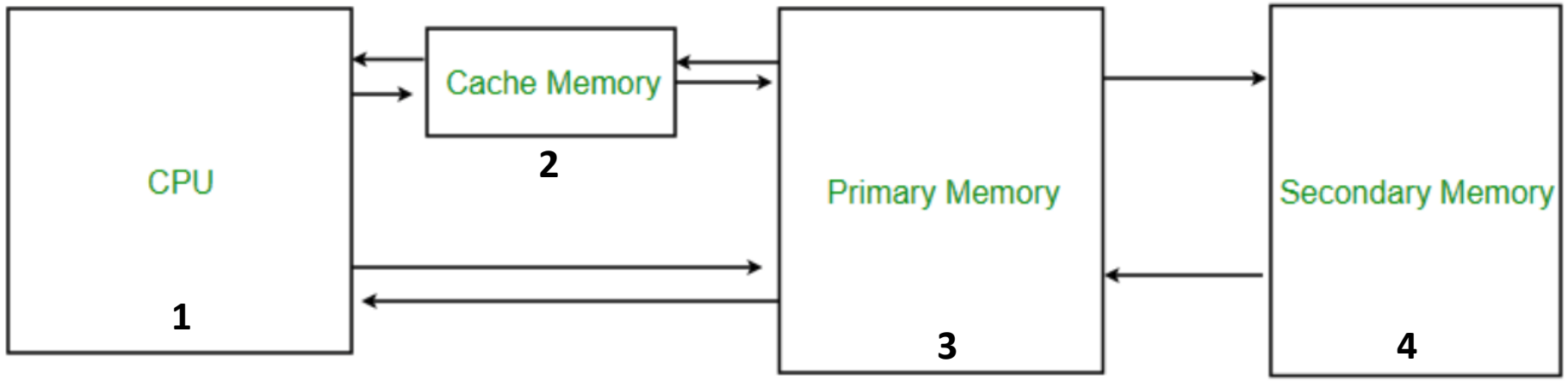
RDBMS Transactions
Locks & Concurrency

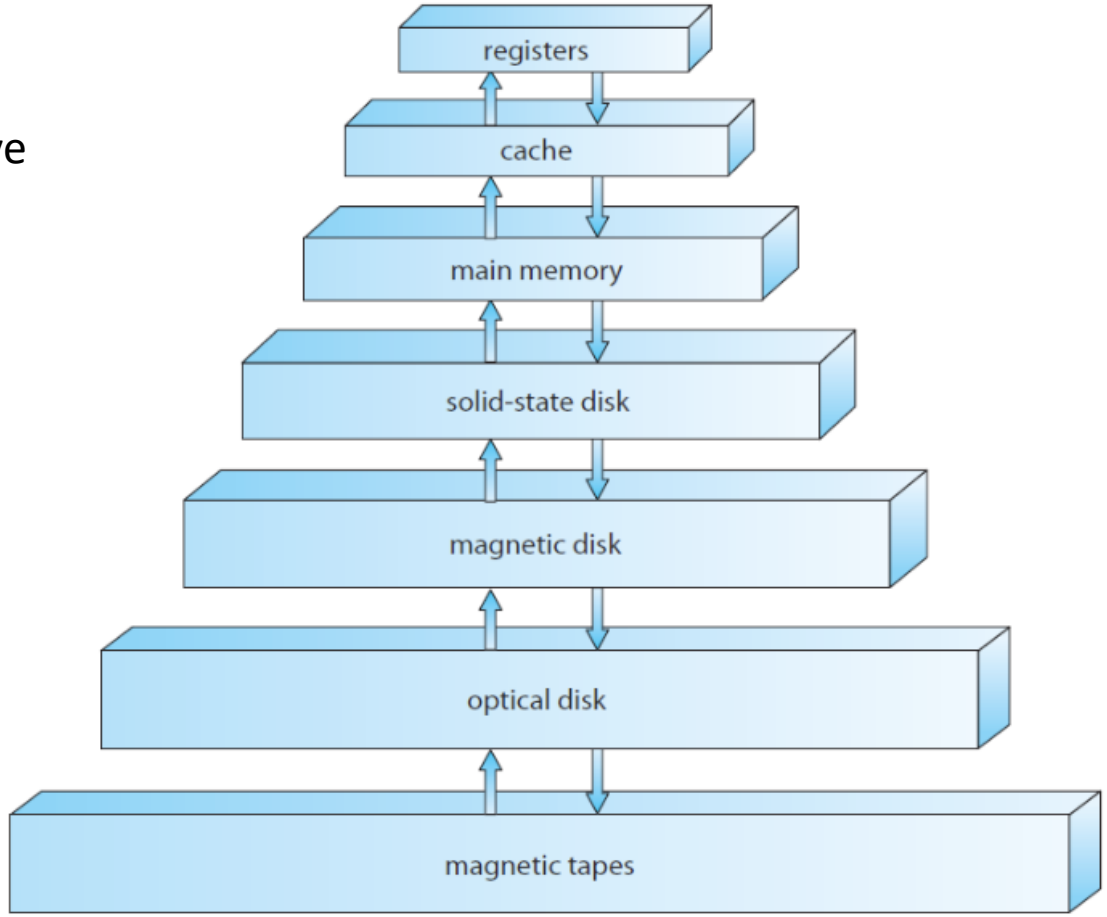
Memory in Computer Architecture



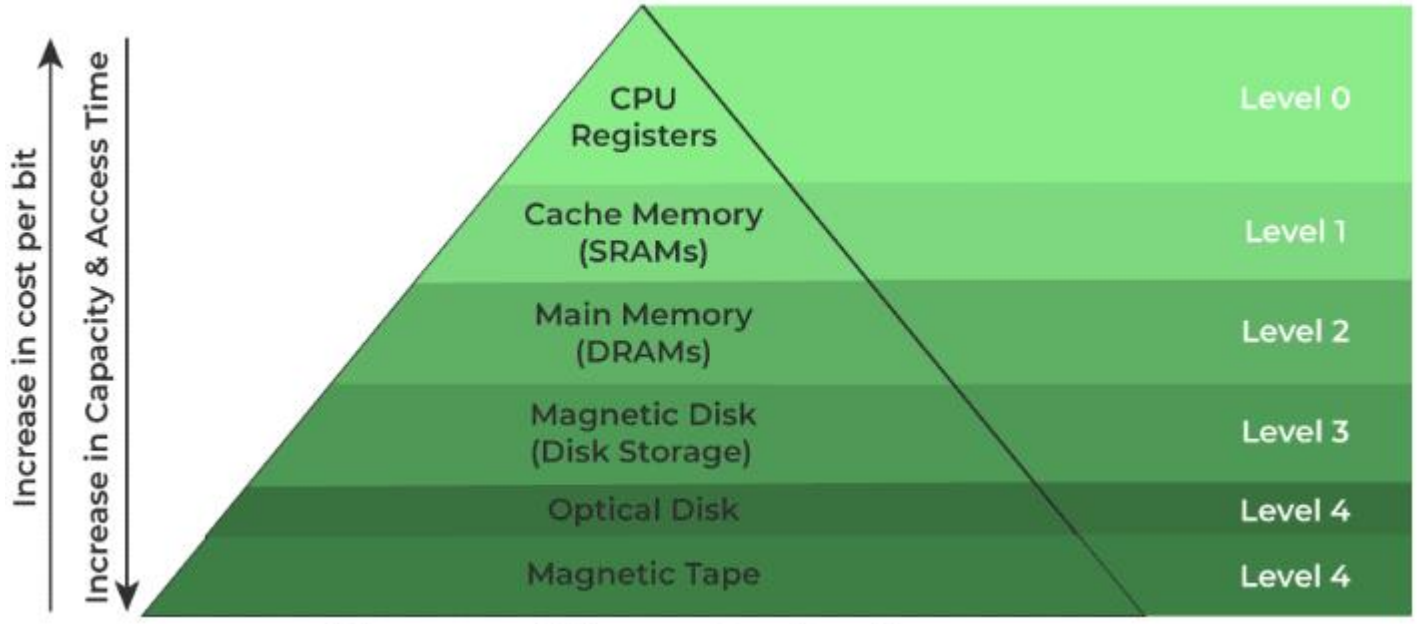
Level 1: **Register** (accumulator, counters, address register, etc.)
Level 2 (**Cache Memory**): fastest memory (faster access time). Data is temporarily stored there
Level 3: (**Main Memory):** where the computer works currently). It is volatile
Level 4 (**Secondary Memory**): external memory not as fats as main mem, but data stays permanently there

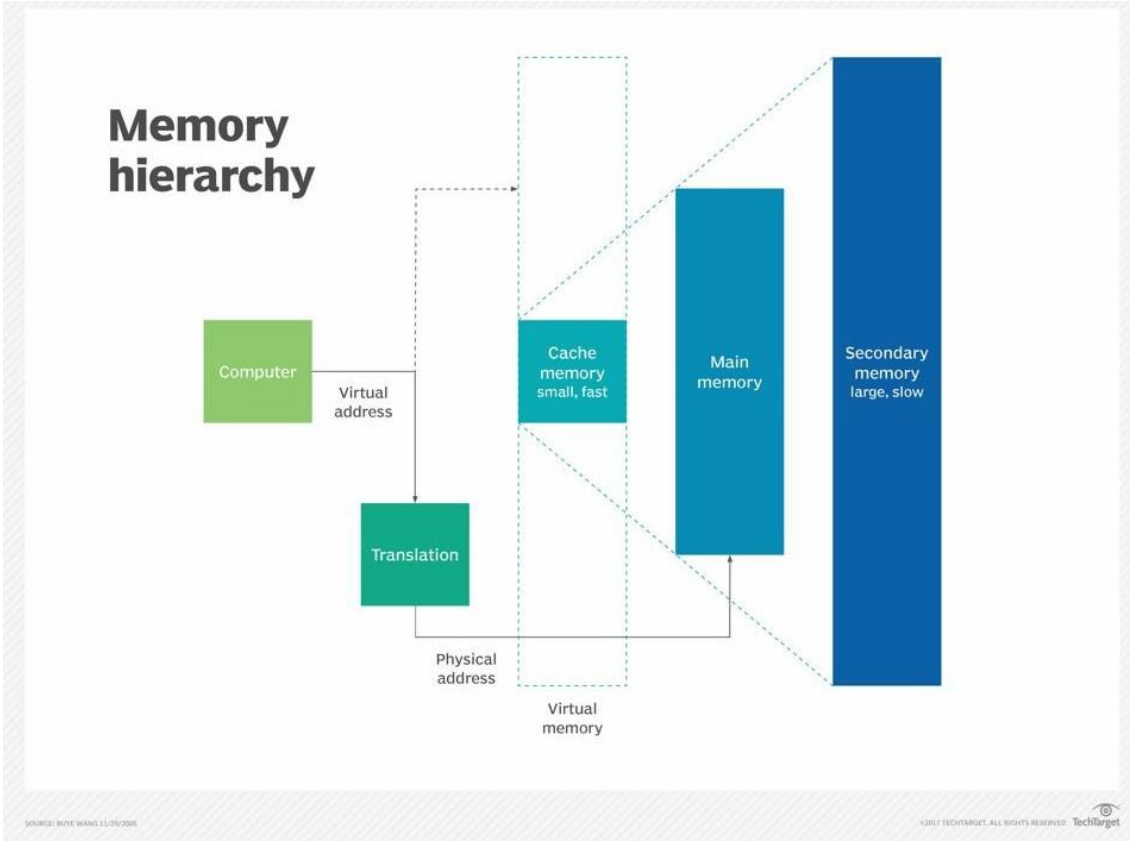
Memory Hierarchy

Faster, expensive

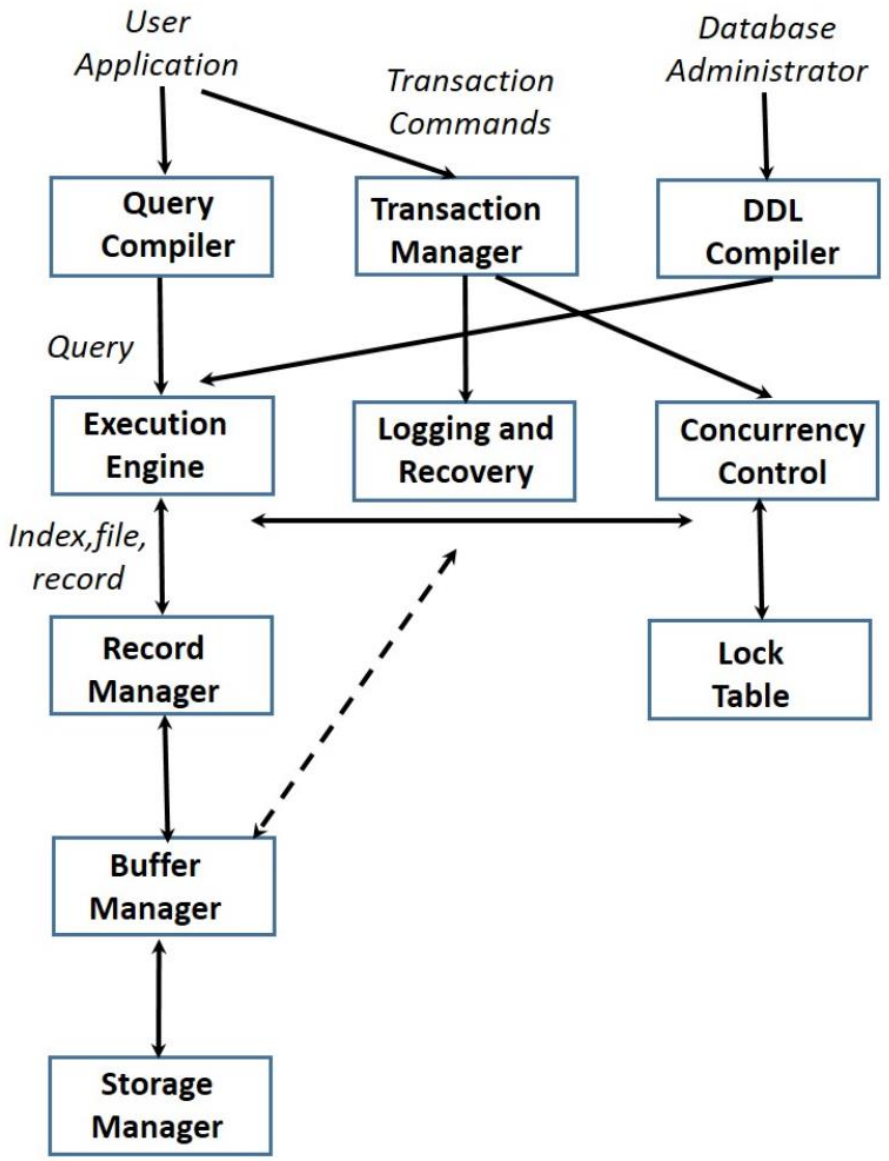


More on Memory Hierarchy





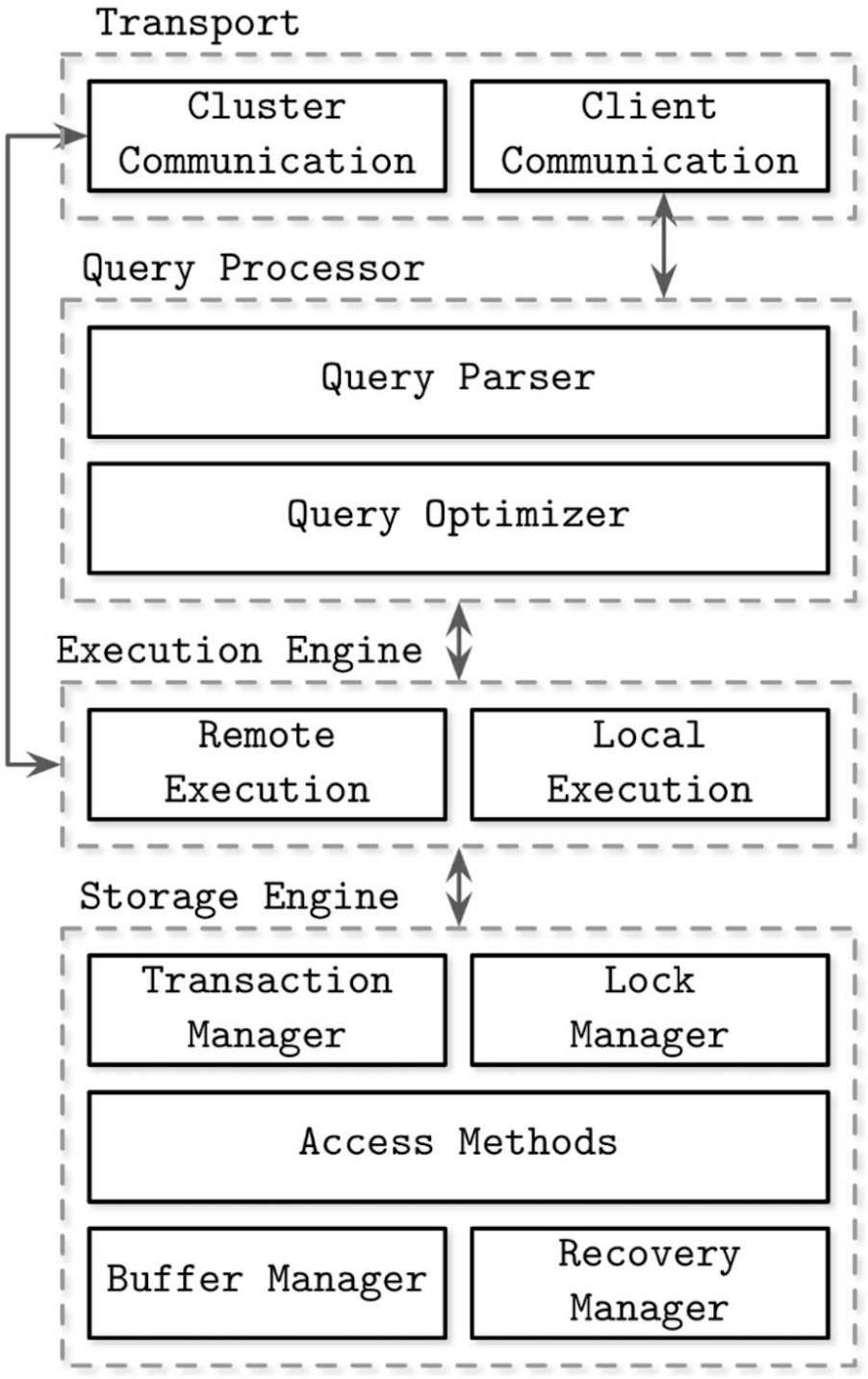
Architecture of a DBMS*



*Source: Mejia Alvarez et al. Main Memory Management on Relational Database Systems, Springer Nature, 2022

Architecture of a DBMS*

- Transport
- Query Processor
- Execution Engine
- Storage Engine



*Source: Alex Petrov. Database Internals. O'Reilly, 2019.

Transaction Processing

- Logging
- Concurrency Control
- Deadlock Resolution

Transaction Processing – Logging*

Logging

- In order to assure durability, every change in the database is logged separately on disk
- The log manager initially writes the log in buffers and negotiates with the buffer manager to make sure that buffers are written to disk eventually

*Source: Mejia Alvarez et al. Main Memory Management on Relational Database Systems, Springer Nature, 2022

Transaction Processing – Concurrency control*

Concurrency Control

- Transactions must appear to execute in isolation.
  - In most systems, there will be many transactions executing, but the individual actions of multiple transactions are executed in such an order that the net effect is the same as if the transactions had in fact executed in their entirety, one-at-a-time.
- A typical scheduler does its work by maintaining locks on certain pieces of the database
  - These locks prevent two transactions from accessing the same piece of data in ways that interact badly.
  - Locks are generally stored in a main-memory lock table –and eventually made persistent
  - The scheduler affects the execution of queries and other database operations by forbidding the execution engine from accessing locked parts of the database.

*Source: Mejia Alvarez et al. Main Memory Management on Relational Database Systems, Springer Nature, 2022

Transaction Processing – Deadlock Resolution*

Deadlock resolution

- As transactions compete for resources through the locks that the scheduler grants, they can get into a situation where none can proceed because each needs something another transaction has.
- The transaction manager has the responsibility to intervene and cancel (rollback or abort) one or more transactions to let the others proceed.

*Source: Mejia Alvarez et al. Main Memory Management on Relational Database Systems, Springer Nature, 2022

Two-phase Locking (2PL)

In databases and transaction processing, two-phase locking (2PL) is a concurrency control method that guarantees serializability

- The protocol uses locks, applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life.

When using the 2PL protocol, locks are applied and removed in two phases:

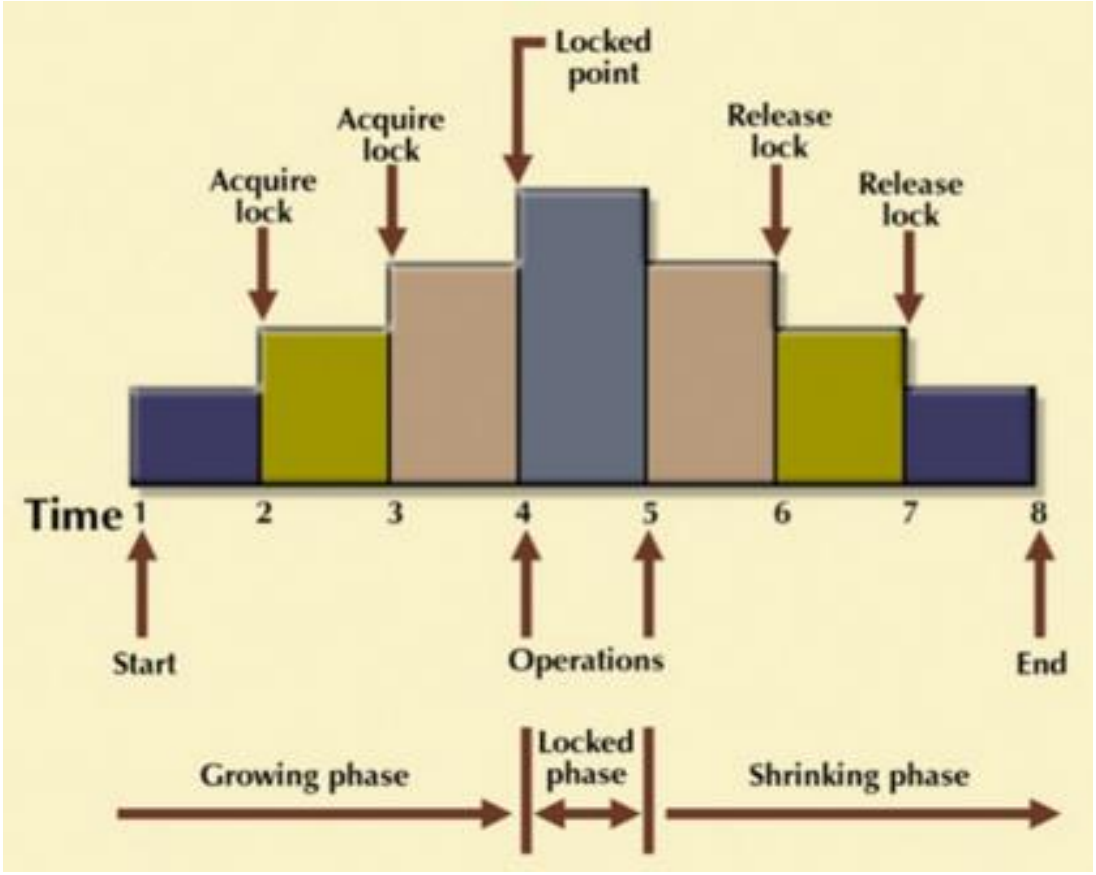
- Growing phase: locks are acquired and no locks are released.
- Shrinking phase: locks are released and no locks are acquired.

Two types of locks are used by the basic protocol

- Shared and Exclusive locks. Refinements of the basic protocol may use more lock types. Using locks that block processes,

2PL may be subject to deadlocks that result from the mutual blocking of two or more transactions.

DBMS Concurrency Control: Two-phase locking



OLTP*

OLTP (Online transactional processing):

- enables the rapid, accurate data processing behind ATMs and online banking, cash registers and ecommerce, and scores of other services we interact with each day.

- enables the real-time execution of large numbers of database transactions by large numbers of people, typically over the internet.

*Source: IBM

OLTP*

Process a large number of relatively simple transactions

- Usually insertions, updates, and deletions to data, as well as simple data queries (for example, a balance check at an ATM).

Enable multi-user access to the same data, while ensuring data integrity

- OLTP systems rely on concurrency algorithms to ensure that no two users can change the same data at the same time and that all transactions are carried out in the proper order.
- This prevents people from using online reservation systems from double-booking the same room and protects holders of jointly held bank accounts from accidental overdrafts.

Emphasize very rapid processing, with response times measured in milliseconds

- The effectiveness of an OLTP system is measured by the total number of transactions that can be carried out per second.

Provide indexed data sets: These are used for rapid searching, retrieval, and querying.

Are available 24/7/365

- OLTP systems process huge numbers of concurrent transactions, so any data loss or downtime can have significant and costly repercussions. A complete data backup must be available for any moment in time. OLTP systems require frequent regular backups and constant incremental backups.

*Source: IBM

OLTP vs. OLAP (online analytical processing)*

OLTP	OLAP
Use a relational database to accommodate large number of concurrent users and frequent queries and updates, while supporting very fast response times	OLAP systems use a multidimensional database—a special kind of database created from multiple relational databases that enables complex queries of involving multiple data facts from current and historical data. Can be organized as a Data Warehouse.
Queries are simple and typically involve just one or a few database records	Queries are complex queries involving large numbers of records.
Transaction and query response times are lightning-fast	Response times are orders of magnitude slower
Modify data frequently	Do not modify data at all
Workloads involve a balance of read and write	Workloads are read-intensive.
Databases require relatively little storage space	Databases work with enormous data sets and typically have significant storage space requirements.
Require frequent or concurrent backups;	Can be backed up far less frequently.

OLTP systems often serve as a source of information for OLAP systems

*Source: IBM

OLTP Examples*

- ATM machines (this is the classic, most often-cited example) and online banking applications
- Credit card payment processing (both online and in-store)
- Order entry (retail and back-office)
- Online bookings (ticketing, reservation systems, etc.)
- Record keeping (including health records, inventory control, production scheduling, claims processing, customer service ticketing, and many other applications)

*Source: IBM

OLTP System Design

Rollback segments

- Rollback segments are the portions of database that record the actions of transactions in the event that a transaction is rolled back.

Clusters

- A cluster is a schema that contains one or more tables that have one or more columns in common. It improves the performance of joins

Discrete transactions

- A discrete transaction defers all change to the data until the transaction is committed.

Block size

- The data block size should be a multiple of the operating system's block size within the maximum limit to avoid unnecessary I/O.

Buffer cache size

- SQL statements should be tuned to use the database buffer cache to avoid unnecessary resource consumption

Dynamic allocation of space to tables and rollback segments

Transaction processing monitors and the multi-threaded server

- A transaction processing monitor is used for coordination of services. It is like an operating system on its own

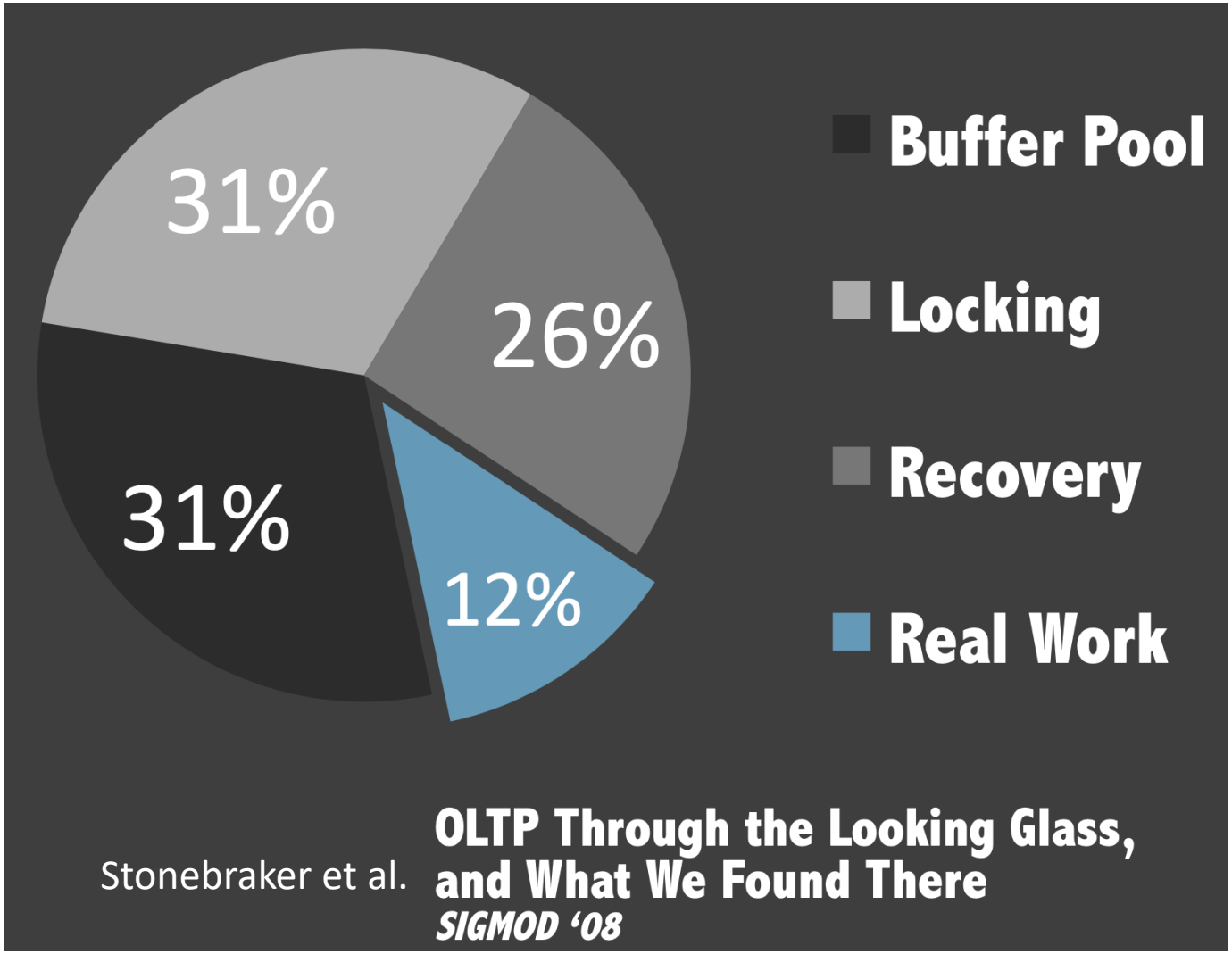
Partition (database)

- Partition use increases performance for sites that have regular transactions while still maintaining availability and security.

Database tuning

- With database tuning, an OLTP system can maximize its performance as efficiently and rapidly as possible.

OLTP Workload Split



Log System Structure (Journals & Journaling)

What do we need to know in order to be able to reverse an UPDATE transaction?

- Where it is happening
  - Table Space -> Table -> Relation/Row
- What is being affected
  - Attribute name/location
  - Value **BEFORE** update
  - Value **AFTER** update
- When the transaction happened
  - Time Stamp
- Status: completed or not (commited?)
- Unique Transaction ID (it could derivate from Time Stamp)
- Who made it
  - User ID of the person running the transaction

Journal/Log (possible) Structure

T_k

TransID	Table	Attribute	BEFORE Image	AFTER Image	Completed? Commited?	Time Stamp	UserID Owner Transaction

A Journal/Log is a collection of Transactions, with Structure T_k