Database Systems
Lecture11 – Chapter 12: Physical Storage Systems

Beomseok Nam (남범석)
bnam@skku.edu

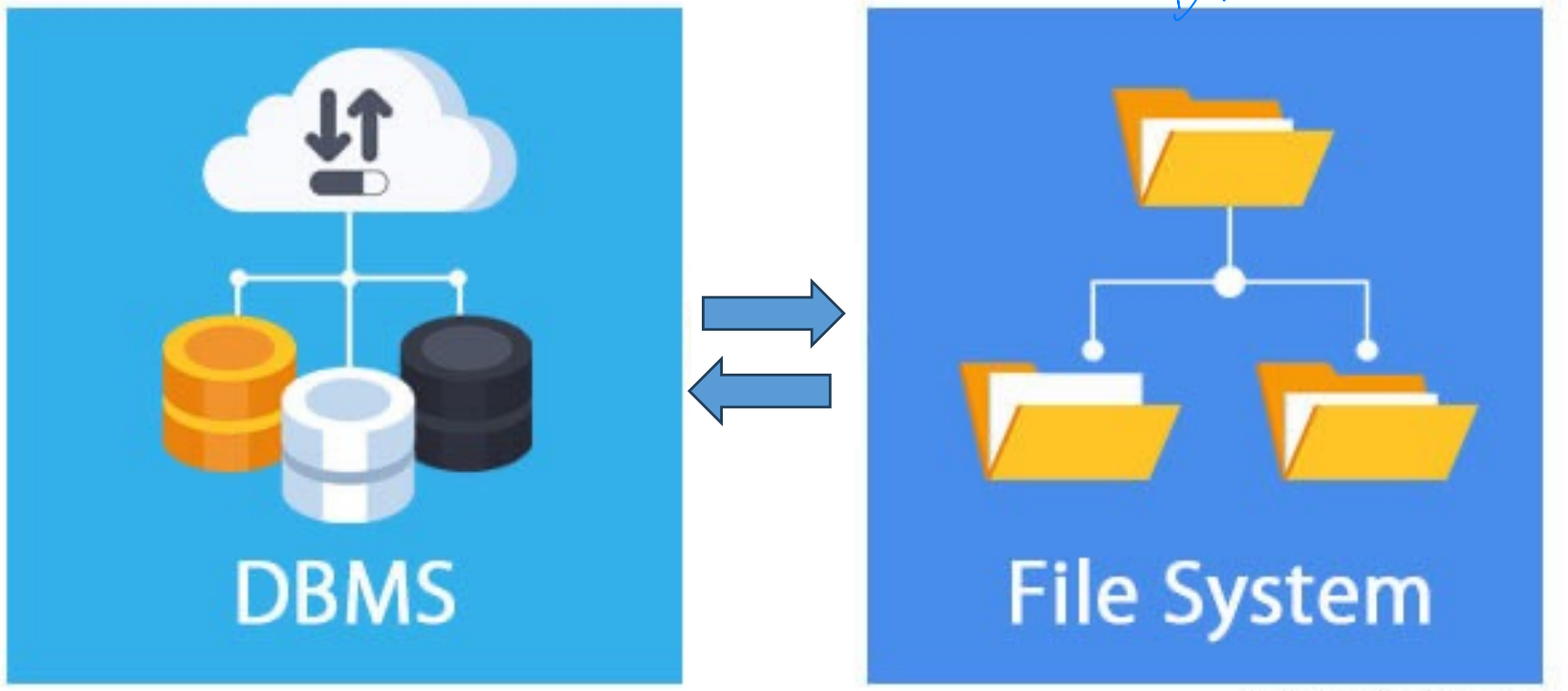
Table of Contents We Skip For Now

- Chap 8: Complex Data Types
  - XML, JSON, etc
- Chap 9: Application Development
  - HTML, Servlet, JavaScript, etc
- Chap 10: Big Data
  - Hadoop
- Chap 11: Data Analytics
  - Data Warehousing, OLAP, Data Mining

Part III: Storage Management

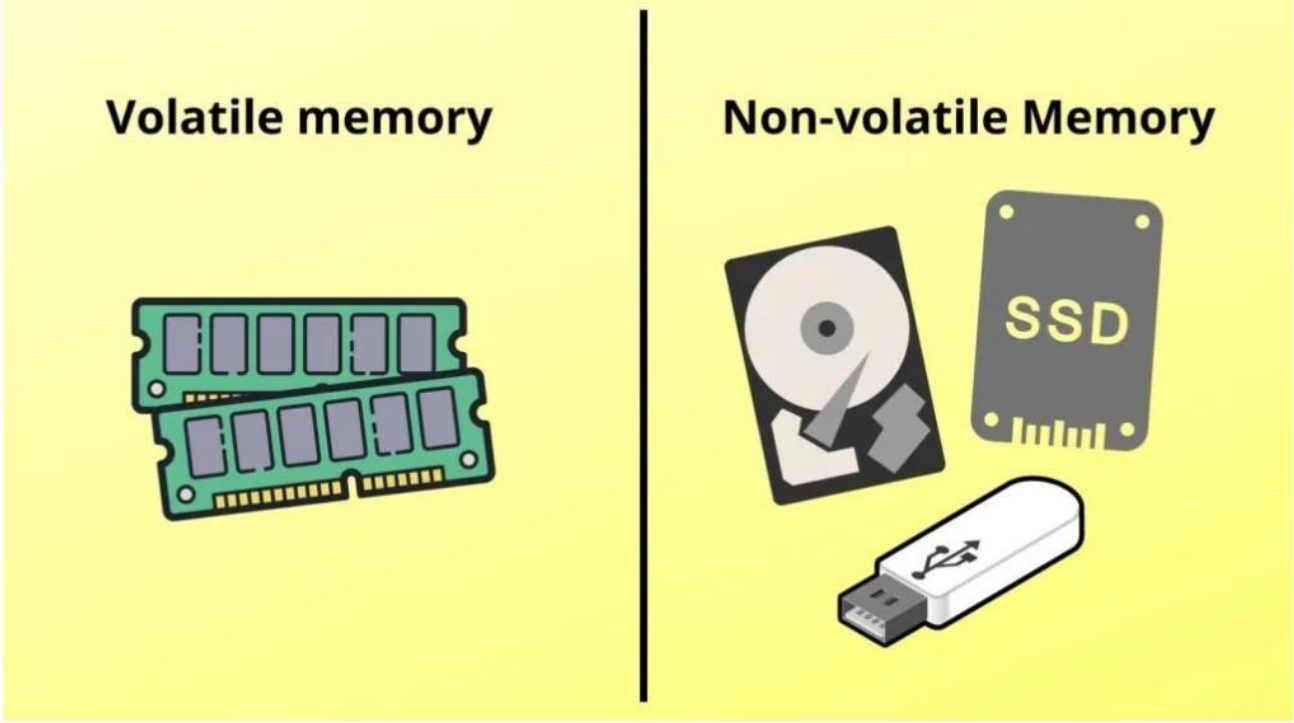
- DBMS is a system S/W designed to manage databases

- DBMS interacts with file systems for **durability**



Classification of Storage Media

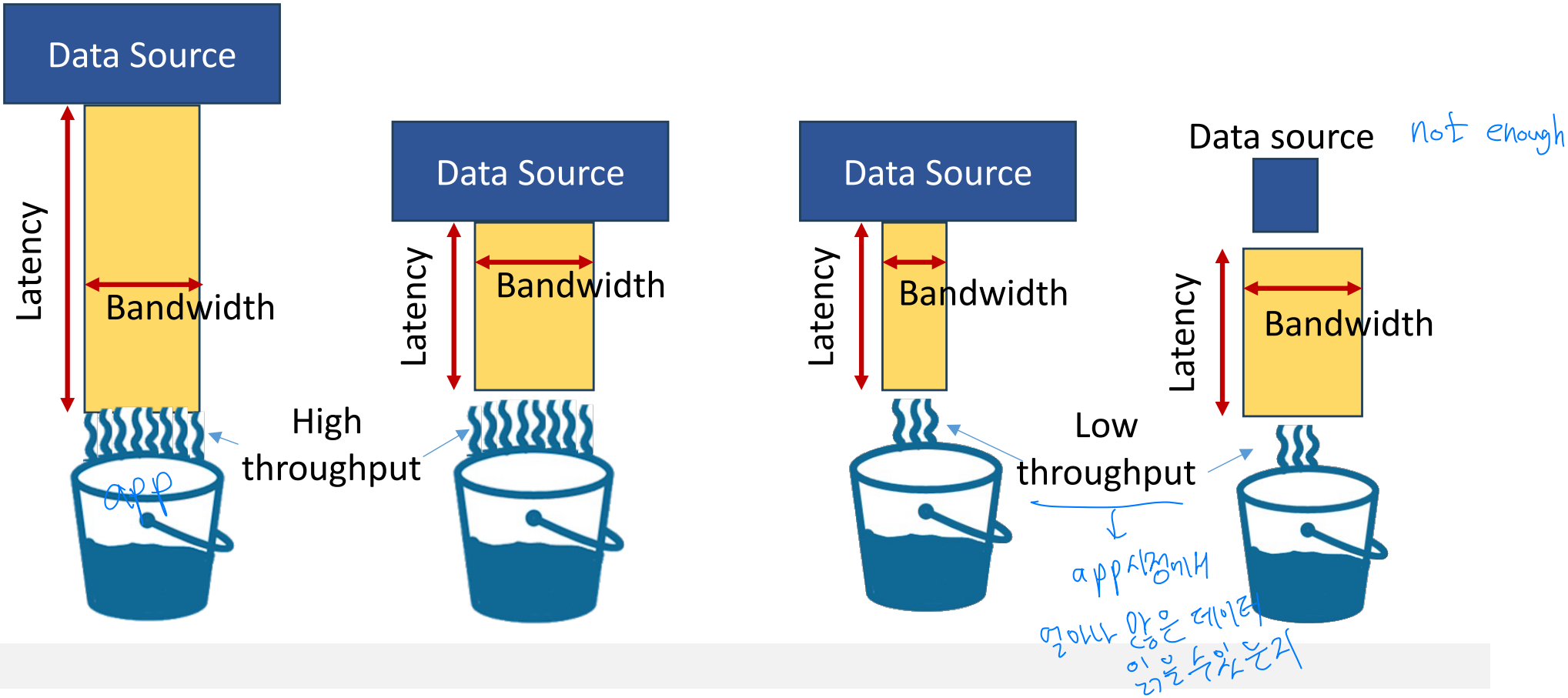
- Can differentiate storage into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage**:
    - Contents persist even when power is switched off.



Performance of Storage Media

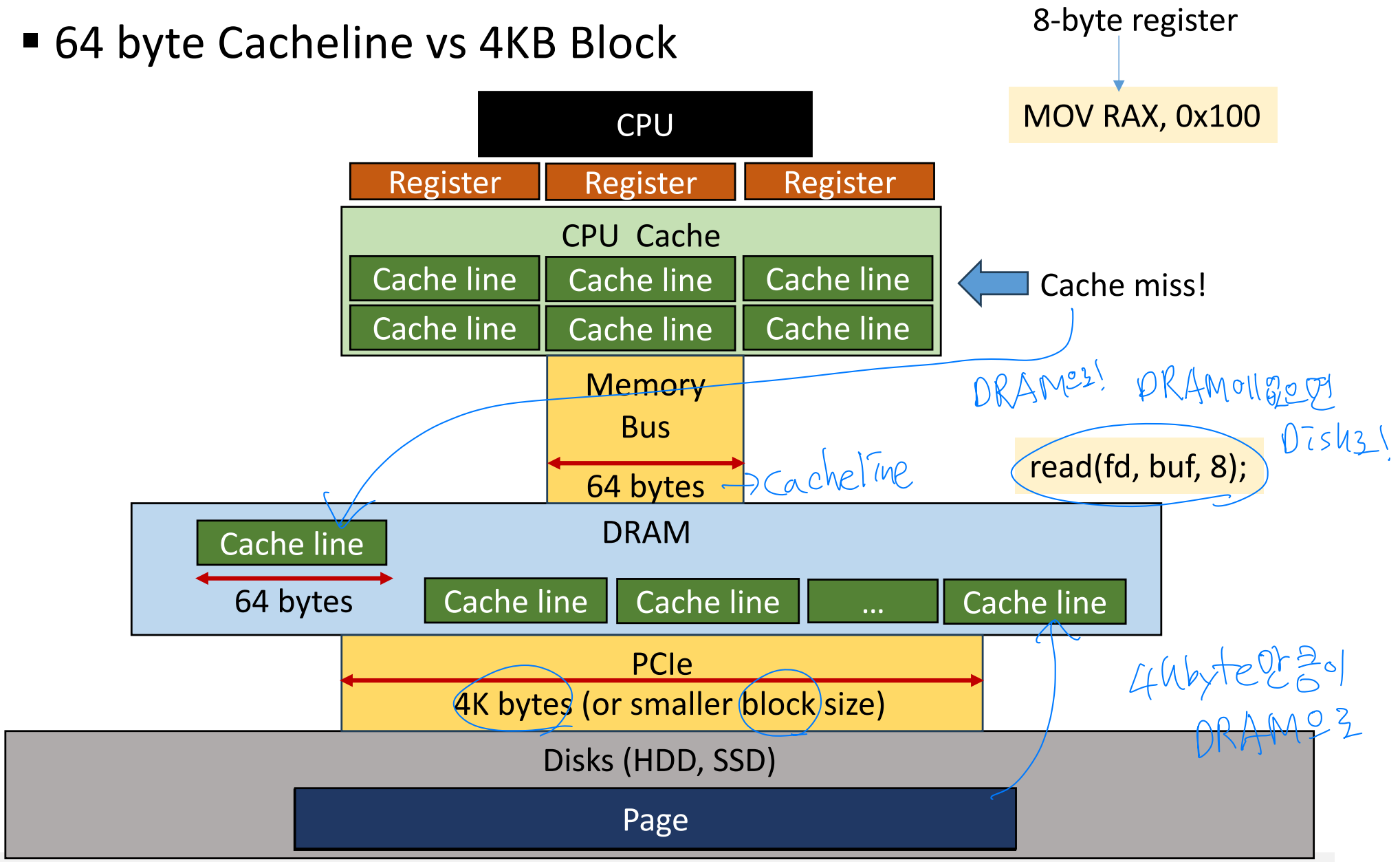
- Factors affecting choice of storage media
  - Latency → 거리 (멀다, 시간적개념)
  - Bandwidth → 얼마나 많은 데이터전송가능한지? (device 시점)
  - Cost

Bandwidth metric: I/O operations per second (IOPS)



IO Granularity of Storage Media

- 64 byte Cacheline vs 4KB Block



Before HDD, there was Magnetic Tapes

Tape Drive for Sequential Access

Tape Head

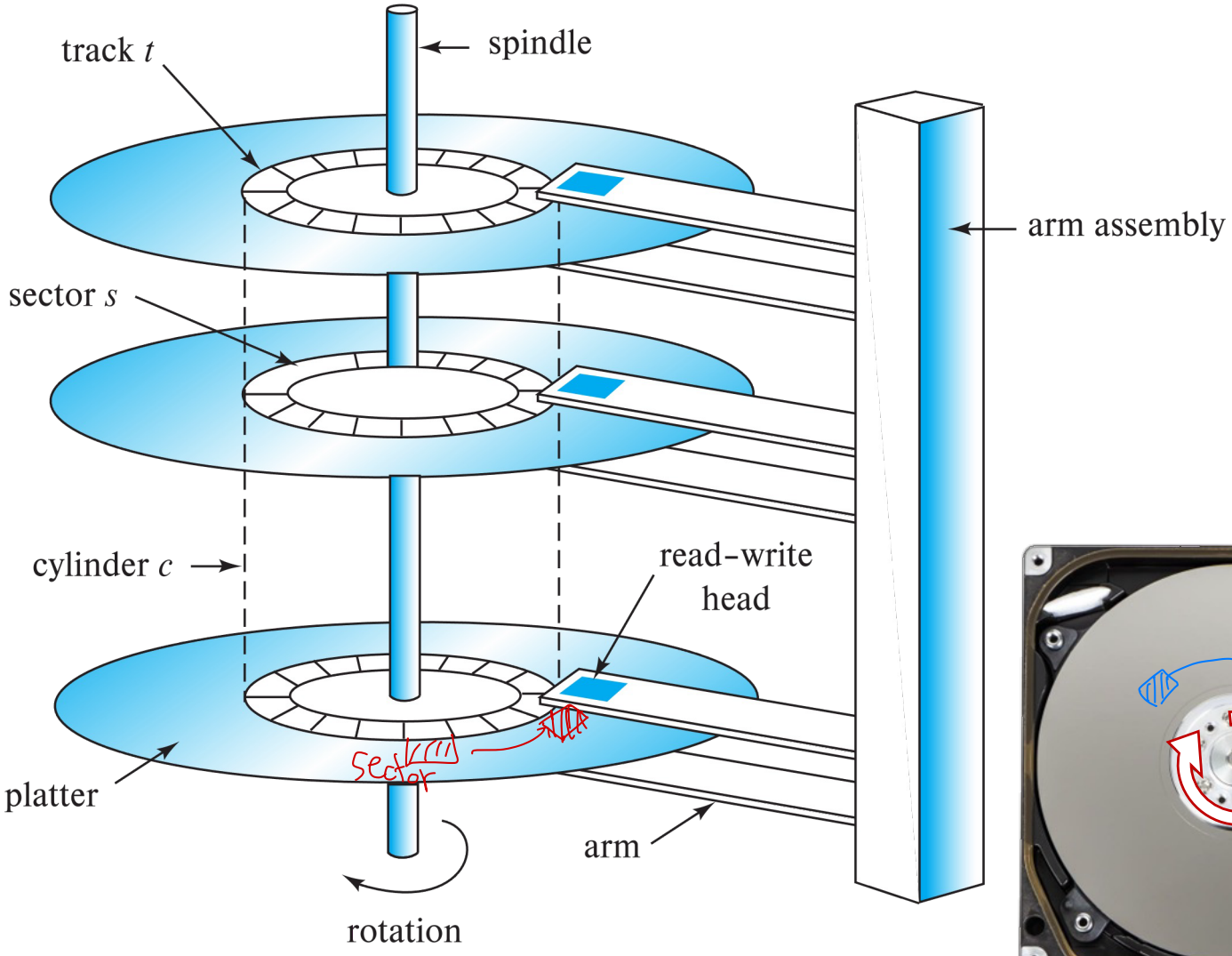
develop

Floppy Disks for Random Access

- 1-Dimensional movement
  - read through the tape to find the desired data
  - Tape head does not move
- Still popular as backup devices in large data centers

- 2-Dimensional movement
  - Circular magnetic disk spins
  - Tape head moves

Magnetic Hard Disk Mechanism



Schematic diagram of magnetic disk drive

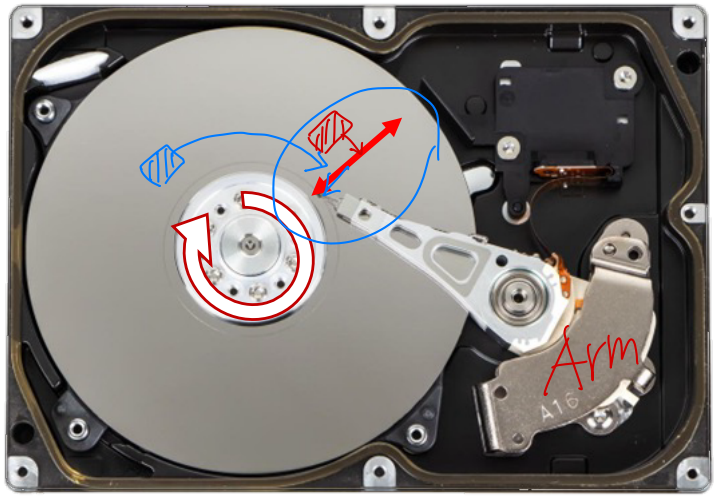


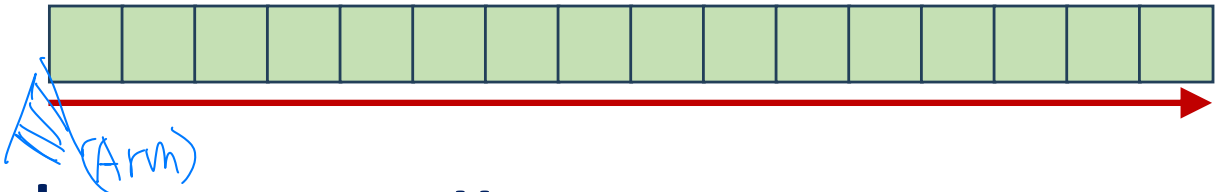
Photo of magnetic disk drive

Performance Measures of Disks

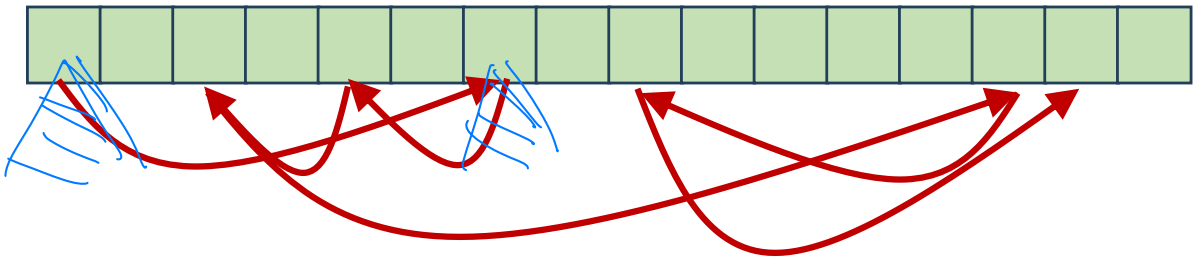
- **Access time** – the time it takes from when an I/O request is issued to when data transfer begins. Consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
  - Overall latency is 5 to 20 msec depending on disk model
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 200 MB per second max rate, lower for inner tracks

Performance Measures (Cont.)

- **Disk block** is a logical IO unit for storage
  - 4 to 16 kilobytes typically
    - Problem of Large block: unnecessary data transfer
- **Sequential access pattern**
  - Disk seek required only for first block



- **Random access pattern**
  - Each access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks



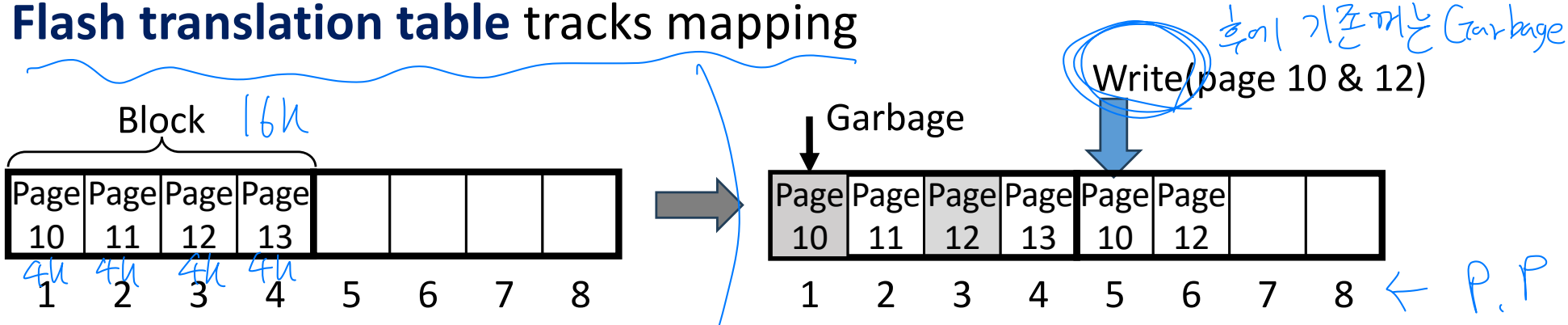
Flash Storage

- NAND flash
  - page: 512 bytes to 4 KB
    - 20 to 100 microseconds for a page read
  - Page can only be written once
    - Must be erased to allow rewrite

- **Solid state disks** (SSD)
  - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
  - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe

Flash Storage (Cont.)

- Erase happens in units of **erase block**
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- **Remapping** of logical page addresses to physical page addresses
- **Flash translation table** tracks mapping



FTL

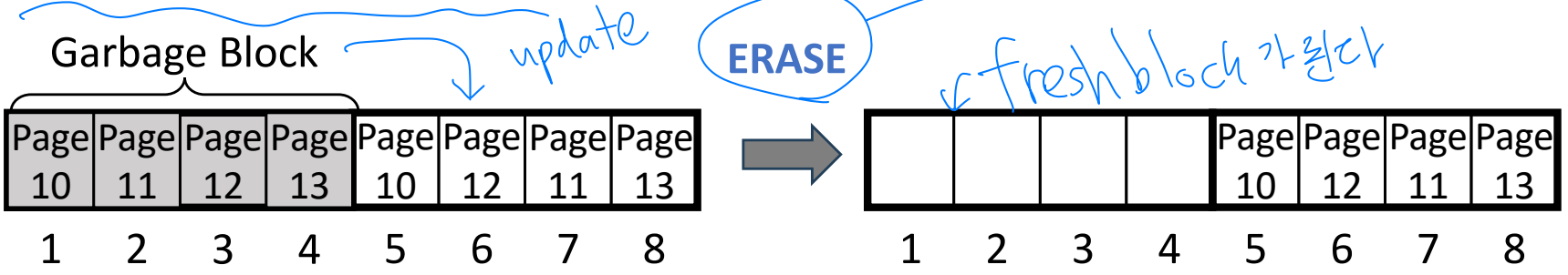
Logical Page Address	10	11	12	13
Physical Page Addr.	1	2	3	4

FTL

LPA	10	11	12	13
PPA	5	2	6	4

Flash Storage (Cont.)

- Erase happens in units of **erase block**
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- **Remapping** of logical page addresses to physical page addresses
- **Flash translation table** tracks mapping



FTL

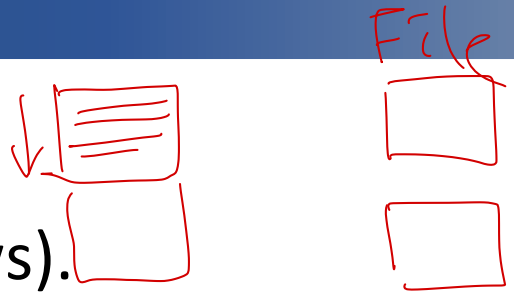
LPA	10	11	12	13
PPA	5	7	6	8

FTL

LPA	10	11	12	13
PPA	5	7	6	8

- After 100,000 to 1,000,000 erases, block becomes unusable
  - **wear leveling** becomes necessary

File Organization



- Each Table = File(s)
- Each file is a sequence of records (rows).
- A record is a sequence of fields (attributes).

record << 4KB

  - We assume that records are smaller than a disk block.
  - We will consider variable length records later

Fixed-Length Records

- Simple approach:
  - File offset for record k is n * (k – 1), where n is the size of record.

fixed size!

int name salary (int)

	4 bytes	10 bytes	10 bytes	4 bytes
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	Califeri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000

1개의 tuple → 28byte

28 x 6 = 168

168
+ 4
172

Katz의 시작주소 : 172byte

Q: What's the file offset for the name of record 6?

Fixed-Length Records

- Let's delete record *k:*
  - **Option 1) move records *k + 1*, . . ., *n* to *k*, . . . , *n* – 1**
  - Option 2)
  - Option 3)

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	Califeri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000

Fixed-Length Records

- Let's delete record *k:*
  - **Option 1) move records *k + 1*, . . ., *n* to *k*, . . . , *n* – 1**
  - Option 2)
  - Option 3)

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	Califeri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000

Fixed-Length Records

- Let's delete record *k:*
  - **Option 2) move record *n* to *k***
  - Option 3)
- E.g. Record 3 deleted and replaced by record 9

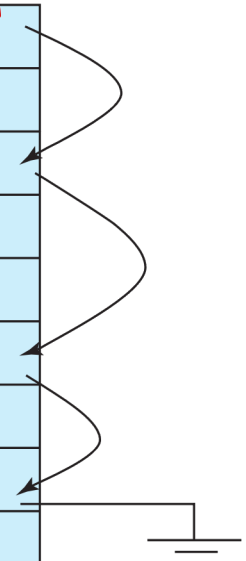
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 9	76766	Crick	Biology	72000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califeri	History	62000
record 8	76543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

Fixed-Length Records

- Let's delete record *k:*
  - **Option 3) do not move records, but link all free records on a *free list***

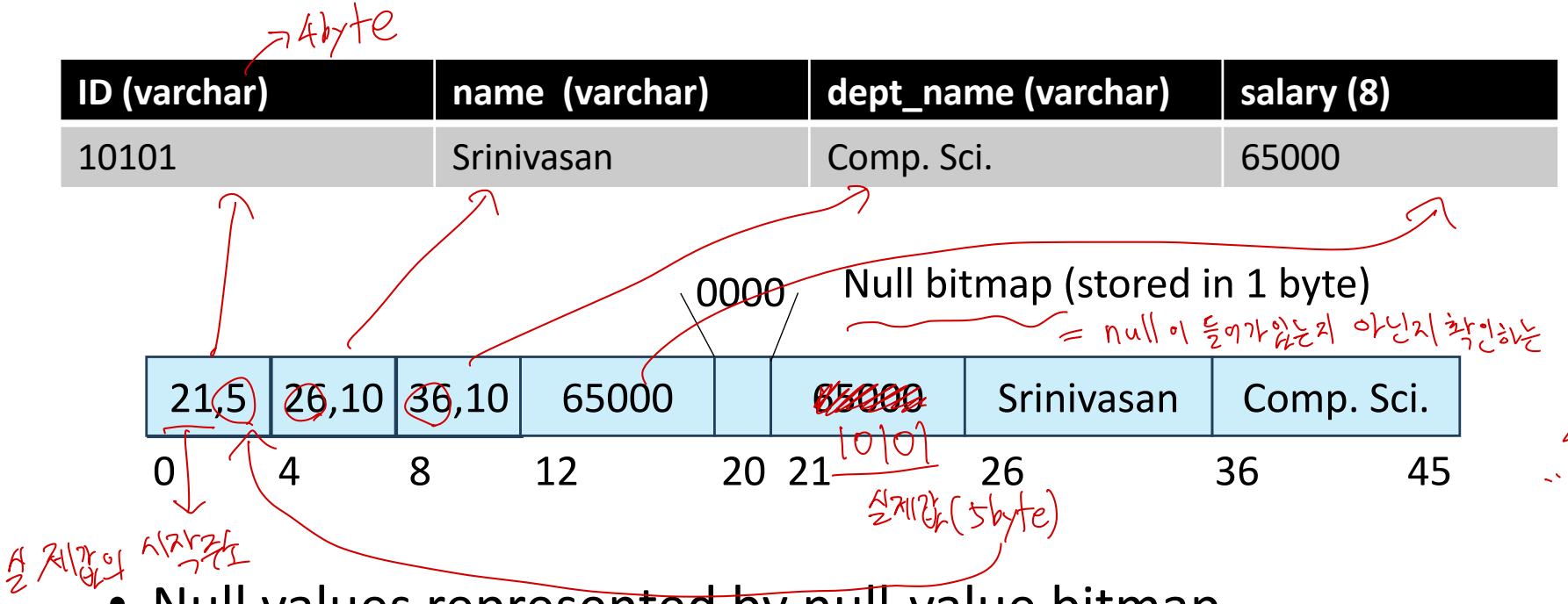
Free space를 가리키는 linked list update

header				Free
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				Free
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



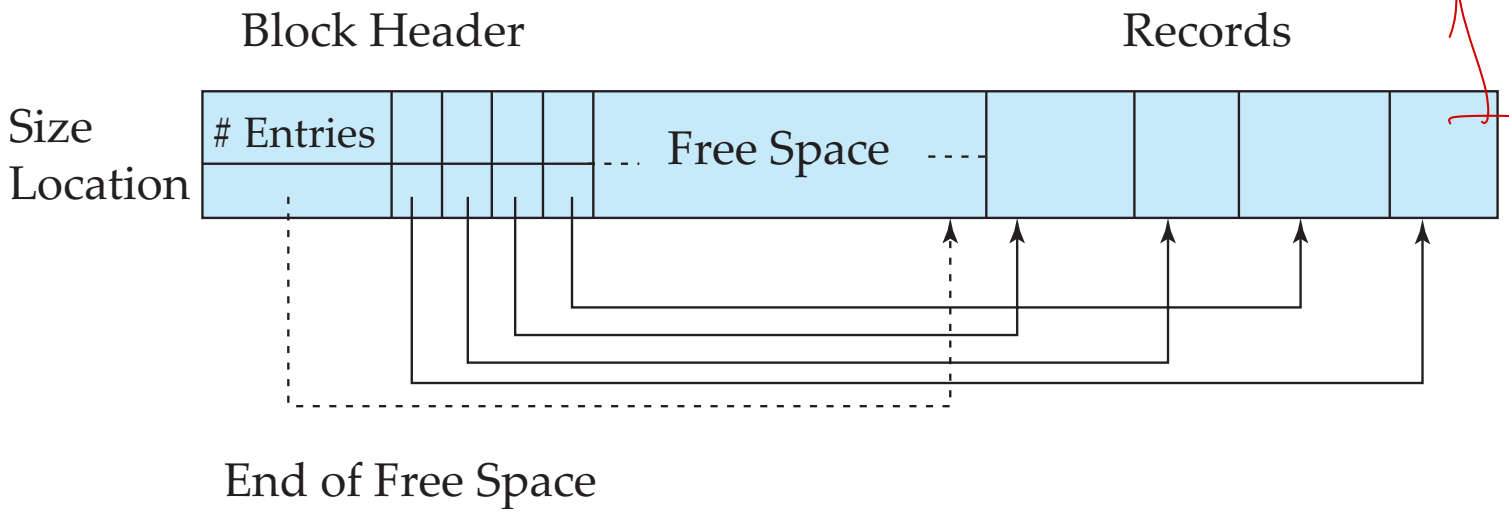
Variable-Length Records

- Variable-length record
  - **A tuple with column(s) of varchar type**
  - An array of <offset, length> pairs points to actual data stored after the fixed-length array



- Null values represented by null-value bitmap

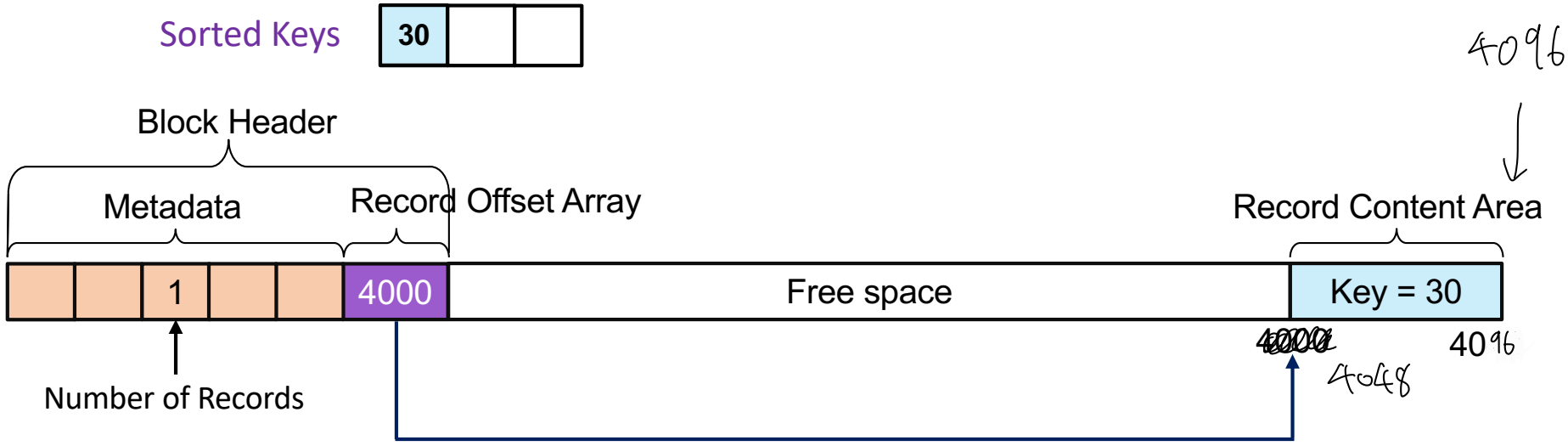
Page Layout with Variable-Length Records



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record

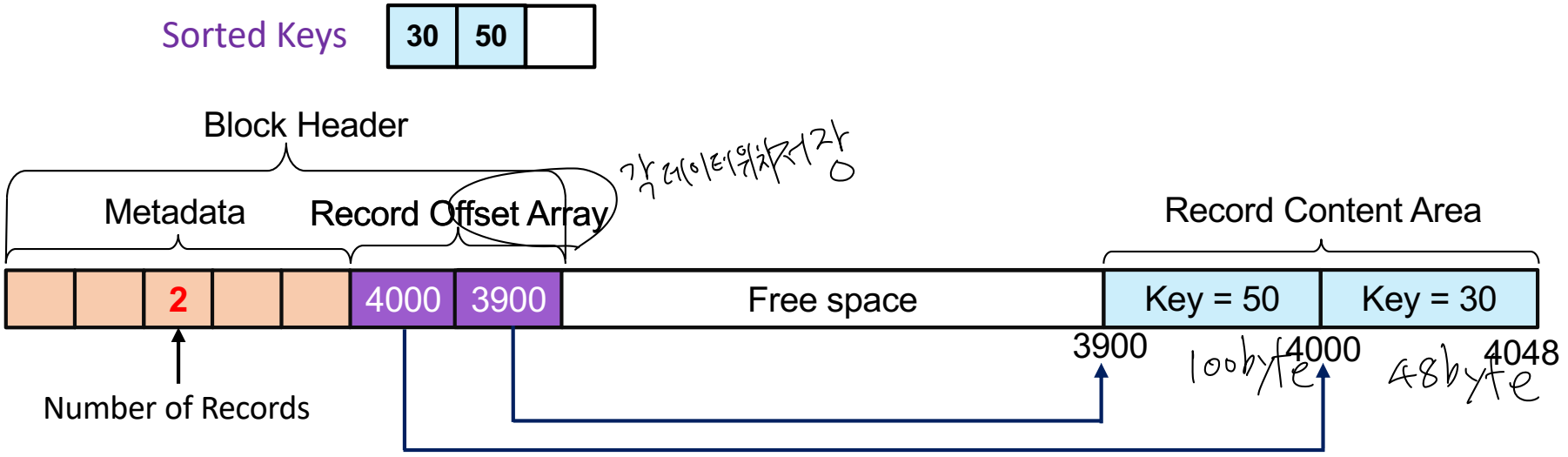
Slotted Page Structure

- **Slotted page** can sort records without moving them
  - Eg. Insert variable length records:
    primary key 30 (48 bytes) →
    primary key 50 (100 bytes) →
    primary key 40 (50 bytes)



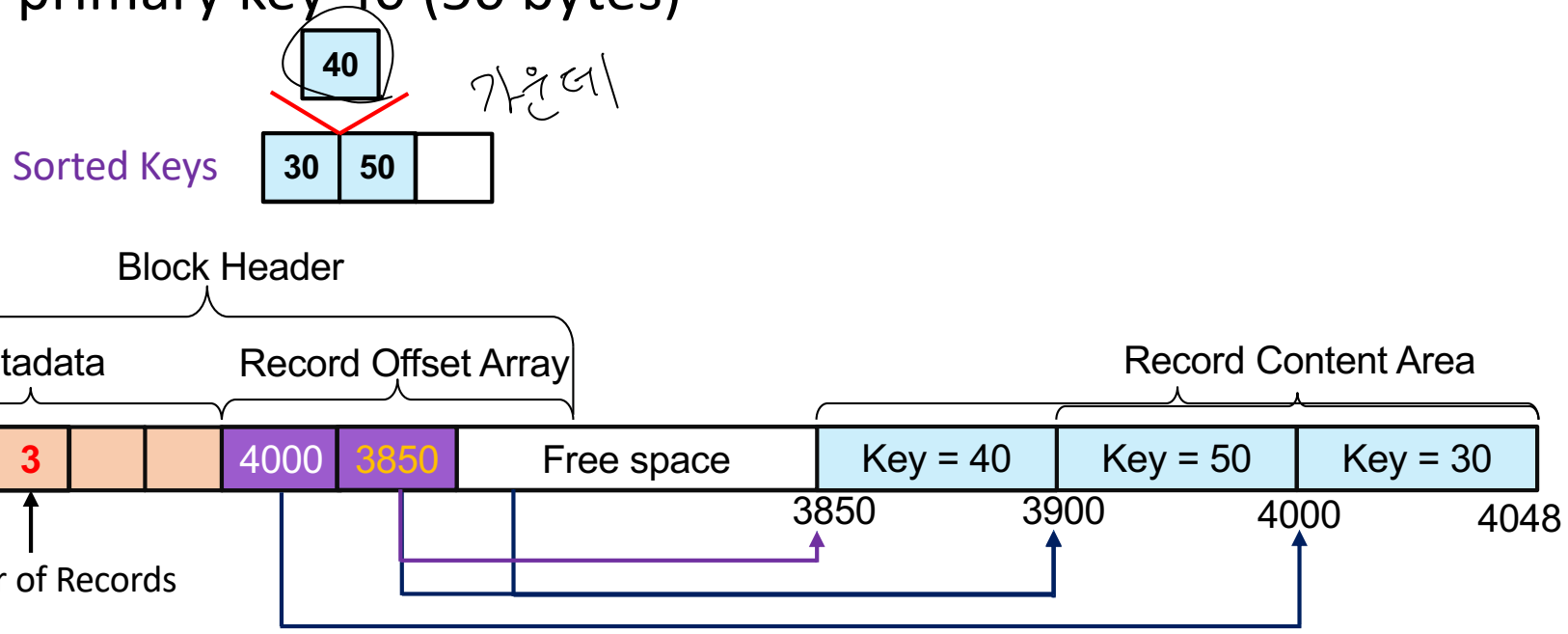
Slotted Page Structure

- **Slotted page** can sort records without moving them
  - Eg. Insert variable length records with
    primary key 30 (48 bytes) →
    primary key 50 (100 bytes) →
    primary key 40 (50 bytes)



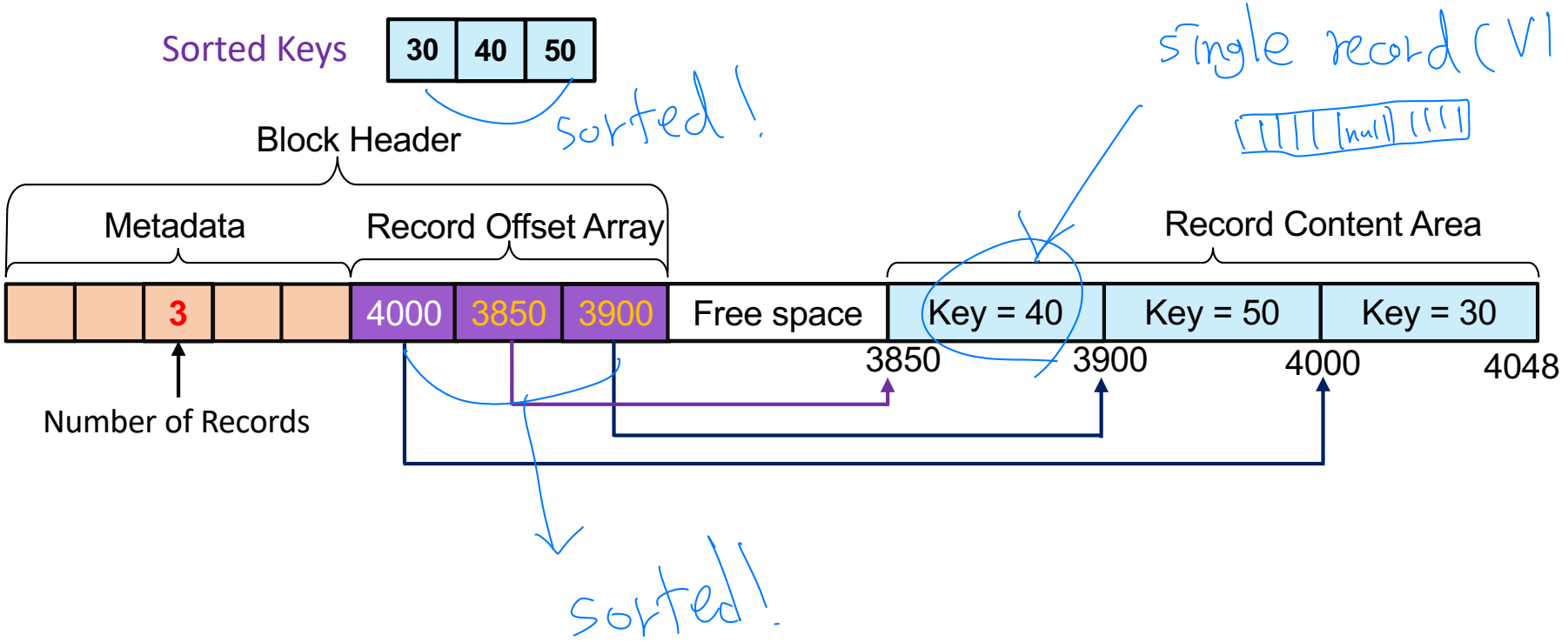
Slotted Page Structure

- **Slotted page** can sort records without moving them
  - Eg. Insert variable length records with
    primary key 30 (48 bytes) →
    primary key 50 (100 bytes) →
    primary key 40 (50 bytes)



Slotted Page Structure

- **Slotted page** can sort records without moving them
  - Eg. Insert variable length records with
    primary key 30 (48 bytes) →
    primary key 50 (100 bytes) →
    primary key 40 (50 bytes)

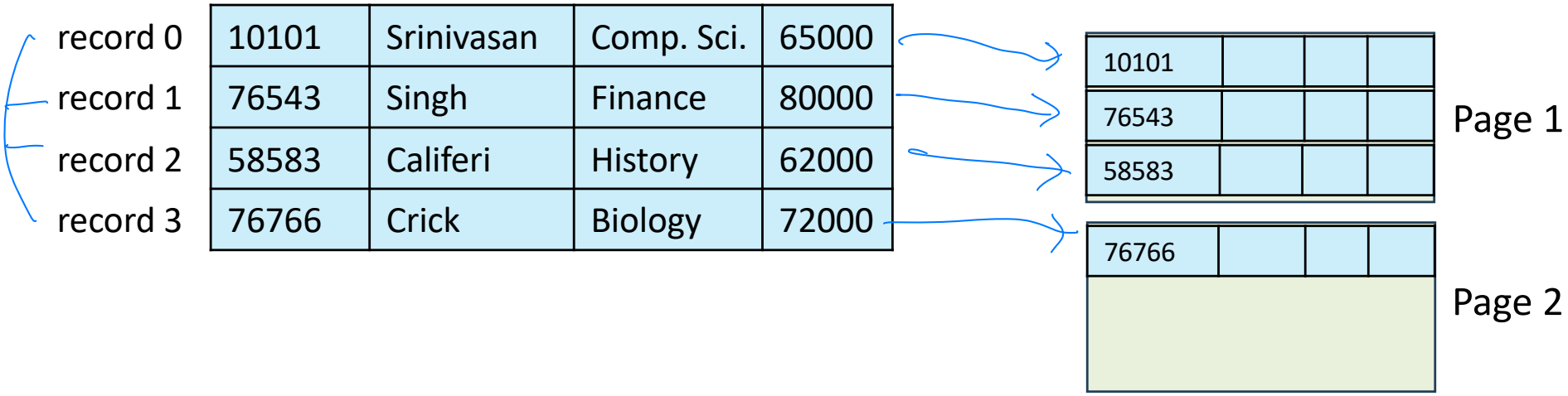


Organization of Records in Files

- **Heap** – record can be placed in any free space in the file
- **Sequential** – store records in sorted order by key
- **B$^+$-tree file organization**
  - Dynamic ordered index (i.e., sorted by key)
  - More on this in Chapter 14
- **Hashing**
  - Unordered but fast index
  - More on this in Chapter 14

Heap File

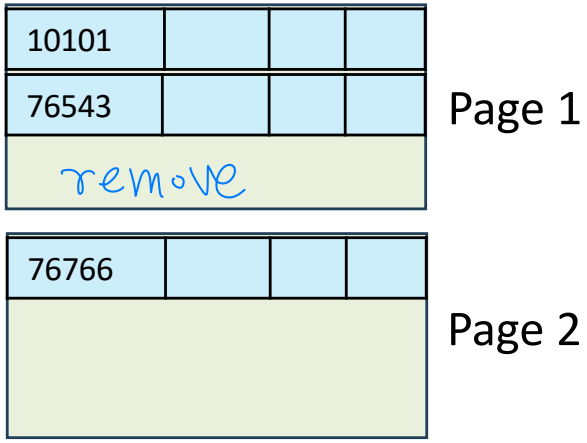
- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated



Heap File

- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated

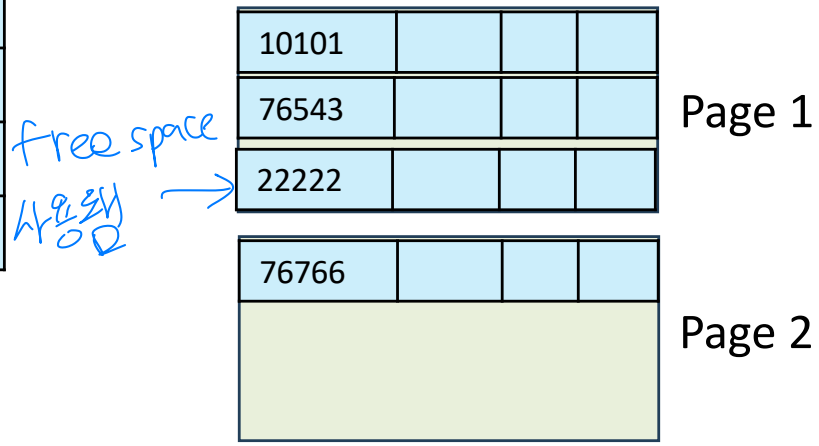
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	76543	Singh	Finance	80000
record 3	76766	Crick	Biology	72000



Heap File

- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated

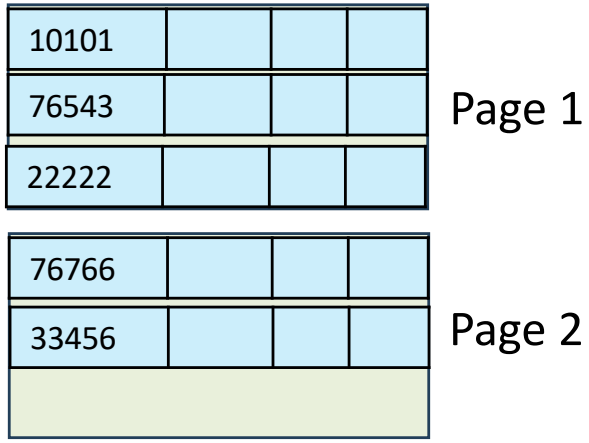
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	76543	Singh	Finance	80000
record 3	76766	Crick	Biology	72000
record 4	22222	Einstein	Physics	95000



Heap File

- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated

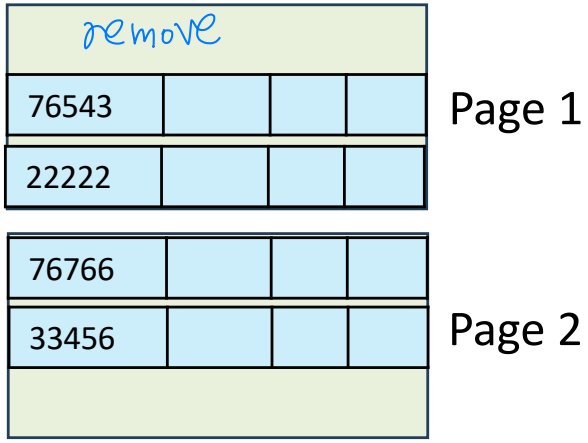
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	76543	Singh	Finance	80000
record 3	76766	Crick	Biology	72000
record 4	22222	Einstein	Physics	95000
record 5	33456	Gold	Physics	87000



Heap File

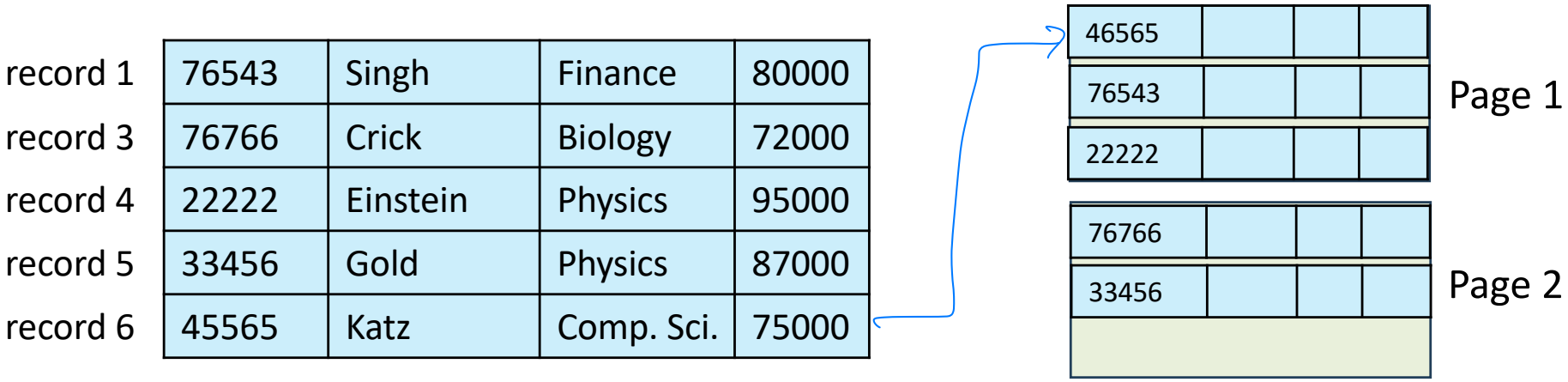
- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated

record 1	76543	Singh	Finance	80000
record 3	76766	Crick	Biology	72000
record 4	22222	Einstein	Physics	95000
record 5	33456	Gold	Physics	87000



Heap File

- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated

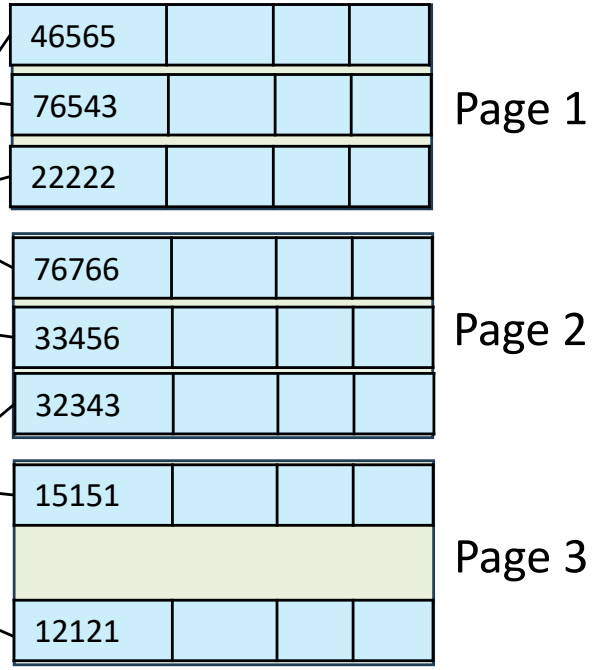


Heap File

- Similar to Record Content Area of Slotted Page Structure
  - Records can be placed anywhere in the file
  - Records do not move once allocated

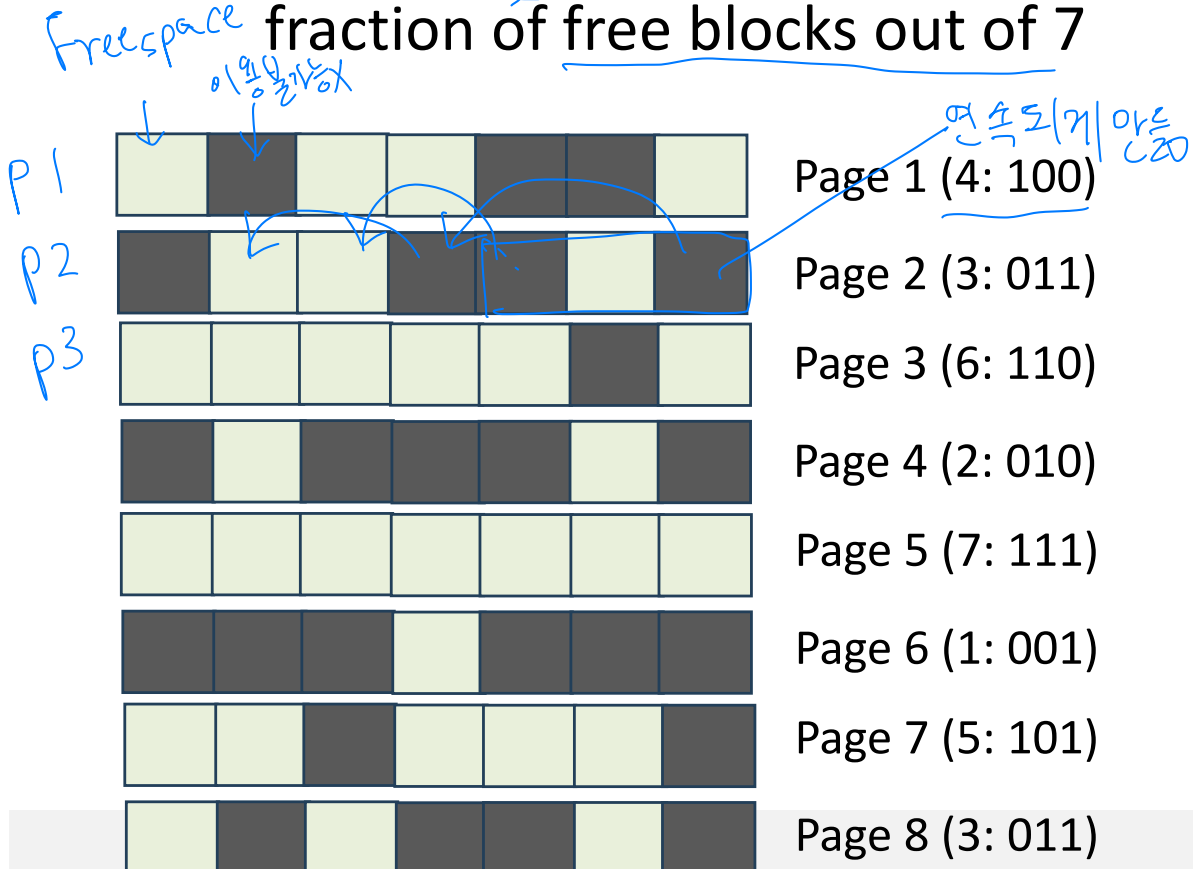
Heapfile에서 table을 읽을 때 Random position이 된다 (순서가 없다)

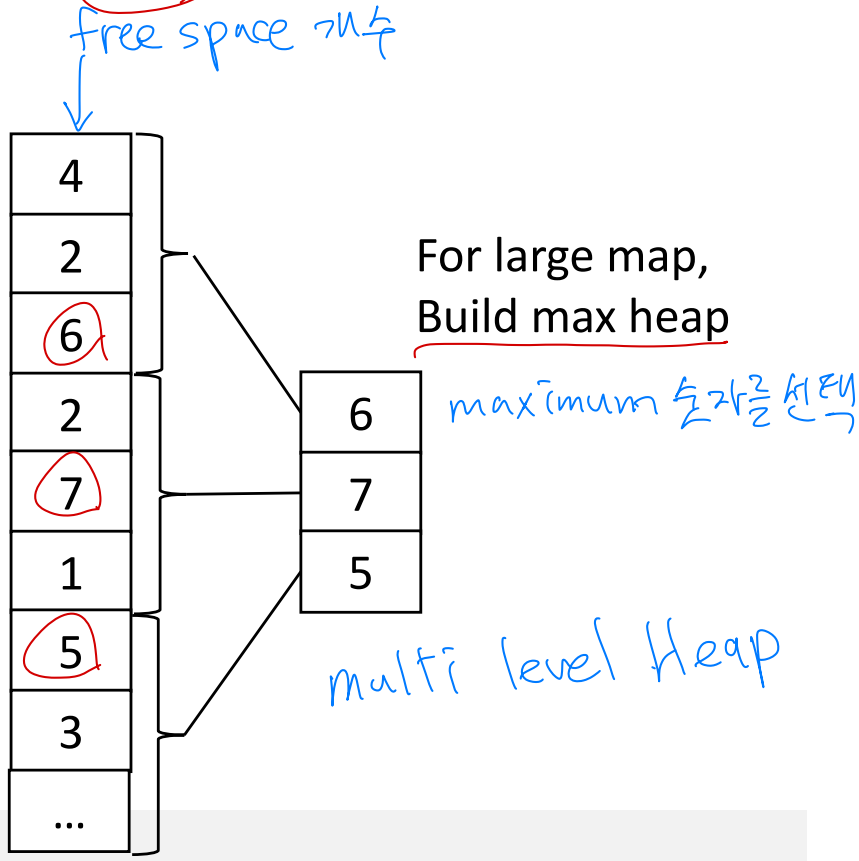
record 1	76543	Singh	Finance	80000
record 3	76766	Crick	Biology	72000
record 4	22222	Einstein	Physics	95000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 10	15151	Mozart	Music	40000
record 12	12121	Wu	Finance	90000
record 15	32343	El Said	History	60000



Heap File

- Q: How to find free space in a heap file?
- A: Free-space map
  - Each entry indicates fraction of free blocks
  - Each entry size is a few bits ~ a byte
  - E.g., 3 bits per block, value divided by 2^3=8 indicates fraction of free blocks out of 7

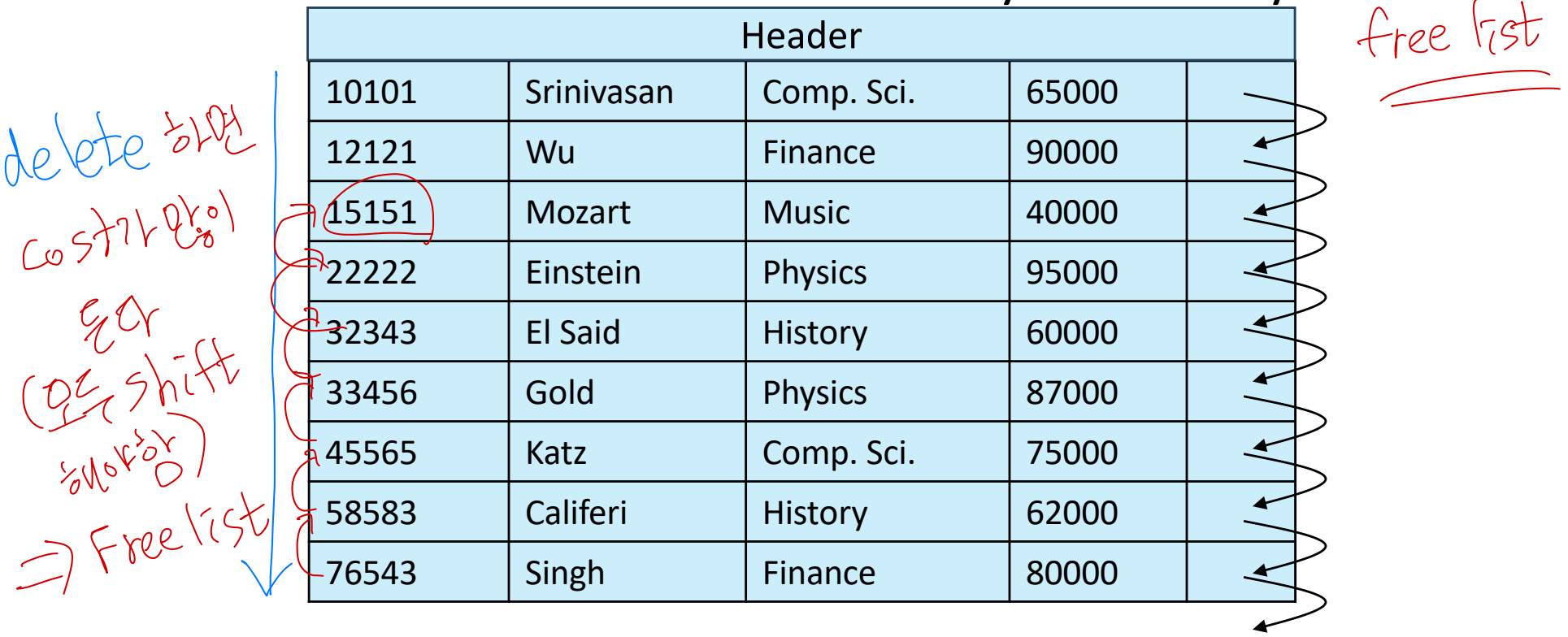




Sequential File Organization

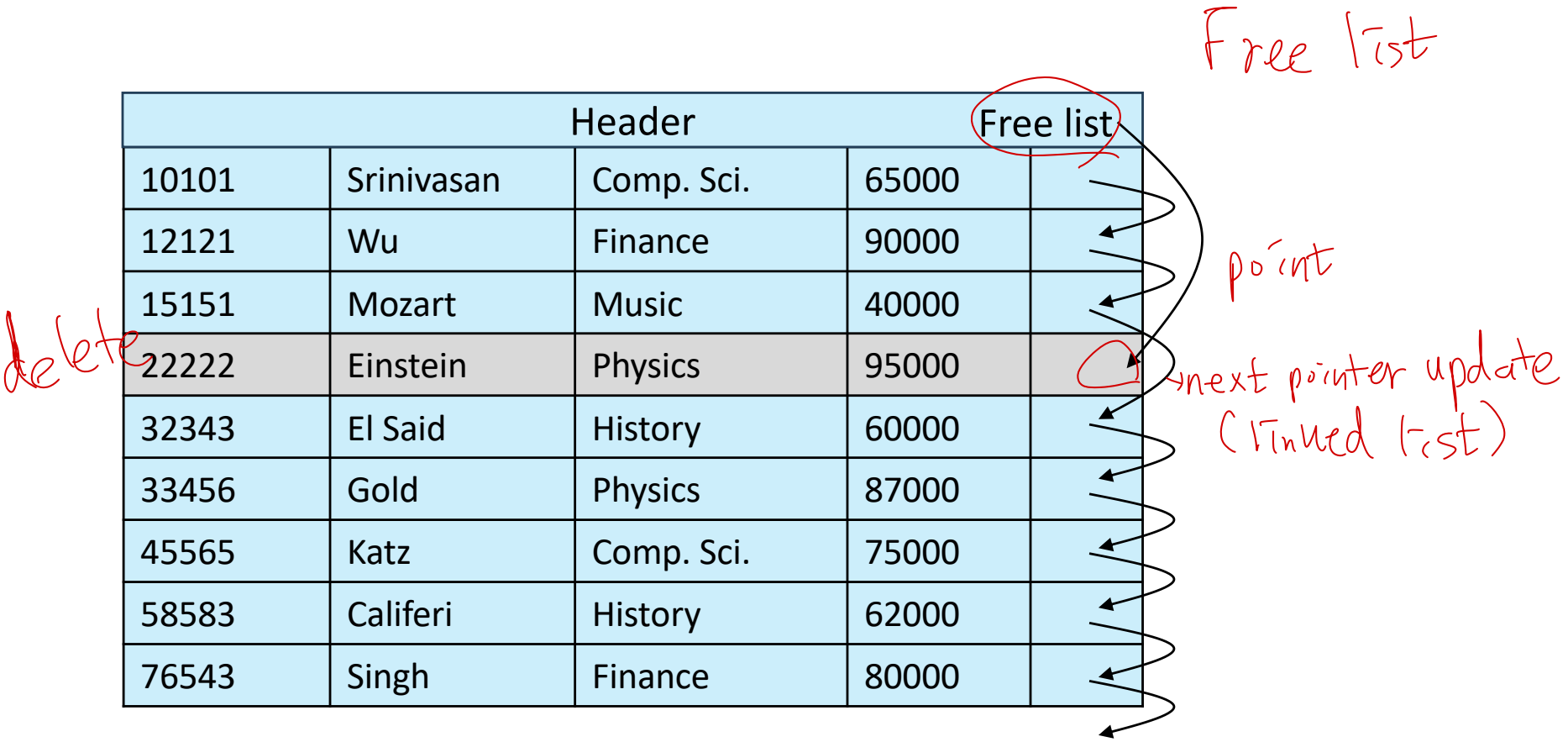
Tape Drive (순차적)

- Suitable for applications that perform sequential writes and sequential reads of the entire file
  - E.g., LSM tree-based key-value stores (RocksDB)
- The records in the file are ordered by search-key



Sequential File Organization (Cont.)

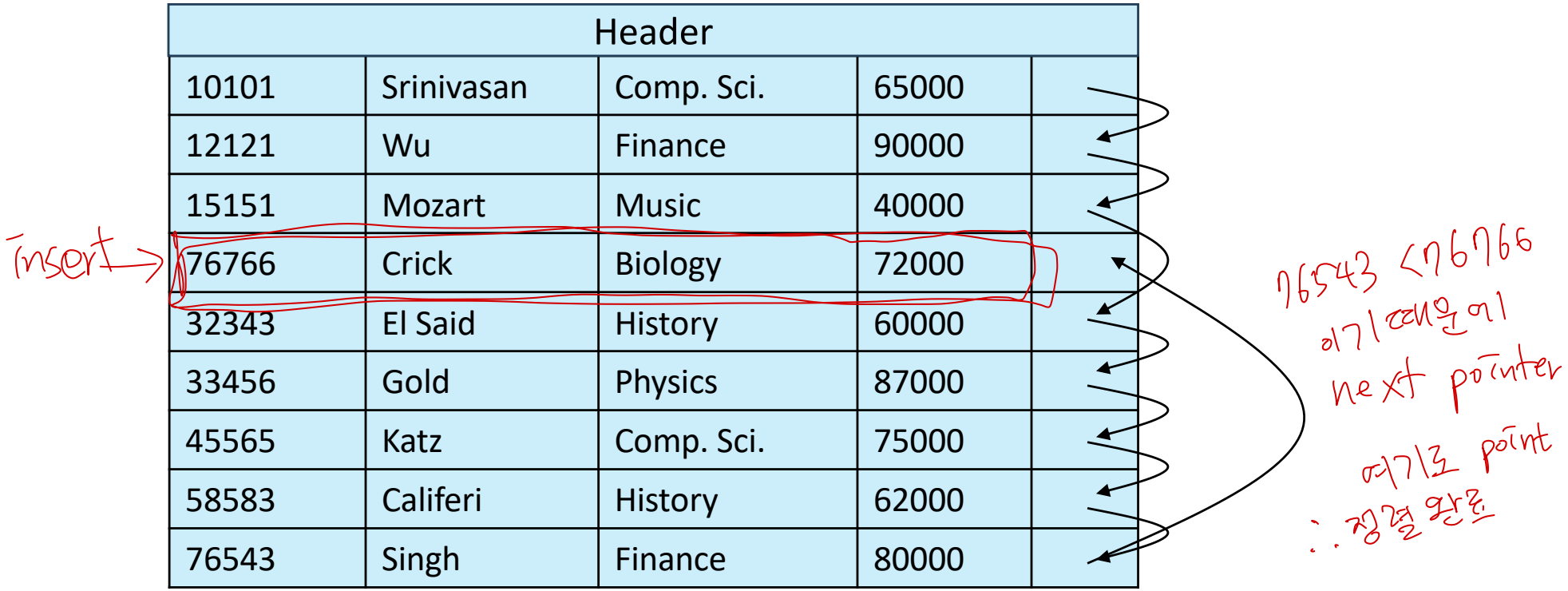
- Deletion – use pointer chains



Header				Free list
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	Califeri	History	62000	
76543	Singh	Finance	80000	

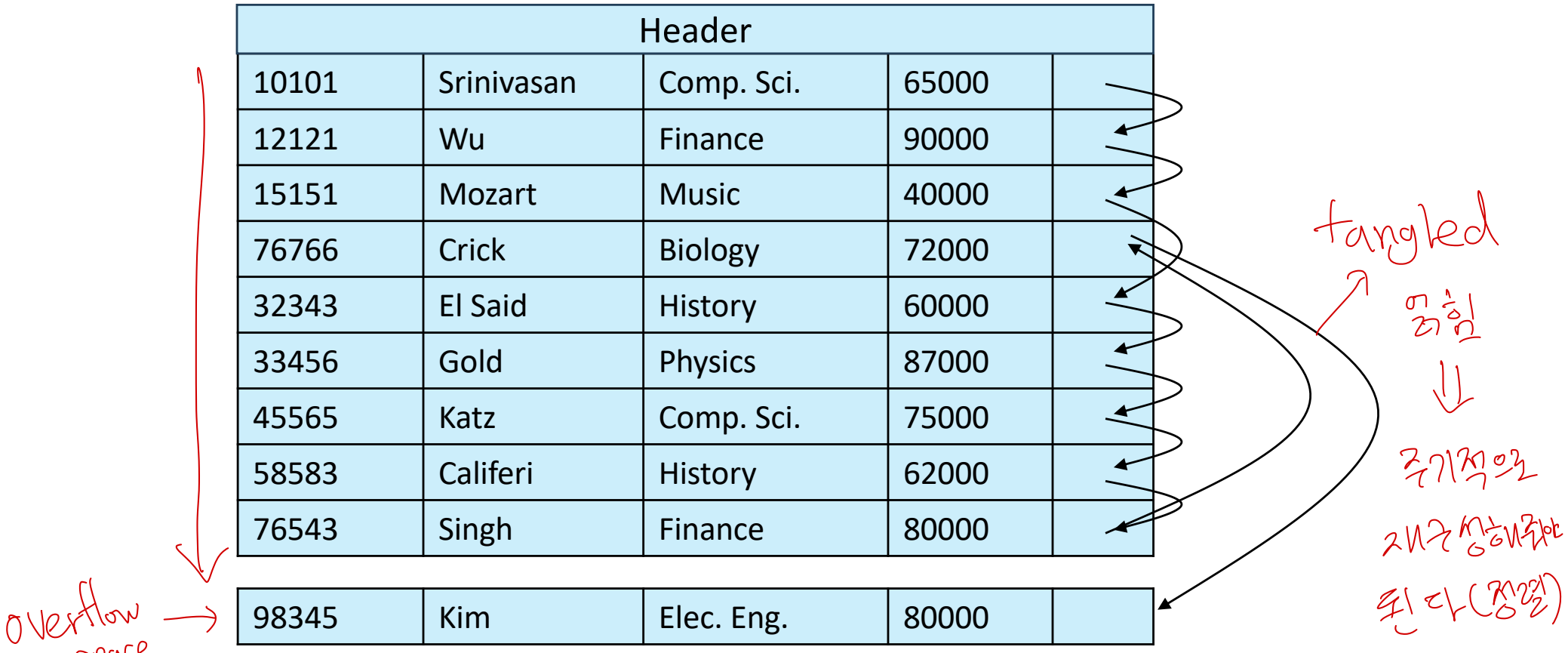
Sequential File Organization (Cont.)

- Insertion
  - if there is free space, insert there



Sequential File Organization (Cont.)

- Insertion
  - if no free space, insert the record in an overflow block



- Need to completely reorganize the file from time to time to restore sequential order

Multitable Clustering File

- Store several tables in one file
  - good for queries involving ⋈
  - bad for queries involving only table

department

⋈

instructor

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

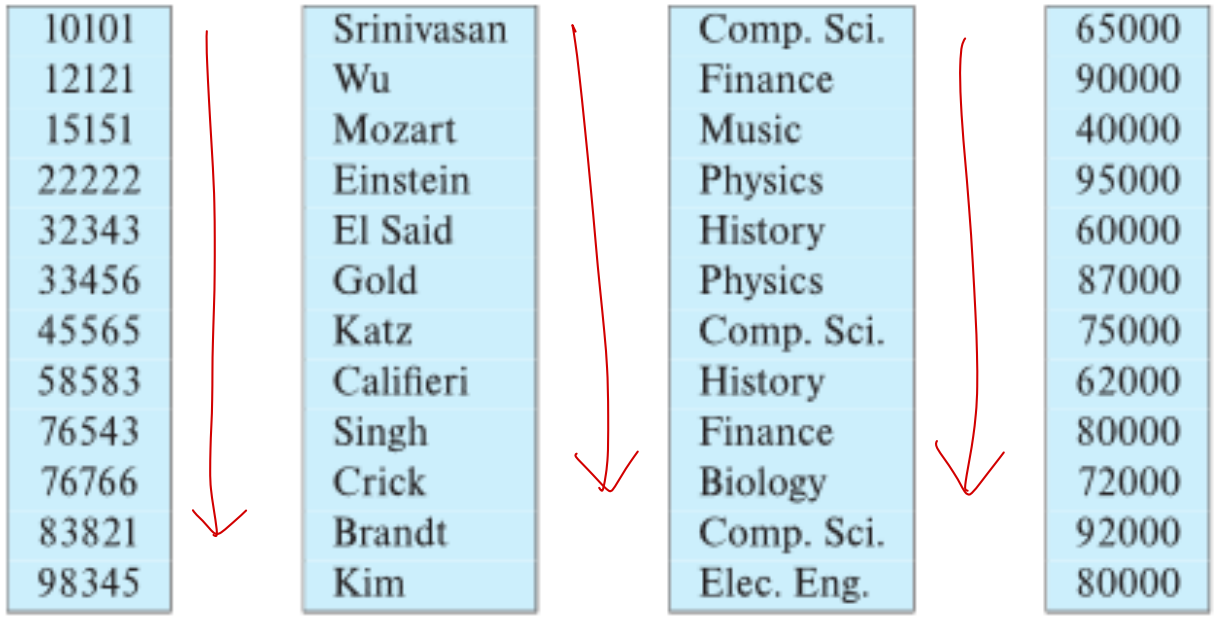
ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

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

Column-Oriented Storage

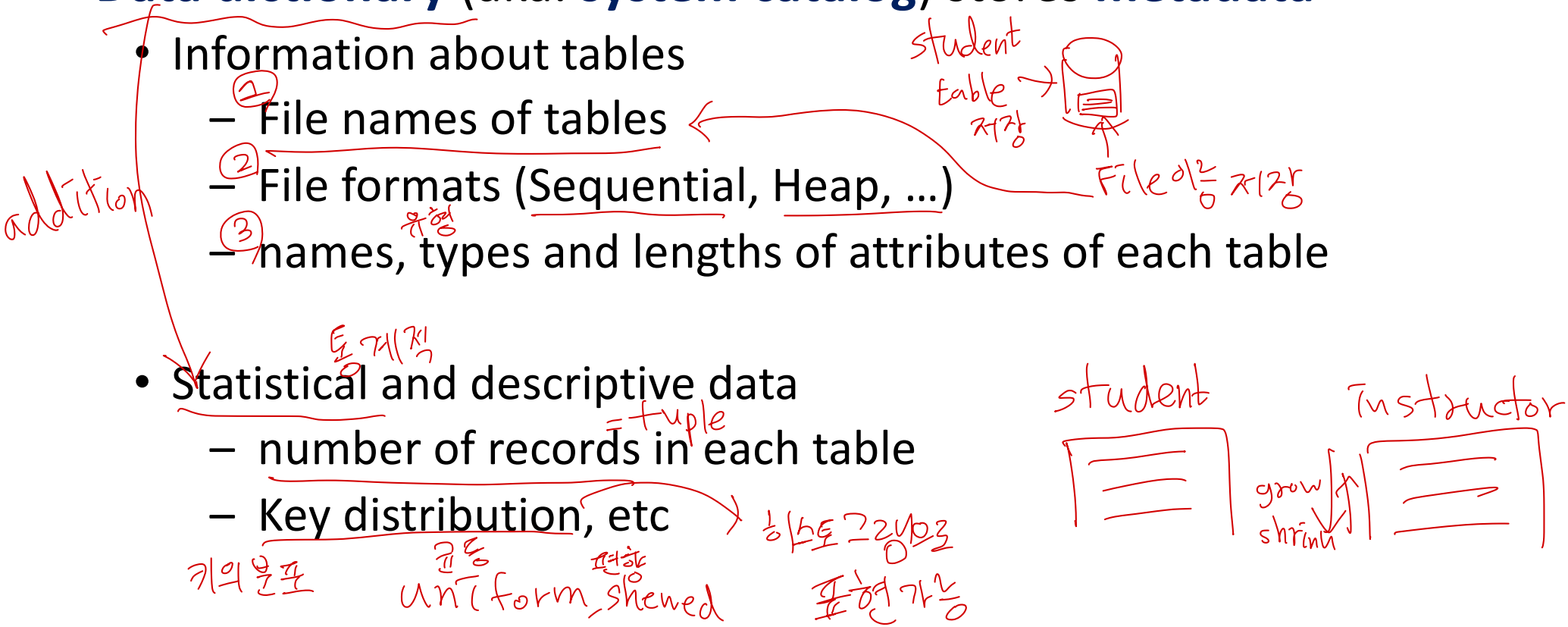
- Also known as **columnar representation**
- Store each attribute of a relation separately
  - Commonly used in NoSQL
- Example



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

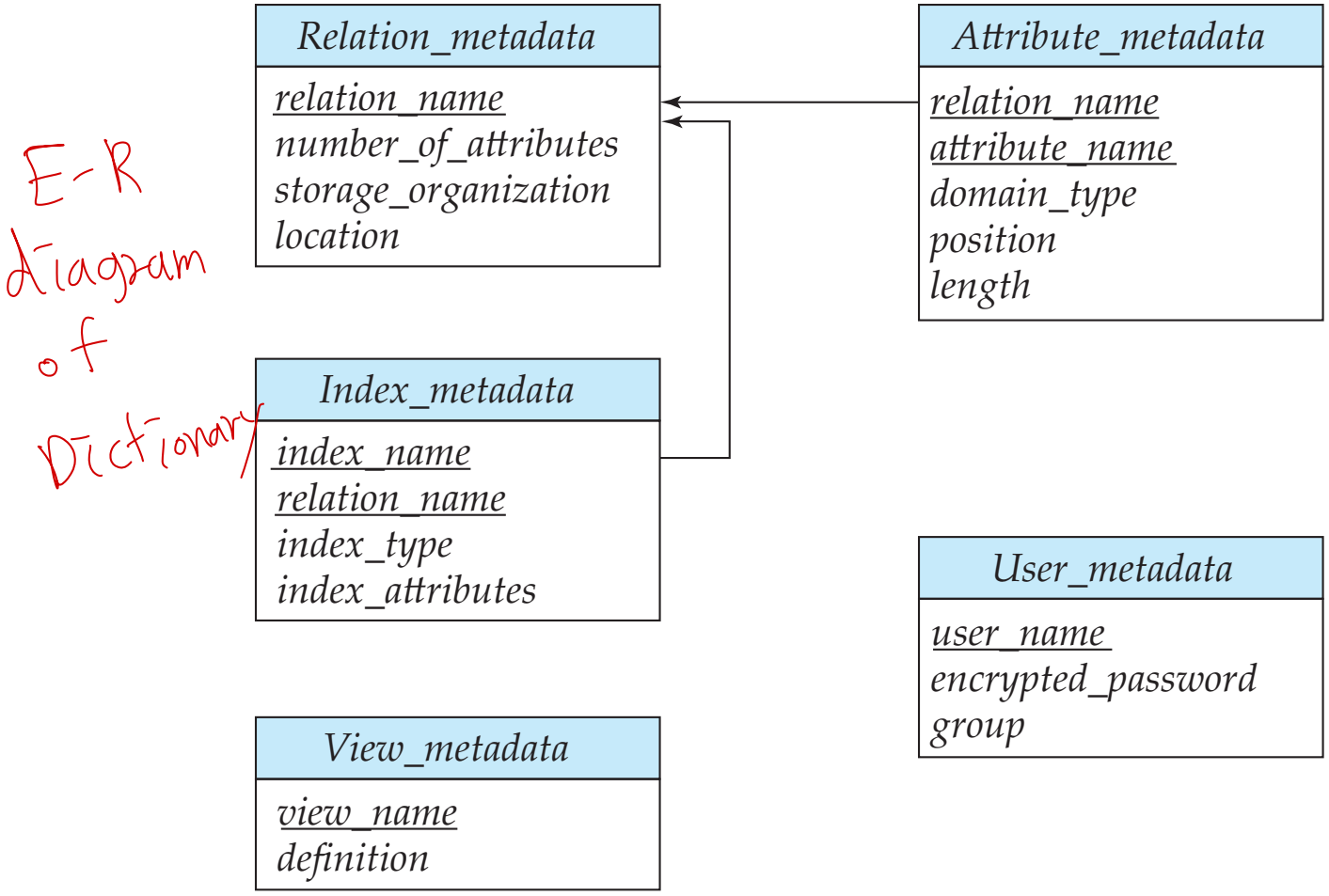
Data Dictionary

- **Data dictionary** (aka. **system catalog**) stores **metadata**
  - Information about tables
    - File names of tables
    - File formats (Sequential, Heap, …)
    - names, types and lengths of attributes of each table
  - Statistical and descriptive data
    - number of records in each table
    - Key distribution, etc
  - Information about indexes (Chapter 14)



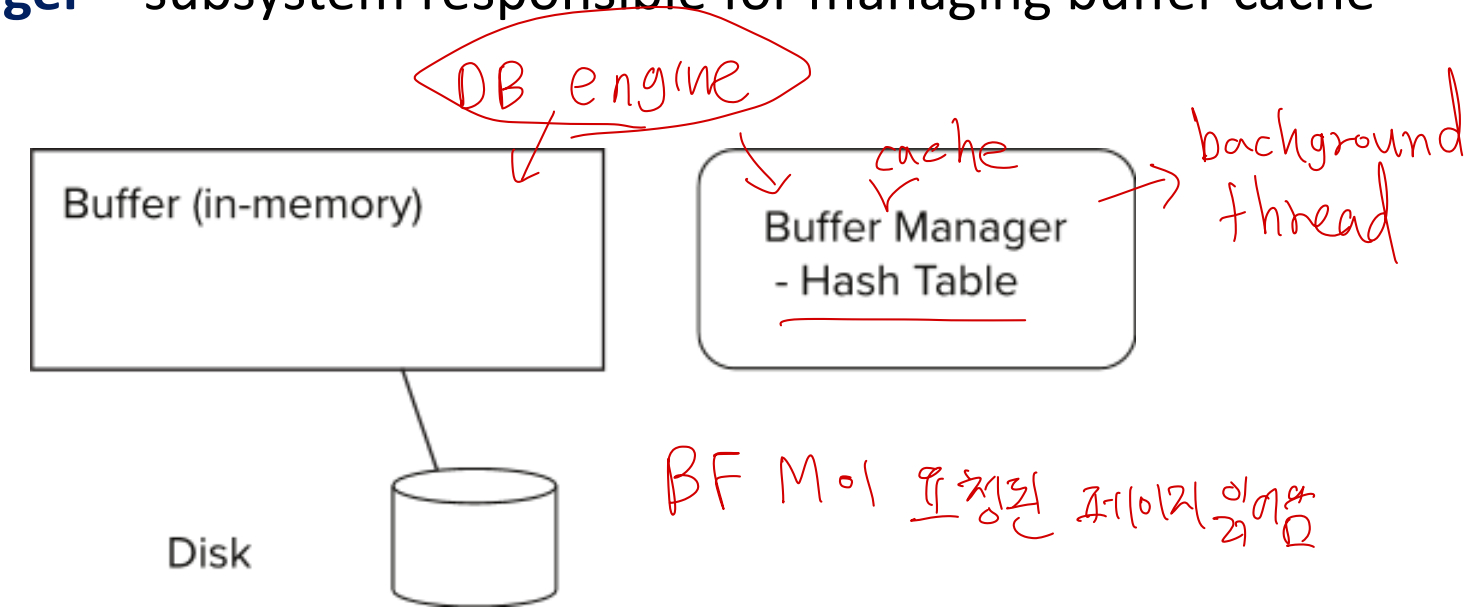
Data Dictionary as a Database

- Data Dictionary is also a Database
- Relational representation



Storage Access

- Block (Page) = a unit of storage allocation and data transfer (I/O).
- DBMS tries to minimize the number of block transfers
  - by keeping as many blocks as possible in main memory.
- **Buffer** (or **Buffer Cache**) – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for managing buffer cache

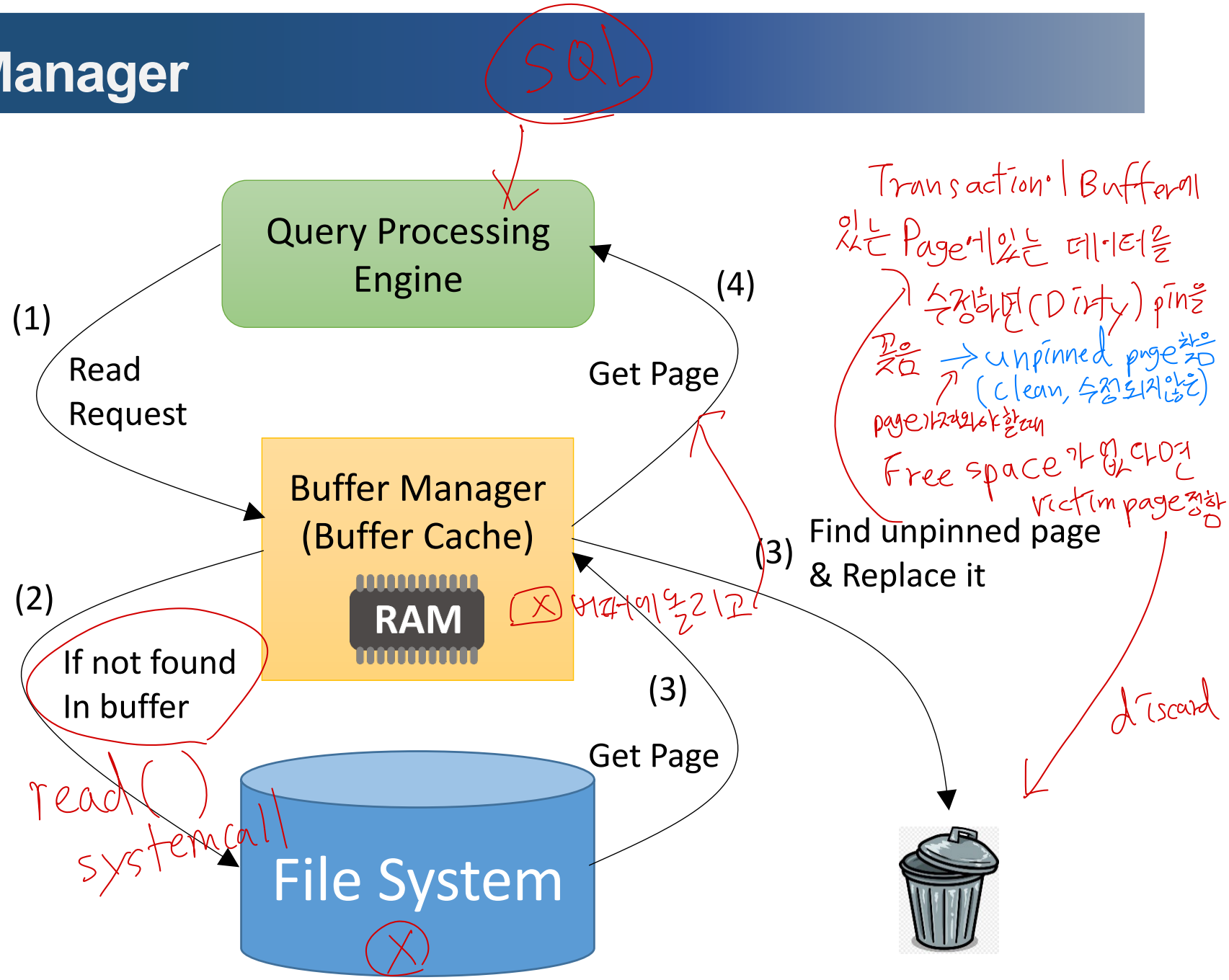


Buffer Manager

- DBMS calls on the buffer manager when it needs a block
  - 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 some other block to make space for the new block.
      - Replaced block written back to disk if it was modified (i.e., dirty)
    - Reads the block from the disk to the buffer, and returns the memory address of the block to the requester.

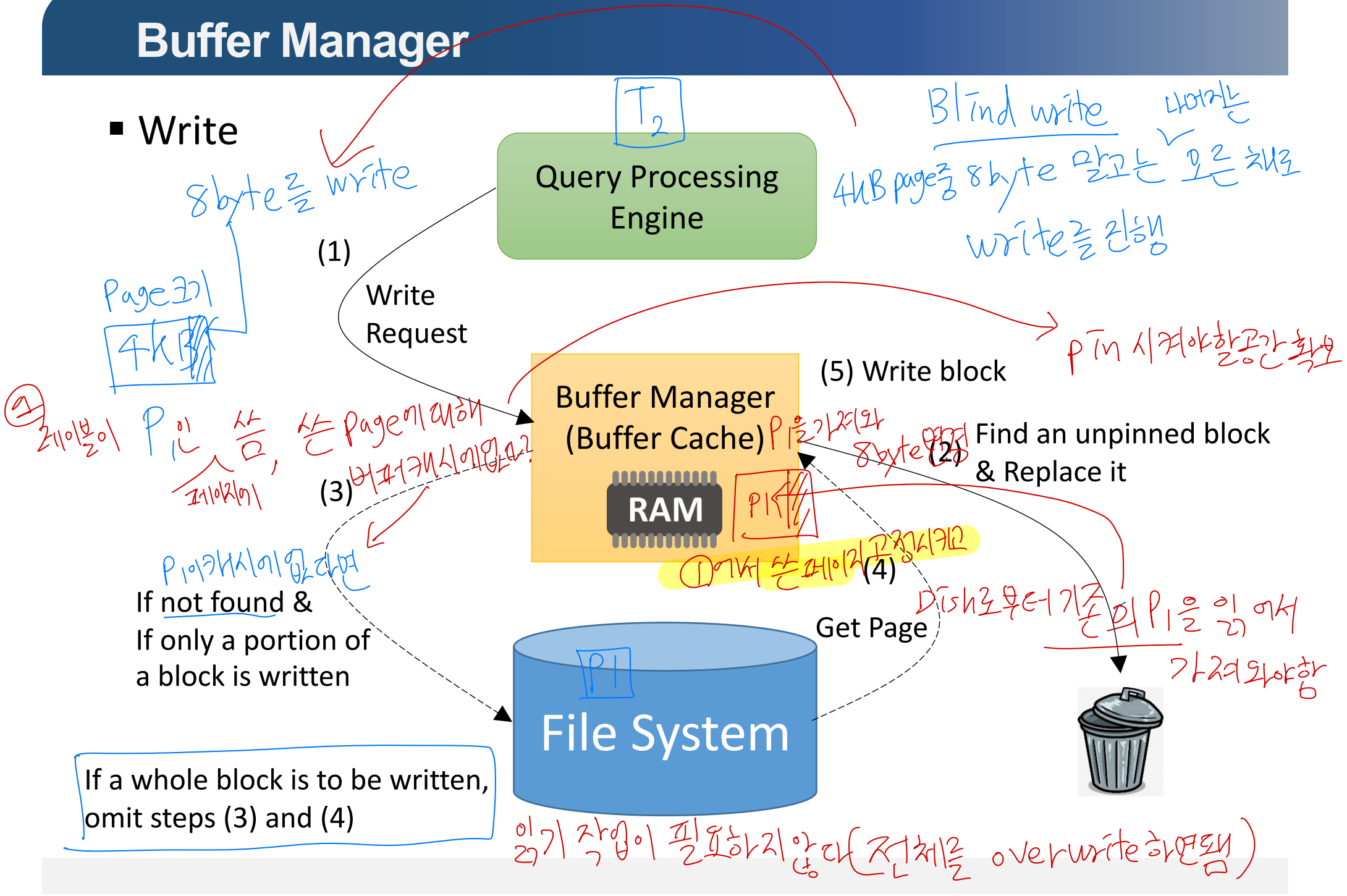
Buffer Manager

- Read



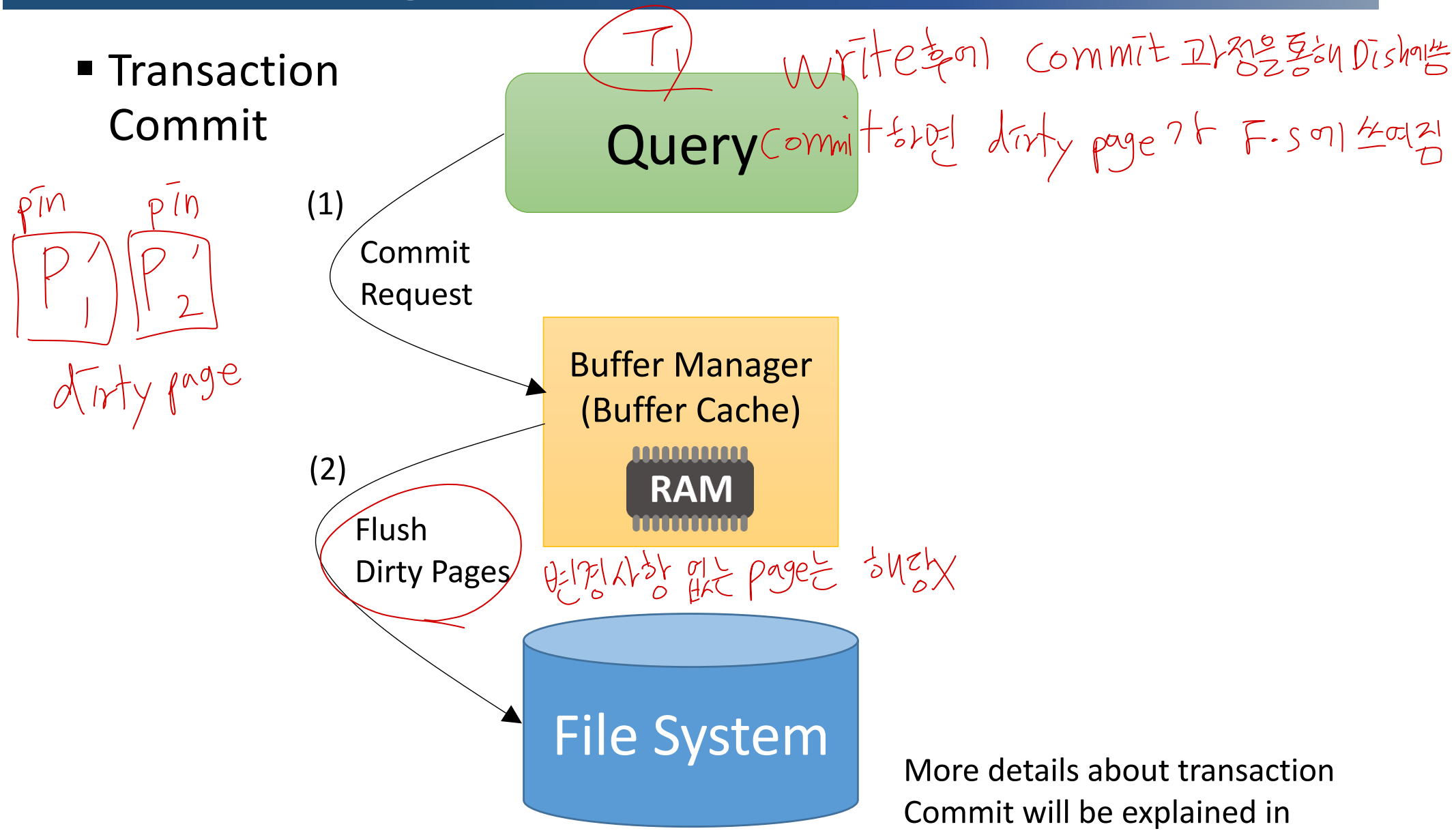
Buffer Manager

- Write



Buffer Manager

- Transaction Commit



More details about transaction Commit will be explained in Chapter 19.

Buffer Manager & Replacement Policy

- **Pinned block** is not allowed to be written to disks
  - **Pin** before reading/writing data from a block
  - **Unpin** when read /write is complete
  - Multiple concurrent pin/unpin operations possible
    - Keep a pin count
    - Can be evicted only if pin count = 0
- **Shared and exclusive locks on buffer**
  - Readers: need shared lock (read lock)
  - Writers: need exclusive lock (write lock)
  - **Locking rules:**
    - Only one process can get exclusive lock at a time
    - Shared lock cannot be given if exclusive lock is held
    - Multiple processes may be given shared lock concurrently

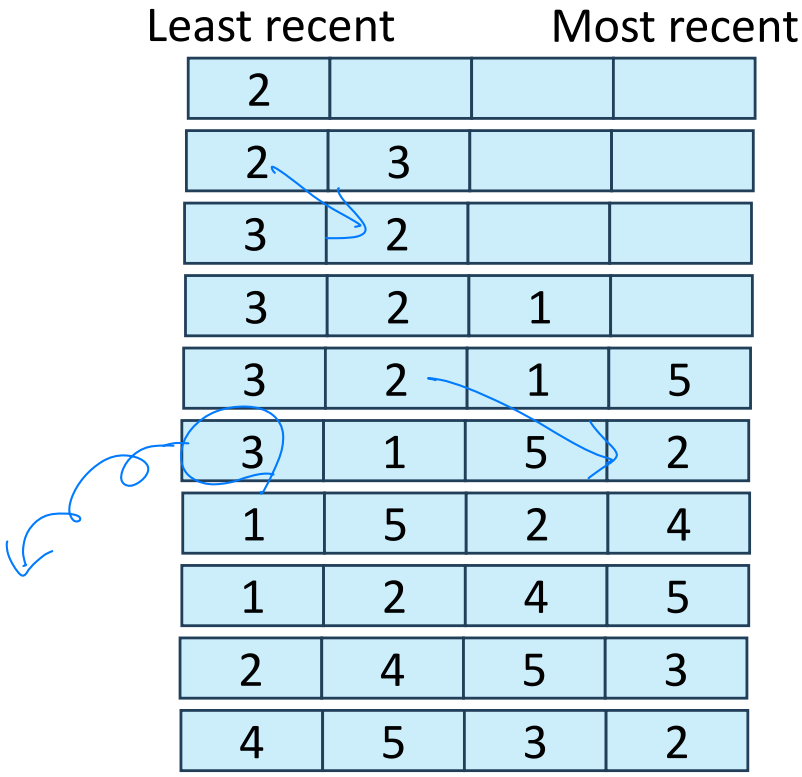
Buffer-Replacement Policies

- Most popular policy in OS: **Least Recently Used** (LRU)
  - Idea behind LRU
    - use past block reference patterns to predict future references → Heuristic policy

Block Reference Sequence

Page Request	Hit/Miss	Evict
2	Miss	
3	Miss	
2	Hit	
1	Miss	
5	Miss	
2	Hit	
4	Miss	3
5	Hit	
3	Miss	1
2	Hit	

Buffer Cache



Buffer-Replacement Policies

- Most popular policy in OS: **Least Recently Used** (LRU)
  - LRU can be bad for database queries
  - Sequential Flooding ← Join 때문에 LRU안좋음 (DBMS는!)
    - Example: when computing the join

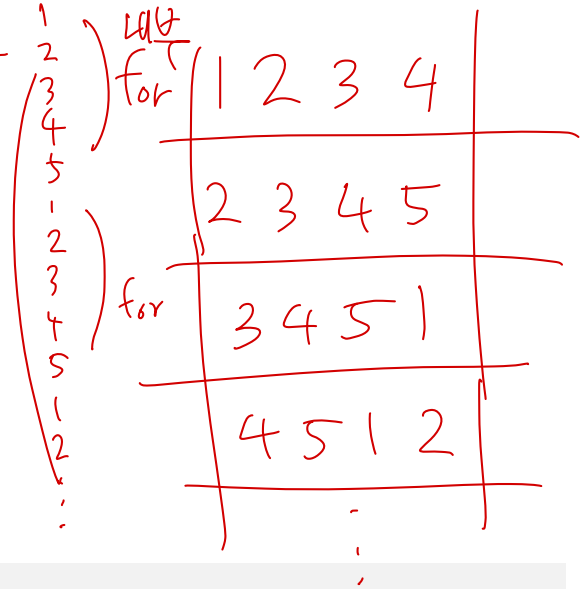
```
for each tuple tr of r do
    for each tuple ts of s do
        if the tuples tr and ts match …
```

    - If ts is larger than the buffer cache, no cache hits

순차적으로 읽다가

캐시가 바로바로 채워지고 이전에 있은 데이터가 계속 밀려남

나중에 재사용되는데



Buffer-Replacement Policies in DBMS

- DBMS access disks in well-defined patterns
  - DBMS is aware of the disk access patterns for each query processing algorithm, e.g., sequential scan, join, etc.

metadata이용 → 가 query에 맞는 효율적인 정책을 선택할 수 있다

- Data Dictionary manages statistics
  - Probability that a request will reference a particular table
  - Size of tables
  - Key distribution of tables
  - Etc.

- DBMS can do better than LRU

Buffer-Replacement Policies in DBMS

- **Toss-immediate** strategy
  - Evicts a block as soon as the final record of that block has been processed even if free space is not required

Join → 자주사용하지 않는 테이블에 대해 주로사용

avoid cache update

Block Reference Sequence

Page Request	Hit/Miss	Evict
…	…	
2	Hit	
4	Miss	
5	Hit	
3	Hit	
2	Hit	

Disk로부터 fetch는 일어나는 변경X

Buffer Cache

Least recent			Most recent
3	2	1	5
3	2	1	5
3	2	1	5
3	2	1	5
3	2	1	5
3	2	1	5

Buffer-Replacement Policies in DBMS

- **Most recently used (MRU) strategy**
  - After a block is processed, the block becomes the candidate for replacement, i.e., most recently used block.

왜냐 temporal locality의 경향이라는 것을 알고 MRU사용

Block Reference Sequence

Page Request	Hit/Miss	Evict
…	…	
2	Hit	
4	Miss	2
5	Hit	
3	Hit	
2	Miss	

Buffer Cache

Least recent Most recent

3	2	1	5
3	1	5	2
3	1	5	4
3	1	4	5
1	4	5	3
1	4	5	2