Database Systems
Lecture19 – Chapter 18: Concurrency Control
(Part II: Timestamp-based Protocol)

Beomseok Nam (남범석)
bnam@skku.edu

Observation

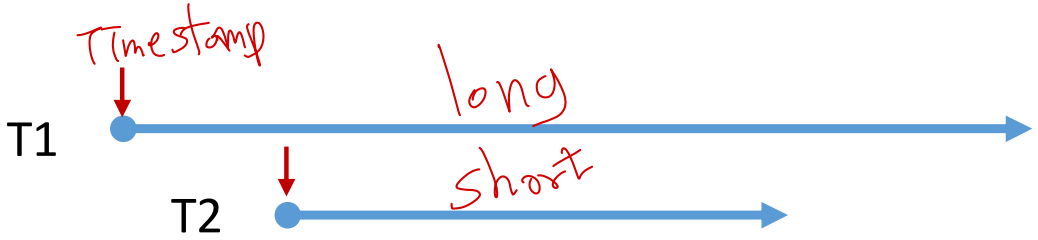
- If ***conflicts*** between transactions are ***rare*** and most transactions are ***short-lived***, then acquiring locks adds unnecessary overhead.

lock은 성능↓
⇒ 성능을 향상시켜
⇒ expensive
(∵ OS에 요청을 보내는 것이라)

- Concurrency control protocol could be optimized for the no-conflict case.

Timestamp-Based Protocols

- Each transaction T_i has a unique timestamp TS(T_i).
  - *Timestamp* is assigned when *a transaction starts*



  - Newer transactions have timestamps greater than earlier ones
    - TS(old T1) < TS(new T2) (TS: timestamp)
  - Timestamp could be based on a logical counter, not real time.

=Tx 시작시점

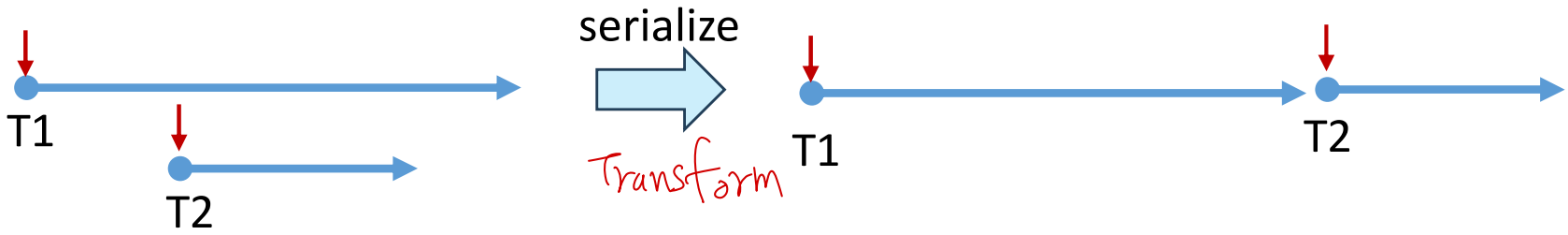
논리적인 증가값 (Txid와 같은)

- Timestamp-Based Protocols
  - Timestamp-Ordering Protocol (TSO)
  - Validation-based Protocol (OCC)

최적화기법

Timestamp-Based Protocols

- ***Timestamp-Ordering Protocol*** (***TSO***) serializes transactions such that
  - **timestamp order = serializability order**
  - If TS(T1) < TS(T2), the schedule is equivalent to the serial schedule where T1 → T2



- ***Validation-based Protocol*** (***OCC***) serializes transactions such that
  - **validation time order = serializability order**

Timestamp-Ordering Protocol

The **timestamp ordering (TSO) protocol**

- Every object Q is tagged with timestamp of the last txn that successfully did read/write:
  - **W-timestamp**(*Q*) : Write timestamp on Q
  - **R-timestamp**(*Q*) : Read timestamp on Q

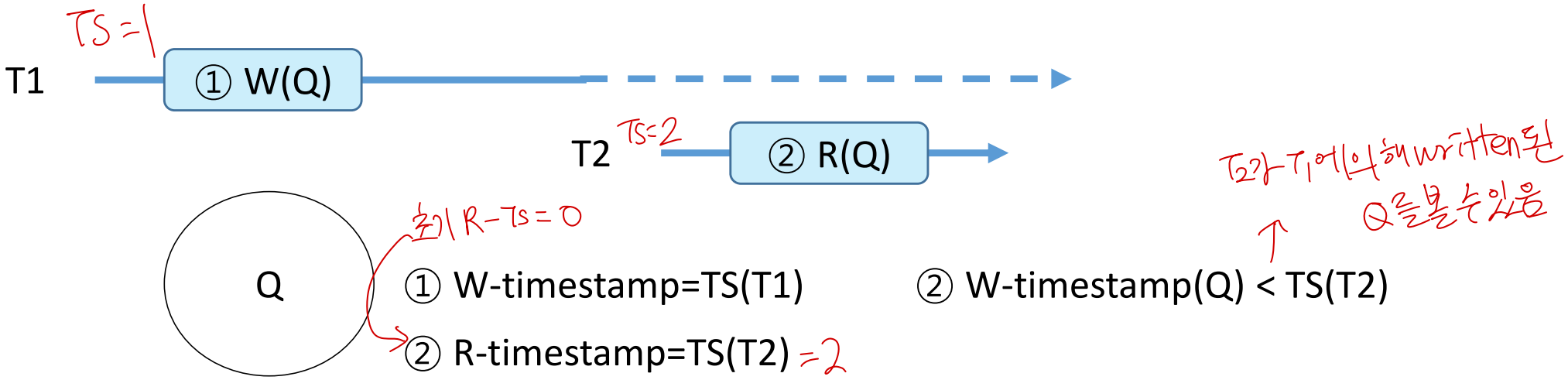
- Rules for Read/Write
  - Conflicting operations are executed in timestamp order
  - Out of order operations trigger transaction rollback

Timestamp-Ordering Protocols (Cont.)

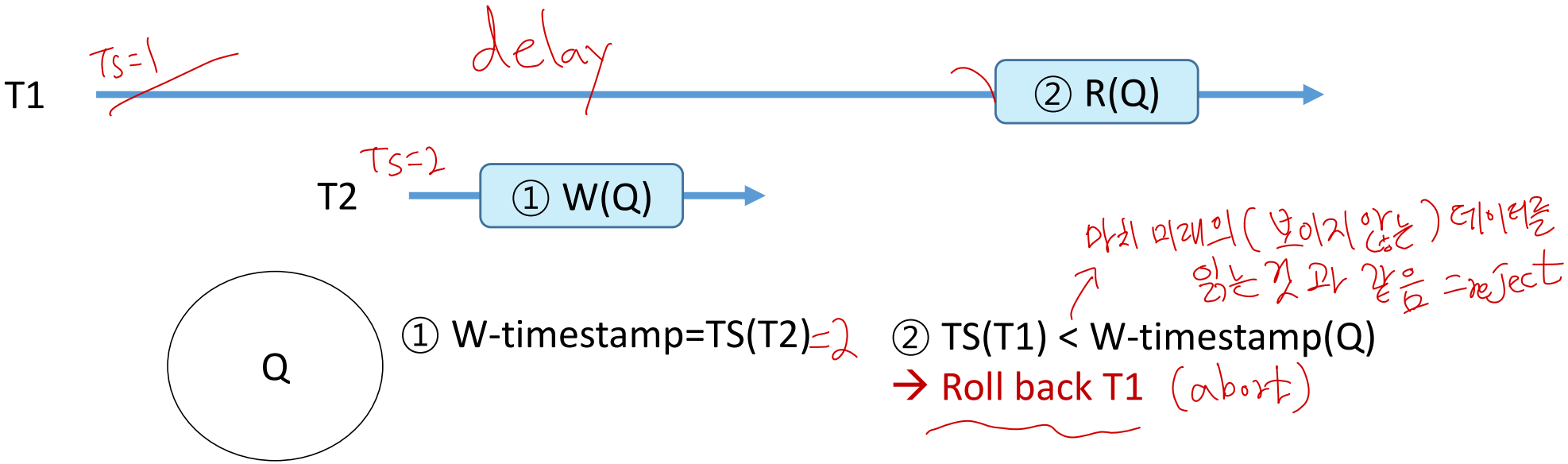
- Suppose a transaction T_i issues a **read**(Q)

  1. If **W-timestamp**(Q) ≤ **TS**(T_i)
     - the **read** is **allowed**, and
     - R-timestamp(Q) is set to max(R-timestamp(Q), TS(Ti)).
  2. If **TS**(T_i) ≤ **W-timestamp**(Q)
     - T_i needs to read Q that was already overwritten.
     - **Reject read** and **rollback** T_i

Timestamp-Ordering Protocols (Cont.)



VS.



Timestamp-Ordering Protocols (Cont.)

- Suppose that transaction T_i issues **write**(Q).

  1. If TS(T_i) < R-timestamp(Q)
     - DBMS assumed that T_i would **never write Q**
     - Younger transaction already **read Q**

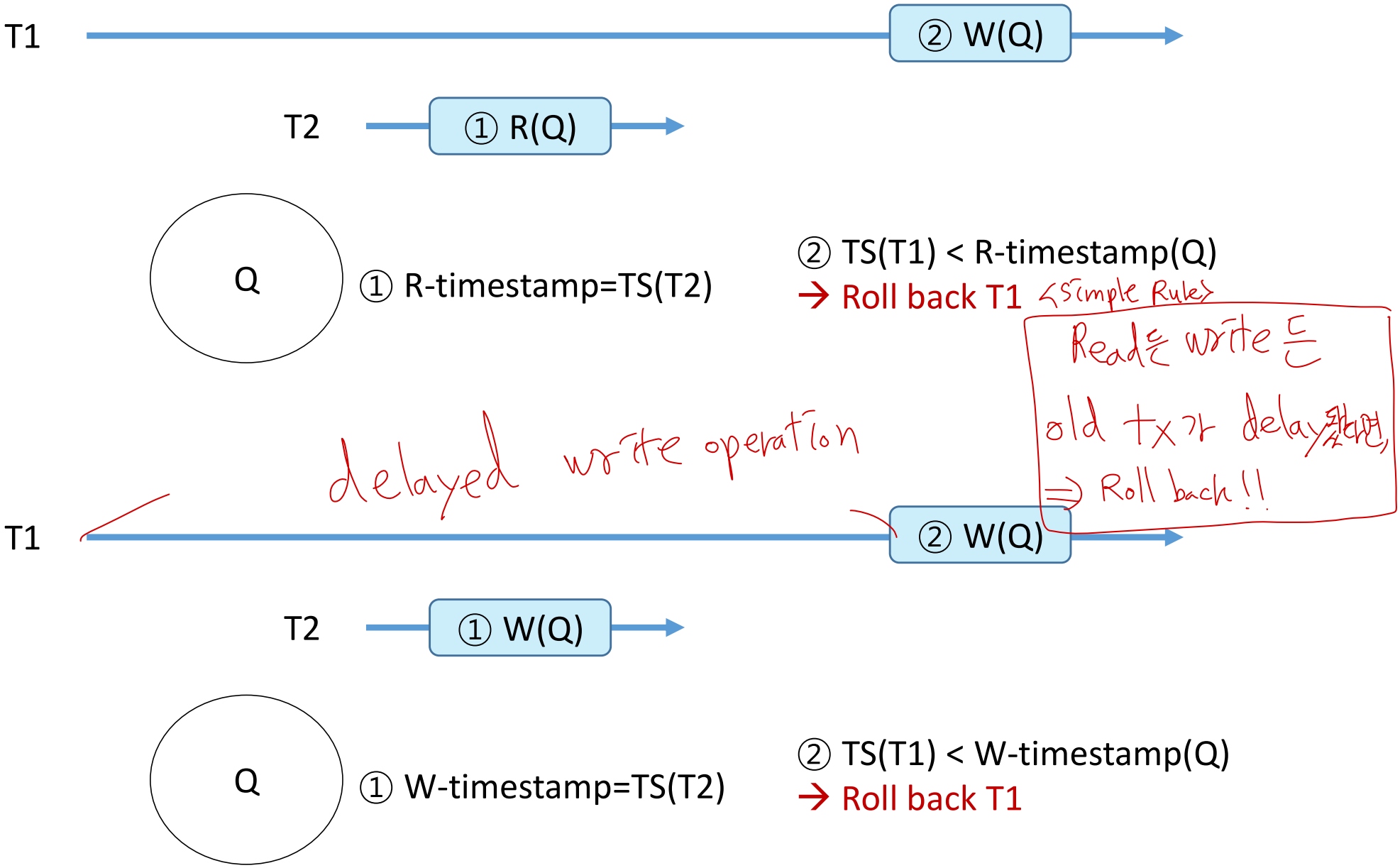
     ➢ **Reject write** and **rollback T_i**

  2. If TS(T_i) < W-timestamp(Q)
     - DBMS assumed that T_i would **never write Q**
     - Younger transaction already **wrote Q.**

     ➢ **Reject write** and **rollback T_i**

  3. Otherwise
     - **Allow write**
     - Set W-timestamp(Q) to TS(T_i)

Timestamp-Ordering Protocols (Cont.)



Example: Schedule Under TSO

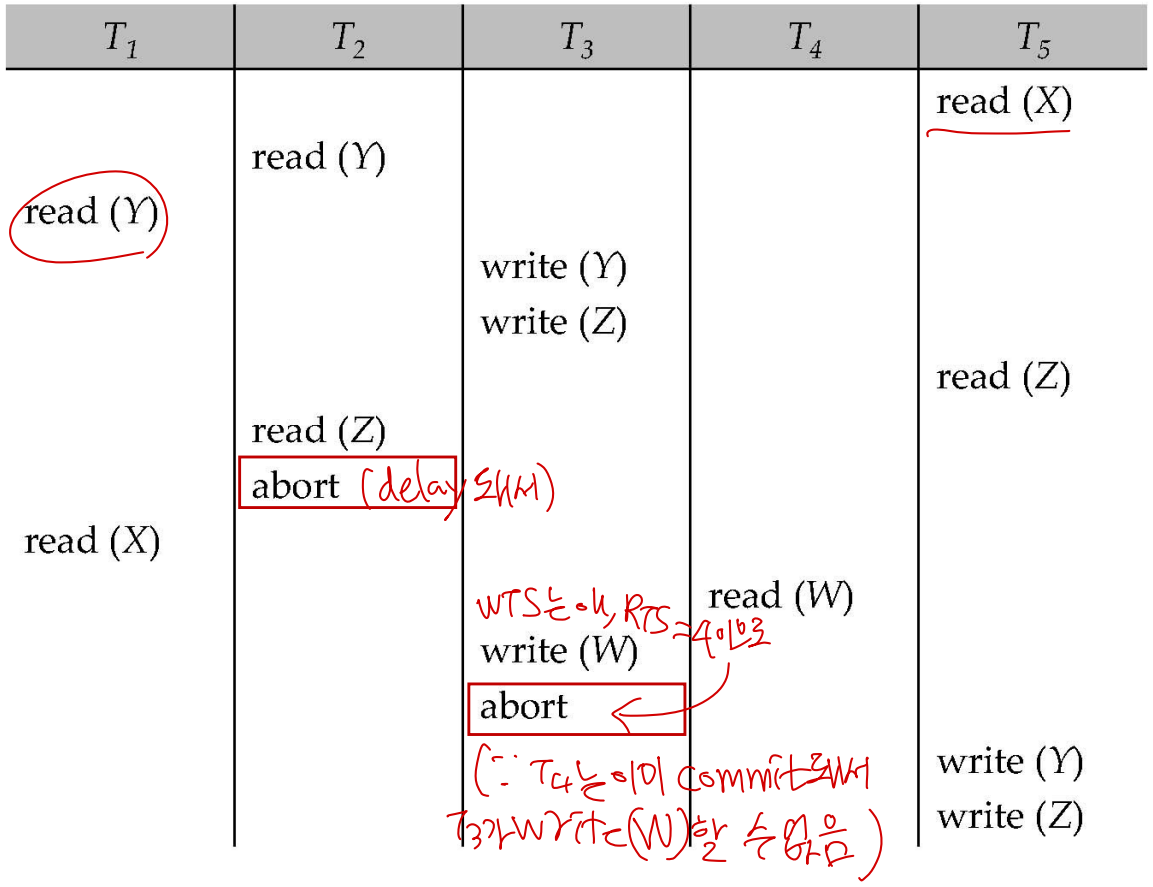
- Is this schedule valid under TSO?

  Assume TS(T_{25}) = 25 and
  TS(T_{26}) = 26

	R-TS	W-TS
A	0	0
B	0	0

T_{25}	T_{26}
read(B)	
	read(B)
	$B := B - 50$
	write(B)
read(A)	
	read(A)
display($A + B$)	
	$A := A + 50$
	write(A)
	display($A + B$)

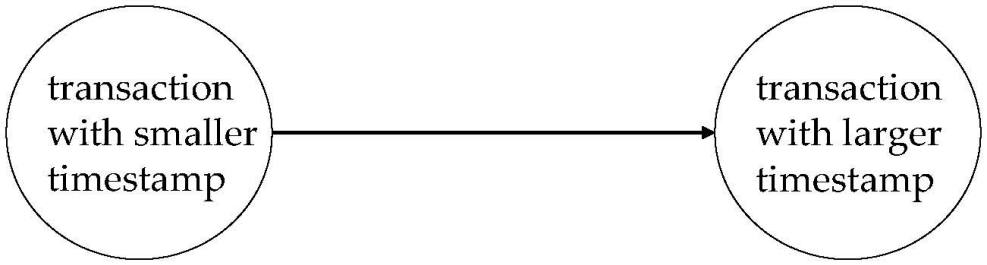
Another Example Under TSO



	R-TS	W-TS
X	0	0
Y	0	0
Z	0	0
W	0	0

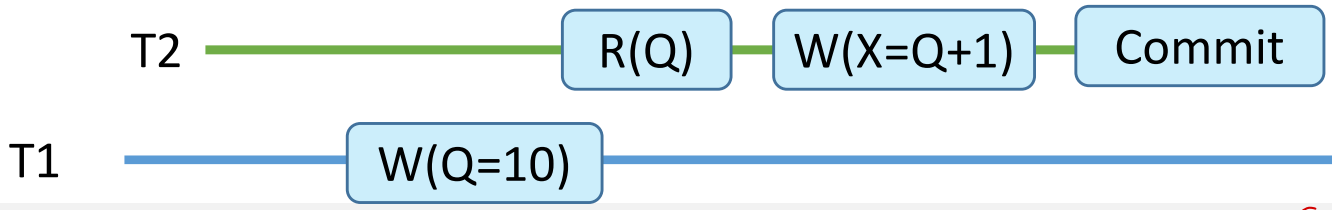
Correctness of Timestamp-Ordering Protocol

- TSO **guarantees serializability**
  - All edges in the precedence graph follow timestamp order
  - No cycle in the graph



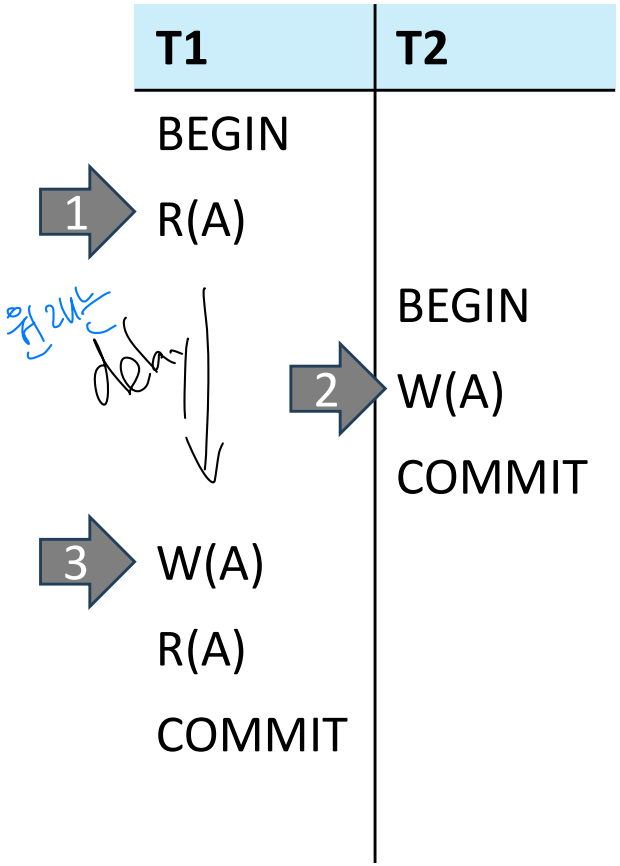
Tx간의 순서그들의 Ts값순서에 따라 결정된다
⇒ serializability을 보장한다!

- TSO ensures **freedom from deadlock**
  - no transaction ever waits
- Limitations
  - may **not** be **cascade-free**
  - may **not** be **recoverable**
    - A schedule is recoverable if a transaction commits only after the transactions it depends on have committed.



⇒ 이상황이면 T2는 회복불가능 (Commit을 나중에 하더라도 Q값때문에 T2도 Abort해야함)
cascade-rollback issue

Yet Another Example Under TSO



1

	R-TS	W-TS
A	1	0

2

	R-TS	W-TS
A	1	2

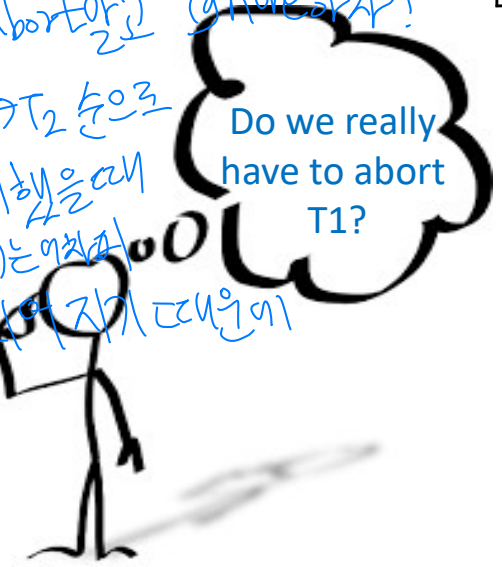
3

Violation:
TS(T1) < W-TS(Q)

Yet Another Example Under TSO

Blind write개념 (전데이터가뭔지 상관없이 덮어씀)

T1	T2
BEGIN	
1 R(A)	
	BEGIN
	2 W(A)
	COMMIT
abort 3 W(A)	← abort말고 Ignore하자!
R(A)	T1→T2 순으로
COMMIT	해치웠을때
	T1 W(A)는 예전에
	덮어쓰여졌어야 했는데



1

	R-TS	W-TS
Q	1	0

2

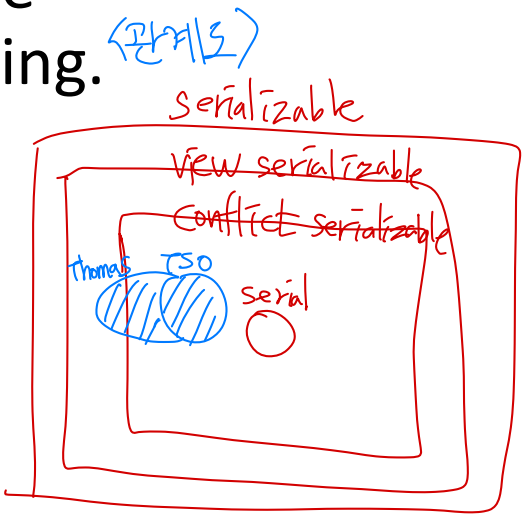
	R-TS	W-TS
Q	1	2

Violation:
TS(T1) < W-TS(Q)

Thomas' Write Rule

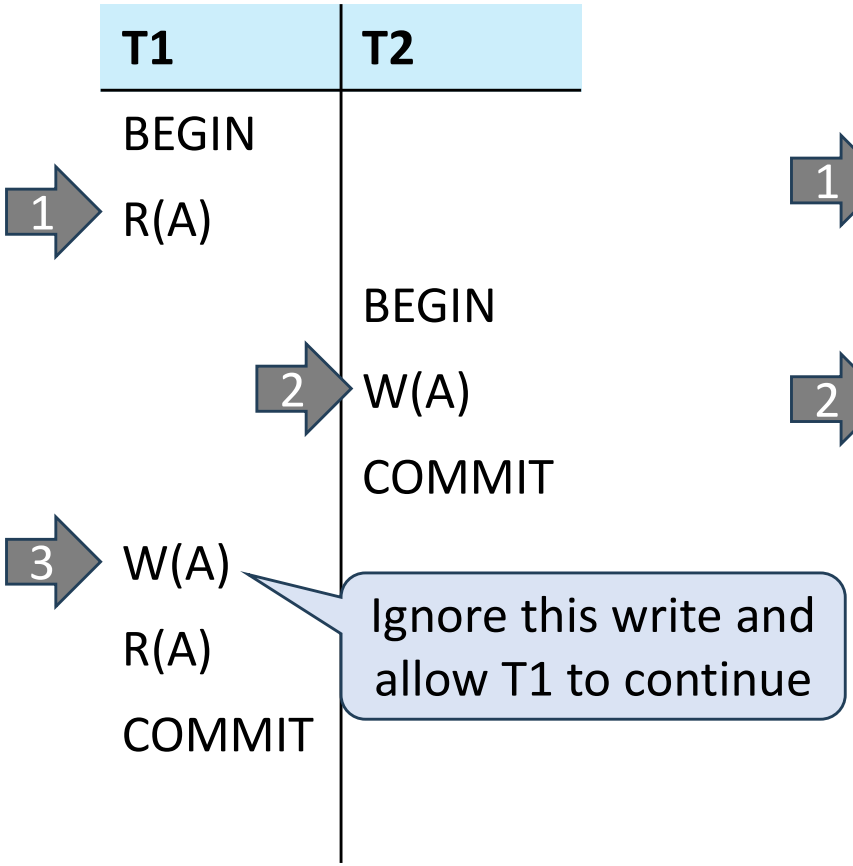
young 타x의 write 가있다면
old 타x의 write operation을 무시해도된다!

- if $TS(T_i) < R\text{-timestamp}(Q)$
  - Abort and restart T_i
- if $TS(T_i) < W\text{-timestamp}(Q)$
  - ***Thomas Write Rule***: Ignore the write to allow the transaction to continue executing without aborting. (현재는 abort 이지만, 무시함)
  - This violates timestamp order of T_i.
- Else
  - Allow T_i to write Q and update W-timestamp(Q)



- Thomas' Write Rule allows greater potential concurrency.
  - Allows some view-serializable schedules that are not conflict-serializable.

Thomas' Write Rule



1

	R-TS	W-TS
Q	1	0

2

	R-TS	W-TS
Q	1	2

Validation-based Concurrency Control (Optimistic Concurrency Control - OCC)

Validation-Based Protocol (OCC)

- Idea: can we use **commit time as serialization order**?
- To do so:
  - Postpone writes to end of transaction (연기하다)
  - Keep track of data items read/written by transaction
  - **Validation** performed at commit time, detect any out-of-serialization order reads/writes ⇒ conflict를 commit 전에 valid한지 판단
- Also called as **Optimistic Concurrency Control (OCC)** since transaction executes fully in the hope that all will go well during validation

비교
Ts0 r() w() ⇒ check Ts : Ts를 비교해 매순간 판단
OCC ⟶ commit : 모두 허용해주고 commit전에 conflict가 없는지 확인 후 없다면 commit
Validation

Validation-Based Protocol (OCC)

- Execution of transaction T_i is done in three phases.

1. Read and execution phase:

Transaction T_i writes only temporary local variables

2. Validation phase

Transaction T_i performs a "validation test" to determine if local variables can be written without violating serializability.

3. Write phase

If T_i is validated, the updates are applied to the database; otherwise, T_i is rolled back.

- We assume for simplicity that the validation and write phase occur together, atomically and serially
  - I.e., only one transaction executes validation/write at a time.
  - But, the three phases of concurrently executing transactions can be interleaved.

Validation-Based Protocol (OCC)

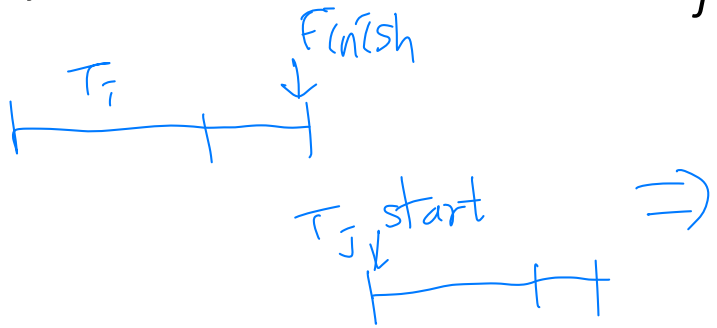
- Each transaction T_i has 3 timestamps
  - **StartTS**(T_i) : the time when T_i started its execution
  - **ValidationTS**(T_i): the time when T_i entered its validation phase
  - **FinishTS**(T_i) : the time when T_i finished its write phase
- Validation tests use above timestamps and read/write sets to ensure that *serializability order is determined by validation time*
  - Thus, TS(T_i) = ValidationTS(T_i)
- Validation-based protocol gives greater degree of concurrency than locking/TSO if probability of conflicts is low.

Validation Test for Transaction T_j

- If for all T_i with TS (T_i) < TS (T_j) either one of the following condition holds:
  - **finishTS**(T_i) < **startTS**(T_j)
    - execution is not concurrent
  - **startTS**(T_j) < **finishTS**(T_i) < **validationTS**(T_j) **and** T_j does not read any data item written by T_i ,
    - there is no dependency

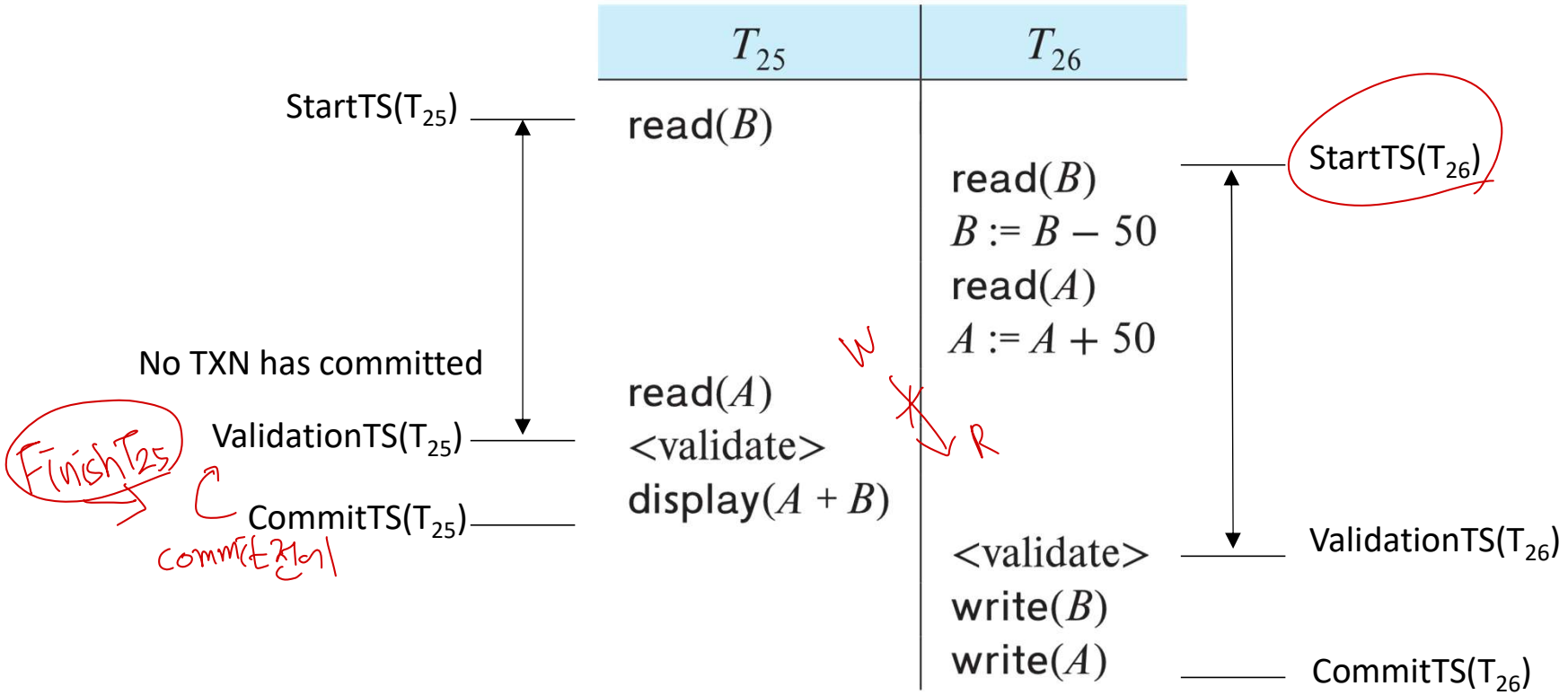
  then validation succeeds and T_j can be committed.

- Otherwise, validation fails and T_j is aborted.



Schedule Produced by Validation

- Example of schedule produced using validation



startTS(T_{26}) < finishTS(T_{25}) < validationTS(T_{26})

T_{25} has committed, but it is read-only

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

Database

Data	Value	W-TS
A	0	0
-	-	

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

Database

Data	Value	W-TS
A	0	0
-	-	

copy

T1's Workspace

Data	Value
A	0
-	-

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

StartTS(T2)=2

Database

Data	Value	W-TS
A	0	0
-	-	

T1's Workspace

Data	Value
A	0
-	-

T2's Workspace

Data	Value
A	0
-	-

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

Write (완전히 완료 Flush 하는 단계)

Database

Data	Value	W-TS
A	0	0
-	-	

T1's Workspace

Data	Value
A	0
-	-

T2's Workspace

Data	Value
A	0
-	-

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

FinTS(T2)=4

Database

Data	Value	W-TS
A	0	0
-	-	

T1's Workspace

Data	Value
A	0
-	-

T2's Workspace

Data	Value
A	0
-	-

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

FinTS(T2)=4

Database

Data	Value	W-TS
A	0	0
-	-	

T1's Workspace

Data	Value
A	1
-	-

Validation-Based Protocol (OCC) - Example

T1	T2
BEGIN Read-Exec Phase	
R(A)	BEGIN Read-Exec Phase
	R(A)
	Validate
	Write
	Commit
W(A)	
Validate	
Write	
Commit	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

FinTS(T2)=4

ValTS(T1)=5

Database

Data	Value	W-TS
A	1	5
-	-	

T1's Workspace

Data	Value
A	1
-	-

$S_{T_1} < F_{T_2} < V_{T_1}$ + W-R 충돌없으면 ⇒ valid

Validation-Based Protocol (OCC) – Another Example

T1	T2
BEGIN Read-Exec Phase	
Read(A)	BEGIN Read-Exec Phase
	Read(A)
	A := A+1
	Validate
	Write(A)
	Commit
A := A+1	
Validate	
Write(A)	
Commit?	

StartTS(T1)=1

Database

Data	Value	W-TS
A	0	
-	-	

Validation-Based Protocol (OCC) – Another Example

T1	T2
BEGIN Read-Exec Phase	
Read(A)	BEGIN Read-Exec Phase
	Read(A)
	A := A+1
	Validate
	Write(A)
	Commit
A := A+1	
Validate	
Write(A)	
Commit?	

StartTS(T1)=1

Database

Data	Value	W-TS
A	0	
-	-	

T1's Workspace

Data	Value
A	0
-	-

Validation-Based Protocol (OCC) – Another Example

T1	T2
BEGIN Read-Exec Phase	
Read(A)	BEGIN Read-Exec Phase
	Read(A)
	A := A+1
	Validate
	Write(A)
	Commit
A := A+1	
Validate	
Write(A)	
Commit?	

StartTS(T1)=1

StartTS(T2)=2

Database

Data	Value	W-TS
A	0	
-	-	

T1's Workspace

Data	Value
A	0
-	-

T2's Workspace

Data	Value
A	0
-	-

Validation-Based Protocol (OCC) – Another Example

T1	T2
BEGIN	
Read-Exec Phase	
Read(A)	BEGIN
	Read-Exec Phase
	Read(A)
	A := A+1
	Validate
	Write(A)
	Commit
A := A+1	
Validate	
Write(A)	
Commit?	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

Database

Data	Value	W-TS
A	0	
-	-	

T1's Workspace

Data	Value
A	0
-	-

T2's Workspace

Data	Value
A	1
-	-

Validation-Based Protocol (OCC) – Another Example

T1	T2
BEGIN Read-Exec Phase	
Read(A)	BEGIN Read-Exec Phase
	Read(A)
	A := A+1
	Validate
	Write(A)
	Commit
A := A+1	
Validate	
Write(A)	
Commit?	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

FinTS(T2)=4

Database

Data	Value	W-TS
A	1	3
-	-	

T1's Workspace

Data	Value
A	0
-	-

T2's Workspace

Data	Value
A	1
-	-

Validation-Based Protocol (OCC) – Another Example

T1	T2
BEGIN Read-Exec Phase	
Read(A)	BEGIN Read-Exec Phase
	Read(A)
	A := A+1
	Validate
	Write(A)
	Commit
A := A+1	
Validate	
Write(A)	
Commit?	

StartTS(T1)=1

StartTS(T2)=2

ValTS(T2)=3

FinTS(T2)=4

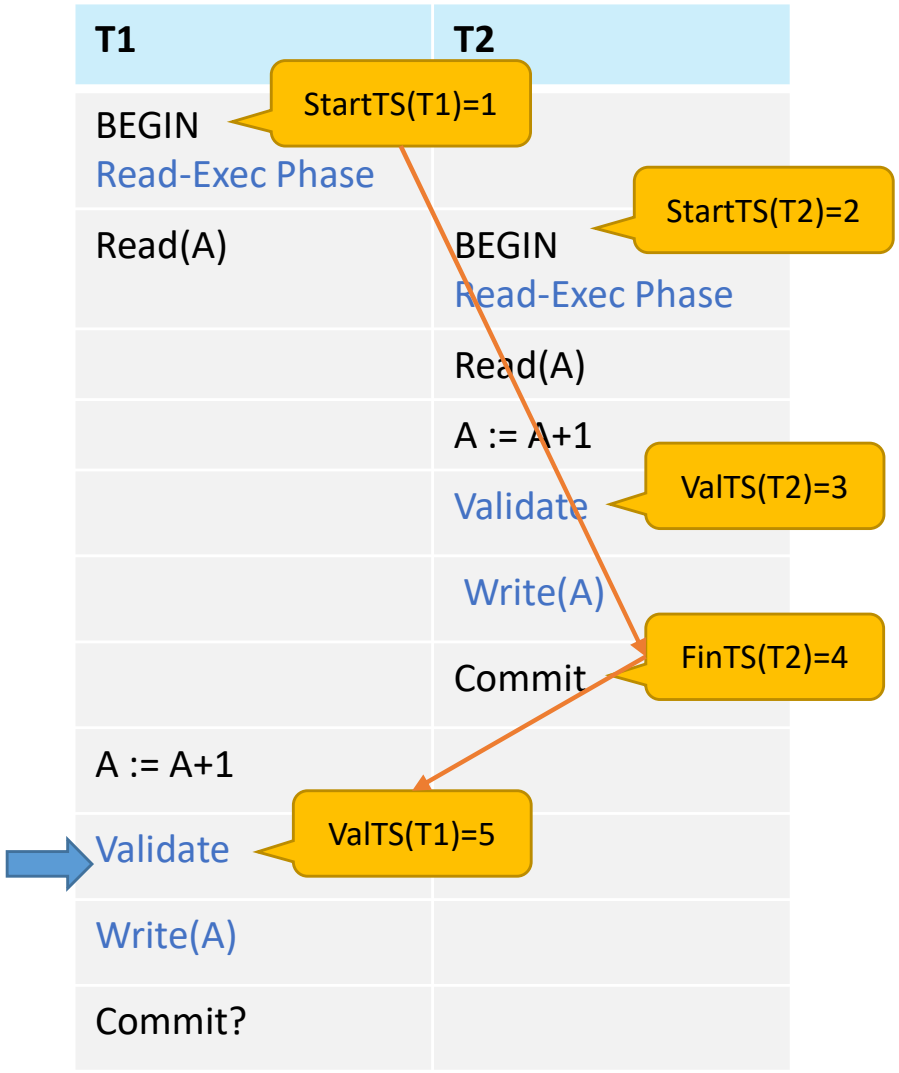
Database

Data	Value	W-TS
A	1	3
-	-	

T1's Workspace

Data	Value
A	1
-	-

Validation-Based Protocol (OCC) – Another Example



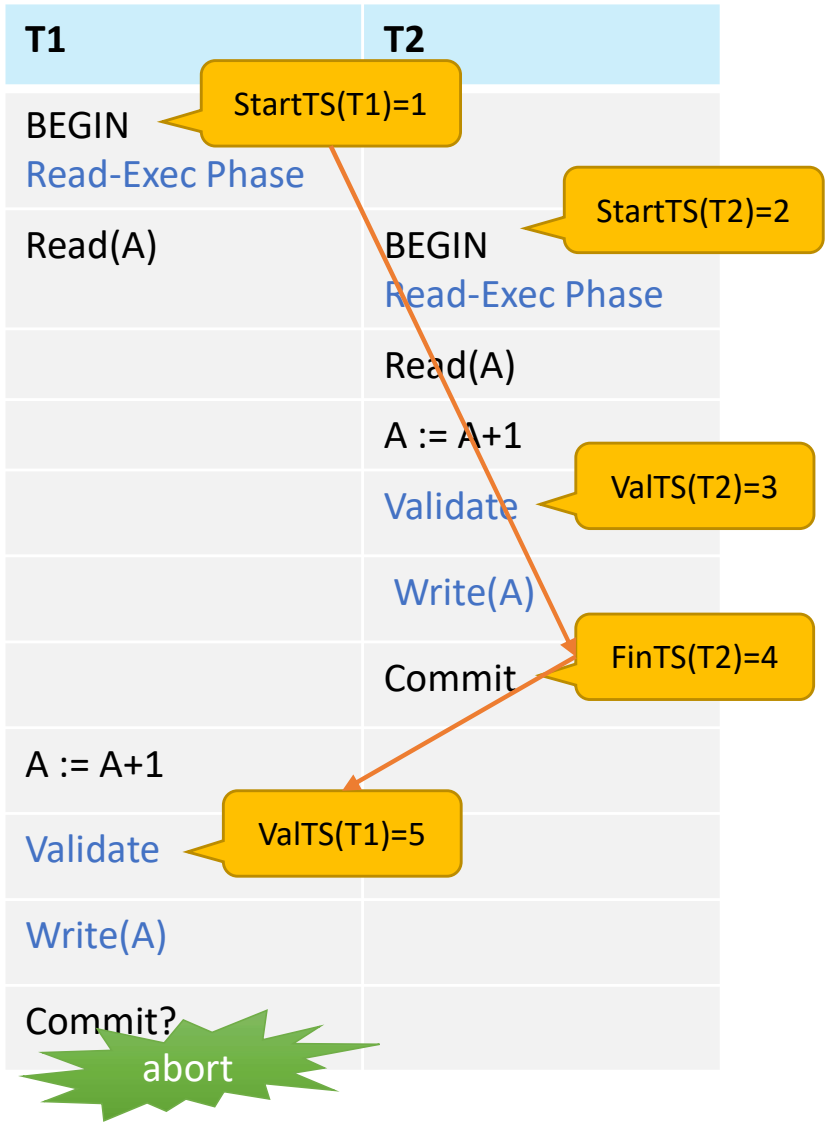
Database

Data	Value	W-TS
A	1	3
-	-	

T1's Workspace

Data	Value
A	1
-	-

Validation-Based Protocol (OCC) – Another Example



Database

Data	Value	W-TS
A	1	3
-	-	

T1's Workspace

Data	Value
A	1
-	-

W-R 의존이 존재

startTS(T_j) < **finishTS**(T_i) < **validationTS**(T_j) **and** T_j has read a data item written by T_i ,

→ There is dependency!