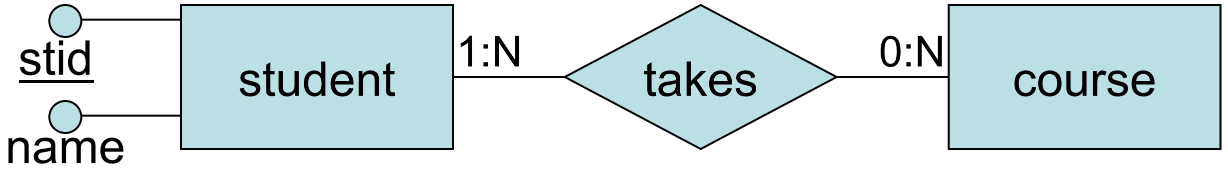
Physical Storage Organization

Outline

- Where and How data are stored?
  - physical level
  - logical level

Building a Database: High-Level

- Design conceptual schema using a data model, e.g. ER, UML, etc.



Building a Database: Logical-Level

- Design logical schema, e.g. relational, network, hierarchical, object-relational, XML, etc schemas
- Data Definition Language (DDL)

CREATE TABLE student
(cid char(8) primary key,name varchar(32))

student	
<u>cid</u>	name

Populating a Database

- Data Manipulation Language (DML)

INSERT INTO student VALUES ('00112233', 'Paul')

student	
cid	name
00112233	Paul

Transaction operations

- Transaction: a collection of operations performing a single logical function

```
BEGIN TRANSACTION transfer
  UPDATE bank-account SET balance = balance - 100 WHERE account=1
  UPDATE bank-account SET balance = balance + 100 WHERE account=2
COMMIT TRANSACTION transfer
```

- A failure during a transaction can leave system in an inconsistent state, eg transfers between bank accounts.

Where and **How** all this information is stored?

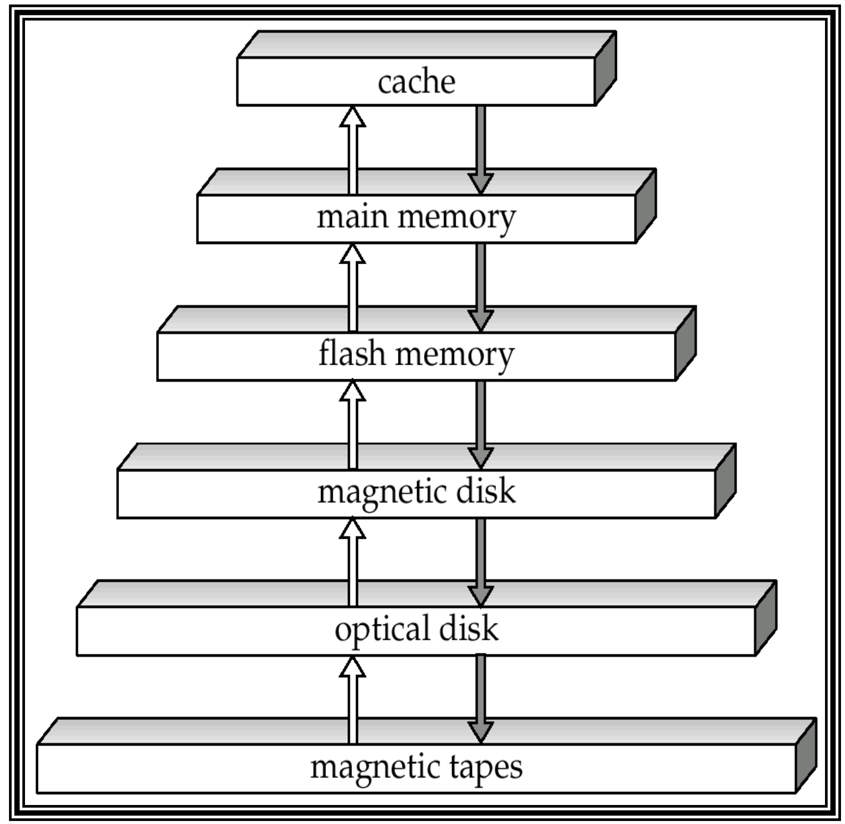
- Metadata: tables, attributes, data types, constraints, etc
- Data: records
- Transaction logs, indices, etc

Where: In Main Memory?

- Fast!
- But:
  - Too small
  - Too expensive
  - Volatile

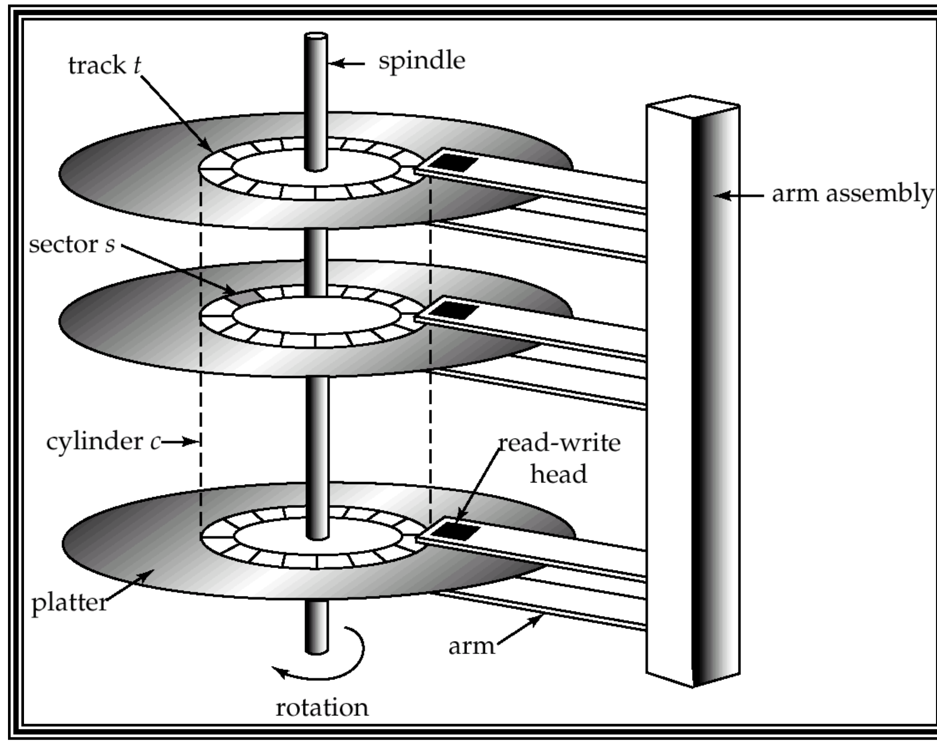
Physical Storage Media

- Primary Storage
  - Cache
  - Main memory
- Secondary Storage
  - Flash memory
  - Magnetic disk
- Offline Storage
  - Optical disk
  - Magnetic tape



Magnetic Disks

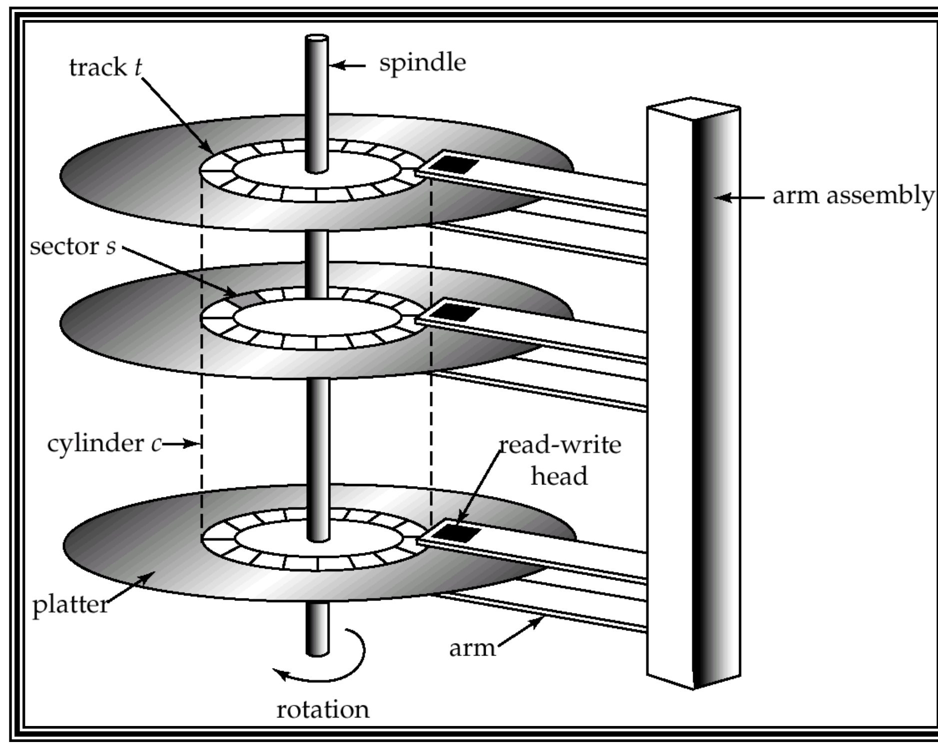
- Random Access
- Inexpensive
- Non-volatile



How do disks work?

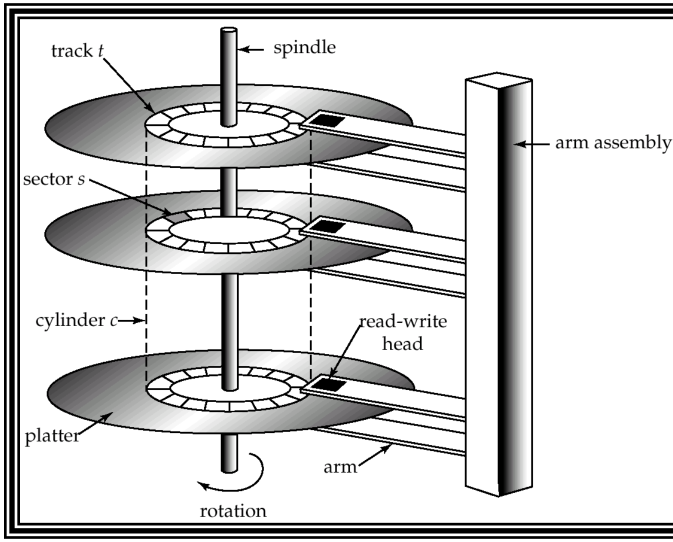
- Platter: covered with magnetic recording material
- Track: logical division of platter surface
- Sector: hardware division of tracks
- Block: OS division of tracks
  - Typical block sizes:

    512 B, 2KB, 4KB
- Read/write head



Disk I/O

- Disk I/O := block I/O
  - Hardware address is converted to Cylinder, Surface and Sector number
  - Modern disks: Logical Sector Address 0…n
- Access time: time from read/write request to when data transfer begins
  - Seek time: the head reaches correct track
    - Average seek time 5-10 msec
  - Rotation latency time: correct block rotated under head
    - 5400 RPM, 15K RPM
    - On average 4-11 msec
- Block Transfer Time



Optimize I/O

- Database system performance I/O bound
- Improve the speed of access to disk:
  - Scheduling algorithms
  - File Organization
- Introduce disk redundancy
  - Redundant Array of Independent Disks (RAID)
- Reduce number of I/Os
  - Query optimization, indices

~~**Where**~~ and **How** all this information is stored?

- Metadata: tables, attributes, data types, constraints, etc
- Data: records
- Transaction logs, indices, etc

- A collection of **files (or tables)**
  - **Physically** partitioned into **pages or data blocks**
  - **Logically** partitioned into **records**

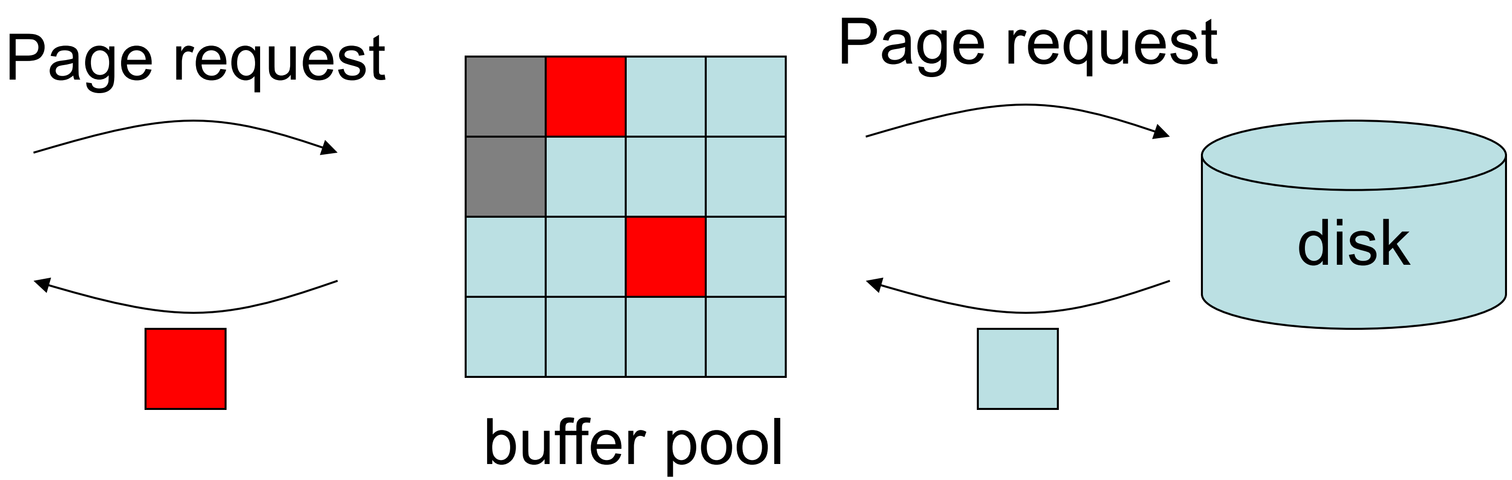
Storage Access

- A collection of files
  - Physically partitioned into pages
  - Typical database page sizes: 2KB, 4KB, 8KB
  - Reduce number of block I/Os := reduce number of page I/Os
  - How?

- Buffer Manager

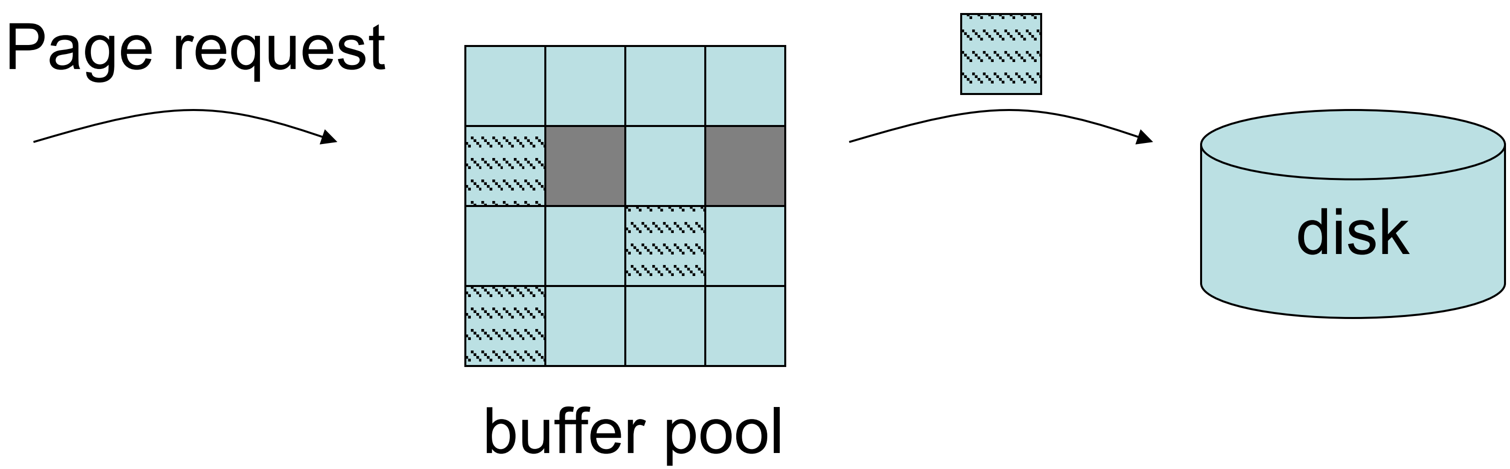
Buffer Management (1/2)

- Buffer: storing a page copy
- Buffer manager: manages a pool of buffers
  - Requested page in pool: hit!
  - Requested page in disk:
    - Allocate page frame
    - Read page and pin
- Problems?



Buffer Management (2/2)

- What if no empty page frame exists:
  - Select victim page
  - Each page associated with dirty flag
  - If page selected dirty, then write it back to disk
- Which page to select?
  - Replacement policies (LRU, MRU)



Disk Arrays

- Single disk becomes bottleneck
- Disk arrays
  - instead of single large disk
  - many small parallel disks
    - read N blocks in a single access time
    - concurrent queries
    - tables spanning among disks
- Redundant Arrays of Independent Disks (RAID)
  - 7 levels (0-6)
  - reliability
  - redundancy
  - parallelism

RAID Technology

- A natural solution is a large array of small independent disks acting as a single higher-performance logical disk.
- A concept called **data striping** is used, which utilizes parallelism to improve disk performance.
- Data striping distributes data transparently over multiple disks to make them appear as a single large, fast disk.

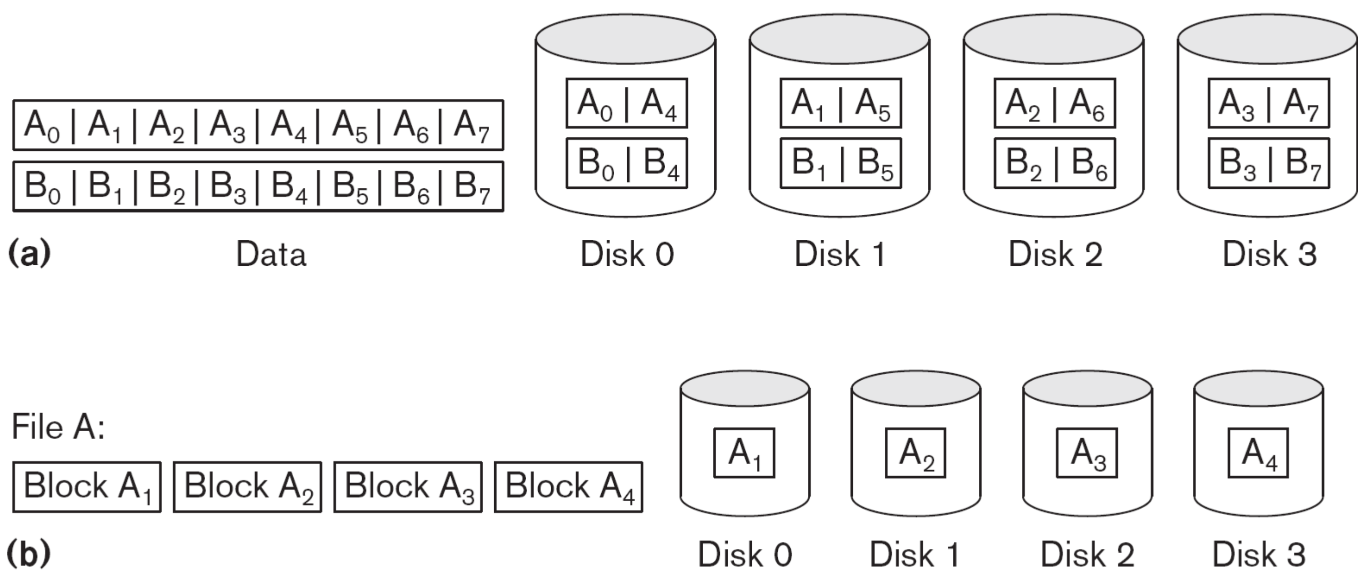


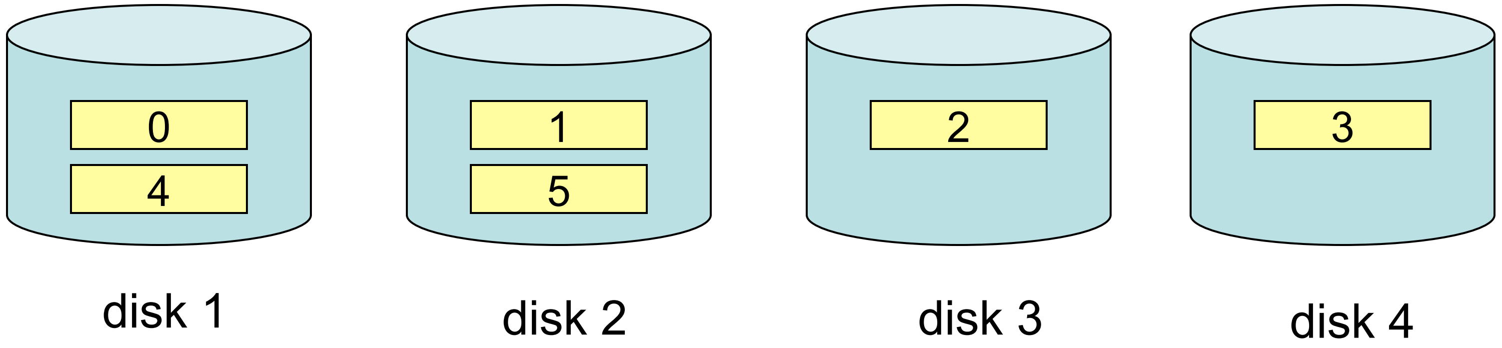
Figure 17.13
Striping of data across multiple disks. (a) Bit-level striping across four disks. (b) Block-level striping across four disks.

RAID Technology (cont.)

- Different raid organizations were defined based on different combinations of the two factors of granularity of data interleaving (striping) and pattern used to compute redundant information.
  - Raid level 0 has no redundant data and hence has the best write performance at the risk of data loss
  - Raid level 1 uses mirrored disks.
  - Raid level 2 uses memory-style redundancy by using Hamming codes, which contain parity bits for distinct overlapping subsets of components. Level 2 includes both error detection and correction.
  - Raid level 3 uses a single parity disk relying on the disk controller to figure out which disk has failed.
  - Raid Levels 4 and 5 use block-level data striping, with level 5 distributing data and parity information across all disks.
  - Raid level 6 applies the so-called P + Q redundancy scheme using Reed-Soloman codes to protect against up to two disk failures by using just two redundant disks.

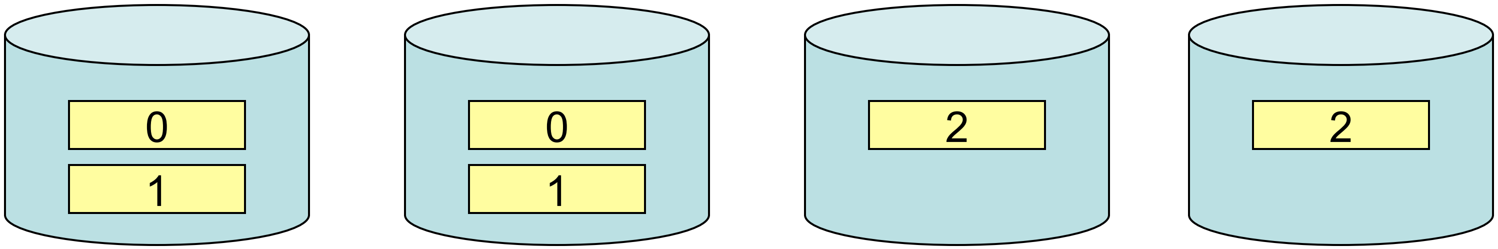
RAID level 0

- Block level striping
- No redundancy
- maximum bandwidth
- automatic load balancing
- best write performance
- but, no reliability



Raid level 1

- Mirroring
  - Two identical copies stored in two different disks
- Parallel reads
- Sequential writes
- transfer rate comparable to single disk rate
- most expensive solution



disk 1

disk 2
mirror of disk 1

disk 3

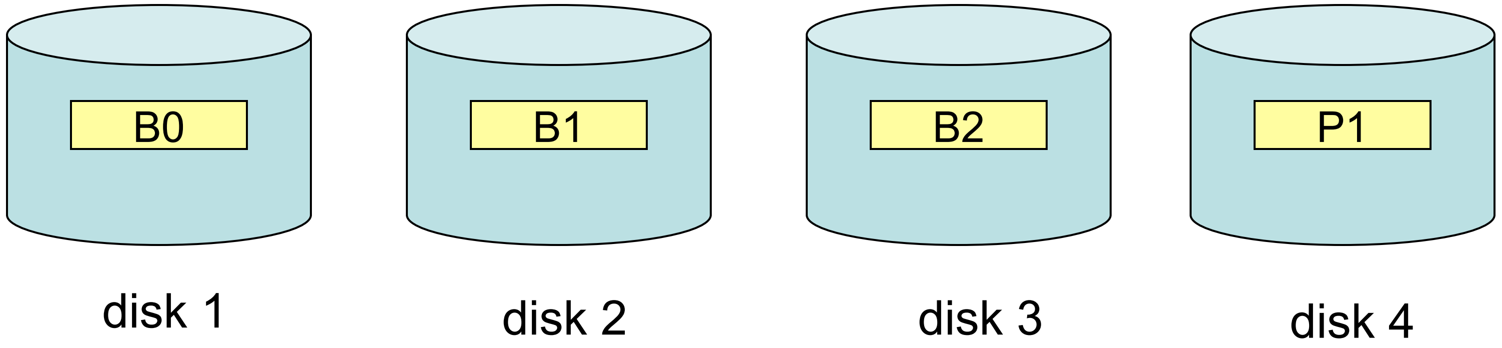
disk 4
mirror of disk 3

RAID levels 2 and 3

- bit level striping (next bit on a separate disk)
- error detection and correction
- RAID 2
  - ECC error correction codes (Hamming code)
  - Bit level striping, several parity bits
- RAID 3
  - Byte level striping, single parity bit
  - error detection by disk controllers (hardware)
- RAID 4
  - Block level striping, single parity bit

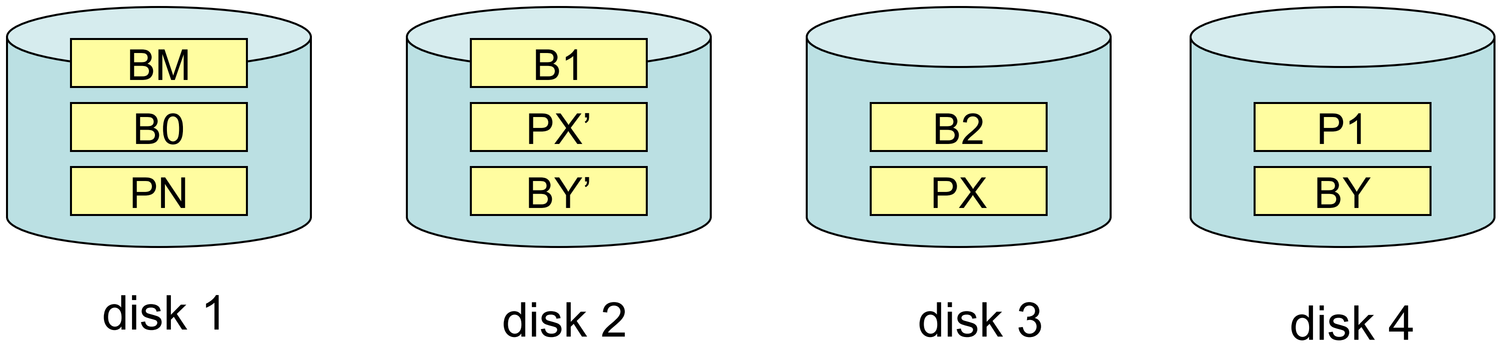
RAID level 4

- block level striping
- parity block for each block in data disks
  - P1 = B0 **XOR** B1 **XOR** B2
  - B2 = B0 **XOR** B1 **XOR** P1
- an update:
  - P1' = B0' **XOR** B0 **XOR** P1 (every update -> must write parity disk)



RAID level 5 and 6

- subsumes RAID 4
- parity disk not a bottleneck
  - parity blocks distributed on all disks
- RAID 6
  - tolerates two disk failures
  - P+Q redundancy scheme
    - 2 bits of redundant data for each 4 bits of data
  - more expensive writes



What pages contain logically?

- Files:
  - Physically partitioned into pages (or blocks)
  - Logically partitioned into **records**
- Each file is a sequence of records
- Each record is a sequence of fields

student	
cid	name
00112233	Paul

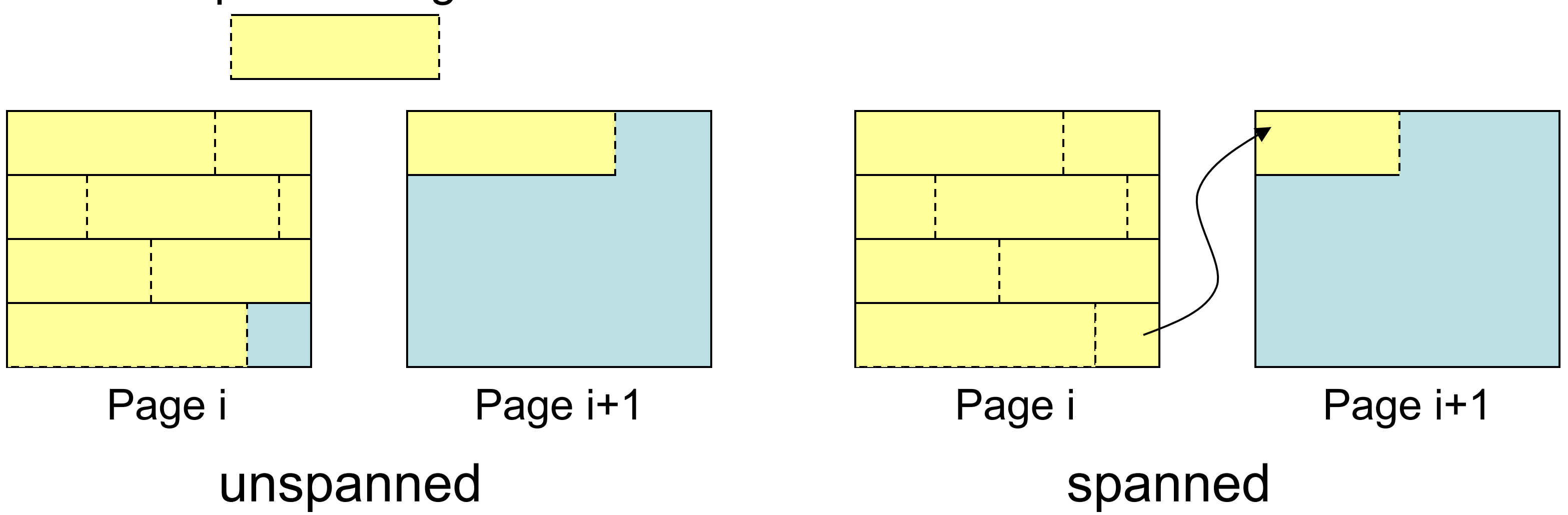
student record:

00112233	Paul

8 + 4 = 12 Bytes

Page Organization

- Student record size: 12 Bytes
- Typical page size: 2 KB
- Record identifiers: <Page identifier, offset>
- How records are distributed into pages:
  - Unspanned organization
    - Blocking factor = $\left\lfloor \frac{pagesize}{recordsize} \right\rfloor$
  - Spanned organization



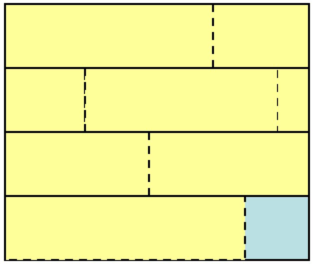
Blocking

- **Blocking**:
  - Refers to storing a number of records in one block on the disk.
- Blocking factor (**bf**) refers to the number of records per block.
- There may be empty space in a block if an integral number of records do not fit in one block.
- File records can be **unspanned** or **spanned**
  - **Unspanned**: no record can span two blocks
  - **Spanned**: a record can be stored in more than one block

The physical disk blocks that are allocated to hold the records of a file can be *contiguous, linked, or indexed*.

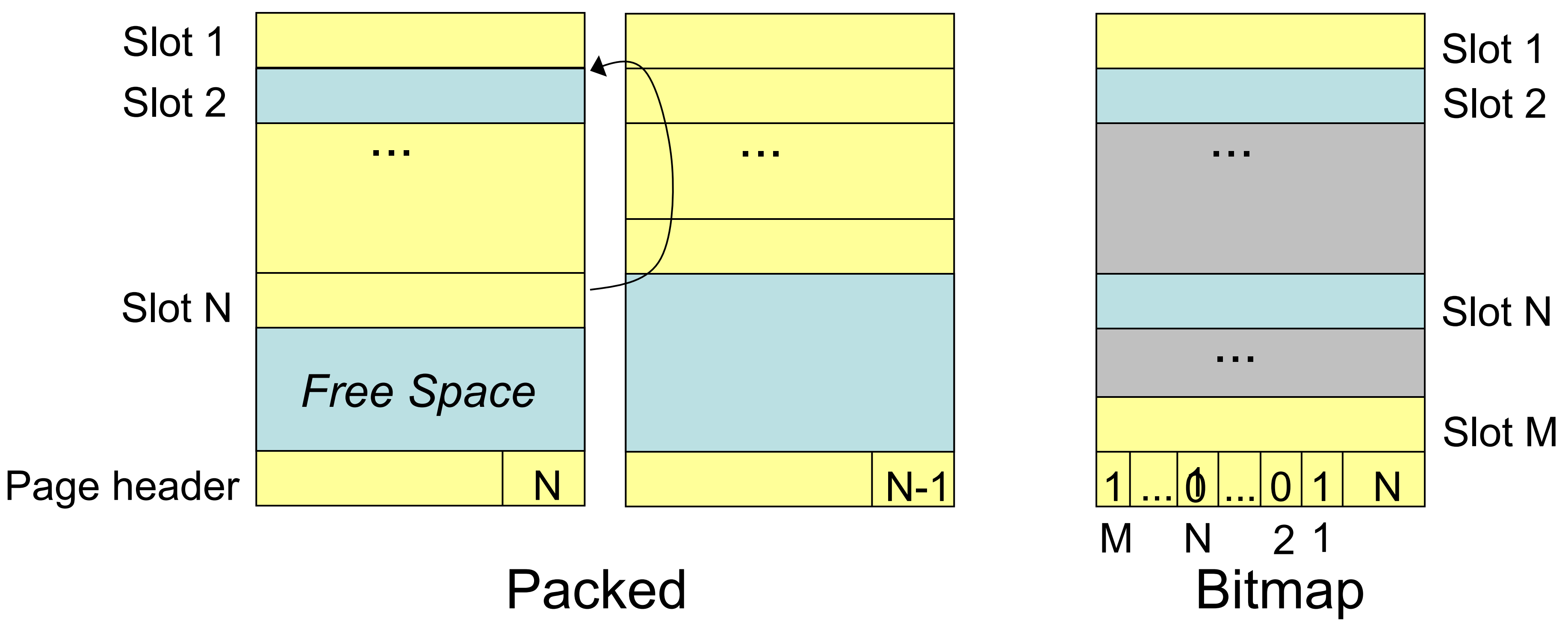
What if a record is deleted?

- Depending on the type of records:
  - Fixed-length records
  - Variable-length records



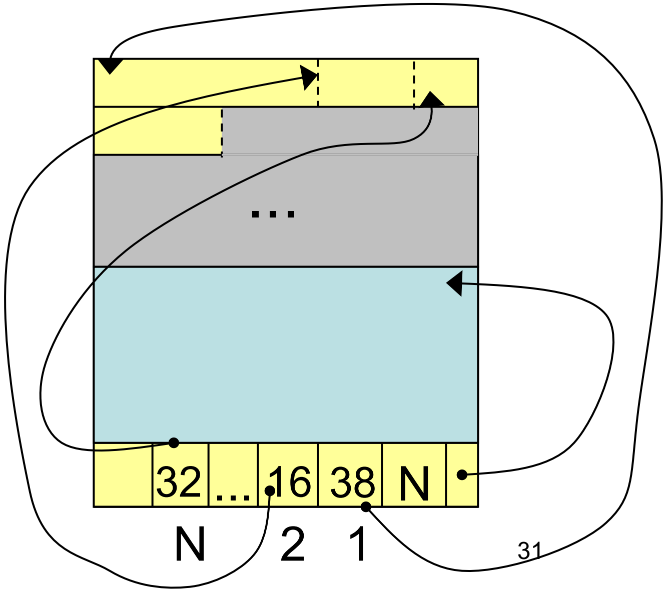
Fixed-length record files

- Upon record deletion:
  - Packed page scheme
  - Bitmap



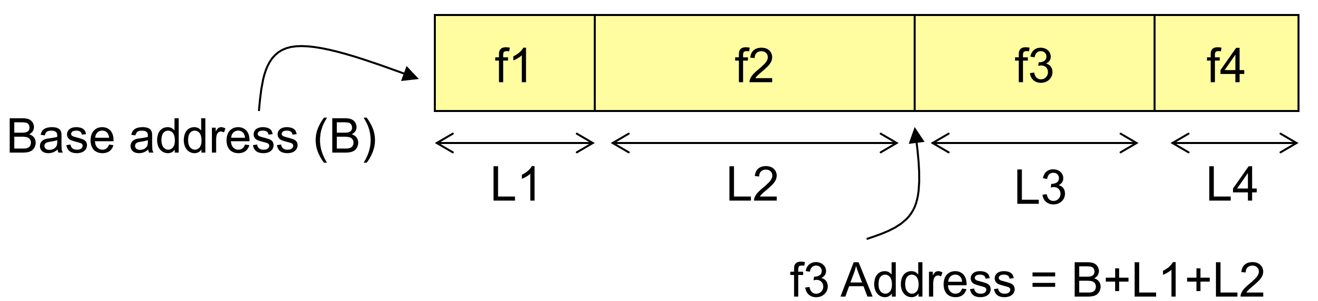
Variable-length record files

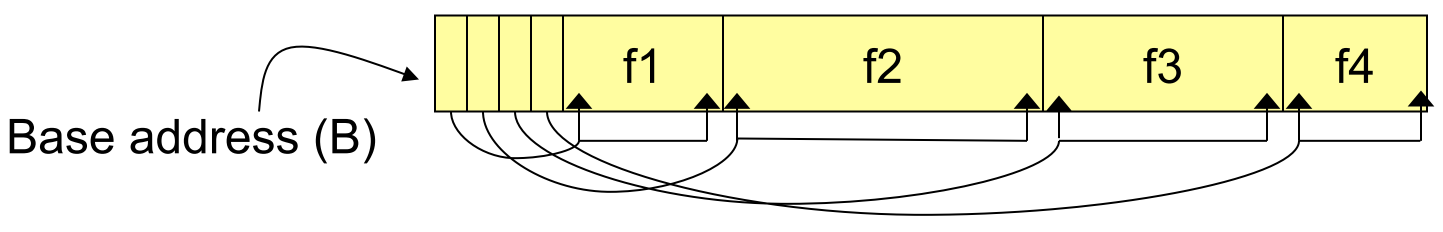
- When do we have a file with variable-length records?
  - Column datatype: variable length
  - create table t (field1 int, field2 varchar2(n))
- Problems:
  - Holes created upon deletion have variable size
  - Find large enough free space for new record
- Could use previous approaches: maximum record size
  - a lot of space wasted
- Use slotted page structure
  - Slot directory
  - Each slot storing offset, size of record
  - Record IDs: page number, slot number



Record Organization

- Fixed-length record formats
  - Fields stored consecutively
- Variable-length record formats
  - Array of offsets
  - NULL values when start offset = end offset





Operation on Files

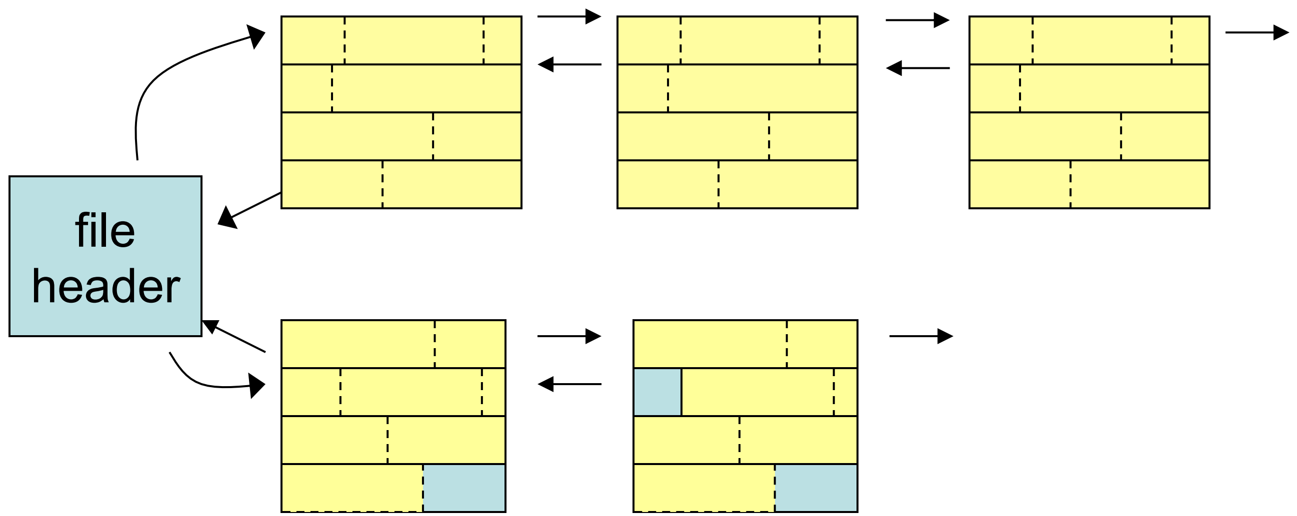
- Typical file operations include:
  - **OPEN**: Prepares the file for access, and associates a pointer that will refer to a *current* file record at each point in time.
  - **FIND**: Searches for the first file record that satisfies a certain condition, and makes it the current file record.
  - **FINDNEXT**: Searches for the next file record (from the current record) that satisfies a certain condition, and makes it the current file record.
  - **READ**: Reads the current file record into a program variable.
  - **INSERT**: Inserts a new record into the file & makes it the current file record.
  - **DELETE**: Removes the current file record from the file, usually by marking the record to indicate that it is no longer valid.
  - **MODIFY**: Changes the values of some fields of the current file record.
  - **CLOSE**: Terminates access to the file.
  - **REORGANIZE**: Reorganizes the file records.
    - For example, the records marked deleted are physically removed from the file or a new organization of the file records is created.
  - **READ_ORDERED**: Read the file blocks in order of a specific field of the file.

File Organization
(later we study it in a more detailed way)

- Heap files: unordered records
- Sorted files: ordered records
- Hashed files: records partitioned into buckets

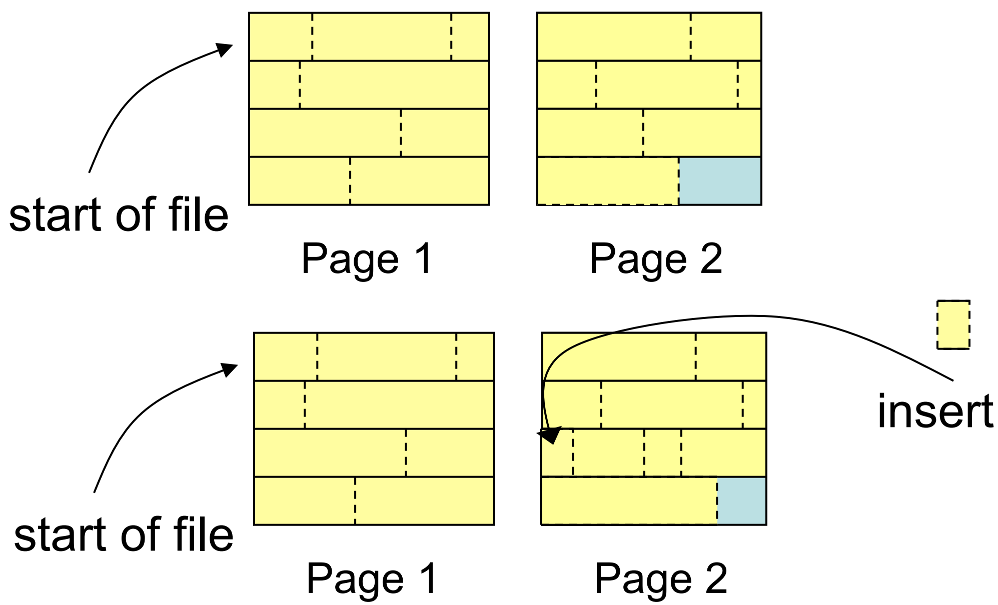
Heap Files

- Simplest file structure
- Efficient insert
- Slow search and delete
  - Equality search: half pages fetched on average
  - Range search: all pages must be fetched



Sorted (Ordered) files

- Sorted records based on **ordering field**
  - If ordering field same as key field, **ordering key field**
- Slow inserts and deletes
- Fast logarithmic search

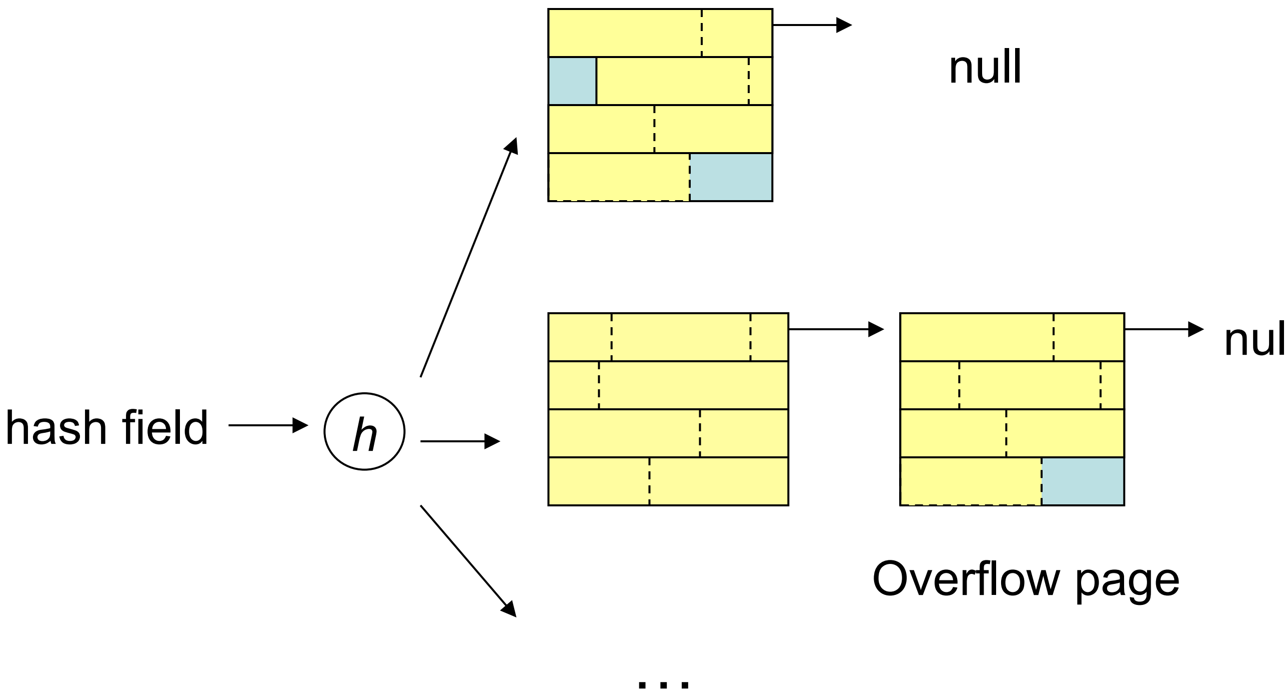


Sorted (Ordered) Files

- Also called a **sequential** file.
- File records are kept sorted by the values of an *ordering field*.
- Insertion is expensive: records must be inserted in the correct order.
  - It is common to keep a separate unordered *overflow* (or *transaction*) file for new records to improve insertion efficiency; this is periodically merged with the main ordered file.
- A **binary search** can be used to search for a record on its *ordering field* value.
  - This requires reading and searching $\log_2$ of the file blocks on the average, an improvement over linear search.
- Reading the records in order of the ordering field is quite efficient.

Hashed Files

- Hash function *h* on **hash field** distributes pages into buckets
- Efficient equality searches, inserts and deletes
- No support for range searches



Hashed Files

- Hashing for disk files is called **External Hashing**
- The file blocks are divided into M equal-sized **buckets**, numbered $bucket_0$, $bucket_1$, ..., $bucket_{M-1}$.
  - Typically, a bucket corresponds to one (or a fixed number of) disk block.
- One of the file fields is designated to be the **hash key** of the file.
- The record with hash key value K is stored in bucket i, where i=h(K), and h is the **hashing function**.
- Search is very efficient on the hash key.
- Collisions occur when a new record hashes to a bucket that is already full.
  - An overflow file is kept for storing such records.
  - Overflow records that hash to each bucket can be linked together.

Hashed Files

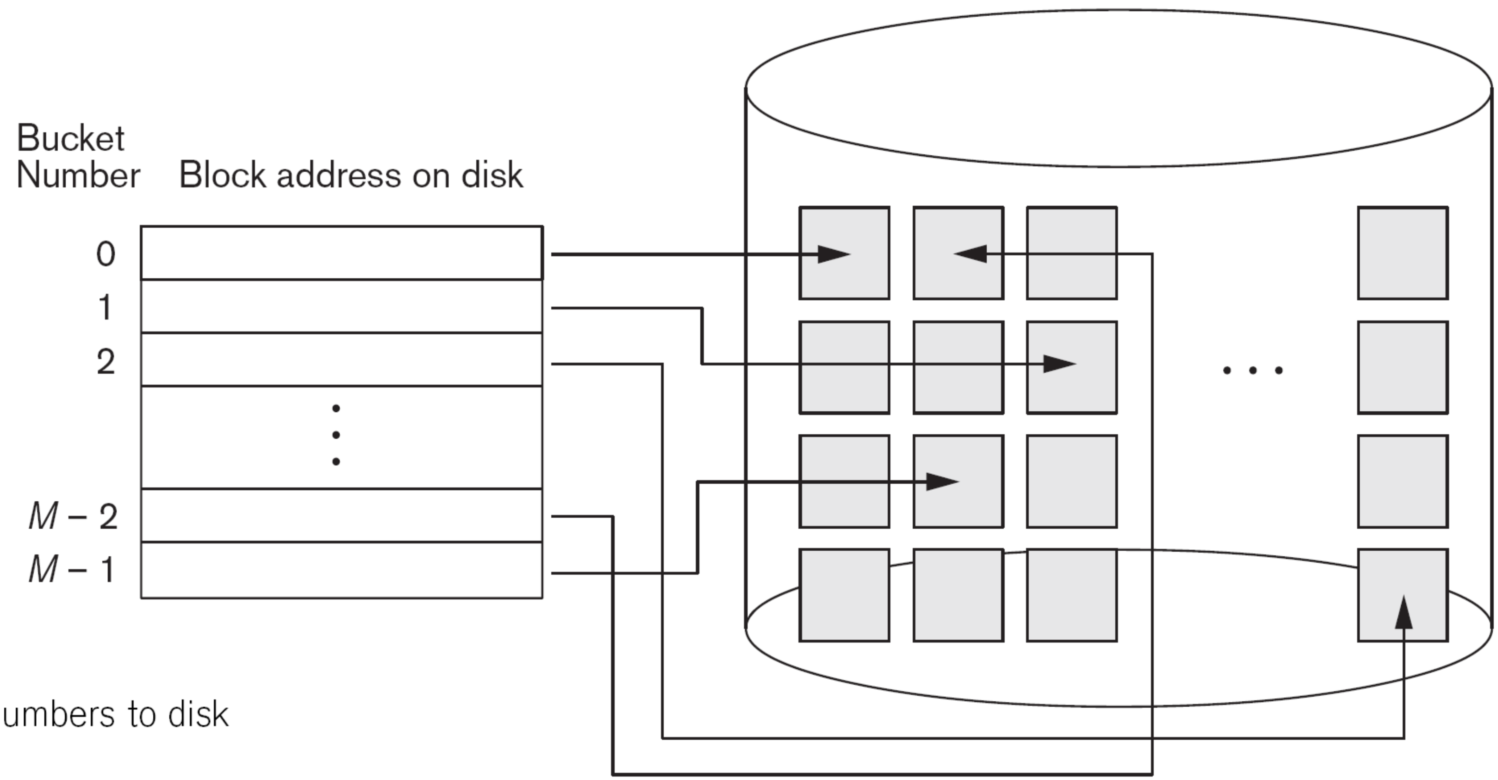


Figure 17.9
Matching bucket numbers to disk block addresses.

Summary (1/2)

- Why Physical Storage Organization?
  - understanding low-level details which affect data access
  - make data access more efficient
- Primary Storage, Secondary Storage
  - memory fast
  - disk slow but non-volatile
- Data stored in files
  - partitioned into pages physically
  - partitioned into records logically
- Optimize I/Os
  - scheduling algorithms
  - RAID
  - page replacement strategies

Summary (2/2)

- File Organization
  - how each file type performs
- Page Organization
  - strategies for record deletion
- Record Organization