IF3140 – Sistem Basis Data
Performance Tuning

SEMESTER I TAHUN AJARAN 2024/2025

Modified from Silberschatz's slides, "Database System Concepts", 7th ed.

Sumber

Silberschatz, Korth, Sudarshan: "Database System Concepts", 7th Edition

- **Chapter 25:** Performance Tuning & Performance Benchmark
- **Chapter 12.5:** RAID
- **Chapter 14:** Indexing

Objectives

Students are able to:

- Explain factors which impact database performance
- Describe several strategies to improve database performance
- Know database design and hardware aspects that respectively affect database transaction efficiency and system performance
- Know performance benchmarking

KNOWLEDGE & SOFTWARE ENGINEERING

Outline

Overview

Database performance can be defined as the optimization of resource use to increase throughput and minimize contention, enabling the largest possible workload to be processed

Performance Tuning

- Adjusting various parameters and design choices to improve system performance (higher throughput)
- **General procedure**:
  - identifying bottlenecks, and
  - eliminating them.
- **Examples**:
  - Long delay in web application → further investigation → full relation scan query → adding an index
  - Long delay → further investigation → unnecessarily complicated query → rewrite the query

KNOWLEDGE & SOFTWARE ENGINEERING

What affects performance?

- Hardware
  - CPU
  - Network
  - RAM
  - Disk
- Database Server parameter settings
- Database Design (schema)
- Indices on Database Tables
- SQL statement

KNOWLEDGE & SOFTWARE ENGINEERING

Bottlenecks Illustration

- Performance of most systems (at least before they are tuned) usually limited by performance of one or a few components:
  - **E.g.** 80% of the code may take up 20% of time and 20% of code takes up 80% of time
    - Worth spending most time on 20% of code that take 80% of time
- Bottlenecks may be in hardware (e.g. disks are very busy, CPU is idle), or in software
- Removing one bottleneck often exposes another
- De-bottlenecking consists of repeatedly finding bottlenecks, and removing them → heuristic

Tuning Levels

- Can tune a database system at 3 levels:
  - **<u>Higher level database design</u>**,

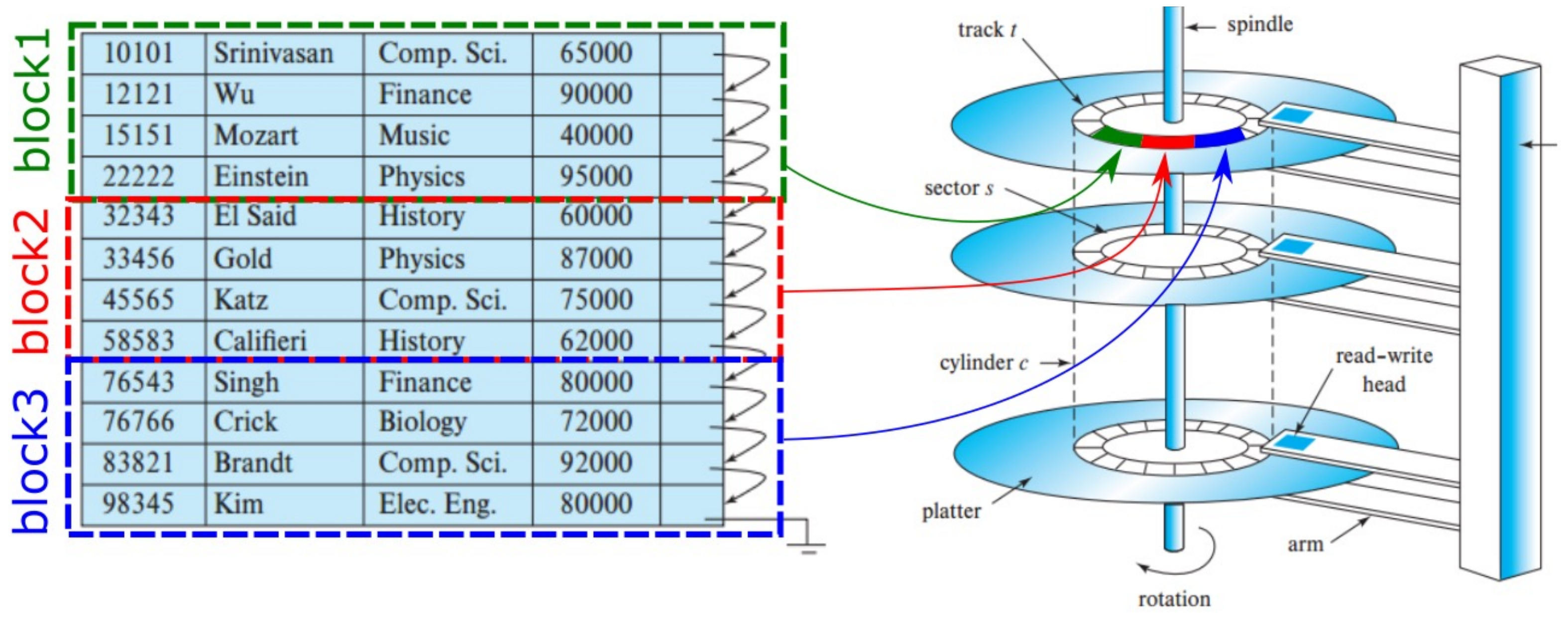
    E.g., Physical schema (indices, materialized view, horizontal splitting), queries, transactions, and logical schema
  - **<u>Database system parameters</u>**

    E.g., Set buffer size to minimize disk I/O, set checkpointing intervals to limit log size.
  - **<u>Hardware</u>**

    E.g., Add disks to speed up I/O, add memory to increase buffer hits, move to a faster processor.

Overview – File Organization

Database performance tuning is generally targeted to **minimize disk I/O** since it dominates the potential bottleneck



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Typical important constants	HDD	SSD
Sector (HDD) /cell (SSD) size	512 B	2 B
Block / pages size	4 KB	4 KB
Random block read time (disk I/O)	10 ms Block access time:5-20ms Block transfer time: 0.1ms	0.1 ms

* Block access time in memory = ~100 ns (nano)

KNOWLEDGE & SOFTWARE ENGINEERING

Overview

- After ER design, schema refinement, and the definition of views, we have the conceptual schema for our database.
- The next step is to: **(1) choose indices**, **(2) make clustering** decisions, and (3) **to refine the schemas** (if necessary), to meet performance goals.
- We must begin by understanding the ***workload***:
  - The most important queries and how often they arise.
  - The most important updates and how often they arise.
  - The desired performance for these queries and updates.

Understanding the Workload

- For each **query** in the workload:
  - Which **relations** does it **access**?
  - Which **attributes** are **retrieved**?
  - Which attributes are **involved in selection/join** conditions? How selective are these conditions likely to be?
- For each **update** in the workload:
  - Which attributes are involved in **selection/join conditions**? How selective are these conditions likely to be?
  - The type of update (INSERT/DELETE/UPDATE), and the **attributes** that are affected.

KNOWLEDGE & SOFTWARE ENGINEERING

Decisions to Make

- What indices should we create?
  - Which **relations should have indices**? What **attribute(s)** should be the search key? Should we build **several indices**?
- For each index, what **kind of an index** should it be?
  - Clustered? Hash/tree? Dynamic/static? Dense/sparse?
- Should we **make changes** to the **logical schema**?
  - Consider alternative normalized schemas? (Remember, there are many choices in decomposing into BCNF, etc.)
  - Should we "undo" some decomposition steps and settle for a lower normal form? *(Denormalization)*
  - Horizontal partitioning, views, etc.

KNOWLEDGE & SOFTWARE ENGINEERING

Indexing dan Index Tuning

KNOWLEDGE & SOFTWARE ENGINEERING

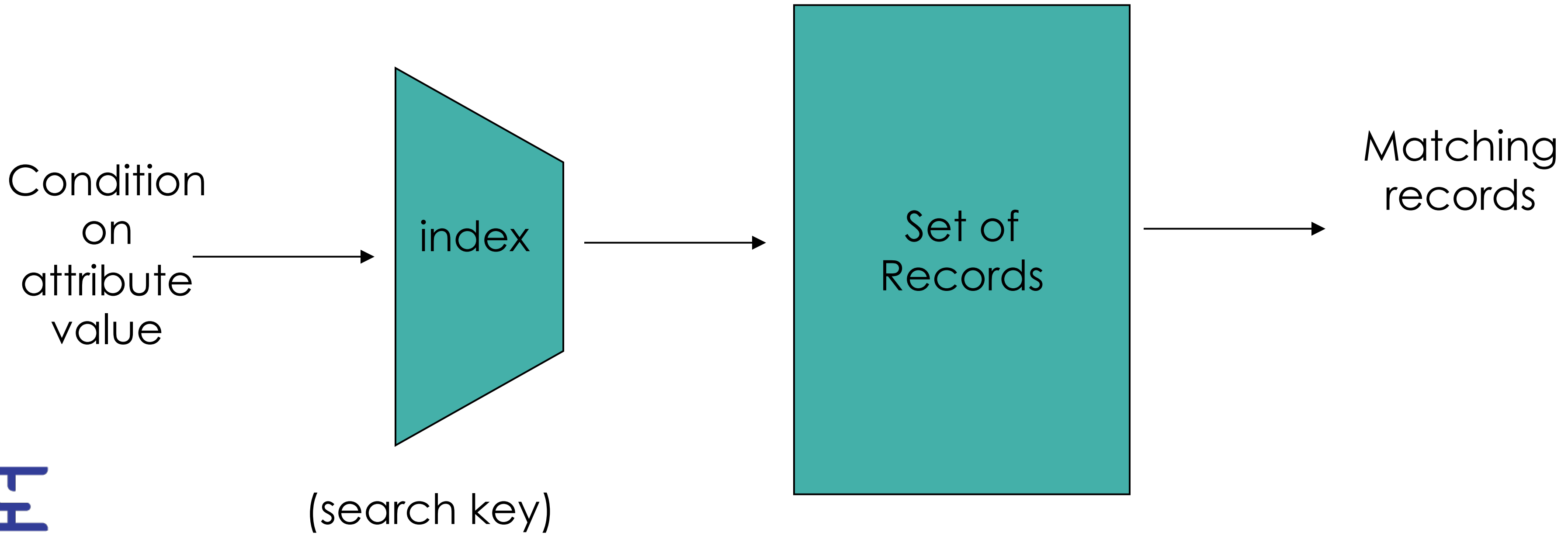
Index

- Index → used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute or set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer

Index

- An index is a data structure that **supports efficient access** to data
- It facilitates searching, sorting, join operation, etc., efficiently instead of scanning all table rows
- Index files are typically **much smaller** than the original file



Schema Tuning

KNOWLEDGE & SOFTWARE ENGINEERING

Schema Tuning (I)

- Choice of logical schema should be guided by workload, in addition to redundancy issues:
  - We might consider **horizontal decompositions.**
  - We might consider **vertical decompositions**.
  - We may settle for a 3NF schema rather than BCNF.
  - Workload may influence choice we make in decomposing a relation into 3NF or BCNF.
  - We may further decompose a BCNF schema!
  - We might **denormalize** (i.e., undo a decomposition step), or we might add fields to a relation.
- If such changes are made after a database in use, called **schema evolution**; might mask changes by defining views

Schema Tuning (2)

- Can be done in several ways:
  - **Splitting tables**
    - Sometimes splitting normalized tables can improve performance
    - Can split tables in two ways: (1) **Horizontally**; (2) **Vertically**
    - Add complexity to the applications
  - **Denormalization**
    - Adding redundant columns
    - Adding derived attributes
    - Collapsing tables
    - Duplicating tables

Query Tuning

KNOWLEDGE & SOFTWARE ENGINEERING

Tuning of Queries: Query Plan

- Optimizer **might not choose best plan**
  - Most DBMS support **EXPLAIN** command to see what plan is being used
- Makes decisions based on existing statistics
  - Statistics may be **old**
    - There is also **ANALYZE** command to re-compute the statistics
- Complex query containing **nested subqueries** are **not optimized** very well by many optimizers. The query can be **re-write using join** (decorrelation technique –see chapter 16.4.4 –)
- **Optimizer hints**: special instructions for the optimizer embedded in the SQL command text

Note: Rewriting query becomes less relevant since optimizers goes better

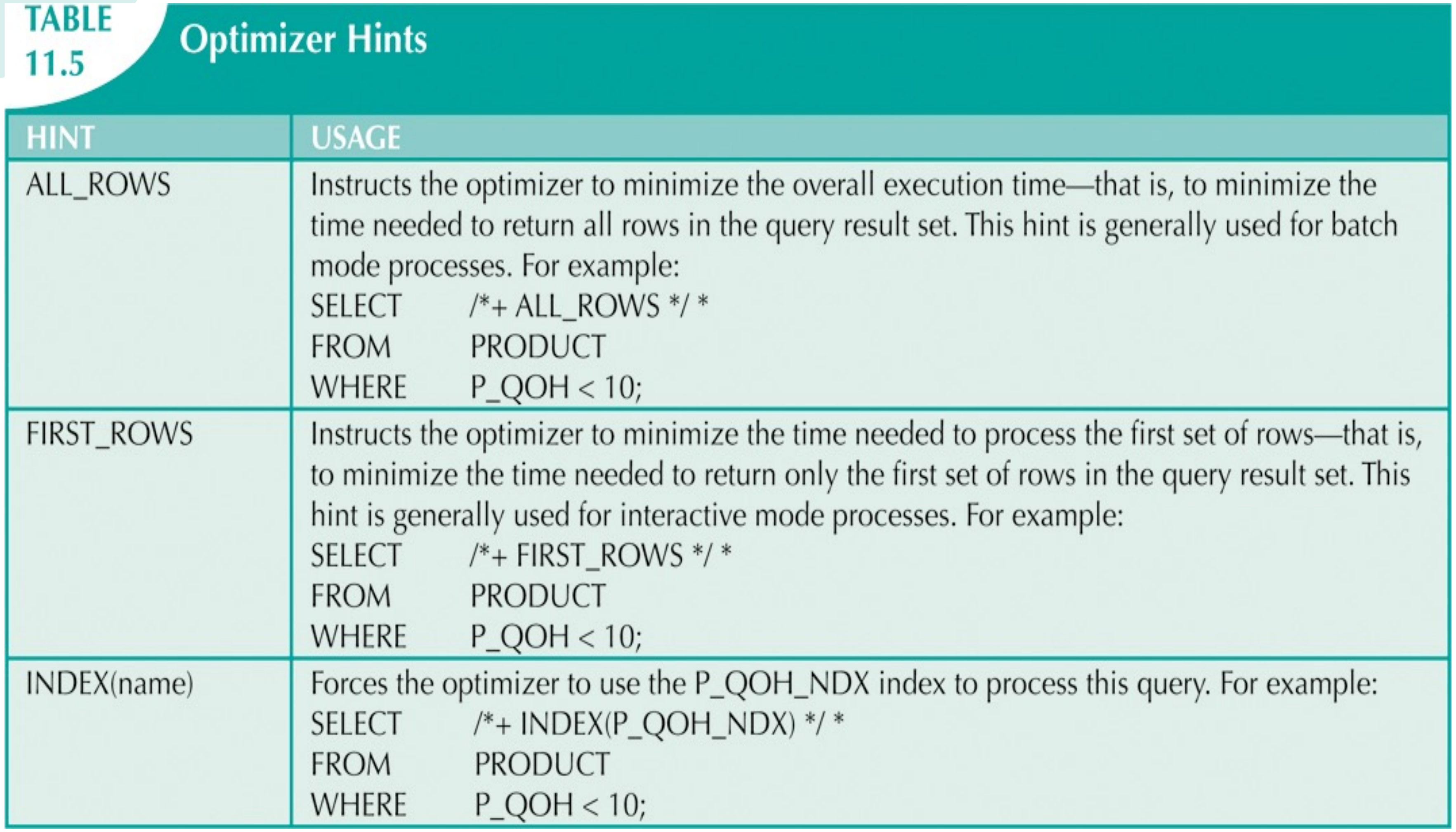


TABLE 11.5 Optimizer Hints

HINT	USAGE

ALL_ROWS	Instructs the optimizer to minimize the overall execution time—that is, to minimize the time needed to return all rows in the query result set. This hint is generally used for batch mode processes. For example:
	SELECT /*+ ALL_ROWS */ *
	FROM PRODUCT
	WHERE P_QOH < 10;

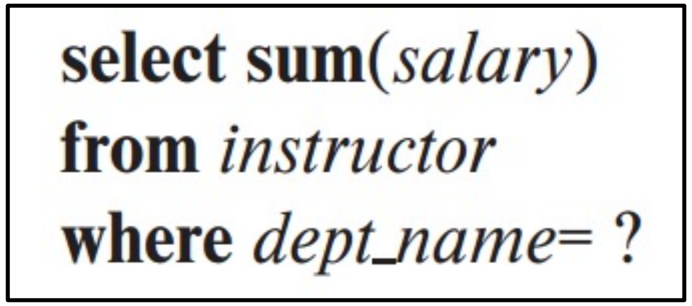
FIRST_ROWS	Instructs the optimizer to minimize the time needed to process the first set of rows—that is, to minimize the time needed to return only the first set of rows in the query result set. This hint is generally used for interactive mode processes. For example:
	SELECT /*+ FIRST_ROWS */ *
	FROM PRODUCT
	WHERE P_QOH < 10;

INDEX(name)	Forces the optimizer to use the P_QOH_NDX index to process this query. For example:
	SELECT /*+ INDEX(P_QOH_NDX) */ *
	FROM PRODUCT
	WHERE P_QOH < 10;

KNOWLEDGE & SOFTWARE ENGINEERING

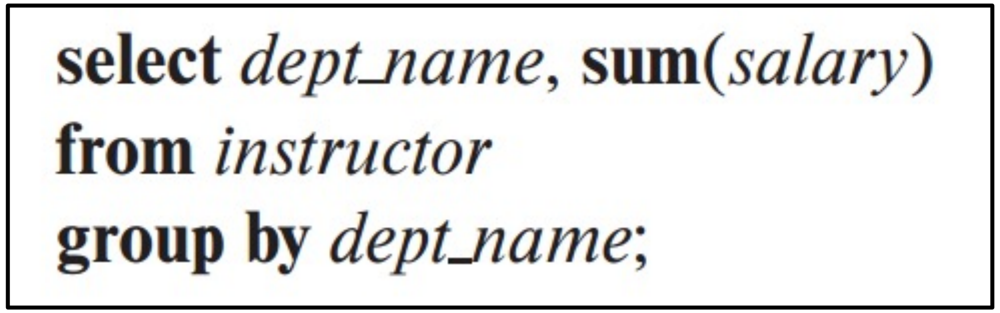
Tuning of Queries: Set Orientation

- In an application program, it is often that a query is executed frequently, but with different values for a parameter.
- **Set orientation** → performing fewer calls to database



```
select sum(salary)
from instructor
where dept_name= ?
```

Instead of this →



```
select dept_name, sum(salary)
from instructor
group by dept_name;
```

use this

Transaction Tuning

KNOWLEDGE & SOFTWARE ENGINEERING

Tuning of Transactions

- Long transactions (typically **read-only**) that examine large parts of a relation **result in lock contention** with update transactions
  - **E.g.** large query to compute bank statistics and regular bank transactions
- To reduce contention
  - Use multi-version concurrency control

    without lock

    - E.g. Oracle "**snapshots**" which support multi-version 2PL
  - Use degree-two consistency (cursor-stability) for long transactions
    - **Drawback:** result may be approximate

KNOWLEDGE & SOFTWARE ENGINEERING

Tuning of Transactions (Cont.)

- **Long update transactions cause several problems**
  - Exhaust lock space
  - Exhaust log space
    - and also greatly increase recovery time after a crash, and may even exhaust log space during recovery if recovery algorithm is badly designed!
- Use **mini-batch** transactions to limit number of updates that a single transaction can carry out.
  - Split large transaction into batch of "mini-transactions", each performing part of the updates
  - In case of failure during a mini-batch, must complete its remaining portion on recovery, to ensure atomicity.

Hardware Tuning

Tuning of Hardware

- Even well-tuned transactions typically require a few I/O operations
  - Typical disk supports about **100 random I/O** operations per second
  - Suppose each transaction requires just **2 random I/O operations.** Then to support ***n* transactions per second**, we need to **stripe data** across **n/50 disks** (ignoring skew)
- Number of I/O operations per transaction can **be reduced** by **keeping more data in memory**
  - If all data is in memory, I/O needed only for writes
  - Keeping frequently used data in memory reduces disk accesses, reducing number of disks required, but has a memory cost

Hardware Tuning: Which Data to Keep in Memory?

- Economic consideration is popularly used to answer question above, which is **finding the break even point** between **disk access cost** and **memory cost**.
- Disk access cost
  - If a block is accessed once in ***m*** second, then:

$$\textit{disk access cost} = \frac{\textit{price per disk}}{\textit{number of access per second} \times m}$$

- Memory cost by keeping a block of data

$$\textit{memory cost} = \frac{\textit{price per MB of memory}}{\textit{number of blocks per MB}}$$

Hardware Tuning: Which Data to Keep in Memory?

- In 1987, price per disk = \$30,000 with 15 random access/s, while price per 1MB memory = \$5,000 with block size = 1KB.
- Suppose we have blocks of data that are **accessed once per 2 seconds**, then:
  - $\textit{disk access cost} = \frac{price\ per\ disk}{number\ of\ access\ per\ second \times m} = \frac{\$30{,}000}{15\ access/s \times 2s} = \$1{,}000\ per\ block\ access$
  - $\textit{memory cost} = \frac{price\ per\ MB\ of\ memory}{number\ of\ blocks\ per\ MB} = \frac{\$5{,}000}{1{,}000\ blocks} = \$5\ per\ block$
- Buffering this data:
  - Needs \$5 memory cost, reduces \$1,000 disk access cost → **WORTH!**
- To find the break even point of ***m***, thus:

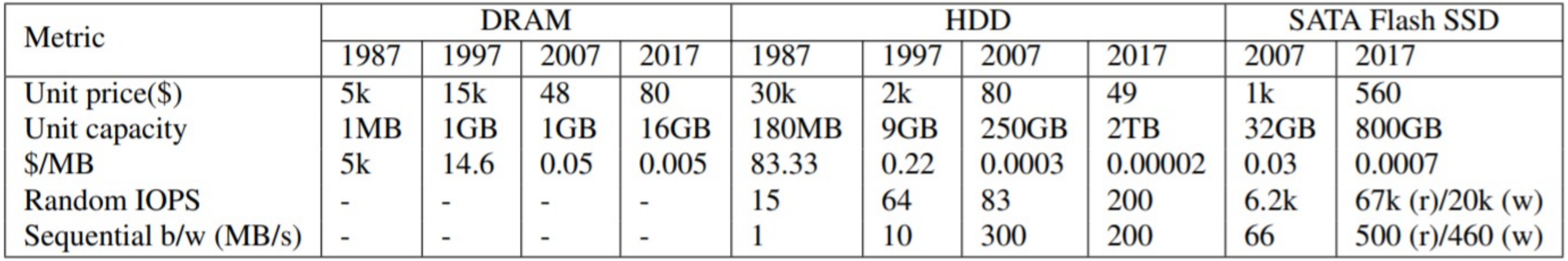
$$\boldsymbol{m} = \frac{price\ per\ disk}{price\ per\ MB\ memory} \times \frac{numb.\ blocks\ per\ MB}{numb.\ disk\ acess} = \frac{30{,}000}{5{,}000} \times \frac{1{,}000}{15} = 400\ s\ \approx 5\ mins.$$

Thus, in that time, data accessed every 400s or less is worth to buffer in the memory → buy more memory!

Hardware Tuning: Five-Minute Rule

- Using formula and data table below, we can get **m** from time to time:

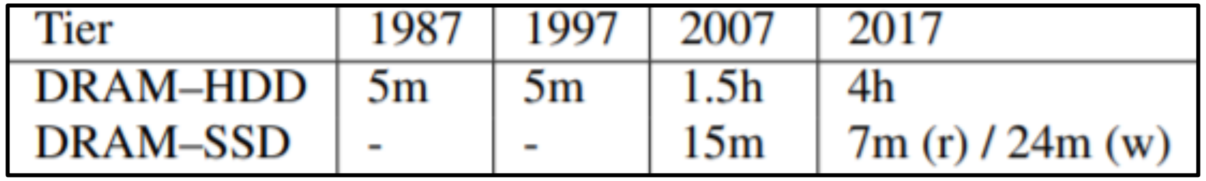
$$\boldsymbol{m} = \frac{price\ per\ disk}{price\ per\ MB\ memory} \times \frac{numb.blocks\ per\ MB}{numb.disk\ acess}$$



Metric	DRAM				HDD				SATA Flash SSD	
	1987	1997	2007	2017	1987	1997	2007	2017	2007	2017
Unit price($)	5k	15k	48	80	30k	2k	80	49	1k	560
Unit capacity	1MB	1GB	1GB	16GB	180MB	9GB	250GB	2TB	32GB	800GB
$/MB	5k	14.6	0.05	0.005	83.33	0.22	0.0003	0.00002	0.03	0.0007
Random IOPS	-	-	-	-	15	64	83	200	6.2k	67k (r)/20k (w)
Sequential b/w (MB/s)	-	-	-	-	1	10	300	200	66	500 (r)/460 (w)

Source: Appuswamy, et. all. "The five minute rule thirty years later...." 2017.

In 1987, *m* = 400s ≈ 5 minutes
→ **five minute rule!**



Tier	1987	1997	2007	2017
DRAM–HDD	5m	5m	1.5h	4h
DRAM–SSD	-	-	15m	7m (r) / 24m (w)

Hardware Tuning: One-Minute Rule

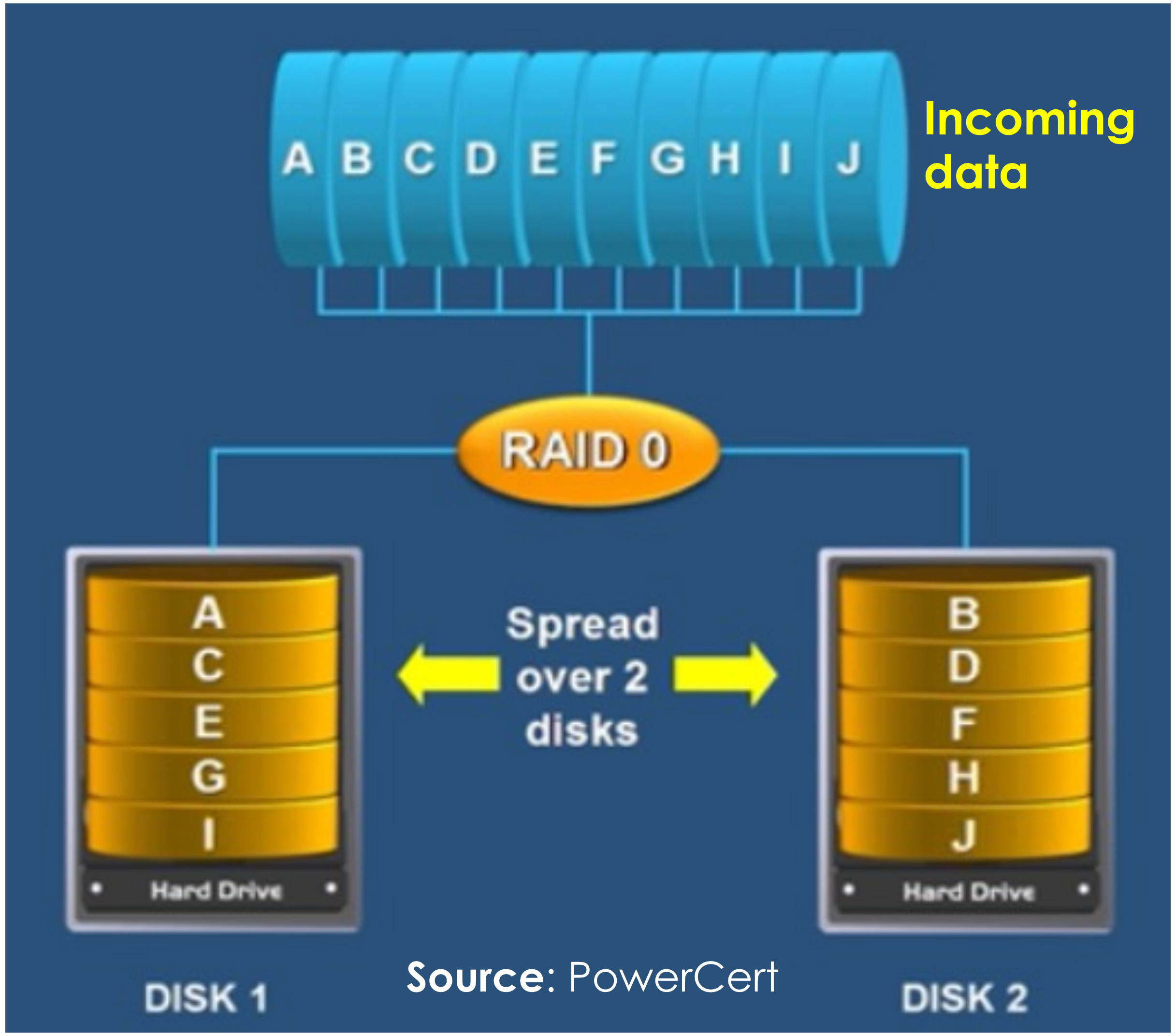
- Prices of disk and memory have changed greatly over the years, but the ratios have not changed much from 1987 to 1997, so during this period:
  - Rules remain as **5 minute**, not 1 hour or 1 second rules (for random access)
  - But rules change after that period of time, e.g., in 2007 and 2017 which are 1.5 hours and **4 hours** rules!
- **For sequentially accessed data**, more blocks can be read per second. Assuming sequential reads of 1MB of data at a time:
  1-minute rule: sequentially accessed data that is accessed once or more in a minute should be kept in memory

Hardware Tuning: RAID

- Redundant Array of Independent Disc (RAID):
  - Disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - **high capacity** and **high speed** by using multiple disks in parallel,
    - **high reliability** by storing data redundantly, so that data can be recovered even if a disk fails
  - **Stripping** – strip data across multiple disk to increase transfer rate (bit-level stripping and block level stripping)
  - **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure

Hardware Tuning: RAID 0

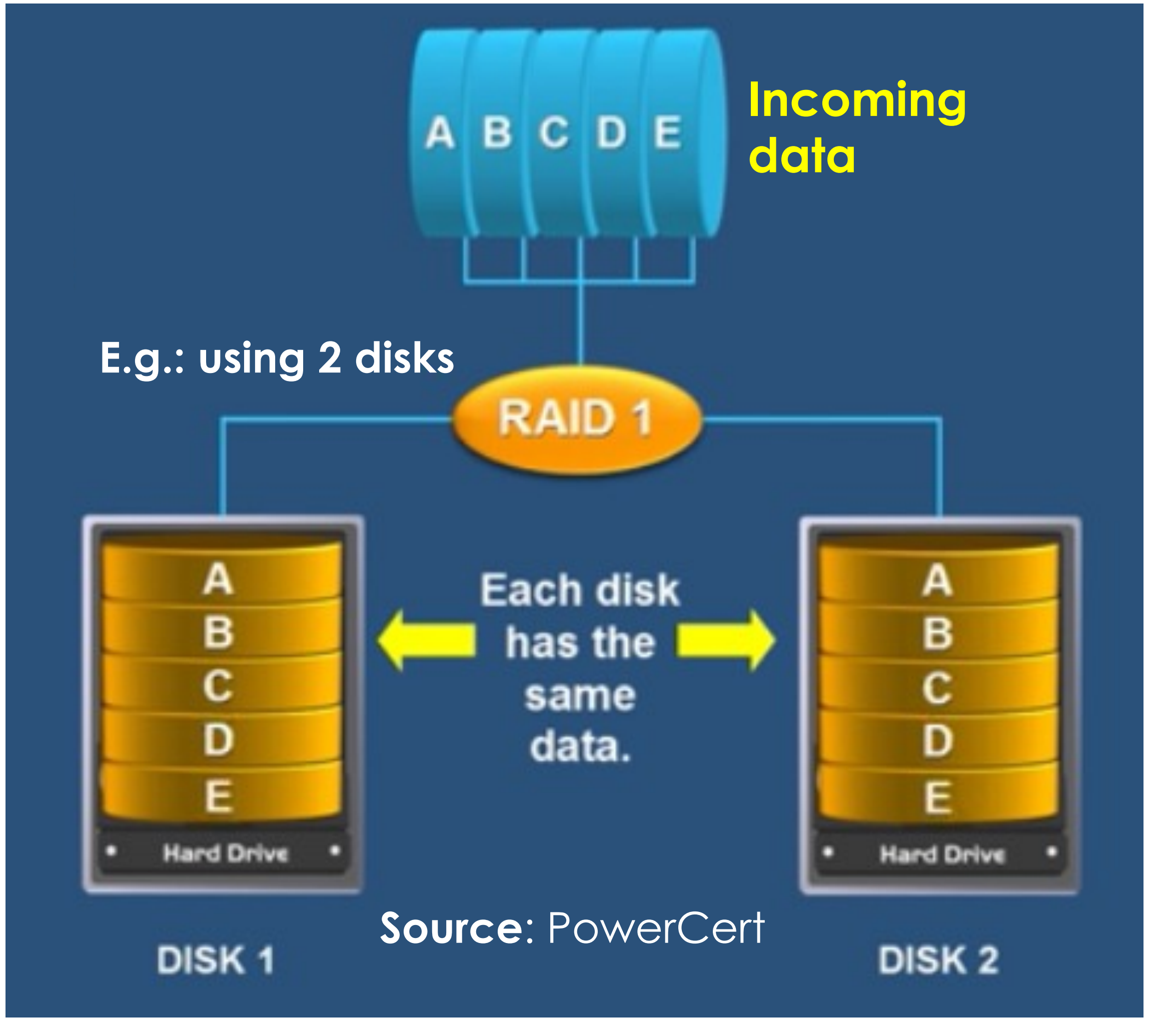
- Benefit:
  - Strip the data across multiple disk → **fast read**
- Drawback:
  - **No fault tolerance** → if a disk fails, the data is lost



Source: PowerCert

Hardware Tuning: RAID 1

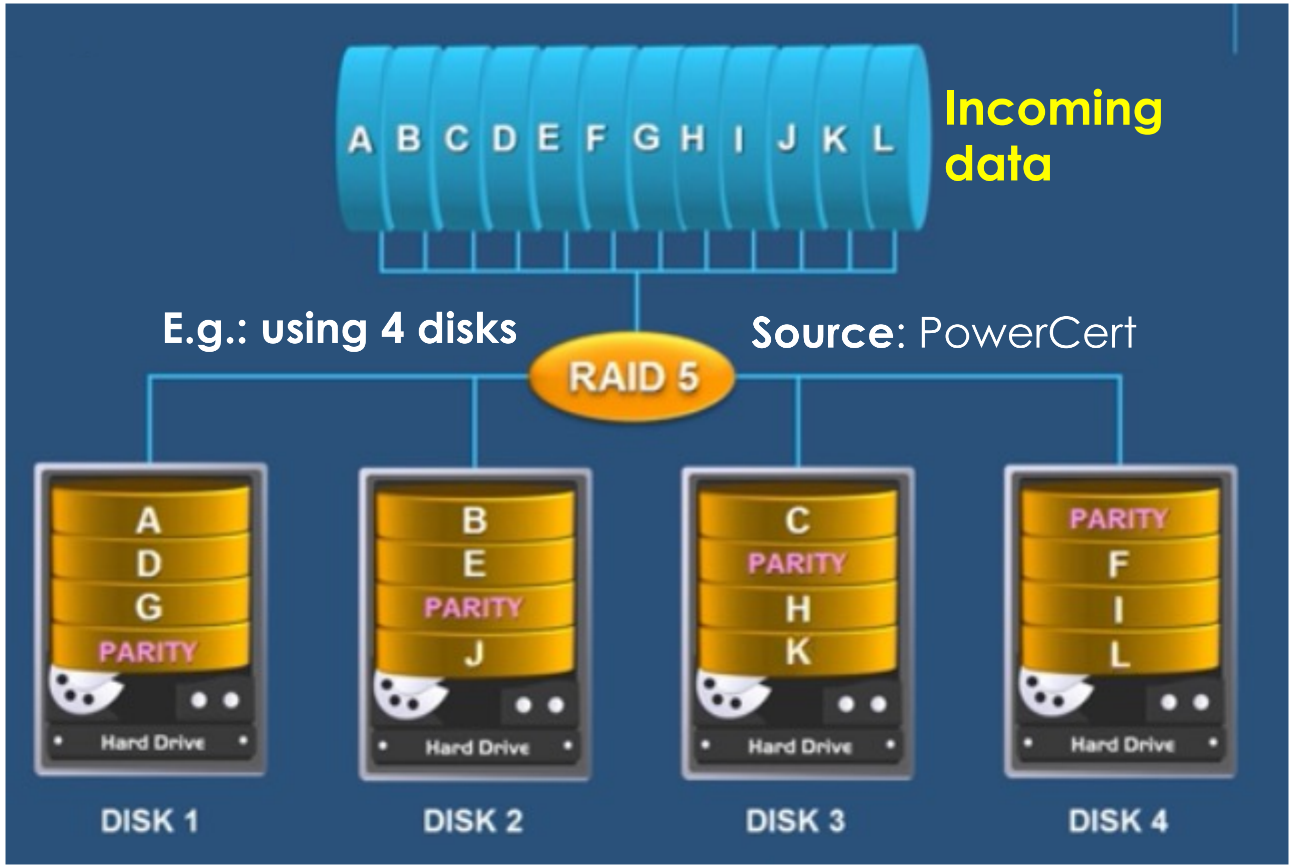
- Benefit:
  - Copy (mirror) the data across multiple disk → **fault tolerant**
  - Relatively easy to rebuild upon the failure
- Drawback:
  - **Need more disk** to store the data
    - **E.g.**, for 1 data copy, only 50% disk for the actual data



Source: PowerCert

Hardware Tuning: RAID 5

- Benefit:
  - Store 1 parity data → **fault tolerant** up to 1 failure
- Drawback:
  - **Need more disk** to store the data
    - **E.g.**, for 4 disks setup, only 75% disk for the actual data
  - More cost to write, and rebuild upon failure
- **Raid 6**: Similar to raid 5, but store more redundancies

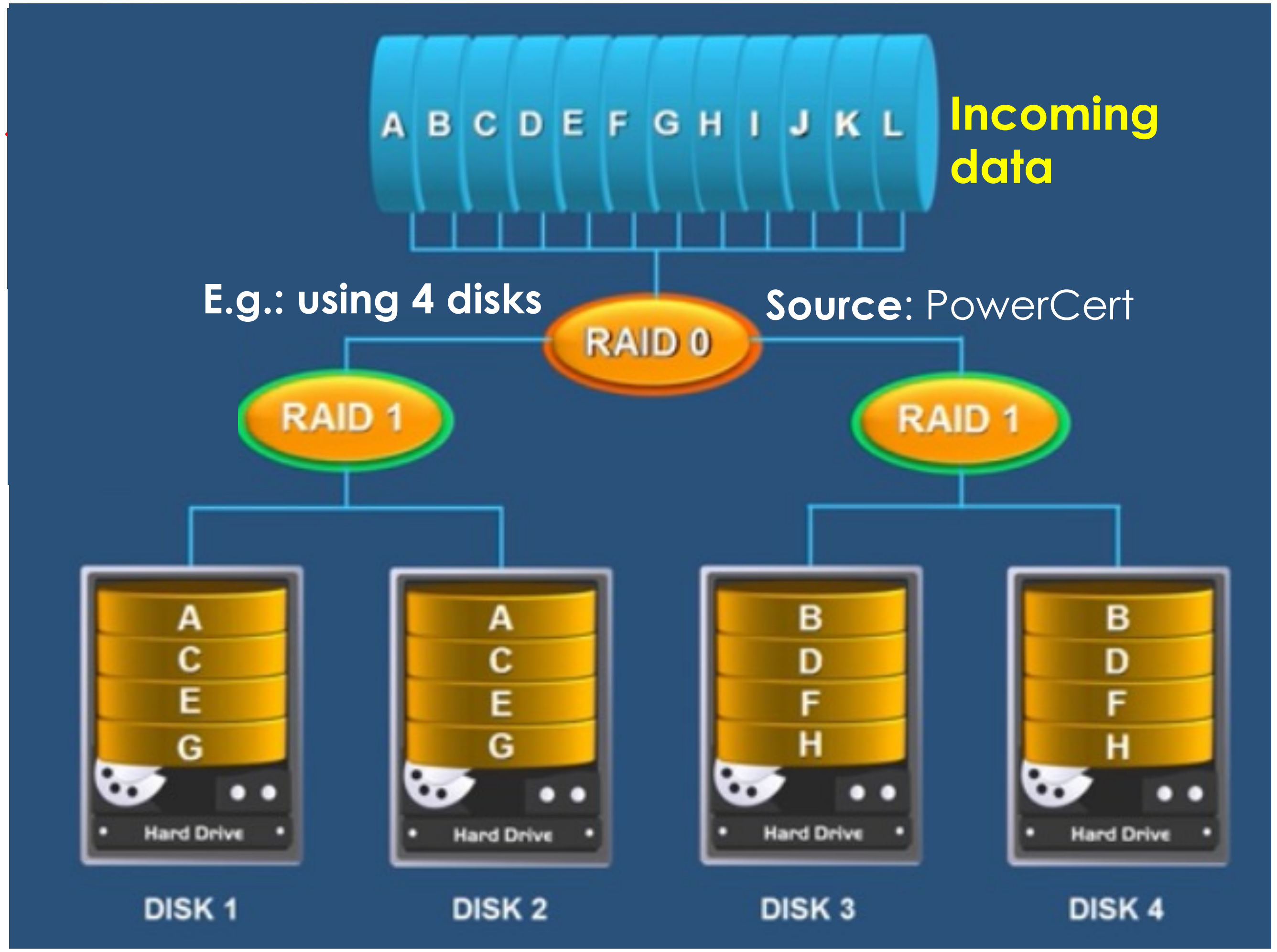


E.g.: using even parity D1, D2, D3, D4 = 0,1,1,0. Any disk failure, we still can recover.

Hardware Tuning: RAID 10

- Benefit:
  - Combine raid 0 (stripping) and raid 1 (redundancy) → **fast read and fault tolerance**
- Drawback:
  - **Need more disk** (at least 4) to store the data
    - **E.g.**, for 4 disks setup, only 50% disk for the actual data

Note: Raid 2, 3, 4 are not used in practice anymore.



Hardware Tuning: Choice of RAID Level

- To use RAID 1 or RAID 5?
  - Depends on ratio of reads and writes
    - RAID 5 requires 2 block reads and 2 block writes to write out one data block, meanwhile RAID 1 only needs 2 block writes
- If an application requires **r reads** and **w writes** per second
  - **RAID 1** requires **r + 2w I/O** operations per second
  - **RAID 5** requires **r + 4w I/O** operations per second
- For reasonably large r and w, this requires lots of disks to handle workload
  - RAID 5 may require more disks than RAID 1 to handle load!
  - Apparent saving of number of disks by RAID 5 (by using parity, as opposed to the mirroring done by RAID 1) may be illusory!
- **Rule of Thumb:** RAID 5 is fine when writes are rare and data is very large, but RAID 1 is preferable otherwise

Performance Simulation

- Performance simulation using queuing model useful to predict bottlenecks as well as the effects of tuning changes, even without access to real system
- Simulation model is quite detailed, but usually omits some low level details
  - Model service time, but disregard details of service
  - **E.g.** approximate disk read time by using an average disk read time
- Experiments can be run on model, and provide an estimate of measures such as average throughput/response time
- Parameters can be tuned in model and then replicated in real system
  - **E.g.** number of disks, memory, algorithms, etc.

Performance Benchmark

KNOWLEDGE & SOFTWARE ENGINEERING

Performance Benchmarks

- Suites of tasks used to quantify the performance of software systems
- Important in comparing database systems, especially as systems become more standards compliant.
- Commonly used performance measures:
  - Throughput (transactions per second, or tps)
  - Response time (delay from submission of transaction to return of result)
  - Availability or mean time to failure

Performance Benchmarks (Cont.)

- Beware when computing average throughput of different transaction types
  - **E.g.**, suppose a system runs transaction type **A at 99 tps** and transaction **type B at 1 tps**.
  - Given an equal mixture of types A and B, throughput **is not (99+1)/2 = 50 tps**.
  - Running one transaction of each type takes time 0.01+1=1.01 seconds, giving a throughput of **1.98 tps**.
  - To compute average throughput of $\boldsymbol{n}$ transactions with tps of $\boldsymbol{t_i}$, use **harmonic mean**:

$$\frac{\boldsymbol{n}}{\boldsymbol{1/t_1 + 1/t_2 + \cdots + 1/t_n}}$$

  - **Interference** (e.g. lock contention) makes even this incorrect if different transaction types run concurrently

KNOWLEDGE & SOFTWARE ENGINEERING

Database Application Classes

- **Online transaction processing (OLTP)**
  - requires high concurrency and clever techniques to speed up commit processing, to support a high rate of update transactions.
- **Decision support applications**
  - including online analytical processing, or OLAP applications
  - require good query evaluation algorithms and query optimization.
- Architecture of some database systems tuned to one of the two classes
  - E.g. Teradata is tuned to decision support
- Others try to balance the two requirements
  - E.g. Oracle, with snapshot support for long read-only transaction

KNOWLEDGE & SOFTWARE ENGINEERING

Benchmarks Suites

- The **Transaction Processing Council** (**TPC**) benchmark suites are widely used.
  - **TPC-A** and **TPC-B**: simple **OLTP application** modeling a bank teller application with and without communication
    - Not used anymore
  - **TPC-C**: complex OLTP application modeling an inventory system
    - Current standard for OLTP benchmarking

Benchmarks Suites (Cont.)

- TPC benchmarks (cont.)
  - **TPC-D**: complex **decision support application** with 17 queries
    - Refined by TPC-H and TPC-R
  - **TPC-H**: (H for ad hoc) based on TPC-D with some extra queries
    - Models ad hoc queries which are not known beforehand
      - Total of 22 queries with emphasis on aggregation
    - prohibits materialized views
    - permits indices only on primary and foreign keys
  - **TPC-R**: (R for reporting) same as TPC-H, but without any restrictions on materialized views and indices
  - **TPC-W**: (W for Web) End-to-end Web service benchmark modeling a Web bookstore, with combination of static and dynamically generated pages

KNOWLEDGE & SOFTWARE ENGINEERING

TPC Performance Measures

- TPC performance measures
  - **transactions-per-second** with specified constraints on response time
  - **transactions-per-second-per-dollar** accounts for cost of owning system
- TPC benchmark requires database sizes to be scaled up with increasing transactions-per-second
  - reflects real world applications where more customers means more database size and more transactions-per-second
- External audit is mandatory to claim TPC performance numbers
  - TPC performance claims can be trusted

KNOWLEDGE & SOFTWARE ENGINEERING

TPC Performance Measures

- Two types of tests for TPC-H and TPC-R
  - **Power test**: runs queries and updates sequentially, then takes mean to find queries per hour
  - **Throughput test**: runs queries and updates concurrently
    - multiple streams running in parallel each generates 22 queries, with one parallel update stream
  - **Composite query per hour metric**: square root of product of power and throughput metrics
  - **Composite price/performance metric**: dividing the system price by the metric

End of Chapter

KNOWLEDGE & SOFTWARE ENGINEERING