MaxDB

Internals Workshop
Version 7.8

Heike Gursch
Christiane Hienger

THE BEST-RUN BUSINESSES RUN SAP™

MaxDB

Introduction to MaxDB Internals

Christiane Hienger

Overview

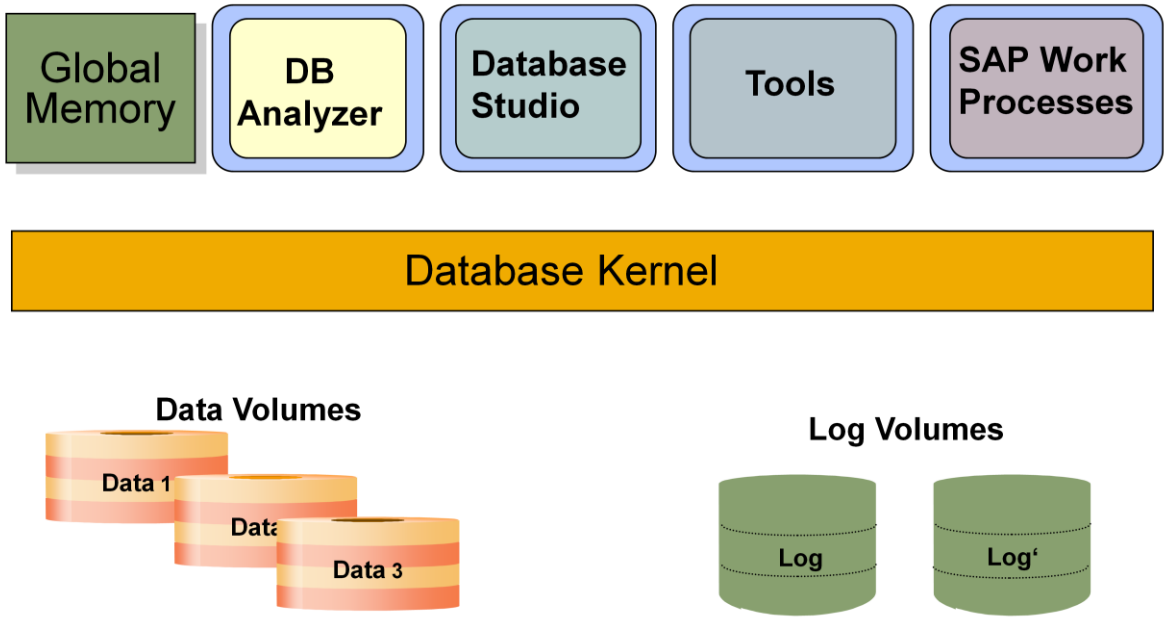
Database Kernel

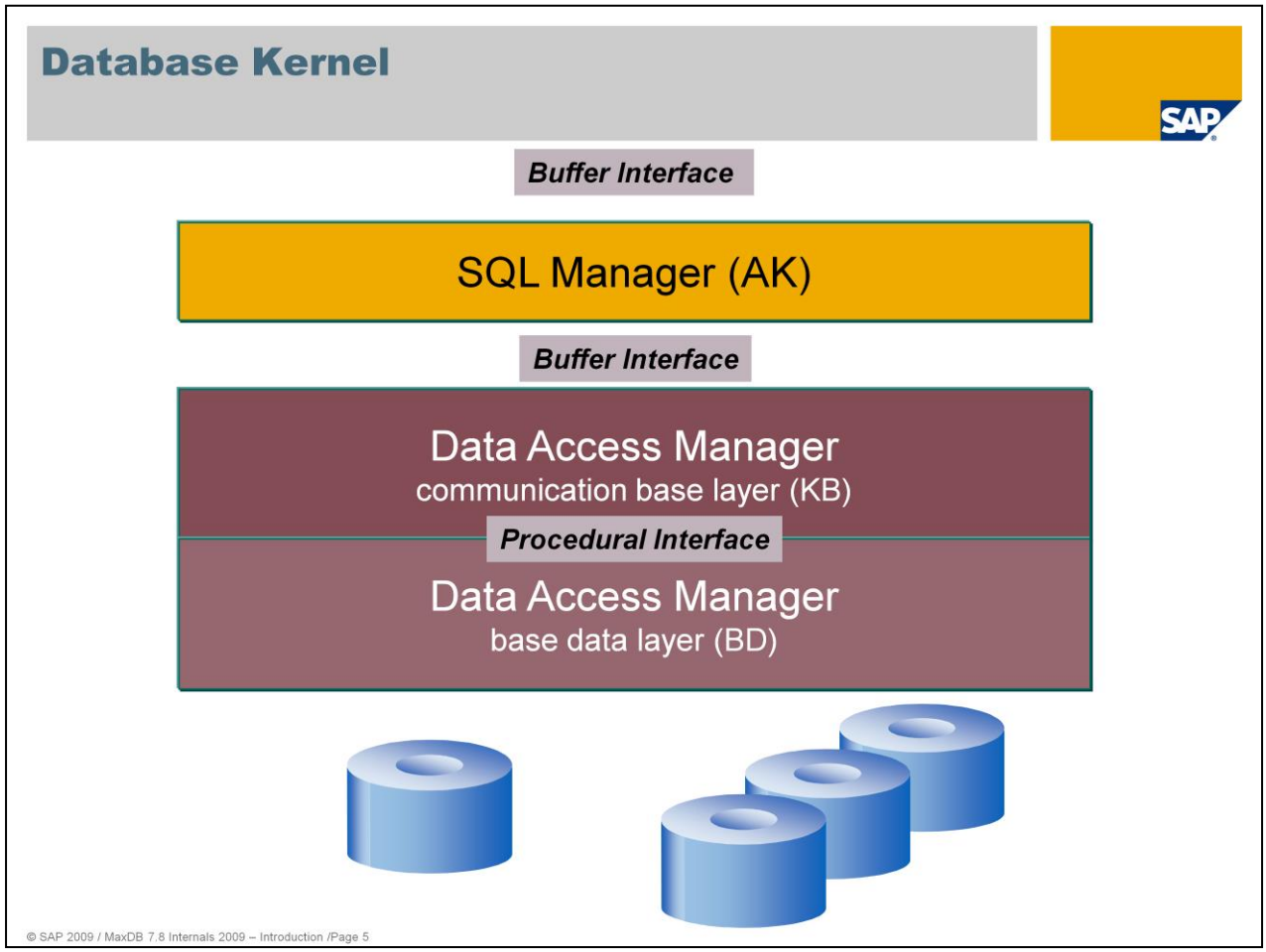
Process Layer

Memory Level

Data Storage

Environment





The MaxDB kernel consists of three levels:

SQL statements sent to the database system are received by the highest level, the SQL Manager (previously known as the application communication layer).

They are then sent in compressed form to the Data Access Manager. The Data Access Manager is divided into the communication base layer (KB) and the base data (BD) layer.

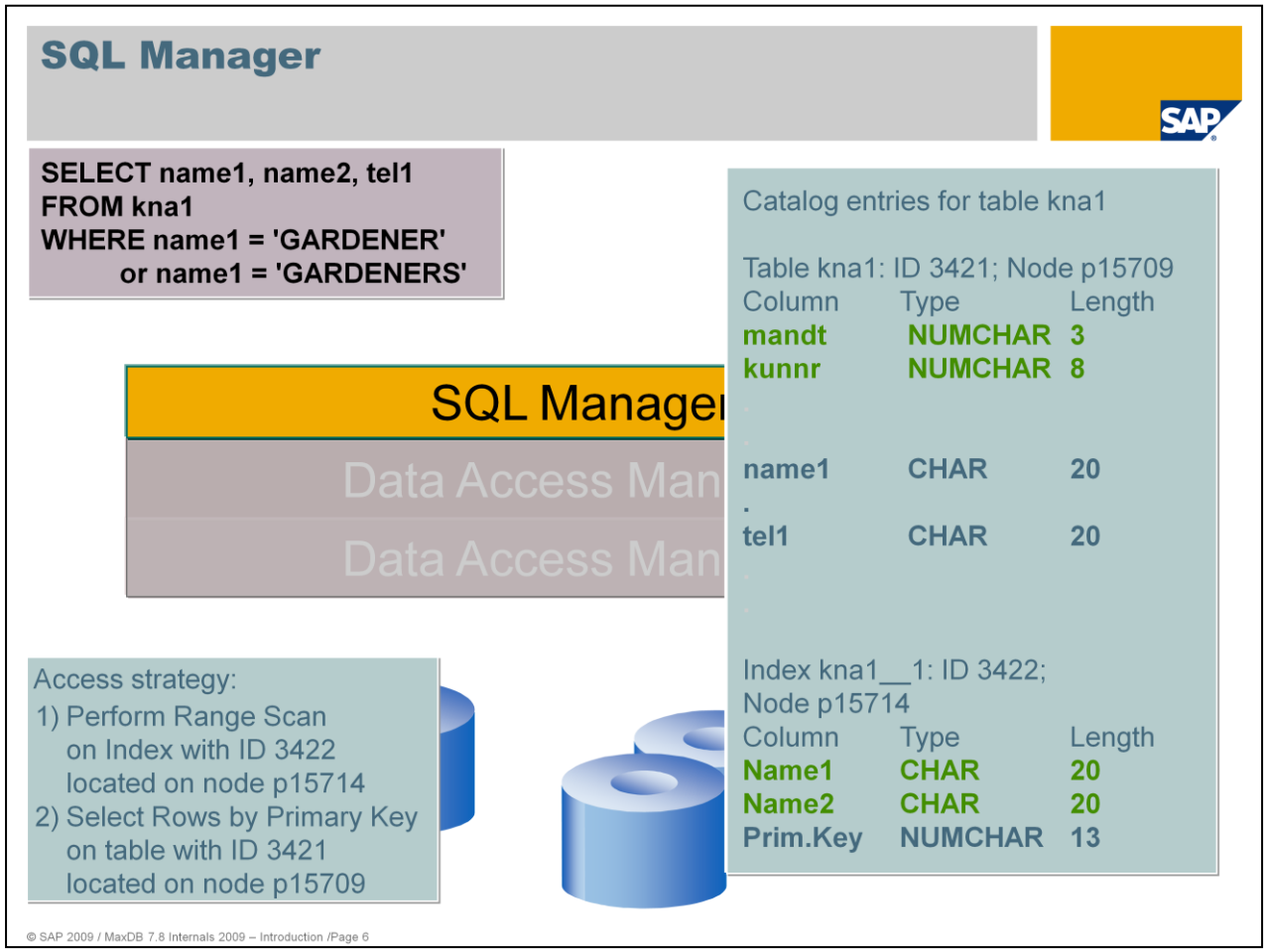
The necessary individual queries are determined on the communication base layer.

The required data is then procured on the lower-level base data layer.

The data records are then sent through the layers to the querying application.

In recent years, the KB and BD layers have increasingly merged.

Since version 7.4 there has no longer been a sharp division between the two layers.



Let's have a look at a relatively simple SELECT statement:

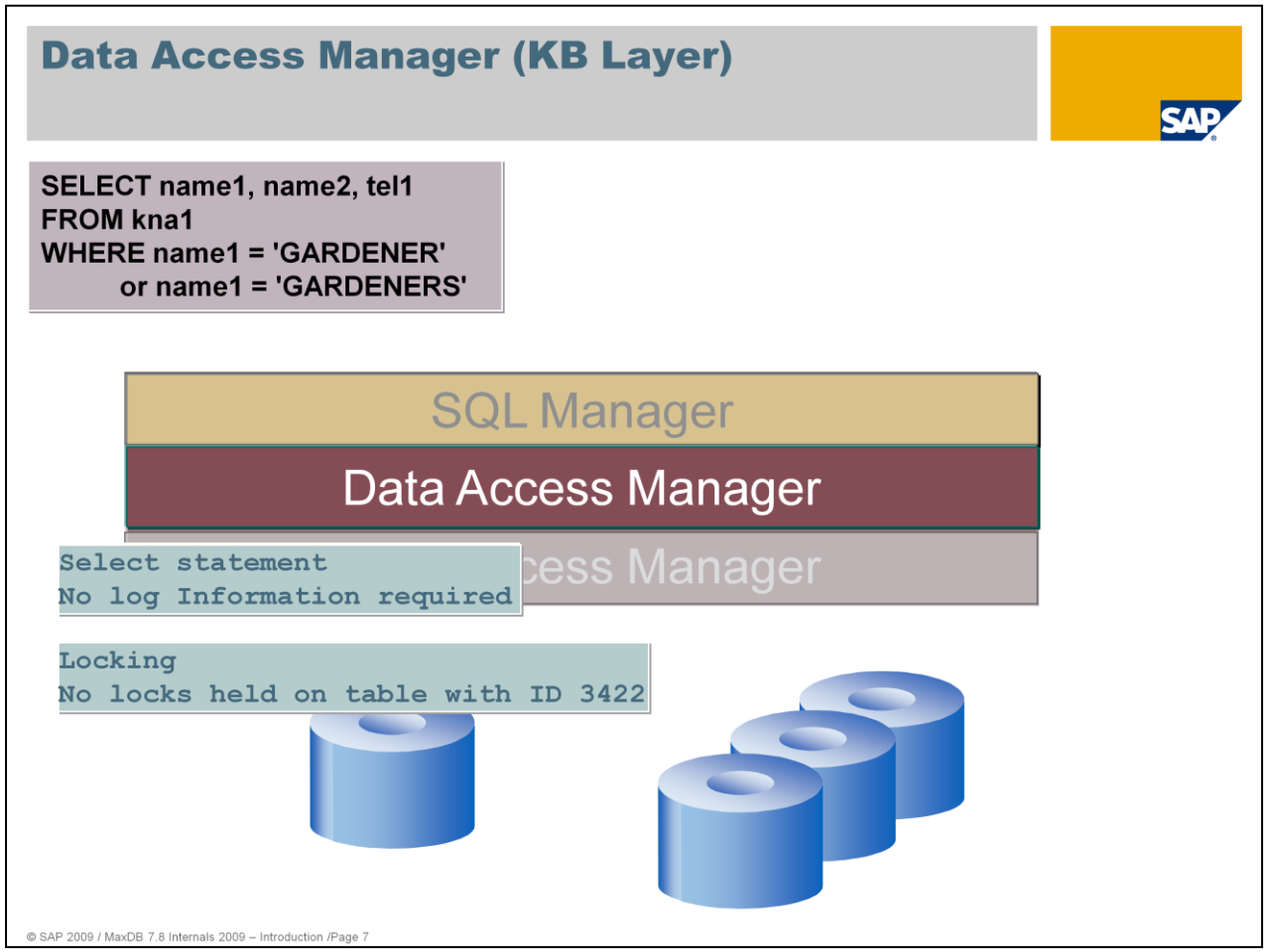
The SQL Manager receives an SQL statement that was sent as an SQL packet and first checks the syntax for correctness.

It converts the incoming SQL packets into stack code with various entries, each with a length of 8 bytes.

Catalog management, that is, the determination of information about the existing tables and their columns, also takes place on this layer.

Finally, the Optimizer in the SQL Manager decides if the data should be accessed via a secondary index.

In our simple example, we assume that the Optimizer has opted for an access strategy via the secondary index kna1_1 .



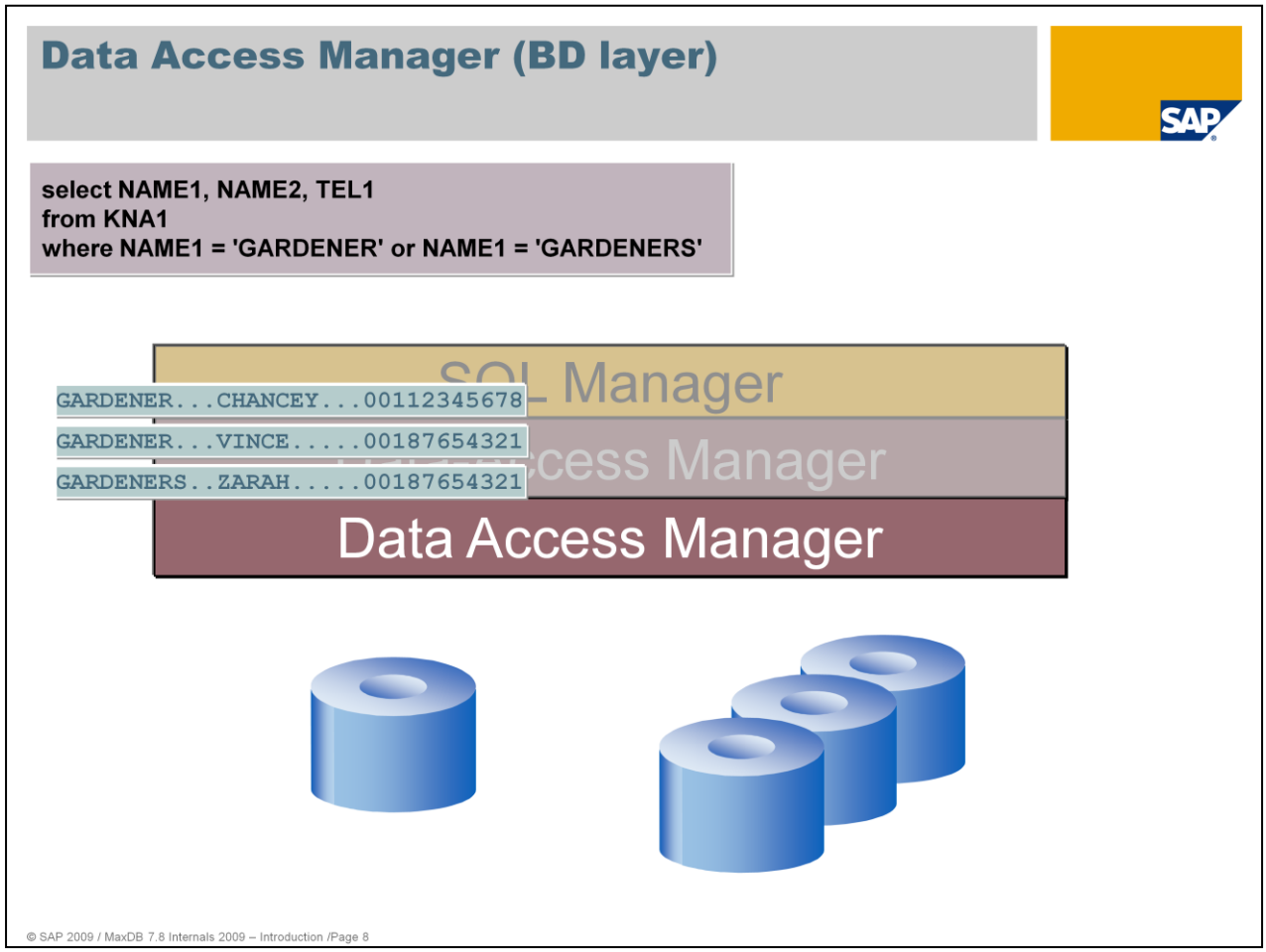
Transaction management and lock management are executed on the communication base layer of the Data Access Manager.

Transaction management manages information about active transactions in the database as well as the changes they make to data.

Lock management provides memory lists containing all objects of an instance that are locked at a certain point in time as well the transactions that correspond to them.

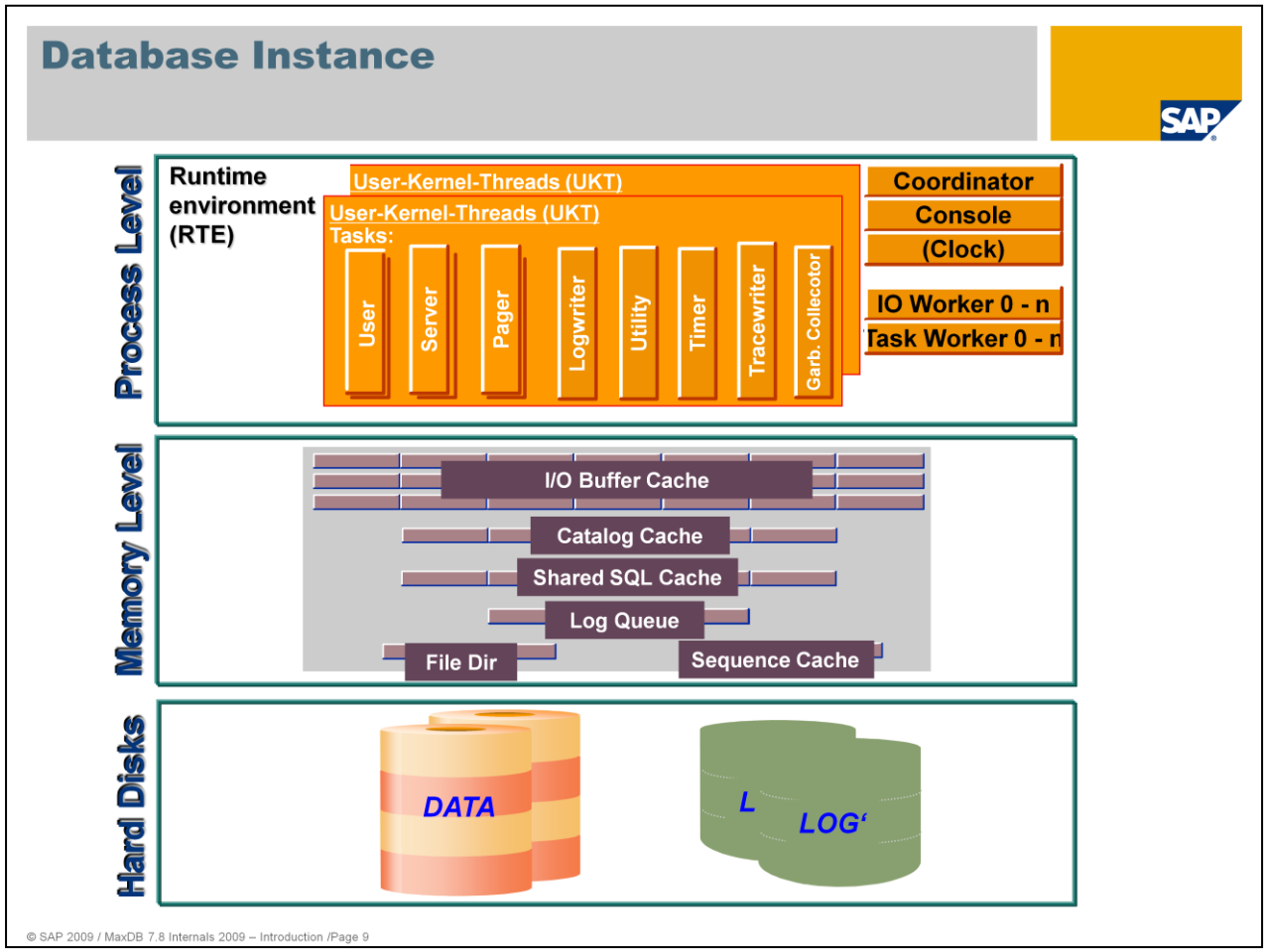
In the case of complex SQL statements (such as joins), access to individual tables and indexes is coordinated.

In our simple example, the KB layer will first access the index table kna1_1 and then, using the determined index entries, directly access the appropriate rows of table kna1.



The base data layer of the Data Access Manager executes accesses to data that is either still located in the data volumes or is already in the cache.

The BD layer returns the results records to the KB layer.

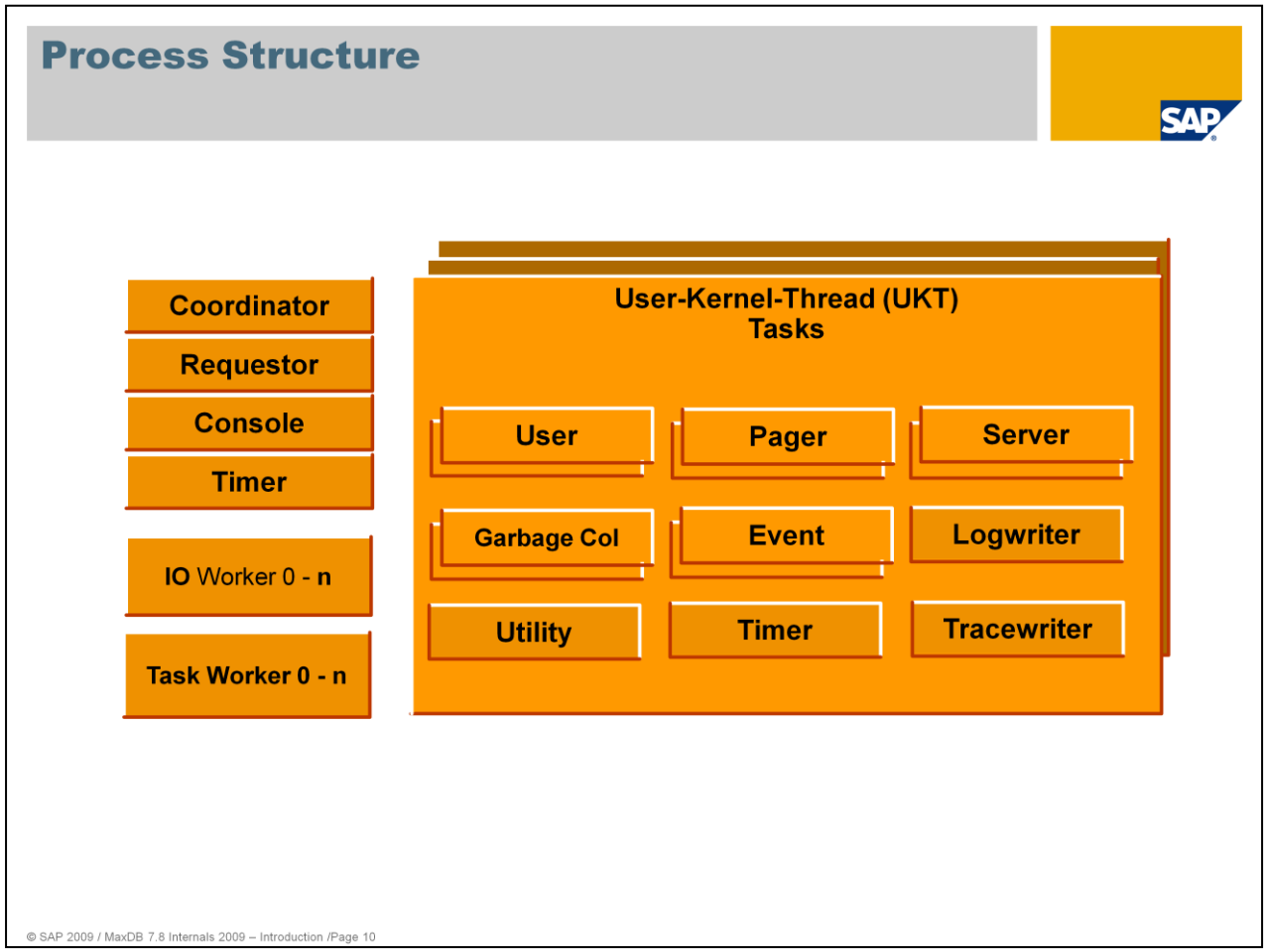


A database instance consists of the following three parts:

Database kernel (process level)

Caches (memory level)

Volumes (hard disk level)

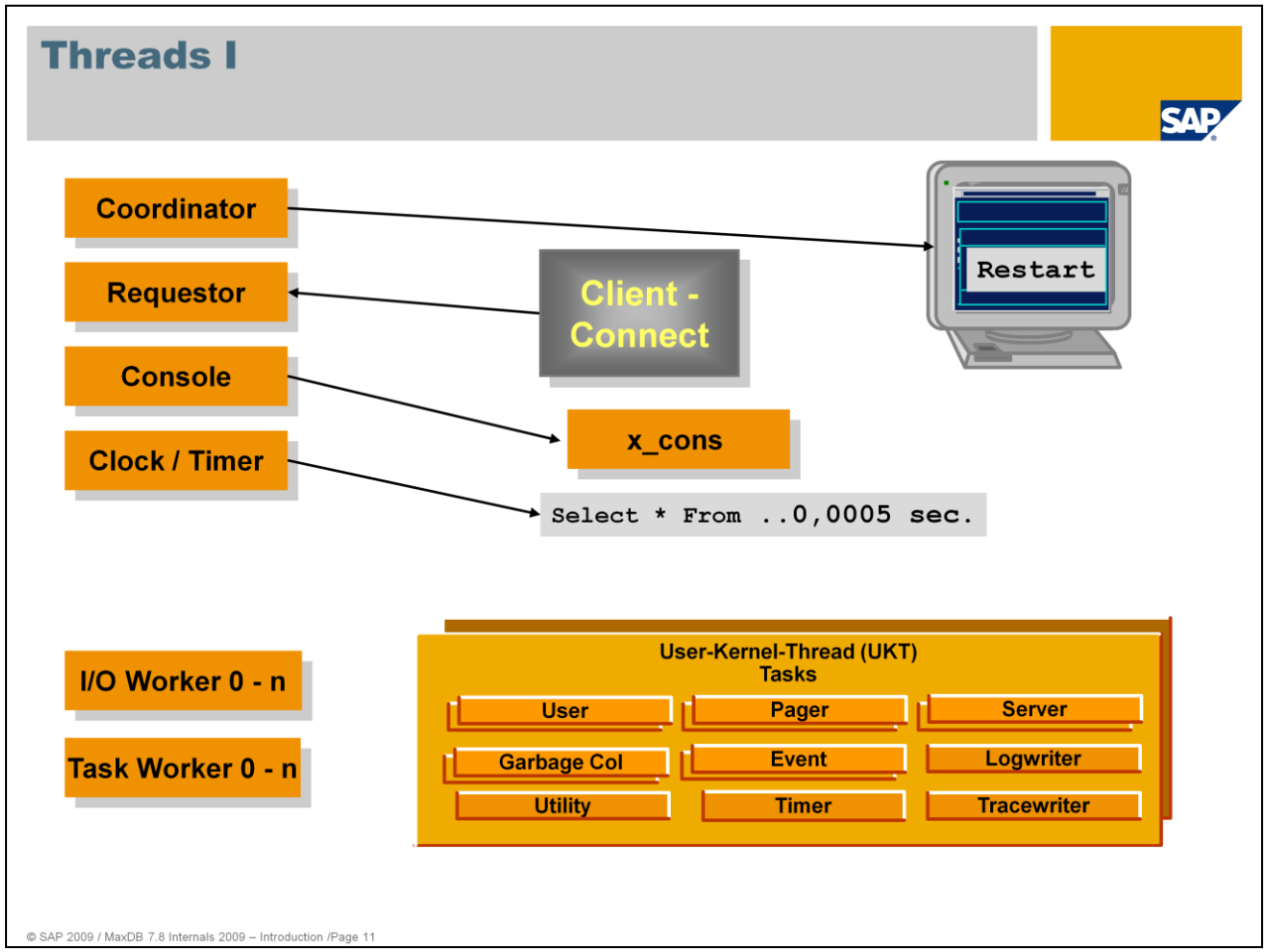


A user kernel thread forms a subset of all tasks (internal tasking).

The database kernel runs as one process divided into threads. Threads can be active in parallel on several processors within the operating system. Threads perform various tasks.

User kernel threads (UKT) consist of several tasks that perform various tasks. This tasking enables more efficient coordination of tasks than operating system tasking that employs individual threads.

The runtime environment (RTE) defines the structure of the process and the user kernel thread.



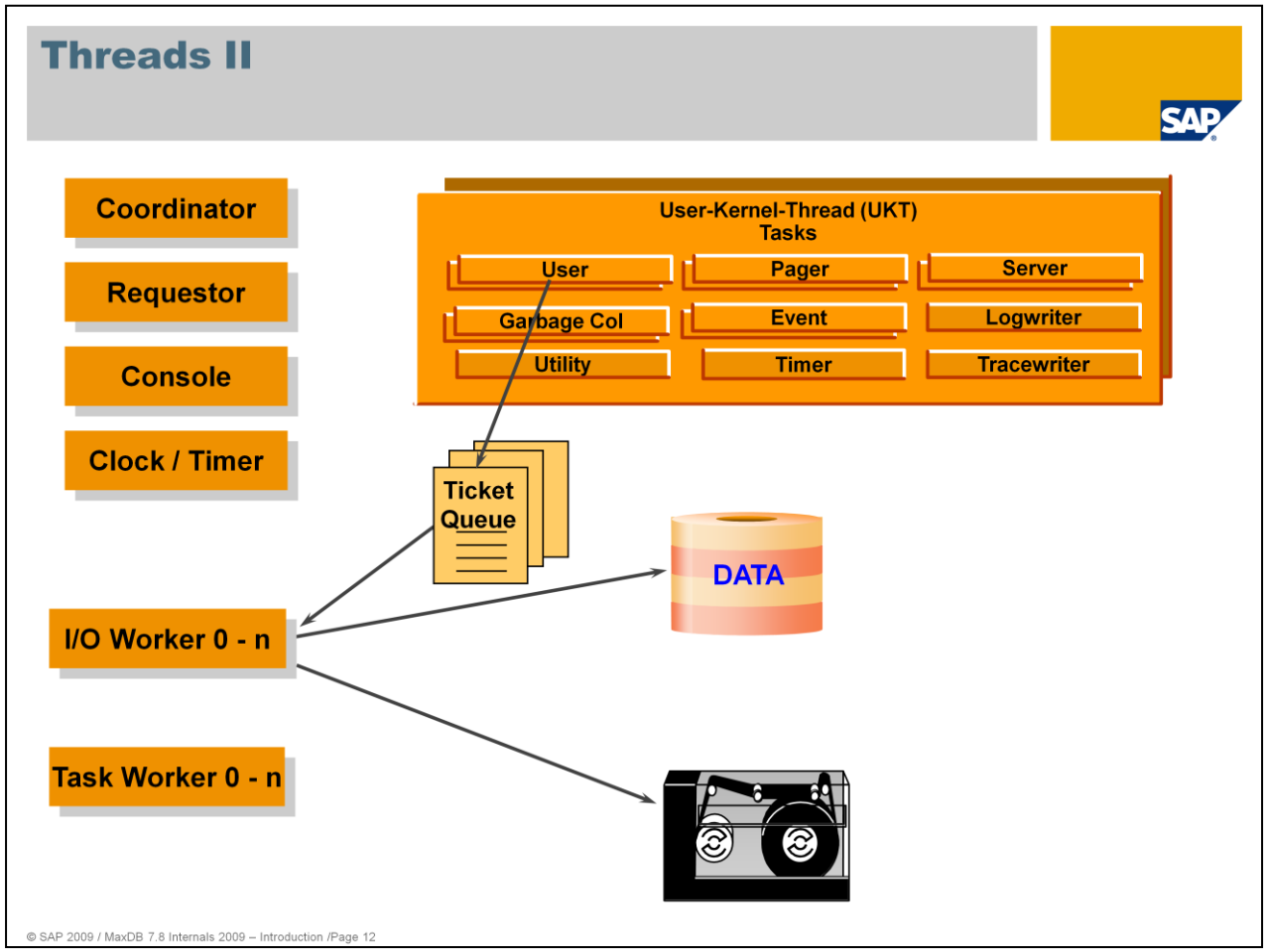
When the runtime environment is started, that is, when the database instance is started in the Admin state, first the **coordinator thread** is generated. This thread is of particular importance.

- When started, the coordinator thread uses database parameters to determine the configuration of the memory and process of the database instance. For this reason, changes to database parameters take effect only after you have restarted the database instance.
- The coordinator thread also coordinates the start procedures of the other threads and monitors them while the database instance is in operation.
- If operating errors occur, the coordinator thread can stop other threads.

The **requestor thread** receives logon requests from the user processes to the database. The logon is assigned to a task within the user kernel thread.

The **console thread** collects information about the internal states of the database kernel when x_cons is being used.

The **clock thread** and the **timer thread** calculate internal times, for example to determine how much time was required to execute an SQL statement.



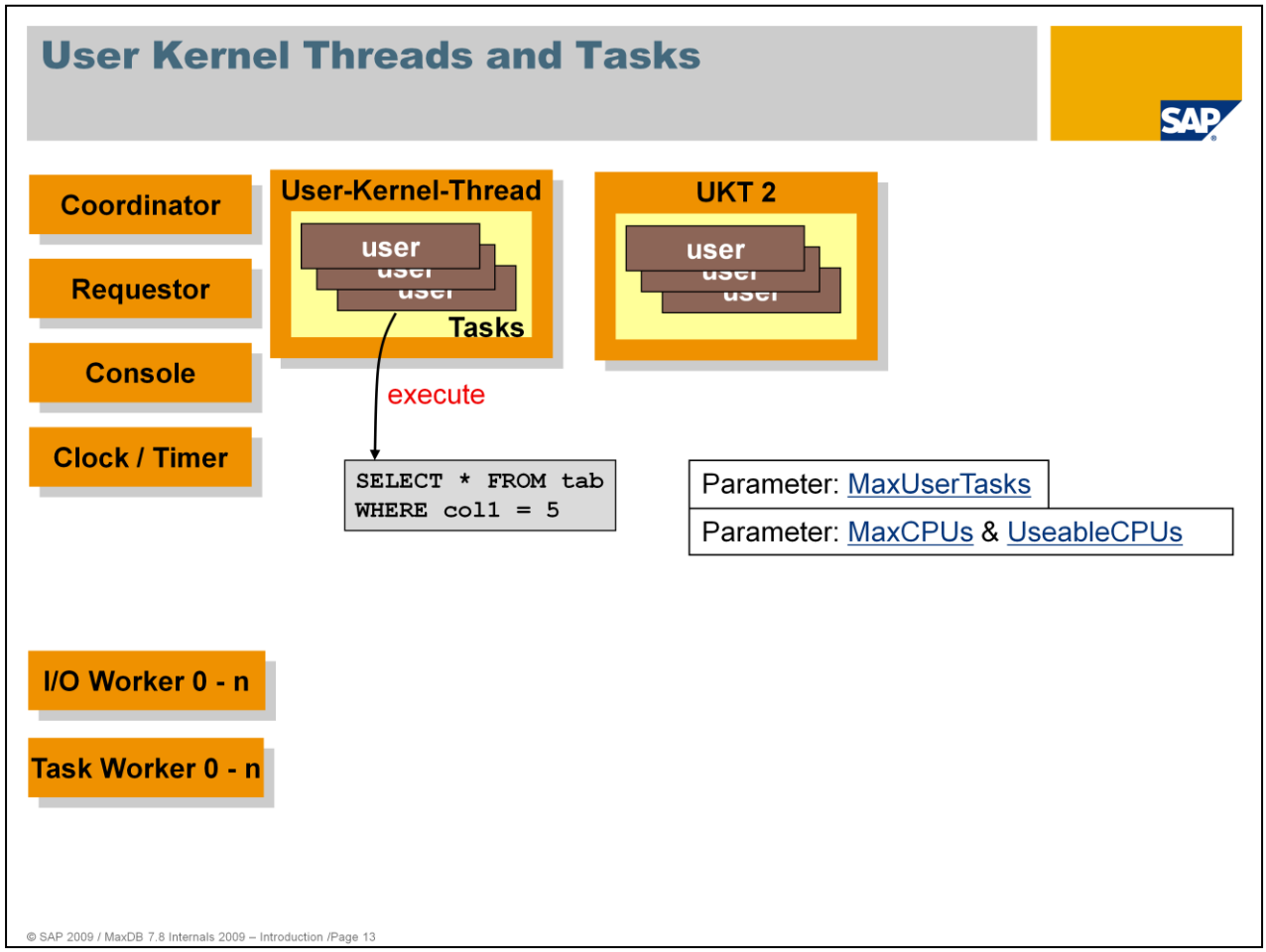
I/O threads are responsible for processing the write and read requests to and from data and log volumes that are requested by the corresponding tasks. MaxDB supports asynchronous I/O requests. On Windows, the asynchronous I/O of the operating system is used.

Until version 7.6 the number of **I/O threads** is primarily dependent on the number of volumes in the database instance. As a rule, two **I/O threads** are activated for each data and log volume and one for the writing of the database trace

As of version 7.7 **I/Oworker threads** are taken from a pool and activated on request; I/O is done asynchronously. After finishing the I/O requests the workers are returned to the pool.

As of verison 7.8 the user tasks utilize the asynchronous I/O for scans. The user tasks send several parallel I/O orders to the I/O systems. They don't wait until the I/O system has read every single block from disk.

The Task Worker Threads are not used for I/O requests. User tasks use task Worker threads to execute orders asynchronously e.g. In hot stand by environemnt to send the log position to the stand by node.



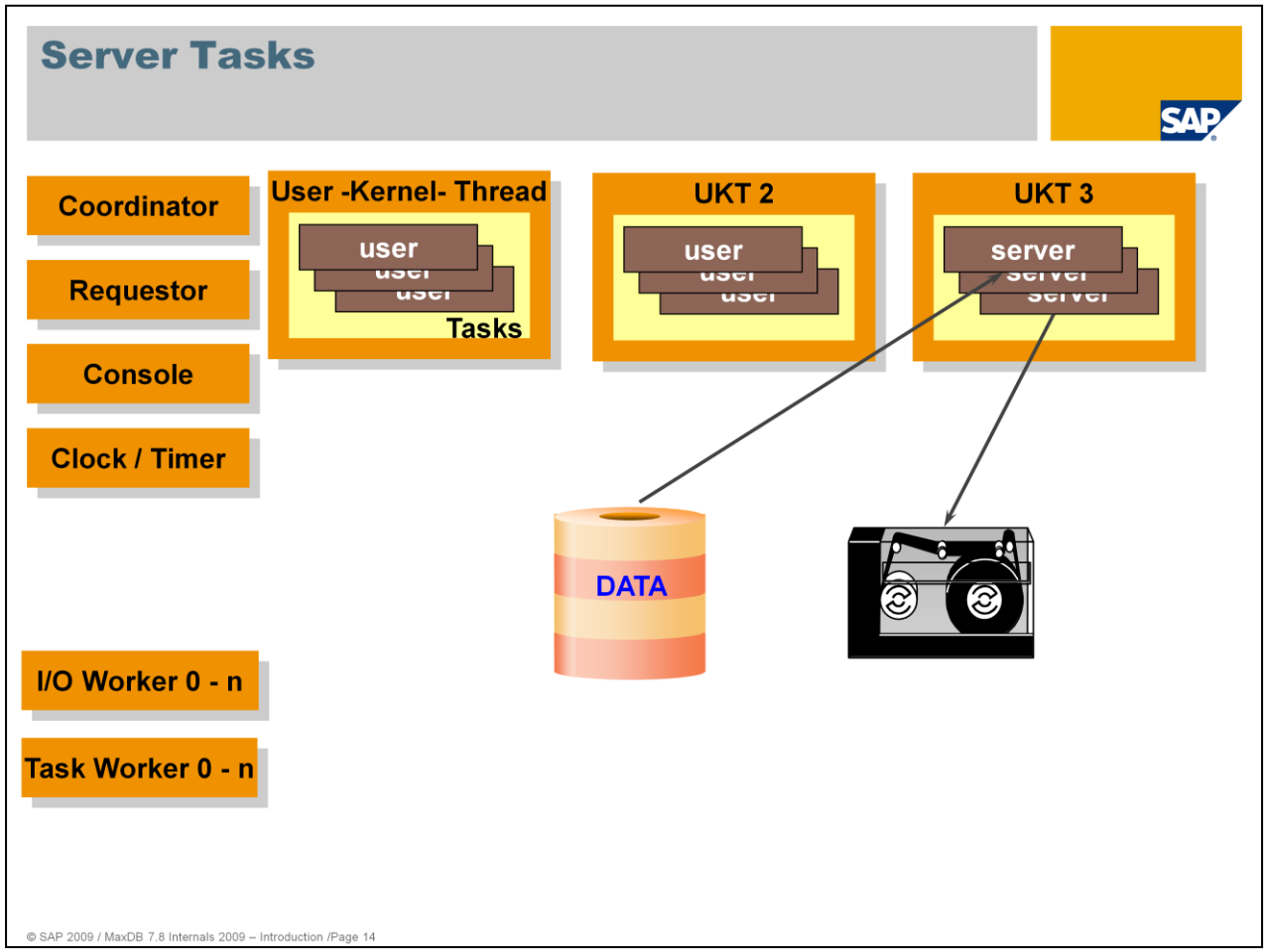
Each user session is assigned exactly one **user task** at logon.
The maximum number of available user tasks is determined by the database parameter **MaxUserTasks**. This parameter also restricts the number of user sessions that can be logged on to the database system simultaneously.
The database parameter **MaxTaskStackSize** determines the memory usage of the user tasks.

The general database parameter **MaxCPUs** specifies the number of user kernel threads among which the user tasks are distributed. Other tasks and global threads use very little CPU time. The parameter **MaxCPUs** allows you to specify how many processors the database should use in parallel.

The parameter **UseableCPUs** allows an online adjustment of the number of used user kernel thread. This makes dynamic configuration changes according the available CPUs in the system possible.

As of version 7.4.03, user tasks can switch from one UKT to another if the previously-responsible UTK is overburdened. This results in better scaling for multiprocessor servers (SMP). To use this function, set the parameter **LoadBalancingCheckInterval** to a value greater than 0.

As of version 7.8 Load Balancing is released for MaxDB and liveCache instances and used by default. The scheduler immediately moves the task to an idle user kernel thread if the current thread is overloaded.

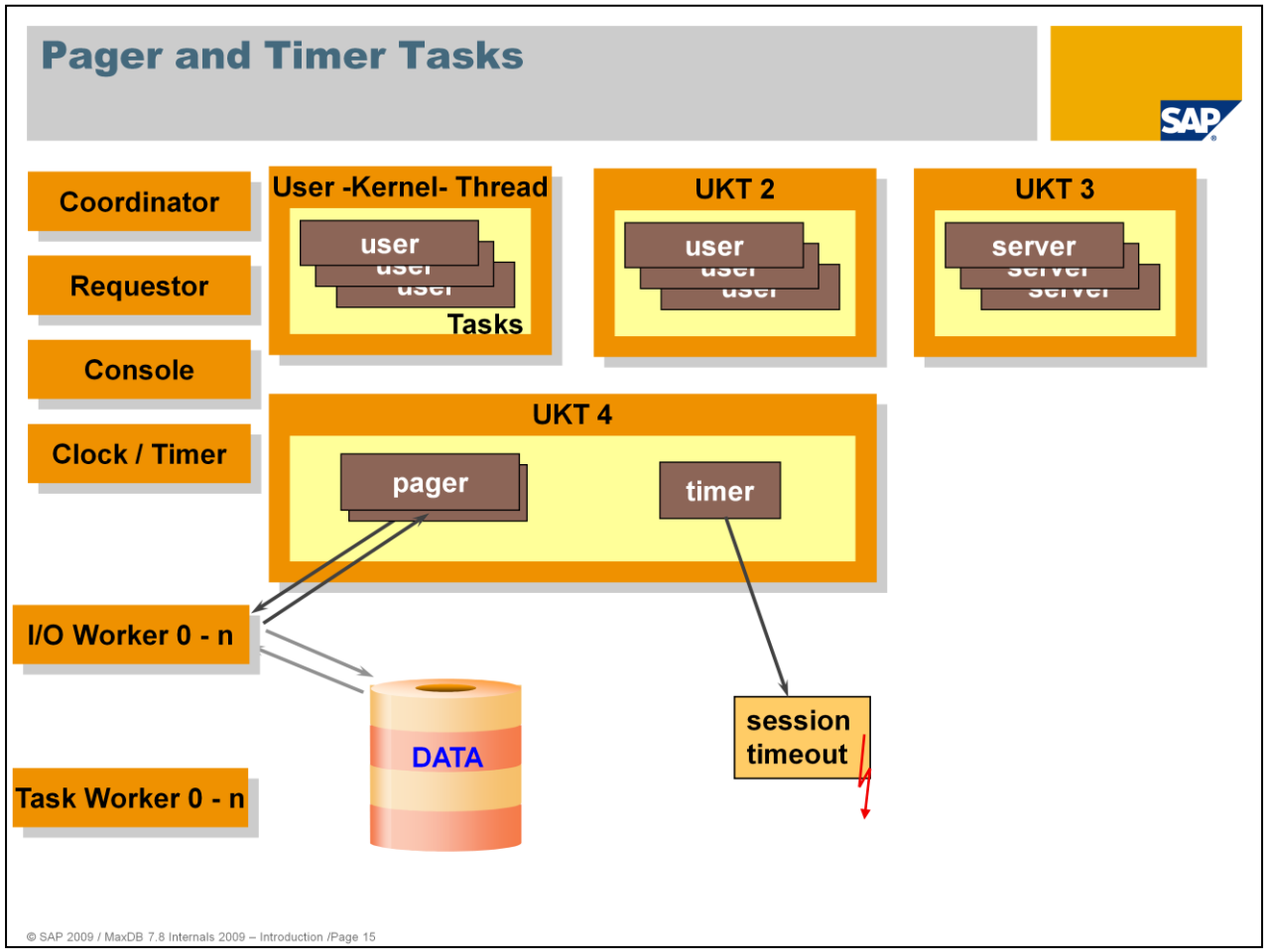


Server tasks are primarily used to back up data. Some server tasks read from the data volumes; others write to the backup medium.

The CREATE INDEX statement instructs the server tasks to read the table data in parallel from the data volumes.

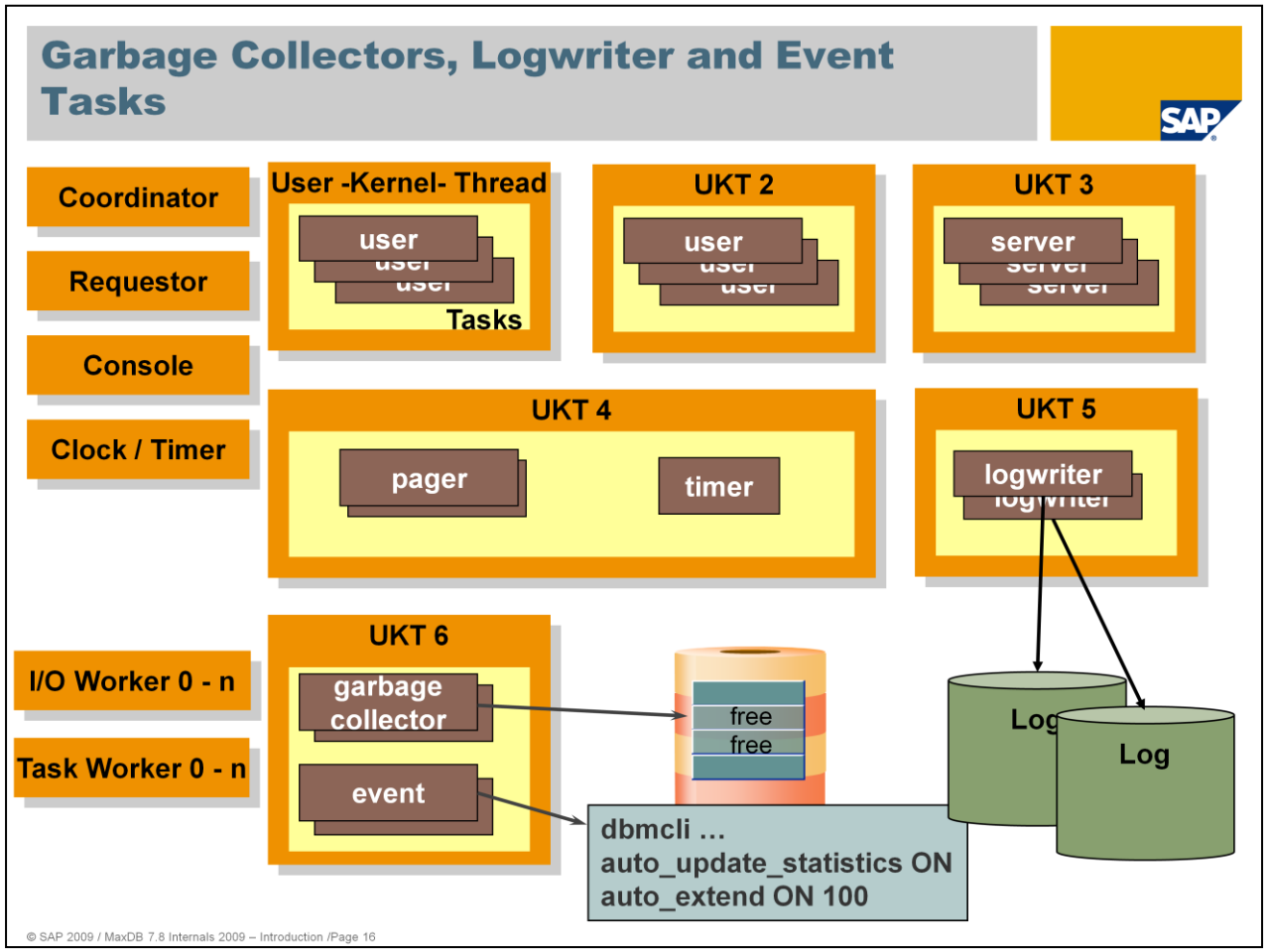
The system automatically calculates the number of server tasks needed in the configuration of the database instance from the number of data volumes and the number of backup devices.

As of version 7.6 the server tasks in certain cases are named reflecting their assigned task.



Pager tasks are responsible for writing data from the data cache to the data volumes. They become active when a savepoint is being executed.
The system calculates the number of pagers. It depends primarily on the data cache size and the number of data volumes.

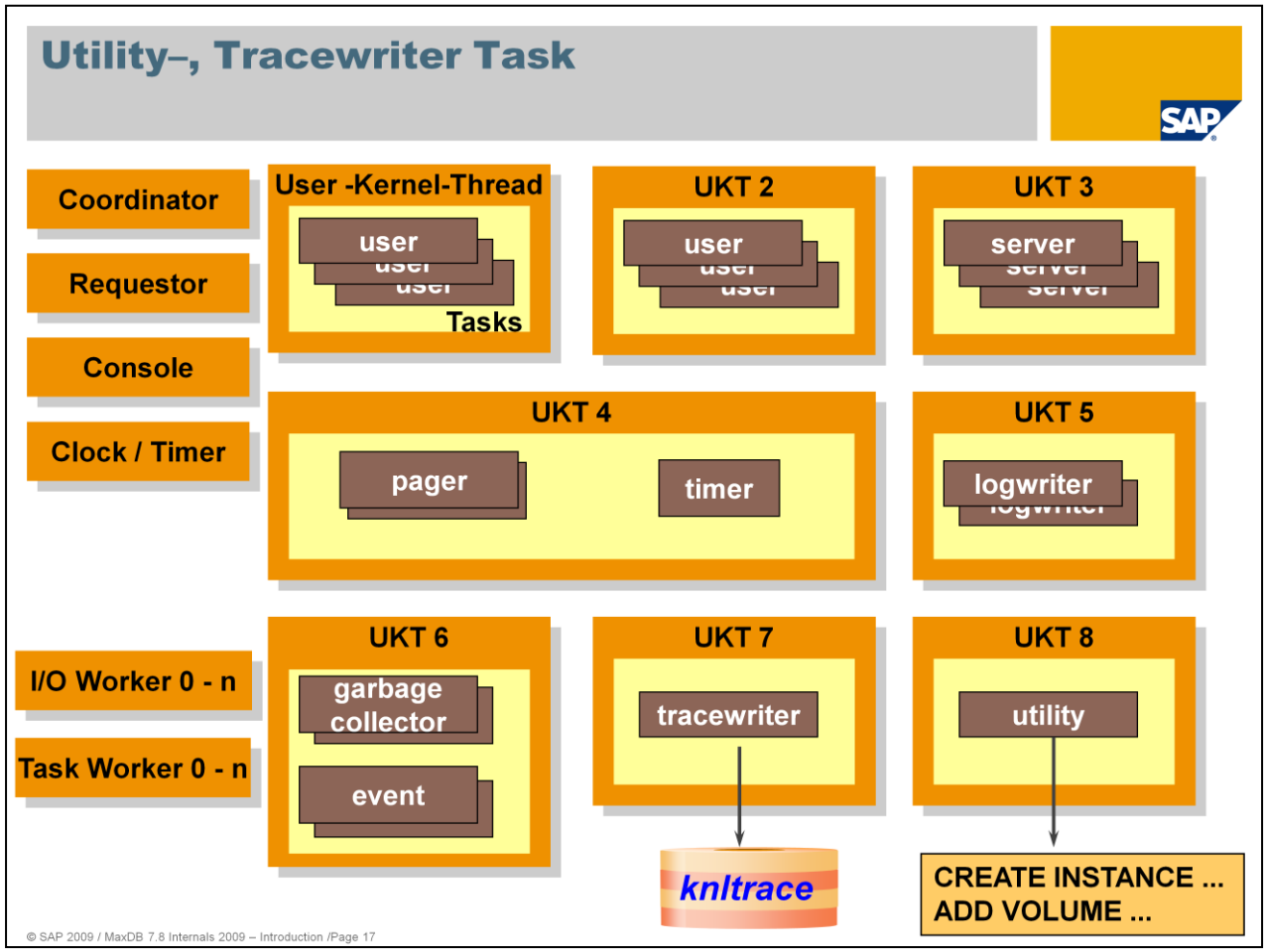
The **timer task** is used for handling all types of timeout situations (such as session timeouts and lock request timeouts).



The **Logwriter task** is responsible for writing redo log entries in the log volumes.

Garbage collectors release undo log files to free space management. With DROP TABLE statement, Garbage collectors delete the data in the tables asynchronously. Users do not have to wait for all data to be deleted.

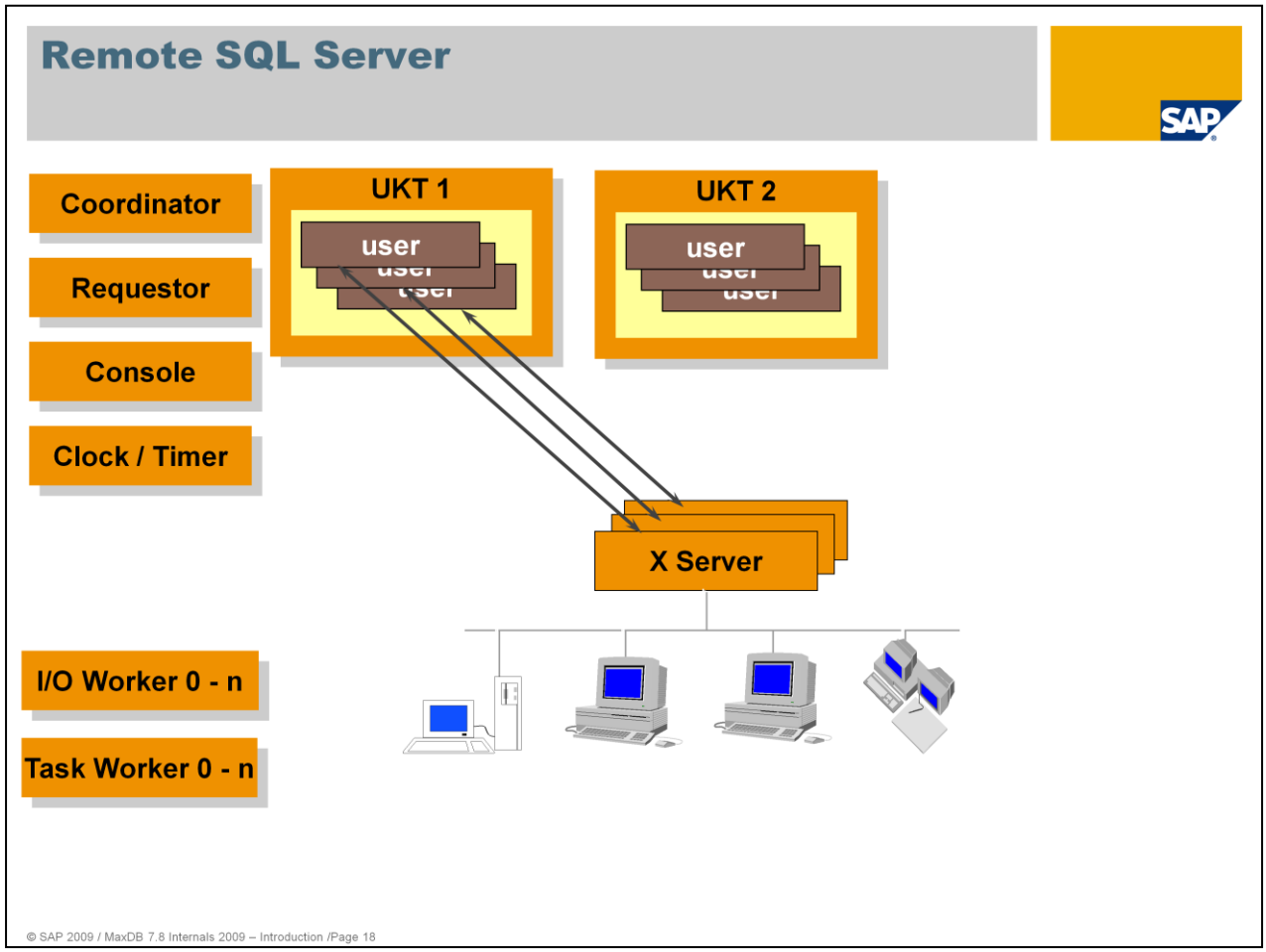
Event tasks send messages about database events to the Database Manager (e.g. DBMGUI). You can use the Event Dispatcher to define reactions. For example, you can have the database enlarge automatically when it gets full (auto_extend). You can also have the Event Dispatcher automatically update statistics when certain events occur (auto_update_statistics).



MaxDB offers the possibility of writing a special log, the database trace. The active tasks write the trace data to a buffer. If requested, the **trace writer task** writes the data from the buffer to the file knltrace.

The **utility task** is reserved exclusively for the administration of the database instance.
Automatic log backup can be executed in parallel with other administration tasks as it does not occupy a utility session after it is activated.

As of version 7.6 all administration actions are executed via user tasks. For compatibility reasons, the utility task will be retained for the foreseeable future although it will not be used by the DBM server in future versions.

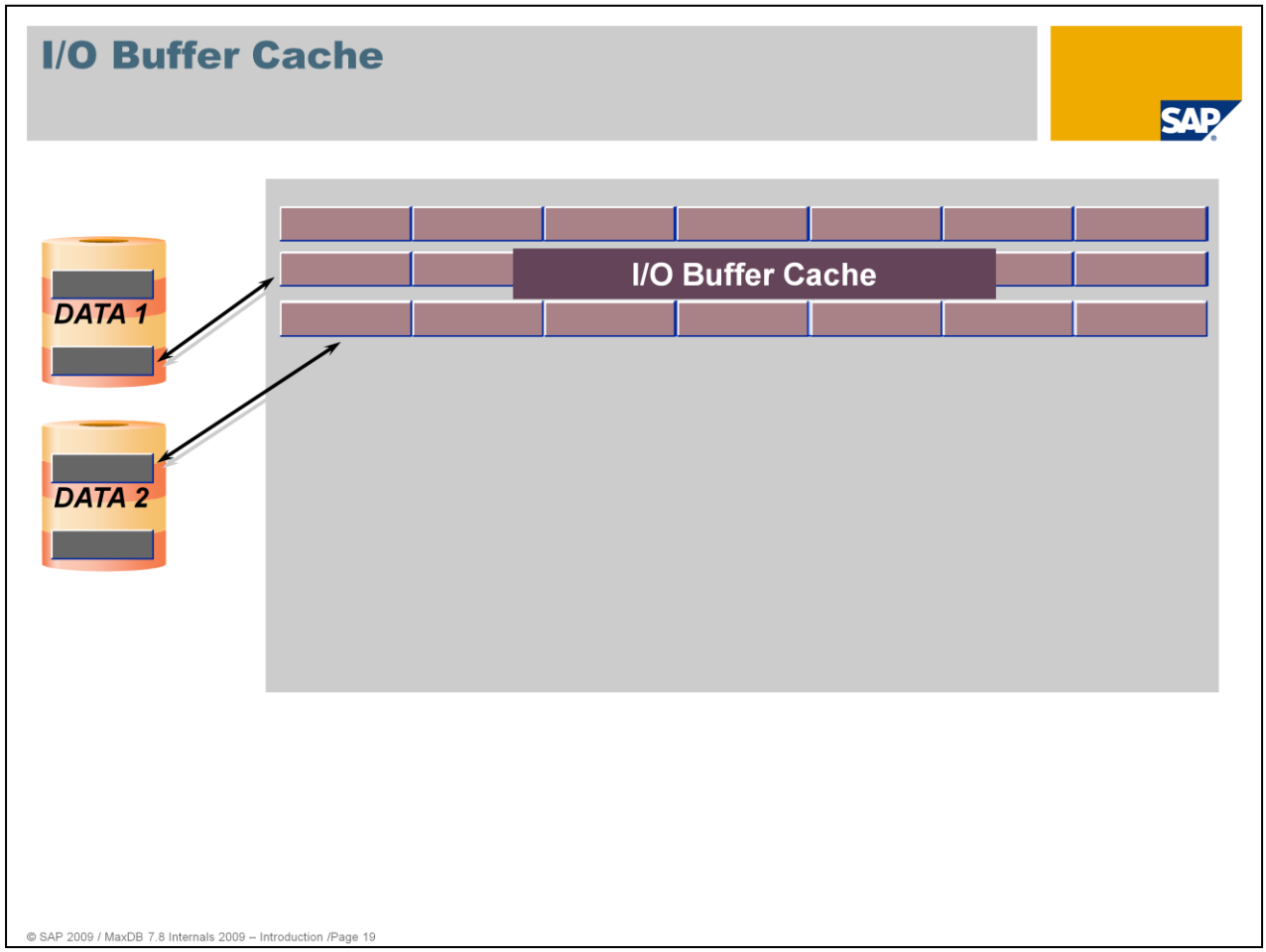


The **X-Server** is the communication server of the database system. It listens at a service port for connection requests from database applications and database tools. In the process list, this process is called **vserver**. A new vserver process is generated for every user process that logs on to the database remotely. The generating process serves the user; the new process waits for the next user logon. On Windows, an additional thread is started for the user logon.

On Windows, the X-Server runs as a service.

Local user sessions communicate with the database instance via a shared memory.

MaxDB 7.8 introduced the isolated software installation. Every database installed for SAP application uses it's own port number. Clients first connect to a global listener which returns the instance specific port number. The client then connects to the x_server assigned to the instance.



Read and write operations to the volumes are buffered in order to save time-critical disk accesses. The corresponding main memory structures are called caches. They can be dimensioned according to the user profile.

The **I/O buffer cache** contains the last read- or write-accessed pages of the data volumes. It is shared by all simultaneously active users.

The hit rate, that is, the ratio of successful accesses to the total number of accesses to the **I/O buffer cache**, is a crucial measure of performance. It should be greater than 98%. Successful access means that the required data was already available in the data cache.

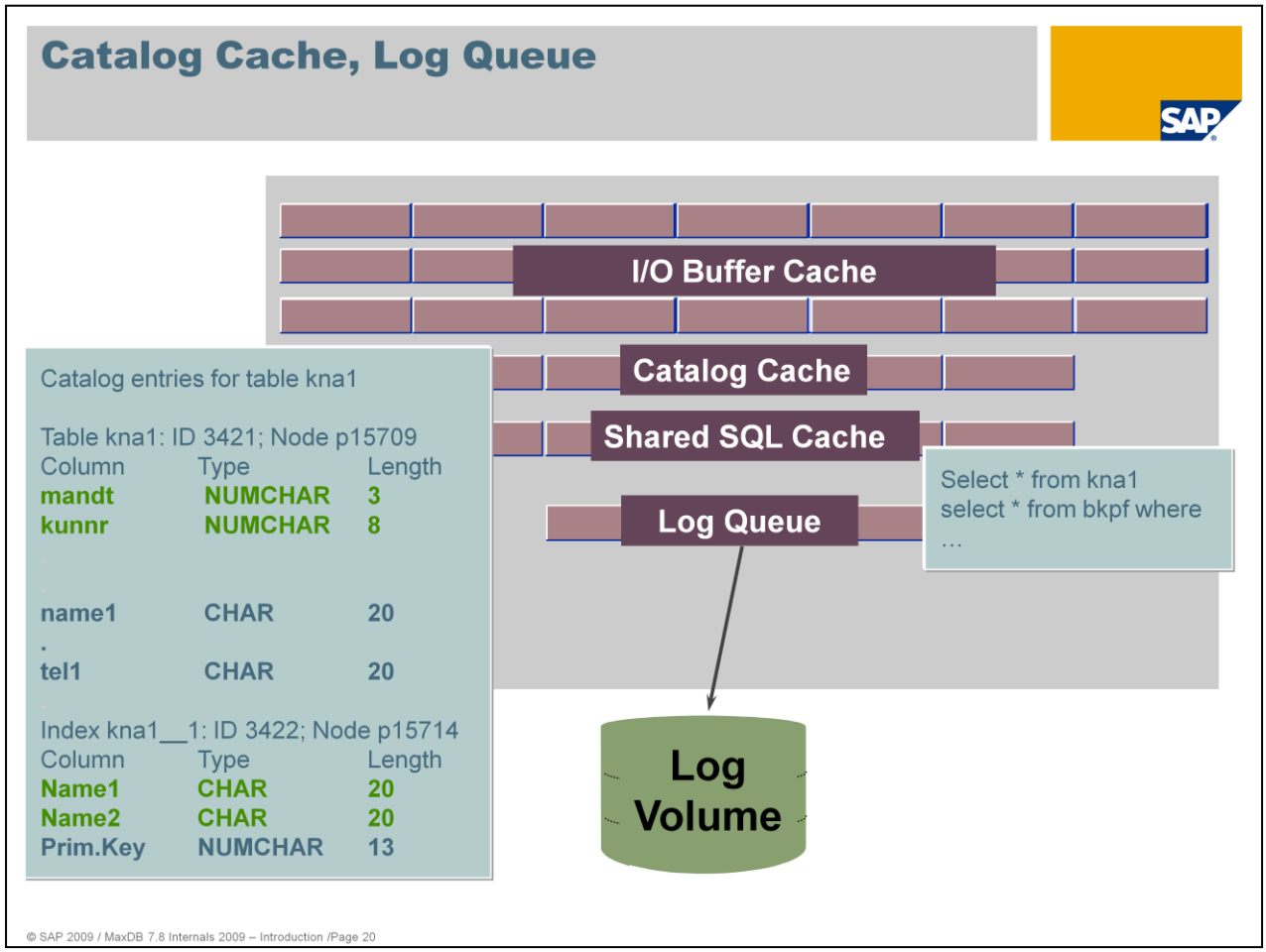
In addition to data pages, the I/O buffer cache also contains **converter pages**. Converter pages, like data pages, are stored in the data volumes. They store the assignment of the logical data page numbers to their physical position in the data volumes.

The number of converter pages is calculated automatically. It can increase with data growth. When deletions are performed, converter pages are released.

All converter pages are held in the cache. Each converter page contains 1861 entries that are managed for data pages.

The database parameter used for setting the size of the I/O buffer cache is called **CacheMemorySize**.

In versions < 7.4, data pages and converter pages were managed in separate caches, the data cache and the converter cache.



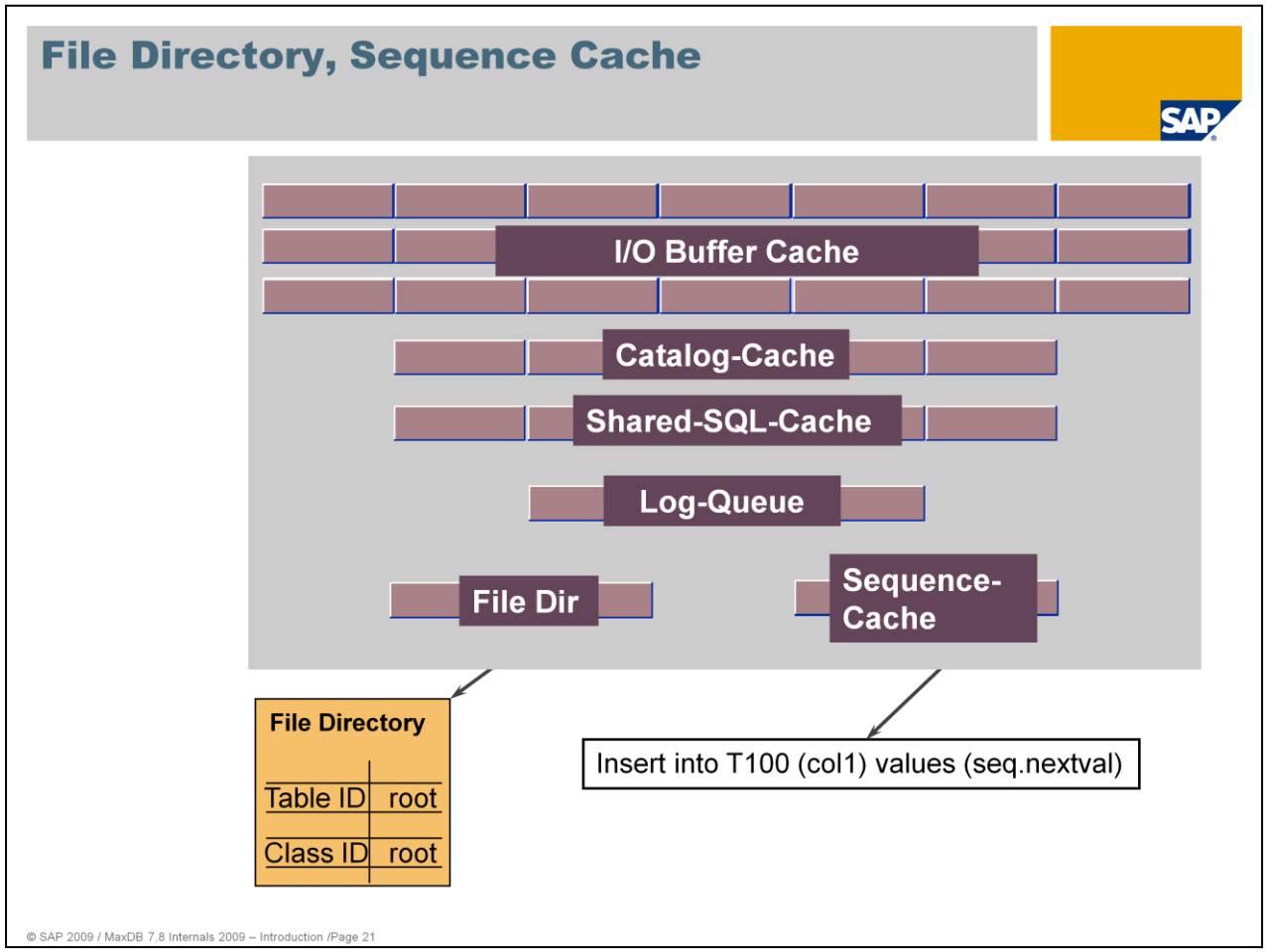
In the **catalog cache**, the database system stores user-specific data and global data from the database catalog. Data that has been displaced from the catalog cache is moved to the data cache. As of version 7.6 the catalog cache is a really shared cache. The information of this cache is shared among all users. In older versions, each user task was assigned to a separate area. The total of all catalog caches can increase up to the value that has been configured with the database parameter CAT_CACHE_SUPPLY.

All statements to be executed will be held in the **shared SQL cache** along with their execution plans. The shared SQL cache is shared by all users, that is, a statement is only accepted once. When shared SQL is not being used, the statements of each user are kept separately in the catalog cache. The setting NO for database parameter SHAREDSQL deactivates use of this cache.

A statement, together with the statement text, is stored only once in the shared SQL cache. This allows you to see which statements are active in the database at any time. The shared SQL manager also collects monitor data. Shared SQL offers the following advantages over versions prior to 7.6:

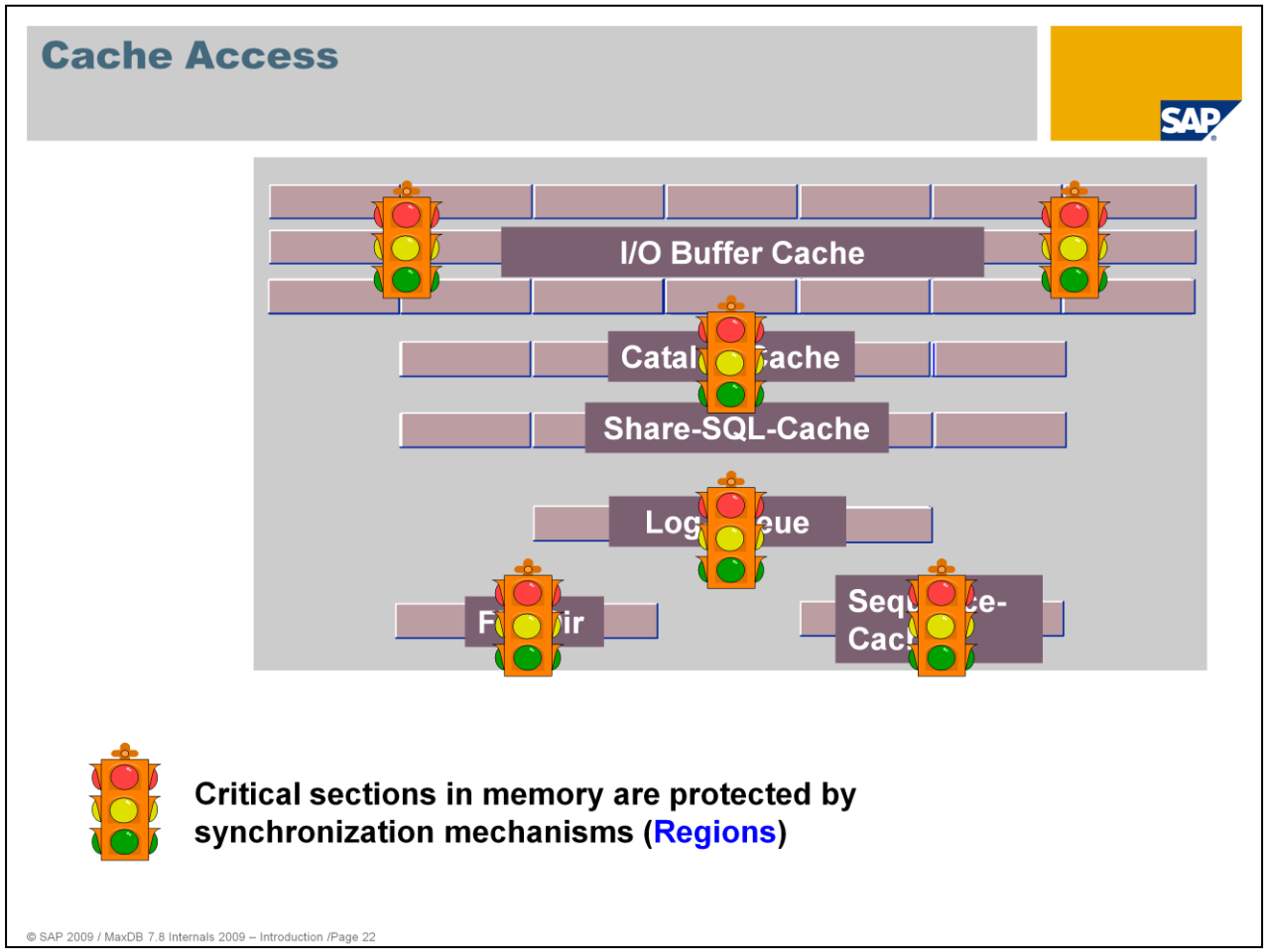
- A statement only has to be prepared once rather for each user.
- Statements are stored only once. That saves space in the main memory.
- Storing the data in the main memory offers better monitoring possibilities.

The **log queue** allows log entries to be written to the log volumes asynchronously and increases the likelihood of group commits. In a group commit, several write transactions are completed in the log area with an I/O.



The **file directory** is required for the internal organization of the database instance. The assignments of the root pages of the B*-trees to the table IDs and a type flag are stored here. The type flag specifies whether the table contains primary data, secondary key or LONG data. With version 7.6 implementation of the file directory has changed. It now holds addtional figures about numbers of rows and table sizes. Thus asking for the number of rows of even extremely large tables using “SELECT COUNT(*) FROM <table>” can be executed very fast.

The **sequence cache** stores current data on number generators.

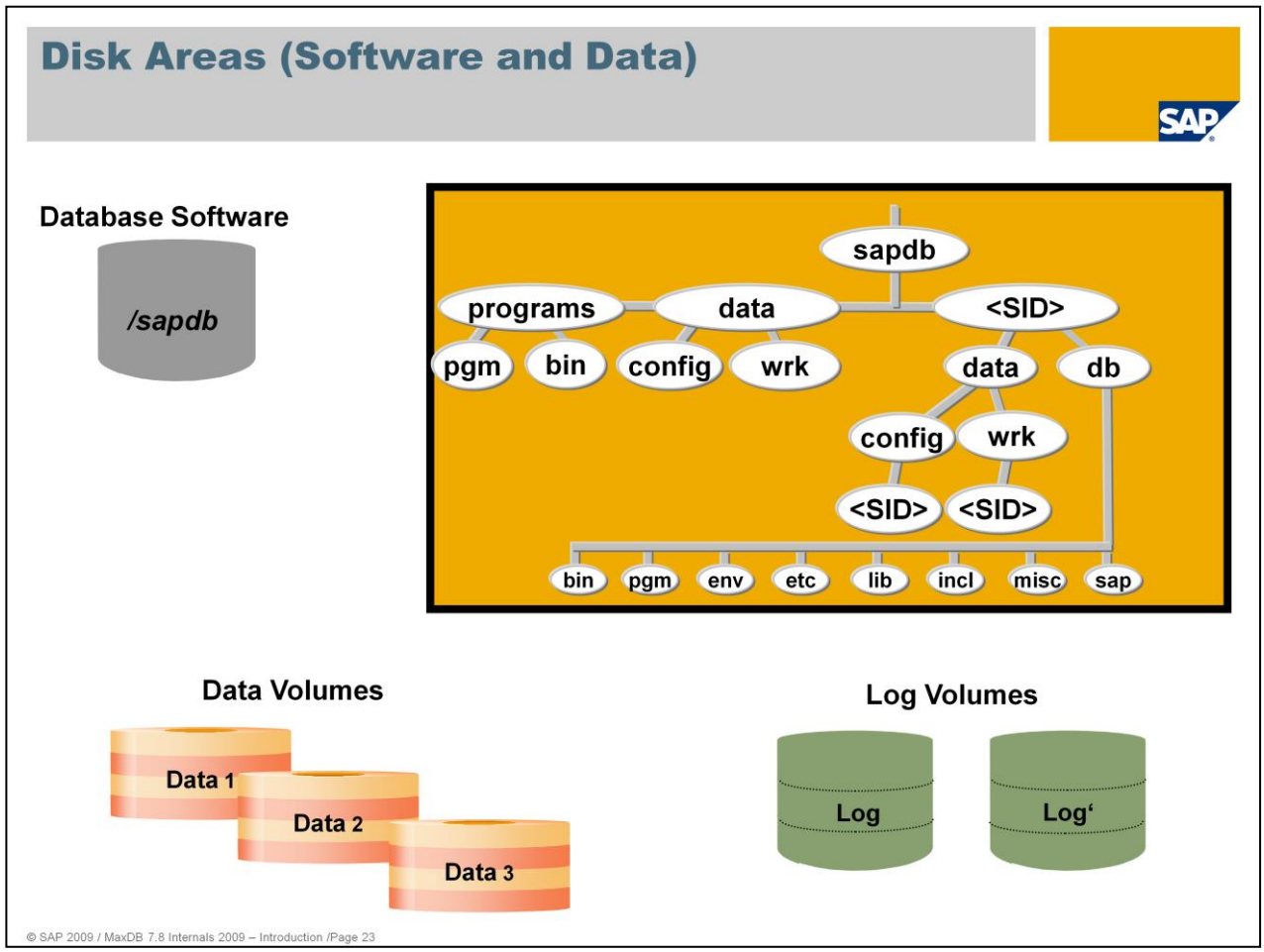


Accesses to caches can be synchronized over one or more regions. Depending on its size, the data cache is comprised of 8 to 64 segments of the same size, each of which is protected by exactly one region.

If a task or thread accesses a critical section, the region locks this session for all other tasks or processes.

Other main memory structures are also managed via synchronization mechanisms provided by the database.

Reader-writer locks are used to synchronize the shared SQL cache. Reader-writer locks are used in version 7.5 and up. In contrast to regions, reader-writer locks make it possible to distinguish between shared and exclusive locks.



A physical disk or part of a physical disk, respectively, is denoted with the item „volume“.

A database instance is installed on **three disk areas**:

- Data volumes
- Log volumes and
- Database software.

User data (tables, indexes, etc.), the SQL catalog and converter pages are stored in the **data volumes**. Data belonging to one table is evenly distributed on all data volumes by the use of an internal striping.

Changes on data are stored in form of redo log entries within the **log volumes**. This makes sure that in case of a required recovery all changes that are not contained in the recent complete data backup can be reloaded.

To guarantee secure database operation make sure to mirror the log volumes (set database parameter UseMirroredLog=YES). If the log volumes are not mirrored by the database itself, the disks have to be mirrored physically or by the operation system.

Redo log entries only contain changes of transactions, i.e. the after images. Undo log entries are administered separate from redo log entries in the data area.

Database software delivered by SAP comprises executable programs, source texts and utilities allowing to generate database processes and to work with the database instance. The software is installed within the file system in a defined directory with sub directories. Log and diagnosis files generated during database operation are also stored here.

Thank you!