

Fehlerbehandlung (Recovery)

- Fehlerarten
- Auswirkung der Speicherhierarchie
- Protokollierung von Änderungen
- Wiederanlauf nach Fehler
- (Sicherungspunkte)
- Media-Recovery



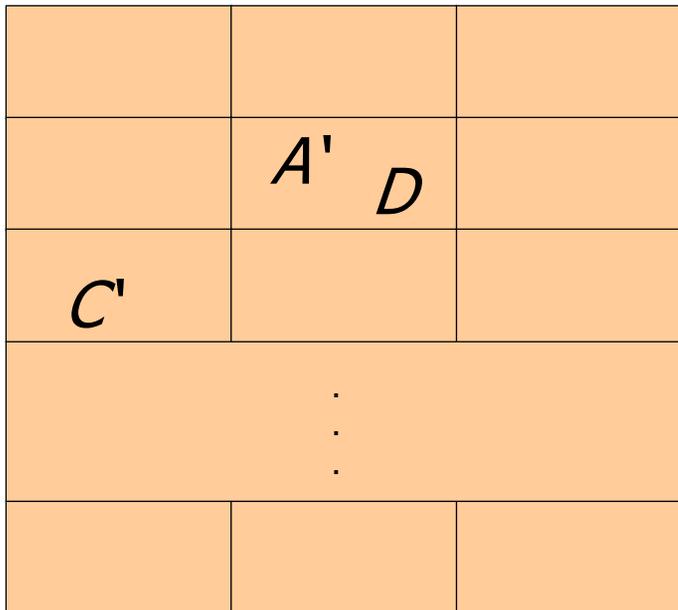
Fehlerbehandlung (Recovery)

Fehlerklassifikation

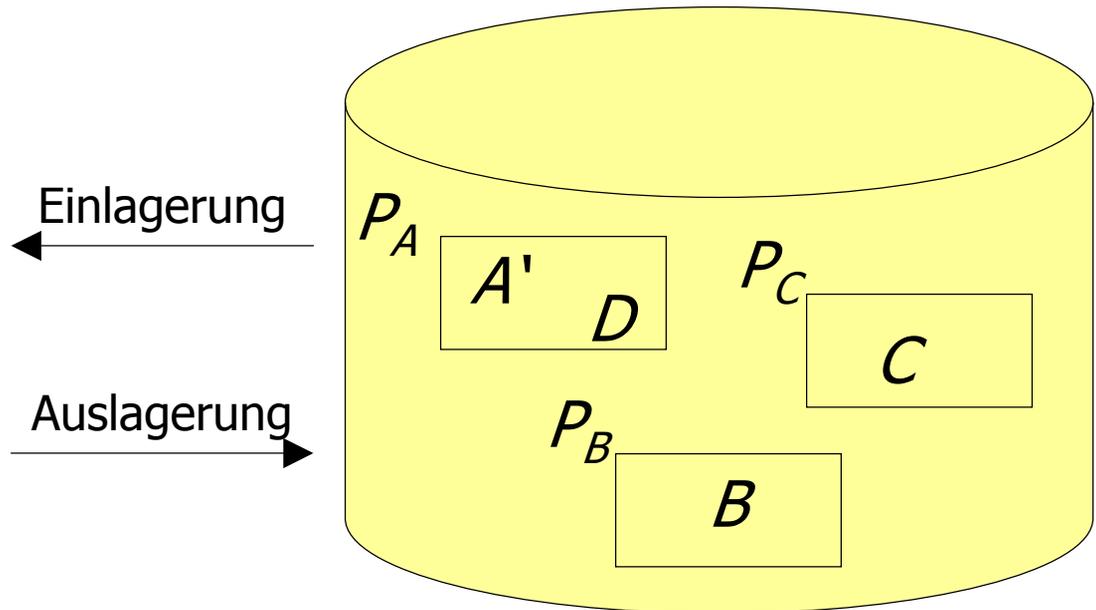
1. Lokaler Fehler in einer noch nicht festgeschriebenen (committed) Transaktion
 - Wirkung muss zurückgesetzt werden
 - *R1-Recovery*
2. Fehler mit Hauptspeicherverlust
 - Abgeschlossene TAs müssen erhalten bleiben (*R2-Recovery*)
 - Noch nicht abgeschlossene TAs müssen zurückgesetzt werden (*R3-Recovery*)
3. Fehler mit Hintergrundspeicherverlust
 - *R4-Recovery*

Zweistufige Speicherhierarchie

DBMS-Puffer



Hintergrundspeicher



Die Speicherhierarchie

Ersetzung von Puffer-Seiten

- \neg steal: Bei dieser Strategie wird die Ersetzung von Seiten, die von einer noch aktiven Transaktion modifiziert wurden, ausgeschlossen.
- steal: Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung, falls neue Seiten eingelagert werden müssen.

Einbringen von Änderungen abgeschlossener TAs

- Force-Strategie: Änderungen werden zum Transaktionsende auf den Hintergrundspeicher geschrieben.
- \neg force-Strategie: geänderte Seiten können im Puffer verbleiben.

Auswirkungen auf Recovery

	force	\neg force
\neg steal	<ul style="list-style-type: none">● kein Redo● kein Undo	<ul style="list-style-type: none">● Redo● kein Undo
steal	<ul style="list-style-type: none">● kein Redo● Undo	<ul style="list-style-type: none">● Redo● Undo

... nötig.

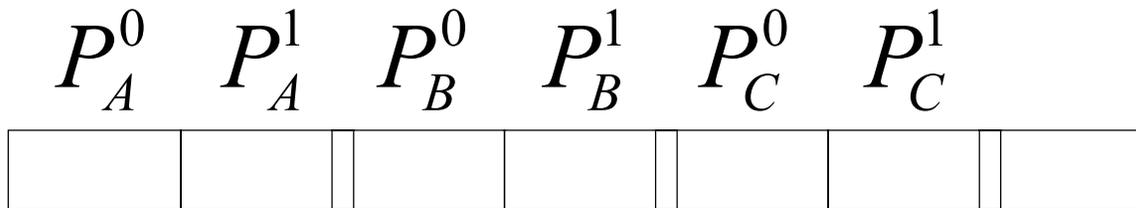
Einbringungsstrategien

Update in Place

- jede Seite hat genau eine „Heimat“ auf dem Hintergrundspeicher
- der alte Zustand der Seite wird überschrieben
- Recovery durch Log-Dateien

Twin-Block-Verfahren

- jede Seite existiert zweimal auf dem Hintergrundspeicher
- Bsp.: Anordnung von Seiten P_A , P_B , und P_C :



- wird nur noch selten genutzt

Schattenspeicherkonzept

- nur geänderte Seiten werden dupliziert
- weniger Redundanz als beim Twin-Block-Verfahren

Hier zugrunde gelegte Sytemkonfiguration

- *steal*

- „dreckige Seiten“ können in der Datenbank (auf Platte) geschrieben werden

- *¬force*

- geänderte Seiten sind möglicherweise noch nicht auf die Platte geschrieben

- *update-in-place*

- Es gibt von jeder Seite nur eine Kopie auf der Platte

- *Kleine Sperrgranulate*

- auf Satzebene
- also kann eine Seite gleichzeitig „dreckige“ Daten (einer noch nicht abgeschlossenen TA) und „committed updates“ enthalten
- das gilt sowohl für Puffer- als auch Datenbankseiten

Protokollierung von Änderungsoperationen

Struktur der Log-Einträge

[LSN, TransaktionsID, 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.
- *Transaktionskennung TA* der Transaktion, die die Änderung durchgeführt hat.
- *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.

Protokollierung von Änderungsoperationen II

Struktur der Log-Einträge II

[LSN, TransaktionsID, PageID, Redo, Undo, PrevLSN]

- Die *Redo* -Information gibt an, wie die Änderung nachvollzogen werden kann.
- Die *Undo* -Information beschreibt, wie die Änderung rückgängig gemacht werden kann.
- *PrevLSN*, einen Zeiger auf den vorhergehenden Log-Eintrag der jeweiligen Transaktion. Diesen Eintrag benötigt man aus Effizienzgründen.

Beispiel einer Log-Datei

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]

Logische oder physische Protokollierung

Physische Protokollierung

Es werden Inhalte / Zustände protokolliert:

1. **before-image** enthält den Zustand vor Ausführung der Operation
2. **after-image** enthält den Zustand nach Ausführung der Operation

Logische Protokollierung

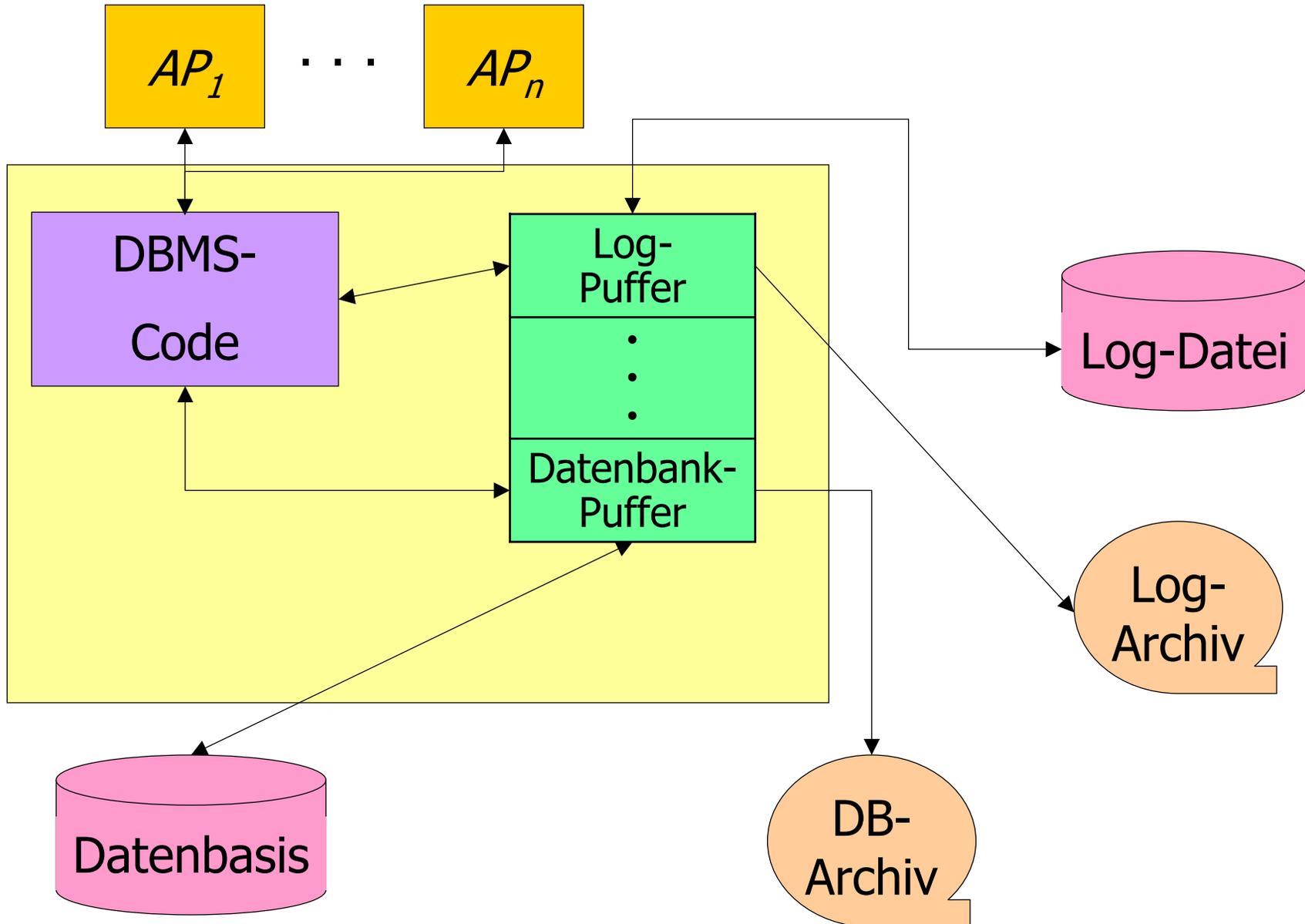
- das *Before-Image* wird durch Ausführung des Undo-Codes aus dem *After-Image* generiert und
- das *After-Image* durch Ausführung des Redo-Codes aus dem *Before-Image* berechnet.

Speicherung der Seiten-LSN

Die „Herausforderung“ besteht darin, beim Wiederaufbau zu entscheiden, ob man das Before- oder das After-Image auf dem Hintergrundspeicher vorgefunden hat.

Dazu wird auf jeder Seite die LSN des jüngsten diese Seite betreffenden Log-Eintrags gespeichert.

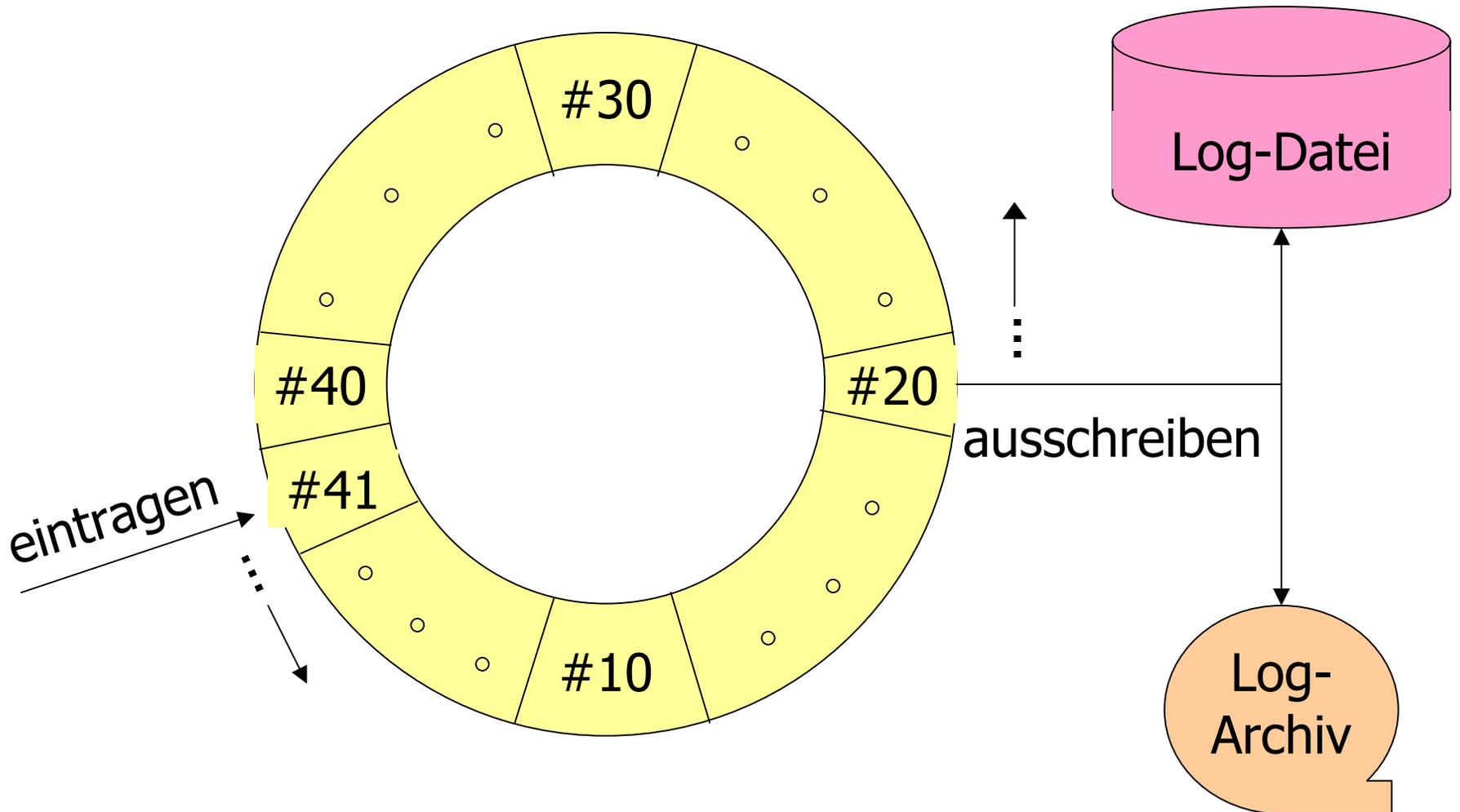
Schreiben der Log-Information



Schreiben der Log-Information

- Die Log-Information wird zweimal geschrieben
 1. Log-Datei für schnellen Zugriff
 - R1, R2 und R3-Recovery
 2. Log-Archiv
 - R4-Recovery

Anordnung des Log-Ringpuffers



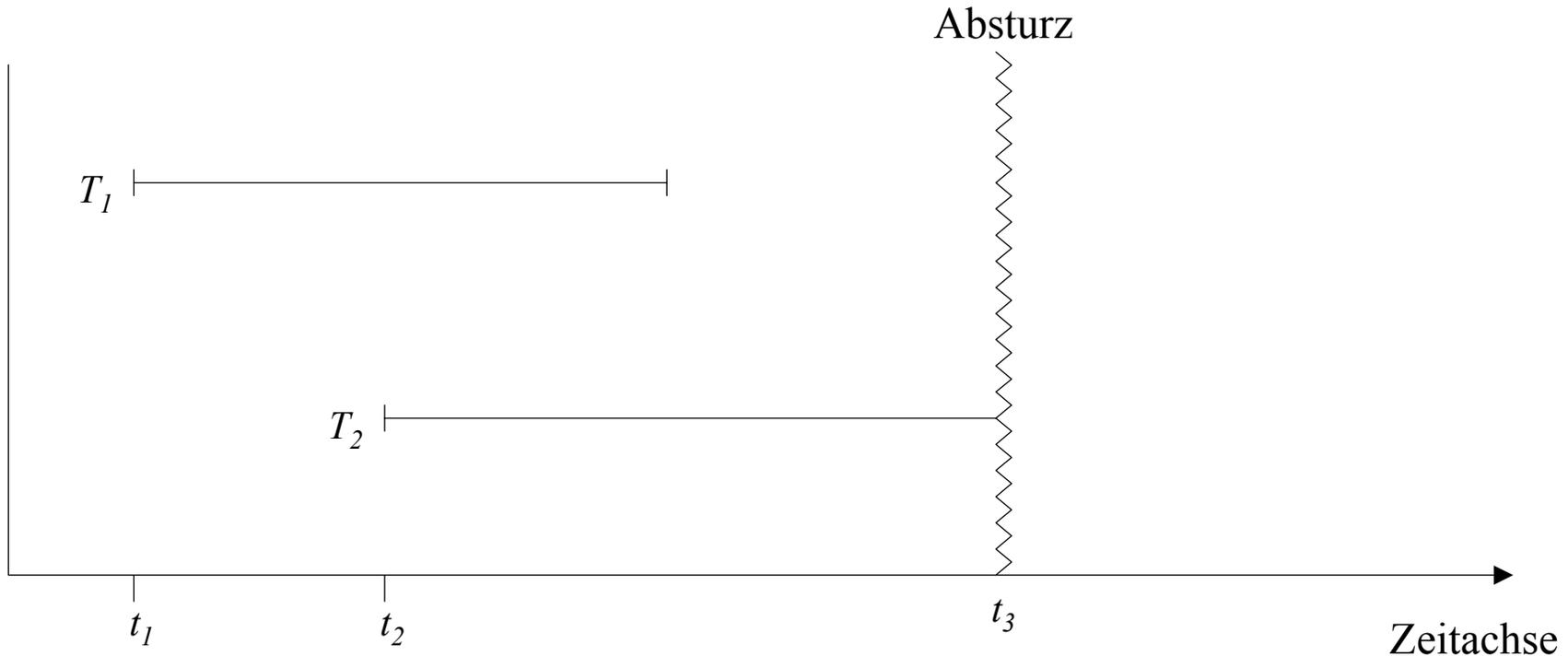
Das WAL-Prinzip

Write Ahead Log-Prinzip

1. Bevor eine Transaktion festgeschrieben (**committed**) wird, müssen alle „zu ihr gehörenden“ Log-Einträge ausgeschrieben werden. (→ redo sicherstellen)
2. Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in das temporäre und das Log-Archiv ausgeschrieben werden. (→ undo sicherstellen)

Wiederaanlauf nach einem Fehler

Transaktionsbeginn und -ende relativ zu einem Systemabsturz



- Transaktionen der Art T_1 müssen hinsichtlich ihrer Wirkung vollständig nachvollzogen werden. Transaktionen dieser Art nennt man *Winner*.
- Transaktionen, die wie T_2 zum Zeitpunkt des Absturzes noch aktiv waren, müssen rückgängig gemacht werden. Diese Transaktionen bezeichnen wir als *Loser*.

Drei Phasen des Wiederanlaufs

1. Analyse:

- Die temporäre Log-Datei wird von Anfang bis zum Ende analysiert,
- Ermittlung der *Winner*-Menge von Transaktionen des Typs T_1
- Ermittlung der *Loser*-Menge von Transaktionen der Art T_2 .

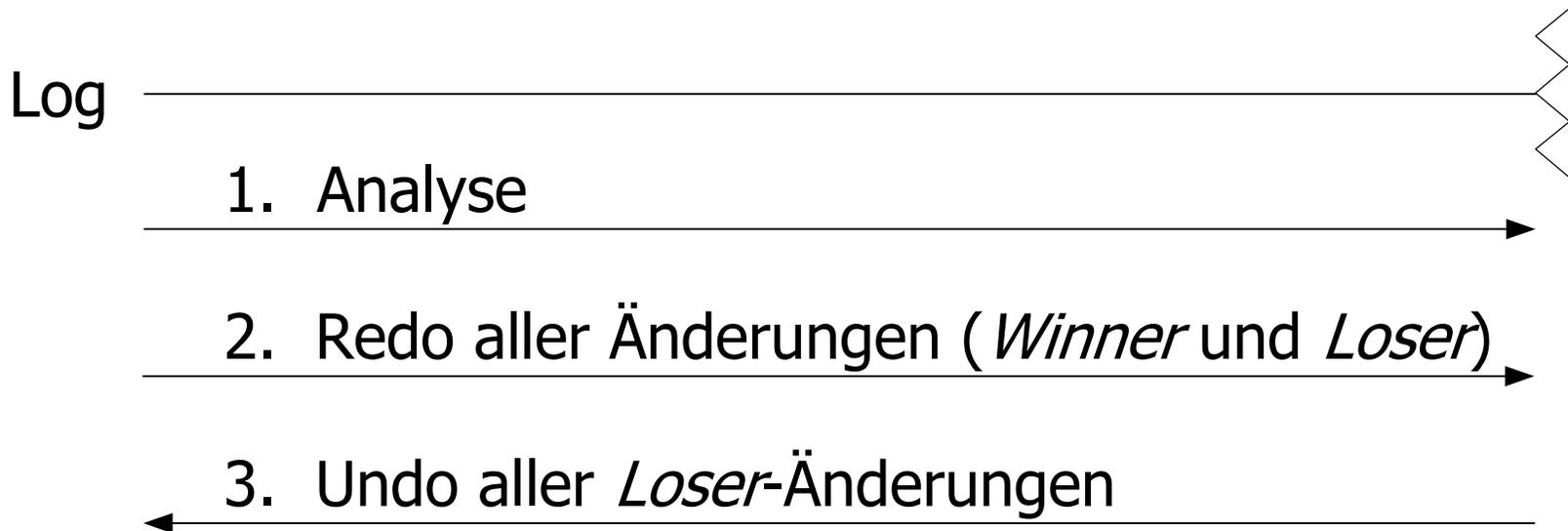
2. Wiederholung der Historie:

- *alle* protokollierten Änderungen werden in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht.

3. Undo der Loser:

- Die Änderungsoperationen der *Loser*-Transaktionen werden in umgekehrter Reihenfolge ihrer ursprünglichen Ausführung rückgängig gemacht.

Wiederanlauf in drei Phasen



Fehlertoleranz (Idempotenz) des Wiederanlaufs

$$\text{undo}(\text{undo}(\dots(\text{undo}(a))\dots)) = \text{undo}(a)$$

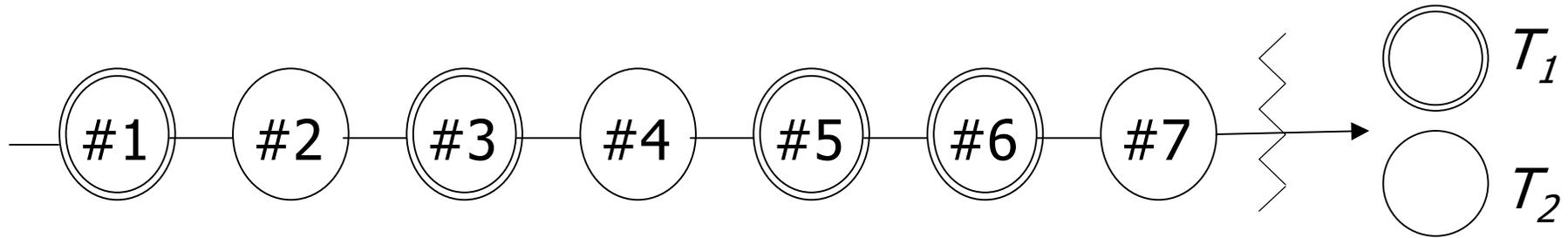
$$\text{redo}(\text{redo}(\dots(\text{redo}(a))\dots)) = \text{redo}(a)$$

- Auch während der Recoveryphase kann das System abstürzen!
→ auch das Recovery muss geloggt werden.

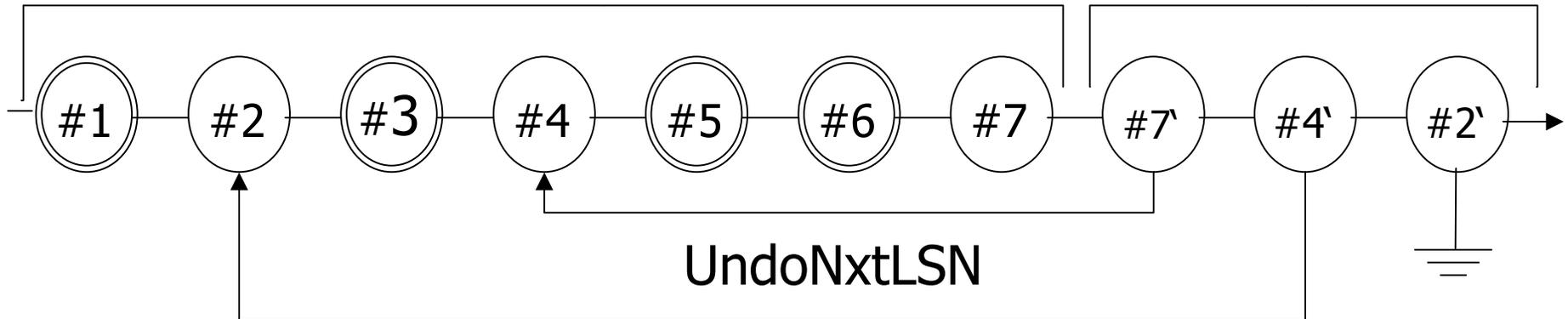
Beispiel einer Log-Datei

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]

Kompensationseinträge im Log



Wiederanlauf und Log



- Kompensationseinträge (CLR: compensating log record) für rückgängig gemachte Änderungen.

- #7 ist CLR für #7
- #4 ist CLR für #4

Wie bei der doppelten Buchführung darf im Log nicht „radiert“ werden.

Logeinträge nach abgeschlossenem Wiederanlauf

[#1, T₁, BOT, 0]

[#2, T₂, BOT, 0]

[#3, T₁, P_A, A- = 50, A+ = 50, #1]

[#4, T₂, P_C, C+ = 100, C- = 100, #2]

[#5, T₁, P_B, B+ = 50, B- = 50, #3]

[#6, T₁, commit, #5]

[#7, T₂, P_A, A- = 100, A+ = 100, #4]

<#7', T₂, P_A, A+ = 100, #7, #4>

<#4', T₂, P_C, C- = 100, #7', #2>

<#2', T₂, -, -, #4', 0>

Logeinträge

nach abgeschlossenem Wiederanlauf II

- CLR's sind durch spitze Klammern <...> gekennzeichnet.
- der Aufbau eines CLR ist wie folgt
 - LSN
 - TA-Identifikator
 - betroffene Seite
 - Redo-Information
 - PrevLSN
 - UndoNxtLSN (Verweis auf die nächste rückgängig zu machende Änderung)
- CLR's enthalten keine Undo-Information
 - warum nicht?

R4-Recovery / Media-Recovery

Recovery nach einem Verlust der materialisierten Datenbasis

