Physical DB design and tuning: outline

- Designing the Physical Database Schema
  - Tables, indexes, logical schema
- Database Tuning
  - Index Tuning
  - Query Tuning
  - Transaction Tuning
  - Logical Schema Tuning
- DBMS Tuning

Relational DB design

- Database design phases:
  - (a) Requirement Analysis,
  - (b) Conceptual design
  - (c) Logical design
  - (d) Physical design
- Physical Design Goal: definition of appropriate storage structures for a specific DBMS, to ensure the application performance desired

Physical design: needed information

- Logical relational schema and integrity constraints
- Statistics on data (table size and attribute values)
- Workload
    - Critical queries and their frequency of use
    - Critical updates and their frequency of use
    - Performance expected for critical operations
- Knowledge about the DBMS
    - Data organizations,
    - Indexes and
    - Query processing techniques supported

Workload definition

- Critical operations: those performed more frequently and for which a short execution time is expected.

Workload definition

- For each critical query
  - Relations used
  - Attributes of the result
  - Attributes used in conditions (restrictions and joins)
  - Selectivity factor of the conditions
- For each critical update
  - Type (INSERT/DELETE/UPDATE)
  - Attributes used in conditions (restrictions and joins)
  - Selectivity factor of the conditions
  - Attributes that are updated

The use of ISUD table

Insert, **S**elect, **U**pdate, **D**elete

<---- Employee Attributes ------>

APPLICATION	FREQUENCY	%DATI	NAME	SALARY	ADDRESS
Payroll	monthly	100	S	S	S
NewEmployee	quartly	0.1	I	I	I
DeleteEmployee	quartly	0.1	D	D	D
UpdateSalary	monthly	10	S	U	

Decisions to be taken

- The physical organization of relations
- Indexes
- Index type
- ... Logical schema transformation to improve performance

Decisions for relations and indexes

- Storage structures for relations:
  - heap (small data set, scan operations, use of indexes)
  - sequential (sorted static data)
  - hash (key equality search), usually static
  - tree (index sequential) (key equality and range search)
- Choice of secondary index, considering that
  - they are extremely useful
  - slow down the updated of the index keys
  - require memory

How to choose indexes

- Use a DBMS application, such as DB2 Design Advisor, SQL Server DB Tuning Advisor, Oracle Access Advisor.

How to choose indexes

- Tips: don’t use indexes
  - on small relations,
  - on frequently modified attributes,
  - on non selective attributes (queries which returns ≥ 15% of data)
  - on attributes with values long string
- Define indexes on primary and foreign keys
- Consider the definition of indexes on attributes used on queries which requires sorting: ORDER BY, GROUP BY, DISTINCT, Set operations

How to choose indexes (cont.)

- Evaluate the convenience of indexes on attributes that can be used to generate index-only plans.
- Evaluate the convenience of indexes on selective attributes in the WHERE:
  - hash indexes, for equality search
  - B^+tree, for equality and range search, possibly clustered
  - multi-attributes (composite), for conjunctive conditions
- to improve joins with IndexNestedLoop or MergeJoin, grouping, sorting, duplicate elimination.
- Attention to disjunctive conditions

How to choose indexes (cont.)

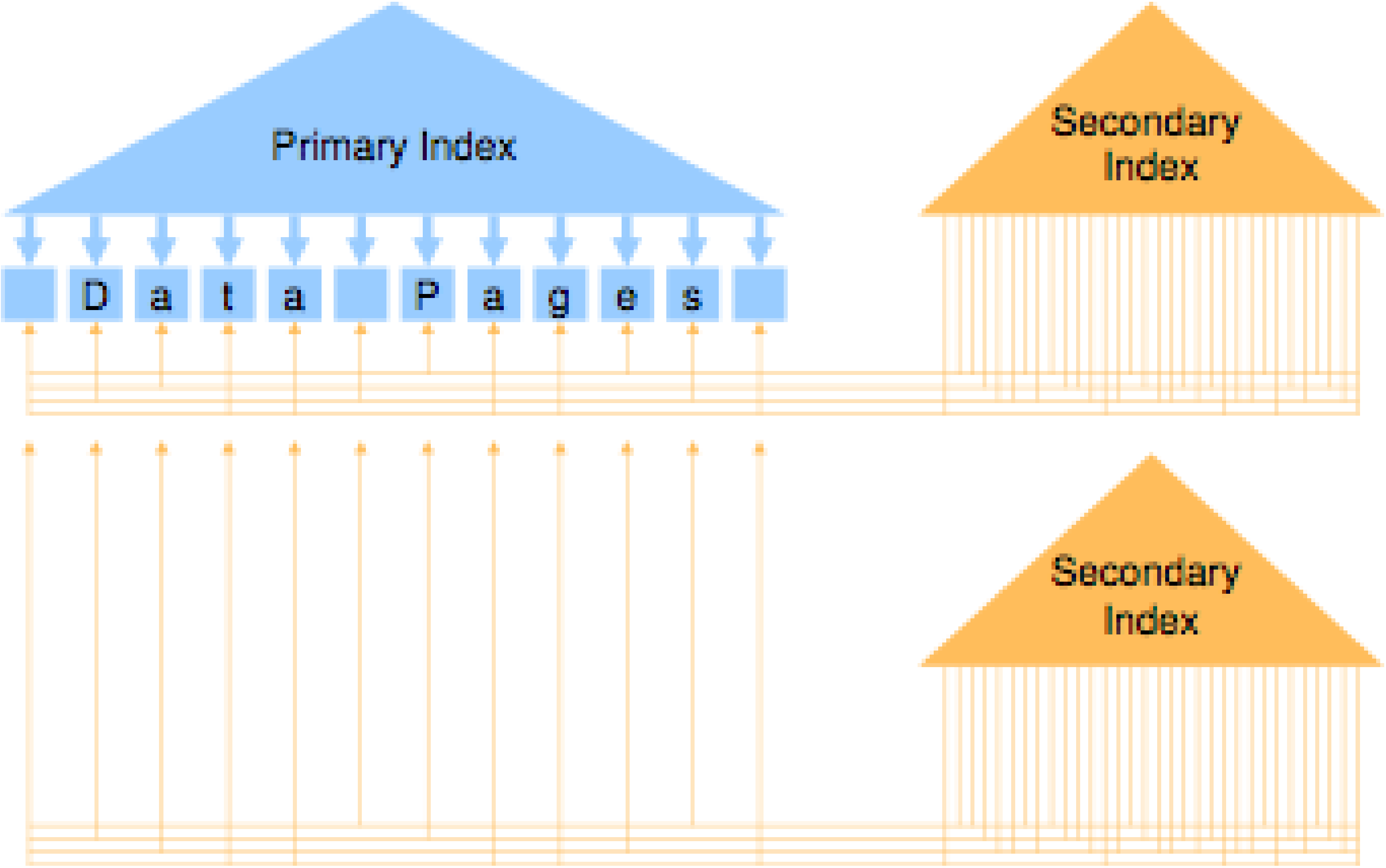
- Local view: indexes useful for one query (simple)

- Global view: indexes useful for a workload
- Index subsumption: some indexes may provide similar benefit
  - An index Idx1(a, b, c) can replace indexes Idx2(a, b) and Idx3(a)
- Index merging: two indexes supporting two different queries can be merged into one index supporting both queries.
- When there are a lot of data to load, create indexes after data loading

How to choose indexes: the global view

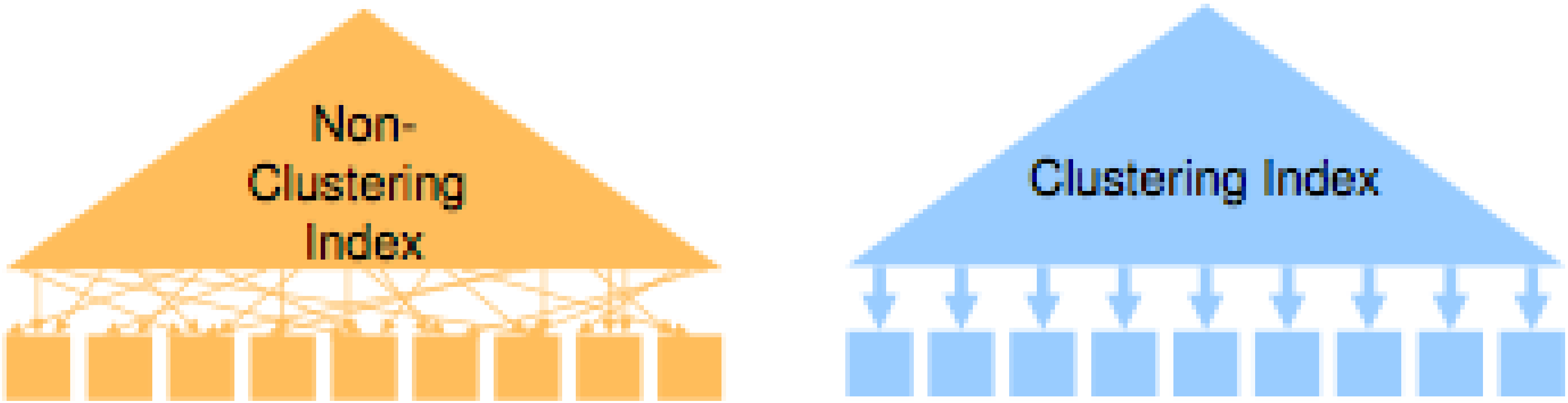
1. Identify critical queries
2. Create possible indexes to tune single queries
3. From set of all indexes remove subsumed indexes
4. Merge indexes where possible indexes
5. Evaluate benefit/cost ratio for remaining indexes
   (need to consider frequency of queries/index usage)
6. Pick optimal index configuration satisfying storage constraints.

Primary organizations and secondary indexes

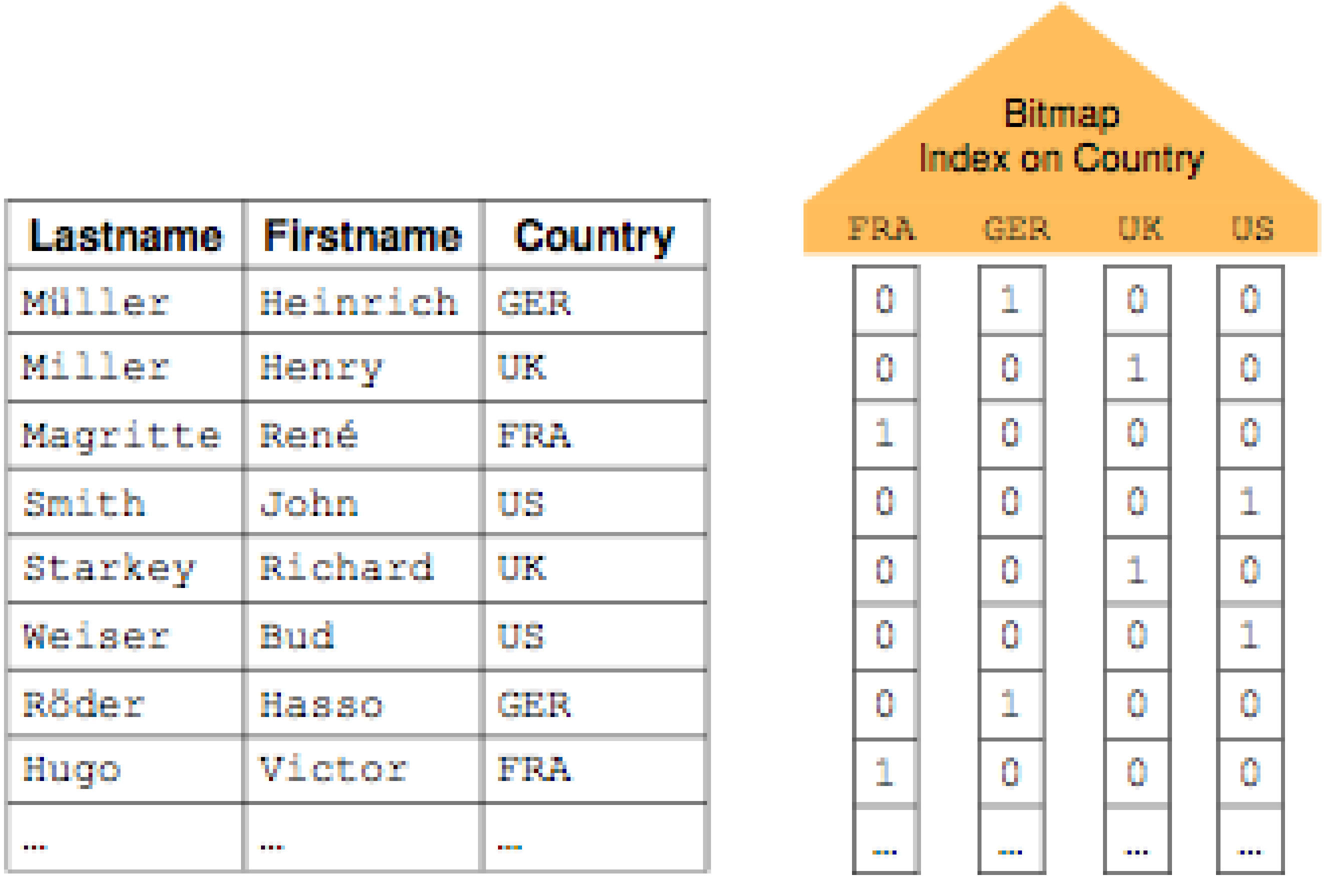


```
CREATE [UNIQUE...] INDEX Nome [ USING {BTREE | HASH | RTREE} ]
      ON Table (Attributes+Ord)
```

Clustering indexes



Bitmap indexes



Lastname	Firstname	Country
Müller	Heinrich	GER
Miller	Henry	UK
Magritte	René	FRA
Smith	John	US
Starkey	Richard	UK
Weiser	Bud	US
Röder	Hasso	GER
Hugo	Victor	FRA
...	...	...

Bitmap
Index on Country

FRA	GER	UK	US
0	1	0	0
0	0	1	0
1	0	0	0
0	0	0	1
0	0	1	0
0	0	0	1
0	1	0	0
1	0	0	0
...	...	...	...

DB schema for the examples

Lecturers(PkLecturer INT, Name VARCHAR(20), ResearchArea VARCHAR(1), Salary INT, Position VARCHAR(10), FkDepartment INT)

Departments(PkDepartment INT, Name VARCHAR(20), City VARCHAR(10))

	Departments	Lecturers
Nrec	300	10 000
Npag	30	1 200
Nkey(IdxSalary)		50 (min=40, max=160)
Nkey(IdxCity)	15	

Primary organization definitions in SQL

CREATE TABLE Table (Attributes) < physical aspects >;

CREATE TABLE Table (Attributes + keys) **ORGANIZED INDEX**; (ORACLE)

CREATE TABLE Table (Attributes + keys) **ORGANIZE BY DIMENSIONS (Att)**; (DB2)

The definition of indexes

Do not index!

Few records in this table

```
SELECT   Name
FROM      Departments
WHERE    City = ‘PI’
```

Do not index!

Possibly too many updates

```
SELECT   StockName, StockPrice
FROM    Stocks
WHERE    Stockprice > 50
```

The index is created implicitly on a PK

```
SELECT   Name
FROM    Lecturers
WHERE    PkLecturer = 70
```

The definition of (multi-attributes) indexes

How many indexes on a Table ?

A secondary index on **Position** or on **Salary**?

A secondary index on **Position** and another on **Salary**?

A secondary index on <**Position, Salary>**?

How many indexes use the DBMS for AND?

```
SELECT  Name
FROM    Lecturers
WHERE   Position = ‘P’
        AND Salary BETWEEN 50 AND 60
```

An index on **Position, ResearchArea** or **ResearchArea Position** to speed up GBY.

Better on **ResearchArea Position** because it also returns result sorted

```
SELECT  ResearchArea,Position
        COUNT(*)
FROM    Lecturers
GROUP BY Position, ResearchArea
ORDER BY  ResearchArea
```

The creation of clustered index

With a not very selective predicate **Salary**>70, a clustered index may still be useful

```
SELECT   Name
FROM      Lecturers
WHERE    Salary > 70
```

If there are a few lecturers with a **Salary =70**, a clustered index on **Salary** can be more useful than an unclustered one

```
SELECT   Name
FROM     Lecturers
WHERE     Salary = 70
```

A clustered index on **FkDepartment** can be useful here

```
SELECT    FkDepartment, COUNT(*)
FROM       Lecturers
WHERE     Salary > 70
GROUP BY FkDepartment
```

The creation of indexes for index-only plans

For Index-Only plans clustered indexes are not required.

Index on su FkDepartment

```
SELECT  DISTINCT FkDepartment
FROM    Lecturers ;
```

Index on Salary

```
SELECT Salary, COUNT(*)
FROM   Lecturers
GROUP BY Salary;
```

Index on FkDepartment for INL

```
SELECT  D.Name
FROM    Lecturers L, Departments D
WHERE  FkDepartment=PkDepartment;
```

Index on <FkDepartment,Salary>

```
SELECT   FkDepartment, MIN(Salary)
FROM     Lecturers
GROUP BY FkDepartment;
```

Index-only plans

Some DBMSs allow the definition of indexes on some attributes, **and to include** also others which are not part of the index key.

DB2: CREATE UNIQUE INDEX *Name* **ON** *Table* (*Attrs*) **INCLUDE** (*OtherAttrs*);

SQL Server: the clause UNIQUE is optional

Index on **FkDepartment**

INCLUDE Salary

```
SELECT   FkDepartment, MIN(Salary)
FROM     Lecturers
GROUP BY FkDepartment;
```

Concluding remarks

- The implementation and maintenance of a database application to meet specific performance requirements is a very complex task
- Table organizations and index selections are fundamental to improve query performance, but ...
  - Indexes must be really useful because of their memory and update costs
  - An index should be useful for different queries
- Clustered indexes are very useful, but only one for table can be defined
- The order of attributes in multi-attribute (composite) indexes is important

Database tuning: what is the goal?

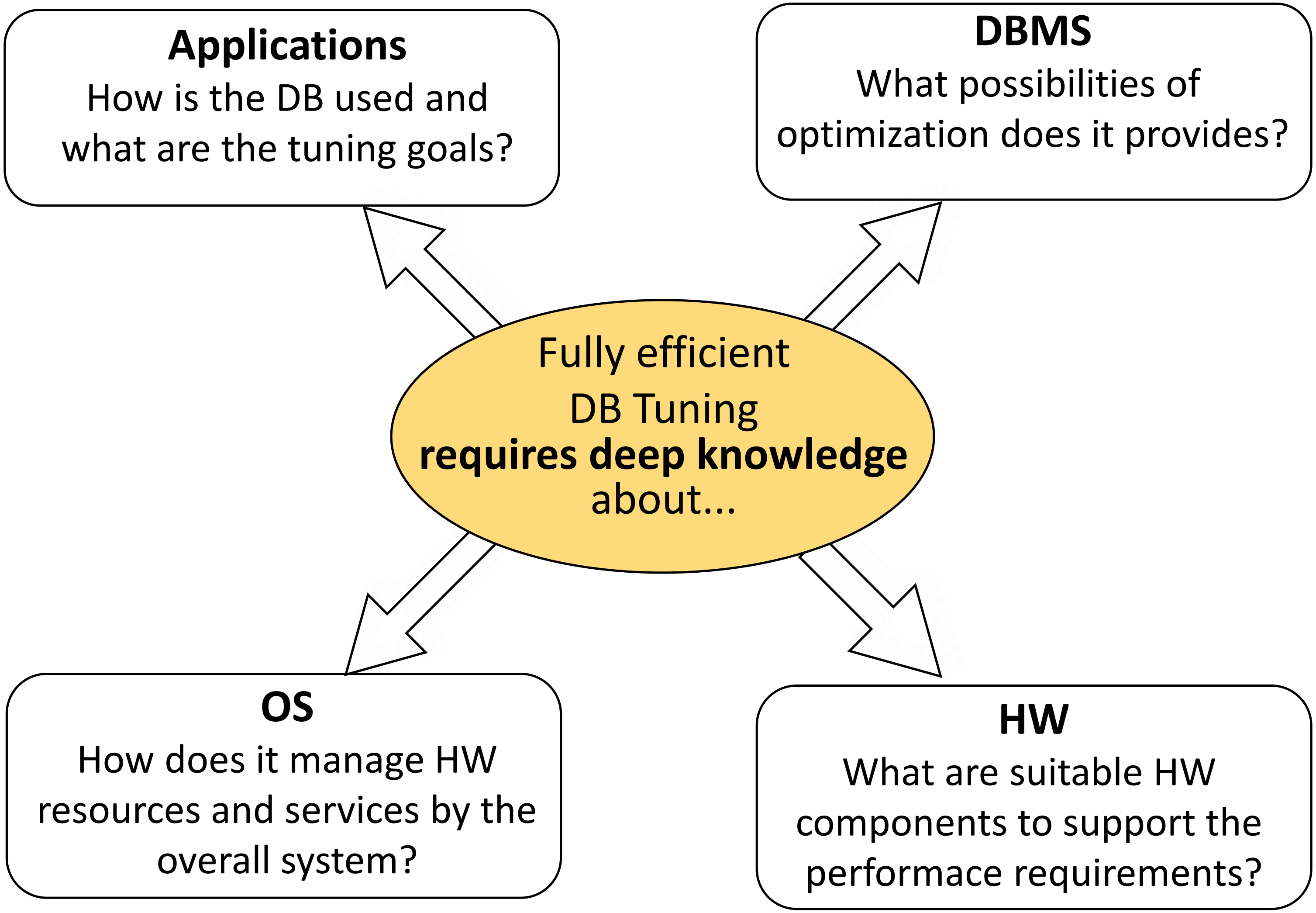
- Improve performance
- Database tuning mostly refers to query (or DB applications) performance.

What can be tuned ?

Applications	Queries, Transactions, ...
DBMS	System configuration, DB schema, storage structures, ...
OS	System parameters and configuration for IO, ...
HW	Components for CPU, main memory, hard disks, backup solutions, network, ...

Here the focus is on Database Tuning

Needed knowledge



Applications
How is the DB used and what are the tuning goals?

DBMS
What possibilities of optimization does it provides?

Fully efficient DB Tuning **requires deep knowledge** about...

OS
How does it manage HW resources and services by the overall system?

HW
What are suitable HW components to support the performace requirements?

Schallehn: Database Tuning and Self-Tuning 2012

Who does the tuning? WHEN? WHAT KNOWLEDGE?

DB and application designers

During DB development (physical DB design) and initial testing.

DB designers must **have knowledge about applications**, **and good knowledge** about the **DBMS**, but may be **only fair to no knowledge** about **OS** and **HW**.

DB administrators (DBA)

During ongoing DBMS maintenance. Adjustment to changing requirements, data volume, new HW.

DBA **have knowledge** about DBMS, OS, and HW. **And about applications ?**

DBA **knowledge** about applications depends on the given organizational structure

---

DB experts
(consultants, in-house experts)

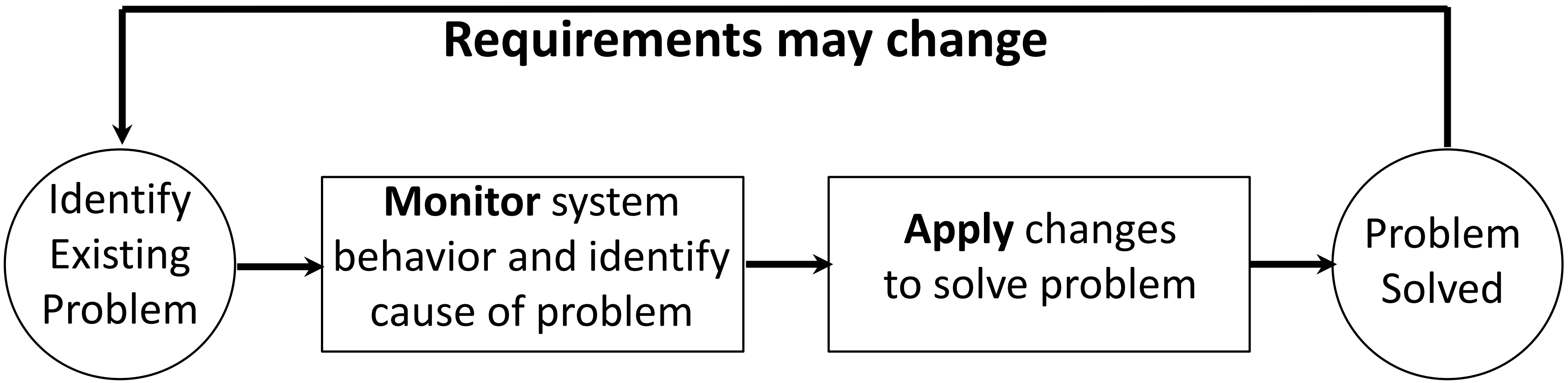
During system re-design, troubleshooting or fire fighting (emergency actions)

DB consultants usually **have strong knowledge** about DBMS, OS and HW, but **have little knowledge** about current applications.

Schallehn: Database Tuning and Self-Tuning 2012

Database tuning as a process

Overall system continuously changes
Data volume, # of user, # of queries,
usage patterns, hardware, etc.



Current performance requirements not fulfilled

Observe and measure relevant quantities, e.g. queries time.

Adjust system parameters, storage structures etc.

Schallehn: Database Tuning and Self-Tuning 2012

Basic principles: controlling trade-offs

Database tuning very often is a process of decision about **costs** for a solution compared to its **benefits**.

Costs: monetary costs (for HW, SW), working hours or more techinical costs (resource consumption, impact on other aspects)

Benefits: improved performance (monetary effects most often not easily quantifiable)

Examples

Adding indexes -> **benefit**: better query performance
costs: more disk memory, more update time

Denormalization -> **benefit**: better query performance
costs: need to control redundancy within tables

Replace disk by RAID -> **benefit**: better I/O performance
costs: HW costs

Schallehn: Database Tuning and Self-Tuning 2012

Basic principles: Pareto principle

80/20 Rule: by applying 20% of the effort one can achieve 80% of the desired effects.

Consequences for DB tuning

100% effect = **full optimized system**

Full optimized system probably beyond necessary requirements.

"**a little bit of DB tuning can help a lot**"

Hence, one does not need to be an expert on all levels of the system to be able to implement a reasonable solution...

Schallehn: Database Tuning and Self-Tuning 2012

Database tuning

When a database has unexpected bad performances we must revise:

Physical Design: the selection of indexes or their type, looking at the access plans generated by the optimizer

Query and Transaction Definitions

DB Logical Design

DBMS: buffer and page size, disk use, log management.

Hardware: number of CPU, disk types.

To begin

- Select the queries with low performance (either the critical one or those which do not satisfy the users)
- Analyze physical query plans looking at
  - physical operators for tables
  - sorting of intermediate results
  - physical operators used in the query plans
  - physical operators for the logical operators

Index tuning

One of the most often applied tuning measures

Great benefits with little effort (if applied correctly).

Strong support within all available DBMS

(index structures, index usage controlled by optimizer)

Storage cost of additional disk memory most often acceptable.

Cost for index updates.

Cost for locking overhead and lock conflicts.

Query tuning

- SELECT with OR condition are rewritten as one predicate IN, or with UNION of SELECT
- SELECT with AND of predicates are rewritten as a condition with BETWEEN
- Rewrite SUBQUERIES as join
- Eliminate useless DISTINCT and ORDER BY from SELECT or SUBSELECTS
- Avoid the definition of temporary views with grouping and aggregations
- Avoid aggregation functions in SUBQUERIES

Query tuning (cont.)

Avoid expressions with index attributes (e.g. Salary*2=100)

Avoid useless HAVING or GROUP BY

```
SELECT   Position, MIN(Salary)
FROM      Lectures
GROUP BY Position
HAVING   Position IN(‘P1’,‘P2”)
```

```
SELECT   Position, MIN(Salary)
FROM      Lectures
WHERE      Position IN(‘P1’,‘P2”)
GROUP BY Position
```

$$\sigma_\phi({}_A\gamma_F(E)) \equiv {}_A\gamma_F(\sigma_\phi(E))$$

```
SELECT    MIN(Salary)
FROM     Lectures
GROUP BY Position
HAVING   Position = ‘P1’
```

```
SELECT    MIN(Salary)
FROM      Lectures
WHERE      Position = ‘P1’
```

Transactions tuning

- Do not block data during db loading, as well as during read-only transactions
- Split complex transactions in smaller ones
- Select the right block granularity, if possible (long T, table; medium T, page; short T, record)
- Select the right isolation level among those provided by SQL
  - SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }

Isolation levels in SQL

READ UNCOMMITTED, **record READ** only without locks

Problem: **dirty read** (read data updated by other active **T**)

Account (No INTEGER PRIMARY KEY, Name CHAR(30), Balance FLOAT);

```
-- T1.begin:
UPDATE Account
SET Balance = Balance - 200.00
WHERE No = 123;
```

```
-- T2.begin: RU


SELECT AVG(Balance)FROM
Account; COMMIT;
```

Isolation levels in SQL

READ COMMITTED, **shared read** locks are released immediately, **exclusive** locks until the T commit

Problem: avoid dirty read, but unrepeatable reads or loss of updates

```
-- T1.begin:

UPDATE Account
SET Balance = Balance - 200.00
WHERE No = 123;   COMMIT;
```

```
-- T2.begin:              RC
SELECT AVG(Balance)
FROM   Account;
```

Isolation levels in SQL

REPEATABLE READ, shared and exclusive locks on records until the end of the transaction

Problem: avoid the previous problems, but not the “**phantom records**” problem:

```
-- T1.begin:
```

```
-- T2.begin:  RR
SELECT AVG(Balance)
FROM   Account;
```

Isolation levels in sql (cont.)

- SERIALIZABLE, multi-granularity locks: tables read by a T cannot be updated.
- Good, but the number of transactions which can be executed concurrently is considerably reduced.

DBMS isolation levels

- Commercial DBMS may
  - provide some isolation levels only,
  - not have the same isolation level by default
  - have other isolation levels (e.g. SNAPSHOT)

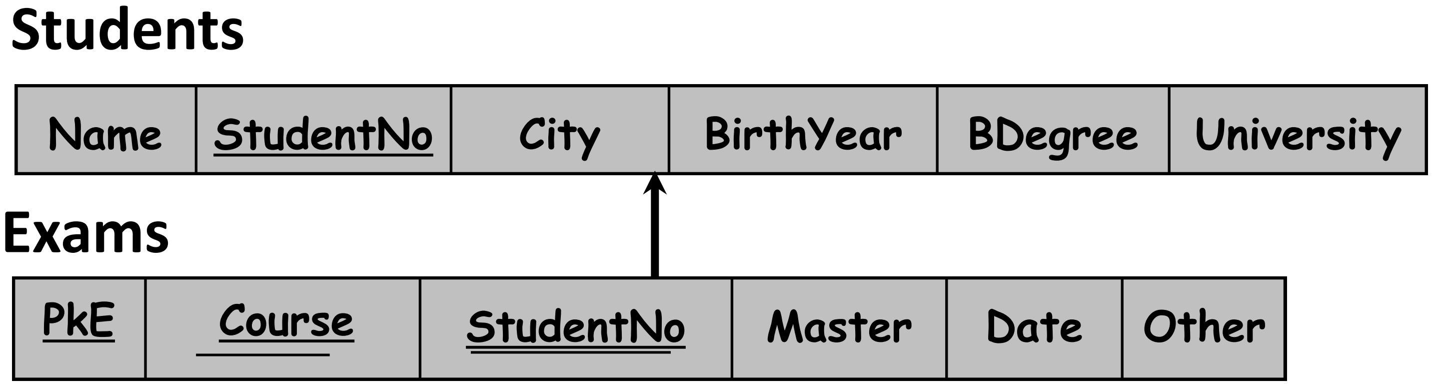
Logical schema tuning

- Types of logical schema restructuring:
  - Vertical Partitioning.
  - Horizontal Partitioning
  - Denormalization
- Unlike changes to the physical schema (physical independence), changes to the logical schema (schema evolution) require views creation for the logical independence.

Logical schema tuning

- Partitioning: splitting a table for performance
  - Horizontal: on a property
  - Vertical: R1(pk, Name, Surname) R2(pk, Address, …)
- Normalization: divide Students from Exams to avoid anomalies
- Denormalization: store Students and Exams into one table:
  - increases update time but makes join faster

Vertical partitioning (projections)



Students

Name	StudentNo	City	BirthYear	BDegree	University

Exams

PkE	Course	StudentNo	Master	Date	Other

Critical Query:

Find the number of exams passed and the number of students who have done the test by course, and by academic year.

ExamForAnalysis

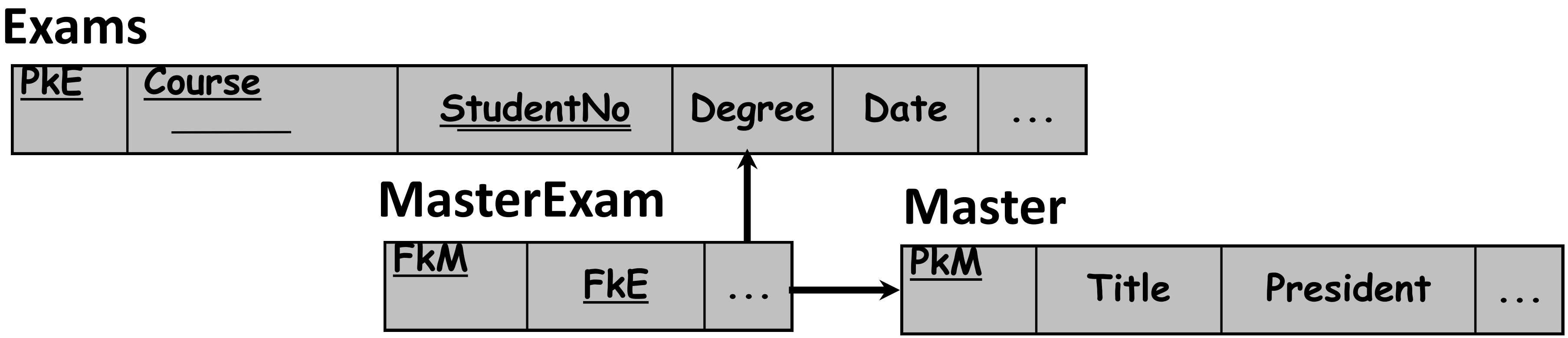
PkE	Course	Master	Date

ExamsOther

PkE	StudentNo	Other

The decomposition must preserve data...

Horizontal partitioning (selections)



Exams

PkE | Course | StudentNo | Degree | Date | ...

MasterExam

FkM | FkE | ...

Master

PkM | Title | President | ...

Critical Query:

For a study program with title **X** and a course with less than 5 exams passed, find the number of exams, by course, and academic year

MasterXExams

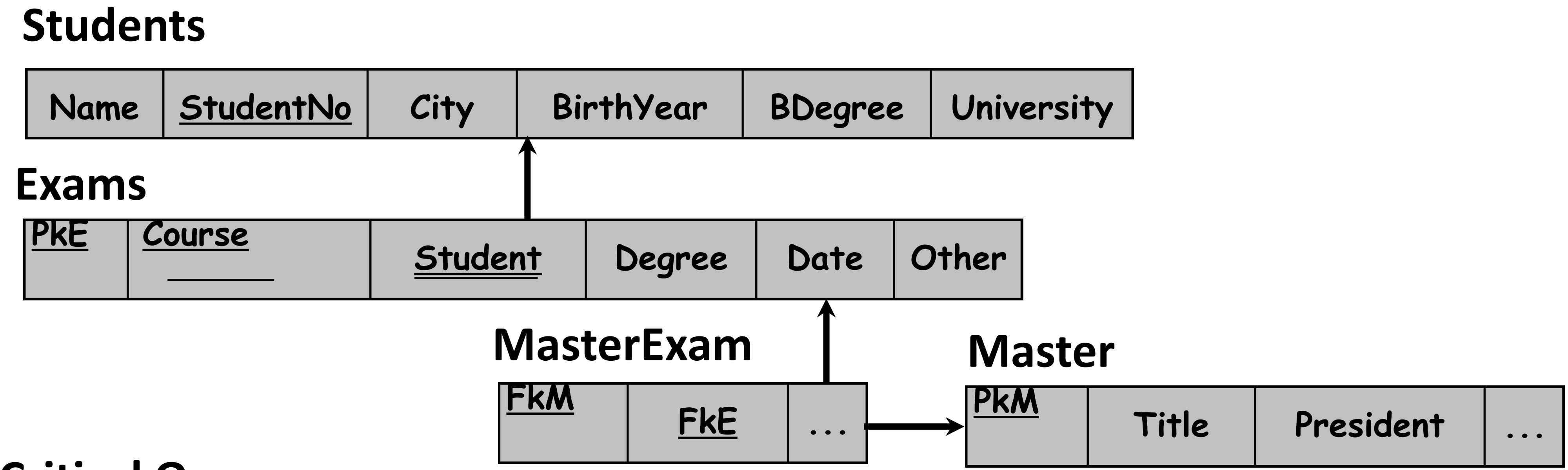
PkE	Course	StudentNo	Degree	Date	...

MasterYExams

PkE	Course	StudentNo	Degree	Date	...

...

Denormalization (attribute replication)



Students

Name	StudentNo	City	BirthYear	BDegree	University

Exams

PkE	Course	Student	Degree	Date	Other

MasterExam

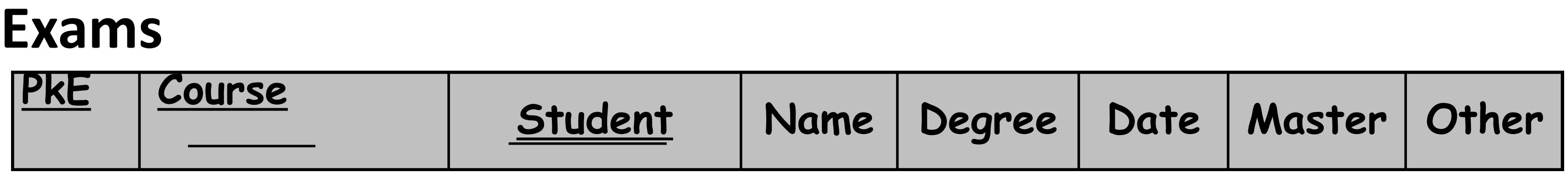
FkM	FkE	. . .

Master

PkM	Title	President	. . .

Critical Query:

For a student number N, find the student name, the master program title and the grade of exams passed



Exams

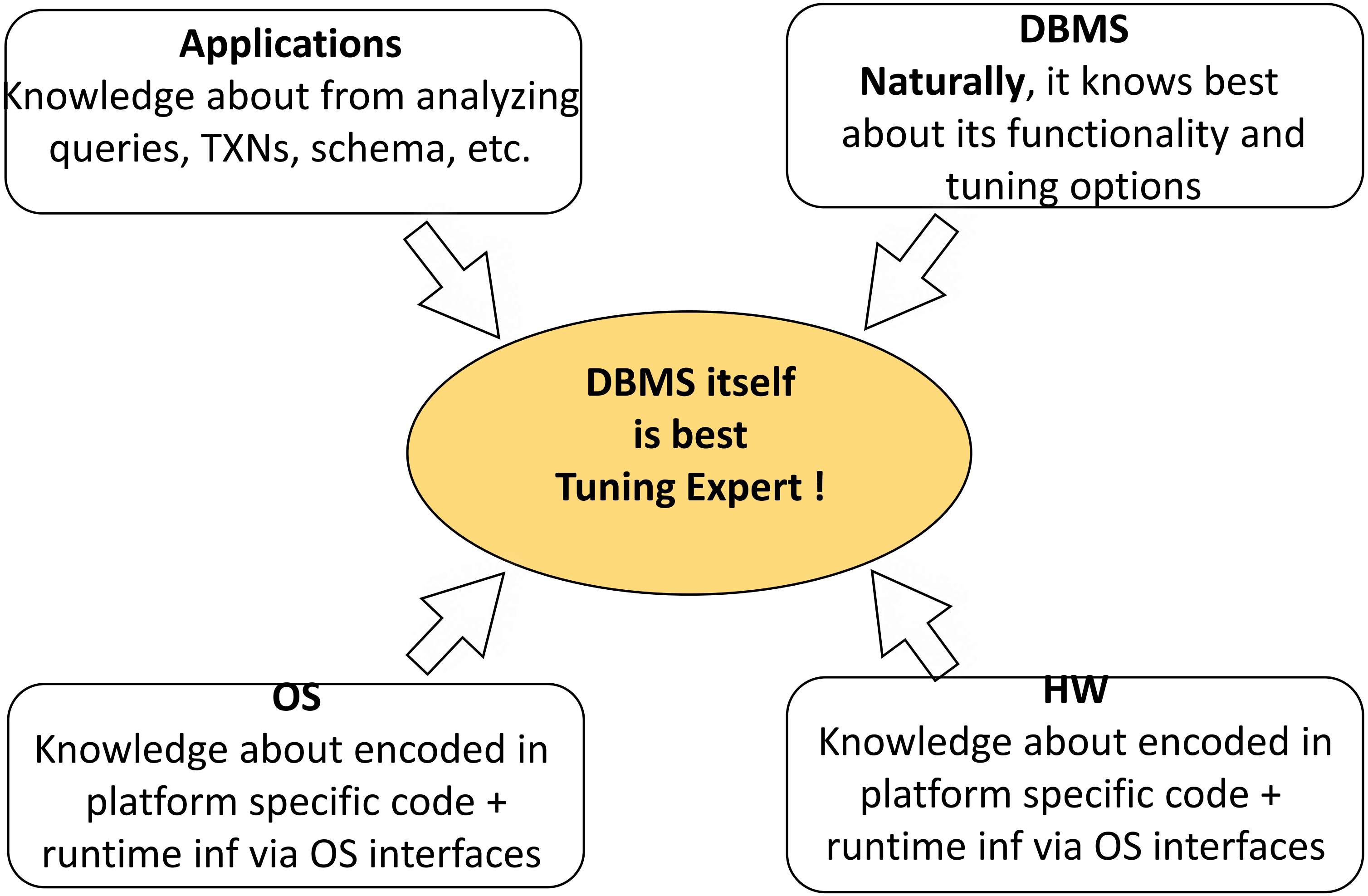
PkE	Course	Student	Name	Degree	Date	Master	Other

Finally: **Schema fusion** of relations (1:1) **and View materialization**

DBMS tuning

- Tune the transaction manager (log management and storage, checkpoint frequency, dump frequency)
- Tune the buffer size (as well as interactions with the operating system)
- Disk management (allocation of memory for tablespaces, filling factor for pages and files, number of preloaded pages, accesses to files)
- Use of distributed and parallel databases

Database self-tuning



Applications
Knowledge about from analyzing queries, TXNs, schema, etc.

DBMS
Naturally, it knows best about its functionality and tuning options

DBMS itself is best Tuning Expert !

OS
Knowledge about encoded in platform specific code + runtime inf via OS interfaces

HW
Knowledge about encoded in platform specific code + runtime inf via OS interfaces

Database self-tuning

- Databases are getting better and better at this
- This is clearly the way to go
- But there will be still space for a good DBA

Exercise

- Tables:
  - Sales(Date,FKShop,FKCust,FKProd,UnitPrice,Q,TotPrice)
  - Shops(PKShop,Name,City,Region,State)
  - Customer(PKCust,Nome,FamName,City,Region,State,Income)
  - Products(PKProd,Name,SubCategory,Category,Price)

Exercise

- Sales: 100.000.000, 1.000.000; Shops: 500, 2; Customers: 100.000, 1.000; Products: 10.000, 100

```
SELECT Sh.Region, Month(S.Date),Sum(TotPrice)
FROM Sales S join Shops Sh on FKShops=PKShops
GROUP BY Sh.Region, Month(S.Date)
```

- Propose a primary organization based on this query
- Compute the cost of an optimal access plan based on this organization
- Add a condition: WHERE 1/1/2017 < Date