CS 2550 / Spring 2006
Principles of Database Systems

04 – Storage

Alexandros Labrinidis
University of Pittsburgh

Roadmap

- Overview of Physical Storage Media
- Magnetic Disks
- Introduction to RAID
- File Organization
- Organization of Records in Files

Physical Storage Media Taxonomy

- 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
- Can differentiate storage into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage**:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory.

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
    - Capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years)
  - **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs.

Physical Storage Media (Cont.)

- **Flash memory**
  - Data survives 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
  - But writes are slow (few microseconds), erase is slower
  - Cost per unit of storage roughly similar to main memory
  - Widely used in embedded devices such as digital cameras
  - also known as EEPROM (Electrically Erasable Programmable Read-Only Memory)

Magnetic Disks

- Data is stored on spinning disk, and read/written magnetically
- 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 (more on this later)
- **direct-access** – possible to read data on disk in any order, unlike magnetic tape
- Hard disks vs floppy disks
- Capacities range up to roughly 100 GB currently
  - Much larger capacity and cost/byte than main memory/flash memory
  - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
- Survives power failures and system crashes
  - disk failure can destroy data, but is very rare

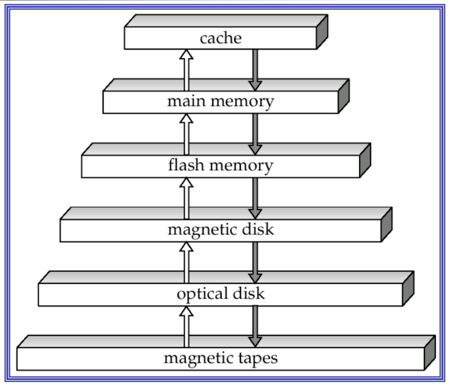
Physical Storage Media (Cont.)

- **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, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
  - **Juke-box** systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

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)
  - tape can be removed from drive $\Rightarrow$ storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available for storing massive amounts of data
    - hundreds of terabytes (1 terabyte = 10^9 bytes) to even a petabyte (1 petabyte = 10^{12} bytes)

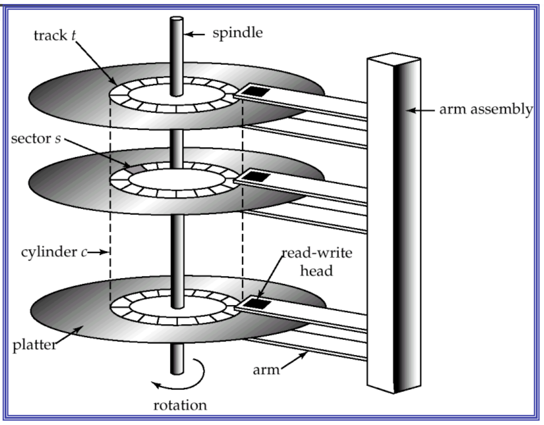
Storage Hierarchy



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

Magnetic Hard Disk Mechanism



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 16,000 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: 200 (on inner tracks) to 400 (on outer tracks)
- 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

Magnetic Disks (Cont.)

- Earlier generation disks were susceptible to head-crashes
  - Surface of earlier generation disks had metal-oxide coatings which would disintegrate on head crash and damage all data on disk
  - Current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted
- **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

Performance Measures of Disks

- Cost
- Size
- Access Time
- Data Transfer Rate
- Mean time to failure

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 (II)

- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 4 to 8 MB per second is typical
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. ATA-5: 66 MB/second, SCSI-3: 40 MB/s
    - Fiber Channel: 256 MB/s

Performance Measures of Disks (III)

- **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 30,000 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

Roadmap

- Overview of Physical Storage Media
- Magnetic Disks
- Introduction to RAID
- File Organization
- Organization of Records in Files

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
- 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

Reliability through 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

Performance through Parallelism

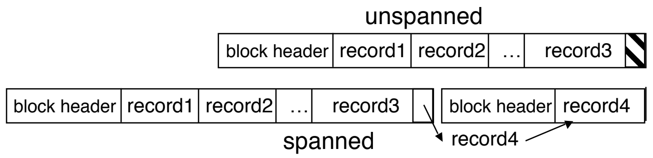
- Two main goals of parallelism in a disk system:
  1. Load balance multiple small accesses to increase throughput
  2. Parallelize large accesses to reduce response time.
- Improve transfer rate by striping data across multiple disks.
- **Bit-level striping** – split the bits of each byte across multiple 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
- **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

Roadmap

- Overview of Physical Storage Media
- Magnetic Disks
- Introduction to RAID
- File Organization
- Organization of Records in Files

Data Elements

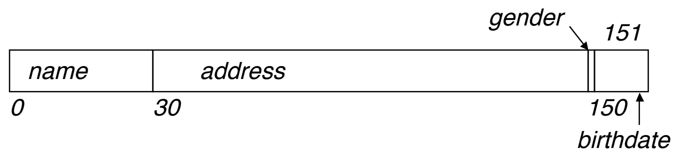
- **Field**: a database attribute (sequence of bytes)
- **Record**: sequence of fields (that describe an entity)
- **Block**: sequence of records
  - Unspanned (no record can span two blocks)
  - Spanned
- **File**: sequence of blocks



Fixed-Length Records

```
CREATE TABLE MovieStar (
  name      CHAR(30),
  address   CHAR(120),
  gender    CHAR(1),
  birthdate DATE
)
```

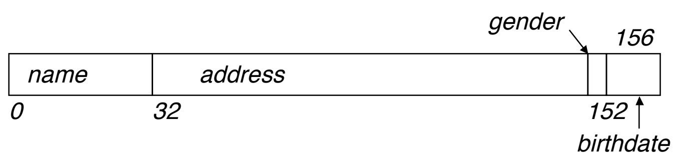
- Fields are stored in sequence as the corresponding attributes are declared
- DATE: 10-char string YYYY-MM-DD
  Fixed-length character string char(10)
  - example: 2002-09-15



Fixed-Length Records - Alignment

```
CREATE TABLE MovieStar (
  name        CHAR(30),
  address     CHAR(120),
  gender      CHAR(1),
  birthdate   DATE
)
```

- Each record within a block starts at a byte that is multiple of 4
- Each field within a record starts at a byte off-set from the beginning of the record that is multiple of 4



File Organization

```
CREATE TABLE deposit (
  account_number   CHAR(10),
  branch_name      CHAR(22),
  balance          REAL
)
```

- Fixed-length records
- 10 + 22 + 8 = 40 bytes

Record 0	A-102	Oakland	400
Record 1	A-305	Shadyside	350
Record 2	A-101	Downtown	700
Record 3	A-222	Squirrel Hill	500
Record 4	A-217	Shadyside	900
Record 5	A-110	Waterfront	340
Record 6	A-257	Oakland	600
Record 7	A-403	Downtown	250

File Organization – Updates I

Record 0	A-102	Oakland	400
Record 1	A-305	Shadyside	350
Record 3	A-222	Squirrel Hill	500
Record 4	A-217	Shadyside	900
Record 5	A-110	Waterfront	340
Record 6	A-257	Oakland	600
Record 7	A-403	Downtown	250

remove Record 2

Record 0	A-102	Oakland	400
Record 1	A-305	Shadyside	350
Record 3	A-222	Squirrel Hill	500
Record 4	A-217	Shadyside	900
Record 5	A-110	Waterfront	340
Record 6	A-257	Oakland	600
Record 7	A-403	Downtown	250
Record 8			

add Record 8

File Organization – Updates II

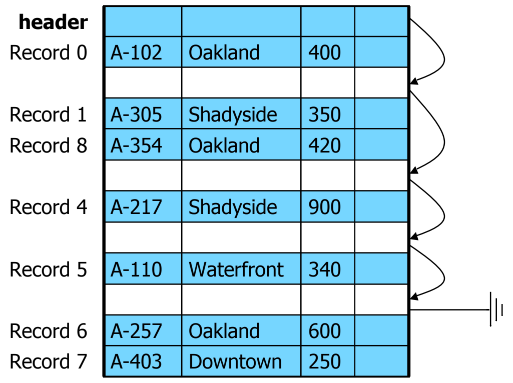
Record 0	A-102	Oakland	400
Record 1	A-305	Shadyside	350
Record 3	A-222	Squirrel Hill	500
Record 4	A-217	Shadyside	900
Record 5	A-110	Waterfront	340
Record 6	A-257	Oakland	600
Record 7	A-403	Downtown	250

remove Record 2

Record 0	A-102	Oakland	400
Record 1	A-305	Shadyside	350
Record 8	A-354	Oakland	420
Record 3	A-222	Squirrel Hill	500
Record 4	A-217	Shadyside	900
Record 5	A-110	Waterfront	340
Record 6	A-257	Oakland	600
Record 7	A-403	Downtown	250

add Record 8

File Organization – Free List



header				
Record 0	A-102	Oakland	400	
Record 1	A-305	Shadyside	350	
Record 8	A-354	Oakland	420	
Record 4	A-217	Shadyside	900	
Record 5	A-110	Waterfront	340	
Record 6	A-257	Oakland	600	
Record 7	A-403	Downtown	250	

Variable-Length Attributes

- example: type `VARCHAR(18)`
  - length + data:

10	L	A	B	R	I	N	I	D	I	S

  - null-terminated:

L	A	B	R	I	N	I	D	I	S	▮

  - maximum length:

L	A	B	R	I	N	I	D	I	S								

Fixed-length Records

- Fixed-length records cannot span separate blocks
- Variable-length fields are allocated their maximum length
- **Pros:**
  - fixed field length simplifies insertion, deletion etc
  - no space is needed for storing extra administrative info for the fields within record
  - equally fast access to all fields
  - every offset is pre-compiled and stored in DB Catalog
- **Cons:**
  - block internal fragmentation due to unspanned organization
  - record internal fragmentation (max specified length for every field)
  - more disk accesses for reading a given number of records

Variable-Length Records

- Typical in database systems because of:
  - Storage of multiple record types in a file
  - Variable-length attributes
  - Record types that allow repeating fields

Variable-Length Records
Byte-String Implementation

- Attach a special end-of-record symbol (▮) to the end of each record
- Alternatively, store record length at beginning of each record

- Disadvantages:
  - Not easy to reuse space which was occupied by a deleted record
  - No space for record to grow longer (must move record that needs to grow)

Variable-Length Records
Byte-String Implementation Example

Record 0	Oakland	A-102	400	A-205	300	A-104	320	▮
Record 1	Shadyside	A-305	350	▮				
Record 2	Downtown	A-101	700	▮				
Record 3	Squirrel Hill	A-222	500	A-323	450	▮		
Record 4	Shadyside	A-217	900	A-406	200	▮		

Variable-Length Records
Preceding length field Implementation

- Variable-length field store their length at the beginning
  - Trade-off between extra space and internal fragmentation
- Fields are stored in the order in which they are declared
  - For fixed-length the offset is stored in the catalog
  - For variable-length, the offset is computed from the heading
- Pros:
  - No record internal fragmentation
- Cons:
  - Access cost for a field is proportional to the distance from the beginning of the record
  - Null field (headers) must be there
  - Records cannot be fragmented over separate blocks – small tuples
- Optimization: pre-compile and store the offset of all preceding fixed-length fields in each variable-length field

Variable-Length Records
Fixed-length Implementation

- Use one or more fixed-length records to represent one variable-length record

- Reserved space – padding

- Linked list method

- Chained blocks via pointers
  - *Anchor block*: first records of a chain
  - *Overflow* block: subsequent records of a chain

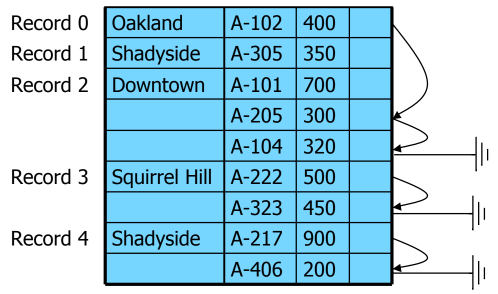
Variable-Length Records
Fixed-length Implementation Example – 1

- Reserved-space Method

Record 0	Oakland	A-102	400	A-205	300	A-104	320
Record 1	Shadyside	A-305	350	▮	▮	▮	▮
Record 2	Downtown	A-101	700	▮	▮	▮	▮
Record 3	Squirrel Hill	A-222	500	A-323	450	▮	▮
Record 4	Shadyside	A-217	900	A-406	200	▮	▮

Variable-Length Records
Fixed-length Implementation Example – 2

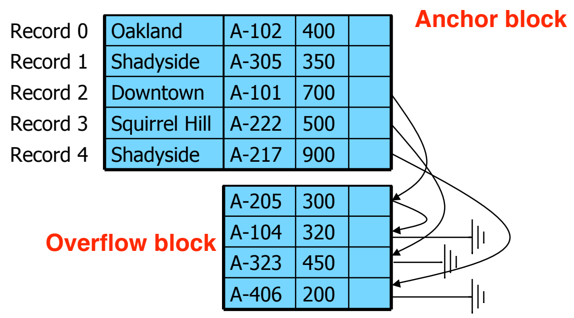
- Linked-List Method



Record 0	Oakland	A-102	400	
Record 1	Shadyside	A-305	350	
Record 2	Downtown	A-101	700	
		A-205	300	
		A-104	320	
Record 3	Squirrel Hill	A-222	500	
		A-323	450	
Record 4	Shadyside	A-217	900	
		A-406	200	

Variable-Length Records
Fixed-length Implementation Example – 3

- Chained-Blocks Method



Record 0	Oakland	A-102	400	
Record 1	Shadyside	A-305	350	
Record 2	Downtown	A-101	700	
Record 3	Squirrel Hill	A-222	500	
Record 4	Shadyside	A-217	900	

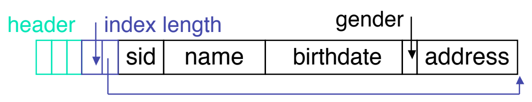
A-205	300
A-104	320
A-323	450
A-406	200

Variable-Length Records
Sequence of Self-Identifying Fields Implementation

- Each field is fully equipped with length indicator and internal id
- Pros:
  - Simple and most flexible of all mechanisms
    - no distinction between fixed and variable length fields.
    - no assumption on the attribute ordering in the catalog.
  - no need to store null.
  - easy extension.
  - no problem spanning records across blocks.
  - supports vertical fragmentation for load balancing.
- Cons:
  - No pre-compilation. Access to all fields is not equal.
  - access cost for a field is proportional to its distance from the beginning of the record.
  - null-values require the whole record to be scanned

Variable-Length Records
Prefix Pointers to the Fields Implementation

- A prefix array after the record heading, contains one pointer per field.
  - the difference between two successive pointers is the length of the field in the second one belongs to.
- Optimization: pointers are only kept for variable-length fields



- The pointer array serves as a rudimentary catalog for the record.
  - it works like the fixed-length field/fixed-length record scheme but with computation of the offset difference (no-precompilation).
- Cons: increase the size of records.

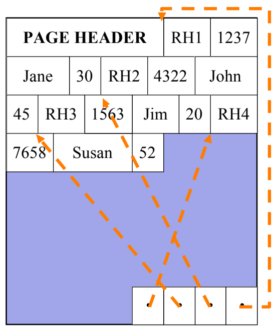
Variable-Length Records
Slotted-Page Implementation

- Slotted-page structure is industry standard
- records can move within the page
- records are allocated contiguously, starting at the beginning of the block (or starting at the beginning of the block)
- End of page: has pointers to records (or start of page: has pointers to records)
- external pointers point only to the header

Slotted Pages Example

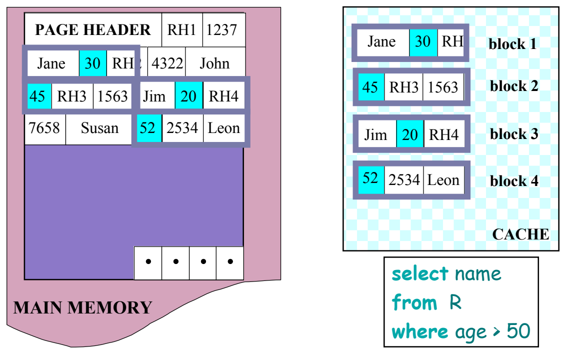
R

RID	SSN	Name	Age
1	1237	Jane	30
2	4322	John	45
3	1563	Jim	20
4	7658	Susan	52
5	2534	Leon	43
6	8791	Dan	37



- Records are stored sequentially
- Offsets to start of each record at end of page

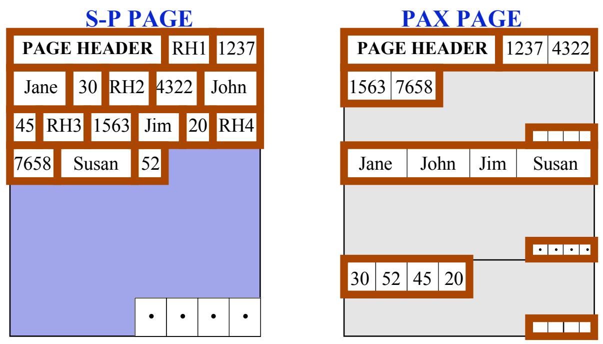
Predicate Evaluation using S-P



```
select name
from R
where age > 50
```

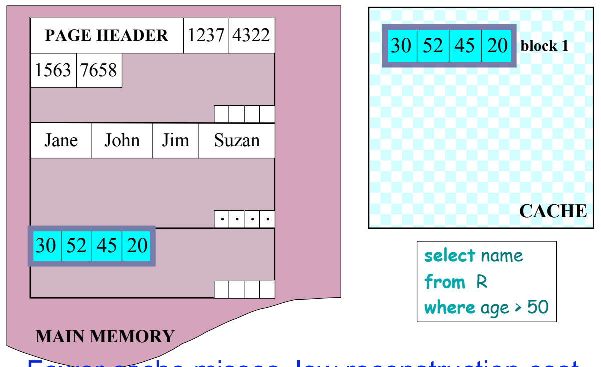
S-P push non-referenced data to the cache

Partition Attributes Across (PAX)
(by Natassa Ailamaki, CMU)



Partition data *within* the page for spatial locality

Predicate Evaluation using PAX
(by Natassa Ailamaki, CMU)



```
select name
from R
where age > 50
```

Fewer cache misses, low reconstruction cost

Roadmap

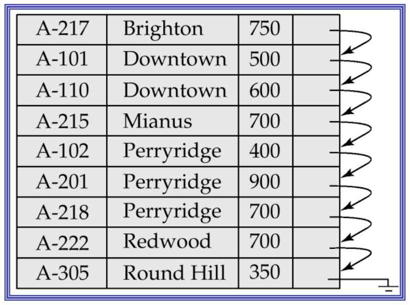
- Overview of Physical Storage Media
- Magnetic Disks
- Introduction to RAID
- File Organization
- Organization of Records in Files

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 **clustering file organization** records of several different relations can be stored in the same file
  - Why: store related records on the same block to minimize I/O

Sequential File Organization

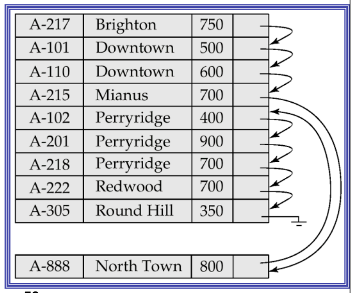
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



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.)

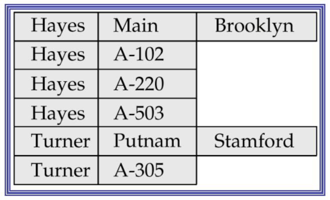
- **Deletion** – use pointer chains
- **Insertion** – locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time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



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:*

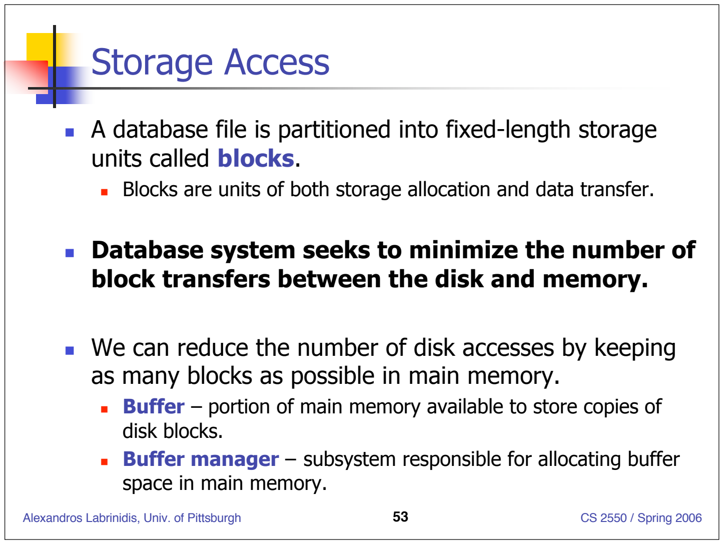


Hayes	Main	Brooklyn
Hayes	A-102	
Hayes	A-220	
Hayes	A-503	
Turner	Putnam	Stamford
Turner	A-305	

- ★ good for queries involving depositor ⋈ customer, and for queries involving one single customer and his accounts
- ★ bad for queries involving only customer
- ★ results in variable size records

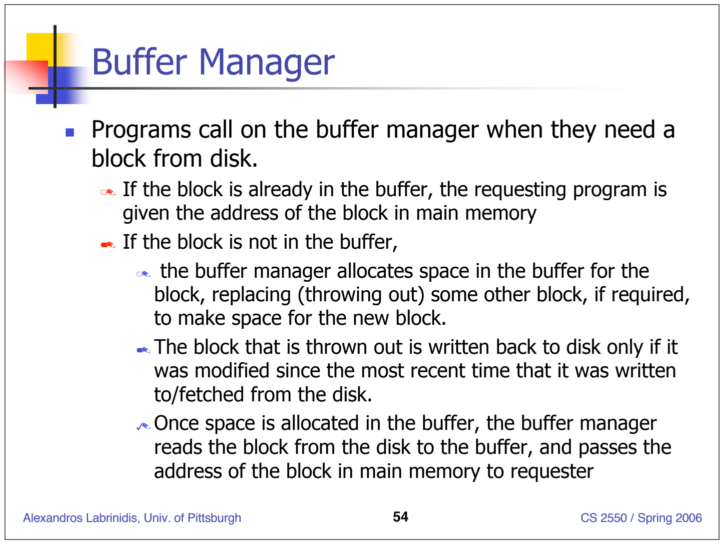
Roadmap

- Overview of Physical Storage Media
- Magnetic Disks
- Introduction to RAID
- File Organization
- Organization of Records in Files
- **Buffer Management**



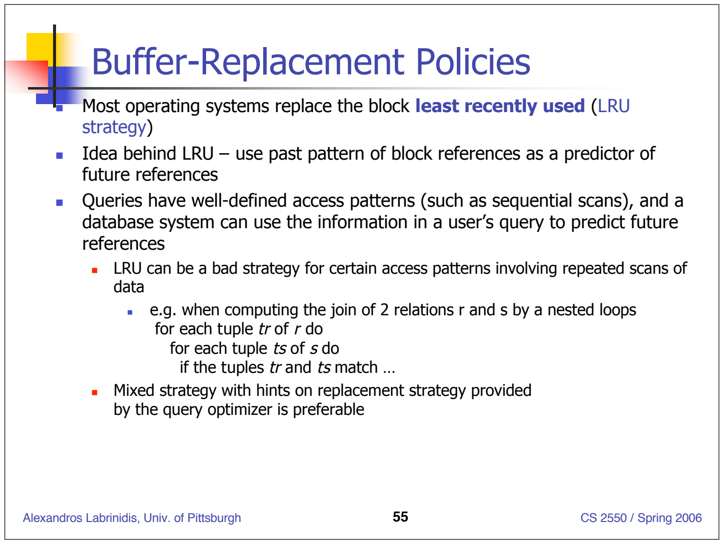
Storage Access

- 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.



Buffer Manager

- Programs call on the buffer manager when they need a block from disk.
  - If the block is already in the buffer, the requesting program is given the address of the block in main memory
  - If the block is not in the buffer,
    - the buffer manager allocates space in the buffer for the block, replacing (throwing out) some other block, if required, to make space for the new block.
    - The block that is thrown out is written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
    - Once space is allocated in the buffer, the buffer manager reads the block from the disk to the buffer, and passes the address of the block in main memory to requester

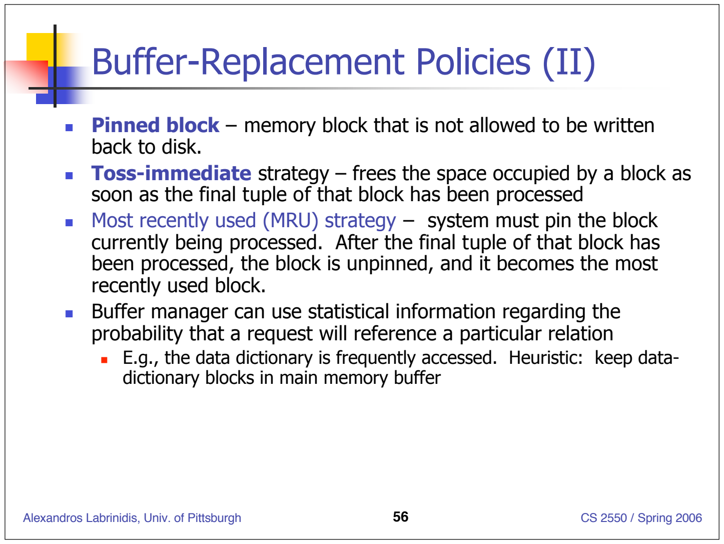


Buffer-Replacement Policies

- 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
    - e.g. 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 (II)

- **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.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer