Datenbanken

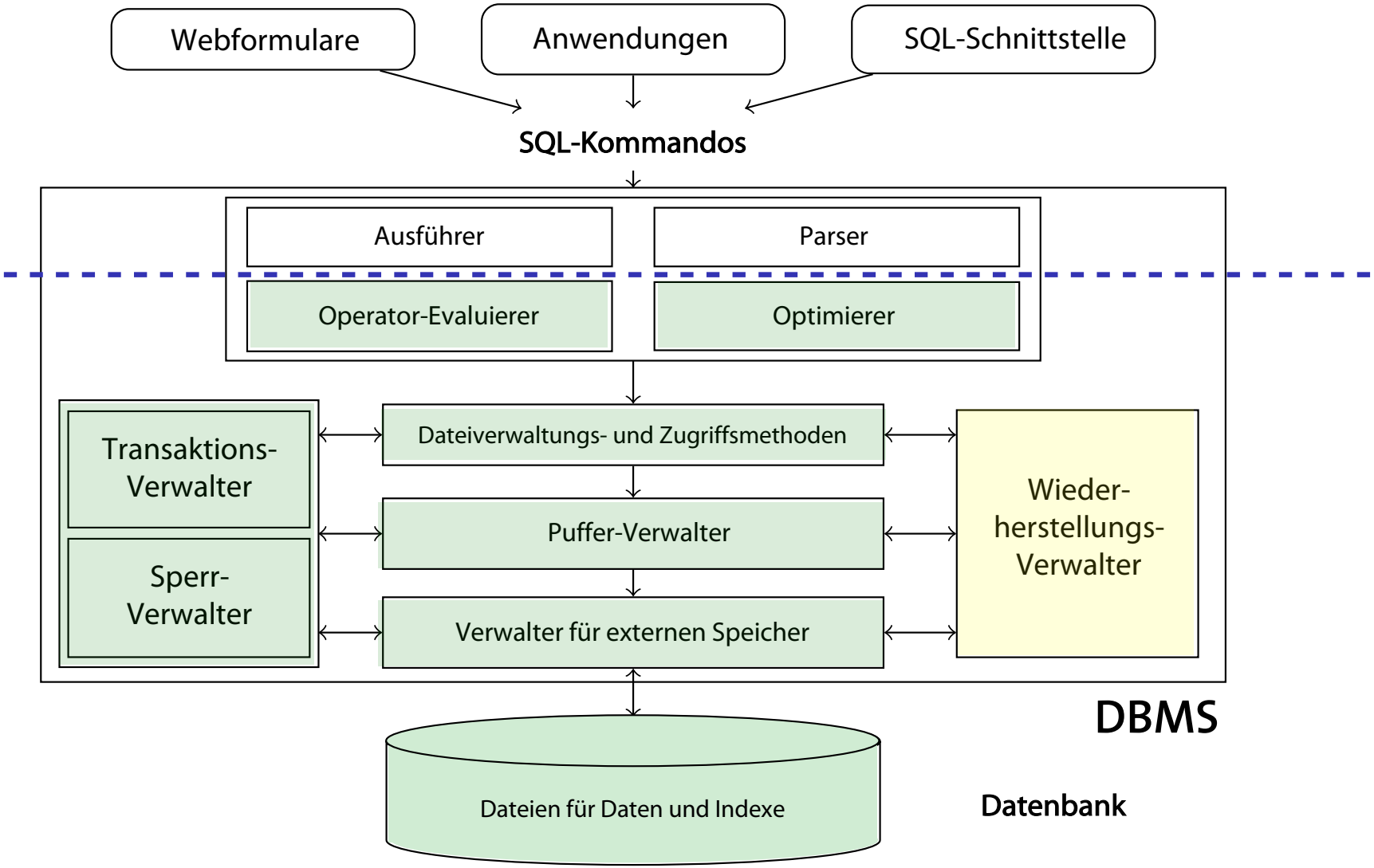
Transaktionsmanagement Teil 3

Dr. Özgür Özçep

Universität zu Lübeck

Institut für Informationssysteme

Wiederherstellung (Recovery)



Motivation

Wiederherstellung nach Fehlern

Drei Typen von Fehlern

- Transaktionsfehler (Prozessfehler)
  - Eine Transaktion wird abgebrochen (abort)
  - Alle Änderungen müssen ungeschehen gemacht werden

Wiederherstellung nach Fehlern

Drei Typen von Fehlern

- Transaktionsfehler (Prozessfehler)
  - Eine Transaktion wird abgebrochen (abort)
  - Alle Änderungen müssen ungeschehen gemacht werden
- Systemfehler
  - Datenbank- oder Betriebssystem-Crash, Stromausfall, o.ä.
  - Änderungen im Hauptspeicher sind verloren
  - Sicherstellen, dass keine Änderungen mit Commit verloren gehen (oder ihre Effekte wieder herstellen) und alle anderen Transaktionen ungeschehen gemacht werden

Wiederherstellung nach Fehlern

Drei Typen von Fehlern

- Transaktionsfehler (Prozessfehler)
  - Eine Transaktion wird abgebrochen (abort)
  - Alle Änderungen müssen ungeschehen gemacht werden
- Systemfehler
  - Datenbank- oder Betriebssystem-Crash, Stromausfall, o.ä.
  - Änderungen im Hauptspeicher sind verloren
  - Sicherstellen, dass keine Änderungen mit Commit verloren gehen (oder ihre Effekte wieder herstellen) und alle anderen Transaktionen ungeschehen gemacht werden
- Medienfehler (Gerätefehler)

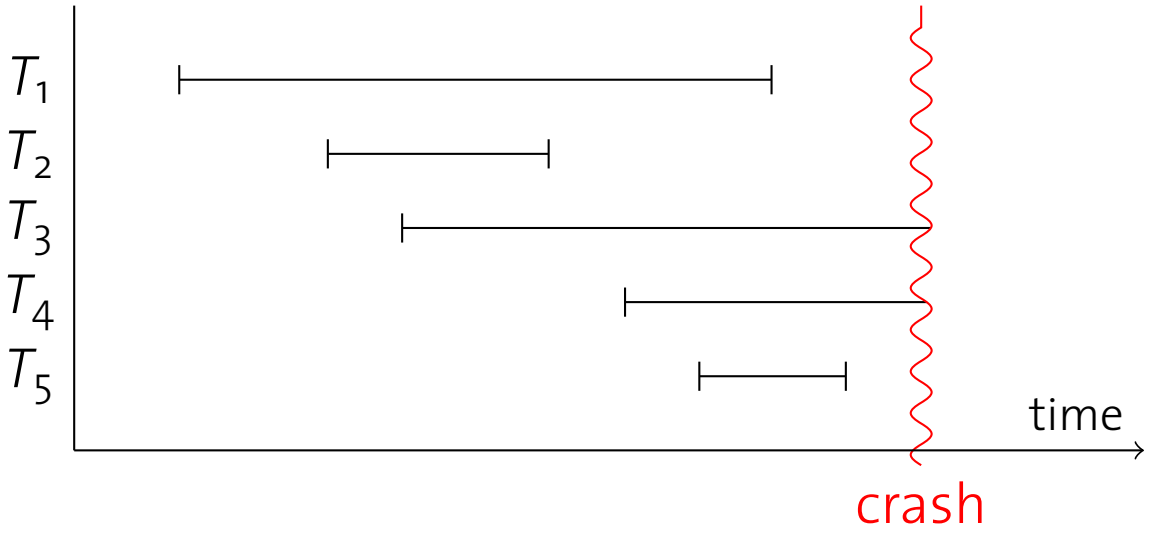
Wiederherstellung nach Fehlern

Drei Typen von Fehlern

- Transaktionsfehler (Prozessfehler)
- Systemfehler
- Medienfehler (Gerätefehler)
  - Crash von Festplatten, Feuer, Wassereinbruch
  - Wiederherstellung von externen Speichermedien

Trotz Fehler müssen Atomarität und Durabilität garantiert werden (ACID-Bedingungen)

Beispiel: System- oder Medienfehler



- Transaktionen T_1, T_2 und T_5 wurden vor dem Ausfall erfolgreich beendet → Dauerhaftigkeit: Es muss sicher-gestellt werden, dass die Effekte beibehalten werden oder wiederhergestellt werden können (redo)
- Transaktionen T_3 und T_4 wurden noch nicht beendet → Atomarität: Alle Effekte müssen rückgängig gemacht werden

Arten von Speichern

Wir nehmen an, es gibt drei Arten von Speichern

- Flüchtige Speicher
  - Wird vom Pufferverwalter verwendet
    (für Seitencache und auch Transaktions-Verwaltungsdaten)
- Nicht-flüchtige Speicher
  - Festplatten, Solid-State Drives
- Stabile Speicher
  - Nicht-flüchtiger Speicher, der alle drei Arten von Fehlern überlebt. Stabilität kann durch Replikation auf mehrere Platten erhöht werden (auch: Bänder)

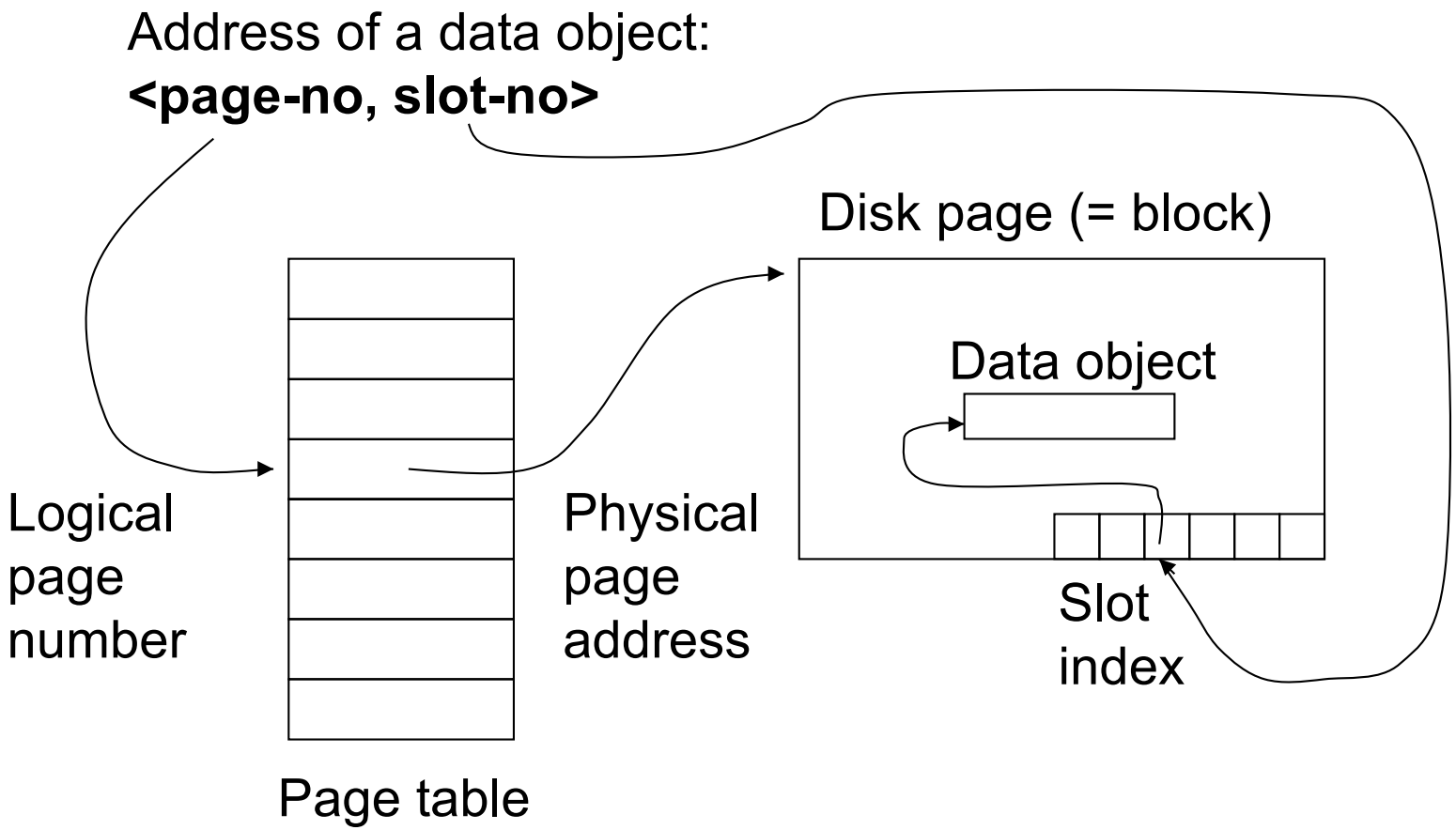
Vergleiche Arten von Fehler und Arten von Speichern

Schattenseiten

Schatten-Seiten-Verwaltung

- Fehler können zu jeder Zeit auftreten, also muss das System jederzeit in einen konsistenten Zustand zurückführbar sein
- Dies kann durch Redundanz erreicht werden
- Schatten-Seiten (eingeführt durch IBMs „System R“)
  - Von jeder Seite werden zwei Versionen geführt
  - Aktuelle Version/current (Arbeitskopie, copy-on-write)
  - Schatten-Seite/shadow (konsistente Version auf nicht-flüchtigem Speicher zur Wiederherstellung)

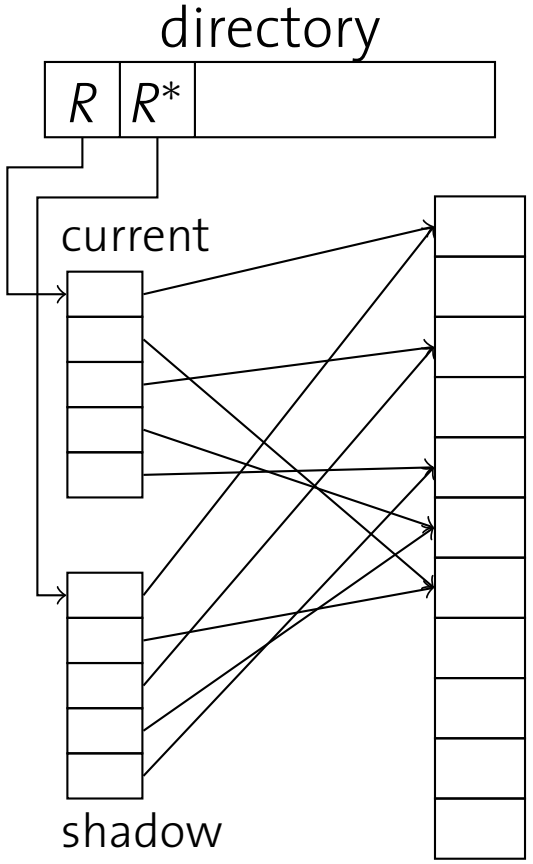
Verwalter für externen Speicher: Indirekte Adressierung



Shadow-Paging: Funktionen

- **Anforderung einer Seite:**
  - Ergänze Seitentabelle um Eintrag
  - Kopiere neuen Seitentabelleneintrag in die Schattentabelle

- **Schreiben einer Seite** (z.B. bei Verdrängung aus Puffer)**:**
  - Fordere neue Seite an
  - Schreibe Pufferinhalt
  - Trage Zeiger auf neue Seite in aktuelle Seitentabelle ein (überschreibe darin enthaltenen Zeiger auf das Original)

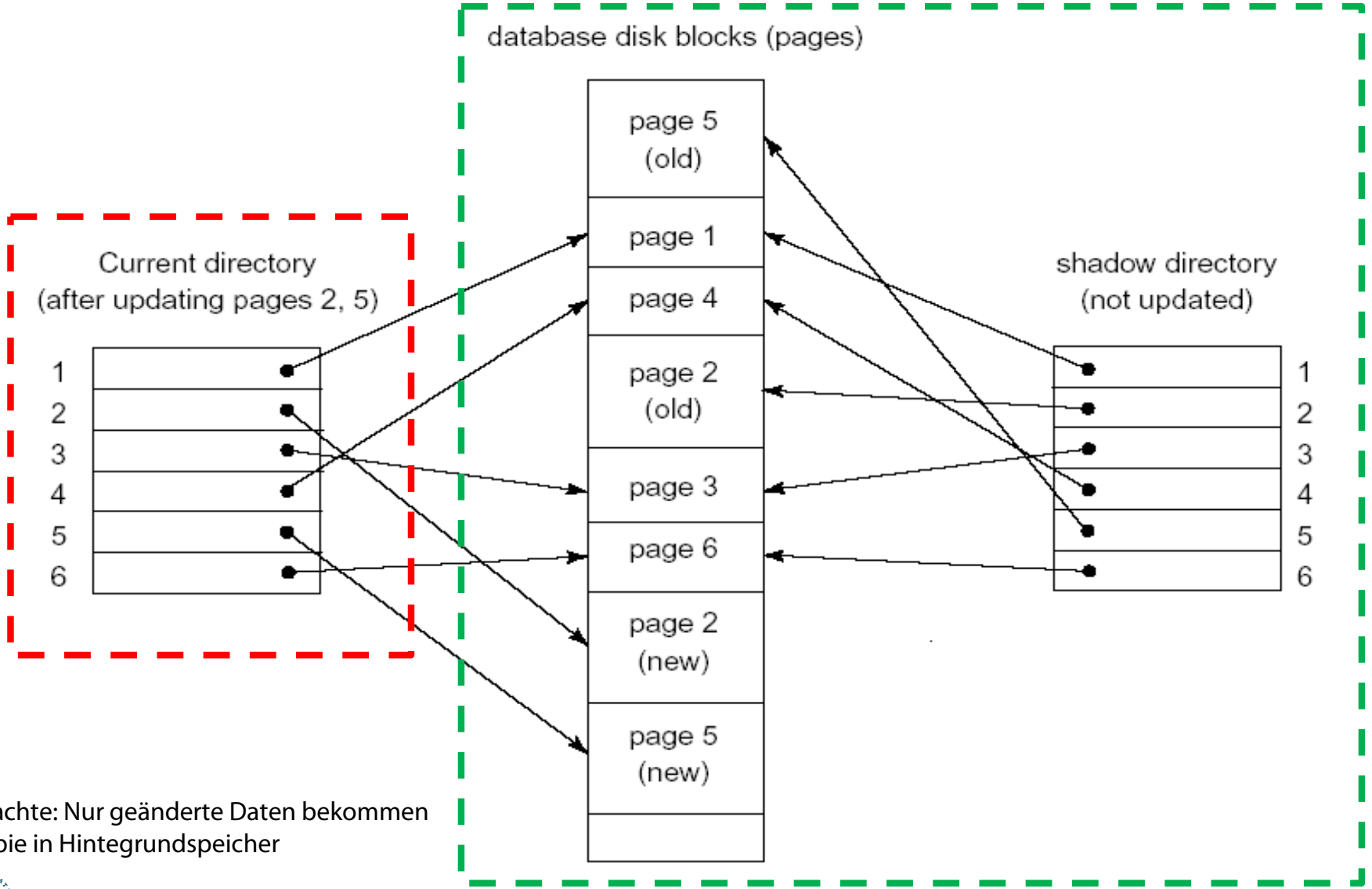


(Ausgangszustand: Current = shadow)

Beispiel

Nichtflüchtiger Speicher

Flüchtiger Speicher



Beachte: Nur geänderte Daten bekommen Kopie in Hintegrundspeicher

Shadow-Paging: Funktionen (2)

- **Commit:**
  - Übernehme aktuelle Tabelle bzw. die darin modifizierten Seiten in nichtflüchtigen Speicher
    - Atomare Ausführung ist nicht trivial
    - Kopie der Relation und atomares Umsetzen der Basisreferenz
  - Verwirf Schattentabellenseiten, d.h. gib referenzierte alte Seiten wieder frei
- **Abort/Recovery:**
  - Bzgl. alter Seiten ist nichts zu tun (es wurde auf Kopie gearbeitet)
  - Aktuelle Version überschrieben durch Schattenversion
  - Gib nicht referenzierte Seiten im nichtflüchtigen Speicher wieder frei (Garbage Collection von modifizierten Seiten)

Schatten-Seiten: Diskussion

- Wiederherstellung schnell für ganze Relationen/Dateien
- Um Persistenz (Durabilität) sicherzustellen, müssen modifizierte Seiten bei einem Commit in nicht-flüchtigen Speicher (z.B. Festplatte) geschrieben werden (force to disk)
- Nachteile:
  - Hohe I/O-Kosten, keine Verwendung von Cache möglich
  - Langsame Antwortzeiten
- Besser: No-Force-Strategie, Verzögerung des Schreibens
- Transaktion muss neu abgespielt werden können (Redo), auch für Änderungen, die nicht gespeichert wurden ->

  Logging nötig

Gray et al.. The Recovery Manager of the System R Database Manager. ACM Comp. Surv., vol. 13(2), June 1981.

Schatten-Seiten: Diskussion

- Schatten-Seiten ermöglichen den Einsatz von „frame stealing“, das Stehlen der Rahmen im Pufferverwalter. Seiten werden möglicherweise sofort auf die Platte geschrieben (sogar bevor Transaktion erfolgreich beendet wird)
  - Stehlen erfolgt durch andere Transaktionen
  - Stehlen kann nur erfolgen, wenn Seite nicht gepinnt/fixiert ist
  - Geänderte Seiten (dirty pages) werden auf die Platte geschrieben
- Diese Änderungen müssen ungeschehen gemacht werden während der Wiederherstellung
  - Leicht möglich durch Schatten-Seiten

ARIES-Wiederherstellungsmethode

Effekte, die Wiederherstellung berücksichtigen muss

- Entscheidungen zur Strategie haben Auswirkungen auf das, was bei der Wiederherstellung erfolgen muss

	force	**no force**
no steal	no redo no undo	must redo no undo
steal	no redo must undo	must redo must undo

- Bei der Kombination **steal** und **no force** wird zur Erhöhung der Nebenläufigkeit und der Performanz ein **redo** und ein **undo** implementiert

Write-Ahead-Log (WAL)

- Die ARIES[1]-Wiederherstellungsmethode verwendet ein Write-Ahead-Log zur Implementierung der notwendigen redundanten Datenhaltung
- Datenseiten werden in situ (update-in-place) modifiziert
- Für ein Undo müssen Undo-Informationen in eine Logdatei auf nicht-flüchtigem Speicher geschrieben werden, bevor eine geänderte Seite auf die Platte geschrieben wird

- Zur Persistenzsicherung muss zur Commit-Zeit Redo-Information sicher gespeichert werden (No-Force-Strategie: Daten auf der Platte enthalten alte Information)

[1] Algorithm for Recovery and Isolation Exploiting Semantics
Mohan et al. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks UsingWrite-Ahead Logging. ACM TODS, vol. 17(1), March 1992.

Inhalte des Write-Ahead-Logs

LSN	Type	TX	Prev	Page	UNxt	Redo	Undo
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

LSN (Log Sequence Number)

- Monoton steigende Zahl, um Einträge zu identifizieren
  Trick: Verwende Byte-Position des Log-Eintrags

Typ (Log Record Type)

- Repräsentiert, ob Update-Eintrag (UPD), End-of-Transaction-Eintrag (EOT), Compensation-Log-Record (CLR)

TX (Transaktions-ID)

- Transaktionsbezeichner (falls anwendbar)

Inhalte des Write-Ahead-Logs (Forts.)

Prev (Previous Log Sequence Number)

- LSN des vorigen Eintrags von der gleichen Transaktion (falls anwendbar, am Anfang steht '-')

Page (Page Identifier)

- Seite, die aktualisiert wurde (nur für UPD und CLR)

UNxt (LSN Next to be Undone)

- Nur für CLR: Nächster Eintrag der Transaktion, der während des Zurückrollens bearbeitet werden muss

Redo

- Information zum erneuten Erzeugen einer Operation

Undo

- Information zum Ungeschehenmachen einer Operation

Beispiel

Transaction 1	Transaction 2
$a \leftarrow$ `read(`A`);`	
	$c \leftarrow$ `read(`C`);`
$a \leftarrow a - 50$;	
	$c \leftarrow c + 10$;
`write(`a`,`A`);`	
	`write(`c`,`C`);`
$b \leftarrow$ `read(`B`);`	
$b \leftarrow b + 50$;	
`write(`b`,`B`);`	
`commit;`	
	$a \leftarrow$ `read(`A`);`
	$a \leftarrow a - 10$;
	`write(`a`,`A`);`
	`commit;`

LSN	Type	TX	Prev	Page	UNxt	Redo	Undo
1	UPD	T_1	–	$\cdots$		$A := A - 50$	$A := A + 50$
2	UPD	T_2	–	$\cdots$		$C := C + 10$	$C := C - 10$
3	UPD	T_1	1	$\cdots$		$B := B + 50$	$B := B - 50$
4	EOT	T_1	3	$\cdots$			
5	UPD	T_2	2	$\cdots$		$A := A - 10$	$A := A + 10$
6	EOT	T_2	5	$\cdots$			

Logische Protokollierung
(alternativ: physikalisch)

Redo/Undo-Information

ARIES nimmt seitenorientiertes Redo an

- Keine anderen Seiten müssen angesehen werden, um eine Operation erneut zu erzeugen
- Z.B.: Physikalisches Logging
  - Speicher von Byte-Abbildern von (Teilen von) Seiteninhalten
  - Vorher-Abbild (Abbild vor der Operation)
  - Nachher-Abbild (Abbild nach der Operation)
  - Wiederherstellung unabhängig von Objekten
    - Objekt-Struktur braucht nicht bekannt zu sein, nur Seitenstruktur relevant
- Logisches Redo (A = A + 50 oder ‚Füge Tupel ein in R')
  - Redo wird vollständig durchgeführt, auch Indexeinträge würden neu generiert, inkl. Aufspaltung usw.

Redo/Undo-Information

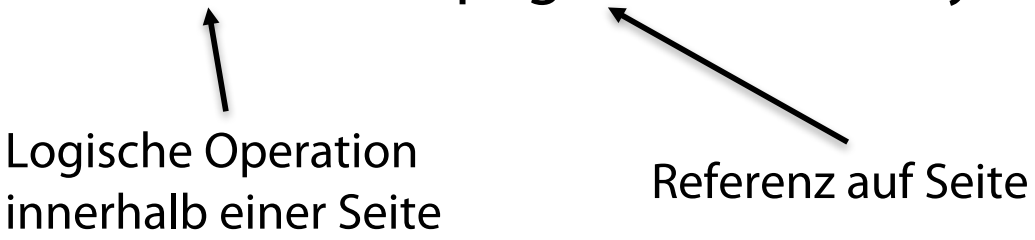
- ARIES unterstützt logisches Undo
- Seitenorientiertes Undo kann zu kaskadierenden Rückroll-Situationen führen
  - T_1 und T_2 manipulieren dieselbe Seite, aber konfliktfrei unterschiedliche Ressourcen
  - Ein Abbruch von einer Transaktion führt zwangsläufig zu einem unnötigen Abbruch der anderen Transaktion
- Logisches Undo erhöht also die Nebenläufigkeit
- Aber: Eine einzelne logische Operation wie insert tuple into table R führt mehre Schritte nach sich:
  - Insert tuple into data pages ,
  - insert into index
  - etc.

Kompromiss

- Kombiniere beide Ansätze: physikalisch auf Seiten zugreifen, logisch innerhalb einer Seite

- Beispiel Logeintrag

  [...,insert,...,page 4711,...,record value r]
  [...,ix insert,...,ix page 0815,...,ix key: k1, rid: v]
  [...,ix insert,...,ix page 4242,...,ix key: k2, rid: v]



Schreiben von Log-Einträgen

- Aus Performanzgründen werden Log-Einträge zunächst in den flüchtigen Speicher geschrieben
- Zu bestimmten Zeiten werden die Einträge bis zu einer bestimmten LSN in stabilen Speicher geschrieben
  - Alle Einträge bis zum EOT einer Transaktion T werden auf die Platte geschrieben, wenn T erfolgreich beendet (um ein Redo der Effekte von T vorzubereiten)
  - Wenn eine Datenseite p auf die Platte geschrieben wird, werden vorher die letzten Modifikationen von p im WAL auf die Platte geschrieben (zur Vorbereitung eines Undo)
- Die Größe des Logs nimmt immer weiter zu (s. aber Schnappschüsse weiter unten)

Normaler Verarbeitungsmodus

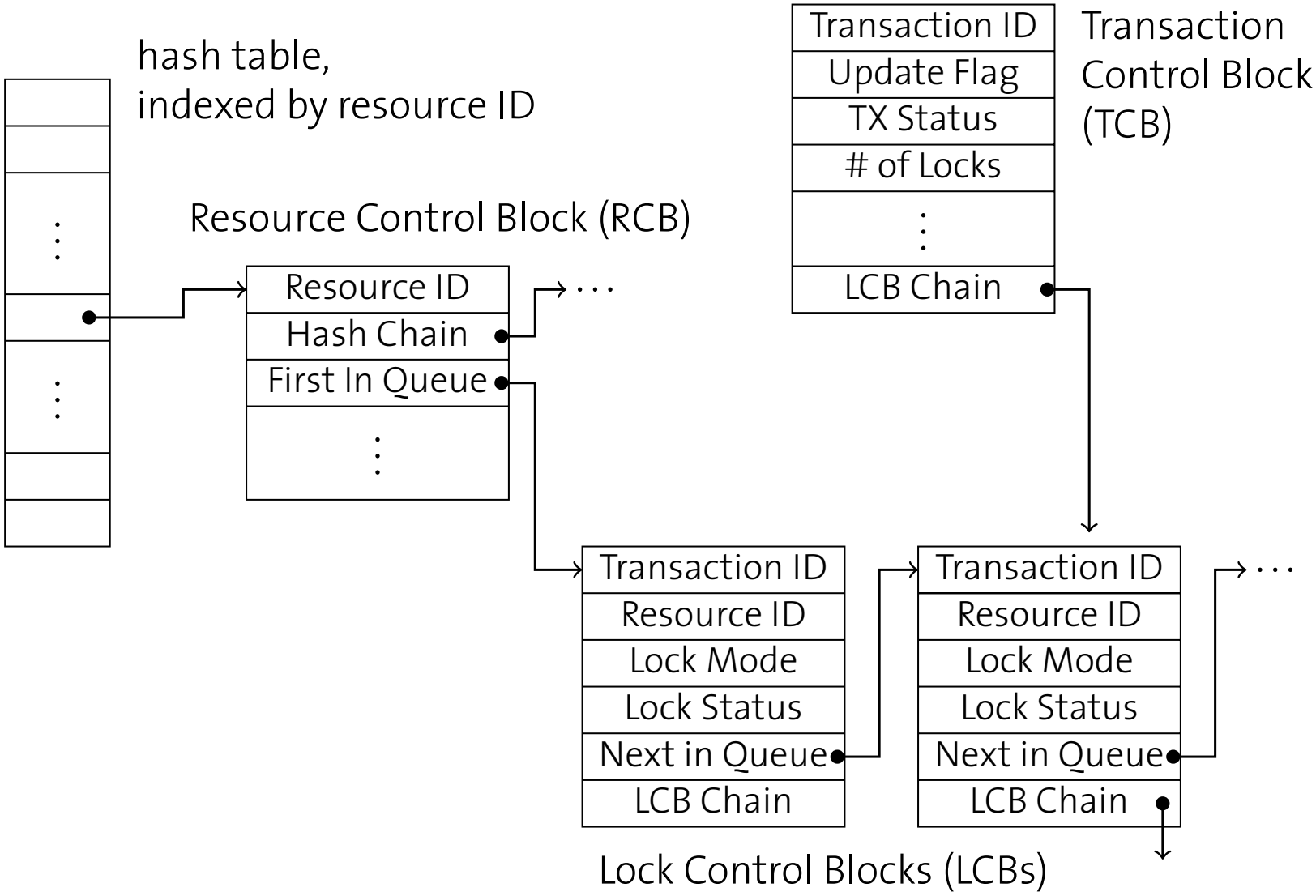
Während der normalen Transaktionsverarbeitung werden zwei Dinge im Transaktionskontrollblock gespeichert

- LastLSN (Last Log Sequence Number)
  - LSN des letzten geschriebenen Log-Eintrags für die Transaktion
- UNxt (LSN Next to be Undone)
  - LSN des nächsten Eintrags, der beim Rückrollen betrachtet werden muss

Wenn eine Aktualisierung einer Seite p durchgeführt wird

- wird ein Eintrag r ins WAL geschrieben und
- die LSN von r im Seitenkopf von p gespeichert

Datenstruktur zur Buchführung



Rückrollen einer Transaktion

Schritte zum Rückrollen einer Transaktion T:

- Abarbeiten des WAL in Rückwärtsrichtung
- Beginn bei Eintrag, auf den UNxt im Transaktionskontrollblock von T zeigt
- Finden der übrigen Einträge von T durch Verfolgen der Prev- und UNxt-Einträge im Log

Undo-Operationen modifizieren ebenfalls Seiten

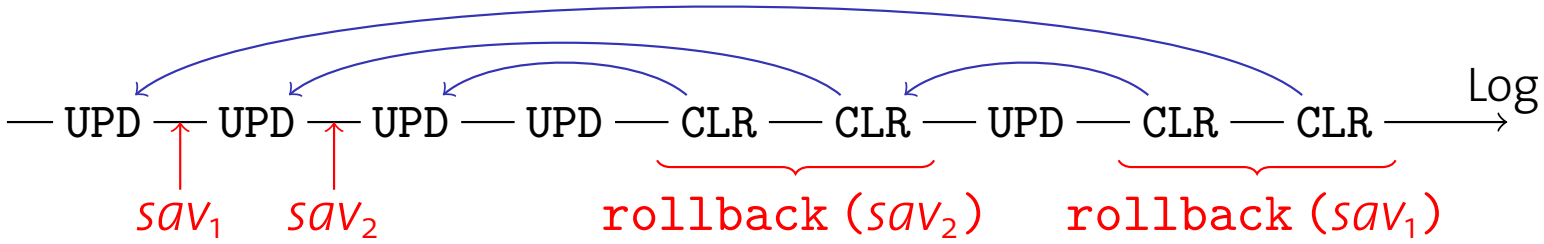
- Logging der Undo-Operationen im WAL
- Verwendung von Compensation-Log-Record (CLRs) für diesen Zweck

Rückrollen einer Transaktion

```
1  Function: rollback (SaveLSN, T)
2  UndoNxt ← T.UNxt ;
3  while SaveLSN < UndoNxt do
4      LogRec ← read log entry with LSN UndoNxt ;
5      switch LogRec.Type do
6          case UPD
7              perform undo operation LogRec.Undo on page LogRec.Page ;
8                      write log entry
                   < LSN',   CLR, T, T.LastLSN, LogRec.Page, LogRec.Prev, ··· , ∅⟩ ;
9              In page header of LogRec.Page: Set LSN = LNS'
10             T.LastLSN ← LNS'
11         case CLR
12             UndoNxt ← LogRec.UNxt ;
13     T.UNxt ← UndoNxt ;
```

Rückrollen einer Transaktion

- Transaktionen können auch partiell zurückgerollt werden (zur SaveLSN)
- Das UNxt-Feld in einem CLR zeigt auf den Logeintrag vor demjenigen, der ungeschehen gemacht wurde



Wiederherstellung nach Ausfall

Neustart nach einem Systemabsturz in drei Phasen

1. Analyse-Phase:
   - Lies Log in Vorwärtsrichtung
   - Bestimme Transaktionen, die aktiv waren, als der Absturz passierte (Pechvögel/loser transactions)
2. Redo-Phase:
   - Spiele Log erneut ab (in Vorwärtsrichtung), um das System in den Zustand vor dem Fehler zu bringen
3. Undo-Phase:
   - Rolle Pechvögel-Transaktionen zurück, indem das Log in Rückwärtsrichtung abgearbeitet wird (wie beim normalen Zurückrollen)

Analyse-Phase

```
1  Function: analyze ()
2  foreach log entry record LogRec do
3      switch LogRec.Type do
4          create transaction control block for LogRec.TX if necessary ;
5          case UPD or CLR
6              LogRec.TX.LastLSN ← LogRec.LSN ;
7              if LogRec.Type = UPD then
8                  LogRec.TX.UNxt ← LogRec.LSN ;
9              else
10                 LogRec.TX.UNxt ← LogRec.UNxt ;
11         case EOT
12             delete transaction control block for LogRec.TX ;
```

Redo-Phase

```
1 Function: redo ()
2 foreach log entry record LogRec do
3     switch LogRec.Type do
4         case UPD or CLR
5             v ← pin (LogRec.Page) ;
6             if v.LSN < LogRec.LSN then
7                 perform redo operation LogRec.Redo on v ;
8                 v.LSN ← LogRec.LSN ;
9             unpin (v, ···) ;
```

Auch beim Wiederherstellen können Abstürze eintreten

- Undo und Redo einer Transaktion müssen idempotent sein
  - redo(redo(T)) = redo(T) // z.B. nicht zweimal dasselbe Tupel einfügen bei insert
  - undo(und(T)) = undo(T)
- Prüfe LSN vor der Redo-Operation (Zeile 6)

Redo-Phase

- Beachte, dass alle Operationen (auch solche von Pechvögeln) in chronologischer Ordnung erneut durchgeführt werden
- Nach der Redo-Phase ist das System im gleichen Zustand, wie zum Fehlerzeitpunkt
  - Einige Log-Einträge sind noch nicht auf der Platte, obwohl erfolgreich beendete Transaktionen ihre Änderung geschrieben hätten. Alle anderen müssten ungeschehen gemacht werden
- Wir müssen hinterher alle Effekte von Pechvögeln ungeschehen machen
- Als Optimierung kann man den Puffermanager instruieren, geänderte Seiten vorab zu holen (Prefetch)

Undo-Phase

- Die Undo-Phase ist ähnlich zum Rückrollen im normalen Betrieb
- Es werden mehrere Transaktionen auf einmal zurückgerollt (alle Pechvögel)
- Alle Pechvögel werden vollständig zurückgerollt

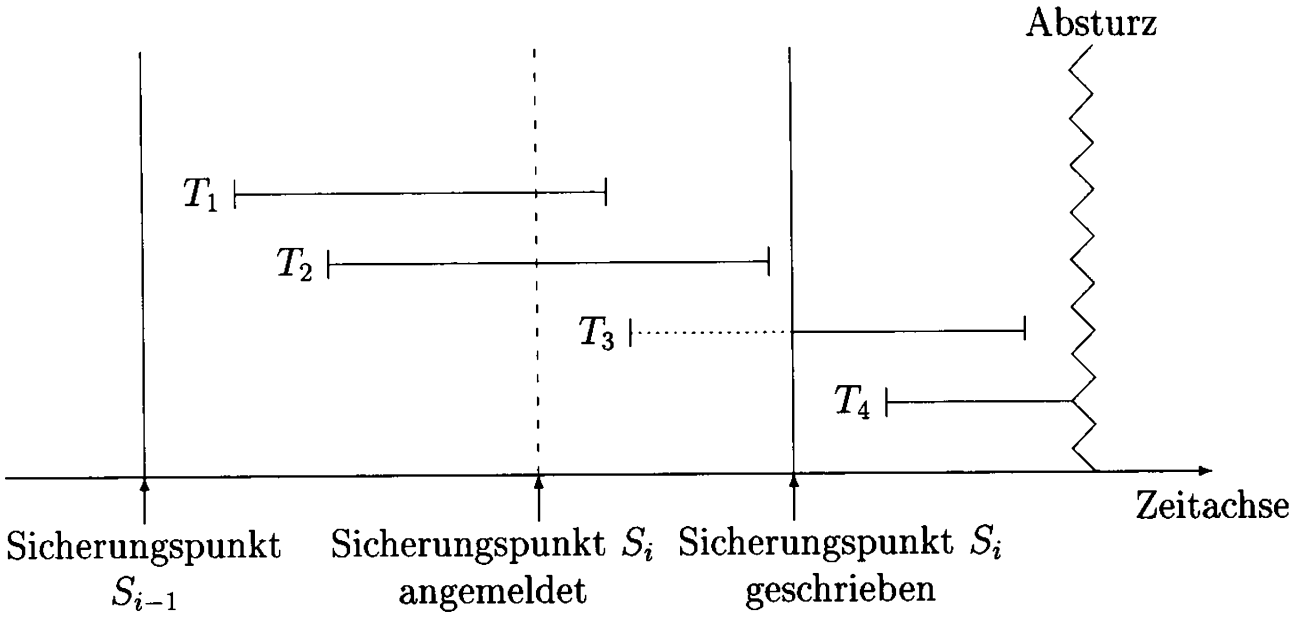
Undo-Phase

```
1  Function: undo ()
2  while transactions (i.e., TCBs) left to roll back do
3      T ← TCB of loser transaction with greatest UNxt ;
4      LogRec ← read log entry with LSN T.UNxt ;
5      switch LogRec.Type do
6          case UPD
7              perform undo operation LogRec.Undo on page LogRec.Page ;
8                      - write log entry
                   < LSN', CLR, T, T.LastLSN, LogRec.Page, LogRec.Prev, ··· , ∅⟩ ;
9              set LSN = LSN' in page header of LogRec.Page ;
10             T.LastLSN ← LSN
11         case CLR
12             UndoNxt ← LogRec.UNxt ;
13     T.UNxt ← UndoNxt ;
14     if T.UNxt = '–' then
15         write EOT log entry for T ;
16         delete TCB for T ;
```

Checkpointing

- WAL ist eine immer-wachsende Log-Datei, die bei der Wiederherstellung gelesen wird
- In der Praxis sollte die Datei nicht zu groß werden (Wiederherstellung dauert zu lange)
- Daher wird ab und zu ein sog. Checkpoint erstellt
  - Schwergewichtiger Checkpoint:
    Speicherung aller geänderter Seiten auf der Platte, dann Verwaltungsinformation für Checkpoint schreiben (Redo kann dann ab Checkpoint erfolgen)
  - Leichtgewichtiger Checkpoint („Fuzzy checkpointing“):
    Speichere Information bzgl. geänderter Seiten in Log, aber keine Seiten (Redo ab Zeitpunkt kurz vor Checkpoint)

Beispiel schwergewichtiger Checkpoint



- Nach Absturz Logdatei ab S_i nötig.
- Hierfür
  - Für T_1 und T_2 nichts zu tun (da bereits persistent)
  - T_3 nach Anmeldung von S_i verzögert;
  - Merke Änderungen von T_3 (genutzt für Redo)
  - Merke Änderungen von T_4 (genutzt für Undo)

Leichtgewichtiger Checkpoint

Schreibe periodisch Checkpoint in drei Phasen

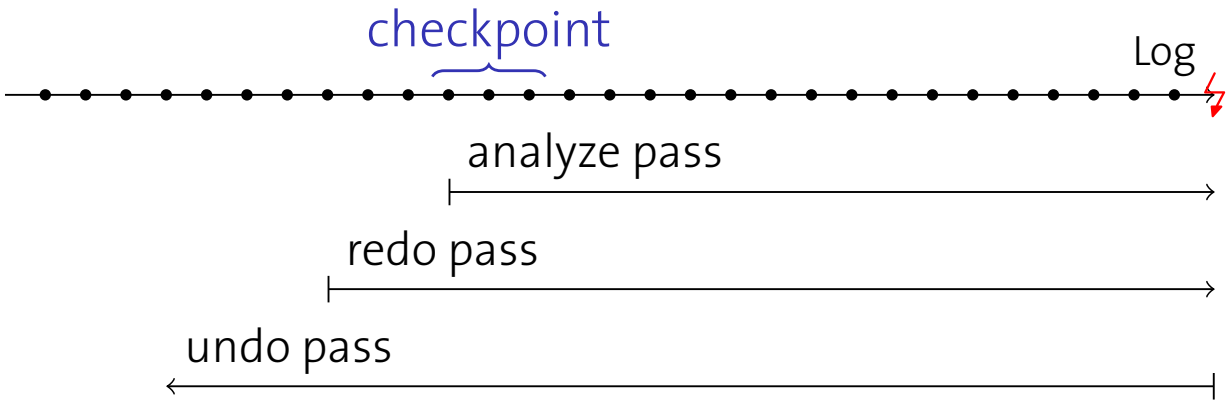
1. Schreibe Begin-Checkpoint-Logeintrag BCK
2. Sammle Information
   - über geänderte Seiten im Pufferverwalter und dem LSN ihrer letzten Modifikationsoperation und
   - über alle aktiven Transaktionen (und ihrer LastLSN und UNxt TCB-Einträge)

   Schreibe diese Information in End-Checkpoint Eintrag ECK
3. Lass Master-Record in bekannter Position auf Platte auf LSN vom BCK-Logeintrag zeigen

Wiederherstellung mit leichtgew. Checkpoint

Während der Wiederherstellung

- Starte Analyse beim BCK-Eintrag im Master-Record (anstelle vom Anfang des Logs)
- Wenn der ECK-Eintrag gelesen wird
  - Bestimme kleinste LSN für die Redo-Verarbeitung und
  - Erzeuge TCBs für alle Transaktionen im Checkpoint



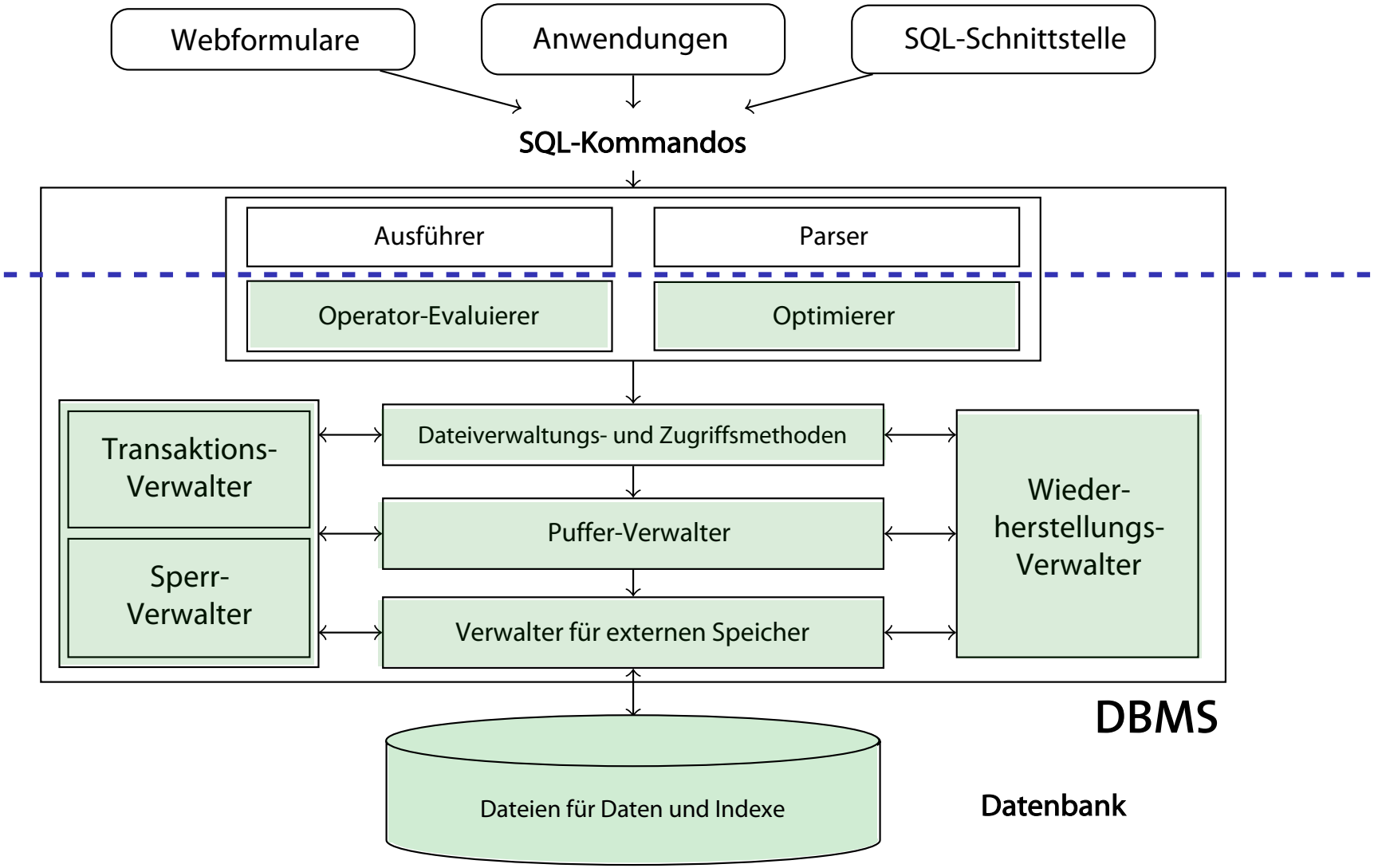
Medien-Wiederherstellung

- Bisher für Wiederherstellung angenommen, dass DB und Log-Datei nicht beschädigt wurden (keine Medienfehler).
- Um Medienfehler zu kompensieren, muss periodisch eine Sicherungskopie erstellt werden (nicht-flüchtiger Speicher)
  - Kann während des Betriebs erfolgen, wenn WAL auch gesichert wird
  - Wenn Pufferverwalter verwendet wird, reicht es, das Log vom Zeitpunkt des Backups zu archivieren
- Anderer Ansatz:
  Spiegeldatenbank auf anderem Rechner

Zusammenfassung Transaktionsmanagement

- ACID und Serialisierbarkeit
  - Vermeiden von Anomalien durch Nebenläufigkeit
  - Serialisierbarkeit reicht für Isolation
- Zwei-Phasen-Sperrprotokoll
  - 2PL ist eine praktikable Technik, um Serialisierbarkeit zu garantieren (meist wird strikte 2PL verwendet)
  - In SQL-92 können sog. Isolationsgrade eingestellt werden (Abschwächung der ACID-Bedingungen)
- Nebenläufigkeit in B-Bäumen
  - Spezialisierte Protokolle (WTL) zur Flaschenhalsvermeidung
- Wiederherstellung (ARIES)
  - Write-Ahead-Log, Checkpoints

Das Gesamtbild der Architektur



Klausur

- Für erste Klausur alles relevant bis auf Recoverymanagement (dieser Foliensatz: Transaktionsmanagement Teil 3).
- Übungen bilden gute Vorbereitung
  - Insbesondere Verständnisfragen (Ja-Nein-Fragen mit Begründung) werden vorkommen
  - Aber: Keine langen Rechenaufgaben

- Organisatorisches
  - Raum checken in Moodle