

5. Crash- und Medien-Recovery

■ Crash-Recovery

- Restart-Prozedur
- Redo-Recovery
- Einsatz von Compensation Log Record / Redo ALL (Aries)-Verfahren
- Beispiel

■ Medien-Recovery (Behandlung von Gerätefehlern)

- Alternativen
- Inkrementelles Dumping
- Erstellung transaktionskonsistenter Archivkopien
- Single Page Recovery



Crash-Recovery

- jüngster transaktionskonsistenter DB-Zustand aus permanenter DB und (temporärer) Log-Datei herzustellen
- Idempotenz der Recovery: Fehler während Recovery müssen behandelbar sein
- bei Update-in-Place (NonATOMIC):
 - Zustand der permanenten DB nach Crash unvorhersehbar (nicht aktionskonsistent)
 - ein Block der permanenten DB ist entweder
 - aktuell oder
 - veraltet (NOFORCE) \Rightarrow REDO oder
 - „schmutzig“ (STEAL) \Rightarrow UNDO



Crash-Recovery

(NonAtomic, Steal, NoForce, Fuzzy Checkpoint)

- Log-Datei wird 3-mal gelesen:

1. Analyse-Lauf (vom letzten Checkpoint bis zum Log-Ende):

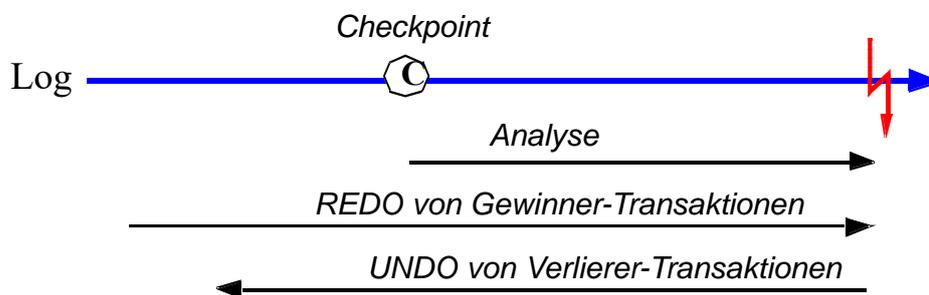
- Bestimmung von **Gewinner-** und **Verlierer-Transaktionen** sowie der Seiten, die von ihnen geändert wurden

2. REDO-Lauf: Vorwärtslesen des Logs (Startpunkt MinDirtyPageLSN)

- Wiederholung der Änderungen von Gewinner-Transaktionen (*selektives Redo*) bzw. von allen Transaktionen (*vollständiges Redo*), sofern erforderlich (LSN-Vergleich)
- betroffene Seiten sind einzulesen

3. UNDO-Lauf: Rücksetzen der Verlierer-Transaktionen

- Rückwärtslesen des Logs bis zum BOT-Satz der ältesten Verlierer-Transaktion

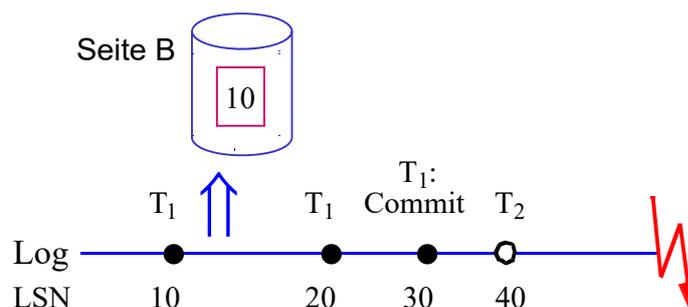


Redo-Recovery

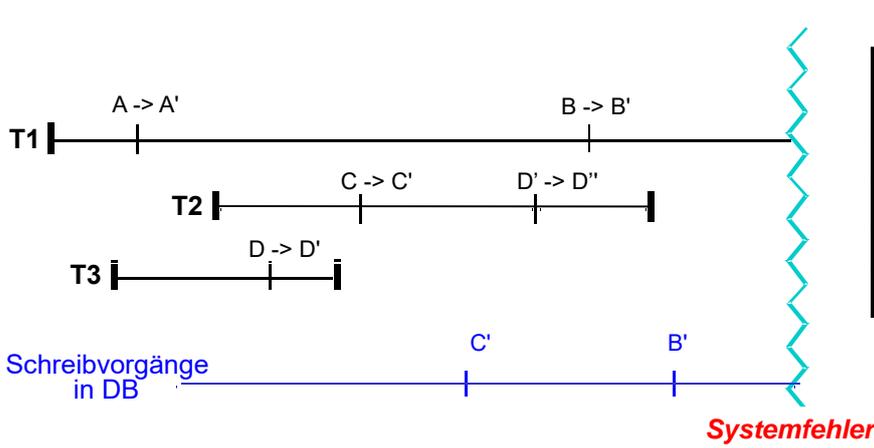
- physiologisches und physisches Logging: Notwendigkeit einer Redo-Aktion für Log-Satz L wird über PageLSN der betroffenen Seite B angezeigt

```
if (B nicht gepuffert) then (lies B in den Hauptspeicher ein);  
if LSN(L) > PageLSN(B) then do;  
    REDO (Änderung aus L);  
    PageLSN(B) := LSN(L);  
end;
```

- wiederholte Anwendung des Log-Satzes (z.B. nach mehrfachen Fehlern) erhält Korrektheit (Idempotenz der Recovery)



Beispiel



DB-Inhalt

Seite	Page-LSN
A	5
B'	90
C'	70
D	8

Log-Inhalt

LSN	Log-Satz
10	BOT (T1)
20	BOT (T3)
30	T1, A -> A'
40	BOT (T2)
50	T3, D -> D'
60	Commit (T3)
70	T2, C -> C'
80	T2, D' -> D''
90	T1, B -> B'
100	Commit (T2)

Analyselauflauf:

Verlierer: T1
 Gewinner: T2, T3
 relevante Seiten: A, B, C, D

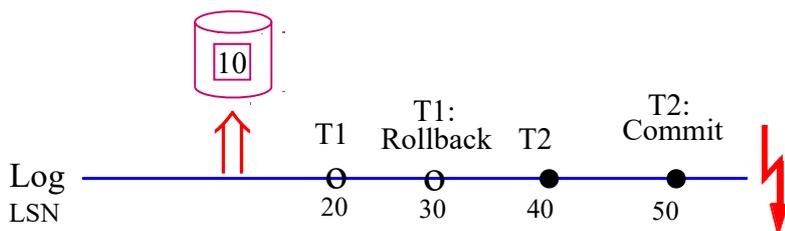
Redo-Lauf: Redo (T3, D, 50), PageLSN (D) := 50

Undo-Lauf:



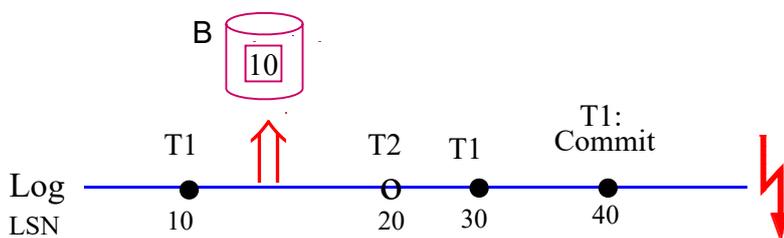
Probleme bei LSN-Verwendung (für Undo)

- UNDO für Verlierer, wenn PageLSN >= LSN (Undo-Log-Satz) ???
- Problem 1: Transaktionsrücksetzungen



Redo-Lauf: Änderung von T2 wird wiederholt (Page-LSN := 40)
 Undo-Lauf: Änderung 20 von T1 wird zurückgesetzt (da 20 < 40) -> Fehler

- Problem 2: Satzsperrungen

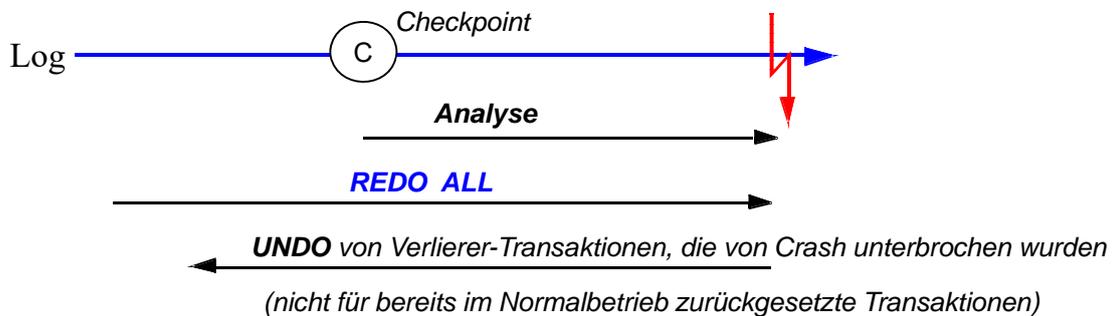


REDO von Änderung 30 (Page-LSN := 30)
 UNDO von Änderung 20, obwohl nicht in der Seite vorhanden



Lösung der Probleme

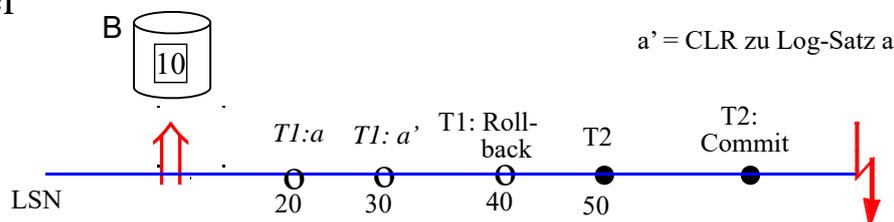
- Protokollieren der Undo-Operationen durch *Compensation Log Records (CLR)*
- vollständiges Redo oder "*Repeating History*", d.h. im Redo-Lauf werden alle Änderungen (auch von Verlierer-Transaktionen) wiederholt



- Umsetzung durch *ARIES-Protokoll (Algorithm for Recovery and Isolation Exploiting Semantics)*
 - entwickelt von Mohan et al. (IBM Research)
 - realisiert in mehreren kommerziellen DBS

Compensation Log Records (CLR)

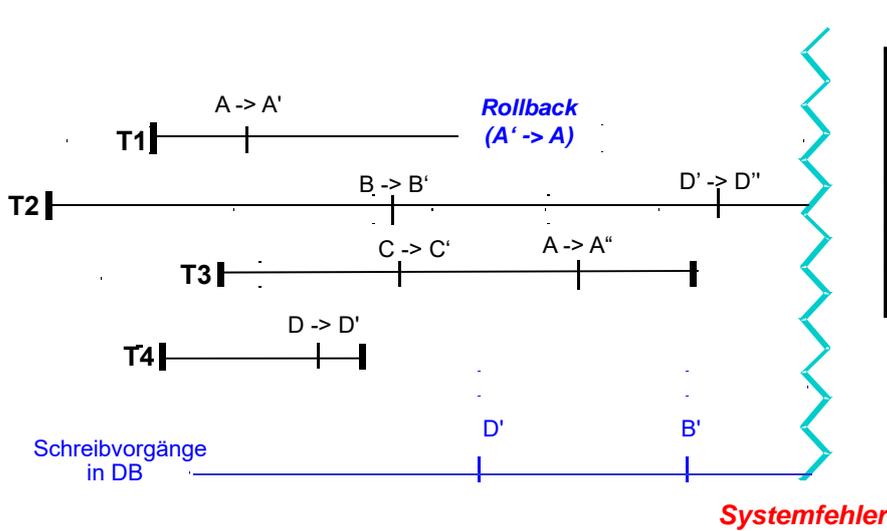
- *Compensation Log Record (CLR)* = Log-Satz einer Undo-Operation
- CLR wird geschrieben
 - für jede Seitenänderung beim Rollback im Normalbetrieb
 - für jede Undo-Operation während der Crash-Recovery
- Beispiel



- vollständiges Redo: Redo (T1, B, 20), PageLSN (B) := 20
Redo (T1, B, 30), PageLSN (B) := 30
Redo (T2, B, 50), PageLSN (B) := 50

- CLR-Einsatz bei „Repeating History“ (ARIES)
 - Redo-Lauf : Wiederholung von Rollback-Operationen durch Anwendung der CLR-Sätze
 - Undo-Lauf: nur für Transaktionen, die bei Rechnerausfall aktiv waren (ohne LSN-Vergleich!)
 - Undo-Operationen wiederum zu protokollieren (CLR's)
 - PageLSN erhält LSN des CLR-Satzes

Beispiel



DB-Inhalt

Seite	Page-LSN
A	5
B'	60
C	10
D'	40

Log-Inhalt

LSN	Log-Satz
	...
20	BOT (T3)
30	T1, A -> A'
40	T4, D -> D'
50	Commit (T4)
60	T2, B -> B'
70	T3, C -> C'
80	T1, A' - A (CLR)
90	Rollback (T1)
100	T3, A -> A''
110	Commit (T3)
120	T2, D' -> D''

Analyselauflauf:

- Verlierer-Transaktionen: T1, T2; davon T2 noch aktiv zum Crash-Zeitpunkt
- Gewinner: T3, T4
- relevante Seiten: A, D, B, C

Vollständiges Redo:

Redo (T1, A, 30) PageLSN (A) := 30
 Redo (:=
 Redo (:=
 Redo (:=

Undo-Lauf: nur T2 betroffen



Recovery für Externspeicher (Magnetplatten)

■ Spiegelplatten

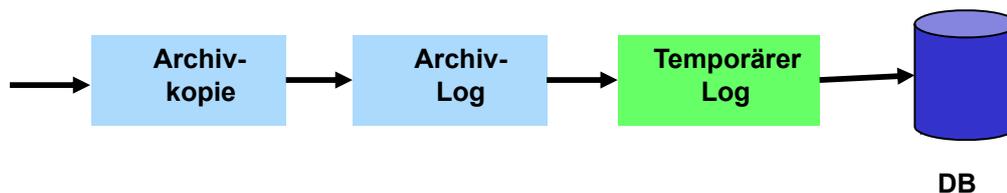
- schnellste und einfachste Lösung
- hohe Speicherkosten
- Doppelfehler nicht auszuschließen

■ Disk-Arrays (RAID-5)

- geringere Speicherkosten als Spiegelplatten, jedoch auch geringere Fehlertoleranz
- langsamerer Zugriff für einzelne Blöcke, dafür Unterstützung von E/A-Parallelität

■ Alternative: Archivkopie + Archiv-Log

- Archivkopie + Archiv-Log sind längerfristig verfügbar zu halten
- Führen von Generationen der Archivkopie
- Duplex-Logging für Archiv-Log



■ Archiv-Log offline aus (temporärer) Log-Datei ableitbar

■ Erstellung von Archivkopien und Archiv-Log erfolgt segmentorientiert

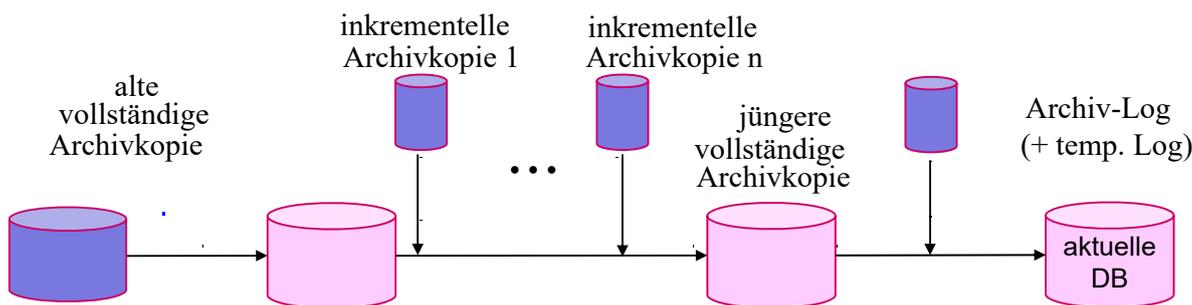


Erstellung der Archivkopie (Dump)

- Unterbrechung des Änderungsbetriebs zur Erstellung einer DB-Kopie i.a. nicht tolerierbar
- Alternativen:
 - **Online-Erstellung einer vollständigen Archivkopie (parallel zum Änderungsbetrieb)**
 - **Incremental Dumping**
 - Ableiten neuer Generationen aus 'Urkopie'
 - nur Änderungen seit der letzten Archivkopie werden protokolliert
 - Offline-Erstellung einer aktuelleren Kopie
- unterschiedliche Konsistenzgrade:
 - **Fuzzy Dump**
 - Kopieren der DB im laufenden Betrieb, kurze Lesesperren
 - (nur „saubere“ Seiten im Dump, keine Undo-Notwendigkeit, sondern nur REDO)
 - bei Plattenfehler Archiv-Log ab Beginn der Dump-Erstellung anzuwenden
 - **aktionskonsistente Archivkopie** (Voraussetzung bei logischem Operations-Logging)
 - **transaktionskonsistente Archivkopie** (Voraussetzung bei logischem Transaktions-Logging)
 - Black-/White-Verfahren
 - Copy-on-Update-Verfahren

Inkrementelles Dumping

- nur seit der letzten Archivkopie-Erstellung geänderte DB-Seiten werden archiviert



- Erkennung geänderter Seiten
 - Archivierungs-Bit pro Seite -> alle Seiten für Dump-Erstellung zu lesen
 - sehr hoher E/A-Aufwand
- besser: Verwendung separater Datenstrukturen (Bitlisten)
 - Änderungsbit zeigt Notwendigkeit, Seite in den nächsten Dump zu schreiben
 - Setzen des Änderungsbits falls $(\text{PageLSN der ungeänderten Seite}) < (\text{LSN zu Beginn des letzten Dumps})$

Black-/White-Verfahren

- Erzeugung transaktionskonsistenter Archivkopien
- Paint-Bit pro Seite:
 - weiß: Seite wurde noch nicht überprüft
 - schwarz: Seite wurde bereits verarbeitet
- Modified-Bit pro Seite zeigt an, ob eine Änderung seit Erstellung der letzten Archivkopie erfolgte (inkrementelles Dumping)
- spezieller Schreibprozess zur Erstellung der Archivkopie
 - färbt alle weißen Seiten schwarz und schreibt geänderte Seiten in Archivkopie:

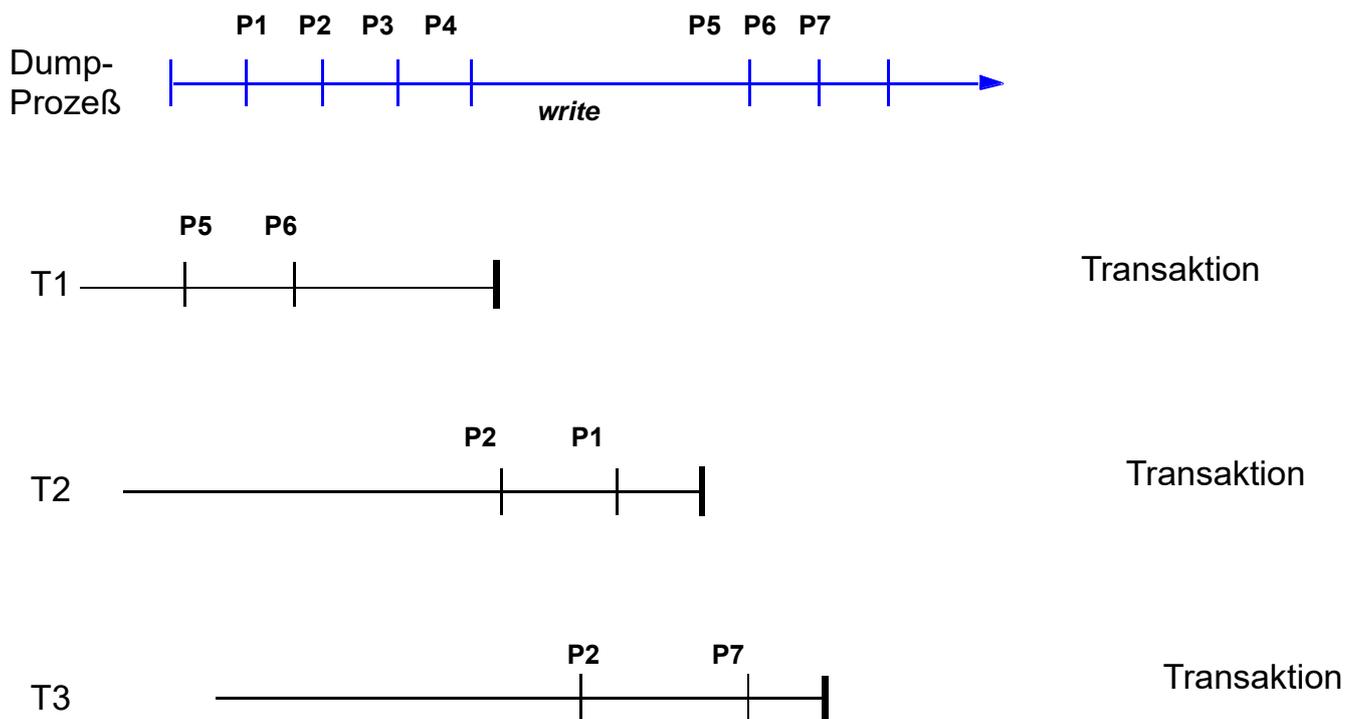
```

WHILE there are white pages DO;
    lock any white page; // short read lock
    IF page is modified THEN DO;
        write page to archive copy;
        clear modified bit;
    END;
    change page color;
    release page lock;
END;
    
```

- Transaktionen, die sowohl weiße als auch schwarze Objekte geändert haben ('graue Transaktionen'), werden zurückgesetzt ('Farbtest' am Transaktionsende)



Black-/White-Verfahren: Beispiel

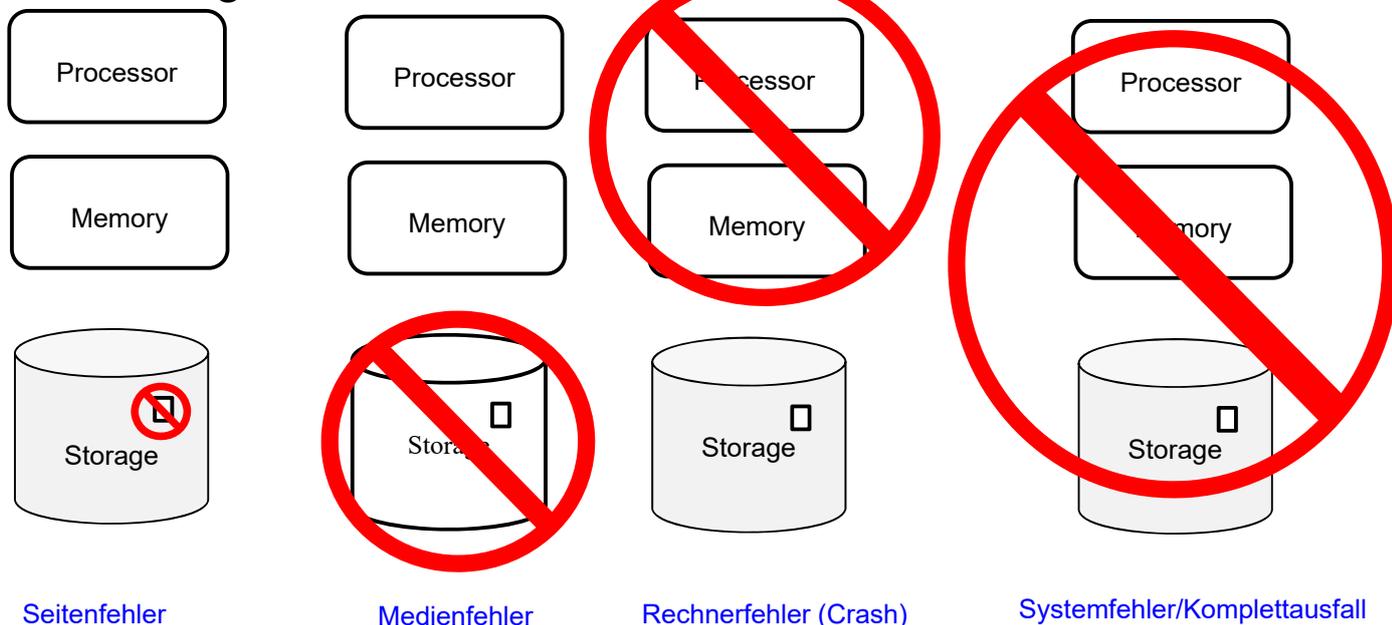


Black-White-Verfahren: Copy on Update

- Rücksetzung von Transaktionen zu umgehen
- Beste Lösung: Copy on Update (“save some”)
 - es werden nur schwarze Transaktionen zugelassen, d.h. Dump beinhaltet keine Änderung seit Dump-Beginn
 - während Dump-Prozess wird bei Änderung eines weißen Objektes Kopie mit Before-Image der Seite angelegt
 - Dump-Prozess greift auf Before-Images zu
 - Archivkopie entspricht DB-Schnappschuss bei Dump-Beginn
- Merkmale
 - transaktionskonsistenter Dump
 - keine Rücksetzungen / Blockierungen von Transaktionen
 - geringer Speicheraufwand für Before-Images
 - Ähnlichkeit zu Mehrversionen-Synchronisation mit Dump als langer Lese-Transaktion

Single Page Recovery*

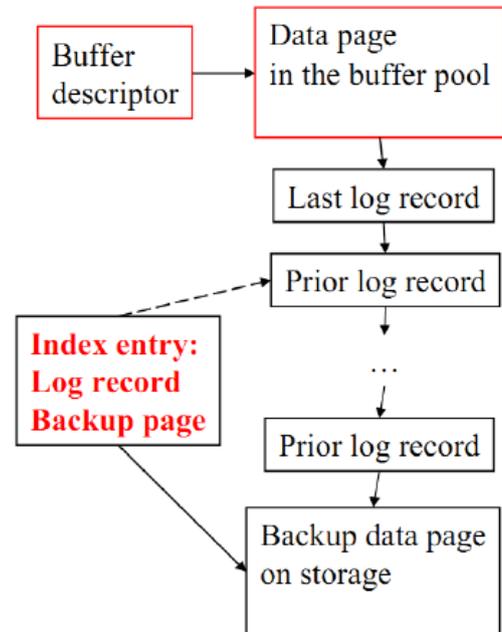
- häufiger Spezialfall der Medien-Recovery
 - nur eine Seite bzw. einzelne Seiten fehlerhaft
 - komplette Medien-Recovery dauert zu lange
 - dedizierte Unterstützung wünschenswert
- neue Fehlerkategorie?



*G. Graefe, H.A. Kuno: *Definition, Detection, and Recovery of Single-Page Failures, a Fourth Class of Database Failures*. PVLDB 5(7): (2012)

Single Page Recovery (2)

- Beschleunigung durch
 - Rückwärts-Verkettung von Log-Sätzen pro Seite
 - Einsatz eines sog. *Page Recovery Indexes* mit Verweis auf letzte Seitenkopie (in vollständiger oder inkrementeller Archivkopie) sowie den letzten Log-Eintrag der Seite auf Log-Datei
- Seiten-Recovery
 - Einspielen der letzten Seitenkopie
 - Durchgang aller Log-Sätze von letzten bis zu erstem seit der Kopieerstellung
 - Anwendung der Redo-Sätze in umgekehrter Reihenfolge



Zusammenfassung

- Crash-Recovery
 - Analyse-, Redo-, Undo-Lauf auf temporärer Log-Datei
 - Redo über Vergleich PageLSN mit LSN der Log-Sätze
 - Notwendigkeit von Compensation Log Records
 - vollständiges Redo (ARIES) unterstützt Satzsperrern
 - Idempotenz: Crashes während Recovery können behandelt werden
- Erstellung von Archivkopien der DB (Dumping)
 - inkrementelle vs. vollständige Dumps
 - "Fuzzy Dumps" vs. aktions/transaktionskonsistente Dumps
 - transaktionskonsistente Archivkopie durch "Copy on Update"
- schnelle Rekonstruktion einzelner Seiten wünschenswert