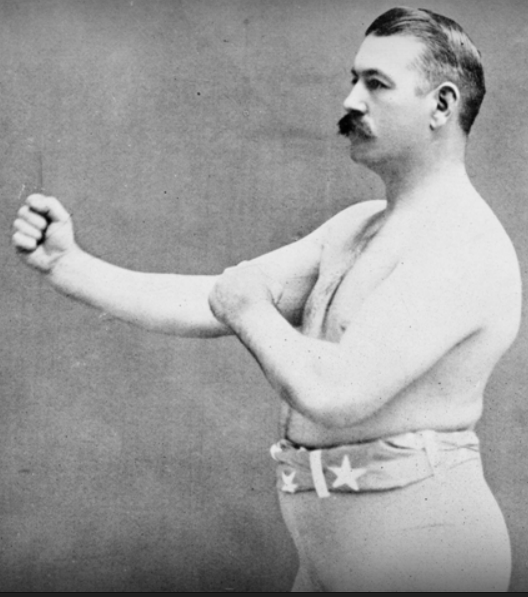
15-721
DATABASE SYSTEMS



Lecture #05 – Concurrency Control Part III

Andy Pavlo // Carnegie Mellon University // Spring 2016

TODAY'S AGENDA

Stored Procedures
Optimistic Concurrency Control
Course Projects

OBSERVATION

Disk stalls are (almost) gone when executing txns in an in-memory DBMS.

There are still other stalls when an app uses **conversational** API to execute queries on DBMS
→ ODBC/JDBC
→ DBMS-specific wire protocols

CONVERSATIONAL DATABASE API

Application

```
BEGIN
  SQL
  Program Logic
  SQL
  Program Logic
  ⋮
COMMIT
```

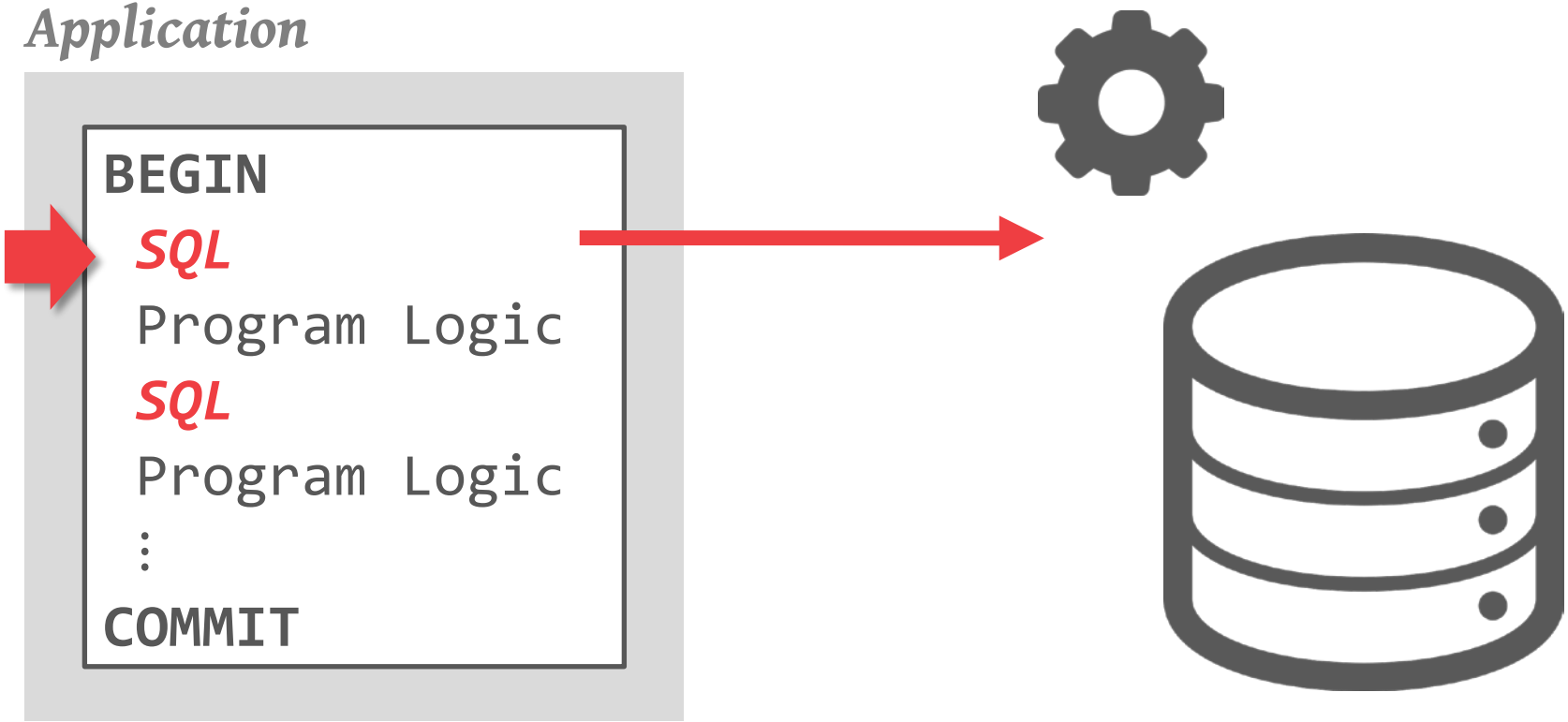
CONVERSATIONAL DATABASE API

Application

```
BEGIN
  SQL
  Program Logic
  SQL
  Program Logic
  ⋮
COMMIT
```

CONVERSATIONAL DATABASE API

Application



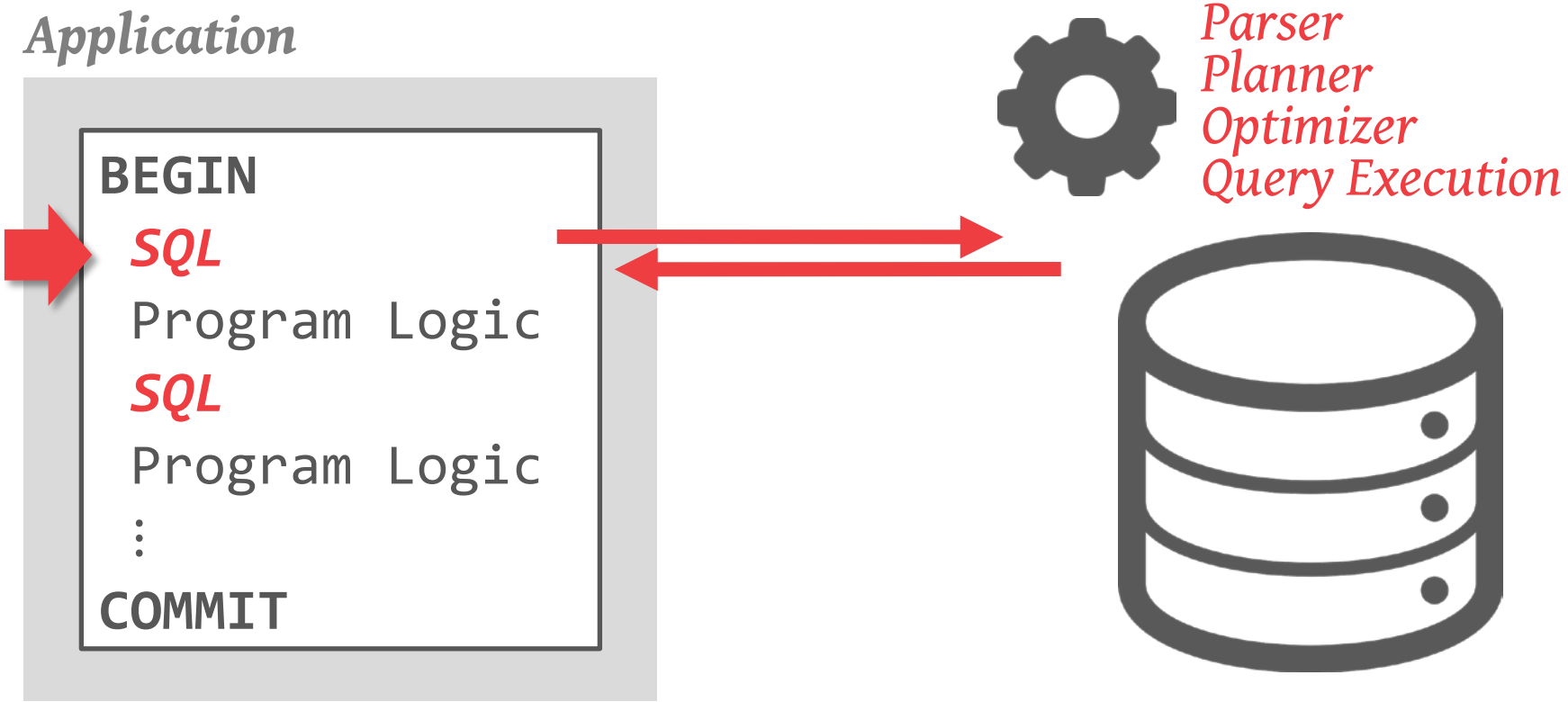
CONVERSATIONAL DATABASE API

Application

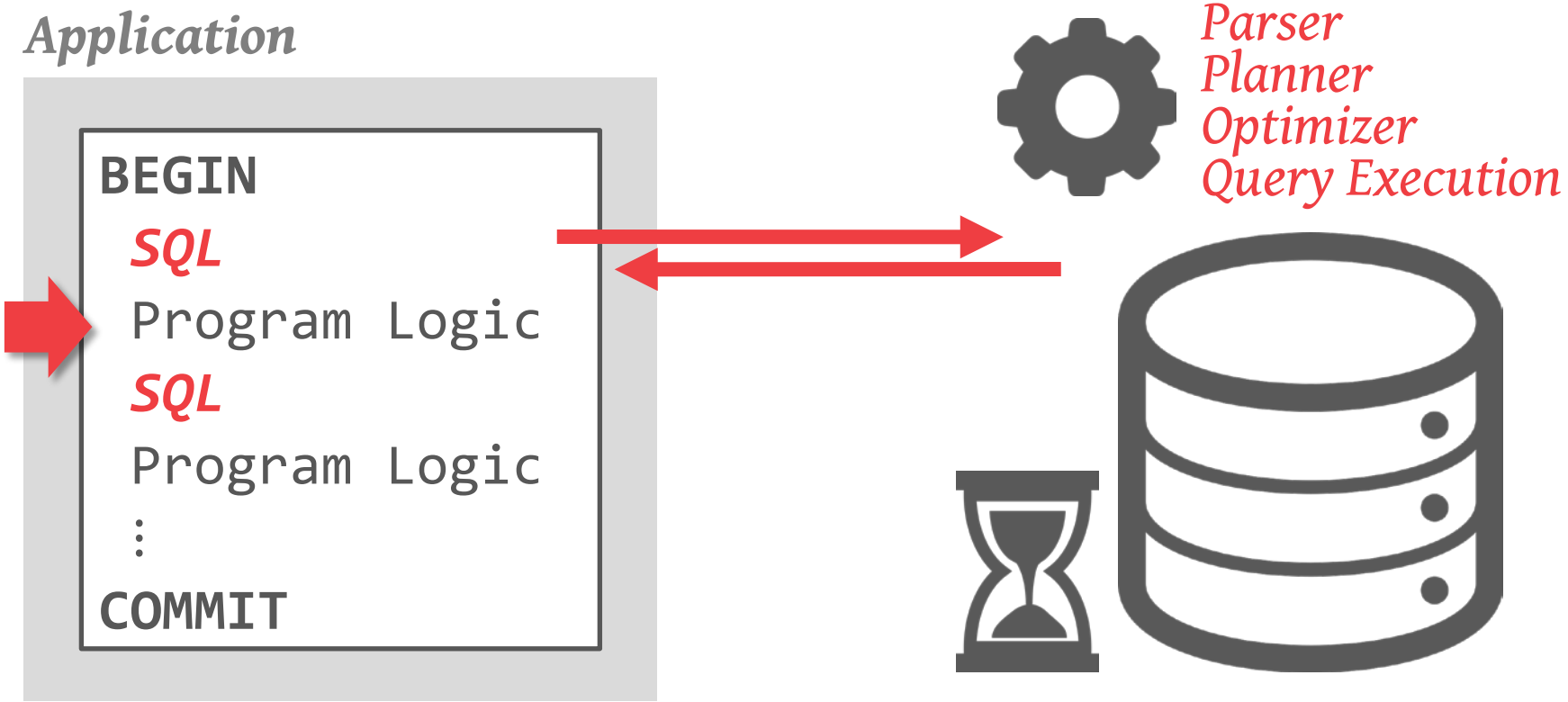
```
BEGIN
  SQL
  Program Logic
  SQL
  Program Logic
  ⋮
COMMIT
```

Parser
Planner
Optimizer
Query Execution

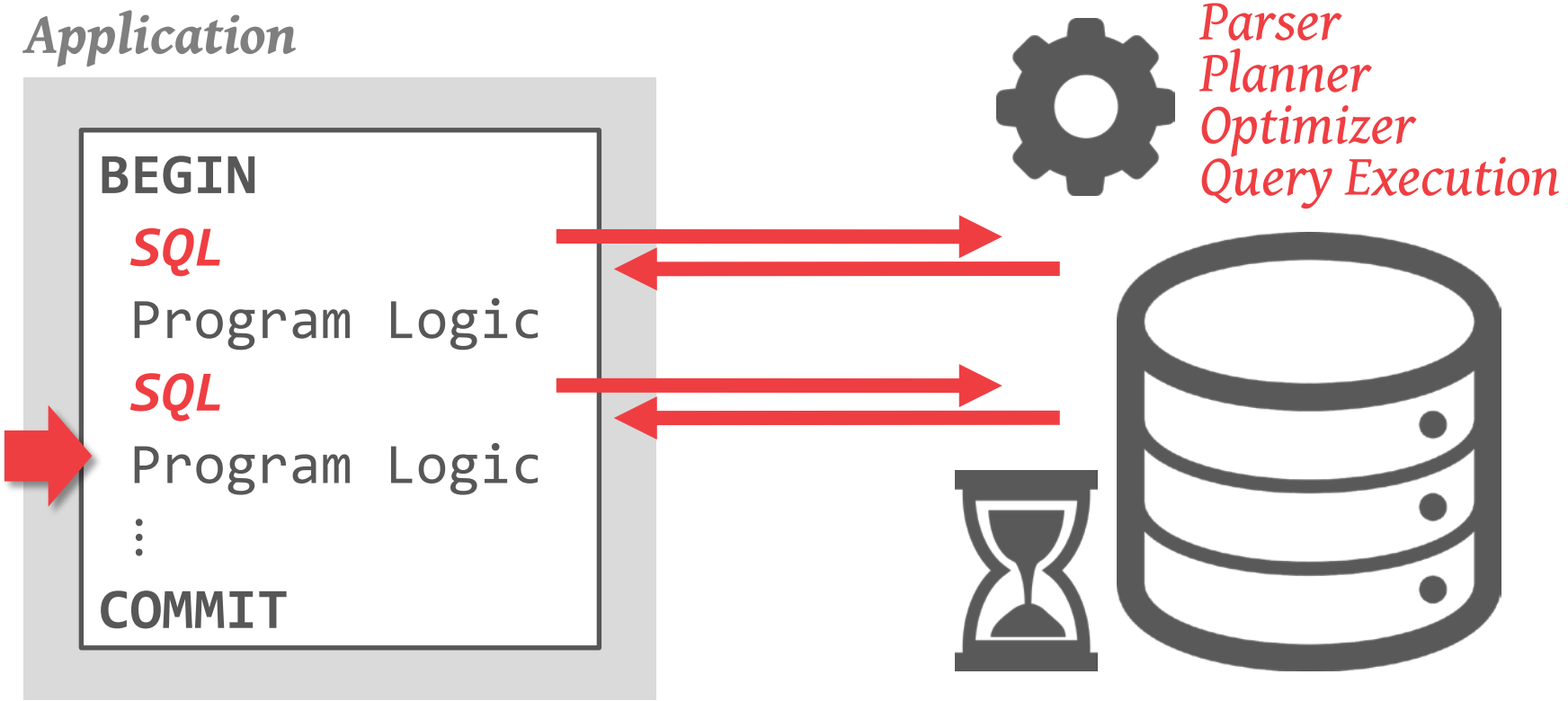
CONVERSATIONAL DATABASE API



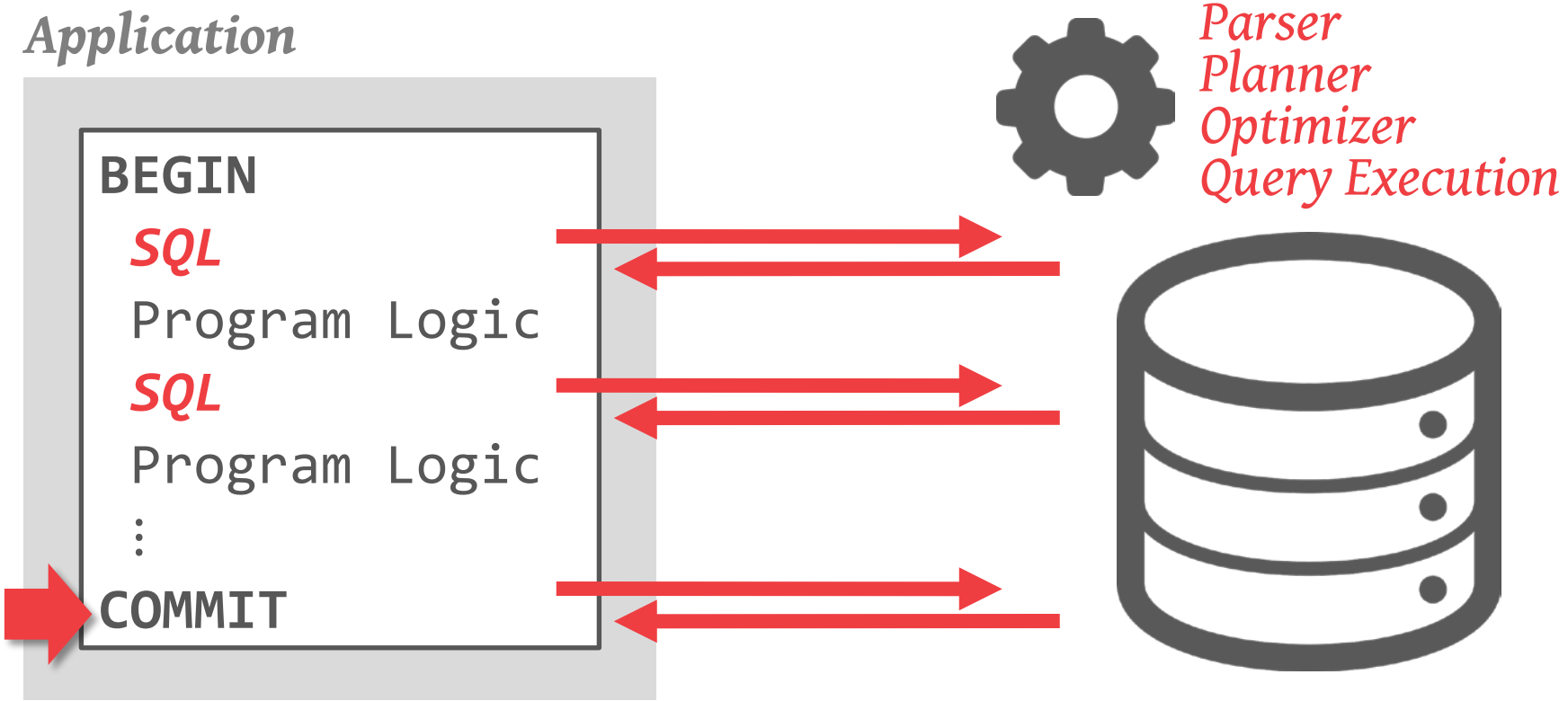
CONVERSATIONAL DATABASE API



CONVERSATIONAL DATABASE API



CONVERSATIONAL DATABASE API



SOLUTIONS

Prepared Statements
→ Removes query preparation overhead.

Query Batches
→ Reduces the number of network roundtrips.

Stored Procedures
→ Removes both preparation and network stalls.

STORED PROCEDURES

A **stored procedure** is a group of queries that form a logical unit and perform a particular task on behalf of an application directly inside of the DBMS.

Programming languages:
→ **SQL/PSM** (standard)
→ **PL/SQL** (Oracle / IBM / MySQL)
→ **Transact-SQL** (Microsoft)

STORED PROCEDURES

Application

```
BEGIN
  SQL
  Program Logic
  SQL
  Program Logic
  ⋮
COMMIT
```

STORED PROCEDURES

Application

PROC(x)

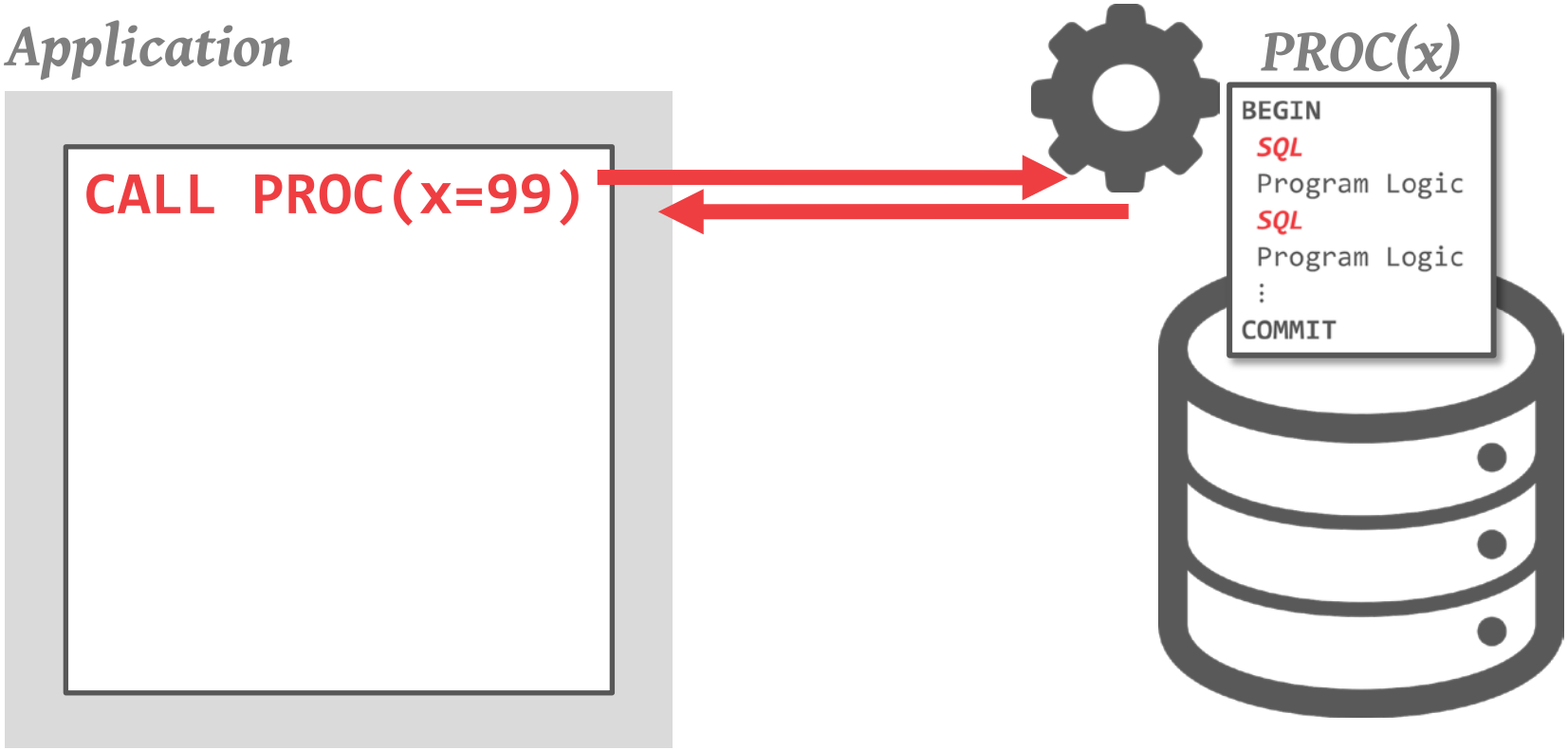
STORED PROCEDURES

Application

```
CALL PROC(x=99)
```

PROC(x)

STORED PROCEDURES



STORED PROCEDURE EXAMPLE

```
CREATE PROCEDURE testProc
  (num INT, name VARCHAR) RETURNS INT
  BEGIN
    DECLARE cnt INT DEFAULT 0;
    LOOP
      INSERT INTO student VALUES (cnt, name);
      SET cnt := cnt + 1;
      IF (cnt > 15) THEN
        RETURN cnt;
      END IF;
    END LOOP;
  END;
```

ADVANTAGES

Reduce the number of round trips between application and database servers.

Increased performance because queris are pre-compiled and stored in DBMS.

Procedure reuse across applications.

DISADVANTAGES

Not as many developers know how to write SQL/PSM code.
→ Safe Languages vs. Sandbox Languages

Outside the scope of the application so it is difficult to manage versions and hard to debug.

Probably not be portable to other DBMSs.

DISADVANTAGES

Not as many developers know how to write SQL/PSM code.

→ Safe Languages vs. Sandbox Languages

Outside the scope of the application so it is difficult to manage versions and hard to debug.

Probably not be portable to other DBMSs.

OPTIMISTIC CONCURRENCY CONTROL

Timestamp-ordering scheme where txns copy data read/write into a private workspace that is not visible to other active txns.

When a txn commits, the DBMS verifies that there are no conflicts.

First proposed in 1981 at CMU by H.T. Kung.

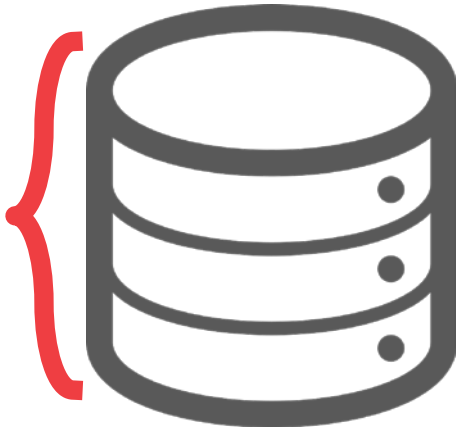
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

OPTIMISTIC CONCURRENCY CONTROL

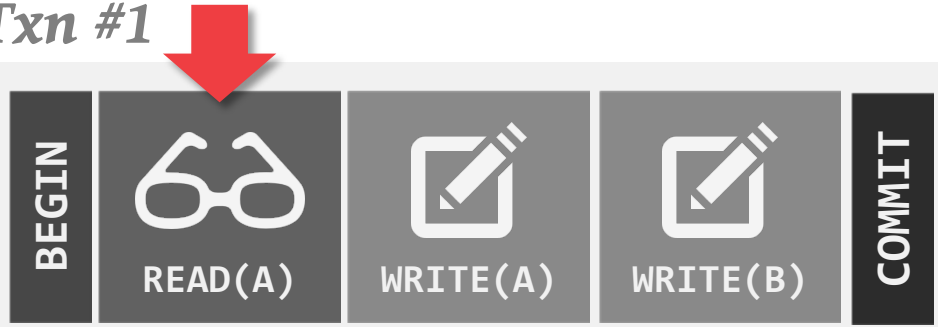
Txn #1

Record	Value	Write Timestamp
A	123	10000
B	456	10000

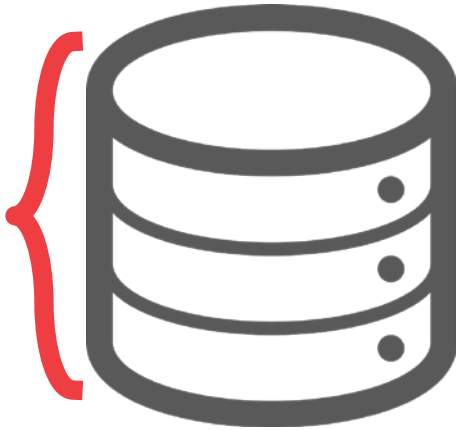


OPTIMISTIC CONCURRENCY CONTROL

Txn #1



Record	Value	Write Timestamp
A	123	10000
B	456	10000

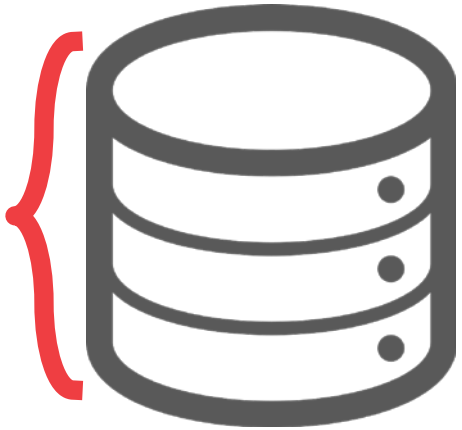


OPTIMISTIC CONCURRENCY CONTROL

Txn #1

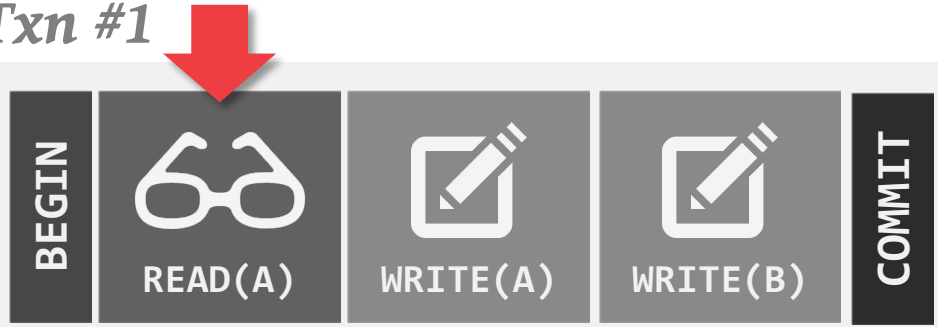
Read Phase

Record	Value	Write Timestamp
A	123	10000
B	456	10000

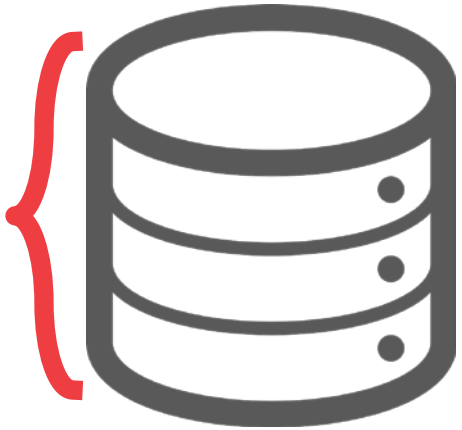


OPTIMISTIC CONCURRENCY CONTROL

Txn #1



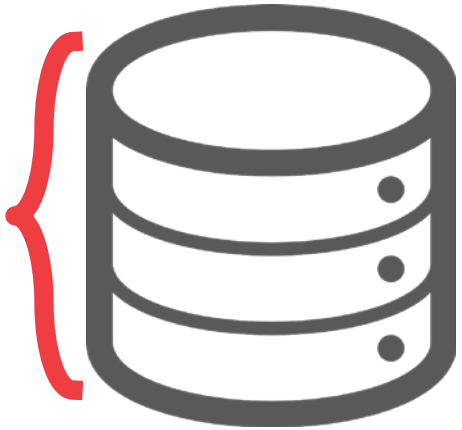
Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Record	Value	Write Timestamp
A	123	10000
B	456	10000



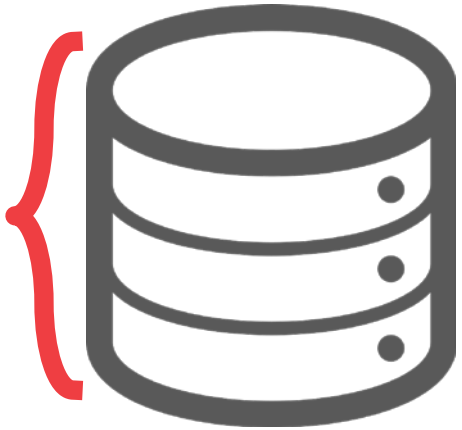
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Workspace

Record	Value	Write Timestamp

Record	Value	Write Timestamp
A	123	10000
B	456	10000



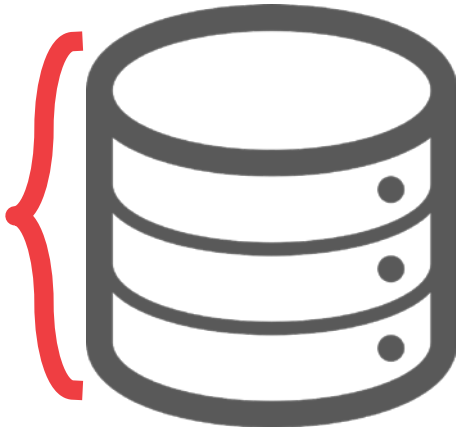
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Workspace

Record	Value	Write Timestamp
A	123	10000

Record	Value	Write Timestamp
A	123	10000
B	456	10000



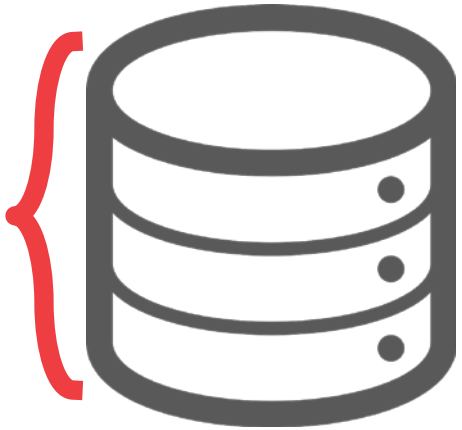
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Workspace

Record	Value	Write Timestamp
A	123	10000

Record	Value	Write Timestamp
A	123	10000
B	456	10000



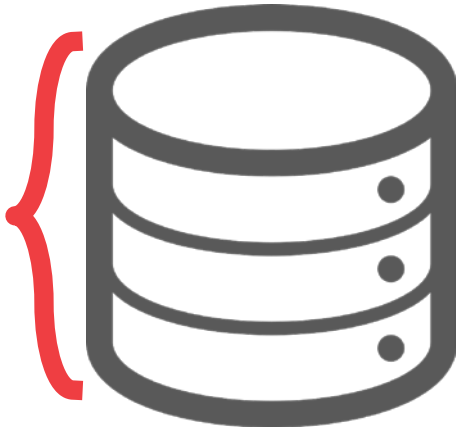
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Workspace

Record	Value	Write Timestamp
A	123	10000

Record	Value	Write Timestamp
A	123	10000
B	456	10000



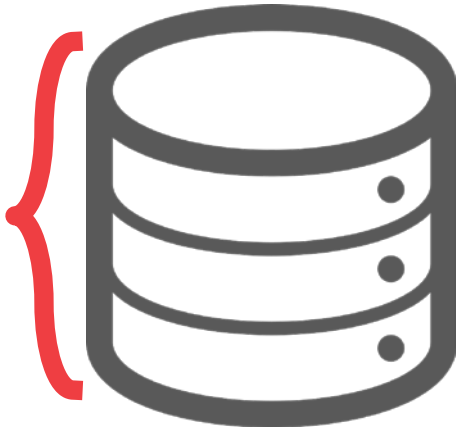
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Workspace

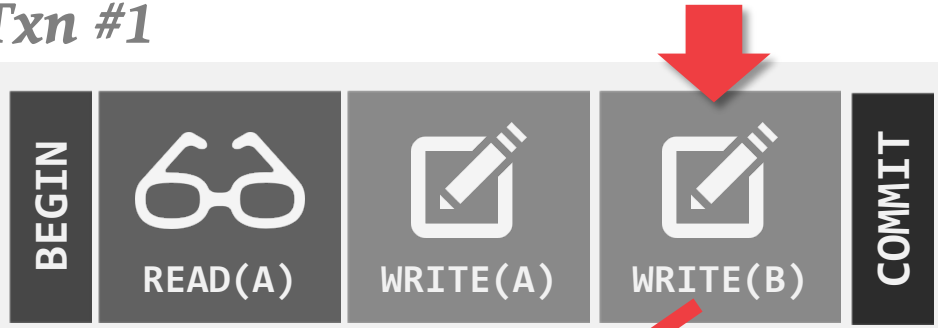
Record	Value	Write Timestamp
A	888	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

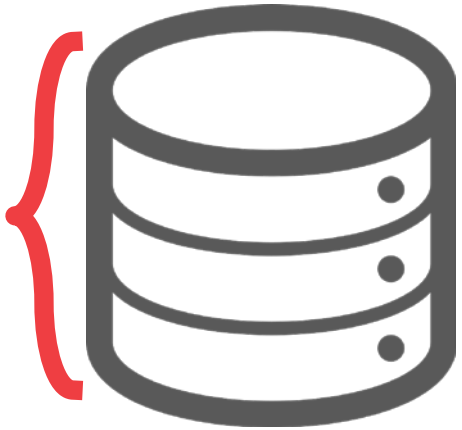
Txn #1



Workspace

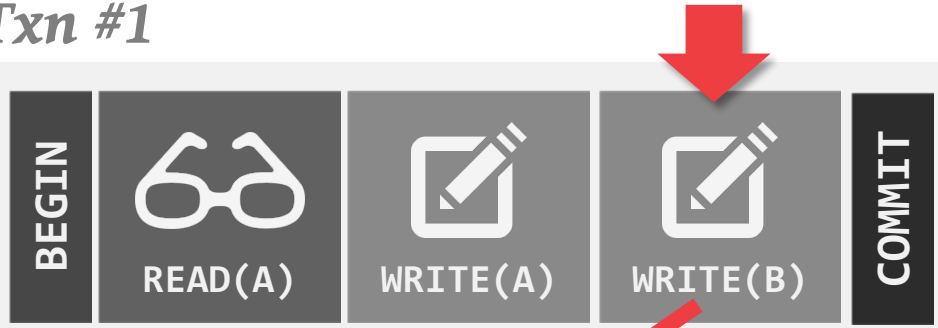
Record	Value	Write Timestamp
A	~~888~~	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

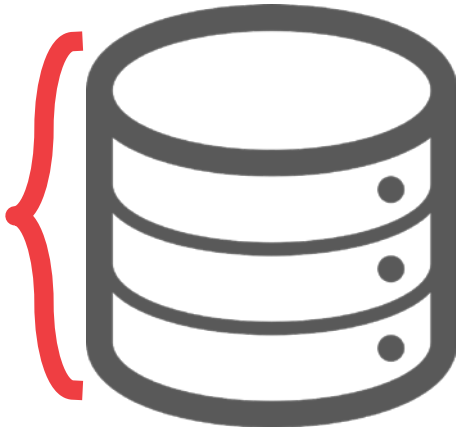
Txn #1



Workspace

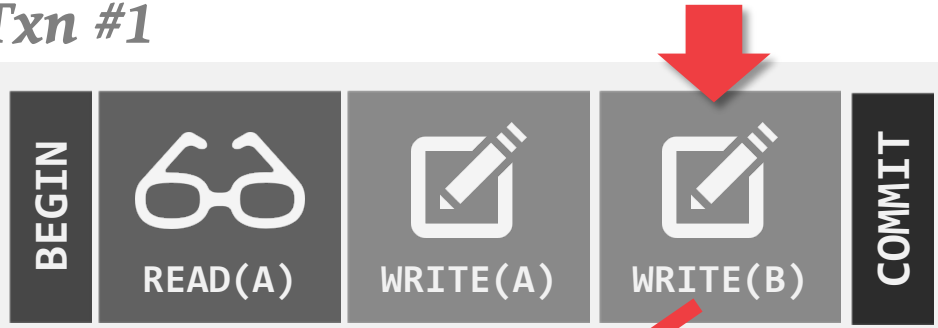
Record	Value	Write Timestamp
A	~~888~~	∞
B	456	10000

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

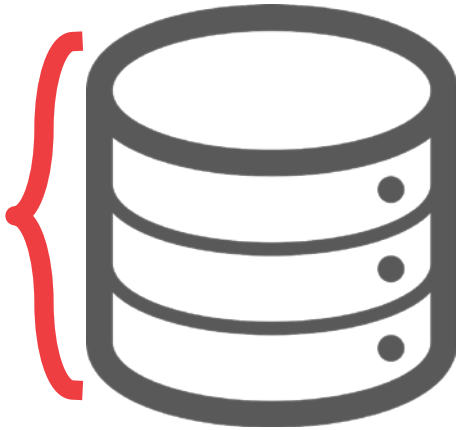
Txn #1



Workspace

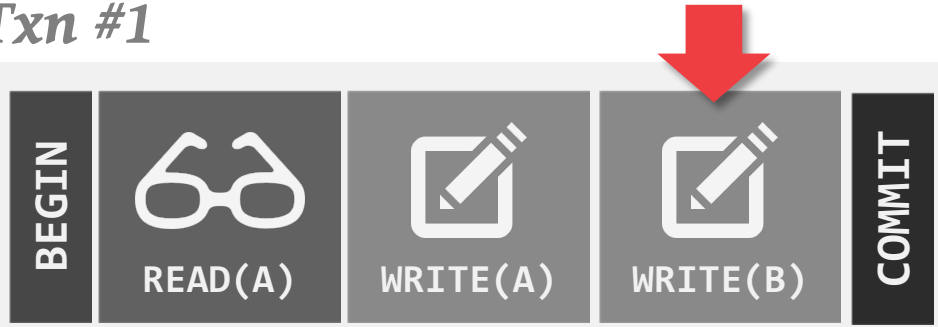
Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

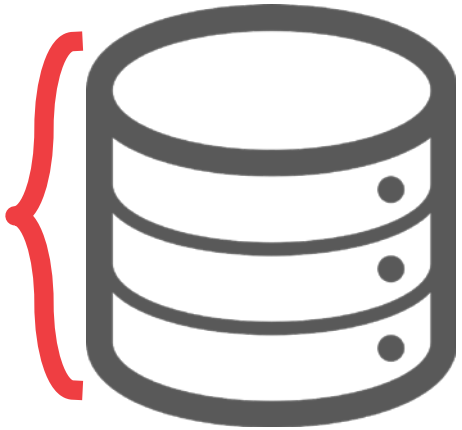
Txn #1



Workspace

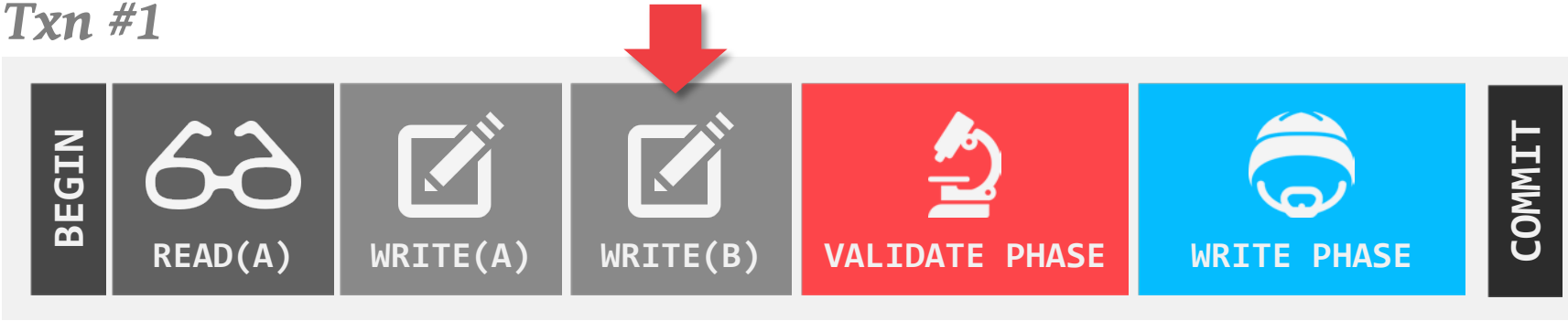
Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

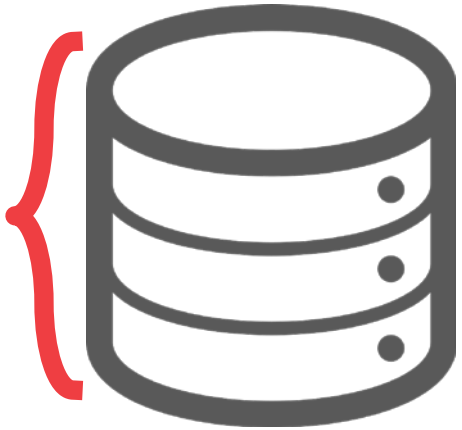
Txn #1



Workspace

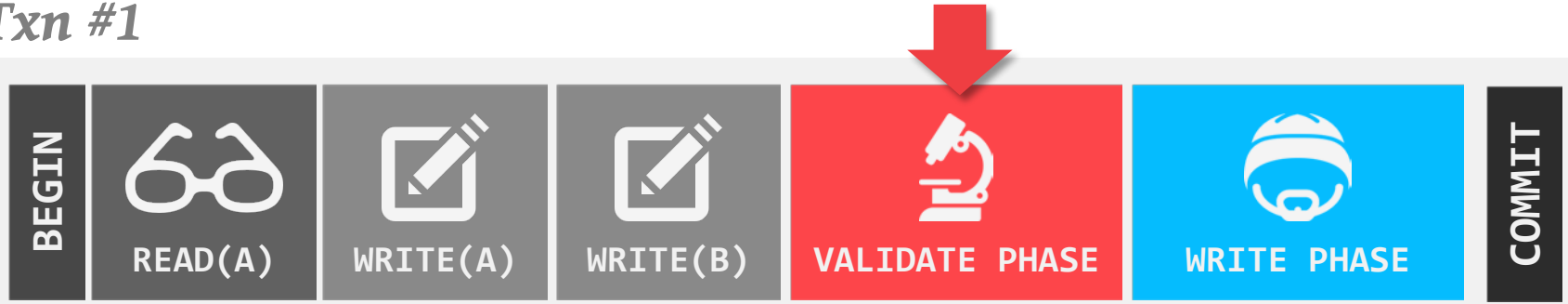
Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

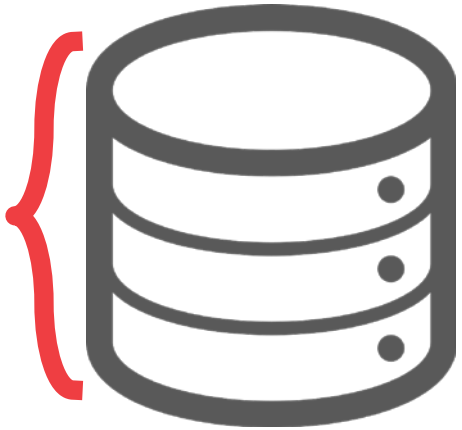
Txn #1



Workspace

Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



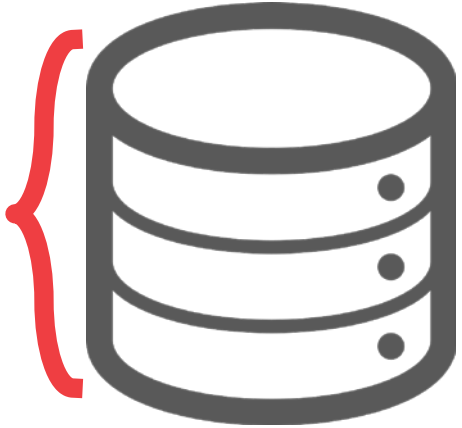
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Workspace

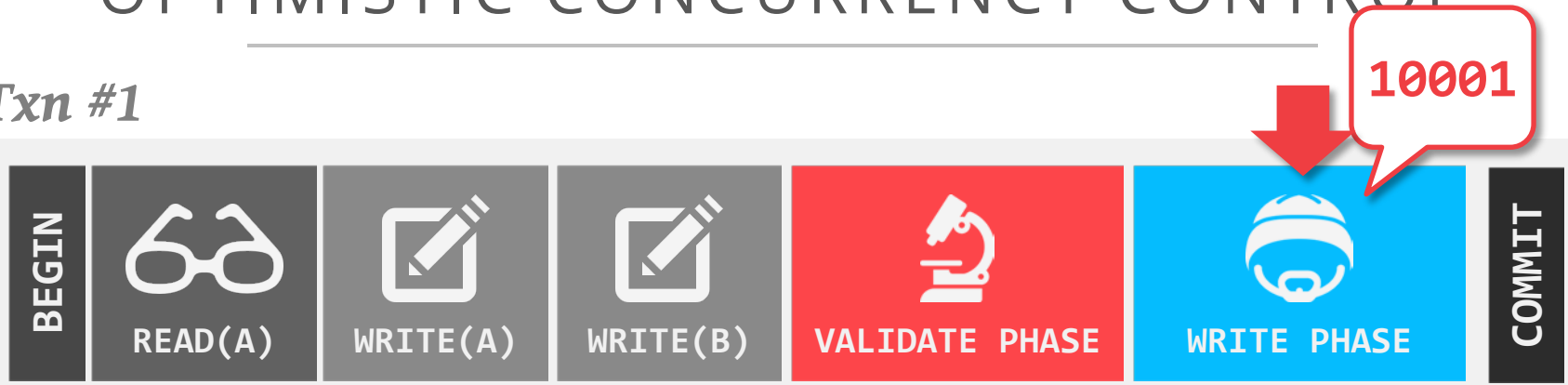
Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

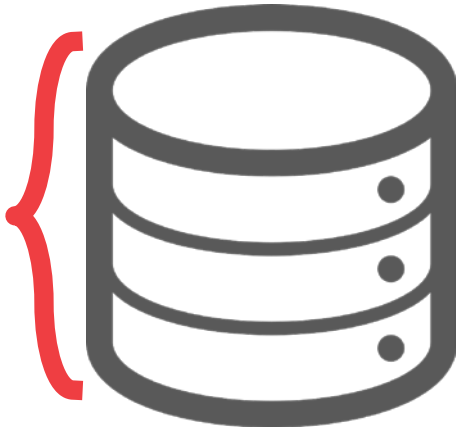
Txn #1



Workspace

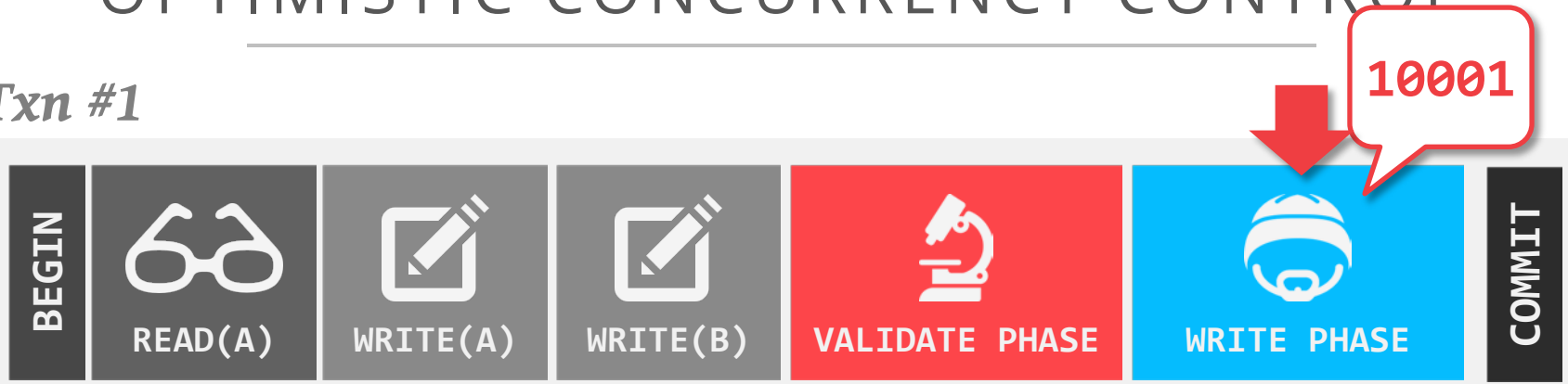
Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	123	10000
B	456	10000



OPTIMISTIC CONCURRENCY CONTROL

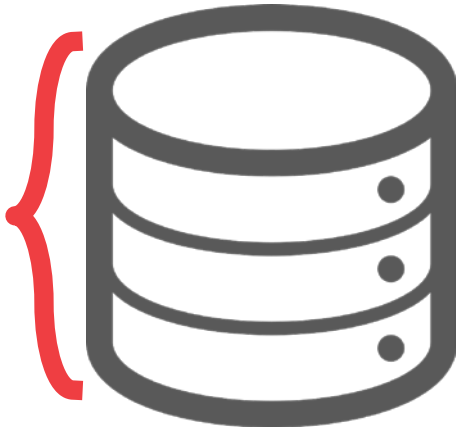
Txn #1



Workspace

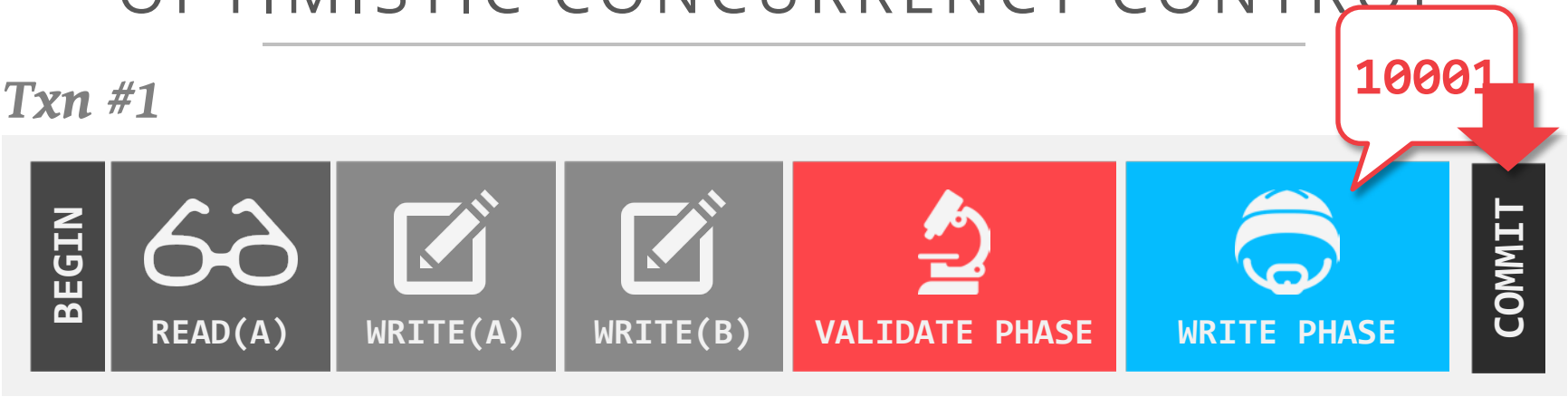
Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	888	10001
B	999	10001

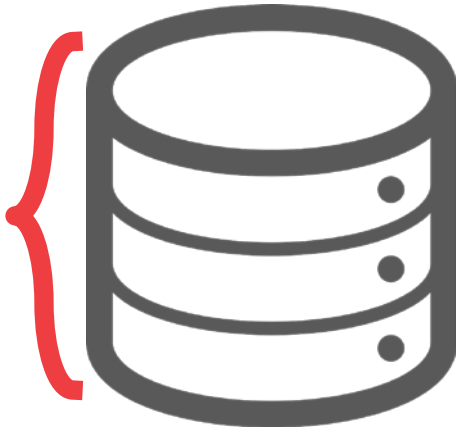


OPTIMISTIC CONCURRENCY CONTROL

Txn #1



Record	Value	Write Timestamp
A	888	10001
B	999	10001



READ PHASE

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

The DBMS copies every tuple that the txn accesses from the shared database to its workspace ensure repeatable reads.

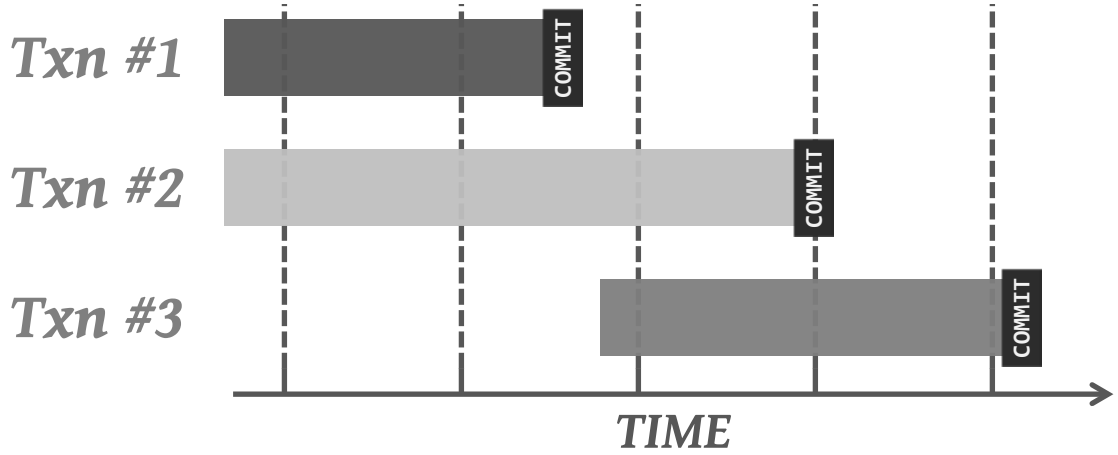
VALIDATION PHASE

When the txn invokes **COMMIT**, the DBMS checks if it conflicts with other txns.

Two methods for this phase:
→ Backward Validation
→ Forward Validation

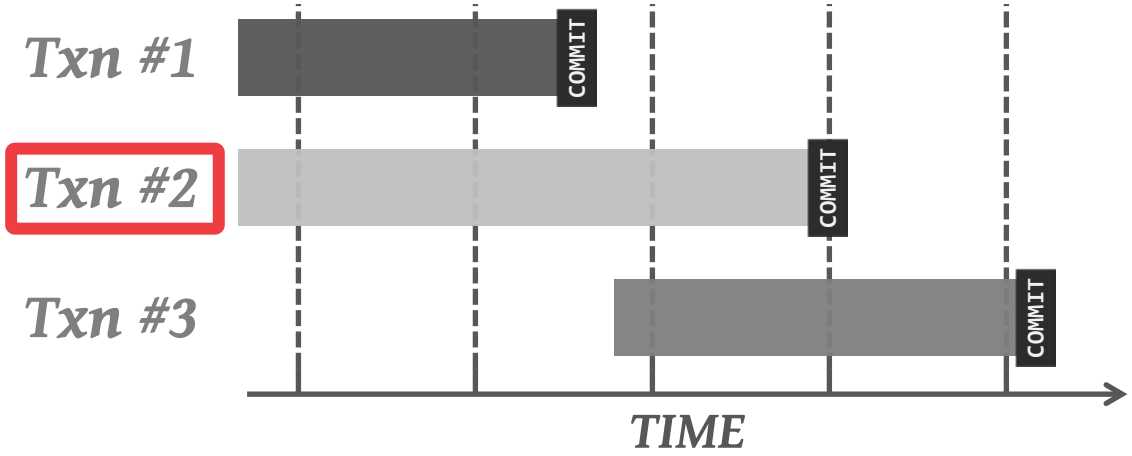
BACKWARD VALIDATION

Check whether the committing txn intersects its read/write sets with those of any txns that have **already** committed.



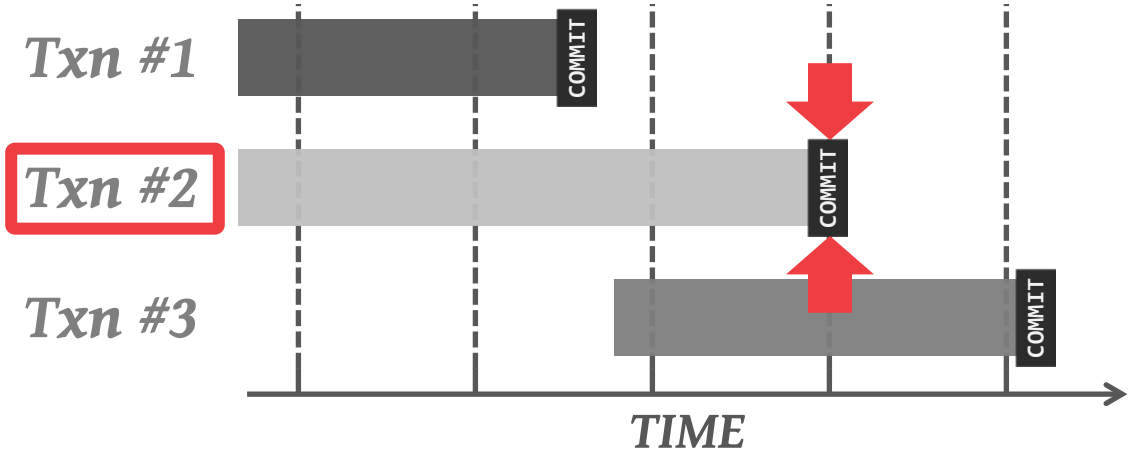
BACKWARD VALIDATION

Check whether the committing txn intersects its read/write sets with those of any txns that have **already** committed.



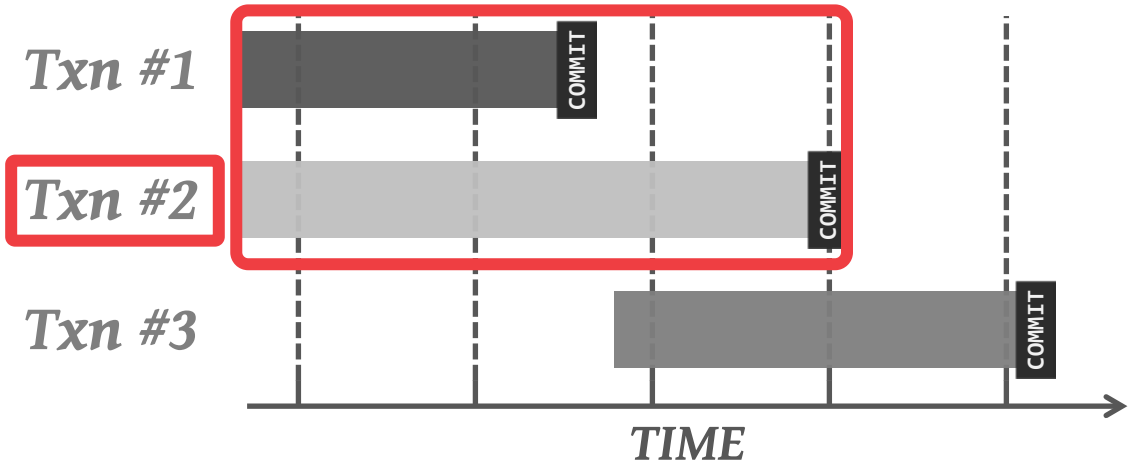
BACKWARD VALIDATION

Check whether the committing txn intersects its read/write sets with those of any txns that have **already** committed.



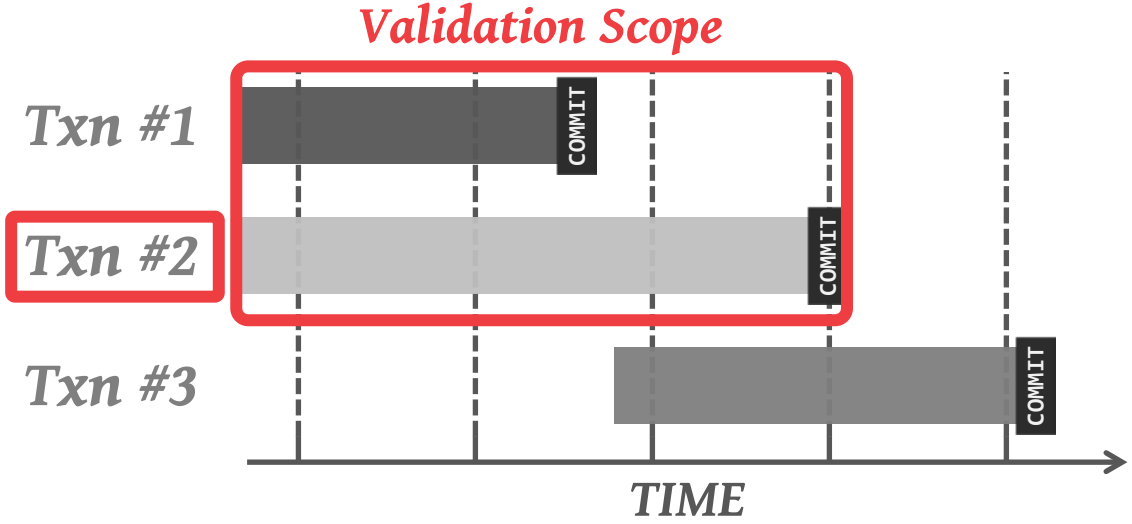
BACKWARD VALIDATION

Check whether the committing txn intersects its read/write sets with those of any txns that have **already** committed.



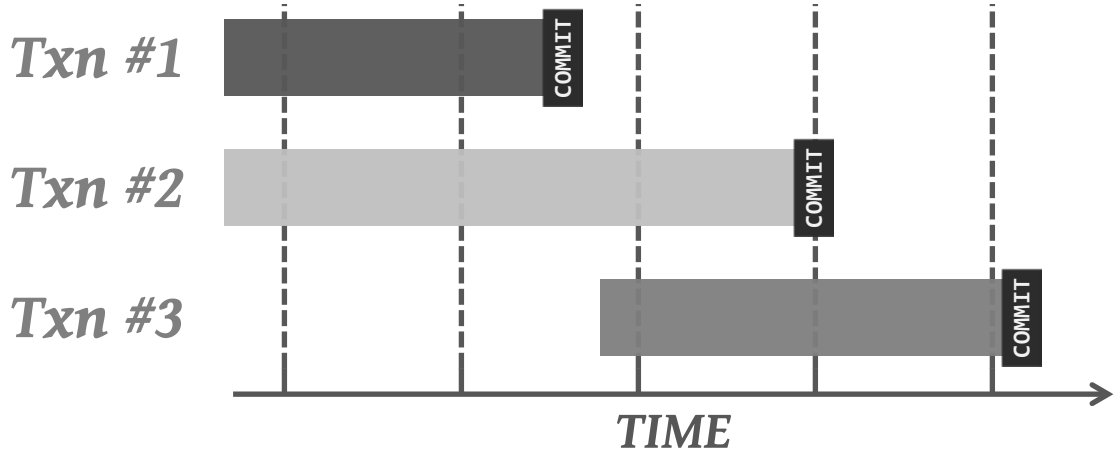
BACKWARD VALIDATION

Check whether the committing txn intersects its read/write sets with those of any txns that have **already** committed.



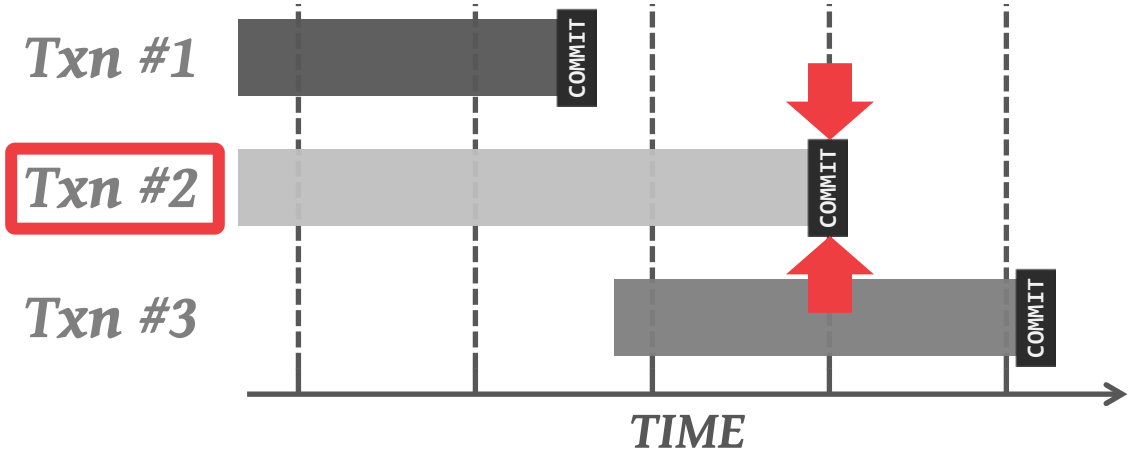
FORWARD VALIDATION

Check whether the committing txn intersects its read/write sets with any active txns that have **not** yet committed.



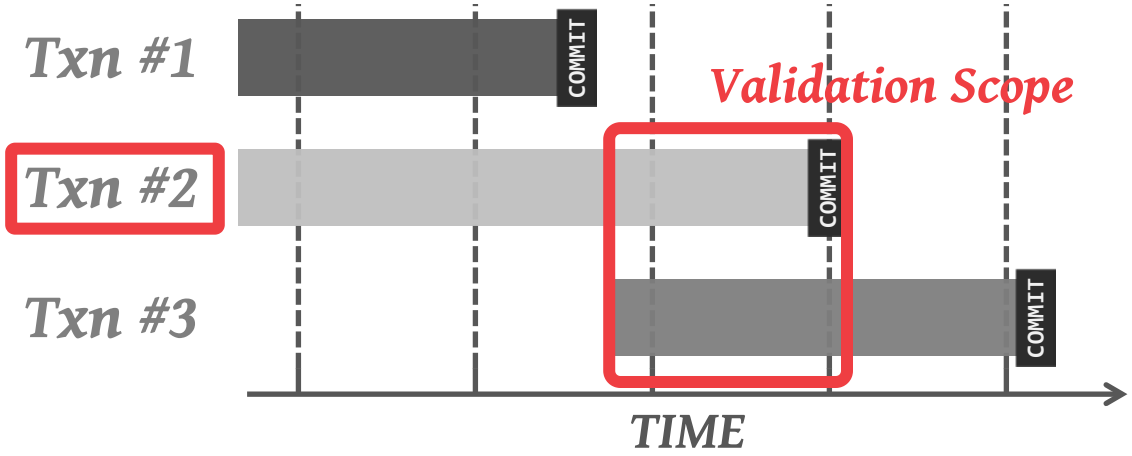
FORWARD VALIDATION

Check whether the committing txn intersects its read/write sets with any active txns that have **not** yet committed.



FORWARD VALIDATION

Check whether the committing txn intersects its read/write sets with any active txns that have **not** yet committed.



VALIDATION PHASE

Original OCC uses serial validation.

Parallel validation means that each txn must check the read/write sets of other txns that are trying to validate at the same time.

→ Each txn has to acquire locks for its write set records in some **global order**.

→ The txn does not need locks for read set records.

WRITE PHASE

The DBMS propagates the changes in the txn's write set to the database and makes them visible to other txns.

As each record is updated, the txn releases the lock acquired during the Validation Phase.

MODERN OCC

Harvard/MIT Silo

MIT/CMU TicToc

SILO

Single-node, in-memory OLTP DBMS.
→ Serializable OCC with parallel backward validation.
→ Stored procedure-only API.

No writes to shared-memory for read txns.

Batched timestamp allocation using **epochs**.

Pure awesomeness from Eddie Kohler.

SPEEDY TRANSACTIONS IN MULTICORE IN-MEMORY DATABASES
SOSP 2013

SILO: EPOCHS

Time is sliced into fixed-length epochs (40ms).

All txns that start in the same epoch will be committed together at the end of the epoch.

→ Txns that span an epoch have to refresh themselves to be carried over into the next epoch.

Worker threads only need to synchronize at the beginning of each epoch.

SILO: TRANSACTION IDS

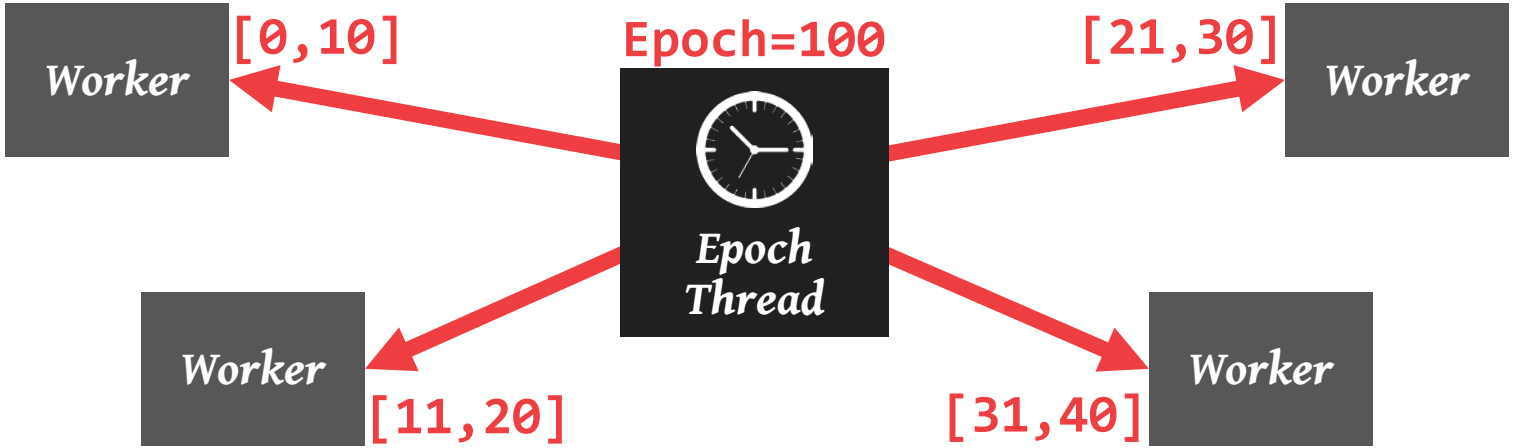
Each worker thread generates a unique txn id based on the current epoch number and the next value in its assigned batch.

SILO: TRANSACTION IDS

Each worker thread generates a unique txn id based on the current epoch number and the next value in its assigned batch.

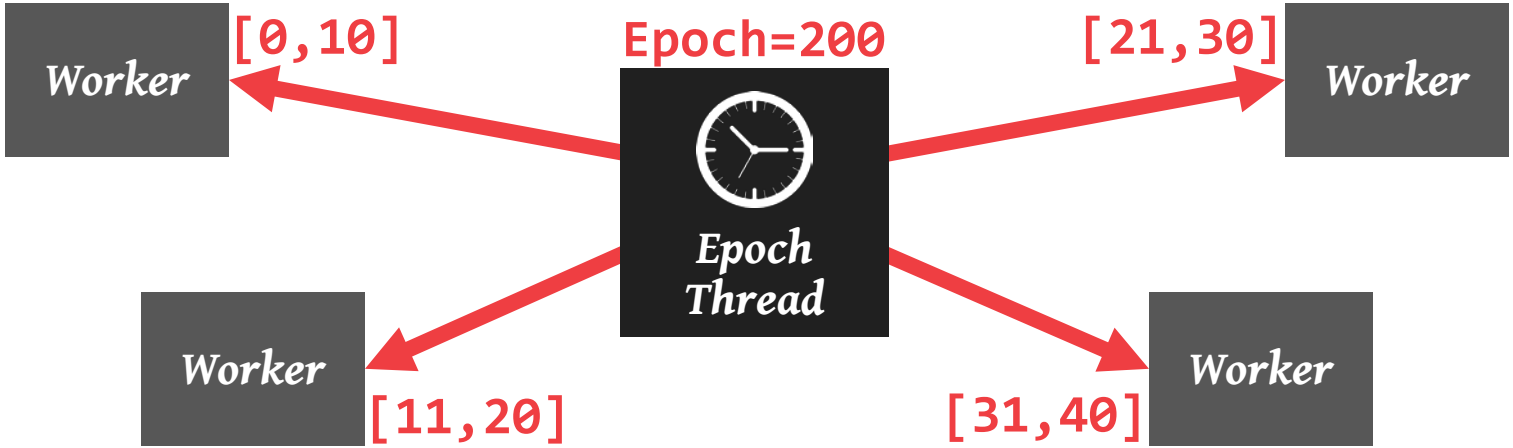
SILO: TRANSACTION IDS

Each worker thread generates a unique txn id based on the current epoch number and the next value in its assigned batch.



SILO: TRANSACTION IDS

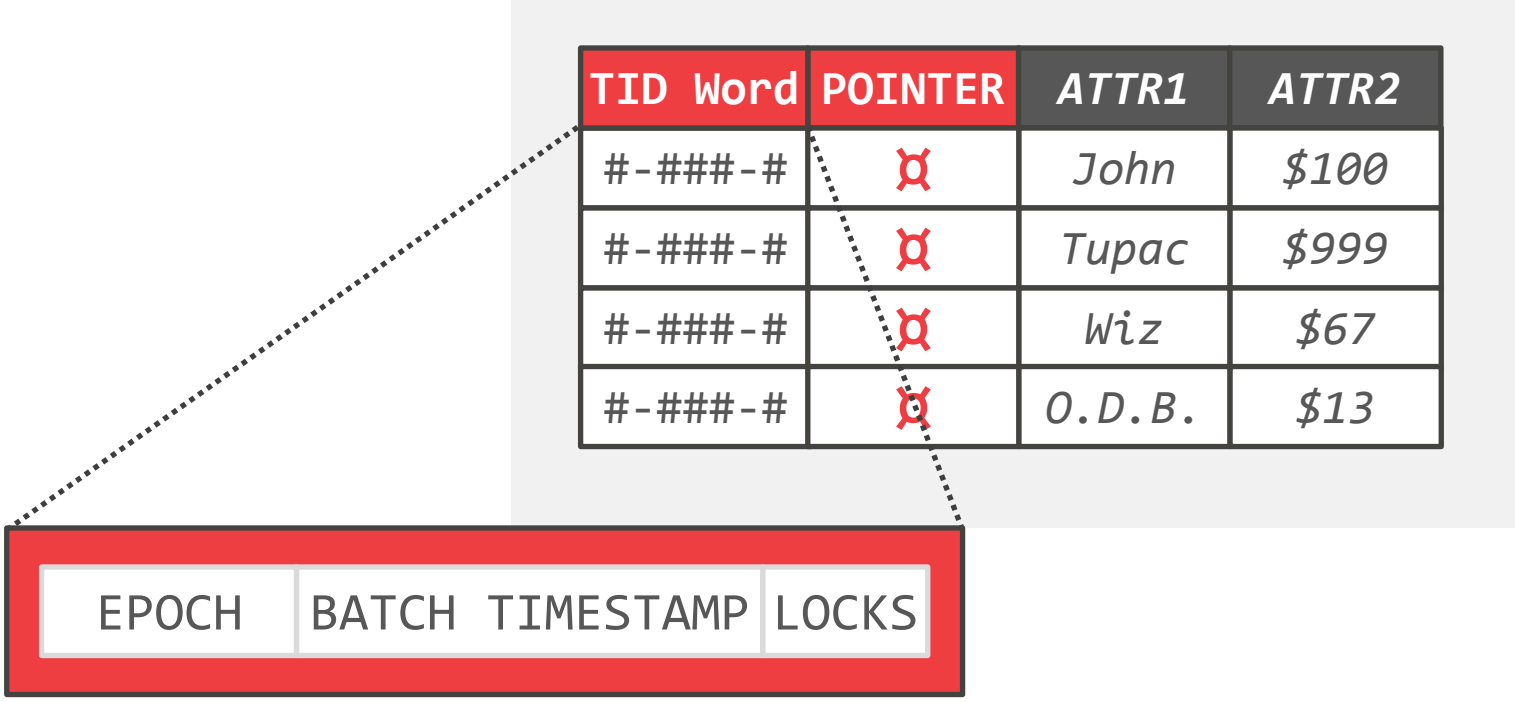
Each worker thread generates a unique txn id based on the current epoch number and the next value in its assigned batch.



SILO: COMMIT PROTOCOL

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

SILO: COMMIT PROTOCOL



SILO: COMMIT PROTOCOL

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

Step #1: Lock Write Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

Step #1: Lock Write Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

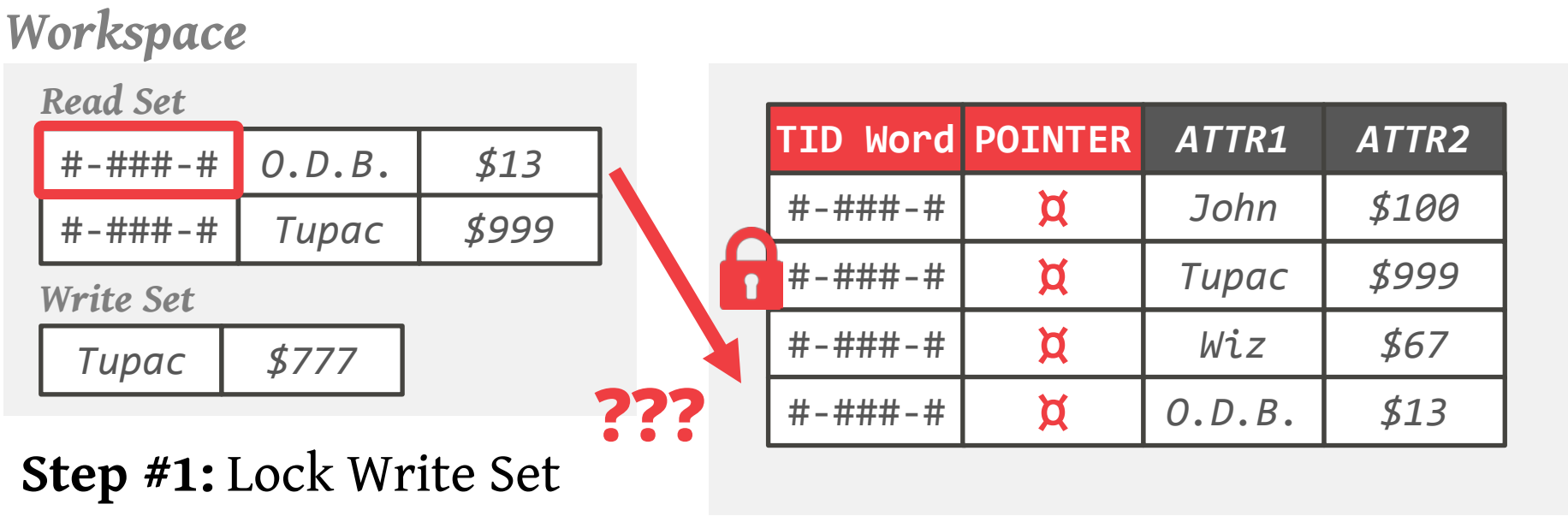
Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#	⌘	John	$100
#-###-#	⌘	Tupac	$999
#-###-#	⌘	Wiz	$67
#-###-#	⌘	O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

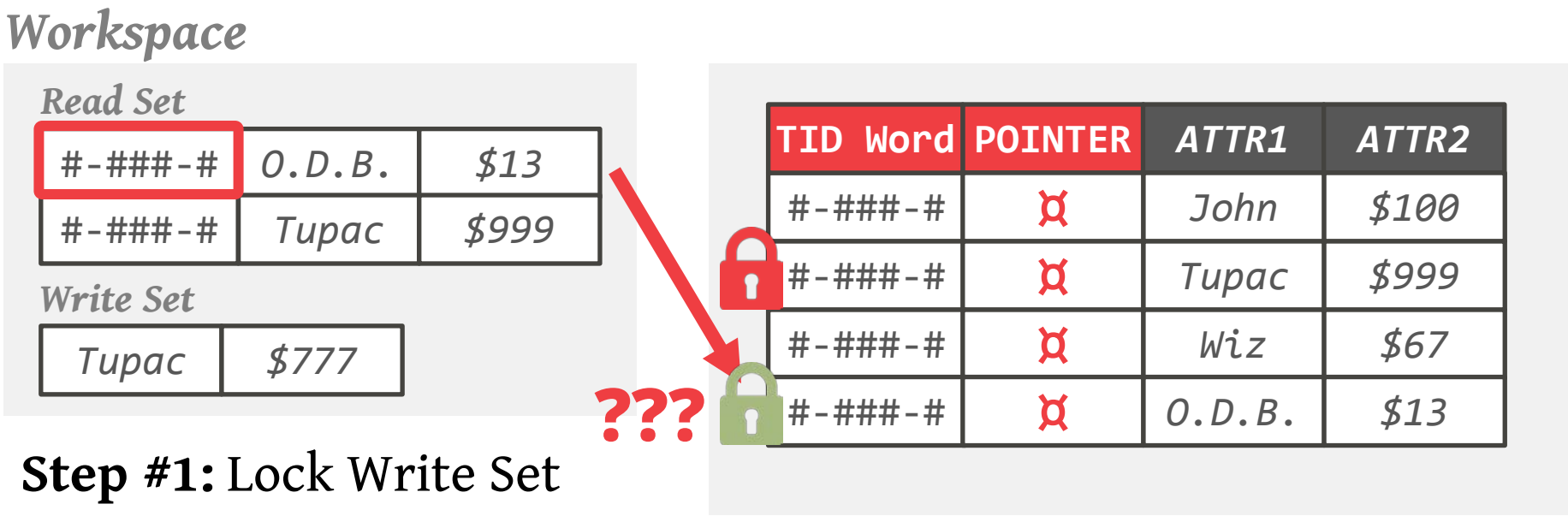
SILO: COMMIT PROTOCOL



Step #1: Lock Write Set

Step #2: Examine Read Set

SILO: COMMIT PROTOCOL



Step #1: Lock Write Set

Step #2: Examine Read Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#	⌘	John	$100
#-###-#	⌘	Tupac	$999
#-###-#	⌘	Wiz	$67
#-###-#	⌘	O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#	⌘	John	$100
#-###-#	⌘	Tupac	$999
#-###-#	⌘	Wiz	$67
#-###-#	⌘	O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#	⌘	John	$100
#-###-#	⌘	Tupac	$999
#-###-#	⌘	Wiz	$67
#-###-#	⌘	O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#		John	$100
#-###-#		Tupac	$999
#-###-#		Wiz	$67
#-###-#		O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

Step #3: Install Write Set

SILO: COMMIT PROTOCOL

Workspace

Read Set

#-###-#	O.D.B.	$13
#-###-#	Tupac	$999

Write Set

Tupac	$777

TID Word	POINTER	ATTR1	ATTR2
#-###-#	⌘	John	$100
#-###-#	⌘	Tupac	$777
#-###-#	⌘	Wiz	$67
#-###-#	⌘	O.D.B.	$13

Step #1: Lock Write Set

Step #2: Examine Read Set

Step #3: Install Write Set

SILO: GARBAGE COLLECTION

Cooperative threads GC.

Each worker thread marks a deleted object with a **reclamation epoch**.

→ This is the epoch after which no thread could access the object again, and thus can be safely removed.

→ Object references are maintained in thread-local storage to avoid unnecessary data movement.

SILO: RANGE QUERIES

DBMS handles phantoms by tracking the txn's scan set (node set) on indexes.

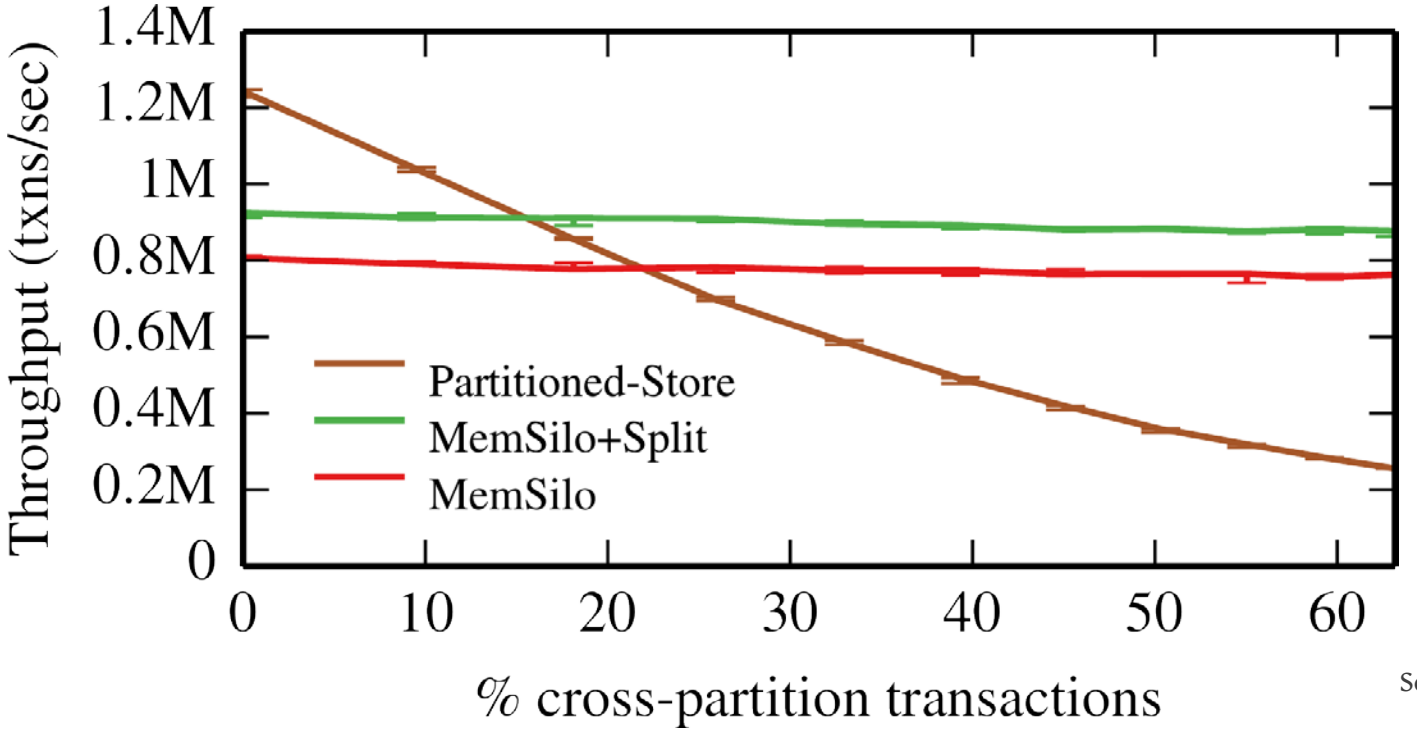
→ Have to include "virtual" entries for keys that do not exist in the index.

→ This is the same technique used in Hekaton.

We will discuss key-range and index gap locking next class...

SILO: PERFORMANCE

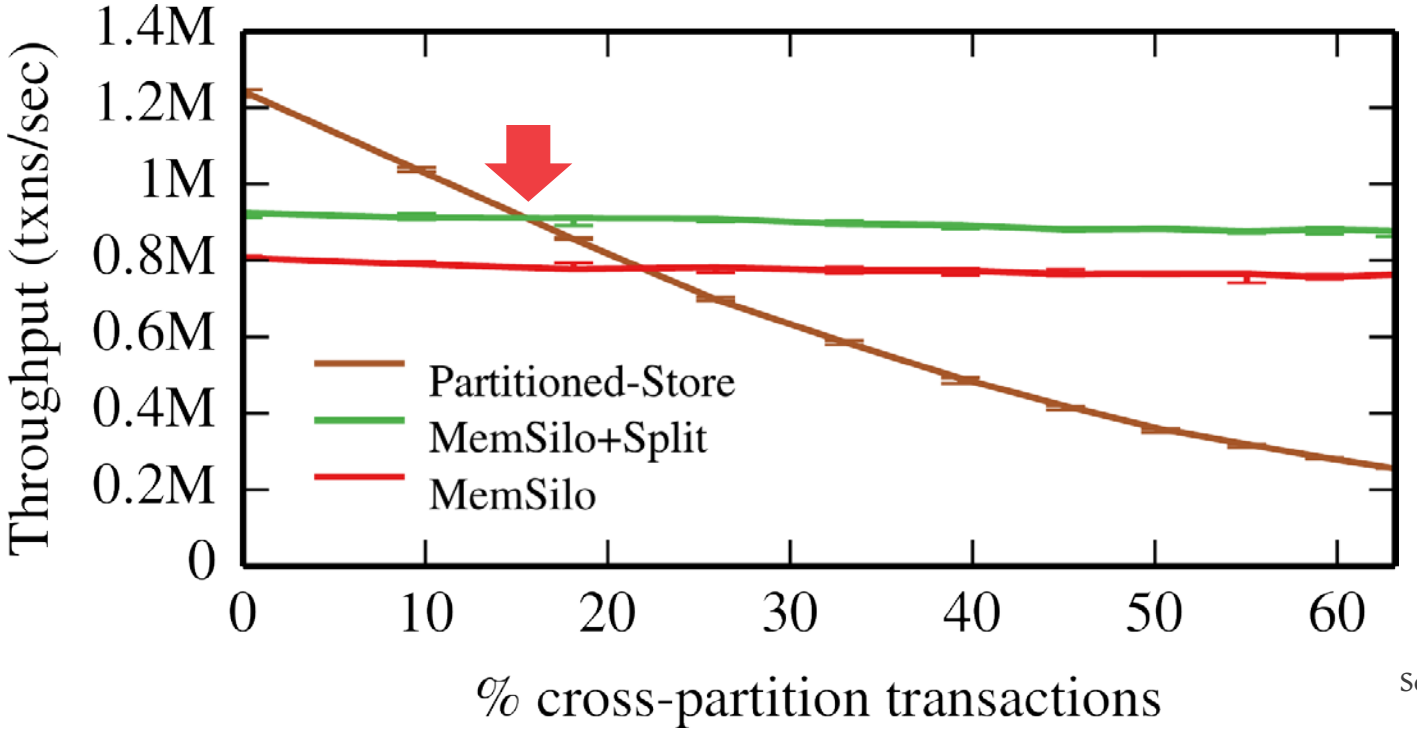
Database: TPC-C with 28 Warehouses
Processor: 4 sockets, 8 cores per socket



Source: Eddie Kohler

SILO: PERFORMANCE

Database: TPC-C with 28 Warehouses
Processor: 4 sockets, 8 cores per socket



Source: Eddie Kohler

TICTOC

Serializable OCC implemented in DBx1000.
→ Parallel backward validation
→ Stored procedure-only API

No global timestamp allocation.
Txn timestamps are derived from records.

TICTOC: TIME-TRAVELING OPTIMISTIC CONCURRENCY CONTROL
SIGMOD 2016

TICTOC: RECORD TIMESTAMPS

Write Timestamp (W-TS):

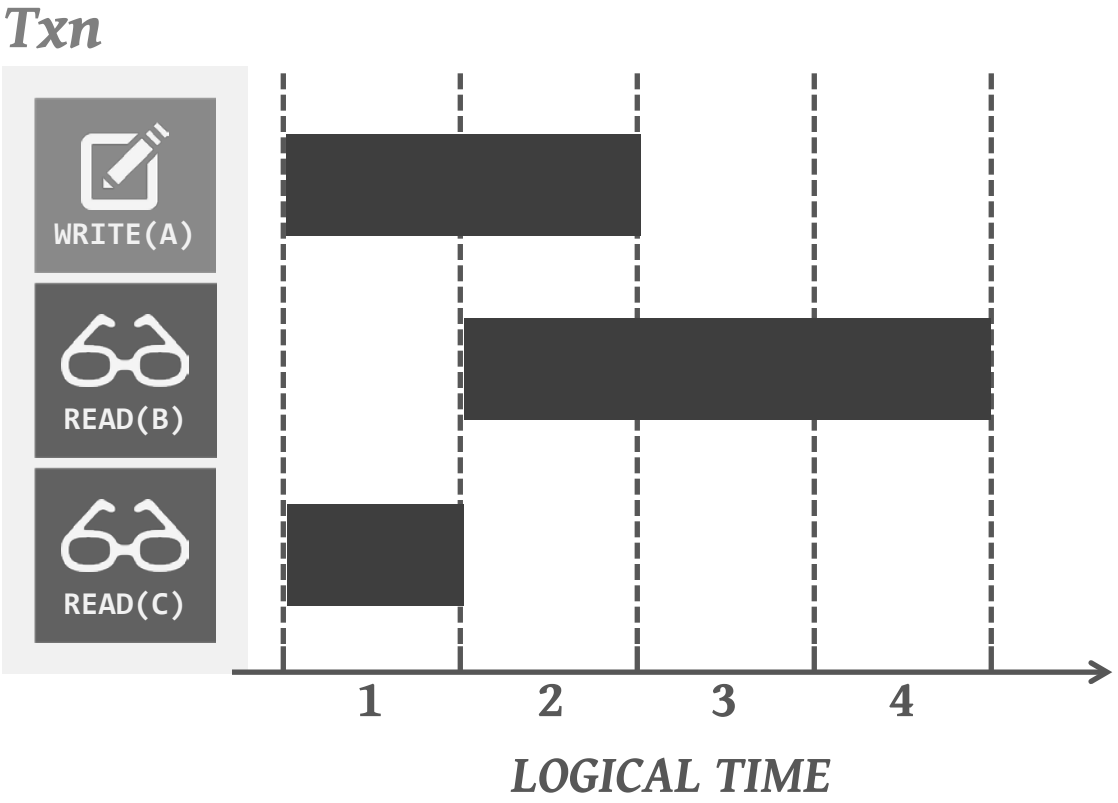
→ The logical timestamp of the last committed txn that wrote to the record.

Read Timestamp (R-TS):

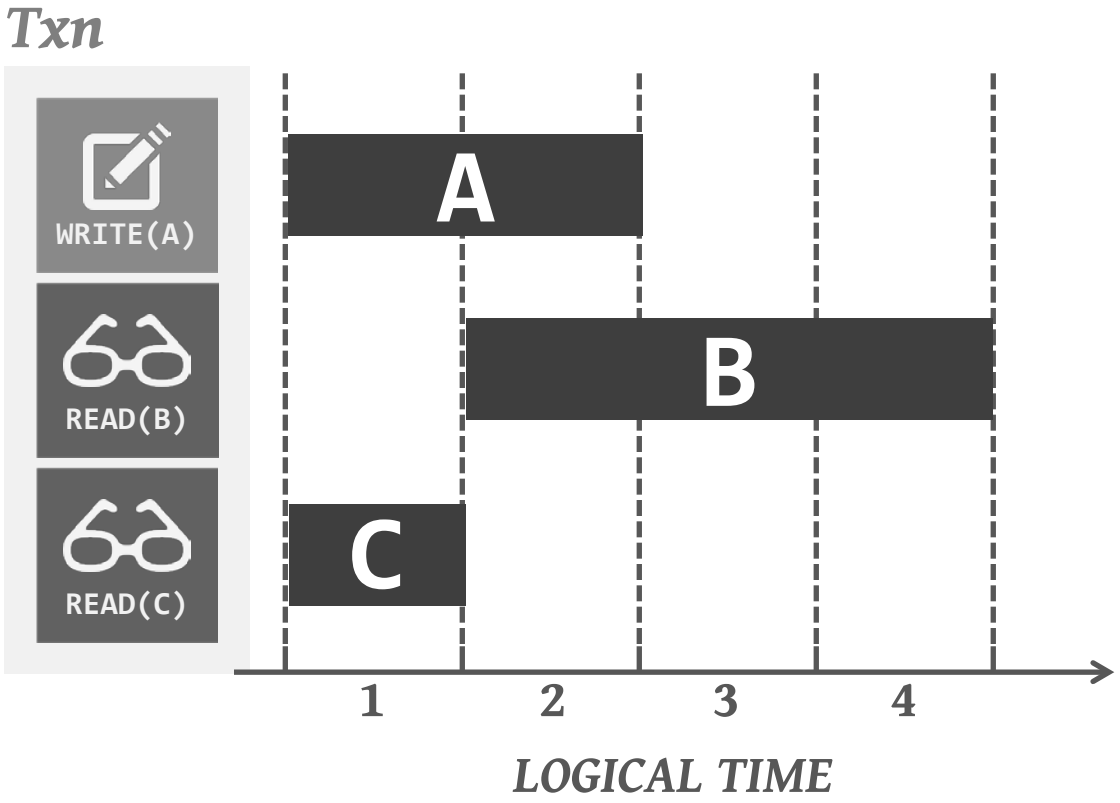
→ The logical timestamp of the last txn that read the record.

A record is considered valid from W-TS to R-TS

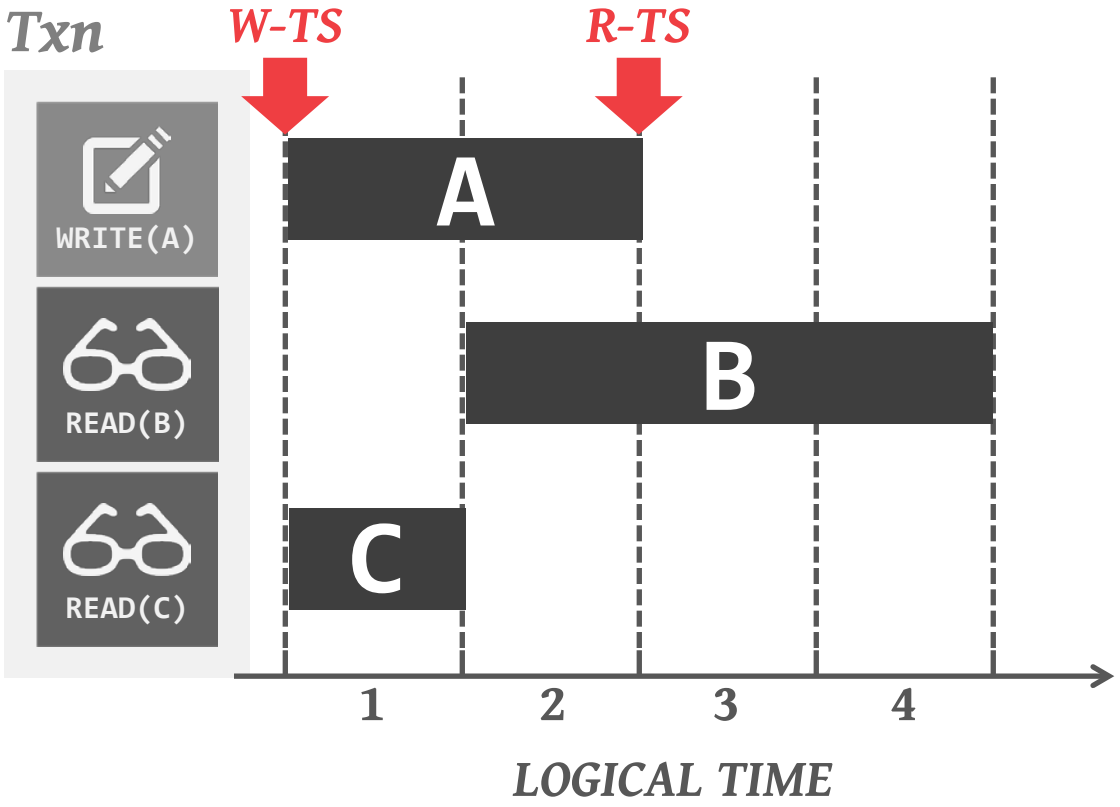
TICTOC: VALIDATION PHASE



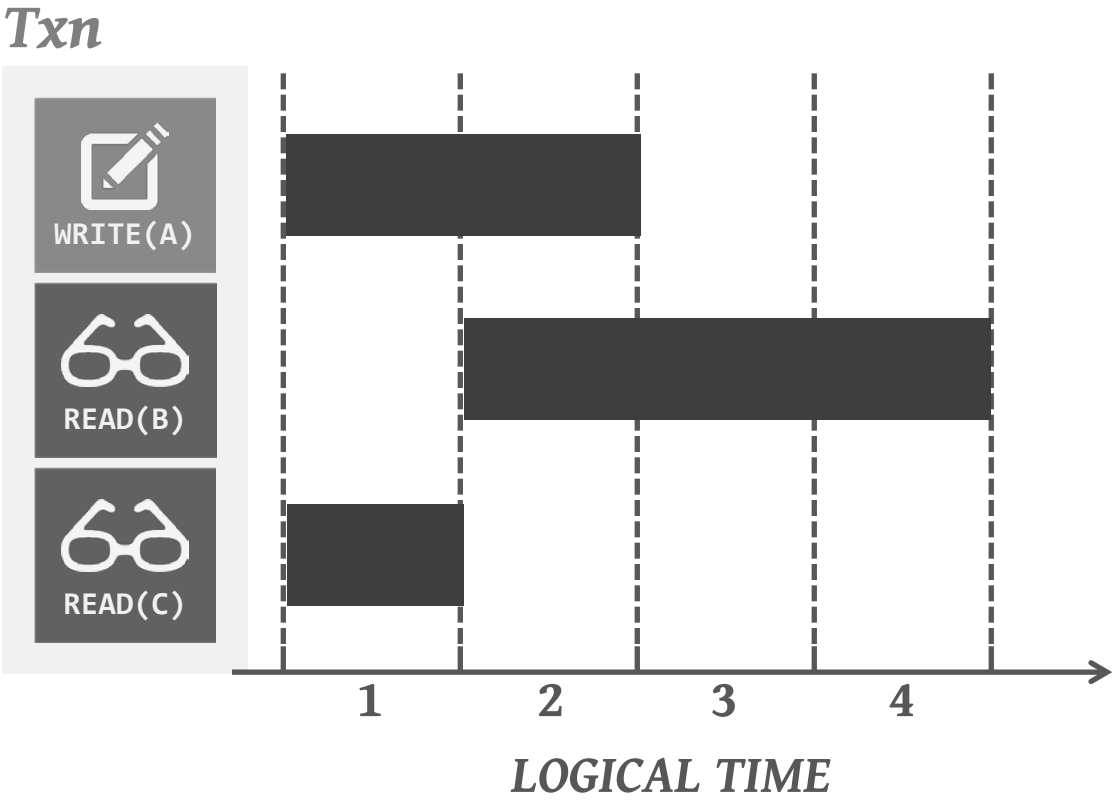
TICTOC: VALIDATION PHASE



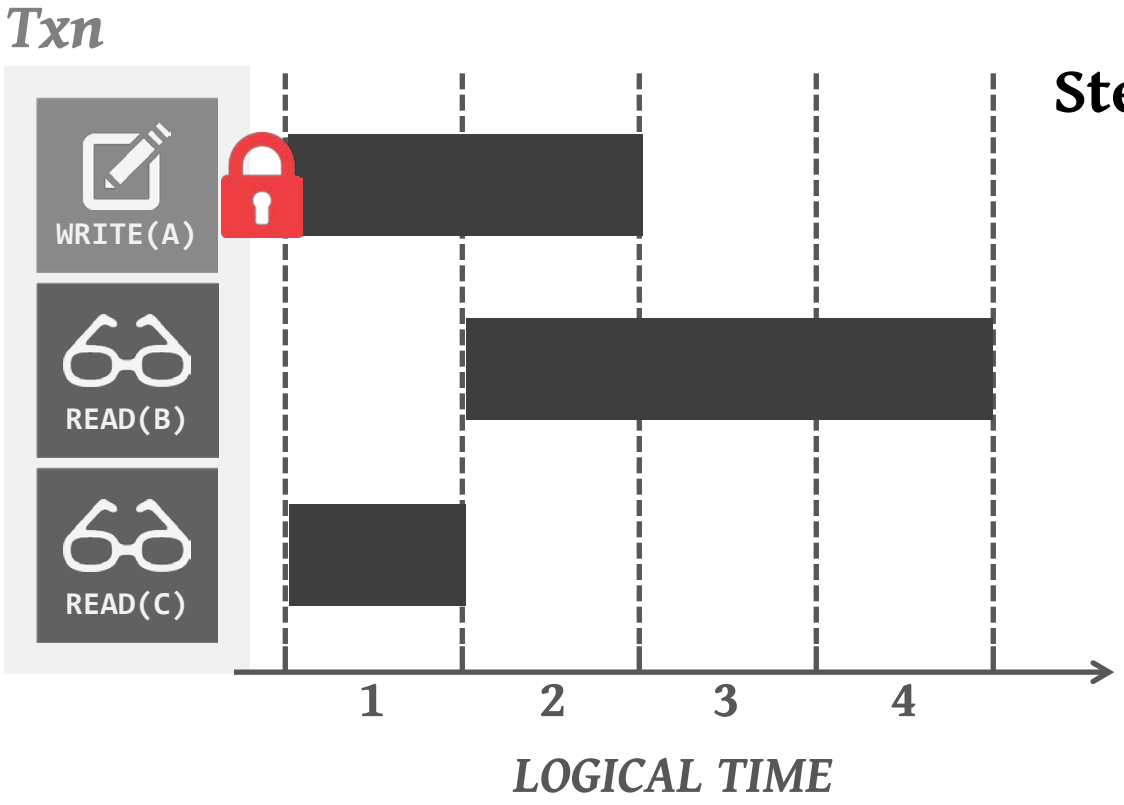
TICTOC: VALIDATION PHASE



TICTOC: VALIDATION PHASE

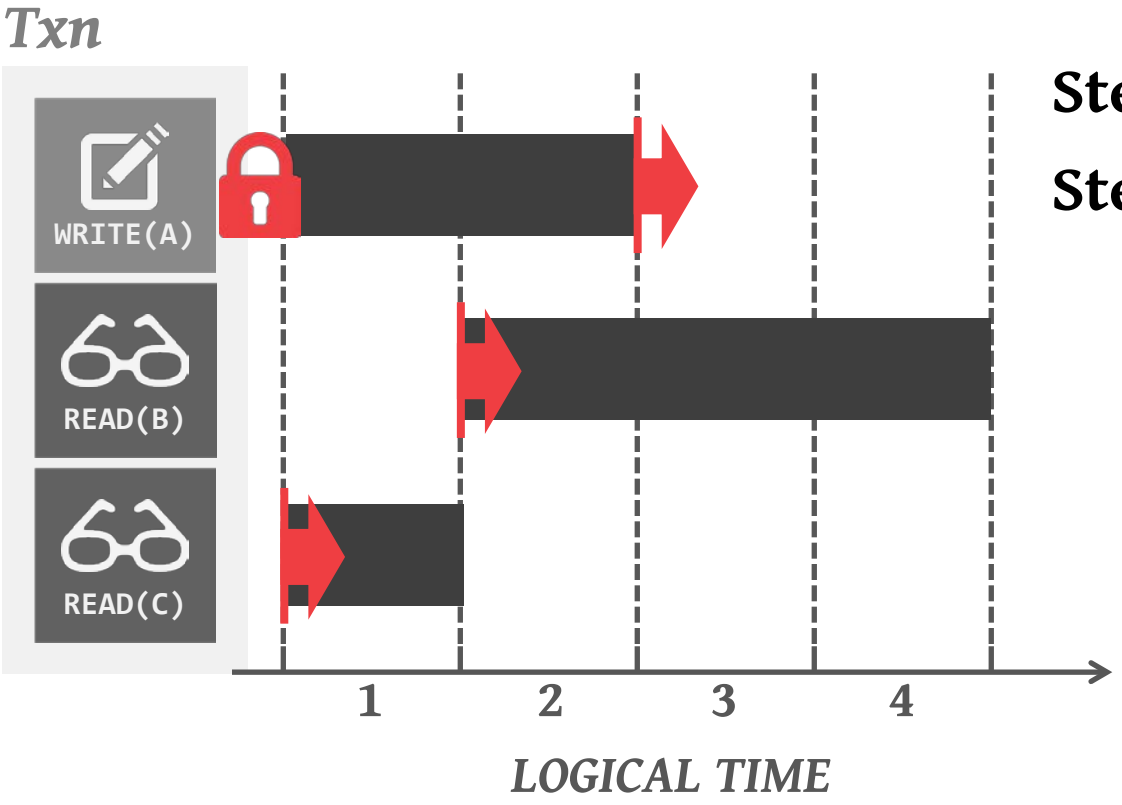


TICTOC: VALIDATION PHASE



Step #1: Lock Write Set

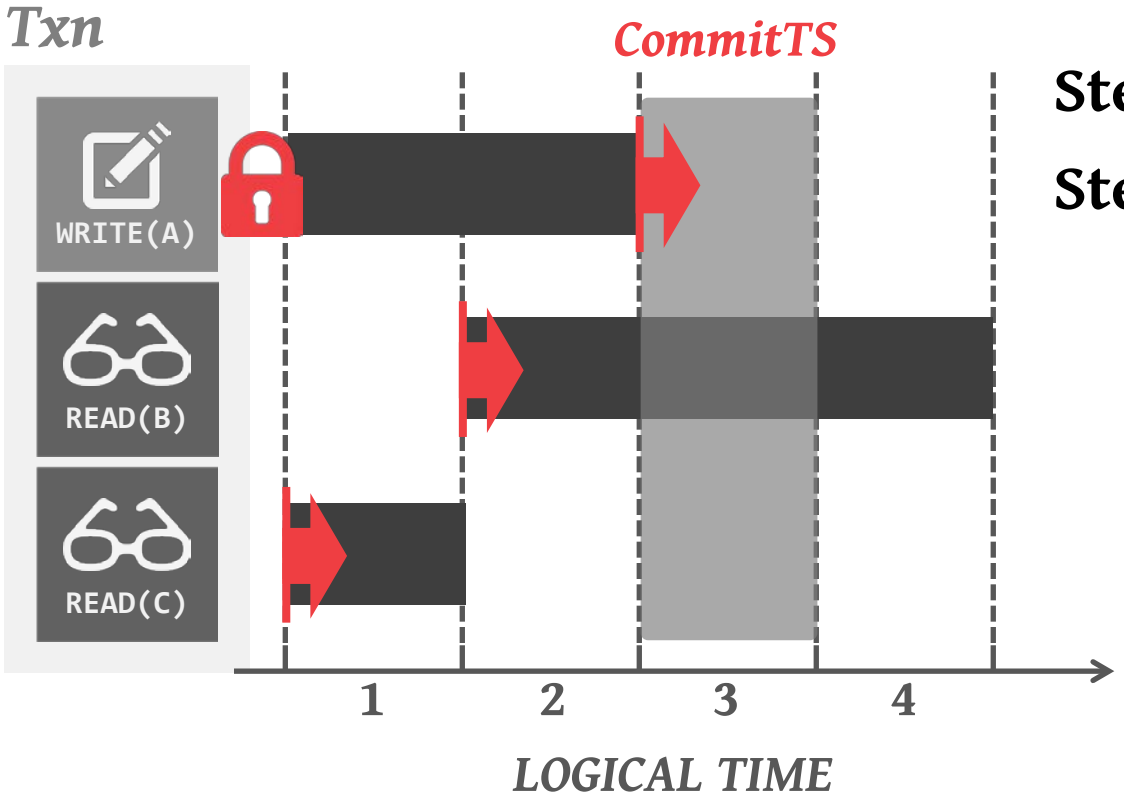
TICTOC: VALIDATION PHASE



Step #1: Lock Write Set

Step #2: Compute CommitTS

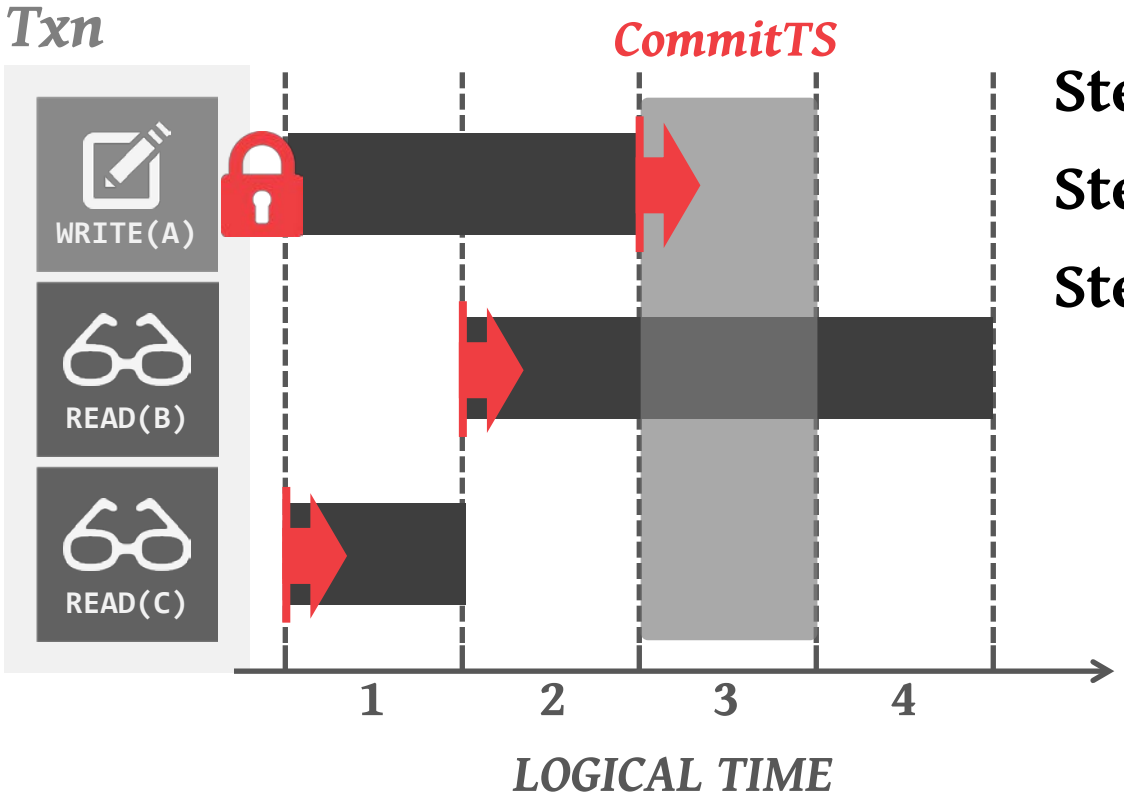
TICTOC: VALIDATION PHASE



Step #1: Lock Write Set

Step #2: Compute CommitTS

TICTOC: VALIDATION PHASE

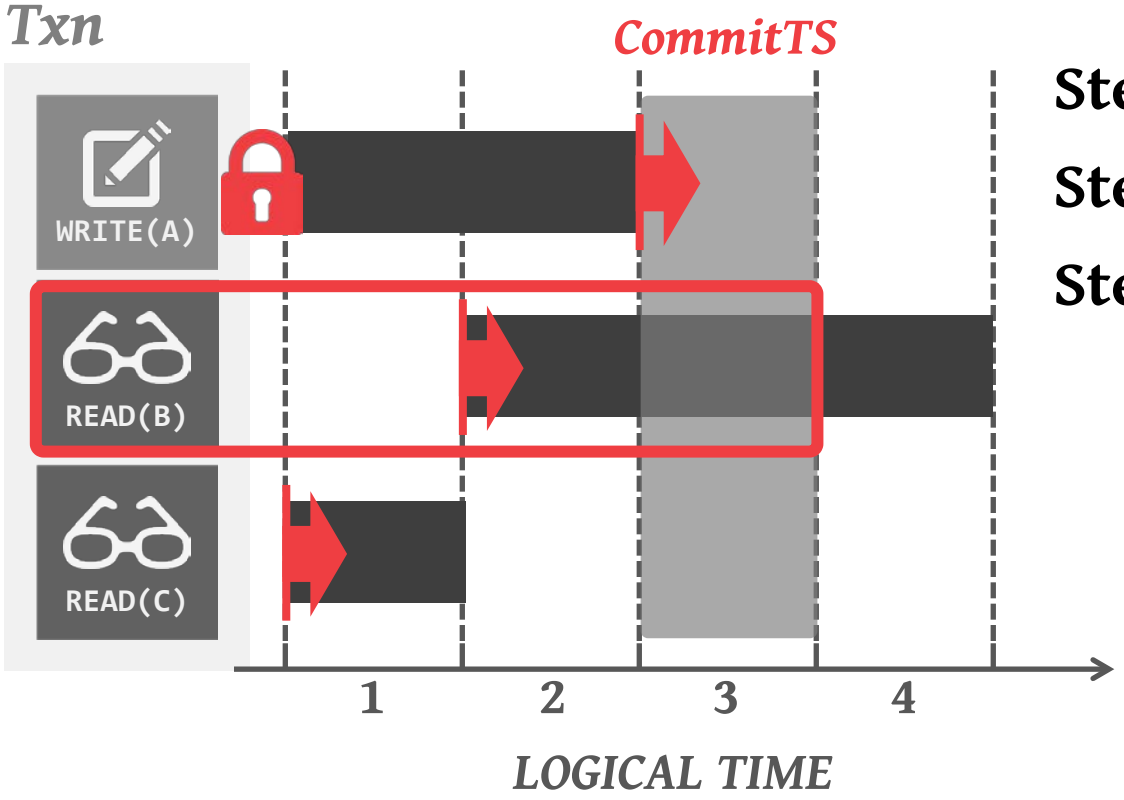


Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

TICTOC: VALIDATION PHASE

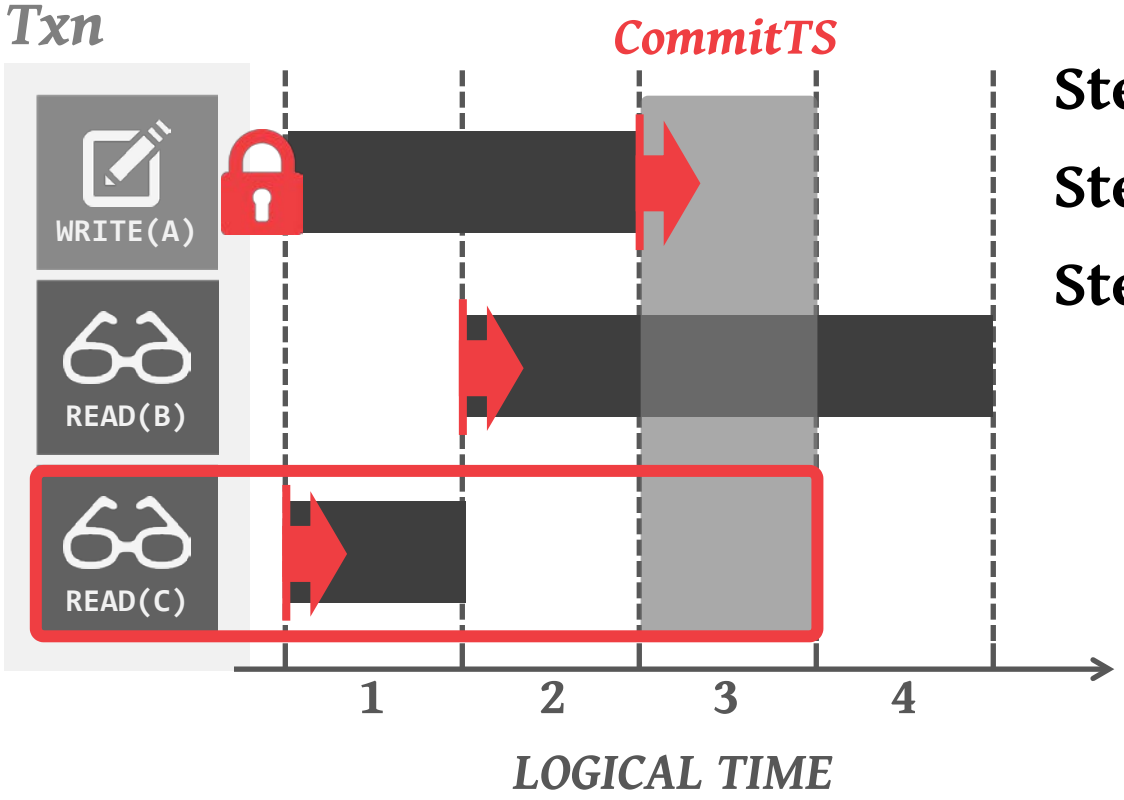


Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

TICTOC: VALIDATION PHASE

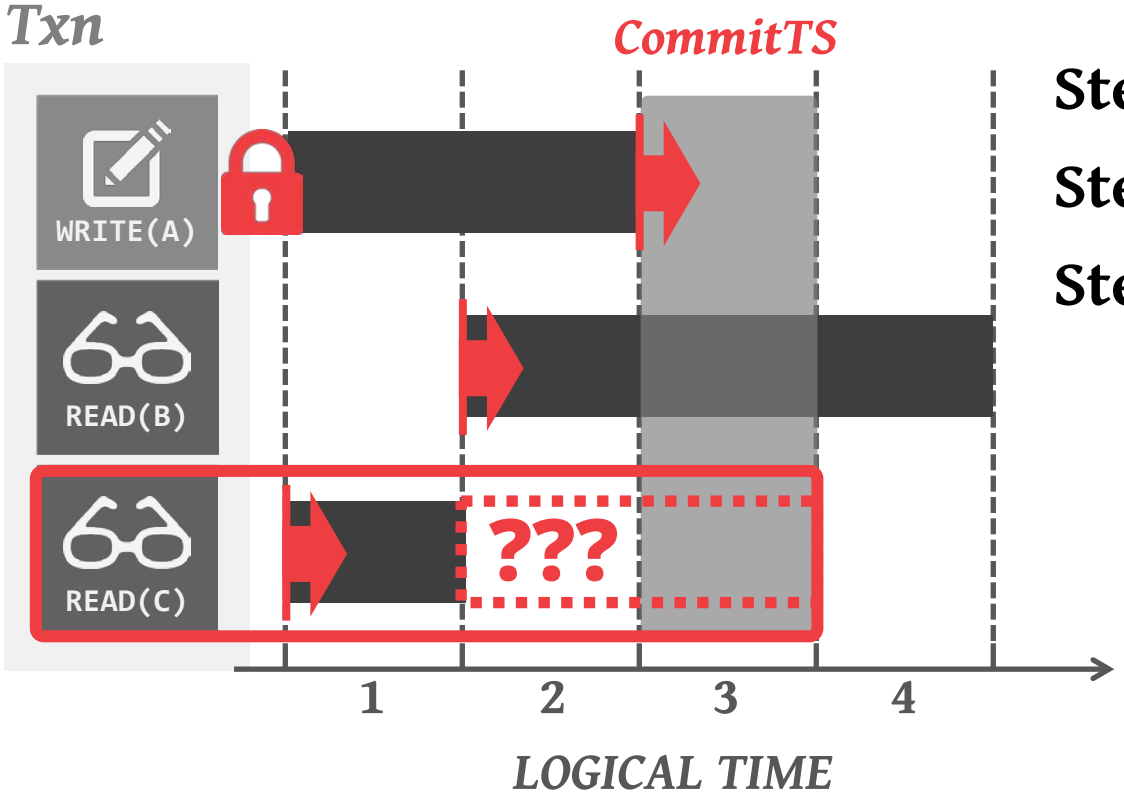


Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

TICTOC: VALIDATION PHASE

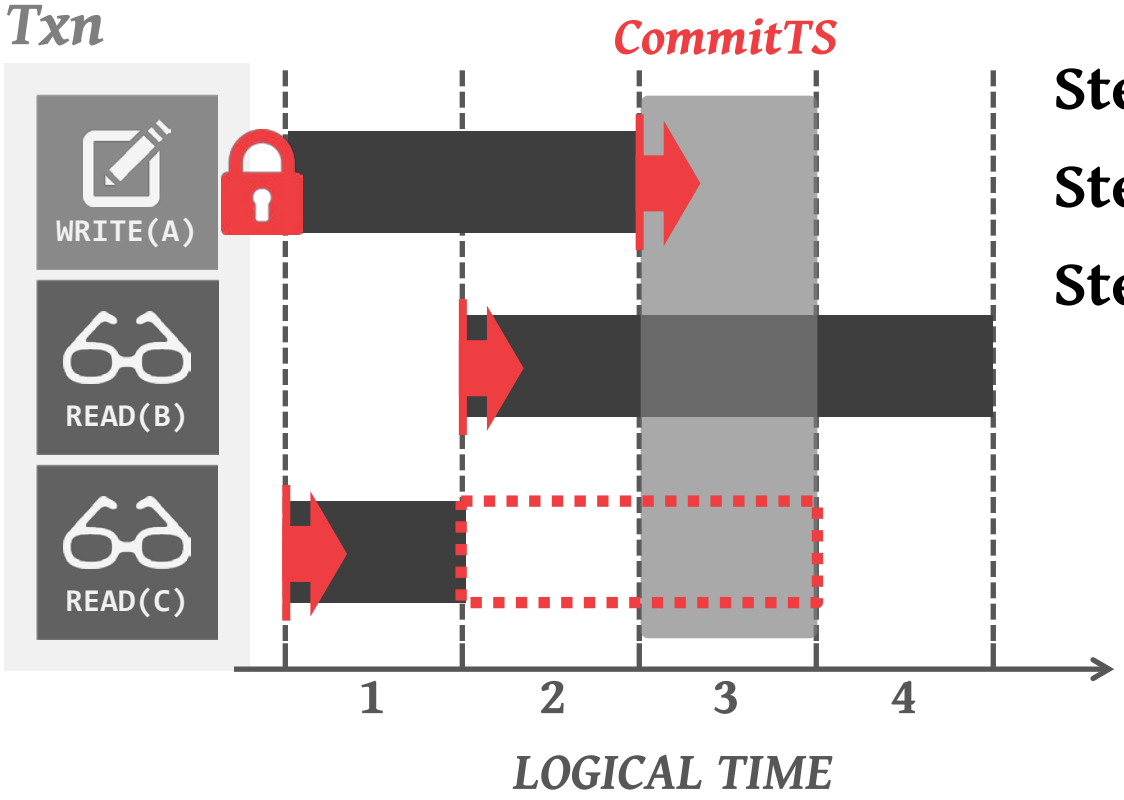


Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

TICTOC: VALIDATION PHASE

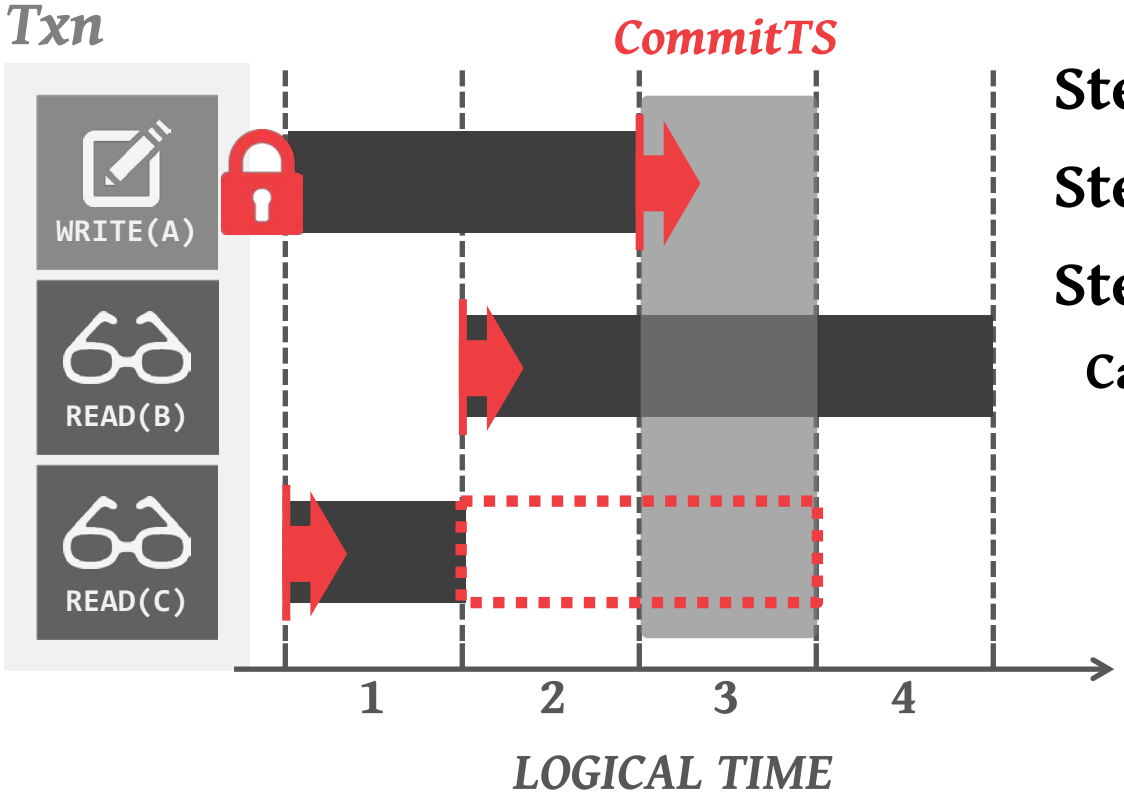


Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

TICTOC: VALIDATION PHASE



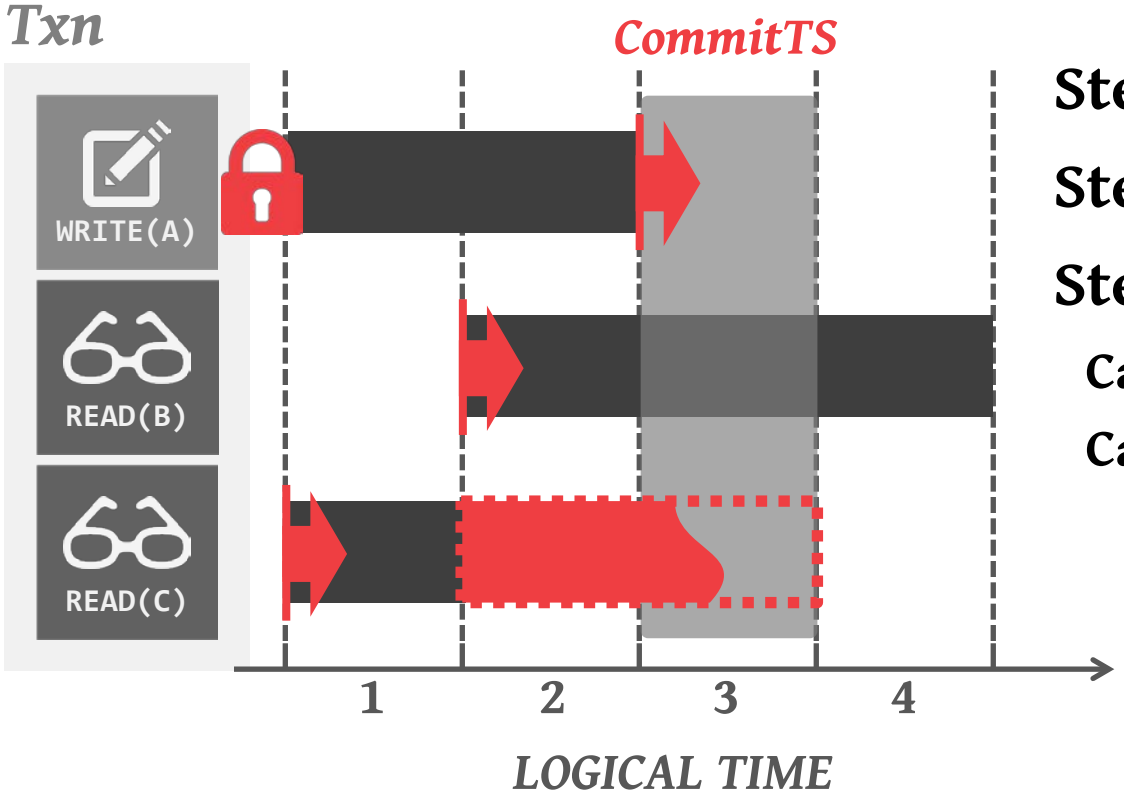
Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

Case 1: Latest Version

TICTOC: VALIDATION PHASE



Step #1: Lock Write Set

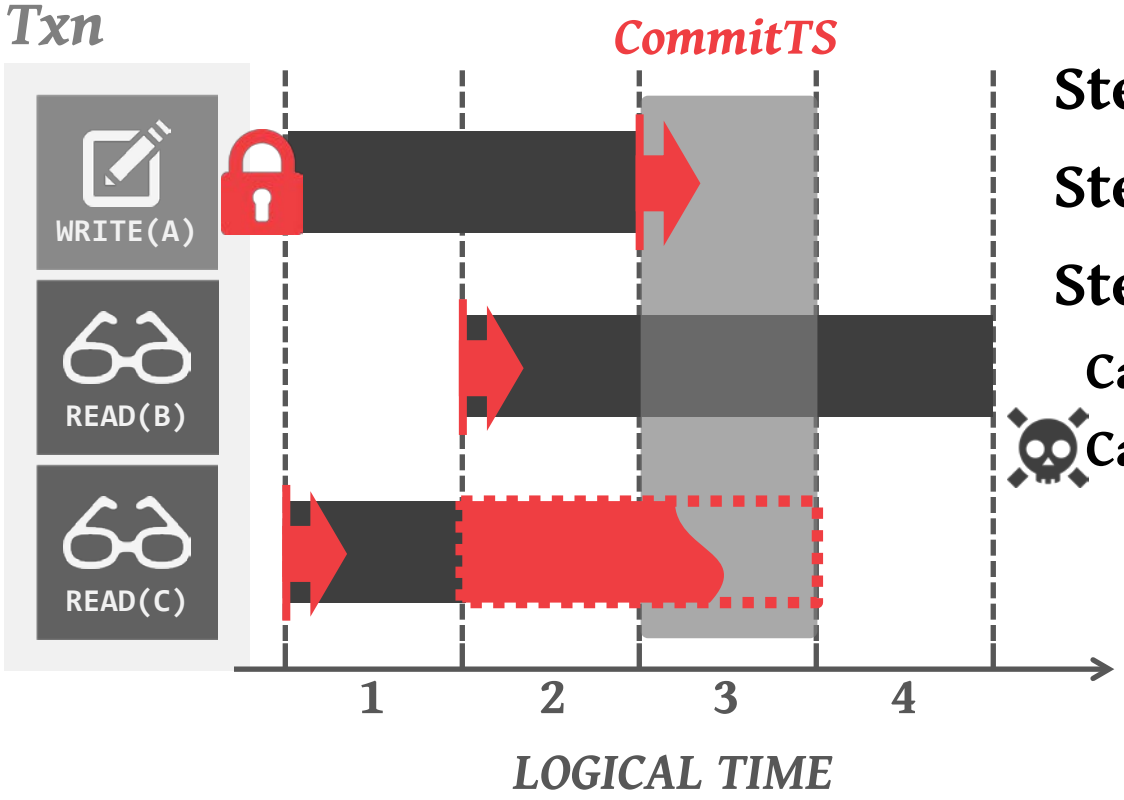
Step #2: Compute CommitTS

Step #3: Validate Read Set

Case 1: Latest Version

Case 2: Updated Before CommitTS

TICTOC: VALIDATION PHASE



Step #1: Lock Write Set

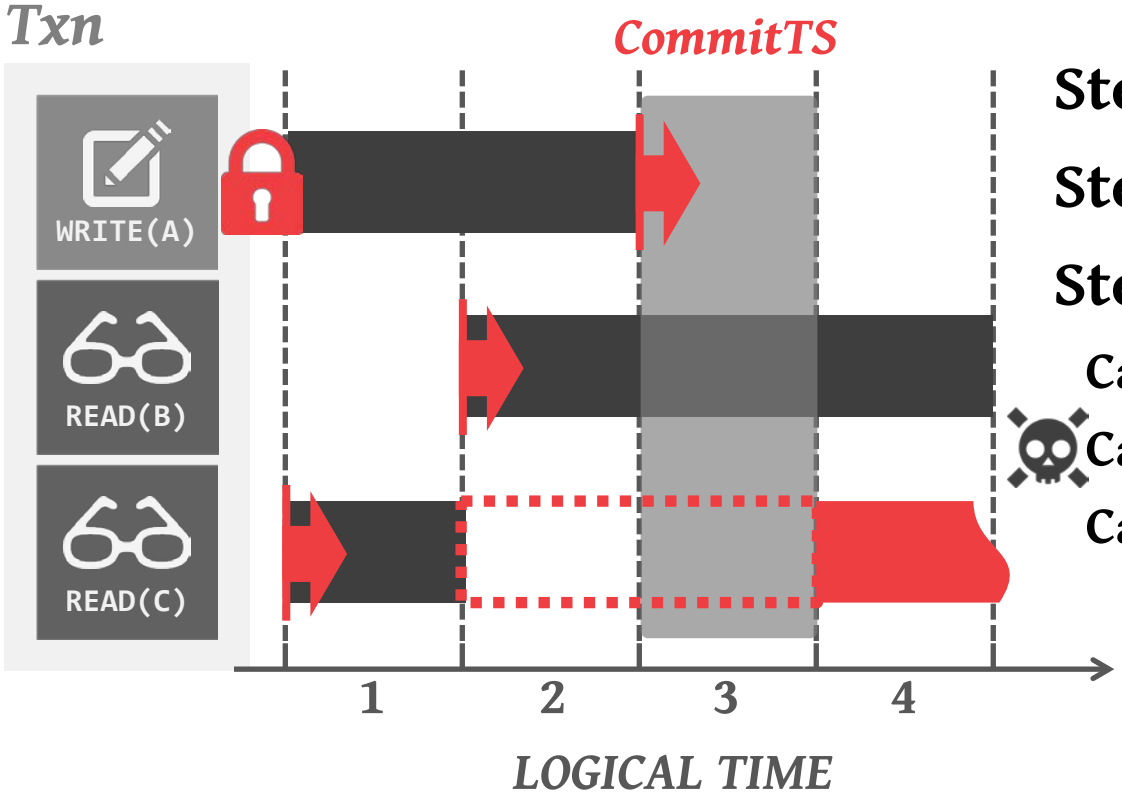
Step #2: Compute CommitTS

Step #3: Validate Read Set

Case 1: Latest Version

Case 2: Updated Before CommitTS

TICTOC: VALIDATION PHASE



Step #1: Lock Write Set

Step #2: Compute CommitTS

Step #3: Validate Read Set

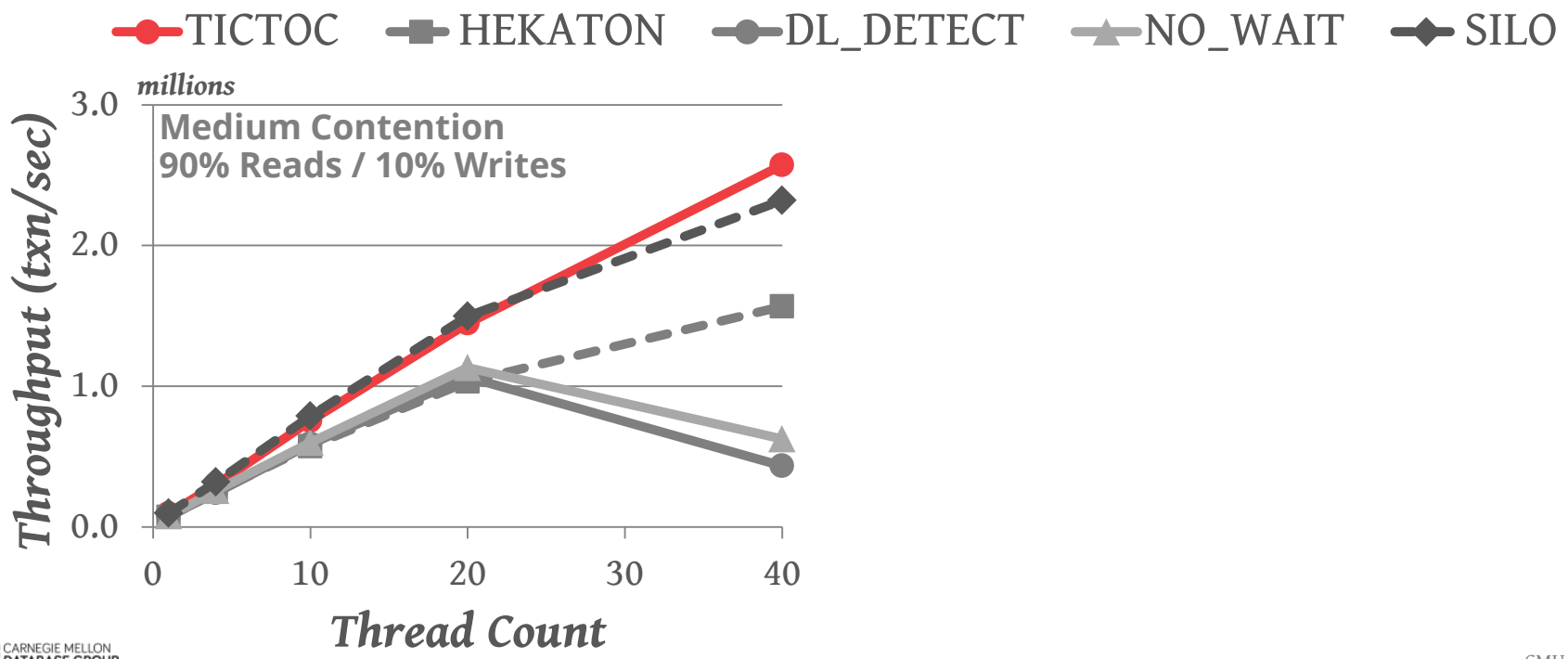
Case 1: Latest Version

Case 2: Updated Before CommitTS

Case 3: Updated After CommitTS

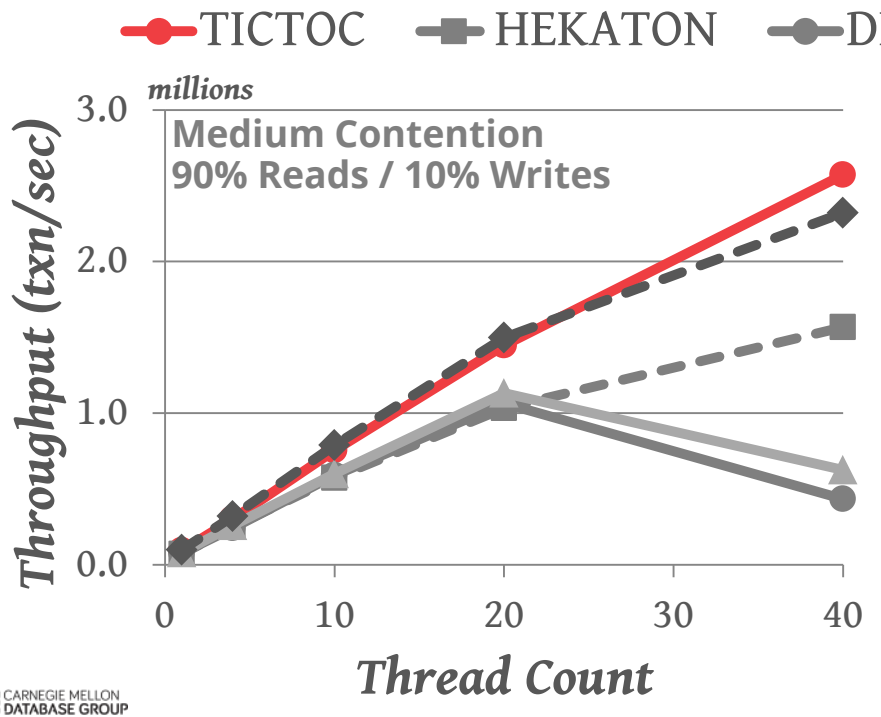
TICTOC: PERFORMANCE

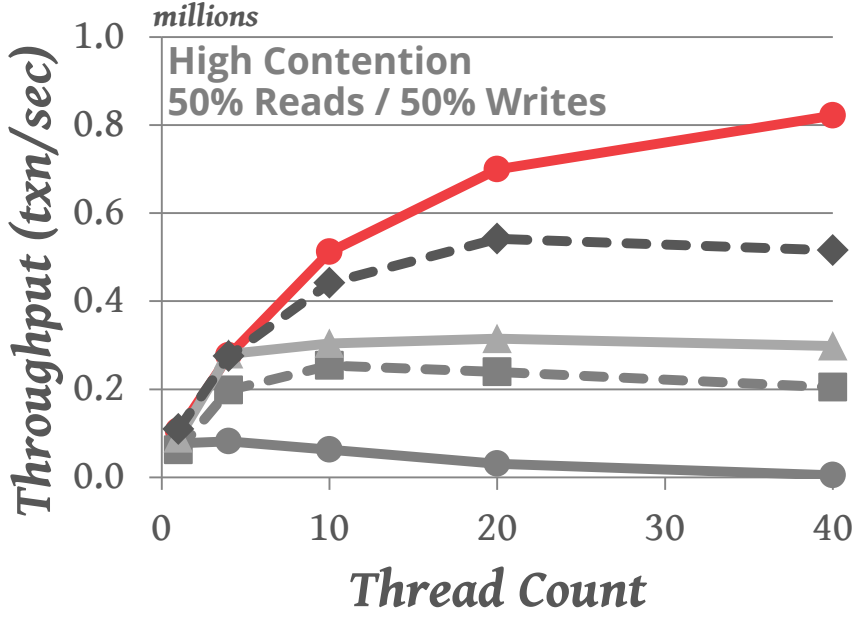
Database: 10GB YCSB
Processor: 4 sockets, 10 cores per socket



TICTOC: PERFORMANCE

Database: 10GB YCSB
Processor: 4 sockets, 10 cores per socket





PARTING THOUGHTS

OCC and MVCC are more or less equivalent

→ The difference is in when to check for conflicts and where to store in-flight data for active txns.

Trade-off between aborting txns early or later.

→ **Early**: Avoid wasted work for txns that will eventually abort, but has checking overhead.

→ **Later**: No runtime overhead but lots of wasted work under high contention.

COURSE PROJECTS

Reminder: Project #1 is due Feb 8th @ 11:59pm

We will get Autolab working this weekend.

See project specification for how to do proper debugging and logging.

I will post a form to sign up for project groups.

NEXT CLASS

Index Locking