THE UNIVERSITY of EDINBURGH

Advanced Databases

Spring 2020

Lecture #14:

Concurrency Control II

Milos Nikolic

Concurrency Control

Locking is a conservative approach in which conflicts are prevented

Disadvantages of locking:

- Lock management overhead
- Deadlock detection/resolution
- Lock contention for heavily used objects

If conflicts are **rare** and most txns are **short-lived**, then forcing txns to wait to acquire locks adds a lot of overhead

A better approach it to optimise for the no-conflict case

Concurrency Control Approaches

Two-Phase Locking (2PL)

Determine serializability order of conflicting ops at runtime while txns execute

Timestamp-Based Concurrency Control

Determine serializability order of txns before they execute

Optimistic Concurrency Control

Check for conflicts when txns commit, if necessary abort

Multi-Version Concurrency Control *(not covered in this course)*

Writers make a ”new” copy while readers use an appropriate “old” copy

Timestamp-Based Concurrency Control

Use timestamps to determine the serializability order of transactions

Each txn T_i is assigned a unique fixed timestamp

- Let $TS(T_i)$ be the timestamp allocated to transaction T_i
- Timestamps are monotonically increasing
- Different schemes assign timestamps at different times during execution

If $TS(T_i) < TS(T_j)$, then the DBMS must ensure that the execution schedule is equivalent to a **serial** schedule where T_i appears before T_j

Timestamp-Based Concurrency Control

Transactions read and write objects without locks

Every object X is tagged with timestamp of the last txn that successfully did read/write

- WTS(X) – Write timestamp on X
- RTS(X) – Read timestamp on X

Check timestamp for every operation

- If txn tries to access an object “from the future”, it aborts and restarts

Timestamp-Based CC – Reads

If $TS(T_i) < WTS(X)$

- This violates timestamp order of T_i with regards to the writer of X
- Abort and restart T_i

Else

- Allow T_i to read X
- Update $RTS(X)$ to $\max(RTS(X), TS(T_i))$
- Have to make a local copy of X to ensure repeatable reads for T_i

Timestamp-Based CC – Writes

If $TS(T_i) < RTS(X)$ or $TS(T_i) < WTS(X)$

Abort and restart T_i

Else

Allow T_i to write X and update $WTS(X)$

Also have to make a local copy of X to ensure repeatable reads for T_i

Timestamp-Based CC – Example #1

$TS(T_1) = 1$

Schedule

$TS(T_2) = 2$

TIME

T_1	T_2
BEGIN	
R(B)	
	BEGIN
	R(B)
	W(B)
R(A)	
	R(A)
	W(A)
COMMIT	**COMMIT**

Database

Object	RTS	WTS
A	0	0
B	0	0

Check $TS(T_1) \geq WTS(B)$ ✓

Timestamp-Based CC – Example #1

$TS(T_1) = 1$

$TS(T_2) = 2$

TIME

Schedule

T_1	T_2
BEGIN	
R(B)	
	BEGIN
	R(B)
	W(B)
R(A)	
	R(A)
	W(A)
COMMIT	**COMMIT**

Database

Object	RTS	WTS
A	0	0
B	1	0

Check $TS(T_1) \geq WTS(B)$ ✓

Set $RTS(B) = TS(T_1)$

Timestamp-Based CC – Example #1

$TS(T_1) = 1$ Schedule $TS(T_2) = 2$

TIME

T_1	T_2
BEGIN	
R(B)	
	BEGIN
	R(B)
	W(B)
R(A)	
	R(A)
	W(A)
COMMIT	COMMIT

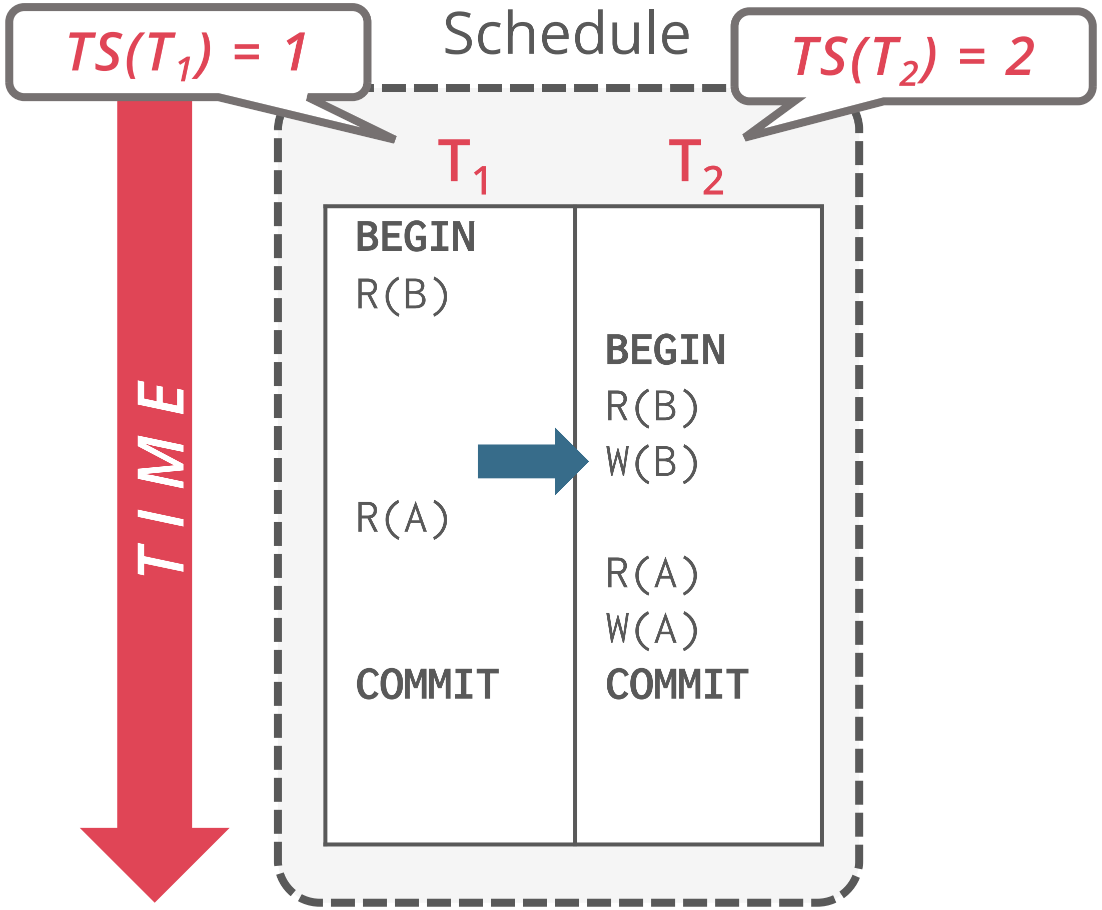
Database

Object	RTS	WTS
A	0	0
B	2	0

Check $TS(T_2) \geq WTS(B)$ ✓

Set $RTS(B) = TS(T_2)$

TIMESTAMP-BASED CC – EXAMPLE #1



Database

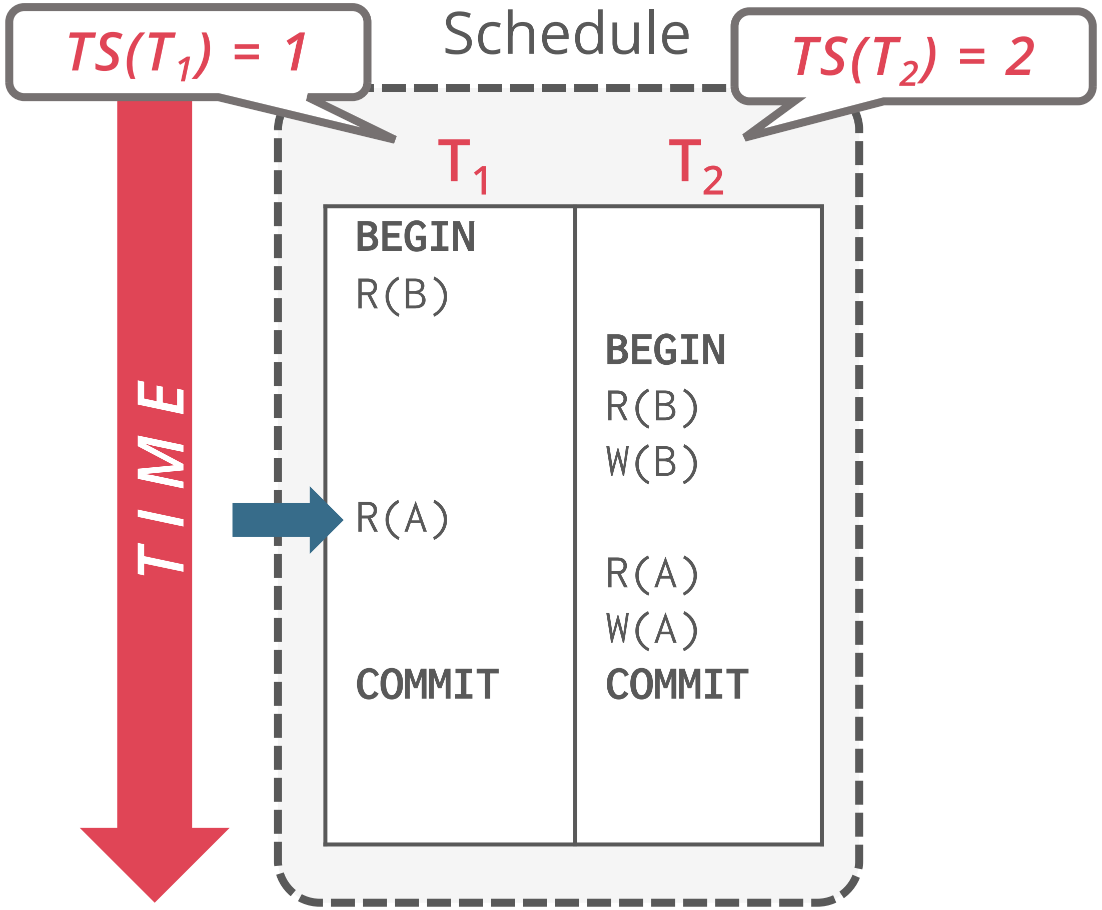
Object	RTS	WTS
A	0	0
B	2	2

Check $TS(T_2) \geq RTS(B) = 2$ ✓

$TS(T_2) \geq WTS(B) = 0$ ✓

Set $WTS(B) = TS(T_2)$

Timestamp-Based CC – Example #1



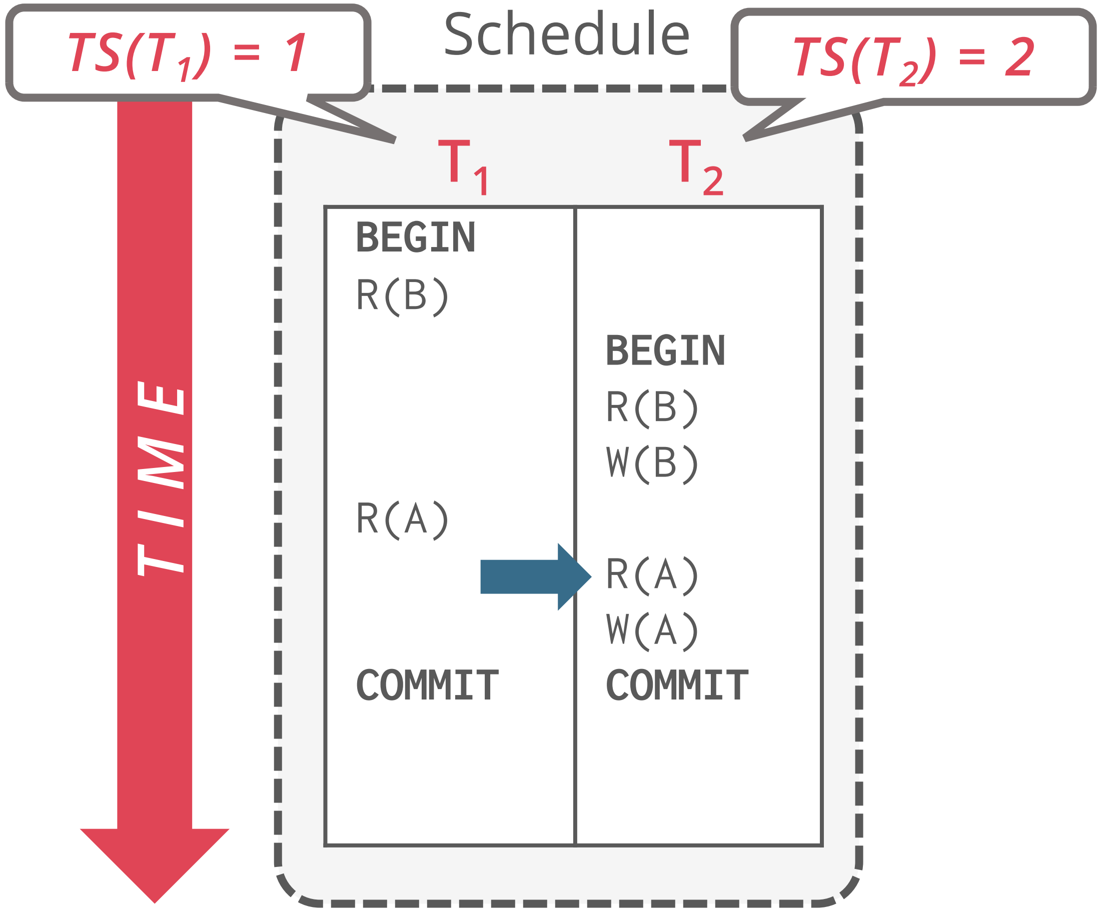
Database

Object	RTS	WTS
A	1	0
B	2	2

Check $TS(T_1) \geq WTS(A)$ ✓

Set $RTS(A) = TS(T_1)$

TIMESTAMP-BASED CC – EXAMPLE #1



Database

Object	RTS	WTS
A	2	0
B	2	2

Check $TS(T_2) \geq WTS(A)$ ✓

Set $RTS(A) = TS(T_2)$

TIMESTAMP-BASED CC – EXAMPLE #1

$TS(T_1) = 1$

Schedule

$TS(T_2) = 2$

TIME

T_1	T_2
BEGIN	
R(B)	
	BEGIN
	R(B)
	W(B)
R(A)	
	R(A)
	W(A)
COMMIT	**COMMIT**

No violations so both txns are safe to commit

Database

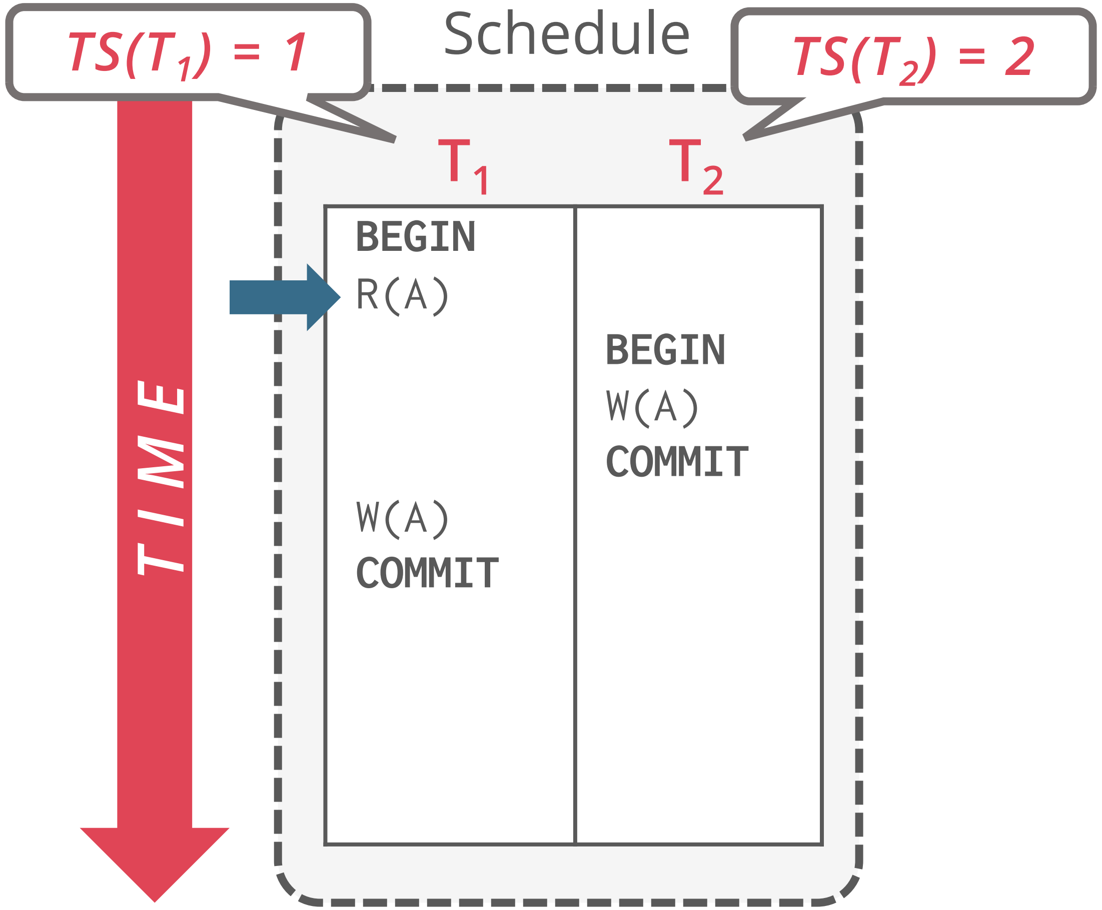
Object	RTS	WTS
A	2	2
B	2	2

Check $TS(T_2) \geq RTS(A) = 2$ ✓

$TS(T_2) \geq WTS(A) = 0$ ✓

Set $WTS(A) = TS(T_2)$

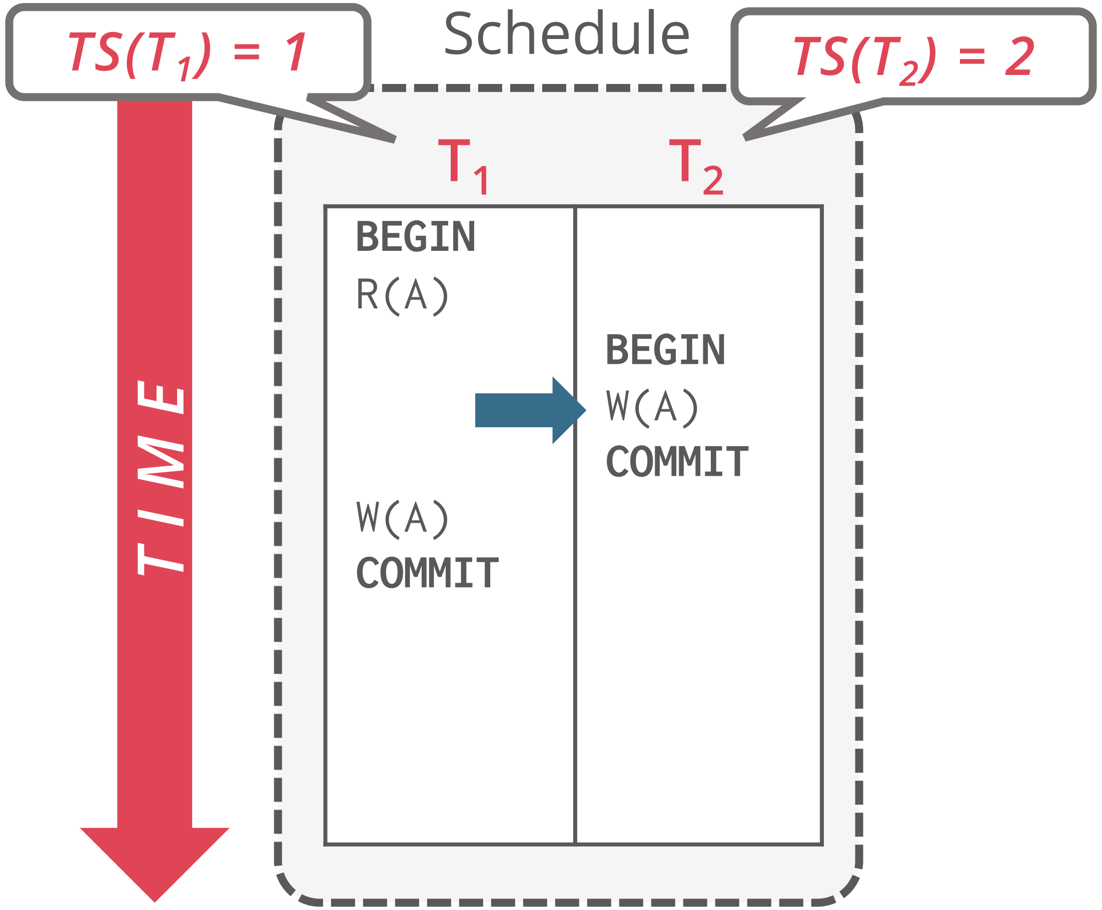
TIMESTAMP-BASED CC – EXAMPLE #2



Database

Object	RTS	WTS
A	1	0
B	0	0

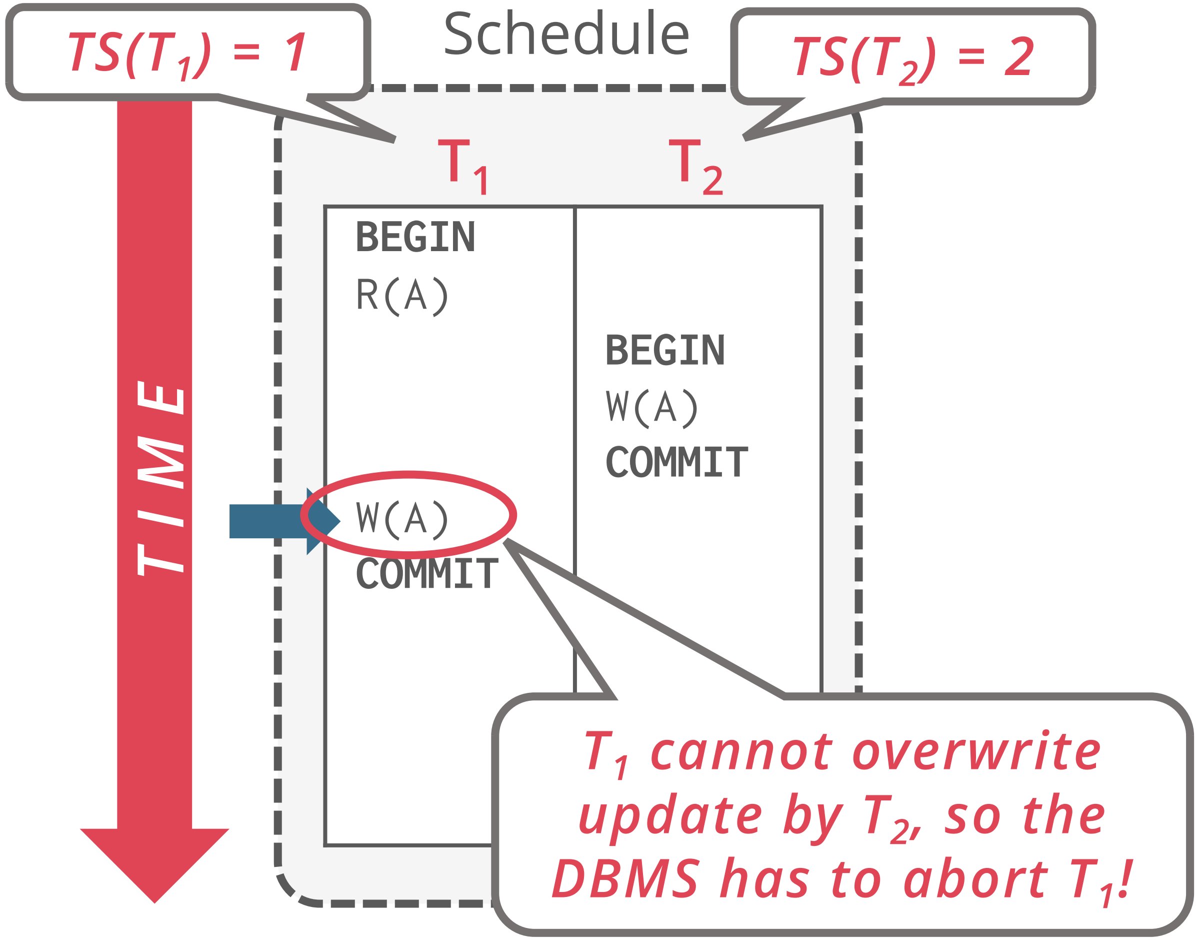
TIMESTAMP-BASED CC – EXAMPLE #2



Database

Object	RTS	WTS
A	1	2
B	0	0

TIMESTAMP-BASED CC – EXAMPLE #2



Database

Object	RTS	WTS
A	1	2
B	0	0

Violation: $TS(T_1) < WTS(A)$

Thomas Write Rule

If $TS(T_i) < RTS(X)$

Abort and restart T_i

If $TS(T_i) < WTS(X)$

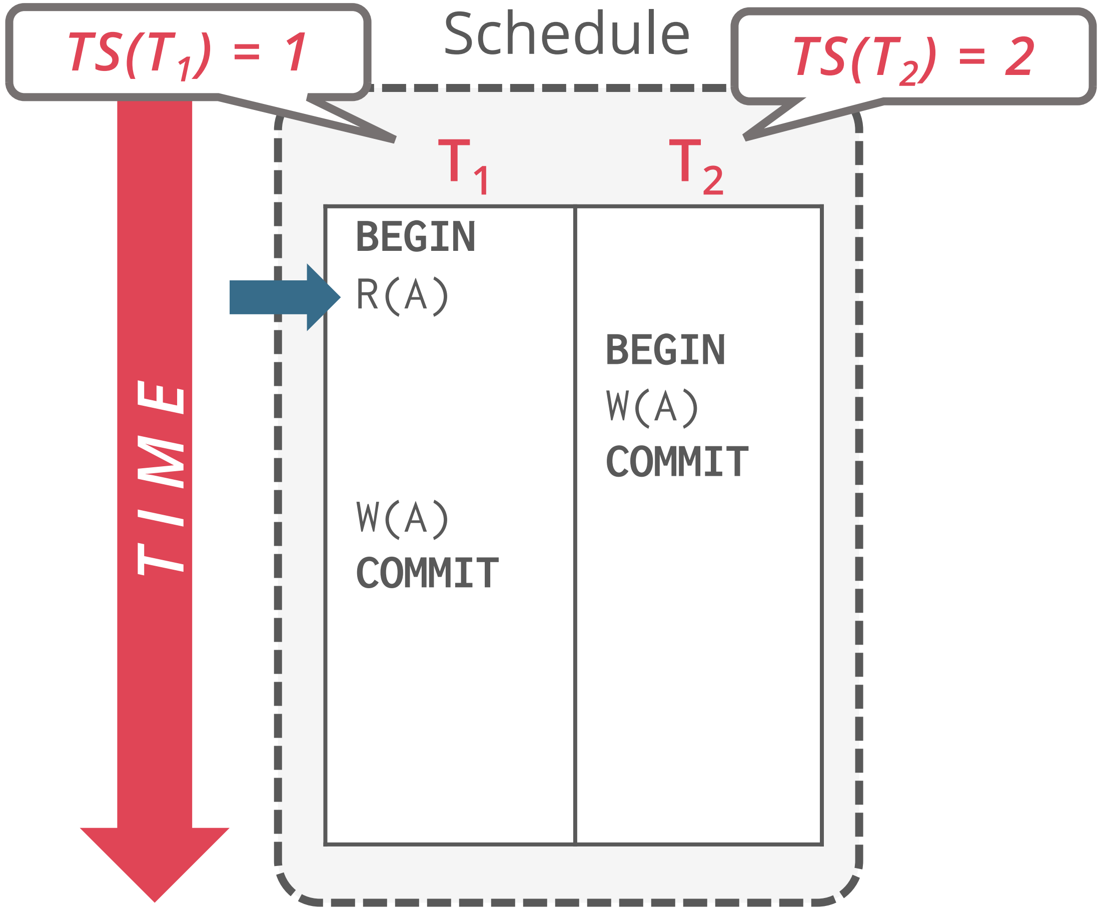
Thomas Write rule: Ignore the write and allow the txn to continue

This violates timestamp order of T_i

Else

Allow T_i to write X and update $WTS(X)$

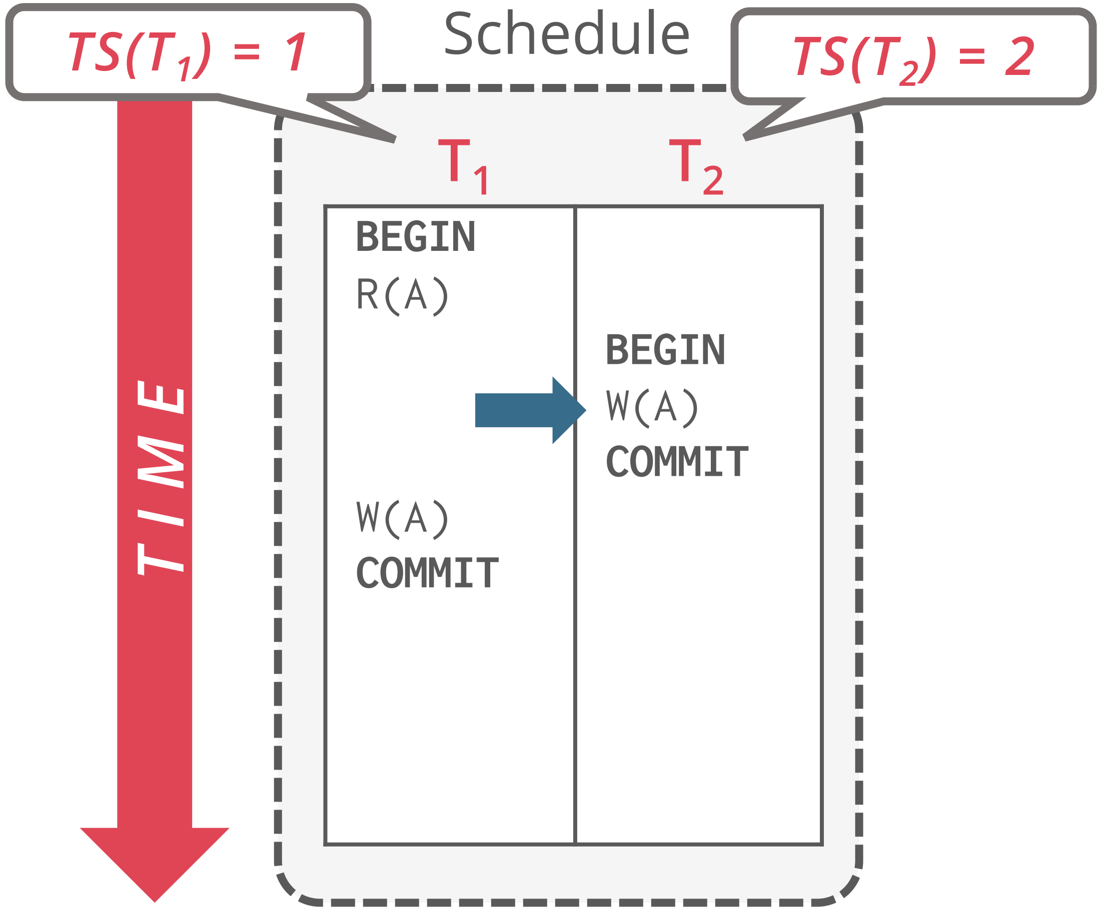
TIMESTAMP-BASED CC – EXAMPLE #2



Database

Object	RTS	WTS
A	1	0
B	0	0

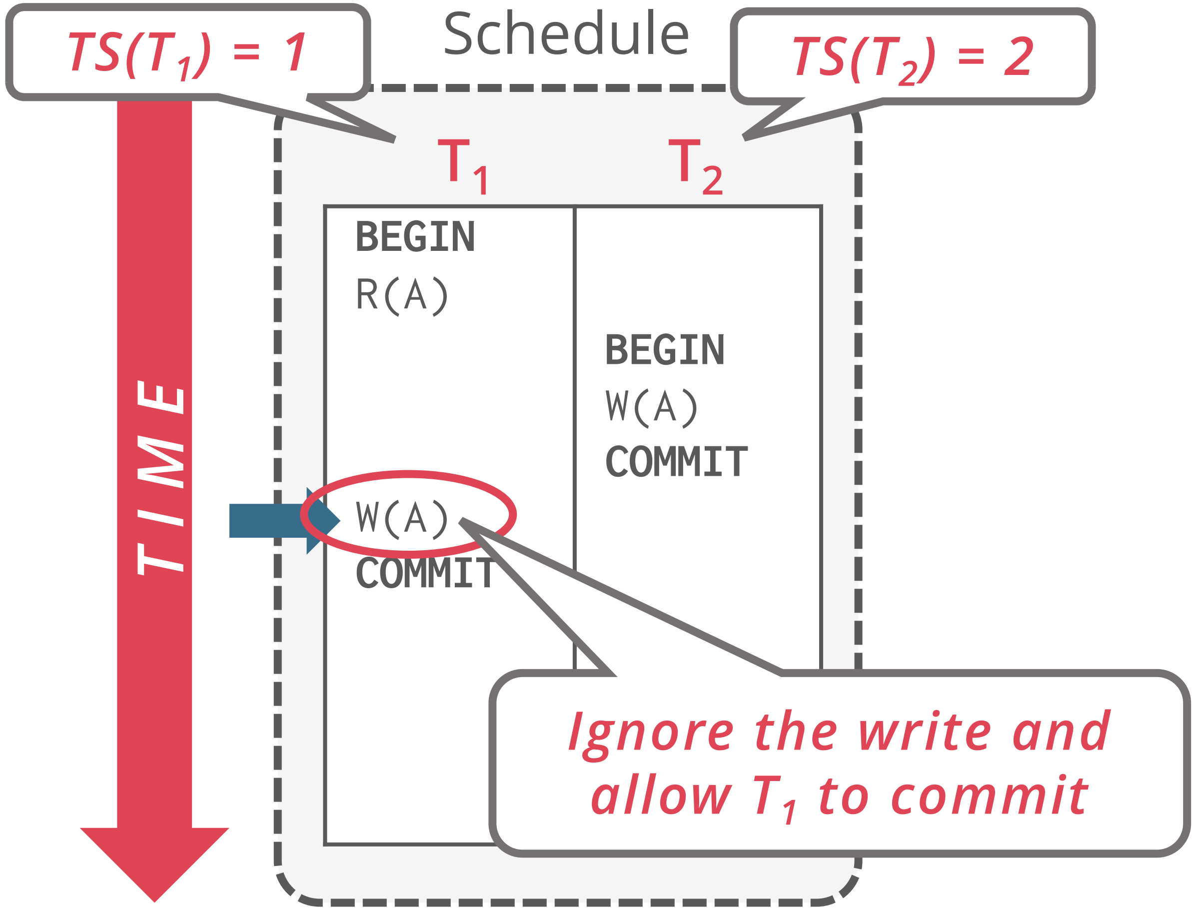
TIMESTAMP-BASED CC – EXAMPLE #2



Database

Object	RTS	WTS
A	1	2
B	0	0

Timestamp-Based CC – Example #2



Database

Object	RTS	WTS
A	1	2
B	0	0

We do not update WTS(A)

Timestamp-Based CC

If the Thomas Write Rule is **not** used, generates conflict serializable schedules

- No deadlocks because no txn ever waits
- Possibility of starvation for long txns if short txns keep causing conflicts

If the Thomas Write Rule is used, may generate some non-conflict serializable schedules

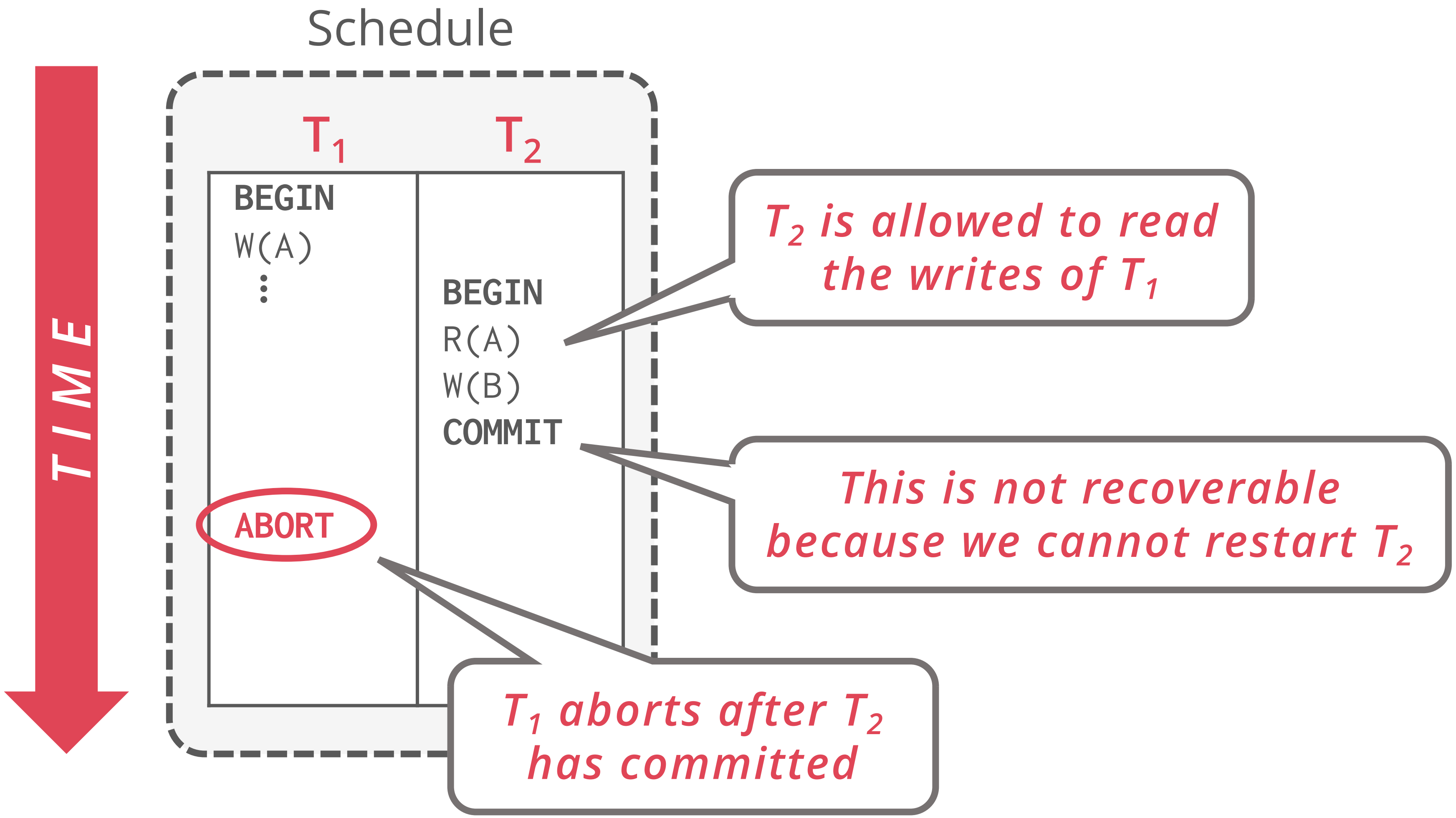
Permits schedules that are not **recoverable**

Recoverable Schedules

A schedule is **recoverable** if txns commit only after all txns whose changes they read, commit

Otherwise, the DBMS cannot guarantee that txns read data that will be restored after recovering from a crash

RECOVERABLE SCHEDULES



Timestamp-Based CC – Observations

Timestamp CC can be modified to allow only recoverable schedules

Buffer all writes until writer commits by writing to a private workspace, but update WTS(X) when the write is allowed

Block readers T (where TS(T) > WTS(X)) until writer of X commits

Performance issues

High overhead from copying data to txn's workspace and updating timestamps

Long running txns can get starved

The likelihood that a txn will read something from a newer txn increases

Optimistic Concurrency Control

The DBMS creates a **private workspace** for each transaction

- Any object read is copied into workspace
- Modifications are applied to workspace

When a txn wants to commit, the DBMS compares workspace write set to see whether it conflicts with other txns

If there are no conflicts, the write set is applied to the “global” database

OCC PHASES

#1 – Read Phase

Track the read/write sets of txns and store their writes in a private workspace

#2 – Validation Phase

When a txn commits, check whether it conflicts with other txns

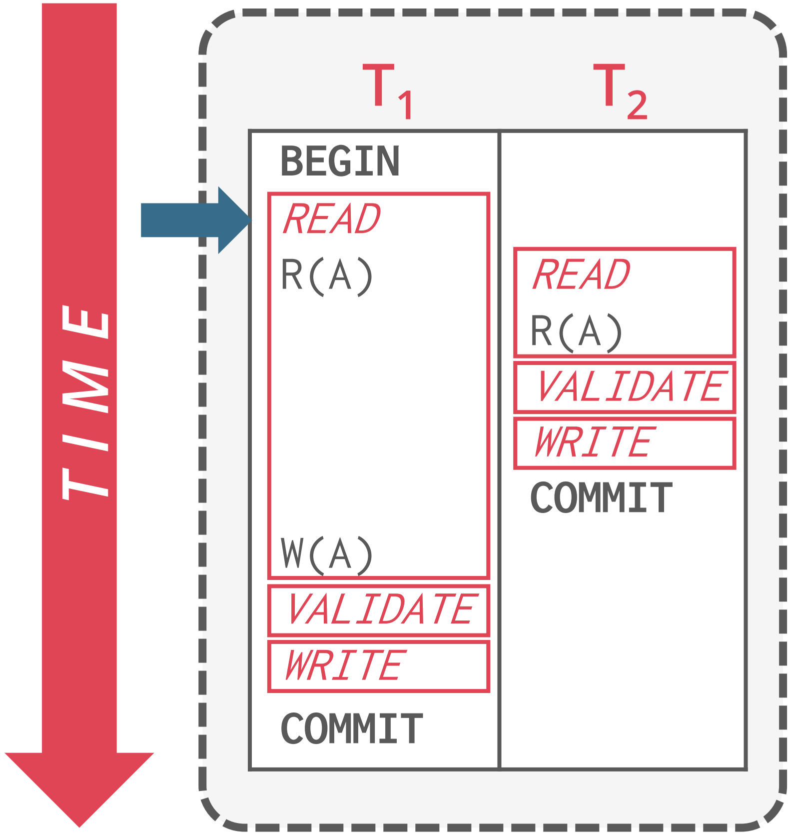
#3 – Write Phase

If validation succeeds, apply private changes to database

Otherwise, abort and restart the txn

OCC – Example

Schedule



Database

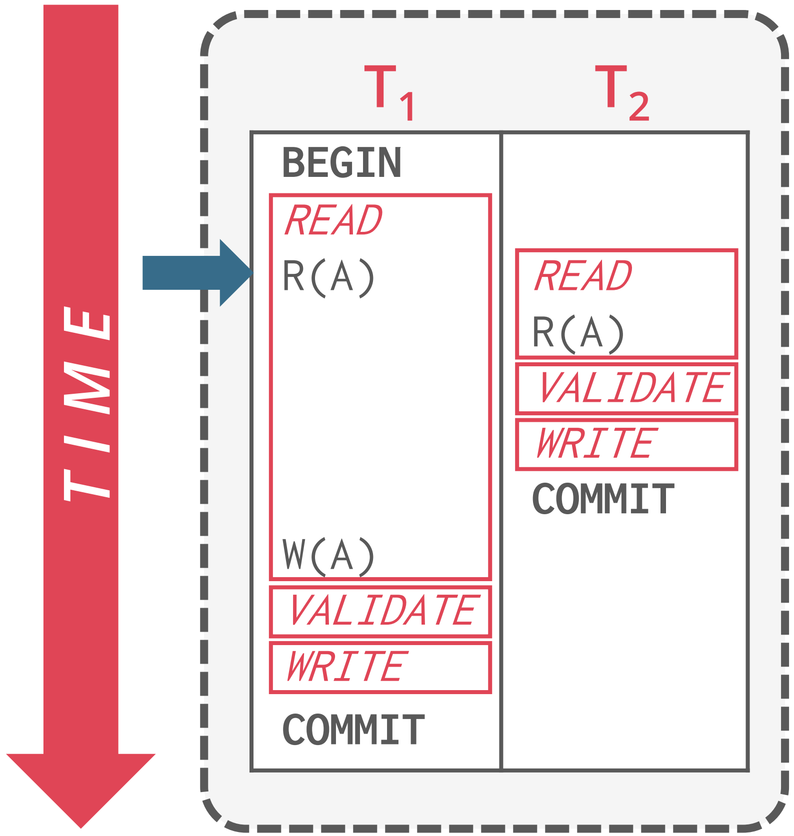
Object	Value	WTS
A	123	0

T_1 Workspace

Object	Value	WTS

OCC – EXAMPLE

Schedule



Database

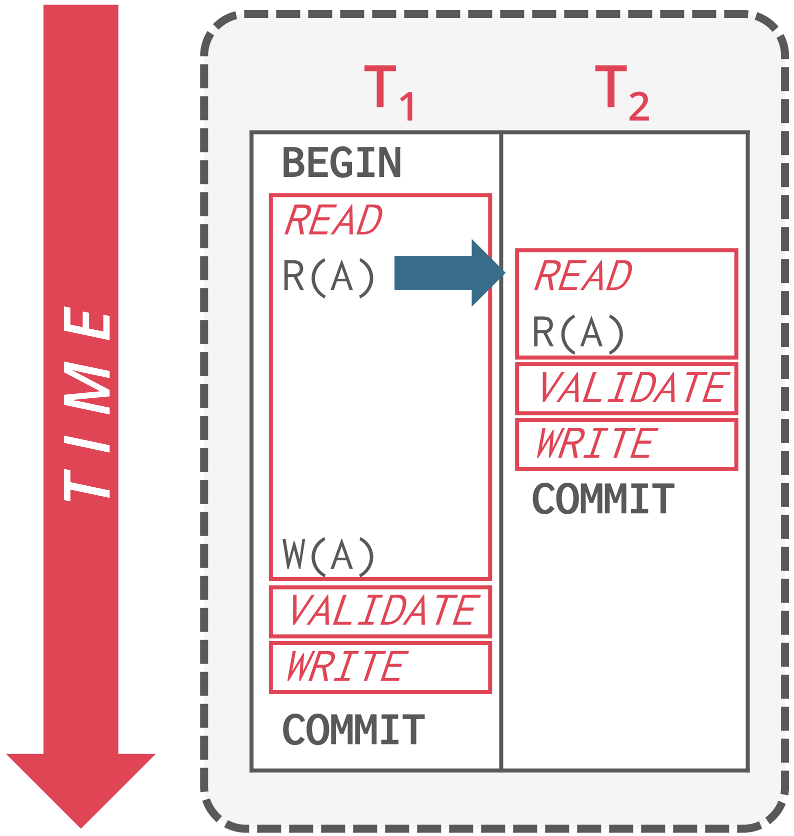
Object	Value	WTS
A	123	0

T$_1$ Workspace

Object	Value	WTS
A	123	0

OCC – EXAMPLE

Schedule



Database

Object	Value	WTS
A	123	0

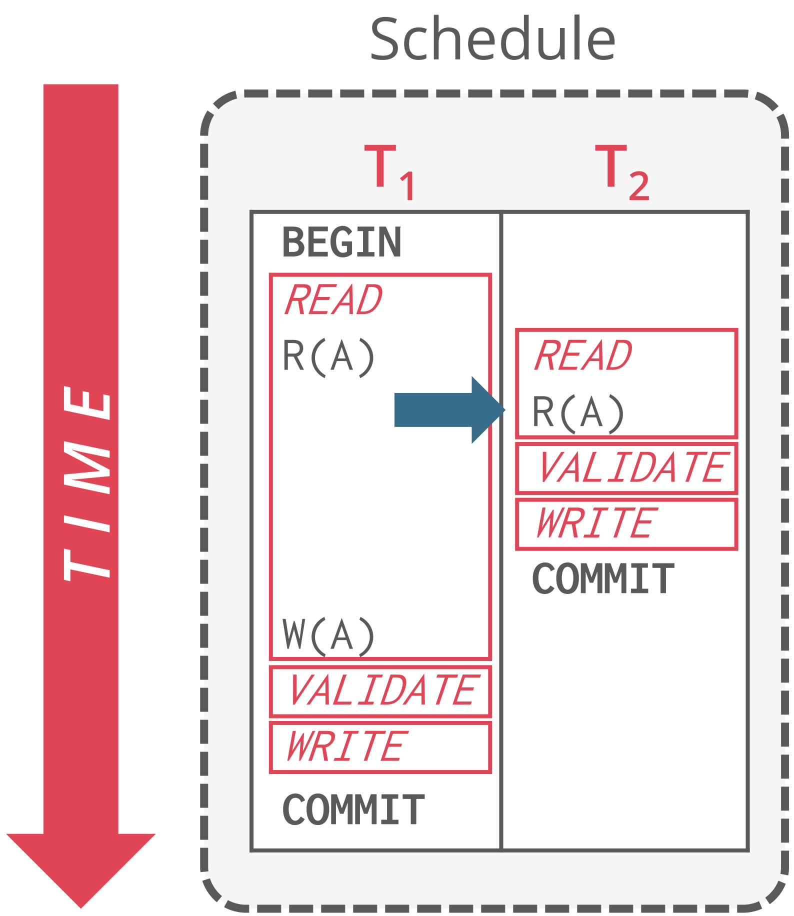
T_1 Workspace

Object	Value	WTS
A	123	0

T_2 Workspace

Object	Value	WTS

OCC – EXAMPLE



Database

Object	Value	WTS
A	123	0

T_1 Workspace

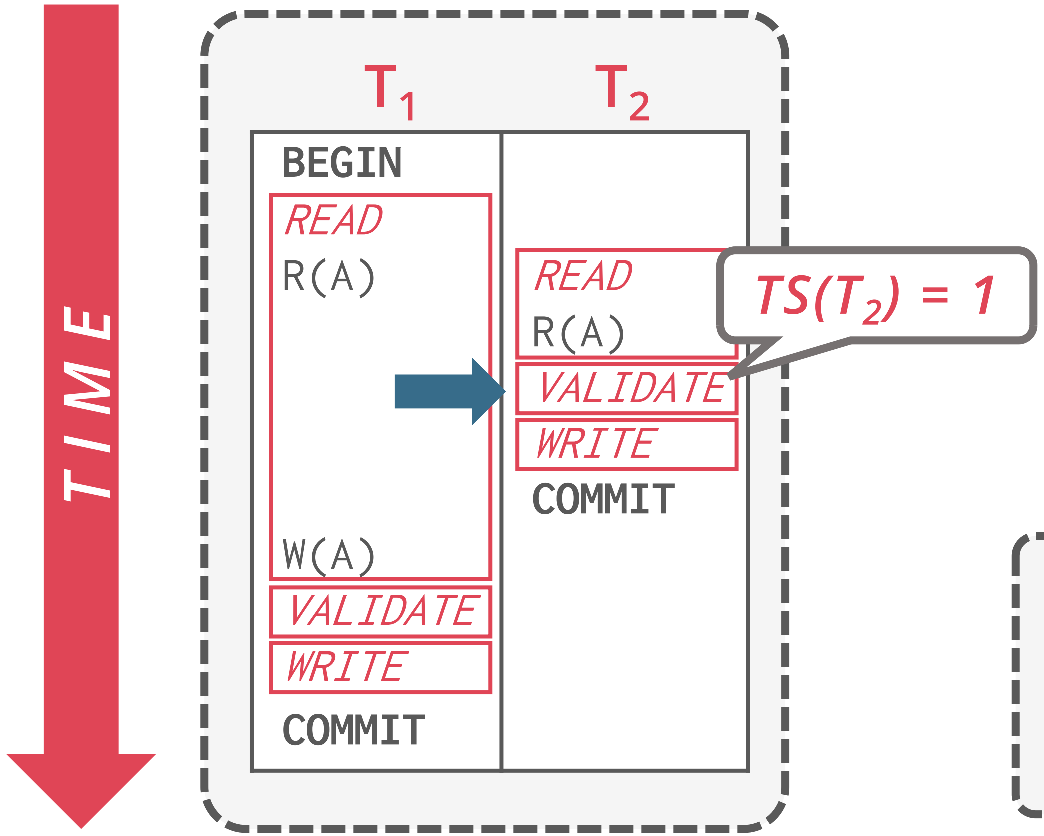
Object	Value	WTS
A	123	0

T_2 Workspace

Object	Value	WTS
A	123	0

OCC – EXAMPLE

Schedule



Database

Object	Value	WTS
A	123	0

T_1 Workspace

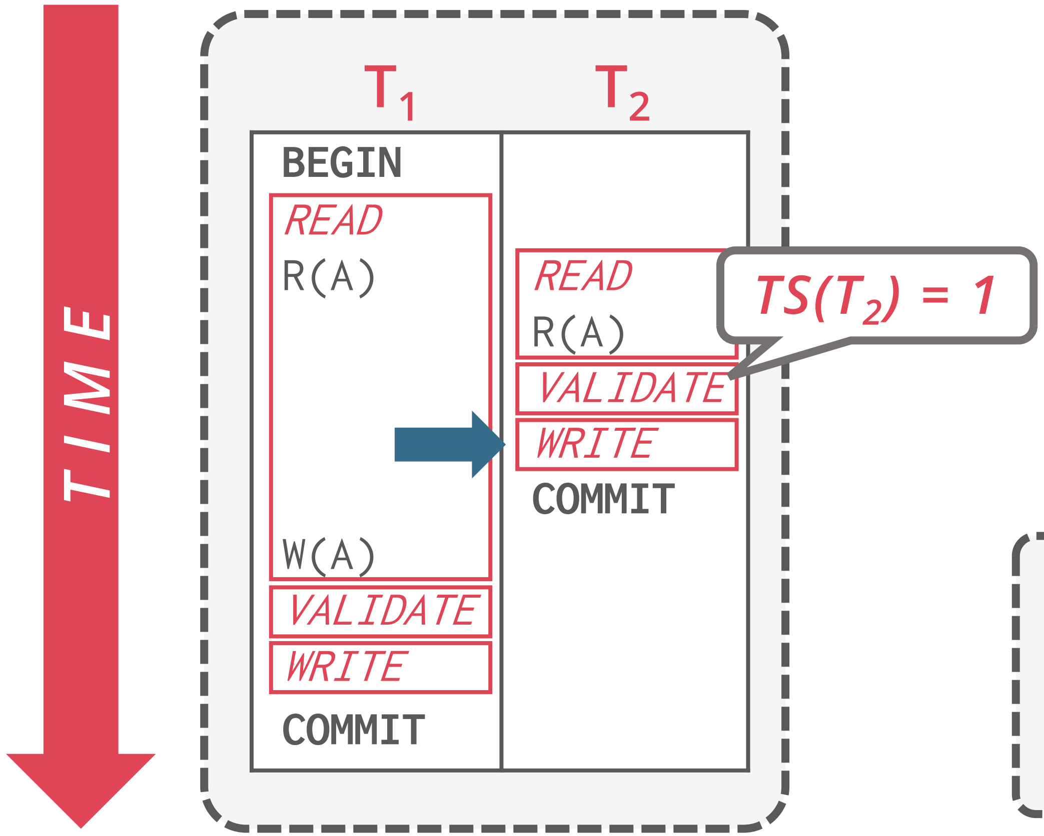
Object	Value	WTS
A	123	0

T_2 Workspace

Object	Value	WTS
A	123	0

OCC – Example

Schedule



Database

Object	Value	WTS
A	123	0

T_1 Workspace

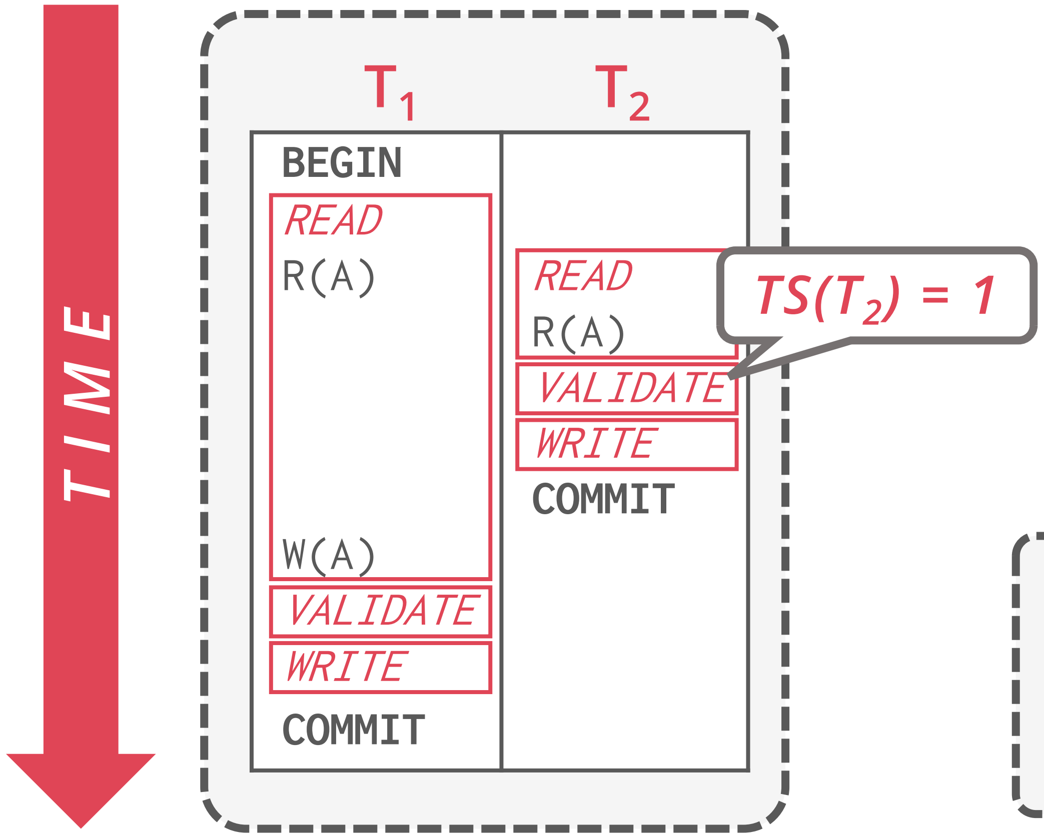
Object	Value	WTS
A	123	0

T_2 Workspace

Object	Value	WTS
A	123	0

OCC – EXAMPLE

Schedule



Database

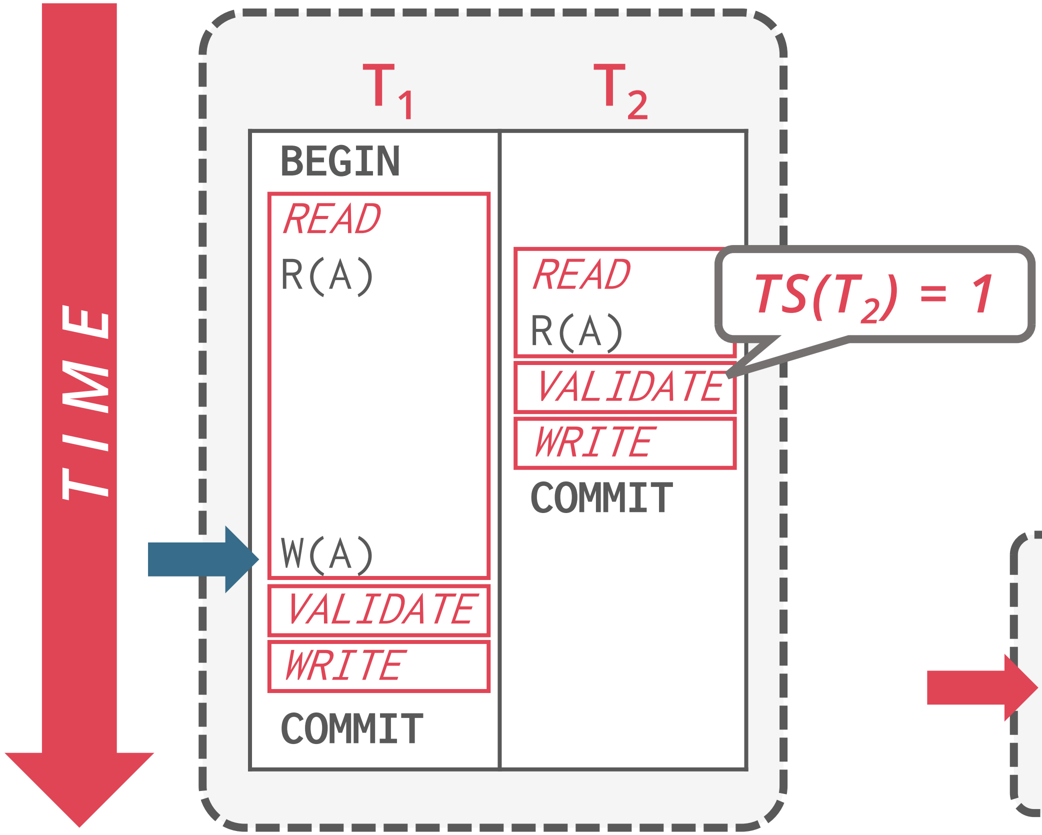
Object	Value	WTS
A	123	0

T_1 Workspace

Object	Value	WTS
A	123	0

OCC – Example

Schedule



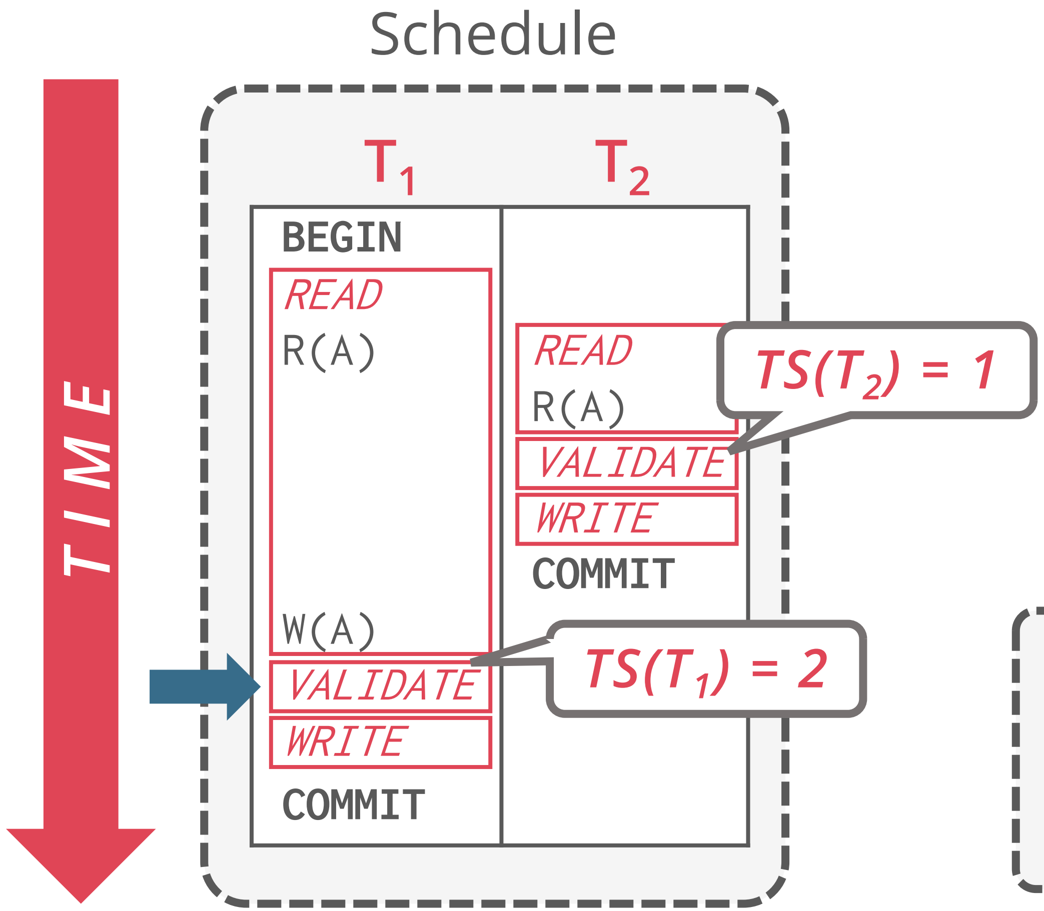
Database

Object	Value	WTS
A	123	0

T_1 Workspace

Object	Value	WTS
A	456	∞

OCC – EXAMPLE



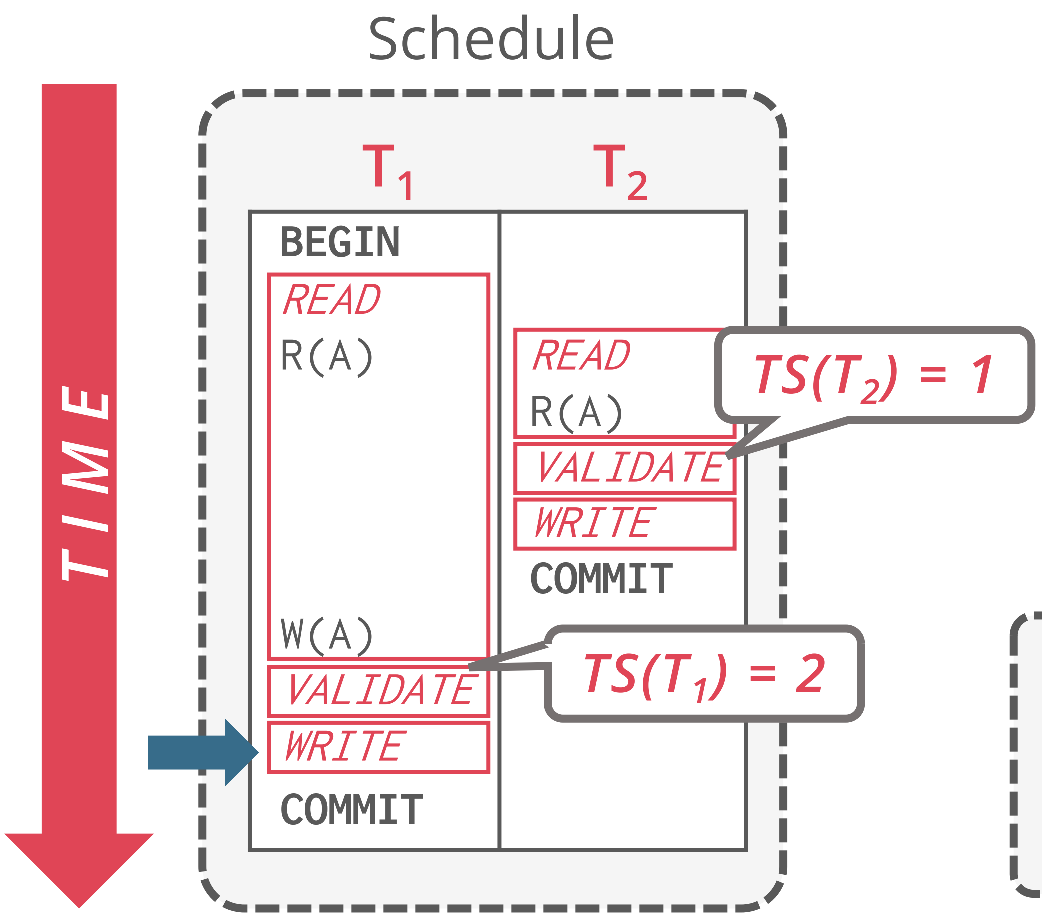
Database

Object	Value	WTS
A	123	0

T_1 Workspace

Object	Value	WTS
A	456	∞

OCC – EXAMPLE



Database

Object	Value	WTS
A	456	2

T_1 Workspace

Object	Value	WTS
A	456	∞

OCC – Validation Phase

The DBMS must guarantee only serializable schedules are permitted

Each txn's timestamp is assigned at the beginning of validation phase

Check the timestamp ordering of the committing txn T_j with respect to each **committed** txns T_i such that $TS(T_i) < TS(T_j)$

Record read set and write set while txns are running

- **ReadSet(T_i):** set of objects read by txn T_i
- **WriteSet(T_i):** set of objects modified by T_i

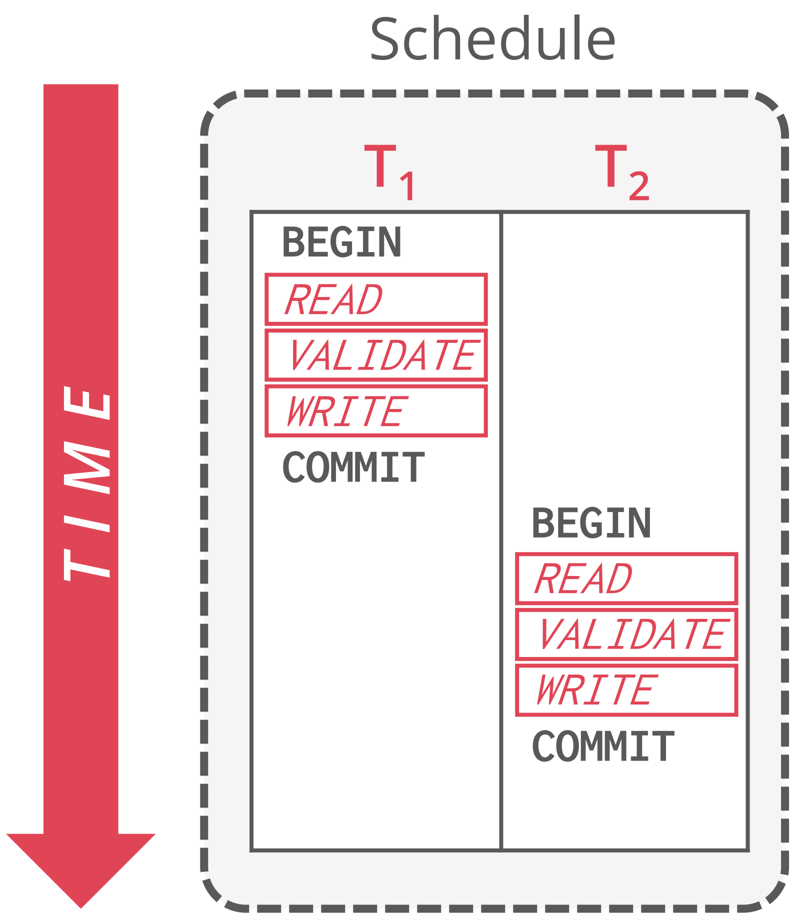
Validation Steps

If $TS(T_i) < TS(T_j)$, then one of the following three conditions must hold:

1. T_i completes all three phases before T_j begins
2. T_i completes before T_j starts its **Write** phase, and
   WriteSet(T_i) $\cap$ **ReadSet(T_j)** $= \emptyset$
3. T_i completes its **Read** phase before T_j completes its **Read** phase, and
   WriteSet(T_i) $\cap$ **ReadSet(T_j)** $= \emptyset$, and
   WriteSet(T_i) $\cap$ **WriteSet(T_j)** $= \emptyset$

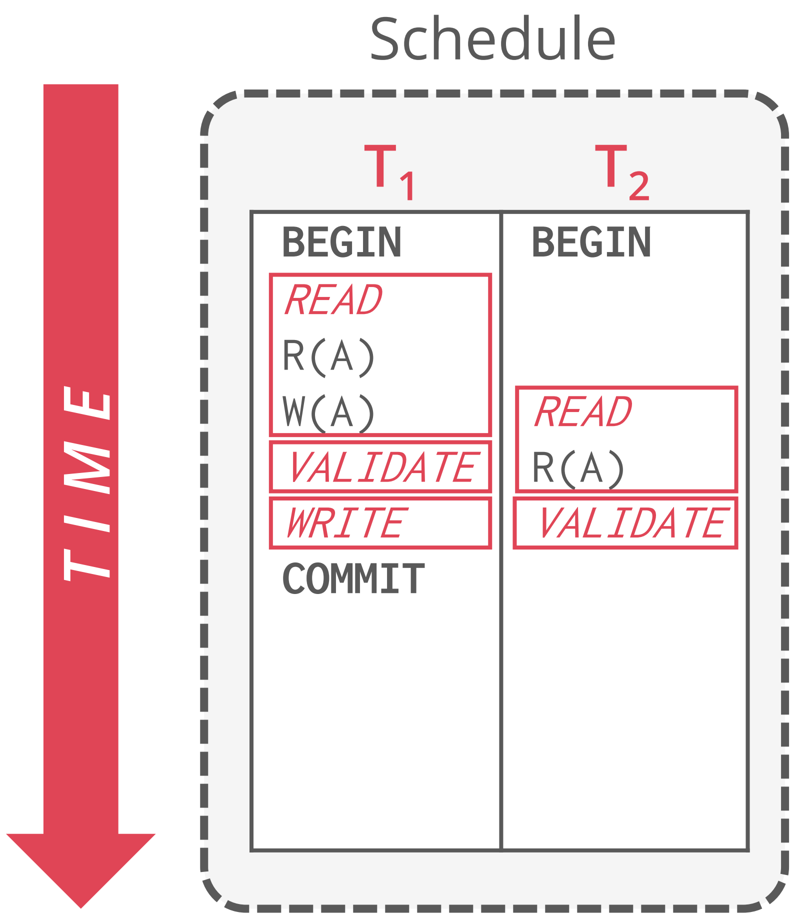
Validation and **Writes** phases are executed inside a critical section

OCC – VALIDATION STEP #1



T_i completes all three phases before T_j begins

OCC – VALIDATION STEP #2



For all T_i and T_j such that $TS(T_i) < TS(T_j)$:

T_i completes before T_j starts its **Write** phase, and

T_i does not write to any object read by T_j

WriteSet(T_i) $\cap$ **ReadSet(T_j)** $= \emptyset$

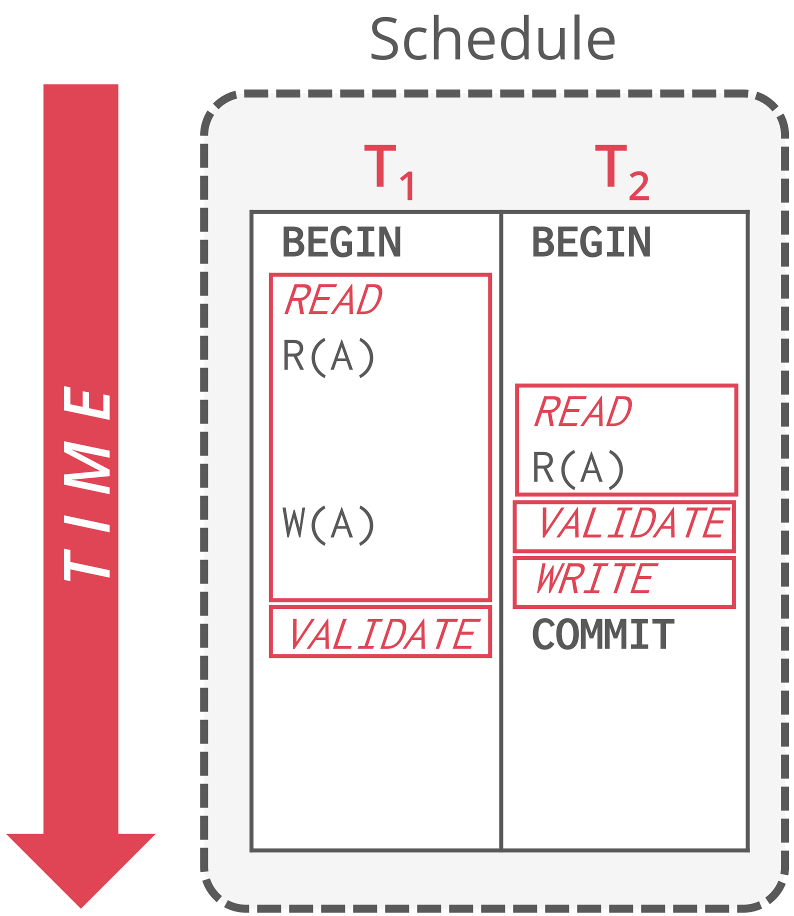
Example:

$TS(T_1) = 1$, $TS(T_2) = 2$ (based on validation starts)

Validation for T_2 fails because

WriteSet(T_1) $\cap$ **ReadSet(T_2)** $= \{A\}$

OCC – VALIDATION STEP #2



For all T_i and T_j such that $TS(T_i) < TS(T_j)$:

T_i completes before T_j starts its **Write** phase, and

T_i does not write to any object read by T_j

WriteSet$(T_i) \cap$ **ReadSet**$(T_j) = \emptyset$

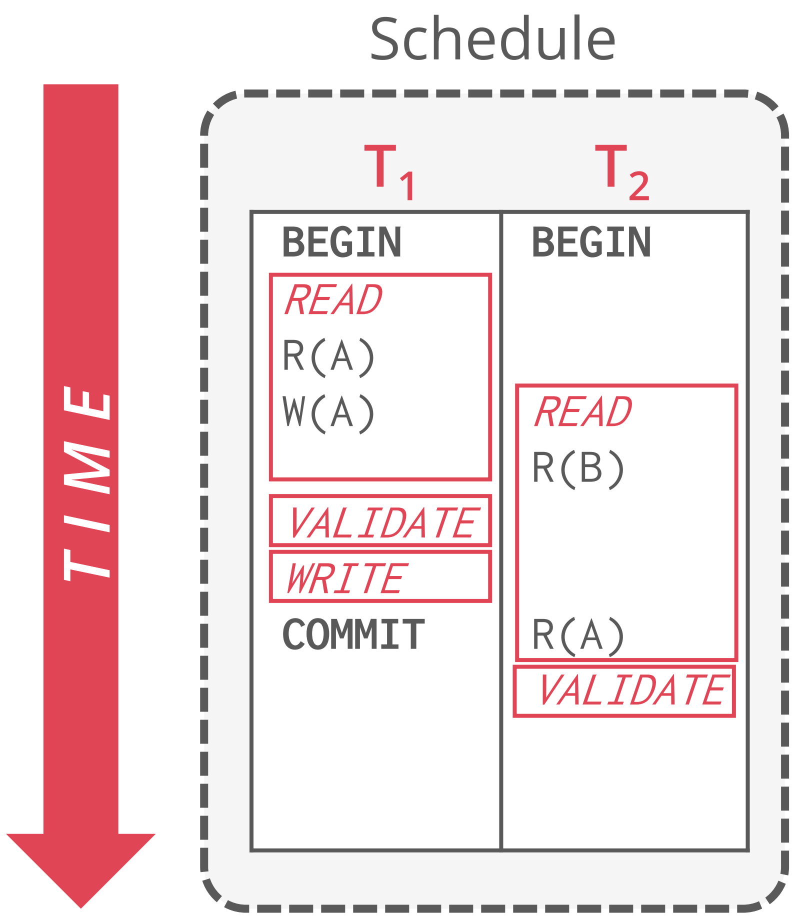
Example:

$TS(T_1) = 2$, $TS(T_2) = 1$ (based on validation starts)

Validation for T_1 is successful because

WriteSet$(T_2) \cap$ **ReadSet**$(T_1) = \emptyset$

OCC – Validation Step #3



For all T_i and T_j such that $TS(T_i) < TS(T_j)$:

T_i completes its **Read** before T_j completes its **Read**, and

T_i does not write to any object read or written by T_j

WriteSet(T_i) ∩ **ReadSet**(T_j) = ∅

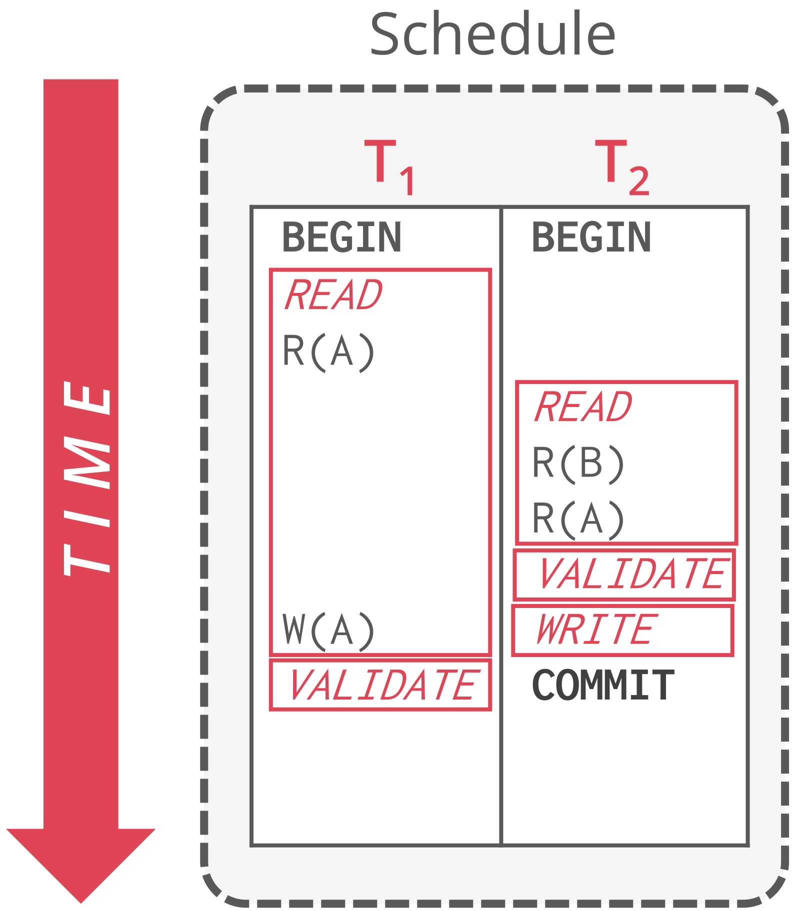
WriteSet(T_i) ∩ **WriteSet**(T_j) = ∅

Example:

$TS(T_1) = 1$, $TS(T_2) = 2$ (based on validation starts)

Validation for T_2 fails: **WriteSet**(T_1) ∩ **ReadSet**(T_2) = {A}

OCC – Validation Step #3



For all T_i and T_j such that $TS(T_i) < TS(T_j)$:

T_i completes its **Read** before T_j completes its **Read**, and

T_i does not write to any object read or written by T_j

WriteSet(T_i) $\cap$ **ReadSet(T_j)** = Ø

WriteSet(T_i) $\cap$ **WriteSet(T_j)** = Ø

Example:

$TS(T_1) = 2$, $TS(T_2) = 1$ (based on validation starts)

Validation for T_1 is successful since **WriteSet(T_2)** = Ø

OCC – Observations

OCC works well when the # of conflicts is low

- All txns are read-only (ideal)
- Txns access disjoint subsets of data

If the database is large and the workload is not skewed, then there is low probability of conflict, so again locking is wasteful

Performance issues

- High overhead of copying data locally
- Validation / Write phase bottlenecks
- Aborts are more wasteful than in 2PL as they only occur **<u>after</u>** a txn has executed

Conclusion ☺

Crazy Concurrency