Implementierung von Datenbanksystemen

Kapitel 2: Externspeicherverwaltung

Kapitel 2: Externspeicherverwaltung

- Abbildung von Dateien und Blöcken
  - Zweistufige Speicherhierarchie
  - Allgemeine Aufgaben der Externspeicherverwaltung
- Dateisystem
  - Operationen, Adressierung und Abbildung
  - Verfahren der Blockzuordnung
    - statisch
    - dynamische Extent-Zuordnung
    - dynamische Block-Zuordnung
- Maßnahmen zur Fehlertoleranz
  - Lesen und Schreiben von Blöcken
  - Nutzung von Speicherredundanz
- Abbildung von Segmenten und Seiten
  - Segmentkonzept
  - verzögerte Einbringstrategien
    - Schattenspeicherkonzept (shadow page mechanism)
    - Zusatzdatei (differential file)
  - Bewertung der Seitenzuordnungsverfahren

Externspeicherverwaltung

Mengenorientierte DB-Schnittstelle
Sprachen wie
SQL, QBW, etc.

Adressierungseinheiten:	Relationen, Sichten, Tupel	Logische Datenstrukturen
Hilfsstrukturen:	externe Schemabeschreibung, Integritätsregeln	
Adressierungseinheiten:	externe Sätze, Sets, Schlüssel, Zugriffspfade	

Satzorientierte DB-Schnittstelle
FIND NEXT Satzname,
STORE Satzname,
etc.

Adressierungseinheiten:	externe Sätze, Sets, Schlüssel, Zugriffspfade	Logische Zugriffspfad-strukturen
Hilfsstrukturen:	Zugriffspfaddaten, interne Schemabeschreibung	
Adressierungseinheiten:	interne Sätze, B*-Bäume, Hash-Tabellen, usw.	

Interne Satzschnittstelle
Speichere Satz,
Füge Eintrag in B*-Baum ein,
etc.

Adressierungseinheiten:	interne Sätze, B*-Bäume, Hash-Tabellen, usw.	Speicherungs-strukturen
Hilfsstrukturen:	Freispeicher-Info., Adreßtab., Seitenindices, usw.	
Adressierungseinheiten:	Seiten, Segmente	

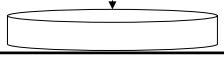
DB-Pufferschnittstelle
Stelle Seite j bereit,
Gib Seite j frei

Adressierungseinheiten:	Seiten, Segmente	Seitenzuordnungs-strukturen
Hilfsstrukturen:	Seitentabellen, Blocktabellen, etc.	
Adressierungseinheiten:	Blöcke, Dateien	

Dateischnittstelle
Lies Block k,
Schreibe Block k

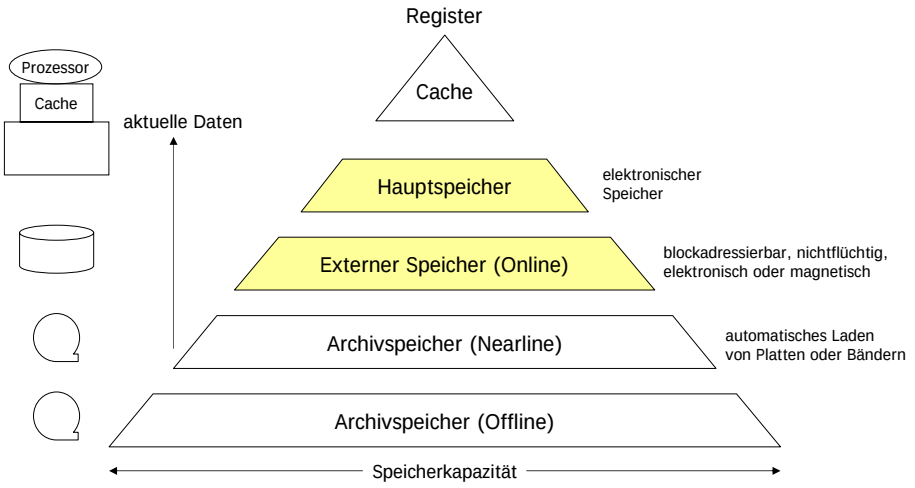
Adressierungseinheiten:	Blöcke, Dateien	Speicherzuordnungs-strukturen
Hilfsstrukturen:	Freispeicher-Info Extent-Tabellen, Dateikataloge, etc.	
Adressierungseinheiten:	Spuren, Zylinder, Kanäle, etc.	

Geräteschnittstelle



Externspeicherverwaltung

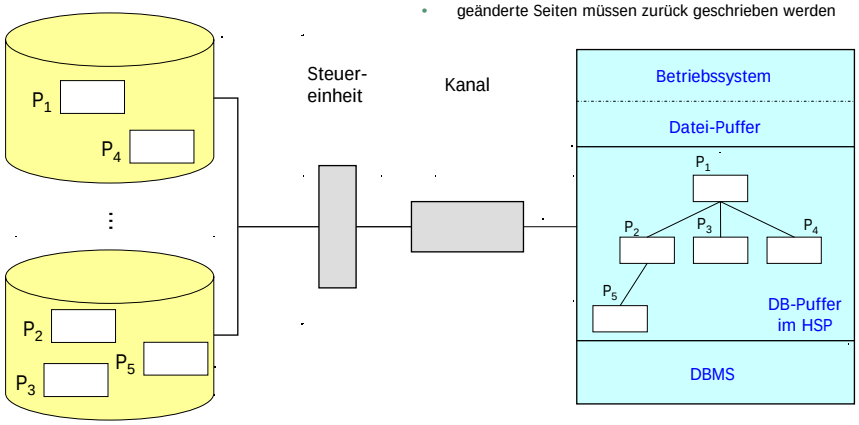
Speicherhierarchie



Modell einer zweistufigen Speicherhierarchie

- Vereinfachte E/A-Architektur:

- Modell für Externspeicheranbindung
- vor Bearbeitung sind die Seiten in den DB-Puffer zu kopieren
- geänderte Seiten müssen zurück geschrieben werden



Magnetplattenspeicher
< 10 ms

Zugriffslücke
> 10^5

Hauptspeicher
< 100 ns

Allgemeine Aufgaben der Externspeicherverwaltung

- Verwaltung externer Speichermedien
- Verbergen von Geräteeigenschaften
- Abbildung von physischen Blöcken auf externen Speicher
- Kontrolle des Datentransports von/zum Hauptspeicher
- ggf. Realisierung einer mehrstufigen Speicherhierarchie
- Fehlertoleranzmaßnahmen (stabiler Speicher, Spiegelplatten etc.)

Kapitel 2: Externspeicherverwaltung

- Abbildung von Dateien und Blöcken
  - Zweistufige Speicherhierarchie
  - Allgemeine Aufgaben der Externspeicherverwaltung
- Dateisystem
  - Operationen, Adressierung und Abbildung
  - Verfahren der Blockzuordnung
    - statisch
    - dynamische Extent-Zuordnung
    - dynamische Block-Zuordnung
- Maßnahmen zur Fehlertoleranz
  - Lesen und Schreiben von Blöcken
  - Nutzung von Speicherredundanz
- Abbildung von Segmenten und Seiten
  - Segmentkonzept
  - verzögerte Einbringstrategien
    - Schattenspeicherkonzept (shadow page mechanism)
    - Zusatzdatei (differential file)
  - Bewertung der Seitenzuordnungsverfahren

Speicherzuordnungsstrukturen

Abbildung von Dateien und Blöcken

- Beispiele für Aufrufe von Operation an der oberen Schnittstelle

```
PAM  UPAMDAT, RD, FECB=ESTBLOCK, HP=1
...
UPAMDAT  FCB FCBTYPE=PAM, LINK=EIN, IOAREA1=EINBER
```

- Abbildungsfunktion: Blocknummer → cyl-#, track-#, rec-#
- Durchführung der E/A:

Armpositionierung:	EXCP → CCW	} I/O-Prozessor
Spurauswahl:	EXCP → CCW	
Satzübertragung:	EXCP → CCW	

- Eigenschaften der oberen Schnittstelle
  - Menge durchnummerierter Blöcke innerhalb von Dateien
  - Lese- und Schreibzugriff über die Blocknummer
  - Blockzugriff kann auf Lesefehler führen, Blöcke können alte oder ungültige Daten enthalten

Dateikonzept

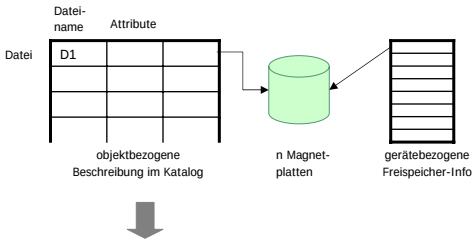
- Dateien repräsentieren externe Speichermedien für Anwendungsprogramme in einer geräteunabhängigen Weise. Das Dateisystem verdeckt die Eigenschaften physischer Speichergeräte und bietet als abstrakte Sicht Dateien, die als „lange Byte-Vektoren“ behandelt werden können.

DB – Speicher
≙
Menge von Dateien

- Gründe für ein Dateikonzept
  - selektive Aktivierung von Dateien: on/offline-Problem
  - dynamische Definition
  - temporäre Dateien
  - Einsatz unterschiedlich schneller Speichermedien
  - kürzere Adresslängen
- Dateisystem
  - erlaubt eine Reihe von Operationen (vereinfachte Definition)
  - ist als lokales Dateisystem oder als eigenständiger Datei-Server realisiert
  - verwaltet typischerweise einen hierarchischen Namensraum
- Operationen:
  - CREATE/DELETE zum Anlegen und Löschen einer Datei
  - OPEN/CLOSE zur Vorbereitung und Beendigung der Verarbeitung einer Datei
  - READ/WRITE zum Lesen und Schreiben eines Blocks einer Datei

Realisierung eines Dateisystems

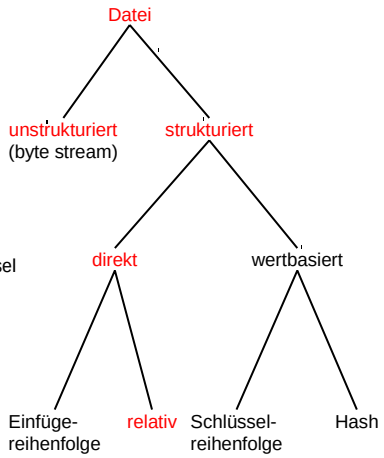
- Dateikatalog für alle Dateien
  - feste Position
  - effiziente Implementierung und Verarbeitung
- Freispeicherverwaltung für Externspeicher
  - formatierte Bitlisten
  - hierarchische Struktur
- Anlegen/Reservieren von Speicherbereichen
  - Erstzuweisung
  - Erweiterung
- Einheit des physischen Zugriffs: Block
  - Blöcke variabler Länge?
  - feste Blocklänge pro Datei
  - chained I/O wichtig für hohe E/A-Leistung



- Objektbezogene Beschreibung durch Dateideskriptor
  - Dateiname, OwnerID, . . .
  - Zugriffskontrollliste
  - Zeitinformation über Erzeugung, letzter Zugriff, letzte Archivierung, ...
  - Dateigröße, Externspeicherzuordnung, ...

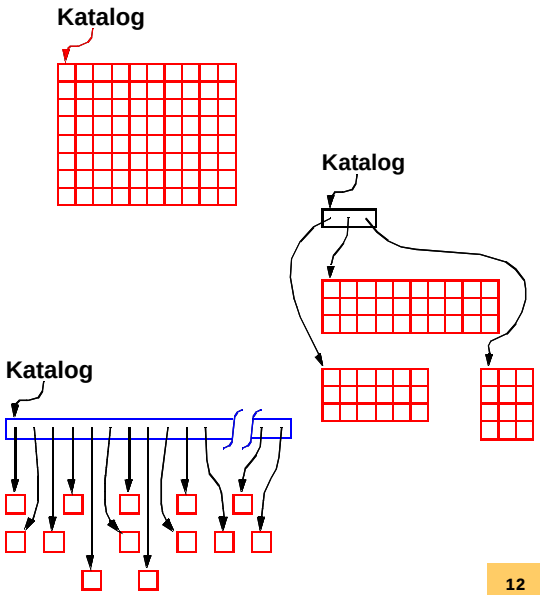
Dateiorganisationsformen

- Dateisystem
  - verwaltet die erzeugten Dateien und führt die Lese-/Schreibzugriffe durch
  - hält einen Deskriptor für jede Datei
- Organisationsformen
  - Datei in Einfügereihenfolge (entry-sequenced): Einfügen an Dateiende, Satzadresse ist RBA (relative Byteadresse), sequentieller Zugriff oder direkter Zugriff über RBA
  - relative Datei: Organisationsform ist ARRAY OF RECORDS
  - wertbasierte Dateien erlauben Zugriff über Satzschlüssel
  - Datei in Schlüsselreihenfolge (key-sequenced): Speicherung der Sätze in Schlüsselreihenfolge, direkter Zugriff über Schlüssel und sortiert-sequentieller Zugriff. Bezeichnungen in existierenden Systemen (logische Zugriffsmethoden): SAM, ISAM, VSAM, . . .
  - Hash-Datei: direkter Zugriff über Schlüsseltransformation

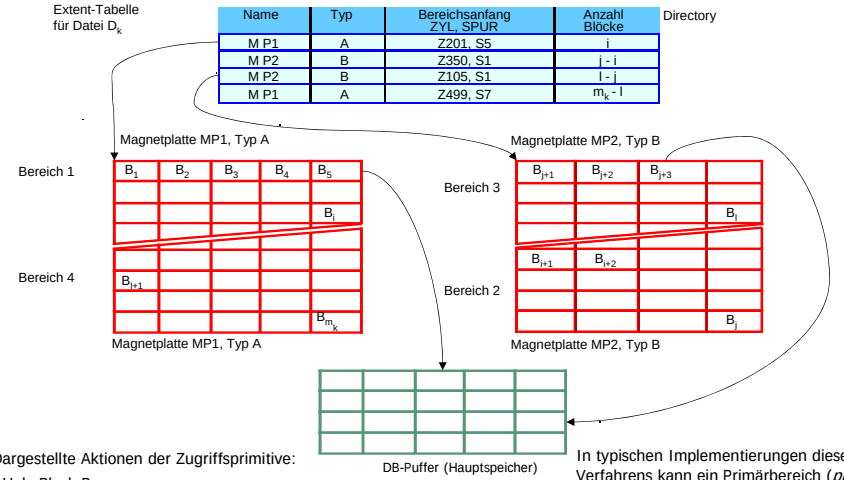


Blockzuordnungsverfahren

- Statische Datei-Zuordnung
  - direkte Adressierung
  - minimale Zugriffskosten
  - keine Flexibilität
- Dynamische Extent-Zuordnung
  - Adressierung über eine kleine Tabelle
  - geringe Zugriffskosten
  - mäßige Flexibilität
- Dynamische Block-Zuordnung
  - Adressierung über eine große Tabelle
  - hohe Zugriffskosten
  - maximale Flexibilität



Blockzuordnung durch Extent-Tabellen



Dargestellte Aktionen der Zugriffsprimitive:
- Hole Block B_5
- Hole Block B_{j+3}

Falls ein Block zum Schreiben in den DB-Puffer geholt wird, ist er nach Freigabe wieder zurückzuschreiben

In typischen Implementierungen dieses Verfahrens kann ein Primärbereich (*primary extent*) angelegt werden, der um bis zu 15 Sekundärbereiche (*secondary extents*, mit jeweils gleicher Größe) erweitert werden kann.

Kapitel 2: Externspeicherverwaltung

- Abbildung von Dateien und Blöcken
  - Zweistufige Speicherhierarchie
  - Allgemeine Aufgaben der Externspeicherverwaltung
- Dateisystem
  - Operationen, Adressierung und Abbildung
  - Verfahren der Blockzuordnung
    - statisch
    - dynamische Extent-Zuordnung
    - dynamische Block-Zuordnung
- Maßnahmen zur Fehlertoleranz
  - Lesen und Schreiben von Blöcken
  - Nutzung von Speicherredundanz
- Abbildung von Segmenten und Seiten
  - Segmentkonzept
  - verzögerte Einbringstrategien
    - Schattenspeicherkonzept (shadow page mechanism)
    - Zusatzdatei (differential file)
  - Bewertung der Seitenzuordnungsverfahren

Maßnahmen zur Fehlertoleranz

- MTTF von verschiedenen Plattenfehlern

Type of Error	MTTF	Recovery	Consequences
Soft data read error	1 hour	Retry or ECC	None
Recoverable seek error	6 hours	Retry	None
Maskable hard data read error	3 days	ECC	Remap to new sector and rewrite good data
Unrecoverable data read error	1 year	None	Remap to new sector, old data lost
Device needs repair	5 years	Repair	Data unavailable
Miscorrected data read error	10 years	None	Read wrong data

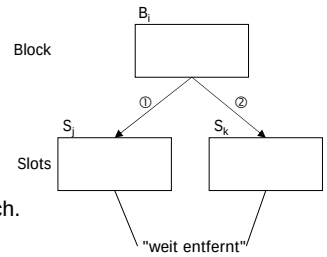
Quelle: [GR93]

Maßnahmen zur Fehlertoleranz

- Einfaches Lesen
  - Lesen kann durch transiente Fehler gestört werden.
  - Zusatzmaßnahme: erfolgloses Lesen wird n-mal wiederholt (sicheres Lesen)
- Einfaches Schreiben
  - Schreiben des Blocks als atomare Aktion (ganz oder überhaupt nicht) kann nicht garantiert werden.
  - Schreiben kann durch transiente und dauerhafte Fehler zu falschen Resultaten führen.
  - Crash kann teilweise geschriebenen Block zurücklassen.
- Sicheres Schreiben (read-after-write)
  - Nach einem einfachen Schreiben wird der Block wieder gelesen und mit dem Originalblock verglichen.
  - Operationsfolge wird wiederholt, bis Block erfolgreich geschrieben ist.
  - Schreiben ist gesichert gegen transiente Fehler.
  - Problem bei Crash nicht behoben.

Maßnahmen zur Fehlertoleranz

- Stabiles Schreiben (duplexed write)
  - Jeder Block hat eine Versionsnummer, die bei jedem stabilen Schreiben erhöht wird.
  - Jeder Block wird in festgelegter Reihenfolge in zwei verschiedene Slots S_j und S_k geschrieben
  - Prinzip: Stabiler Speicher
  - Annahme: Ein Block wird nicht in einen falschen Slot geschrieben, sonst ist read-after-write erforderlich.
  - Lesen von B_i erfolgt erst von Slot S_j. Falls es erfolgreich ist, wird angenommen, dass es sich um die jüngste gültige Version von B_i handelt.
  - Falls das Lesen von Slot S_j scheitert, wird Slot S_k gelesen.
  - Da ein Systemausfall nur einen Schreibvorgang unterbrechen kann, ist bei Wiederanlauf stets eine Version des Blockes verfügbar.
  - Schreiben ist somit gesichert gegen dauerhafte Fehler. Dass beide Versionen nicht lesbar sind, gilt als unerwartet.

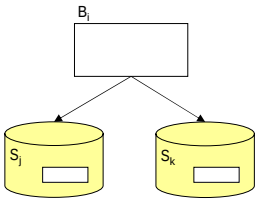


- einfaches Schreiben: ① dann ② (synchron)
- Atomarität für einen Block
- verschiedene Platten, Kontroller, E/A-Pfade

Weitere Prinzipien der Speicherredundanz

- Einbringeinheit: 1 Block
  - Fehlertoleranzmaßnahme für einen Block gegen Schreibfehler, Systemausfall, Block- (Spur-, Geräte-) Zerstörung
- Schreiben mit Logging (logged write)
  - Nach dem Lesen wird ein Block B_i mit seinem alten Inhalt auf einen sicheren Platz L geschrieben.
  - Nach Aktualisierung wird B_i mit einem einfachen Schreiben auf seinen alten Platz zurück geschrieben (ggf. read-after-write).
  - Falls das Schreiben erfolgreich war, wird die Kopie von B_i auf L nicht mehr gebraucht.

- Prinzip: Spiegelplatten



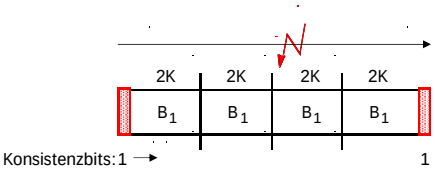
  - auf Platten- oder Dateibasis
  - synchron oder asynchron
  - gleichzeitig oder zeitlich versetzt (Schattenprozess)
  - keine Atomarität automatisch durch Algorithmus

Weitere Prinzipien der Speicherredundanz

- Erkennung von fehlerhaften Blöcken
  - Schutz vor gewissen Verfälschungen:
    - Parity-Bits
    - Prüfsummen
  - Platten-Hardware kann durch Parity-Bits eigenständig herausfinden, ob ein Sektor vollständig geschrieben wurde oder nicht
    - ausreichend, wenn ein Block ganz in einem Sektor (hier gleich Slot) gespeichert werden kann
- Slot = Sektor
  - Benutzung von jeweils einem Bit im ersten und letzten Byte eines Blocks
  - Beiden Konsistenzbits werden jeweils identische Werte zugewiesen
- Multi-Sektor-Slots
  - Fehlerhafter Block wird nur erkannt, wenn beim Schreiben der Sektoren die durch den Block vorgegebene Reihenfolge eingehalten wird
  - SCSI-Laufwerke nehmen selbständig zur Leistungsoptimierung eine Umordnung der Schreibvorgänge auf den Sektoren vor
  - Prüfsumme über den gesamten Block reduziert die Wahrscheinlichkeit erheblich, ein partielles Schreiben zu übersehen, kann dies jedoch nicht ausschließen (enorm teuer)
  - Entsprechend den n Sektoren wird ein Block (logisch) unterteilt; aus jedem dieser Teile wird ein Bit genommen und als Prüfbit betrachtet. Diese n Bits werden mit identischen Werten belegt und bei jedem Schreiben invertiert.
    - Wie lässt sich erreichen, dass diese n Bits nicht die Benutzerdaten im Block verfälschen?

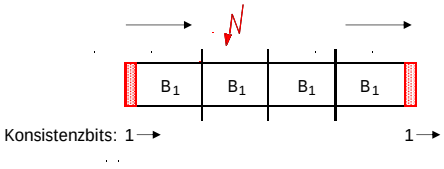
Erkennen unvollständig geschriebener Blöcke

- Sequentielles Schreiben

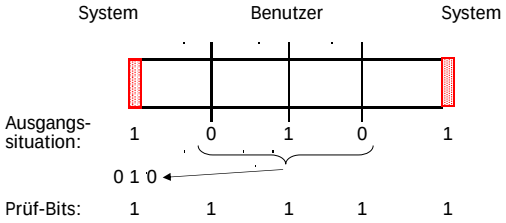


für sequentielles Schreiben ok!

- Problem: Schreiben „außer der Reihe“



- Lösung: Schreiben von Prüf-Bits in jeden Sektor



Quelle: [Moh95]

Kapitel 2: Externspeicherverwaltung

- Abbildung von Dateien und Blöcken
  - Zweistufige Speicherhierarchie
  - Allgemeine Aufgaben der Externspeicherverwaltung
- Dateisystem
  - Operationen, Adressierung und Abbildung
  - Verfahren der Blockzuordnung
    - statisch
    - dynamische Extent-Zuordnung
    - dynamische Block-Zuordnung
- Maßnahmen zur Fehlertoleranz
  - Lesen und Schreiben von Blöcken
  - Nutzung von Speicherredundanz
- Abbildung von Segmenten und Seiten
  - Segmentkonzept
  - verzögerte Einbringstrategien
    - Schattenspeicherkonzept (shadow page mechanism)
    - Zusatzdatei (differential file)
  - Bewertung der Seitenzuordnungsverfahren

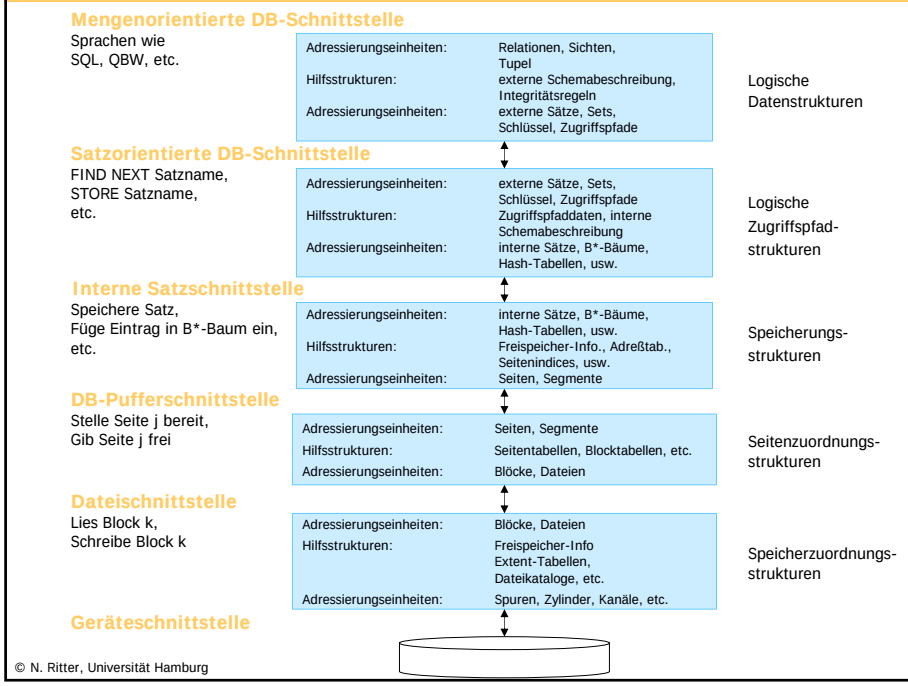
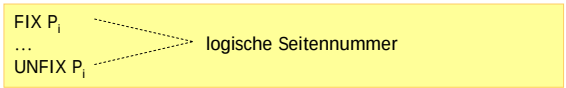


Abbildung von Segmenten und Seiten

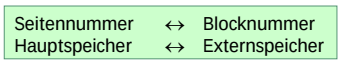
- Vorteile eines Segmentkonzeptes
  - Ermöglichung verzögerter Einbringstrategien
  - selektive Einführung von zusätzlichen Attributen, z.B. zur Erhöhung der Fehlertoleranz
  - Segmente als Einheiten des Sperrens, der Recovery und der Zugriffskontrolle
  - Bei geeigneter Abbildung auf Dateien bleiben Vorteile des Dateikonzeptes erhalten
- Anmerkung:
  - In verschiedenen DBS werden bestimmte Segmenttypen auch als Tablespaces bezeichnet. Sie dienen der Speicherung von Tabellen (Relationen) sowie ggf. Indexstrukturen. In manchen DBS werden Indexstrukturen in eigenen Segmenttypen (sog. Indexspaces) verwaltet. Die Abbildung von Tablespaces auf mehrere Dateien ist dabei möglich.
- Zusammen mit der DB-Pufferverwaltung (siehe Kapitel 3)
  - Aufteilung des logischen DB-Adressraumes in Segmente mit sichtbaren Seitengrenzen
  - Bereitstellen und Freigeben von Seiten im DB-Puffer
  - Vorbereitung von E/A-Anforderungen an die Dateiverwaltung
  - Optimierung von Ersetzungsstrategien
  - Unterstützung von Segmenten verschiedenen Typs
  - Neue Aufgaben: Verwaltung von Seiten variabler Länge und von langen Objekten

Seitenzuordnungsstrukturen
Abbildung von Segmenten und Seiten

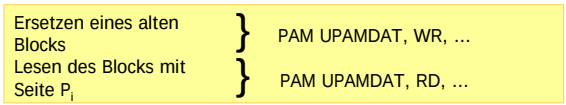
- Beispiel: Operationen an der oberen Schnittstelle



- Abbildungsfunktion:



- Aufgaben:



- Eigenschaften der oberen Schnittstelle
  - Linearer Adressraum, aufgeteilt in Seiten fester Länge
  - Innerhalb eines Segmentes können die Moduln der nächst höheren Schicht frei adressieren wie in einem virtuellen Adressraum
  - Ein DB-Segment ist nicht-flüchtig (wenn nicht explizit anders vereinbart)

Segmenttypen in einem DBMS

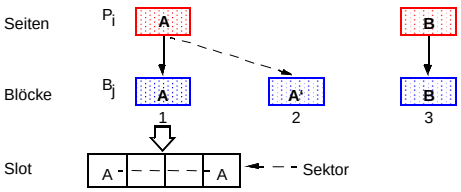
- Klassifikation von Segmenttypen

Eigenschaften \ Segment-Typen	Segment-Typ 1	Segment-Typ 2	Segment-Typ 3	Segment-Typ 4	Segment-Typ 5
Benutzung	öffentlich	privat	privat	privat	privat
Lebensdauer	permanent	permanent	permanent	permanent	temporär in Transaktion
Öffnen und Schließen	automatisch durch System		explizit durch Benutzer		
Wiederherstellung im Fehlerfall	automatisch durch System		explizit durch Benutzer	kein Wiederherstellungsmechanismus	

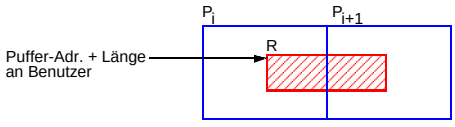
- Beispiele für die Verwendung der Segmenttypen
  - Typ 1: Katalog, Schema-Information, Log, alle gemeinsam benutzbaren DB-Teile
  - Typ 2: Teile der DB, die für bestimmte Benutzer bzw. Benutzergruppen reserviert sind
  - Typ 3: Lokale Kopien von Teilen der DB (Sichten) für einzelne Benutzer (Snapshots)
  - Typ 4: Hilfsdateien für Benutzerprogramme
  - Typ 5: Temporärer Speicher z. B. für Sortierprogramme

Warum sichtbare Seiten und Segmente?

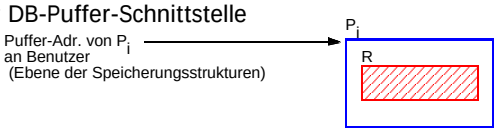
- Explizite Abbildung auf Blöcke und Dateien erhöht Fehlertoleranz



- DB-Puffer mit Satzschnittstelle?
  - „lineare“ Hauptspeicherabbildung: Probleme unsichtbarer Seitengrenzen
  - „spanned record facility“
  - benachbarte Seiten müssen im DB-Puffer benachbart angeordnet sein
  - erhebliche Abbildungs- und Ersetzungsprobleme

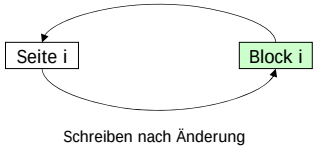


- Sichtbare Seitengrenzen an der DB-Puffer-Schnittstelle



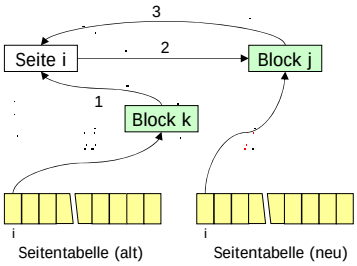
Verfahren zur Seitenzuordnung

- Direkte Seitenzuordnung (update-in-place)

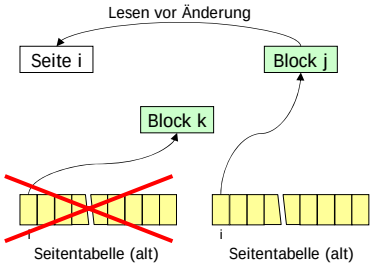


Schreiben nach Änderung

- Indirekte Seitenzuordnung
  1. Lesen vor Änderung
  2. Schreiben nach Änderung
  3. Lesen nach Änderung

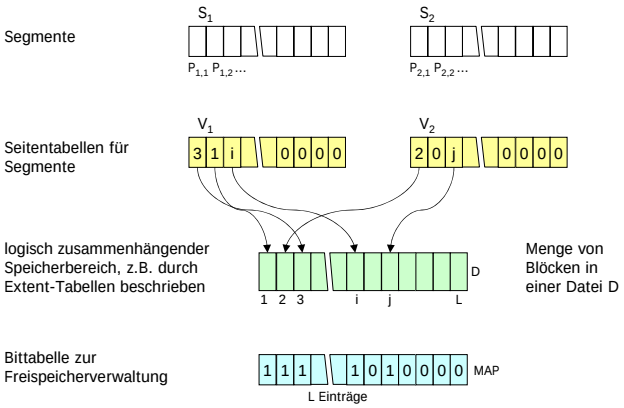


- Zustand nach verzögertem Einbringen (mengenorientiert)



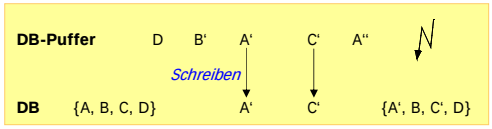
Prinzip der indirekten Seitenzuordnung

- Abbildung von Seitennummer → Blocknummer mit Hilfe der Seitentabelle

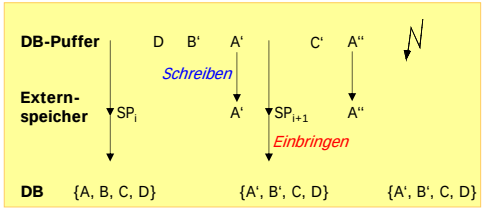


Einbringen von Seiten – Konsistenz der DB nach Crash

- direktes Einbringen:
  $P_i \Rightarrow B_j$; $P_i' \Rightarrow B_j'$

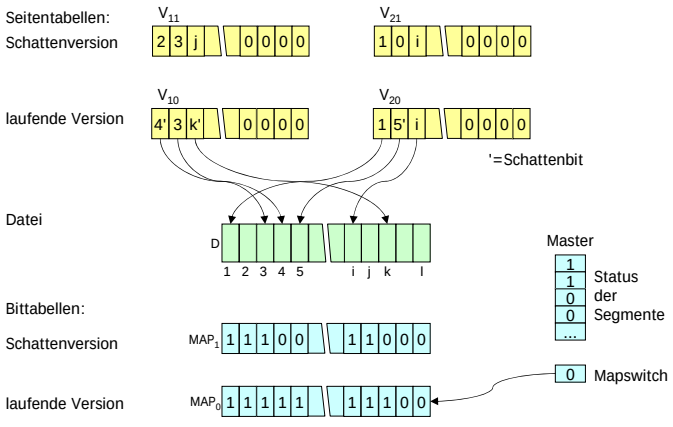


  - Schreiben = Einbringen
  - DB nach Crash: chaos-konsistent!
- verzögertes Einbringen:
  $P_i \Rightarrow B_j$; $P_i' \Rightarrow B_k$; $j \neq k$



  - Schreiben ≠ Einbringen
  - DB nach Crash: aktionskonsistent (operationskonsistent)!
  - geeignete Wahl der Sicherungspunkte SP_i : Operationsgrenzen beachten!

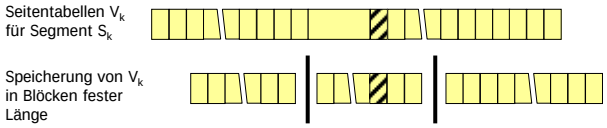
Schattenspeicher-Verfahren: Prinzip



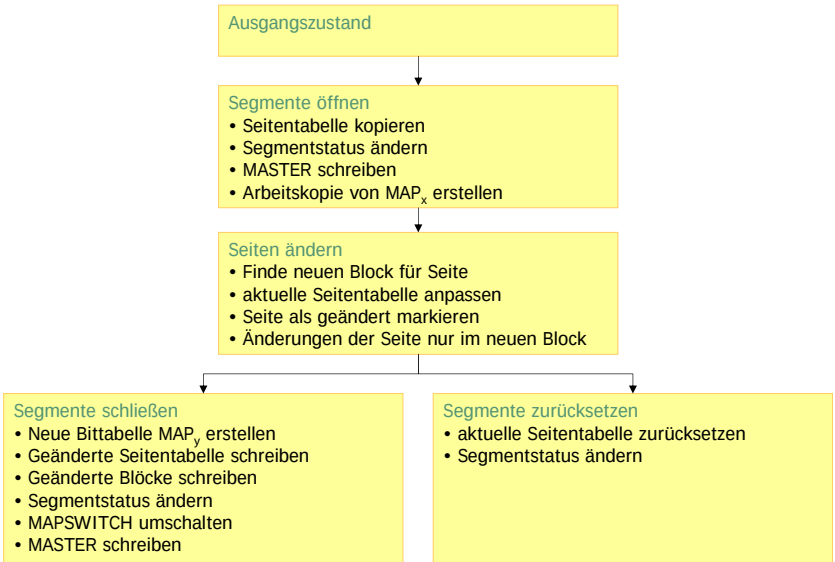
Quelle: [Lor77]

Schattenspeicher-Verfahren: Speicherverwaltung

- Prinzip der indirekten Seitenzuordnung
- Zerlegung von Seitentabelle V_k in einzelne Blöcke
  - Auch die Seitentabelle muss im linearen Adressraum untergebracht und auf Blöcke abgebildet werden.



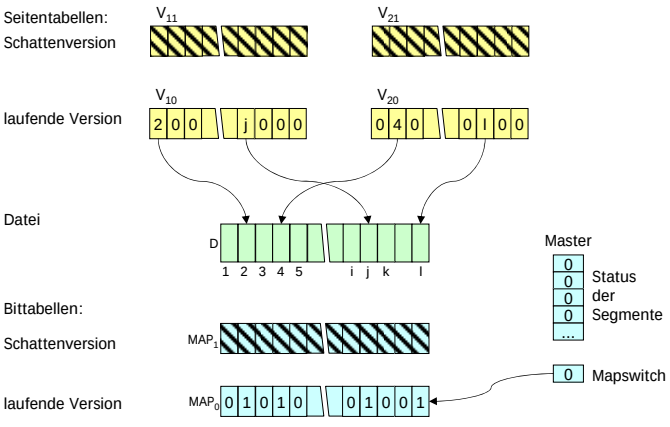
Ablauf eines Änderungsintervalls



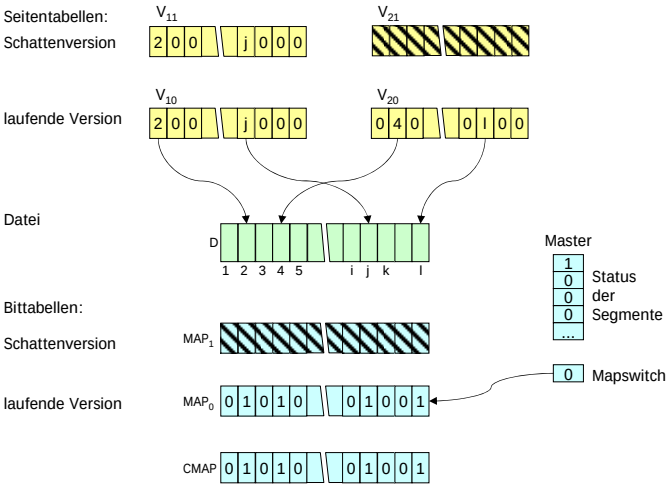
Segment öffnen

- Ausgangszustand:
  - Status (i) enthält den Eröffnungszustand für Segment i (hier: alle Segmente geschlossen).
  - MAPSWITCH zeigt an, welche der beiden (gleichberechtigten) Tabellen Map_0 und Map_1 das aktuelle Verzeichnis belegter Blöcke enthält.
- Wenn Segment k für Änderungen geöffnet werden soll, sind folgende Schritte auszuführen:
  - kopiere V_{k0} nach V_{k1}
  - STATUS(k) := 1
  - Schreibe MASTER in einer ununterbrechbaren Operation aus
  - Lege im Hauptspeicher eine Arbeitskopie CMAP von der aktuelle Bittabelle MAP_x an.

Ausgangszustand



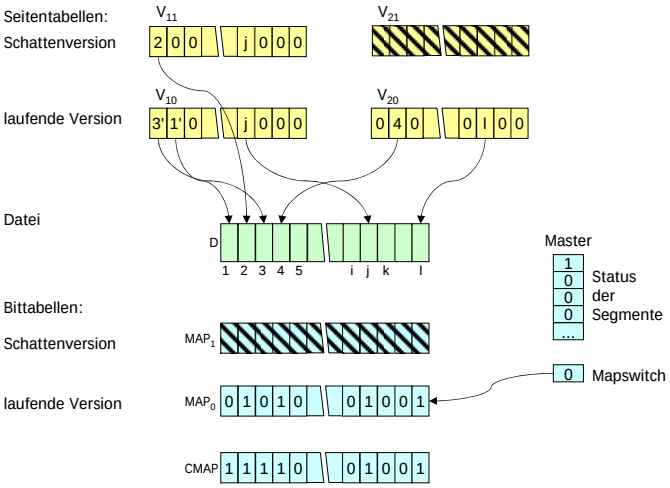
Segment öffnen



Seite ändern

- Wenn eine Seite P_i erstmalig seit Eröffnen des Segments geändert werden soll, sind folgende Aktionen auszuführen:
  - Lies Seite P_i aus Block j = $V_{k0}(i)$
  - Finde einen freien Block j' in CMAP
  - $V_{k0}(i)$ = j'
  - Markiere Seite P_i in $V_{k0}(i)$ als geändert
  - Bei weiteren Änderungen von P_i wird Block j' verwendet.
- Anmerkungen:
  - Auf einer Blockmenge D können gleichzeitig mehrere Segmente für den Änderungsbetrieb geöffnet sein.
  - Map_0, Map_1 und CMAP beziehen sich auf die Blockmenge D. CMAP enthält für alle Segmente die mit Schatten- bzw. aktuellen Seiten belegten Blöcke.
  - Einer Seite wird nur bei der erstmaligen Änderung nach Eröffnen des Segmentes ein neuer Block zugewiesen.

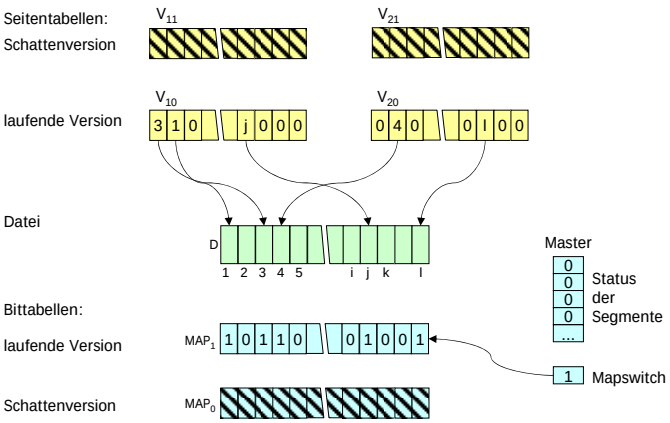
Seite ändern



Änderungsintervall beenden

- Beenden eines Änderungsintervalls für Segment k:
  - Erzeuge die Bitliste mit der aktuellen Speicherbelegung in der bisher nicht genutzten Bittabelle MAP_y (neue Blöcke belegt, alte freigegeben)
  - Schreibe MAP_y
  - Schreibe V_{k0}
  - Schreibe alle geänderten Blöcke
  - Markiere Segment k als geschlossen (STATUS(k) := 0)
  - MAP_y wird aktuelle Bittabelle (MAPSWITCH = y)
  - Schreibe MASTER in einer ununterbrechbaren Operation aus

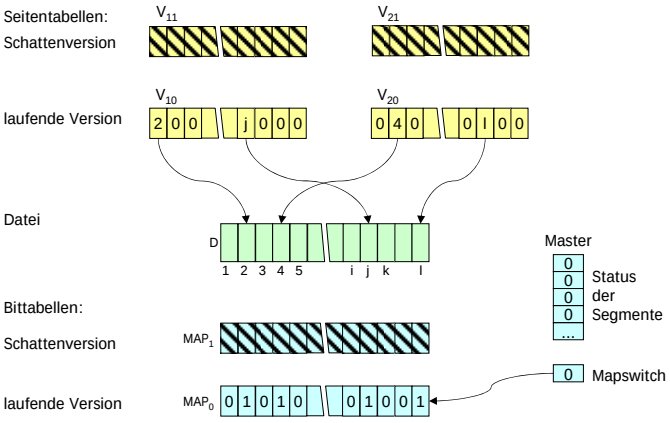
Änderungsintervall beenden



Segmente zurücksetzen

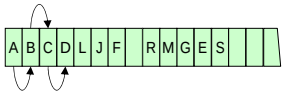
- Zum Zurücksetzen geöffneter Segmente muss lediglich V_{k1} in V_{k0} kopiert und STATUS(k) auf 0 gesetzt werden.

Segmente zurücksetzen

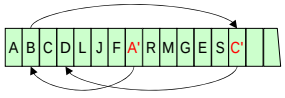


Erhaltung der physischen Clusterbildung

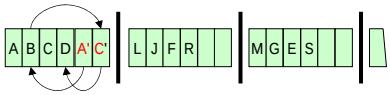
- Wegen der dynamischen Neuzuordnung von Blöcken können beim Schattenspeicherverfahren physische Nachbarschaften von logisch zusammengehörigen Seiten bei Änderungen i.A. nicht aufrechterhalten werden.
  - Bsp.: Die Seiten A, B, C, D mögen in allen Transaktionen in eben dieser Reihenfolge berührt werden:



  - Die Änderung von A → A' und C → C' führt zu:



  - Eine Verbesserung kann durch Einteilung der Datei in physische Cluster der Größe p und der Segmente in logische Cluster der Größe l sowie deren gegenseitige Zuordnung erreicht werden. Beispiel mit p=6, l=4:



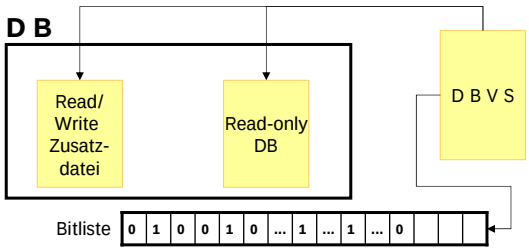
  - Bei Plattenspeichern wird ein physisches Cluster i.A. ein Zylinder sein

Bewertung des Schattenspeicher-Verfahrens

- Vorteile
  - Rücksetzen auf letzten konsistenten Zustand sehr einfach
  - Flexiblere Schreibprotokolle für Log-Daten: Pufferung bis zum Umschalten auf einen neuen Zustand möglich
  - Logisches Logging möglich, da stets operationskonsistenter Zustand verfügbar ist
  - Bei katastrophalen Fehlern ist die Wahrscheinlichkeit höher, einen brauchbaren Zustand der DB zu rekonstruieren
- Nachteile
  - Hilfsstrukturen (MAP und Seitentabellen V_i) werden so groß, dass Blockzerlegung notwendig wird
  - Die Seitentabellen V_i belegen etwa 0.1-0.2% der DB-Größe, was bei großen DB's ($\geq$ n GB) zu einem hohen Anteil von Seitenfehlern im DB-Puffer für die Zugriffe auf V_i führen kann
  - Periodische Sicherungspunkte erzwingen Ausschreiben des gesamten DB-Puffers
  - Physische Clusterbildung logisch zusammengehöriger Seiten wird beeinträchtigt bzw. zerstört
  - Zusätzlicher Speicherplatz für die Doppelbelegung;
    lange Batch- (Änderungs-)Programme werden dadurch schlecht unterstützt

Zusatzdatei-Verfahren

- Idee:
  Wenn eine Datei für Änderungsbetrieb geöffnet wird, wird eine zusätzliche, temporäre Datei angelegt, in welche während des Änderungsintervalls alle modifizierten Blöcke geschrieben werden. Die eigentliche Datei wird dabei nicht verändert. Am Ende des Änderungsintervalls werden alle geänderten Blöcke aus der temporären in die permanente Datei kopiert. Das Einbringen der Änderungen erfolgt also verzögert.
- Prinzip:



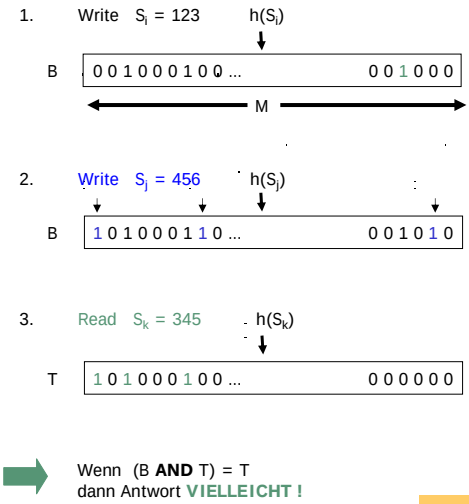
  - Das wesentliche Problem besteht darin, während des Änderungsbetriebes für eine gegebene Seiten-Nummer zu entscheiden, ob die aktuelle Version in der temporären oder permanenten Datei steht.
  - Nutzung einer Bitliste, die anzeigt, ob eine Seite möglicherweise geändert wurde
  - Hashabbildung erlaubt Begrenzung des Speicherplatzbedarfs

Quelle: [SL76]

Bloom-Filter

- **Wirkungsweise**
  - Bitliste B der Länge M im Hauptspeicher
  - Signatur des Satzes in B bei seiner Änderung
  - Hash-Funktionen zur Satzabbildung adressieren x Bits, die in B auf 1 gesetzt werden
- **Aufsuchen von Satz S_i**
  - Erzeugen der charakteristischen x Bits in temporärer Bitliste T
  - **AND**-Operation von T und B in Erg
  - wenn alle x Bits in Erg gesetzt
    → VIELLEICHT
    sonst → NEIN

- **Beispiel:**



Wenn (B **AND** T) = T
dann Antwort **VIELLEICHT !**

Zusammenfassung

- Speicherzuordnungsstrukturen erfordern effizientes Dateikonzept
  - viele Dateien variierender, nicht statisch festgelegter Größe
  - Wachstum und Schrumpfung erforderlich
  - permanente und temporäre Dateien
- Empfohlene Datei-Eigenschaften:
  - direkter und sequentieller Blockzugriff
  - Blockgröße pro Datei definierbar
  - Blockzuordnung über dynamische Extents
  - Dynamische Blockzuordnung (bei UNIX als mehrstufiger Baum) untauglich für große Dateien
- Segmentkonzept
  - erlaubt die Realisierung zusätzlicher Attribute für die DB-Verarbeitung (Recovery, Clusterbildung für Relationen usw.)
- Zweistufige Abbildung
  - von Segment/Seite auf Datei/Block und diese auf Slots der Magnetplatte erlaubt Einführung von Abbildungsredundanz durch verzögertes Einbringen
- Verzögerte Einbringstrategien
  - sind teurer als direkte, besitzen jedoch implizite Fehlertoleranz
  - sie belasten den Normalbetrieb zugunsten der Recovery
- Direkte Einbringstrategie (update-in-place)
  - einfach zu implementieren
  - keine Zusatzkosten zur Ausführungszeit für die Seitenzuordnung
  - Fehlertoleranz nur durch explizite Logging- u. Recovery-Funktionen

Ergänzende Literatur zu diesem Kapitel

[GR93] J. Gray, A. Reuter: Transaction Processing – Concepts and Techniques, Morgan Kaufmann, 1993.

[Lor77] R. A. Lorie: Physical Integrity in a Large Segmented Database. In: ACM TODS, Vol. 2, No. 1, 1977.

[Moh95] C. Mohan: Disk read-write optimizations and data integrity in transaction systems using write-ahead logging. In: Proc. of the 11th International Conference on Data Engineering, Taipei, Taiwan, March 6-10, 1995.

[SL76] D. G. Severance, G. M. Lohman: Differential Files: Their Application to the Maintenance of Large Databases. In: ACM TODS, Vol. 1, No. 3, 1976.