Dresden Database Systems Group

2. Abbildung von Seiten auf Blöcke

Architecture of Database Systems

Semantik relationaler Datenbanken

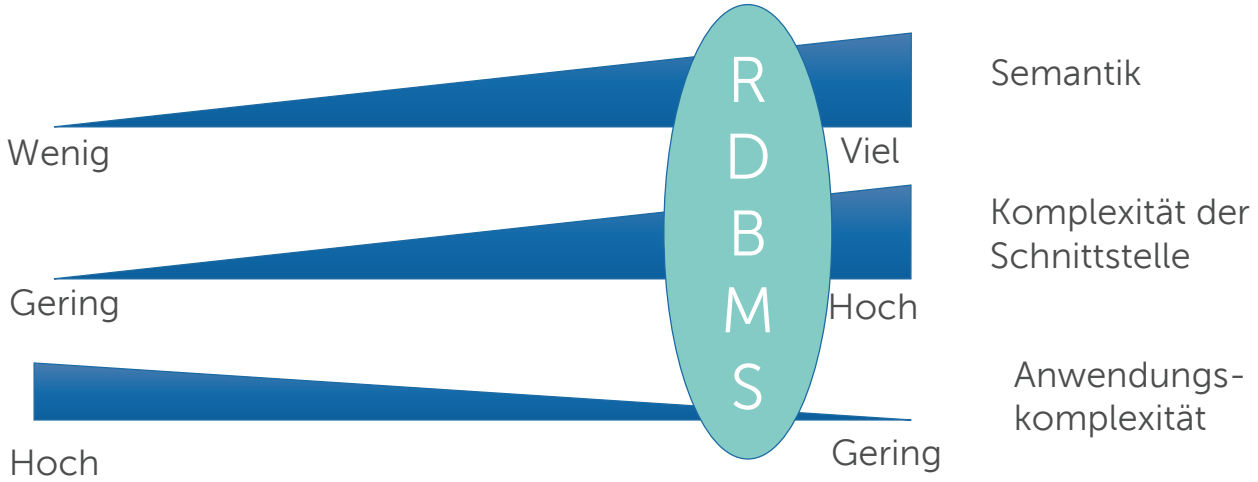
Dresden Database Systems Group

Wozu Dient Eine Datenbank?

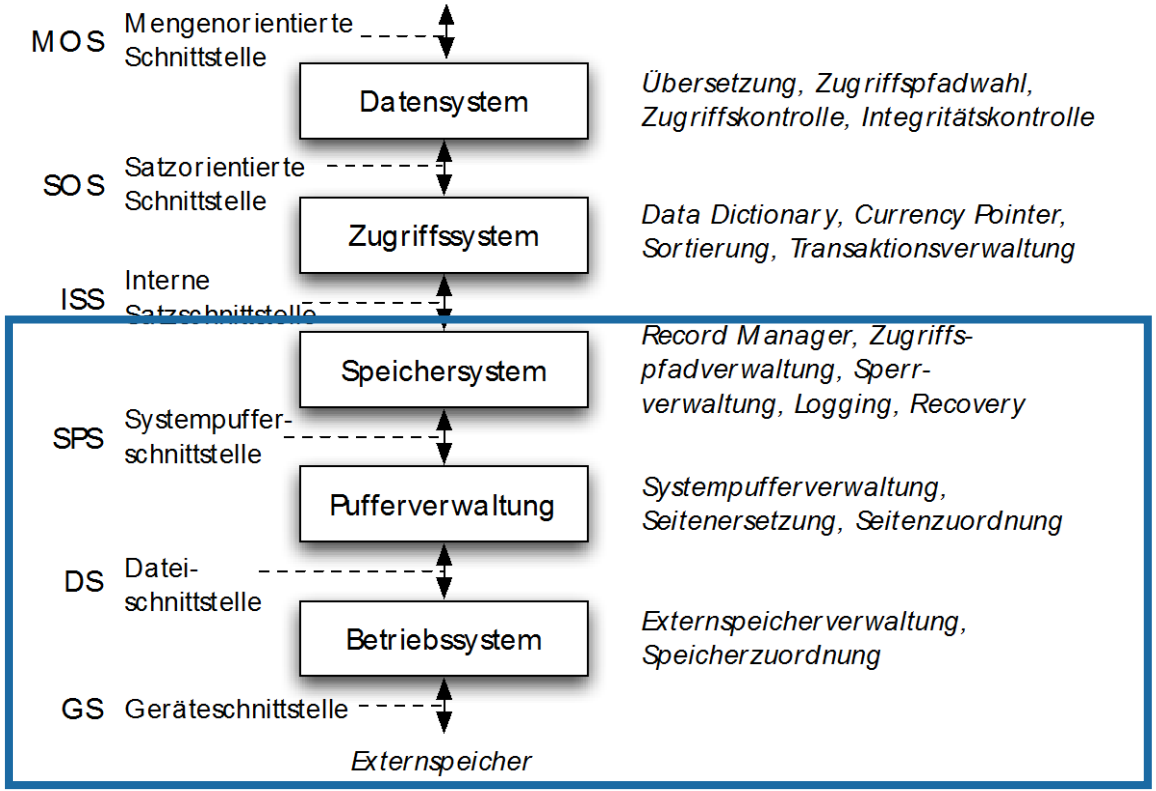
- Daten permanent sichern
- Daten anderen bereitstellen
- Daten auffinden
- Daten manipulieren

Hierzu muss die Datenbank die Daten „verstehen“
→ Was ist die **Semantik**?

Storage Service



5-Schichten-Architektur (RDBMS)



Schichten für Storage Service

Transaktionen und ACID

Dresden Database Systems Group

Definition

- Eine Transaktion ist eine Folge von Operationen (Aktionen), die die Datenbank von einem konsistenten Zustand in einen konsistenten, eventuell veränderten, Zustand überführt, wobei das ACID-Prinzip eingehalten werden muss.

Aspekte

- Semantische Integrität: Korrekter (konsistenter) DB-Zustand nach Ende der Transaktion
- Ablaufintegrität: Fehler durch „gleichzeitigen" Zugriff mehrerer Benutzer auf dieselben Daten vermeiden

ACID-Eigenschaften

- Atomicity (Atomarität)
  Transaktion wird entweder ganz oder gar nicht ausgeführt
- Concistency (Konsistenz)
  Datenbank ist vor Beginn und nach Beendigung einer Transaktion jeweils in einem konsistenten Zustand
- Isolation (Isolation)
  Nutzer, der mit der Datenbank arbeitet, sollte den Eindruck haben, dass er mit dieser Datenbank alleine arbeitet
- Durability (Dauerhaftigkeit/Persistenz)
  nach erfolgreichem Abschluss einer Transaktion muss das Ergebnis dieser Transaktion „dauerhaft" in der Datenbank gespeichert werden

Speicherhierarchie

Speichermedien

Dresden Database Systems Group

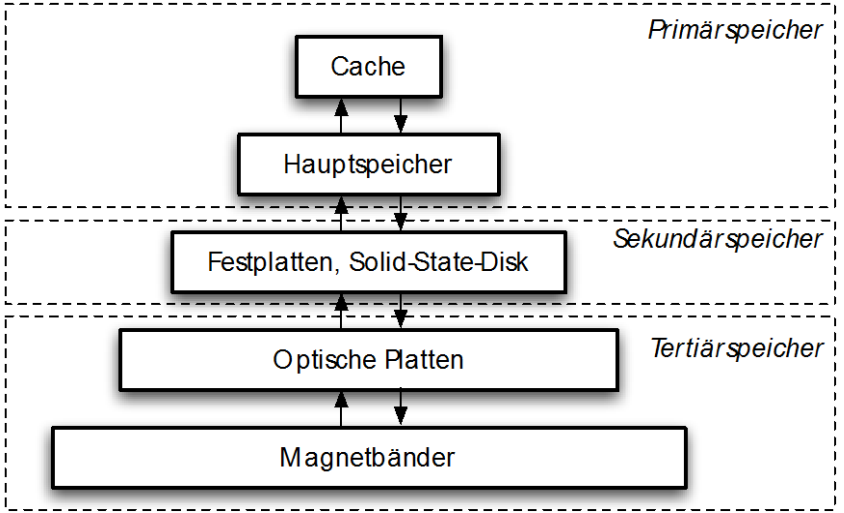
Verschiedene Zwecke

- Daten zur Verarbeitung bereitstellen
- Daten langfristig speichern (und trotzdem schnell verfügbar halten)
- Daten sehr langfristig und preiswert archivieren unter Inkaufnahme etwas längerer Zugriffszeiten

In diesem Abschnitt

- Speicherhierarchie
- Magnetplatte
- Kapazität, Kosten, Geschwindigkeit

Speicherhierarchie



Speichermedien

Dresden Database Systems Group

Primärspeicher

- Cache und Hauptspeicher
- flüchtig, d.h. Persistenz kann nicht garantiert werden
- sehr schnell, Zugriff auf Daten fein granular: jedes Byte adressierbar

Sekundärspeicher

- Sekundärspeicher oder Online-Speicher
- meist Plattenspeicher, nicht-flüchtig
- Granularität des Zugriff gröber: Blöcke, oft 512 Bytes
- Zugriffslücke: Faktor 10^5 langsamerer Zugriff
- als auf Primärspeicher

	Primär	Sekundär	Tertiär
Geschwindigkeit	schnell	langsam	sehr langsam
Preis	teuer	preiswert	billig
Stabilität	flüchtig	stabil	stabil
Größe	klein	groß	sehr groß
Granulate	fein	grob	grob

Tertiärspeicher

- zur langfristigen Datensicherung (Archivierung) oder kurzfristigen Protokollierung
- üblich: optische Platten, Magnetbänder
- „Offline Speicher" meist Wechselmedium
- Nachteil: Zugriffslücke extrem groß

Zugriffslücke

Dresden Database Systems Group

Entwicklung

- Magnetplatten pro Jahr 70% mehr Speicherdichte
- Magnetplatten pro Jahr 7% schneller
- Prozessorleistung pro Jahr um 70% angestiegen
- Zugriffslücke zwischen Hauptspeicher (Primär) und Magnetplattenspeicher (Sekundär) beträgt 10^5

Größen

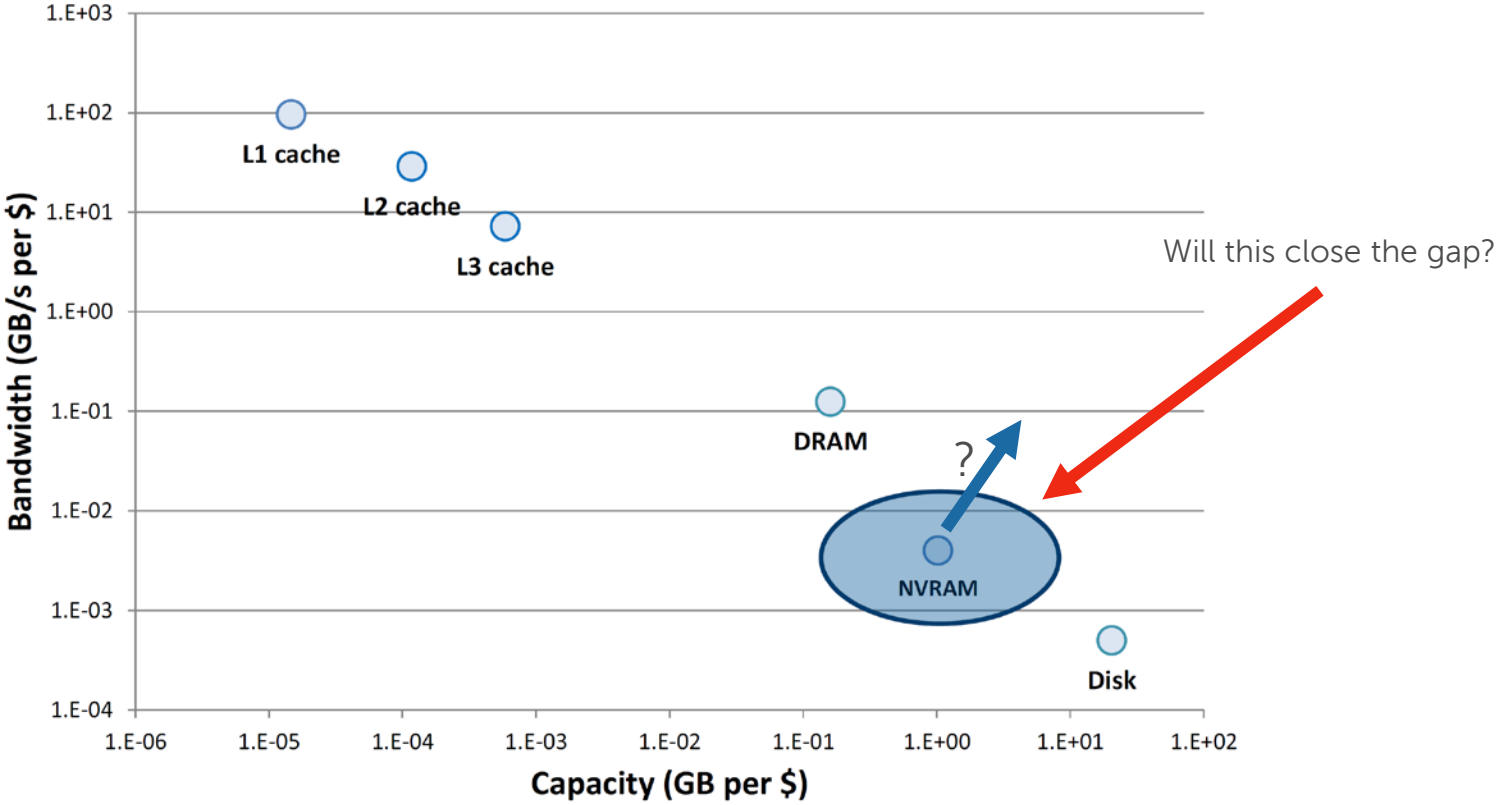
- ns für Nanosekunden (also 10^{-9} Sekunden), ms für Millisekunden (10^{-3} Sekunden)

- KB (KiloByte = 10^3 Byte), MB (MegaByte = 10^6 Bytes), GB (GigaByte = 10^9 Bytes) und TB (TeraByte = 10^{12} Byte)

Zugriffslücke in Zahlen

Speicherart	**typische Zugriffszeit**		**typische Kapazität**
	time	CPU cycles	
Cache-Speicher	6 ns	12	512 KB bis 32 MB
Hauptspeicher	60 ns	120	128 MB bis 8 GB
— Zugriffslücke 10^5 —			
Magnetplatten-speicher	8-12 ms	$16*10^6$	160 GB bis 2 TB
Platten-Farm oder -Array	12 ms	$24*10^6$	im TB-Bereich

Memory Hierarchy



Cache-Hierarchie

Dresden Database Systems Group

Eigenschaften der Speicherhierarchie

- Ebene x (etwas Ebene 3, der Hauptspeicher) hat wesentlich schneller Zugriffszeit als Ebene (x+1) (etwas Ebene 4, der Sekundärspeicher)
- aber gleichzeitig einen weitaus höheren Preis pro Speicherplatz
- und deshalb eine weitaus geringer Kapazität
- Lebensdauer der Daten erhöht sich mit der Höhe der Ebenen

Zugriffslücke abschwächen

- Unterschiede zwischen den Zugriffsgeschwindigkeiten auf die Daten) vermindern → Cache-Speicher speichern auf Ebene x Daten von Ebene (x+1) zwischen

Lokalität des Zugriffs

- Caching-Prinzip funktioniert nicht, wenn immer neue Daten benötigt werden
- in den meisten Anwendungsfällen: Lokalität des Zugriffs
- d.h., Großteil der Zugriffe (in den meisten Fällen über 90%) auf Daten aus dem jeweiligen Cache
- Deshalb: Pufferverwaltung des Datenbanksystems wichtiges Konzept

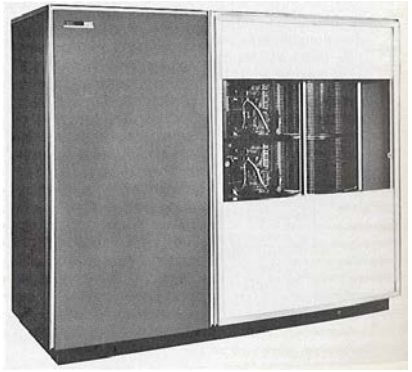
Sekundärspeicher

Dresden Database Systems Group

Typische Merkmale und Entwicklung von Sekundärspeicher

Year	Capacity	Latency	Bandwidth
1983	30MB	48,3ms	0,6MB/s
1994	4,3GB	12,7ms	9MB/s
2003	73,4GB	5,7ms	86MB/s
2009	2TB	5,1ms	95MB/s
2016	8TB	4.16ms	150MB/s
2016 SSD	500GB	50µs	300MB/s

5MB HDD circa 1956



28MB HDD circa 1961

The more that things change ...

HDDs Vs. SSDs Vs. SCM

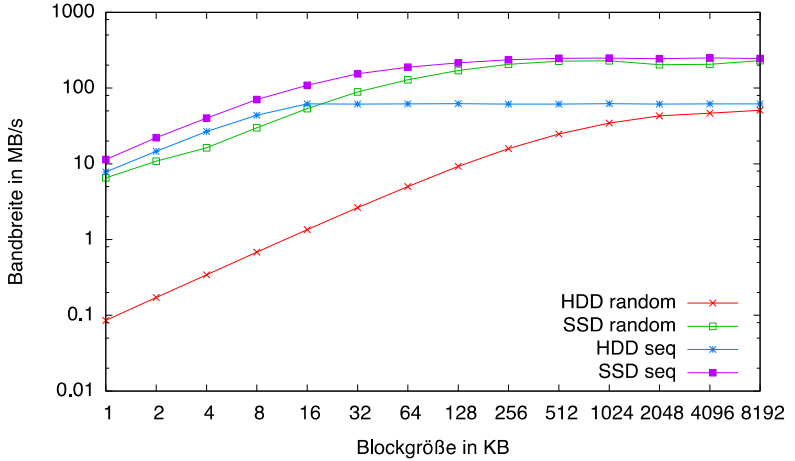
Characteristic	HDD
Storage Granularity	512 B (or 4KB on new HDDs)
Latency	~2 ms
Bandwidth	~120 MB/s
Resilience	∞

Flash Disk

Dresden Database Systems Group

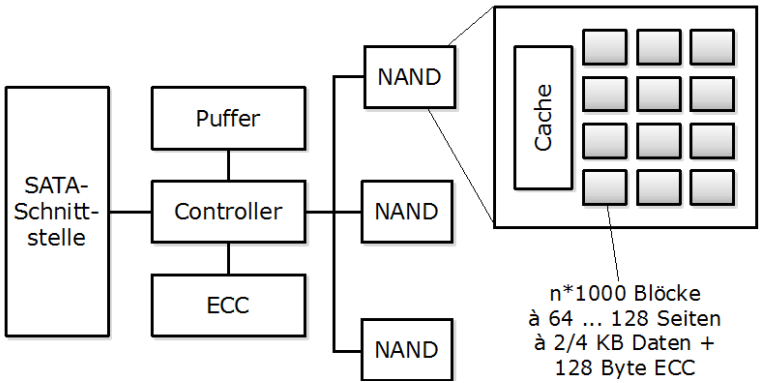
Characteristics

- kleinere Blockgrößen lassen sich effizienter adressieren, sollten aber ein Vielfaches der Flash-Seiten sein
- wahlfreie Lesezugriffe sind effizienter als auf Magnetplatten, sollten aber auf Größen von ca. 4 bis 16 MB begrenzt werden
- konkurrierende IO-Zugriffe sind bis zu einem gewissen Maße ohne negativen Performanzeinfluss durchführbar



Aufbau

- basierend auf EEPROMs in NAND- oder NOR-Technologie
- Arrays (=Flash-Block mit ca. 128 KB) von Speicherzellen, entweder ein Bit (SLC) oder 2-4 Bit (MLC)
- MLC sind langsamer und haben verkürzte Lebensdauer
- initial ist jedes Bit auf 1 gesetzt, durch Reprogrammierung auf 0
- Löschen zurück auf 1 nur für ganzen Block
- Konsequenz: langsames Löschen, begrenzte Lebensdauer



SSD versus HDD comparison

Attribute	SSD (Solid State Drive)	HDD (Hard Disk Drive)
Power Draw / Battery Life	Less power draw, averages 2 – 3 watts, resulting in 30+ minute battery boost	More power draw, averages 6 – 7 watts and therefore uses more battery
Cost	Expensive, roughly $0.20 per gigabyte (based on buying a 1TB drive)	Only around $0.03 per gigabyte, very cheap (buying a 4TB model)
Capacity	Typically not larger than 1TB for notebook size drives; 4TB max for desktops	Typically around 500GB and 2TB maximum for notebook size drives; 10TB max for desktops
Operating System Boot Time	Around 10-13 seconds average bootup time	Around 30-40 seconds average bootup time
Noise	There are no moving parts and as such no sound	Audible clicks and spinning can be heard
Vibration	No vibration as there are no moving parts	The spinning of the platters can sometimes result in vibration
Heat Produced	Lower power draw and no moving parts so little heat is produced	HDD doesn't produce much heat, but it will have a measurable amount more heat than an SSD due to moving parts and higher power draw
Failure Rate	Mean time between failure rate of 2.0 million hours	Mean time between failure rate of 1.5 million hours
File Copy / Write Speed	Generally above 200 MB/s and up to 550 MB/s for cutting edge drives	The range can be anywhere from 50 – 120MB / s
Encryption	Full Disk Encryption (FDE) Supported on some models	Full Disk Encryption (FDE) Supported on some models
File Opening Speed	Up to 30% faster than HDD	Slower than SSD
Magnetism Affected?	An SSD is safe from any effects of magnetism	Magnets can erase data

HDDs Vs. SSDs Vs. SCM

Characteristic	HDD	SSD
Storage Granularity	512 B (or 4KB on new HDDs)	512 KB (Erase-and-write mechanism)
Latency	~2 ms	~50 µs
Bandwidth	~120 MB/s	~300 MB/s
Resilience	∞	~10^6 writes

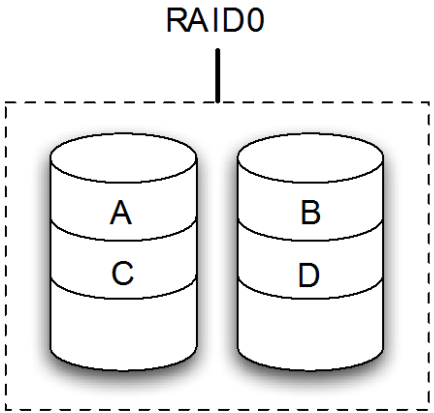
Speicherarrays: RAID

Dresden Database Systems Group

Speicherarrays

- Kopplung billiger Standardplatten unter einem speziellen Controller zu einem einzigen logischen Laufwerk
- Verteilung der Daten auf die verschiedenen physischen Festplatten übernimmt Controller
- zwei gegensätzliche Ziele
  - Erhöhung der Fehlertoleranz (Ausfallsicherheit, Zuverlässigkeit)
  - Effizienzsteigerung durch Parallelität des Zugriffs

Beispiel



Ziele

Dresden Database Systems Group

Erhöhung der Fehlertoleranz

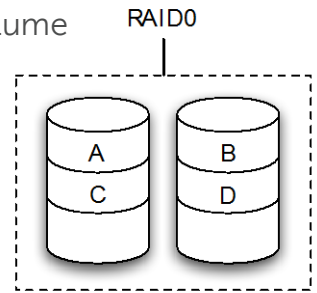
- Nutzung zusätzlicher Platten zur Speicherung von Duplikaten (Spiegeln) der eigentlichen Daten → bei Fehler: Umschalten auf Spiegelplatte
- bestimmte RAID-Levels (1, 0+1) erlauben eine solche Spiegelung
- Alternative: Kontrollinformationen wie Paritätsbits nicht im selben Sektor wie die Originaldaten, sondern auf einer anderen Platte speichern
- RAID-Levels 2 bis 6 stellen durch Paritätsbits oder Error Correcting Codes (ECC) fehlerhafte Daten wieder her
- ein Paritätsbit kann einen Plattenfehler entdecken und bei Kenntnis der fehlerhaften Platte korrigieren

Erhöhung der Effizienz

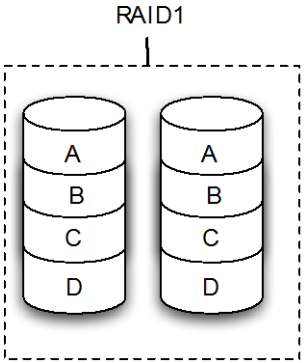
- Datenbank auf mehrere Platten verteilen, die parallel angesteuert werden können → Zugriffszeit auf große Datenmengen verringert sich fast linear mit der Anzahl der verfügbaren Platten
- Verteilung: bit-, byte-, oder blockweise
- höhere RAID-Levels (ab Level 3) verbinden Fehlerkorrektur und block- oder bitweises Verteilen von Daten
- Unterschiede:
  - schnellerer Zugriff auf bestimmte Daten
  - höherer Durchsatz für viele parallel anstehende Transaktionen durch eine Lastbalancierung des Gesamtsystems

RAID-Levels

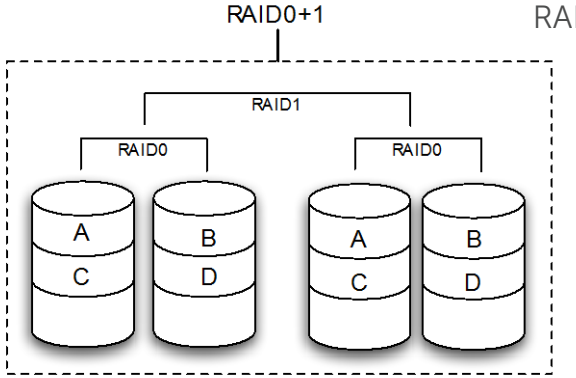
RAID0 → striped volume

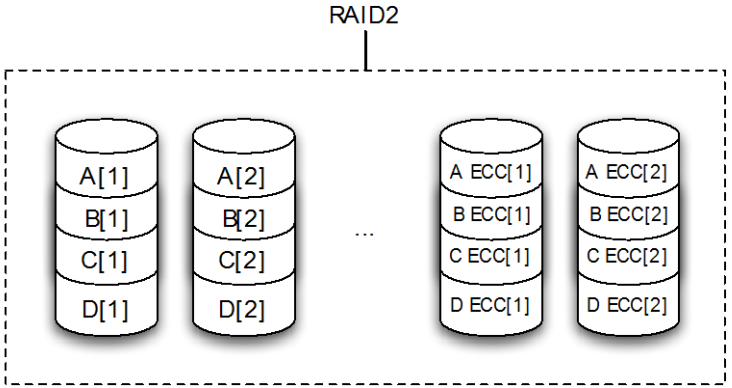


RAID1 → mirror

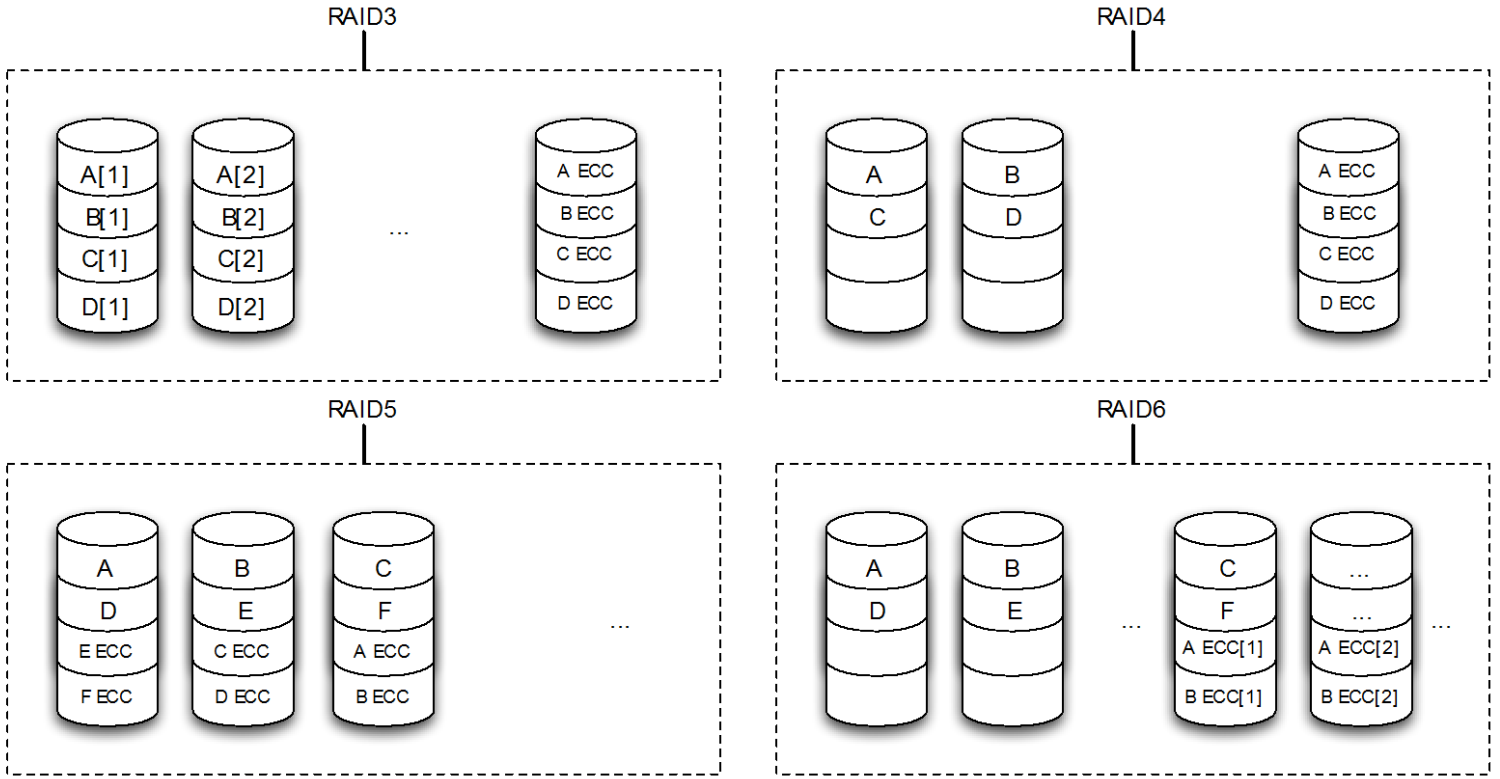


RAID0+1 → striped and mirrored





RAID-Levels (2)



Tertiärspeicher

Dresden Database Systems Group

Sicherungsmedien

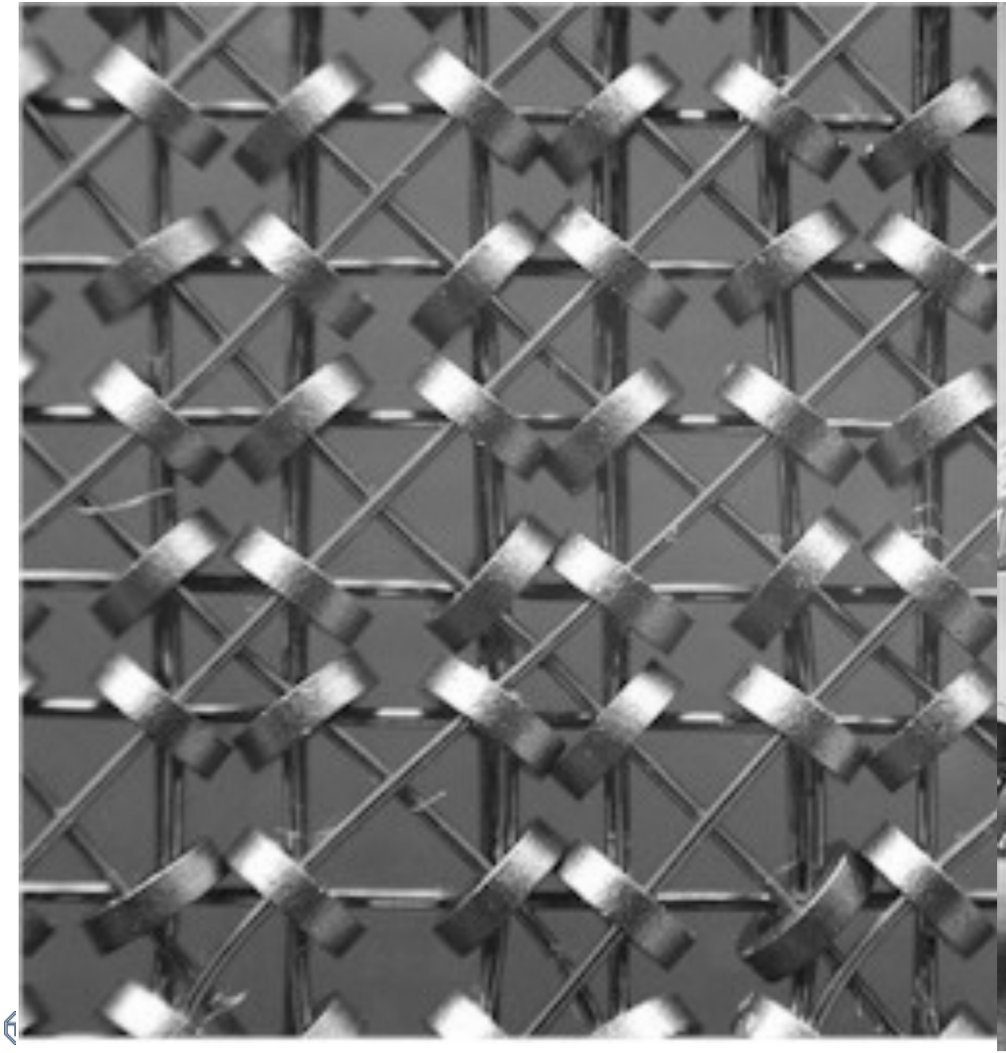
- wenig oft benutzte Teil der Datenbank, die eventuell sehr großen Umfang haben (Text, Multimedia) „billiger" speichern als auf Magnetplatten
- aktuell benutzte Datenbestände zusätzlich sichern (archivieren)
- Tertiärspeicher: Medium austauschbar
  - **offline:** Medien manuell wechseln (optische Platten, Bänder)
  - **nearline:** Medien automatisch wechseln (Jukeboxes, Bandroboter)

Langzeitarchivierung

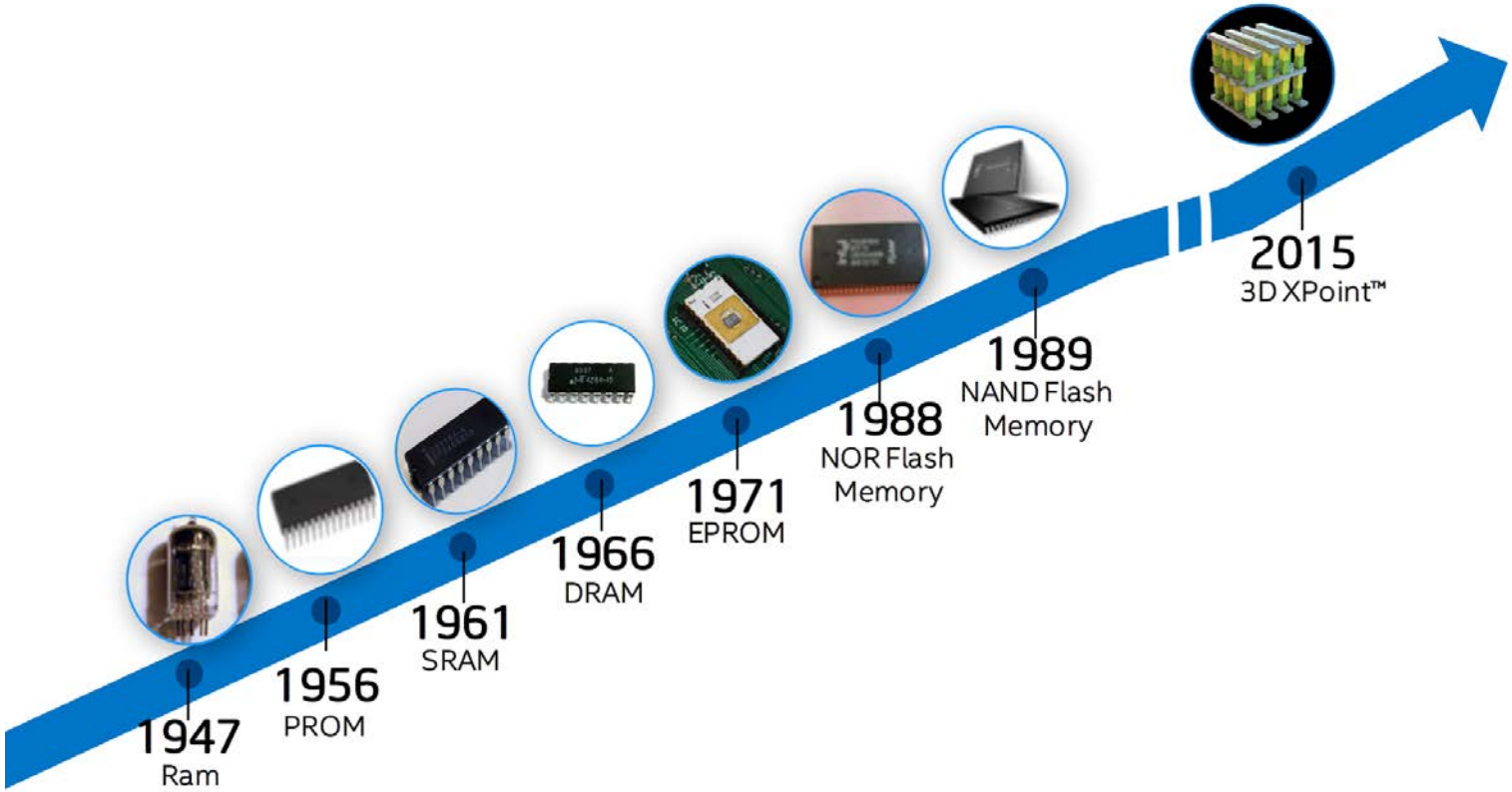
- Lebensdauer, Teilaspekte
  - physische Haltbarkeit des Mediums garantiert die Unversehrtheit der Daten: 10 Jahre für Magnetbänder, 30 Jahre für optische Platten, Papier???
  - Vorhandensein von Geräten und Treibern garantiert die Lesbarkeit von Daten: Geräte für Lochkarten oder 8-Zoll-Disketten
  - zur Verfügung stehende Metadaten garantieren die Interpretierbarkeit der Daten
  - Vorhandensein von Programmen, die auf den Daten arbeiten können, garantieren die Wiederverwendbarkeit von Daten
- Beispiel
  - NCAR's AMSTAR – 30PB, z.B. Klimadaten 180TB pro Monat

Non-Volatile Memory

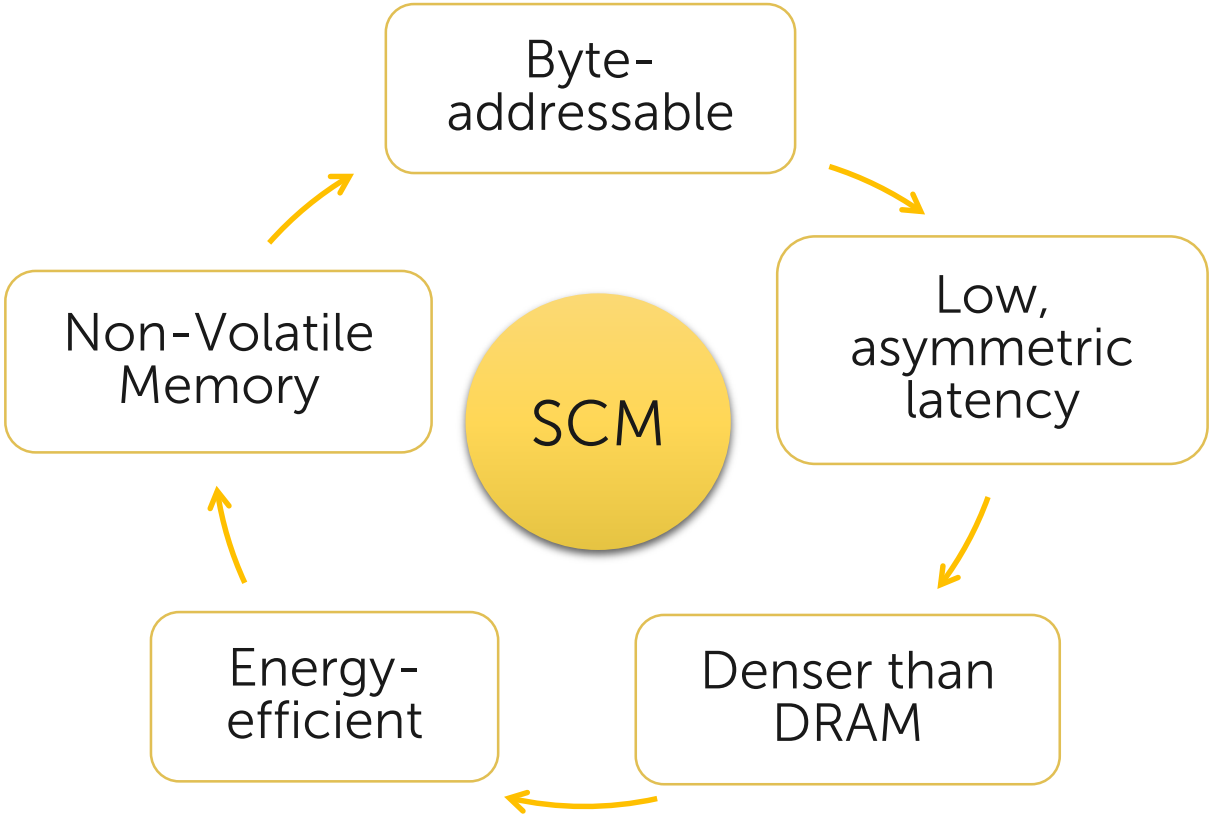
ase



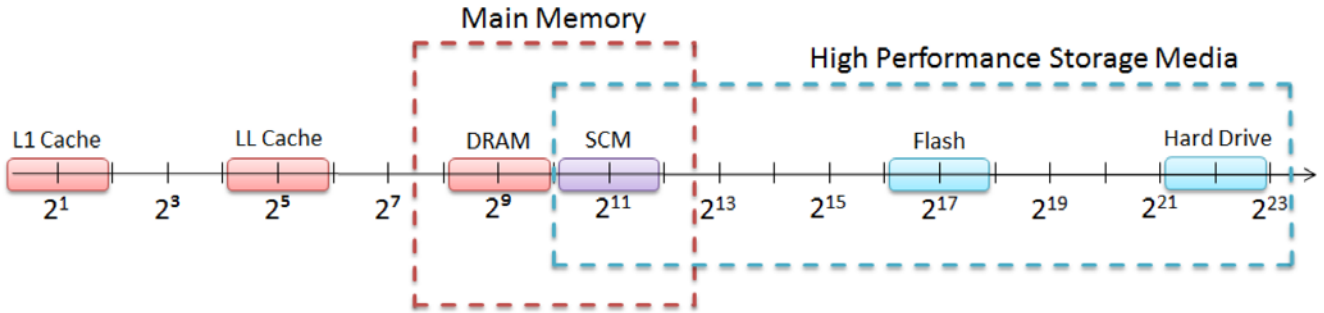
The Memory Timeline



Storage Class Memory



Storage Class Memory / Non-Volatile RAM



Adapted from: M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In ISCA 2009

Emerging Technology

- Examples: MRAM(IBM), FRAM (Ramtron), PCM(Samsung)
- Merging Point between storage and memory

Advantages

- ~4x denser than DRAM
- SCM does not consume energy if not used
- SCM is cheaper, persistent, byte-addressable

Drawbacks

- Number of writes is limited (life expectancy 2-10 years)
- SCM has higher latency than DRAM
  - Read latency ~2x slower than DRAM
  - Write latency ~10x slower than DRAM

HDDs Vs. SSDs Vs. SCM

Dresden Database Systems Group

Characteristic	HDD	SSD	SCM
Storage Granularity	512 B (or 4KB on new HDDs)	512 KB (Erase-and-write mechanism)	64 B
Latency	~2 ms	~50 µs	~200 ns
Bandwidth	~120 MB/s	~300 MB/s	~1 GB/s
Resilience	∞	~10^6 writes	~10^8 writes

SCM can be accessed directly with load/store semantics

SCM Access // File Level

Dresden Database Systems Group

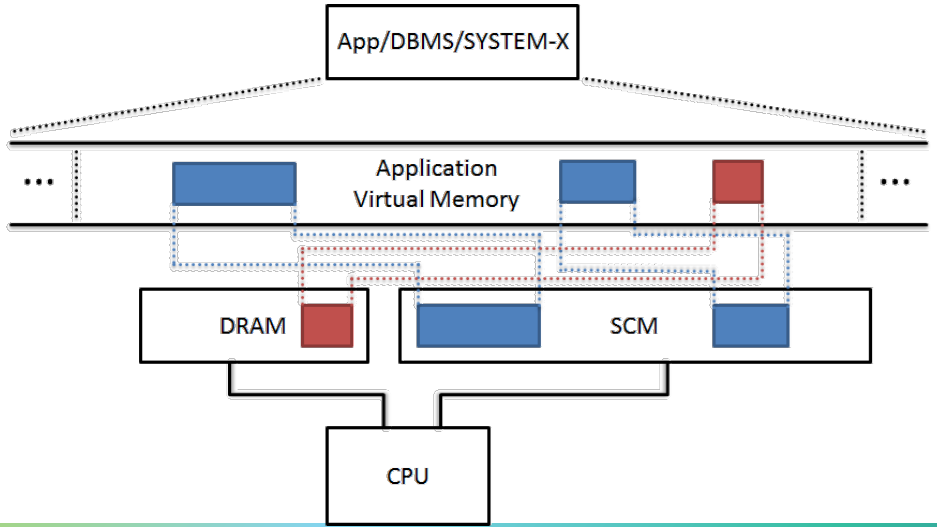
Hybrid Storage Model

- Application may distinguish between
  - Traditional memory
  - Persistent memory blocks
- Files names are used as identification handle

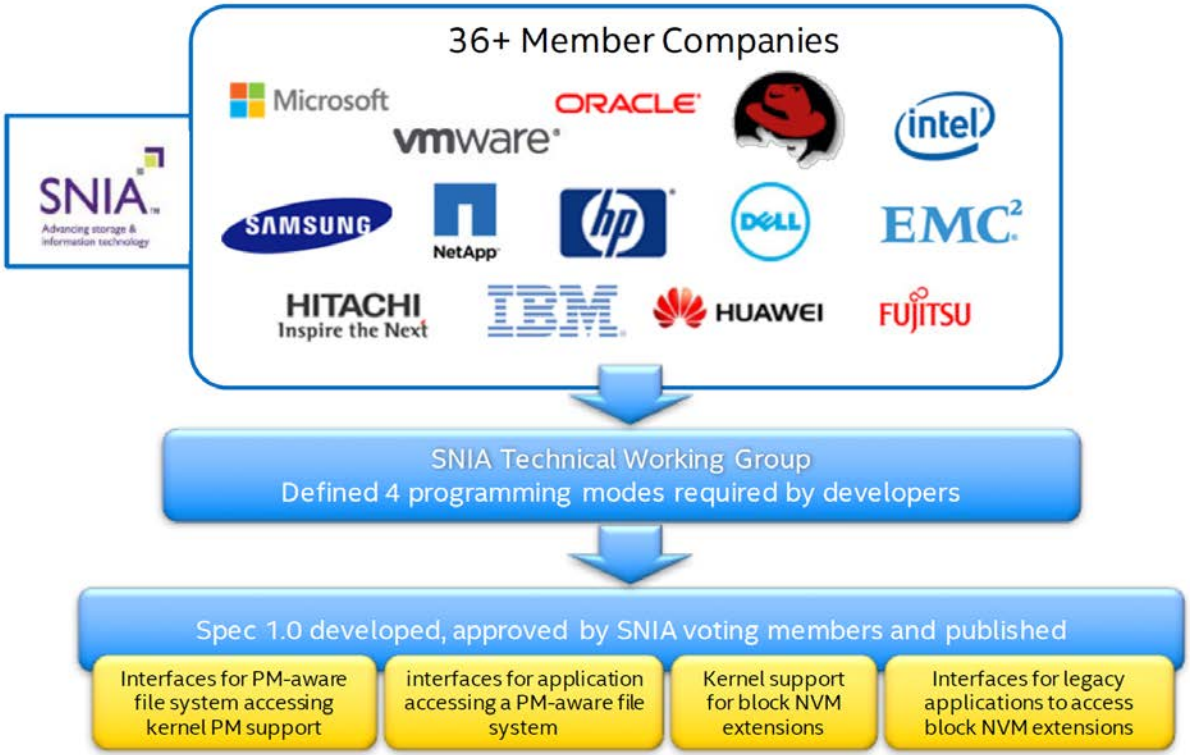
Transparent model

- Application is not aware of different types of memories

... mode is decided during boot time

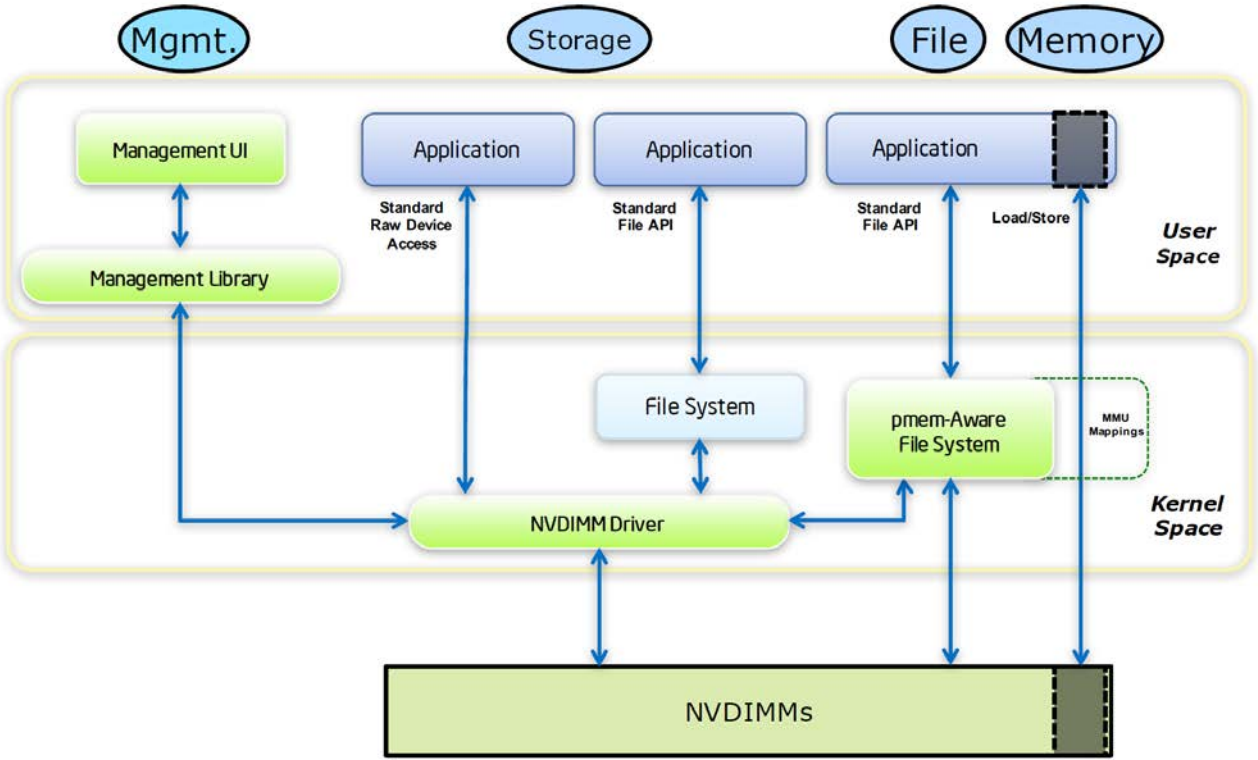


Open NVM Programming Model



http://snia.org/sites/default/files/NVMProgrammingModel_v1.pdf

Software Architecture



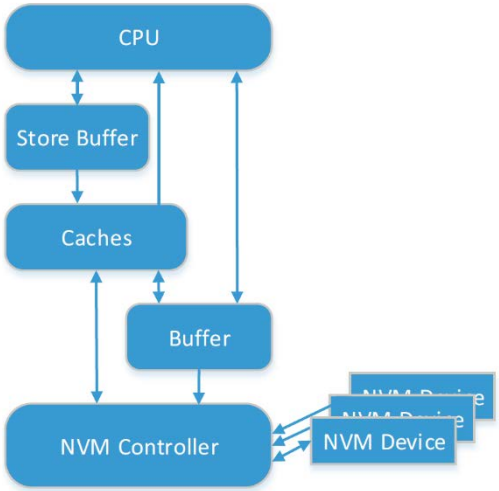
SCM Programming Challenges

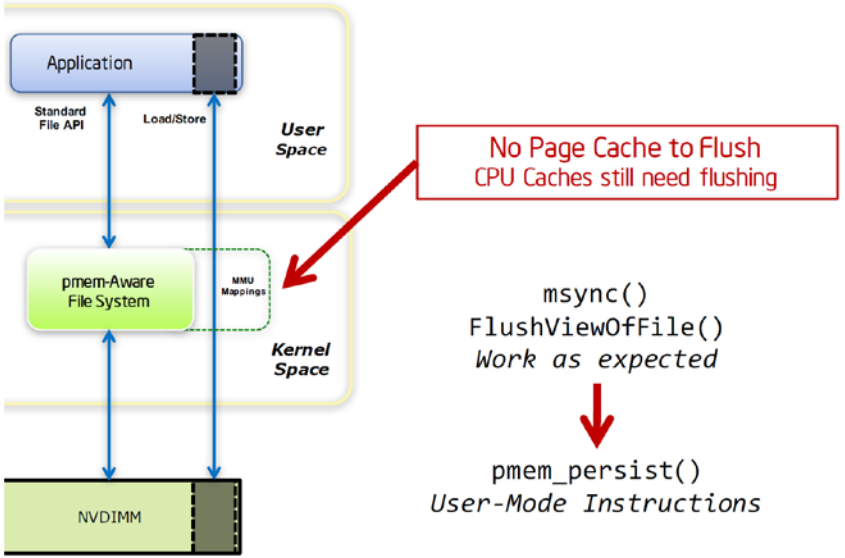
Dresden Database Systems Group

MANY HW COMPONENTS ON THE PATH FROM CPU TO FINAL STORAGE

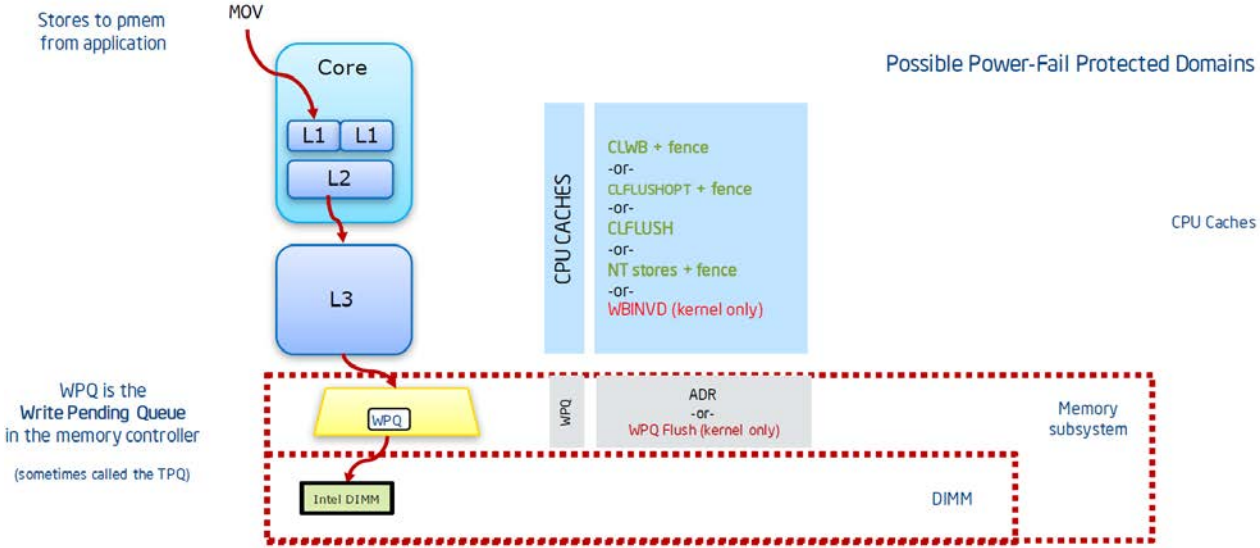
- Software has little control over when data is persisted in SCM

→ Need to enforce the order of SCM memory operations using CLFLUSH, MFENCE, SFENCE, etc.





Power-fail Protected Domains



- Do applications need to comprehend CPU CACHE flushing?
  - Yes, flushing CPU caches is required for everything but custom platforms
  - Perhaps some future generation will flush CPU caches on power failure
- Do applications need to comprehend WPQ Flush?
  - No, current NVDIMM-N ecosystem does not
  - Stores are considered persistent without WPQ Flush
  - Kernel can use WPQ Flush for special cases like register writes
- The NVM Libraries (NVML) provide APIs that follow these rules

Programming Model Summary

Dresden Database Systems Group

Direct access to pmem exposed as *memory-mapped files*

Semantics of memory-mapped files work as expected

- Stores are guaranteed persistent when sync API is called
  - msync+MS_SYNC/fsync or FlushViewOfFile+FlushFileBuffers

Only when OS says it is safe:

- Stores to already mapped pmem may be flushed from user space
  - CLFLUSH or CLFLUSHOPT+fence or CLWB+fence or NTstore+fence
- The way OS says it is safe is OS-specific
  - NVML provides pmem_is_pmem(), defined to be true for this case only
- Any other changes, like file attributes, can't use this (use API instead)

Only when OS says platforms supports it (future use):

- CPU caches are part of the persistence domain
  - Any store to pmem can be considered persistent as soon as it is visible
- The way OS says platform supports it is OS-specific
  - NVML's pmem_persist() will flush/not flush CPU caches appropriately
- Standard API may flush to smaller failure domain than optimized flush

```
open(…);
mmap(…);

strcpy(pmem, "andy rudoff");

pmem_persist(pmem, 12);
```

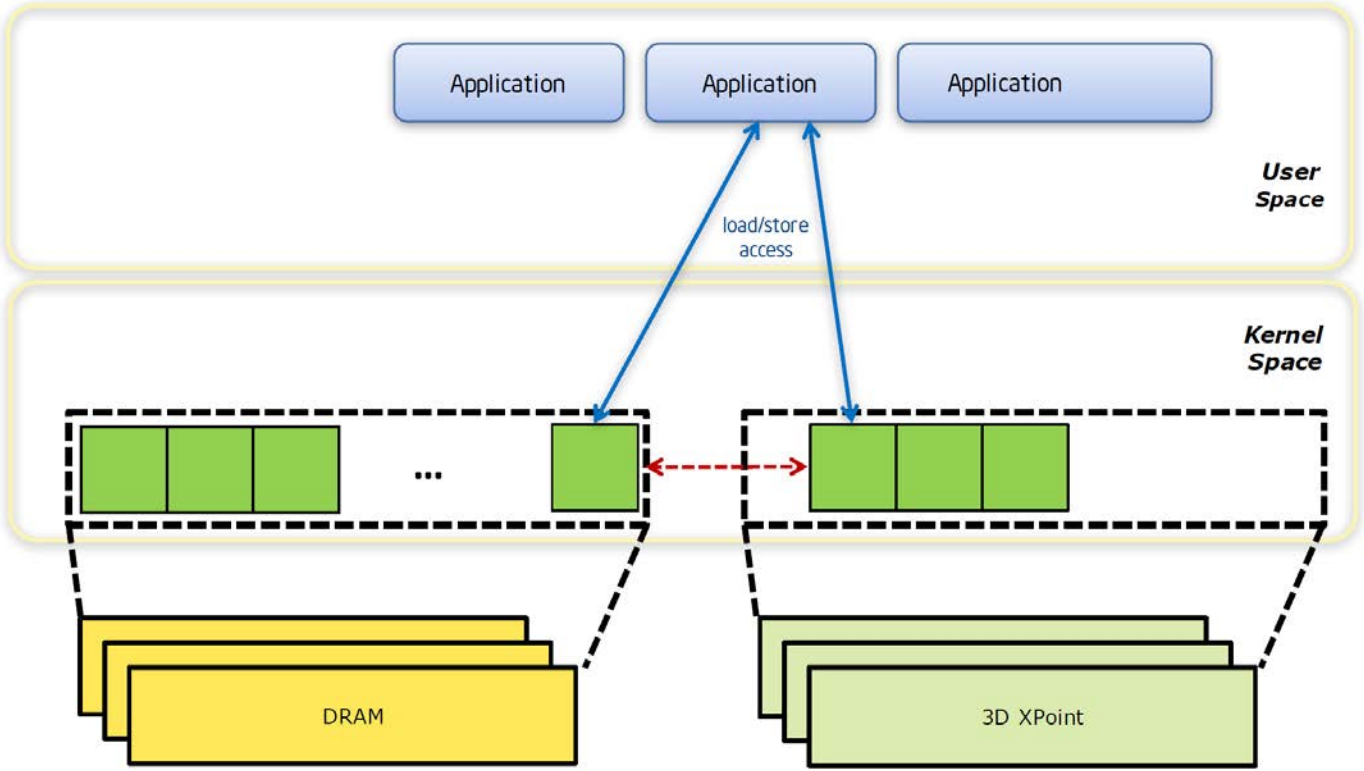
crash

Which Result?

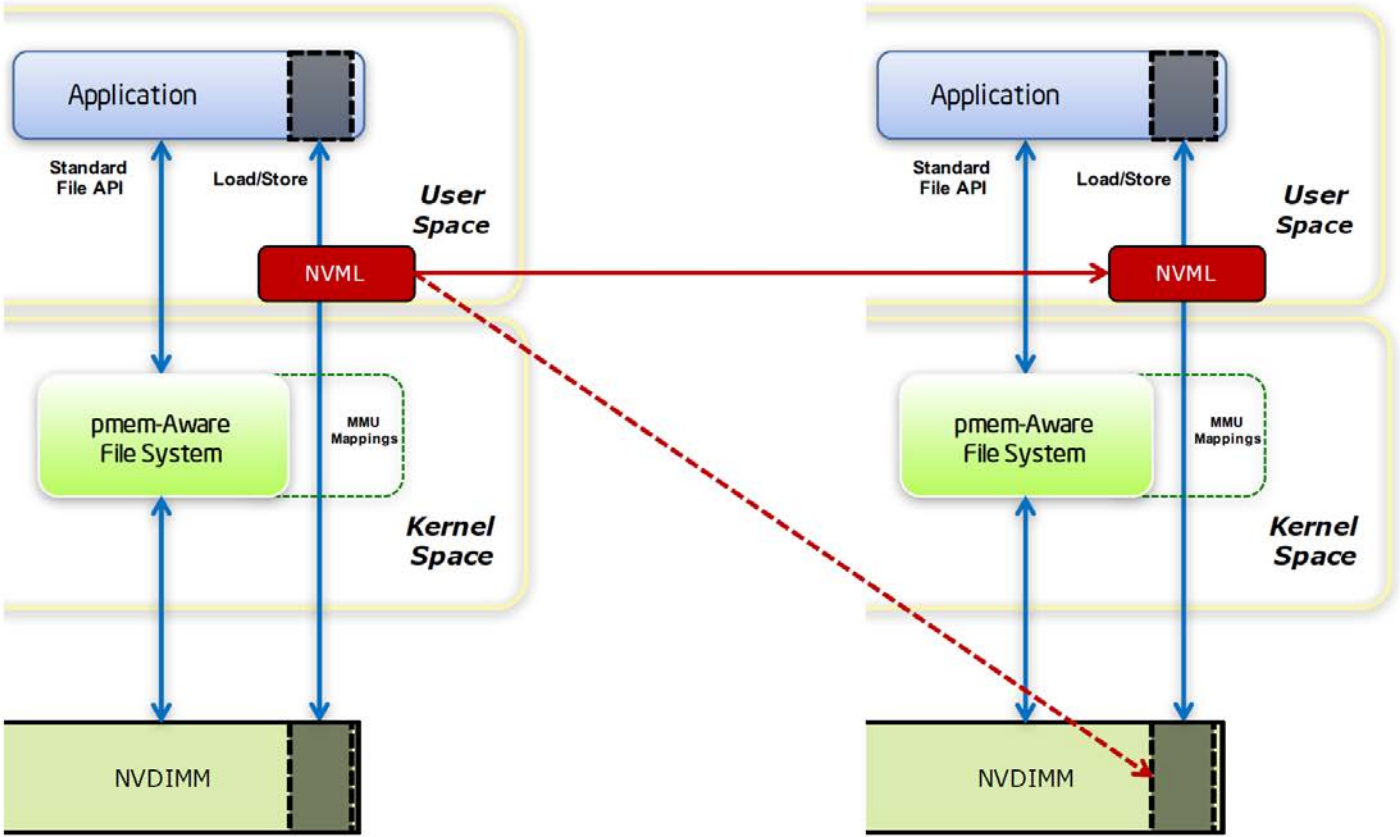
1. "\0\0\0\0\0\0\0\0…"
2. "andy\0\0\0\0\0\0…"
3. "andy rud\0\0\0\0\0…"
4. "\0\0\0\0\0\0\0off\0\0..."
5. "andy rudoff\0"

Advanced pmem Memory Management

Volatile use case for pmem in kernel



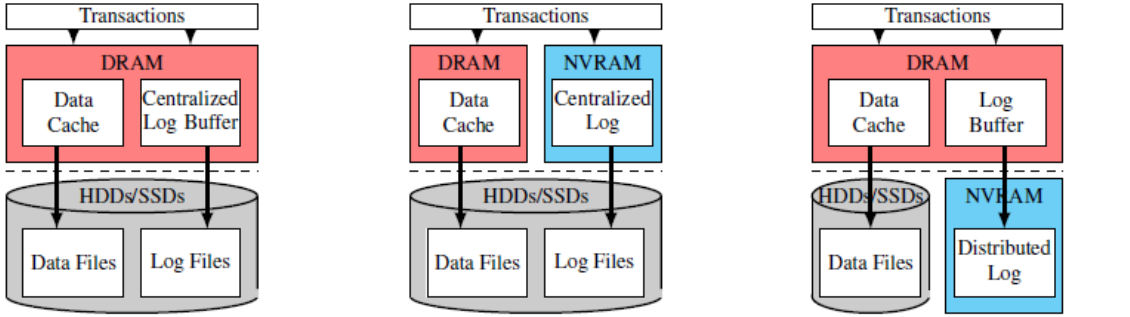
Replication: The Tip of the Iceberg



Potential Database Evolution/Revolution?

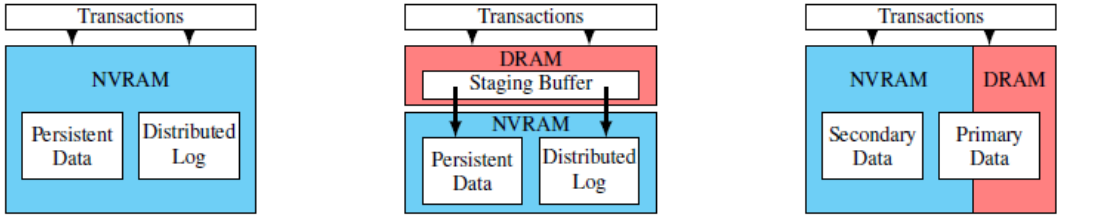
Dresden Database Systems Group

DRAM is hitting costs
and power consumption limits!



(a) Traditional Architecture (b) Log disk-replacement. (c) Distributed log on NVRAM.

Figure 2: Improving database traditional architecture with NVRAM.



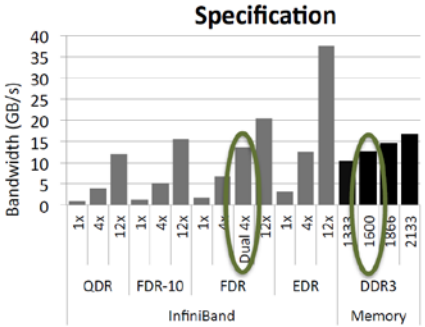
(a) In-Place Updates (b) NVRAM-Group Commit (c) Hybrid NVRAM-DRAM.

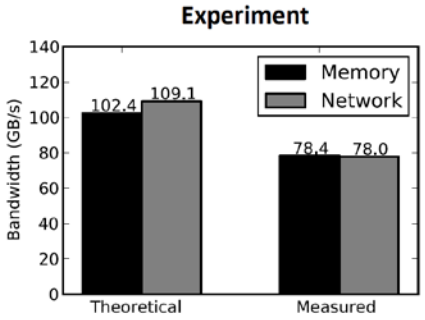
Figure 3: NVRAM-enabled database architectures.

Fast Networks: Infiniband & RDMA

Dresden Database Systems Group

Network vs. Memory Bandwidth:





Machine: 2 Sockets, 4 NICs

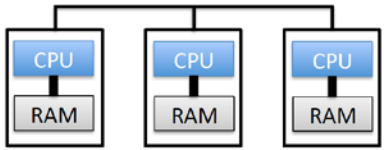
⇒ Network bandwidth is not a bottleneck anymore (Latency is still 10x higher for remote access)

TIME TO RE-THINK CLUSTER-BASED ARCHITECTURES

- Requires to redesign traditional/mature algorithms, e.g. joins → Example 1
- Allows to devise completely new database scenarios, e.g. in cloud settings → Example 2

Distributed Main-Memory Architectures: From Slow to Fast Networks

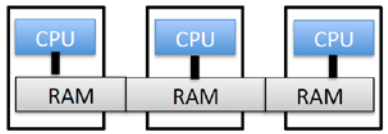
Slow Networks:



Main Goal: Minimize Network Cost

- **Co-Partition Data** but not always possible (ad-hoc workloads, shared state)
- **Communication-avoiding** Algorithms (e.g., semi- or bloom-joins)

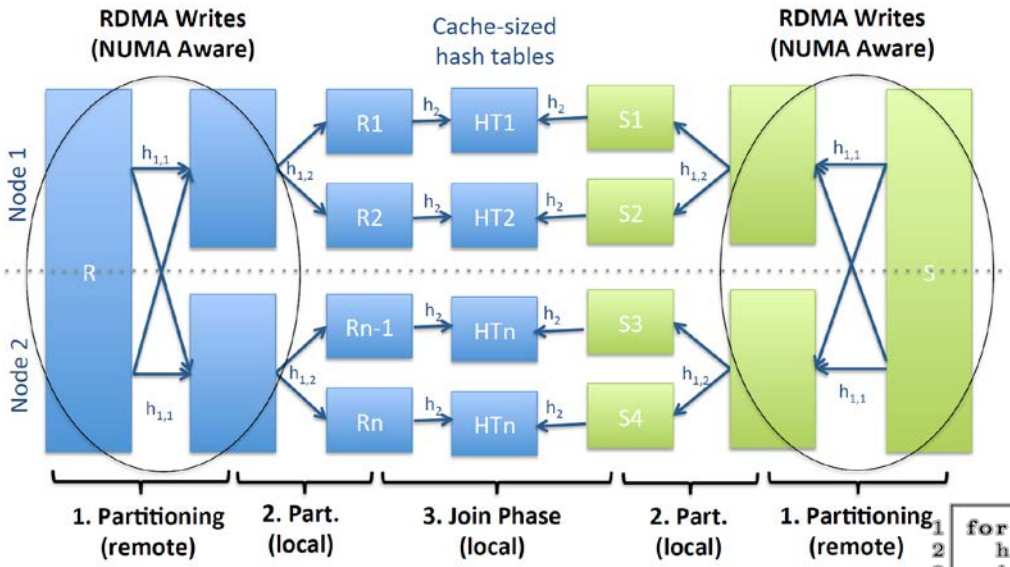
Fast Networks + RDMA:

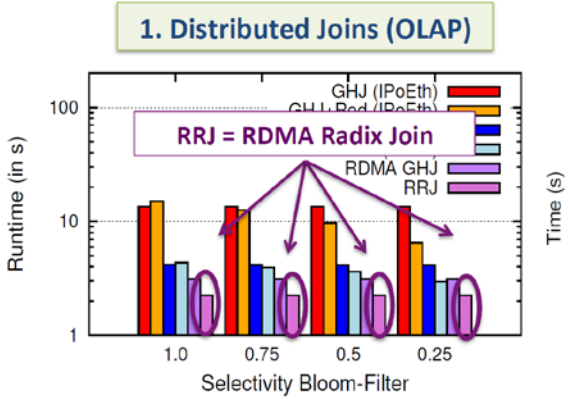


Main Goal: CPU / Cache Efficiency

- **Distributed NUMA-aware and Cache-efficient** Algorithms
- Hide Latency using RDMA: **Overlap Communication + Computation**

Example 1: Radix Join





Workload:

- **Joins:** Classic Dist. Join (Slow Net) | Classic Dist. Join (Fast Net) | RDMA Joins (Fast Net)
- **Data:** 512M records per table x 2 on 4 servers (1Gb Ethernet + FDR 4x IB)

Listing 1: Remote Radix-Partitioning

```
1  for each tuple r in R do{
2    h = radix-hash(key(r));
3    buffer[h].append(r);
4
5    if(buffer.isFull()){
6      copy_counter[h]++;
7      if(partition[h] is local){
8        memcpy(buffer[h], partition[h]);
9      }
10     else{
11       signalled = (counter[h]==N);
12       RDMA_WRITE(buffer[h], partition[h], signalled);
13     }
14     buffer.emtpy();
15   }
16 }
```

Example 2: Memory Extensions

Dresden Database Systems Group

Accelerating Relational Databases by Leveraging Remote Memory and RDMA

Feng Li Sudipto Das Manoj Syamala Vivek R. Narasayya

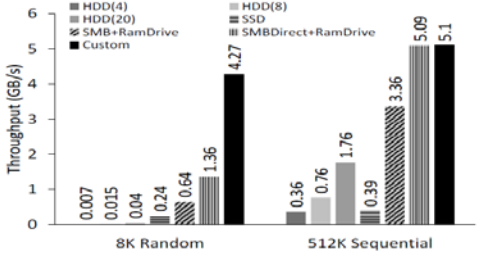


Figure 3: I/O micro benchmark throughput.

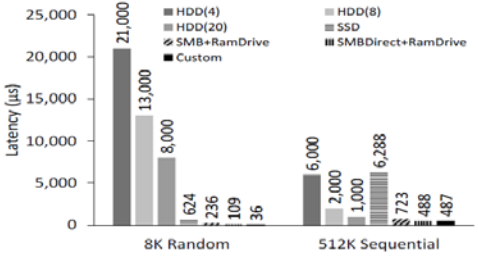


Figure 4: I/O micro benchmark latency.

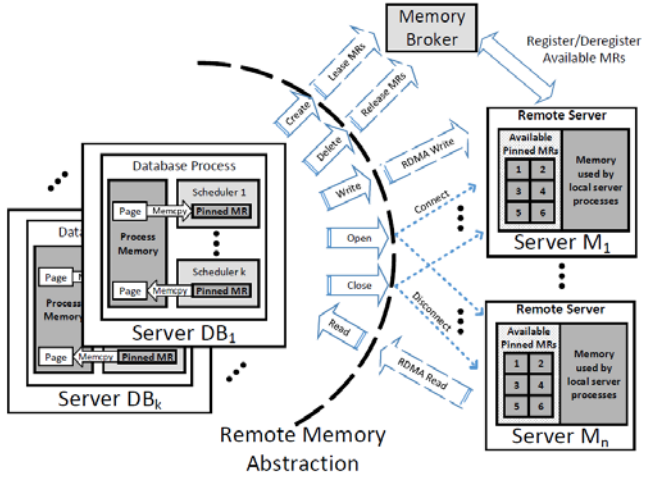
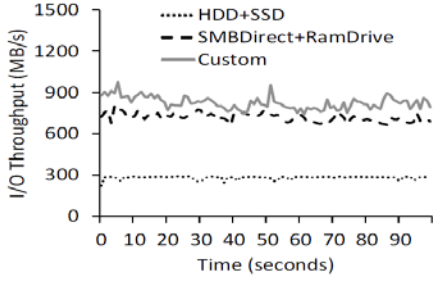


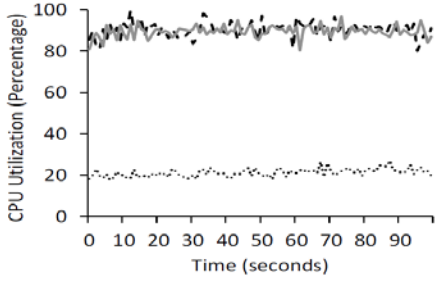
Figure 1: Integrating remote memory into an RDBMS.

Table 5: Design alternatives evaluated.

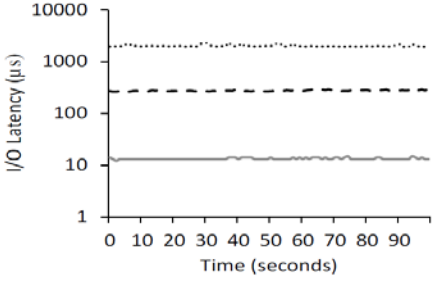
Designs	Data Files	TempDB	BP Ext	Protocol
HDD	HDD	HDD	N/A	N/A
HDD+SSD	HDD	SSD	SSD	N/A
SMB+ RamDrive	HDD	Remote Memory	Remote Memory	SMB
SMBDirect +RamDrive	HDD	Remote Memory	Remote Memory	SMB Direct
Custom	HDD	Remote Memory	Remote Memory	NDSPI
Local Memory	HDD	SSD	N/A	N/A



(a) I/O throughput



(b) CPU utilization



(c) I/O latency

Blöcke und Seiten

Speichersystems eines DBMS

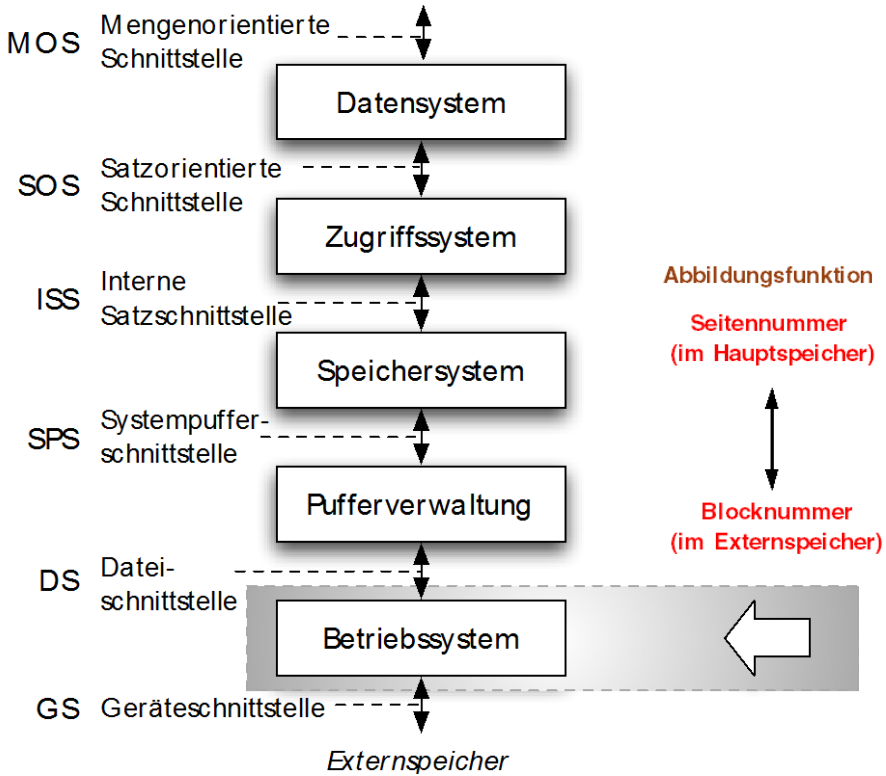
Dresden Database Systems Group

Abstraktion von Speicherungs- oder Sicherungsmedium

- Modell: Folge von Blöcken

Aufgaben

- Bereitstellen eines linearen, logischen, potentiell unendlichen, aktuell aber endlichen Adressraumes (Segment) mit sichtbaren Seitengrenzen
- freie Adressierbarkeit innerhalb eines Segmentes wie in einem virtuellen Adressraum
- Abbildung von Seiten aus Segmenten auf physische Blöcke von Dateien
- Pufferverwaltung i.e.S., d.h. Auswahl einer zu ersetzenden Seite, falls Puffer voll, und Anforderung der gewünschten Seiten



Dateisystems

Dresden Database Systems Group

Dienste des Dateisystems

- Allokation oder Deallokation von Speicherplatz
- Holen oder Speichern von Seiteninhalten
- Allokation möglichst so, dass logisch aufeinanderfolgende Datenbereiche (etwa einer Relation) auch möglichst in aufeinanderfolgenden Blöcken der Platte gespeichert werden
- Nach vielen Update-Operationen: Reorganisationsmethoden
- Freispeicherwaltung

Abbildung der Datenstrukturen

- Abbildung der konzeptuellen Ebene auf interne Datenstrukturen
- Unterstützung durch Metadaten (im Data Dictionary, etwa das interne Schema)

Betriebssystemdateien

Dresden Database Systems Group

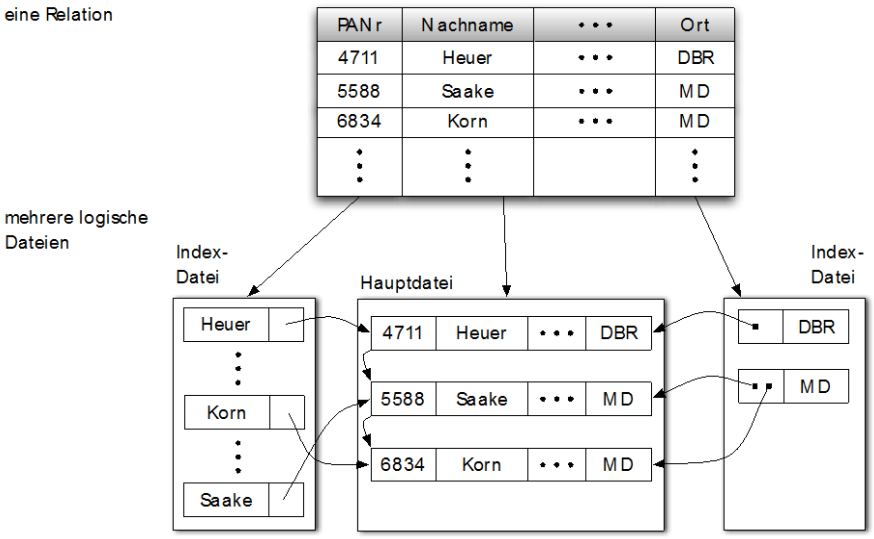
Alternativen

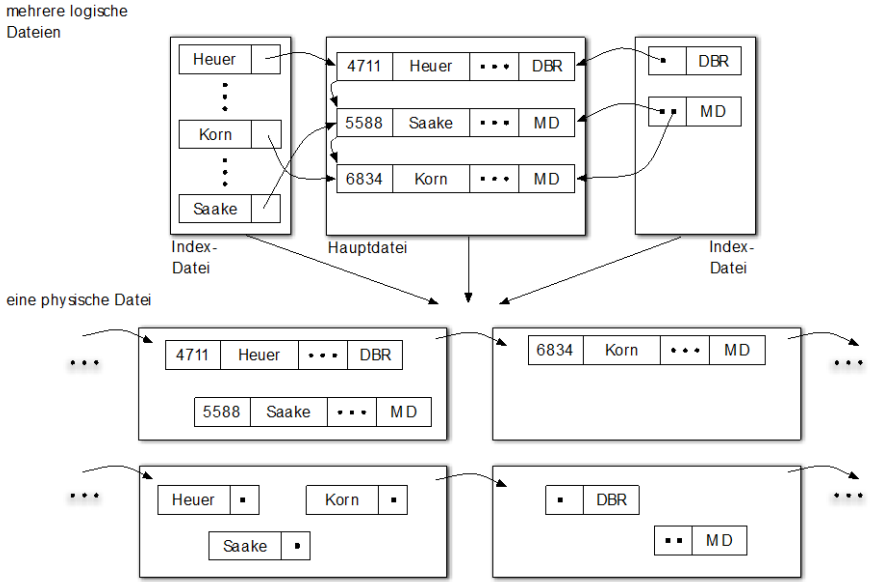
- jede Relation oder jeder Zugriffspfad in genau einer Betriebssystem-Datei
- ein oder mehrere BS-Dateien, DBMS verwaltet Relationen und Zugriffspfade selbst innerhalb dieser Dateien
- DBMS steuert selbst Magnetplatte an und arbeitet mit den Blöcken in ihrer Ursprungsform (raw device)

Warum nicht immer BS-Dateiverwaltung?

- Betriebssystemunabhängigkeit
- in 32-Bit-Betriebssystemen: Dateigröße 4GB maximal
- BS-Dateien auf maximal einem Medium
- betriebssystemseitige Pufferverwaltung von Blöcken des Sekundärspeichers im Hauptspeicher genügt nicht den Anforderungen des Betriebssystems

Übliche Form der Speicherung





Block/Seite/Segment

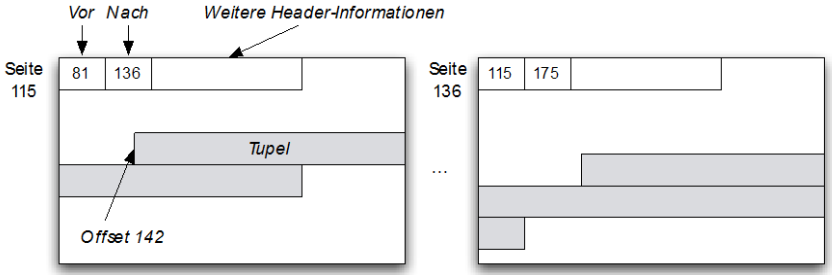
Dresden Database Systems Group

Block

- kleinste adressierbare Einheit auf Externspeicher
- Zuordnung zu Seiten im Hauptspeicher
  - festen Größenverhältnis: 1, 2, 4 oder 8 Blöcke → Seite
  - hier „ein Block --- eine Seite“

Seite

- Verwaltung im Hauptspeicher
- höhere Schichten des DBMS adressieren über Seitennummer
- Organisation der Seiten: doppelt verkettet Listen



Segmente (Tablespaces)

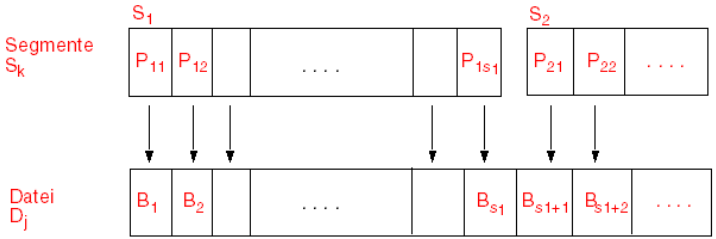
- Einheiten des Sperrens, der Recovery und der Zugriffskontrolle
- Verhältnis von Segment zu Seite → **1:N**

- Verhältnis Segment zu Datei: **sehr systemabhängig**
  - 1:1 alle Seiten eines Segmentes werden in Blöcken einer Datei gespeichert
  - N:1 Seite mehrerer Segmente werden in Blöcken einer Datei gespeichert
  - 1:N Seiten eines Segmentes werden in Blöcken unterschiedlicher Dateien gespeichert.

Seitenabbildung

Direkte Seitenadressierung

- aufeinanderfolgende Seiten (bezgl. des virtuellen Adressraumes) werden auf aufeinanderfolgende Blöcke einer Datei abgebildet

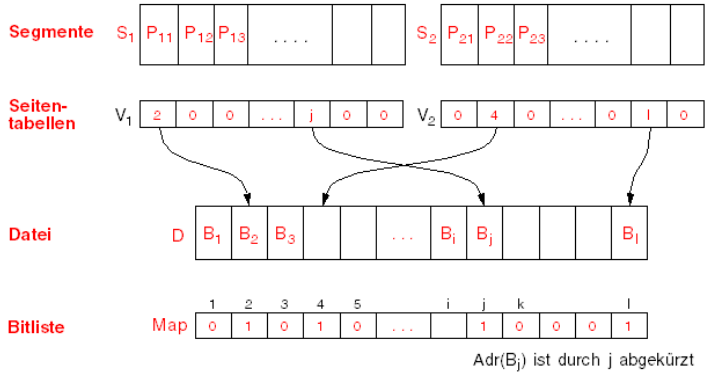


Eigenschaften

- keine Fragmentierungsprobleme wegen der geforderten Übereinstimmung von Seiten- und Blockgröße (Lk = Lj)
- jede Seite Pki ∈ Sk wird genau ein Block Bjl ∈ Dj zugeordnet

Indirekte Seitenadressierung

- Maximum an Flexibilität bei der Blockzuordnung
- benötigt zwei Hilfsstrukturen



- Für jedes Segment Sk existiert eine Seitentabelle Vk, die für jede Seite einen Eintrag (4 Bytes) mit der aktuellen Blockzuordnung besitzt.
- Für die Datei D existiert eine Bitliste die ihre aktuelle Belegung beschreibt, d. h., die für jeden Block angibt, ob er momentan eine Seite enthält oder nicht.

Gründe für die Sichtbarkeit von Seitengrenzen

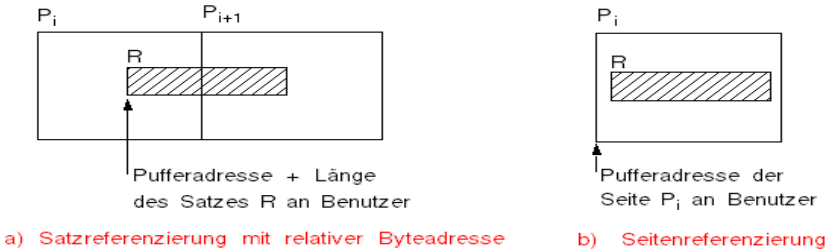
Dresden Database Systems Group

Vergleich mit "Virtueller Hauptspeicher aus BS"

- Virt. HS: keine Kachelgrenzen zu den Anwendungsprogrammen hin sichtbar.
- Frage: Warum Erhaltung der Seitengrenzen für ein DBS?

"Spanned Record Facility"

- benachbarte Seiten müssen im DB-Puffer benachbart angeordnet sein
- resultiert in erheblichen Abbildungs- und Ersetzungsproblemen



a) Satzreferenzierung mit relativer Byteadresse

b) Seitenreferenzierung

- deshalb: sichtbare Seitengrenzen an der DB-Speicherschnittstelle

Einbringstrategien

Einbringstrategien für Änderungen

Dresden Database Systems Group

Einbringen = Ablegen auf einem nicht-flüchtigen Speicher ("materialisieren")

Grundlegende Idee

- Speichern muss nicht notwendigerweise identisch mit dem Einbringen einer Seite in den Datenbestand sein !
- Einbringen kann verzögert stattfinden

Zeitpunkt des Einbringens im Kontext einer Transaktion

- vor einem Commit ("steal" versus "no steal")
- nach einem Commit ("force" versus "no force")

Eine Einbringstrategie legt fest,

- in welchen Block eine modifizierte Seite abgespeichert wird -> **Seitenzuordnung**

Unterscheidung in

- direkte Seitenzuordnung
- indirekte Seitenzuordnung

Abhängigkeiten zur Ersetzungsstrategie

Dresden Database Systems Group

... vor COMMIT Zeitpunkt

- STEAL
  - geänderte Seiten können jederzeit, insbesondere vor EOT der ändernden Transaktion, ersetzt und in die Datenbank eingebracht werden
  - Flexibilität mit Blick auf Seitenersetzung
  - Transaktionsfehler können in schmutzigen Seiten münden → UNDO-Recovery
- NOSTEAL (¬ STEAL)
  - Seiten mit schmutzigen Änderungen dürfen nicht ersetzt werden
  - keine UNDO-Recovery auf der materialisierten DB vorzusehen
  - Probleme bei langen Änderungstransaktionen

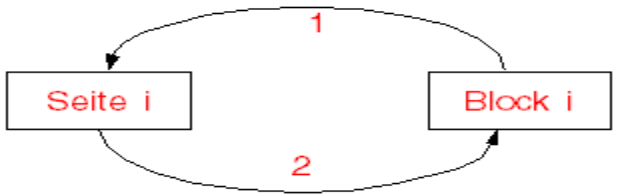
... nach COMMIT Zeitpunkt

- FORCE
  - alle geänderten Seiten werden spätestens im Zuge der Commit-Behandlung in die materialisierte Datenbank eingebracht
  - Hoher punktueller Schreibaufwand aber keine REDO-Recovery notwendig
- NOFORCE (¬ FORCE)
  - Geänderte Seiten dürfen über den Commit-Zeitpunkt im Hauptspeicher verbleiben
  - Nachziehen festgeschriebener Änderungen notwendig → REDO-Recovery

Arten von Einbringstrategien

Dresden Database Systems Group

Direkte Seitenzuordnung



1 Lesen vor Änderung
2 Schreiben nach Änderung

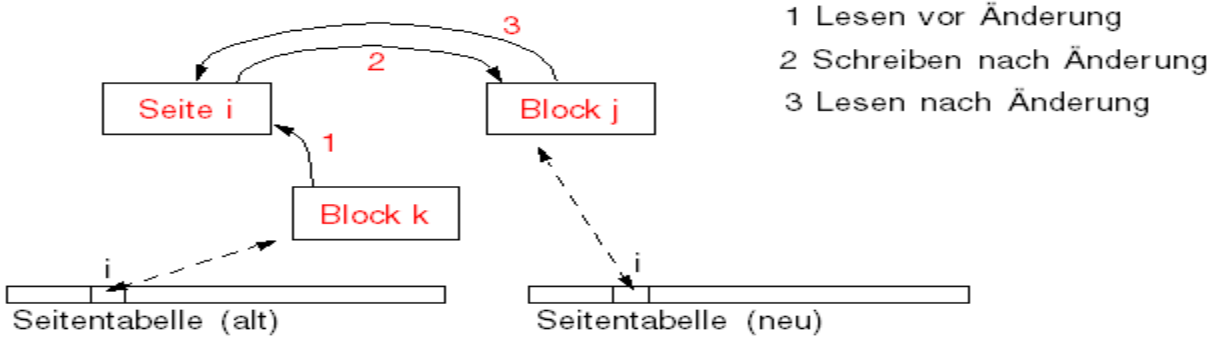
- Beim Ausschreiben aus dem Hauptspeicher **ersetzt** eine Seite genau den Block, von dem sie beim Einlagern in den Puffer gelesen wurde ("Update-in-place").
- **Vorteil:** Einfachheit; immer nur ein Block pro Seite.
- **Nachteil:** Keine Unterstützung der Recovery.
  Der alte Zustand der Seite muss als Log-Information **vor** dem Einbringen der geänderten Seite auf einen sicheren Speicher geschrieben werden
  ("**W**rite-**A**head-**L**og," WAL-Prinzip).

Indirekte Seitenzuordnung

Dresden Database Systems Group

Indirekte Seitenzuordnung

- Beim Ausschreiben aus dem Hauptspeicher wird eine Seite in einen freien Block geschrieben. Der ursprüngliche Block bleibt unverändert.
- Auch nach einem Hauptspeicherfehler ist eine konsistente Datenbank in den alten Blöcken erhalten!



Schattenspeicherkonzept

Dresden Database Systems Group

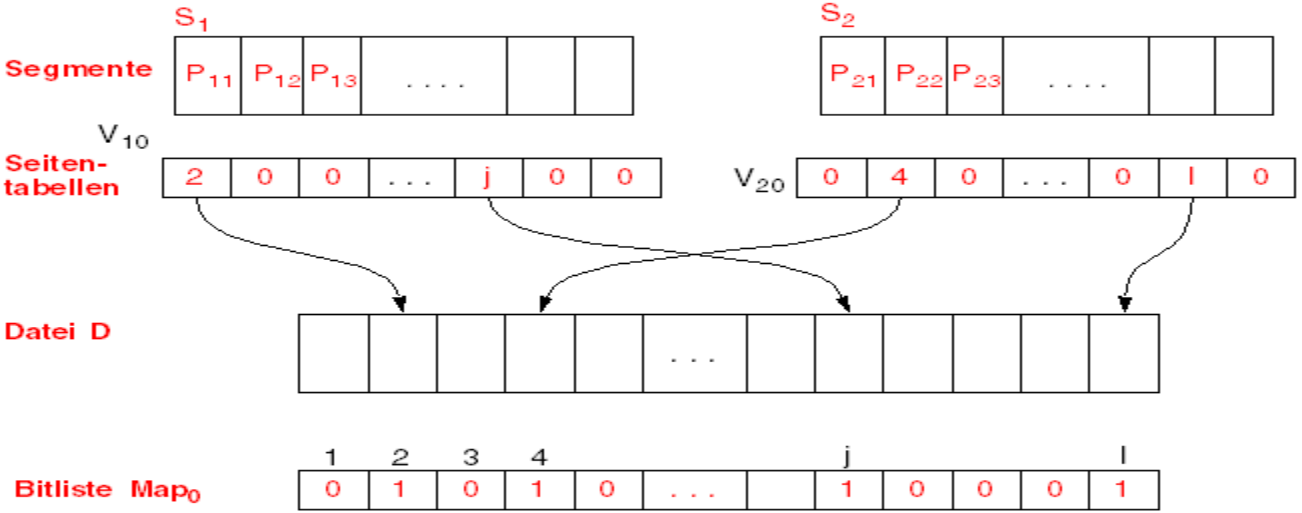
Beispiel einer indirekten Seitenzuordnung

- Verwendung: fehlertolerante Systeme
- Realisiert im System R, weiterentwickelt im TOSP (Transaktions-orientiertes Schattenspeicherkonzept)

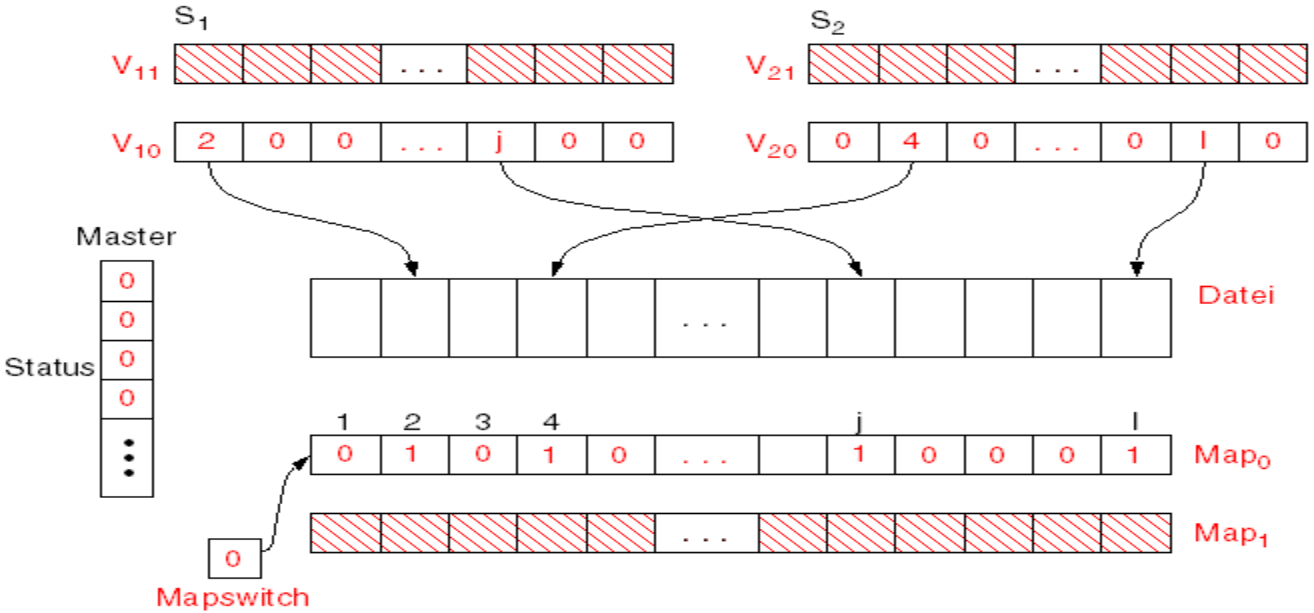
Grundidee

- Inhalte aller Seiten eines Segments werden in einem Sicherungsintervall D_t (z.B. 5min) in einem speicherkonsistenten Zustand unverändert gehalten
- Sicherungspunkte sind segmentorientiert (nicht transaktionsorientiert)
- Ist ein Segment geschlossen, erfüllt sein Inhalt bestimmte, von höheren Schichten kontrollierte Konsistenzbedingungen
- Auf einer Blockmenge D können gleichzeitig mehrere Segmente für den Änderungsbetrieb geöffnet werden
- Wenn in einem Segment S_i mit den Seiten P_{ij} $(1 \leq j \leq s_i)$ insgesamt $h_i \leq s_i$ belegt sind, so gilt für den Speicherbedarf z: $h_i \leq z \leq 2h_i$

Indirekte Seitenadressierung



Datenstrukturen im Schattenspeicherkonzept



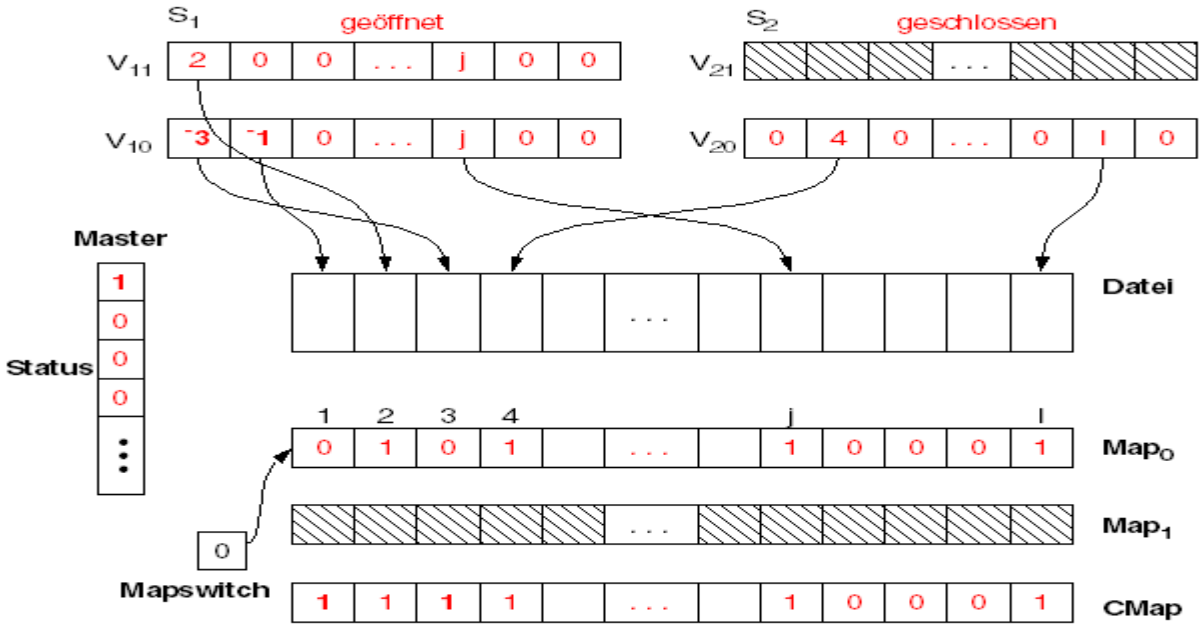
Datenstrukturen im Schattenspeicherkonzept (2)

Dresden Database Systems Group

Erläuterung

- schraffierte Strukturen
  - in der Ausgangssituation nicht benutzte Speicherbereiche
  - werden erst nach der Eröffnung für den Änderungsbetrieb erforderlich
- Status (i)
  - enthält den Eröffnungszustand für Segment i (zu Beginn: alle Segmente geschlossen)
- MAPSWITCH
  - zeigt an, welcher der beiden (gleichberechtigten) Freispeichertabellen Map_0 und Map_1 das aktuelle Verzeichnis belegter Blöcke enthält

Änderungsbetrieb im Schattenspeicherkonzept



Funktionsprinzip des Schattenspeicherkonzepts

Dresden Database Systems Group

Eröffnung eines Segmentes k für Änderungen

- kopiere V_{k0} nach V_{k1} // Anlegen der Schattenseitenzuordnungstabelle
- STATUS(k) := 1
- Schreibe MASTER in einer ununterbrechbaren Operation aus
- Lege im Hauptspeicher eine Arbeitskopie CMAP von MAP_0 an

Erstmalige Änderung einer Seite P_I

(erstmalig seit Eröffnung des Segments)

- 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, d.h. einer Seite wird nur bei der erstmaligen Änderung nach Eröffnen des Segmentes ein neuer Block zugewiesen.

Funktionsprinzip des Schattenspeicherkonzepts

Dresden Database Systems Group

Beenden eines Änderungsintervalls

- Erzeuge die Bitliste mit der aktuellen Speicherbelegung in MAP_1 (neue Blöcke belegt, alte Blöcke freigegeben)
  - Merke:
    - Blöcke mit neuen/modifizierten Seiten können durch den Änderungsvermerk aufgefunden werden
    - Die Freigabe der alten Blöcke kann erst zu diesem Zeitpunkt erfolgen, da sonst die alten Blöcke vor dem Einbringen der neuen Version mit anderen Inhalten überschrieben werden könnten.
- Schreibe MAP_1 (kein Überschreiben von MAP_0)
- Schreibe V_{k0}
- Schreibe alle geänderten Blöcke
- STATUS(k) := 0, MAPSWITCH = 1 (MAP_1 ist aktuell)
- Schreibe MASTER in einer ununterbrechbaren Operation aus.

Merke

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

Bewertung Schattenspeicherkonzept

Dresden Database Systems Group

Vorteile für Recovery

- Rücksetzen auf den letzten konsistenten Zustand ist einfach und billig
- Protokollieren des Zustandes vor einer Änderung (WAL-Protokoll) kann vermieden werden (flexibleres Schreiben der Log-Files)
- logisches Protokollieren ist möglich, da stets operationskonsistenter Zustand verfügbar ist

Nachteile

- Hilfsstrukturen (Seitentabellen V und Bittabellen M) werden so groß, dass sie in Blöcke zerlegt und durch einen speziellen Ersetzungsalgorithmus in einem eigenen Puffer verwaltet werden müssen
  (Seitentabellen belegen etwa 0.1-0.2% der gesamten Datenbank)
- zusätzlicher Speicherplatz für die Doppelbelegung
- physische Cluster-Bildung logisch zusammengehöriger Blöcken gehen verloren
  - alternativ: Einteilung der Datei in physische Cluster der Größe p (typischerweise Zylinder einer Festplatte) und der Segmente in logische Cluster der Größe l.
  - Seiten werden nun zumindest innerhalb des Clusters auf dem Medium abgelegt.

Zusatzdatei-Verfahren

Dresden Database Systems Group

(Weiteres Beispiel einer indirekten Seitenzuordnung)

Grundidee

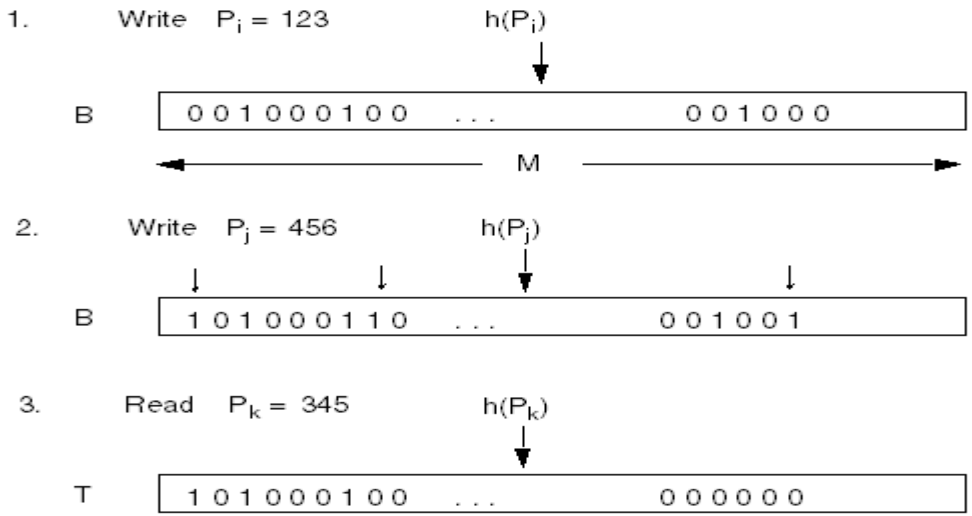
- bei Öffnung eines Änderungsbetriebs wird eine zusätzliche temporäre Datei angelegt
- alle modifizierten Seiten werden auf Blöcke dieser Zusatzdatei abgebildet, die Originaldatei wird nicht verändert
- Änderungen werden am Ende eines Änderungsintervalls in die Originaldatei verzögert eingebracht

Problem

- wie kann entschieden werden, ob die aktuelle Version einer Seite in der Zusatz- oder Originaldatei steht?
- Bitliste zeigt an, ob eine Seite in dem aktuellen Änderungsintervall bereits verändert wurde und sich somit die gültige Version in der Zusatzdatei befindet
  - Bloom-Filter
    verlässliche Aussagen darüber, dass eine Seite nicht verändert wurde

Bloom-Filter

- Einschränkung des Hauptspeicherbedarfs für die Bitliste B der Länge M
- notwendig: Hashfunktion h() über Seitennummern S_i



Bloom-Filter (2)

Dresden Database Systems Group

Beim Schreiben einer Seite

- Bitstring T der Länge M ist Ergebnis der Hashfunktion h() angewandt auf die Seitennummer S_i.
- Der Bitstring B wird ersetzt durch (B OR T)

Beim Lesen einer Seite

- T := $h(S_k)$
- Falls **(B AND T) != T**
  ist, dann ist die Seite nicht verändert worden und befindet sich in der Originaldatei
- Andernfalls, d.h. falls (B AND T) == T
  ist, dann ist die Seite **VIELLEICHT** geändert worden.

Zusammenfassung

Dresden Database Systems Group

Speicherzuordnungsstrukturen erfordern effizientes Dateikonzept

- viele Dateien variierender, nicht statisch festgelegter Größe
- Wachstum und Schrumpfung erforderlich; permanente und temporäre Dateien

Unterscheidung in

- direkte Seitenadressierung (Seite im festen Block)
- indirekte Seitenadressierung (über eine Indirektionsstufe)

Zweistufige Abbildung

- von Segment/Seite auf Datei/Block erlaubt Einführung von Abbildungsredundanz durch verzögertes Einbringen

Einbringstrategien

- direkte Einbringstrategie (update-in-place)
- verzögerte Einbringstrategie