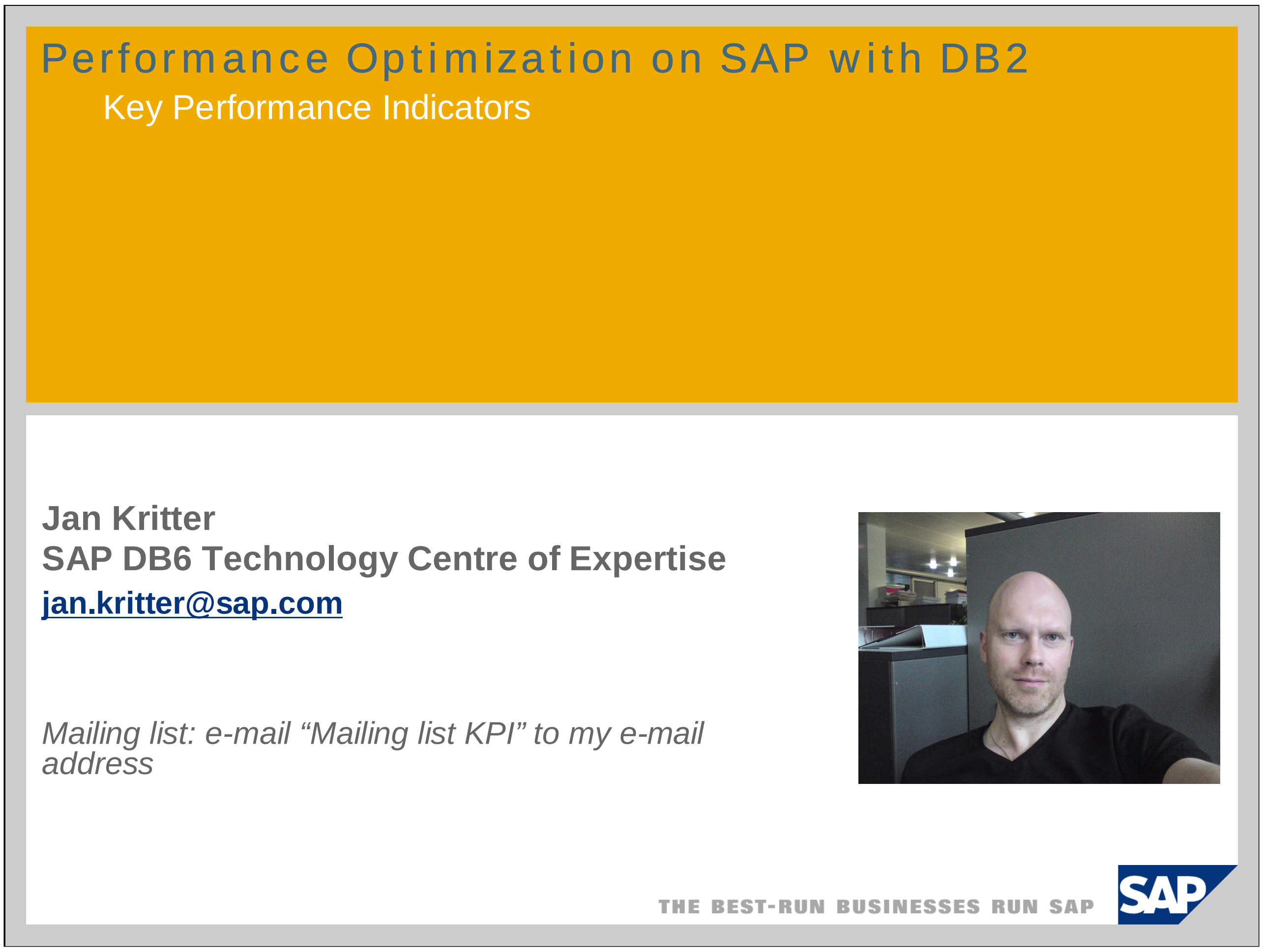
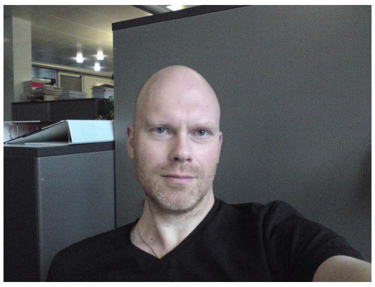


Performance Optimization on SAP with DB2

Key Performance Indicators

Jan Kritter
SAP DB6 Technology Centre of Expertise
jan.kritter@sap.com

Mailing list: e-mail "Mailing list KPI" to my e-mail address



THE BEST-RUN BUSINESSES RUN SAP

Short presentation of myself:

- databases since 1996
- starting 2000: DB6 (development support, defect support, SAP @ IBM Toronto Lab)
- since May 2005: CoE – DB6 performance

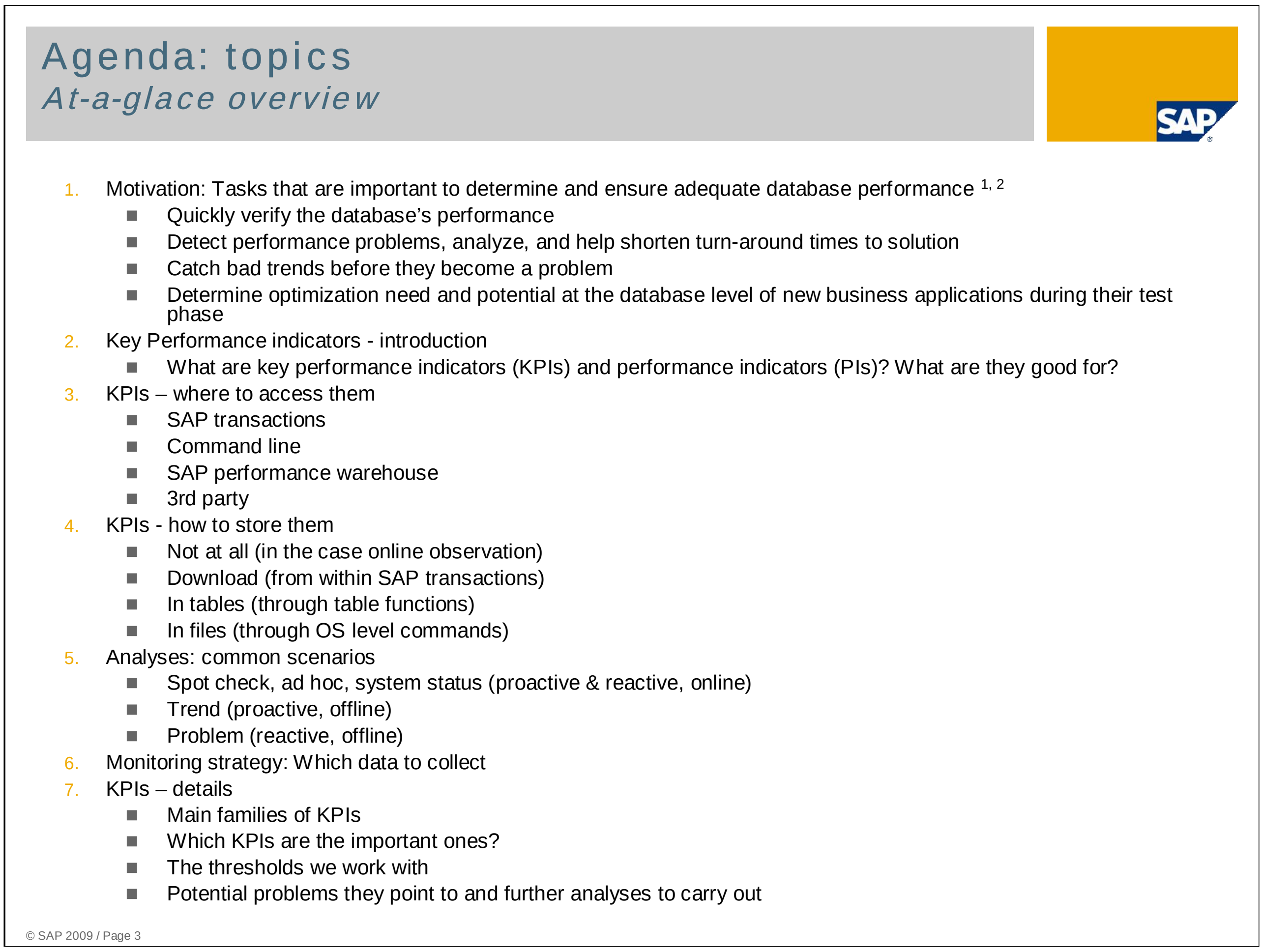
Possibility to contact me via e-mail, although I might not have the time to answer right away

- comments, questions, suggestion welcome

MAILING LIST: KPIs and monitoring and performance tuning related to them is a huge area and hence, this presentation's contents are still a work in progress; anybody interested can send me an e-mail to be notified of further developments, such as monitoring scripts, new and extended versions of this presentation, possibly a discussion group to be created and others…

Overview

- Performance is key, response time & throughput are critical to decide if the following are met or missed:
  - Service level agreements
  - User expectations
  - Smooth operation of your business
- KPIs help to isolate areas with sub-optimal performance and hence enable you to:
  - Quickly verify the database performance's health
  - Catch trends before they become a problem
  - Determine optimization potential of new business applications
  - Help shorten turn-around times to problem solution if a problem occurs
- Value proposition:
  - Nuts and bolts on what KPIs to pay attention to most
  - How to proactively and reactively analyze your system and tune it accordingly
- You learn
  - Which database performance indicators are the most important ones for performance monitoring and tuning
  - What they mean from a technical perspective
  - What influences them and what in turn is affected by them
  - How to monitor them
  - What thresholds typically work well for SAP systems



Agenda: topics

At-a-glace overview

1. Motivation: Tasks that are important to determine and ensure adequate database performance [1, 2]
   - Quickly verify the database's performance
   - Detect performance problems, analyze, and help shorten turn-around times to solution
   - Catch bad trends before they become a problem
   - Determine optimization need and potential at the database level of new business applications during their test phase
2. Key Performance indicators - introduction
   - What are key performance indicators (KPIs) and performance indicators (PIs)? What are they good for?
3. KPIs – where to access them
   - SAP transactions
   - Command line
   - SAP performance warehouse
   - 3rd party
4. KPIs - how to store them
   - Not at all (in the case online observation)
   - Download (from within SAP transactions)
   - In tables (through table functions)
   - In files (through OS level commands)
5. Analyses: common scenarios
   - Spot check, ad hoc, system status (proactive & reactive, online)
   - Trend (proactive, offline)
   - Problem (reactive, offline)
6. Monitoring strategy: Which data to collect
7. KPIs – details
   - Main families of KPIs
   - Which KPIs are the important ones?
   - The thresholds we work with
   - Potential problems they point to and further analyses to carry out

1: The new 9.7 monitoring approach of wait events is explicitly excluded here – it will eventually be addressed in another session

2: Only level 0 and 1 are shown, deeper levels have been neglected here and will be shown and addressed on the more detailed slides only

Agenda: topics overview

1. Motivation for this session: Tasks that enable you to determine and ensure adequate database performance
2. Key Performance indicators - introduction
3. KPIs – where to access them
4. KPIs - how to store them
5. Analyses: common scenarios
6. Monitoring strategy: Which data to collect
7. KPIs – details

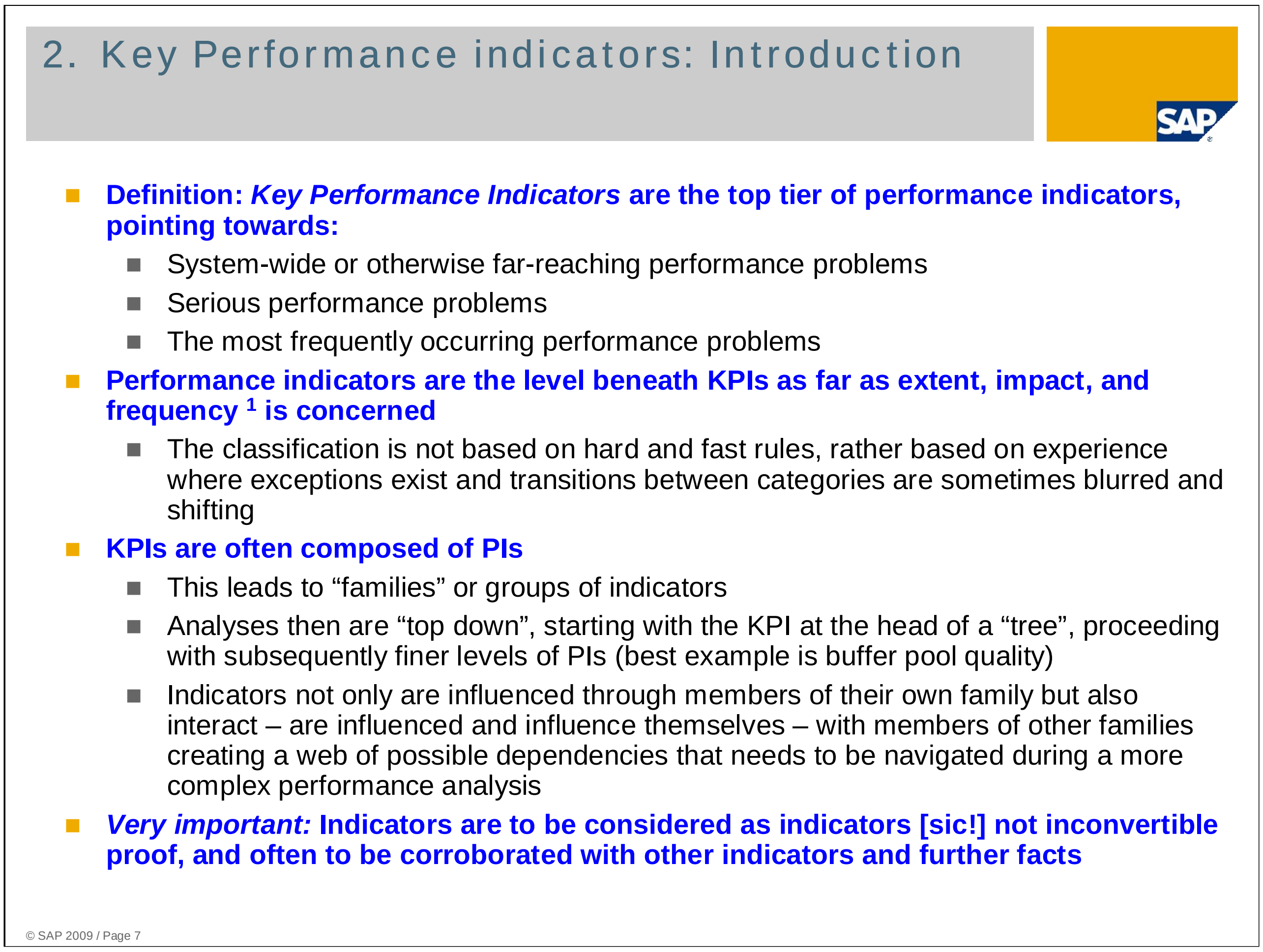
1. Motivation: Tasks that enable you to determine and ensure adequate database performance (I)

- **Quickly verify the database’s performance**
  - Check the database’s performance with one look: use a one-line SQL statement to collect all KPIs (easy to copy and paste e.g. into transaction ST04 – Diagnostics – SQL Commands)
  - Check for expensive SQL commands currently in the SQL cache
- **Detect performance problems, analyze, and help shorten turn-around times to solution: ad hoc analysis**
  - First, get a quick overview at-a-glance (cf. above)
  - If KPI values point towards the existence of performance problems, drill down into the indicator family / families for which KPIs are found to be out of bounds using SQL table functions on dbm, db, buffer pool, table space, table, and SQL
- **Catch bad trends before they become a problem**
  - Regularly collect KPIs’ values and check their evolution
  - For further, more comprehensive analyses check historic data, e.g. repeatedly collected dbm, db, buffer pool, table space, table, and SQL data

1. Motivation: Tasks that enable you to determine and ensure adequate database performance (II)

- **Determine optimization need and potential at the database level of new business applications during their test phase**
  - Run the application while you collect data:
    - For an overview or especially if the application runs only a short time and executes a small number of steps: a KPI snapshot at the beginning and at the end of the application's execution
    - For more in-depth data also collect dbm, db, buffer pool, table space, table, and SQL data
    - For finer grained data, especially if the application runs for a long time and is complex collect both the KPIs and table function output repeatedly as often as every few seconds

      NB: be very careful to space out subsequent data collections such that they do not overlap (i.e. one is still running when the next is already kicked off) nor that they produce unacceptable load, hence performance overhead. This requires to carefully test the data collector on a representative (regarding workload, data volume, and available resources) system.



2. Key Performance indicators: Introduction

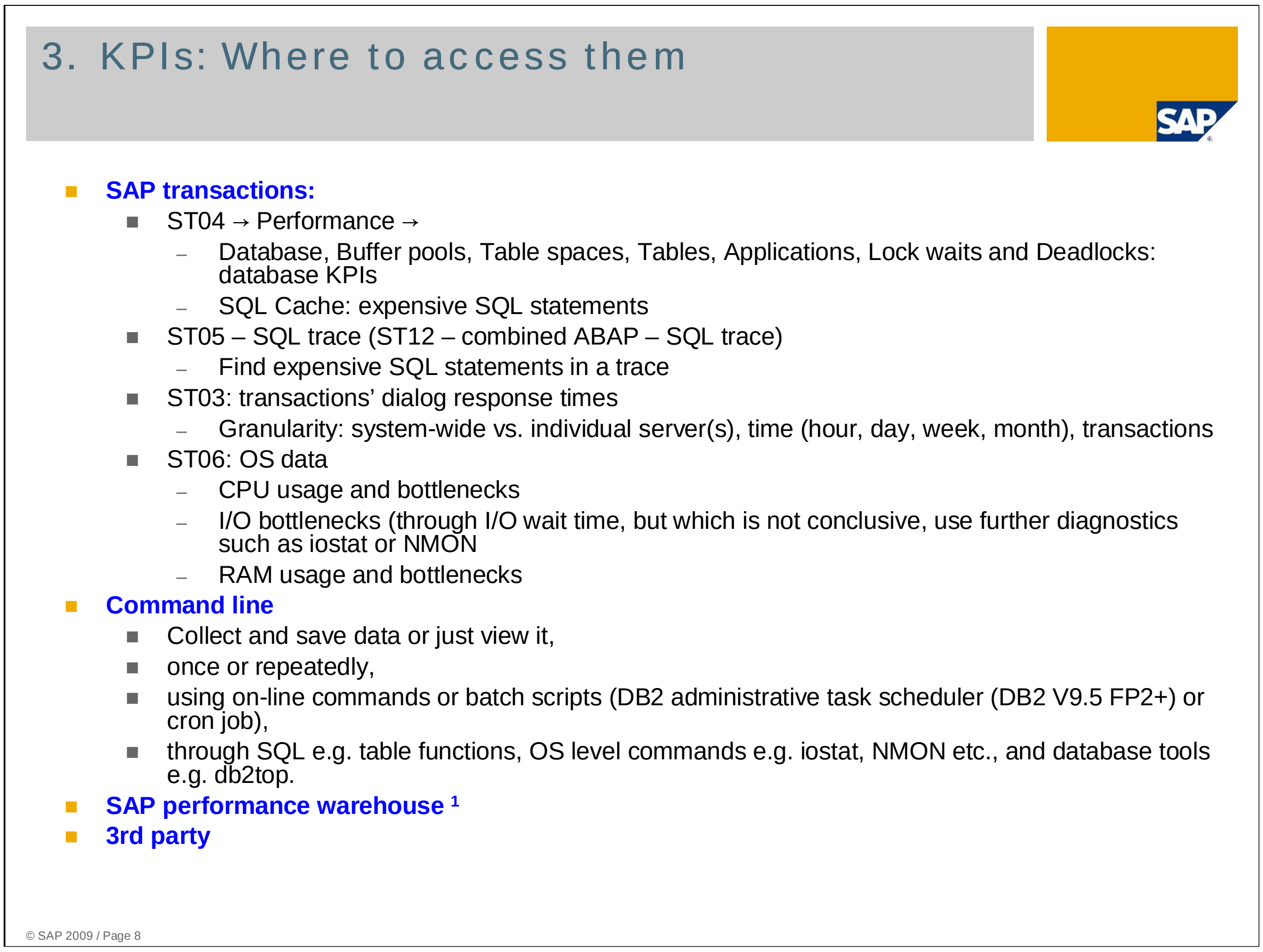
- **Definition: *Key Performance Indicators* are the top tier of performance indicators, pointing towards:**
  - System-wide or otherwise far-reaching performance problems
  - Serious performance problems
  - The most frequently occurring performance problems
- **Performance indicators are the level beneath KPIs as far as extent, impact, and frequency [1] is concerned**
  - The classification is not based on hard and fast rules, rather based on experience where exceptions exist and transitions between categories are sometimes blurred and shifting
- **KPIs are often composed of PIs**
  - This leads to “families” or groups of indicators
  - Analyses then are “top down”, starting with the KPI at the head of a “tree”, proceeding with subsequently finer levels of PIs (best example is buffer pool quality)
  - Indicators not only are influenced through members of their own family but also interact – are influenced and influence themselves – with members of other families creating a web of possible dependencies that needs to be navigated during a more complex performance analysis
- ***Very important:* Indicators are to be considered as indicators [sic!] not inconvertible proof, and often to be corroborated with other indicators and further facts**

1:

Extent: how widely the system is affected (the system as a whole vs. localized)

Impact: how strongly performance is affected

Frequency: how often (according to our experience) the problem occurs



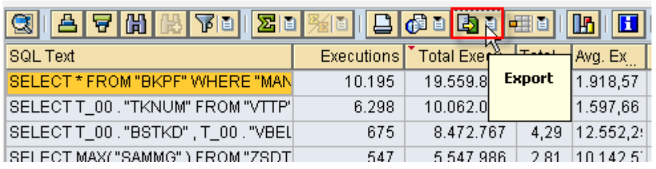
3. KPIs: Where to access them

- **SAP transactions:**
  - ST04 → Performance →
    - Database, Buffer pools, Table spaces, Tables, Applications, Lock waits and Deadlocks: database KPIs
    - SQL Cache: expensive SQL statements
  - ST05 – SQL trace (ST12 – combined ABAP – SQL trace)
    - Find expensive SQL statements in a trace
  - ST03: transactions' dialog response times
    - Granularity: system-wide vs. individual server(s), time (hour, day, week, month), transactions
  - ST06: OS data
    - CPU usage and bottlenecks
    - I/O bottlenecks (through I/O wait time, but which is not conclusive, use further diagnostics such as iostat or NMON
    - RAM usage and bottlenecks
- **Command line**
  - Collect and save data or just view it,
  - once or repeatedly,
  - using on-line commands or batch scripts (DB2 administrative task scheduler (DB2 V9.5 FP2+) or cron job),
  - through SQL e.g. table functions, OS level commands e.g. iostat, NMON etc., and database tools e.g. db2top.
- **SAP performance warehouse** [1]
- **3rd party**

1: Not addressed here, refer to the remote expert session on this topic by Steffen Siegmund (the developer of the SAP performance warehouse)

4. KPIs: How to store them

- Not at all (in the case online observation)
- Download (from within some SAP transactions e.g. SQL cache, buffer pools, table spaces, etc.)



SQL Text	Executions	Total Exe	Total	Avg. Ex...
SELECT * FROM "BKPF" WHERE "MAN	10.195	19.559.8	[illegible]	1.918,57
SELECT T_00 . "TKNUM" FROM "VTTP'	6.298	10.062.0	[illegible]	1.597,66
SELECT T_00 . "BSTKD" , T_00 . "VBEL	675	8.472.767	4,29	12.552,2
SELECT MAX("SAMMG") FROM "ZSDT	547	5.547.986	2,81	10.142,5

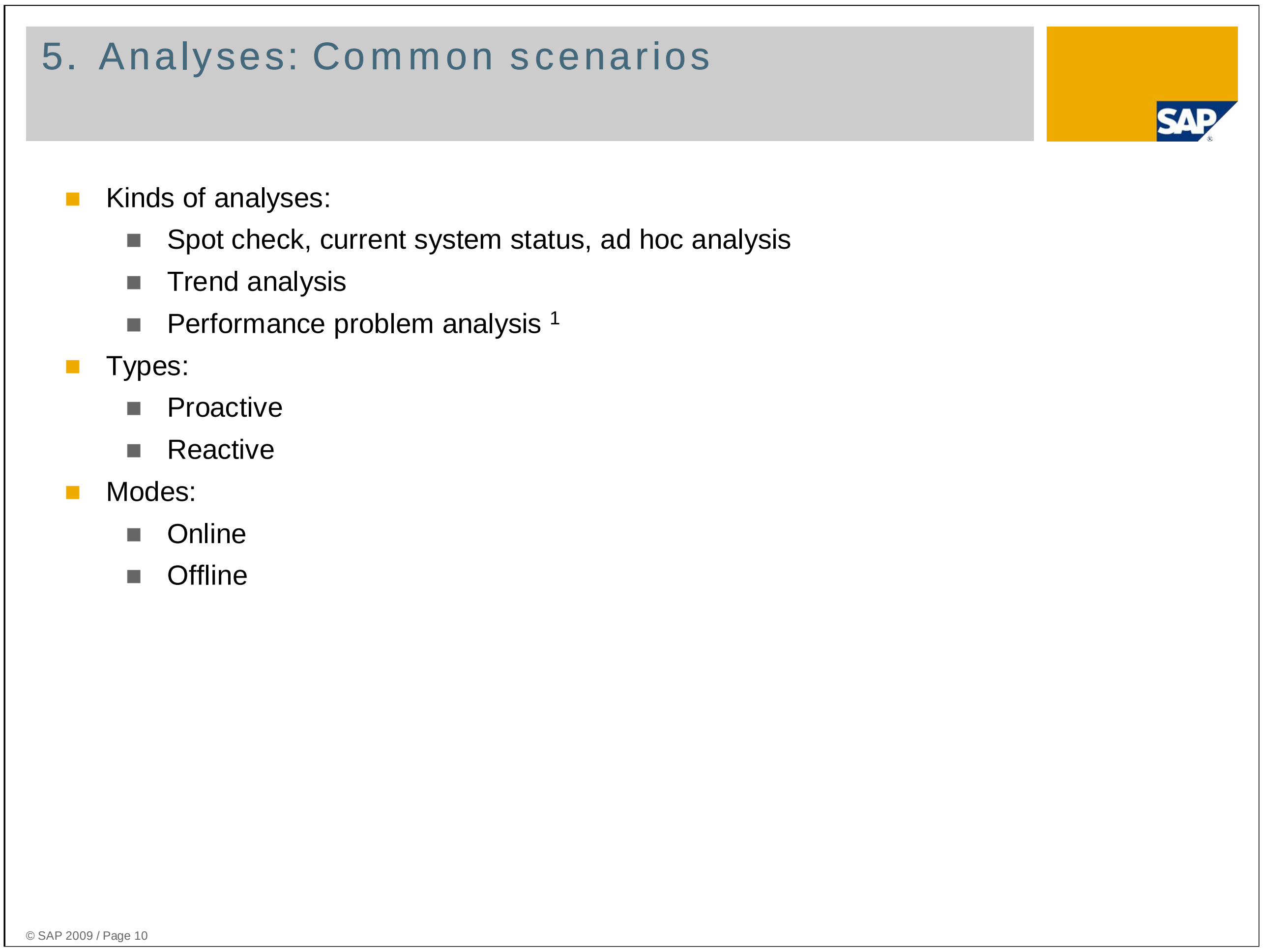
- In tables (through table functions)

```
# buffer pools

OBJECT=BP
SCHEMA=SAPMON
BP_TBL_NAME='SNAP_'`echo $OBJECT`'_'`date '+%G%m%d'`
BP_EXPORT_FILE=$TBL_NAME'.IXF'

db2 "create table $SCHEMA.$BP_TBL_NAME as (select * from table (SNAPSHOT_BP('$DB', -1)) as T_BP) with no data in $TS_NAME"
db2 "insert into $SCHEMA.$BP_TBL_NAME select * from table (SNAPSHOT_BP('$DB', -1)) as T_BP"
db2 "export to $BP_EXPORT_FILE of IXF select * from $SCHEMA.$BP_TBL_NAME"
```

- In files (through OS level commands) [1]

```
56> db2 get dbm cfg > dbmcfg.txt
57> nmon -f -s 10 -c 180 -p -Y -A -d -w 8 -K -^ -NN -P -V
```

1: db2support, _the_ data collector *par excellence* for DB6 defect problems is not used here, as it collects easily several hundred MB worth of data of which few are performance relevant, hence to specifically collect only performance-relevant data is more effective

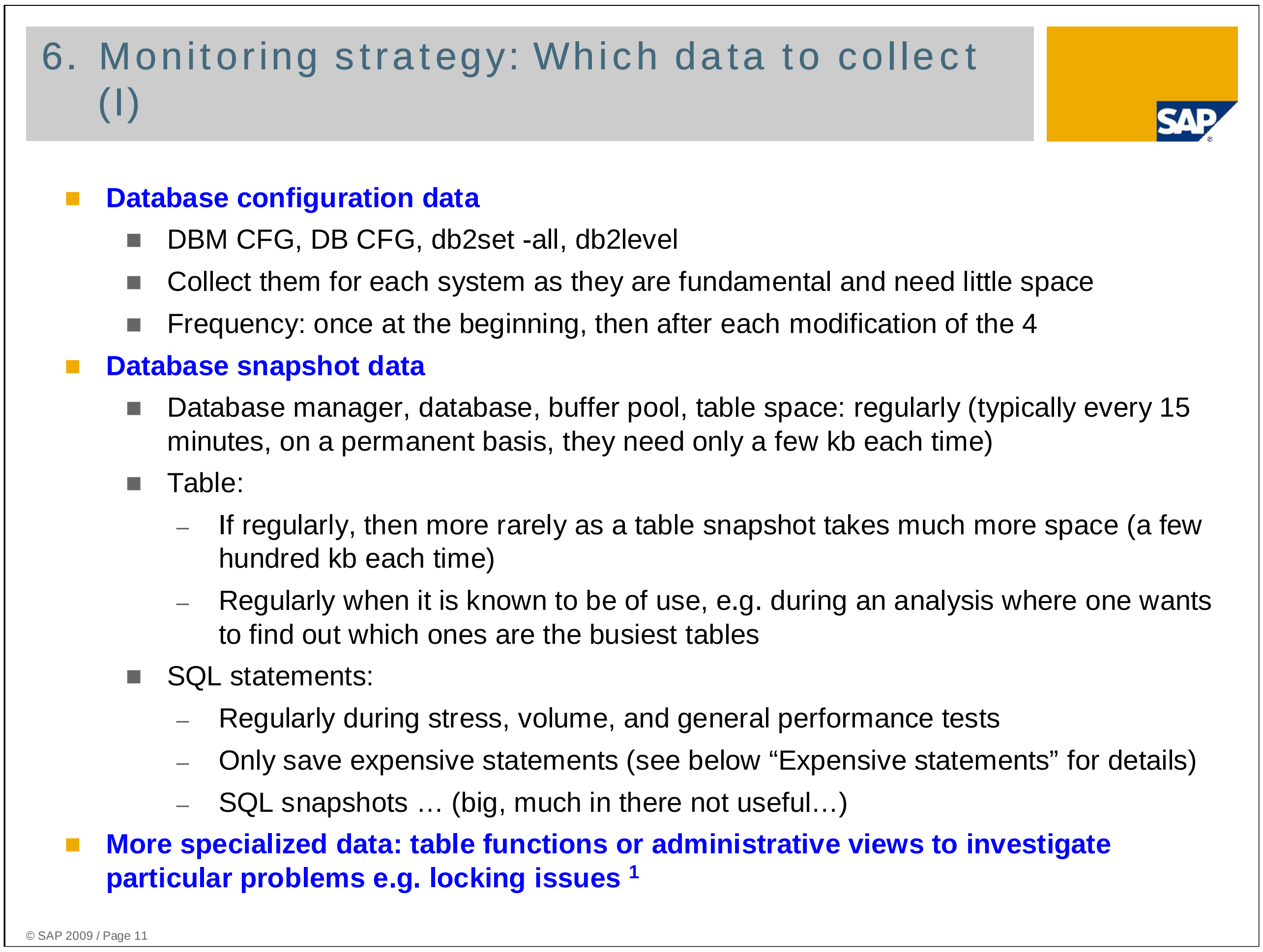


5. Analyses: Common scenarios

- Kinds of analyses:
  - Spot check, current system status, ad hoc analysis
  - Trend analysis
  - Performance problem analysis [1]
- Types:
  - Proactive
  - Reactive
- Modes:
  - Online
  - Offline

1: “ad hoc analysis” and “performance problem analysis” can be very similar in nature, the difference being made is that the former tends to be more spur-of-the-moment, online, in-situ, less deep, the latter more premeditated, using offline data, deeper-going.

Again, this is more a general classification with often blurring boundaries, especially the former often leading to the latter, sort of to determine through a spot check whether a performance problem might exist, seeing a possible one carrying on with a somewhat more thorough ad hoc analysis and upon finding more indicators for a problem starting a full-blown in-depth performance problem analysis.



6. Monitoring strategy: Which data to collect (I)

- **Database configuration data**
  - DBM CFG, DB CFG, db2set -all, db2level
  - Collect them for each system as they are fundamental and need little space
  - Frequency: once at the beginning, then after each modification of the 4
- **Database snapshot data**
  - Database manager, database, buffer pool, table space: regularly (typically every 15 minutes, on a permanent basis, they need only a few kb each time)
  - Table:
    - If regularly, then more rarely as a table snapshot takes much more space (a few hundred kb each time)
    - Regularly when it is known to be of use, e.g. during an analysis where one wants to find out which ones are the busiest tables
  - SQL statements:
    - Regularly during stress, volume, and general performance tests
    - Only save expensive statements (see below “Expensive statements” for details)
    - SQL snapshots … (big, much in there not useful…)
- **More specialized data: table functions or administrative views to investigate particular problems e.g. locking issues [1]**

1: These monitoring elements exist in the snapshots too, but here they are presented as collections of monitoring elements grouped for a specific purpose; which is why they are called administrative _views_ → DB2 online documentation “Snapshot monitor SQL Administrative Views“

6. Monitoring strategy: Which data to collect (II)

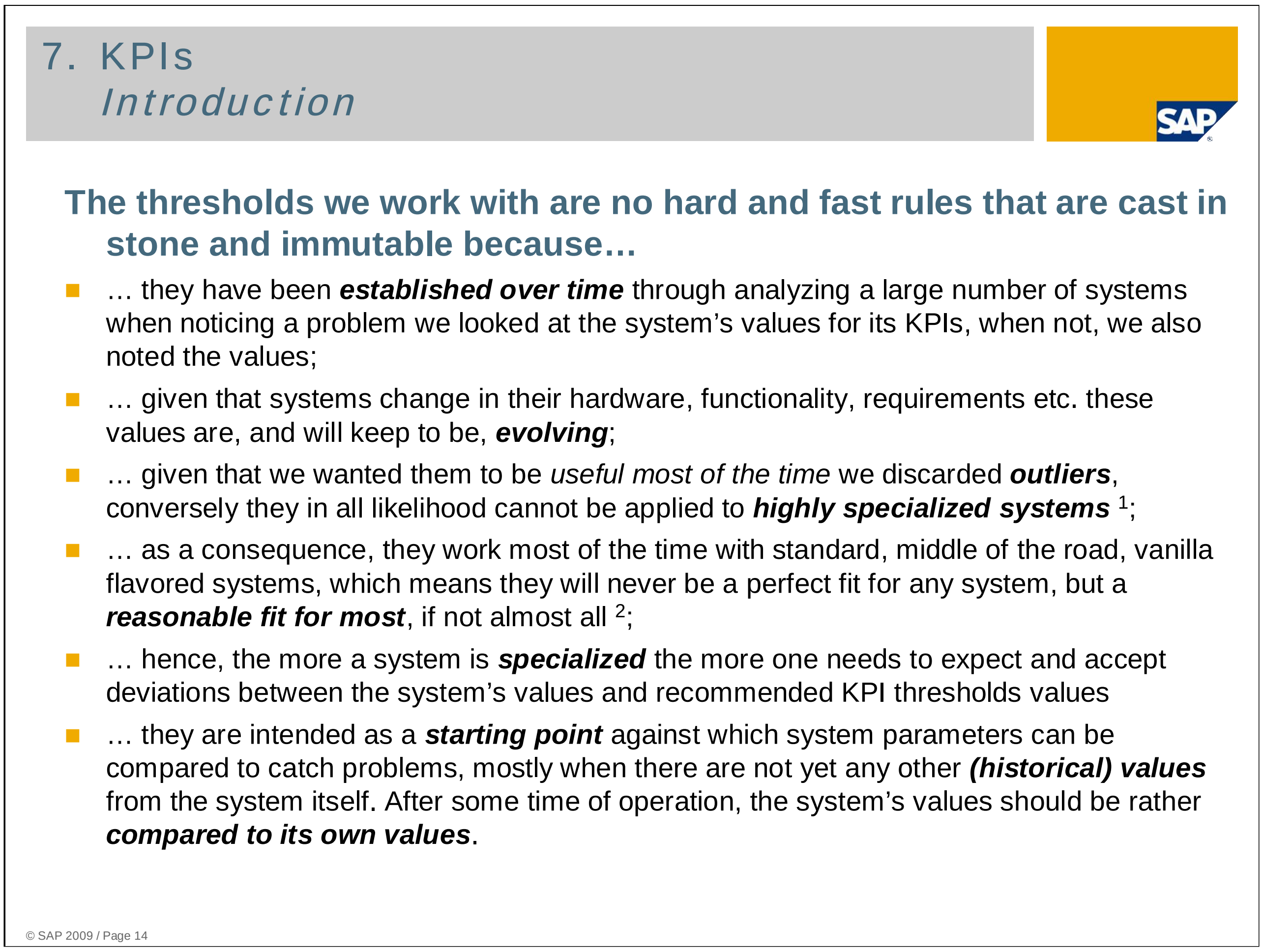
- **Storage sub-system activity data**
  - Storage sub-system bottlenecks are the most common performance bottlenecks together with expensive application code and expensive SQL statements
  - For AIX and Linux the best tool is NMON (Nigel's monitor) and its analytical tool NMON Analyzer (which also transforms NMON data into very useful graphics
    - NMON allows in particular to monitor CPU, RAM, and I/O bottlenecks
    - A typical use is:
      `nmon –f -s 10 -c 180 -p -Y -A -d -w 8 -K -^ -NN -P -V`
  - On other systems iostat and vmstat can be used, but which are not as thorough or comfortable
- **Network data**
  - Rarely a problem but in case of a go-live of an application or system that needs significant bandwidth it should be established that the network will not be a bottleneck
- **SAP response times**
  - Total response time, CPU response time, database response time (→ transaction ST03)

- ***Note: System comparison*** – it may be beneficial to compare several different systems against another:
  - To identify situations where the combined workload of them produces bottlenecks
  - During times when one shows a performance trough
  - Shared resources such as same storage sub-system, CPU and RAM if they share an LPAR or if several LPARs share these resources

7. KPIs
Overview

In this section we look at the details of KPIs and related topics that must be taken into account to make full use of the information KPIs offer:

- **Introduction: on thresholds**
- **Performance indicators – absolute values, changes, and trends**
  - How performance indicators can be used
  - Interpretation of deviation in performance indicators
  - Change and constancy
  - Deviation: What is normal, what is not
  - When things are different: temporary changes vs. permanent deviations or changes
- **Triage: Before you analyze – how to decide what to analyze and what not**
- **KPIs: details**
  - The most important areas of performance problems
  - The details provided on the individual problems



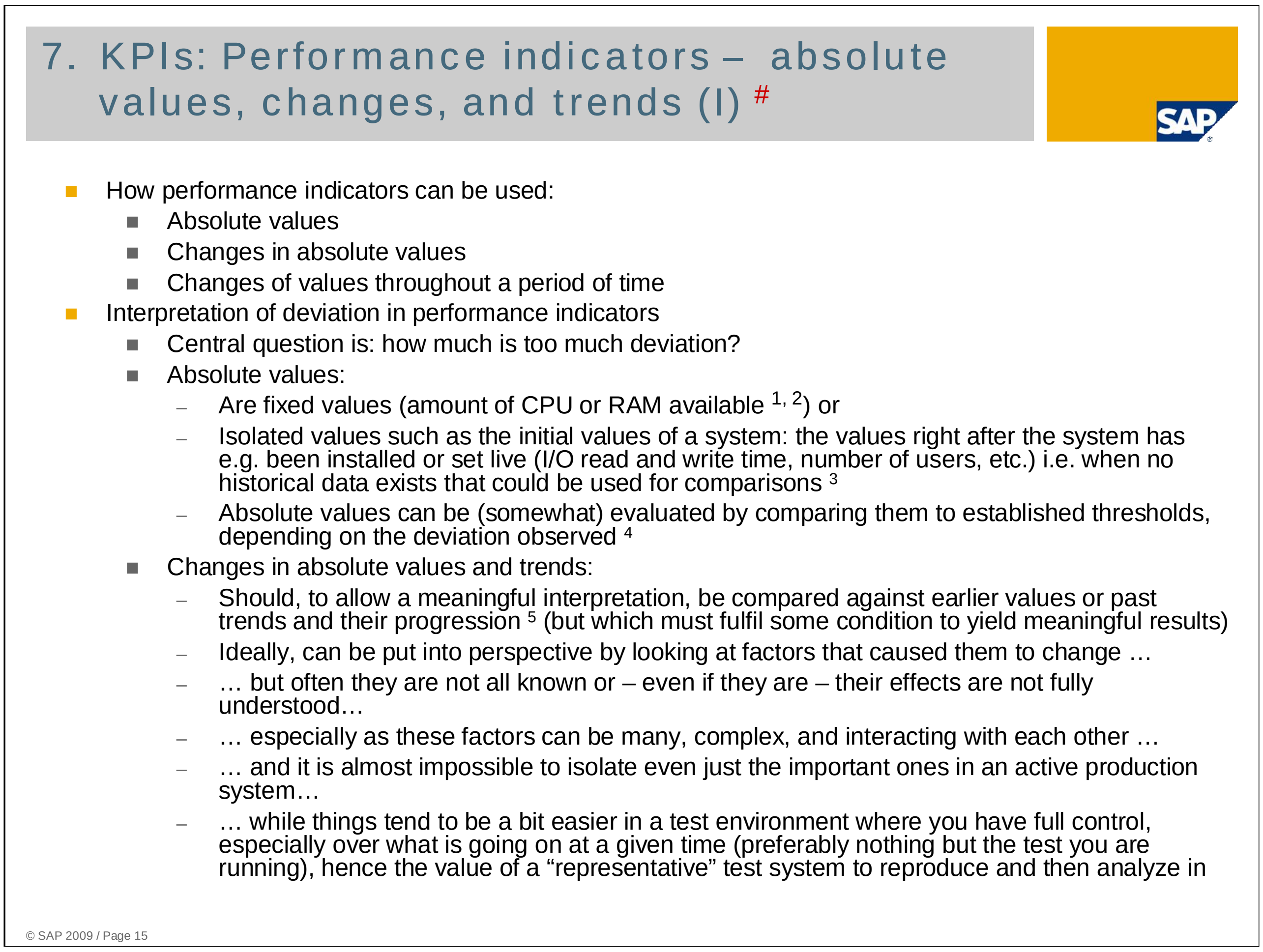
7. KPIs
Introduction

The thresholds we work with are no hard and fast rules that are cast in stone and immutable because…

- … they have been ***established over time*** through analyzing a large number of systems when noticing a problem we looked at the system’s values for its KPIs, when not, we also noted the values;
- … given that systems change in their hardware, functionality, requirements etc. these values are, and will keep to be, ***evolving***;
- … given that we wanted them to be *useful most of the time* we discarded ***outliers***, conversely they in all likelihood cannot be applied to ***highly specialized systems*** [1];
- … as a consequence, they work most of the time with standard, middle of the road, vanilla flavored systems, which means they will never be a perfect fit for any system, but a ***reasonable fit for most***, if not almost all [2];
- … hence, the more a system is ***specialized*** the more one needs to expect and accept deviations between the system’s values and recommended KPI thresholds values
- … they are intended as a ***starting point*** against which system parameters can be compared to catch problems, mostly when there are not yet any other ***(historical) values*** from the system itself. After some time of operation, the system’s values should be rather ***compared to its own values***.

1: Not necessarily as far as the application is concerned but as far as performance-influencing features like widely and abruptly varying workloads, exotic hardware

2: according to “Life can be described with a Gaussian profile” :)



7. KPIs: Performance indicators – absolute values, changes, and trends (I)

- How performance indicators can be used:
  - Absolute values
  - Changes in absolute values
  - Changes of values throughout a period of time
- Interpretation of deviation in performance indicators
  - Central question is: how much is too much deviation?
  - Absolute values:
    - Are fixed values (amount of CPU or RAM available [1, 2]) or
    - Isolated values such as the initial values of a system: the values right after the system has e.g. been installed or set live (I/O read and write time, number of users, etc.) i.e. when no historical data exists that could be used for comparisons [3]
    - Absolute values can be (somewhat) evaluated by comparing them to established thresholds, depending on the deviation observed [4]
  - Changes in absolute values and trends:
    - Should, to allow a meaningful interpretation, be compared against earlier values or past trends and their progression [5] (but which must fulfil some condition to yield meaningful results)
    - Ideally, can be put into perspective by looking at factors that caused them to change …
    - … but often they are not all known or – even if they are – their effects are not fully understood…
    - … especially as these factors can be many, complex, and interacting with each other …
    - … and it is almost impossible to isolate even just the important ones in an active production system…
    - … while things tend to be a bit easier in a test environment where you have full control, especially over what is going on at a given time (preferably nothing but the test you are running), hence the value of a “representative” test system to reproduce and then analyze in

#: Slides 15 to 18 were not shown in the presentation but only referred to and are here included as promised to provide further information on KPIs

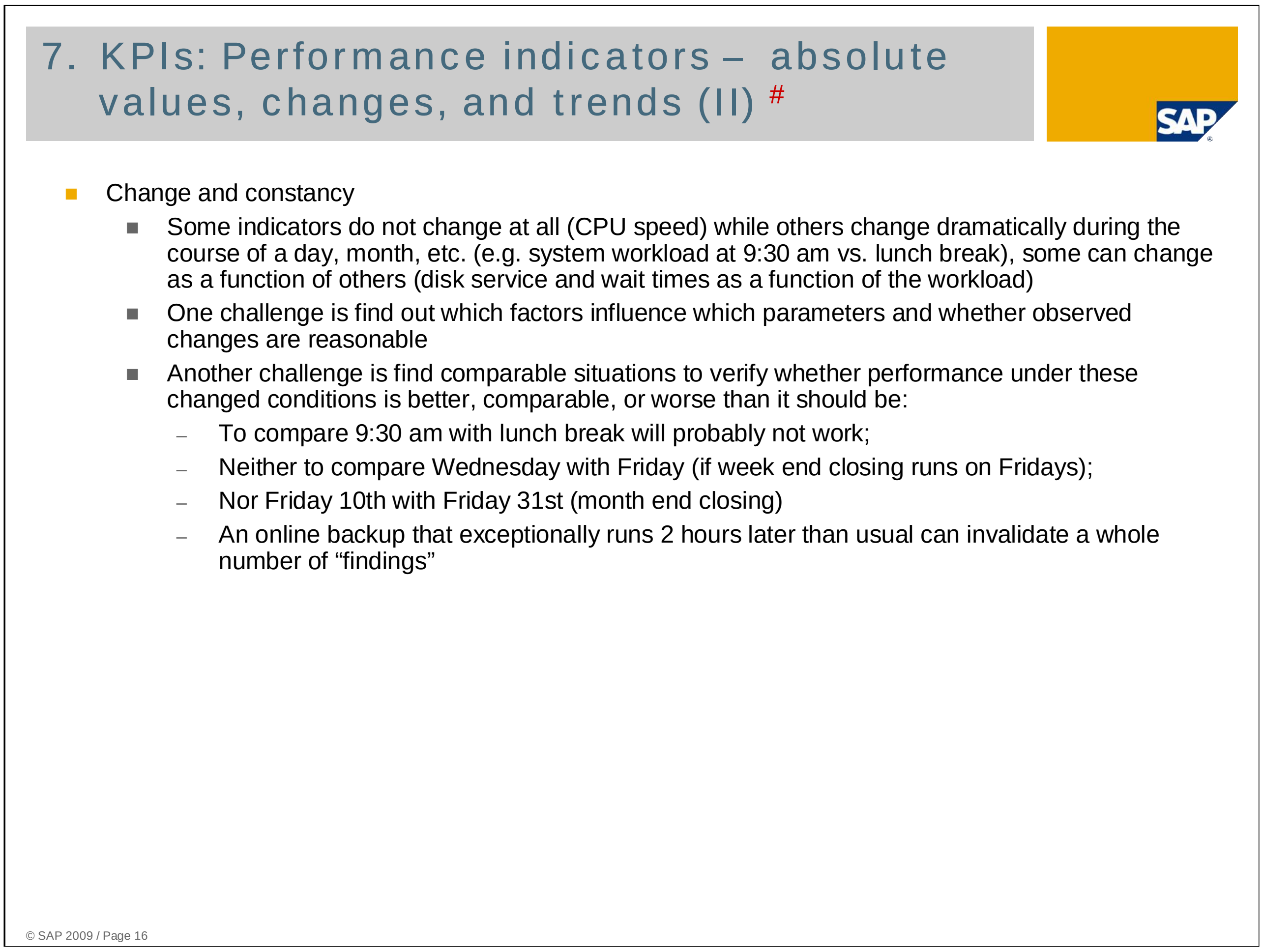
1: Which can even hold for LPARs using virtualization e.g. the maximum CPU entitlement or the CPU cap (if an LPAR is capped regarding CPU)

2: Fixed, that is, until they are changed by adding e.g. RAM

3: Which is why thresholds were created as crutches; consequently, once the system has been running for some time current values should be compared rather to history data rather than “one size fits all” general threshold values

4: These thresholds have been established over time through past analyses of systems, thus have proven by now to be useful in a (often rather extensive) number of contexts which makes them suitable as general values against which to measure systems. (First and foremost in the absence of anything better!) However, systems are vastly different which means that “one size fits all”, “middle of the road” values are usually helpful but not the best. Often they would turn out to be significantly off the mark (if the system and its performance were to be thoroughly analyzed), either too ambitious (because the present system perfectly tuned would still fall short of a threshold e.g. because of innate limitation such as available hardware resources or workload) or by far not ambitious enough (for the same reasons). The solution for this is, after a system runs stably for a while, to carry out an in depth analysis of its individual performance aspects and their influences to detect and realize further performance potential.

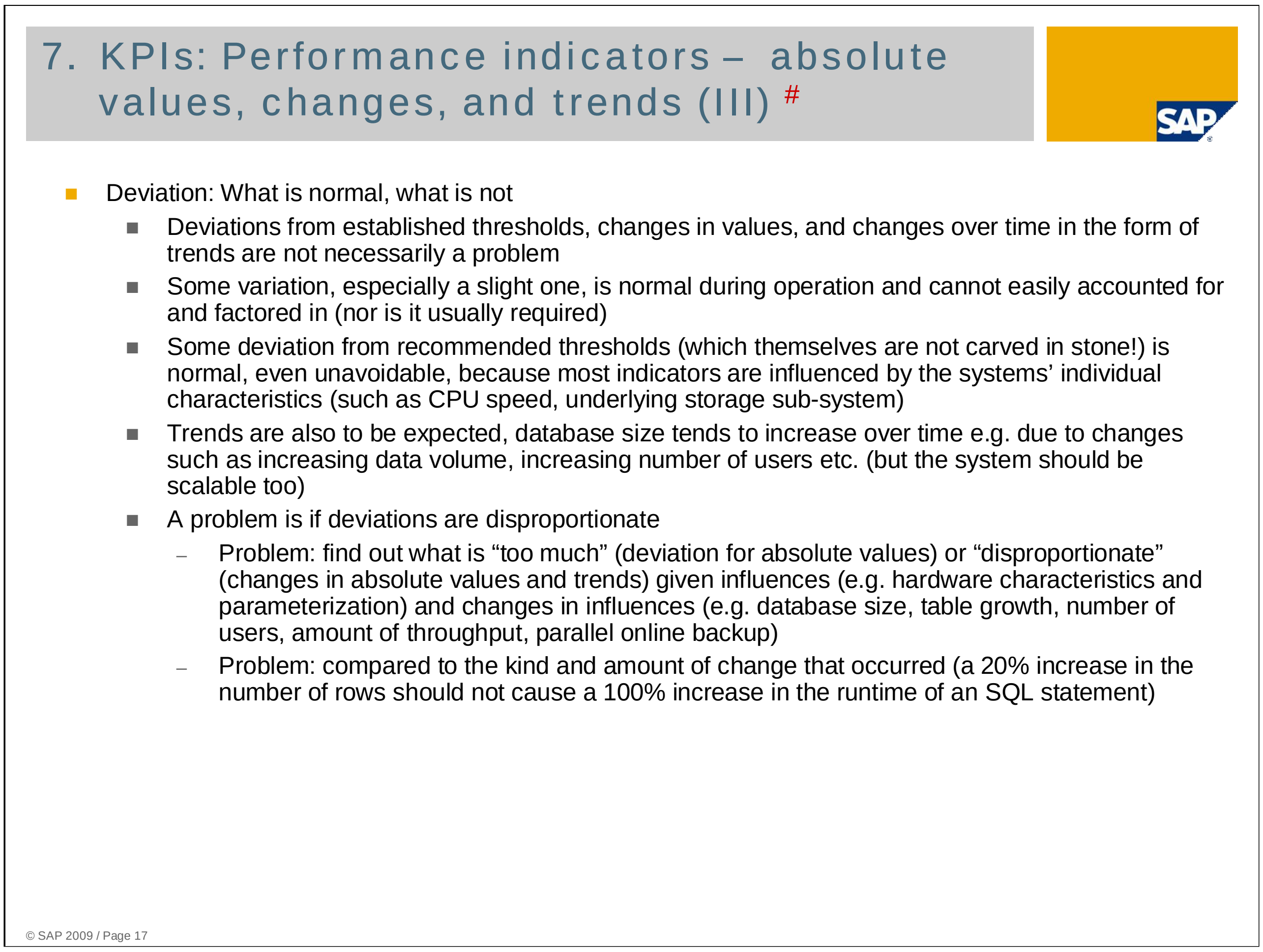
5: Initially, to compare system values with thresholds is a good idea, first because there is often little other to compare with (unless there is already a similar, comparable installation of which the values can be used) but also to see how far away from established values a system is, second because a deviation, especially if it is important, has a good chance of having for a cause a significant problem such as system setup, parameterization, possibly even sizing. However, once a system has been up and running under representative conditions (“business as usual” mode) comparisons against the system’s own, historical data becomes more and more important. Differences and changes against its own, earlier data, new trends or changes in existing trends become much more important signals to be heeded than comparisons with standard thresholds



7. KPIs: Performance indicators – absolute values, changes, and trends (II)

- Change and constancy
  - Some indicators do not change at all (CPU speed) while others change dramatically during the course of a day, month, etc. (e.g. system workload at 9:30 am vs. lunch break), some can change as a function of others (disk service and wait times as a function of the workload)
  - One challenge is find out which factors influence which parameters and whether observed changes are reasonable
  - Another challenge is find comparable situations to verify whether performance under these changed conditions is better, comparable, or worse than it should be:
    - To compare 9:30 am with lunch break will probably not work;
    - Neither to compare Wednesday with Friday (if week end closing runs on Fridays);
    - Nor Friday 10th with Friday 31st (month end closing)
    - An online backup that exceptionally runs 2 hours later than usual can invalidate a whole number of “findings”

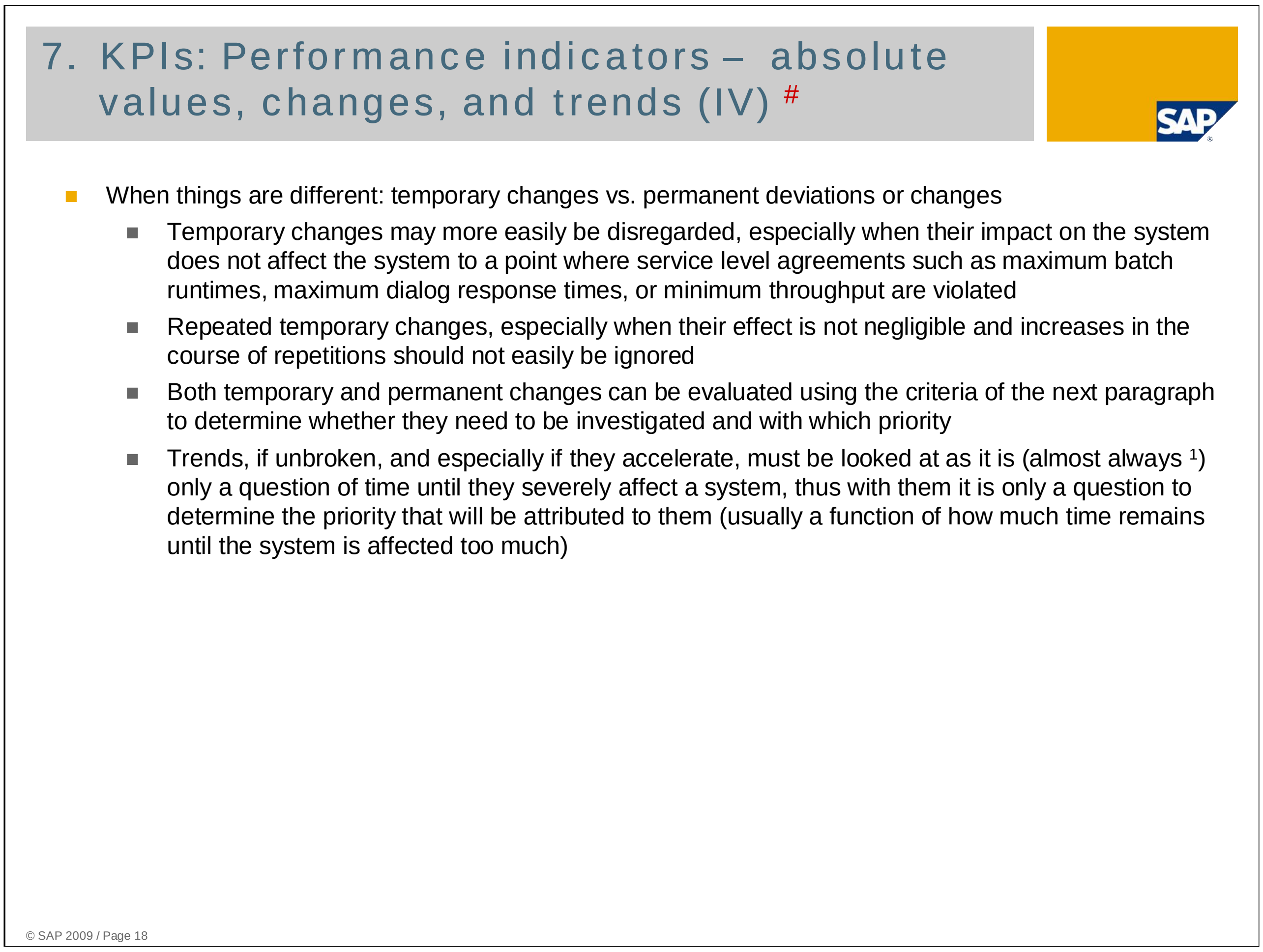
#: Slides 15 to 18 were not shown in the presentation but only referred to and are here included as promised to provide further information on KPIs



7. KPIs: Performance indicators – absolute values, changes, and trends (III)

- Deviation: What is normal, what is not
  - Deviations from established thresholds, changes in values, and changes over time in the form of trends are not necessarily a problem
  - Some variation, especially a slight one, is normal during operation and cannot easily accounted for and factored in (nor is it usually required)
  - Some deviation from recommended thresholds (which themselves are not carved in stone!) is normal, even unavoidable, because most indicators are influenced by the systems' individual characteristics (such as CPU speed, underlying storage sub-system)
  - Trends are also to be expected, database size tends to increase over time e.g. due to changes such as increasing data volume, increasing number of users etc. (but the system should be scalable too)
  - A problem is if deviations are disproportionate
    - Problem: find out what is "too much" (deviation for absolute values) or "disproportionate" (changes in absolute values and trends) given influences (e.g. hardware characteristics and parameterization) and changes in influences (e.g. database size, table growth, number of users, amount of throughput, parallel online backup)
    - Problem: compared to the kind and amount of change that occurred (a 20% increase in the number of rows should not cause a 100% increase in the runtime of an SQL statement)

#: Slides 15 to 18 were not shown in the presentation but only referred to and are here included as promised to provide further information on KPIs

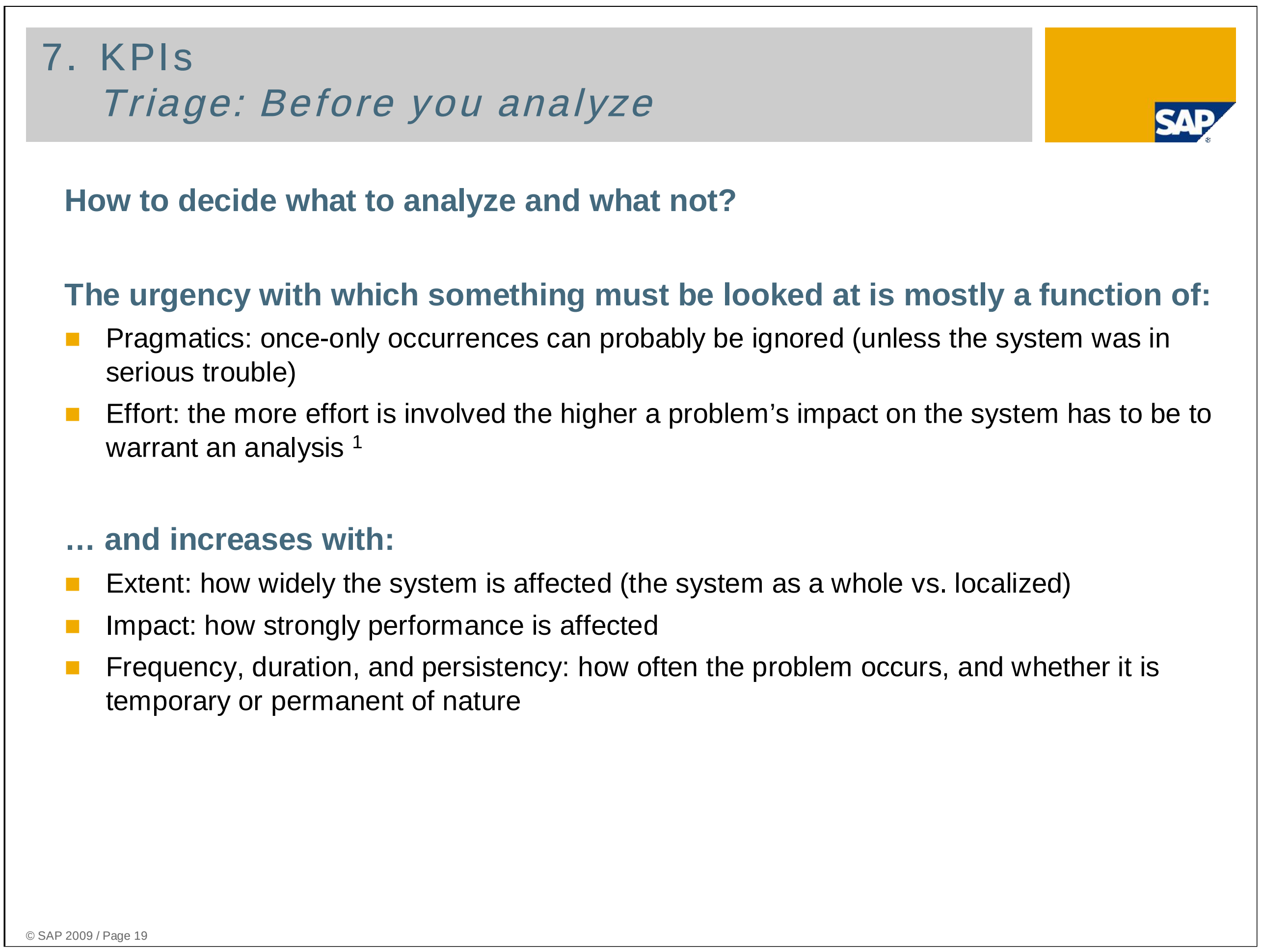


7. KPIs: Performance indicators – absolute values, changes, and trends (IV)

- When things are different: temporary changes vs. permanent deviations or changes
  - Temporary changes may more easily be disregarded, especially when their impact on the system does not affect the system to a point where service level agreements such as maximum batch runtimes, maximum dialog response times, or minimum throughput are violated
  - Repeated temporary changes, especially when their effect is not negligible and increases in the course of repetitions should not easily be ignored
  - Both temporary and permanent changes can be evaluated using the criteria of the next paragraph to determine whether they need to be investigated and with which priority
  - Trends, if unbroken, and especially if they accelerate, must be looked at as it is (almost always [1]) only a question of time until they severely affect a system, thus with them it is only a question to determine the priority that will be attributed to them (usually a function of how much time remains until the system is affected too much)

#: Slides 15 to 18 were not shown in the presentation but only referred to and are here included as promised to provide further information on KPIs

1: An example for an exception is a scenario that scales logarithmically such as table growth where the table is accessed through a fully selective index



7. KPIs
Triage: Before you analyze

How to decide what to analyze and what not?

The urgency with which something must be looked at is mostly a function of:

- Pragmatics: once-only occurrences can probably be ignored (unless the system was in serious trouble)
- Effort: the more effort is involved the higher a problem's impact on the system has to be to warrant an analysis [1]

… and increases with:

- Extent: how widely the system is affected (the system as a whole vs. localized)
- Impact: how strongly performance is affected
- Frequency, duration, and persistency: how often the problem occurs, and whether it is temporary or permanent of nature

1: This, unfortunately, often only becomes clear in the course of an analysis

7. KPIs: Details
Overview

This section contains:

- **The most important areas of performance problems:**
  - High response times
  - Resource bottlenecks (CPU, RAM, I/O)
  - Buffer pool problems
  - Locking problems
  - Sorting problems
  - Expensive SQL statements [1]

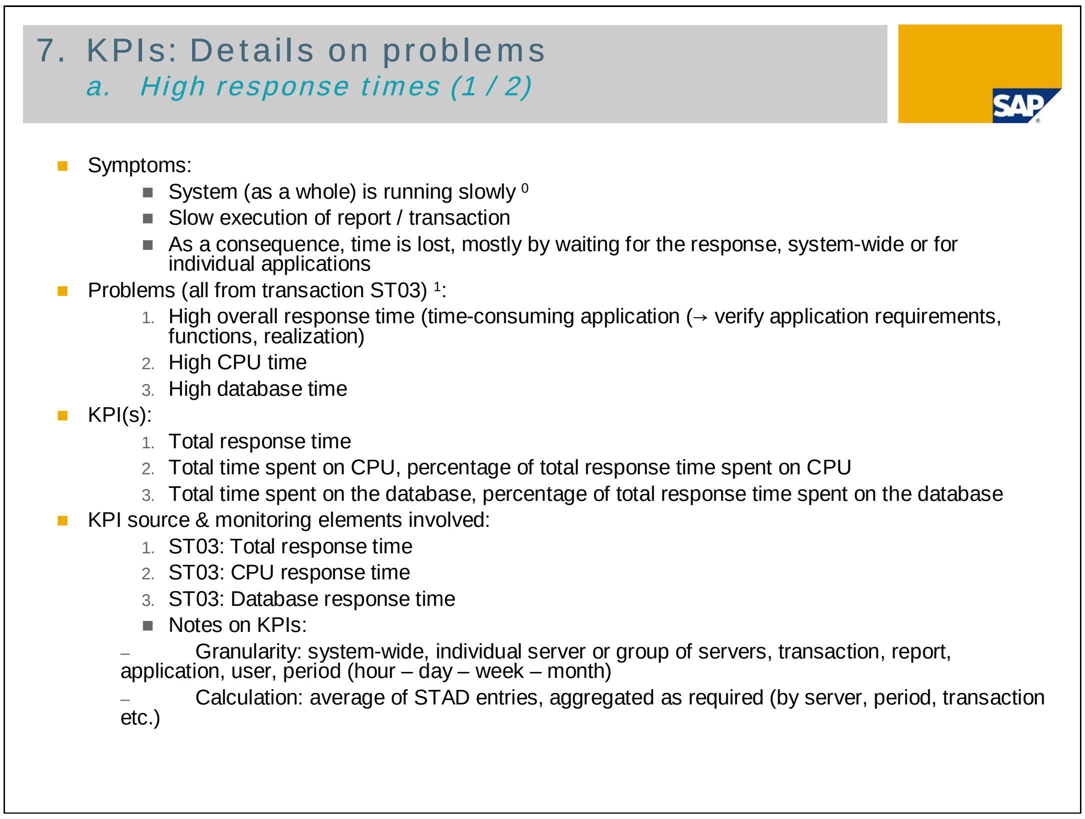
- **The details provided on the individual problems:**
  - Symptoms – Problems – KPIs – KPI source & monitoring elements involved – Thresholds – Analysis – Remedies and comments

1: Expensive SQL statements will be addressed in 2 separate remote expert session to be held in Q1/2010

7. KPIs: Details
Details provided on problems

For each of the most important problems that can affect performance the following information will be provided:

- **Symptoms:** For each family of problems, what are its most important symptoms?
- **Problems:** What, for each family, are the problems that occur when things go cactus?
- **KPIs:** Which KPI(s) indicate the problem?
- **KPI source & monitoring elements involved:** Where can the KPIs be found and what database monitoring elements are involved in a KPI?
- **Thresholds:** What threshold values have proven to be the most useful?
- **Analysis:** What, if any, further steps to analyze the problem need to be taken?
  - Ways to confirm or refute the current hypothesis as to what the cause(s) might be
  - Ways to dig deeper into the problem
- **Remedies and comments:**
  - What corrective measures exist to solve the performance problem?
  - Further things worthy to be mentioned



7. KPIs: Details on problems

a. High response times (1 / 2)

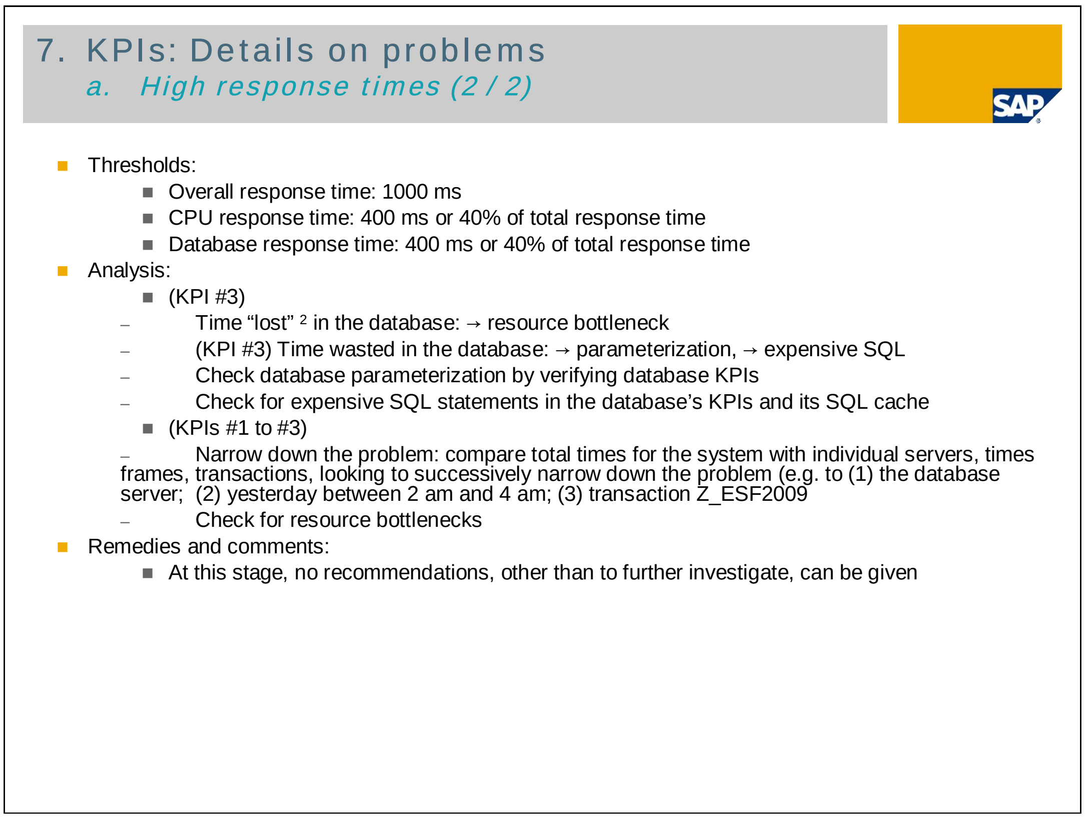
- Symptoms:
  - System (as a whole) is running slowly [0]
  - Slow execution of report / transaction
  - As a consequence, time is lost, mostly by waiting for the response, system-wide or for individual applications
- Problems (all from transaction ST03) [1]:
  1. High overall response time (time-consuming application (→ verify application requirements, functions, realization)
  2. High CPU time
  3. High database time
- KPI(s):
  1. Total response time
  2. Total time spent on CPU, percentage of total response time spent on CPU
  3. Total time spent on the database, percentage of total response time spent on the database
- KPI source & monitoring elements involved:
  1. ST03: Total response time
  2. ST03: CPU response time
  3. ST03: Database response time
  - Notes on KPIs:

– Granularity: system-wide, individual server or group of servers, transaction, report, application, user, period (hour – day – week – month)

– Calculation: average of STAD entries, aggregated as required (by server, period, transaction etc.)

0: or least seems to, most likely to the users

1: Of course there is the possibility of time being spent somewhere else; other fields available in ST03 as well as performance-related information to be found elsewhere are excluded here, as we focus on the database performance and directly related areas; for more information ABAP performance, the performance area most closely related and interwoven with database performance see → ABAP Performance Tuning, by Hermann Gahm, for information on other areas of SAP performance see → SAP Performance Optimization, by Thomas Schneider; for ABAP performance



7. KPIs: Details on problems

a. High response times (2 / 2)

- Thresholds:
  - Overall response time: 1000 ms
  - CPU response time: 400 ms or 40% of total response time
  - Database response time: 400 ms or 40% of total response time
- Analysis:
  - (KPI #3)
    - Time “lost” [2] in the database: → resource bottleneck
    - (KPI #3) Time wasted in the database: → parameterization, → expensive SQL
    - Check database parameterization by verifying database KPIs
    - Check for expensive SQL statements in the database’s KPIs and its SQL cache
  - (KPIs #1 to #3)
    - Narrow down the problem: compare total times for the system with individual servers, times frames, transactions, looking to successively narrow down the problem (e.g. to (1) the database server; (2) yesterday between 2 am and 4 am; (3) transaction Z_ESF2009
    - Check for resource bottlenecks
- Remedies and comments:
  - At this stage, no recommendations, other than to further investigate, can be given

2: “lost” means here “time spent” but nothing to show for, caused by:

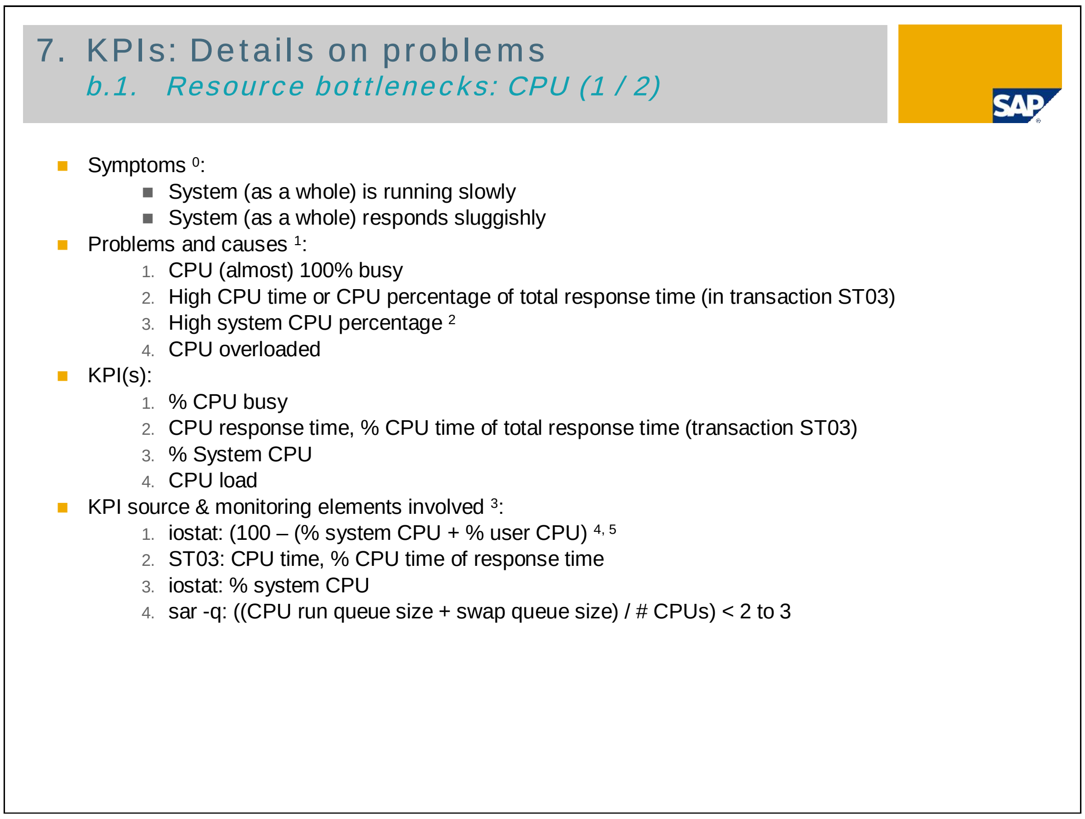
(a) waiting for an available CPU (→ CPU bottleneck);

(b) waiting for data to be swapped in (RAM bottleneck);

(c) waiting for data to be delivered from the storage sub-system (I/O bottleneck)

(d) additionally, sort, lock, and problems with expensive SQL statements can contribute

→ basically everything…



7. KPIs: Details on problems

b.1. Resource bottlenecks: CPU (1 / 2)

- Symptoms ^{0}:
  - System (as a whole) is running slowly
  - System (as a whole) responds sluggishly
- Problems and causes ^{1}:
  1. CPU (almost) 100% busy
  2. High CPU time or CPU percentage of total response time (in transaction ST03)
  3. High system CPU percentage ^{2}
  4. CPU overloaded
- KPI(s):
  1. % CPU busy
  2. CPU response time, % CPU time of total response time (transaction ST03)
  3. % System CPU
  4. CPU load
- KPI source & monitoring elements involved ^{3}:
  1. iostat: (100 – (% system CPU + % user CPU) $^{4, 5}$
  2. ST03: CPU time, % CPU time of response time
  3. iostat: % system CPU
  4. sar -q: ((CPU run queue size + swap queue size) / # CPUs) < 2 to 3

0: This can happen permanently or intermittently

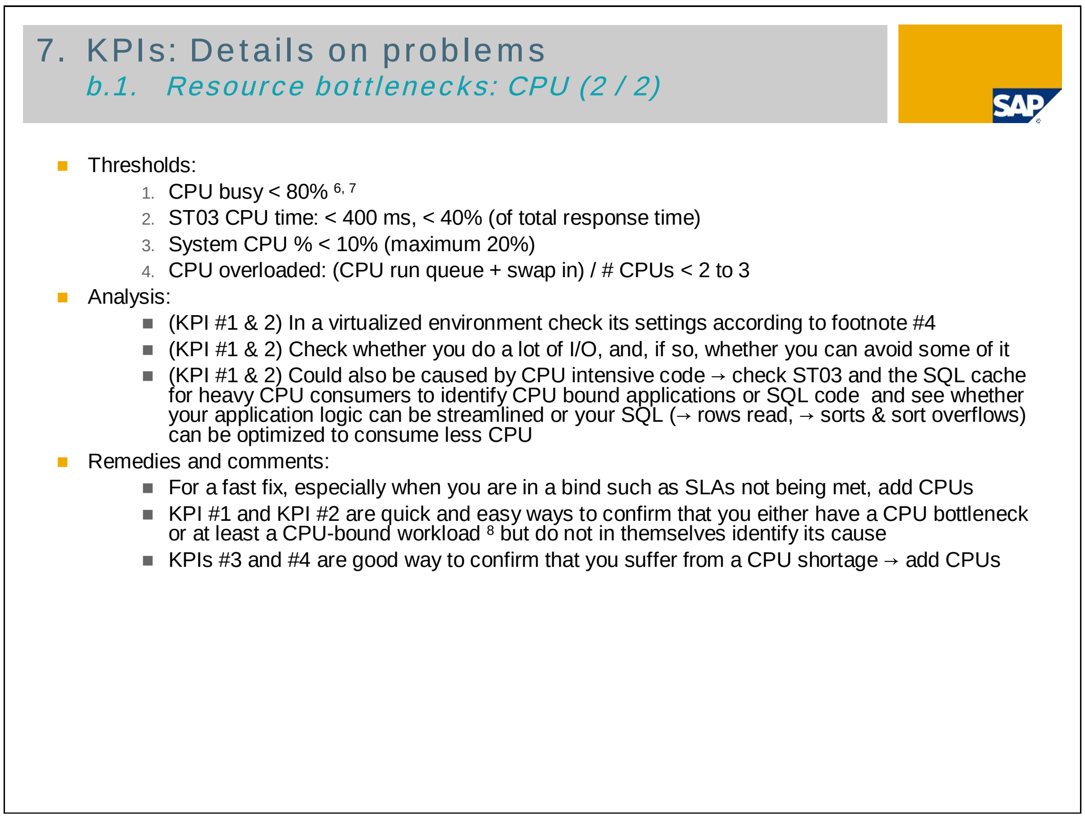
1: Here, as an effort to keep the number of possible answers manageable, we’ll concentrate on causes that are related to CPU bottlenecks only

2: CPU 100% busy (or close to it) causes more context switching, which is part of the system CPU; user CPU is when the CPU spends time working on an application

3: All bottleneck KPIs can be found on AIX and Linux in the NMON monitor

4: An additional element to be considered in case CPU is 100% busy (or close to it) is if the 100% busy situation occurs in a virtualized setting. Then the CPU consumption compared to entitlement & whether CPU is capped or uncapped may have a bearing. 100% CPU busy in a capped LPAR means the system has reached its CPU entitlement and is not allowed to exceed it even if there are CPU cycles available on the box → check the CPU utilization of the box as a whole, if there are cycles left consider uncapping the LPAR or increasing the LPAR’s entitlement (you may also need to increase the number of virtual CPUs); 100% CPU busy in an uncapped LPAR means that even though the LPAR could request more CPU than what it is entitled to, there are no CPU cycles available on the box → the box as a whole has a CPU shortage

5: vmstat also is a possibility (and a score of others…)



7. KPIs: Details on problems

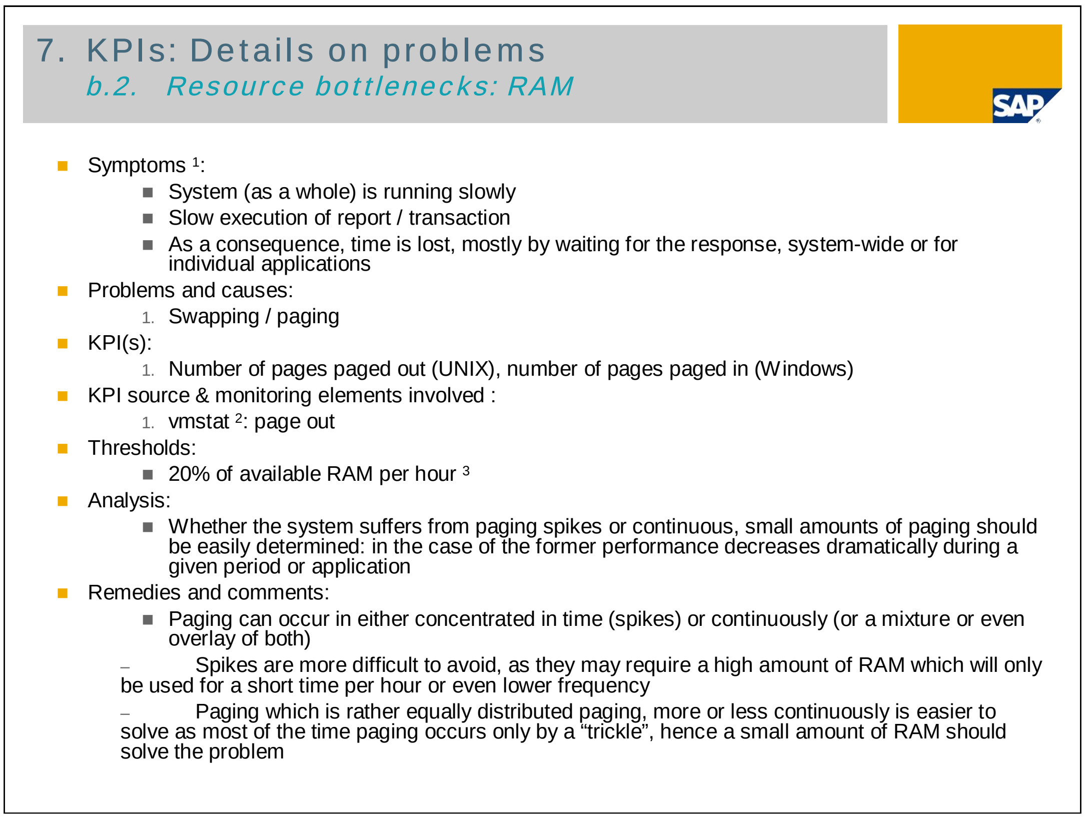
b.1. Resource bottlenecks: CPU (2 / 2)

- Thresholds:
  1. CPU busy < 80% [6, 7]
  2. ST03 CPU time: < 400 ms, < 40% (of total response time)
  3. System CPU % < 10% (maximum 20%)
  4. CPU overloaded: (CPU run queue + swap in) / # CPUs < 2 to 3
- Analysis:
  - (KPI #1 & 2) In a virtualized environment check its settings according to footnote #4
  - (KPI #1 & 2) Check whether you do a lot of I/O, and, if so, whether you can avoid some of it
  - (KPI #1 & 2) Could also be caused by CPU intensive code → check ST03 and the SQL cache for heavy CPU consumers to identify CPU bound applications or SQL code and see whether your application logic can be streamlined or your SQL (→ rows read, → sorts & sort overflows) can be optimized to consume less CPU
- Remedies and comments:
  - For a fast fix, especially when you are in a bind such as SLAs not being met, add CPUs
  - KPI #1 and KPI #2 are quick and easy ways to confirm that you either have a CPU bottleneck or at least a CPU-bound workload [8] but do not in themselves identify its cause
  - KPIs #3 and #4 are good way to confirm that you suffer from a CPU shortage → add CPUs

6: I/O wait time is considered “idle” time as the CPU during I/O wait time has nothing else to do but wait for I/O request

7: 80% are a (near) real-time value; the traditional, established average is recommended to be as low as 20%; however, caution should be exercised with this value as it only provides relative safety against peaks where CPU utilization hits 100% → to verify system CPU percentage or run queue + swap in will be of no help as they, too, will equally have any peaks averaged out over time

8: Which could also explain why you have a CPU bottleneck, that is when sizing has been carried out for an average workload, not taking into account that some slices of the workload are particularly CPU-bound



7. KPIs: Details on problems

b.2. Resource bottlenecks: RAM

- Symptoms [1]:
  - System (as a whole) is running slowly
  - Slow execution of report / transaction
  - As a consequence, time is lost, mostly by waiting for the response, system-wide or for individual applications
- Problems and causes:
  1. Swapping / paging
- KPI(s):
  1. Number of pages paged out (UNIX), number of pages paged in (Windows)
- KPI source & monitoring elements involved :
  1. vmstat [2]: page out
- Thresholds:
  - 20% of available RAM per hour [3]
- Analysis:
  - Whether the system suffers from paging spikes or continuous, small amounts of paging should be easily determined: in the case of the former performance decreases dramatically during a given period or application
- Remedies and comments:
  - Paging can occur in either concentrated in time (spikes) or continuously (or a mixture or even overlay of both)
    - Spikes are more difficult to avoid, as they may require a high amount of RAM which will only be used for a short time per hour or even lower frequency
    - Paging which is rather equally distributed paging, more or less continuously is easier to solve as most of the time paging occurs only by a “trickle”, hence a small amount of RAM should solve the problem

1: This can happen permanently or intermittently

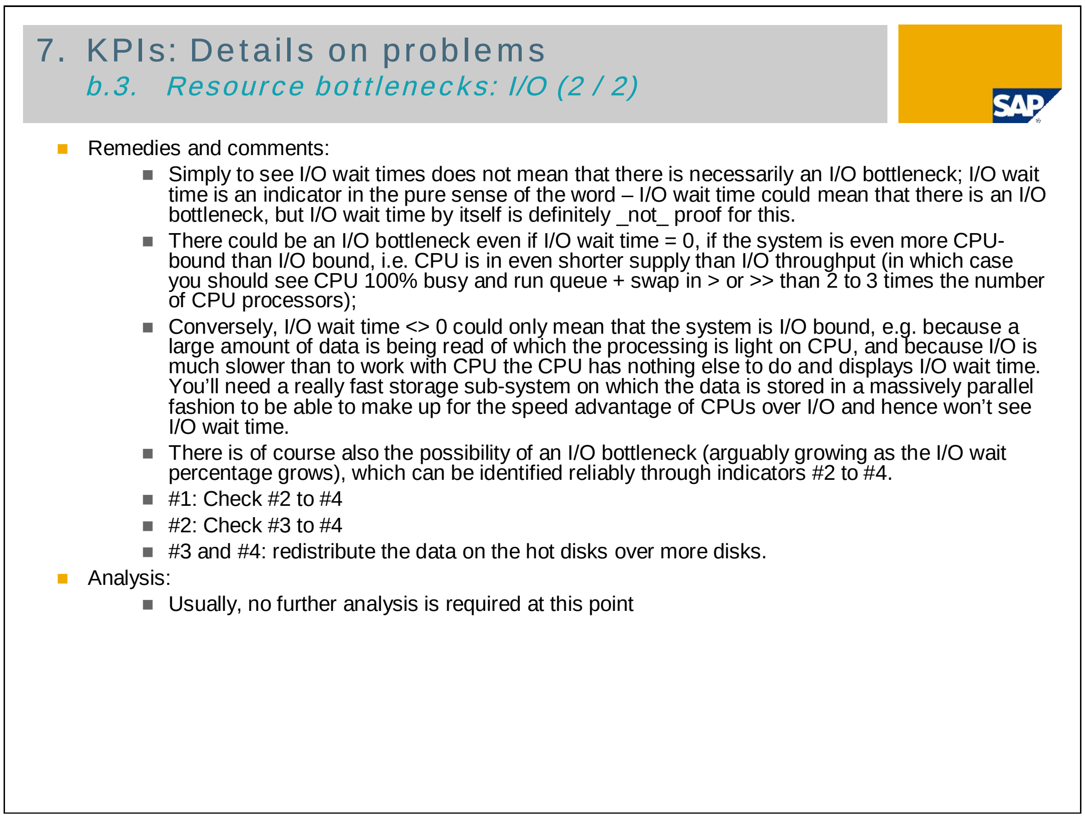
2: and others of course…

3: This value has been established historically, but may not be valid anymore, especially if you have a system for which a constant, high performance is required. Depending on the storage sub-system available, swapping comes with a performance penalty of between 10 and 1000, usually in the low hundreds, compared to working in memory.

7. KPIs: Details on problems

b.3. Resource bottlenecks: I/O (1 / 2)

- Symptoms:
  - System-wide performance problems
  - Performance problem during certain times or for certain applications
- Problems and causes:
  1. I/O bottleneck: disks / host devices are 100% busy
- KPI(s):
  1. I/O wait time
  2. Disk busy
  3. Disk service times
  4. Disk wait times
- KPI source & monitoring elements involved:
  1. iostat: iowait
  2. nmon -d: Busy
  3. nmon -d: Service
  4. nmon -d: Wait
- Thresholds:
  1. iowait = 0, or > 0: it depends [1]
  2. Disk busy: ~ 100% (in itself not conclusive either – like I/O wait time), #3 and #4 need to grow too
  3. Service time: Growing as the device approaches 100%
  4. Wait time: Growing (typically from 0 to non-0 values) as the device approaches 100%



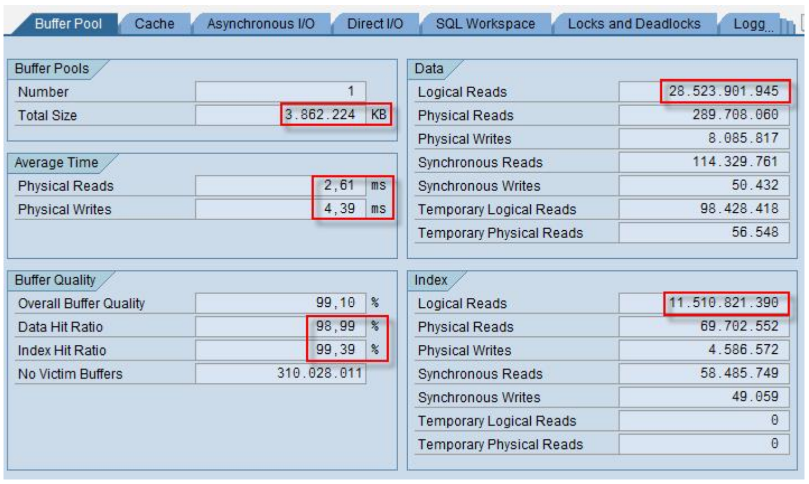
7. KPIs: Details on problems

b.3. Resource bottlenecks: I/O (2 / 2)

- Remedies and comments:
  - Simply to see I/O wait times does not mean that there is necessarily an I/O bottleneck; I/O wait time is an indicator in the pure sense of the word – I/O wait time could mean that there is an I/O bottleneck, but I/O wait time by itself is definitely _not_ proof for this.
  - There could be an I/O bottleneck even if I/O wait time = 0, if the system is even more CPU-bound than I/O bound, i.e. CPU is in even shorter supply than I/O throughput (in which case you should see CPU 100% busy and run queue + swap in > or >> than 2 to 3 times the number of CPU processors);
  - Conversely, I/O wait time <> 0 could only mean that the system is I/O bound, e.g. because a large amount of data is being read of which the processing is light on CPU, and because I/O is much slower than to work with CPU the CPU has nothing else to do and displays I/O wait time. You'll need a really fast storage sub-system on which the data is stored in a massively parallel fashion to be able to make up for the speed advantage of CPUs over I/O and hence won't see I/O wait time.
  - There is of course also the possibility of an I/O bottleneck (arguably growing as the I/O wait percentage grows), which can be identified reliably through indicators #2 to #4.
  - #1: Check #2 to #4
  - #2: Check #3 to #4
  - #3 and #4: redistribute the data on the hot disks over more disks.
- Analysis:
  - Usually, no further analysis is required at this point

7. KPIs: Details on problems

c. *Buffer pool problems (screenshot)*



Buffer Pools

Number	1	
Total Size	3.862.224	KB

Average Time

Physical Reads	2,61	ms
Physical Writes	4,39	ms

Buffer Quality

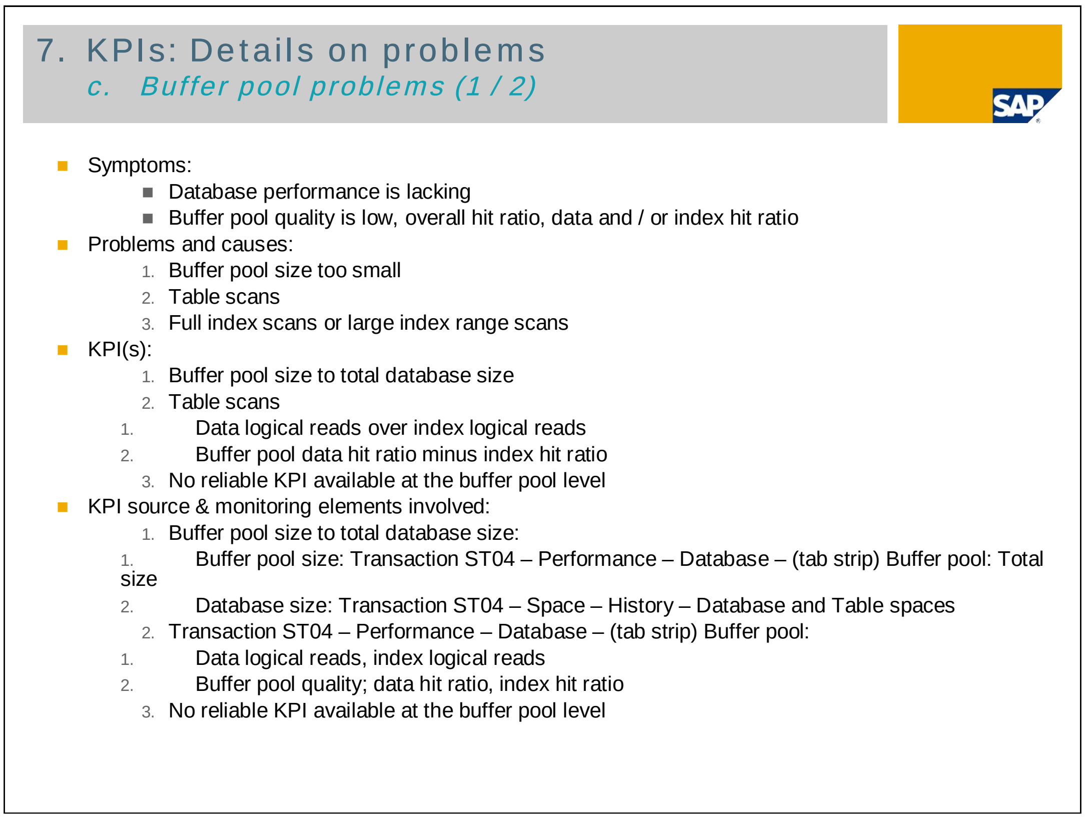
Overall Buffer Quality	99,10	%
Data Hit Ratio	98,99	%
Index Hit Ratio	99,39	%
No Victim Buffers	310.028.011	

Data

Logical Reads	28.523.901.945
Physical Reads	289.708.060
Physical Writes	8.085.817
Synchronous Reads	114.329.761
Synchronous Writes	50.432
Temporary Logical Reads	98.428.418
Temporary Physical Reads	56.548

Index

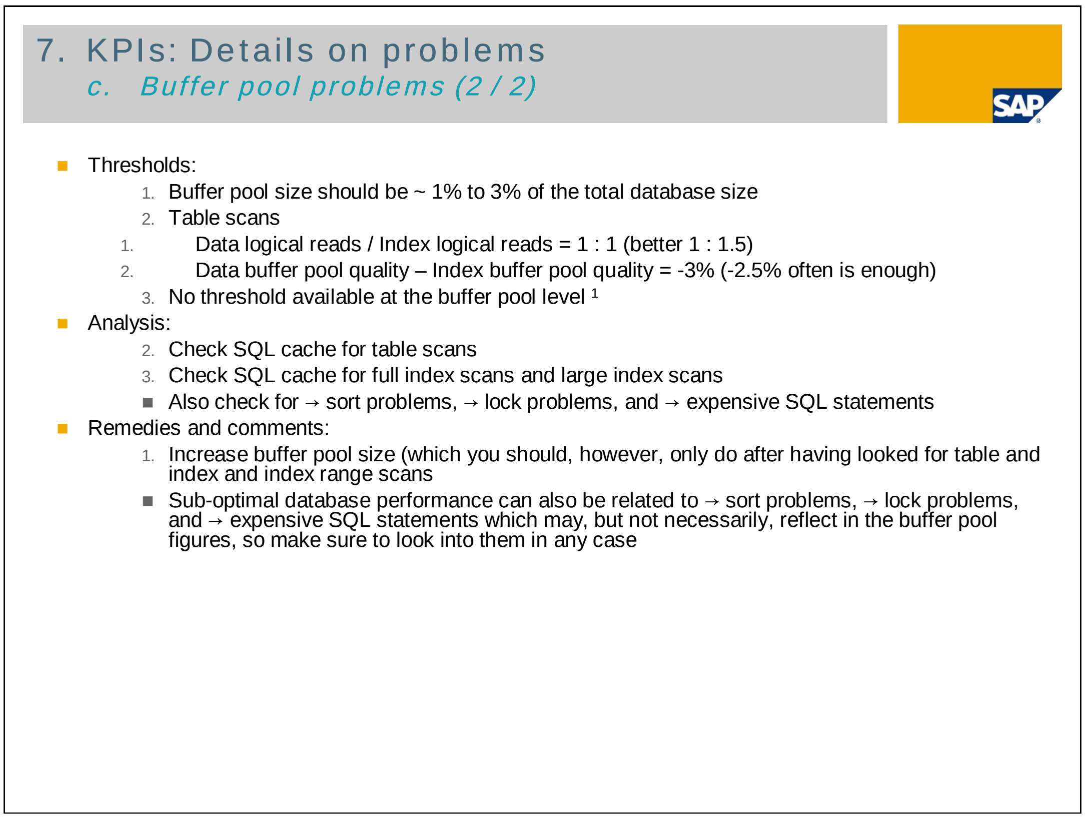
Logical Reads	11.510.821.390
Physical Reads	69.702.552
Physical Writes	4.586.572
Synchronous Reads	58.485.749
Synchronous Writes	49.059
Temporary Logical Reads	0
Temporary Physical Reads	0



7. KPIs: Details on problems

c. Buffer pool problems (1 / 2)

- Symptoms:
  - Database performance is lacking
  - Buffer pool quality is low, overall hit ratio, data and / or index hit ratio
- Problems and causes:
  1. Buffer pool size too small
  2. Table scans
  3. Full index scans or large index range scans
- KPI(s):
  1. Buffer pool size to total database size
  2. Table scans
     1. Data logical reads over index logical reads
     2. Buffer pool data hit ratio minus index hit ratio
  3. No reliable KPI available at the buffer pool level
- KPI source & monitoring elements involved:
  1. Buffer pool size to total database size:
     1. Buffer pool size: Transaction ST04 – Performance – Database – (tab strip) Buffer pool: Total size
     2. Database size: Transaction ST04 – Space – History – Database and Table spaces
  2. Transaction ST04 – Performance – Database – (tab strip) Buffer pool:
     1. Data logical reads, index logical reads
     2. Buffer pool quality; data hit ratio, index hit ratio
  3. No reliable KPI available at the buffer pool level



7. KPIs: Details on problems

c. Buffer pool problems (2 / 2)

- Thresholds:
  1. Buffer pool size should be ~ 1% to 3% of the total database size
  2. Table scans
     1. Data logical reads / Index logical reads = 1 : 1 (better 1 : 1.5)
     2. Data buffer pool quality – Index buffer pool quality = -3% (-2.5% often is enough)
  3. No threshold available at the buffer pool level [1]
- Analysis:
  2. Check SQL cache for table scans
  3. Check SQL cache for full index scans and large index scans
  - Also check for → sort problems, → lock problems, and → expensive SQL statements
- Remedies and comments:
  1. Increase buffer pool size (which you should, however, only do after having looked for table and index and index range scans
  - Sub-optimal database performance can also be related to → sort problems, → lock problems, and → expensive SQL statements which may, but not necessarily, reflect in the buffer pool figures, so make sure to look into them in any case

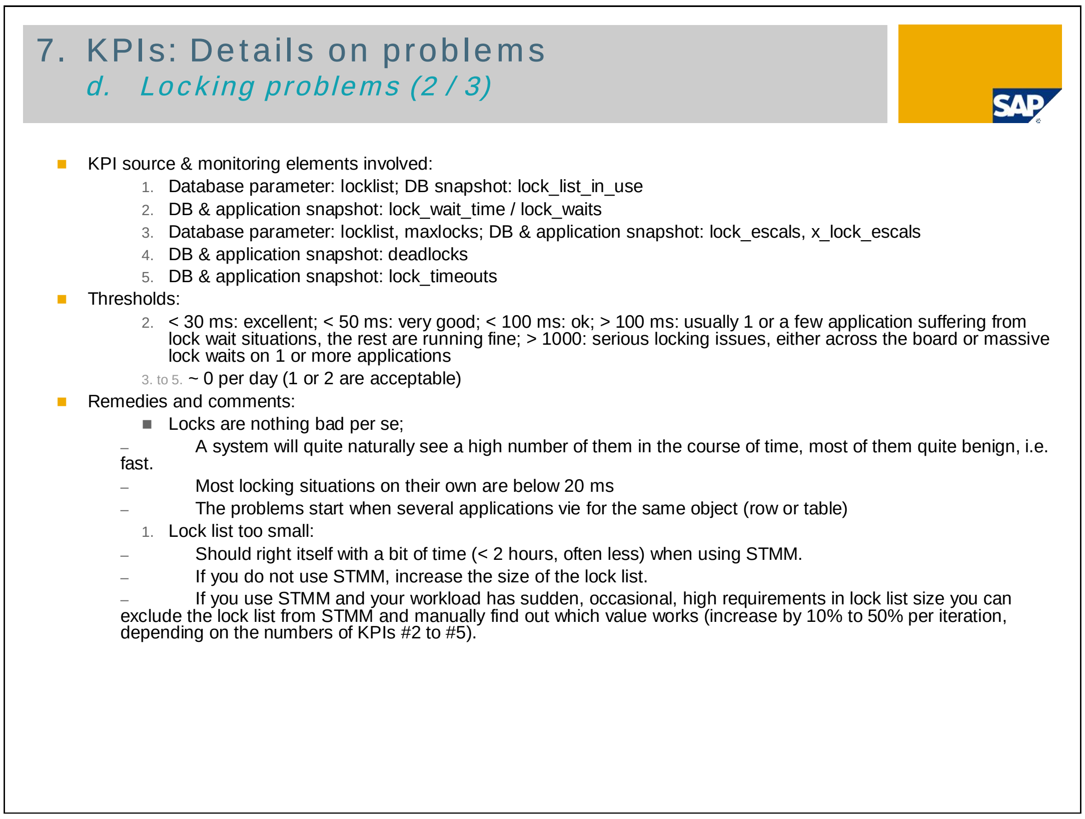
1: While theoretically the KPI for full index scans and expensive index scans is the inverse of table scans this (almost) never occurs in real life. (Ok, in the unlikely event of your buffer pool data hit ratio being 2.5% or more better than the index hit ratio or the ratio of data logical reads over index logical reads being 0.75 or less you know where to look for. Let me know when this happens, I would love to have a look at that system.

In real life you'll find full index scans or large index range scans by looking at the number of index rows read, per execution and in absolute numbers, in the buffer pool.

7. KPIs: Details on problems

d. Locking problems (1 / 3)

- Symptoms:
  - Performance of one or several applications leave to be desired
  - Performance rapidly degrades with increasing workload
  - Applications take an extremely long time to complete
  - Applications are terminated, a short dump occurs
- Problems and causes:
  1. Lock list is too small, causing lock and exclusive lock escalations, which in turn can cause deadlocks
  2. High time waited for a lock to be released so that the current application may acquire it: Data is locked by a badly performing application (holding locks for too long a time, i.e. not releasing them as soon as possible or a sub-optimal application logic, requiring to hold locks for an extended time) resulting in extended lock wait times; this is often exacerbated in the case of a highly parallelized, badly performing application where then a large number of work processes running in parallel
  3. (Exclusive) Lock escalations occur: an application uses either the maximum of lock space allowed for a single application or the lock list runs full
  4. Deadlock: bad application logic (usually the case) or simply bad luck; a deadlock is detected by the deadlock monitor, its default is to check for deadlock every 300 seconds i.e. 5 minutes
  5. Lock timeout: if an application holds locks that are held longer than a maximum of time (default is 3600 seconds i.e. 1 hour) the application is terminated and the lock released
- KPI(s) (all transaction ST04 – Performance – Database – (tab strip) Locks and Deadlocks:
  1. Lock list
  2. Lock waits - Average time waited
  3. Lock escalations & Exclusive lock escalations
  4. Locks – Deadlocks detected
  5. Lock – Lock timeouts



7. KPIs: Details on problems

d. Locking problems (2 / 3)

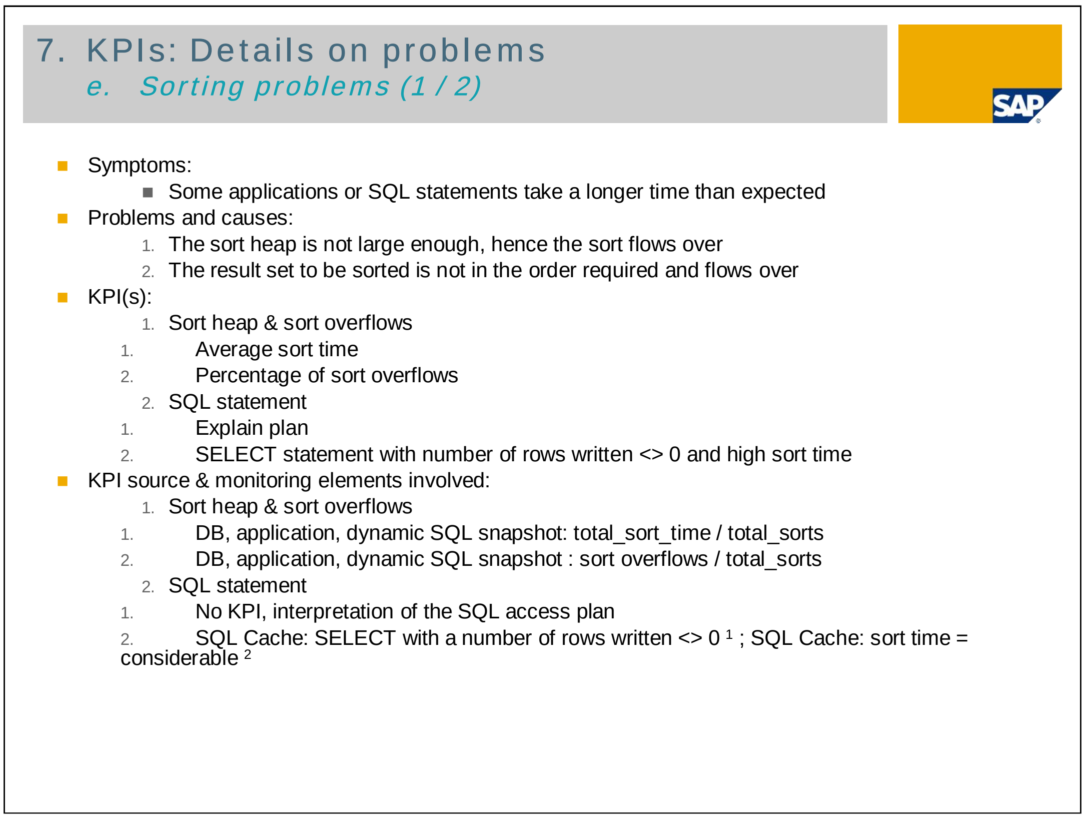
- KPI source & monitoring elements involved:
  1. Database parameter: locklist; DB snapshot: lock_list_in_use
  2. DB & application snapshot: lock_wait_time / lock_waits
  3. Database parameter: locklist, maxlocks; DB & application snapshot: lock_escals, x_lock_escals
  4. DB & application snapshot: deadlocks
  5. DB & application snapshot: lock_timeouts
- Thresholds:
  2. < 30 ms: excellent; < 50 ms: very good; < 100 ms: ok; > 100 ms: usually 1 or a few application suffering from lock wait situations, the rest are running fine; > 1000: serious locking issues, either across the board or massive lock waits on 1 or more applications

  3. to 5. ~ 0 per day (1 or 2 are acceptable)
- Remedies and comments:
  - Locks are nothing bad per se;
    - A system will quite naturally see a high number of them in the course of time, most of them quite benign, i.e. fast.
    - Most locking situations on their own are below 20 ms
    - The problems start when several applications vie for the same object (row or table)
  1. Lock list too small:
     - Should right itself with a bit of time (< 2 hours, often less) when using STMM.
     - If you do not use STMM, increase the size of the lock list.
     - If you use STMM and your workload has sudden, occasional, high requirements in lock list size you can exclude the lock list from STMM and manually find out which value works (increase by 10% to 50% per iteration, depending on the numbers of KPIs #2 to #5).

7. KPIs: Details on problems

d. Locking problems (3 / 3)

- Analysis:
  1. (lock list too small) There is no high water mark for the lock list (contrary to the sort heap); take regular snapshots of the lock list size to see whether the lock list is often close to its limit and match the numbers with the times of #2 to #5. Note that this is a moot point if you use STMM (and if you have the occasional spike in space required in the lock list STMM may be too slow to adapt), see also → remedies
  2. (average time waited for lock)
     - Carry out a lock wait analysis using db6util -ls (lock situation)
     - Collect regular snapshots on applications looking for their number of lock waits and time waited on locks
     - Once the application is identified take an ST12 trace (combined ABAP – SQL trace) looking for expensive ABAP statements (which may cause the lock wait situation) and long running SQL statements, especially that run a surprisingly long time for a good access plan (which may be the statements suffering from lock wait situations; in particular, if the application at different times (and with an already “warmed up” buffer pool) has significantly different runtimes the APAB statement and SQL statements that show largely differing times are likely to be involved in lock wait situations
  3. ((exclusive) lock escalation)
     - Take regular lock snapshots and application snapshots to see when and where lock escalation occur
     - Collect lock_help (administrative view) information and filter for lock_object_type = table lock, lock_mode = IX (intent exclusive), SIX (share with intention exclusive), X (exclusive lock), or Z (super exclusive lock), lock_escalation (small integer)
  4. (deadlock)
     - Carry out a deadlock analysis using db6util -ld (lock – deadlock)
     - Collect regular snapshots on applications looking for their number of deadlocks
     - Run the application, monitoring it in SM50 or SM66; once it is stuck for more than a few seconds go look up its report and main action (highlight the work process and select “Choose” or “Details”) and look into the corresponding application using its PID in ST04 – Performance – Application for further details
  5. (timeout)
     - Collect regular snapshot information on applications looking for their number of lock timeouts to identify which applications suffer
     - Collect regular snapshot_lockwait information (table function), filtering for the application identified above
     - Carry out a lock wait analysis using db6util -ls (lock situation) to catch the problem when it occurs



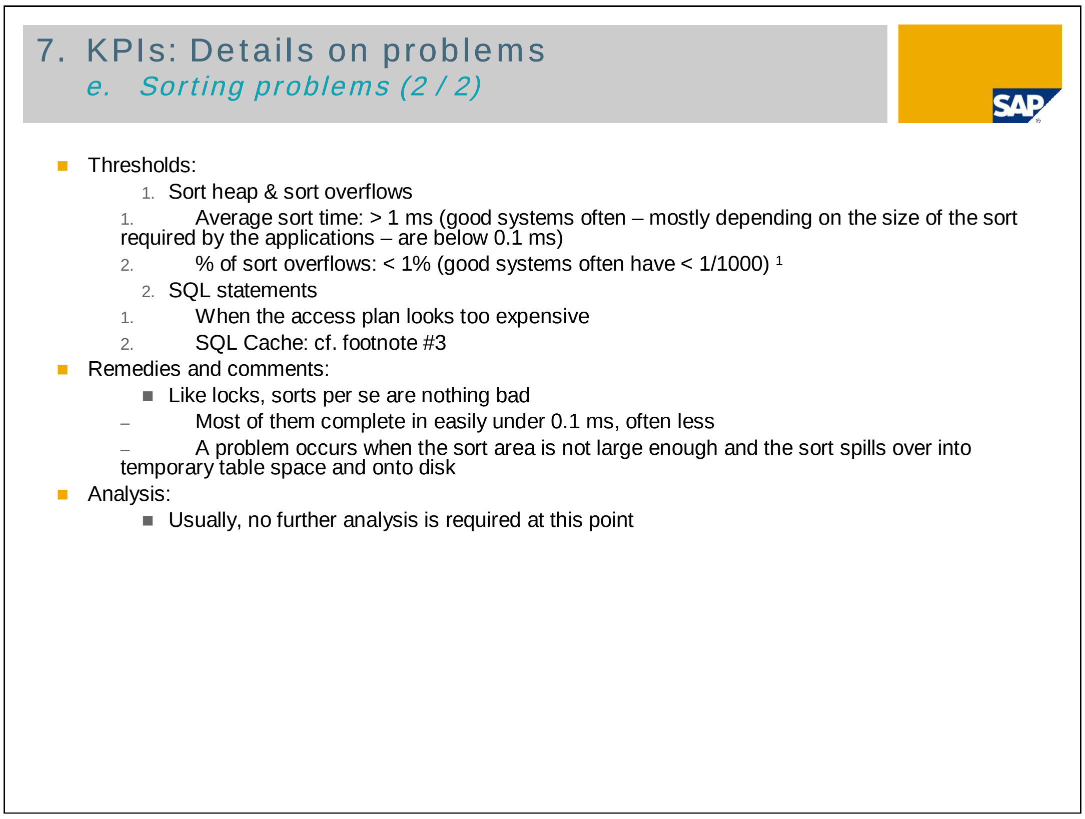
7. KPIs: Details on problems

e. Sorting problems (1 / 2)

- Symptoms:
  - Some applications or SQL statements take a longer time than expected
- Problems and causes:
  1. The sort heap is not large enough, hence the sort flows over
  2. The result set to be sorted is not in the order required and flows over
- KPI(s):
  1. Sort heap & sort overflows
     1. Average sort time
     2. Percentage of sort overflows
  2. SQL statement
     1. Explain plan
     2. SELECT statement with number of rows written <> 0 and high sort time
- KPI source & monitoring elements involved:
  1. Sort heap & sort overflows
     1. DB, application, dynamic SQL snapshot: total_sort_time / total_sorts
     2. DB, application, dynamic SQL snapshot : sort overflows / total_sorts
  2. SQL statement
     1. No KPI, interpretation of the SQL access plan
     2. SQL Cache: SELECT with a number of rows written <> 0 [1] ; SQL Cache: sort time = considerable [2]

1: A SELECT should have 0 number of rows written; sorts and joins are carried out in the sort heap; as long as the sort is carried out in the sort heap the number of rows written stays at 0; only if the sort spills over onto disk is the counter of rows written started

2: The meaning of “considerable”: Not every sort overflow is worth tuning, one that spills and takes 10 ms will not help you gain much time if you avoid the sort (you gain 10 ms) or at least the spill (you’ll gain around 9 or 9.5 ms); only small overflows that occur _very_ often or massive overflow that come with considerable sort time are worth looking into

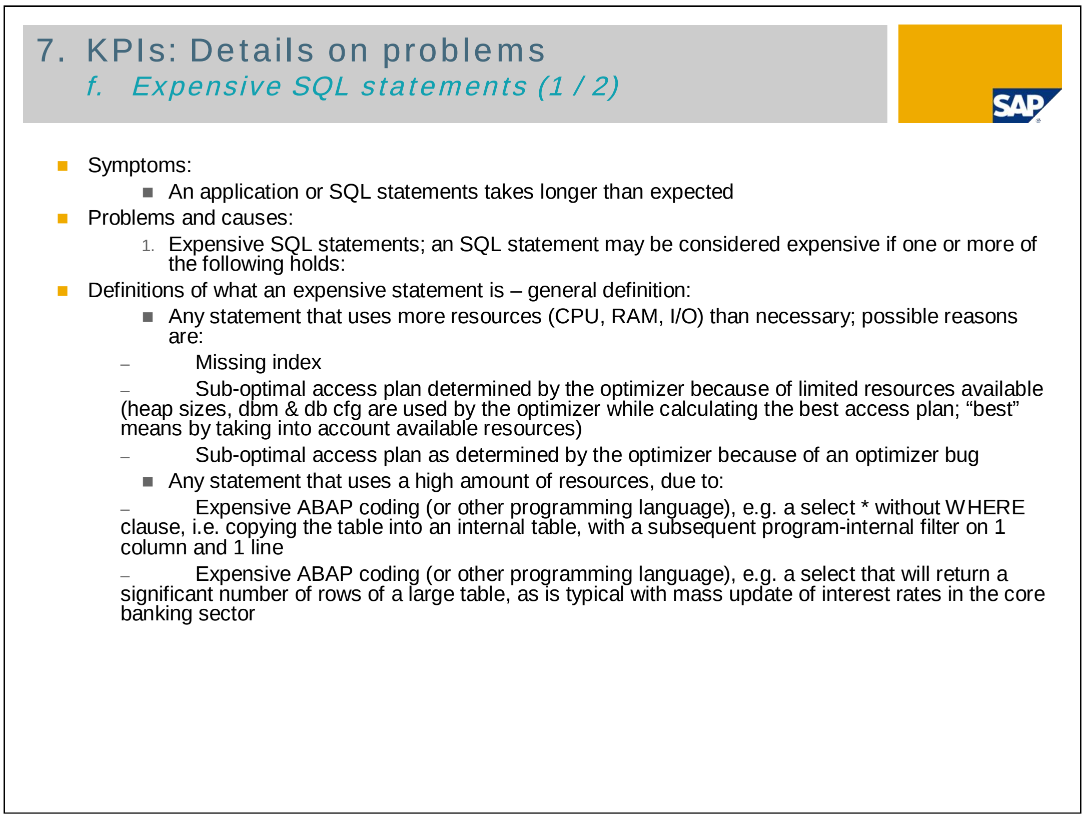


7. KPIs: Details on problems

e. Sorting problems (2 / 2)

- Thresholds:
  1. Sort heap & sort overflows
     1. Average sort time: > 1 ms (good systems often – mostly depending on the size of the sort required by the applications – are below 0.1 ms)
     2. % of sort overflows: < 1% (good systems often have < 1/1000) [1]
  2. SQL statements
     1. When the access plan looks too expensive
     2. SQL Cache: cf. footnote #3
- Remedies and comments:
  - Like locks, sorts per se are nothing bad
    - Most of them complete in easily under 0.1 ms, often less
    - A problem occurs when the sort area is not large enough and the sort spills over into temporary table space and onto disk
- Analysis:
  - Usually, no further analysis is required at this point

1: A low percentage, however, can be deceptive, giving you a false sense of security: few sorts can be extremely expensive, which means that not many applications will suffer (as opposed to a high percentage of sort overflows), but the ones that do suffer a lot, which can result in incredibly bad performance



7. KPIs: Details on problems

f. Expensive SQL statements (1 / 2)

- Symptoms:
  - An application or SQL statements takes longer than expected
- Problems and causes:
  1. Expensive SQL statements; an SQL statement may be considered expensive if one or more of the following holds:
- Definitions of what an expensive statement is – general definition:
  - Any statement that uses more resources (CPU, RAM, I/O) than necessary; possible reasons are:
    - Missing index
    - Sub-optimal access plan determined by the optimizer because of limited resources available (heap sizes, dbm & db cfg are used by the optimizer while calculating the best access plan; “best” means by taking into account available resources)
    - Sub-optimal access plan as determined by the optimizer because of an optimizer bug
  - Any statement that uses a high amount of resources, due to:
    - Expensive ABAP coding (or other programming language), e.g. a select * without WHERE clause, i.e. copying the table into an internal table, with a subsequent program-internal filter on 1 column and 1 line
    - Expensive ABAP coding (or other programming language), e.g. a select that will return a significant number of rows of a large table, as is typical with mass update of interest rates in the core banking sector

7. KPIs: Details on problems

f. Expensive SQL statements (2 / 2)

- Definitions of what an expensive statement is – most current definitions for expensive SQL statements (in DB2):
  - Long-running SQL statement
  - Statement that uses up a high percentage of the total time spent in the database
  - High percentage of all rows read
  - High number of rows read per execution
  - High percentage of all rows written when the statement is a SELECT statement: this shows a sort overflow or a join that overflowed
  - High number of rows written per execution with a SELECT statement
- Expensive SQL statements, how to identify them, how to analyze them, and how to tune them is an extensive field in its own right and will not be addressed in detail here but in (at least) 2 separate expert sessions (currently slated for Q1/2010)