# 第7章: 存储管理

## Storage Management

邹兆年

哈尔滨工业大学
计算机科学与技术学院
海量数据计算研究中心
电子邮件: znzou@hit.edu.cn

2021年春



## Outline[1]



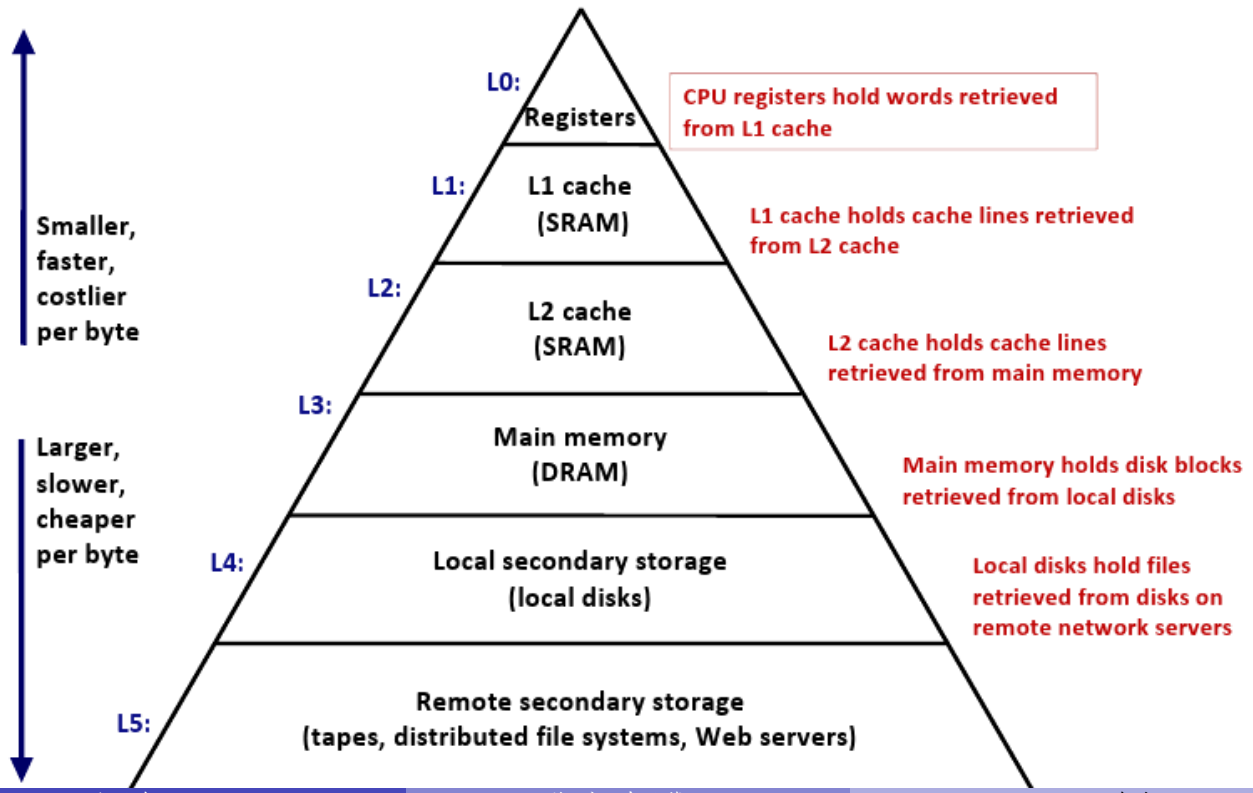
[1] Updated on April 8, 2021

# Storage Media



# The Memory Hierarchy (存储层级)

Memory in a computer system is arranged in a *hierarchy*



## Primary Storage (主存储器)

Data in primary storage can be operated on directly by CPUs using `load/store`

- Registers (寄存器)
- Cache (高速缓存)
- Main memory (内存)

Primary storage is byte-addressable (按字节寻址)



## Secondary Storage (二级存储器)

Data in secondary storage cannot be processed directly by CPUs; It must first be copied into primary storage using `read/write`

- Magnetic disks (磁盘)/Hard disk drives (HDD, 机械硬盘)
- Flash memory (闪存)/Solid state drives (SSD, 固态硬盘)

Secondary storage is block-addressable (按块寻址) and online (联机)

# Tertiary Storage (三级存储器)

Data in tertiary storage must first be copied into secondary storage

- Magnetic tape (磁带)
- Optical disks (光盘)
- Network Storage (网络存储)

Tertiary storage is block-addressable and offline (脱机)



# Access Time (访存时间)

| Access time (ns) | Storage media |
|---:|---|
| 0.5 | L1 cache |
| 7 | L2 cache |
| 100 | DRAM |
| 150,000 | SSD |
| 10,000,000 | HDD |
| 30,000,000 | Network storage |
| 1,000,000,000 | Tape |

## Data Transfer Between Levels

- Cache
  ↕ Unit: cache lines (缓存行), size: 64B
- Main memory
  ↕ Unit: blocks (块)/pages (页), size: 512B–16KB
- Secondary storage
  ↕ Unit: blocks/pages, size: 512B–16KB
- Tertiary storage



## Categories of Storage Media

### Volatile Storage (易失存储器)

Data in volatile storage is lost when the computer is restarted (after a shutdown or a crash)

- Primary storage

### Non-volatile Storage (非易失存储器)

Data in non-volatile storage is retained when the computer is restarted

- Secondary storage
- Tertiary storage

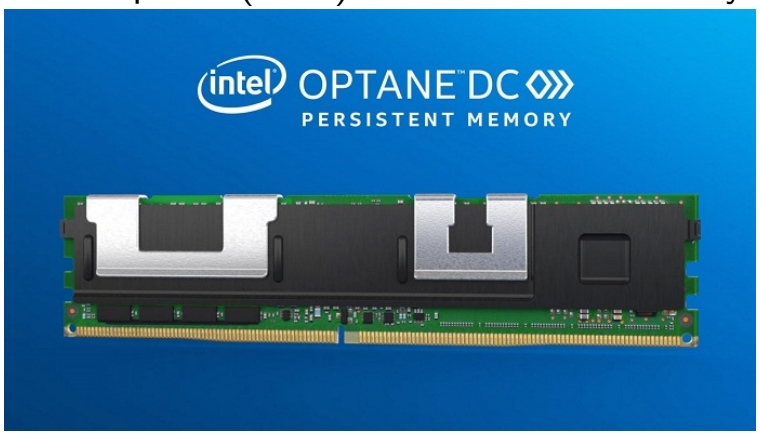
Non-volatile main memory is emerging!

# Non-volatile Main Memory (NVM, 非易失主存)

NVM is also known as persistent memory or storage-class memory (SCM)

- Byte-addressable
- Non-volatile (persistent)

Intel Optane (傲腾) DC Persistent Memory

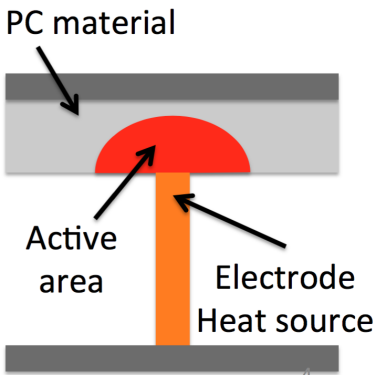



Persistent Memory Developing Toolkit (PMDK) (`http://pmem.io`)



# Phase-Change Memory (PCM, 相变存储器)

- PCM stores data using phase-change material (相变材料)
  - Amorphous phase (非晶态): high resistivity $\longrightarrow$ 0
  - Crystalline phase (晶态): low resistivity $\longrightarrow$ 1
- Set phase via current pulse
  - Fast cooling $\rightarrow$ Amorphous
  - Slow cooling $\rightarrow$ Crystalline



# Characteristics of NVM

| Characteristics | DRAM | PCM | Memristor | MRAM | SSD | HDD |
|---|---|---|---|---|---|---|
| Volatility | Yes | No | No | No | No | No |
| Addressability | Byte | Byte | Byte | Byte | Block | Block |
| Volume | GB | TB | TB | TB | TB | TB |
| Read latency | 60ns | 50ns | 100ns | 20ns | 25$\mu$s | 10ms |
| Write latency | 60ns | 150ns | 100ns | 20ns | 300$\mu$s | 10ms |
| Energy per bit | 2pJ | 2pJ | 100pJ | 0.02pJ | 10nJ | 0.1J |
| Endurance | $10^{16}$ | $10^{10}$ | $10^{8}$ | $10^{15}$ | $10^{5}$ | $10^{16}$ |

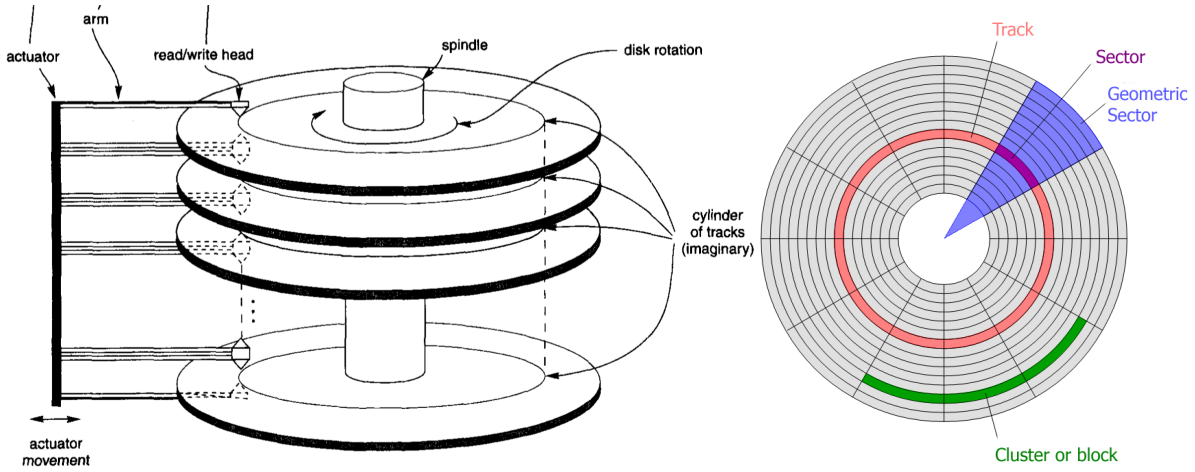
NVM will become a critical level in the memory hierarchy and plays important roles in computer systems and DBMS



# Magnetic Disks (磁盘)

A magnetic disk is composed by two components:

- Disk assembly: sectors (扇区) $\subset$ tracks (磁道) $\subset$ cylinders (柱面)
- Head assembly: disk heads (磁头) and disk arms (磁臂)



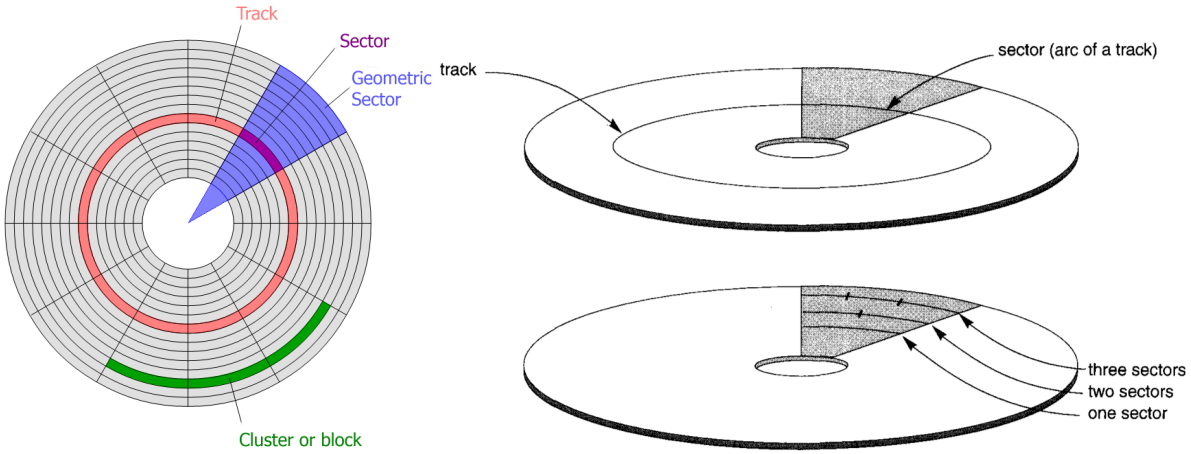
## Sectors (扇区)

Every track on disks is divided into smaller sectors

- The division of a track into sectors is hard-coded on the disk surface and cannot be changed

Sector organizations

- Approach #1: Sectors subtend a fixed angle
- Approach #2: Sectors maintain a uniform recording density






## Disk Blocks (磁盘块)/Pages (页)

The OS divides a track into equal-sized disk blocks (or pages) during disk formatting

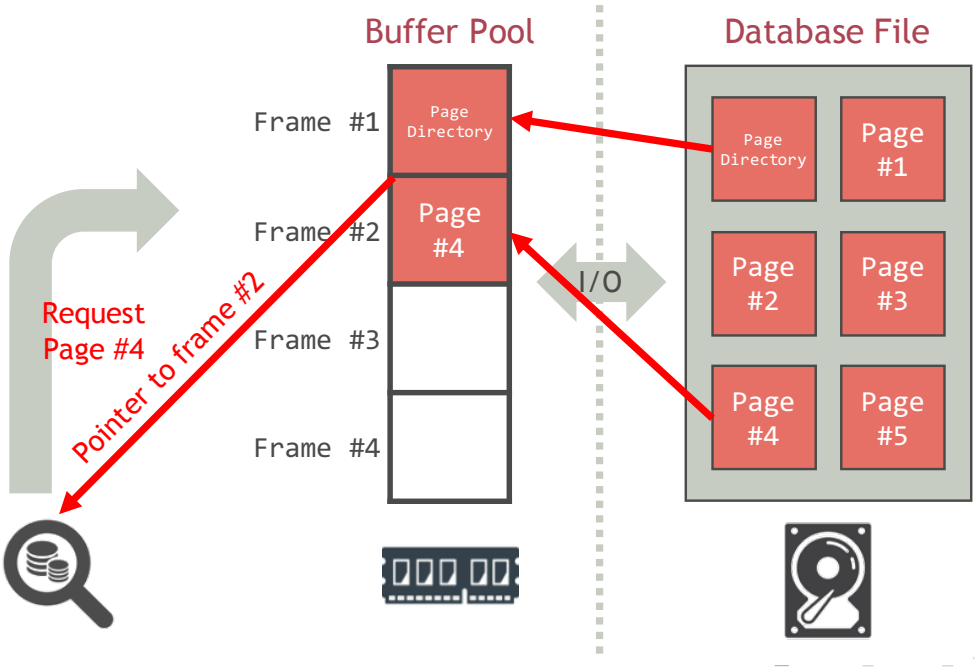
- The size of a disk block can be set as a multiple of the sector size
- Block size is fixed during disk formatting and cannot be changed dynamically
- Typical disk block size is 512B–4KB

### Logical Block Address (LBA, 逻辑块地址)

The LBA of a disk block is a number between 0 and $n-1$ (assuming the total capacity of the disk is $n$ blocks)

# Disk-Oriented Database Storage

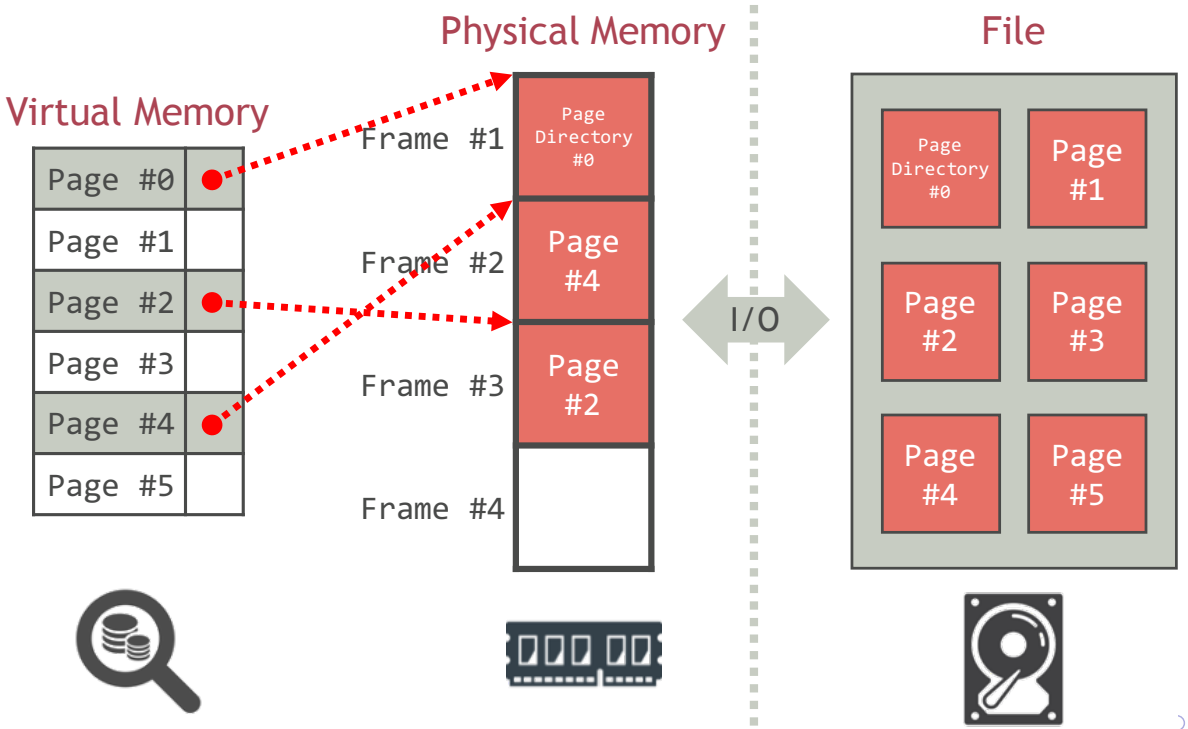
- Data is persistently stored on secondary storage
- Data is loaded into main memory when it is going to be read or modified (but not yet found in main memory)





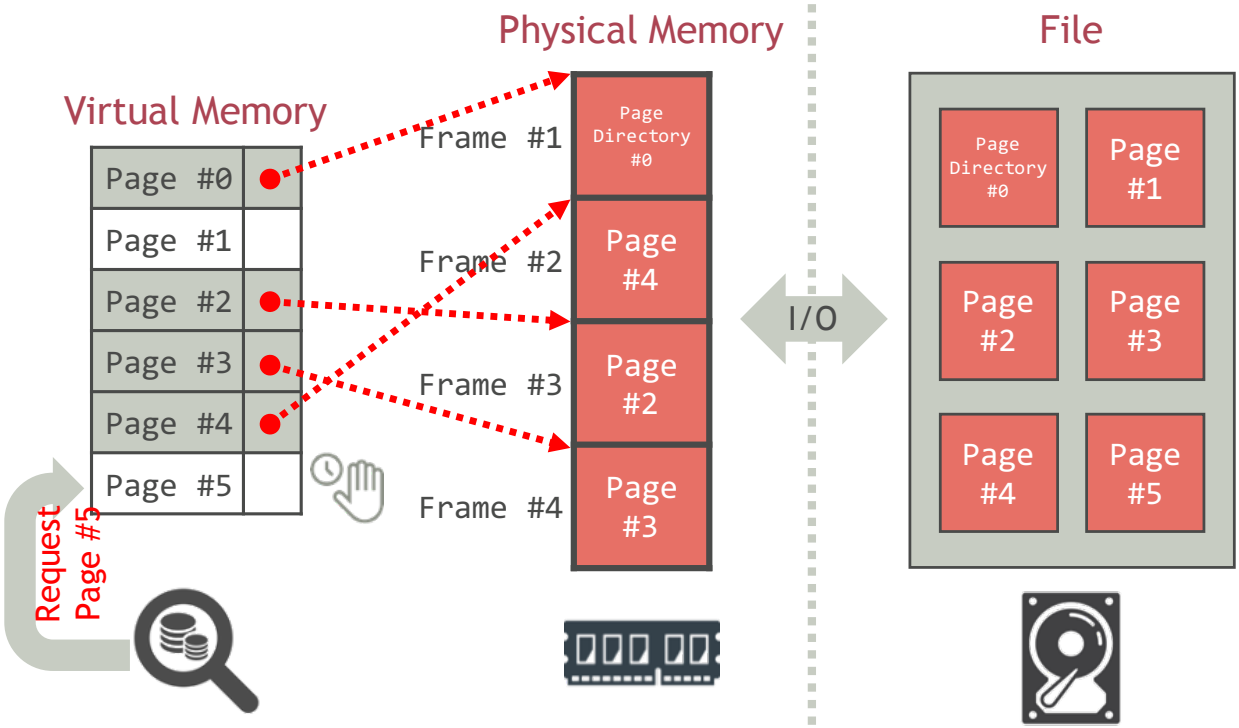

# Why Not Use the OS?

One can use memory mapping (mmap, 内存映射) to store the contents of a file into a process' address space



# Why Not Use the OS? (Cont'd)

The OS is responsible for moving the files' pages in and out of memory






# Disk-Oriented Database Storage (Cont'd)

## 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

## Spatial Control (空间方面的控制)

- Where to write pages on disk
- The goal is to keep pages that are used together often as physically close together as possible on disk (data locality)

## Temporal Control (时间方面的控制)

- When to read pages into memory, and when to write them to disk
- The goal is to minimize the number of stalls from having to read data from disk

# Representation of Databases on Disks



# Representation of Databases on Disks
## Value Representation

## Representation of Numbers

### INT/INTEGER/BIGINT/SMALLINT/TINYINT (整数)

- C/C++ representation

### FLOAT/DOUBLE/REAL (浮点数)

- IEEE-754 standard
- Typically faster than arbitrary precision numbers, but can have rounding errors

### NUMERIC/DECIMAL (定点数)

- Arbitrary precision and scale
- Used when rounding errors are unacceptable
- Typically stored in an exact, variable-length binary representation with additional meta-data (like a `VARCHAR` but not stored as a string)



## Representation of Strings

### CHAR(n)/BINARY(n) (定长字符串/字节串)

- an array of $n$ bytes
- If the length of a string is shorter than $n$, the array is padded with a special null character
- Example: `'cat'` is represented as | 'c' | 'a' | 't' | 0 | 0 | in `CHAR(5)`

### VARCHAR/VARBINARY/TEXT/BLOB (变长字符串/字节串)

- Header with length, followed by data bytes
- Example: `'cat'` is represented as | 3 | 'c' | 'a' | 't' | in `VARCHAR(5)`
- C/C++ null-terminated string representation is not used

### BIT(n) (定长位串)

- an array of $\lceil n/8 \rceil$ bytes
- Example: `BIT(12)`, 010111110011 $\rightarrow$ | 01011111 | | 00110000 |

## Representation of Other Types of Values

### TIME/DATE/DATETIME/TIMESTAMP（时间/日期）

- 32/64-bit integer of (micro)seconds since Unix epoch

### ENUM（枚举型值）

- An enumerated type with $n$ items is represented as `BIT`($\lceil \log_2 n \rceil$)
- Example: `ENUM(RED, GREEN, BLUE, YELLOW)`, `RED` $\rightarrow$ `00`, `GREEN` $\rightarrow$ `01`, `BLUE` $\rightarrow$ `10`, `YELLOW` $\rightarrow$ `11`

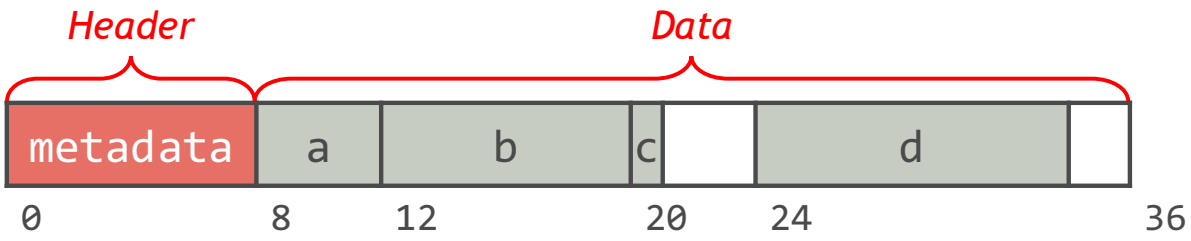


## Representation of Databases on Disks
## Tuple Layout

## Tuple Layout

A tuple (元组) is essentially a sequence of bytes

- It is the job of the DBMS to interpret those bytes into attribute types and values
- The DBMS's catalogs (目录) contain the schema information about tables that the DBMS uses to figure out the tuple's layout

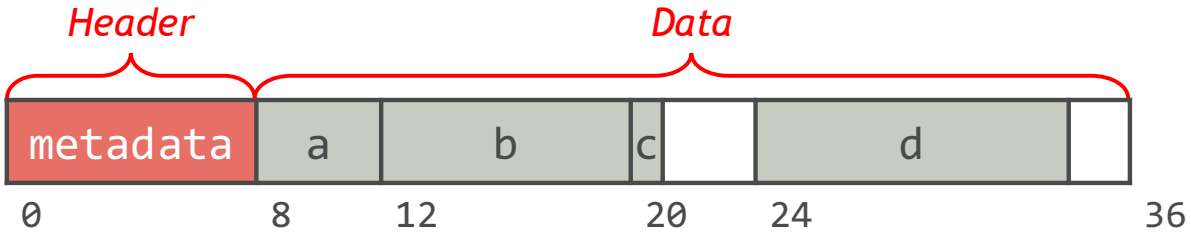





## Tuple Header (头部)

Each tuple is prefixed with a header that contains meta-data about it

- A pointer to where the DBMS stores the schema for this type of tuple
- The length of the tuple
- Visibility info (concurrency control)
- A bitmap for `NULL` values

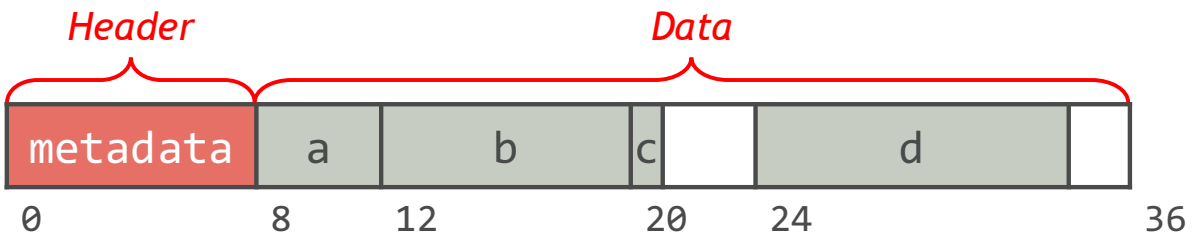


## Tuple Data

The tuple data part concatenates all attribute values of the tuple

- Attributes are typically stored in the order that you specify when you create the table
- Each attribute value aligns at an offset from the beginning of the tuple by a multiple of 4 bytes/8 bytes

```
CREATE TABLE Foo (
  a INT NOT NULL,
  b BIGINT NOT NULL,
  c CHAR NOT NULL,
  d VARCHAR(10) NOT NULL);
```






# Representation of Databases on Disks
# Page Layout

# Database Pages (数据库页面)

A page is a fixed-size block of data (512B–16KB)

- It can contain tuples, meta-data, indexes, log records ...
- Most DBMSs do not mix page types
- Some DBMSs require a page to be self-contained

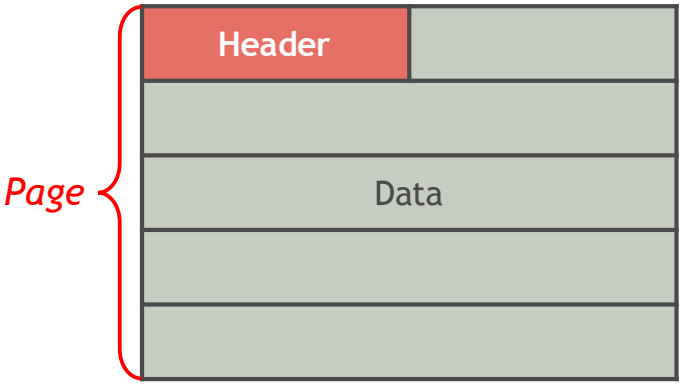
Each page is given a unique identifier as its page ID (页号)

- The DBMS uses an indirection layer to map page IDs to physical locations



# Page Layout (页面布局)

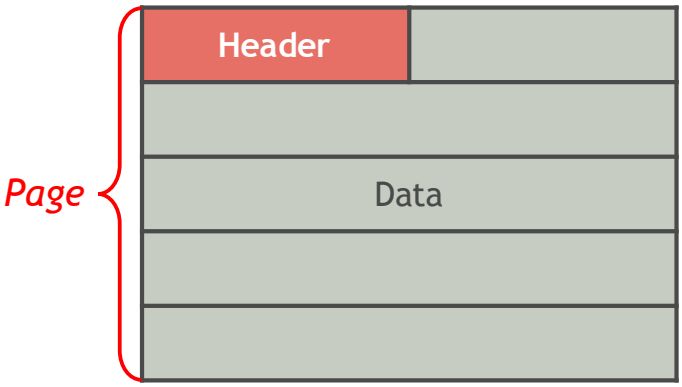
A page is comprised of the page header and the page data



## Page Header

Every page contains a header of metadata about the page's contents

- Page size
- Checksum (校验码)
- DBMS version
- Transaction visibility
- Compression information



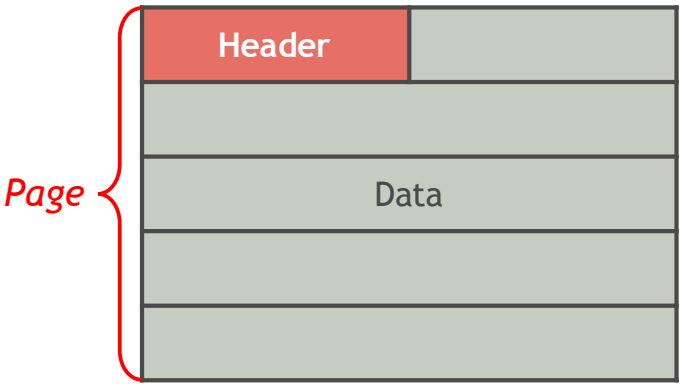



## Page Data

Here we only consider pages storing a number of tuples

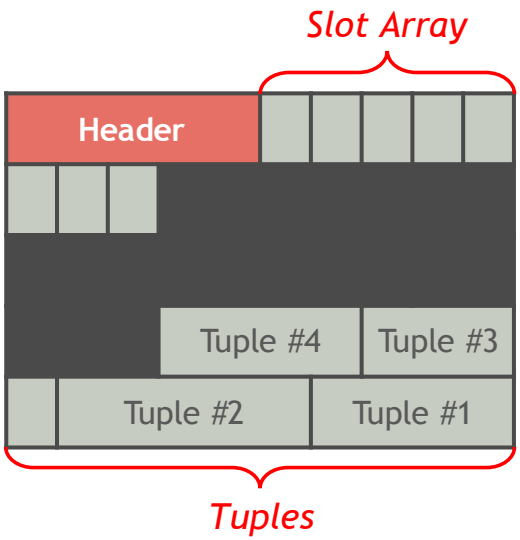
How to organize the data stored inside of a page?

- Approach #1: Tuple-oriented
- Approach #2: Log-structured



# Tuple-Oriented Page Layout

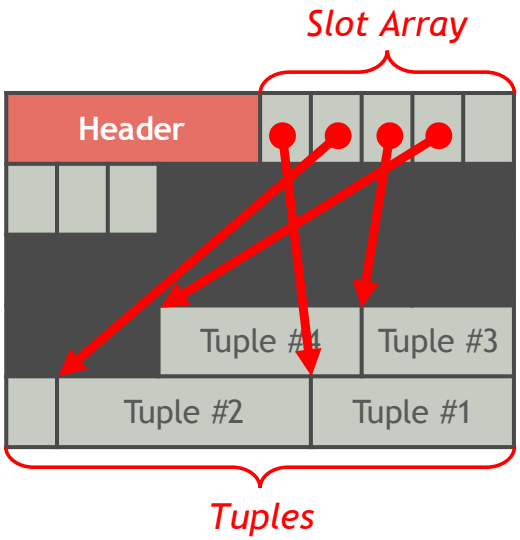
The most common page layout scheme is called slotted pages (分槽页面)






# Slot Pages (分槽页面)

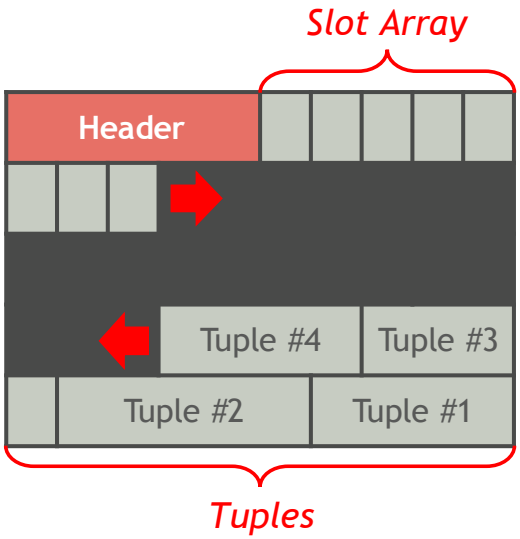
The slot array maps the used "slots" to the tuples' starting position offsets



## Slotted Pages (Cont'd)

The header of a slotted page keeps track of

- The # of used slots
- The offset of the starting location of the last slot used



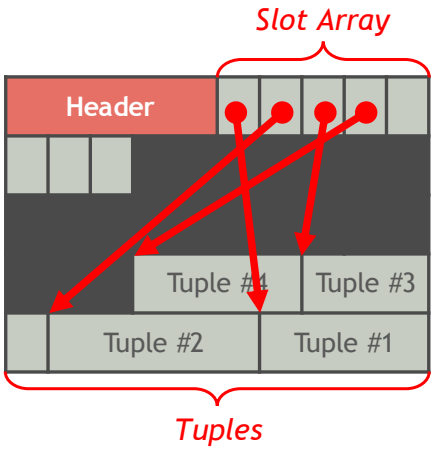



## Record ID (记录号)

To keep track of individual tuples, the DBMS assigns each tuple with a unique record identifier

The most common record ID is `(PageID, Slot#)`

- `PageID`: the ID of the page containing the tuple
- `Slot#`: the # of the slot where the tuple is stored in the page



## Large Values

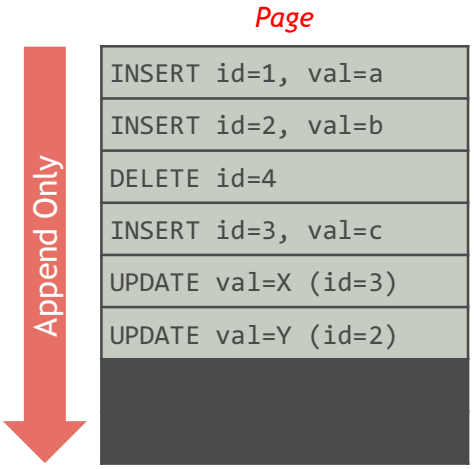
- Most DBMSs don't allow a tuple to exceed the size of a single page
- To store values that are larger than a page, the DBMS uses separate overflow storage pages



## Log-Structured Page Layout

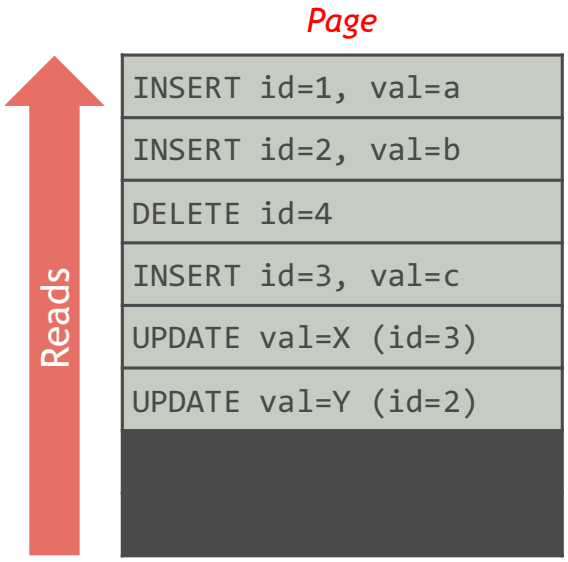
Instead of storing tuples in pages, the DBMS only appends log records (日志记录) to the file of how the database was modified

- Inserts store the entire tuple
- Deletes mark the tuple as deleted
- Updates contain the delta of just the attributes that were modified



# Log-Structured Page Layout (Cont'd)

To read a tuple, the DBMS scans the log backwards and "recreates" the tuple to find what it needs

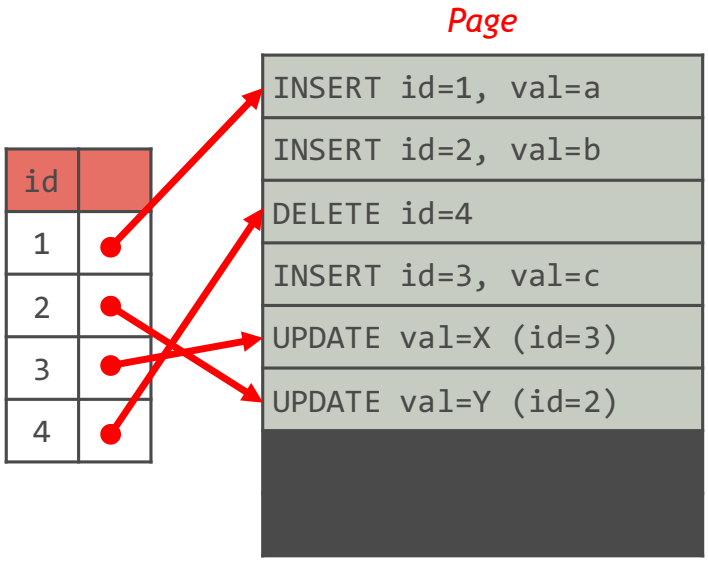



(id=1, val=a); (id=2, val=Y); (id=3, val=X); id=4 deleted;



# Log-Structured Page Layout (Cont'd)

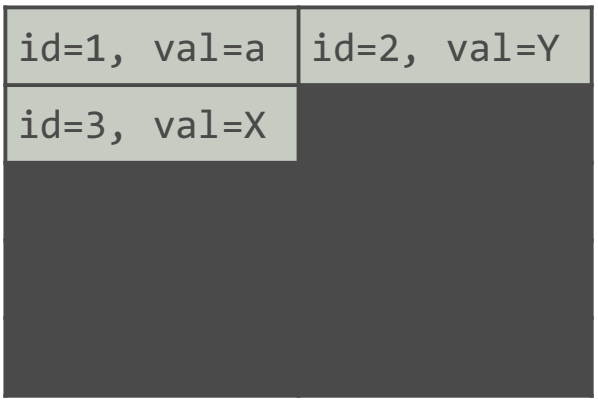
Build indexes to allow the DBMS to jump to locations in the log



# Log-Structured Page Layout (Cont'd)

Periodically compact the log

*Compacted Page*






# Representation of Databases on Disks
# File Organization

## File Storage

The DBMS stores a database as one or more files on disk

- The OS doesn't know anything about the contents of these files

The storage manager is responsible for maintaining a database's files. It organizes the files as a collection of pages.

- It tracks data read/written to pages
- It tracks the available space



## Page Storage Architecture

Different DBMSs manage pages in files on disk in different ways

- Approach #1: Heap File Organization
- Approach #2: Sequential/Sorted File Organization
- Approach #3: Hashing File Organization (Chapter 8)

# Heap File Organization (堆文件组织)

A heap file (堆文件) is an unordered collection of pages where tuples that are stored in random order

- Create/get/write/delete pages
- Must also support iterating over all pages

Need meta-data to keep track of what pages exist and which ones have free space

Two ways to represent a heap file

- Approach #1: Linked lists
- Approach #2: Page directory

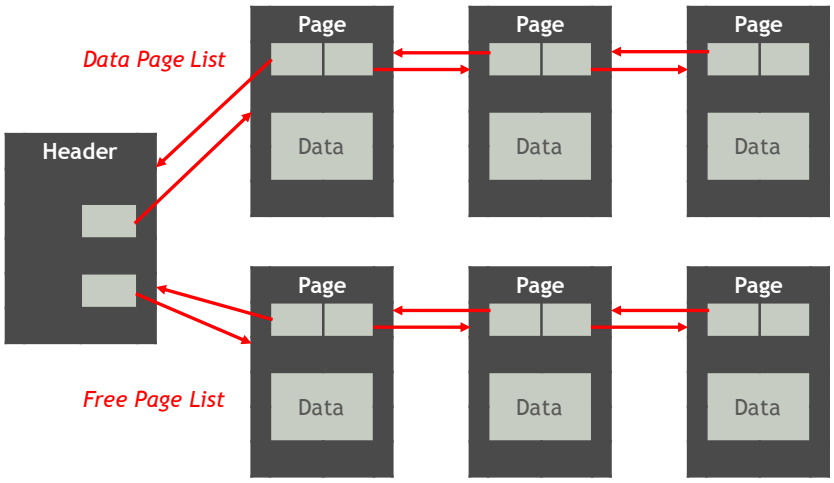


# Heap Files: Linked Lists (链表法)

Maintain a header page at the beginning of the file that stores two pointers

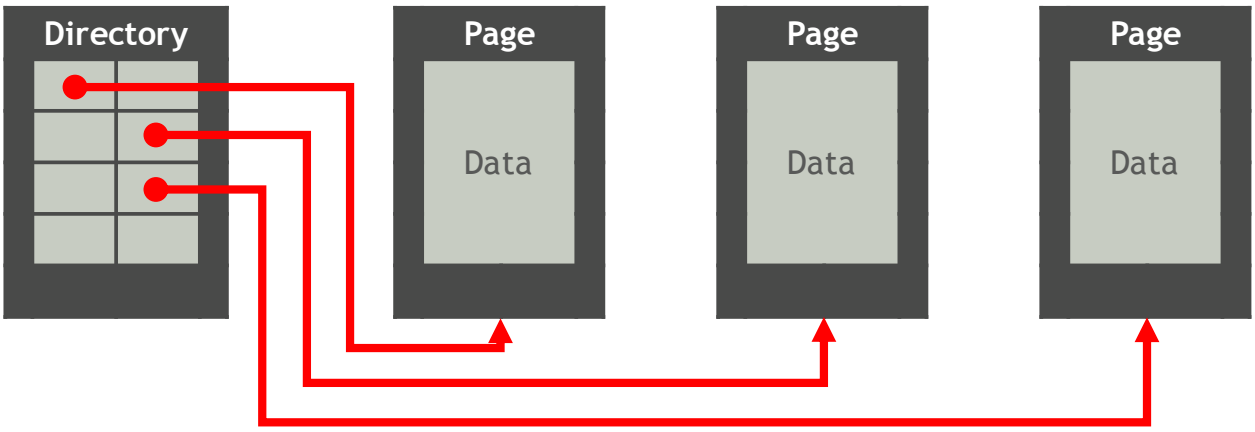
- The head of the free page list
- The head of the data page list

Each page keeps track of the number of free slots in itself



## Heap Files: Page Directory (页目录法)

- The DBMS maintains special pages that tracks the location of data pages in the database files
- The directory also records the number of free slots per page
- The DBMS has to make sure that the directory pages are in sync with the data pages






## Sequential/Sorted File Organization (顺序/有序文件组织)

A sequential/sorted file is an ordered collection of pages where tuples that are stored in sorted key order

- Ordered files are rarely used unless a primary index is used

# System Catalogs



# System Catalogs (系统目录)

A DBMS stores meta-data about databases in its internal catalogs

- Tables, columns, indexes, views
- Users, permissions
- Internal statistics

Almost every DBMS stores a database's catalog in itself

## System Catalogs (系统目录)

You can query the DBMS's internal `INFORMATION_SCHEMA` catalog to get info about the database

- ANSI standard set of read-only views that provide info about all of the tables, views, columns, and procedures in a database

DBMSs also have non-standard shortcuts to retrieve this information

**Example (MySQL System Catalogs)**

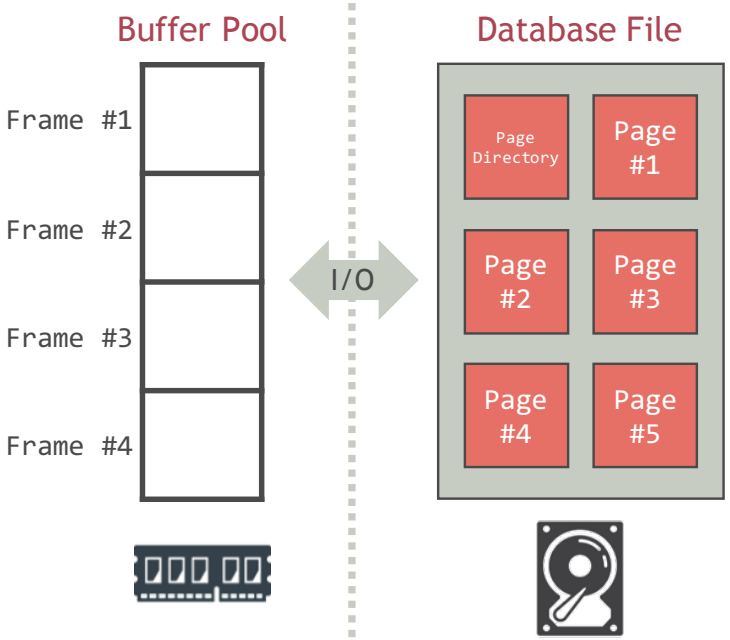
- List all the tables in the current database: `show tables;`
- List the schema of a table: `describe student;`



## Buffer Management

# Buffer Pool (缓冲池)

- The available memory region is partitioned into an array of fixed-size pages, which are collectively called the buffer pool (缓冲池)
- The pages in the buffer pool are called frames (页框)

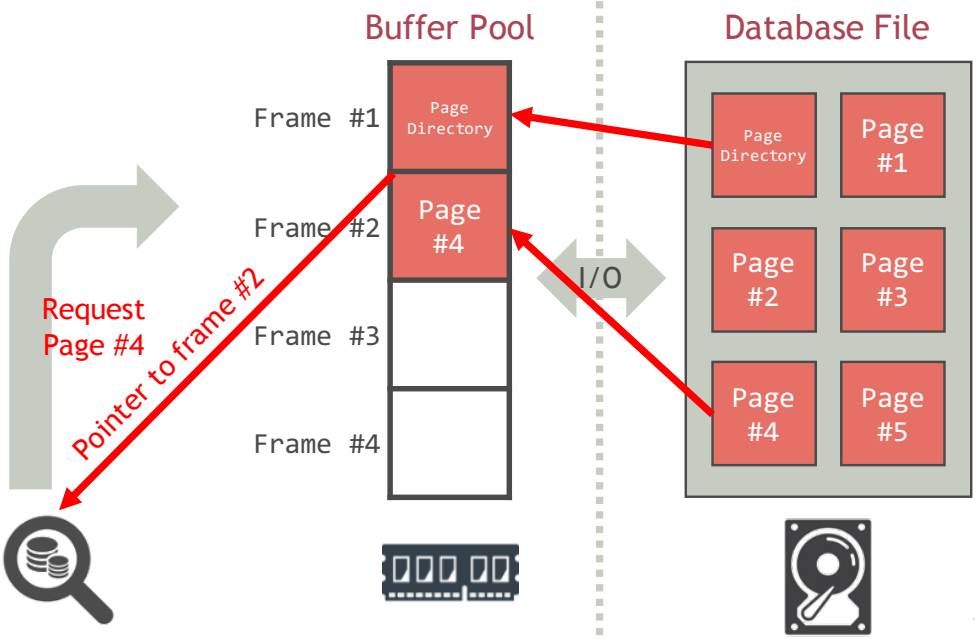





# Buffer Manager (缓冲区管理器)

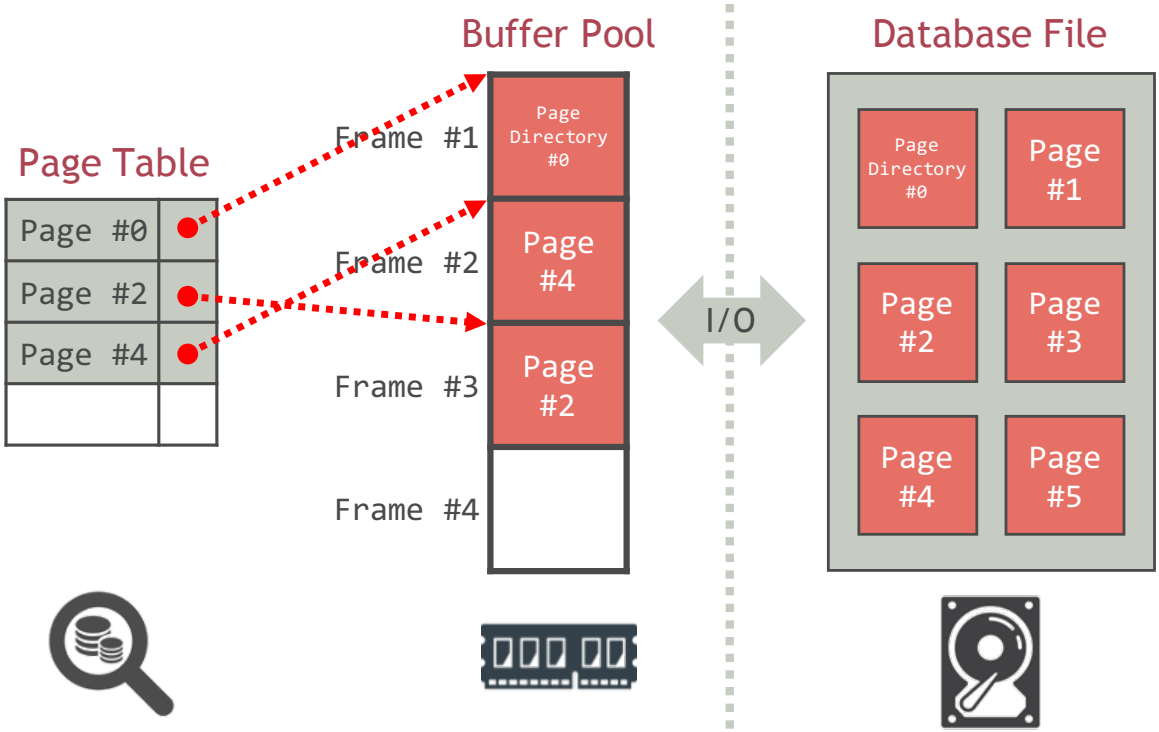
The buffer manager is responsible for bringing pages into the buffer pool as needed

- The buffer manager decides what existing page in the buffer pool to replace to make space for the new page (if the buffer pool is full)



# Buffer Pool Internals: Page Table (页表)

The page table keeps track of pages that are currently in the buffer pool





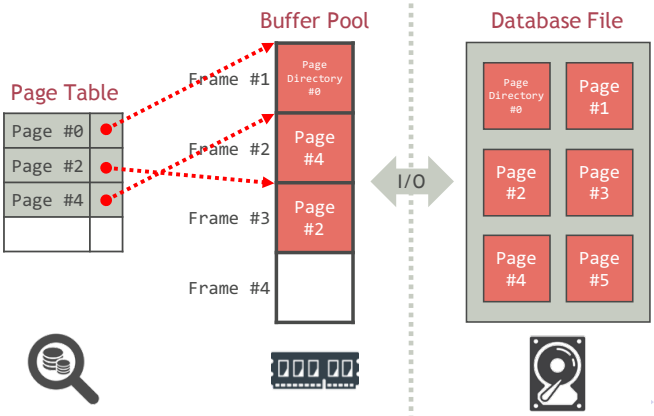

# Page Table vs. Page Directory

The page directory is the mapping from page IDs to page locations in the database files

- All changes must be recorded on disk to allow the DBMS to find on restart

The page table is the mapping from page IDs to a copy of the page in buffer pool frames

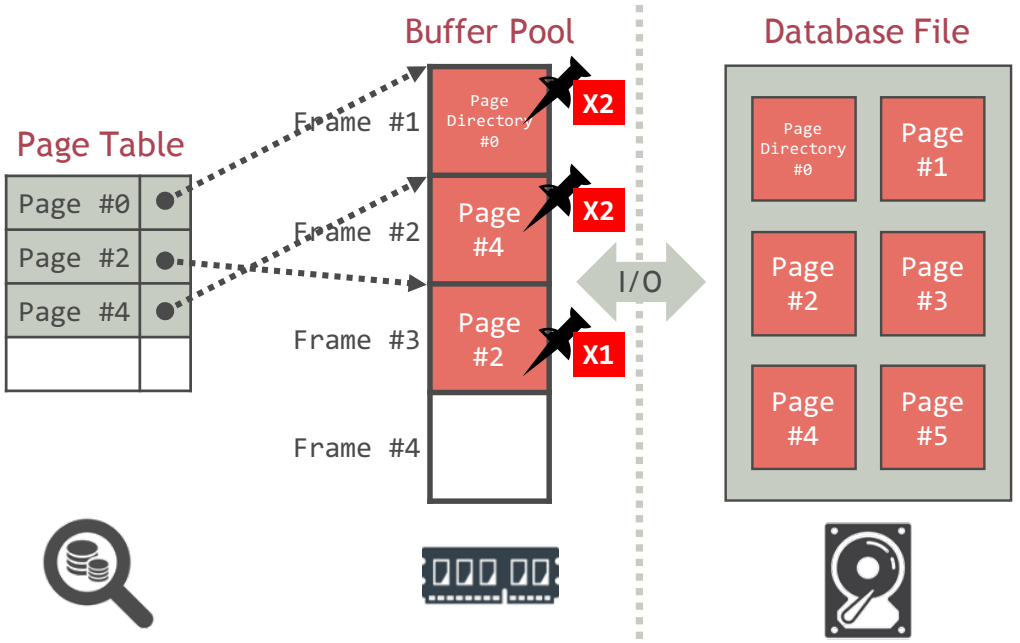
- This is an in-memory data structure that does not need to be stored on disk



# Buffer Pool Internals: Frame's Meta-Data

The buffer manager maintains two variables for each frame

- `pin_count`: the number of times that the page currently in the frame has been requested but not released, i.e., the number of current users of the page

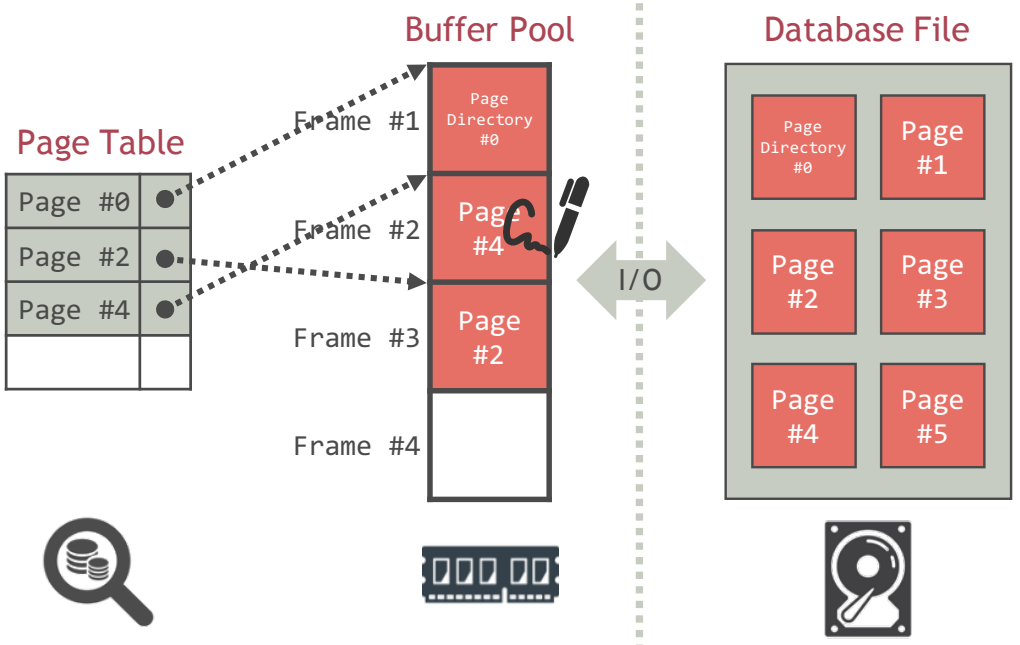





# Buffer Pool Internals: Frame's Meta-Data

The buffer manager maintains two variables for each frame

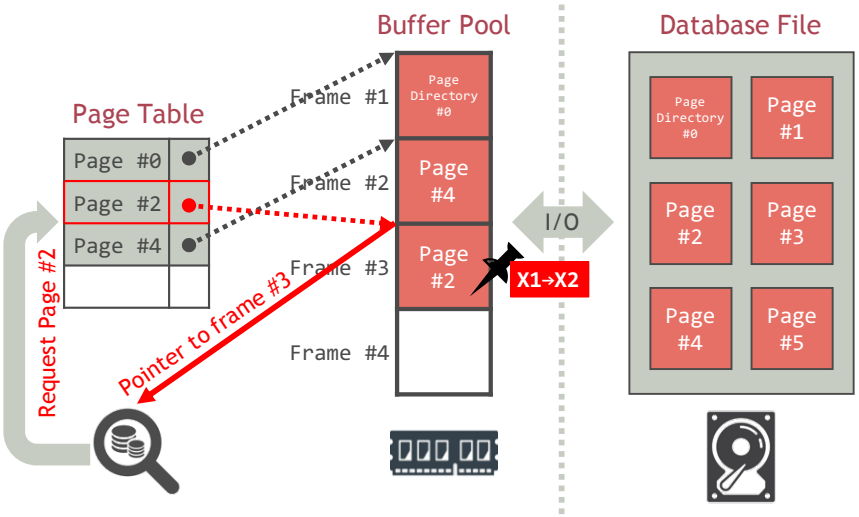
- `dirty`: the status whether the page in the frame has been modified since it was brought into the buffer pool



# Page Requests (请求页)

1. Check the page table to see if some frame contains the requested page $P$
2. If $P$ is in the buffer pool, pin page $P$, i.e., increment the `pin_count` of the frame containing $P$
3. Return the pointer of the frame containing $P$

Example: Request page #2





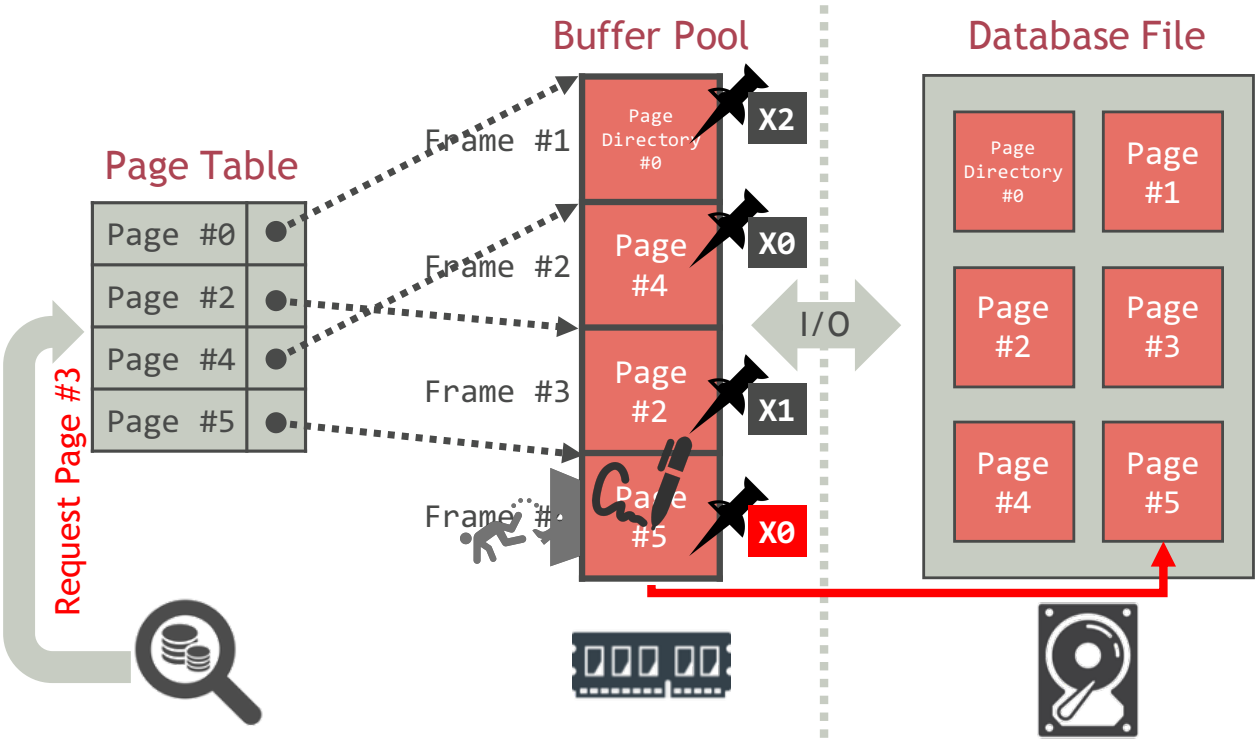

# Page Requests (Cont'd)

If the requested page is not in the buffer pool,

1. (选替换页) Chooses a frame with `pin_count` = 0 for replacement, using the replacement policy, and increments its `pin_count`
2. (写回脏页) If the `dirty` bit for the replacement frame is on, writes the page it contains to disk
3. (读请求页) Reads the requested page into the replacement frame

# Page Requests (Cont'd)

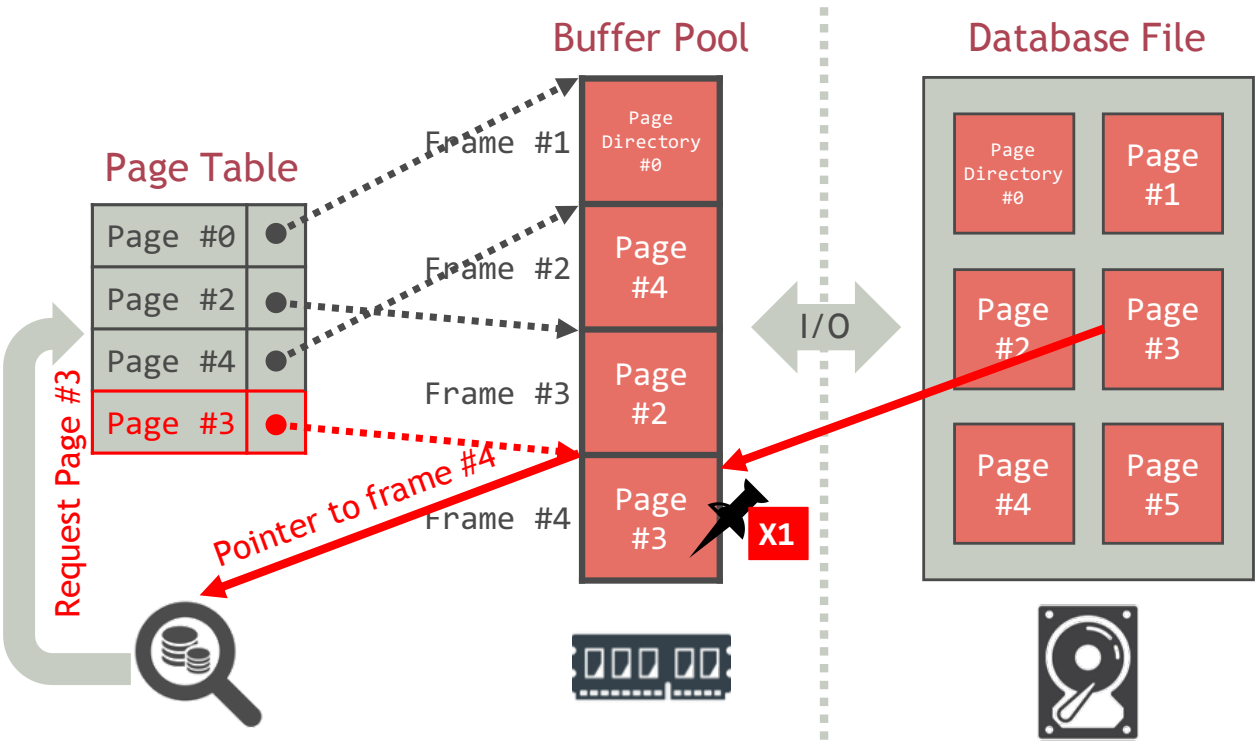
Example: Request page #3






# Page Requests (Cont'd)

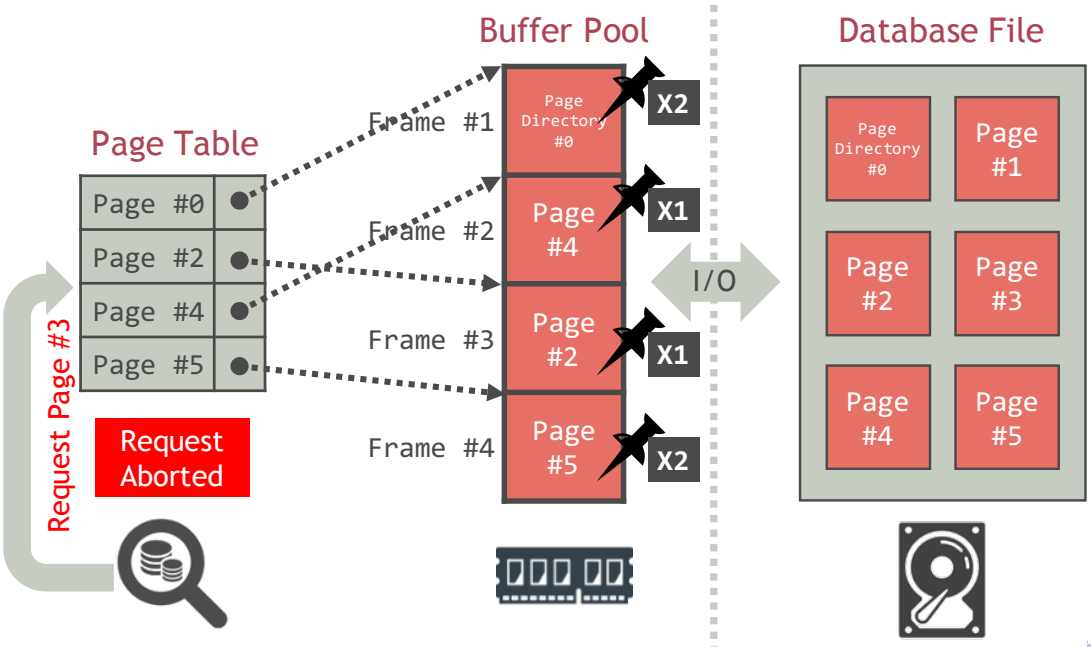
Example: Request page #3



# Page Requests (Cont'd)

If no page in the buffer pool has `pin_count` = 0, the buffer manager must wait until some page is released before responding to the page request

In practice, the transaction requesting the page may simply be aborted

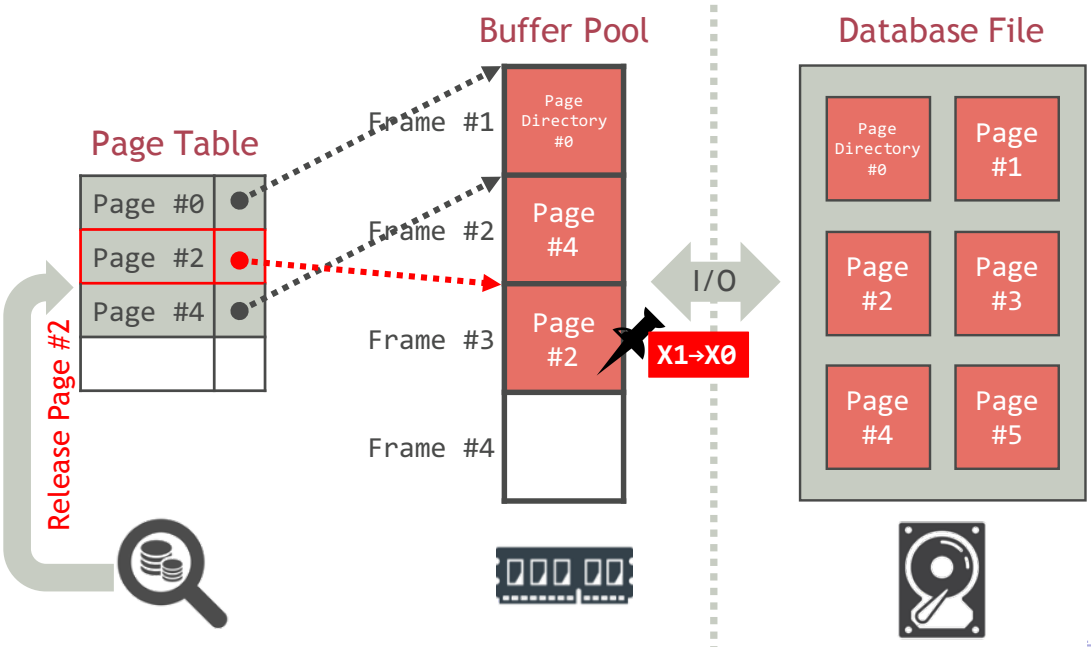





# Page Releases (页释放)

Unpin the released page, i.e., decrement the `pin_count` of the frame containing the page

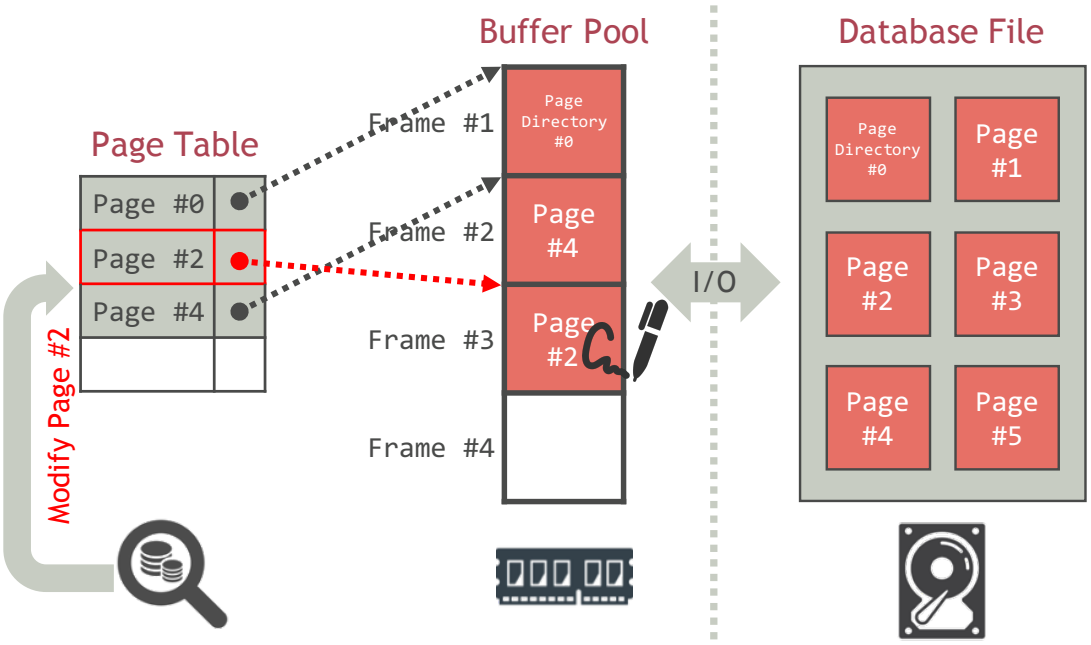
Example: Release page #2



# Page Modifications (页修改)

Set the `dirty` bit for the frame containing the modified page

Example: Modify page #2

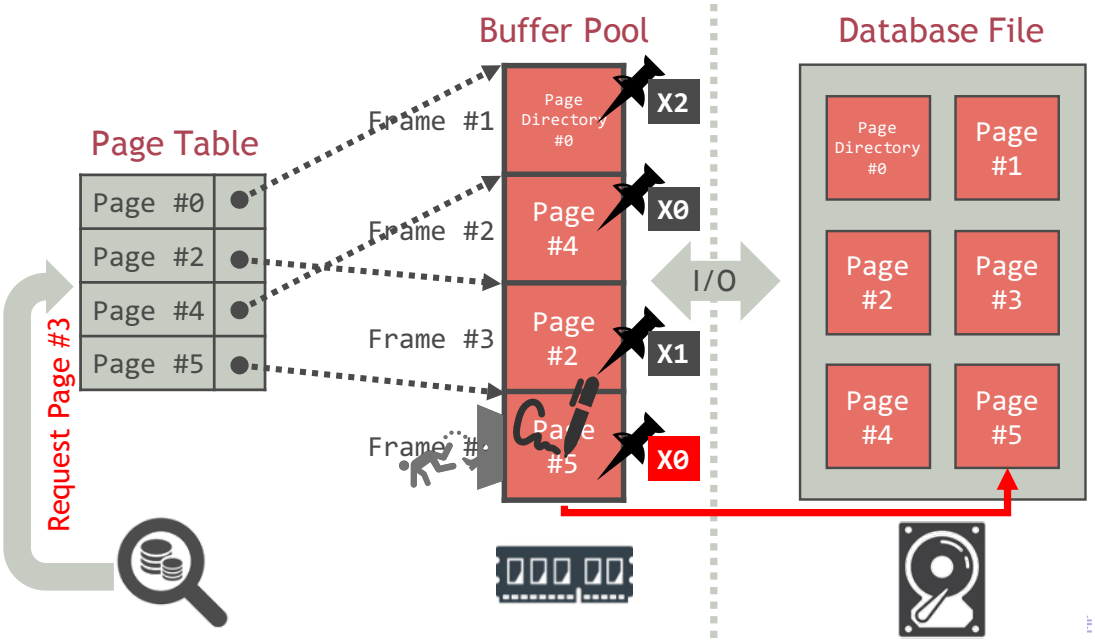





# Page Replacement Policies (页替换策略)

When the buffer manager needs to free up a frame to make room for a new page, it must decide which page to evict from the buffer pool

- The page replacement policy can affect the time taken for database operations considerably



# Least Recently Used (LRU) Relpacement Policy

Maintain a timestamp of when each page was last accessed

When the DBMS needs to evict a page, select the one with the oldest timestamp

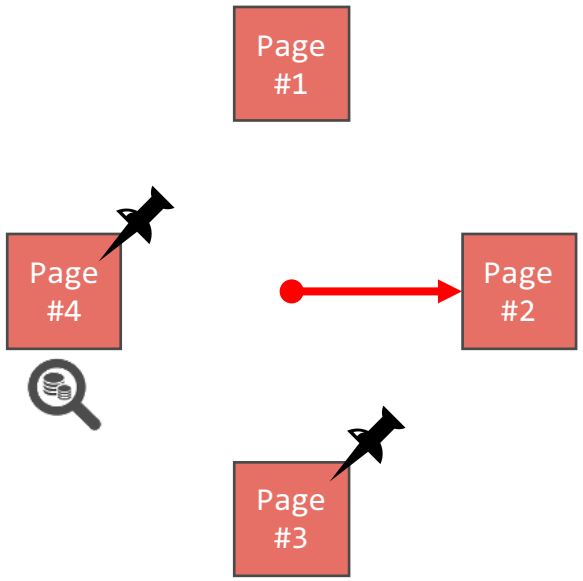
- Keep the pages in sorted order to reduce the search time on eviction



# Clock Replacement Policy (时钟替换策略)

The clock replacement policy is an approximation of LRU without needing a separate timestamp per page

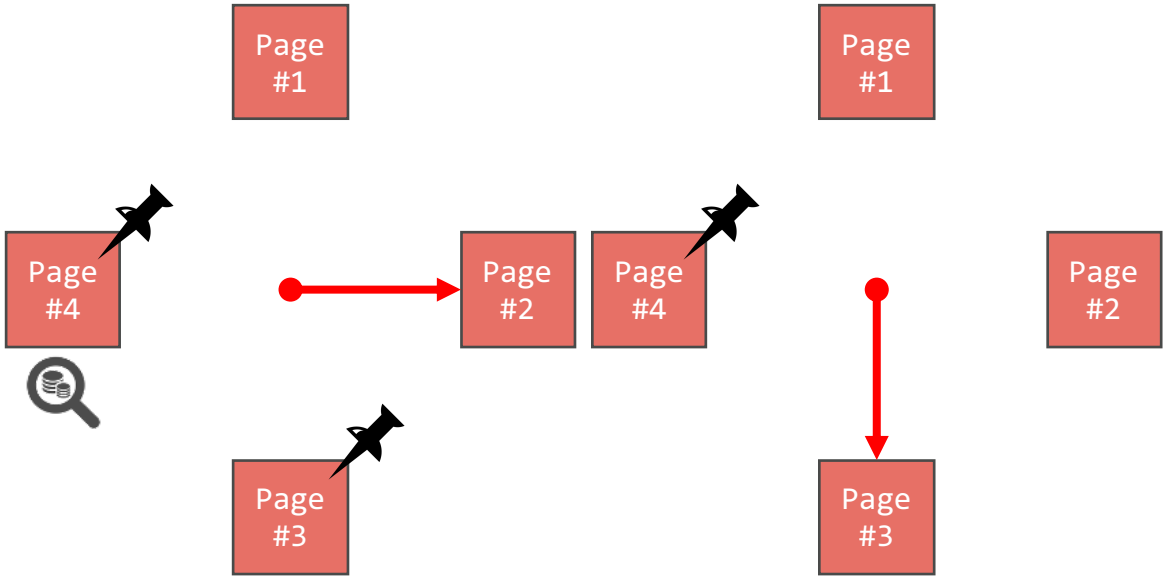
- Each frame has a reference bit, which is either 0 or 1
- When a page is read into a frame or when the page in a frame is accessed, its reference bit is set to 1



# Clock Replacement Policy (Cont'd)

The frames are organized in a circular buffer with a "clock hand"

- The clock hand rotates clockwise
- Upon sweeping, check if a page's reference bit is set to 1
- If yes, set the reference bit to 0

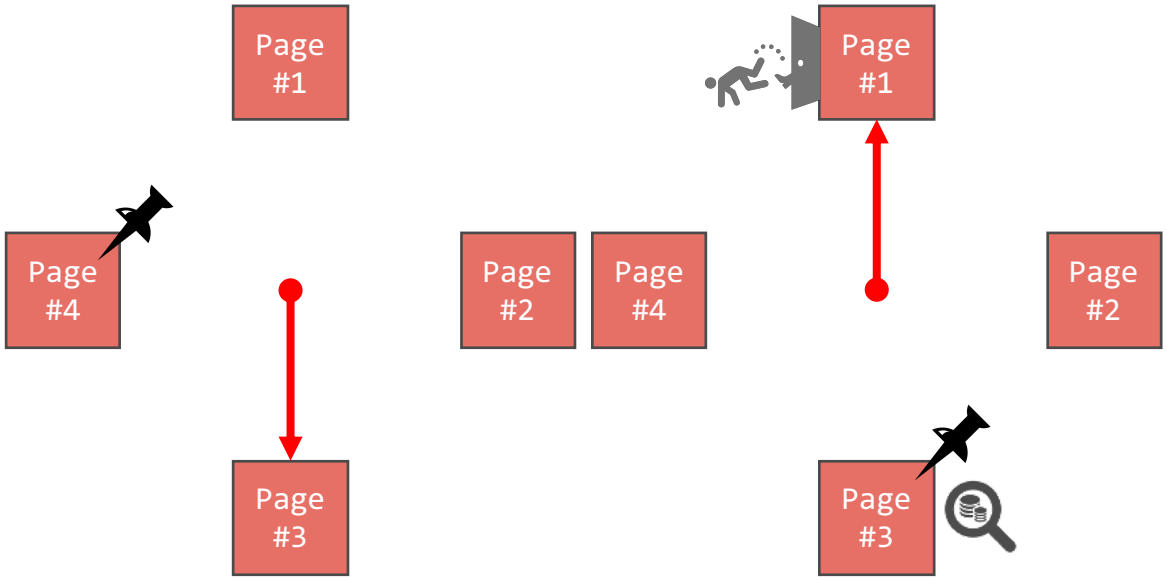





# Clock Replacement Policy (Cont'd)

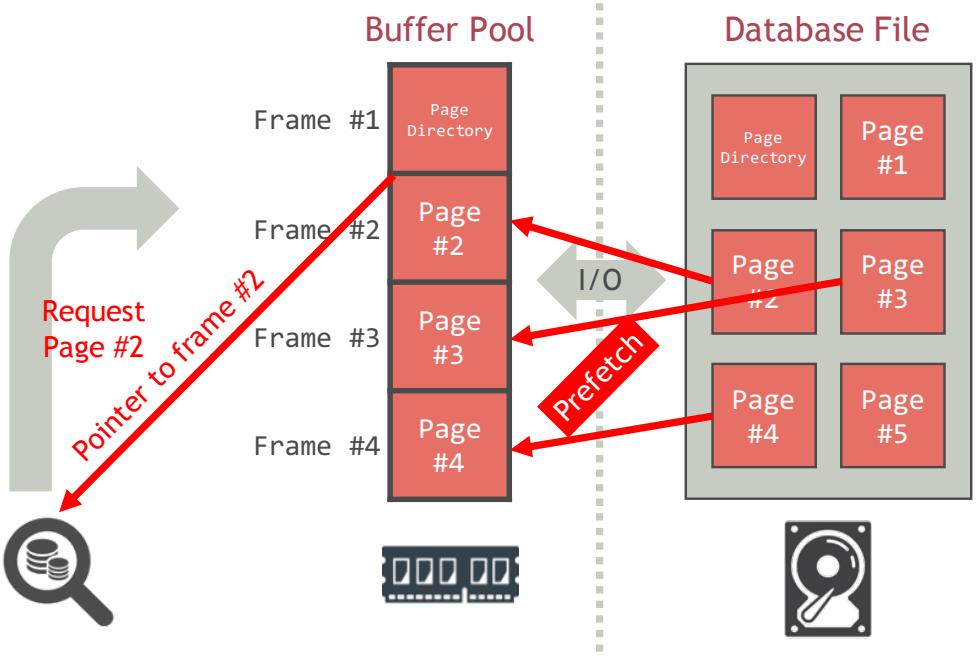
When the buffer manager needs a frame for a new page,

- Look for the first frame with reference bit 0
- Evict the page in the frame



# Prefetching (预取)

To handle page requests more efficiently, the buffer manager often prefetch pages that are going to be requested in near future



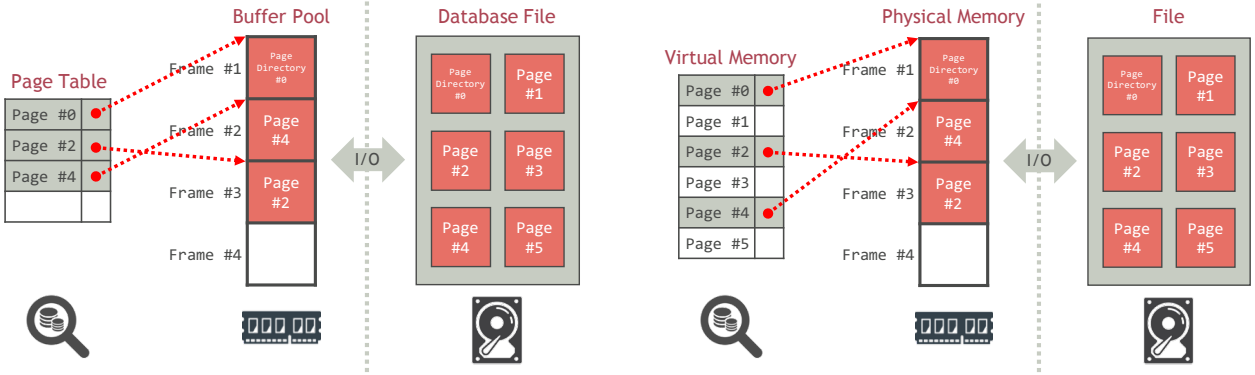



# Buffer Pool VS Virtual Memory

## Similarities

- Similar goal: Both provide access to more data that cannot fit in main memory
- Similar operations: Both bring in pages from disk to main memory as needed, replacing pages no longer needed in main memory

*Why can't we build a DBMS using the virtual memory capability of an OS?*



## Buffer Management VS Virtual Memory (Cont'd)

Why can't we build a DBMS using the virtual memory capability of an OS?

### Reason 1 (Page reference pattern prediction)

A DBMS can often predict the order in which pages will be accessed, or page reference patterns, much more accurately than is typical in an OS environment

### Implication

- Page Replacement: The ability to predict page reference patterns allows for a better choice of pages to replace
- Page Prefetching: Being able to predict page reference patterns enables the buffer manager to anticipate the next several page requests and fetch the corresponding pages into memory before the pages are requested



## Buffer Management VS Virtual Memory (Cont'd)

Why can't we build a DBMS using the virtual memory capability of an OS?

### Reason 2 (Forcing page write)

- A DBMS requires the ability to explicitly force a page to disk, that is, to ensure that the copy of the page on disk is updated with the copy in memory
- The OS command to write a page to disk may be implemented by essentially recording the write request and deferring the actual modification of the disk copy. If the system crashes in the interim, the effects can be catastrophic for a DBMS

### Implication

A DBMS must be able to ensure that certain pages in the buffer pool are written to disk before certain other pages to implement the Write Ahead Logging (WAL) protocol for crash recovery

# Summary

1. Storage Media
2. Representation of Databases on Disks
   - Value Representation
   - Tuple Layout
   - Page Layout
     - Tuple-Oriented Page Layout
     - Log-Structured Page Layout
   - File Organization
3. System Catalogs
4. Buffer Management