Storing Data: Disks and Files

Some Announcements

Myexperience Surveys are available for this course

- T1 2022 Survey Period: **11 - 28 April**
  - *"The survey is opened to students after they have completed the majority of the given course, so responses are based on their experience across the trimester"*

Pending: Detailed **Examination time** to be released next week

- So far the plan is around 14:00 – 17:00

Some Announcements (2)

Course questions:

1. I felt part of a learning community.
2. The assessment tasks were relevant to the course content.
3. What were the best things about this course? [open comments]
4. What could be improved? [open comments]

etc.

Teaching questions:

1. This teacher encouraged student participation.
2. The best features of this person's teaching were... [Open comments]
3. This person's teaching could be improved by… [Open comments]

etc.

Memory Hierarchy

- ***Primary Storage***: main memory.

    fast access, expensive.

- ***Secondary storage:*** hard disk.

    slower access, less expensive.

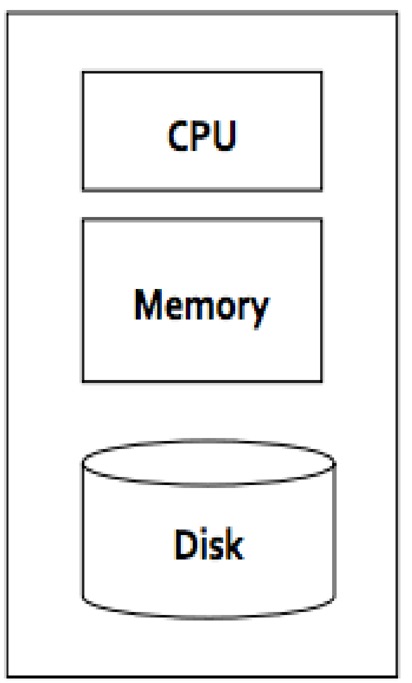
- ***Tertiary storage***: tapes, cd, etc.

    slowest access, cheapest.

Primary Storage

Main memory:

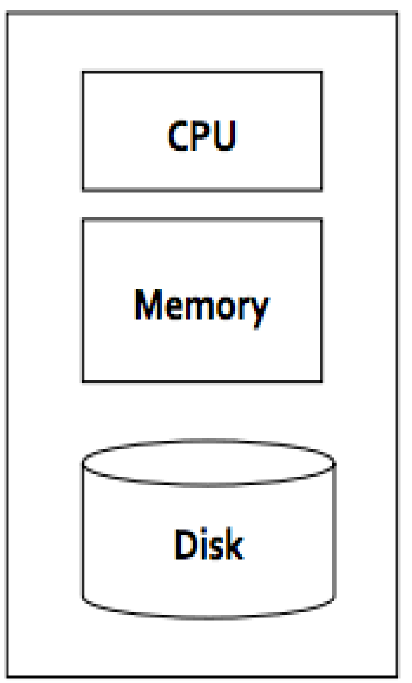
- Fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
- Generally too small (or too expensive) to store the entire database
- **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs.



Secondary Storage

Magnetic-disk

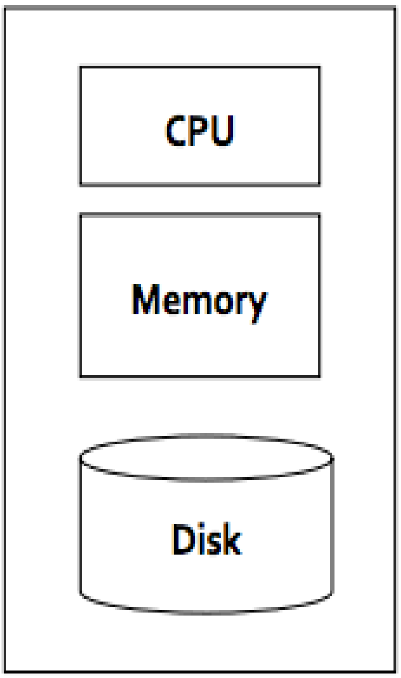
- Data is stored on spinning disk, and read/written magnetically
- Primary medium for the long-term storage of data; typically stores entire database.
- **Data must be moved from disk to main memory for access, and written back for storage**
  - **Much slower access than main memory**
- **Direct-access** – possible to read data on disk in any order.
- Survives power failures and system crashes
  - Recall: disk failure can destroy data, but is rare



CPU cost vs I/O cost

The implementation Issues

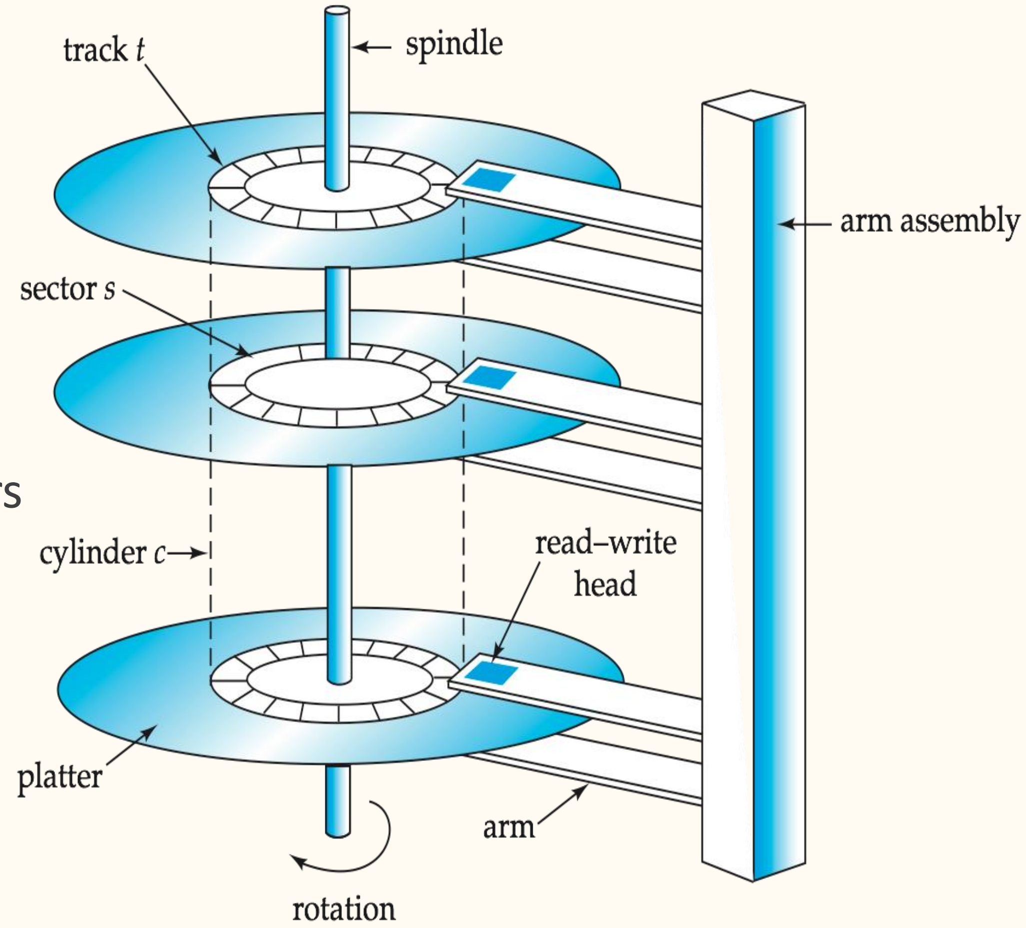
- There are two main costs, CPU cost and I/O (Input/Output) cost.
  - CPU cost is to process data in main memory.
  - I/O cost is to read/write data from/into disk.
- The dominating cost is I/O cost. For query processing in DBMS, CPU cost can be ignored.
- The key issue is to reduce I/O cost.
  - It is to reduce the number of I/O accesses.
- What is I/O cost?
  - A block (or page) to be read/written from/into disk is one I/O access (or one disk-block/page access).



Magnetic Hard Disk Mechanism

Characteristics of disks:

- collection of platters
- each platter = set of tracks
- each track = sequence of sectors (blocks)



NOTE: Diagram simplifies the structure of actual disk drives

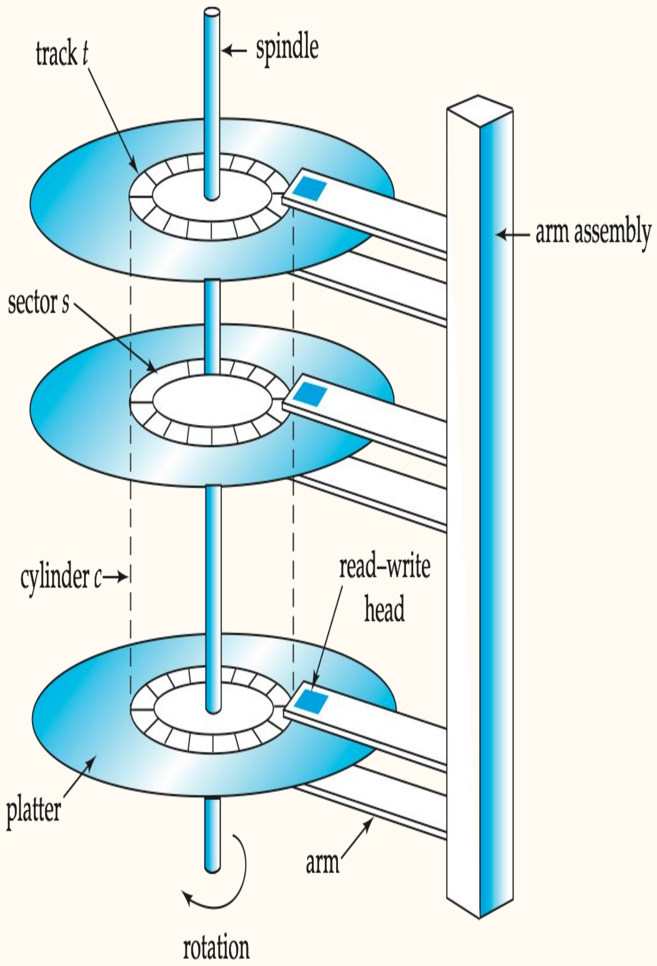
Performance Measures of Disks

Access time – the time it takes from when a read or write request is issued to when data transfer begins.

- **Seek time** – time it takes to reposition the arm over the correct track.
  - Average seek time is 1/2 the worst case seek time.
  - 4 to 10 milliseconds on typical disks
- **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
  - Average latency is 1/2 of the worst case latency.
  - 4 to 11 milliseconds on typical disks (5400 to 15000 **rpm** (revolutions per minute))

Data-transfer rate – the rate at which data can be retrieved from or stored to the disk.

- 25 to 100 MB per second max rate.



Disks (2)

Characteristics of disks:

- transfer unit: 1 block (e.g. 512B, 1KB)
- access time depends on proximity of heads to required block access
- access via block address (p, t, s)

Disk Space Management

- *Improving Disk Access*:
  - ➢ Use knowledge of data access patterns.
    - E.g. two records often accessed together: put them in the same block (clustering)
    - E.g. records scanned sequentially: place them in consecutive sectors on same track
  - ➢ Keep Track of Free Blocks
    - Maintain a list of free blocks
  - Using OS File System to Manage Disk Space
    - extend OS facilities, but not rely on the OS file system.
    - (portability and scalability)

Storage Access

Data must be in memory for the DBMS to operate on it.

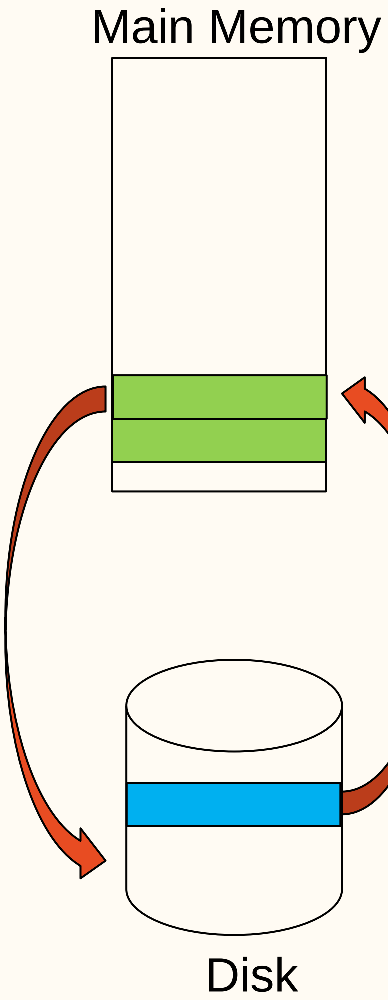
A database file is partitioned into fixed-length storage units called **blocks**. Blocks are units of both storage allocation and data transfer.

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.



Disk-Block Access

1. Smallest process unit is **a block**: If a single record in a block is needed, the entire block is transferred.
2. Data are transferred between disk and main memory in units of blocks.
3. A relation is stored as a **file** on **disk**.
4. A file is a sequence of blocks, where a block is a fixed-length storage unit.
5. A block is also called a page.

Block Movements

The database is partitioned into fixed-length storage units called blocks.

Blocks are the units of data transfer to and from disk and may contain several data items.

We shall assume that no data item spans two or more blocks. This assumption is realistic for most data-processing applications, such as a bank or a university.

Transactions input information from the disk into main memory and then output the information back onto the disk.

The input and output operations are done in block units.

The blocks residing on the disk are referred to as **physical blocks**; the blocks residing temporarily in main memory are referred to as **buffer blocks**.

The area of memory where blocks reside temporarily is called the disk buffer.

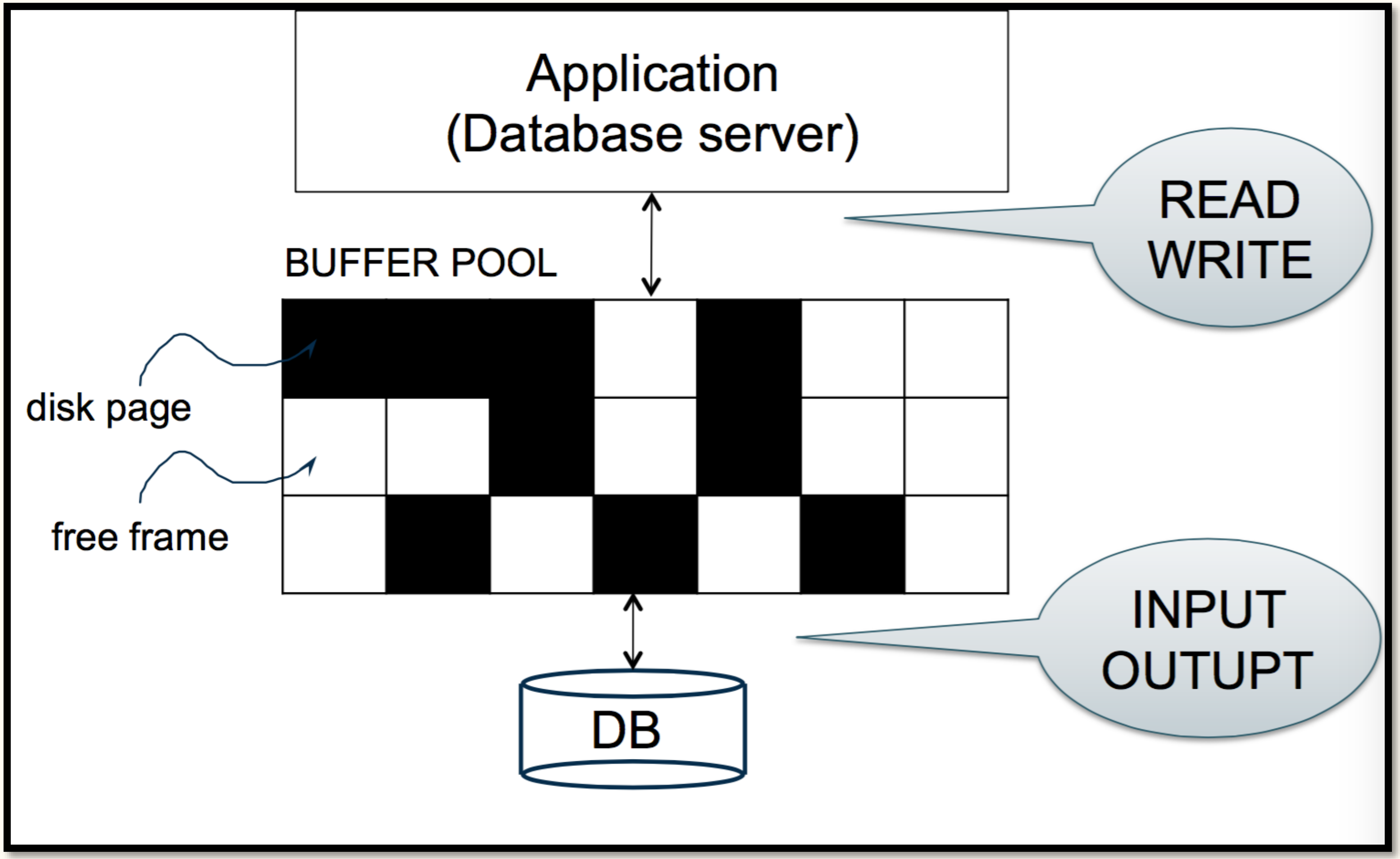
Blocks

Block movements between disk and main memory are initiated through the following two operations:

1. input(B) transfers the physical block B to main memory.

2. output(B) transfers the buffer block B to the disk and replaces the appropriate physical block there.

Buffer Management in a DBMS



Buffer Manager

Manages traffic between disk and memory by maintaining a ***buffer pool*** in main memory.

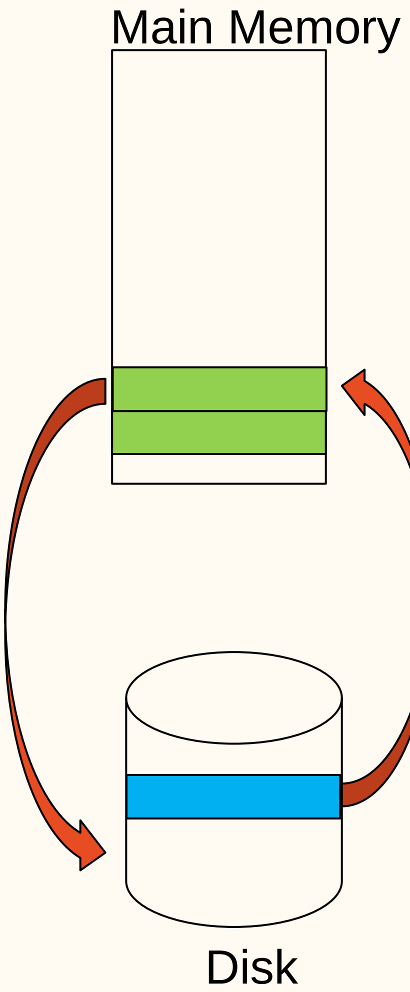
Buffer pool

- Collection of *page slots* (frames) which can be filled with copies of disk block data.
- One page = 4096 Bytes = One block

Buffer Manager

Programs call on the buffer manager when they need a block from disk.

1. If the block is already in the buffer, buffer manager returns the address of the block in main memory
2. If the block is not in the buffer, the buffer manager
   1. Allocates space in the buffer for the block
      1. **Replacing** (throwing out) some other block, if required, to make space for the new block.
      2. 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.
   2. Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester.



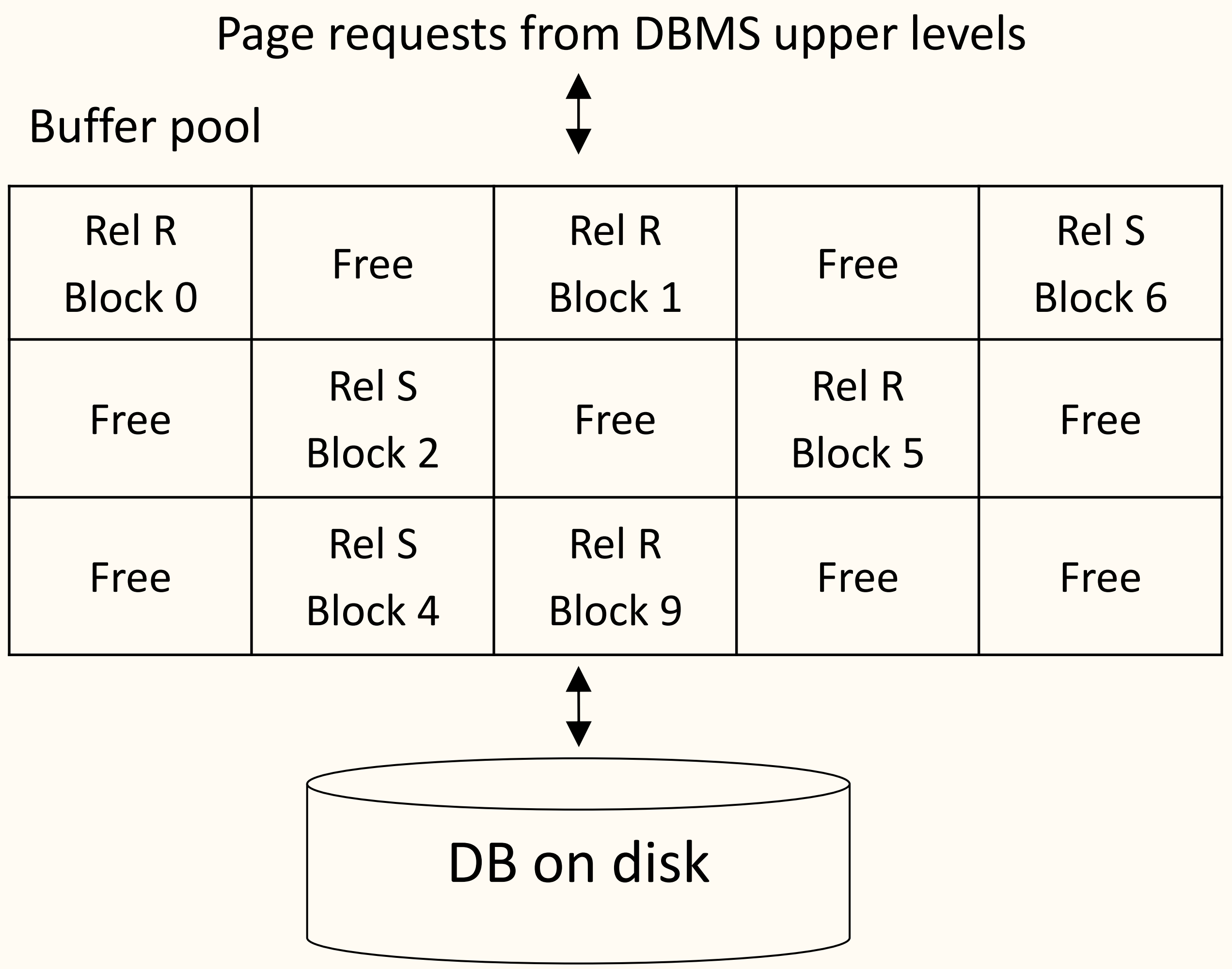
Read

1. Compute the data block that contains the item to be read
2. Either
   - find a buffer containing the block, or
   - read from disk into a buffer
3. **Copy** the value from the buffer.

Write

1. Compute the disk block containing the item to be written
2. Either
   - find a buffer containing the block, or
   - read from disk into a buffer
3. **Copy** the new value into the buffer
4. At some point (maybe later), write the buffer back to disk.

Buffer Pool


Page requests from DBMS upper levels
Buffer pool
Rel R Block 0
Free
Rel R Block 1
Free
Rel S Block 6
Free
Rel S Block 2
Free
Rel R Block 5
Free
Free
Rel S Block 4
Rel R Block 9
Free
Free
DB on disk

Buffer Pool

- The ***request_block*** operation
  - ➢ If block ***is*** already in buffer pool:
    - no need to read it again
    - use the copy there (unless write-locked)
  - ➢ If block ***is not*** in buffer pool yet:
    - need to read from hard disk into a free frame
    - if no free frames, need to remove block using *a buffer replacement policy*.
- The ***release_block*** function indicates that block is no longer in use
  - ➢ good candidate for removal / replacing

Buffer Pool

For each frame, the manager keeps track of:

- whether it is currently in use
- whether it has been modified since loading (***dirty bit***)
- how many transactions are currently using it (***pin count***)
- (in some implementations) time-stamp for most recent access

Buffer Pool

- **The *release_block* Operation**
  - Decrement pin count for specified page.

    No real effect until replacement required.
- **The *write_block* Operation**
  - Updates contents of page in pool
  - Set dirty bit on
  - (Doesn’t actually write to disk, until been replaced, or forced to commit)
- **The *force_block* operation**
  - "commits" by writing to disk.

Buffer-Replacement Policies

- Least Recently Used (LRU)
  - release the frame that has not been used for the longest period.
  - intuitively appealing idea but can perform badly

Buffer-Replacement Policies

Most systems replace the block **least recently used** (**LRU** strategy)

Idea behind LRU – use **past** pattern of block references as a predictor of **future** references

- *The one I used just now may be needed very soon*.

An example: department (2 blocks) and instructor (4 blocks)

A buffer has 4 blocks.

An SQL

- select * from department, instructor
  where department.dept_name = instructor.dept_name

department

D1 D2	D3 D4

instructor

D1 D4	D2 D3	D1 D2	D1 D4

Buffer-Replacement Policies

SQL:

select * from department, instructor
where department.dept_name = instructor.dept_name

Algorithm:

for each tuple *t* of ***department*** do
  for each tuple *s* of ***instructor*** do
    if the t and s have the same department name
      output all the attributes

Department

D1 D2	D3 D4
b1	**b2**

Instructor

D1 D4	D2 D3	D1 D2	D1 D4
b1	**b2**	**b3**	**b4**

Buffer

b1			
b1	**b1**	**b2**	**b3**

Now we need to access b4. But we need to replace it. Which do we replace? b1 !

Buffer-Replacement Policies

- Least Recently Used (LRU)
  - release the frame that has not been used for the longest period.
  - intuitively appealing idea but can perform badly
- First in First Out (FIFO)
  - need to maintain a queue of frames
  - enter tail of queue when read in

Buffer-Replacement Policies

- Least Recently Used (LRU)
  - ➢ release the frame that has not been used for the longest period.
  - ➢ intuitively appealing idea but can perform badly
- First in First Out (FIFO)
  - ➢ need to maintain a queue of frames
  - ➢ enter tail of queue when read in
- Most Recently Used (MRU):
  - ➢ release the frame used most recently

Buffer-Replacement Policies

- Least Recently Used (LRU)
  - release the frame that has not been used for the longest period.
  - intuitively appealing idea but can perform badly
- First in First Out (FIFO)
  - need to maintain a queue of frames
  - enter tail of queue when read in
- Most Recently Used (MRU):
  - release the frame used most recently

◦ No one is guaranteed to be better than the others.

Quite dependent on the application.

Example1:

Data pages: P1, P2, P3, P4

Queries:

Q1: read P1; Q2: read P2;

Q3: read P3; Q4: read P1;

Q5: read P2;

Buffer:

P1 Q4	**P2** Q5	**P3** Q3

Q6: read P4:

- **LRU**: Replace P3
- MRU: Replace P2
- FIFO: Replace P1
- Random: randomly choose one buffer to replace

Example 2:

Data pages: P1, P2, …, P11

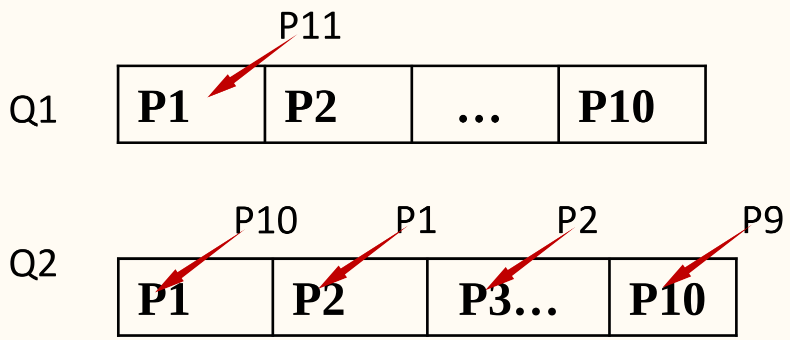
Queries:

Q1: read P1, P2,…, P11;

Q2, read P1, P2,…, P11;

Q3: Read P1, P2,…,P11

Buffer: 10 pages LRU/FIFO:



Behaviour : We need to get in/out every page

MRU: performs the best in this case.

No Force Policy

Force policy:

- Force-output all modified blocks to disk when they commit.

No-force policy:

- Allows a transaction to commit even if it has modified some blocks that have not yet been written back to disk.
- All the recovery algorithms described so far works even with a no-force policy.
- As a result, the standard approach taken by most systems is the no-force policy.

Steal Policy

No-steal policy:

- Blocks modified by a transaction that is still active should not be written to disk.

Steal policy:

- Allows the system to write modified blocks to disk even if the transactions that made those modifications have not all committed.
- If the write-ahead logging rule is followed, all the recovery algorithms we study in the chapter work correctly even with the steal policy.
- Standard approach taken by most systems is the steal policy

Block (Page) Formats

A block is a collection of *slots*.

Each slot contains a record.

A record is identified by rid = <*page id, slot number*>.

Record Formats

Records are stored within fixed-length blocks.

- ***Fixed-length***: each field has a fixed length as well as the number of fields.

33357462	Neil Young	Musician	0277
4 bytes	40 bytes	20 bytes	4 bytes

  - Easy for intra-block space management.
  - Possible waste of space.

- ***Variable-length***: some field is of variable length.

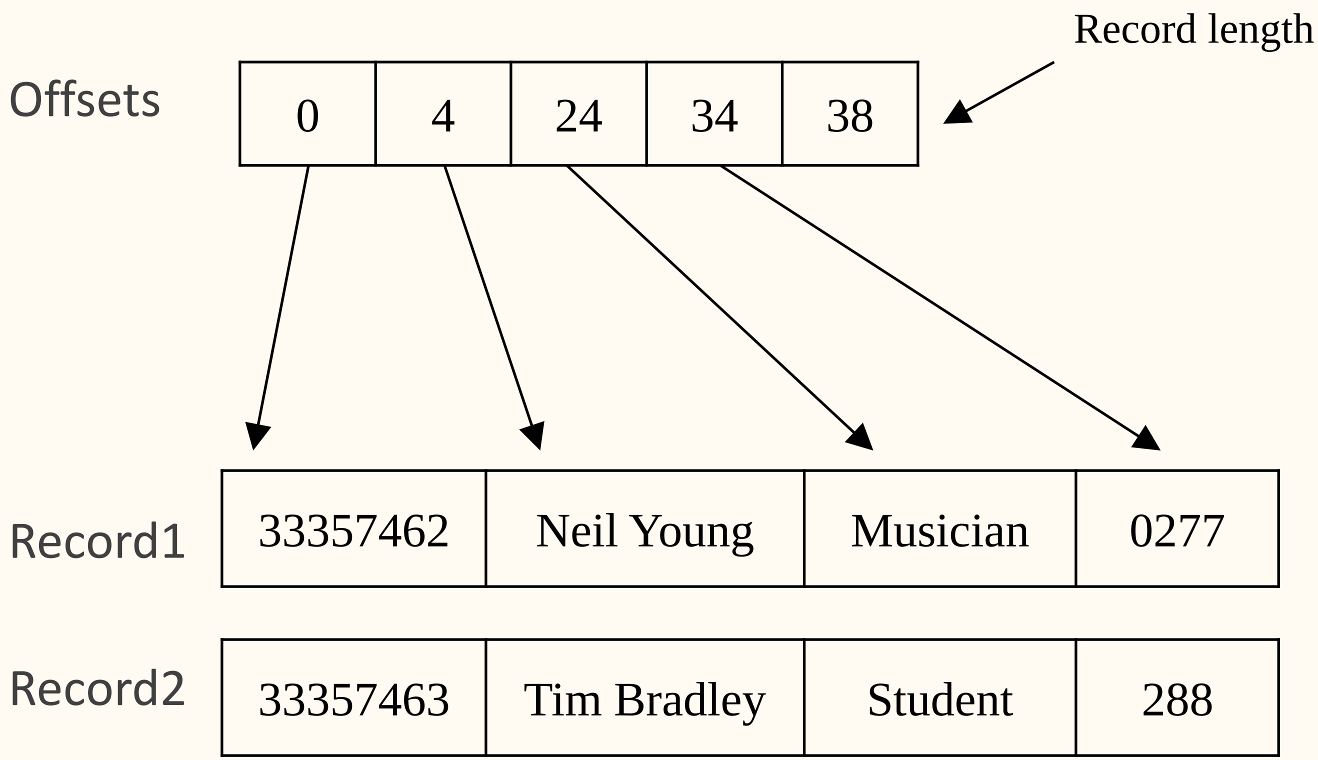
33357462	Neil Young	Musician	0277
4 bytes	10 bytes	8 bytes	4 bytes

  - complicates intra-block space management
  - does not waste (as much) space.

Fixed-Length

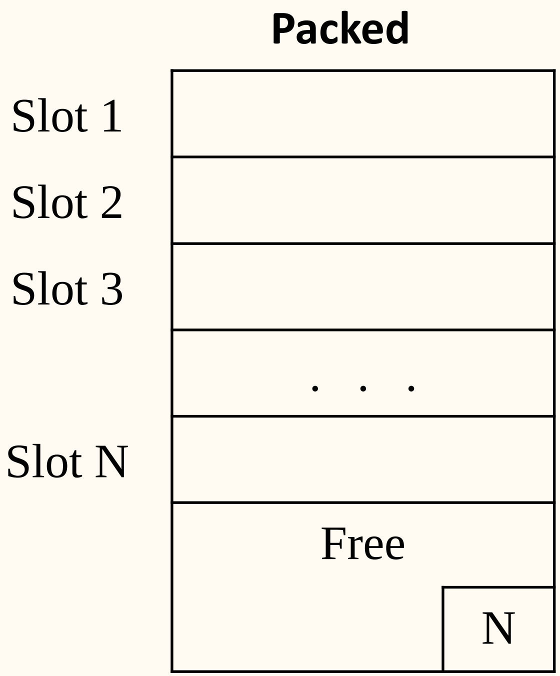
Encoding scheme for fixed-length records:

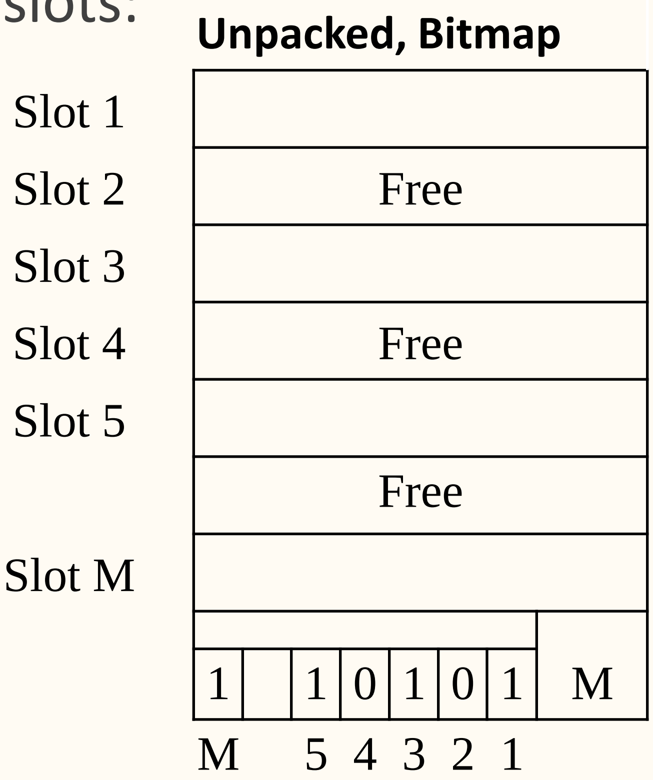
- length + offsets stored in header



Fixed-Length Records

For fixed-length records, use record slots:





Insertion: occupy first free slot; packed more efficient.

Deletion: (a) need to compact, (b) mark with 0; unpacked more efficient.

Fixed-Length Records

Simple approach:

- Store record i starting from byte $n \times (i - 1)$, where n is the size of each record.

Consider three ways in deleting record i:

- 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 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

Variable-Length

Encoding schemes where attributes are stored **in order**.

- Option1: Prefix each field by length

$\boxed{4}$ xxxx $\boxed{10}$ Neil Young $\boxed{8}$ Musician $\boxed{4}$ xxxx

- Option 2: Terminate fields by delimiter

33357462/Neil Young/Musician/0277/

- Option 3: Array of offsets

					33357462	**Neil Young**	**Musician**	**0277**

Variable-Length Records (1)

Another encoding scheme: attributes are not stored in order.

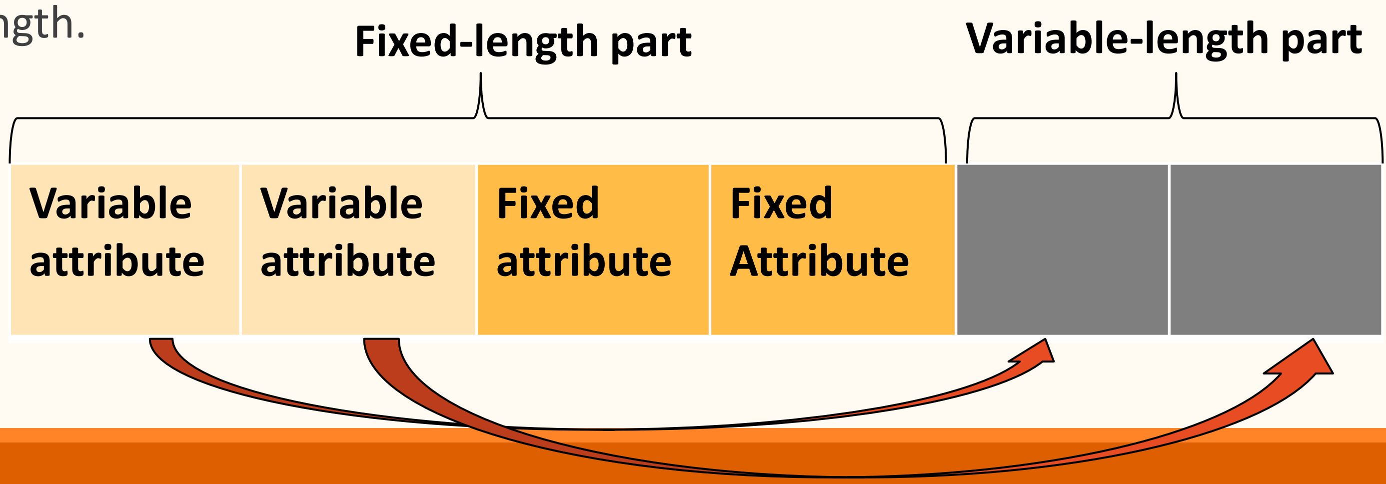
Fixed-length part followed by variable-length part.

- (b) The fixed-length part is to tell where we can find the data if it is a variable-length data field.
- (c) The variable-length part is to store the data.

Variable length attributes are represented by fixed size (offset, length) in the fixed-length part, and keep attribute values in the variable-length part.

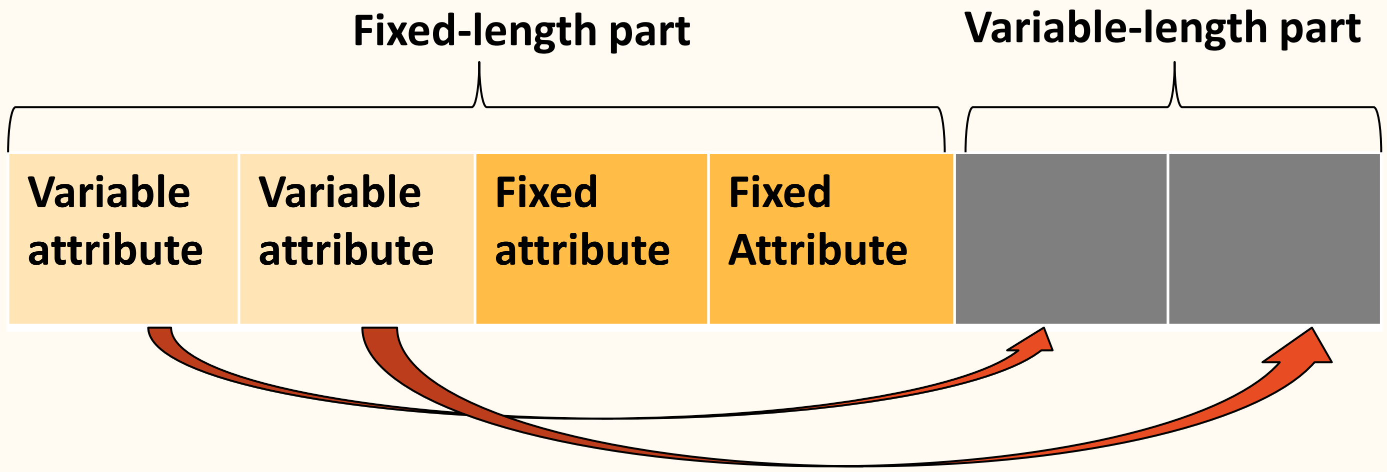
Fixed length attributes store attribute values in the fixed-length part.

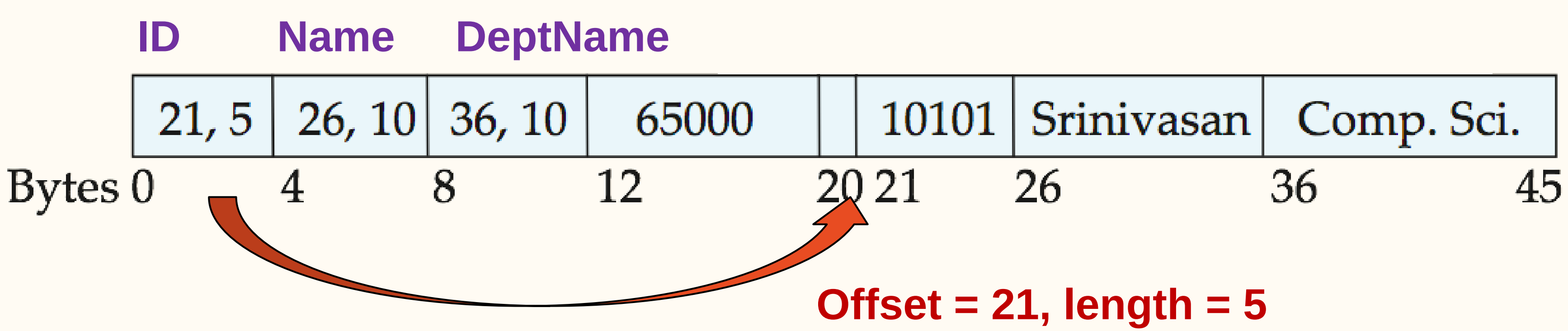
Suppose there is a relation with 4 attributes: 2 fixed-length and 2 variable-length.



Variable-Length Records (2)

Example: a tuple of (**ID, Name, DeptName**, Salary) where the **first three** are variable length.





Managing a Relation

From a relation to disk sectors

- A relation will be stored in a file in the database management system.
- A file will be managed by Operating System (OS) as a sequence of disk-pages.
- A disk-page will be stored on disk as sectors.

From a relation to fields (attribute value or column value)

- A relation is a sequence of records.
- A record is a sequence of data fields.
  - There are fixed-length data fields (say using char(n))
  - There are variable-length data fields (say using varchar(n))
- If all data fields are fixed-length, we use fixed-length records.
- If at least one data field is variable-length, we use variable-length records.

Notes

Reminder:

- The basic store unit on disk (in memory) is block (page)
- We will use page/block interchangeably.
- One page consists of multiple data records.

Key Learning Outcomes

- Buffer replacement polies: how does each policy work

- Record / Page management

Next Week: Advanced Topic + Revision

Acknowledgement

Some examples in the slides are modified from Database System Concepts, 6th Ed. Silberschatz, Korth and Sydarshan.