Data Storage

Βάσεις Δεδομένων, 6ο Εξάμηνο, 2025

Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων

Σχολή Ηλεκτρολόγων Μηχ/κών και Μηχ/κών Η/Υ

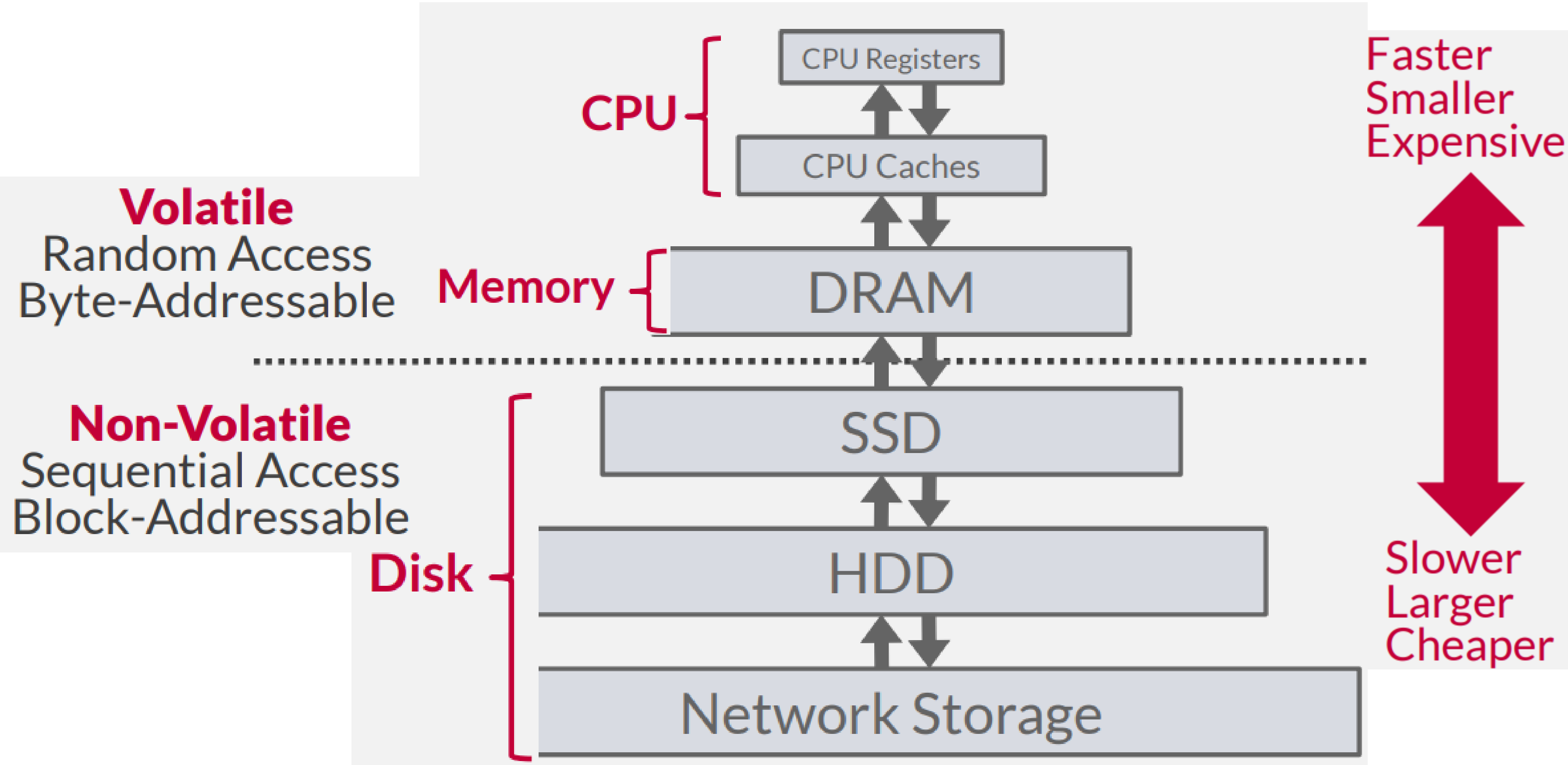
Database System Concepts, 7th Ed.

Includes some material from CMU 15-445/645, Fall 2023

Classification of Physical Storage Media

- Can differentiate storage into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage**:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory.
- Factors affecting choice of storage media include
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability

Storage Hierarchy



Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins.
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.

Latency Numbers Every Programmer Should Know

1 ns	L1 Cache Ref	**1 sec**
4 ns	L2 Cache Ref	**4 sec**
100 ns	DRAM	**100 sec**
16,000 ns	SSD	**4.4 hours**
2,000,000 ns	HDD	**3.3 weeks**
~50,000,000 ns	Network Storage	**1.5 years**
1,000,000,000 ns	Tape Archives	**31.7 years**

Source: https://colin-scott.github.io/personal_website/research/interactive_latency.html

Performance Measures (Cont.)

- **Disk block** is a logical unit for storage allocation and retrieval
  - 4 to 16 kilobytes typically
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
- **Sequential access pattern**
  - Successive requests are for successive disk blocks
    - Disk seek required only for first block
- **Random access pattern**
  - Successive requests are for blocks that can be anywhere on disk
    - Each access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks
- **DBMS will want to maximize sequential access**.
  - Algos try to reduce number of writes to random pages so that data is stored in contiguous blocks.
- **I/O operations per second (IOPS):**
  - Number of random block reads that a disk can support per second

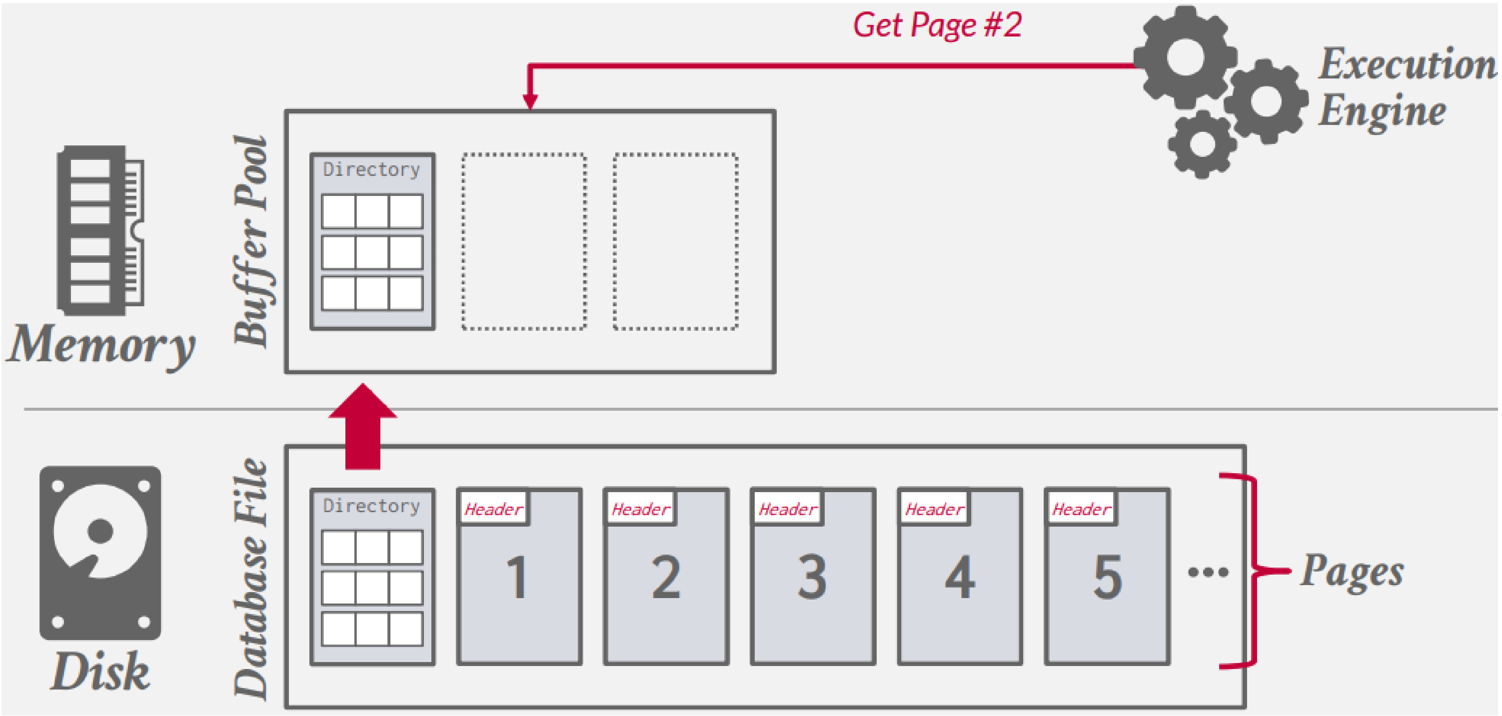
Performance Measures (Cont.)

- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
  - Typically 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

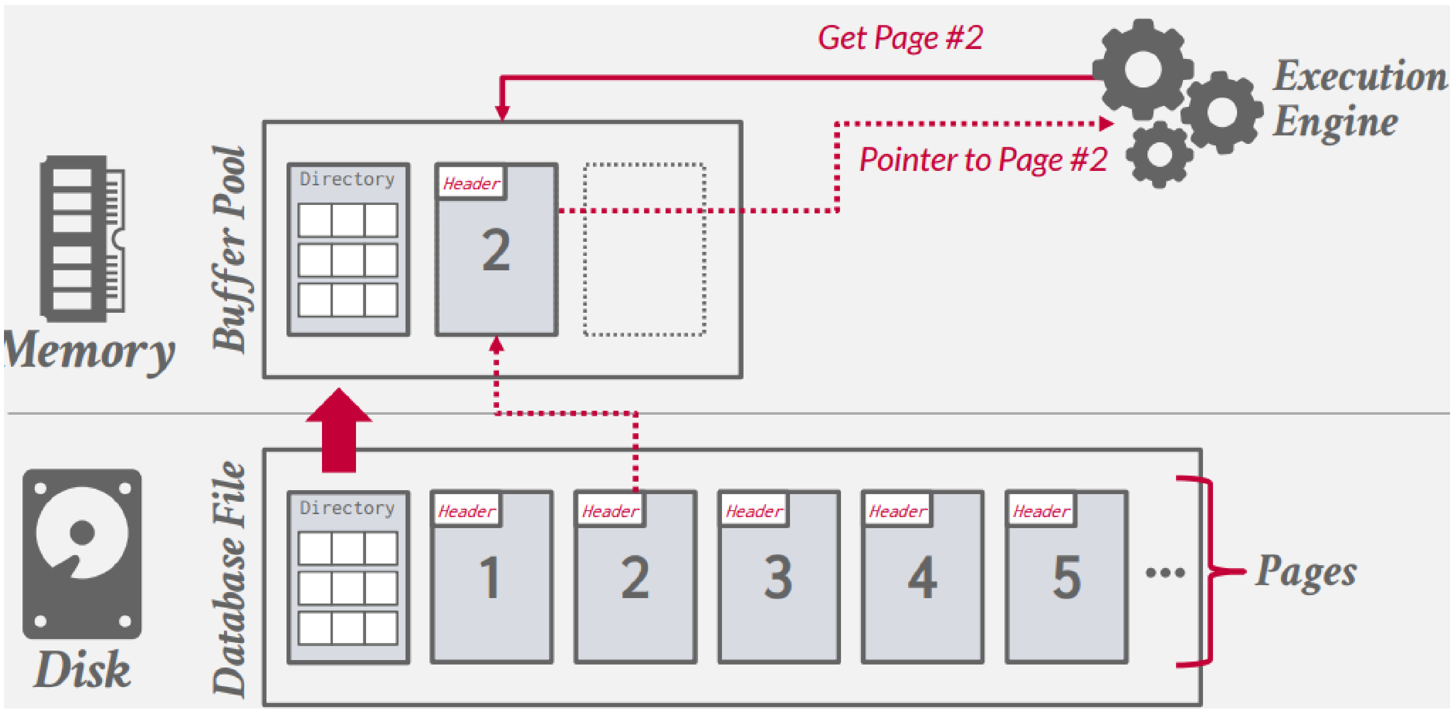
DBMS Design Goals

- Allow the DBMS to manage databases that exceed the amount of memory available.
- Reading/writing to disk is expensive, so it must be managed carefully to avoid large stalls and performance degradation.
- Random access on disk is usually much slower than sequential access, so the DBMS will want to maximize sequential access.

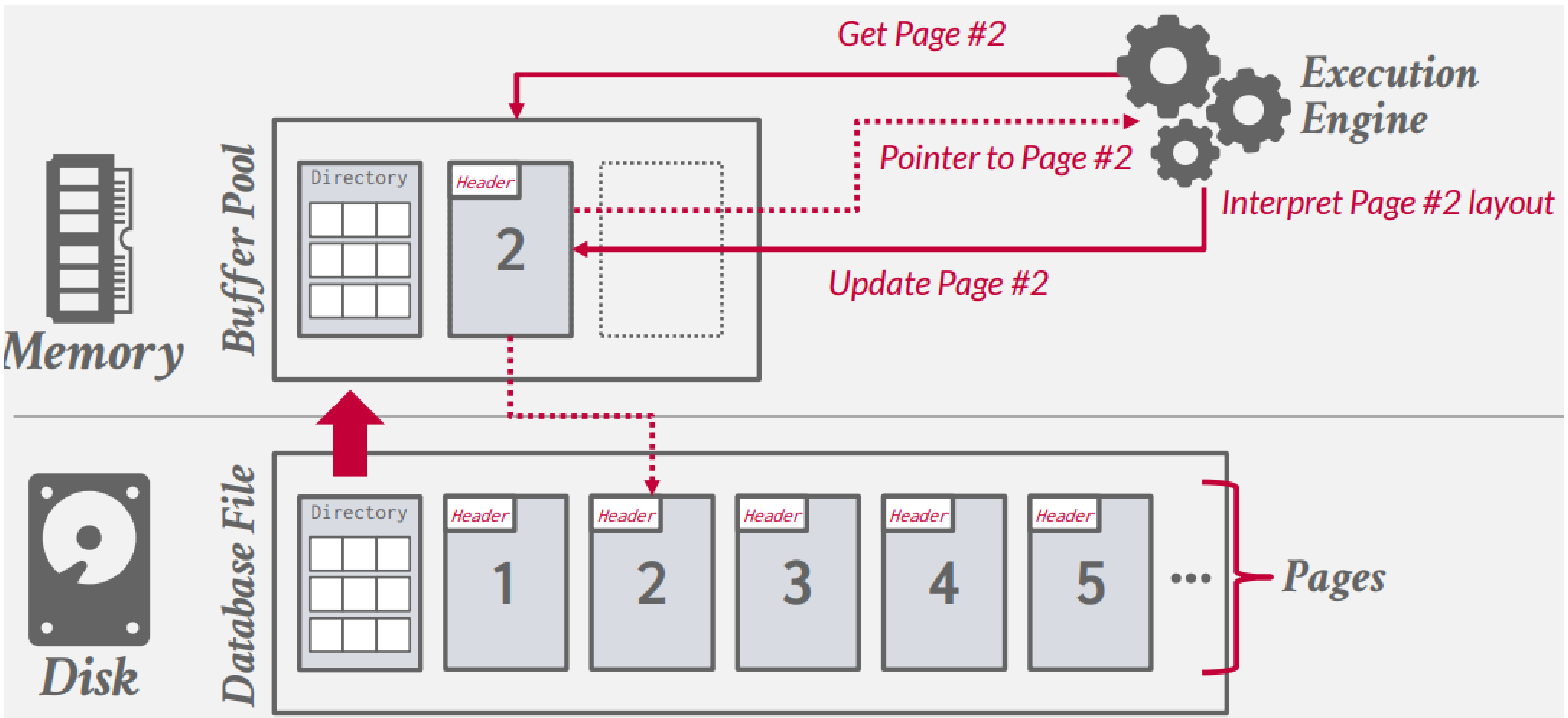
Disk-based DBMS



Disk-based DBMS

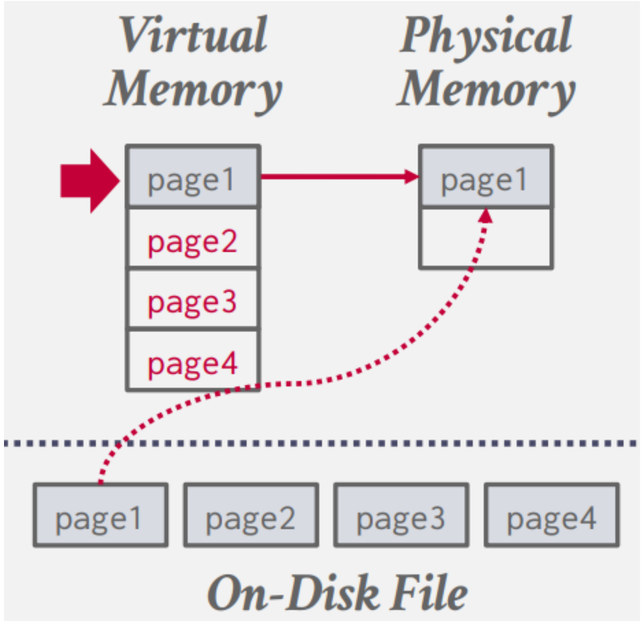


Disk-based DBMS



Why not use the OS?

- The DBMS can use memory mapping (**mmap**) to store the contents of a file into the address space of a program.
- The OS is responsible for moving file pages in and out of memory, so the DBMS does not need to worry.
- Problem #1: Transaction Safety
  - OS can flush dirty pages at any time.
- Problem #2: I/O Stalls
  - DBMS doesn't know which pages are in memory. The OS will stall a thread on page fault.
- Problem #3: Error Handling
  - Difficult to validate pages. Any access can cause a SIGBUS that the DBMS must handle.
- Problem #4: Performance Issues



Morale of the story

- The DBMS (almost) always wants to control things itself and can do a better job than the OS.
  - Flushing dirty pages to disk in the correct order.
  - Specialized prefetching.
  - Buffer replacement policy.
  - Thread/process scheduling.

- The OS is <u>not</u> your friend

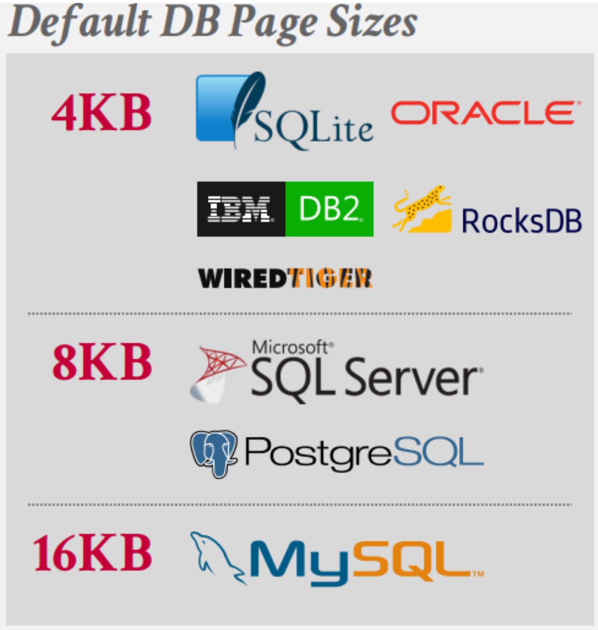
- Today's class: How the DBMS represents the database in files on disk

File Storage – Storage Manager

- The DBMS stores a database as one or more files on disk, typically in a proprietary format.
  - The OS does not know anything about the contents of these files.
- The **storage manager** is responsible for maintaining the database files.
  - Some do their own scheduling for reads and writes to improve spatial and temporal locality of pages.
- It organizes the files as a collection of pages.
  - Tracks data read/written to pages.
  - Tracks the available space.
- A DBMS typically does not maintain multiple copies of a page on disk.

Database Pages (or Blocks)

- A page is a fixed-size block of data.
  - It can contain tuples, meta-data, indexes, log records, …
  - Most systems do not mix page types.
  - Some systems require a page to be self-contained.
- Each page is given a unique identifier.
  - The DBMS uses an indirection layer to map page IDs to physical locations.
- There are three different notions of "pages" in a DBMS:
  - Hardware Page (usually 4KB)
  - OS Page (usually 4KB, x64 2MB/1GB)
  - Database Page (512B-32KB)
- A hardware page is the largest block of data that the storage device can guarantee failsafe writes.



File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records.* A record is a sequence of *fields*.
- Files are logically divided into **blocks (or pages).**
- One approach:
  - Assume record size is fixed.
  - Each file has records of one particular type only.
  - Different files are used for different relations.

  This case is easiest to implement; will consider variable length records later
- Assume that records are smaller than a disk block

Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte *n* ∗ *i*, where *n* is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - **move records *i* + 1, . . ., *n* to *i, . . . , n* – 1**
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

Record 3 deleted

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - **move record *n* to *i***
  - do not move records, but link all free records on a *free list*

Record 3 deleted and replaced by record 11

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

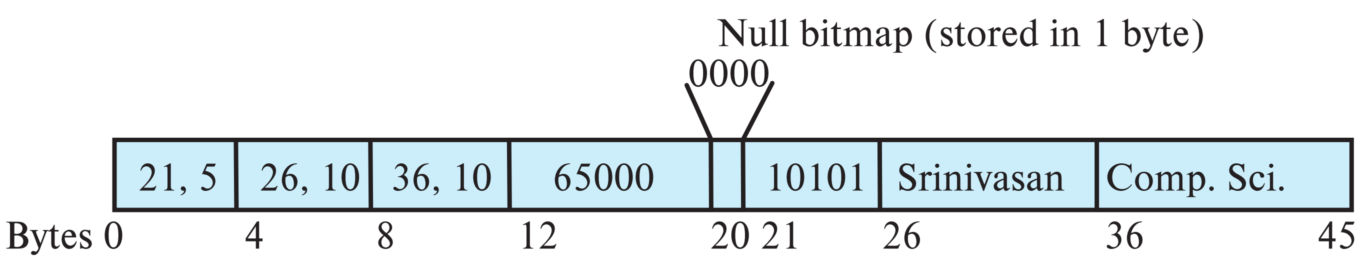
Fixed-Length Records

- Deletion of record *i*: alternatives:
  - move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1
  - move record *n* to *i*
  - **do not move records, but link all free records on a *free list***

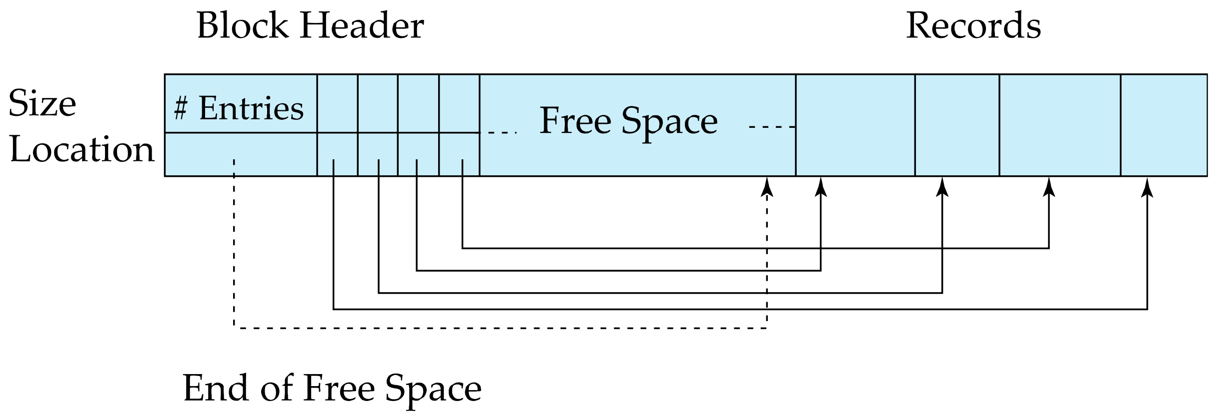
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models).
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

Storing Large Objects

- E.g., blob/clob types
- Records must be smaller than pages
- Alternatives:
  - Store as files managed by database
  - Store as files in file systems
    - Issues with file deletions (foreign key constraints), authorization

Organization of Records in Files (page layout)

- For any page storage architecture, we need to decide how to organize the data inside of the page.
  - We are assuming that we are only storing tuples in a row-oriented storage model
- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- In a **multitable clustering file organization,** records of several different relations can be stored in the same file
  - Motivation: store related records in the same block to minimize I/O
- **B^+-tree file organization**
  - Ordered storage even with inserts/deletes
  - More on this in the next lecture (Chapter 14 in the book)
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed
  - More on this in Chapter 14

Heap File Organization

- Records can be placed anywhere in the file where there is free space
- Records usually do not move once allocated
- Important to be able to efficiently find free space within file
- **Free-space map**
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free
  - In example below, 3 bits per block, value divided by 8 indicates fraction of block that is free

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

  - Can have second-level free-space map
  - In example below, each entry stores maximum from 4 entries of first-level free-space map

4	7	2	6

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a **search-key**

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an **overflow block**
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Multitable Clustering File Organization (cont.)

- Good for queries involving *department* $\bowtie$ *instructor*, and for queries involving one single department and its instructors
- Bad for queries involving only *department*
- Results in variable size records
- Can add pointer chains to link records of a particular relation

Partitioning

- **Table partitioning**: Records in a relation can be partitioned into smaller relations that are stored separately
- E.g., *transaction* relation may be partitioned into *transaction_2018, transaction_2019, etc.*
- Queries written on *transaction* must access records in all partitions
  - Unless query has a selection such as *year*=2019, in which case only one partition in needed
- Partitioning
  - Reduces costs of some operations such as free space management
  - Allows different partitions to be stored on different storage devices
    - E.g., *transaction* partition for current year on SSD, for older years on magnetic disk

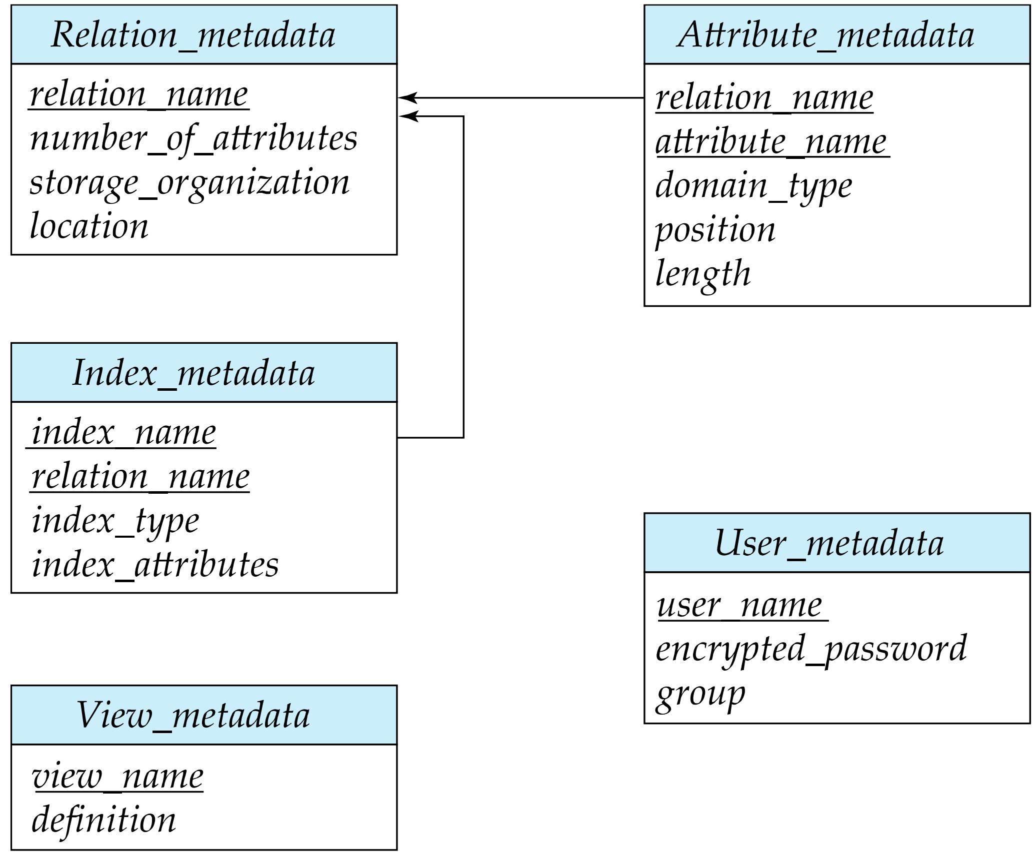
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as:

- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices (Chapter 14)

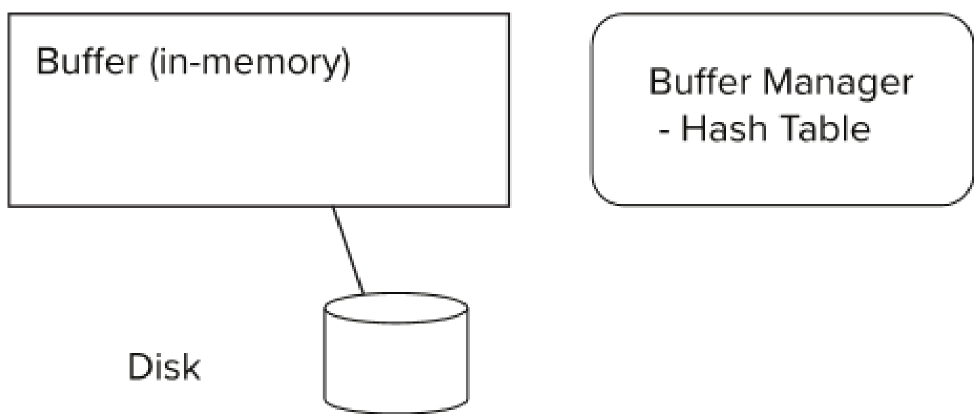
Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



Storage Access

- Blocks are units of both storage allocation and data transfer.
- The Database System seeks to **minimize the number of block transfers between the disk and memory**. We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
- **Buffer** – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory.



Buffer Manager

- Programs call on the buffer manager when they need a block from disk.
  - If the block is already in the buffer, buffer manager returns the address of the block in main memory
  - If the block is not in the buffer, the buffer manager
    - Allocates space in the buffer for the block
      - Replacing (throwing out) some other block, if required, to make space for the new block.
      - Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
    - Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester.

Buffer Manager

- **Buffer replacement strategy**
- **Pinned block:** memory block that is not allowed to be written back to disk
  - **Pin** done before reading/writing data from a block
  - **Unpin** done when read /write is complete
  - Multiple concurrent pin/unpin operations possible
    - Keep a pin count, buffer block can be evicted only if pin count = 0
- **Shared and exclusive locks on buffer**
  - Needed to prevent concurrent operations from reading page contents as they are moved/reorganized, and to ensure only one move/reorganize at a time
  - Readers get shared lock, updates to a block require exclusive lock
  - **Locking rules:**
    - Only one process can get exclusive lock at a time
    - Shared lock cannot be concurrently with exclusive lock
    - Multiple processes may be given shared lock concurrently

Buffer-Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
  - Idea behind LRU – use past pattern of block references as a predictor of future references
  - LRU can be bad for some queries
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
- Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable
- Example of bad access pattern for LRU: when computing the join of 2 relations r and s by a nested loop

  for each tuple *tr* of *r* do
    for each tuple *ts* of *s* do
      if the tuples *tr* and *ts* match …

Buffer-Replacement Policies (Cont.)

- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy** – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer

Column-Oriented Storage

- Also known as **columnar representation**
- Store each attribute of a relation separately
- Example

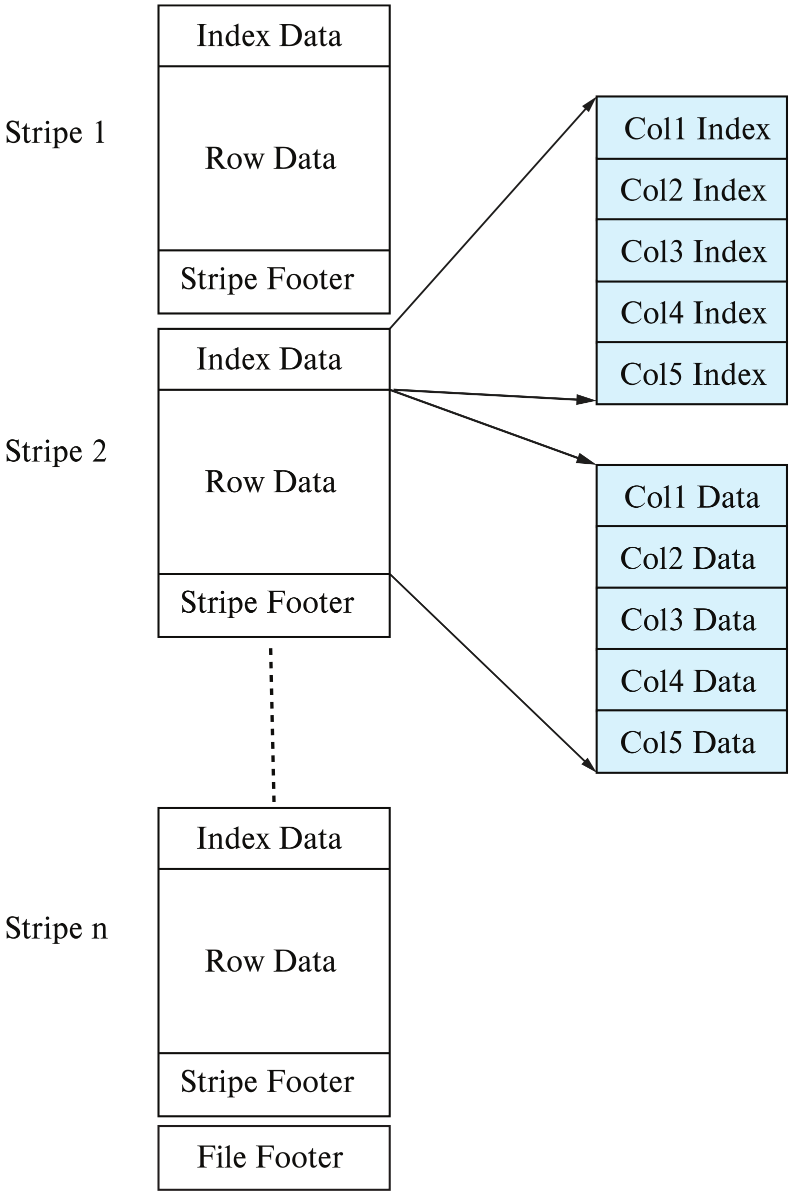
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Columnar Representation

- Benefits:
  - Reduced I/O if only some attributes are accessed
  - Improved CPU cache performance
  - Improved compression
  - **Vector processing** on modern CPU architectures
- Drawbacks
  - Cost of tuple reconstruction from columnar representation
  - Cost of tuple deletion and update
  - Cost of decompression
- Columnar representation found to be more efficient for decision support than row-oriented representation
- Traditional row-oriented representation preferable for transaction processing
- Some databases support both representations
  - Called **hybrid row/column stores**

Columnar File Representation

- ORC and Parquet: file formats with columnar storage inside file
- Very popular for big-data applications
- Orc file format shown on the right
  - Stripe: A sequence of tuples occupying several hundred megabytes
  - The Row Data area stores a compressed representation of the values for the first column, followed by the second column, and so on
- Some column-store systems allow groups of columns to be stored together, instead of breaking up each column into a different file



Storage Organization in Main-Memory Databases

- Can store records directly in memory without a buffer manager
- Column-oriented storage can be used in-memory for decision support applications
  - Compression reduces memory requirement
- Records are directly allocated in memory, and measures are taken to avoid memory fragmentation over time.
- For column-oriented storage, logical arrays are divided into multiple physical arrays with an indirection table to avoid data reallocation during appends.

