Database Systems
Lecture20 – Chapter 18: Concurrency Control (Part III)

Beomseok Nam (남범석)
bnam@skku.edu

Multi-Version Concurrency Control

Multi-Version Concurrency Control (MVCC)

- The DBMS maintains ***multiple versions*** of an object:

  →When a transaction ***writes*** to an object,

  the DBMS ***creates a new version*** of that object.

  →When a transaction ***reads*** an object,

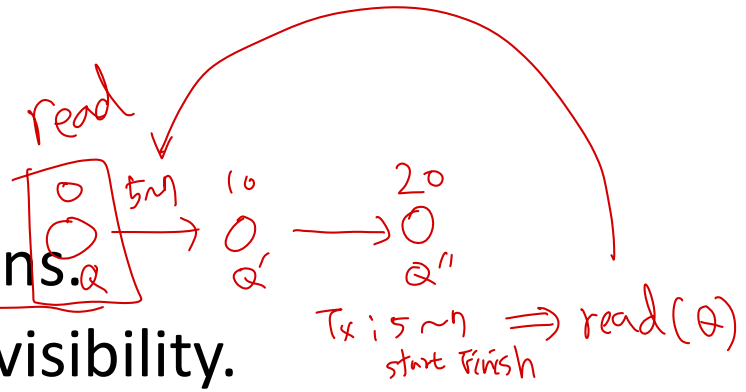
  it ***reads the newest*** version that existed when the transaction started.

※ overwrite 하지않고 Copy-on-write → 옛 young tx 면 latest version 이 아닐수

version 을 남기기

Multi-Version Concurrency Control (MVCC)

- Key ideas:
  - Writers do not block readers.
  - Readers do not block writers.
  - Use timestamps to label versions.
  - Use timestamps to determine visibility.
  - Multi-versioning allows the DBMS to support time-travel queries.



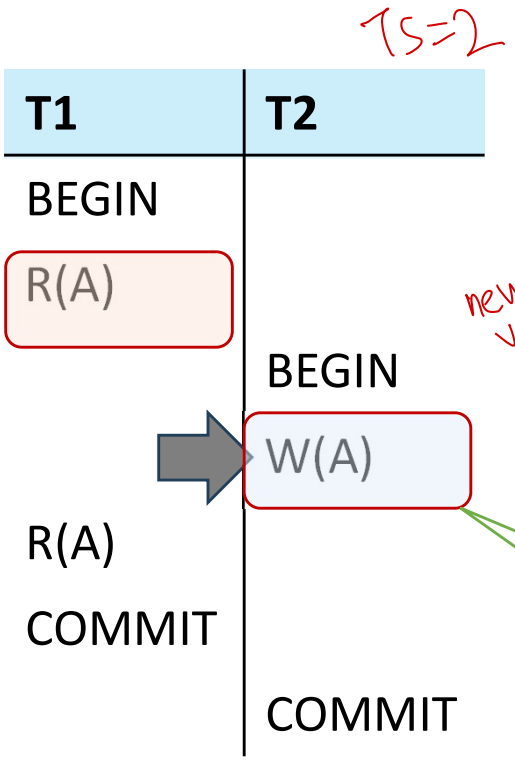
MVCC – Example #1

T1	T2
BEGIN	
R(A)	
	BEGIN
	W(A)
R(A)	
COMMIT	
	COMMIT

New version이 생기는 시간

Obj.	Version	Value	Begin	End
A	0	123	0	-

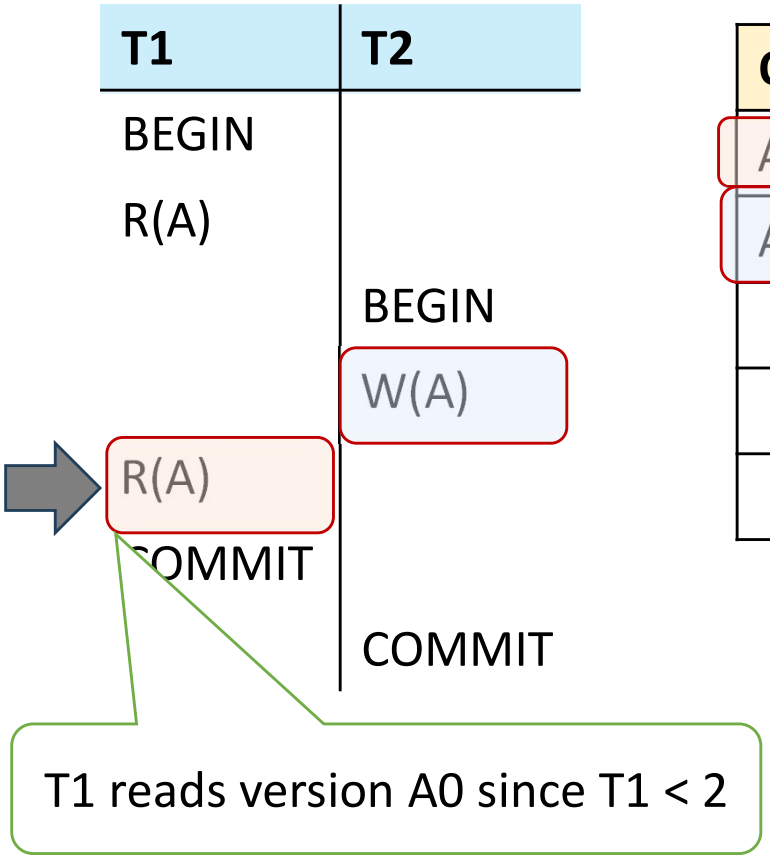
MVCC – Example #1



Obj.	Version	Value	Begin	End
A	0	123	0	2
A	1	456	2	-

T2 creates version A1 and sets A0 End-TS.

MVCC – Example #1



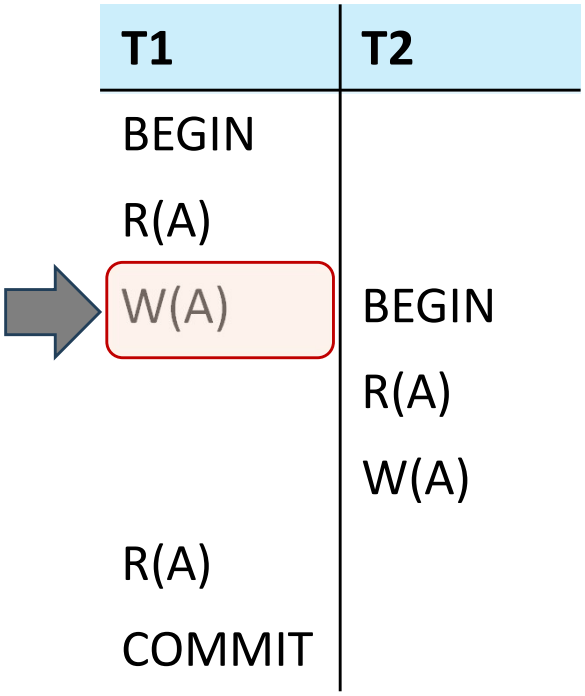
Obj.	Version	Value	Begin	End
A	0	123	0	2
A	1	456	2	-

MVCC – Example #2

T1	T2
BEGIN	
R(A)	
W(A)	BEGIN
	R(A)
	W(A)
R(A)	
COMMIT	

Obj.	Version	Value	Begin	End
A	0	123	0	-

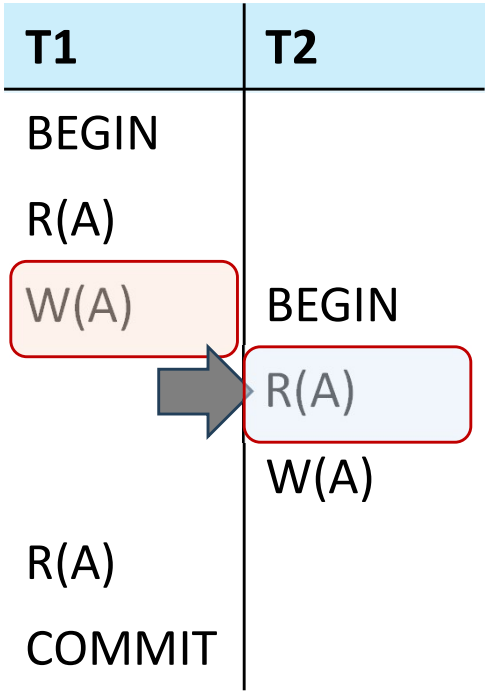
MVCC – Example #2



T1	T2
BEGIN	
R(A)	
W(A)	BEGIN
	R(A)
	W(A)
R(A)	
COMMIT	

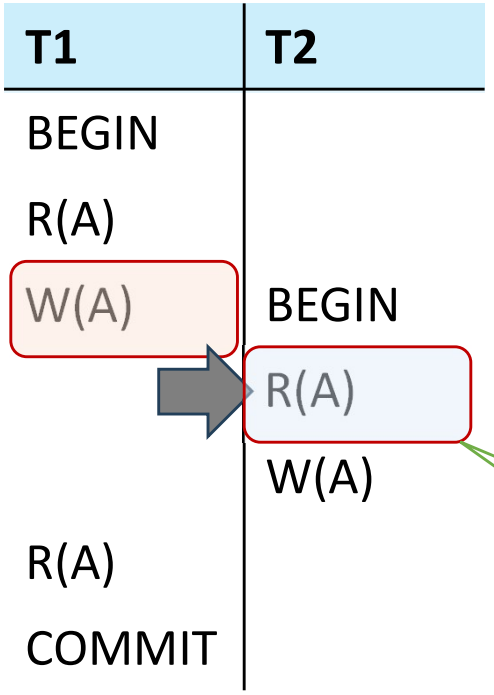
Obj.	Version	Value	Begin	End
A	0	123	0	1
A	1	456	1	-

MVCC – Example #2



Obj.	Version	Value	Begin	End
A	0	123	0	1
A	1	456	1	-

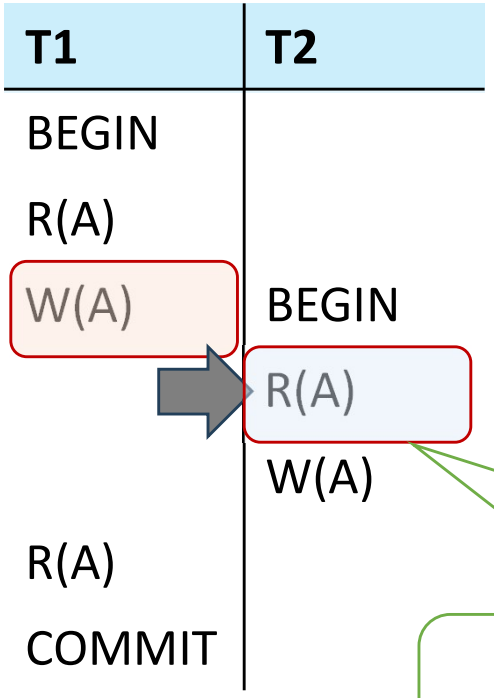
MVCC – Example #2



Obj.	Version	Value	Begin	End
A	0	123	0	1
A	1	456	1	-

Which version does T2 read?

MVCC – Example #2

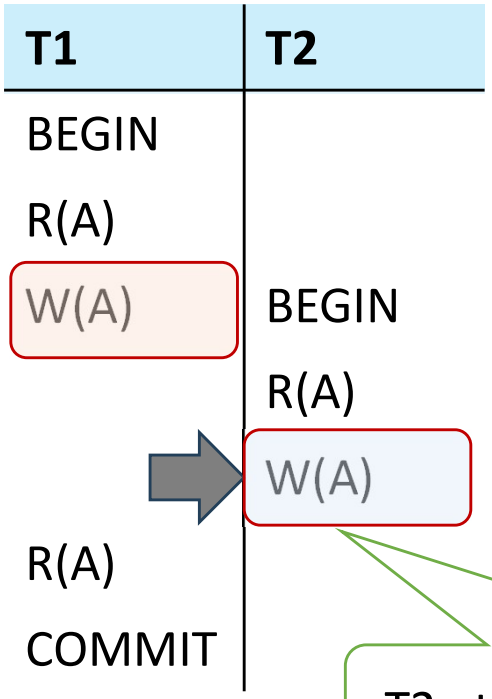


Obj.	Version	Value	Begin	End
A	0	123	0	1
A	1	456	1	-

T2 reads ver. 0 in Committed Read or stronger consistency modes because T1 has not committed yet.

⇒ T2는 dirty data에 접근하면 안됨
⇒ old에 접근 (version 0)

MVCC – Example #2



Obj.	Version	Value	Begin	End
A	0	123	0	1
A	1	456	1	-
A	2	789	2	-

→ Write-Write conflict

T_1이 아직 실행 중이라 T_2 write불가능

T2 attempts to create a new version

Multiversion Schemes

- Several variants to resolve conflicts:
  - **Multiversion Timestamp Ordering (MVTO)**
  - **Multiversion Two-Phase Locking (MV2PL)**
  - **Snapshot isolation** → first-committer-wins-rule 이용

Snapshot Isolation

Snapshot Isolation

- Motivation
  - OLAP (decision support query) transactions that read large amounts of data have concurrency conflicts with OLTP transactions that update a few rows
  - Poor performance
- Snapshot Isolation: Give snapshot of database to every transaction
  - follows Optimistic Concurrency Control
  - **Takes snapshot of committed data at start**
  - Always reads/modifies data in its own snapshot
  - Updates of concurrent transactions are not visible
  - **First-committer-wins rule**:
    - Commits only if no other concurrent transaction has already written data that the transaction intends to write.

Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0

T1 snapshot

Y
Y_0=0

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0

T1 snapshot

Y
Y_1=1

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation ∈ MVCC

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
	Y_1=1	

new version

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
	Y_1=1	

T2 snapshot

X
X_0=0

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
	Y_1=1	

T2 snapshot

X	Y
X_0=0	Y_1=1

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation

Database

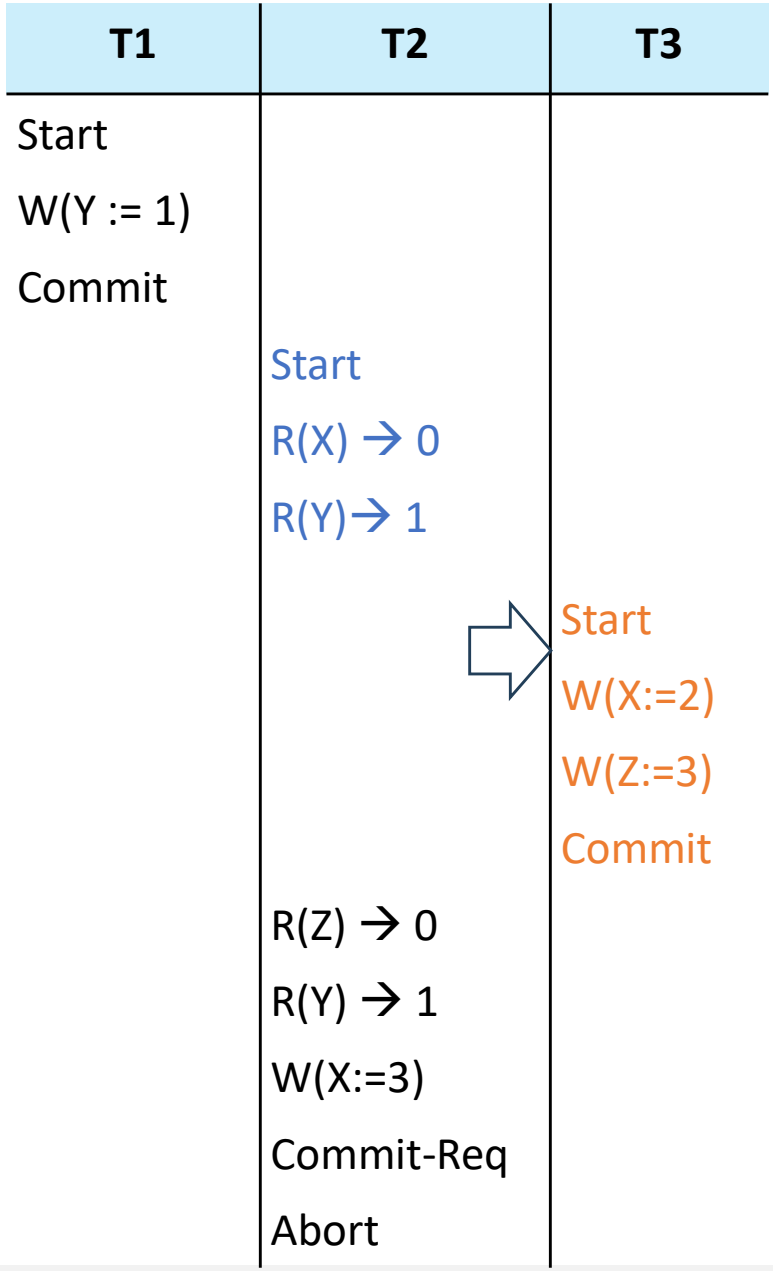
X	Y	Z
X_0=0	Y_0=0	Z_0=0
	Y_1=1	

T2 snapshot

X	Y
X_0=0	Y_1=1

T3 snapshot

X
X_0=0



Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
	Y_1=1	

T2 snapshot

X	Y
X_0=0	Y_1=1

T3 snapshot

X	Z
X_3=2	Z_0=0

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
	Y_1=1	

Flush

T2 snapshot

X	Y
X_0=0	Y_1=1

T3 snapshot

X	Z
X_3=2	**Z_3=3**

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

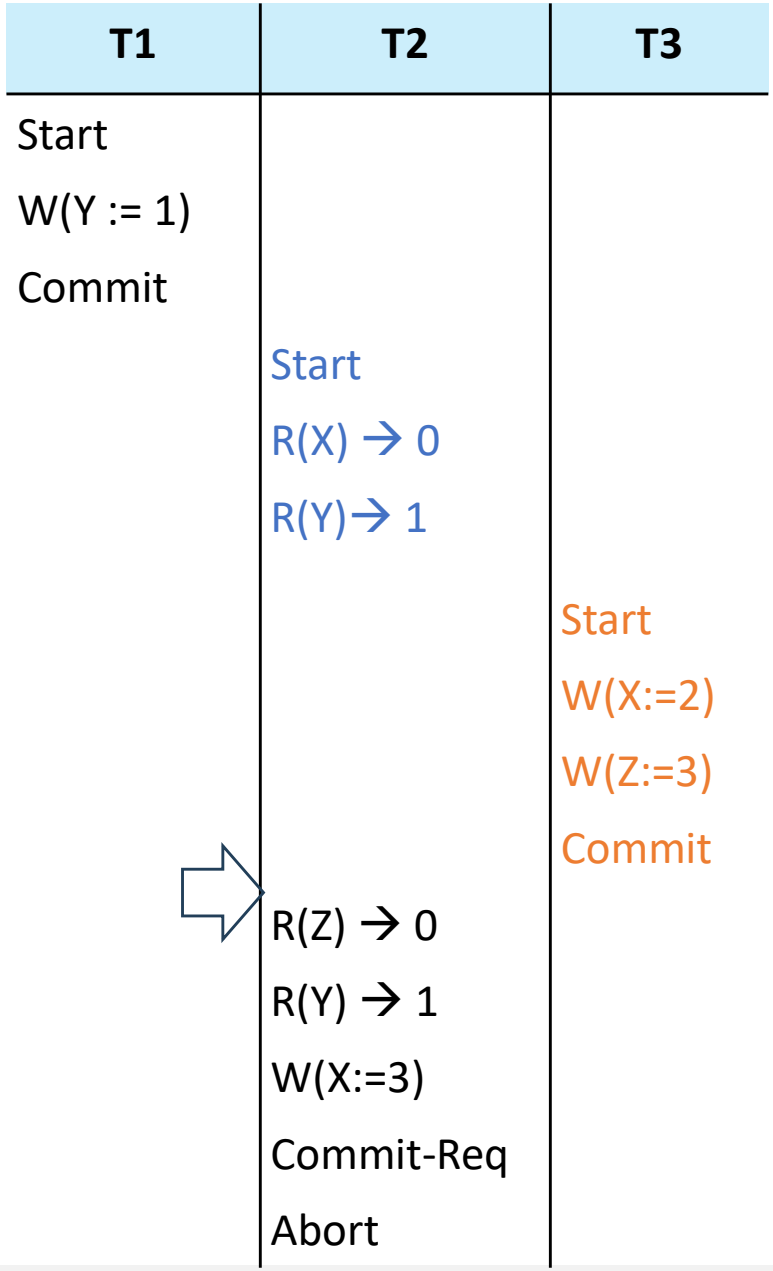
Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
X_3=2	Y_1=1	**Z_3=3**

T2 snapshot

X	Y	Z
X_0=0	Y_1=1	Z_0=0



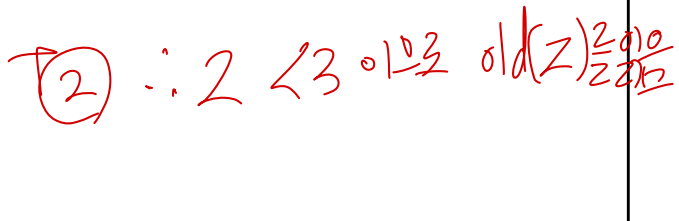
Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
X_3=2	Y_1=1	**Z_3=3**

T2 snapshot

X	Y	Z
X_0=0	Y_1=1	Z_0=0



T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
X_3=2	Y_1=1	**Z_3=3**

T2 snapshot

X	Y	Z
X_2=3	Y_1=1	Z_0=0

T1	T2	T3
Start		
W(Y := 1)		
Commit		
	Start	
	R(X) → 0	
	R(Y)→ 1	
		Start
		W(X:=2)
		W(Z:=3)
		Commit
	R(Z) → 0	
	R(Y) → 1	
	W(X:=3)	
	Commit-Req	
	Abort	

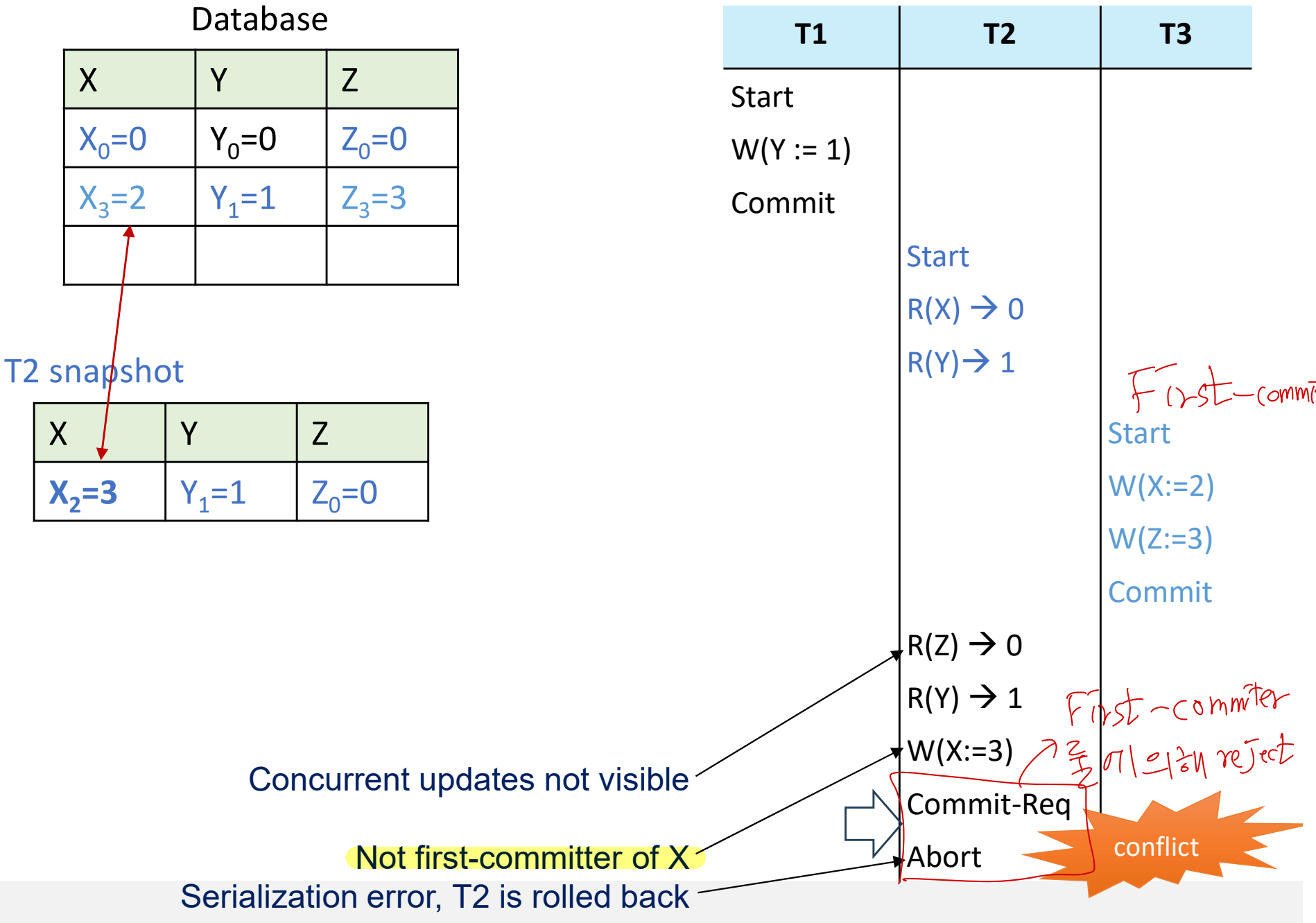
Snapshot Isolation

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
X_3=2	Y_1=1	Z_3=3

T2 snapshot

X	Y	Z
X_2=3	Y_1=1	Z_0=0



Snapshot Isolation – another example (1)

Database

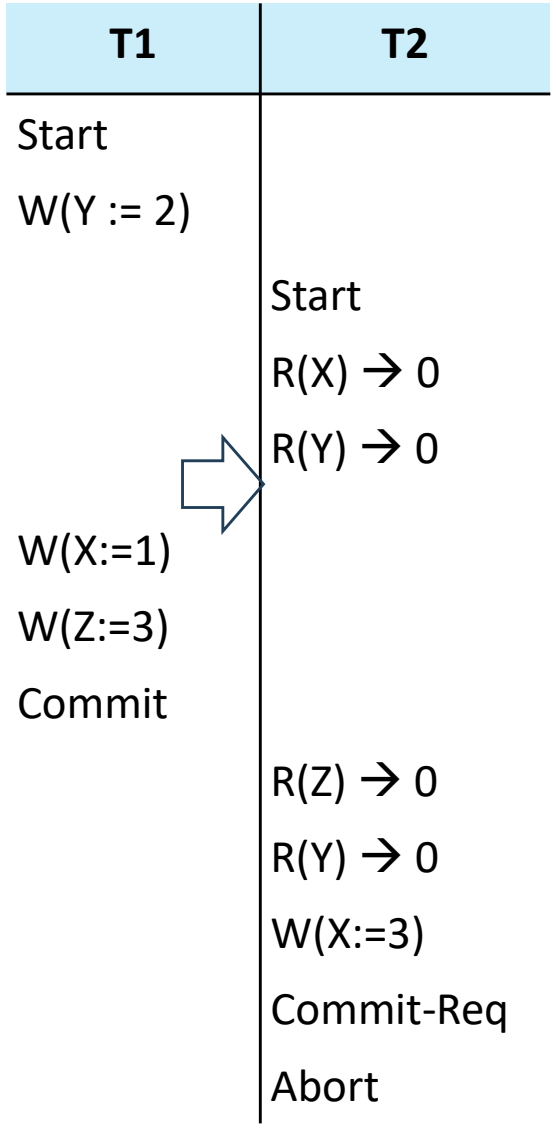
X	Y	Z
X_0=0	Y_0=0	Z_0=0

T1 snapshot

Y
Y_1=2

T2 snapshot

X	Y
X_0=0	Y_0=0



Snapshot Isolation – another example (2)

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0

T1 snapshot

X	Y	Z
X_1=1	Y_1=2	Z_1=3

T2 snapshot

X	Y
X_0=0	Y_0=0

T1	T2
Start	
W(Y := 2)	
	Start
	R(X) → 0
	R(Y) → 0
W(X:=1)	
W(Z:=3)	
Commit	
	R(Z) → 0
	R(Y) → 0
	W(X:=3)
	Commit-Req
	Abort

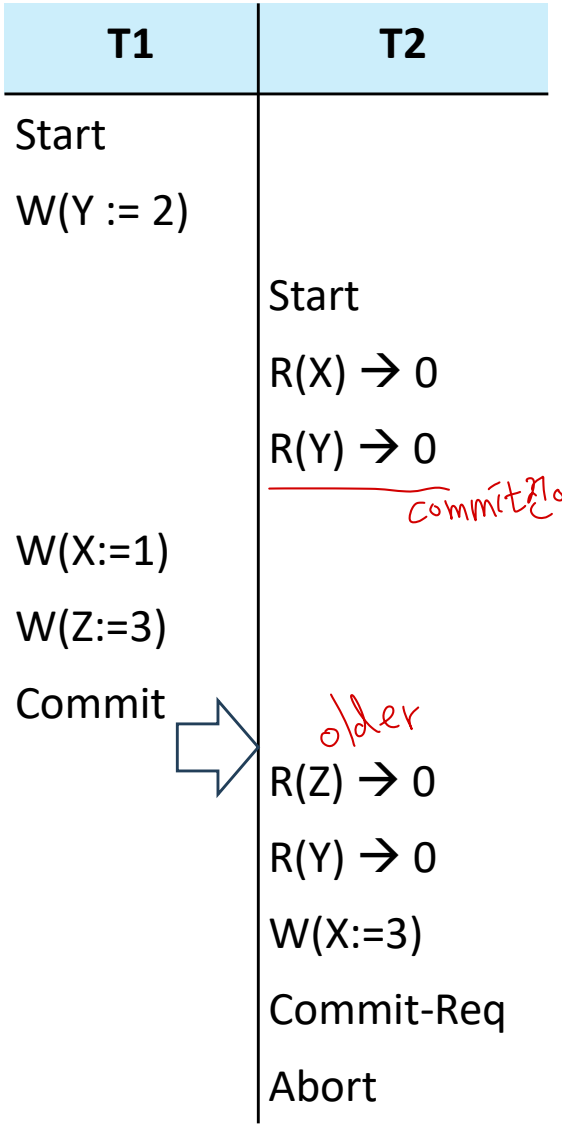
Snapshot Isolation – another example (3)

Database

X	Y	Z
X_0=0	Y_0=0	Z_0=0
X_1=1	Y_1=2	Z_1=3

T2 snapshot

X	Y	Z
X_0=0	Y_0=0	Z_0=0



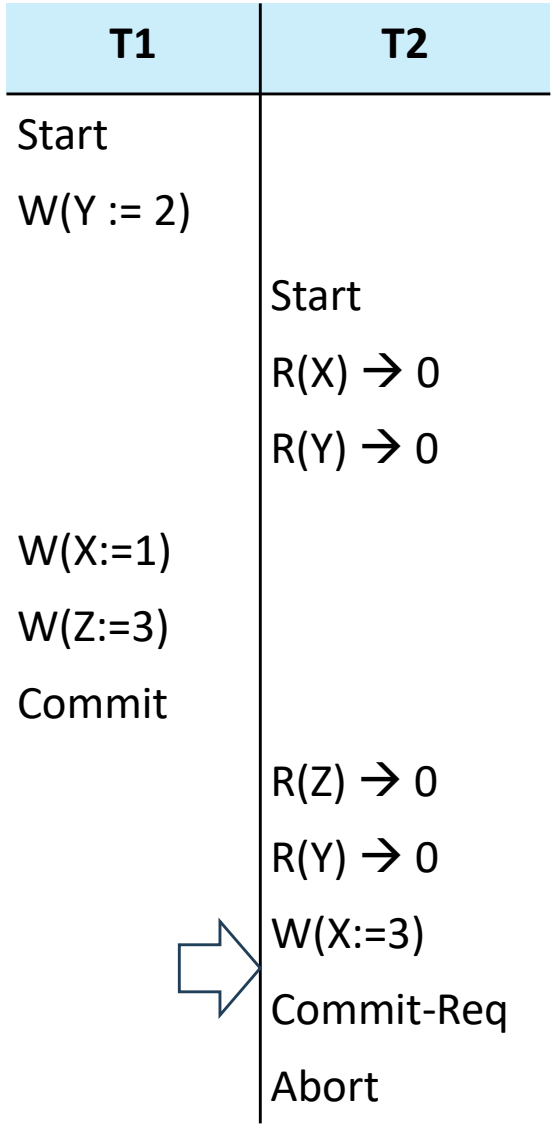
Snapshot Isolation – another example (3)

Database

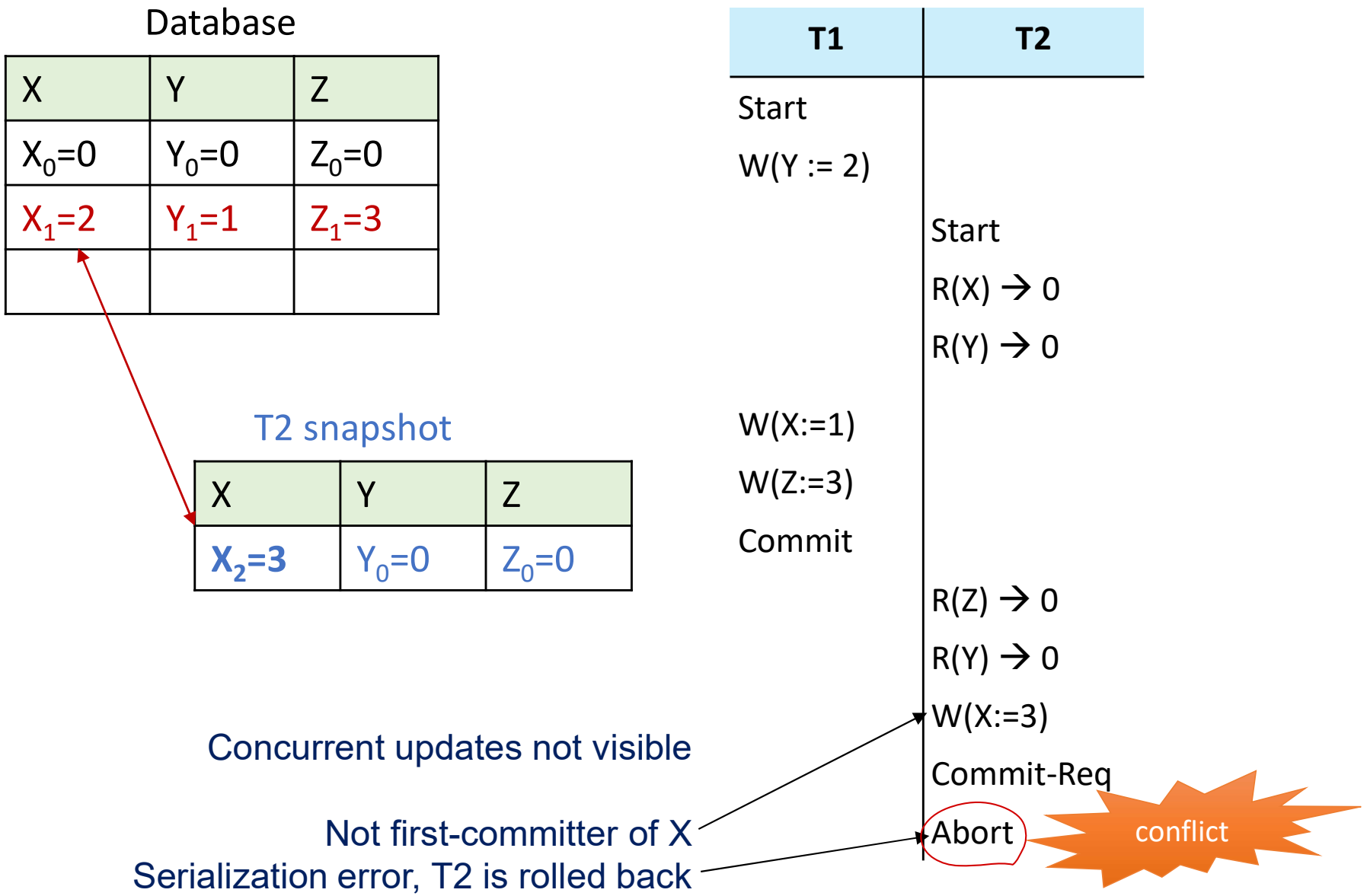
X	Y	Z
X_0=0	Y_0=0	Z_0=0
X_1=1	Y_1=2	Z_1=3

T2 snapshot

X	Y	Z
X_2=3	Y_0=0	Z_0=0



Snapshot Isolation – another example (4)



Benefits of Snapshot Isolation

- Reads are never blocked (Good for OLAP queries) → 매우 빠르게 실행
- Avoids several anomalies
  - No dirty read
    - no read of uncommitted data
  - No lost update ← ignore
    - Lost update is the update overwritten by another transaction that did not see the update
  - No non-repeatable read
    - I.e., if read is executed again, it will see the same value

↳ 같은 쿼리, 같은 데이터

Snapshot Isolation Anomaly

- Snapshot Isolation does not always give serializable executions
  - Serializable: among two concurrent txns, one sees the effects of the other
  - In SI: neither sees the effects of the other
- Result: Integrity constraints can be violated

Example: Skew Write

- Initially A = 3 and B = 17
  - Serial execution: A = ??, B = ??
  - if both transactions start at the same time, with snapshot isolation: A = ?? , B = ??

AB(3,17)를 serial로 생각해봐.
$T_1 \rightarrow T_2$ 생각해보면
(17,17) → (17,17)
$T_2 \rightarrow T_1$ 생각해보면
(3,3) → (3,3)
둘을 하나라고봄

storage

A	B
A_0=3	B_0=17

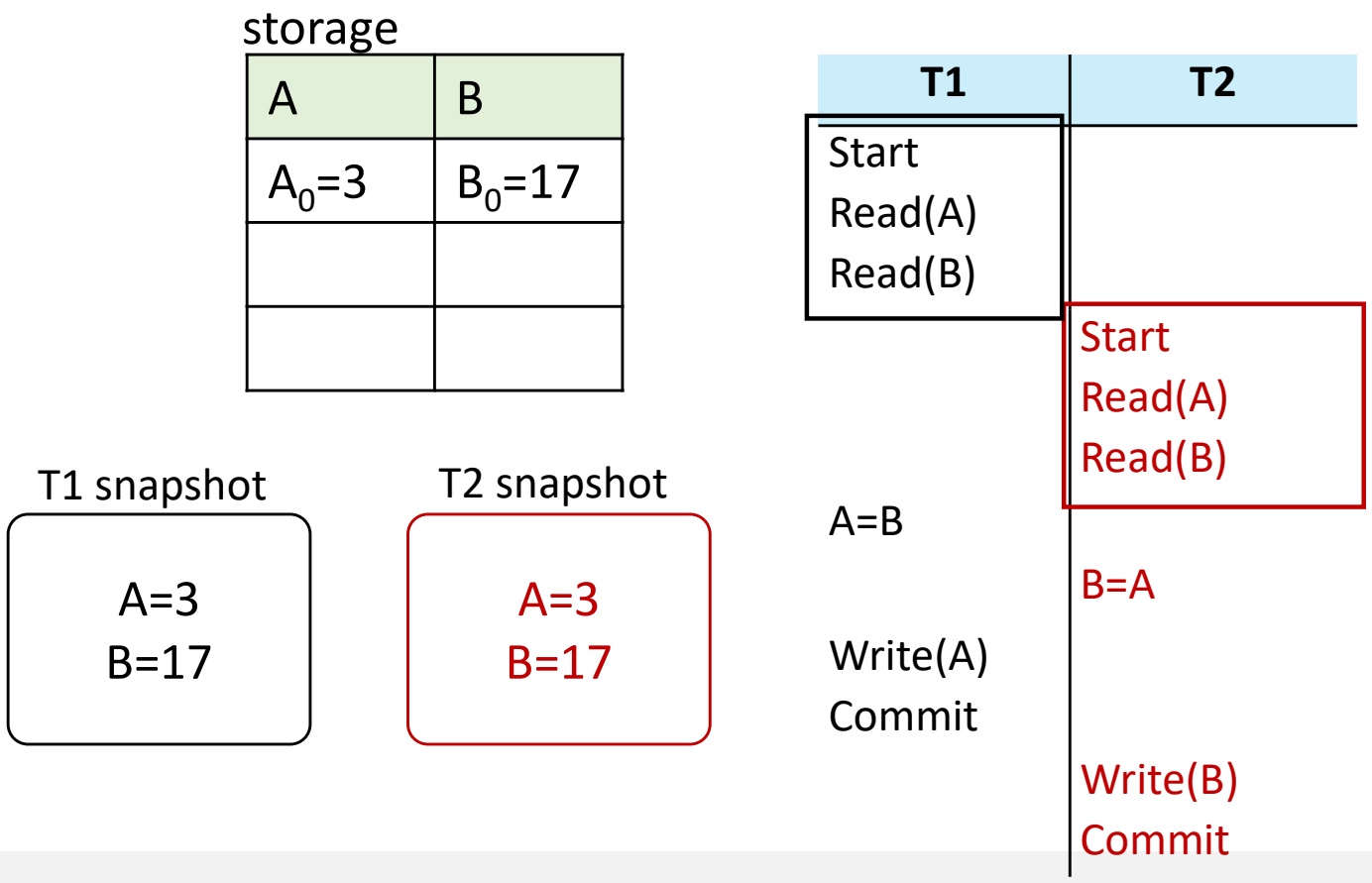
T1 snapshot

A=3
B=17

T1	T2
Start	
Read(A)	
Read(B)	
	Start
	Read(A)
	Read(B)
A=B	
	B=A
Write(A)	
Commit	
	Write(B)
	Commit

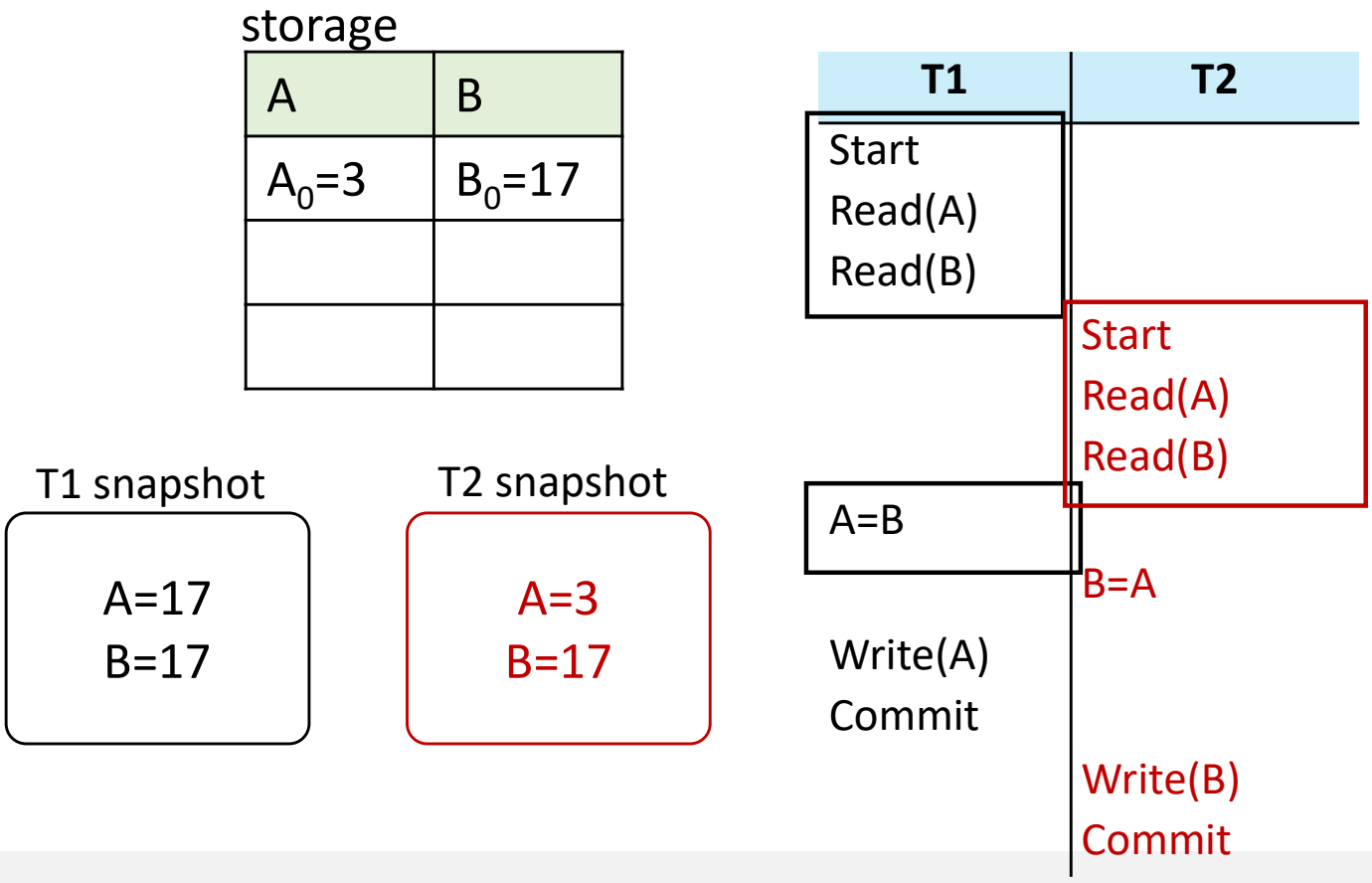
Example: Skew Write

- Initially A = 3 and B = 17
  - Serial execution: A = ??, B = ??
  - if both transactions start at the same time, with snapshot isolation: A = ?? , B = ??



Example: Skew Write

- Initially A = 3 and B = 17
  - Serial execution: A = ??, B = ??
  - if both transactions start at the same time, with snapshot isolation: A = ?? , B = ??



Example: Skew Write

- Initially A = 3 and B = 17
  - Serial execution: A = ??, B = ??
  - if both transactions start at the same time, with snapshot isolation: A = ?? , B = ??

storage

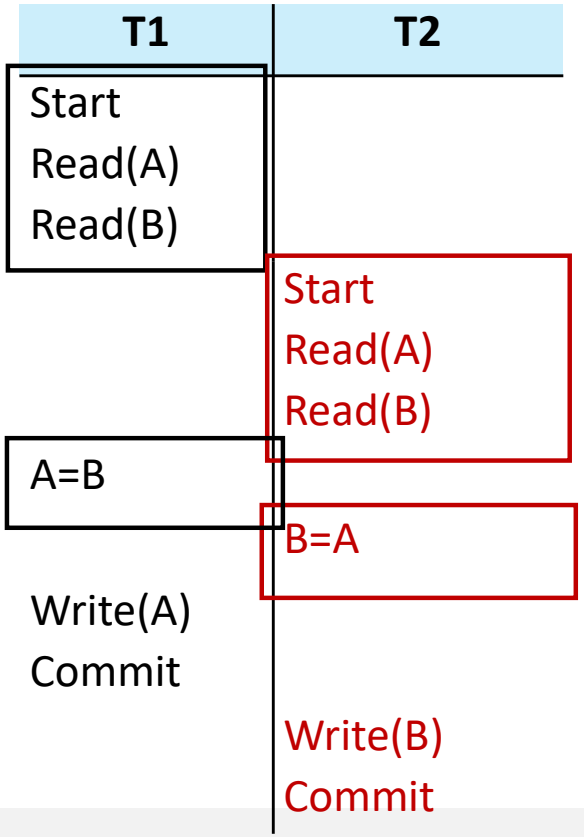
A	B
A_0=3	B_0=17

T1 snapshot

A=17
B=17

T2 snapshot

A=3
B=3



Example: Skew Write

- Initially A = 3 and B = 17
  - Serial execution: A = ??, B = ??
  - if both transactions start at the same time, with snapshot isolation: A = ?? , B = ??

storage

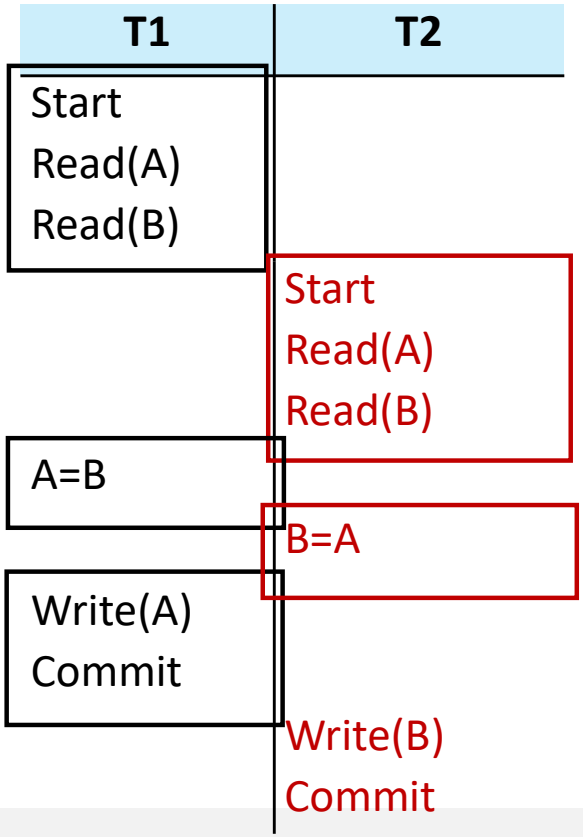
A	B
A_0=3	B_0=17
A_1=17	

T1 snapshot

A=17
B=17

T2 snapshot

A=3
B=3



Example: Skew Write

- Initially A = 3 and B = 17
  - Serial execution: A = ??, B = ??
  - if both transactions start at the same time, with snapshot isolation: A = ?? , B = ??

storage

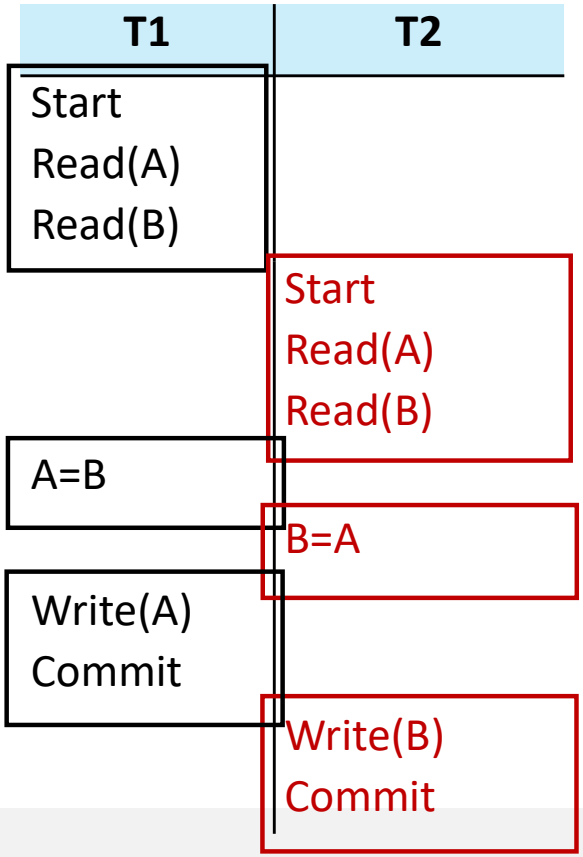
A	B
A_0=3	B_0=17
A_1=17	B_2=3

T1 snapshot

A=17
B=17

T2 snapshot

A=3
B=3



⇒ AB = (17, 3)

serial의 결과값 아님!

∴ Snapshot isolation은 Not serializable!

serializable을 보장하지 못함 = skew write

Skew Write

- **Skew Write** is not very common in practice
- Skew also occurs with inserts, e.g.,
  - Find max order number among all orders
  - Create a new order with order number = previous max + 1
  - Two transaction can both create order with same number

snapshot

T_i	T_j
read(A)	
read(B)	
	read(A)
	read(B)
A=B	
	B=A
write(A)	
	write(B)

A	B
A_0=3	B_0=17
A_1=17	B_2=3

serial

T1	T2
Read(A)	
Read(B)	
A=B	
Write(A)	
	Read(A)
	Read(B)
	B=A
	Write(B)

A	B
A_0=3	B_0=17
A_1=17	B_2=17

serial

T1	T2
	Read(A)
	Read(B)
	B=A
	Write(B)
Read(A)	
Read(B)	
A=B	
Write(A)	

A	B
A_0=3	B_0=17
A_1=3	B_2=3

Working Around Snapshot Isolation Anomalies

skew write를 for update를 통해 피할 수있다

- Can work around SI anomalies for specific queries by using **select .. for update** (supported in Oracle)
  - Example
    - **select max**(orderno) **from** orders **for update**
    - read value into local variable maxorder
    - insert into orders (maxorder+1, …)
  - **select for update (SFU) clause** treats all data read by the query as if it were also updated, preventing concurrent updates
  - Can be added to queries to ensure serializability in many applications
    - Does not handle phantom phenomenon/predicate reads though

Multi-Version Timestamp Ordering (MVTO)

Multiversion Timestamp Ordering (MVTO)

- Each data item Q has a sequence of versions $<Q_1, Q_2, \ldots, Q_m>$.

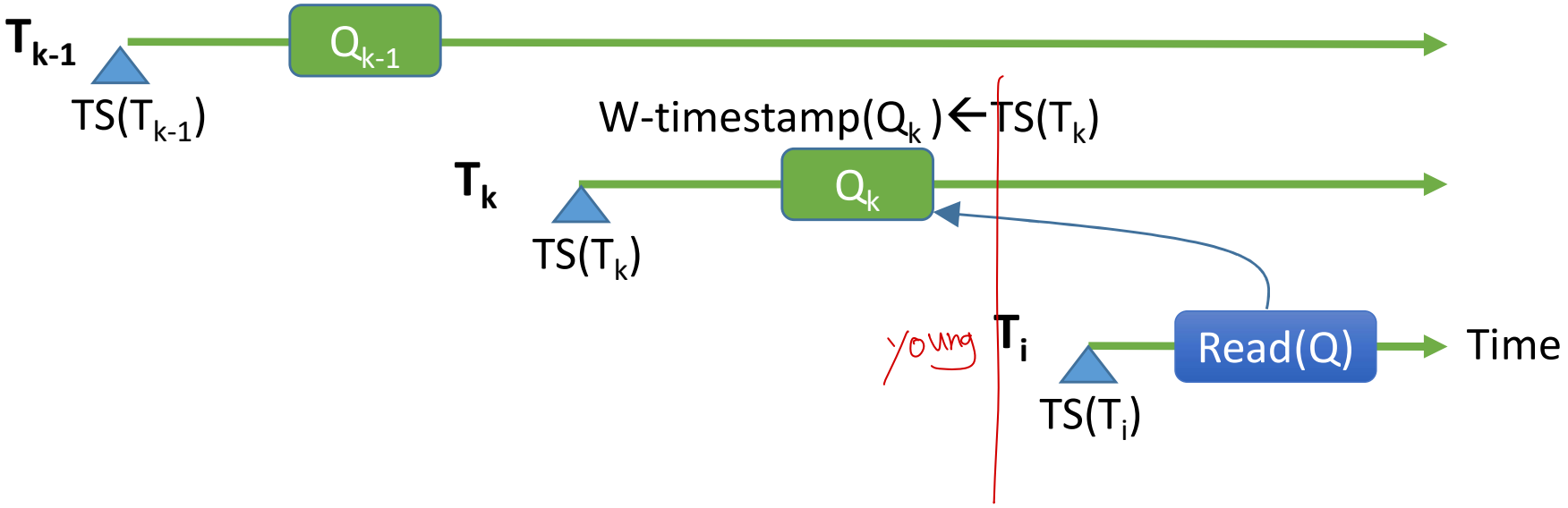
TSO 와 비슷
차이점은 version 여러개를 저장

- Each version Q_k contains three data fields:
  - **Content**
    - the value of version Q_k.
  - **W-timestamp**(Q_k)
    - timestamp of the transaction that created version Q_k
  - **R-timestamp**(Q_k)
    - largest timestamp of a transaction that successfully read version Q_k

Multiversion Timestamp Ordering (Cont)

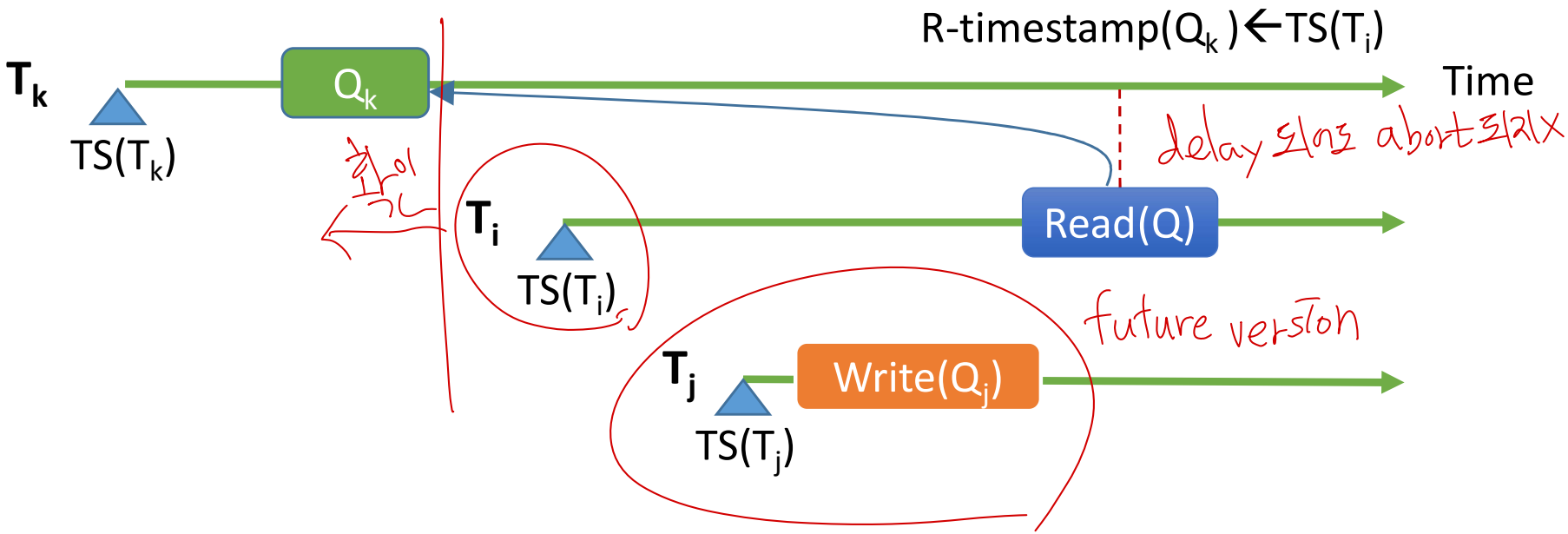
- Let Q_k denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS(T_i).
  1. If transaction T_i issues a **read**(Q), then
     - return the content of version Q_k
     - If R-timestamp(Q_k) < TS(T_i),
       set R-timestamp(Q_k) ← TS(T_i),

W-TS ≤ TS(T_i) 인 애를 찾기 가장 큰 값으로!



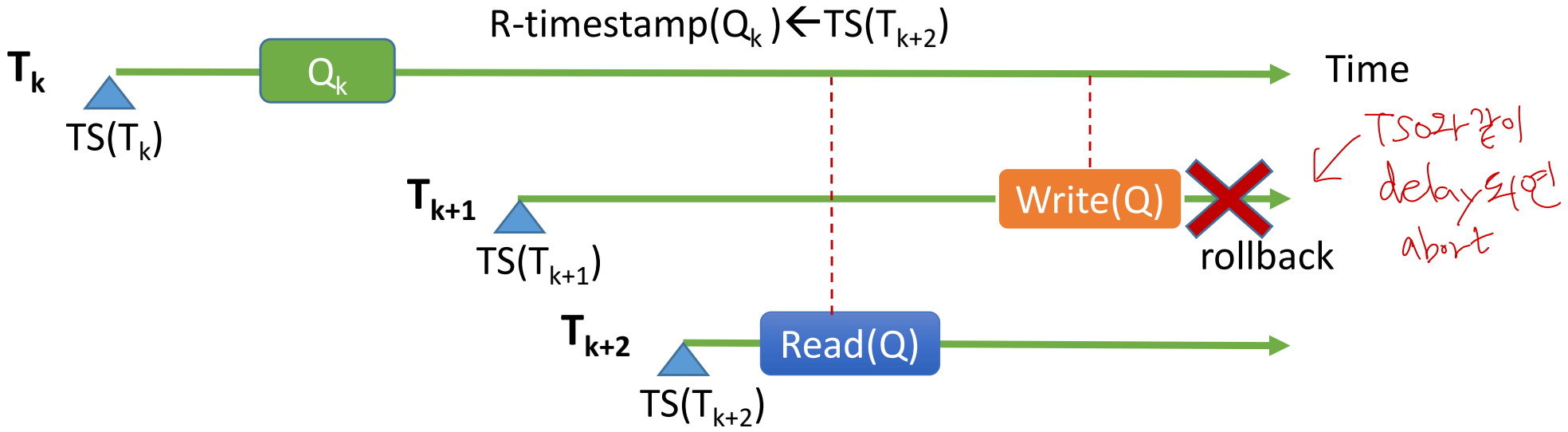
Multiversion Timestamp Ordering (Cont)

- Let Q_k denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS(T_i).
  1. If transaction T_i issues a **read**(Q), then
     - return the content of version Q_k
     - If R-timestamp(Q_k) < TS(T_i),
       set R-timestamp(Q_k) ← TS(T_i),



Multiversion Timestamp Ordering (Cont)

2. If transaction T_i issues a **write**(Q)
   1. if TS(T_i) < R-timestamp(Q_k), roll back transaction T_i.
   2. if TS(T_i) = W-timestamp(Q_k), overwrite Q_k.
   3. Otherwise, create a new version Q_i
      - Set W-timestamp(Q_i) and R-timestamp(Q_i) to TS(T_i).



Multiversion Timestamp Ordering (Cont)

- Observations
  - **Reads always succeed** → historical data에 접근하기에
  - **Writes may be rejected**

- MVTO guarantees serializability

Multi-Version Two-Phase Locking (MV2PL)

강동식 2PL과 비슷 (갖비싼 lock방식채용)

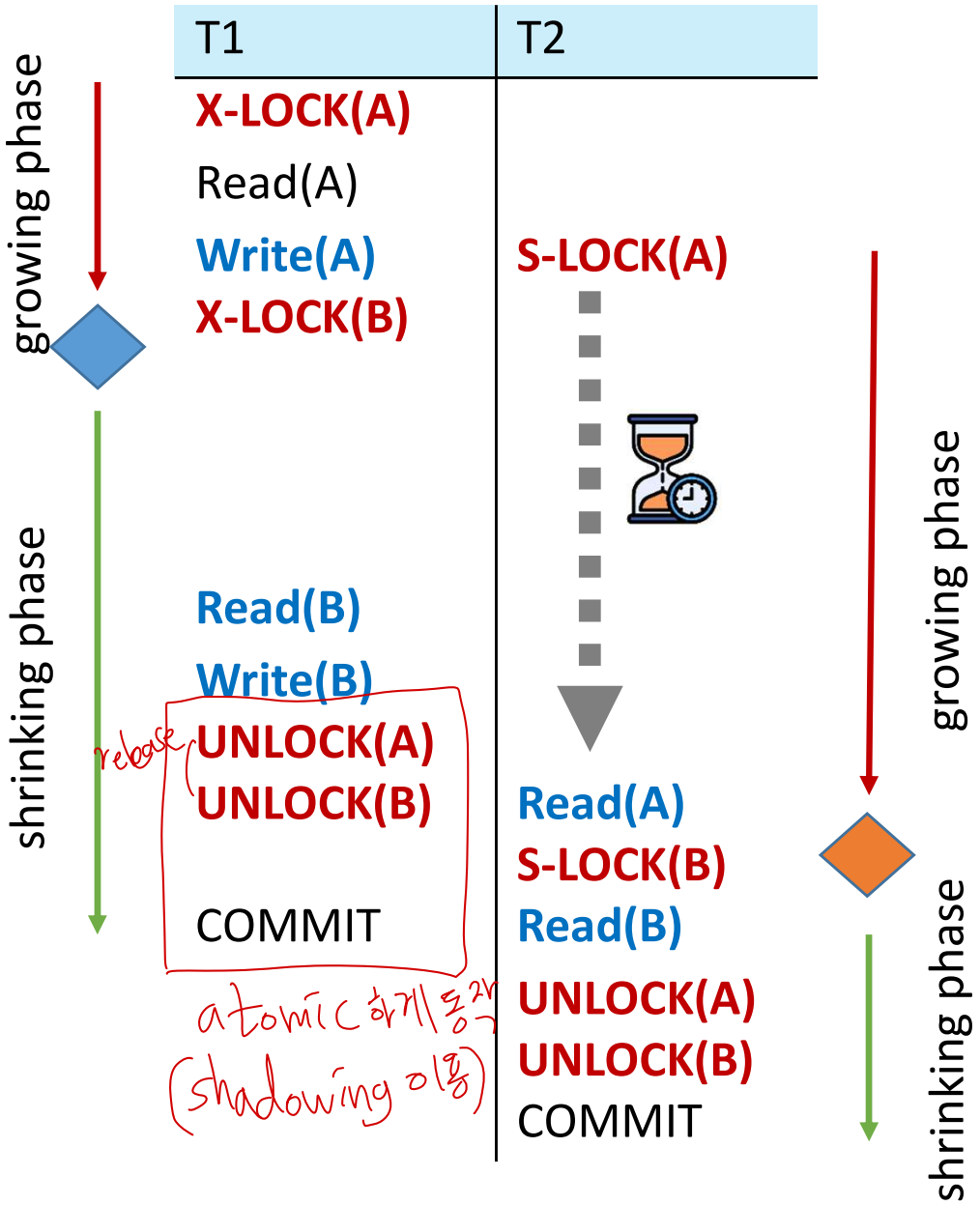
Multiversion Two-Phase Locking (MV2PL)

- Main Goal: Differentiate read-only transactions from write transactions

동시성 concurrency 레벨↓

- **Update transactions** use strict two-phase locking. → commit할 준비가 되면, 모든 lock 해제
  - Read(Q) returns the latest version of the item
  - The first **write** of Q by T_i creates a new version Q_i
    - W-timestamp(Q_i) set to ∞ initially → Read가 W-TS를 읽을 텐데 아직 commit 되지 않았음을 표시해두기 위해
  - When T_i commits:
    - Set TS(T_i) = **ts-counter** + 1
    - Set W-timestamp(Q_i) = TS(T_i)
    - **ts-counter** = **ts-counter +** 1

Strict Two-Phase Locking (2PL) - Review



Implementation of MV2PL

T10	T11
S-Lock(A)	
Read(A)	
X-Lock(B)	
Write(B)	

Database

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	1	∞
B_1	0	0	1	∞

- Txns use the tuple's Read-CNT field as SHARED lock.
- If TXN-ID is zero, then the transaction acquires the SHARED lock by incrementing the Read-CNT field.

Implementation of MV2PL

T10	T11
S-Lock(A)	
Read(A)	
X-Lock(B)	
Write(B)	

Database

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	1	1	∞
B_1	0	0	1	∞

- Txns use the tuple's Read-CNT field as SHARED lock.
- If TXN-ID is zero, then the transaction acquires the SHARED lock by incrementing the Read-CNT field.

Implementation of MV2PL

T10	T11
S-Lock(A)	
Read(A)	
X-Lock(B)	
Write(B)	

Database

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	1	1	∞
B_1	**10**	**1**	1	∞

- Use TXN-ID and Read-CNT together as EXCLUSIVE lock.
- If both TXN-ID and Read-CNT are zero, then transaction acquires the EXCLUSIVE lock by setting both of them.

Implementation of MV2PL

T10	T11
S-Lock(A)	
Read(A)	
X-Lock(B)	
Write(B)	

Database

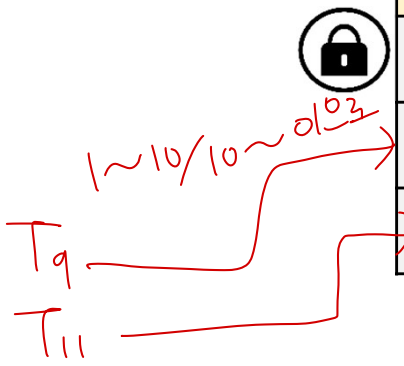
	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	1	1	∞
B_1	10	1	1	∞
B_2	**10**	**0**	**∞**	**∞**

new version

- Use TXN-ID and Read-CNT together as EXCLUSIVE lock.
- If both TXN-ID and Read-CNT are zero, then transaction acquires the EXCLUSIVE lock by setting both of them.

Implementation of MV2PL

T10	T11
S-Lock(A)	
Read(A)	
X-Lock(B)	
Write(B)	
Unlock(A)	
Unlock(B)	
Commit	

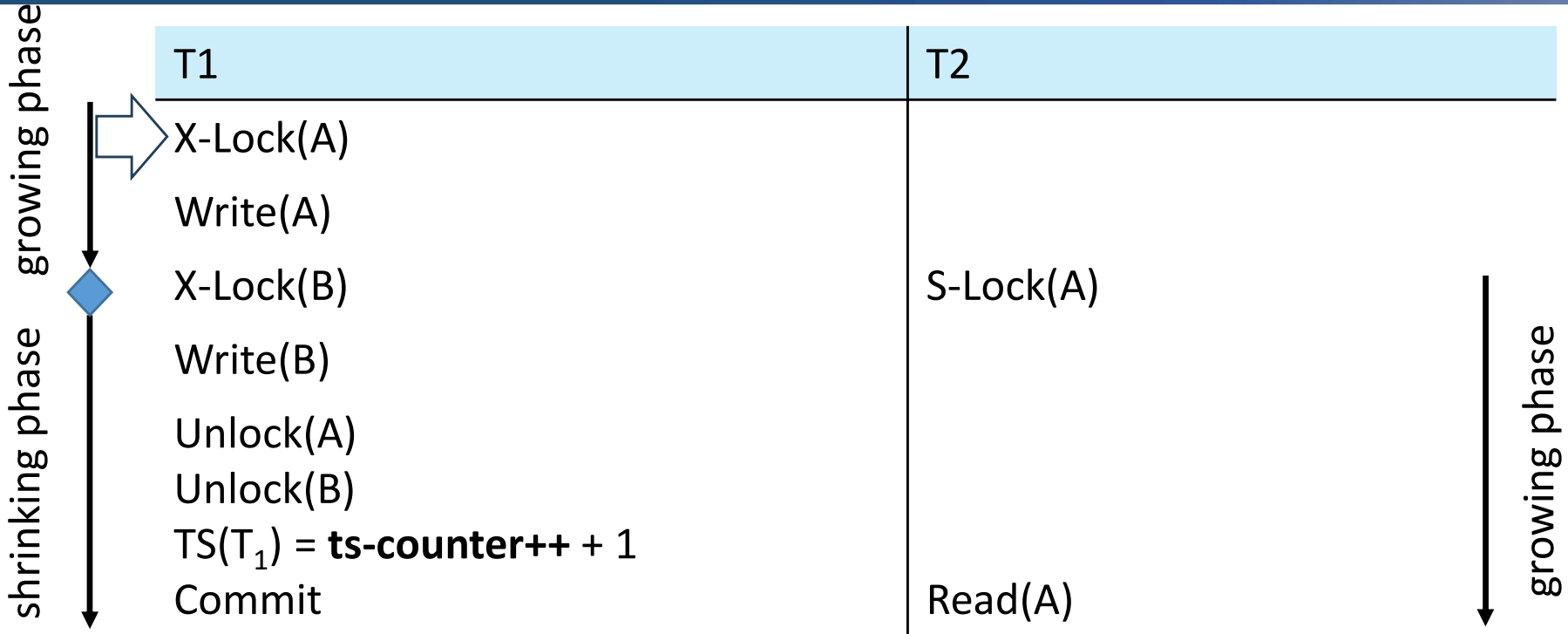


Database

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	1	∞
B_1	0	0	1	10
B_2	0	0	10	∞

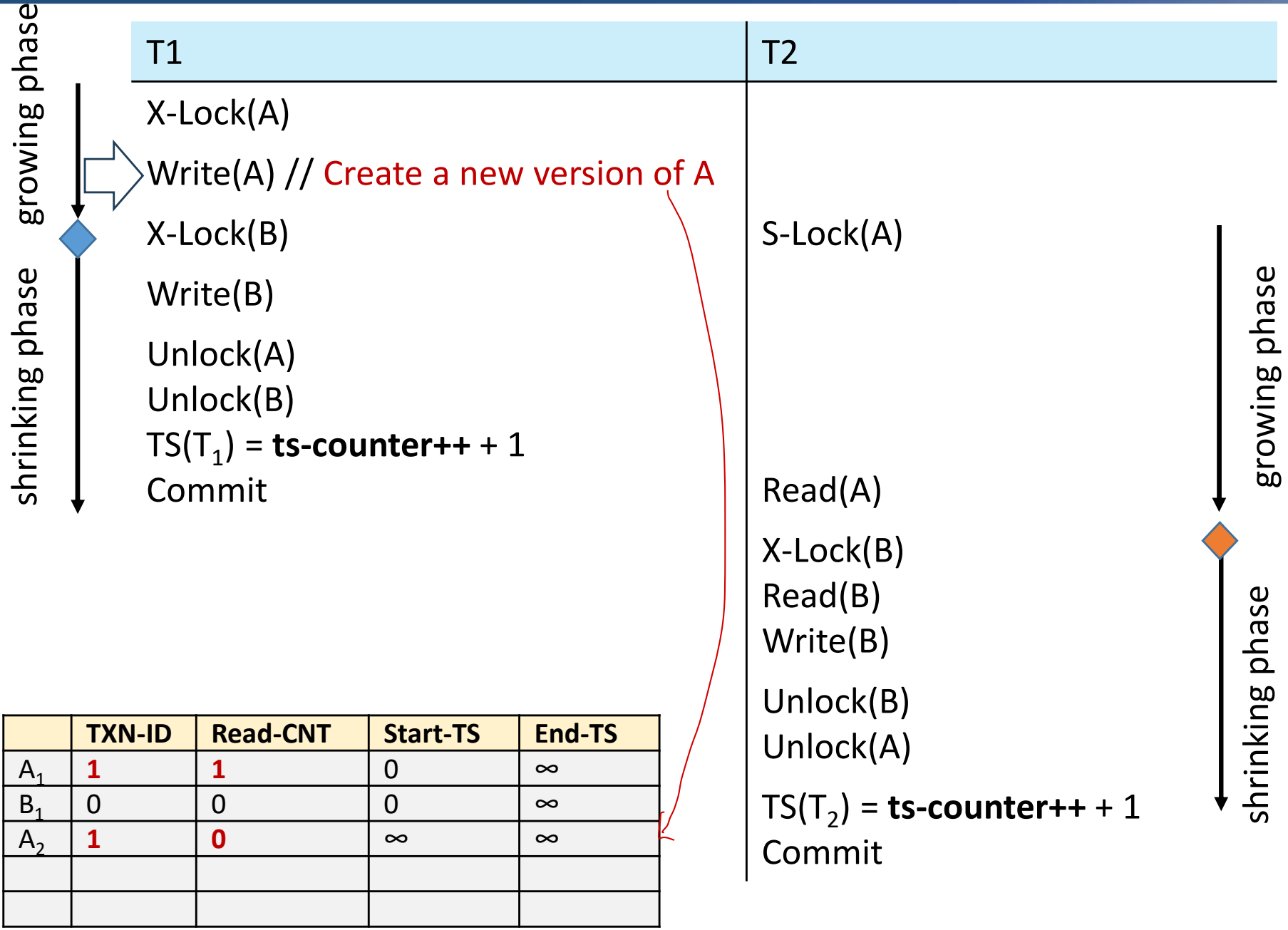
- Use TXN-ID and Read-CNT together as EXCLUSIVE lock.
- If both TXN-ID and Read-CNT are zero, then transaction acquires the EXCLUSIVE lock by setting both of them.

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example



	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	1	1	0	-
B_1	0	0	0	-

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example



	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	**1**	**1**	0	∞
B_1	0	0	0	∞
A_2	**1**	**0**	∞	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example

growing phase

shrinking phase

T1	T2
X-Lock(A)	
Write(A)	
X-Lock(B) // acquires X-lock	S-Lock(A) // tries to lock A_1
Write(B)	⇒ Block
Unlock(A)	
Unlock(B)	
$TS(T_1)$ = **ts-counter++** + 1	
Commit	Read(A)
	X-Lock(B)
	Read(B)
	Write(B)
	Unlock(B)
	Unlock(A)
	$TS(T_2)$ = **ts-counter++** + 1
	Commit

growing phase

shrinking phase

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	1	1	0	∞
B_1	**1**	**1**	0	∞
A_2	1	0	∞	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example

growing phase

shrinking phase

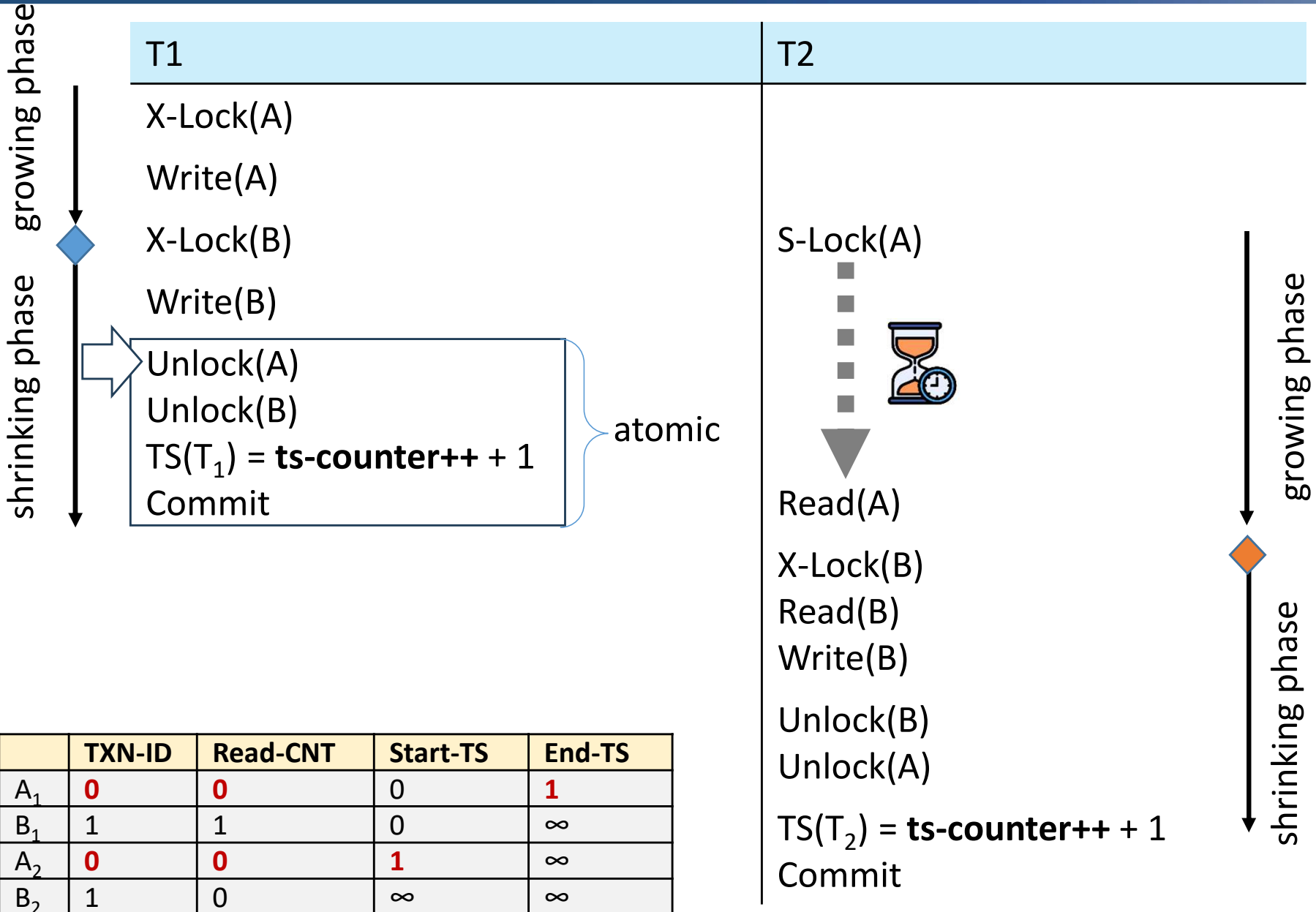
T1	T2
X-Lock(A)	
Write(A)	
X-Lock(B)	S-Lock(A)
Write(B) // Create a new version of B	
Unlock(A)	
Unlock(B)	
$TS(T_1)$ = **ts-counter++** + 1	
Commit	Read(A)
	X-Lock(B)
	Read(B)
	Write(B)
	Unlock(B)
	Unlock(A)
	$TS(T_2)$ = **ts-counter++** + 1
	Commit

growing phase

shrinking phase

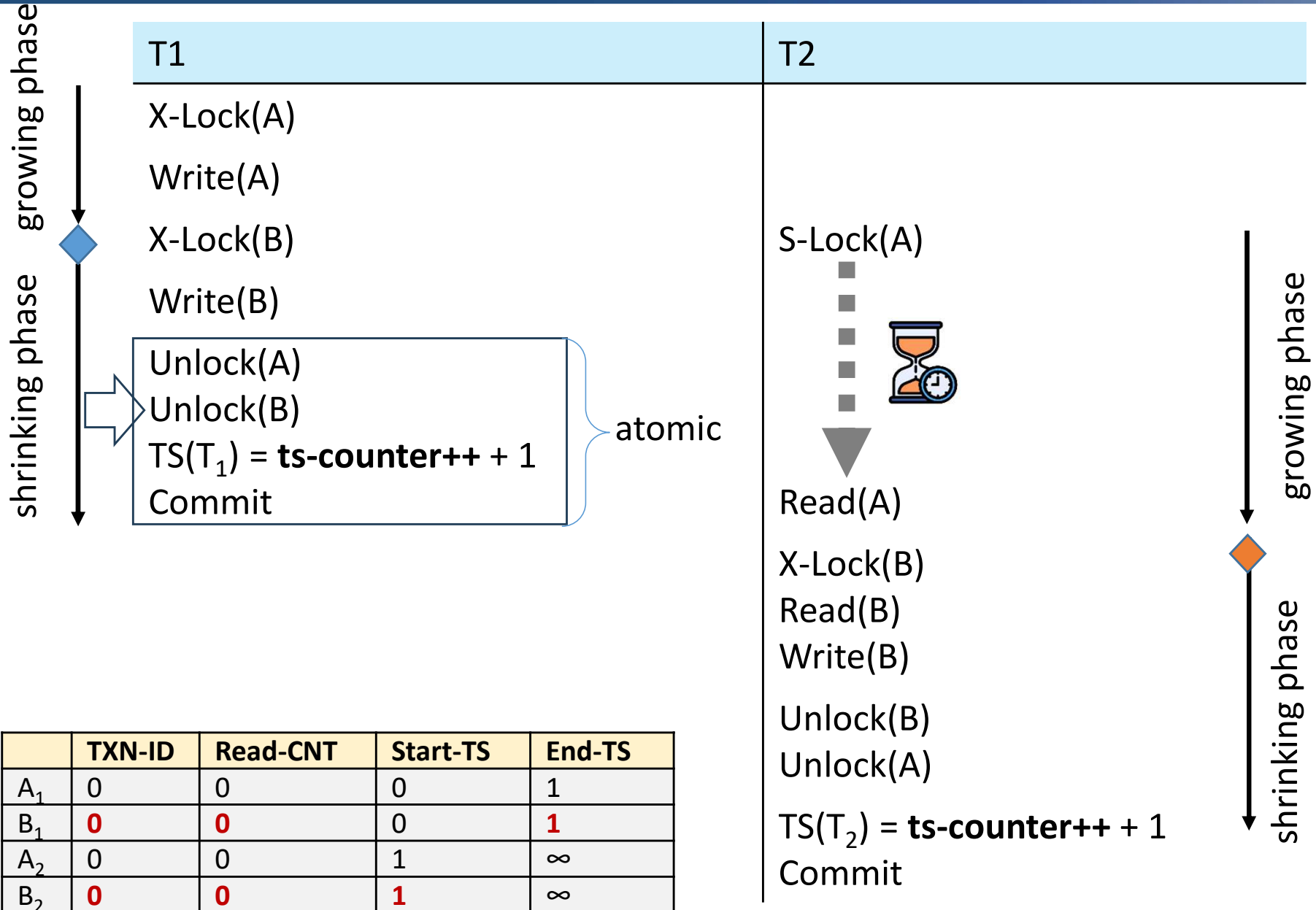
	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	1	1	0	∞
B_1	**1**	**1**	0	∞
A_2	1	0	∞	∞
B_2	**1**	**0**	∞	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example



	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	**0**	**0**	0	**1**
B_1	1	1	0	∞
A_2	**0**	**0**	**1**	∞
B_2	1	0	∞	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example



	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	0	1
B_1	**0**	**0**	0	**1**
A_2	0	0	1	∞
B_2	**0**	**0**	**1**	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example

growing phase

shrinking phase

T1	T2
X-Lock(A)	
Write(A)	
X-Lock(B)	S-Lock(A)
Write(B)	
Unlock(A)	
Unlock(B)	
$TS(T_1)$ = **ts-counter++** + 1	
Commit	Read(A) // reads A_2
	X-Lock(B)
	Read(B)
	Write(B)
	Unlock(B)
	Unlock(A)
	$TS(T_2)$ = **ts-counter++** + 1

growing phase

shrinking phase

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	0	1
B_1	0	0	0	1
A_2	0	**1**	1	∞
B_2	0	0	1	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example

growing phase

shrinking phase

T1	T2
X-Lock(A)	
Write(A)	
X-Lock(B)	S-Lock(A)
Write(B)	
Unlock(A)	
Unlock(B)	
$TS(T_1)$ = **ts-counter++** + 1	
Commit	
	Read(A)
	X-Lock(B)
	Read(B) // reads B_2
	Write(B)
	Unlock(B)
	Unlock(A)
	$TS(T_2)$ = **ts-counter++** + 1

growing phase

shrinking phase

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	0	1
B_1	0	0	0	1
A_2	0	1	1	∞
B_2	**2**	**1**	1	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example

growing phase

shrinking phase

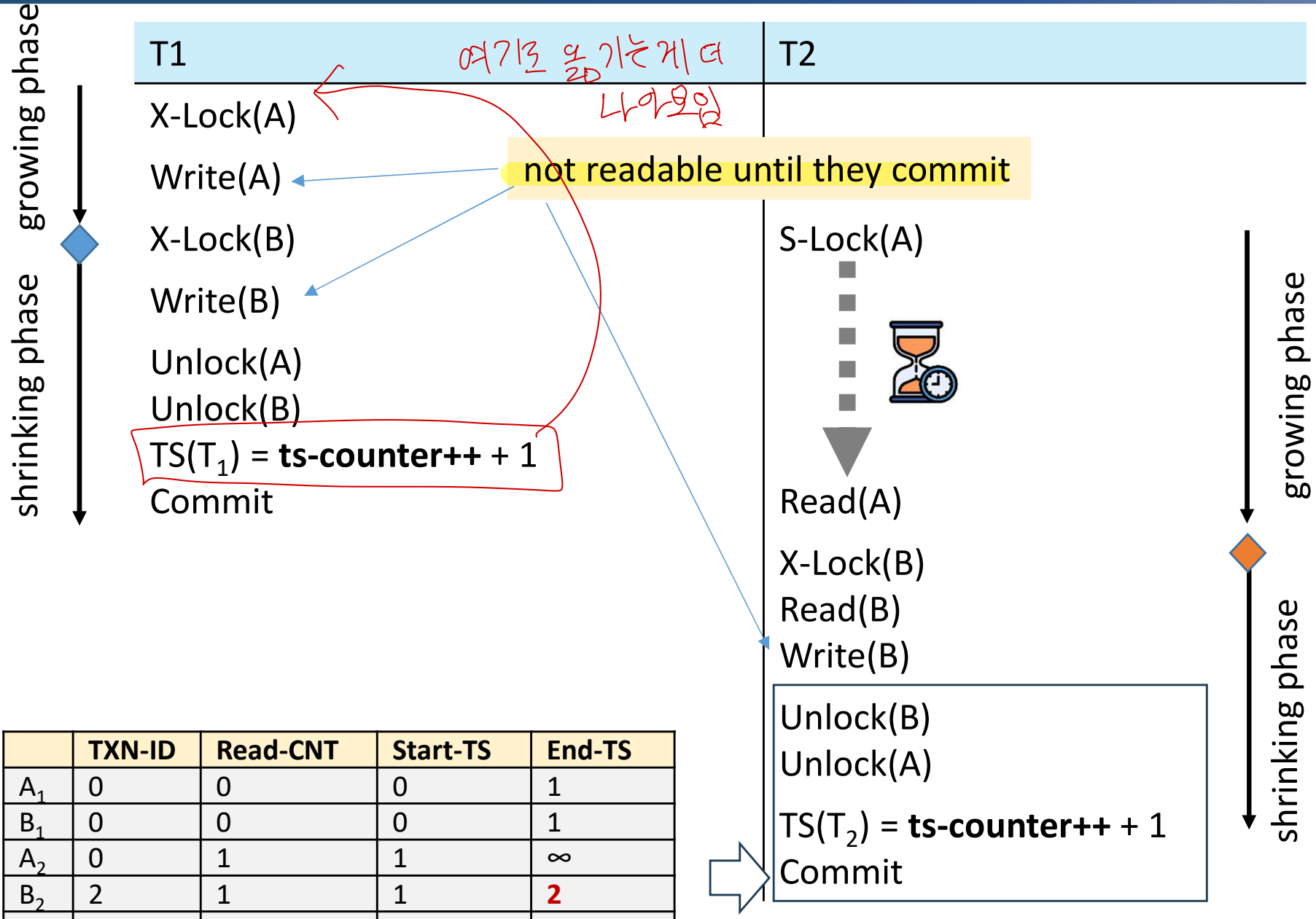
T1	T2
X-Lock(A)	
Write(A)	
X-Lock(B)	S-Lock(A)
Write(B)	
Unlock(A)	
Unlock(B)	
$TS(T_1)$ = **ts-counter++** + 1	
Commit	
	Read(A)
	X-Lock(B)
	Read(B)
	Write(B) // creates new B
	Unlock(B)
	Unlock(A)
	$TS(T_2)$ = **ts-counter++** + 1

growing phase

shrinking phase

	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	0	1
B_1	0	0	0	1
A_2	0	1	1	∞
B_2	2	1	1	∞
B_3	**2**	**0**	∞	∞

Multi-Version Two-Phase Locking Protocol (MV2PL) - Example



	TXN-ID	Read-CNT	Start-TS	End-TS
A_1	0	0	0	1
B_1	0	0	0	1
A_2	0	1	1	∞
B_2	2	1	1	**2**
B_3	2	0	**2**	∞

MVCC: Implementation Issues

- Creation of multiple versions increases storage overhead
  - Extra tuples
  - Extra space in each tuple for storing version information
- Versions need to be ***garbage collected***
  - E.g., if Q has two versions Q5 and Q9, and the oldest active transaction has timestamp > 5, Q5 will never be required again

<기존 방식과의 차이점>
multiversion → historical data를 읽어서 기존에 abort되는 경우를 해결해준다.