Datenbanksysteme

Wintersemester 2015/16

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Welche Aspekte von

ACID

werden durch Recovery adressiert?

Implikationen von ACID auf Anforderungen zu Recovery

Durability

- Änderungen an der Datenbank, die durch erfolgreich(!) abgeschlossene (d.h. committed) Transaktionen verursacht wurden, müssen dauerhaft gespeichert sein.
- D.h. bei einem Crash der DB muss nach Wiederanlauf geschaut werden, ob dies tatsächlich der Fall ist.

Atomicity

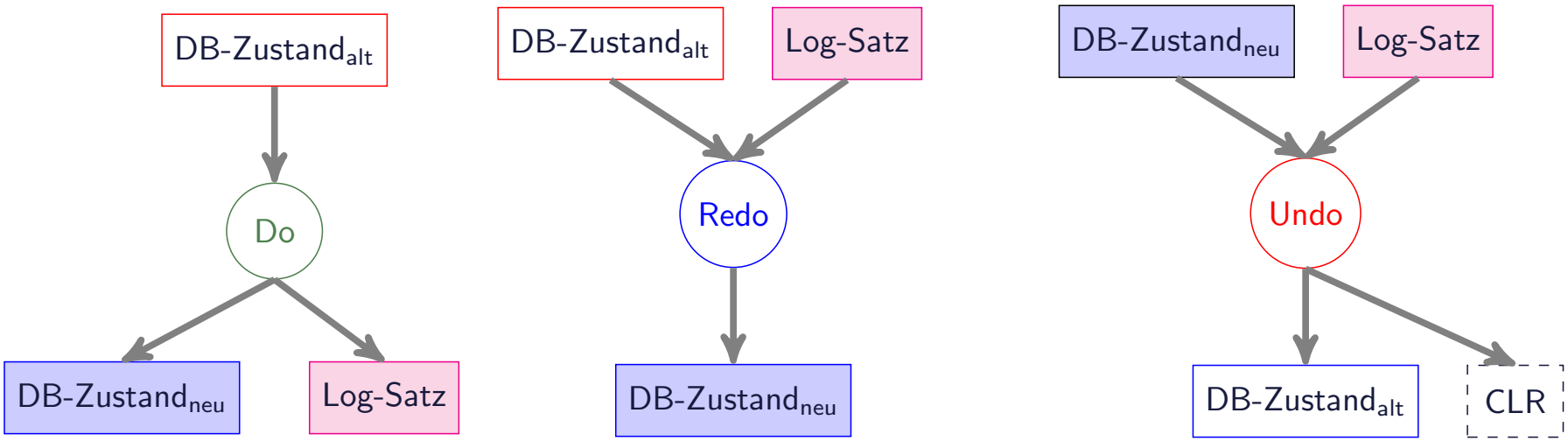
- Falls eine Transaktion noch nicht erfolgreich abgeschlossen wurde und ein DB-Crash auftritt, muss beim Wiederanlauf darauf geachtet werden, dass etwaige Änderungen in der Datenbasis rückgängig gemacht werden.

Vorsorge für den Fehlerfall

Logging

- Sammlung redundanter Daten bei Änderungen im Normalbetrieb, als Voraussetzung für Recovery
- Einsatz im Fehlerfall (Undo-, Redo-Recovery)

Do-Redo-Undo-Prinzip



Crash-Recovery

Im folgenden wird Crash-Recovery betrachtet.

Idee/Ziel

- **Wiederanlauf (Restart) des DBS nach einem Systemfehler**
- Ziel: Nach Wiederanlauf den jüngsten transaktionskonsistenzen DB-Zustand wiederherstellen, der zum Fehlerzeitpunkt gültig war.
- **Dazu wird materialisierte Datenbasis und Log-Datei benutzt.**
- Zusätzliche Anforderung: **Idempotenz!** D.h. bei mehrfacher Anwendung der Recovery muss dies stets zum selben Ergebnis führen.

Abhängigkeit von System-Konfiguration

- Methoden zur Crash-Recovery sind wesentlich durch die zugrunde liegende System-Konfiguration bestimmt (Satz oder Seitensperren, steal oder no steal, etc), wie auf folgenden Folien motiviert.

Undo und Redo

Pufferinhalt geht verloren, was dann?

Undo

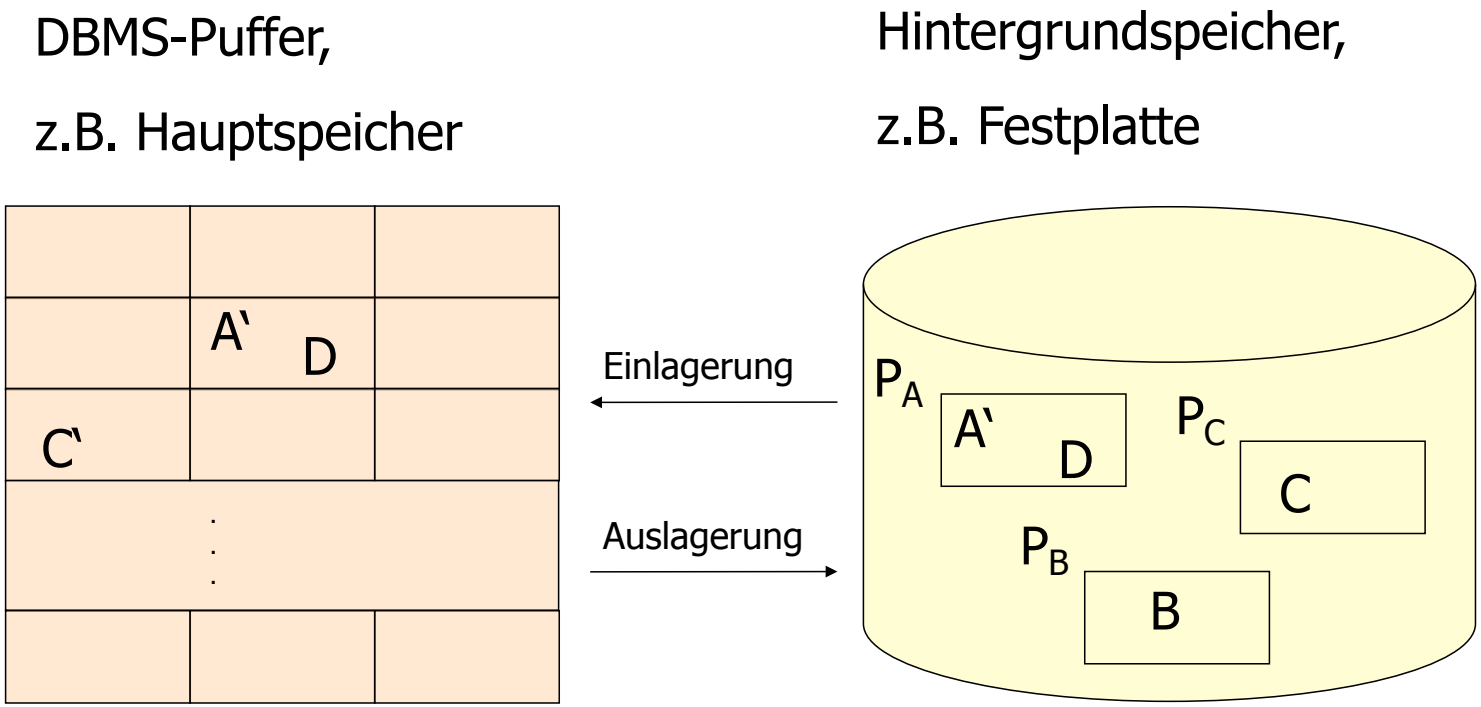
Alle durch nicht abgeschlossene Transaktionen schon in die materialisierte Datenbasis eingebrachten Änderungen müssen rückgängig gemacht werden.

Redo

Alle noch nicht in die materialisierte Datenbasis eingebrachten Änderungen durch abgeschlossene Transaktionen müssen nachvollzogen werden.

Zweistufige Speicherhierarchie

- Seiten P_i
- Datensätze A,B,C,D,..
- **Werden später den Fall betrachten, dass Datensätze gesperrt werden, also versch. TA die gleiche Seite bearbeiten können.**



Einbringungsstrategie

Update in Place:

- jede Seite hat genau eine "Heimat" auf dem Hintergrundspeicher
- der alte Zustand einer Seite wird überschrieben

Twin-Block-Verfahren:

- Jede Seite hat zwei Versionen
- Anordnung für Seiten P_A, P_B und P_C

P_A^0	P_A^1	P_B^0	P_B^1	P_C^0	P_C^1	...

Schattenspeicherkonzept:

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

Die Speicherhierarchie

Ersetzung von Puffer-Seiten:

- ¬**steal:** Ersetzung von Seiten, die von einer noch aktiven Transaktion modifiziert wurden, ausgeschlossen ⇒ "dreckige" Seiten (dirty pages) müssen im Puffer bleiben.
- **steal:** Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung, falls neue Seiten eingelagert werden müssen, auch "dreckige" Seiten.

Einbringen von Änderungen abgeschlossener TAs:

- **force:** Änderungen werden bei commit auf den Hintergrundspeicher geschrieben (flush).
- ¬**force:** geänderte Seiten können auch nach commit im Puffer verbleiben.

dirty page = Inhalt der Seite im HS ≠ Inhalt der Seite auf FS

Auswirkungen auf Recovery

Undo: entfernt ungültige Einträge aus der DB.

Redo: fügt gültige Einträge in die DB sein.

	force	**¬force**
¬steal		
steal		

- Was ist besser? steal oder ¬steal?
- Was ist besser? force oder ¬force?
- Annahme: Schreiboperationen von Seiten müssen atomar sein.

Auswirkungen auf Recovery

Undo: entfernt ungültige Einträge aus der DB.

Redo: fügt gültige Einträge in die DB sein.

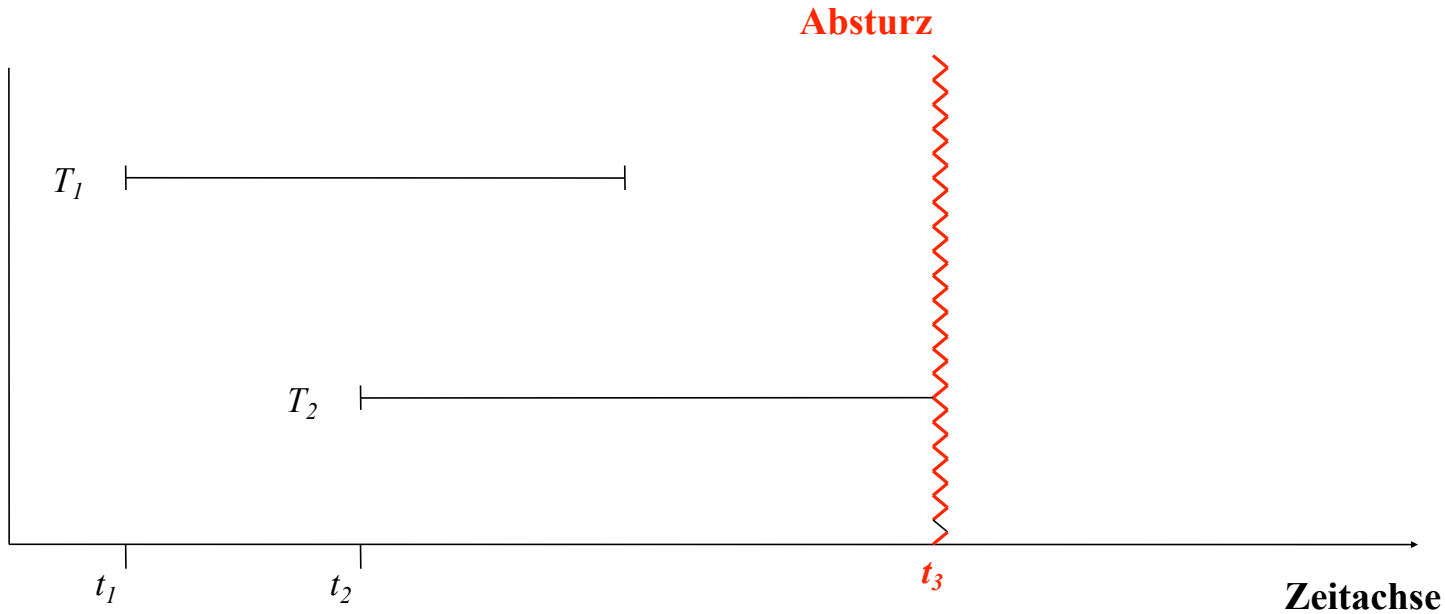
	force	**¬force**
¬steal	• kein Undo • kein Redo ⇒ **keine** Recovery	• Redo • kein Undo
steal	• kein Redo • Undo	• Redo • Undo

- Was ist besser? steal oder ¬steal?
- Was ist besser? force oder ¬force?
- Annahme: Schreiboperationen von Seiten müssen atomar sein.

Hier zugrunde gelegte Systemkonfiguration

- **steal**
  "dreckige Seiten" können in die 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, kleiner als eine Seite**
  auf Satzebene. Also kann eine Seite gleichzeitig "dreckige" Daten (einer noch nicht abgeschlossenen TA) und "committed updates" enthalten.

Wiederanlauf nach einem Fehler



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

Beispiel einer Log-Datei

Schritt	T_1	T_2	Log-Record [LSN, TA, PageID, Redo, Undo, PrevLSN]
1.	**BOT**		[#1,T_1, **BOT**, 0]
2.	r(A,a1)		
3.		**BOT**	[#2,T_2, **BOT**, 0]
4.		r(C,c2)	
5.	a1 := a1 - 50		
6.	w(A,a1)		[#3,T_1,PA,A-=50,A+=50,#1]
7.		c2 := c2 + 100	
8.		w(C,c2)	[#4,T_2,PC,C+=100,C-=100,#2]
9.	r(B,b1)		
10.	b1 := b1 + 50		
11.	w(B,b1)		[#5,T_1,PB,B+=50,B-=50,#3]
12.	**commit**		[#6,T_1,**commit**,#5]
13.		r(A,a2)	
14.		a2 := a2 - 100	
15.		w(A,a2)	[#7,T_2,PA,A-=100,A+=100,#4]
16.		**commit**	[#8,T_2,**commit**,#7]

WAL-Prinzip und Commit-Regel bei Log-basierter Recovery

Write-Ahead-log-Prinzip (WAL)

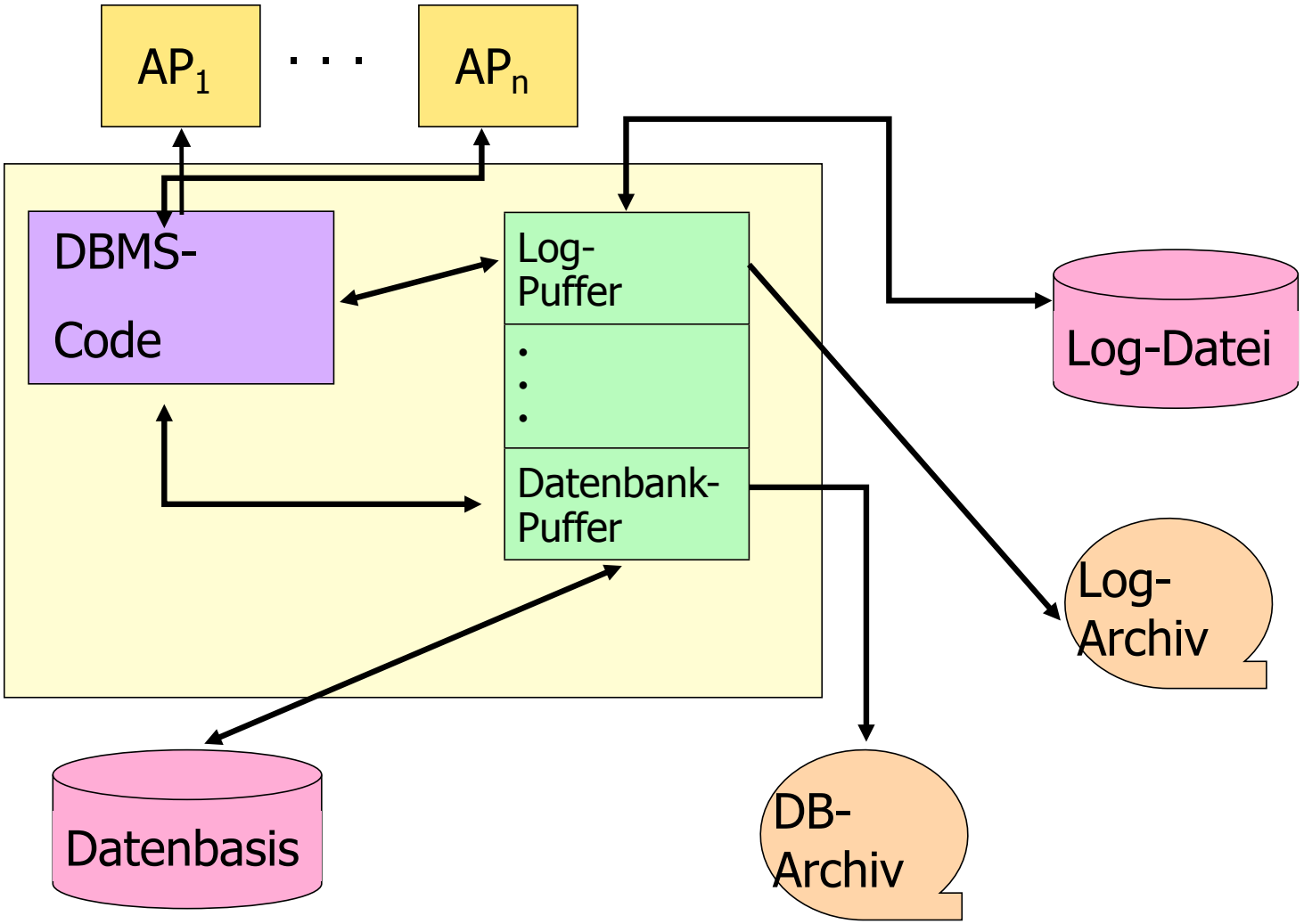
Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in die Log-Datei und das Log-Archiv ausgeschrieben werden.

Commit-Regel (Force-Log-at-Commit)

Bevor eine Transaktion festgeschrieben (**committed**) wird, müssen alle "zu ihr gehörenden" Log-Einträge auf stabilen Storage ausgeschrieben werden.

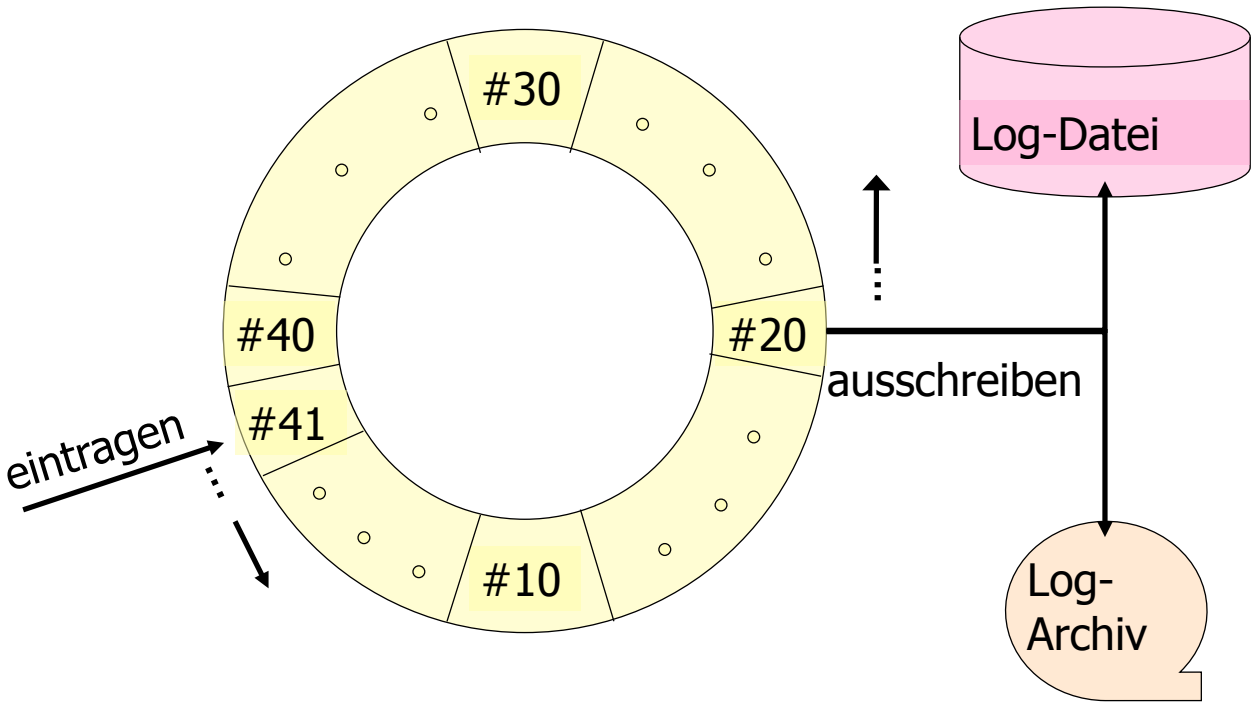
C. Mohan et al. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging.
`http://www.cs.berkeley.edu/~brewer/cs262/Aries.pdf`

Schreiben von Log-Informationen



- Log-Informationen werden zweimal geschrieben: Log-Datei für schnellen Zugriff und Log-Archiv

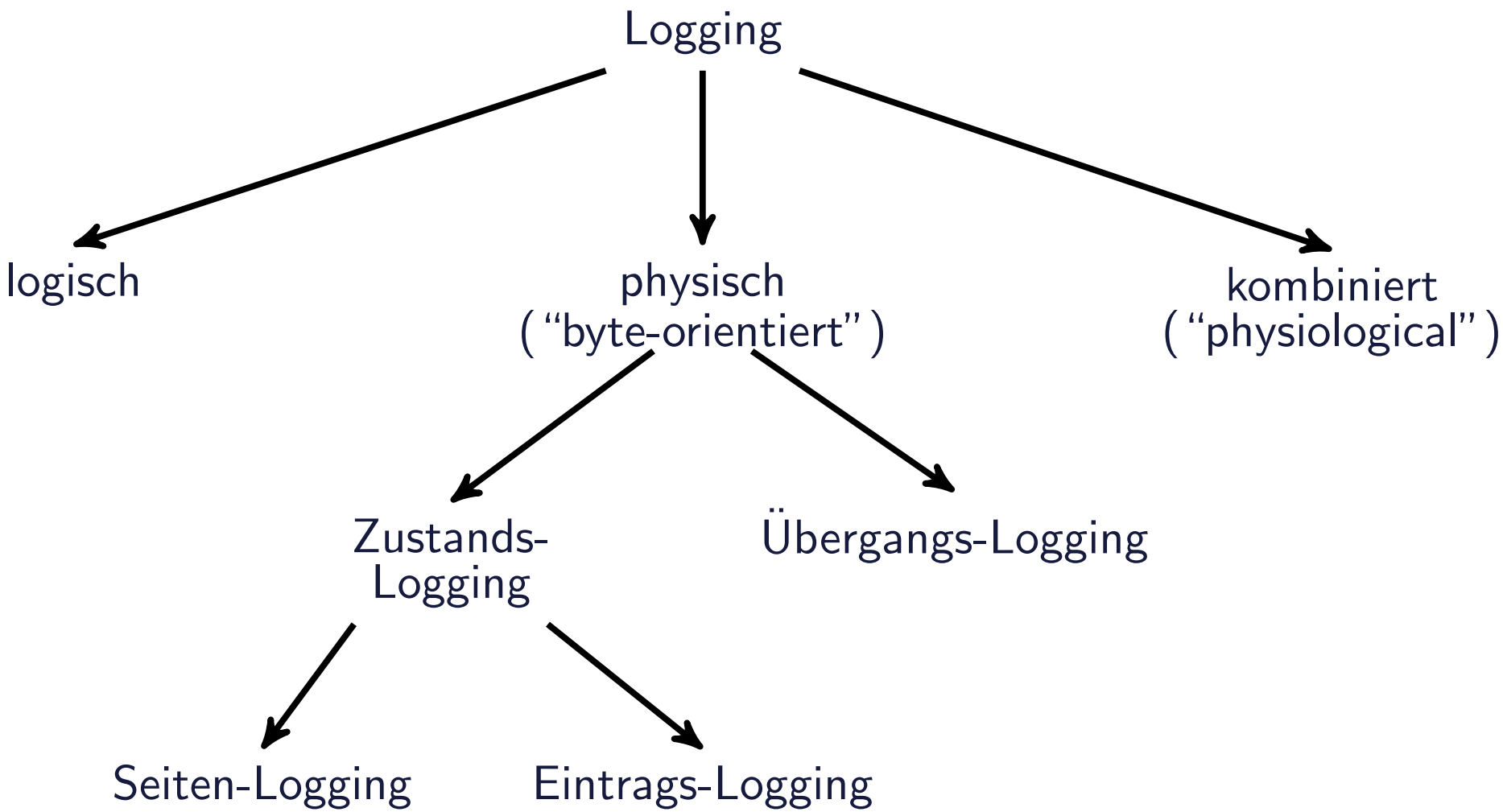
Anordnung des Log-Ringpuffers



- Kontinuierliches Ausschreiben
- Aber Achtung: WAL und Commit-Regel beachten!
- Log-Puffer ist normalerweise kleiner als DB-Puffer
- Groß genug um i.d.R. laufende Transaktionen zu enthalten, so dass beim Rücksetzen einer TA dies anhand des Puffers gemacht werden kann.

Klassifikation von Logging-Verfahren

Wie können Redo- und Undo-Informationen protokolliert werden?



Physische Protokollierung: Zustands-Logging

Zustands-Logging Protokollierung

- Physischer Zustand von Objekten (Seiten, Bereiche von Seiten, ...) wird im Log gespeichert:

1. **before-image** enthält den Zustand vor Ausführung der Operation, z.B. als byte-array
2. **after-image** enthält den Zustand nach Ausführung der Operation, z.B. als byte-array

Seiten-Logging

- Einfachste Form des Zustands-Loggings ist Seiten-Logging: bei jeder Änderung wird Kopie der Seite vor und nach der Änderung im Log abgelegt.
- Macht Recovery einfach. Aber große Nachteile im Normalbetrieb: Log wird sehr groß und hoher I/O Aufwand.
- Effizienter: **Eintrags-Logging**, also Images bzgl. geändertem Eintrag in Seite.

Physische Protokollierung: Übergangs-Logging

Übergangs Protokollierung

1. Idee: Log-Umfang soll reduziert werden im Vergleich zu Zustands-Logging.
2. Speichere daher nur Zustandsdifferenz
3. So dass mit Hilfe der Zustandsdifferenz bei UNDO auf ursprünglichen Zustand kommt
4. und bei REDO auf der neue aus dem alten Zustand berechnet werden kann.

Differenzen-Logging

- Differenzbildung von altem und neuem Zustand durch XOR ($\oplus$) Operation.
- Erzeugt viele 0-Einträge, die dann noch komprimiert werden können, bzw. Differenzen auf Satz/Eintrags-Ebene.

Zustands- vs. Übergangs-Logging (Beispiel)

	Zustands-Logging	**Differenzen-Logging**
Normalbetrieb Änderung der Seite A 1.) $A_1 \rightarrow A_2$ 2.) $A_2 \rightarrow A_3$	Protokollierung der Before- und After-Images 1.) A_1, A_2 2.) A_2, A_3	Protokollierung der XOR-Differenzen 1.) $D_1 := A_1 \oplus A_2$ 2.) $D_2 := A_2 \oplus A_3$
Redo-Recovery (Startzustand A_1 liegt vor)	Ersetzen (Überschreiben) von A_1 durch A_2 bzw. A_3	$A_2 := A_1 \oplus D_1$ $A_3 := A_2 \oplus D_2$
Undo-Recovery (Endzustand A_3 liegt vor)	Ersetzen (Überschreiben) von A_3 durch A_2 bzw. A_1	$A_2 := A_3 \oplus D_2$ $A_1 := A_2 \oplus D_1$

Aus dem Buch von Härder und Rahm.

Logische Logging

Logisches Logging

- Eine Form von "Übergangs-Logging", allerdings werden hier die Änderungsoperationen (mit ihren Parametern) gespeichert.
- z.B. $a = a + 10$

Eigenschaften und Unterschiede zu physischem Logging

- Benötigt konsistenten Zustand, damit Operationen anwendbar sind.
- Schwierig bei Update-in-Place.
- Mehr Aufwand bei Redo. DB Operationen müssen ausgeführt werden.
- Ebenso Schwierigkeiten bei Undo, z.B. beim Undo von DELETE Operation, die viele Tupel gelöscht hat.
- Mischform: **Physiologisches Logging** Log-Einträge physisch, Änderungsoperationen wie Verschieben von Datensätzen innerhalb von Seite logisch.

Protokollierung von Änderungsoperationen

Struktur der Log-Records

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

LSN (Log Sequence Number)

- eine eindeutige Kennung des Log-Records
- LSNs müssen monoton aufsteigend vergeben werden,
- die chronologische Reihenfolge der Protokolleinträge kann dadurch ermittelt werden.
- z.B. Offset des Log-Records im Log-File

TransaktionsID

- die ID der Transaktion, die die Änderung durchgeführt hat.

PageID

- die ID der Seite, auf der die Änderungsoperation vollzogen wurde
- Wenn eine Änderung mehr als eine Seite betrifft, müssen entsprechend viele Log-Records generiert werden.

Protokollierung von Änderungsoperationen

Struktur der Log-Records

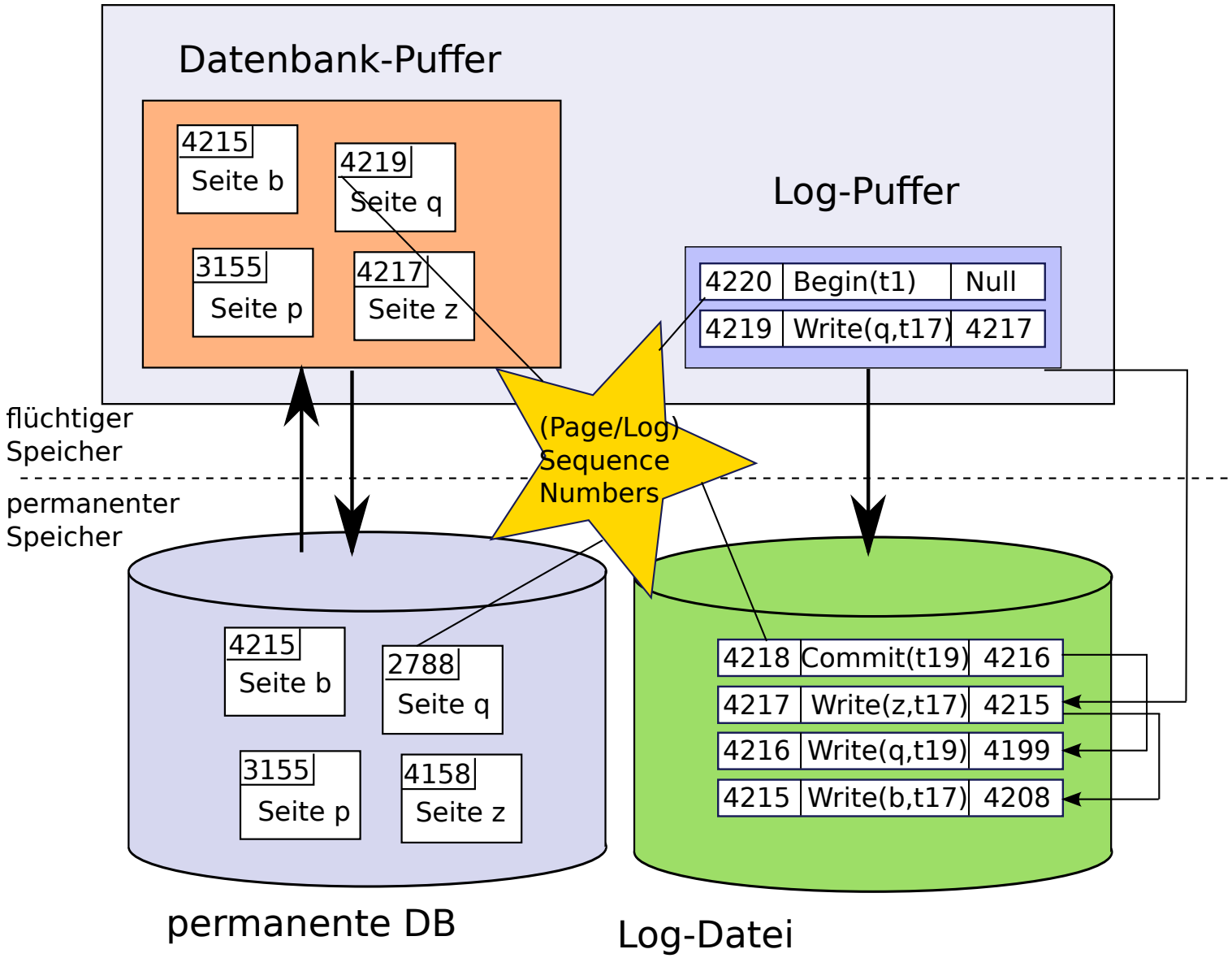
[LSN, TransaktionsID, PageID, Redo, Undo, PrevLSN]

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

Beispiel einer Log-Datei

Schritt	T_1	T_2	Log-Record [LSN, TA, PageID, Redo, Undo, PrevLSN]
1.	**BOT**		[#1,T_1, **BOT**, 0]
2.	r(A,a1)		
3.		**BOT**	[#2,T_2, **BOT**, 0]
4.		r(C,c2)	
5.	a1 := a1 - 50		
6.	w(A,a1)		[#3,T_1,PA,A-=50,A+=50,#1]
7.		c2 := c2 + 100	
8.		w(C,c2)	[#4,T_2,PC,C+=100,C-=100,#2]
9.	r(B,b1)		
10.	b1 := b1 + 50		
11.	w(B,b1)		[#5,T_1,PB,B+=50,B-=50,#3]
12.	**commit**		[#6,T_1,**commit**,#5]
13.		r(A,a2)	
14.		a2 := a2 - 100	
15.		w(A,a2)	[#7,T_2,PA,A-=100,A+=100,#4]
16.		**commit**	[#8,T_2,**commit**,#7]

Übersicht: Setup und Verwendung von LSNs



Seiten-LSN (PageLSN)

Speicherung der Seiten-LSN (PageLSN)

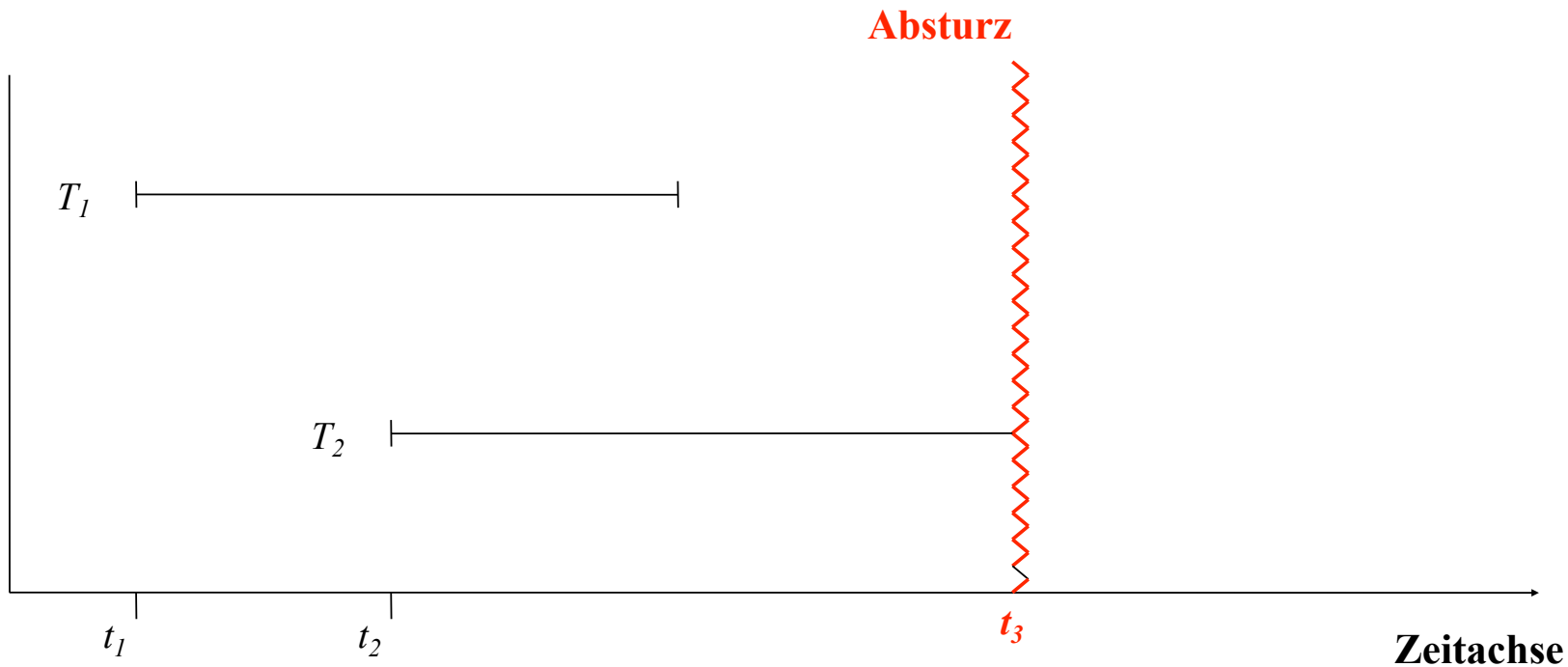
- Die "Herausforderung" besteht darin, beim Wiederanlauf zu entscheiden, welche Version diese Seite hat.
- Dazu wird auf jeder Seite die LSN des jüngsten diese Seite betreffenden Log-Eintrags gespeichert.

LSN zur Erkennung ob Before oder AfterImage

LSN des Log-Eintrags wird mitkopiert, wenn Seite auf Hintergrundspeicher propagiert wird. Daran kann man dann erkennen, ob für einen bestimmten Log-Eintrag das Before-Image oder After-Image in der Seite steht:

- Wenn die LSN der Seite einen kleineren Wert als die LSN des Log-Eintrags enthält, handelt es sich um das Before-Image.
- Ist die LSN der Seite größer oder gleich der LSN des Log-Eintrags, dann war schon das After-Image bzgl. der protokollierten Änderungsoperation auf den Hintergrundspeicher propagiert worden.

Wiederanlauf nach einem Fehler



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

Was wissen wir nach Absturz und was ist zu tun?

Verfügbare Informationen

- Wir sehen das Log. Wir kennen die WAL-Regel und die Commit-Regel, also es kann keine Änderung eine Seite auf der Festplatte geändert haben ohne dass wir die Änderungsoperation im Log sehen!
- Wir haben für die Seiten auf der Festplatte die LSN (PageLSN) der Aktion, die zuletzt die Seite geändert hat.

Was ist zu tun/analysieren?

- Welche der protokollierten Änderungen wurden bereits ausgeführt?
- Welche der Änderungen müssen ausgeführt werden, weil wir zwar den Log-Eintrag sehen, aber die Änderung vor Absturz noch nicht auf Festplatte geschrieben wurde?
- Welche der Änderungen müssen rückgängig gemacht werden?

Drei Phasen des Wiederanlaufs

1. Analyse (Bestimmung des Datenbankzustands)

- Die Log-Datei wird von Anfang bis zum Ende analysiert,
- Ermittlung der Winner-Menge von Transaktionen des Typs T1
- Ermittlung der Loser-Menge von Transaktionen der Art T2.

2. Redo (Vollständige Wiederholung der Historie)

- Alle protokollierten Änderungen werden in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht.
- Auch die Änderungen der Loser!

3. Undo (Entfernen der Loser-Änderungen)

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

Redo und PageLSN

- Log-Satz einer Änderung bezieht sich auf genau eine DB-Seite
- Die im Seitenkopf gespeicherte PageLSN entspricht der LSH des Log-Eintrags, welcher die zuletzt auf der Seite ausgeführte Änderung protokolliert.
- Eine Änderung, deren Log-Eintrag eine LSN aufweist, die kleiner oder gleich der PageLSN ist, befindet sich somit bereits in der Seite und braucht nicht wiederholt zu werden!
- Ansonsten muss Redo ausgeführt werden.

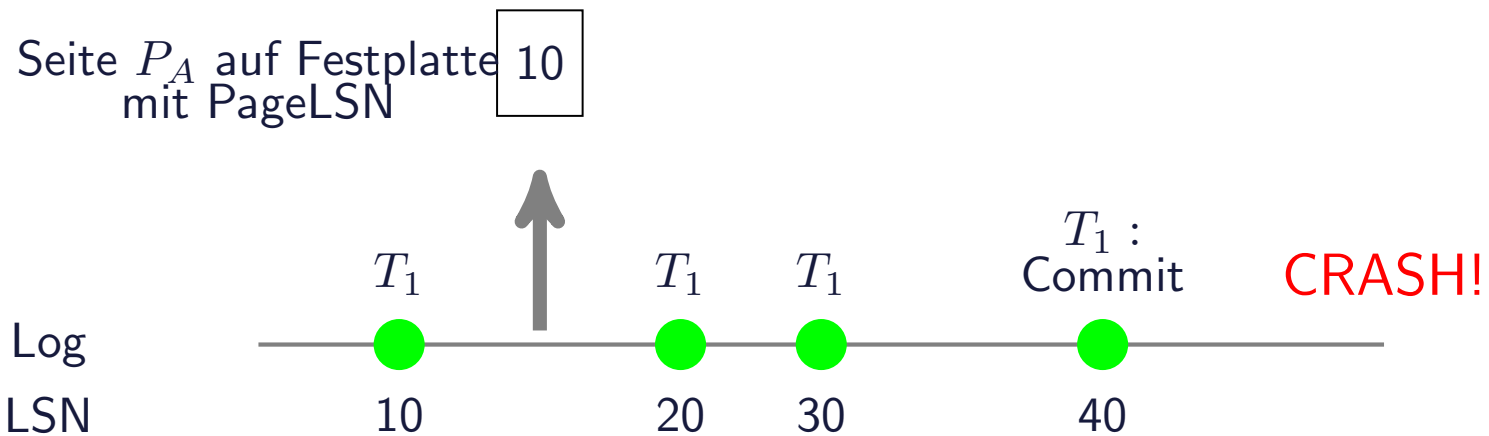
Für Log-Eintrag L und Seite B:

if LSN(L) > PageLSN(B) **then do**
    REDO (Änderung aus L)
    PageLSN(B) := LSN(L)
end

Achtung: Undo wird immer ausgeführt, egal welche LSN in der Seite steht. Weil entweder wurde Änderungs-Aktion ausgeführt vor Crash, oder beim Redo!

Beispiel

- Betrachten wir eine einzelne Transaktion T_1, welche einen Datensatz in einer Seite P_A bearbeitet.
- Die Punkte beschreiben Log-Einträge von Änderungsoperationen
- Nach dem Eintrag mit LSN 10 wird die Seite auf die Festplatte ausgeschrieben (genannt flush), z.B. weil sie verdrängt wurde.
- D.h. die PageLSN wird auf 10 gesetzt.
- Weitere Änderungsoperationen für LSN 20 und 30, aber kein Ausschreiben! Also Änderung nur im DB-Puffer.
- Wiederanlauf: Redo für Operationen mit LSN 20 und 30, aber Redo für LSN 10 nicht nötig.



Selektives vs. vollständiges Redo

- Selektives Redo: Nur Winner TA werden im Redo berücksichtigt.
- Vollständiges Redo: Alle TA (also auch die Loser) werden im Redo berücksichtigt. (Dies betrachten wir hier in der VL).

Verwendung der PageLSN-Werte zur Redo-Recovery erfolgt beim selektiven wie beim vollständigen Redo, wie beschrieben.

Unterschied bei Redo bzw. Undo der Verlierer-Transaktionen

- Selektives Redo funktioniert nur bei Seiten-Sperren
- Wir betrachten hier aber die Verwendung von Satzsperren!

Selektives vs. vollständiges Redo (2)

Angenommen: Satzsperren und selektives Redo. Was geht schief?

- T_1 und T_2 verändern unabhängig verschiedene Sätze in derselben Seite.
- Beim selektiven Redo werden nur Änderungen der Gewinner-Transaktion T_1 wiederholt.
- Also wird PageLSN von 10 auf 30 erhöht.
- Im Undo-Lauf wird dann die Änderung von T_2 zurückgesetzt, da aufgrund der PageLSN 30 davon ausgegangen wird, dann Änderung 20 in der Seite enthalten ist (ist sie aber nicht!).