Georgia Tech

Lecture 16: Multi-Version Concurrency Control

CREATING THE NEXT®

Today's Agenda

Recap

Optimistic Concurrency Control

- The DBMS creates a private workspace for each txn.
  - Any object read is copied into workspace.
  - Modifications are applied to workspace.
- When a txn commits, the DBMS compares workspace **write set** to see whether it conflicts with other txns.
- If there are no conflicts, the write set is installed into the **global database**.

OCC Phases

- **Phase 1 – Read**:
  - Track the read/write sets of txns and store their writes in a private workspace.
- **Phase 2 – Validation**:
  - When a txn commits, check whether it conflicts with other txns.
- **Phase 3 – Write**:
  - If validation succeeds, apply private changes to database. Otherwise abort and restart the txn.

Today's Agenda

- Multi-Version Concurrency Control
- Design Decisions
  - Concurrency Control Protocol
  - Version Storage
  - Garbage Collection
  - Index Management

Multi-Version Concurrency Control

Multi-Version Concurrency Control

- The DBMS maintains multiple **physical versions** of a single **logical object** in the database:
  - When a txn writes to an object, the DBMS creates a new version of that object (instead of private workspace in OCC)
  - When a txn reads an object, it reads the newest version that existed when the txn started.

MVCC HISTORY

- Protocol was first proposed in 1978 MIT PhD dissertation.
- First implementations was Rdb/VMS and InterBase at DEC in early 1980s.
  - Both were by Jim Starkey, co-founder of NuoDB.
  - DEC Rdb/VMS is now ”Oracle Rdb”
  - InterBase was open-sourced as Firebird.

Multi-Version Concurrency Control

- Writers don't block readers. Readers don't block writers.
- Read-only txns can read a **consistent snapshot** without acquiring locks.
  - Use timestamps to determine visibility.
- Easily support **time-travel queries**.

MVCC – Example 1

TIME

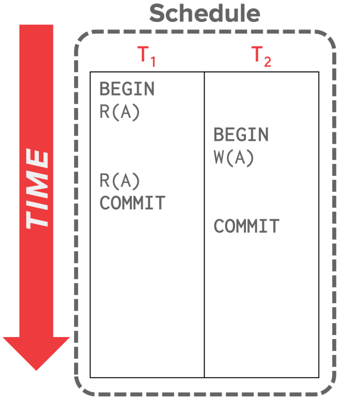
Schedule

T_1	T_2
BEGIN R(A)	
	BEGIN W(A)
R(A) COMMIT	
	COMMIT

Database

Version	Value	Begin	End
A_0	123	0	-

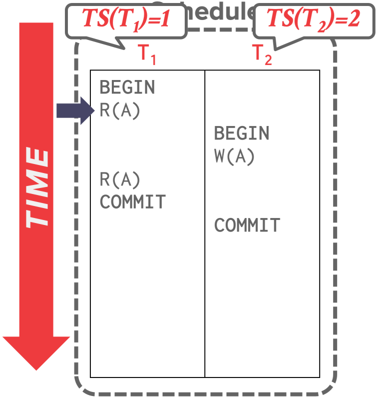
MVCC – Example 1



Database

Version	Value	Begin	End
A_0	123	0	-

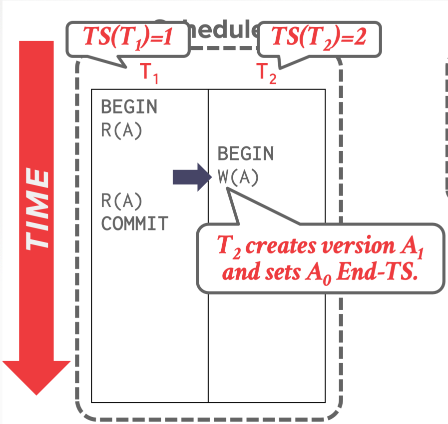
MVCC – Example 1



Database

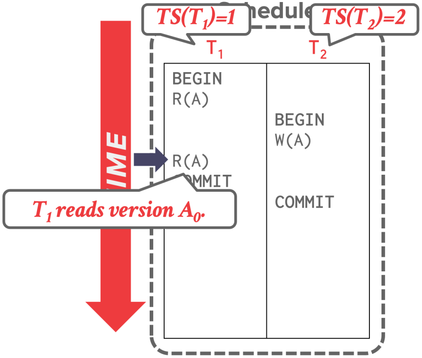
Version	Value	Begin	End
A_0	123	0	-

MVCC – Example 1



Version	Value	Begin	End
A_0	123	0	-
A_1	456	2	-

MVCC – Example 1



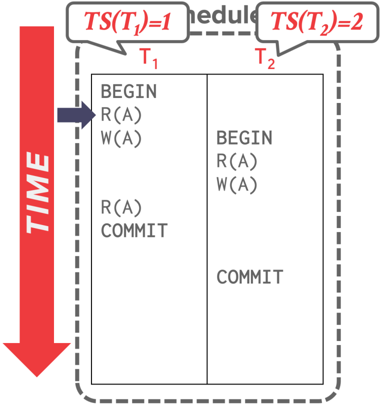
Database

Version	Value	Begin	End
A_0	123	0	2
A_1	456	2	-

Txn Status Table

TxnId	Timestamp	Status
T_1	1	Active
T_2	2	Active

MVCC – Example 2



Database

Version	Value	Begin	End
A_0	123	0	

Txn Status Table

TxnId	Timestamp	Status
T_1	1	Active

MVCC – Example 2

TIME

Schedule

$TS(T_1)=1$ $TS(T_2)=2$

T_1	T_2
BEGIN	
R(A)	
W(A)	
	BEGIN
	R(A)
	W(A)
R(A)	
COMMIT	
	COMMIT

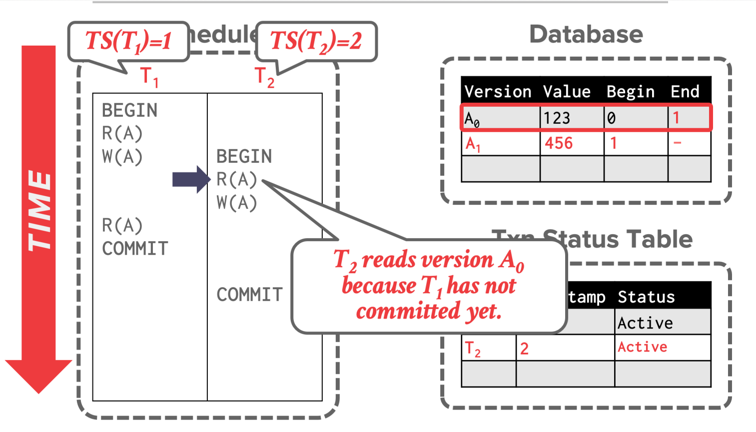
Database

Version	Value	Begin	End
A_0	123	0	1
A_1	456	1	-

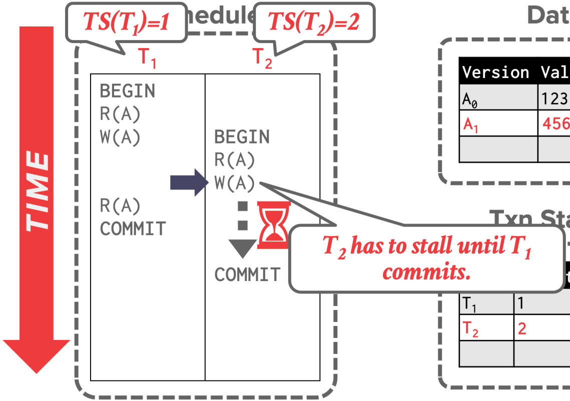
Txn Status Table

TxnId	Timestamp	Status
T_1	1	Active

MVCC – Example 2



MVCC – Example 2



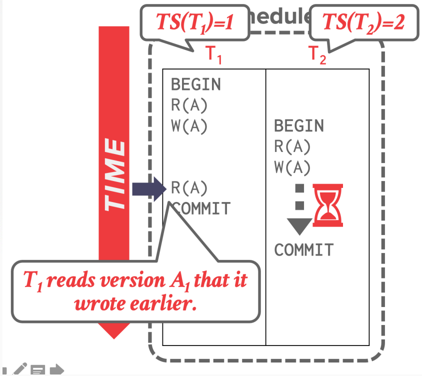
Database

Version	Value	Begin	End
A_0	123	0	1
A_1	456	1	-

Txn Status Table

	tamp	Status
T_1	1	Active
T_2	2	Active

MVCC – Example 2



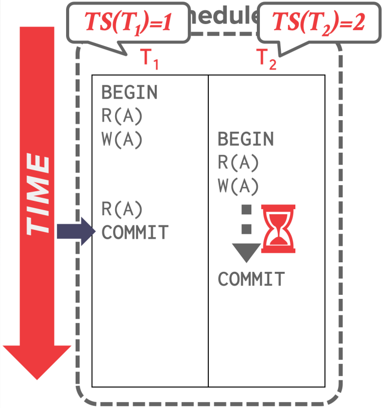
Database

Version	Value	Begin	End
A_0	123	0	1
A_1	456	1	-

Txn Status Table

TxnId	Timestamp	Status
T_1	1	Active
T_2	2	Active

MVCC – Example 2



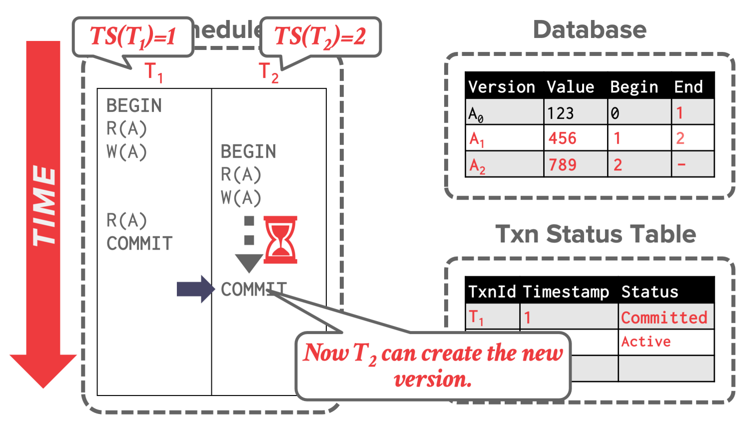
Database

Version	Value	Begin	End
A_0	123	0	1
A_1	456	1	-

Txn Status Table

TxnId	Timestamp	Status
T_1	1	Committed
T_2	2	Active

MVCC – Example 2



Multi-Version Concurrency Control

- MVCC is more than just a Concurrency Control protocol.
- It completely affects how the DBMS manages transactions and the database.
- **Examples:** Oracle, SAP HANA, PostgreSQL, CockroachDB

MVCC Design Decisions

- Concurrency Control Protocol
- Version Storage
- Garbage Collection
- Index Management

Concurrency Control Protocol

Concurrency Control Protocol

- **Approach 1:** Timestamp Ordering
  - Assign txns timestamps that determine serial order.
- **Approach 2:** Optimistic Concurrency Control
  - Three-phase protocol from last class.
  - Use private workspace for new versions.
- **Approach 3:** Two-Phase Locking
  - Txns acquire appropriate lock on physical version before they can read/write a logical tuple.

Version Storage

Version Storage

- The DBMS uses the tuples' pointer field to create a **version chain** per logical tuple.
  - This allows the DBMS to find the version that is visible to a particular txn at runtime.
  - Indexes always point to the **head** of the chain.
- Different storage schemes determine where/what to store for each version.

Version Storage

- **Approach 1:** Append-Only Storage
  - New versions are appended to the same table space.
- **Approach 2:** Time-Travel Storage
  - Old versions are copied to separate table space.
- **Approach 3:** Delta Storage
  - The original values of the modified attributes are copied into a separate delta record space.

Append-Only Storage

- All of the physical versions of a logical tuple are stored in the same table space. The versions are mixed together.
- On every update, append a new version of the tuple into an empty space in the table.

Main Table

VERSION	VALUE	POINTER
A_0	$111	•
A_1	$222	Ø
B_1	$10	Ø

Georgia Tech

Append-Only Storage

- All of the physical versions of a logical tuple are stored in the same table space. The versions are mixed together.
- On every update, append a new version of the tuple into an empty space in the table.

Main Table

VERSION	VALUE	POINTER
A_0	\$111	
A_1	\$222	Ø
B_1	\$10	Ø
A_2	\$333	Ø

Append-Only Storage

- All of the physical versions of a logical tuple are stored in the same table space. The versions are mixed together.
- On every update, append a new version of the tuple into an empty space in the table.

Main Table

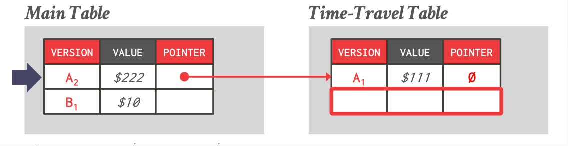
VERSION	VALUE	POINTER
A_0	$111	●
A_1	$222	●
B_1	$10	Ø
A_2	$333	Ø

Version Chain Ordering

- **Approach 1:** Oldest-to-Newest (O2N)
  - Just append new version to end of the chain.
  - Have to traverse chain on look-ups.
- **Approach 2:** Newest-to-Oldest (N2O)
  - Have to update index pointers for every new version.
  - Don't have to traverse chain on look ups.

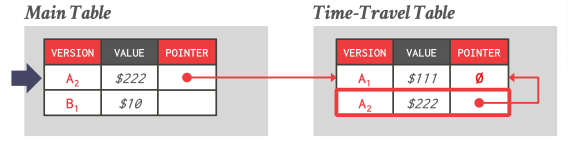
Time-Travel Storage

- On every update, copy the current version to the time-travel table. Update pointers.
- Overwrite master version in the main table. Update pointers.



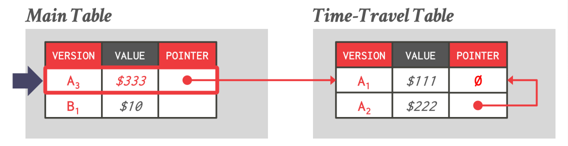
Time-Travel Storage

- On every update, copy the current version to the time-travel table. Update pointers.
- Overwrite master version in the main table. Update pointers.



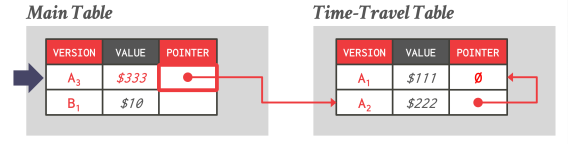
Time-Travel Storage

- On every update, copy the current version to the time-travel table. Update pointers.
- Overwrite master version in the main table. Update pointers.



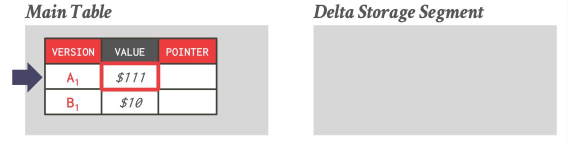
Time-Travel Storage

- On every update, copy the current version to the time-travel table. Update pointers.
- Overwrite master version in the main table. Update pointers.



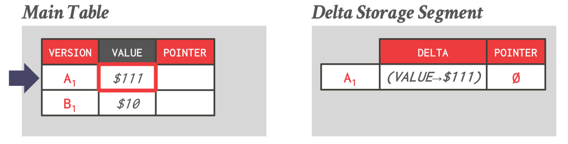
Delta Storage

- On every update, copy only the values that were modified to the delta storage and overwrite the master version.
- Txns can recreate old versions by applying the delta in reverse order.



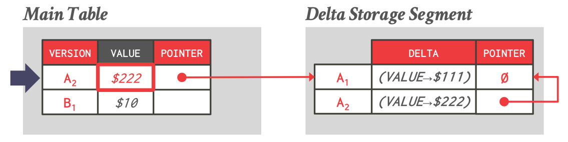
Delta Storage

- On every update, copy only the values that were modified to the delta storage and overwrite the master version.
- Txns can recreate old versions by applying the delta in reverse order.



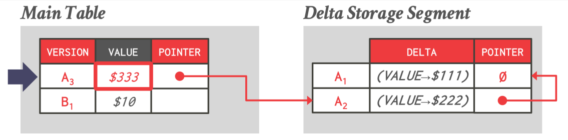
Delta Storage

- On every update, copy only the values that were modified to the delta storage and overwrite the master version.
- Txns can recreate old versions by applying the delta in reverse order.



Delta Storage

- On every update, copy only the values that were modified to the delta storage and overwrite the master version.
- Txns can recreate old versions by applying the delta in reverse order.



Garbage Collection

Garbage Collection

- The DBMS needs to remove **reclaimable** physical versions from the database over time.
  - No active txn in the DBMS can **see** that version (SI).
  - The version was created by an aborted txn.
- Two additional design decisions:
  - How to look for expired versions?
  - How to decide when it is safe to reclaim memory?

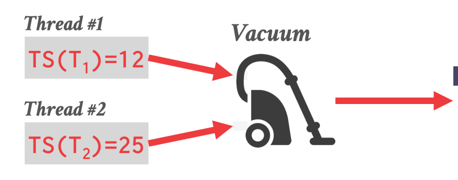
Garbage Collection

- **Approach 1:** Tuple-level
  - Find old versions by examining tuples directly.
  - **Background Vacuuming** vs. **Cooperative Cleaning**
- **Approach 2:** Transaction-level
  - Txns keep track of their old versions so the DBMS does not have to scan tuples to determine visibility.

Tuple-level GC

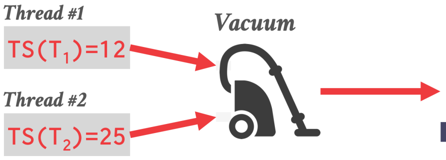
- **Background Vacuuming:**
- Separate thread(s) periodically scan the table and look for reclaimable versions.
- Works with any storage.

Tuple-level GC



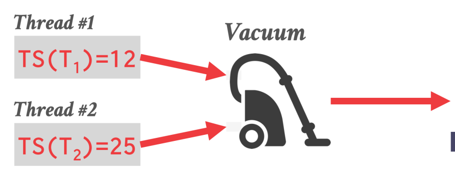
VERSION	BEGIN	END
A_{100}	1	9
B_{100}	1	9
B_{101}	10	20

Tuple-level GC



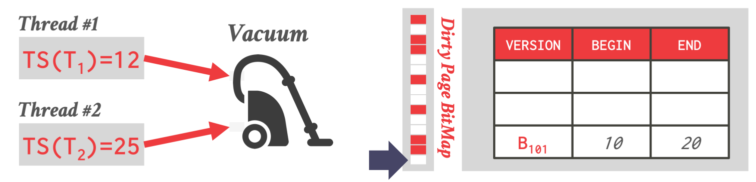
VERSION	BEGIN	END
A_{100}	1	9
B_{100}	1	9
B_{101}	10	20

Tuple-level GC



VERSION	BEGIN	END
B_{101}	10	20

Tuple-level GC



Tuple-level GC

- **Cooperative Cleaning:**
- Worker threads identify reclaimable versions as they traverse version chain.
- Only works with O2N.

Tuple-level GC

Tuple-level GC

Tuple-level GC

Tuple-level GC

Transaction-level GC

- Each txn keeps track of its read/write set.
- The DBMS determines when all versions created by a finished txn are no longer visible.

Index Management

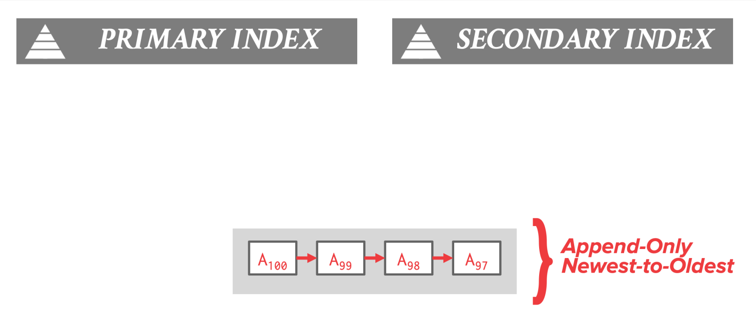
Index Management

- Primary key indexes point to version chain head.
  - How often the DBMS has to update the pkey index depends on whether the system creates new versions when a tuple is updated.
  - If a txn updates a tuple's pkey attribute(s), then this is treated as an DELETE followed by an INSERT.
- Secondary indexes are more complicated...

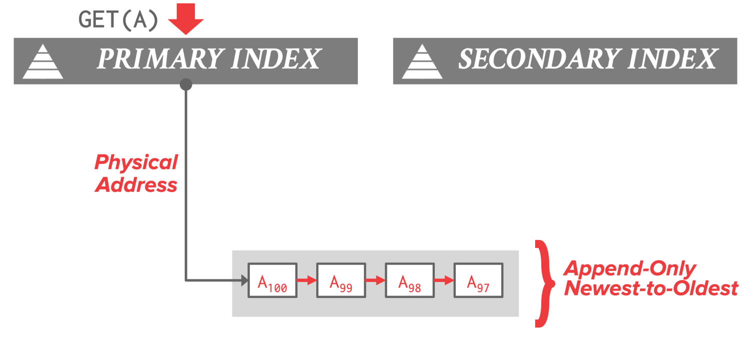
Secondary Indexes

- **Approach 1:** Physical Pointers
  - Use the physical address to the version chain head.
- **Approach 2:** Logical Pointers
  - Use a fixed identifier per tuple that does not change.
  - Requires an extra indirection layer.
  - Primary Key vs. Tuple Id

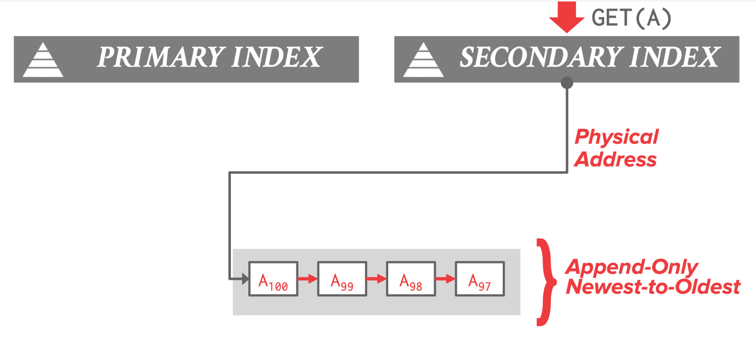
Physical Pointers



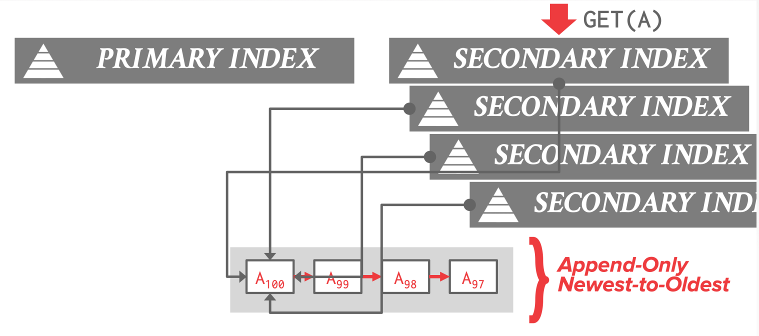
Physical Pointers



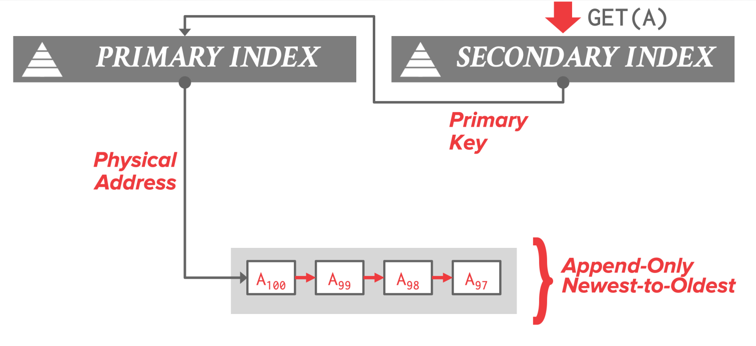
Physical Pointers



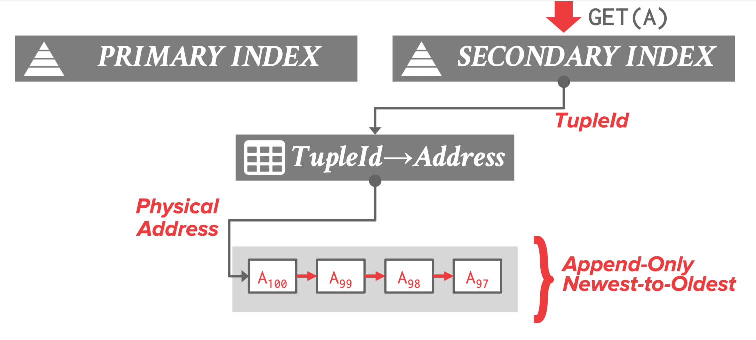
Physical Pointers



Logical Pointers



Logical Pointers



MVCC Implementations

DBMS	Protocol	Version Storage	Garbage Collection	Indexes
Oracle	MV2PL	Delta	Vacuum	Logical
Postgres	MV-2PL/MV-TO	Append-Only	Vacuum	Physical
MySQL-InnoDB	MV-2PL	Delta	Vacuum	Logical
HYRISE	MV-OCC	Append-Only	–	Physical
Hekaton	MV-OCC	Append-Only	Cooperative	Physical
MemSQL	MV-OCC	Append-Only	Vacuum	Physical
SAP HANA	MV-2PL	Time-travel	Hybrid	Logical
NuoDB	MV-2PL	Append-Only	Vacuum	Logical
HyPer	MV-OCC	Delta	Txn-level	Logical

Conclusion

- MVCC is the widely used scheme in DBMSs.
- Even systems that do not support multi-statement txns (*e.g.*, NoSQL) use it.

Next Class

- Advanced topics in Concurrency Control