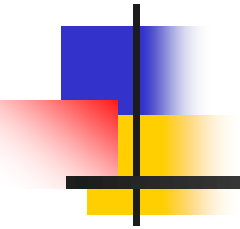
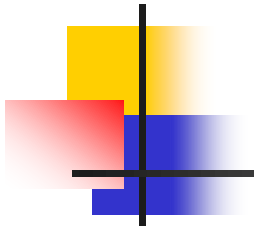
Advanced Database Applications:

CS562 -- Spring 2020



George Kollios
Boston University



Prof. George Kollios
Office: MCS 104
Office Hours: Monday 11:00am-12:30pm
Tuesday 1:00pm-2:30pm
TF: Dimitrios Staratzis
Office: MCS 105C
Office Hours: Wednesday 12:00-1:00pm
Friday 1:30-2:30pm

Web: `http://www.cs.bu.edu/faculty/gkollios/cs562s20`

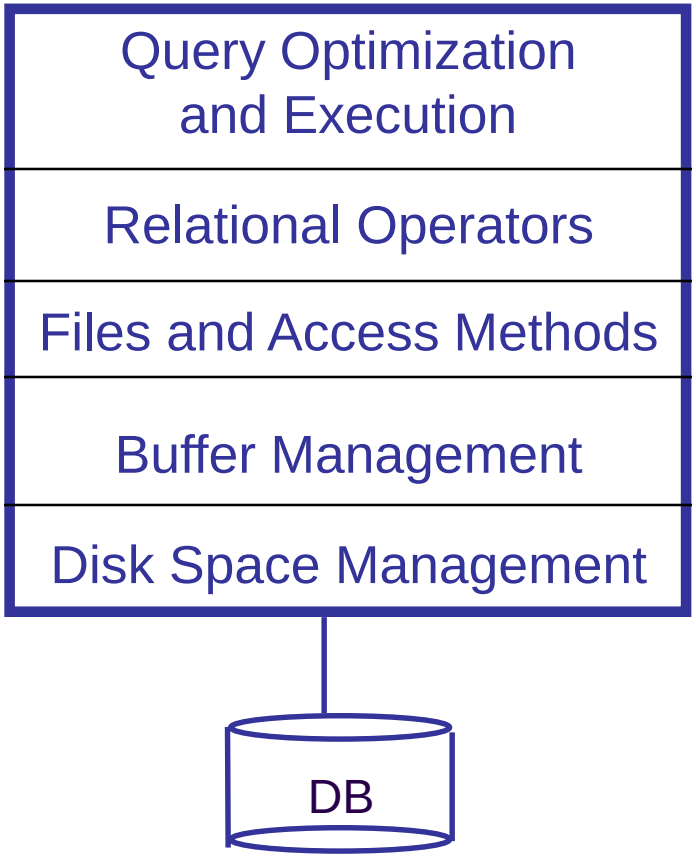
Piazza site for CS 562

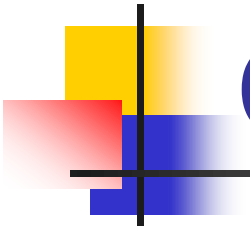
History of Database Technology

- 1960s: Data collection, database creation, IMS and network DBMS
- 1970s: Relational data model, relational DBMS implementation
  - E.F. Codd http://en.wikipedia.org/wiki/Edgar_F._Codd
- 1980s: RDBMS, advanced data models (extended-relational, OO, deductive, etc.) and application-oriented DBMS (spatial, scientific, engineering, etc.)
- 1990s—2000s: Data mining and data warehousing, multimedia databases.
- 2010s-: Data on the cloud, privacy, security. Social network data (facebook, twitter, etc), Web 3.0 and more
- Turing Awards: E.F. Codd 1981, Jim Gray 1998, Mike Stonebraker 2014

Structure of a RDBMS

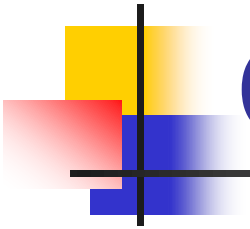
- A DBMS is an OS for data!
- A typical RDBMS has a layered architecture.
- Data are stored on hard disks (more soon)





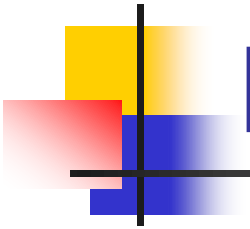
Overview of the course

- Geo-Spatial Database Systems
  - Maps, Intelligent Transportation, etc
  - GIS, CAD/CAM, EOSDIS project NASA
  - Manages points, lines and regions
- Temporal Database Systems
  - Billing, medical records
- Spatio-temporal Databases
  - Moving objects, changing regions, etc



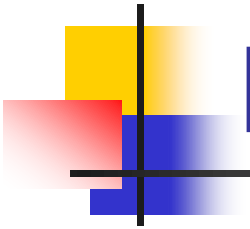
Overview of the course

- Multimedia databases
  - A multimedia system can store and retrieve objects/documents with text, voice, images, video clips, etc
- Time series databases
  - Stock market, ECG, trajectories, etc



Multimedia databases

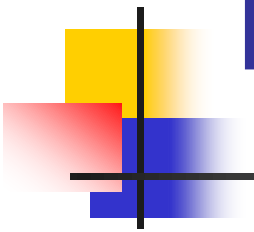
- Applications:
  - Digital libraries, entertainment, office automation
  - Medical imaging: digitized X-rays and MRI images (2 and 3-dimensional)
- Query by content: (or QBE)
  - Efficient
  - ‘Complete’ (no false dismissals)



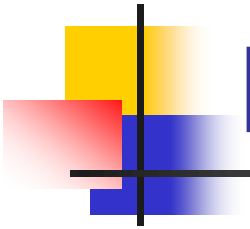
Databases on the Cloud

- Cloud computing is a new trend
- Data are stored “in the cloud”, accessed from everywhere
- System should maximize utility, availability, minimize response time
- Use of large clusters (data centers)
  - MapReduce

Map-Reduce (Hadoop)



- Map-Reduce is used to execute simple programs on top of very large amounts of data (Big Data) using hundreds to thousands of computers
- Cluster computing and Data Centers (HDFS)
- Queries on top of MR (Hive, Pig)
- Key-value stores, NoSQL systems

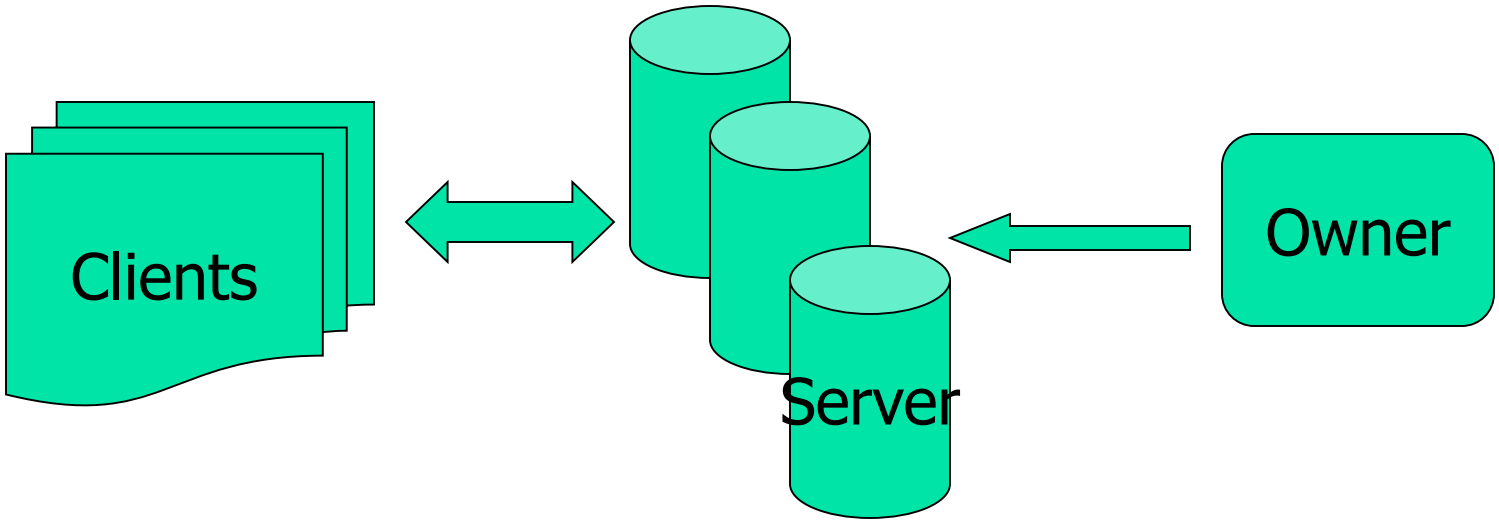


Database Security and Privacy

- Data stored on the Cloud can increase security and privacy risks
- Data published on-line also increase privacy risks
- Basic ideas for increasing confidentiality and integrity in databases on the Cloud
- Differential Privacy for data publishing

Cloud and Database Outsourcing

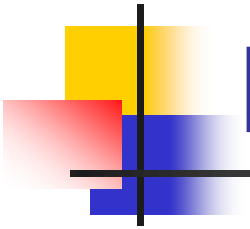
Owner(s): publish database
Servers: host database and provide query services
Clients: query the owner's database through servers



Query execution issues.
Security Issues: untrusted or compromised servers

Probabilistic databases and Entity Resolution (tentative)

- How you can answer queries when data records are not precise?
  - "a record belongs to a database" is a probabilistic event
  - a record is the answer to a query is a probabilistic event
- How can you find if multiple records refer to the same subject?



Back to reality…

- Grading:
  - Assignments: 0.45
    - 4 Homeworks
    - 3 Programming Projects
  - Midterm: *probably* on March 23, in class: 0.25
  - Final: May 7 at 3:00pm : 0.3

Data on Disk and B+-trees

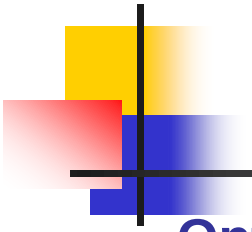
Storage Media: Players

- **Cache** – fastest and most costly form of storage; volatile; managed by the computer system hardware.
- **Main memory**:
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
  - **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs.
  - But… CPU operates only on data in main memory

Storage Media: Players

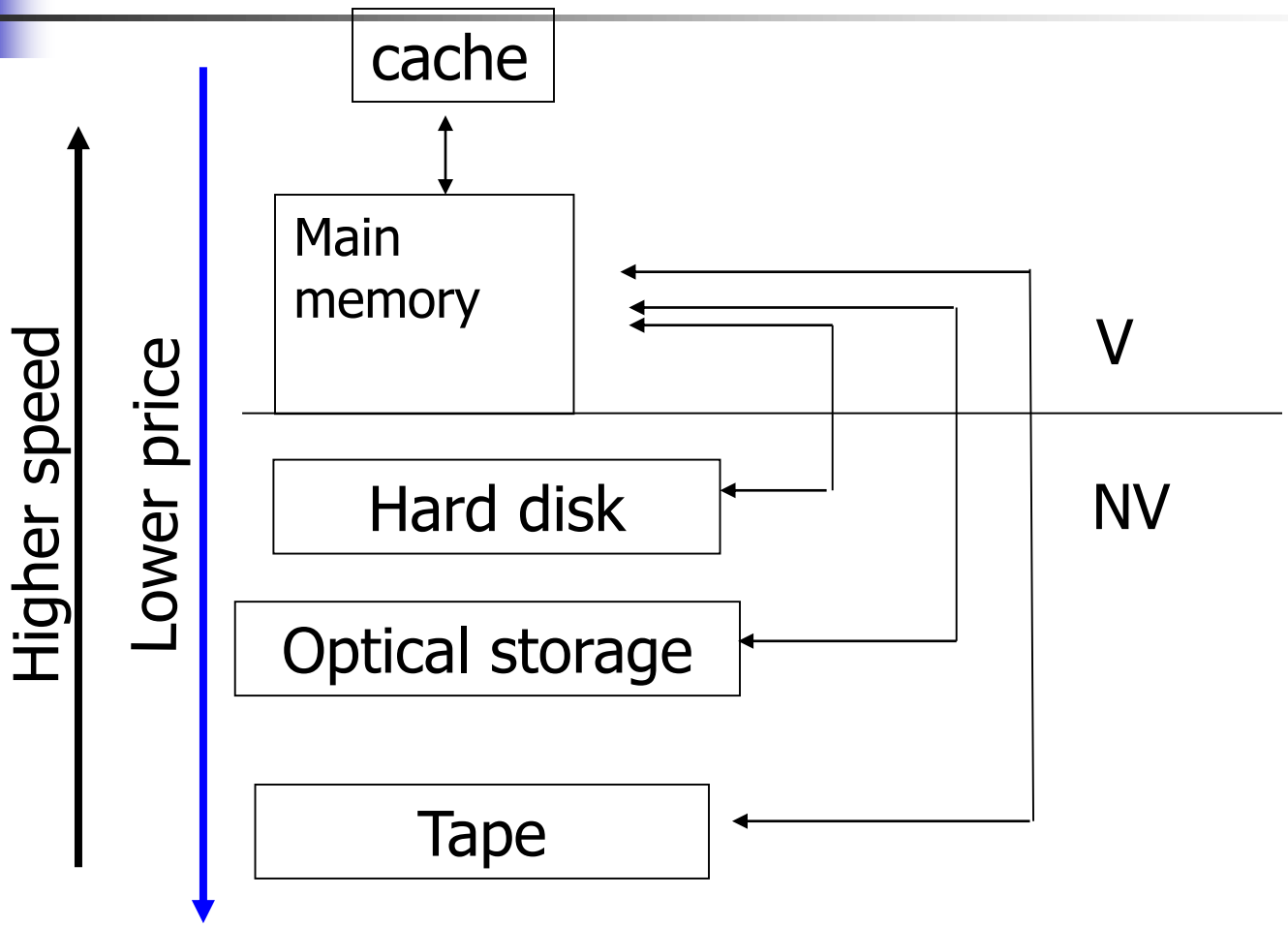
- **Disk**
  - Primary medium for the long-term storage of data; typically stores entire database.
  - random-access – possible to read data on disk in any order, unlike magnetic tape
  - Non-volatile: data survive a power failure or a system crash, disk failure less likely than them
- New technology: Solid State Disks and Flash disks

Storage Media: Players



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

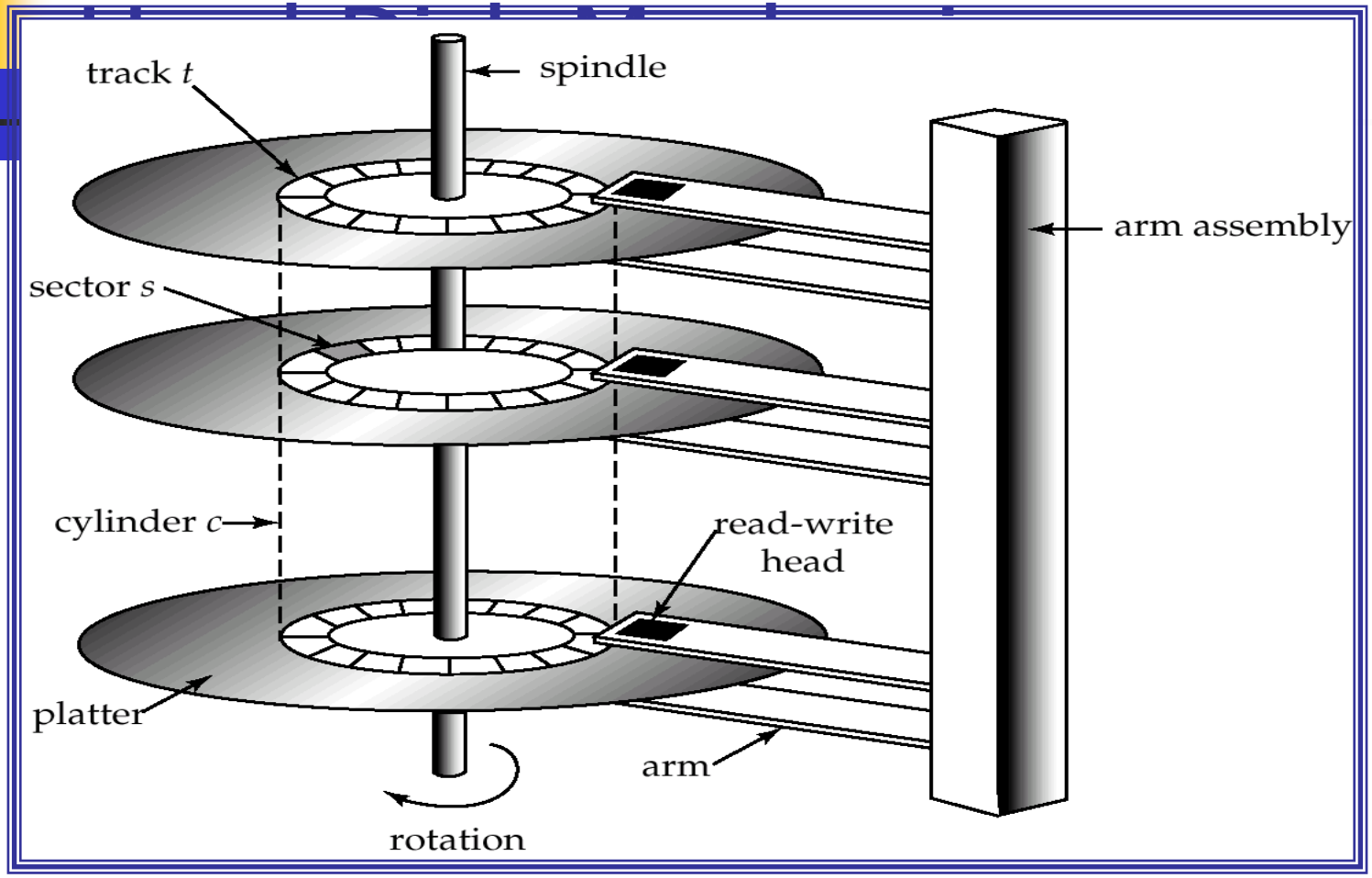
Memory Hierarchy

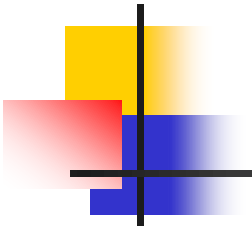


Memory Hierarchy

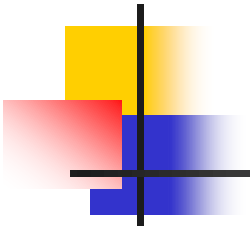
- Data transfers
  - cache – mm : OS/hardware controlled
  - mm – disk : <- reads, -> writes controlled by DBMS
  - disk – CD-Rom or DVD
  - disk – Tapes

Backups (off-line)


track t
spindle
arm assembly
sector s
cylinder c
read-write
head
platter
arm
rotation



- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
- Surface of platter divided into circular **tracks**
- Each track is divided into **sectors.**
  - A sector is the smallest unit of data that can be read or written.
- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written as sector passes under head
- **Block: a sequence of sectors**
- **Cylinder** *i* consists of *i*th track of all the platters



"Typical" Values

Diameter:	1 inch → 10 inches
Cylinders:	100 → 2000
Surfaces:	1 or 2
(Tracks/cyl)	2 → 30
Sector Size:	512B → 50K
Capacity:	4 MB → 8 TB

Performance Measures of Disks

Measuring Disk Speed

- **Access time** – consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
  - **(Rotational) latency time** – time it takes for the sector to be accessed to appear under the head.
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.

Analogy to taking a bus:

1. Seek time: time to get to bus stop
2. Latency time; time spent waiting at bus stop
3. Data transfer time: time spent riding the bus

Example

Seagate : Barracuda ST3000DM001

Capacity:3 TB

Interface: SATA 6Gb/s

RPM: 7200 RPM

Seek time: <8.5 ms avg

Latency time?:

7200/60 = 120 rotations/sec

1 rotation in 8.3 ms => So, Av. Latency = 4.16 ms

Random vs sequential I/O

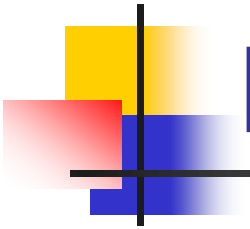
- Ex: 1 KB Block
  - Random I/O: ~ 15 ms.
  - Sequential I/O: ~ 1 ms.

Rule of Thumb

Random I/O: Expensive
Sequential I/O: Much less ~10-20 times

Performance Measures (Cont.)

- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
  - Typically 5 to 10 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 30,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

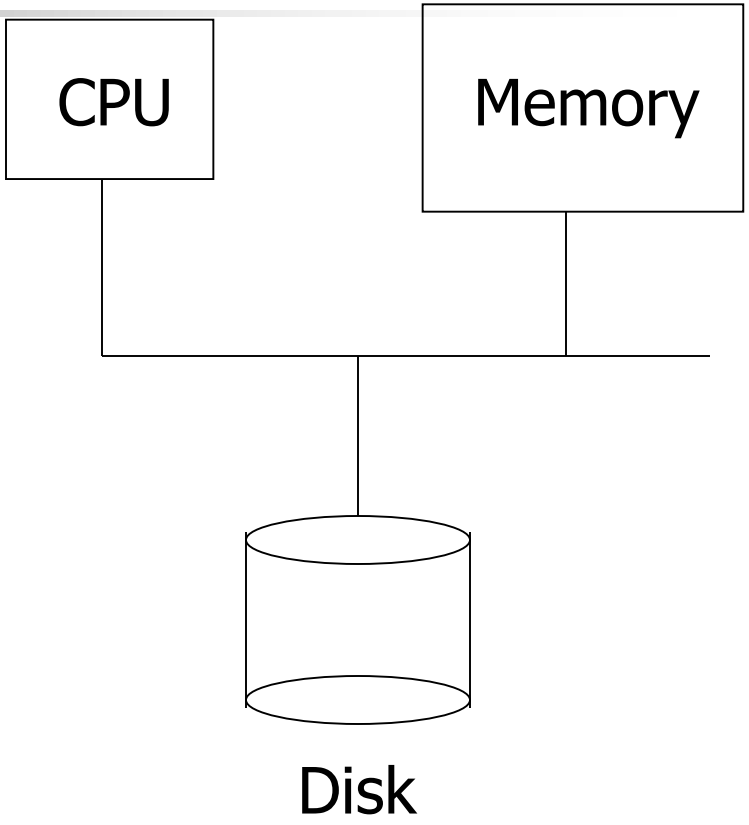


Data Access

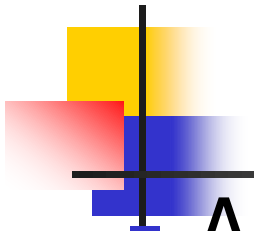
- Data Access Method (aka Index)
  - data structure that allows efficient access of data stored on hard disk for specific queries (query types)
- Query types:
  - Exact match
  - Range Query
  - …. more
- With New Data types and Applications =>
  - New Indexing and Query algorithms!!
  - New Systems!!!

Model of Computation

- Data stored on disk(s)
- Minimum transfer unit: a page = b bytes or B records (or block)
- N records -> N/B = n pages
- I/O complexity: in number of pages



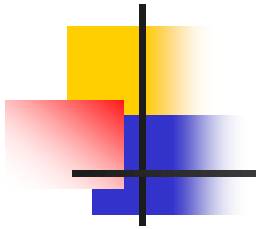
I/O complexity



- An ideal index has space O(N/B), update overhead O(1) or O($\log_B$(N/B)) and search complexity O(α/B) or O($\log_B$(N/B) + α/B)

  where α is the number of records in the answer
- But, sometimes CPU performance is also important… minimize cache misses -> don't waste CPU cycles

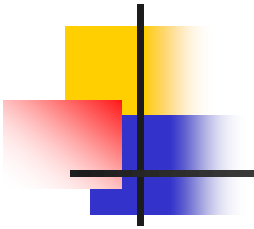
B+-tree



- Records must be ordered over an attribute, SSN, Name, etc.
- Queries: exact match and range queries over the indexed attribute: “find the name of the student with ID=087-34-7892” or “find all students with gpa between 3.00 and 3.5”

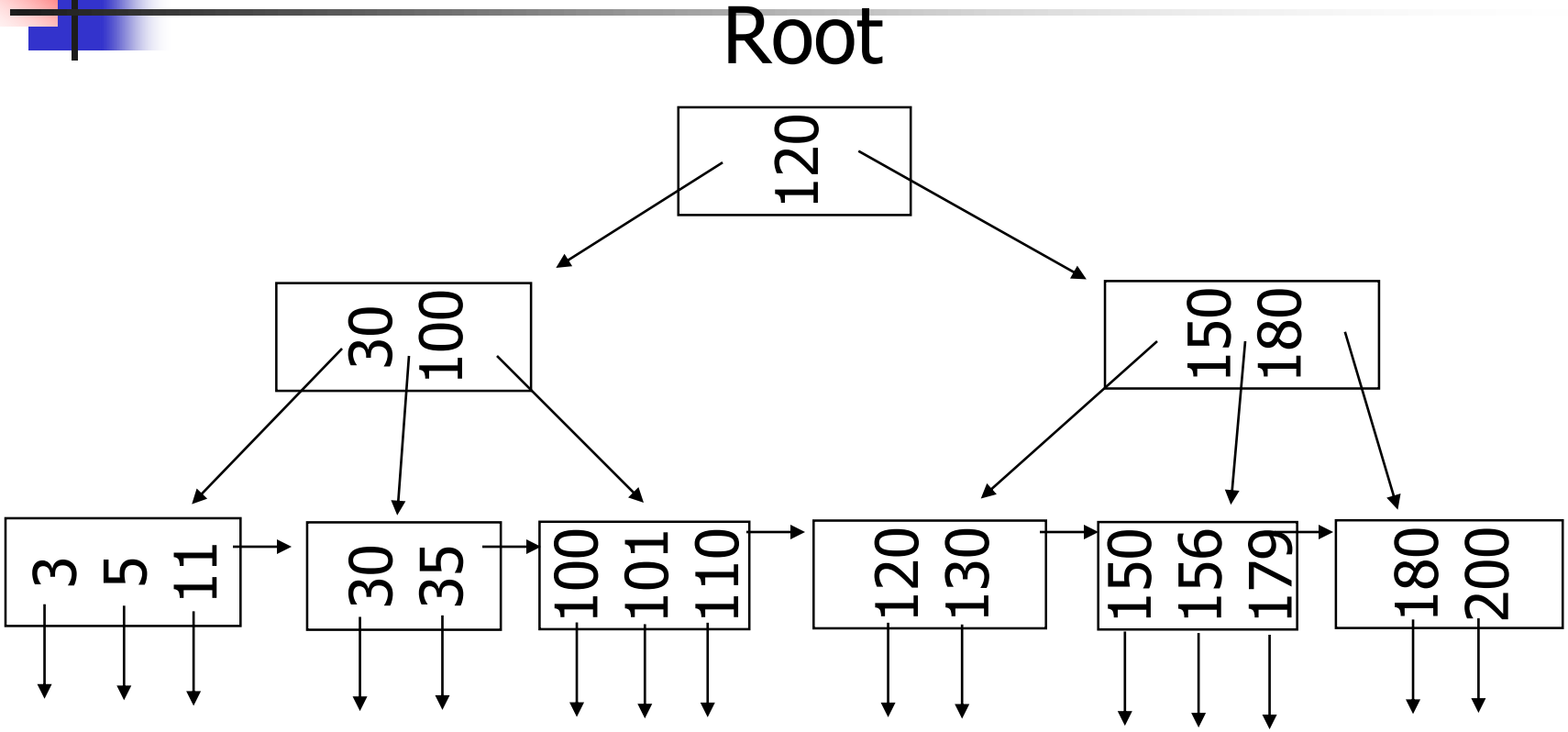
- Optimal 1-dimensional Index for range queries!

B+-tree:properties

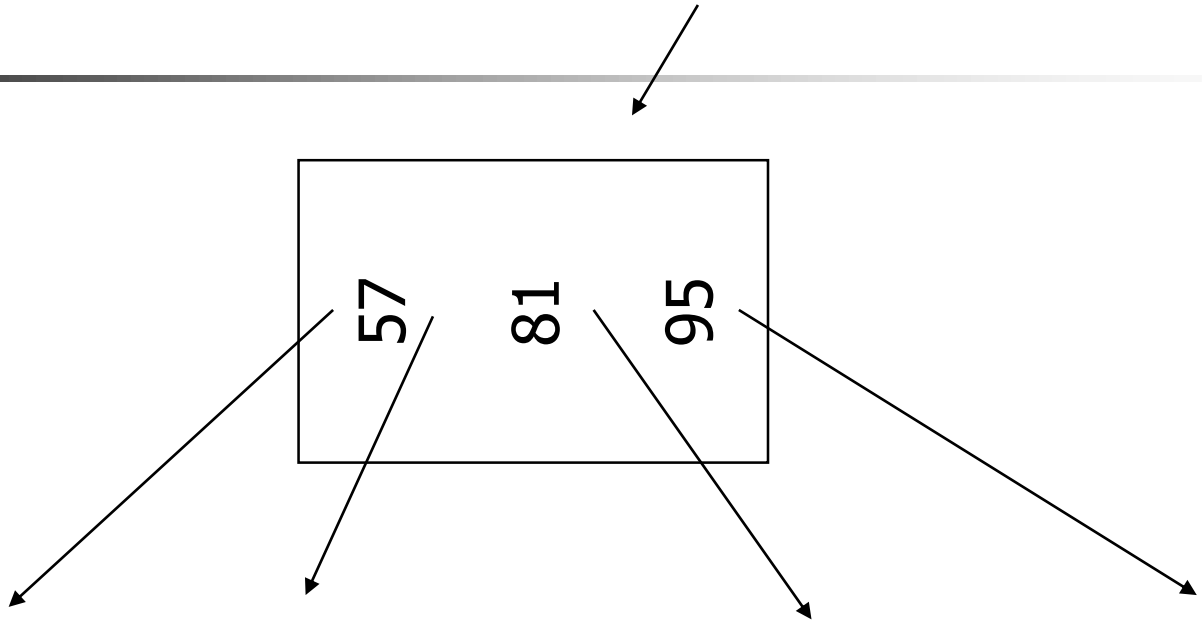


- Insert/delete at log $_F$ (N/B) cost; keep tree *height-balanced*. (F = fanout)
- Minimum 50% occupancy (except for root). Each node contains **d** <= *m* <= 2**d** entries/pointers.
- Two types of nodes: **index** nodes and **data** nodes; each node is 1 page (disk based method)

Example

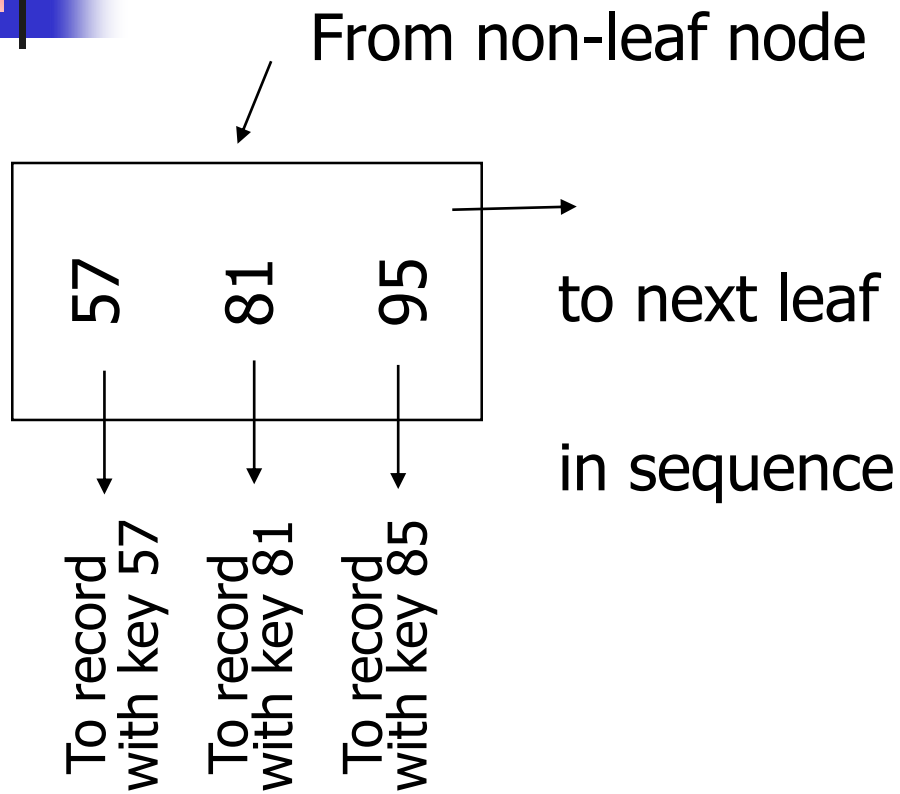

Root
120
30
100
150
180
3
5
11
30
35
100
101
110
120
130
150
156
179
180
200

Index node



to keys	to keys	to keys	to keys
< 57	57≤ k<81	81≤k<95	95≤

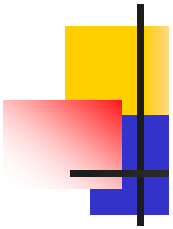
Data node




```
Struct {
Key   real;
Pointr *long;
} entry;

Node entry[B];
```

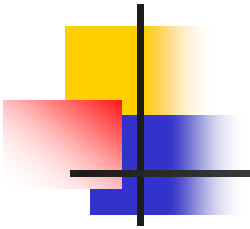
Insertion



- Find correct leaf *L.*
- Put data entry onto *L*.
  - If *L* has enough space, *done*!
  - Else, must *split* *L (into L and a new node L2)*
    - Redistribute entries evenly, **copy up** middle key.
    - Insert index entry pointing to *L2* into parent of *L*.
- This can happen recursively
  - To split index node, redistribute entries evenly, but **push up** middle key. (Contrast with leaf splits.)
- Splits "grow" tree; root split increases height.
  - Tree growth: gets *wider* or *one level taller at top*.

Deletion

- Start at root, find leaf *L* where entry belongs.
- Remove the entry.
  - If L is at least half-full, *done!*
  - If L has only **d-1** entries,
    - Try to **re-distribute**, borrowing from **sibling** *(adjacent node with same parent as L)*.
    - If re-distribution fails, **merge** *L* and sibling.
- If merge occurred, must delete entry (pointing to *L* or sibling) from parent of *L*.
- Merge could propagate to root, decreasing height.



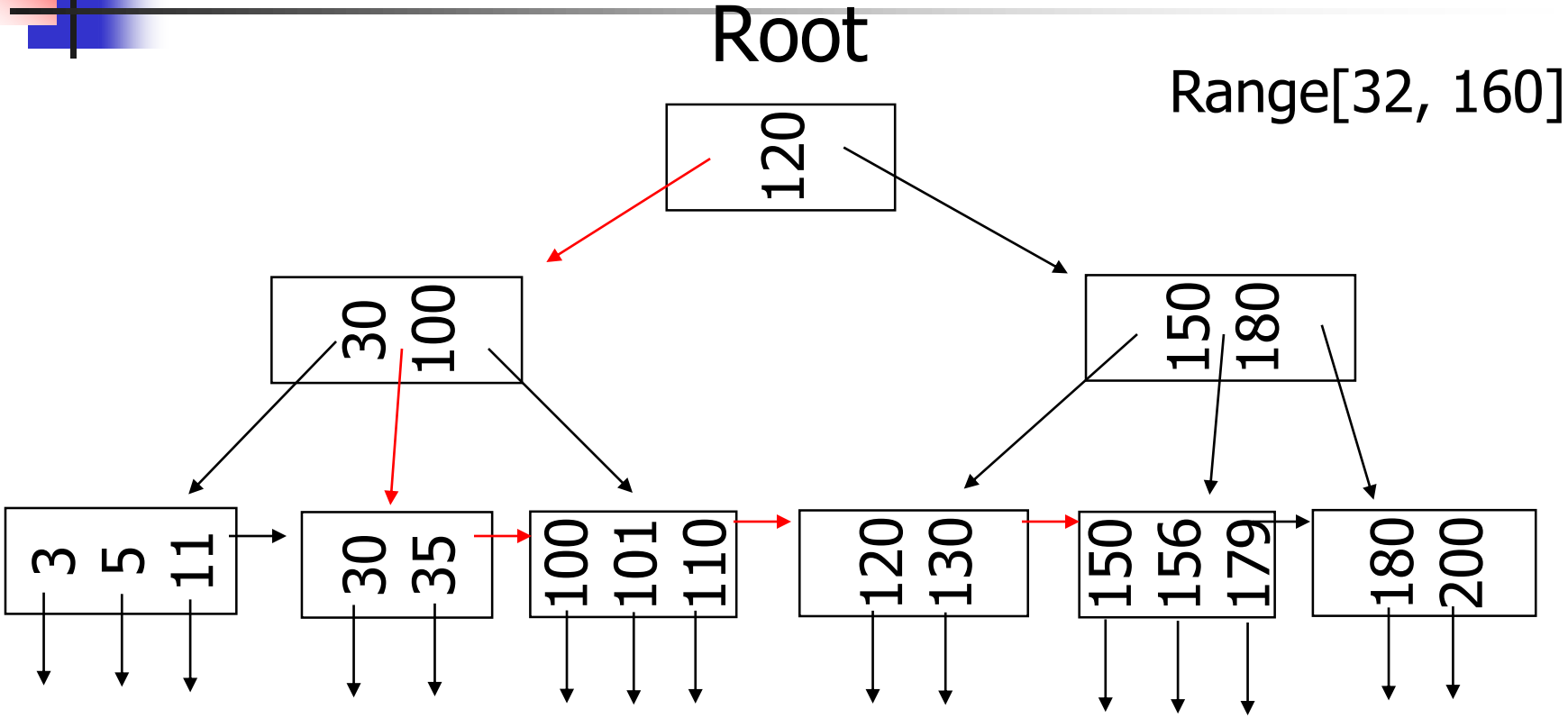
Characteristics

- Optimal method for 1-d range queries:

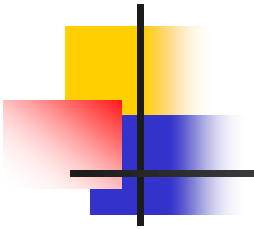
Space: $O(N/B)$, Updates: $O(\log_B(N/B))$, Query:$O(\log_B(N/B) + \alpha/B)$

- Space utilization: 67% for random input
- B-tree variation: index nodes store also pointers to records

Example


Root
Range[32, 160]
120
30
100
150
180
3
5
11
30
35
100
101
110
120
130
150
156
179
180
200

Other issues



- Internal node architecture [Lomet01]:
  - Reduce the overhead of tree traversal.
    - Prefix compression: In index nodes store only the prefix that differentiate consecutive sub-trees. Fanout is increased.
- Cache sensitive B+-tree
  - Place keys in a way that reduces the cache faults during the binary search in each node.
  - Eliminate pointers so a cache line contains more keys for comparison.