

Lecture

# Database Implementation Techniques / Databases II

*OvGU Magdeburg, WinSem 2009/2010*

Sandro Schulze, Gunter Saake

{sanschul,saake}@iti.cs.uni-magdeburg.de.

# 3. Storage Management

---

- Storage media
- Storage arrays: RAID
- Backup media: Tertiary storage
- Structure of (physical) storage
- Pages, records and addressing
- Buffer management in detail
- Cryptographic methods

# Storage Media

---

- Several purposes:
  - ◆ Provide data for processing
  - ◆ Long-term data storage (and at the same time having them available anyway)
  - ◆ Archiving data in the long term and cheap by accepting longer access time
- In this section:
  - ◆ Storage hierarchy
  - ◆ Hard-disk/magnetic disk
  - ◆ Capacity, costs, speed

# Storage Hierarchy

---

1. Extreme fast processor with register
2. Very fast *cache memory*
3. Fast *main memory*
4. Slow *secondary storage* with randomized access
5. Very slow *nearline tertiary storage* with automatically provided storage media
6. Extreme slow *offline tertiary storage* with manually provided storage media

Tertiary storage: CD-R (Compact Disk Recordable), CD-RW (Compact Disk ReWritable), DVD (Digital Versatile Disks), magnetic tapes, e.g., DLT (Digital Linear Tape)

# Cache Hierarchy (I)

---

- Characteristics of storage hierarchy
  - ◆ Level  $x$  (e.g., level 3, main memory) has a considerably faster access time than level  $x + 1$  (e.g., level 4, secondary storage)
  - ◆ At the same time, much higher costs per memory/storage space
  - ◆ Thus, much lower capacity
  - ◆ Durability of data increase corresponding to the levels

# Cache Hierarchy (II)

---

- Reducing *access vacancy* (differences between access times on data)  $\Rightarrow$  cache storage buffers the data of level  $x + 1$  on level  $x$ :
  - ◆ *Cache* (equivalent to main memory cache)  $\Rightarrow$  faster semiconductor technology for the supply of data to the processor (level 2 in storage hierarchy)
  - ◆ *Disk storage cache* in main memory: *Buffer*
  - ◆ Cache in the context of WWW (accessing data over HTTP): Part of disk storage, which buffers parts of data, provided by the internet

# Access Vacancy

---

- Magnetic disk 70% more storage density per year
- Magnetic disk only 7% faster per year
- CPU power increased by 70% per year
- Access vacancy between main memory and magnetic disk is  $10^5$
- Measurements:
  - ◆ *ns* for nanoseconds (i.e.,  $10^{-9}$  seconds) , *ms* for milliseconds (i.e.,  $10^{-3}$  seconds)
  - ◆ KB (KiloByte =  $10^3$  bytes), MB (MegaByte =  $10^6$  bytes), GB (GigaByte =  $10^9$  bytes) und TB (TeraByte =  $10^{12}$  bytes)

# Numerical Access Vacancy (2005)

---

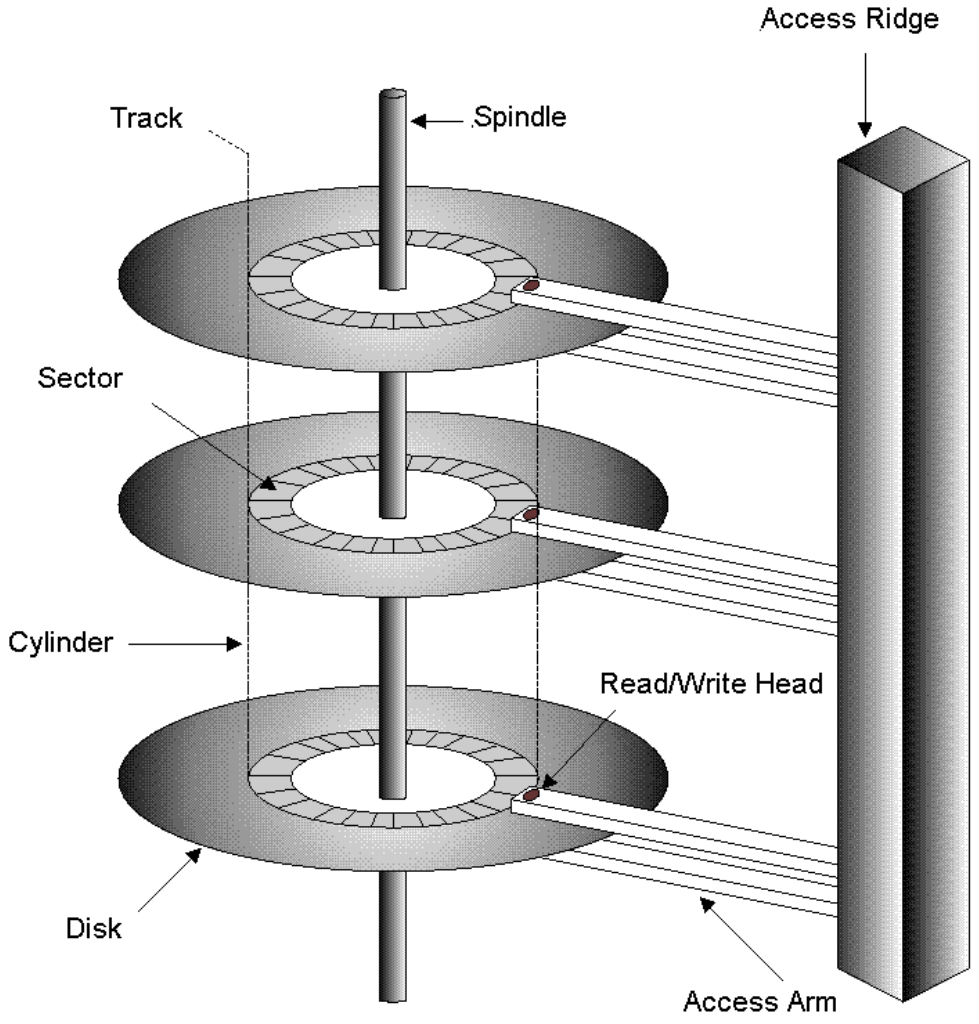
Type of storage	typical access time	typical capacity
Cache	6 ns	512 KB up to 32 MB
Main memory	60 ns	32 MB up to 1 GB
— <i>Access Vacancy</i> $10^5$ —		
Magnetic disk	12 ms	1 GB up to 10 GB
Disk farm or array	12 ms	in the range of TB

# Access Locality

---

- Caching principle does not work, if always new data (from higher levels) is needed
- In most use cases: *Locality* of (data) access
- This means, mostly the access takes place on data provided by the cache of the respective level (usually more than 90%)
- Hence: Buffer management of database system is an important concept

# Hard-Disk/Magnetic Disk



# Recording Component

---

- Disk pack with 5 to 10 disks
- Up to 10.000 revolutions per minute (rpm)
- For every disk surface a read/write head (i.e. between 10 and 20)
- Disk surface: concentric circles (*Tracks*)
- Stacked tracks of all disk surfaces: *Cylinder*
- Tracks consist of *Sectors* (512 bytes, 1 KB)
- Sector contains data for autocorrection of errors — parity bits or *Error Correcting Codes* (ECC)
- Smallest access unit on disk: *Block* (multiple of sector size)

# Positioning Component

---

- Addressing of blocks realized by cylinder/track/sector number
- Access time (from request to the transfer of block content):
  - ◆ *Seek time* (time for moving access arm)
  - ◆ *Latency time* (time for rotating to the respective sector )
  - ◆ *Data-transfer time* (time for data transfer)

# Typical Characteristics of Hard-Disks

---

<b>Characteristic</b>	<b>1998</b>	<b>1985</b>	<b>1970</b>
Average seek time	8 ms	16 ms	30 ms
Revolution time	6 ms	16.7 ms	16.7 ms
Track capacity	100 KB	47 KB	13 KB
Disk surface(s)	20	15	19
Cylinder	5000	2655	411
Capacity	10 GB	1.89 GB	93.7 MB

# Controller and further Developments

---

## (Disk) Controller

- Transfer rate: 40 MB per second
- IDE- or SCSI-Controller

## Further developments

- *Disk farms*  $\Rightarrow$  loose coupling of few but big disks; distribution of data on disk realized by operating system or database system
- *Disk arrays*, esp. RAID systems (*Redundant Array of Inexpensive Disks*): High amount of cheap standard disks (8 bis 128) organized by an intelligent controller  $\Rightarrow$  Increase of fault tolerance (reliability) or increase of parallelism of access (increase in efficiency)

# Storage Capacity and Costs I

---

<b>Size</b>	<b>Information or Medium</b>
1 KB	= 1.000 (precise: 1024 Byte)
0.5 KB 30 KB	Book page as text scanned, packed book page
1 MB	= 1.000.000 (...)
5 MB 20 MB 500 MB	The bible as text scanned book CD-ROM; Oxford English Dictionary
1 GB	= 1.000.000.000 (...)
4.7 GB 10 GB 100 GB 200 GB	Digital Versatile Disk (DVD) packed movie one floor of a library Capacity of a video tape

# Storage Capacity and Costs II

---

<b>Size</b>	<b>Information or Medium</b>
1 TB	= 1.000.000.000.000 (...)
1 TB	Library with 1M books
1 TB	huge (external) hard-disk
11 TB	big(gest) Data Warehouse (Wal-Mart)
20 TB	huge storage/disk array
20 TB	Library of Congress books stored as text
1 PB	= 1.000.000.000.000.000 (...)
1 PB	scanned books of a national library
15 PB	worldwide hard-disk production in 1996
200 PB	worldwide magnetic tape production in 1996

# Storage Medium and Costs (ca. 1999)

---

<b>Storage Type</b>	<b>Storage Medium</b>	<b>Preis (Euro/MB)</b>
Main Memory	64-MB-SDRAMs	0.5000 Euro
Secondary Storage	10-GB-IDE-HD	0.0275 Euro
Tertiary Storage	opt. disk 5 GB	0.0160 Euro
	650-MB-CD-RW	0.0150 Euro
	650-MB-CD-R	0.0015 Euro
	70-GB-DLT-Band	0.0010 Euro

# Current Developments

---

- Classical storage hierarchy in change
- Flash storage
  - ◆ Well-known from digi cams, MP3 player, PDA, etc.
  - ◆ Block by block deletion before *every* writing
    - Sequential reading identical with disk
    - Random reading considerably faster
    - Writing is more slowly
    - Durability is limited to 100.000 until 1.000.000 writing processes
  - ◆ 256 GB Chips for 40 Dollar predicted for 2012
    - Even the still more expensive than disk (approx. by factor 10)
- RAM storage ⇒ Even faster, random access
  - ◆ Main memory databases

# Storage Arrays: RAID

---

- Connection of cheap standard magnetic tapes to one logical device supervised by a certain controller
- Distribution of data over the several physical disks is realized by the controller
- Two opposed objectives:
  - ◆ Increasing the fault tolerance (system stability, reliability) by redundancy
  - ◆ Increase in efficiency through parallelism of access

# Increasing of Fault Tolerance

---

- Usage of additional disks for storage of duplicates (mirrors) of the original data  $\Rightarrow$  Failures: switch over to mirror disk
- Particular RAID-Levels (1, 0+1) permit such a mirroring
- Alternative: Control information, e.g., parity bits, are stored on a separate disk instead of the same sector as the original data
- RAID levels 2 until 6 restore defective data using parity bits or Error Correcting Codes (ECC)
- Parity bit can detect and correct a media error (if the defective disk is known)

# Increase of Efficiency (I)

---

- Database distributed over several disks, that can be accessed parallel  $\Rightarrow$  Access time decreases nearly linear to the number of disks available
- Distribution
  - ◆ Bit by bit (i.e., in the case of 8 available disks, one byte can be distributed)
  - ◆ Byte by byte (bytewise)
  - ◆ Block by block (blockwise)

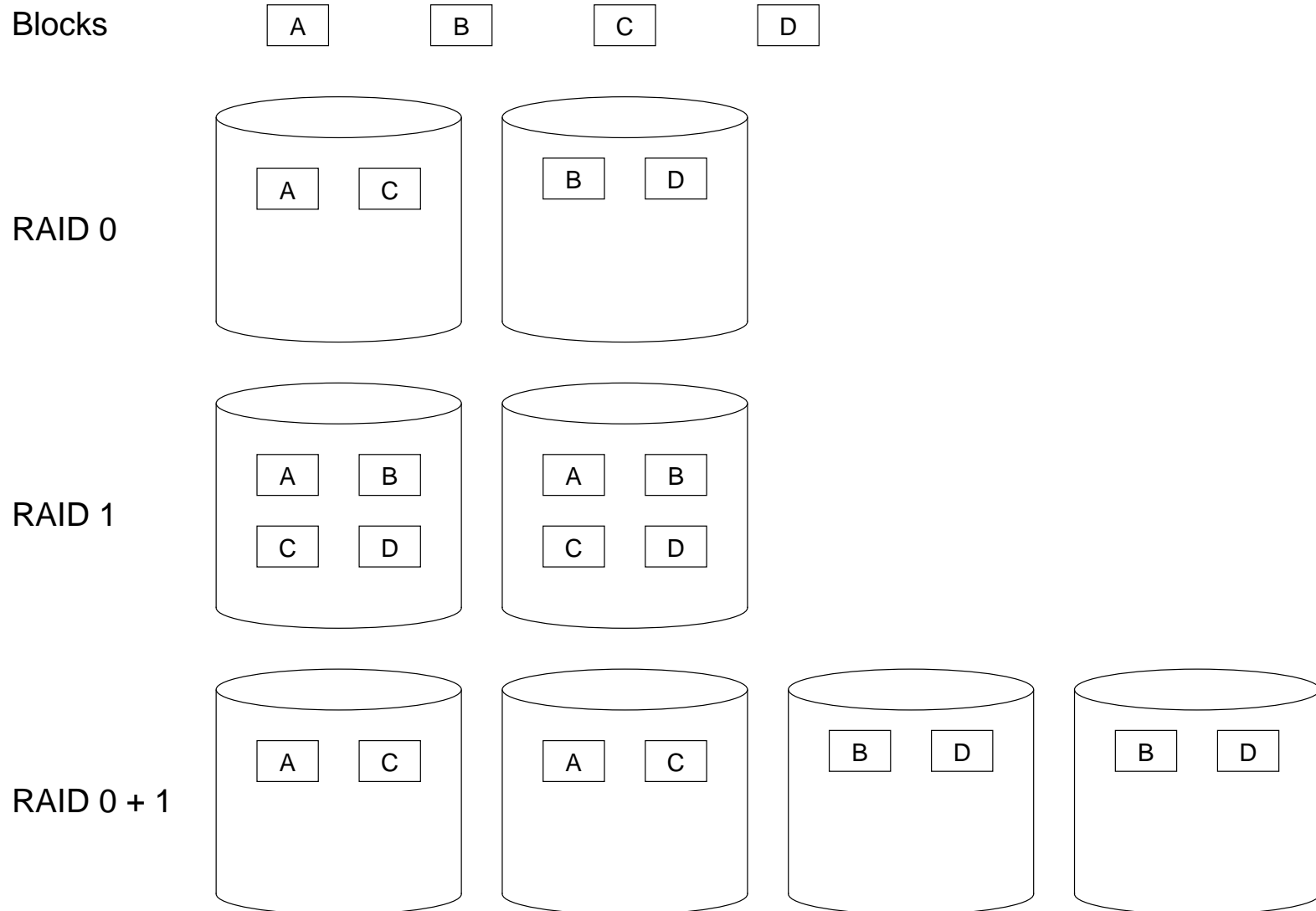
# Increase of Efficiency (II)

---

- Higher RAID levels (from level 3) combine error correction and bit- or blockwise distribution of data
- Differences (to lower levels):
  - ◆ *faster access* on certain data
  - ◆ *increased throughput* for many transactions (queued for parallel processing) by load balancing the overall system

# RAID Levels 0, 1 und 0+1

---



# RAID Level 0

---

- Data blocks are distributed on physical disks (available for the RAID system) by rotation principle (*Striping*)
- Here: blockwise Striping
- Advantages:
  - ◆ Parallel reading of data of sequenced blocks
  - ◆ Load balancing if many parallel read TAs are queued (needed blocks exist probably on different disks)
- Disadvantages:
  - ◆ No speedup for random access on one specific block
  - ◆ No redundancy or control information

# RAID Level 1

---

- Blocks (to be stored) are mirrored/duplicated on several disks
- Disadvantages:
  - ◆ Solely increase of efficiency: load balancing for reading of parallel transactions
- Advantages
  - ◆ In case of errors simply switch to one of the mirror disks
  - ◆ Subsequently, the defective disk can be replaced and filled with correct data

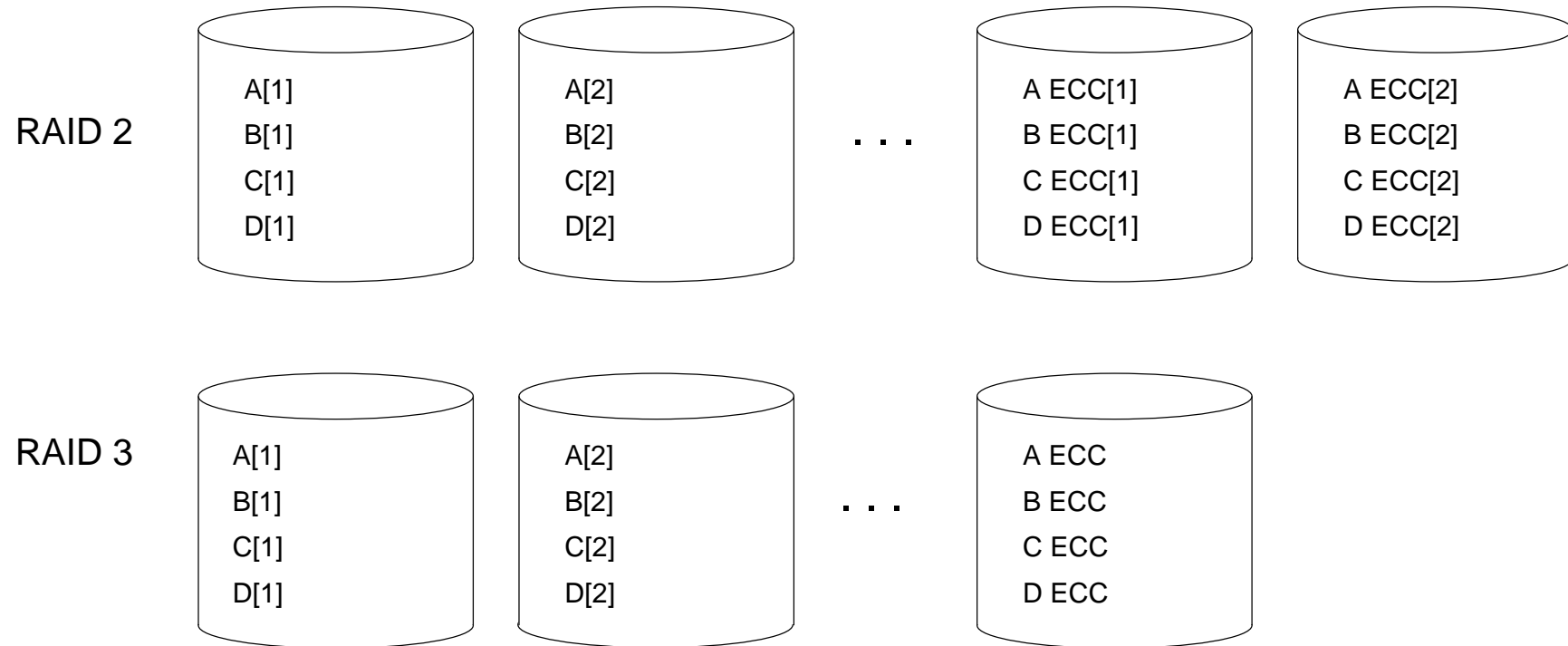
# RAID Level 0+1

---

- Combination of level 0 and 1 to obtain the advantages of both
- Foundation for higher RAID-Levels: Combination of Striping for increase of efficiency and actions for error correction are retained
- However, doubling the memory space requirements is avoided
- Instead of mirror disks, control information, e.g., parity bits or Error Correcting Codes (ECC) are used

# RAID Levels 2 und 3

---



# RAID Level 2

---

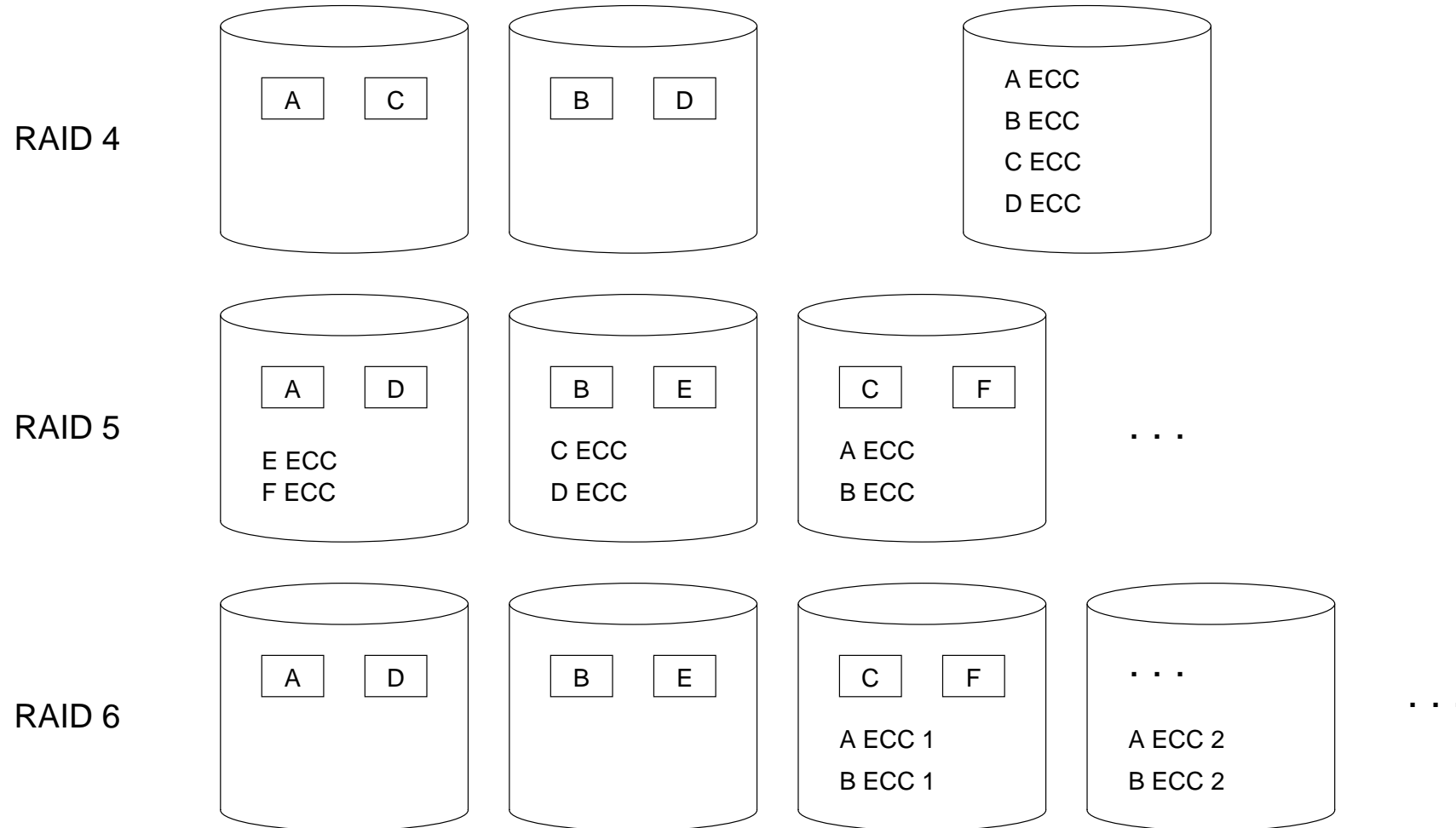
- Bitwise Striping
- Parity bits or extended ECC on additional disks
- Reading in one data requires 8 parallel read operations
- Access not more efficient, but the overall throughput is octuplicated (i.e., eight times higher)

# RAID Level 3

---

- Principle: Disk detects error  $\Rightarrow$  byte can be reconstructed with the help of the 7 remaining bits and *one* parity bit
- Parity bits use only one single, dedicated disk
- Control information are reduced to the basics, and thus, memory space is reduced further

# RAID Levels 4 to 6



# RAID Level 4

---

- Blockwise Striping
- One dedicated disks for parity bits
- Small data sets are readable more efficient, since only one physical disk is affected

# RAID Level 5

---

- No additional, dedicated disk for storage of parity bits
- Instead, parity bits are distributed over existing disks
- Hence, bottleneck of level 3 and 4 is removed: So far, every write operation on parity disk as well  $\Rightarrow$  every write operation waits for dedicated disk
- Improved load balancing by usage of different disks

# RAID Level 6

---

- Based on RAID level 5
- More control information on data disk
- More than one media error can be corrected
- Only barely implemented

# Overview RAID Levels

---

	0	1	0+1	2	3	4	5	6
Striping blockwise	✓		✓			✓	✓	✓
Striping bitwise				✓	✓			
Copy		✓	✓					
Parity				✓	✓	✓	✓	✓
Parity dedicated disk					✓	✓		
Parity distributed							✓	
Detecting several errors								✓

# Comparison of particular RAID Level (I)

---

## Application profile

- Number of read operations
- Number of write operations
- Requirements regarding system stability

# Comparison of particular RAID Level (II)

---

## Characteristics

- Fastest error correction: Level 1 (well-suited for log files)
- Solely increase of efficiency: Level 0 (well-suited for video server)
- Level 3 and 5 improvements of level 2 and 4
- Level 3 and 5 for working with huge amounts of data
- Level 3 increases overall throughput
- Level 5 improves random access
- Level 6 would be improvement of level 5, but rarely realized
- Archive media are not (!) replaced by RAID systems

# Backup Media: Tertiary Storage

---

## Requirements

- Less used parts of database, with probably huge size (Text, Multimedia) should be stored „cheaper“ than with standard mag tapes
- Currently used databases have to be back upped additionally (archiving)

## Tertiary storage: Medium replaceable

- *offline*: Media have to be changed manually
- *nearline*: Media are changed automatically (*jukeboxes, mag tape robots*)

# Optical Disks

---

- CD-ROM, CD-R, CD-RW; DVD, DVD-R, DVD-RW, DVD+RW, ...
- Differences: executable operations, storage capacity
- Storage capacity CD: approx. 700 MB
- Storage capacity DVD: 4.7 GB to 17 GB data
- Optical disks and devices quite cheap
- Although random access is possible, the access is very slow (about 250 ms)
- Durability is comparatively high (approx. 30 years)
- Popular as storage for data required rarely but continuous

# Tapes

---

- Very cheap medium with low cost devices
- Access yet more slowly than optical disks
- Sequential access: Transfer rate between 1 and 10 MB per second (acceptable)
- Random access: in the range of some minutes (on average)
- Very high capacity in the range of GBs (DLT (Digital Linear Tape) contains up to 70 GB packed data)
- Popular as archive storage

# Jukeboxes and Robots

---

- Shut down / Rewind,
- Remove media from device,
- Mounting the medium,
- Grab new medium,
- Loading of media and boot up

cost over 20 seconds with optical disks and more than 2 minutes with tapes

- Typical capacities: Tape robots with more than 10.000 cartridges (range of petabytes possible)

# Long-term Backup/Archival

---

Aspects of durability:

- Physical stability of the medium ensures the *integrity* of data
- Availability of devices and drivers ensure the *readability* of data
- Available meta data ensure the *interpretability* of data
- Availability of programs, processing or working on the data ensure the *reusability* of data

# Physical Stability: Integrity

---

- 10 years for magnetic tapes, 30 years for optical disks
- In comparison to classical archival media, e.g., paper, very low
- However, integrity of data is even longer guaranteed than other aspects

# Readability

---

- Problem: Which systems, used today, can read tapes, floppy disks and punchcards mostly used in the 60s and 70s ?
- new devices/systems would have to maintain numerous parameters like recording formats, type of parity information etc. for several decades
- Furthermore, procedure (of recording) not standardized or standardization is enhanced continuously

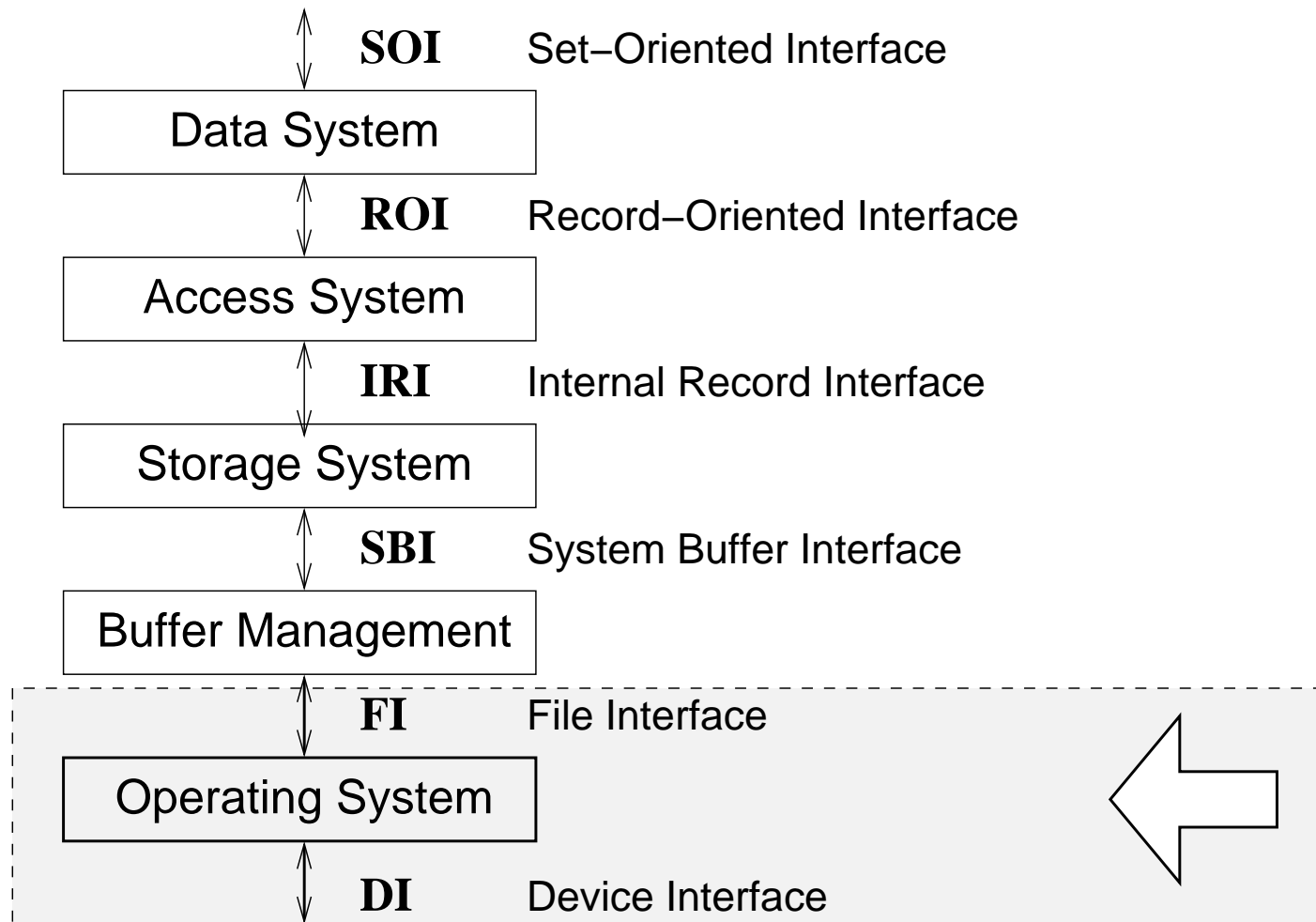
# Interpretability

---

- Different codes for representation of characters (EBCDIC, ASCII, 16-Bit-Unicode)
- Document formats: Which (current) SW can read formatted text documents, created by word processing SW in the 70s or 80s ?
- Solution: Storage in open standards like page description languages (Postscript) or markup languages (HTML, XML)

# Management of Physical Storage

Abstraction of specific storage and backup media  
Model: Sequence of Blocks



# Operating System Files (I)

---

- Every relation or access path in one OS file respectively
- One or more OS files, relations and access paths are managed by the DBS itself within these files
- DBS itself manages the hard-disk and uses blocks in their original form (*raw device*)

# Operating System Files (II)

---

Why not always using OS file management?

- Platform (OS) independence
- In 32-Bit operating systems: file size at most 4 GB
- OS files only on one medium (storable)
- OS-specific buffer management (i.e., blocks of secondary storage are used in main memory) does not meet the requirements of DBS

# Blocks and Pages

---

- Assignment of physical blocks to *pages*
- Usually with constant multiplier: 1, 2, 4 or 8 blocks (of one track) on one page
- here (for simplicity): „one block — one page“
- Addressing of higher levels of DBS via page numbers

# Services

---

- Allocation or deallocation of memory space
- Fetch or write of page content
- Allocation in a way, that logical consecutive data areas (e.g., a relation) are preferably stored in consecutive blocks on disk
- After numerous update operations: *Reorganisation methods*
- *(Free) storage management*: Doubly-linked list of pages

# Mapping of Data Structures

---

- Mapping of conceptual level to internal data structures
- Supported by *meta data* (in the Data Dictionary, e.g., the internal schema)

<b>Conc. level</b>	<b>Internal level</b>	<b>File system/Disk</b>
Relations →	Log. files →	Phys. files
Tuple →	Records →	Pages/Blocks
Attribute values →	Fields →	Bytes

# Alternatives of Mapping

---

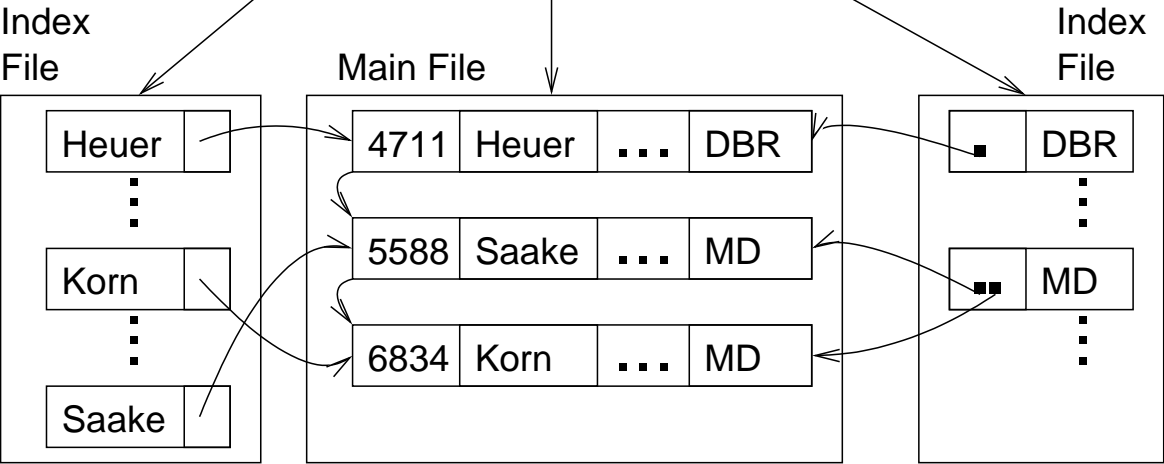
- Example 1: Every relation in one logical file, these files alltogether in a single physical file
- Example 2: Cluster storage – several relations in one logical file

# Common Form of Storage (I)

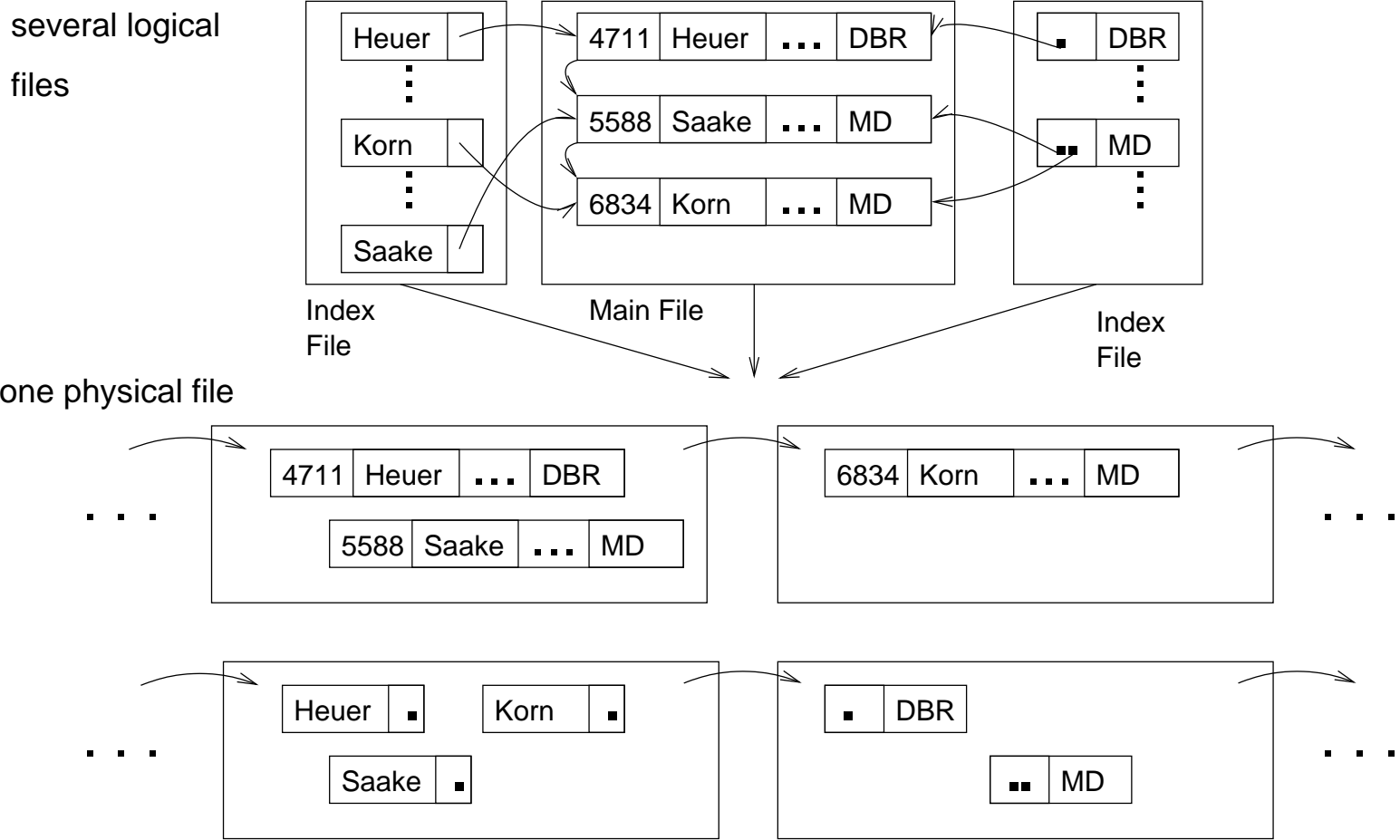
Relation

PANr	Surname	. . .	Place
4711	Heuer	. . .	DBR
5588	Saake	. . .	MD
6834	Korn	. . .	MD
⋮	⋮		⋮

several logical files



# Common Form of storage(II)



# Adaptation of Records to Blocks

---

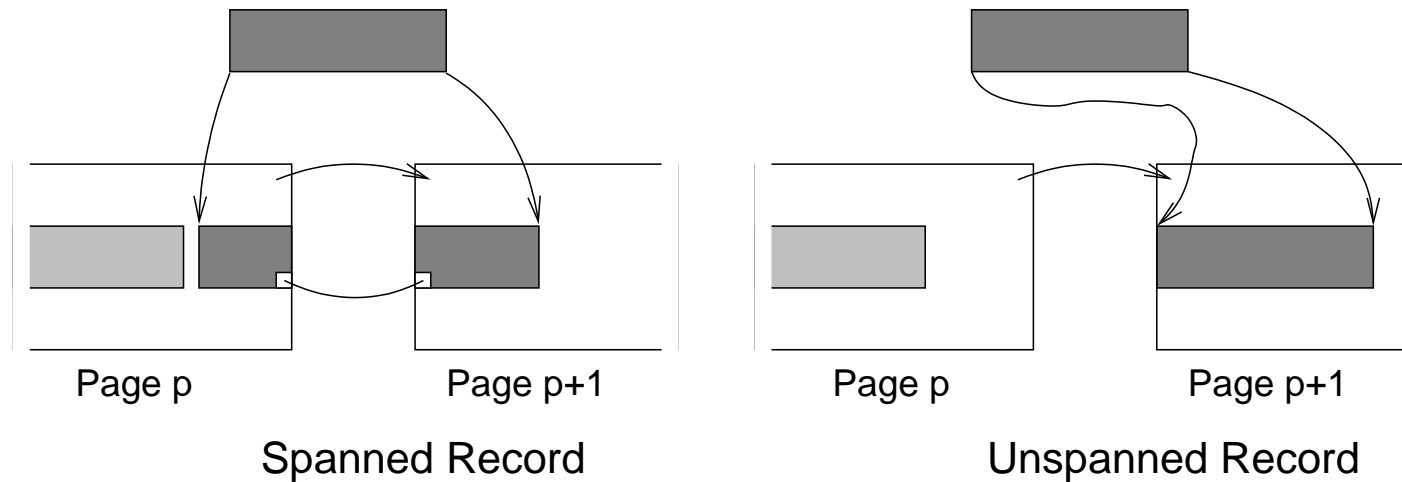
- Records (maybe of variable length) have to be adapted to blocks, consisting of a constant number of bytes:

## *Blocking*

- Blocking depends on length of data fields/records (variable or fix)
  - ◆ Records with variable length: Higher effort for read/write operations, record length has to be redetermined
  - ◆ Records with fixed length: Higher storage effort (overhead)

# Blocking Techniques

---

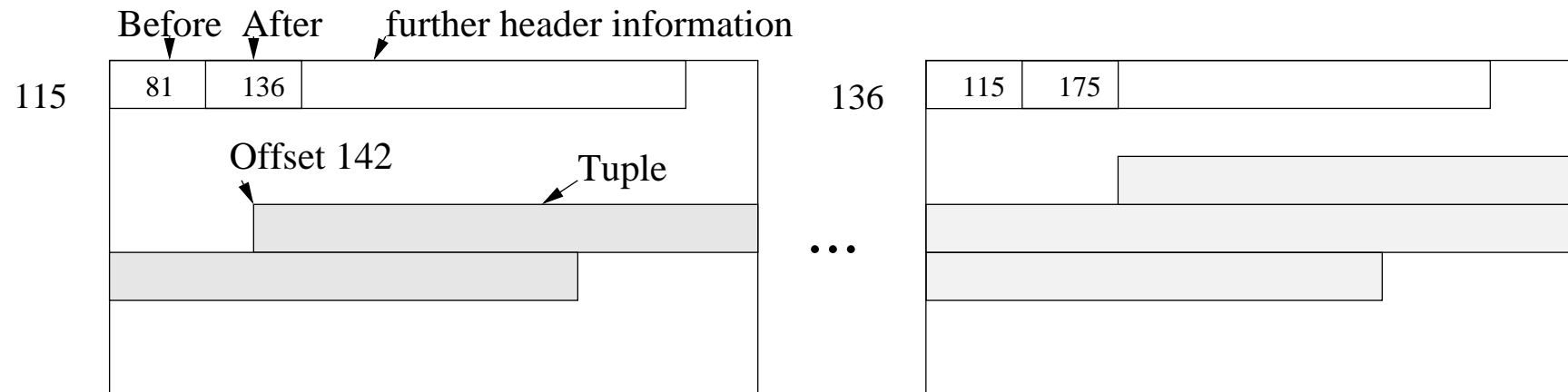


- *Unspanned Records*: every record in at most one block
- *Spanned Records*: record probably in several blocks
- Standard: Unspanned records (only in the case of BLOBs or CLOBs spanned records are used)

# Pages, Records and Addressing

---

- Structure of pages (in storage management):  
Double-linked list (*Free-List*)
- Free pages in (free) storage management



# Page

---

- *Header*
  - ◆ Information about predecessor and successor page
  - ◆ Probably page number itself
  - ◆ Information about typ of record
  - ◆ Free space (available)
- *Records*
- Unassigned bytes

# Page: Addressing of Records

---

- Addressable units
  - ◆ Cylinder
  - ◆ Tracks
  - ◆ Sectors
  - ◆ Blocks or pages
  - ◆ Records in blocks or pages
  - ◆ Data fields/areas in records (e.g., a certain attribute)
- Example: Address of record realized by page number and offset (relative address from top of the page in bytes)

(115, 142)

# Page Access as Bottleneck

---

- Measurement for the speed of DB operations: Number of page accesses on secondary storage (caused by access vacancy)
- Rule of thumb: Time of access  $\Leftarrow$  Quality of access path  $\Leftarrow$  Number of required pages accesses
- Main memory operations not arbitrarily negligible

next slides: **Types of Records**

# Pinned records

---

- *Pinned records* are bounded to their position
- Example: References (directly on record) by pointer from other, hardly detectable page(s); e.g., moving of record (115, 142) to page 136
- Risk: Pointer referencing to nothing (*dangling pointers*)

# Unpinned records

---

- *Unpinned records*: References by „logical pointer“
- E.g.: Matriculaion number for referencing to a student record
- This logical pointer can be mapped to current address at a central place (e.g., index file for student relation)
- Disadvantage: Moving of record requires loading both directly affected pages (source and target page of moving operation) as well as loading the index page
- This is resolved by TID concept

# Records with fixed length

---

SQL: Data types with fixed and variable length

- *char(n)* string with fixed length  $n$
- *varchar(n)* string with variable length with maximum length  $n$

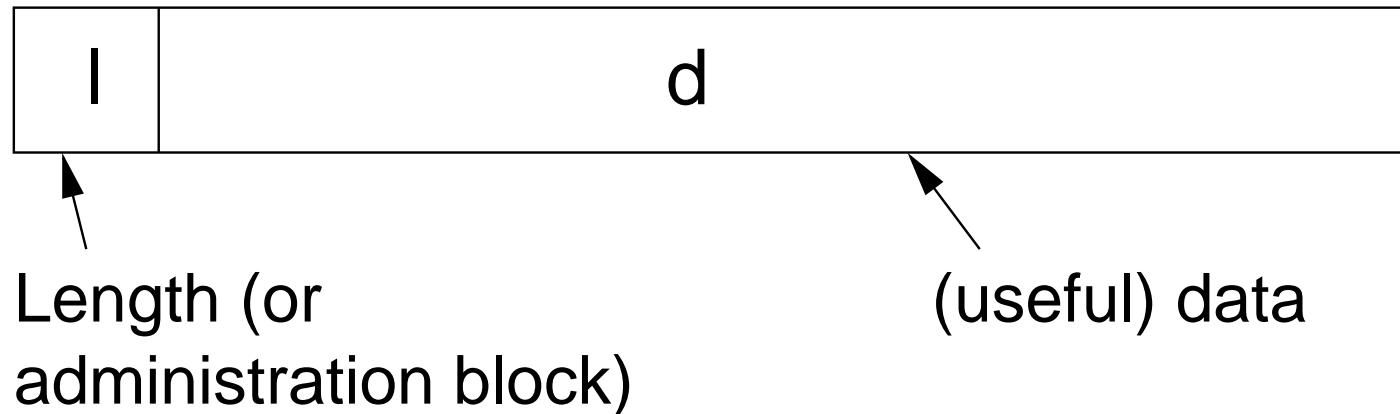
Structure of records, if all data fields with fixed length:

1. *Administration block* with
  - Type of record (if different types on one page allowed)
  - Deletion bit
2. *Free space* for adjusting the offset
3. *User data* of record

# Records with variable length

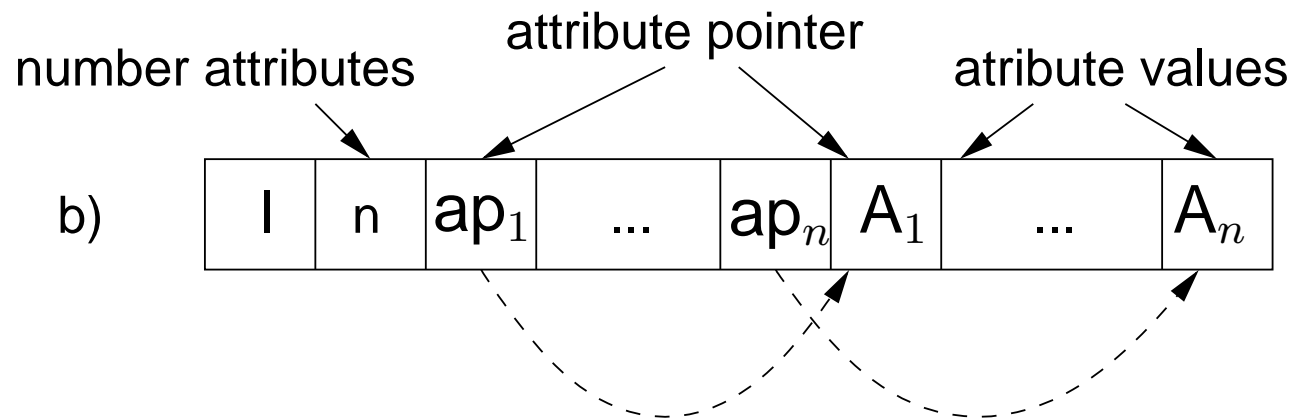
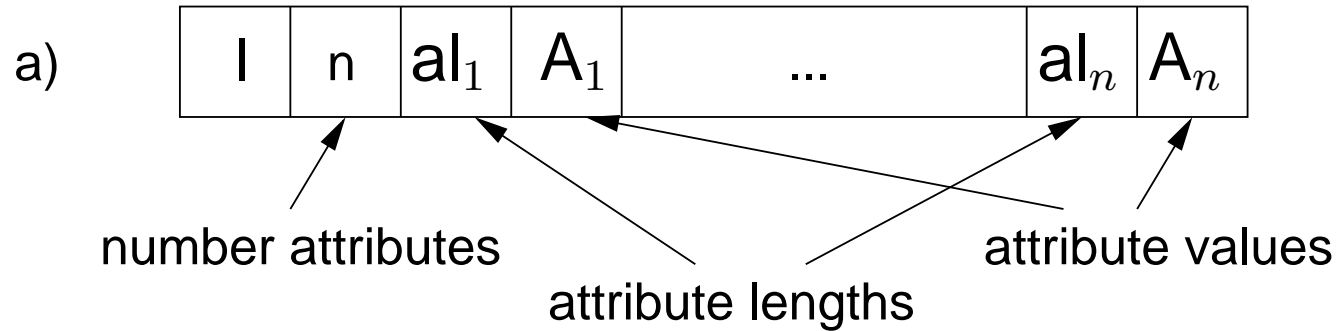
---

- Required in administration block: Record length  $l$ , so that the length of the user data area  $d$  is known



# Records with variable length (II)

---



# Storage of Records with variable Length

---

- *Strategy a)*: Every data field with variable length  $A_i$  starts with a *length pointer*  $al_i$ , specifying the length of the following data field
- *Strategy b)*: After length pointer  $l$  and number of attributes at the beginning of the record, a pointer field  $ap_1, \dots, ap_n$  for all data fields with variable length is established
- Advantage of strategy b): easier navigation within one record (even for records in pages  $\Rightarrow$  TID)

# Application of variable Records

---

„Repetition Groups“: List of values with same data type

- Strings of variable length, e.g., *varchar(n)* are repetition groups with *char* as basis data type  $\Rightarrow$  mathematical, this is the Kleene closure  $(char)^*$
- Attribute values with a set or list nature, which should be stored denormalized within the record (e.g., storage as nested relation or cluster storage); for a list of *integer* values this would be  $(integer)^*$
- Address field for index file, which points to several records for one attribute value (*secondary index*), i.e.,  $(pointer)^*$

# Huge, unstructured Records

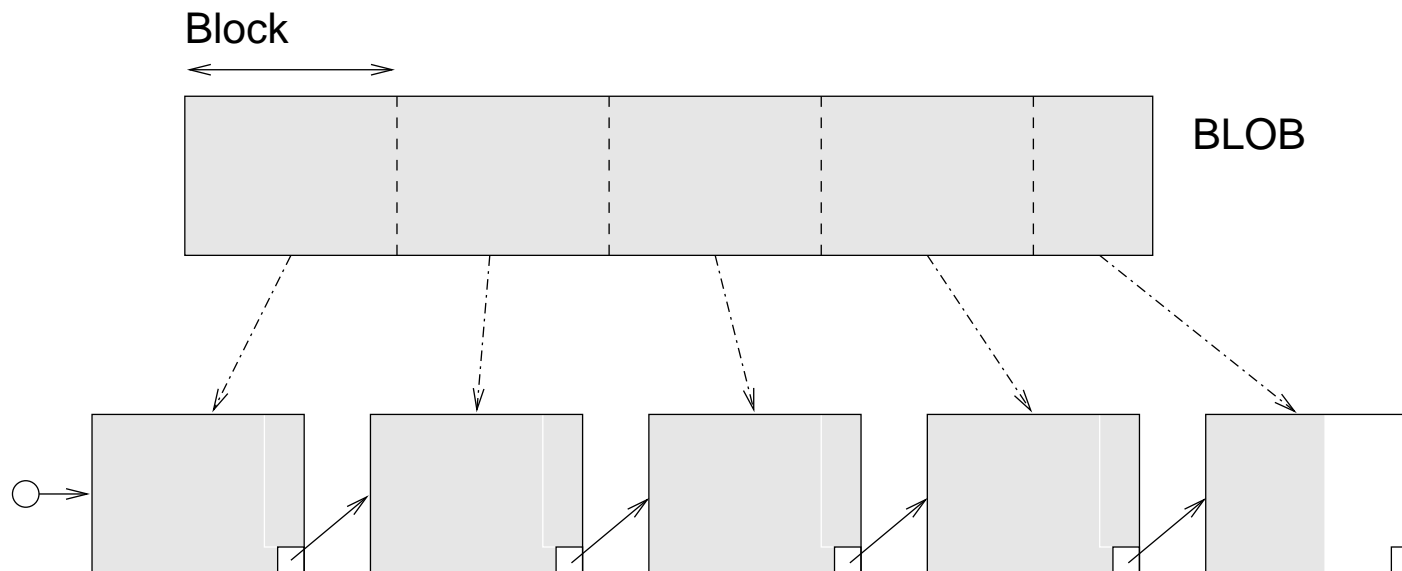
---

- RDBS data types for very large, unstructured Information:
  - ◆ *Binary Large Objects (BLOBs)*: Byte sequences like images, audio or video sequences
  - ◆ *Character Large Objects (CLOBs)*: Sequences of ASCII characters (unstructured ASCII text)
- long fields generally exceed the limit of one page, hence only non-BLOB fields are stored on original page

# BLOB Storage: Solution 1

---

- Pointer as attribute value: Pointer points to the beginning of page or block list, which contains the BLOB



- Advantage for insert, delete or modify operation
- Disadvantage for random access within the BLOB

# BLOB Storage: Solution 2

---

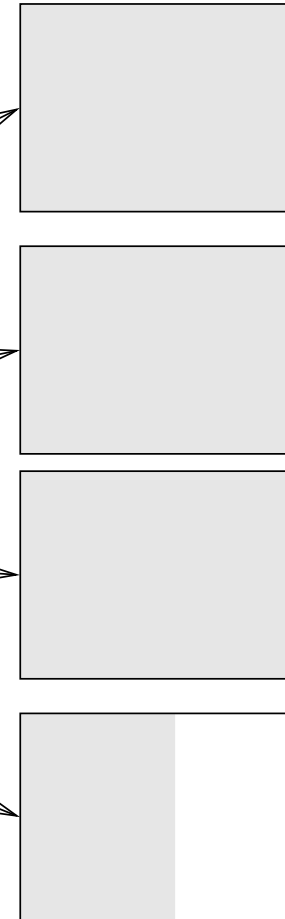
- BLOB directory as attribute value:
  - ◆ Size of BLOB
  - ◆ Further administration information
  - ◆ Several pointers, referencing to single pages
- Advantage: fast access on partitions of the BLOB
- Disadvantage: Fixed, limited (maximum) size of the BLOB (e.g., Gigabyte BLOB; 8-byte addressing, page size 1 KB  $\Rightarrow$  8 MB for one BLOB directory)
- More efficient: B-Tree for storage of BLOBs (v.i.)

# BLOB Storage: Solution 2 (II)

---

BLOB Directory

BLOB size	
Administration info.	
Pointer to block 1	
Pointer to block 2	
Pointer to block 3	
...	
Pointer to Block k	



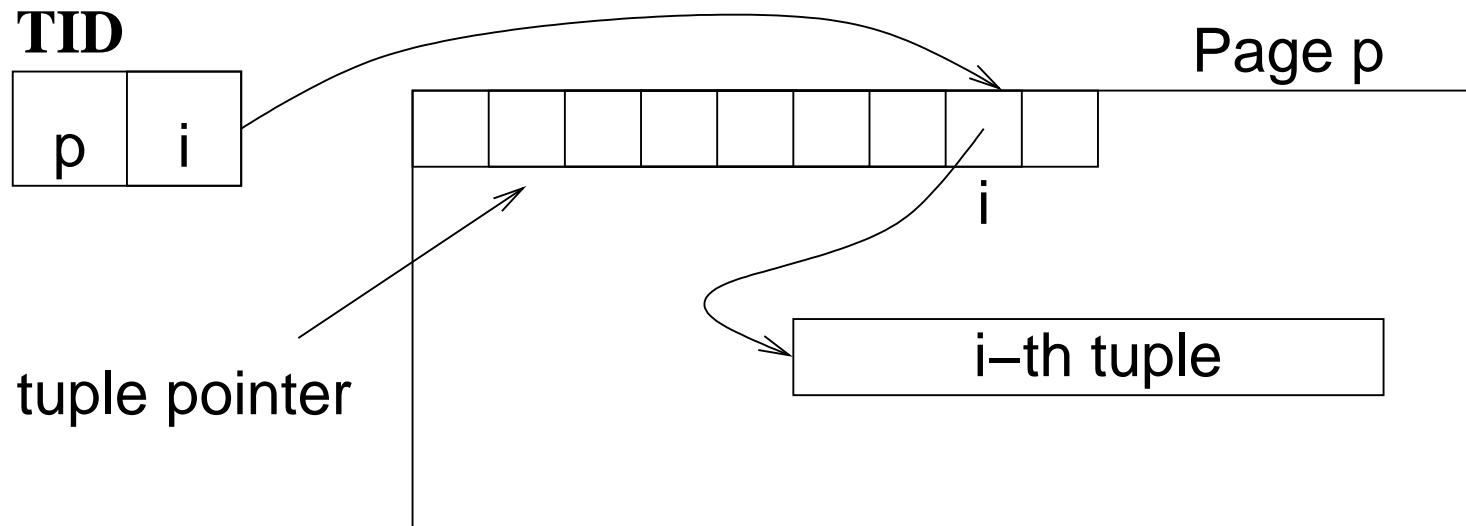
# Addressing: TID Concept

---

- *Tupel-Identifier* (TID) record address consisting of page number and offset
- Offset is used to reference within a single page  $\Rightarrow$  an offset value of  $i$  points to the  $i$ -th entry in a list of *tuple pointer* at the beginning of the page
- Each tuple pointer contains offset value
- Movements within a page: All references from „outside“ remain unchanged
- Movements to another page: Old record is replaced by new TID pointer
- This two-staged reference not desirable for efficiency reasons: Reorganisation in periodical intervals

# TID Concept: One-Stage Reference

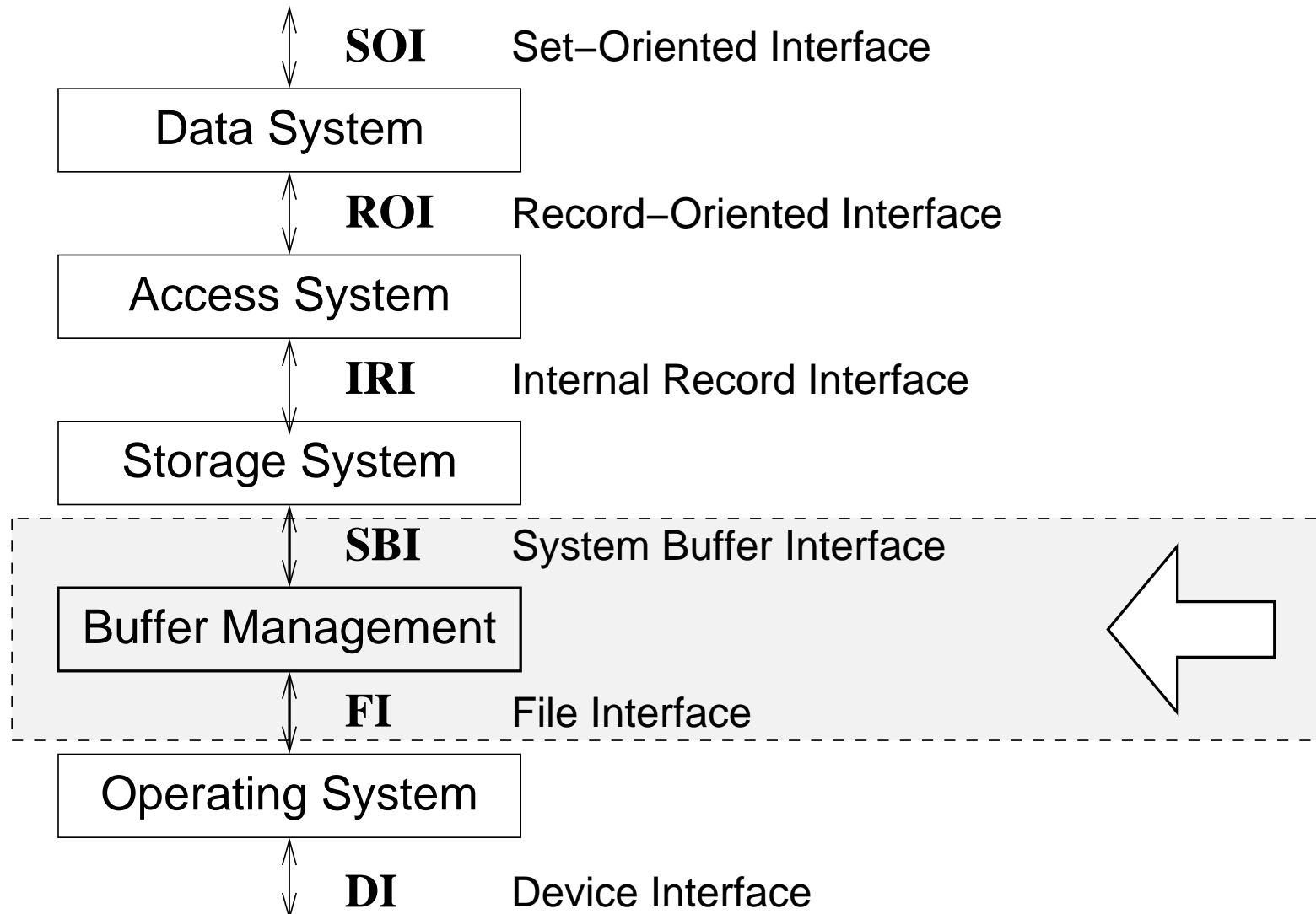
---





# Buffer Management in Detail

---



# Functions of Buffer Management

---

- *Buffer*: Designated part of the main memory
- Divided into *buffer frames*, each frame can contain one page from the hard-disk
- Functions:
  - ◆ Buffer management has to search required page in the buffer ⇒ efficient *search techniques*
  - ◆ Parallel DB transactions: clever *storage allocation* within the buffer
  - ◆ Buffer full: adequate *page replace strategies*
  - ◆ *Differences between OS buffer and DB buffer*
  - ◆ Specific application of buffer management: *shadow storage concept*

# Searching a Page

---

- *Direct Search*: Linear search without additional means
- *Indirect Search*:
  - ◆ *Ordered or unordered table*: all pages of buffer are observed
  - ◆ *Linked list*: enables a faster, ordered insertion
  - ◆ *Hash table*: with adequate hash function an advantageous effort for searching and modifying is possible

# Storage Allocation in the Buffer

---

for several transactions to be executed in parallel

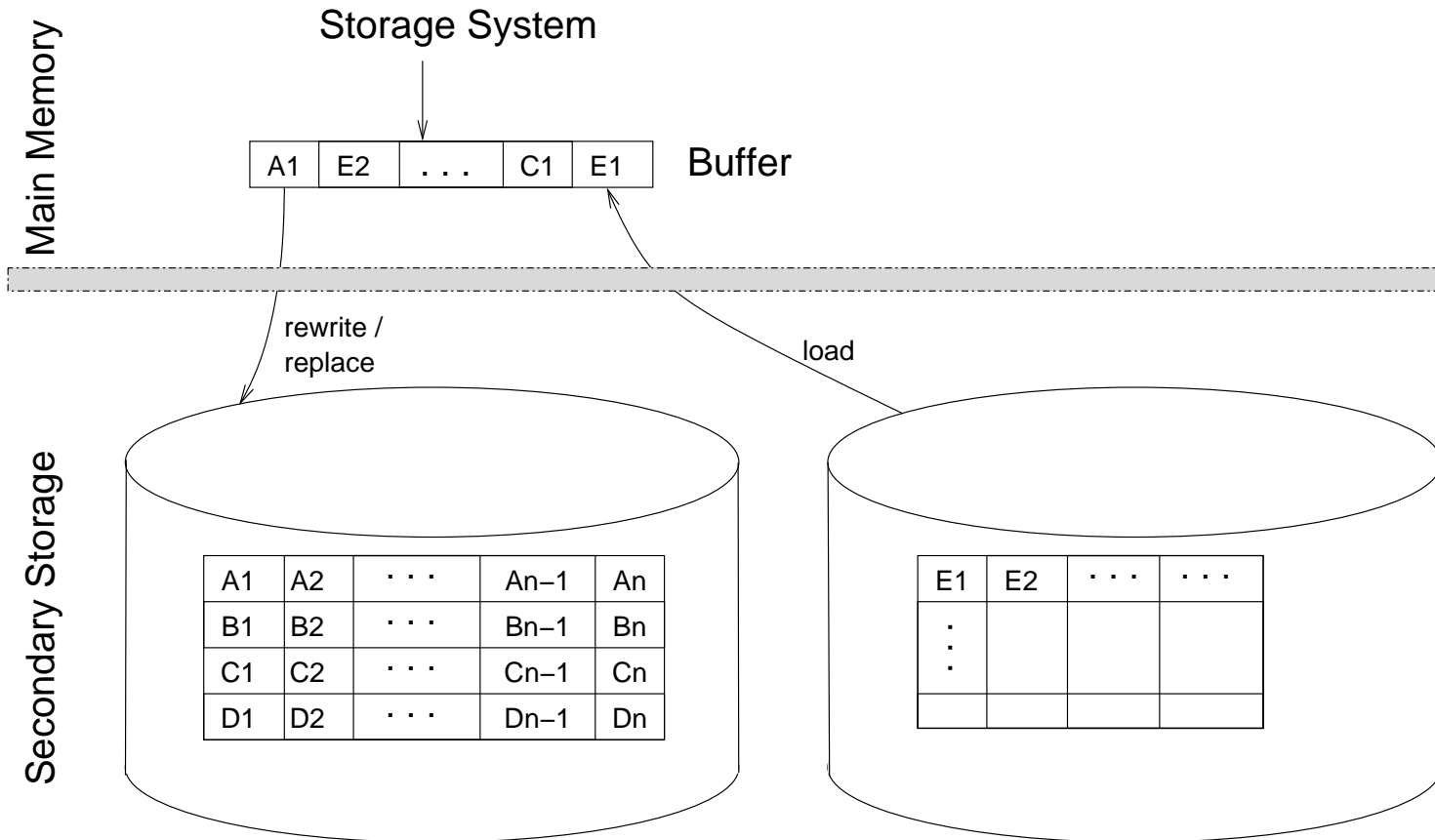
- *Local strategies*: Disjunctive parts of the buffer are made available for every transaction (size can be determined statically before execution of TA or dynamically at program runtime)
- *Global strategies*: Access behaviour of all TAs determines storage allocation (pages, jointly referenced by several TAs can be regarded better this way)
- *Strategies related to page type*: Partition of the buffer: Buffer frames for data pages, access path pages, data dictionary pages, ...

# Page Replace Strategies

---

- Storage system requests page  $E_2$ , which is not in the buffer
- All buffer frames are reserved
- Before loading  $E_2$ , a buffer frame has to be cleared
- The page has to be selected by the strategies described below
- If page has been modified within the buffer, *rewrite* the page to the hard-disk
- If page has only be read within the buffer, it can be overwritten(*replaced*)

# Page Replace schematic



# Page Replace: Techniques (I)

---

- *Demand-paging technique*: Exactly one page in the buffer is replaced by requested page
- *Prepaging technique*: Besides the requested page further pages, potentially required in future, are loaded into the buffer (e.g., reasonable for BLOBs)
- *Optimal strategy*: Which page has the maximum distance to the next usage? (not feasible, future reference behaviour not predictable)

~> Feasible techniques exhibit no knowledge over future reference behaviour

- *Fortune strategy*: Same reuse probability is assigned to every page

# Page Replace: Techniques (II)

---

- Suitable, realizable techniques should use previous reference behaviour auf Seiten nutzen, for estimating the expected value for reuse
  - ◆ Better than fortune strategy
  - ◆ Approximation to optimal strategy

# Characteristics of established Strategies

---

- Age of page in the buffer:
  - ◆ Age of page after „storage“ (the global strategy (G))
  - ◆ Age of page after last time of reference (strategy of youngest behaviour (Y))
  - ◆ Age of page is not considered (–)
- *Number* of references to page in the buffer:
  - ◆ Number of all references to a page (the global strategy (G))
  - ◆ Only number of last references to the page is considered (strategy of youngest behaviour (Y))
  - ◆ Number of references is not considered (–)

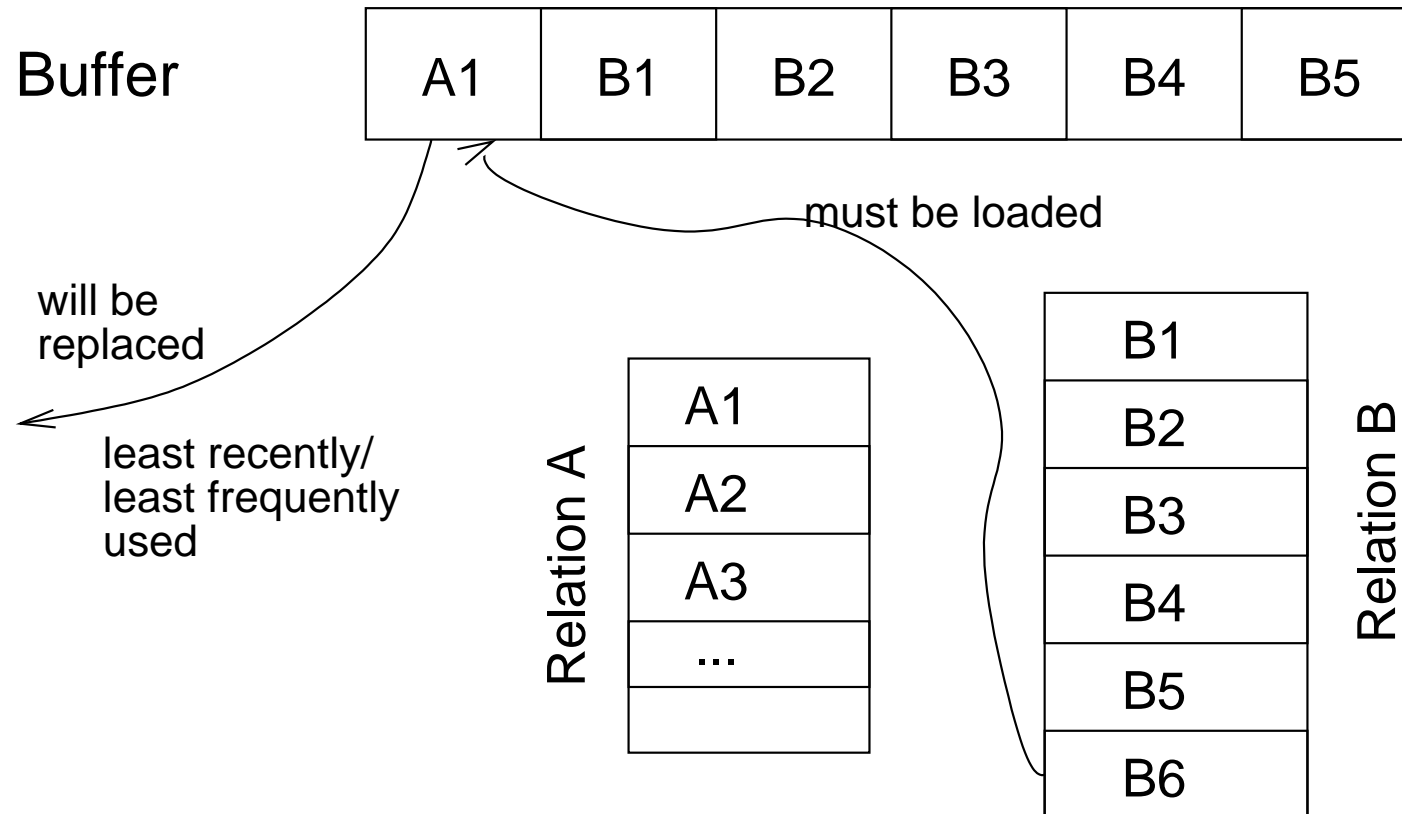
# Classification of established Strategies

---

<b>Technique</b>	<b>Principle</b>	<b>Age</b>	<b>Anzahl</b>
FIFO	oldest page replaced	G	–
LFU (least frequently used)	replace page with least frequency	–	G
LRU (least recently used)	replace page, not referenced for longest time (System R)	Y	Y
DGCLOCK (dyn. generalized clock)	logging the replace frequency of important pages	G	YG
LRD (least reference density)	replace page with least reference density	YG	G

# Lack of Appropriateness of OS Buffer (I)

Natural Join of relations  $A$  and  $B$  (corresponding sequence of pages:  $A_i$  bzw.  $B_j$ )  $\rightsquigarrow$  Implementation: *Nested-Loop*



# Lack of Appropriateness of OS Buffer (II)

---

- FIFO:  $A_1$  replaced, since oldest page in buffer
- LRU:  $A_1$  replaced, since page has only been used during the first step for reading the compared tuple

## Problem

- For next step,  $A_1$  is required again
- Further „build up“: for loading  $A_1$ ,  $B_1$  has to be removed (but needed in the next step) usw.

## Page Replace Strategies of DBS

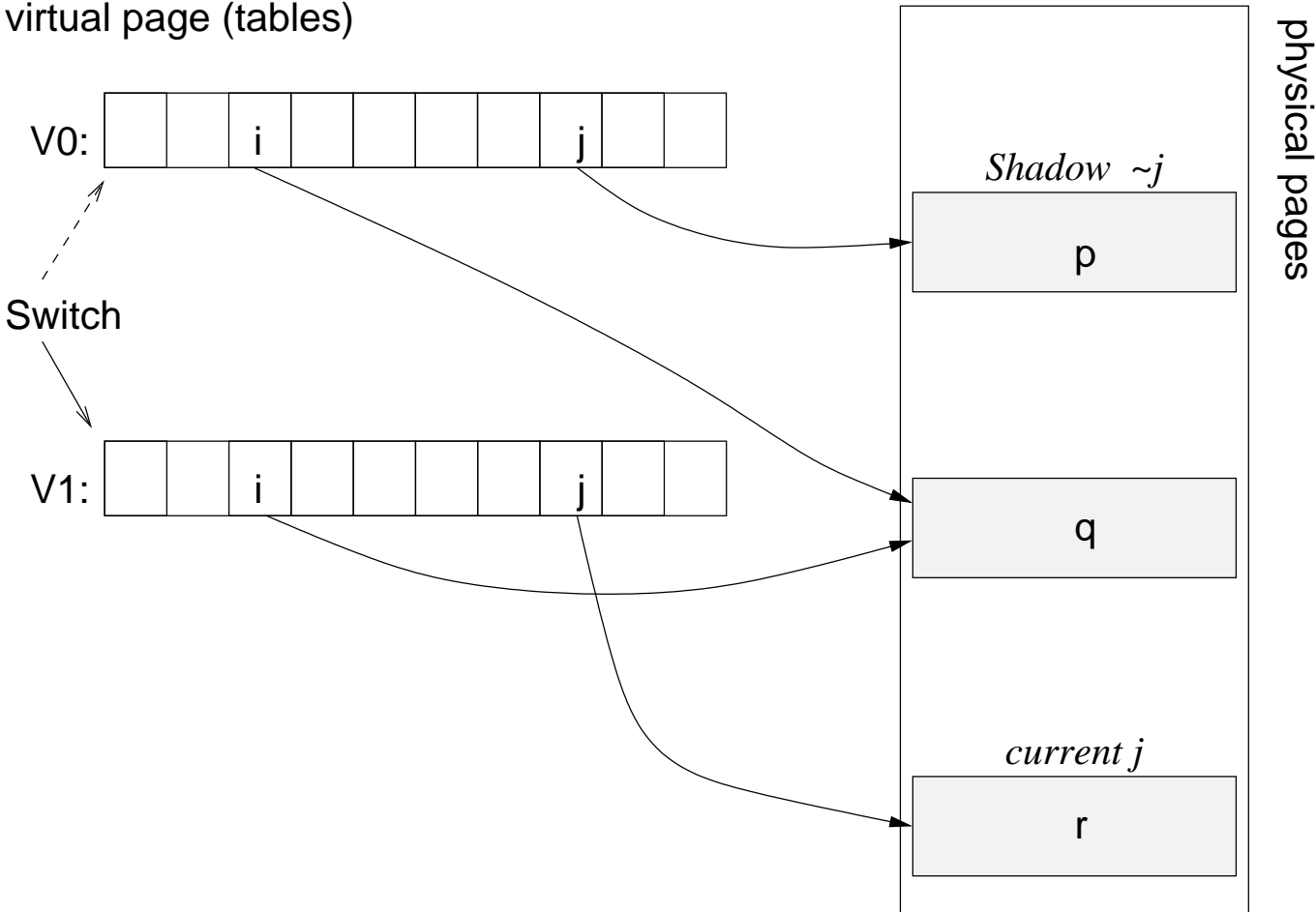
- *Fixing of pages*
- *Release of pages*
- *Rewriting a page (e.g., at the end of a TA!)*

# Shadow Storage Concept (I)

---

- Modification of buffer concept
- If page has to be (re)written to the disk during a TA: not to original page, but to a new page
- Two parallel auxiliary structures (instead of one)
- Auxiliary structures: *virtual page tables*

# Shadow Storage Concept (II)



# Shadow Storage Concept (III)

---

- In case of TA abort, only access on original version  $V_0$  necessary for recovery of the old state (before TA started) of the pages
- Toggle between both versions of virtual page tables using a „switch“
- TA committed successfully  $\Rightarrow V_1$  becomes original version and  $V_0$  references to new shadow pages
- Disadvantage: Formerly coherent page areas of one relation are „scattered“ over secondary storage (after change operations on DB)

# Cryptographic Methods

---

- Prohibit unauthorized access on DB
  - ◆ Within DBS: Right management using **grant**
  - ◆ On OS level: file encryption against „Dump“
  - ◆ Network: file encryption and, if data are transferred, secure channels (SSL)
- Common methods
  - ◆ Data Encryption Standard
  - ◆ Public-Key techniques: RSA, PGP