



PA152: Efficient Use of DB  
1. Introduction

Vlastislav Dohnal

# Motivation

## ■ End-users report slow responses

```
SELECT s.RESTAURANT_NAME, t.TABLE_SEATING, to_char(t.DATE_TIME,'Dy, Mon FMDD') AS THEDATE,
to_char(t.DATE_TIME,'HH:MI PM') AS THETIME,to_char(t.DISCOUNT,'99') || '%' AS AMOUNTVALUE,t.TABLE_ID,
s.SUPPLIER_ID, t.DATE_TIME, to_number(to_char(t.DATE_TIME,'SSSS')) AS SORTTIME
FROM TABLES_AVAILABLE t, SUPPLIER_INFO s,
(SELECT s.SUPPLIER_ID, t.TABLE_SEATING, t.DATE_TIME, max(t.DISCOUNT) AMOUNT, t.OFFER_TYPE
FROM TABLES_AVAILABLE t, SUPPLIER_INFO s
WHERE t.SUPPLIER_ID = s.SUPPLIER_ID
and (TO_CHAR(t.DATE_TIME, 'MM/DD/YYYY') != TO_CHAR(sysdate, 'MM/DD/YYYY')
or TO_NUMBER(TO_CHAR(sysdate, 'SSSS')) < s.NOTIFICATION_TIME - s.TZ_OFFSET)
and t.NUM_OFFERS > 0 and t.DATE_TIME > SYSDATE and s.CITY = 'SF'
and t.TABLE_SEATING = '2' and t.DATE_TIME between sysdate and (sysdate + 7)
and to_number(to_char(t.DATE_TIME, 'SSSS')) between 39600 and 82800
and t.OFFER_TYPE = 'Discount'
GROUP BY s.SUPPLIER_ID, t.TABLE_SEATING, t.DATE_TIME, t.OFFER_TYPE) u
WHERE t.SUPPLIER_ID=s.SUPPLIER_ID and u.SUPPLIER_ID=s.SUPPLIER_ID and t.SUPPLIER_ID=u.SUPPLIER_ID
and t.TABLE_SEATING = u.TABLE_SEATING and t.DATE_TIME = u.DATE_TIME
and t.DISCOUNT = u.AMOUNT and t.OFFER_TYPE = u.OFFER_TYPE

and (TO_CHAR(t.DATE_TIME, 'MM/DD/YYYY') != TO_CHAR(sysdate, 'MM/DD/YYYY')
or TO_NUMBER(TO_CHAR(sysdate, 'SSSS')) < s.NOTIFICATION_TIME - s.TZ_OFFSET)
and t.NUM_OFFERS > 2 and t.DATE_TIME > SYSDATE and s.CITY = 'SF'
and t.TABLE_SEATING = '2' and t.DATE_TIME between sysdate and (sysdate + 7)
and to_number(to_char(t.DATE_TIME, 'SSSS')) between 39600 and 82800 and t.OFFER_TYPE = 'Discount'
ORDER BY AMOUNTVALUE DESC, t.TABLE_SEATING ASC, upper(s.RESTAURANT_NAME) ASC,
SORTTIME ASC, t.DATE_TIME ASC
```

# Motivation (cont.)

## Execution Plan

---

```
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=165 Card=1 Bytes=106)
1  0  SORT (ORDER BY) (Cost=165 Card=1 Bytes=106)
2  1  NESTED LOOPS (Cost=164 Card=1 Bytes=106)
3  2  NESTED LOOPS (Cost=155 Card=1 Bytes=83)
4  3  TABLE ACCESS (FULL) OF 'TABLES_AVAILABLE' (Cost=72 Card=1 Bytes=28)
5  3  VIEW
6  5  SORT (GROUP BY) (Cost=83 Card=1 Bytes=34)
7  6  NESTED LOOPS (Cost=81 Card=1 Bytes=34)
8  7  TABLE ACCESS (FULL) OF 'TABLES_AVAILABLE' (Cost=72 Card=1 Bytes=24)
9  7  TABLE ACCESS (FULL) OF 'SUPPLIER_INFO' (Cost=9 Card=20 Bytes=200)
10 2  TABLE ACCESS (FULL) OF 'SUPPLIER_INFO' (Cost=9 Card=20 Bytes=460)
```

Access method

Evaluation costs

# Course's Objectives

- Understand query processing
  - Identify weaknesses and optimize
- Implement advanced indexing
- Design storage for DB
  - Know important file structures
- Understand and implement high availability

# Course Outline

## ■ Data storage

Storage hierarchy, RAID, failures, ...

## ■ Data formatting

Records, blocks, ...

## ■ Indexing

Trees, hashing, ...

## ■ Query processing

Cost estimates, joining relations, ...

## ■ Query optimization

Creating indexes, views, table partitioning, ...

# Course Outline

- Database tuning

Tuning relation schema, db monitoring, ...

- Transaction processing

Concurrent processing, locking, logging, deadlocks, ...

- Access control

access rights, roles, rewrite rules, ...

# Recommended Reading

## ■ Books

### □ Database Systems Implementation

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
- Prentice Hall, 2000
- Signature in FI library: D89

### □ Database Systems: The Complete Book

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
- 2nd edition, Prentice Hall, 2009
- Signature in FI library: D147

# Credits

- Materials are based on presentations:
  - Courses CS245, CS345, CS345
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
    - Stanford University, California



# Requirements to pass

- There are two possible completion types
  - Exam ('zk')
  - Credit ('z')
  - Double-check requirements of your study program
- Home assignments and exam
  - [see in course's study materials](#)

# Knowledge Prerequisites

- Relational model
- Query languages
  - SQL and relational algebra
- File organizations
  - Sequential file, ...
- Most covered in a bachelor-degree course:
  - PB154 Fundamentals of Database Systems
  - PB168 Fundamentals of Information and Database Systems

# Basic Terminology

- „Database“
  - “programmed” by most of coders
  - needed by every business
  - included in most applications
  - queried by SQL
- Relational model
  - Structure – data in relations (tables)
  - Operations – querying, updates
    - SQL, relational algebra
- Database system (Database management system = DBMS)
  - collection of tools for storing and processing of data
- Database
  - data that are processed, interesting for a business
  - collection of relations, integrity constraints, indexes, ...
  - database schema vs. database instance

# Example

```
select * from student where  
    program='N-SWE' and field='DEV'
```

## ■ Processing:

1. Check the query and get attributes of student
2. Read rows from student and evaluate the condition

No	UČO	Name	Type of completion	Covid	Program	Field
1.	<a href="#">4x4x7x</a>	<b>Martin</b>	zk	● ONT	N-SWE	DEV
2.	<a href="#">4x5x9x</a>	<b>Laura</b>	zk	● ONT	N-SWE	DEV
3.	<a href="#">5x6x0x</a>	<b>Martin</b>	zk	● ONT	N-SWE	DEV
4.	<a href="#">4x5x5x</a>	<b>Peter</b>	zk	● ONT	N-SWE	DEV
5.	<a href="#">4x5x2x</a>	<b>Martin</b>	zk	● ONT	N-SWE	DEV
6.	<a href="#">4x4x4x</a>	<b>Michal</b>	zk	● ONT	N-SWE	DEV

# Example 2

`select A,B from R,S where condition`

## ■ Query processing

1. Read dictionary and get attributes of R and S
2. Read R and for each line do:
  - a. Read S and for each line do:
    - i. Join the lines (records/tuples)
    - ii. Evaluate condition
    - iii. If valid, project and add to result

# Implementation of example

- Relations stored in files on disk

- Relation *student* in `/usr/db/student`

```
4x4x7x # Martin # zk # ONT # N-SWE # DEV ↵  
4x5x9x # Laura # zk # ONT # N-SWE # DEV ↵  
5x6x0x # Martin # zk # ONT # N-SWE # DEV ↵  
4x5x5x # Peter # zk # ONT # N-SWE # DEV ↵  
4x5x2x # Martin # zk # ONT # N-SWE # DEV ↵  
4x4x4x # Michal # zk # ONT # N-SWE # DEV ↵  
:  
:
```

- Data schema is not present

# Implementation of example

- List of existing relations in *dictionary*
  - /usr/db/dictionary

```
student # UČO # INT # Name # STR # Type  
of completion # STR # Covid # STR #  
Program # STR # Field # STR ←  
R # name # STR # id # INT # dept STR ... ←  
R2 # C # STR # A # INT ... ←  
  
⋮
```

# Implementation Issues

## ■ Storage

- Bizarre formatting – separators and new lines
  - Change in a value leads to change in whole file
- Respecting underlying technology?

## ■ Querying costly

- Indexes?

## ■ Data consistency

- Primary keys, references, ...

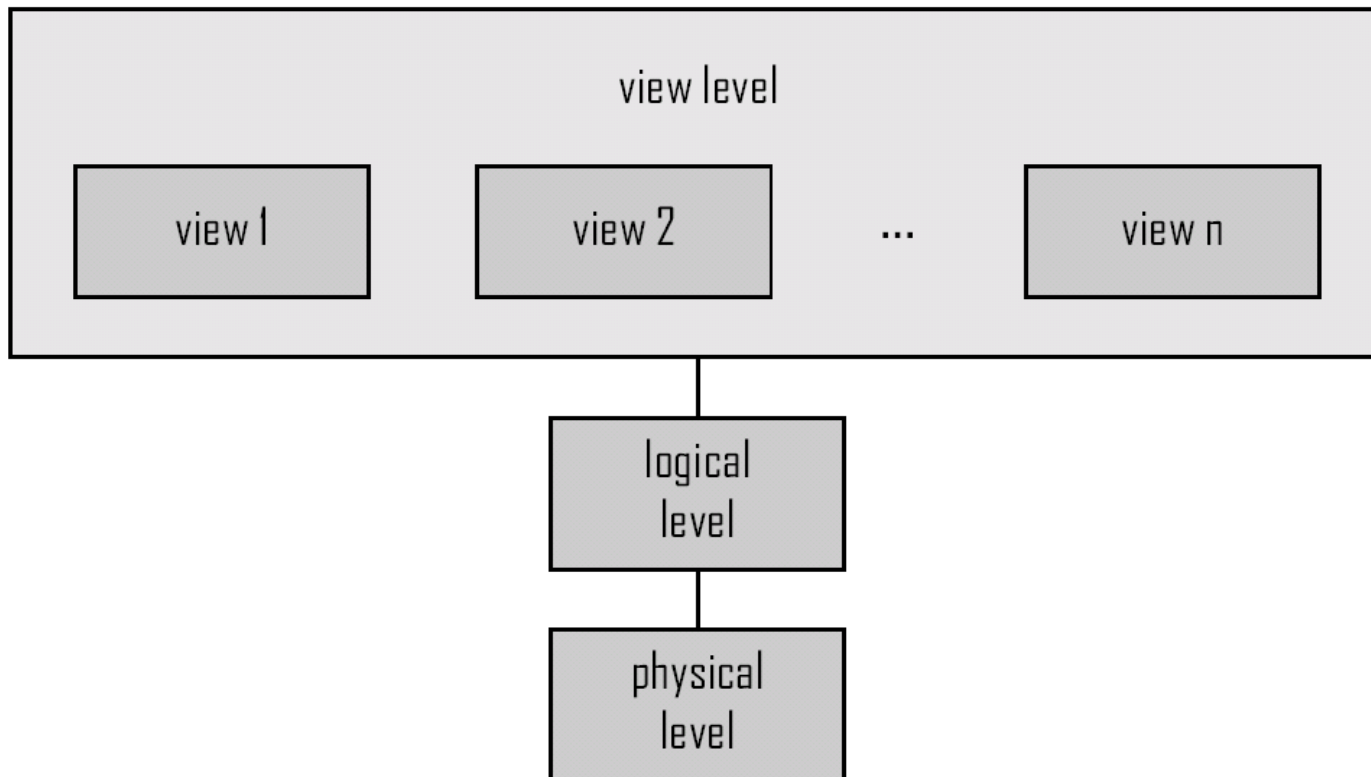


# Implementation Issues

- No concurrent processing
- No reliability
  - Data loss frequent
  - Operation may not be finished
- No access control
  - Accessed checked on filesystem level only
  - Rights too coarse
- No API, no GUI

# Database System

- DBMS (Database Management System)
  - Architecture:



# DBMS Main Components

## ■ Storage Manager

- manages blocks on disks
- manages buffers/caches

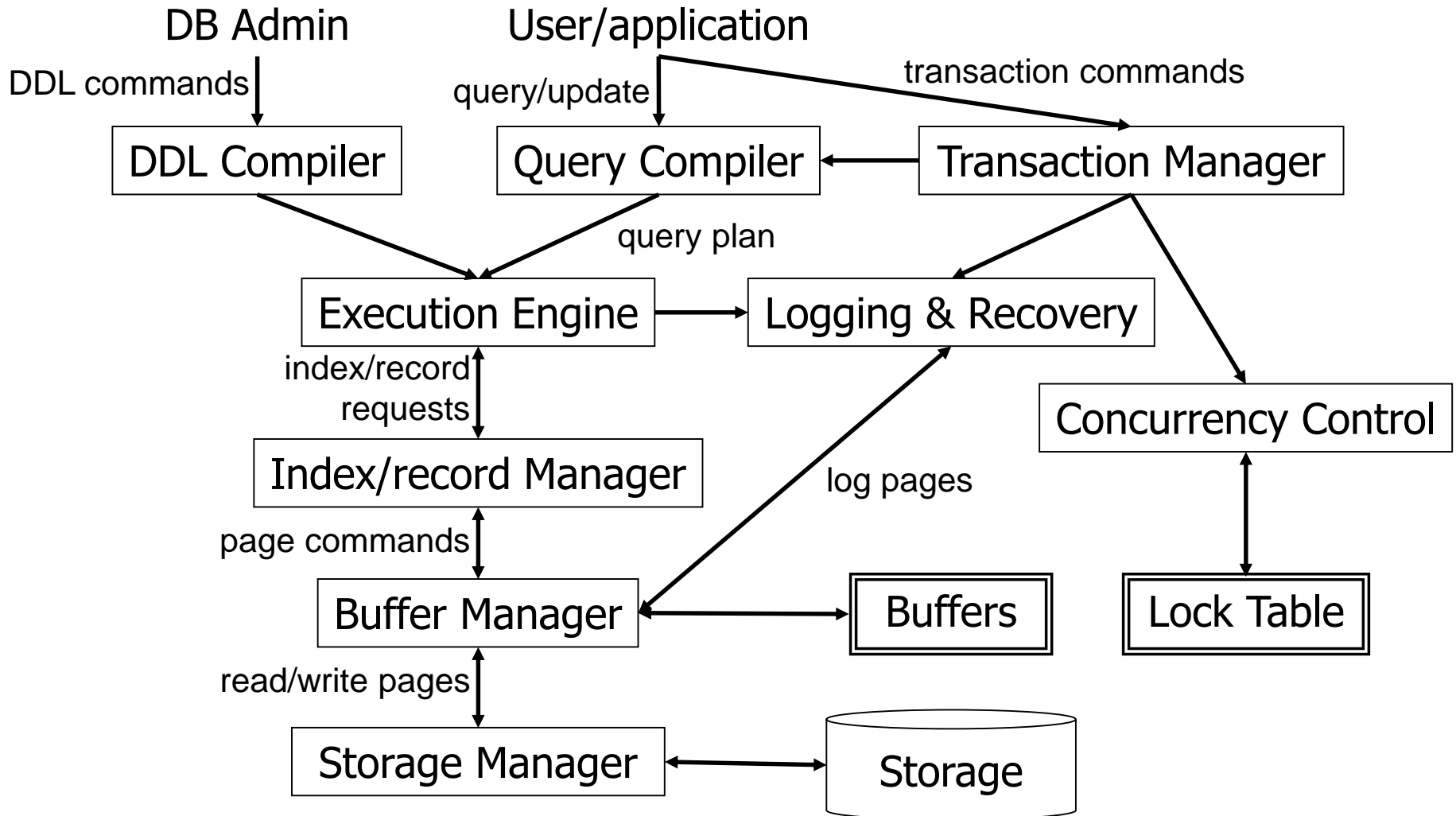
## ■ Query Processor

- query translation, optimization
- query execution

## ■ Transaction Manager

- atomicity, consistency, isolation, durability of transaction processing

# DBMS Components



# Storage Hierarchy



- Primary

- cache

- CPU

- main (operation)

- RAM

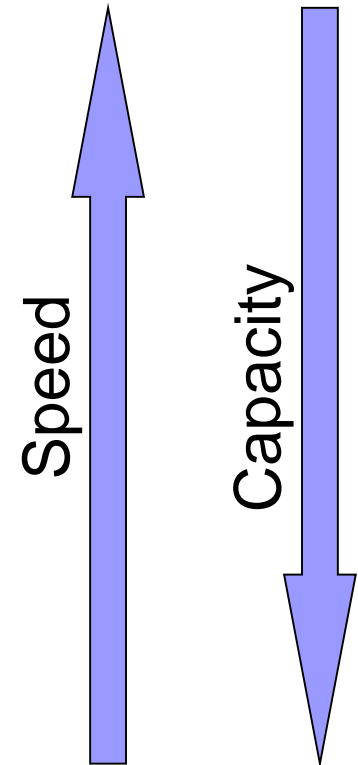
- Secondary

- disk, flash

- Tertiary

- tapes, optical discs

- for backups



# Empirical Laws

## ■ Number of transistors

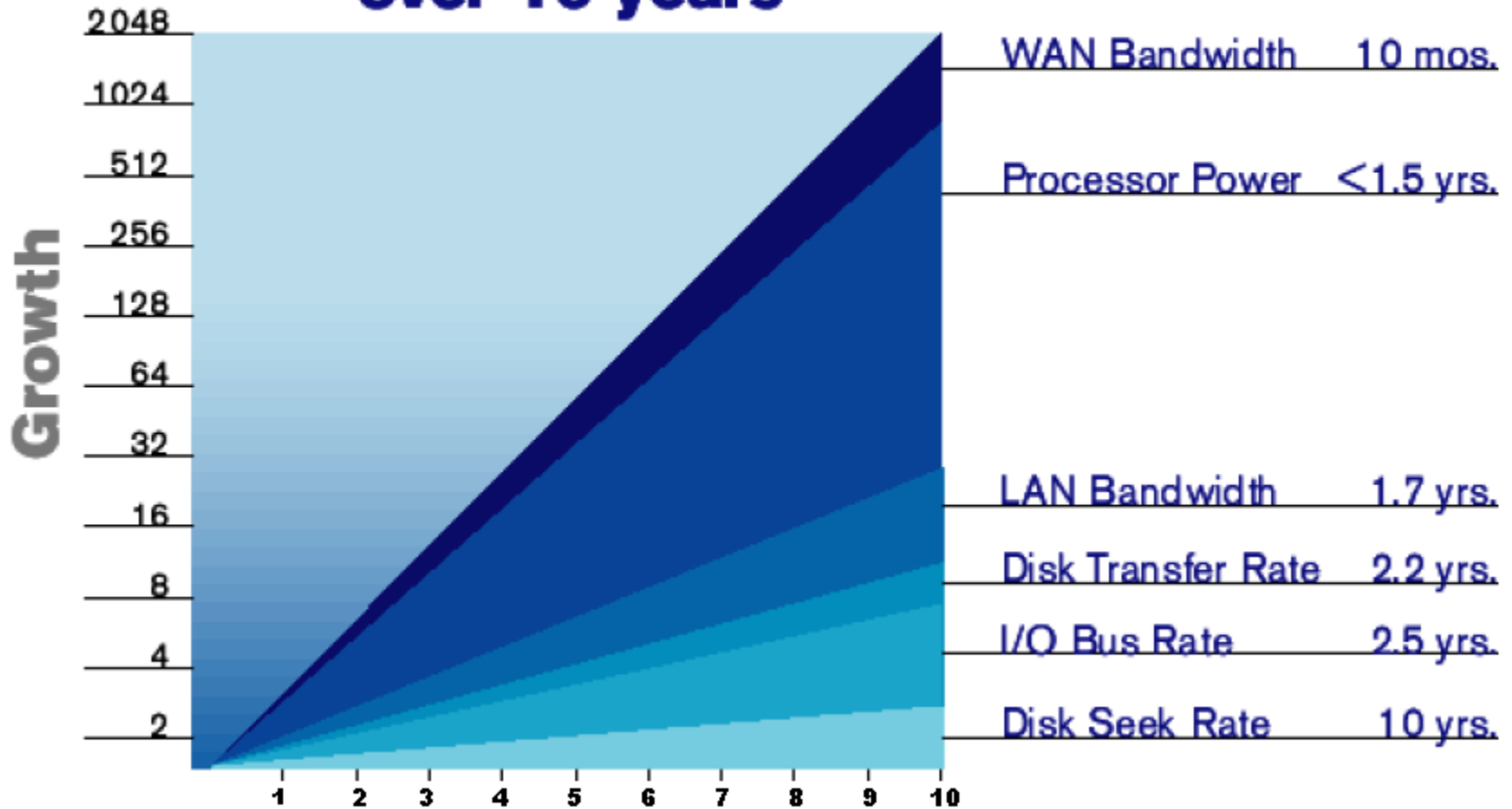
- Density doubles each 2 years
- CPU speed
- Memory capacity
- so-called Moore's Law

## ■ Disk capacity

- Kryder's Law – 1000 times more in 15 years
- Caveat: not applicable to disk speed!

# Empirical Laws

## Technology Growth Rates over 10 years



# Storage Type

## ■ Cache

- Fastest, most expensive, volatile

## ■ RAM

- Fast – 10-100 ns ( $1 \text{ ns} = 10^{-9} \text{ s}$ )
- Too small or expensive for whole database
- Volatile

## ■ Flash

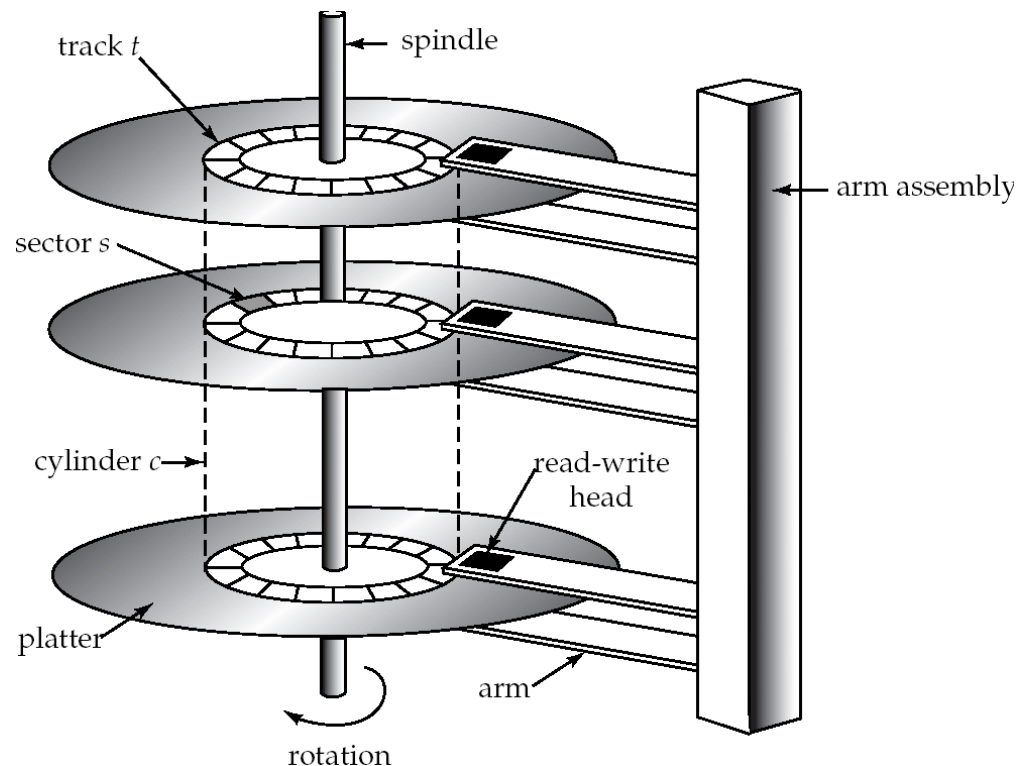
- Fast reads, non-volatile ( $25 \text{ us} = 25 \cdot 10^{-6} \text{ s}$ )
- Slow writes
  - erase first, (whole region/bank) ( $2 \text{ ms} = 2 \cdot 10^{-3} \text{ s}$ )
  - write then ( $250 \text{ us} = 250 \cdot 10^{-6} \text{ s}$ )
- Limited write cycles



# Storage Type

## ■ Magnetic disk

- Large capacity, non-volatile
- Reads and writes of the same speed (10ms)



# Magnetic disk

## ■ Access time

- Time from a request for reading/writing to the beginning of data transfer

## ■ Blocking data

- disk sector/block is atomic unit

## ■ Access time components

- Seek time – 4-10 ms

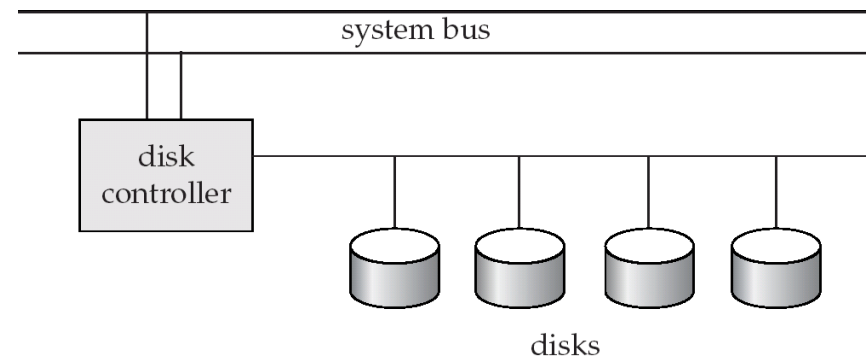
- Move heads to the correct track
- Average seek time =  $\frac{1}{2}$  worst case seek time

- Rotational delay – 4-11ms (5400-15k rpm)

- Time for revolving to the correct sector
- Average latency =  $\frac{1}{2}$  worse case latency

# Magnetic disk (cont.)

- Transfer rate
  - Speed of reading/writing from/to a disk
  - 50-200MB/s, lower for inner tracks
- Speed of controller
  - More drives connected to one controller
  - SATA 3.2 (16Gb/s) – 2 GB/s
  - SAS-3 (12Gb/s) – 1.17 GB/s



# Magnetic disk (cont.)

## ■ Properties

- Slow random read

  - access time can be up to 20ms

- Fast sequential read

## ■ Optimization in HW

- Cache

  - Buffers for writing, battery-backups or flash

- Algorithms for arm moving minimization

  - Algorithm „elevator“

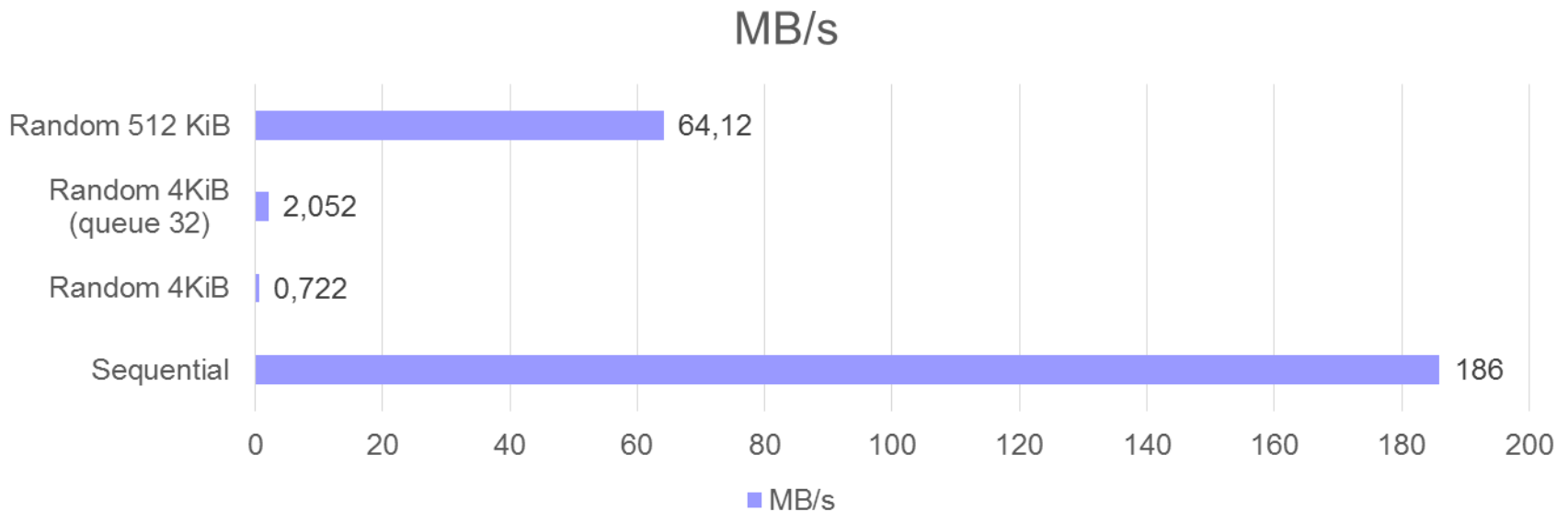
  - Work well when many concurrent requests

# Magnetic disk (cont.)

- Example SATAII disk, 7200rpm, 100MB/s
  - Avg seek time = 8.9ms
  - Avg latency =  $(1/(7200/60))*0.5=0.00417s=4.17ms$
  - Reading a sector (512B) = 13.07ms + 4.88μs
  - 10MB continuous = 13.07ms + 100ms = 113ms
    - consecutive sector reading includes also
      - track-to-track seek – platter / cylinder change
      - is neglected
  - 10MB randomly =  $20480*13.07488ms = 268s$

# Magnetic disk (cont.)

Western Digital 10EZEX 1TB, SATA3, 7200 RPM, sustained transfer rate 150 MB/s



# Disk ops

- Access in blocks (atomic unit)
  - Group of consecutive sectors (of 512KB)
  - Typically, 4KiB
- Block reading
- Block writing
  - Write and verify (rotate disk + reading!)
- Block update
  - Read
  - Change in memory
  - Write and verify

# Algorithms in DBMS

- Work with blocks
  - block size in DB – 4KiB – 16KiB
  - block size in FS – 1-4KiB
  - block size in device – 512B, 4KiB
- Minimize random accesses
- Cost for reading and writing taken as same
  - Typical simplification



# Solid State Drives

- Disk storage on „flash“ memory
- No moving parts
- More resistant to damage
- Quiet, low access time and delay
- 4x more expensive than HDD (per GB)

# SSD – Flash memory

## ■ NAND chips

### □ SLC (single-level cells)

- stores 1-bit information (2 voltage states)
- fastest, highest cost

### □ MLC (multi-level cells)

- stores mostly 2-bit information

### □ TLC (triple-level cells)

- stores 3-bit information (8 voltage states)
- slowest, least cost

### □ QLC (quad-level cells)

# SSD – Flash performance

	SLC	MLC	TLC	HDD	RAM
P/E cycles	100k	10k	5k	*	*
Bits per cell	1	2	3	*	*
Seek latency ( $\mu$ s)	*	*	*	9000	*
Read latency ( $\mu$ s)	25	50	100	2000-7000	0.04-0.1
Write latency ( $\mu$ s)	250	900	1500	2000-7000	0.04-0.1
Erase latency ( $\mu$ s)	1500	3000	5000	*	*
<i>Notes</i>	* metric is not applicable for that type of memory				

Source: <https://www.extremetech.com/extreme/210492-extremetech-explains-how-do-ssds-work>

# SSD – Flash memory

- Terminology shift! (block → page)
- Organization
  - Pages of 4KiB organized into blocks (64/128 pages)
  - Much faster reading than writing
    - Write: Only in “a new block”
      1. write a new copy of the changed data to a fresh block
      2. remap the file pointers
      3. then erase the old block later when it has time
  - Write restrictions – 1k-100k overwrite cycles
  - Reliability increased ECC sums
    - Hamming, Reed-Solomon

# SSD – Flash memory

- A single NAND chip is relatively slow
  - SLC NAND
    - ~25  $\mu\text{s}$  to read a 4 KiB page
    - ~250  $\mu\text{s}$  to commit a 4 KiB page
    - ~2 ms to erase a 256 KiB block
- Data striping (RAID0) to improve performance

# Lecture's Takeaways

- Terminology revision
- Categorization of storage
- Organization principles of HDD and SSD

# Student's DB at FI

- Tech info <https://www.fi.muni.cz/tech/unix/databases.html.en>
  - Follow instructions to create an account in MySQL, PostgreSQL, Oracle
- PostgreSQL <http://www.postgresql.org/>
  - Free even for commercial deployment
  - Install pgAdmin <http://www.pgadmin.org/> for connecting to DB
    - Host name: db.fi.muni.cz
    - Port: 5432
    - Database: pgdb
    - CAVEAT: Access is restricted to FI's intranet, so use VPN or Putty
      - FI's VPN: <https://www.fi.muni.cz/tech/unix/vpn.html.en>
      - Putty: see the next slide

# Student's DB at FI

## ■ Connecting via Putty tunnel

- Install Putty
- Setup connection to aisa.fi.muni.cz (login and password as to a FI's computer)
- Configure forwarded port as follows and **then in PgAdmin use localhost as the host name.**

