SAP® MaxDB™

Expert Session – Shadow Page Algorithm

IMS NW MaxDB / liveCache
September 2012

THE BEST-RUN BUSINESSES RUN SAP™

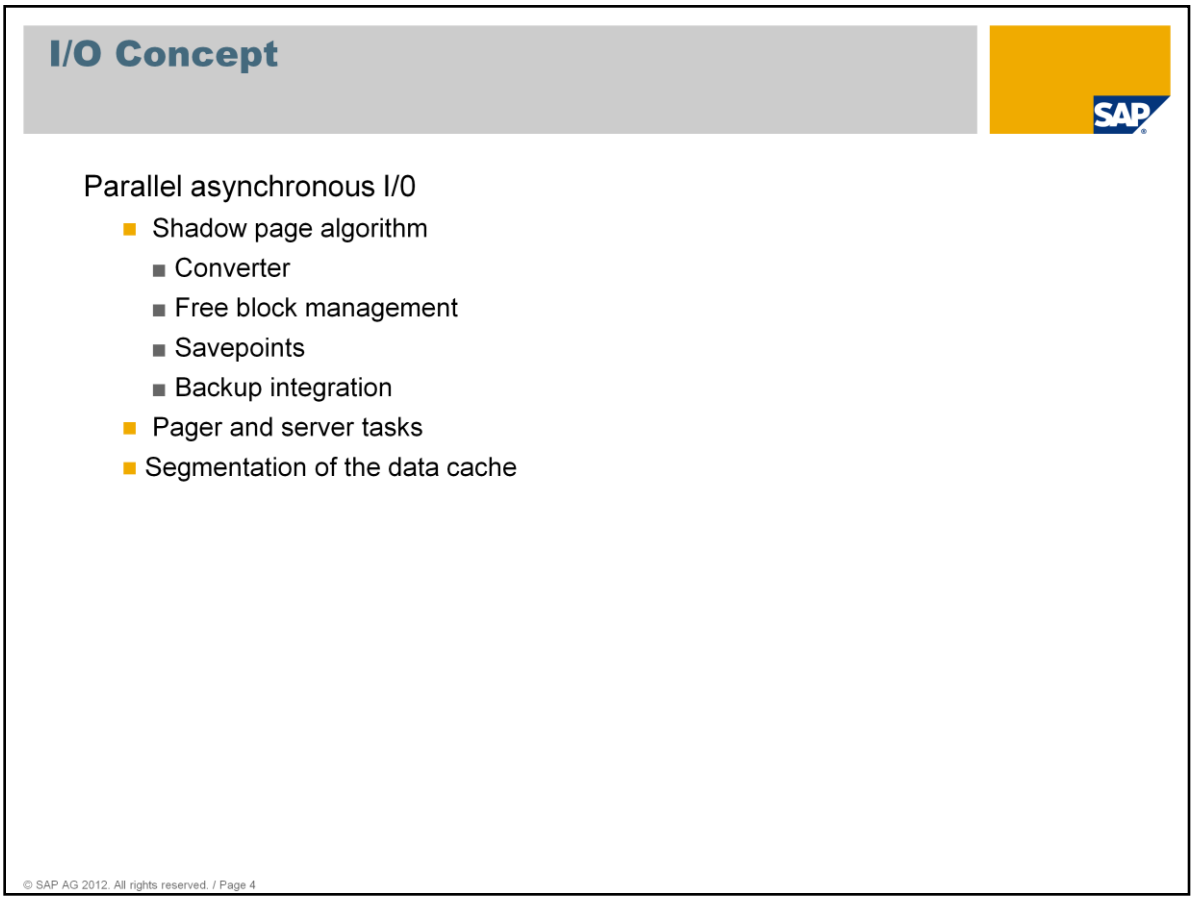
Expert Session

Shadow Page Algorithm

IMS NW MaxDB / liveCache
Heike Gursch
Christiane Hienger
September 26, 2012

Agenda

1. Parallel asynchronous I/O
2. Shadow Page Algorithm
3. Parallel Backup
4. Pager and Server Tasks
5. Segmentation of the Data Cache
6. Snapshots



The I/O concept of the database works according to the shadow storage administration principle. The core elements are: optimized support of symmetrical multi-processor systems; the transfer of as many I/Os as possible to asynchronous execution; and highly-optimized data backup performance suited to the dimensions of modern databases.

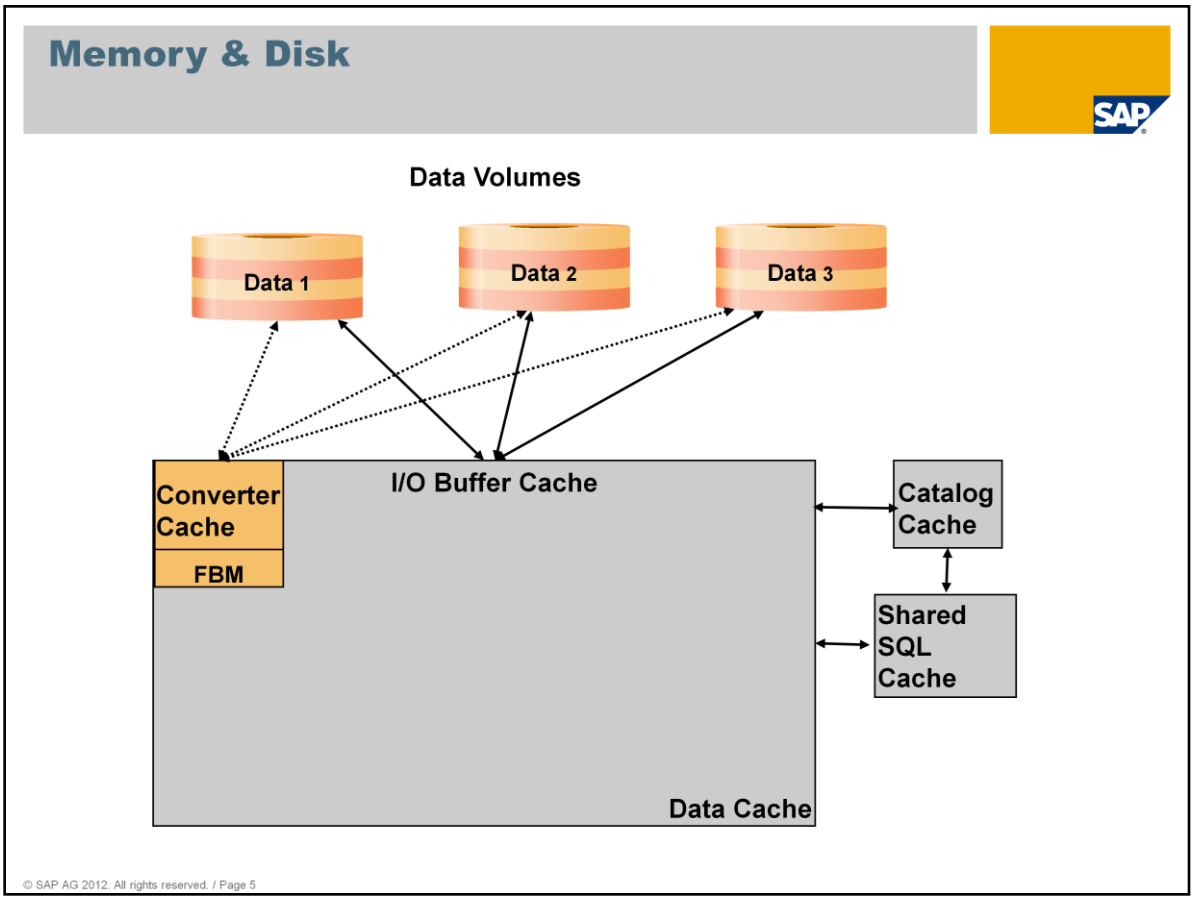
A user task should not be forced to wait for I/O processes to come to an end. All change operations are executed in the main memory. The I/O subsystem must ensure that the system always retains its ability to restart at the point of termination.

The shadow storage administration distinguishes between originals and copies. When the system is restarted after termination, the valid states of the data pages are automatically recognized. The concept is based on savepoint cycles, which are closed by a savepoint. A completed cycle is specified by the version number of its savepoint. This number is referred to as the 'savepoint version' or 'converter version'.

The different versions of the data pages that arise as a result of the savepoint cycles are administered in the converter. Here the originals and copies of the logical data pages are assigned physical blocks. Thus the location at which a logical data page is stored can change from savepoint cycle to savepoint cycle.

Another structure is employed for the administration of the data volumes (FBM: Free Block Manager). As the logical data pages no longer have a definite location in the memory, the FBM administers the states of the physical blocks. These structures enable optimal performance for data backup as well.

The data cache was optimized with regard to SMP support by the use of pager and server tasks that work in parallel.

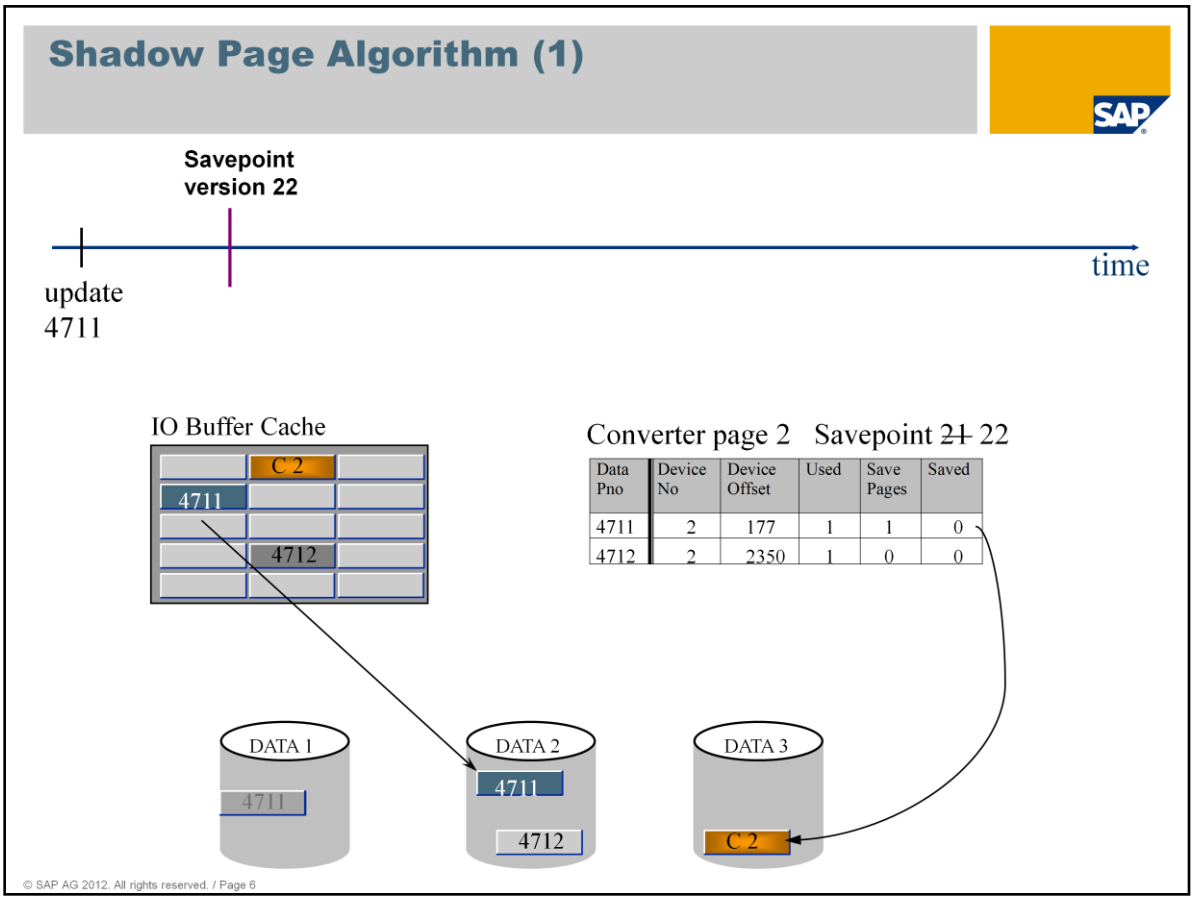


The introduction of shadow storage administration brought the launch of the converter. From Version 7.4, the converter is no longer stored in the system devspace but distributed across the data volumes. Every I/O access to a data page retrieves its information from the converter. For that reason, the complete contents of the converter are kept in the main memory (converter cache).

Free block management (FBM) for free blocks in the data volumes is kept in the main memory and is reconstituted with each restart using information from the converter.

Other important caches include the data cache, which contains the most recently used data pages; the catalog cache, which contains dictionary information on the used object and, if shared SQL is not active, buffers execution plans; and the shared SQL cache, which contains executed SQL statements, execution plans and monitor data.

The next few pages will present the concept of shadow memory using examples.



In our example, the database instance is in the online operational state. The last completed savepoint cycle has the number 21.

The IO buffer contains data and converter pages.

In the converter pages, we find the positions of the data pages in the data volumes.

Data page 4711 will now be changed in the cache in savepoint cycle 22. Initially, this change does not lead to a write operation to the data volumes.

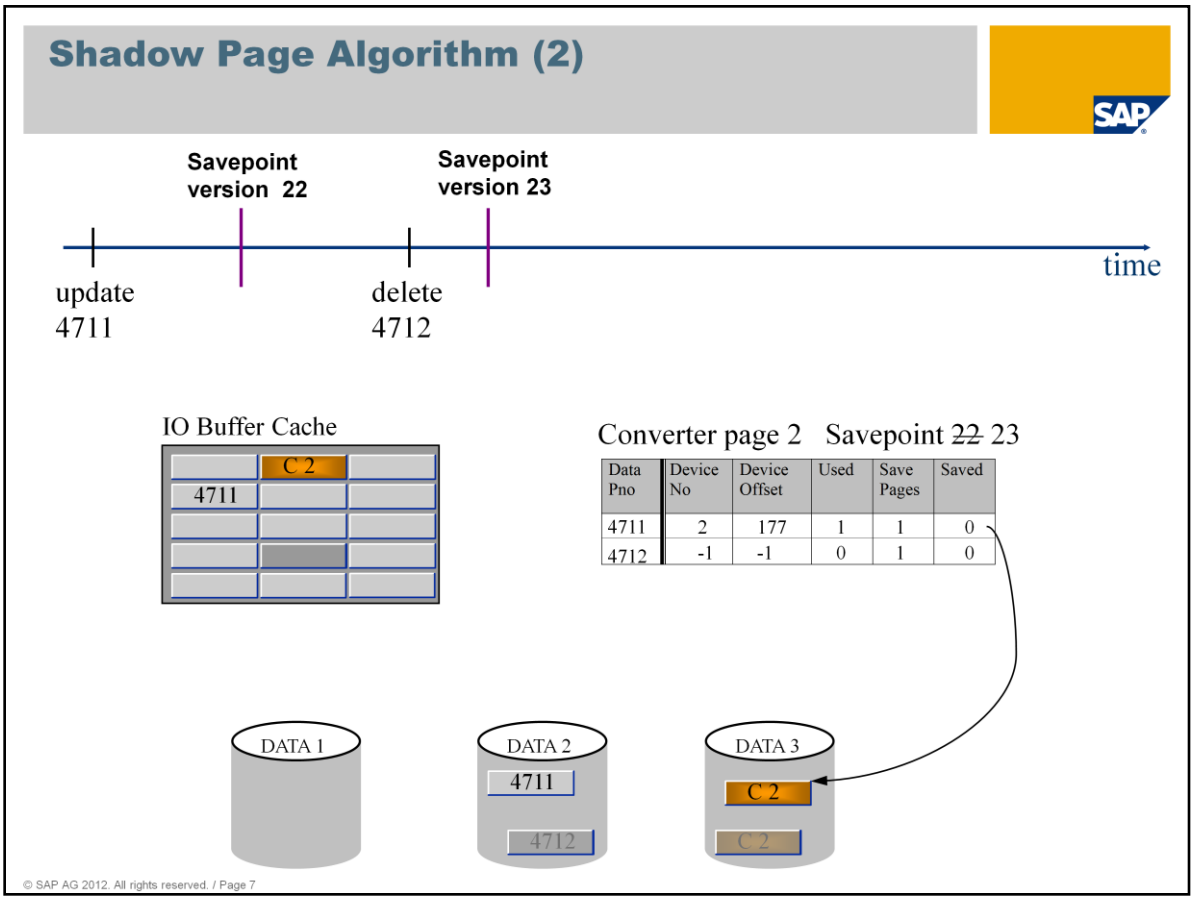
The savepoint cycle is closed with a savepoint. First, all changed pages are written to the data volumes. For data page 4711, the FBM determines position 177 in data volume 2. Data page 4711 is written to the determined position in the data volume and the position data is written to the converter page. The data page is not written at the position that was valid for savepoint 21.

After all changed data pages have been written to the data volumes, the savepoint versions in the changed converter pages are set to the number of the savepoint cycle and the converter pages written to the data volumes.

Each converter page stores the positions of up to 1861 data pages.

A converter page contains the following information about each data page:

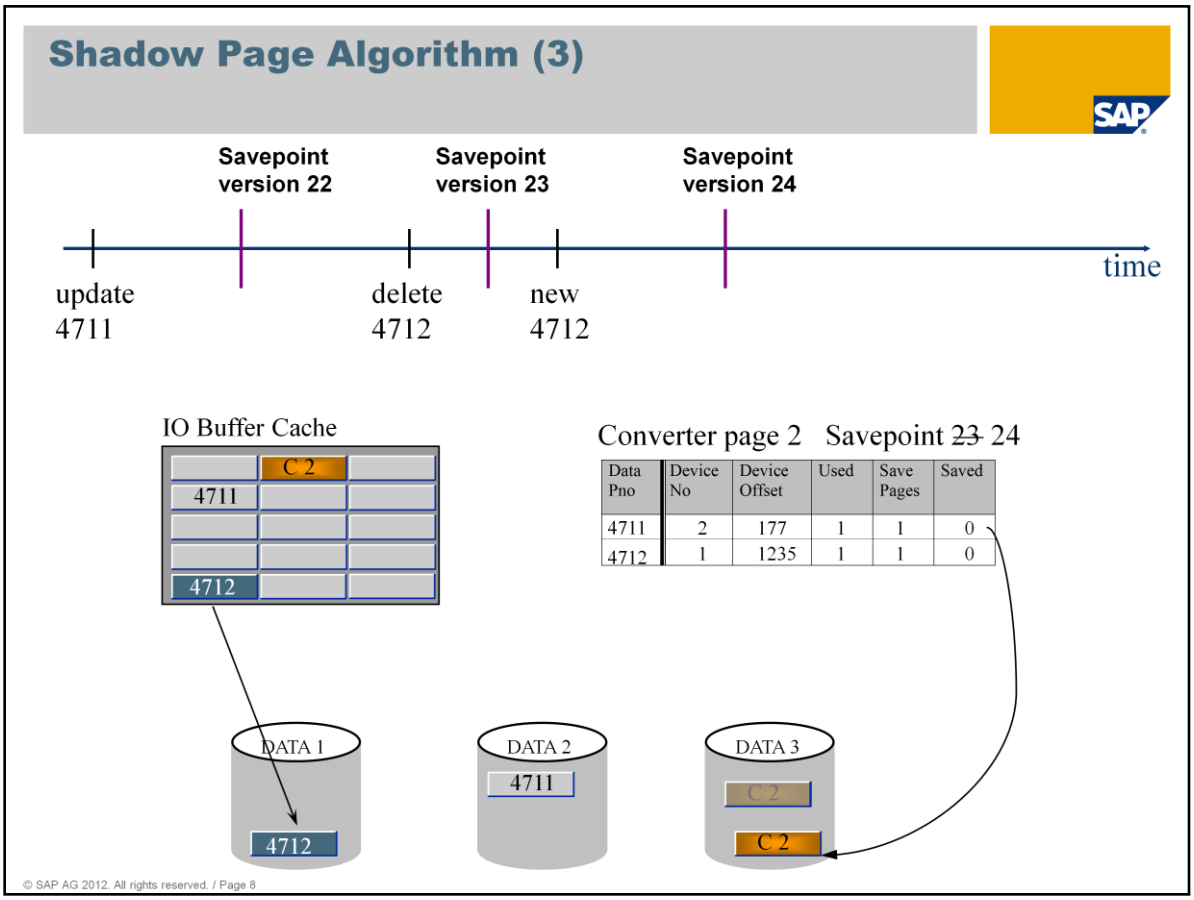
- the number of the data volume,
- the position within the data volume,
- a flag indicating whether the data page is in use (required to indicate used data pages that have not yet been written to a data volume),
- a flag indicating whether the data page is relevant for incremental backups,
- a flag indicating whether the data page was already backed up in the incremental backup.
  This way the information on the data page to be saved is not lost in the case of an aborted incremental backup.



In savepoint cycle 23, a user deletes all entries in data page 4712. Upon release, the page is immediately marked as free in the converter.

Only the converter page is re-written at the time of the savepoint. Data page 4712 is not re-written. The converter page is written to a new position in a data volume.

After the completion of the savepoint, the former position of data page 4712 in the data volumes is marked as free in the FBM.

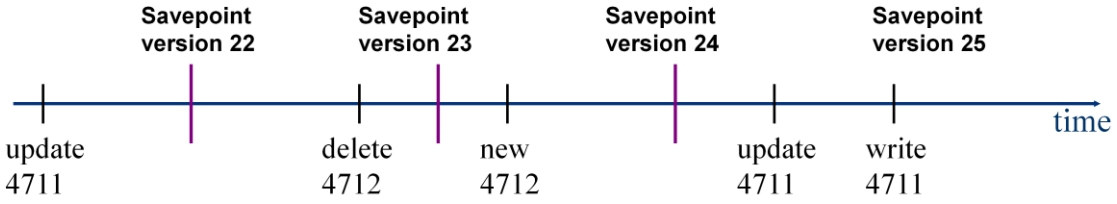


In savepoint 24, the database uses page 4712 for new data. The data page could already have been reassigned in savepoint cycle 23.

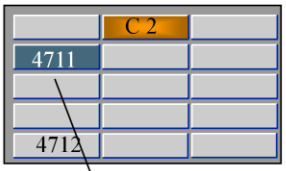
The new data page is marked as used in the converter. But no position for a data volume has been entered yet.

Initially, changes take place only in the cache. Upon completion of the savepoint cycle, the data page is written and its position entered in the converter. The converter page is written to a new position in a data volume.

Shadow Page Algorithm (4)

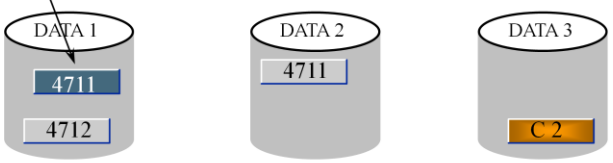


IO Buffer Cache



Converter page 2 Savepoint 24

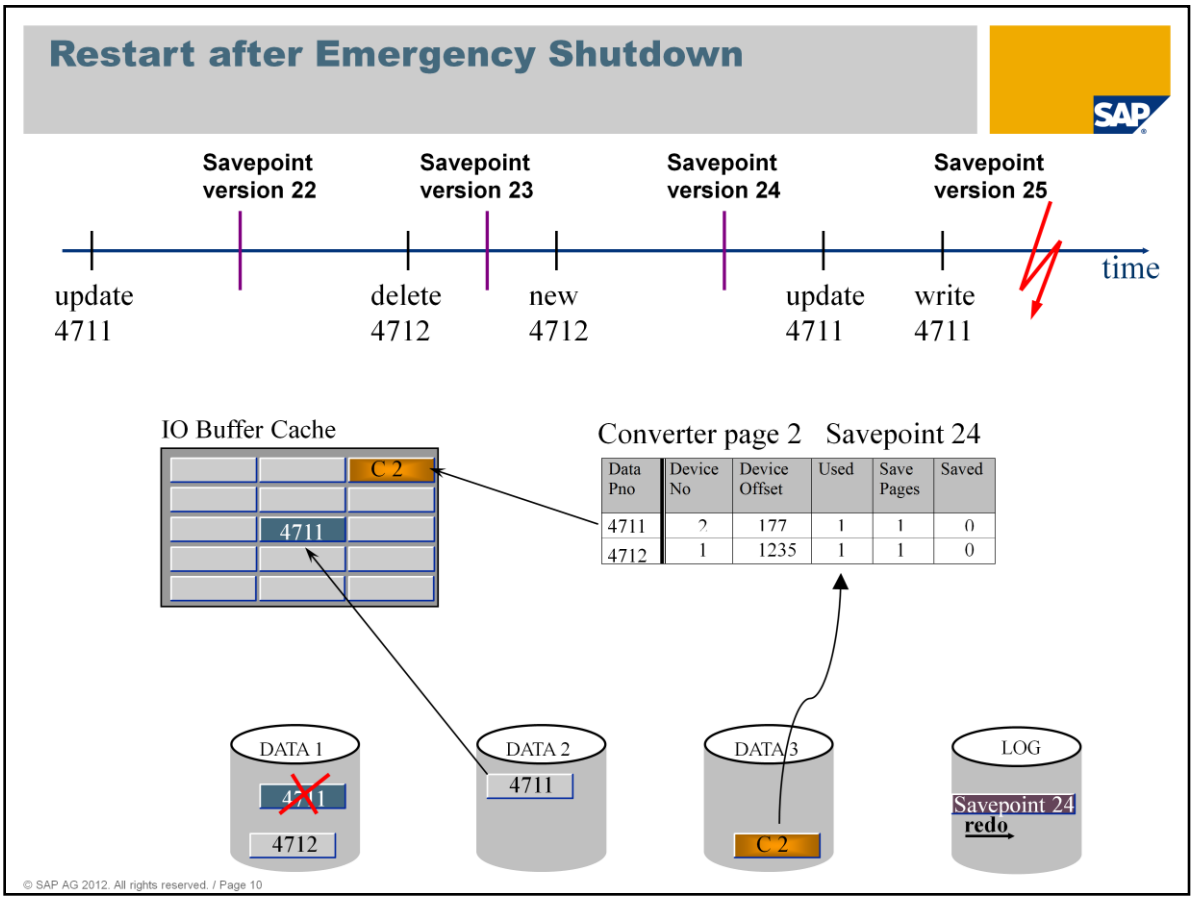
Data Pno	Device No	Device Offset	Used	Save Pages	Saved
4711	1	438	1	1	0
4712	1	1235	1	1	0



Data page 4711 is changed again in savepoint 25.

A data page can be written even before the completion of the savepoint cycle. The savepoint itself generally takes a few seconds as several data pages are written.

Data page 4711 is written and the new position entered in the converter page. The savepoint is not yet complete.



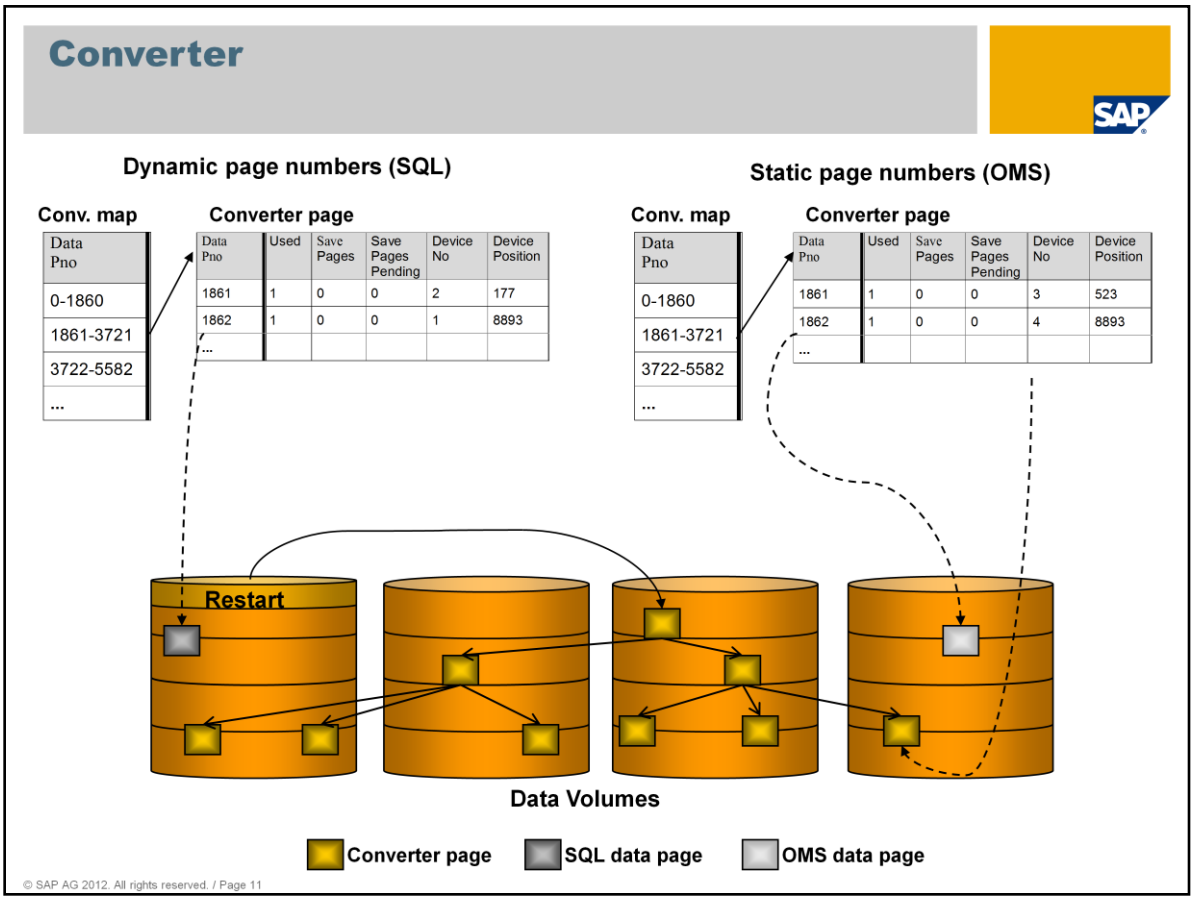
Before savepoint 25 is complete, an emergency shutdown occurs. The next restart reads the valid converter information for savepoint 24 from the data volumes.

Free block management sets up when the converter is read. The position occupied by data page 4711 through the write operation in savepoint 25 is marked as free.

Data page 4711 can be read via the old valid position.

The starting point in the restart page for savepoint 24 is known. The redo log entries are redone starting from this point. The change to data page 4711 during savepoint cycle 25 will be redone if the transaction that performed the change was completed with Commit before the emergency shutdown took place.

The writing of the restart page to position 2 in data volume one completes a savepoint cycle.



In versions < 7.4, the converter was implemented as a static array. The ability to enlarge an instance in online mode was limited. The limit was set with the parameter MAXDATAPAGES. This parameter defined the size of the converter. So the converter was generally larger than necessary.

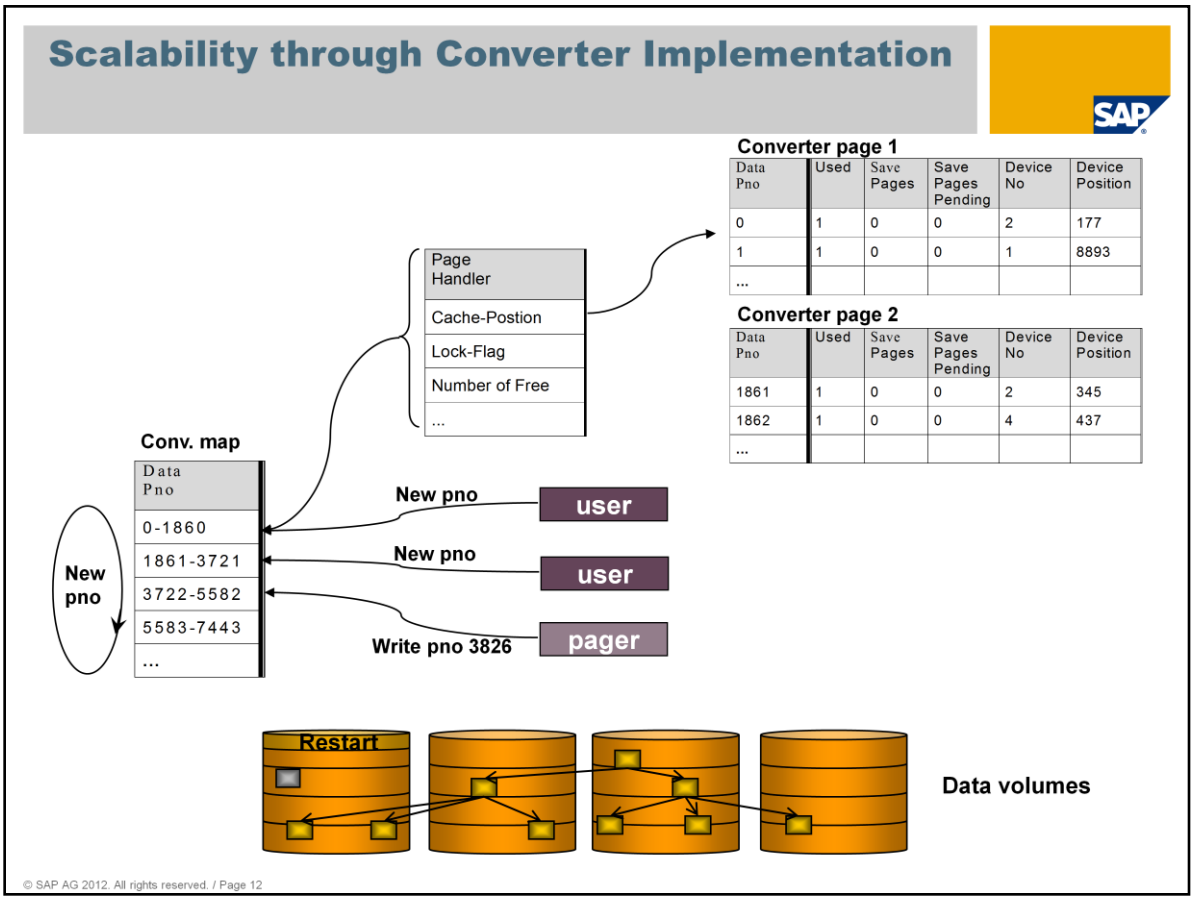
From Version 7.4, the converter can grow and shrink dynamically. The converter pages are distributed across all data volumes. Upon restart, read access to the converter pages is done via a tree structure. The tree has 3 levels: a root level, an index level and a leaf level. Upon restart, the database finds the root page of the converter via the restart page which is usuallyat the beginning of the first data volume. It contains the positions of the index pages. For their part, the index pages contain positions of the leaf pages. The leaf pages are not necessarily sorted. From Version 7.4.3., the restart reads the converter pages in parallel.

The tree as a whole contains pages for 3 converters, one each for

- Static page numbers for OMS data (live cache)
- Dynamic page numbers for permanent data that is not OMS data
- Page numbers for temporary pages

Static and dynamic data pages are handled separately for the following reasons:

- Changes to relational data are logged in the log without position information for the data records.
- Changes to OMS data are done on the basis of an object ID. The object ID contains the number of the data page that contains the object.
- Without this separation, under some circumstances it might not be possible to restore objects in a log recovery due to the corresponding data page number for relational having been assigned.



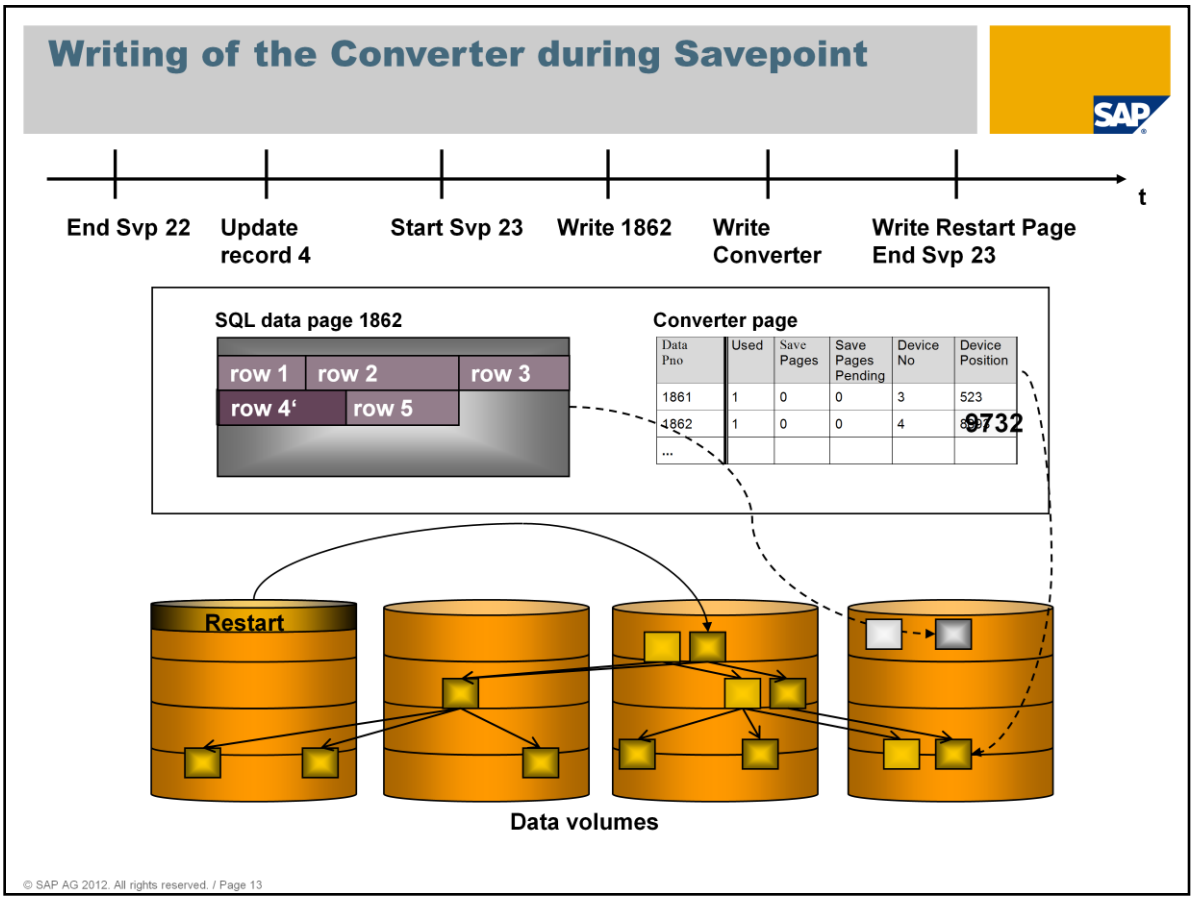
A converter page has a size of 8192 bytes. It contains references for 1861 data pages. A database with 500 GB of used data requires a converter of roughly 278 MB.

When the converter is read during the restart, a converter map is generated in the cache. The converter pages in the cache do not have fixed positions. The position of a converter page is determined via the converter map each time it is accessed.

For each converter page, the converter map contains the cache position and the number of free entries as well as administration information for the savepoint.

Accesses to converter pages are synchronized through the use of converter regions. Each entry in the converter map is assigned a region. You can set the number of regions with the parameter ConverterStripes (until 7.6 CONVERTER_REGIONS). This allows several users to access and change converter pages at the same time.

From 7.4, free page numbers are no longer determined by way of a PNO pool. They are determined directly from the converter. Free entries in the converter pages are concatenated via main memory structures. So several users working at the same time can use new pages very quickly.



During the savepoint the database kernel writes all changed data pages to the data volumes. It enters the new positions in the converter.

The kernel writes the changed converter pages in the last savepoint phase. The pages are not written to their original positions. Because the position of the converter pages changes, the corresponding converter index and converter root are also changed. These pages are also written to new positions.

When all changed data and converter pages have been written, the position of the converter root is entered in the restart page. The restart page for the old savepoint is overwritten in the data volume.

The savepoint is complete when the restart page is written. This ensures that the kernel can always restart from the last completed savepoint.

Free Block Management

- FBM: Free Block Manager
- Bit list per data volume
  - Used capacity states:
    - free
    - occupied
    - free after savepoint
  - Backup states
    - free
    - backup
- exists in the memory
- is built during start of the DB

Volume 1:

device offset	page state	backup state
434	occupied	free
1235	occupied	free

Volume 2:

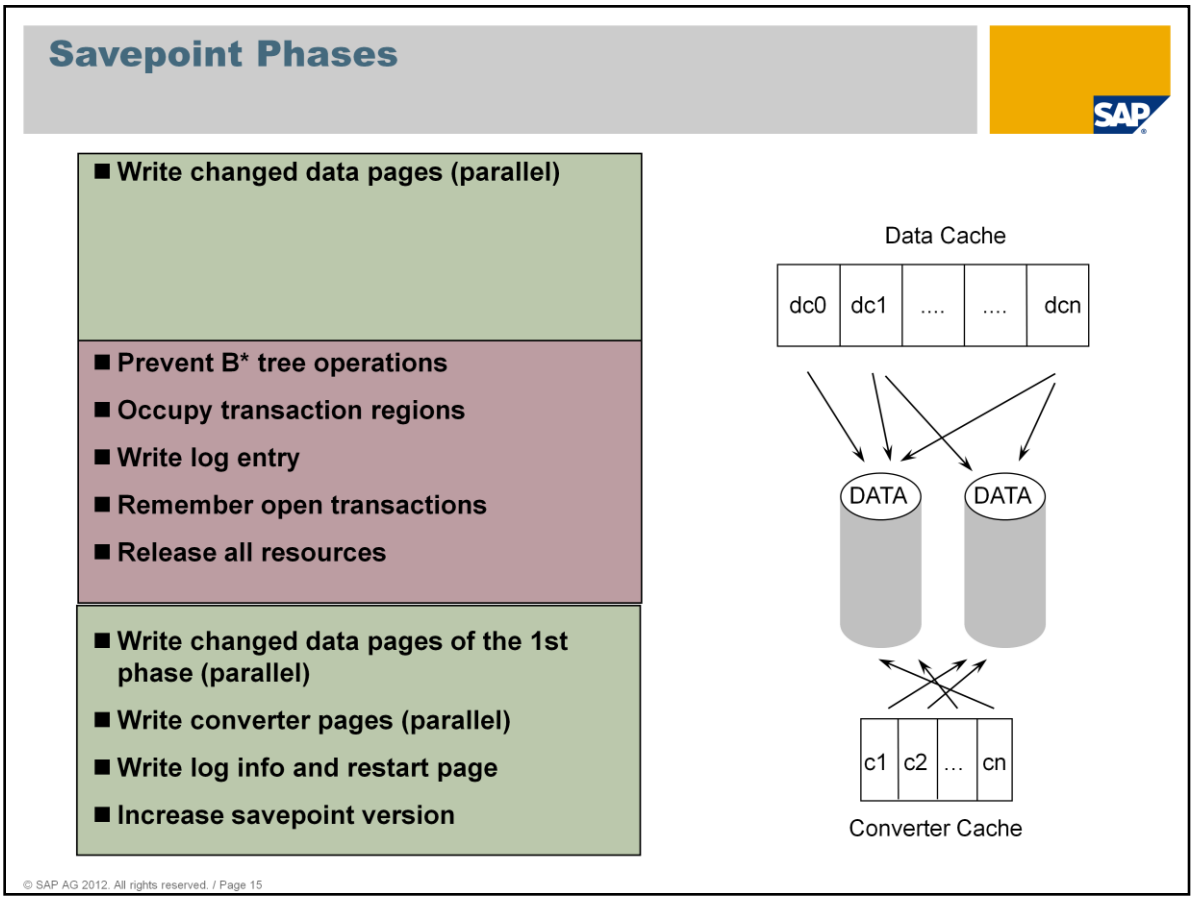
device offset	page state	backup state
177	free after sp	backup

The Free Block Manager, which only exists in the memory, administers all data devices using a bit list for each device. This includes the used capacity status and the backup status. The possible statuses are:

- **Free:**
  The block is free and can be allocated.
- **Occupied:**
  The block is occupied.
- **Free after savepoint:**
  The block can be released after the current savepoint has been successfully completed.
- **Backup:**
  The block belongs to a backup that is in process. When the block has been backed up, that is, written to the backup medium, the status is reset.

If a block has been selected for a backup in process, the used capacity status of a block can change. A block can only be reallocated when the used capacity and backup statuses are both "free."

From version 7.4, the Free Block Manager is part of converter management.



The savepoint is a core function of the I/O concept. The illustration shows what happens during a savepoint.

The savepoint writes the data from the data cache and the converter cache to the corresponding data volumes. Due to the size of the two caches, this cannot be carried out as a synchronous action; the system would be blocked for too long. There has to be a short phase in which the caches can be securely flushed, but this must be kept to a minimum.

The standard is for savepoints to occur at intervals of 10 minutes. To minimize the amount of data to be flushed in the protected section (marked red), the savepoint begins by flushing the data cache parallel to operation. The data cache is processed by several pagers simultaneously. The largest share of pages is flushed in this phase.

In the second phase, a flag is set which prohibits clearing operations on B* trees. It is also prohibited to open new transactions during this phase. All pages that were changed in the course of the first phase are marked as savepoint relevant. An open trans file is created for open transactions.

In the last phase, all pages that were marked during the second phase are flushed. The flags are reset. First, all changed pages are written to the data volumes. The savepoint is complete when the restart page is written. Afterwards the savepoint version (number) is updated.

The protected phase of the savepoint is generally quite short and goes unnoticed by the end user.

Backup Phases

Savepoint before the backup starts

- The data belonging to one savepoint contains all necessary undo information of open transactions. Thus the database is transaction consistent. It can be set to ONLINE mode with a restart without any log.

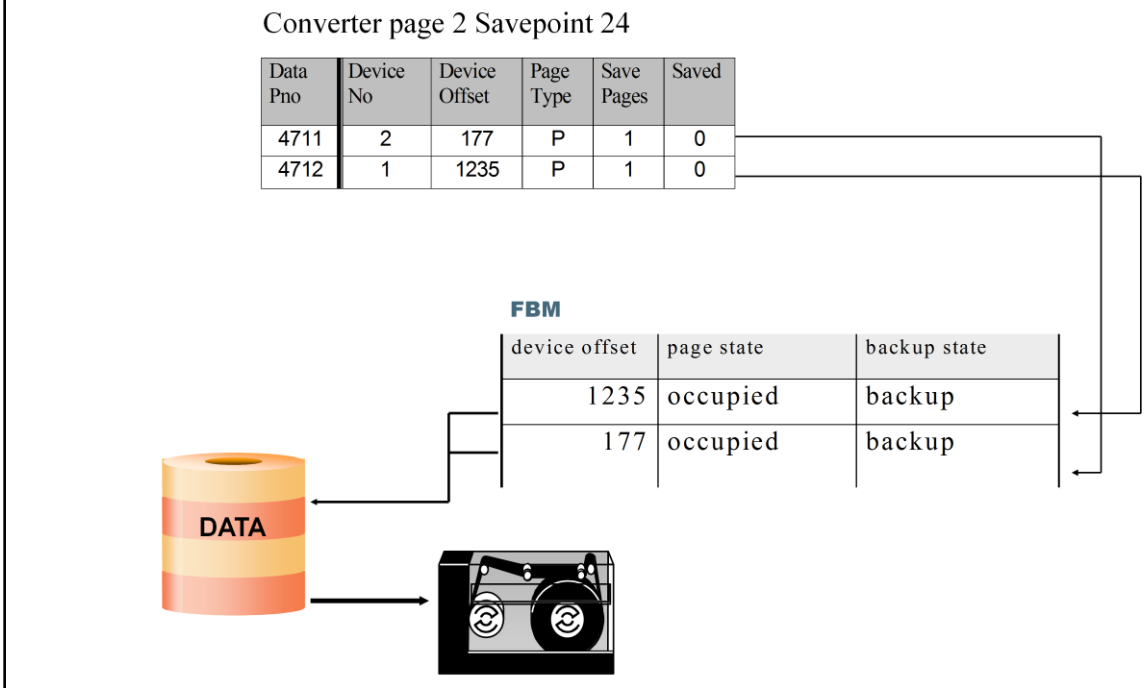
The savepoint looks for data pages relevant for backup in the converter and sets the backup flag in the FBM.

Parallel backup through server tasks along the FBM

Data backups are carried out with a block size of 8 x 8 KB and can be parallelized.

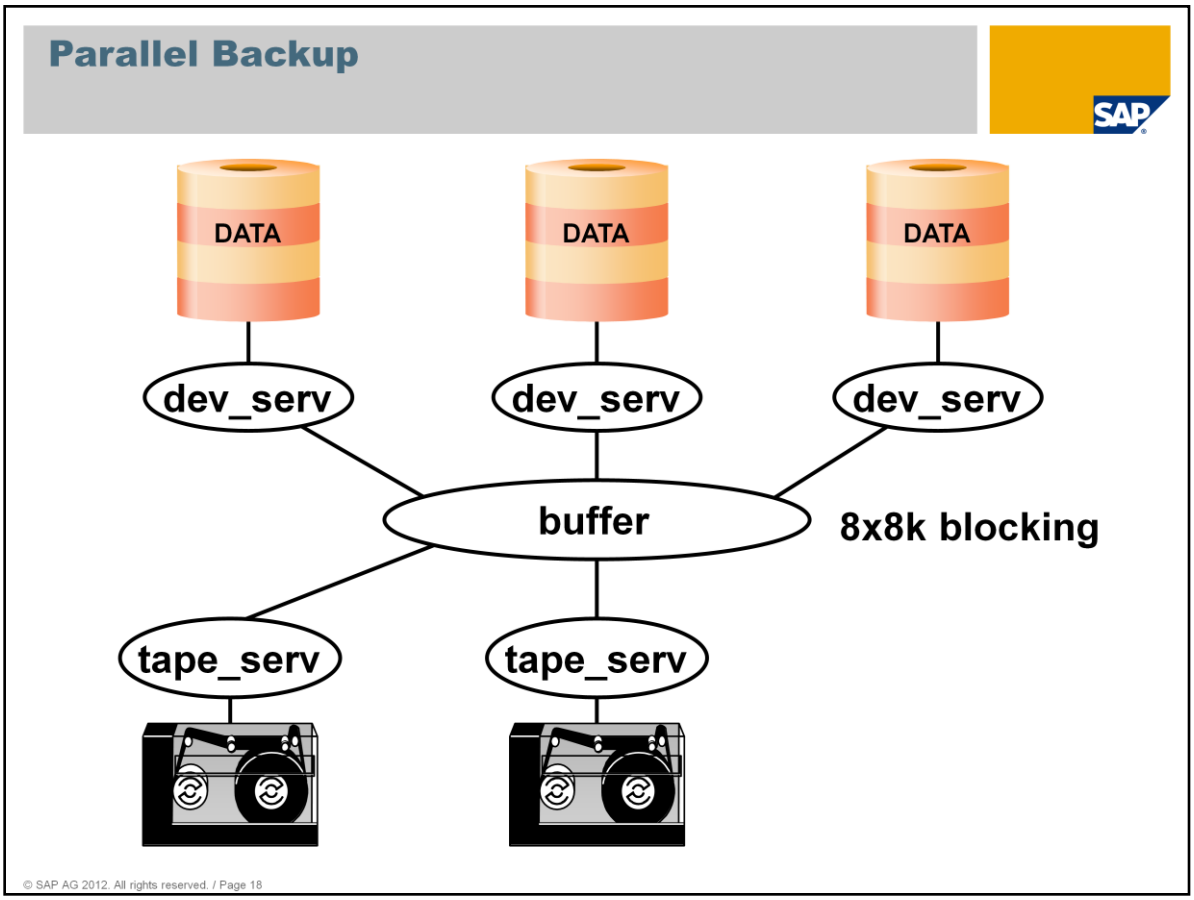
Data backups start with a savepoint. A backup includes the data existing at the time of the savepoint. Subsequent changes are not included in the backup. The database can write further savepoints while the backup is in process.

Backup along the FBM



The savepoint that is executed at the start of the data backup determines the savepoint version of the backup. Through the converter, the data blocks on the data devices that are valid for this savepoint are determined for all data pages and the backup status is set in the FBM.

The data backup uses the bit lists of the FBM. There the data blocks are combined into 64 KB I/O units. This procedure executes one task per data volume. The backup status is reset following each block backup. Data blocks that have been freed for use can only be reallocated when the backup status has been reset.



This illustration depicts a data transfer from the data volumes to the backup media. Each volume has a task that puts the 64 KB units into a buffer. One task per backup device reads the blocks from the buffer and stores it on the backup medium.

The limits of this process are posed either by the access speed of the data volumes, the writing performance of the backup devices or the transport layer (e.g. network) between the database server and the backup devices. As long as these limits are not reached, the process scales with any other backup device in parallel operation.

SAP note 1676903 “SAP MaxDB: Runtime analysis of data backup” provides detailed information about how to investigate the performance of a backup.

Server Tasks and Pager Tasks

Server:

- Backup, Create Index, CHECK DATA

Pager:

- Read the converter during the restart of the instance
- Savepoints
- write_ahead of changed data pages from data cache to the data volumes

Are indicated in the Taskcluster with sv and pg (former dw – datawriter).

In the task display (x_cons, CCMS) with server and pager

The auxiliary tasks have reduced stack requirements and should run in one or more dedicated UKTs.

Server tasks or pager tasks have a reduced stack requirement as they do not have to carry out syntax analysis or related activities, but only process pre-translated requests.

To prevent the server tasks from negatively influencing the user tasks due to their high throughput, they are created in their own thread (UKT) in the standard.

During times in which a system is working with a moderate to low I/O load, pager tasks perform so-called write-ahead operations. This means that data pages that have been changed in the data cache are written to the data volumes ahead of time, i.e. before a savepoint or displacement. This in turn means a reduction of the burden on the first phase of the coming savepoint as there is substantially less I/O to be handled. In general, a favorable setting of the pager tasks can ensure a consistently low and largely asynchronous I/O load.

More information about server task configuration -> see note 1672994.

Data Cache - Segmentation

The data cache is divided into segments of the main memory (regions).

Number of regions is adjustable (8 – 1024, the default value depends on the cache size).

More parallelism by multiplication of regions (MaxDB internal synchronization mechanism)

Pages are definitely assigned to a segment by a hash function.

During savepoints the segments can be written through in parallel by pager tasks.

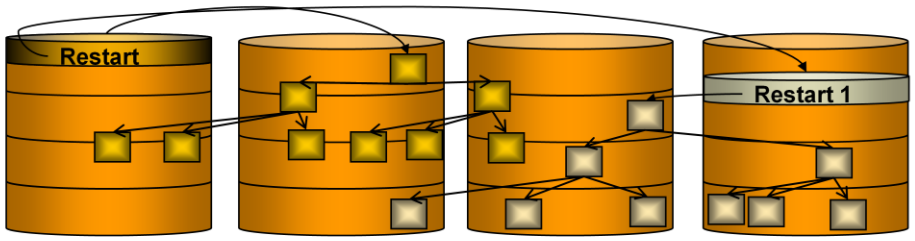
The data cache was divided into segments to enable better SMP support and to accelerate savepoints. Each segment is secured by its own region. The data pages are uniquely assigned to a segment, that is, a data page with the page number 4711 (pno), for example, is always administered in the second cache segment.

As of version 7.6.03 the number of possible segments has increased from 64 to 1024.

Snapshot

Freezing the data area

- Create Snapshot
- Restore Snapshot (ADMIN)
- Drop Snapshot
- Typical usage:
  - Very fast point-in-time recovery (e.g. for SAP upgrades, installation of support packages)
  - Restore of training systems to a defined state



From version 7.5, you can freeze the data area of a database instance using a snapshot.

In versions 7.5 and 7.6 a snapshot is generated in the ADMIN state. As of 7.7 it is also possible to create it in ONLINE mode. Later you can reset the data to its state at the time of the snapshot and/or delete the snapshot.

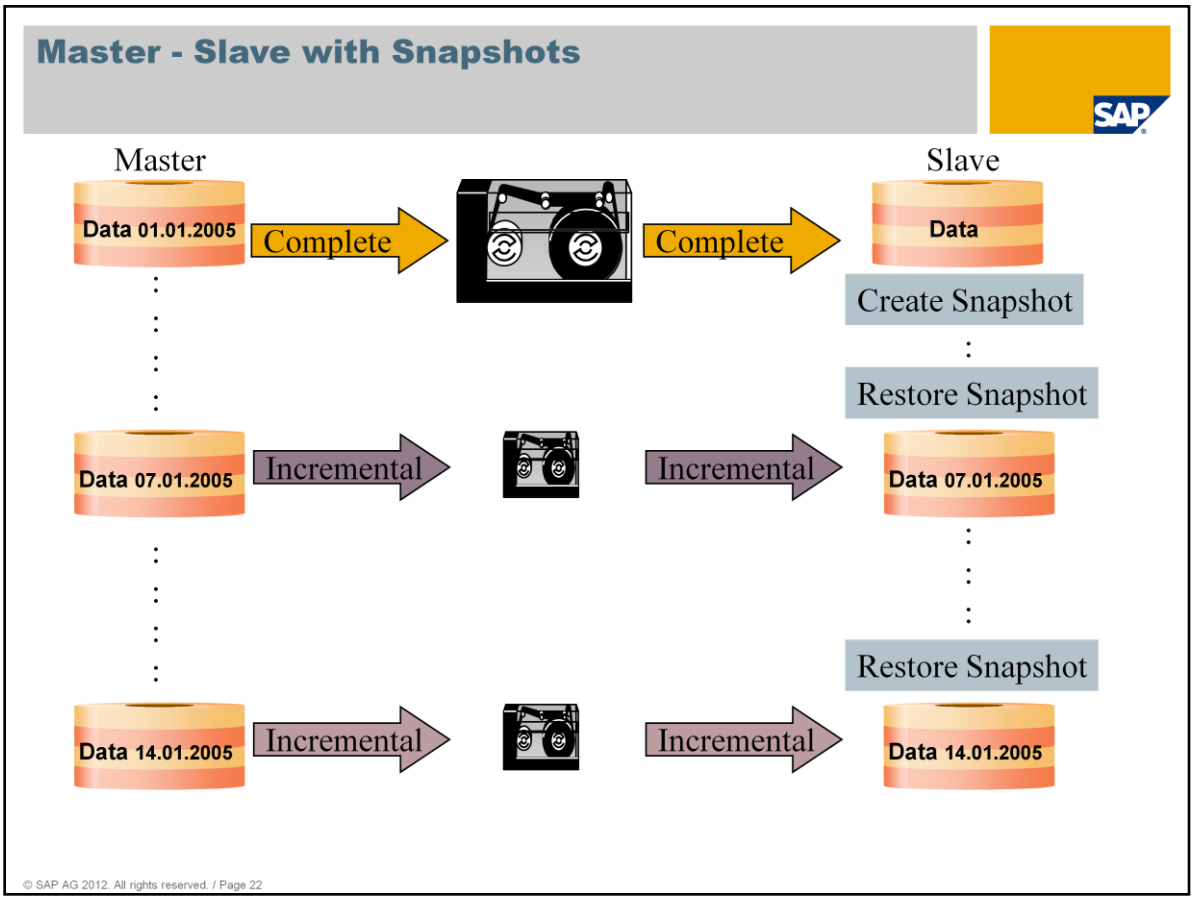
With the CREATE_SNAPSHOT command, the database kernel copies the restart page (which is usually located on the second block of the first data volume) to another position. The complete converter is also copied. The original restart record contains a reference to the restart record that corresponds to the snapshot.

With the command RESTORE_SNAPSHOT, the current converter is deleted. All blocks that are no longer needed are marked as free in the FBM. The log is formatted and thus the state HISTLOST occurs. At the next restart, the instance works with the data as they were at the time of the CREATE_SNAPSHOT.

The statement DROP_SNAPSHOT deletes the restart record and the corresponding converter that is relevant for the snapshot. The FBM marks all blocks that are no longer needed as free.

Up to 7.6 MaxDB supports only a single snapshot, as of 7.7 several snapshots can be generated. Operating the instance with one or several snapshot(s) uses more of the capacity of the data area.

For more information see SAP FAQ note 1423732 SAP MaxDB Snapshots



MaxDB offers the possibility of using snapshots to synchronize a master and one or more slave instances.

Create the slave instance as a homogeneous system copy using Backup/Restore. Before the first restart of the slave instance, generate a snapshot.

To transfer changes in the master instance to the slave instance, reset the slave instance to the snapshot. Then import an incremental backup from the master instance. You can reset the slave instance to the snapshot as often as you like and import incremental backups from the master instance.

This procedure works until a complete backup is created in the master instance. Then new incremental backups no longer match the snapshot in the slave instance. To synchronize it with the master, you can import a complete data backup into the slave instance.

Questions and Answers

Thank You!
Bye, Bye – And Remember Next Sessions

2013	Next sessions will be provided in 2013.
	Did you miss a session? Have a look under http://maxdb.sap.com/training

