CS222/CS122C:
Principles of Data Management

Lecture #01
Fall 2019

Instructor: Prof. Chen Li
Department of Computer Science
UC Irvine

Overview

- Homepage
  - https://grape.ics.uci.edu/wiki/public/wiki/cs222-2019-fall
- Canvas page: https://canvas.eee.uci.edu/courses/20422
- Piazza page
  - https://piazza.com/uci/fall2019/cs222cs122c
  - Sign up ASAP

Related courses at ICS

- CS122A: Introduction to Data Management
- CS122B: Projects in databases and web applications
- CS122C/222: Principles in data management
- CS223: Transaction Processing & Distributed Systems
- CS224: Advanced topics
- CS221: Information Retrieval

Pre-requisites

- CS122A or equivalent
- Data structures, algorithms, OS
- C++ programming skills
- Willingness to write code to make it work!

Grading

- Four-Part Programming Project: 50%
- Final Exam: 50%
- "2-week window" to do a rebuttal

Textbooks

- Required: Database Management Systems, 3rd edition, by R. Ramakrishnan and J. Gehrke, McGraw Hill, 2003.
- Recommended: Readings in Database Systems, 4th edition, by J. Hellerstein and M. Stonebraker, MIT Press, 2005.

Team Projects

- 1: file and record management
- 2: relation manager
- 3: index manager
- 4: query engine

- C++
- 48-hour grace period with a 10% penalty

Managing submissions on github

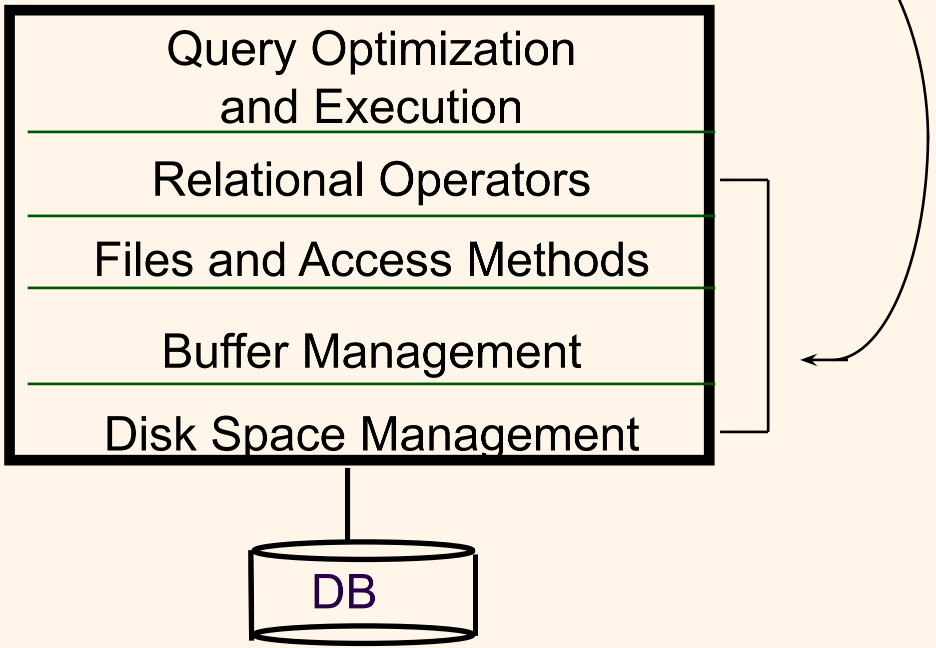
- All code and submissions will be handled on github.
- We will use the copy and timestamp of the "master" as your submission and whether you are using the grace period.
- For a team of 2 students, each member is expected to do the best effort to make equal contributions. We reserve the rights to deduct points for **both** students if we see unbalanced contributions.

Next: Overview of DBMS

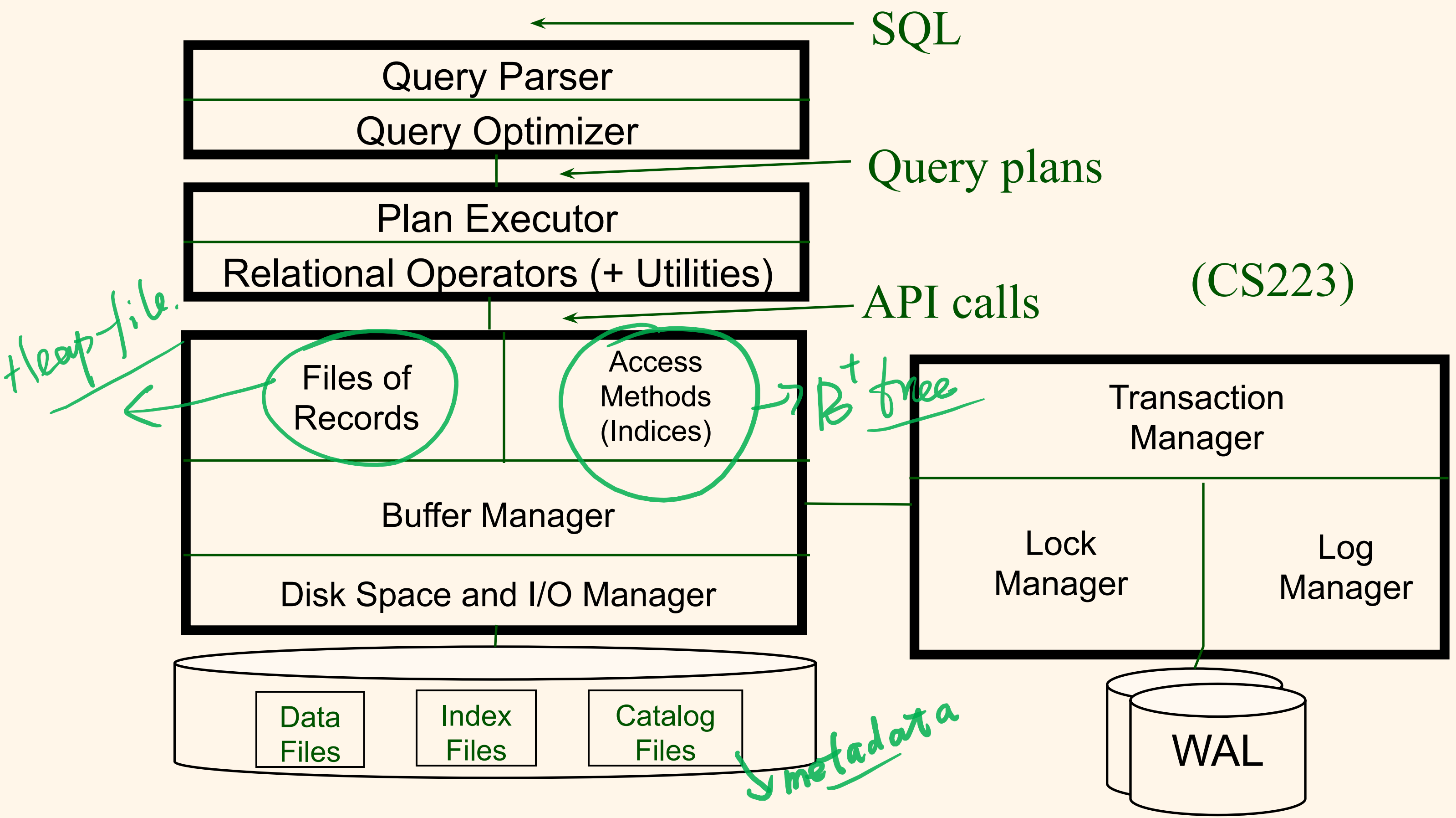
Structure of a DBMS

These layers must consider concurrency control and recovery

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components (CS 223).
- This is one of several possible architectures; each system has its own variations.



DBMS Structure In More Detail



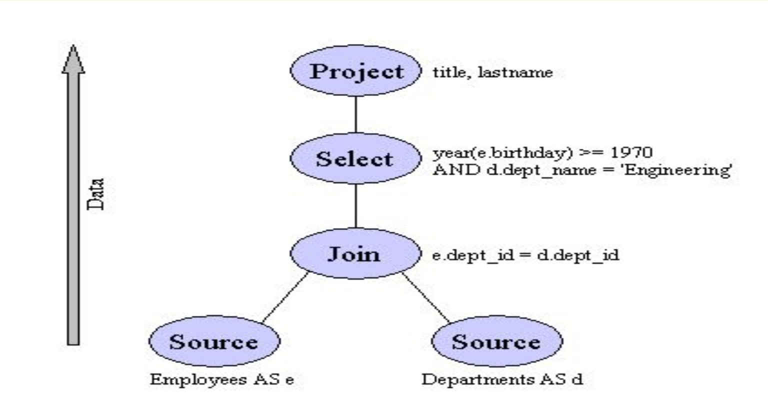
Components' Roles

- Query Parser
  - Parse and analyze SQL query
  - Produce data structure capturing SQL statement and the "objects" that it refers to in the system catalogs
- Query optimizer (often w/2 steps)
  - Rewrite query logically
  - Perform cost-based optimization
  - Goal is a "good" query plan considering
    - Physical table structures
    - Available access paths (indexes)
    - Data statistics (if known)
    - Cost model (for relational operations)

(Cost differences can be orders of magnitude!)

Components' Roles (continued)

- Plan Executor + Relational Operators
  - Runtime side of query processing
  - Usually based on "tree of iterators" model, e.g.:



  - Nodes are relational operators (actually they are physical implementations of the various operators)

Components' Roles (continued)

- Files of Records
  - OSs usually have byte-stream based APIs
  - DBMSs instead provide record-based APIs
    - Record = set of fields
    - Fields are typed
  - Records reside on pages of files
- Access Methods
  - Index structures for access based on field values
  - We'll look at tree-based, hash-based, and spatial structures (including the time-tested B+ tree)
  - Peer layer to record-based files (to map from field values to lists of RIDs or lists of primary keys)

Components' Roles (continued)

- Buffer Manager
  - DBMS answer to main memory management
  - Cache of pages from files and indices
  - "DB-oriented" page replacement scheme(s)
  - All disk page accesses go via the buffer pool
  - Also interacts with logging/recovery management (to support undo/redo and thus data consistency)
- Disk Space and I/O Managers
  - Manage space on disk (pages), including extents
  - Also manage I/O (sync, async, prefetch, …)

Components' Roles (continued)

- System Catalog
  - Info about physical data (volumes, table spaces, …)
  - Info about tables (name, columns, types, … ; also constraints, keys, etc., etc.)
  - Data statistics (e.g., value distributions, counts, …)
  - Info about indexes (types, target tables, …)
  - And so on!
    - Views, triggers, security, …
- Transaction Management (CS 223)
  - ACID: Atomicity, Consistency, Isolation, Durability
  - Lock Manager for C+I
  - Log Manager for A+D

A Brief History of Databases

- Pre-relational era: 1960's, early 1970's
- Codd's seminal paper: 1970
- Basic RDBMS R&D: 1970-80 (System R, Ingres)
- RDBMS improvements: 1980-85
- Relational goes mainstream: 1985-90
- Distributed DBMS research: 1980-90
- Parallel DBMS research: 1985-95
- Extensible DBMS research: 1985-95
- OLAP and warehouse research: 1990-2000
- Stream DB and XML DB research: 2000-2010
- Big data R&D: 2005-present

So What's the Plan?

- We'll start working our way up the architectural stack next time
- You should also start on the 4-part course project right away
- Immediate to-do's for you are:
  - Read the materials indicated on the wiki
  - Get yourself signed up on Piazza
  - Review SQL and chapters 1-8 if need be
  - Start on part 1 of the project (solo) today!

Next

- Disks and files
- Project 1 Overview
  - Paged File Manager
  - Record-Based File Manager

Disks and Files

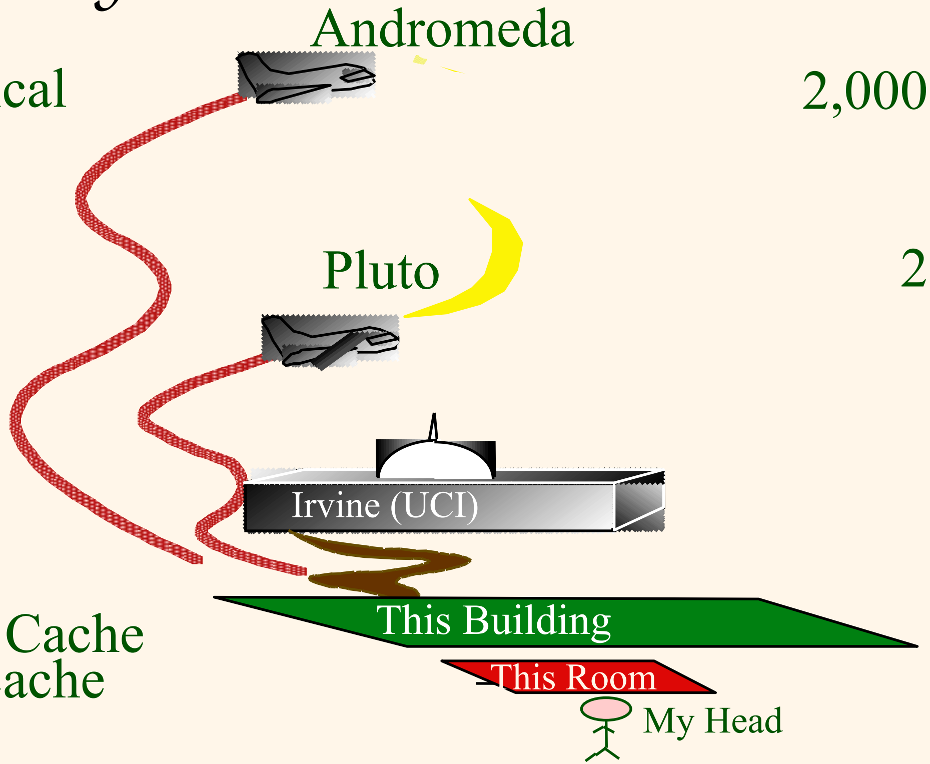
- DBMS stores information on ("hard") disks.
- This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Why Not Store Everything in Main Memory?

- *Costs too much.* Dell wants (in early 2014) \$65 for 500GB of disk, \$600 for 256GB of SSD, and \$57 for 4GB of RAM (\$0.13, \$2.34, \$14.25 per GB)
- *Main memory is volatile.* We want data to be saved between runs. (Obviously!)
- Your typical (basic) storage hierarchy:
  - Main memory (RAM) for currently used data
  - Disk for the main database (secondary storage)
  - Tapes for archiving older versions of the data (tertiary storage)
- And we also have L1 & L2 caches, SSD, …

Storage Hierarchy & Latency (Jim Gray): How Far Away is the Data?

10^9	Tape /Optical Robot	2,000 years
10^6	Disk	2 years
100	Memory	1.5 hr
10	On Board Cache	10 min
2	On Chip Cache	
1	Registers	1 min

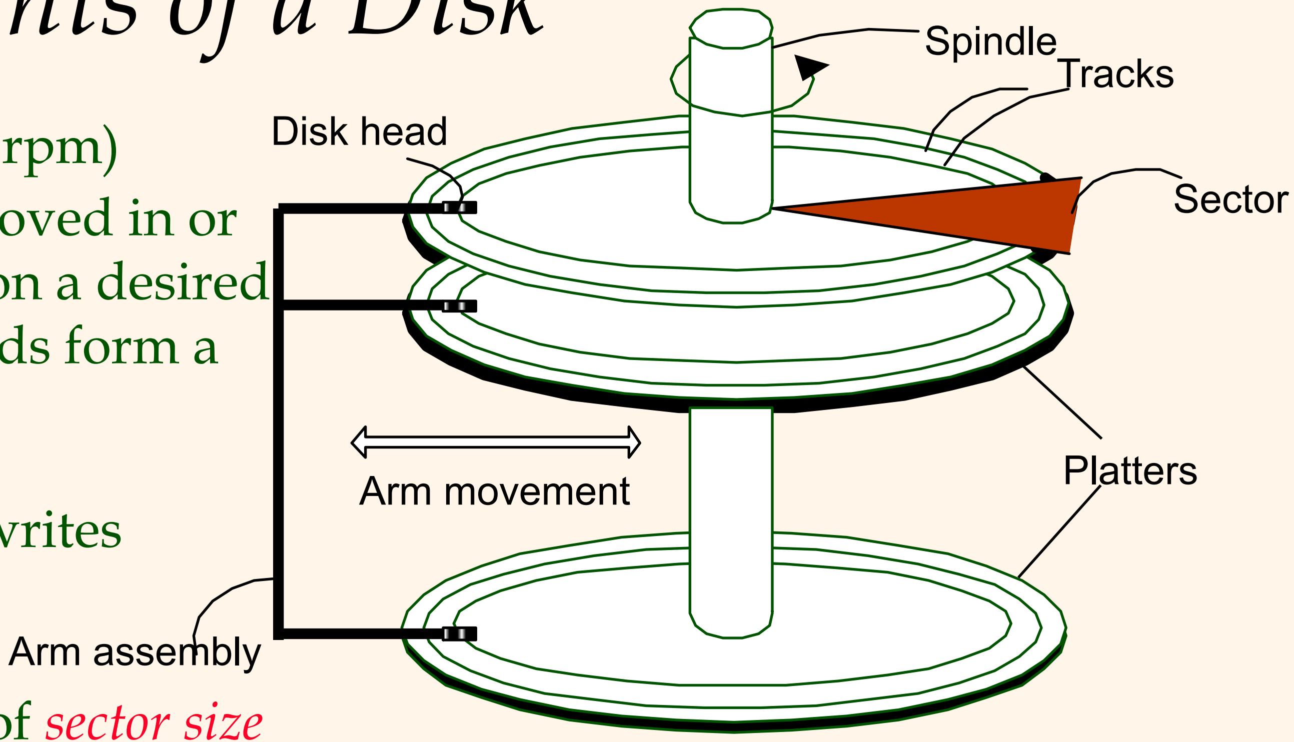


Disks

- Secondary storage device of choice.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has a *major* impact on DBMS performance!
  - (SSDs simplify things a bit in this respect)

Components of a Disk

- The platters spin (5400 rpm)
- The arm assembly is moved in or out to position a head on a desired track Tracks under heads form a *cylinder* (imaginary!)
- Only one head reads/writes at any one time.
- *Block size* is a multiple of *sector size* (which is fixed)



Accessing a Disk Page

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20 msec
  - Rotational delay varies from 0 to 10 msec
  - Transfer rate is about 1 msec per 4KB page *(old)*
- Key to lower I/O cost: Reduce seek/rotation delays! Hardware vs. software solutions?

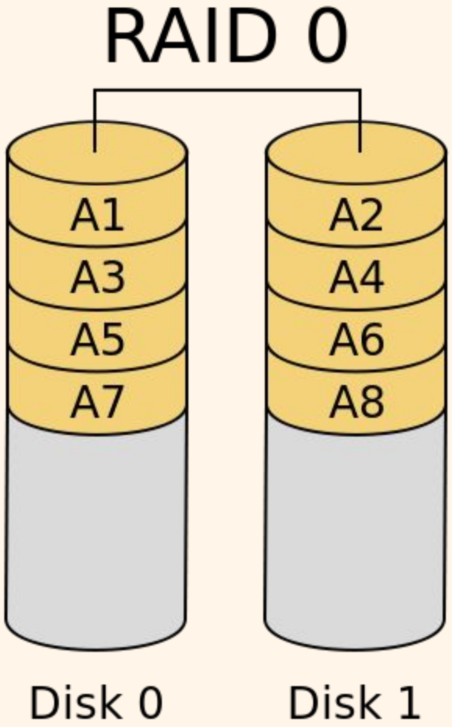
Arranging Pages on Disk

- \`*Next*\` block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Blocks in a file should be arranged sequentially on disk (by \`next') in order to minimize seek and rotational delay
- For a sequential scan, <u>*prefetching*</u> several pages at a time is a big win!

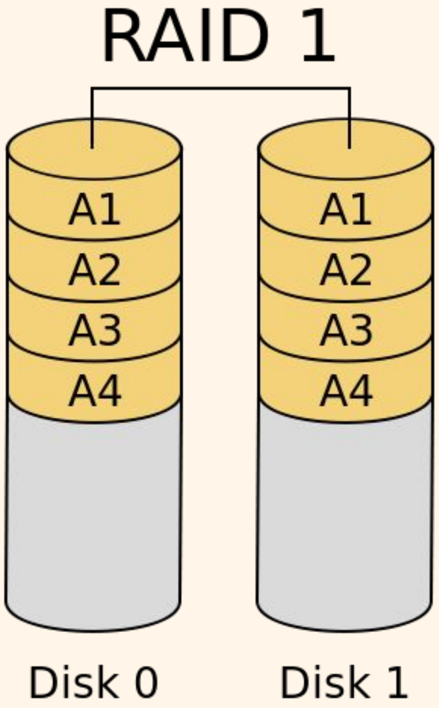
RAID (Redundant Array of Inexpensive Disks)

- Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
- Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks => more failures. Redundant information allows reconstruction of data if a disk fails.

RAID 0: No redundancy (just striping)

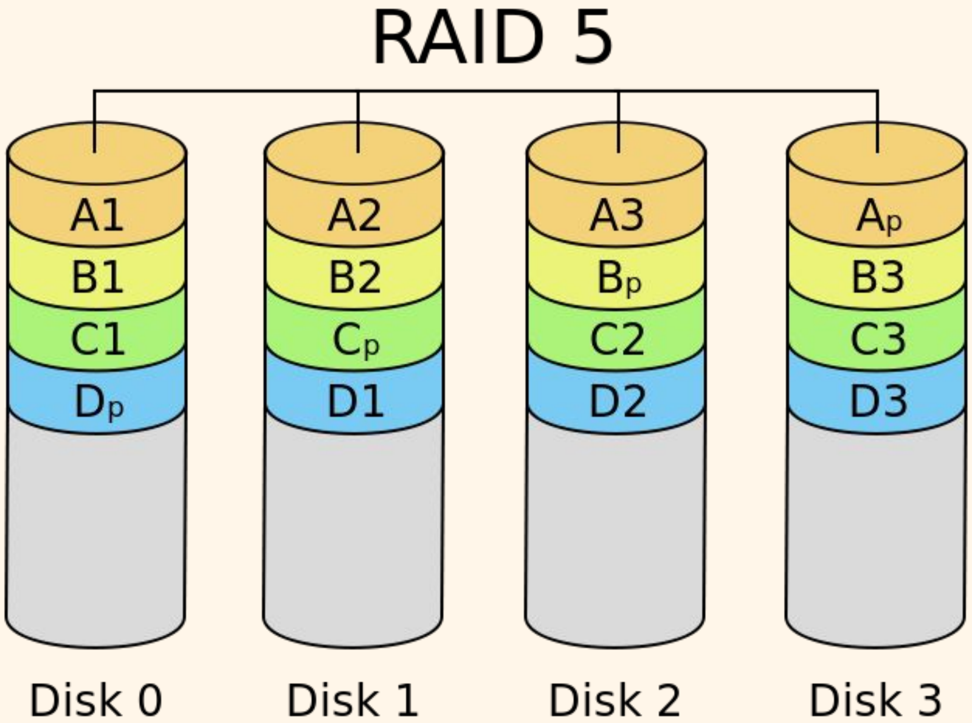


RAID 1: Mirrored (two identical copies)



when do the revise operation,
update both disks.

safer than RAID 0



P → parity

$$A_p = A_1 \oplus A_2 \oplus A_3$$

$$\Downarrow$$

$$A_1 = A_2 \oplus A_3 \oplus A_p$$

Recover A_1

when write a new data, also need to update the Ap...

Ap,Bp,Cp,Dp在不同的disk：读取的时候，可以在四个disk上load data，而不是在三个disk上读取 -> load balance.

recover one table using the other tables.

Disk Space Management

- Lowest layer of DBMS software manages the space on disk.
- Higher levels call upon this layer to:
  - allocate/deallocate a page
  - read/write a page
- A request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don't need to know how this is done or how free space is managed.

Project 1 Overview