Vorlesung Implementierung von Datenbanksystemen

14. Beispiel Oracle Database

Prof. Dr. Klaus Meyer-Wegener

Wintersemester 2019/20

Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

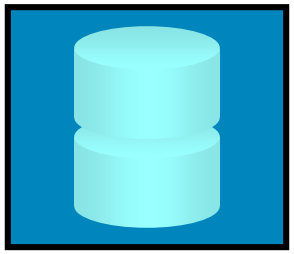
Überblick

- **Ausschnitte aus den Oracle-Schulungsunterlagen**
  - Oracle Database 11g: Administration Workshop I,
  - Oracle Database 11g: Administration Workshop II und
  - Oracle9i Database Administration Fundamentals I

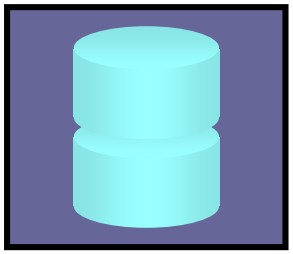
- **Storage Architecture**
- **Index Structures**
- **Server Architecture**
  - Memory Structure
  - Process Structure
- **Interacting with the Database**
- **Redo, Undo, Recovery**

(von mir umgeordnet und die PowerPoint-Notizen auf Deutsch in die Folien integriert)

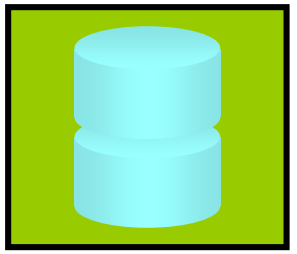
Database Storage Architecture



Control files

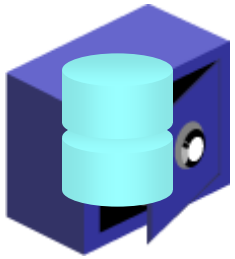


Data files

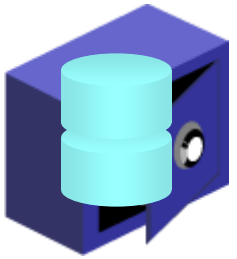


Online redo log files

Parameter file

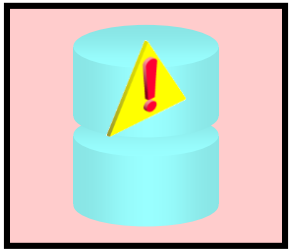


Backup files



Archived redo log files

Password file



Alert log and trace files

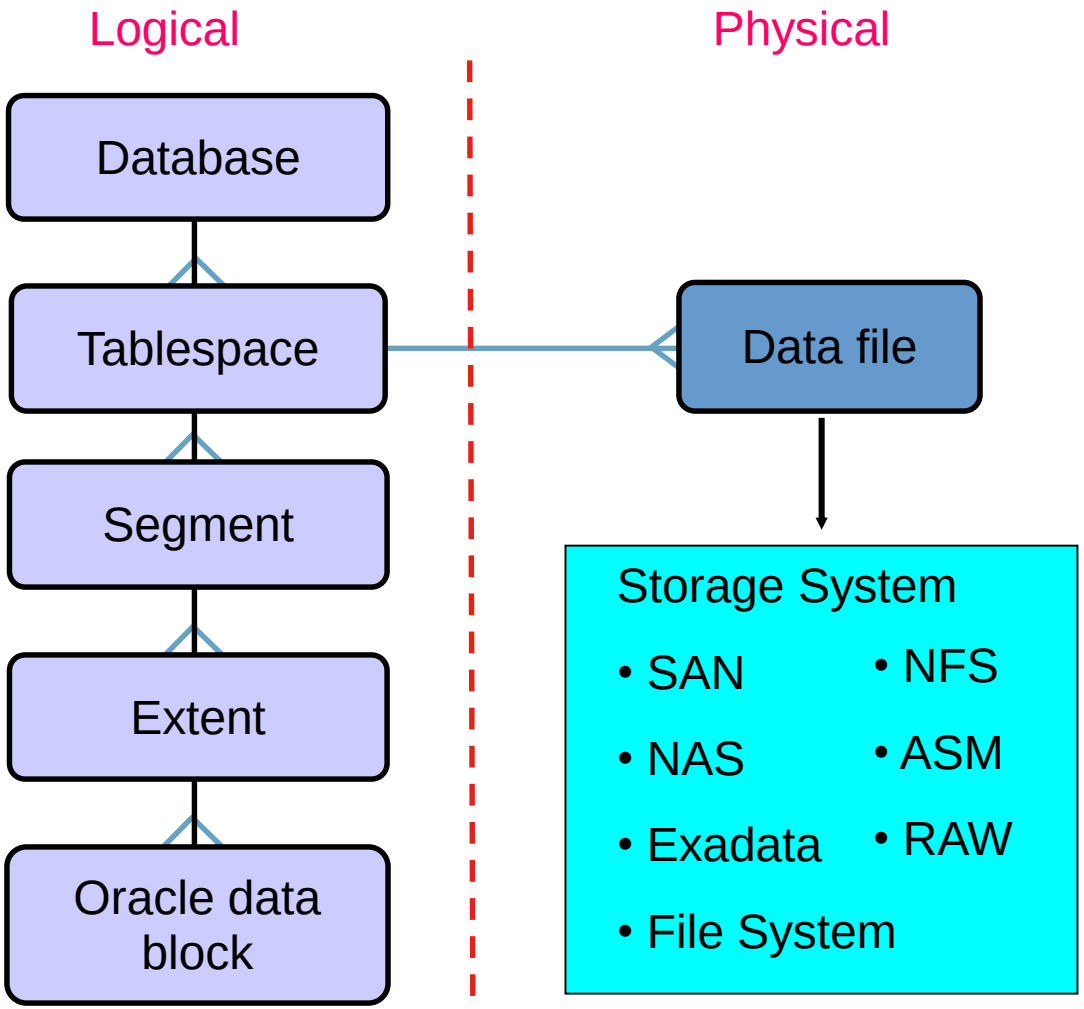
Dateien

- **Control files:**
  - Zentraler Einstiegspunkt – unbedingt erforderlich
  - Metadaten für Backup
- **Data files:**
  - Die eigentlichen Daten, Metadaten, Schema (Data Dictionary)
- **Online redo log files:**
  - Temporäre Protokolldatei (auch für Undo)
- **Parameter file:**
  - Konfiguration der DB-Instanz
- **Password file:**
  - Erlaubt Administratoren die Fernwartung der DB-Instanz
- **Backup files:**
  - Archivkopie der Datenbank
- **Archived redo log files:**
  - Inkrementelle Archivkopie-Dateien

Dateien (2)

- **Trace files:**
  - Wird von jedem Server- und Hintergrund-Prozess geschrieben
  - Aufzeichnung interner Fehler mit Zusatzinfo
  - Teils für den Administrator, teils für Oracle Support Service
- **Alert log file:**
  - Spezielle Trace-Einträge
  - Chronologisches Protokoll von Nachrichten und Fehlern
  - Sollte regelmäßig überprüft werden

Logical and Physical Database Structures



Logische und physische DB-Strukturen

- **Databases, Tablespaces und Data Files (Dateien)**
  - Datenbank aufgeteilt in zwei oder mehr Tablespaces
  - Eine Datei (oder mehrere) pro Tablespace
    für alle Daten aller Segmente im Tablespace
- **Tablespaces**
  - gruppieren zusammenhängende Strukturen
  - Z.B. alle Segmente einer bestimmten Anwendung,
    um die Administration zu vereinfachen
- **Data Blocks**
  - Seite
  - Kleinstes Granulat
  - Größe wird für jeden Tablespace bei seiner Erzeugung festgelegt
  - Einheit der Speicher-Allokation und -Freigabe

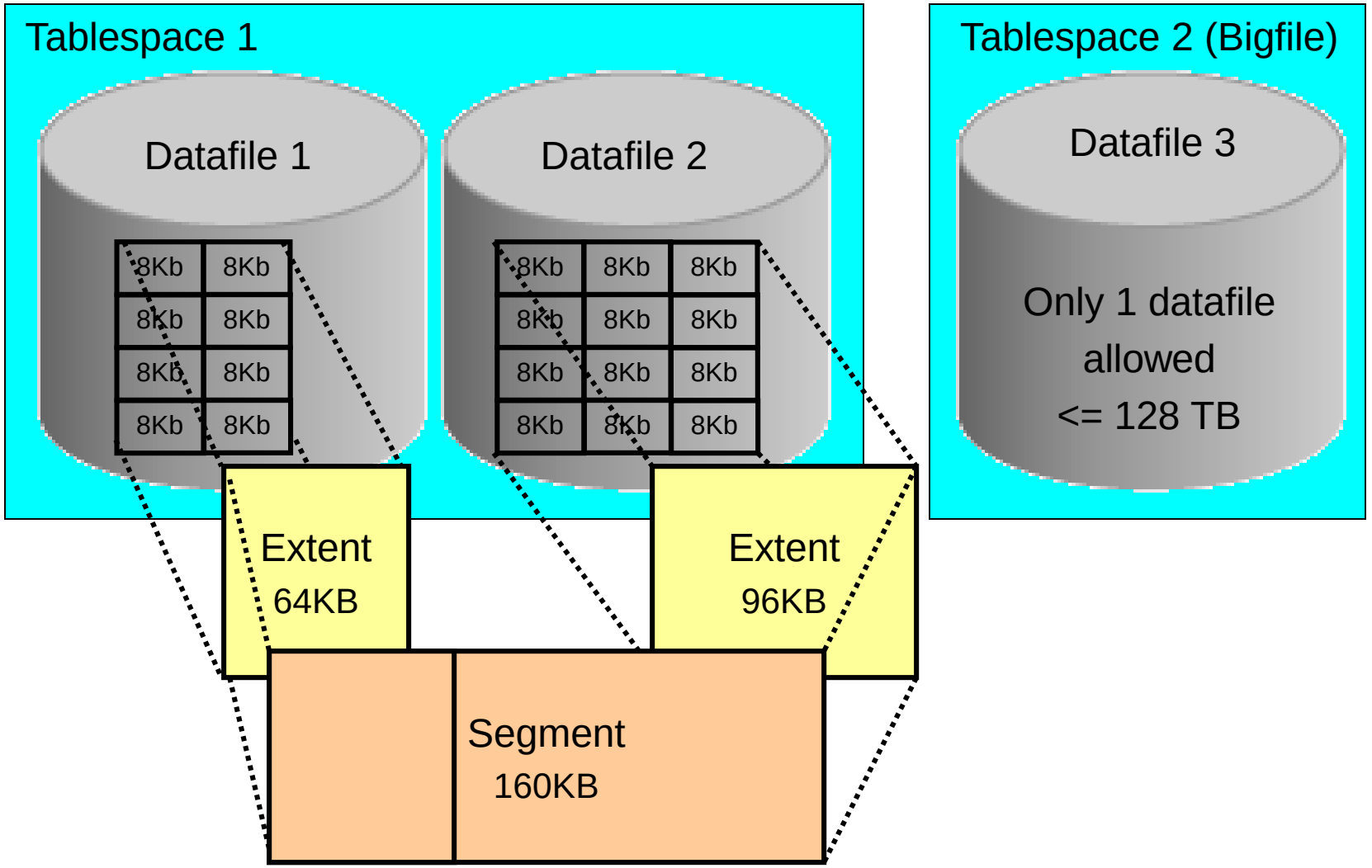
Logische und physische DB-Strukturen (2)

- **Extents**
  - Anzahl von Blöcken, die in einer Allokation zusammen angefordert wurden
  - Speichern eine bestimmte Art von Information
  - Logisch zusammenhängend, aber evtl. physisch verstreut (RAID striping, Dateisystem-Implementierung)
- **Segments**
  - Menge von Extents für eine bestimmte logische Struktur
- **Data segments**
  - Jede nicht geclusterte, nicht indexierte Tabelle hat ein eigenes Segment.
  - Ausnahmen: externe Tabellen, globale temporäre Tabellen, partitionierte Tabellen (dort ein Segment pro Partition)
  - Ein Cluster hat nur ein Segment für alle seine Tabellen.

Logische und physische DB-Strukturen (3)

- **Index segments**
  - Für jeden Index eins, mit allen seinen Daten
  - Bei partitionierten Indexen hat jede Partition eins.
- **Undo segments**
  - Ein Undo-Tablespace für jede DB-Instanz,
    enthält etliche Undo-Segmente
  - Verwendet für die Bereitstellung eines konsistenten DB-Zustands
    für lesende Transaktionen (während schon Änderungen laufen)
  - Und natürlich für Undo
- **Temporary segments**
  - Für die Dauer einer SQL-Anweisung
  - Entweder für jeden Benutzer eins oder für alle zusammen

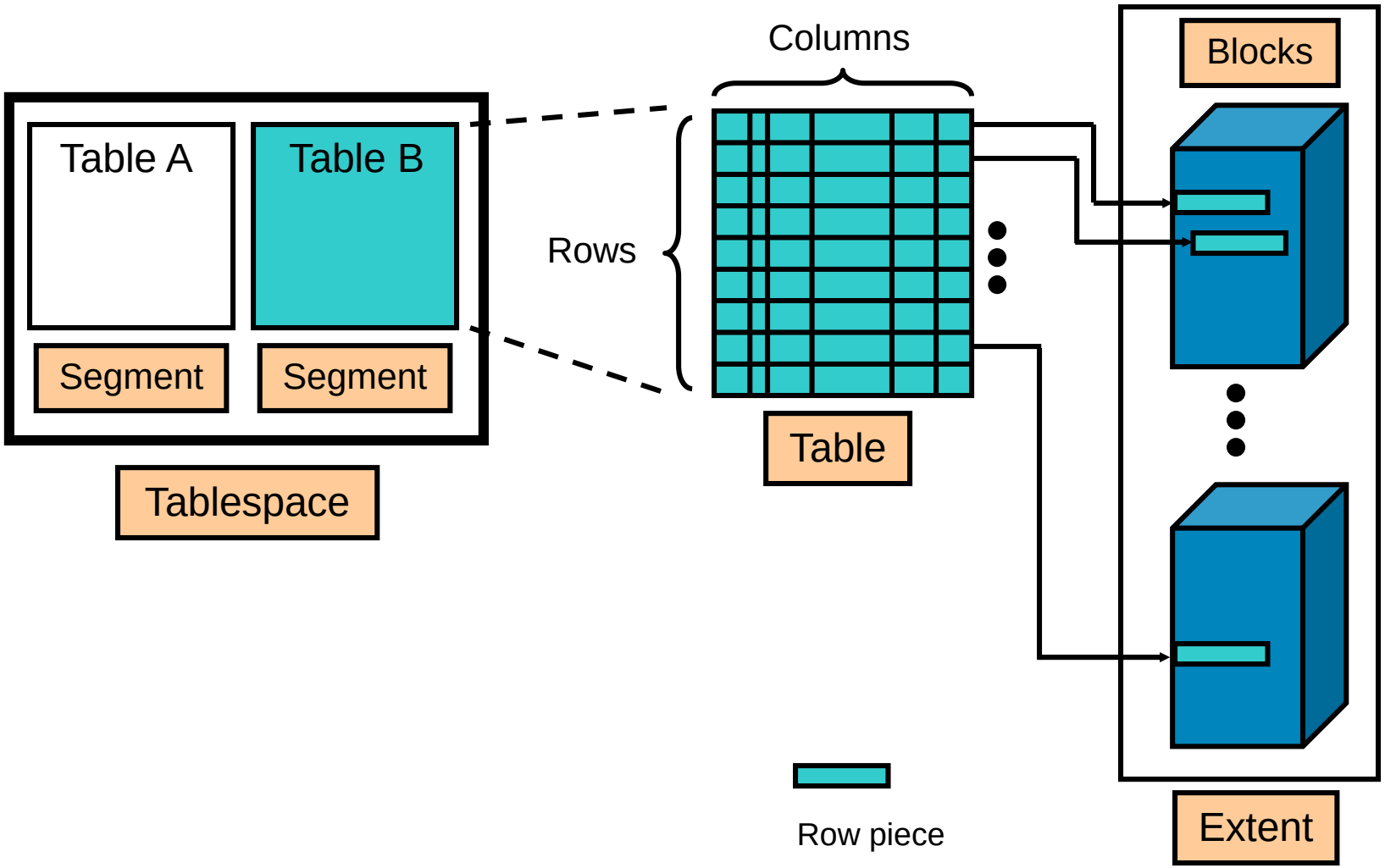
Tablespaces and Data Files



Tablespaces und Dateien

- **Tablespaces (Wdh.)**
  - Fassen logische Einheiten zusammen
  - Zwei oder mehrere: `SYSTEM` und `SYSAUX`
  - Verwenden eine oder mehrere Dateien für die Speicherung
- **Beispiel auf der letzten Folie:**
  - Tablespace 1 besteht aus zwei Dateien.
  - Segment von 160 KB Größe geht über beide Dateien, besteht deshalb aus zwei Extents.
  - Erstes Extent hat 64 KB und liegt in Datei 1, zweites hat 96 KB und liegt in Datei 2.
  - Beide Extents bestehen aus zusammenhängenden Blöcken von je 8 KB.
- **Bigfile Tablespaces:**
  - Nur eine Datei (die oft sehr groß wird)
  - Maximale Größe durch Art der Row-ID (siehe unten) vorgegeben: Blockgröße des Tablespace multipliziert mit 2^{36}, also z.B. 128 TB für eine Blockgröße von 32 KB.

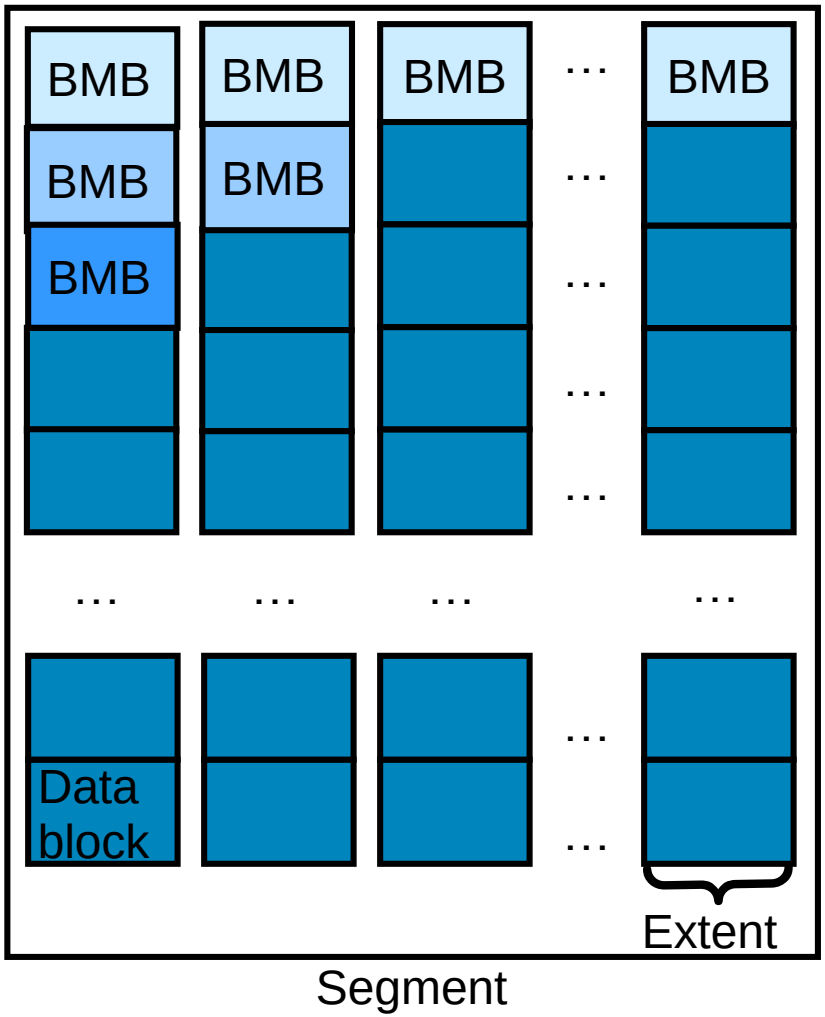
How Table Data Is Stored



Speicherung von Tabellen

- **Erzeugung**
  - Bei Erzeugung einer Tabelle
    wird ein Segment angelegt für ihre Daten.
- **Zeilen (Tupel, rows)**
  - In Blöcken abgelegt als Zeilen-Stücke (row piece, Sätze)
  - Zeile in mehreren Stücken,
    - wenn zu lang für einen Block (chained row)
    - oder durch Änderung zu groß geworden (migrated row)
  - Auch bei Tabellen mit mehr als 255 Spalten
    - Dann evtl. sogar mehrere Stücke im selben Block (intra-block chaining)

Free Space Management Within Segments

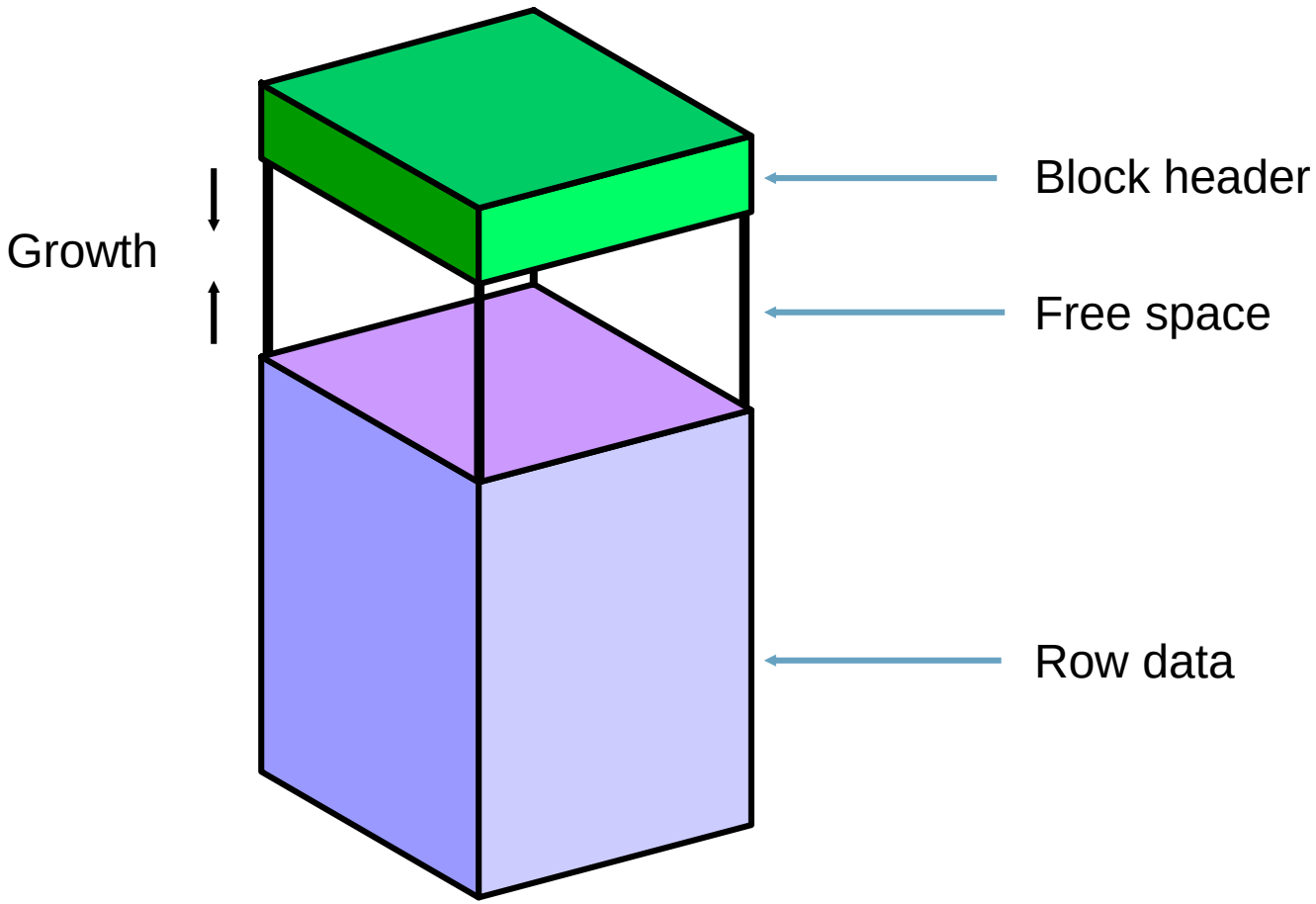


- Tracked by bitmap blocks (BMBs) in segments
- Benefits:
- More flexible space utilization
- Run-time adjustment
- Multiple process search of BMBs

Freispeicherverwaltung in Segmenten

- **Bitmaps**
  - Einschalten von "Automatic Segment Space Management" bei Erzeugung eines Tablespace
  - Gilt für alle Segmente, die ab dann darin angelegt werden
- **Menge von Bitmap Blocks (BMBs)**
  - Organisiert als Baumstruktur
  - Wurzel im Segment Header, enthält Verweise auf alle inneren Knoten
  - Blattknoten mit Belegungsinformation für zusammenhängende Folge von Datenblöcken
  - Maximal drei Ebenen im Baum

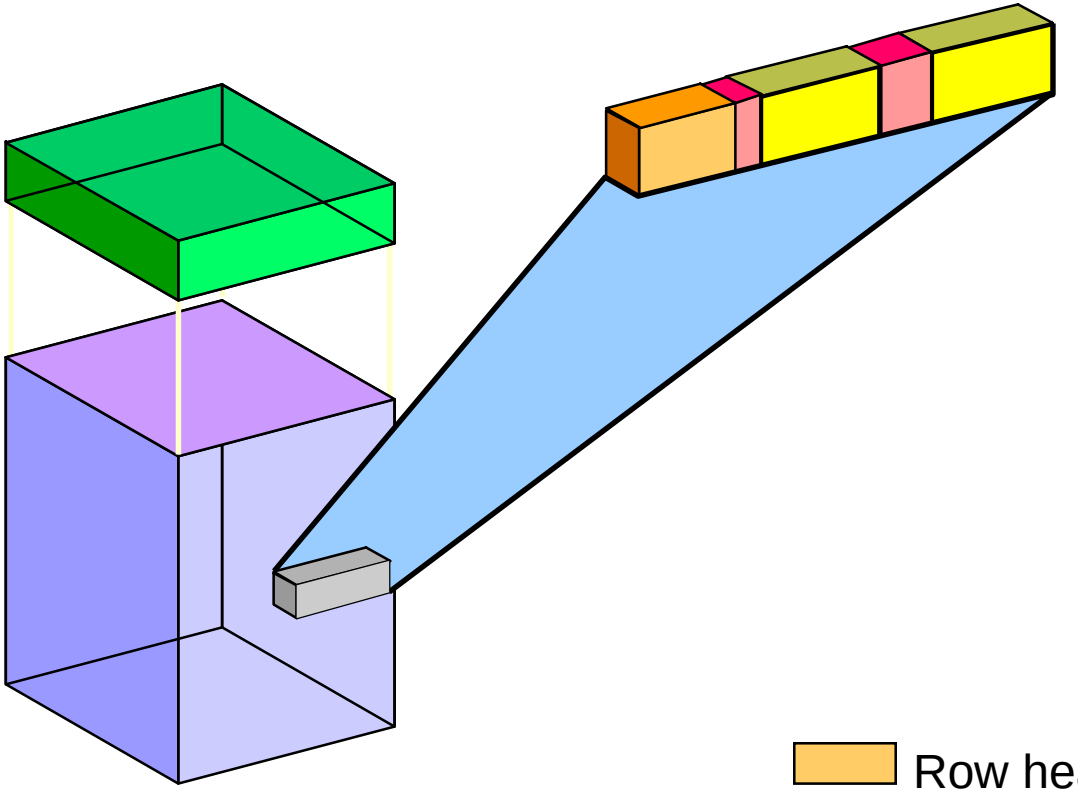
Database Block: Contents



Aufbau der Blöcke (Seiten)

- **Block header:**
  - Enthält Segment-Typ (z.B. Tabelle, Index), Datenblockadresse, Tabellenverzeichnis, Zeilenverzeichnis (Positionsindex) und Transaktionseinträge von jeweils ca. 23 Bytes, die bei Änderungen von Zeilen im Block benutzt werden
  - Wächst vom Anfang des Blocks her
    (durch neue Einträge im Zeilenverzeichnis und neue Transaktionseinträge)
- **Row data:**
  - Die Zeilen-Stücke: Sätze
  - Wächst vom Ende des Blocks her
- **Freier Speicherplatz:**
  - In der Mitte des Blocks
  - Am Anfang noch zusammenhängend, kann aber fragmentiert werden
  - Wird ggf. durch den Oracle-Server defragmentiert

Structure of a Row



Database block

Row header

Column length

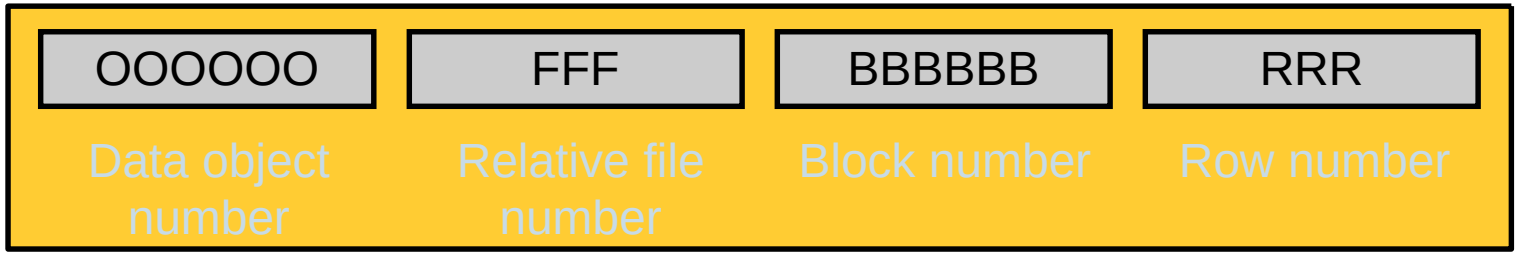
Column value

Struktur eines Satzes

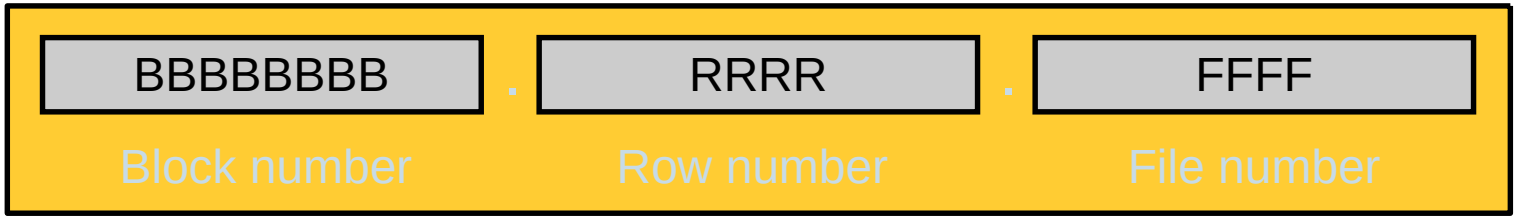
- **Variable Länge**
- **Spalten (Attribute) in der Reihenfolge ihrer Definition**
  - NULL-Werte
    - werden nur am Ende weggelassen
    - Ansonsten durch ein Byte (Längenfeld) dargestellt
- **Header:**
  - Anzahl der Spalten, Verkettungsinformation, Sperren
- **Daten:**
  - Für jede Spalte ein Längenfeld von einem Byte;
    bei sehr großen Werten (mehr als 250 Bytes) drei Bytes
  - Wert unmittelbar dahinter abgelegt
  - Kein Abstand erforderlich zwischen Sätzen
  - Jeder Satz ist im Zeilenverzeichnis (Positionsindex) eingetragen
    mit Zeiger auf den Anfang des Satzes.

ROWID Format

- Extended `ROWID` Format



- Restricted `ROWID` Format



ROWID

- **Analog zum TID**
- **Extended ROWID**
  - Braucht 10 Bytes Speicher und wird mit 18 Zeichen lesbar angezeigt
  - **Data object number:**
    - Wird jedem Datenobjekt (Tabelle, Index, …) bei seiner Erzeugung zugeordnet; eindeutig in der Datenbank (32 bits, 6 Zeichen)
  - **Relative file number:**
    - Eindeutig für jede Datei in einem Tablespace (10 bits, 3 Zeichen)
  - **Block number:**
    - Laufende Nummer des Blocks in der Datei (22 bits, 6 Zeichen)
  - **Row number:**
    - Laufende Nummer des Eintrags im Zeilenverzeichnis im Block-Header (16 bits, 3 Zeichen)
  - Lesbare Anzeige mit den 64 möglichen Zeichen A-Z, a-z und 0-9

ROWID (2)

- **Restricted ROWID**
  - Oracle7 und früher
  - Nur 6 Bytes Speicher
  - Dateinummern waren noch eindeutig in einer Datenbank.
    - Daher nicht mehr als 1.022 Dateien in einer Datenbank
    - Jetzt: in einem Tablespace
  - Immer noch verwendet in nicht-partitionierten Indexen und nicht-partitionierten Tabellen, weil alle Index-Einträge auf Zeilen (Sätze) im selben Segment verweisen.
- **Lokalisierung einer Zeile mit der ROWID:**
  - Segment liegt immer in genau einem Tablespace.
  - Also kann mit der Datenobjekt-Nummer der Tablespace identifiziert werden, in dem die Zeile (der Satz) gespeichert ist.

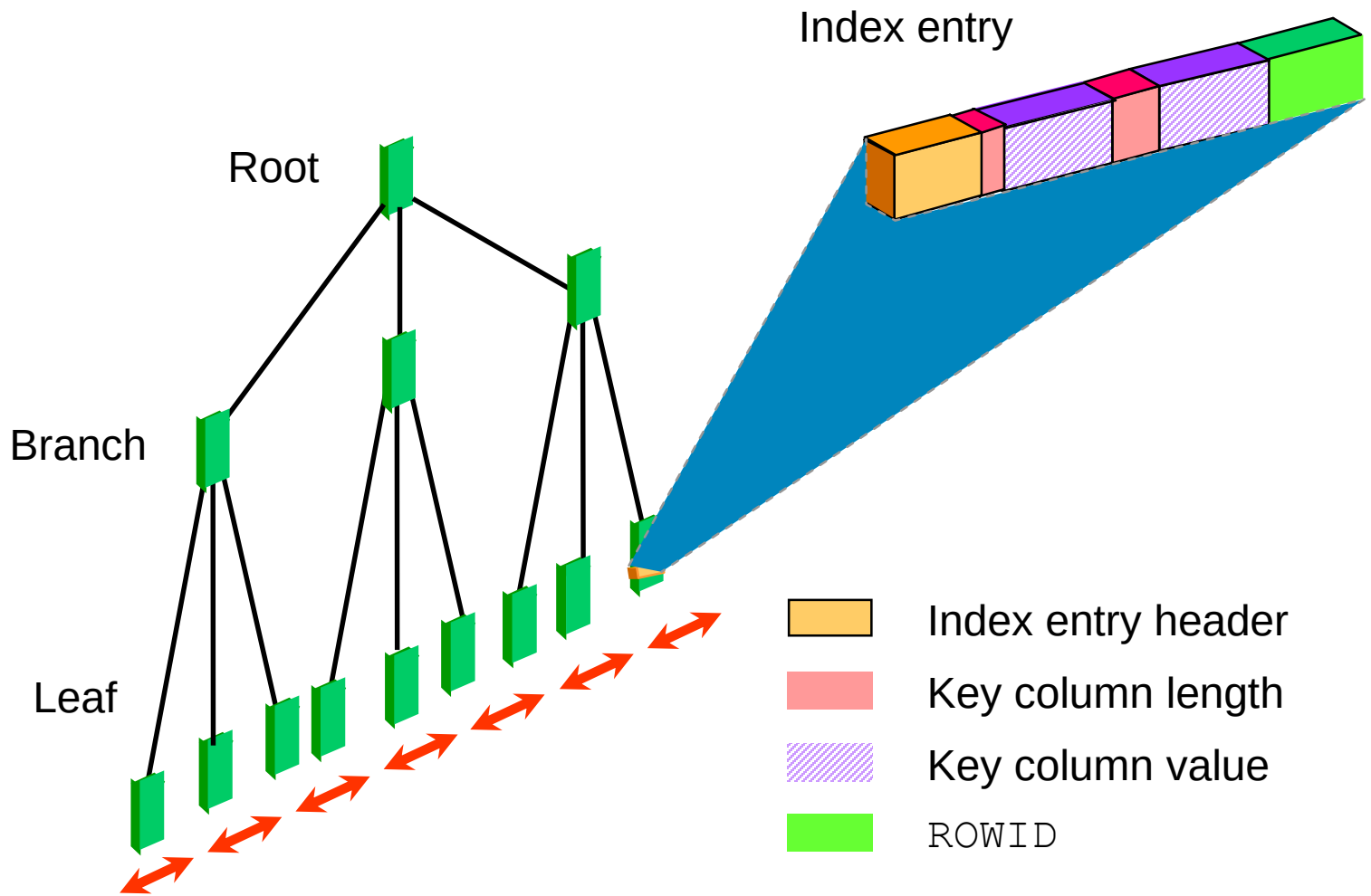
Indexstrukturen

- **Baumstruktur**
  - Kein Hashing, aber B-Baum und Bitmap
- **Single-column Index:**
  - Nur eine Spalte (ein Attribut)
- **Concatenated Index:**
  - Auch: composite index
  - Mehrere Spalten zusammen
    - Reihenfolge wählbar
    - Müssen auch nicht direkt hintereinander definiert worden sein
  - Maximal 32 Spalten
  - Gesamtgröße darf Hälfte der Blockgröße nicht überschreiten.

Indexstrukturen (2)

- **Unique and Nonunique Indexes:**
  - Klar
- **Function-based Indexes:**
  - Index-Einträge sind Werte einer Funktion (eines Ausdrucks), die auf eine Spalte oder mehrere Spalten einer Zeile angewendet wird.
  - Siehe Übung!
- **Domain Indexes:**
  - Für spezielle Datentypen (Text, Vektorgraphik)
  - Erzeugt, verwaltet und benutzt von einem eigenen Index-Typ
  - Nur single-column erlaubt
  - Spalten sind atomar, Objekte oder Large Objects (LOBs)
- **Partitioned and Nonpartitioned Indexes:**
  - Partitionierung über mehrere Segmente oder sogar Tablespaces
  - Erhöht Nebenläufigkeit (Sperrgranulat) und Wartbarkeit
  - Oft verwendet zusammen mit partitionierten Tabellen; dann je eine Index-Partition pro Tabelle-Partition.

B-Tree Index


Index entry
Root
Branch
Leaf
Index entry header
Key column length
Key column value
ROWID

B-Baum

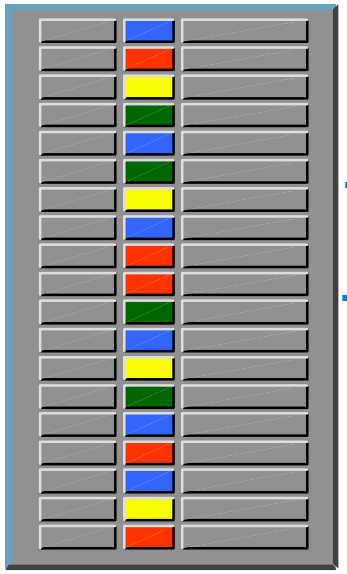
- **Liste von ROWIDs für jeden Schlüsselwert**
  - Also nur als **Sekundärorganisation**
- **Struktur:**
  - Index-Einträge nur in den Blattknoten, also **B*-Baum**
  - Doppelte Verkettung der Blattknoten wie gehabt
- **Index-Einträge:**
  - Header: Zahl der Spalten und Sperrinformation
  - Für jede Spalte das Paar von Länge und Wert
  - ROWID

B-Baum (2)

- **Blattknoten:**
  - Schlüsselwerte werden wiederholt, wenn sie in mehreren Zeilen vorkommen, falls der Index nicht komprimiert wird.
  - Kein Index-Eintrag, wenn alle Schlüsselwerte NULL sind
    - Suche mit IS NULL in der WHERE-Klausel benutzt den Index also nicht.
  - Verwendung der Restricted ROWID
- **Wirkung von Änderungsoperationen:**
  - Einfügen einer Zeile erzeugt einen neuen Index-Eintrag.
  - Löschen einer Zeile markiert nur den Index-Eintrag als gelöscht.
  - Änderungen werden als Löschen und Einfügen realisiert.

Bitmap Indexes

Table



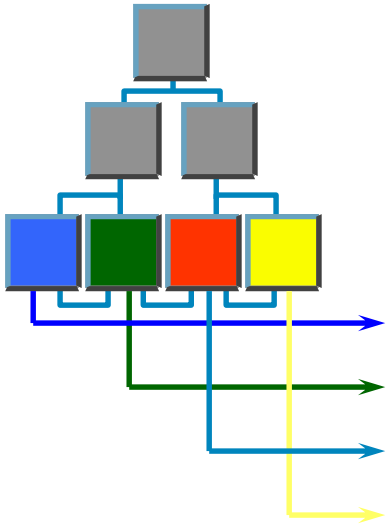
File 3

Block 10

Block 11

Block 12

Index



key	start ROWID	end ROWID	bitmap
<Blue,	10.0.3,	12.8.3,	10001001000100101 00>
<Green,	10.0.3,	12.8.3,	00010100001001000 00>
<Red,	10.0.3,	12.8.3,	01000000110000010 01>
<Yellow,	10.0.3,	12.8.3,	00100010000010000 10>

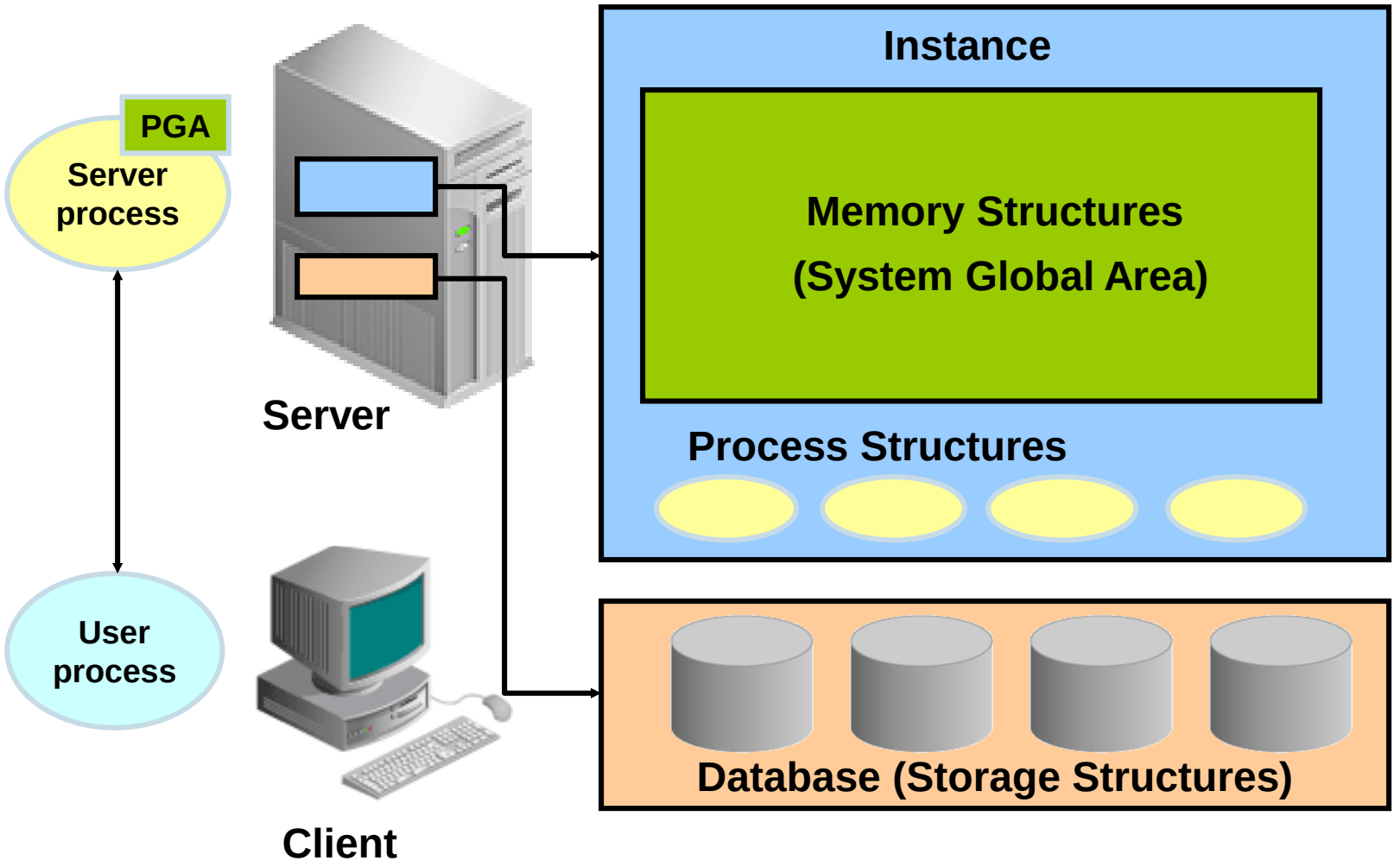
Bitmap-Index

- **Vorteile (wie bekannt):**
  - Bei vielen Zeilen und wenigen Werten in den Schlüsselspalten (Geschlecht, Familienstand)
  - Bei Anfragen mit mehreren Bedingungen in den WHERE-Klausel, die mit OR verknüpft sind
  - Bei vielen lesenden und wenigen schreibenden Zugriffen auf die Schlüsselspalten
- **Struktur:**
  - Ebenfalls als B*-Baum organisiert,
    aber mit einer Bitmap in den Index-Einträgen an Stelle der ROWIDs
  - Jedes Bit gehört zu einer bestimmten ROWID.

Bitmap-Index (2)

- **Blattknoten (= ein Index-Eintrag):**
  - Header: Zahl der Spalten und Sperrinformation
  - Für jede Spalte das Paar von Länge und Wert
  - Start ROWID: erste Zeile der Tabelle, zu der das erste Bit im Bitmap gehört
  - End ROWID: letzte Zeile der Tabelle, zu der das letzte Bit im Bitmap gehört
  - Bitmap komprimiert gespeichert
  - Verwendung der Restricted ROWID

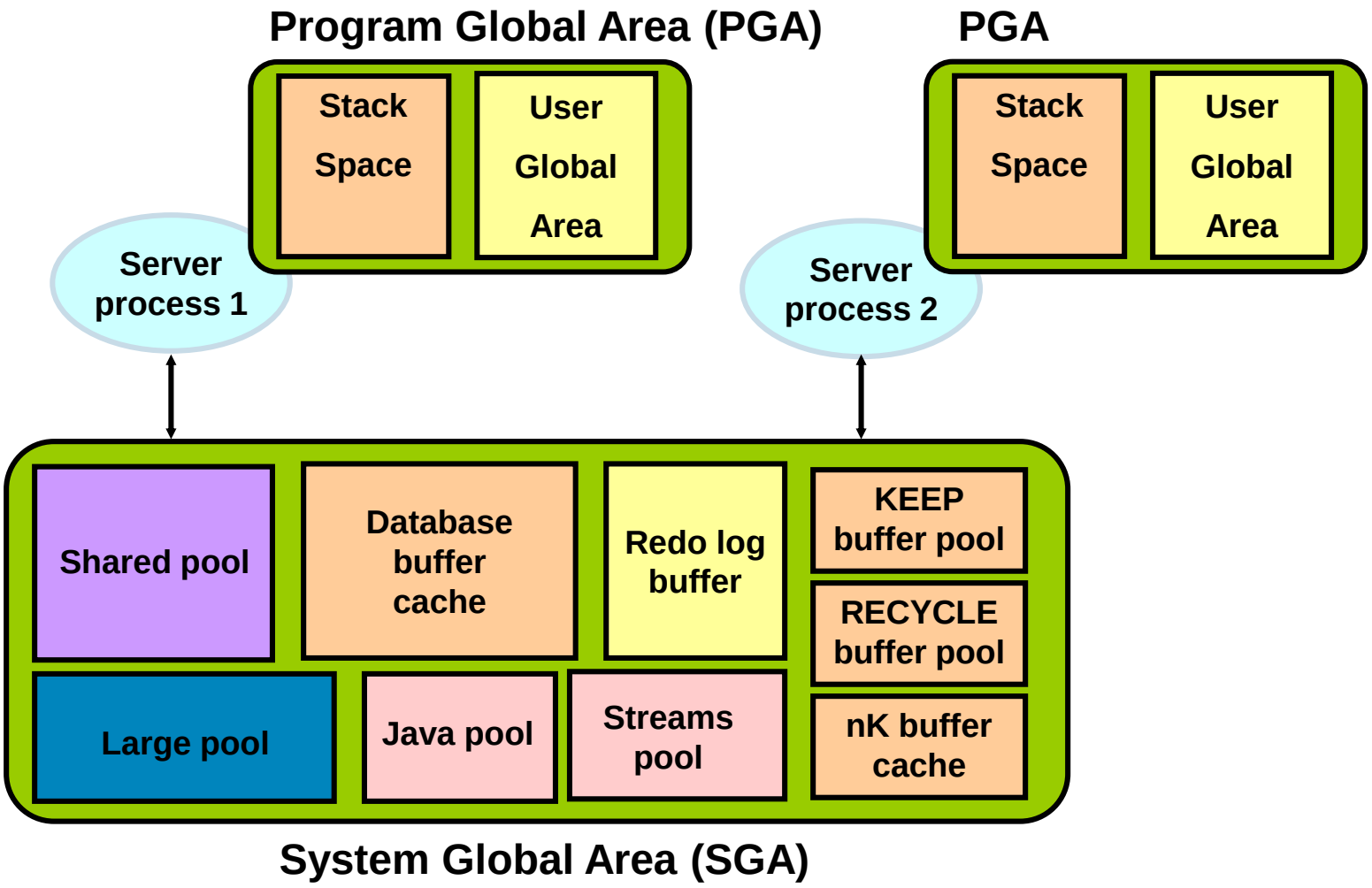
Oracle Database Server Architecture: Overview



Oracle-Datenbank-Instanzen

- **Oracle DBS besteht aus:**
  - Oracle-Datenbank und
  - DB-Instanz
- **DB-Instanz**
  - Datenstrukturen und Prozesse, die damit assoziiert sind
  - System Global Area (SGA)
  - Hintergrundprozesse
- **Datenbank assoziieren: Mounting**
- **Anmerkung:**
  - Oracle Automatic Storage Management (ASM) verwendet ebenfalls das Konzept einer Instanz mit Datenstrukturen und Prozessen, ist aber nicht mit einer bestimmten Datenbank assoziiert.

Oracle Database Memory Structures



Datenstrukturen

- **Im Hauptspeicher ("Memory")**
  - Code, gemeinsame Daten, Benutzer-spezifische Daten
- **System Global Area (SGA):**
  - Gemeinsam für eine DB-Instanz
  - Zugriff durch alle Server- und Hintergrund-Prozesse
  - Beispiele: Blöcke im Puffer, gemeinsame SQL-Bereiche
- **Program Global Areas (PGA):**
  - Getrennt für jeden einzelnen Server- oder Hintergrund-Prozess

System Global Area (SGA)

- **Shared pool:**
  - Diverse Dinge für alle Benutzer
- **Database buffer cache:**
  - Datenbank-Puffer
- **KEEP buffer pool:**
  - Spezieller Puffer für Blöcke, die länger im Hauptspeicher bleiben sollen
- **RECYCLE buffer pool:**
  - Spezieller Puffer, um schnell Blöcke wiederverwenden oder aus dem Puffer entfernen zu können
- **nK buffer cache:**
  - Spezieller Puffer für Blöcke mit abweichender Größe
- **Redo log buffer:**
  - Zwischenspeicher für Redo-Information, bis sie auf die Log-Datei auf Platte geschrieben werden kann

SGA (2)

- **Large pool:**
  - Optionaler Bereich, der größeren Speicher für bestimmte Prozesse bereitstellt, etwa für Backup und Recovery sowie für I/O-Server-Prozesse
- **Java pool:**
  - Für sämtlichen Session-spezifischen Java-Code und die Daten der JVM
- **Streams pool:**
  - Für Oracle Streams zum Speichern von Information, die von "capture" und "apply" benötigt wird

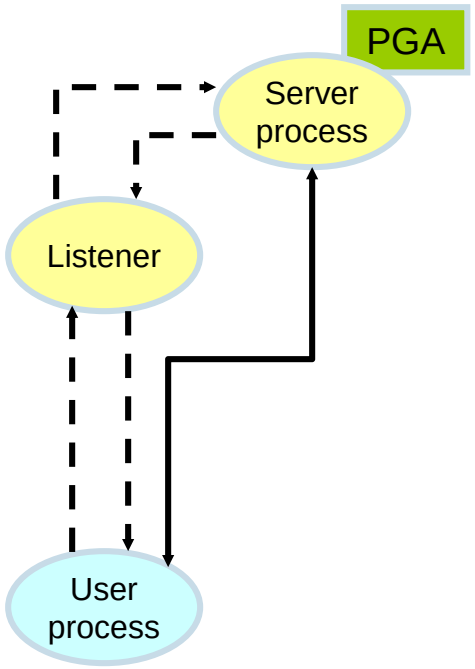
- **Dynamic SGA Infrastructure**
  - Die Größe von Database buffer cache, Shared pool, Large pool, Java pool und the Streams pool kann sich ändern, ohne dass die Instanz neu gestartet warden muss.

Program Global Area (PGA)

- **Für jeden Server-Prozess getrennt**
  - Bedient einen Client
  - Zwei Teile: Stack Space und User Global Area (UGA)

- **Initialisierungsparameter**
  - Für die Erzeugung und Verwaltung der Datenstrukturen
  - Oracle kann sie auch selbst setzen.
    - Dann nur noch Zielgröße (MEMORY_TARGET) und Maximalgröße (MEMORY_MAX_TARGET) für den Speicher setzen.

Process Structures



Instances (ASM and Database separate)

System Global Area (SGA)

Background processes

Required: DBW*n* CKPT LGWR SMON PMON RECO

Optional: ARC*n* ASMB RBAL Others

Grid Infrastructure Processes (ASM and Oracle Restart)

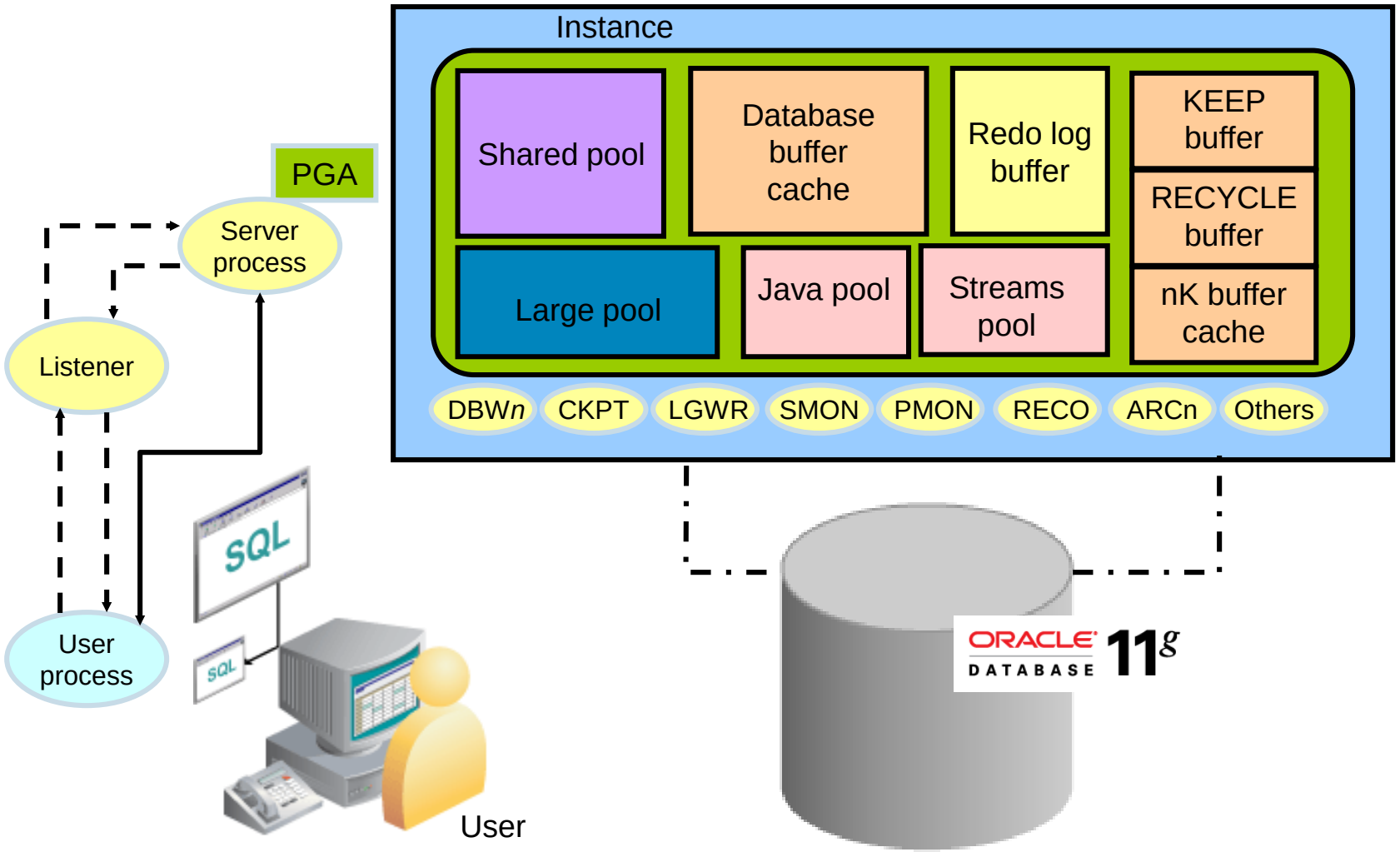
ohas ocssd diskmon

orarootagent oraagent cssdagent

Prozess-Strukturen

- **Server-Prozesse**
  - Bearbeiten Anfragen der Benutzerprozesse
  - Diese kommunizieren erst mit dem **Listener**,
    der einen Server-Prozess in einer definierten Umgebung erzeugt.
  - Parsen SQL-Anweisungen und führen sie aus
  - Holen benötigte Datenblöcke aus den Dateien in den Datenbank-Puffer in der SGA (falls sie nicht schon in der SGA vorhanden waren)
  - Liefern die Ergebnisse in passender Form an die Benutzerprozesse zurück
- **Hintergrund-Prozesse**
  - Performance verbessern, viele Benutzer bedienen

Oracle Database Server Architecture: Overview



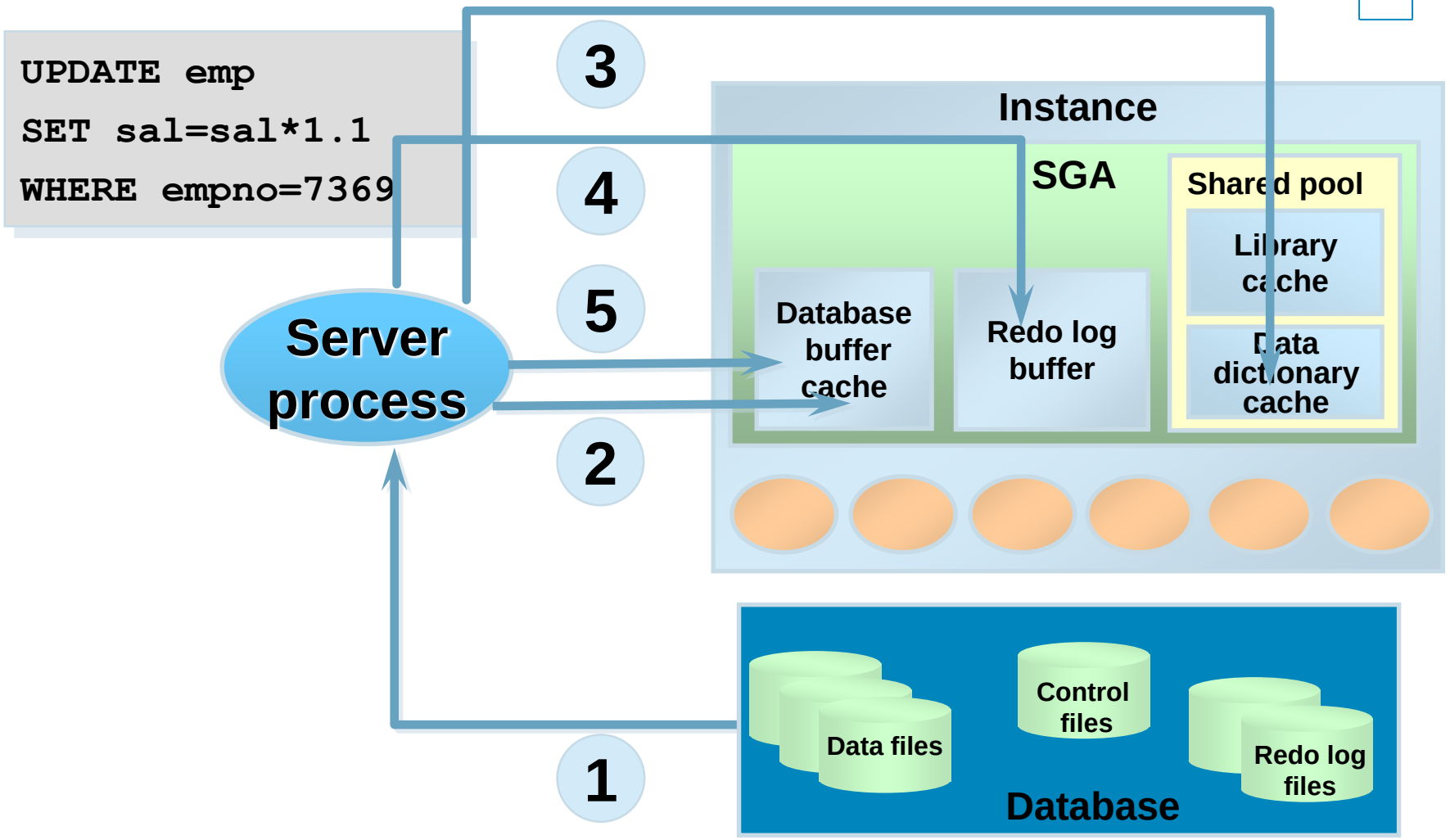
Interaktion mit einer Oracle-Datenbank

1. Eine DB-Instanz läuft auf einem Knoten;
   oft als *Host* oder *Datenbank-Server* bezeichnet.
2. Ein Benutzer startet eine Anwendung in einem Benutzer-Prozess.
   Diese Anwendung nimmt Verbindung zum Server auf.
   Die Verbindung kann lokal, Client-Server oder auch Middleware sein.
3. Server hat einen **Listener**, der über den erforderlichen Oracle-Net-Services-Handler verfügt. Listener erkennt die Verbindungsaufforderung der Anwendung und erzeugt einen zugeordneten Server-Prozess.
4. Benutzer führt SQL-Anweisungen aus und beendet die Transaktion (commit).
5. Server-Prozess nimmt die Anweisungen entgegen und prüft im Shared Pool, ob die **Shared SQL Area** eine identische Anweisung enthält. Falls ja, prüft der Server-Prozess die Zugriffsberechtigungen und führt dann die Anweisung mit den Inhalten der Shared SQL Area aus. Andernfalls wird eine neue Shared SQL Area angelegt, mit der die Anweisung dann geparst und ausgeführt wird.

Interaktion mit einer Oracle-Datenbank (2)

6. Server-Prozess holt die erforderlichen Daten, entweder aus Dateien oder aus dem Datenbank-Puffer.
7. Server-Prozess modifiziert die Daten in der SGA. Wenn die Transaktion erfolgreich abgeschlossen wird, schreibt der **Log-Writer-Prozess** (LGWR) die Daten unmittelbar in die Redo-Log-Datei. Der **Database-Writer-Prozess** (DBW*n*) schreibt die geänderten Blöcke erst dann auf die Platte, wenn es effizient möglich ist (NoFORCE).
8. Bei erfolgreichen Transaktionen sendet der Server-Prozess eine Nachricht über das Netz an die Anwendung. Bei Misserfolg wird eine Fehlermeldung gesendet.
9. Während dieser ganzen Prozedur laufen auch die anderen Hintergrund-Prozesse weiter und achten auf Bedingungen, die ein Eingreifen erfordern könnten. Außerdem führt der Datenbank-Server auch noch andere Transaktionen aus und verhindert Konflikte, wenn Transaktionen auf dieselben Daten zugreifen.

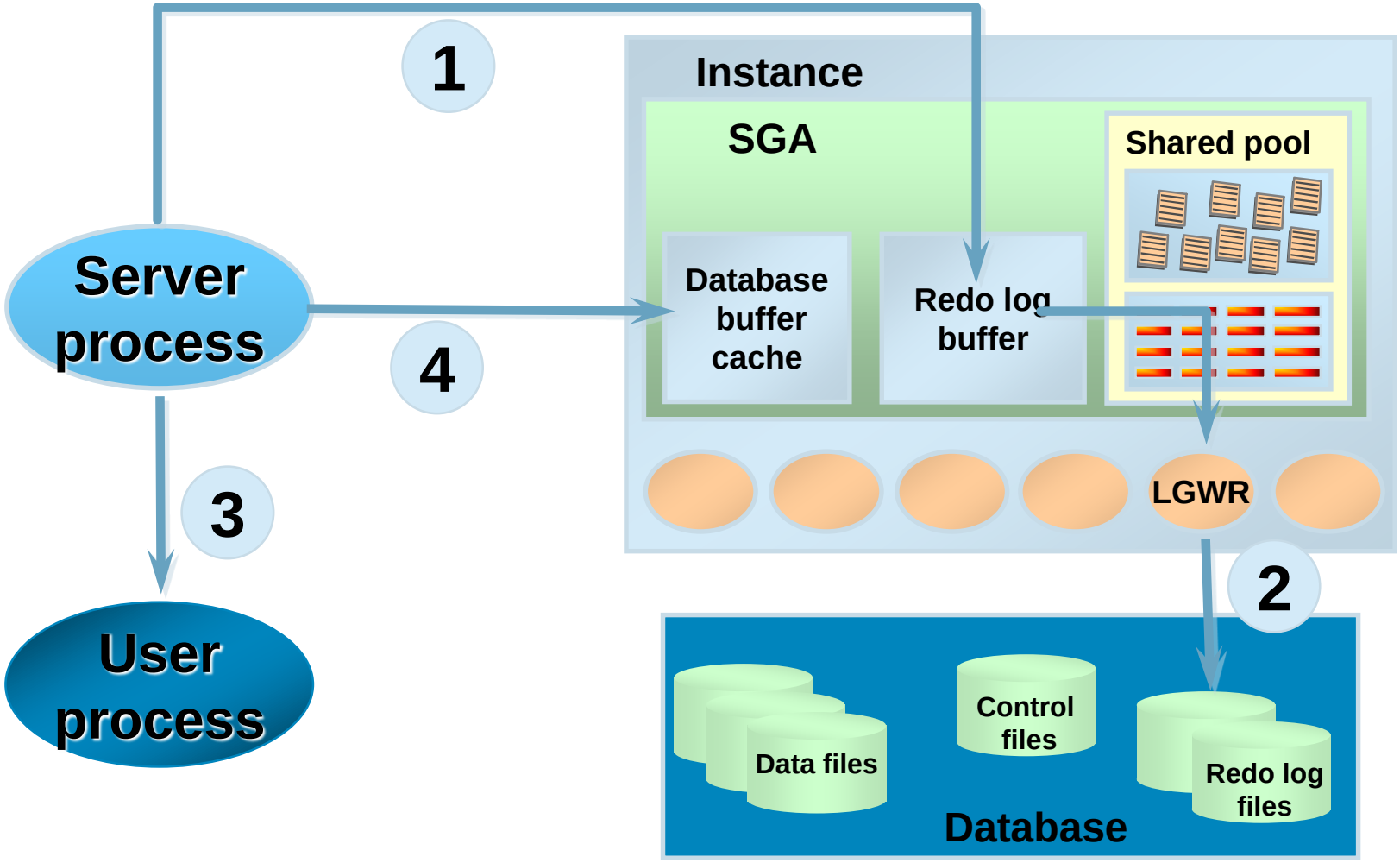
Processing a DML Statement



Verarbeitung einer Änderungsanweisung

- **Schritte:**
  1. Der Server-Prozess liest die Daten- und Rollback-Blöcke aus der Datei, falls sie nicht bereits im Puffer sind.
  2. Kopien der gelesenen Blöcke werden im Puffer abgelegt.
  3. Der Server-Prozess setzt Sperren auf den Daten.
  4. Der Server-Prozess protokolliert die durchzuführenden Änderungen für Rollback (Before-Image) und für Wiederherstellung (After-Image) in den Redo-Log-Puffer.
  5. Der Server-Prozess schreibt das Before-Image in den Rollback-Block und ändert den Daten-Block, beides im Datenbank-Puffer. Die beiden geänderten Blöcke werden als "dirty" markiert, d.h. sie haben nicht mehr den gleichen Inhalt wie die entsprechenden Blöcke auf der Platte.
- **Anmerkung:**
  - Die Verarbeitung von DELETE oder INSERT erfolgt in ähnlichen Schritten. Das Before-Image für ein DELETE enthält die Attributwerte der gelöschten Zeile. INSERTs benötigen nur die Identifikation der Zeile als Information im Rollback.

COMMIT Processing



COMMIT-Verarbeitung

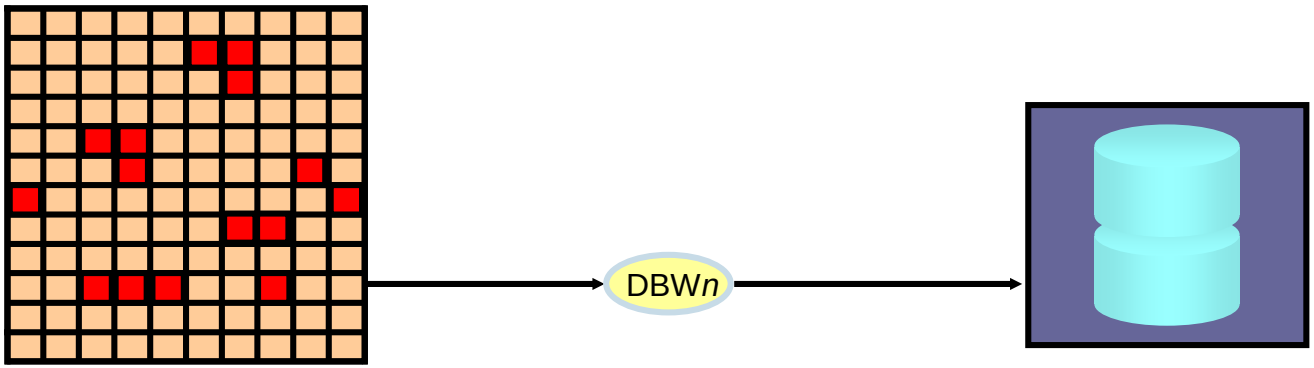
- **System Change Number (SCN)**
  - Immer wenn eine Transaktion erfolgreich endet, weist Oracle eine SCN zu. Diese sind monoton aufsteigend und eindeutig innerhalb der Datenbank. Die SCN wird von Oracle als interner Zeitstempel benutzt, um Datenzugriffe zu synchronisieren und um Lesekonsistenz zu gewährleisten. Oracle macht damit Konsistenzprüfungen, ohne sich auf Datum und Uhrzeit des Betriebssystems verlassen zu müssen.
- **Schritte der COMMIT-Verarbeitung:**
  - Der Server-Prozess schreibt einen COMMIT-Satz zusammen mit der SCN in den Redo-Log-Puffer.
  - LGWR schreibt zusammenhängend alle Einträge des Redo-Log-Puffers bis zum diesem COMMIT-Satz (einschließlich) in die Redo-Log-Datei. Danach kann Oracle garantieren, das die Änderungen auch im Fehlerfall nicht mehr verlorengehen.
  - Der Benutzer wird informiert, dass das COMMIT vollständig abgeschlossen ist.
  - Der Server-Prozess schreibt Informationen, die anzeigen, dass die Transaktion vollständig ist und dass die Sperren freigegeben werden können. Das Schreiben der geänderten Pufferinhalte in die Dateien kann davon unabhängig vom DBWR vorgenommen werden (NoFORCE). Es kann sowohl vor als auch nach dem COMMIT erfolgen.

COMMIT-Verarbeitung (2)

- **Anmerkung:**
  - Das Zurücksetzen einer Transaktion veranlasst den LGWR nicht, auf Platte zu schreiben. Oracle setzt unvollständige Transaktionen immer zurück, wenn es Fehler behandelt. Falls ein Fehler nach einem Rollback auftritt und noch bevor die Rollback-Einträge (als durchgeführt) auf die Platte geschrieben werden können, genügt das Fehlen des COMMIT-Satzes, um sicherzustellen, dass die Änderungen der Transaktion zurückgesetzt werden.
- **Vorteile dieses Fast COMMIT (= NoFORCE):**
  - Sequenzielles Schreiben in die Protokoll-Dateien ist schneller als das verstreute Schreiben von Daten-Blöcken.
  - Nur minimale Information, die unbedingt erforderlich ist, um Änderungen zu protokollieren, wird in die Protokoll-Dateien geschrieben, während beim Schreiben in die Datenbank immer ganze Blöcke geschrieben werden müssen.
  - Redo-Log-Sätze mehrerer Transaktionen, die zur gleichen Zeit abgeschlossen werden, können in einer einzigen Schreiboperation gesichert werden (sog. "Group-Commit").
  - Falls der Redo-Log-Puffer nicht zu voll ist, wird nur eine einzige synchrone Schreiboperation pro Transaktion benötigt (oder sogar weniger als eine).
  - Der Umfang einer Transaktion beeinflusst die Zeit, die für ein COMMIT benötigt wird, nicht.

Database Writer Process (DBWn)

- **Writes modified (dirty) buffers in the database buffer cache to disk:**
  - Asynchronously while performing other processing
  - To advance the checkpoint



Database-Writer-Prozess (DBW*n*)

- **Schreibt den Inhalt von Pufferseiten in die Dateien**
  - Modifizierte ("dirty") Seiten
  - Einer (DBW*0*) genügt für die meisten Systeme,
    aber man kann weitere einrichten (DBW*1* bis DBW*9* und DBW*a* bis DBW*z*),
    um die Schreibgeschwindigkeit zu erhöhen, wenn die Daten oft geändert werden. Allerdings zahlt sich das nur bei Mehrprozessorsystemen aus.
  - Wird eine Seite im Puffer geändert, wird sie als "dirty" markiert und in die Sicherungspunkt-Warteschlange eingetragen. Diese Warteschlange wird in SCN-Reihenfolge organisiert. Das entspricht der Reihenfolge der Redos, die in den Redo-Log geschrieben werden.
  - Fällt die Zahl der freien Pufferseiten unter einen internen Schwellenwert, schreibt DBW*n* nicht so häufig genutzte Seiten in die Dateien zurück, beginnend mit dem Ende der LRU-Kette.
  - DBW*n* schreibt außerdem vom Ende der Sicherungspunkt-Warteschlange her, damit die Sicherungspunkte neu gesetzt werden können.

DBW*n* (2)

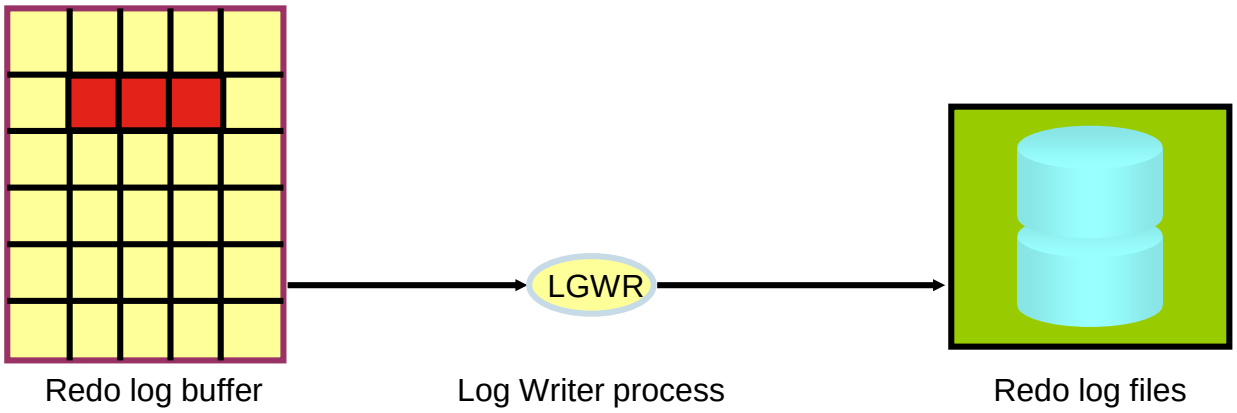
- **Redo Byte Address (RBA)**
  - Steht in der SGA
  - Zeiger auf Position im Redo-Strom, an der die Recovery beginnen soll.
  - Wird alle drei Sekunden vom CKPT-Prozess in die Steuerdatei geschrieben.
  - Da DBW*n* geänderte Seiten in SCN-Reihenfolge schreibt und der Redo-Log ebenfalls in SCN-Reihenfolge abgelegt wird, kann DBW*n* den Zeiger immer dann weiterrücken, wenn er geänderte Seiten aus der LRU-Liste zurückschreibt. Deshalb beginnt die Recovery das Redo an der annähernd korrekten Position und vermeidet unnötige E/A. Man nennt das auch Inkrementelles Checkpointing.
- **Anmerkung:**
  - Es gibt weitere Anlässe für den DBW*n* zu schreiben, z.B. wenn ein Tablespace auf "read-only" oder "offline" gesetzt wird. Dann wird kein inkrementeller Sicherungspunkt gesetzt, weil alle geänderten Seiten der zugehörigen Dateien einfach in die Datenbank geschrieben werden, ohne auf die SCN-Reihenfolge zu achten.

DBW*n* (3)

- **LRU-Algorithmus**
  - Hält die häufig verwendeten Blöcke im Puffer (wie gehabt).
  - CACHE-Option kann für bestimmte Tabellen gesetzt werden, damit ihre Seiten länger im Puffer bleiben.
- **Initialisierungsparameter DB_WRITER_PROCESSES**
  - Legt die Zahl der DBW*n*-Prozesse fest.
  - Maximum ist 36.
  - Wird er beim Hochfahren nicht angegeben, bestimmt Oracle den Wert auf der Basis der Prozessorzahl und -gruppen.
- **Bedingungen für das Schreiben geänderter Pufferseiten durch DBW*n*:**
  - Findet ein Server-Prozess nach dem Scannen einer gegebenen Zahl von Pufferseiten keine freie, stößt er DBW*n* zum Schreiben an. DBW*n* schreibt asynchron.
  - DBW*n* schreibt, um den Sicherungspunkt weiterzurücken. Er entspricht der ältesten geänderten Seite im Puffer.
- **DBW*n* schreibt immer mehrere Blöcke auf einmal (batch, multiblock).**
  - Die Zahl variiert mit dem Betriebssystem.

Log Writer Process (LGWR)

- Writes the redo log buffer to a redo log file on disk
- Writes:
  - When a user process commits a transaction
  - When the redo log buffer is one-third full
  - Before a DBWn process writes modified buffers to disk
  - Every 3 seconds

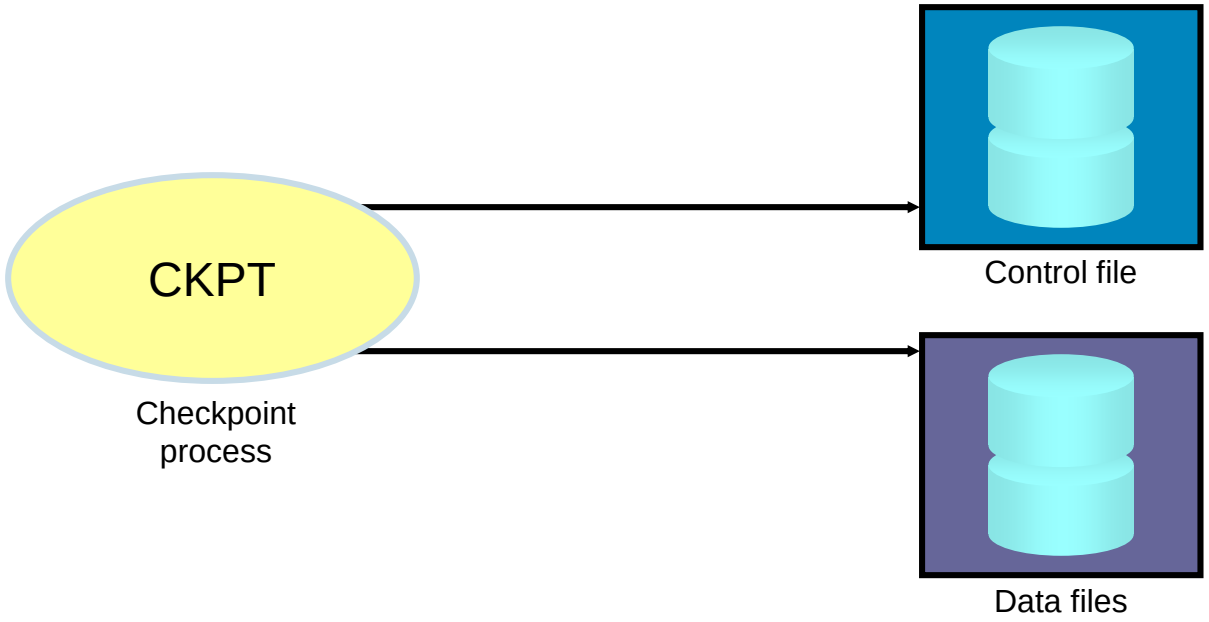


Log-Writer-Prozess (LGWR)

- **Verwaltet den Redo-Log-Puffer**
  - Schreibt dessen Einträge in die Redo-Log-Dateien auf der Platte.
  - Immer alle, die seit dem letzten Schreiben eingetragen wurden.
- **Zyklischer Puffer**
  - Während LGWR in die Protokolldatei schreibt, können Server-Prozesse neue Einträge schon dort ablegen, wo bereits ausgeschrieben wurde.
  - LGWR schreibt normalerweise schnell genug, um sicherzustellen, dass immer genug Platz ist für neue Einträge, selbst wenn viele Zugriffe auf den Redo-Log erfolgen.
  - LGWR schreibt einen zusammenhängenden Abschnitt des Puffers auf die Platte.
- **Anlässe:**
  - Benutzerprozess beendet eine Transaktion mit COMMIT (WAL)
  - Redo-Log-Puffer ist zu einem Drittel voll
  - Bevor ein DBW*n*-Prozess geänderte Seiten auf die Platte schreibt (falls nötig)
  - Alle drei Sekunden

Checkpoint Process (CKPT)

- Records checkpoint information in
  - Control file
  - Each data file header

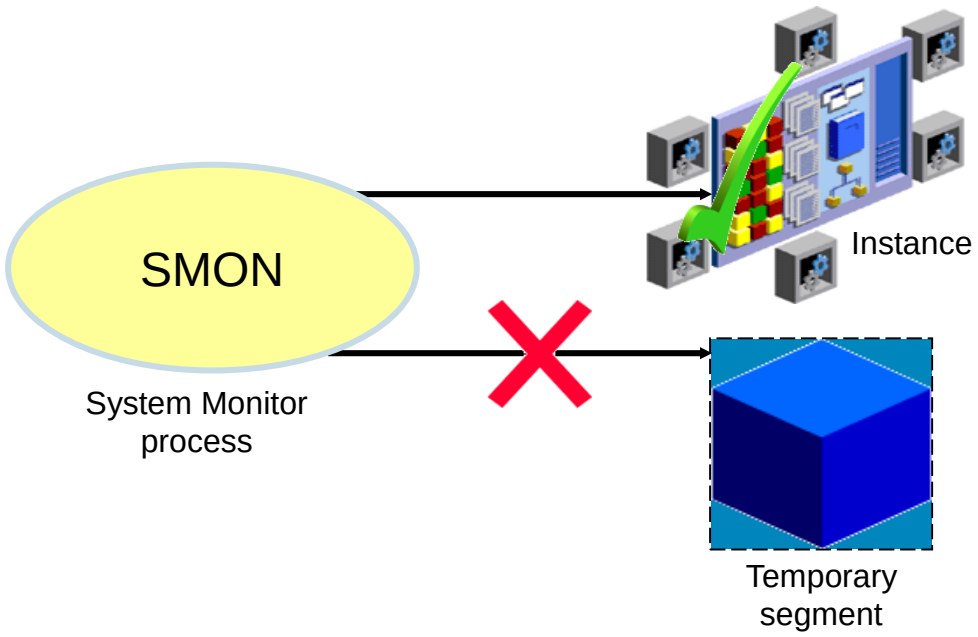


Checkpoint-Prozess (CKPT)

- **Sicherungspunkt**
  - Datenstruktur, die eine System Change Number (SCN) in der Redo-Abfolge einer Datenbank definiert.
  - Werden in der Steuerungsdatei (Control File) und im Header jeder Datei vermerkt.
  - Das macht der CKPT-Prozess.
  - Der schreibt selbst keine Seiten auf die Platte; das macht DBW*n*.
  - Die SCN im Datei-Header stellt sicher, dass alle Änderungen an der Datenbank, die vor dieser SCN durchgeführt wurden, bereits in der Datei auf Platte enthalten sind.

System Monitor Process (SMON)

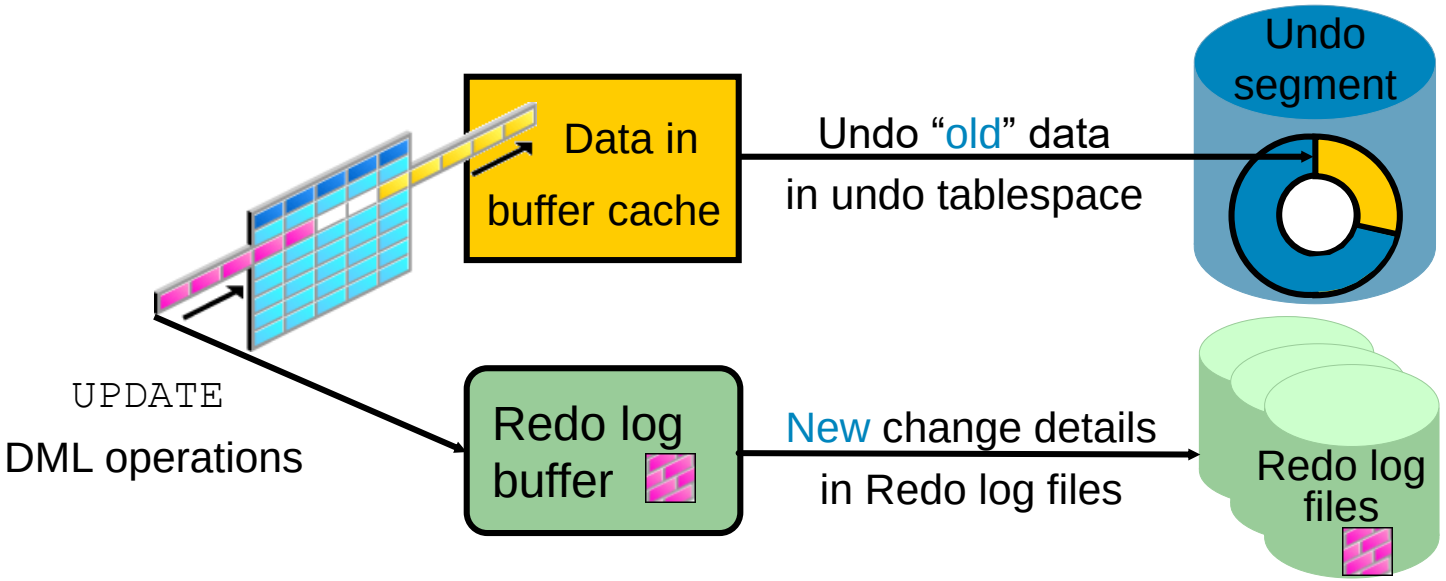
- Performs recovery at instance startup
- Cleans up unused temporary segments



System-Monitor-Prozess (SMON)

- **Recovery beim Hochfahren**
  - Falls erforderlich
  - Außerdem temporäre Segmente entfernen, die nicht mehr benötigt werden.
  - Falls abgeschlossene Transaktionen in der Recovery übergangen werden mussten, weil es Lesefehler bei Dateien gab oder Dateien nicht verfügbar waren, werden sie wiederhergestellt, wenn der Tablespace oder die Datei wieder geöffnet werden.
  - SMON prüft regelmäßig, ob diese Verarbeitung benötigt wird. Andere Prozesse dürfen SMON aufrufen, wenn sie es für nötig halten.

Transactions and Undo Data



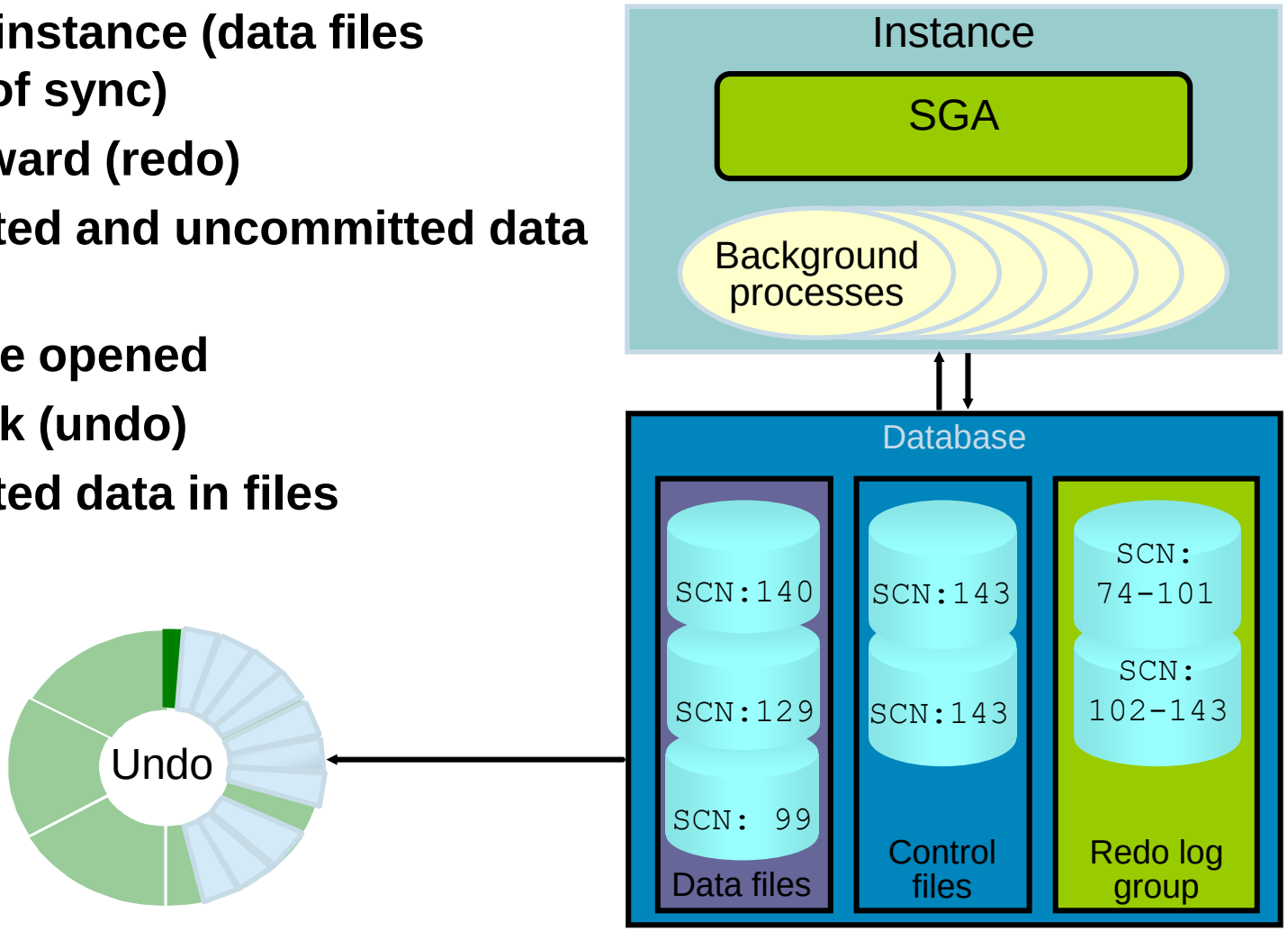
- Each transaction is assigned to only one undo segment.
- An undo segment can service more than one transaction at a time.

Transaktionen und Undo-Information

- **Undo-Segment**
  - Wird einer Transaktion zugeordnet, wenn sie beginnt.
    - Die Zuordnung kann im Dynamic Performance View `V$TRANSACTION` nachgesehen werden.
  - Besteht wie alle Segmente aus Extents, die ihrerseits aus Blöcken bestehen
  - Kann wachsen und schrumpfen je nach Bedarf
  - Verhält sich wie ein zyklischer Puffer
  - Während die Transaktion läuft, wird vor einer Änderung von Daten der Originalzustand in das Undo-Segment kopiert.
- **Trick:**
  - Undo-Segmente sind Segmente!
  - Änderungen an ihnen erzeugen ebenfalls Redo-Einträge.
  - D.h. Redo nach einem Ausfall stellt auch die Undo-Segmente wieder her.
  - Können danach für Rollback genutzt werden.

Phases of Instance Recovery

1. Startup instance (data files are out of sync)
2. Roll forward (redo)
3. Committed and uncommitted data in files
4. Database opened
5. Roll back (undo)
6. Committed data in files



Recovery-Phasen

- **Beim Öffnen von Dateien:**
  - SCN im Header mit SCN in der Steuerungsdatei vergleichen.
  - Bei Abweichung Redo-Daten anwenden.
  - Nachdem alle Dateien aktualisiert wurden, wird die Datenbank geöffnet und die Benutzer können sich wieder anmelden.
- **Anmerkung:**
  - Anwendung des Redo-Logs versetzt *alle* Transaktionen wieder in ihren Zustand zum Zeitpunkt des Fehlers – auch die noch nicht abgeschlossenen.
  - Nach dem Öffnen der Datenbank werden diese noch nicht abgeschlossenen Transaktionen zurückgesetzt (Undo).
  - Am Ende dieser Rollback-Phase enthalten die Dateien nur noch Daten von abgeschlossenen Transaktionen ("committed data").