LMU LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH
DEPARTMENT INSTITUTE FOR INFORMATICS
DATABASE SYSTEMS GROUP

Skript zur Vorlesung:

Datenbanksysteme II

Sommersemester 2013

Kapitel 4
Recovery

Vorlesung: PD Dr. Peer Kröger

http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_II

Dieses Skript basiert im Wesentlichen auf den Skripten zur Vorlesung Datenbanksysteme II an der LMU München von

Prof. Dr. Christian Böhm (SoSe 2007),
PD Dr. Peer Kröger (SoSe 2008) und
PD Dr. Matthias Schubert (SoSe 2009)

4. Recovery

4. Recovery

4.1 Einleitung

Fehler- und Recovery-Arten

- Transaktions-Recovery
  - ***Transaktionsfehler***: Lokaler Fehler einer noch nicht festgeschriebenen TA, z.B. durch
    - Fehler im Anwendungsprogramm
    - Expliziter Abbruch der TA durch den Benutzer (ROLLBACK)
    - Verletzung von Integritätsbedingungen oder Zugriffsrechten
    - Rücksetzung aufgrund von Synchronisationskonflikten
  - Behandlung durch ***Rücksetzen***
    - *Lokales UNDO*: der ursprüngliche DB-Zustand wie zu BOT wird wiederhergestellt, d.h. Rücksetzen aller Aktionen, die diese TA ausgeführt hat
    - Transaktionsfehler treten relativ häufig auf
      → Behebung innerhalb von Millisekunden notwendig

Fehler- und Recovery-Arten (cont.)

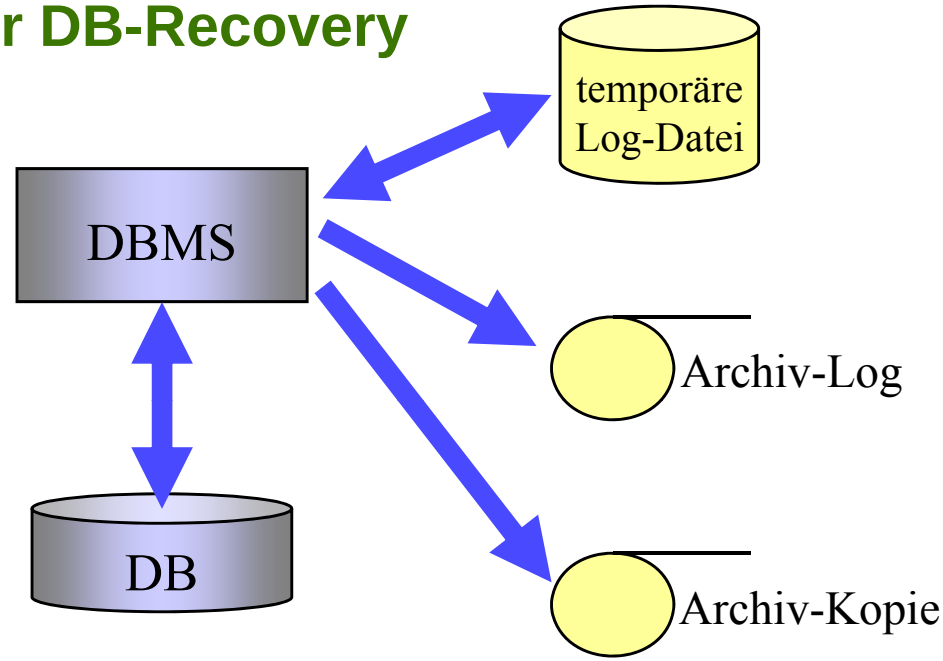
- Crash Recovery
  - ***Systemfehler***: Fehler mit Hauptspeicherverlust, d.h. permanente Speicher sind *nicht* betroffen, z.B. durch
    - Stromausfall
    - Ausfall der CPU
    - Absturz des Betriebssystems, …
  - Behandlung durch ***Crash Recovery*** (Warmstart)
    - *Globales UNDO*: Rücksetzen aller noch nicht abgeschlossenen TAs, die bereits in die DB eingebracht wurden
    - *Globales REDO*: Nachführen aller bereits abgeschlossenen TAs, die noch nicht in die DB eingebracht wurden
    - Systemfehler treten i.d.R. im Intervall von Tagen auf
      → Recoverydauer einige Minuten

Fehler- und Recovery-Arten (cont.)

- Geräte-Recovery
  - ***Medienfehler***: Fehler mit Hintergrundspeicherverlust, d.h. Verlust von permanenten Daten, z.B. durch
    - Plattencrash
    - Brand, Wasserschaden, …
    - Fehler in Systemprogrammen, die zu einem Datenverlust führen
  - Behandlung durch ***Geräte-Recovery*** (Kaltstart)
    - Aufsetzen auf einem früheren, gesicherten DB-Zustand (Archivkopie)
    - *Globales REDO:* Nachführen aller TAs, die nach dem Erzeugen der Sicherheitskopie abgeschlossenen wurden
    - Medienfehler treten eher selten auf (mehrere Jahre)
      → Recoverydauer einige Stunden / Tage
    - Wichtig: regelmäßige Sicherungskopien der DB notwendig

4.1 Einleitung

Systemkomponenten der DB-Recovery



- Behandlung von Transaktions- und Systemfehlern

  DB + temporäre Log-Datei → DB

- Behandlung von Medienfehlern

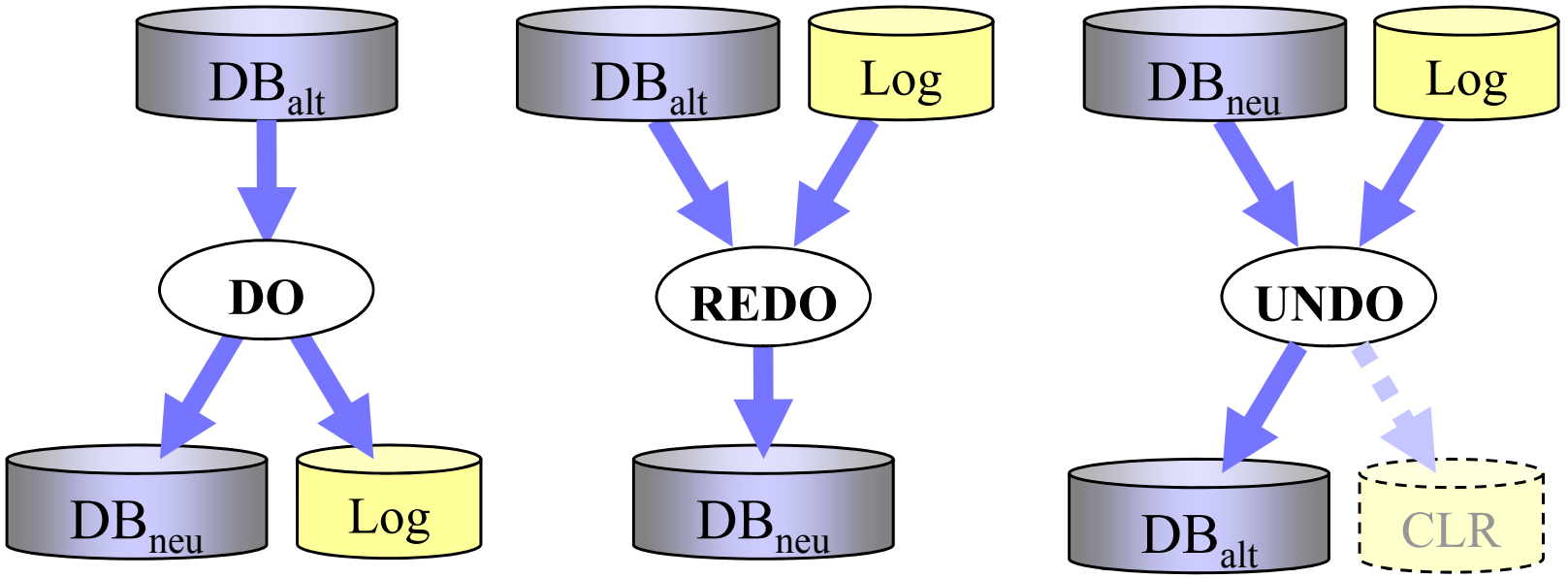
  Archiv-Kopie + Archiv-Log → DB

4. Recovery

4.2 Logging-Techniken

Aufgaben des Logging

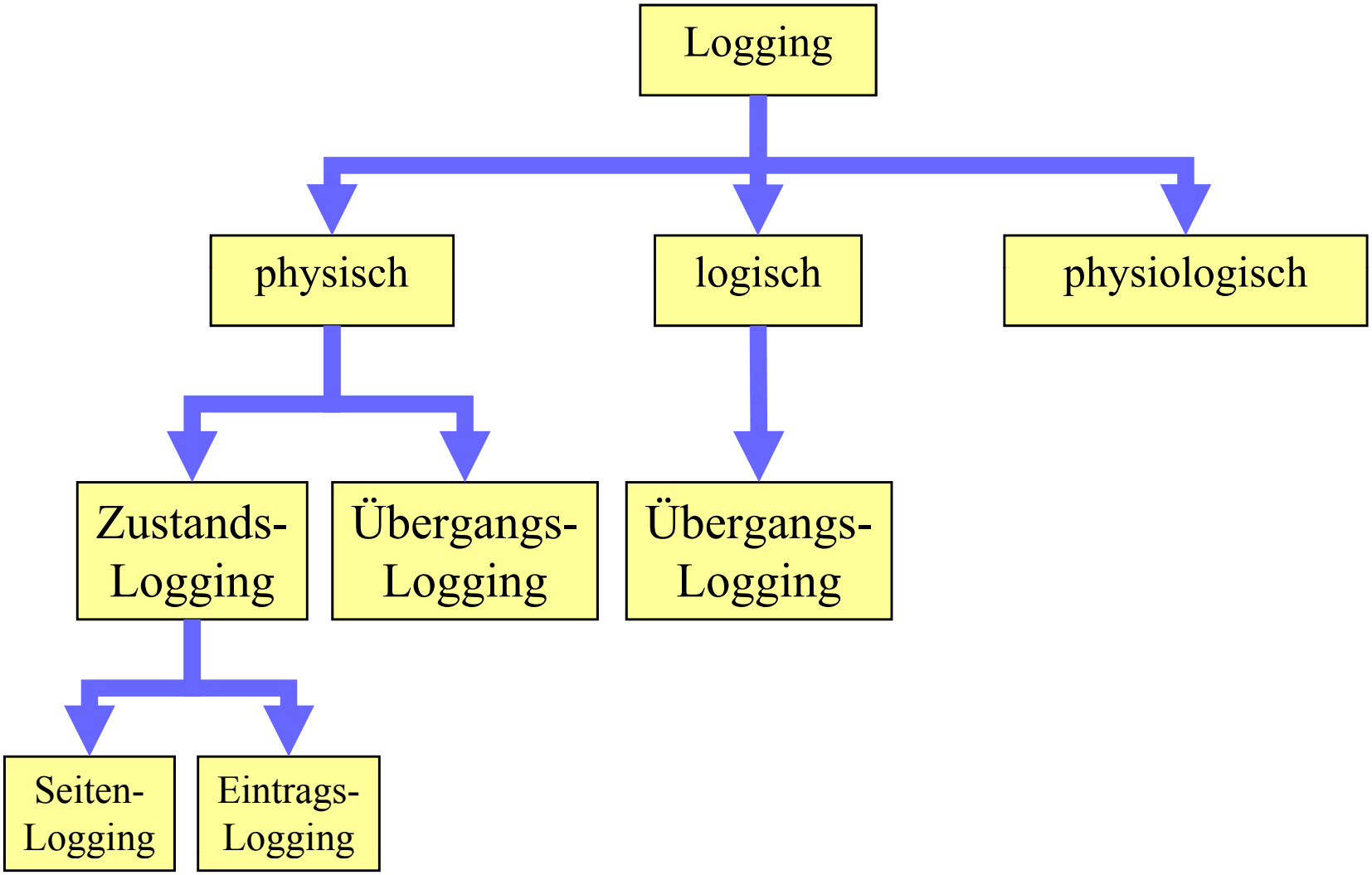
- Für jede Änderungsoperation auf der Datenbank im Normalbetrieb (**DO**) benötigt man Protokolleinträge für
  - **REDO**: Information zum Nachvollziehen der Änderungen erfolgreicher TAs
  - **UNDO**: Information zum Zurücknehmen der Änderungen unvollständiger TAs



CLR = Compensation Log Record (zur Behandlung von Fehlern während der Recovery)

4.2 Logging-Techniken

Klassifikation von Logging-Verfahren



4.2 Logging-Techniken

Physisches Logging

- Protokoll auf der Ebene der physischen Objekte (Seiten, Datensätze, Indexeinträge)

- Zustandslogging
  - Protokollierung der Werte vor und nach jeder Änderung: Alte Zustände *BFIM* (Before-Images) und neue Zustände *AFIM* (After-Images) der geänderten Objekte werden in die Log-Datei geschrieben

- Übergangslogging
  - Protokollierung der Zustandsdifferenz zwischen BFIM und AFIM

4.2 Logging-Techniken

- Zustandslogging auf Seitenebene
  - vollständige Kopien von Seiten werden protokolliert
  - Recovery sehr einfach und schnell (Seiten einfach zurückkopieren)
  - sehr großer Logumfang und hohe I/O-Kosten auch bei nur kleinen Änderungen
  - Seitenlogging impliziert Seitensperren → hohe Konfliktrate bei Synchronisation
- Zustandslogging auf Eintragsebene
  - statt ganzer Seiten werden nur tatsächlich geänderte Einträge protokolliert
  - kleinere Sperrgranulate als Seiten möglich
  - Protokollgröße reduziert sich typischerweise um mind. 1 Größenordnung
  - Log-Einträge werden in Puffer gesammelt → wesentlich weniger Plattenzugriffe
  - Recovery ist aufwändiger: zu ändernde Datenbankseiten müssen vollständig in den Hauptspeicher geladen werden, um die Log-Einträge anwenden zu können

4.2 Logging-Techniken

- Übergangslogging
  - Protokollierung der Zustandsdifferenz zwischen *BFIM* und *AFIM*
  - Aus *BFIM* muss *AFIM* berechenbar sein (u.u.)
  - Realisierbar durch XOR-Operation $\oplus$ (eXclusive-OR)[1]:

	Zustands-Logging	**Übergangs-Logging**
DO Änderung $A_{alt} \rightarrow A_{neu}$	Protokollierung von $BFIM = A_{alt}$, $AFIM = A_{neu}$	Protokollierung von $D = A_{alt} \oplus A_{neu}$
REDO (in DB liegt A_{alt})	Überschreibe A_{alt} mit *AFIM*	$A_{neu} = A_{alt} \oplus D$
UNDO (in DB liegt A_{neu})	Überschreibe A_{neu} mit *BFIM*	$A_{alt} = A_{neu} \oplus D$

[1] XOR-Operation:

XOR:
$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

4.2 Logging-Techniken

Logisches Logging

- Spezielle Form des Übergangs-Logging: Protokollierung von Änderungsoperationen mit ihren aktuellen Parametern
- Protokoll auf hoher Abstraktionsebene ermöglicht kurze Log-Einträge
- Probleme für **REDO**: Änderungen umfassen typischerweise mehrere Seiten (Tabelle, Indexe)
  - Atomares Einbringen der Mehrfachänderungen schwierig
  - Logische Änderungen sind aufwändiger durchzuführen als physische Änderungen
- Probleme für **UNDO**: Mengenorientierte Änderungen können sehr aufwändige Protokolleinträge verursachen:
  - Bsp.: `DELETE FROM Products WHERE Group = 'G1'`
    => **UNDO** erfordert viele Einfügungen, falls Produktgruppe G1 umfangreich ist
  - Bsp.: `UPDATE Products SET Group = 'G2' WHERE Group = 'G1'`
    => **UNDO** muss alte und neue Produkte der Gruppe G2 unterscheiden

4.2 Logging-Techniken

Physiologisches Logging

- Kombination von physischem und logischem Logging: Protokollierung von elementaren Operationen innerhalb einer Seite
  - Physical-to-a-page
    - Protokollierungseinheiten sind geänderte Seiten
    - gut verträglich mit Pufferverwaltung und direktem (atomarem) Einbringen
  - Logical-within-a-page
    - logische Protokollierung der Änderungen auf einer Seite
- Bewertung
  - Log-Einträge beziehen sich nicht auf mehrere Seiten wie bei logischem Logging
  - Dadurch einfachere Recovery als bei logischem Logging
  - Log-Datei ist länger als bei logischem Logging aber kürzer als bei physischem Logging
  - Flexibler als physisches Logging wegen variabler Objektpositionen auf Seiten.

4.2 Logging-Techniken

Die Log-Datei

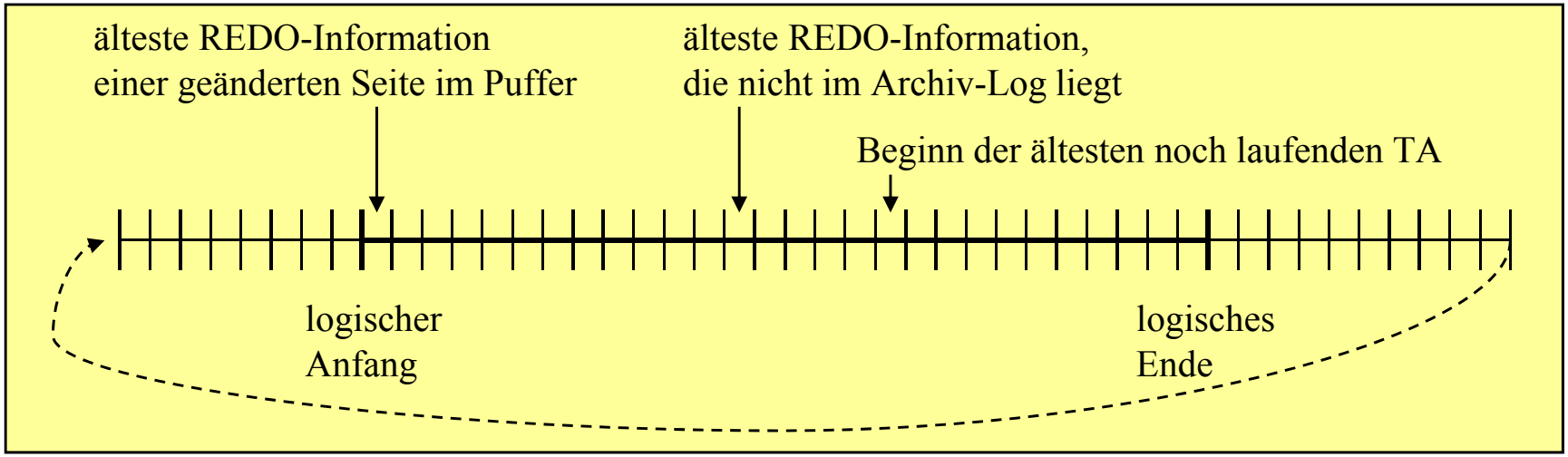
- Art der Protokolleinträge
  - Beginn, Commit und Rollback von Transaktionen
  - Änderungen des DB-Zustandes durch Transaktionen
  - Sicherungspunkte (Checkpoints)
- Komponenten von Änderungseinträgen: (LSN, TA-Id, Page-Id, REDO, UNDO, PrevLSN)
  - LSN (Log Sequence Number): eindeutige Kennung des Log-Eintrags in chronologischer Reihenfolge
  - TA-Id: eindeutige Kennung der TA, die die Änderung durchgeführt hat
  - Page-Id : Kennung der Seite auf der die Änderungsoperation vollzogen wurde (ein Eintrag pro geänderter Seite)
  - REDO: gibt an, wie die Änderung nachvollzogen werden kann
  - UNDO: beschreibt, wie die Änderung rückgängig gemacht werden kann
  - PrevLSN: Zeiger auf vorhergehenden Log-Eintrag der jeweiligen TA (Effizienzgründe)

4.2 Logging-Techniken

- Die Log-Datei ist eine ***sequentielle*** Datei: Schreiben neuer Protokolldaten an das aktuelle Dateiende

  => Ringpuffer-Organisation

  - Log-Daten sind für ***Crash-Recovery*** nur begrenzte Zeit relevant:
    - **UNDO**-Sätze für erfolgreich beendetete TAs werden nicht mehr benötigt
    - Nach Einbringen der Seite in die DB wird REDO-Information nicht mehr benötigt



- **REDO**-Information für ***Geräte-Recovery*** ist im Archiv-Log zu sammeln

4.2 Logging-Techniken

- Beispiel

Ablauf T_1	**Ablauf T_2**	**Log-Eintrag** *(LSN, TA-Id, Page-Id, REDO, UNDO, PrevLSN)*
begin		(#1, T_1, begin, 0)
read(A, a_1)		
	begin	(#2, T_2, begin, 0)
	read(C, c_2) //80	
$a_1 := a_1 - 50$		
write(A, a_1)		(#3, T_1, p_A, A-=50, A+=50, #1)
	$c_2 := 100$	
	write(C, c_2)	(#4, T_2, p_C, C=100, C=80, #2)
read(B, b_1) //70		
$b_1 := 50$		
write(B, b_1)		(#5, T_1, p_B, B=50, B=70, #3)
commit		(#6, T_1, commit, #5)
	read(A, a_2)	
	$a_2 := a_2 - 100$	
	write(A, a_2)	(#7, T_2, p_A, A-=100, A+=100, #4)
	commit	(#8, T_2, commit, #7)

(hier: logisches Logging)