

Fehlerbehandlung und Recovery

VU Datenbanksysteme vom 28.10.2015

Reinhard Pichler

Arbeitsbereich Datenbanken und Artificial Intelligence
Institut für Informationssysteme
Technische Universität Wien

Transaktionsverwaltung

- ▶ Systemkonfiguration
- ▶ Protokollierung
- ▶ Recovery
- ▶ Sicherungspunkte (Checkpointing)

Systemkonfiguration

- ▶ Ersetzungsstrategien
- ▶ Ausschreiben von Änderungen
- ▶ Einbringstrategien

Fehlerklassifikation und Fehlerbehandlung

Lokaler Fehler in einer noch nicht festgeschriebenen (committed) Transaktion:

- ▶ Wirkung muss (rasch!) zurückgesetzt werden
- ▶ R1-Recovery (lokales Undo)

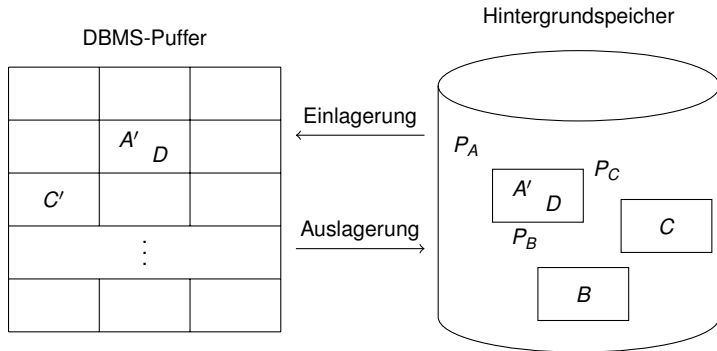
Fehler mit Hauptspeicherverlust:

- ▶ R2-Recovery (globales Redo): Abgeschlossene TAs müssen erhalten bleiben
- ▶ R3-Recovery (globales Undo): Noch nicht abgeschlossene TAs müssen zurückgesetzt werden.

Fehler mit Hintergrundspeicherverlust:

- ▶ R4-Recovery

Zweistufige Speicherhierarchie



Ersetzung von Pufferseiten

Fixieren von Seiten im Datenbankpuffer:

- ▶ Während des Zugriffs auf eine Seite darf diese auf keinen Fall aus dem Puffer verdrängt werden.
- ▶ Danach ist ein Verdrängen grundsätzlich möglich (unabhängig davon, ob die Transaktion noch weitergeht).

Strategien zur Ersetzung von Pufferseiten:

- steal** Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung, falls neue Seiten eingelagert werden müssen.
- ¬steal** Seiten, die von einer noch aktiven Transaktion modifiziert wurden, dürfen nicht ersetzt werden.

Ausschreiben von Änderungen

Strategien zum Ausschreiben von Änderungen abgeschlossener Transaktionen:

- force** Änderungen werden zum Transaktionsende auf den Hintergrundspeicher geschrieben.
- ¬force** Auch nach Transaktionsende sind geänderte Seiten eventuell nicht auf den Hintergrundspeicher geschrieben worden.

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

Einige Gründe gegen $\neg\text{steal/force}$

- ▶ Erzwungene Propagierung aller Änderungen zum Transaktionsende ist teuer.
- ▶ Manche Seiten („hot spots“) würden ausgeschrieben werden, obwohl sie nicht aus dem Datenbankpuffer verdrängt werden.
- ▶ Ausschreiben aller geänderten Seiten zum Transaktionsende müsste atomar geschehen („alles oder nichts“).
- ▶ Kombination $\neg\text{steal/force}$ ist nur möglich, wenn Transaktionen immer ganze Seiten sperren.

Einbringstrategien

„Direkte“ Einbringstrategie:

- ▶ Update-in-Place
 - ▶ Jede Seite hat genau eine „Heimat“ auf der Platte
 - ▶ Wenn eine (modifizierte) Seite aus dem Datenbankpuffer verdrängt wird, wird sie genau an dieser Stelle auf die Platte geschrieben.
 - ▶ Der alte Zustand der Seite wird dadurch überschrieben.

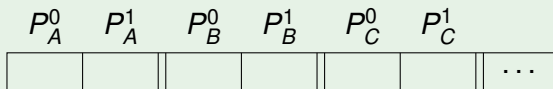
Einbringstrategien

„Indirekte“ Einbringstrategien:

- ▶ Twin-Block-Verfahren

Beispiel

Twin-Block-Anordnung der Seiten P_A , P_B , und P_C .



- ▶ Schattenspeicherkonzept
 - ▶ nur geänderte Seiten werden dupliziert
 - ▶ weniger Redundanz als beim Twin-Block-Verfahren
 - ▶ in der Praxis zu komplex

Übliche Systemkonfiguration

- ▶ steal: Eine Seite, die von einer nicht abgeschlossenen TA modifiziert wurde, kann auf Platte ausgeschrieben werden.
- ▶ \neg force: Geänderte Seiten sind bei Beendigung der Transaktion möglicherweise noch nicht auf die Platte geschrieben.
- ▶ update-in-place: Es gibt von jeder Seite nur eine Kopie auf der Platte.
- ▶ kleine Sperrgranulate (Kapitel Mehrbenutzersynchronisation) auf Datensatzebene, d.h.: eine Seite kann Datenbankänderungen sowohl von abgeschlossenen als auch von nicht abgeschlossenen Transaktionen enthalten.

Protokollierung

- ▶ Aufbau der Log-Datei
- ▶ Schreiben der Log-Information (WAL-Prinzip)

Log-Datei

Motivation:

- ▶ (wegen steal): Die Seiten auf der Platte können Änderungen von noch nicht abgeschlossenen Transaktionen enthalten, d.h. Undo muss möglich sein.
- ▶ (wegen \neg force): Die Seiten auf der Platte enthalten eventuell nicht alle Änderungen von bereits abgeschlossenen Transaktionen, d.h.: Redo muss möglich sein.

Lösung: Zusätzliche Information in Log-Datei (= Protokoll-Datei).

- ▶ Log-Datei enthält alle für Redo bzw. Undo erforderlichen Informationen über Änderungsoperationen.
- ▶ Außerdem enthält Log-Datei Informationen über Beginn und Ende von Transaktionen.

Struktur der Log-Einträge

Definition

Struktur der Log-Einträge:

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

LSN (Log Sequence Number):

- ▶ Eindeutige Kennung des Log-Eintrags.
- ▶ LSNs müssen monoton aufsteigend vergeben werden.
- ▶ Die chronologische Reihenfolge der Protokolleinträge kann dadurch ermittelt werden.

TransaktionsID:

- ▶ Kennung der Transaktion, die Änderung durchgeführt hat.

PageID:

- ▶ Kennung der Seite, die geändert wurde.
- ▶ Wenn eine Änderung mehr als eine Seite betrifft, müssen entsprechend viele Log-Einträge generiert werden.

Struktur der Log-Einträge

Definition

Struktur der Log-Einträge:

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

Redo:

- ▶ Gibt an, wie die Änderung nachvollzogen werden kann.

Undo:

- ▶ Gibt an, wie die Änderung rückgängig gemacht werden kann.

PrevLSN:

- ▶ Zeiger auf den vorhergehenden Log-Eintrag der jeweiligen Transaktion.
- ▶ Diesen Eintrag benötigt man aus Effizienzgründen (insbesondere für das lokale Undo).

Beispiel

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 vs. physische Protokollierung

Physische Protokollierung: Redo- und Undo-Eintrag enthalten Inhalte/Zustände.

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

Logische Protokollierung: Redo- und Undo-Eintrag enthalten Anweisungen für das Erreichen der Inhalte/Zustände.

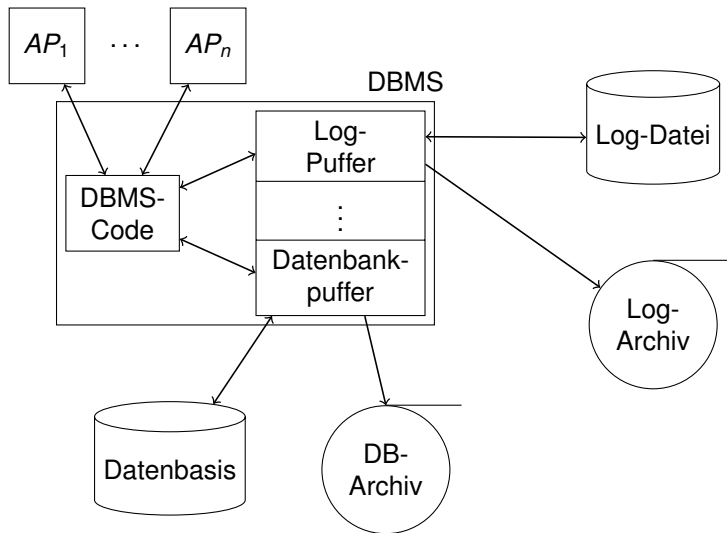
- ▶ Das **before-image** wird durch Ausführung des Undo-Codes aus dem *after-image* generiert.
- ▶ Das **after-image** wird durch Ausführung des Redo-Codes aus dem *before-image* berechnet.

Logische vs. physische Protokollierung

Speicherung der Seiten-LSN:

- ▶ Beim Wiederanlauf muss das DBMS erkennen können, ob auf einer bestimmten Seite das before- oder das after-image auf dem Hintergrundspeicher steht.
- ▶ Dies ist insbesondere bei logischer Protokollierung notwendig.
- ▶ Dazu wird auf jeder Seite die LSN des jüngsten diese Seite betreffenden Log-Eintrags gespeichert.

Schreiben der Log-Information

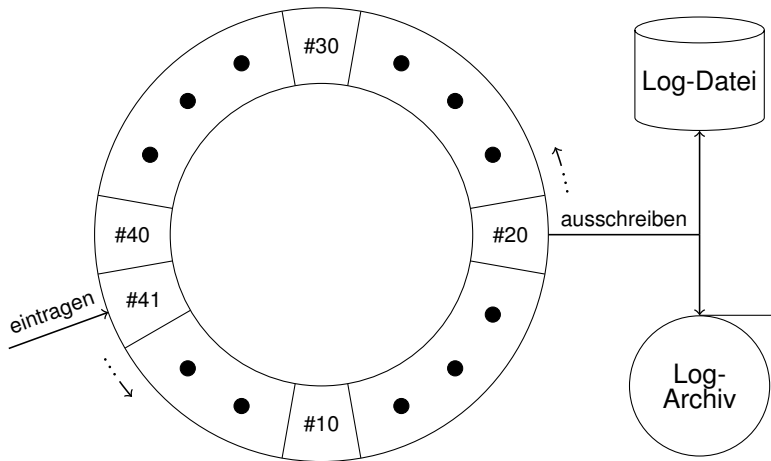


Schreiben der Log-Information

Die Log-Information wird zwei Mal geschrieben:

1. Log-Datei für schnellen Zugriff:
R1, R2 und R3-Recovery
2. Log-Archiv:
R4-Recovery

Anordnung des Log-Ringpuffers



WAL-Prinzip

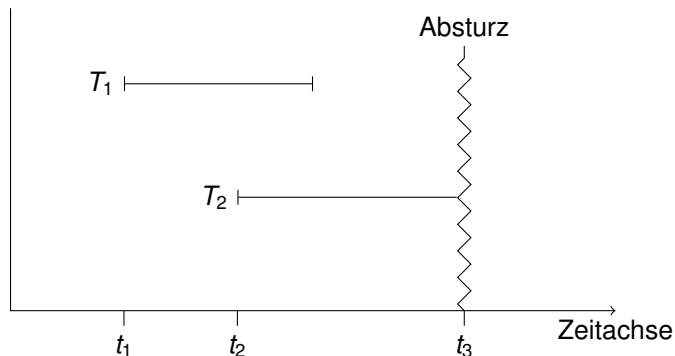
- ▶ Das **kontinuierliche Ausschreiben** aus dem Log-Ringpuffer vermeidet „stoßweises“ Ausschreiben bei Beendigung einer Transaktion.
- ▶ Es muss dabei aber das WAL-Prinzip (= write ahead log) eingehalten werden:
 - ▶ **für Redo:** Bevor eine Transaktion festgeschrieben (committed) wird, müssen alle „zu ihr gehörenden“ Log-Einträge ausgeschrieben werden.
 - ▶ **für Undo:** Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, ausgeschrieben werden.
 - ▶ Dabei muss die chronologische Reihenfolge der Log-Einträge laut Log-Ringpuffer erhalten bleiben.

Recovery

- ▶ Wiederanlauf nach Systemabsturz: R2/R3-Recovery
- ▶ R1-Recovery
- ▶ R4-Recovery

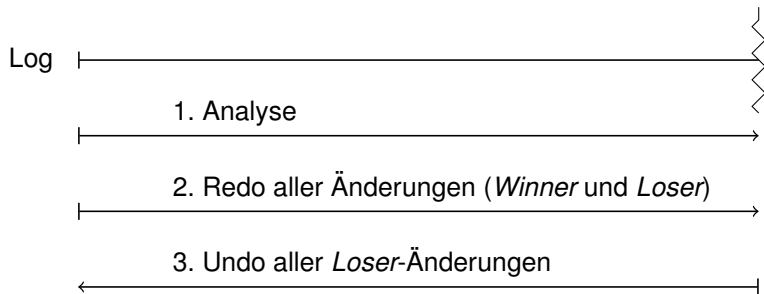
Wiederanlauf nach einem Fehler

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 nennt man **Loser**.

Drei Phasen des Wiederanlaufs



1. Analyse:

- ▶ Die Log-Datei wird von Anfang bis zum Ende analysiert.
- ▶ Ermittlung der **Winner**-Menge, das sind Transaktionen, für die ein BOT und ein commit in der Log-Datei gefunden wurden.
- ▶ Ermittlung der **Loser**-Menge, das sind Transaktionen für die ein BOT aber kein commit in der Log-Datei gefunden wurde.

Drei Phasen des Wiederanlaufs

2. Redo-Phase:

- ▶ Alle protokollierten Änderungen (sowohl von Winner- als auch von Loser-Transaktionen) werden in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht (d.h.: geordnet nach LSN und nicht nach Transaktionen).
- ▶ Durch Vergleich der LSN des Log-Eintrags mit der LSN der betroffenen Seite wird jeweils ermittelt, ob bereits das after-image auf der Platte steht.
- ▶ Falls $\text{log-LSN} > \text{Seiten-LSN}$, dann muss die Redo-Operation durchgeführt werden und die Seiten-LSN mit log-LSN überschrieben werden.

Drei Phasen des Wiederanlaufs

3. Undo-Phase:

- ▶ Die Änderungsoperationen der Loser-Transaktionen werden in umgekehrter Reihenfolge ihrer ursprünglichen Ausführung rückgängig gemacht (d.h. geordnet nach absteigender LSN und nicht nach Transaktionen).
- ▶ Für jede Undo-Operation kommt ein neuer Eintrag (**CLR**, compensation log record) in die Log-Datei (siehe unten).

Fehlertoleranz (Idempotenz) des Wiederanlaufs

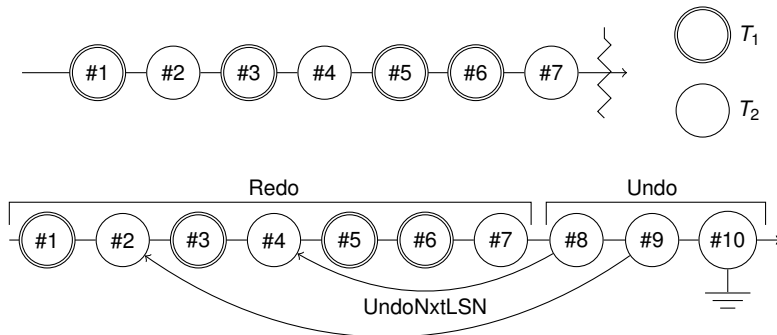
- ▶ Problem: auch während der Recovery-Phase kann das System abstürzen.
- ▶ Anforderung:

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

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

- ▶ Realisierung der Idempotenz:
 - ▶ Redo-Phase: mittels LSN (in jedem Log-Eintrag bzw. auf jeder Seite)
 - ▶ Undo-Phase: mittels CLRs

Kompensationseinträge im Log



Kompensationseinträge (CLR, compensation log record) für rückgängig gemachte Änderungen und für BOT:

- ▶ #8 ist CLR für #7
- ▶ #9 ist CLR für #4
- ▶ #10 ist CLR für #2 (= BOT)

Log-Einträge nach abgeschlossenem Wiederanlauf

Beispiel

[#1, T_1 , **BOT**, 0]
[#2, T_2 , **BOT**, 0]
[#3, T_1 , P_A , $A-=50$, $A+=50$, #1]
[#4, T_2 , P_C , $C+=100$, $C-=100$, #2]
[#5, T_1 , P_B , $B+=50$, $B-=50$, #3]
[#6, T_1 , **commit**, #5]
[#7, T_2 , P_A , $A-=100$, $A+=100$, #4]
⟨#8, T_2 , P_A , $A+=100$, #7, #4⟩
⟨#9, T_2 , P_C , $C-=100$, #8, #2⟩
⟨#10, T_2 , -, -, #9, 0⟩

Erläuterung:

- ▶ #7, #8, #9: PrevLSN
- ▶ #4, #2, 0: UndoNxtLSN

Log-Einträge nach abgeschlossenem Wiederanlauf

Notation: CLRs in spitzen Klammern $\langle \dots \rangle$

Definition

Aufbau eines CLR:

$\langle LSN, TransaktionsID, PageID, Redo, PrevLSN, UndoNxtLSN \rangle$

- ▶ *UndoNxtLSN*: Verweis auf die nächste rückgängig zu machende Änderung derselben Transaktion.
- ▶ CLRs enthalten keine Undo-Information: Im Falle eines neuerlichen Systemabsturzes soll die Undo-Operation, für die das CLR angelegt wurde, nicht zurückgenommen sondern fortgesetzt werden.

Wiederanlauf bei neuerlichem Absturz:

- ▶ CLRs sind immer Teil der Redo-Phase!

Systemabsturz während des Wiederanlaufs

Beispiel

Absturz nach dem CLR mit LSN = #8:

- ▶ Redo-Phase: Log-Records #1 bis #8.
Änderungsoperationen werden natürlich nur dann durchgeführt, wenn die betroffene Seite auf der Platte eine kleinere LSN hat.
- ▶ Undo-Phase: Laut UndoNextLSN-Eintrag des letzten CLR muss das Undo von T_2 bei LSN = 4 fortgesetzt werden.

Beispiel

Absturz nach dem CLR mit LSN = #10:

- ▶ Redo-Phase: Log-Records #1 bis #10 (wobei wiederum die LSNs der betroffenen Seiten beachtet werden müssen).
- ▶ Undo-Phase: Wegen UndoNextLSN = 0 im letzten CLR ist keine Undo-Operation für T_2 mehr erforderlich.

R1-Recovery: Lokales Zurücksetzen einer Transaktion

Erforderliche Schritte:

- ▶ Ermittlung des letzten Log-Eintrags dieser Transaktion (üblicherweise wird ein Zeiger auf den letzten Log-Eintrag aller aktiven Transaktionen im Hauptspeicher gehalten.)
- ▶ Lokales Undo: Die Änderungsoperationen dieser Transaktion werden in umgekehrter Richtung rückgängig gemacht. Genau wie beim globalen Undo werden CLRs angelegt.

Effizienz:

- ▶ Durch die Rückwärtsverkettung mittels PrevLSN sind die Log-Einträge von T leicht zu finden.
- ▶ Bei realistischer Größe des Ringpuffers sind die meisten Log-Einträge der aktiven Transaktionen noch im Hauptspeicher. Lokales Undo geht somit schnell.

Partielles Zurücksetzen einer Transaktion

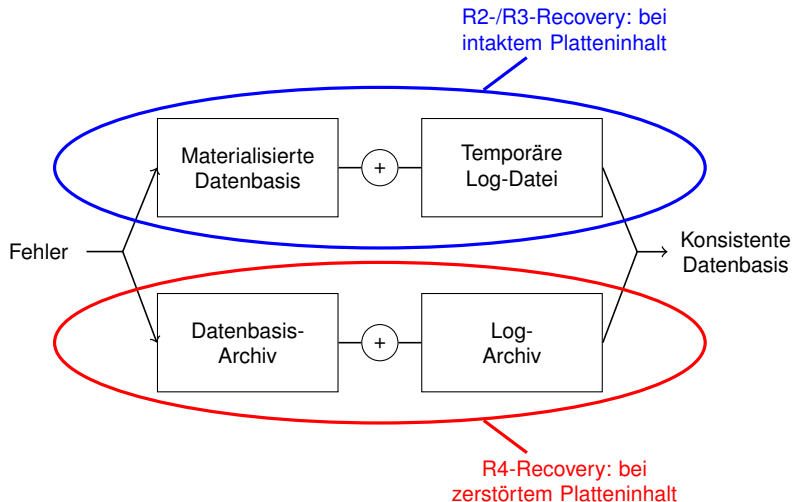
Erforderliche Schritte:

- ▶ Im Prinzip gleich wie lokales Undo, d.h. es werden Änderungsoperationen zurückgesetzt und entsprechende CLRs angelegt.
- ▶ Unterschied: Es wird nicht die gesamte Transaktion zurückgesetzt sondern nur ein Teil (und zwar der letzte Teil zurück bis zum gewünschten Rücksetzpunkt).

Bemerkung:

- ▶ Partielles Zurücksetzen ist notwendig für die Realisierung von Rücksetzpunkten (savepoints)

R4-Recovery / Media-Recovery



Sicherungspunkte (Checkpointing)

- ▶ unterschiedliche Sicherungspunkt-Qualitäten

Warum Sicherungspunkte?

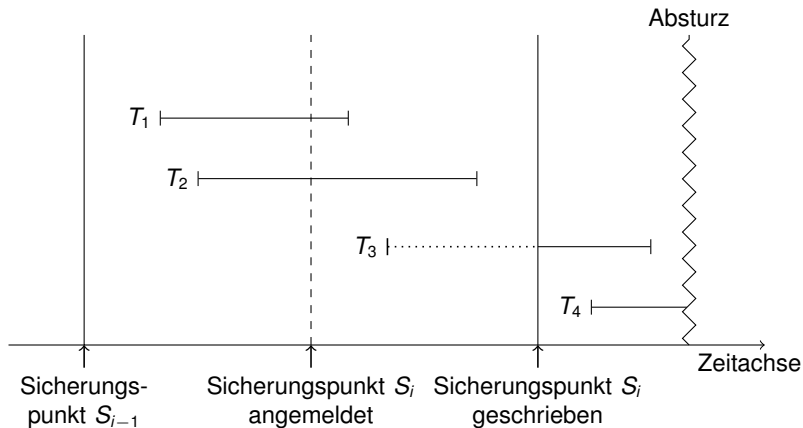
Problem bei bisher vorgestellter Recovery-Methode:

- ▶ Die Log-Datei wird mit der Zeit immer größer.
- ▶ Dementsprechend dauert der Wiederanlauf immer länger.

Lösung:

- ▶ Durch das erzwungene Ausschreiben von geänderten Seiten wird garantiert, dass die Log-Datei erst ab einem bestimmten Log-Eintrag benötigt wird.
- ▶ Bemerkung: Die minimale LSN, die im Falle eines Wiederanlaufs noch benötigt wird, hängt vom Zeitpunkt *und* von der Qualität des Sicherungspunktes ab.

Transaktionskonsistente Sicherungspunkte



Transaktionskonsistente Sicherungspunkte

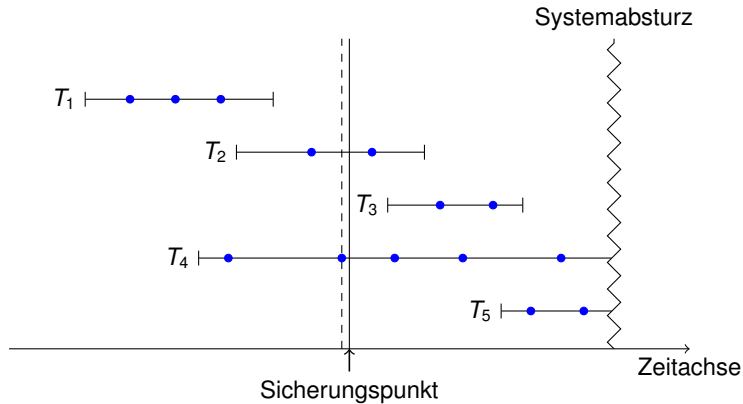
Ziel:

- ▶ Der Platteninhalt soll alle Änderungen von Transaktionen, die zum Zeitpunkt S_i abgeschlossen sind, enthalten. D.h., beim Wiederanlauf kein Redo über S_i hinaus nötig.
- ▶ Zum Zeitpunkt S_i darf es keine aktiven Transaktionen geben. D.h., beim Wiederanlauf kein Undo über S_i hinaus nötig.

Problem:

- ▶ Zwischen Anmelden und Festschreiben des Sicherungspunktes dürfen keine neuen Transaktionen gestartet werden. Das ist üblicherweise nicht akzeptabel.

Aktionskonsistente Sicherungspunkte



Aktionskonsistente Sicherungspunkte

Idee:

- ▶ Alle gerade angefangenen Änderungsoperationen sollen beendet werden. Danach werden alle modifizierten Seiten (= Dirty Pages) ausgeschrieben.
- ▶ Es wird nur das Starten der nächsten Änderungsoperation (aber nicht das Starten neuer Transaktionen) verzögert.

Wiederanlauf:

- ▶ Analyse-Phase setzt bei S_i auf.
- ▶ Kein **Redo** über S_i hinaus nötig.
- ▶ Im allgemeinen ist ein **Undo** über S_i hinaus nötig, und zwar bis zur *MinLSN* (= die kleinste LSN der zum Sicherungszeitpunkt aktiven TAs).

Unschärfe (fuzzy) Sicherungspunkte

Idee:

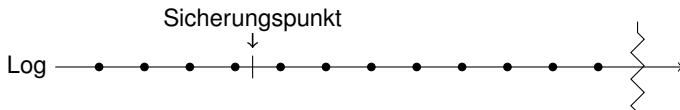
- ▶ Seiten sollen kontinuierlich ausgeschrieben werden. D.h., kein abruptes Ausschreiben vieler Seiten wegen des Sicherungspunktes.
- ▶ Statt dessen werden nur die Kennungen aller modifizierten Seiten (= Dirty Pages) ausgeschrieben.
- ▶ Zusätzlich wird *MinDirtyPageLSN* (= minimale LSN, deren Änderungen noch nicht ausgeschrieben wurden) verwaltet und bei einem Sicherungspunkt auf Platte geschrieben.

Problem:

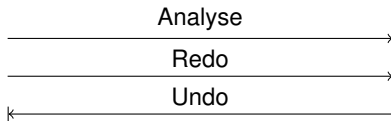
- ▶ „hot-spots“ (laufend benötigte Seiten) werden lange nicht ausgeschrieben. Ausschreiben wird erzwungen, wenn eine Seite mehrmals in der „Dirty Pages“-Menge war.

Drei Sicherungspunkt-Qualitäten

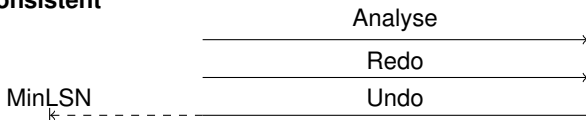
Zusammenfassung



(a) transaktionskonsistent



(b) aktionskonsistent



(c) unscharf (fuzzy)

