

---

# Concurrency And Recovery

# Knowledge Objectives

---

1. Refresh the basics on concurrency control
2. Explain the assumptions behind pessimistic strategies
3. Describe the time-stamping strategy
4. Explain the needed modifications for adding recoverability to time-stamping
5. Explain the benefits of dynamic time-stamping
6. Name the two main difficulties for distributed concurrency control
7. Elaborate on distributed time-stamping
8. Enumerate the components of a DBMS related to recovery and their associated tasks
9. Explain the 3 log rules

# Understanding Objectives

---

1. Given a history and a time-stamping concurrency control mechanism (either static or dynamic), simulate the behavior of the system
2. Explain how the CAP theorem can be adapted to produce CA, CP or AP systems
3. Formally justify why contention and consistency delay forfeit linear scalability

# Application Objectives

---

1. Make physical design decisions to achieve CP, AP, and CA systems (considering the CAP theorem)
2. Within AP systems (considering the CAP theorem), make different physical decisions to strength or relax C

# Reminder: Challenges in Data Distribution

---

## I. Distributed DB design

- Node distribution
- Data fragments
- Data allocation (replication)

## II. Distributed DB catalog

- Fragmentation trade-off: Where to place the DB catalog
  - Global or local for each node
  - Centralized in a single node or distributed
  - Single-copy vs. Multi-copy

## III. Distributed query processing

- Data distribution / replication
- Communication overhead

## IV. Distributed transaction management

- How to enforce the ACID properties
  - Replication trade-off: Queries vs. Data consistency between replicas (updates)
  - Distributed recovery system
  - Distributed concurrency control system

---

Concurrency and Recovery

# BASICS ON CONCURRENCY

# ACID properties

---

- *A*tomicity
- *C*onsistency
- *I*solation
- *D*urability

*Enforcing these properties is not only responsibility of the DBMS, but also of DBA, users, and application programmers*

# Activity: Basics on Concurrency

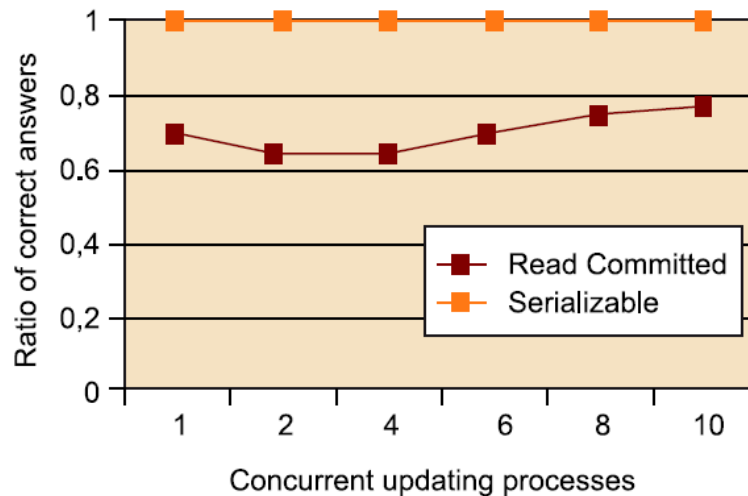
- ❑ Objective: Refresh the kind of interferences and isolation levels according to the SQL standard
- ❑ Tasks:
  1. (5') By pairs, for each isolation level described below think of an example showing the kind of interference (in brackets) it tries to avoid
    - I. What is the difference between Read Uncommitted and Unrepeatable Read?
    - II. And between Unrepeatable Read and Phantoms?
  2. Think of a kind of system (e.g., online market, a bank, etc.) that you could accommodate at each isolation level
  3. (5') Discussion
- **Read Uncommitted**: Avoids Lost Update (or Write-Write) interferences, which appear when data written by a Tx is lost, because another one overwrites it before the first Tx commits.
- **Read Committed**: Avoids Read Uncommitted (or Write-Read) interferences, which typically appear when a transaction reads (and uses) a value written by another Tx and the one who wrote it does not commit its results.
- **Repeatable Read**: Avoids Unrepeatable Read (or Read-Write) interferences, which appear when a tx reads some data, another Tx overwrites this data, and then the first one tries to read the same data again.
- **Serializable**: Avoids Inconsistent Analysis (or Phantoms) interferences, which appear when the result of accessing several granules is affected by changes made by other Txs, so that it does neither reflect the state before nor after the execution of those other Txs.



# Trade-Off: Performance Vs. Consistency

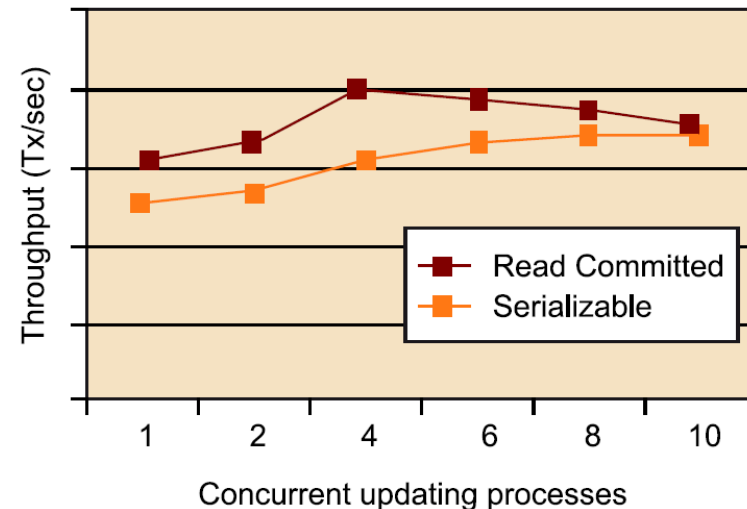
## Consistency (Ratio of correct answers)

Percentage of correct results depending on the number of concurrent transactions



## Performance (System throughput)

Throughput (transactions per second) depending on the number of concurrent transactions.



# Challenge IV: Distributed Tx management

---

- ❑ ACID properties are not always necessary
  - All can be relaxed
- ❑ Relaxing Consistency and Durability
  - Entails data loss
  - Save synchronization time
- ❑ Relaxing Atomicity and Isolation
  - Generate interferences
  - Save locks and contention

---

Concurrency and Recovery

# TIME-STAMPING

# Time-Stamping Concurrency Control

---

- ❑ Pessimistic technique
  - Not as much as locking
- ❑ Imposes a total order among transactions
  - Guarantees a history equivalent to the serial one following the order of BoTs
- ❑ When a potential conflict arrives, the order between transactions conflicting is checked
  - If the order is violated, the current transaction is canceled

# Structures for Time-Stamping

---

- For each transaction
  - Timestamp of the BoT
- For each granule
  - Timestamp of the youngest transaction reading it
    - $TSR(G)$
  - Timestamp of the youngest transaction writing it
    - $TSW(G)$

# Time-Stamping Algorithms

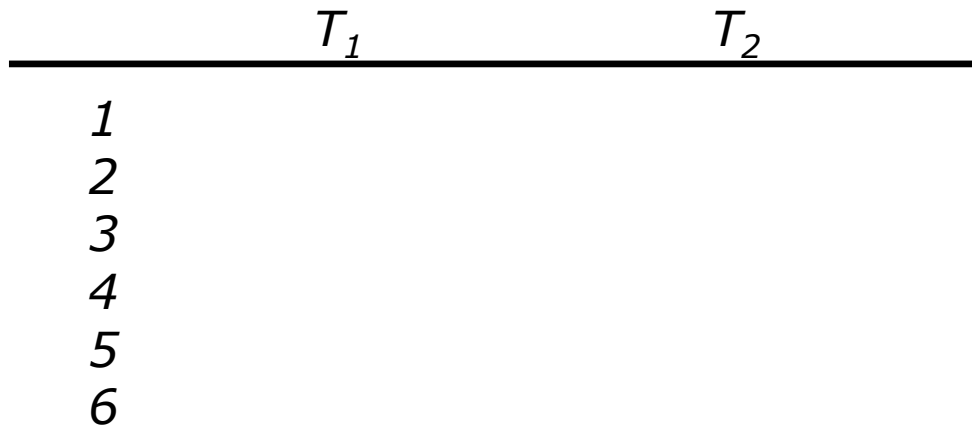
---

```
procedure read(T, G) is  
  if TSW(G)  $\leq$  TS(T) then  
    TSR(G) = max(TSR(G), TS(T));  
    R(G);  
  else  
    abort(T);  
  endif  
endProcedure
```

```
procedure write(T, G) is  
  if TSW(G)  $\leq$  TS(T) and TSR(G)  $\leq$  TS(T) then  
    TSW(G) = TS(T);  
    W(G);  
  else  
    abort(T);  
  endIf  
endProcedure
```

# Example of Time-Stamping (I)

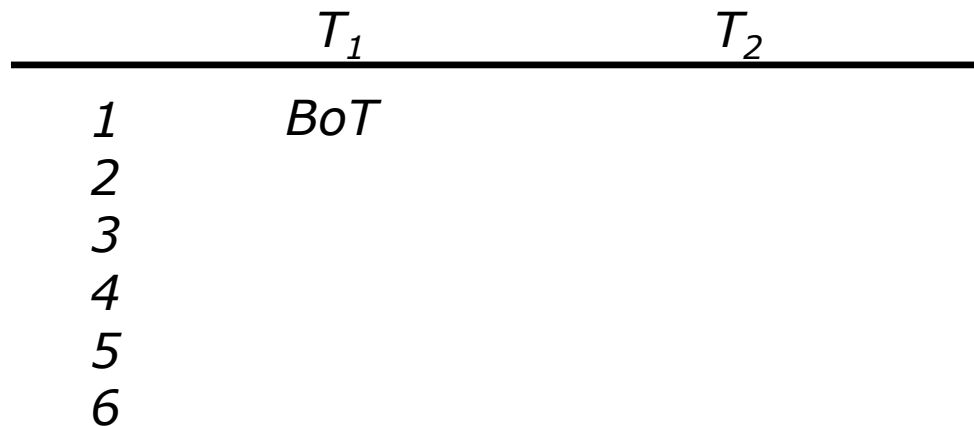
---



$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (I)



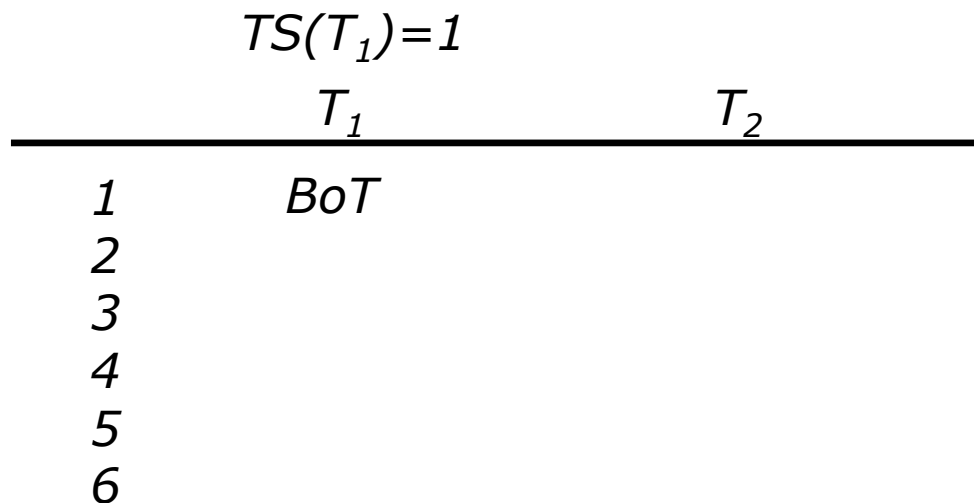
$$TSR(A)=0$$

$$TSW(A)=0$$



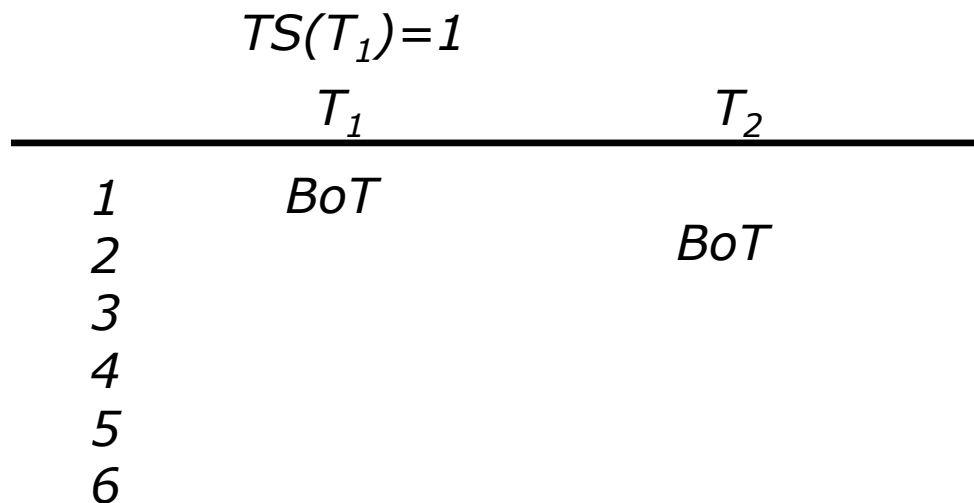
# Example of Time-Stamping (I)

---



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (I)



$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (I)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3		
4		
5		
6		

$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (I)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		
5		
6		

$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (I)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		
5		
6		

$TSR(A)=1$

$TSW(A)=0$

# Example of Time-Stamping (I)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		
6		

$TSR(A)=1$

$TSW(A)=0$

# Example of Time-Stamping (I)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		
6		

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (I)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		$W(A)$
6		

$TSR(A)=2$

$TSW(A)=0$



# Example of Time-Stamping (I)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		$W(A)$
6		

$TSR(A)=2$

$TSW(A)=2$

# Example of Time-Stamping (I)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		$W(A)$
6	$R(A)$	

$TSR(A)=2$

$TSW(A)=2$

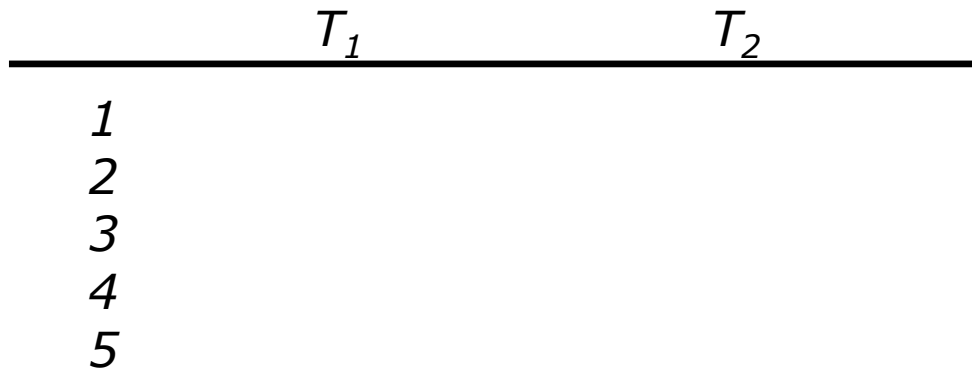
# Example of Time-Stamping (I)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		$W(A)$
6	$R(A)$	
	↓	
	$Abort(T_1)$	
	↓	
	$Restart(T_1)$	

$TSR(A)=2$   
 $TSW(A)=2$

# Example of Time-Stamping (II)

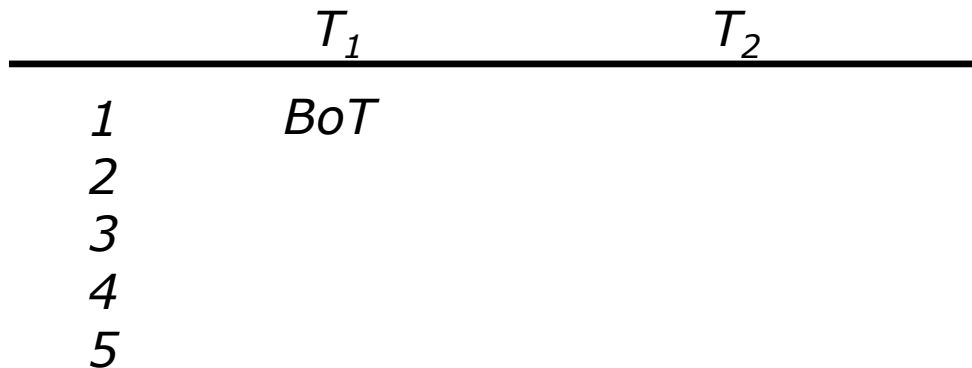
---



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (II)

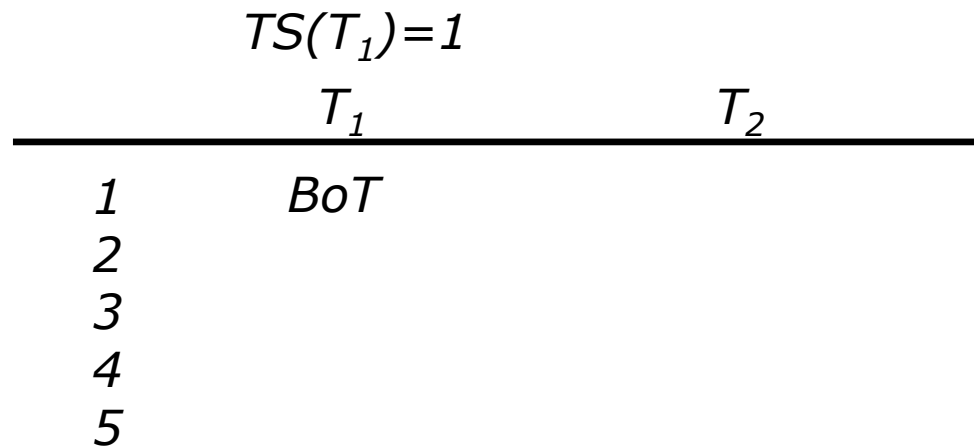
---



$$TSR(A)=0$$

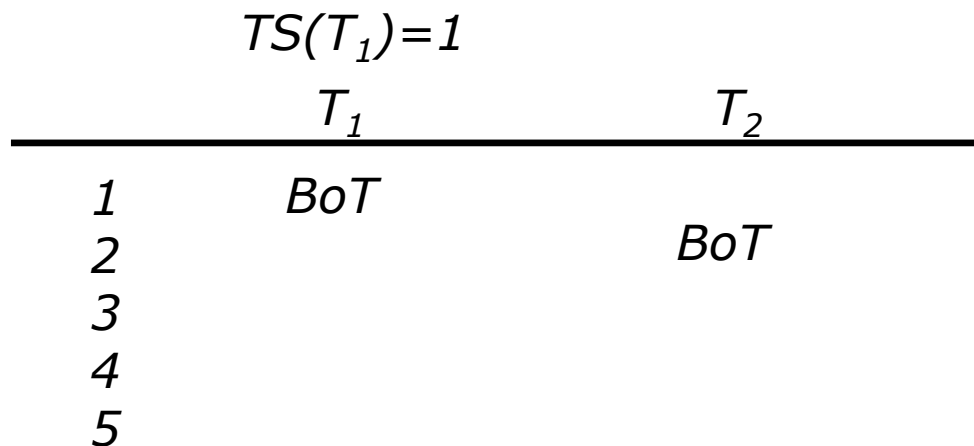
$$TSW(A)=0$$

# Example of Time-Stamping (II)



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (II)



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (II)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3		
4		
5		

$TSR(A)=0$

$TSW(A)=0$



# Example of Time-Stamping (II)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		
5		

$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (II)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		
5		

$TSR(A)=1$

$TSW(A)=0$

# Example of Time-Stamping (II)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		

$TSR(A)=1$

$TSW(A)=0$

# Example of Time-Stamping (II)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5		

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (II)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5	$W(A)$	

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (II)

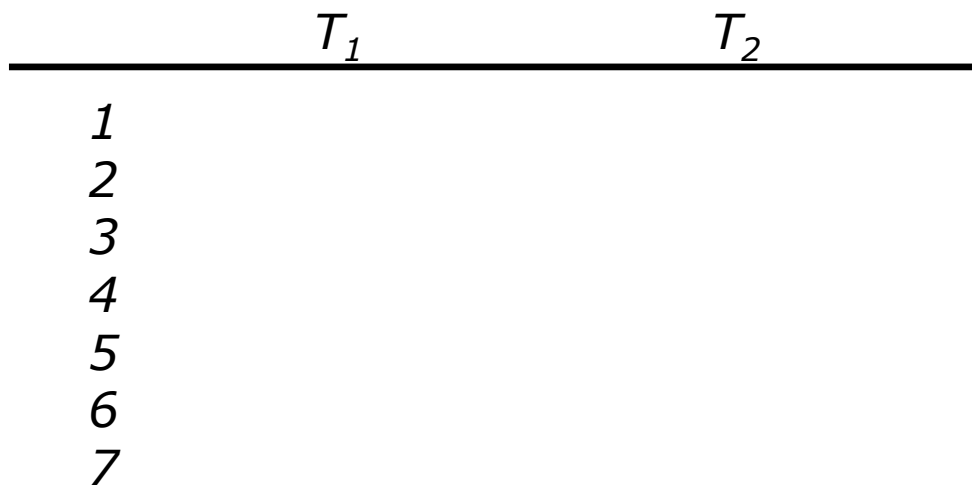
	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		$R(A)$
5	$W(A)$	
	↓	
	$Abort(T_1)$	
	↓	
	$Restart(T_1)$	

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (III)

---

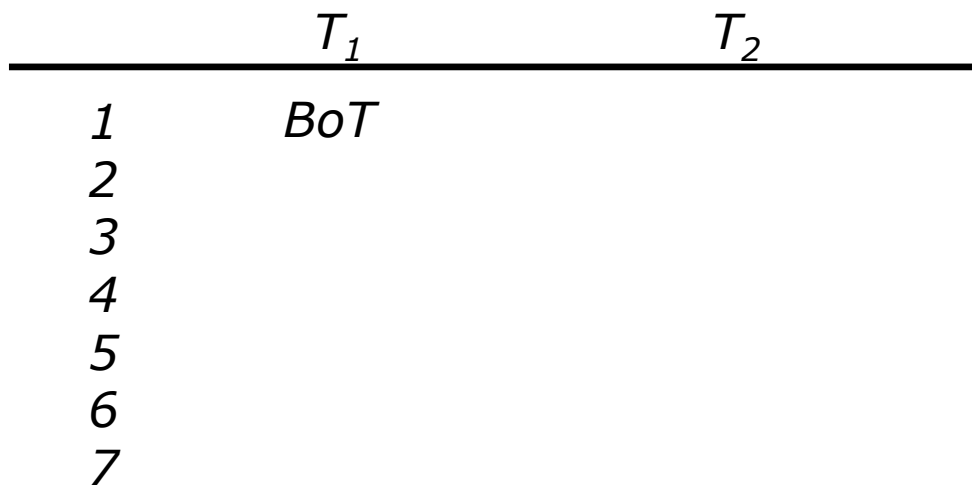


$$TSR(A)=0$$

$$TSW(A)=0$$

# Example of Time-Stamping (III)

---

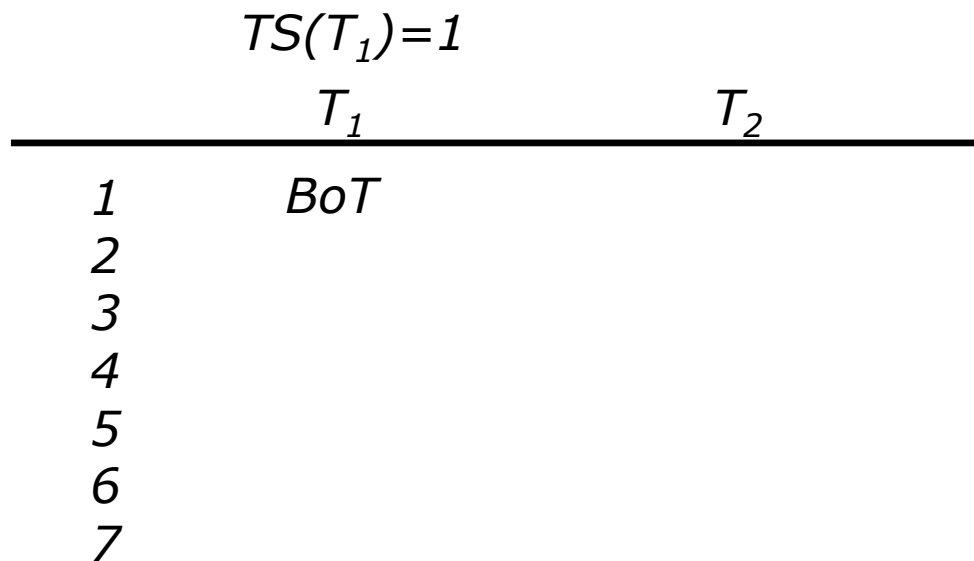


$$TSR(A)=0$$

$$TSW(A)=0$$

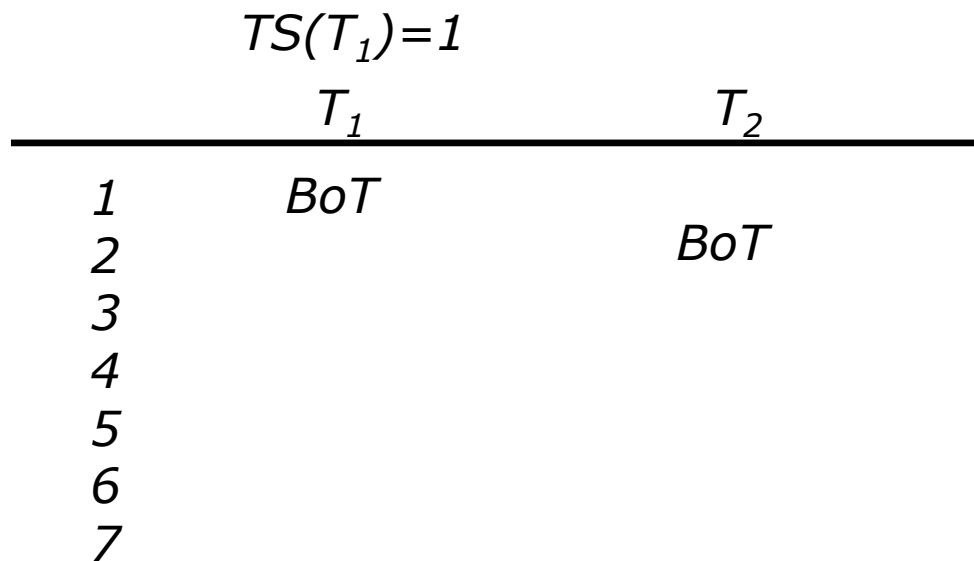


# Example of Time-Stamping (III)



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (III)



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (III)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3		
4		
5		
6		
7		

$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		
5		
6		
7		

$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4		
5		
6		
7		

$TSR(A)=1$

$TSW(A)=0$

# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		
6		
7		

$TSR(A)=1$

$TSW(A)=0$

# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		
6		
7		

$TSR(A)=1$

$TSW(A)=1$

# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		$R(A)$
6		
7		

$TSR(A)=1$

$TSW(A)=1$



# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		$R(A)$
6		
7		

$TSR(A)=2$

$TSW(A)=1$

# Example of Time-Stamping (III)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		$R(A)$
6		$Commit$
7		

$TSR(A)=2$

$TSW(A)=1$

# Example of Time-Stamping (III)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		$R(A)$
6		$Commit$
7	$Rollback(T_1)$	

$TSR(A)=2$

$TSW(A)=1$


# Example of Time-Stamping (III)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3	$R(A)$	
4	$W(A)$	
5		$R(A)$
6		$Commit$
7	$Rollback(T_1)$	

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (III)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	BoT	
2		BoT
3	$R(A)$	
4	$W(A)$	
5		$R(A)$
6		Commit
7	Rollback( $T_1$ )	
		Restart( $T_2$ )

$TSR(A)=2$

$TSW(A)=0$

# Enforcing Recoverability

## a) Check it at commit time

- a) If  $T_2$  reads a value of  $T_1$  (being  $TS(T_1) < TS(T_2)$ ), then  $T_2$  has to wait the end of  $T_1$  (and finish in the same way)
- b) As soon as a transaction aborts, we abort all transactions that read values written by it

## b) Check it at operation time

procedure read( $T_i, G$ ) is  
     if  $TSW(G) \leq TS(T_i)$  **and**  $T_{w(G)}$  **comitted** then  
         ...  
     endProcedure

procedure write( $T_i, G$ ) is  
     if  $TSW(G) \leq TS(T_i)$  and  $TSR(G) \leq TS(T_i)$  **and**  $T_{w(G)}$  **comitted** then  
         ...  
     endProcedure

# Enforcing Recoverability

## a) Check it at commit time

- a) If  $T_2$  reads a value of  $T_1$  (being  $TS(T_1) < TS(T_2)$ ), then  $T_2$  has to wait the end of  $T_1$  (and finish in the same way)
- b) As soon as a transaction aborts, we abort all transactions that read values written by it

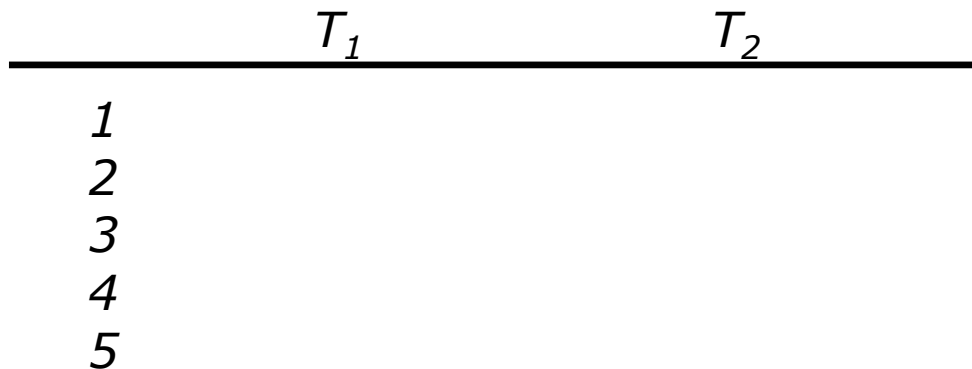
## b) Check it at operation time

procedure read( $T_i, G$ ) is  
     if  $TSW(G) \leq TS(T_i)$  **and**  $T_{w(G)}$  **comitted** then  
     ...  
endProcedure

procedure write( $T_i, G$ ) is  
     if  $TSW(G) \leq TS(T_i)$  and  $TSR(G) \leq TS(T_i)$  **and**  $T_{w(G)}$  **comitted** then  
     ...  
endProcedure

# Example of Time-Stamping (IV)

---



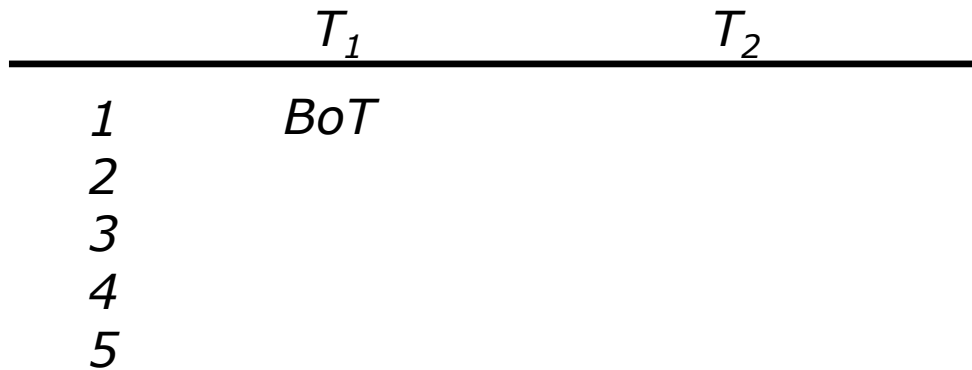
$TSR(A)=0$

$TSW(A)=0$



# Example of Time-Stamping (IV)

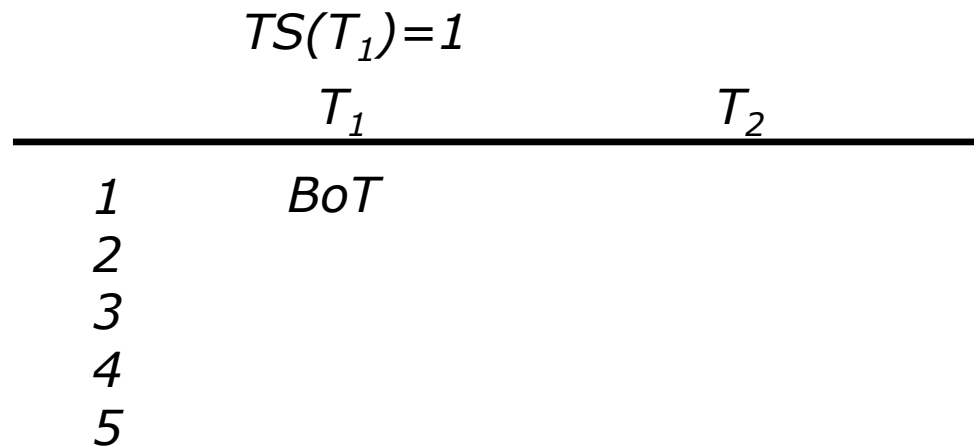
---



$TSR(A)=0$   
 $TSW(A)=0$

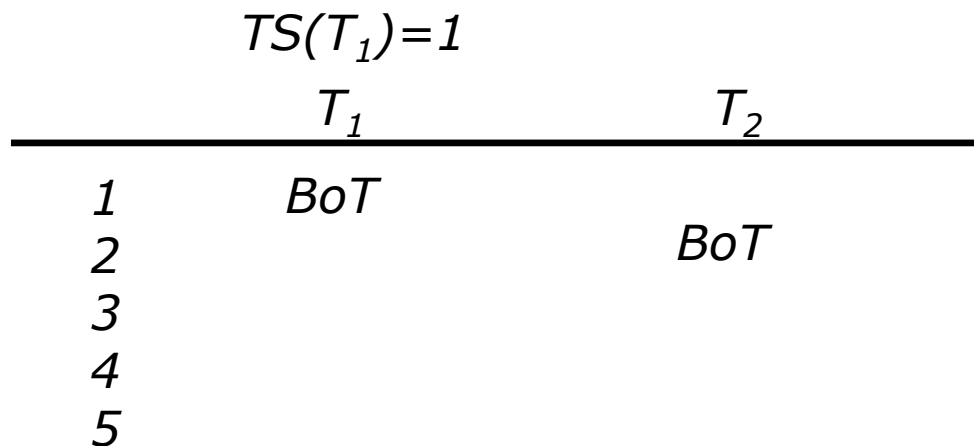
# Example of Time-Stamping (IV)

---



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (IV)



$TSR(A)=0$   
 $TSW(A)=0$

# Example of Time-Stamping (IV)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3		
4		
5		

$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (IV)

	$TS(T_1)=1$ $T_1$	$TS(T_2)=2$ $T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		
5		

$TSR(A)=0$

$TSW(A)=0$

# Example of Time-Stamping (IV)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		
5		

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (IV)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5		

$TSR(A)=2$

$TSW(A)=0$

# Example of Time-Stamping (IV)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5		

$TSR(A)=2$

$TSW(A)=2$



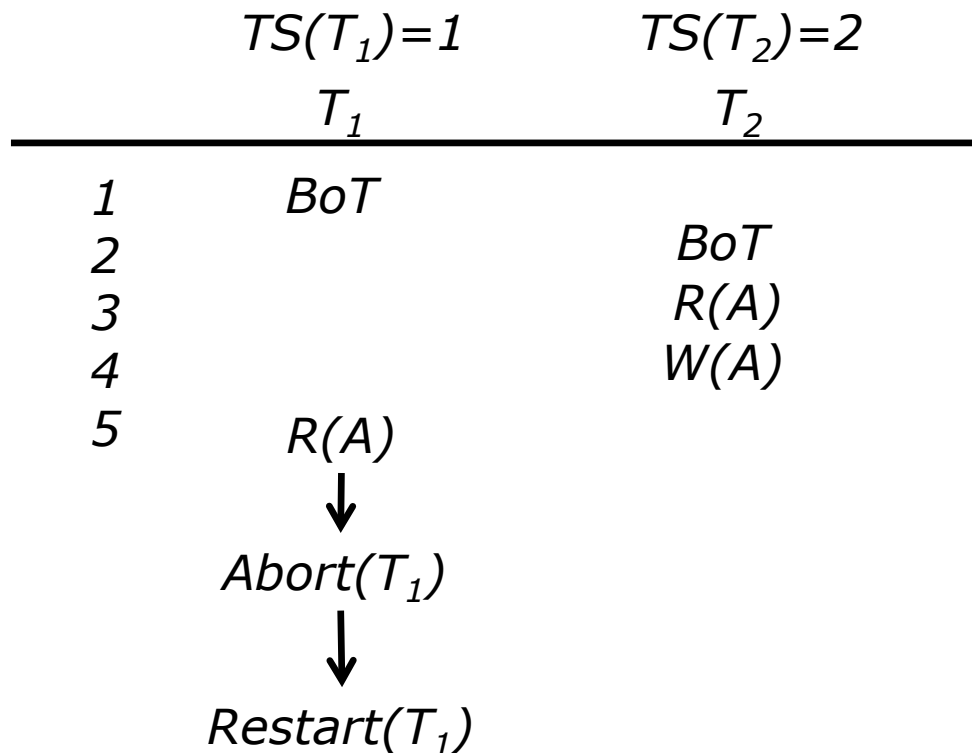
# Example of Time-Stamping (IV)

	$TS(T_1)=1$	$TS(T_2)=2$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5	$R(A)$	

$TSR(A)=2$

$TSW(A)=2$

# Example of Time-Stamping (IV)



$TSR(A)=2$

$TSW(A)=2$

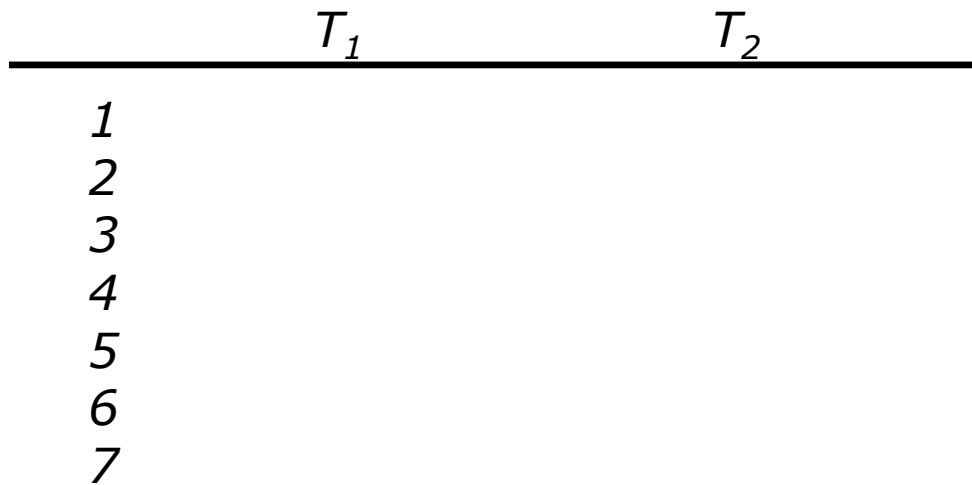
# Dynamic Time-Stamping

- Delay the assignment of the timestamp to the transactions as much as possible
  - Until there is a potential conflict between the current transaction and another one
    - Inject this code in *read* and *write* procedures

```

foreach  $T_i \in \text{setOfActiveTx}$  do
  if ( $G \in \text{RS}(T_i) \cap \text{WS}(T)$ ) or ( $G \in \text{WS}(T_i) \cap \text{RS}(T)$ ) then
    if  $\text{TS}(T_i) == \text{null}$  and  $\text{TS}(T) == \text{null}$  then
      assign both timestamps so that  $\text{TS}(T_i) < \text{TS}(T)$ 
    elseif  $\text{TS}(T_i) == \text{null}$  or  $\text{TS}(T) == \text{null}$  then
      assign one timestamp so that  $\text{TS}(T_i) < \text{TS}(T)$ 
    endif
  endif
endforeach
  
```

# Example of Time-Stamping (V)



$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{\}$

$WS(T_2)=\{\}$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		
3		
4		
5		
6		
7		

$$TSR(A)=0$$

$$TSW(A)=0$$

$$RS(T_1)=\{\}$$

$$WS(T_1)=\{\}$$

$$RS(T_2)=\{\}$$

$$WS(T_2)=\{\}$$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		
4		
5		
6		
7		

$$TSR(A)=0$$

$$TSW(A)=0$$

$$RS(T_1)=\{\}$$

$$WS(T_1)=\{\}$$

$$RS(T_2)=\{\}$$

$$WS(T_2)=\{\}$$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		
5		
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{\}$

$WS(T_2)=\{\}$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		
5		
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{\}$



# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5		
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{\}$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5		
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5	$R(A)$	
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

# Example of Time-Stamping (V)

	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5	$R(A)$	
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{A\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

# Example of Time-Stamping (V)

	$TS(T_1)=2$	$TS(T_2)=1$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5	$R(A)$	
6		
7		

$TSR(A)=0$

$TSW(A)=0$

$RS(T_1)=\{A\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

# Example of Time-Stamping (V)

	$TS(T_1)=2$	$TS(T_2)=1$
	$T_1$	$T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5	$R(A)$	
6		
7		

$TSR(A)=2$

$TSW(A)=1$

$RS(T_1)=\{A\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

# Example of Time-Stamping (V)

	$TS(T_1)=2$ $T_1$	$TS(T_2)=1$ $T_2$
1	$BoT$	
2		$BoT$
3		$R(A)$
4		$W(A)$
5	$R(A)$	
6		$Commit$
7		

$TSR(A)=2$

$TSW(A)=1$

$RS(T_1)=\{A\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

# Example of Time-Stamping (V)

	$TS(T_1)=2$ $T_1$	$TS(T_2)=1$ $T_2$
1	<i>BoT</i>	
2		<i>BoT</i>
3		$R(A)$
4		$W(A)$
5	$R(A)$	
6		<i>Commit</i>
7	<i>Commit</i>	

$TSR(A)=2$

$TSW(A)=1$

$RS(T_1)=\{A\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{A\}$

$WS(T_2)=\{A\}$

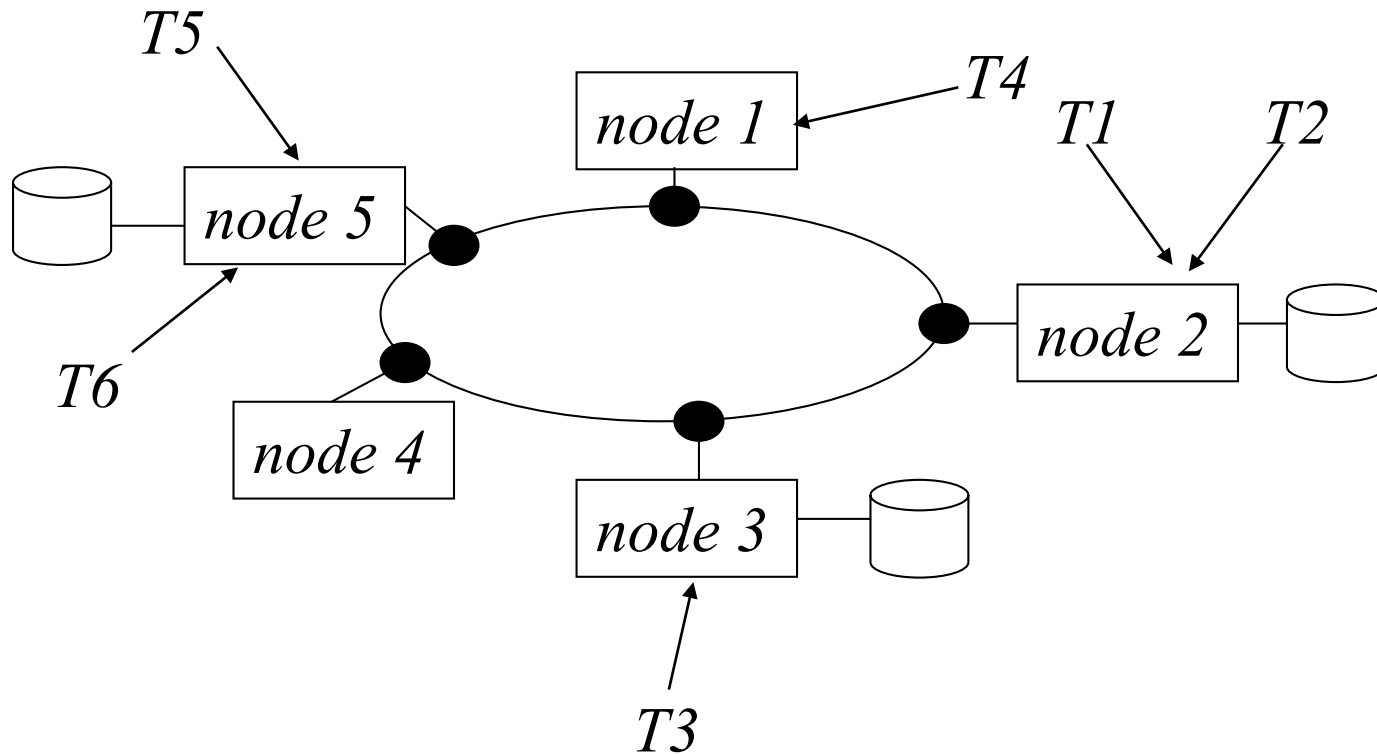


---

Concurrency and Recovery

# CONCURRENCY CONTROL IN DISTRIBUTED DATABASES

# Concurrency Control in DDBs



# Motivation: Global Serializability

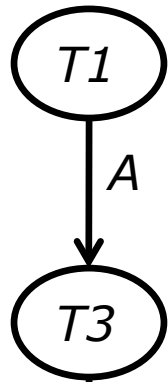
BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A)					R(C)
C					W(C)
		R(B)	R(C)		
		W(B)	C		
	R(B)				R(D)
	C			R(D)	
		R(A)		C	
		W(A)			W(D)
		C			C

# Motivation: Global Serializability

*T1*

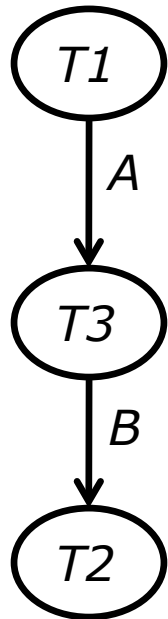
BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A)					R(C)
C					W(C)
		R(B)	R(C)		
		W(B)	C		
	R(B)				R(D)
	C			R(D)	
		R(A)		C	
		W(A)			W(D)
		C			C

# Motivation: Global Serializability



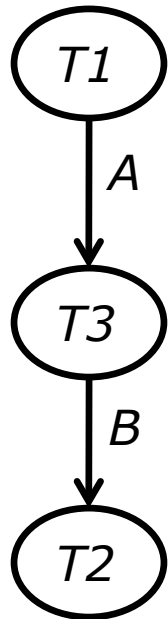
BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A)					R(C)
C					W(C)
		R(B)	R(C)		
		W(B)	C		
	R(B)				R(D)
	C			R(D)	
		R(A)		C	
		W(A)			W(D)
		C			C

# Motivation: Global Serializability

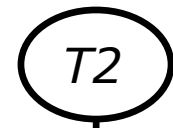


BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A) C					R(C) W(C)
		R(B) W(B)	R(C) C		
	R(B) C				R(D)
		R(A) W(A) C		R(D) C	
					W(D) C

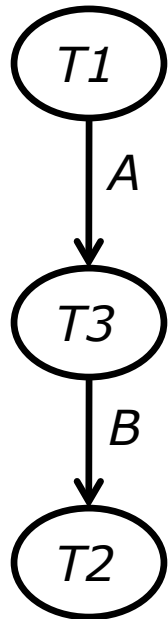
# Motivation: Global Serializability



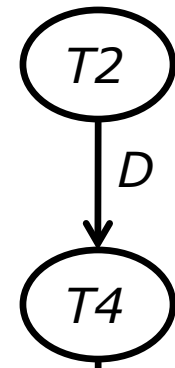
BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A) C					R(C) W(C)
		R(B) W(B)	R(C) C		
	R(B) C				R(D)
		R(A) W(A) C		R(D) C	
					W(D) C



# Motivation: Global Serializability

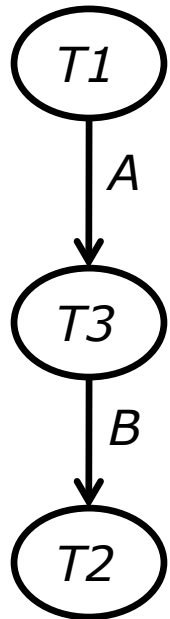


BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A) C					R(C) W(C)
		R(B) W(B)	R(C) C		
	R(B) C				R(D)
		R(A) W(A) C		R(D) C	
					W(D) C

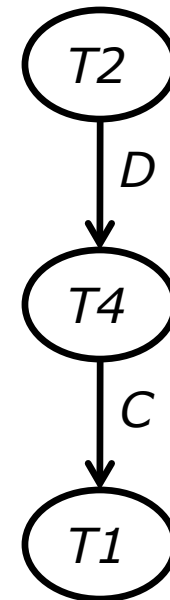




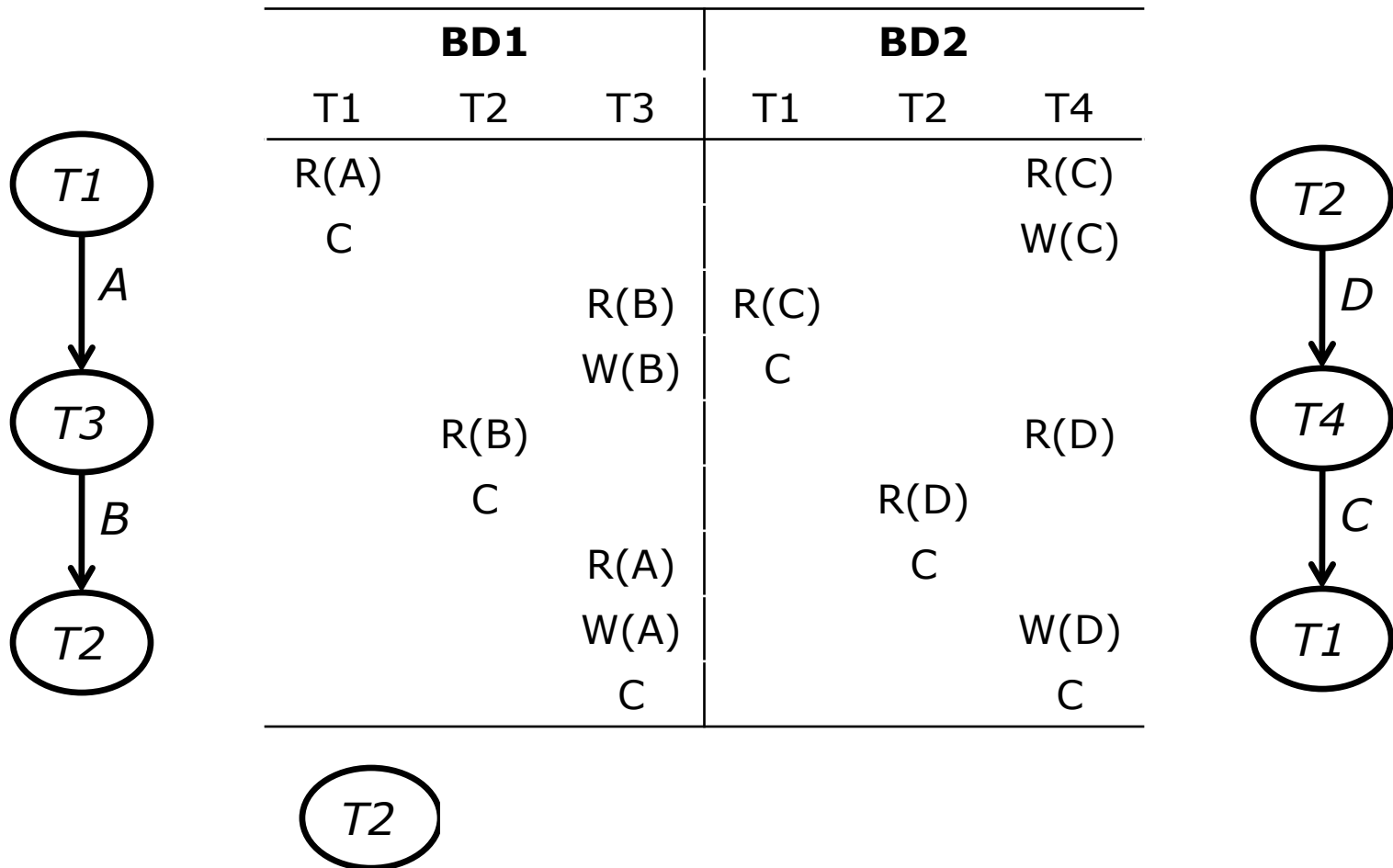
# Motivation: Global Serializability



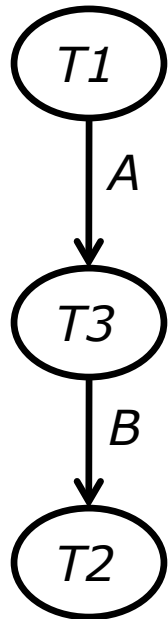
BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A) C					R(C) W(C)
		R(B) W(B)	R(C) C		
	R(B) C				R(D)
		R(A) W(A) C	R(D) C		W(D) C



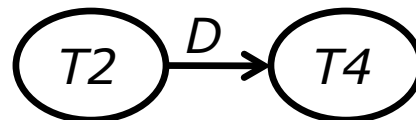
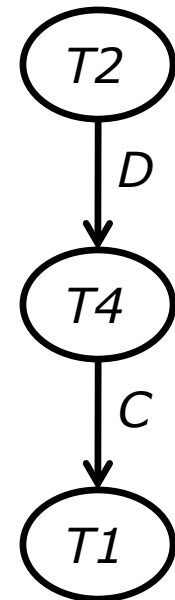
# Motivation: Global Serializability



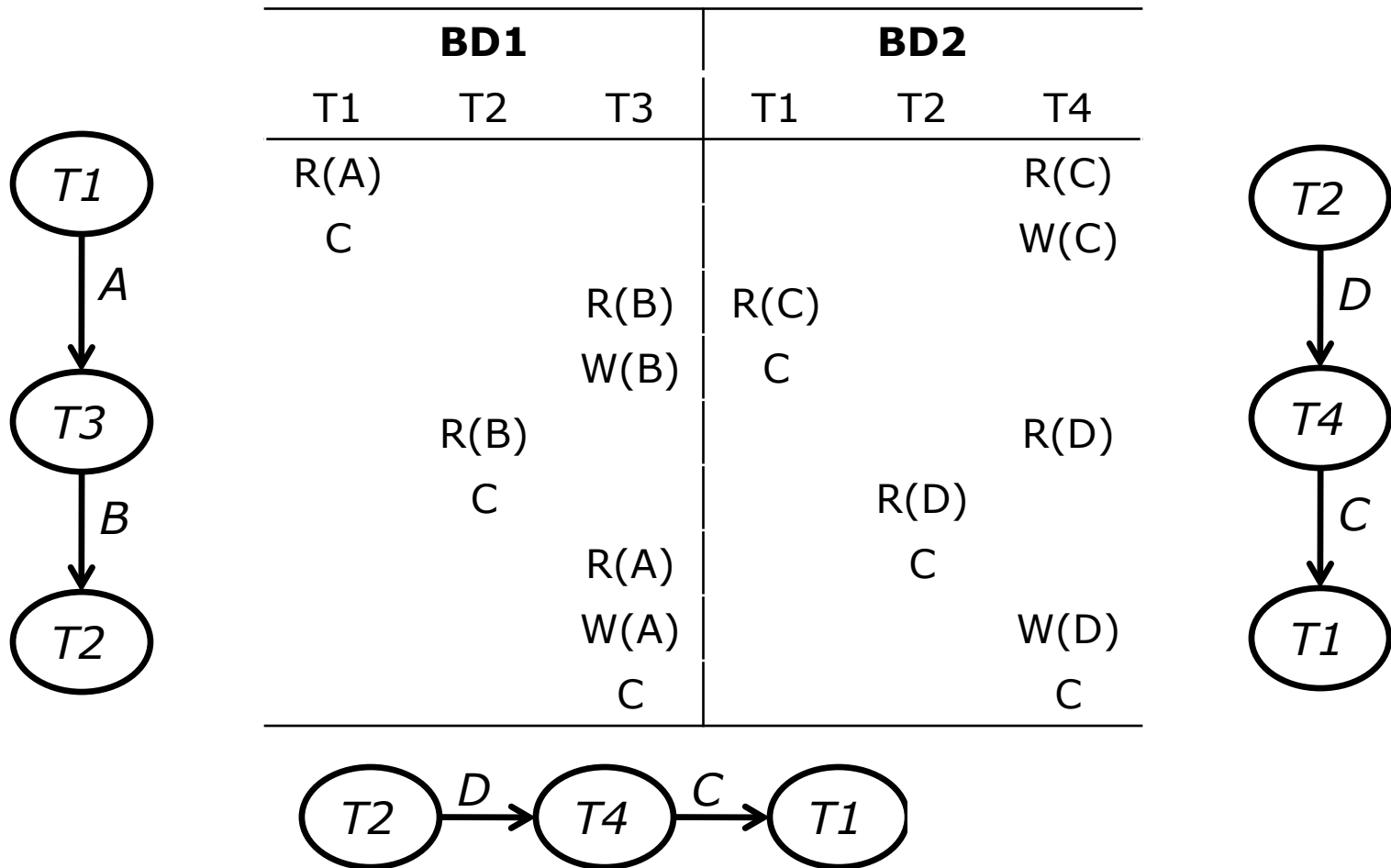
# Motivation: Global Serializability



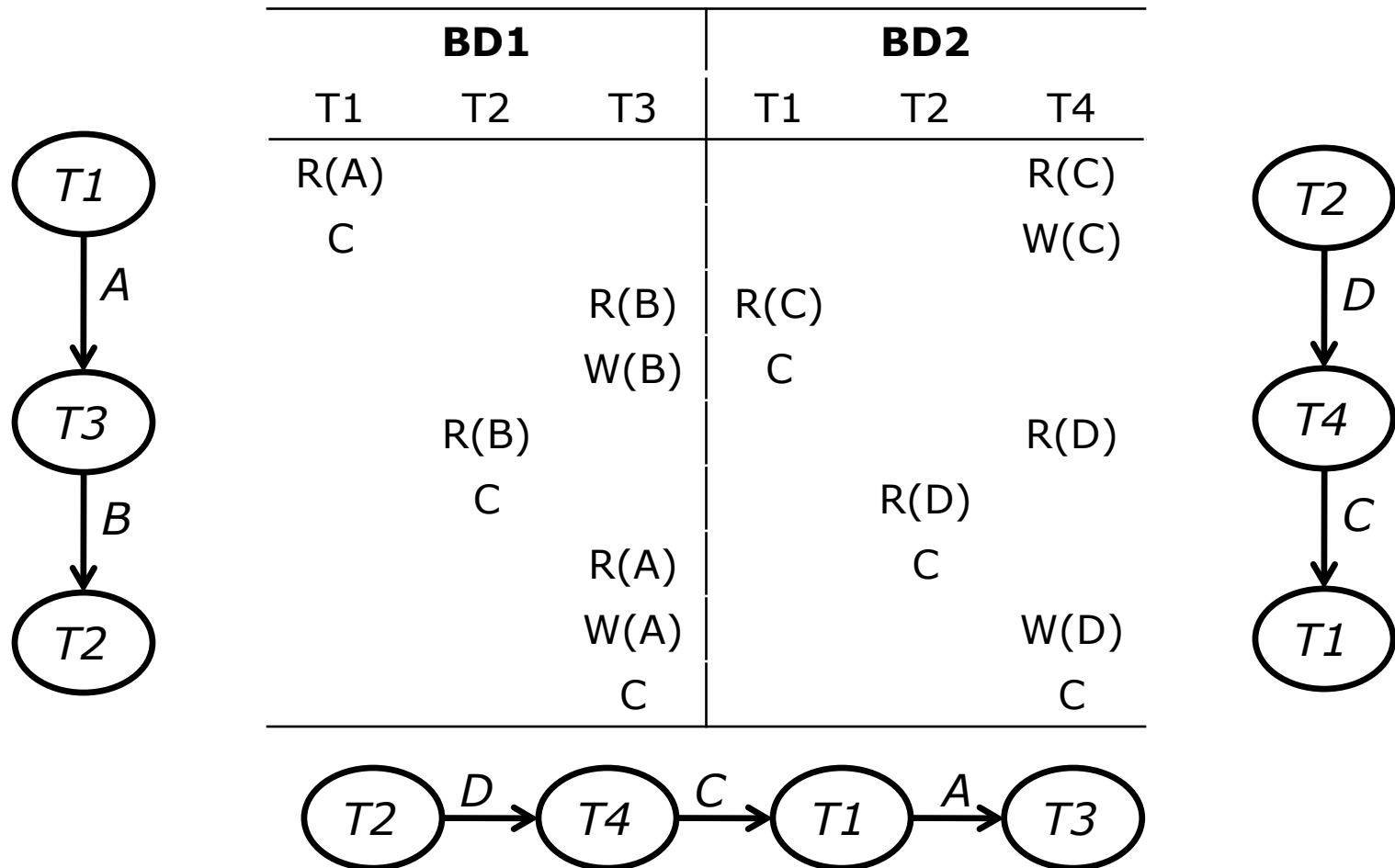
BD1			BD2		
T1	T2	T3	T1	T2	T4
R(A) C					R(C) W(C)
		R(B) W(B)	R(C) C		
	R(B) C				R(D)
		R(A) W(A) C	R(D) C		W(D) C



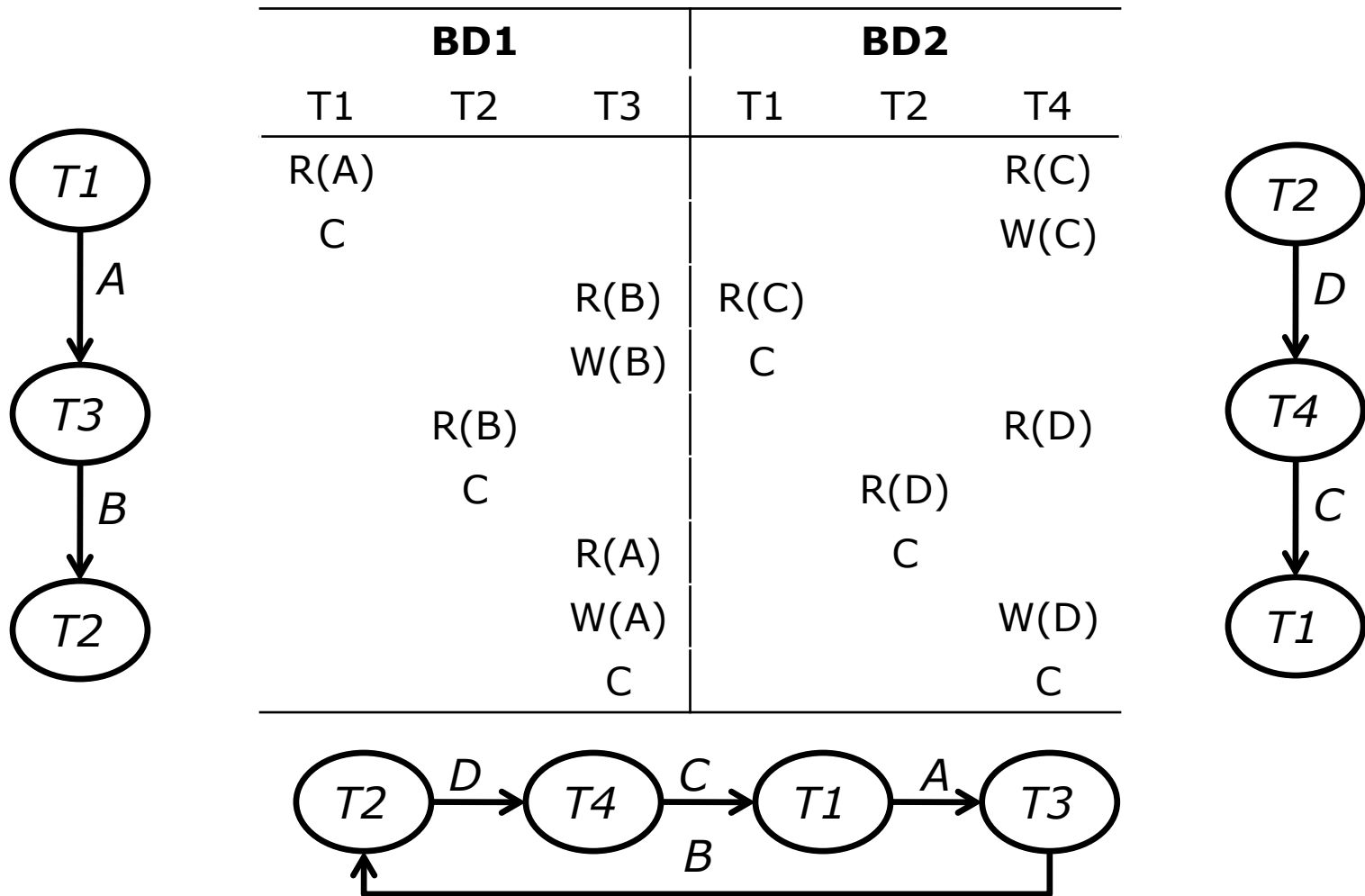
# Motivation: Global Serializability



# Motivation: Global Serializability



# Motivation: Global Serializability



# Problems in Distributed Transactions

---

- ❑ What needs to be solved
  - Guarantee **global** serializability
    - ❑ Detect **global** deadlocks
      - If using locking concurrency control
  - Guarantee **global** atomicity and recoverability
- ❑ As shown, it cannot be solved locally
  - A global protocol must be set

# Distributed Time-Stamping

- Massively used in distributed systems
  - Works exactly as the local version
- The only difference is the need of **globally unique timestamps**
- To such end, the Lamport clocks axiom is used
  - Lamport clocks axiom
    - "A message always arrives after it was sent"
  - Each timestamp contains
    - Time
    - Originating node
  - Clock
    - Every local action increases the clock
    - Every external message sets the local clock to  $C = \max(C, TS(T_x).T + 1)$
  - Time comparison
 
$$TS(T_1) < TS(T_2) \Leftrightarrow TS(T_1).T < TS(T_2).T \vee (TS(T_1).T = TS(T_2).T \wedge TS(T_1).N < TS(T_2).N)$$



# Example of Distributed TS

## □ Timestamp comparisons:

TS(T <sub>1</sub> )		TS(T <sub>2</sub> )		Comparison
T	N	T	N	TS(T <sub>1</sub> ) < TS(T <sub>2</sub> )
4	1	5	2	true
5	1	4	2	false
4	1	4	2	true
4	2	4	1	false
4	1	4	1	impossible

---

Concurrency and Recovery

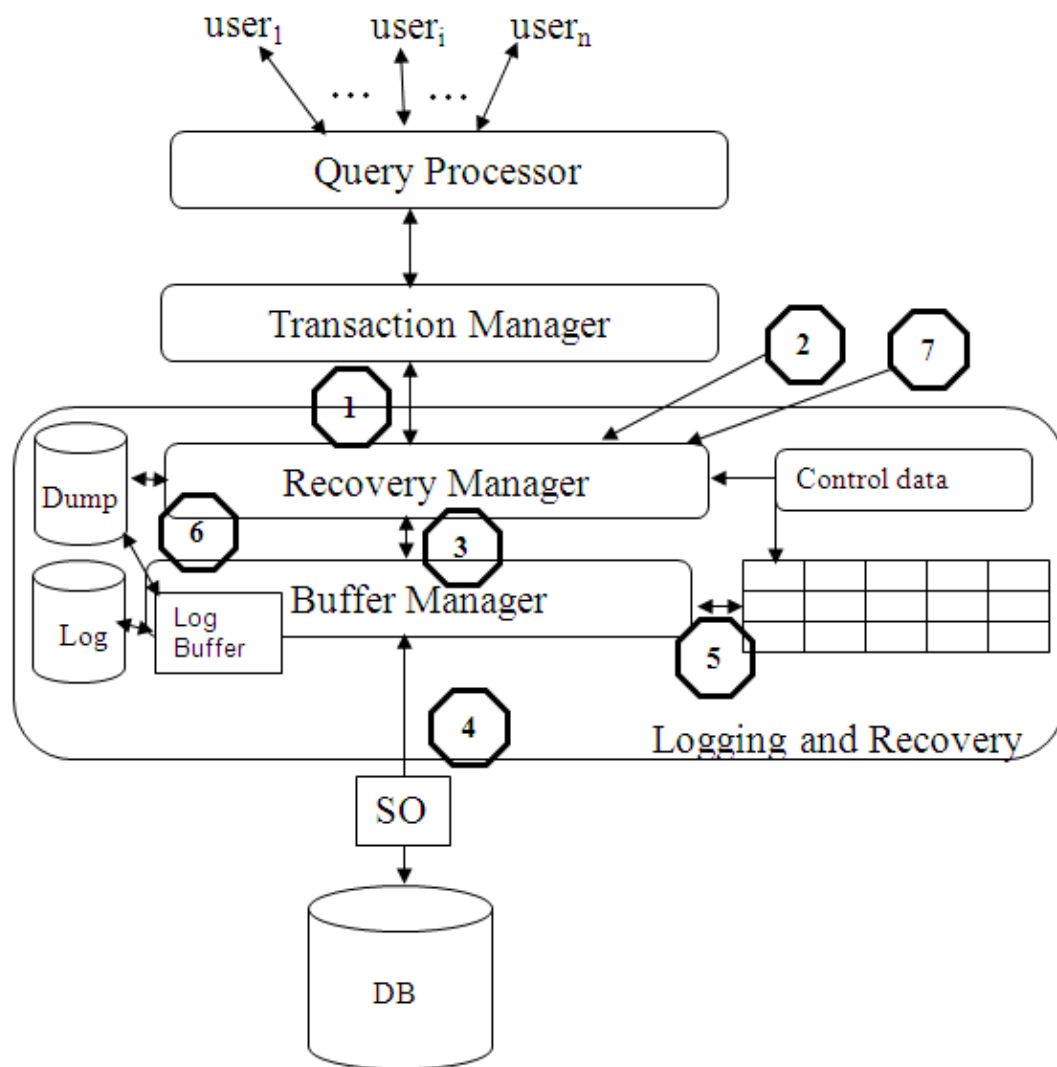
# **BASICS ON RECOVERY**

# Motivation

---

- ❑ Failures must be masked to the users of transaction-based systems
  - Transaction failure
    - ❑ Voluntary rollbacks
    - ❑ Aborted by the DBMS
      - Deadlocks, interferences...
  - System and media failures (crash recovery)
    - ❑ E.g., hard disks or network failures
  - Disasters (archive recovery)
- ❑ Guarantee the atomicity (A) and durability (D) properties of ACID
  - Undo recovery for failed transactions (A)
  - Redo recovery for committed transactions (D)
- ❑ Logging is the task of collecting redundant data needed for recovery
  - Protocol data recording transactions executed and which changes have been performed by them
  - Checkpoints

# DBMS Components



1. *R, W, C, Abort*
2. *Restart: bring the DB back to a consistent state by undoing and redoing.*
3. *Fetch Ops. (from external memory to the buffer pool and then flush -i.e., to the permanent DB- ).*
4. *read\_page, write\_page, performed by the OS under DBMS petition.*
5. *Ops. over the buffer pool*
6. *Logging of writing operations.*
7. *Dump the DB.*

# The Log: Rules

---

- ❑ To force sufficient log information reaches the stable log
  - Redo rule (*commit rule*): Redo log information must be written, at the latest, in phase 1 of commit
  - Undo rule (*write ahead logging*): Undo log information must be written to log before *flushing* the pages to disk
  - Log information must not be discarded from the temporary log file, unless it is guaranteed that it is not longer necessary for recovery
    - ❑ The corresponding page has reached the permanent DB

---

Concurrency and Recovery

# DISTRIBUTED RECOVERY

# Distributed Recovery

---

- ❑ New types of failure must be considered
  - Communication link failure, network partitioning, delayed / lost messages, remote site failure
- ❑ Distributed commit protocols
  - Two-phase commit (2PC)
  - Three-phase commit (3PC)
  - **But they are typically not implemented in NOSQL since 2PC and 3PC compromise the availability of the system**
- ❑ Logs, however, are still used in distributed systems
  - Typically:
    - ❑ In the primary servers storing the catalog
    - ❑ In the secondary servers storing the primary replica when primary versioning is activated
  - In such cases, it is a regular log acting locally (not globally):
    - ❑ Before- and after-images
    - ❑ High-level actions
    - ❑ Actions of the distributed commit protocols are also logged
      - Force-written

---

Concurrency and Recovery

# CAP THEOREM



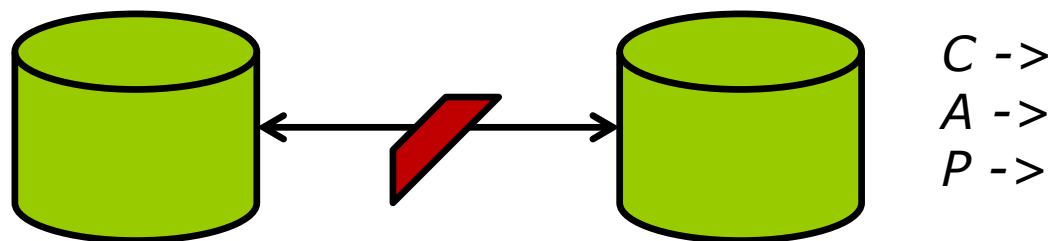
# On the Need of a New Architecture

---

- ❑ Distribution is needed to overcome the 3 challenges presented
  - To provide scalability, efficiency (by means of parallelism), reliability / availability
    - ❑ But RDBMS do not meet them (RDBMS bottlenecks)
- ❑ **CAP theorem formulation**: There is a trade-off in distributed systems; either availability or consistency can be always guaranteed (not both)
  - RDBMS choose consistency
  - NOSQL systems, most of the times, choose availability

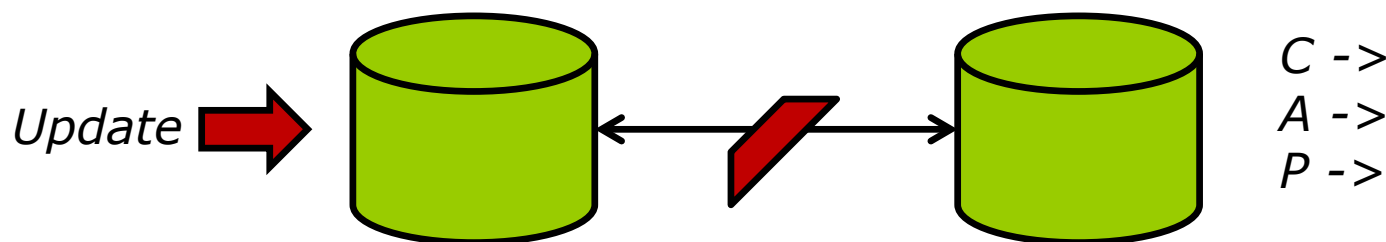
# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



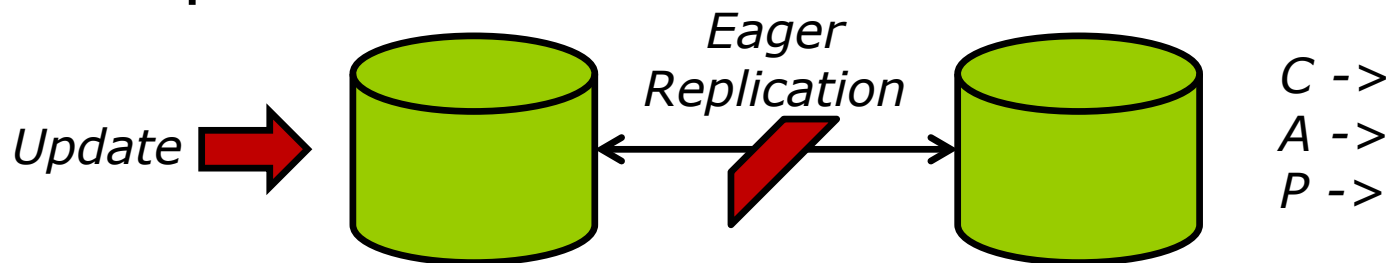
# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



# CAP Theorem

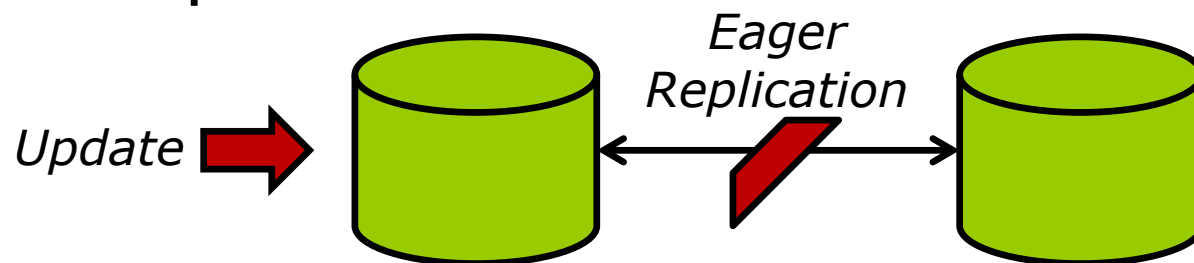
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).

## □ Example:



### **ACID**

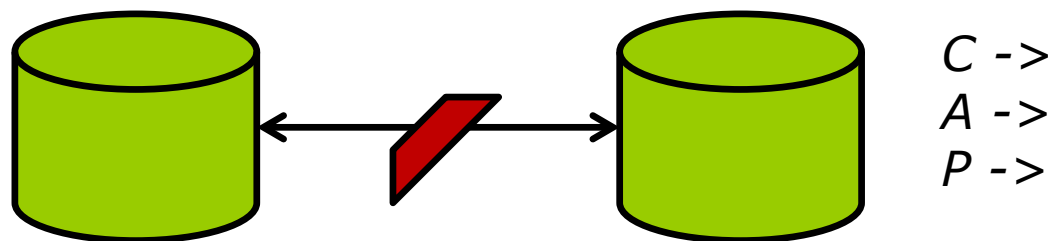
C -> OK  
 A -> NO  
 P -> OK

# Strong Consistency

- ❑ Consistency does not always mean to update ALL other replicas. It can be achieved with less updates
- ❑ Definitions
  - $N$ : #replicas
  - $W$ : #replicas that have to be written
  - $R$ : #replicas that need to be read
- ❑ Typical configurations
  - Fault tolerant system  $\Rightarrow N=3; R=2; W=2$
  - Massive replication for read scaling  $\Rightarrow R=1$
  - ROWA  $\Rightarrow R=1; W=N$ 
    - ❑ Fast read
    - ❑ Slow write (low probability of succeeding)
  - Inconsistency window  $\Rightarrow W < N$
  - Eventually consistent  $\Rightarrow R+W \leq N$ 
    - ❑ Both sets may not overlap
  - Potential conflict  $\Rightarrow W < (N+1)/2$
- ❑ Strong consistency is only guaranteed if  $W+R > N$

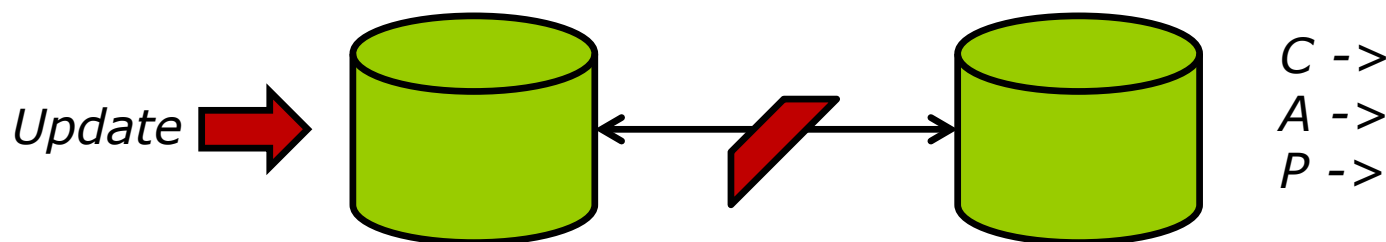
# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



# CAP Theorem

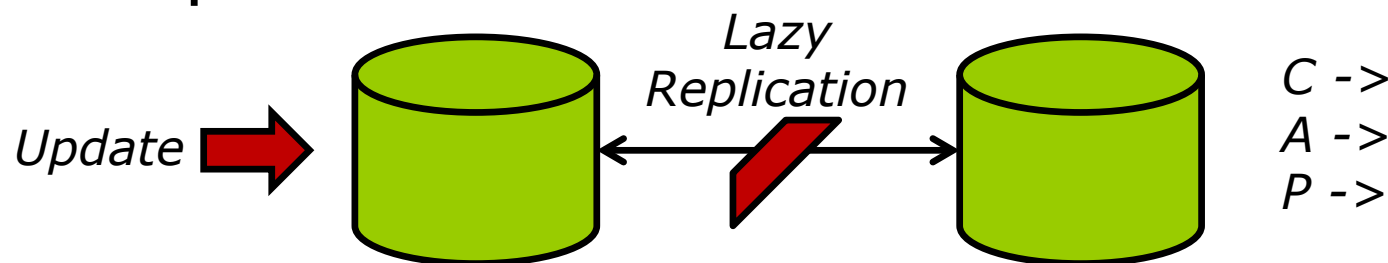
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:





# CAP Theorem

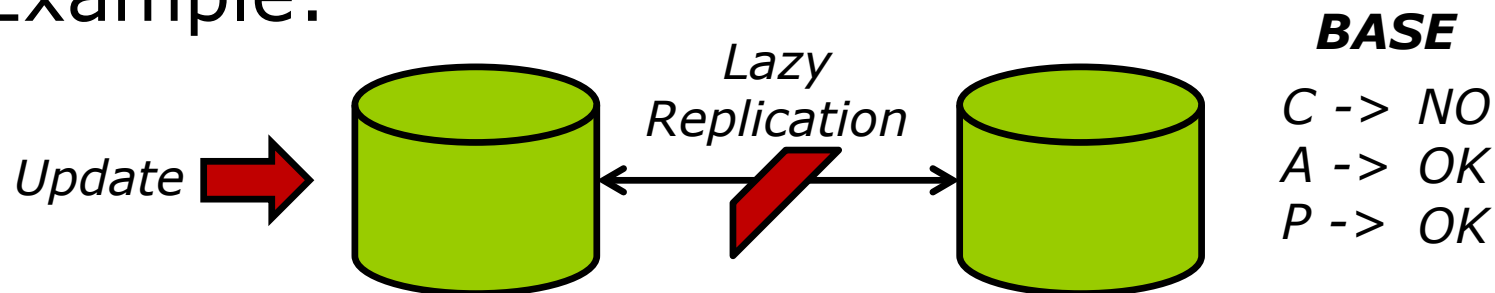
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



# CAP Theorem

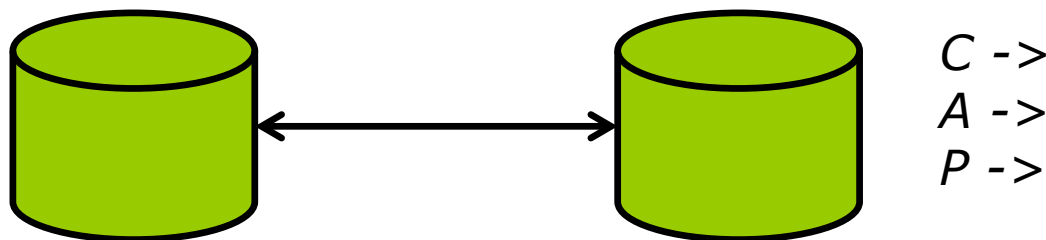
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).

## □ Example:



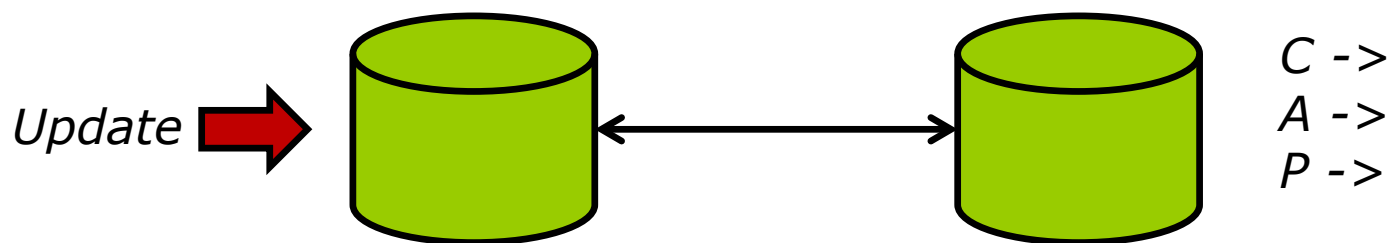
# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



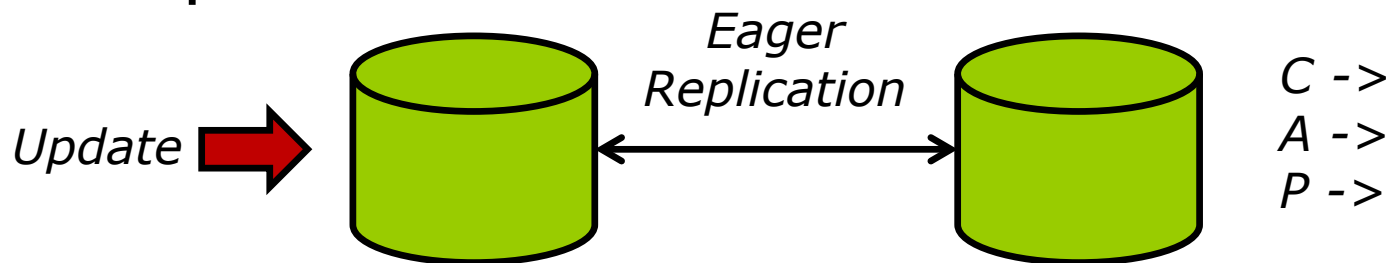
# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



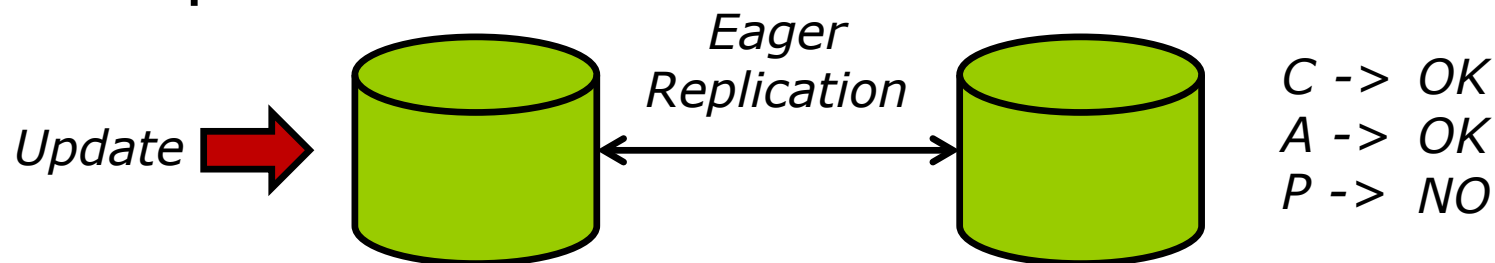
# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



# CAP Theorem

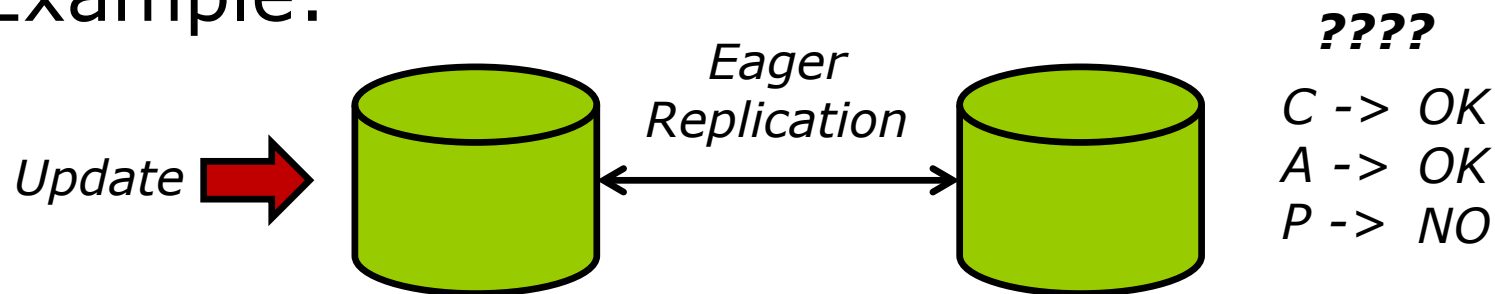
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



# CAP Theorem

- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties:*
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).

## □ Example:



# CAP Theorem Revisited

- ❑ The CAP theorem is not about choosing two out of the three **forever and ever**
  - Distributed systems are not always partitioned
- ❑ Without partitions: CA
- ❑ Otherwise...
  - Detect a partition
    - ❑ Normally by means of latency (time-bound connection)
  - Enter an explicit partition mode limiting some operations choosing either:
    - ❑ CP (i.e., ACID by means of e.g., 2PCP or PAXOS) or,
      - If a partition is detected, the operation is aborted
    - ❑ AP (i.e., BASE)
      - The operation goes on and we will tackle this next
  - If AP was chosen, enter a recovery process commonly known as *partition recovery* (e.g., compensate mistakes and get rid of inconsistencies introduced)
    - ❑ Achieve consistency: Roll-back to consistent state and apply ops in a deterministic way (e.g., using time-stamps)
      - Reduce complexity by only allowing certain operations (e.g., Google Docs)
      - Commutative operations (concatenate logs, sort and execute them)
    - ❑ Repair mistakes: Restore invariants violated
      - Last writer wins



# CAP Theorem: Examples (~DIY!)

- ❑ Yahoo's PNUTS (AP): Assumes data is partitioned according to where the user is
  - Remote copies are maintained asynchronously
  - The primary copy is local (to decrease latency)
- ❑ Facebook (AP): Unique primary copy
  - Primary copy is always in one location (higher latency)
  - After 20'' the user's traffic reverts to a closer copy (which hopefully, by that time, should reflect the change)
- ❑ HTML5 (AP): Full Availability
  - On-client persistent storage (primary copy)
  - Allows to go offline -> massive recovery process
- ❑ Google (CP): Considers primary partitions (data centers in USA, Germany, Sweden, etc.)
  - CA within partitions
  - CP between partitions using PAXOS (e.g., Megastore)

# Summary

---

## □ Concurrency

- Time-stamping
- Distributed Concurrency Control
  - Distributed Time-Stamping

## □ Recovery

- Main components
- Logging
  - Necessary Rules
  - Taxonomy of Crash Algorithms
    - Steal / No-steal, Force / No-force

## □ CAP Theorem

# Bibliography

---

- ❑ G. Gardarin and P. Valduriez. *Relational Databases and Knowledge bases*. Addison Wesley, 1989
- ❑ T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. 3rd Edition, Prentice Hall, 2011
- ❑ K. Ramamrithan and P. Crhysanthhis. *Advances in Concurrency Control and Transaction Processing*. IEEE Executive Briefing. IEEE, 1997
- ❑ P. Bernstein, et. al. *Concurrency Control in Database Systems*. Addison Wesley, 1987
- ❑ R. Ramakrishnan and J. Gehrke. *Database Management Systems*. 3rd Edition, McGraw-Hill, 2003
- ❑ L. Liu and M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009