Concurreny and Recovery

Managing the ACID properties in distributed systems

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

ACID Properties

- *Atomicity*
- *Consistency*
- *Isolation*
- *Durability*

Concurrency System

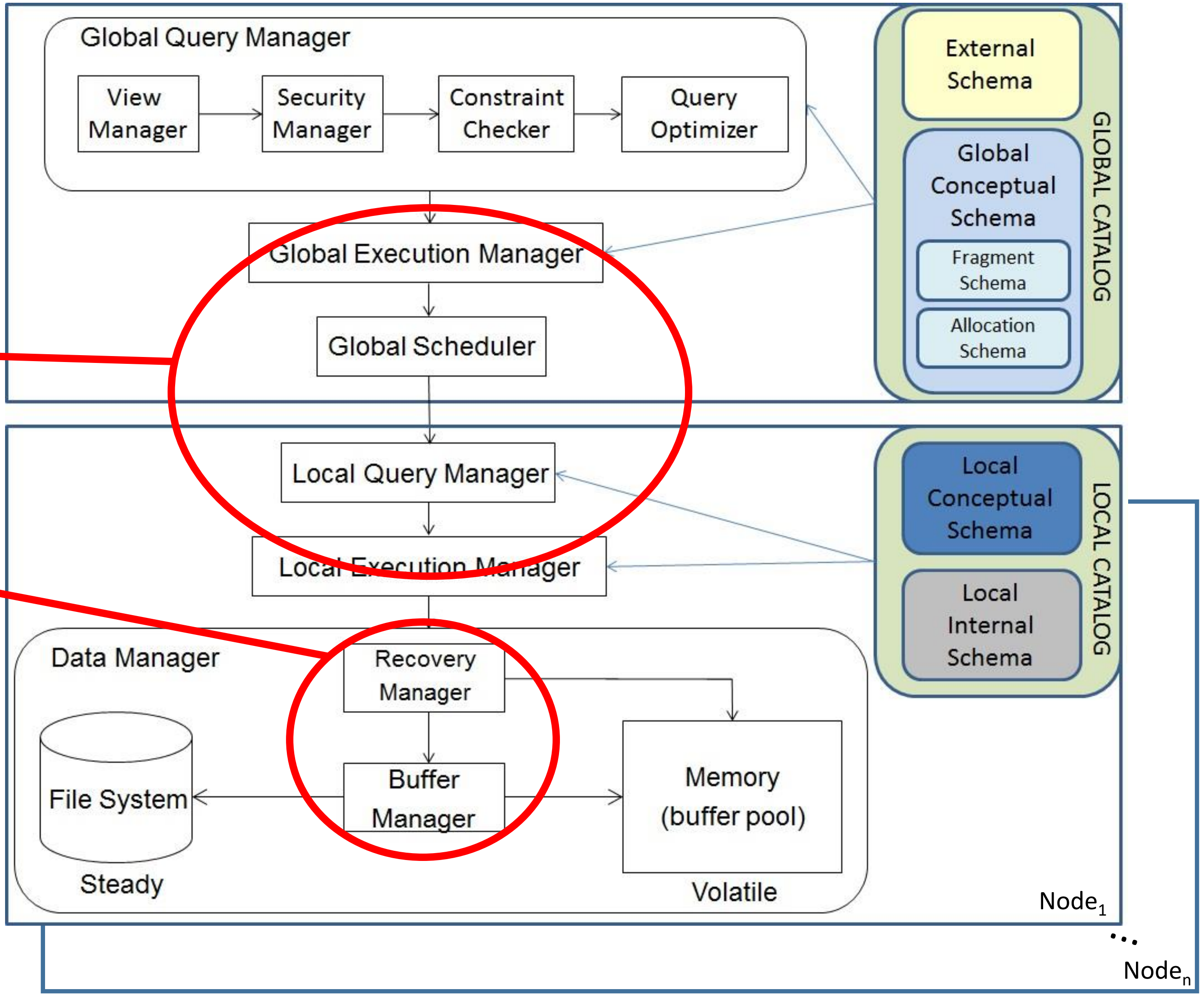
Recovery System

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

Distributed DBMS Architecture (recap)

Concurrency manager and scheduler: guarantees CI (from ACID) and schedules the execution of subqueries over the distributed fragments. They are responsible for briding between the global layer and each node

Recovery manager: interacts with the buffer and memory to guarantee AD (from ACID)



Challenge IV: Distributed Tx Management

- ACID properties are not necessary in all scenarios
  - There cases where all 4 properties can be relaxed
- Relaxing Atomicity and Durability
  - Entails data loss
  - Save synchronization time
- Relaxing Consistency and Isolation
  - Generate interferences
  - Save locks and contention

Basics on Concurrency

Recall of basic concepts

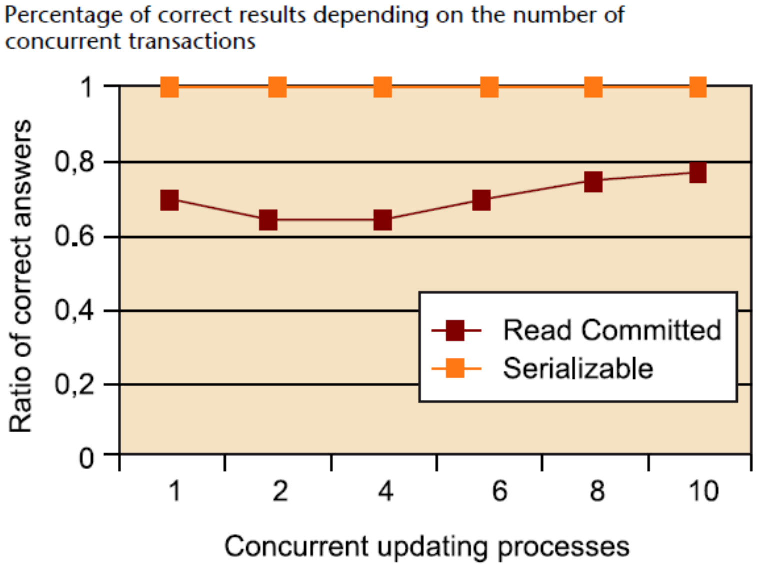
Activity: Basics on Concurrency

- 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

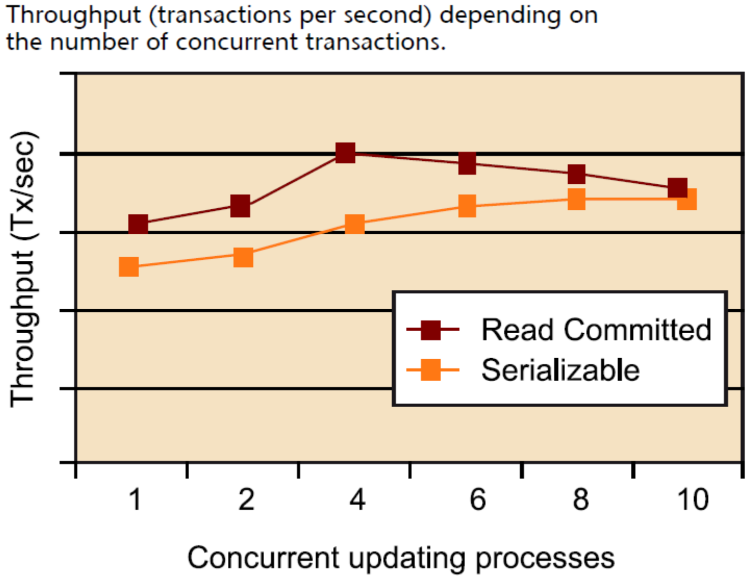
- **Read Uncommitted**: Avoids Lost Update (or Write-Write) interferences, which appear when data written by a Tx is lost, because another one overwrites it before the first Tx commits.
- **Read Committed**: Avoids Read Uncommitted (or Write-Read) interferences, which typically appear when a transaction 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)



Performance
(System throughput)



TIME-STAMPING

Concurrency and Recovery

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 Beginning of Transactions (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 T
  - Timestamp of the BoT (Begin of Transaction)
    - TS(T)
- For each granule G
  - 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)≤ TS(T) then
    TSR(G)= max(TSR(G), TS(T));
    R(G);
   else
    abort(T);
   endif
endProcedure
```

Intuition: if the last transation writting on G did so before T started, no problem

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

Intuition: if the last transation writting or reading G did so before T started, no problem

Example of Time-Stamping (I)

	T_1	T_2
1		
2		
3		
4		
5		
6		

$\boxed{TSR(A)=0}$

TSW(A)=0

Example of Time-Stamping (I)

	T_1	T_2
1	BoT	
2		
3		
4		
5		
6		

$\boxed{TSR(A)=0}$

TSW(A)=0

Example of Time-Stamping (I)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		
3		
4		
5		
6		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (I)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		BoT
3		
4		
5		
6		

$TSR(A)=0$

$TSW(A)=0$

Example of Time-Stamping (I)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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		

$\boxed{TSR(A)=0}$

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		

$\boxed{TSR(A)=1}$

$TSW(A)=0$

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		

$\boxed{TSR(A)=2}$

$TSW(A)=0$

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		

$\boxed{TSR(A)=2}$

$TSW(A)=0$

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)	

$\boxed{TSR(A)=2}$

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)

$\boxed{TSR(A)=2}$

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)	

$\downarrow$

Abort(T_1)

$\downarrow$

Restart(T_1)

$\boxed{TSR(A)=2}$

TSW(A)=2

Repeatable read interference avoided!

Example of Time-Stamping (II)

	T_1	T_2
1		
2		
3		
4		
5		

$\boxed{TSR(A)=0}$

TSW(A)=0

Example of Time-Stamping (II)

	T_1	T_2
1	BoT	
2		
3		
4		
5		

$\boxed{TSR(A)=0}$

TSW(A)=0

Example of Time-Stamping (II)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		
3		
4		
5		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (II)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		BoT
3		
4		
5		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (II)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (II)

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

$\boxed{TSR(A)=1}$

$TSW(A)=0$

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		

$\boxed{TSR(A)=1}$

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		

$\boxed{TSR(A)=2}$

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)	

$\boxed{TSR(A)=2}$

$TSW(A)=0$

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)	

$\boxed{TSR(A)=2}$

$TSW(A)=0$

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)	

$\downarrow$

Abort(T_1)

$\downarrow$

Restart(T_1)

$\boxed{TSR(A)=2}$

$TSW(A)=0$

Note an interference still did not happen but the system aborts T_1 to avoid problems (i.e., pessimistic technique)

Example of Time-Stamping (III)

	T_1	T_2
1		
2		
3		
4		
5		
6		

TSR(A)=0

TSW(A)=0

Example of Time-Stamping (III)

	T_1	T_2
1	BoT	
2		
3		
4		
5		
6		

TSR(A)=0

TSW(A)=0

Example of Time-Stamping (III)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		
3		
4		
5		
6		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (III)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		BoT
3		
4		
5		
6		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (III)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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		

TSR(A)=2

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		

$\boxed{TSR(A)=2}$

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		

$\boxed{TSR(A)=2}$

$TSW(A)=2$

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		

TSR(A)=2

TSW(A)=2

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	↓ Abort(T_1) ↓ Restart(T_1)	

$\boxed{TSR(A)=2}$

TSW(A)=2

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	↓	
	Abort(T_1)	
	↓	
	Restart(T_1)	

$\boxed{TSR(A)=2}$

$TSW(A)=2$

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	↓	
	Abort(T_1)	
	↓	
	Restart(T_1)	

$\boxed{TSR(A)=2}$

TSW(A)=2

Note an interference still did not happen but the system aborts T_1 to avoid problems (i.e., pessimistic technique)

Example of Time-Stamping (IV)

	T_1	T_2
1		
2		
3		
4		
5		
6		
7		

$\boxed{TSR(A)=0}$

TSW(A)=0

Example of Time-Stamping (IV)

	T_1	T_2
1	BoT	
2		
3		
4		
5		
6		
7		

$\boxed{TSR(A)=0}$

TSW(A)=0

Example of Time-Stamping (IV)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		
3		
4		
5		
6		
7		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (IV)

$TS(T_1)=1$

	T_1	T_2
1	BoT	
2		BoT
3		
4		
5		
6		
7		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

Example of Time-Stamping (IV)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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		
6		
7		

$\boxed{TSR(A)=0}$

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		
6		
7		

$\boxed{TSR(A)=1}$

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		
6		
7		

$\boxed{TSR(A)=1}$

$TSW(A)=0$

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	W(A)	
5		
6		
7		

$\boxed{TSR(A)=1}$

$TSW(A)=1$

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)
6		
7		

$\boxed{TSR(A)=1}$

TSW(A)=1

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	W(A)	
5		R(A)
6		
7		

$\boxed{\text{TSR(A)=2}}$

TSW(A)=1

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)
6		Commit
7		

$\boxed{TSR(A)=2}$

TSW(A)=1

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)
6		Commit
7	Rollback(T_1)	

$\boxed{TSR(A)=2}$

TSW(A)=1

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)
6		Commit
7	Rollback(T_1)	

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		R(A)
6		Commit
7	Rollback(T_1)	
		Restart(T_2)

$\boxed{TSR(A)=2}$

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		R(A)
6		Commit
7	Rollback(T_1)	

$\downarrow$?

Restart(T_2)

$\boxed{TSR(A)=2}$

$TSW(A)=0$

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		R(A)
6		Commit
7	Rollback(T_1)	
		Restart(T_2)

$\boxed{TSR(A)=2}$

$TSW(A)=0$

Time-stamping as-it-is, does not prevent the read committed interference!

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

<u>procedure</u> read(T_i, G) <u>is</u>
    <u>if</u> TSW(G)≤ TS(T_i) **<u>and</u>** **$T_{W(G)}$ comitted** <u>then</u>
      …
<u>endProcedure</u>

<u>procedure</u> write(T_i ,G) <u>is</u>
    <u>if</u> TSW(G) ≤ TS(T_i) <u>and</u> TSR(G) ≤ TS(T_i) **<u>and</u>** **$T_{W(G)}$ comitted** <u>then</u>
    …
<u>endProcedure</u>

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

<u>procedure</u> read(T_i, G) <u>is</u>
    <u>if</u> $TSW(G) \leq TS(T_i)$ **<u>and</u>** **$T_{W(G)}$ comitted** <u>then</u>
      …
<u>endProcedure</u>

Additional logic to be executed for every single read and write !!!

<u>procedure</u> write(T_i ,G) <u>is</u>
    <u>if</u> $TSW(G) \leq TS(T_i)$ <u>and</u> $TSR(G) \leq TS(T_i)$ **<u>and</u>** **$T_{W(G)}$ comitted** <u>then</u>
    …
<u>endProcedure</u>

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

<u>procedure</u> read(T_i, G) <u>is</u>
  <u>if</u> TSW(G)≤ TS(T_i) **<u>and</u>** **$T_{W(G)}$ comitted** <u>then</u>
    …
<u>endProcedure</u>

Additional logic to be executed for every single read and write !!!

<u>procedure</u> write(T_i ,G) <u>is</u>
  <u>if</u> TSW(G) ≤ TS(T_i) <u>and</u> TSR(G) ≤ TS(T_i) **<u>and</u>** **$T_{W(G)}$ comitted** <u>then</u>
    …
<u>endProcedure</u>

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)
6		
7		

TSR(A)=2

TSW(A)=1

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)
6		Commit
7		

TSR(A)=2

TSW(A)=1

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)
6		Commit
7	Rollback(T_1)	

$\boxed{TSR(A)=2}$

TSW(A)=1

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)
6		Commit
7	Rollback(T_1)	

$\boxed{TSR(A)=2}$

$TSW(A)=0$

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		R(A)
6		Commit
7	Rollback(T_1)	↓
		Abort(T_2)
		↓
		Restart(T_2)

$\boxed{TSR(A)=2}$

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		R(A)
6		Commit
7	Rollback(T_1)	↓
		Abort(T_2)
		↓
		Restart(T_2)

$\boxed{TSR(A)=2}$

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		R(A)
6		Commit
7	Rollback(T_1)	↓
		Abort(T_2)
		↓
		Restart(T_2)

$\boxed{TSR(A)=2}$

TSW(A)=0

Read committed interference avoided!

Dynamic Time-Stamping

- Dynamic time-stamping smooths the impact of <u>unnecessary aborts</u>
- It delays the assignment of the timestamp to the transactions as much as possible (i.e., **until there is a potential conflict between the current transaction and another one**)
  - Injects this code in the *read* and *write* procedures

<u>foreach</u> $T_i \in$ setOfActiveTx <u>do</u>
    <u>if</u> ($G \in RS(T_i) \cap WS(T)$) <u>or</u> ($G \in WS(T_i) \cap RS(T)$) <u>then</u>
        <u>if</u> $TS(T_i)$==null <u>and</u> $TS(T)$==null <u>then</u>
            assign both timestamps so that $TS(T_i)<TS(T)$
        <u>elsif</u> $TS(T_i)$==null <u>or</u> $TS(T)$==null <u>then</u>
            assign one timestamp so that $TS(T_i)<TS(T)$
        <u>endif</u>
    <u>endif</u>
<u>endforeach</u>

Example of Time-Stamping (V)

	T_1	T_2
1		
2		
3		
4		
5		
6		
7		

$\boxed{TSR(A)=0}$

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		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{\}$

$WS(T_2)=\{\}$

Example of Time-Stamping (V)

	T_1	T_2
1	BoT	
2		BoT
3		
4		
5		
6		
7		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

$RS(T_2)=\{\}$

$WS(T_2)=\{\}$

Example of Time-Stamping (V)

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

$\boxed{TSR(A)=0}$

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		

$\boxed{TSR(A)=0}$

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		

$\boxed{TSR(A)=0}$

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		

$\boxed{TSR(A)=0}$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

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

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

Example of Time-Stamping (V)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

$RS(T_1)=\{\}$

$WS(T_1)=\{\}$

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

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

Example of Time-Stamping (V)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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

$WS(T_1)=\{\}$

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

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

Example of Time-Stamping (V)

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

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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

$WS(T_1)=\{\}$

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

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

Potential conflict! We cannot delay anymore assigning TS to these two Txs

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		

Potential conflict!
We cannot delay anymore assigning TS to these two Txs

$\boxed{TSR(A)=0}$

$TSW(A)=0$

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

$WS(T_1)=\{\}$

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

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

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		

$\boxed{TSR(A)=2}$

$TSW(A)=1$

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

$WS(T_1)=\{\}$

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

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

Example of Time-Stamping (V)

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

$\boxed{TSR(A)=2}$

$TSW(A)=1$

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

$WS(T_1)=\{\}$

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

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

Example of Time-Stamping (V)

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

$\boxed{TSR(A)=2}$

$TSW(A)=1$

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

$WS(T_1)=\{\}$

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

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

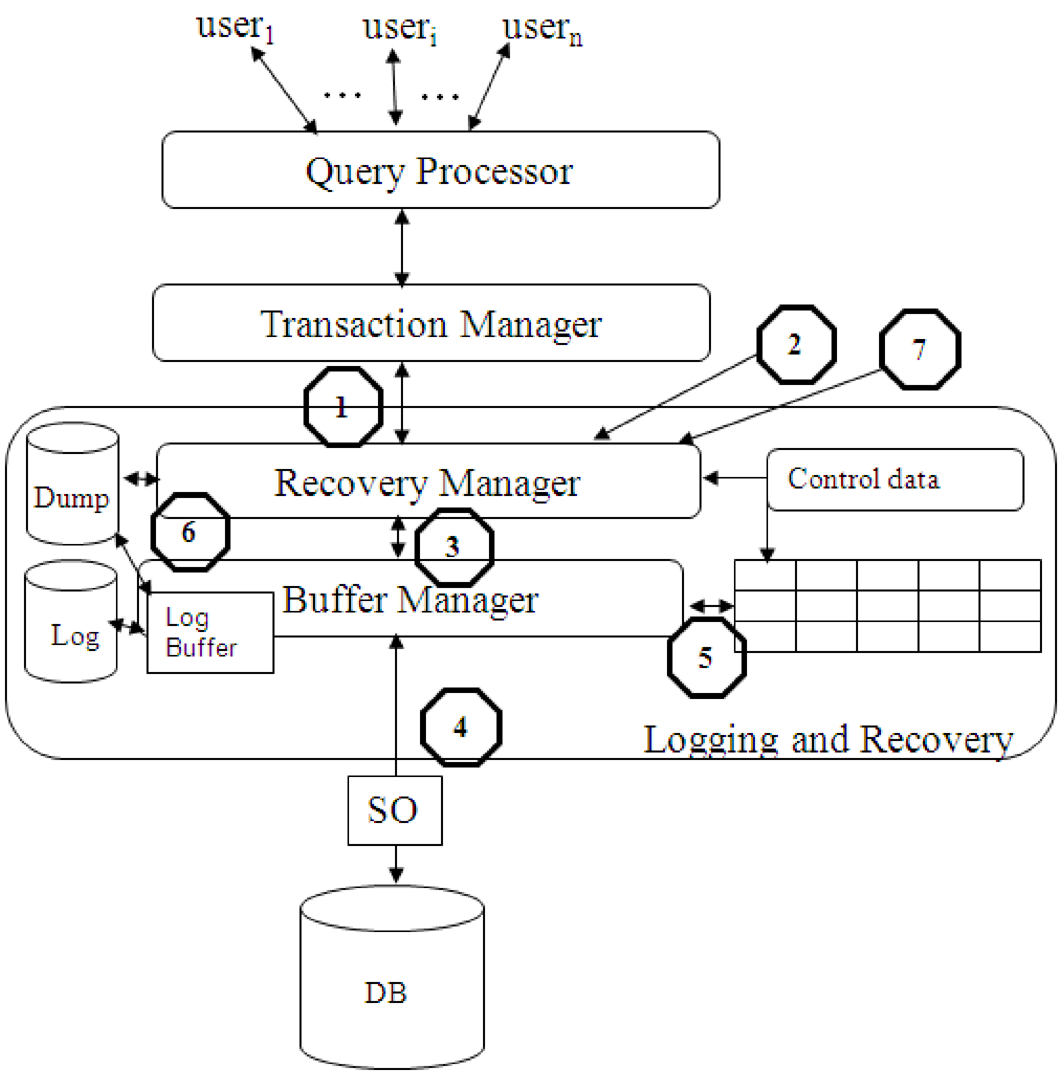
Basics on Recovery

Recall Basic Concepts

Recovery System

- Failures must be masked to the users of transaction-based systems
  - Transaction failure
    - Voluntary rollbacks
    - Aborted by the DBMS
      - Deadlocks, interferences detected by time-stamp, etc.
  - 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, the following rules must apply:
  - 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

CAP Theorem

On the Need of New Architectures

New Architectures to the Rescue

- To build more efficient recovery systems, NOSQL systems rethink the DBMS components and how the **transaction manager** interacts with the **recovery manager** and this one with the **buffer manager**

New Architectures to the Rescue

- To build more efficient recovery systems, NOSQL systems rethink the DBMS components and how the **transaction manager** interacts with the **recovery manager** and this one with the **buffer manager**
  - The transaction manager sometimes is completely removed (read-only systems). Most of the times, it is implemented by using dynamic-timestamping or an optimistic strategy

New Architectures to the Rescue

- To build more efficient recovery systems, NOSQL systems rethink the DBMS components and how the **transaction manager** interacts with the **recovery manager** and this one with the **buffer manager**
  - The transaction manager sometimes is completely removed (read-only systems). Most of the times, it is implemented by using dynamic-timestamping or an optimistic strategy
  - The recovery and buffer managers, in most NOSQL, does not resemble the traditional approach and it mostly relies on the concept of replicas
    - We will see specific architectures during the course and how they manage it

New Architectures to the Rescue

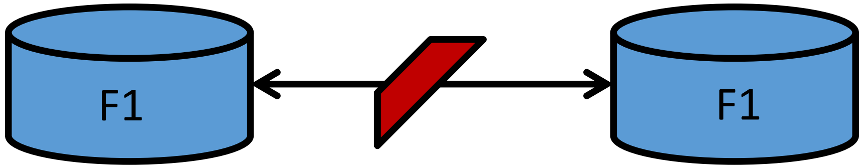
- To build more efficient recovery systems, NOSQL systems rethink the DBMS components and how the **transaction manager** interacts with the **recovery manager** and this one with the **buffer manager**
  - The transaction manager sometimes is completely removed (read-only systems). Most of the times, it is implemented by using dynamic-timestamping or an optimistic strategy
  - The recovery and buffer managers, in most NOSQL, does not resemble the traditional approach and it mostly relies on the concept of replicas
    - We will see specific architectures during the course and how they manage it
  - Logs are still used in distributed systems, but for specific components. In such cases, it is a regular log acting locally (not globally):
    - In **the primary servers** storing the catalog
    - In the **secondary servers storing the primary replica** when primary versioning is activated

Limitations of a Data-Sharing Distributed System

- Nevertheless, there is a fundamental theoretical limitation that affect **any** distributed system
- <u>CAP theorem formulation</u>: *Any networked shared-data system can have at most two of the three following 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).

CAP Theorem

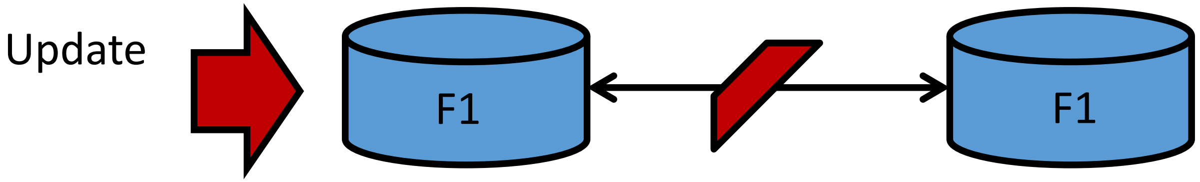
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

CAP Theorem

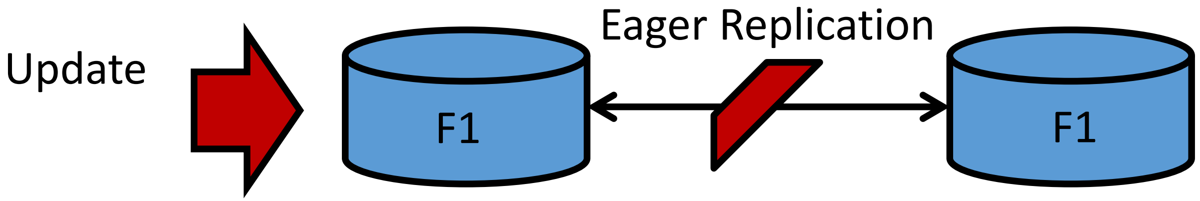
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

CAP Theorem

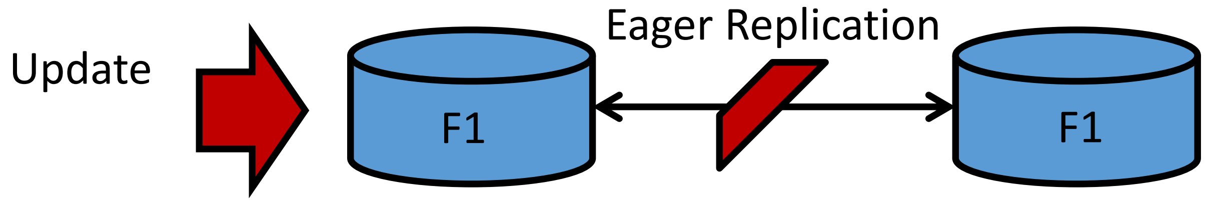
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

CAP Theorem

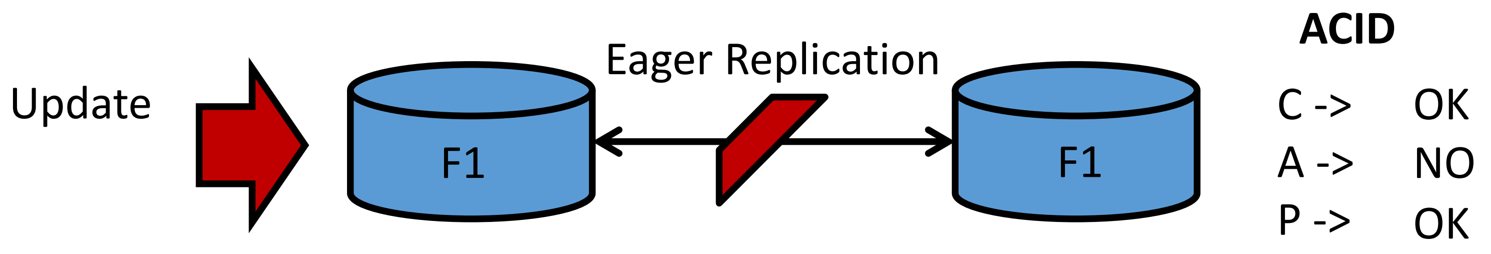
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



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

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:

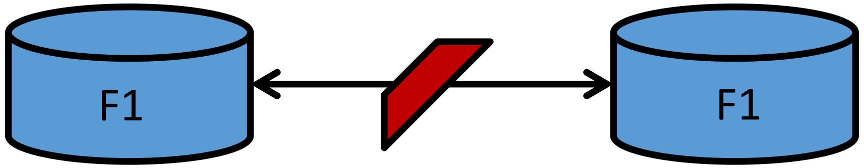


Strong Consistency

- 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<=N
    - Both sets may not overlap
  - Potential conflict $\Rightarrow$ W<(N+1)/2
- Strong consistency is only guaranteed if W+R>N

CAP Theorem

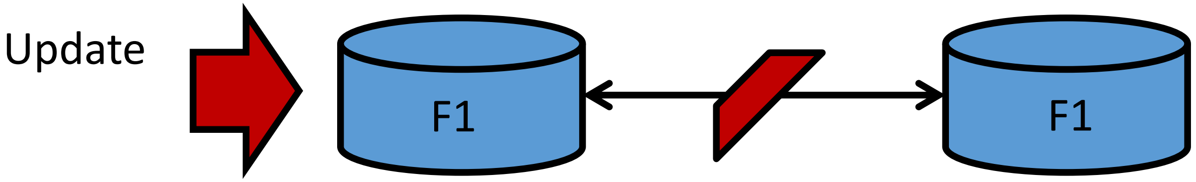
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

CAP Theorem

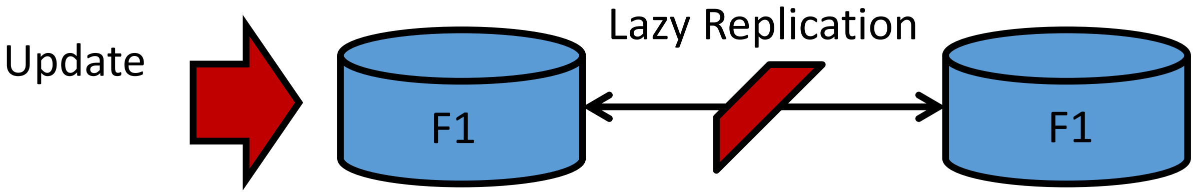
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

CAP Theorem

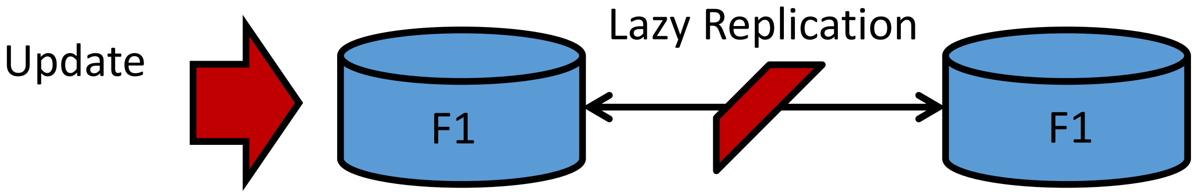
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

CAP Theorem

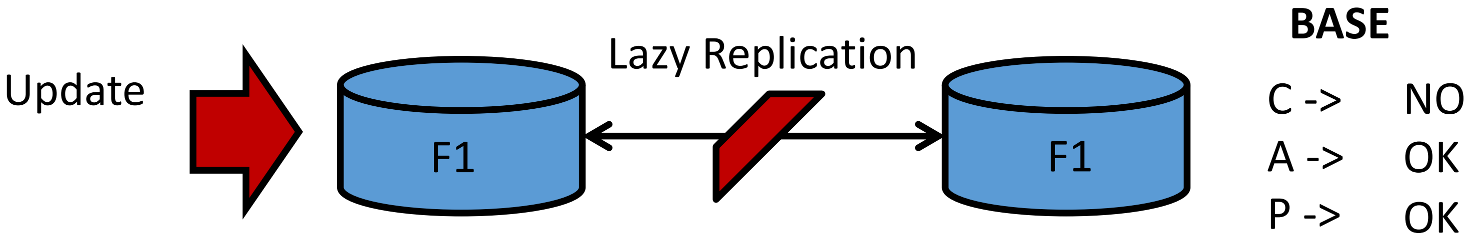
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



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

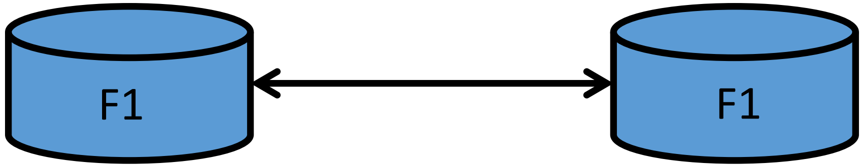
CAP Theorem

- 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

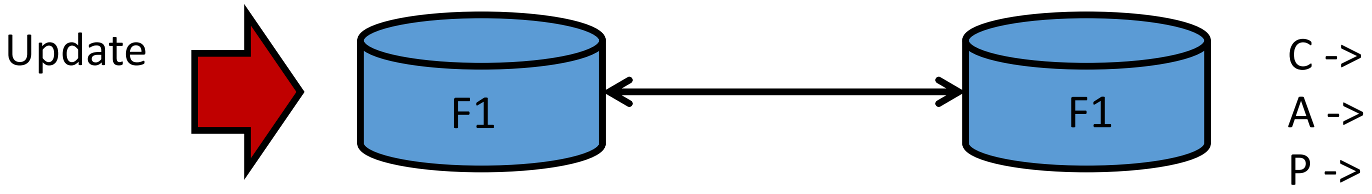
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



C ->
A ->
P ->

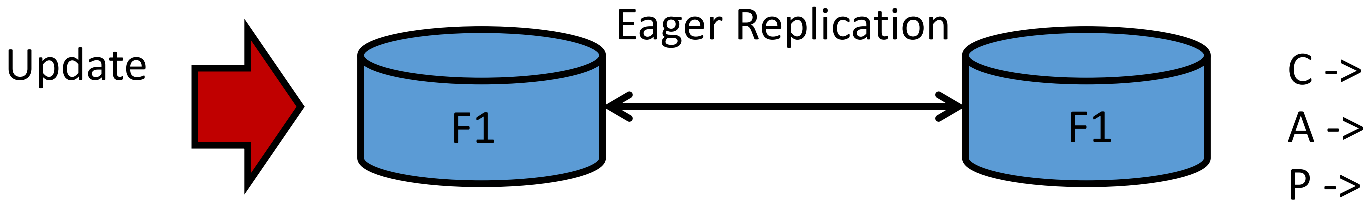
CAP Theorem

- 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:



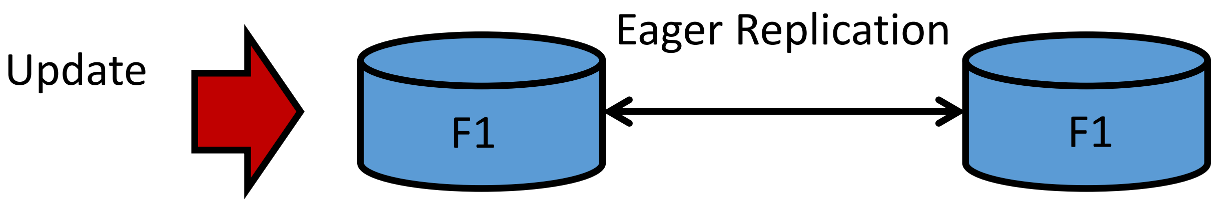
CAP Theorem

- 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

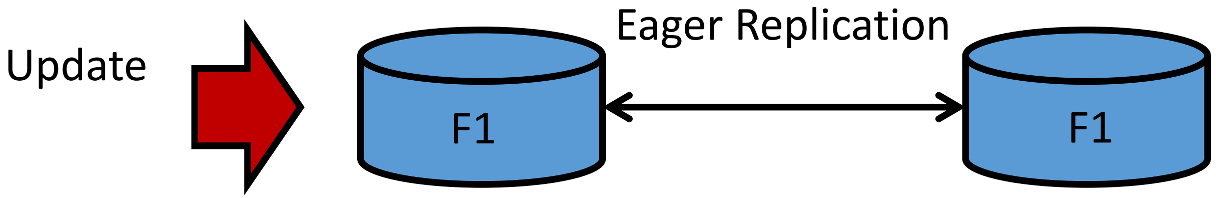
- Eric Brewer. CAP Theorem: *any networked shared-data system can have at most two of three desirable properties*:
  - consistency (C) equivalent to having a single up-to-date copy of the data;
  - high availability (A) of that data (for updates); and
  - tolerance to network partitions (P).
- Example:



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

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:



????

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

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

Summary

- Concurrency
  - Time-stamping
  - Distributed Concurrency Control
    - Distributed Time-Stamping
- Recovery
  - Main components
  - Logging
- CAP Theorem
  - Limitations of a data-sharing distributed system

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. Crhysanthis. 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