数据库系统原理

Database System Principle

邵蓥侠

Email：shaoyx@bupt.edu.cn

北京邮电大学计算机学院

计算机应用技术中心

PART 5

DATA STORAGE AND QUERY

Introduction to Chapter12 and Chapter13

- In view of physical DB, two issues should be addressed
  - data organization in DBS, i.e. physical storage structure of data — Chapter12
  - data access in DBS — Chapter13

- 目标

 1. 开发数据库应用系统时，根据DBMS提供的机制，选择合适的数据库物理结构

2. 在数据库表上合理设置索引，提高数据查询速度

Chapter 13

Storage and File Structure

Main Contents in This Chapter

- File organization (at physical level, § 13.2)
- Organization of records in files (at logical level, § 13.3),

  i.e. file structures
- Data-dictionary Storage (§ 13.4)
- Data Buffer (§ 13.5)

文件逻辑/物理组织 in Operating System Concepts

- 逻辑结构：

  流式文件, 基于记录文件

- 物理结构：以block为单位，进行存储

  contiguous, linked, indexed

13.2 File Organization

- The database is stored as a collection of **DB *files***.
- Each file is a sequence of *records.* A record is a sequence of fields.
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

  This case is easiest to implement; will consider variable length records later.

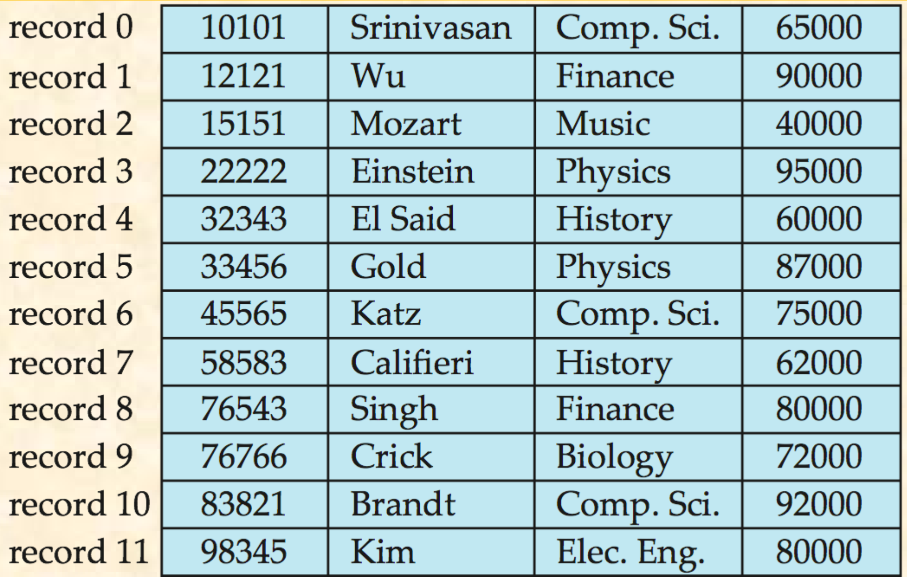
File Organization (conmt.)

- Each file is a sequence of ***records***, and *a* ***relational table*** is a set of ***tuples***
- /*以元组为单位的DB逻辑文件在以*记录*为单位的OS物理文件中如何实现??
- How to represent the ***tuple*** as the record in files?
  - fixed-length records
  - variable-length records

13.2.1 Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte *n* ∗ (*i* – 1), where *n* is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

instructor(ID, name, dept, salary)



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

Fixed-Length Records

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

all records are stored in a contiguous space

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

Deleting record 3 and compacting

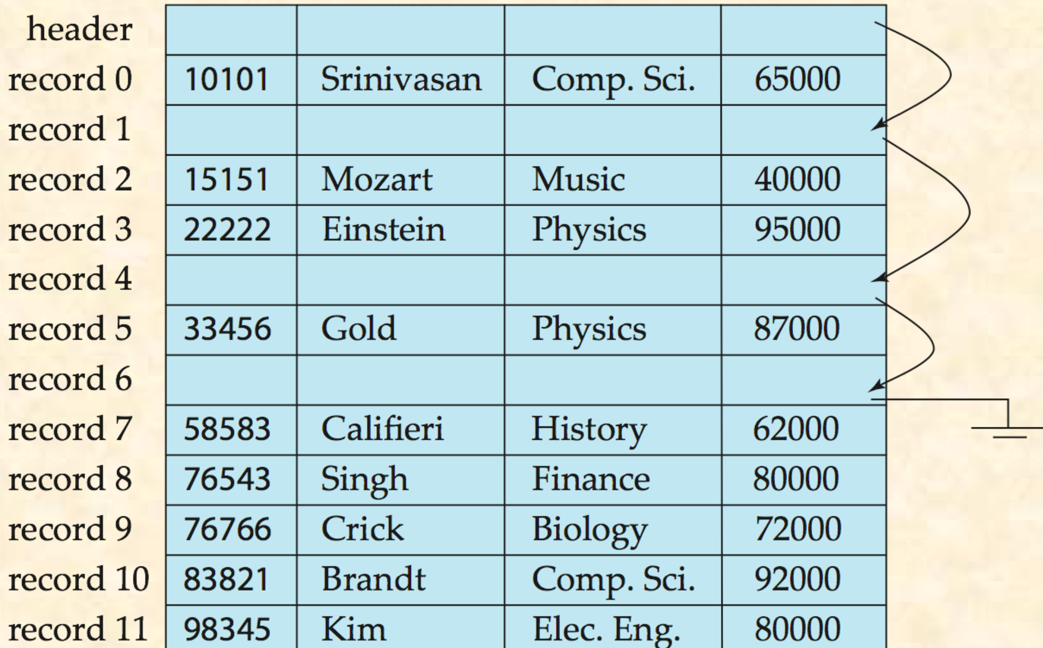
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766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

Deleting record 3 and moving last record

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

Free Lists

- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they “point” to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)



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

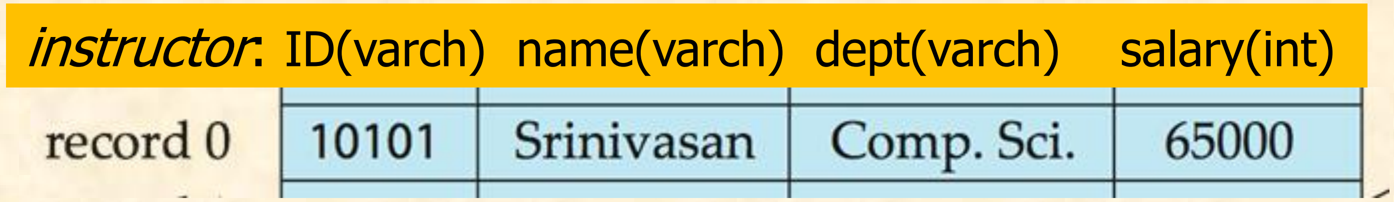
13.2.2 Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file, e.g. Multitable Clustering File Organization in 13.3.3
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields, e.g. multi-set, multi-value attribute (used in some older data models).
- E.g. **type** *account-list* = **record**

```
        branch-name: char(22)
        account-info: array[1.. ∞] of
              record
                  account-number : char(10)
                  balance :   real
              end
    end
```

Variable-Length Records

- Attributes in the record are stored in order
  - fixed length attributes at first, and then the variable-length attributes
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



instructor:	ID(varch)	name(varch)	dept(varch)	salary(int)
record 0	10101	Srinivasan	Comp. Sci.	65000

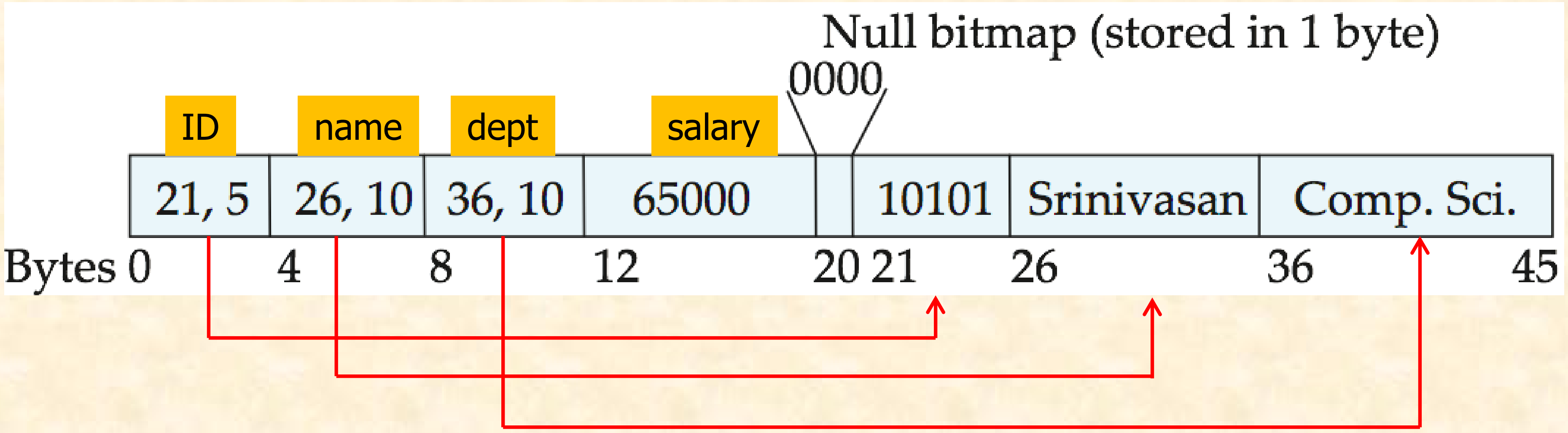


Fig. record 0 is stored in contiguous 46 bytes (0-45)

- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap

- **The slotted page** (e.g. SQL Server2005) structure is often used to organize variable-length records

Variable-Length Records: Slotted Page Structure

- The storage space is divided into fixed-sized *slotted pages*, and records are allocated to slotted pages, e.g. MS SQL Server

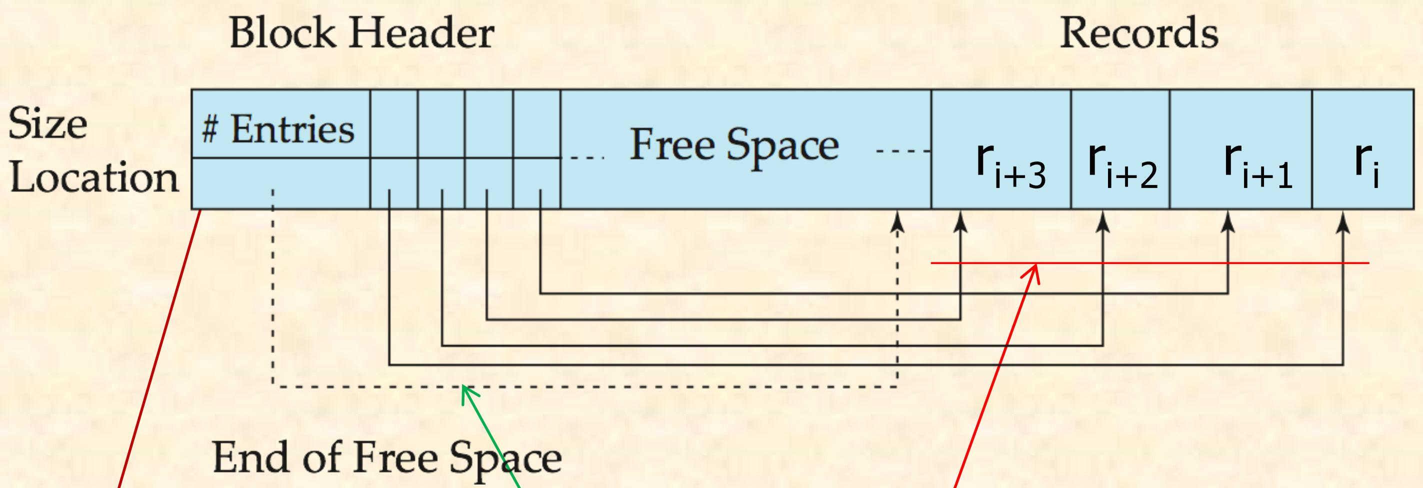
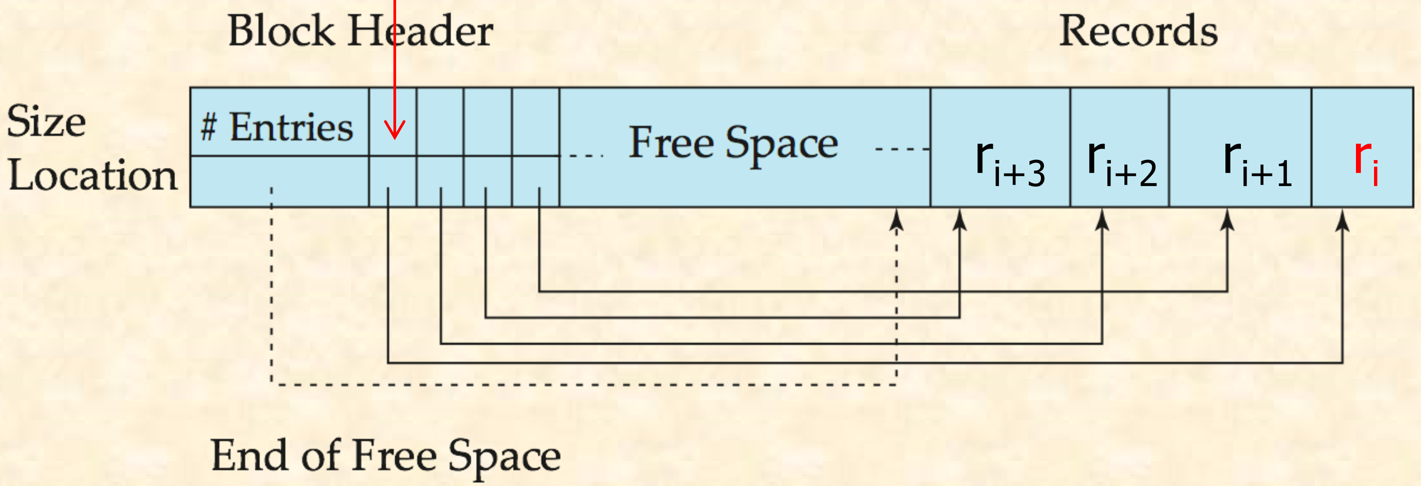


Fig. record$_i$ to record$_{i+3}$ in a page

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

- Records can be moved around within a page to keep them contiguous with no empty space between them
  - e.g. r_i is deleted
  - entry in the header must be updated

- Pointers should not point directly to record — instead they should point to the entry for the record in header
  - e.g. the pointer to r_i



13.3 Organization of Records in Files

- /* 文件包括多个记录，这些记录在磁盘上的存放属于文件(记录)结构问题，即纪录的组织结构
- /*对于某种结构的文件如何去查找、插入、删除记录，属于文件的存取方法
- /*文件的记录结构决定了文件的存取方法

- At the logical file level, from viewpoints of users, DB file can be viewed as a set of ***records***, these records are logically organized in one of the following ways
  - **heap, sequential, hash, clustering**
  - logical organization of records also determine for file users how to access the records, i.e. access methods

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

13.3.1 Heap File Organization

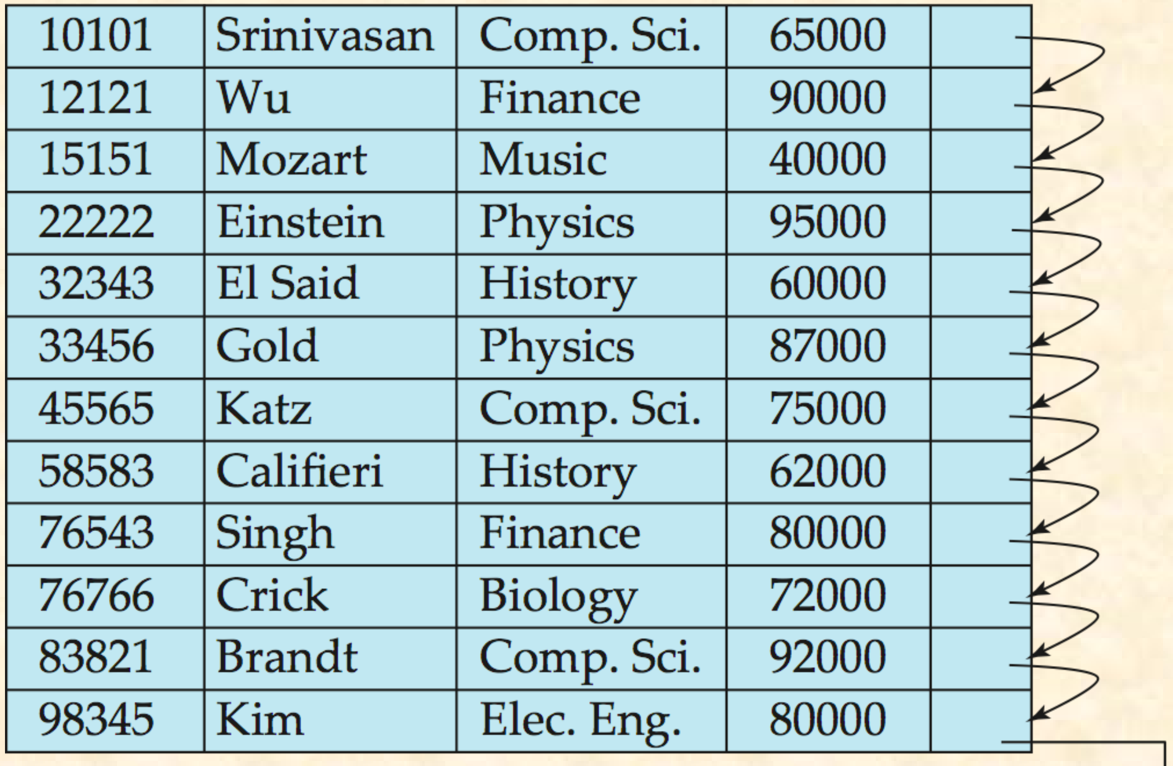
- Any record can be placed anywhere in the file where there is space for the record
- There is no ordering of records
- ***Typically, there is a single file for a relation***
- /* 一般以记录的输入顺序为序，决定了文件中记录顺序
- /*记录的存储顺序与记录中的主键无关

13.3.2 Sequential File Organization

- Records are ***logically*** ordered by search-keys, that is, records are stored in sequential order, based on the value of the search key of each record
    - it is desirable that the records are also ***physically*** stored in search-key order, or as close to search-key order as possible
        - to minimize the number of block accesses in sequential processing
- From viewpoints of DB, sequential access to the file
- Need to reorganize the file from time to time to restore sequential order

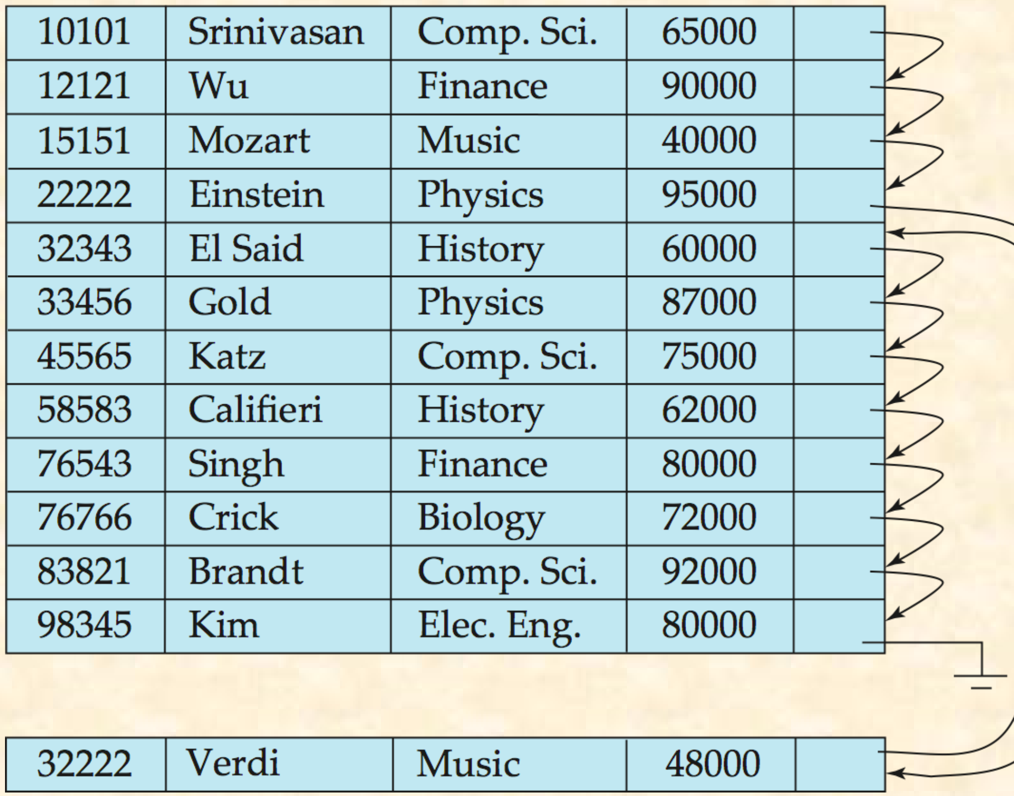
Sequential File Organization

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



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

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



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

32222	Verdi	Music	48000	

13.3.3 *Clustering* File Organization

- /*一个DB文件可以存储来自多个关系的记录。不同关系中有联系的记录存放在同一block中，以提高查找和I/O速度
- Store several relations in one file using a **multitable clustering** file organization, i.e. records of several different relations are stored in the same file
    - motivation: store related records on the same block to minimize I/O

Multitable Clustering File Organization

/一个DB文件存储来自多个关系的纪录。不同关系中有联系的纪录存放在同一block中，以提高查找和I/O速度

department

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

instructor

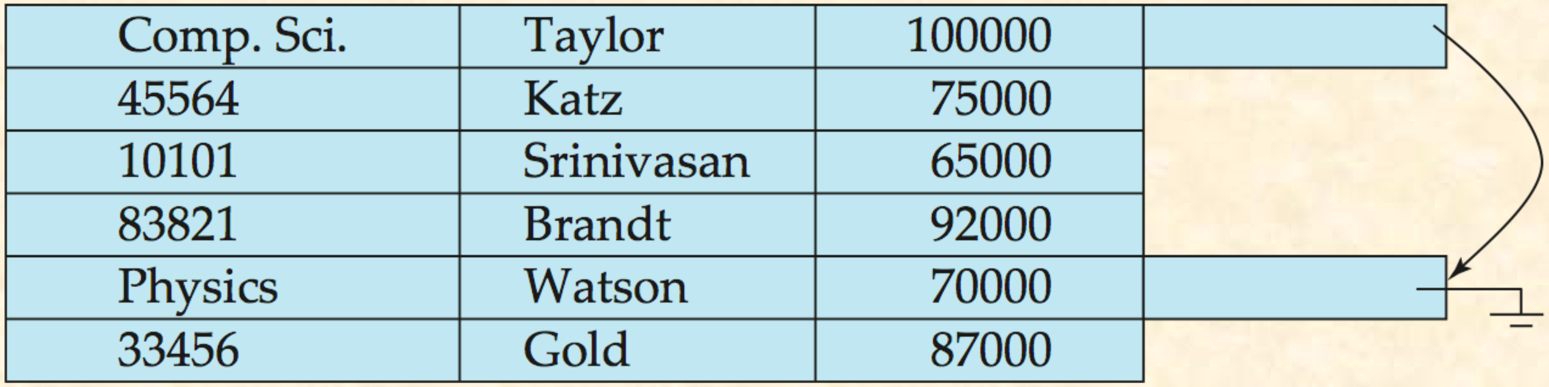
ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

Multitable Clustering File Organization

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

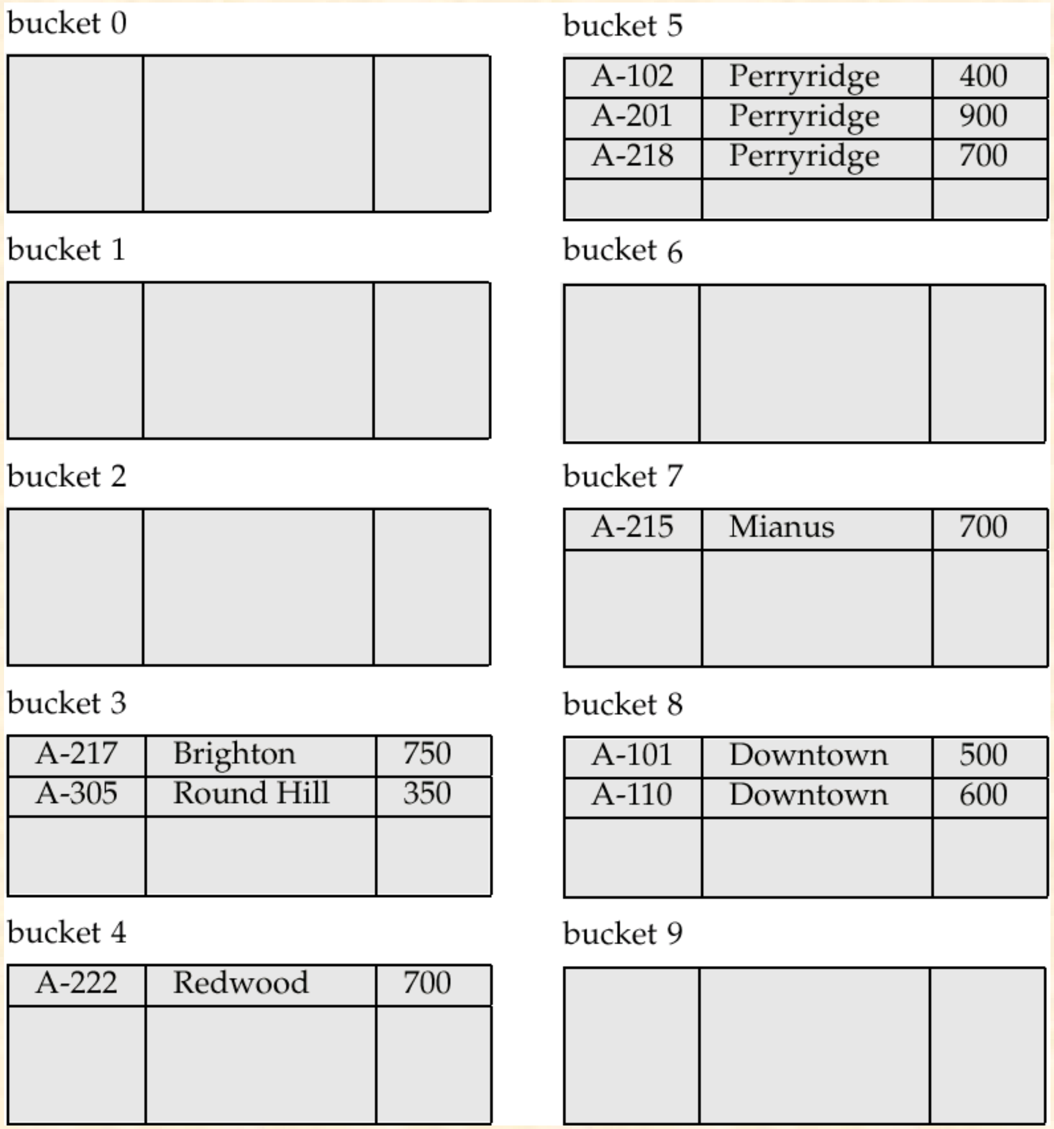


Comp. Sci.	Taylor	100000	
45564	Katz	75000	
10101	Srinivasan	65000	
83821	Brandt	92000	
Physics	Watson	70000	
33456	Gold	87000	

Hashing File Organization

- The file records are stored in a number of ***buckets***, the #bucket is the address of the record
- A hash function on some attributes of file records (i.e. search key) is used to determine the addresses of the records in the file
- Hash function
  - h : {search key} → {physical address of the records}
  - the result of the function specifies in which block/bucket of the file the record should be placed

Hash file organization of ***account*** file, using *branch-name* as the search key



bucket 0

bucket 1

bucket 2

bucket 3

A-217	Brighton	750
A-305	Round Hill	350

bucket 4

A-222	Redwood	700

bucket 5

A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700

bucket 6

bucket 7

A-215	Mianus	700

bucket 8

A-101	Downtown	500
A-110	Downtown	600

bucket 9

13.4 Data Dictionary Storage

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

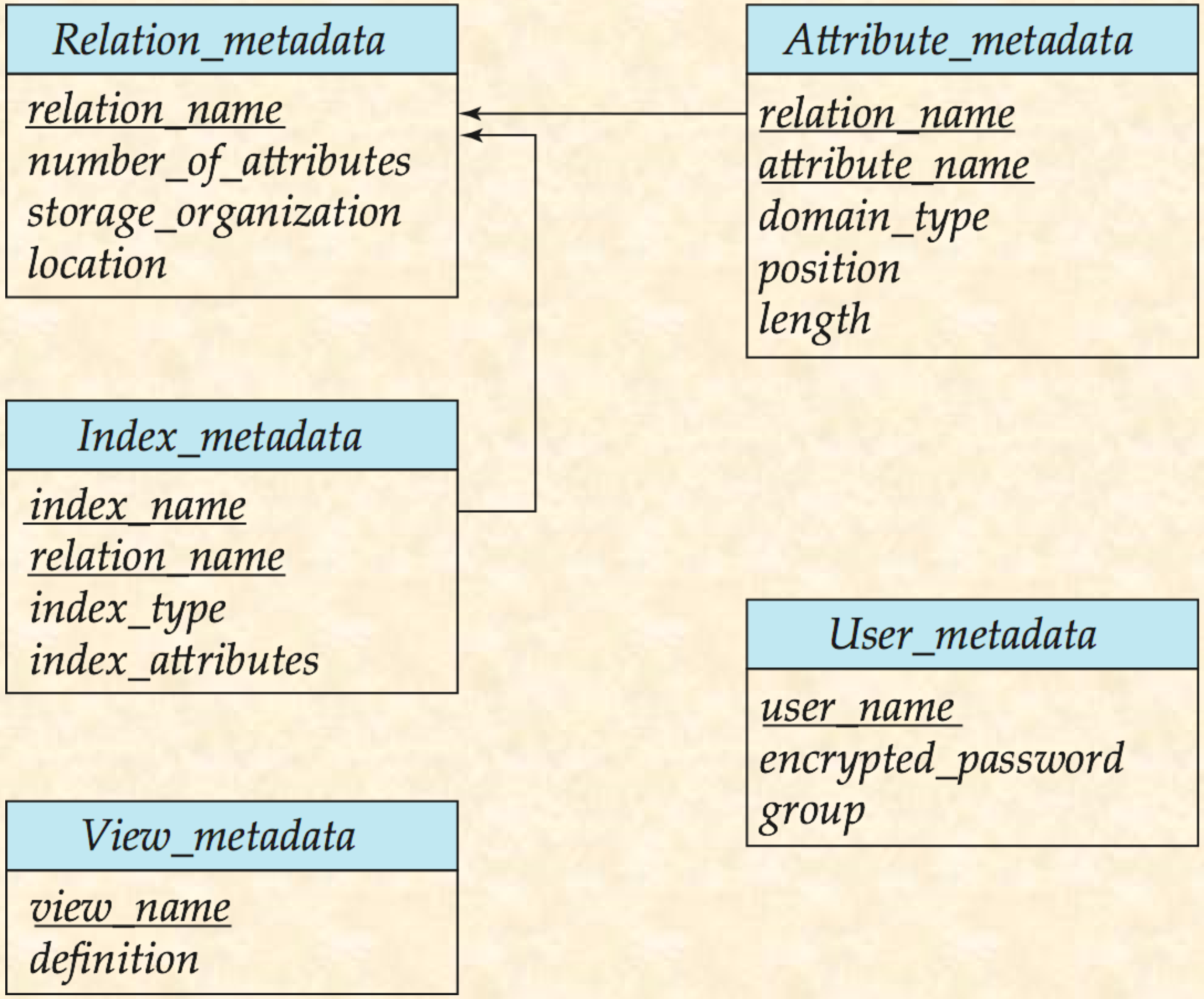
- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation

Data Dictionary Storage (cont.)

- Physical file organization information
  - how relation is stored (sequential/hash/…)
  - physical location of relation
- Information about indices (Chapter 14)

Relational Representation of System Metadata

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



13.5 Buffer Management

- 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
  - e.g. in MS SQL Server, the buffer size is an adjustable parameter that has impact on the system performance, e.g. throughput

13.5.1 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.

13.5.2 Buffer-Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
- Idea behind LRU – use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data

Buffer-Replacement Policies

- LRU can be a bad strategy for certain access patterns involving repeated scans of data
  - For example: when computing the join of 2 relations r and s by a nested loops

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

- Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

Buffer-Replacement Policies (Cont.)

- **Pinned block** – memory block that is not allowed to be written back to disk.
- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy** – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.

```
for each tuple i of instructor do
    for each tuple d of department do
        if i[dept_name] = d[dept_name]
        then begin
                let x be a tuple defined as follows:
                x[ID] := i[ID]
                x[dept_name] := i[dept_name]
                x[name] := i[name]
                x[salary] := i[salary]
                x[building] := d[building]
                x[budget] := d[budget]
                include tuple x as part of result of instructor ⋈ department
            end
    end
end
```

Figure 13.13 Procedure for computing join.

Appendix A 略
§ 10.1-10.4 Physical Storage Media

10-I-1. Storage-device hierarchy (§ 10.1)

- As shown in Fig.10.1
- Hierarchy
  - primary storage, secondary storage, tertiary storage
- Performance index
  - reliability, speed, capacity, cost

Performance Indicators

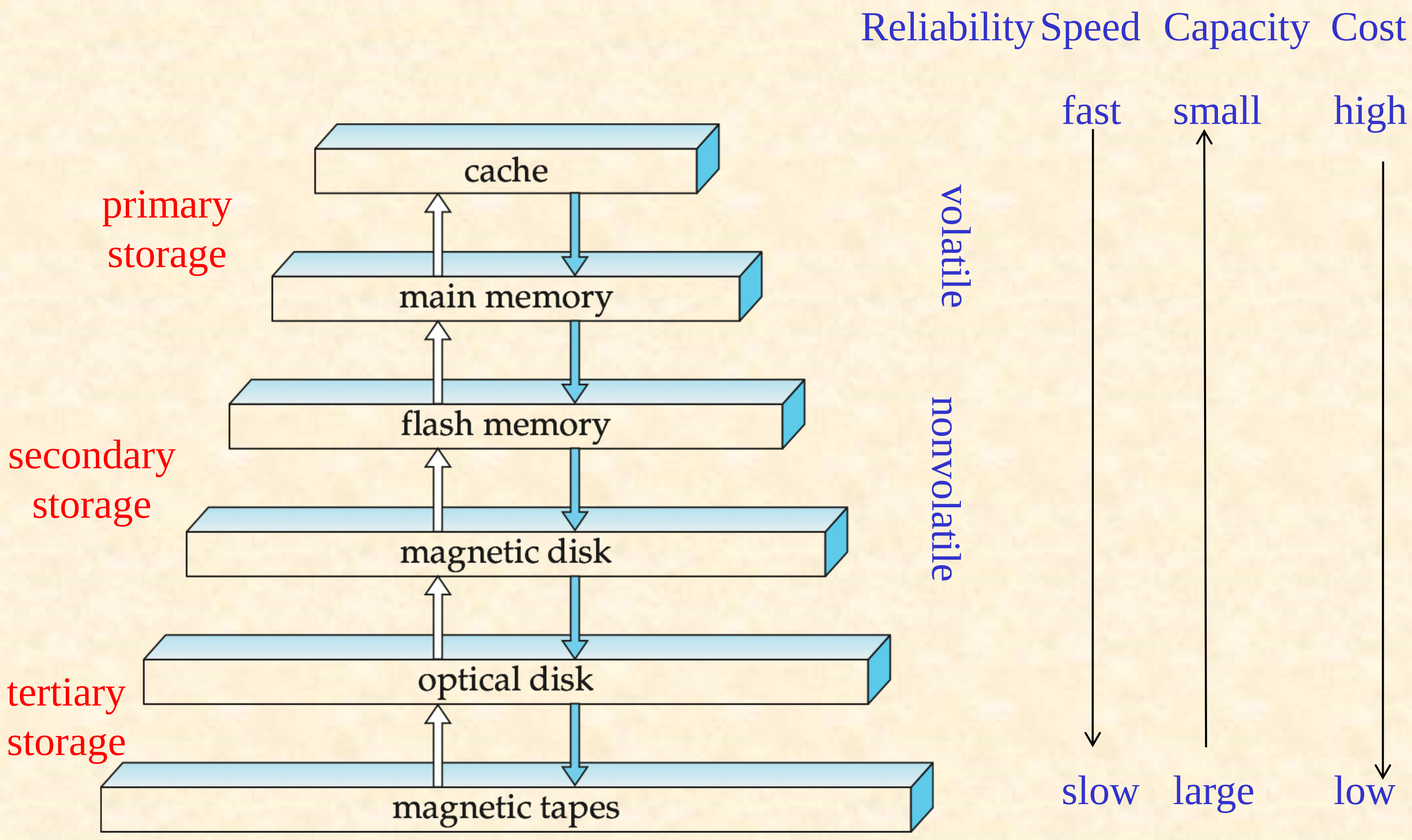


Fig.11.1 Storage-device hierarchy

Storage Hierarchy (Cont.)

- **primary storage:** Fastest media but volatile (cache, main memory).
- **secondary storage:** next level in hierarchy, non-volatile, moderately fast access time
  - also called **on-line storage**
  - E.g. flash memory, magnetic disks
- **tertiary storage:** lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage**
  - E.g. magnetic tape, optical storage

11-I-2 Magnetic Disk

- The disk structure is shown in Fig.11.2
- ***Access time*** is taken as the performance measures of disks
  - the time it takes from when a read or write request is issued to when data transfer begins
- ***Access time*** includes
  - **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.)

磁柱

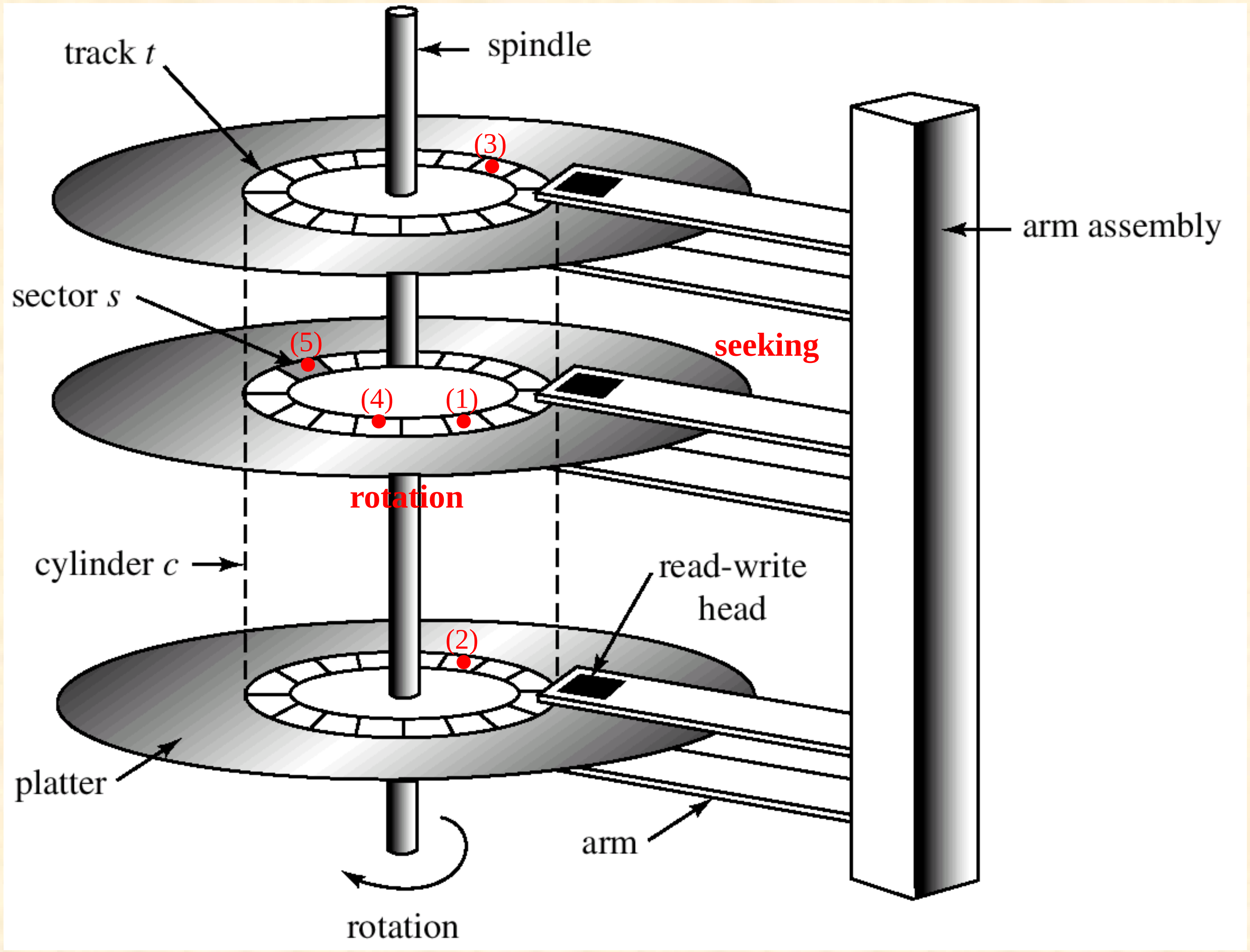


Fig.11.2 Moving-head disk mechanism

北京邮电大学

Magnetic Disks

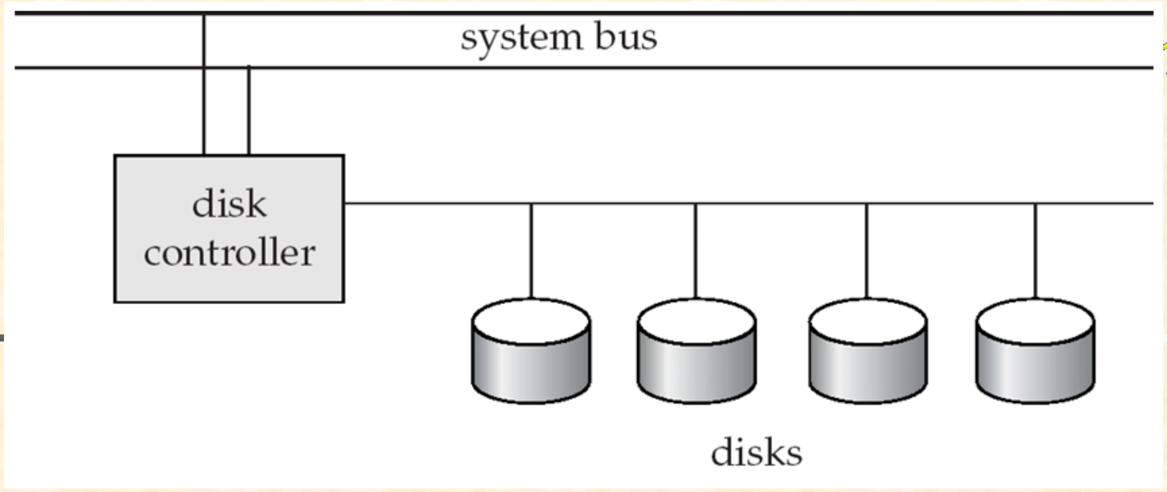
- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information.
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors.**
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)

Magnetic Disks

- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - multiple disk platters on a single spindle (1 to 5 usually)
  - one head per platter, mounted on a common arm.
- **Cylinder** *i* consists of *i*th track of all the platters

Magnetic Disks (Cont.)

- **Disk controller** – interfaces between the computer system and the disk drive hardware.
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Computes and attaches **checksums** to each sector to verify that data is read back correctly
    - If data is corrupted, with very high probability stored checksum won’t match recomputed checksum
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors



- Multiple disks connected to a computer system through a controller
  - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
  - ATA (AT adaptor) range of standards
  - SATA (Serial ATA)
  - SCSI (Small Computer System Interconnect) range of standards
  - SAS (Serial Attached SCSI)
  - Several variants of each standard (different speeds and capabilities)

Disk Subsystem

- Disks usually connected directly to computer system
- In **Storage Area Networks (SAN)**, a large number of disks are connected by a high-speed network to a number of servers
- In **Network Attached Storage (NAS)** networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
    - Average seek time is 1/2 the worst case seek time.
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 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 r.p.m.)

Performance Measures of Disks

- **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, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. SATA: 150 MB/sec, SATA-II 3Gb (300 MB/sec)
    - Ultra 320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

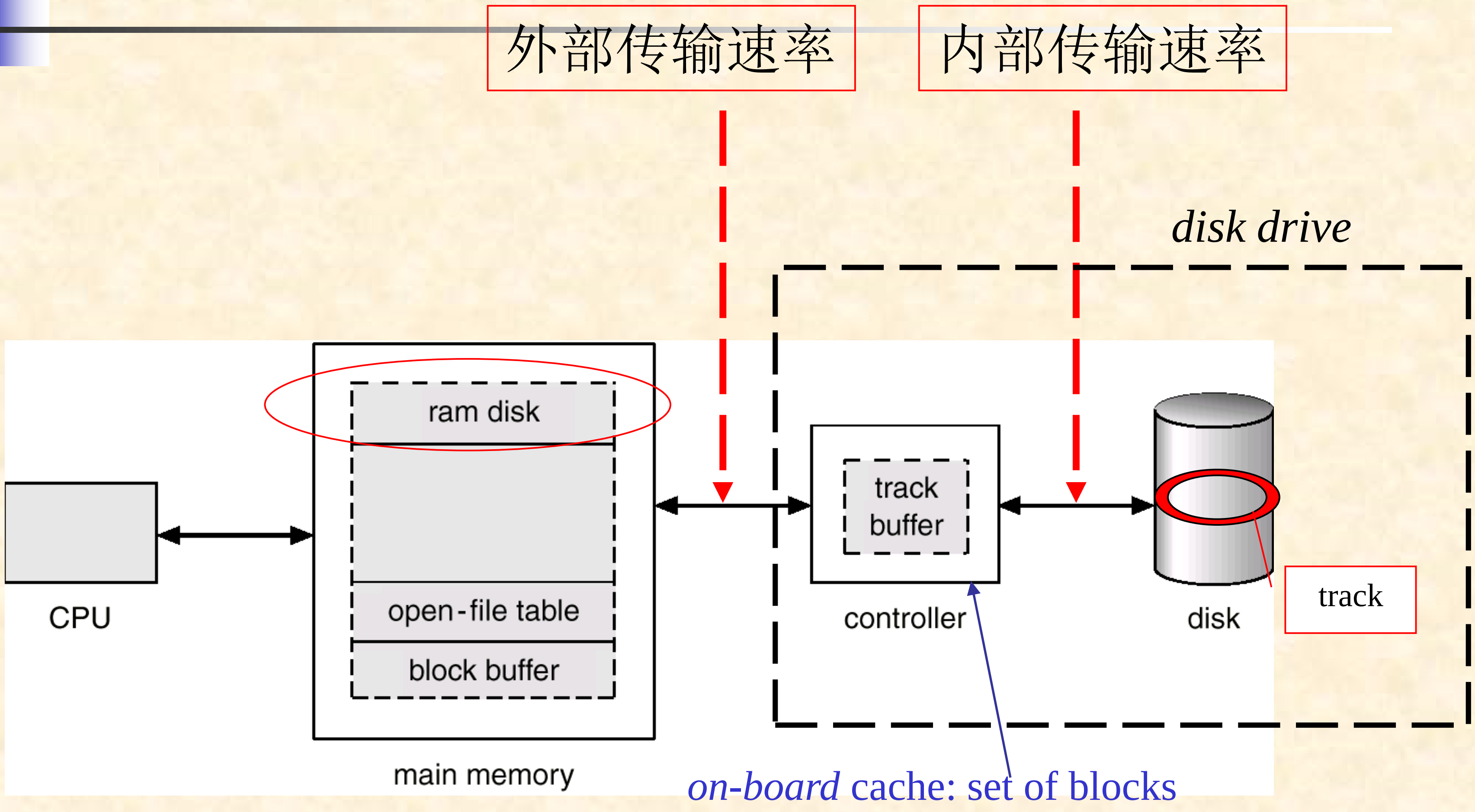


Fig. 11.11.0 on-board in device controller

Overview and Disk Structure (cont.)

- 外部传输速率
  - 硬盘缓存cache（磁盘控制器中的I/O 寄存器）与内存间的传输速率
  - 也称为：突发数据传输速率、接口速率

- 内部传输速率
  - 磁头在盘片上读写数据速率，如50MB/s
  - 也称为：（最大最小）持续传输速率

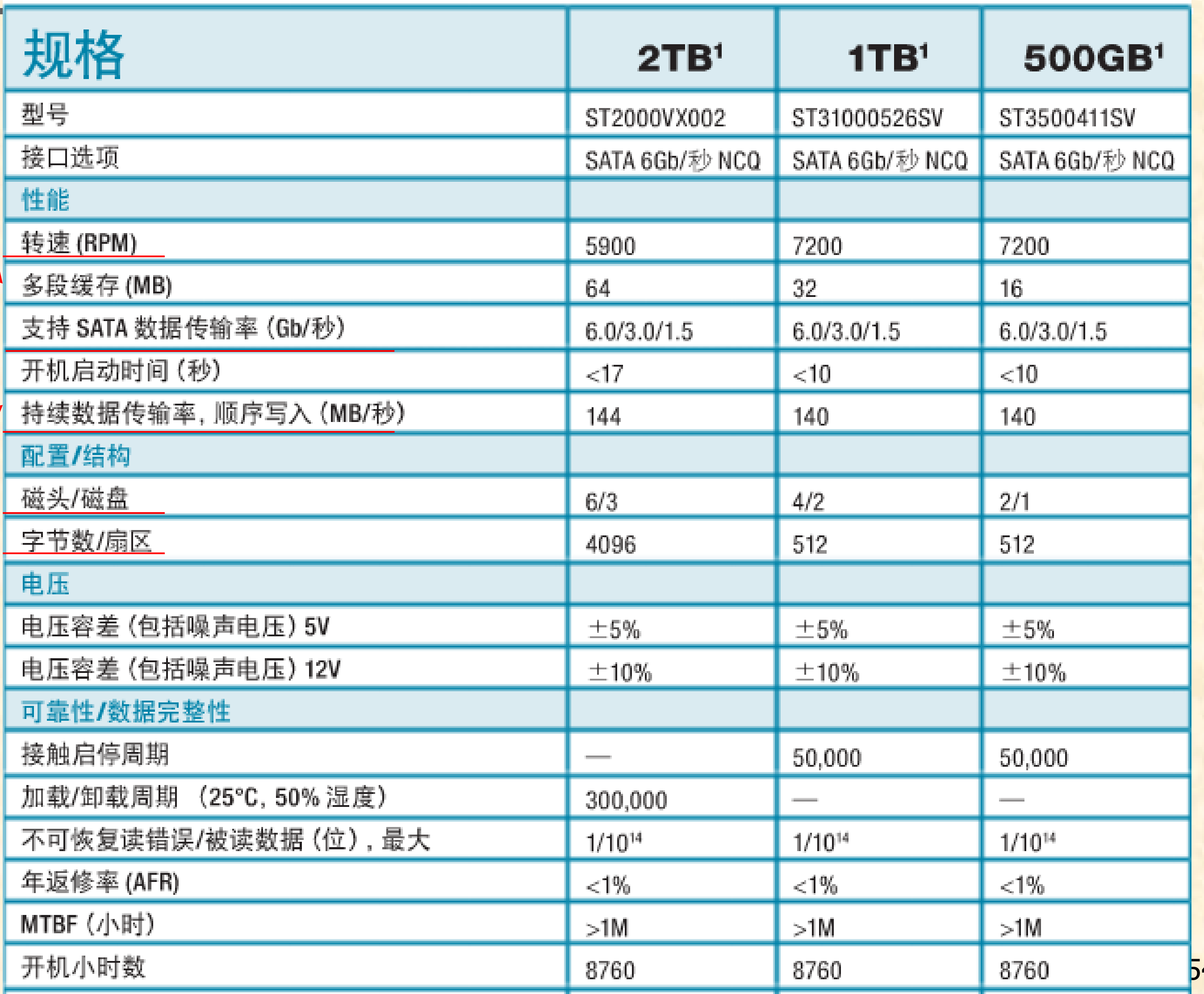
Segate(希捷) SV35 Series ——面向视频监控的高性能、大容量硬盘

硬盘和内存间cache/buffer

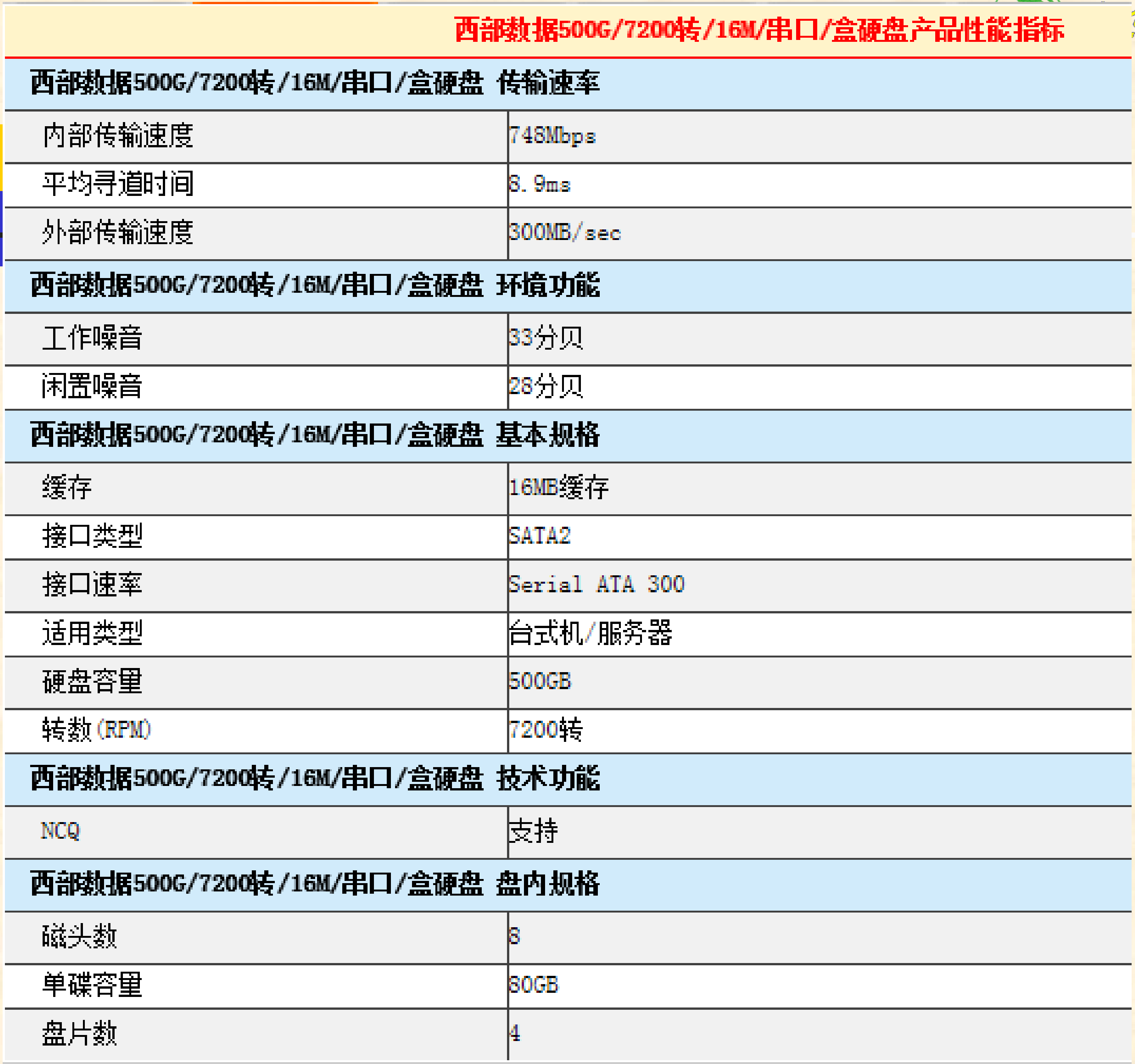
外部传输速率

内部传输速率

平均寻道时间 8.5ms



规格	2TB[1]	1TB[1]	500GB[1]
型号	ST2000VX002	ST31000526SV	ST3500411SV
接口选项	SATA 6Gb/秒 NCQ	SATA 6Gb/秒 NCQ	SATA 6Gb/秒 NCQ
性能			
转速 (RPM)	5900	7200	7200
多段缓存 (MB)	64	32	16
支持 SATA 数据传输率 (Gb/秒)	6.0/3.0/1.5	6.0/3.0/1.5	6.0/3.0/1.5
开机启动时间 (秒)	<17	<10	<10
持续数据传输率，顺序写入 (MB/秒)	144	140	140
配置/结构			
磁头/磁盘	6/3	4/2	2/1
字节数/扇区	4096	512	512
电压			
电压容差 (包括噪声电压) 5V	±5%	±5%	±5%
电压容差 (包括噪声电压) 12V	±10%	±10%	±10%
可靠性/数据完整性			
接触启停周期	—	50,000	50,000
加载/卸载周期 (25°C, 50% 湿度)	300,000	—	—
不可恢复读错误/被读数据 (位)，最大	$1/10^{14}$	$1/10^{14}$	$1/10^{14}$
年返修率 (AFR)	<1%	<1%	<1%
MTBF (小时)	>1M	>1M	>1M
开机小时数	8760	8760	8760



西部数据500G/7200转/16M/串口/盒硬盘产品性能指标

西部数据500G/7200转/16M/串口/盒硬盘 传输速率	
内部传输速度	748Mbps
平均寻道时间	8.9ms
外部传输速度	300MB/sec
西部数据500G/7200转/16M/串口/盒硬盘 环境功能	
工作噪音	33分贝
闲置噪音	28分贝
西部数据500G/7200转/16M/串口/盒硬盘 基本规格	
缓存	16MB缓存
接口类型	SATA2
接口速率	Serial ATA 300
适用类型	台式机/服务器
硬盘容量	500GB
转数(RPM)	7200转
西部数据500G/7200转/16M/串口/盒硬盘 技术功能	
NCQ	支持
西部数据500G/7200转/16M/串口/盒硬盘 盘内规格	
磁头数	8
单碟容量	80GB
盘片数	4

Performance Measures (Cont.)

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

Optimization of Disk-Block Access

- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes
- **Disk-arm-scheduling** algorithms order pending accesses to tracks so that disk arm movement is minimized
  - **elevator algorithm**:

Optimization of Disk Block Access (Cont.)

- **File organization** – optimize block access time by organizing the blocks to correspond to how data will be accessed
  - E.g. Store related information on the same or nearby cylinders.
  - Files may get **fragmented** over time
    - E.g. if data is inserted to/deleted from the file
    - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
    - Sequential access to a fragmented file results in increased disk arm movement
  - Some systems have utilities to defragment the file system, in order to speed up file access

Optimization of Disk Block Access (Cont.)

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  - Non-volatile RAM: battery backed up RAM or flash memory
    - Even if power fails, the data is safe and will be written to disk when power returns
  - Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - *Writes can be reordered to minimize disk arm movement*

Optimization of Disk Block Access (Cont.)

- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
  - **Journaling file systems** write data in safe order to NV-RAM or log disk
  - Reordering without journaling: risk of corruption of file system data

北京邮电大学

Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - used widely for storage, since it is much cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
  - transfer rate around 20 MB/sec
  - **solid state disks**: use multiple flash storage devices to provide higher transfer rate of 100 to 200 MB/sec

Flash Storage

- erase is very slow (1 to 2 millisecs)
    - erase block contains multiple pages
    - **remapp**ing of logical page addresses to physical page addresses avoids waiting for erase
        - **translation table** tracks mapping
            - also stored in a label field of flash page
        - remapping carried out by **flash translation layer**
    - after 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
        - **wear leveling**

RAID

- **RAID: Redundant Arrays of Independent Disks**
  - disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - high capacity and high speed by using multiple disks in parallel,
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails

RAID

- The chance that some disk out of a set of *N* disks will fail is much higher than the chance that a specific single disk will fail.
  - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
  - Techniques for using redundancy to avoid data loss are critical with large numbers of disks
- Originally a cost-effective alternative to large, expensive disks
  - I in RAID originally stood for ``inexpensive''
  - Today RAIDs are used for their higher reliability and bandwidth.
    - The “I” is interpreted as independent

Improvement of Reliability via Redundancy

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or **shadowing**)
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - Reads can take place from either disk
  - If one disk in a pair fails, data still available in the other
    - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
        - Except for dependent failure modes such as fire or building collapse or electrical power surges

Improvement of Reliability via Redundancy

- **Mean time to data loss** depends on mean time to failure, and **mean time to repair**
  - E.g. MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of $500*10^6$ hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)

Improvement in Performance via Parallelism

- Two main goals of parallelism in a disk system:
  1. Load balance multiple small accesses to increase throughput
  2. Parallelize large accesses to reduce response time.
- Improve transfer rate by striping data across multiple disks.
- **Bit-level striping** – split the bits of each byte across multiple disks
  - In an array of eight disks, write bit *i* of each byte to disk *i*.
  - Each access can read data at eight times the rate of a single disk.
  - But seek/access time worse than for a single disk
    - Bit level striping is not used much any more

Improvement in Performance via Parallelism

- **Block-level striping** – with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1
  - Requests for different blocks can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel

RAID Levels

- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
  - Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics

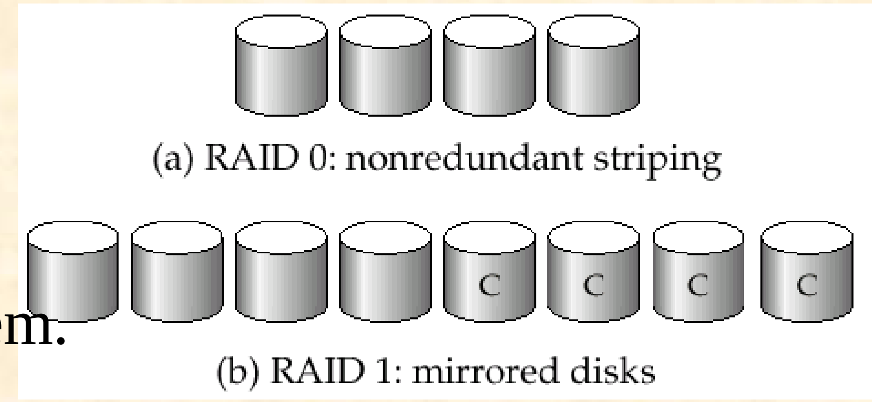
RAID Level 0: Block striping; non-redundant.

Used in high-performance applications where data loss is not critical.

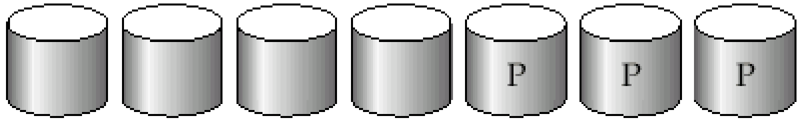
RAID Level 1: Mirrored disks with block striping

Offers best write performance.

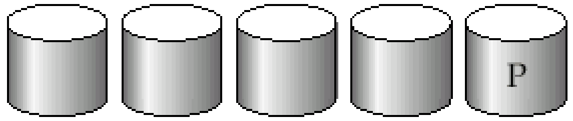
Popular for applications such as storing log files in a database system.



- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.
- **RAID Level 3**: Bit-Interleaved Parity
  - a single parity bit is enough for error correction, not just detection, since we know which disk has failed
    - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
    - To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)

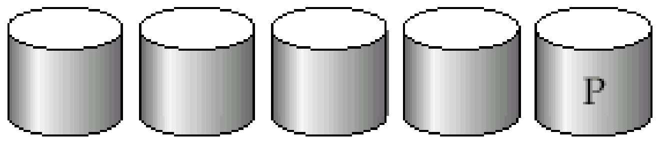


(c) RAID 2: memory-style error-correcting codes



(d) RAID 3: bit-interleaved parity

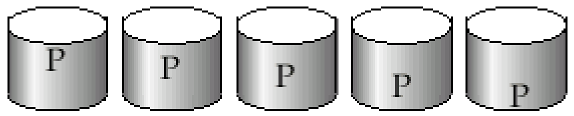
- **RAID Level 3** (Cont.)
  - Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O.
  - Subsumes Level 2 (provides all its benefits, at lower cost).
- **RAID Level 4:** Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.
  - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.



(e) RAID 4: block-interleaved parity

- **RAID Level 4** (Cont.)
    - Provides higher I/O rates for independent block reads than Level 3
        - block read goes to a single disk, so blocks stored on different disks can be read in parallel
    - Provides high transfer rates for reads of multiple blocks than no-striping
    - Before writing a block, parity data must be computed
        - Can be done by using old parity block, old value of current block and new value of current block (2 block reads + 2 block writes)
        - Or by recomputing the parity value using the new values of blocks corresponding to the parity block
            - More efficient for writing large amounts of data sequentially
    - Parity block becomes a bottleneck for independent block writes since every block write also writes to parity disk

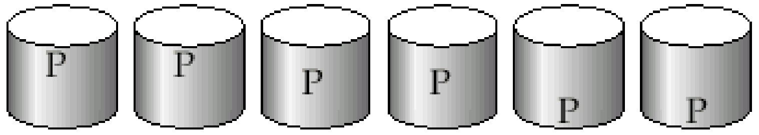
- **RAID Level 5:** Block-Interleaved Distributed Parity; partitions data and parity among all *N* + 1 disks, rather than storing data in *N* disks and parity in 1 disk.
  - E.g., with 5 disks, parity block for *n*th set of blocks is stored on disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks.



(f) RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

- **RAID Level 5** (Cont.)
  - Higher I/O rates than Level 4.
    - Block writes occur in parallel if the blocks and their parity blocks are on different disks.
  - Subsumes Level 4: provides same benefits, but avoids bottleneck of parity disk.
- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost; not used as widely.



(g) RAID 6: P + Q redundancy

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - E.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5

Choice of RAID Level

- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement, which block striping (level 5) avoids
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

Choice of RAID Level (Cont.)

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - so there is often no extra monetary cost for Level 1!

Choice of RAID Level (Cont.)

- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

Hardware Issues

- **Software RAID**: RAID implementations done entirely in software, with no special hardware support
- **Hardware RAID**: RAID implementations with special hardware
    - Use non-volatile RAM to record writes that are being executed

Hardware Issues

- Beware: power failure during write can result in corrupted disk
  - E.g. failure after writing one block but before writing the second in a mirrored system
  - Such corrupted data must be detected when power is restored
    - Recovery from corruption is similar to recovery from failed disk
    - NV-RAM helps to efficiently detected potentially corrupted blocks
      - Otherwise all blocks of disk must be read and compared with mirror/parity block

Hardware Issues

- **Latent failures**: data successfully written earlier gets damaged
  - can result in data loss even if only one disk fails
- **Data scrubbing:**
  - continually scan for latent failures, and recover from copy/parity
- **Hot swapping**: replacement of disk while system is running, without power down
  - Supported by some hardware RAID systems,
  - reduces time to recovery, and improves availability greatly

Hardware Issues

- Many systems maintain spare disks which are kept online, and used as replacements for failed disks immediately on detection of failure
  - Reduces time to recovery greatly
- Many hardware RAID systems ensure that a single point of failure will not stop the functioning of the system by using
  - Redundant power supplies with battery backup
  - Multiple controllers and multiple interconnections to guard against controller/interconnection failures

Optical Disks

- Compact disk-read only memory (CD-ROM)
  - Removable disks, 640 MB per disk
  - Seek time about 100 msec (optical read head is heavier and slower)
  - Higher latency (3000 RPM) and lower data-transfer rates (3-6 MB/s) compared to magnetic disks
- Digital Video Disk (DVD)
  - DVD-5 holds 4.7 GB , and DVD-9 holds 8.5 GB
  - DVD-10 and DVD-18 are double sided formats with capacities of 9.4 GB and 17 GB
  - Blu-ray DVD: 27 GB (54 GB for double sided disk)
  - Slow seek time, for same reasons as CD-ROM

Optical Disks

- Record once versions (CD-R and DVD-R) are popular
  - data can only be written once, and cannot be erased.
  - high capacity and long lifetime; used for archival storage
  - Multi-write versions (CD-RW, DVD-RW, DVD+RW and DVD-RAM) also available

Magnetic Tapes

- Hold large volumes of data and provide high transfer rates
  - Few GB for DAT (Digital Audio Tape) format, 10-40 GB with DLT (Digital Linear Tape) format, 100 GB+ with Ultrium format, and 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Tapes are cheap, but cost of drives is very high
- Very slow access time in comparison to magnetic and optical disks
  - limited to sequential access.
  - Some formats (Accelis) provide faster seek (10s of seconds) at cost of lower capacity

Magnetic Tapes

- Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another.
- Tape jukeboxes used for very large capacity storage
  - Multiple petabyes (10^{15} bytes)

Appendix B
Data Organization and Access in DBS

- Data organization in DBS

- Data access in DBS
  - in small DBS
  - in medium or large scale DBS

北京邮电大学

略

Appendix B —I Data Organization in DBS

- DBS provides the services of data organization and access for DBS user
- From viewpoints of DBS users, data is organized as
  - relational ***tables***, ***tuples***, and ***attributes***
    - *user-oriented* logical organization of data
- From viewpoints of operating systems, data is stored as ***OS file***, also known as ***DB file***
  - DB file is logically organized as a sequence of ***records*** (P415, § 11.6)
  - DB file resides on secondary storages, i.e. disks, in unit of ***block***

Appendix B—I
Data Organization in DBS (cont.)

- DBS数据的基本逻辑单位—元组/数据行与OS文件中数据的基本逻辑单位—文件记录的对应关系
  - 1个元组对应于文件系统的1个逻辑记录
    - fixed-length record: § 11.6.1
    - variable-length record: § 11.6.2
  - 1个关系表对应于1个OS文件，或1个文件存储多个关系表的数据

- record与磁盘基本存储单位—block的对应关系关系
  - in DBS，1个block包含多个文件逻辑记录

Appendix B —I I
Data Access in DBS

- 在一些小型关系数据库系统（如dbase、Foxbase）中，关系表被存储在单独的文件中，DBMS利用OS文件系统作为其DBMS物理层实现基础，利用OS文件系统实现数据组织与访问
  - 操作系统为DB分配所需的磁盘block
    - 有可能将逻辑上相邻的数据分配到磁盘上不同的区域，导致连续访问时效率下降
  - 用户对DB文件数据的访问需要借助于OS文件系统提供的文件结构和存取路径
    - DBMS对OS依赖太大，不利于数据库系统的移植

Appendix B —I I
Data Access in DBS (cont.)

- DBS数据访问过程中，DBMS的storage management（Fig.1.4) 、OS文件系统、I/O子系统完成如下地址变换
  - DB关系表中元组/数据行 到 DB文件中数据的逻辑地址（文件名，记录号）间的映射
  - DB文件记录号到外设上数据物理地址（记录所在的物理块号和具体地址）的映射
- Fig. A.1

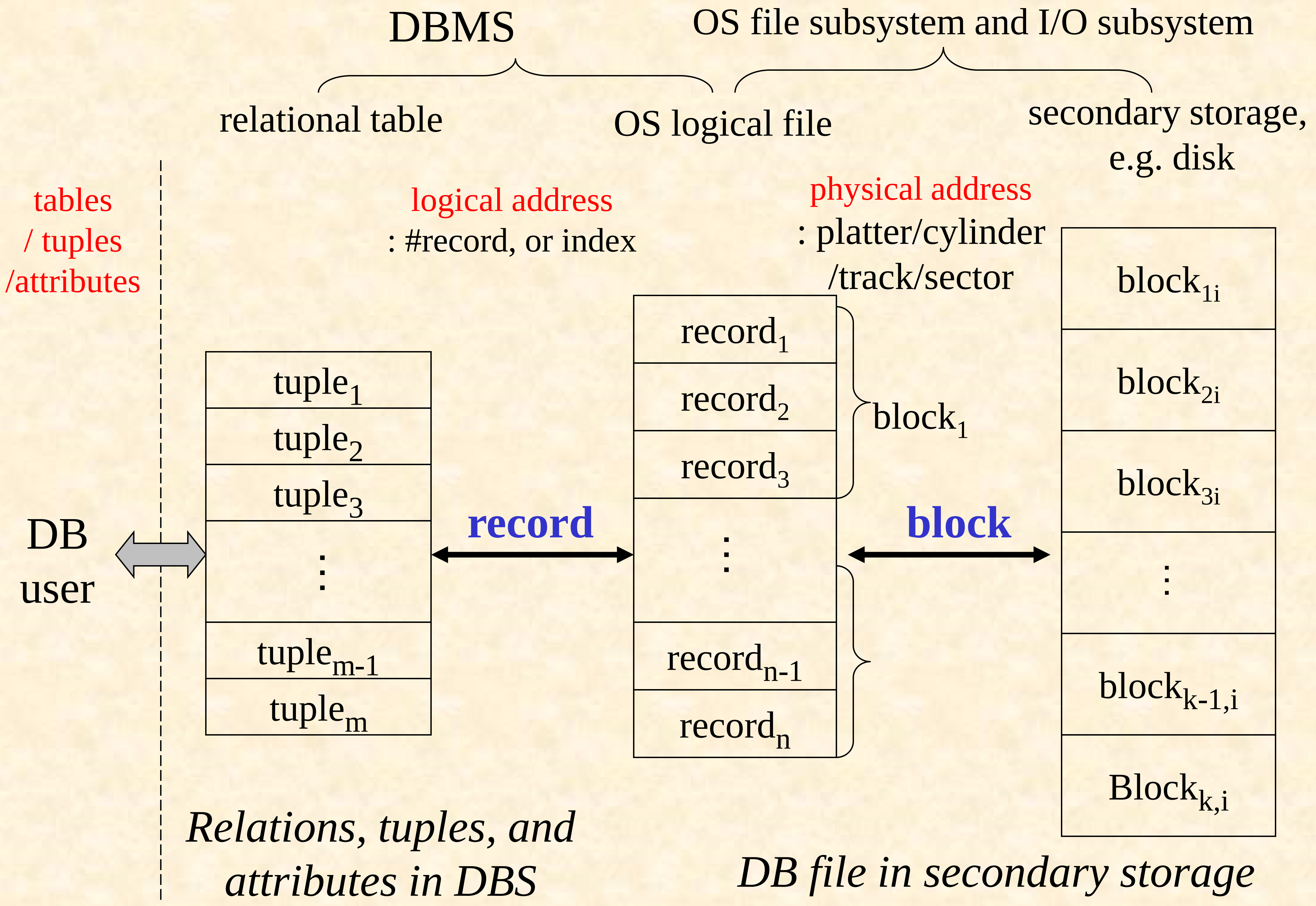


Fig.A.1 Data Access in DBS-I

Appendix B —III
Data Access in DBS

- 一些现代大型关系数据库系统，如SQL Server、Oracle等，为了适应数据的动态变化、提供数据快速访问路径，DB数据仍然以DB文件的方式存在，但对DB文件的管理并不直接依赖于OS文件系统
- 在创建数据库时，DBMS向OS一次性申请所需磁盘空间，在此磁盘存储空间中，DBMS独立地为数据库中的数据设计存储和访问结构
  - 分配管理DB文件，提供数据访问机制

- SQL Server as Case Study
  - Fig.A.3, Fig.A.4, Fig.A.5
  - 借鉴了操作系统虚拟内存管理技术中page/frame的思想

Appendix B —III
Data Access in DBS (cont.)

- SQL Server中，一个用户数据库由用于存储表(table)和索引(index)的磁盘存储空间构成，这些空间被分配在一个或多个操作系统文件上
- 数据库被划分许多逻辑页(page)，每个逻辑页大小为8KB
  - 每个数据库文件由多个从0开始连续编号的页组成。每个页逻辑上划分为许多从1开始连续编号的槽(slot)，每个槽用于存放一个数据库表中的数据行/元组/纪录

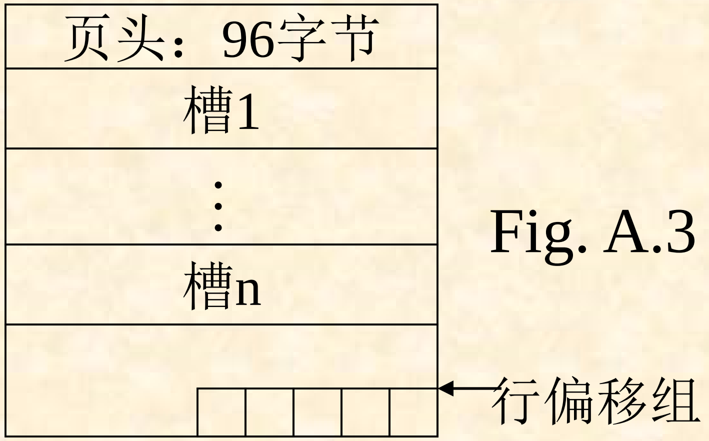


Fig. A.3 Page in SQL Server

Appendix B —III
Data Access in DBS (cont.)

- 对于1个DB表中的数据行/元组/纪录，SQL Server用该表所在的数据库文件ID、数据行所在页ID和槽ID来标识
  - 数据行的逻辑地址：<文件ID, 页ID, 槽ID>
- SQL Server以盘区为单位，为DB文件的页分配磁盘上的物理存储空间
  - 盘区大小为64K，占用8个连续的页
  - DB文件中数据行/元组/纪录物理地址：数据行所在盘区的物理地址，如block#
- SQL Server利用全局分配映象GAM和共享全局分配映象SGAM 这2个数据结构纪录每个盘区的使用情况
  - 是否空闲、是否共享

Appendix B —III
Data Access in DBS (cont.)

- SQL Server利用索引分配映象IAM数据结构纪录每个数据库文件的页与该页所在物理盘区间的对应关系
- SQL Server利用索引实现对关系表中的数据行/元组的访问
  - Index: search key → <文件ID, 页ID, 槽ID>
  - SQL Server可对关系表的主键自动建立索引
- SQL Server数据存储与访问机制
  - Fig. A.4
  - 2级地址映射
    - 基于索引的search key → 逻辑地址：<文件ID, 页ID, 槽ID>
    - 基于IAM的数据行逻辑地址→数据行物理地址（盘区物理地址）

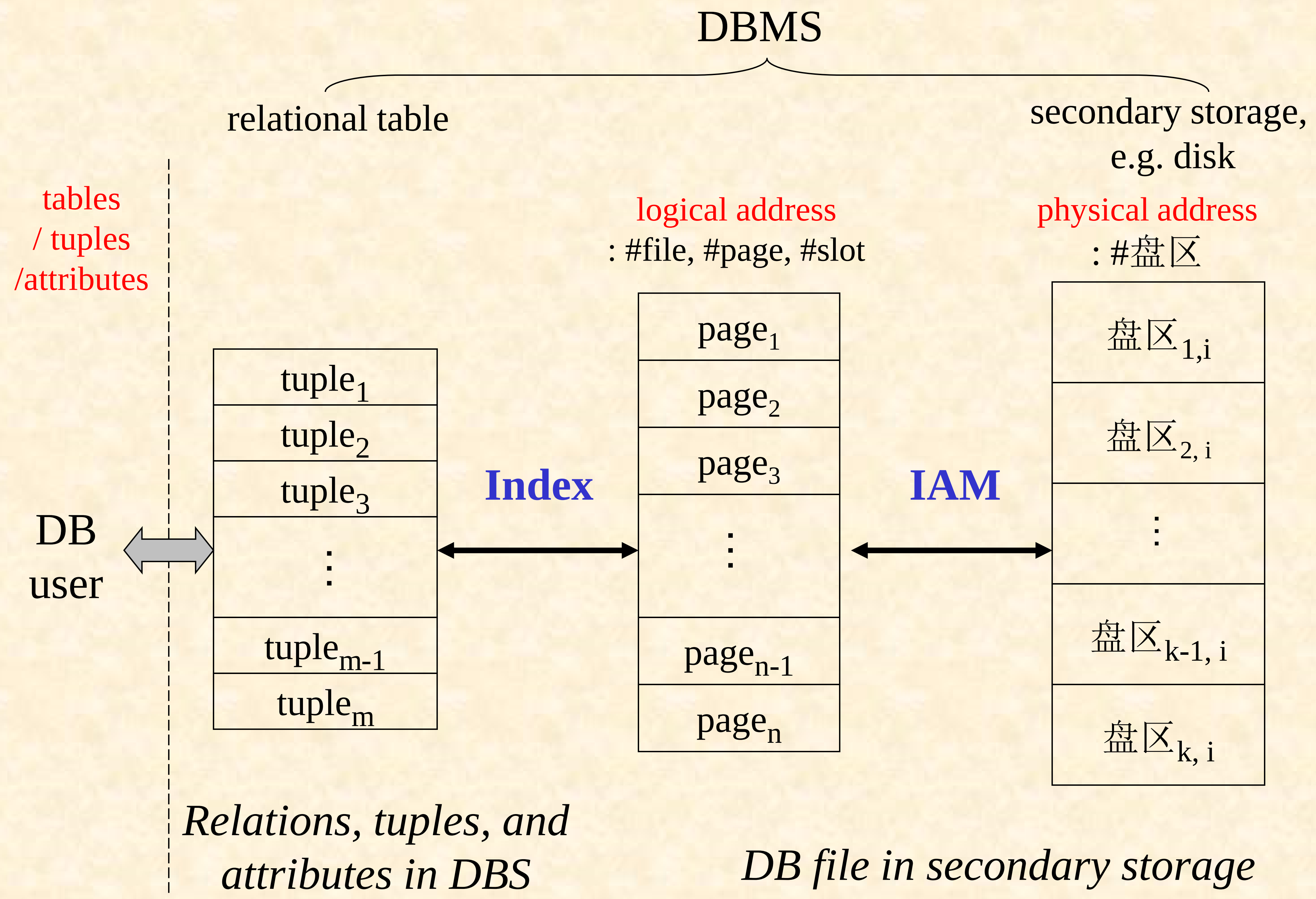


Fig.A.4 Data Access in SQL Server