数据库系统架构

徐辰
华东师范大学
数据科学与工程学院
cxu@dase.ecnu.edu.cn

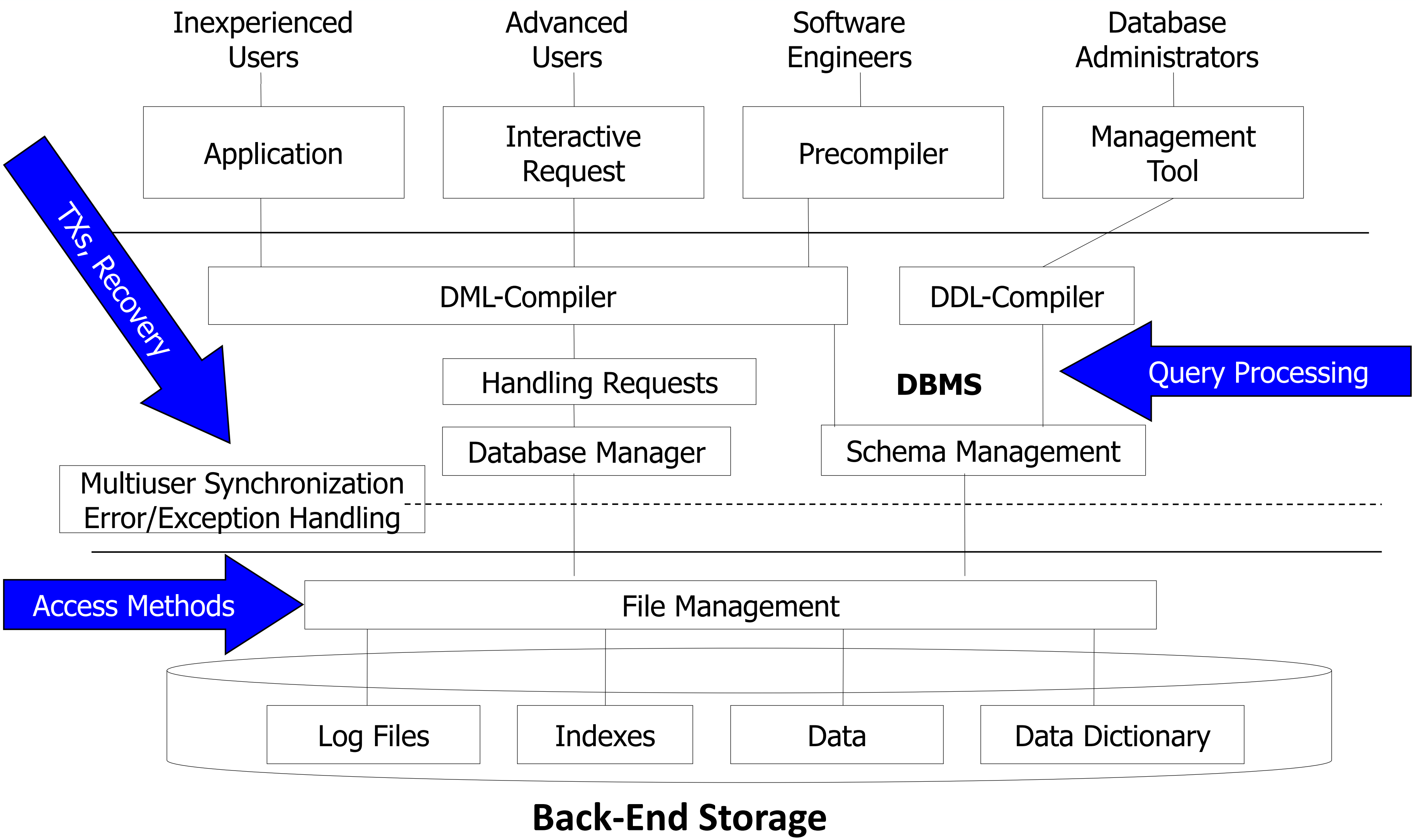
Overview and Architecture

Prof. Dr. Volker Markl
with slide material from
Felix Naumann
Prof. Dr. Ulf Leser
Prof. Dr. Johann Christoph Freytag
Prof. Dr. Kai-Uwe Sattler
Prof. Dr. Alfons Kemper, Dr. Eickler
Prof. Dr. Hector-Garcia Molina

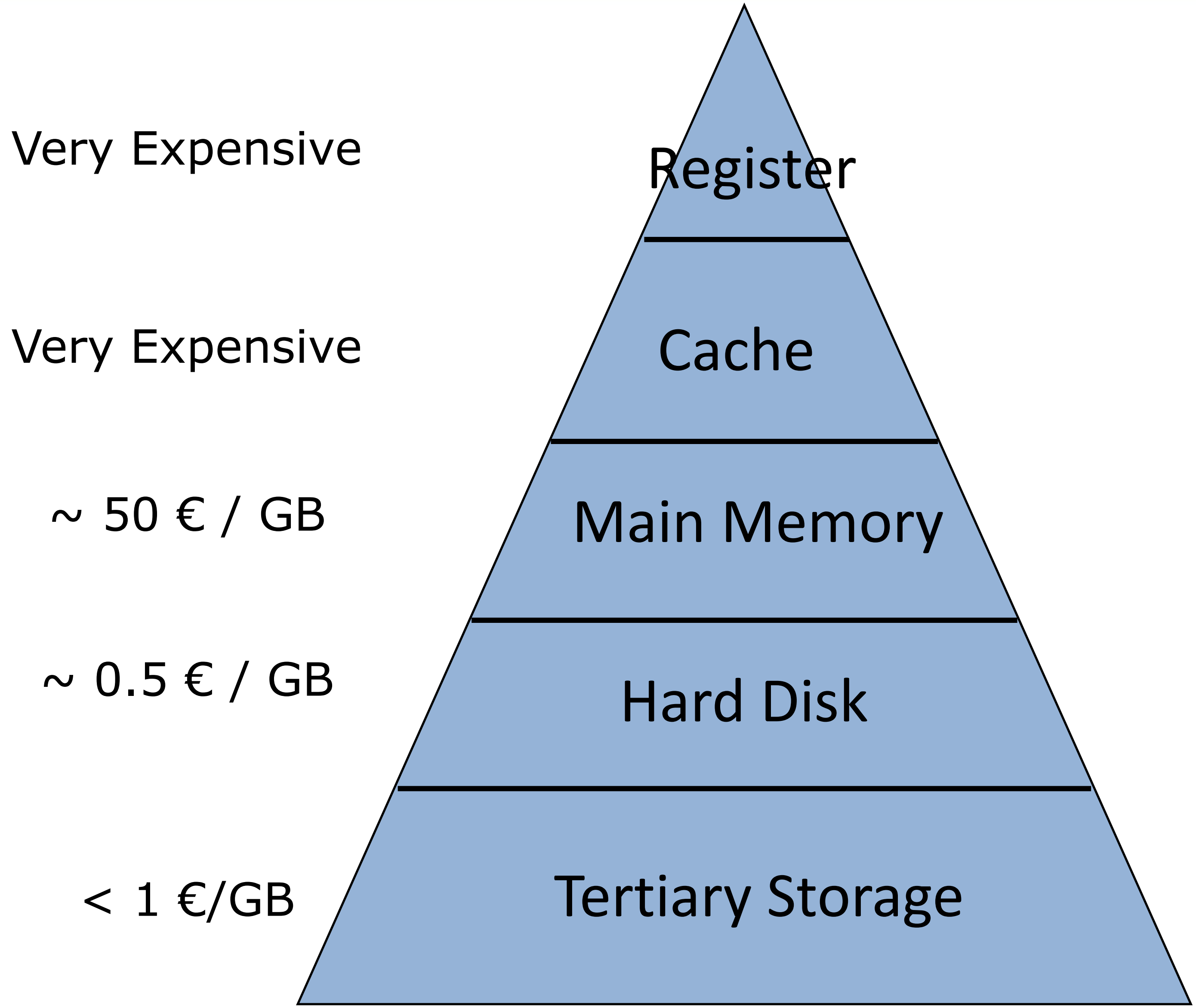
Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

http://www.dima.tu-berlin.de/

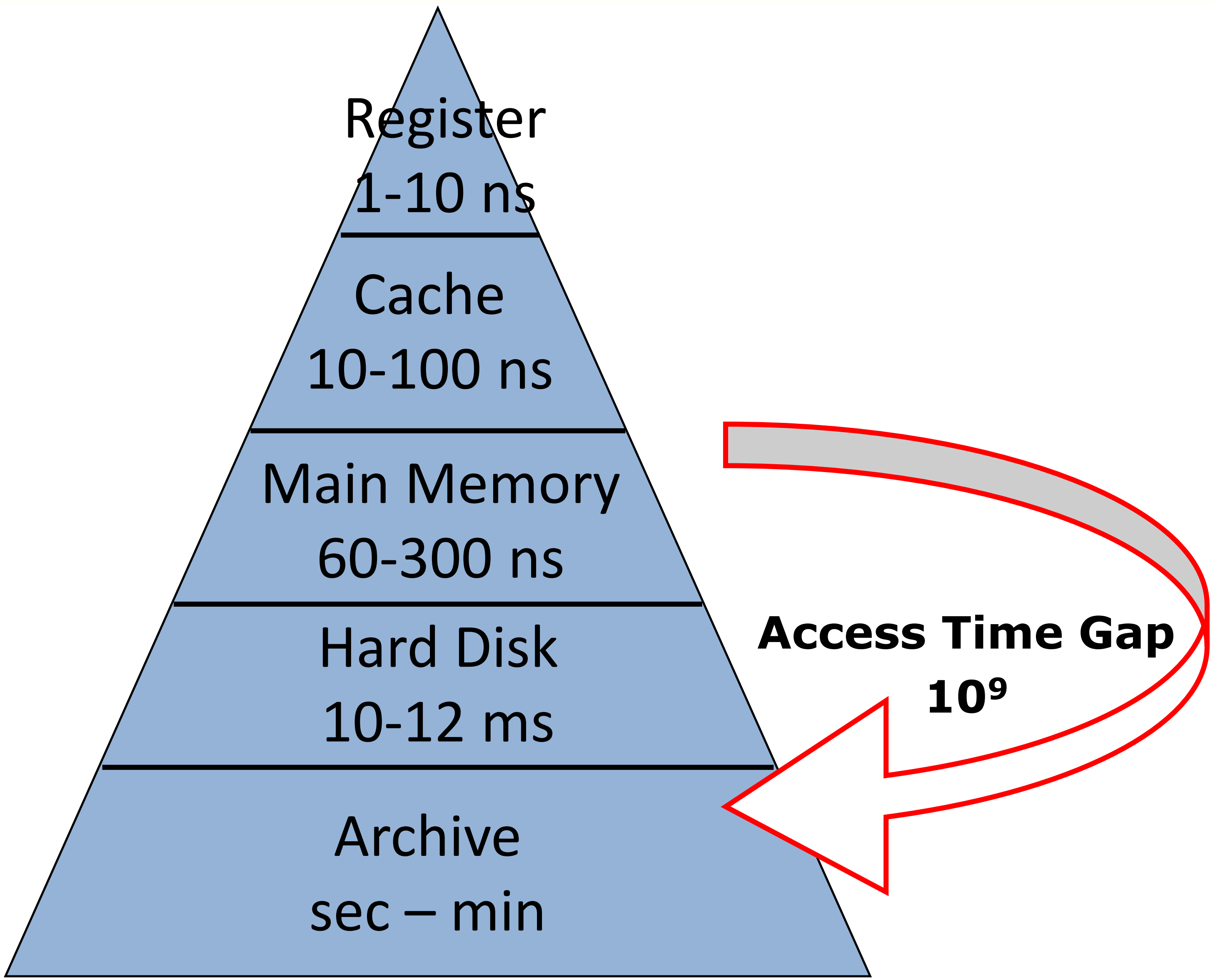
Architecture



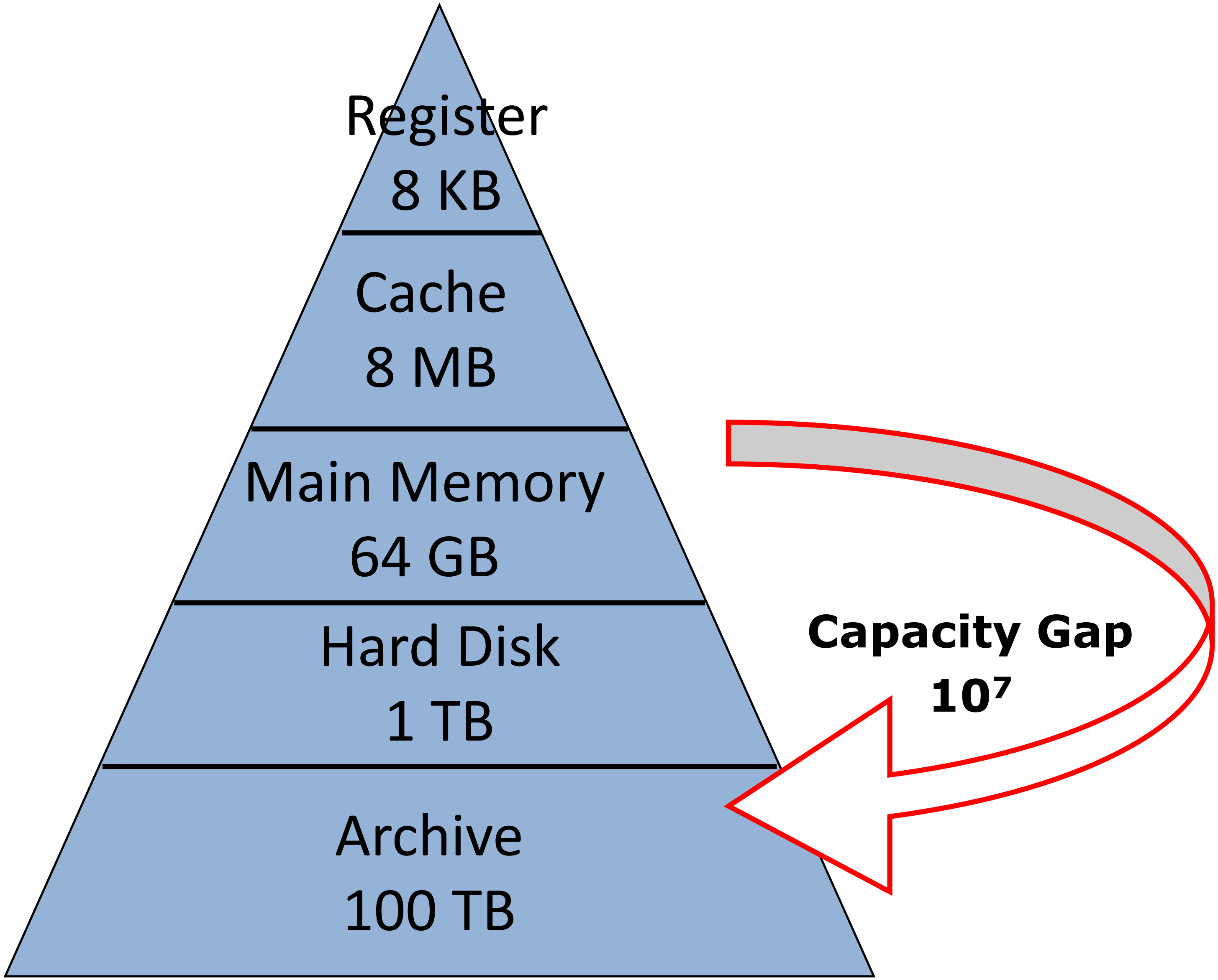
Overview: Memory Hierarchy



Overview: Memory Hierarchy



Overview: Memory Hierarchy



Three Levels of Data Representation

- Conceptual level
  - Relations, Tuple
  - Values of attributes
- Logical level
  - Files
  - Records
  - Fields
- Physical level
  - Drives
  - Blocks
  - Cylinders and Sectors

5 Layer Architecture

Bulk oriented access		
	Data model level	Query compilation, Access path selection, Access control, Integrity checks
Record oriented access		
	Logical access	Sorting, Transaction management, Cursors, Data Dictionary
Internal record interface		
	Memory structure	Physical Record Manager, Index Manager, Locking, Log / Recovery
Buffer interface		
	Buffer management	Page management, Buffer management, Caching
File interface		
	Operating system	Peripheral memory management
Device interface		

Objects and Operations

Interface	Level	Operations
Bulk oriented access		
	Data model level	SQL: select ... from ... Where Grant access to ... Create index on ...
Record oriented access		
	Logical access	OPEN – FETCH – CLOSE (Tab o. Index) STORE record
Internal record interface		
	Memory structure	STORE record on pages, UPDATE all access paths, Implementation B*-Tree
Buffer interface of operating system		
	Buffer management	READ page WRITE page
File interface		
	Operating system	Disc driver MOVE head ...
Device interface		

Interfaces

- Set-oriented interface
  - Access to sets of tuple by a declarative language
  - SELECT ... FROM ... WHERE ...
  - Monitoring of data integrity and authorization
- Record-oriented interface
  - Access to typed tuple
  - Access through logical access paths (Indexes, Scans)
  - Open/Next/Close Interface
  - Partition management
- Generic record interface
  - Access to uniform and un-typed tuple
  - Locking
  - Mapping tuples (logical objects) to pages

Interfaces

- Buffer interface
  - Uniform access to all blocks within the virtual address space
  - Mapping of virtual block addresses to physical block addresses
  - Synchronization of blocks (cache management, concurrent access) ("locking", but different to "transaction locks", often called "latching" or "pinning")

- File interface
  - Access to physical blocks
  - Managing the mapping between block and segment, tablespaces, files
  - Software-RAID

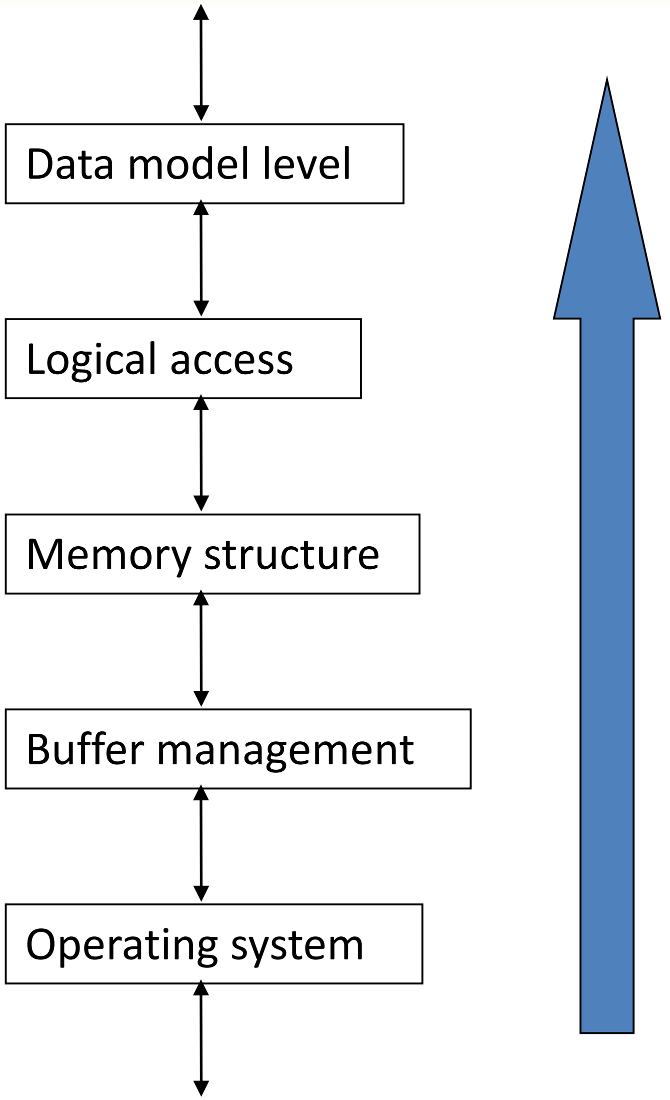
- Device interface
  - Access to hard drive data
  - Addressing discs – Disc, Track, Sector
  - Controller cache, Prefetching
  - Hardware RAID

5 Layer Architecture

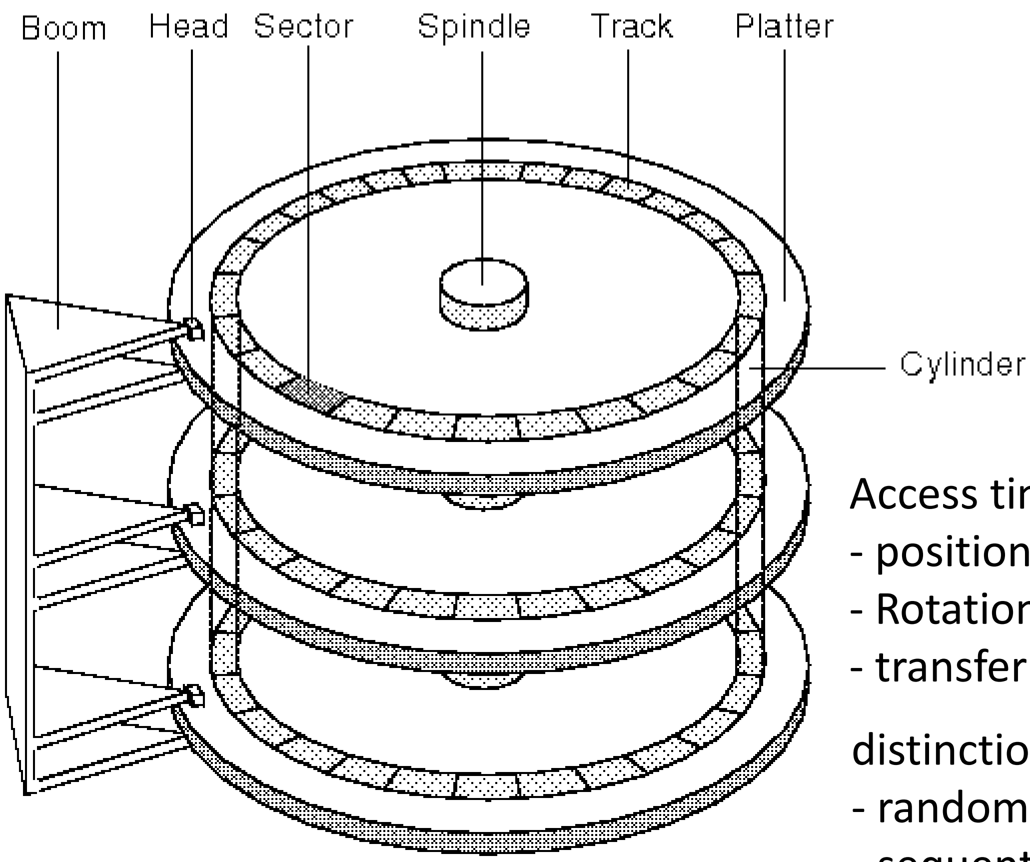
- **Idealized representation**
  - No need to strictly stick to that model
  - Some techniques cut through layers, e.g. synchronization, recovery, …
- Combination of layers is possible
  - E.g. „Record oriented and internal record interface"
- Often a direct access to another layer
  - Prefetching: Caching needs information about the actual workload; not only about the actual tuple
    - From layer logical record layer to buffer/OS layer
    - Perhaps from data model layer to buffer/OS layer
  - The optimizer needs information about physical allocation of blocks on disc
    - From OS layer to logical record/data model layer
  - Thus: In many DBMS implementations, the principle of **„Information Hiding"** is not 100% adhered to for performance reasons

Bottom Up

- Many topics cannot simply be associated with a single layer
  - Locking
  - Recovery
  - Request optimization
  - ...



Magnetic Hard Disk



Access time for a disk page:
- positioning time (track) (~4-8 ms)
- Rotational delay (sector) (~8 ms for 7200rpm)
- transfer time (sector)   (> 1 GB/s)

distinction:
- random I/O
- sequential I/O

disk page# = f(cylinder#, platter#, track# , sector#)
usual size: (2, 4 ,8, 16, 32, 64, 128 kB)

A) SSD

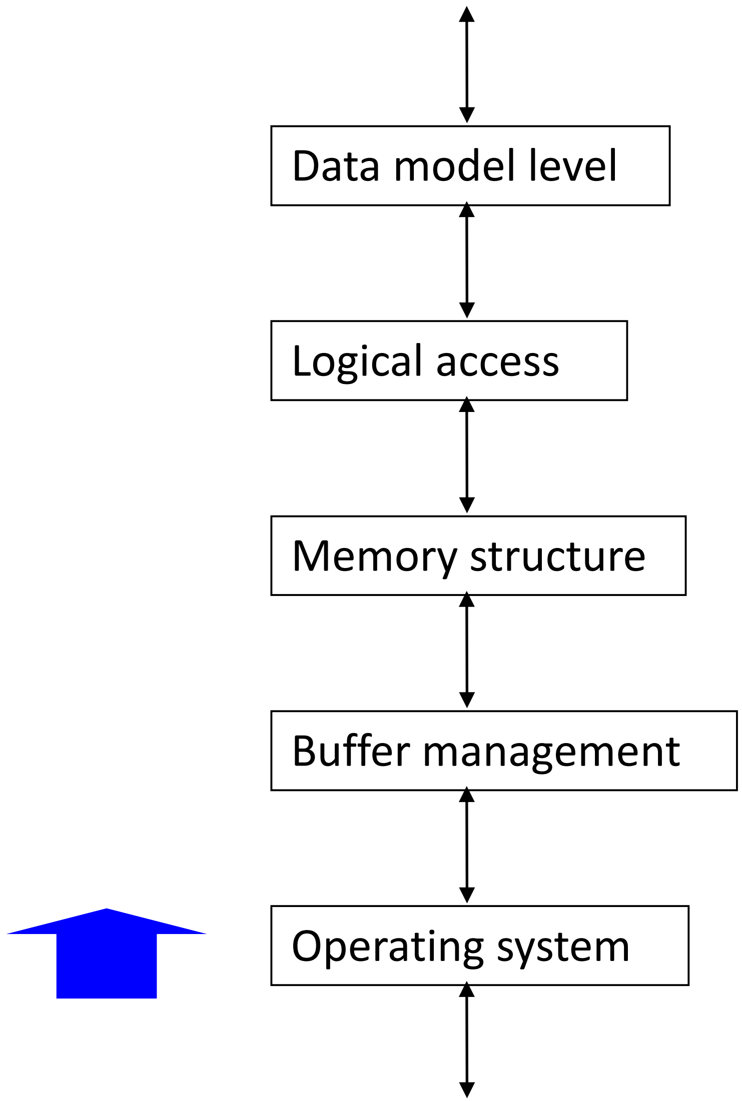
B) GMR

Which new technology may revolutionize data access in the upcoming years?

C) RAID

D) SCSI

Bottom Up



Next week:
Mapping records and blocks to files

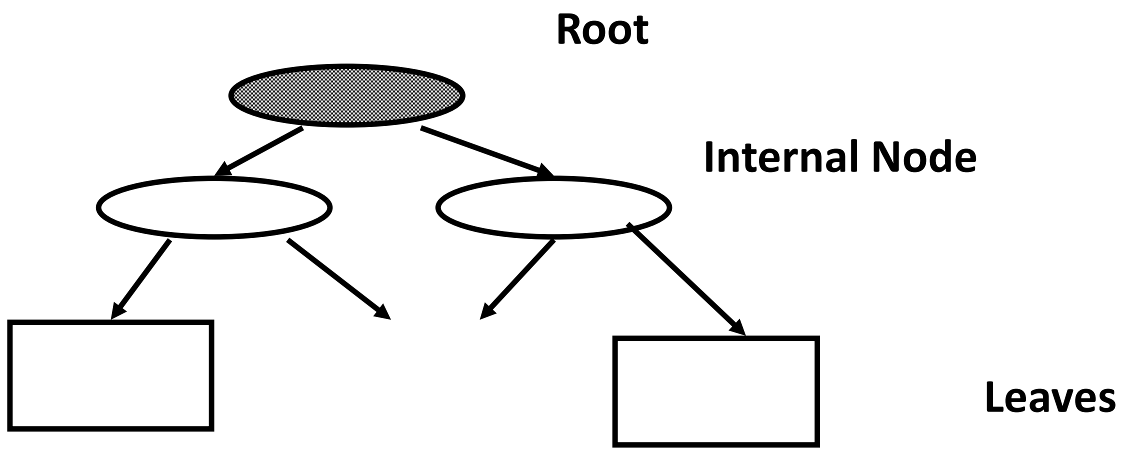
- Sequential File
  - Access to records by record/tuple identifier (“rid” or” tid”)
      - »

1522	Bond	...
123	Mason	...
...	...	...
1754	Miller	...

  - Operations:
    - INSERT(Record): Move to end of file and add, O(1)
    - SEEK(TID): Sequential scan, O(n)
      - » FIRST (File): O(1)
      - » NEXT(File): O(1)
      - » EOF (File): O(1)
    - DELETE(rid): Seek rid; flag as deleted
    - REPLACE(rid, Record): Seek rid; write record
      - » What happens if records have variable length?

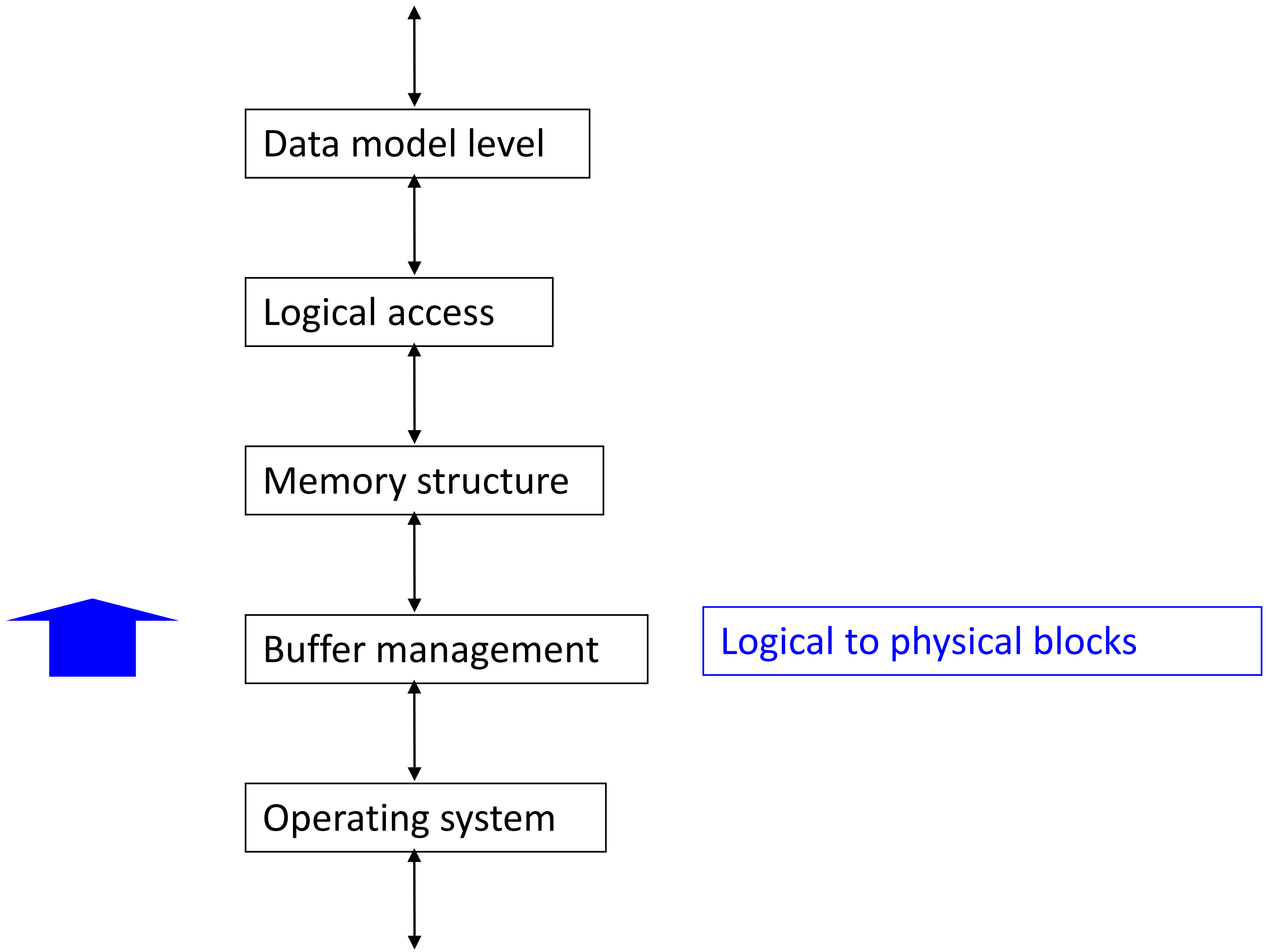
Introduction to Access Methods 2

- Index File
- Access by search key (note: not necessarily data model key)

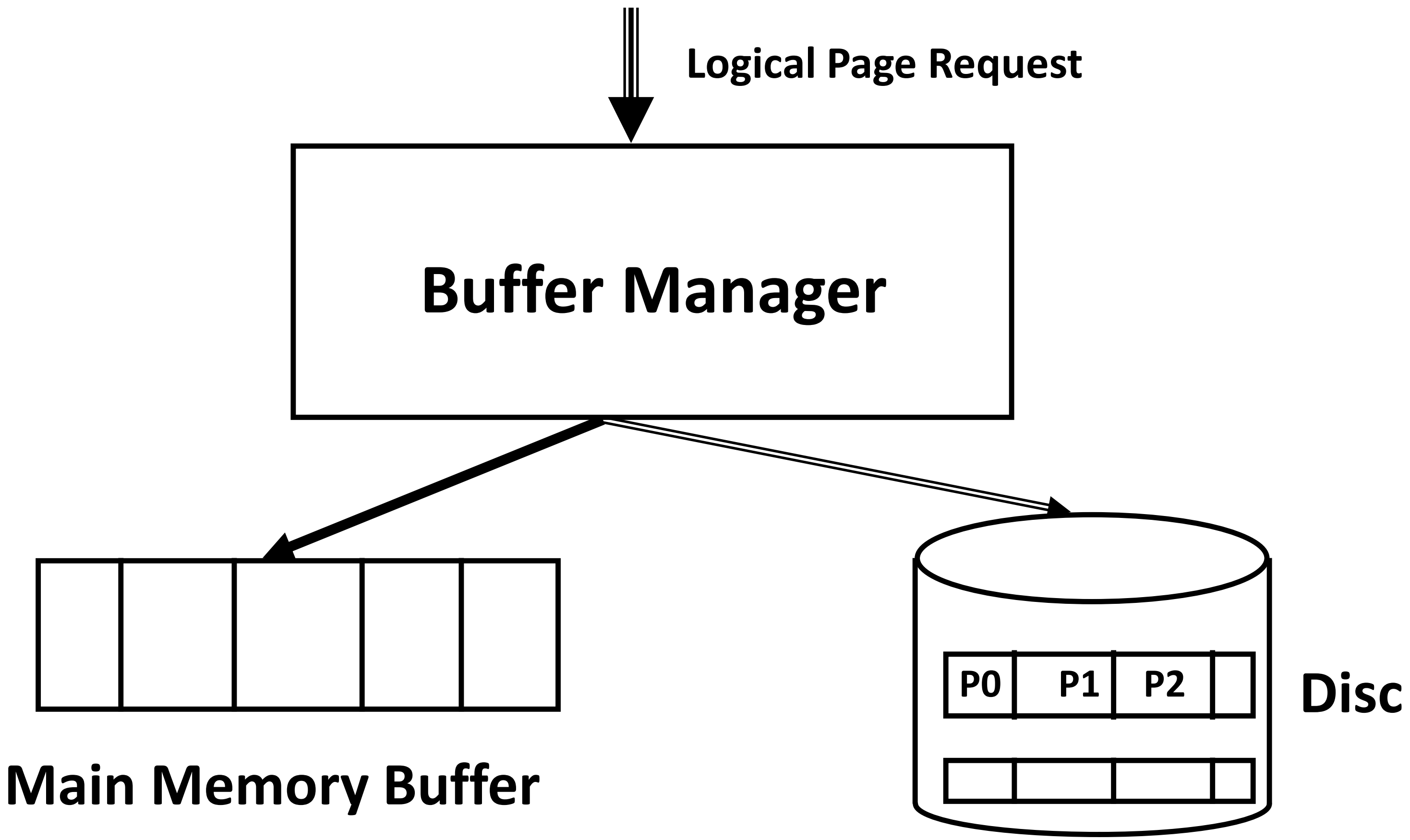


- Operations:
  - SEEK(key): Use order in TIDs; O(log(n))
    - Only if tree is perfectly balanced
  - INSERT(key): Seek key and insert; might require restructuring
  - DELETE(key): Seek key and remove; might require restructuring
  - REPLACE(key): Seek key and write
    - Variable size keys?

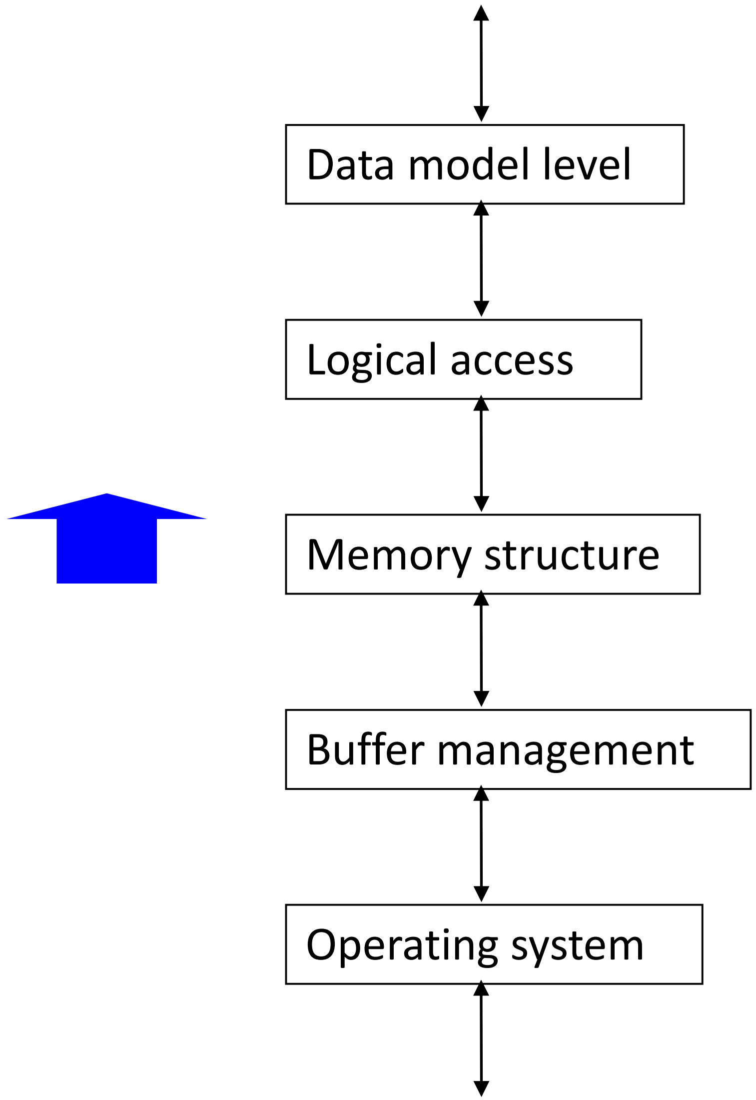
Bottom Up


Data model level
Logical access
Memory structure
Buffer management
Logical to physical blocks
Operating system

Caching = Buffer Management



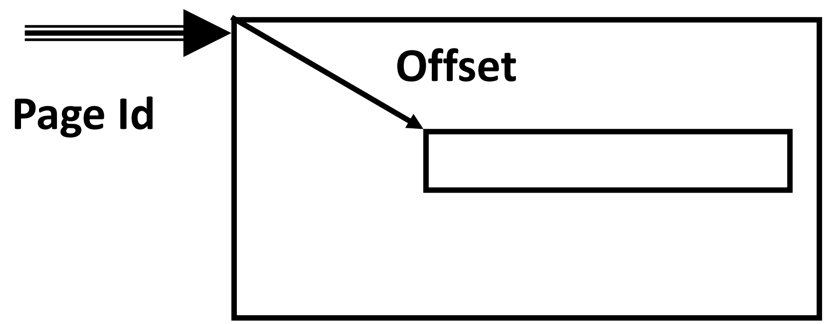
Bottom Up



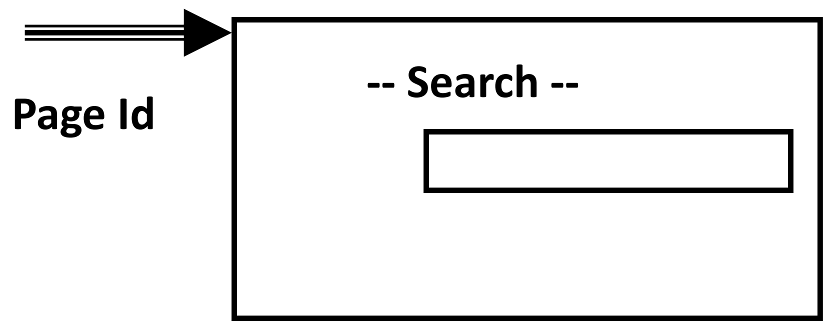
In 2 weeks:
From records to virtual blocks;
index structures

Record Addressing

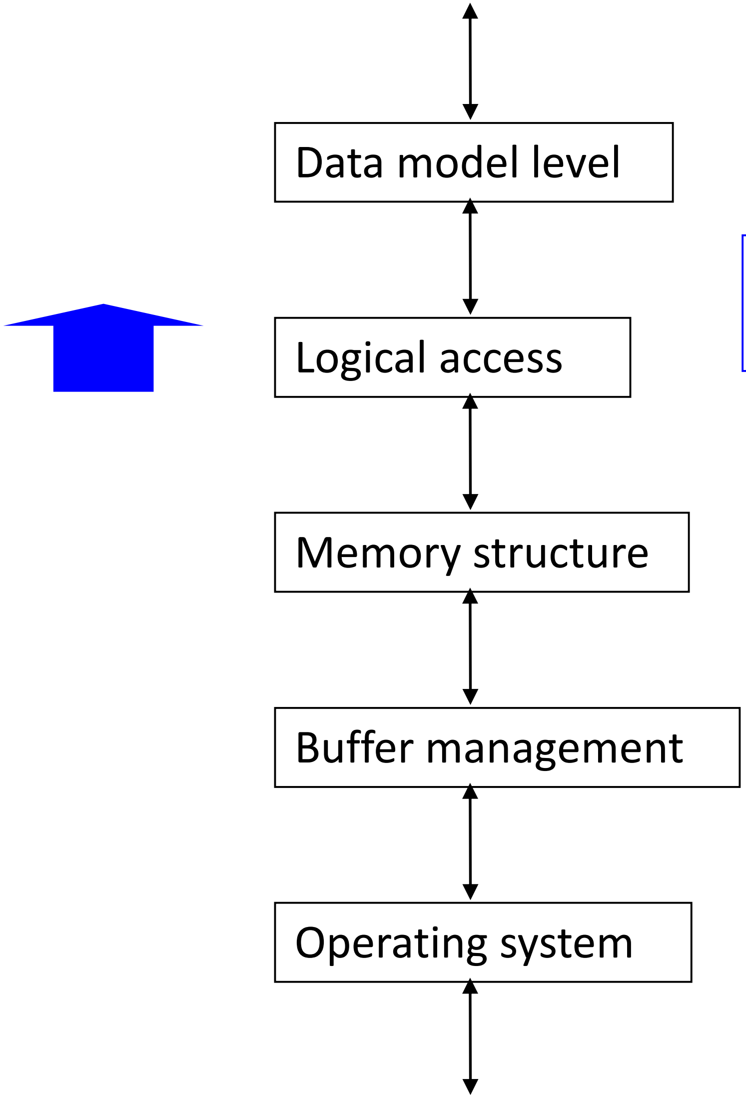
- Mapping alternatives
  - absolute addressing: rid = <PageId, Offset>



  - absolute addressing + search: rid = <PageId>

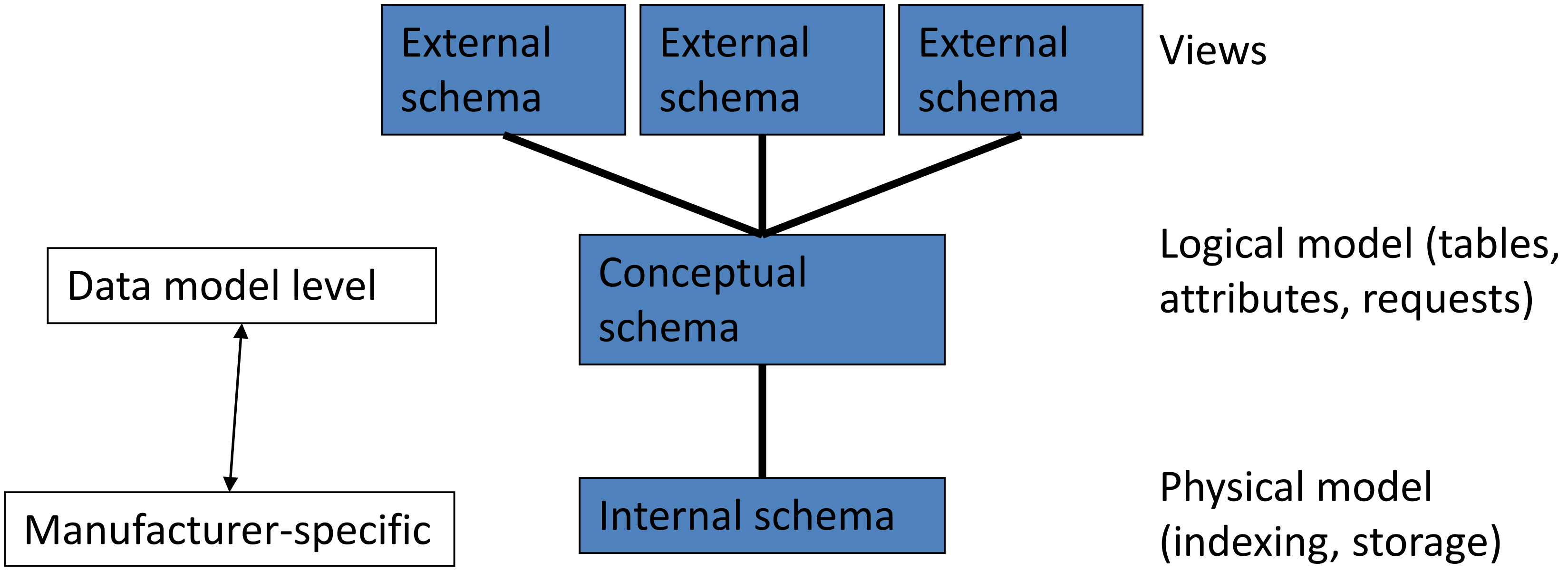


Bottom Up

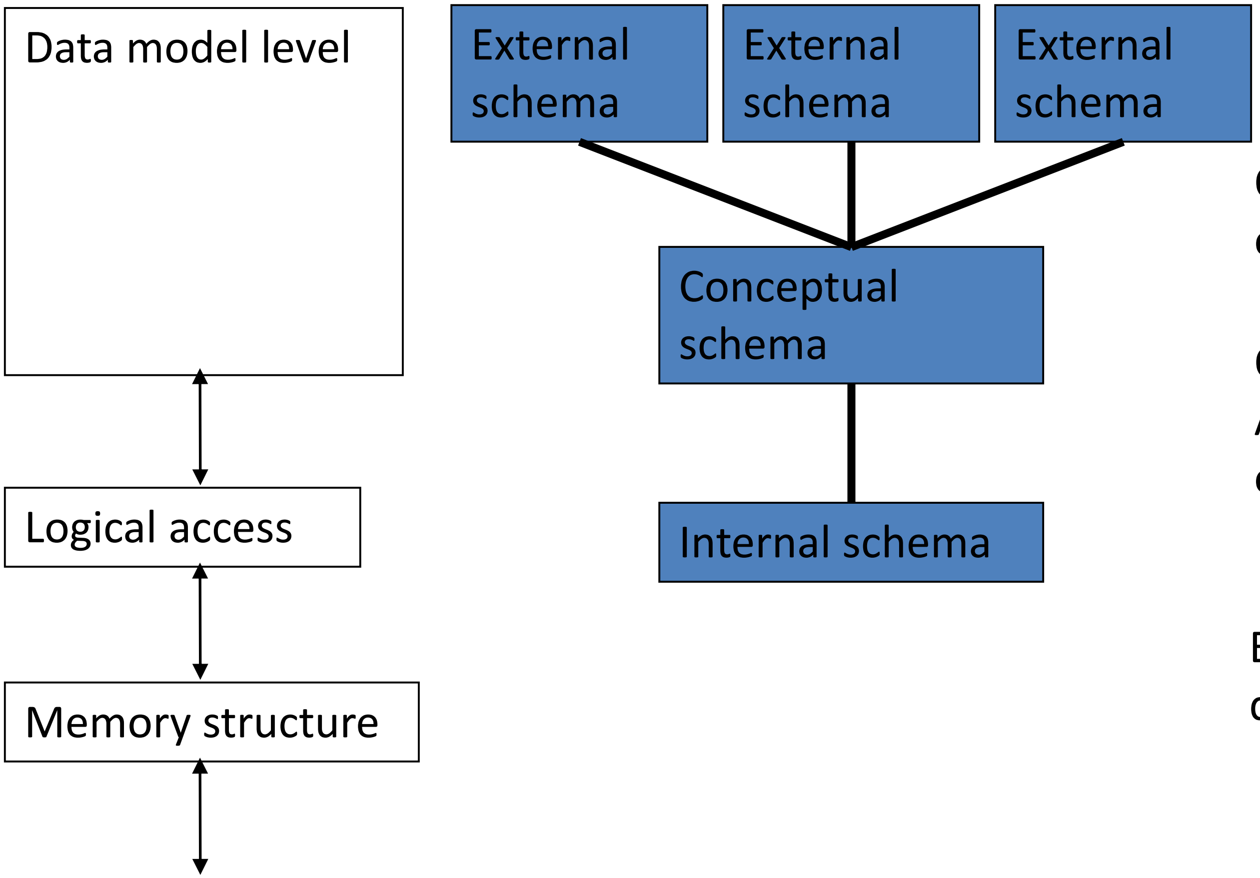


In ca. 3-4 weeks:
Query Processing/Optimization

Three Layer Model



Query Processing



Query transformation; View expansion

Query translation:
Access paths, choosing index, order of joins, optimization

Execution of the chosen query plan

Queries and Languages

- User Languages:
  - SQL, QUEL, Embedded-SQL, 4GL
- Data Definition Language (DDL)
  - Create, delete, change relations and other DB objects
    - Tablespaces and partitions
    - Indexes and views
    - Users
  - Authorization and authentication
  - (Manage processes, system parameter, transaction isolation level, …)
  - Manipulate metadata
- Data Manipulation Language (DML)
  - Read, delete, create, update tuples
  - Manipulate data

Query Processing

- Declarative query

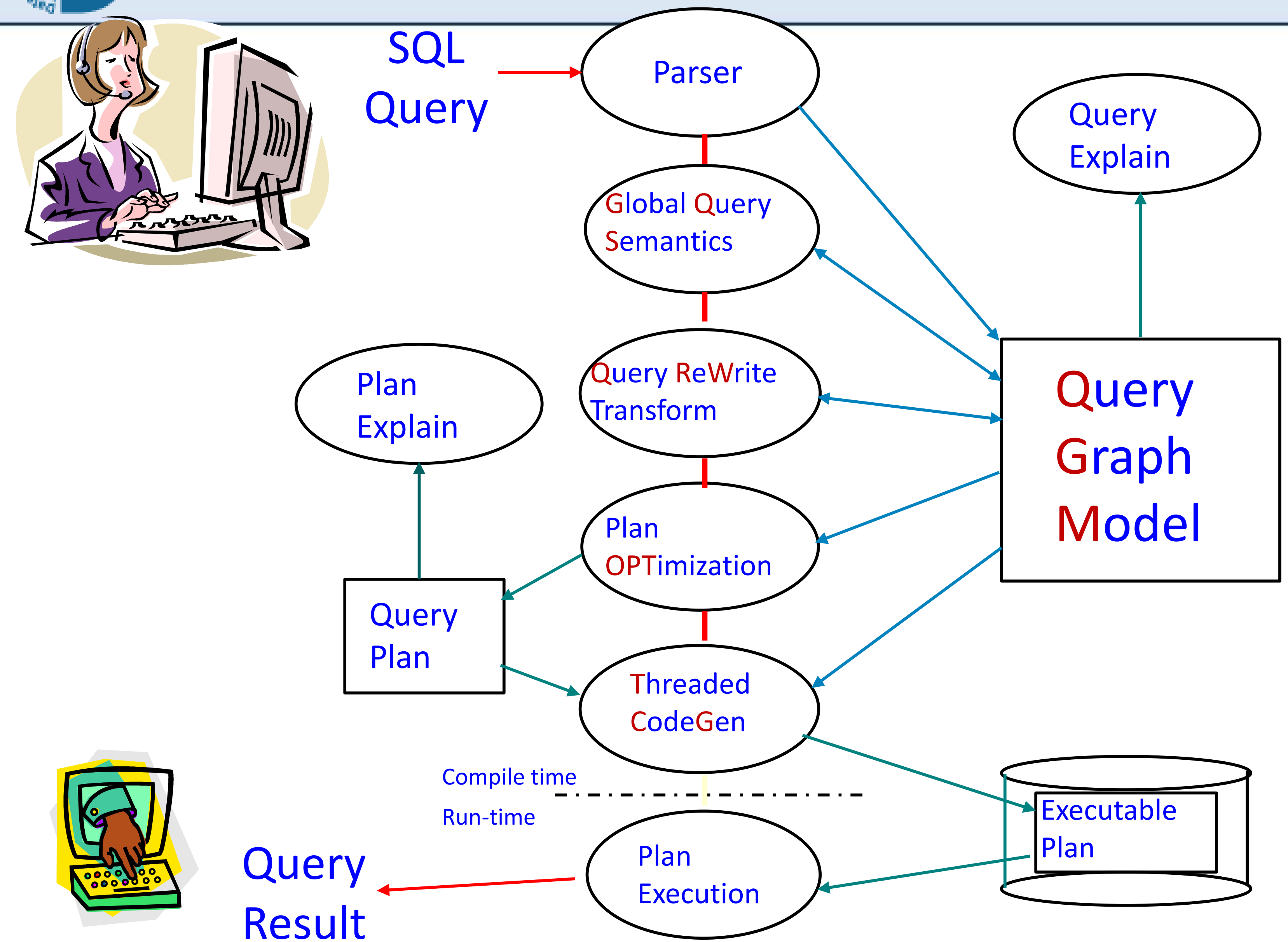
```
SELECT Name, Address, Checking, Balance
FROM   customer C, account A
WHERE Name = "Bond" and C.Account# = A.Account#
```

- Generate a "Query Execution Plan"

```
FOR EACH c in CUSTOMER DO
   IF c.Name = "Bond" THEN
       FOR EACH  a IN ACCOUNT DO
           IF a.Account# = c.Account# THEN
»                  Output ("Bond", c.Address, a.Checking, a.Balance)
```

- Query Execution Plan (QEP)
  - Procedural Specification
  - Semantically equivalent to query

Query Processing



Query Processing – the Problem

- There are many, many possible QEPs for a given query

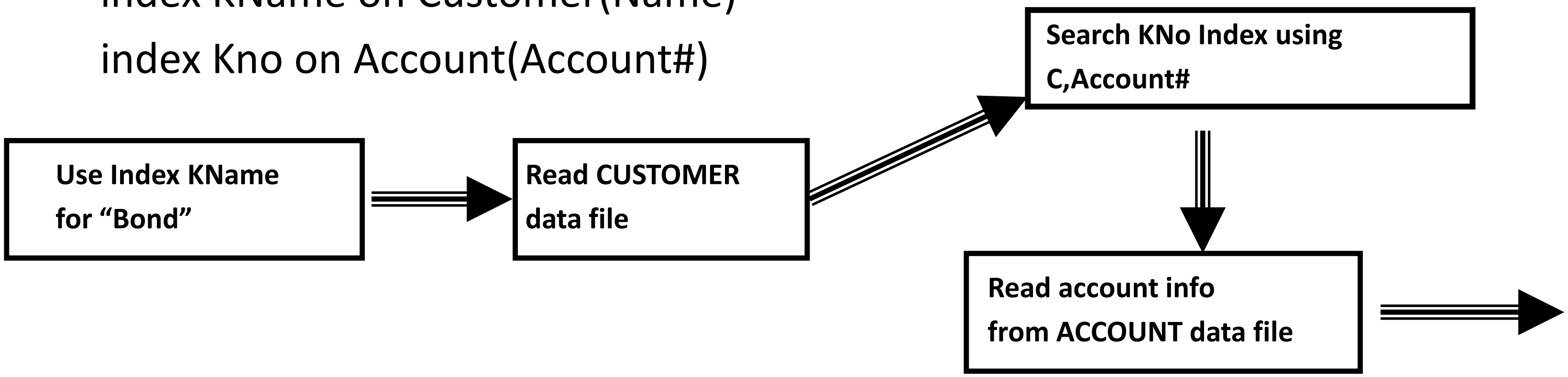
```
FOR EACH a in ACCOUNT DO
    FOR EACH  c IN CUSTOMER DO
        IF a.Account# = c.Account# THEN
            IF c.Name = "BOND" THEN
                Output ("Bond", c.Address, a.Checking, a. Balance)

FOR EACH c in CUSTOMER WITH Name="Bond" BY INDEX DO
    FOR EACH  a IN ACCOUNT DO
        IF a.Account# = c.Account# THEN
            Output ("Bond", c.Address, a.Checking, a. Balance)
```

Query Optimization

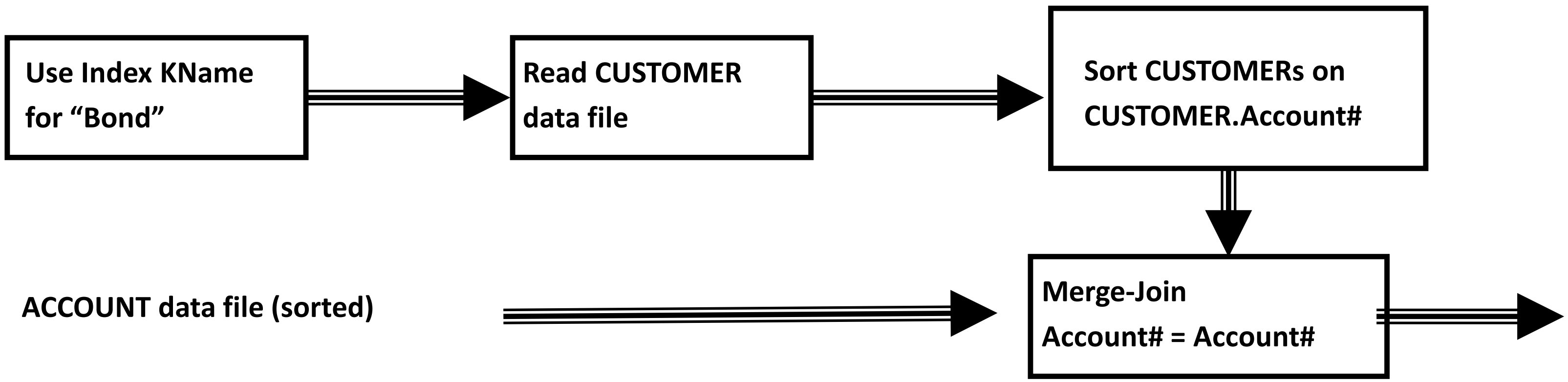
QEP 1: Suppose

index KName on Customer(Name)

index Kno on Account(Account#)



QEP 2: Account File is sorted



Query Optimization

- Goal: Find the “best” QEP
  - Fastest
  - Quickest first response
  - absolute/average

- One Method: Prove equivalence by rewriting algebra terms
  - P1: σName=Bond(Account ⋈ Customer)
  - P2: Account ⋈ σName=Bond(Customer)

- Another Method: Enumerate all possible QEP and find “best”
  - Usually cannot be performed exhaustively
  - Optimization Methods: Dynamic, Greedy, Genetic Algorithms, Tabu-Search, etc.

- Good Approximation for “best”: Minimize cost or size of intermediate results
  - This is a local criteria
  - Might mislead
    - Expensive subplan with sorted result
    - Cheap subplan with unsorted result
  - Commercial DBMS use interesting properties (like sort order) and only compare QEPs with the same properties

Join Methods

- Suppose the previous query would contain no selection
- Can‛t we do better than “1.2E8 comparisons for join, …”
- Nested loop join has complexity O(m*n)
  - m,n: sizes of joined relations
- Other methods
  - Sort-merge join
    - First sort relations in O(n*log(n)+m*log(m))
    - Merge results in O(m+n)
    - Might be better, but ...
      - » external sorting is expensive
      - » Doesn‛t pay off if relations already in cache
  - Hash join, …
- Note: Usual Complexities Measure Number of Comparisons
  - This is “main-memory” viewpoint
  - Should not be used for I/O tasks
  - For data intensive operations, we need to look at number of I/Os (or communications) as bottleneck

SQL is declarative: specifies what data is needed, not how to get it

```
SELECT   DISTINCT o.name,a.driver
FROM     owner o,
                  car c,
                  demographics d ,
                  accidents a
WHERE
         c.ownerid = o.id AND
         o.id = d.ownerid AND
         c.id = a.id AND
         c.make = 'Mazda' AND
         c.model = '323' AND
         o.country3 = 'EG' AND
         o.city = 'Cairo' AND
         d.age < 30 ;
```

Find owner and driver of Mazda 323s that have been involved in accidents in Cairo, Egypt, where the age of the driver has been less than 30

Why does Optimization matter?

```
SELECT o.name,a.driver
FROM owner o,
        car c,
        demographics d ,
        accidents a
WHERE
     c.ownerid = o.id AND
     o.id = d.ownerid AND
     c.id = a.id AND
     c.make = 'Mazda'
     AND
     c.model = '323' AND
     o.country3 = 'EG'
     AND
     o.city = 'Cairo' AND
     d.age < 30 ;
```

2 hours and 20 minutes

```
                 RETURN(0)
                 >7 [EO]
                    NLJN[1]
                  0.000781925
         /------------+----------\
        7                        N/A
      NLJN[2]                  FETCH[0]
    0.000781925                    1
     /------+-----\                  |
   563        ~0.0124334        N/A
 TEMP_SCAN[0]  FETCH[1]      ISCAN[1]
 0.628674      0.00124377        1
    |                            |
   |
   563         ~14422              N/A
  SORT[0]      ISCAN[2]
  ACCIDENTS
 0.628674      753.723
  2.60179e+06
    |              |
   563           N /A
  HSJN[1]        CAR
 0.628674      867276
 /------+------\
162015      1000
SCAN[1]    FETCH[0]
164832     2.3113
  |          |
605999     1000
DEMO.      ISCAN[2]
605999     2.3113
             |
            N/A
           OWNER
           606000
```

50 seconds

```
            RETURN(0)
             >7 [EO]
             NLJN[1]
             22.4859
          /-----+-----\
         7           N/A
  TEMP_SCAN[0]   FETCH[0]
     22.4859          1
        |             |
        7            N/A
     SORT[0]      ISCAN[1]
     22.4859          1
        |             |
        7            N/A
     HSJN[1]      ACCIDENTS
     22.4859      2.60179e+06
   /------+-----\
162015         10
SCAN[1]       HSJN[1]
164832        82.6686
  |           /--+--\
605999     14422      1000
DEMO.    FETCH[0]   FETCH[1]
605999   22237.8    2252.79
            |          |
          14422      1000
        ISCAN[2]   ISCAN[1]
        22237.8    1325.92
            |          |
           N/A        N/A
           CAR       OWNER
         867276     606000
```

- Statistics are useful but
  - Need to be stored and accessed
  - Need to be kept current
  - Difficult problem!
- Query transformation and optimization needs data dictionary
  - Semantic parsing of query: Which relations exist?
  - Which indexes exists?
  - Cardinality estimates of relations?
  - Size of buffer for in-memory sorting?
  - ...

Table_name	Att_name	Att_type	size	Avg_size
Customer	Name	Varchar2	100	24
Customer	account#	Int	8	8
Customer	...			

Access Control

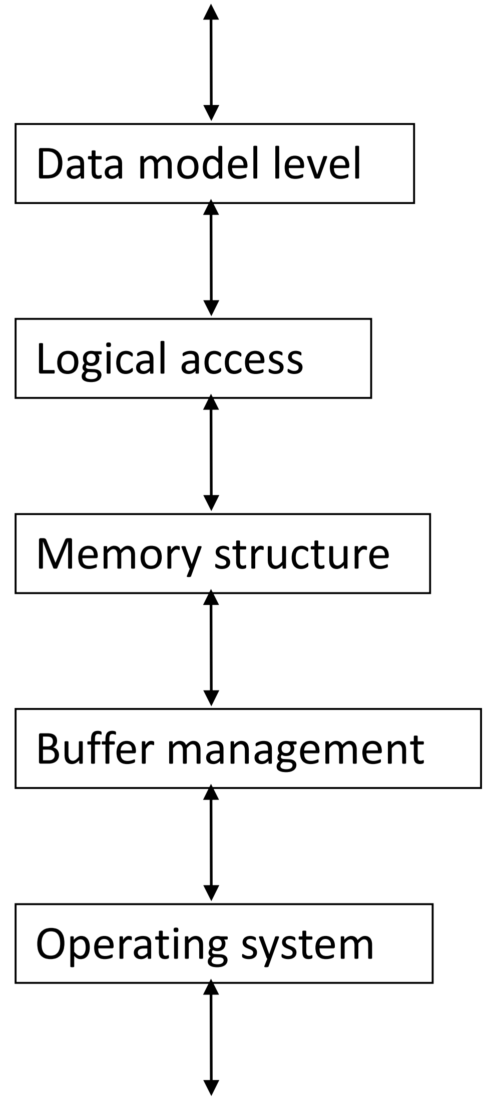
- Read and write access on objects
- Read and write access on system operations
  - Create user, kill session, export database, …
- GRANT, REVOKE Operations
- Example:

  GRANT ALL PRIVILIGES ON ACCOUNT TO Freytag WITH GRANT OPTION

  – "User Freytag has Read/Write access to the ACCOUNT relation
  – it is possible for Freytag to grant this rights to others"
- No complete protection
  - Granularity of access rights usually relation/attribute – not tuple
  - Access to data without DBMS
  - Ask several questions to derive requested data
  - In addition: file protection, encryption of data

Bottom Up

In ca. 8 weeks:
Transactions, Schedules,
Recovery



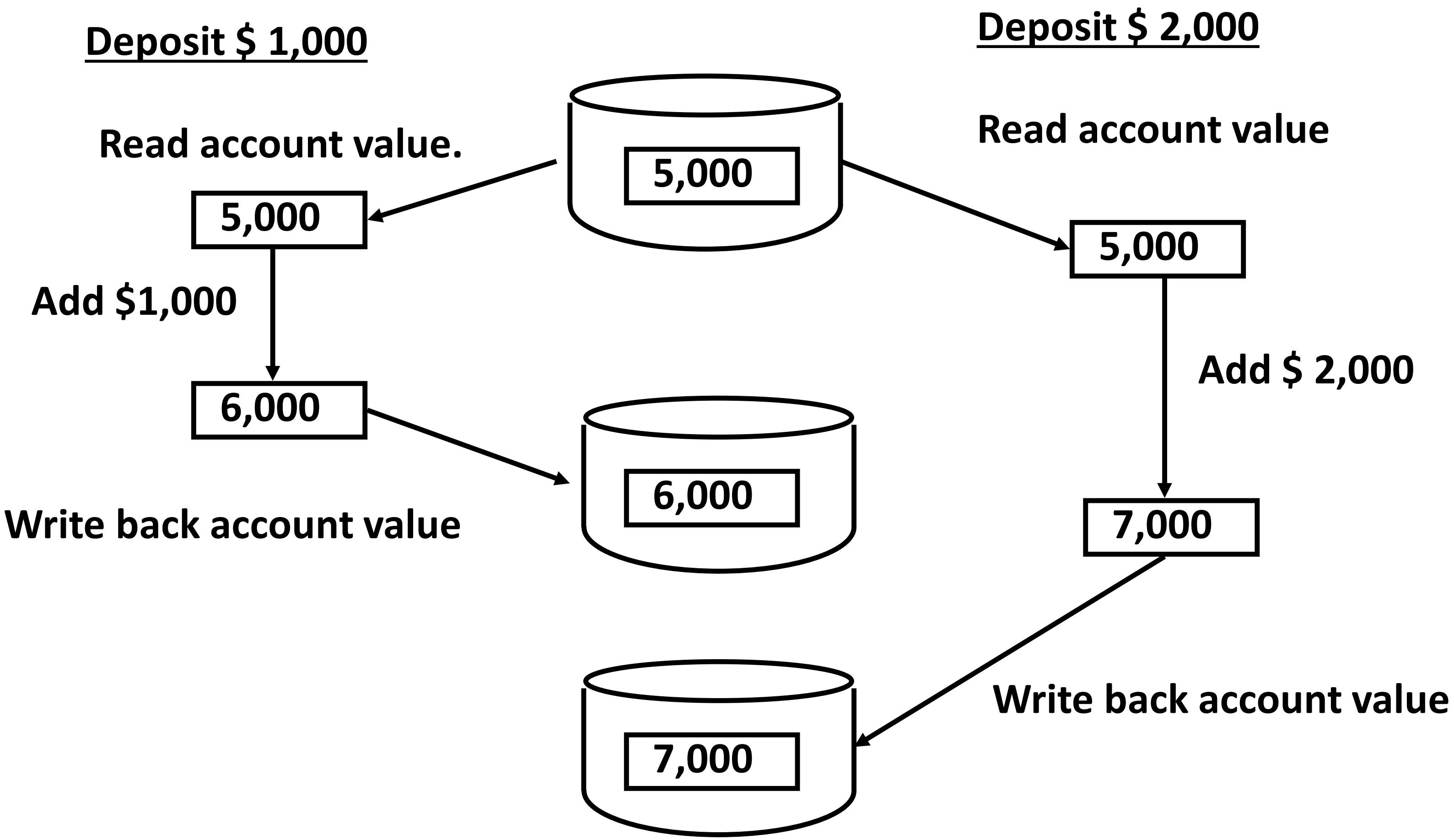
Transactions (TA)

- Transaction: "Logical unit of work"

```
Begin_Transaction
    UPDATE ACCOUNT
        » SET Savings = Savings + 1M
        » SET Checking = Checking - 1M
    WHERE Account# = 007;
    INSERT JOURNAL <007, NNN, "Transfer", ...>
End_Transaction
```

- ACID properties:
  - A: Atomic Execution
  - C: Consistent DB state after updates
  - I: Isolation: No influence on result by concurrent executions
  - D: Durability: Updates are reflected in the database

Why Transactions? Lost Update Problem



```
T1: read A;            T2: read B;
    A := A - 10;           B := B - 20;
    write A;               write B;
    read B;                read C;
    B := B + 10;           C := C + 20;
    write B;               write C;
```

Schedule S_1		Schedule S_2		Schedule S_3	
T_1	T_2	T_1	T_2	T_1	T_2
read A		**read** A		**read** A	
$A-10$			**read** B	$A-10$	
write A		$A-10$			**read** B
read B			$B-20$	**write** A	
$B+10$		**write** A			$B-20$
write B			**write** B	**read** B	
	read B	**read** B			**write** B
	$B-20$		**read** C	$B+10$	
	write B	$B+10$			**read** C
	read C		$C+20$	**write** B	
	$C+20$	**write** B			$C+20$
	write C		**write** C		**write** C

??

- When are two schedules „conflict-free"?
  - when they are serializable
  - when they are equivalent to a serial schedule
  - Prove serializability of schedules

- Checking after execution is wasteful
  - Synchronization protocols
  - Guarantee only serializable schedules
  - Require certain well-behavior of transactions
  - Methods
    - Two phase locking
    - Multi-version synchronization
    - Timestamp synchronization

Deadlocks

- Locking is powerful, yet caution is necessary

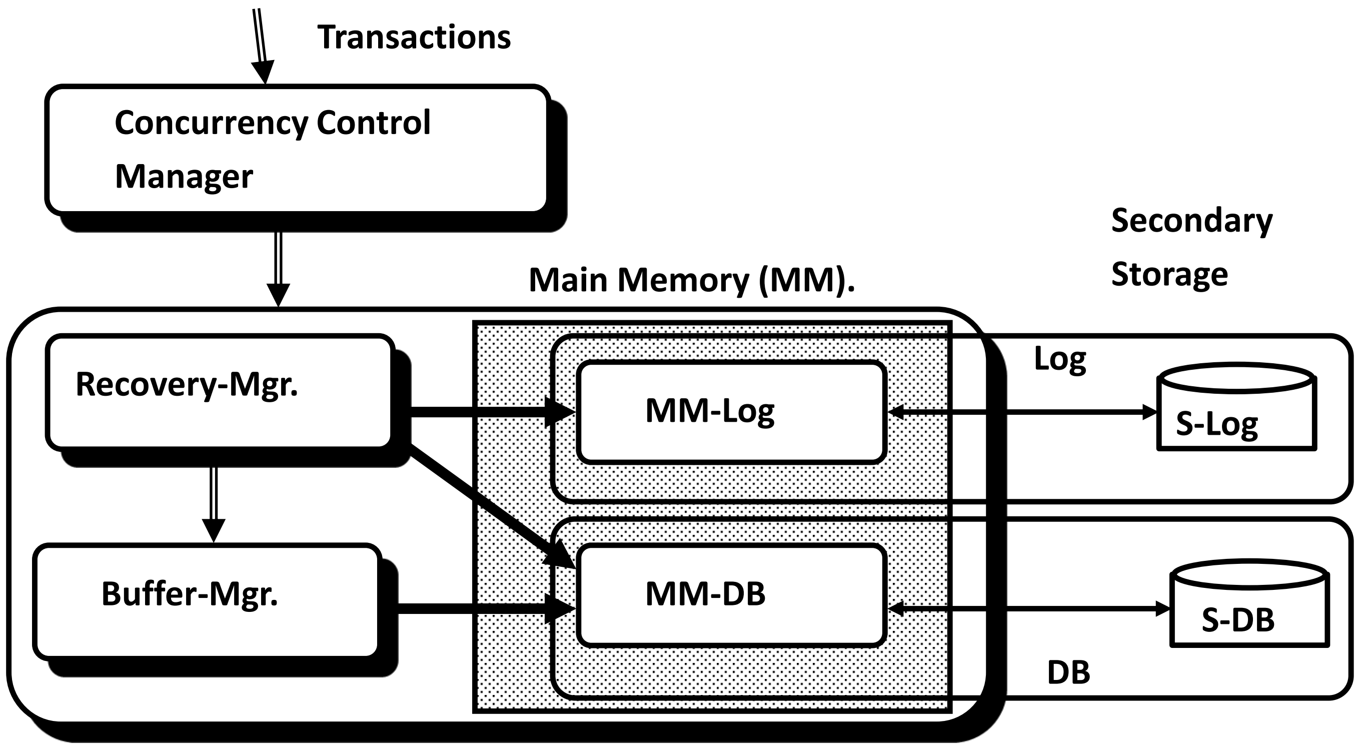
$$
\begin{array}{llll}
T_1: & \texttt{lock}\ A; & T_2: & \texttt{lock}\ B; \\
 & \ldots; & & \ldots; \\
 & \texttt{lock}\ B; & & \texttt{lock}\ A; \\
 & \ldots; & & \ldots; \\
 & \texttt{unlock}\ A; & & \texttt{unlock}\ B; \\
 & \texttt{unlock}\ B; & & \texttt{unlock}\ A;
\end{array}
$$

- Runs into deadlock
- Deadlocks need to be discovered by database
  - If not avoided by synchronization protocol – very costly
- Manage locks, lock-waits, lock-times, etc.
  - Data dictionary

Transaction Manager

- Synchronization is the “I” in ACID
- Transaction manager is responsible for
  - Concurrency control
    - Concurrent access to data objects
    - Synchronization & locking
    - Deadlock detection and deadlock resolution
  - Logging & recovery
    - Compensate for system und transaction errors
    - Based on log files (redundant storage of information)
    - Error recovery protocols – undo; redo

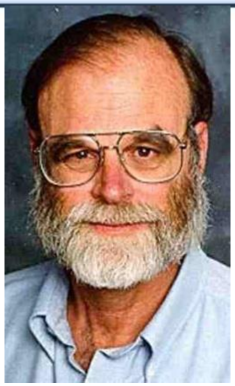
Recovery – Broad Principle



- Store data redundantly
  - Save old values
- Uses different file format, adapted to different access characteristics
  - Sequential write, rare reads

Trivia 2

Pat Selinger



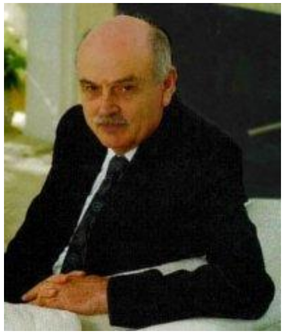
Jim Gray

Who made groundbreaking inventions in the area of:

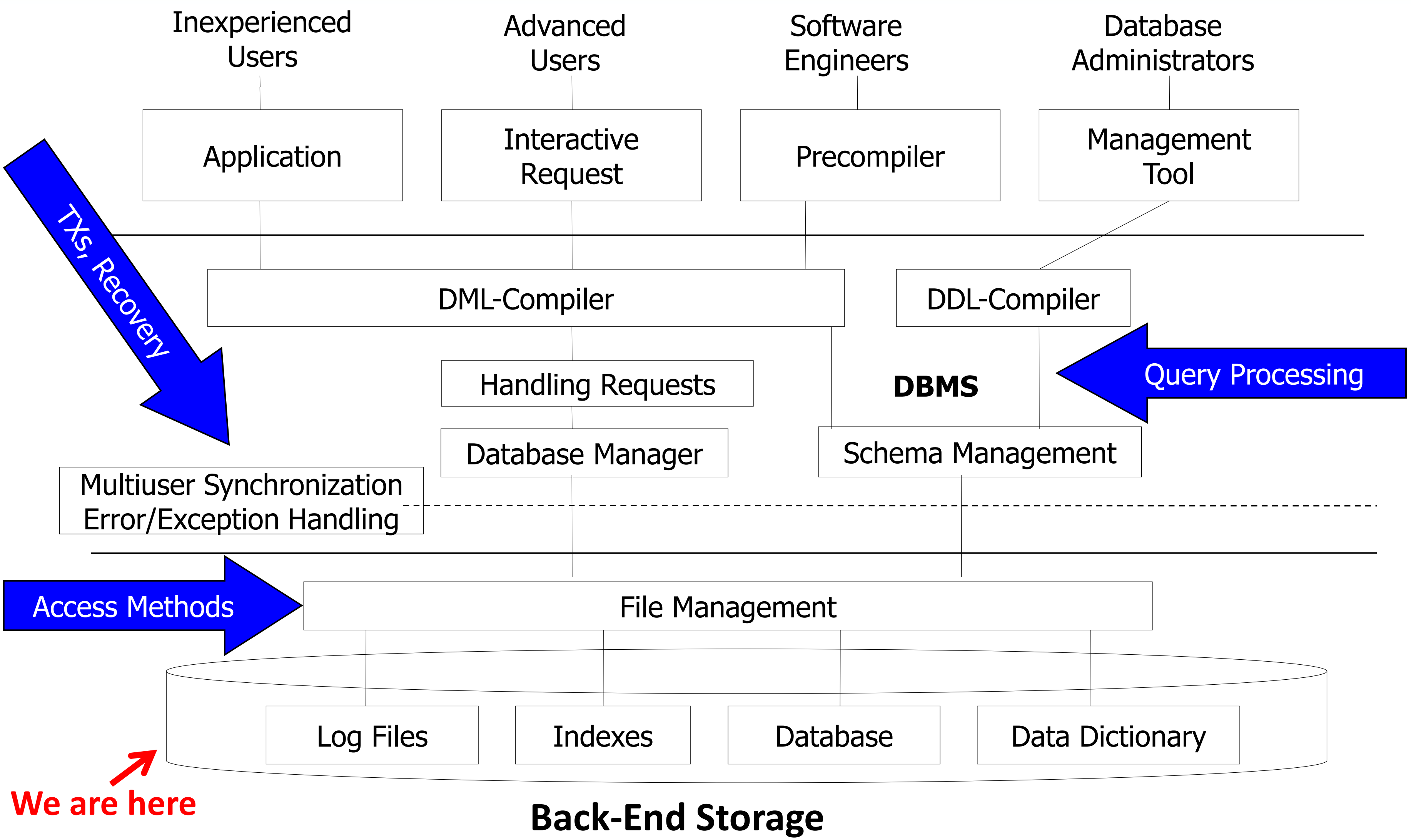
- Query Optimization
- Transaction Processing
- Indexing
- Relational Databases

Rudolf Bayer

Edgar Codd



What is next?


Inexperienced Users
Advanced Users
Software Engineers
Database Administrators
Application
Interactive Request
Precompiler
Management Tool
TXs, Recovery
DML-Compiler
DDL-Compiler
Handling Requests
DBMS
Query Processing
Database Manager
Schema Management
Multiuser Synchronization
Error/Exception Handling
Access Methods
File Management
Log Files
Indexes
Database
Data Dictionary
We are here
Back-End Storage