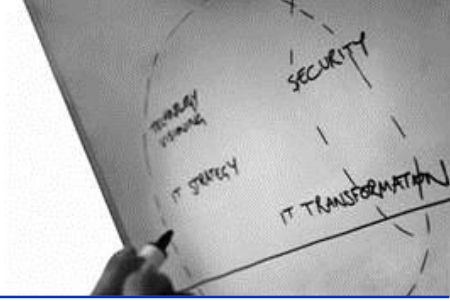
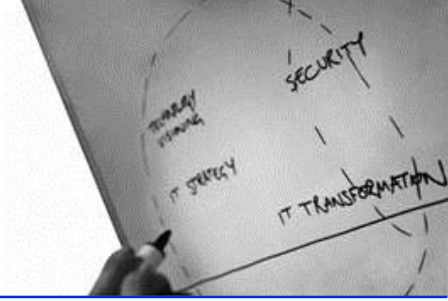
Large databases and performance
Lecture UiO 29. oct. 2015

Audun Faaberg - Accenture

Agenda



1. Introduction.
2. Tune the SW (reduce the IO)
   a. Indices
   b. Efficient SQL
   c. Efficient code & design
3. SOA, object orientation, distance
4. New directions
5. Performance engineering
6. Performance evaluation techniques
7. Real life examples

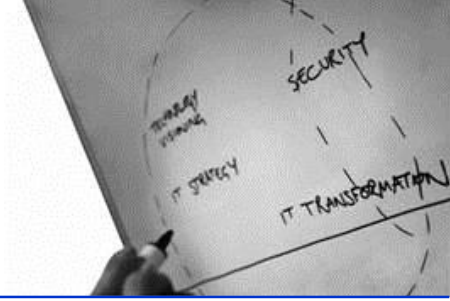


About me

- Senior manager in Accenture Technology Consulting, leading the Performance engineering group in Norway & the Nordics.
- Worked in Arthur Andersen / Andersen Consulting / Accenture since 1989.
- Specialise in technical project management in large multi platform environments.
- Key words are data base technology, performance, go-live, and problem management
- DB2, Oracle, MSSQL, Sybase, Unix, Linux, z/OS, Cobol, C, Java, Tuxedo, MQ, CICS, WebLogic, WebSphere, MIIS ++
- audun.faberg@accenture.com

About Accenture

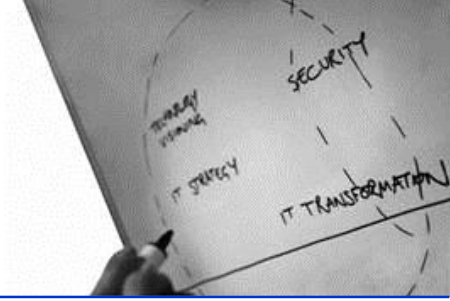
- 358.000 employees, worldwide (31. Aug 2015).
- Offices in 200 cities / 56 countries.
- Largest country: India.
- 1200 employees, Norway.
- Offices in Oslo, Bergen, Stavanger.
- In the Nordics – Helsinki, Stockholm, Gothenburg, Copenhagen, + nearshore centre in Riga.



Disclaimer :-)

- I have worked in Norway, Denmark, France, Italy, Brazil, The Czech Republic, Malaysia, Germany – MEANING – “a telco” is not Telenor, “a bank” is not DnB NOR, “an oil company” is not StatoilHydro……
- Specific numbers may be old (page size, cache size, IO speed). But the logic should still apply, though of course the massive changes currently may lead to this logic producing another result…..
- This is not a full course of tuning in large databases, it is to make you aware of what is out there…

1. Introduction
Basic arithmetic



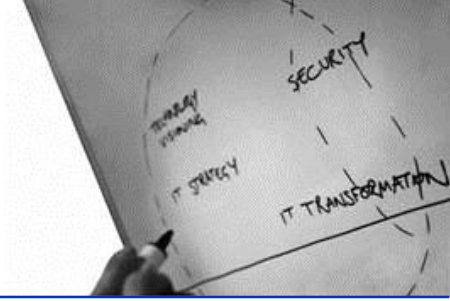
0.000 sec x 6,000,000 = 0 sec

Many designers think this is the database speed

0.012 sec x 6,000,000 = 72000 sec = 20 hours

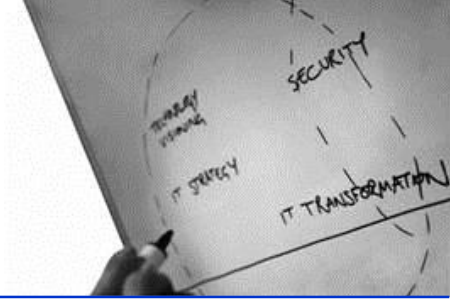
And this may be the real database speed

1. Introduction
Poor performance



- Poor performance in the database and the code using the database is the most common reason for poor system performance.
- Poor performance may render an otherwise good system useless. Or despised. Or lead to ineffective organisations and numerous coffee breaks….
- Problems with performance may cause large delays in the final phases of a project, though should be more manageable than other issues that may occur at this stage.
- Finally, there is a lack of understanding that optimising code may drastically alter CPU &IO consumption. Therefore, more HW is the normal (often wrong) answer.

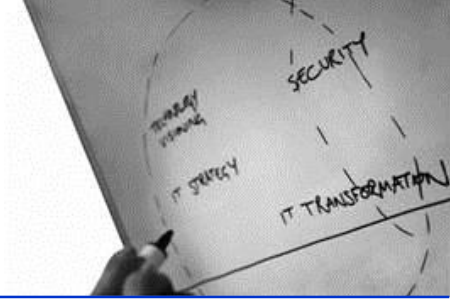
1. Introduction
Why is there still a problem



- HW is faster. CPU, disks, network, memory, everything is much faster now than 10 years ago.
- Moore's law: Integrated circuits would double in performance every 18 months.
- Why is there still a problem?

- Niklaus Wirth's law: Software is getting slower more rapidly than hardware becomes faster.
- To be more specific – Gate's law: The speed of commercial software generally slows by fifty percent every 18 months.

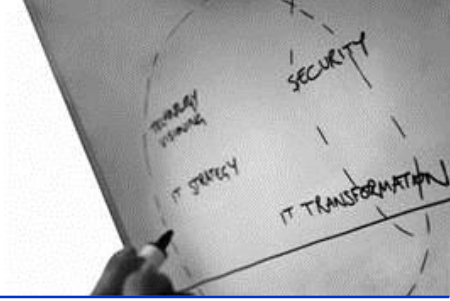
1. Introduction
Why is there still a problem



- Databases are now more finely modelled, catching more data, and creating complexities an order of magnitude greater.
- Flexible, parameterised, configurable system use many small parameter tables, which are used in every select. So earlier when a select referenced 2-3 tables, it may now reference 8-10 tables to fetch the same data. (And beware if a system is “generic”)
- Greater ambitions – meaning – now much larger data volumes are stored. 10-15 years ago, we were stingy when designing databases.

- And – even if CPUs double their speed every 18 month, disk IO speed has only increased 10-100 x since the 1960s.

- Shift to client – server – and now network computing: Adds network latency, integration with other systems, much larger data volumes, more complex user interfaces, more complex processes and a large degree of aggregation of information.



1. Introduction
Example of data modelling

Loan	
Cust_id	**Balance**
10	2,405,256
20	1,530,203

Earlier modelling,
1 record pr loan

2 million customers

2 million rows

Loan	
Cust_id	**Loan-_series_id**
10	1,562,030
20	1,830,203

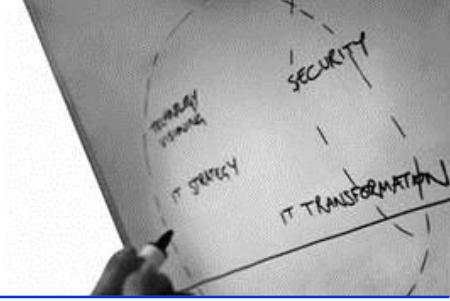
Recent modelling,
241 records pr loan

482 million rows

Functionally more flexible, but at a cost

Loan_series (one record for every month throughout 20 years)					
Loan_series_id	**Year**	**Month**	**Org_amount**	**Remaining**	**Interests**
1,562,030	2008	01	2,405,256	2,395,256	23,952
1,562,030	2008	02	2,405,256	2,375,256	23,752
…	…	…	…	…	
1,562,030	2028	12	2,405,256	20,000	200

2. Tune the SW



Note: Tuning of SQL is typically a 40 hours introduction course + 5 years experience.

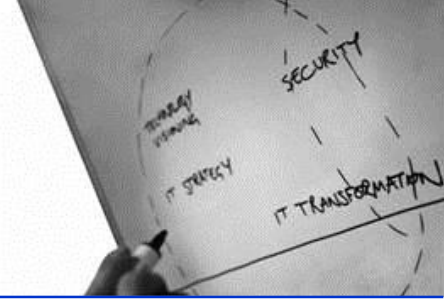
This is just a broad overview, so you will know there is more (much more) to know.

– Indices
– Efficient SQL
– Efficient code & design

Basic principles

- Divide and conquer
- Minimise the fetch of everything

2. Tune the SW
A DBMS model



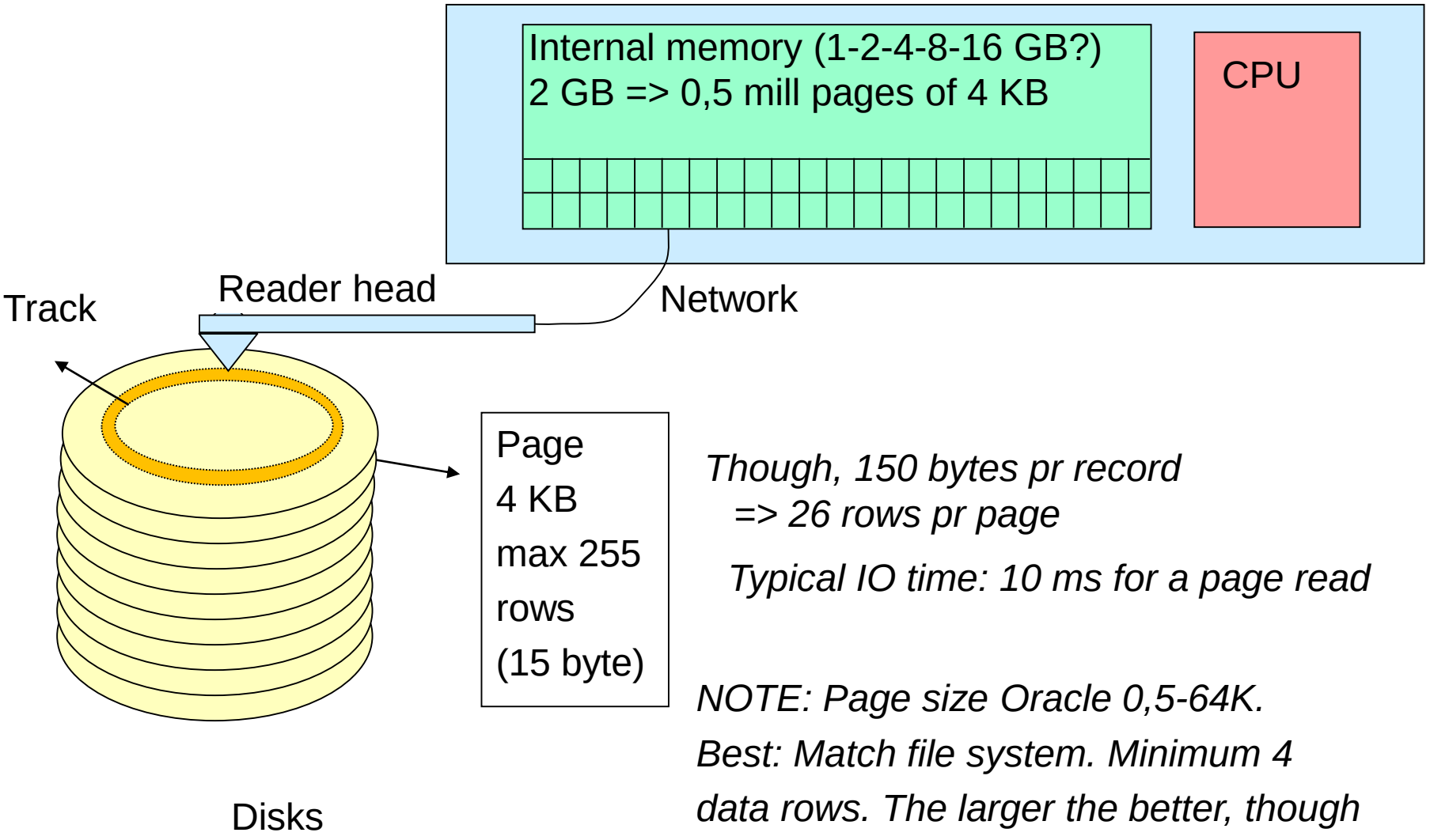


Figure: Audun Faaberg

Though, 150 bytes pr record
=> 26 rows pr page

Typical IO time: 10 ms for a page read

NOTE: Page size Oracle 0,5-64K.
Best: Match file system. Minimum 4 data rows. The larger the better, though concurrency may dictate otherwise

2. Tune the SW – a. indices
What are indices – real world example

- Oslo Map
  - Map pages
  - Index pages
- Index
  - Carl Berners Plass 16 G3
  - Tullinløkka 10 F2

Figure: Jan Haugland

2. Tune the SW – a. indices
What are indices

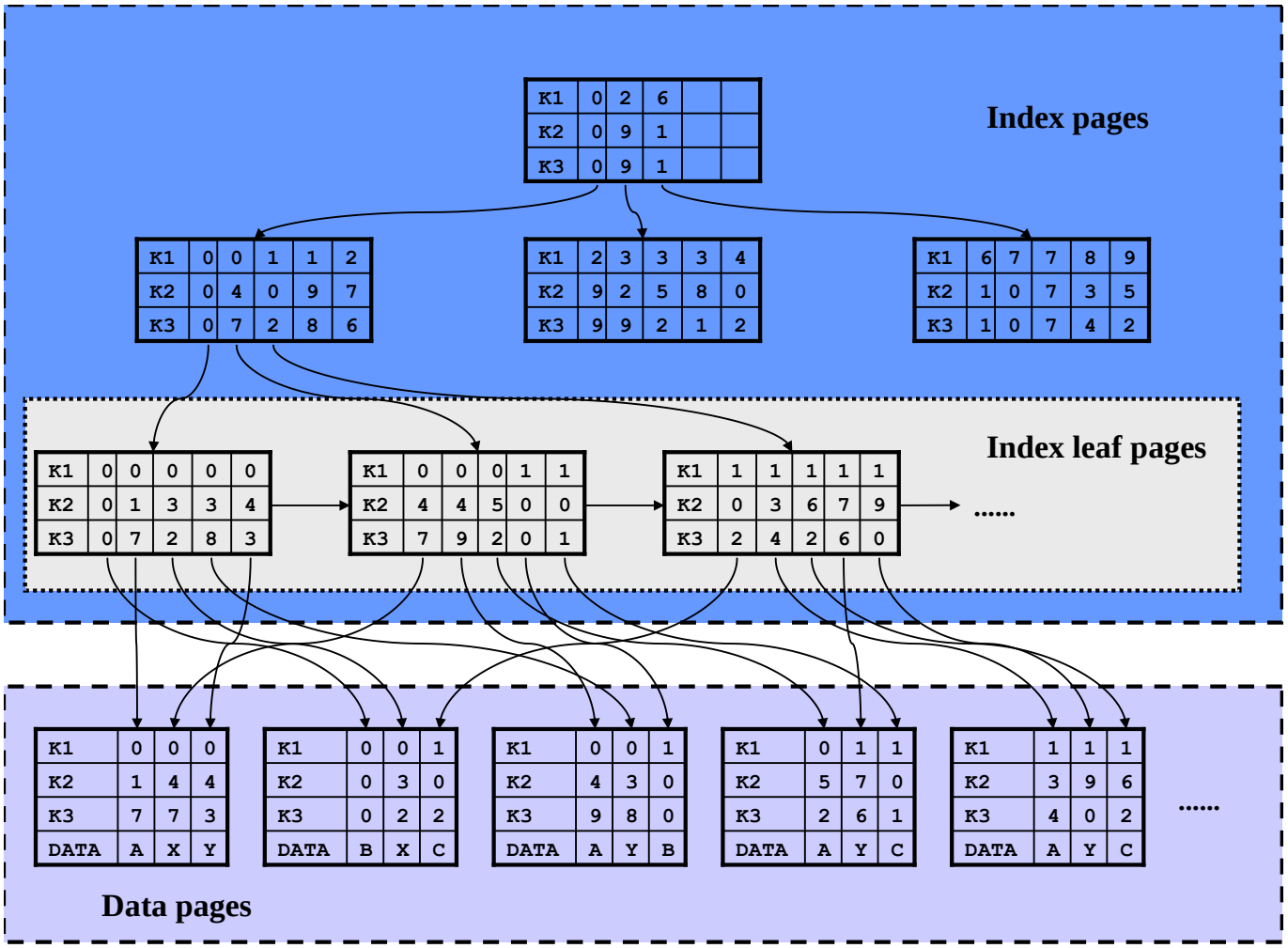
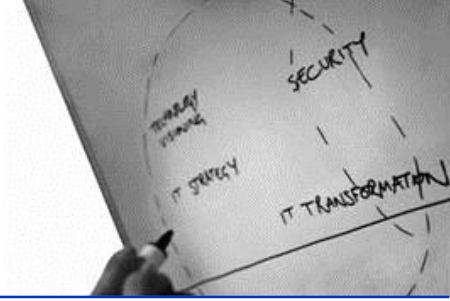


Figure: Jan Haugland

2. Tune the SW – a. indices
What are indices



- A tree downwards through index pages, and on the leaf pages there is a pointer to the very page and an offset for the row in question.
- The indices are stored in index spaces (corresponding to table spaces)
- The disk space used by the index spaces may be as large as for the table spaces.
- Since the columns of an index is (normally) fewer than in the complete row, more are stored in a page (though max 255). Thus a index scan is faster than a table scan.

- May dramatically lower the number of pages read to find a row. Read through 3-4 layers of indices (pages), versus scanning the whole table with thousands of pages.

2. Tune the SW – a. indices
Full Table Scan

```
SELECT K2
  FROM SIMPLE_TABLE
 WHERE DATA = 'X'
```

- All yellow keys match
- All blue values returned
- All red pages scanned

- Let us assume the table has 50 mill rows, 20 rows pr page. (page 4K, row 200 bytes).
- In mean 1.250.000 page reads to find a random row.
- 10 ms pr page read….. 12.500 sec = 3,5 hours

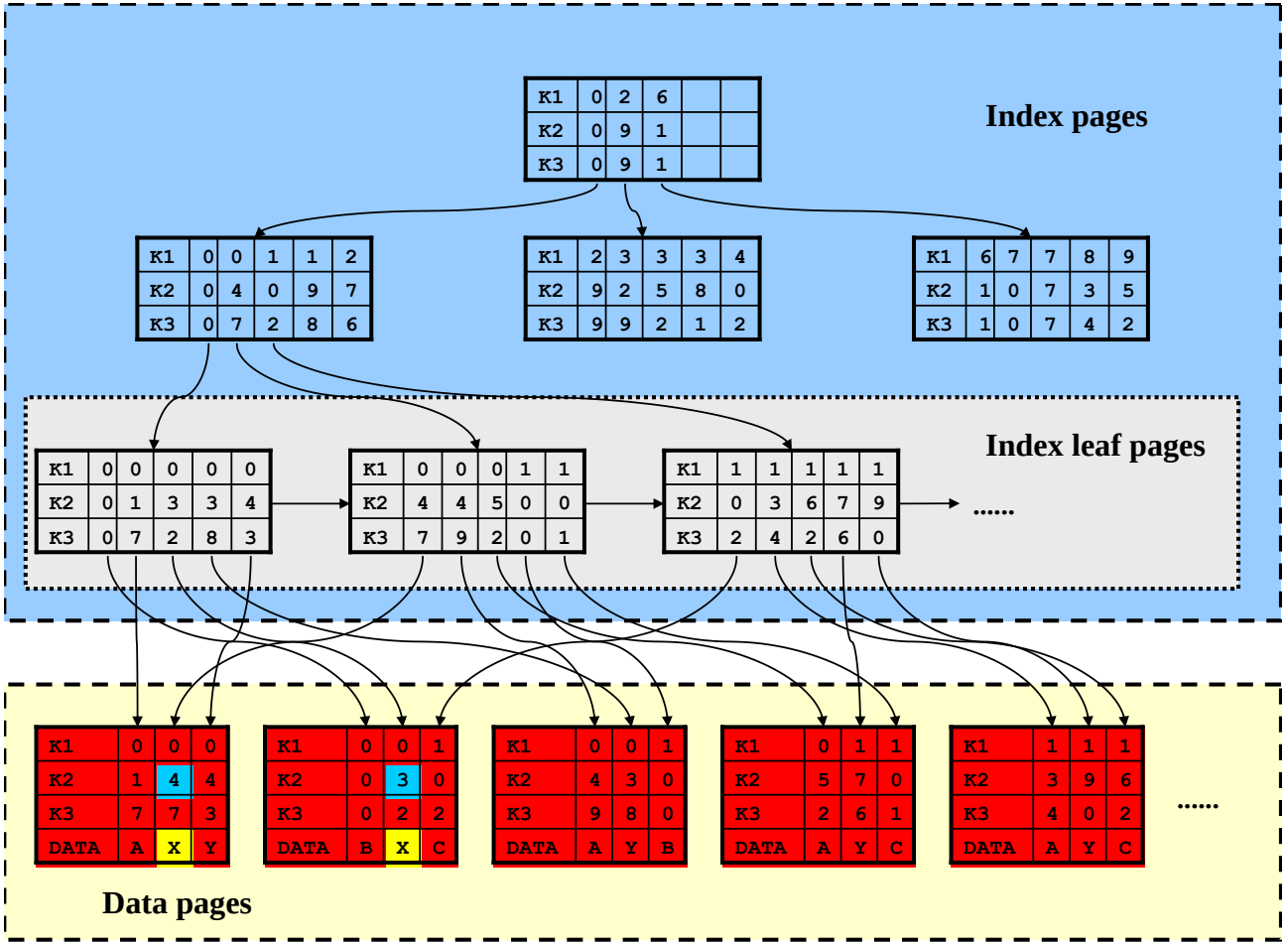


Figure: Jan Haugland

2. Tune the SW – a. indices
Matching Index Scan

```
SELECT DATA
  FROM SIMPLE_TABLE
 WHERE K1 = 0
   AND K2 = 3
```

- All green keys match
- All yellow index entries used
- All blue values returned
- All red pages scanned

- Let us assume the table has 50 mill rows, 20 rows pr page.
- Indeces: 20 bytes – 200 on each page. 250.000 leaf pages, need 3 levels of index pages.
- 5 IOs -> 50 ms.

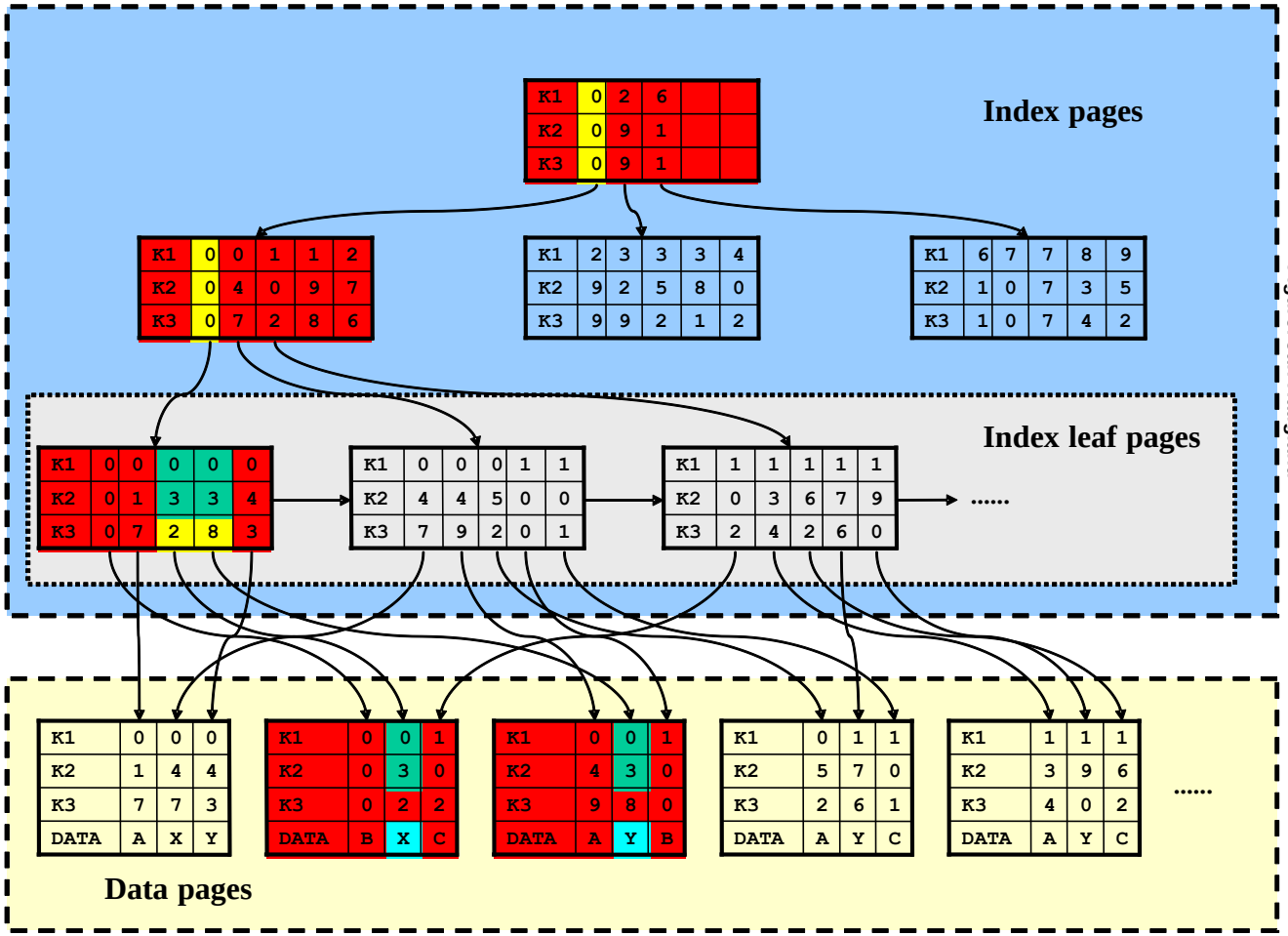


Figure: Jan Haugland

2. Tune the SW – a. indices
Non-matching Index Scan

```
SELECT DATA
  FROM SIMPLE_TABLE
 WHERE K3 = 7
```

- All yellow keys match
- All blue values returned
- All red pages scanned

- Let us assume the table has 50 mill rows, 20 rows pr page.
- Indices: 20 bytes – 200 on each page. 250.000 leaf pages
- 125.000 IOs -> 1250 s.

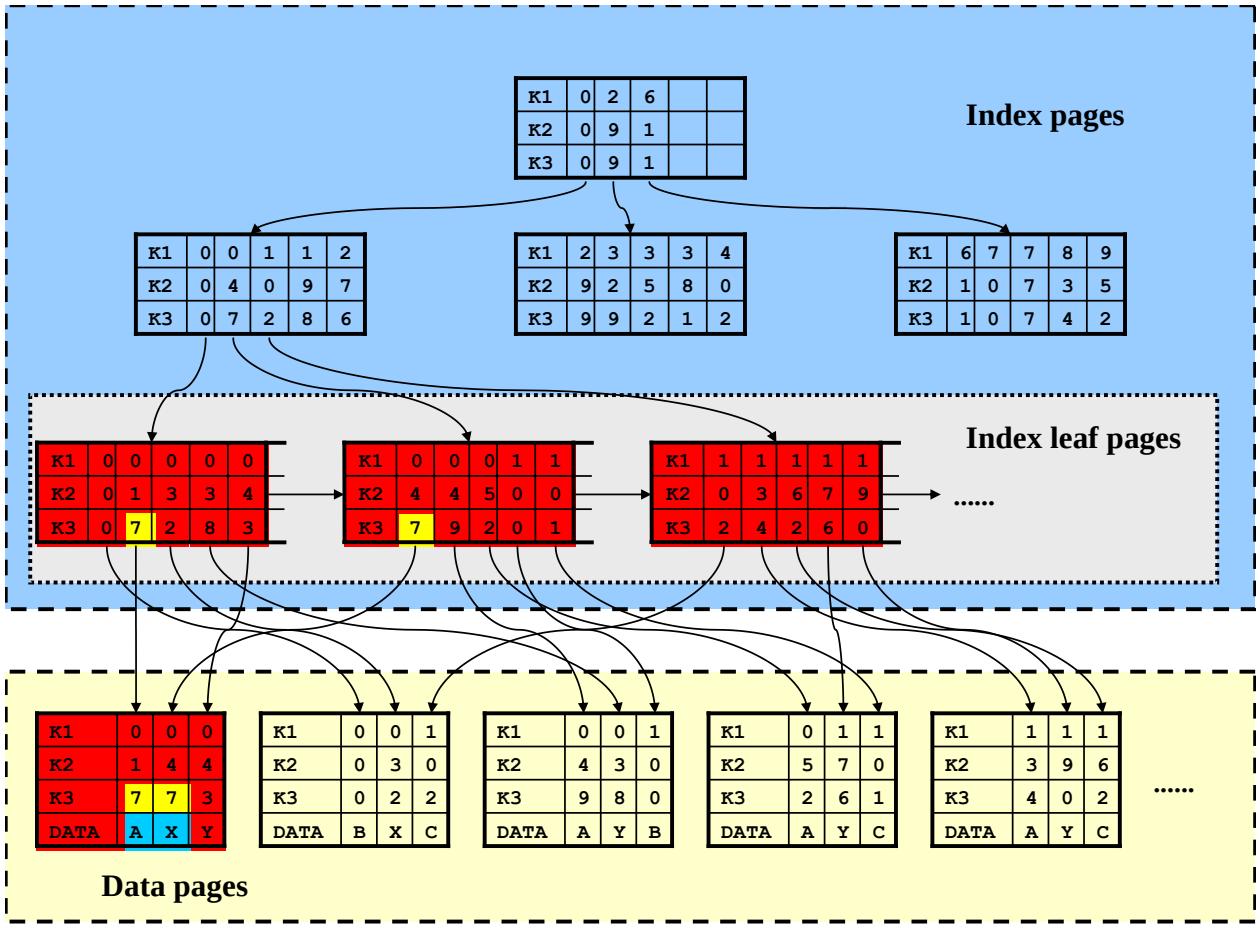


Figure: Jan Haugland

2. Tune the SW – a. indices
Specify your select

```
SELECT K2, K3
  FROM SIMPLE_TABLE
 WHERE K3 = 7
```

- Will result in a scan of the index leaf pages
- No read of data pages necessary.
- This is one reason to avoid SELECT * and rather specify the columns.
- Sometimes we add a missing column to the index
- If you have many hits, you may save 50% of the IO.
- Few hits, negligible gain.

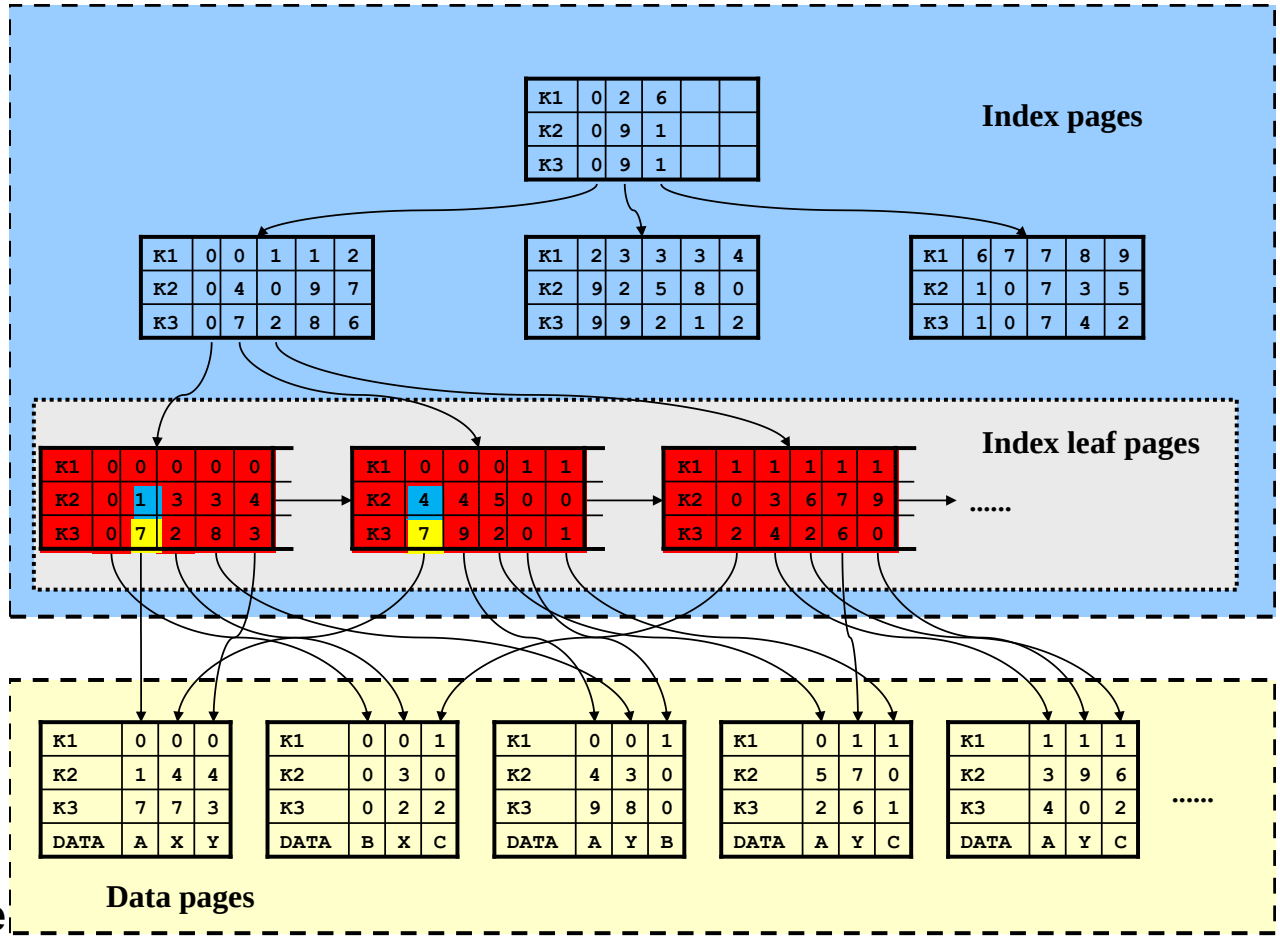
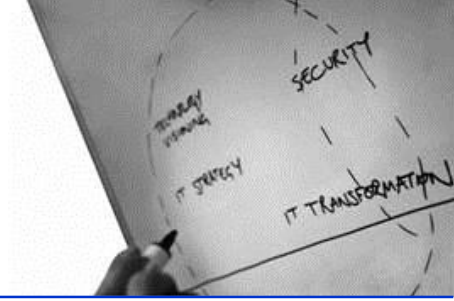


Figure: Jan Haugland

2. Tune the SW – a. indices
Sequential prefetch



- A correction – the table scans are in fact more efficient than depicted in the earlier examples.
- A mechanism “Sequential prefetch” (or “scatter read”) is invoked when the DBMS discovers that it is reading in sequence through the pages (typically 3 pages in sequence within 10 page reads).

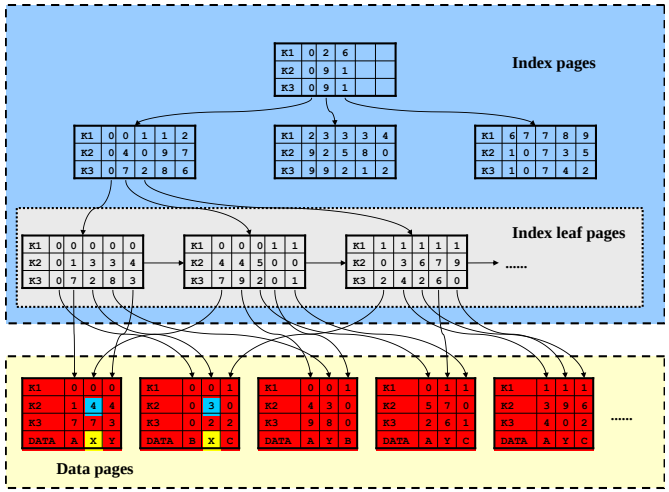
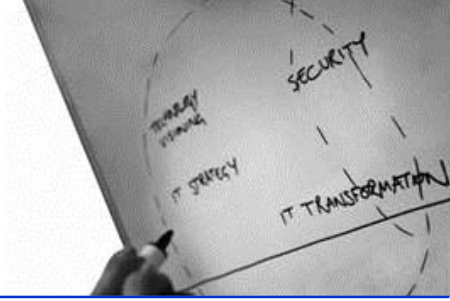


Figure: Jan Haugland

- Starts to read 50 and 50 pages, typically at 30 ms (compared to 10 ms for one page). Leading to 25.000 read operations in a full table scan, or 750 / 2 sec = 6,25 minutes to find a random row (mean).
- Also the non-matching index scan will start prefetching. Leading to 4.000 read operations, or 120 / 2 sec = 60 sec.
- This can be utilised in large batch reads!
- Typically – if you try to select more than 10% of the rows in a table, the optimiser will go for a table scan.

2. Tune the SW – a. indices
Function index




```
select lname, empno, sal
from emp where
upper(lname) = 'FAABERG';
```

- A standard select will not find an index to ‘FAABERG’.
- Result is a table scan, and convert every ename to upper.
- A function index is an index with a function value store.
- DBMS follows directly the function index, and uses the pointer down to the data page.

- Note also that the same problem arises when comparing the text "123" vs. the number 123 !!

Eriksen	Erik	123
Faaberg	Audun	154
Faaberg	Rasmus	549
Horpen	Hallvor	798
Tallaksen	Tallak	101

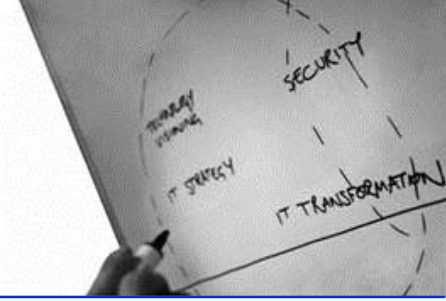
Original table

Function index

ERIKSEN
FAABERG
HORPEN
TALLAKSEN

Figure: Audun Faaberg

2. Tune the SW – b. efficient SQL Optimiser



- Cost based optimiser: Uses statistics, and weighs operations.

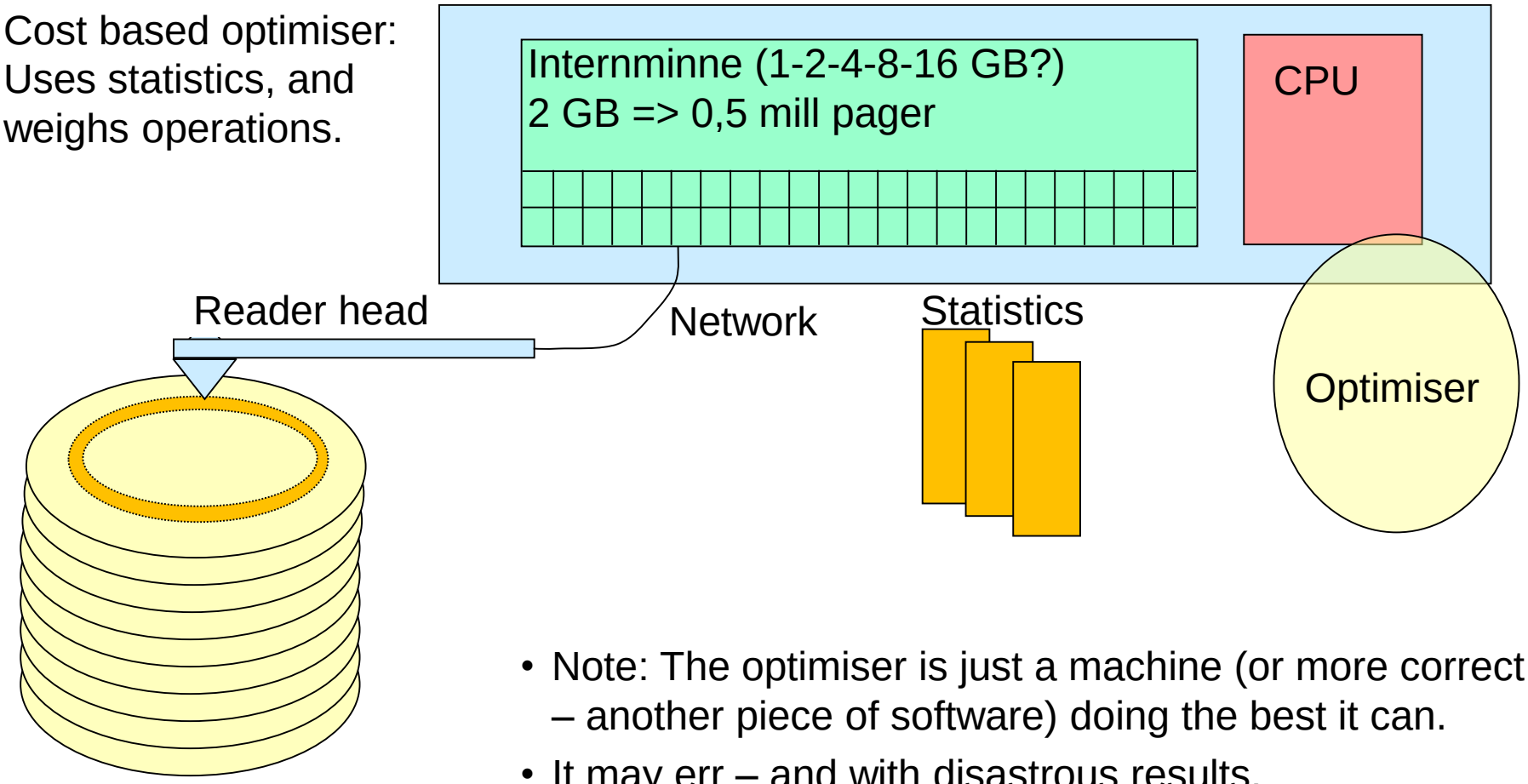
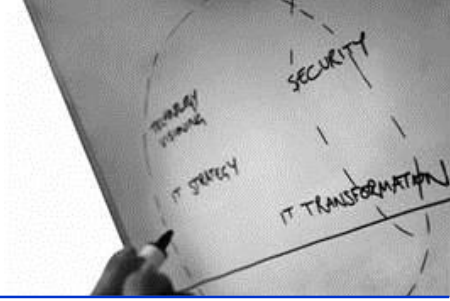


Figure: Audun Faaberg

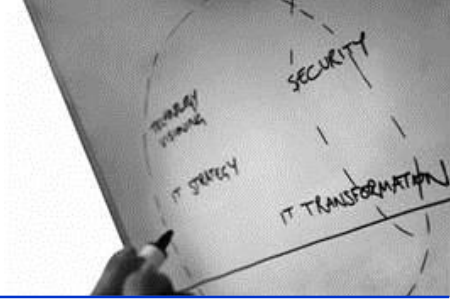
- Note: The optimiser is just a machine (or more correct – another piece of software) doing the best it can.
- It may err – and with disastrous results.
- A DBA I know – after several hours of testing different setups: "Finally I framed the optimiser!"

2. Tune the SW – b. efficient SQL
Access path



- Access path is the way and sequence the DBMS applies rules. Using an index? Joins – in which order? Sort?
- It may be necessary to understand the access path.
- A database simulator tool may help you.
- In large projects with large database – we sometimes have a centralised function approving all SQL (typically testing in with the simulator… or on a large test database).
- You set up the simulator with the estimated number of rows in the different tables, indicates a cardinality / distribution (meaning – for large projects this is no small effort!)

2. Tune the SW – b. efficient SQL
Looking for the millionaire

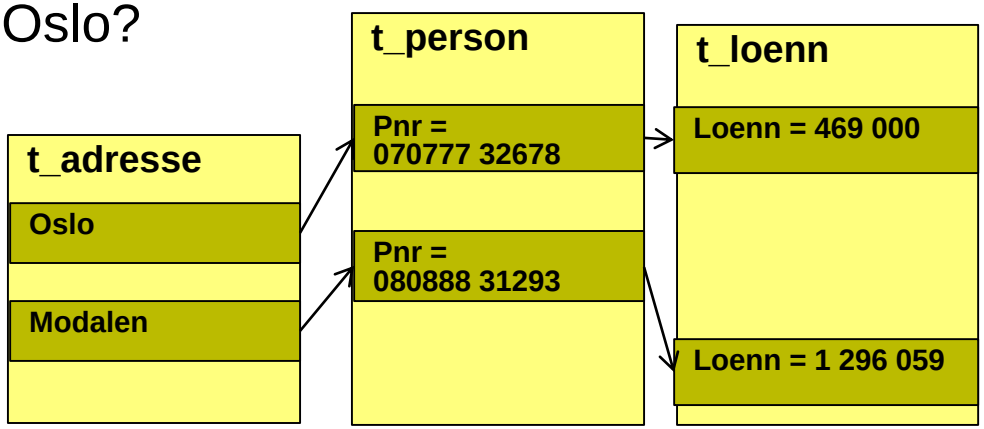


Before we start looking at SQLs and access paths - let us look at the real world. Tax is fun.
How would you find the millionaires in Modalen county (one of the smallest counties in Norway). By hand, by sifting through index cards.

a) Give me index cards of the millionaires in Norway, with the county added on. Read through the index cards.

b) Give me all index cards of Modalen. I will scan through all of it.

And for Oslo?



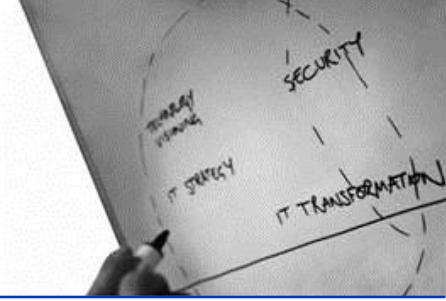
Distribution info

Modalen: 274 taxpayers

Oslo: 439 272 taxpayers

Over 1 mill in Norway: 60 261

2. Tune the SW – b. efficient SQL Access path



Consider the employee table

```
select lname, empno, sal
from emp where
upper (lname) = 'FAABERG';
```

Eriksen	Erik	123
Faaberg	Audun	154
Faaberg	Rasmus	549
Horpen	Hallvor	798
Tallaksen	Tallak	101

With no function index:

```
0 SELECT STATEMENT Optimizer=COST
1 0 TABLE ACCESS (FULL) OF 'EMPLOYEE_TABLE'
```

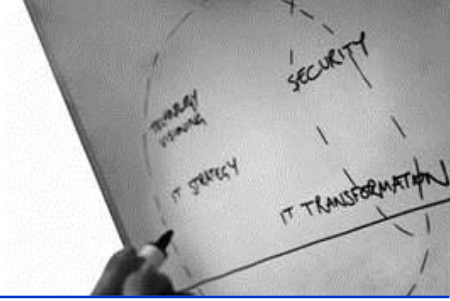
50

With an function index on upper(ename):

```
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 INDEX (RANGE SCAN) OF 'UPPER_ENAME_IDX' (NON-UNIQUE)
```

1

2. Tune the SW – b. efficient SQL Access path



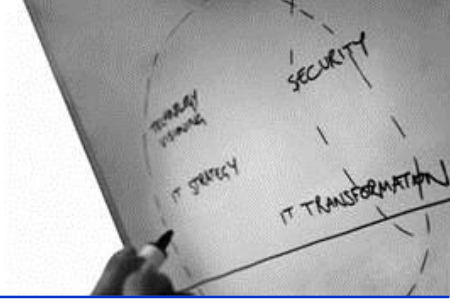

```
SELECT e.employee_id, e.first_name, e.last_name, e.salary
FROM employees e
WHERE EXISTS (SELECT 1
              FROM orders o
              WHERE e.employee_id = o.sales_rep_id
              AND o.customer_id = 144);
```

subselect has reference to column in the outer select

```
ID OPERATION                    OPTIONS          OBJECT_NAME OPT COST
---- ------------------------- -------------- ----------- --- ----
0 SELECT STATEMENT                                        CHO
1 FILTER
2 TABLE ACCESS                  FULL             EMPLOYEES   ANA 155
3 TABLE ACCESS                  BY INDX ROWID    ORDERS      ANA
3 4 INDEX                       RANGE SCAN       ORD_CUST_IX ANA 1
```

Correlated subselect

2. Tune the SW – b. efficient SQL Access path



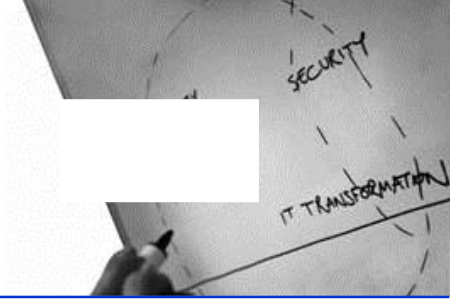
A rewrite of the SQL

```
SELECT e.employee_id, e.first_name, e.last_name, e.salary
FROM employees e
WHERE e.employee_id IN
        (SELECT o.sales_rep_id
            FROM orders o WHERE o.customer_id = 144);
```

Non-correlated subselect

```
ID OPERATION            OPTIONS         OBJECT_NAME OPT COST
---- ----------------- -------------- ----------- --- ----
0 SELECT STATEMENT                                 CHO
1 NESTED LOOPS                                        5
2  VIEW                                               3
3   SORT                UNIQUE                         3
4    TABLE ACCESS       FULL            ORDERS      ANA 1
5     TABLE ACCESS      BY INDEX ROWID  EMPLOYEES   ANA 1
6      INDEX            UNIQUE SCAN     EMP_ID_PK   ANA 1
```

2. Tune the SW – b. efficient SQL
Correlated subselect - execution




```
SELECT e.employee_id, e.first_name, e.last_name, e.salary
  FROM employees e WHERE EXISTS
  (SELECT 1 FROM orders o WHERE e.employee_id =
  o.sales_rep_id
   AND o.customer_id = 144);
```

1

Full table scan
select employee,
first_name,
last_name,
salary
from employees

employees

employee_id	last_name	first_name	salary
100002	hansen	nils	###
100003	kvam	per	###
100007	torsen	johan	###
100008	eri	johan	###
100010	hoff	magne	###
100011	sand	knut	###
100012	ludvigsen	anders	###
100013	knutsen	jørgen	###
100017	….	….	…
100018	….	….	…

2

Sets up a candidate list (with all rows in the table)

candidate list:

100002 ….	….
100003 ….	….
100007 ….	….
100008 ….	….
100010 ….	….
100011 ….	….
100012 ….	….
100013 ….	….
100017 ….	….
100018 ….	….

3

Check done for every single candidate via index

orders

order_id	customer_id	sales_rep_id
200102	201	100010
202012	144	100012
259037	090	100008
310807	144	100007
338765	937	100007
348999	144	100007
560312	771	100012

4 **Create candidate list**

candidate list

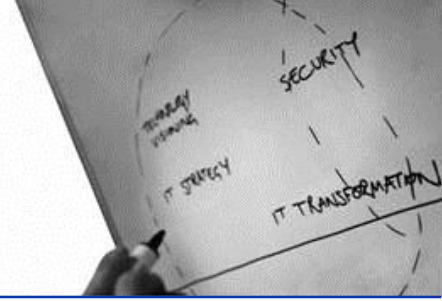
202012	144	100012
310807	144	100007
348999	144	100007

5

Scann candidate list

The candidate list for orders is always the same, still it must be created for every single employee….. In one pass it will match 202012, in another pass it will Match 310807 and 148999.

2. Tune the SW – b. efficient SQL
Non correlated subselect - execution



1

Uses index on customer

SELECT sales_rep_id
FROM orders
WHERE customer_id = 144

orders

order_id	customer_id	sales_rep_id
200102	201	100010
202012	144	100012
259037	090	100008
310807	144	100007
338765	937	100007
348999	144	100007
560312	771	100012

employees

employee_id	last_name	first_name	salary
100002	hansen	nils	###
100003	kvam	per	###
100007	torsen	johan	###
100008	eri	johan	###
100010	hoff	magne	###
100011	sand	knut	###
100012	ludvigsen	anders	###
100013	knutsen	jørgen	###
100017	….	….	…
100018	….	….	…

2

Set up a candidate list (just 3 hits)

candidate list

202012	144	100012
310807	144	100007
348999	144	100007

3

Scann candidate list

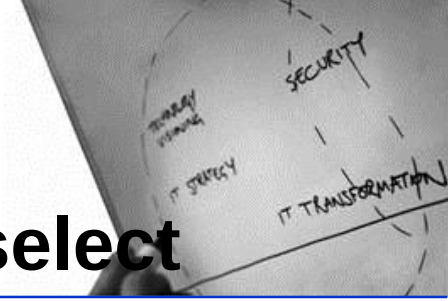
4

Complete candidate list with lookup via index

SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE employee_id = 100007

```
SELECT e.employee_id, e.first_name, e.last_name, e.salary
FROM employees e
WHERE e.employee_id IN
        (SELECT o.sales_rep_id
            FROM orders o WHERE o.customer_id = 144);
```

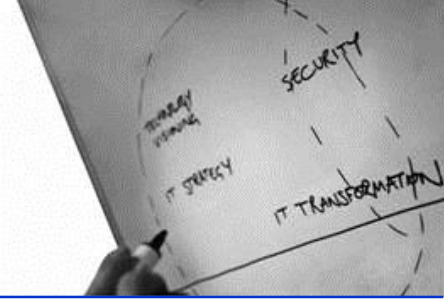
2. Tune the SW – b. efficient SQL
Correlated versus non correlated subselect



- A subselect is correlated if it has references to columns in the outer select.
- OK as extra refinement and filtering. Extremely expencive as main filtering.

- A subselect is non correlated if it does not have references to columns in the outer select, meaning you can execute the subselect independantly – and as the first SQL in a stepwise exwecution plan.
- OK as main filtering. Extremely expencive as extra refinement filtering.

2. Tune the SW – b. efficient SQL
Missing join predicate



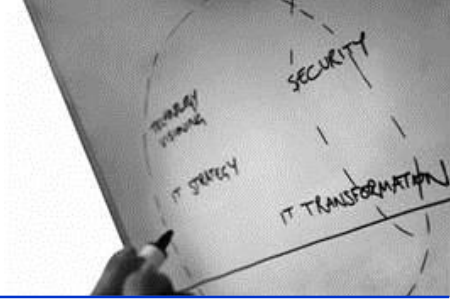

```
SELECT H.SAK_ID
      ,L.KRAVLINJE_ID
  FROM T_KRAVHODE   H
      ,T_KRAVLINJE L
```

This is the infamous Carthesian product

$A \times B = \{ (a,b) \mid a \in A \wedge b \in B \}$

T_KRAVHODE	
KRAVHODE_ID	SAK_ID
10	100
20	200
T_KRAVLINJE	
KRAVHODE_ID	KRAVLINJE_ID
20	2000
30	3000
QUERY RESULT	
SAK_ID	KRAVLINJE_ID
100	2000
100	3000
200	2000
200	3000

2. Tune the SW – b. efficient SQL
Why are carthesians disastrous?




```
SELECT P.Person_number
      ,A.Bank_account
  FROM T_Person P
      ,T_Account A
```

Here Hans Alnes is matched with ALL accounts in Norway (22,5 millions of them)

Thereafter Kari Thune is matched with ALL accounts

Giving a list of 4,5 * 22,5 million² = 101,25 mill mill

101 250 000 000 000 items

25 million seconds (seq prefetch)

```
SELECT P.Person_number
      ,A.Bank_account
  FROM T_Person P
      ,T_Account A
  WHERE P.Person_number = A.Person_number
```

Here Hans Alnes is matched with his 5 accounts

Thereafter Kari Thune is matched with her 5 accounts

Giving a list of 4,5 million * 5 = 22,5 mill

(30-40 seconds with seq prefetch)

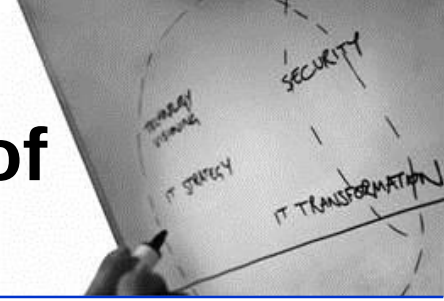
Person	
Person_number	**Name**
05056x47126	Hans Alnes
09118y10017	Kari Thune

Account	
Person_number	**Account_number**
05056x47126	1533 289 08971
05056x47126	1533 289 08988
05056x47126	In mean 5 accounts
09118y10017	1540 780 01122
09118y10017	9833 010 89876
09118y10017	In mean 5 accounts

4,5 million persons

22,5 million accounts

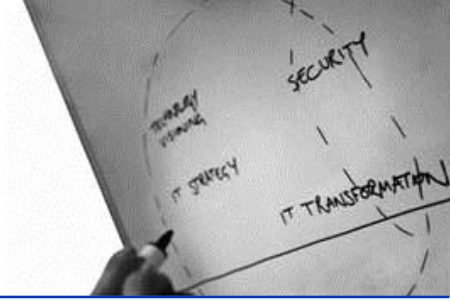
2. Tune the SW – b. efficient SQL
Can I predict the execution sequence of a compound statement?



- No sequence granted, but most likely something like:

```
select mandatory1.x                          (7)
      ,optional.y
  from mandatory1                            (2 or 3)
inner join mandatory2                        (3 or 2)
    on mandatory1.z = mandatory2.z
left outer join optional                     (4)
    on optional.u = mandatory2.u
 where mandatory2.w = ?
   and mandatory1.a in
         (non-correlated subselect)          (1)
   and exists (correlated subselect)         (5)
order by mandatory.x                         (6)
```

2. Tune the SW – b. efficient SQL Connection statement cache



- A DBMS must translate the SQL statements sent to it. This is a CPU-demanding process (finally…. till now we have mostly looked at IO and memory….).
  - Load into shared pool
  - Syntax parse (correct SQL as such)
  - Semantic parse (are all table & column names correct, check dictionary)
  - Optimisation (create access plan with info from db statistics)
  - Create executable
- You may set up each connection with a cache of SQL statements already translated,.
- Requires the SQL to be exact the same. Is case sensitive. Must use bind variables, not values.

```
select order_id, account_id
    from order_item
where account_id = :OrderId
```

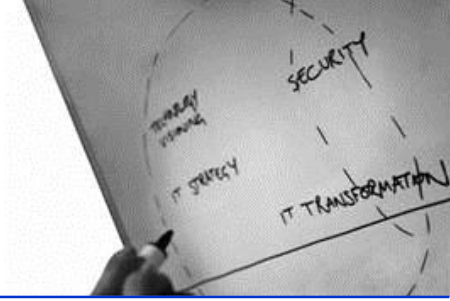
Does not match neither

```
select order_id, account_id
    from order_item
where account_id = 158293
```

- Hint: Always user bind variables, even when you work with a constant. And use the same variable name

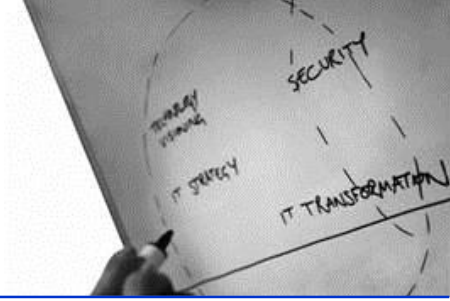
```
select Order_Id, Account_Id
    from Order_Item
where Account_Id = :OrderId
```

2. Tune the SW – b. efficient SQL
SQL tuning



- The SQLs you meet in real life are often much more complex than the examples I have given.
- Most important tool – sql statistics (all DBMSs have some way for gathering this).
- A large system may have thousands of SQLs spread out in the code (or as stored procedures referenced in the code).
- In a problem situation, normally a handful (5-10-20) SQLs are causing problem. Though many more may be inefficient….
- First of all, identify them.
- Look for logical reads and physical reads in statistics, thus identifying the problem candidates.
- Candidates may be:
  - Light SQLs, somewhat inefficient, but very frequently executed
  - Heavy SQLs with massive reads (logical and/or physical)

2. Tune the SW – b. efficient SQL Tools - Detector

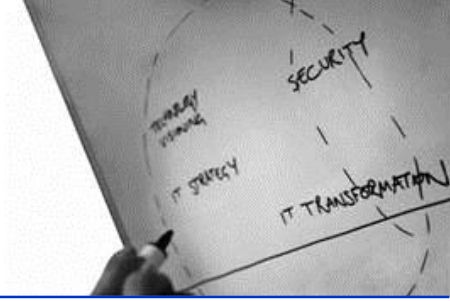


09.02.2009
kl 0800-1200

PROGRAM	SQL	CPUPCT	INDB2_CPU	GETPAGE
K411S024	8798695	19.92%	19:51.943399	25044646
K415B940	7206008	5.83%	05:12.778640	42480939
K231B510	521364	4.97%	04:26.795714	12158914
K278U950	4060	4.03%	03:36.202277	13502081
K411S025	4072793	3.75%	03:21.168218	10610520
K278BAN1	16086	2.79%	02:29.905171	8802622
DSNESM68	8655	2.54%	02:16.268729	23541223
K411S103	1966527	1.93%	01:43.911804	4951095
K2300211	3068353	1.76%	01:34.334010	3433748

Start optimising from the top.
Use information in the tool.
Optimse CPU-consumption? IO? Elapsed time?

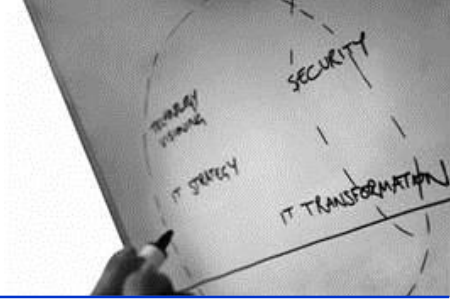
2. Tune the SW – b. efficient SQL Example 1




```
DECLARE C_TREKKDATA_3 CURSOR FOR
SELECT DISTINCT A.KREDITORS_REF
, A.KODE_TREKKALT
, A.SATS
, A.BELOP_SALDOTREKK
, A.BELOP_TRUKKET
, A.DATO_OPPFOLGING
, O.TSS_OFFNR
FROM V1_ANDRE_TREKK A
, V1_TREKK_I_FAGOMR F
, V1_TSS_SORTDATA O
WHERE A.TREKKVEDTAK_ID = :H
AND A.LOPENR = 9999
AND :H = 9999
AND F.KODE_FAGOMRAADE = "IT26"
AND O.KREDITOR_ID_TSS = :J
AND O.LOPENR = 9999
FOR FETCH ONLY
```

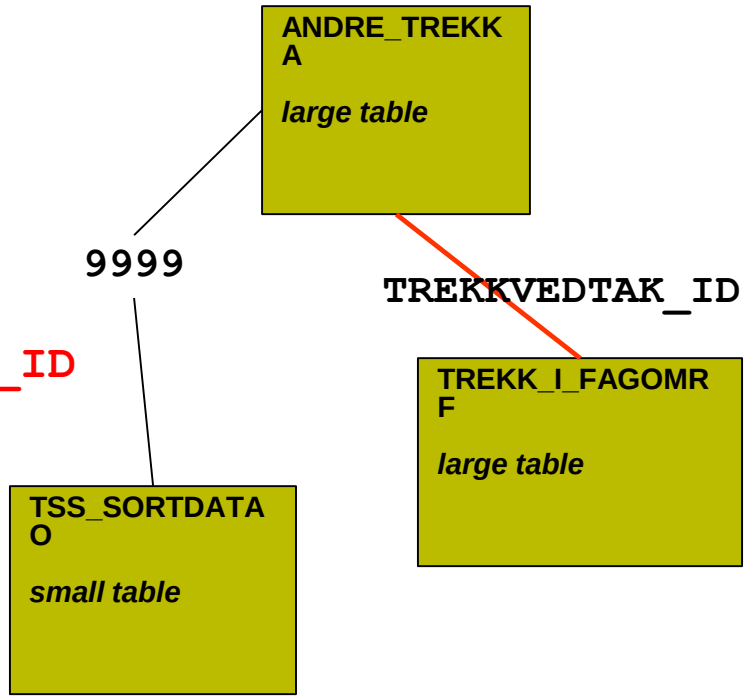
- Real volumes, meaning 5-25 millions in A & F
- 15 CPU hours

2. Tune the SW – b. efficient SQL
Example 1 - answer

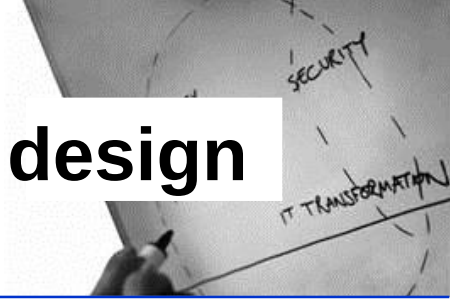



```
DECLARE C_TREKKDATA_3 CURSOR FOR
SELECT DISTINCT A.KREDITORS_REF
, A.KODE_TREKKALT
, A.SATS
, A.BELOP_SALDOTREKK
, A.BELOP_TRUKKET
, A.DATO_OPPFOLGING
, O.TSS_OFFNR
FROM V1_ANDRE_TREKK A
, V1_TREKK_I_FAGOMR F
, V1_TSS_SORTDATA O
WHERE A.TREKKVEDTAK_ID = :H
AND A.LOPENR = 9999
AND :H = 9999
AND F.TREKKVEDTAK_ID = A.TREKKVEDTAK_ID
AND F.KODE_FAGOMRAADE = "IT26"
AND O.KREDITOR_ID_TSS = :J
AND O.LOPENR = 9999
FOR FETCH ONLY
```

- Same volumes, almost same select
- 3 CPU seconds

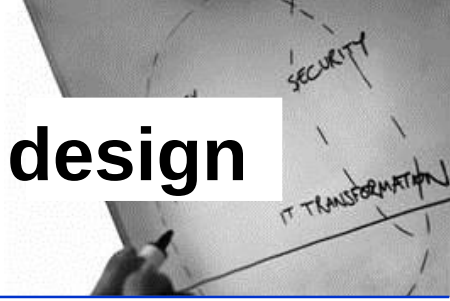


2. Tune the SW – c. Efficient code and design Introduction



- Now we have looked into how to how to make the SQL to execute more efficient
- Still, the DBMS has to execute the SQLs sent to it.
- Next focus should be to reduce the numbers of calls to SQL. (Remember the Axe Law: Don’t use it if you don’t mean it).
- Note: In a large project, this must be conveyed to the designers and the programmers early on. May be expensive to remove general problems afterwards.

2. Tune the SW – c. Efficient code and design
The post number lookup

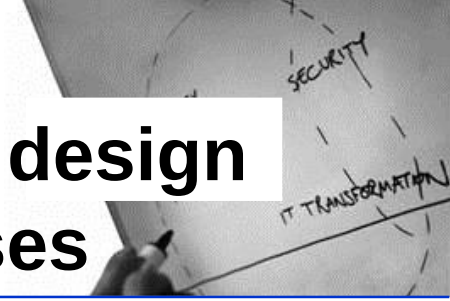


- Do not read over and over again the same value from the DB.
- Example: Verifying address information from 4 million customers.
- Reading the post number table pr customer record -> 4 million reads.
- This specific read may take 1-1,5 hours of a large run.

- Read the whole post number table into memory. 10.000 reads, after a short time a multiple page read (40 pages – 2 IOs of 50 ms) -> 0,1 second.
- In reality the difference will be much smaller, post number table could be pinned in Keep Buffer. But still you have to invoke the DBMS subsystem, with some 10 000 CPU instructions, as compared to a internal table read.

- The difference is virtually null on small volumes (on which the programmer typically test), on large volumes the difference is rather inconvenient.

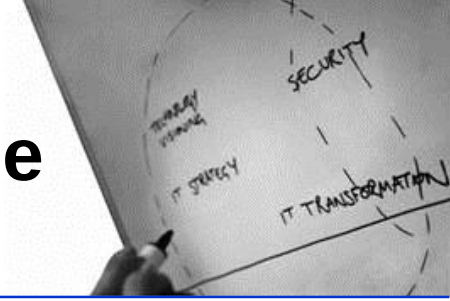
2. Tune the SW – c. Efficient code and design
Some final words - Solid state databases



- Much of current DBA wisdom is to reduce the number of physical gets, due to the fact that disks are order of magnitude slower than RAM.
- The most popular disk of the 1980's was the refrigerator-sized 3380 disks, which contained only 1.2 gig of storage at the astronomical cost of over $200,000. In today's 2012 dollars, disk in the 1980's costs more than $5,000 per megabyte.
- ~~Today, you can buy 100 GB disks for $100, and 100 GB of RAM Disk (solid-state disk) for $100,000.~~ (This was 2007)
- Today, you can buy 3 TB disks for $130, and 100 GB of RAM Disk (solid-state disk) for $100. (nnn.no = 120 GB SSD, 699 kr)
- Meaning, current wisdom regarding IO time is not valid.
- In this environment, **the focus is to reduce the number of logical reads** (and to reduce CPU), since the systems now are CPU constrained.

- (We are close to this unknowingly, due to the fact that many high scale disk cabinets have 50GB or more disk cache, and we typically operate with a cache hit rate of 95-99,5%).

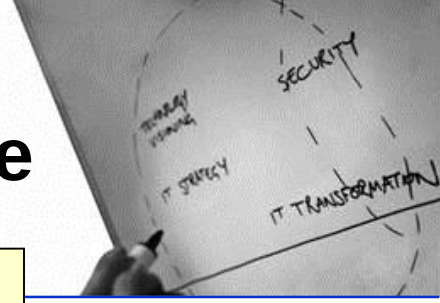
3. SOA, Object orientation and distance

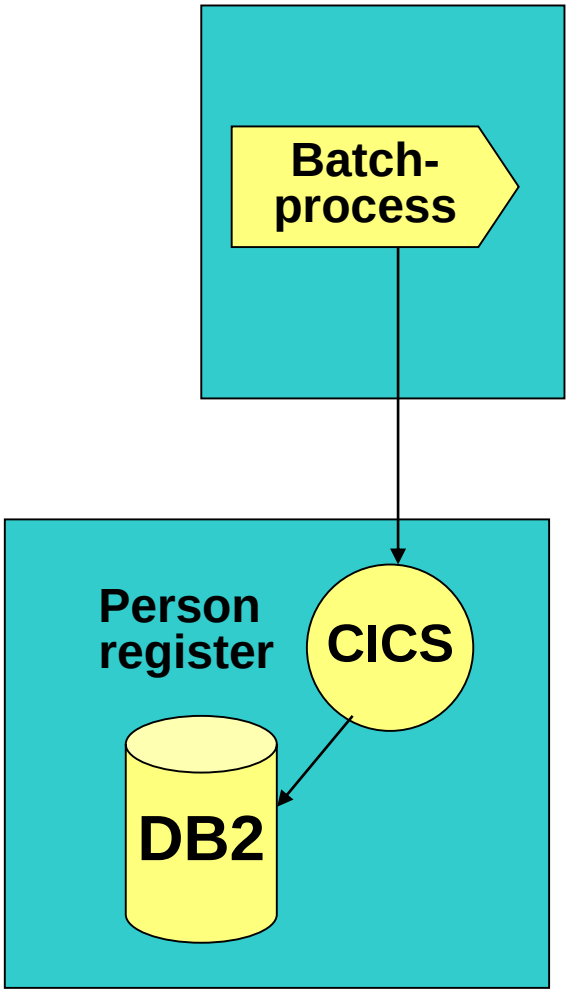


- In SOA, you present services. This call gives you for instance “all product information on customer x”. It returns an object, which the code manipulates.
- What may be hidden for the developer, is that this makes 50 database calls to the system’s own database, it performs 5 calls to other systems, each with their fair amount of database calls, and if you are lucky, an out of the house call to an external credit rating company. All in all, it takes 5-6 seconds for a normal private customer.
- What if Statoil is the customer?

- This distance is correct object orientation. If a developer has an object and methods that work correctly, he/she shall not worry about the implementation of these objects.
- But on the other hand, for performance it is important to know the underlying infrastructure (both software and hardware).
- My favourite quote: “System X is but a property in my parameter file”….

3. SOA, Object orientation and distance Example



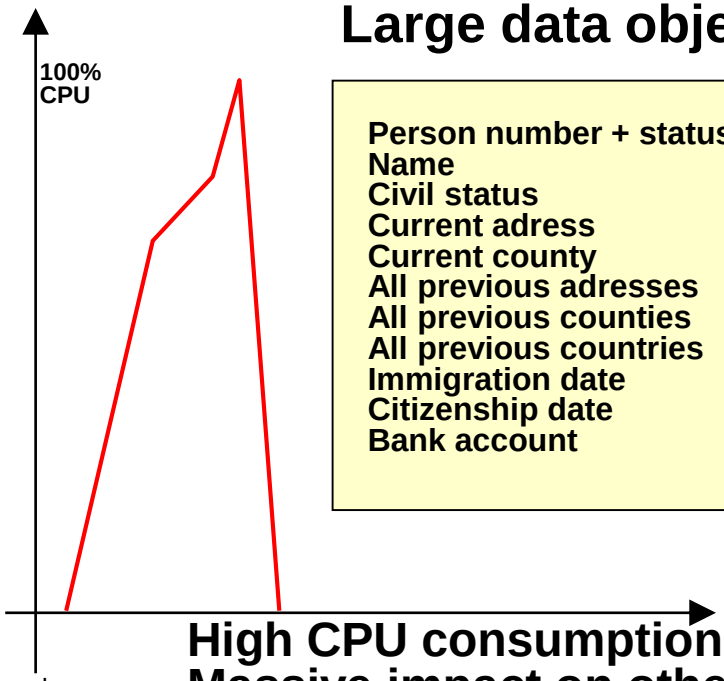


Person number + status
Name
Civil status
Current adress
Current county
All previous adresses
All previous counties
All previous countries
Immigration date
Citizenship date
Bank account

Phone number
Mobile number
Email adress
Foreign stays
Person status
Description of person status
Incapable and date
Filial used
and 7 more

95% is thrown away in the Batch-process

Large data object is created

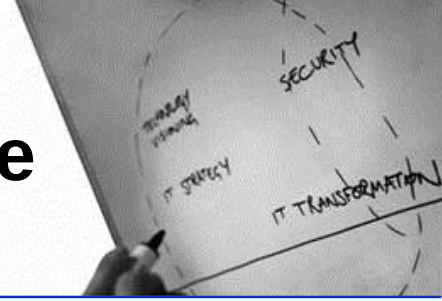


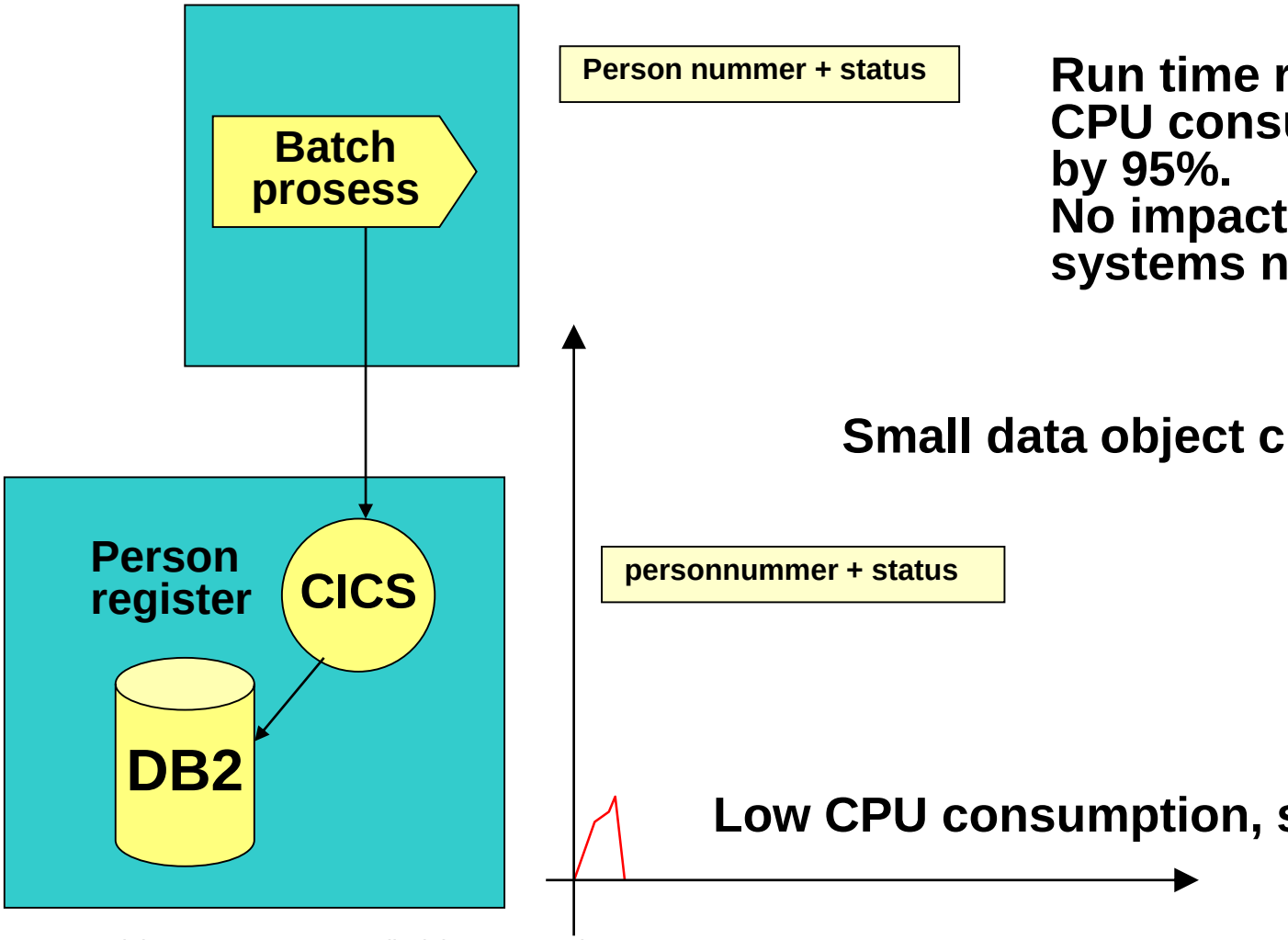
Person number + status
Name
Civil status
Current adress
Current county
All previous adresses
All previous counties
All previous countries
Immigration date
Citizenship date
Bank account

Phone number
Mobile number
Email adress
Foreign stays
Person status
Description of person status
Incapable and date
Filial used
and 7 more

High CPU consumption on DB server, Massive impact on other systems

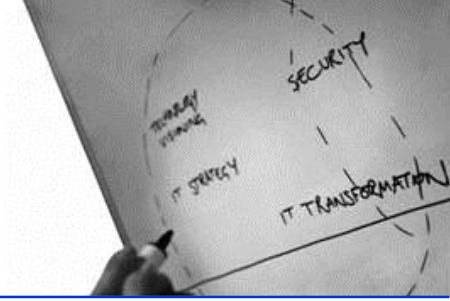
3. SOA, Object orientation and distance
New solution





Run time reduced 90%
CPU consumption reduced by 95%.
No impact on other systems now.

4. New directions



The traditional relational database is very good for certain operations, and not so good for other:

Excells in:

•Finding one or few rows via indexes. (That is often pre-defined searches).

•Transactional handling, for instance flight booking, concert ticketing.

•Storing structured data in a space efficient way.

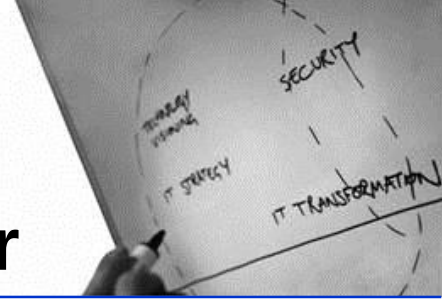
Not so good in:

•Searching through large data volumes with joins through multiple tables. (Analytics).

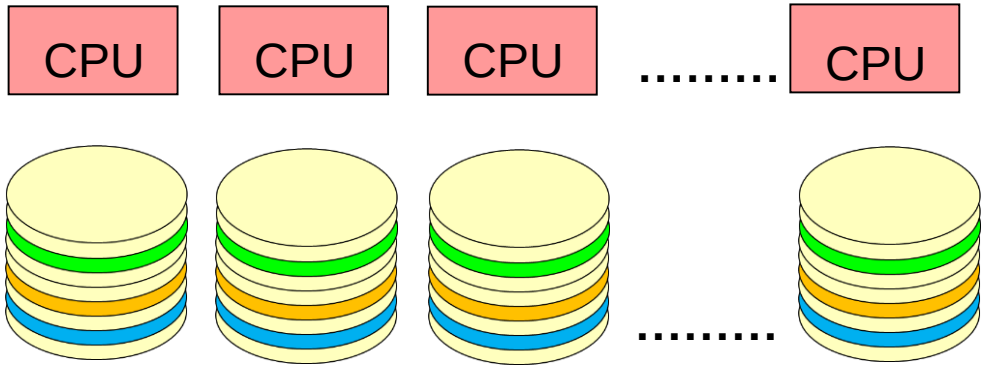
•Storing less structured data. (Comments in a blog, or Facebook).

•Storing and retrieving large volumes of read only data.

4. New directions
SQL accellerators - divide and conquer



Lots of CPUs with their own disc rack



TableA

TableB

TableC

All SQLs are executed as table scans.

Data for all join tables are fetched in 1 scan.

The tables are copies of the actual operational database

Different strategies for synchronisation:

- Daily complete load.
- Continous synch

Tables are distributed (striped) over a large number of disc racks.

Advantage: Speed.

Before	-->>	After
16 hours		39 seconds
1 hour		8 seconds
∞		1:30 min:sec

Disadvantage:

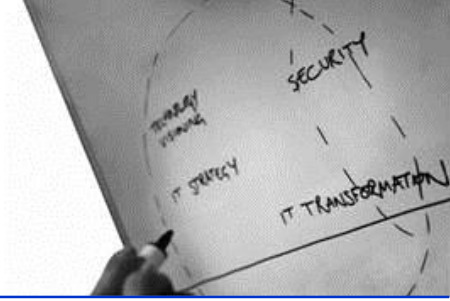
0,001 sec	1-2 seconds

I work currently with a SQL accellerator with 80 CPUs….though the range is

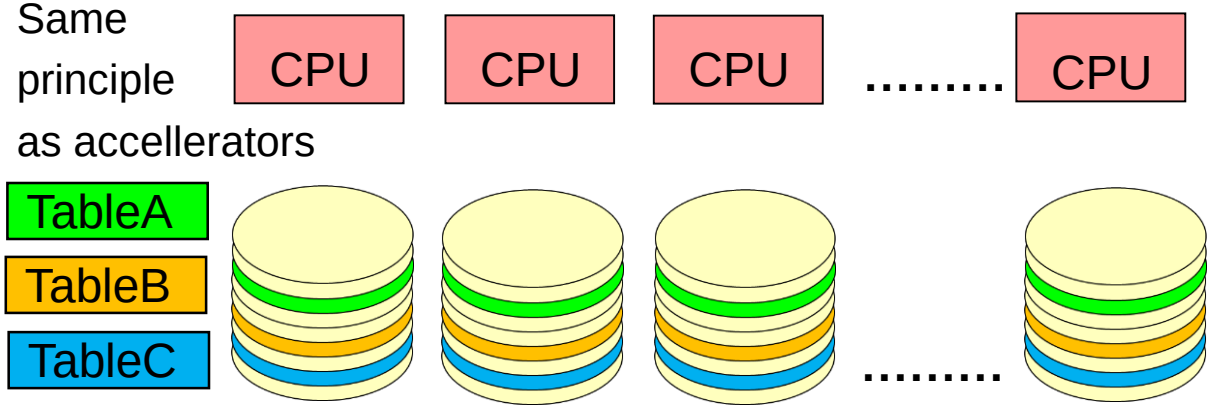
40 – 80 – 140 – 280 – 560 – 1120

CPUs and corresponding disc racks

4. New directions
NoSQL – or NOSQL?



Same principle as accellerators



Not all data you want to store is higly structured and tabular.

Not all data is strictly transactional and must be persisted in an all or nothing strategy.

Not all data is of a type where you need 100% consistency control.

Not all data is write / update / delete. A lot is write once, read often.

No SQL or
Not Only SQL

Many different principles and solutions.

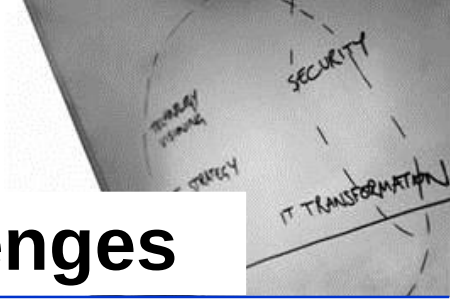
http://nosql-database.org

NoSQL

Examples:
Hadoop, Cassandra
MongoDB, GenieDB

Traditional applications where NoSQL may help: Payments archive in bank. Electricity metering. And many more, the industry is held back by traditional thinking.....

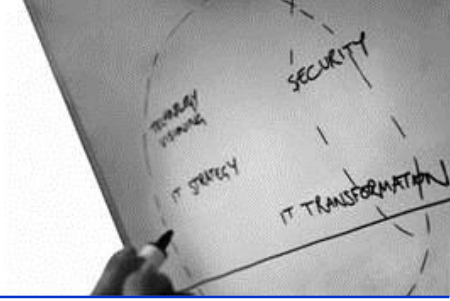
New applications: Social medias, mass data monitoring, data which is not updated, rather reentered (exam results? And many others). And where strict transactional control is not the issue.



5. Performance engineering
PE through the project phases - challenges

Analysis	Design	Build	Test	Run
PROBLEMS	**PROBLEMS**	**PROBLEMS**	**PROBLEMS**	**PROBLEMS**
1. Assumed High Performance 2. No EXPLICIT definition of performance requirements	1. Design independent of performance requirement 2. Lack knowledge of Data volumes and scalability 3. Design reviews do not focus on performance	1. Lack of coding guidelines 2. Code profiling not part of Build 3. Lack of expertise in profiler tools usage 4. Code reviews do not focus on Performance	1. Lack of explicit Performance test plan 2. Lack of test environment / test data to simulate production load 3. Insufficient Load Stress Stability tests 4. No Explicit performance sign offs due to lack of clear requirements	1. Lack of proactive monitoring 2. Critical problems in production 3. High time to resolve critical problems 4. Lack of expertise in monitoring tools 5. Risk with credibility / possible penalties

1. Introduction
Basic arithmetic



0.000 sec x 6,000,000 = 0 sec

Many designers think this is the database speed

0.012 sec x 6,000,000 = 72000 sec = 20 hours

And this may be the real database speed