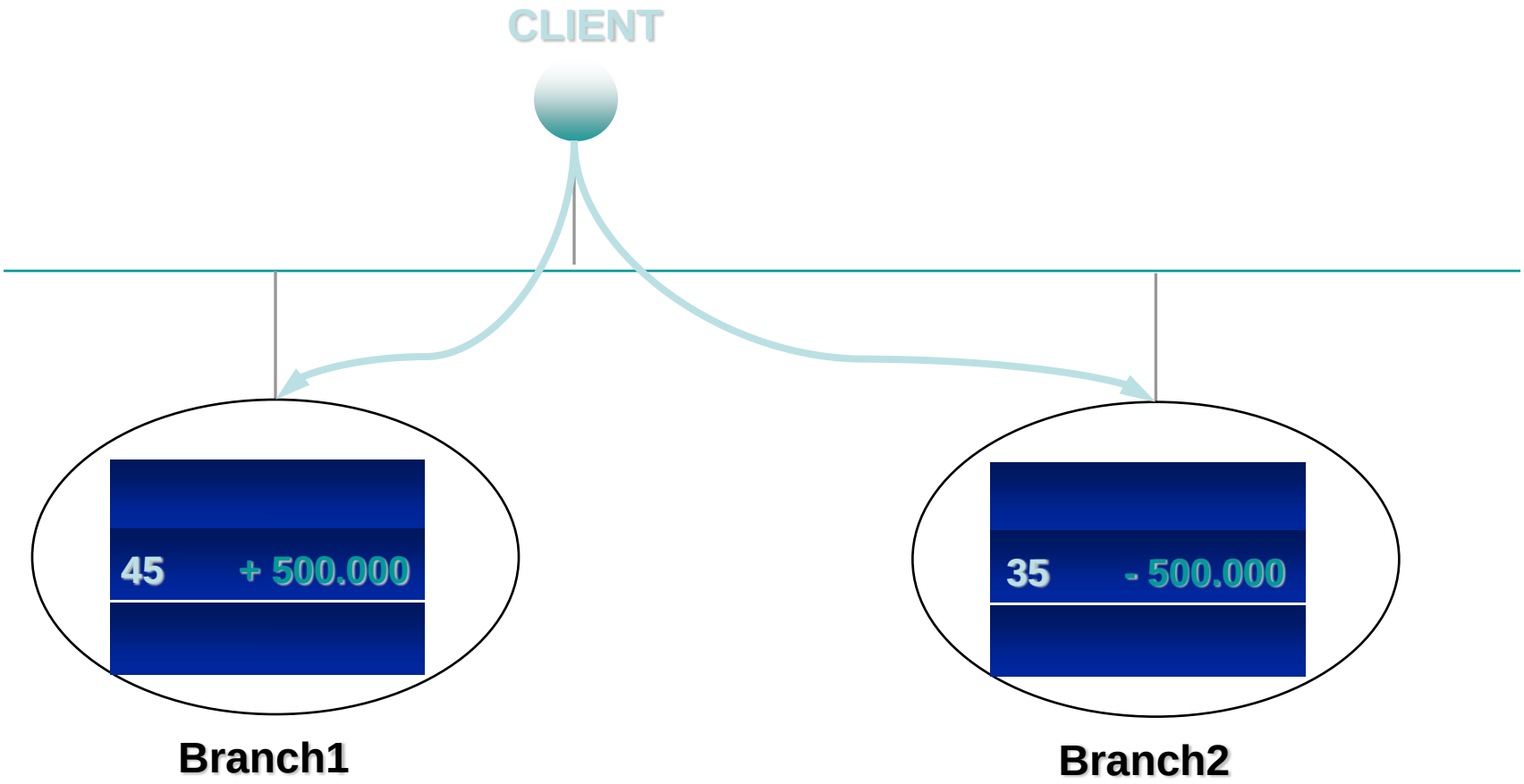
Advanced Databases

4 Commit Protocols

Distributed Transactions

```
begin transaction
        update Account1@1
        set Balance=Balance + 500.000
        where AccNum=45;
        update Account2@2
        set Balance=Balance - 500.000
        where AccNum=35;
commit work
end transaction
```

Distributed Transactions



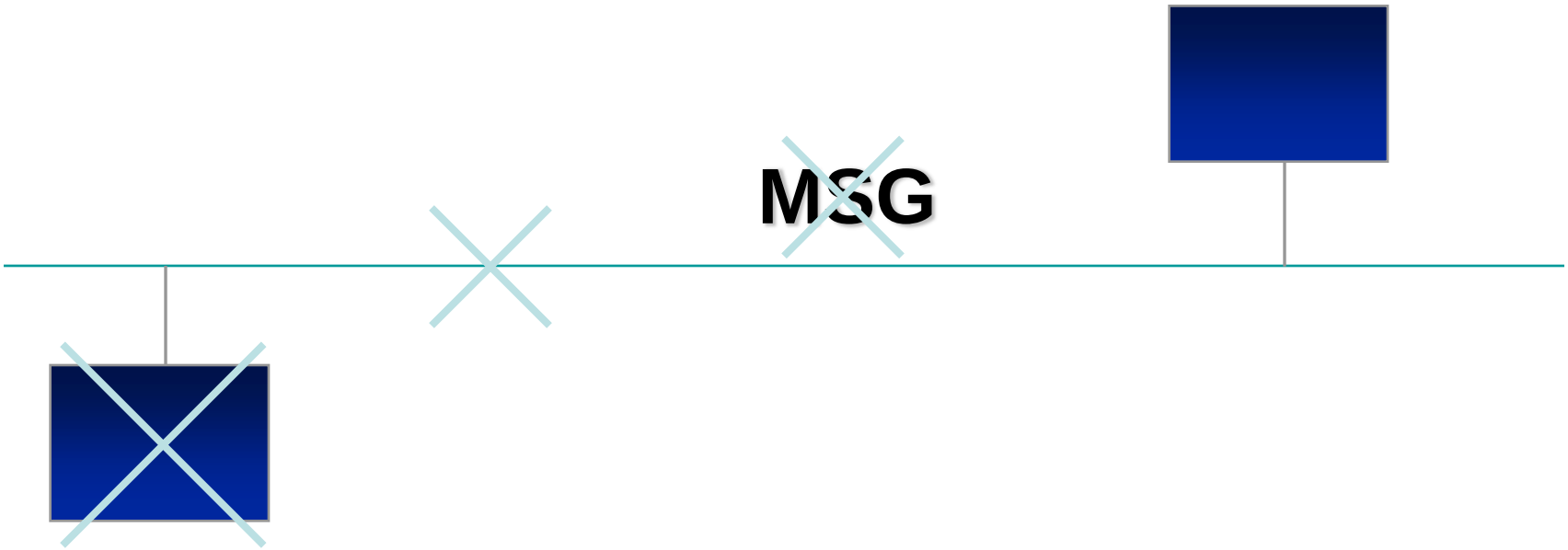
Commit Protocols

ACID Properties of Distributed Execution

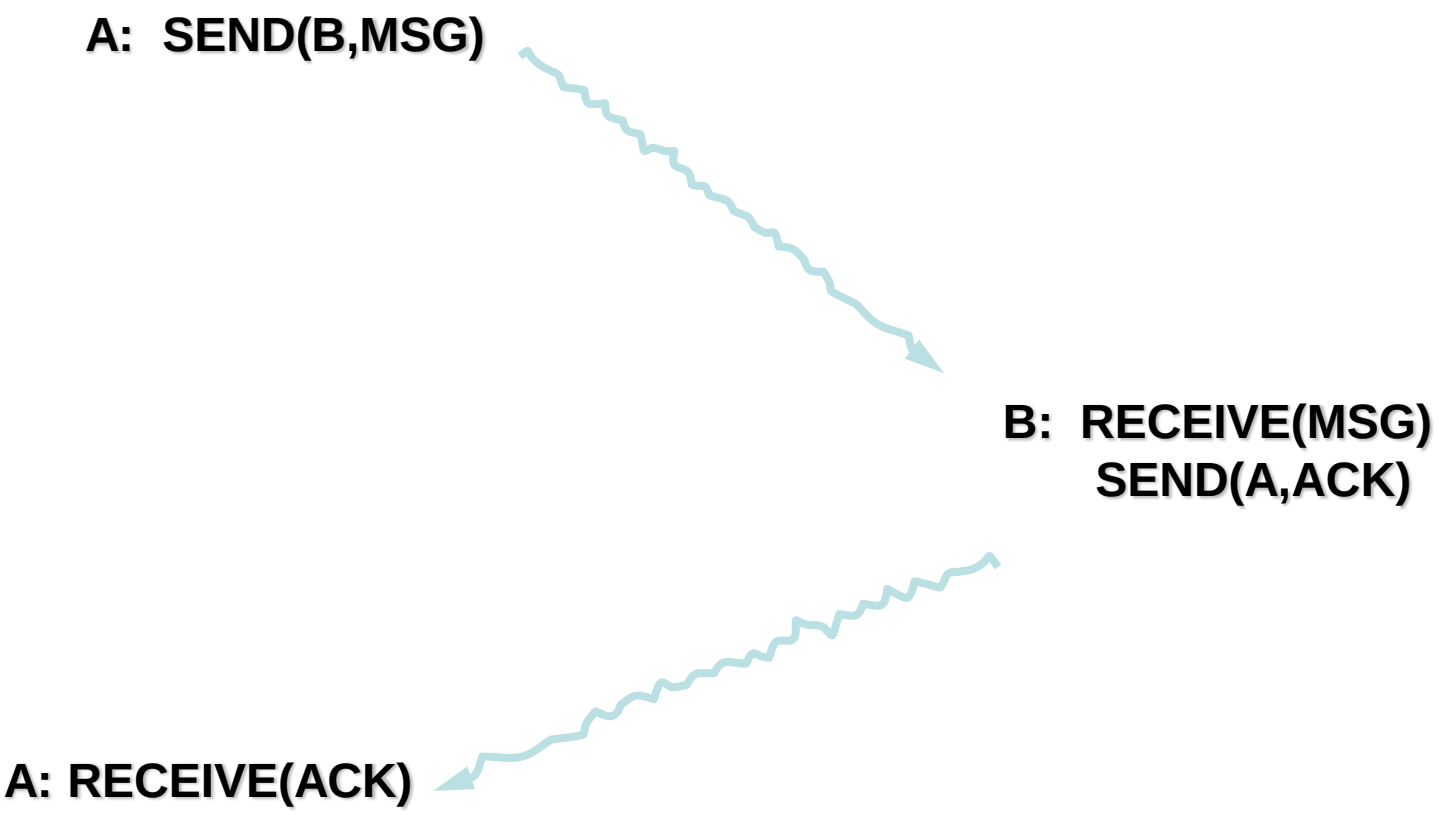
- Isolation
  - If each sub-transaction is two-phase, the transaction is globally serializable
- Consistency
  - If each sub-transaction preserves local integrity, data are globally consistent
- Durability
  - If each sub-transaction handles logs correctly, data are globally persistent
- Atomicity
 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

Faults in a Distributed System

- Node failures
- Message losses
- Network partitioning



Distributed Protocols and Message Losses



Two-phase Commit Protocol

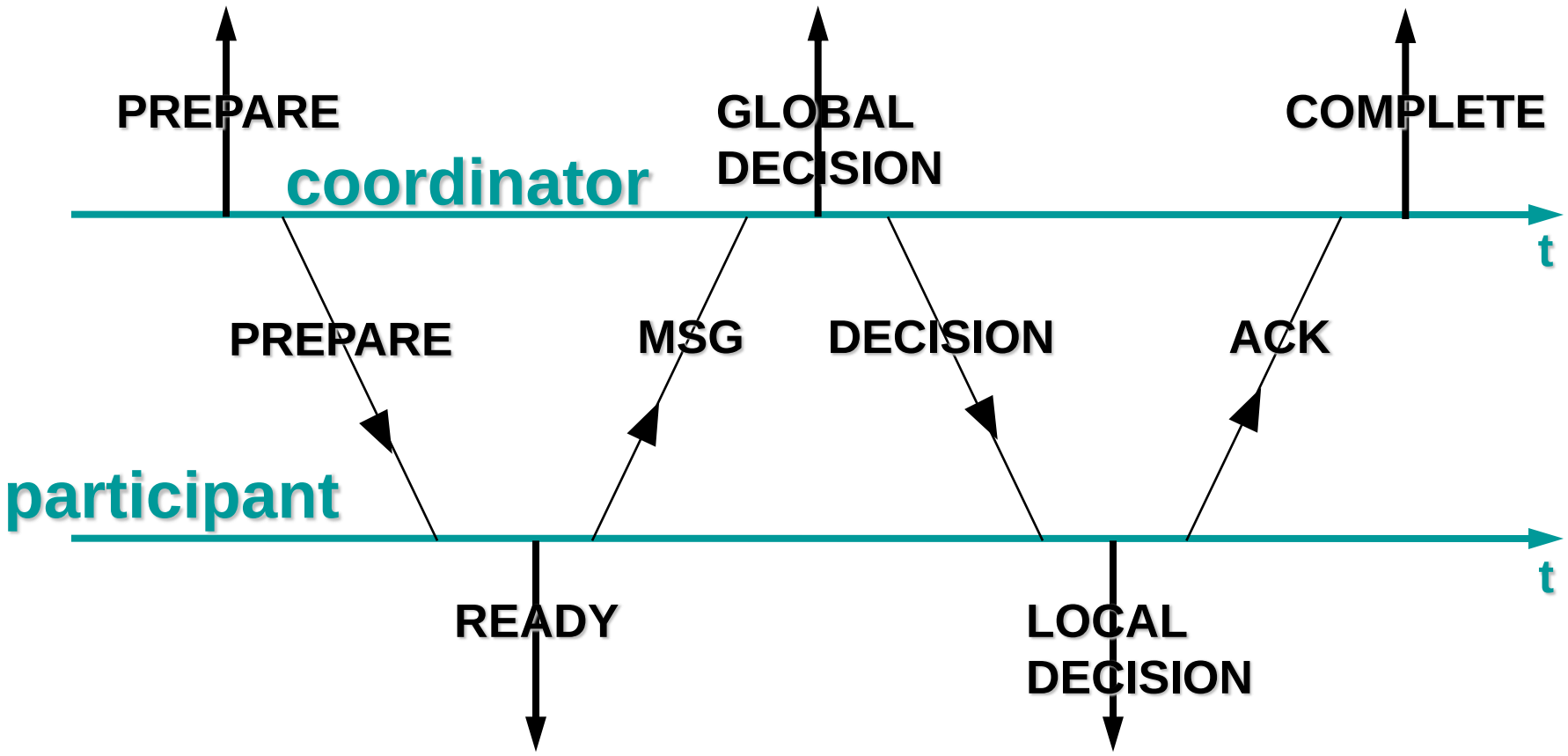
- Protocol that guarantees atomicity of distributed sub-transactions
- Protagonists:
  - A coordinator (Transaction Manager, TM)
  - Several participants (Resource Manager, RM)
- Similar to a marriage
  - Phase one: the decision is declared
  - Phase two: the marriage is ratified

New Log Records

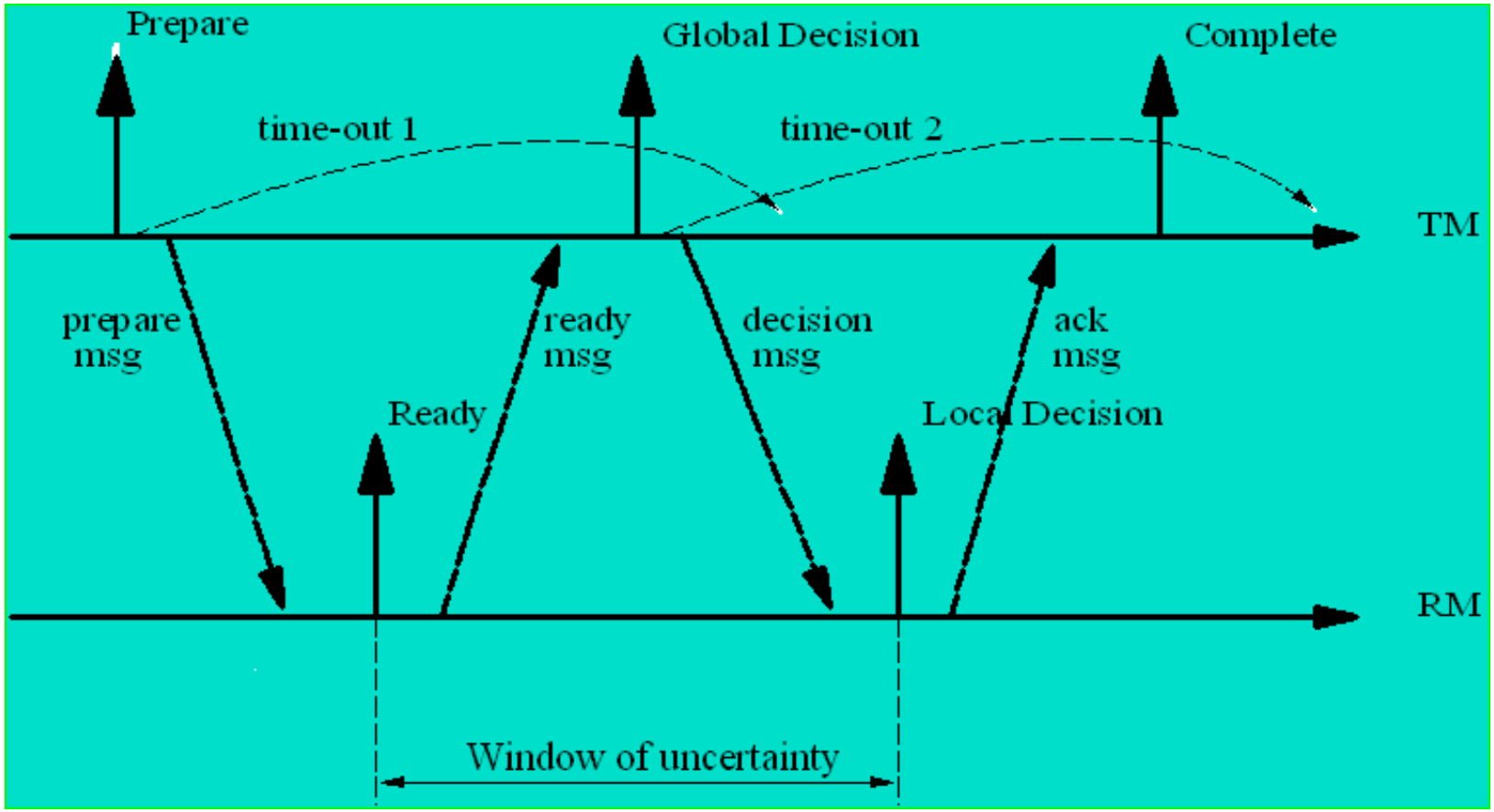
- In the coordinator's log
  - **`prepare`**: participants' identity
  - **`global commit/abort`**: decision
  - **`complete`**: end of protocol
- In the participant's log
  - **`ready`**: availability to participate to commit
  - **`local commit/abort`**: decision received

Commit Protocols

Diagram of the Two-phase Commit Protocol



Protocol with Time-out and Window of Uncertainty

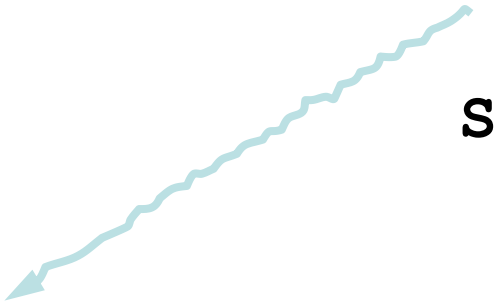


Phase 1

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

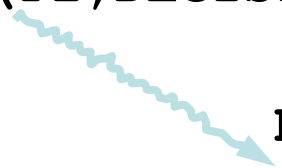


```
Pi: RECEIVE(C,PREPARE)
    IF OK THEN WRITE-LOG(READY)
                READY=YES
           ELSE READY=NO
    SEND(C,READY)
```



Phase 2

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



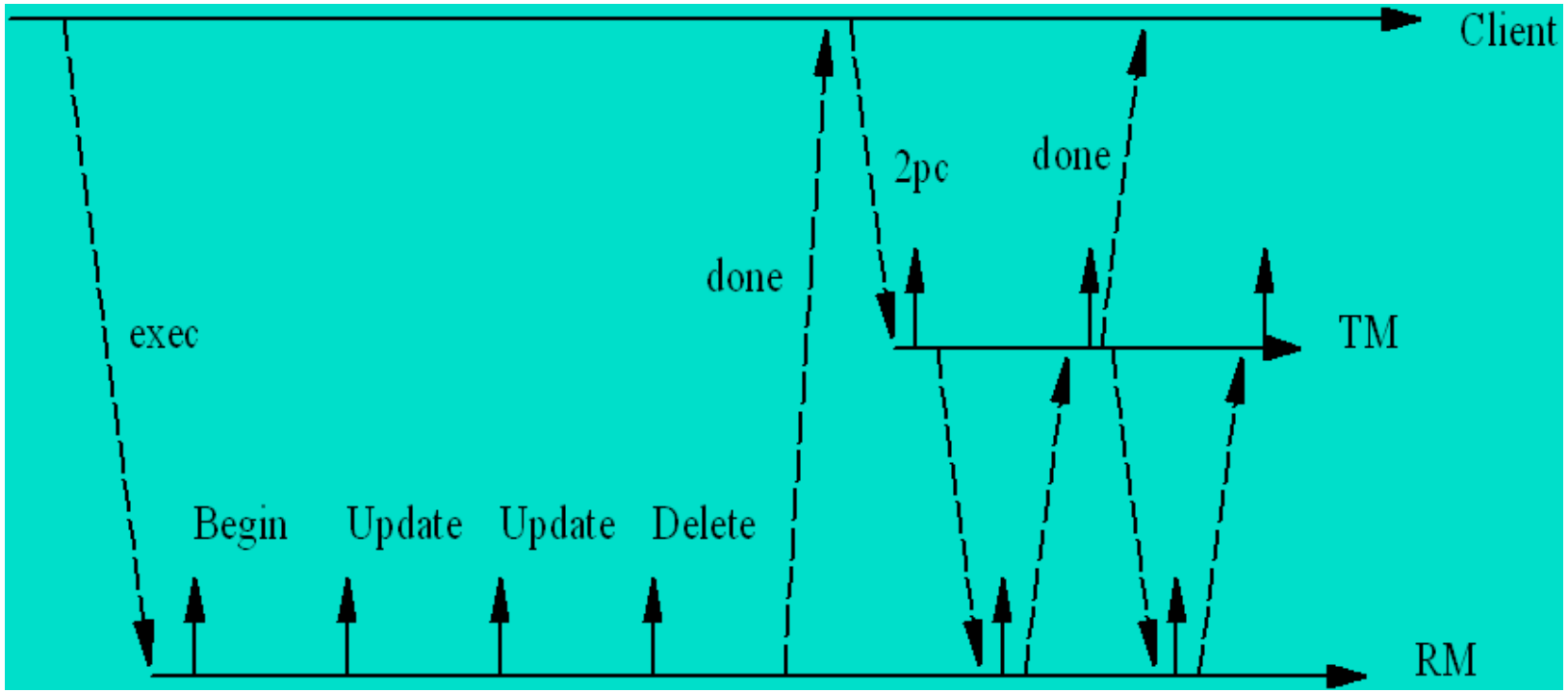
```
Pi: RECEIVE(C,DECISION)
      IF COMMIT THEN WRITE-LOG(COMMIT)
                ELSE WRITE-LOG(ABORT)
      SEND (C,ACK)
```



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

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

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*; when it is a **`commit`**, the actions are *redone*; recovery doesn't depend on the protocol
  - If it is a **`ready`**, the failure has occurred during the two-phase commit. The participant is *in doubt* about the result of the transaction
- During the warm restart protocol, the identifier of the transactions in doubt are collected. For each of them the final transaction outcome must be requested to the TM (*remote* recovery request)

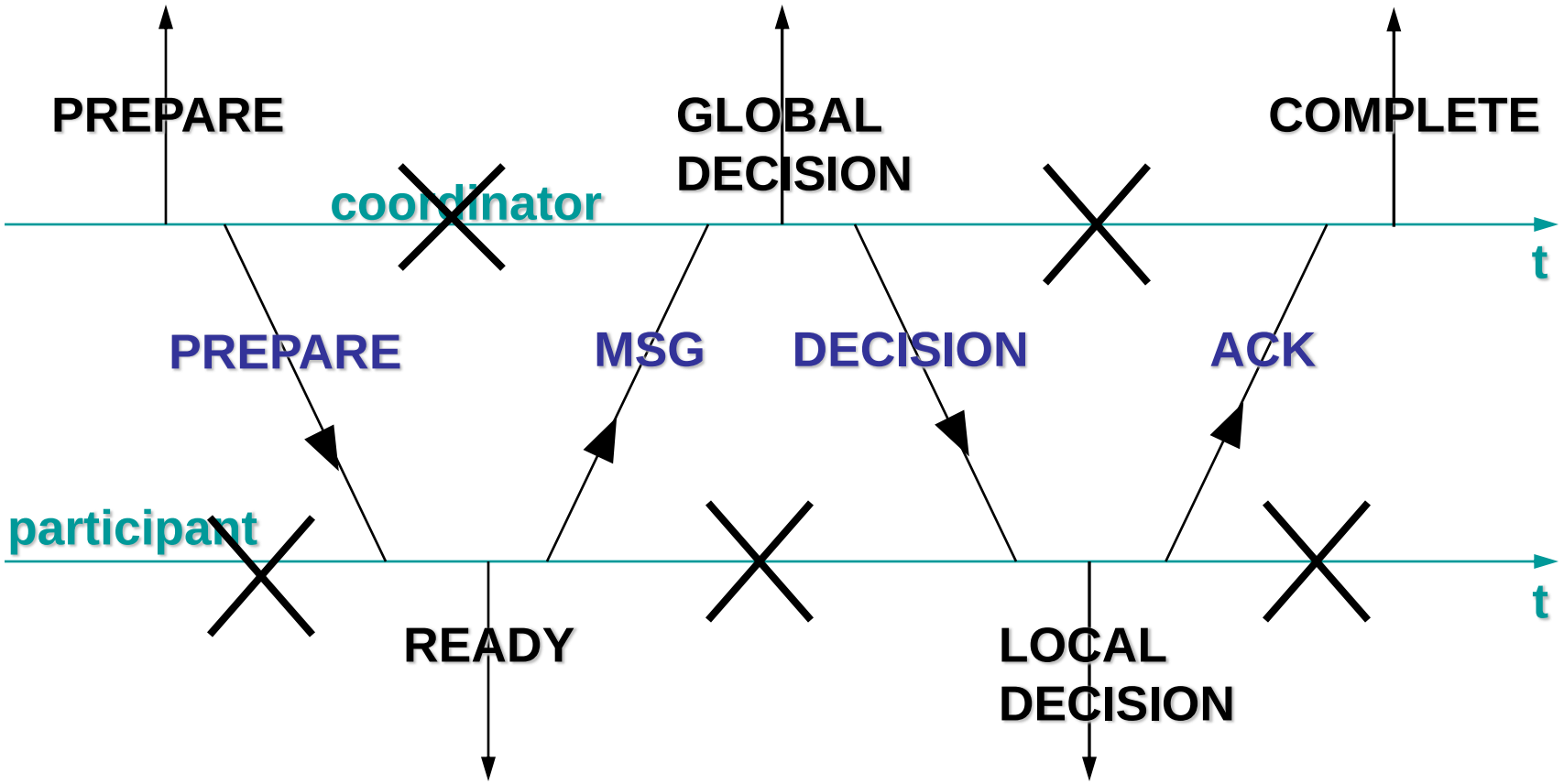
Recovery of the Coordinator

- 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. 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

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 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

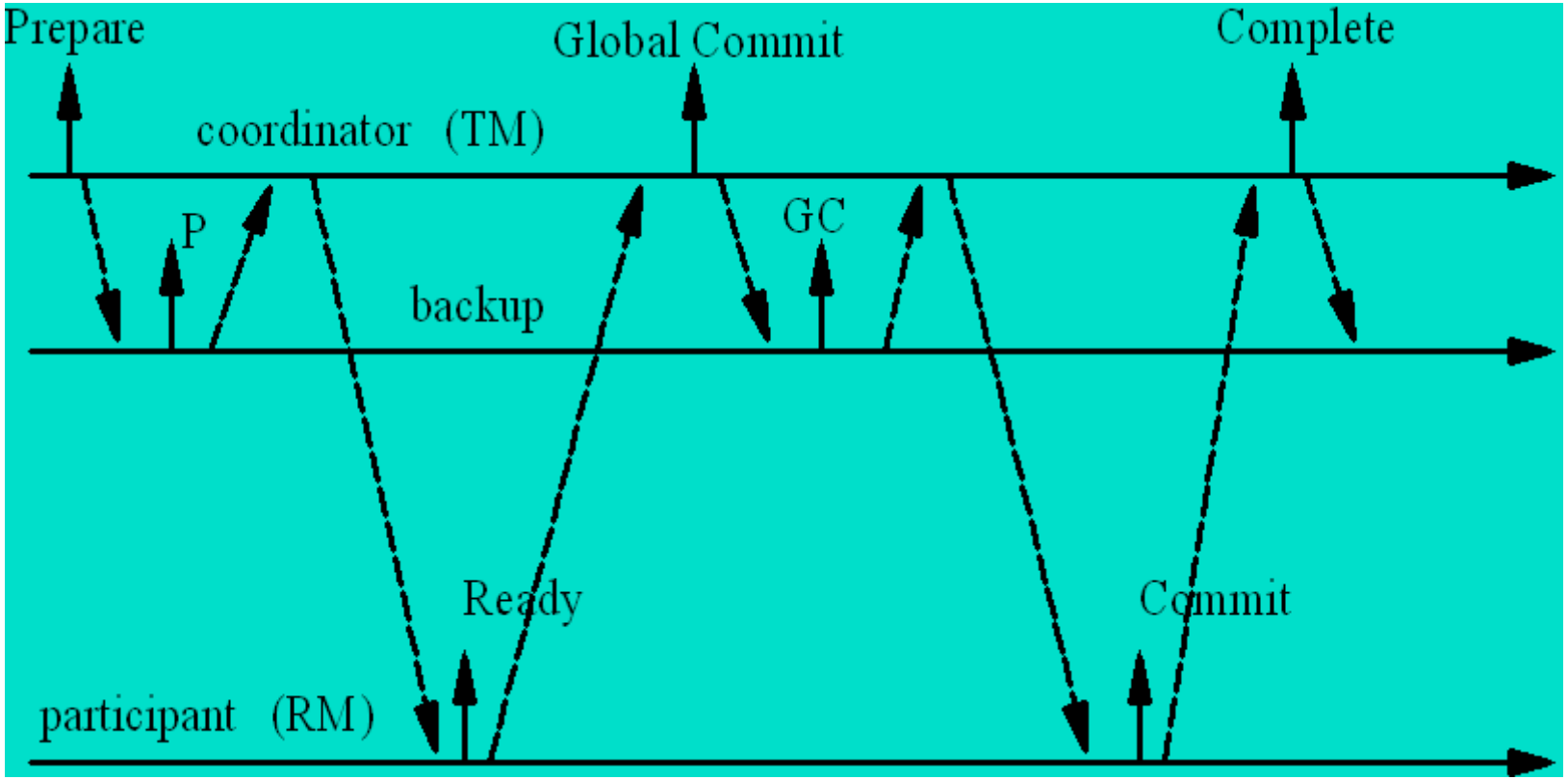
Blocking, Uncertainty, Recovery Protocols

- An RM in a "ready" state loses its autonomy and awaits the decision of the TM. 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

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

Diagram of the Four-phase Commit Protocol

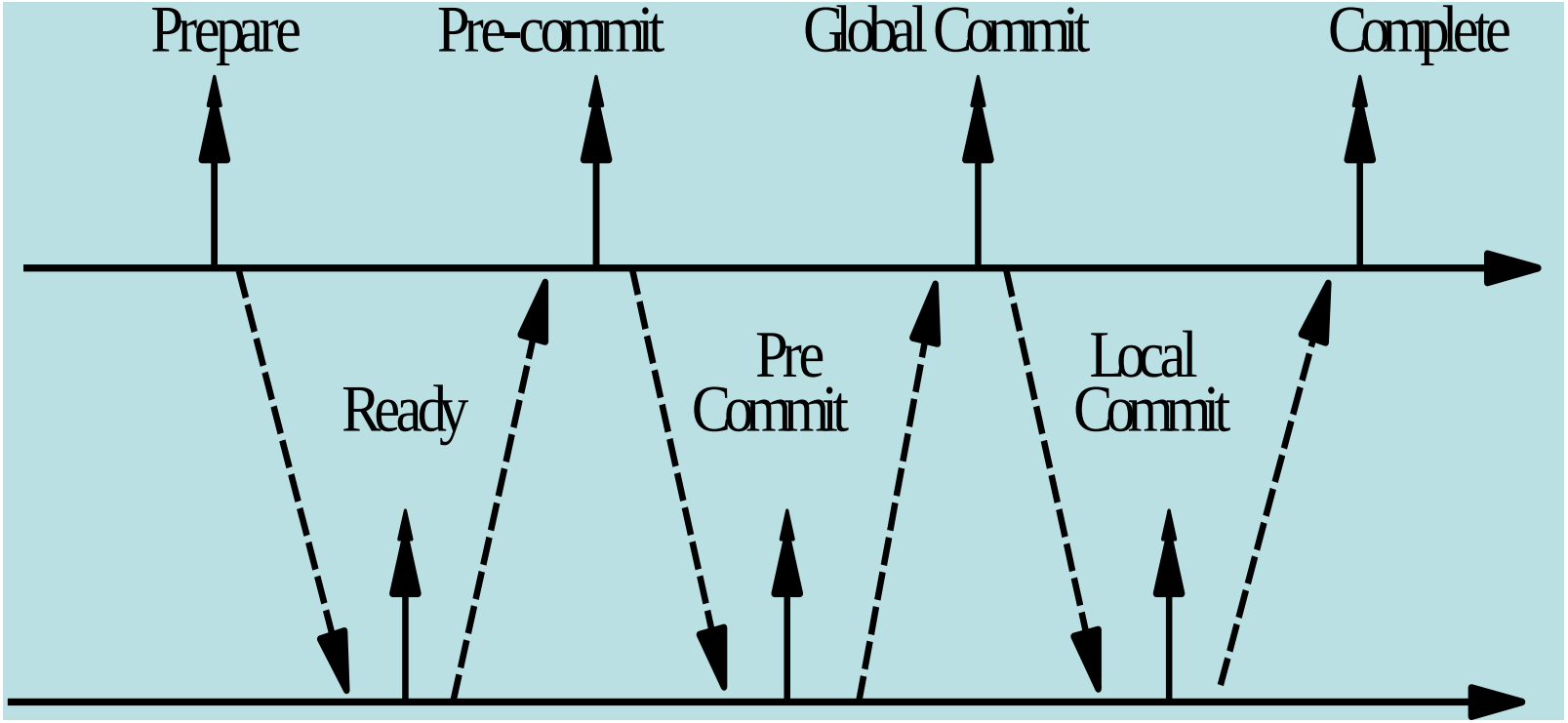


Three-phase Commit Protocol

- Idea: thanks to a third phase, each participant can become a TM
- The "elected" participant looks at its log:
  - If the last record is **`ready`**, then it can impose a **`global abort`**
  - If the last record is **`pre-commit`**, it can impose a **`global commit`**
- Shortcomings:
  - It lengthens the window of uncertainty
  - It is not resilient to network partitioning

Commit Protocols

Diagram of the Three-phase Commit Protocol

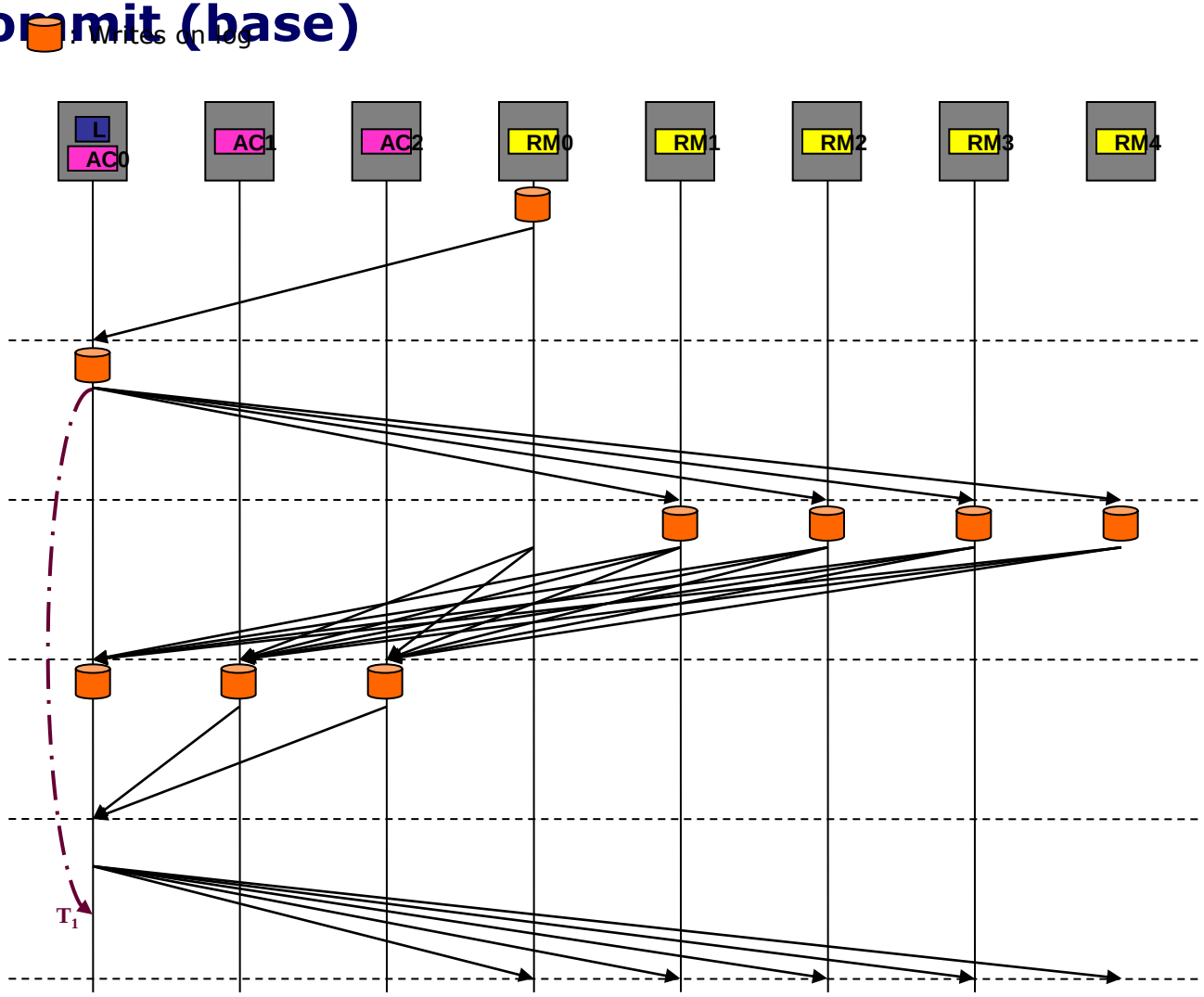


Paxos Commit Protocol

- Idea: combine a "consensus protocol" with a commit protocol so as to guarantee one decision even in the presence of partitions.
- Paxos consensus protocol: with a network that cannot fail more than F times, establish one "initiator" and 2F+1 "acceptors"; the protocol guarantees consensus with a majority (F+1) of acceptors. Requires 3 phases.
- Paxos commit protocol (Gray-Lamport, ACM-TODS 2007) sets some RM (out of N) as acceptors and guarantees commit/rollback decision with N(F+3) -3 messages.
- The idea: acceptors have to be co-ordinated to reach consensus, a lot of messages are saved by making RM and acceptors coincident.
- Two-phase commit is a special case of Paxos commit with F=0 (coordinator=acceptor).

Commit Protocols

Paxos Commit (base)



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)

Features of X-Open DTP

- RMs are passive: the 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

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