Carnegie Mellon University

DATABASE SYSTEMS

Log-Structured Storage

LECTURE #05 » 15-445/645 FALL 2025 » PROF. ANDY PAVLO

ADMINISTRIVIA

Homework #2 is due Sunday Sept 21st @ 11:59pm

Project #1 is due Sunday Sept 29th @ 11:59pm

UPCOMING DATABASE EVENTS

CMU-DB Industry Affiliates Visit Day

→ Monday Sept 15th: Research Talks + Poster Session
→ Tuesday Sept 16th: Company Info Sessions
→ All events are open to the public.

Carnegie Mellon University
Database Group
Industry Affiliates

Sign-up for Company Info Sessions (**@54**)

Add your Resume if You Want a Database Job (**@55**)

LAST CLASS

We introduced the buffer pool manager as the location of where the DBMS stores copies of database pages it retrieves from non-volatile storage.

TODAY'S AGENDA

Buffer Pool Optimizations
Tuple-Oriented Storage
Index-Organized Storage
Log-Structured Storage
⚡**DB Flash Talk: SingleStore**

BUFFER POOL OPTIMIZATIONS

Multiple Buffer Pools

Pre-Fetching

Scan Sharing

MULTIPLE BUFFER POOLS

The DBMS does not always have a single buffer pool for the entire system.

→ Multiple buffer pool instances
→ Per-database buffer pool
→ Per-page type buffer pool

Partitioning memory across multiple pools helps reduce latch contention and improve locality.

→ Avoids contention on LRU tracking meta-data.

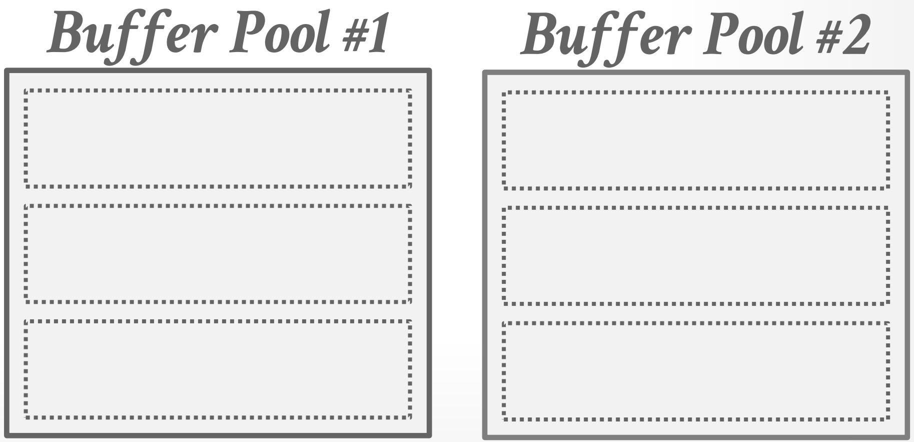
MULTIPLE BUFFER POOLS

Approach #1: Object Id

→ Embed an object identifier in record ids and then maintain a mapping from objects to specific buffer pools.

Q1 `GET RECORD #123`

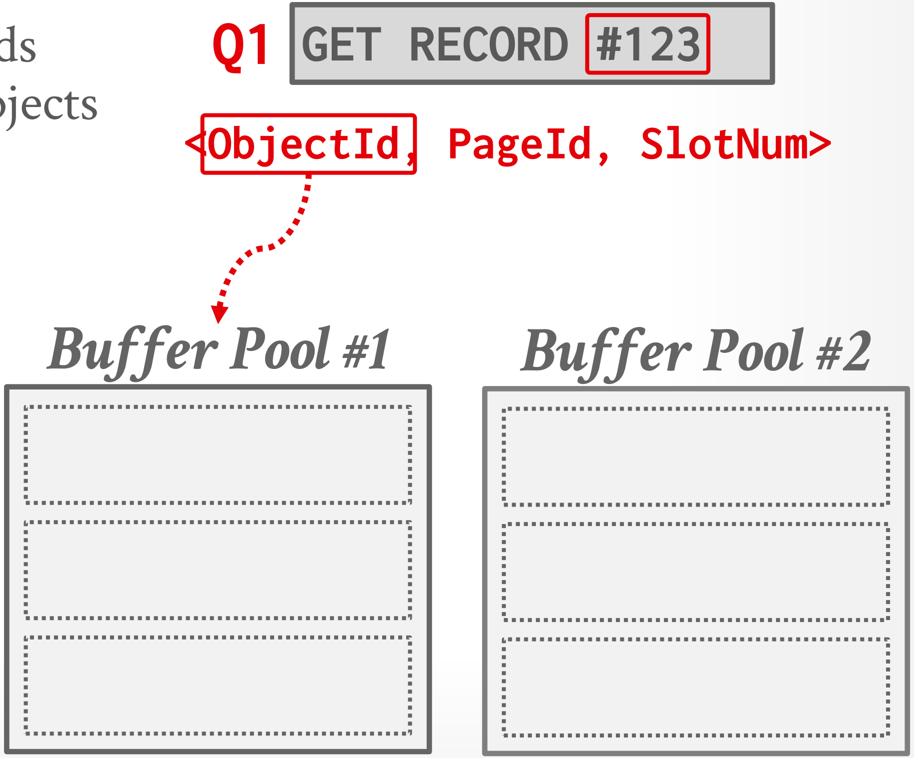
`<ObjectId, PageId, SlotNum>`



MULTIPLE BUFFER POOLS

Approach #1: Object Id

→ Embed an object identifier in record ids and then maintain a mapping from objects to specific buffer pools.



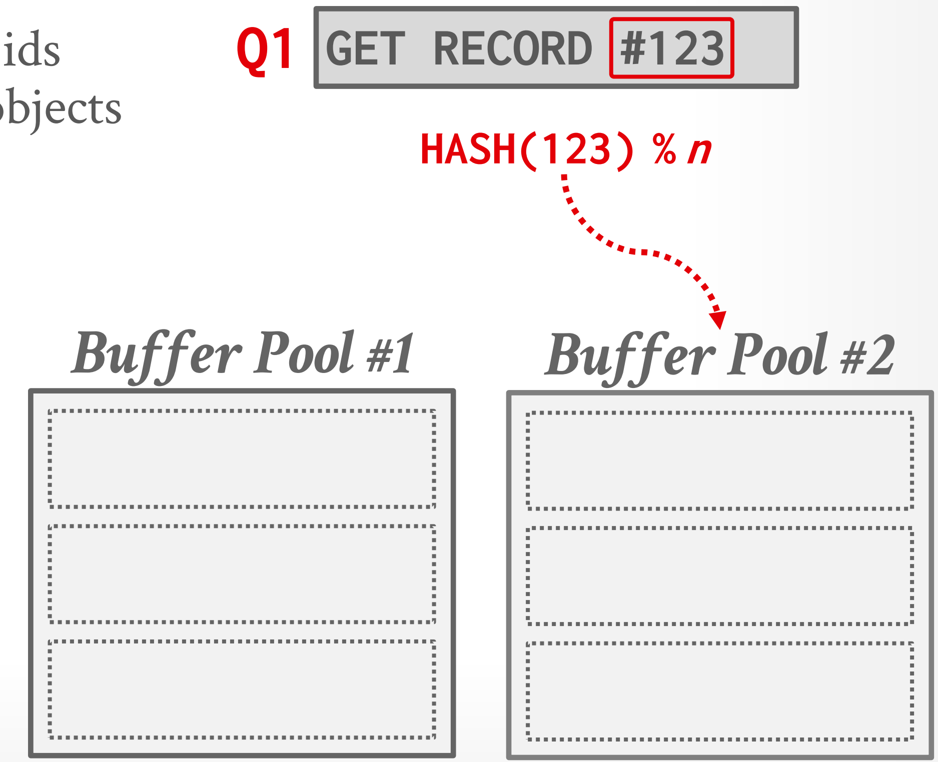
MULTIPLE BUFFER POOLS

Approach #1: Object Id

→ Embed an object identifier in record ids and then maintain a mapping from objects to specific buffer pools.

Approach #2: Hashing

→ Hash the page id to select which buffer pool to access.

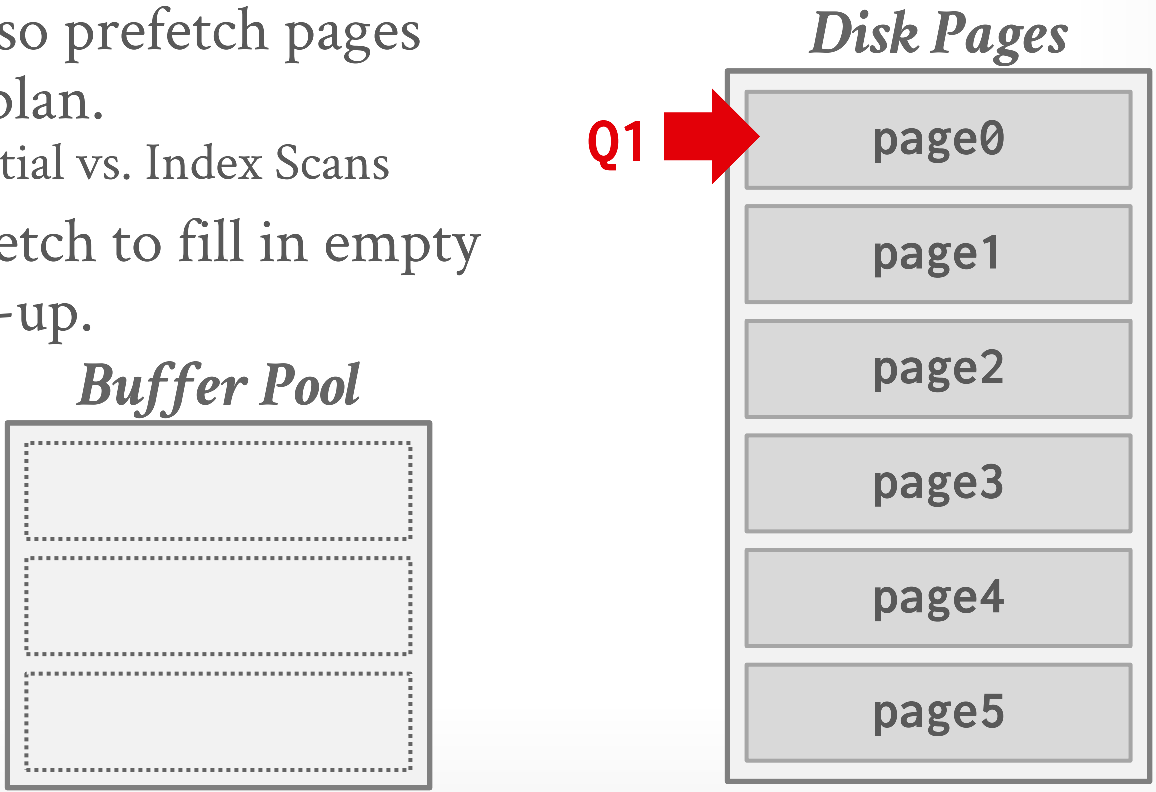


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

→ Examples: Sequential vs. Index Scans

Some DBMS prefetch to fill in empty frames upon start-up.

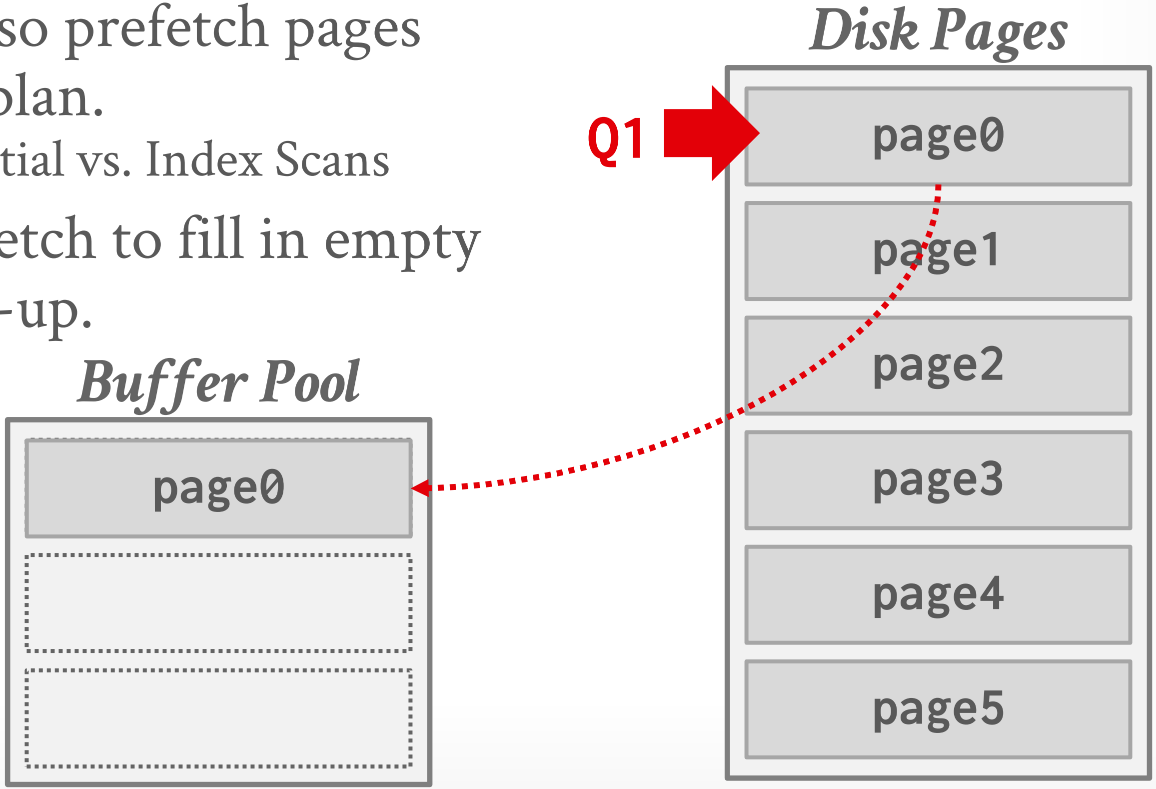


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

→ Examples: Sequential vs. Index Scans

Some DBMS prefetch to fill in empty frames upon start-up.

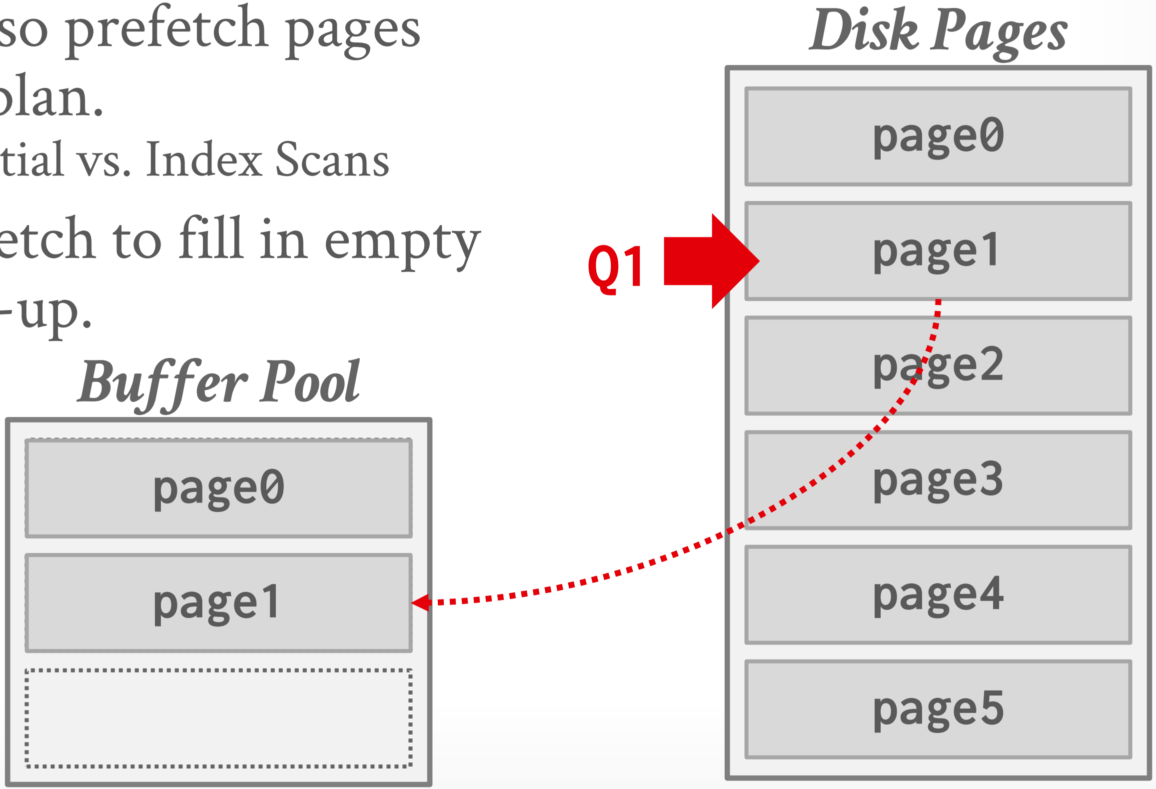


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

→ Examples: Sequential vs. Index Scans

Some DBMS prefetch to fill in empty frames upon start-up.

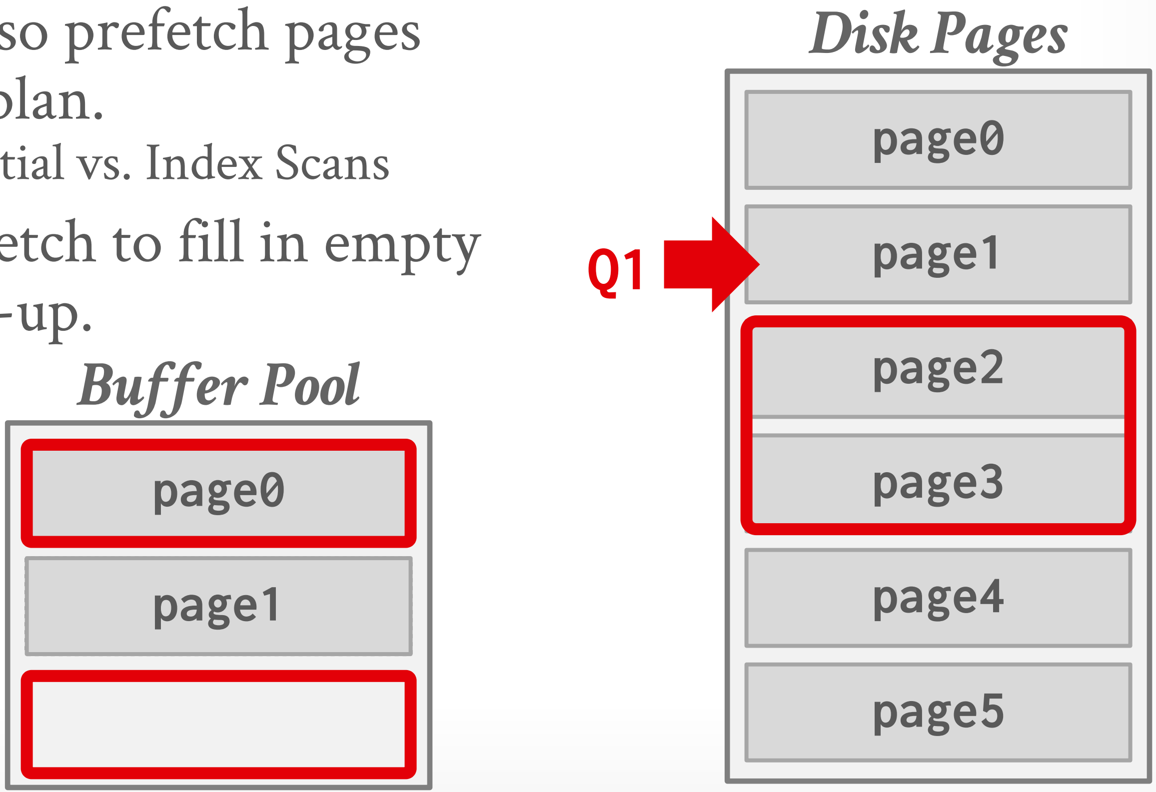


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

→ Examples: Sequential vs. Index Scans

Some DBMS prefetch to fill in empty frames upon start-up.

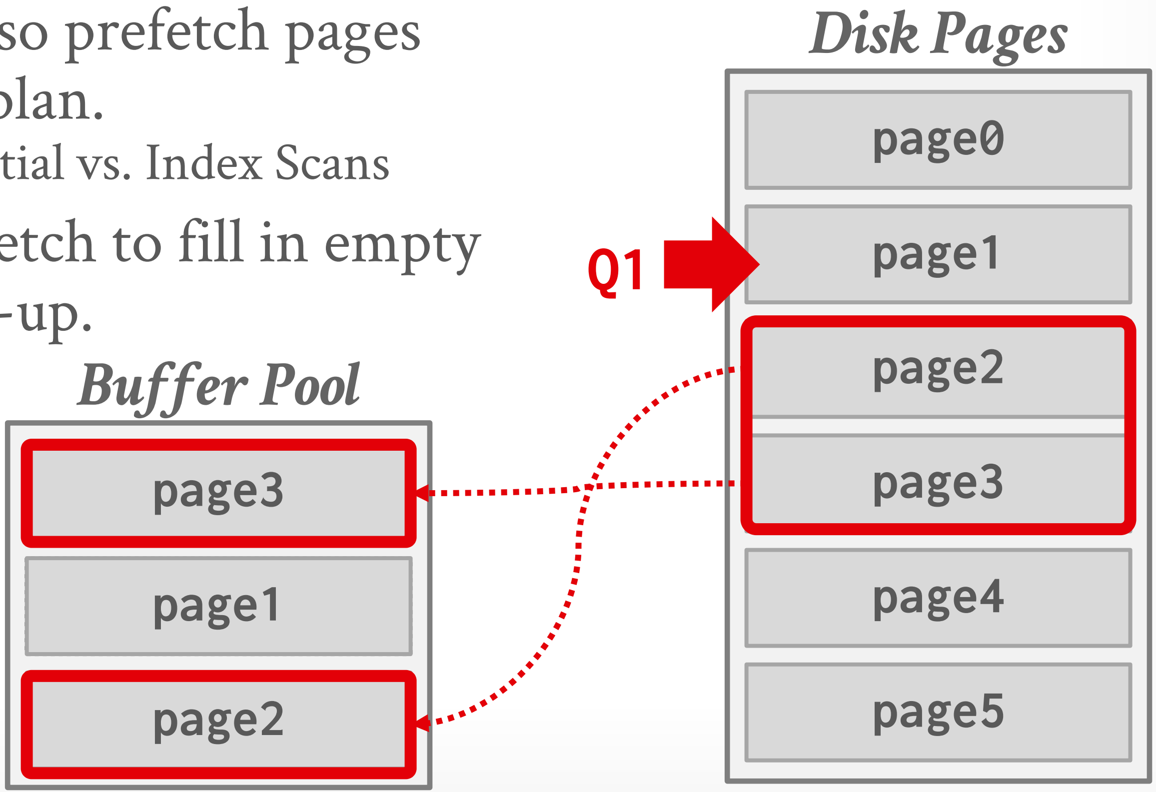


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

→ Examples: Sequential vs. Index Scans

Some DBMS prefetch to fill in empty frames upon start-up.

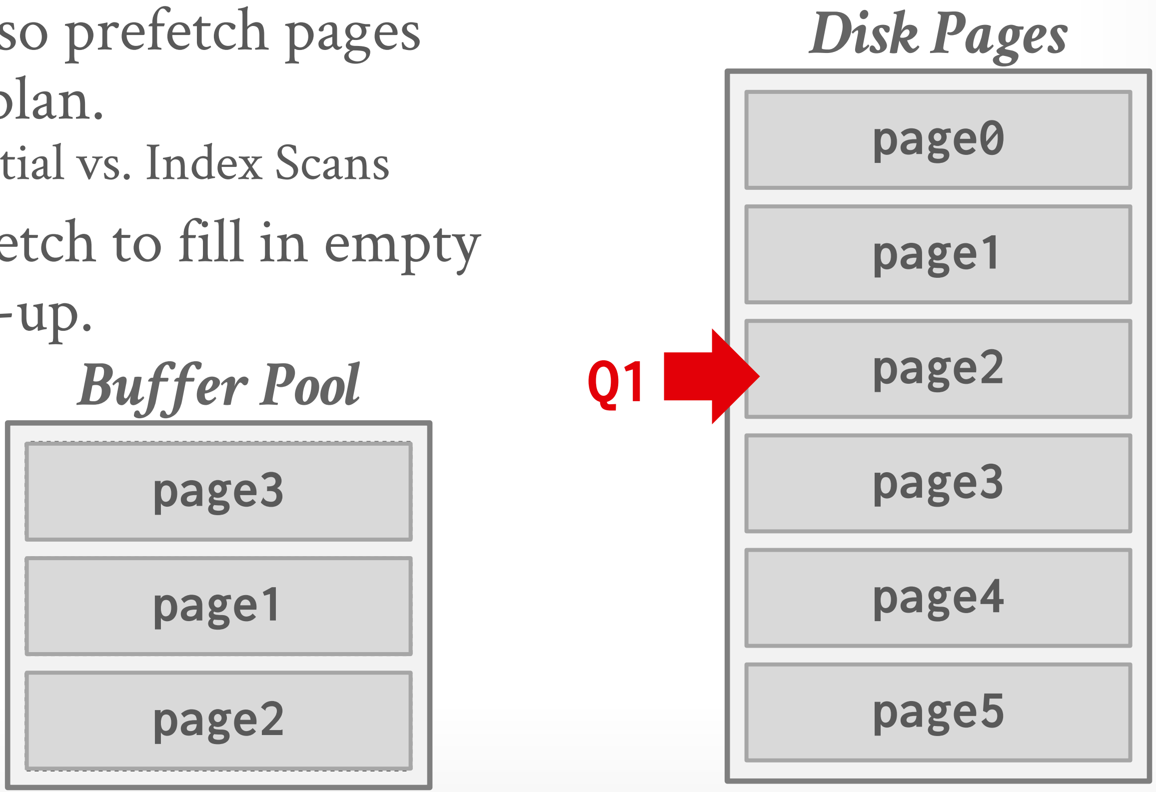


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

→ Examples: Sequential vs. Index Scans

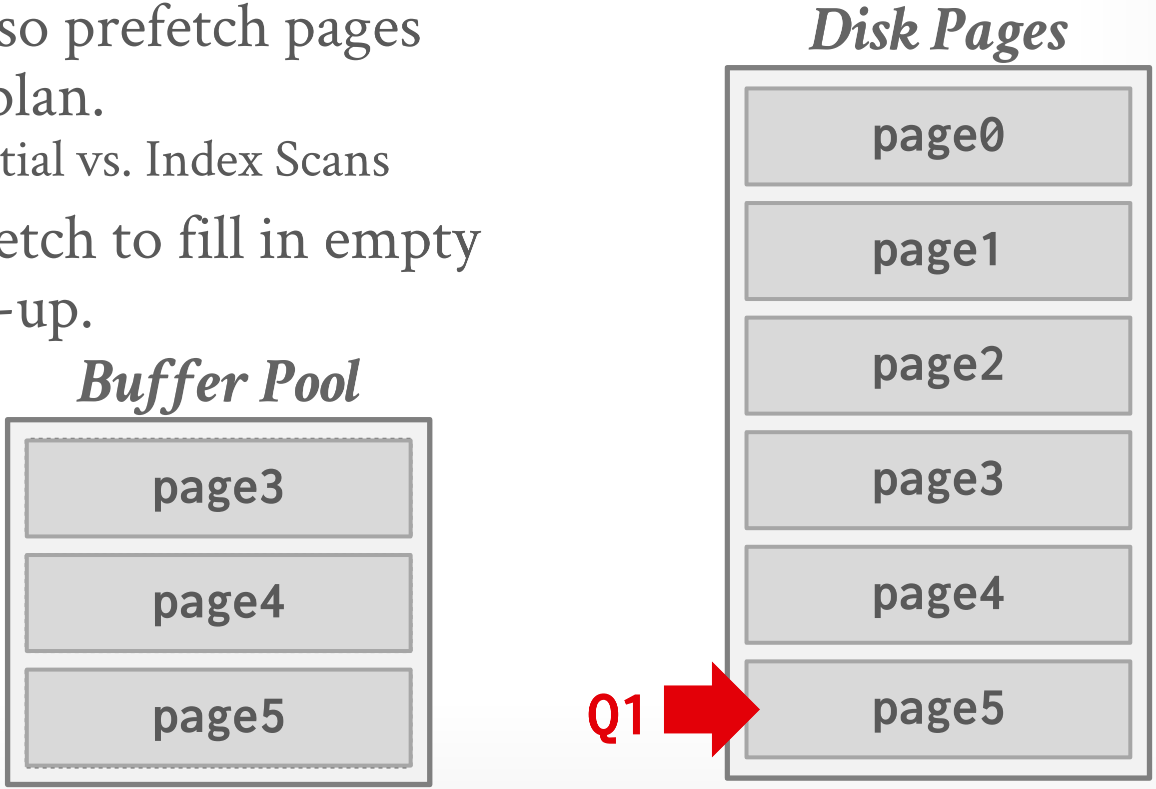
Some DBMS prefetch to fill in empty frames upon start-up.



PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.
→ Examples: Sequential vs. Index Scans

Some DBMS prefetch to fill in empty frames upon start-up.



PRE-FETCHING

Q1

```
SELECT * FROM A
 WHERE val BETWEEN 100 AND 250
```

Buffer Pool

Disk Pages

index-page0

index-page1

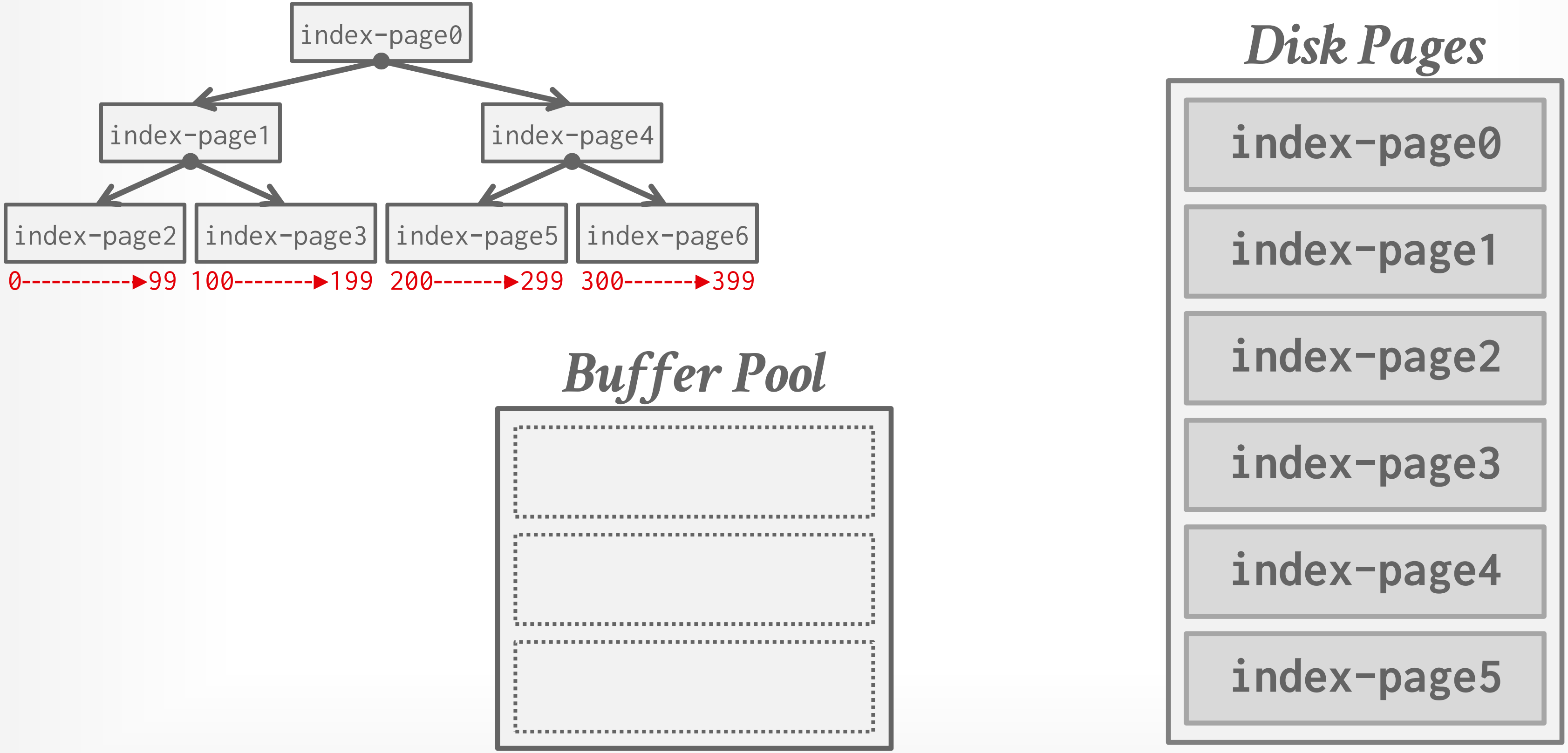
index-page2

index-page3

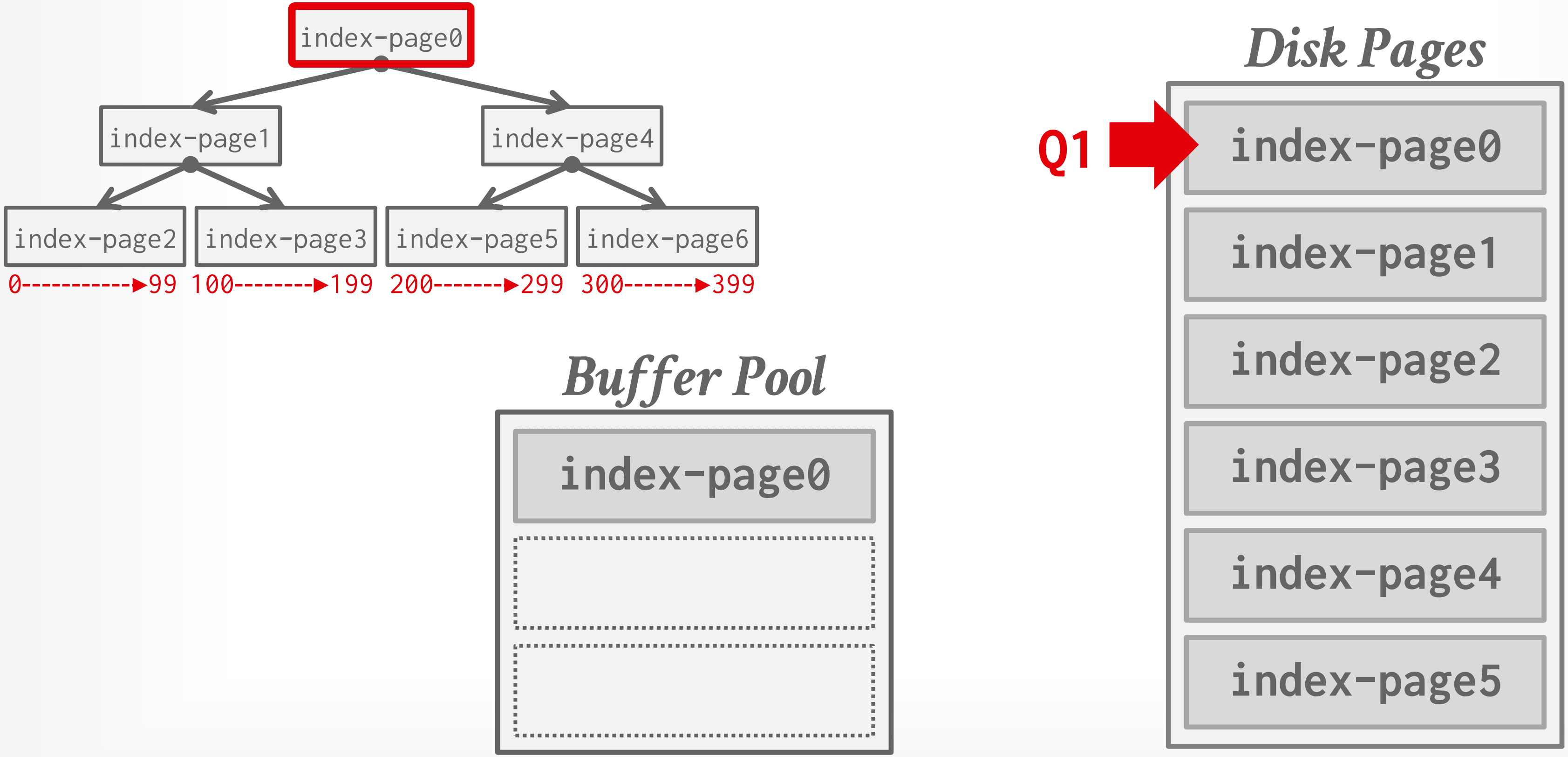
index-page4

index-page5

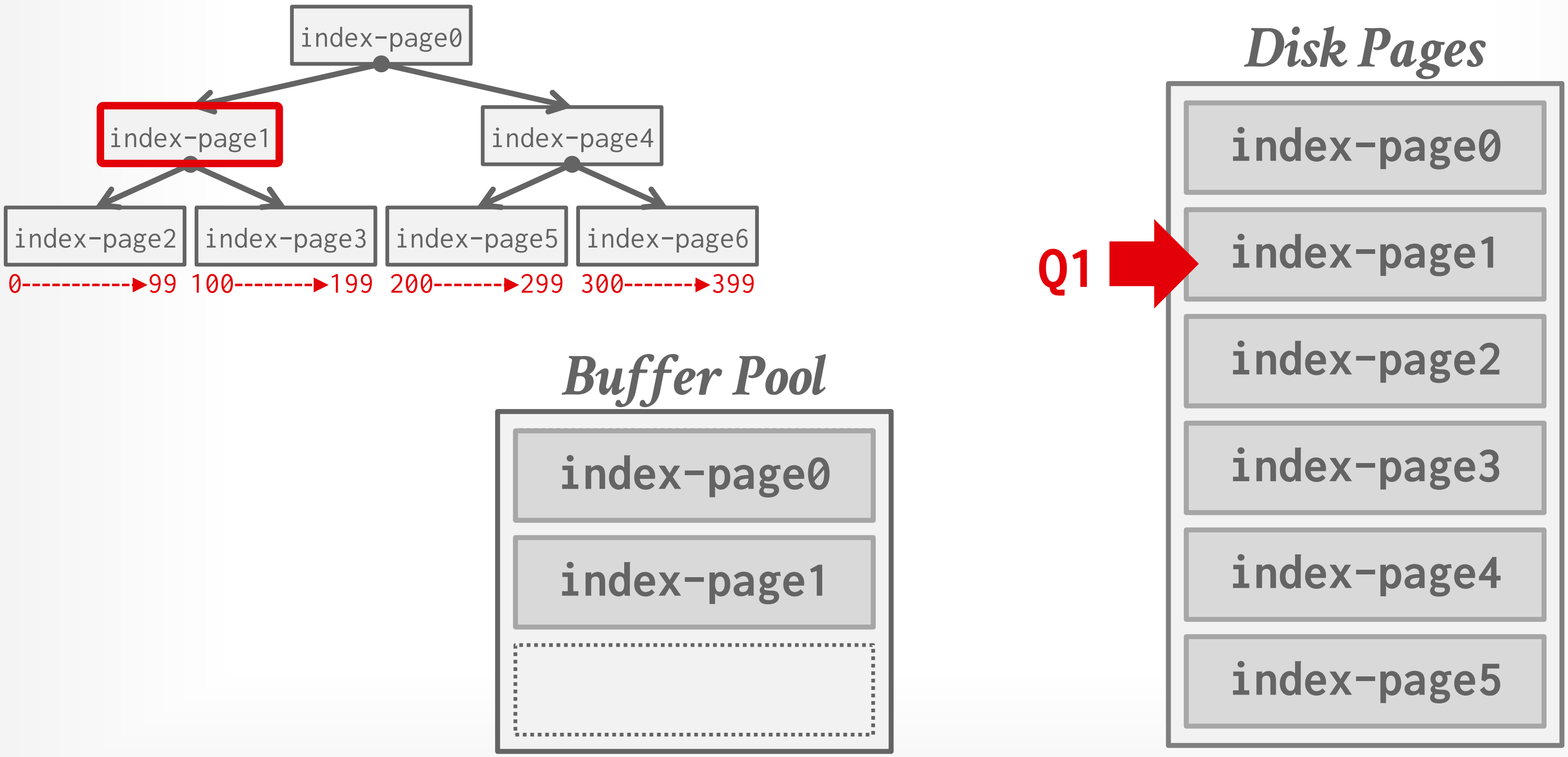
PRE-FETCHING



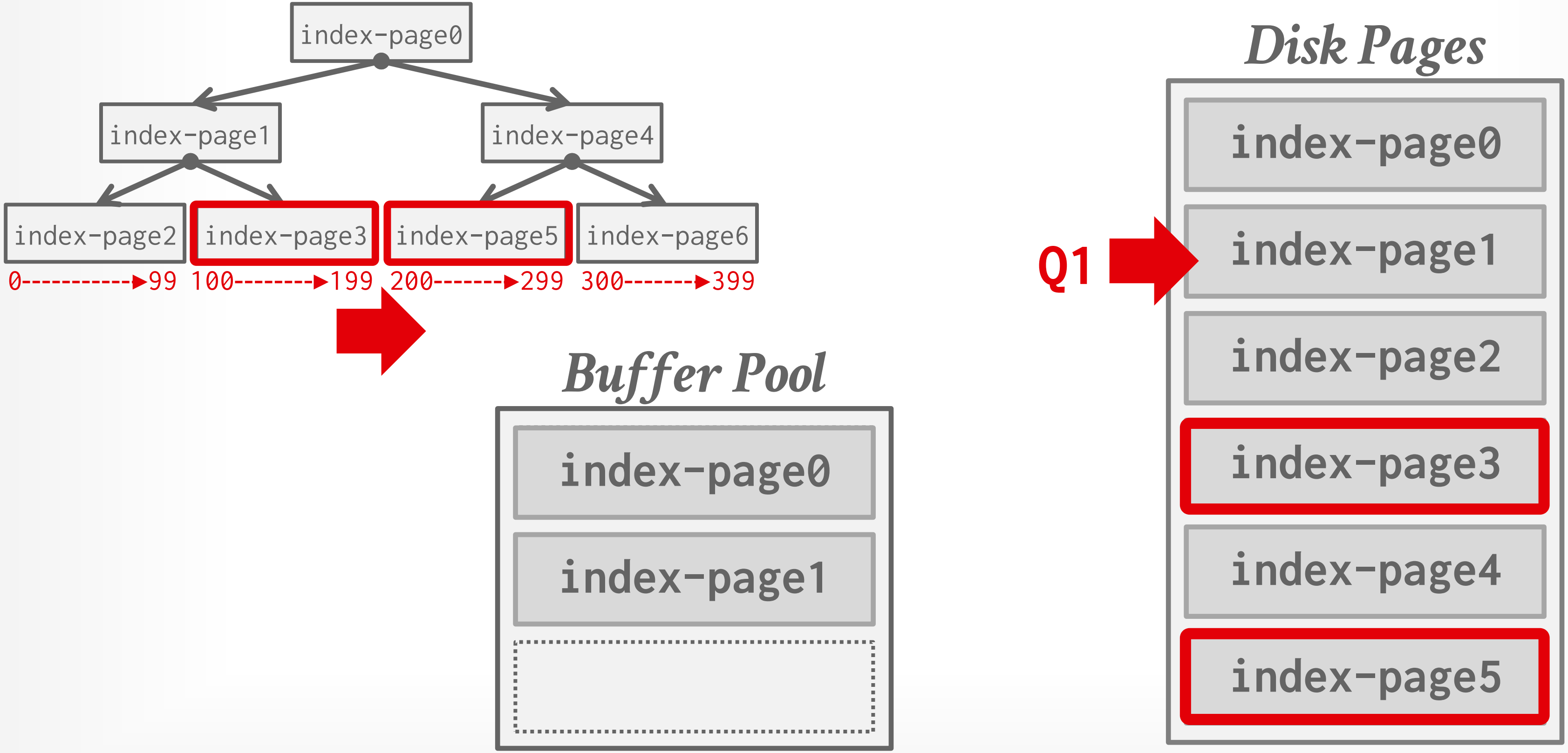
PRE-FETCHING



PRE-FETCHING



PRE-FETCHING



SCAN SHARING

Allow multiple queries to attach to a single cursor that scans a table.

→ Also called ***synchronized scans***.

→ This is different from result caching.

Examples:

→ Fully supported in DB2, MSSQL, Teradata, and Postgres.

→ Oracle only supports cursor sharing for identical queries.

SCAN SHARING

Allow multiple queries to attach to a single cursor that scans a table.

→ Also called ***synchronized scans***.

→ This is different from result caching.

Examples:

→ Fully supported in DB2, MSSQL, Teradata, and Postgres.

→ Oracle only supports cursor sharing for identical queries.

SCAN SHARING

Allow multiple queries to attach to a single cursor that scans a table.

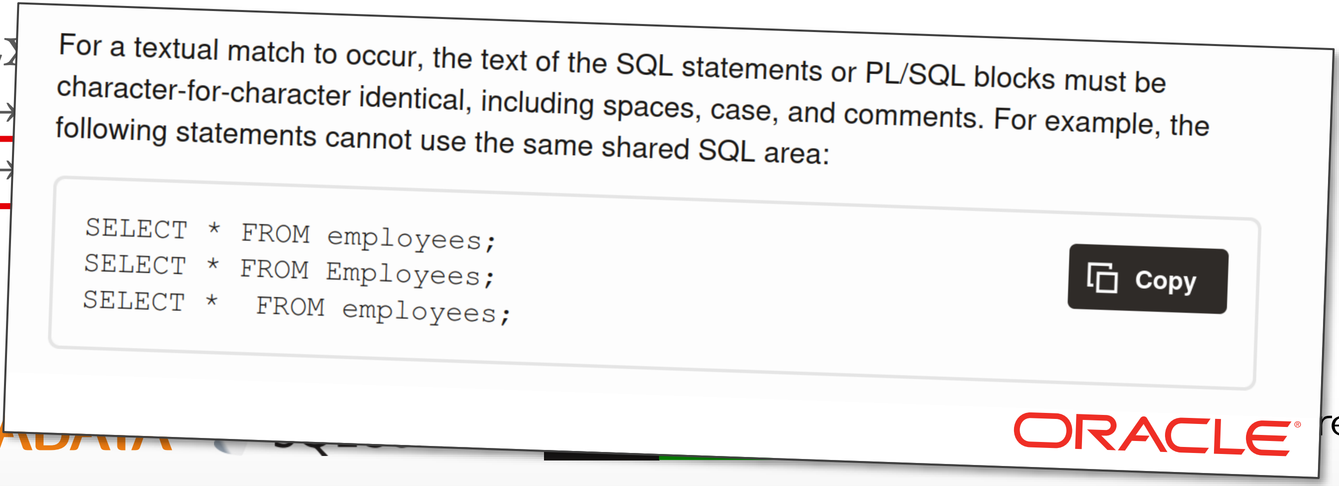
→ Also called ***synchronized scans***.

→ This is different from result caching.

Ex

→

→



For a textual match to occur, the text of the SQL statements or PL/SQL blocks must be character-for-character identical, including spaces, case, and comments. For example, the following statements cannot use the same shared SQL area:

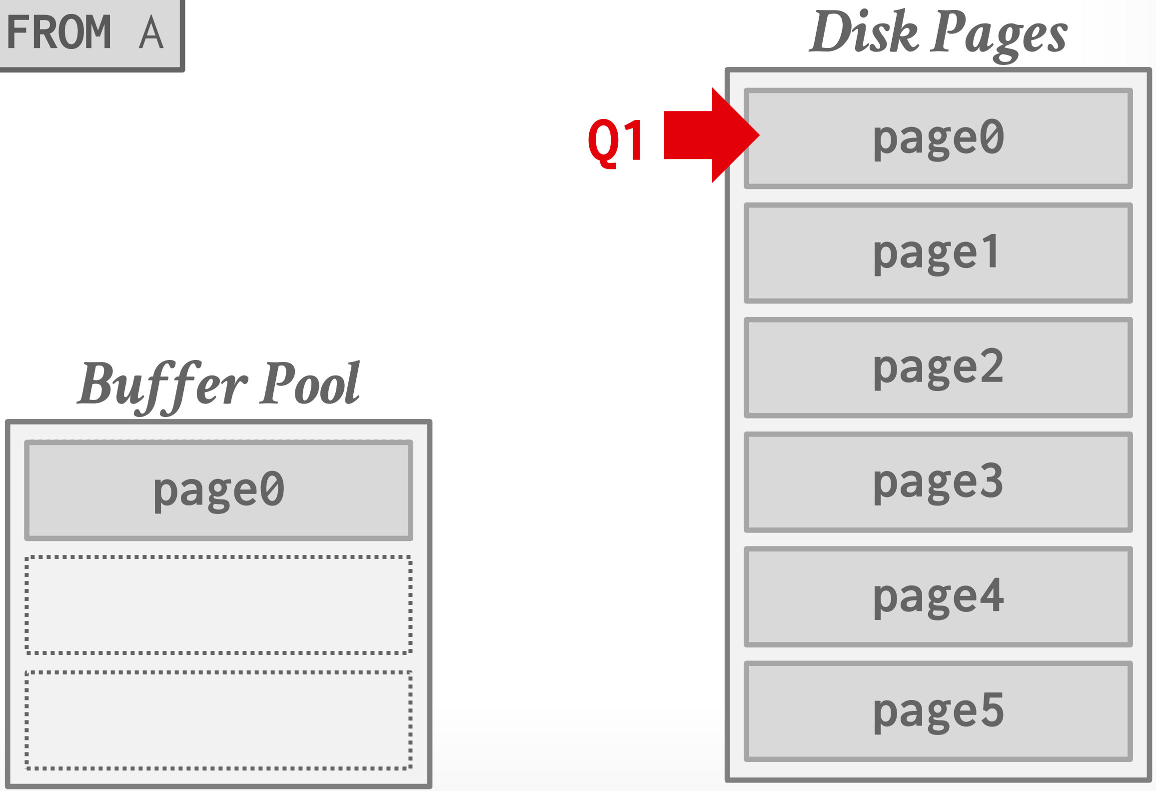
```
SELECT * FROM employees;
SELECT * FROM Employees;
SELECT *  FROM employees;
```

Copy

reSQL

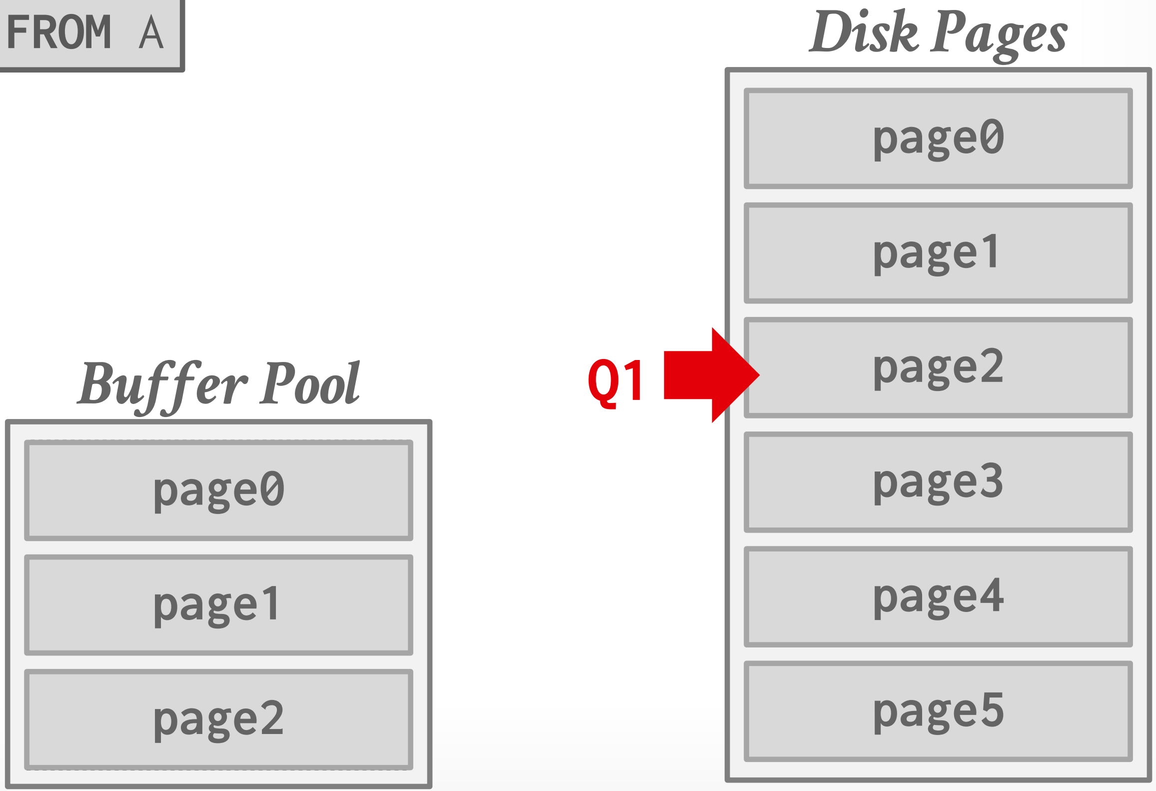
SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



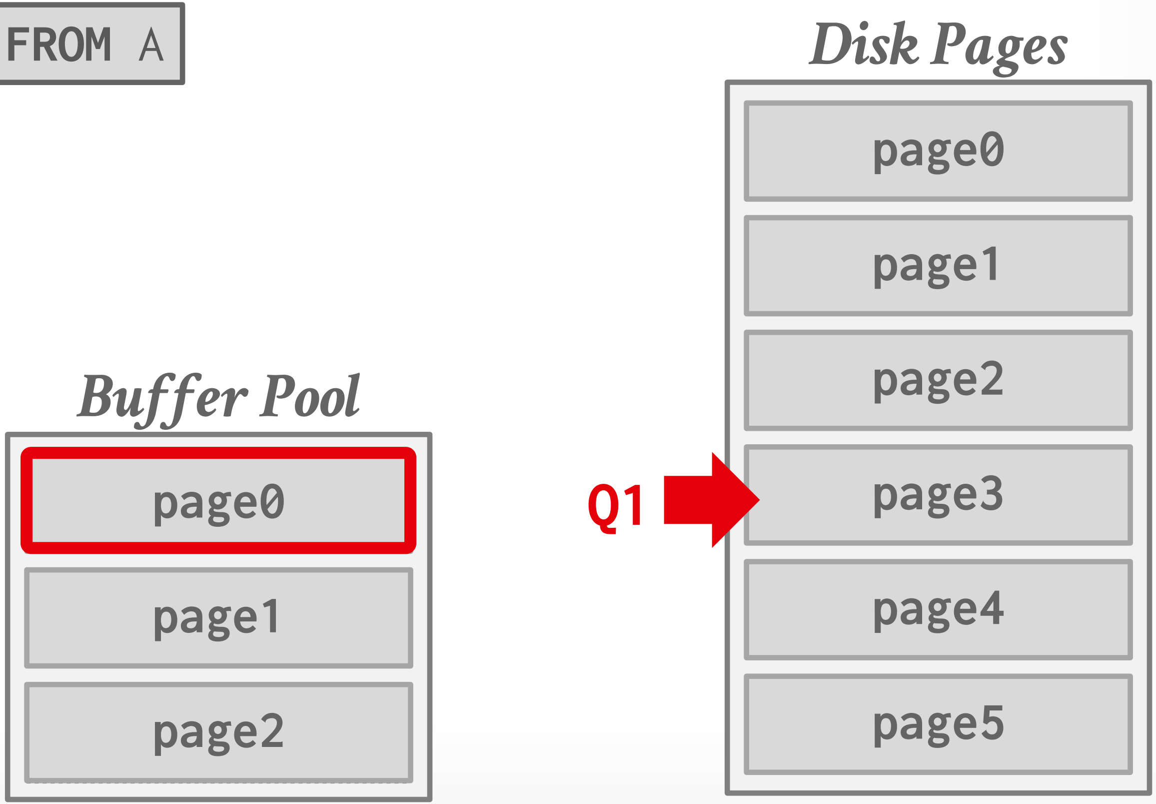
SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



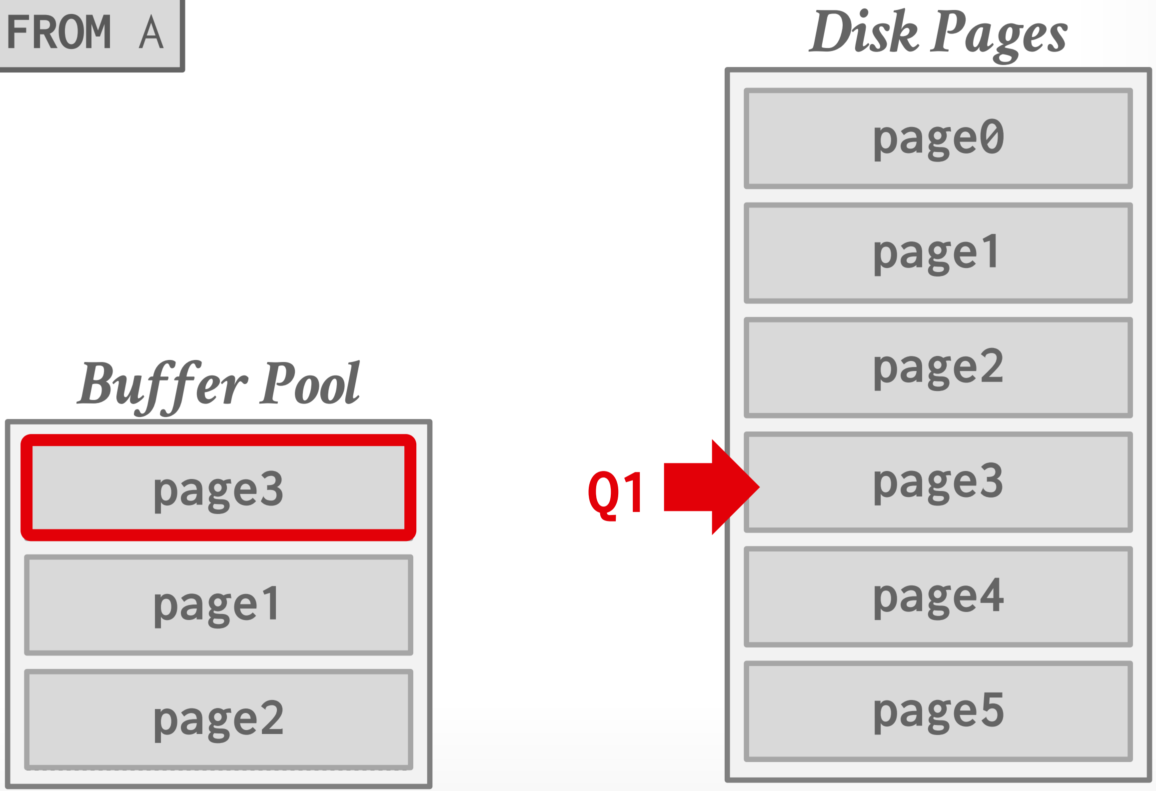
SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



SCAN SHARING

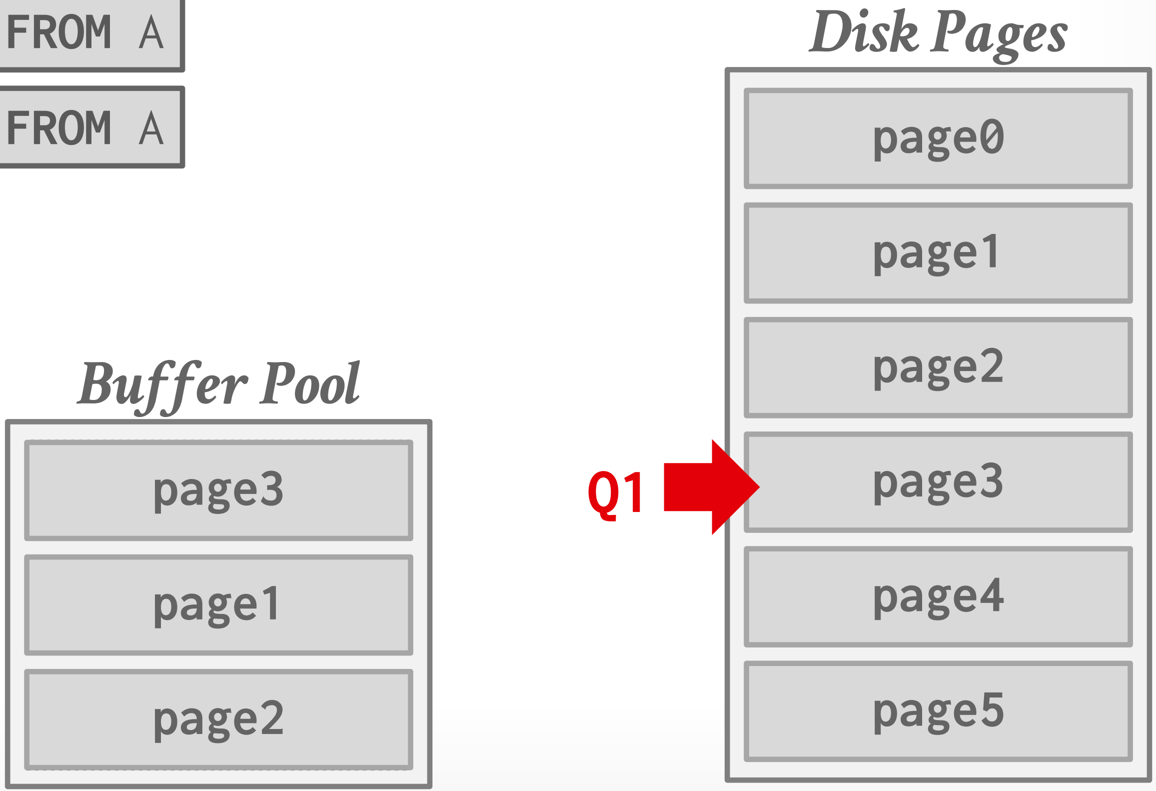
Q1 `SELECT SUM(val) FROM A`



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

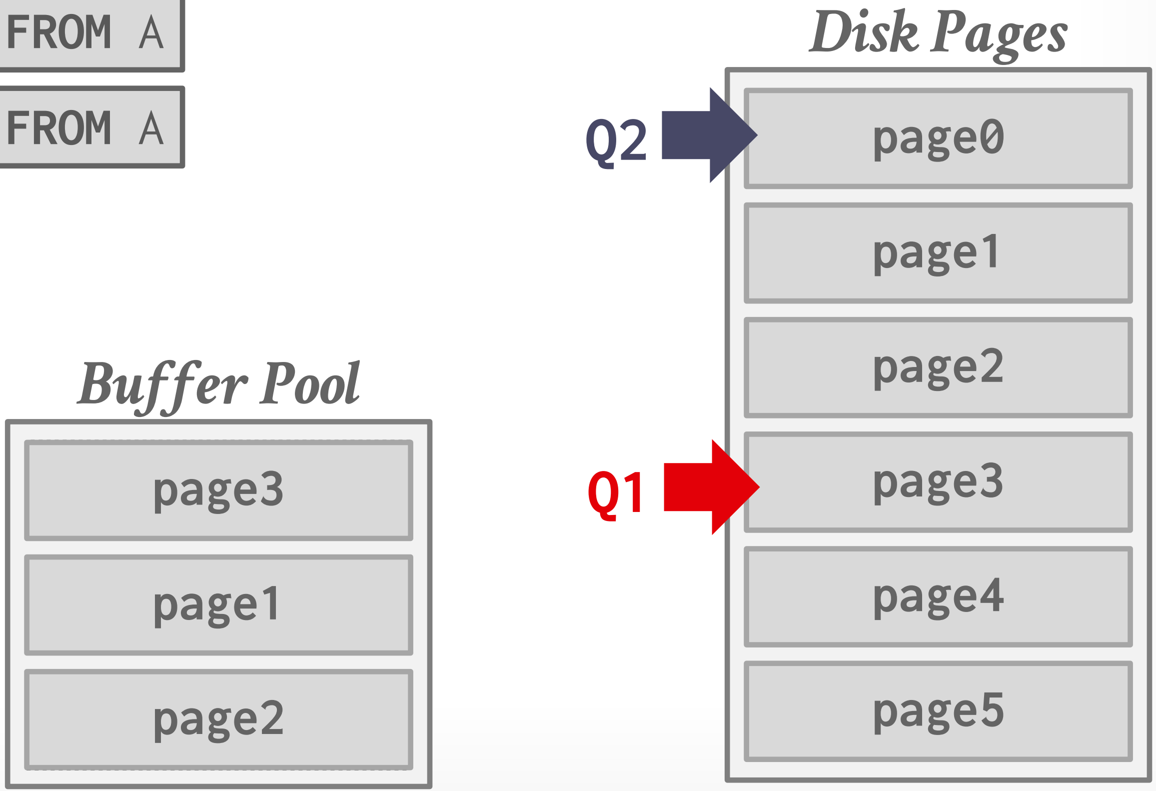
Q2 `SELECT AVG(val) FROM A`



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

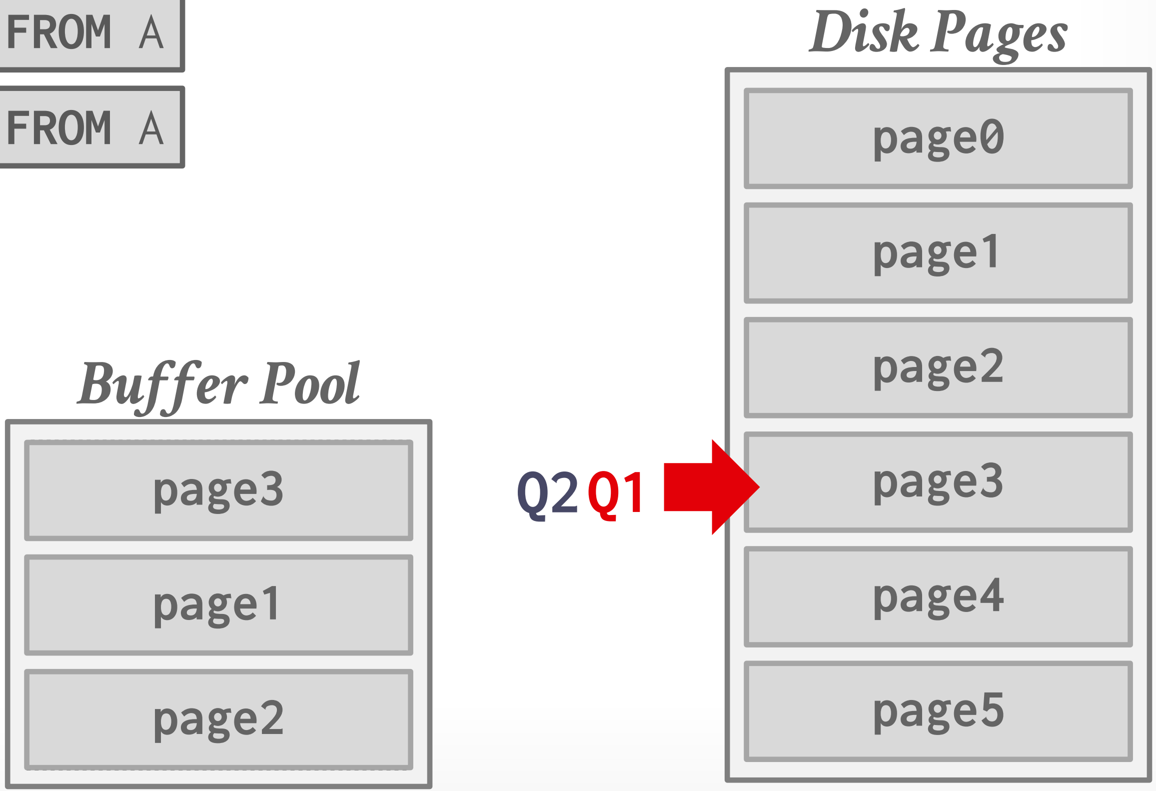
Q2 `SELECT AVG(val) FROM A`



SCAN SHARING

Q1
```
SELECT SUM(val) FROM A
```

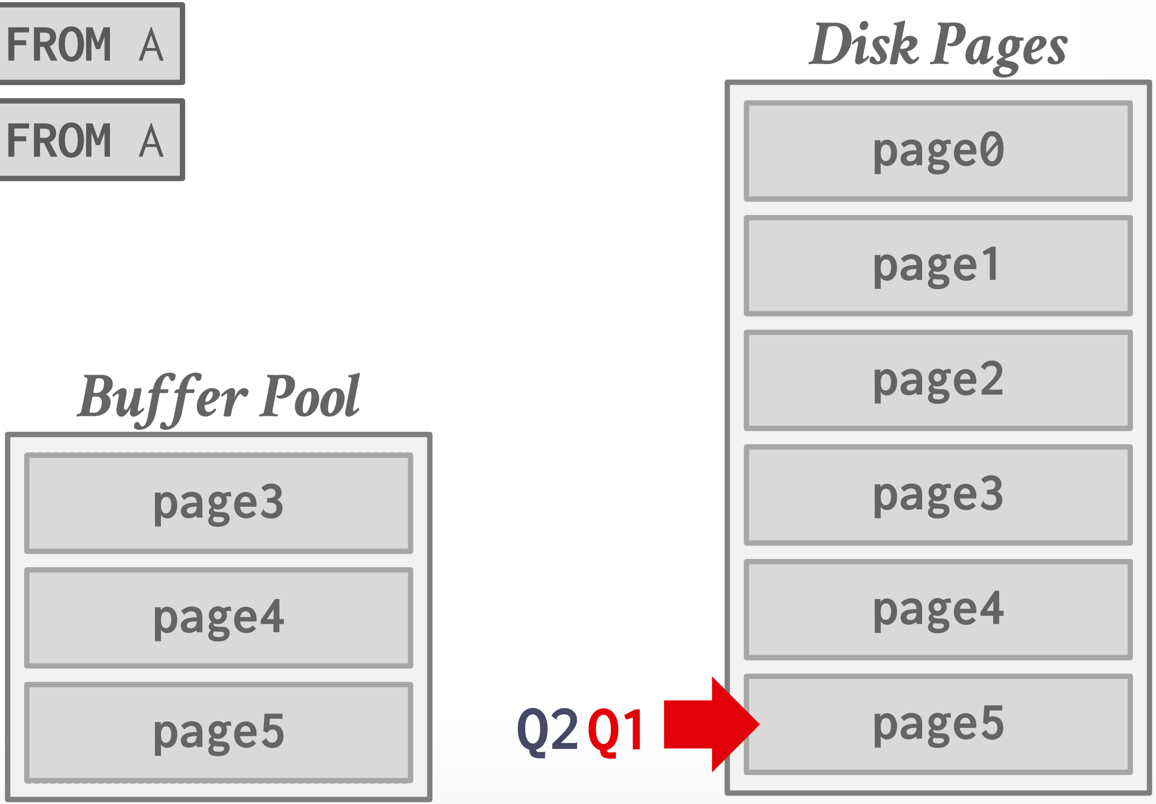
Q2
```
SELECT AVG(val) FROM A
```



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

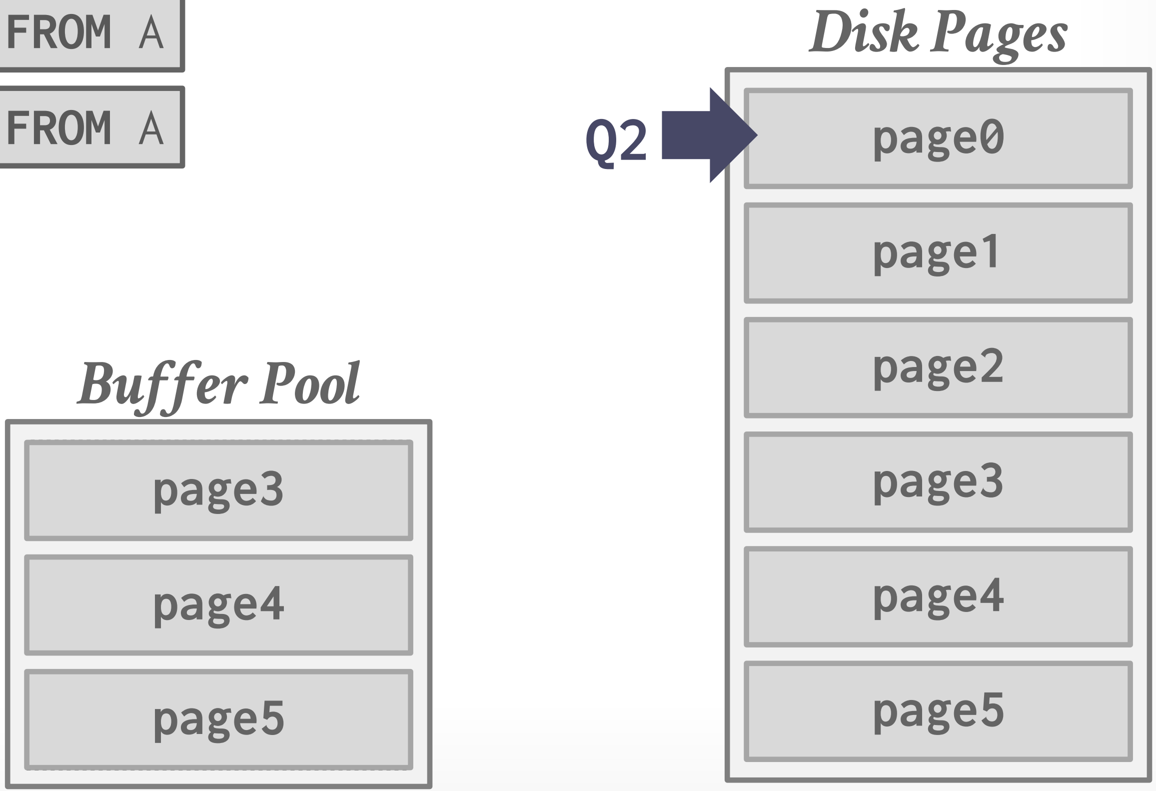
Q2 `SELECT AVG(val) FROM A`



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

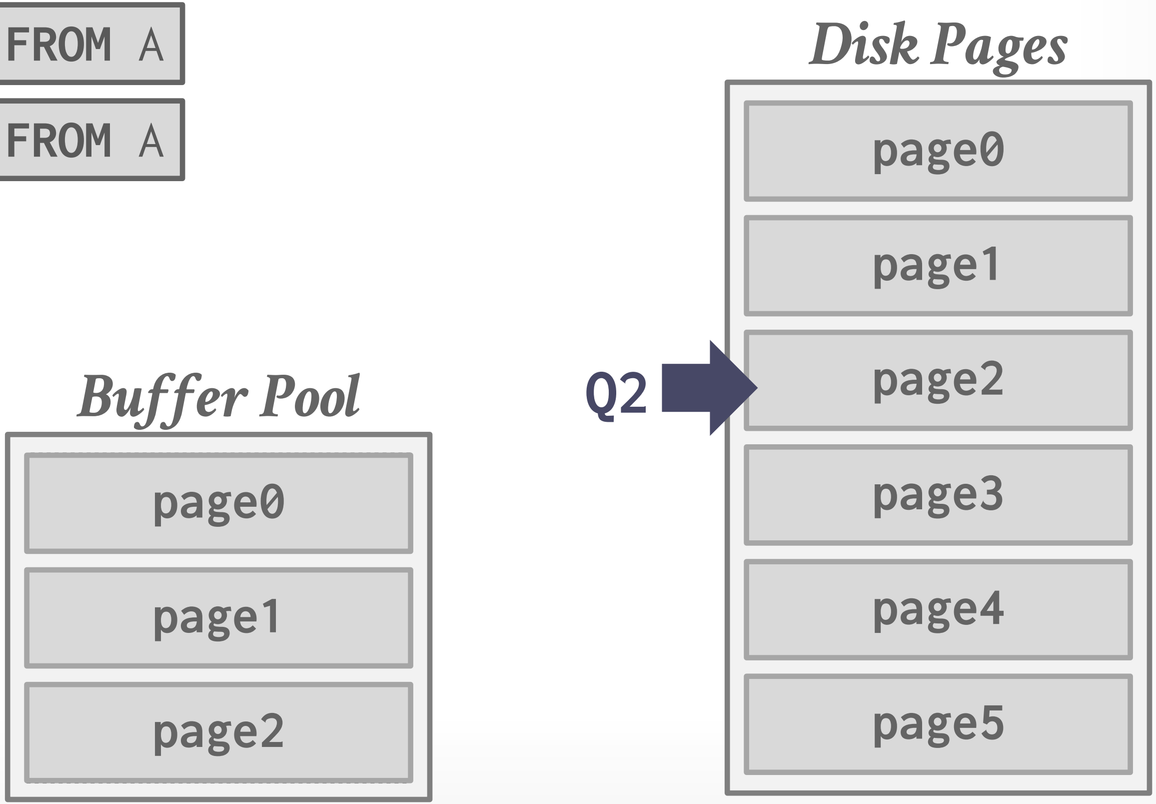
Q2 `SELECT AVG(val) FROM A`



SCAN SHARING

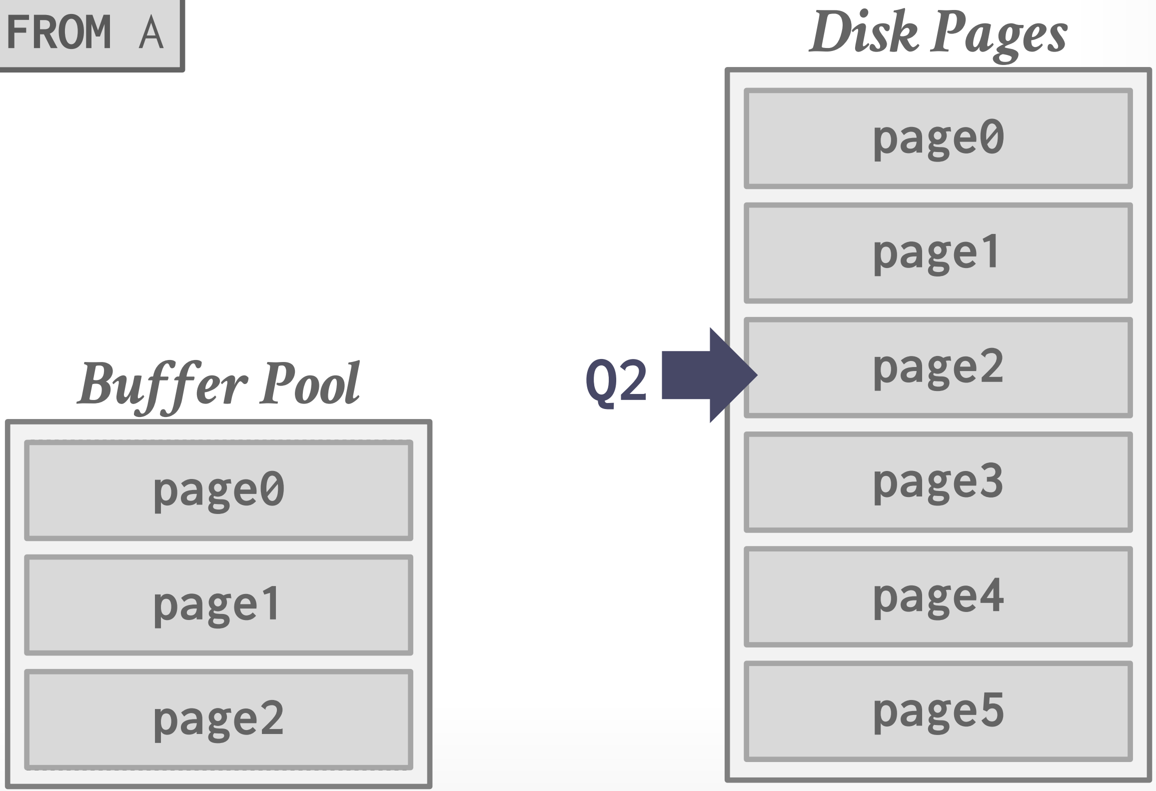
Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Q2' `SELECT * FROM A LIMIT 100`

Disk Pages

page0
page1
page2
page3
page4
page5

Buffer Pool

page0
page1
page2

PREVIOUSLY

We presented a disk-oriented architecture where the DBMS assumes that the primary storage location of the database is on non-volatile disk.

We then discussed a page-oriented storage scheme for organizing tuples across heap files.

PAGE LAYOUT

For any page storage architecture, we now need to decide how to organize the data inside of the page.

→ We are still assuming that we are only storing tuples in a row-oriented storage model.

→ We will also assume that each tuple fits in a single page.

Approach #1: Tuple-oriented Storage

Approach #2: Index-organized Storage ← Today

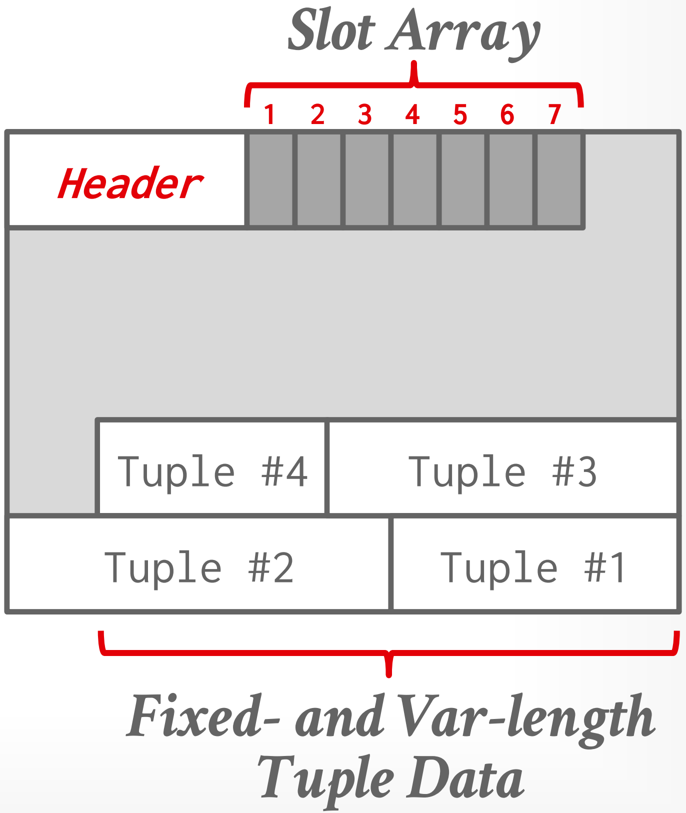
Approach #3: Log-structured Storage

SLOTTED PAGES

The most common page layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.

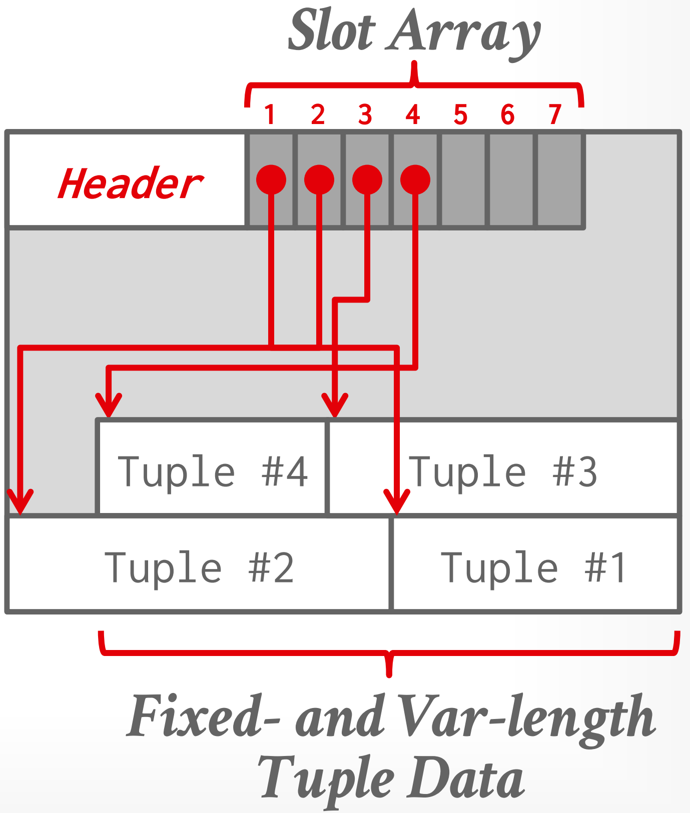


SLOTTED PAGES

The most common page layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.

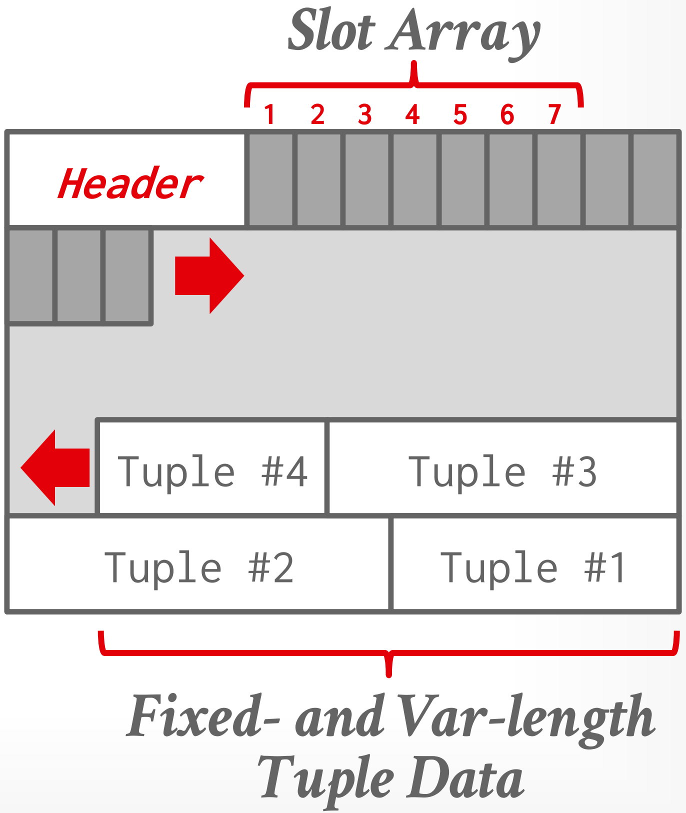


SLOTTED PAGES

The most common page layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.

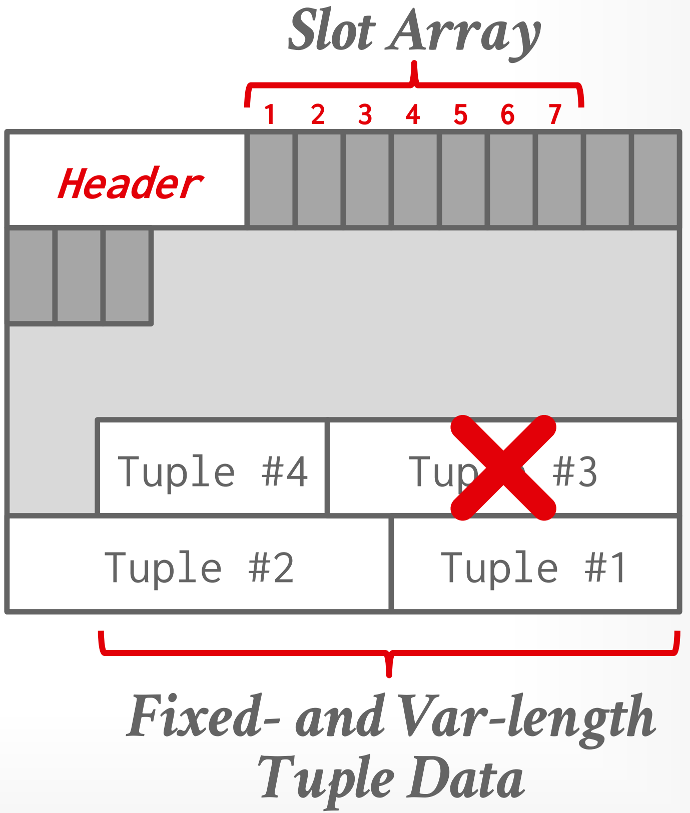


SLOTTED PAGES

The most common page layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.

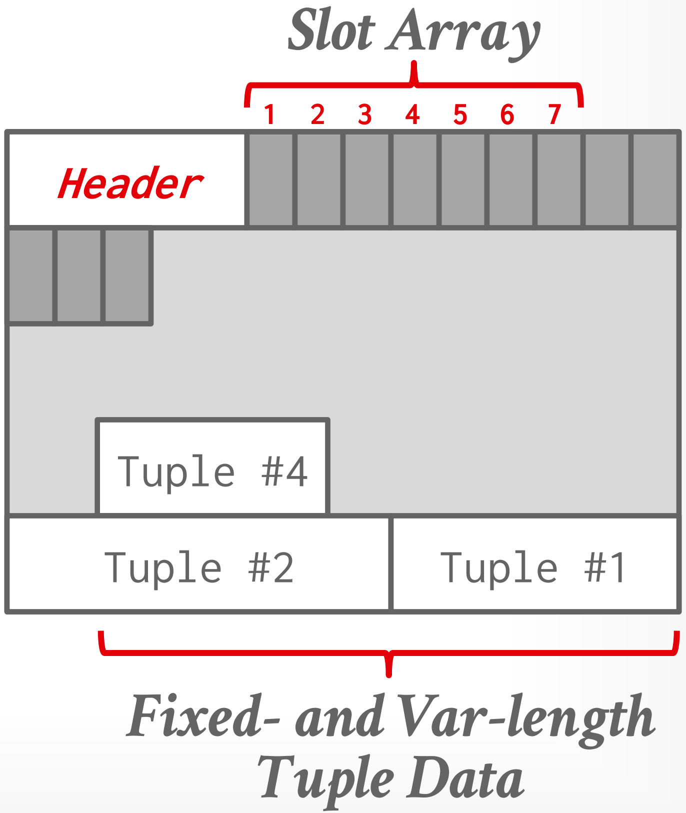


SLOTTED PAGES

The most common page layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.

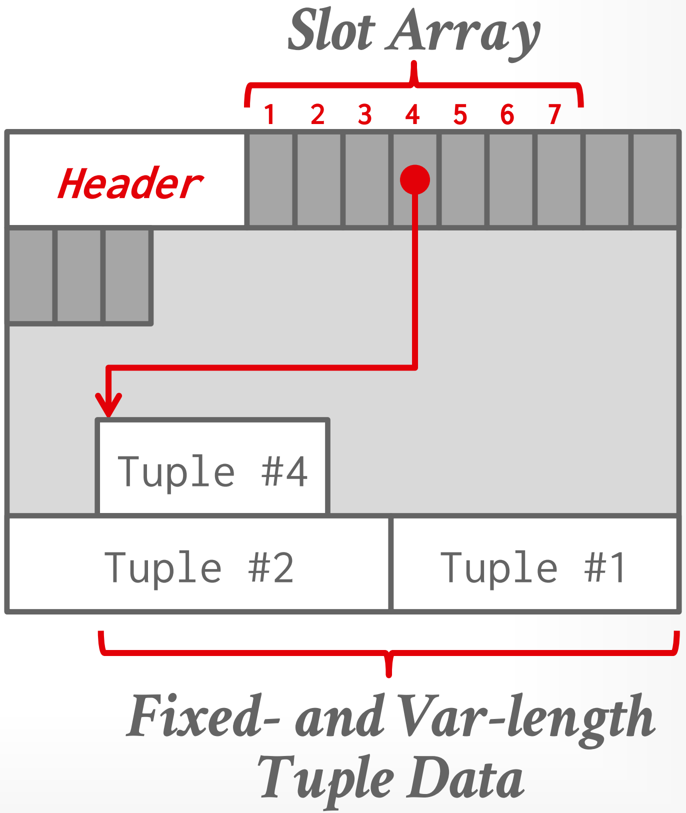


SLOTTED PAGES

The most common page layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.

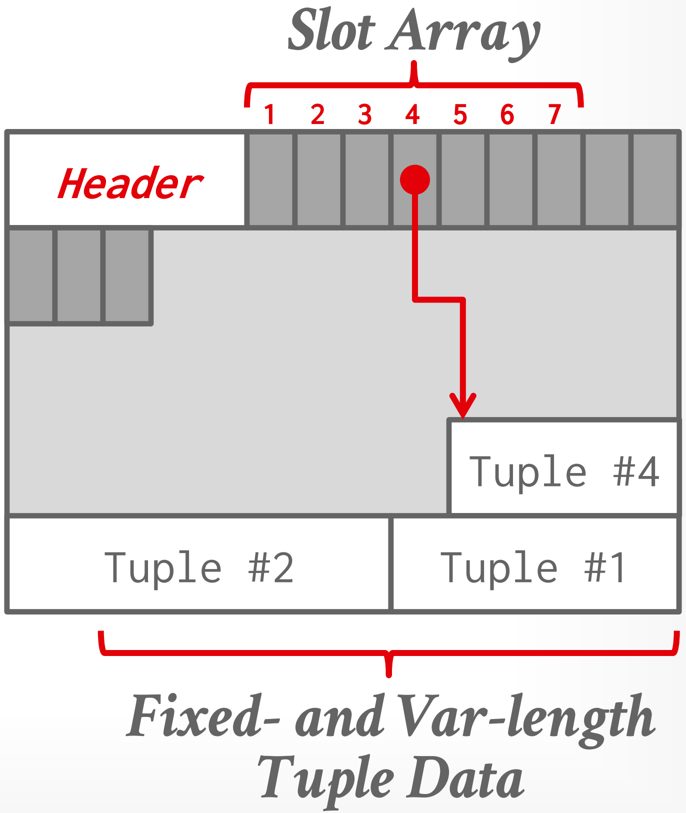


SLOTTED PAGES

The most common page layout scheme is called <u>slotted pages</u>.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.



RECORD IDS

The DBMS assigns each logical tuple a unique **record identifier** that represents its physical location in the database.

→ Example: File Id, Page Id, Slot #
→ Most DBMSs do not store ids in tuple.
→ SQLite uses `ROWID` as the true primary key and stores them as a hidden attribute.

Applications should never rely on these IDs to mean anything.

Record Id Sizes

INGRES	TID	*4-bytes*
PostgreSQL	CTID	*6-bytes*
SQLite	ROWID	*8-bytes*
Microsoft SQL Server	%%physloc%%	*8-bytes*
Firebird	RDB$DB_KEY	*8-bytes*
ORACLE	ROWID	*10-bytes*

TUPLE-ORIENTED STORAGE: READS

Get an existing tuple using its record id:

→ Check page directory to find location of page.
→ Retrieve the page from disk (if not in memory).
→ Find offset in page using slot array.

The DBMS relies on <u>indexes</u> to find individual tuples because the tables are inherently unsorted.

But what if the DBMS could keep tuples sorted automatically using an index?

TUPLE-ORIENTED STORAGE: READS

Get an existing tuple using its $\boxed{\textbf{record id:}}$

→ Check page directory to find location of page.
→ Retrieve the page from disk (if not in memory).
→ Find offset in page using slot array.

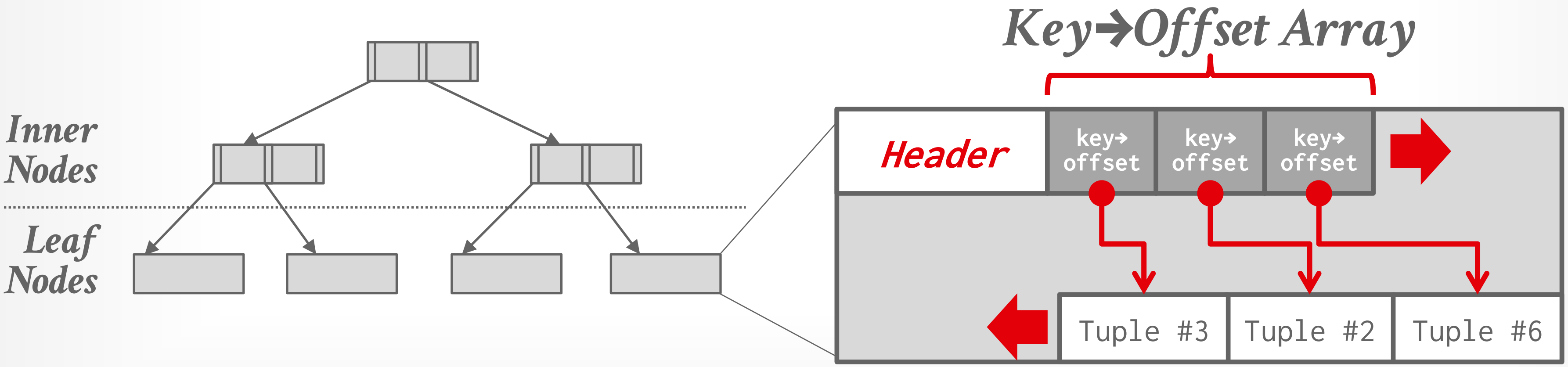
The DBMS relies on indexes to find individual tuples because the tables are inherently unsorted.

But what if the DBMS could keep tuples sorted automatically using an index?

INDEX-ORGANIZED STORAGE

DBMS stores a table's tuples as the value of an index data structure.

→ Still use a page layout that looks like a slotted page.
→ Tuples are typically sorted in page based on key.



TUPLE-ORIENTED STORAGE: WRITES

Insert a new tuple:

→ Check page directory to find a page with a free slot.
→ Retrieve the page from disk (if not in memory).
→ Check slot array to find empty space in page that will fit.

Update an existing tuple using its record id:

→ Check page directory to find location of page.
→ Retrieve the page from disk (if not in memory).
→ Find offset in page using slot array.
→ If new data fits, overwrite existing data.
Otherwise, mark existing tuple as deleted and insert new version in a different page.

TUPLE-ORIENTED STORAGE

Problem #1: Fragmentation

→ Pages are not fully utilized (unusable space, empty slots).

Problem #2: Useless Disk I/O

→ DBMS must fetch entire page to update one tuple.

Problem #3: Random Disk I/O

→ Worse case scenario when updating multiple tuples is that each tuple is on a separate page.

What if the DBMS cannot overwrite data in pages and could only create new pages?

→ E
xamples: HDFS, Google Colossus, S3 Express

LOG-STRUCTURED STORAGE

Instead of storing tuples in pages and updating the in-place, the DBMS maintains a log that records changes to tuples.

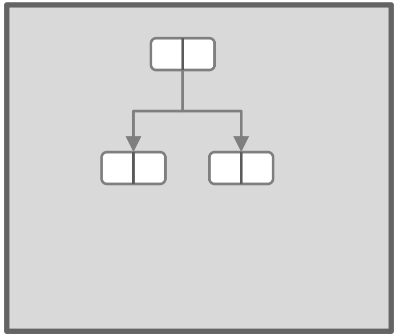
- → Each log entry represents a tuple **PUT**/**DELETE** operation.
- → Originally proposed as log-structure merge trees (LSM Trees) in 1996.

The DBMS applies changes to an in-memory data structure (***MemTable***) and then writes out the changes sequentially to disk as sorted-string tables (***SSTables***).

LOG-STRUCTURED STORAGE

PUT (key101,a_1)

MemTable

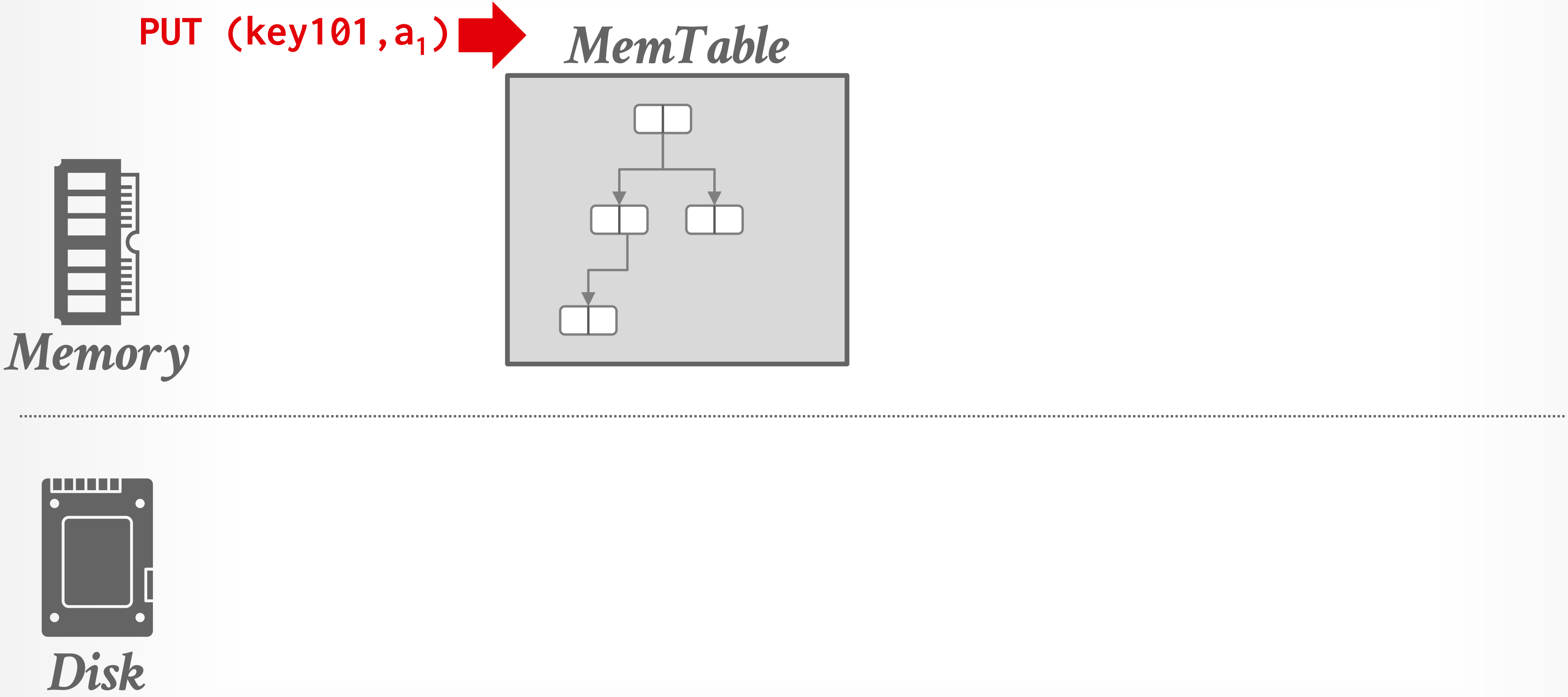


Memory

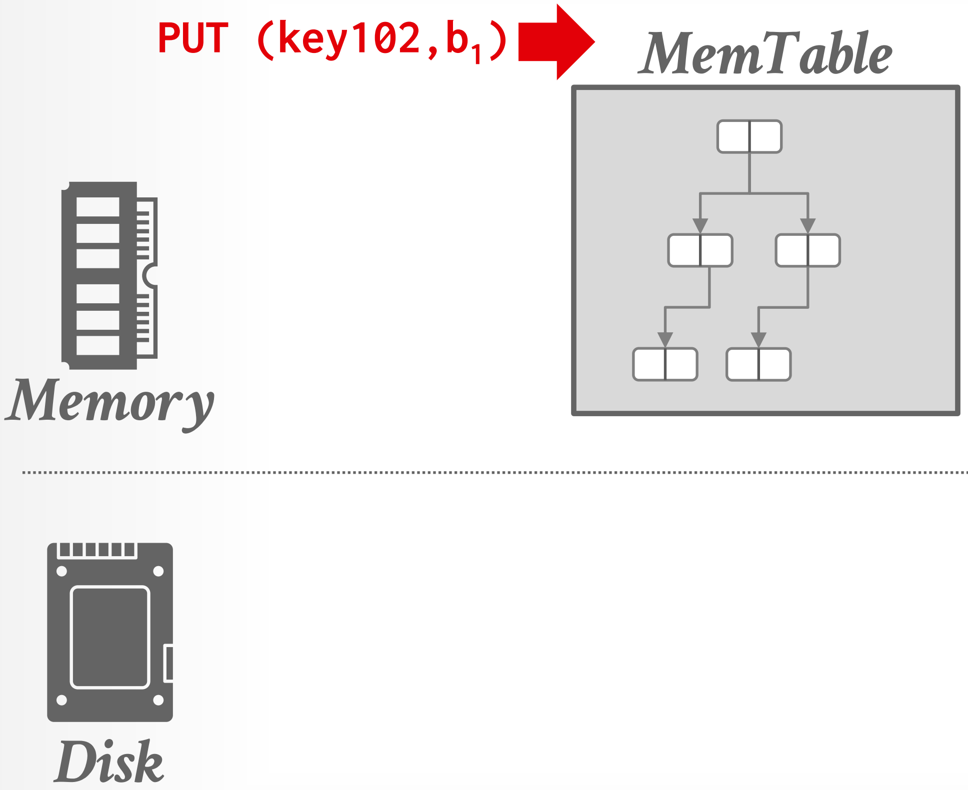
Disk

LOG-STRUCTURED STORAGE

PUT (key101,a_1)

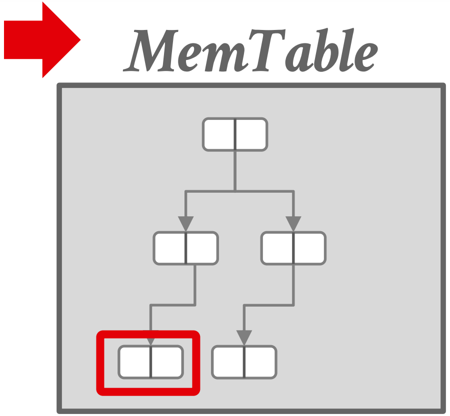


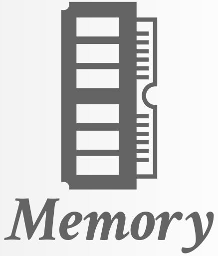
LOG-STRUCTURED STORAGE

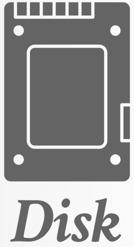


LOG-STRUCTURED STORAGE

PUT (key101, a_2)



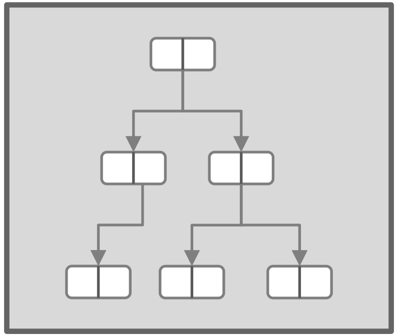


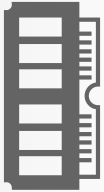


LOG-STRUCTURED STORAGE

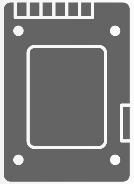
PUT (key103,c_1)

MemTable



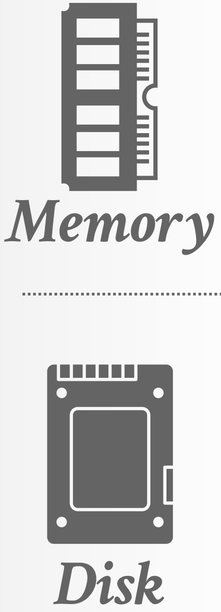


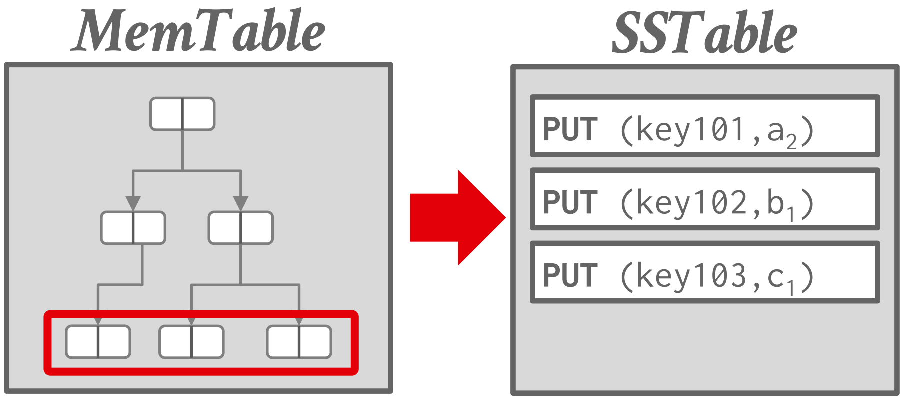
Memory



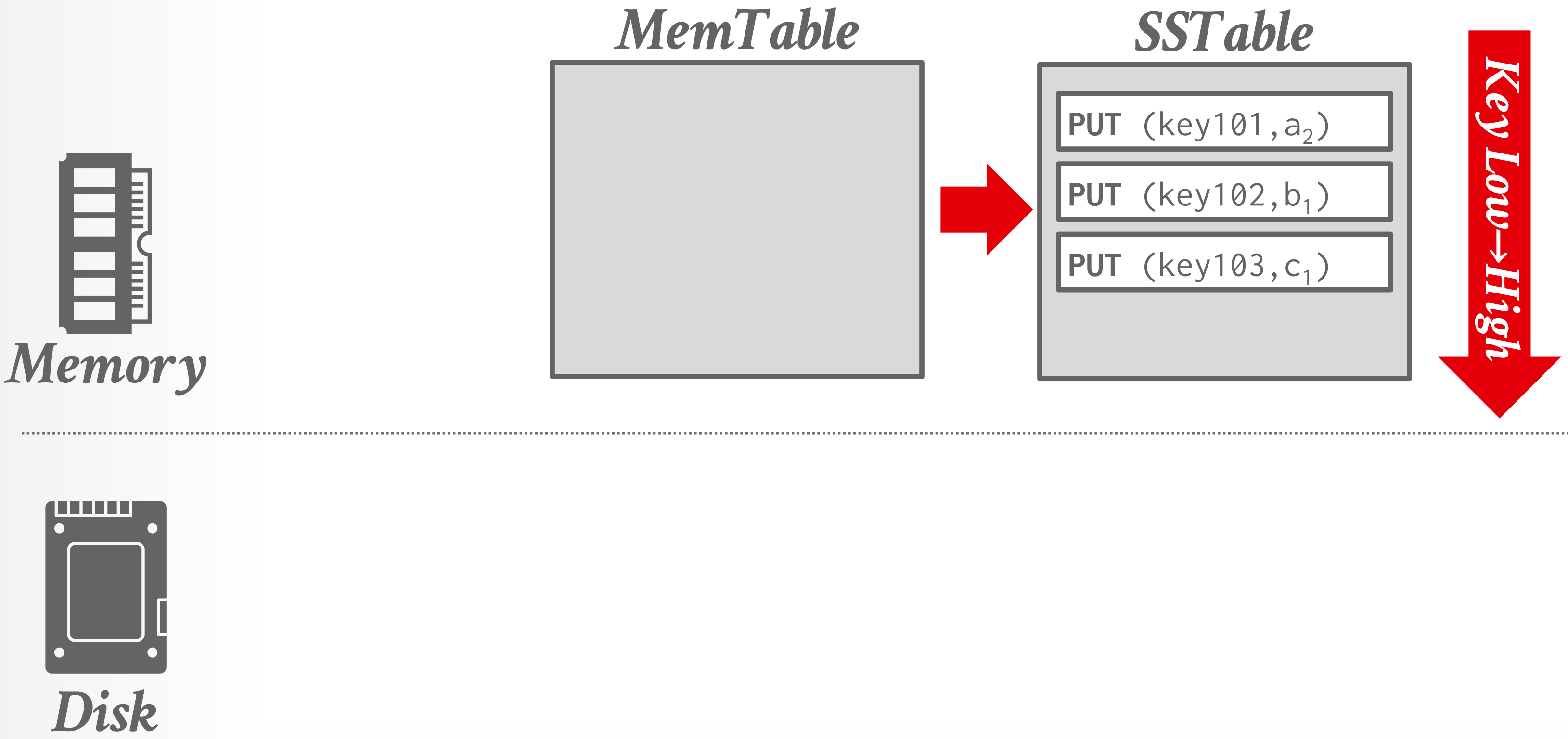
Disk

LOG-STRUCTURED STORAGE

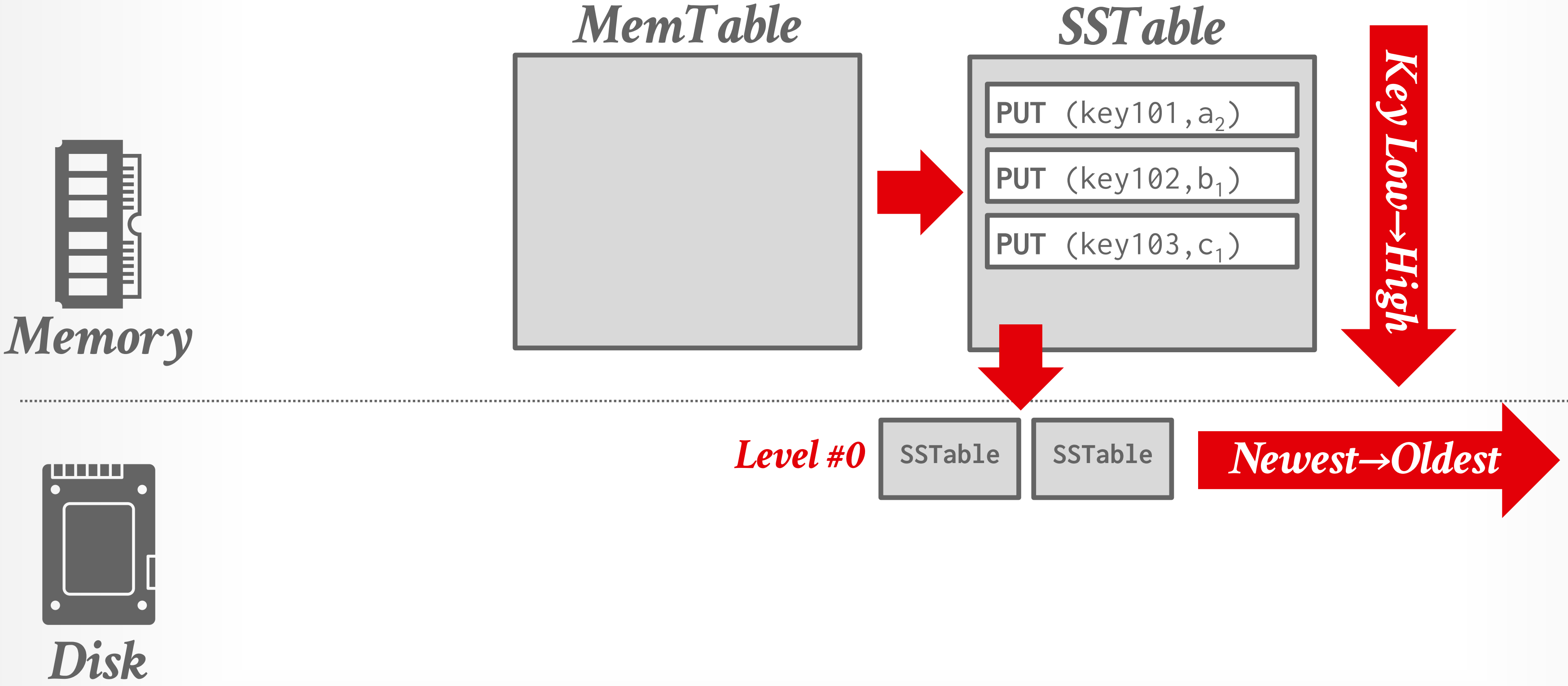




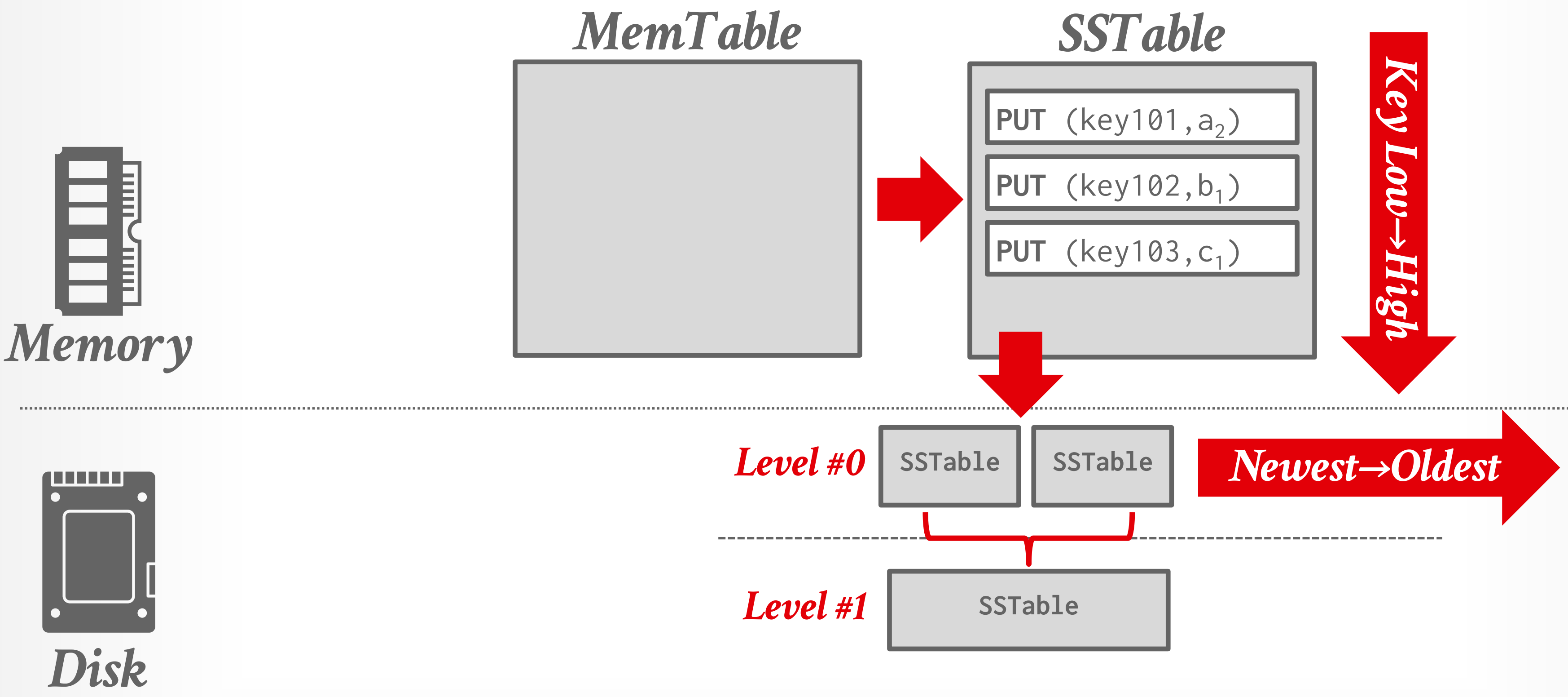
LOG-STRUCTURED STORAGE



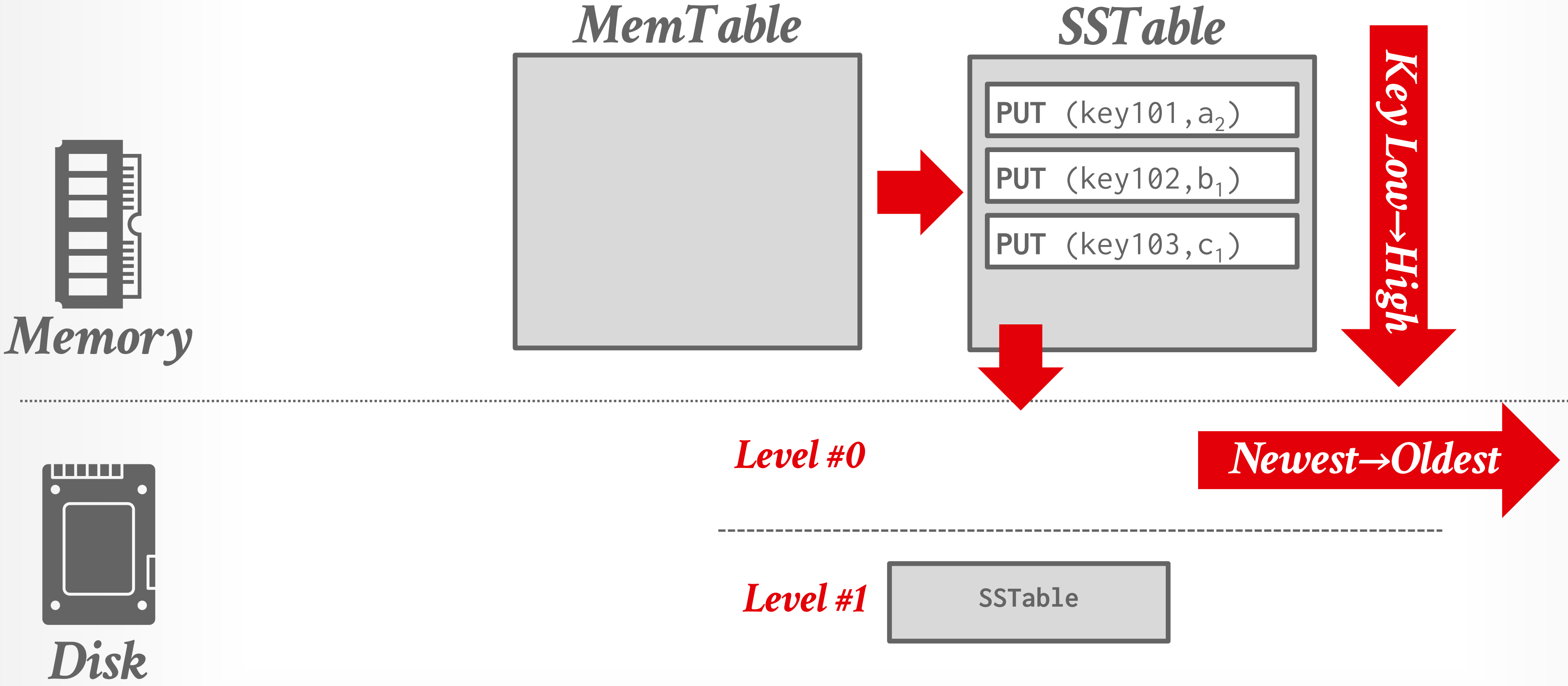
LOG-STRUCTURED STORAGE



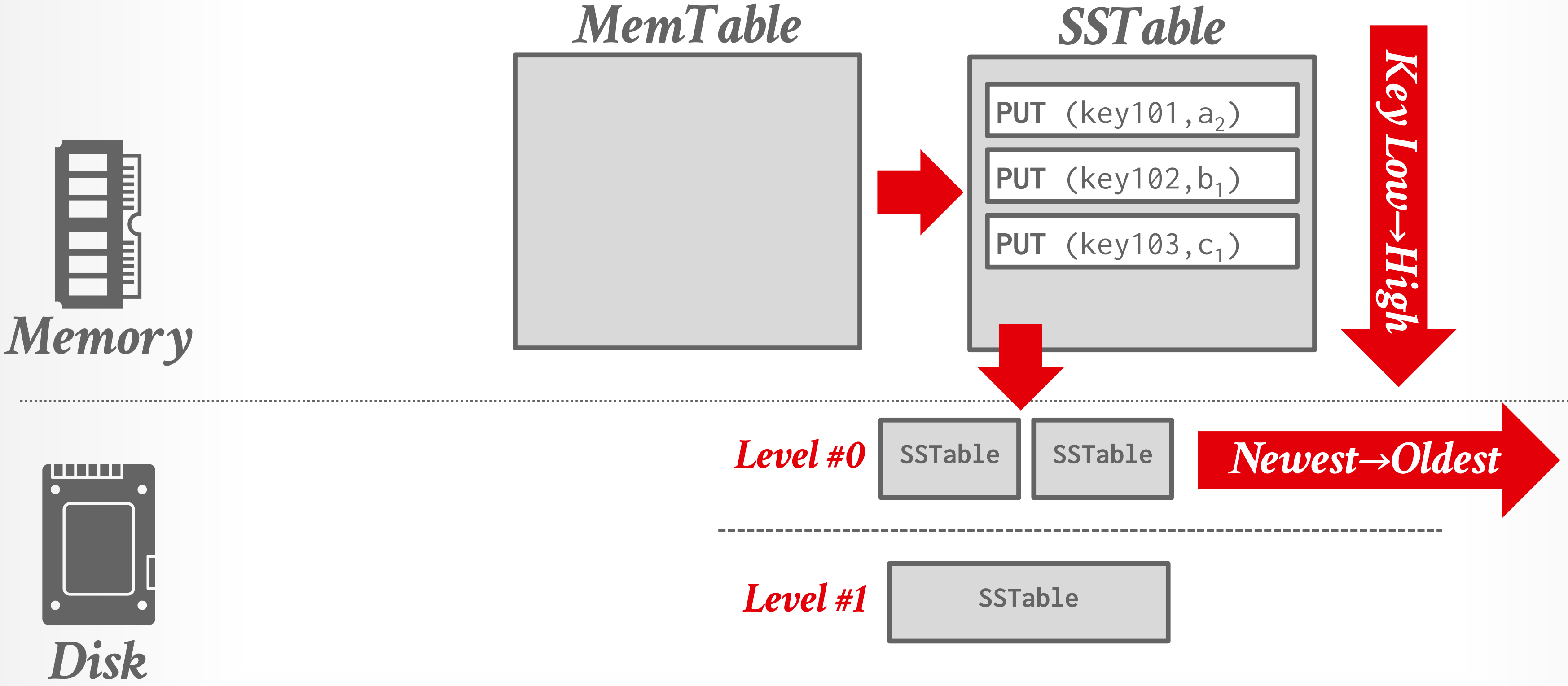
LOG-STRUCTURED STORAGE



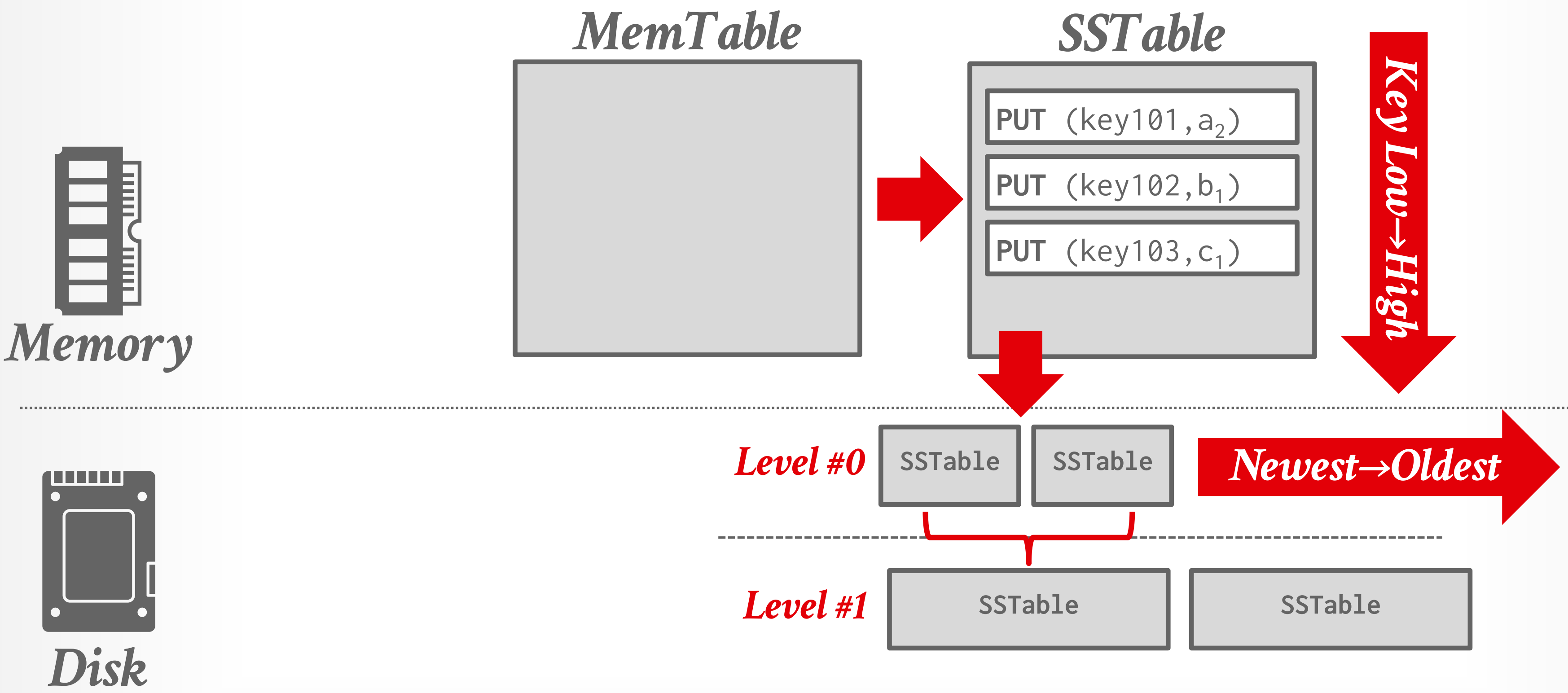
LOG-STRUCTURED STORAGE



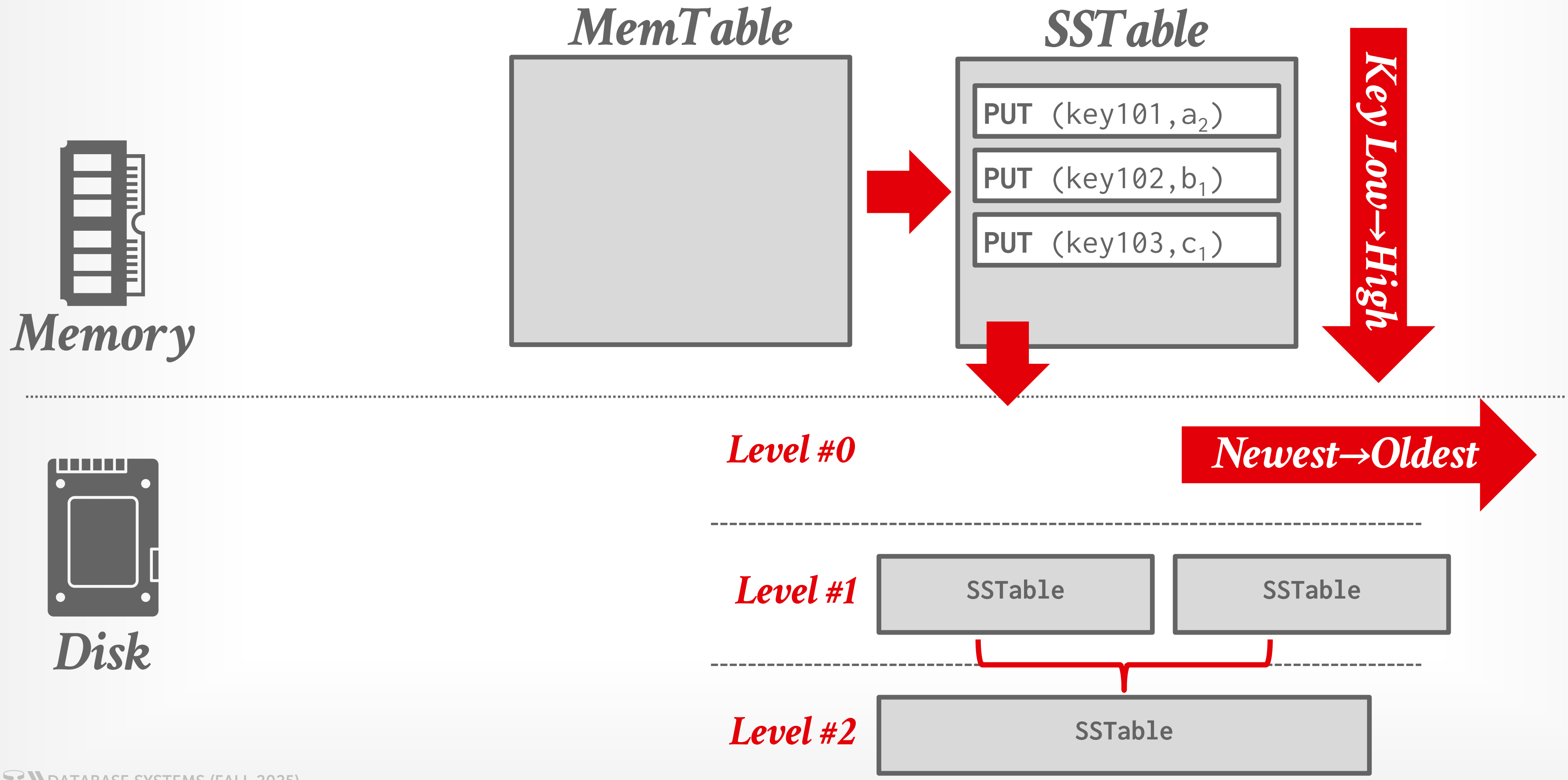
LOG-STRUCTURED STORAGE



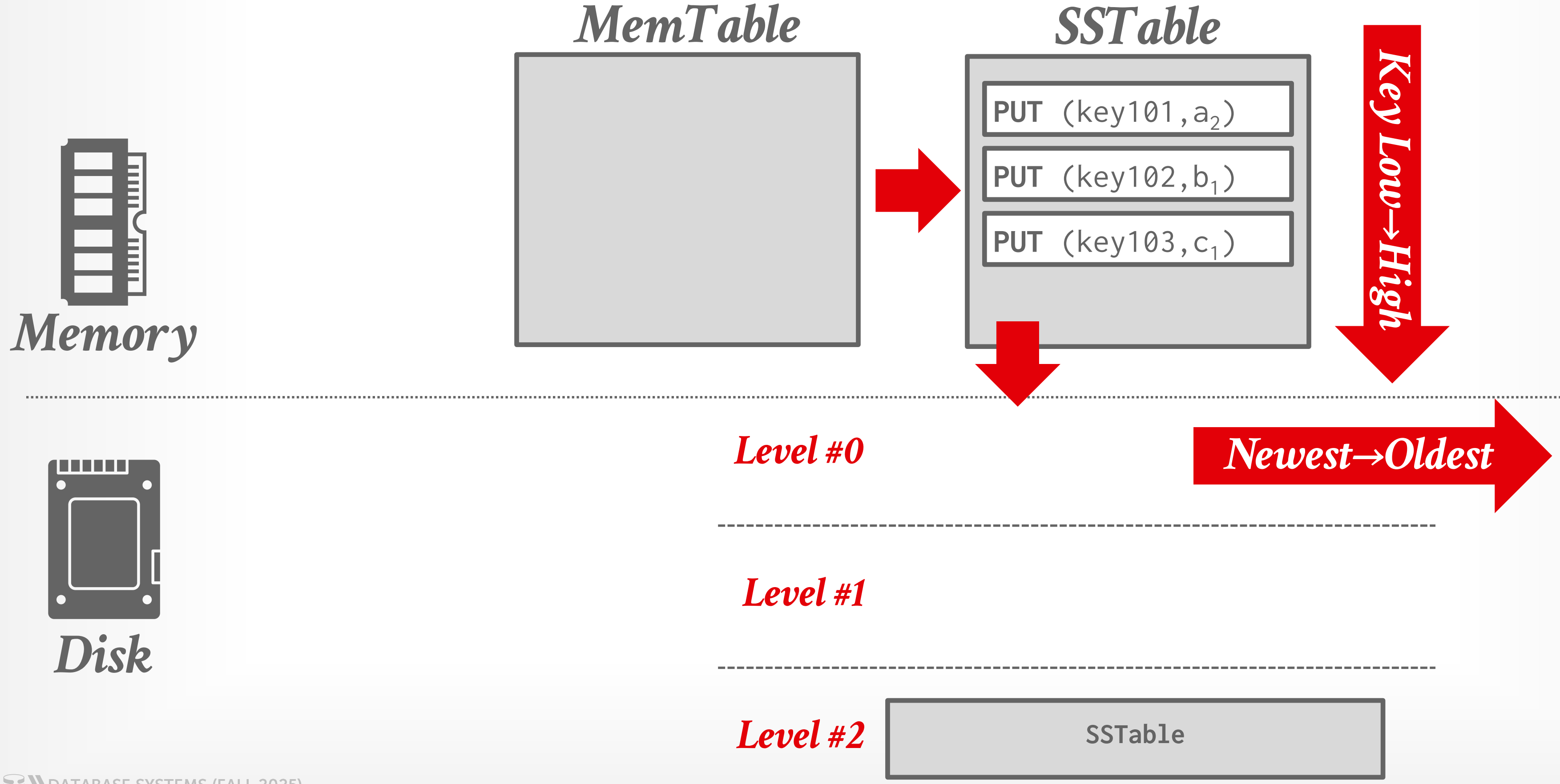
LOG-STRUCTURED STORAGE



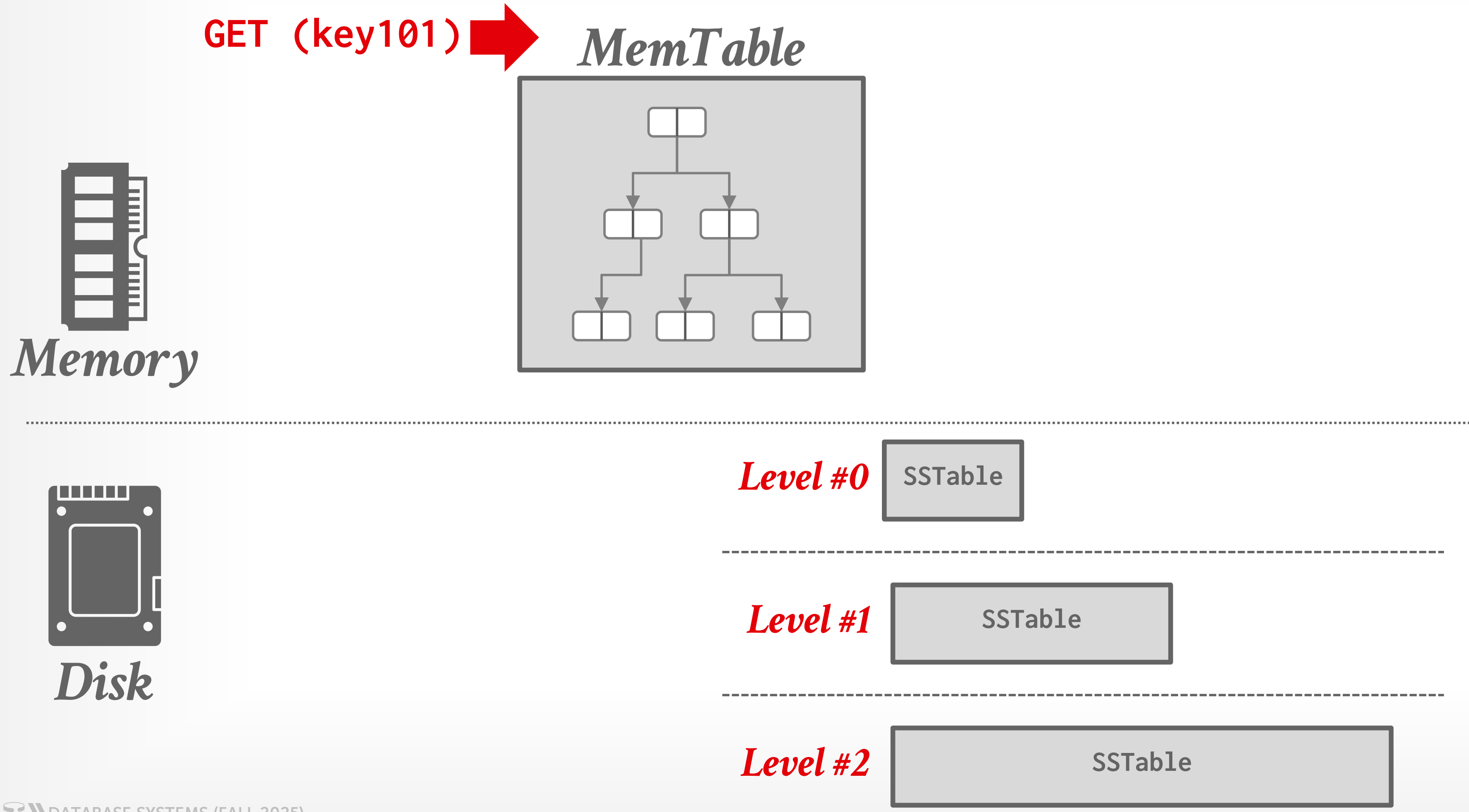
LOG-STRUCTURED STORAGE



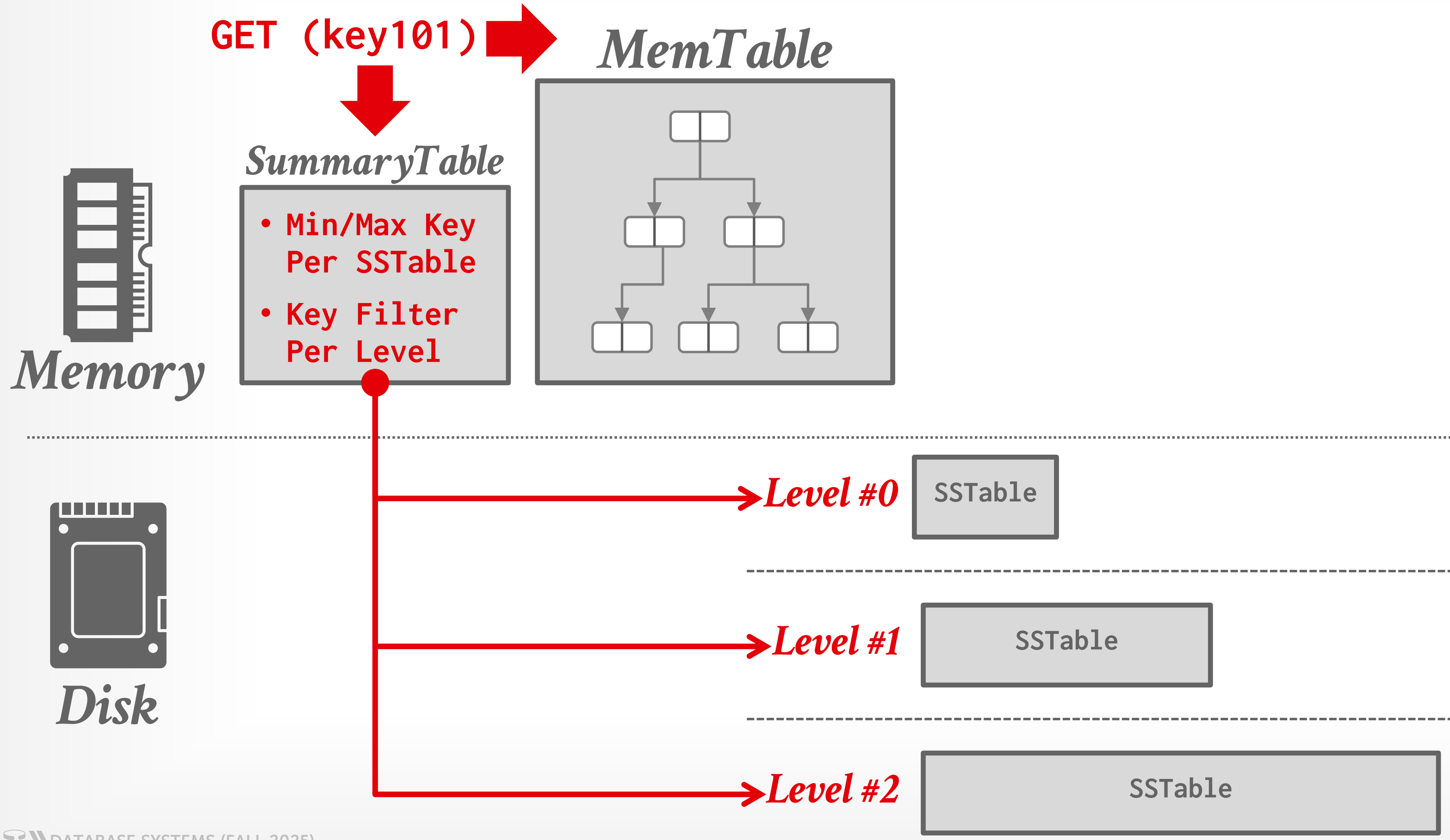
LOG-STRUCTURED STORAGE



LOG-STRUCTURED STORAGE



LOG-STRUCTURED STORAGE

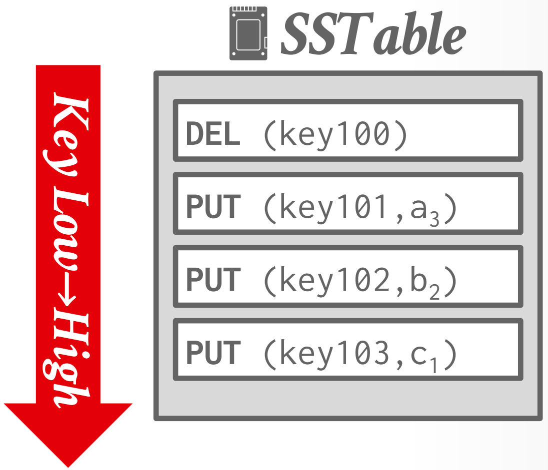


LOG-STRUCTURED STORAGE

Key-value storage that appends log records on disk to represent changes to tuples (**PUT**, **DELETE**).

→ Each log record must contain the tuple's unique identifier.
→ Put records contain the tuple contents.
→ Deletes marks the tuple as deleted.

As the application makes changes to the database, the DBMS appends log records to the end of the file without checking previous log records.



LOG-STRUCTURED COMPACTION

Periodically compact data files to reduce wasted space and speed up reads.
→ Keep "latest" values for each key using a sort-merge algorithm.

SSTable

```
DEL (key100)
PUT (key101,a3)
PUT (key102,b2)
PUT (key103,c1)
```

\+

SSTable

```
PUT (key101,a2)
PUT (key102,b1)
DEL (key103)
PUT (key104,d2)
```

→

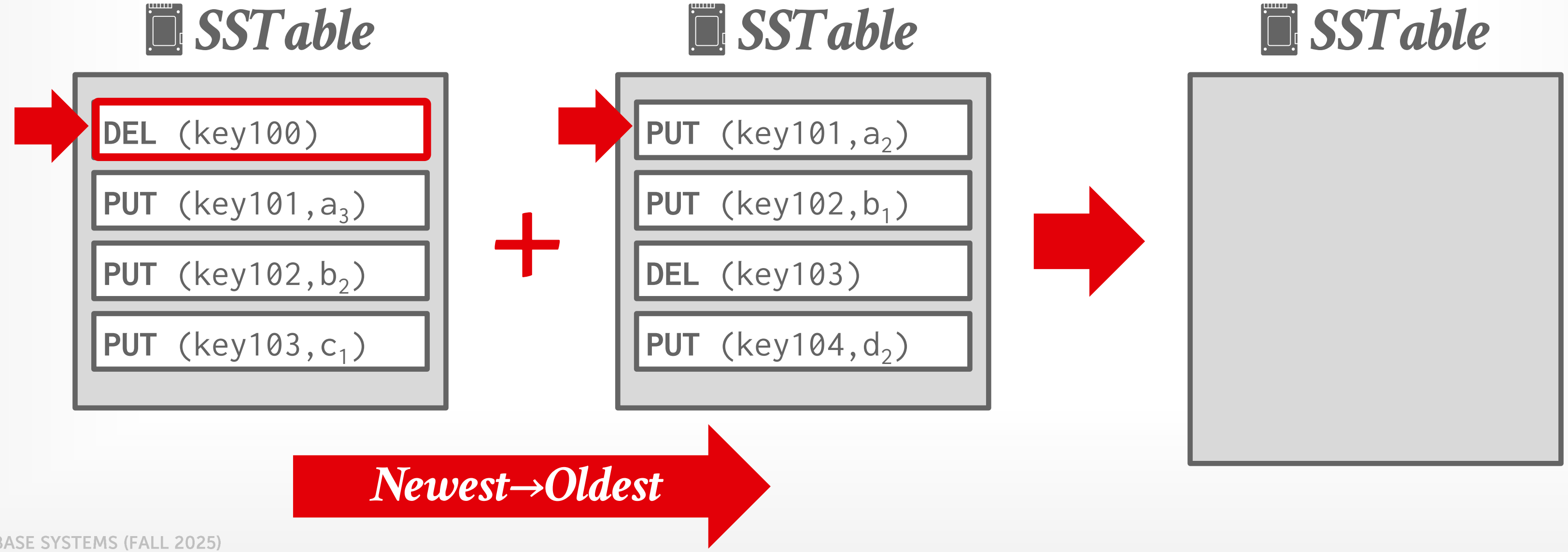
SSTable

Newest→Oldest

LOG-STRUCTURED COMPACTION

Periodically compact data files to reduce wasted space and speed up reads.

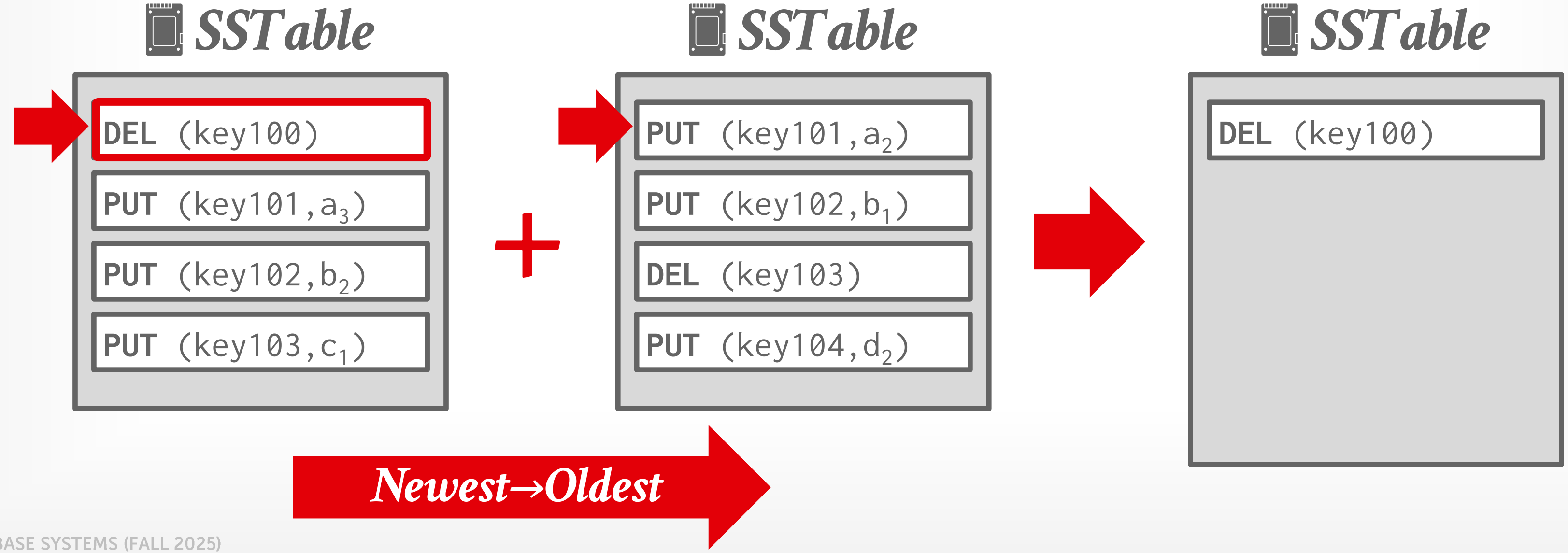
→ Keep "latest" values for each key using a sort-merge algorithm.



LOG-STRUCTURED COMPACTION

Periodically compact data files to reduce wasted space and speed up reads.

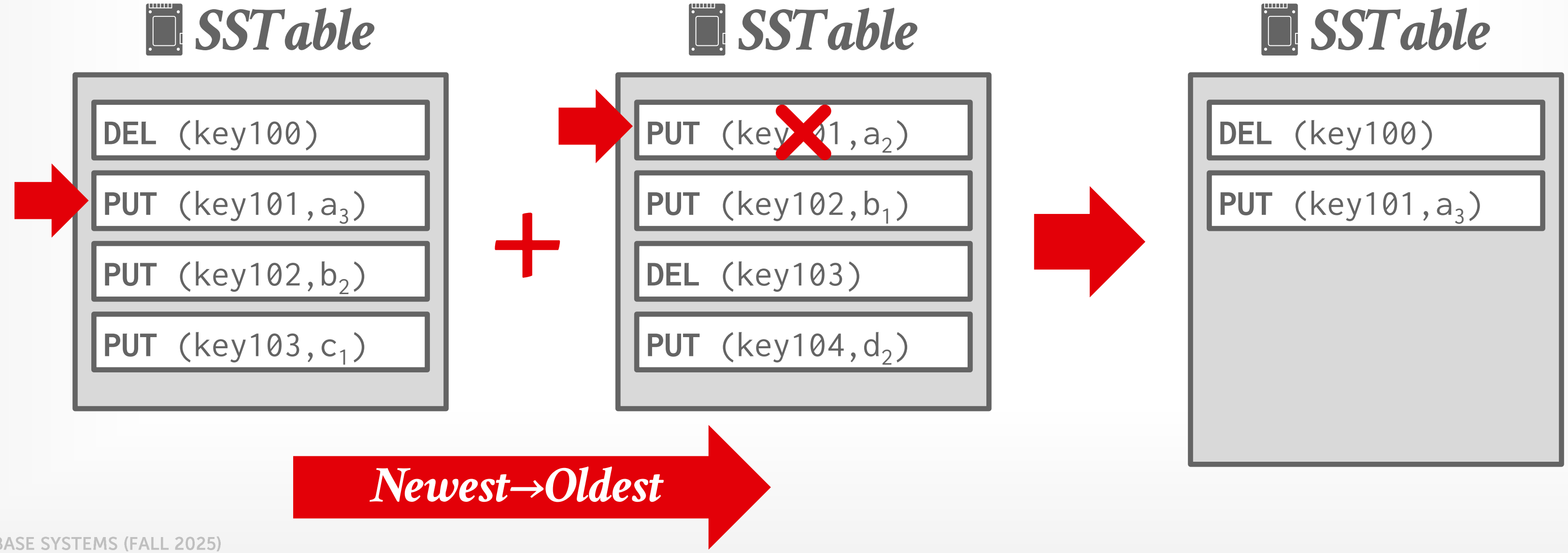
→ Keep "latest" values for each key using a sort-merge algorithm.



LOG-STRUCTURED COMPACTION

Periodically compact data files to reduce wasted space and speed up reads.

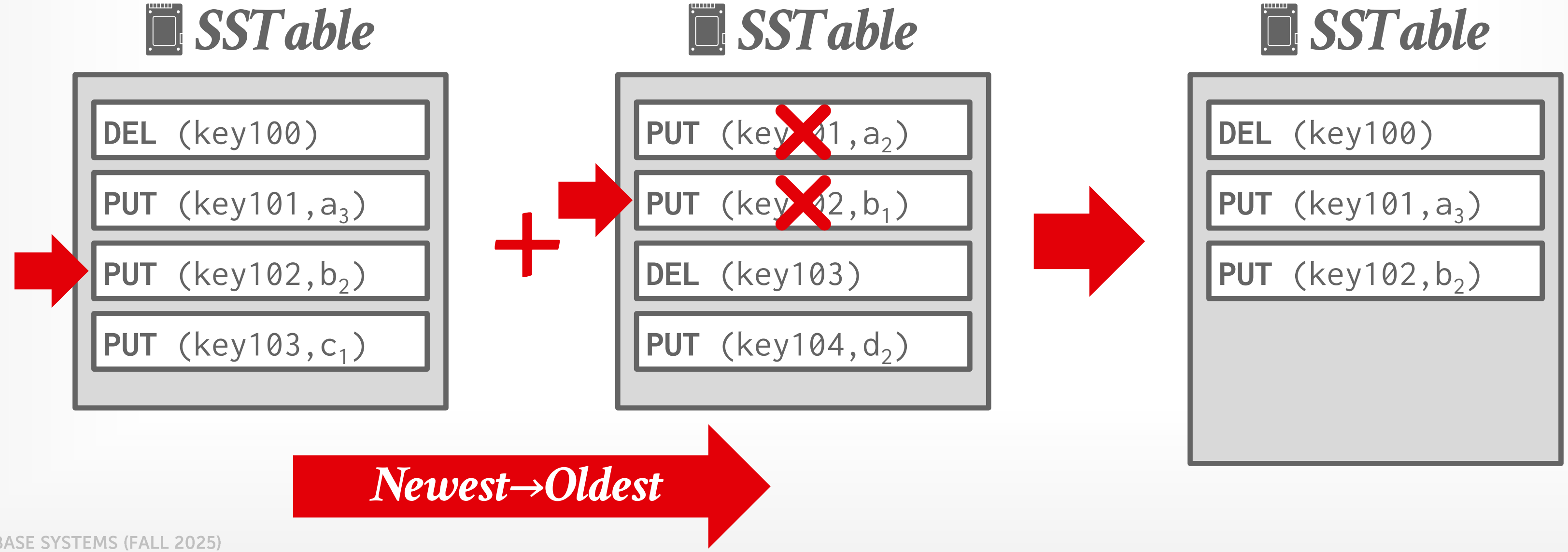
→ Keep "latest" values for each key using a sort-merge algorithm.



LOG-STRUCTURED COMPACTION

Periodically compact data files to reduce wasted space and speed up reads.

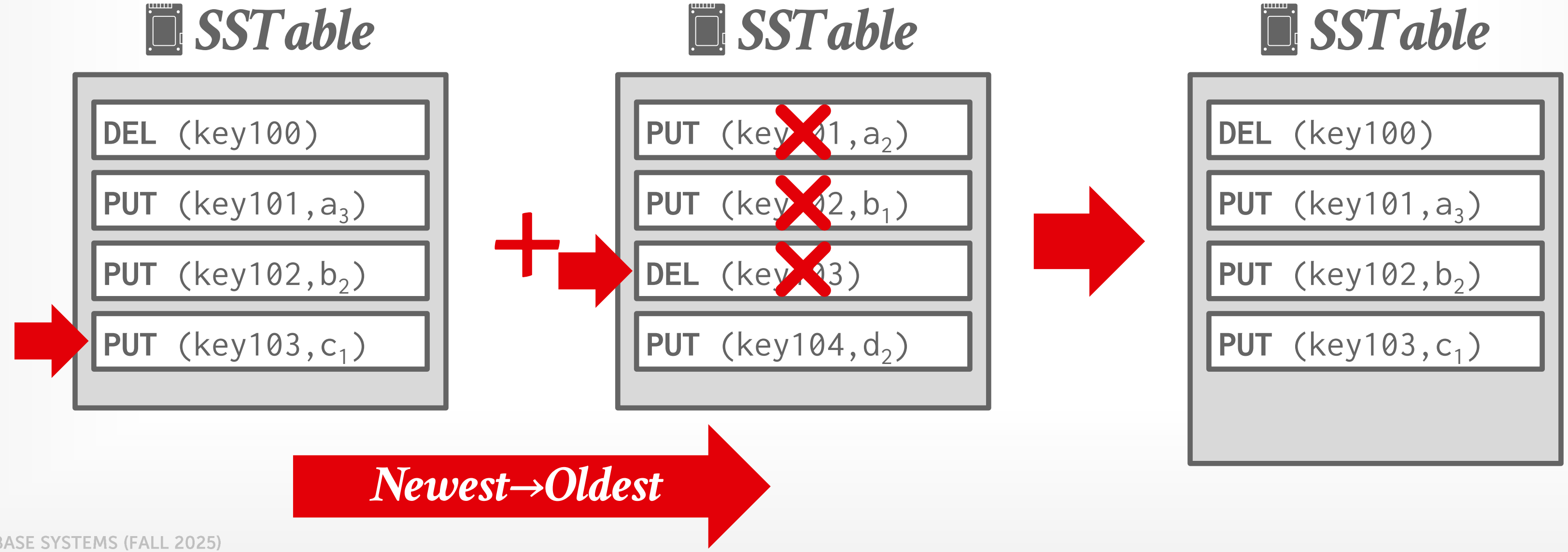
→ Keep "latest" values for each key using a sort-merge algorithm.



LOG-STRUCTURED COMPACTION

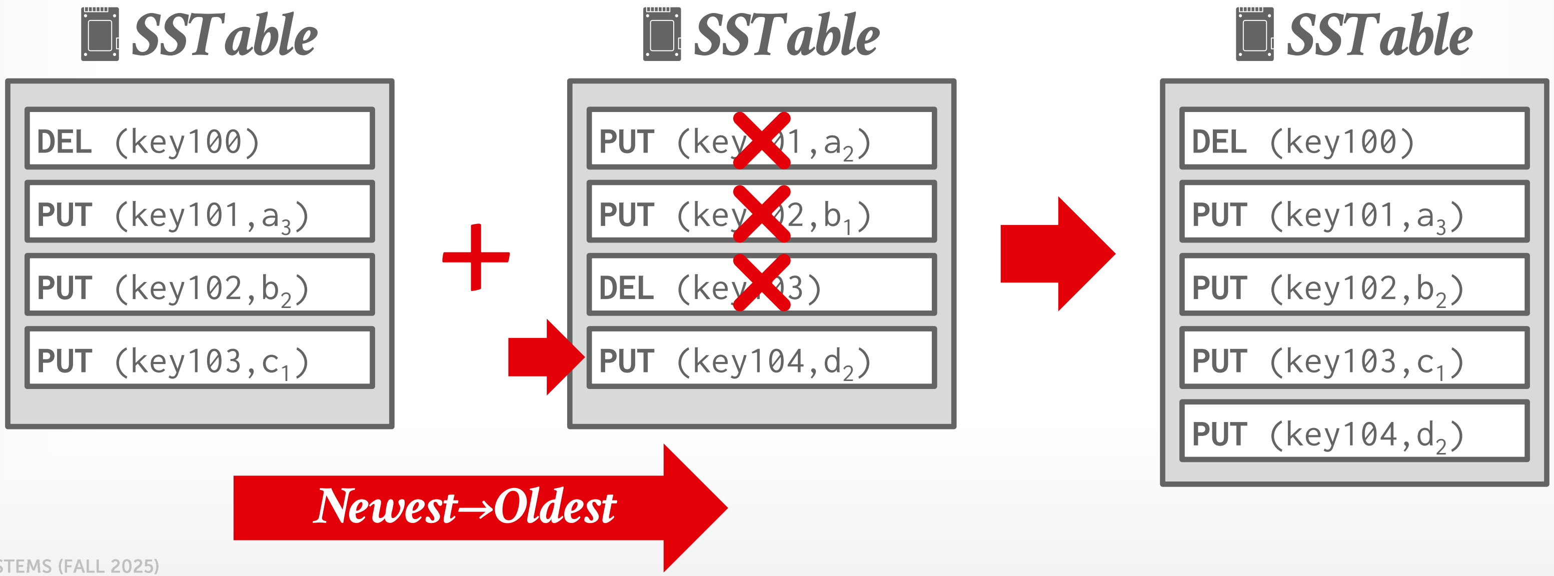
Periodically compact data files to reduce wasted space and speed up reads.

→ Keep "latest" values for each key using a sort-merge algorithm.



LOG-STRUCTURED COMPACTION

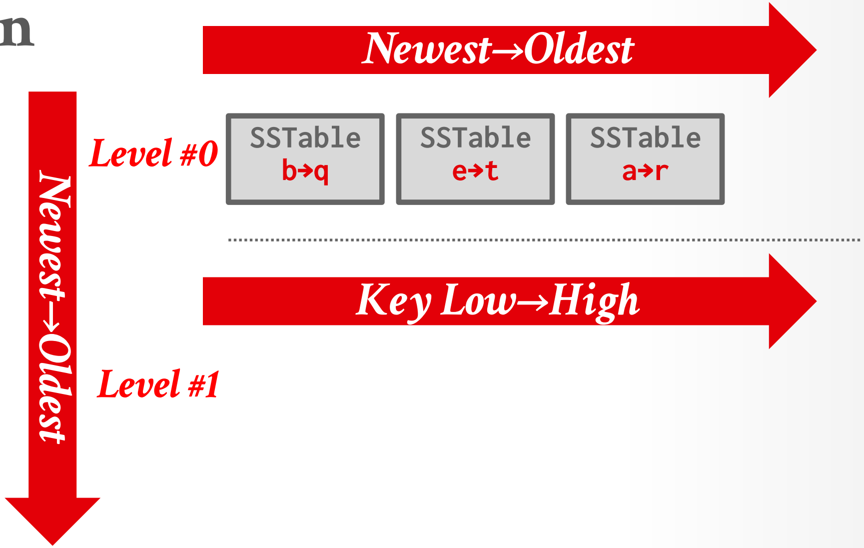
Periodically compact data files to reduce wasted space and speed up reads.
→ Keep "latest" values for each key using a sort-merge algorithm.



LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.
→ SSTables in a level are non-overlapping on key ranges (except Level #0).
→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.
→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.



Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

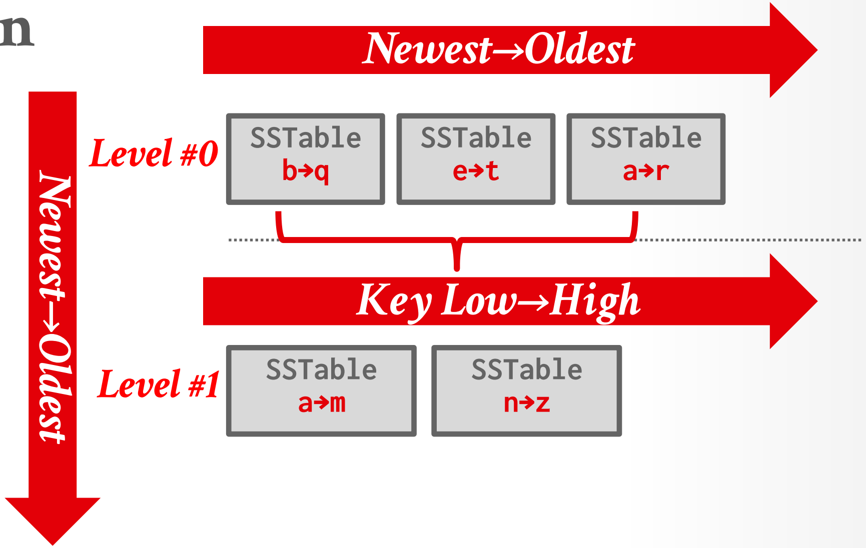
Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.

→ SSTables in a level are non-overlapping on key ranges (except Level #0).

→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.

→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.

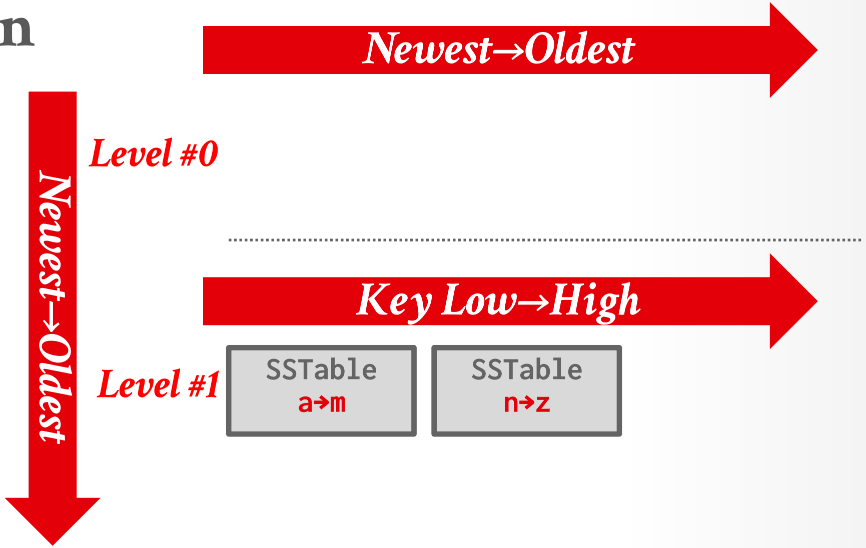


Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.
→ SSTables in a level are non-overlapping on key ranges (except Level #0).
→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.
→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.



Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

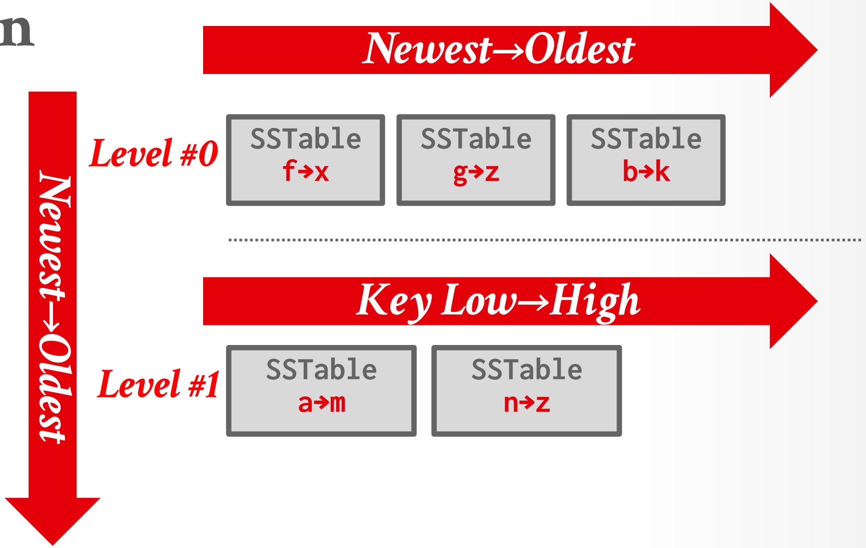
Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.

→ SSTables in a level are non-overlapping on key ranges (except Level #0).

→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.

→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.



Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

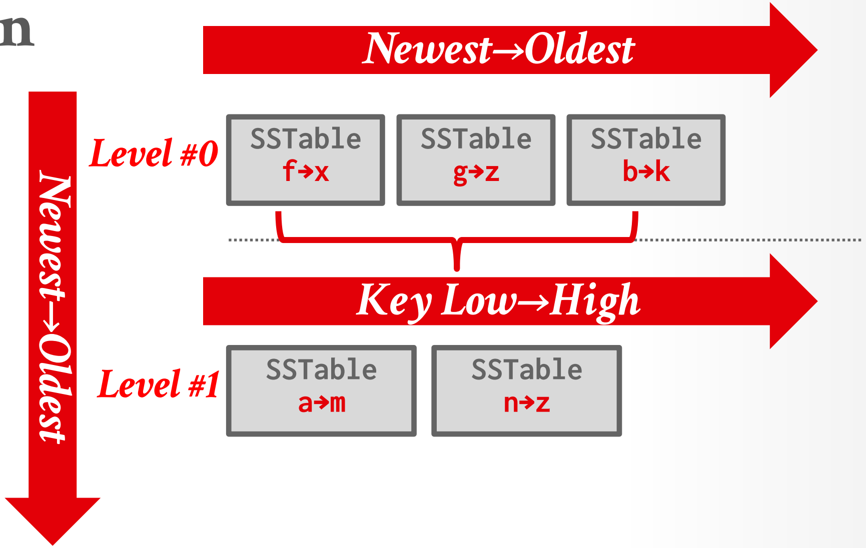
Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.

→ SSTables in a level are non-overlapping on key ranges (except Level #0).

→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.

→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.

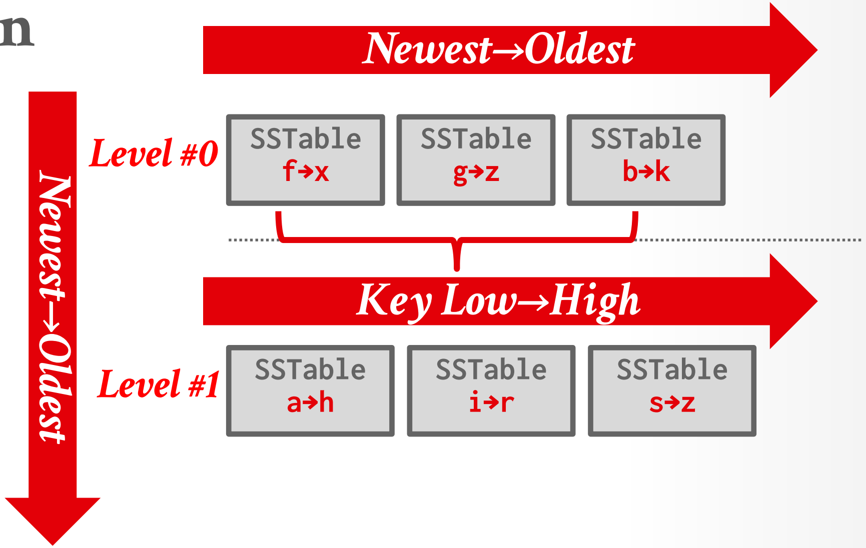


Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.
→ SSTables in a level are non-overlapping on key ranges (except Level #0).
→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.
→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.



Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

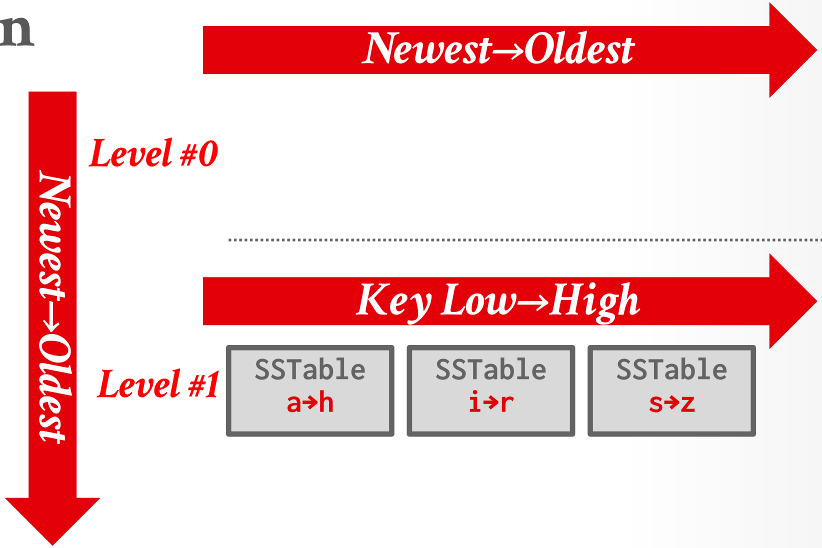
Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.

→ SSTables in a level are non-overlapping on key ranges (except Level #0).

→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.

→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.

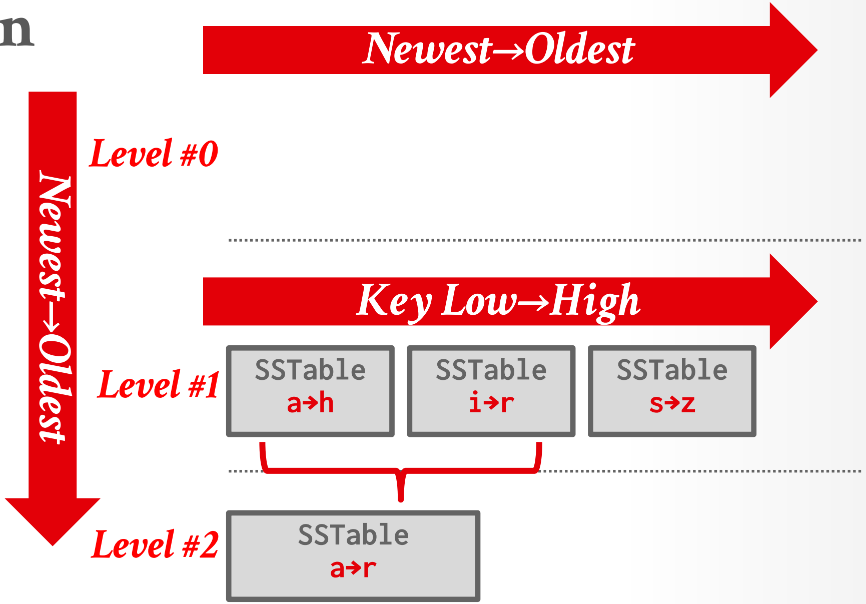


Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.
→ SSTables in a level are non-overlapping on key ranges (except Level #0).
→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.
→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.



Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

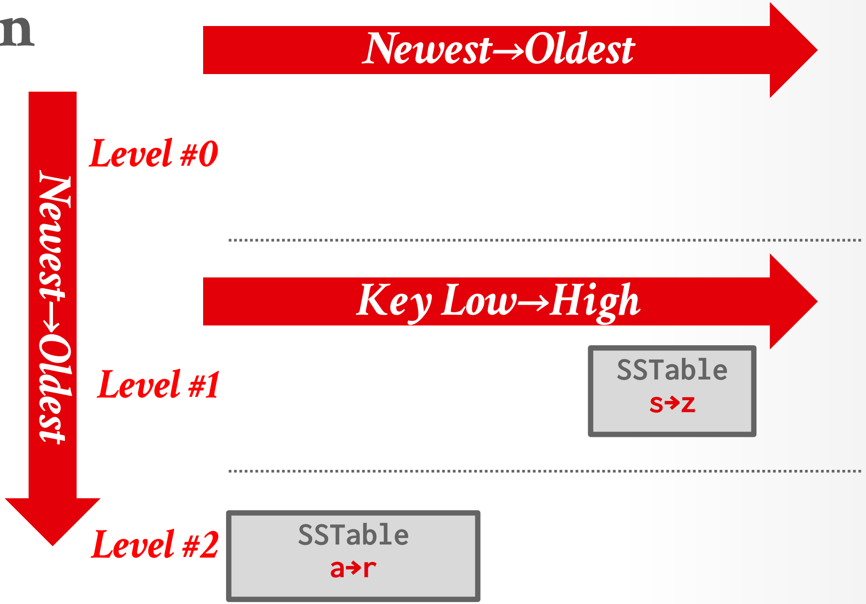
Approach #1: Leveled Compaction

→ Data is organized into levels with SSTable size limit per level.

→ SSTables in a level are non-overlapping on key ranges (except Level #0).

→ Level #0 contains SSTables recently flushed from memory and contain overlapping ranges.

→ Compactions merge a file from a level into the next lower level, maintaining sorted, non-overlapping key ranges.



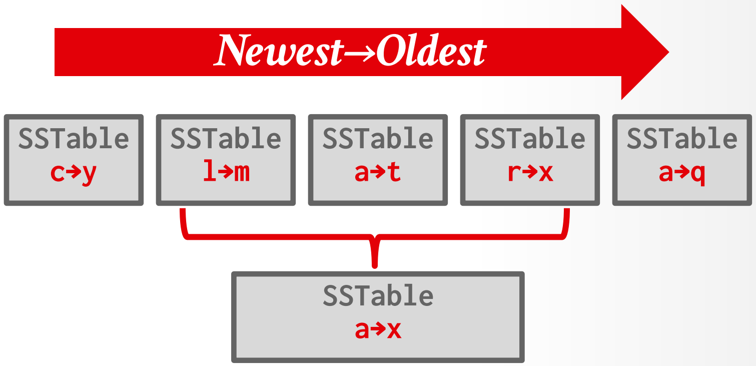
Better for read-heavy workloads.

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Universal Compaction

→ SSTables reside in a single "universal" level (i.e., no multi-level hierarchy).

→ DBMS triggers compaction when too many SSTables overlap in key ranges or exceed size thresholds.



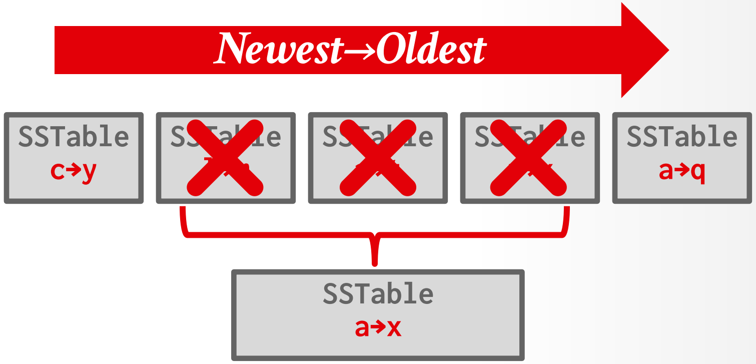
Better for insert-heavy workloads and time-oriented queries

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Universal Compaction

→ SSTables reside in a single "universal" level (i.e., no multi-level hierarchy).

→ DBMS triggers compaction when too many SSTables overlap in key ranges or exceed size thresholds.

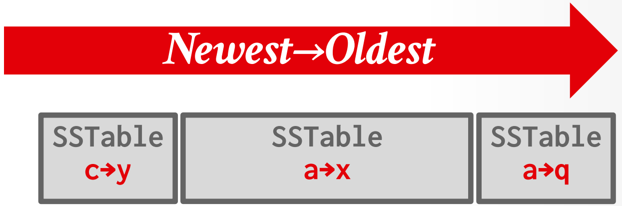


Better for insert-heavy workloads and time-oriented queries

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Universal Compaction
→ SSTables reside in a single "universal" level (i.e., no multi-level hierarchy).
→ DBMS triggers compaction when too many SSTables overlap in key ranges or exceed size thresholds.

Better for insert-heavy workloads and time-oriented queries

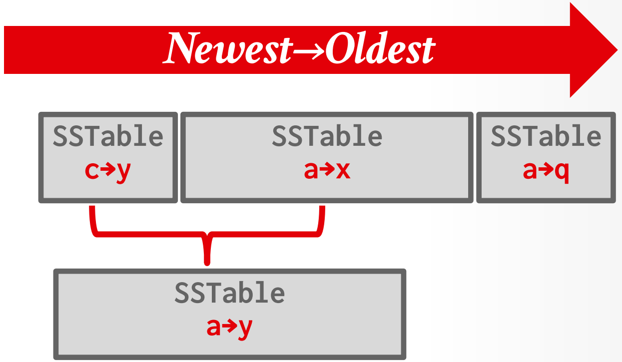


LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Universal Compaction

→ SSTables reside in a single "universal" level (i.e., no multi-level hierarchy).

→ DBMS triggers compaction when too many SSTables overlap in key ranges or exceed size thresholds.



Better for insert-heavy workloads and time-oriented queries

LOG-STRUCTURED COMPACTION STRATGEGIES

Approach #1: Universal Compaction

→ SSTables reside in a single "universal" level (i.e., no multi-level hierarchy).

→ DBMS triggers compaction when too many SSTables overlap in key ranges or exceed size thresholds.

Better for insert-heavy workloads and time-oriented queries

DISCUSSION

Log-structured storage managers are more common today than in previous decades.

→ This is partly due to the proliferation of RocksDB.

What are some downsides of this approach?

→ Write-Amplification.

→ Compaction is expensive.

CONCLUSION

Log-structured storage is an alternative approach to the tuple-oriented architecture.

→ Ideal for write-heavy workloads because it maximizes sequential disk I/O.

The storage manager is not entirely independent from the rest of the DBMS.

NEXT CLASS

Breaking your preconceived notion that a DBMS stores everything as rows…

PROJECT #1

You will build the first component of your storage manager.

→ ARC Replacement Policy
→ Disk Scheduler
→ Buffer Pool Manager Instance

We provide you with the basic APIs for these components.

BusTub

Due Date:
Sunday Sept 29th @ 11:59pm

https://15445.courses.cs.cmu.edu/fall2025/project1

TASK #1 — ARC REPLACEMENT POLICY

Build a data structure that tracks the usage of pages using the ARC policy. Dynamically adjust whether to favor recency or frequency in eviction decisions.

General Hints:

→ Your eviction algorithm needs to check the "pinned" status of each page.

→ You are allowed to use STL containers for internal lists (e.g., MRU, MFU).

TASK #2 — DISK SCHEDULER

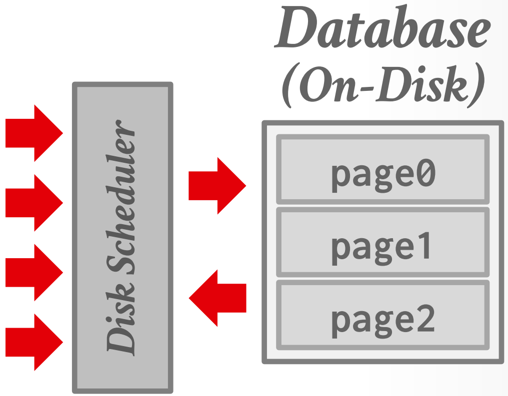
Create a background worker to read/write pages from disk.

→ Single request queue but each request can contain multiple requested pages.

→ Simulates asynchronous IO using **`std::promise`** for callbacks.

It's up to you to decide how you want to batch, reorder, and issue read/write requests to the local disk.

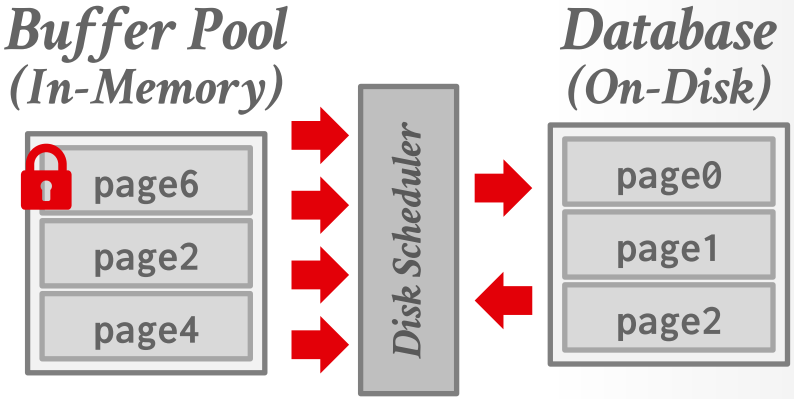
Make sure it is thread-safe!



TASK #3 — BUFFER POOL MANAGER

Use your ARC replacer to manage the allocation of pages.

→ Need to maintain internal data structures to track allocated + free pages.
→ Implement page guards.
→ Use whatever data structure you want for the page table.



Make sure you get the order of operations correct when pinning!

THINGS TO NOTE

Do **not** change any file other than the ones listed in the project specification. Other changes will **not** be graded.

The projects are cumulative, and we do **not** provide solutions.

Post any questions on Piazza or come to office hours, but we will **not** help you debug.

CODE QUALITY

We will automatically check whether you are writing good code.

→ Google C++ Style Guide
→ Doxygen Javadoc Style

You need to run these targets before you submit your implementation to Gradescope.

→ `make format`
→ `make check-clang-tidy-p1`

EXTRA CREDIT

Gradescope Leaderboard runs your code with a specialized in-memory version of BusTub.

The top 20 fastest implementations in the class will receive extra credit for this assignment.
→ **#1:** 50% bonus points
→ **#2–10:** 25% bonus points
→ **#11–20:** 10% bonus points

The student with the most bonus points at the end of the semester will be added to the BusTub trophy!

PLAGIARISM WARNING

The homework and projects must be your own original work. They are not group assignments.

→ You may not copy source code from other people or the web.

→ You are allowed to use generative AI tools.

Plagiarism is not tolerated. You will get lit up.

→ Please ask instructors (not TAs!) if you are unsure.

See CMU's Policy on Academic Integrity for additional information.