Vorlesung Implementierung von Datenbanksystemen

6. Puffer

Prof. Dr. Klaus Meyer-Wegener

Wintersemester 2019/20

Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

6. Pufferverwaltung

- **Neben sequenziellem Zugriff nun auch direkter Zugriff auf Sätze**
  - Über Satzadresse oder Schlüssel
- **Im Hauptspeicher Platz für *n* Blöcke (*n* > 1)**
  - Hängt ab von Größe des Hauptspeichers, Zahl der Prozesse usw.
  - Größenordnung heute 10^4 – 10^6
- **(Buffer-) Frame**
  - Für Aufnahme eines Blocks vorgesehener Abschnitt des (virtuellen) Hauptspeichers
  - Deutsch: Pufferrahmen oder Kachel

6.1 Blockzugriffe

- **Zugriff auf Block *i***
  - Sog. **logischer Zugriff** (manchmal auch: "logische Referenz" des Blocks)
- **Zwei Fälle:**
  - Block bereits im Puffer
  - Block muss von Platte eingelesen werden (**physischer Zugriff**)
    - Auch **Einlagern** eines Blocks (in den Puffer) genannt
      - Eine E/A-Operation
- **Einlagern verdrängt einen anderen Block aus dem Puffer**
  - War geändert worden: muss auf Platte zurückgeschrieben werden
    - Noch eine E/A-Operation
  - War nicht geändert worden: kann direkt überschrieben werden

6.2 Ersetzungsverfahren

Welcher der *n* Blöcke im Puffer wird verdrängt?

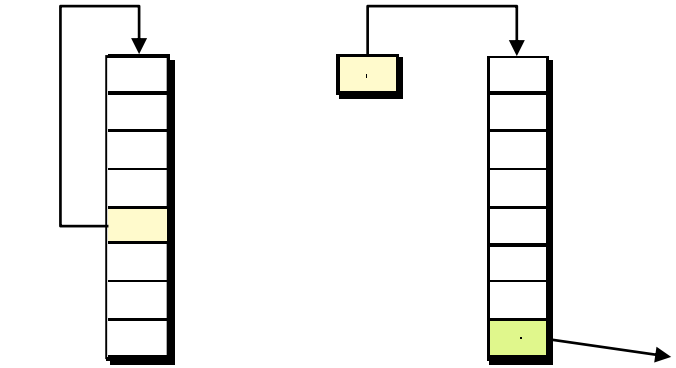
- **Ziel:**
  - Minimierung der Zahl der physischen Zugriffe
    bei gegebener Zahl von logischen Zugriffen
- **Deshalb:**
  - Häufig benutzte Blöcke (z.B. Wurzel B-Baum) im Puffer halten
- **Ersetzungsstrategie**
  - wählt den zu verdrängenden Block aus
- **Kriterien:**
  - "**Alter**" eines Blocks
    - Zahl der logischen Zugriffe (auf beliebige Blöcke) seit dem Einlagern
      - D.h. der logische Zugriff ist die Zeiteinheit
  - **Benutzungshäufigkeit** eines Blocks
    - Zahl der logischen Zugriffe auf diesen Block

Ersetzungsverfahren (2)

- **First in, first out (FIFO)**
  - **bewertet nur das Alter**:
    - Der Block, der am längsten im Puffer ist, wird ersetzt.
  - Ungünstig beim Direktzugriff:
    - Häufig benutzte Blöcke sollen ja gerade im Puffer bleiben – und dort "alt" werden.
- **Least frequently used (LFU)**
  - **bewertet nur die Häufigkeit**:
    - Der Block, auf den am seltensten zugegriffen wurde, wird ersetzt.
  - Sequenzielles Lesen heißt: Auf jeden Block wird genau einmal zugegriffen.
    - Kriterium nicht anwendbar!
  - Direktzugriff: Häufig genutzte Blöcke werden nun gehalten.
    - Aber: hat ein Block einmal eine hohe Häufigkeit auf sich versammelt, bleibt die, auch wenn inzwischen seit Minuten (Stunden?) nicht mehr auf ihn zugegriffen wurde!

Ersetzungsverfahren (3)

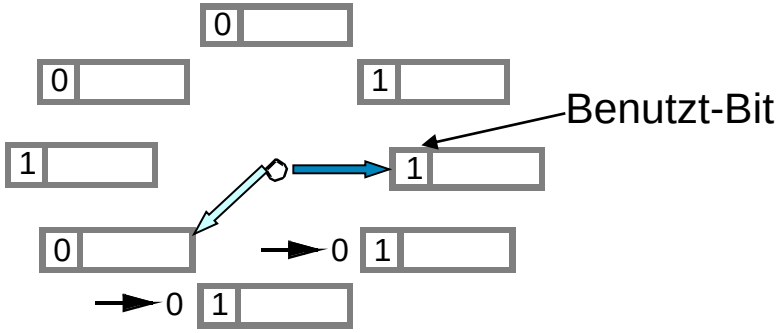
- **Least recently used (LRU)**
  - **bewertet das Alter seit dem letzten Zugriff**, nicht seit dem Einlagern
    - und damit indirekt auch die Häufigkeit
  - Wie? Quasi eine **verkettete Liste** aller Blöcke im Puffer:
    - Beim Zugriff auf einen Block im Puffer wird dieser Block ausgekettet und als erster Block wieder eingehängt.
    - Bei Verdrängung wird der letzte Block der Kette ersetzt (also der, auf den am längsten nicht zugegriffen wurde).
    - Ein eingelagerter Block kommt an den Anfang der Kette.
  - Veranschaulichung (Stapel):



  - Sehr gut, aber aufwändig!

Ersetzungsverfahren (4)

- **CLOCK (Second Chance)**
  - Idee: LRU-Verhalten mit einfacher Implementierung erreichen
- **Prinzip:**
  - **Benutzt-Bit** eines Blocks im Puffer wird bei Zugriff auf 1 gesetzt.
  - Bei Verdrängung zyklische Suche mit dem **Auswahlzeiger**:
    - Falls Benutzt-Bit = 1, wird es auf 0 gesetzt.
    - Falls Benutzt-Bit = 0, wird Block ersetzt.
    - Zeiger wandert zum nächsten Block.



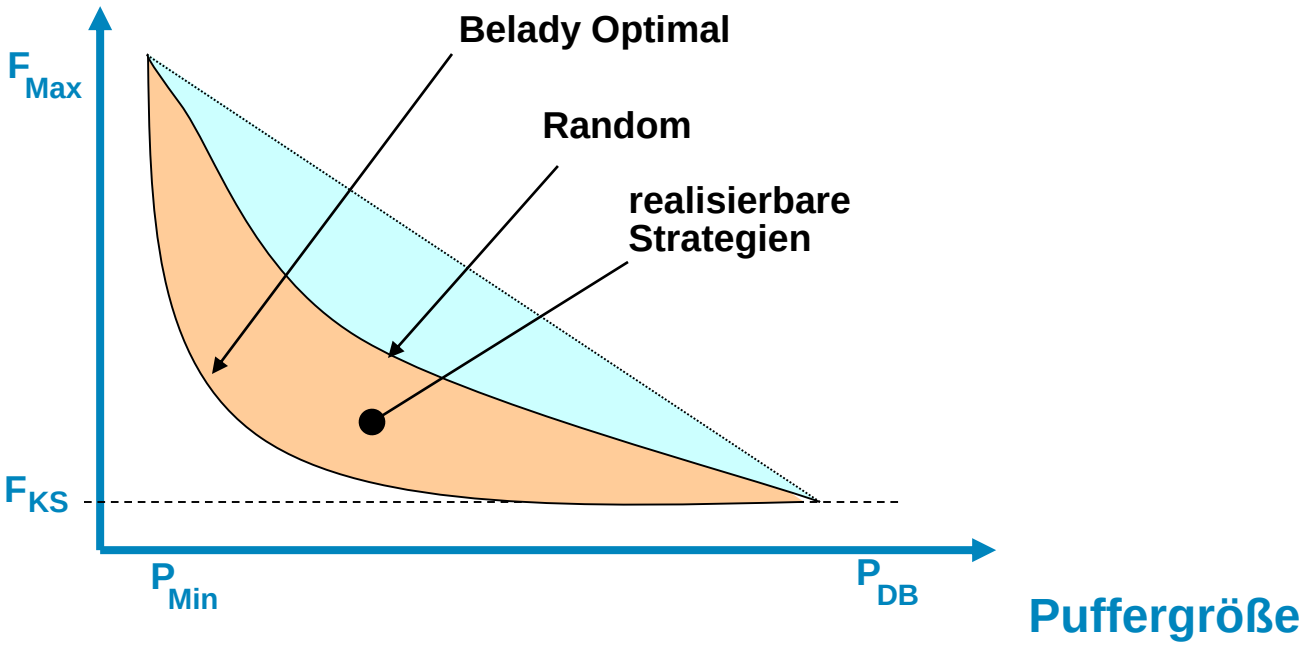
- **Charakterisierung:**
  - Jeder Block "überlebt" mindestens zwei Zeigerumläufe.

Ersetzungsverfahren (5)

- **Weitere Verfahren:**
  - G-CLOCK, LRD usw.
  - Versuchen ebenfalls, LRU-Qualität mit weniger Aufwand zu erreichen
  - Noch mehr Kriterien als nur ein Benutzt-Bit

- **Wichtige Restriktion:**
  - Blöcke, die noch benutzt (gelesen, geändert) werden, sind nicht ersetzbar.
    - Unterschied zum Paging in Betriebssystemen:
      - DBVS kennt die einzelnen Zugriffe auf einen Block im Puffer nicht.

Ersetzungsverfahren (6)



P_{min} = minimale Größe des Puffers (= 1)
P_{DB} = Datenbankgröße
F_{KS} = physische Blockzugriffe bei Kaltstart
(Jeder angesprochene Block muss zumindest einmal eingelesen werden.)

6.3 Schnittstelle einer Pufferverwaltung

- **Entscheidung für bestimmte Ersetzungsstrategie und Einbringstrategie**
  - vor Benutzern (höheren Software-Schichten) verbergen!
  - Dadurch ggf. änderbar
- **Einkapselung der Pufferverwaltung**
  - Zugänglich nur über folgende Operationen:

```
char *Buffer::fix ( BlockFile File, int BlockNo, char Mode );
```

    - Logischer Zugriff: stellt Block im Puffer zur Verfügung und liefert Anfangsadresse zurück
    - **Mode** gibt an, ob Block nur gelesen oder auch geändert werden soll
    - Name **fix** macht deutlich, dass Block vor Verdrängung geschützt ist, bis Bearbeitung abgeschlossen (einzelne Zugriffe auf Block im Puffer für Pufferverwaltung nicht wahrnehmbar)

```
void Buffer::unfix ( char *BufferAddress )
```

    - gibt Block im Puffer zur Ersetzung frei

Schnittstelle einer Pufferverwaltung (2)

- **Änderungen in der Satzverwaltung:**

  (in allen Implementierungen von Zugriffsoperationen
  für sequenzielle, direkte und Schlüsselzugriffs-Dateien):

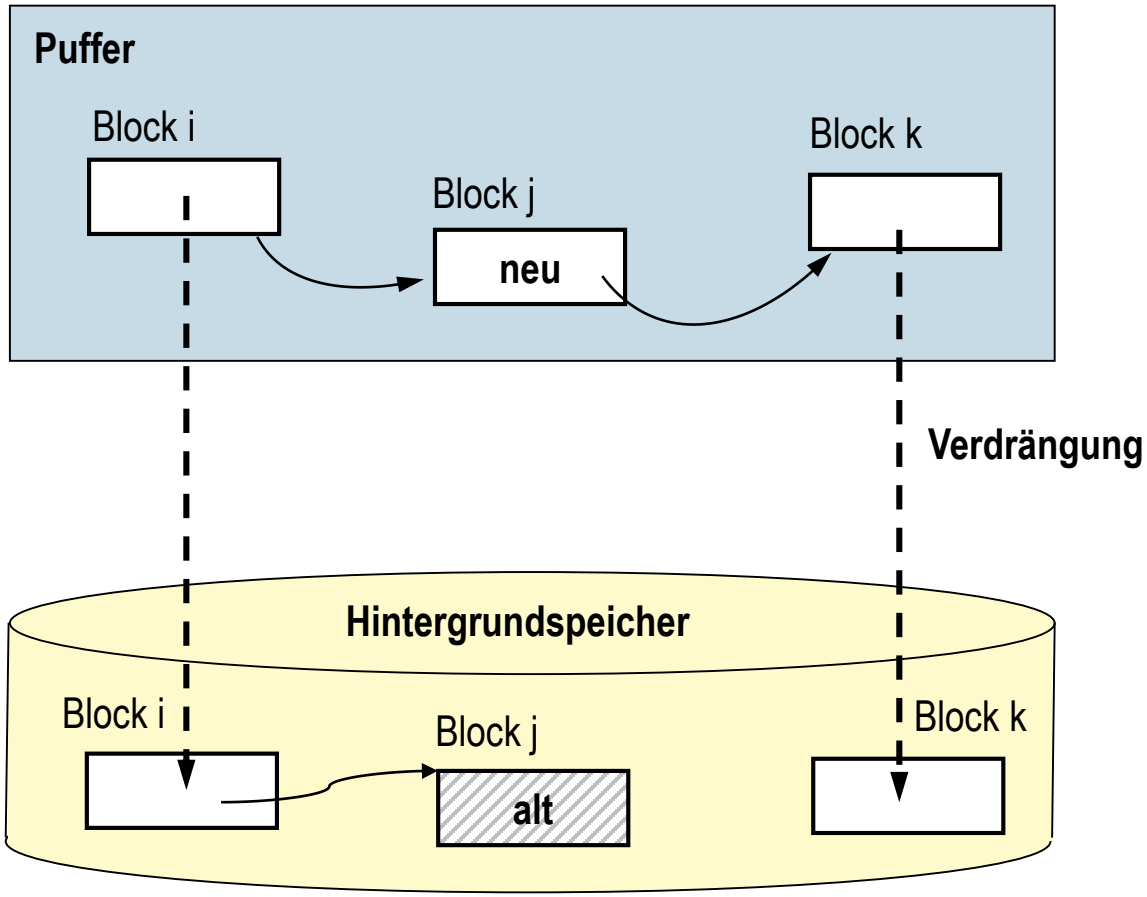
  - **read**(-Block)-Aufrufe
    - durch **fix**-Aufrufe ersetzen
  - **unfix-**Aufrufe
    - an geeigneter Stelle einfügen (wenn man mit der Bearbeitung eines Blocks fertig ist, z.B. vor dem Lesen des nächsten Blocks)
  - **write**(-Block)-Aufrufe
    - weglassen oder durch **unfix**-Aufrufe ersetzen
    - Block muss beim **fix** mit **Mode** = "w" geladen worden sein, wird dann von Pufferverwaltung bei Verdrängung geschrieben.
  - Auch ein leerer Block muss mit **fix** im Puffer bereitgestellt werden.
    - Pufferverwaltung liest ihn natürlich nicht von Platte ein, reserviert aber Kachel und führt ggf. auch Verdrängung durch.

6.4 Fehlersituationen

- **Betriebssystem-Absturz, Hardware-Fehler, Stromausfall:**
  - Hauptspeicher-Inhalt verloren – einschl. Puffer!
  - Ersetzungsstrategie hat entschieden,
    was auf der Platte steht (und damit vorhanden ist) und
    was noch im Puffer war (und damit verschwunden ist).
  - Blöcke auf der Platte wahrscheinlich inkonsistent!
    (alt und neu passen nicht zusammen)
- **Beispiel: B-Baum nach Splitt**
  - Neuer Knoten schon auf Platte geschrieben,
    aber übergeordneter Knoten dort noch der alte.
  - Neuer Knoten wird nicht beachtet!
    Seine Hälfte der Sätze fehlt plötzlich.
- **Muss von Hand mit hohem Aufwand bereinigt werden.**
- **Pufferverwaltung sollte Unterstützung bieten**

Fehlersituationen (2)

- **Veranschaulichung:**



Vorbeugemaßnahmen

- **Einbringstrategie:**
  - Bei Verdrängung
    alte Blockinhalte auf der Platte nicht überschreiben,
    sondern neue Blockinhalte in andere Slots schreiben.
  - Erst am Schluss
    (wenn alle neuen Blockinhalte auf der Platte sind)
    auf einen Schlag (ununterbrechbar) umschalten von alt auf neu.
    - Slots mit alten Blockinhalten dann freigeben und beim nächsten Mal selbst wieder für neue Blockinhalte verwenden.
- **Bei Fehler dann:**
  - Neue Blöcke ignorieren,
    dadurch alten (konsistenten) Zustand wiederherstellen und
    Programm nochmal laufen lassen.

6.5 Einbringstrategien

- **Einbringstrategien für Änderungen**
  - **Einbringen** = Ablegen auf einem nicht-flüchtigen Speicher derart, dass es nach einem Ausfall verwendet werden kann
  - Grundlegende Idee:
    - Speichern (beim Verdrängen aus dem Puffer) muss nicht notwendigerweise sofort ein Einbringen in den Datenbestand sein !
    - Einbringen kann auch verzögert stattfinden.
      - Nämlich erst dann, wenn auch noch andere, dazu gehörende Blöcke gespeichert worden sind.
- **Begriffliche Trennung in Block und Seite**
  - Seite = Block im Puffer (d.h. genau so groß)
  - Anwender (höhere Software-Schicht) arbeitet nur noch mit Seiten.
  - Flexible Zuordnung zu Blöcken
  - Insbesondere: *mehrere* Blöcke für eine Seite
    - Z.B. eben alter und neuer Inhalt derselben Seite

Segment

- **Linearer, logischer, potenziell unendlicher, tatsächlich aber endlicher Adressraum mit sichtbaren Seitengrenzen**
- **Entspricht der Datei**
  - Folge von Seiten
  - Einheit des Sperrens, der Wiederherstellung und der Zugriffskontrolle
  - Zuordnung zu Dateien: systemabhängig

    1:1 – alle Seiten eines Segments (und nur sie) in den Blöcken einer Datei gespeichert

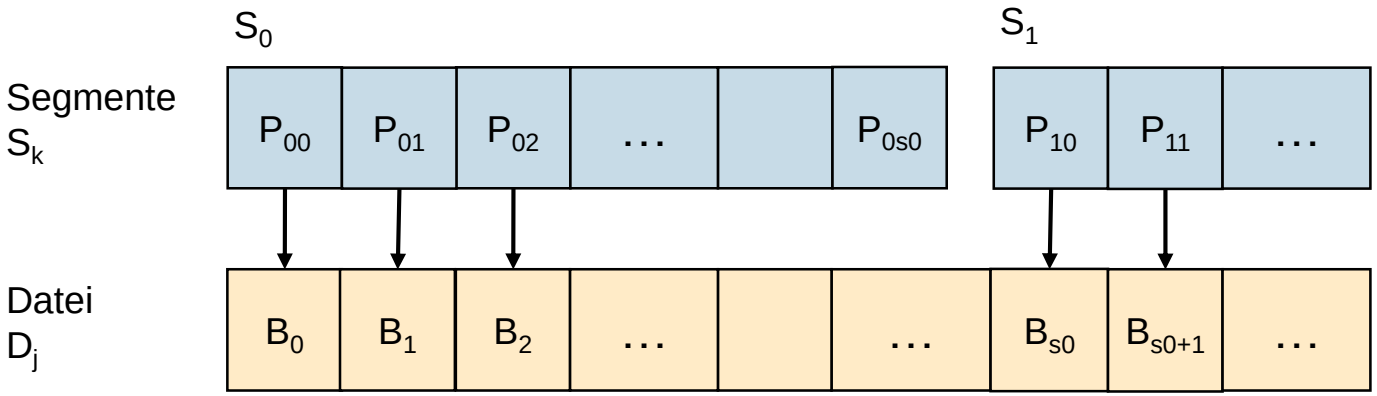
    N:1 – Seiten mehrerer Segmente zusammen in den Blöcken einer Datei gespeichert

    1:N – Seiten eines Segments in Blöcken unterschiedlicher Dateien gespeichert
- **Segmenttypen**
  - Selektive Einführung zusätzlicher Attribute
    - Katalog, Schema-Informationen, Log, alle gemeinsam benutzbaren Daten
    - Daten, die für bestimmte Benutzer oder Benutzergruppen reserviert sind
    - Private Kopien von Daten für einzelne Benutzer (Snapshots)
    - Hilfsdateien für Benutzerprogramme
    - Temporärer Speicher, z.B. für Sortierprogramme

Seitenzuordnung

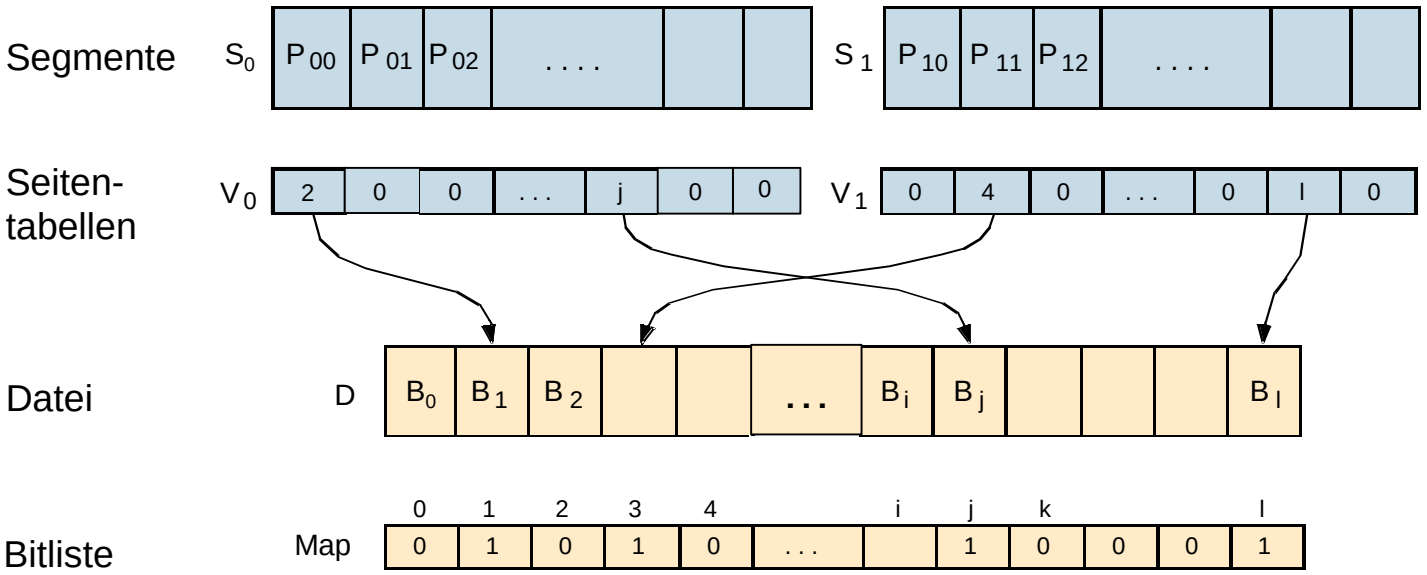
- **Aufgabe:**
  - Welche Blöcke für eine Seite?
- **Möglichkeiten:**
  - **Direkte** Seitenzuordnung
    - Aufeinander folgende Seiten werden auf aufeinander folgende Blöcke einer Datei abgebildet
    - Keine Hilfsstruktur benötigt, nur erste Blocknummer merken
  - **Indirekte** Seitenzuordnung
    - Mehr Flexibilität bei der Zuordnung zu Blöcken
    - Erfordert Hilfsstruktur (Array) mit Blocknummer zu jeder Seite

Direkte Seitenzuordnung



- **Eigenschaften**
  - Keine Fragmentierungsprobleme wegen der geforderten Übereinstimmung von Seiten- und Blockgröße ($L_k == L_j$)
  - Jeder Seite $P_{ki} \in S_k$ genau ein Block $B_{jl} \in D_j$ zugeordnet
  - Funktioniert natürlich nur für N:1- oder 1:1-Verhältnis von Segment (Datenbanksystem) und Datei (Betriebssystem)

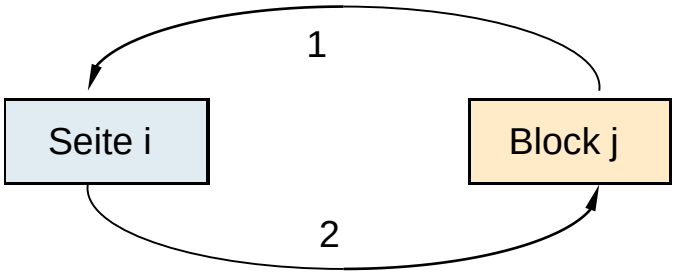
Indirekte Seitenzuordnung



- Für jedes Segment S_k existiert eine **Seitentabelle** V_k, die für jede Seite einen Eintrag (4 Bytes) mit der aktuellen Blockzuordnung enthält.

- Für die Datei D existiert eine **Bitliste** "Map", die ihre aktuelle Belegung beschreibt, d.h. die für jeden Block angibt, ob er momentan eine Seite enthält oder nicht (Freispeicher-verwaltung).

Direkte Seiteneinbringung



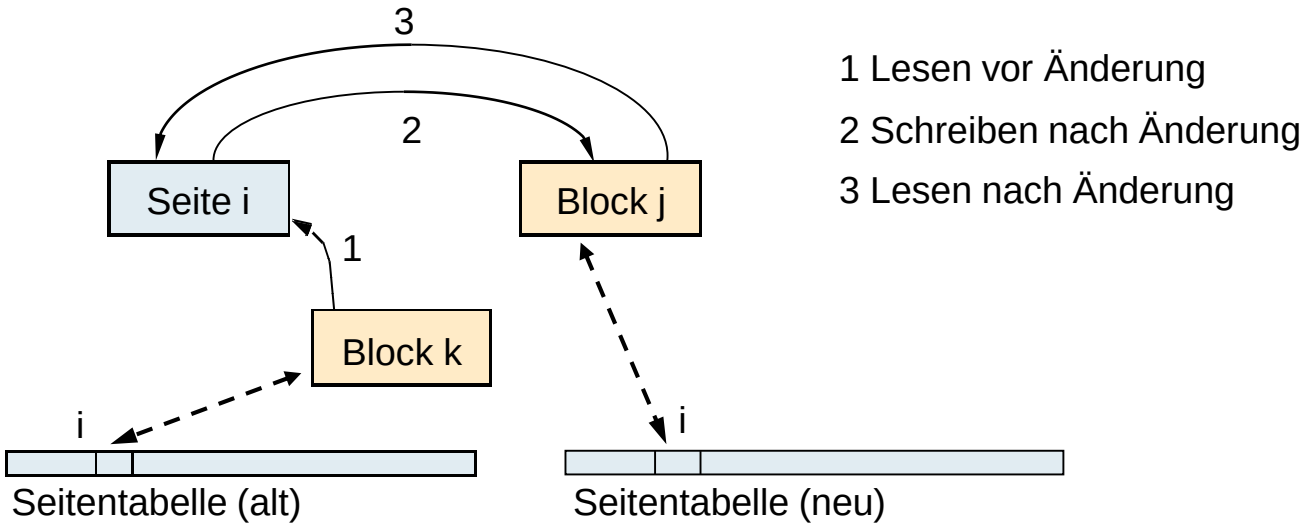
1 Lesen vor Änderung

2 Schreiben nach Änderung

- Beim Verdrängen aus dem Puffer *ersetzt* eine Seite genau den Block, aus dem sie beim Einlagern in den Puffer gelesen wurde ("**update in place**").
- **Vorteil:** Einfachheit; immer nur ein Block pro Seite
- **Nachteil:** Keine Unterstützung der Wiederherstellung nach Ausfällen
  - Der alte Zustand der Seite muss als Protokollinformation *vor* (!) dem Einbringen der geänderten Seite (und damit dem Vernichten des alten Seiteninhalts) auf einen sicheren Speicher geschrieben werden ("**W**rite-**A**head **L**og", **WAL**-Prinzip, s. unten).

Indirekte Seiteneinbringung

- Beim Verdrängen aus dem Puffer wird eine Seite *in einen freien Block* geschrieben. Der ursprüngliche Block bleibt unverändert.
- Auch nach einem Hauptspeicherverlust ist eine konsistente Datenbank in den alten Blöcken verfügbar!



Indirekte Seiteneinbringung (2)

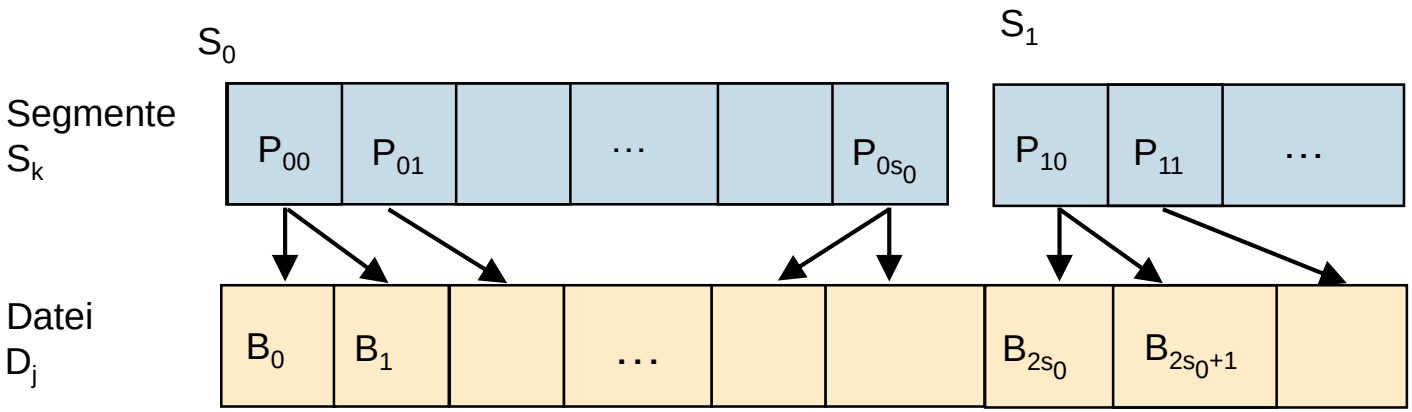
- **Noch zu klären: Wann und wie wird man die alten Blöcke los?**
  - Man muss wissen, wann alle zusammengehörenden Blöcke gespeichert wurden.
    - Beim Herunterfahren – falls es das gibt
    - Wenn gerade mal kein Programm läuft – falls es das gibt
    - Sicherungspunkt

- **Techniken:**
  - Schattenspeicher
  - Twin Slots
  - (Zusatzdatei)

Schattenspeicher – Prinzip

- **Benötigt indirekte Seitenzuordnung**
- **Inhalte aller Seiten eines Segments werden in einem Sicherungsintervall Δt in einem konsistenten Zustand unverändert gehalten.**
  - Sicherungspunkte (Endpunkte der Sicherungsintervalle) sind segmentorientiert,
    - wirken für alle Nutzer des Segments gemeinsam.
  - Sicherungspunkt besteht aus: belegten Seiten, Seitentabelle V_k, Bitliste M_j auf stabilem Speicher
  - Im Fehlerfall: segmentorientiertes Zurückgehen auf letzten Sicherungspunkt
- **In einem Segment S_k mit den Seiten P_{ki} ($0 \leq i \leq s_k - 1$) sind $h_k \leq s_k$ Seiten belegt.**
  - Speicherbedarf z: $h_k \leq z \leq 2\, h_k$
- **Details im Anhang**

Twin Slots – Prinzip

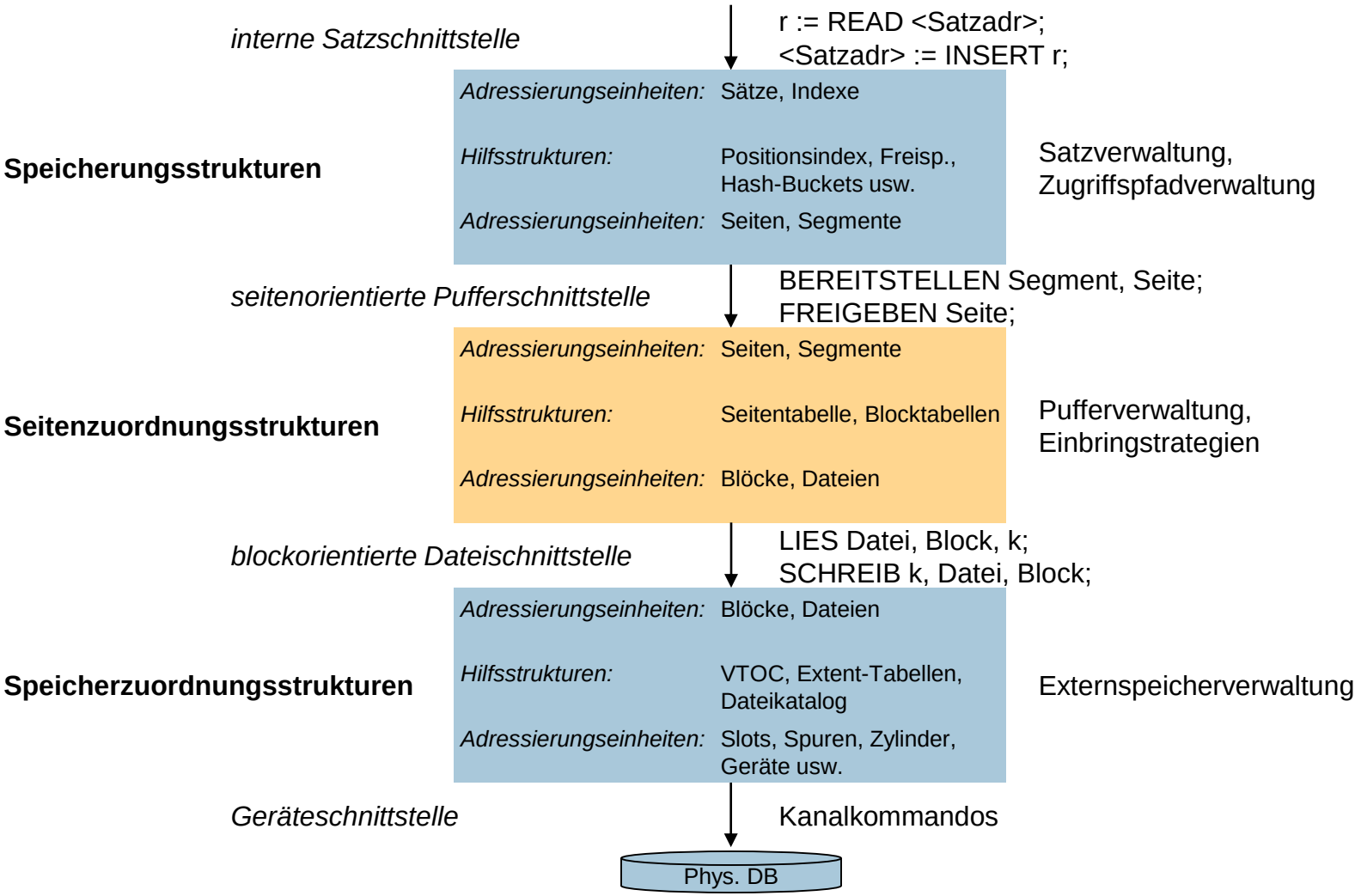


- **Eigenschaften**
  - Direkte Seitenzuordnung
  - Doppelter Speicherplatzbedarf
  - Seite muss eine Art Versionsnummer enthalten, damit man weiß, welche die aktuellere ist.
  - Immer beide Blöcke lesen!
  - Dann bei Änderungen den älteren überschreiben

6.6 Zusammenfassung

- **Zweistufige Abbildung**
  - von Segment/Seite auf Datei/Block
    erlaubt Einführung von Abbildungsredundanz
    durch verzögertes Einbringen
- **Verzögertes (indirektes) Einbringen**
  - Teurer als direktes, besitzt jedoch implizite Fehlertoleranz
  - Geringe Kosten für Protokollieren und Wiederherstellung
  - Belastet den Normalbetrieb zugunsten der Wiederherstellung
    (Verwaltung der Seitentabellen)
  - Schwierige Implementierung
- **Direktes Einbringen ("update in place")**
  - Einfach zu implementieren
  - Keine zusätzlichen Kosten zur Ausführungszeit für die Seitenzuordnung
  - Fehlertoleranz nur durch explizite Protokollierung und Wiederherstellung
- **Verallgemeinerung: Transaktionskonzept (s. unten)**

Resultierende Schicht



Anhang

- **Lokalität**
- **Schattenspeicher im Detail**

Lokalität

- **Puffer kann nur wirken, wenn Lokalität bei den Seitenzugriffen (logischen Referenzen) vorliegt.**
  - Z.B. 80-20-Regel: 80% der Zugriffe entfallen auf 20% der Seiten.
- **Ungleichverteilung (Schiefe, Schräge, Skew)**
- **Kann man erwarten, weil einige Seiten (z.B. Wurzel eines B-Baums) häufiger referenziert werden müssten.**

- **Kann man es für einen gegebenen Seitenreferenz-String auch prüfen?**

Bestimmung der Lokalität

- **Working-Set-Strategie**
  - **Working-Set** ***W*(*t*, *n*)** := Menge der verschiedenen Seiten, die von einer Transaktion innerhalb der ***n*** letzten Referenzen, vom Zeitpunkt ***t*** aus rückwärts gerechnet, angesprochen wurden.
    - ***n*** = Fenstergröße
  - Working-Set-Größe ***w*(*t*, *n*)** := |*W*(*t*, *n*)|
- **Beispiel:**

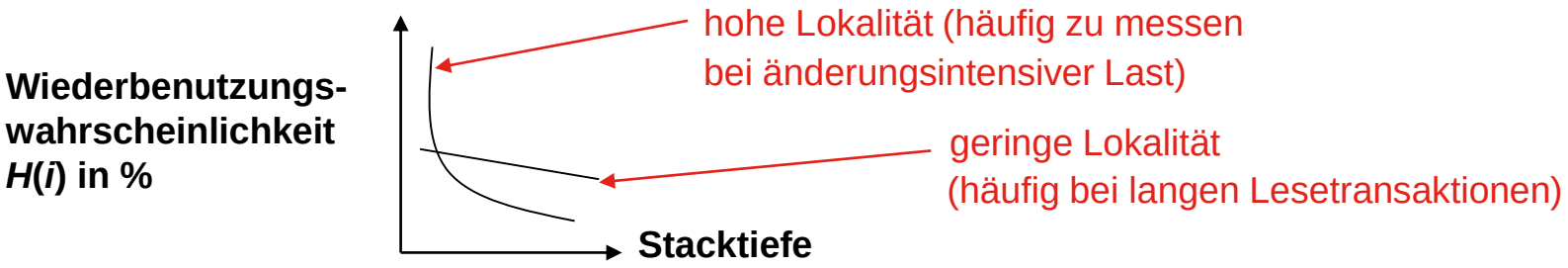
TA1:	A	A		C			A	A			A	G	H	
TA2:			B		D	E			E	F				F
								t1		t2			t3	

TA1:	*W*(t1, 5) = {A, C}	*w*(t1, 5) = 2
TA2:	*W*(t2, 5) = {B, D, E, F}	*w*(t2, 5) = 4
TA1:	*W*(t3, 5) = {A, G, H}	*w*(t3, 5) = 3

- **Mittlere Working-Set-Größe *w*(*n*)**
- **Mittlere Working-Set-Lokalität *L*(*n*) = 1 – *w*(*n*) / *n* mit *n* > 1**

Bestimmung der Lokalität (2)

- **LRU-Stacktiefenverteilung**
  - LRU-Stack enthält alle im Intervall [1, *t*] angesprochenen Seiten einer Transaktion in der Reihenfolge ihres Alters
    - D.h. die zuletzt angesprochene ganz oben, darunter die unmittelbar vorher angesprochene usw.
  - Ermittlung der LRU-Stacktiefenverteilung (Wiederbenutzungshäufigkeit):
    - Für jede Position des LRU-Stacks wird ein **Zähler** geführt.
    - Wird die angesprochene Seite in der Stacktiefe *i* gefunden, so wird der *i*-te Zähler um 1 erhöht.
  - Nach Bearbeitung des Referenz-Strings gibt der *i*-te Zähler die Häufigkeit *H(i)* an, mit der Seite in Stacktiefe *i* wiederbenutzt wird.



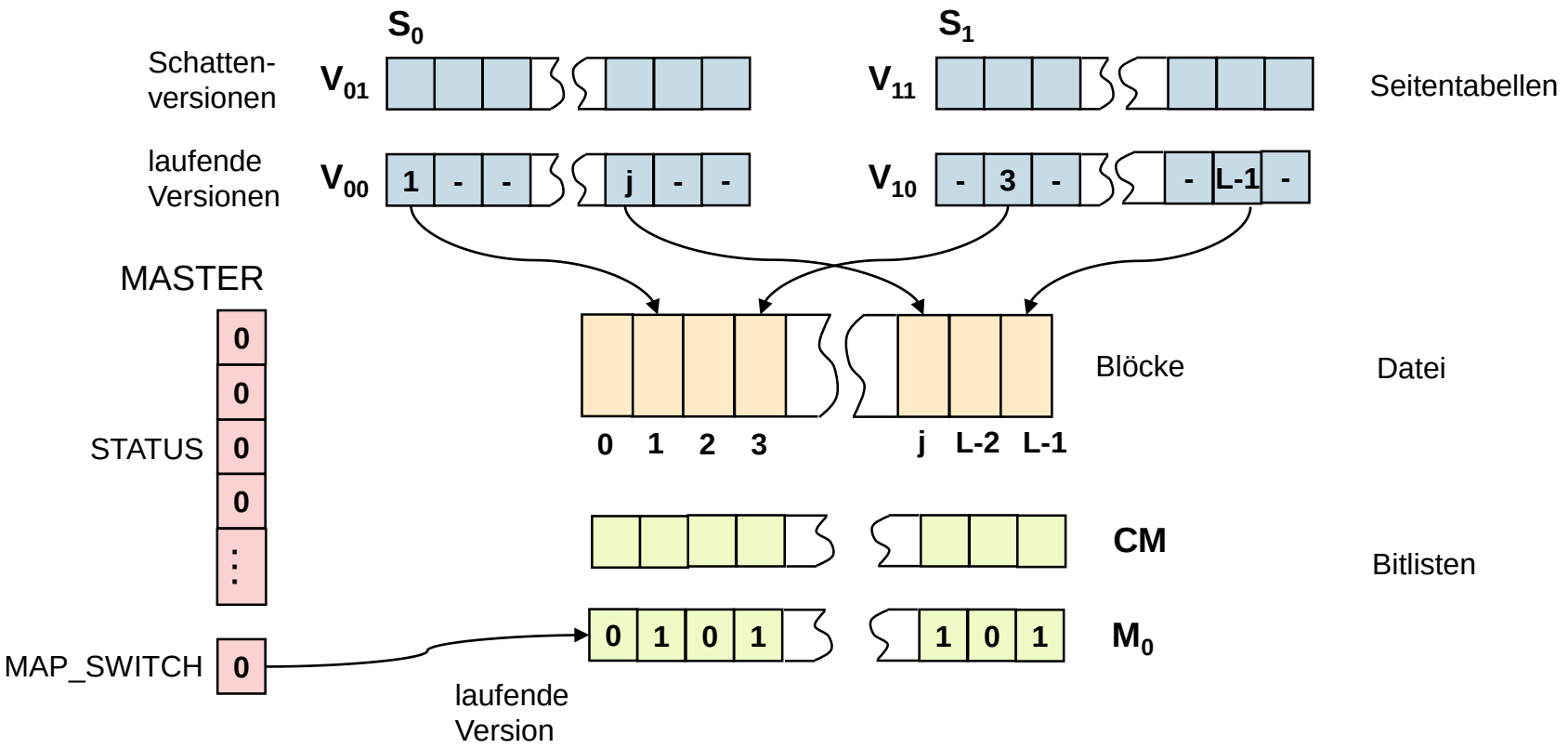
Bestimmung der Lokalität (3)

- **Page-Fault-Frequency-Strategie**
  - Basiert auf LRU-Stacktiefenverteilung
- **Verfahren:**
  - Wenn auch eine LRU-Ersetzungsstrategie benutzt wird, dann lässt sich aus *H(i)* die zu erwartende **Fehlseitenrate** *F* angeben.
  - Fehlseitenrate
    - *N* = Puffergröße bzw. Partitionsgröße
    - *D* = Zahl verschiedener Seiten im Referenzstring

$$F = \sum_{i=N+1}^{D} H(i)$$

Schattenspeicher – operational

- **Ausgangszustand: Segmente S_0 und S_1 geschlossen**



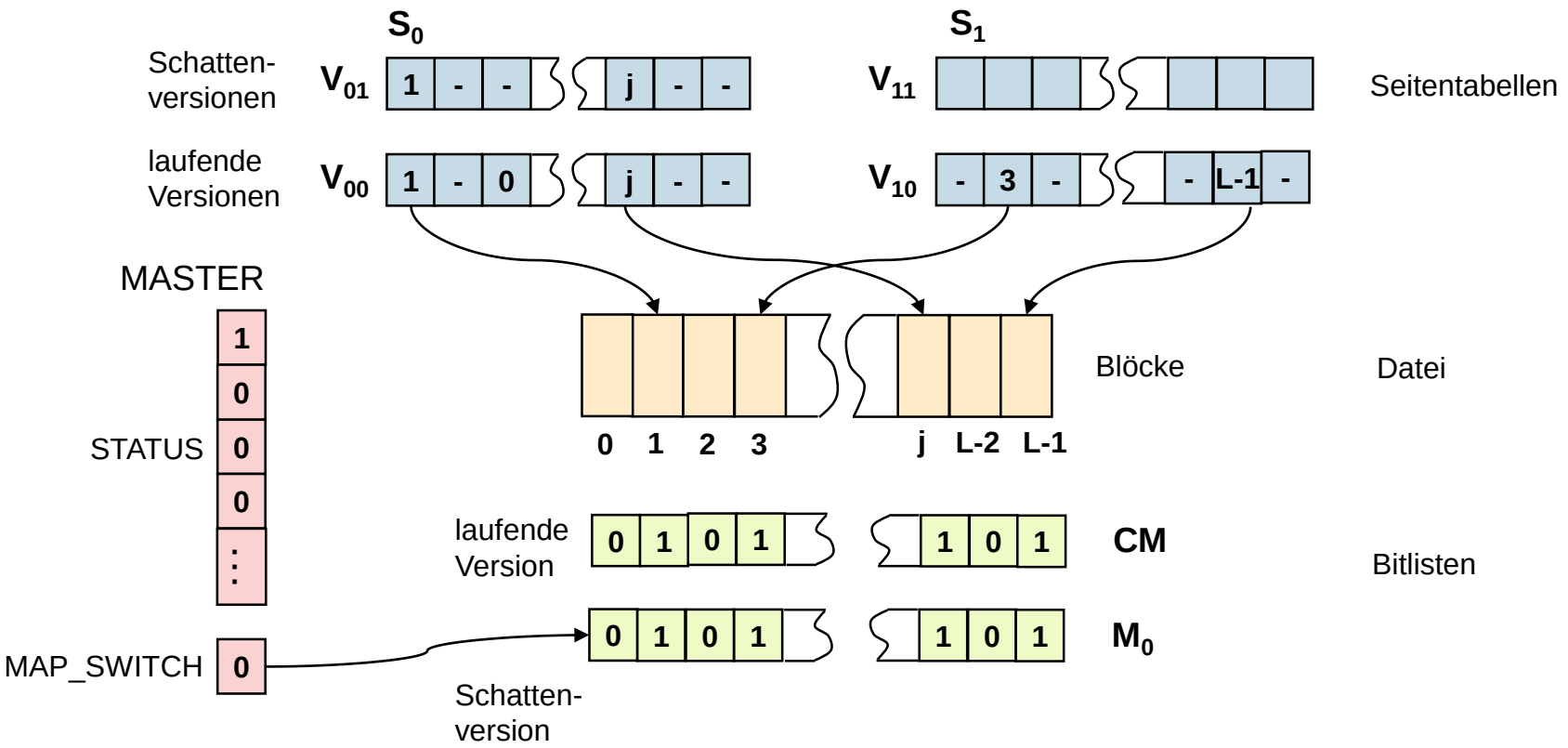
Schattenspeicher – operational (2)

- **Öffnen eines Segments S_k**
  - Kopieren von V_{k0} nach V_{k1} (auf der Platte)
  - STATUS[k] := 1
  - MASTER ununterbrechbar auf die Platte schreiben (z.B. doppelt)
  - Erzeugen von *CM* als Kopie von M_j ($j \in \{0, 1\}$ je nach MAP_SWITCH)

- **Nach Verlust des Hauptspeicherinhalts zeigt STATUS[k], dass V_{k1} gültig ist.**
  - V_{k0} dagegen i.Allg. unvollständig

Schattenspeicher – operational (3)

- **Segment S_0 für Änderungen geöffnet**

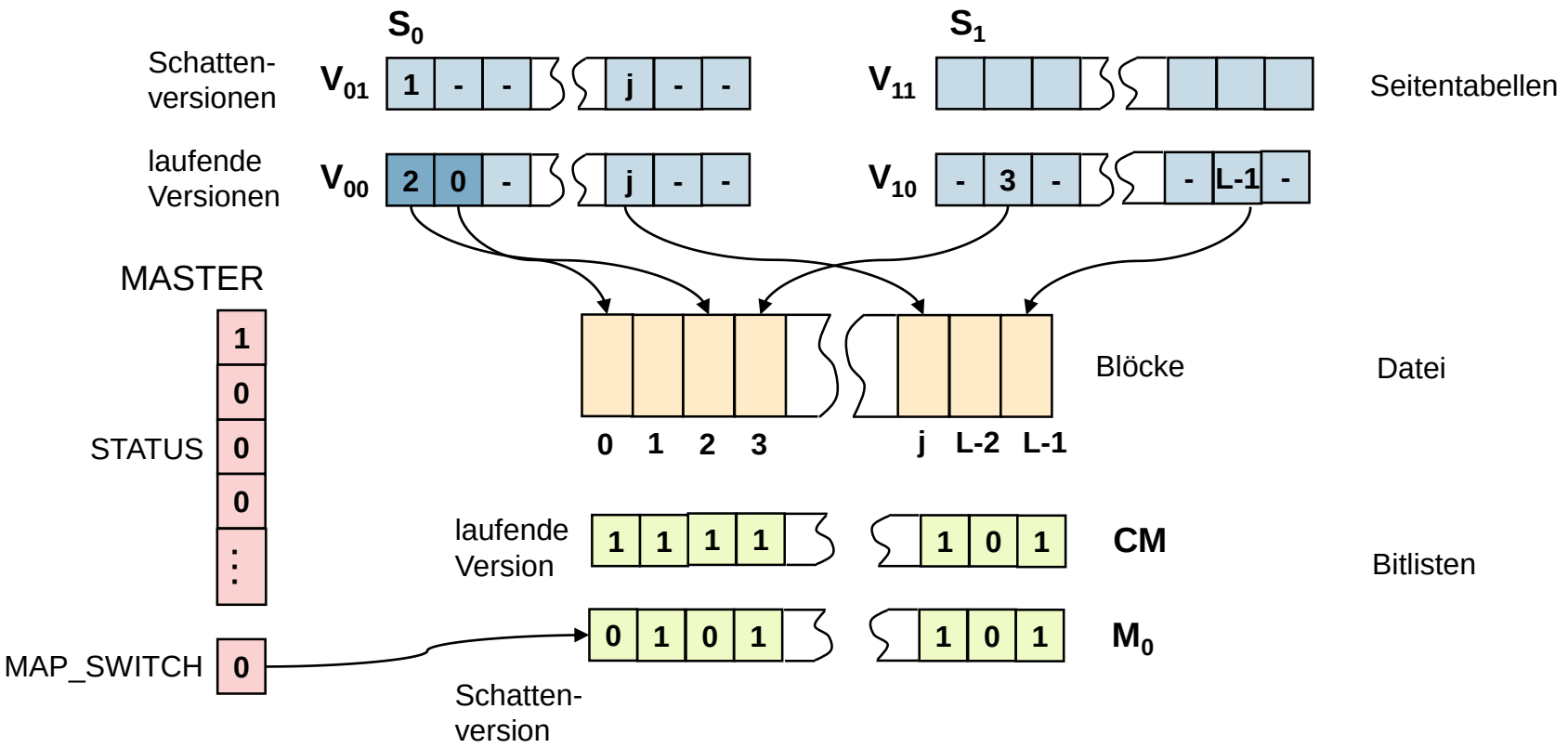


Schattenspeicher – operational (4)

- **Modifikation einer Seite:**
  Ausschreiben der Seite P_{ki} mit bisherigem Block $b = V_{k0}[i]$
  - Für P_{ki} muss ein neuer Block b' gefunden werden.
  - Suchen von b' mit $CM[b'] = 0$ (unbenutzter Block)
  - $CM[b'] := 1$
  - Setze $V_{k0}[i] : = b'$
  - P_{ki} nach b' schreiben
  - Schattenbit in $V_{k0}[i]$ setzen
- **Block b mit altem Zustand der Seite P_{ki} bleibt als "Schatten" über V_{k1} erreichbar.**
- **Bei weiteren Änderungen wird P_{ki} immer wieder in Block b' geschrieben.**

Schattenspeicher – operational (5)

- **Seiten P_{01} und P_{02} geändert:**



Schattenspeicher – operational (6)

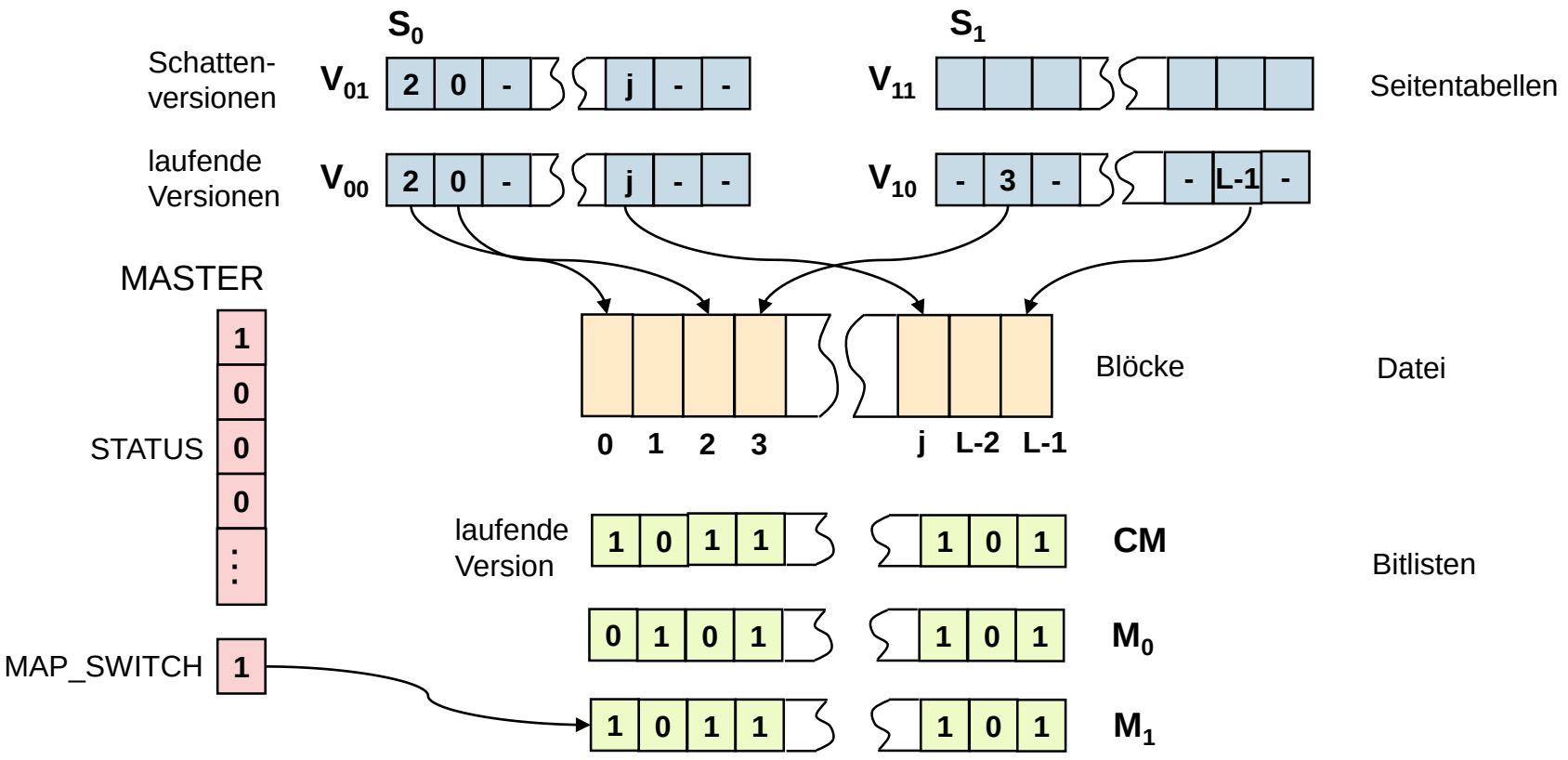
- **Sicherungspunkt für Segment S_k:**
  Nach Ablauf von Δt werden geänderte Seiten übernommen und Schattenseiten freigegeben.
  - M_1 als Kopie von M_0 erzeugen
    (oder umgekehrt, je nach MAP_SWITCH)
  - Für alle $V_{k0}[i] = b'$ mit gesetztem Schattenbit und $V_{k1}[i] = b$:
    - $M_1[b] := 0$; $M_1[b'] := 1$
    - V_{k0} und M_1 auf die Platte schreiben
    - STATUS[k] := 0
    - MAP_SWITCH := 1
    - MASTER ununterbrechbar auf die Platte schreiben (z.B. doppelt)
  - Segment S_k ist damit geschlossen.
- **Auf der Platte sind nun V_{k0} und M_1 als gültig verzeichnet.**

Schattenspeicher – operational (7)

- **Sicherungspunkt für Segment S_k (Forts.):**
  - Anschließend wird S_k gleich wieder geöffnet.
  - Für alle $V_{k0}[i] = b$' mit gesetztem Schattenbit und $V_{k1}[i] = b$:
    - $CM[b] := 0$
    - Schattenbit löschen
    - V_{k0} nach V_{k1} kopieren (auf der Platte)
    - STATUS[k] = 1
    - MASTER ununterbrechbar auf die Platte schreiben (z.B. doppelt)

Schattenspeicher – operational (8)

- **Sicherungspunkt für S_0:**



Schattenspeicher – operational (9)

- **Zurücksetzen auf letzten Sicherungspunkt (bei laufendem System):**
  - Für alle $V_{k0}[i] = b$ mit gesetztem Schattenbit:
    - $CM[b] := 0$
    - $V_{k0}[i] := V_{k1}[i]$
- **Reparatur einer Datei nach Systemausfall:**
  - Für alle Segmente S_k mit STATUS[k] = 0 (geschlossen):
    - keine Maßnahme erforderlich
  - Für alle Segmente S_k mit STATUS[k] = 1 (offen):
    - V_{k1} nach V_{k0} kopieren
      - Zurück zur alten, konsistenten Version
    - STATUS[k] := 0
  - MASTER ununterbrechbar auf die Platte schreiben

Schattenspeicher – Bewertung

- **Rücksetzen auf einen konsistenten Zustand ist billig.**
- **WAL (s. unten) kann vermieden werden, um flexibleres Schreiben der Protokolldatei zuzulassen.**
- **Cluster-Eigenschaften von Blöcken gehen verloren.**
- **Bei großen Datenbanken werden Hilfsstrukturen (Seitentabellen V_k, Bittabellen M_j) leider zu aufwändig.**
  - Sie müssen in Blöcke zerlegt und durch einen speziellen Ersetzungsalgorithmus in einem eigenen Puffer verwaltet werden.
- **Konzept geeignet für kleinere DB; direktes Einbringen besser für größere DB**
  - Also eigentlich für alle …