Chapter 7: Storage Management

Part 1: Storage Structures

Zhaonian Zou

Massive Data Computing Research Center
School of Computer Science and Technology
Harbin Institute of Technology, China
Email: znzou@hit.edu.cn

Spring 2019

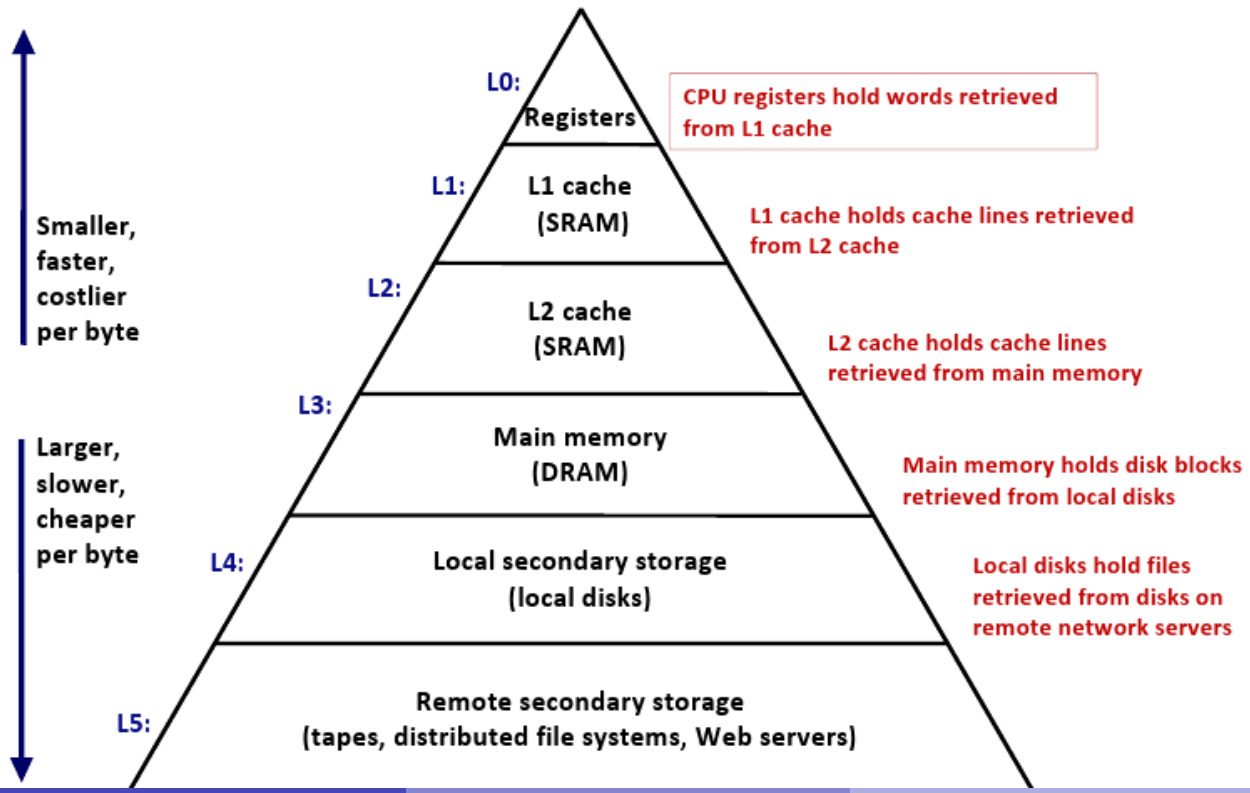
Outline[1]

[1] Updated on March 31, 2019

7.1 Storage Media

The Memory Hierarchy

Memory in a computer system is arranged in a hierarchy



Categories of Storage Media

- Primary storage (主存储器): storage media that can be operated on directly by the CPU using `load/store`
  - Byte-addressable (按字节寻址)
  - Registers (寄存器)
  - Cache (高速缓存)
  - Main memory (内存)
- Secondary storage (二级存储器): data in secondary storage cannot be processed directly by the CPU; it must first be copied into primary storage using `read/write`
  - Block-addressable (按块寻址) & online (联机)
  - Magnetic disks (磁盘)
  - Solid state drives (SSD, 固态硬盘)
- Tertiary storage (三级存储器): data in tertiary storage must first be copied into secondary storage
  - Block-addressable & offline (脱机)
  - Magnetic tape (磁带)
  - Optical disks (光盘)

Transfer of Data Between Levels

- Cache
  - ↕ Unit: cache lines (缓存行), 32B
- Main memory
  - ↕ Unit: blocks (块)/pages (页), 4KB–64KB
- Secondary storage
  - ↕ Unit: blocks (or pages), 4KB–64KB
- Tertiary storage

Data locality (数据局部性): data in the same unit will be needed at about the same time

Virtual Memory (虚拟存储器)

- Virtual memory is managed by the operating system
- Virtual memory is not a level of the memory hierarchy

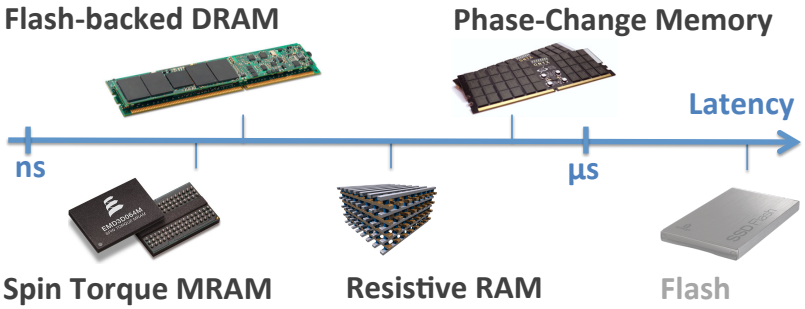
Volatile Storage and Non-volatile Storage

- Volatile storage (易失存储器): data in volatile storage media 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 media is retained when the computer is restarted (after a shutdown or a crash)
  - Secondary storage
  - Tertiary storage
- Non-volatile primary storage is emerging!

Non-volatile Byte-addressable Main Memory (NVM)

- Non-volatile byte-addressable main memory (NVM) is an emerging memory technology
- NVM is also known as persistent memory (持久内存)

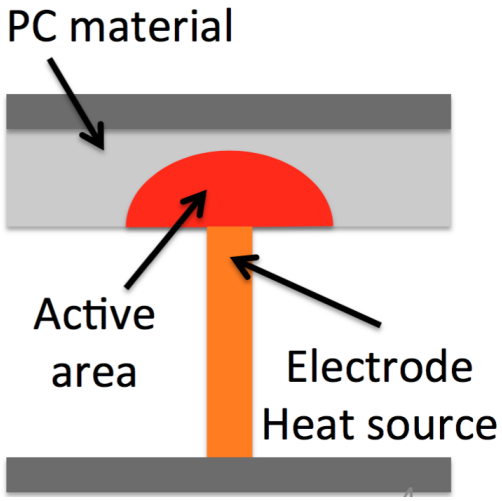
NVM Technologies



- Persistent
- Short access time
- Byte Addressable

Phase-Change Memory (PCM, 相变存储器)

- Store data within phase-change material (相变材料)
  - Amorphous phase (非晶态): high resistivity (0)
  - Crystalline phase (晶态): low resistivity (1)
- Set phase via current pulse
  - Fast cooling → Amorphous
  - Slow cooling → Crystalline



Characteristics of NVM

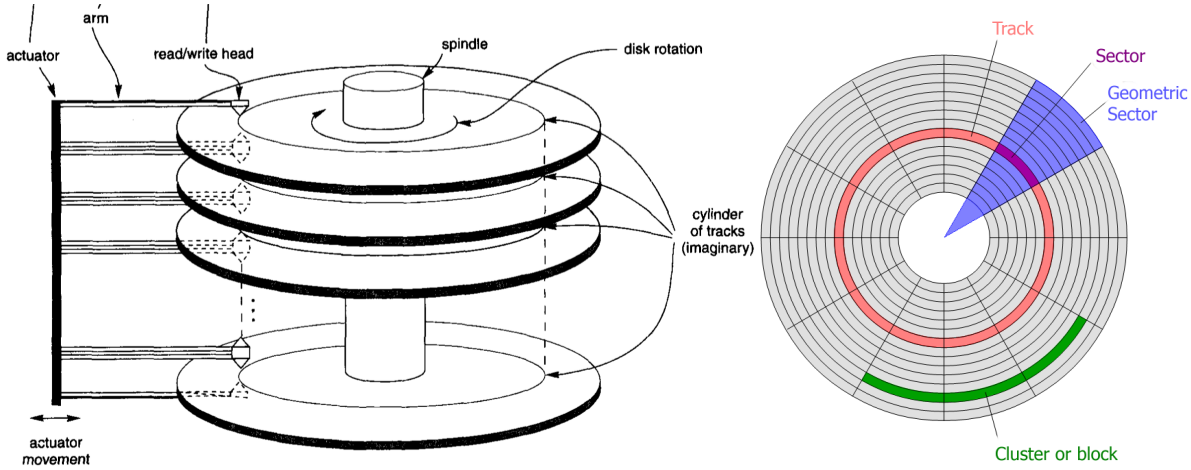
Characteristics	DRAM	PCM	Memristor	MRAM	SSD	HDD
Volatility	Yes	No	No	No	No	No
Addressability	Byte	Byte	Byte	Byte	Block	Block
Volume	GB	TB	TB	TB	TB	TB
Read latency	60ns	50ns	100ns	20ns	25μs	10ms
Write latency	60ns	150ns	100ns	20ns	300μs	10ms
Energy per bit	2pJ	2pJ	100pJ	0.02pJ	10nJ	0.1J
Endurance	10^{16}	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
- Intel, Persistent Memory Developing Toolkit (PMDK) (`http://pmem.io`)

Magnetic Disks (磁盘)

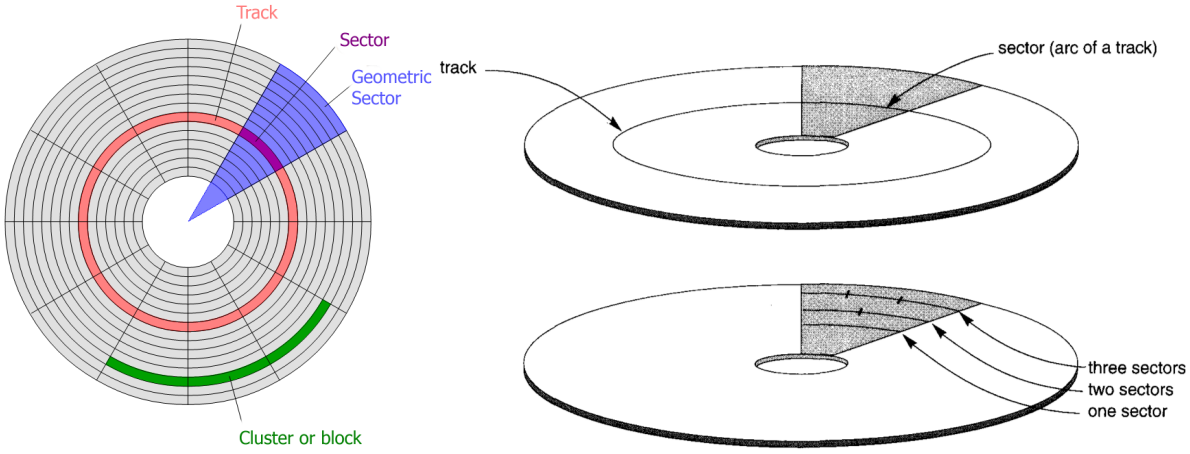
A magnetic disk is composed by two components:

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



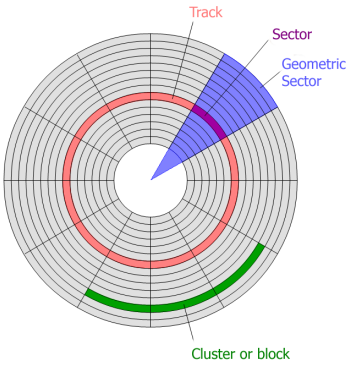
Sectors (扇区)

- Because a track usually contains a large amount of information, it is divided into smaller sectors
- The division of a track into sectors is hard-coded on the disk surface and cannot be changed
- Different sector organizations on disk
  - Sectors subtending a fixed angle
  - Sectors maintaining a uniform recording density



Disk Blocks (Pages)

- The division of a track into equal-sized disk blocks (or pages) is set by the OS during disk formatting
  - The size of a disk block can be set as a multiple of the sector size during disk formatting
  - Block size is fixed during disk formatting and cannot be changed dynamically
  - Typical disk block sizes range from 512 to 4096 bytes
- Logical block address (LBA): The LBA of a disk block is a number between 0 and $n-1$ (assuming the total capacity of the disk is n blocks)



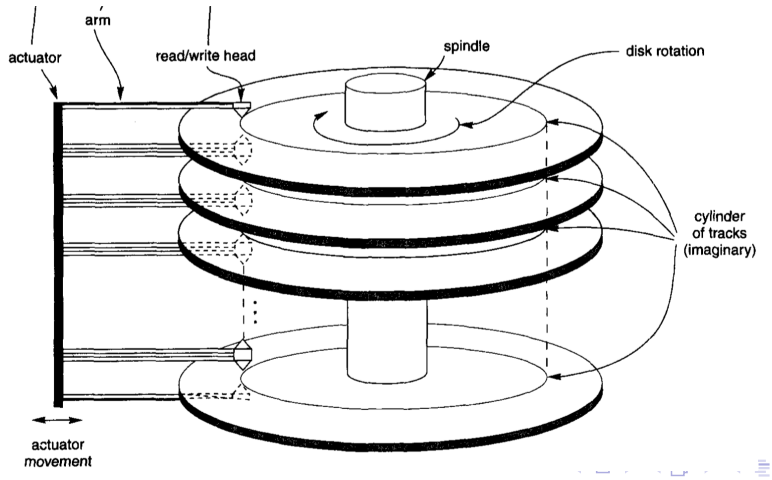
Disk Controller (磁盘控制器)

A disk controller interfaces a disk drive to the computer

- Data on disks must be read to the main memory for the DBMS to operate on it
- The unit for data transfer between disk and main memory is a block (even if a single item on a block is needed)

How does a disk controller work?

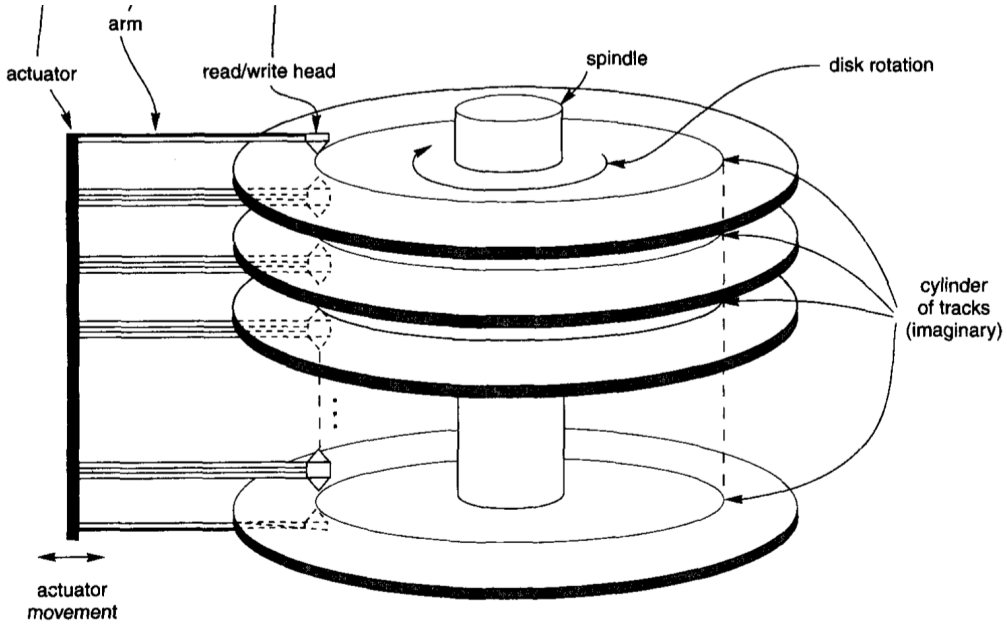
1. 寻道: Select the desired track
2. 旋转: Select the desired block on the desired track
3. 传输: Transfer data between the desired block and the main memory
4. 预取: Transfer several consecutive blocks on the same track or cylinder (This eliminates the seek time and rotational delay for all but the first block and can result in a substantial saving of time when numerous contiguous blocks are transferred)



Lentency of Disks

Latency of disks = seek time + rotational delay + transfer time ≈ 10 ms

1. Seek time (寻道时间) ≈ 0–10 ms
2. Rotational latency (旋转延迟) ≈ 0–10 ms
3. Block transfer time (传输时间) < 1 ms



Disk Scheduling (磁盘调度)

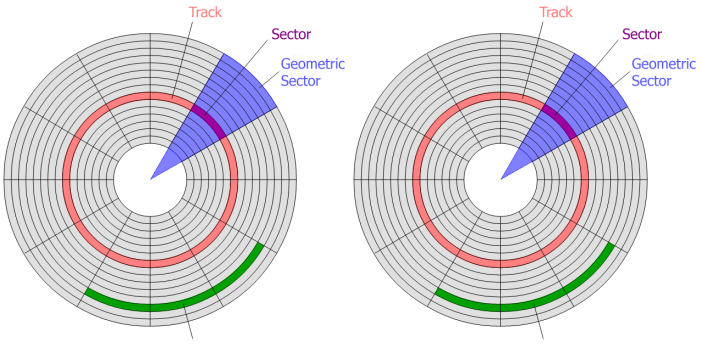
Disk scheduling is determining the order for processing a set of block read/write requests to minimize the time latency

The Elevator Algorithm

- Let $dir \in \{IN, OUT\}$ be the current moving direction of the disk heads
- (前方有请求, 继续前进) If there are pending block read/write requests in the moving direction *dir*, move the disk heads ahead to the nearest track on which read/write is requested
- (前方无请求, 立即掉头) If there is no pending block read/write request ahead in the moving direction *dir*, change the direction *dir*

Accelerating Access to Magnetic Disks

- Exploiting the advantages of sequential access (顺序访问) to disk blocks: Organize data records strategically on disks because of the geometry and mechanics of disks: If two records are frequently used together, place them close together.
  - In decreasing order of closeness: on the same block $\prec$ on the same track $\prec$ on the same cylinder $\prec$ on an adjacent cylinder
- Utilizing Parallelism (并行化)
  - Using multiple disks
  - Mirroring disks (磁盘镜像)
- Exploiting data locality (数据局部性)
  - Prefetching pages (预取)



The I/O Computation Model

- Reading or writing a disk block is called an I/O (for input/output) operation (输入/输出操作)
- The time for moving blocks to or from disk usually dominates the time taken for database operations
- In the I/O model, the number of disk I/O's performed by a database operation is a good approximation of the time needed by the operation

7.2 Storage Failures

Types of Storage Failures

- Intermittent failure (间发故障): an attempt to read or write a sector is unsuccessful, but can be finally successful with repeated tries
- Media decay (介质损坏): a bit or more are permanently corrupted
- Write failure (写故障): we neither write successfully nor can we retrieve the previously written sector possibly due to power outage during the writing
- Disk crash (磁盘崩溃): the entire disk is permanently unreadable

Checksums (校验和)

- Checksum (校验和): some additional bits contained in a sector, which are set depending on the data stored in the sector
- Parity checksum (奇偶校验和): the checksum is set based on the parity of all the bits in the sector
  - Example: 01101000 $\rightarrow$ 1
  - Property: the number of 1's among a collection of bits and their parity bits always even
- We can increase the chances of detecting errors if we keep several parity bits. For example, 8 parity bits, the ith parity bit is the checksum for the ith bit of every byte

Byte 1	11110000
Byte 2	10101010
Byte 3	11001100
Checksum	10010110

Redundant Array of Independent Disks (RAID)

Redundant Array of Independent Disks (RAID, 独立磁盘冗余阵列)

- Data disks (数据磁盘)
- Redundant disks (冗余磁盘)

RAID 1

- Mirroring disks is often referred to as RAID 1
- 1 data disk $\longleftrightarrow$ 1 redundant disk (mirror)

	Block 1	Block 2	...	Block n
Data disk	b_1	b_2	...	b_n
Redundant disk (mirror)	b_1	b_2	...	b_n

RAID 4

- Multiple data disks $\longleftrightarrow$ 1 redundant disk
- All the blocks on all the disks have the same number of bits
- The ith block of the redundant disk consists of parity checks for the ith blocks of all the data disks

	Block i
Data disk 1	11110000
Data disk 2	10101010
Data disk 3	00111000
Redundant disk	01100010

RAID 4 (Cont'd)

- Write: update the redundant disk after writing the data disk

	Block i			Block i
Data disk 1	11110000		Data disk 1	11110000
Data disk 2	10101010	$\Rightarrow$	Data disk 2	11001100
Data disk 3	00111000		Data disk 3	00111000
Redundant disk	01100010		Redundant disk	00000100

- Recovery: recover the data disk as the parity check of all the other disks

	Block i			Block i
Data disk 1	11110000		Data disk 1	11110000
Data disk 2	????????	$\Rightarrow$	Data disk 2	10101010
Data disk 3	00111000		Data disk 3	00111000
Redundant disk	01100010		Redundant disk	01100010

- RAID 4 preserves data unless there are two almost simultaneous disk crashes

RAID 5

- In RAID 4, each write to a data disk results in a write to the redundant disk (unbalanced)

	Block 0	Block 1	Block 2	Block 3	Block 4	...
Disk 0	D	D	D	D	D	...
Disk 1	D	D	D	D	D	...
Disk 2	R	R	R	R	R	...

- In RAID 5, each disk is treated as the redundant disk for some of the blocks
- If there are n disks numbered 0 through n, we could treat the (i mod n)-th block of disk i as redundant for all the (i mod n)-th blocks of the other disks

	Block 0	Block 1	Block 2	Block 3	Block 4	...
Disk 0	R	D	D	R	D	...
Disk 1	D	R	D	D	R	...
Disk 2	D	D	R	D	D	...

RAID 6

- Multiple data disks $\longleftrightarrow$ multiple redundant disks
- RAID 6 can deal with simultaneous crashes of multiple disks
- RAID 6 is based on the simplest error-correcting code—Hamming code (汉明码)

Example (RAID 6)

- 7 disks
- Disk 1–4 are data disks
- Disk 5–7 are redundant disks

Data disks				Redundant disks		
disk 1	disk 2	disk 3	disk 4	disk 5	disk 6	disk 7
⋆	⋆	⋆		⋆		
⋆	⋆		⋆		⋆	
⋆		⋆	⋆			⋆

RAID 6 (Cont'd)

Example (RAID 6, Write)

After writing disk 2, write its redundant disks (disk 5 and disk 6)

Data disks				Redundant disks		
disk 1	disk 2	disk 3	disk 4	disk 5	disk 6	disk 7
⋆	⋆	⋆		⋆		
⋆	⋆		⋆		⋆	
⋆		⋆	⋆			⋆

Disk	Data
1	11110000
2	10101010
3	00111000
4	01000001
5	01100010
6	00011011
7	10001001

$\Rightarrow$

Disk	Data
1	11110000
2	00001111
3	00111000
4	01000001
5	11000111
6	10111110
7	10001001

RAID 6 (Cont'd)

Example (RAID 6, Recovery)

- Disks 2 and 5 crash simultaneously. Recover them as follows
  1. Recover disk 2 using disks 1, 4 and 6
  2. Recover disk 5 using disks 1, 2 and 3

Data disks				Redundant disks		
disk 1	disk 2	disk 3	disk 4	disk 5	disk 6	disk 7
⋆	⋆	⋆		⋆		
⋆	⋆		⋆		⋆	
⋆		⋆	⋆			⋆

Disk	Data
1	11110000
2	????????
3	00111000
4	01000001
5	????????
6	00011011
7	10001001

$\Rightarrow$

Disk	Data
1	11110000
2	10101010
3	00111000
4	01000001
5	????????
6	00011011
7	10001001

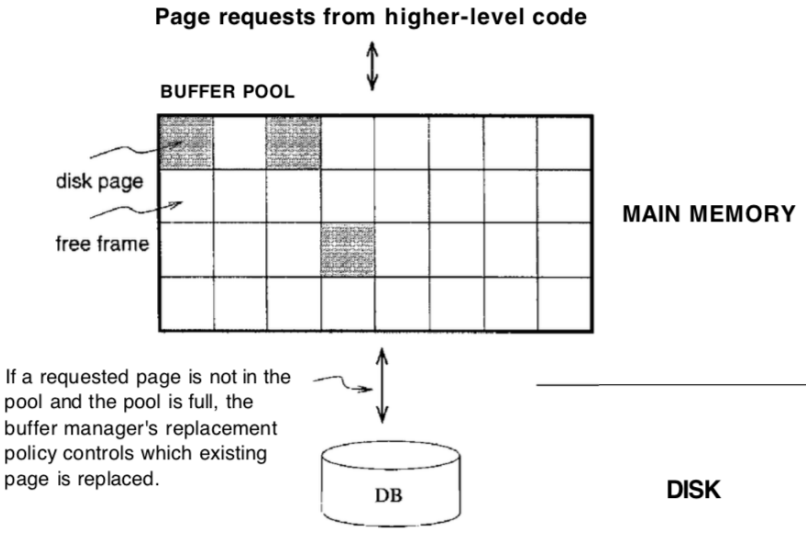
$\Rightarrow$

Disk	Data
1	11110000
2	10101010
3	00111000
4	01000001
5	01100010
6	00011011
7	10001001

7.3 Buffer Management

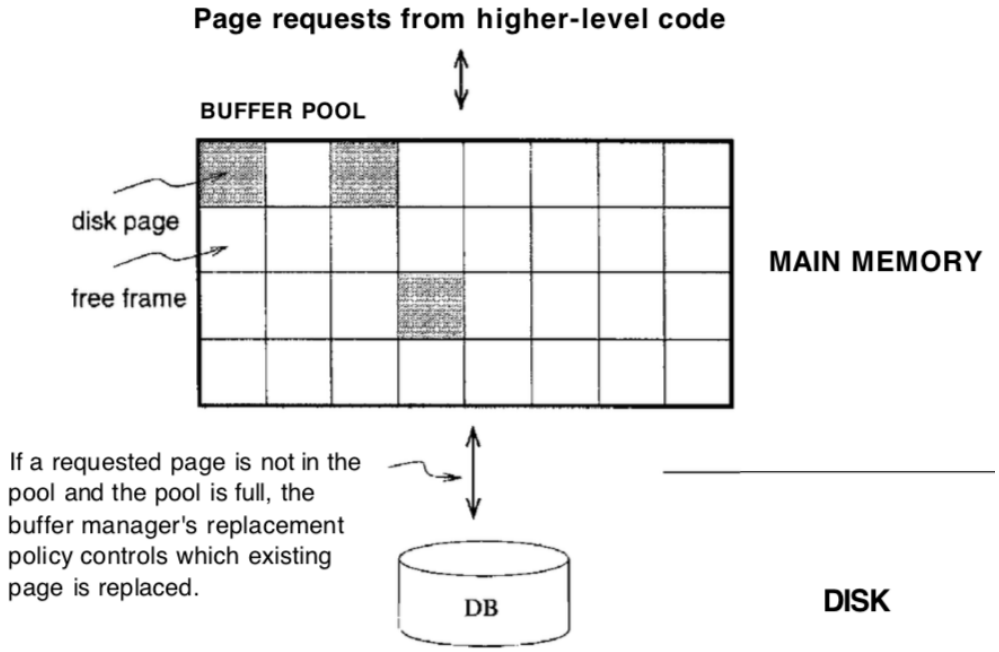
Buffer Manager (缓冲区管理器)

- All the data cannot be brought into main memory at one time
- The buffer manager in the DBMS is responsible for bringing pages into main memory as needed
- In the process, the buffer manager decides what existing page in main memory to replace to make space for the new page



Buffer Pool (缓冲池)

- The buffer manager partitions the available main memory into a collection of pages, which collectively called the buffer pool
- The pages in the buffer pool are called frames (页框)



Frames (页框)

The buffer manager maintains two variables for each frame in the buffer pool

- `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. The page is called *pinned* if `pin_count` > 0
- `dirty`: the status whether the page in the frame has been modified since it was brought into the buffer pool
- Initialization: When the buffer pool is intialized, the buffer manager sets `pin_count = 0` and `dirty = false` for every frame in the pool

Handling Page Requests (页请求)

When a page is requested by a transaction, the buffer manager works as follows:

1. Checks the buffer pool to see if some frame contains the requested page and, if so, pins the page, i.e., increments the `pin_count` of that frame. If the page is not in the pool, the buffer manager brings it in as follows:
   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
2. Returns the (main memory) address of the frame containing the requested page to the requestor

Handling Page Releases (页释放)

A transaction that requests a page must also release the page when it is no longer needed, by informing the buffer manager. When a page is released, the buffer manager works as follows:

1. Unpins the page, i.e., decrement the `pin_count` of the frame containing the page

Handling Page Modifications (页修改)

When a page is modified by a transaction, the buffer manager works as follows:

1. Sets the `dirty` bit for the frame containing the modified page

Page Replacement (页替换)

- If no page in the buffer pool has `pin_count` $= 0$ and a page that is not in the pool is requested, 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 in this situation.
- $\star$ Transaction management in DBMS complicate page replacement
  - Writing a dirty page modified by an *uncommitted* transaction to disk may avoid the *atomicity*
  - When two or more transactions modify the same page in the buffer at the same time, the data may become inconsistent

Page Replacement Policies (页替换策略)

The policy used to choose a page for replacement can affect the time taken for database operations considerably

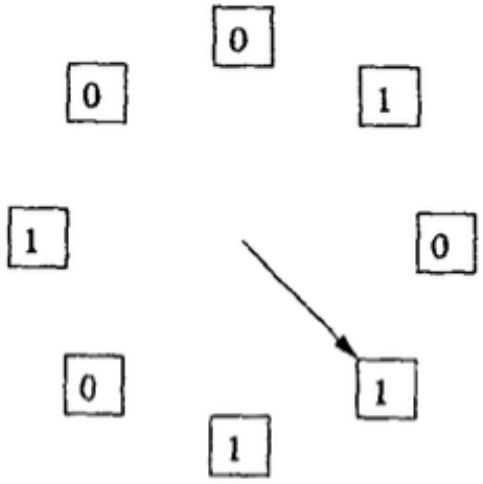
- Least Recently Used (LRU): The least recently used page with `pin_count` = 0 will be replaced
- Clock Replacement (A Variant of LRU): The behavior is similar to LRU, but the overhead is less
- First In First Out (FIFO)
- Most Recently Used (MRU)

Implementation of LRU

- Use a queue of pointers to frames with `pin_count` = 0 in the buffer pool
- The frames are ordered by the latest access time
- A frame is added to the end of the queue when it becomes a candidate for replacement (that is, when its `pin_count` goes to 0)
- The page in the frame at the head of the queue is chosen for replacement

Clock Replacement

- Each frame has a flag, which is either 0 or 1
- When a block is read into a frame or when the contents of a frame is accessed, its flag is set to 1
- The block in a frame flagged by 0 can be thrown out
- When the buffer manager needs a frame for a new block, it looks for the first 0 it can find, rotating clockwise. If it passes 1's, it sets them to 0



Buffer Management in DBMS v.s. Virtual Memory in OS

Similarities

- Similar goal: Both provide access to more data that cannot fit in main memory
- Similar idea: 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 in DBMS v.s. Virtual Memory in OS

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 in DBMS v.s. Virtual Memory in OS

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

7.4 Representation of Data

Representation of Data

- Data → files (文件)
- File → records (记录)
- Record → fields (字段)

Representation of Fields

Numbers

- `INTEGER`: a sequence of 2 or 4 bytes
- `FLOAT`: a sequence of 4 or 8 bytes

Fixed-Length Character Strings

- `CHAR(n)`: a sequence of n bytes
- If the length of a string is shorter than n, then the array is filled out with a special null character $\perp$
- Example: `CHAR(5)`, `'cat'` $\rightarrow$ `cat`$\perp\perp$

Representation of Fields (Cont'd)

Variable-Length Character Strings

- `VARCHAR(n)`: a sequence of $l + 1$ bytes, where l is the actual length of the string
- Representation method 1: length + string
- Example: `VARCHAR(5)`, `'cat'` $\rightarrow$ 3`cat`
- Representation method 2: null-terminated string
- Example: `VARCHAR(5)`, `'cat'` $\rightarrow$ `cat`$\perp$

Date & time

- `INTEGER`

Bit strings

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

Representation of Fields (Cont'd)

Enumerated Types

- A enumerated type with n items = `BIT`($\lceil \log_2 n \rceil$)
- Example: {`RED, GREEN, BLUE, YELLOW`}, `RED` $\rightarrow$ `00`, `GREEN` $\rightarrow$ `01`, `BLUE` $\rightarrow$ `10`, `YELLOW` $\rightarrow$ `11`

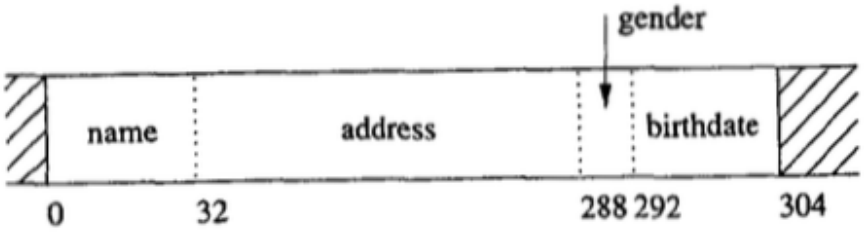
BLOBs (Binary Large Objects)

- BLOBs represents images, digitized video or audio streams, or free text
- A BLOB data item is typically stored separately from its record in a pool of disk blocks, and a pointer to the BLOB is included in the record

Format of Fixed-Length Records

- Concatenate all fields to form a record
- The record starts at a byte within its block that is a multiple of 4 (or 8 if the machine has a 64-bit processor)
- All fields within the record start at a byte that is offset from the beginning of the record by a multiple of 4 (or 8 if the machine has a 64-bit processor)

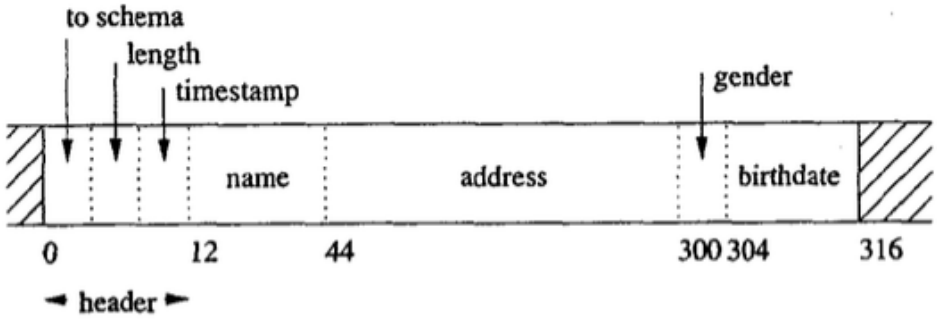
```
CREATE TABLE MovieStar(
name CHAR(30) PRIMARY KEY,
address VARCHAR(255),
gender CHAR(1),
birthdate DATE);
```



Record Header

- A pointer to a place where the DBMS stores the schema for this type of record
- The length of the record
- Timestamps indicating the time the record was last modified, or last read

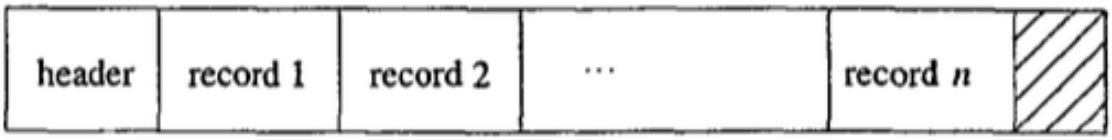
```
CREATE TABLE MovieStar(
name CHAR(30) PRIMARY KEY,
address VARCHAR(255),
gender CHAR(1),
birthdate DATE);
```



Packing Fixed-Length Records into Blocks

There is an optional block header

- Links to one or more other blocks that are part of a network of blocks
- Information about which relation the tuples of this block belong to
- A "directory" giving the offset of each record in the block
- Timestamp(s) indicating the time of the block's last modification and/or access



Record Address

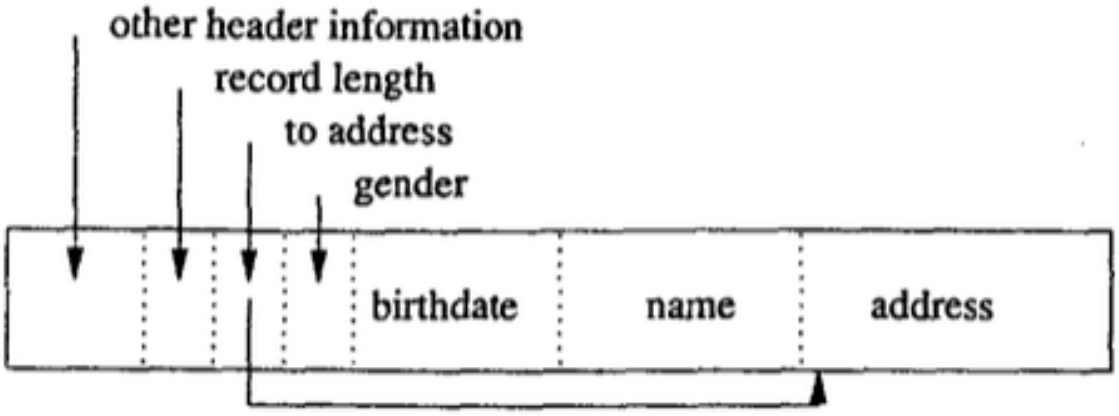
We can think of a page as a collection of slots (like frames in the buffer pool), each of which contains a record

- Method 1: A record is identified by using the pair (*page_id*, *slot_number*)
- Method 2: A record is assigned a unique integer as its record id (rid) and maintain a table that lists the page and slot of the corresponding record for each rid

rid	page id	slot number

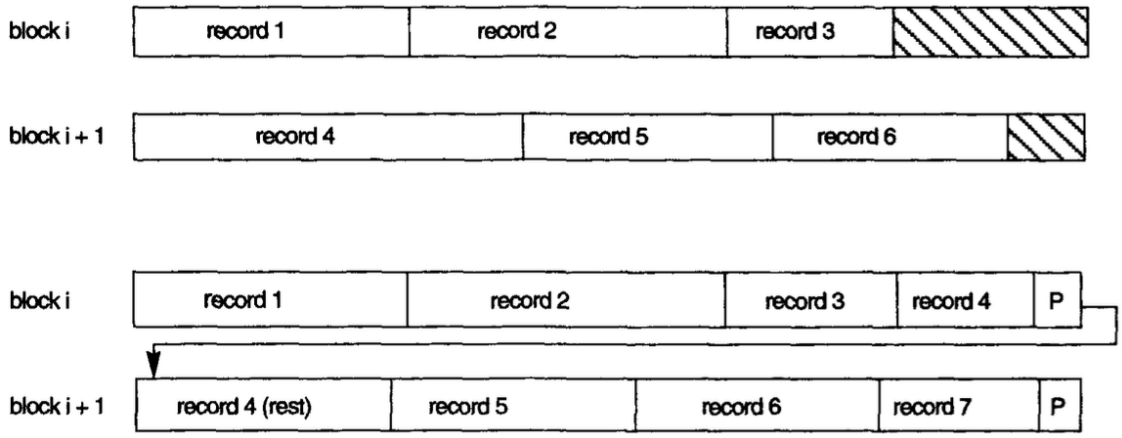
Format of Variable-Length Records

- Put all fixed-length fields ahead of the variable-length fields
- Put pointers to the beginnings of all the variable-length fields



Packing Variable-Length Records into Blocks

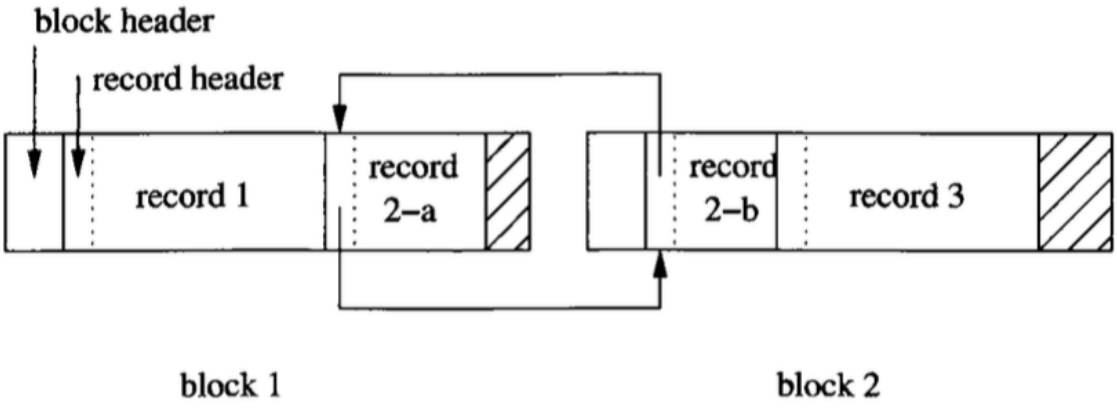
- Unspanned organization (不跨块组织法): records are not allowed to cross block boundaries
- Spanned organization (跨块组织法): records can span more than one block



Spanned Record Organization

The portion of a record in a block is called a record fragment

- Each record header must contain a bit telling whether or not it is a fragment
- If it is a fragment, then it needs bits telling whether it is the first or the last fragment for its record
- If there is a next and/or previous fragment for the same record, then the fragment needs pointers to these other fragments



Modifying a Variable-Length Record

- Modifying a field may cause it to grow, which requires us to shift all subsequent fields to make space for the modification
- A modified record may no longer fit into the space remaining on its page. If so, it may have to be moved to another page.
  - If a record is referred to by a pair (*page_id*, *offset*), the pointer to the modified record is no longer valid. We may have to leave a forwarding address on this page to identify the new location of the record.
  - If a record is identified by a unique record id, we must change the mapping from the record id to the address of the record

File Organizations

- Files of unordered records (heap files)
  - Records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file
  - Heap files are often used with additional access paths, such as the secondary indexes
- Files of ordered records (sorted files)
  - Records are ordered according to the ordering key
  - Ordered files are rarely used in database applications unless an additional access path, called a primary index, is used

Summary

1. Storage Media
   - The memory hierarchy
   - Primary storage, secondary storage, tertiary storage
   - Volatile storage, non-volatile storage
   - Magnetic disks: sectors, blocks (pages), tracks, cylinders, disk controller, disk scheduling
2. Storage Failures
   - Checksums: parity checksums
   - RAID: RAID 1, RAID 4, RAID 5, RAID 6
3. Buffer Management
   - Buffer manager: buffer pool, frames
   - Page requests, page release, page modification
   - Page replacement policies: LRU, Clock, FIFO, MRU
4. Representation of Data
   - Representation of fields
   - Record organization
   - Block organization
   - File organization