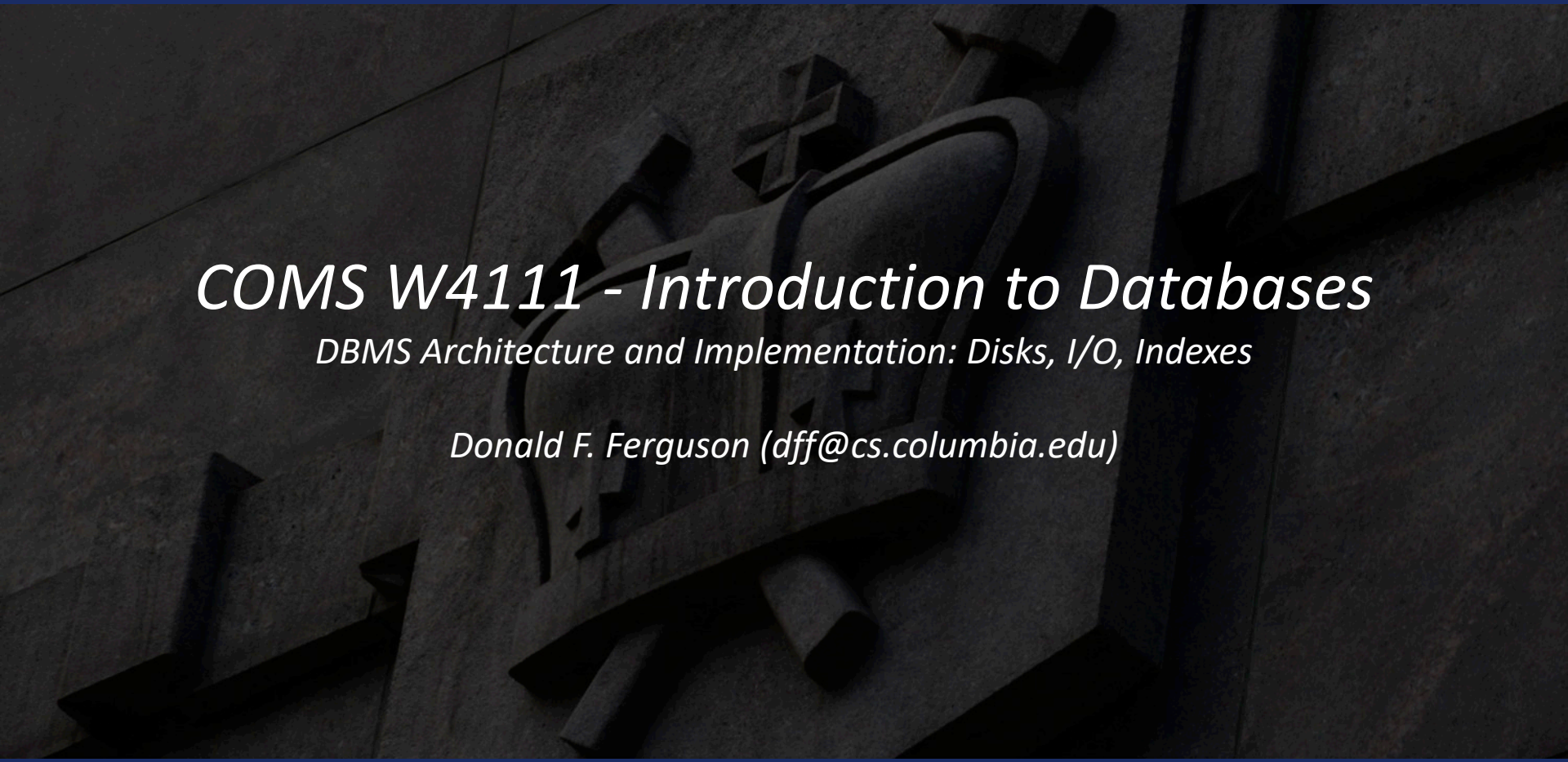
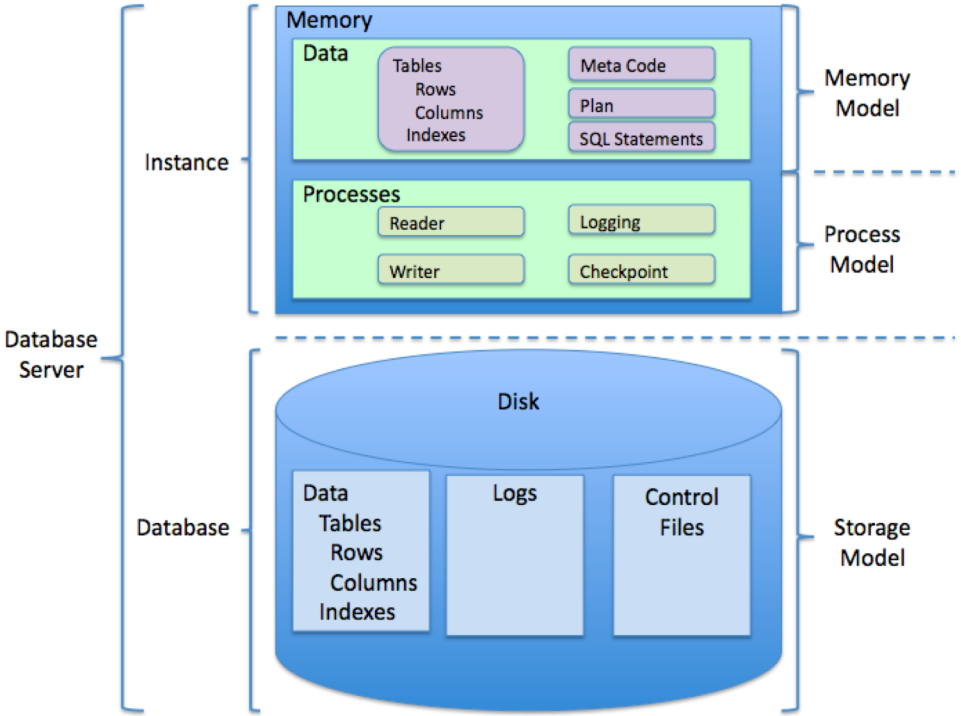


COMS W4111 - Introduction to Databases

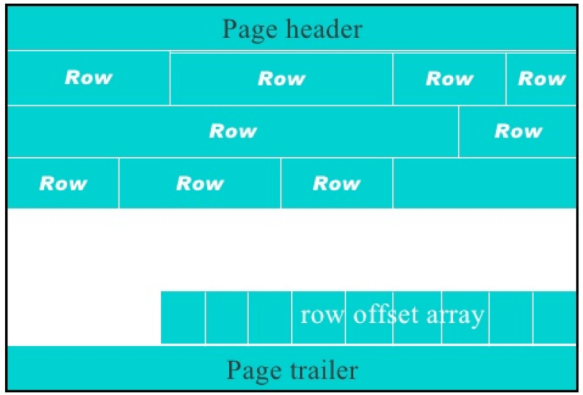
DBMS Architecture and Implementation: Disks, I/O, Indexes

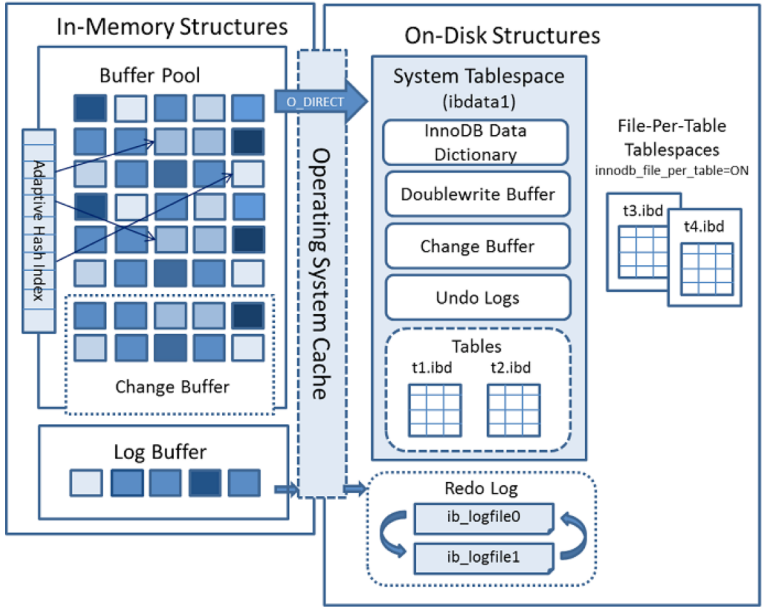
Donald F. Ferguson (dff@cs.columbia.edu)



InnoDB Pages

A page consists of: a page header, a page trailer, and a page body (rows or other contents).





INNOBASE

Module II – DBMS Architecture and Implementation

Overview and Reminder

Database Management System Reminder

What is a database? In essence a database is nothing more than a collection of information that exists over a long period of time, often many years. In common parlance, the term *database* refers to a collection of data that is managed by a DBMS. The DBMS is expected to:

1. Allow users to create new databases and specify their *schemas* (logical structure of the data), using a specialized *data-definition language*.

- User view covered for the relational model.
- We will cover the implementation technology is module II.

Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina (Author), Jeffrey D. Ullman (Author), Jennifer Widom (Author)

Database Management System Reminder

Covered User View

2. Give users the ability to *query* the data (a "query" is database lingo for a question about the data) and modify the data, using an appropriate language, often called a *query language* or *data-manipulation language*.

3. Support the storage of very large amounts of data — many terabytes or more — over a long period of time, allowing efficient access to the data for queries and database modifications.

4. Enable *durability*, the recovery of the database in the face of failures, errors of many kinds, or intentional misuse.

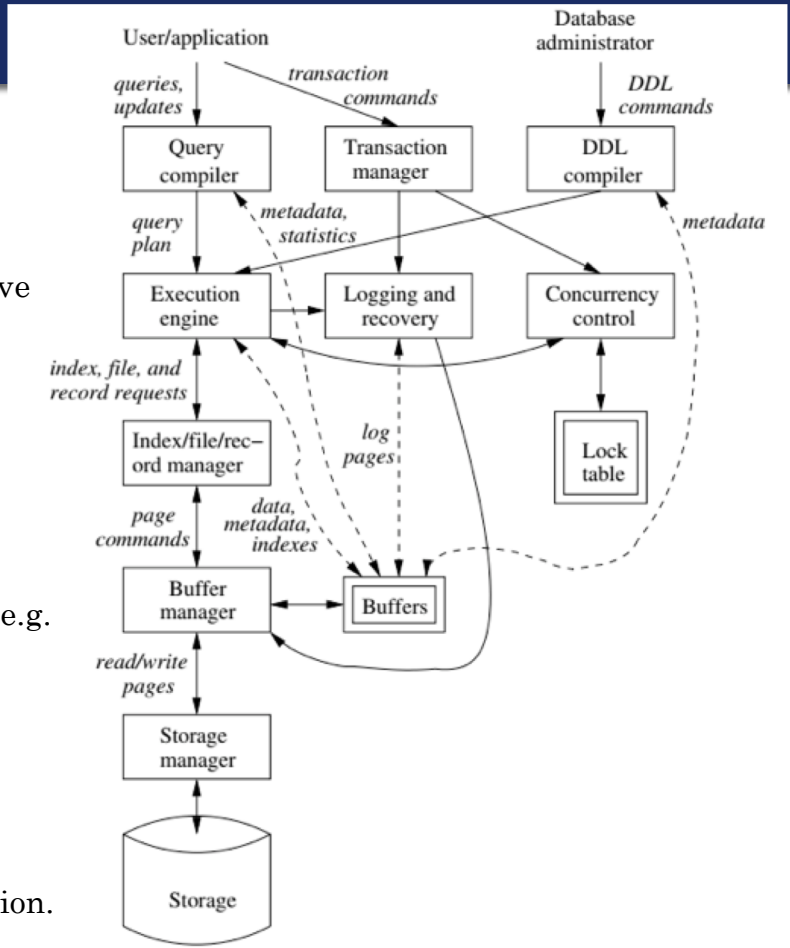
5. Control access to data from many users at once, without allowing unexpected interactions among users (called *isolation*) and without actions on the data to be performed partially but not completely (called *atomicity*).

Implementation technology is focus for next part of course.

Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina (Author), Jeffrey D. Ullman (Author), Jennifer Widom (Author)

DBMS Subsystems

- Query compiler
  - Parses *declarative query statement* and compiles into
  - *Optimized, imperative* execution plan (language)
- Execution engine runs plan and schedules sub-operations to improve performance.
- Index/file/record manager manages data structures that improve locating specific rows.
- Buffer manager controls movement of data into/out of memory to improve performance and ensure atomicity and durability.
- Storage manager places/retrieves rows on/from secondary storage, e.g. disks.
- Transaction manager schedules DB operations to ensure isolation.
- Logging and recovery manager ensure atomicity, isolation and durability in the face of failures.
- Concurrency control support transaction manager to achieve isolation.



Disks, I/O, Indexes

Disks as Far as the Eye Can See

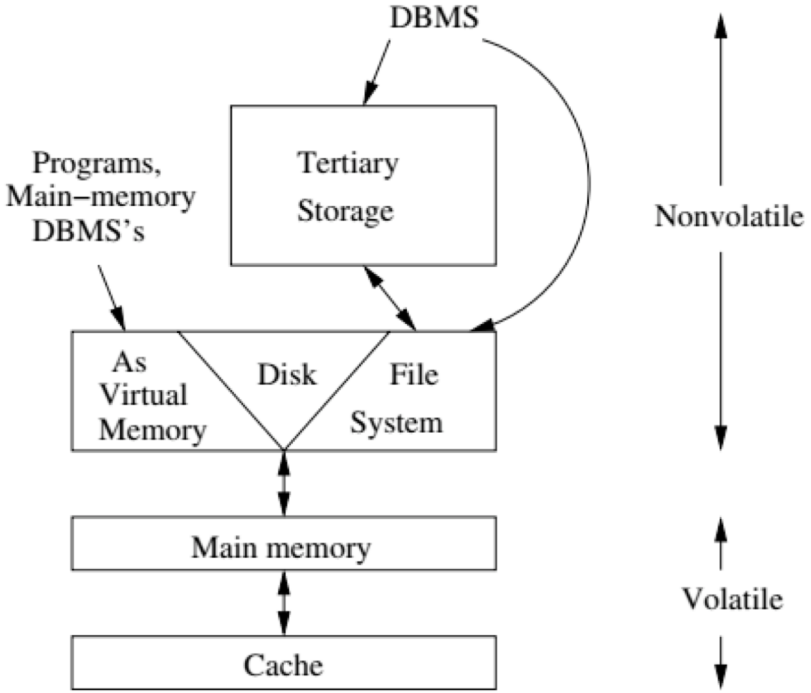


Figure 13.1: The memory hierarchy

Slower
Cheaper
Unlimited

Faster
More expensive
Limited

Memory Hierarchy

Storage Technology

Price, Performance & Capacity

Technologies	Capacity (GB)	Latency (microS)	IOPs	Cost/IOPS ($)	Cost/GB ($)
Cloud Storage	*Unlimited*	*60,000*	*20*	*17c/GB*	*0.15/month*
Capacity HDDs	*2,500*	*12,000*	*250*	*1.67*	*0.15*
Performance HDDs	*300*	*7,000*	*500*	*1.52*	*1.30*
SSDs (write)	*64*	*300*	*5000*	*0.20*	*13*
SSDs (read only)	*64*	*45*	*30,000*	*0.03*	*13*
DRAM	*8*	*0.005*	*500,000*	*0.001*	*52*

- These numbers are ancient.
- Looking for more modern numbers.
- But, does give an idea of
  - Price
  - Performance
- The general observation is that
  - Performance goes up 10X/level.
  - Price goes up 10x per level.
- Note: One major change is improved price performance of SDD relative to HDD for large data.

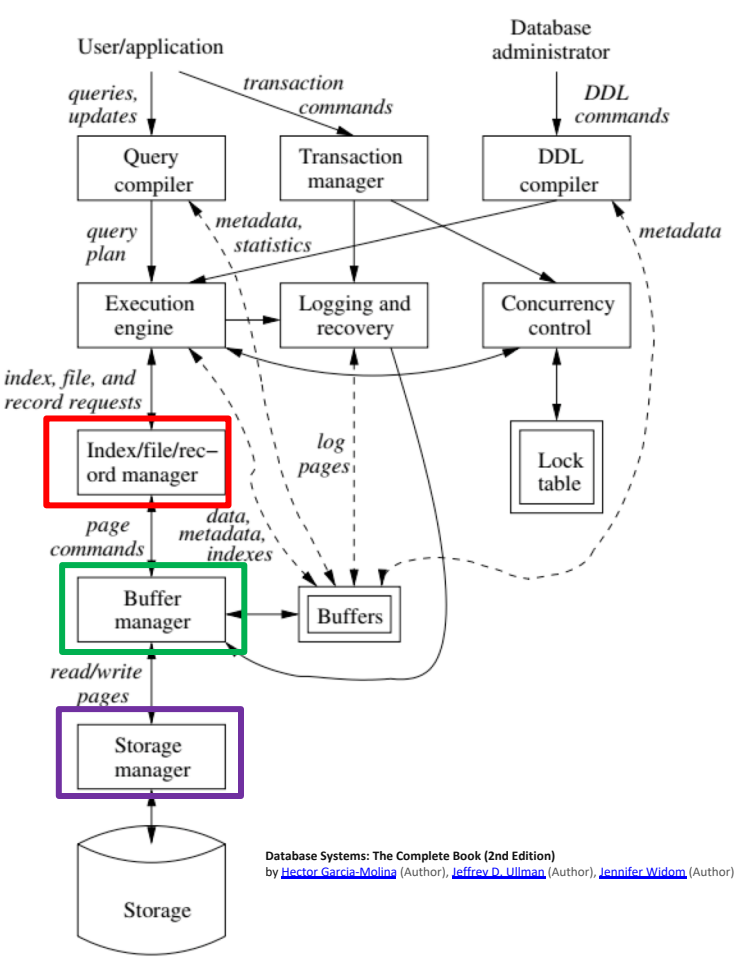
Another Perspective: Normalized Access Times

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

- These numbers are ancient.
- Looking for more modern numbers.
- But, does give an idea of relative performances
- Assume 1 CPU (clock cycle) is 0.3 ns
  - Maps to 3 Ghz
  - My MacBook is 4x2.7 Ghz
- Normalize by mapping
  - 1 CPU cycle to
  - 1 Second
- Represents access latency
  - Actual (old) values.
  - Normalized value

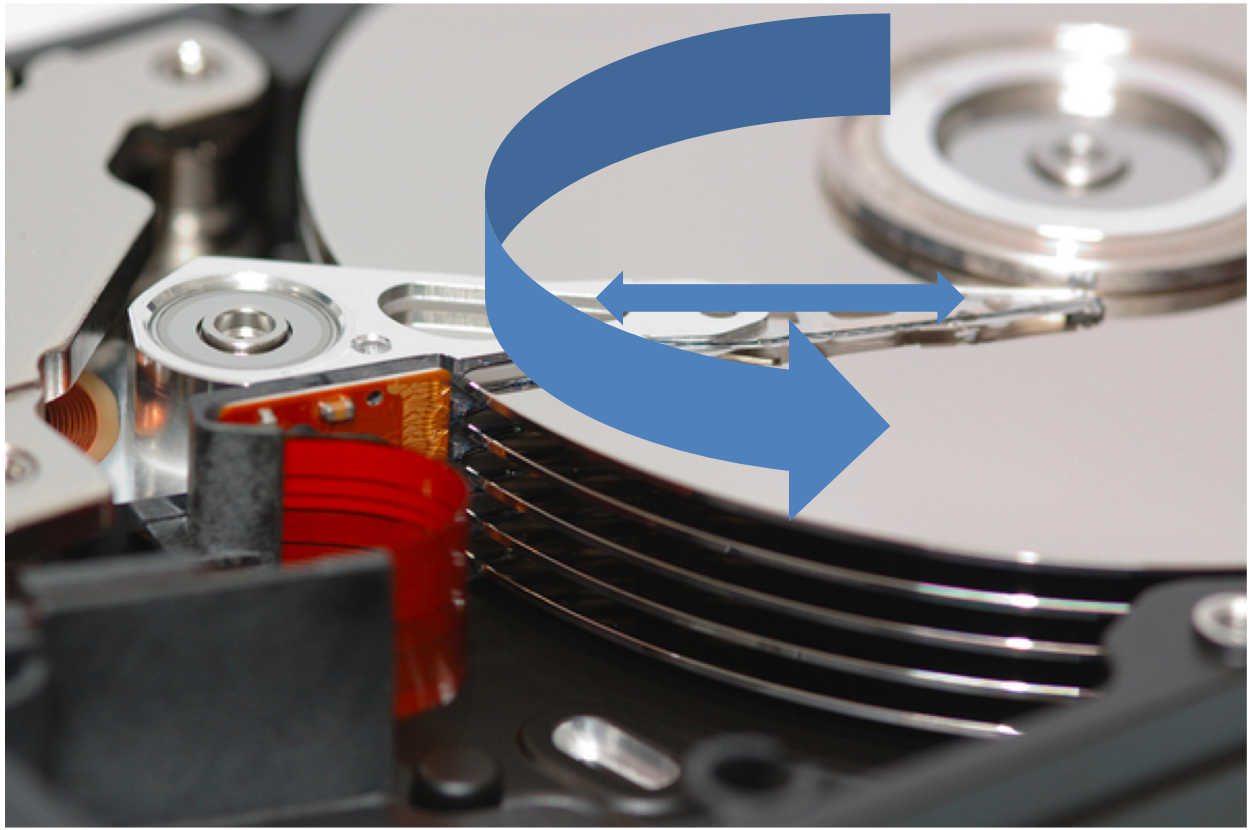
Data Management

- Find things quickly.
- Access things quickly.
- Load/save things quickly.



Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina (Author), Jeffrey D. Ullman (Author), Jennifer Widom (Author)

Hard Disk Drive



Disk Configuration

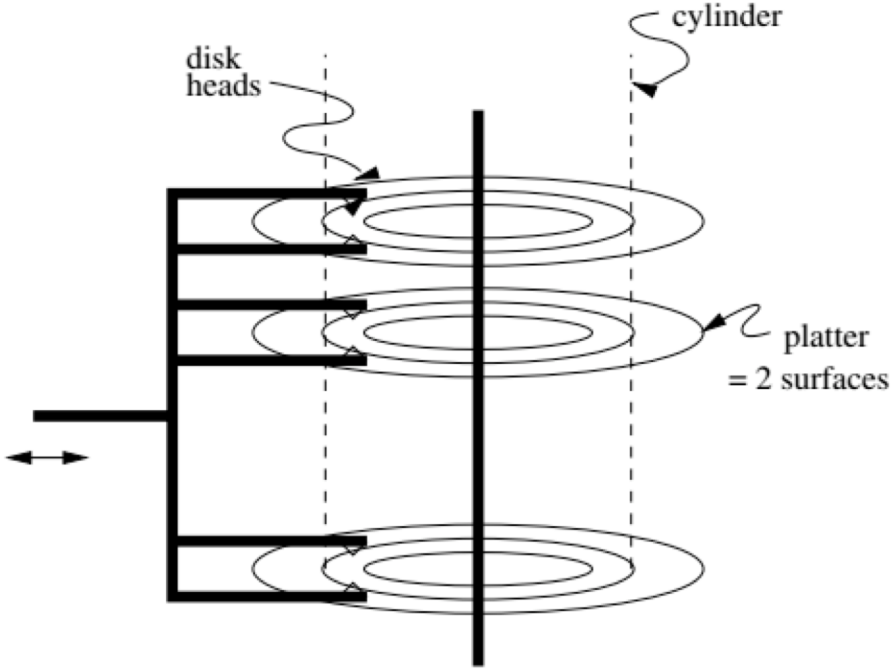


Figure 13.2: A typical disk

Components of disk I/O delay

Seek: Move head to cylinder/track.

Rotation: Wait for sector to get under head

Transfer: Move data from disk to memory.

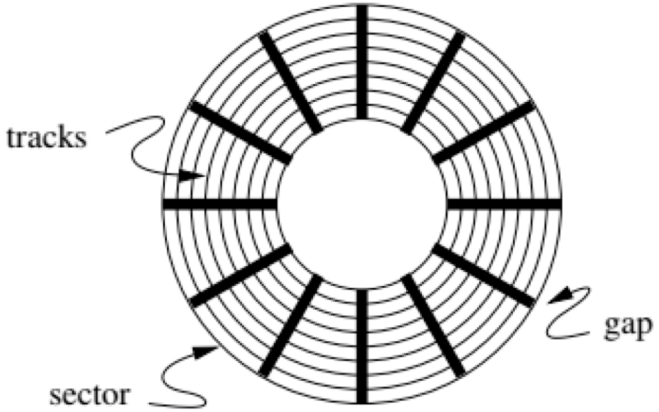


Figure 13.3: Top view of a disk surface

Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina (Author), Jeffrey D. Ullman (Author), Jennifer Widom (Author)

Implications

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

- I grew up in a tiny town in a very rural area.
- The ”stores” were very far away and spread out.
- If we were out of milk and we needed a cup of milk, we
  - Bought all the milk we would need for a while
  - AND
  - Anything else we needed or might need that was in the store area.
- Disk I/O is the same way
  - Getting a block is as cheap as getting a byte.
  - Try to optimize your travel time by planning your trip.

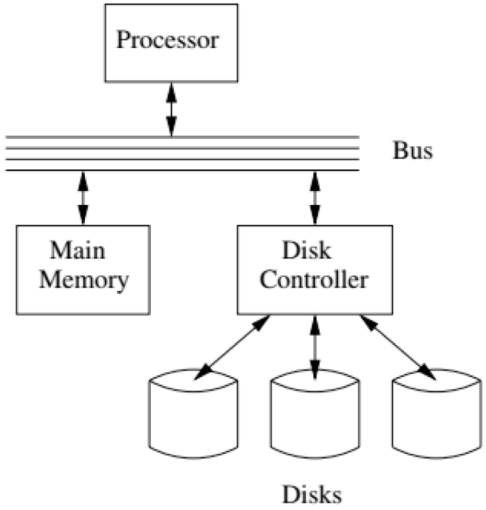
Data Access Patterns

- Think about accessing tuples in a database. The access patterns are not completely random. For example,
  - SELECT … ORDER BY … ➔ *I look for things in sequence.*
  - SELECT … WHERE last_name="Smith" ➔ *I look for sets of things.*
  - CREATE VIEW AS SELECT … FROM JOIN … ON A.x=B.y ➔ *I look for things in pairs of sets.*
- Optimizing I/O performance
  - Group records into blocks if accessed together.
  - Group blocks onto the same sector/cylinder if accessed together/sequentially ➔ Eliminates seek time.
  - Schedule access:
    - If I have to read cylinders 1, 9, 2, 8, 3
    - A better ordering is 1,2,3,9,8.
  - Stripe across disks: Read one block from 10 disks instead one 10 from 1.
  - Pre-fetch and preload.

Disk Scheduling

- An example – *Elevator Algorithm*
  - Assume
    - The seek time from track n to m is ABS(n-m).
    - The I/O request queue is for sectors on tracks 1, 9, 2, 8, 3.
    - The head is currently on track 5.
  - Total seek time for FIFO processing
    - Seek order is 5→1→9→2→8→3
    - Total seek time is ABS(5-1) + ABS(1-9) + ABS(9-2) + ABS(2-8) + ABS(8-3) = 30
  - If the head goes all the way up and all the way down, like an elevator,
    - The seek order 5→8→9→3→2→1
    - Total seek time is ABS(5-8) + ABS(8-9) + ABS(9-3) + ABS(3-2) + ABS(2-1) = 12
- There is a rich body of research and algorithms on optimizing
  - Blocks on tracks/sectors.
  - Seek/rotation optimization.

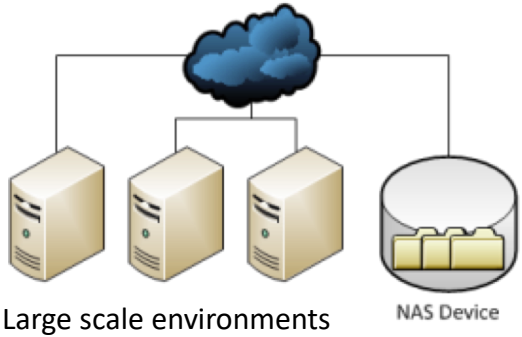
I/O Architecture



How we normally think of disks and I/O.

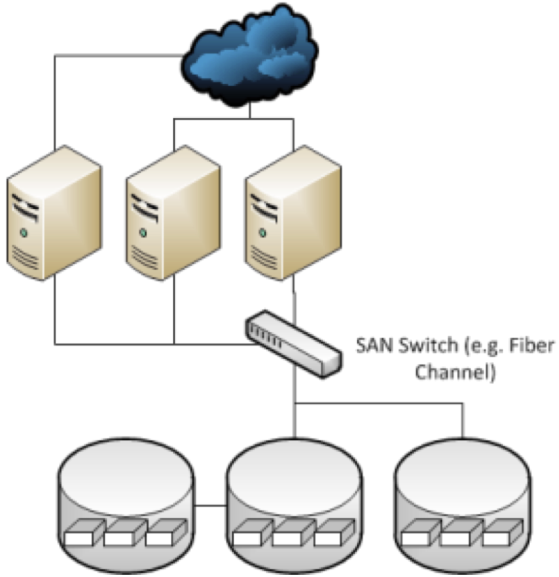
Network Attached Storage

- Shared storage over shared network
- File system
- Easier management



Storage Area Network

- Shared storage over dedicated network
- Raw storage
- Fast, but costly

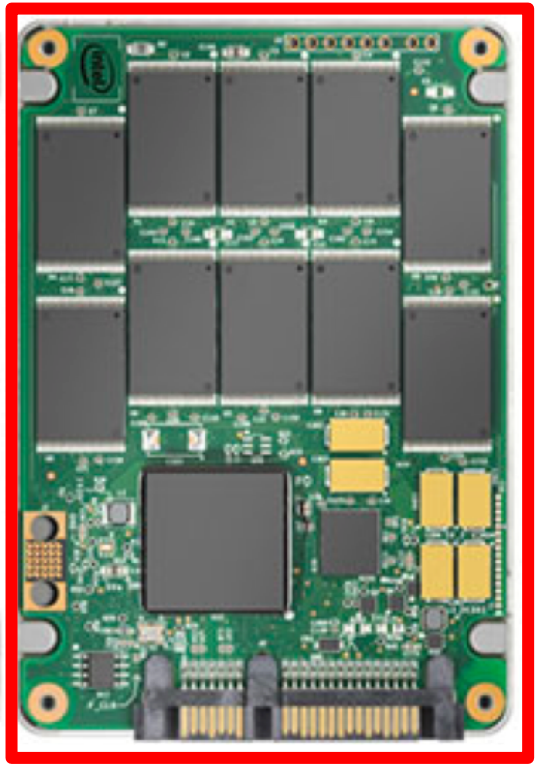


Large scale environments

- The bus-controller connection is over some kind of network.
- The disk controller is at the disks, and basically a "computer" with SW.
- Network is either
  - Standard communication network, or
  - Highly optimized I/O network.

Hard Disk versus Solid State Disk

Hard Disk Drive



Solid State Drive

I/O Architecture

Solid State Disks are becoming increasingly preferred over HDD.

1. Faster (no rotation or seek latency).
2. No moving parts → more reliable.

Even for HDDs

1. Disk controllers supporting DBs are very sophisticated.
2. A lot of the placement and scheduling optimization occurs in controller.

Seek and latency considerations are less important to the DB engine, but …

- The disk is still very far away.
- Some of the optimization are still important.

And, availability is still an important consideration.

I/O Architecture

Redundant
Array
of
Independent
Disks

(RAID)

RAID (https://en.wikipedia.org/wiki/RAID)

"RAID (redundant array of independent disks) is a data storage virtualization technology that combines multiple physical disk drive components into a single logical unit for the purposes of data redundancy, performance improvement, or both. (…)

RAID 0 consists of striping, without mirroring or parity. (…)
RAID 1 consists of data mirroring, without parity or striping. (…)
RAID 2 consists of bit-level striping with dedicated Hamming-code parity. (…)
RAID 3 consists of byte-level striping with dedicated parity. (…)
RAID 4 consists of block-level striping with dedicated parity. (…)
RAID 5 consists of block-level striping with distributed parity. (…)
RAID 6 consists of block-level striping with double distributed parity. (…)

Nested RAID

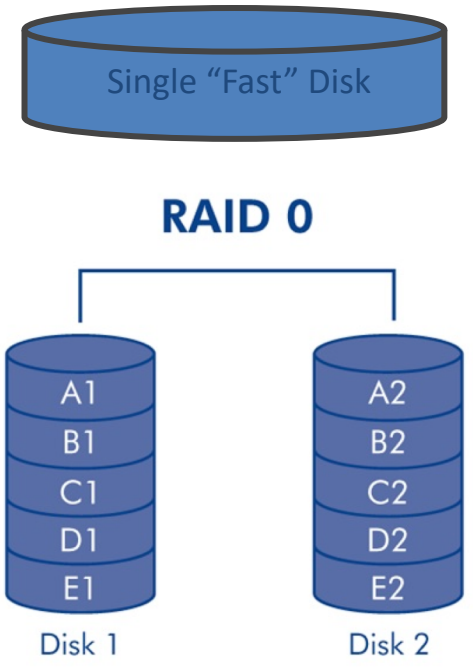
- RAID 0+1: creates two stripes and mirrors them. (…)
- RAID 1+0: creates a striped set from a series of mirrored drives. (…)
- **JBOD RAID N+N:** With JBOD (*just a bunch of disks*), (…)"

"RAID
ation technology
that co
it for the
purpos
RAID
RAID
RAID
RAID
RAID
RAID
Nested
–
–
–

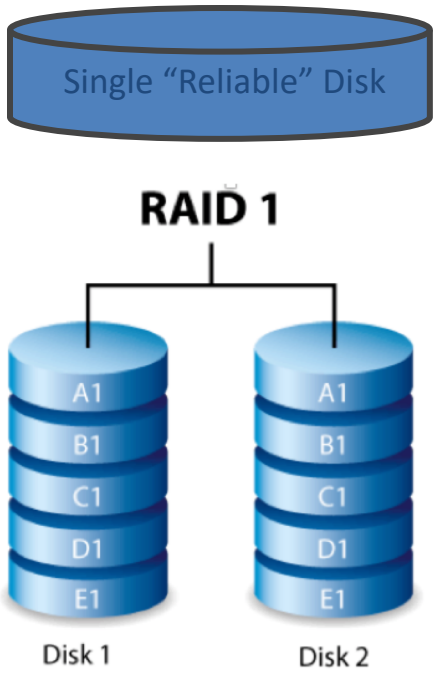
- Clearly, we got pretty excited and inventive defining all of these options.
- My experience and some independent analysis indicates that the most common levels are
  - RAID-0: Striping for performance.
  - RAID-1: Duplication for availability.
  - RAID-5: Striping and parity availability.
- And a lot of environments do this in software using "just a bunch of disks."
- And you still have to deal with, "Don spilled coffee of the disk array controller."

RAID-0 and RAID-1

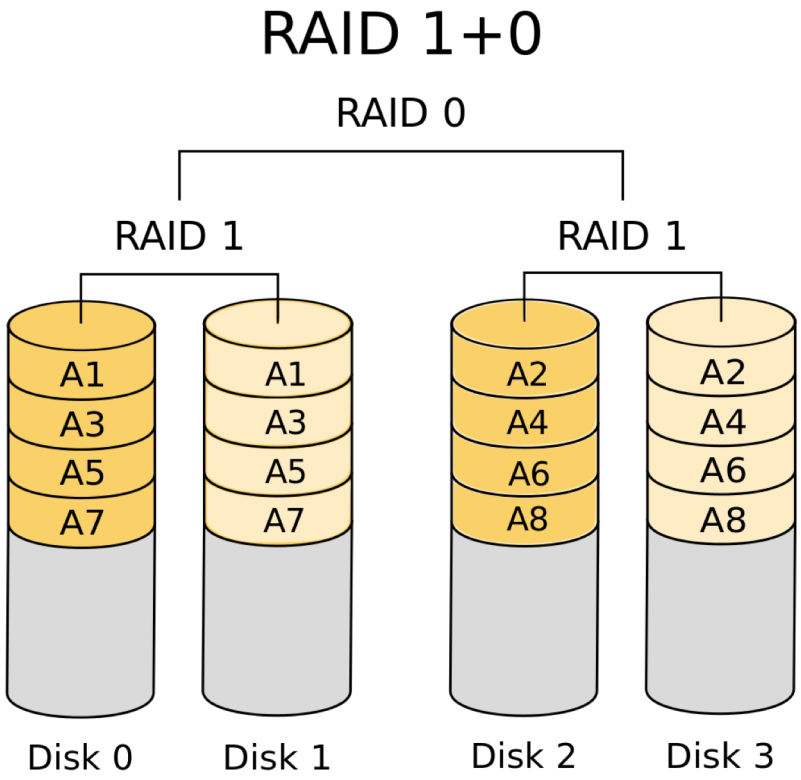
Two physical disks make
one single, logical **fast** disk



Two physical disks make
one single, logical **reliable** disk



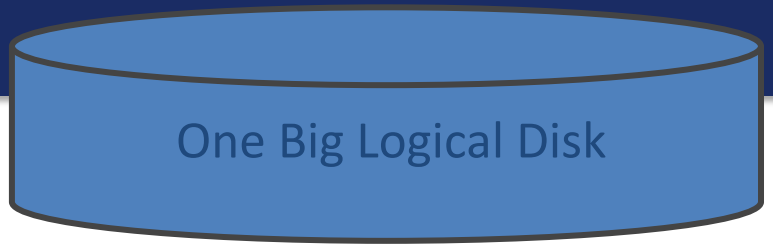
Mixed RAID Modes



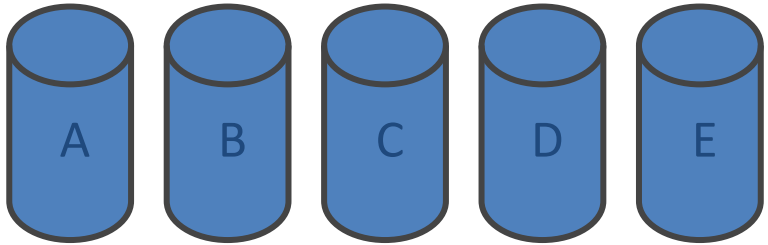
Stripe
And
Mirror

RAID-5

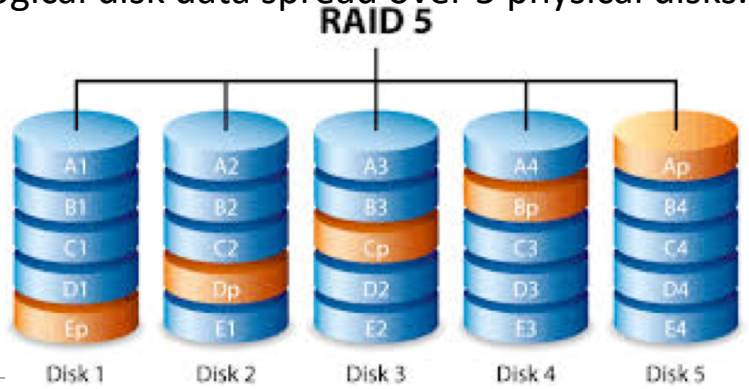
- Improved performance through parallelism
  - Rotation/seek
  - Transfer
- Availability uses *parity blocks*
  - Suppose I have 4 different data blocks on the logical drive A: A1, A2, A3, A4.
  - Parity function: **Ap** = P(A1,A2,A3,A4)
  - Recovery function: A2=R(**Ap**,A1,A3,A4)
- During normal operations:
  - Read processing simply retrieves block.
  - Write processing of A2 updates A2 and Ap
- If an individual disk fails, the RAID
  - Read
    - Continues to function for reads on non-missing blocks.
    - Implements read on missing block by recalculating value.
  - Write
    - Updates block and parity block for non-missing blocks.
    - Computes missing block, and calculates parity based on old and new value.
  - Over time
    - "Hot Swap" the failed disk.
    - Rebuild the missing data from values and parity.



Is actually 5 smaller "logical" disks.



Logical disk data spread over 5 physical disks.



Very Simple Parity Example

- Even-Odd Parity
  - b[i] is an array of bits (0 or 1)
  - P(b[i]) =
    - 0 is an even number of bits = 1. {P([0,1,1,0,1,1])= 0
    - 1 if an odd number of bits = 1. {P(0,0,1,0,1,1]= 1
  - Given an array with one missing bit and the parity bit, I can re-compute the missing bit.
    - Case 1: [0,?,1,0,1,1] has P=0. There must be an EVEN number of ones and ?=1.
    - Case 2: [0,?,1,0,1,1] has P=1. There must be an ODD number of ones and ?=0.
- Block Parity applies this to a set of blocks bitwise
  - A1=[0 1,0,0,1,1]
  - A2=[1 1,1,0,0,0]
  - A3=[0 0,0,1,0,1]
  - Pa=[1 0,1,1,1,0]

➔ If I am missing a block and have the parity block, I can re-compute the missing block bitwise from remaining blocks and parity block.

Database Address Space

Data Address Space (Overly Simply)

- The block/storage system represents block addresses with
  - A logical ID
  - That is simply a string, e.g. UUID or has some structure.
- There is a logical address to physical address mapping table that maps a logical address to a physical address
  - Block/storage engine ID.
  - An opaque address string that *the specific engine understands and manages*.
- For example, in a storage area network, the physical address might be
  - IP address of storage server.
  - ID of disk attached to the server.
  - {Cylinder, Head Number, Sector}

Any reference to a block, e.g. in an index node, uses the logical address.

Figuring out where, when and why to put a block in a specific engine/domain is an important optimization function in DMBS.

Dat[illegible]verly Simply)

Straightforward:
- s-sever11.nas.myco.com
- /var/dev1

- [illegible]ents block addresses with
  - [illegible]ply a string, e.g. UUID or has some structure
- There i[illegible]ical address to physical addr[illegible]
  logical ad[illegible]s to a physical address
  - Block/s[illegible]ge engine ID.
  - An opaq[illegible]address string that *the specific e[illegible]*
- For example, [illegible]n a storage area network, the [illegible]ess might be
  - IP address of storage server.
  - ID of disk attached to the server.
  - {Cylinder, Head Number, Sector}

Most devices surface *Logical Block Addressing.*

Any reference to a block, e.g. in an index node, uses the logical address.

Figuring out where, when and why to put a block in a specific engine/domain is an important optimization function in DMBS.

Logical Block Addressing (https://gerardnico.com/wiki/data_storage/lba)

3 - The LBA scheme

LBA	C	H	S
0	0	0	0
1	0	0	1
2	0	0	2
3	0	0	3
4	0	0	4
5	0	0	5
6	0	0	6
7	0	0	7
8	0	0	8
9	0	0	9
10	0	1	0
11	0	1	1
12	0	1	2
13	0	1	3
14	0	1	4
15	0	1	5
16	0	1	6
17	0	1	7
18	0	1	8
19	0	1	9
Cylinder 0			

LBA	C	H	S
20	1	0	0
21	1	0	1
22	1	0	2
23	1	0	3
24	1	0	4
25	1	0	5
26	1	0	6
27	1	0	7
28	1	0	8
29	1	0	9
30	1	1	0
31	1	1	1
32	1	1	2
33	1	1	3
34	1	1	4
35	1	1	5
36	1	1	6
37	1	1	7
38	1	1	8
39	1	1	9
Cylinder 1			

- CHS addresses can be converted to LBA addresses using the following formula:

$$LBA = ((C \times HPC) + H) \times SPT + S - 1$$

where,

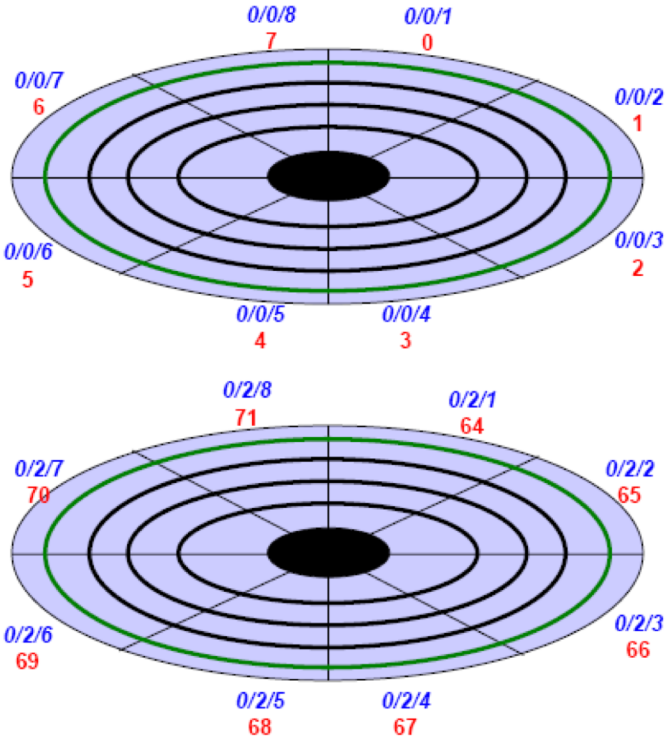
- C, H and S are the cylinder number, the head number, and the sector number
- LBA is the logical block address
- HPC is the number of heads per cylinder
- SPT is the number of sectors per track

Devices have configuration and metadata APIs that allow storage manager to

- Map between LBA and CHS
- To optimize block placement
- Based on access patterns, statistics, data schema, etc.

Another View

(http://www.c-jump.com/CIS24/Slides/DiskDrives/D01_0150_chs_vs_lba.htm)



Simple example: 4 sides, 8 sectors/side, 4 tracks/side (cylinders) yields 128 total sectors...

The green track is cylinder 0. On the first side (side 0), sectors are numbered 0/0/1 thru 0/0/8 in CHS format, or 0 thru 7 in LBA.

On cylinder 1 of the first side, sectors are numbered 1/0/1 thru 1/0/8 (8 thru 15).

The last sector on side 0 is numbered 4/0/8 (31).

Second side (side 1) starts:	*0/1/1*	*32*
Second side ends:	*4/1/8*	*63*
Third side (side 2) starts:	*0/2/1*	*64*
Third side ends:	*4/2/8*	*95*
Fourth side (side 3) starts:	*0/3/1*	*96*
Fourth side ends:	*4/3/8*	*127*

Summary

- There is a broad category of storage devices
  - "Just a disk"
  - RAID-0, RAID-1, RAID-5, RAID-X/Y
  - Network/cloud addressable devices
  - SSD
- A device has configuration information, often via APIs
  - Device geometry
  - Reliability metrics
  - IOPs, and sub-elements (seek, rotate, etc).
- Storage management consists of using statistics and query (optimization patterns) to
  - Arrange records into blocks for storage and IO efficiency.
  - Place blocks on devices at LBAs to meet performance and IO objectives.

Summary

- Almost all of this complexity and functionality is hidden from
  - Database developers.
  - Database designers and administrators.
- Data center, cloud and large scale infrastructure architects
  - Evaluate and design disk technology and architecture
  - In combination with other critical technology, e.g.
    - Networking
    - Process/compute nodes.
- Some customer, optimized, advanced data solutions are a combination of
  - Custom, optimized database implementation.
  - Co-designed with underlying infrastructure design.
- The disk arrays and controllers do a lot of the low-level functions, but …
  - The database admin using DB analytics understands the access patterns.
  - And uses DB tools to choose, optimize and configure storage options.

Record and Block Management

Terminology

- A tuple in a relation maps to a *record*. Records may be
  - *Fixed length*
  - *Variable length*
  - *Variable format* (which we will see in Neo4J, DynamoDB, etc).
- A *block*
  - Is the unit of transfer between disks and memory (buffer pools).
  - Contains multiple records, usually but not always from the same relation.
- The *database address space* contains
  - All of the blocks and records that the database manages
  - Including blocks/records containing data
  - And blocks/records containing free space.

Fixed Length Record

- Sample DDL

```
CREATE TABLE `products` (
  `product_id` char(8) NOT NULL,
  `product_name` varchar(16) NOT NULL,
  `product_description` char(8) NOT NULL,
  `product_brand` enum('IBM','HP','Acer','Lenovo','Some really long brand name') NOT NULL,
  PRIMARY KEY (`product_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- Why
  - Is this fixed length? product_brand and product_name are clearly variable length.
  - Isn't char(8) to short for a product description?

Hypothetical Explanations

"In computing, **internationalization and localization** are means of adapting computer software to different languages, regional differences and technical requirements of a target locale.[1] Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes. Localization is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text. Localization (which is potentially performed multiple times, for different locales) uses the infrastructure or flexibility provided by internationalization (which is ideally performed only once, or as an integral part of ongoing development)." (https://en.wikipedia.org/wiki/Internationalization_and_localization)

- Retrieving a product description may require two elements
  - Language code, e.g. (EN, DE, ES, …)
  - Description ID, e.g. 01105432
  - (EN, 01105432) and (DE, 01105432) are the same description in different languages (English and German)
  - The description ID is an attribute of the product.
  - The language is an attribute or property of the user, install location, etc.

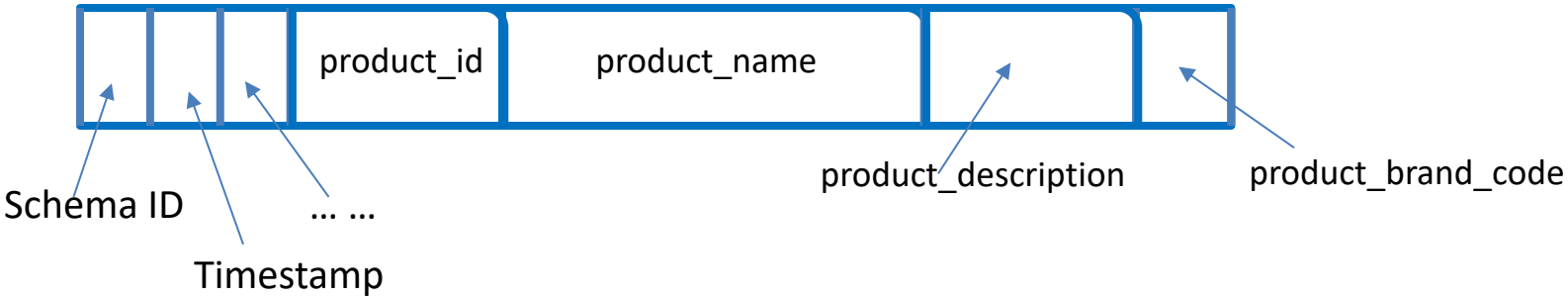
Hypothetical Explanations

- This concept is
  - An element of data modeling and design
  - Not DBMS implementation or data management.
- I introduce the concept because
  - Important, general data model topic we have not covered.
  - Pattern is a reason why many records we think would be variable are often fixed.
    - VARCHAR field in a tuple is often
    - A pointer to a value in some other record.

Hypothetical Explanation

- `product_name` ***varchar***(16) **NOT NULL**
  - (NB: Assumes that the product_name is not localized).
  - Length can be 0 .. 15 characters.
  - But,
    - If the value is almost always between 12 and 16 characters.
    - The database engine may decide to store in a fixed length area.
    - Because the relatively small savings available from treating like a variable length record
    - Does not justify the added complexity involved in variable size record management.
- `product_brand` enum(**'IBM'**,**'HP'**,**'Acer'**,**'Lenovo'**,**'Some really long brand name'**)
  - The DB engine would/could represent with a 1 byte code in a lookup table.
  - This enables simplified, fixed length record management
  - And saves space because
    - 1,000 products in the catalog from 'Some really long brand'
    - Requires 1,000 bytes instead of 27,000 bytes.
  - Repeated values:
    - The same value occurring repeatedly in multiple distinct tuples is common, even when not an enum, e.g.
      - Country names, City names, street names, last names, first names, …
      - Foreign keys in many-to-many associative entities.
    - DBMS storage management engine may detect the situation, and replace values with symbol lookups.

Fixed Length Record



The record has

- Database specific header fields, e.g.
  - Format information
  - Last modified
- Areas for each of the fixed length fields.

Packing Fixed Length Blocks

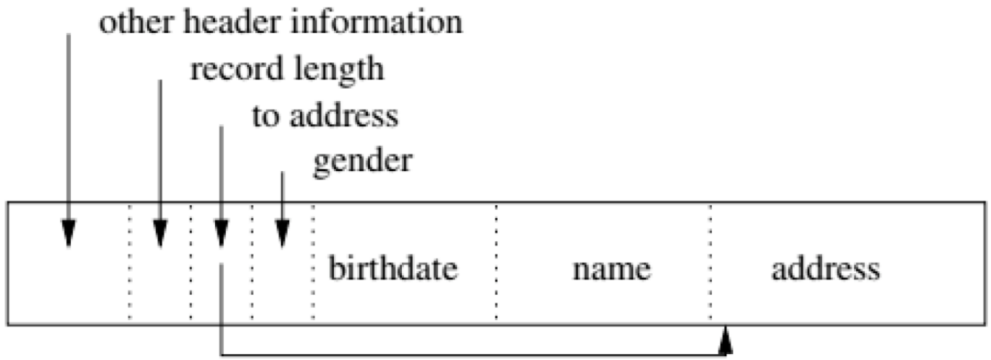
Database Systems: The Complete Book (2nd Edition)
by Hector Garcia-Molina and Jeffrey D. Ullman

Figure 13.17: A typical block holding records

In addition to the records, there is a *block header* holding information such as:

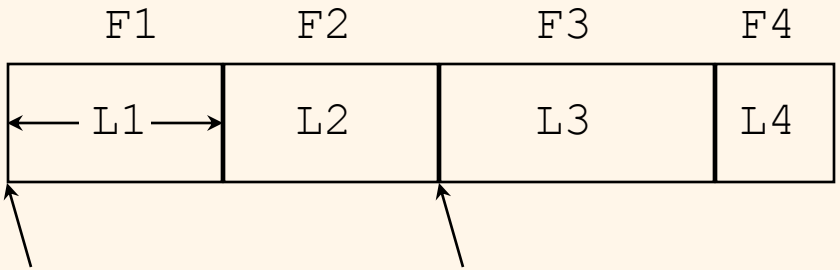
1. Links to one or more other blocks that are part of a network of blocks such as those that will be described in Chapter 14 for creating indexes to the tuples of a relation.
2. Information about the role played by this block in such a network.
3. Information about which relation the tuples of this block belong to.
4. A "directory" giving the offset of each record in the block.
5. Timestamp(s) indicating the time of the block's last modification and/or access.

Variable Length Record



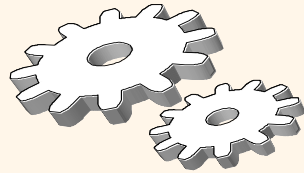
- Same DBMS specific header information as fixed-length, plus
  - Record length.
  - Offset into record for each of the fields.
- Variable sized areas for each of the variable sized fields.

Record Formats: Fixed Length

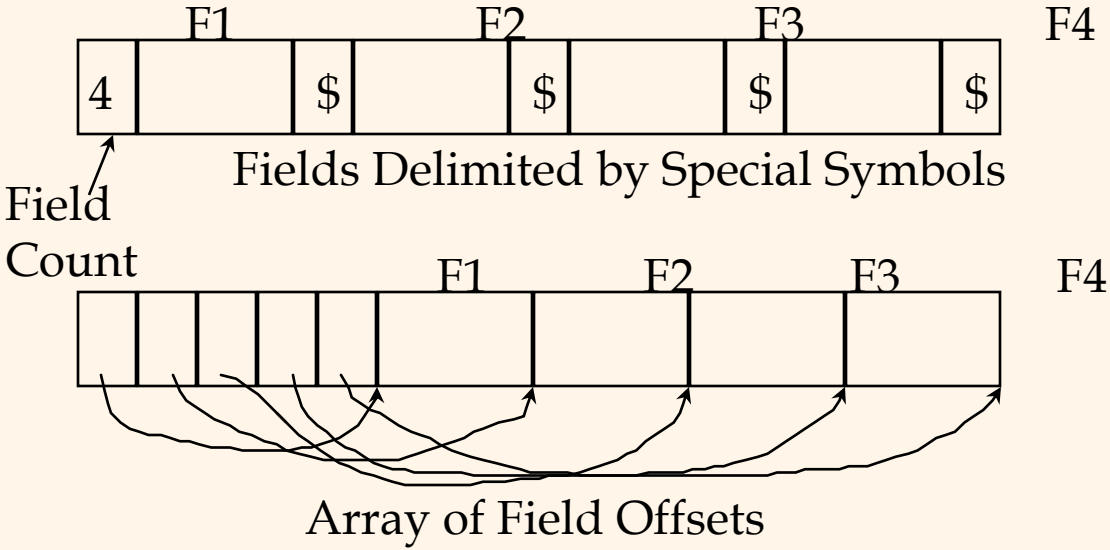


- Information about field types same for all records in a file; stored in *system catalogs.*
- Finding *i'th* field does not require scan of record.

Record Formats: Variable Length



- Two alternative formats (# fields is fixed):



☛ Second offers direct access to i'th field, efficient storage of *nulls* (special *don't know* value); small directory overhead.

INSERT, DELETE, UPDATE

Fixed Length Records

Variable Length Records

INSERT, UPDATE and DELETE

- Is pretty straightforward for fixed length records
- But can get pretty interesting for variable length records, e.g.
  - What if INSERTed R5 is bigger than E1 or E2?
  - How do I expand R2?
  - Deleting R2 leaves a small gap → Fragmentation?

One (Simple) Solution is …

The block does not look like this in memory.

Variable Length Records

It always looks like this in memory …

Variable Length Records

- INSERT always goes into the start of E1.
- DELETE or UPDATE R4:
  - Move (R2,R3) to end of R1
  - If R4 updated, move to end of new position of R3.

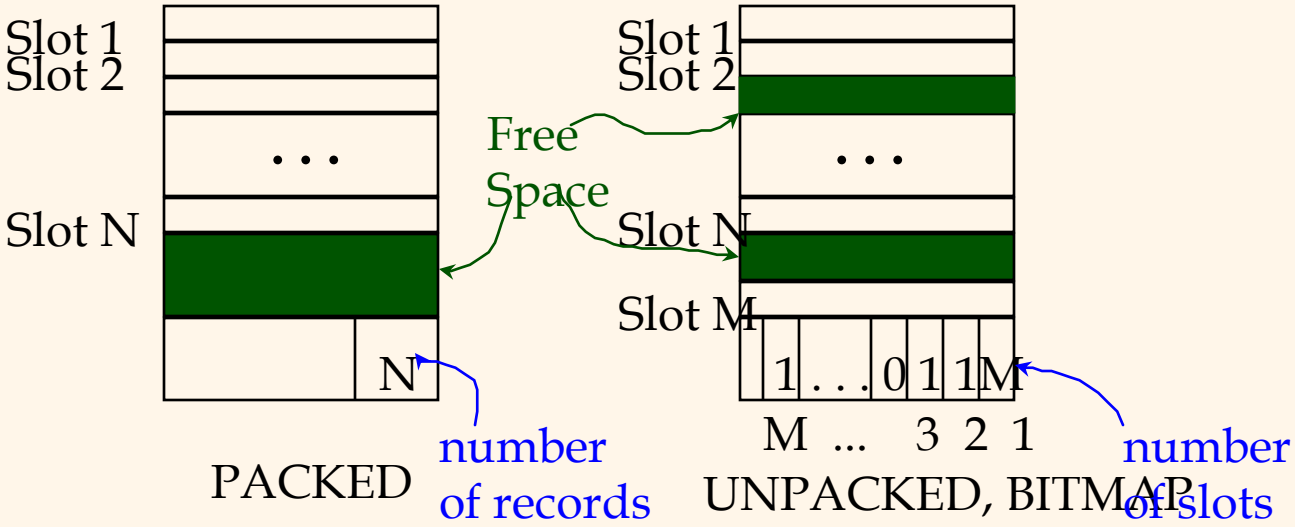
One (Simple) Solution is …

Before UPDATE (and resize) of R4

After UPDATE (and resize) of R4

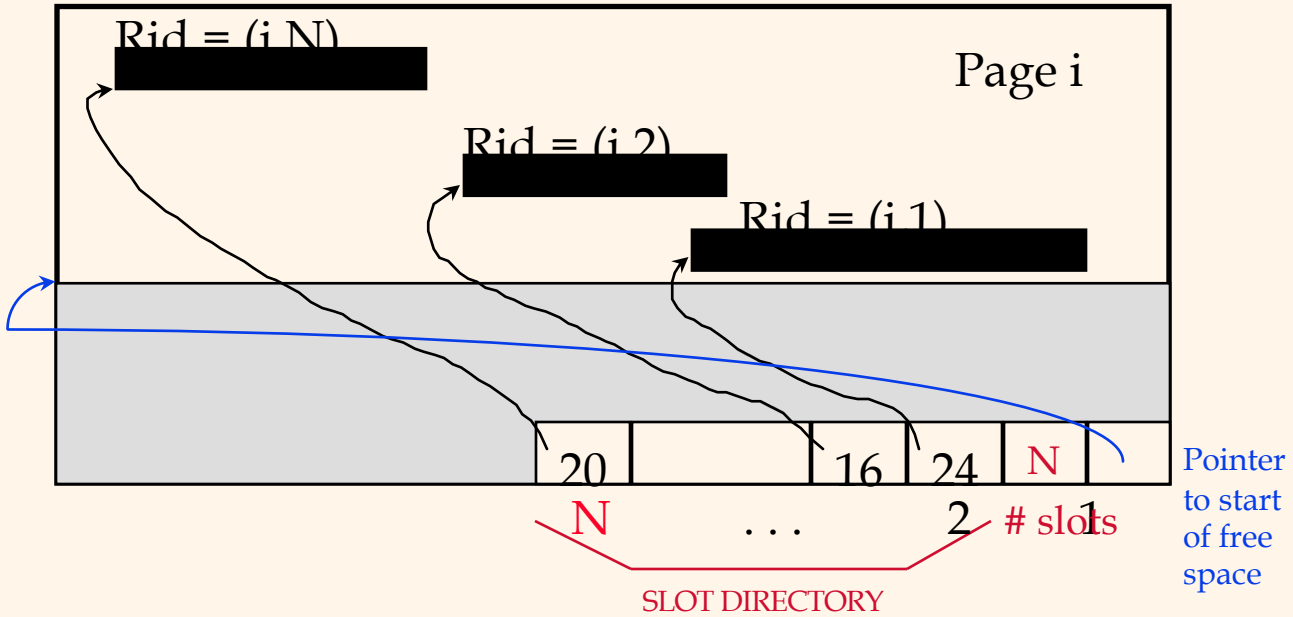
And this is the format the engine writes to disk.

Page Formats: Fixed Length Records



- *Record id* = *<page id, slot #>. In first alternative, moving records for free space management changes rid; may not be acceptable.*

Page Formats: Variable Length Records



- *Can move records on page without changing rid; so, attractive for fixed-length records too.*

Block and Record Addresses

Blocks and records have addresses

- Index tables map an index entry to {block, record}
- SELECT scan on a table needs a logical linked list of the blocks holding records, e.g. "next block."
- Index elements are themselves blocks, and need to reference other nodes in the index.

- But, …
  - The blocks can be all over the place, and
  - Each "place" has a different address format and access protocol.

Cloud 2

RACKSPACE® CLOUD BLOCK STORAGE

Cloud 1

AWS Storage Gateway Service

amazon web services EBS

Disk Room 1

Disk Room 2

Tape Library

Data Address Space (Overly Simple)

- The block/storage system represents block addresses with
  - A logical ID
  - That is simply a string, e.g. UUID or has some structure.
- There is a logical address to physical address mapping table that maps a logical address to a physical address
  - Block/storage engine ID.
  - An opaque address string that *the specific engine understands and manages*.
- For example, in a storage area network, the physical address might be
  - IP address of storage server.
  - ID of disk attached to the server.
  - {Cylinder, Track, Sector}

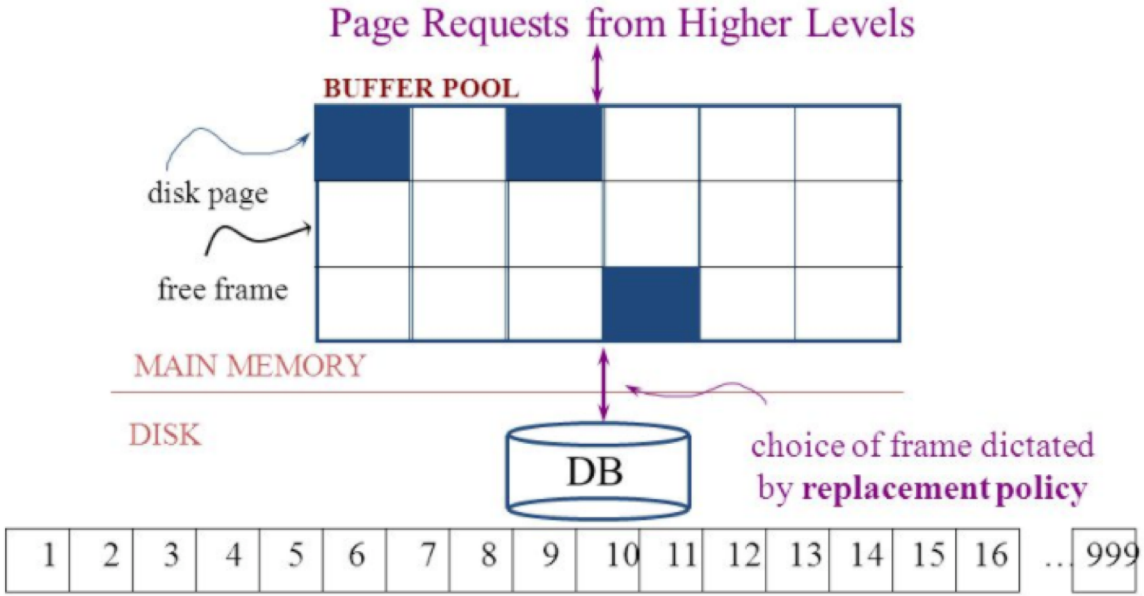
Any reference to a block, e.g. in an index node, uses the logical address.

Figuring out where, when and why to put a block in a specific engine/domain is an important optimization function in DMBS.

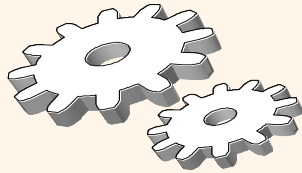
Buffer Pools

The Logical Concept

- The DBMS and queries can only manipulate in-memory blocks and records.
- A very, very, very small fraction of all blocks fit in memory.



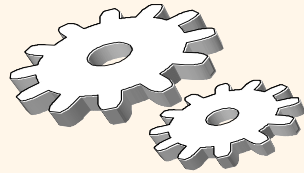
When a Page is Requested ...



- If requested page is not in pool:
  - Choose a frame for *replacement*
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- *Pin* the page and return its address.

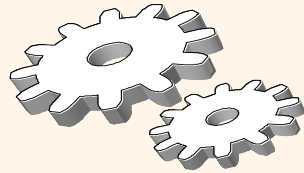
☛ *If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!*

More on Buffer Management



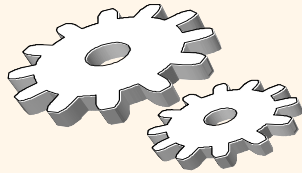
- Requestor of page must unpin it, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- Page in pool may be requested many times,
  - a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0.
- CC & recovery may entail additional I/O when a frame is chosen for replacement. (*Write-Ahead Log* protocol; more later.)

Buffer Replacement Policy



- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock, MRU etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*.
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

DBMS vs. OS File System



OS does disk space & buffer mgmt: why not let OS manage these tasks?

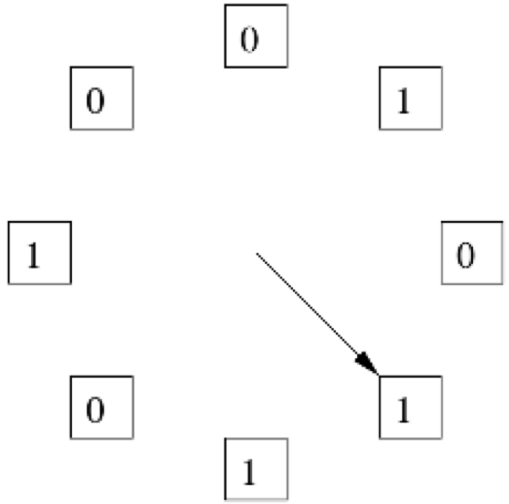
- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.

Replacement Policy

- The *replacement policy* is one of the most important factors in database management system implementation and configuration.
- A very simple, introductory explanation is (https://en.wikipedia.org/wiki/Cache_replacement_policies).
  - There are a lot of possible policies.
  - The *most* efficient caching algorithm would be to always discard the information that will not be needed for the longest time in the future. This optimal result is referred to as Bélády's optimal algorithm/simply optimal replacement policy or the clairvoyant algorithm.
- All implementable policies are an attempt to approximate knowledge of the future based on knowledge of the past.
- Least Recently Used is based on the simplest assumption
  - The information that will not be needed for the longest time.
  - Is the information that has not been accessed for the longest time.

The “Clock Algorithm”

- LRU is (perceived to be) expensive
  - Maintain timestamp for each block.
  - Update and resort blocks on access.
- The “Clock Algorithm” is a less expensive approximation.
  - Arrange the frames (places blocks can go) into a logical circle like the seconds on a clock face.
  - Each frame is marked 0 or 1.
    - Set to 1 when block added to frame.
    - Or when application accesses a block in frame.
  - Replacement choice
    - Sweep second hand clockwise one frame at a time.
    - If bit is 0, choose for replacement.
    - If bit is 1, set bit to zero and go to next frame.
- The basic idea is. On a clock face
  - If the second hand is currently at 27 seconds.
  - The 28 second tick mark is “the least recently touched mark.”



Replacement Algorithm

The algorithms are more sophisticated in the real world, e.g.

- "Scans" are common, e.g. go through a large query result in order (will be more clear when discussing cursors).
  - The engine knows the current position in the result set.
  - Uses the sort order to determine which records will be accessed soon.
  - Tags those blocks as not replaceable.
  - (A form of clairvoyance).
- Not all users/applications are equally "important."
  - Classify users/applications into priority 1, 2 and 3.
  - Sub-allocate the buffer pool into pools P1, P2 and P3.
  - Apply LRU within pools and adjust pool sizes based on relative importance.
  - This prevents
    - A high access rate, low-priority application from taking up a lot of frames
    - Result in low access, high priority applications not getting buffer hits.

Summary

- Like all other aspects of DMBS implementation
  - Block management
  - Buffer management

  Are incredibly complex, heavily researched topics.

- This course section has just skimmed the surface
  - Awareness of concepts.
  - Foundation for future more advanced courses.
  - Understand concepts to consider, research, prototype, etc. if you have to implement your own data management engine.