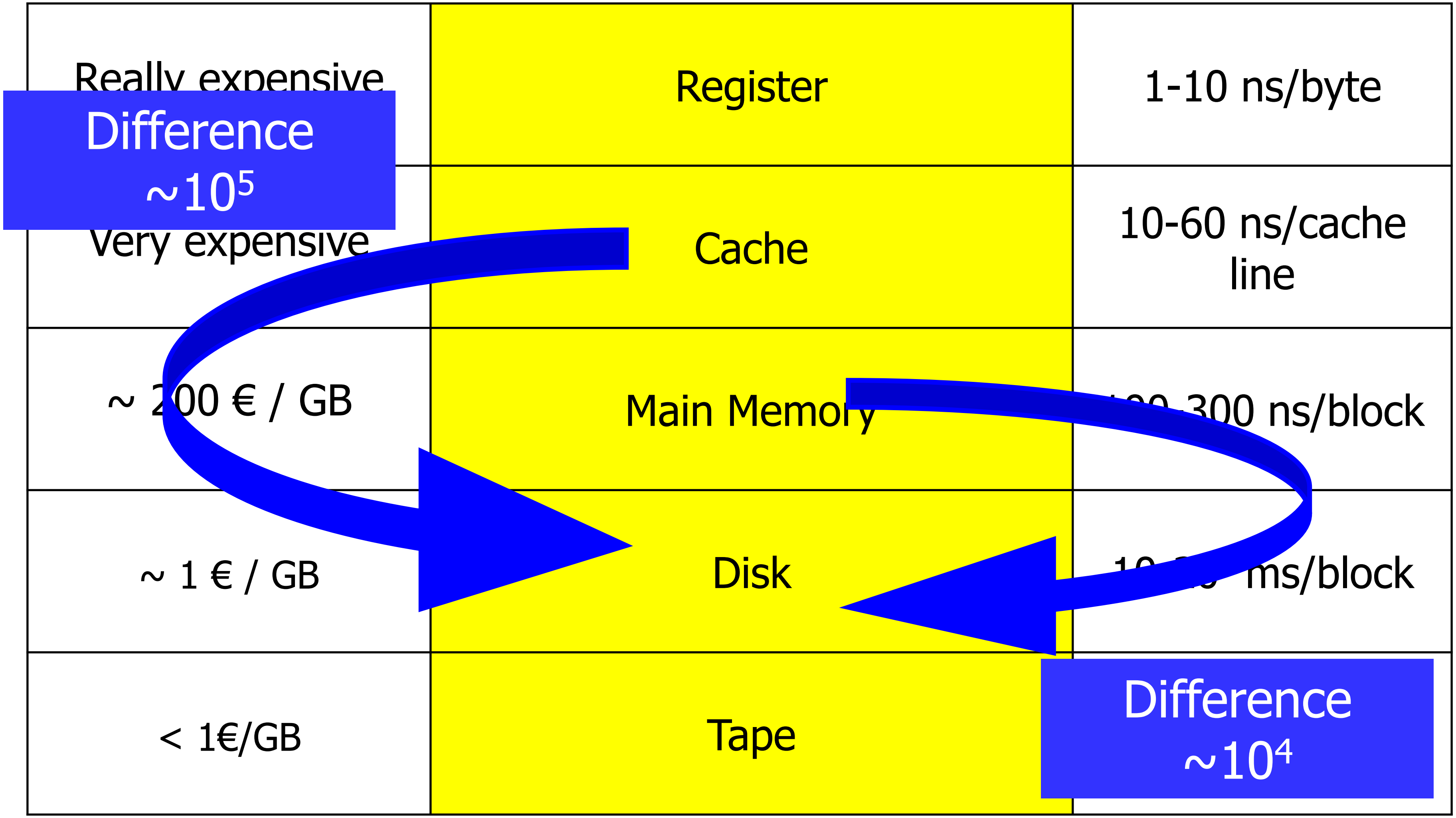
Datenbanksysteme II:
Overview and General Architecture

Ulf Leser

Table of Content

2010: Price versus speed



Really expensive	Register	1-10 ns/byte
Very expensive	Cache	10-60 ns/cache line
~ 200 € / GB	Main Memory	100-300 ns/block
~ 1 € / GB	Disk	10-20 ms/block
< 1€/GB	Tape	

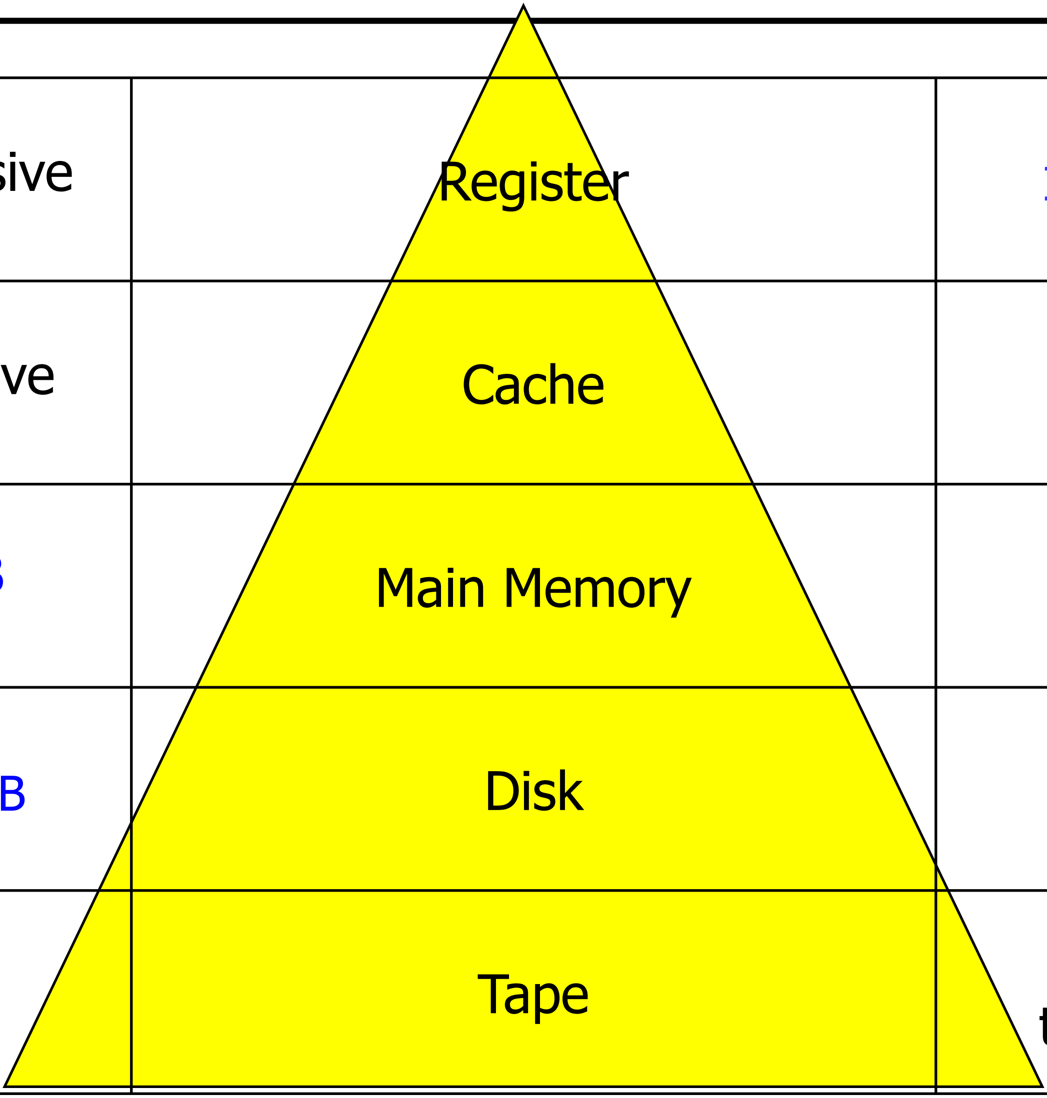
Difference $\sim 10^5$

Difference $\sim 10^4$

2010: Storage Hierarchy

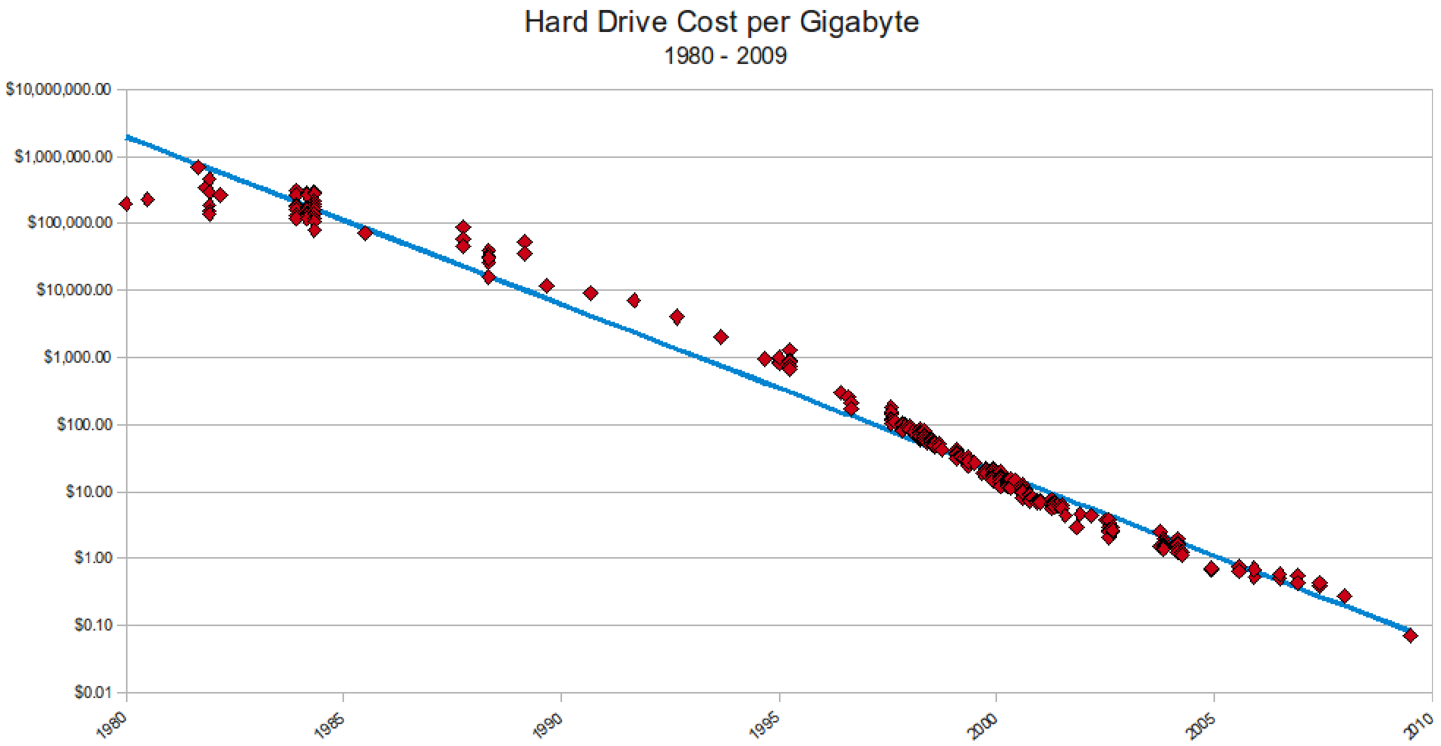
Really expensive	Register	1-4 byte
Very expensive	Cache	1-4 MB
~ 200 € / GB	Main Memory	1-16 GB
~ 1 € / GB	Disk	512GB – 1TB discs
< 1€/GB	Tape	“Infinite” tape robots

2016: Storage Hierarchy



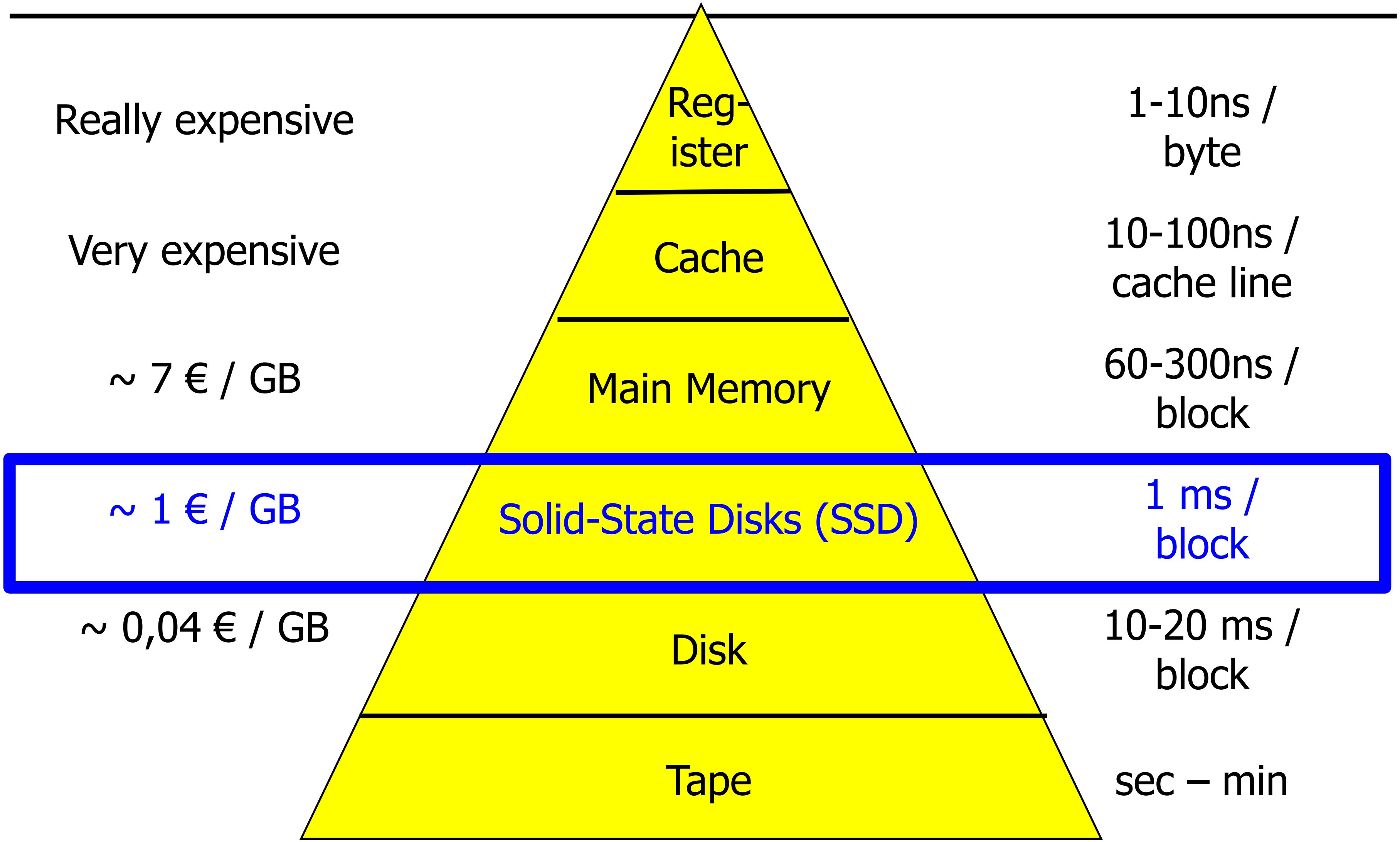
Really expensive	Register	1 – 32 byte
Very expensive	Cache	1-16 MB
~ 7 € / GB	Main Memory	16-256 GB
~ 0,04 € / GB	Disk	1-16 TB
	Tape	“Infinite” tape robots

Costs Drop Faster than you Think



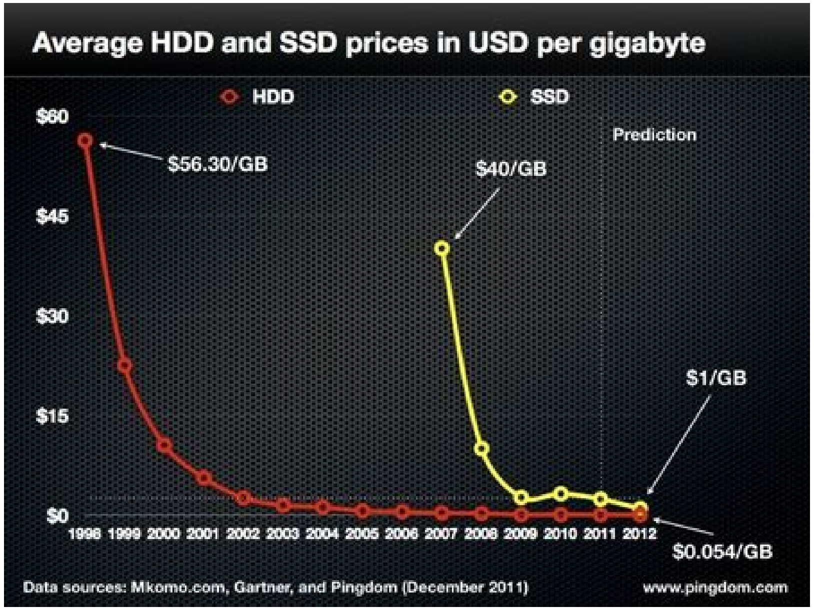
Source: http://analystfundamentals.com/?p=88

New Players



Really expensive	Register	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

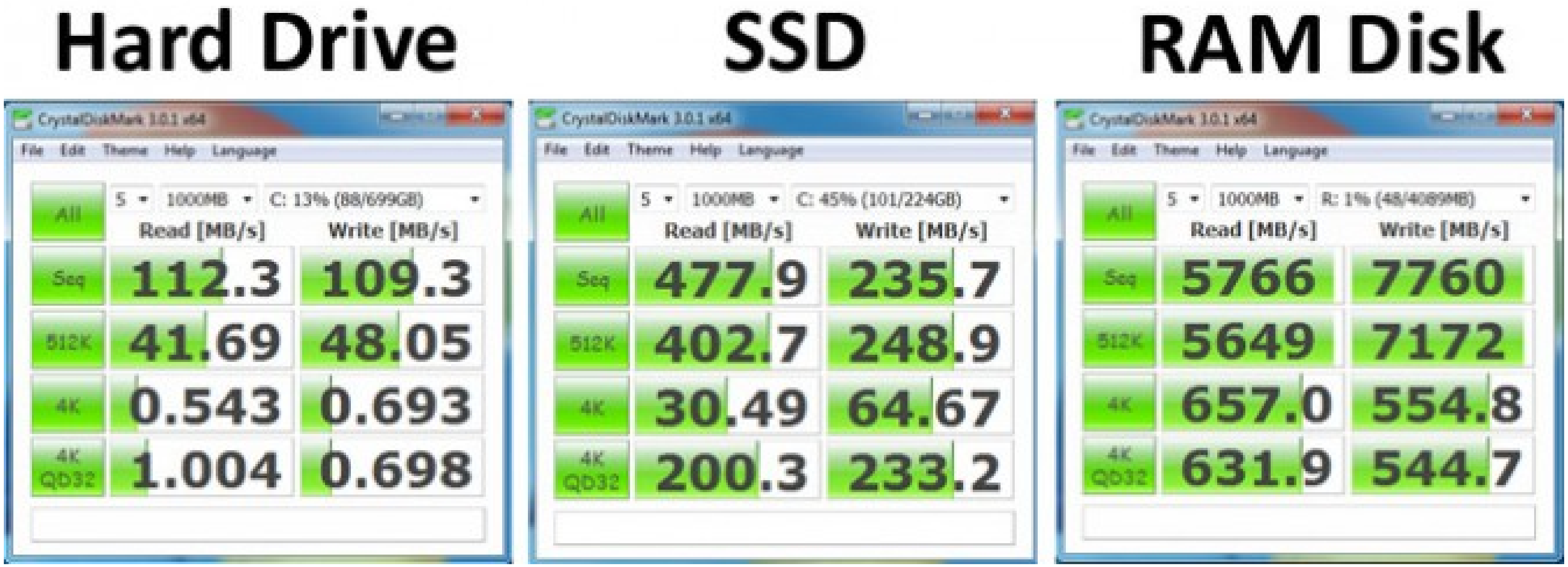
New Players



Source: http://www.tomshardware.com/news/ssd-hdd-solid-state-drive-hard-disk-drive-prices,14336.html
https://www.pcworld.com/article/3011441/

Characteristics

random access != sequential



Hard Drive (5, 1000MB, C: 13% (88/699GB))

	Read [MB/s]	Write [MB/s]
Seq	112.3	109.3
512K	41.69	48.05
4K	0.543	0.693
4K QD32	1.004	0.698

SSD (5, 1000MB, C: 45% (101/224GB))

	Read [MB/s]	Write [MB/s]
Seq	477.9	235.7
512K	402.7	248.9
4K	30.49	64.67
4K QD32	200.3	233.2

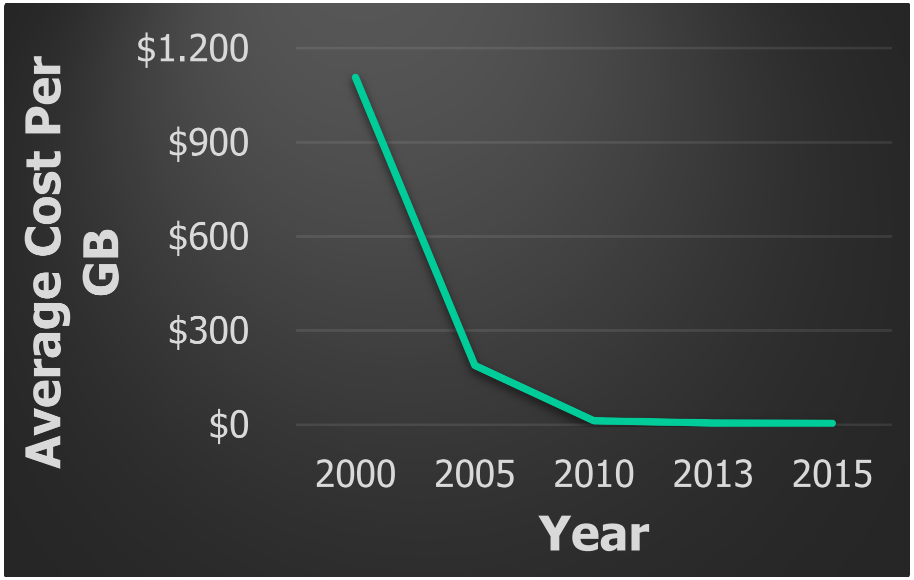
RAM Disk (5, 1000MB, R: 1% (48/4089MB))

	Read [MB/s]	Write [MB/s]
Seq	5766	7760
512K	5649	7172
4K	657.0	554.8
4K QD32	631.9	544.7

read != write

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

Prize of Main Memory



- 2014: 1TB DRAM ~ 5000€
- 2016: Laptops with 16GB, desktops with 32GB, servers with 128GB
- 2019: Mobiles with 32GB, servers with >1TB
- My Guess: 99% of all commercial databases are smaller than 100GB
  - Research: Main memory databases

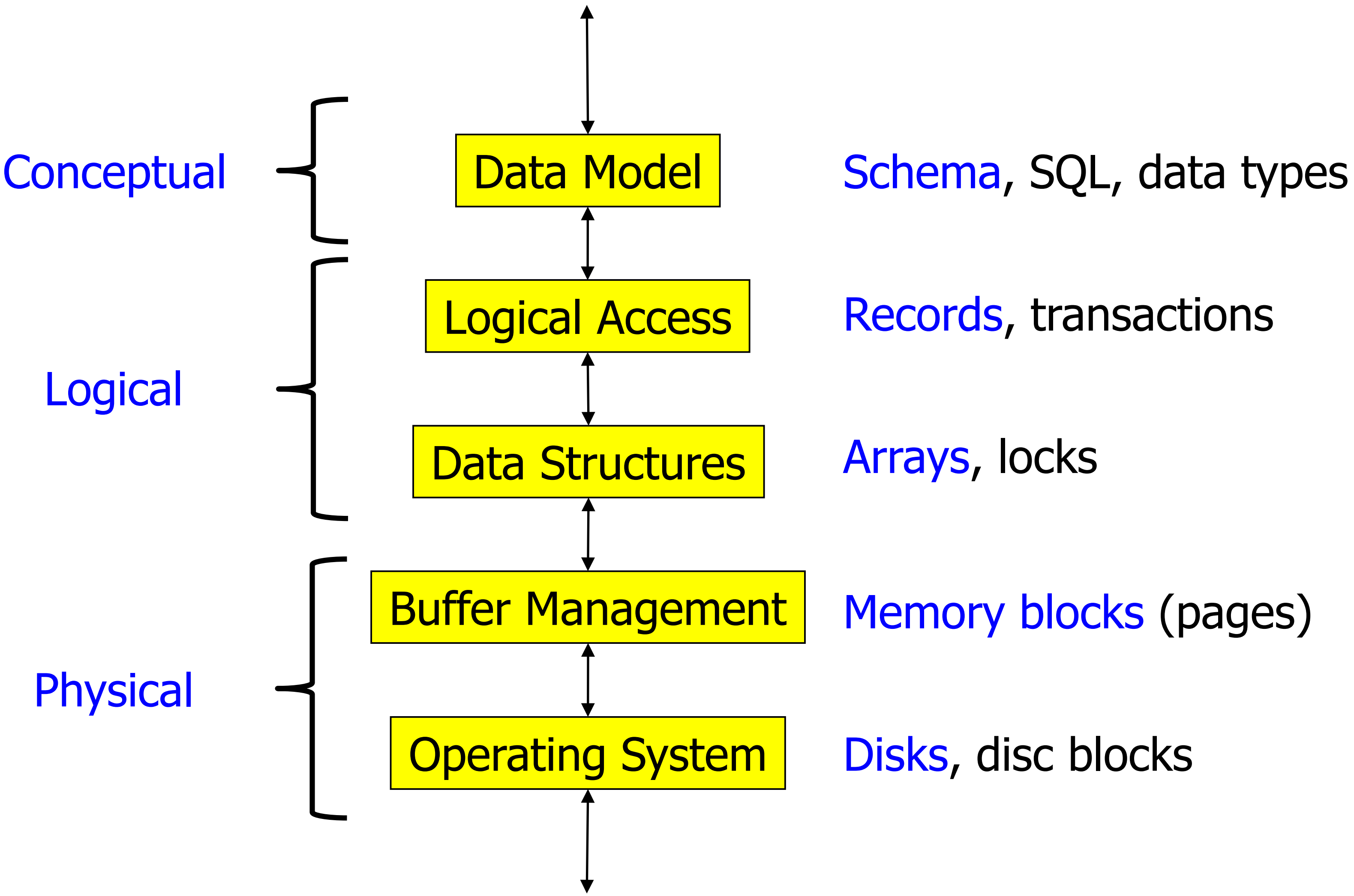
Consequences

- Dealing with memory hierarchy is core concern of DBMS
  - Another issue is multi-core
- This lecture will mostly focus on disk versus RAM
- Similar problems for cache-RAM, disk-SSD, …
- Many differences between storage media
  - Speed, durability, size, cost
  - Block sizes
  - Read/write, random-access/sequential
  - Error rates, longevity
  - …

Table of Content

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

Five Layer Architecture



Conceptual

Data Model — Schema, SQL, data types

Logical

Logical Access — Records, transactions

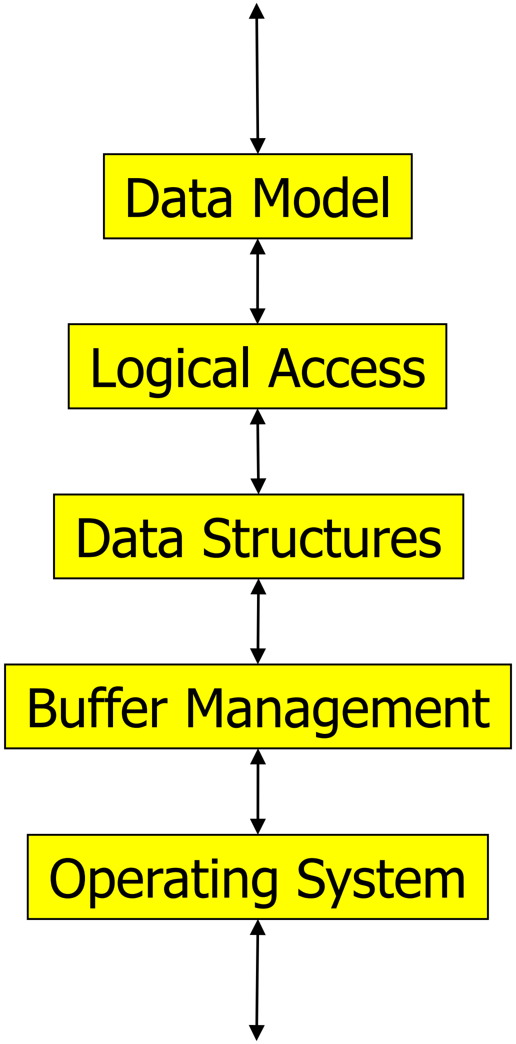
Data Structures — Arrays, locks

Physical

Buffer Management — Memory blocks (pages)

Operating System — Disks, disc blocks

Tasks



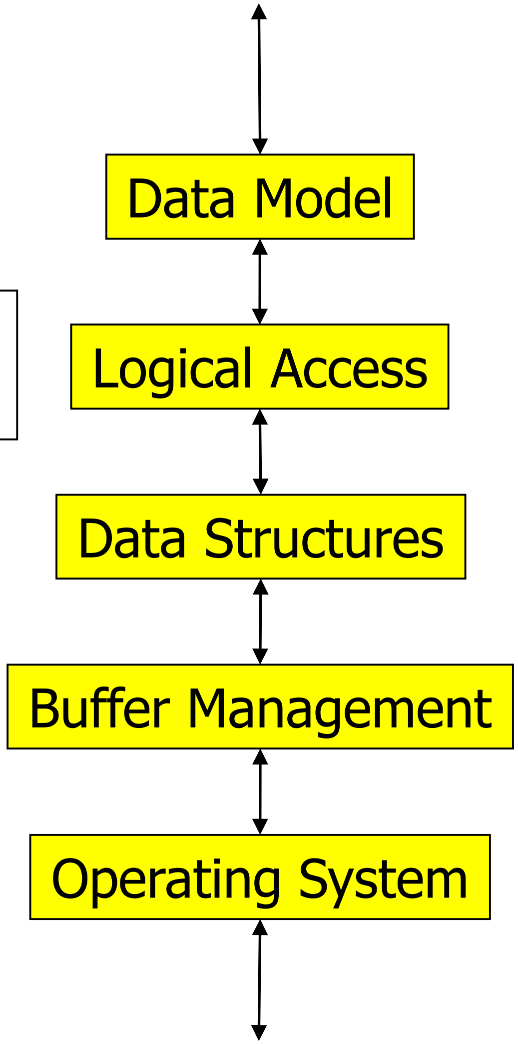
Query optimization
Access control
Integrity constraints

Sort
Transaction processing
Cursor management

Physical record manager
Index manager
Lock manager
Log / Recovery

Block management
Caching

Operations



SQL: select ... from ... Where
Grant access to ...
Create index on ...

Data Model

OPEN – FETCH –CLOSE
STORE Record

Logical Access

RECORDs in pages
access paths, indexes

Data Structures

READ page
WRITE page

Buffer Management

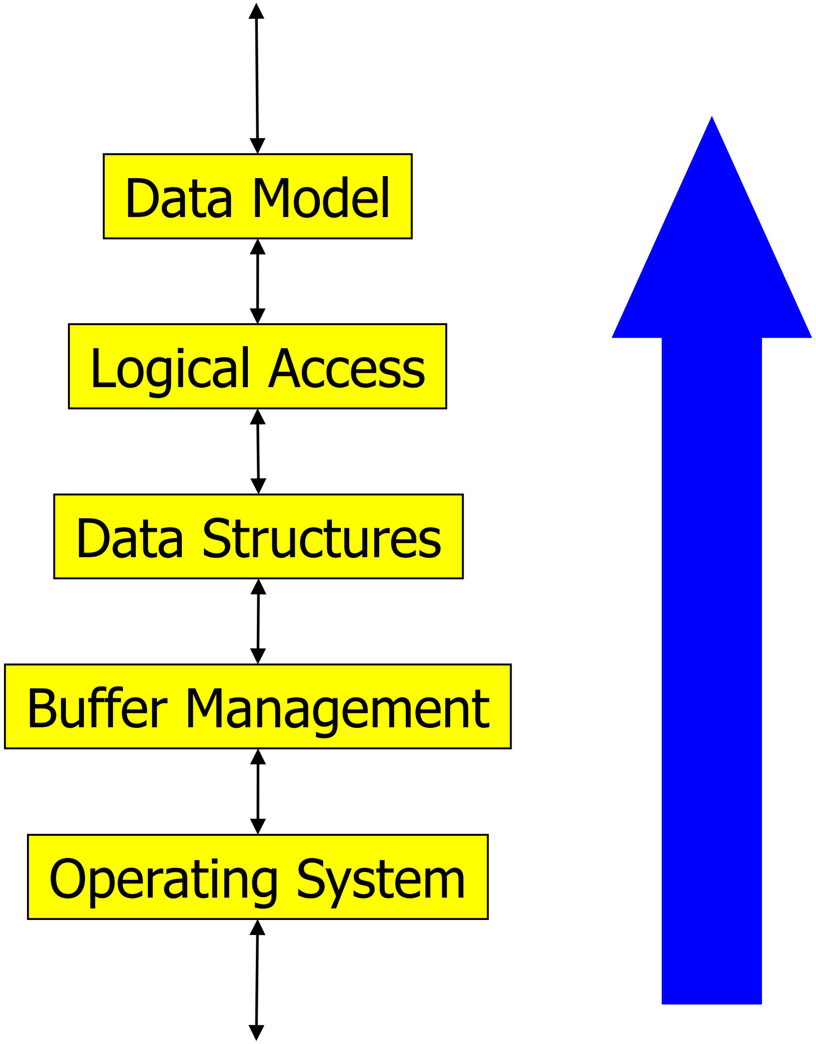
Operating System

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

Bottom-Up



Data Model

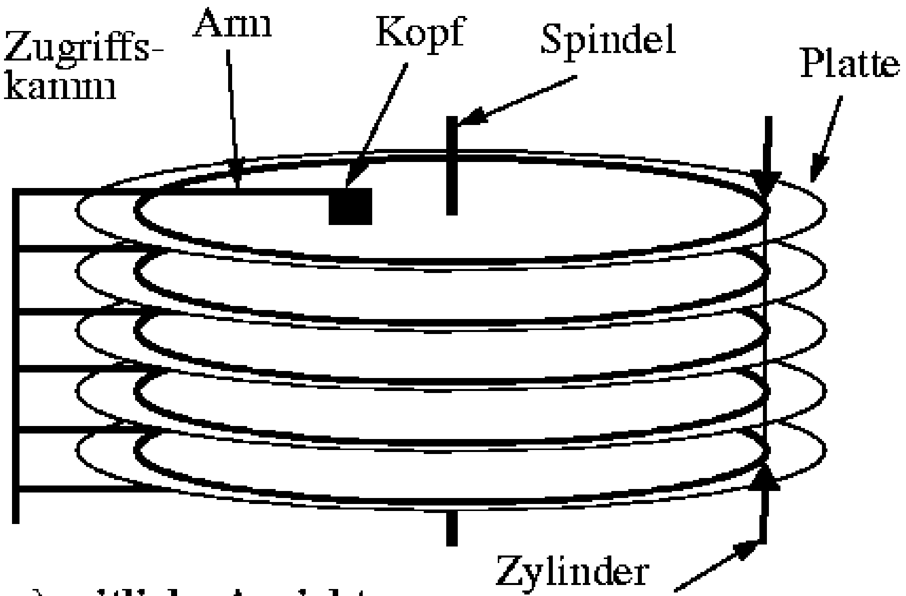
Logical Access

Data Structures

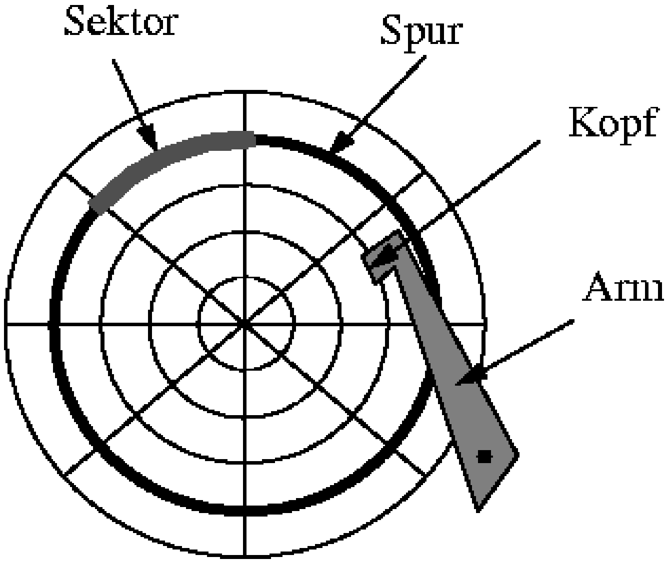
Buffer Management

Operating System

Classical Discs



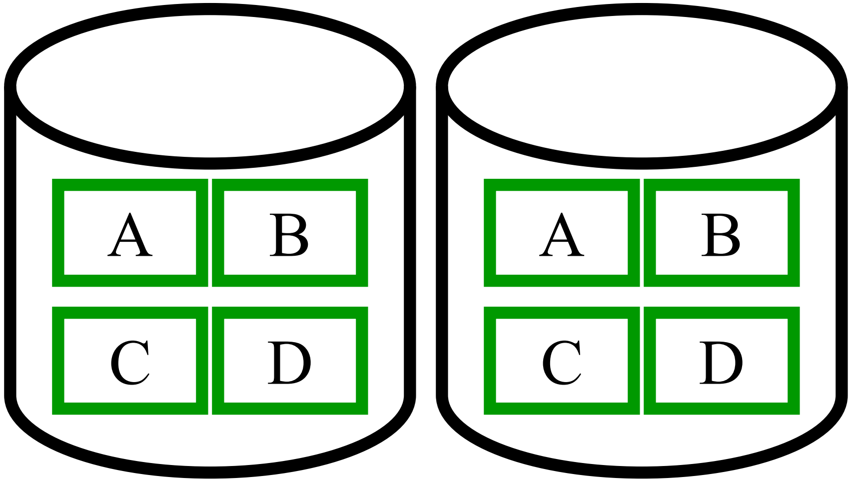
a) seitliche Ansicht



b) Draufsicht

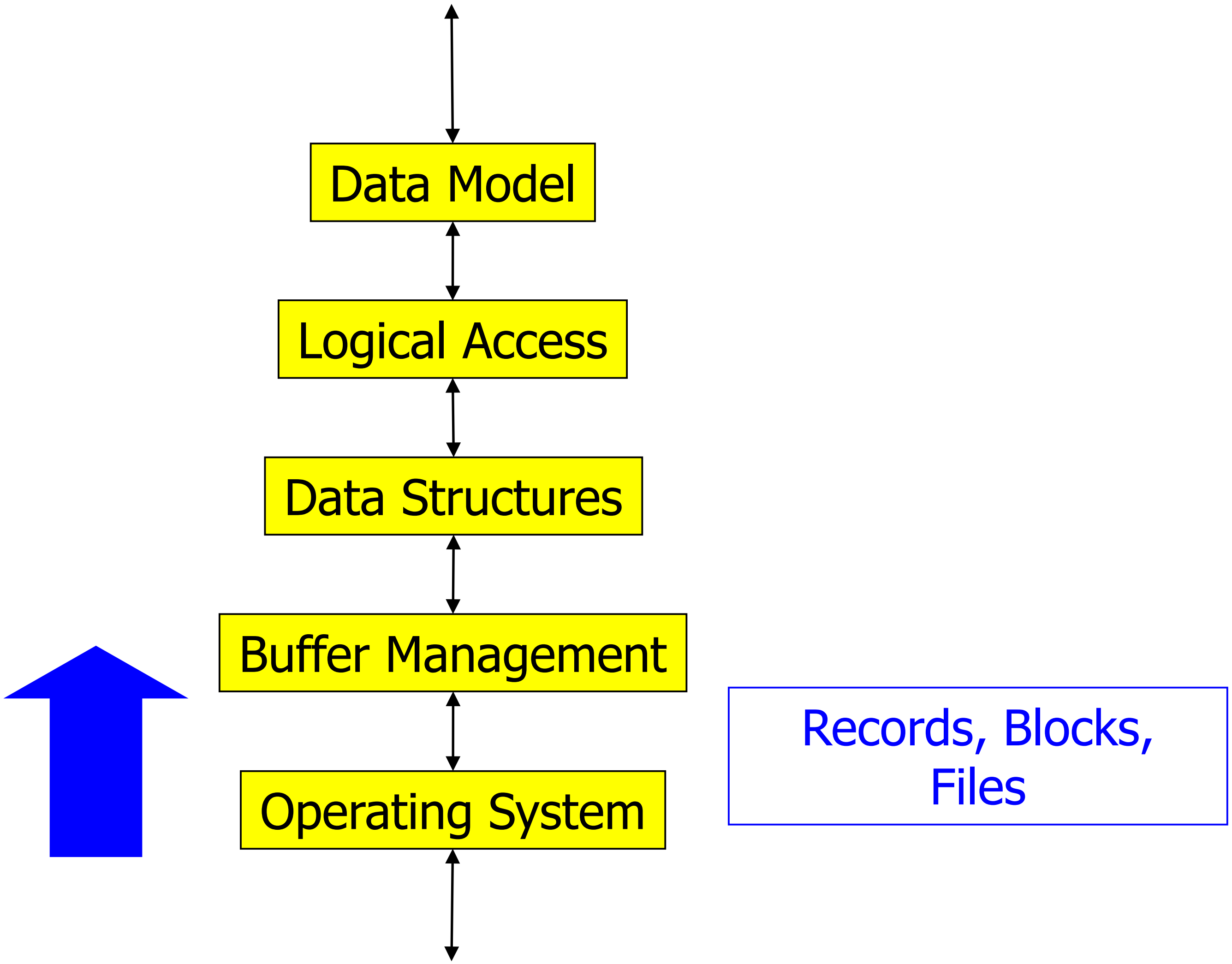
- Durable, slow, cheap, large, robust (compared to …)
- In principle: Same read/write speed
- Much difference between random-access / scan

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	...

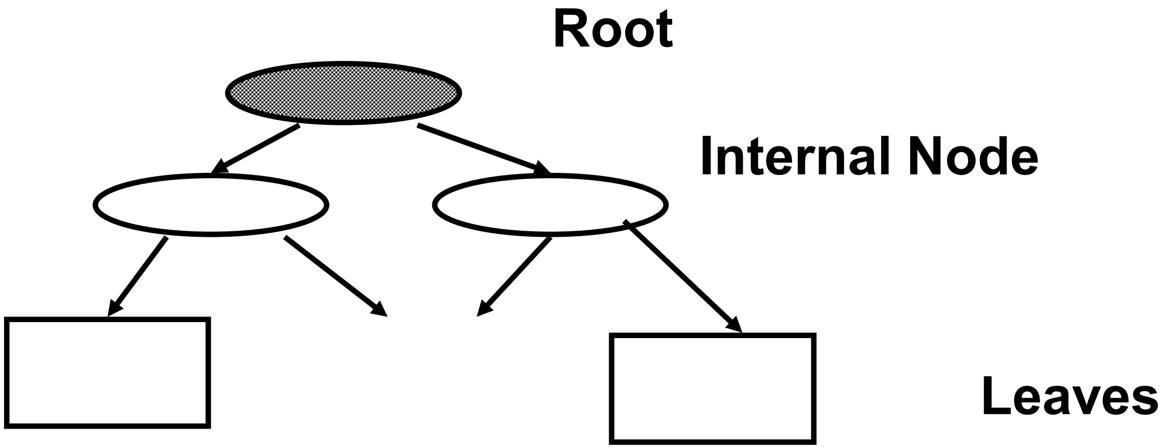
- Operations
  - INSERT(Record): Move to end of file and add, O(1)
  - SEEK(TID): Sequential scan, O(n)
    - FIRST (File): O(1)
    - NEXT(File): O(1)
    - EOF (File): O(1)
  - DELETE(TID): Seek TID; flag as deleted, O(n)
  - REPLACE(TID, Record): Seek TID; write record, O(n)
    - What happens if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

Access Methods: Sequential sorted Files

123	Mason	…
1522	Bond	…
…	…	…
1754	Miller	…

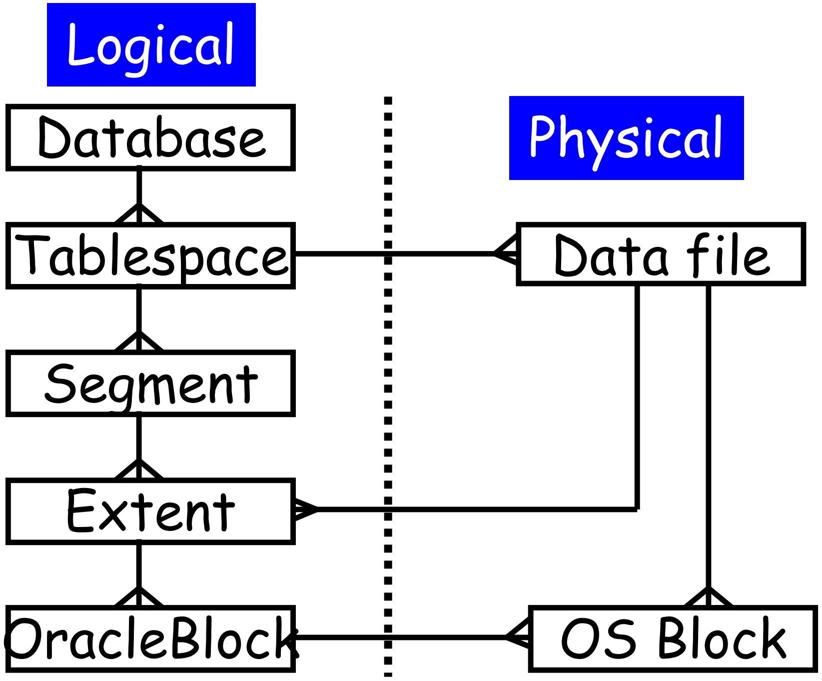
- Operations
  - SEEK(TID): Bin search, O(log(n))
    - But a lot of random access
    - Might be slower than scanning the file
  - INSERT(Record): Seek(TID), move records by one, O(n)
    - This is terribly expensive
  - …

Indexed Files



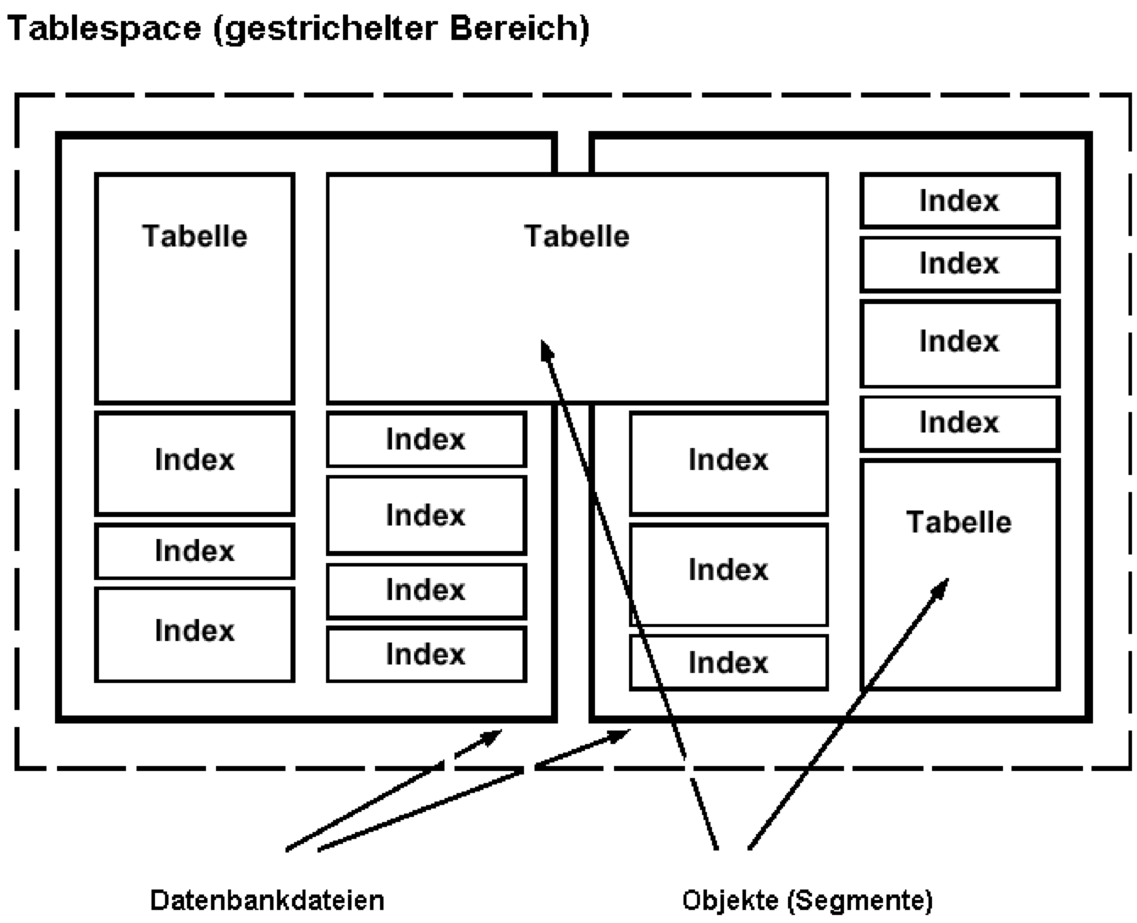
- Operations
  - SEEK(TID): Using order in TIDs: O(log(n))
    - Only if tree is balanced
    - Only if tree is ordered by the right attribute
  - INSERT(TID): Seek TID and insert; possibly restructuring
  - …

Storage in Oracle

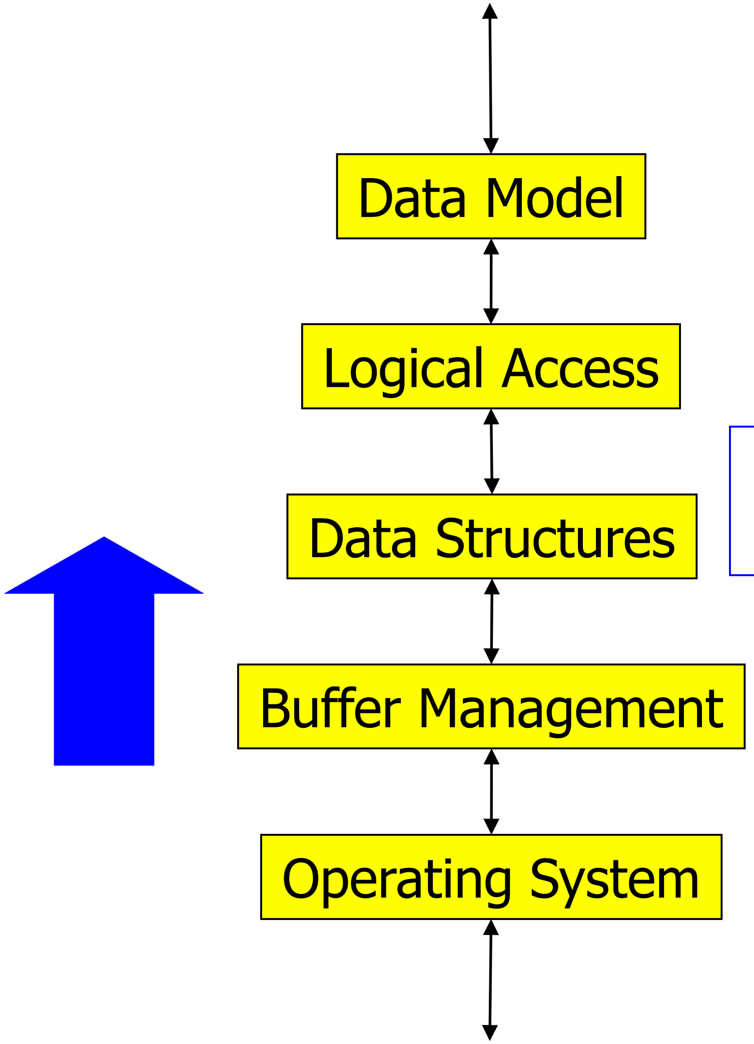


- Data files are assigned to tablespaces
  - May consist of multiple files
  - All data from one object (table, index) are in one tablespace
    - But table and index can be in different ones
  - Backup, quotas, access, …
- Extents: Continuous sequences of blocks on disc
- Space is allocated in extents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

Managing space in Oracle



Bottom-Up



Data Model

Logical Access

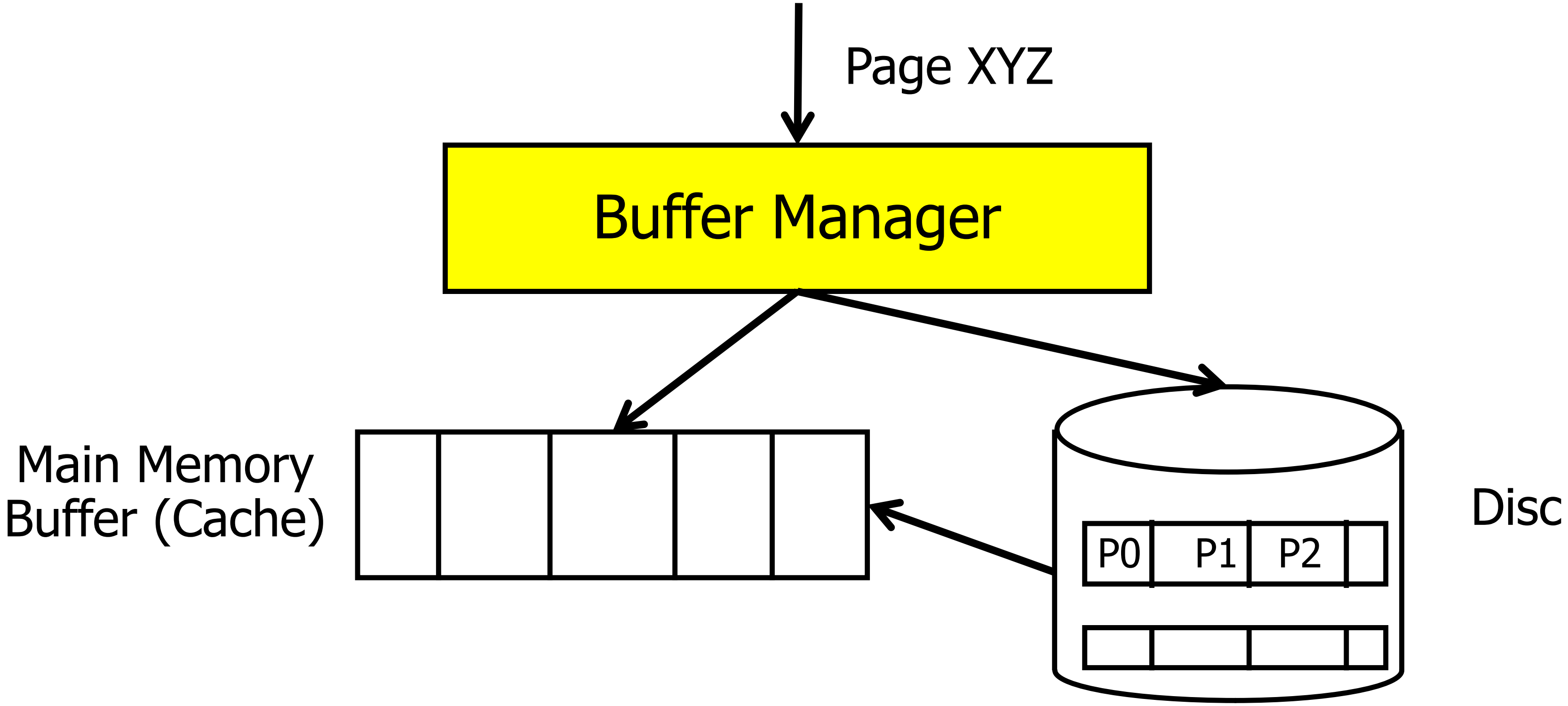
Data Structures

Buffer Management

Operating System

Virtual – physical blocks, access paths

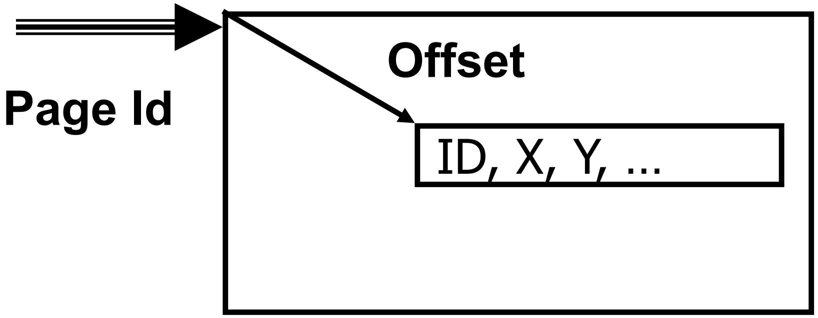
Caching = Buffer Management



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

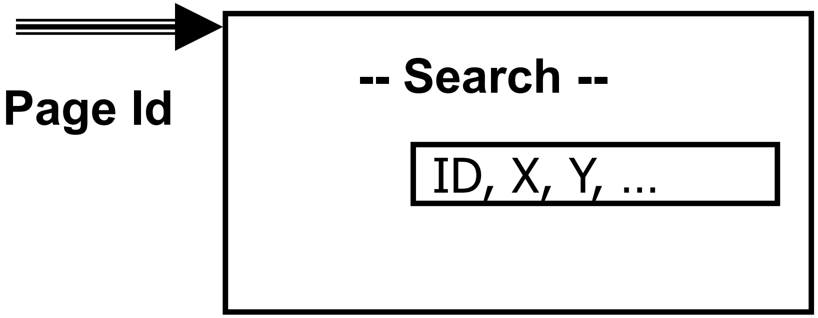
From Buffers to Records

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



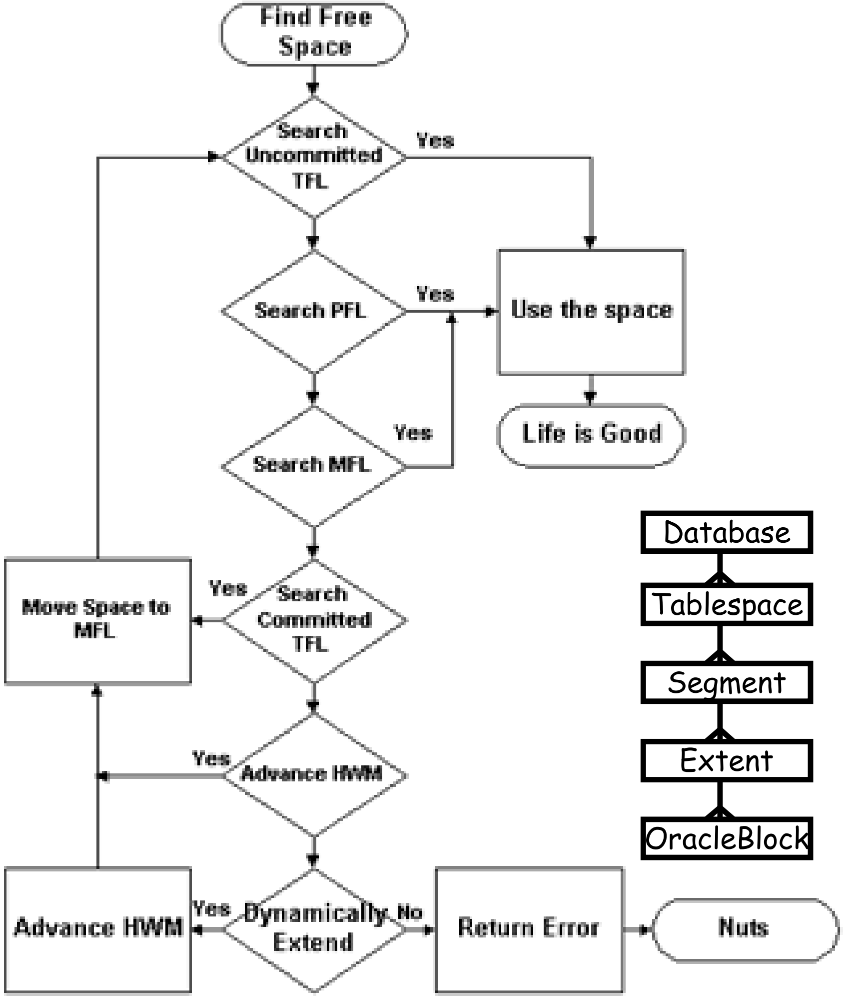
- Pro: Fast access
- Con: Records cannot be moved

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



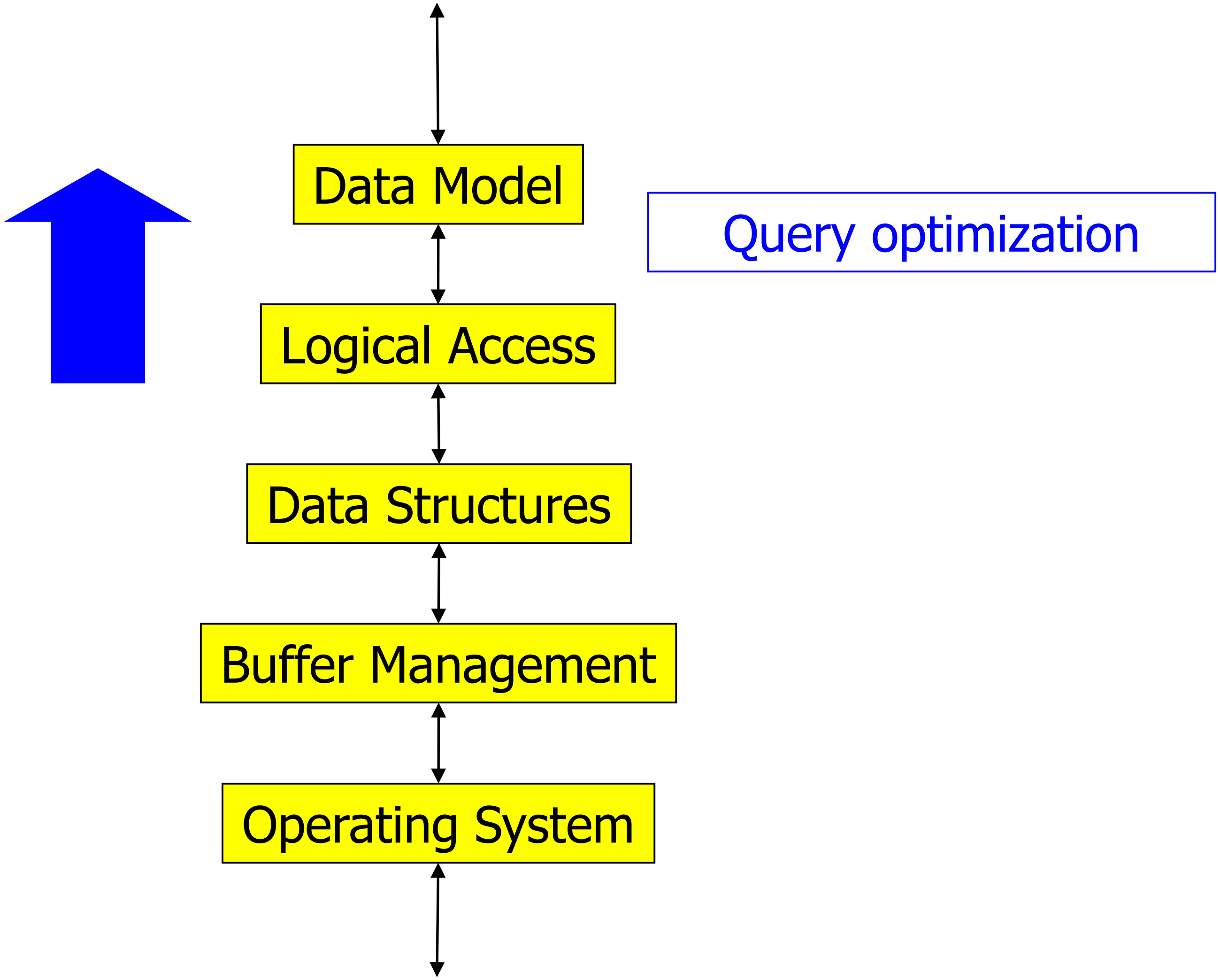
- Pro: Records can be moved within page
- Con: Slower access

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

Bottom-Up



Data Model

Logical Access

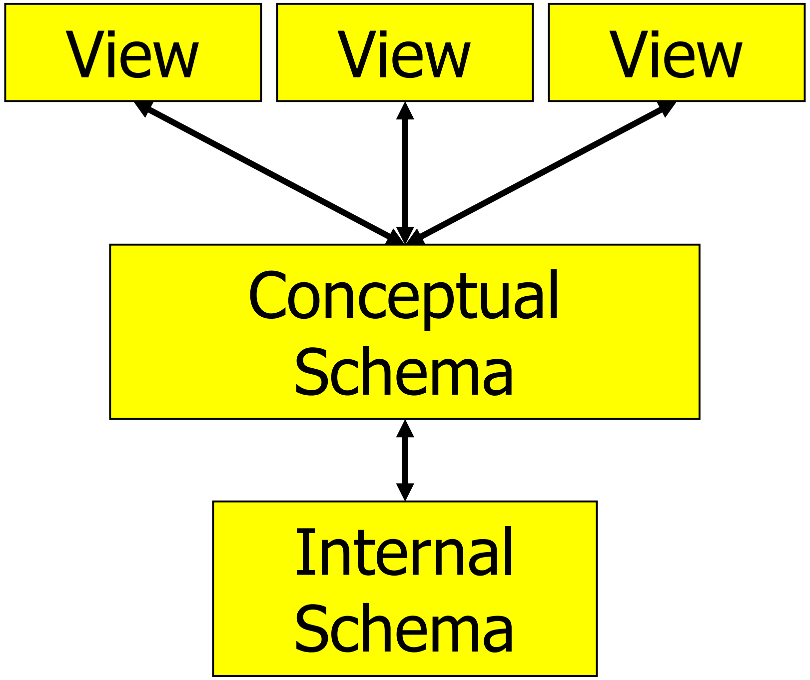
Data Structures

Buffer Management

Operating System

Query optimization

The ANSI/SPARC Three Layer-Model



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)
```

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 implementation
  - 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

Cost-Based Optimizer

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

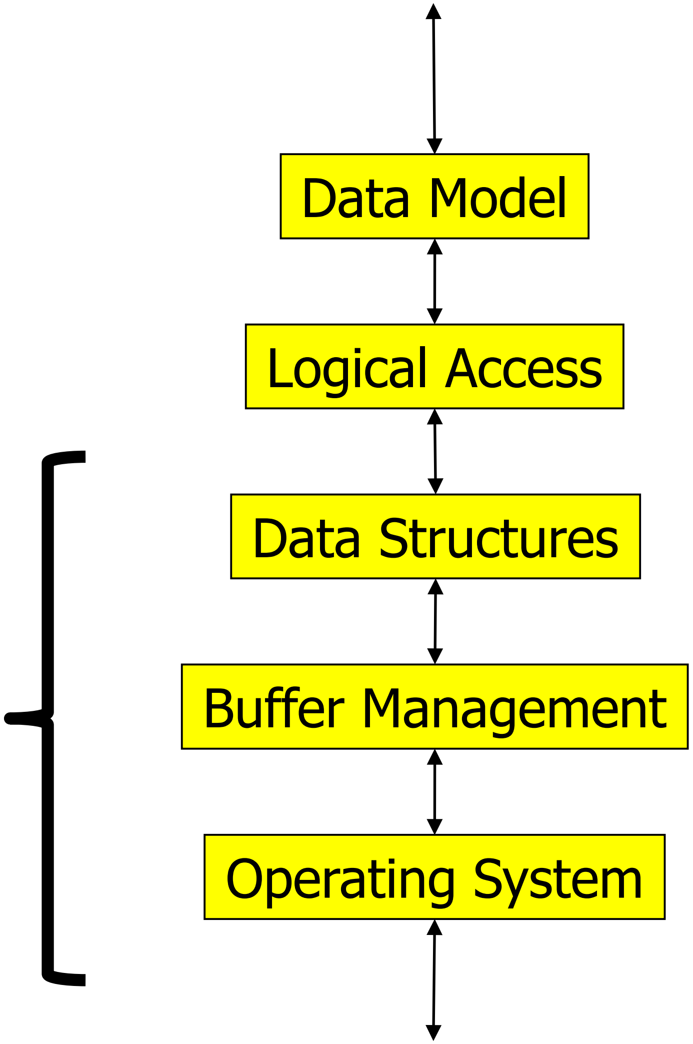
```
FOR EACH a in ACCOUNT DO
   FOR EACH  c IN CUSTOMER DO
      IF a.Account# = c.Account# THEN
        IF c.Name = "BOND" THEN …
```

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

Bottom-Up



Transactions, serializability, recovery

Data Model

Logical Access

Data Structures

Buffer Management

Operating System

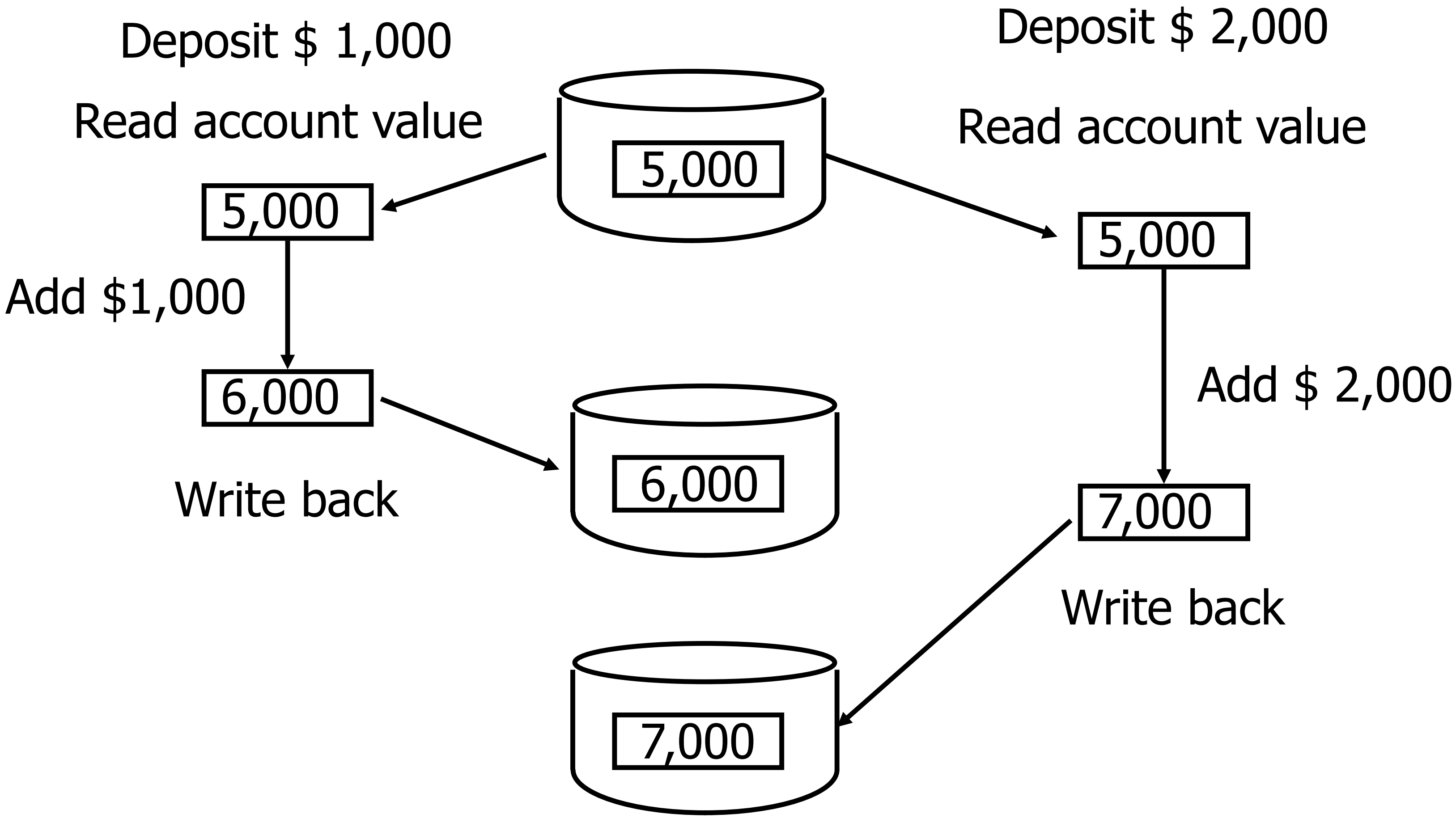
Transactions (TX)

- 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 schedule „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