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
Application

BEGIN
  SQL
  Program Logic
  SQL
  Program Logic
  :
  COMMIT
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

BEGIN

SQL
Program Logic

SQL
Program Logic

;
COMMIT

Parser
Planner
Optimizer
Query Execution
Application

BEGIN

SQL
Program Logic
SQL
Program Logic
;
COMMIT

Parser
Planner
Optimizer
Query Execution
CONVERSATIONAL DATABASE API

Application

BEGIN

SQL
Program Logic

SQL
Program Logic

;

COMMIT

Parser
Planner
Optimizer
Query Execution

CMU 15-721 (Spring 2016)
CONVERSATIONAL DATABASE API

Application

BEGIN
SQL
Program Logic
SQL
Program Logic
:
COMMIT
CONVERSATIONAL DATABASE API

Application

BEGIN

SQL
Program Logic

SQL
Program Logic

:

COMMIT

Parser
Planner
Optimizer
Query Execution
SOLUTIONS

Prepared Statements
→ Removes query preparation overhead.

Query Batches
→ Reduces the number of network roundtrips.

Stored Procedures
→ Removes both preparation and network stalls.
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)
Application

```
BEGIN
  SQL
  Program Logic
  SQL
  Program Logic
  ...
  COMMIT
```
STORED PROCEDURES

Application

PROC(x)

```
BEGIN
SQL
Program Logic
SQL
Program Logic
: COMMIT
```
STORED PROCEDURES

Application

CALL PROC(x=99)

PROC(x)

BEGIN
SQL
Program Logic
SQL
Program Logic
:
COMMIT
STORED PROCEDURES

Application

CALL PROC(x=99)

PROC(x)

BEGIN
SQL
Program Logic
SQL
Program Logic
:
COMMIT
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 queries 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*

```
BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT
```
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

- **BEGIN**
- **READ(A)**
- **WRITE(A)**
- **WRITE(B)**
- **COMMIT**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

```
BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT
```

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

Read Phase

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

txn #1

BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT

Record | Value | Write Timestamp
--- | --- | ---
A | 123 | 10000
B | 456 | 10000
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

BEGIN  
READ(A)  
WRITE(A)  
WRITE(B)  
COMMIT

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCE CONTROL

**Txn #1**

BEGIN
- READ(A)
- WRITE(A)
- WRITE(B)

COMMIT

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

BEGIN

READ(A)  WRITE(A)  WRITE(B)

COMMITS

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
**OPTIMISTIC CONCURRENCY CONTROL**

**Txn #1**

BEGIN

READ(A)

WRITE(A)

WRITE(B)

COMMIT

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
Txn #1

BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT

Workspace

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENcy CONTROL

Txn #1

BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT

Workspace

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

BEGIN

**READ(A)**

**WRITE(A)**

**WRITE(B)**

COMMIT

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT

Workspace

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

BEGIN
READ(A)
WRITE(A)
WRITE(B)
COMMIT

Workspace

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

**Txn #1**

BEGIN
- READ(A)
- WRITE(A)
- WRITE(B)

VALIDATE PHASE

WRITE PHASE

COMMIT

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>

CMU 15-721 (Spring 2016)
**OPTIMISTIC CONCURRENCY CONTROL**

**Txn #1**

BEGIN
- READ(A)
- WRITE(A)
- WRITE(B)

VALIDATE PHASE
- WRITE PHASE

**Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
OPTIMISTIC CONCURRENCY CONTROL

Txn #1

BEGIN
- READ(A)
- WRITE(A)
- WRITE(B)

VALIDATE PHASE

WRITE PHASE

COMMIT

Workspace

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
**OPTIMISTIC CONCURREN CY CONTROL**

**Txn #1**

- **BEGIN**
  - READ(A)
  - WRITE(A)
  - WRITE(B)

- **VALIDATE PHASE**
  - WRITE PHASE

- **Workspace**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>123</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>10000</td>
</tr>
</tbody>
</table>
### Optimistic Concurrency Control

**Txn #1**

<table>
<thead>
<tr>
<th>BEGIN</th>
<th>READ(A)</th>
<th>WRITE(A)</th>
<th>WRITE(B)</th>
<th>VALIDATE PHASE</th>
<th>WRITE PHASE</th>
<th>COMMIT</th>
</tr>
</thead>
</table>

#### Workspace

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>10001</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>10001</td>
</tr>
</tbody>
</table>
# Optimistic Concurrency Control

## Txn #1

**BEGIN**
- READ(A)

**WRITE(A)**

**WRITE(B)**

**VALIDATE PHASE**

**WRITE PHASE**

**COMMIT**

<table>
<thead>
<tr>
<th>Record</th>
<th>Value</th>
<th>Write Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>888</td>
<td>10001</td>
</tr>
<tr>
<td>B</td>
<td>999</td>
<td>10001</td>
</tr>
</tbody>
</table>
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.

![Diagram showing the timeline of transactions](image)
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.

TIME

COMMIT

COMMIT

COMMIT

Txn #1

Txn #2

Txn #3
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.

TIME

Txn #1

Txn #2

Txn #3
FORWARD VALIDATION

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

- **Txn #1**
- **Txn #2**
- **Txn #3**

TIME
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.
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

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-###-#</td>
<td>X</td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#-###-#</td>
<td>X</td>
<td>Tupac</td>
<td>$999</td>
</tr>
<tr>
<td>#-###-#</td>
<td>X</td>
<td>Wiz</td>
<td>$67</td>
</tr>
<tr>
<td>#-###-#</td>
<td>X</td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
</tbody>
</table>
### SILO: COMMIT PROTOCOL

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-#####-#</td>
<td>☐</td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#-#####-#</td>
<td>☐</td>
<td>Tupac</td>
<td>$999</td>
</tr>
<tr>
<td>#-#####-#</td>
<td>☐</td>
<td>Wiz</td>
<td>$67</td>
</tr>
<tr>
<td>#-#####-#</td>
<td>☐</td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
</tbody>
</table>

**EPOCH | BATCH TIMESTAMP | LOCKS**

---

CMU 15-721 (Spring 2016)
## SILO: COMMIT PROTOCOL

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-###-#</td>
<td>☐</td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#-###-#</td>
<td>☐</td>
<td>Tupac</td>
<td>$999</td>
</tr>
<tr>
<td>#-###-#</td>
<td>☐</td>
<td>Wiz</td>
<td>$67</td>
</tr>
<tr>
<td>#-###-#</td>
<td>☐</td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
</tbody>
</table>
SILO: COMMIT PROTOCOL

Workspace

Read Set

TID Word | POINTER | ATTR1 | ATTR2
---|---|---|---
#-###-# | O.D.B. | $13 |
#-###-# | Tupac | $999 |

Write Set

TID Word | POINTER | ATTR1 | ATTR2
---|---|---|---
#-###-# | Tupac | $777 |
#-###-# | O.D.B. | $13 |
SILO: COMMIT PROTOCOL

Workspace

Read Set

<table>
<thead>
<tr>
<th>#-###-#</th>
<th>O.D.B.</th>
<th>$13</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-###-#</td>
<td>Tupac</td>
<td>$999</td>
</tr>
</tbody>
</table>

Write Set

| Tupac   | $777   |

Step #1: Lock Write Set

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-###-#</td>
<td>O.D.B.</td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#-###-#</td>
<td>Tupac</td>
<td>$999</td>
<td></td>
</tr>
<tr>
<td>#-###-#</td>
<td>Wiz</td>
<td>$67</td>
<td></td>
</tr>
<tr>
<td>#-###-#</td>
<td>O.D.B.</td>
<td>$13</td>
<td></td>
</tr>
</tbody>
</table>
SILO: COMMIT PROTOCOL

**Step #1: Lock Write Set**

**Workspace**

<table>
<thead>
<tr>
<th>Read Set</th>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-###-#</td>
<td>O.D.B.</td>
<td></td>
<td>$13</td>
<td></td>
</tr>
<tr>
<td>#-###-#</td>
<td>Tupac</td>
<td></td>
<td>$999</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write Set</th>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tupac</td>
<td>#</td>
<td>O.D.B.</td>
<td>$13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>O.D.B.</td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#</td>
<td>Tupac</td>
<td></td>
<td>$999</td>
</tr>
<tr>
<td>#</td>
<td>Wiz</td>
<td></td>
<td>$67</td>
</tr>
<tr>
<td>#</td>
<td>O.D.B.</td>
<td></td>
<td>$13</td>
</tr>
</tbody>
</table>
SILO: COMMIT PROTOCOL

Workspace

Read Set

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>backpage</td>
<td>O.D.B.</td>
<td>$13</td>
<td></td>
</tr>
<tr>
<td>backpage</td>
<td>Tupac</td>
<td>$999</td>
<td></td>
</tr>
</tbody>
</table>

Write Set

| Pointer | Tupac | $777 |

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 |

Step #1: Lock Write Set

Step #2: Examine Read Set

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-#####-#</td>
<td>O.D.B.</td>
<td>$13</td>
<td></td>
</tr>
<tr>
<td>#-#####-#</td>
<td>Tupac</td>
<td>$999</td>
<td></td>
</tr>
<tr>
<td>#-#####-#</td>
<td>Wiz</td>
<td>$67</td>
<td></td>
</tr>
<tr>
<td>#-#####-#</td>
<td>O.D.B.</td>
<td>$13</td>
<td></td>
</tr>
</tbody>
</table>
**SILO: COMMIT PROTOCOL**

### Workspace

<table>
<thead>
<tr>
<th>Read Set</th>
<th>Write Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-####-#</td>
<td>Tupac</td>
</tr>
<tr>
<td></td>
<td>$777</td>
</tr>
<tr>
<td>O.D.B.</td>
<td>$13</td>
</tr>
<tr>
<td>Tupac</td>
<td>$999</td>
</tr>
</tbody>
</table>

### Step #1: Lock Write Set

### Step #2: Examine Read Set

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-####-#</td>
<td></td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#-####-#</td>
<td></td>
<td>Tupac</td>
<td>$999</td>
</tr>
<tr>
<td>#-####-#</td>
<td></td>
<td>Wiz</td>
<td>$67</td>
</tr>
<tr>
<td>#-####-#</td>
<td></td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
</tbody>
</table>
Step #1: Lock Write Set
Step #2: Examine Read Set
### SILO: COMMIT PROTOCOL

**Workspace**

<table>
<thead>
<tr>
<th>Read Set</th>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-####-#</td>
<td>O.D.B.</td>
<td>$13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#-####-#</td>
<td>Tupac</td>
<td>$999</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write Set</th>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tupac</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$777</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Step #1:** Lock Write Set

**Step #2:** Examine Read Set
**Workspace**

**Read Set**
- `-###-#` O.D.B. $13
- `-###-#` Tupac $999

**Write Set**
- Tupac $777

---

**Step #1: Lock Write Set**

**Step #2: Examine Read Set**

**SILO: COMMIT PROTOCOL**

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-###-#</code></td>
<td></td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td><code>-###-#</code></td>
<td></td>
<td>Tupac</td>
<td>$999</td>
</tr>
<tr>
<td><code>-###-#</code></td>
<td></td>
<td>Wiz</td>
<td>$67</td>
</tr>
<tr>
<td><code>-###-#</code></td>
<td></td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
</tbody>
</table>
SILO: COMMIT PROTOCOL

Workspace

<table>
<thead>
<tr>
<th>Read Set</th>
<th>Write Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-#####-#</td>
<td>O.D.B. $13</td>
</tr>
<tr>
<td>#-#####-#</td>
<td>Tupac $999</td>
</tr>
<tr>
<td>Tupac $777</td>
<td></td>
</tr>
</tbody>
</table>

- Step #1: Lock Write Set
- Step #2: Examine Read Set
SILO: COMMIT PROTOCOL

Workspace

<table>
<thead>
<tr>
<th>Read Set</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#-####-#</td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
<tr>
<td>#-####-#</td>
<td>Tupac</td>
<td>$999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write Set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tupac</td>
<td>$777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID Word</th>
<th>POINTER</th>
<th>ATTR1</th>
<th>ATTR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#-####-#</td>
<td>o</td>
<td>John</td>
<td>$100</td>
</tr>
<tr>
<td>#-####-#</td>
<td>o</td>
<td>Tupac</td>
<td>$999</td>
</tr>
<tr>
<td>#-####-#</td>
<td>o</td>
<td>Wiz</td>
<td>$67</td>
</tr>
<tr>
<td>#-####-#</td>
<td>o</td>
<td>O.D.B.</td>
<td>$13</td>
</tr>
</tbody>
</table>

Step #1: Lock Write Set
Step #2: Examine Read Set
Step #3: Install Write Set
### Step #1: Lock Write Set

### Step #2: Examine Read Set

### Step #3: Install Write Set
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
SILO: PERFORMANCE

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

Source: Eddie Kohler
CMU 15-721 (Spring 2016)
TICTOC

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

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

Txn

WRITE(A)
READ(B)
READ(C)

LOGICAL TIME

1 2 3 4
TICTOC: VALIDATION PHASE

Txn

WRITE(A)
READ(B)
READ(C)

LOGICAL TIME

1 2 3 4
TICTOC: VALIDATION PHASE

Txn

WRITE(A)

READ(B)

READ(C)

W-TS

R-TS

LOGICAL TIME

1 2 3 4

A B C
TICTOC: VALIDATION PHASE

**Txn**

- WRITE(A)
- READ(B)
- READ(C)

LOGICAL TIME

1 2 3 4
TICTOC: VALIDATION PHASE

**Step #1: Lock Write Set**
TICTOC: VALIDATION PHASE

Txn

Step #1: Lock Write Set
Step #2: Compute CommitTS
**TICTOC: VALIDATION PHASE**

**Step #1:** Lock Write Set

**Step #2:** Compute CommitTS
TICTOC: VALIDATION PHASE

**Txn**

- **WRITE(A)**
- **READ(B)**
- **READ(C)**

**Logical Time**

1. **Step #1:** Lock Write Set
2. **Step #2:** Compute CommitTS
3. **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

**Txn**

- **WRITE(A)**: Step #1: Lock Write Set
- **READ(B)**: Step #2: Compute CommitTS
- **READ(C)**: Step #3: Validate Read Set

**LOGICAL TIME**

1. Lock Write Set
2. Compute CommitTS
3. Validate Read Set

CommitTS
**TICTOC: VALIDATION PHASE**

**Step #1:** Lock Write Set

**Step #2:** Compute CommitTS

**Step #3:** Validate Read Set

---

**LOGICAL TIME**

- **1:** WRITE(A)
- **2:** READ(B)
- **3:** ???
- **4:** READ(C)
**TICTOC: VALIDATION PHASE**

**Txn**

1. **Step #1:** Lock Write Set
2. **Step #2:** Compute CommitTS
3. **Step #3:** Validate Read Set

**LOGICAL TIME**

- **WRITE(A)**
- **READ(B)**
- **READ(C)**

CommitTS
**TICTOC: VALIDATION PHASE**

**Txn**

- **WRITE(A)**
- **READ(B)**
- **READ(C)**

**CommitTS**

**LOGICAL TIME**

1 2 3 4

**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

**Txn**

- WRITE(A)
- READ(B)
- READ(C)

**CommitTS**

1. **Step #1:** Lock Write Set
2. **Step #2:** Compute CommitTS
3. **Step #3:** Validate Read Set

- **Case 1:** Latest Version
- **Case 2:** Updated Before CommitTS

**LOGICAL TIME**

1 2 3 4
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

---

**LOGICAL TIME**

1 2 3 4

**Txn**

WRITE(A)

READ(B)

READ(C)
TICTOC: PERFORMANCE

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

Throughput (txn/sec)

Thread Count

Medium Contention
90% Reads / 10% Writes

millions

TICTOC  HEKATON  DL_DETECT  NO_WAIT  SILO
TICTOC: PERFORMANCE

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

Throughput (txn/sec) vs Thread Count

- TICTOC
- HEKATON
- DL_DETECT
- NO_WAIT
- SILO

Medium Contention
90% Reads / 10% Writes

High Contention
50% Reads / 50% Writes
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.
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