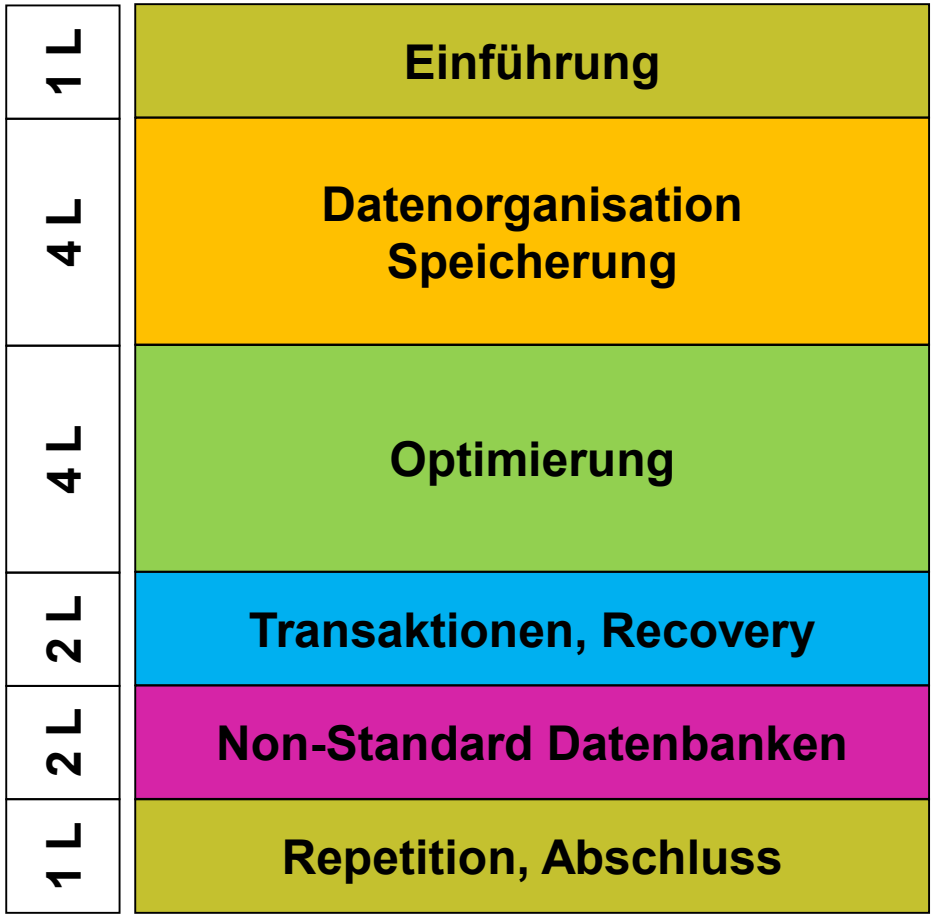
Zürcher Hochschule
für Angewandte Wissenschaften
zh aw School of Engineering
InIT Institut für angewandte Informationstechnologie

Transaktionen

Lehrbuch Kapitel 10.1-10.3, 10.8, 11.1-11.2

Gliederung

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie



"You are here"

Rückblick

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Wichtigste Anforderungen an Transaktionsverarbeitung
- Probleme konkurrenter Transaktionen:
  - Lost Update
  - Dirty Read
  - Non-Repeatable Read
  - Phantom Read
- ACID-Eigenschaften eines Transaktionssystems
- Aspekte von Nebenläufigkeit und Transaktionen in der Praxis
- Scheduler, Schedule, Serialisierbarkeit

Lernziele heute

- Konzept der Sperren verstehen
- Wissen, was Blocking, Livelock und Deadlock bedeuten
- Grundlagen von Recovery kennen:
  1. Recovery-Komponenten eines DBMSs
  2. Fehlerklassen
  3. Protokollieren von Transaktionen: Logging und Sicherungspunkte

Transaktionsverwaltung: Sperrverfahren

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Sperren (Locks) auf Datenbankobjekten
  1. Abfrage der Sperre vor jedem Zugriff auf ein Datenbankobjekt und
  2. setzen einer Sperre wenn verfügbar

- Erweiterung jeder Transaktion durch spezifische Operationen:
  - Lese-Sperre (Share Lock): read lock (rl) – read unlock (ru)
    andere Transaktionen weiterhin lesend auf das Datenbankobjekt zugreifen und Lesesperren auf dieses Datenbankobjekt setzen. Sie können jedoch keine Schreibsperre setzen)
  - Schreib-Sperre (Exclusive Lock): write lock (wl) – write unlock (wu)
    das betreffende Datenbankobjekt nur dieser Transaktion zur Verfügung und kann nur durch diese Transaktion geändert werden
  - Unlock: read unlock (ru) und write unlock (wu) werden üblicherweise zu unlock (u) zusammengefasst.

Transaktionsverwaltung: Sperrverfahren

Regeln zur Sperrdisziplin:

- Schreibzugriff w(x) nur nach Setzen einer Schreibsperre wl(x) möglich.
- Lesezugriffe r(x) nur nach Setzen einer Lesesperre rl(x) oder wl(x) erlaubt.
- Eine Schreibsperre w(x) kann nur gesetzt werden, wenn auf x keine Sperren existiert.
- Eine Lesesperre r(x) kann nur gesetzt werden, wenn auf x keine Schreibsperre existiert.
- Nach u(x) darf die Transaktion kein erneutes rl(x) oder wl(x) ausführen.
- Eine Transaktion darf eine Sperre der selben Art auf demselben Objekt nicht nochmals anfordern.
- Beim Commit/Rollback müssen alle Sperren aufgehoben werden.

Transaktionsverwaltung: Sperrverfahren

Sperren führen zu folgenden (vier) Problemen:

1. Blocking: Eine gesperrte Ressource zwingt andere Prozesse zu warten, bis diese wieder freigegeben wird → Reduktion des Durchsatzes an Transaktionen.

Lösung:

- Lösung: Keine, ohne Sperren funktioniert die Transaktionsverwaltung nicht (→ siehe Praxishinweise weiter hinten).

Transaktionsverwaltung: Sperrverfahren

2. Verhungern, Livelock: Eine Transaktion kommt nie dran, weil immer wieder andere vorher berücksichtigt werden.
    1. T_1 sperrt x
    2. T_2 will x sperren, muss aber warten
    3. T_3 will danach x sperren, muss auch warten
    4. T_1 gibt x frei
    5. T_3 kommt vor T_2 an eine Zeitscheibe, sperrt x
    6. T_2 will weiterhin x sperren, muss aber warten
    7. T_4 will danach x sperren, muss auch warten
    8. T_3 gibt x frei
    9. T_4 kommt vor T_2 an die nächste Zeitscheibe ...

Lösung: RDBMS muss geeignet («fair») auswählen.

Transaktionsverwaltung: Sperrverfahren

3. Deadlock (Verklemmung): Eine Menge von Transaktionen sperren sich gegenseitig, wenn jede Transaktion der Menge auf ein Ereignis wartet, das nur durch eine andere Transaktion der Menge ausgelöst werden kann.
   Da alle am Deadlock beteiligten Transaktionen warten, kann keine ein Ereignis auslösen, so dass eine andere geweckt wird. Also warten alle beteiligten Transaktionen ewig. Beispiel:

t_1	t_2
$wl(x)$	
	$wl(y)$
$wl(y)$	
	$wl(x)$
Verklemmung!	

Lösung: RDBMS erkennt Deadlocks durch Zyklensuche im sogenannten „wer wartet auf wen“-Graphen:

- „Opfer“-Transaktion auswählen und zurücksetzen
- Zu einem späteren Zeitpunkt erneut starten

Transaktionsverwaltung: Sperrverfahren

4. Phantom-Read: Sperren als Lösung? Beispiel:
   - Ein Bonus von 100.000 Euro soll gleichmässig auf alle Mitarbeiter der Firma verteilt werden
   - Parallel dazu wird ein neuer Mitarbeiter eingefügt

Zeit	T_1	T_2
1	`select count(*) into X` `from Mitarbeiter;`	
2		`insert into Mitarbeiter` `values (Meier, 50.000, …);`
3		**`commit;`**
4	`update Mitarbeiter` `set Gehalt = Gehalt + 100.000/X;`	
5	**`commit;`**	

Transaktionsverwaltung: Sperrverfahren

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Lösung 1: Zusätzlich zu den Tupeln muss auch der Zugriffsweg, auf dem man zu den Objekten gelangt ist, gesperrt werden.
  - Beispiel: `SELECT COUNT(*) INTO X FROM Mitarbeiter`
    - Alle Mitarbeiter (bzw. deren Primärschlüssel-Index) müssen mit einer RL-Sperre belegt werden.
    - Beim Einfügen eines neuen Mitarbeiter wird dies erkannt und T2 muss warten.
  - Sperre kann ggf. auch selektiver sein – z.B.: `SELECT COUNT(*) INTO X FROM Mitarbeiter WHERE PNr BETWEEN 1000 AND 2000`
    - Nur die Mitarbeiter mit der entsprechenden PNr müssen gesperrt werden (z.B. Index-Bereich von PNr[1000,2000])
- Lösung 2: Von der Tabelle vorgängig einen Snapshot erstellen (siehe auch Mehrversionensynchronisation).
- Lösung 3 (unabhängig vom Isolationslevel): Die Tabelle ganz sperren (SQL-Server-Syntax): `SELECT * FROM Mitarbeiter WITH (TABLOCKX)`

Transaktionsverwaltung: Hinweise für die Praxis

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Blockierung:
  - Lange laufende Abfragen vermeiden, kurze Transaktionen
  - Ineffiziente Abfragen optimieren
  - Vernünftig indizieren und Indexe verwenden
  - Keine Benutzereingaben innerhalb von Transaktionen
  - Sperr-Timeouts verwenden
  - …

- Deadlocks:
  - Objekte immer in derselben Reihenfolge ansprechen
  - 'Teure' Objekte zuletzt ansprechen
  - Angepassten Isolationslevel verwenden

Recovery

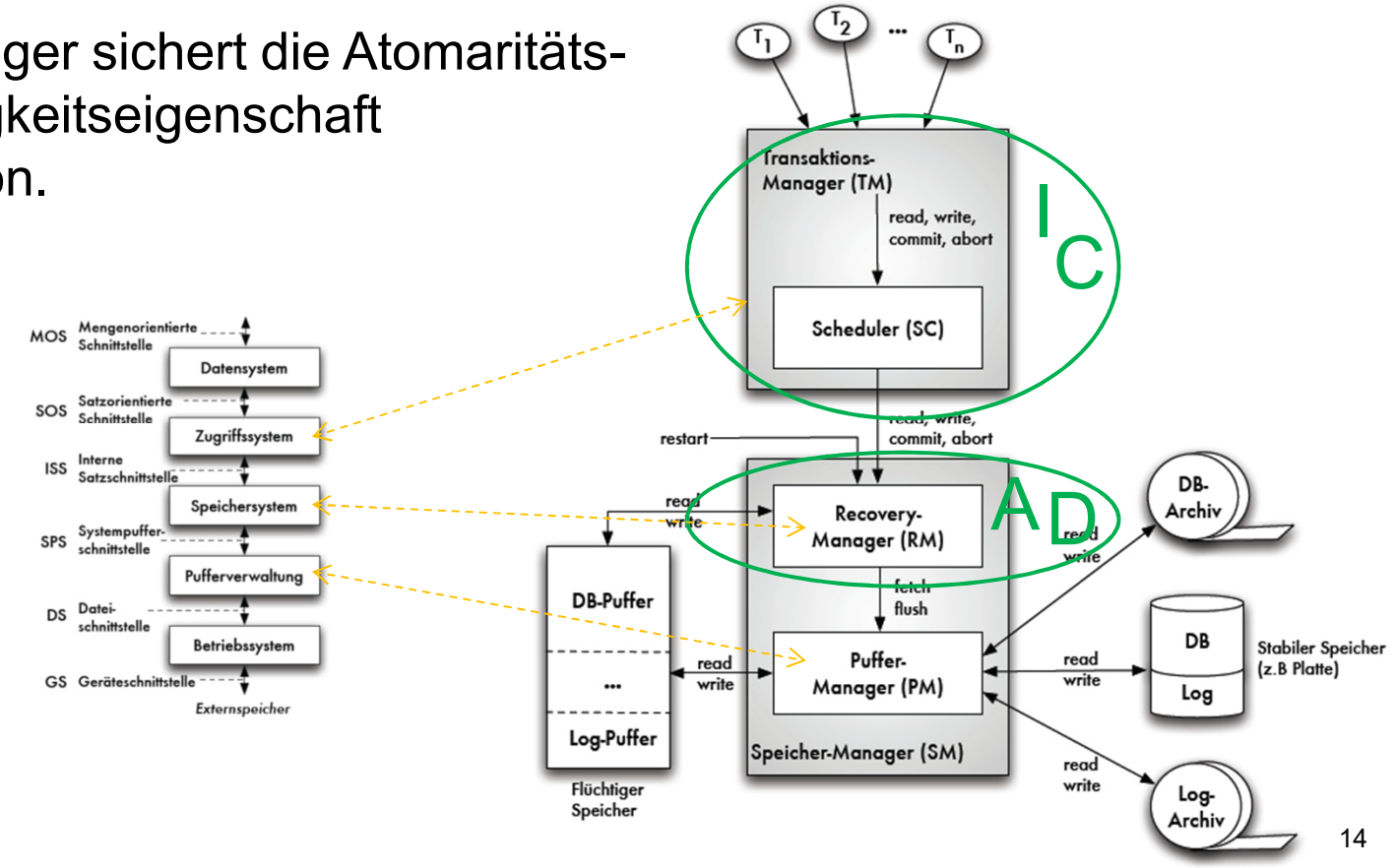
- Recovery (Wiederherstellung):
  Alle Massnahmen zur Wiederherstellung verloren gegangener Datenbestände.

- Behandlungsstrategien:
  Je nach Fehlerart müssen unterschiedliche Behandlungsstrategien ausgeführt werden

Recovery, Transaktion und ACID

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Transaktions-Manager (TM) / Scheduler (SC) wahrt die Isolation- und Konsistenzeigenschaft einer Transaktion.
- Recovery-Manager sichert die Atomaritäts- und Dauerhaftigkeitseigenschaft einer Transaktion.

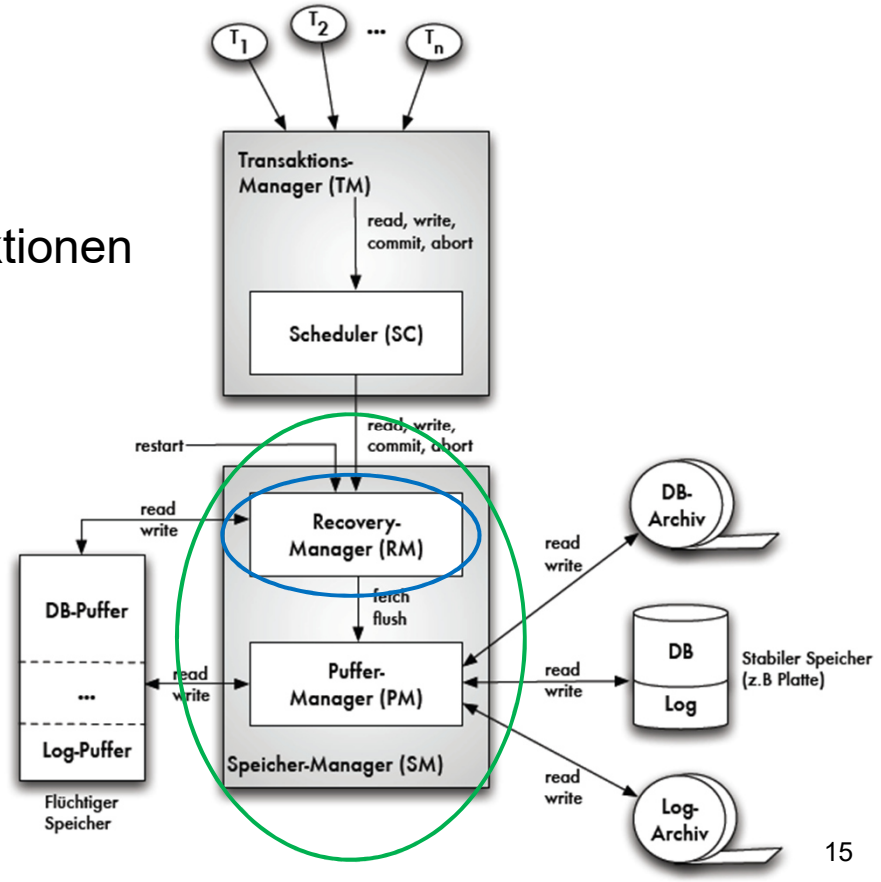


Recovery-Komponenten

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

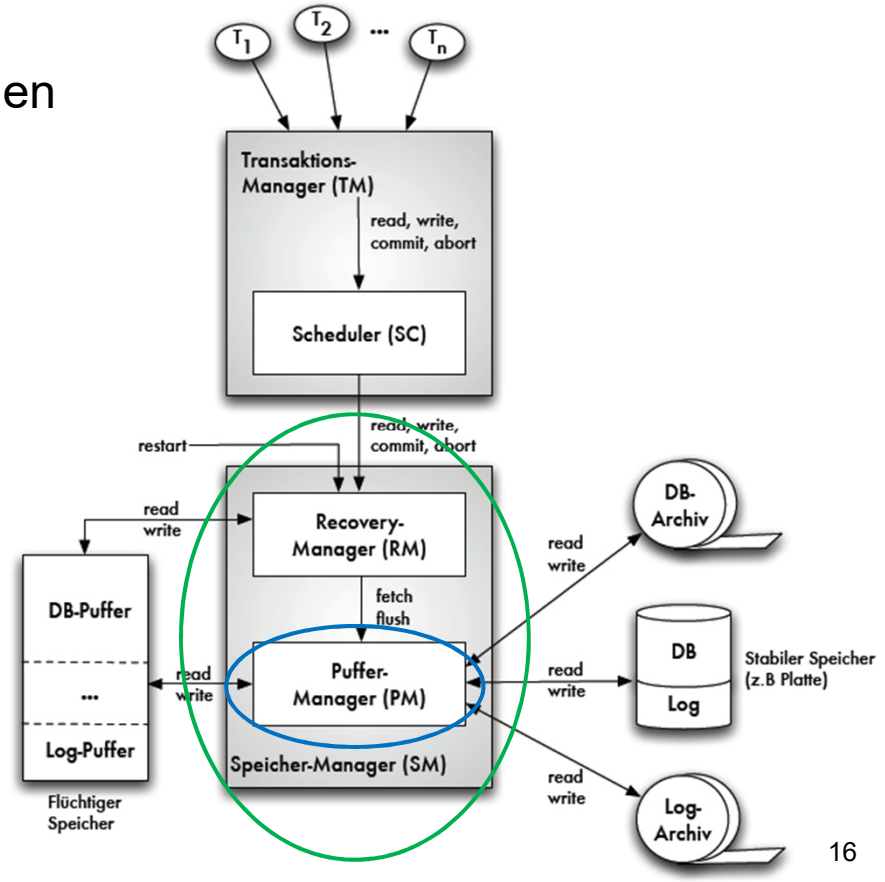
Der Speicher-Manager (SM), besonders wichtig für das Recovery, bildet die Schnittstelle zwischen flüchtigem und stabilem Speicher, er umfasst den Recovery-Manager und den Puffer-Manager.

1. Recovery-Manager (RM): sorgt dafür, dass nach einem Fehler:
   - alle Änderungen „committeter“ Transaktionen im stabilen Speicher abgelegt werden
   - keine Änderungen von aktiven oder abgebrochenen Transaktionen im stabilen Speicher verbleiben



Recovery-Komponenten

2. Puffer-Manager (PM) verwaltet den Puffer (DB- und Log-Puffer):
   - holt Daten (Seiten) vom stabilen Speicher in den Puffer
   - schreibt Daten (Seiten) vom Puffer in den stabilen Speicher
   - ersetzt Daten (Seiten) im Falle eines „Pufferüberlaufs“

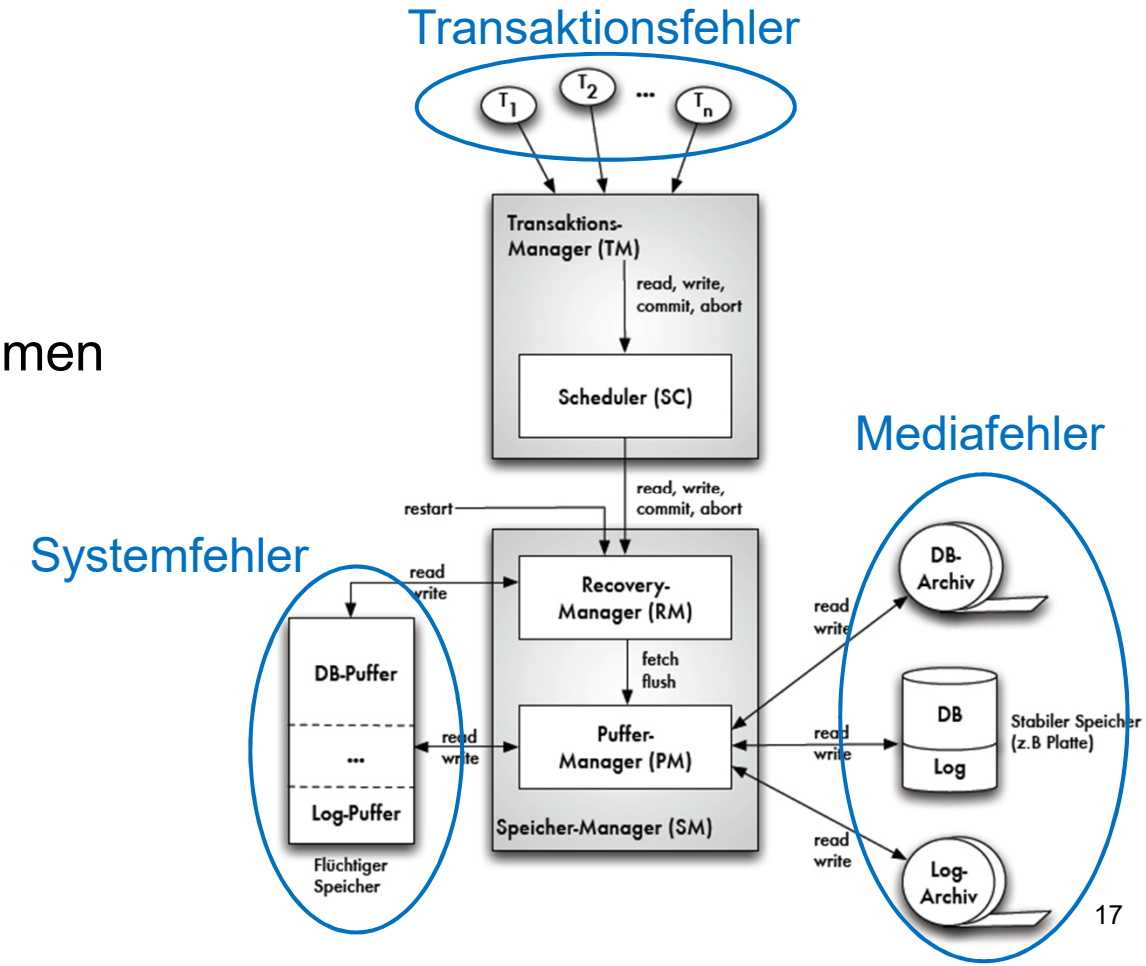


Fehlerklassifikation

Klassifikation der Fehler nach dem betroffenen Bereich des DBMS:

1. Transaktionsfehler
2. Systemfehler
3. Mediafehler

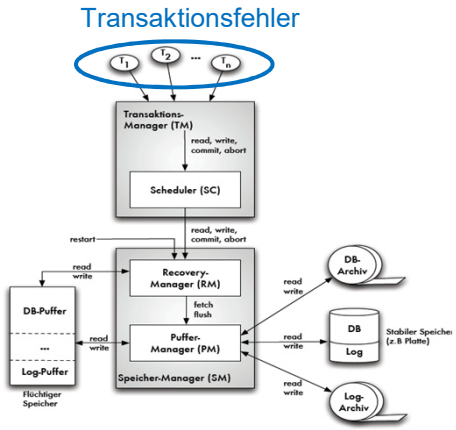
→ versch. Recovery-Massnahmen



Fehlerklassifikation: Transaktionsfehler

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

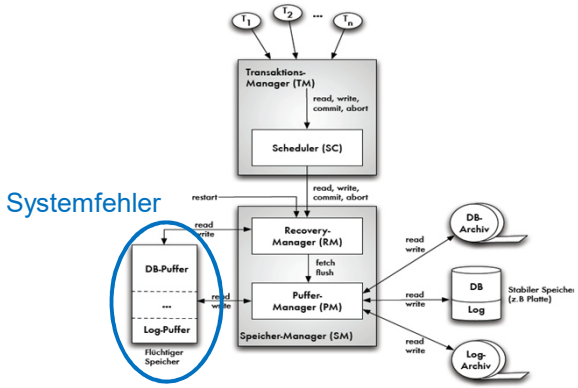
- Transaktionsfehler:
  - haben den Abbruch der jeweiligen Transaktion zur Folge
  - haben keinen Einfluss auf den Speicher des Systems
    → lokaler Fehler
- Typische Transaktionsfehler:
  - Fehler im Anwendungsprogramm (z.B. Division durch 0)
  - Transaktionsabbruch durch den Benutzer
    (z.B. unzulässige Dateneingabe, Timeout bei Inaktivität)
  - Transaktionsabbruch durch das System (z.B. Deadlock, Verletzung von Integritätsbedingungen, Zugriffsrechte verletzt)
- Behandlung:
  - „Isoliertes“ Zurücksetzen aller Änderungen der Transaktion: UNDO (Rollback) aller Datenänderungen



Fehlerklassifikation: Systemfehler

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Systemfehler:
  - Folge: Zerstörung der Daten im Hauptspeicher (flüchtige DB)
  - betreffen jedoch nicht den Hintergrundspeicher (permanente DB)



- Typische Systemfehler:
  - DBMS-Fehler (z.B. Konfigurationsfile fehlerhaft)
  - Betriebssystemfehler (Seitenfehler, ungültiger Befehl)
  - Hardware-Fehler (z.B. Stromausfall, Fehler im Memory)
- Behandlung:
  - Nachvollziehen der Änderungen, die von abgeschlossenen Transaktionen nicht in die DB eingebracht wurden (REDO)
  - Zurücksetzen der Änderungen, die von nicht beendeten Transaktionen in die DB eingebracht wurden (UNDO).

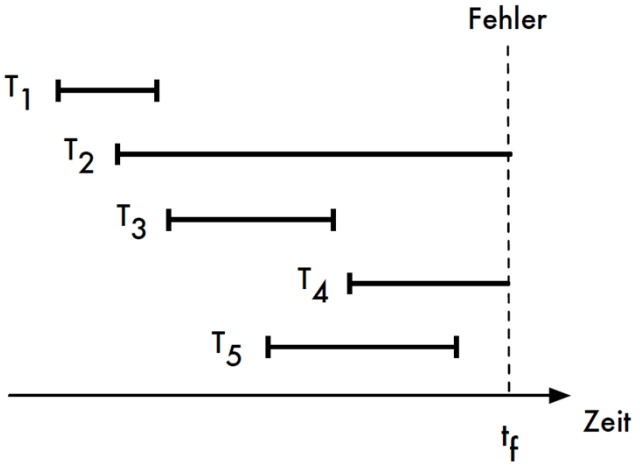
Fehlerklassifikation: Systemfehler

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Beispiel Systemfehler: flüchtiger Speicher zum Zeitpunkt t_f ist verloren oder unbrauchbar.

- Transaktionszustände:
  - zum Fehlerzeitpunkt noch aktive Transaktionen (T_2 und T_4)
  - bereits vor dem Fehlerzeitpunkt beendete Transaktionen (T_1, T_3 und T_5)

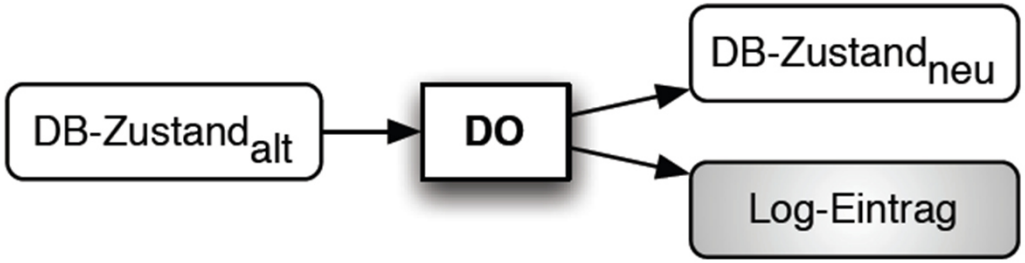
- Probleme:
  - Dauerhaftigkeitseigenschaft → Effekte von T_1, T_3 und T_5 müssen dauerhaft in der DB sein
  - Atomaritätseigenschaft → Effekte von T_2 und T_4 dürfen nicht in der DB sein



Fehlerklassifikation: Systemfehler

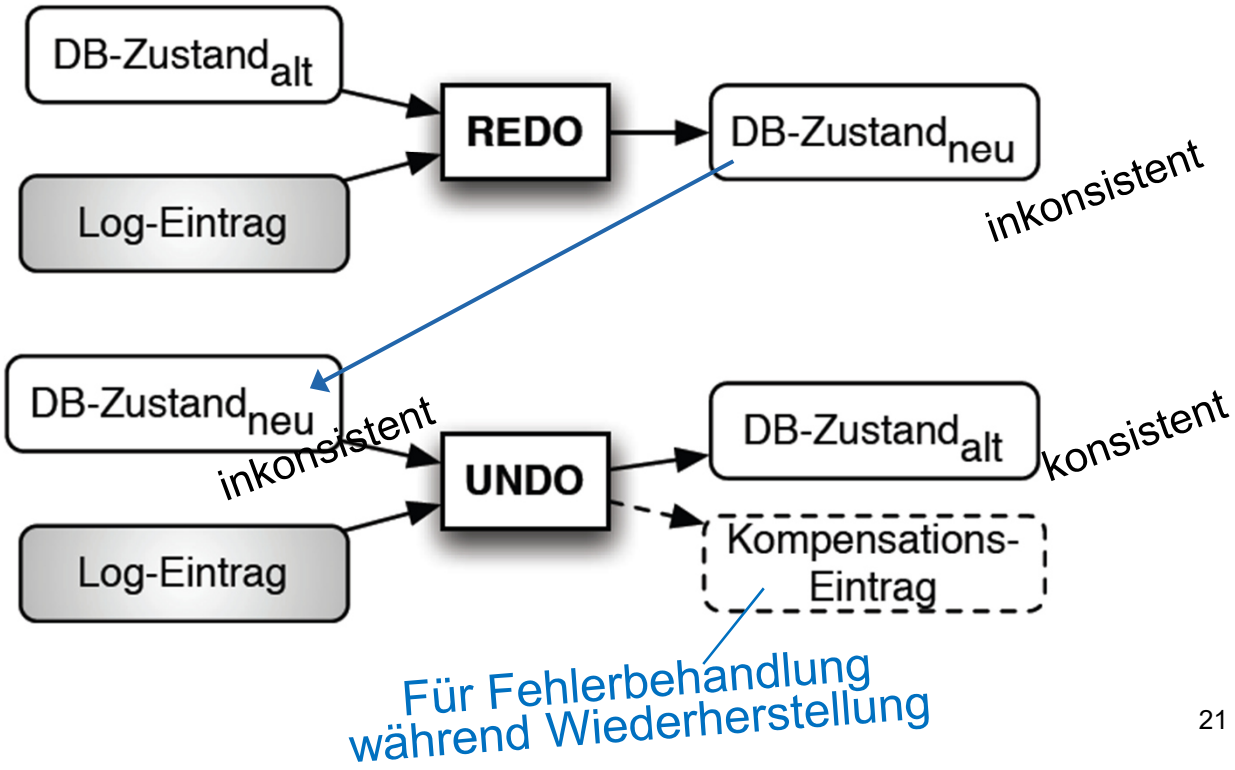
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Normaler Betrieb:



Wiederherstellung nach Ausfall in 2 Phasen:

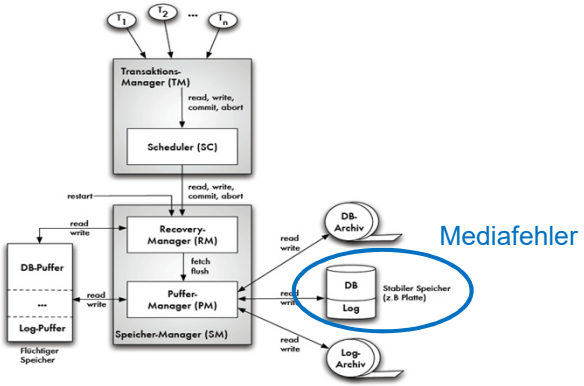
1. REDO
2. UNDO



Fehlerklassifikation: Mediafehler

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Mediafehler
  - Verlust von Daten der stabilen Datenbank
- Typische Mediafehler:
  - „Head-Crashes“, Controller-Fehler, …
  - Naturgewalten wie Feuer oder Erdbeben
  - Operator-Fehler / Fehler im Programm



- Behandlung:

  Voraussetzung: Es existieren Sicherungskopien (DB-Archiv, Log-Archiv) auf separatem „Medium“ (ev. auch Log-Datei auf eigenem Medium)
  - Einspielen der verlorenen DB-Dateien vom DB-Archiv ($\rightarrow$ DB-Restore)
  - Anwenden aller Transaktionslogs aus dem Log-Archiv, die seit dem Anlegen des DB-Archivs erzeugt wurden
  - Die Datenbank muss durch REDO und UNDO in einen konsistenten Zustand gebracht werden (analog Systemfehler)

Logging

- Zur Wiederherstellung muss eine Historie aller Änderungen protokolliert (Logging) werden: Logbuch (‘’Transaction Log’’)
- Physisches Log:
  - die alten und die neuen Werte der geänderten Daten werden gespeichert
  - die alten Werte nennt man Before Images und die neuen After Images.

Hansel and Gretel left behind a trail of crumbs which would allow them to retrace their steps (by following the trail backwards) and would allow their parents to find them by following the trail forwards. This was the first undo and redo log. Unfortunately, a bird ate the crumbs and caused the first log failure.

Jim Gray

Logging: Physisches Logging

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Logbuch:

zu Beginn A=10, B=0

Schritt	T_1	T_2	Log	
1	`lock` A		$(T_1, begin)$	Vor dem ersten Schritt wird (T_1; begin) eingetragen
2	`read` A			
3	$A := A - 1$			
4	`write` A		$(T_1, A, 10, 9)$	Vor Schritt 4 folgt (T_1;A; 10; 9)
5	`lock` B			
6	`unlock` A			
7		`lock` A	$(T_2, begin)$	Vor Schritt 7 beginnt T_2 mit (T_2; begin)
8		`read` A		
9		$A := A \times 2$		
10	`read` B			
11		`write` A	$(T_2, A, 9, 18)$	Vor Schritt 11 wird (T_2;A; 9; 18) eingetragen
12		`commit`	$(T_2, commit)$	Vor Schritt 12 folgt Abschluß von T_2 : (T_2; commit)
13		`unlock` A		
14	$B := A/B\downarrow$		$(T_1, abort)$	Nach Schritt 14 wird Abbruch (T_1; abort) geschrieben

Division durch 0

Logging: Physisches Logging

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Beispiel für physische Log Einträge:

Schritt	T_1	T_2	Log
1	`lock` A		$(T_1, begin)$
2	`read` A		
3	$A := A - 1$		
4	`write` A		$(T_1, A, 10, 9)$
5	`lock` B		
6	`unlock` A		
7		`lock` A	$(T_2, begin)$
8		`read` A	
9		$A := A \times 2$	
10	`read` B		
11		`write` A	$(T_2, A, 9, 18)$
12		`commit`	$(T_2, commit)$
13		`unlock` A	
14	$B := B/A \downarrow$		$(T_1, abort)$

```
[ #1, T1, BOT ]
[ #2, T1, PA, 10, 9, #1 ]
[ #3, T2, BOT ]
[ #4, T2, PA, 9, 18, #3 ]
[ #5, T2, commit, #4 ]
[ #6, T1, abort, #2 ]
 ...
```

Beispiel SQL-Server-Log (9 von über 100 Spalten angezeigt)

	current lsn	operation	transaction id	previous lsn	allocunitname	page id	offset in row	description	log record
1	000009aa:00000110:0001	LOP_BEGIN_XACT	0000:000067f6	00000000:00000000:0000	NULL	NULL	NULL	user_transaction;0x83845e48bc256648b9989b9dff831c56	0x00004C0000000000000000000000200F6670
2	000009aa:00000110:0003	LOP_MODIFY_COLUMNS	0000:000067f6	000009aa:00000110:0001	SystemData.NumberRange	0001:00001168	0		0x00003E00AA0900001001000001000200F6670
3	000009aa:00000110:0004	LOP_COMMIT_XACT	0000:000067f6	000009aa:00000110:0001	NULL	NULL	NULL		0x00005000AA0900001001000001000200F6670

Logging: Physisches Logging

Ein Eintrag im Log hat folgenden grundsätzlichen Aufbau:

[LSN, TA, PageID, Undo, Redo, PrevLSN] *

- LSN — Log-Sequence-Number, eindeutige und aufsteigende Nummerierung der Log-Einträge
- TA — Transaktionskennung
- PageID — Seitennummer
- Redo — REDO-Information; After-Image
- Undo — UNDO-Information; Before-Image
- PrevLSN — Verweis auf den vorherigen Eintrag der selben Transaktion

Das WAL-Prinzip

Wie muss das Schreiben des Logs (Log!) erfolgen, so dass die gepufferten **Log**-Seiten (nicht Daten) rechtzeitig auf dem stabilen Speicher sind?

Das Write Ahead Log-Prinzip (WAL-Prinzip) fordert die Einhaltung folgender zwei Regeln:

1. Vor dem COMMIT einer Transaktion müssen alle zugehörigen (gepufferten) Log-Einträge (''log write on commit'') auf stabilen Speicher ausgelagert werden (Logbuch).
   Diese Regel ist notwendig, um ein REDO durchführen zu können.
2. Vor dem Auslagern einer modifizierten (gepufferten) Seite in die persistente Datenbank (stabiler Speicher) müssen alle zugehörigen (gepufferten) Log-Einträge zur Seite auf stabilen Speicher ausgelagert werden (Logbuch).
   Diese Regel ermöglicht das UNDO bei abgebrochenen Transaktionen.

Sicherungspunkte (SP)

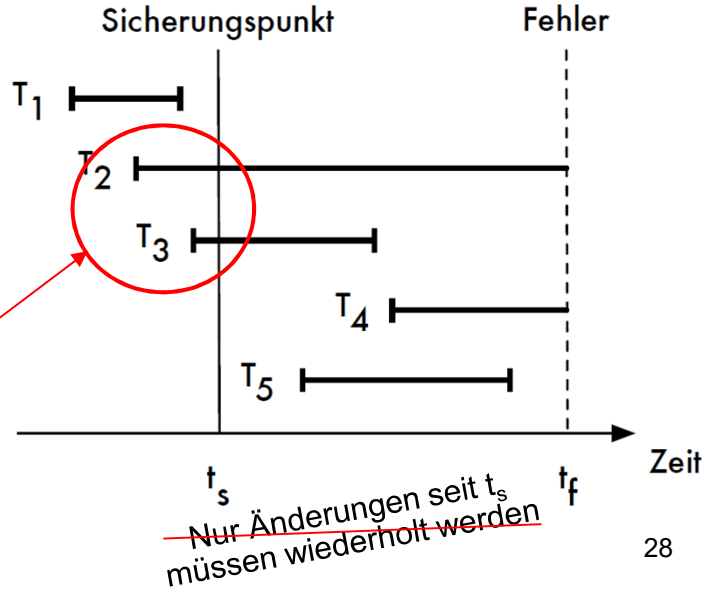
- Problem beim Recovery:

  Wir wissen nicht, wie alt die älteste Seite im Puffer war, wir müssen das gesamte Log einspielen (seit Start der DB)

- Lösung: Sicherungspunkte (checkpoint)

  Beim Aufruf eines Sicherungspunktes werden:

  1. Die geänderte Seiten (dirty pages) der flüchtigen DB (Puffer) in die persistente DB geschrieben.
  2. Der Puffer des Logs ebenfalls auf stabilen Speicher geschrieben (für UNDO).
  3. Der Checkpoint wird im Logbuch registriert.



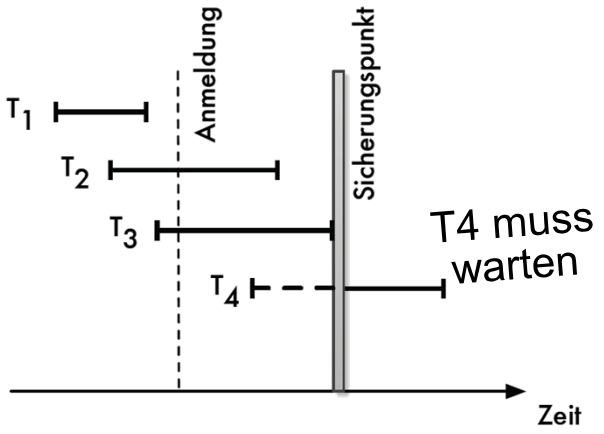
Offenes Problem:
UNDO vor Checkpoint!

Sicherungspunkte (SP)

- Arten von Sicherungspunkten:
  1. Direkte Sicherungspunkte:
     1. Transaktionskonsistente Sicherungspunkte
     2. Aktionskonsistente Sicherungspunkte (einzelner SQL-Befehl)
  2. Unscharfe / indirekte Sicherungspunkte

Transaktionskonsistente SP

- Transaktionskonsistente Sicherungspunkte:
  - Alle Änderungen werden in einem Moment, in dem keine Transaktionen aktiv sind, vom Puffer in die DB geschrieben.
- Ablauf:
  1. Sicherungspunkt anmelden
  2. Neu ankommende Transaktionen müssen warten
  3. Aktive Transaktionen werden zu Ende geführt
  4. Sobald alle aktiven Transaktionen beendet wurden, werden alle geänderten Seiten auf die Platte geschrieben
  5. Der Checkpoint wird im Logbuch registriert
- Vorteil / Nachteil:
  - \+ Späteres Recovery braucht keine Veränderungen vor diesem Punkt mehr zu berücksichtigen
  - Nachteil: Lässt Benutzer unter Umständen lange warten
- Analoges Vorgehen bei aktionskonsistenten SP



Unscharfe SP

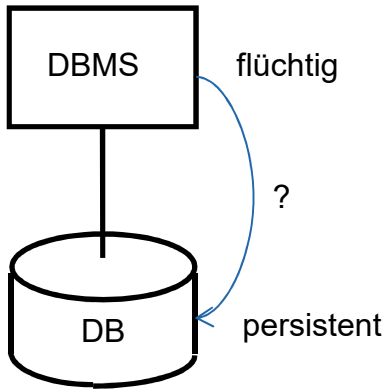
- Direkte Sicherungspunkte (transaktions- oder aktionskonsistent) führen zu Performance-Verlust (man wartet bis alle Transaktionen abgeschlossen sind), daher: unscharfe / indirekte (fuzzy) SP[1)]
  - keine Transaktions- bzw. Aktionskonsistenz
  - „unscharfer“ Zustand der Datenbank
- Ablauf:
  1. Bei unbeeinträchtigter Aktivität der DB asynchrones Schreiben geänderter Seiten
  2. Protokollierung von Statusinformationen: Laufende Transaktionen, geänderte Seiten, etc. (der Checkpoint wird im Logbuch registriert)
- Vorteil / Nachteil
  - \+ geringerer Aufwand, speziell bei grossen Puffern
  - – REDO und UNDO nicht auf letzten scharfen Sicherungspunkt beschränkt

Recoverystrategien

Anforderungen an Recoverystrategien:

- Schnelle Wiederherstellung nach Fehler → hohe Verfügbarkeit
- Geringer Overhead (Ressourcenbedarf) im Normalbetrieb
- Korrektes Verhalten unter allen Bedingungen
- Einfachheit und Testbarkeit

Recoverystrategien: Puffer



Recoverystrategien als Kombination / in Abhängigkeit von:

- Seitenersetzungsstrategie:
  welche Seite darf aus Puffer ausgelagert werden?
- Propagierungsstrategie:
  wann muss eine Seite aus Puffer in DB eingebracht werden?
- Einbringstrategie:
  wie erfolgt das überschreiben der Seite (auf dem Externspeicher: in Place/Twin Block/…)?

Recoverystrategie: UNDO/REDO

Zürcher Hochschule
für Angewandte Wissenschaften
zh aw School of Engineering
InIT Institut für angewandte Informationstechnologie

Es können auch während des Recoverys Fehler auftreten, der Recovery-Prozess muss daher fehlertolerant sein.

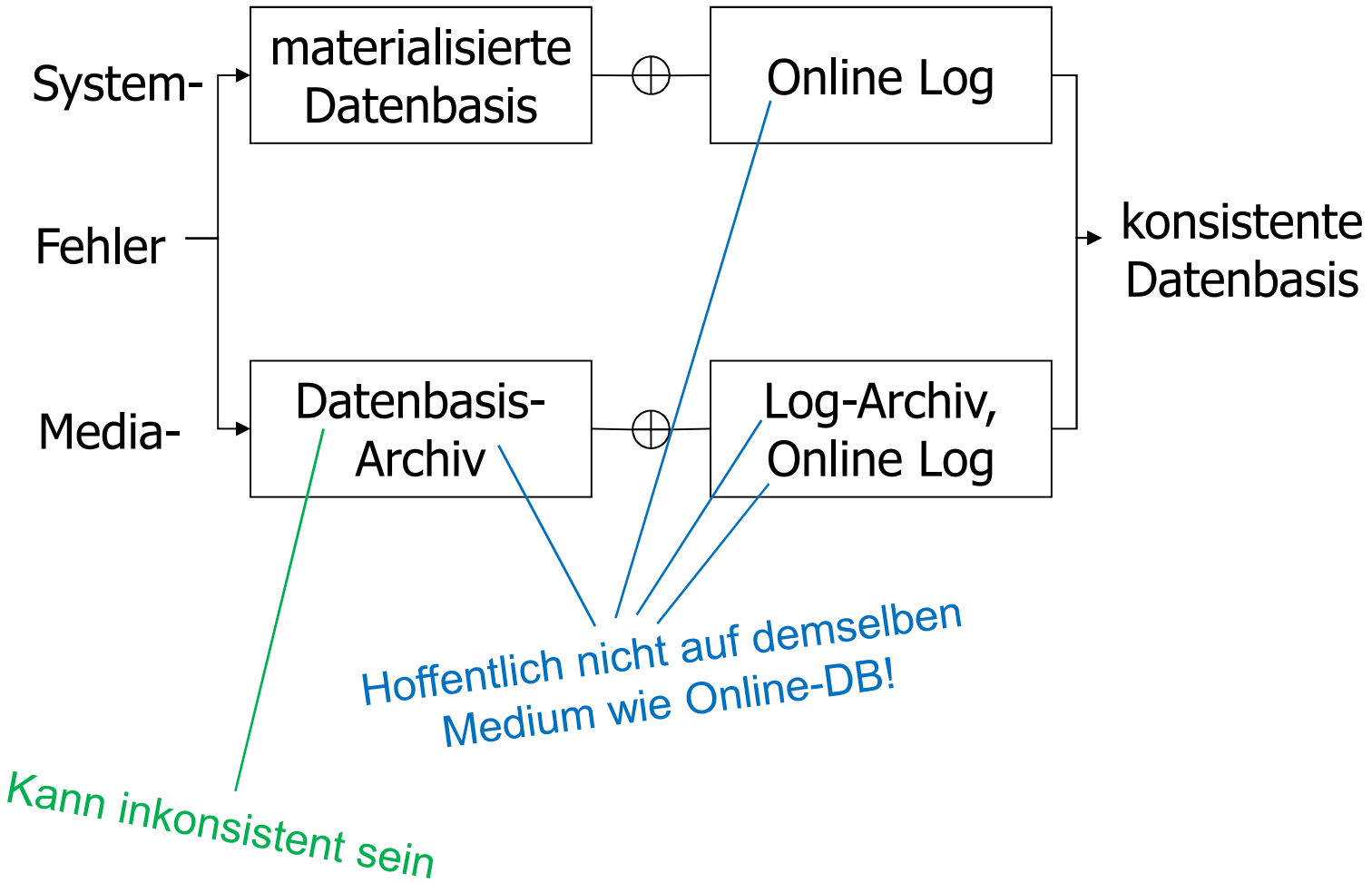
Idempotenz Recovery: Ein mehrfach gestartetes Recovery hat den selben Effekt wie ein einfach gestartetes Recovery.

```
redo(redo(…(redo(x)))) = redo(x)
undo(undo(…(undo(x)))) = undo(x)
```

Hierfür sind spezifische Massnahmen notwendig.

Vergleich Ablauf Recovery für System- und Mediafehler

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie



Vergleich Ablauf Recovery für System- und Mediafehler

- Wiederherstellung nach Systemfehler
  - Ausgangspunkt: materialisierte (permanente) Datenbasis :
    - physisch intakt
    - bezüglich der Transaktionen inkonsistent.
  - Wiederherstellung
    - Anwenden der Transaktionslogs der Online Log-Datei beginnend bei dem letzten abgeschlossenen Sicherungspunkt
- Wiederherstellung nach Mediafehler
  - Ausgangspunkt: materialisierte (permanente) Datenbasis :
    - physisch defekt
    - bezüglich der Transaktionen inkonsistent.
  - Wiederherstellung
    - Einspielen (Restore) der Dateien der Datenbasis vom Archiv (Sicherungskopien)
    - Einspielen der Dateien vom Log-Archiv
    - Anwenden der Transaktionslogs aus dem Log-Archiv und der Online Log-Datei beginnend beim ersten Sicherungspunkt zum Zeitpunkt der Sicherungskopie

Ausblick

- Das nächste Mal: Data warehouse-Systeme
- Lesen: Nichts!