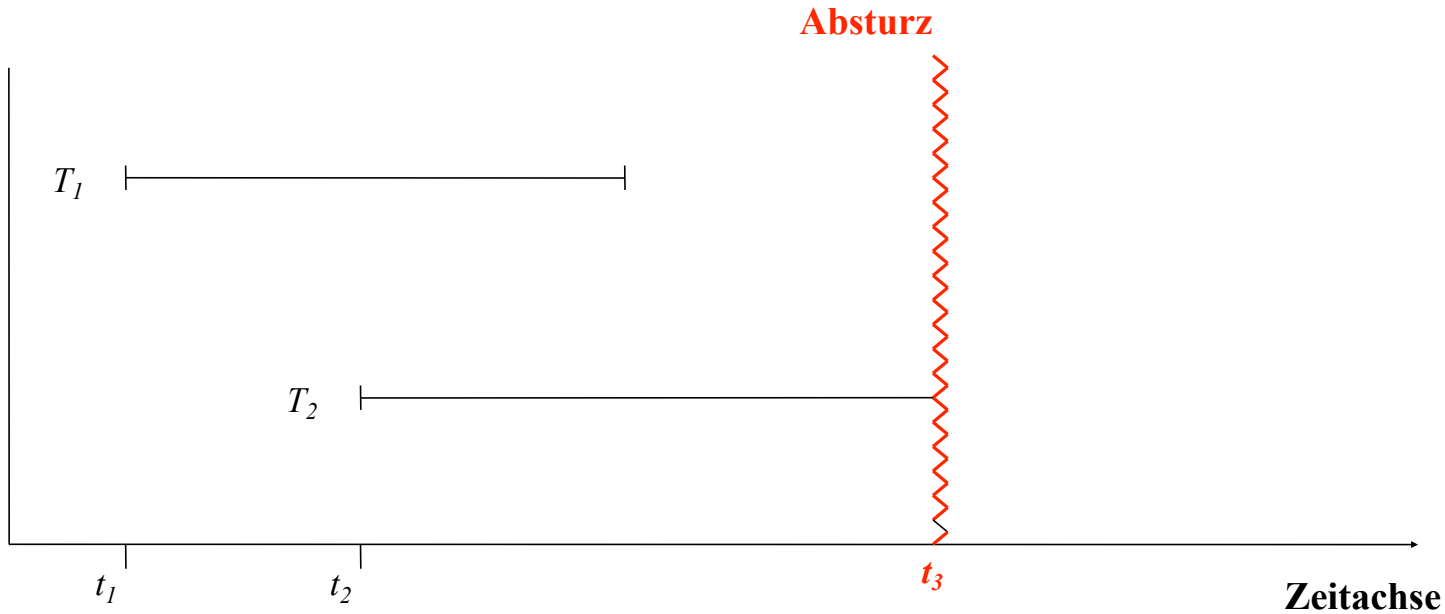
Datenbanksysteme

Wintersemester 2016/17

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

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)

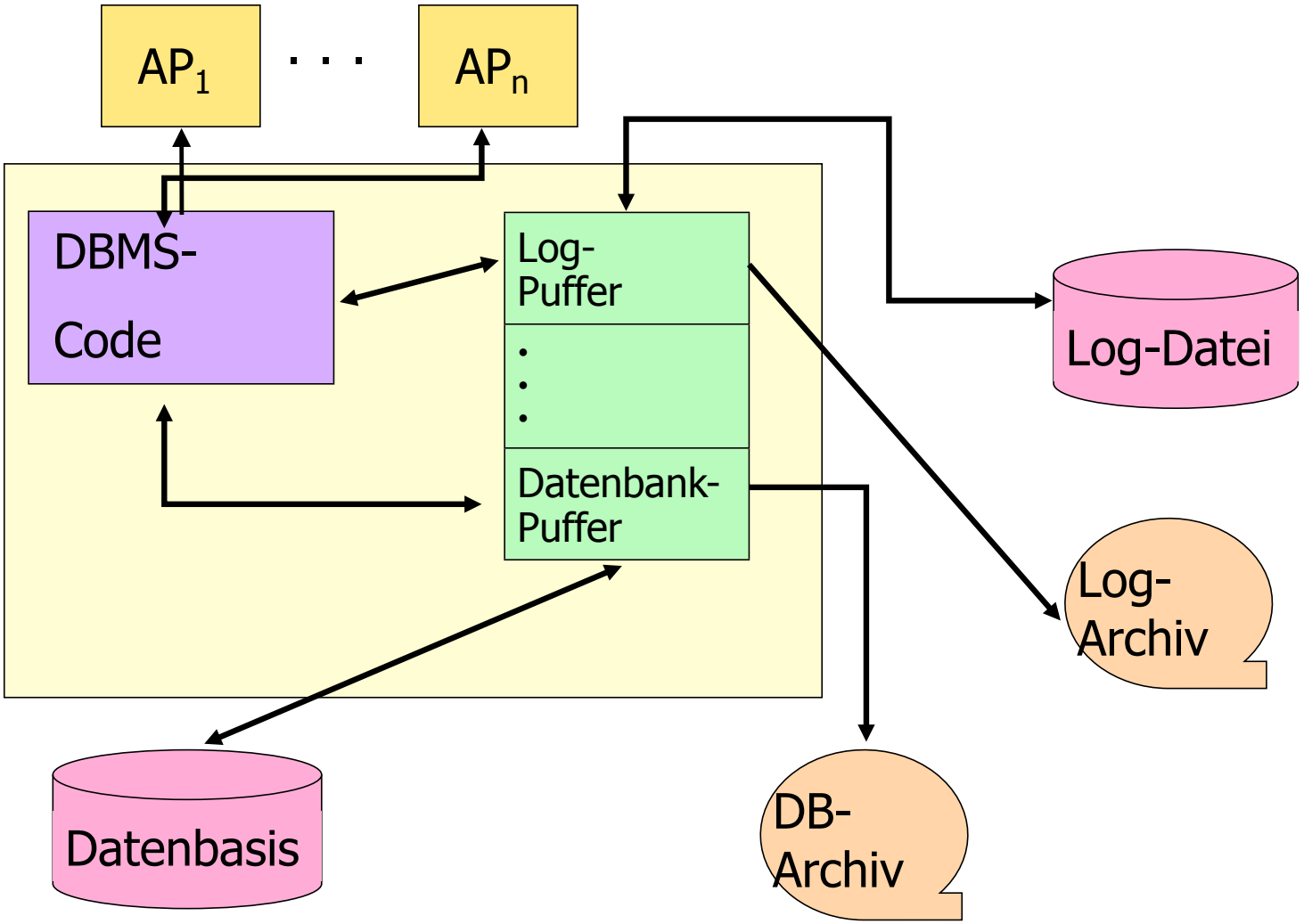
Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in die Log-Datei und das Log-Archiv ausgeschrieben werden.

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

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

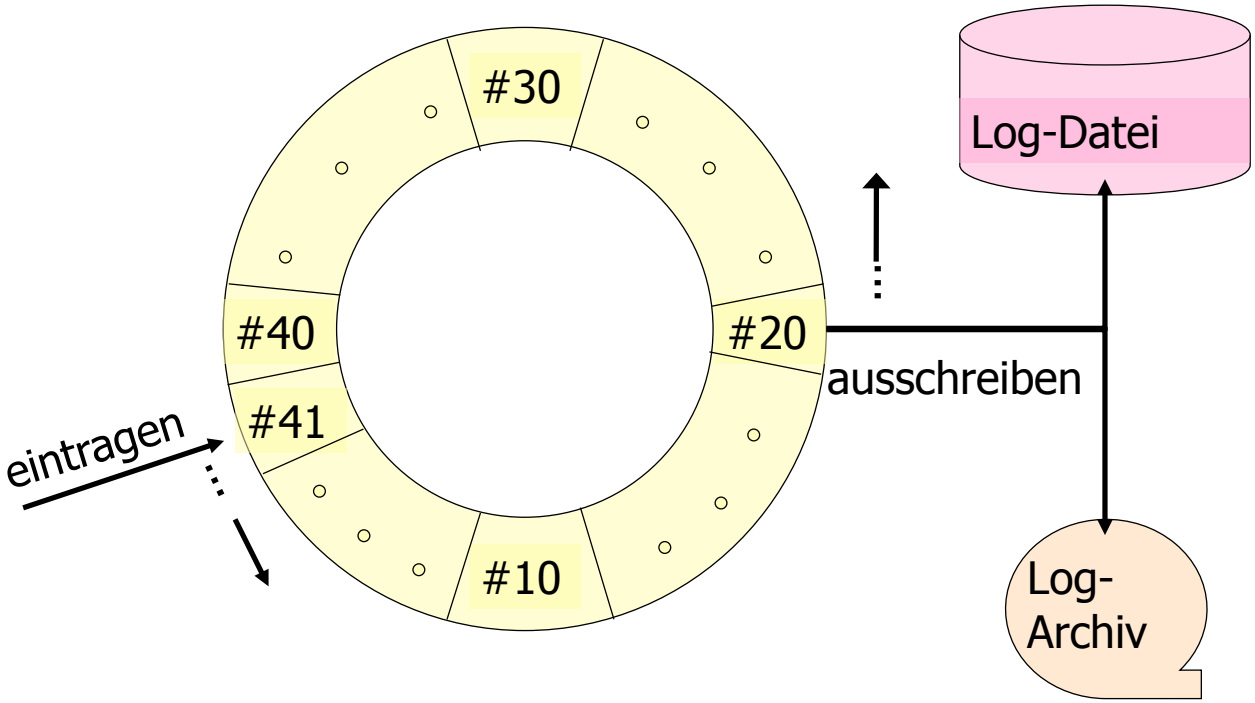
C. Mohan et al. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging.
`http://www.cs.berkeley.edu/~brewer/cs262/Aries.pdf`

Schreiben von Log-Informationen



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

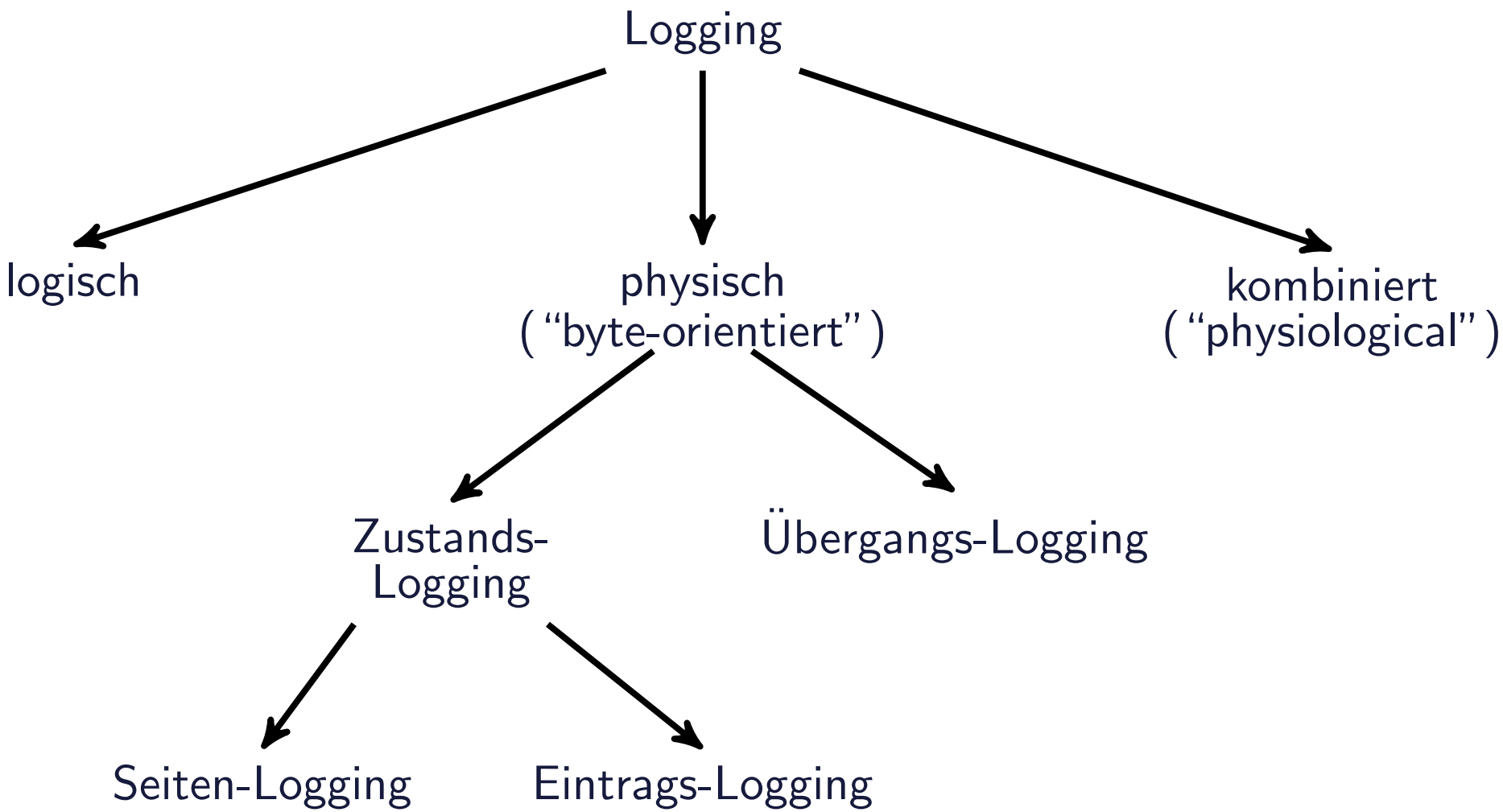
Anordnung des Log-Ringpuffers



- 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

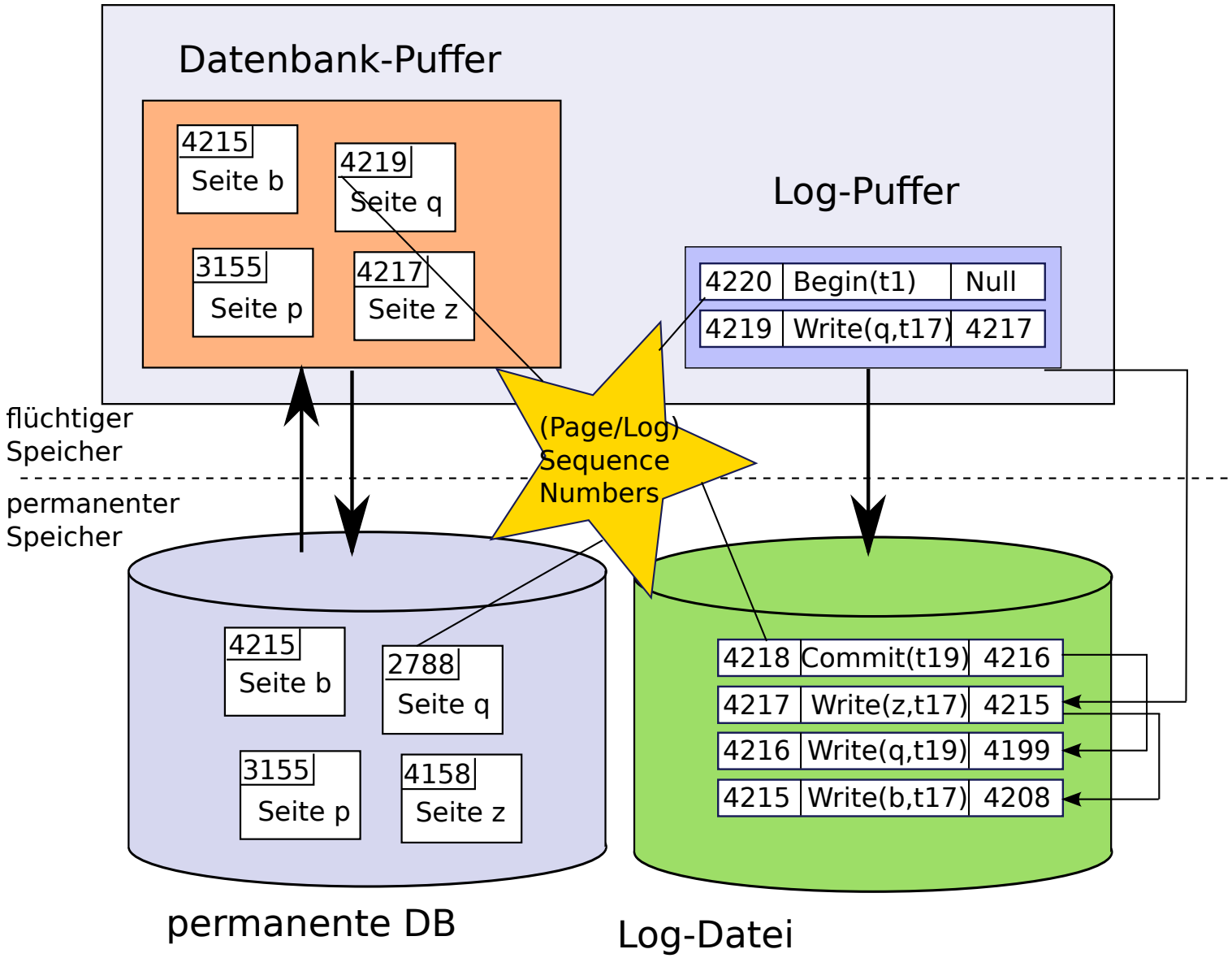
[LSN, TransaktionsID, PageID, Redo, Undo, PrevLSN]

- 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)

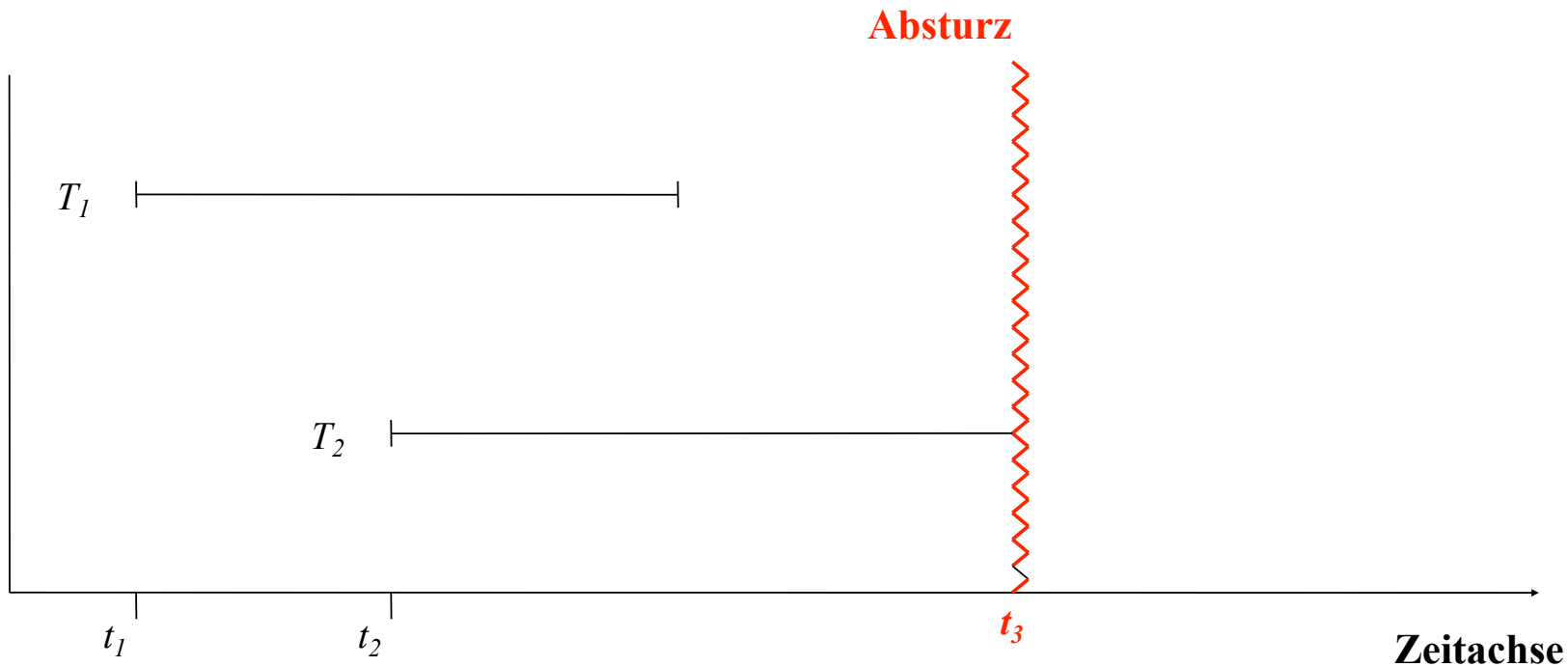
- Die "Herausforderung" besteht darin, beim Wiederanlauf zu entscheiden, welche Version diese Seite hat.
- Dazu wird auf jeder Seite die LSN des jüngsten diese Seite betreffenden Log-Eintrags gespeichert.

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.

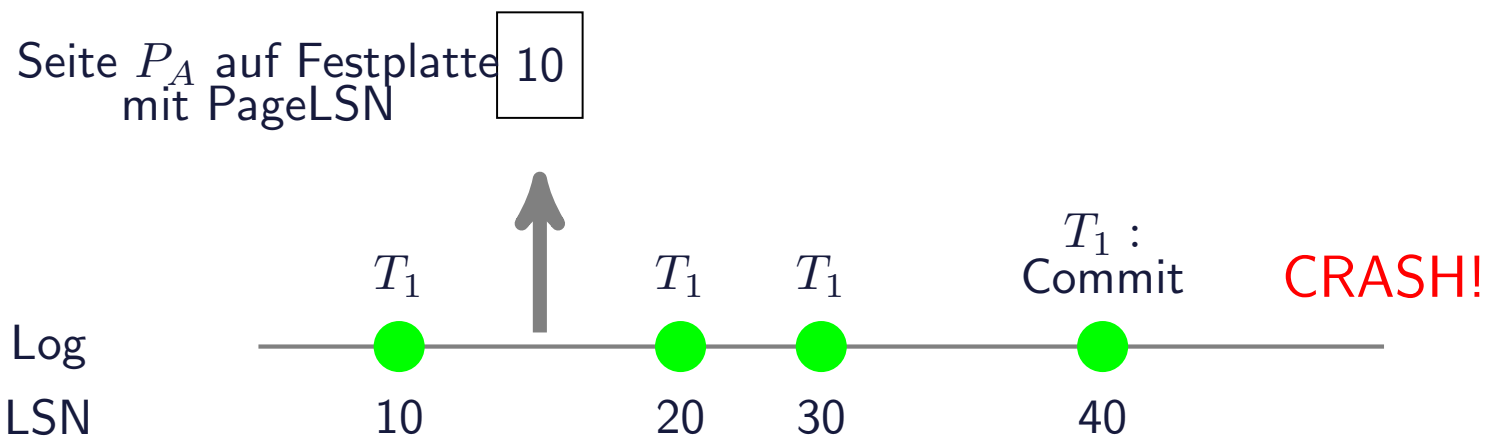
Für Log-Eintrag L und Seite B:

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.

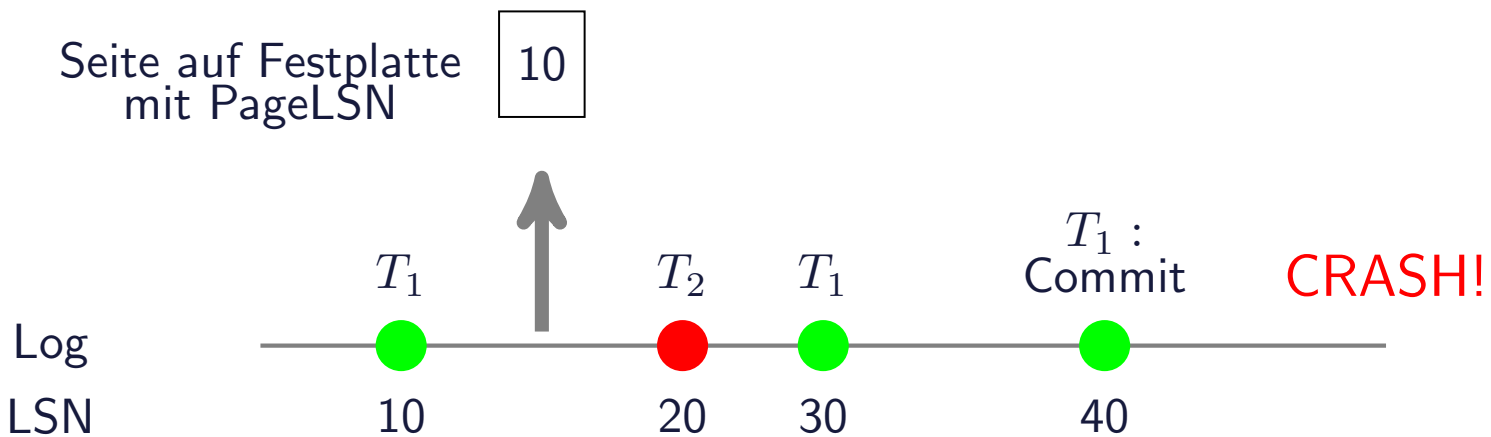
Unterschied bei Redo bzw. Undo der Verlierer-Transaktionen

- 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!).



Fehlertoleranz (Idempotenz) des Wiederanlaufs

Zu jeder ausgeführten Aktion a gilt:

- $undo(undo(...(undo(a))...)) = undo(a)$
- $redo(redo(...(redo(a))...)) = redo(a)$

Achtung: auch während der Recoveryphase kann das System abstürzen!

- Recovery funktioniert auch dann!

Fehlertoleranz (Idempotenz) des Wiederanlaufs (2)

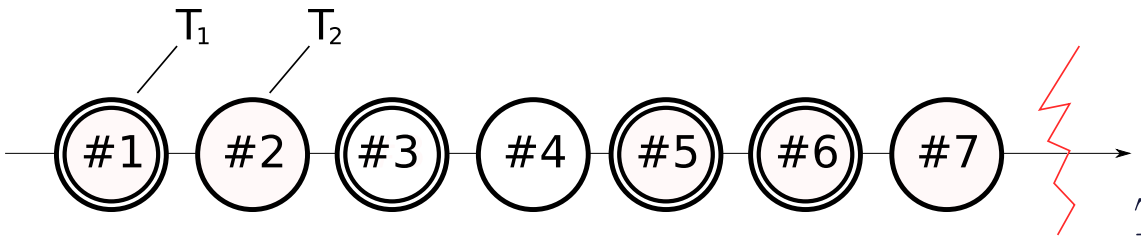
Der Redo Fall:

- LSN des Log-Records, für den ein Redo (**tatsächlich**) ausgeführt wird, wird in der Seite (PageLSN!) eingetragen
- Dadurch: nach Absturz nicht "versehentlich" nochmal ausgeführt

Der Undo Fall:

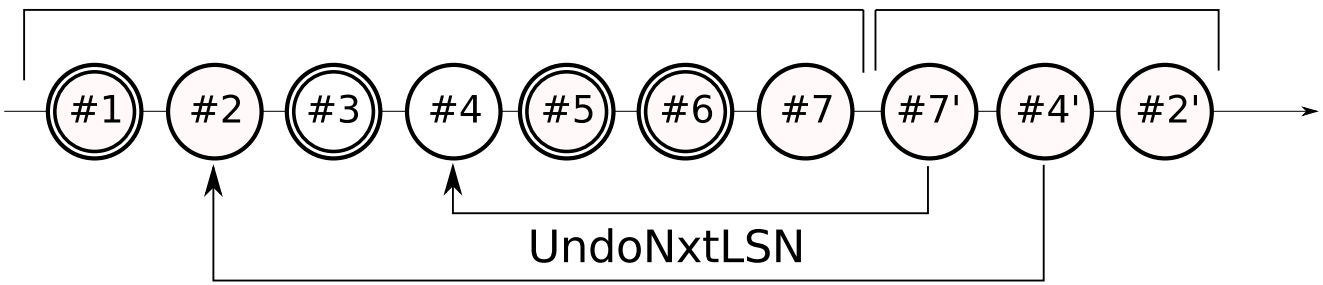
- Kompensations-Protokolleinträge (**CLR**, compensation log record)
- Für jede Undo-Operation wird ein CLR angelegt.
- Auch hier: LSN des CLR-Log-Eintrags wird als PageLSN übernommen.

Kompensationseinträge im Log



T_1 = Gewinner, T_2 = Verlierer

Wiederanlauf und Log



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

- $\#7'$ ist CLR für $\#7$
- $\#4'$ ist CLR für $\#4$
- Achtung: Natürlich müssen alle LSN fortlaufend sein (hier nur zur Veranschaulichung z.B. $4'$ genannt)

Logeinträge nach abgeschlossenem Wiederanlauf

[#1, T_1, BOT, 0]
[#2, T_2, BOT, 0]
[#3,T_1, PA, A-=50, A+=50, #1]
[#4,T_2, PC, C+=100, C-=100, #2]
[#5,T_1, PB, B+=50, B-=50, #3]
[#6,T_1, commit, #5]
[#7,T_2, PA, A-=100, A+=100,#4]
<#7‘,T_2, PA, A+=100, #7,#4>
<#4‘‘,T_2, PC, C-=100, #7’,#2>
<#2‘,T_2,-,-,#4‘,0>

Logeinträge nach abgeschlossenem Wiederanlauf

CLRs sind durch spitze Klammern $< \ldots >$ gekennzeichnet. Und haben folgenden Aufbau:

- LSN
- ID der Transaktion
- betroffene Seite
- Redo-Information
- PrevLSN
- UndoNxtLSN (Verweis auf die nächste rückgängig zu machende Änderung)

Anmerkungen

- **Die Redo-Anweisung eines CLR entspricht der während der Undo-Phase des Wiederanlaufs ausgeführten Undo-Operation.**
- CLRs enthalten keine Undo-Information. Warum?

Idempotenz des Wiederanlaufs

[#1, T_1, BOT, 0]
[#2, T_2, BOT, 0]
[#3,T_1, PA, A-=50, A+=50, #1]
[#4,T_2, PC, C+=100, C-=100, #2]
[#5,T_1, PB, B+=50, B-=50, #3]
[#6,T_1, commit, #5]
[#7,T_2, PA, A-=100, A+=100,#4]
<#7‘,T_2, PA, A+=100, #7,#4>
<#4“,T_2, PC, C-=100, #7’,#2>
<#2‘,T_2,-,-,#4‘,0>

- Was passiert bei Wiederanlauf?
- Log wie links gegeben.
- **Redo Phase** schaut ob alle “normalen” Änderungen durchgeführt sind und ebenso ob die Kompensationen ausgeführt wurden.
- **Undo Phase** sieht, dass UndoNxtLSN von $\#2'$ Null ist und merkt, dass T_2 vollständig rückgängig gemacht wurde.

Idempotenz des Wiederanlaufs (2)

Jetzt: Log Datei nur bis (einschließlich) $\#7'$

[#1, T_1, BOT, 0]
[#2, T_2, BOT, 0]
[#3,T_1, PA, A-=50, A+=50, #1]
[#4,T_2, PC, C+=100, C-=100, #2]
[#5,T_1, PB, B+=50, B-=50, #3]
[#6,T_1, commit, #5]
[#7,T_2, PA, A-=100, A+=100,#4]
<#7',T_2, PA, A+=100, #7,#4>

- Was passiert bei Wiederanlauf?
- **Redo Phase** schaut ob alle "normalen" Änderungen durchgeführt sind und ebenso ob die Kompensation $\#7'$ ausgeführt wurde.

- In **Undo Phase** wird offensichtlich $\#7'$ nicht rückgängig gemacht, da es ja ein CLR ist. Sondern es wird der UndoNxtLSN-Zeiger benutzt und gemerkt, dass die nächste Operation die rückgängig gemacht werden muss Operation $\#4$ ist. Die in LSN 4 stehende Undo-Operation wird ausgeführt, wobei der CLR $\#4'$ angelegt wird. In $\#4$ steht PrevLSN-Veweis auf $\#2$, die als nächstes kompensiert wird und mit $\#2'$ protokolliert wird.

Lokales Zurücksetzen einer Transaktion

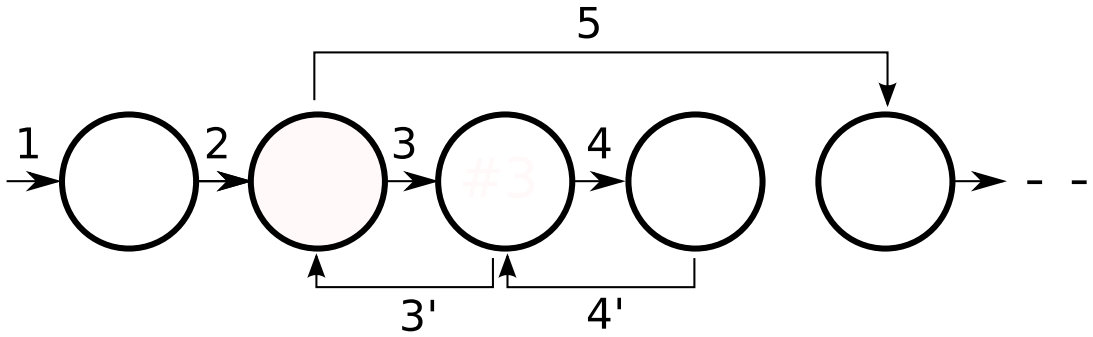
Isoliertes Zurücksetzen einer einzelnen Transaktion

- **Analog zum Rücksetzen von Verlierer-Transaktionen nach Wiederanlauf**
- Abarbeiten der zur Transaktion gehörenden Log-Einträge in chronologisch umgekehrter Reihenfolge.
- Dies kann nun aus dem Log-Puffer geschehen, da hier davon ausgegangen wird, dass der Hauptspeicher intakt ist.
- Mit den PrevLSN kann man sehr effizient zurücklaufen
- Und Undo-Operationen ausführen
- Aber: Vorher noch mit Compensation Log Record protokollieren!

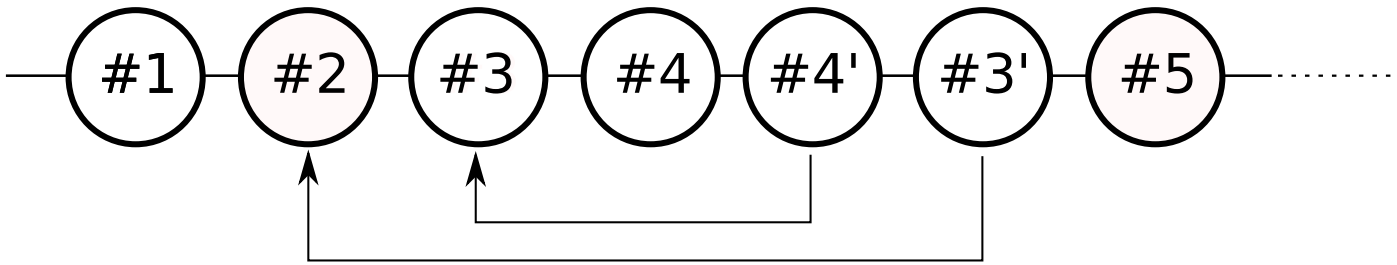
Lokales Zurücksetzen einer Transaktion (2)

Z.B. partielles Zurücksetzen einer Transaktion

Historie



Log



- Schritte 3 und 4 werden zurückgenommen
- notwendig für die Realisierung von **savepoints** einer TA

Lokales Zurücksetzen einer Transaktion (3)

- Zurücksetzen durch Einfügen von Operationen, die die Änderungen rückgängig machen.
- Also "kompensieren".

Vorgehen

- Transaktion "sagt": abort
- Log-Eintrag für abort Operation wird angelegt
- Kompensations-Operationen (Undo) der TA werden ausgeführt und (natürlich!) protokolliert, mit CLRs.
- Auch wird wieder die LSN des CLR-Eintrags als PageLSN übernommen!

Lokales Zurücksetzen einer Transaktion (4)

Absturz während Abort

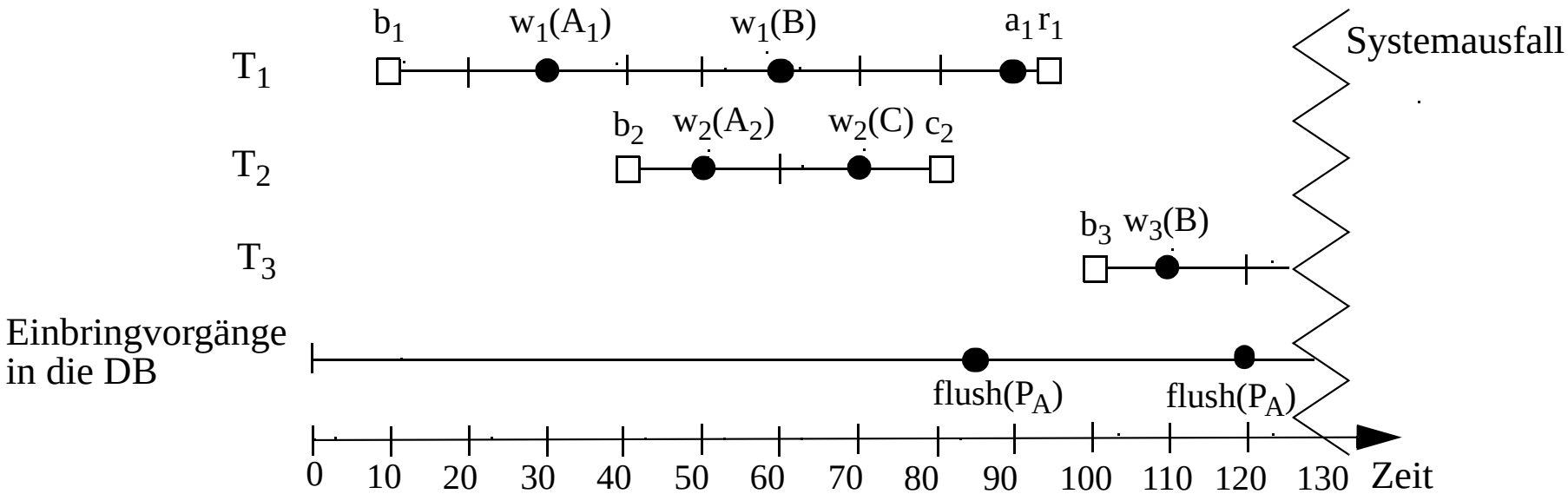
- Die CLRs werden im Falle eines Wiederanlaufs in der Redo-Phase überprüft ob bereits ausgeführt.
- In der Undo-Phase werden, wenn nötig, noch nicht protokollierte Kompensationen (wie bei allen Verlierer-TA) rückgängig gemacht.

Rollback Log-Eintrag

- Sozusagen als Commit (Festschreiben) einer Transaktion die Abort ausgeführt hat
- Mit Log Eintrag `rollback` (oder r)
- (Komplett) zurücksgesetzte Transaktion wird dann als Gewinner betrachtet, bei Wiederanlauf.
- Alternativ (implizit): UndoNextLSN=Null?
- Literatur nicht immer eindeutig bzw. konkret diesbzgl.

Komplexeres Beispiel

- Transaktionen T_1, T_2 und T_3
- Datensatz A in Seite P_A, B in P_B und C in P_C
- Zwei Mal Auslagerung ("flush')' der Seite P_A
- Wie sieht Log-Puffer, Log-Datei, DB-Puffer und Seite auf Festplatte aus?



Komplexeres Beispiel

Zeit	Aktion	DB-Puffer	DB-Eintrag	Log-Puffer	Log-Datei
10	b_1				
30	$w_1(A_1)$				
40	b_2				
50	$w_2(A_2)$				
60	$w_1(B)$				
70	$w_2(C)$				
80	c_2				
85	flush(P_A)				
90	a_1				
91					
92					
95	r_1				
100	BOT_3				
110	$w_3(B)$				
120	flush(P_A)				

Absturzpunkt

Übersicht Recovery: Einfacher Redo-Winners Ansatz

Im Folgenden eine Übersicht zu verschiedenen Ansätzen / Konfigurationen des Crash-Recovery

- Annahme: Volle (physikalische) Before- und After-Images
- Sperren von Seiten
- Kein Rollback
- Wie sieht Redo aus? Einfaches Anwenden der Redo Informationen der Gewinner Transaktionen
- Wieso reicht das aus? Letztes After-Image "gewinnt".
- Wegen Locking werden Verlierer-TA (ohne Konflikte) rein am Ende vor Crash ausgeführt.
- Undo der Verlierer Transaktionen.
- Keine PageLSN, keine CLRs nötig.

Übersicht Recovery: Nun (Physio)logisches Logging beim Redo-Winners Ansatz

- Es werden nicht komplette physikalische Images (byte[]) gespeichert im Log
- Sondern Beschreibung wie Änderungsoperation gearbeitet hat (A+=10 oder diff)
- Kompakter, aber Redo nicht mehr idempotent!
- Also nicht mehr (wie auf vorheriger Folie) einfaches Anwenden der Gewinner Redos
- Sondern: Wir brauchen nun PageLSNs
- Was ist mit Idempotenz der Undos? PageLSNs funktionieren auch hier (Achtung, Annahme: Seitensperren)

Übersicht Recovery: Nun auch: Rollback beim Redo-Winners Ansatz

- Wo ist das Problem?
- Zurücksetzen von Transaktionen führt dazu, dass nun nicht mehr die Verlierer "am Ende" des Logs (der Ausführung) stehen.
- Lösung: Kompensations-Operationen bei abort.
- Vollständig abgebrochene TA werden zu Gewinnern.
- Was passiert bei Absturz während Rollback?
- Man braucht CLRs um Idempotenz zu gewährleisten.

Übersicht Recovery: Nun: Sperrgranularität: Datensatz

- Redo-Winners (Selektives Redo)? Funktioniert nicht (siehe Erklärung weiter vorne)
- Also: vollständiges Redo (Redo-History): Sowohl Gewinner als auch Verlierer werden nachvollzogen

Wer es noch genauer wissen möchte, inkl. Pseudocode und Korrektheitsbeweise, Details gibt es im Buch von Weikum und Vossen.