Datenbanksysteme II: Overview and General Architecture

Ulf Leser

Table of Content

- Storage Hierarchy
- 5-Layer Architecture
- Overview: Layer-by-Layer

2010: Price versus speed

Really expensive	Register	1-10 ns/byte
Very expensive	Cache	10-60 ns/cache line
~ 200 € / GB	Main Memory	[illegible]300 ns/block
~ 1 € / GB	Disk	10[illegible] ms/block
< 1€/GB	Tape	

Difference ~10^5

Difference ~10^4

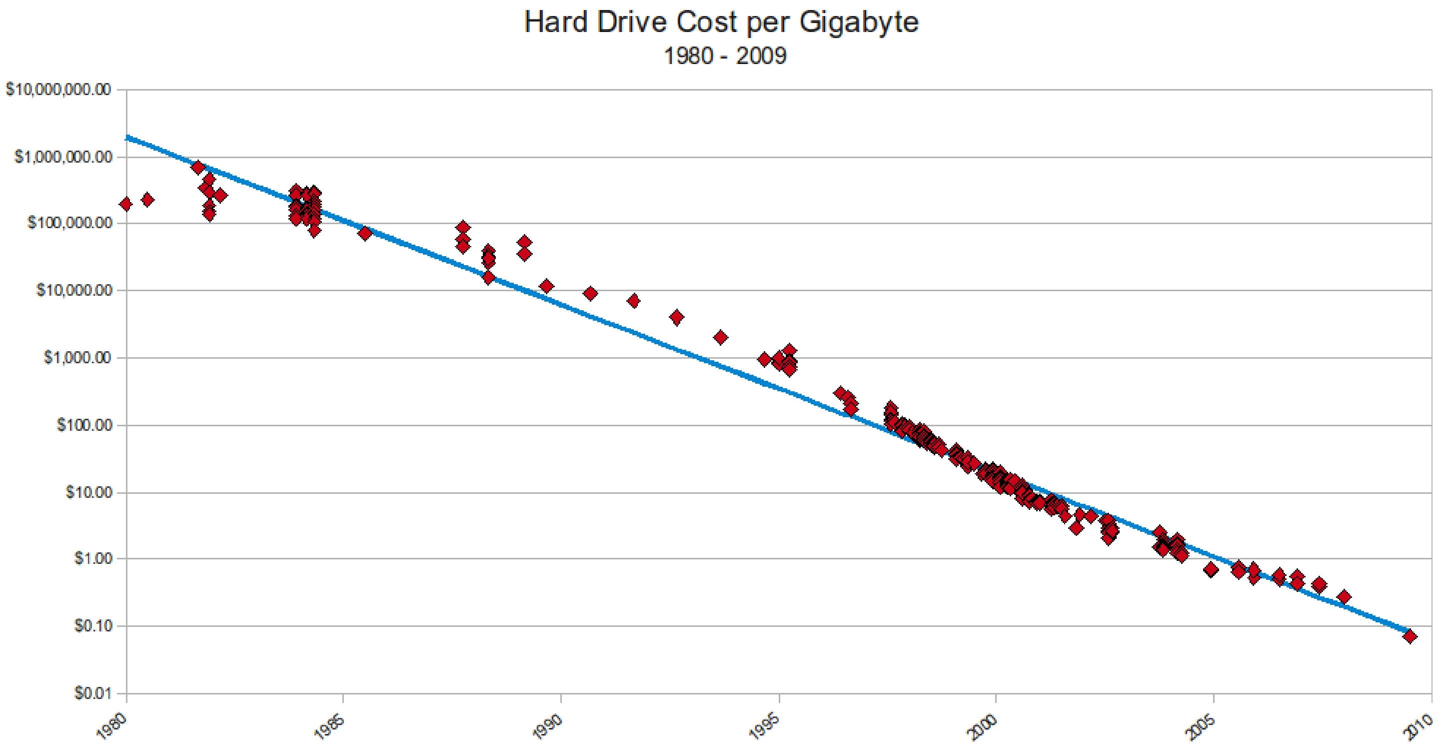
2010: Storage Hierarchy

Really expensive	Register	1-4 byte
Very expensive	Cache	1-4 MB
~ 200 € / GB	Main Memory	1-16 GB
~ 1 € / GB	Disk	512GB – 1TB discs
< 1€/GB	Tape	“Infinite” tape robots

2016: Storage Hierarchy

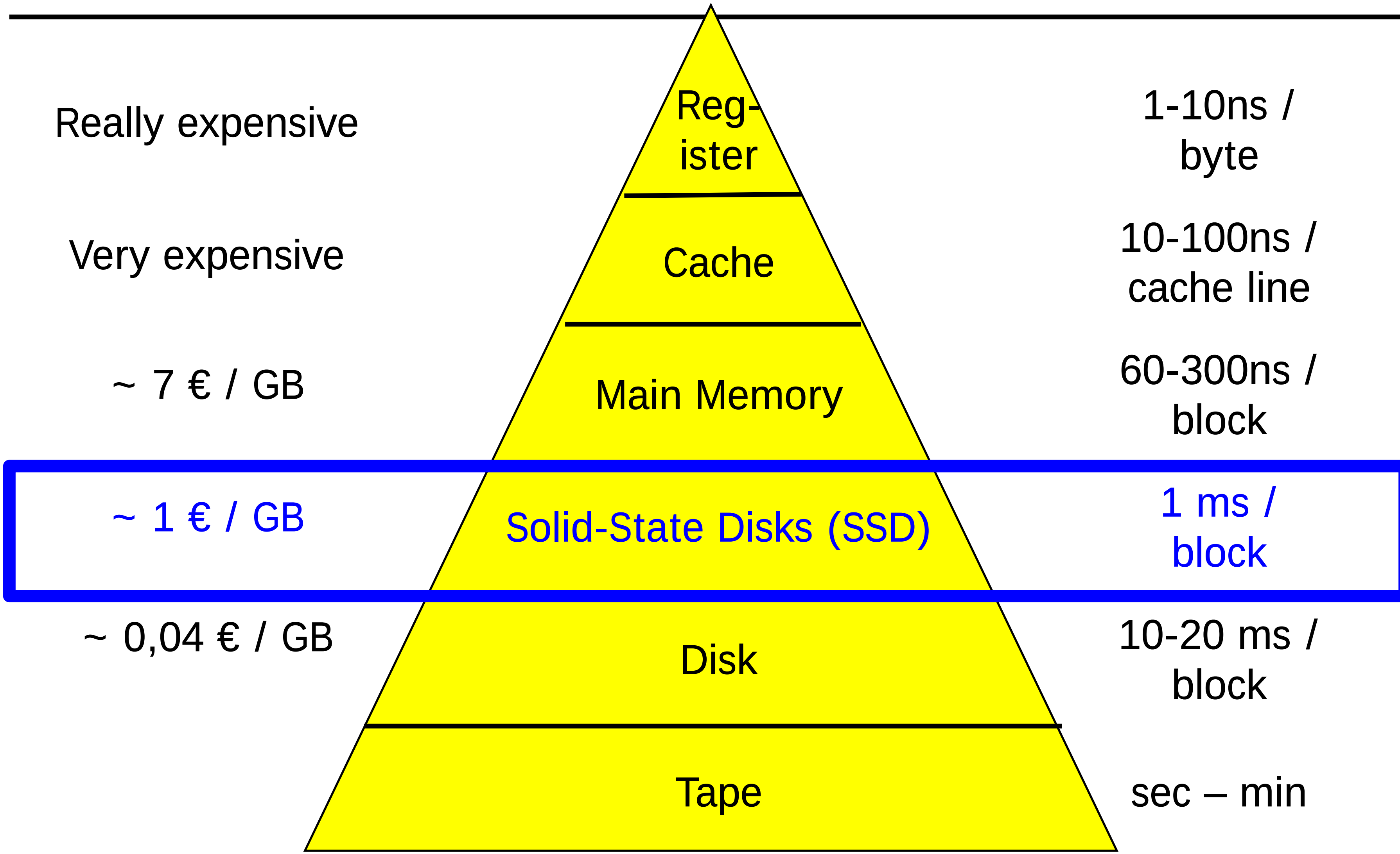
Really expensive	Register	1 – 32 byte
Very expensive	Cache	1-16 MB
~ 7 € / GB	Main Memory	16-256 GB
~ 0,04 € / GB	Disk	1-16 TB
	Tape	“Infinite” tape robots

Costs Drop Faster than you Think



Source: http://analystfundamentals.com/?p=88

New Players


Really expensive
Reg-
ister
1-10ns /
byte
Very expensive
Cache
10-100ns /
cache line
~ 7 € / GB
Main Memory
60-300ns /
block
~ 1 € / GB
Solid-State Disks (SSD)
1 ms /
block
~ 0,04 € / GB
Disk
10-20 ms /
block
Tape
sec – min

New Players

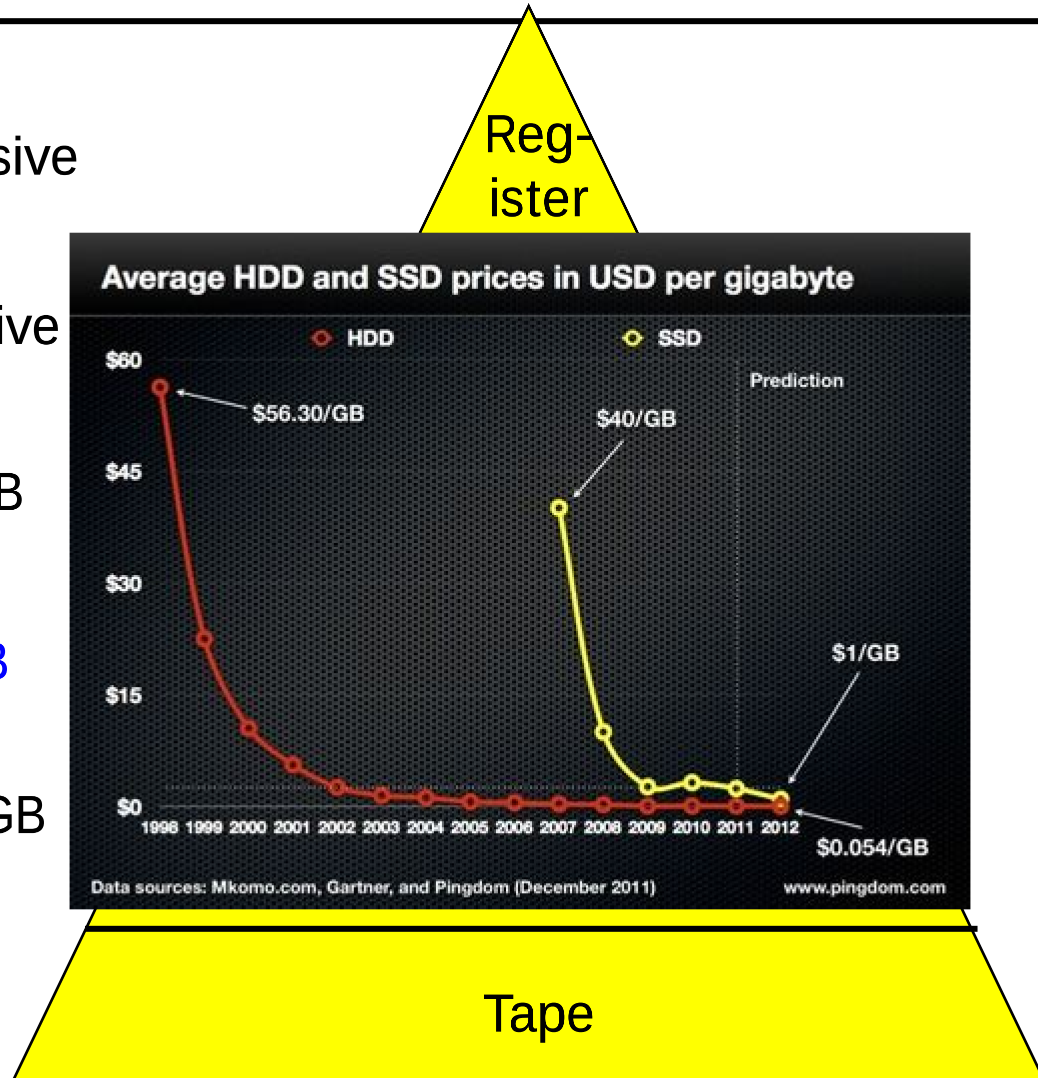
Really expensive

Very expensive

~ 15 € / GB

~ 1 € / GB

~ 0,04 € / GB



1-10ns / byte

10-100ns / cache line

60-300ns / block

1 ms / block

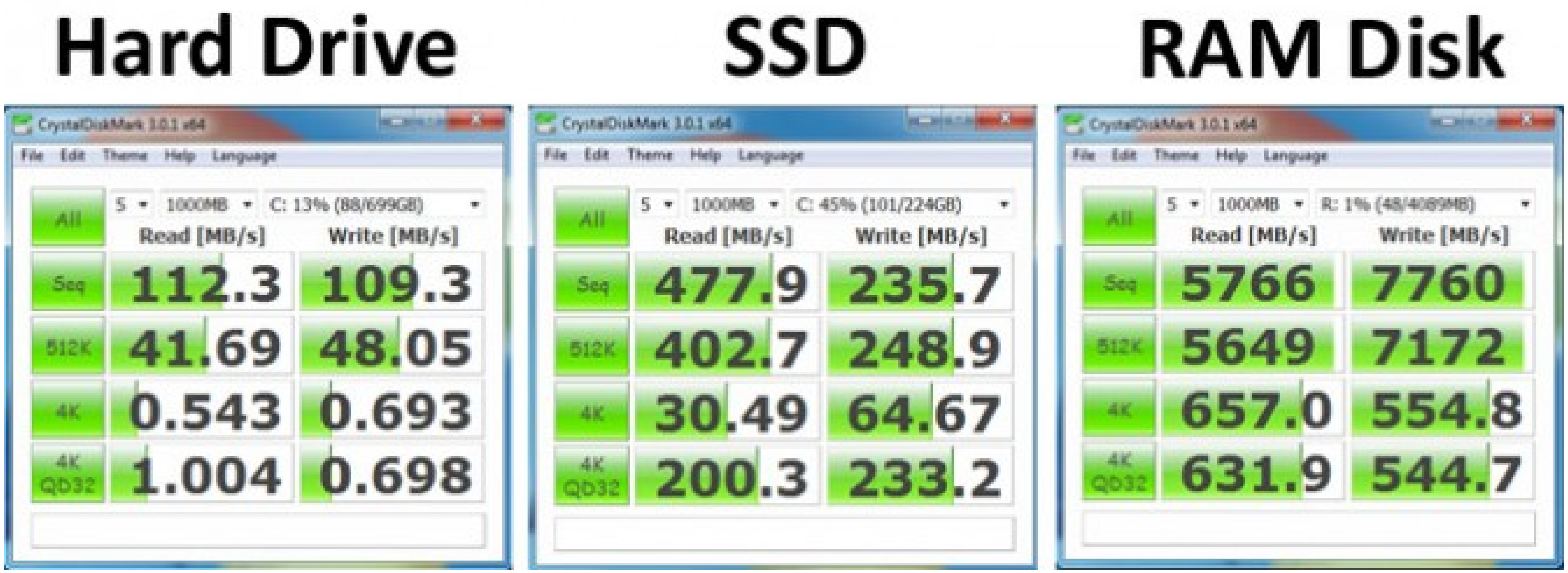
10-20 ms / block

sec – min

Source: http://www.tomshardware.com/news/ssd-hdd-solid-state-drive-hard-disk-drive-prices,14336.html

Characteristics

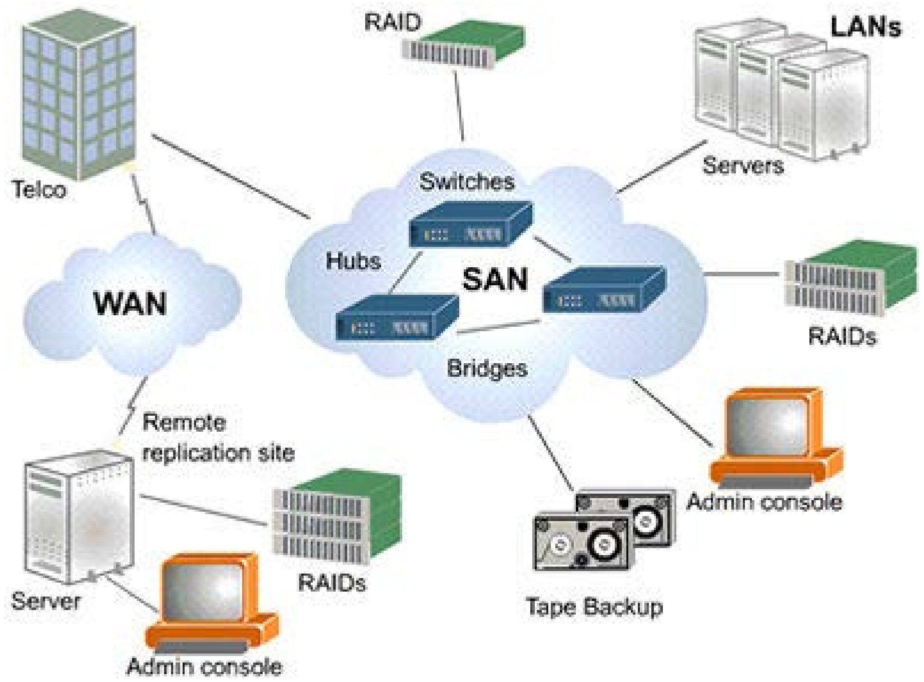
random access != sequential



read != write

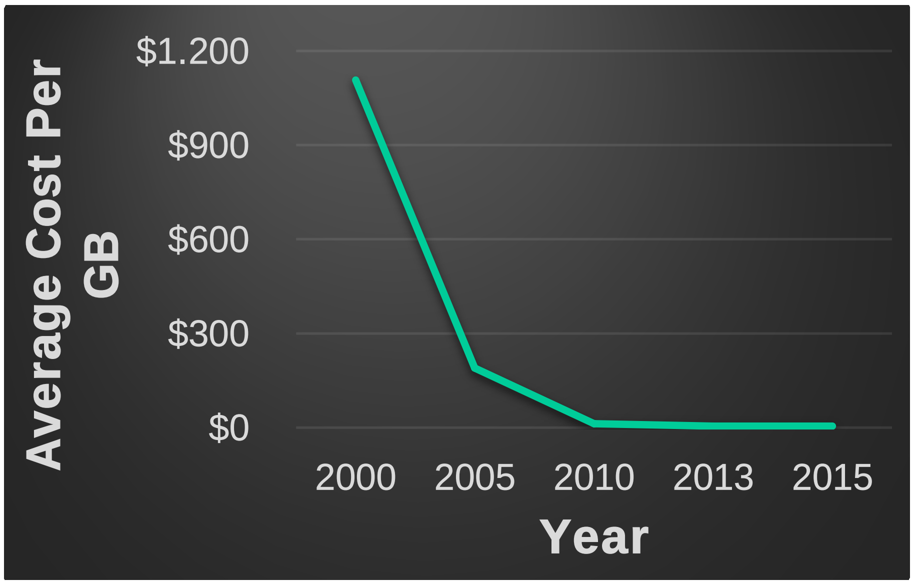
Quelle: http://blog.laptopmag.com/faster-than-an-ssd-how-to-turn-extra-memory-into-a-ram-disk

Storage Area Networks (SAN)



- Dedicated subsystem providing storage (and only storage)
- Virtualization of resources
- Facilitates management, storage assignment, backup etc.

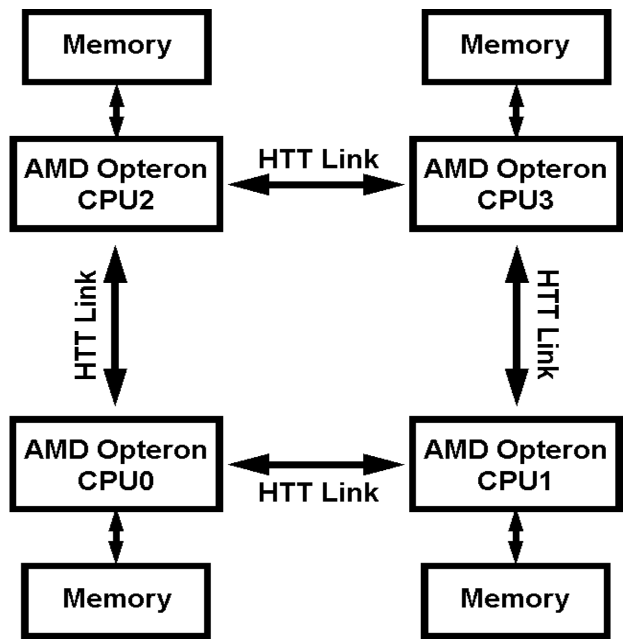
Prize of Main Memory



- 2014: 1TB DRAM ~ 5000€
- 2016: Laptops with 16GB, desktops with 32GB, servers with 128GB
- Guess: 99% of all commercial databases are smaller than 100GB

New: Multi-Core with NUMA

- Modern CPUs can easily have 4-8 cores, each 2 threads
- 4 CPUs in one server is standard
- Add hyper-threading
- 128 hardware threads
- Future: Servers with 1000+ threads (exascale)
  - Network on a chip: Caching, routing, ...



Quelle: http://ixbtlabs.com/articles2/cpu/rmma-numa2.html

Consequences

- Dealing with memory hierarchy is core concern of DBMS
  - Speed of access
  - Durability of changes
- This lecture will mostly focus on disk versus RAM
- Similar problems for cache-RAM, disk-SSD, ...
- Differences exist
  - Block sizes
  - Heterogeneous pattern: Read/write, random-access/sequential
  - Durability
  - Error rates, long-evity
  - ...
  - Very active area of research

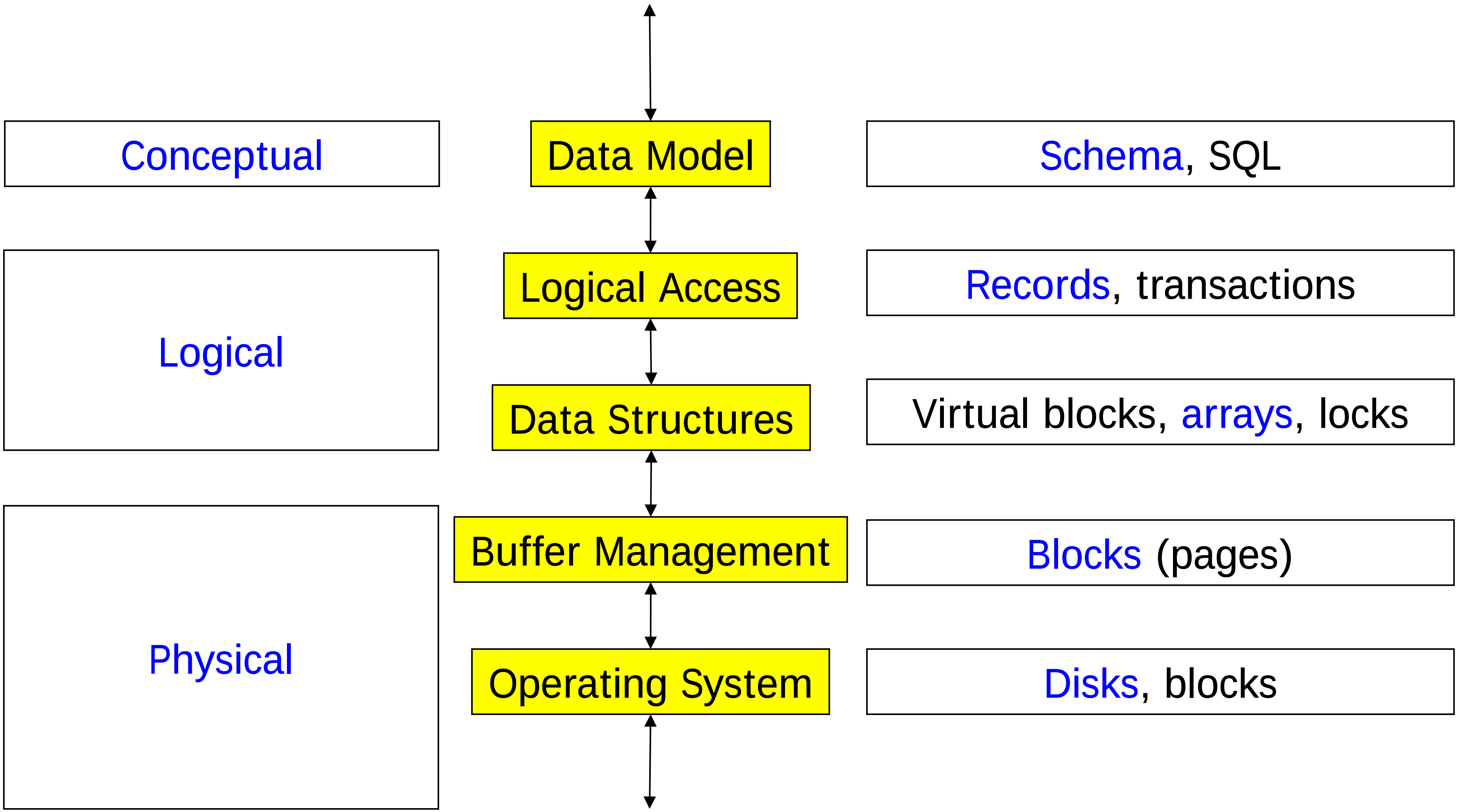
Table of Content

- Storage Hierarchy
- 5-Layer Architecture
- Overview: Layer-by-Layer

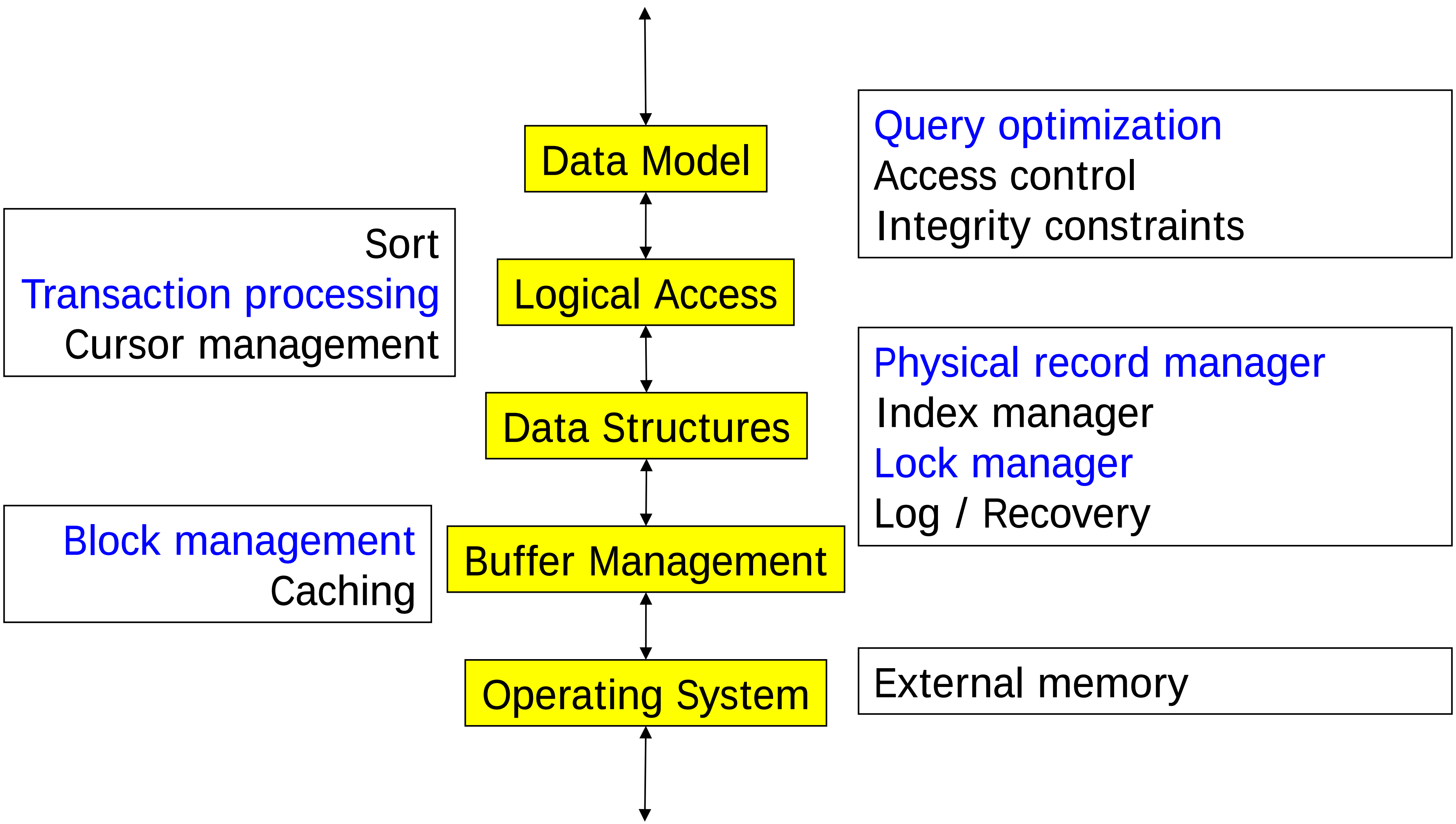
Overview

- Databases are complex software artifacts
- Need to be sliced into layers
- Hardware-induced layers: Memory hierarchy
- Abstraction-induced layers: Tuple – array – byte stream
  - Conceptual – logical - physical
  - Separation of concern
  - Information hiding

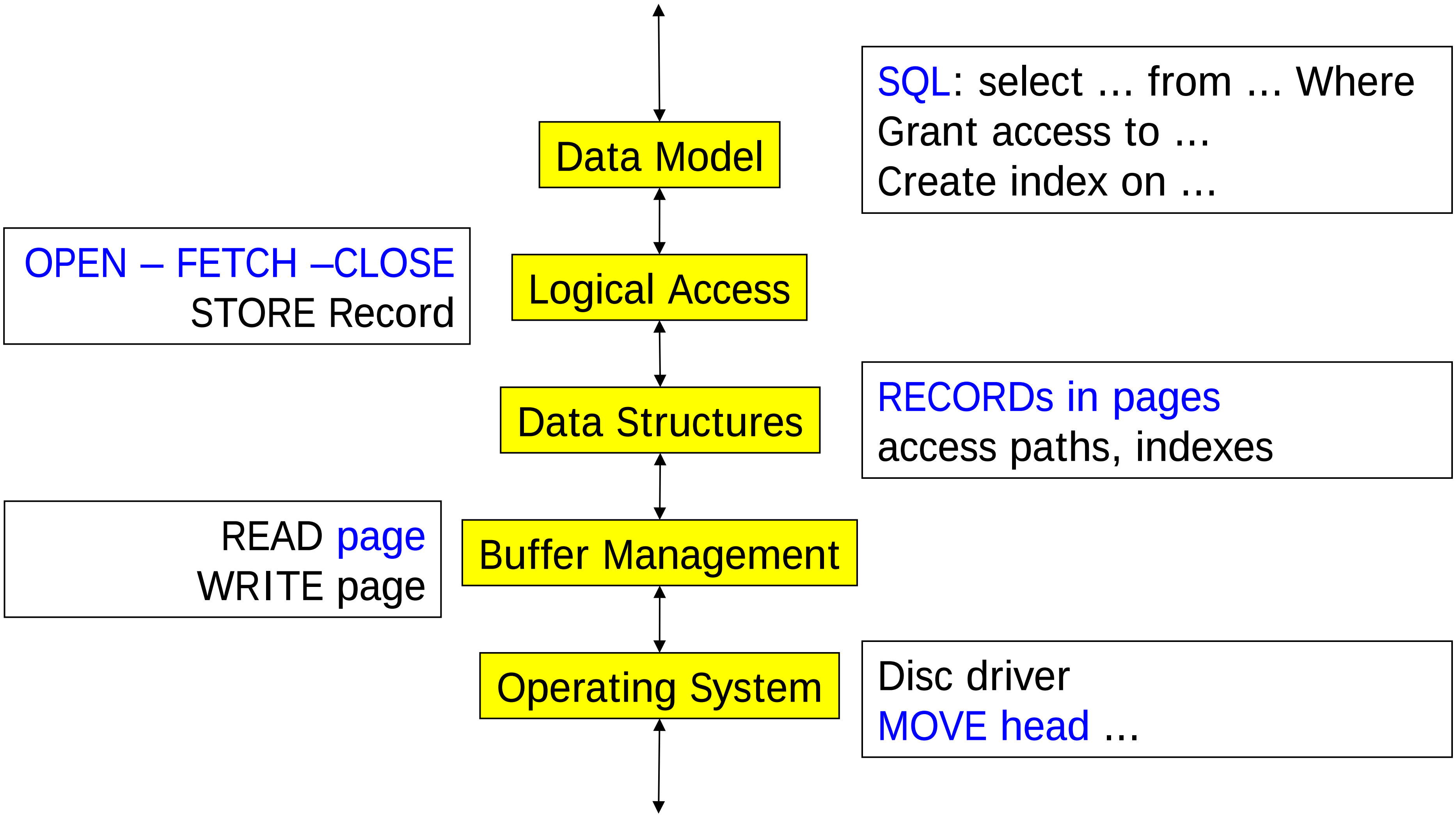
Five Layer Architecture



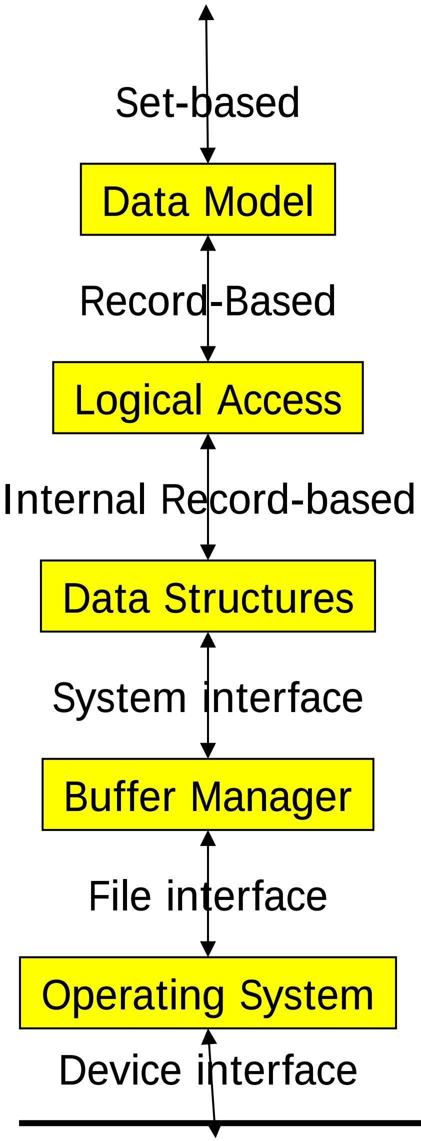
Tasks


Data Model
Logical Access
Data Structures
Buffer Management
Operating System
Sort
Transaction processing
Cursor management
Block management
Caching
Query optimization
Access control
Integrity constraints
Physical record manager
Index manager
Lock manager
Log / Recovery
External memory

Operations


SQL: select … from … Where
Grant access to …
Create index on …
Data Model
OPEN – FETCH –CLOSE
STORE Record
Logical Access
Data Structures
RECORDs in pages
access paths, indexes
READ page
WRITE page
Buffer Management
Operating System
Disc driver
MOVE head …

Interfaces



Set-based access using declarative language

Record-based access using logical access path

Record-based access using physical data structures

Byte access in virtual address space

Block access (software RAID)

Disc controller (Caching, Prefetching, Hardware RAID)

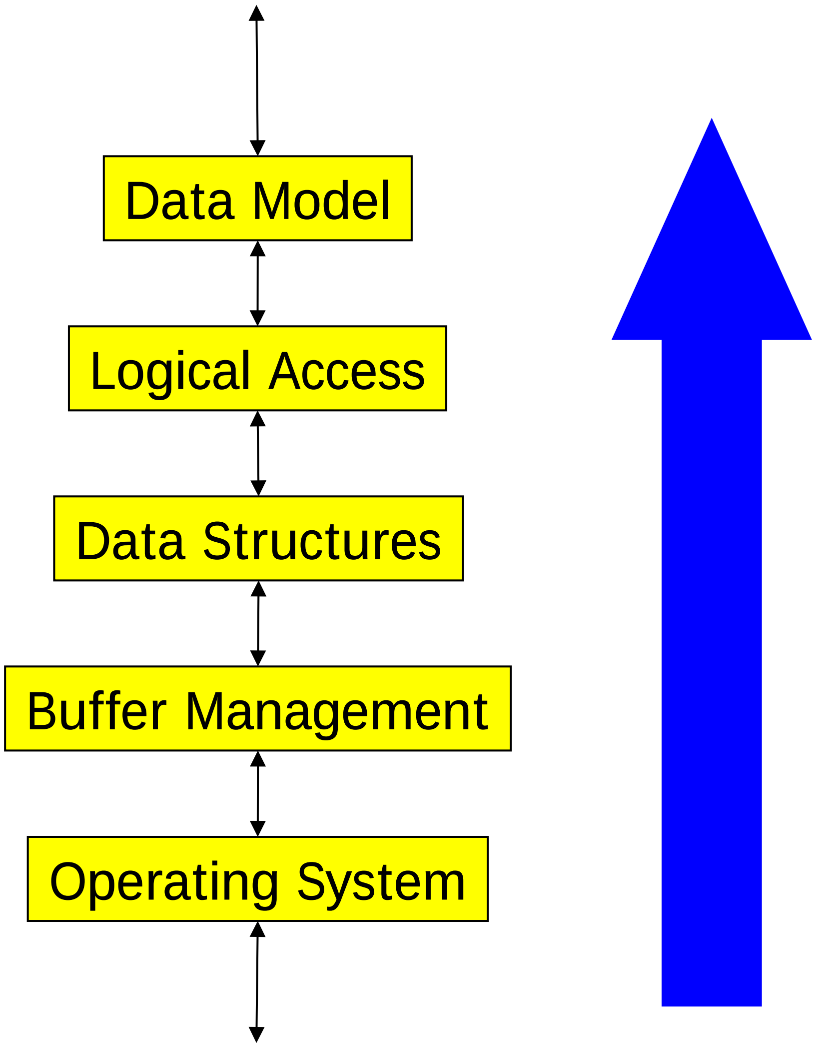
Note: Idealized Representation

- Layers may be merged
  - E.g. logical and internal record-based layers
- Not all functionality can be assigned to exactly one layer
  - E.g. recovery, optimization
- Layers sometimes must access non-neighboring layers
  - Prefetching needs to know the query
    - Layer 4 to Layer 1/2
  - Optimizer needs to know about physical data layout
    - Layer 1 to layer 4/5
  - Breaks information hiding principle

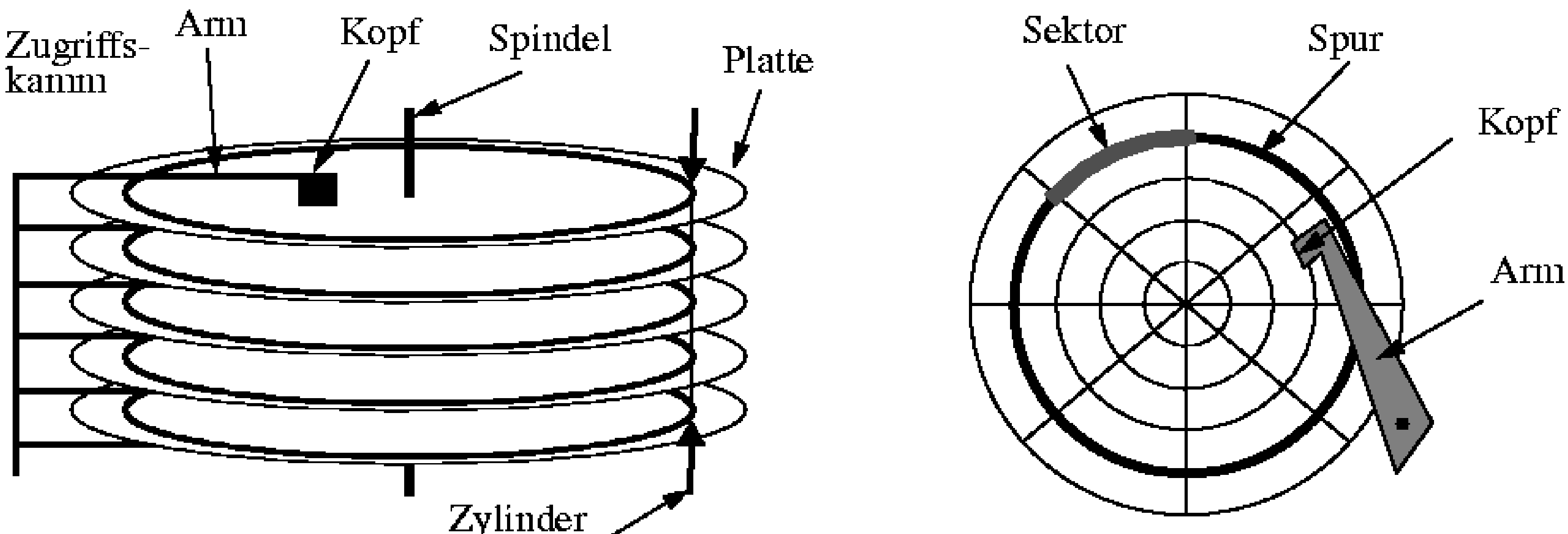
Table of Content

- Storage Hierarchy
- 5-Layer Architecture
- Overview: Layer-by-Layer

Bottom-Up

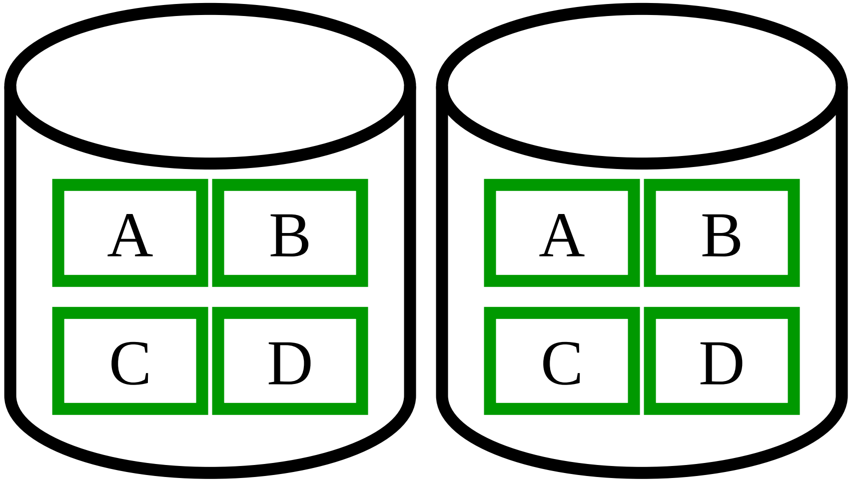

Data Model
Logical Access
Data Structures
Buffer Management
Operating System

Classical Discs



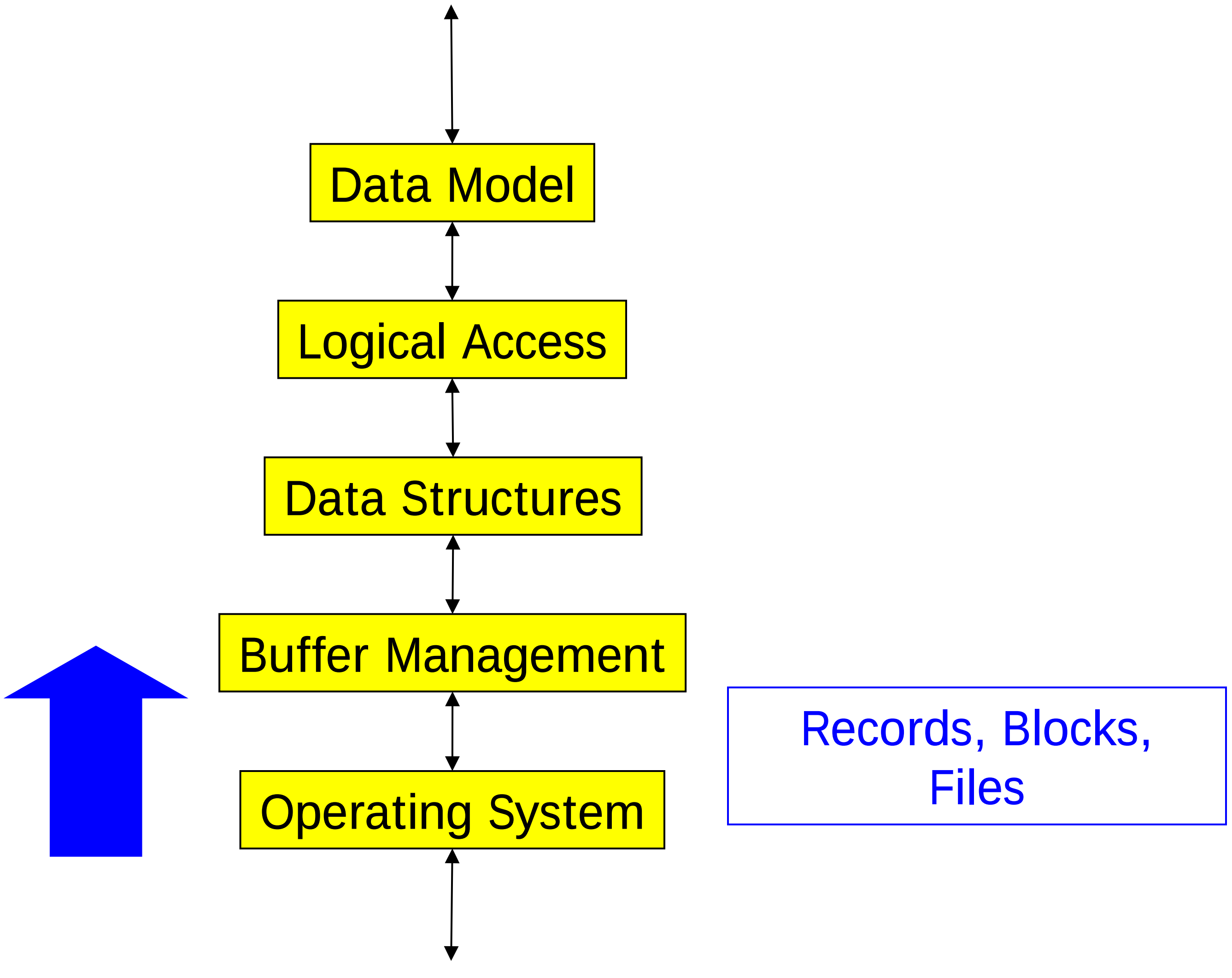
a) seitliche Ansicht **b) Draufsicht**

RAID 1: Mirroring



- Redundancy: Fail-safety and access speed
  - Increased read performance, write perf. not affected (parallel write)
  - Disc crash (one) can be tolerated
  - Be careful about dependent components (controller, power, ...)
- Drawbacks
  - Which value is correct in case of divergence in the two copies?
  - Space consumption doubles

Bottom-Up


Data Model
Logical Access
Data Structures
Buffer Management
Operating System
Records, Blocks,
Files

Access Methods: Sequential Unsorted Files

- Access to records by record/tuple identifier (RID or TID)

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

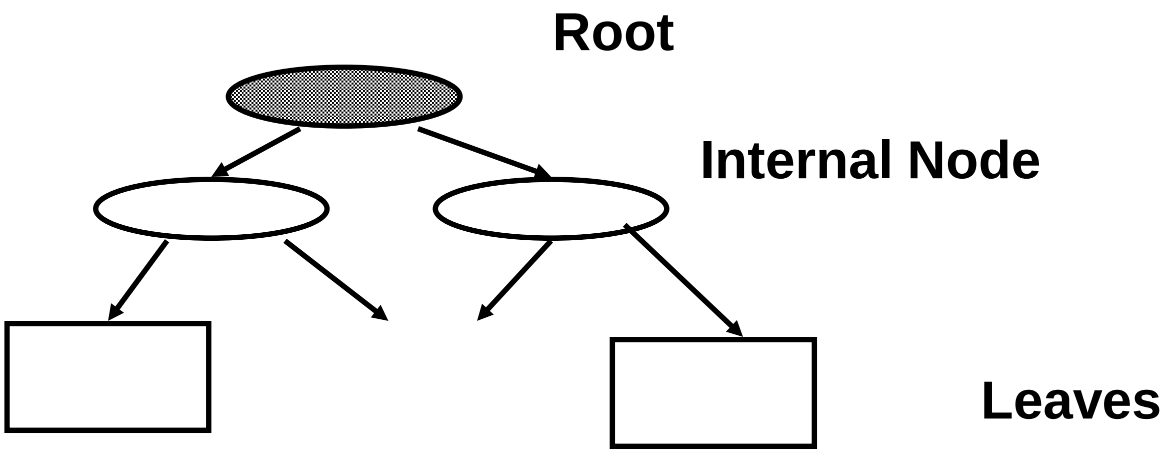
- Operations
  - INSERT(Record): Move to end of file and add, O(1)
  - SEEK(TID): Sequential scan, O(n)
    - FIRST (File): O(1)
    - NEXT(File): O(1)
    - EOF (File): O(1)
  - DELETE(TID): Seek TID; flag as deleted, O(n)
  - REPLACE(TID, Record): Seek TID; write record, O(n)
    - What happens if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

Access Methods: Sequential sorted Files

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

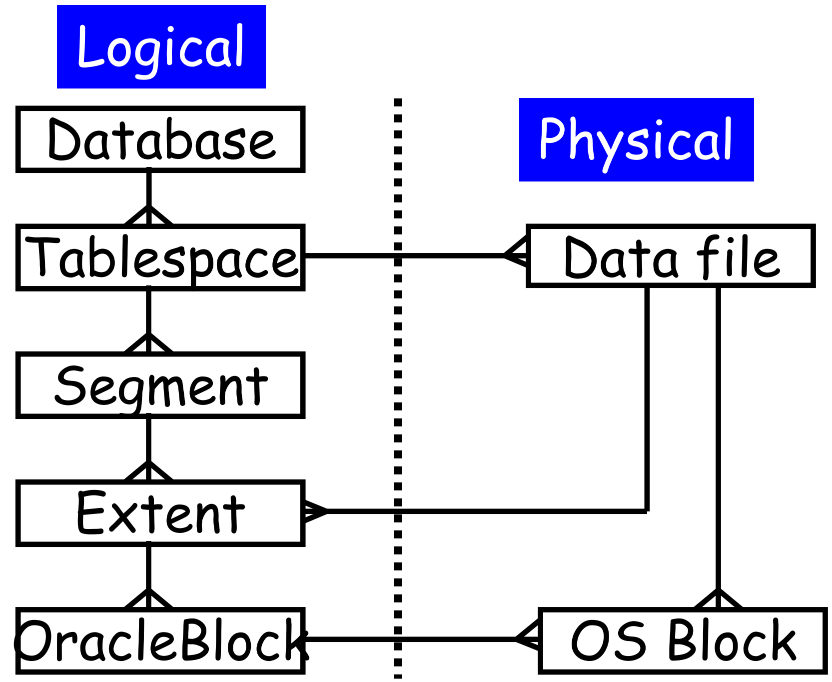
- Operations
  - SEEK(TID): Bin search, O(log(n))
    - But a lot of random access
    - Might be slower than scanning the file
  - INSERT(Record): seek(TID), move subsequent records by one
    - This is terribly expensive – O(n) reads and writes
  - ...

Indexed Files



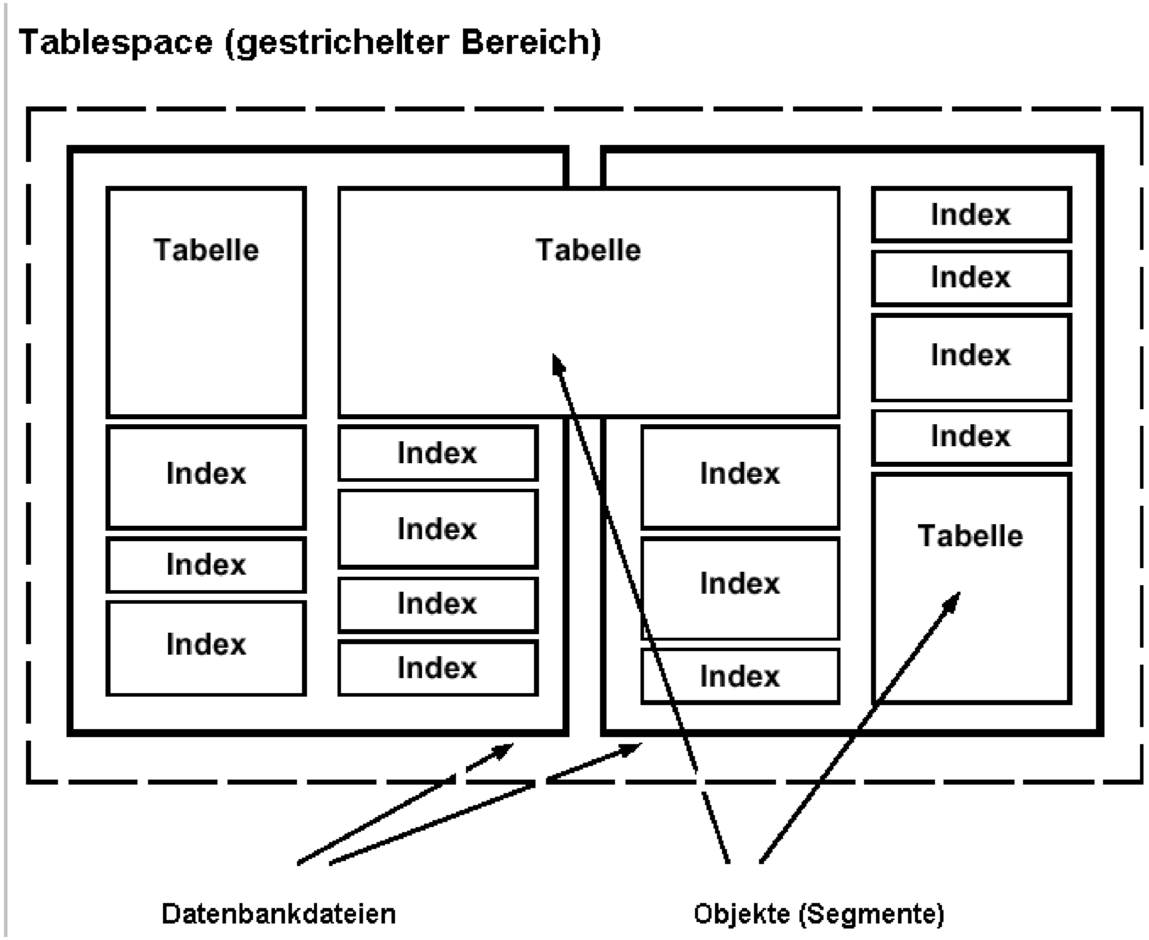
- Operations
  - SEEK(TID): Using order in TIDs; O(log(n))
    - Only if tree is balanced; only if tree is ordered by the right value
  - INSERT(TID): Seek TID and insert; possibly restructuring
  - ...

Storage in Oracle

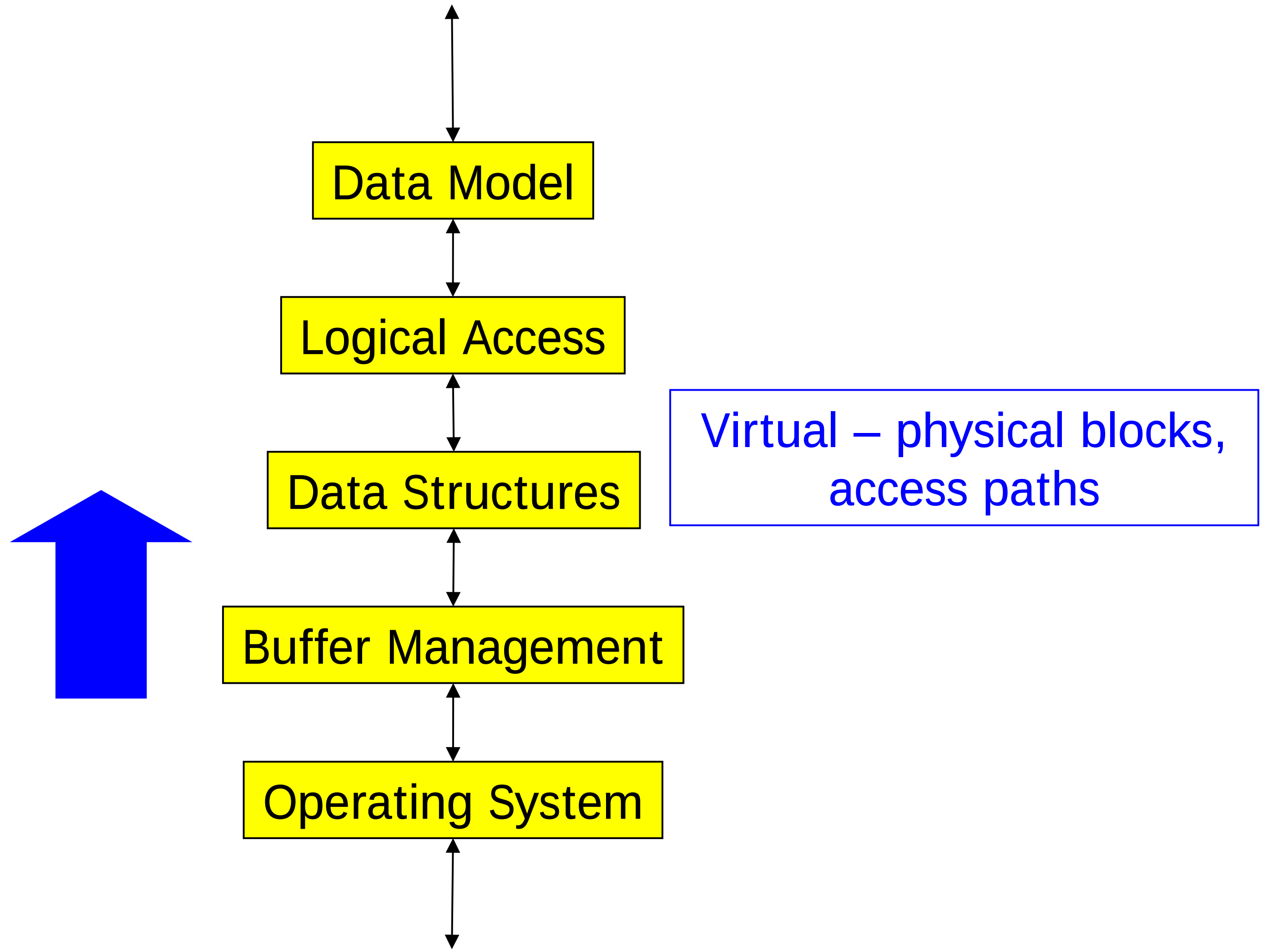


- Data files are assigned to tablespaces
  - May consist of multiple files
  - All data from one object (table, index) are in one tablespace
  - Backup, quotas, access, ...
- Extents: Continuous sequences of blocks on disc
- Space is allocated in extents (min, next, max, ...)
- Segments logically group all extents of an object

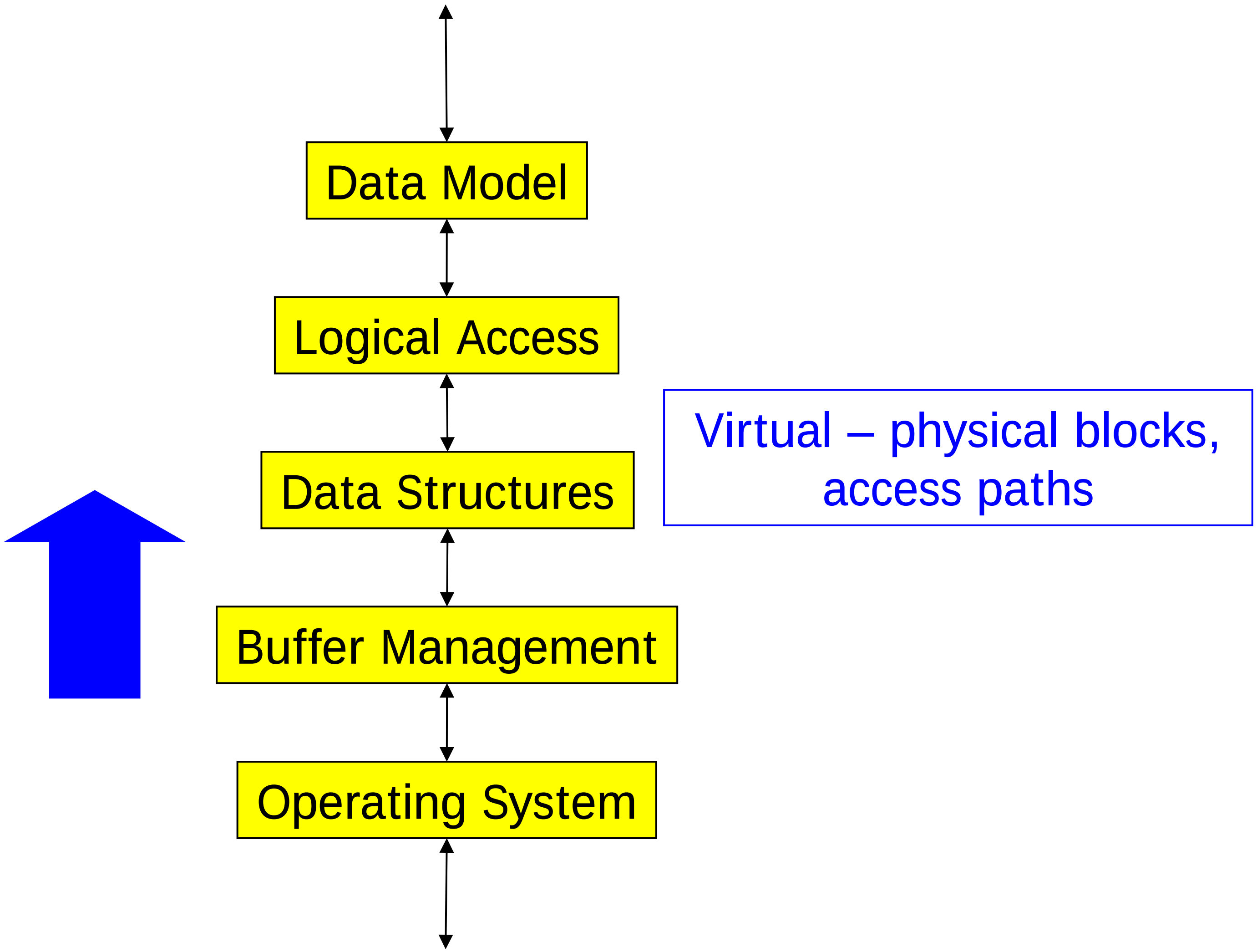
Managing space in Oracle



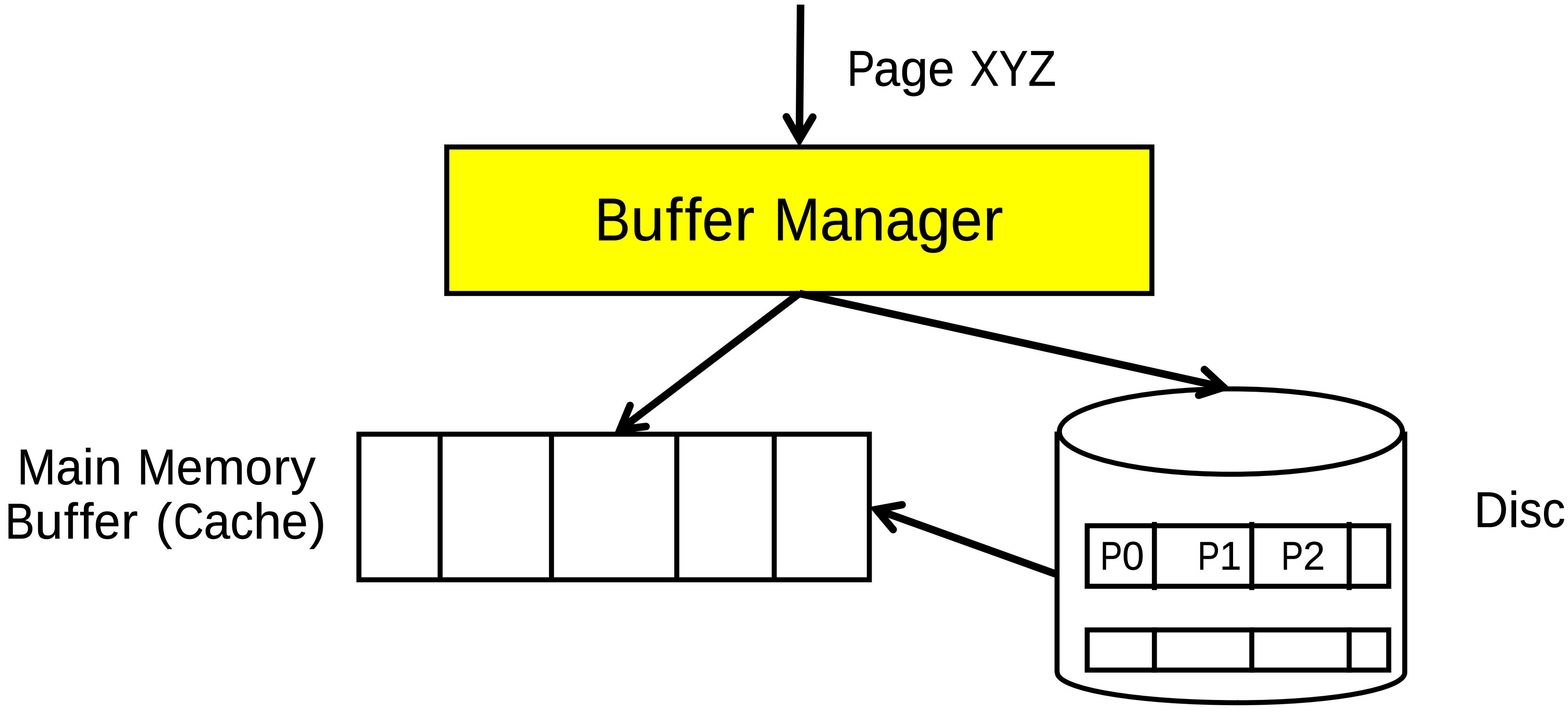
Bottom-Up


Data Model
Logical Access
Virtual – physical blocks,
access paths
Data Structures
Buffer Management
Operating System

Bottom-Up


Data Model
Logical Access
Data Structures
Buffer Management
Operating System
Virtual – physical blocks, access paths

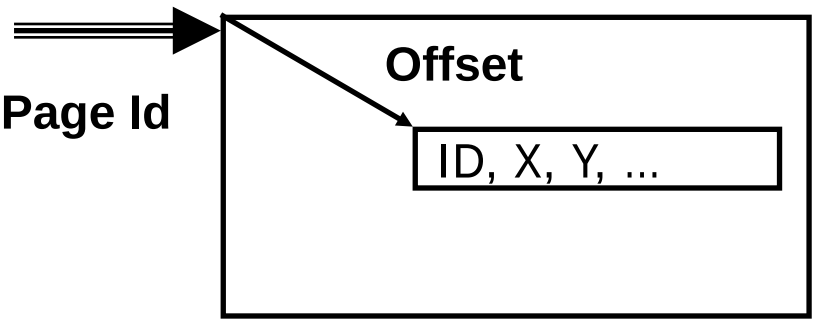
Caching = Buffer Management



- Which blocks should be cached – for how long?
- Caching data blocks? Index blocks?
- Competition: Intermediate data, data buffers, sort buffer, ...

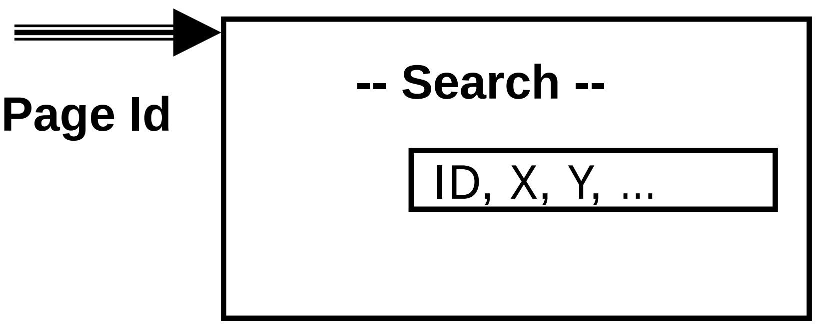
From Buffers to Records

- Absolute addressing: TID = <PageId, Offset, ID>



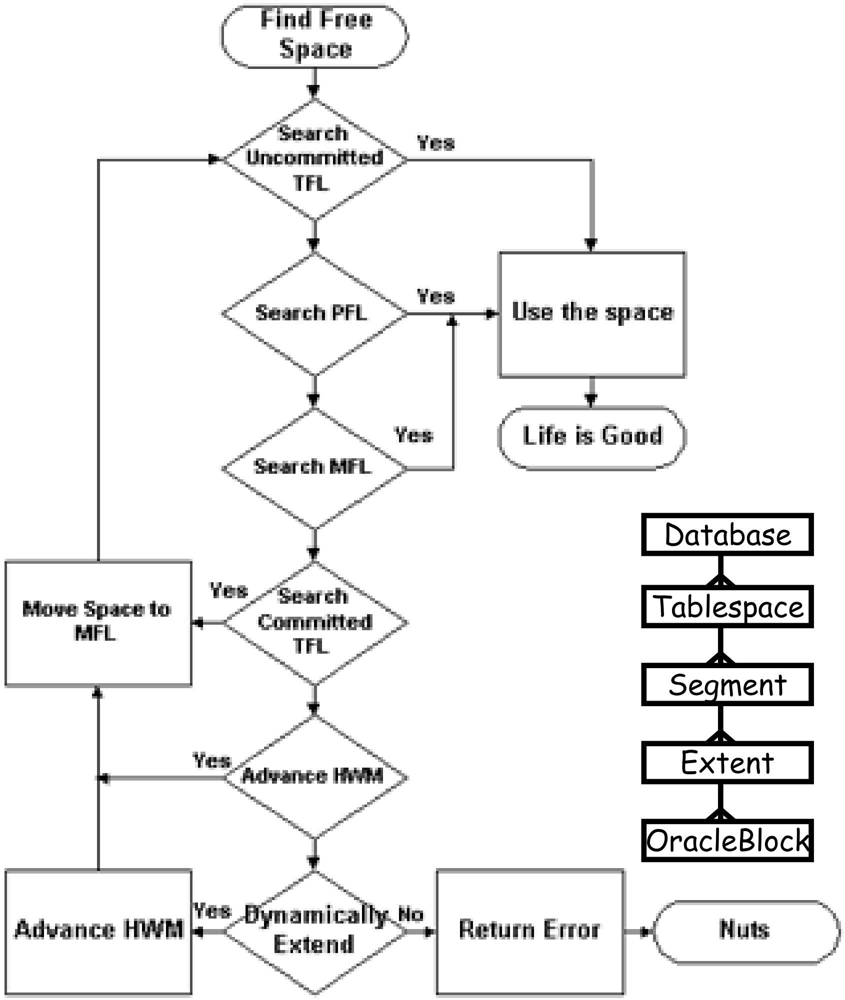
- Pro: Fast access
- Con: Records cannot be moved

- Absolute addressing + search: TID = <PageId,ID>



- Pro: Records can be moved within page
- Con: Slower access

Free Space, TX, and Concurrent Processes



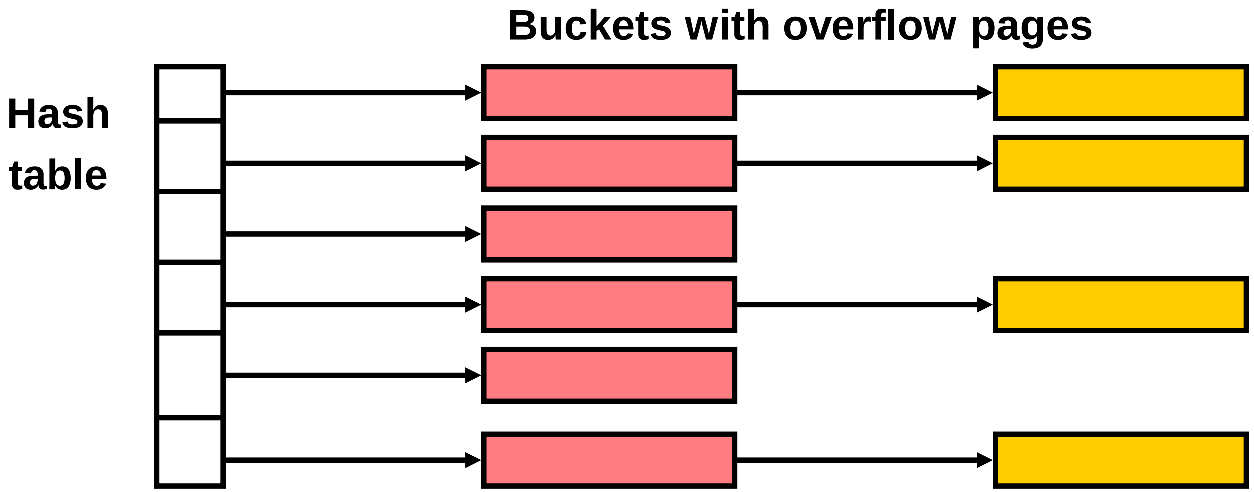
- Oracle procedure for finding free space
- Free space managed at the level of segments
  - Logical database objects
- Explanation
  - TFL: transaction free list
  - PFL: process free list
  - MFL: master free list
  - HWM: High water mark

Records - Blocks

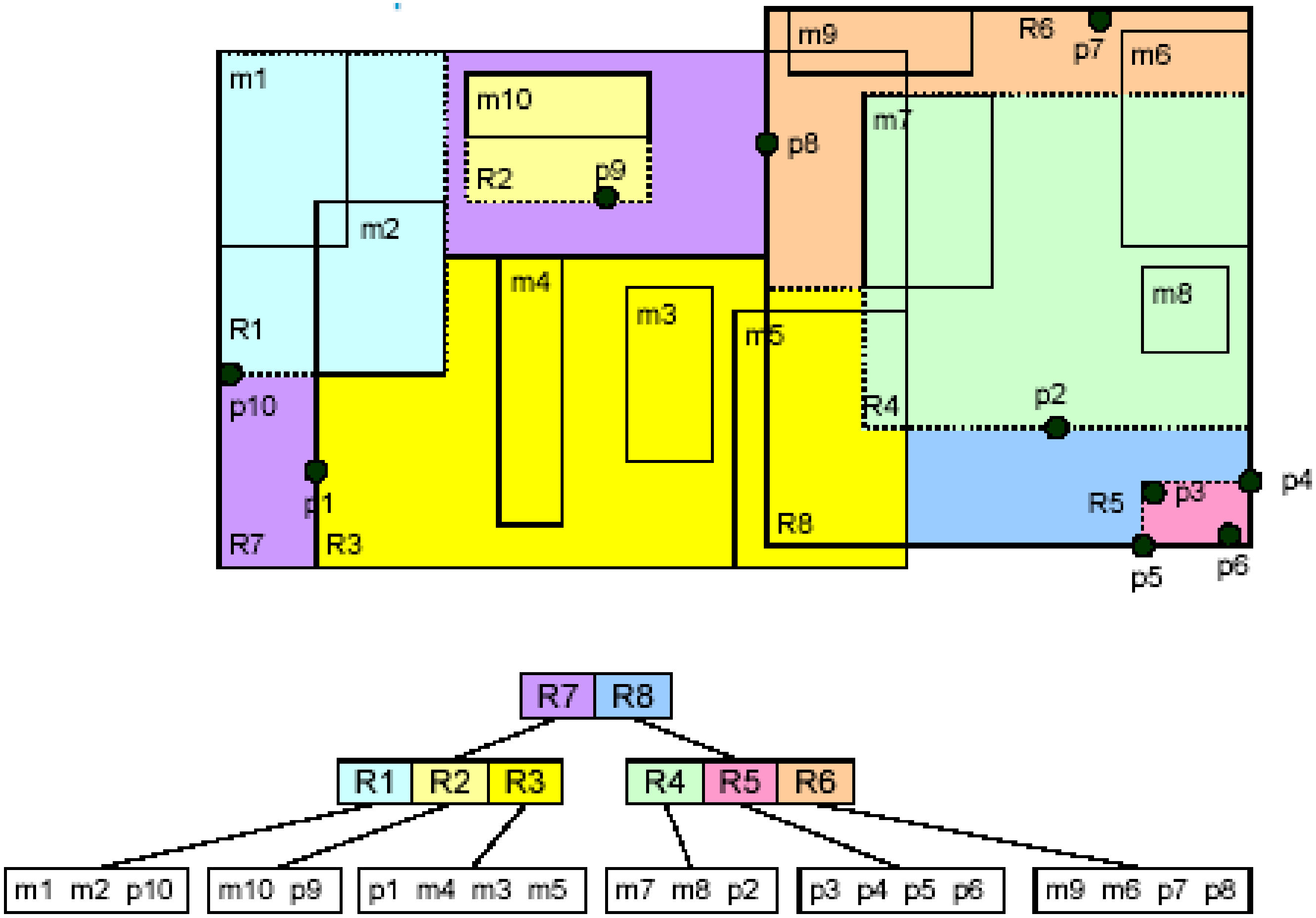
- Records can be placed arbitrarily within blocks
  - TID need to encode the position (block ...)
  - Pro: Flexibility; moving records is comparably simple
  - Con: Finding a record by value requires scanning the entire file
- Record values can determine the block in which they are stored
  - Underspecified: Which value?
  - Pro: Finding a record by the distinguished value is faster
  - Con: Space management becomes much more difficult
    - Almost empty blocks, expensive re-organizations, ...

Hash-based Files

- Hash file consists of
  - Set of m buckets (one or more blocks)
  - A hash function h(K) = {0 ,…m-1 } on a set K of keys;
  - A hash table (bucket directory) with pointers to buckets
- Pro: Easier to handle than sorted file, faster than raw file
- Contra: Unpredictable performance, one attribute rules

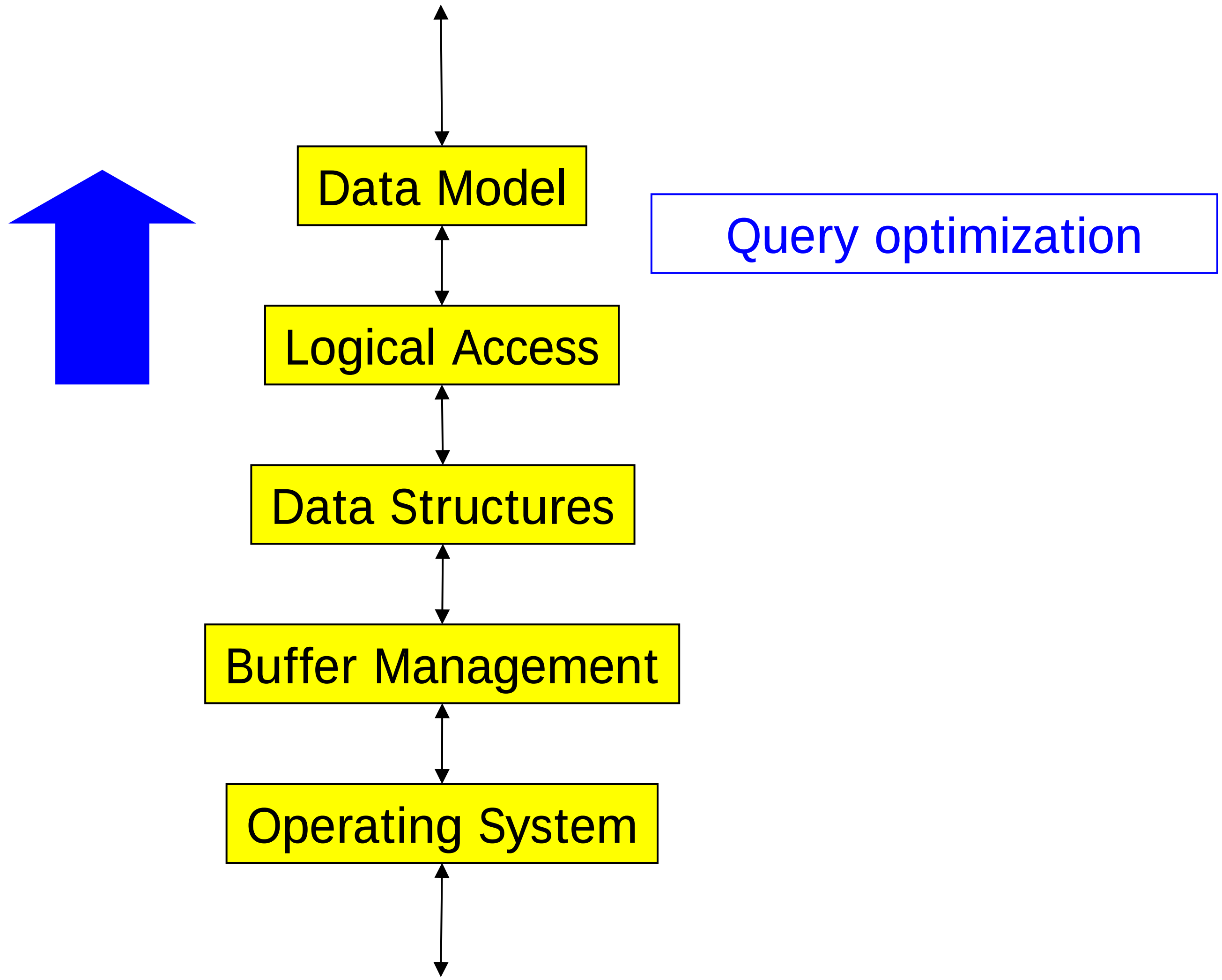


Multidimensional Shapes: R-Trees

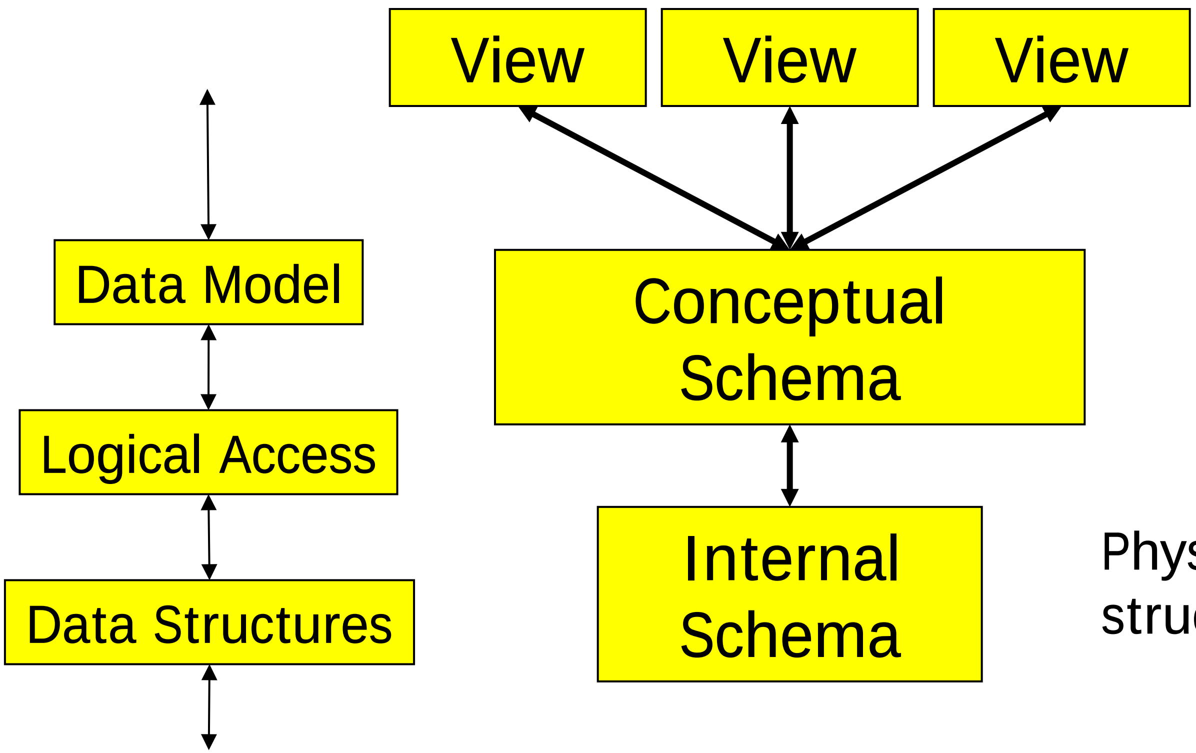


Quelle: Geppert, Data Warehousing, VL SoSe 2002

Bottom-Up


Data Model
Query optimization
Logical Access
Data Structures
Buffer Management
Operating System

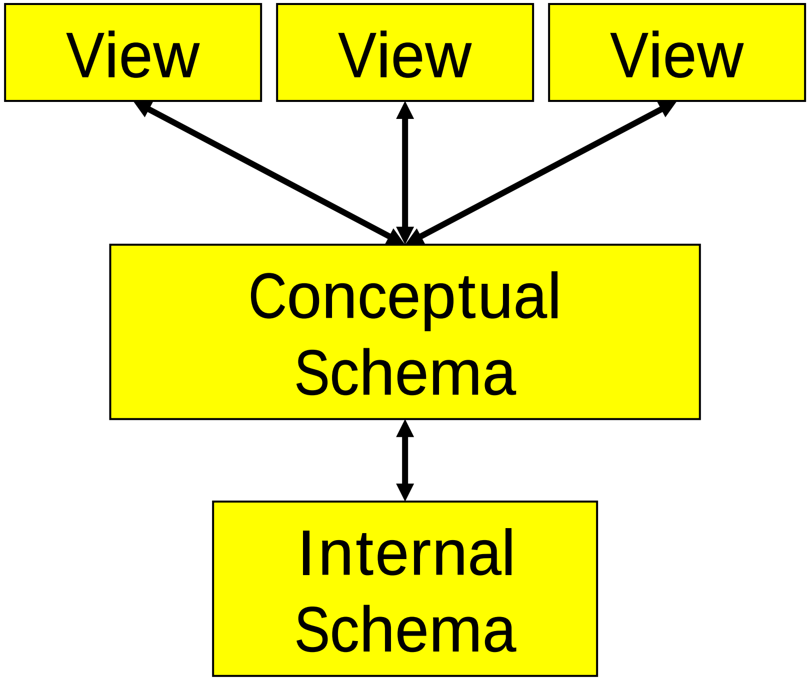
The ANSI/SPARC Three Layer-Model



Logical model: Tables, attributes, constraints, ...

Physical model: Data structures, indexes, ...

Query Execution



Query rewriting, view expansion

Query execution plan generation and optimization: Access paths, join order, ...

Execution of operators, pipelining

Query Processing

- Declarative query

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

- Translated in procedural Query Execution Plan (QEP)

```
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)
```

- Semantically equivalent: Always compute the same result, irrespectively of the DB content

One Query – Many QEPs

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

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

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

…

Query optimization

- Task: Find the (hopefully) fastest QEP
- Two interdependent levels: Best plan, best impl.
  - Different QEPs by algebraic rewriting
    - P1: $\sigma_{Name=Bond}(Account \bowtie Customer)$
    - P2: $Account \bowtie \sigma_{Name=Bond}(Customer)$
  - Different QEPs by different operator implementations
    - P1': Access by scan, hash-join
    - P1'': Access by index, nested-loop-join
- Plan space: Enumerate and evaluate (some? all?) QEPs
- Optimization goal: Minimize size of intermediate results
  - Might miss optimality in terms of runtime
    - Expansive subplan with sorted result
    - Cheap subplan with unsorted result

Rule-Based Optimizer

- Use rules-of-thumbs
  - Push selections as far as possible
  - Push projections as far as possible
  - Use indexes whenever possible
  - Always prefer sort-merge join
  - Order joins: Tables with more selections first
  - ...
- Does not use information about current size of relations and indexes or distribution of values
- Does not use expected effects of operators in the query (selectivity)

Cost-Based Optimizer

- Use statistics on current state of relations
  - Size, value distribution, fragmentation, cluster factors, ...

```
FOR EACH a in ACCOUNT DO
   FOR EACH  c IN CUSTOMER DO
      IF a.Account# = c.Account# THEN
        IF c.Name = "BOND" THEN ...
```

  - Let selectivity of $\sigma_{Name=Bond}$ be 1%, |Customer|=10.000, |Account|=12.000, Customer:Account is 1:N
  - Performs ...
    - Join: 10.000 * 12.000 = 120M comparisons
    - Produces ~12.000 intermediate result tuples
    - Filters down to ~120 results

Cost-Based Optimizer

- Use statistics on current state of relations
  - Size, value distribution, fragmentation, cluster factors, ...

```
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)
```

  - Same setting
  - Performs
    - Reads some index blocks to find 100 customers
      - But these are read using random access
    - Join: 100*12.000= 1.2M comparisons
    - Produces 120 results

Join methods

- Suppose the previous query would contain no selection
- Can‘t we do better than “Join: 120M comparisons”
- Join methods
  - Nested loop join: O(m*n) key comparisons
  - Sort-merge join
    - First sort relations in O(n*log(n)+m*log(m))
    - Merge results in O(m+n)
    - Sometimes better, sometimes worse
  - Hash join, index-join, grace-join, zig-zag join, ...
- Note: Complexity here measures number of comparisons
  - This is a “main-memory” viewpoint
  - Must not be used for IO tasks

Data Dictionary

- Query execution needs metadata: Data dictionary
  - Semantic parsing of query: Which relations exist?
  - Which indexes exists?
  - Cardinality estimates of relations?
  - Size of buffer for in-memory sorting?
  - ...

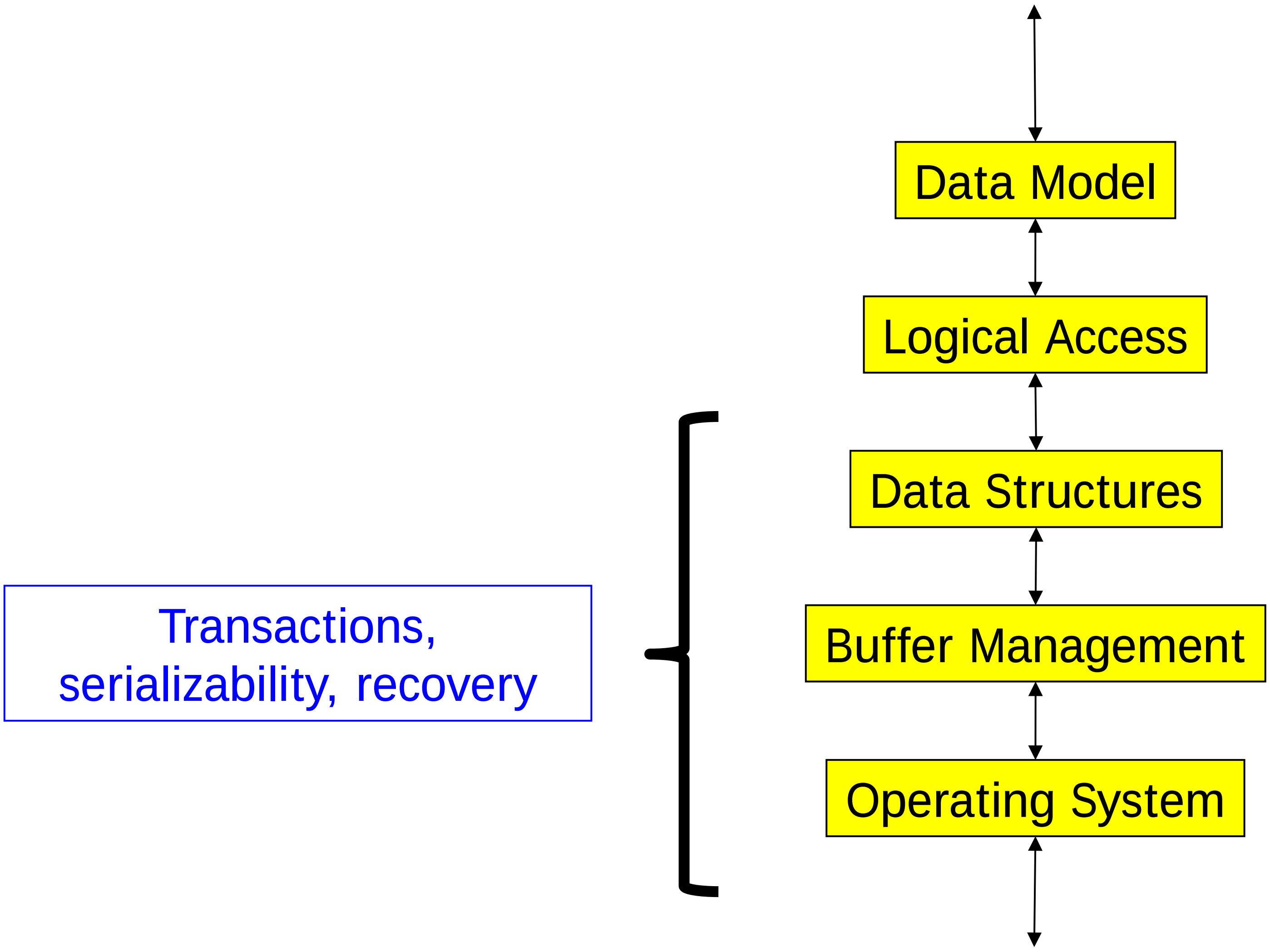
Table_name	**Att_name**	**Att_type**	**size**	**Avg_size**
Customer	Name	Varchar2	100	24
Customer	account#	Int	8	8
Customer	...			

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
- No complete protection
  - Granularity of access rights usually relation/attribute – not tuple
    - Use views, label-based access control
  - Access to data without DBMS (at OS level)
  - Complement with file protection, encryption of data

Bottom-Up


Data Model
Logical Access
Data Structures
Buffer Management
Operating System
Transactions, serializability, recovery

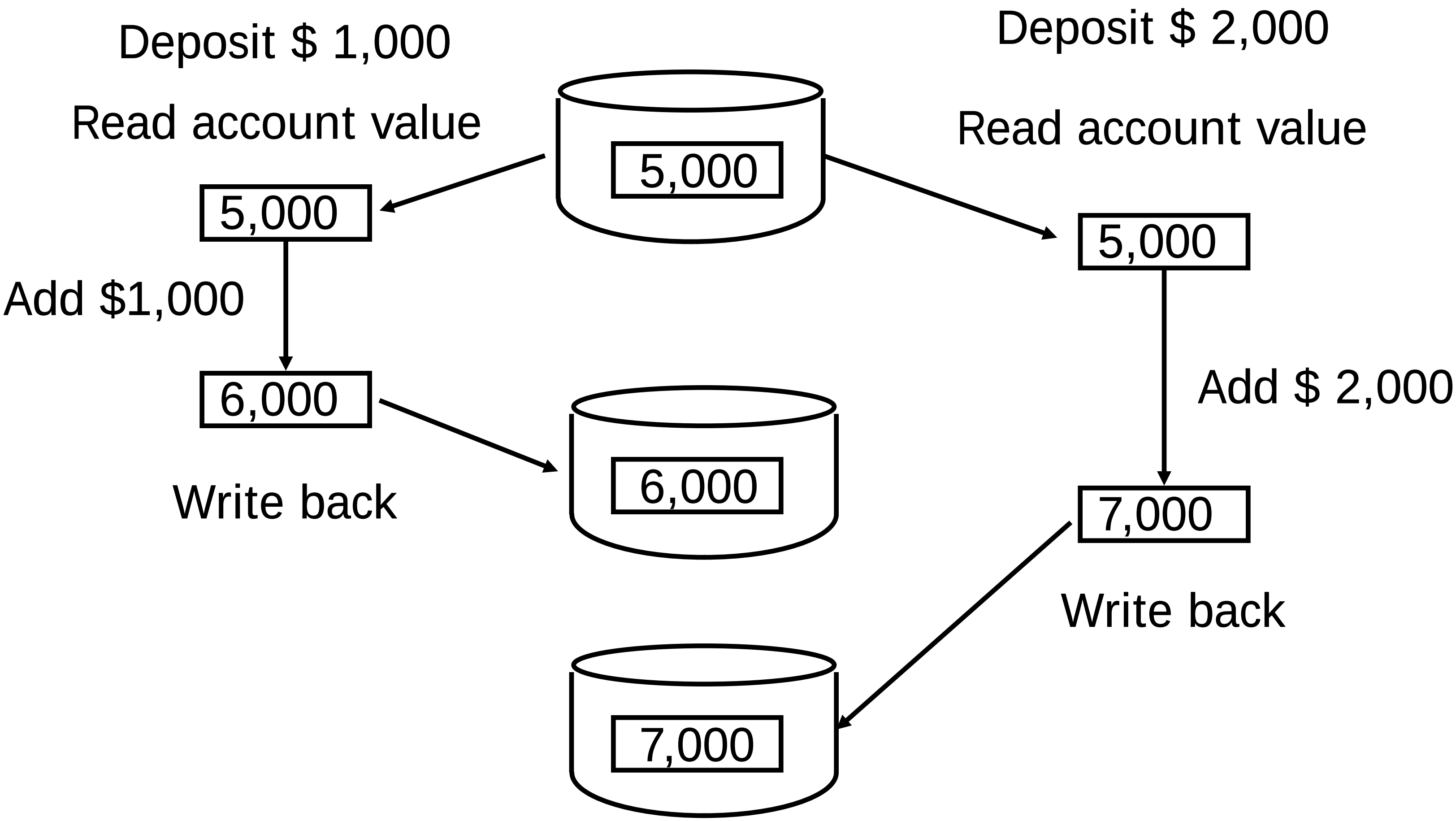
Transactions (TX)

- 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
  - Atomic execution
  - Consistent DB state after commits
  - Isolation: No influence on result by concurrent TX
  - Durability: After commit, changes are reflected in the database

Lost Update Problem



Synchronization and schedules

```
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;
```

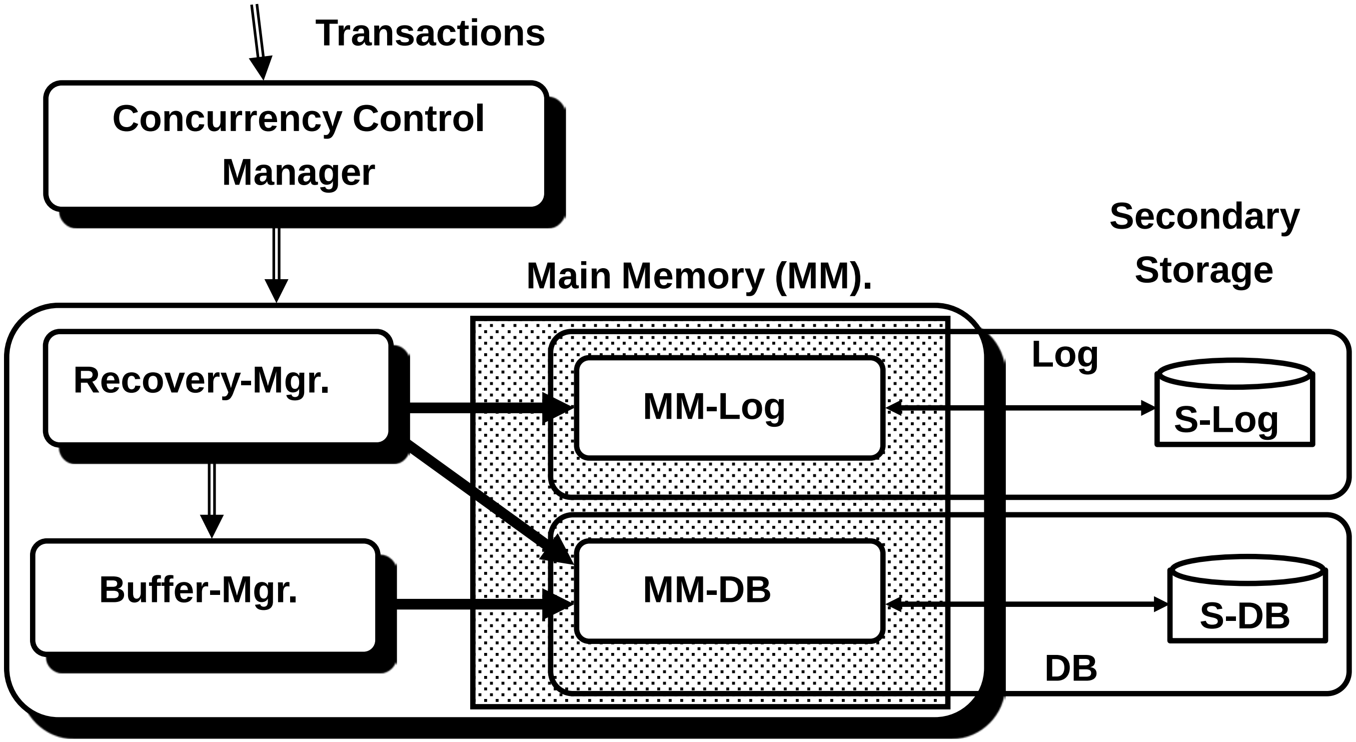
Schedule S_1		Schedule S_2		Schedule S_3	
T_1	T_2	T_1	T_2	T_1	T_2
read A		**read** A		**read** A	
$A - 10$			**read** B	$A - 10$	
write A		$A - 10$			**read** B
read B			$B - 20$	**write** A	
$B + 10$		**write** A			$B - 20$
write B			**write** B	**read** B	
	read B	**read** B			**write** B
	$B - 20$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

?

Synchronization and locks

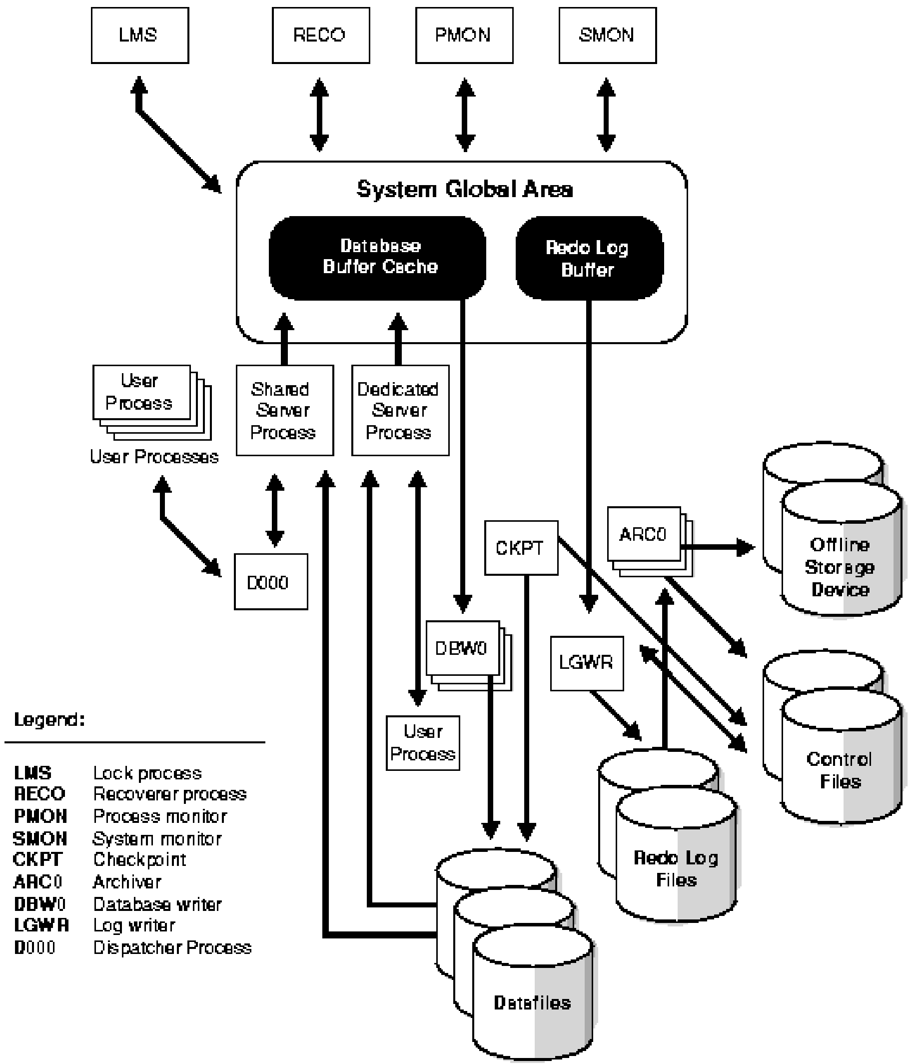
- When is a schedules „fine“?
  - When it is serializable
  - I.e., when it is equivalent to a serial schedule
  - Proof serializability of schedules
- Strategy: Blocking everything is dreadful
- Strategy: Checking after execution is wasteful
- Synchronization protocols
  - Guarantee to produce only serializable schedules
  - Require certain well-behavior of transactions
    - Two phase locking, multi-version synchronization, timestamp synchronization, ...
- Be careful with deadlocks

Recovery – Broad Principle



- Store data redundantly: Save old values
- Different formats for different access characteristics

So many managers …



Oracle processes

- LMS Lock manager (only clustered dbs)
- RECO Recovery of distributed transactions
- PMON Control and restart of all processes
- SMON Recovery at start-up after failure
- CKPT Checkpointing
- ARC0 Archiving of Redo-Log data
- DBW Writing of database blocks
- LGW Writing of Redo-Log blocks
- D Dispatcher für multithreaded servers