

CMPT 454

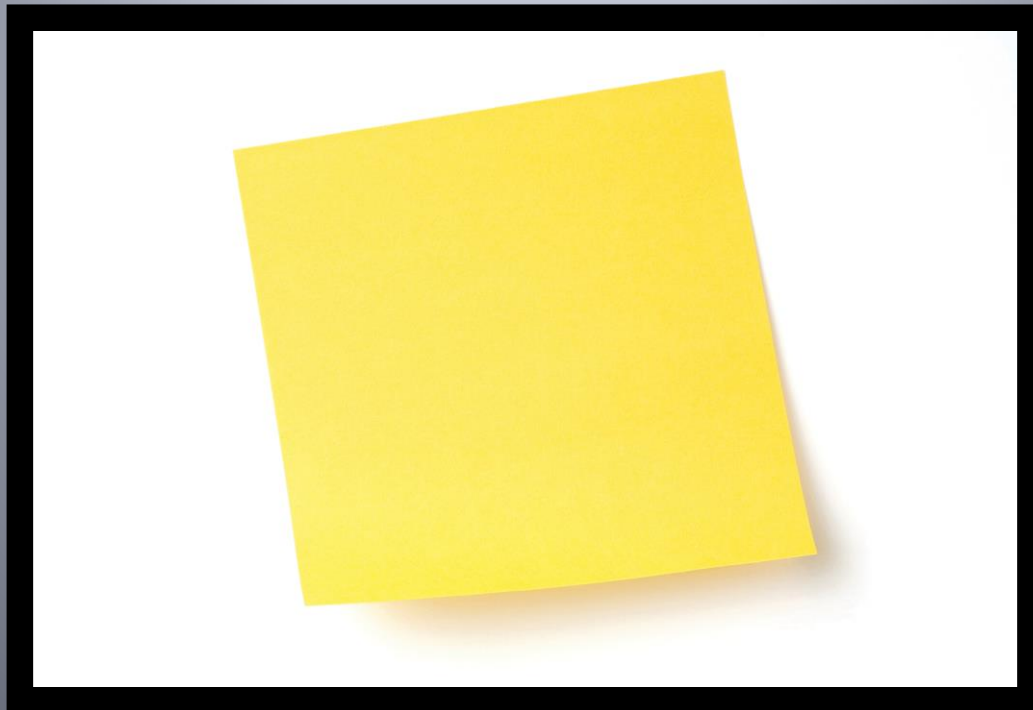
Data Storage and Disk Access

Data Storage and Disk Access

- Memory hierarchy
- Hard disks
 - Architecture
 - Processing requests
 - Writing to disk
- Hard disk reliability and efficiency
 - RAID
- Solid State Drives
- Buffer management
- Data storage

Relational Database Review

1.1



Database Tables

- A table consists of rows and columns
 - Rows represent records
 - Records must be unique
 - One or more attributes forms a primary key
 - Records are also referred to as tuples
 - Each column represents an attribute
 - Columns must have unique names within the table
- A table is an instance of a schema

<u>sid</u>	first	last	email	...
123	bob	smith	bob@sfu.ca	
456	kate	larson	kate@sfu.ca	
789	ida	chan	ida@sfu.ca	

Schemas

- A schema is defined in the DB
 - Created either by running a CREATE TABLE statement or using the DBMS GUI
 - Schemas are part of the DB metadata
 - Stored in the system catalog
- A schema is associated with one or more constraints
 - Primary key
 - Foreign key(s)
 - Other constraints
- Each column is associated with a domain – i.e. a type

Data Independence

- An application uses a conceptual view when interacting with a DB
 - In a relational DB uses the relational model
 - Interacts with SQL
- The DBMS maps the conceptual view to the physical view
 - Data stored in main memory or secondary storage
 - And is responsible for accessing data from storage devices

Database Applications

- There are two major alternatives for the interface between DBMS and application
- Embedded
 - The application access the DB directly via API function call
- Tiered client-server
 - The application makes a connection with the DBMS via ODBC, JDBC etc.
 - This may entail connecting to a server that connects to the DB server

Embedded DB

- The DBMS is linked to the application at compile time
 - They share the same address space
- Embedded DBs are often used in mobile systems



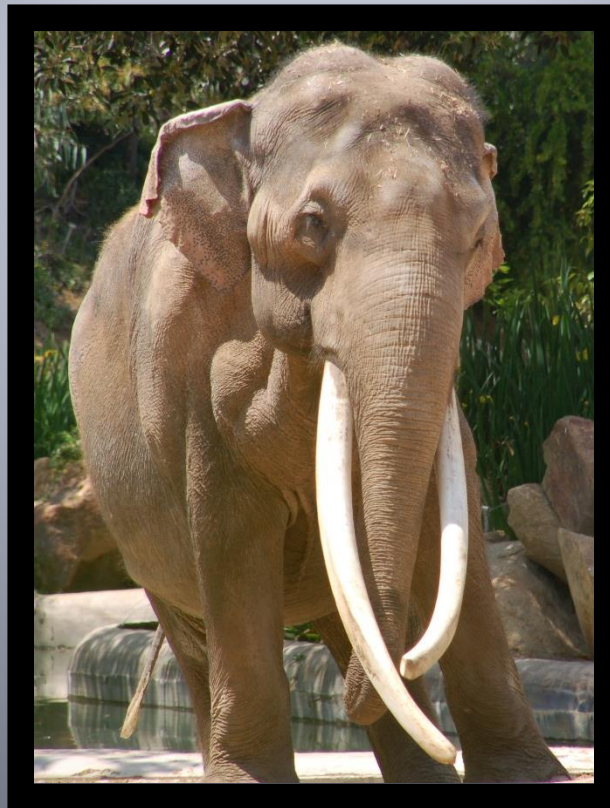
[SQLite](#) is a widely used DBMS

Client-Server Architecture

- The application and DBMS reside in separate machines
 - And communicate through a network
- There are many different possible tiered architectures
 - With different numbers of tiers
- This type of architecture is common and used with most enterprise DBMSs

Memory

1.2



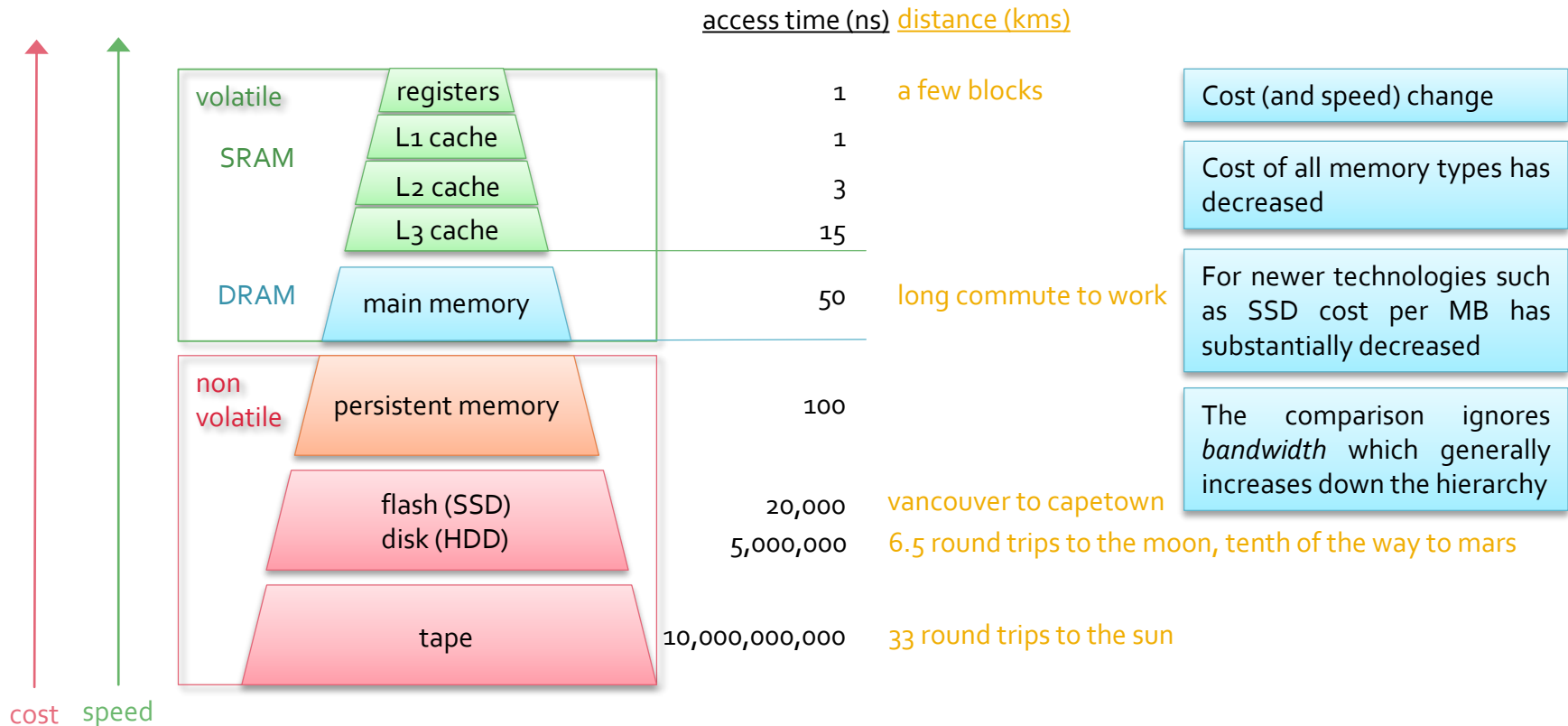
Ideal Memory

- Ideally memory should be
 - Unlimited capacity
 - High bandwidth
 - Instantaneous access
 - Persistent
 - Reliable (never fail)
 - Free
- Unfortunately ...

Of course this is not reality

Instead we have trade-offs
between these qualities

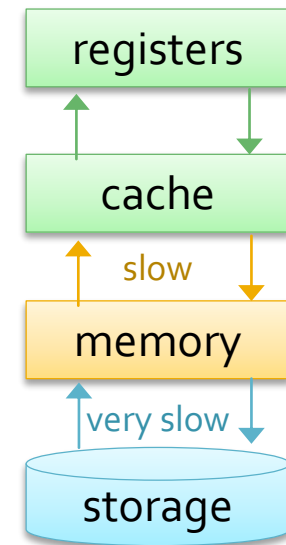
Memory Hierarchy



General interest: [latency comparisons](#)

Persistent Storage

- Database data must be stored in persistent storage
- But must be operated on in main memory
 - And ultimately in registers
- Transfer of data from memory to storage is very time-consuming
 - And should be managed carefully
 - Through buffer management



Types of Secondary Memory

- There are two main types of secondary memory
- Hard disk drives (HDDs)
 - The most widely used secondary memory device
 - Cheap
 - Relatively unreliable
 - Much slower than primary memory
- Solid state drives (SSDs)
 - Faster but more expensive than HDDs
 - Use of SSDs in databases is increasing



Tertiary Memory

- Offline storage for database archives
 - Tertiary storage should have large capacity and low cost
- Examples of tertiary storage devices include
 - Optical drives – CDs and DVDs
 - Magnetic tape
 - A very old storage media that is still used
 - Tape jukeboxes store catalogued banks of tapes

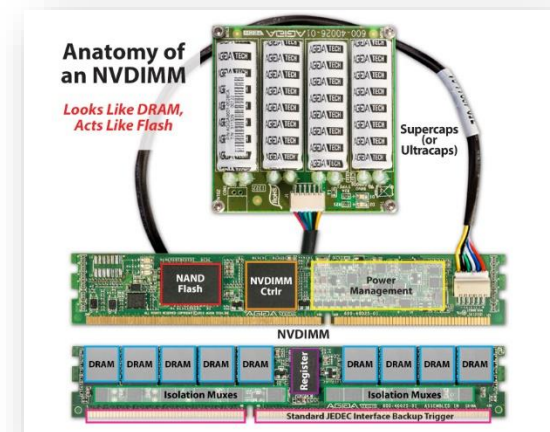


Persistent Memory

- Persistent memory is non-volatile RAM and is also known as
 - NVM – non-volatile memory
 - NVRAM – non-volatile RAM
 - SCM – storage class memory
- Characteristics
 - Byte-addressable
 - Persistent
- There are different types
 - Varied speed, capacity and cost

Types of Persistent Memory

- NVDIMM-N non-volatile dual in-line memory module
 - DRAM paired with flash with a battery
 - Similar performance to DRAM
 - Small capacity and relatively expensive
- NVDIMM-F
 - Flash storage using a DRAM bus
 - Slower than DRAM and closer to flash performance
 - Large capacity and cheap
- Other technologies other technologies are in development
 - Intel 3D XPoint – Optane DC PM, released in 2019
 - Large capacity, performance in between DRAM and flash



Classic DBMS

- Main memory assumed to be much smaller than persistent storage
 - Transaction processing occurs in main memory
 - DB resides in storage
 - Data must be transferred between main memory and storage
- Performance is primarily determined by storage access speed
 - Traditionally stored on HDDs
 - Transitioning to SSDs

In-Memory DBMS

- Reduced price of DRAM allows for large enough main memory to store entire DB
 - Or at least the working set
- Implications
 - No IO during execution of transactions
 - Changes must still be made persistent
 - But can be performed in the background
 - Violates many assumptions of classic DBMS
- Use of persistent memory also possible



Hard Disk Drives

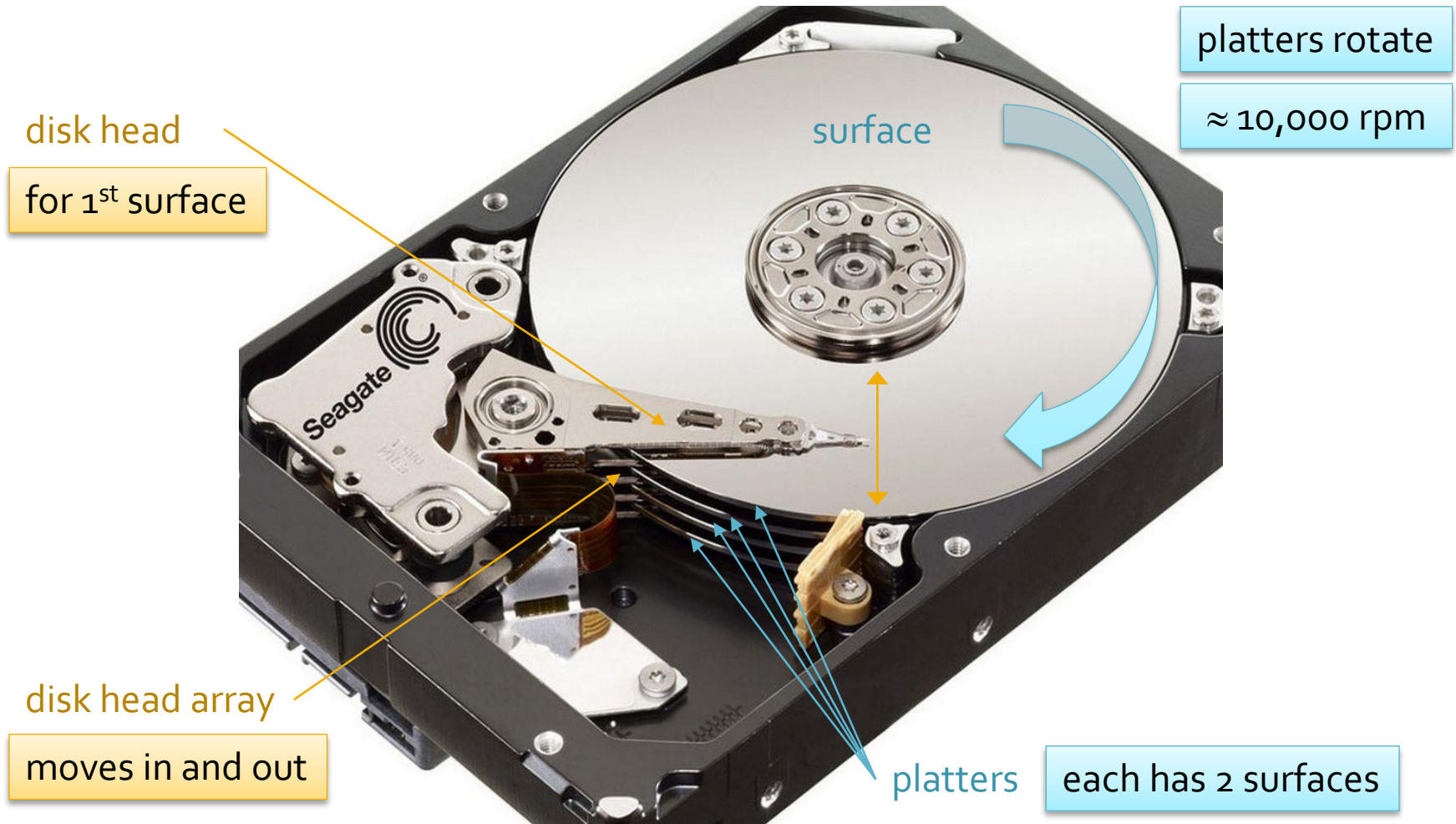
2.1



Hard Disk Drives

- A traditional database requires both persistent and working (transient) memory
 - The data is stored in non-volatile secondary storage to reduce the risk of data loss
 - Hard Disk Drive (HDD) or Solid State Drive (SSD)
 - Data is transferred from disk to main memory
 - Where operations (read, update, write, ...) are performed on the data
- New technologies provide alternatives to HDDs
 - However they are likely to be the primary storage medium for many databases for some time

Hard Disk Anatomy



Magnetic Disks

- Areas on a disk are magnetized to store bit values
 - Grouped into bytes
 - The capacity of disks for personal computers ranges from hundreds of gigabytes to a few terabytes
 - Server and mainframe disks are often 10TB or more
- Disks are made magnetic material
 - Referred to as platters
 - Single or double sided
 - Multiple platters may be grouped together in a disk to increase capacity

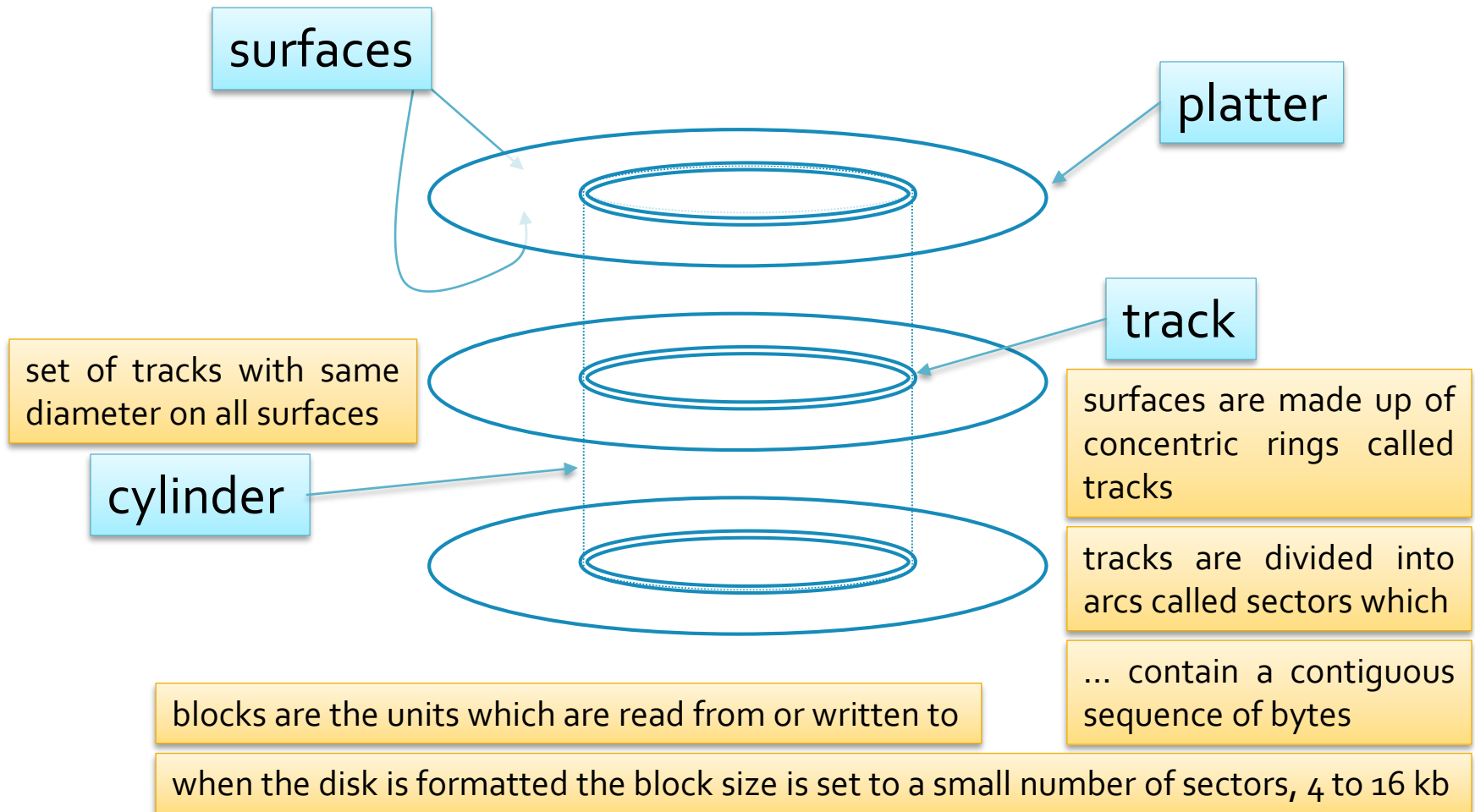


Seagate 10TB
Enterprise Capacity
7200 rpm SATA III
3.5" Internal HDD,
3.5", Spindle Drive,...
\$350.52
US\$264.49
B&H Photo-Video-
Audio



Dell 7.68TB SSD
SAS Read Intensive
12Gbps 2.5in Drive
PM1633a
\$33,200.00
Dell Canada

Magnetic Disk Structure

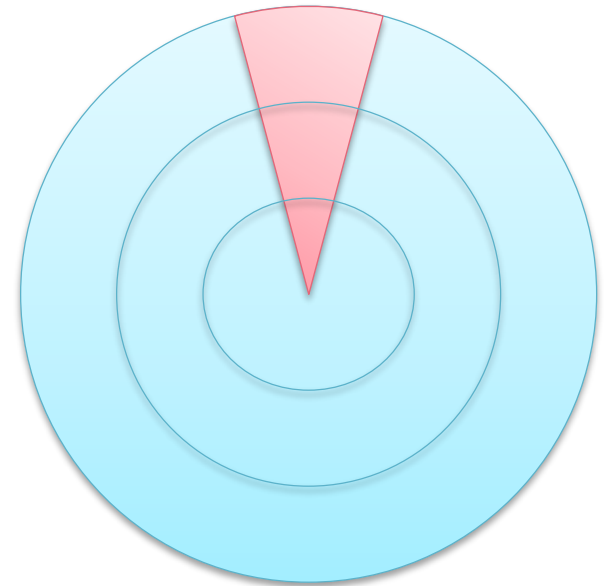


Blocks

- Block sizes vary
 - Typically ranges from 512 to 8,192 bytes
 - Blocks are separated by fixed sized gaps
 - Which contain control information
- Blocks can be addressed by cylinder number, surface number and block number
 - In many modern disk drives a single Logical Block Address (LBA) identifies a block
 - Which are numbered from 0 to block capacity – 1
- Blocks map to pages
 - A higher level abstraction

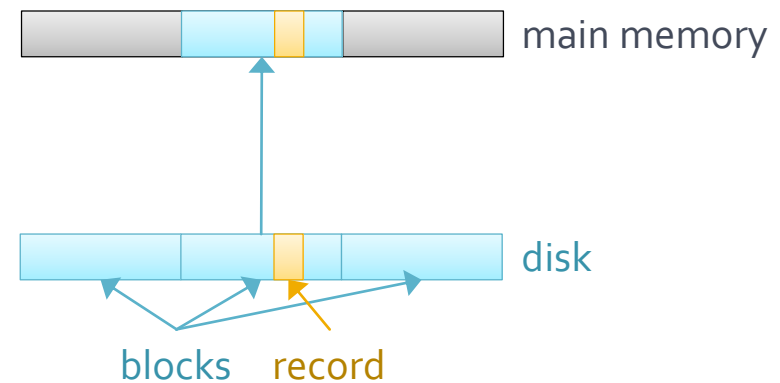
Sectors and Tracks

- A sector is an arc of a track
 - Tracks closer to the centre of the disk have smaller arc lengths
- There are a number of different sector organizations
 - Sectors subtending a fixed angle
 - Sectors on different tracks have different recording densities
 - Uniform recording density
 - The same arc on different tracks holds different numbers of sectors
 - Combination of the two



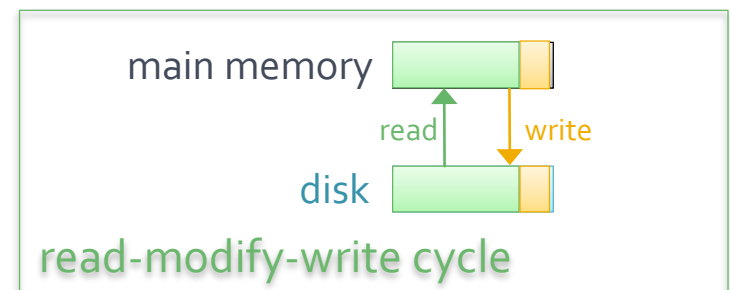
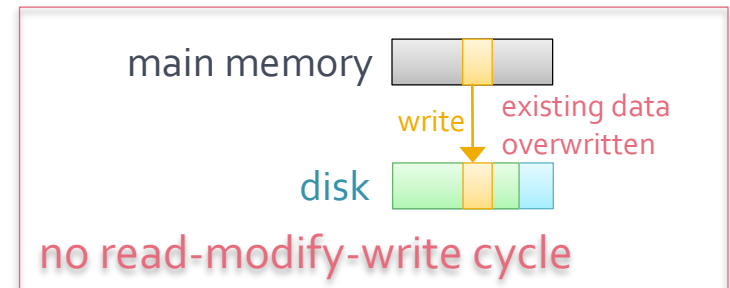
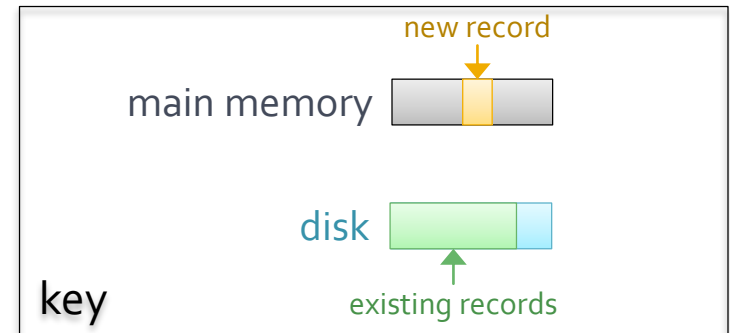
Reading Data from a Disk

- Query processor requests a record
 - Request handled by the buffer manager
 - Data from a disk is read or written in units of a block
 - Blocks typically contain multiple records
 - The desired block is read from disk into main memory



Writing Data to a Disk

- Most writes require an initial read of a disk block
 - Even blind writes
- Consider inserting a new record into a disk block
 - Which contains existing records
- The block must be read first
 - To preserve the existing data
 - Referred to as the read-modify-write cycle



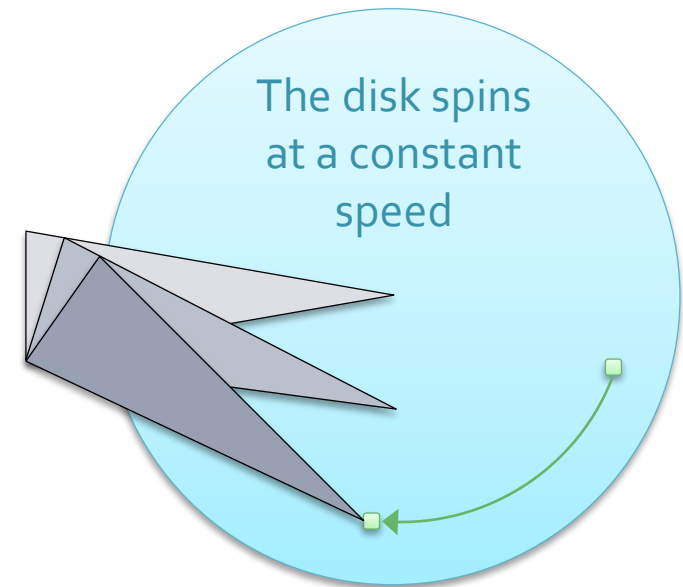
Disk Head Array

- There is one disk head for each surface
 - Moved together as a unit called a *disk head array*
 - All disk heads are in identical positions with respect to their surfaces
 - To read or write a block a disk head must be positioned over it
- Only one disk head can read or write at a time



Accessing Data in a Disk

- To access a block on a disk
 - The disk head pivots over the desired track
 - Seek time – average 4 to 10 ms
 - Wait for leading edge of block to reach the disk head
 - Rotational delay – derived from rpm
 - The desired block is read as it passes underneath the disk head
 - Transfer time – derived from rpm
- Drive controlled by a processor
 - Called the disk controller



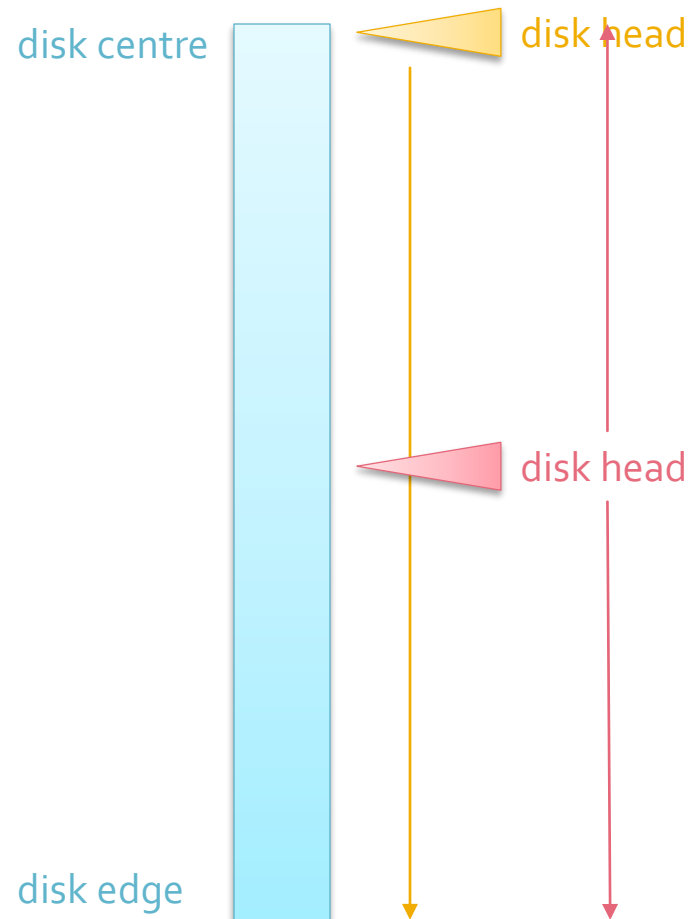
responsible for controlling the actuator, determining when the disc has rotated to a sector, transferring data

Access Time

- The seek time and rotational delay depend on
 - Where the disk head is before the request
 - Which track is being requested
 - How far the disk has to rotate
 - Average rotational delay = $\frac{1}{2}$ * max rotational delay
- The transfer time depends on the request size
 - The transfer time (in ms) for one block equals
 - $(60,000 \div \text{disk rpm}) \div \text{blocks per track}$ 60,000? 1,000(ms) * 60(s)
 - The transfer time (in ms) for an entire track equals
 - $(60,000 \div \text{disk rpm})$
 - For a disk with 10,000 rpm = 6ms

Average Seek Time

- Minimum seek time
 - 0 - the disk head is on the desired track
- Maximum seek time
 - Time to move from the innermost to outermost track
- Average seek time
 - $\frac{1}{3}$ maximum seek time
 - Not $\frac{1}{2}$ maximum seek time
- In practice the disk head does not move at constant speed
 - It must accelerate / decelerate



Access Time Comparison

- Transferring data between main memory and register is fast
 - DRAM ≈ 50 ns
 - Cache ≈ 5 ns
 - Depending on which cache
 - Register ≈ 1 ns
- HDD access is very slow in comparison
 - Can be broken into components
 - Seek time + rotational delay + transfer time
- Main memory vs. disk
 - 15 ms vs. 0.000,060 ms
 - 250,000 times faster



7,200 rpm

4.16 ms average latency



vs.



15 ms – estimate of reading one record

comparison from:
<https://scoutapm.com/blog/understanding-disk-i-o-when-should-you-be-worried>

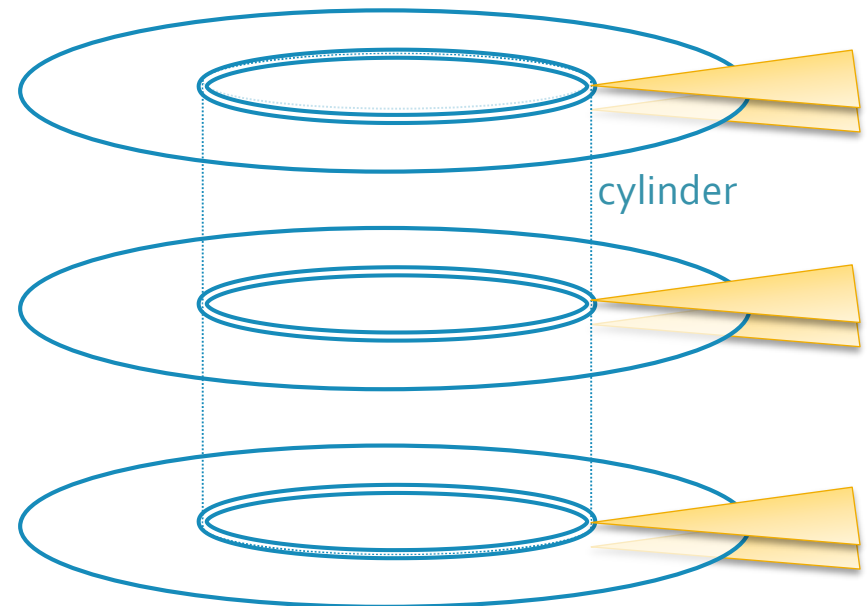
Reducing Disk Access Time

- Disk access is slow
 - Assume average seek time is 10ms
 - The largest components are seek time then rotational delay
 - and average rotational delay is 4ms
- Access two records in adjacent blocks on a track
 - Seek the track, rotate to first block, and transfer two blocks = $10 + 4 + 2 * 1 = 16\text{ms}$
 - assume transfer time of 1ms
- Accessing two records on different tracks
 - Seek the desired track, rotate to the block, and transfer the block, then repeat = $(10 + 4 + 1) * 2 = 30\text{ms}$
- Solution: store related data in close proximity

Storing Data on a Disk

- Approximate order of proximity
 - Same block
 - Adjacent blocks on same track
 - Same track
 - Same track, different cylinder
 - Adjacent cylinder
 -
- In practice
 - Fill tracks in same cylinder with related data
 - Usually from a single table
 - Then fill tracks in adjacent cylinders

The disk head array does not have to be moved to read data from the same track on different platters



Buffering Data

- The *minimum* unit of transfer of is a block
 - Multiple contiguous blocks may be transferred as a unit
 - To a correspondingly sized main memory buffer
 - This is much faster than reading blocks one at a time
 - Since seek time and rotational delay are only incurred once
- Buffer management is important in reducing access time and includes
 - Prefetching
 - Double buffering

... discussed later ...

Disk Requests

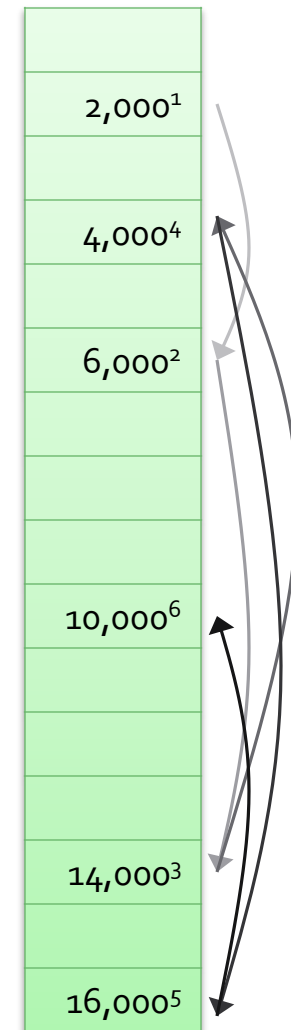
- Requests to read a block (or blocks) are processed in some order based on the disk scheduling algorithm
- There are a variety of such algorithms
 - First-come-first served (FIFO)
 - Elevator and its variants
 - SCAN, LOOK, C-SCAN, C-LOOK, ...
 - Shortest-seek
- Goals
 - Reduce overall access time
 - Avoid *starvation*



FIFO Example

- A *fair* algorithm would take a *first-come, first-serve* approach
 - Insert requests in a queue and process them in the order in which they are received

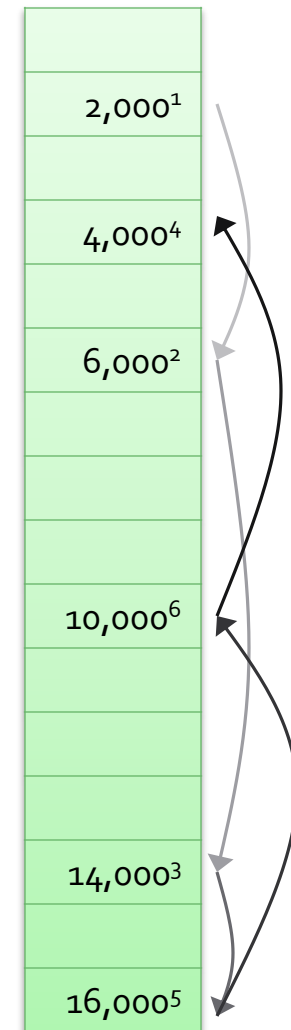
Cylinder	Received	Complete	Moved	Total
2,000	0	5	2,000	2,000
6,000	0	14	4,000	6,000
14,000	0	27	8,000	14,000
4,000	10	43	10,000	24,000
16,000	20	60	12,000	36,000
10,000	30	72	6,000	42,000



Elevator Example

- The *elevator algorithm* generally performs better than FIFO
 - Requests are buffered and the disk head moves in one direction, processing requests
 - The arm then reverses direction

Cylinder	Received	Complete	Moved	Total
2,000	0	5	2,000	2,000
6,000	0	14	4,000	6,000
14,000	0	27	8,000	14,000
16,000	20	35	2,000	16,000
10,000	30	46	6,000	22,000
4,000	30	58	6,000	28,000



HDD Failures

- Hard drives fail
- The failure probability follows a bathtub curve
 - High at the start
 - Lemons
 - High at the end
- Assume a lifespan of around 3 to 5 years
 - Back up your data



HDD after a head crash

Types of Disk Failures

- Intermittent failure
 - Multiple attempts are required to read or write a sector
 - Use checksums to check that incurred data has not been read
- Media decay
 - A bit or a number of bits are permanently corrupted and it is impossible to read a sector
- Write failure
 - A sector cannot be written to or retrieved
 - Often caused by a power failure during a write
- Disk crash
 - The entire disk becomes unreadable

Checksums

- Each sector contains additional bits whose values are based on the data bits in the sector
 - Simple single-bit checksum maintains an even parity
 - Odd (data) bit sum: checksum bit = 1
 - Even (data) bit sum: checksum bit = 0
- Using a single checksum bit allows errors of only one bit to be detected reliably
- Several checksum bits can be maintained to reduce the chance of failing to notice an error
 - e.g. 8 checksum bits, one for each bit position in a byte

e.g. 7 data bits and 1 checksum bit

data: 0111011

1: checksum bit

data bit sum is odd

Problems with Hard Drives

- Compared to main memory hard drives have two major problems
 - They are painfully slow
 - They are sadly unreliable
- Both these issues are, to some extent, addressed by using RAID
 - Or by using SSDs
 - Or an in-memory database

... there is a certain amount of exaggeration going on here ...

RAID

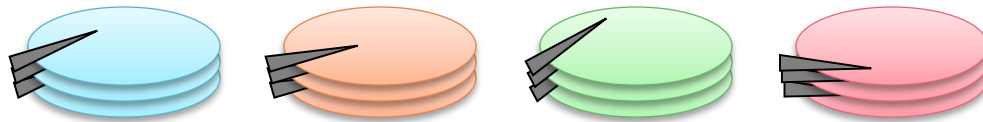
3.1



Problems with Hard Drives Reprise

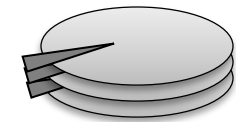
- Hard disks act as bottlenecks for processing
 - DB data is stored on disks, and must be fetched into main memory to be processed
 - Disk access is considerably slower than main memory processing
- Hard disks are also relatively unreliable
 - Disks contain mechanical components that are more prone to failure than electronic components
- One solution is to use multiple disks

Multiple Disks



- Multiple HDDs

- Each disk contains multiple platters
- Disks can be read in parallel, and
- Different disks can read from different cylinders
 - e.g. disk 1 can read data from cylinder 6,000, while disk 2 reads data from cylinder 11,000



- Single HDD

- Multiple platters
- Disk heads are always over the same cylinder

Improving Efficiency

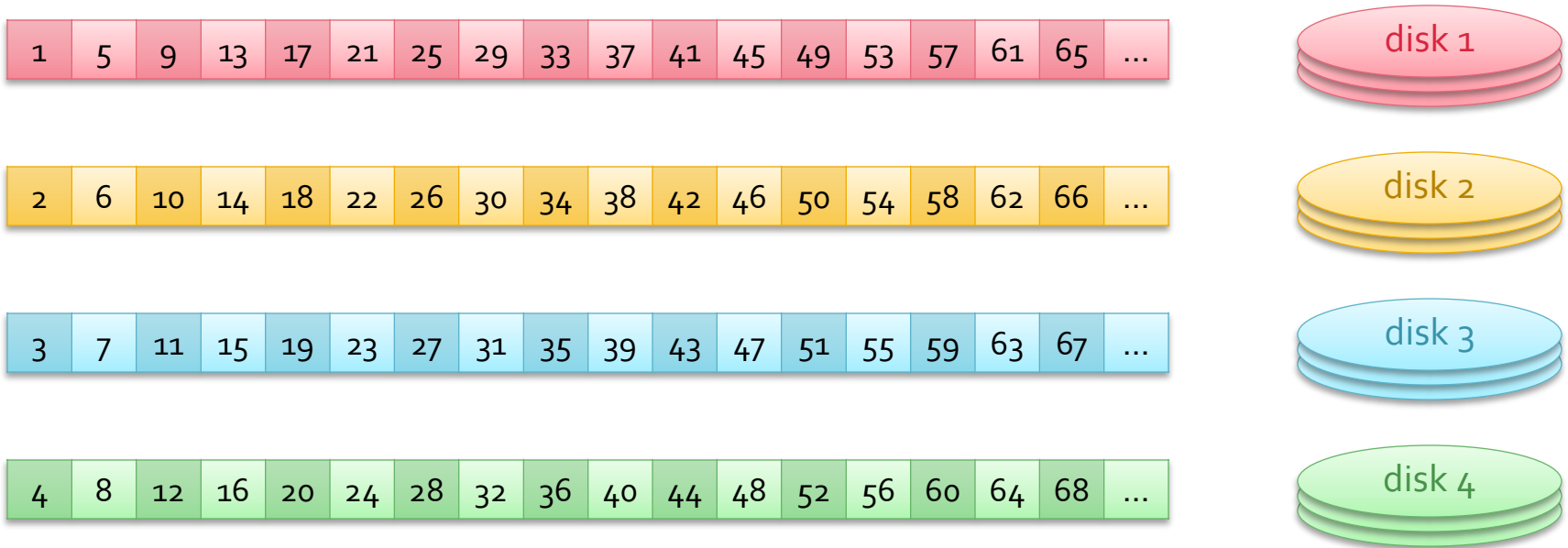
- A disk array gives the user the abstraction of a single large disk
 - When an I/O request is issued the physical disk blocks to be retrieved have to be identified
 - How the data is distributed over the disks in the array affects access time
 - And how many disks are involved in an I/O request
- Data is divided into partitions called *striping units*
 - The striping unit is usually either a block or a bit
 - Striping units are distributed over the disks using a round robin algorithm

Striping

Notional File – the data is divided into striping units of a given size



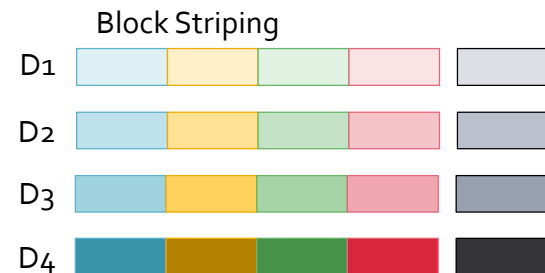
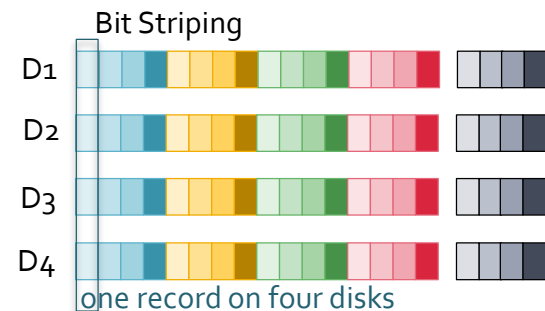
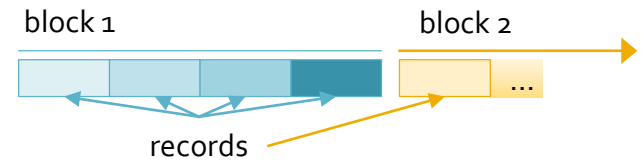
The striping units are distributed across a **RAID** system in a round robin fashion





The size of the striping unit has an impact on the behaviour of the system

Block and Bit Striping

- Assumptions
 - Multiple records fit on a block
 - Four in the example
 - Records are composed of multiple bytes
- Bit striping
 - Bit 1 of the first record on the first block (b_1r_1) on D1, b_2r_1 on D2, b_3r_1 on D3, etc.
 - All disks must be read to recreate a single record
- Block striping
 - Block 1 of the file stored on D1, block 2 on D2, block 3 on D3, etc.
 - Each record is stored on just one disk
 - Remaining disks are available for use



Disk Array Performance

- Assume that a disk array consists of D disks
 - Data is distributed across the disks using data striping
 - How does it perform compared to a single disk?
- Consider four types of requests
 - Random read – reading multiple, unrelated records
 - i.e. records that are not stored close to each other
 - Random write 
 - Sequential read – reading a number of records
 - Such as an entire file or table
 - Stored on more than D blocks in close proximity
 - Sequential write 

Performance in General

- Use all D disks to improve efficiency, and distribute data using block striping
- Random reads and writes
 - Up to D different records can be read from or written to at once
 - Depending on which disks the records reside on
 - Records might reside on different disks, or – if unlucky – all on the same disk
- Sequential reads and writes
 - Related data are distributed over all D disks so performance is D times faster than a single disk
- Writes generally entail reading the data first

But what about *reliability* ...

Measuring HDD Failure

- The standard hard drive failures measure is MTBF, expressed in hours
 - Annualized Failure Rate (AFR) expresses the probability per year a drive will fail
 - $AFR \approx 8766 / MTBF$ 8766? average number of hours in a year
- Modern enterprise hard drives have MTBF values of around 1,000,000
 - Consumer quality drives are around half these values
 - Hard drive reliability has increased substantially over time
- MTBF values are usually published by manufacturers
 - And should be assumed to give a flattering picture of a drive's reliability

Reliability

- Hard disks contain mechanical components and are less reliable than other, purely electronic, components
 - Increasing the number of hard disks *decreases* reliability, reducing the mean-time-between-failures (MTBF)
 - Let's say the MTBF of a hard disk is $\approx 500,000$ hours, or 57 years
- In a disk array the overall MTBF decreases
 - Because the number of disks is greater
 - MTBF of a 100 disk array is 210 days: $(500,000 \div 100) \div 24$
 - This assumes that failures occur independently and
 - The failure probability does not change over time
- Reliability can be improved by storing redundant data

Redundancy

- Reliability of a disk array can be improved by storing redundant data
- If a disk fails the redundant data can be used to reconstruct the data lost on the failed disk
 - The data can either be stored on a separate *check disk* or
 - Distributed uniformly over all the disks
- Redundant data is typically stored using one of two methods
 - *Mirroring*, where each disk is duplicated
 - A *parity scheme*, where sufficient redundant data is maintained to recreate the data in any one disk
 - Other redundancy schemes provide greater reliability

Parity Scheme

- For each bit on the data disks there is a parity bit on a check disk
 - If the sum of the data disks bits is even the parity bit is set to zero
 - If the sum of the bits is odd the parity bit is set to one
- The data on any one failed disk can be recreated bit by bit

0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1 ...

1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 ...

0 0 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1 ...

0 1 1 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 1 1 ...

First three bytes of data disks in a four (data) disk RAID system

1 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 0 1 1 1 ...

Corresponding bytes of the system's check disk

Reading – not affected by parity data

Writing - either

calculate new value of the parity bit from all the data disks or

flip parity bit if corresponding data bit has changed

Introducing RAID

- A RAID system consists of several disks organized to increase performance and improve reliability
 - Performance is improved through *data striping*
 - Reliability is improved through *redundancy*
 - **RAID** stands for **R**edundant **A**rrays of **I**ndependent **D**isks
- There are several RAID schemes or *levels*
 - The levels differ in terms of their
 - Read and write performance,
 - Reliability, and
 - Cost

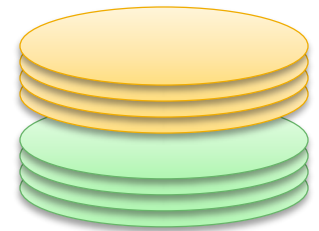
RAID Level 0

- All D disks are used to improve efficiency, and data is distributed using block striping
- No redundant information is kept
- Read and write performance is very good
- Reliability is poor
 - Unless data is regularly backed up a RAID 0 system should only be used when the data is not important
- A RAID 0 system is the cheapest of all RAID levels
 - Because there are no disks used for storing redundant data
 - With D data disks there are 0 check disks



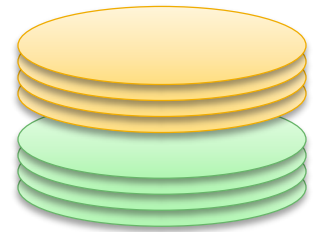
Level 1: Mirrored

- An identical copy is kept of each disk in the system, hence the term *mirroring*
- Read performance is similar to a single disk
 - No data striping, but parallel reads of the duplicate disks can be made improving random read performance
- Write performance is worse than a single disk as the duplicate disk has to be written to
 - Writes to the original and mirror should not be performed simultaneously in case of a global system failure
 - But write performance is superior to most other RAID levels
- Very reliable but costly
 - With **D** data disks, a level 1 RAID system has **2D** disks



Level 1+0: Striping + Mirroring

- Sometimes referred to as *RAID level 10*, combines both striping and mirroring
- Very good read performance
 - Similar to RAID level 0
 - $2D$ times the speed of a single disk for sequential reads
 - Up to $2D$ times the speed of a single disk for random reads
 - Allows parallel reads of blocks that, conceptually, reside on the same disk
- Poor write performance, that is similar to RAID level 1
- Very reliable but the most expensive RAID level
 - Requires $2D$ disks



Writing and Redundant Data

- Writing is the Achilles heel of RAID
 - Data and check disks should not be written to simultaneously
 - Parity information may have to be read before check disks are written
- Bit striping random and sequential writes, block striping sequential writes
 - Write all data disks using r-m-w cycle
 - Recalculate parity data from data disks
 - Write new parity data to check disk
- Block striping random writes
 - Write to single disk using r-m-w cycle
 - Read check disk and calculate parity data
 - Write check disk



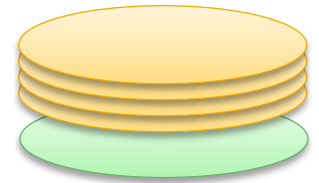
In many RAID systems writing is less efficient than a single disk!

Striping Units Performance

- A RAID system with **D** disks can *read* data up to **D** times faster than a single disk system
- For sequential reads there is no performance difference between bit striping and block striping
- Block striping is more efficient for random reads
 - With bit striping all **D** disks have to be read to recreate a single record (and block) of the data file
 - With block striping a complete record is stored on one disk therefore only that one disk is required to satisfy a random read
- Write performance is similar except that it is affected by the parity scheme
 - Random writes are typically inefficient

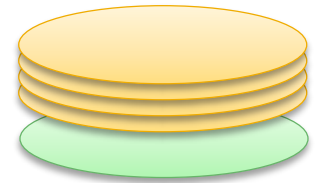
Levels 2 and 3

- Level 2 – Memory Style Error Correcting Code
 - The striping unit is a single bit
 - Uses a scheme that allows the failed disk to be identified
 - Which increases the number of disks required
 - Modern disk controllers can detect a failed disk so this is unnecessary
 - Can only tolerate the loss of a single disk
- Level 3 – Byte Interleaved Parity
 - The striping unit is a byte
 - Random read and write performance is poor as all disks have to be accessed for each request
 - Can tolerate the loss of a single disk
 - Requires $D + 1$ disks



Level 4 – Block Interleaved Parity

- Uses block striping to distribute data over disks
- Uses one redundant disk containing parity data
 - The i th block on the redundant disk contains parity checks for the i th blocks of all data disks
- Good sequential read performance
 - D times single disk speed
- Very good random read performance
 - Disks can be read independently therefore up to D times single disk speed

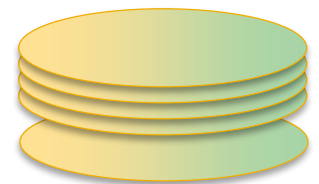


Level 4 – Performance

- Cost is moderate
 - Only one check disk is required
- The system can tolerate the loss of one drive
- Write performance is poor for random writes
 - In which different data disks are written independently
 - For each such write a write to the redundant disk is also required
- Performance can be improved by distributing the redundant data across all disks – RAID level 5

Level 5 – Block-Interleaved Distributed Parity

- The dedicated check disk in RAID level 4 tends to act as a bottleneck for random writes
- RAID level 5 does not have a dedicated check disk but distributes the parity data across all disks
 - Increasing the performance of random writes
 - Read performance is marginally improved
 - Sequential read and write performance is similar to level 4
- Cost is moderate, with the same effective space utilization as level 4
- The system can tolerate the loss of one drive



Multiple Disk Crashes

- RAID levels 4 and 5 can only cope with single disk crashes
 - If multiple disks crash at the same time data will be lost
- It is unlikely that two disks will fail at the same time
 - At least, by coincidence
- Replacing a disk is time consuming
 - Therefore it is possible that a second disk may fail during the rebuilding process
 - RAID level 6 allows systems to deal with multiple disk crashes

Level 6 – P + Q Redundancy

- RAID level 6 records two sets of parity data for each set of bits across all disks
 - Using two different parity schemes
 - The system requires at least 4 disks
 - In a 4 disk system half of the data is parity data
 - As the number of disks increases the percentage of redundant data decreases
- The redundancy data allows a system to recover from two simultaneous disk crashes



Level 6 – Performance

- The parity data is distributed across all disks
 - Read performance is similar to level 5
- Write performance is worse than level 5
 - Sequential writes are slower since two sets of parity data have to be calculated
 - Random writes are considerable slower since the two sets of parity data are on separate disks
 - Which requires a read-modify-write cycle for each disk

Parity Scheme and Reliability

- In real-life RAID systems the disk array is partitioned into *reliability groups*
 - A reliability group consists of a set of data disks and a set of check disks
 - The number of check disks depends on the reliability level that is selected
- Using a RAID system and reliability groups greatly increases reliability over a single disk
 - Making the possibility of failure relatively remote

Which RAID Level?

- RAID level summary
 - Level 0 is cheap, improves performance but not reliability
 - Level 1+0 is better than level 1 and has the best write performance
 - Levels 2 and 4 are always inferior to 3 and 5
 - Level 3 is worse than 4 for random requests
 - Level 5 is a good general-purpose solution
 - Level 6 is appropriate if higher reliability is required
- The choice is usually between 0, 1+0, 5 and 6
 - There are a number of other RAID levels that are also used derived from the levels discussed here
 - RAID 7, RAID-DP, RAID-K, RAID-Z, ...

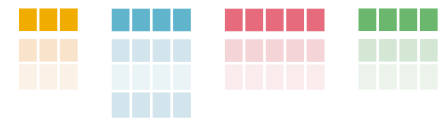
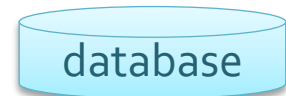
Record and Page Format

4.1



Data Organization

- A database is made up of tables
 - A table is generally represented by a single file
- Tables are collections of records
 - A record describes an entity or relationship
 - And is defined by its attribute values
- Attributes also referred to as fields
 - Each attribute is associated with a type
 - Also known as a domain
 - Information about fields is stored in the *system catalog* **metadata** [SQL Server types](#)

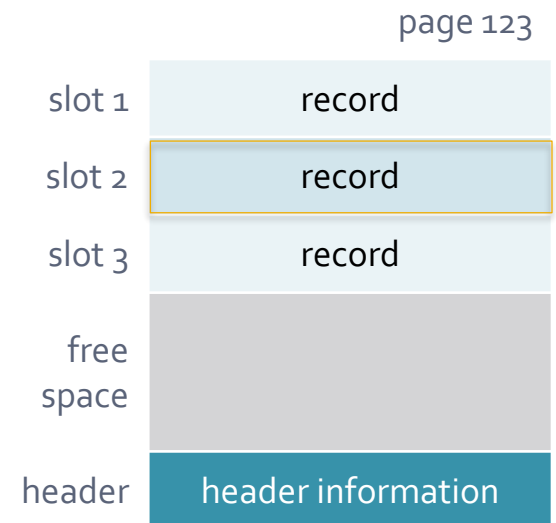


id	name	dob	rate
1	bob	12/3/1974	21.35
2	kate	21/10/2001	17.34
3	sue	11/11/1999	19.22

name	type	size	...
id	int	4	
name	varchar	255	
dob	datetime	8	
rate	real	8	

Locating Records

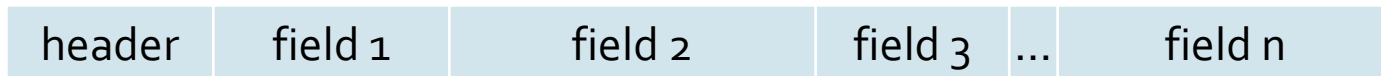
- Records are stored in pages
 - Fixed-size abstract units of storage
 - Generally map to a block on a disk
 - Or a frame in main memory
- Page contents
 - Page header – administrative data
 - Slots – each slot contains a record
- Record IDs
 - Records are uniquely identified by record IDs (RID)
 - Record IDs are page number : slot number pairs



RID: 123,2

Locating Fields – Fixed Length

- Records are either fixed or variable length
 - Fixed length records have some important advantages
 - Easier to rearrange records
 - Less overhead for header data
 - More efficient search
- Fixed length records
 - Both the number of fields and the field length is fixed
 - Given the address of a record a field can be found
 - By referring to the field size in the system catalog
 - This format is sometimes referred to as relative location

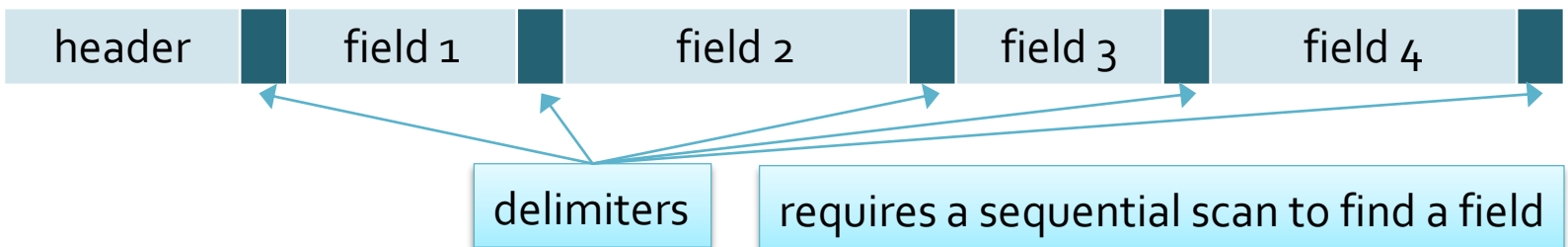


e.g. find field 3

= base address + size(field1) + size (field2)

Variable Length Fields

- In the relational model each record contains the same number of fields
 - However fields may be of variable length
 - e.g. VARCHAR
 - Two tables could be stored in the same file – a mixed file
 - Allows fast access to the results of a join between two related tables
- Alternative 1 – separate attributes with delimiters
 - Delimiters identify the end of a field



Variable Length Fields Alternatives

- There are other alternatives for recording variable length records
- Store additional data in the record header
 - An array containing the length of each field
 - Or just the lengths of the variable length fields, then store all fixed length fields first
 - Allows fields to be found by an offset calculation
- If a record contains a variable *number* of fields other organizations may be used
 - Pointers to a list of fields
 - e.g. a mixed file
 - Preceding the attribute by the attribute identifier

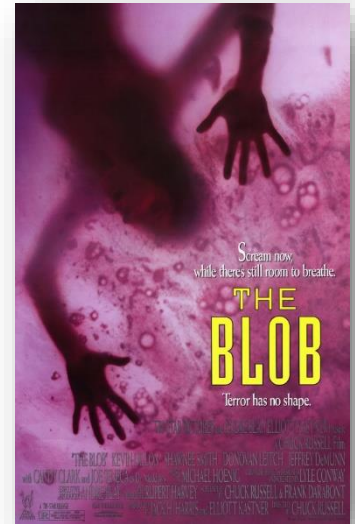
Variable length field sizes may grow / shrink

Gets smaller – wastes space

Gets larger – may not fit in its slot, or in its page, or may even exceed the page size

Large Fields

- Some data types may not fit on a single page
 - Large object data
 - Text, images, video, sound, ...
- LOB data is stored as either binary or character data
 - BLOB – unstructured binary data
 - CLOB, NCLOB – character data
 - BFILE – unstructured binary data in OS files
- LOB data types store and manipulate large blocks of unstructured data
 - The maximum size of a LOB is large
 - At least 8 terabytes in Oracle 10g
 - LOB data must be processed by application programs

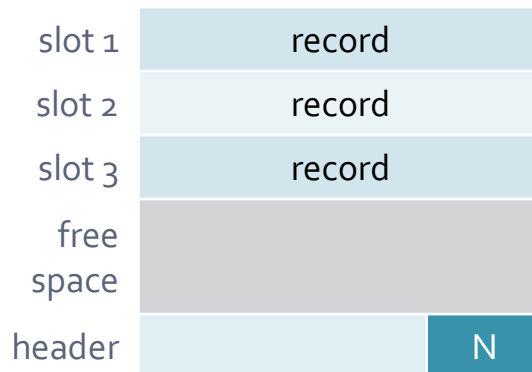


LOB Storage

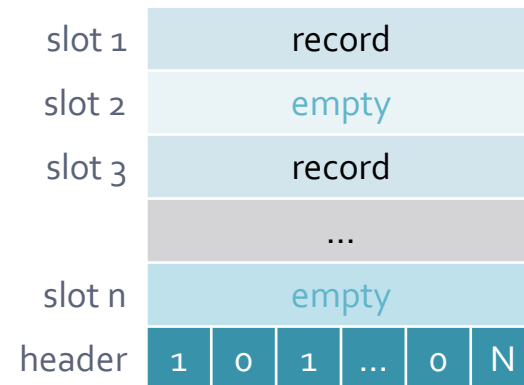
- LOB's have to be stored on a sequence of pages
 - Ideally the pages should be contiguous for efficient retrieval, or
 - Store the LOB on a linked list of blocks
 - Where each block contains a pointer to the next block
- If fast retrieval of LOBs is required they can be striped across multiple disks for parallel access
- It may be necessary to provide an index to a LOB
 - For example indexing by seconds for a movie to allow a client to request small portions of the movie

Fixed Length Record Pages

- Packed
 - Offset calculation to find record
 - Move records up on deletion
- Simple
- But
 - Time consuming to move records
 - External RIDs no longer valid if slot number changes on deletion

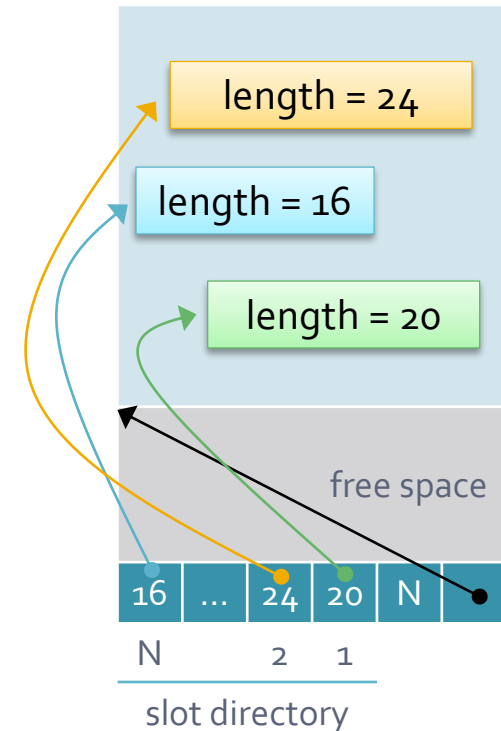


- Bit array
 - Page is divided into slots
 - Header includes a bit array which shows which slots contain records
- Advantages
 - No changes to external references
 - No rearranging records
- But slightly more administration



Variable Length Record Pages

- Maintain a slot directory
 - Each entry is a pair
 - Record offset
 - Record length
- Records can be moved without affecting RID
 - By modifying the slot entry
- Flexible
 - But more complex than fixed record organization



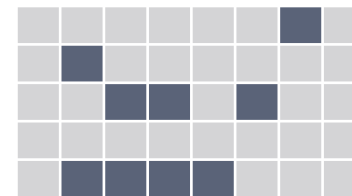
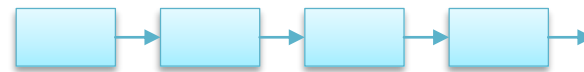
Files

4.2



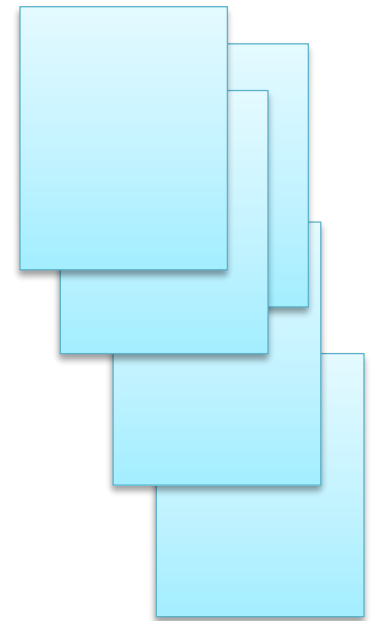
Disk Space Management

- A DB increases and decreases in size
 - The disk space manager for the DBMS maps pages to blocks
 - Over time gaps in sequences of allocated blocks may appear
- Free blocks need to be recorded so that they can be allocated in the future, using either
 - A linked list with the head pointing to the first free block
 - A bitmap where each bit corresponds to a single block
 - Allows for fast identification of contiguous areas of free space



Files and Records

- Pages containing related records are organized into files
 - One file usually corresponds to one table
- Files are expected to span several pages
 - So must allow access to all the pages in the file
- There are different file organizations
 - With their own strengths and weaknesses
- All file organizations support
 - Opening and closing the file
 - Searching for and reading records
 - Inserting, deleting and modifying records
 - Scanning the entire file



Heap Files

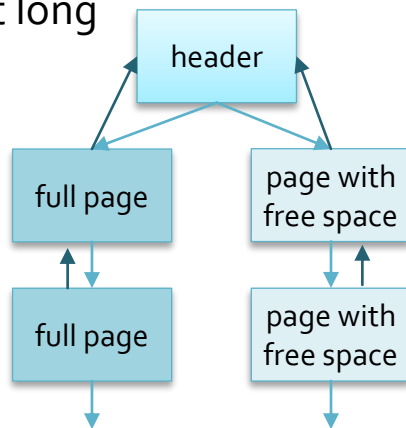
- Heap files are not ordered in any way
 - They guarantee that all of the records in a file can be retrieved by repeatedly requesting the next record
- Inserting new records into heap files is very efficient
 - Searching for records is inefficient
 - As it requires a linear search of the file
 - Deletion and modification of records first requires they be found
- To support file operations it is necessary to
 - Keep track of the pages in the file
 - Keep track of which of those pages contain free space

Fast access to records in a heap file requires additional data structures

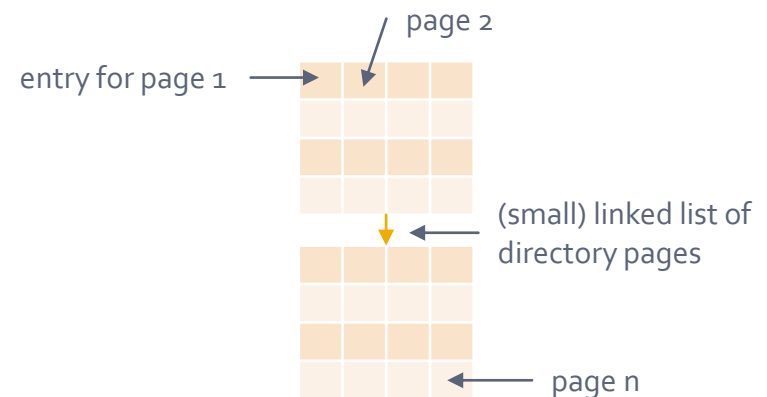
Indices

Heap File Organizations

- Linked List
 - Two doubly linked lists of pages
 - Pages with free space
 - Pages that are full
 - Records page ID of list head
- Finding a page with enough space may be time-consuming
 - Variable-length records may make the list long



- Directory
 - File pages recorded in a directory
 - Entries kept in page order
- Directory entries record
 - Whether or not the page is full
 - The amount of free space
 - Pages need not be visited to see if they contain enough free space



Finding Records in a Heap File

- Heap files do not provide efficient access to records based on (any) search criteria

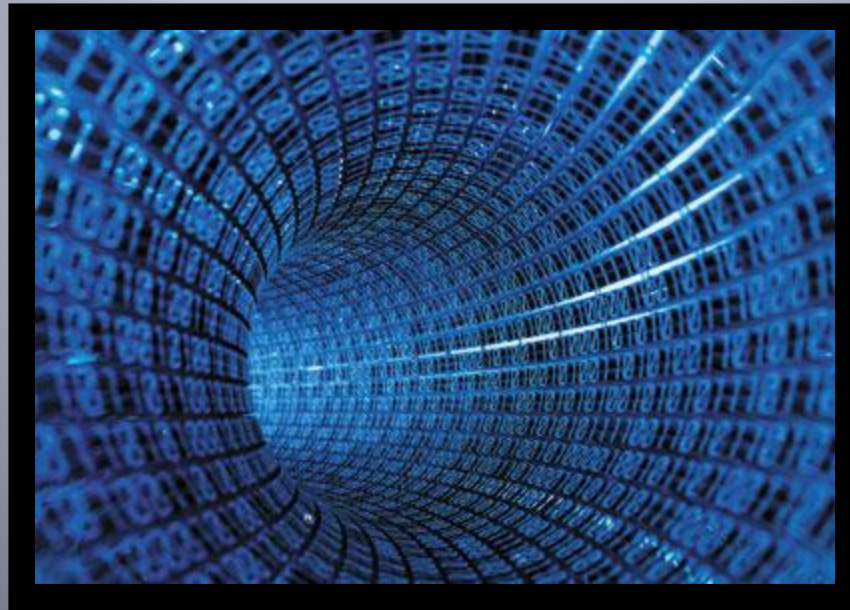
94 97 34 68 65 80 68 94 83 70 56 81 75 79 53 find 81

- Other file organizations support more efficient access to records
 - Ordered files
 - Ordered on some record field
 - Hash files
 - Files of mixed records
 - That contain records of more than one table
 - Files organized as B or B+ trees

Discussed in the section on indices

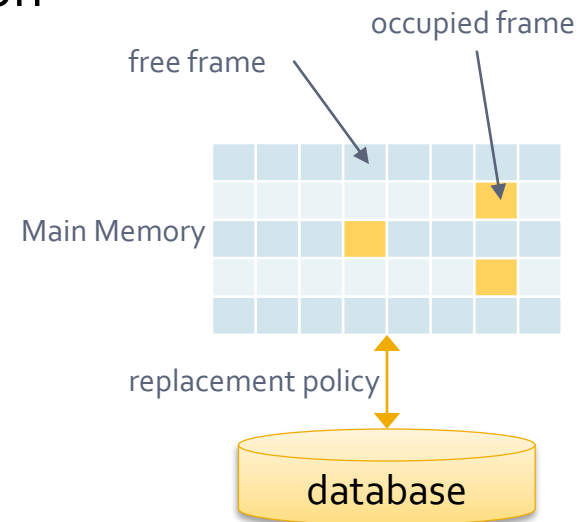
Buffer Management

5.1



The Buffer Manager

- Main memory is partitioned into a collection of frames called the *buffer pool*
 - The buffer manager is responsible for bringing pages from disk to main memory as required
 - Pages are mapped to *frames*
 - Processes inform the buffer manager if a page is no longer required
 - And if it has been modified
- A DB may be many times larger than the buffer pool
 - Accessing an entire table can easily fill up the buffer pool
 - The buffer manager replaces pages by following a *replacement policy*

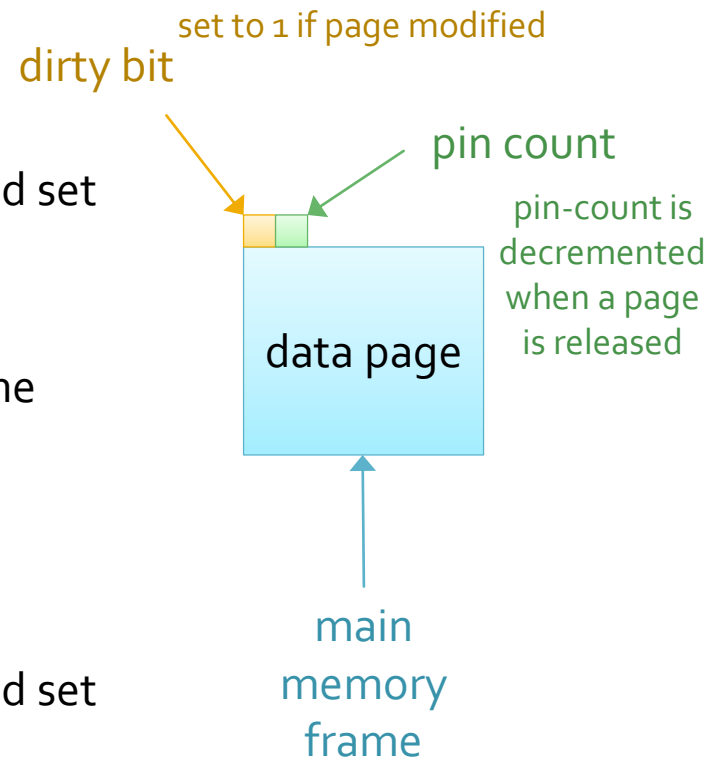


In addition to the buffer pool a data structure maps pages to frames

e.g. a hash table

Page Request Process

- If the page is already in the buffer pool
 - Increment the pin-count (pin the page)
- If there are vacant frames
 - Read requested page into chosen frame and set pin-count to 1
- If there are no vacant frames
 - Select an frame with pin-count = 0 using the replacement policy
 - If the chosen frame is dirty write to disk
 - If there are no frames with pin-count = 0
 - The transaction must wait or abort
 - Read requested page into chosen frame and set pin-count to 1



Buffer Replacement Policies

- Assumption: buffer size much smaller than working set
- The policy used to replace frames can affect the efficiency of database operations
 - Ideally a frame should not be replaced if it will be needed again in the near future
- Buffer replacement policies
 - Random
 - FIFO
 - Least Recently Used (LRU)
 - Clock Replacement
 - Most Recently Used (MRU)

Queue whose entries are frames with pin-count = 0

Replaces frame at front of queue

Requires memory for queue

Assumes frames not recently used are no longer required

Clock Replacement

- A variant of the LRU policy with less overhead
 - Instead of a queue the system requires one bit per frame, and a single variable, called *current*
 - Assume that frames are numbered from 0 to $B-1$
 - Where B is the number of frames
- Each frame has an associated *referenced* bit
 - Which is initially set to 1 when the frame is read or accessed
- The *current variable* is initially 0, and is used to show the next frame to be considered for replacement



Clock Replacement Process

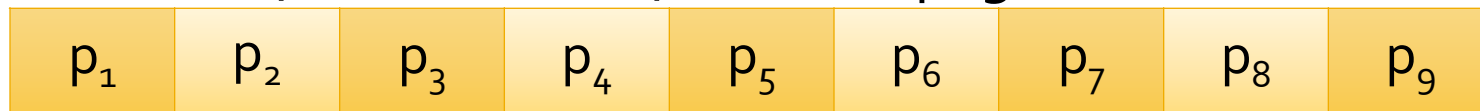
- Consider the *current* frame for replacement
 - If *pin-count* $\neq 0$, increment *current* consider next frame
 - If *pin-count* = 0 and *referenced* bit is 1 make frame a candidate
 - Switch *referenced* to 0 and increment *current*
 - If *pin-count* = 0 and *referenced* is 0
 - Replace the frame 2nd time around replace frame
 - If *current* equals $B-1$ set it to 0 wrap around
- Only replaces frames with *pin-counts* of zero
- Frames with a *pin-count* of zero are only replaced after all older candidates are replaced

Is LRU the Right Policy?

- LRU and clock replacement are *fair* schemes
- They are not always the best strategies for a DB system
 - It is common for some DB operations to require repeated sequential scans of data (e.g. Cartesian products, joins)
 - With LRU such operations may result in *sequential flooding*
- An alternative is the *Most Recently Used* policy
 - This prevents sequential flooding but is a generally poor replacement policy
- Most systems use some variant of LRU
 - Some systems will identify certain operations, and apply MRU for those operations

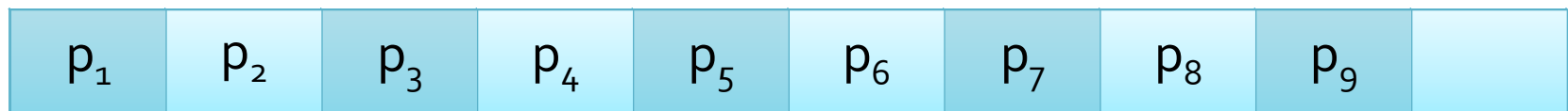
No Sequential Flooding

- Assume that a process requests sequential scans of a file
- The file, shown below, has nine pages



- Assume that the buffer pool has *ten* frames

Buffer Pool

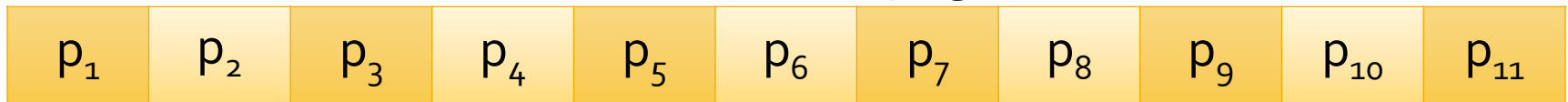


Read page 1 first, then page 2, ... then page 9

All the pages are in the buffer, when the next scan of the file is requested, no further disk access is required!

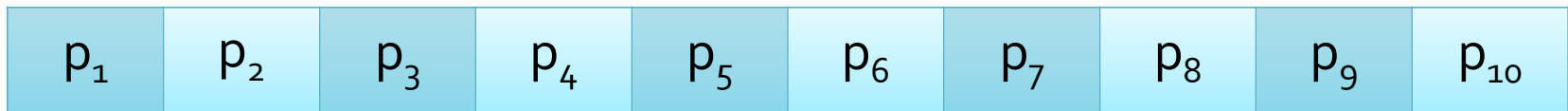
Sequential Flooding

- Assume that a process requests sequential scans of a file
- This file, shown below, has *eleven* pages



- Assume that the buffer pool still has *ten* frames

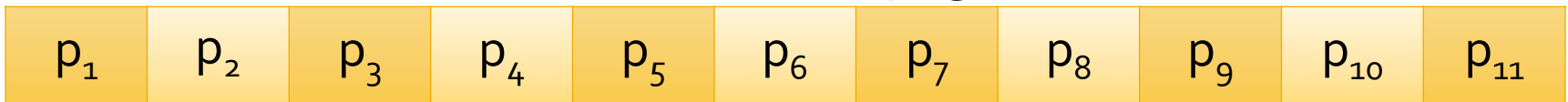
Buffer Pool



Read pages 1 to 10 first, page 11 is still to be read

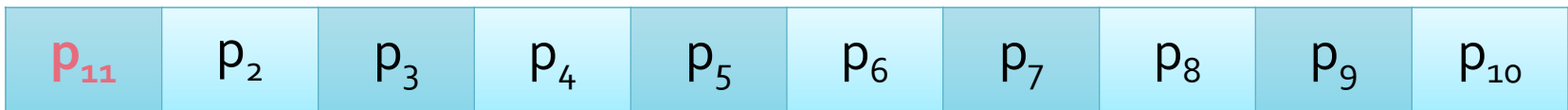
Sequential Flooding

- Assume that a process requests sequential scans of a file
- This file, shown below, has *eleven* pages



- Assume that the buffer pool still has *ten* frames

Buffer Pool

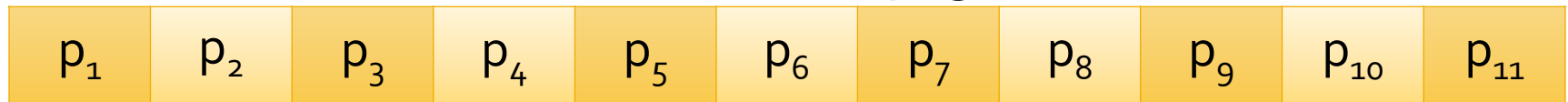


Read pages 1 to 10 first, page 11 is still to be read

Using LRU, replace the appropriate frame, which contains p₁, with p₁₁

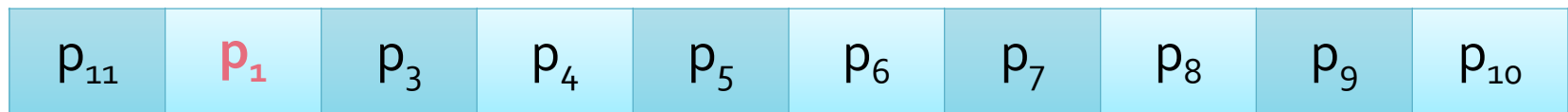
Sequential Flooding

- Assume that a process requests sequential scans of a file
- This file, shown below, has *eleven* pages



- Assume that the buffer pool still has *ten* frames

Buffer Pool



Read pages 1 to 10 first, page 11 is still to be read

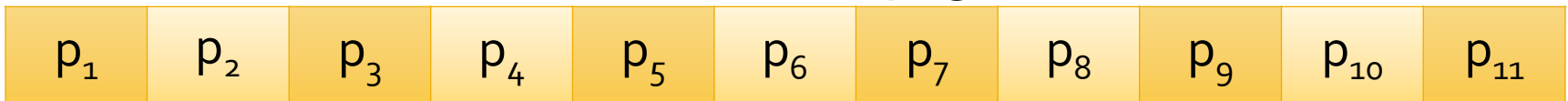
Using LRU, replace the appropriate frame, which contains p₁, with p₁₁

The first scan is complete, start the second scan by reading p₁ from the file

Replace the LRU frame (containing p₂) with p₁

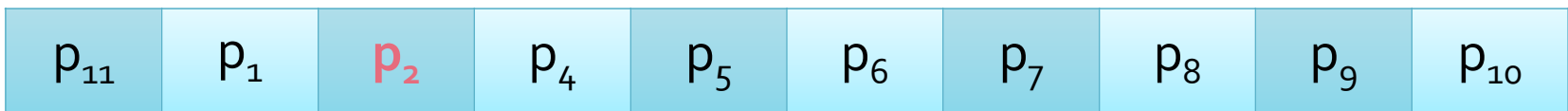
Sequential Flooding

- Assume that a process requests sequential scans of a file
- This file, shown below, has *eleven* pages



- Assume that the buffer pool still has *ten* frames

Buffer Pool



Read pages 1 to 10 first, page 11 is still to be read

Using LRU, replace the appropriate frame, which contains p₁, with p₁₁

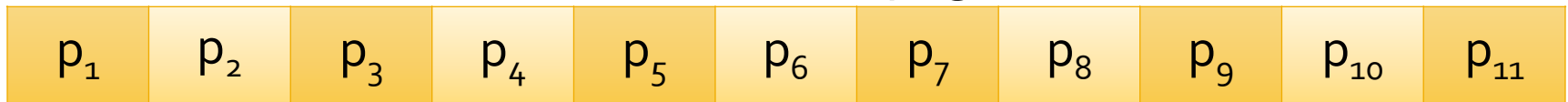
The first scan is complete, start the second scan by reading p₁ from the file

Replace the LRU frame (containing p₂) with p₁

Continue the scan by reading p₂, ...

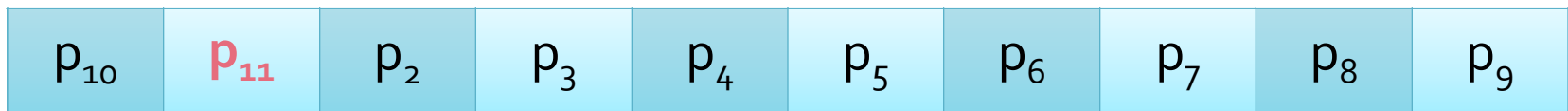
Sequential Flooding

- Assume that a process requests sequential scans of a file
- This file, shown below, has *eleven* pages



- Assume that the buffer pool still has *ten* frames

Buffer Pool



Each scan of the file requires that every page is read from the disk!

In this case LRU is the *worst* possible replacement policy!

DBMS Buffer Management

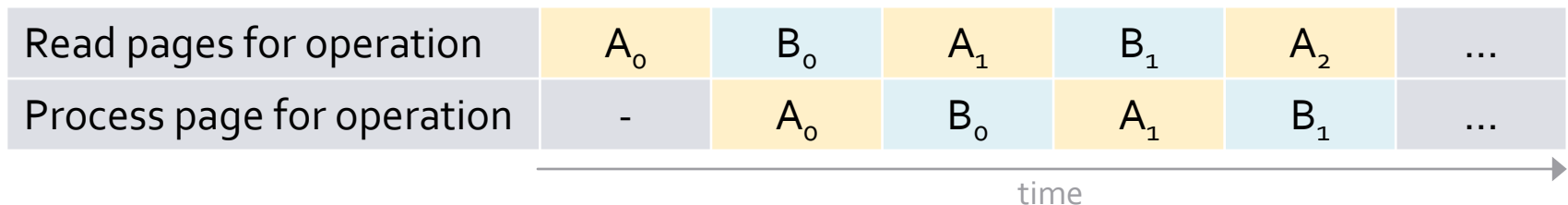
- A DBMS can often predict patterns in the way in which pages are referenced
 - Most page references are generated by processes such as query processing with known patterns of page accesses
 - Knowledge of these patterns allows for a better choice of pages to replace and
 - Allows *prefetching* of pages, where the page requests can be anticipated and performed before they are requested
- A DBMS requires the ability to *force* a page to disk
 - To ensure that the page is updated on a disk
 - This is necessary to implement crash recovery protocols where the order in which pages are written is critical

Prefetching and Double Buffering

- Some DBMS buffer managers predict page requests
 - And fetch pages into the buffer *before* they are requested
 - Known as prefetching
 - Pages are available in the buffer when they are requested, and
 - If the pages to be prefetched are contiguous, the retrieval will be faster than if they had been retrieved individually
 - If the pages are not contiguous, retrieval may still be faster as access to them can be efficiently scheduled
- Prefetching does require additional main memory buffers

Double Buffering

- Prefetching can be combined with buffering
 - Where two processes are interleaved to improve performance
- Data from one process is read into main memory as the CPU acts on the second
 - Disk I/O processors are separate from the CPU so these tasks can be performed in parallel
- Double buffering is an important memory management technique

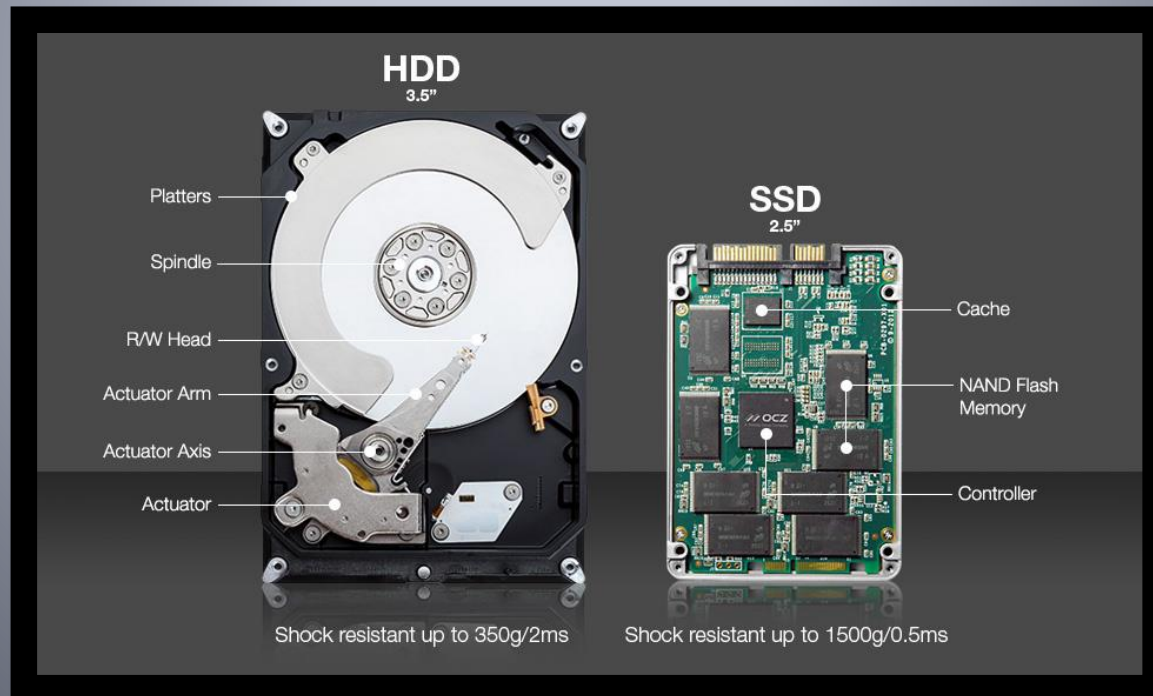


Performance Strategies

- Organize data by cylinders
 - Related data should be stored "close to" each other
- Use a RAID system to improve efficiency or reliability
 - Multiple disks and striping improves efficiency
 - Mirroring or redundancy improves reliability
- Schedule requests using the elevator algorithm
 - Reduces disk access time for random reads and writes
 - Most effective when there are many requests waiting
- Prefetch data in large chunks and use double buffering
 - Speeds up access when needed blocks can be predicted but requires more main memory buffers

Solid State Drives

5.2



Solid State Drives

- Most Solid State Drives (SSDs) use NAND flash memory and do not contain moving parts like an HDD
 - Accessing an SSD does not require seek time or rotational latency and they are therefore considerably faster
 - Flash memory is non-volatile memory that is used by smart-phones, mp3 players and thumb (or USBO) drives
- Like HDDs SSDs contain a controller whose functions include
 - Read and write caching
 - Encryption
 - Error detection and correction
 - Wear leveling
 - Evenly distributing reads and writes across the disk



NAND Flash Memory

- NAND flash architecture is similar to a NAND (negated and) logic gate hence the name
 - Only able to read and write data one page at a time
 - NAND flash memory is non-volatile
 - It does not require power to retain memory
 - Earlier SSDs used DRAM which necessitated an internal power supply
- There are different types of NAND SSD
 - SSD with multiple charge levels
 - Multi-level cell (MLC), Triple Level Cell (TLC), Quad Level Cell (QLC)
 - Single-level cell (SLC)

An SSD *page* is a few kB in size

Re-writing SSD data requires first erasing the entire block – i.e. multiple pages

MLC, TLC, QLC SSDs

- MLC, TLC and QLC cells can store multiple different charge levels
 - And contain more than one bit
 - With four charge levels a cell can store 2 bits
 - With eight, 3 bits, etc.
 - Reading is more complex but more data can be stored per cell
- MLC, TLC and QLC SSDs are cheaper per byte than SLC SSDs
 - However write performance is worse
 - And their lifetimes are shorter

SLC	MLC	TLC	QLC
0	00	000	0000
	01	001	0001
010		0010	
1	10	0011	0011
		0100	0100
	11	0101	0101
		0110	0110
	0111	0111	
	1000	1000	
	1001	1001	
	1010	1010	
	1011	1011	
	1100	1100	
	1101	1101	
1110	1110		
1111	1111		

SLC SSDs

- SLC cells can only store a single charge level
 - They are therefore *on* or *off*, and can contain only one bit
- SLC drives are less complex
 - More reliable with a lower error rate
 - Faster since it is easier to read or write a single charge value
- But SLC drives are more expensive
 - And were typically used for enterprise rather than home use

The trend appears to be towards using TLCs for enterprise solutions rather than SLC

Larger storage size and use of cache or other memory types to mitigate their disadvantages

EFDs

- An EFD is an Enterprise Flash Drive
 - The term was introduced by EMS Corporation
- EFDs are designed for applications requiring high performance
 - Reliability
 - Energy efficiency
 - Consistent performance
- There is no standard for what defines an EFD
 - So SSD manufacturers can claim that their high performing products are EFDs
 - Without meeting any particular requirements

New Technology

- SSD technology continues to evolve
 - Consumer drives are decreasing in price per byte
 - There are multiple types of SSD and multiple form factors
- 3D XPoint Intel Optane
 - New technology
 - Developed by Intel and Micron
 - Multi-purpose
 - SSDs
 - Cache for other drives
 - Persistent memory
 - Faster than NAND flash but more expensive per byte



SSD Performance

- SSD access is entirely electronic and so no seek time or rotational delay is incurred
- Both reads and writes are faster than HDDs
 - However flash memory must be erased before it is written, and entire blocks must be erased
 - Referred to as write amplification
- The performance increase over HDDs is greatest for random reads
- SSDs are considerably more durable than HDDs
 - Primarily due to the lack of moving parts

Appendix

Memory Comparison

Type	Capacity	Access Time	Max Bandwidth	Cost
RAM	4GB – 1TB	30ns	35GB/s	\$100 – \$20,000
SSD	64GB – 1TB	50µs	750MB/s	\$50 – \$600
USB Stick	4GB – 512GB	100µs	50MB/s	\$2 – \$200
HDD	600GB – 8TB	10ms	200MB/s	\$70 – \$500
Optical	50GB – 100GB	180ms	72MB/s	\$100
Tape	2.5TB – 8.5TB	10s – 80s	40 – 250MB/s	\$2,500 – \$30,000
Tape Jukebox	25TB – 2.1m TB	10s – 80s	250MB/s – 1.2PB/s	\$3,000 – \$1m

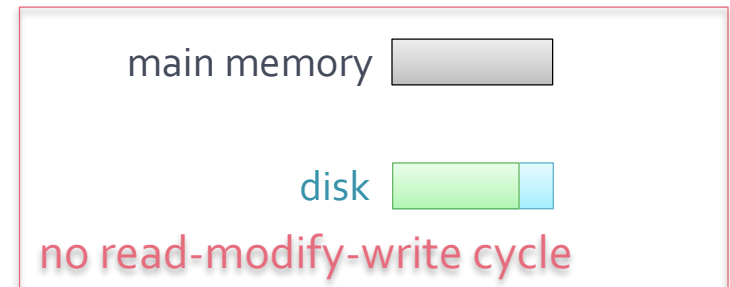
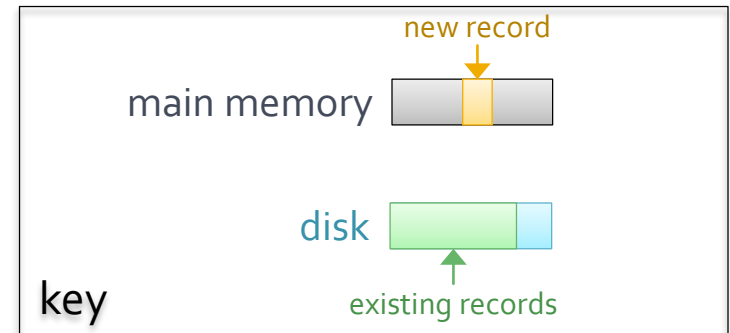
2014 figures – per text (Elmasri, Navathe)

Note that the capacity figures are already quite out of date

In particular RAM and SSD capacity have increased while cost decreased

Writing Data to a Disk 1 (.pdf)

- Most writes require an initial read of a disk block
 - Even blind writes
- Consider inserting a new record into a disk block
 - Which contains existing records
- The block must be read first
 - To preserve the existing data
 - Referred to as the read-modify-write cycle



Writing Data to a Disk 2 (.pdf)

- Most writes require an initial read of a disk block
 - Even blind writes
- Consider inserting a new record into a disk block
 - Which contains existing records
- The block must be read first
 - To preserve the existing data
 - Referred to as the read-modify-write cycle

