Einsatz und Realisierung von Datenbanksystemen

ERDB Übungsleitung

Maximilian {Bandle, Schüle}, Alice Rey, Michael Jungmair

i3erdb@in.tum.de

Folien erstellt von Maximilian Bandle & Alexander Beischl

Organisatorisches
Disclaimer

Die Folien werden von der Übungsleitung allen Tutoren zur Verfügung gestellt.

Sollte es Unstimmigkeiten zu den Vorlesungsfolien von Prof. Kemper geben, so sind die Folien aus der Vorlesung ausschlaggebend.

Falls Ihr einen Fehler oder eine Unstimmigkeit findet, schreibt an i3erdb@in.tum.de mit Angabe der Foliennummer.

Organisatorisches
Bonussystem

- Anwesenheit/Feiertag: +1P
- Abwesenheit: 0P
- Aufgabe vorstellen: +1P

- bei 12+2 = 14P: 0.3 Notenbonus
  (nur auf eine bestandene Klausur)

Organisatorisches
Bei Fragen

- Mail bitte zuerst an Ihren Tutor
- Wenn weiterhin fragen sind mit Tutor in Kopie an: i3erdb@in.tum.de

Organisatorisches
Beginn

Recovery

Recovery

Motivation

- Eine der wichtigsten Aufgaben eines DBMS ist das Verhindern von Datenverlusten durch Systemabstürze

- Die zwei wichtigsten Recovery-Mechanismen sind:
  - Sicherungspunkte (=Backups)
  - Log-Dateien

Recovery

Motivation

- Ein Sicherungspunkt ist ein Schnappschuss des Datenbankinhalts
- Log-Dateien protokollieren alle Änderungen der Datenbasis mit
- Log-Dateien und Backups sollten nie auf der gleichen Maschine gespeichert werden

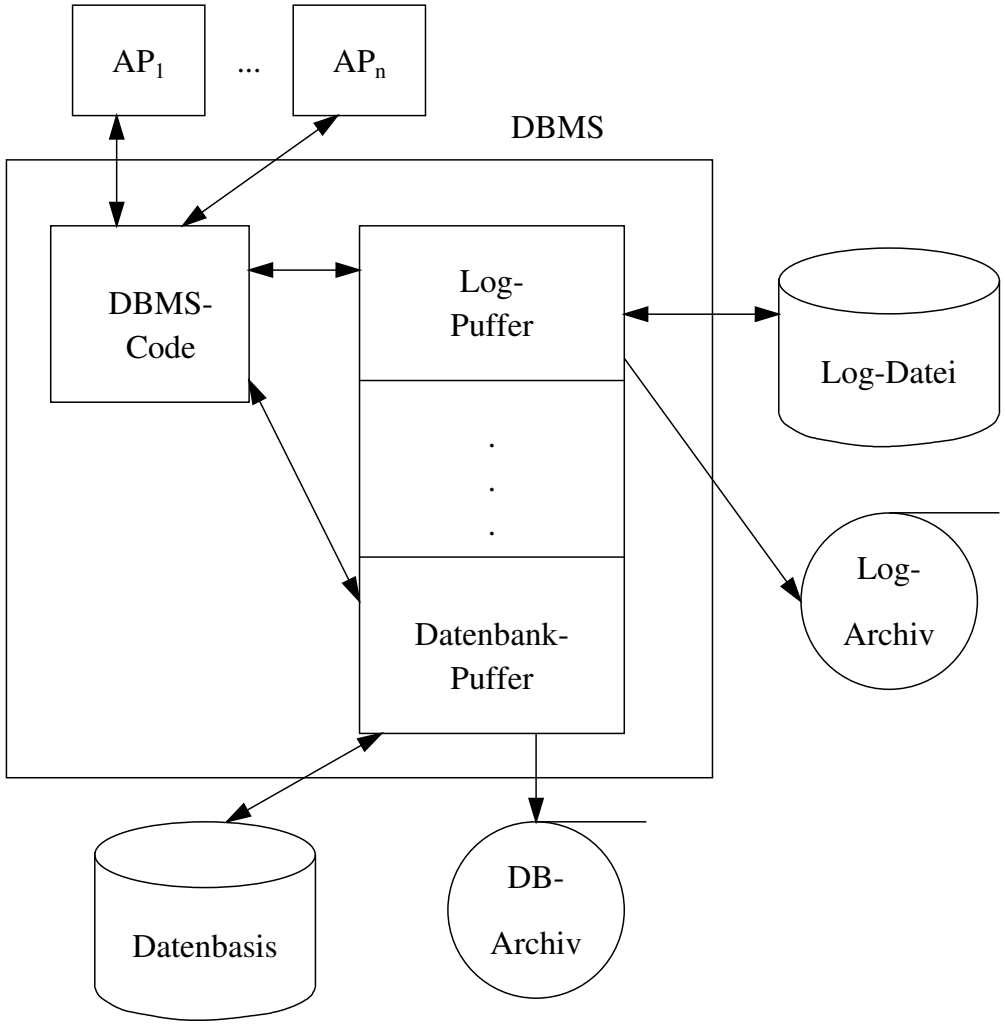
Recovery

Write Ahead Logging

- Write Ahead Log:
  1. Bevor eine Transaktion commited, müssen alle zugehörigen Log-Einträge ausgeschrieben werden (Fehlererkennung)
  2. Vor Auslagern einer modifizierten Seite, müssen alle zugehörigen Log-Einträge in das temporäre und das Log-Archiv geschrieben werden (Fehleraufhebung)

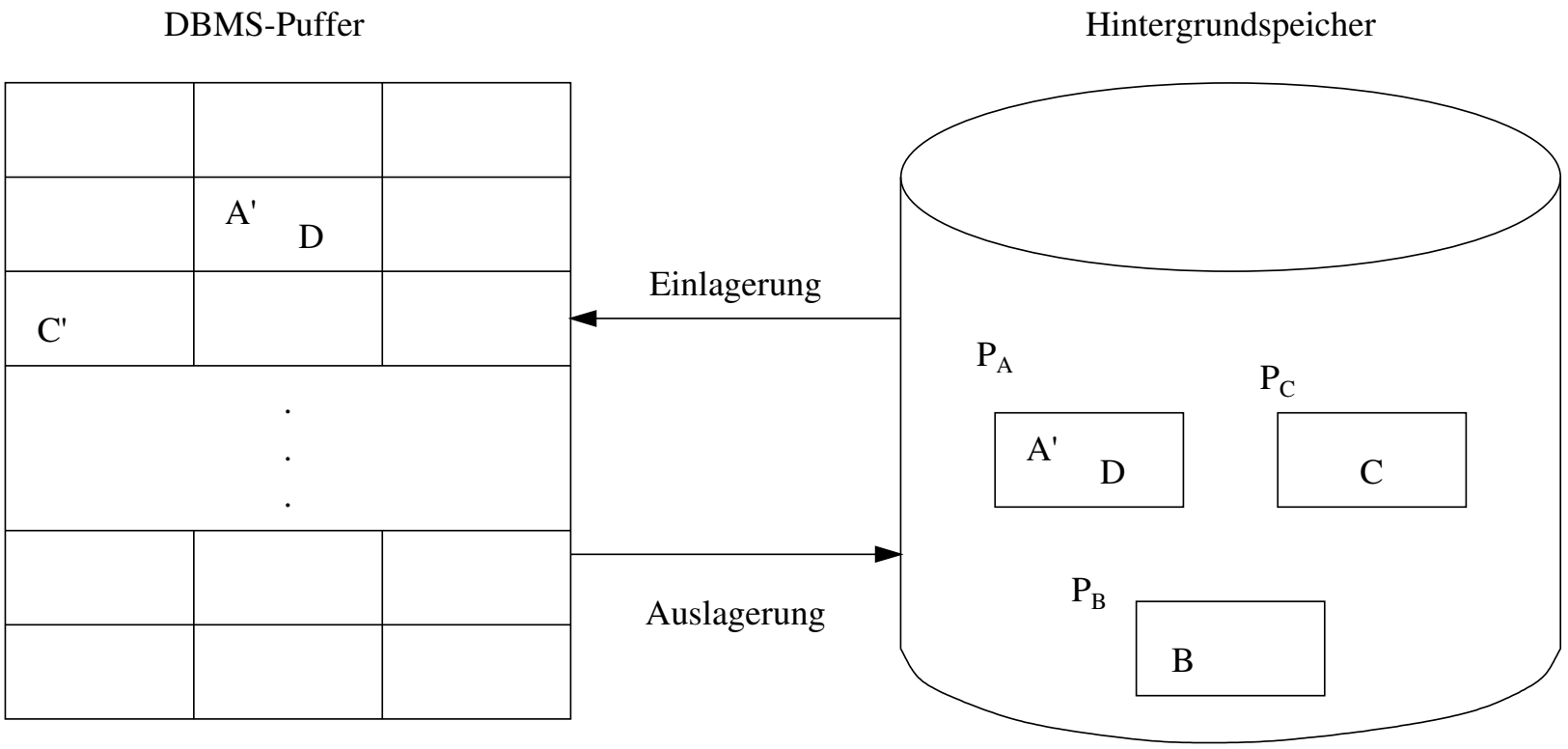
Recovery

Struktur



Recovery

Speicherhierarchie



Recovery

Speicherhierarchie

Ersetzung von Puffer-Seiten

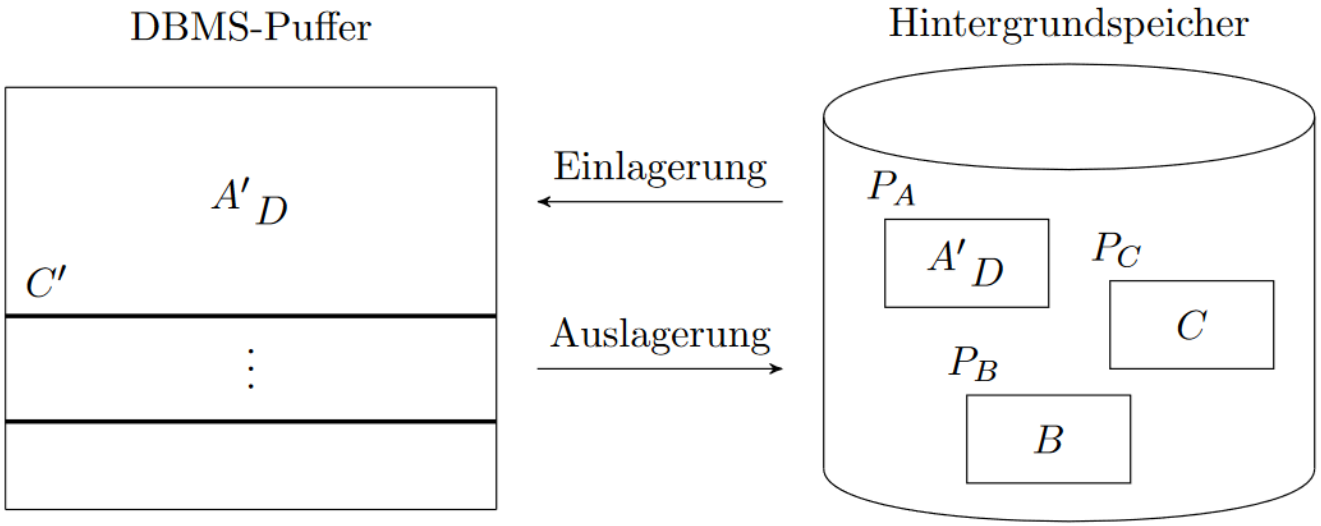
- **¬Steal:** Seiten die noch von einer Transaktion modifiziert werden müssen im Speicher verbleiben.
- **Steal:** Seiten können (fast) immer aus dem Puffer in den Speicher eingelagert werden.

Einbringen von Änderungen abgeschlossener Transaktionen

- **Force:** Änderungen werden direkt nach Durchführung gespeichert
- **¬Force:** Änderungen können im Puffer-Speicher verbleiben

Aufgabe 1

Demonstrieren Sie anhand eines Beispiels, dass man die Strategien *force* und ¬*steal* nicht kombinieren kann, wenn parallele Transaktionen gleichzeitig Änderungen an Datenobjekten innerhalb einer Seite durchführen. Betrachten Sie dazu z.B. die unten dargestellte Seitenbelegung, bei der die Seite P_A die beiden Datensätze A und D enthält. Entwerfen Sie eine verzahnte Ausführung zweier Transaktionen, bei der eine Kombination aus *force* und ¬*steal* ausgeschlossen ist.



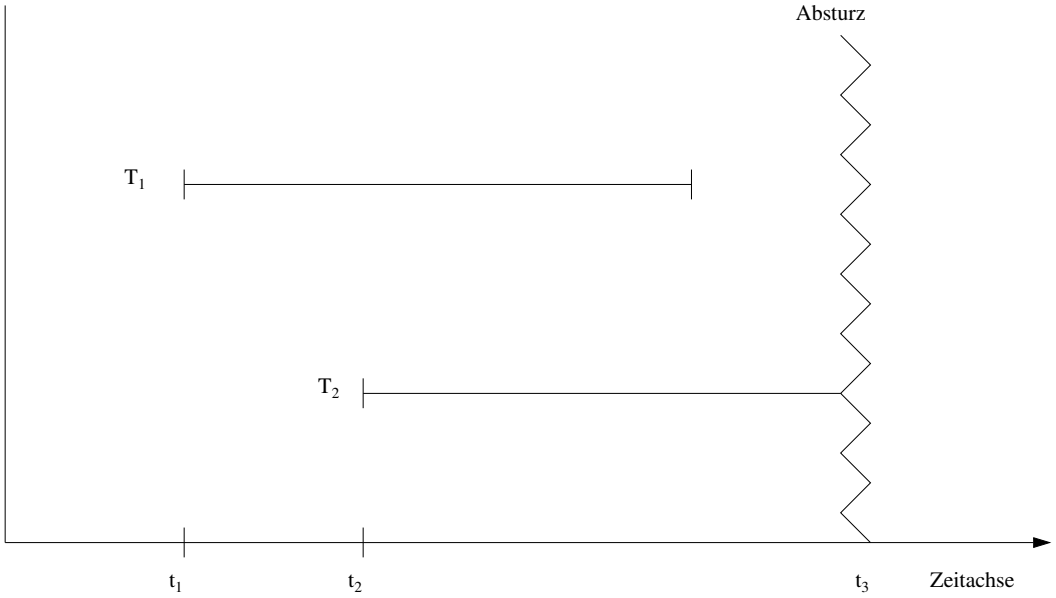
Recovery

ARIES-Protokoll

- ARIES-Protokoll ist ein weit verbreitetes Protokoll zur Fehlerbehebung (für DBMS)
- Eine Log-Datei enthält:
  - **Redo**: alle Änderungen können nachvollzogen werden
  - **Undo**: alle Änderungen können rückgängig gemacht werden

Recovery

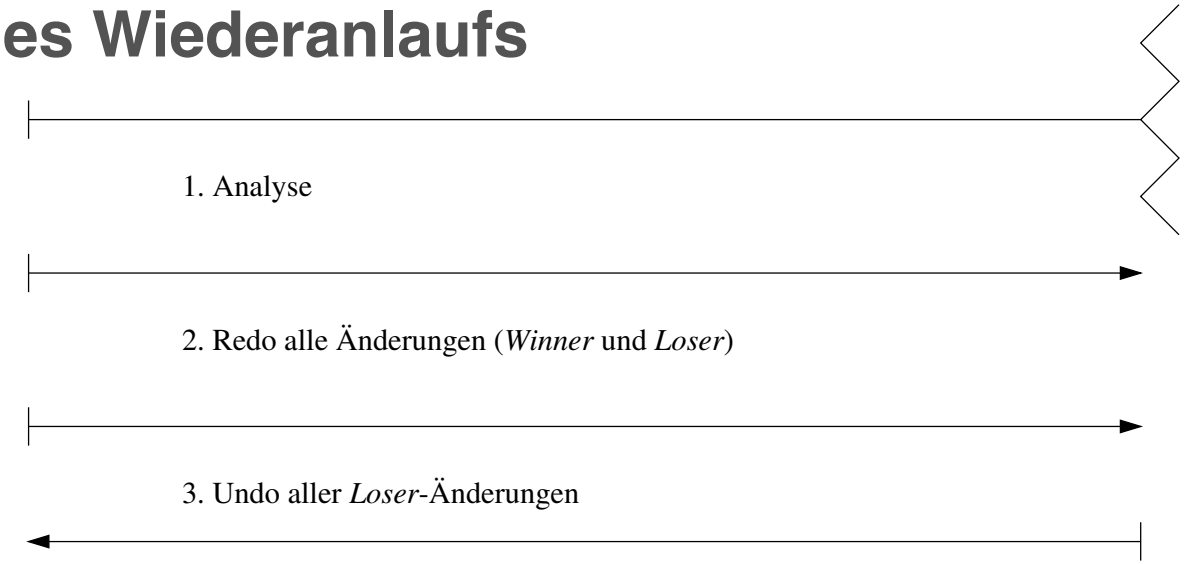
Wiederanlauf nach einem Fehler



- TAs der Art T_1 sind **Winner**: müssen vollständig nachvollzogen werden
- TAs der Art T_2 sind **Loser**: müssen rückgängig gemacht werden

Recovery

Phasen des Wiederanlaufs



1. Analyse:

   Ermitteln der Winner- & Loser-Transaktionen

2. Wiederholung der Historie

   Alle protokollierten Redo-Logs werden in der richtigen Reihenfolge ausgeführt

   => Datenbankzustand während des Absturzes

3. Undo der Loser

   Alle uncomitted TAs werden abgebrochen & ihre Auswirkungen auf die Datenbasis aufgehoben

Aufgabe 2

Überlegen Sie sich, bei welcher Seitenersetzungsstrategie bei einem Wiederanlauf eine Redo- bzw. Eine Undo-Phase notwendig ist. Verwenden Sie in diesem Zusammenhang den Begriff dirty.

Welche der beiden Phasen entfällt bei einer Hauptspeicherdatenbank?

Recovery

Speicherhierarchie - Auswirkungen auf Recovery

	force	¬force
¬steal	kein Undo kein Redo	kein Undo Redo
steal	Undo kein Redo	Undo Redo

Recovery

Speicherhierarchie - Auswirkungen auf Recovery

¬steal & force

– wird eine Seite von 2 TA geändert, so kann die 1. nicht comitten

¬steal & ¬force

– Seiten können nach Transaktionsende ersetzt werden, ohne das die Änderungen in die DB übernommen werden

steal & force

– direktes Einlager beim commit ist teuer, gesammeltes Einlagern ist günstiger

– nicht commited Daten können festgeschrieben werden

steal & ¬force

– Änderungen von (aktiven) TAs können beim Systemabsturz verloren gehen

– nicht commited Daten können festgeschrieben werden

Recovery

ARIES-Protokoll

- ARIES-Protokoll ist ein weit verbreitetes Protokoll zur Fehlerbehebung (für DBMS)

- Eine Log-Datei enthält:
  - **Redo**: alle Änderungen können nachvollzogen werden
  - **Undo**: alle Änderungen können rückgängig gemacht werden

- Es kann logisch oder physisch protokolliert werden:
  - **Logisch**: alle Operationen (“Rechenweg”) werden gesichert
  - **Physisch**: alle Ergebnisse werden gesichert

Recovery

Struktur der Log-Einträge

[LSN, TA, PageID, **Redo**, **Undo**, PrevLSN]

Redo

- Physische Protokollierung: After-Image
- Logische Protokollierung: Code mit dem aus dem Before-Image das After-Image erzeugt werden kann

Undo

- Physische Protokollierung: Before-Image
- Logische Protokollierung: Code mit dem aus dem After-Image das Before-Image erzeugt werden kann

Recovery

Struktur der Log-Einträge

[**LSN**, **TA**, PageID, Redo, Undo, PrevLSN]

LSN (Log Sequence Number)

- Eine eindeutige Kennung des Log-Eintrags
- LSNs müssen monoton aufsteigend vergeben werden
- Die chronologische Reihenfolge der Protokolleinträge kann dadurch ermittelt werden

TA

- Transaktionskennung der Transaktion, die die Änderung durchgeführt hat

Recovery

Struktur der Log-Einträge

[LSN, TA, **PageID**, Redo, Undo, **PrevLSN**]

PageID

- Die Kennung der Seite, auf der die Änderungsoperation vollzogen wurde
- Wenn eine Änderung mehr als eine Seite betrifft, müssen entsprechend
- Viele Log-Einträge generiert werden

PrevLSN

- Zeiger auf den vorhergehenden Log-Eintrag der jeweiligen Transaktion
- Diesen Eintrag benötigt man aus Effizienzgründen

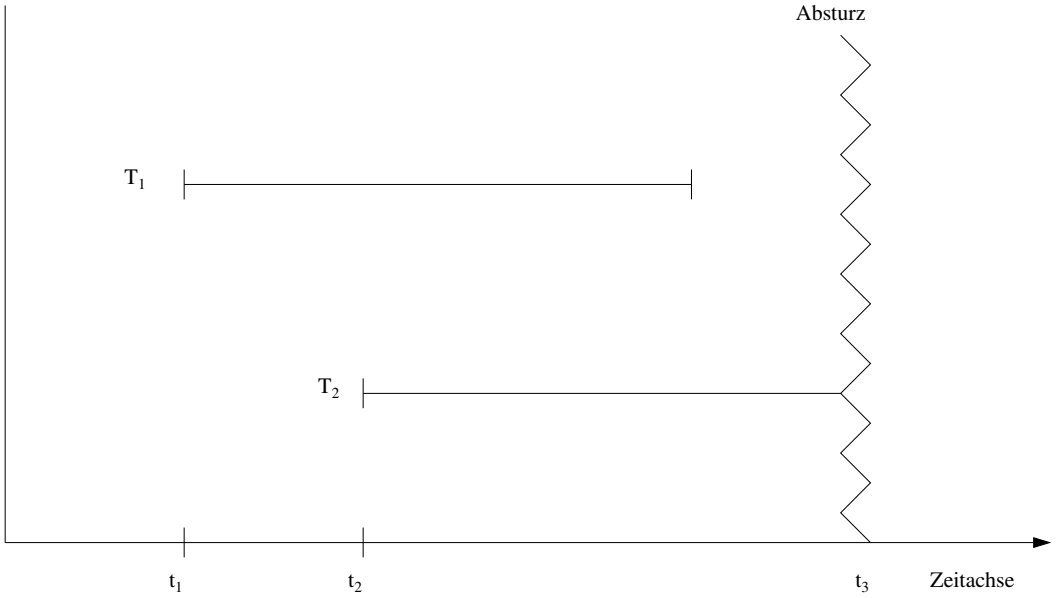
Aufgabe 3

In Abbildung 1 ist die verzahnte Ausführung der beiden Transaktionen T1 und T2 und das zugehörige Log auf der Basis logischer Protokollierung gezeigt. Wie sähe das Log bei physischer Protokollierung aus, wenn die Datenobjekte A, B und C die Initialwerte 1000, 2000 und 3000 hätten?

Schritt	T_1	T_2	Log [LSN,TA,PageID,Redo,Undo,PrevLSN]
1.	**BOT**		$[\#1, T_1, \textbf{BOT}, 0]$
2.	$r(A, a_1)$		
3.		**BOT**	$[\#2, T_2, \textbf{BOT}, 0]$
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		$[\#3, T_1, P_A, A\text{–=}50, A\text{+=}50, \#1]$
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	$[\#4, T_2, P_C, C\text{+=}100, C\text{–=}100, \#2]$
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		$[\#5, T_1, P_B, B\text{+=}50, B\text{–=}50, \#3]$
12.	**commit**		$[\#6, T_1, \textbf{commit}, \#5]$
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	$[\#7, T_2, P_A, A\text{–=}100, A\text{+=}100, \#4]$
16.		**commit**	$[\#8, T_2, \textbf{commit}, \#7\ ]$

Recovery

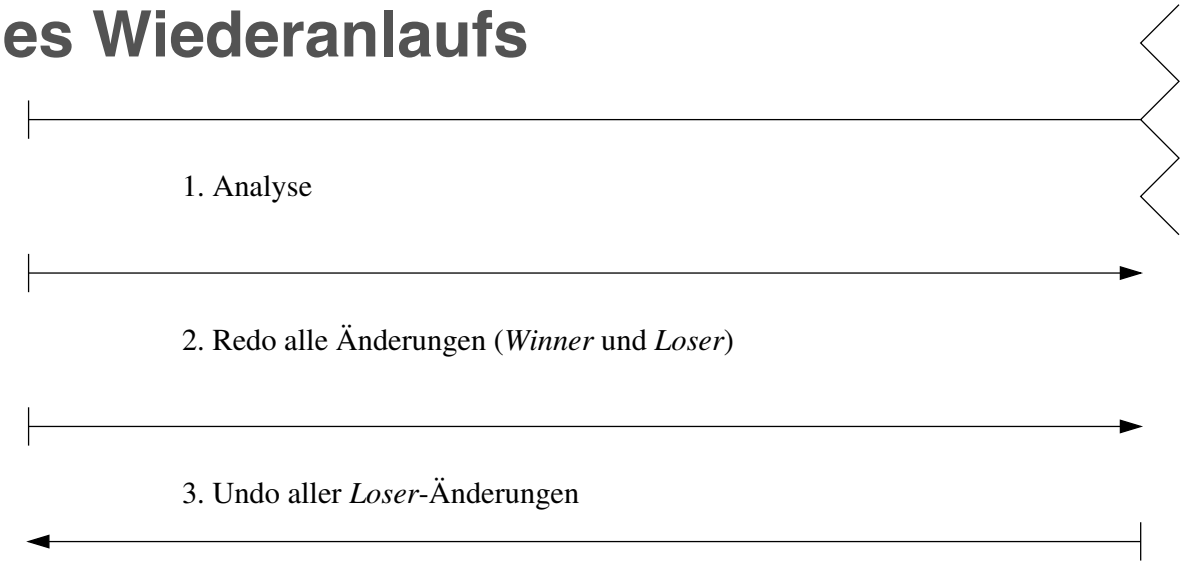
Wiederanlauf nach einem Fehler



- TAs der Art T_1 sind **Winner**: müssen vollständig nachvollzogen werden
- TAs der Art T_2 sind **Loser**: müssen rückgängig gemacht werden

Recovery

Phasen des Wiederanlaufs



1. Analyse:

   Ermitteln der Winner- & Loser-Transaktionen

2. Wiederholung der Historie

   Alle protokollierten Redo-Logs werden in der richtigen Reihenfolge ausgeführt

   => Datenbankzustand während des Absturzes

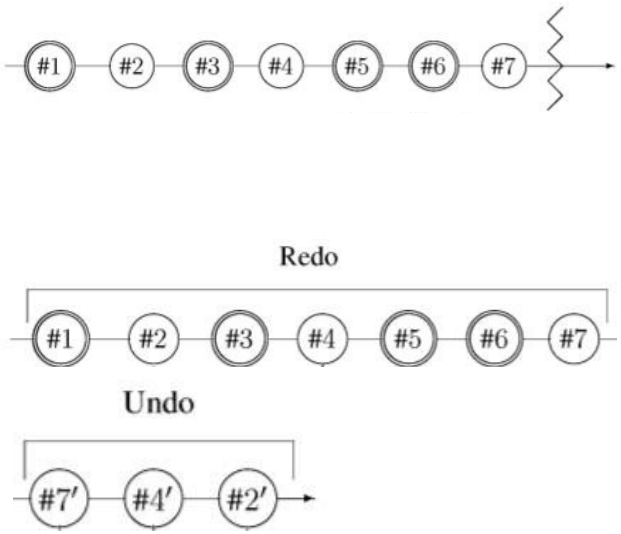
3. Undo der Loser

   Alle uncomitted TAs werden abgebrochen & ihre Auswirkungen auf die Datenbasis aufgehoben

Recovery

Wiederanlauf nach einem Fehler

Die DB stürzt bei LSN #7 ab. Wie sieht die Log-Datei bei logischer Protokollierung nach der Durchführung des Wiederanlaufs aus?

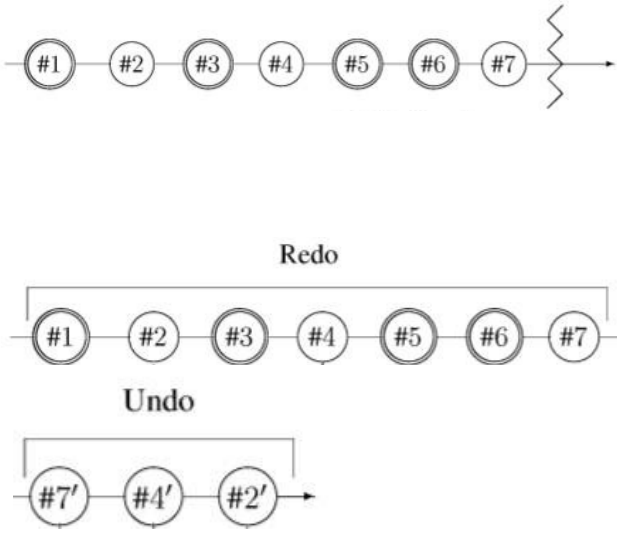


Schritt	T_1	T_2	Log [LSN,TA,PageID,Redo,Undo,PrevLSN]
1.	**BOT**		[#1, T_1, **BOT**, 0]
2.	$r(A, a_1)$		
3.		**BOT**	[#2, T_2, **BOT**, 0]
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		[#3, T_1, P_A, A–=50, A+=50, #1]
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	[#4, T_2, P_C, C+=100, C–=100, #2]
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		[#5, T_1, P_B, B+=50, B–=50, #3]
12.	**commit**		[#6, T_1, **commit**, #5]
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	[#7, T_2, P_A, A–=100, A+=100, #4]
16.		**commit**	[#8, T_2, **commit**, #7]

Recovery

Wiederanlauf nach einem Fehler

Die DB stürzt bei LSN #7 ab. Wie sieht die Log-Datei bei logischer Protokollierung nach der Durchführung des Wiederanlaufs aus?



Lösung

1. Analysephase
   - T_1 Winner
   - T_2 Loser
2. Redo alle Logs bis zum Absturz

   [#1, T_1, BOT, 0]

   ⋮

   [#7, T_2, P_A, A-=100, A+=100, #4]

   Vergleiche Log-Spalte auf der vorherigen Folie
3. Undo der Loser

   <#7', T_2, P_A, A+=100, #7, #4>

   <#4', T_2, P_C, C-=100, #7', #2>

   <#2', T_2, -, -, #4', 0>

(Vgl. Übungsbuch 10.3 mit physik. Protokollierung)

Recovery

Idempotenz

Fehlertoleranz (Idempotenz) des Wiederanlaufs ist wichtig!

DBMS kann auch **während der Recoveryphase** abstürzen!

Formaler muss gelten:

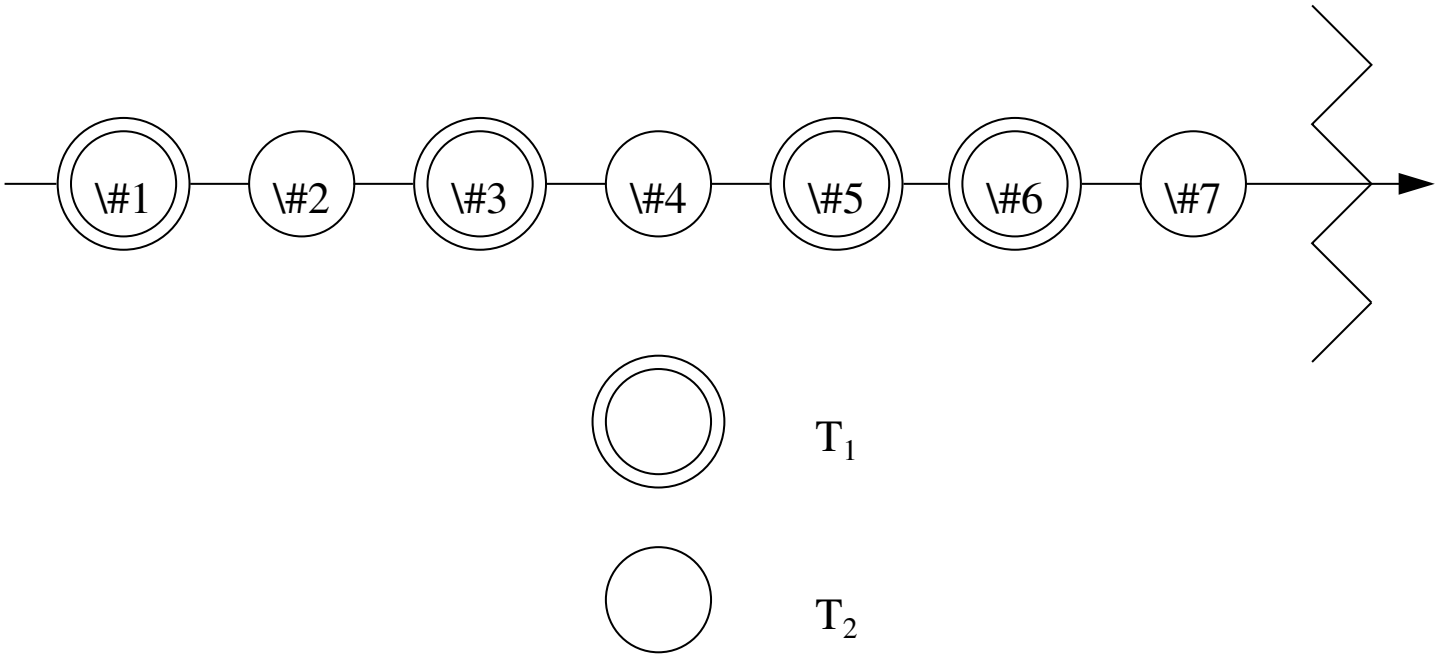
undo(undo(…(undo(x))…)) = undo(x)

redo(redo(…(redo(x))…)) = redo(x)

Recovery

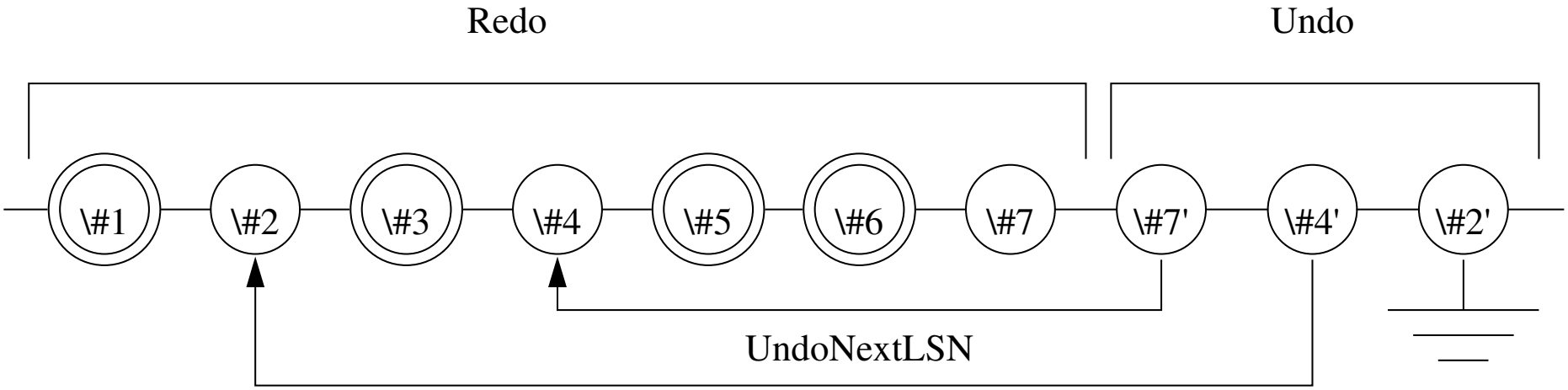
Fehlertoleranz Wiederanlauf

undo(undo(…(undo(a))…)) = undo(a)
redo(redo(…(redo(a))…)) = redo(a)



Recovery

Fehlertoleranz Wiederanlauf



- Kompensiationseinträge (CLR: compensation log record) für rückgängig gemachte Änderungen
- #7’ ist CLR für #7
- #4’ ist CLR für #4

Recovery

Struktur der eines CLR

<LSN, TA, PageID, Redo, PrevLSN, **UndoNextLSN**>

UndoNextLSN

- Verweis auf nächst rückgängig zu machende Änderung

Weitere Unterschiede

- **Konvention:** LSN mit Apostroph
- Spitze statt eckigen Klammern

Restliche Felder wie in Log Record

Aufgabe 4

Leider erhalten wir einen Fehler mit Hauptspeicherverlust der in Abbildung 1 gezeigten Ausführung nach Schritt 13. Welche Transaktion ist ein Winner, welche ein Loser? Geben Sie alle nötigen Kompensations-Rekorde (CLR) an.

Schritt	T_1	T_2	Log [LSN,TA,PageID,Redo,Undo,PrevLSN]
1.	**BOT**		$[\#1, T_1, \textbf{BOT}, 0]$
2.	$r(A, a_1)$		
3.		**BOT**	$[\#2, T_2, \textbf{BOT}, 0]$
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		$[\#3, T_1, P_A, A\text{–=}50, A\text{+=}50, \#1]$
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	$[\#4, T_2, P_C, C\text{+=}100, C\text{–=}100, \#2]$
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		$[\#5, T_1, P_B, B\text{+=}50, B\text{–=}50, \#3]$
12.	**commit**		$[\#6, T_1, \textbf{commit}, \#5]$
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	$[\#7, T_2, P_A, A\text{–=}100, A\text{+=}100, \#4]$
16.		**commit**	$[\#8, T_2, \textbf{commit}, \#7\ ]$

Aufgabe 5

Sie verwenden ein Datenbanksystem mit Write-Ahead-Logging und der Strategie ¬*force* und *steal*.
Die Datenbank verwaltet zwei Datenobjekte A und B mit einem Anfangswert von jeweils 1 (A=1 und B=1).
Sie starten zwei Transaktionen T_1 und T_2 zeitgleich:

Während der Ausführung stürzt Ihre Datenbank ab. Sie wissen nicht, welche Transaktionen erfolgreich ausgeführt worden sind.
Nehmen Sie an, die Datenbank erzeugt ausschließlich **serielle Historien**.
Bevor Sie die Datenbank neu starten, durchsuchen Sie die Festplatte und stellen fest, dass B dort den Wert 20 hat, A den Wert 2.
Auch nach einem Neustart mit erfolgreichem Wiederherstellungsprozess liefert die Datenbank für A den Wert 2.

Welche der Transaktionen hat erfolgreich committed *Winner*, welche nicht *Loser*?

Fragen?