

Lecture

# Database Implementation Techniques / Databases II

*OvGU Magdeburg, WinSem 2009/2010*

Sandro Schulze, Gunter Saake

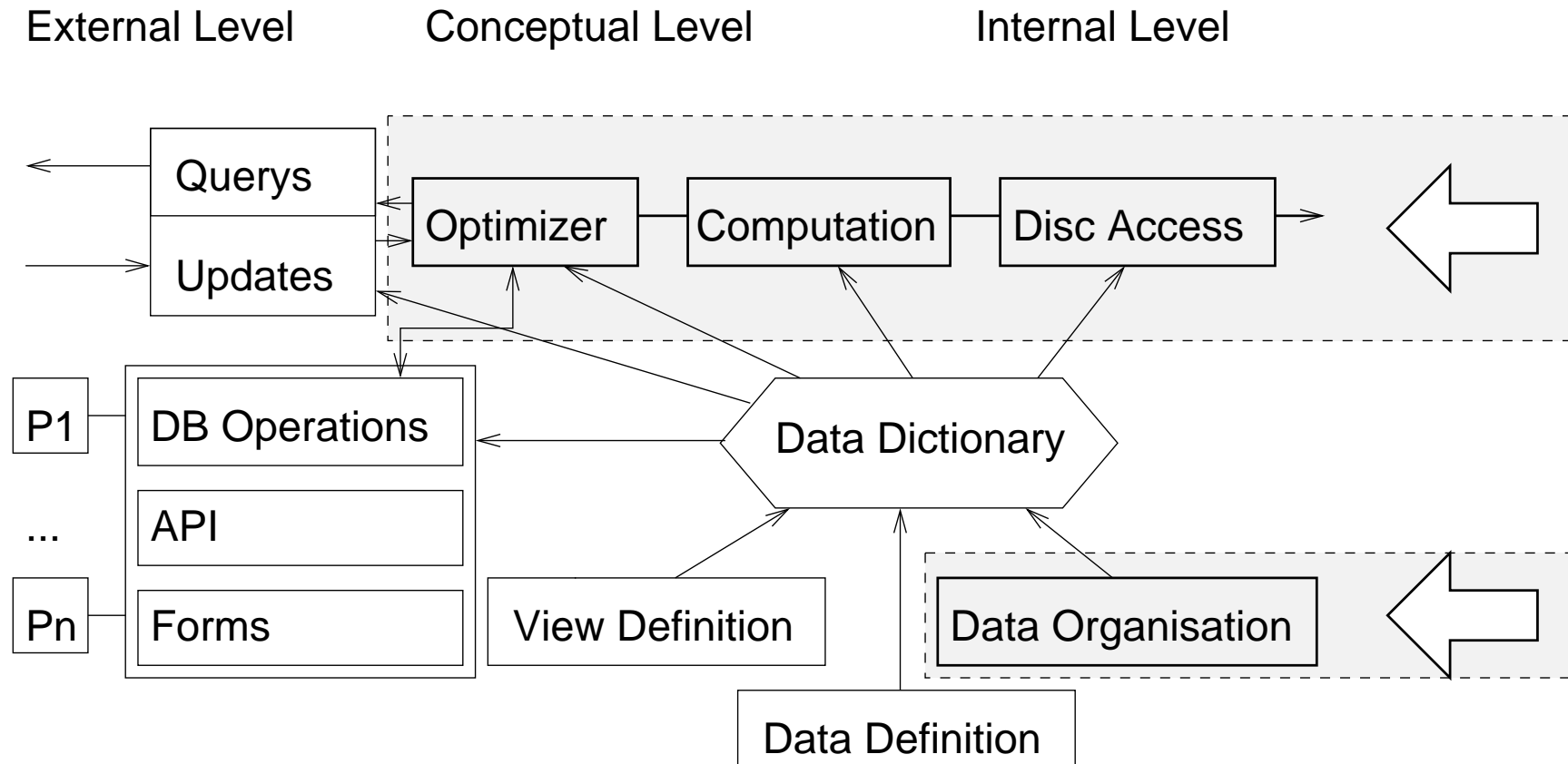
{sanschul,saake}@iti.cs.uni-magdeburg.de.

## 2. Architecture of Database Systems

---

- Problems covered in this chapter
- Layered architecture of a relational DBMS
- Hardware and operating system
- Buffer management
- Storage system
- (Internal) access system
- Data system

# Problems covered (in this chapter)



# Five-Layer Architecture (for DBMS)

---

- Based on the idea of Senko (1973)
- Further developments of Härder (1987)
- Realized in the development of the IBM prototyp *System R*
- Detailed description of transformation components
  - ◆ Step-wise transformation of queries/changes up to the point of access on storage devices
  - ◆ Definition of interfaces between components

# Five-Layer Architecture: Interfaces I

---

- Set-oriented interface (*SOI*)
  - ◆ declarative DML on tables, views and tuples
- Record-oriented interface (*ROI*)
  - ◆ Records, logical files, logical access paths (*Indices*)
  - ◆ navigating access on internal representation of relations
- Internal record interface (*IRI*)
  - ◆ Records, access paths
  - ◆ Manipulation of records and access paths

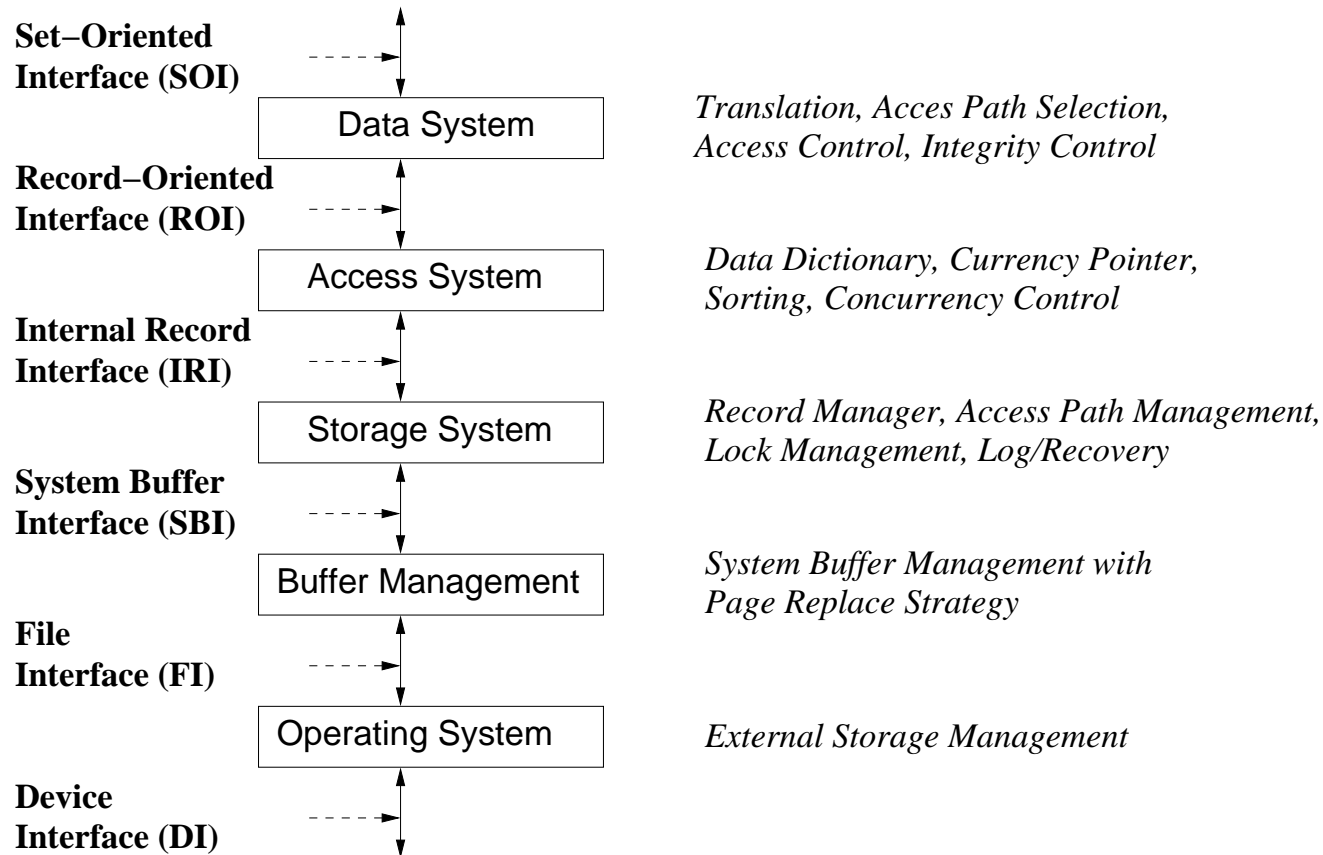
# 5-Layered-Architecture: Interfaces II

---

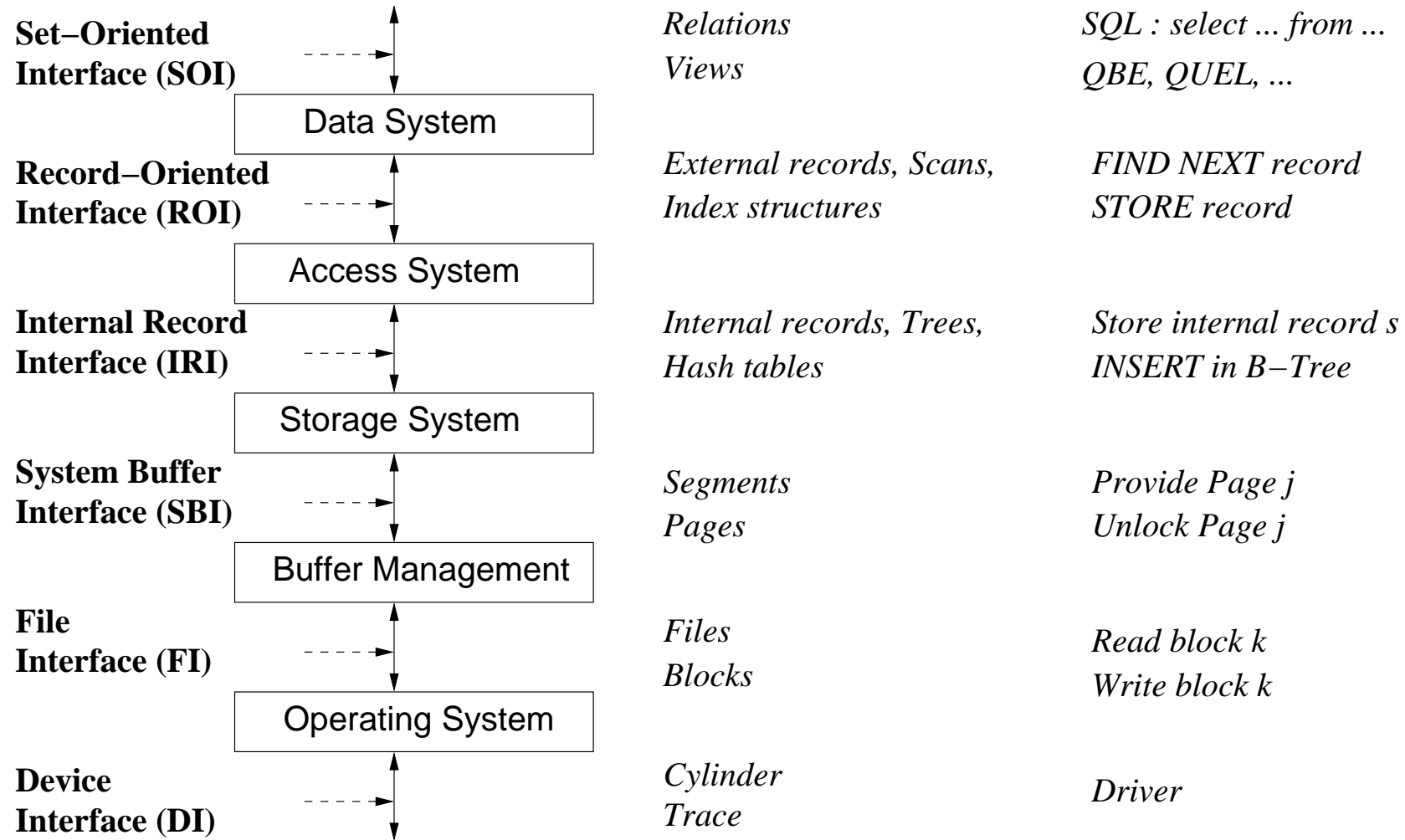
- System buffer interface (*SBI*)
  - ◆ Pages, page addresses
  - ◆ Release and allocation
- File interface (*DS*)
  - ◆ Fetch page, write page
- Device interface (*DI*)
  - ◆ Tracks, cylinder
  - ◆ Movement of disk heads

# 5-Layered-Architecture: Functionality

---



# 5-Layered-Architecture: Objects



# Explanations (I)

---

- Set-oriented interface **SOI**:
  - ◆ declarative data manipulation language (DML) on tables and views (e.g., SQL)
- is converted to the record-oriented interface **ROI** via the file system:
  - ◆ navigating access on internal representation of relations
  - ◆ manipulated objects: typed data sets and internal relations as well as logical access paths (Indices)
  - ◆ Functions of data system: Translation and optimization of SQL queries

# Explanations (II)

---

- is converted to the internal record interface **IRI** via the access system:
  - ◆ Uniform management of internal tuples (no typing)
  - ◆ Storage structures of access paths (concrete operations on B\*-Trees und hash tables) implemented
  - ◆ Multi user mode with transactions is realized

# Explanations (III)

---

- Data structures and operations (of IRI) are converted via storage system to internal pages of a virtual, linear address space
  - ◆ Manipulation of address space by operations of the system buffer interface **SBI**
  - ◆ Typical objects: internal pages, page addresses
  - ◆ Typical operations: Release and allocation of pages, page replace strategy, lock management, maintenance of log book (e.g., writing)
- mapping of internal pages to blocks of file interface **FI** via buffer management
  - ◆ Realization of FI operations on device interface carried out by OS

# Hardware and Operating System

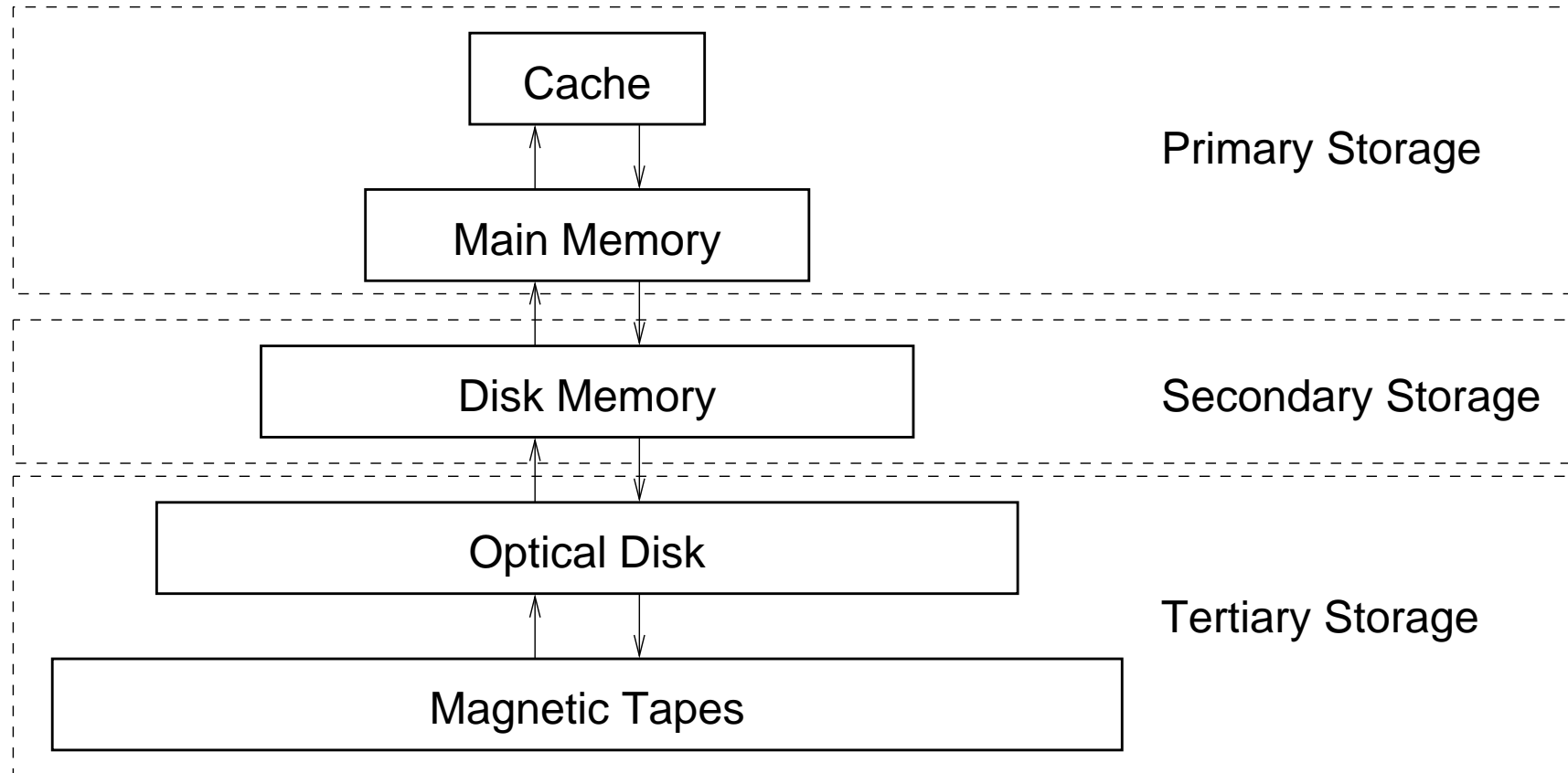
---

- OS layer: foundation for database related layer
  - ◆ Required: Driver (programs) for data access of storage media; caching mechanisms
- Processors and computer architecture: industrial standards (though: database computer available)
- Storage media: specific requirements, result in storage hierarchy

# Classical Storage hierarchy

---

Primary, secondary, and tertiary storage



# Characteristics of Storage Media

---

	<b>Primary</b>	<b>Secondary</b>	<b>Tertiary</b>
Speed	fast	slow	very slow
Price	expensive	low cost	cheap
Stability	volatile	non-volatile	non-volatile
Size	small	big	very big
Granularity	fine	coarse	coarse

# Primary Storage

---

- Cache and main memory
- very fast, fine-grained access on data: byte-by-byte addressable
- 32-Bit addressing: only  $2^{32}$  bytes directly addressable → size of primary storage very restricted
- High purchase costs per byte
- volatile and non-reliable storage medium

# Secondary Storage

---

- Online storage
  - ◆ Usually disk memory, non-volatile and reliable
  - ◆ much bigger, several gigabyte memory space per medium
  - ◆ Purchase costs are considerable less
  - ◆ Data not directly processable
  - ◆ Granularity for access more coarse-grained: Blocks, usually 512 bytes (or a multiple thereof)
  - ◆ Access vacancy: Factor  $10^5$  more slowly access
- Required: intelligent buffer management, good query optimization
- *However: minimized by new technologies, e.g., Flash*

# Tertiary Storage

---

- For long-term data backup (archiving) or short-term logging (journal) of database and database changes
- Secondary storage not useful (too small/expensive)
- several hundreds of gigabyte or even terabyte of data: aka offline storage, archive storage
- Usual: optical disks, magnetic tapes
- „Offline storage“ mostly removable medium
- Drawbacks: Access latency extremely high: Access on sequential medium, get magtape, insert magtape (even with automatization: magtape robots, Jukeboxes)

# Provided Services

---

- Driver (programs) → read(fetch)/write of blocks
- Assignment of blocks to pages
- Enhancement of block information with check sum → detection of write/read error
- Caching mechanisms, to keep and manage already read data in main memory
- Operations of file system of OS (usual: Only one file used by database systems)

# Buffer Management

---

- Management of required blocks (of secondary storage) in main memory
- Memory space only for limited amount of pages in main memory: *Buffer*
- Function of buffer management: Replacement of no longer required pages in buffer (page replace strategies)
- Difference: buffer managed by database system ↔ Cache on operating system level

# Buffer

---

- Size depends on main memory (usually up to 12 GB)
- Nevertheless only a minor fraction of database (less than 1%)
- All read/write operations on pages in the buffer
- Hence, buffer as a bottleneck
- Main memory (available) huge, database quite small  
⇒ whole database (e.g., during system start) in the buffer: *main memory databases*

# Buffer Management: Functions

---

- Allocation of memory space for pages
- Looking for/replacement of pages in the buffer
- Optimization of (work)load sharing between parallel transactions

# Page Access (Procedure) (I)

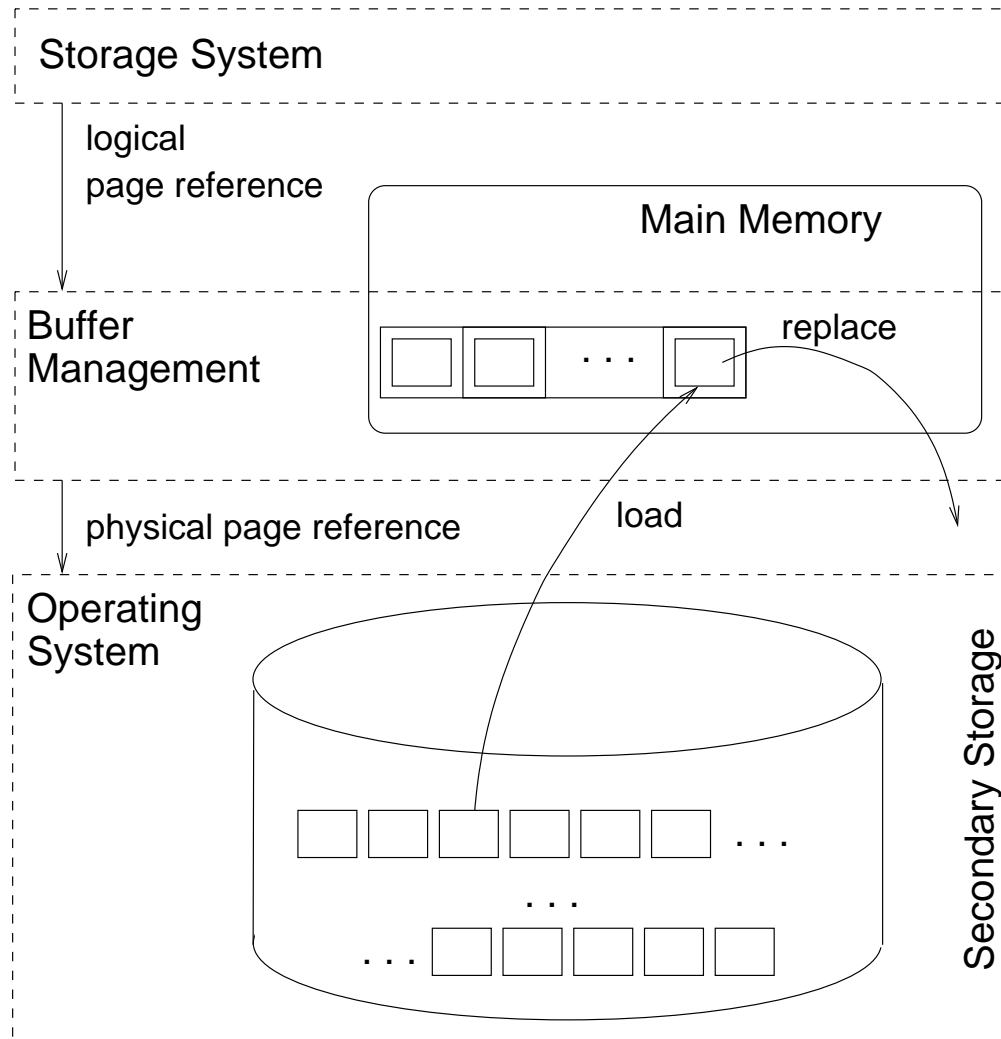
---

Page request of higher layer (storage system) (*logical page reference*)

- Requested page in the buffer: provided to the storage system
- Requested page not in the buffer (*page fault*): *physical page reference* from buffer management to operating system layer ; in general page replace in case of full buffer; if the page to be going to replaced has been changed → write to secondary storage

Effort per I/O operations: 2500 instructions in CPU; 15 to 30 ms for access on secondary storage

# Page Access (Procedure) (II)



# Storage System

---

Buffer: Pages (byte container)  $\leftrightarrow$  Storage system: internal data sets/records  $\leftrightarrow$  Access system: logical data records, internal tuples

<b>(Data) Structure</b>	<b>System Component</b>
Tuple	Data system
Internal tuple or logical data record	Access system
internal data record	Storage system

- Objects in storage system (represented) as internal records
- Auxiliary data, e.g., index entries, are represented as internal records as well

# Addressing of Records

---

- Problem: Update/Change of database  $\Rightarrow$  Efficient update of addresses
- Example: Internal records addressed with offset  $x$ , relatively to begin of page (internal record starts at byte  $x$ )  $\Rightarrow$  changes on this page have impact on used tuple address
- Improvement: *TID concept* (tuple identifier)
  - ◆ Address: Page number and offset in a list of tuple pointer at the begin/head of the page
  - ◆ Entry in pointer field  $\Rightarrow$  offset of record
  - ◆ Position of internal record within the page changed  $\Rightarrow$  only local entry in pointer field has to be changed; all addresses, used „outside“, remain stable

# Types of Records

---

- *Unspanned records* at most on one page; record too large for currently processed page  $\Rightarrow$  request the storage management for new page
- *Spanned records* can span several pages; record too large  $\Rightarrow$  Beginning of record on current page, overflow stored on new page
- *Records with fixed length*: for a certain type of tuples a definite number of bytes (for *string* all attribute values are stored with same number of bytes)
- *Records with variable length*: only the number of bytes really needed is stored (number of bytes per record is variable)

# Access System

---

- Abstraction from internal representation of records to pages
- *Logical data records, internal tuples*
- Internal tuples can be
  - ◆ Elements of a data representation of the conceptual relation *or*
  - ◆ Elements of an access path on the conceptual relation
- Internal tuples consist of fields (corresponding to the attributes of conceptual tuples)
- Typical operations of the access system: *Scans* (internal cursor on files or access paths)

# Index Files

---

- Access path on a file is a file as well: *Index file*
- Index consists of attribute values of conceptual relation (realizing a fast access on relation) as well as a list of tuple addresses
- Assigned addresses point to tuples, which contain the indexed attribute value
- Index on primary key  $\Rightarrow$  List of tuple addresses with singleton entries: *Primary index*
- Index arbitrary set of attributes: *Secondary key* (although attribute values must **not** possess any key characteristics)
- Index on secondary key: *Secondary index*

# Data Operations

---

- Insertion of a data record (**insert**)
- Removal of a data (**remove** oder **delete**)
- Modification of a data record (**modify**)
- Locking for and find a record (**lookup** or **fetch**)

# Data Operations: Types of *lookup*

---

- Given the attribute value for a certain field (column)  $\Rightarrow$  sought-after internal tuples possessing this value: *single-match query*.
- Given a combination of values for a certain combination of fields  $\Rightarrow$  sought-after all tuples, possessing these values:
  - ◆ Values for all values of (index) file: *exact-match query* (*single match query* is a special case)
  - ◆ Values only for some fields of (index) file: *partial-match query*
- Given a range of values for one or more attributes  $\Rightarrow$  sought-after all internal tuples, possessing attribute values within this range: *range query*

# Access on Data Records

---

- Data Records in a file depending on value of primary key
  - ◆ ordered or
  - ◆ hashed (scattered)stored  $\Rightarrow$  fast access over primary key
- Fast access over other attributes (secondary key) realized over index files by default

# File Organisation and Access Paths (I)

---

- *Primary key (PK) / Secondary key (SK):*
  - ◆ Access on PK (only one (distinct) tuple address per attribute value)
  - ◆ Access on SK (several tuple addresses per attribute value possible)
  - ◆ Usual: Primary key determines file organisation, secondary key determines access paths (index files)
- *one-dimensional / multidimensional:*
  - ◆ Support of access for a fixed combination of fields (*exact-match*)
  - ◆ Support of access for a variable combination of fields (*partial-match*)

# File Organisation and Access Paths II

---

- *static / dynamic*:
  - ◆ Static file organisation or access path  $\Rightarrow$  only useful for a fixed number of data records (to be managed)
  - ◆ Dynamic file organisation or access path  $\Rightarrow$  independent of number of data records (automatic, efficient adaptation to growing/shrinking data sets)

# Examples

---

## ■ B-Tree

- ◆ Dynamic, one-dimensional access paths
- ◆ in most database systems definable over several attributes of a file
- ◆ However: only one *exact-match* possible (on this combination of fields)

## ■ Hashing (classical hash method)

- ◆ Static and one-dimensional kind of file organisation
- ◆ Increasing amount of tuples  $\Rightarrow$  increasing number of collisions expected

# Data System (I)

---

- *Optimization*: Set-oriented query (SQL) has to be optimized by the system
  - ◆ Transformation of query expression into more efficient, processable expression (*Query Rewriting, conceptual or logical optimization*)
  - ◆ Selection of most efficient query expression by cost estimation (*cost-based optimization*)
  - ◆ *Internal optimization*: Selection of useful access paths and computation algorithms for query processing for every relational-algebraic operator

# Data System (II)

---

- *Selection of access paths*: determines internal structures to be exploited for query processing
- *Computation*: Selection of algorithm influences response time of a query
- Data system has to select the computation algorithms in cooperation with selection of access paths