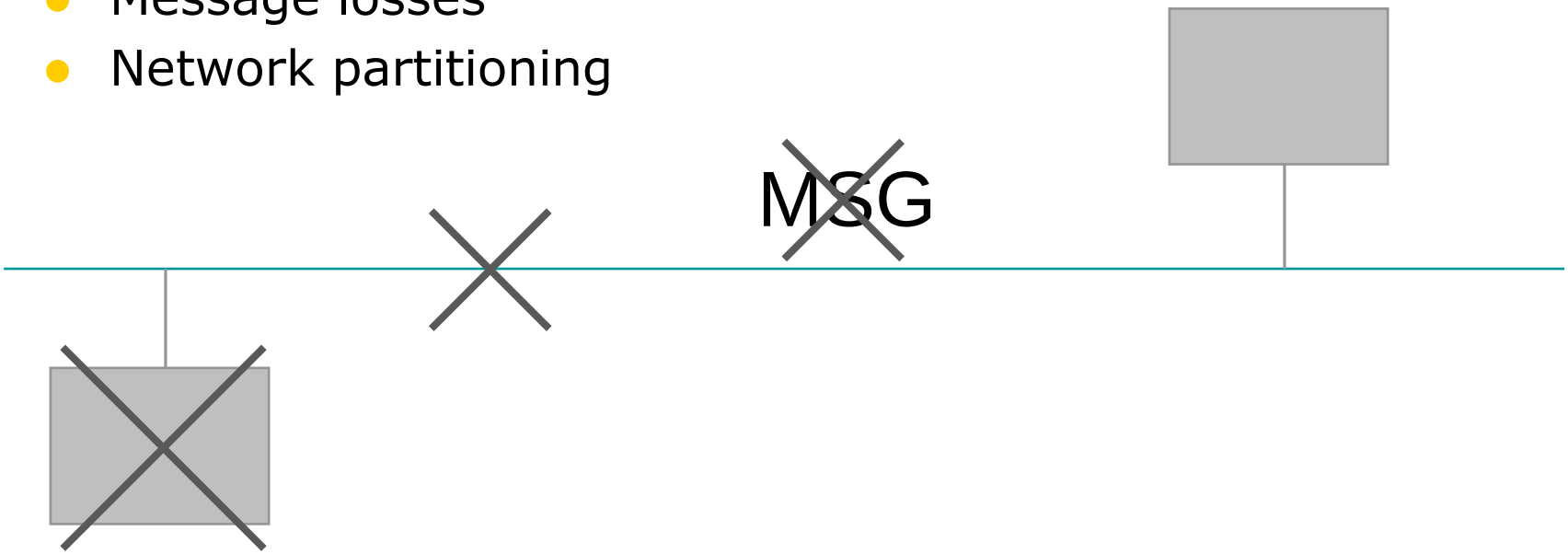
ACID Properties of Distributed Execution

- Isolation
  - If each sub-transaction is 2PL, then the transaction is globally serializable
- Durability
  - If each sub-transaction handles logs correctly, data are globally persistent
- Consistency
  - If each sub-transaction preserves local integrity, data are globally consistent
- **Atomicity**
  - It is the **main problem** of distributed transactions

5 Commit Protocols

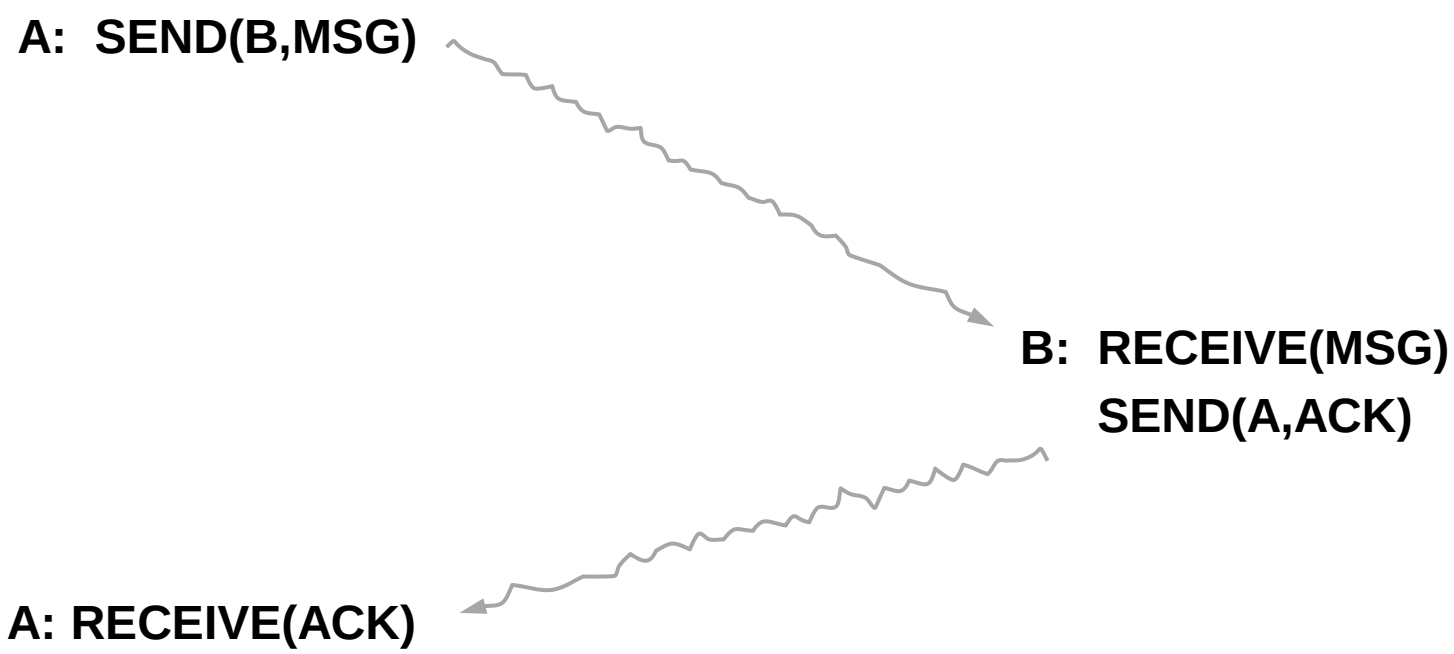
Faults in a Distributed System

- Node failures
- Message losses
- Network partitioning



5 Commit Protocols

Distributed Protocols and Message Losses



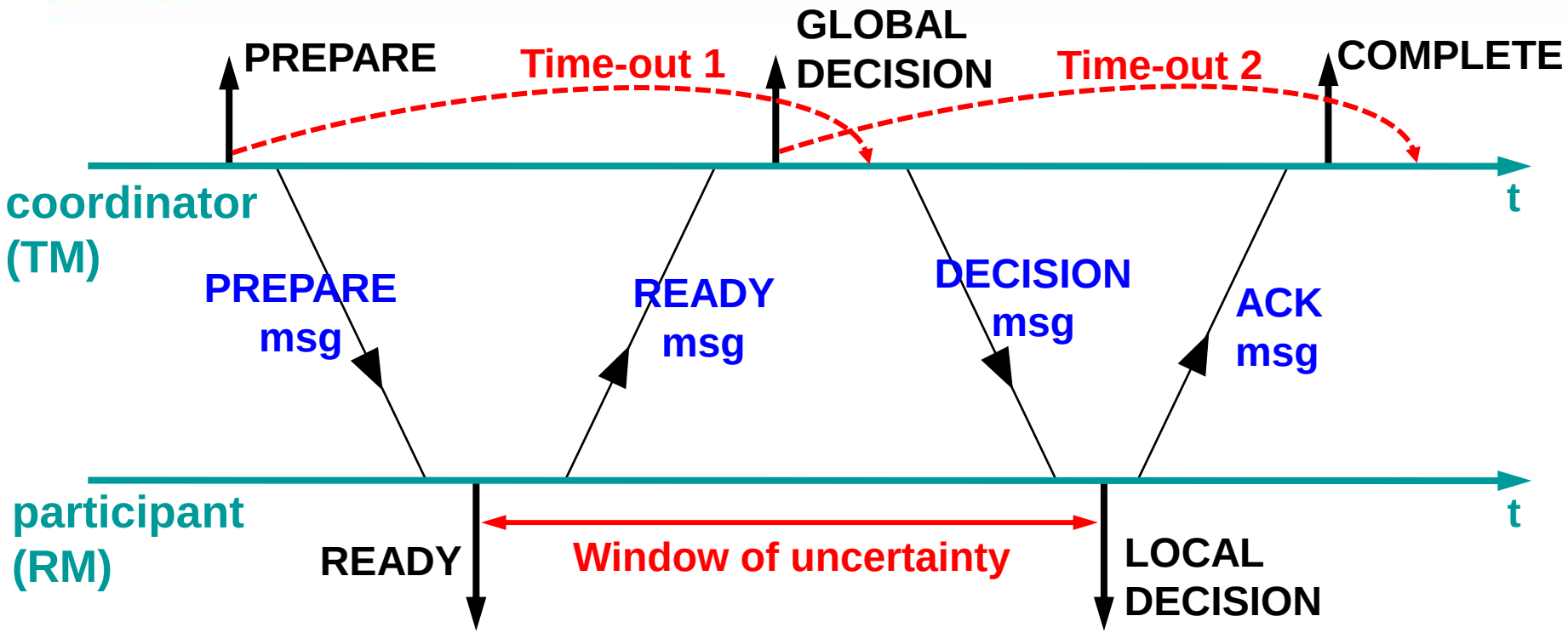
Two-phase Commit (2PC) Protocol

- Protocol that guarantees atomicity of distributed sub-transactions
- Protagonists:
  - A coordinator (Transaction Manager, TM)
  - Several participants (Resource Manager, RM)
- Similar to a marriage
  - Phase one: the decision is declared
  - Phase two: the marriage is ratified

New Log Records

- In the coordinator's log
  - **`prepare`**: participants' identity
  - **`global_commit/abort`**: decision
  - **`complete`**: end of protocol
- In the participant's log
  - **`ready`**: willingness/availability to participate to the commit
  - **`local_commit/abort`**: decision received

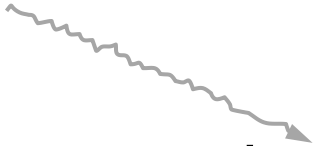
Diagram of the Two-phase Commit Protocol



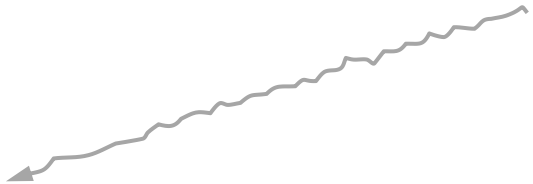
5 Commit Protocols

Phase 1

```
C: WRITE-LOG(PREPARE)
   SET TIME-OUT
   SEND(Pi,PREPARE)
```



```
Pi: RECEIVE(C,PREPARE)
    IF OK THEN WRITE-LOG(READY)
                  READY=YES
           ELSE READY=NO
    SEND(C,READY)
```



5 Commit Protocols

Phase 2

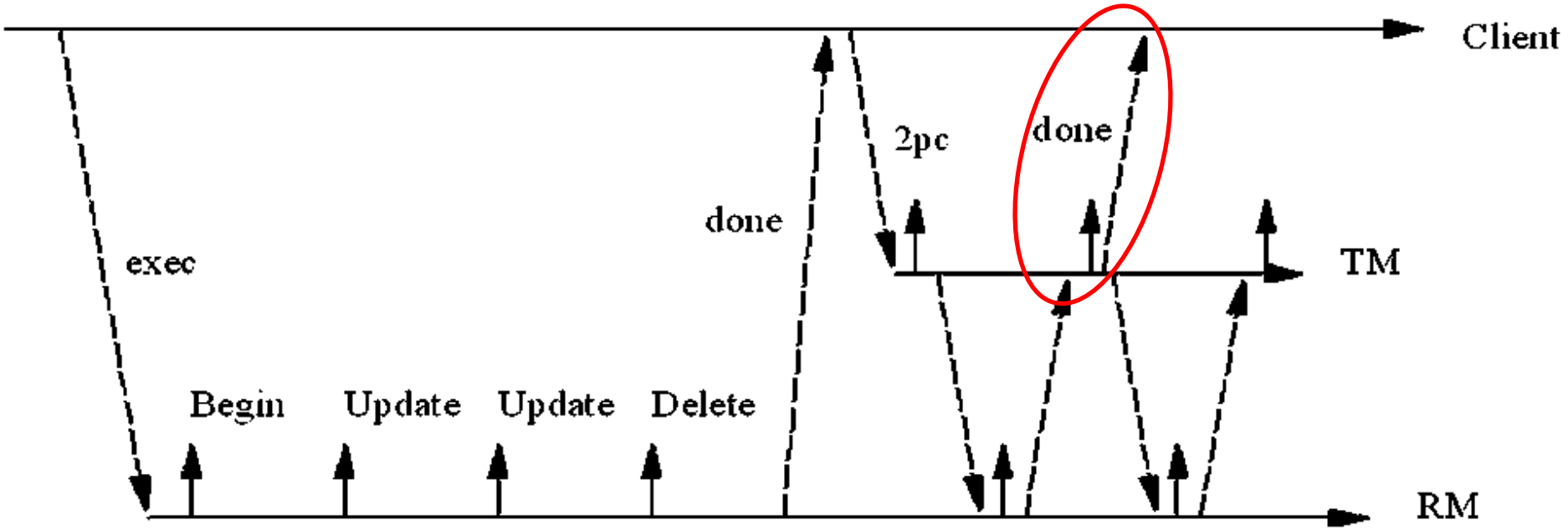
```
C:  RECEIVE(Ci,MSG)
    IF TIME-OUT OR ANY(MSG)==NO
     THEN WRITE-LOG(GLOBAL-ABORT)
          DECISION=ABORT
     ELSE WRITE-LOG(GLOBAL-COMMIT)
          DECISION=COMMIT
    SEND(Pi,DECISION)
```

```
Pi: RECEIVE(C,DECISION)
    IF DECISION==COMMIT
     THEN WRITE-LOG(COMMIT)
     ELSE WRITE-LOG(ABORT)
    SEND(C,ACK)
```

```
C:  RECEIVE(Pi,ACK)
    WRITE-LOG(COMPLETE)
```

5 Commit Protocols

Protocol in the Context of a Complete Transaction



Complexity of the Protocol

- Must be able to handle all possible failures:
  - Failure of the coordinator
  - Failure of one or more participants
  - Message losses

Blocking, Uncertainty, Recovery Protocols

- A RM in a "ready" state loses its autonomy and awaits the decision of the TM (it is *blocked*). A failure of the TM leaves the RM in an uncertain state. The resources acquired by using locks are blocked
- The interval between the writing on the RM's log of the "ready" record and the writing of the **`commit`** or **`abort`** record is called the *window of uncertainty*. The protocol is designed to keep this interval to a minimum
- Recovery protocols are performed by the TM or RM after failures; they recover a correct final state which depends on the global decision of the TM

Recovery of Participants

- Performed by the warm restart protocol
  Depends on the **last** record written in the log:
  - If it is an action or **`abort`** record, the actions are *undone*;
    if it is a **`commit`** record, the actions are *redone*;
    In these cases, recovery is independent of the 2PC protocol
  - If it is a **`ready`** record, then the failure occurred during the two-phase commit (*window of uncertainty*). In this case the participant is ***in doubt*** about the result of the transaction
- During the warm restart procedure, the identifiers of the transactions in doubt are collected (new set). For each of them the final transaction outcome must be asked to the TM (*remote* recovery request)

Recovery of the Coordinator

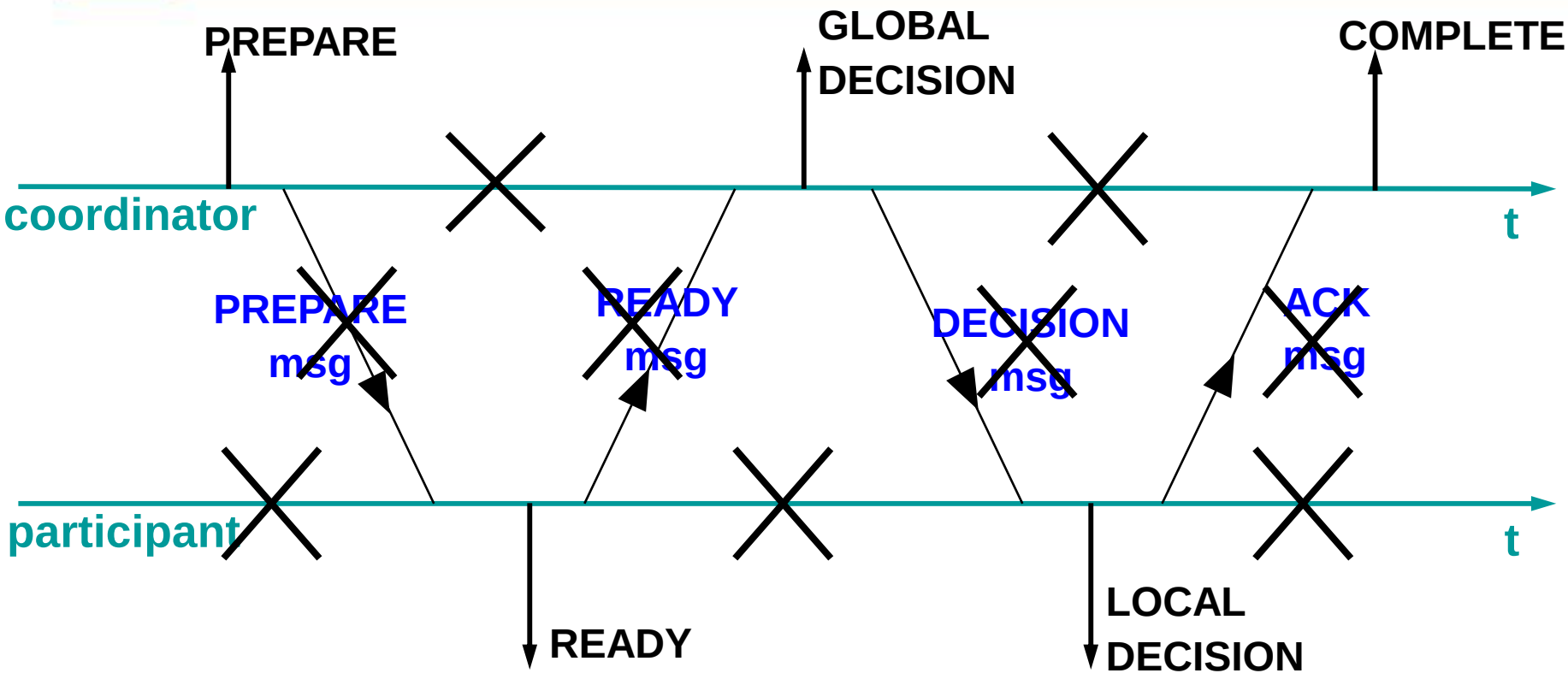
- When the last record in the log is a **`prepare`**, the failure of the TM might have placed some RMs in a blocked situation. Two recovery options:
  - Write **`global abort`** on the log, and then carry out the second phase of the two-phase commit protocol
  - Repeat the first phase, trying to reach a **`global commit`**
- When the last record is a **`global-<decision>`**, some RMs may have been left in a blocked state (decision taken, but not communicated yet). The TM must then repeat the second phase of the protocol

Message Loss and Network Partitioning

- The loss of a **`prepare`** or **`ready`** message are not distinguishable by the TM. In both cases, the TM reaches time-out and a **`global-abort`** decision is made
- The loss of a **`decision`** or **`ack`** message are also indistinguishable. In both cases, the TM reaches time-out and the second phase is repeated
- A *network partitioning* does not cause further problems: **`global-commit`** is reached only if the TM and all the RMs belong to the same partition

5 Commit Protocols

Recovery of the 2PC Protocol



Presumed Abort Protocol

- An optimization used by most DBMSs
  - If a TM receives a "remote recovery" request from an in-doubt RM and it does not know the outcome of that transaction, the TM returns a **`global-abort`** decision as default
- As a consequence, if **`prepare`** and **`global-abort`** are lost, the behavior is anyhow correct => it is not necessary to write them synchronously (force) onto the log
- Furthermore, the **`complete`** record can be omitted
- In conclusion, the only records to be forced are **`ready`**, **`global-commit`** and **`local-commit`**

Read-only Optimization

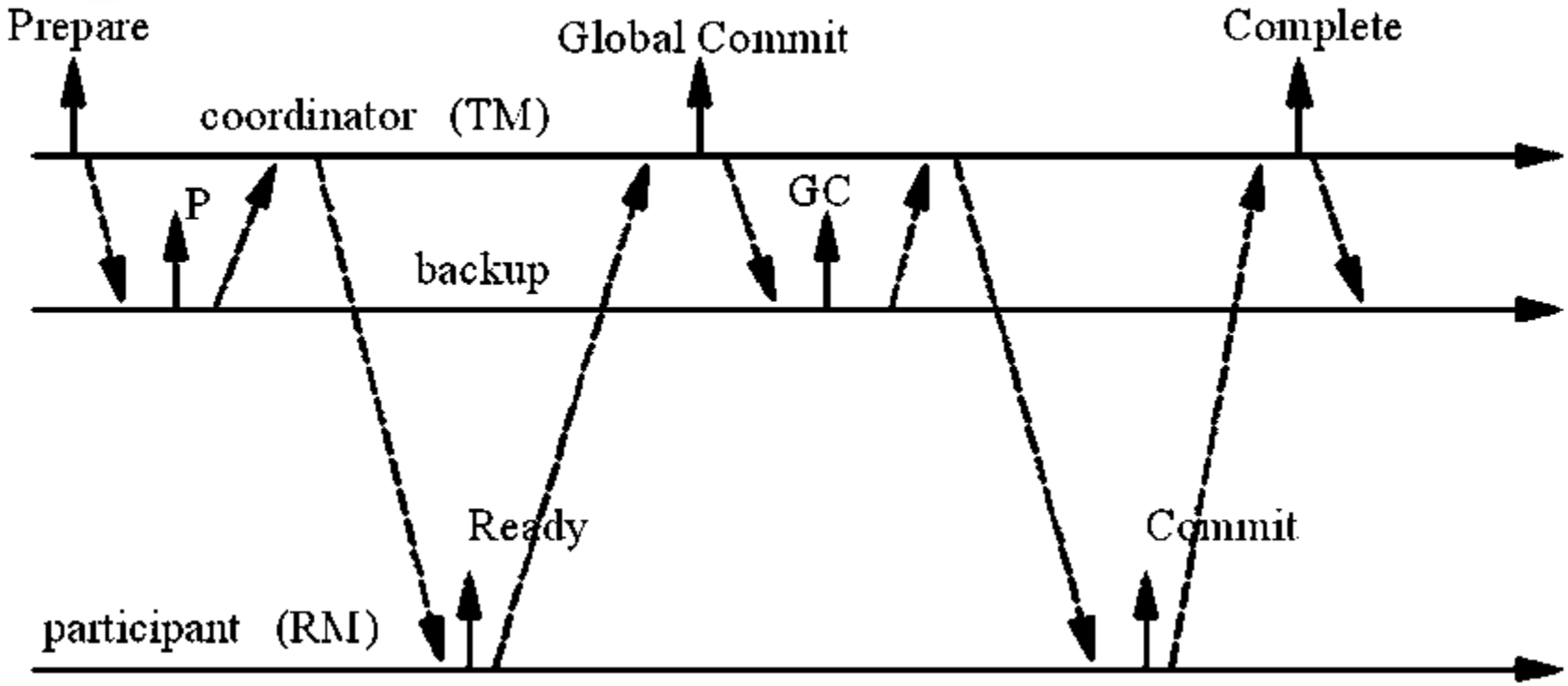
- When a participant is found to have carried out only read operations (no write operations)
  - It responds **`read-only`** to the **`prepare`** message and suspends the execution of the protocol
  - The TM ignores all read-only RMs in the second phase of the protocol

Four-phase Commit Protocol

- The TM process is replicated by a backup process, located on a different node.
  - The TM first informs the backup of its decisions and then communicates with the RMs
- The backup can replace the TM in case of failure
  - When a backup becomes TM, it first activates another backup
  - Then, it continues the execution of the commit protocol

5 Commit Protocols

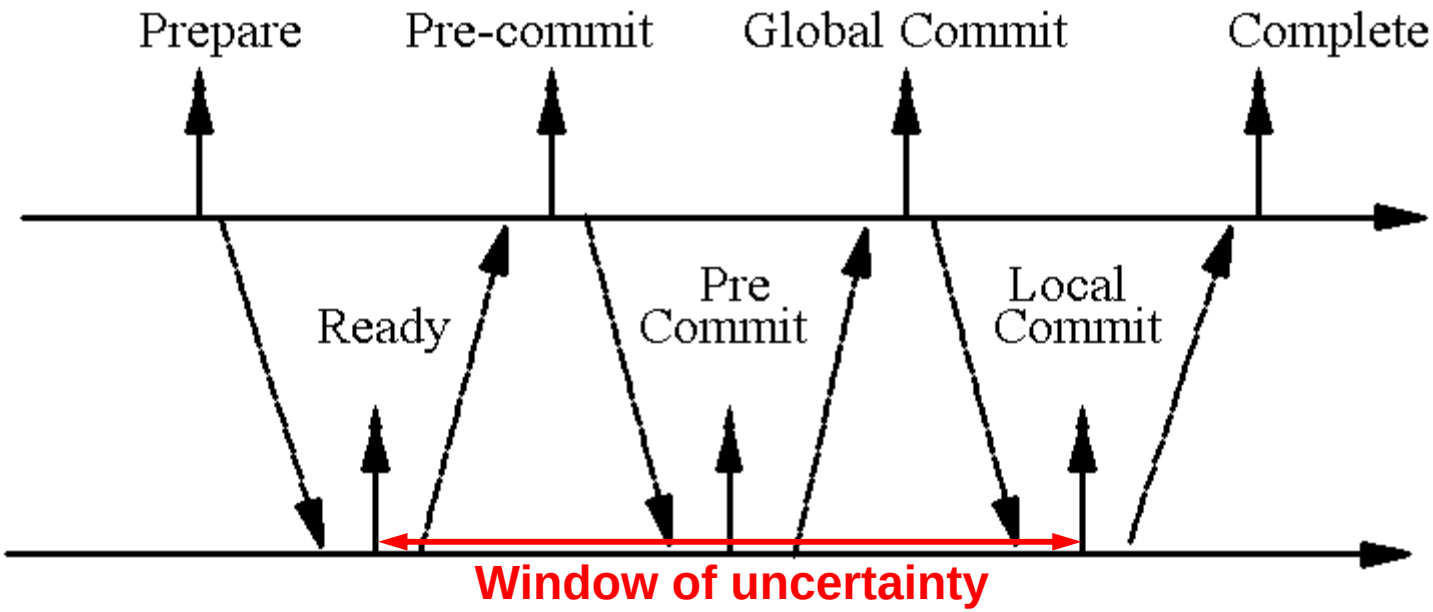
Diagram of the Four-phase Commit Protocol



Three-phase Commit Protocol

- Idea: thanks to a third phase, each participant can become a TM in case of failure of the current TM
- The "elected" participant looks at its log:
  - If the last record is **`ready`**, then it can impose a **`global abort`**
  - If it is **`pre-commit`**, it can impose a **`global commit`**
- Advantages:
  - the protocol is *non-blocking* (RMs are always autonomous)
- Disadvantages:
  - Lengthens the window of uncertainty (in average)
  - Not resilient to network partitioning (inconsistent decisions)

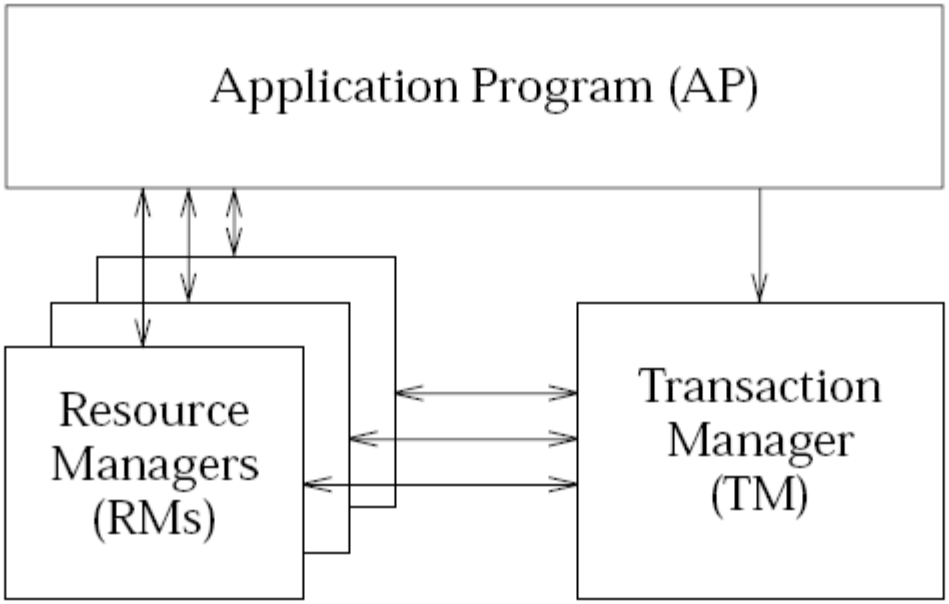
Diagram of the Three-phase Commit Protocol



Standardization of the Protocol

- Standard X-Open Distributed Transaction Processing (DTP):
  - TM interface
    - Defines the services of the **coordinator** offered to a client in order to execute commit of **heterogeneous** participants
  - XA interface
    - Defines the services of passive **participants** that respond to calls from the coordinator (offered by several commercial DBMSs, e.g., DB2®, Informix®, Oracle, Sybase, and Microsoft® SQL Server)

Standardization of the Protocol



(1) AP uses resources from a set of RMs

(2) AP defines transaction boundaries through the TX interface

(3) TM and RMs exchange transaction information

Features of X-Open DTP

- RMs are passive: they respond to remote procedure calls from the TMs
- Protocol: two-phase commit with optimizations (presumed abort and read-only)
- The protocol supports ***heuristic decisions***: after a failure, an operator can impose a heuristic decision (abort or commit)
  - When heuristic decisions raise inconsistencies, the client processes are notified

Features of X-Open DTP

- *Heuristic decisions*:
  - An RM that has prepared to commit a transaction branch may decide to commit or roll back its work independently of the TM. It could then unlock shared resources. This may leave them in an inconsistent state (loss of atomicity → the AP is notified). When the TM ultimately directs an RM to complete the branch, the RM may respond that it has already done so. The RM reports whether it committed the branch, rolled it back or completed it.
  - An RM that reports heuristic completion to the TM must not discard its knowledge of the transaction branch. The TM calls the RM once more to authorise it to forget the branch.

TM Interface

- **`tm_init`** and **`tm_exit`** initiate and terminate the client-TM dialogue
- **`tm_open`** and **`tm_term`** open and close a session with the TM
- **`tm_begin`** begins a transaction
- **`tm_commit`** requests a global commit
- **`tm_abort`** requests a global abort

XA Interface

- **`xa_open`** and **`xa_close`** open and close a TM-RM dialog
- **`xa_start`** and **`xa_end`** activate and complete a new transaction
- **`xa_precomm`** requests that the RM carry out the first phase of the commit protocol
- **`xa_commit`** and **`xa_abort`** communicate the "global decision" to the RM
- **`xa_recover`** initiates an RM recovery; the RM responds to the request with three sets of transactions:
  - Transactions *in doub*t
  - Transactions decided by a *heuristic commi*t
  - Transactions decided by a *heuristic abor*t
- **`xa_forget`** allows an RM to forget transactions decided in a heuristic manner

Commit Protocols

Interactions