Chapter 11: Storage and File Structure

- Overview of Physical Storage Media
- RAID
- File Organization
- Organization of Records in Files
- Data-Dictionary Storage

Classification of Physical Storage Media

– Speed with which data can be accessed

– Cost per unit of data

– Reliability

- data loss on power failure or system crash
- physical failure of the storage device

– Volatility

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

Physical Storage Media

- **Cache** – fastest and most costly form of storage; volatile; managed by the computer system hardware.
- **Main memory**:
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
    - capacities of up to a few Gigabytes widely used currently
  - It is **volatile** as contents of main memory are lost if a power failure or system crash occurs.

Physical Storage Media (Cont.)

- **Flash memory**
  - also known as EEPROM (Electrically Erasable Programmable Read-Only Memory)
  - Data survives even power failure.
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number of write/erase cycles.
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory.
  - Cost per unit of storage roughly similar to main memory
  - Widely used in embedded devices such as digital cameras

Physical Storage Media (Cont.)

- **Magnetic-disk**
  - Primary medium for the long-term storage of data; typically stores entire database.
  - Data must be moved from disk to main memory for access, and written back for storage.
    - Much slower access than main memory
    - **direct-access** – possible to read data on disk in any order

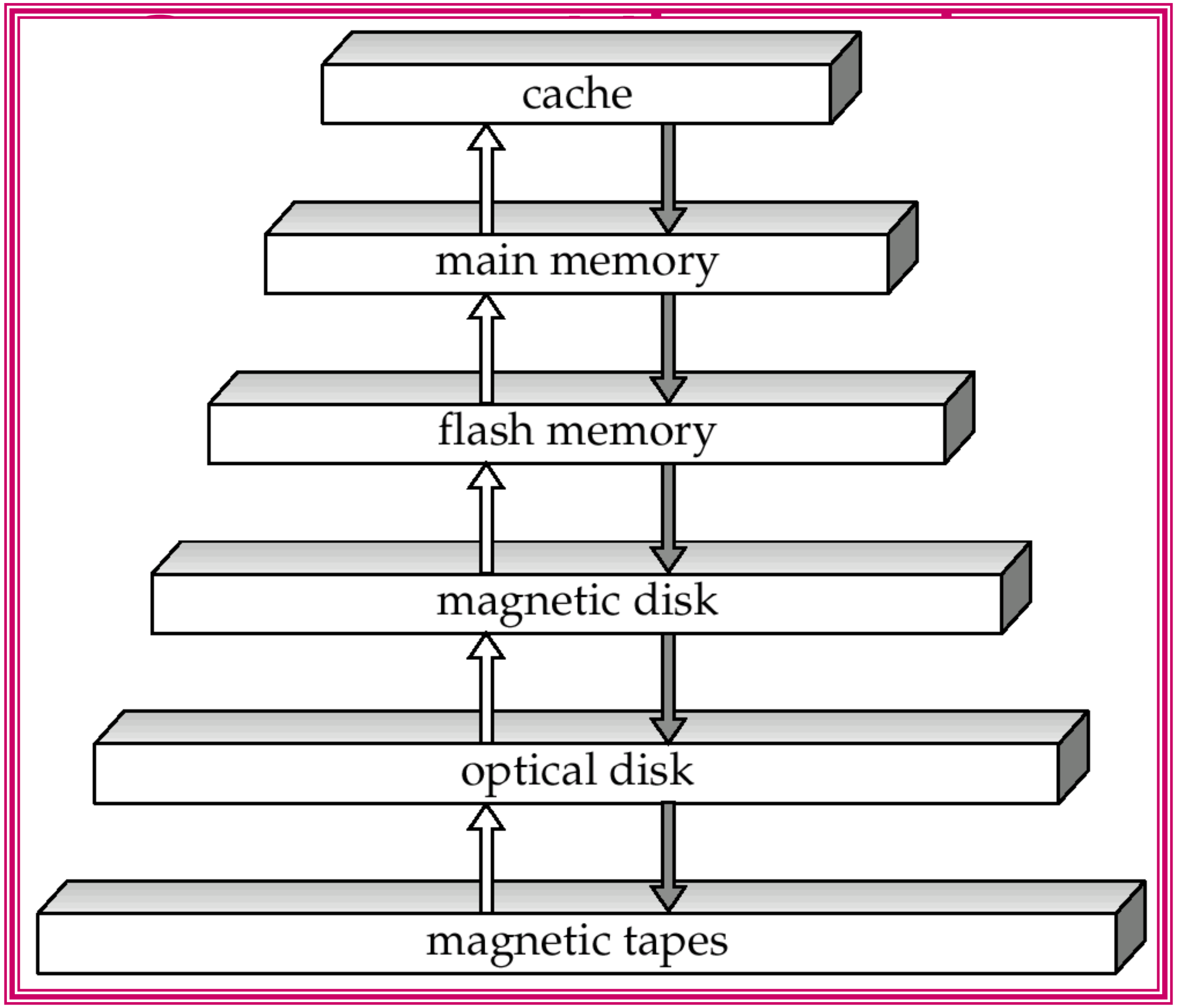
    Capacities range up to roughly several hundreds GB currently
  - Survives power failures and system crashes
    - disk failure can destroy data, but is very rare

Physical Storage Media (Cont.)

- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R and DVD-R)
  - Multiple write versions also available (CD-RW, DVD-RW)
  - Reads and writes are slower than with magnetic disk

Physical Storage Media (Cont.)

- **Tape storage**
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **sequential-access** – much slower than disk
  - very high capacity (40 to 300 GB tapes available)


cache
main memory
flash memory
magnetic disk
optical disk
magnetic tapes

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

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, and
  - high reliability by storing data redundantly, so that data can be recovered even if a disk fails

Improvement of Reliability via Redundancy

- If only one copy of data is stored, disk failure will result in significant loss of data.
- 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 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 10101010 1110011**
– 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.
  - Bit level striping is not used much any more

- **Block-level striping**
  - splits blocks across multiple disks
  - with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1; it uses [i/n]th physical block to store logical block I
  - E.g. with 8 disks, logical block 0 is stored in physical block 0 of disk 1.
  - E.g. logical block 11 is stored in physical block 1 of disk 4.
  - For reading large file, n blocks can be read in parallel from n disks giving high data rate.
  - For single block, data rate is same as single disk but remaining n-1 disk can perform other task.
  - 6 disk 0 1,2,3,4,5
  - n =6 i= 1^{st} block (1 mod 6) = 6 mod 6 = 0

RAID Levels

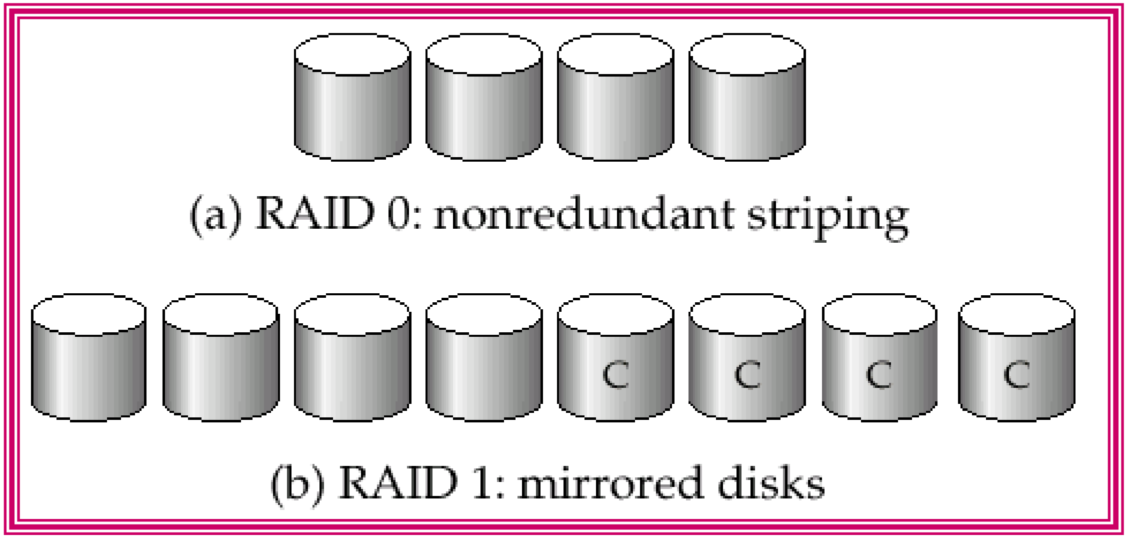
– Mirroring: high reliability but expensive

– Striping: high transfer rate but does not improve reliability.

– 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 lost is not critical.



(a) RAID 0: nonredundant striping

(b) RAID 1: mirrored disks

- **RAID Level 1**: redundant
  - Data are stored twice by writing them to both the data drive (or set of data drives) and a mirror drive (or set of drives)
  - It have Mirrored disks with block striping
- Advantage
  - Offers best write performance.
  - Popular for applications such as storing log files in a database system.
  - Easy to implement.
  - In case a drive fails, data do not have to be rebuild, they just have to be copied to the replacement drive

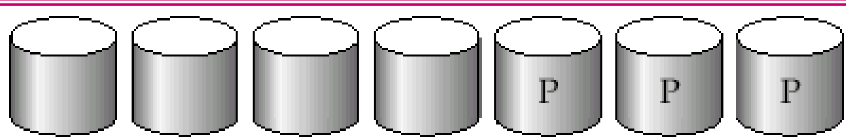
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping. 1101100 111000
- Each byte in a memory system may have a **parity bit**
  - – Memory system have parity bit for error detection and correction.
  - – Each byte has parity bit and even parity scheme is followed.
  - – The idea of error-correcting codes can be used directly in disk arrays bystriping bytes across disks.
  - – For example, the first bit of each byte could be
  - – stored in disk 1, the second bit in disk 2, and so on until the eighth bit is
  - – stored in disk 8, and the error-correction bits are stored in further disks

RAID Levels (Cont.)

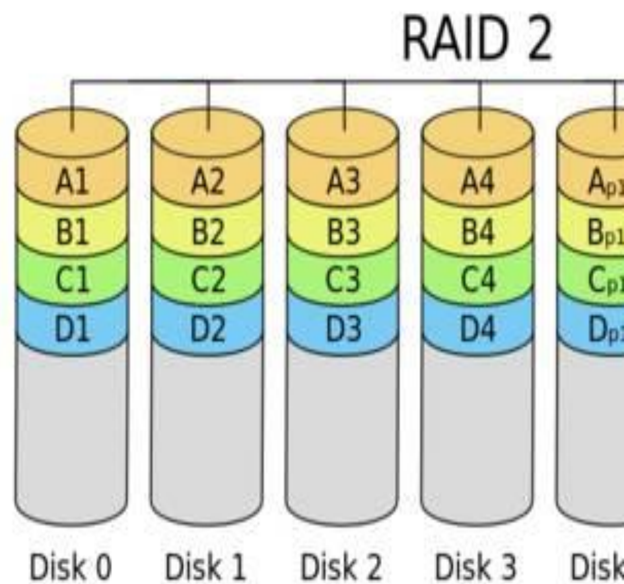
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping. 1111 1111 101
  - Each byte has parity bit and even parity scheme is followed.
  - If one of bit change, it will not match with stored parity. Thus detecting one bit error.
  - For error correction, two or more extra bits are needed to reconstruct single bit if damaged.
  - Striping bytes across disks ie first bit of byte on one disk, 2^{nd} bit on other disk …Error correcting bits are stored in further disks.
  - If one disk fails, remaining bits of byte and associated error correcting bits is used to reconstruct data
  - It requires three disk overhead instead of four disks as in level 1

- P indicates it store errorcorrection bits. 1 ,2
- If one of the disks fails, the remaining bits of the byte and the
- associated error-correction bits can be read from other disks, and can be used to reconstruct the damaged data
- . Error-correcting schemes store 2 or more extra bits, and can reconstruct the data if a single bit gets damaged.

- Disadvantage
  - Partiy disk overhead

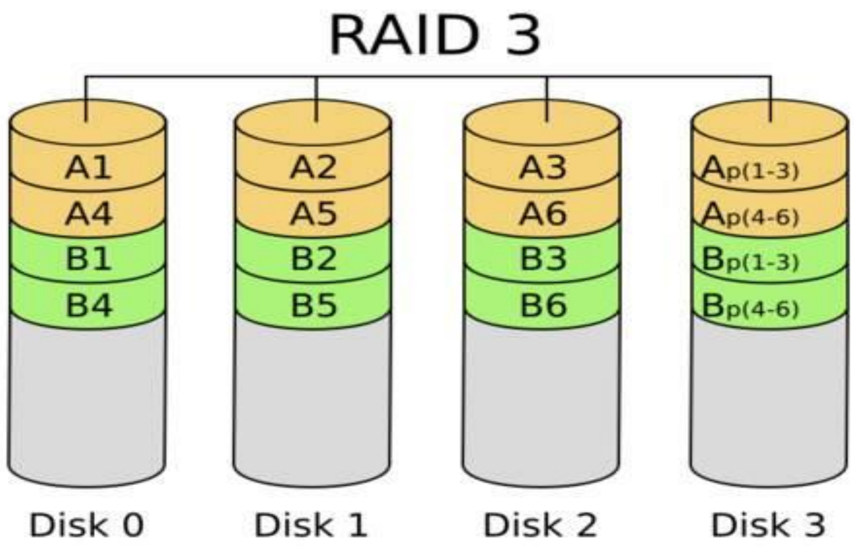


(c) RAID 2: memory-style error-correcting codes

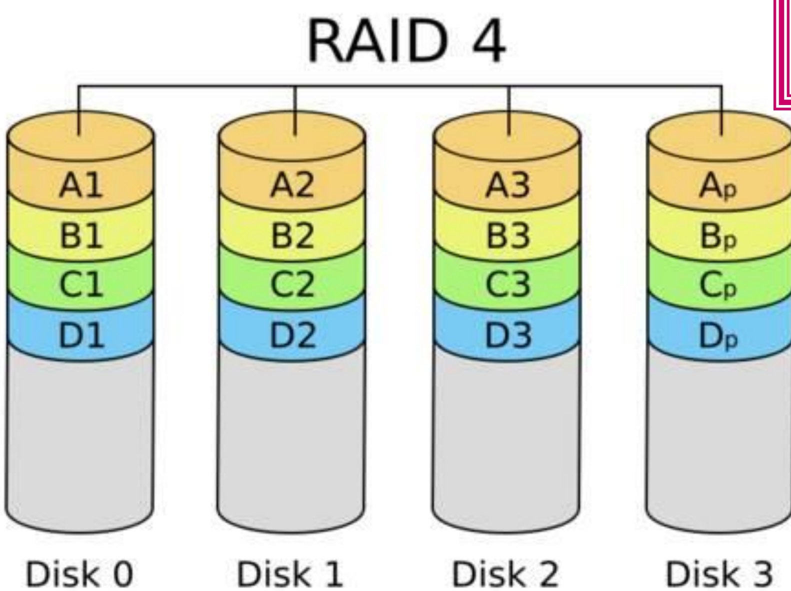


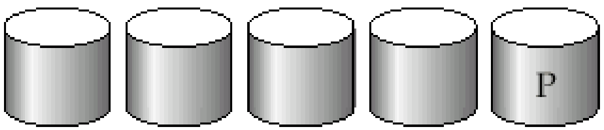
RAID Levels (Cont.)

- **RAID Level 3**: Bit-Interleaved Parity
  - Use single bit parity for error detection and correction both. 1111 1111 1
  - If one byte gets damaged, system knows which sector it is. Each bit in sector is constructed to 0 or 1 by using corresponding bits from byte in other disks. If parity of remaining bits is equal to stored parity, missing bit is 0 otherwise 1.
  - Benefits over level 1 as less number of disks required.
  - As bits are stripped transfer rate for reading and writing a single block becomes N times faster
  - A problem is the disk used for calculating checksums – which is bottleneck in the performance of the entire array.


RAID 3
A1
A4
B1
B4
A2
A5
B2
B5
A3
A6
B3
B6
Ap(1-3)
Ap(4-6)
Bp(1-3)
Bp(4-6)
Disk 0
Disk 1
Disk 2
Disk 3

- **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.
  - If one disk fails, parity block can be used with corresponding blocks from other disks to restore block.





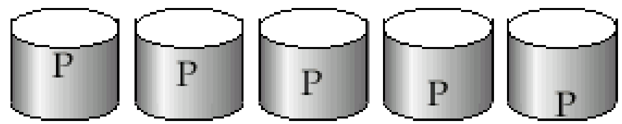
(e) RAID 4: block-interleaved parity

RAID Levels (Cont.)

- **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 Pn for logical blocks 4n, 4n+1,4n+2, 4n+3 is stored in disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks.

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

- A



(f) RAID 5: block-interleaved distributed parity

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost of extra disk storage requirement
  - Performance: Number of I/O operations
  - Performance during failure
  - Performance during rebuild of failed disk
- RAID 0 is used only when data safety is not important
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement

- Level 5 is preferred for applications with low update rate (bcoz four disk block access is required), but large amounts of data
- Level 1 is preferred for all other applications

File Organization

– The database is stored as a collection of *files*. Each file is a logically organized as sequence of *records.* A record is a sequence of fields.

– Records are mapped to disk blocks. Blocks are of fixed size but records are not.

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
- Deletion of record *I*: alternatives*:*

  1 move records *i* + 1, . . ., mto *i*

  2 move record *m* to *I*

  3 Leave space occupied by next insertion

  4 do not move records, but l *list*

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

A102	peridage	300	a103	roundhill

File of Figure 11.6, with Record 2 Deleted and All Records Moved

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

With Record 2 deleted and Final Record Moved

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 8	A-218	Perryridge	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600

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 rec

– Del ch is cal

header				
record 0	A-102	Perryridge	400	
record 1				
record 2	A-215	Mianus	700	
record 3	A-101	Downtown	500	
record 4				
record 5	A-201	Perryridge	900	
record 6				
record 7	A-110	Downtown	600	
record 8	A-218	Perryridge	700	

#

Free Lists (Contd…)

- On insertion we use record pointed by header. Change header pointer to point to next available address.
- If not add new record at end of file.
- Insertion and deletion is easy.

header				
record 0	A-102	Perryridge	400	
record 1				
record 2	A-215	Mianus	700	
record 3	A-101	Downtown	500	
record 4				
record 5	A-201	Perryridge	900	
record 6				
record 7	A-110	Downtown	600	
record 8	A-218	Perryridge	700	

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.
  - Record types that allow repeating fields (used in some older data models).
- Byte string representation
  - A simple method for implementing variable-length records is to attach a special endof-record (⊥) symbol to the end of each record.
  - Attach an *end-of-record* (⊥) control character to the end of each record. Store each record as string of consecutive bytes

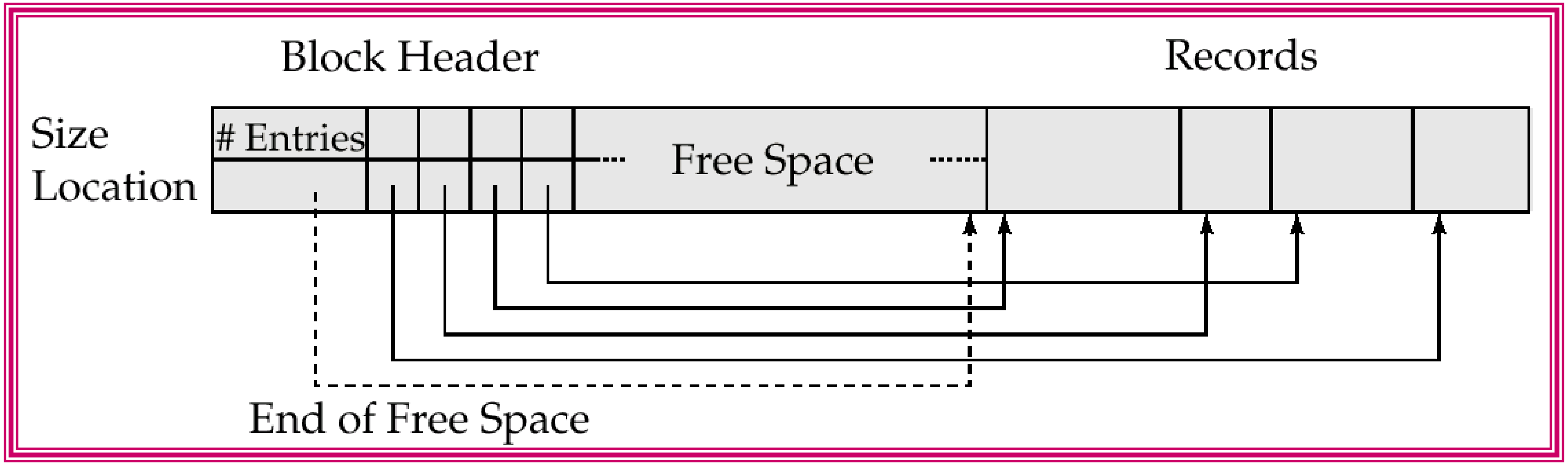
Variable-Length Records

- Disadvantages of byte-string representation.
- Difficulty with deletion as deleted records will generate large number of small fragments in memory
- Difficulty with growth. Such record must be moved and it is costly.

0	Perryridge	A-102	400	A-201	900	A-218	700	⊥
1	Round Hill	A-305	350	⊥				
2	Mianus	A-215	700	⊥				
3	Downtown	A-101	500	A-110	600	⊥		
4	Redwood	A-222	700	⊥				
5	Brighton	A-217	750	⊥				

Modified Byte String Representation: Slotted Page Structure

- Modified Byte String is commonly used for organizing records within a single block.
- The slotted-page structure appears in the beginning of each block
- Slotted page header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Actual records are allocated contiguously in block starting from the end of block
- Free space is contiguous between final entry in header and first record.


Block Header
Records
Size
Location
Entries
Free Space
End of Free Space

Modified Byte String Representation: Slotted Page Structure (Contd…)

- If record is inserted, space is allocated for it at end of free space and entry containing its size and location is added to header

- If record is deleted, space occupied by it is freed and its entry is set to -1. Records are moved and all free space is again between last entry and first record. End of free space pointer in header is updated appropriately.

- Records can be grown or shrunk as long as there is space in block

Variable-Length Records (Cont.)

- Fixed-length representation:
  - reserved space
  - Pointers or List representation
- Reserved space
  - can use fixed-length records of a known maximum length; unused space in shorter records filled with a null or end-of-record symbol.
  - Useful when most record length close to maximum.

0	Perryridge	A-102	400	A-201	900	A-218	700
1	Round Hill	A-305	350	⊥	⊥	⊥	⊥
2	Mianus	A-215	700	⊥	⊥	⊥	⊥
3	Downtown	A-101	500	A-110	600	⊥	⊥
4	Redwood	A-222	700	⊥	⊥	⊥	⊥
5	Brighton	A-217	750	⊥	⊥	⊥	⊥

- Id int(2) , name varchar(50) ,city varchar(50)
- 4 byte + 50 byte + 50 byte = 104 bytes

Pointer Method

- Pointer method
  - A variable-length record is represented by a list of fixed-length records, chained together via pointers.
  - Can be used even if the maximum record length is not known

0	Perryridge	A-102	400	
1	Round Hill	A-305	350	
2	Mianus	A-215	700	
3	Downtown	A-101	500	
4	Redwood	A-222	700	
5		A-201	900	
6	Brighton	A-217	750	
7		A-110	600	
8		A-218	700	

Pointer Method (Cont.)

– Disadvantage to pointer structure; space is wasted in all records except the first in a a chain.

– Solution is to allow two kinds of block in file:

– Anchor block – contains the first records of chain

– Overflow block – contains records other than those that are the first records of chains.

Organization of Records in Files

– Heap – a record can be placed anywhere in the file where there is space. Typically single file for each relation

– 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

- Generally records of each relation is stored in a separate file. In a **clustering file organization** records of several different relations can be stored in the same file
  - store related records on the same block to minimize I/O

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
- Search key is any attribute not necessary primary key
- Pointer in each record points to next record in search key order.
- E.g records are stored in search key order of branch-name

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Sequential File Organization (Cont.)

– Deletion – use pointer chains
– Insertion –locate record that comes before record to be inserted in search key order
  • 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
– If few records in overflow Block, it can be maintained
– With more records, file must be **reorganized** to physical sequential order.

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

A-888	North Town	800	

Clustering File Organization

- Simple file structure stores each relation in a separate file
- Can instead store several relations in one file using a **clustering** file organization
- E.g., clustering organization of *customer* and *depositor:*

Hayes	Main	Brooklyn
Hayes	A-102	
Hayes	A-220	
Hayes	A-503	
Turner	Putnam	Stamford
Turner	A-305	

– good for queries involving depositor ⋈ customer, and for queries involving one single customer and his accounts
– bad for queries involving only customer eg select * from customer;

Data Dictionary Storage

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

- Information about relations
  - names of relations
  - names and types of attributes of each relation
  - Domain and length of attributes
  - names and definitions of views
  - integrity constraints
- User name and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
  - Method of storage of each relation (clustered or non clustered)

- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
    - operating system file name or
    - disk addresses of blocks containing records of the relation
- Information about indices.

End of Chapter