Dare 2023 - Summer School

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

André Costa and David Antunes

TLA+: 2PC and 3PC Spec

Introduction

- Two Phase Commit
- Leslie Lamport's Spec

Introduction

- Two Phase Commit
- Leslie Lamport's Spec

Specification of Atomic Commit

Specification of Atomic Commit

Model:

Crash Recovery

Synchronous System

Specification of Atomic Commit

Model: Crash Recovery

Agreement: Any two processes that decide, decide the same value.

Specification of Atomic Commit

Model: Crash Recovery

Agreement: Any two processes that decide, decide the same value.

Validity (1): If some process starts with the value “no” then “abort” is the only possible decision.

Specification of Atomic Commit

Model: Crash Recovery

Agreement: Any two processes that decide, decide the same value.

Validity (1): If some process starts with the value “no” then “abort” is the only possible decision.

Validity (2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision.

Specification of Atomic Commit

Model: Crash Recovery

Agreement: Any two processes that decide, decide the same value.

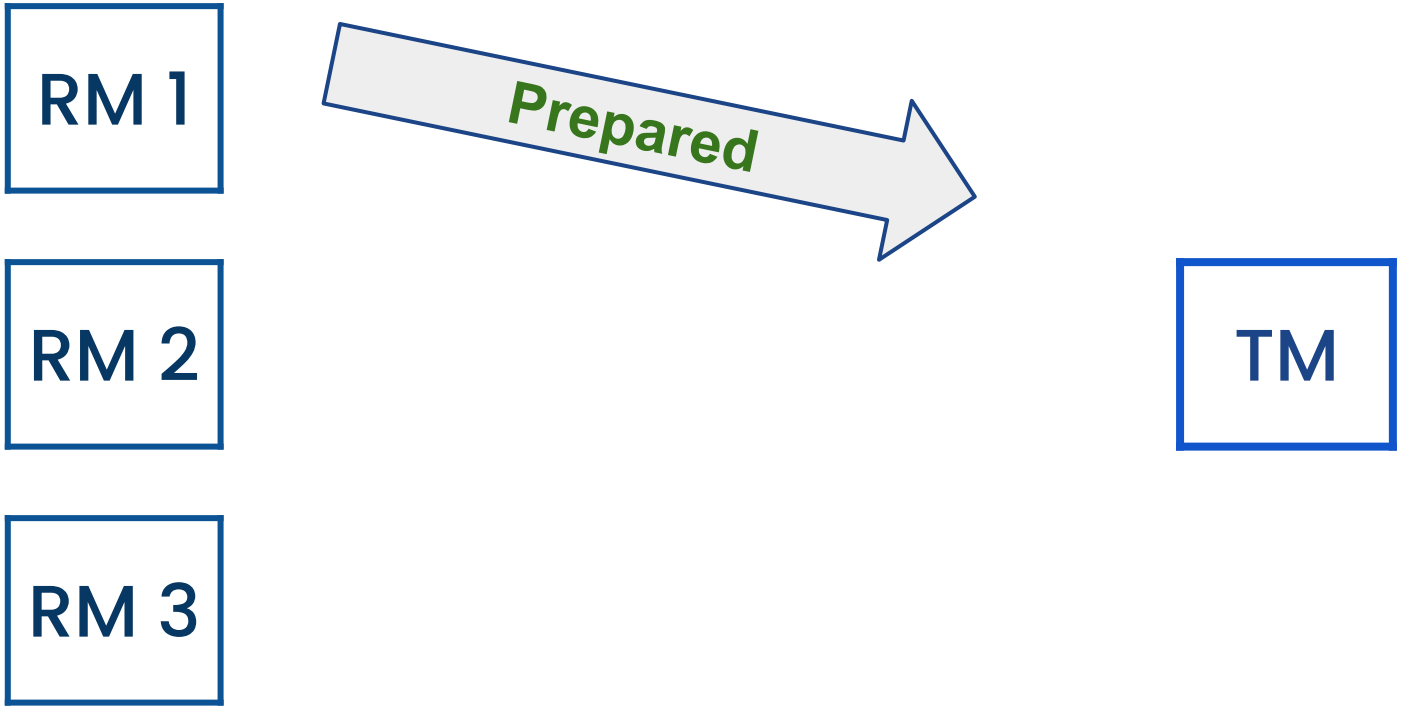
Validity (1): If some process starts with the value “no” then “abort” is the only possible decision.

Validity (2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision.

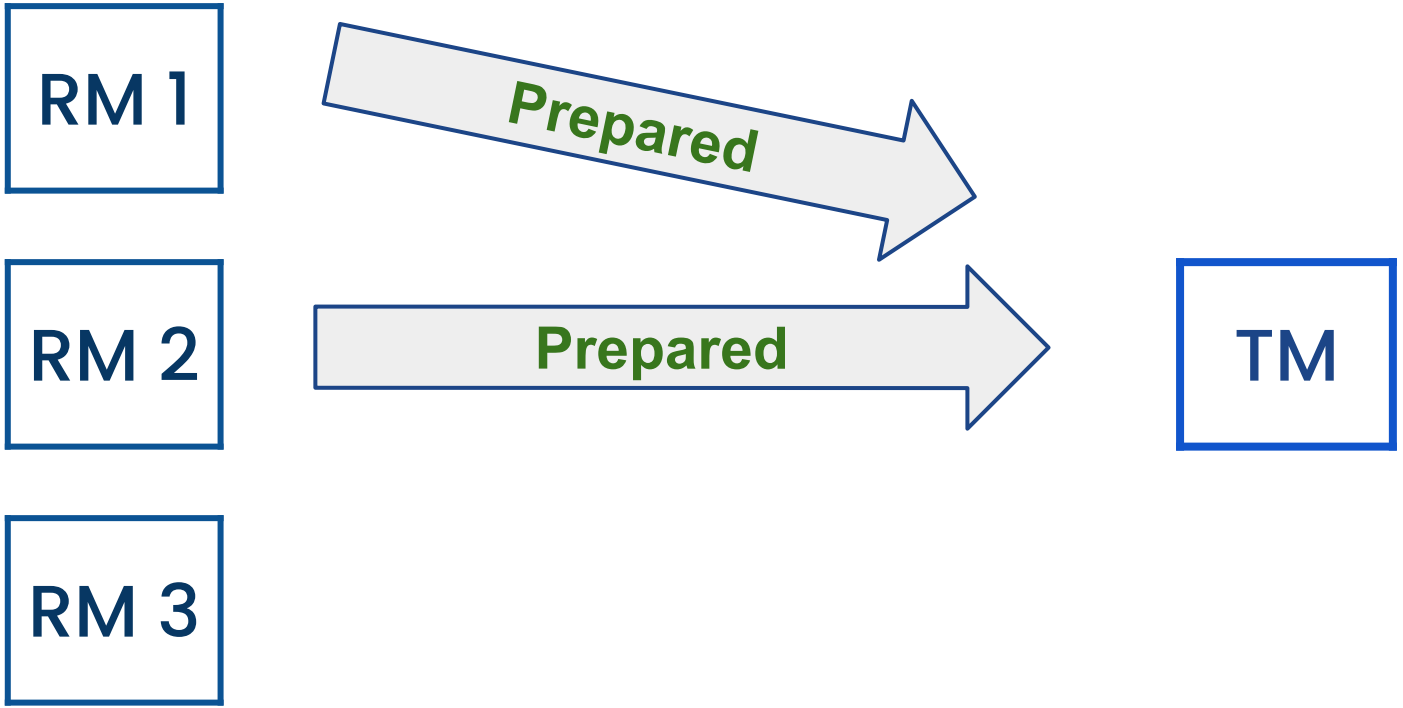
Termination: If eventually all processes recover from all faults, then, eventually all processes decide.

Two Phase Commit

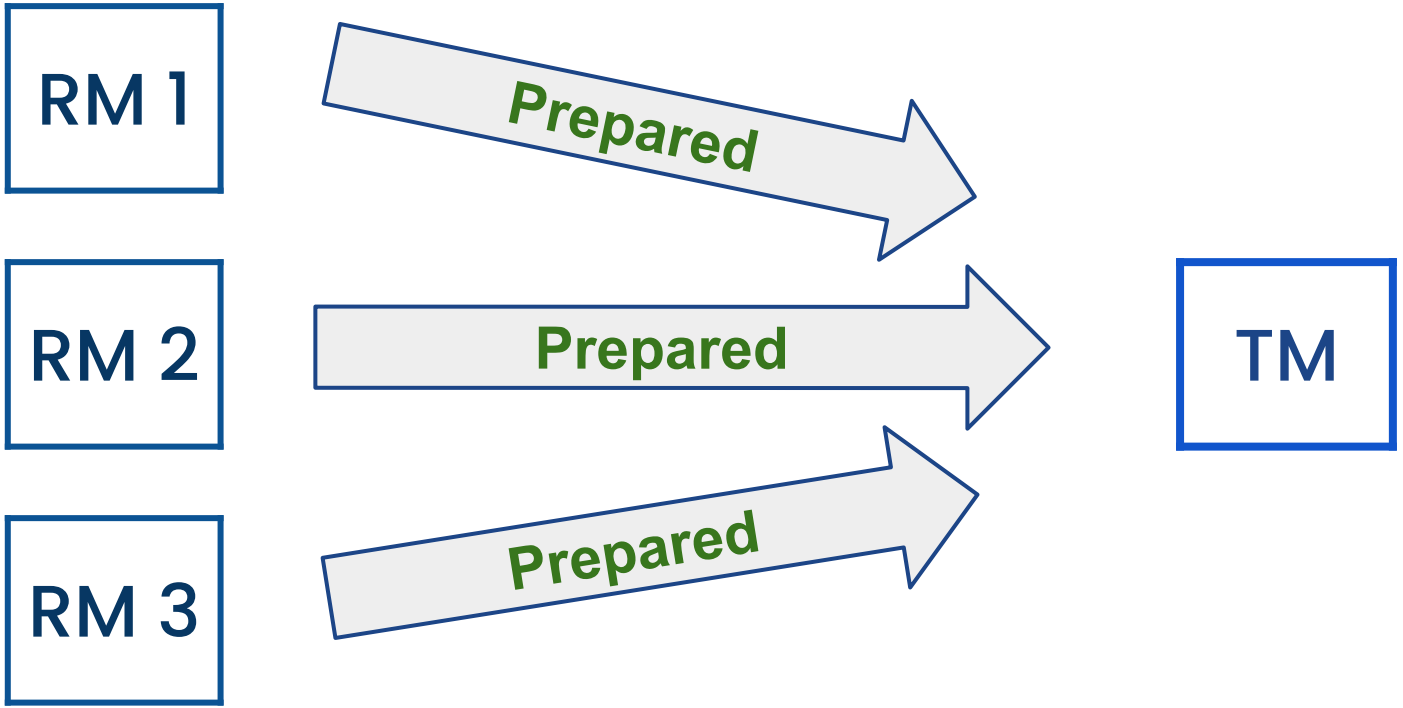
Two Phase Commit



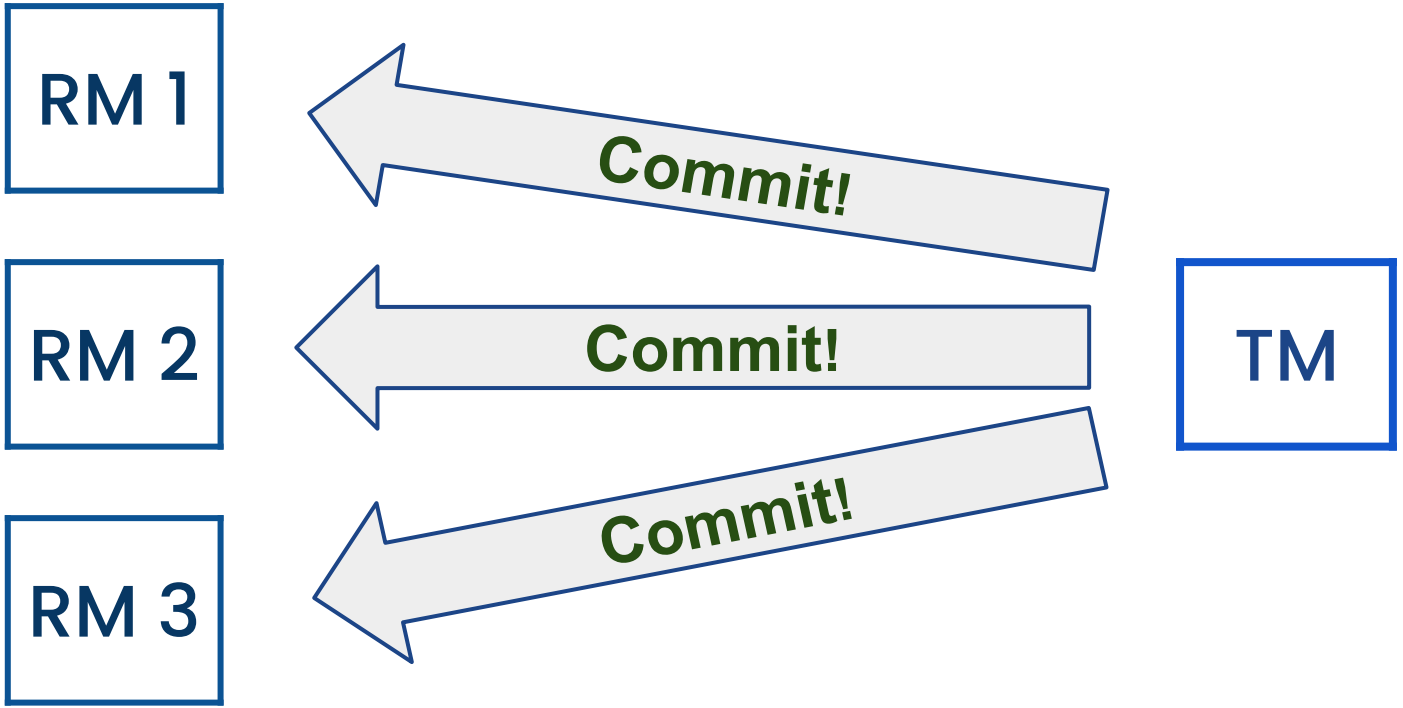
Two Phase Commit



Two Phase Commit

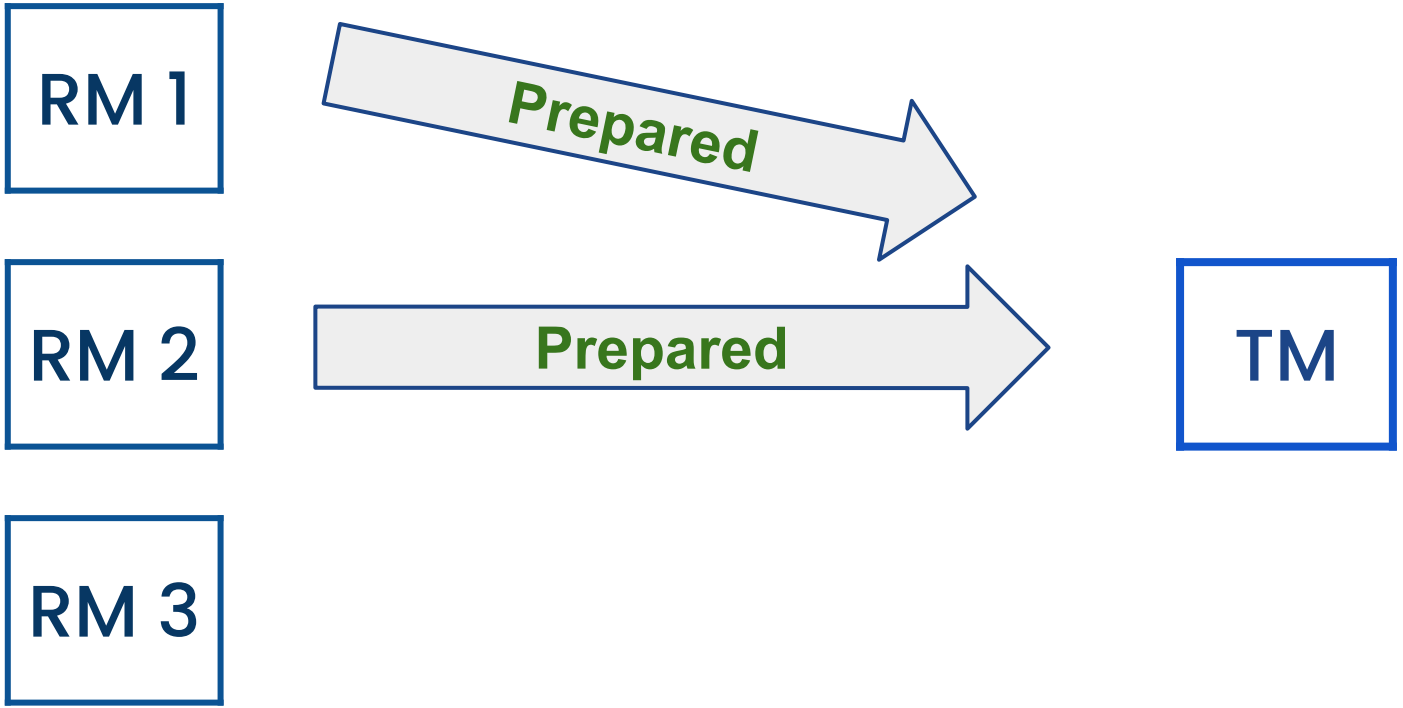


Two Phase Commit

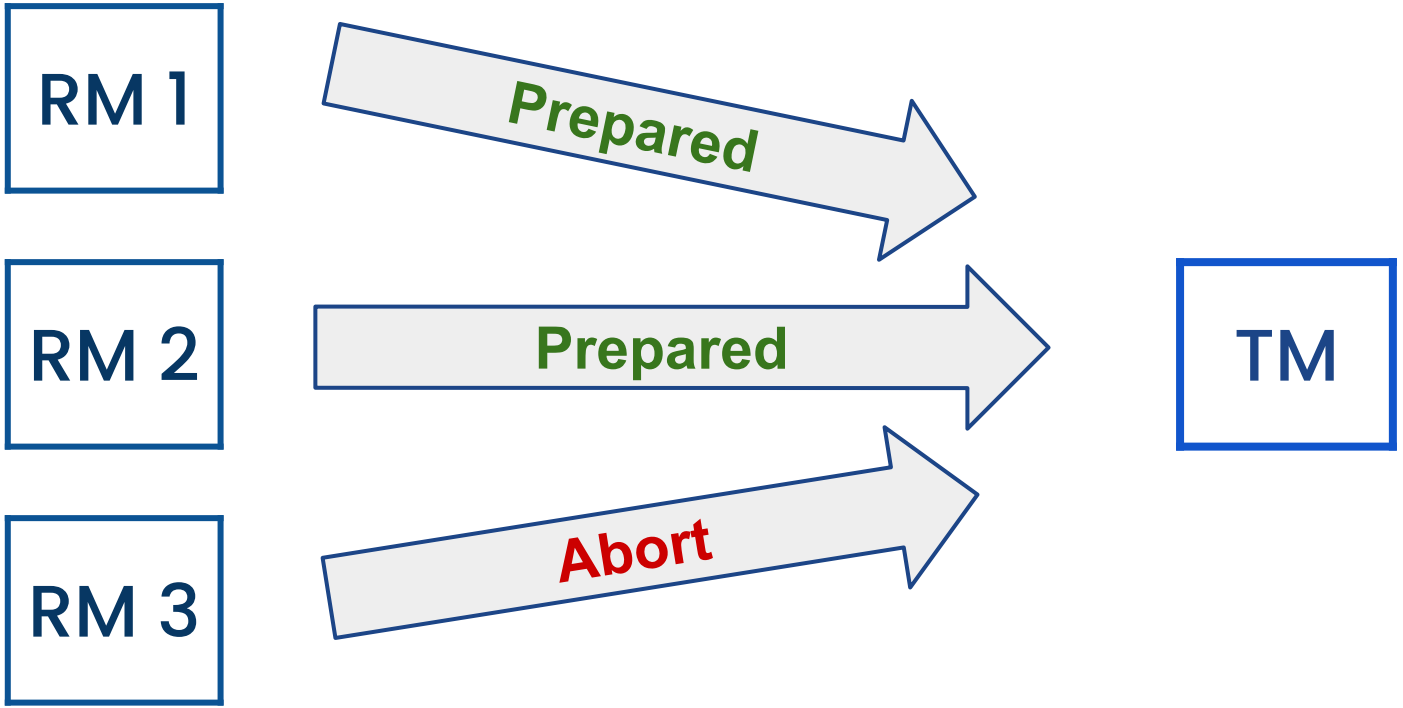


Two Phase Commit

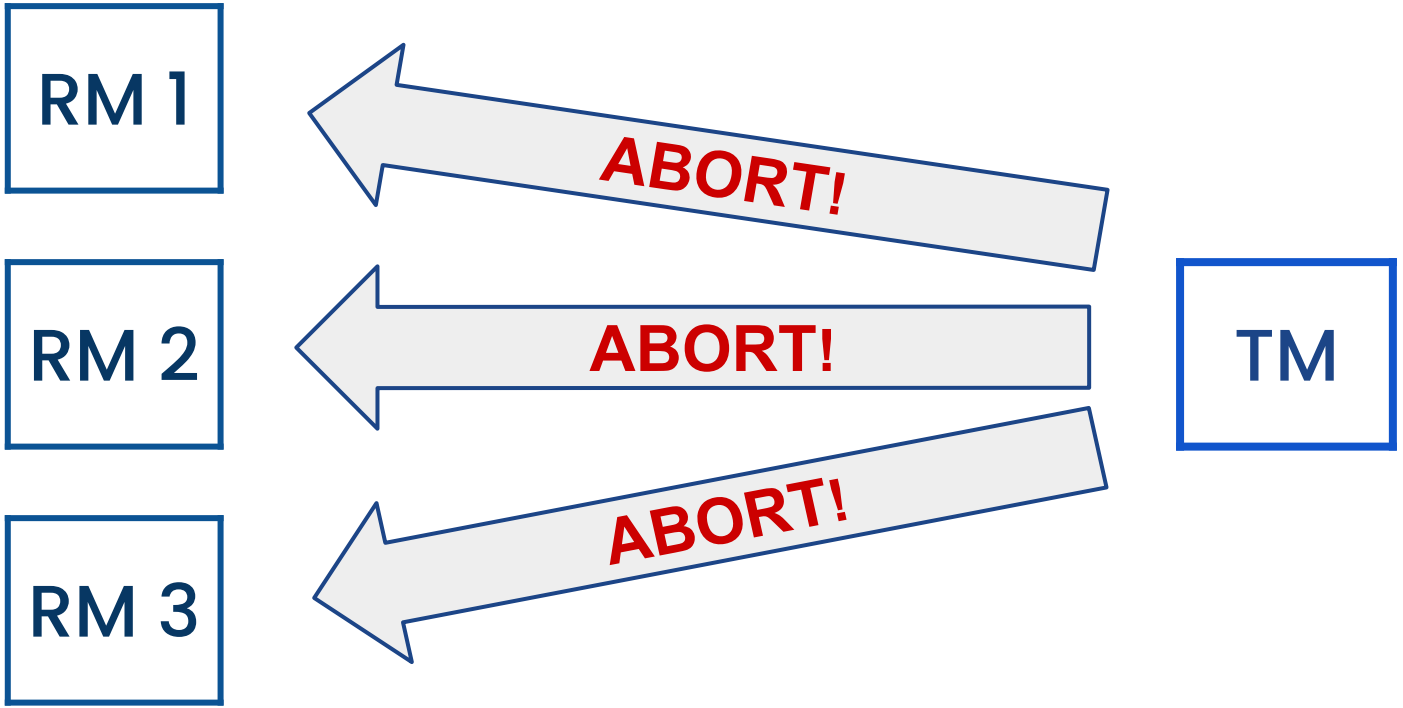
Two Phase Commit



Two Phase Commit



Two Phase Commit



Introduction

- Two Phase Commit
- Leslie Lamport's Spec

Introduction

- Two Phase Commit
- Leslie Lamport's Spec

Leslie Lamport’s Specification

Leslie Lamport's Specification

```
CONSTANT RM \* The set of resource managers
```

Leslie Lamport’s Specification

```
CONSTANT RM \* The set of resource managers

VARIABLES
```

Leslie Lamport's Specification

```
CONSTANT RM  \* The set of resource managers

VARIABLES
 rmState      \* State of each resource manager.
```

Leslie Lamport's Specification

```
CONSTANT RM  \* The set of resource managers

VARIABLES
 rmState     \* State of each resource manager.
 tmState     \* The state of the transaction manager.
```

Leslie Lamport's Specification

```
CONSTANT RM  \* The set of resource managers

VARIABLES
 rmState     \* State of each resource manager.
 tmState     \* The state of the transaction manager.
 tmPrepared  \* The set of RMs from which the TM has received prepared messages
```

Leslie Lamport's Specification

```
CONSTANT RM  \* The set of resource managers

VARIABLES
 rmState     \* State of each resource manager.
 tmState     \* The state of the transaction manager.
 tmPrepared  \* The set of RMs from which the TM has received prepared messages
 msgs        \* The set of all messages ever sent
```

Leslie Lamport’s Specification

Message ==

Leslie Lamport's Specification

```
Message ==
    [type : {"Prepared"}, rm : RM]  \cup  [type : {"Commit", "Abort"}]
```

Leslie Lamport's Specification

```
Message ==
    [type : {"Prepared"}, rm : RM]  \cup  [type : {"Commit", "Abort"}]


TPTypeOK ==
```

Leslie Lamport's Specification

```
Message ==
    [type : {"Prepared"}, rm : RM]  \cup  [type : {"Commit", "Abort"}]


TPTypeOK ==
    /\ rmState \in [RM -> {"working", "prepared", "committed", "aborted"}]
```

Leslie Lamport's Specification

```
Message ==
    [type : {"Prepared"}, rm : RM]  \cup  [type : {"Commit", "Abort"}]


TPTypeOK ==
    /\ rmState \in [RM -> {"working", "prepared", "committed", "aborted"}]
    /\ tmState \in {"init", "committed", "aborted"}
```

Leslie Lamport's Specification

```
Message ==
    [type : {"Prepared"}, rm : RM]  \cup  [type : {"Commit", "Abort"}]


TPTypeOK ==
    /\ rmState \in [RM -> {"working", "prepared", "committed", "aborted"}]
    /\ tmState \in {"init", "committed", "aborted"}
    /\ tmPrepared \subseteq RM
```

Leslie Lamport's Specification

```
Message ==
    [type : {"Prepared"}, rm : RM]  \cup  [type : {"Commit", "Abort"}]


TPTypeOK ==
    /\ rmState \in [RM -> {"working", "prepared", "committed", "aborted"}]
    /\ tmState \in {"init", "committed", "aborted"}
    /\ tmPrepared \subseteq RM
    /\ msgs \subseteq Message
```

Leslie Lamport's Specification

(Actions that lead to Commit)

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
        /\ rmState[rm] = "working"
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "prepared"]
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "prepared"]
    /\ msgs' = msgs \cup {[type |-> "Prepared", rm |-> rm]}
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "prepared"]
    /\ msgs' = msgs \cup {[type |-> "Prepared", rm |-> rm]}

\* Transaction Manager receives prepare message
TMRcvPrepared(rm) ==
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "prepared"]
    /\ msgs' = msgs \cup {[type |-> "Prepared", rm |-> rm]}

\* Transaction Manager receives prepare message
TMRcvPrepared(rm) ==
    /\ tmState = "init"
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "prepared"]
    /\ msgs' = msgs \cup {[type |-> "Prepared", rm |-> rm]}

\* Transaction Manager receives prepare message
TMRcvPrepared(rm) ==
    /\ tmState = "init"
    /\ [type |-> "Prepared", rm |-> rm] \in msgs
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Resource Manager sends prepare message
RMPrepare(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "prepared"]
    /\ msgs' = msgs \cup {[type |-> "Prepared", rm |-> rm]}

\* Transaction Manager receives prepare message
TMRcvPrepared(rm) ==
    /\ tmState = "init"
    /\ [type |-> "Prepared", rm |-> rm] \in msgs
    /\ tmPrepared' = tmPrepared \cup {rm}
```

Leslie Lamport's Specification

(Actions that lead to Commit)

Leslie Lamport's Specification

(Actions that lead to Commit)

* Transaction Manager sends **Commit** message

TMCommit ==

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
        /\ tmState = "init"
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
        /\ tmState = "init"
        /\ tmPrepared = RM
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
        /\ tmState = "init"
        /\ tmPrepared = RM
        /\ tmState' = "committed"
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
        /\ tmState = "init"
        /\ tmPrepared = RM
        /\ tmState' = "committed"
        /\ msgs' = msgs \cup {[type |-> "Commit"]}
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
        /\ tmState = "init"
        /\ tmPrepared = RM
        /\ tmState' = "committed"
        /\ msgs' = msgs \cup {[type |-> "Commit"]}


\* Resource Manager receives Commit message
RMRcvCommitMsg(rm) ==
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
    /\ tmState = "init"
    /\ tmPrepared = RM
    /\ tmState' = "committed"
    /\ msgs' = msgs \cup {[type |-> "Commit"]}

\* Resource Manager receives Commit message
RMRcvCommitMsg(rm) ==
    /\ [type |-> "Commit"] \in msgs
```

Leslie Lamport's Specification

(Actions that lead to Commit)

```
\* Transaction Manager sends Commit message
TMCommit ==
    /\ tmState = "init"
    /\ tmPrepared = RM
    /\ tmState' = "committed"
    /\ msgs' = msgs \cup {[type |-> "Commit"]}

\* Resource Manager receives Commit message
RMRcvCommitMsg(rm) ==
    /\ [type |-> "Commit"] \in msgs
    /\ rmState' = [rmState EXCEPT ![rm] = "committed"]
```

Leslie Lamport’s Specification

Leslie Lamport's Specification

Transaction Manager can decide to abort (even when all RMs want to commit)

Leslie Lamport's Specification

Transaction Manager can decide to abort (even when all RMs want to commit)

```
\* The TM spontaneously aborts the transaction.
TMAbort ==
        /\ tmState = "init"
```

Leslie Lamport's Specification

Transaction Manager can decide to abort
(even when all RMs want to commit)

```
\* The TM spontaneously aborts the transaction.
TMAbort ==
        /\ tmState = "init"
        /\ tmState' = "aborted"
```

Leslie Lamport's Specification

Transaction Manager can decide to abort
(even when all RMs want to commit)

```
\* The TM spontaneously aborts the transaction.
TMAbort ==
        /\ tmState = "init"
        /\ tmState' = "aborted"
        /\ msgs' = msgs \cup {[type |-> "Abort"]}
```

Our 2PC Specification

Our 2PC Specification

Contributions:

Our 2PC Specification

Contributions:

- The Transaction Manager can no longer spontaneously abort transactions.

Our 2PC Specification

Contributions:

- The Transaction Manager can no longer spontaneously abort transactions.
- Resource Managers can crash.

Our 2PC Specification

Contributions:

- The Transaction Manager can no longer spontaneously abort transactions.
- Resource Managers can crash.
- Every **Atomic Commit** Property is checked.

Our 2PC Specification

The Transaction Manager can no longer spontaneously abort transactions:

Our 2PC Specification

The Transaction Manager can no longer spontaneously abort transactions:

```
\* A RM decides to abort.
RMChooseToAbort(rm) ==
        /\ rmState[rm] = "working"
```

Our 2PC Specification

The Transaction Manager can no longer spontaneously abort transactions:

```
\* A RM decides to abort.
RMChooseToAbort(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "aborted"]
    /\ msgs' = msgs \cup {[type |-> "Aborted", rm |-> rm]}
```

Our 2PC Specification

The Transaction Manager can no longer spontaneously abort transactions:

```
\* A RM decides to abort.
RMChooseToAbort(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "aborted"]
    /\ msgs' = msgs \cup {[type |-> "Aborted", rm |-> rm]}

\* The TM aborts the transaction, as requested by some RM.
TMAbort ==
```

Our 2PC Specification

The Transaction Manager can no longer spontaneously abort transactions:

```
\* A RM decides to abort.
RMChooseToAbort(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "aborted"]
    /\ msgs' = msgs \cup {[type |-> "Aborted", rm |-> rm]}

\* The TM aborts the transaction, as requested by some RM.
TMAbort ==
    /\ tmState = "init"
    /\ [type |-> "Aborted", rm |-> rm] \in msgs
```

Our 2PC Specification

The Transaction Manager can no longer spontaneously abort transactions:

```
\* A RM decides to abort.
RMChooseToAbort(rm) ==
    /\ rmState[rm] = "working"
    /\ rmState' = [rmState EXCEPT ![rm] = "aborted"]
    /\ msgs' = msgs \cup {[type |-> "Aborted", rm |-> rm]}

\* The TM aborts the transaction, as requested by some RM.
TMAbort ==
    /\ tmState = "init"
    /\ [type |-> "Aborted", rm |-> rm] \in msgs
    /\ tmState' = "aborted"
    /\ msgs' = msgs \cup {[type |-> "Abort"]}
```

Our 2PC Specification

Resource Managers can crash:

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
        /\ rmState[rm] /= "crashed"
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
    /\ rmState[rm] /= "crashed"
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = rmState[rm]]
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
    /\ rmState[rm] /= "crashed"
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = rmState[rm]]
    /\ rmState' = [rmState EXCEPT ![rm] = "crashed"]
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
    /\ rmState[rm] /= "crashed"
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = rmState[rm]]
    /\ rmState' = [rmState EXCEPT ![rm] = "crashed"]

\* A RM recovers from a crash.
RMRecover(rm) ==
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
    /\ rmState[rm] /= "crashed"
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = rmState[rm]]
    /\ rmState' = [rmState EXCEPT ![rm] = "crashed"]

\* A RM recovers from a crash.
RMRecover(rm) ==
    /\ rmState[rm] = "crashed"
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
    /\ rmState[rm] /= "crashed"
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = rmState[rm]]
    /\ rmState' = [rmState EXCEPT ![rm] = "crashed"]

\* A RM recovers from a crash.
RMRecover(rm) ==
    /\ rmState[rm] = "crashed"
    /\ rmState' = [rmState EXCEPT ![rm] = rmPrevState[rm]]
```

Our 2PC Specification

Resource Managers can crash:

```
\* A RM crashes.
RMCrash(rm) ==
    /\ rmState[rm] /= "crashed"
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = rmState[rm]]
    /\ rmState' = [rmState EXCEPT ![rm] = "crashed"]

\* A RM recovers from a crash.
RMRecover(rm) ==
    /\ rmState[rm] = "crashed"
    /\ rmState' = [rmState EXCEPT ![rm] = rmPrevState[rm]]
    /\ rmPrevState' = [rmPrevState EXCEPT ![rm] = "crashed"]
```

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Agreement: Any two processes that decide, decide the same value

Our 2PC Specification

Every Atomic Commit Property is checked:

Agreement: Any two processes that decide, decide the same value

```
TPAgreement ==
        \A rm1, rm2 \in RM :
            ~ /\ rmState[rm1] = "aborted"
              /\ rmState[rm2] = "committed"
```

Our 2PC Specification

Every Atomic Commit Property is checked:

Agreement: Any two processes that decide, decide the same value

```
TPAgreement ==
        \A rm1, rm2 \in RM :
            ~ /\ rmState[rm1] = "aborted"
              /\ rmState[rm2] = "committed"
```

Translation: No two processes can ever be in different decision states (i.e., decide different things).

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Our 2PC Specification

Every Atomic Commit Property is checked:

Validity (1): If some process starts with the value “no” then “abort” is the only possible decision

Our 2PC Specification

Every Atomic Commit Property is checked:

Validity (1): If some process starts with the value “no” then “abort” is the only possible decision

```
TPValidity1 ==
    (\E rm \in RM : rmState[rm] = "aborted") ~> (tmState = "aborted")
```

Our 2PC Specification

Every Atomic Commit Property is checked:

Validity (1): If some process starts with the value “no” then “abort” is the only possible decision

```
TPValidity1 ==
    (\E rm \in RM : rmState[rm] = "aborted") ~> (tmState = "aborted")
```

Translation: If any process is in aborted state (decided abort), then TM will eventually decide abort.

Our 2PC Specification

Every Atomic Commit Property is checked:

Our 2PC Specification

Every Atomic Commit Property is checked:

Validity (2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision

Our 2PC Specification

Every Atomic Commit Property is checked:

Validity (2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision

```
TPValidity2 ==
~ (\E rm \in RM : rmState[rm] /= "prepared") ~> (tmState = "committed")
```

Our 2PC Specification

Every Atomic Commit Property is checked:

Validity (2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision

```
TPValidity2 ==
~ (\E rm \in RM : rmState[rm] /= "prepared") ~> (tmState = "committed")
```

Translation: If all processes are prepared (which also means it is not crashed), then eventually TM will decide commit.

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Termination: If eventually all processes recover from all faults, then, eventually all processes decide

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Termination: If eventually all processes recover from all faults, then, eventually all processes decide

```
TPTermination ==
(~ (\E rm \in RM : rmState[rm] = "crashed")) ~>
      (~ (\E rm \in RM : rmState[rm] /= "aborted"   /\ rmPrevState[rm] /= "aborted"))
  \/ (~ (\E rm \in RM : rmState[rm] /= "committed" /\ rmPrevState[rm] /= "committed"))
```

Our 2PC Specification

Every **Atomic Commit** Property is checked:

Termination: If eventually all processes recover from all faults, then, eventually all processes decide

```
TPTermination ==
(~ (\E rm \in RM : rmState[rm] = "crashed")) ~>
      (~ (\E rm \in RM : rmState[rm] /= "aborted"   /\ rmPrevState[rm] /= "aborted"))
  \/ (~ (\E rm \in RM : rmState[rm] /= "committed" /\ rmPrevState[rm] /= "committed"))
```

Translation: If eventually no process is crashed, then eventually all will decide (the same thing). A process has decided if: it is in a decided state OR it was in a decided state (and is now crashed).

Our 2PC Specification

TLC reports no errors!

Time	Diameter	States Found	Distinct States	Queue Size
00:04:22	17	61 396	9 756	0

Our Specification

- Two Phase Commit
- Three Phase Commit

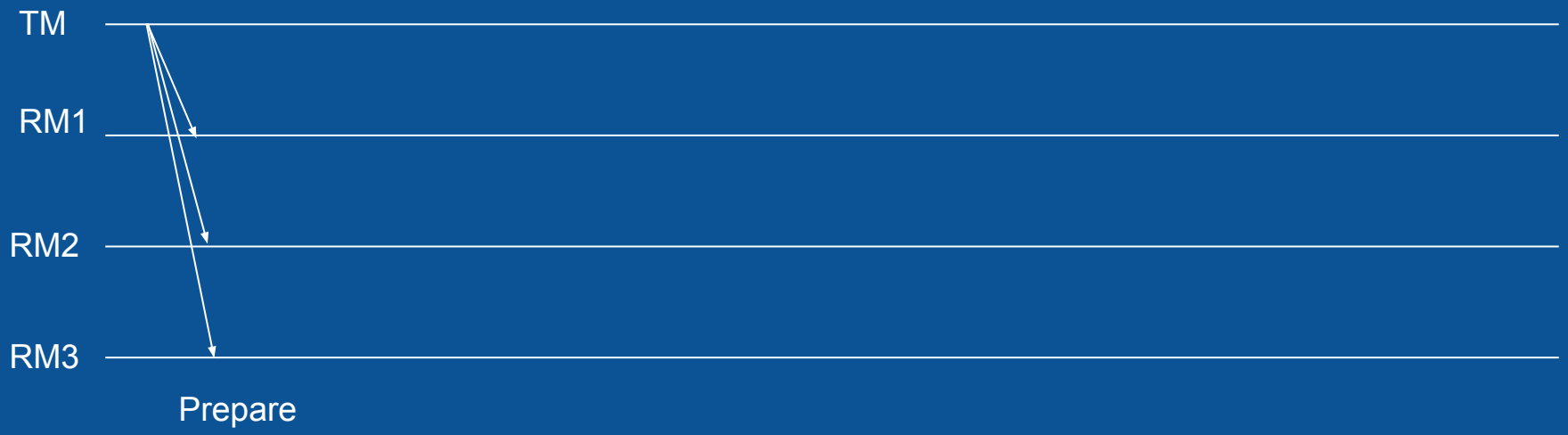
Our Specification

- Two Phase Commit
- Three Phase Commit

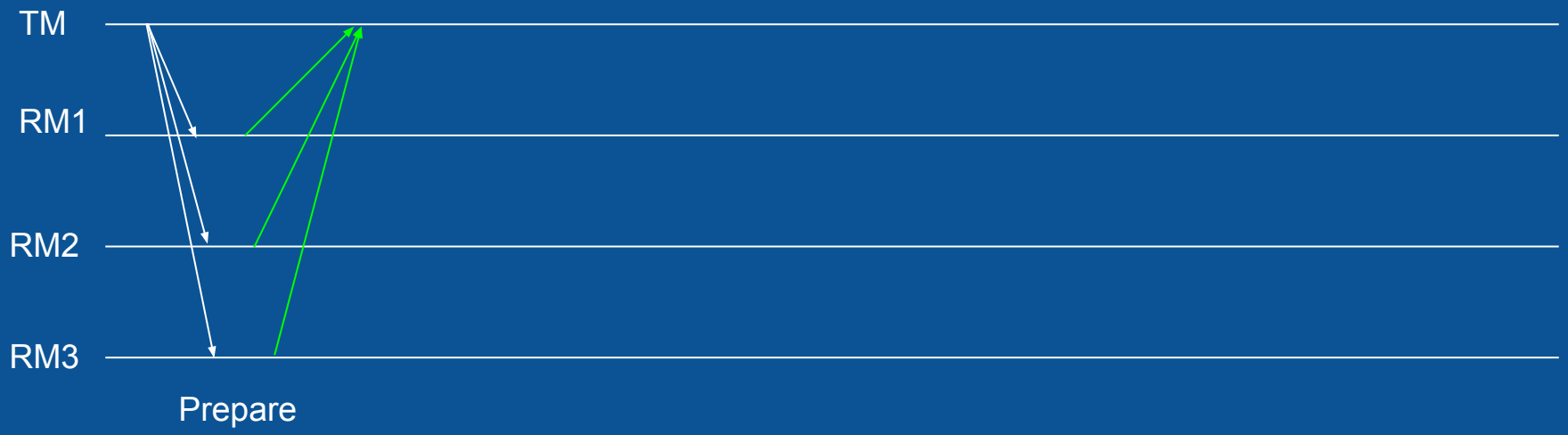
Three-Phase Commit

- Converts 2PC into a non blocking protocol
- Timeouts
- Adds Pre-commit round

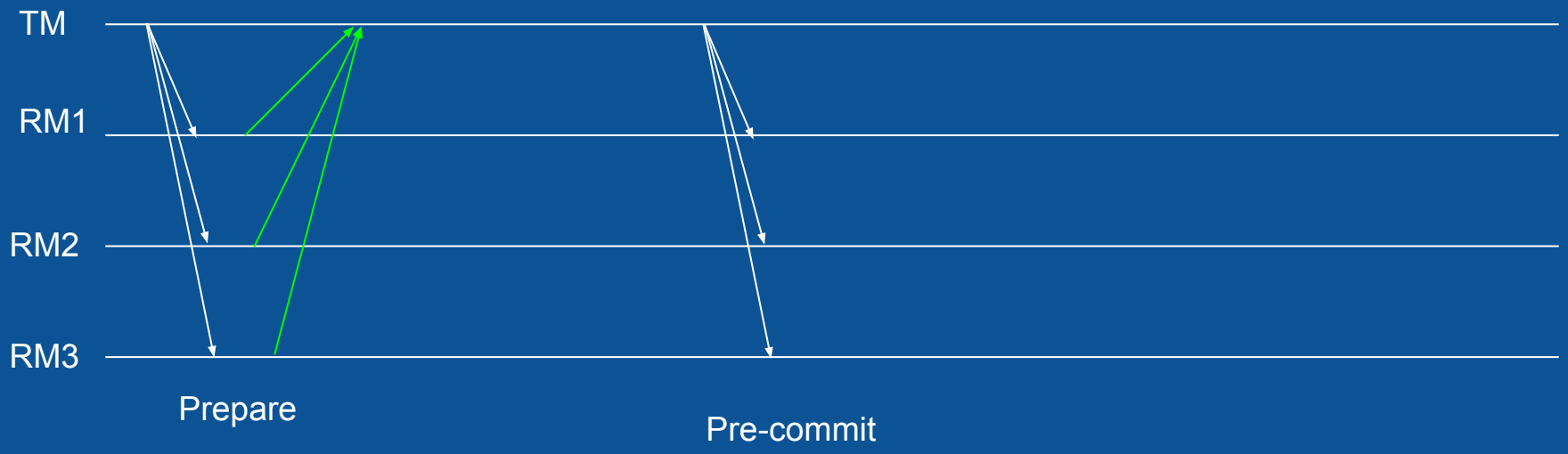
Three-Phase Commit



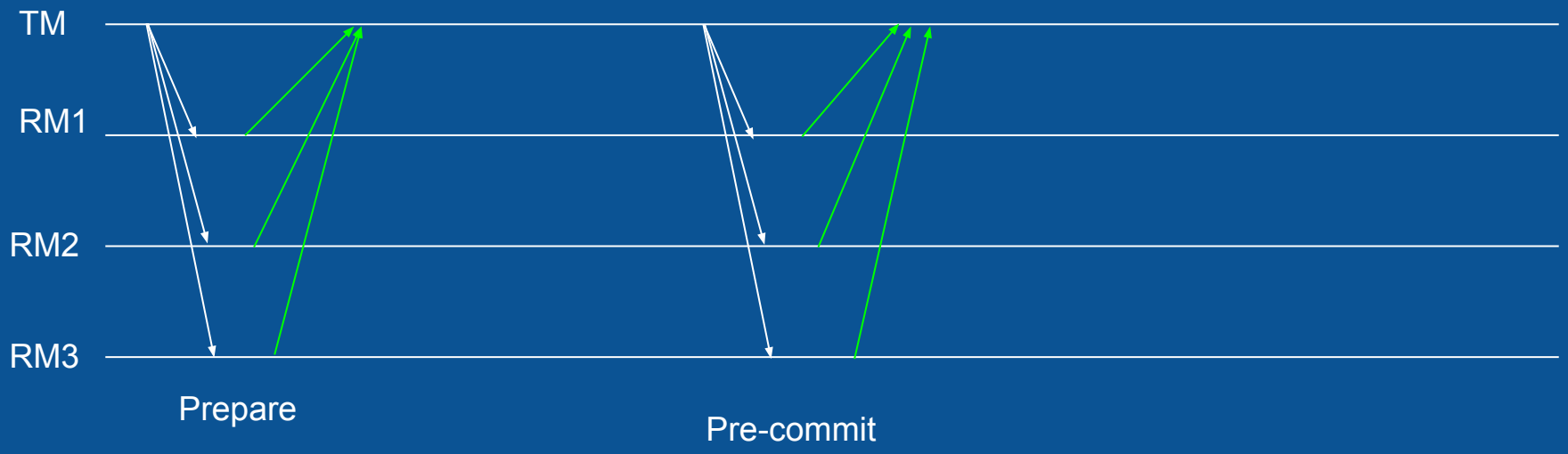
Three-Phase Commit



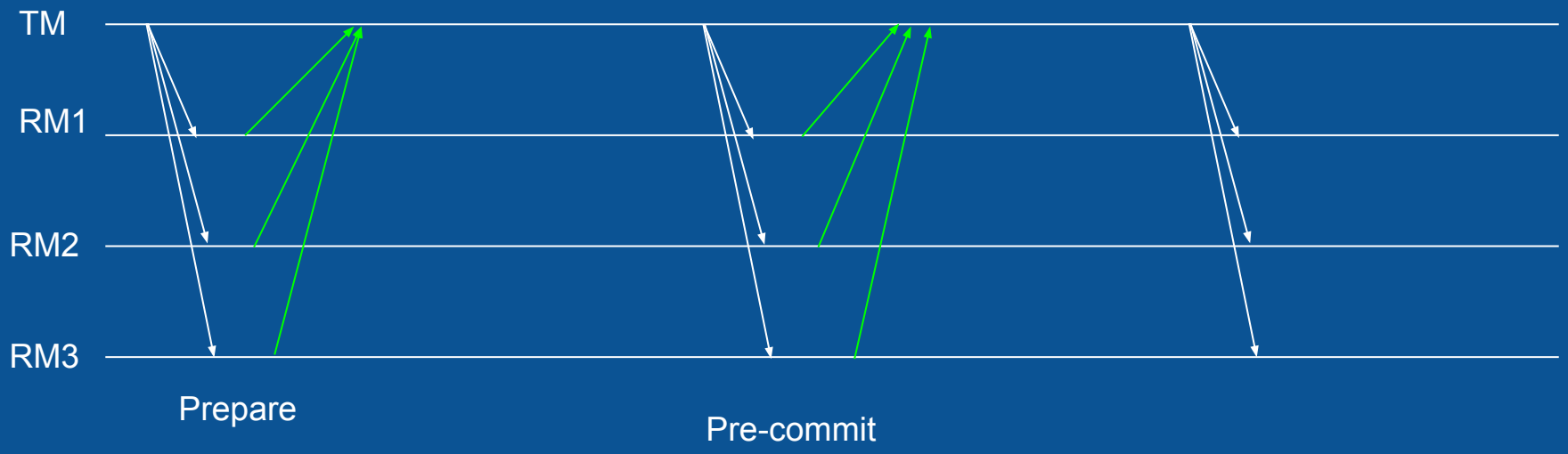
Three-Phase Commit



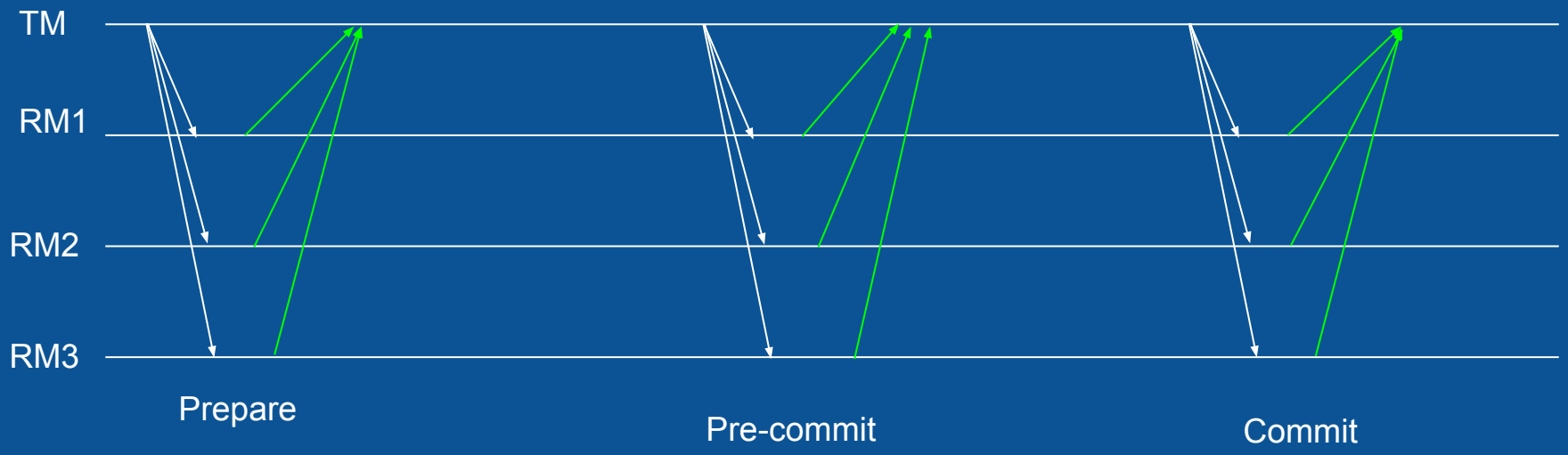
Three-Phase Commit



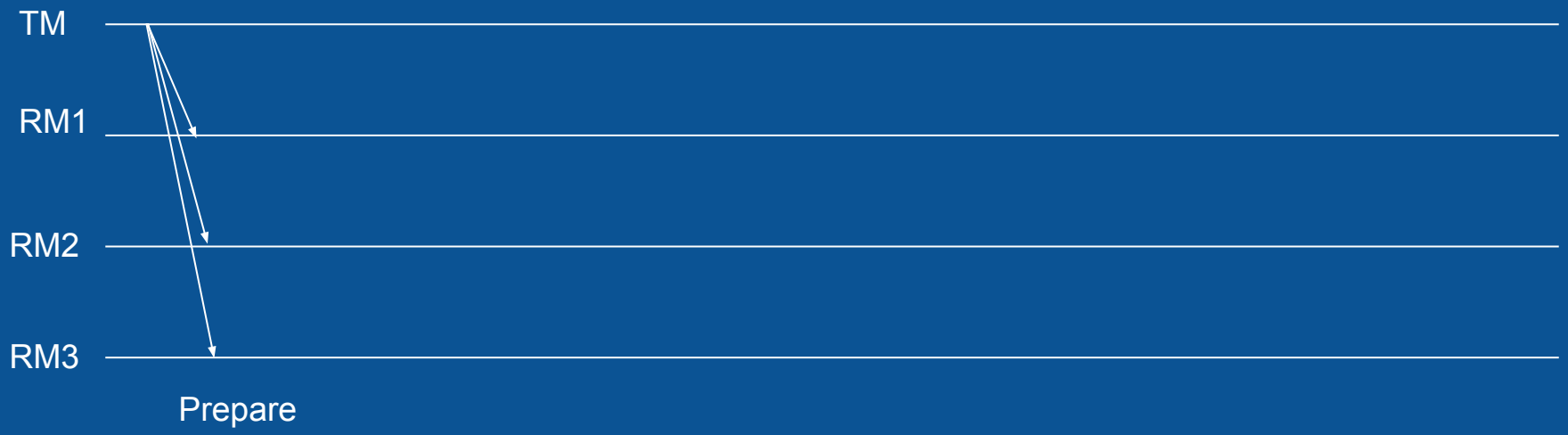
Three-Phase Commit



Three-Phase Commit

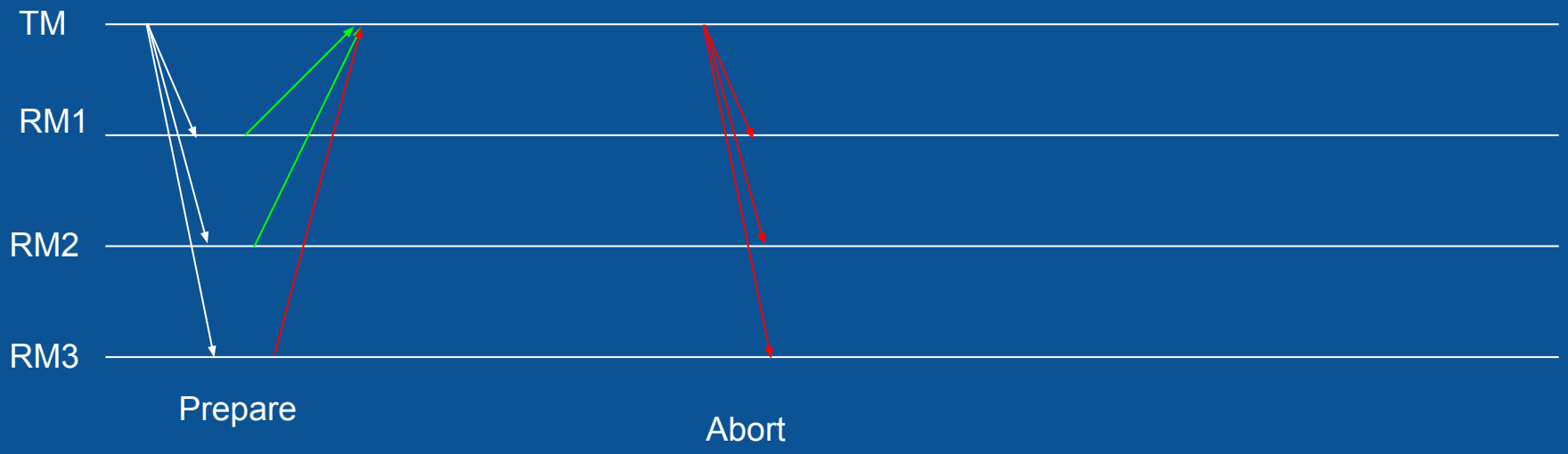


Three-Phase Commit

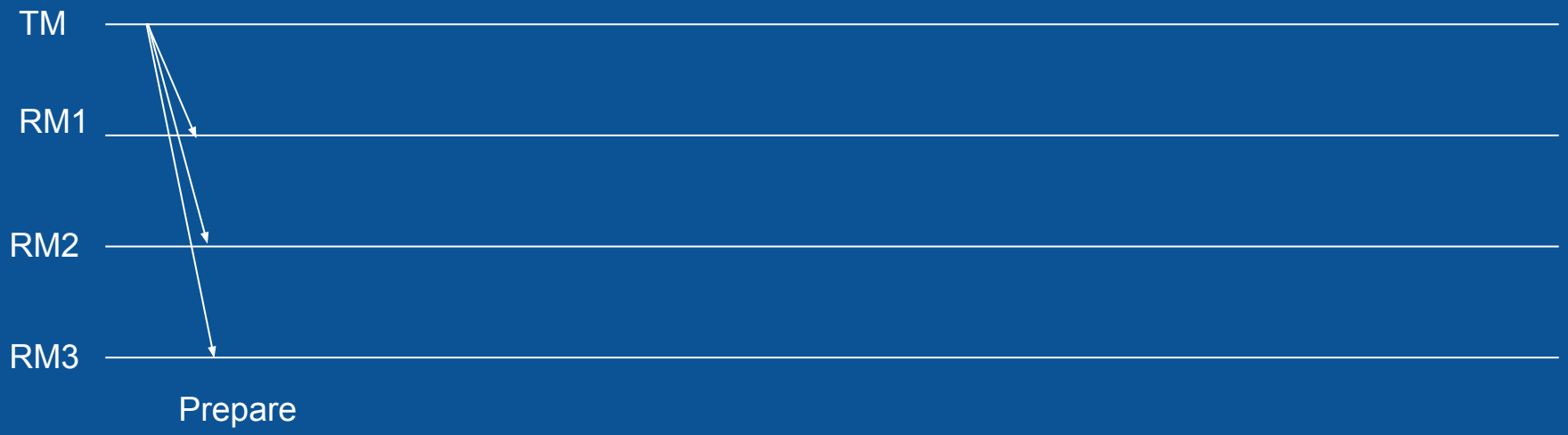


Three-Phase Commit

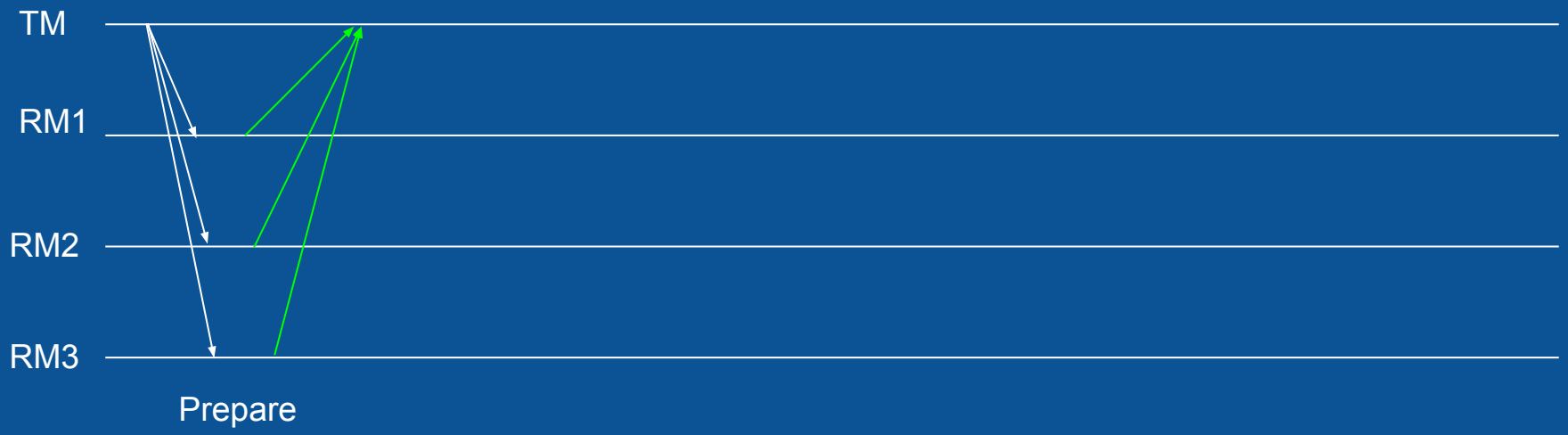
Three-Phase Commit



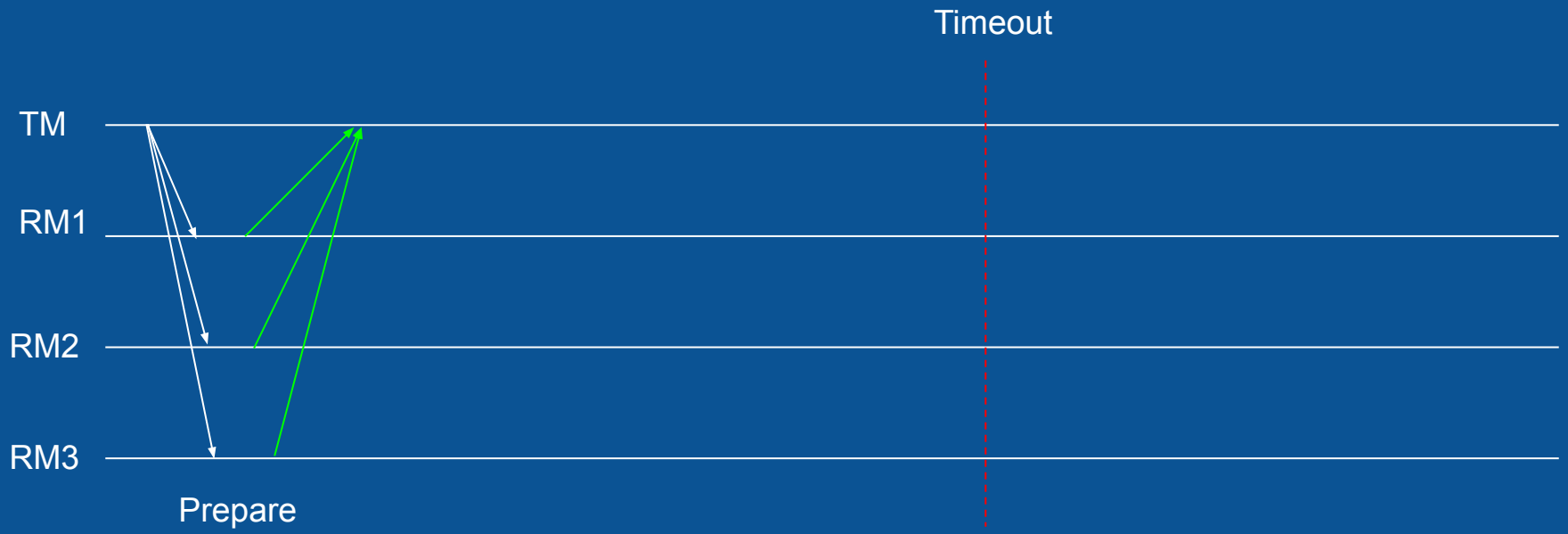
Three-Phase Commit



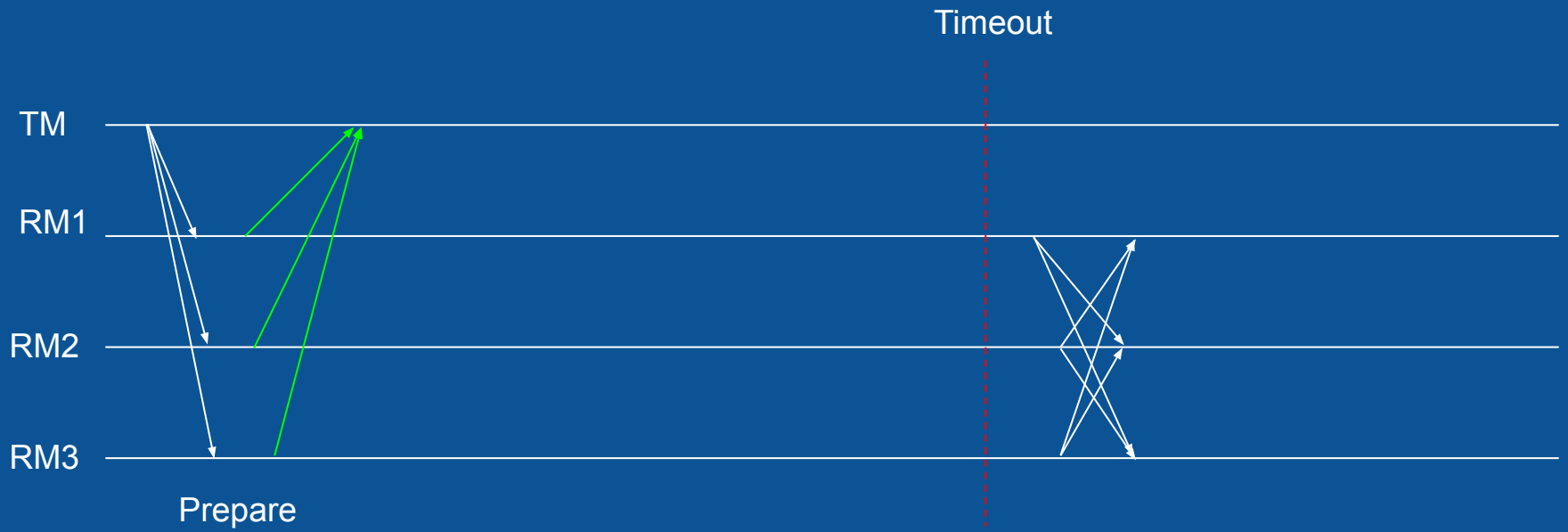
Three-Phase Commit



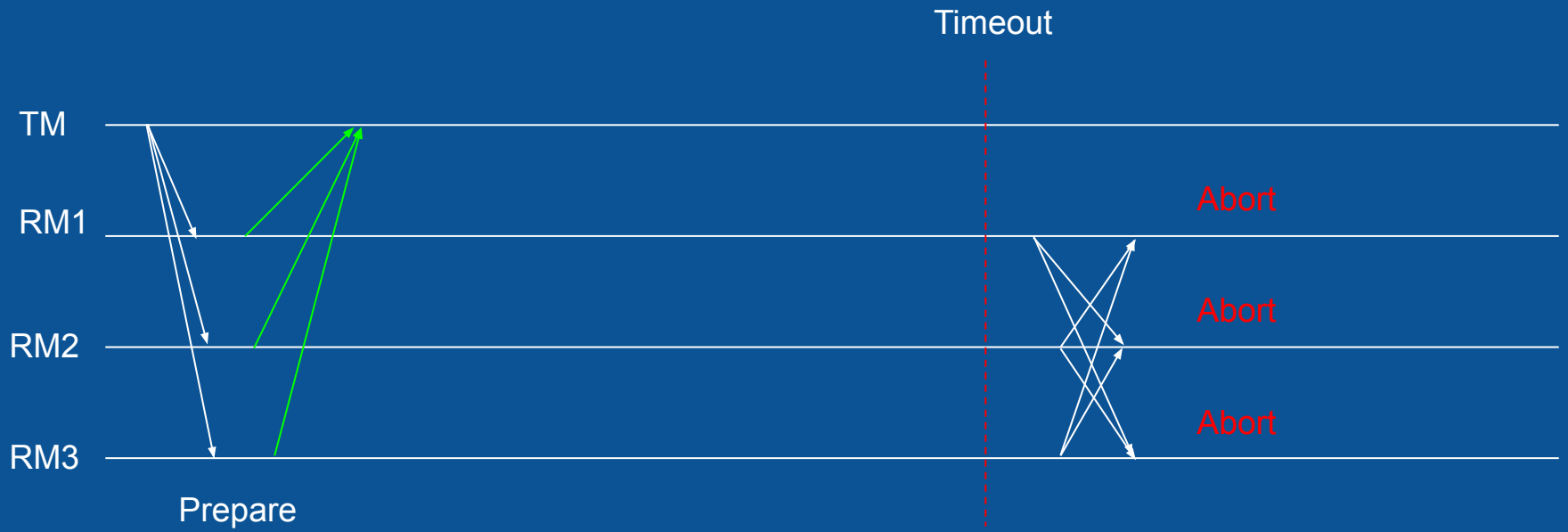
Three-Phase Commit



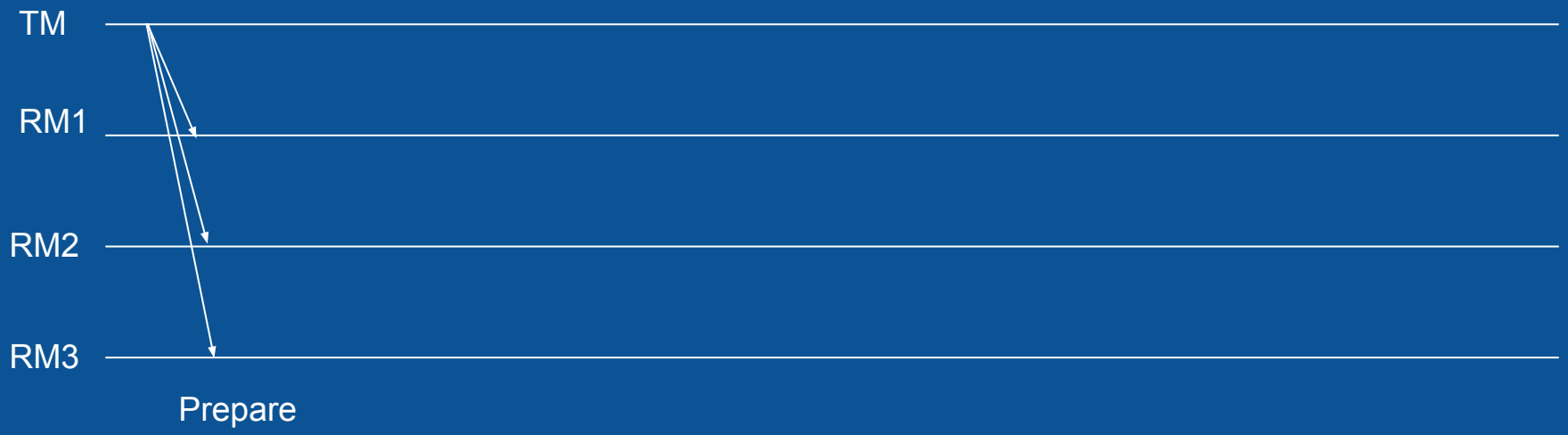
Three-Phase Commit



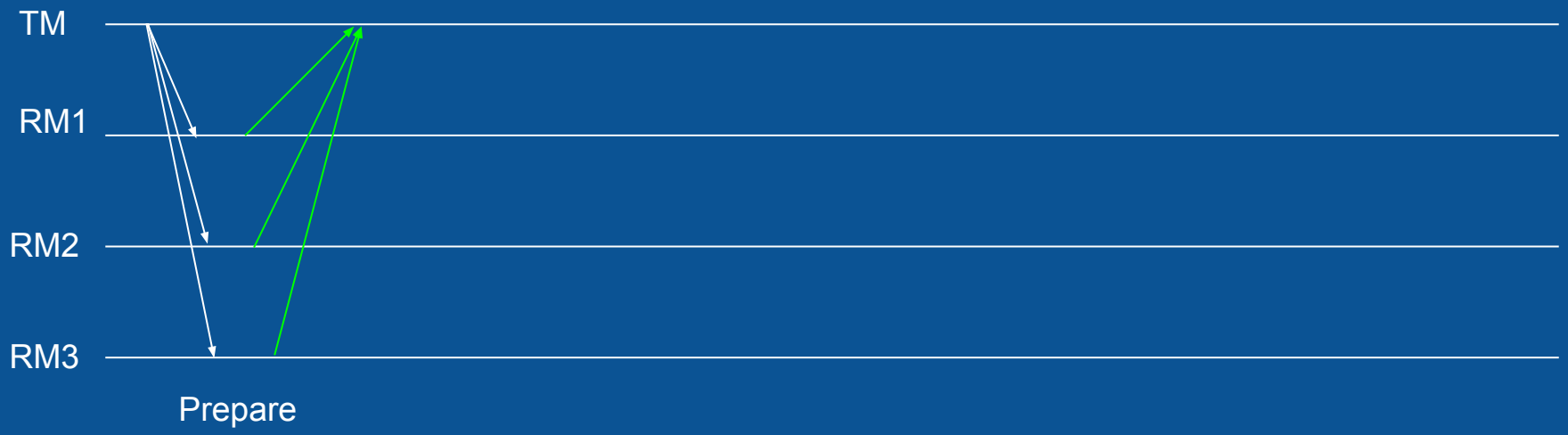
Three-Phase Commit



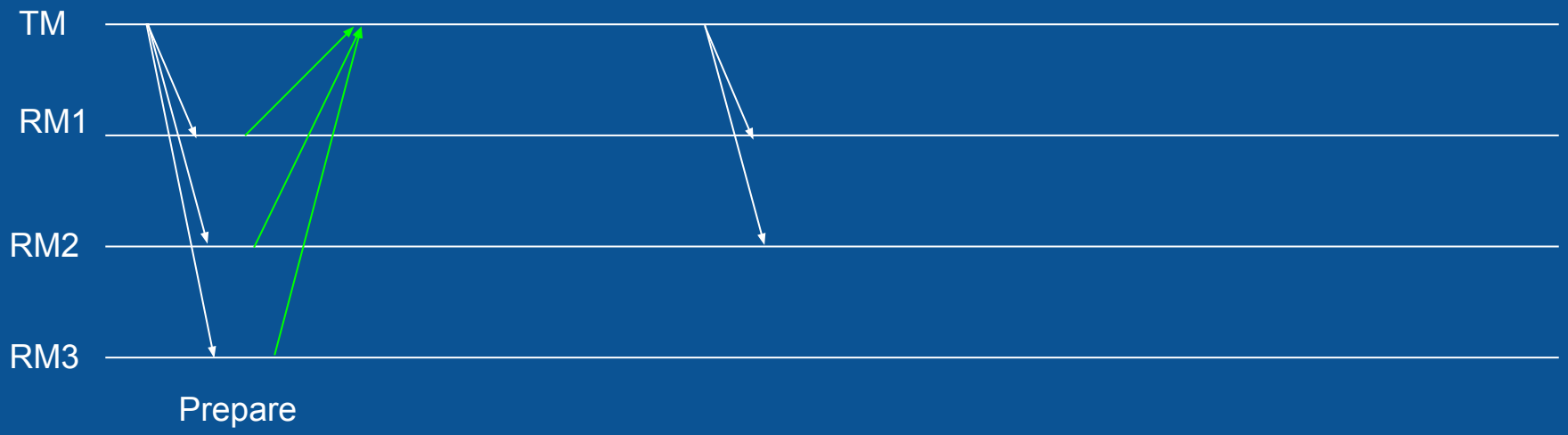
Three-Phase Commit



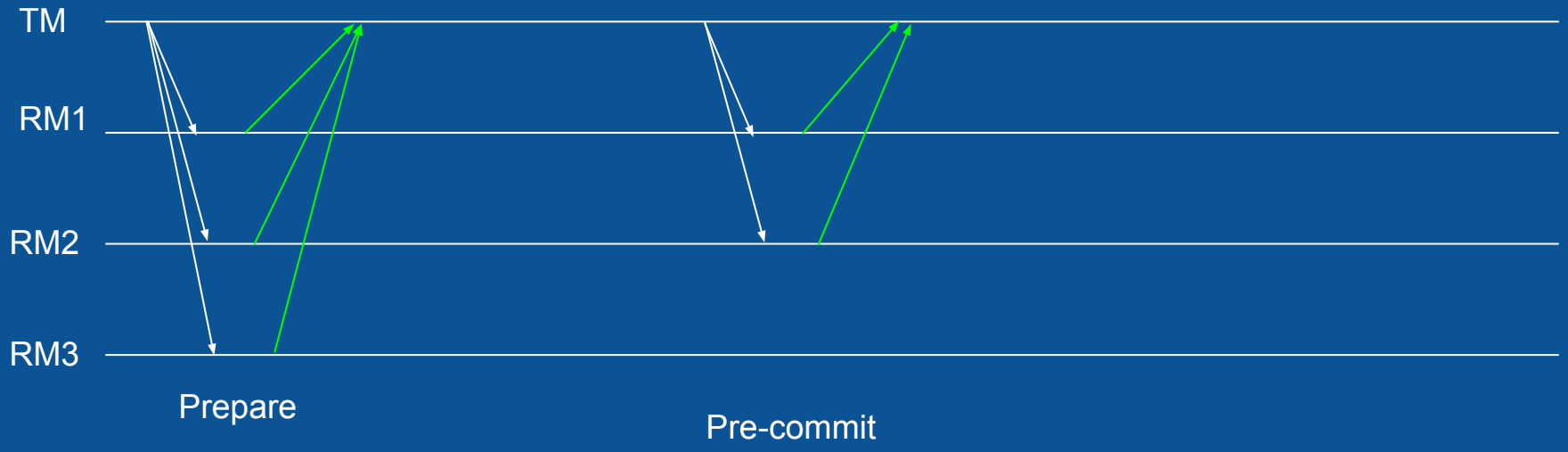
Three-Phase Commit



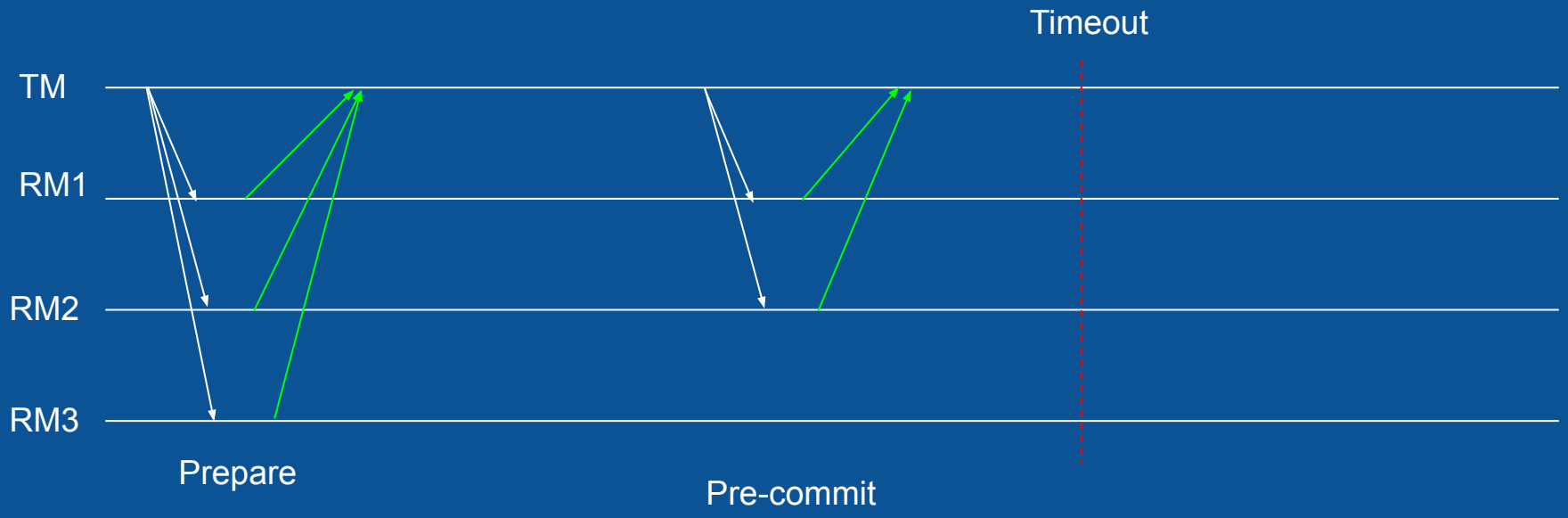
Three-Phase Commit



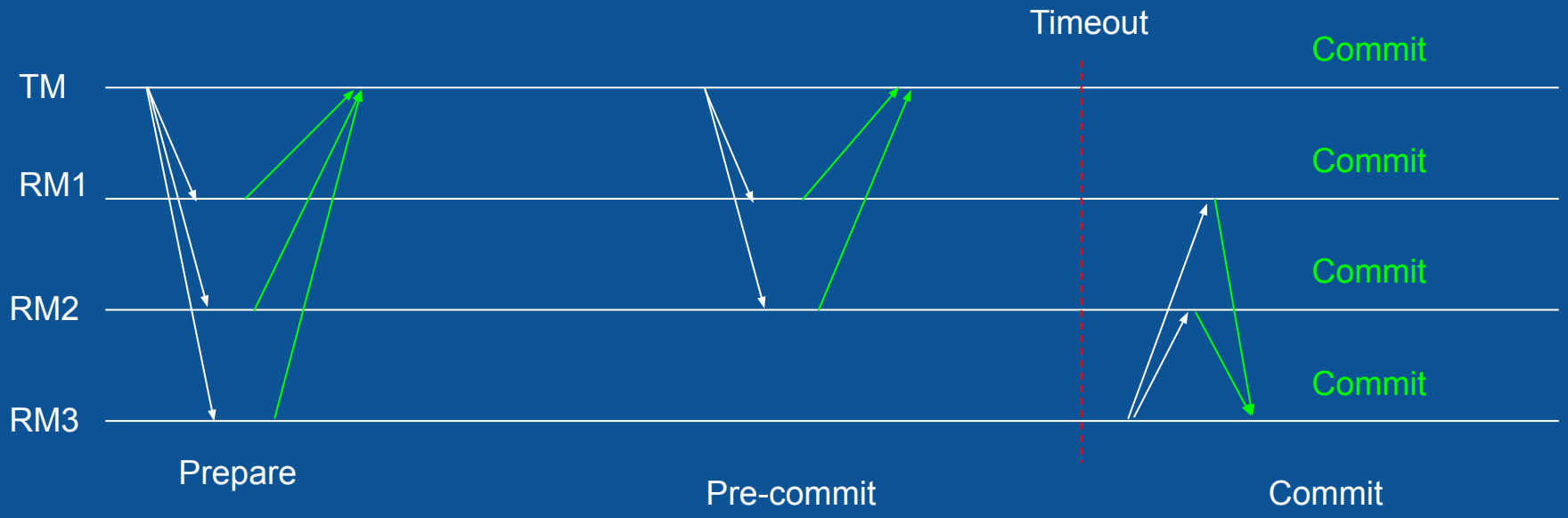
Three-Phase Commit



Three-Phase Commit



Three-Phase Commit



Three-Phase Commit

- Leslie Lamport 2 Phase Commit Specification
- Timeouts
- Pre-commit round

Three-Phase Commit

- Leslie Lamport 2 Phase Commit Specification
- Timeouts
- Pre-commit round

Three-Phase Commit

- Leslie Lamport 2 Phase Commit Specification
- Timeouts
- Pre-commit round

Three-Phase Commit

- Leslie Lamport 2 Phase Commit Specification
- Timeouts
- Pre-commit round

Specification of Atomic Commit

3 Phase Commit Specification:

Three-Phase Commit

```
CONSTANT RM \* The set of resource managers

VARIABLE
        rmState,       \* State of each resource manager
        tmState,       \* State of the transaction manager
        rmPrepared,    \* The set of RMs from which the TM has received prepared
messages
        rmPrecommitted, \* The set of RMs from which the TM has received
pre-committed messages
         msgs,         \* Set of all messages
         timeout       \* Timeout flag
```

Three-Phase Commit

```
\* All possible states the RMs and TM can assume
_WORKING == "working"
_PREPARED == "prepared"
_PREPARE == "prepare"
_PRECOMMIT == "pre-commit"
_PRECOMMITTED == "pre-committed"
_COMMIT == "commit"
_COMMITTED == "committed"
_ABORT == "abort"
_ABORTED == "aborted"
_INIT == "init"
```

Three-Phase Commit

```
\* The set of all possible messages.
Message == [ type: {_PREPARE, _PRECOMMIT, _COMMIT, _ABORT}] \cup
           [ type: {_PREPARED, _PRECOMMITTED, _ABORTED}, rm: RM ]
```

Three-Phase Commit

```
\* The set of all possible messages.
Message == [ type: {_PREPARE, _PRECOMMIT, _COMMIT, _ABORT}] \cup
           [ type: {_PREPARED, _PRECOMMITTED, _ABORTED}, rm: RM ]

\* Initial State of the Spec
Init ==
    /\ rmState = [r \in RM |-> _WORKING ]
    /\ tmState = _INIT
    /\ rmPrepared = {}
    /\ rmPrecommitted = {}
    /\ msgs = {}
    /\ timeout = "off"
```

Three-Phase Commit

```
\* TM receives a prepare message from r
TMRcvPrepare(r) ==
        /\ timeout = "off"
        /\ tmState = _INIT
        /\ [ type |-> _PREPARED, rm  |-> r ] \in msgs
        /\ r \notin rmPrepared
        /\ rmPrepared' = rmPrepared \cup {r}
        /\ UNCHANGED<<tmState, rmState, rmPrecommitted, msgs, timeout>>
```

Three-Phase Commit

```
\* TM receives a prepare message from r
TMRcvPrepare(r) ==
        /\ timeout = "off"
        /\ tmState = _INIT
        /\ [ type |-> _PREPARED, rm  |-> r ] \in msgs
        /\ r \notin rmPrepared
        /\ rmPrepared' = rmPrepared \cup {r}
        /\ UNCHANGED<<tmState, rmState, rmPrecommitted, msgs, timeout>>

\* RM decides to go into prepare state
RMPrepare(r) ==
   /\ timeout = "off"
   /\ rmState[r] = _WORKING
   /\ rmState' = [rmState EXCEPT ![r] = _PREPARED]
   /\ msgs' = msgs \cup {[ type |-> _PREPARED, rm |-> r ]}
   /\ UNCHANGED<<tmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* TM sends pre-commit message after receiving prepared message from RMs
TMPrecommit ==
        /\ timeout = "off"
        /\ tmState = _INIT
        /\ rmPrepared = RM
        /\ rmPrecommitted = {}
        /\ [ type |-> _PRECOMMIT ] \notin msgs
        /\ msgs' = msgs \cup {[ type |-> _PRECOMMIT ]}
        /\ UNCHANGED<<tmState, rmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* TM receives a precommit message from r
TMRcvPrecommit(r) ==
        /\ timeout = "off"
        /\ tmState = _INIT
        /\ [ type |-> _PRECOMMITTED, rm |-> r ] \in msgs
        /\ r \notin rmPrecommitted
        /\ rmPrecommitted' = rmPrecommitted \cup {r}
        /\ UNCHANGED<<tmState, rmState, rmPrepared, msgs, timeout>>
```

Three-Phase Commit

```
\* TM receives a precommit message from r
TMRcvPrecommit(r) ==
        /\ timeout = "off"
        /\ tmState = _INIT
        /\ [ type |-> _PRECOMMITTED, rm |-> r ] \in msgs
        /\ r \notin rmPrecommitted
        /\ rmPrecommitted' = rmPrecommitted \cup {r}
        /\ UNCHANGED<<tmState, rmState, rmPrepared, msgs, timeout>>

\* RM decides to pre-commit after seeing a pre-commit message from TM
RMPrecommit(r) ==
   /\ timeout = "off"
   /\ rmState[r] = _PREPARED
   /\ [ type |-> _PRECOMMIT ] \in msgs
   /\ rmState' = [ rmState EXCEPT ![r] = _PRECOMMITTED ]
   /\ msgs' = msgs \cup {[ type |-> _PRECOMMITTED, rm |-> r ]}
   /\ UNCHANGED<<tmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* TM changes to the commit state and sends message to commit
TMCommit ==
   /\ timeout = "off"
   /\ tmState = _INIT
   /\ rmPrecommitted = RM
   /\ tmState' = _COMMITTED
   /\ msgs' = msgs \cup {[ type |-> _COMMIT ]}
   /\ UNCHANGED<<rmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* TM changes to the commit state and sends message to commit
TMCommit ==
   /\ timeout = "off"
   /\ tmState = _INIT
   /\ rmPrecommitted = RM
   /\ tmState' = _COMMITTED
   /\ msgs' = msgs \cup {[ type |-> _COMMIT ]}
   /\ UNCHANGED<<rmState, rmPrepared, rmPrecommitted, timeout>>

\* RM receives commit message and changes state to commit
RMCommit(r) ==
   /\ timeout = "off"
   /\ rmState[r] = _PRECOMMITTED
   /\ [ type |-> _COMMIT ] \in msgs
   /\ rmState' = [ rmState EXCEPT ![r] = _COMMITTED ]
   /\ UNCHANGED<<tmState, rmPrepared, rmPrecommitted, msgs, timeout>>
```

Three-Phase Commit

```
\* TM decides to abort the transaction
TMAbort ==
   /\ timeout = "off"
   /\ rmPrecommitted = {}
   /\ ~ ([ type |-> _PRECOMMIT ] \in msgs \/ [ type |-> _COMMIT ] \in msgs)
   /\ tmState = _INIT
   /\ tmState' = _ABORTED
   /\ msgs' = msgs \cup {[type |-> _ABORT]}
   /\ UNCHANGED <<rmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* TM decides to abort the transaction
TMAbort ==
   /\ timeout = "off"
   /\ rmPrecommitted = {}
   /\ ~ ([ type |-> _PRECOMMIT ] \in msgs \/ [ type |-> _COMMIT ] \in msgs)
   /\ tmState = _INIT
   /\ tmState' = _ABORTED
   /\ msgs' = msgs \cup {[type |-> _ABORT]}
   /\ UNCHANGED <<rmState, rmPrepared, rmPrecommitted, timeout>>

\* TM decides to abort the transaction after seeing an aborted message from an RM and sends abort message
TMRcvAbort(r) ==
   /\ timeout = "off"
   /\ tmState = _INIT
   /\ [type |-> _ABORTED, rm |-> r] \in msgs
   /\ rmPrecommitted = {}
   /\ tmState' = _ABORTED
   /\ msgs' = msgs \cup {[type |-> _ABORT]}
   /\ UNCHANGED <<rmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* RM Chooses to abort during the prepare phase
RMChooseToAbort(r) ==
   /\ timeout = "off"
   /\ rmState[r] = _WORKING
   /\ rmState' = [rmState EXCEPT ![r] = _ABORTED]
   /\ r \notin rmPrepared
   /\ msgs' = msgs \cup {[type |-> _ABORTED, rm |-> r ]}
   /\ UNCHANGED <<tmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
\* RM Chooses to abort during the prepare phase
RMChooseToAbort(r) ==
    /\ timeout = "off"
    /\ rmState[r] = _WORKING
    /\ rmState' = [rmState EXCEPT ![r] = _ABORTED]
    /\ r \notin rmPrepared
    /\ msgs' = msgs \cup {[type |-> _ABORTED, rm |-> r ]}
    /\ UNCHANGED <<tmState, rmPrepared, rmPrecommitted, timeout>>

\* RM decides to abort after seeing the abort message from TM
RMRcvAbortMsg(r) ==
    /\ timeout = "off"
    /\ rmState[r] = _PREPARED \/ rmState[r] = _WORKING
    /\ [type |-> _ABORT] \in msgs
    /\ rmState' = [rmState EXCEPT ![r] = _ABORTED]
    /\ UNCHANGED <<tmState, rmPrepared, rmPrecommitted, msgs, timeout>>
```

Three-Phase Commit

```
Timeout ==
   /\ ~ tmState = _COMMITTED
   /\ timeout = "off"
   /\ timeout' = "on"
   /\ UNCHANGED<<msgs, rmState, tmState, rmPrepared, rmPrecommitted>>
```

Three-Phase Commit

```
RMWhenTimeout(r) ==
   /\ timeout = "on"
   /\ rmState[r] \notin {_ABORTED, _COMMITTED}
   /\ IF rmPrecommitted # {}
         THEN
            /\ rmState' = [rmState EXCEPT ![r] = _COMMITTED]
         ELSE
            /\ rmState' = [rmState EXCEPT ![r] = _ABORTED]
   /\ UNCHANGED<<msgs,tmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

```
RMWhenTimeout(r) ==
   /\ timeout = "on"
   /\ rmState[r] \notin {_ABORTED, _COMMITTED}
   /\ IF rmPrecommitted # {}
            THEN
               /\ rmState' = [rmState EXCEPT ![r] = _COMMITTED]
            ELSE
               /\ rmState' = [rmState EXCEPT ![r] = _ABORTED]
   /\ UNCHANGED<<msgs,tmState, rmPrepared, rmPrecommitted, timeout>>

TMWhenTimeout ==
   /\ timeout = "on"
   /\ tmState = _INIT
   /\  IF rmPrecommitted # {} /\ ~ [ type |-> _ABORT ] \in msgs
      THEN
         /\ tmState' = _COMMITTED
      ELSE
         /\ tmState' = _ABORTED
   /\ UNCHANGED<<msgs,rmState, rmPrepared, rmPrecommitted, timeout>>
```

Three-Phase Commit

- **Remember:**

```
VARIABLE
    rmState,        \* State of each resource manager
    tmState,        \* State of the transaction manager
    rmPrepared,     \* The set of RMs from which the TM has received prepared messages
    rmPrecommitted, \* The set of RMs from which the TM has received pre-committed messages
     msgs,          \* Set of all messages
     timeout        \* Timeout flag
```

Three-Phase Commit

- **Remember:**

```
VARIABLE
        rmState,       \* State of each resource manager
        tmState,       \* State of the transaction manager
        rmPrepared,    \* The set of RMs from which the TM has received prepared messages
        rmPrecommitted, \* The set of RMs from which the TM has received pre-committed messages
         msgs,         \* Set of all messages
         timeout       \* Timeout flag

TypeOK ==
        /\ rmState \in [ RM -> { _WORKING, _PREPARED, _PRECOMMITTED, _COMMITTED, _ABORTED } ]
        /\ tmState \in { _INIT, _COMMITTED, _ABORTED }
        /\ rmPrepared \subseteq RM
        /\ rmPrecommitted \subseteq RM
        /\ msgs \subseteq Message
        /\ timeout \in {"on", "off"}
```

Three-Phase Commit

```
Agreement ==
        \A rm1, rm2 \in RM : ~ /\ rmState[rm1] = _ABORTED
                               /\ rmState[rm2] = _COMMITTED

Validity ==
        \E r \in RM: rmState[r] = _ABORTED ~> tmState = _ABORTED

Fairness ==
                /\ WF_vars(TMCommit)
                /\ WF_vars(TMAbort)
                /\ WF_vars(TMPrecommit)
                /\ \A r \in RM: WF_vars(TMRcvPrecommit(r))
                /\ \A r \in RM: WF_vars(TMRcvPrepare(r))
                /\ \A r \in RM: WF_vars(TMRcvAbort(r))
                /\ \A r \in RM: WF_vars(TMWhenTimeout)
                /\ \A r \in RM: WF_vars(RMWhenTimeout(r))
```

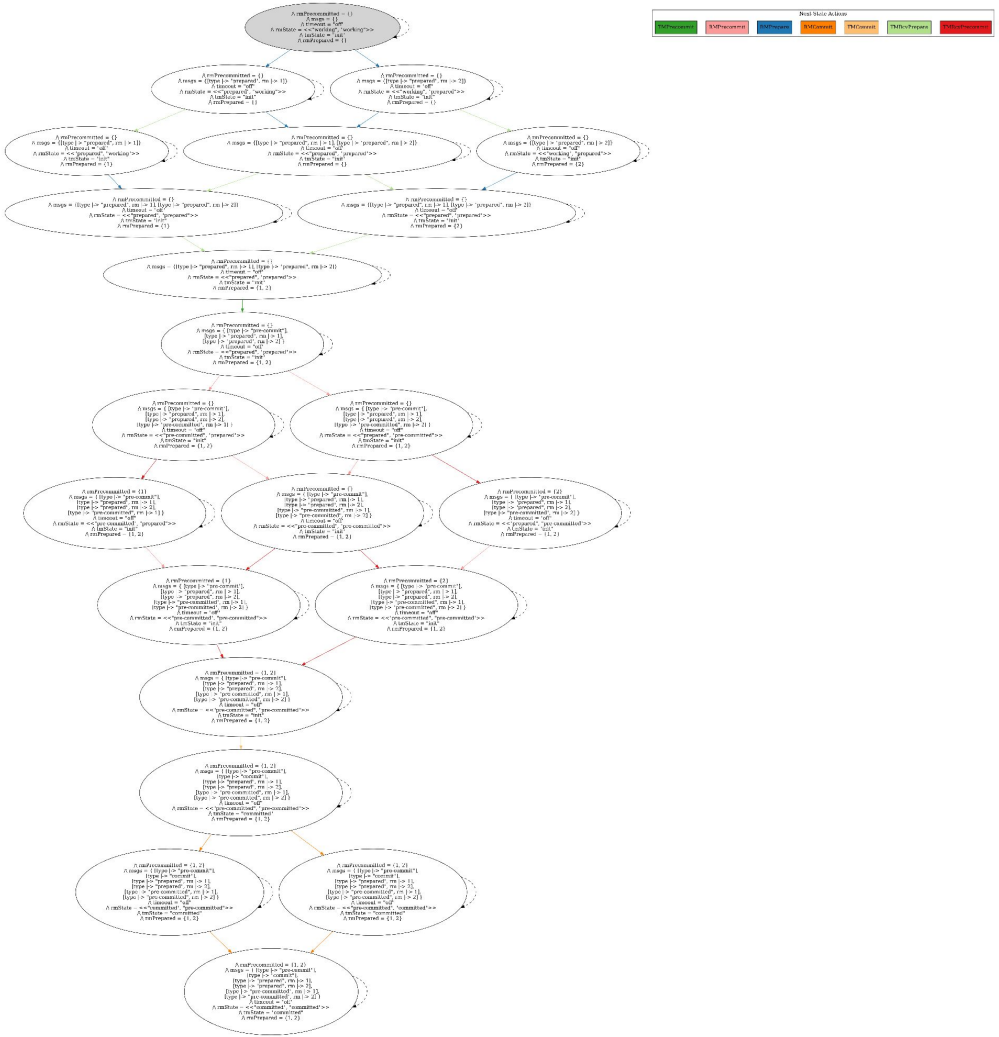
Three-Phase Commit

```
Next ==
    \/ TMPrecommit
    \/ TMCommit
    \/ TMAbort
    \/ Timeout
    \/ TMWhenTimeout
    \/ \E r \in RM:
        \/ TMRcvPrepare(r)
        \/ TMRcvPrecommit(r)
        \/ TMRcvAbort(r)
        \/ RMPrepare(r)
        \/ RMPrecommit(r)
        \/ RMCommit(r)
        \/ RMWhenTimeout(r)
        \/ RMChooseToAbort(r)
        \/ RMRcvAbortMsg(r)
```

Our 3PC Specification

TLC reports no errors!

Time	Diameter	States Found	Distinct States	Queue Size
00:00:48	29	256,756	84,111	0
00:00:05	8	42,827	16,907	8,530
00:00:01	0	1	1	1


Real State Actions
TMPrecommit
TMPrecommit
TMPrepare
TMCommit
TMCommit
TMNoPrepare
TMNoPrecommit

Evaluation Questions

Conclusion

- TLA+ is a useful tool to verify properties of an algorithm
- Complex systems are hard to specify
- Only validates the provided specification

Conclusion

- TLA+ is a useful tool to verify properties of an algorithm
- Complex systems are hard to specify
- Only validates the provided specification

Conclusion

- TLA+ is a useful tool to verify properties of an algorithm
- Complex systems are hard to specify
- Only validates the provided specification

Conclusion

- TLA+ is a useful tool to verify properties of an algorithm
- Complex systems are hard to specify
- Only validates the provided specification

Future Work

Future Work

Two-Phase Commit

- Allow Transaction manager to crash
- Model message loss / Network partitions

Future Work

Two-Phase Commit

- Allow Transaction manager to crash
- Model message loss / Network partitions

Future Work

Two-Phase Commit

- Allow Transaction manager to crash
- Model message loss / Network partitions

Future Work

Three-Phase Commit

- Verify Validity
- Verify Termination
- Model Network Partitions
- Model Process crashes

Future Work

Three-Phase Commit

- Verify Validity
- Verify Termination
- Model Network Partitions
- Model Process crashes

Future Work

Three-Phase Commit

- Verify Validity
- Verify Termination
- Model Network Partitions
- Model Process crashes

Future Work

Three-Phase Commit

- Verify Validity
- Verify Termination
- Model Network Partitions
- Model Process crashes

Thank you!

Thank you!

Questions?