Datenbankanwendung

Wintersemester 2014/15

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

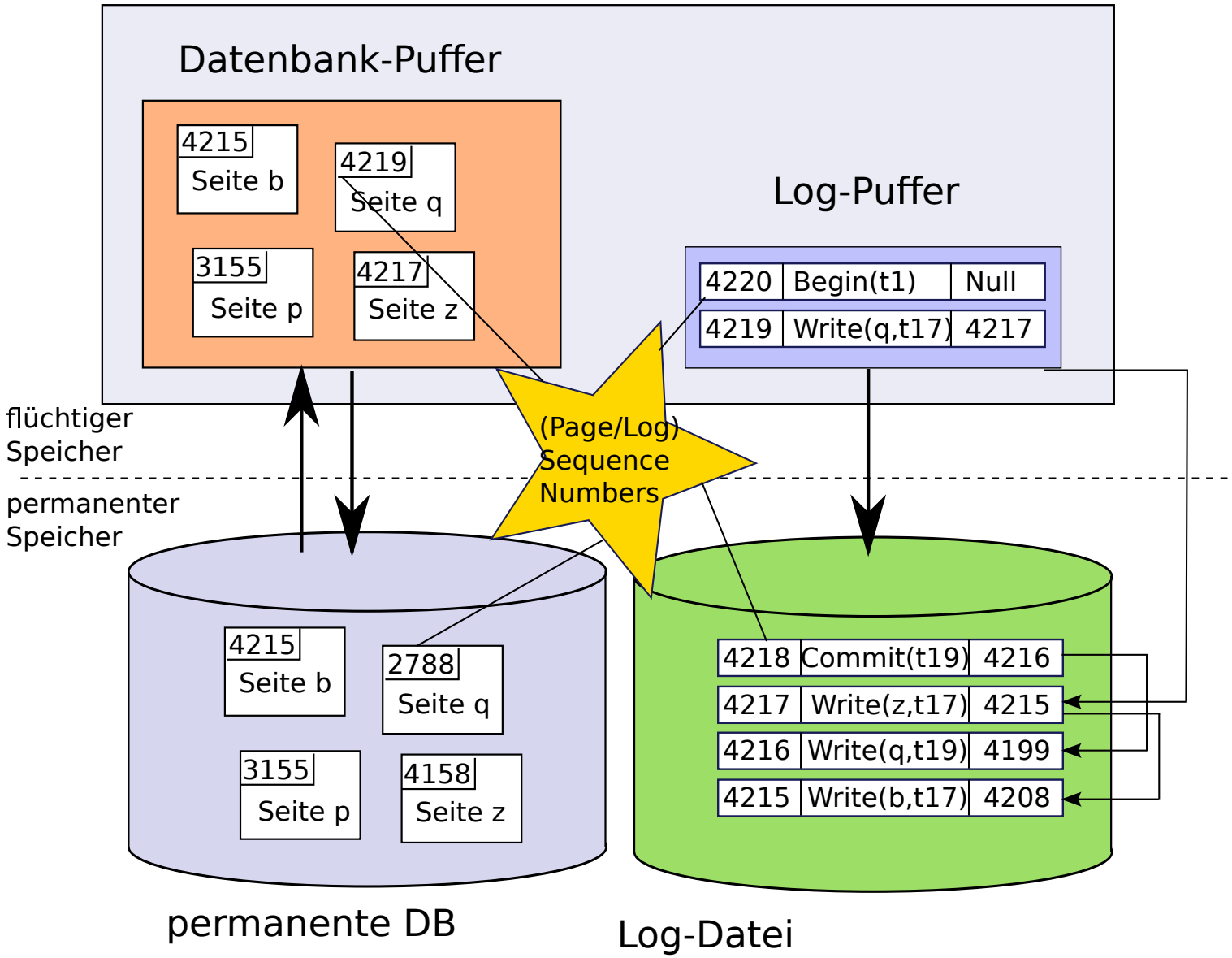
smichel@cs.uni-kl.de

Aus gegebenem Anlass ...

Quelle: bild.de, 27.01.2015

- Ausfall hat (angeblich) ca. 1 Stunde gedauert
- Nehmen wir an, dass dies ein Mal im Jahr geschieht.
- Wie groß ist die Availability? Wie viele "Neunen"?

Übersicht: Setup und Verwendung von LSNs



Wiederholung: Zugrunde gelegte Systemkonfiguration

- **steal**
  "dreckige Seiten" können in die Datenbank (auf Platte) geschrieben werden. Wir brauchen also **Undo**!
- **¬force**
  geänderte Seiten sind **möglicherweise** noch nicht auf die Platte geschrieben. Wir brauchen also **Redo**!
- **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.

Wiederholung: Drei Phasen des Wiederanlaufs

1. Analyse (Bestimmung des Datenbankzustands)

- Die Log-Datei wird von Anfang bis zum Ende analysiert.
- Ermittlung der Gewinner-Transaktionen.
- Ermittlung der Verlierer-Transaktionen.

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.

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 Transaktonen
- 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 VL 19, vor einer Woche)
- 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, oder in diesen Folien, die recht genau dem Buch folgen:
`http://www3.informatik.tu-muenchen.de/old/wwwdb/teaching/ws1213/ts/chapter13.pdf`

Sicherungspunkte (=checkpoints)

Achtung: checkpoint ≠ savepoint (Terminologie)

Beobachtung

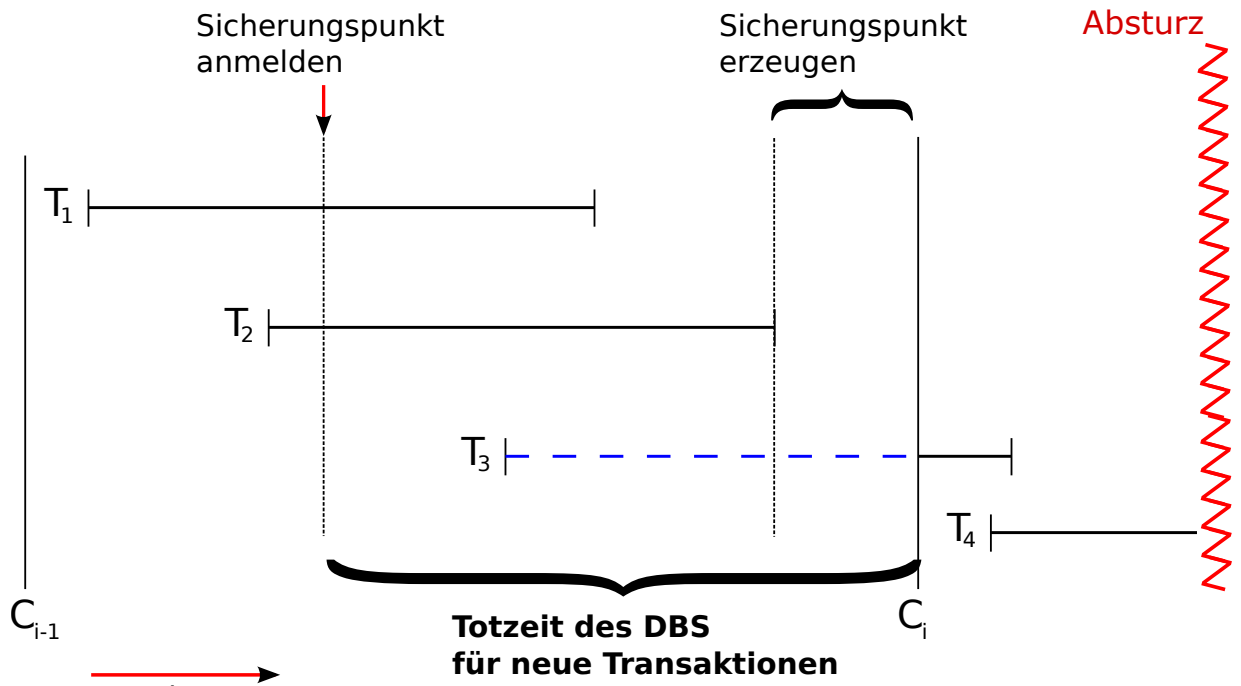
- Mit zunehmender Betriebszeit des Datenbanksystems wird Wiederanlauf immer langwieriger, da die Log-Datei immer umfangreicher wird.

Sicherungspunkte

- Idee: "Markiere" im Log Zeitpunkt, über die man beim Wiederanlauf nicht hinausgehen muss.
- Verschiedene Ansätze.
  - (globale) transaktionskonsistente Sicherungspunkte
  - aktionskonsistente Sicherungspunkte und
  - unscharfe (fuzzy) Sicherungspunkte
- Achtung: "cut-off" Punkt ist nicht unbedingt Zeitpunkt an dem Sicherungspunkt angelegt wird, kann auch älter sein; kleinste noch benötigte LSN wird angegeben.

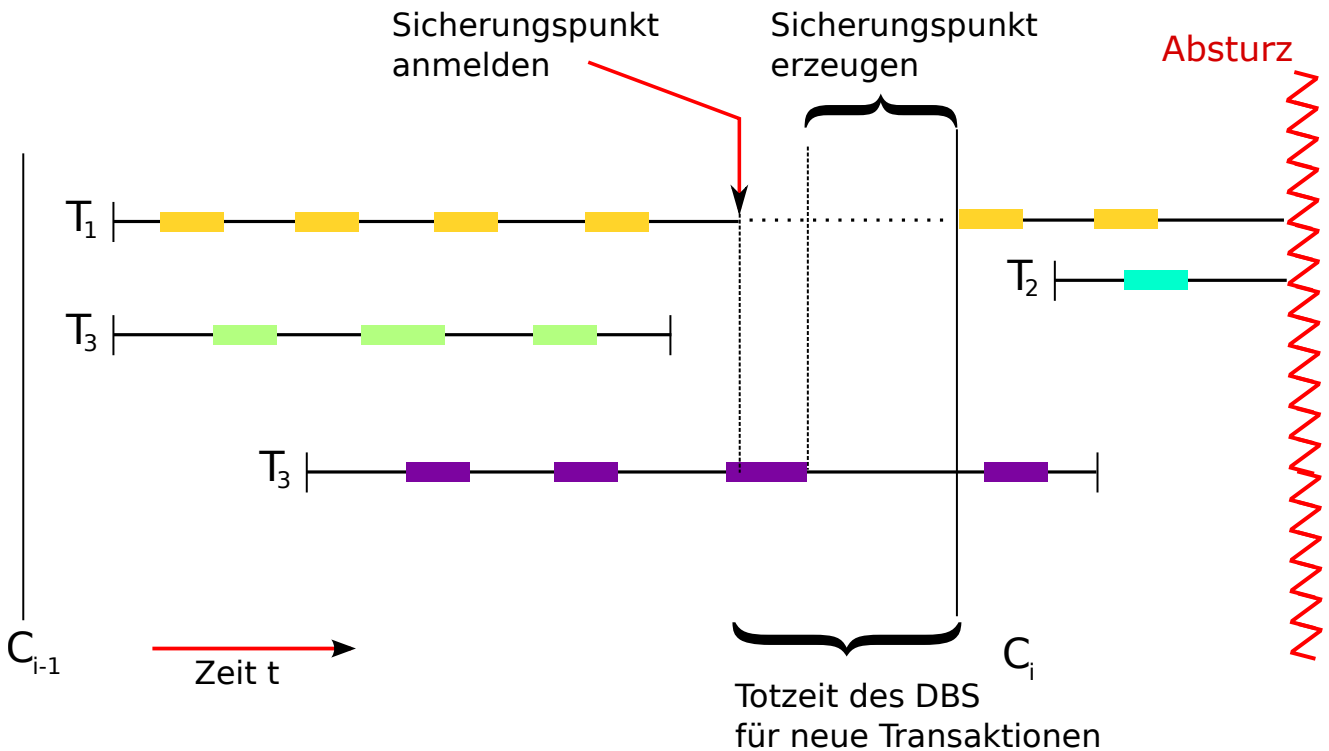
Transaktionskonsistente Sicherungspunkte (TCC)

- Transaction-Consistent Checkpoint (TCC)
- Neu ankommende TA (hier T_3) müssen warten. Laufende TA (hier T_1 und T_2) werden zu Ende ausgeführt.
- Dann schreiben aller modifizierten Seiten auf Hintergrundspeicher.
- Redo und Undo höchstens für TA nach Sicherungspunkt nötig.
- Führt i.a. zu einer sehr langen Totzeit des Systems für den Änderungsbetrieb.



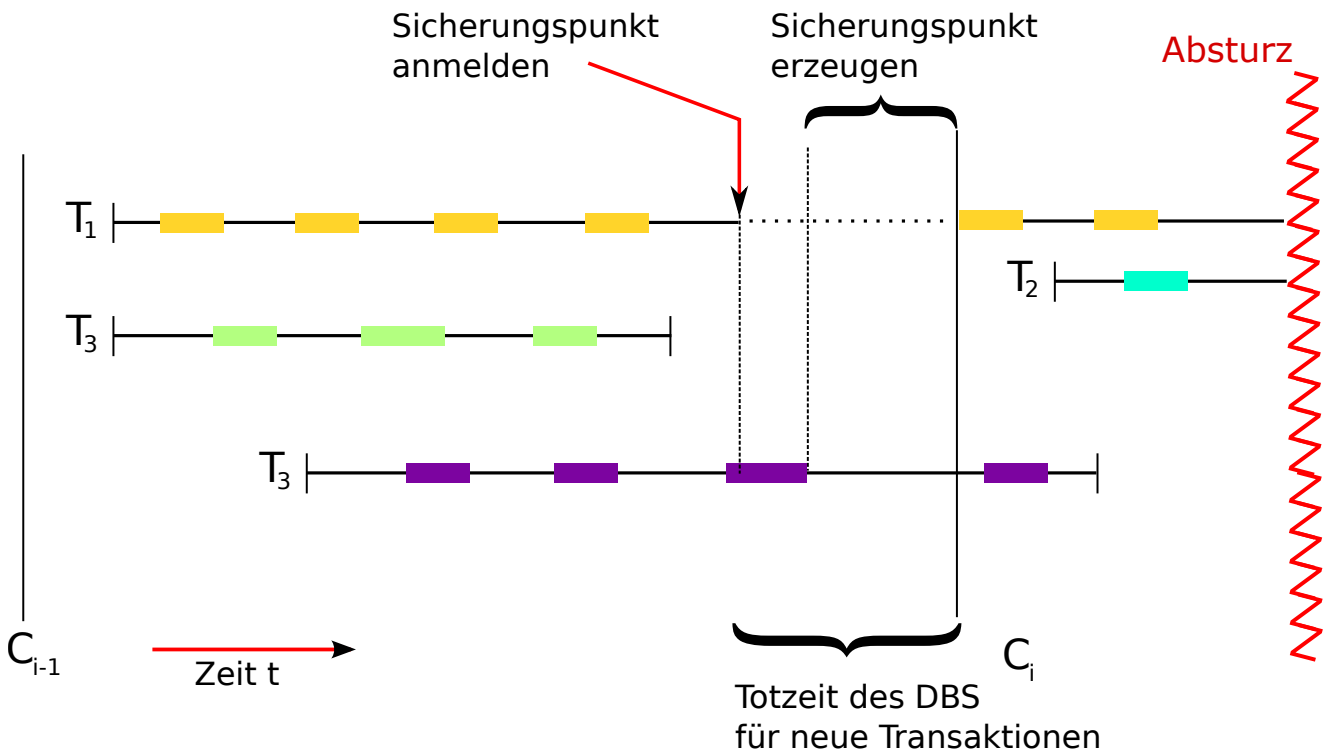
Aktionskonsistente Sicherungspunkte (ACC)

- Action-Consistent Checkpoints (ACC)
- Transaktionsausführung relativ zu einem aktionskonsistenten Sicherungspunkt und einem Systemabsturz.
- Aktive Änderungsoperationen (hier, bei T_4) werden noch ausgeführt und ausgeschrieben.
- Neue Änderungsoperationen müssen warten (hier, bei T_1)



Aktionskonsistente Sicherungspunkte (2)

- Man braucht keine Redo-Informationen, die älter sind als der Zeitpunkt des Schreibens auf Hintergrundspeicher
- Aber man braucht u. U. Undo-Informationen
- Also: Kleinste LSN aller zum Zeitpunkt noch aktiven LSN speichern (=MinLSN) + Liste aller noch aktiven TA



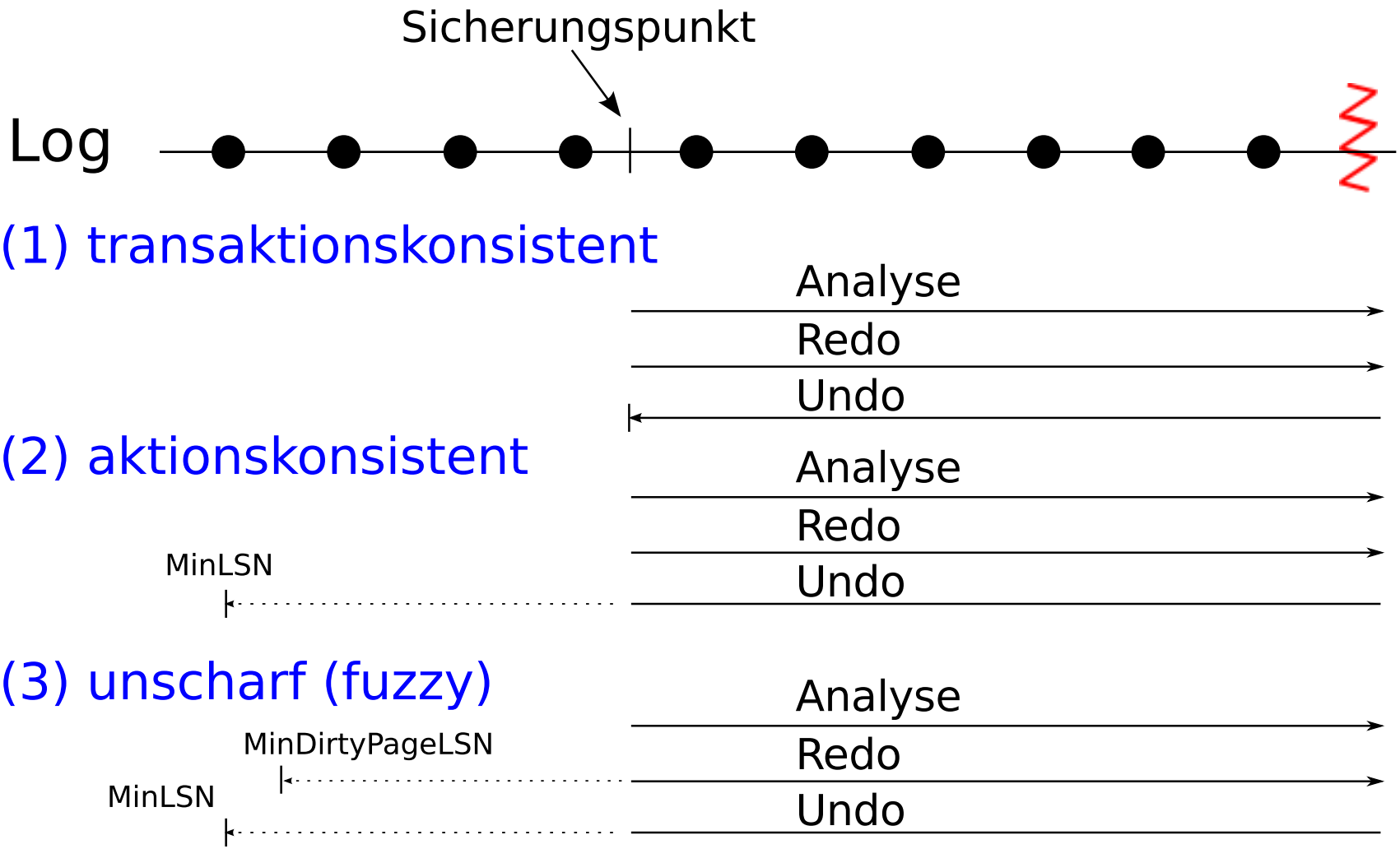
Unscharfe (fuzzy) Sicherungspunkte

- Idee: modifizierte Seiten werden **nicht** ausgeschrieben
- Sondern nur deren Kennung (PageID)
- Und die älteste LSN, die diese Seite hat dreckig werden lassen werden notiert (in Log-Datei).

Terminologie:

- **MinDirtyPageLSN:**
  - Die kleinste LSN, deren Änderungen noch nicht ausgeschrieben wurde
  - D.h. die älteste Änderungsoperation, die eine Seite geändert hat (hat "dreckig" werden lassen).
  - Die kleinste all dieser LSNs wird MinDirtyPageLSN genannt.
  - D.h. bis dahin muss Redo Phase laufen.
- **MinLSN:**
  - Die kleinste LSN der zum Sicherungszeitpunkt aktiven TA
  - Erste/älteste Änderungsoperation aller Loser-TA
  - Die älteste Änderung, die rückgängig gemacht werden muss (im Undo)

Zusammenfassung der drei Arten von Sicherungspunkten



Anmerkung zur Analyse-Phase

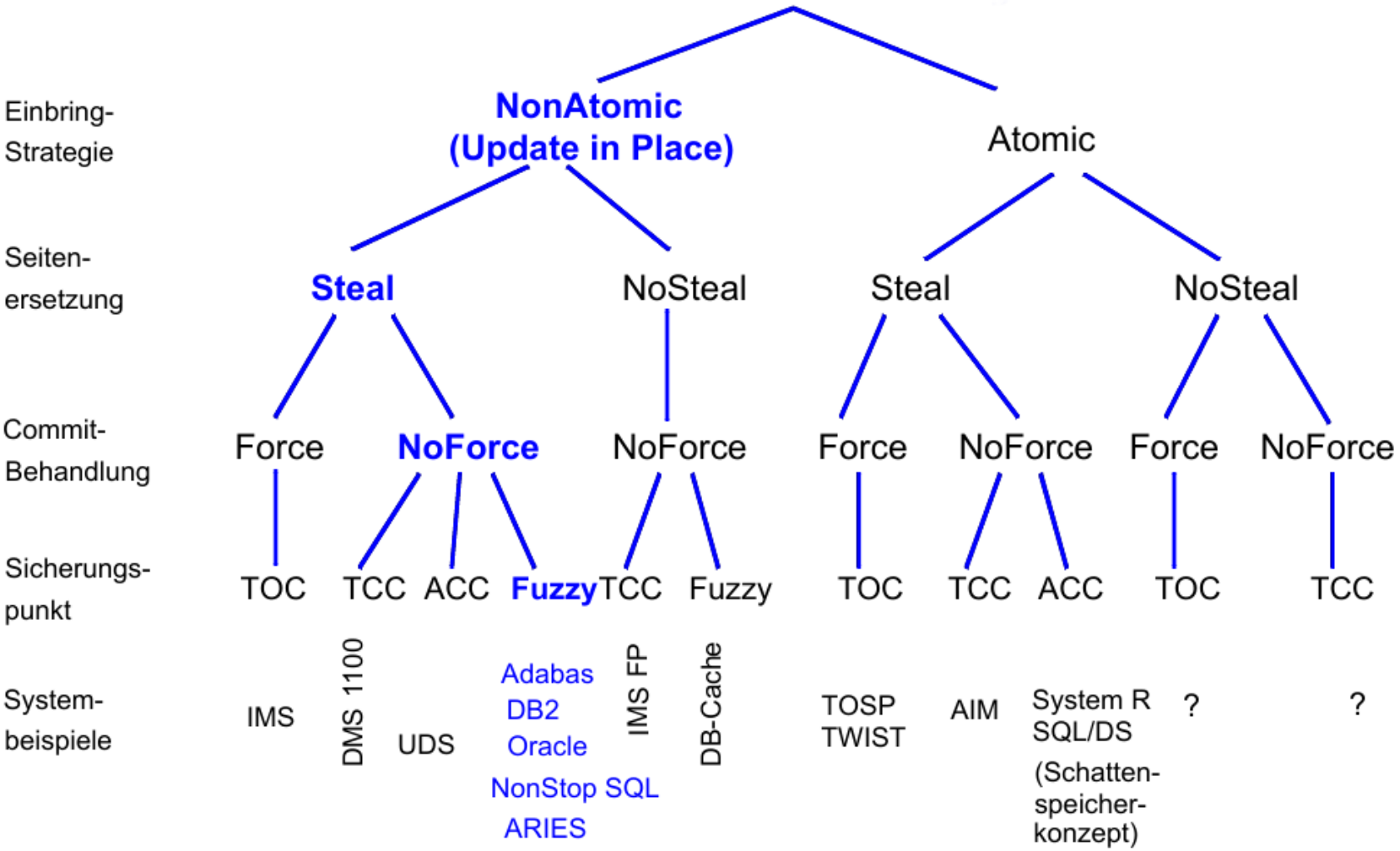
- Analyse-Phase ist mehr oder weniger optional bei dem vollständigen Redo (Redo-History)
- Für Optimierungen der Redo-Phase ist die Analyse-Phase allerdings notwendig (z.B. Prefetching von bekannten DirtyPages zum Redo-Zeitpunkt)
- Ob nun 2 oder 3 Phasen günstiger sind hängt von einigen Dingen ab, z.B. Häufigkeit der Checkpoints.

Weitere Anmerkungen

- Relativ gesehen gibt es sehr wenige Rollbacks und TAs, die zurückgenommen (Undo) werden müssen.
- Anzahl nebenläufiger TA ist obere Schranke für Verlierer-Transaktionen.

Klassifikation der DB Recovery-Verfahren

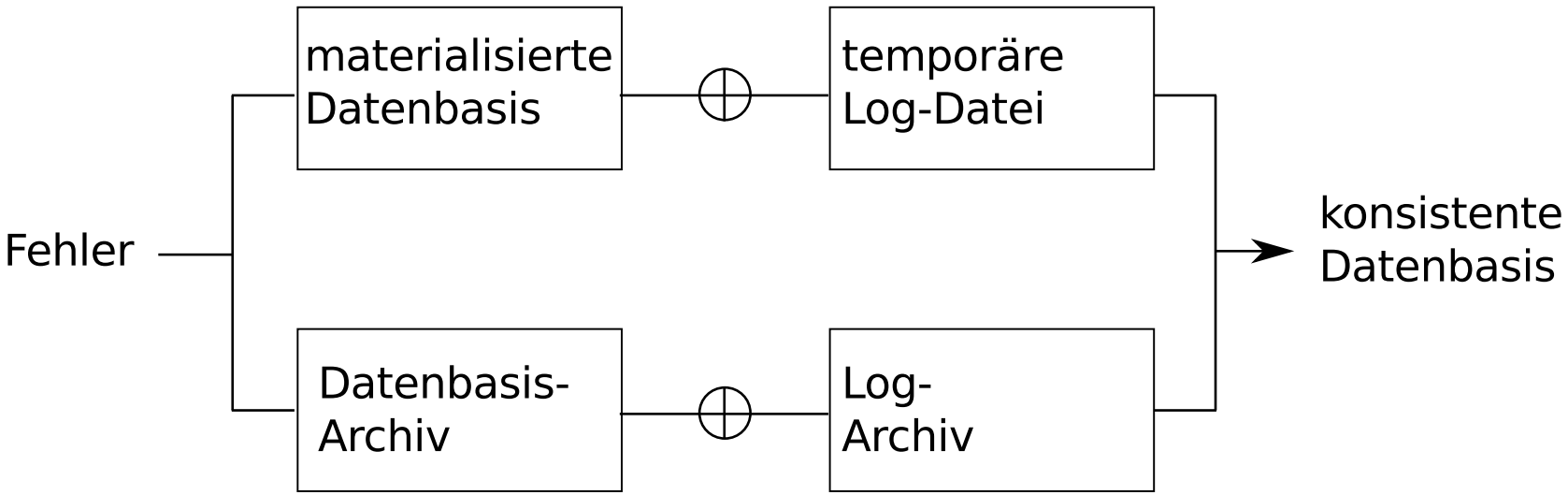
(nur zum Teil hier angesprochen)



In blau hervorgehoben: Hauptsächlich betrachtete Konfiguartion.

Recovery

Recovery nach einem Verlust der materialisierten Datenbasis.



Dies kann auch auf einzelne Seiten angewandt werden z.B. bei einzelnen fehlerhaften Seiten (Media Recovery)

Zusammenfassung Recovery

- Motivation: ACID (Speziell Atomicity und Durability)
- Aber auch: High Availability (durch kleine MTTR), also Effizienz
- Es gibt verschiedene Arten von Recovery (je nach Fehler)
- Haben (hauptsächlich) über Crash-Recovery gesprochen
- Essenz: Speichern von Log-Informationen
- WAL-Prinzip und Commit-Regel
- Redo von Gewinner-TA, Undo von Verlierer-TA
- Idempotenz des Wiederanlaufs
- Sicherungspunkte zum schnelleren Wiederanlauf (nicht das ganze Log betrachten müssen)
- Kurz: Zurücksetzen von Transaktionen

Mehrbenutzersynchronisation – Aspekte

Konzept der Serialisierbarkeit

- Final-State-Serialisierbarkeit (FSR)
- View-Serialisierbarkeit (VSR)
- Konflikt-Serialisierbarkeit (CSR)

Synchronisation

- Basierend auf Sperren
- Basierend auf Zeitstempeln
- Behandlung von Deadlocks

Wiederholung: Das lost-update Problem

t_1	Time	t_2
	/* x = 100 */	
r(x)	1	
	2	$r(x)$
/*update $x := x + 30$ */	3	
	4	/* update $x := x + 20$ */
$w(x)$	5	
	/* $x = 130$ */	
	6	$w(x)$
	/* $x = 120$*/	

Wiederholung: Das lost-update Problem / Notation

- Die Essenz dieses Problems kann durche folgende Sequenz von Lese- und Schreiboperationen ausgedrückt werden:

$$r_1(x)r_2(x)w_1(x)w_2(x)$$

Wiederholung: Das inconsistent-read Problem

Beispiel aus z.B. Anwendung in Bank. Aktueller Stand $x = y = 50$, also $x + y = 100$. Transaktion t_1 berechnet die Summe von x und y, während t_2 einen Wert von 10 von x nach y transferiert.

t_1	Time	t_2
	1	$r(x)$
	2	/* $x := x - 10$ */
	3	$w(x)$
/* $sum := 0$ */	4	
$r(x)$	5	
$r(y)$	6	
/* $sum := sum + x$ */	7	
/* $sum := sum + y$ */	8	
	9	$r(y)$
	10	/* $y := y + 10$ */
	11	$w(y)$

Wiederholung: Das inconsistent-read Problem (2)

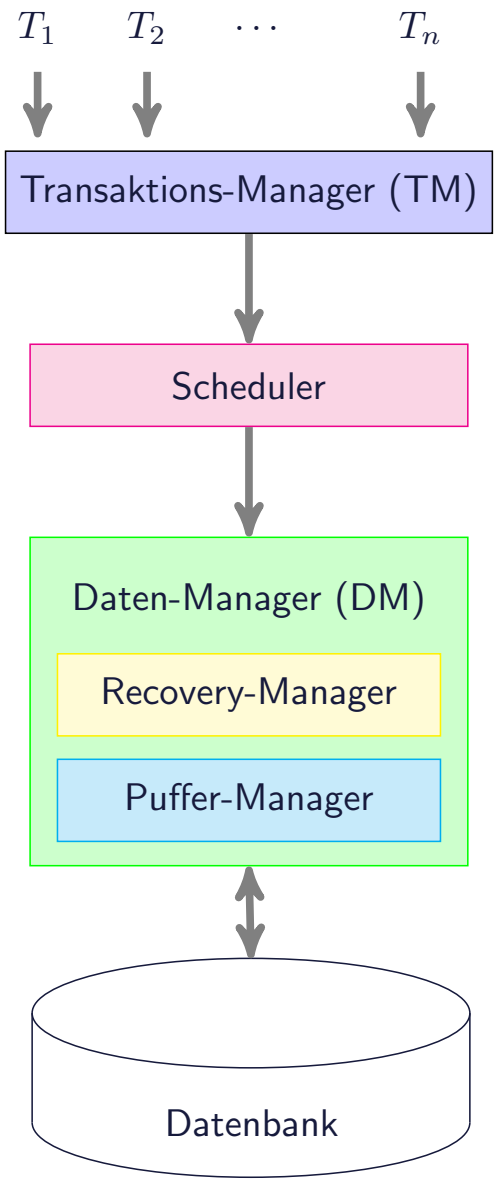
- Offensichtlich ist auch hier wieder das Problem, dass Lese- und Schreiboperationen der einzelnen Transaktionen gemischt ablaufen

$$r_2(x)w_2(x)r_1(x)r_1(y)r_2(y)w_2(y)$$

Wiederholung: Das dirty-read Problem

t_1	Time	t_2
$r(x)$	1	
/* $x := x + 100$ */	2	
$w(x)$	3	
	4	$r(x)$
	5	/* $x := x - 100$ */
failure & rollback	6	
	7	$w(x)$

Der Datenbank-Scheduler



- TM: Informationen über TA, welche Schritte als nächstes ausgeführt werden können.
- Scheduler: Bekommt Schedules vom TM und muss diese in einen serialisierbaren Schedule umwandeln.
- Welcher verarbeitet wird von Daten-Manager (DM)

Aktionen des Schedulers

Scheduler empfängt Schritte der Form r, w, a und c als Eingabe. Diese müssen in einen serialisierbaren Schedule uberführt werden. Dazu kann Scheduler folgende Aktionen durchführen:

1. **Output:** Schritt (r, w, a oder c) wird an Output-Schedule (am Anfang leer) angehängt.
2. **Reject:** Schritt wird nicht ausgegeben (weil z.B. dies die Serialisierbarkeit des Outputs zerstören würde). Also muss die dazugehörige TA abgesprochen werden.
3. **Block:** Schritt wird nicht ausgegeben und auch nicht rejected, sondern als "momentan nicht ausführbar" angesehen und deshalb auf einen späteren Zeitpunkt verschoben.

Es gibt optimistische und pessimistische Scheduler. Sperrbasierte Scheduler (z.B. nach 2PL) gehören zur Klasse der pessimistischen Scheduler.

Aktionen des Daten-Managers

Datenmanager (DM) führt die Schritte eines (serialisierbaren) Schedules aus, in der Reihenfolge wie vom Scheduler gegeben. D.h.

- für ein r: ein Datenobjekt wird gelesen
- für ein w: ein Datenobjekt wird geschrieben
- für ein c: Einleitung der Schritte, um das Ergebnis der TA festzuschreiben
- für ein a: Einlelitung der Schritte, um die TA rückgängig zu machen

Read-/Write- Modell

- Datenbank ist Menge von unteilbaren, uninterpretierten Datenobjekten (z.B. Seiten, Datensätze), $D = \{x, y, z, \ldots\}$
- DB-Anweisungen der Transaktion i beschrieben durch atomare Lese- und Schreiboperationen auf Objekten:
  - $r_i(x)$, $w_i(x)$ zum Lesen bzw. Schreiben des Datenobjekts x
  - c_i, a_i zur Durchführung eines **commits** bzw. **aborts**
- Jeder Wert, der von einer TA t geschrieben wird, ist **potenziell** abhängig von allen Datenobjekten, die t vorher gelesen hat.

Transaktion

- Eine Transaktion t ist eine Partialordnung von Schritten der Form

$$p_i \in \{r(x), w(x)\}$$

  mit $x \in D$.
- Lese- und Schreiboperationen sowie mehrfache Schreiboperationen auf demselben Datenobjekt sind geordnet.
- Eine vollständige TA hat als letzte Operation entweder einen Abbruch a oder ein Commit c

$$t = p_1 \dots p_n a$$

  oder

$$t = p_1 \dots p_n c$$

Historien und Schedules

- Es sei $T = \{t_1, \ldots, t_n\}$ eine Menge von Transaktionen, wobei jedes $t_i \in T$ die Form $t_i = \{op_i, <_i\}$ besitzt, op_i die Menge der Operationen von t_i und $<_i$ ihre Ordnung $(1 \leq i \leq n)$ bezeichnen.
- Eine **Historie** für T ist ein Paar $s = (op(s), <_s)$, so dass:
  (a) $op(s) \subseteq \bigcup_{i=1}^{n} op_i \cup \bigcup_{i=1}^{n} \{a_i, c_i\}$ und $\bigcup_{i=1}^{n} op_i \subseteq op(s)$
  (b) $(\forall i, 1 \leq i \leq n) c_i \in op(s) \Leftrightarrow a_i \notin op(s)$
  (c) $\bigcup_{i=1}^{n} <_i \subseteq <_s$
  (d) $(\forall i, 1 \leq i \leq n)(\forall p \in op_i)\ p <_s a_i$ oder $p <_s c_i$
  (e) Jedes Paar von Operatoren $p, q \in op(s)$ von verschiedenen Transaktionen, die auf dasselbe Datenobjekt zugreifen und von denen wenigstens eine davon eine Schreiboperation ist, sind so geordnet, dass $p <_s q$ oder $q <_s p$ gilt.
- Ein **Schedule** ist ein Präfix einer Historie

Historien und Schedules (2)

Erläuterungen zu den zuvor genannten Punkten (a) bis (e):

- (a) Historie enthält alle Operationen aller Transaktionen
- (b) Historie benötigt eine Terminierungsoperation für jede TA
- (c) Historie bewahrt alle Ordnungen innerhalb der TA
- (d) Terminierungsoperation ist letzte Operation in jeder TA
- (e) Konfliktoperationen sind geordnet.