Two-Phase Commit

The TLA+ Hyperbook

Wang Zhifu 151220117

April 13, 2018

Table of contents

Preliminaries

Records

The definition

$$r \triangleq [prof \mapsto \text{“}Fred\text{”}, num \mapsto 42]$$

defines r to be a record with two fields *prof* and *num*.

The values of its two fields are

$$r.prof = \text{“}Fred\text{”} \text{ and } r.num = 42$$

A record corresponds roughly to a STRUCT in C, except that changing the orders of the fields makes no difference.

$$[prof \mapsto \text{“}Fred\text{”}, num \mapsto 42] = [num \mapsto 42, prof \mapsto \text{“}Fred\text{”}]$$

Records

The definition

$$r \triangleq [prof \mapsto \text{“}Fred\text{”}, num \mapsto 42]$$

defines r to be a record with two fields *prof* and *num*.

The values of its two fields are

$$r.prof = \text{“}Fred\text{”} \text{ and } r.num = 42$$

A record corresponds roughly to a STRUCT in C, except that changing the orders of the fields makes no difference.

$$[prof \mapsto \text{“}Fred\text{”}, num \mapsto 42] = [num \mapsto 42, prof \mapsto \text{“}Fred\text{”}]$$

Records

The definition

$$r \triangleq [\mathit{prof} \mapsto \text{“}\mathit{Fred}\text{”}, \mathit{num} \mapsto 42]$$

defines r to be a record with two fields *prof* and *num*.

The values of its two fields are

$$r.\mathit{prof} = \text{“}\mathit{Fred}\text{”} \text{ and } r.\mathit{num} = 42$$

A record corresponds roughly to a STRUCT in C, except that changing the orders of the fields makes no difference.

$$[\mathit{prof} \mapsto \text{“}\mathit{Fred}\text{”}, \mathit{num} \mapsto 42] = [\mathit{num} \mapsto 42, \mathit{prof} \mapsto \text{“}\mathit{Fred}\text{”}]$$

Records

$[prof : \{\text{“}Fred\text{”}, \text{“}Ted\text{”}, \text{“}Ned\text{”}\}, num : 0..99]$ is the set of all records.

$[prof \mapsto \ldots, num \mapsto \ldots]$ with $prof$ field in $\{\text{“}Fred\text{”}, \text{“}Ted\text{”}, \text{“}Ned\text{”}\}$ and num field in $0..99$.

So $[prof \mapsto \text{“}Ned\text{”}, num \mapsto 24]$ is in this set.

Records

$[prof : \{\text{“}Fred\text{”}, \text{“}Ted\text{”}, \text{“}Ned\text{”}\}, num : 0..99]$ is the set of all records.

$[prof \mapsto \ldots, num \mapsto \ldots]$ with $prof$ field in $\{\text{“}Fred\text{”}, \text{“}Ted\text{”}, \text{“}Ned\text{”}\}$ and num field in $0..99$.

So $[prof \mapsto \text{“}Ned\text{”}, num \mapsto 24]$ is in this set.

Records

$[prof \mapsto \text{“}Fred\text{”}, num \mapsto 42]$ is a function f with domain $\text{“}prof\text{”}, \text{“}num\text{”}$ s.t. $f[\text{“}prof\text{”}] = \text{“}Fred\text{”}$ and $f[\text{“}num\text{”}] = 42$.

$f.prof$ is an abbreviation for $f[\text{“}prof\text{”}]$

$$[f \text{ EXCEPT } ![\text{“}prof\text{”}] = \text{“}Red\text{”}]$$

can be abbreviated as

$$[f \text{ EXCEPT } !.prof = \text{“}Red\text{”}]$$

Records

$[prof \mapsto \text{“}Fred\text{”}, num \mapsto 42]$ is a function f with domain $\text{“}prof\text{”}, \text{“}num\text{”}$ s.t. $f[\text{“}prof\text{”}] = \text{“}Fred\text{”}$ and $f[\text{“}num\text{”}] = 42$.

$f.prof$ is an abbreviation for $f[\text{“}prof\text{”}]$

$$[f \text{ EXCEPT } ![\text{“}prof\text{”}] = \text{“}Red\text{”}]$$

can be abbreviated as

$$[f \text{ EXCEPT } !.prof = \text{“}Red\text{”}]$$

Records

$[prof \mapsto \text{“}Fred\text{”}, num \mapsto 42]$ is a function f with domain $\text{“}prof\text{”}, \text{“}num\text{”}$ s.t. $f[\text{“}prof\text{”}] = \text{“}Fred\text{”}$ and $f[\text{“}num\text{”}] = 42$.

$f.prof$ is an abbreviation for $f[\text{“}prof\text{”}]$

$$[f \text{ EXCEPT } ![\text{“}prof\text{”}] = \text{“}Red\text{”}]$$

can be abbreviated as

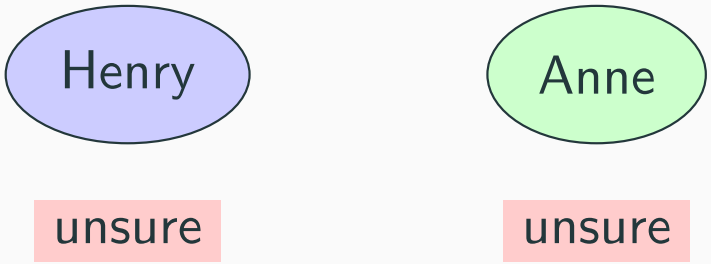
$$[f \text{ EXCEPT } !.prof = \text{“}Red\text{”}]$$

Weddings

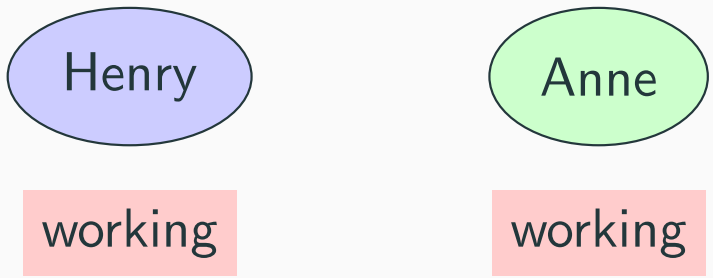
- What Transaction Commit Describes
- TwoPhase Commit Adds the Minister
- A Really Modern Wedding
- A Simplification

What Transaction Commit Describes

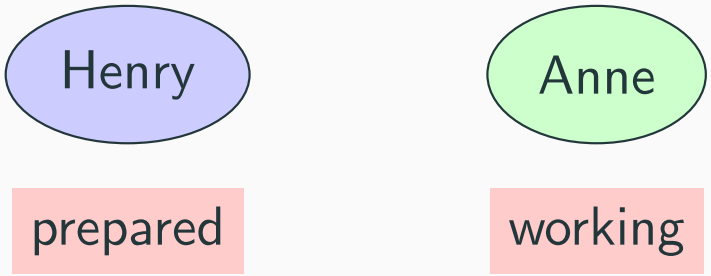
What Transaction Commit Describes



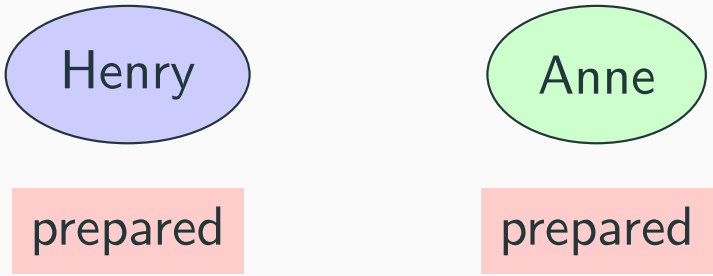
What Transaction Commit Describes



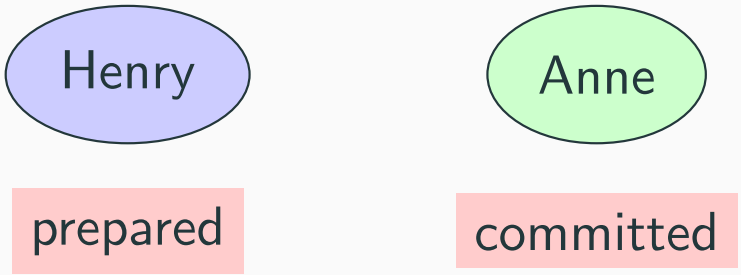
What Transaction Commit Describes



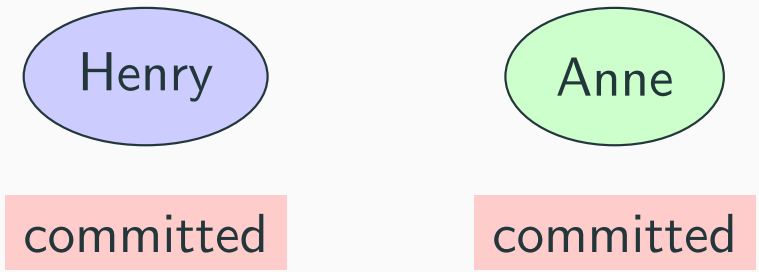
What Transaction Commit Describes



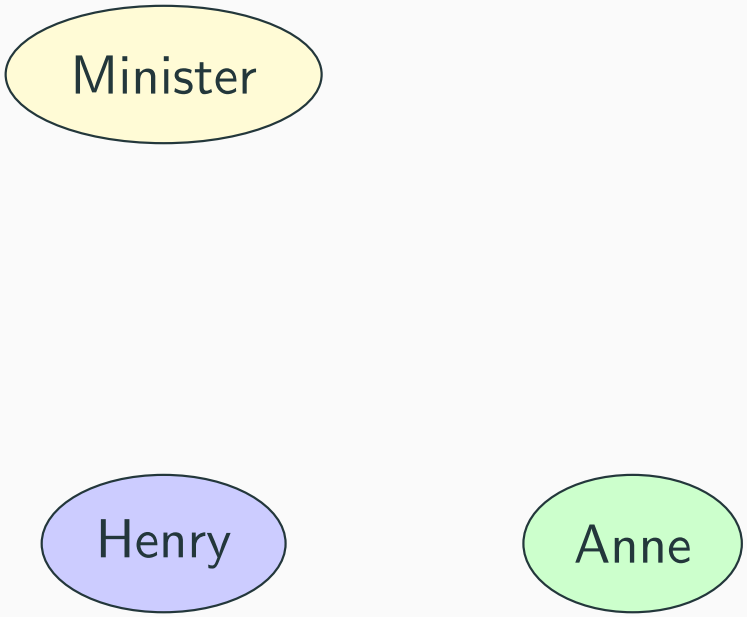
What Transaction Commit Describes



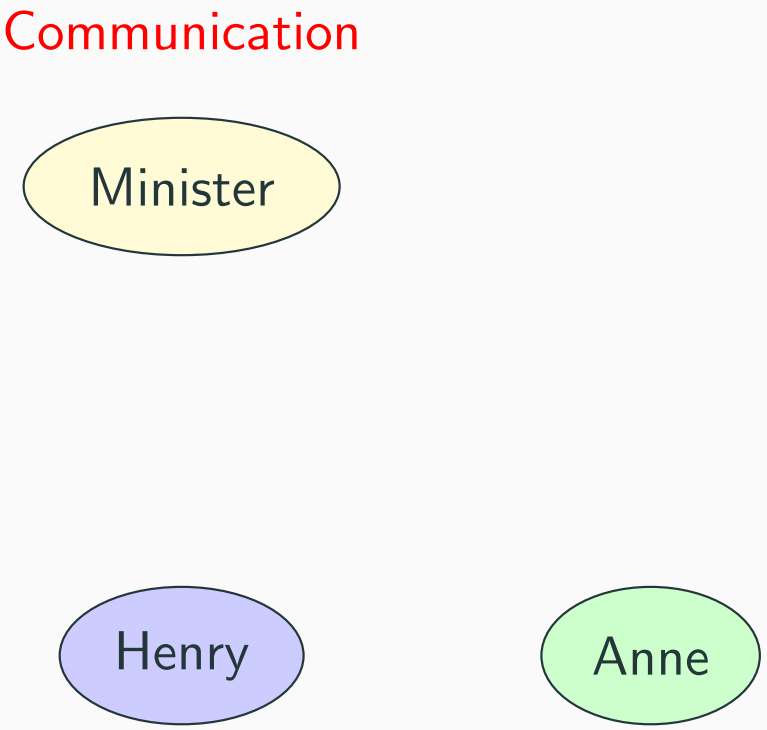
What Transaction Commit Describes



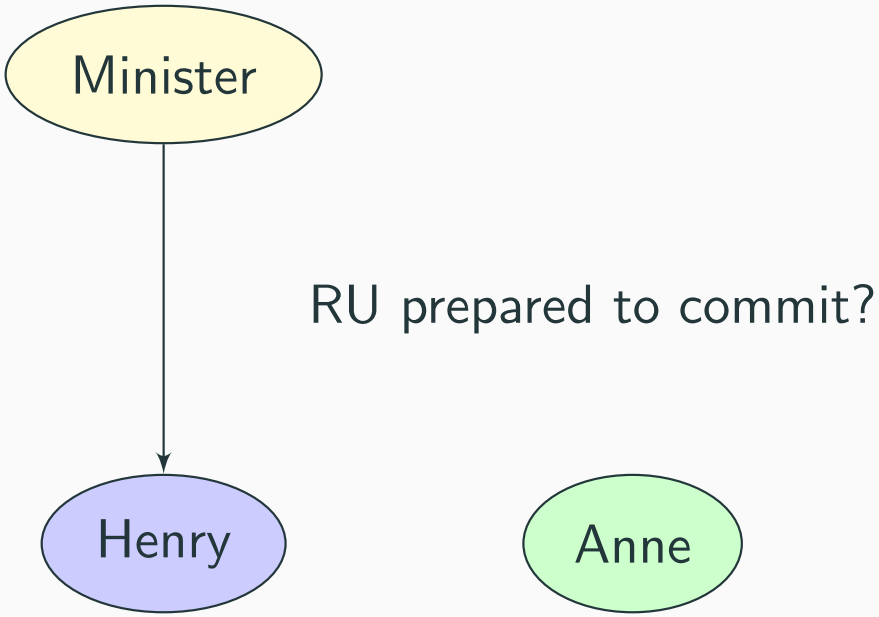
TwoPhase Commit Adds the Minister



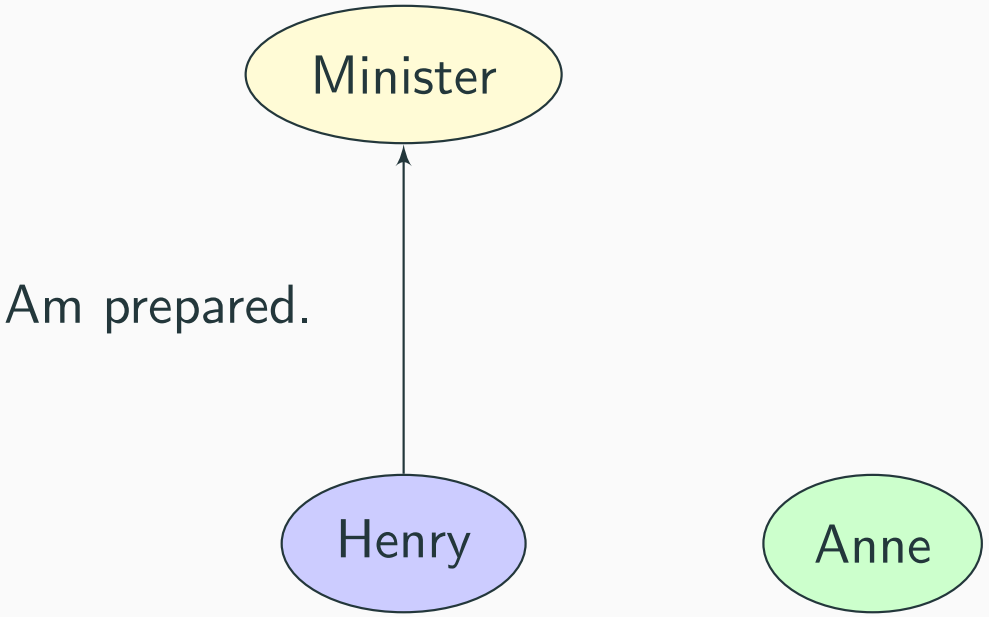
TwoPhase Commit Adds the Minister



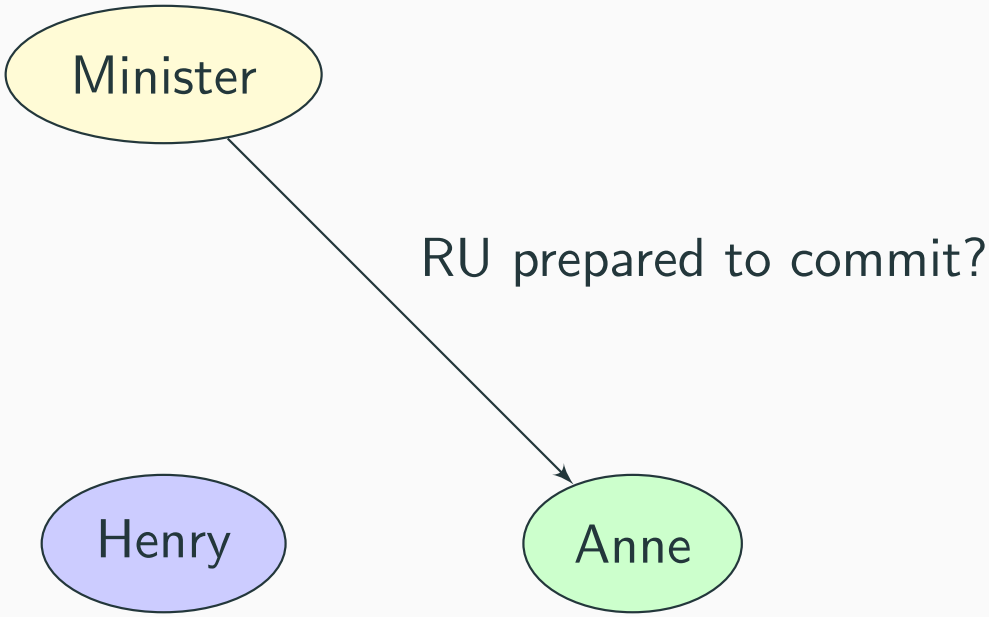
TwoPhase Commit Adds the Minister



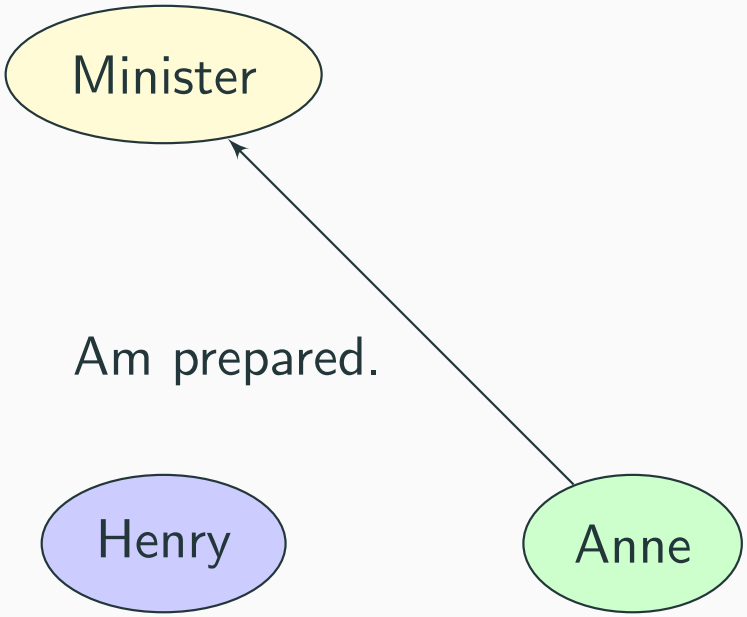
TwoPhase Commit Adds the Minister



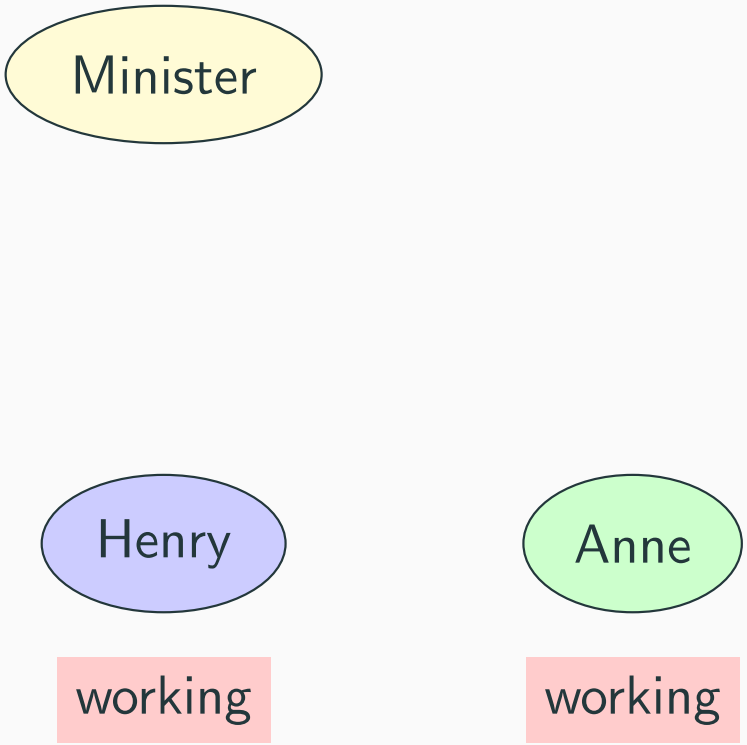
TwoPhase Commit Adds the Minister



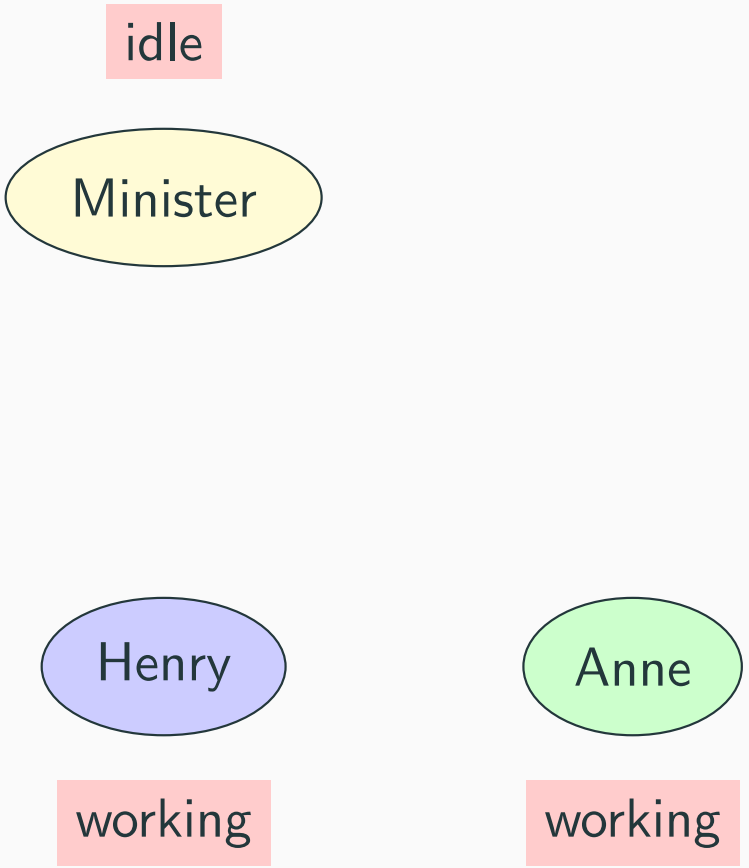
TwoPhase Commit Adds the Minister



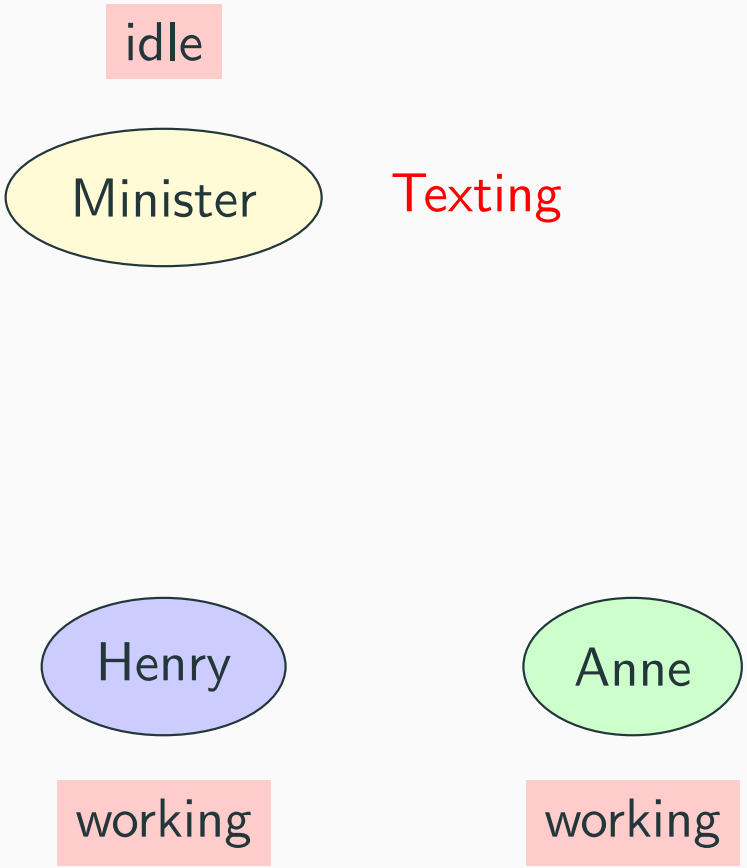
TwoPhase Commit Adds the Minister



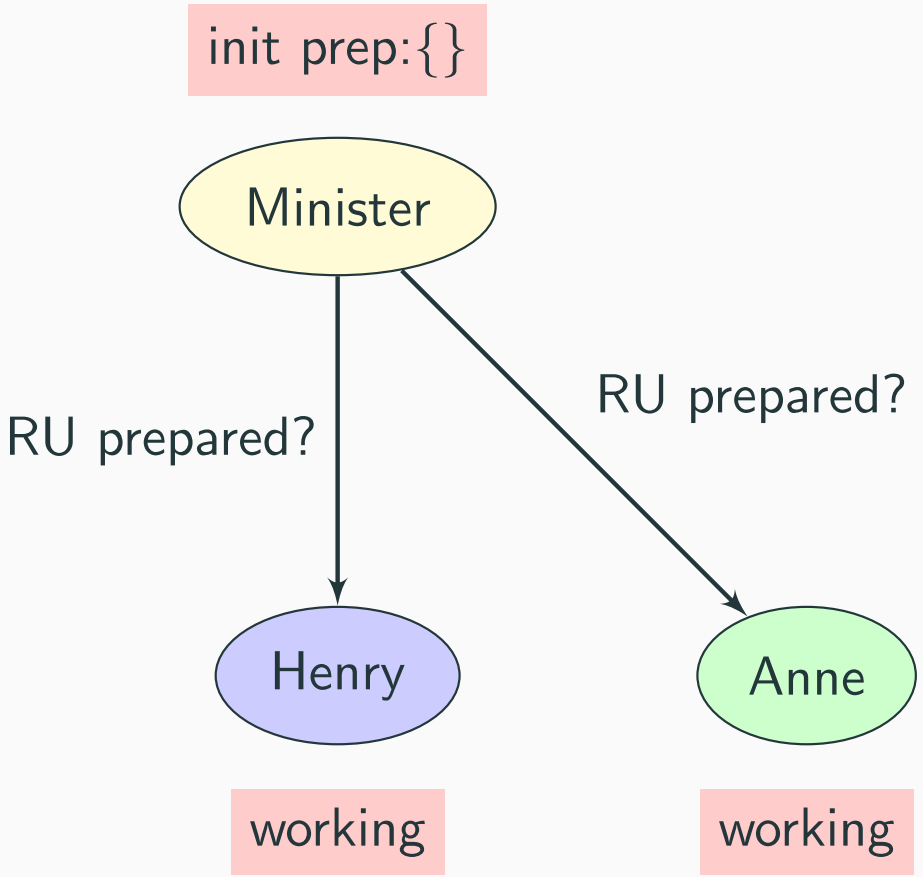
TwoPhase Commit Adds the Minister



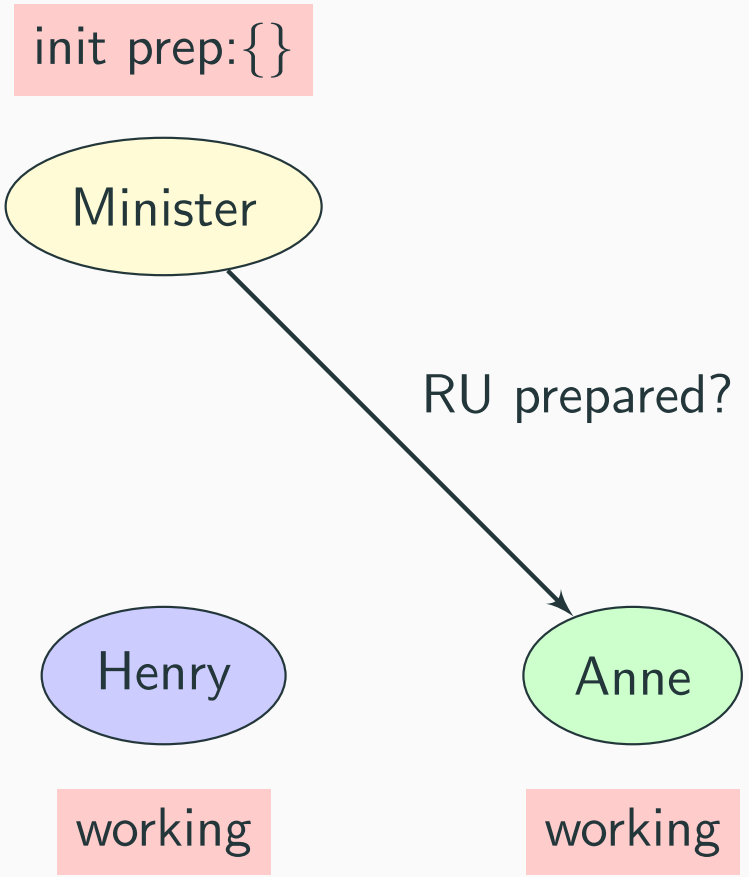
TwoPhase Commit Adds the Minister



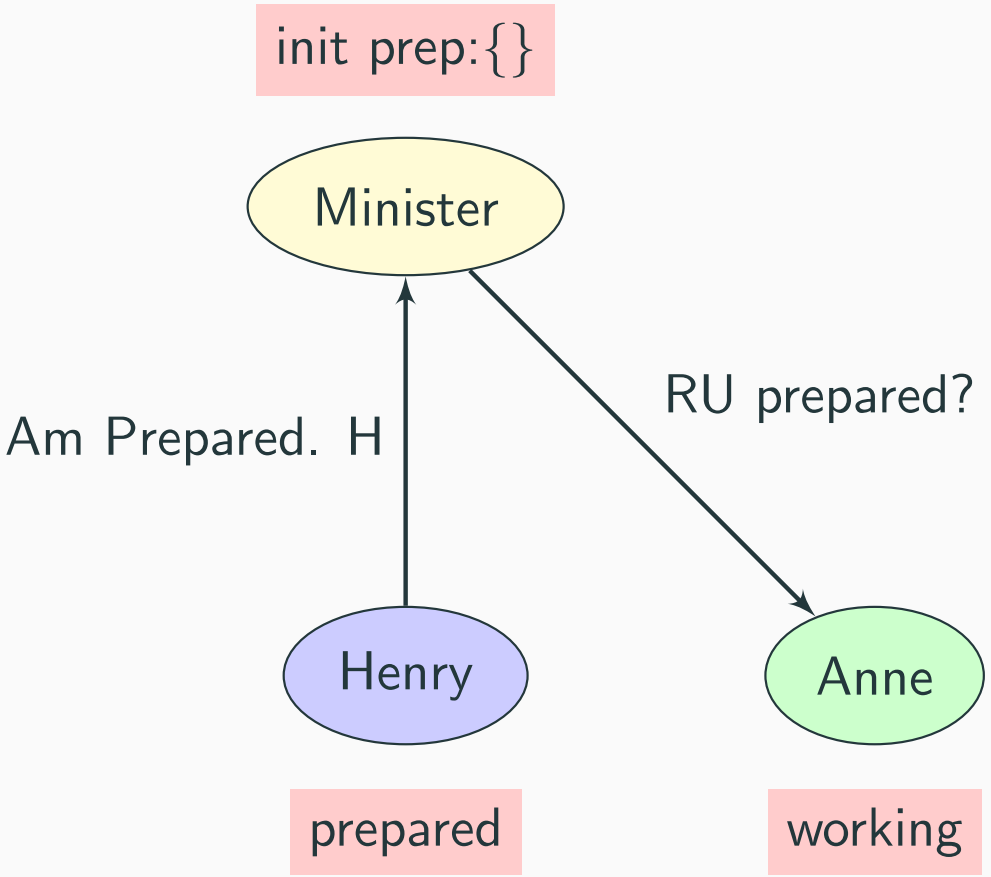
A Really Modern Wedding



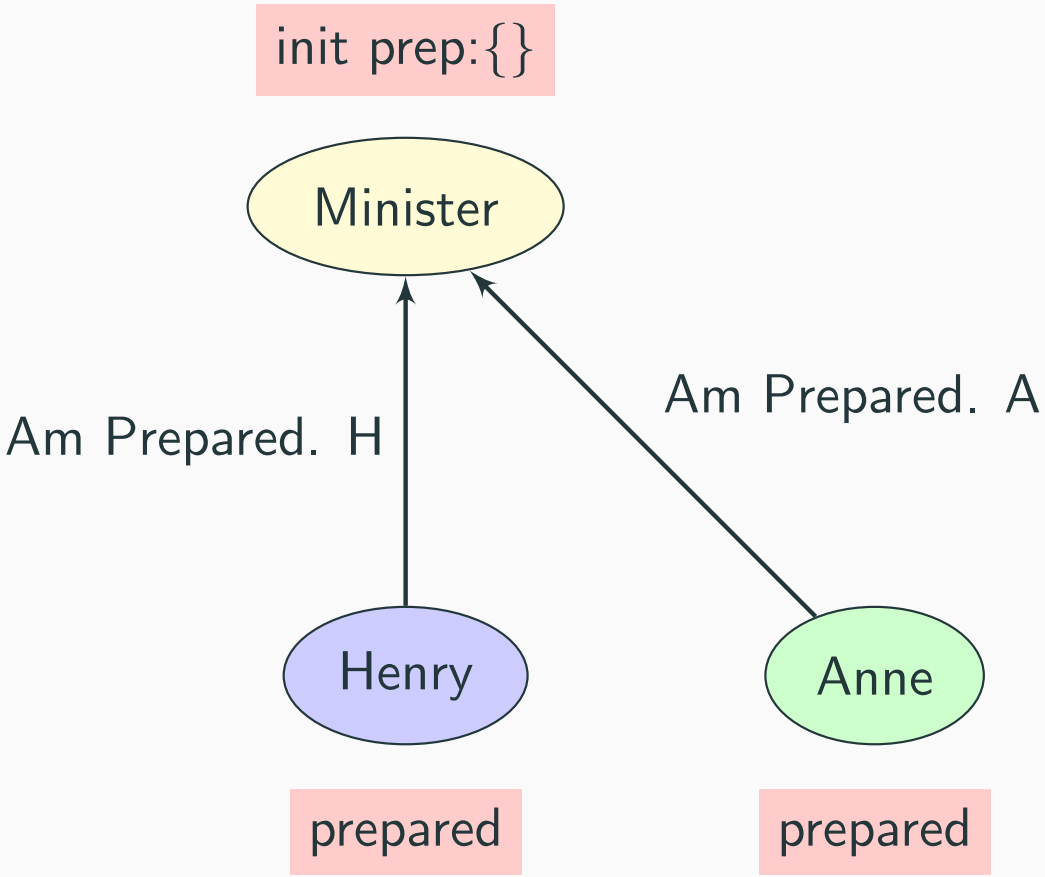
A Really Modern Wedding



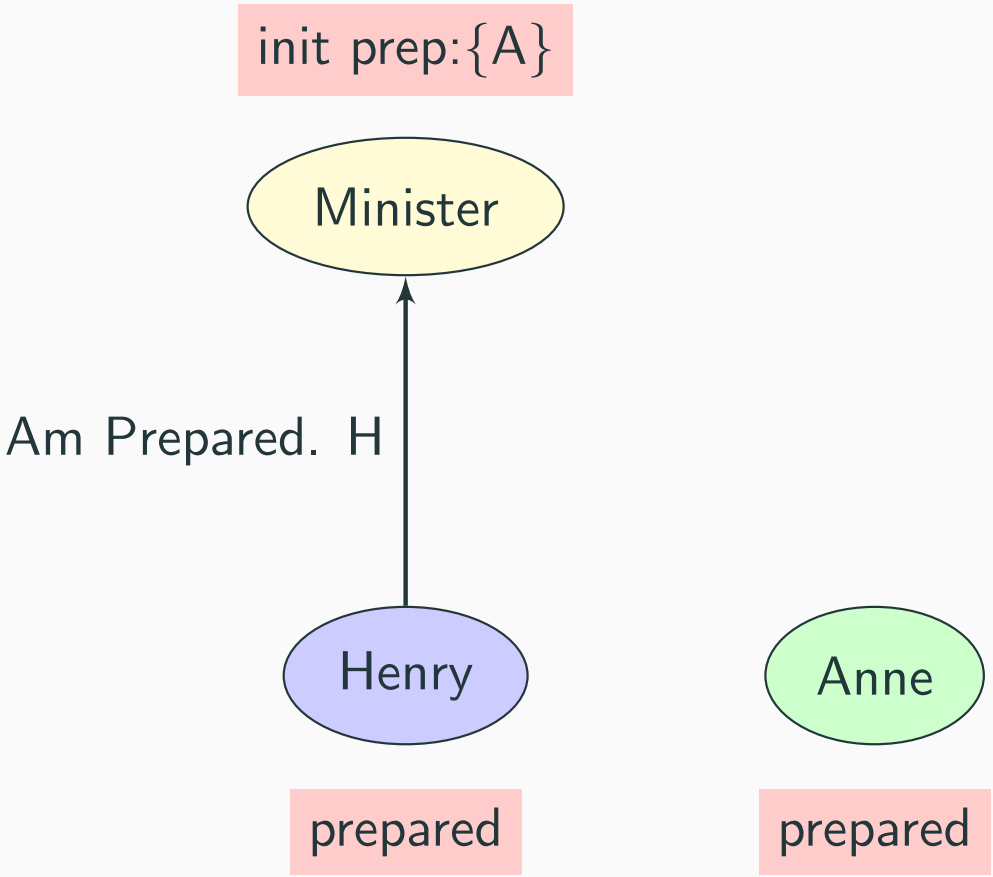
A Really Modern Wedding



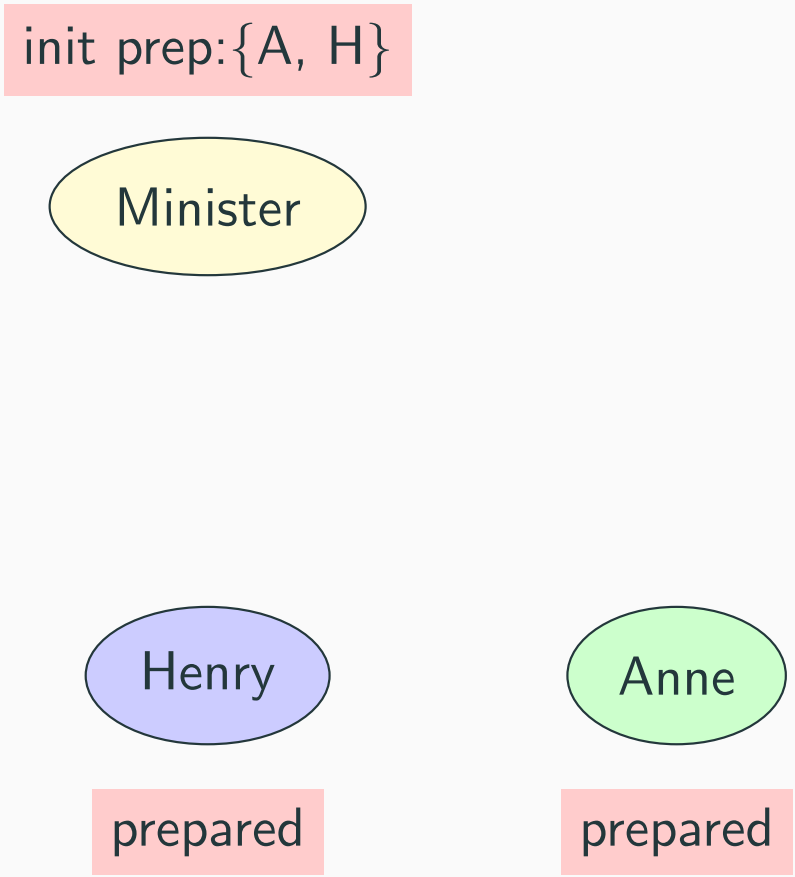
A Really Modern Wedding



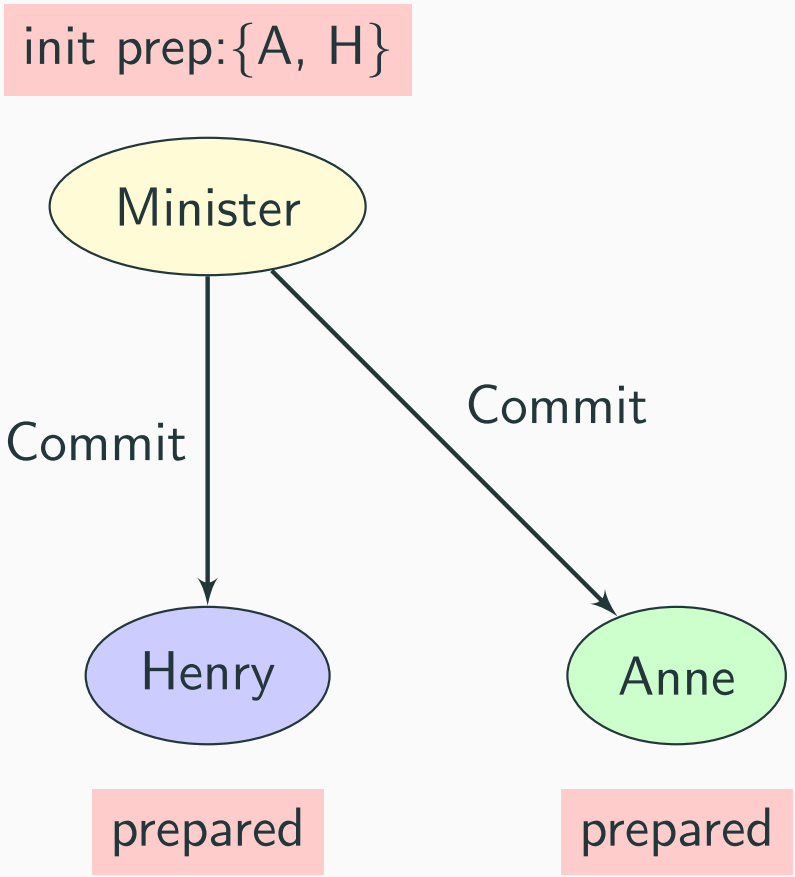
A Really Modern Wedding



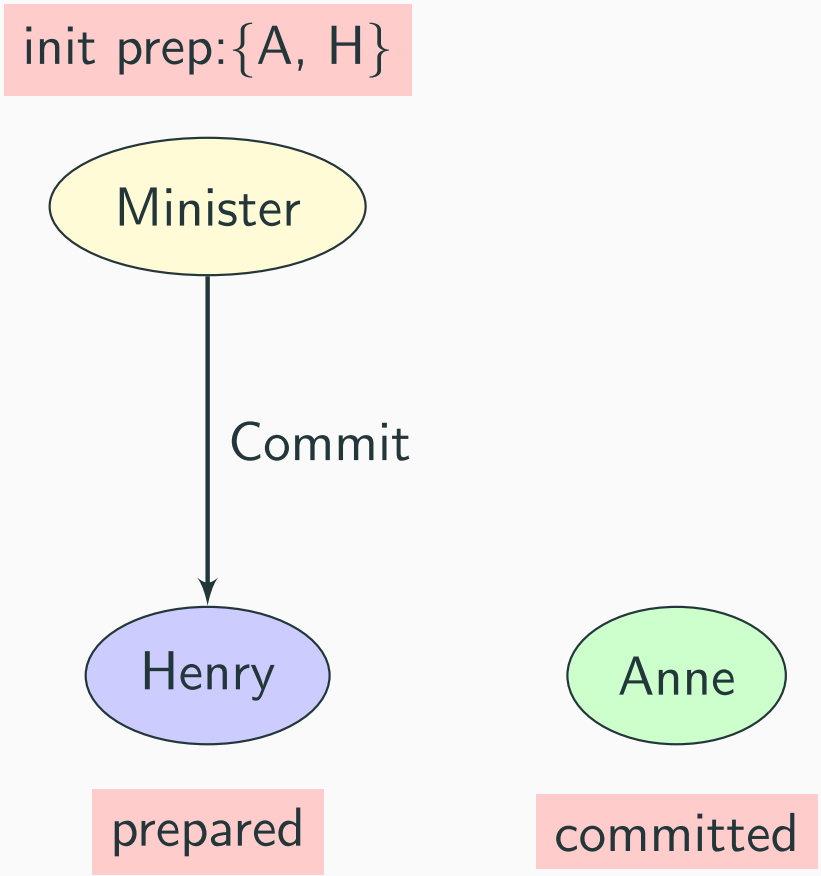
A Really Modern Wedding



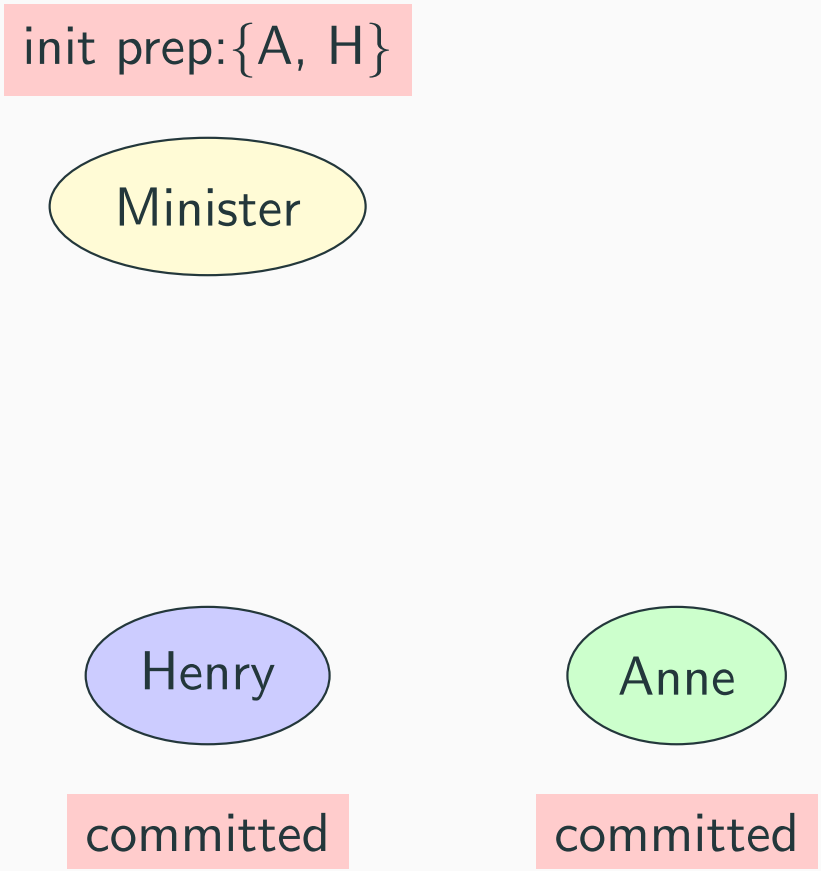
A Really Modern Wedding



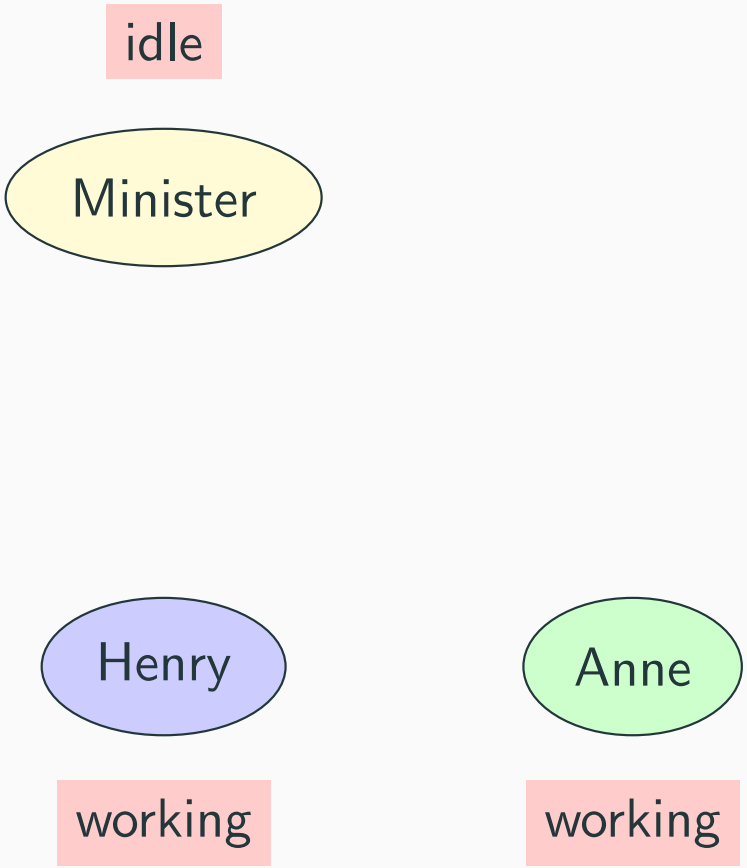
A Really Modern Wedding



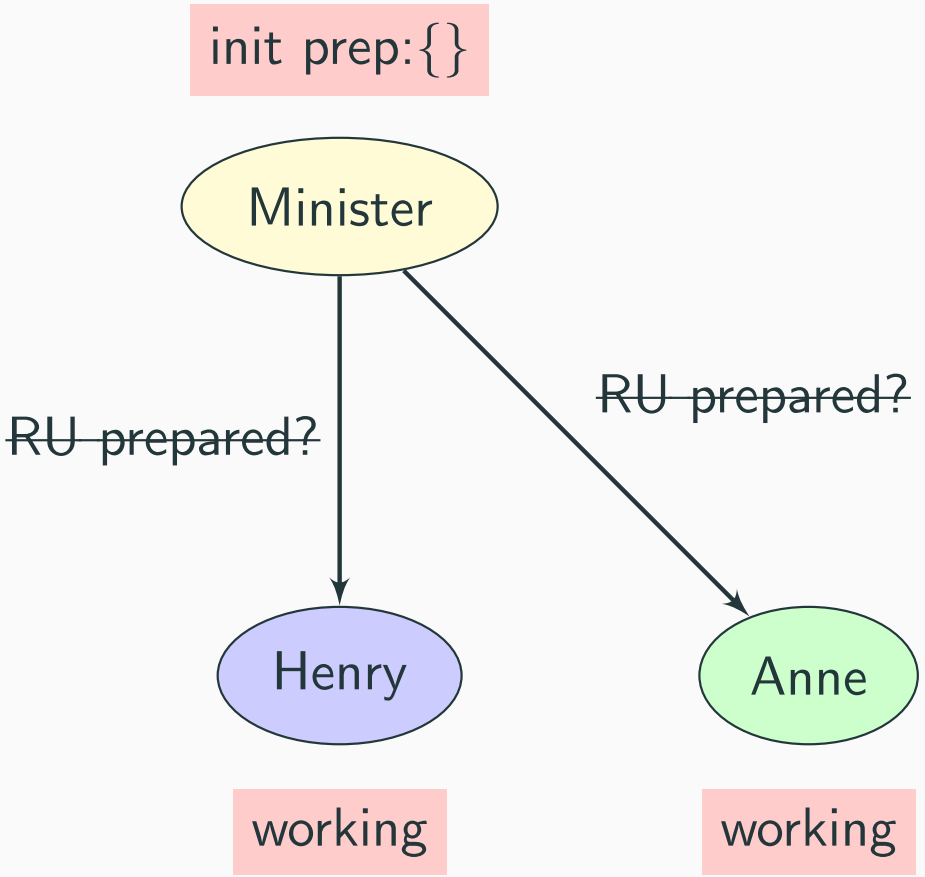
A Really Modern Wedding



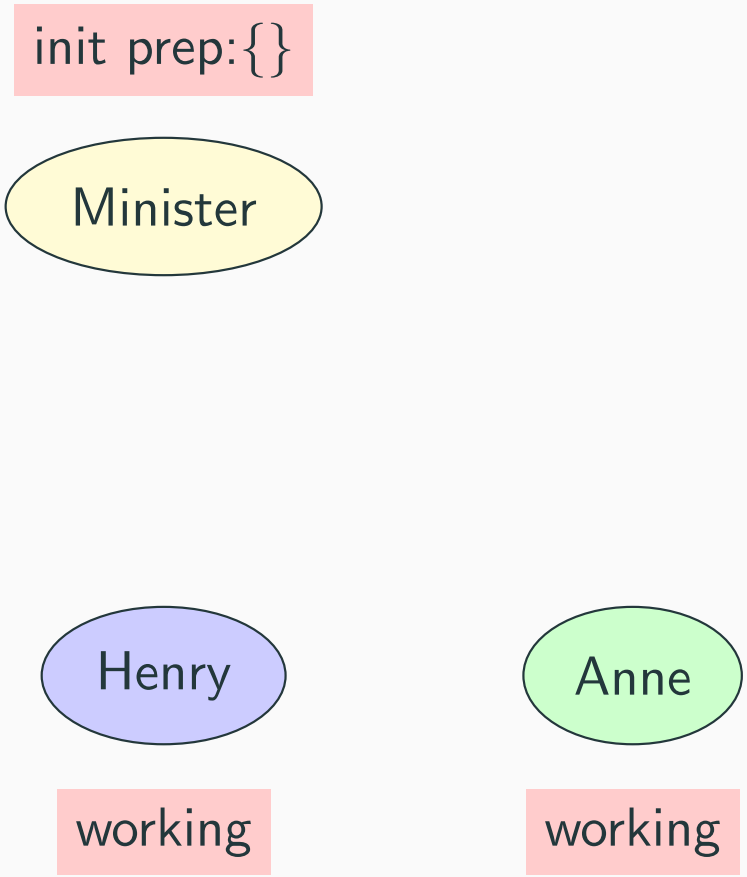
A Simplification



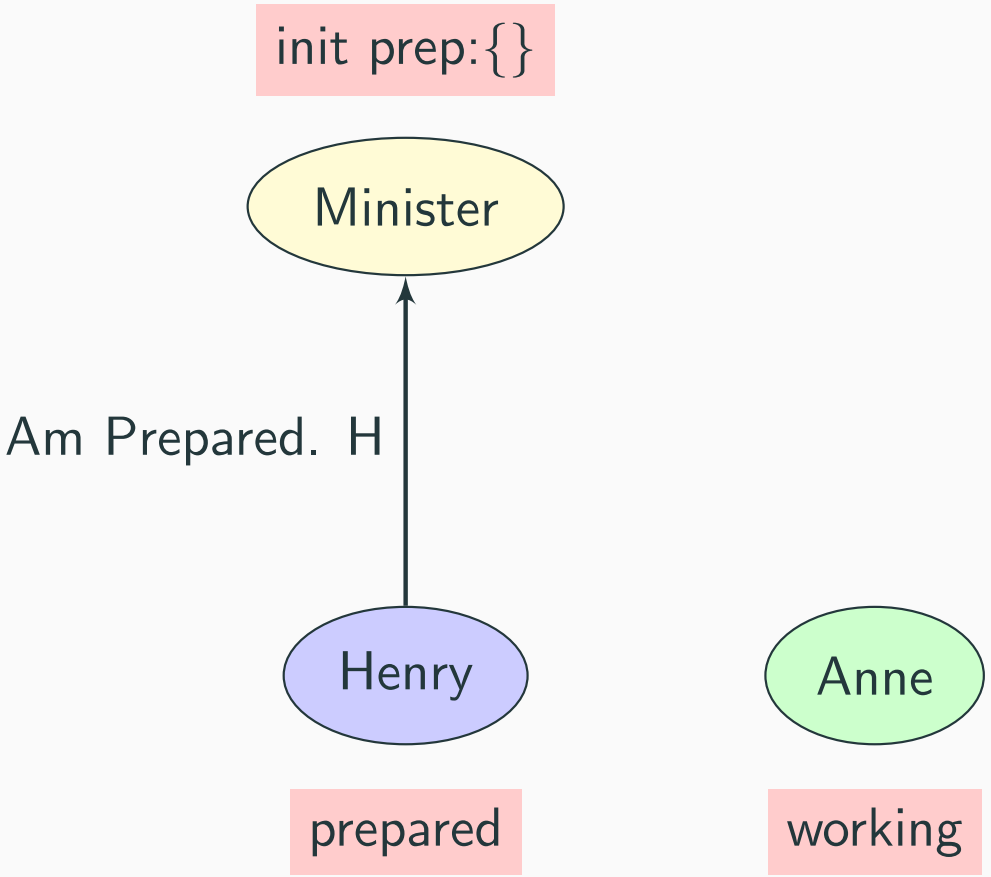
A Simplification



A Simplification



A Simplification



A Simplification

RUPrepared? message not required by *TCommit*.

Simplicity, simplicity, simplicity!

We want the simplest spec that can catch the errors we're looking for, namely, ones that would cause two-phase commit not to satisfy the *TCommit* spec.

The TLA+ Spec

Declarations of the TLA+ Spec

```
CONSTANT RM
VARIABLES rmState, tmState, tmPrepared, msgs
```

RM: set of resource managers

rmState: state of the resource managers

tmState, *tmPrepared*: state of the Transaction Manager

msgs: messages that are in transit

Declarations of the TLA+ Spec

```
CONSTANT RM
VARIABLES rmState, tmState, tmPrepared, msgs
```

RM: set of resource managers

rmState: state of the resource managers

tmState, *tmPrepared*: state of the Transaction Manager

msgs: messages that are in transit

Declarations of the TLA+ Spec

```
CONSTANT RM
VARIABLES rmState, tmState, tmPrepared, msgs
```

RM: set of resource managers

rmState: state of the resource managers

tmState, *tmPrepared*: state of the Transaction Manager

msgs: messages that are in transit

Sending Messages in the TLA+ Spec

$$
\begin{aligned}
&TPTypeOK \triangleq \\
&\land\ rmState \in [RM \to \{\text{“}working\text{”}, \text{“}prepared\text{”}, \text{“}committed\text{”}, \text{“}aborted\text{”}\}] \\
&\land\ tmState \in \{\text{“}init\text{”}, \text{”}done\text{”}\} \\
&\land\ tmPrepared \subseteq RM \\
&\land\ msgs \subseteq Messages
\end{aligned}
$$

Sending Messages in the TLA+ Spec

It should specify only whats required of message passing.

Let *msgs* be the set of all messages ever sent.

A single message can be received by multiple processes.

A process can receive the same message multiple times.

Two-phase commit still works.

Sending Messages in the TLA+ Spec

It should specify only whats required of message passing.

Let *msgs* be the set of all messages ever sent.

A single message can be received by multiple processes.

A process can receive the same message multiple times.

Two-phase commit still works.

Sending Messages in the TLA+ Spec

It should specify only whats required of message passing.

Let *msgs* be the set of all messages ever sent.

A single message can be received by multiple processes.

A process can receive the same message multiple times.

Two-phase commit still works.

Sending Messages in the TLA+ Spec

It should specify only whats required of message passing.

Let *msgs* be the set of all messages ever sent.

A single message can be received by multiple processes.

A process can receive the same message multiple times.

Two-phase commit still works.

Sending Messages in the TLA+ Spec

It should specify only whats required of message passing.

Let *msgs* be the set of all messages ever sent.

A single message can be received by multiple processes.

A process can receive the same message multiple times.

Two-phase commit still works.

Sending Messages in the TLA+ Spec

$$
\begin{aligned}
&TPTypeOK \triangleq \\
&\land\ rmState \in [RM \to \{\text{“}working\text{”}, \text{“}prepared\text{”}, \text{“}committed\text{”}, \text{“}aborted\text{”}\}] \\
&\land\ tmState \in \{\text{“}init\text{”}, \text{”}done\text{”}\} \\
&\land\ tmPrepared \subseteq RM \\
&\land\ msgs \subseteq Messages
\end{aligned}
$$

Sending Messages in the TLA+ Spec

$$Messages \triangleq [type : \text{“}Prepared\text{”}, rm : RM] \cup [type : \text{“}Commit\text{”}, \text{“}Abort\text{”}]$$

$$[type \mapsto \text{“}Prepared\text{”}, rm \mapsto r]$$

represents a Prepared message sent by r to the TM.

$$[type \mapsto \text{“}Commit\text{”}], [type \mapsto \text{“}Abort\text{”}]$$

Each record represents a message sent by the TM to all RMs.

Sending Messages in the TLA+ Spec

$$
\begin{aligned}
&\mathit{TPInit} \triangleq \\
&\land\ \mathit{rmState} = [r \in \mathit{RM} \mapsto \text{“}\mathit{working}\text{”}] \\
&\land\ \mathit{tmState} = \text{“}\mathit{init}\text{”} \\
&\land\ \mathit{tmPrepared} = \{\} \\
&\land\ \mathit{msgs} = \{\}
\end{aligned}
$$

Sending Messages in the TLA+ Spec

$$
\begin{aligned}
&\mathit{TMRcvPrepared}(r) \triangleq \\
&\land\ \mathit{tmState} = \text{“}\mathit{init}\text{”} \\
&\land\ [\mathit{type} \mapsto \text{“}\mathit{Prepared}\text{”}, \mathit{rm} \mapsto r] \in \mathit{msgs} \\
&\land\ \mathit{tmPrepared}' = \mathit{tmPrepared} \cup \{r\} \\
&\land\ \text{UNCHANGED } \langle \mathit{rmState}, \mathit{tmState}, \mathit{msgs} \rangle
\end{aligned}
$$

Enabling conditions: conditions on the first state of a step

All subsequent $\mathit{TMRcvPrepared}(r)$ steps leave all the variables unchanged.

THE REST OF THE SPEC

TMCommit $\triangleq$

It allows steps where the TM sends Commit messages to the RMs and sets tmState to “done”.

It is enabled if tmState equals “init” and tmPrepared equals RM.

```
TMCommit ==
  /\ tmState = "init"
  /\ tmPrepared = RM
  /\ tmState' = "done"
  /\ msgs' = msgs \cup {[type |-> "Commit"]}
  /\ UNCHANGED <<rmState, tmPrepared>>
```

THE REST OF THE SPEC

TMAbort $\triangleq$

The TM sends Abort messages to the RMs and sets tmState to “done”.

It is enabled if tmState equals “init”.

```
TMAbort ==
  /\ tmState = "init"
  /\ tmState' = "done"
  /\ msgs' = msgs \cup {[type |-> "Abort"]}
  /\ UNCHANGED <<rmState, tmPrepared>>
```

THE REST OF THE SPEC

$RMPrepare(r) \triangleq$

RM r sets its state to "*prepared*" and sends a Prepared message to the TM.

It's enabled if $rmState[r]$ equals "*working*".

```
RMPrepare(r) ==
  /\ rmState[r] = "working"
  /\ rmState' = [rmState EXCEPT ![r] = "prepared"]
  /\ msgs' = msgs \cup {[type |-> "Prepared", rm |-> r]}
  /\ UNCHANGED <<tmState, tmPrepared>>
```

THE REST OF THE SPEC

$RMChooseToAbort(r) \triangleq$

When in its “working” state, RM r can go to the “aborted” state.

```
RMChooseToAbort(r) ==
  /\ rmState[r] = "working"
  /\ rmState' = [rmState EXCEPT ![r] = "aborted"]
  /\ UNCHANGED <<tmState, tmPrepared, msgs>>
```

THE REST OF THE SPEC

$RMRcvCommitMsg(r) \triangleq$

$RMRcvAbortMsg(r) \triangleq$

RM r receives a “commit” or “abort” message and sets its state accordingly.

THE REST OF THE SPEC

```
TPNext ==
  \/ TMCommit \/ TMAbort
  \/ \E r \in RM :
       TMRcvPrepared(r) \/ RMPrepare(r) \/ RMChooseToAbort(r)
         \/ RMRcvCommitMsg(r) \/ RMRcvAbortMsg(r)
```

Checking the Spec

TLA+ Toolbox

- Create a New Model
- Check Your Definitions

Model Values

Symmetry Sets All RMs are identical/interchangeable.

Suppose RM $= \{$“r1”, “r2”, “r3”$\}$.

“r1” $\leftrightarrow$ “r3” in one possible state yields a possible state.

Correctness of Two-Phase Commit

We've checked that TypeOK is an invariant of the spec.

We should check that formula TCConsistent of TCommit, which asserts that one RM can't commit and another abort, is also an invariant.

The statement

INSTANCE *TCommit*

imports the definitions from *TCommit* into module *TwoPhase*.

Add the invariant *TCConsistent* to your model and have TLC check it.

Two-phase commit doesn't just maintain the invariance of *TCConsistent*; it implements the specification of transaction commit.

Correctness of Two-Phase Commit

We've checked that TypeOK is an invariant of the spec.

We should check that formula TCConsistent of TCommit, which asserts that one RM can't commit and another abort, is also an invariant.

The statement

INSTANCE *TCommit*

imports the definitions from *TCommit* into module *TwoPhase*.

Add the invariant *TCConsistent* to your model and have TLC check it.

Two-phase commit doesn't just maintain the invariance of *TCConsistent*; it implements the specification of transaction commit.

Correctness of Two-Phase Commit

We've checked that TypeOK is an invariant of the spec.

We should check that formula TCConsistent of TCommit, which asserts that one RM can't commit and another abort, is also an invariant.

The statement

INSTANCE *TCommit*

imports the definitions from *TCommit* into module *TwoPhase*.

Add the invariant *TCConsistent* to your model and have TLC check it.

Two-phase commit doesn't just maintain the invariance of *TCConsistent*; it implements the specification of transaction commit.

Correctness of Two-Phase Commit

We've checked that TypeOK is an invariant of the spec.

We should check that formula TCConsistent of TCommit, which asserts that one RM can't commit and another abort, is also an invariant.

The statement

INSTANCE *TCommit*

imports the definitions from *TCommit* into module *TwoPhase*.

Add the invariant *TCConsistent* to your model and have TLC check it.

Two-phase commit doesn't just maintain the invariance of *TCConsistent*; it implements the specification of transaction commit.

Correctness of Two-Phase Commit

We've checked that TypeOK is an invariant of the spec.

We should check that formula TCConsistent of TCommit, which asserts that one RM can't commit and another abort, is also an invariant.

The statement

INSTANCE *TCommit*

imports the definitions from *TCommit* into module *TwoPhase*.

Add the invariant *TCConsistent* to your model and have TLC check it.

Two-phase commit doesn't just maintain the invariance of *TCConsistent*; it implements the specification of transaction commit.

Thanks for your listening!