DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2019 // JOY ARULRAJ

LECTURE #06: DISK-CENTRIC AND IN-MEMORY DATABASE SYSTEMS

CREATING THE NEXT®

ADMINISTRIVIA

- Project ideas
  - List shared on Piazza
  - Start looking for team-mates!
  - Sign up for discussion slots during office hours

LAST CLASS

- History of DBMSs
  - In a way though, it really was a history of data models
- Data Models
  - Hierarchical data model (tree) (IMS)
  - Network data model (graph) (CODASYL)
  - Relational data model (tables) (System R, INGRES)
- Overarching theme about all these systems
  - They were all disk-based DBMSs

TODAY'S AGENDA

- Disk-centric DBMSs
- In-Memory DBMSs

DISK-CENTRIC DBMSs

ANATOMY OF A DATABASE SYSTEM

Query

- Connection Manager + Admission Control — *Process Manager*
- Query Parser
- Query Optimizer
- Query Executor — *Query Processor*
- Lock Manager (Concurrency Control)
- Access Methods (or Indexes)
- Buffer Pool Manager
- Log Manager — *Transactional Storage Manager*
- Memory Manager + Disk Manager
- Networking Manager — *Shared Utilities*

Source: Anatomy of a Database System

ANATOMY OF A DATABASE SYSTEM

- Process Manager
  - Manages client connections
- Query Processor
  - Parse, plan and execute queries on top of storage manager
- Transactional Storage Manager
  - Knits together buffer management, concurrency control, logging and recovery
- Shared Utilities
  - Manage hardware resources across threads

TOPICS

- Implications of availability of large DRAM chips for database systems
  - Buffer Management
  - Query Processing
  - Concurrency Control
  - Logging and Recovery

BACKGROUND

- Much of the history of DBMSs is about dealing with the limitations of hardware.
- Hardware was much different when the original DBMSs were designed:
  - Uniprocessor (single-core CPU)
  - RAM was severely limited (few MB).
  - The database had to be stored on disk.
  - Disk is slow. No seriously, I mean really slow.

BACKGROUND

- But now DRAM capacities are large enough that most databases can fit in memory.
  - Structured data sets are smaller (e.g., tables with numeric data).
  - Unstructured data sets are larger (e.g., videos).
- So why not just use a "traditional" disk-oriented DBMS with a really large cache?

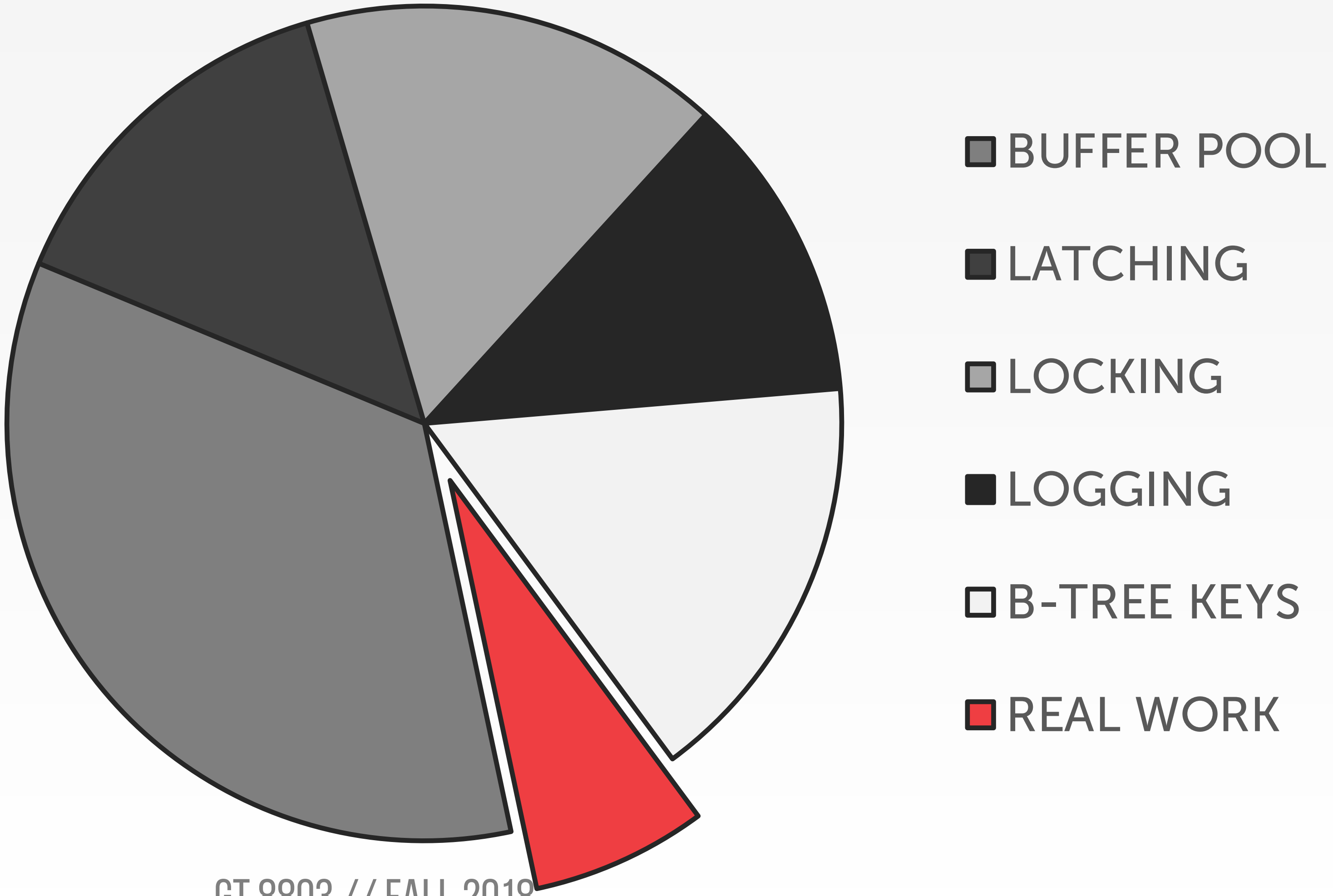
DISK-ORIENTED DBMS OVERHEAD

Measured CPU Instructions

OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

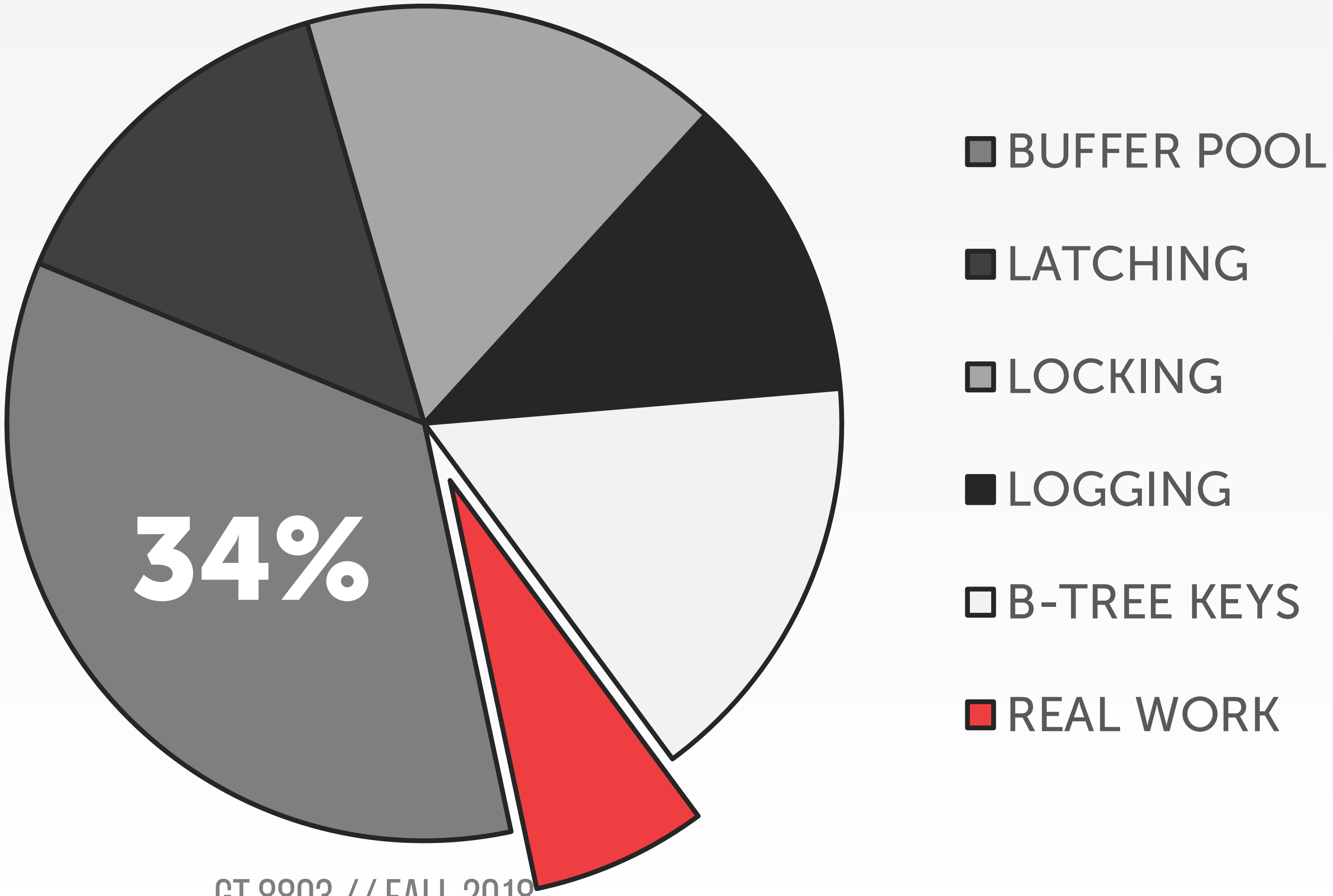
Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

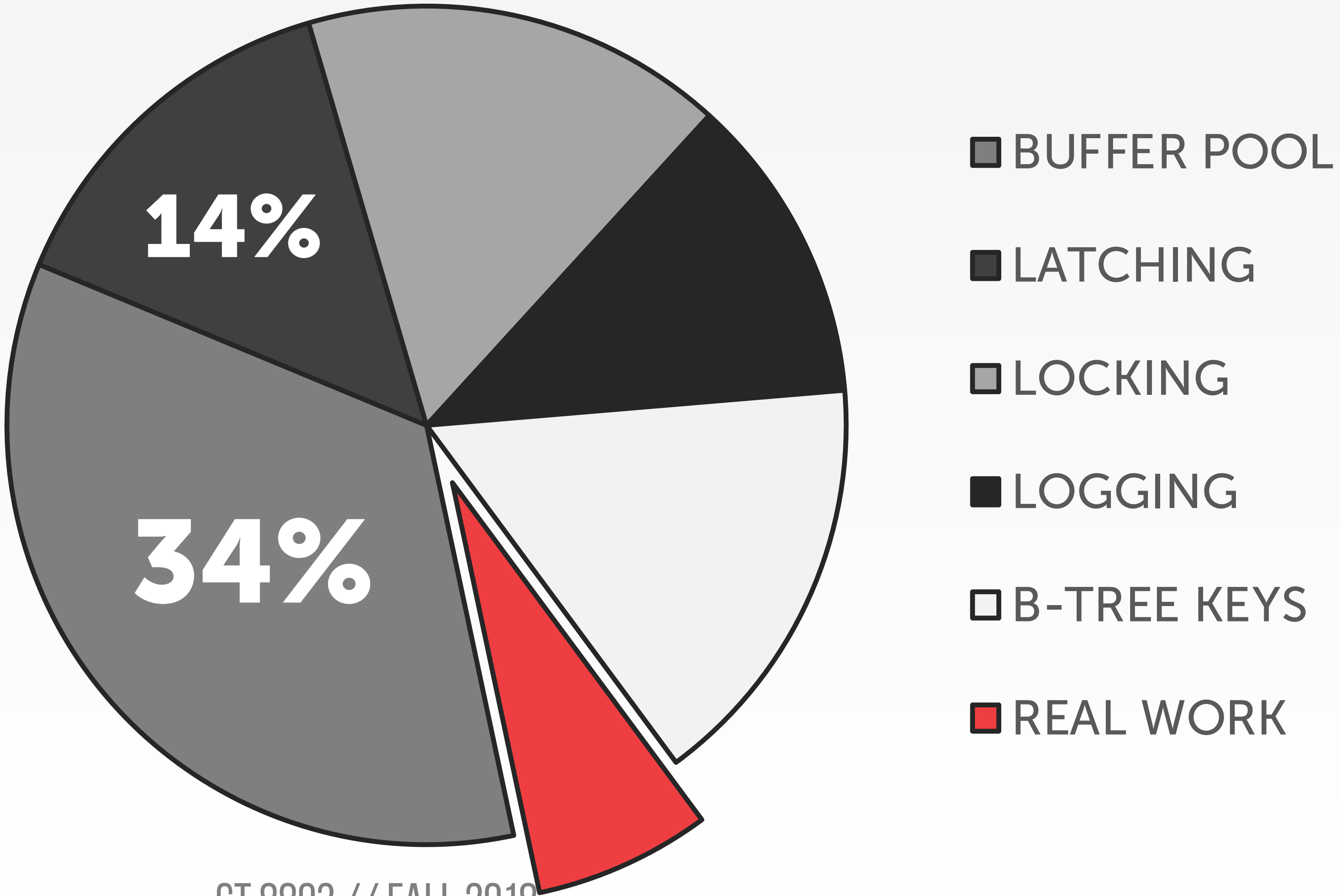
Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

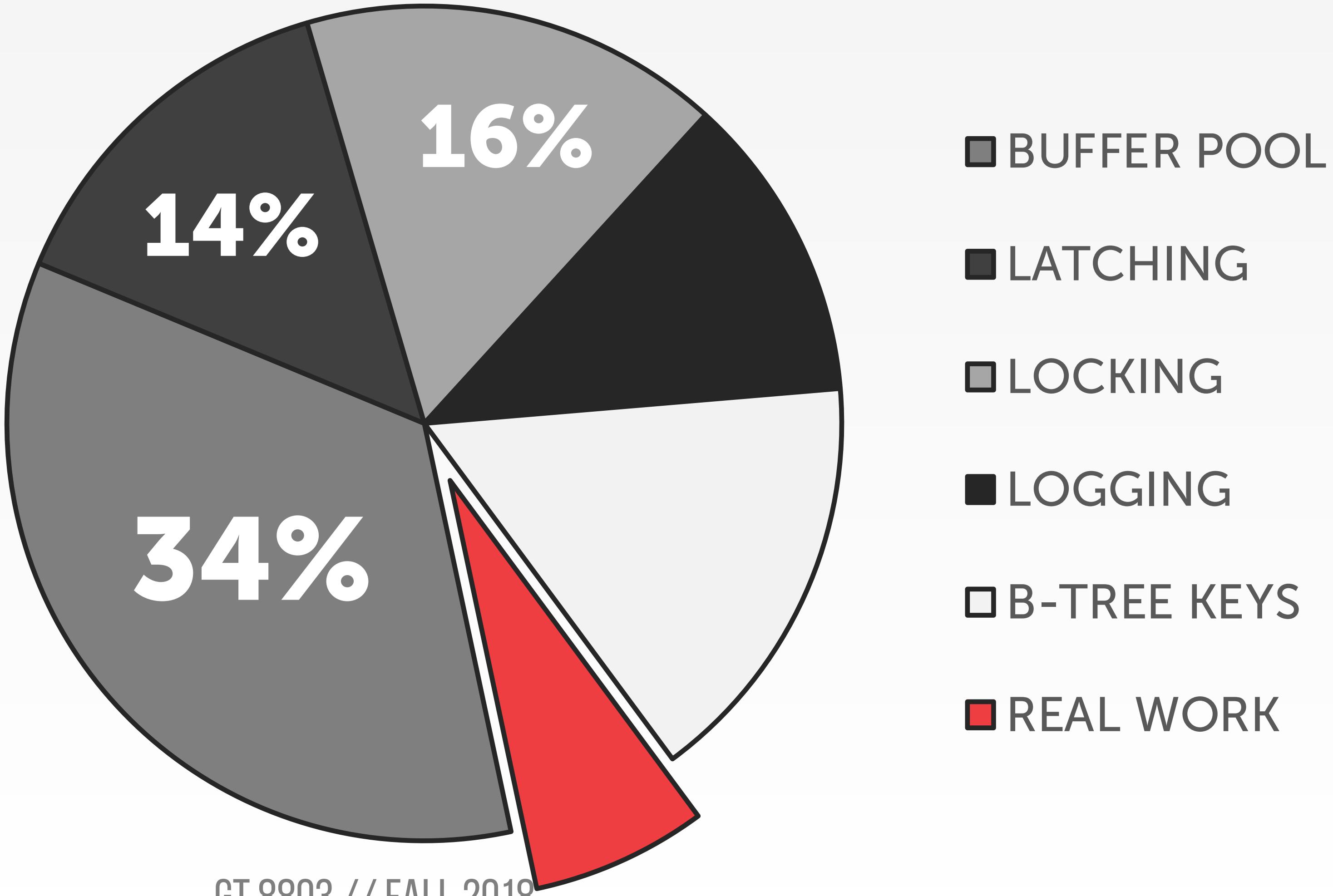
Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

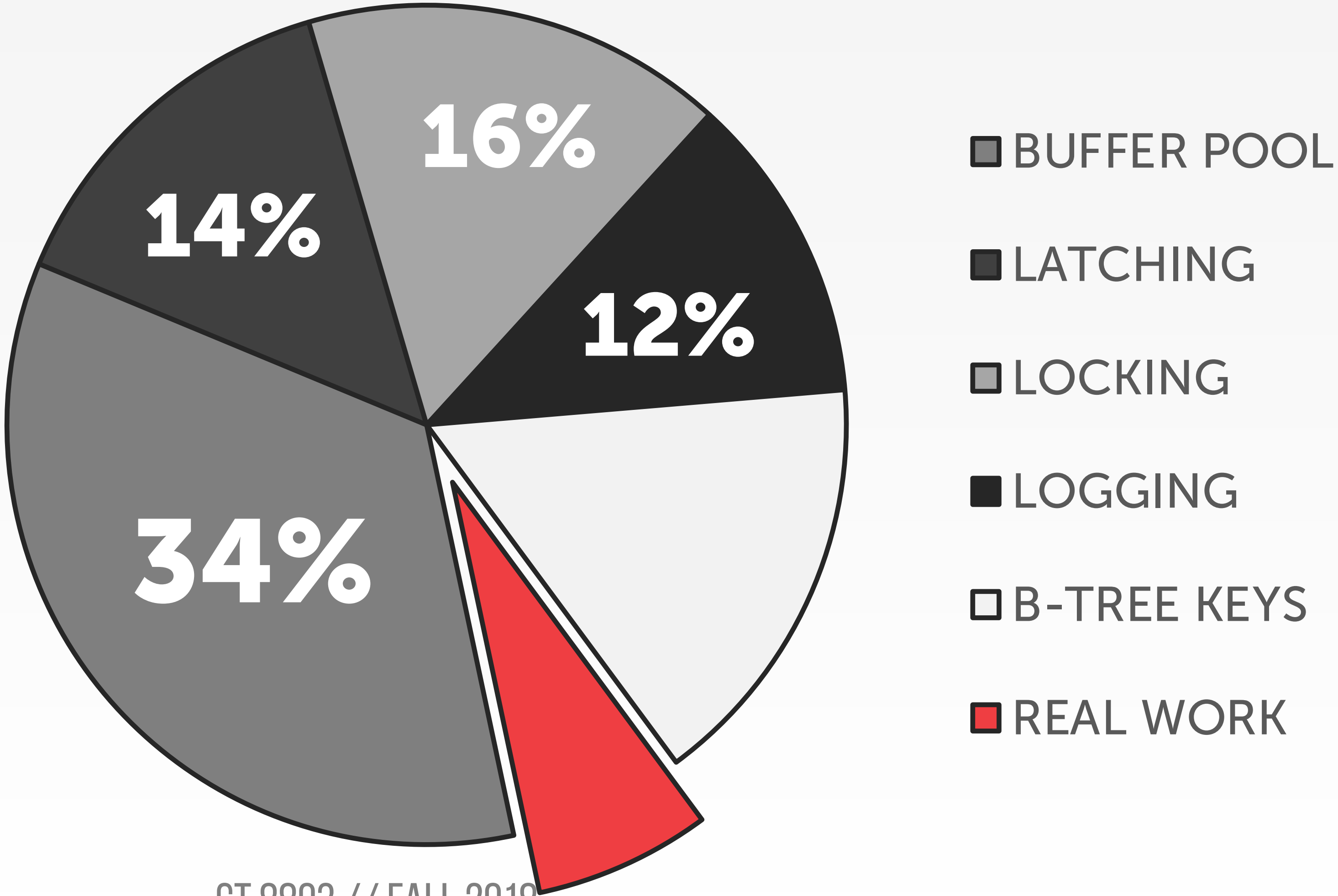
Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

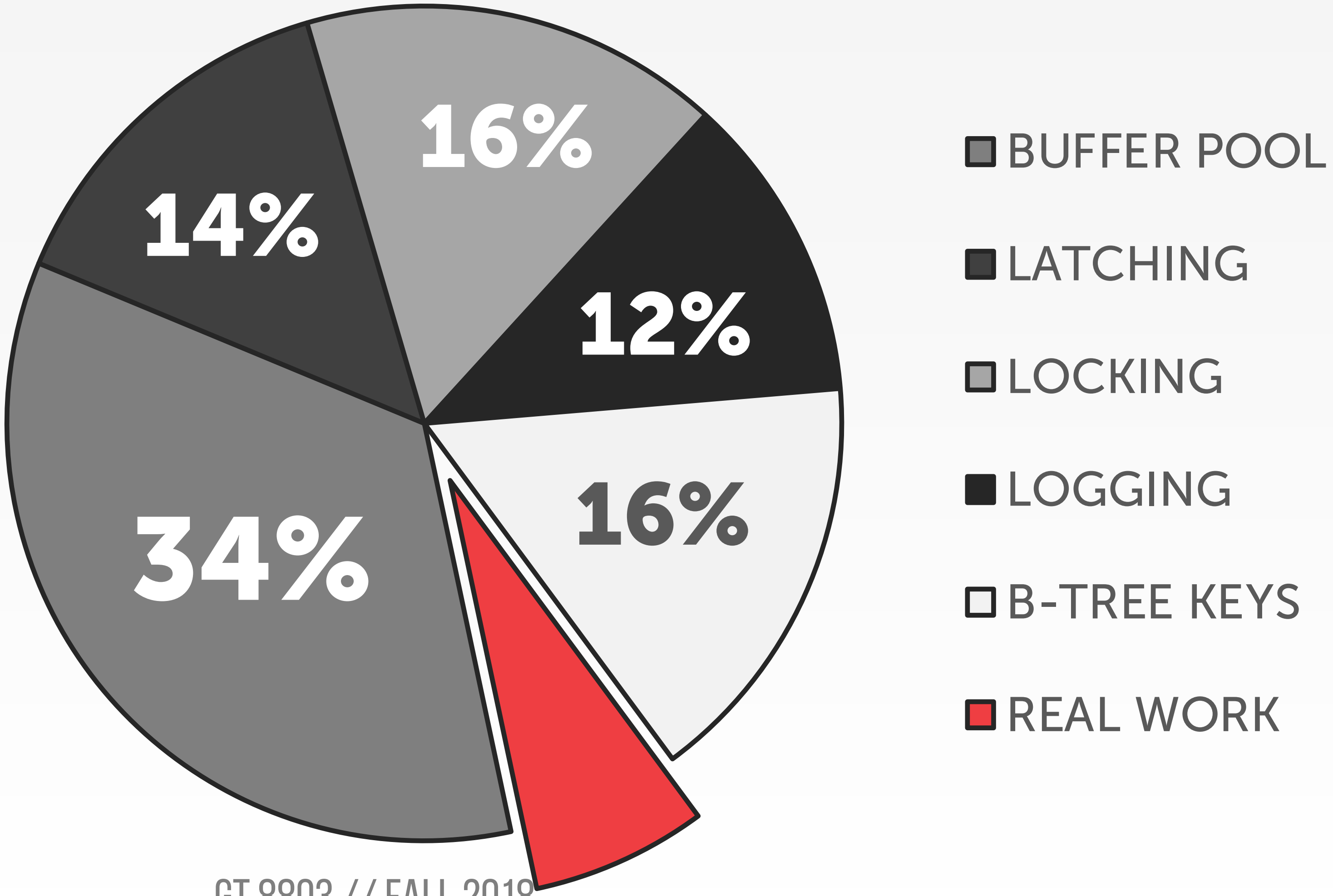
Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

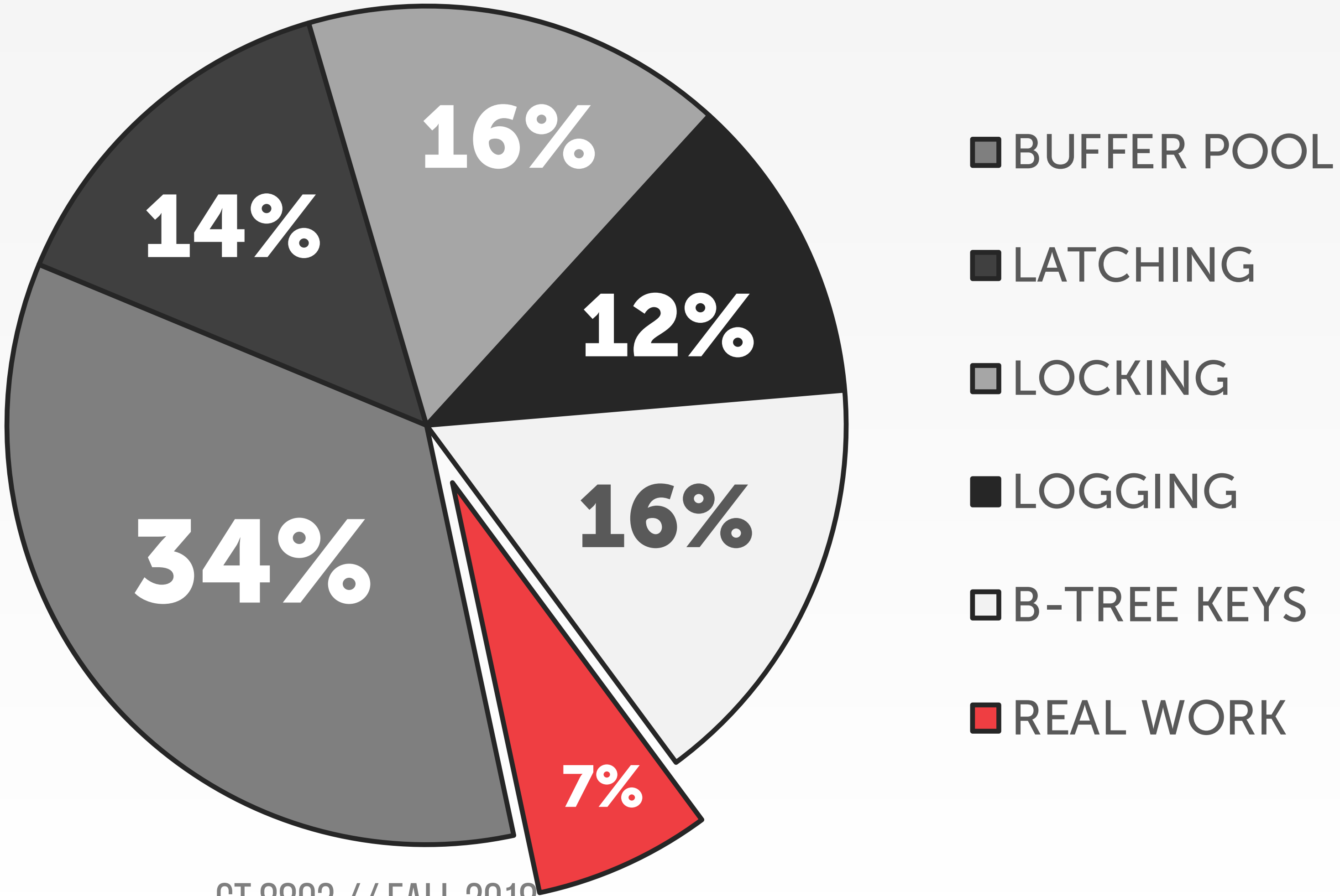
Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

DISK-ORIENTED DBMS OVERHEAD

Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

BUFFER MANAGEMENT

- The primary storage location of the database is on non-volatile storage (e.g., SSD).
  - The database is stored in a **file** as a collection of fixed-length blocks called **slotted pages** on disk.
- The system uses an volatile in-memory buffer pool to cache blocks fetched from disk.
  - Its job is to manage the movement of those blocks back and forth between disk and memory.

BUFFER MANAGEMENT

- When a query accesses a page, the DBMS checks to see if that page is already in memory in a **buffer pool**
  - If it's not, then the DBMS has to retrieve it from disk and copy it into a free frame in the buffer pool.
  - If there are no free frames, then find a page to evict guided by the **page replacement policy**.
  - If the page being evicted is dirty, then the DBMS has to write it back to disk to ensure the **durability** (ACI**D**) of data.

BUFFER MANAGEMENT

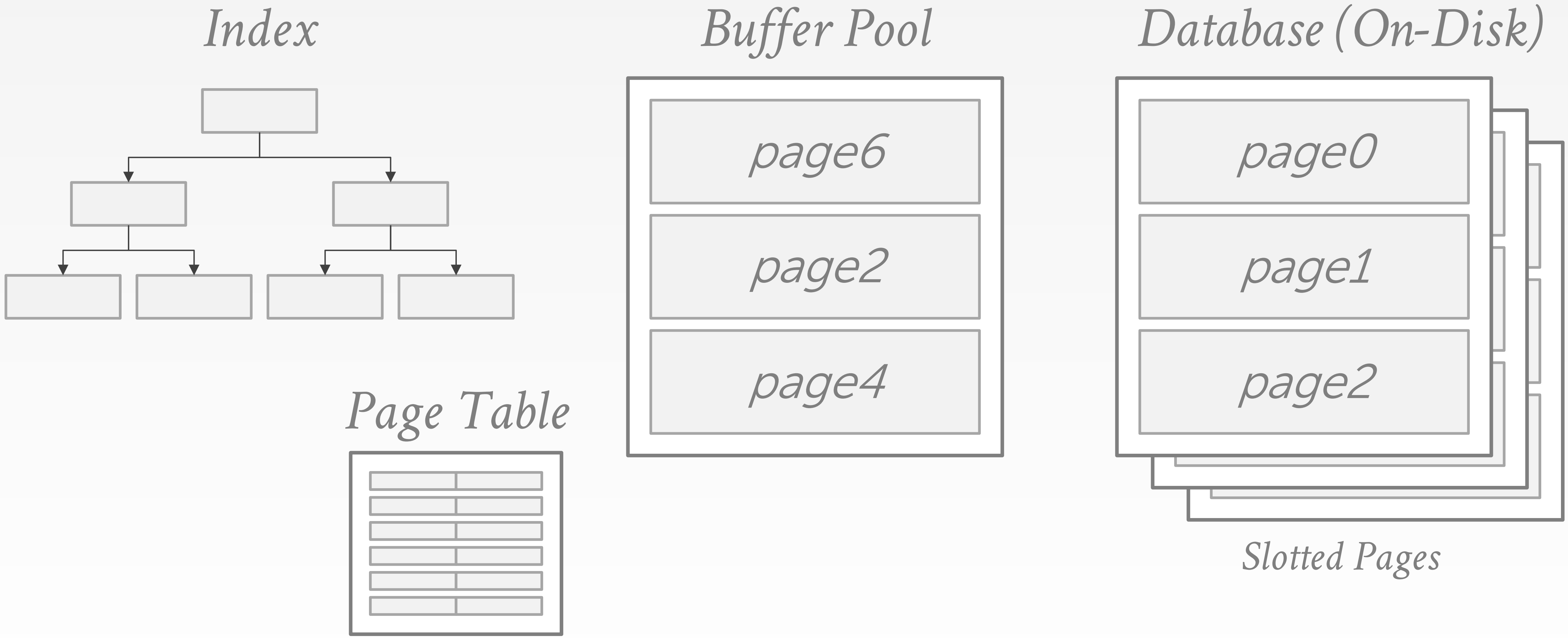
- Page replacement policy is a differentiating factor between open-source and commercial DBMSs.
  - What kind of data does it contain?
  - Is the page dirty?
  - How likely is the page to be accessed in the near future?
  - Examples: LRU, LFU, CLOCK, ARC

BUFFER MANAGEMENT

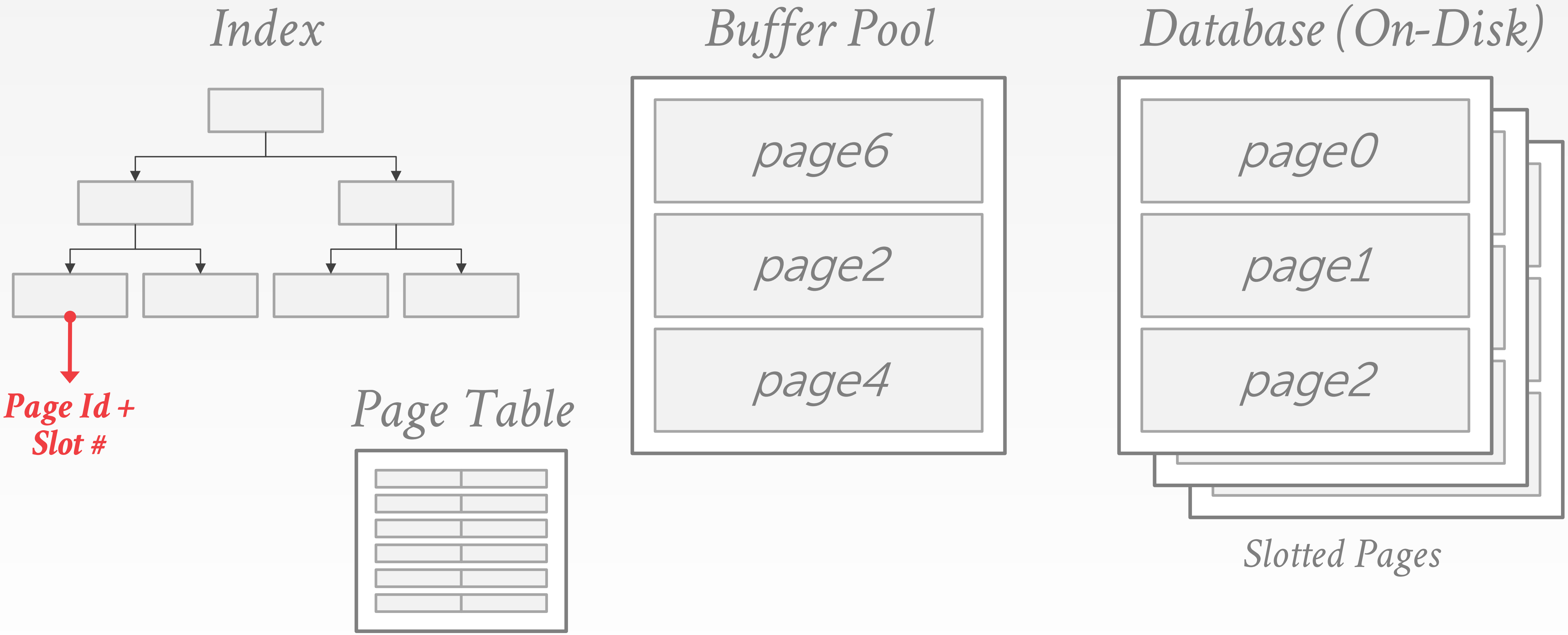
- Once the page is in memory, the DBMS translates any on-disk addresses to their in-memory addresses.

(Page Identifier)	(Page Pointer)
[#100]	[0x5050]

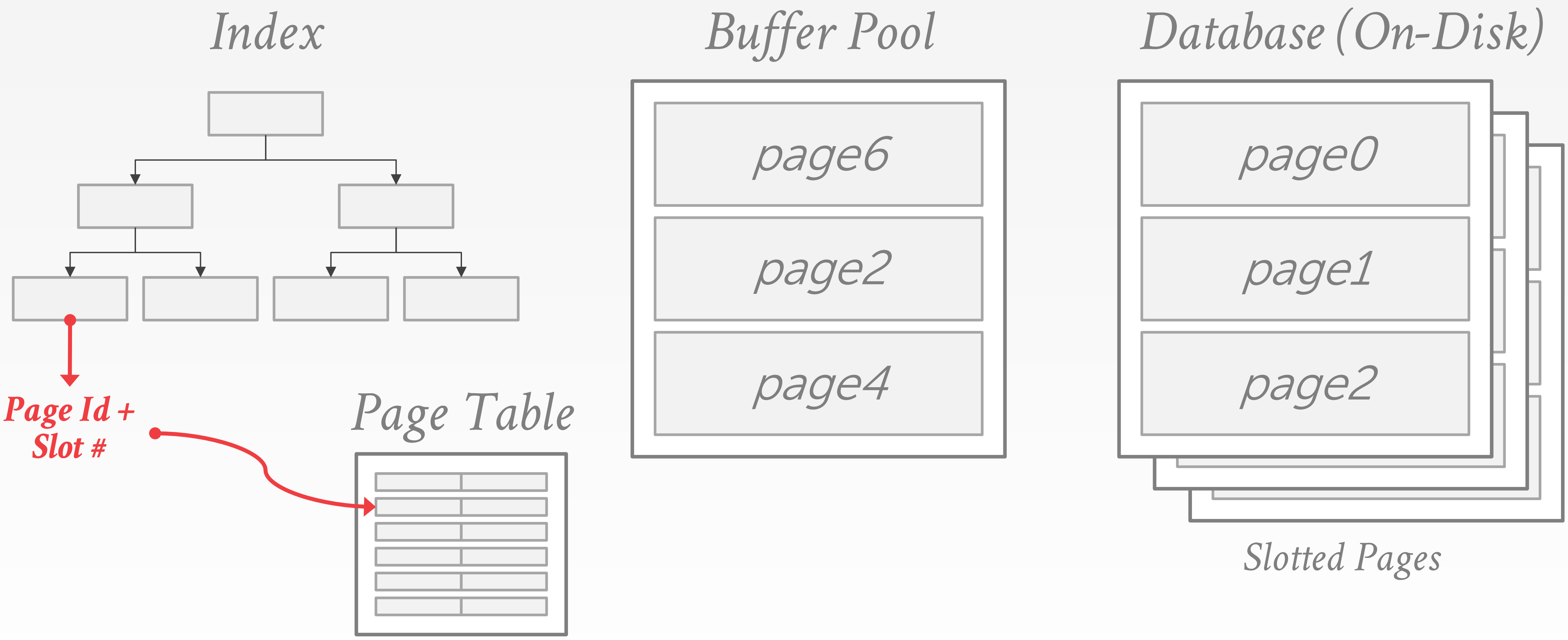
BUFFER MANAGEMENT



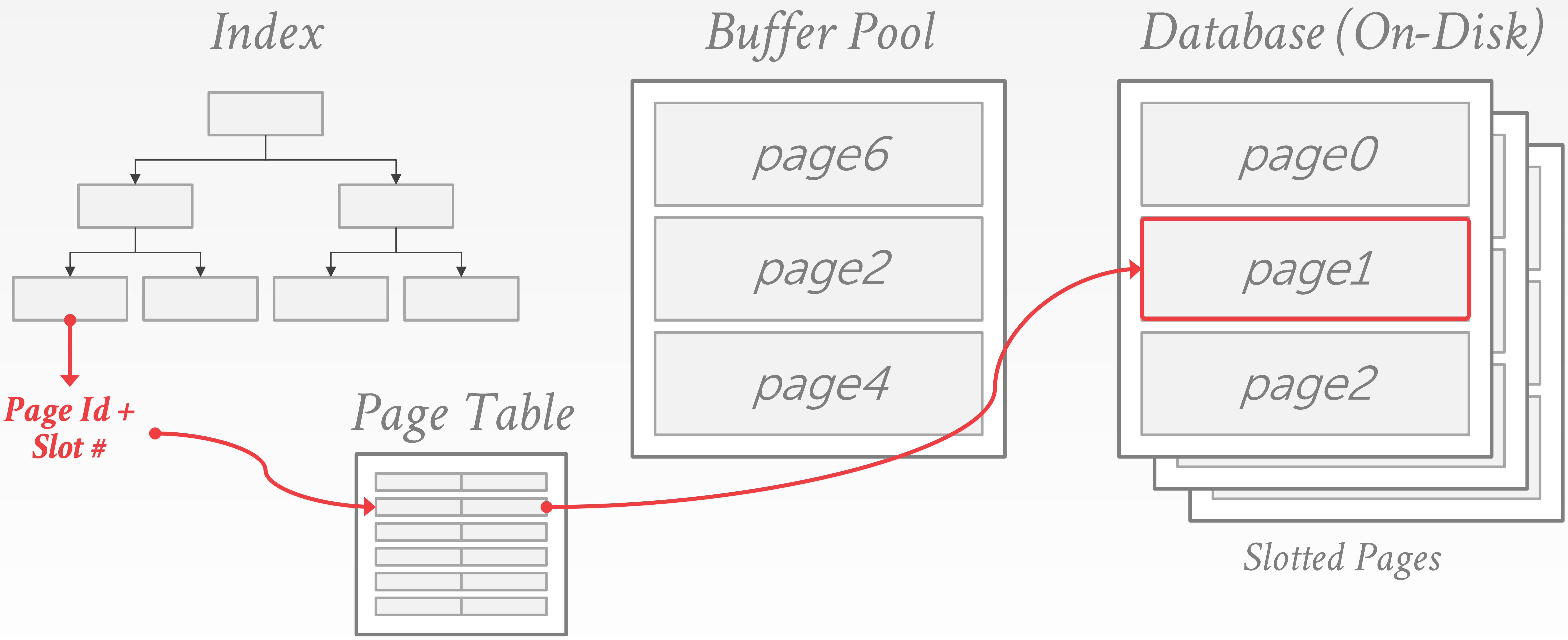
BUFFER MANAGEMENT



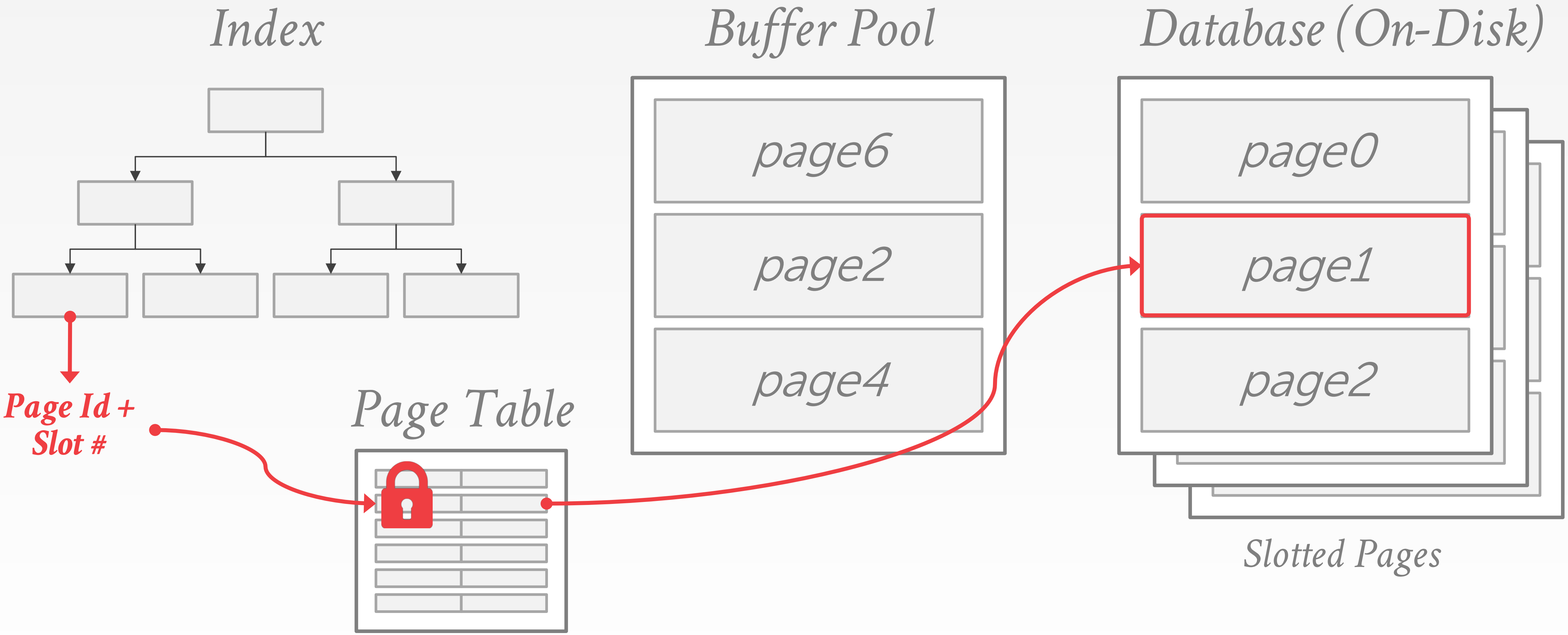
BUFFER MANAGEMENT



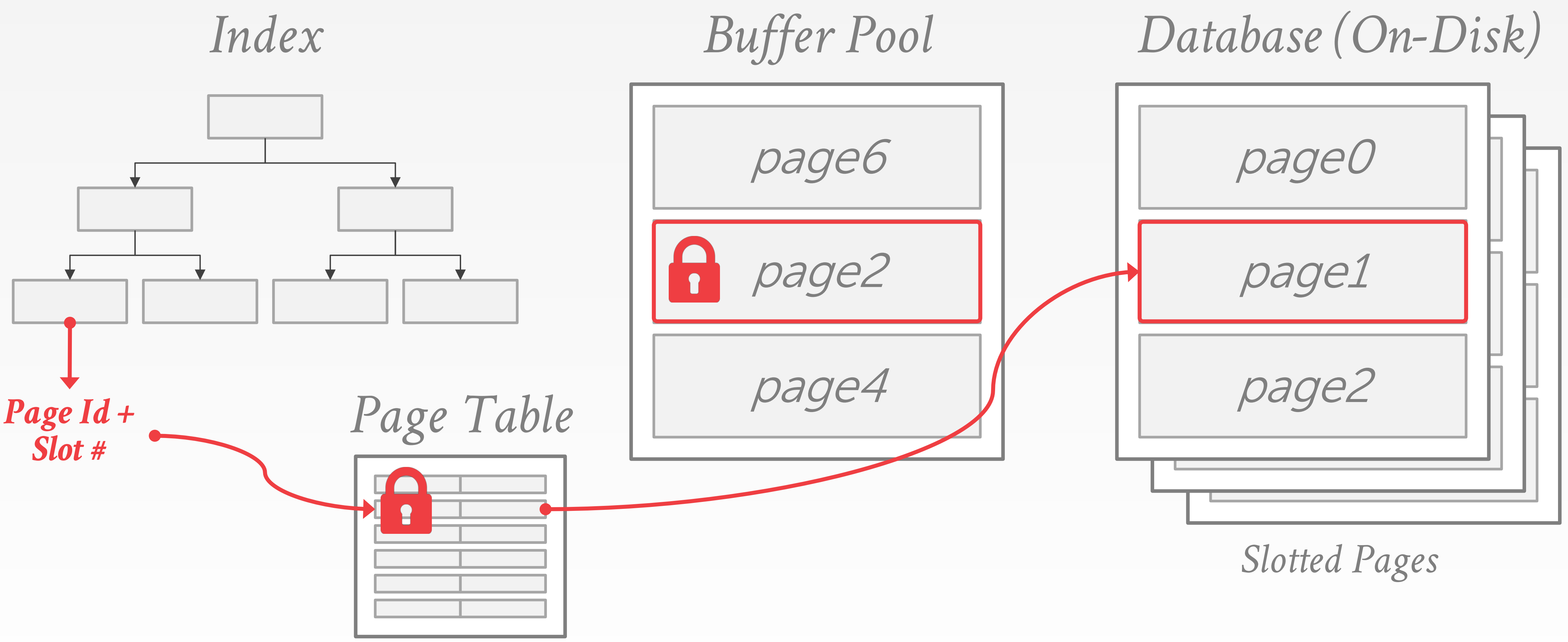
BUFFER MANAGEMENT



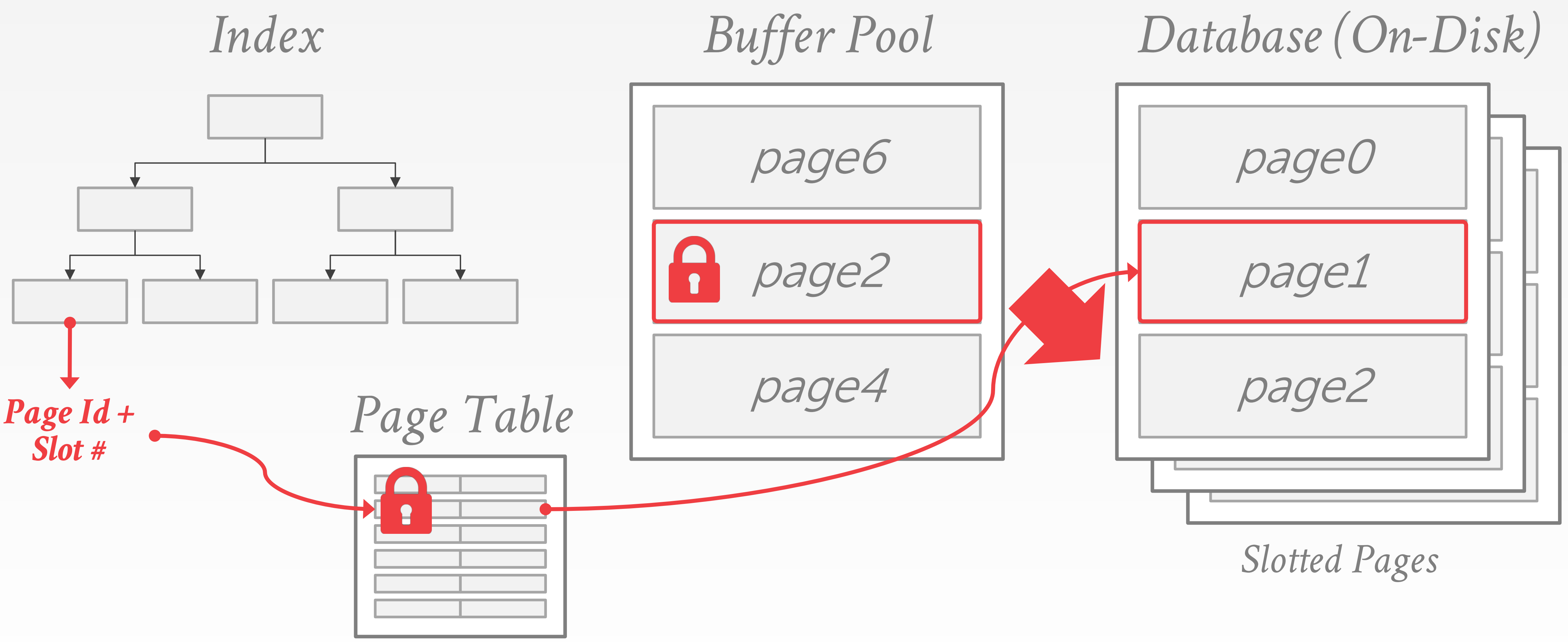
BUFFER MANAGEMENT



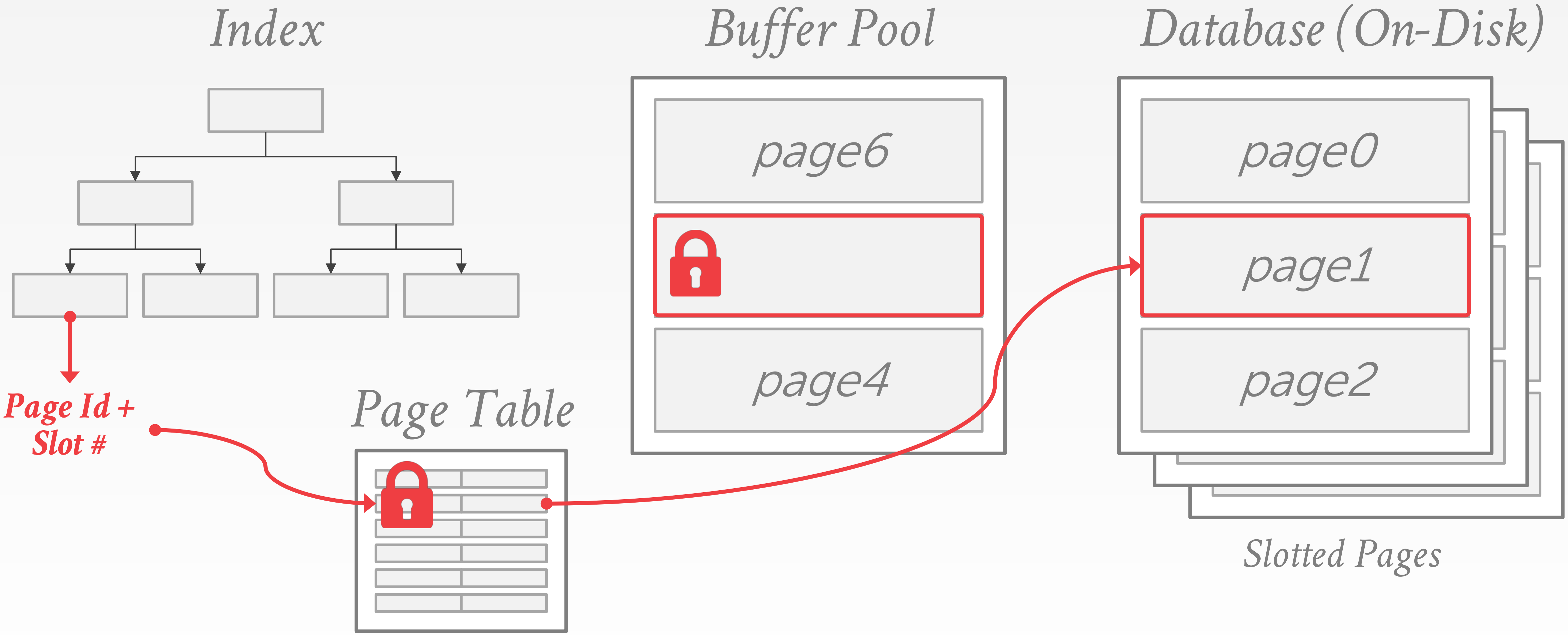
BUFFER MANAGEMENT



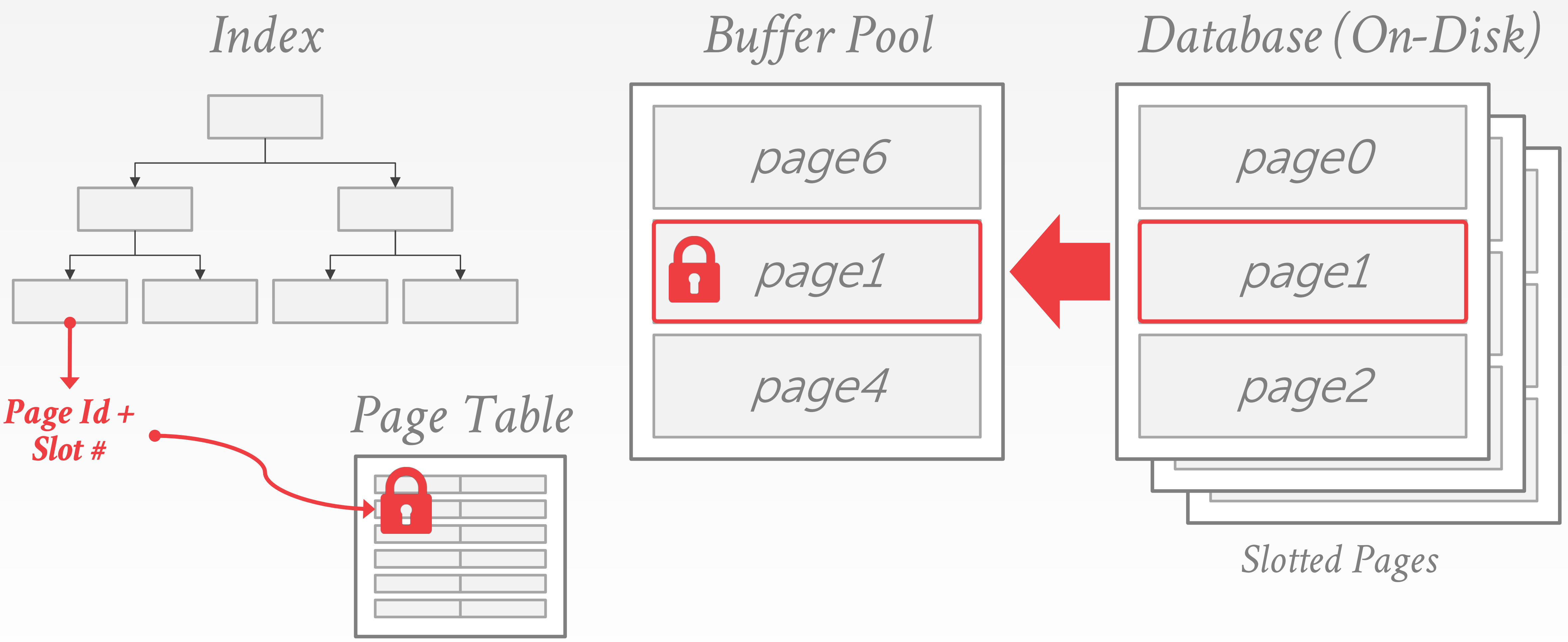
BUFFER MANAGEMENT



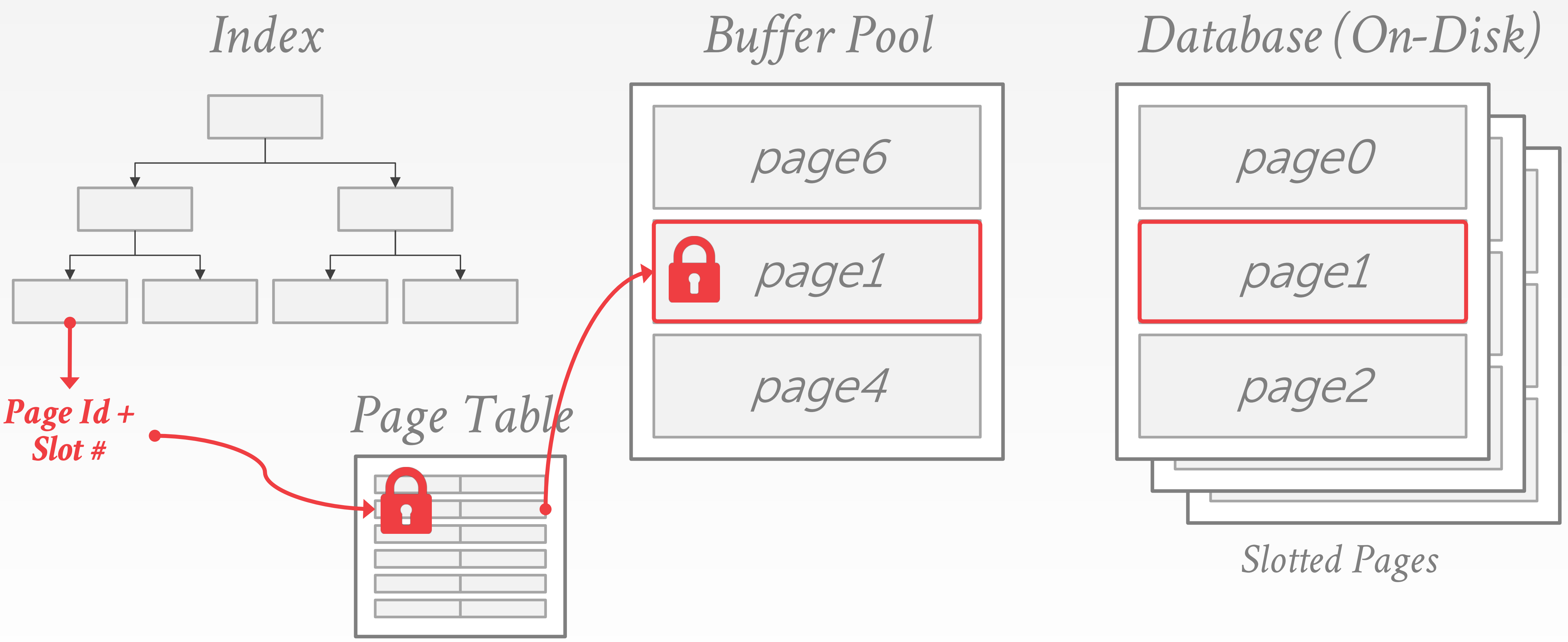
BUFFER MANAGEMENT



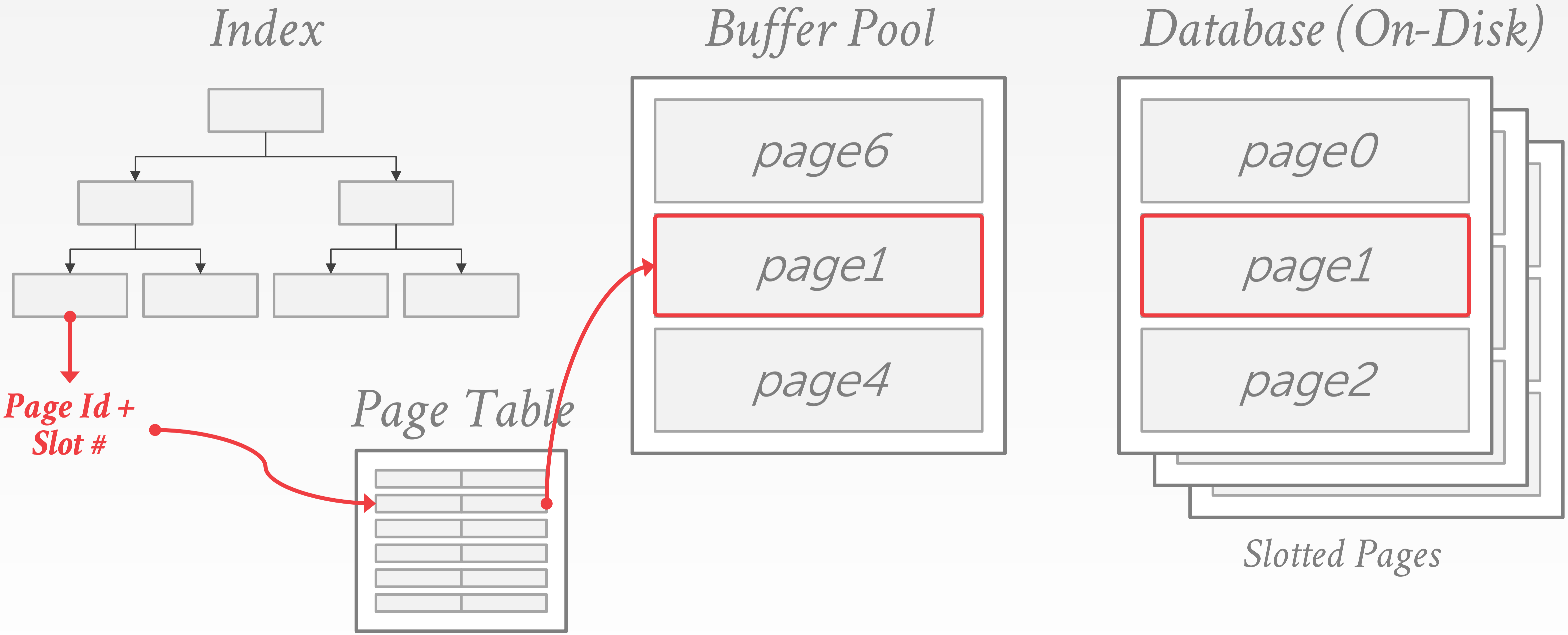
BUFFER MANAGEMENT



BUFFER MANAGEMENT



BUFFER MANAGEMENT



BUFFER MANAGEMENT

- Every tuple access has to go through the buffer pool manager regardless of whether that data will always be in memory.
  - Always have to translate a tuple's record id to its memory location.
  - Worker thread has to **pin** pages that it needs to make sure that they are not swapped to disk.

BUFFER MANAGEMENT

BUFFER MANAGEMENT

- **Q:** What do we gain by managing an in-memory buffer?
  - **A:** Accelerate query processing by storing frequently-accessed pages in fast memory
- **Q:** Can we “learn” an optimal page replacement policy?
  - **A:** Recent paper from Google on learning memory accesses based on LSTM models.

BUFFER MANAGEMENT

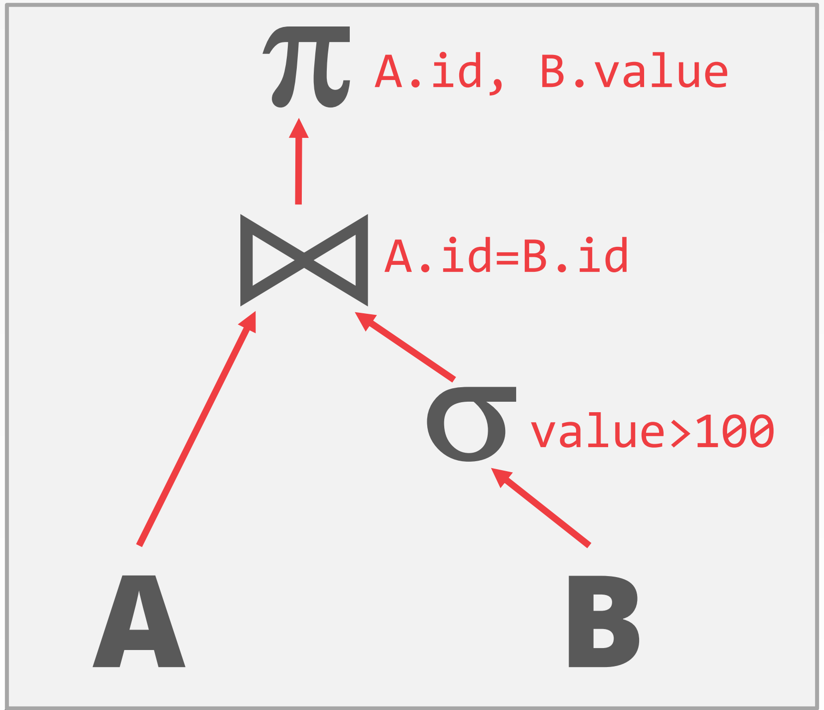
- **Q:** What do we gain by managing an in-memory buffer?
  - **A:** Accelerate query processing by storing frequently-accessed pages in fast memory
- **Q:** Can we “learn” an optimal page replacement policy?
  - **A:** Recent paper from Google on learning memory accesses based on LSTM models.

BUFFER MANAGEMENT

- **Q:** What do we gain by managing an in-memory buffer?
  – **A:** Accelerate query processing by storing frequently-accessed pages in fast memory
- **Q:** Can we “learn” an optimal page replacement policy?
  – **A:** Recent paper from Google on learning memory accesses based on LSTM models.

QUERY PROCESSING

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



Tuple-at-a-time

→ Each operator calls **next** on their child to get the next tuple to process.

Operator-at-a-time

→ Each operator materializes their entire output for their parent operator.

Vector-at-a-time

→ Each operator calls **next** on their child to get the next chunk of data to process.

QUERY PROCESSING

- The best strategy for executing a query plan in a disk-centric DBMS
  - Sequential scans over a table are much faster than random accesses
- The traditional **tuple-at-a-time** iterator model works well
  - Because output of an operator will not fit in limited memory

CONCURRENCY CONTROL

- In a disk-oriented DBMS, the systems assumes that a txn could stall at any time when it tries to access data that is not in memory.

CONCURRENCY CONTROL

- Execute other txns at the same time so that if one txn stalls then others can keep running.
  - This is not because the DBMS is trying to use all cores in the CPU (still focusing on single-core CPUs)
  - We do this to let system make **forward progress** by executing another txn while the current txn is waiting for data to be fetched from disk

CONCURRENCY CONTROL

- Concurrency control policy
  - Responsible for deciding how to interleave operations of concurrent transactions in such a way that it appears as if they are running serially
  - This property is referred to as **serializability** of transactions

CONCURRENCY CONTROL

- Concurrency control policy
  - DBMS has to set locks and latches to ensure the highest level of **isolation** (ACID) between transactions
  - Locks are stored in a separate data structure (**lock table**) to avoid being swapped to disk.

LOGGING & RECOVERY

- This protocol helps ensure the atomicity and durability properties (ACID)
  - Durability: Changes made by **committed** transactions must be present in the database after recovering from a power failure.
  - Atomicity: Changes made by **uncommitted** (in-progress/aborted) transactions must **not** be present in the database after recovering from a power failure.

LOGGING & RECOVERY

- DBMSs use STEAL and NO-FORCE buffer pool management policies.
  - **STEAL:** DBMS can flush pages dirtied by uncommitted transactions to disk.
  - **NO-FORCE:** DBMS is not required to flush all pages dirtied by committed transactions to disk.
  - So all page modifications have to be flushed to the write-ahead log (**WAL**) before a txn can commit

LOGGING & RECOVERY

- Each log entry contains the **before** and **after images** of modified tuples.
  - STEAL: Modifications made by uncommitted transactions that are flushed to disk have to rolled back.
  - NO-FORCE: Modifications made by committed transactions might not have been flushed to disk.

LOGGING & RECOVERY

- Each log entry contains the **before** and **after images** of modified tuples.
  - Recording the before and after images in the log is critical to ensuring atomicity and durability
  - Lots of work to keep track of log sequence numbers (LSNs) all throughout the DBMS.

LOGGING & RECOVERY

LOGGING & RECOVERY

- **Q:** What would happen if we use a NO-STEAL policy?
  - **A:** Cannot support large transactions that make changes larger than the buffer pool
- **Q:** What would happen if we use a FORCE policy?
  - **A:** Performance would drop by orders of magnitude since need to randomly write to disk all the time.

LOGGING & RECOVERY

- **Q:** What would happen if we use a NO-STEAL policy?
  - **A:** Cannot support large transactions that make changes larger than the buffer pool
- **Q:** What would happen if we use a FORCE policy?
  - **A:** Performance would drop by orders of magnitude since need to randomly write to disk all the time.

LOGGING & RECOVERY

- **Q:** What would happen if we use a NO-STEAL policy?
  - **A:** Cannot support large transactions that make changes larger than the buffer pool
- **Q:** What would happen if we use a FORCE policy?
  - **A:** Performance would drop by orders of magnitude since need to randomly write to disk all the time.

TAKEAWAYS

- Disk-oriented DBMSs do a lot of extra stuff because they are predicated on the assumption that data has to reside on disk
- In-memory DBMSs maximize performance by optimizing these protocols and algorithms

IN-MEMORY DBMSs

IN-MEMORY DBMSS

- Assume that the primary storage location of the database is **permanently** in memory.
- Early ideas proposed in the 1980s but it is now feasible because DRAM prices are low and capacities are high.

BOTTLENECKS

- If I/O is no longer the slowest resource, much of the DBMS's architecture will have to change account for other bottlenecks:
  - Locking/latching
  - Cache misses
  - Predicate evaluations
  - Data movement & copying
  - Networking (between application & DBMS)

STORAGE ACCESS LATENCIES

	L3	DRAM	SSD	HDD
Read Latency	~20 ns	60 ns	25,000 ns	10,000,000 ns
Write Latency	~20 ns	60 ns	300,000 ns	10,000,000 ns

LET'S TALK ABOUT STORAGE & RECOVERY METHODS FOR NON-VOLATILE MEMORY DATABASE SYSTEMS
SIGMOD, pp. 707-722, 2015.

STORAGE ACCESS LATENCIES

Jim Gray's analogy:
→Reading from L3 cache: Reading a book on a table
→Reading from HDD: Flying to Pluto to read that book

Because everything fits in DRAM, we can do more sophisticated things in software.

BUFFER MANAGEMENT

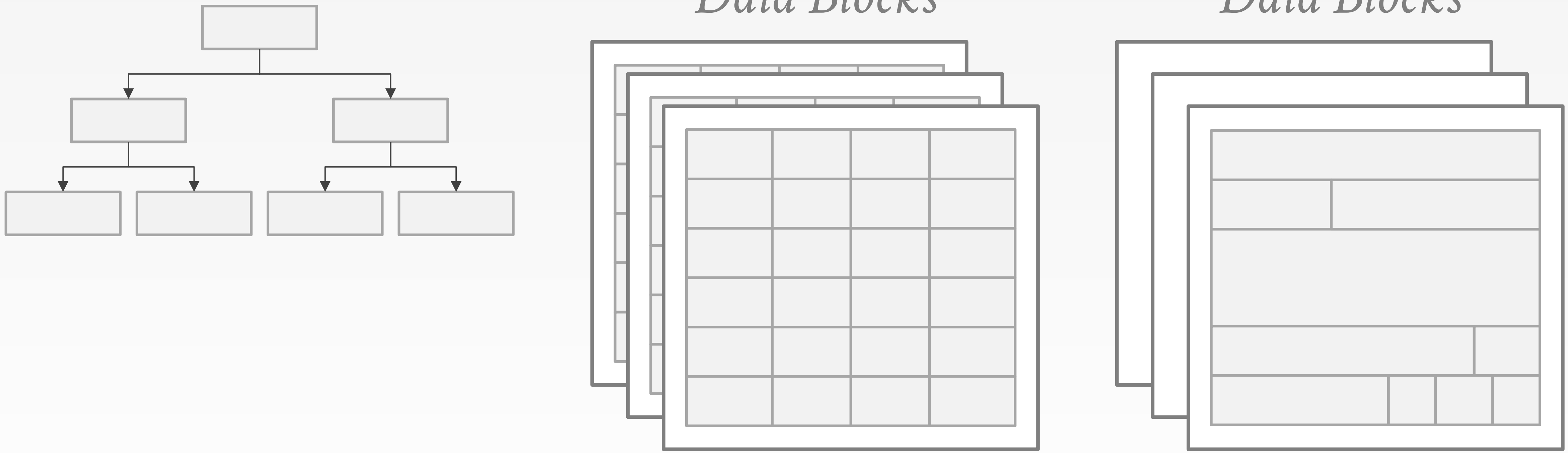
- An in-memory DBMS does not need to store the database in slotted pages but it will still organize tuples in blocks:
  - Direct memory pointers vs. tuple identifiers
  - Separate pools for fixed-length (e.g., numeric data) and variable-length data (e.g., images)
  - Use checksums to detect software errors from trashing the database.

BUFFER MANAGEMENT

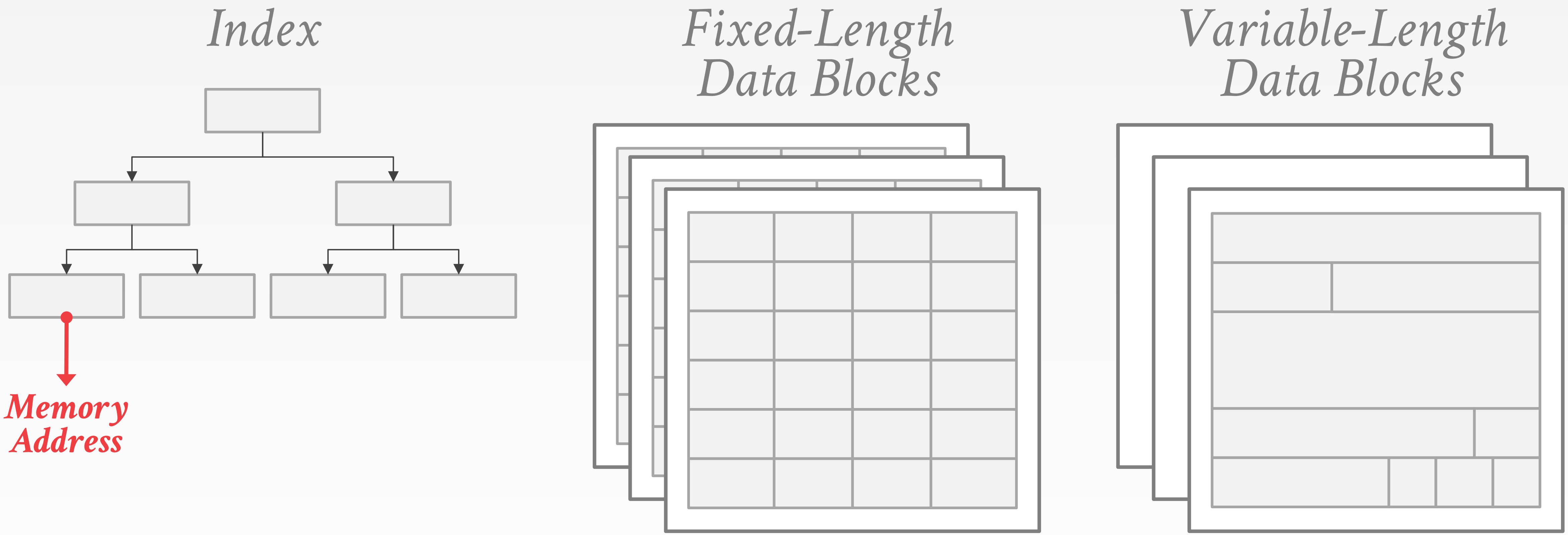
Index

Fixed-Length Data Blocks

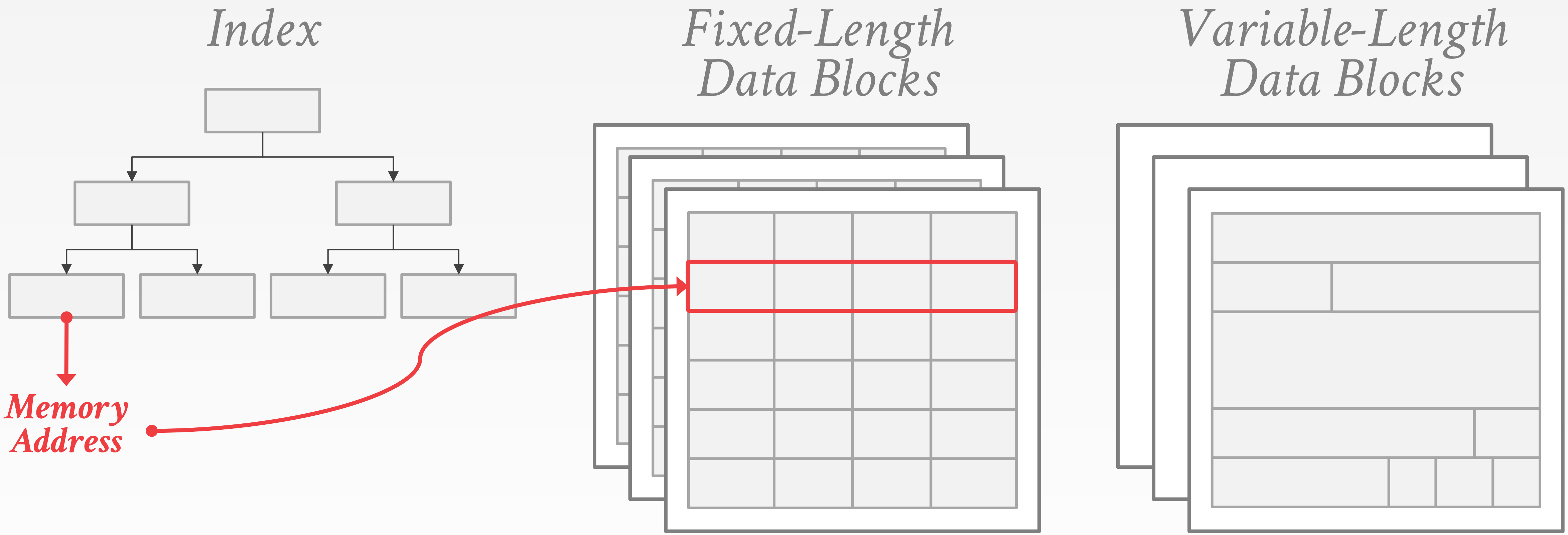
Variable-Length Data Blocks



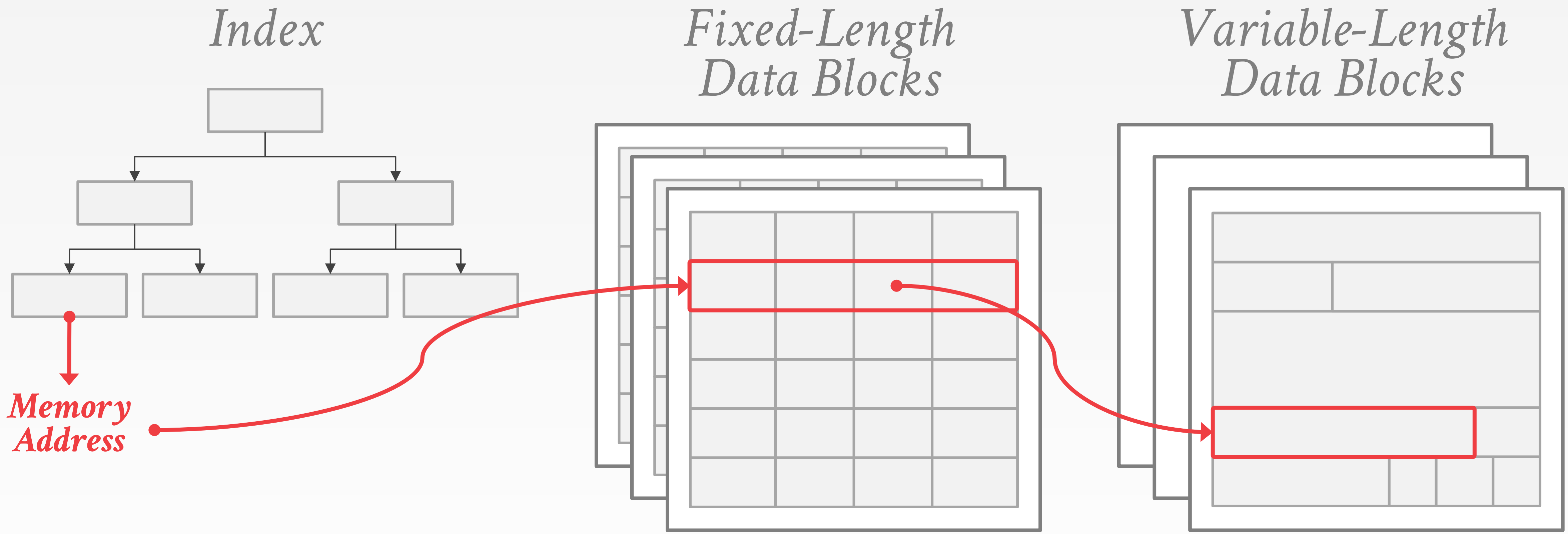
BUFFER MANAGEMENT



BUFFER MANAGEMENT



BUFFER MANAGEMENT

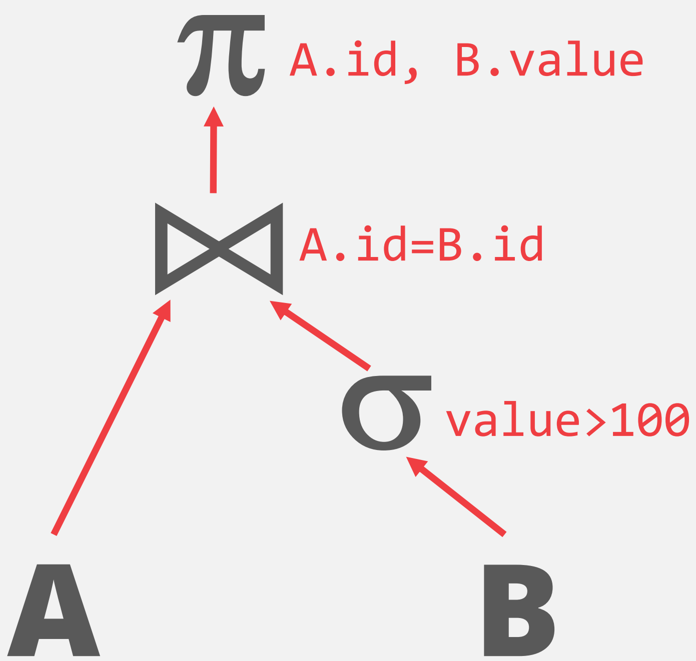


BUFFER MANAGEMENT

- DRAM is fast, but data is not accessed with the same frequency and in the same manner.
  - Hot Data: OLTP Operations (Tweets posted yesterday)
  - Cold Data: OLAP Queries (Tweets posted last year)
- We will study techniques for how to bring back disk-resident data without slowing down the entire system.

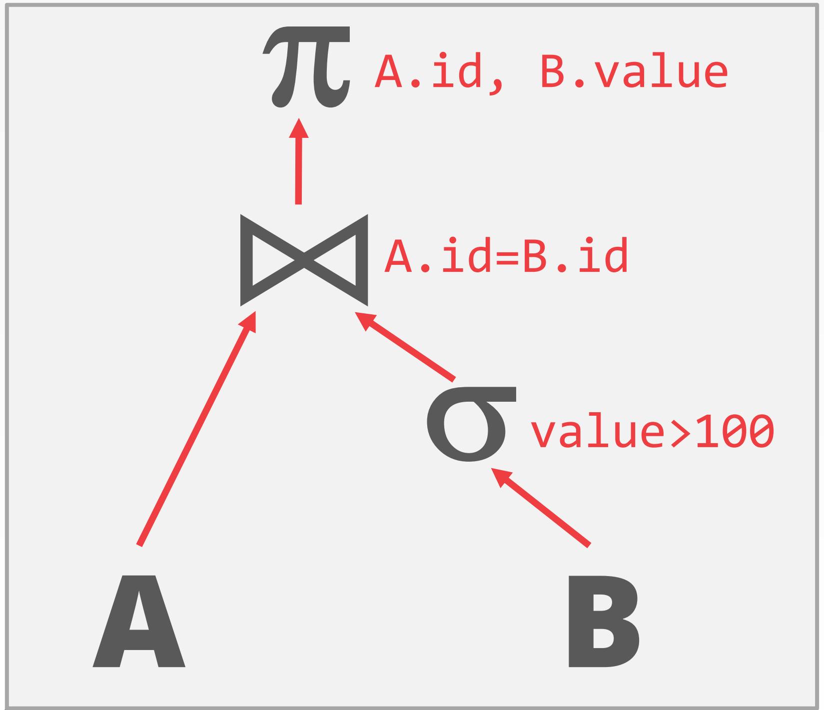
QUERY PROCESSING

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



QUERY PROCESSING

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



Tuple-at-a-time

→ Each operator calls **next** on their child to get the next tuple to process.

Operator-at-a-time

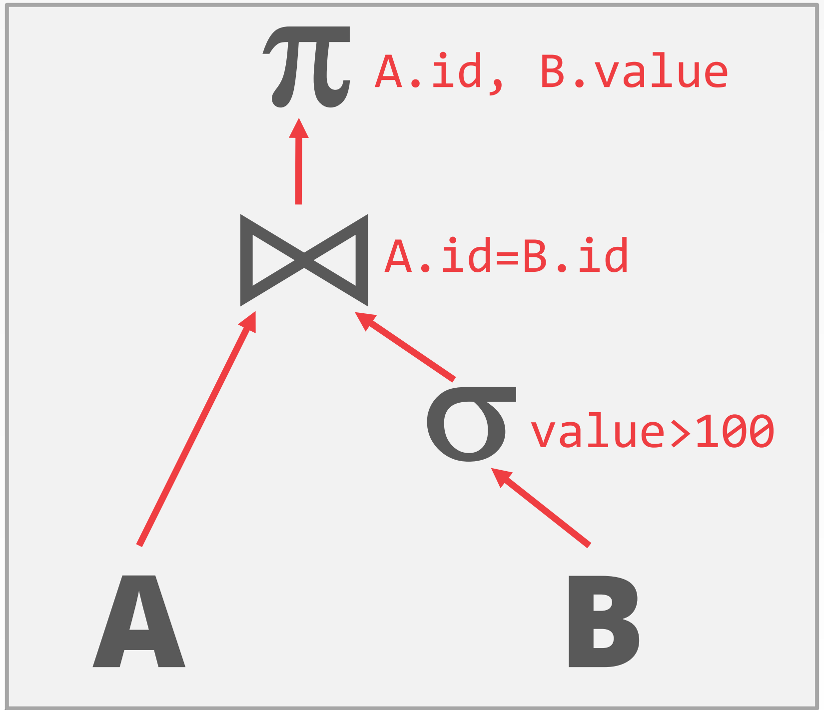
→ Each operator materializes their entire output for their parent operator.

Vector-at-a-time

→ Each operator calls **next** on their child to get the next chunk of data to process.

QUERY PROCESSING

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



Tuple-at-a-time

→ Each operator calls **next** on their child to get the next tuple to process.

Operator-at-a-time

→ Each operator materializes their entire output for their parent operator.

Vector-at-a-time

→ Each operator calls **next** on their child to get the next chunk of data to process.

QUERY PROCESSING

- The best strategy for executing a query plan in a DBMS changes when all of the data is already in memory.
  - Sequential scans are no longer significantly faster than random access.
- The traditional **tuple-at-a-time** iterator model is too slow because of function calls.
  - This problem is more significant in OLAP DBMSs.

QUERY PROCESSING

QUERY PROCESSING

- **Q:** Query processing in in-memory systems: sequential scans or random accesses?
  - **A:** Sequential scans are no longer significantly faster than random access.
- **Q:** Will the traditional tuple-at-a-time iterator work well now?
  - **A:** No, too slow because of function calls (virtual table lookups).

QUERY PROCESSING

- **Q:** Query processing in in-memory systems: sequential scans or random accesses?
  - **A:** Sequential scans are no longer significantly faster than random access.
- **Q:** Will the traditional tuple-at-a-time iterator work well now?
  - **A:** No, too slow because of function calls (virtual table lookups).

QUERY PROCESSING

- **Q:** Query processing in in-memory systems: sequential scans or random accesses?
  - **A:** Sequential scans are no longer significantly faster than random access.
- **Q:** Will the traditional tuple-at-a-time iterator work well now?
  - **A:** No, too slow because of function calls (virtual table lookups).

CONCURRENCY CONTROL

- Observation: The cost of a txn acquiring a lock is the same as accessing data (since the lock data is also in memory).
- In-memory DBMS may want to detect conflicts at a different granularity.
  - **Fine-grained locking** allows for better concurrency but requires more locks.
  - **Coarse-grained locking** requires fewer locks but limits the amount of concurrency.

CONCURRENCY CONTROL

- The DBMS can store locking information about each tuple together with its data.
  - This helps with CPU cache locality.
  - Mutexes are too slow. Need to use CAS instructions.

CONCURRENCY CONTROL

- Disk-oriented DBMSs
  – Stalling during disk I/O
- Memory-oriented DBMSs
  – New bottleneck is contention caused from txns executing on multiple cores trying to access data at the same time.

LOGGING & RECOVERY

- The DBMS still needs a WAL on disk since the system could halt at anytime.
  - Use **group commit** to batch log entries and flush them together to amortize `fsync` cost.
  - May be possible to use more lightweight logging schemes (e.g., only store redo information, NO-STEAL).
  - But since there are no "dirty" pages, there is no need to maintain LSNs all throughout the system.

LOGGING & RECOVERY

- The system also still takes checkpoints to speed up recovery time.
- Different methods for check-pointing:
  - Old idea: Maintain a second copy of the database in memory that is updated by replaying the WAL.
  - Switch to a special "copy-on-write" mode and then write a dump of the database to disk.
  - Fork DBMS process and then have the child process write its contents to disk (using virtual memory).

SUMMARY

- Disk-oriented DBMSs are a relic of the past.
  - Most structured databases fit entirely in DRAM on a single machine.
- The world has finally become comfortable with in-memory data storage and processing.

ANATOMY OF A DATABASE SYSTEM

Query

Connection Manager + Admission Control	*Process Manager*
Query Parser	*Query Processor*
Query Optimizer	
Query Executor	
Lock Manager (Concurrency Control)	*Transactional Storage Manager*
Access Methods (or Indexes)	
Buffer Pool Manager	
Log Manager	
Memory Manager + Disk Manager	*Shared Utilities*
Networking Manager	

Source: Anatomy of a Database System

NEXT LECTURE

- Data Storage
- Assigned Reading
  - **BlazeIt: Fast Exploratory Video Queries using Neural Networks**