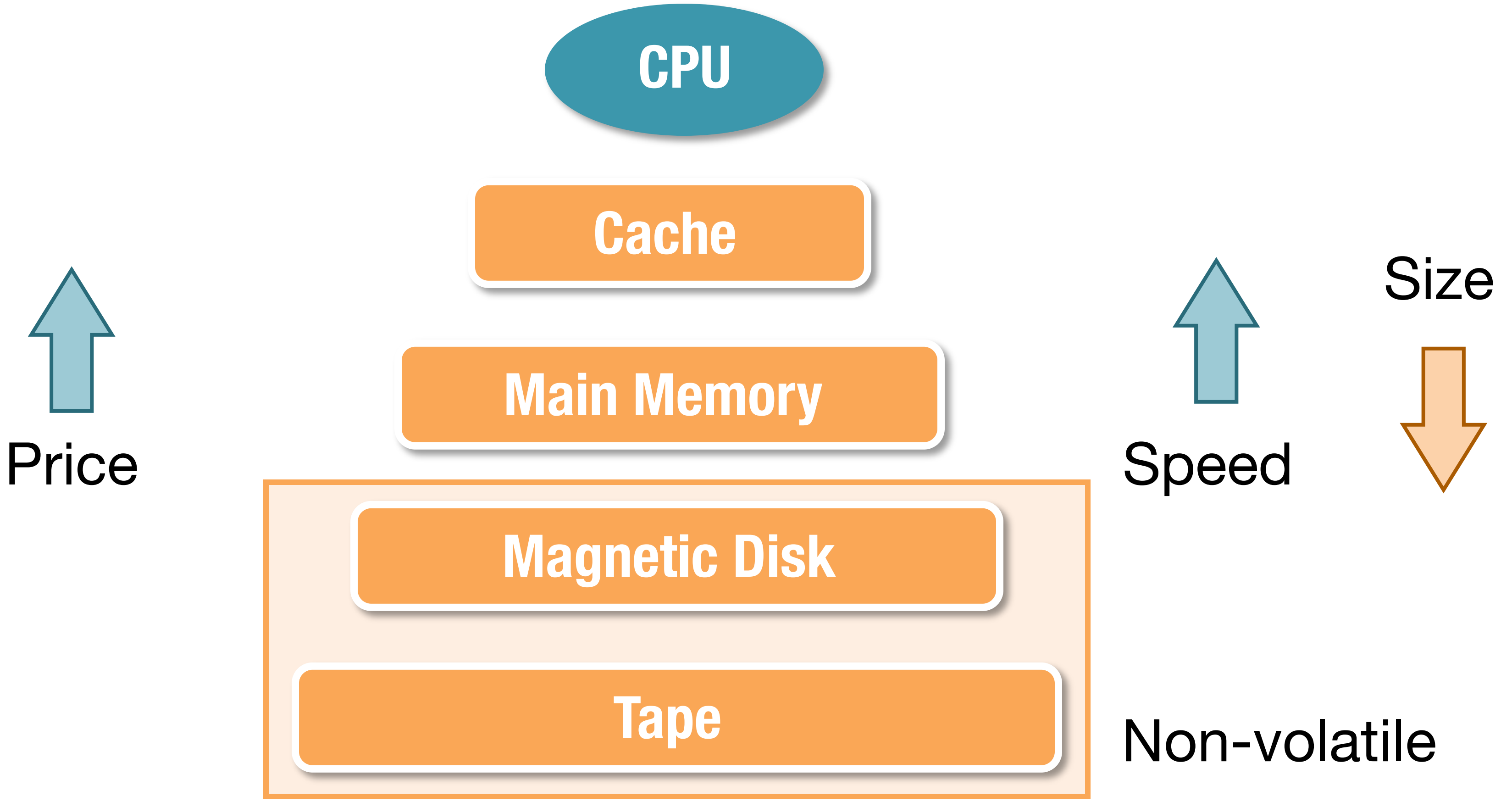
Storage and Indexing

Chapter 8

The Memory Hierarchy

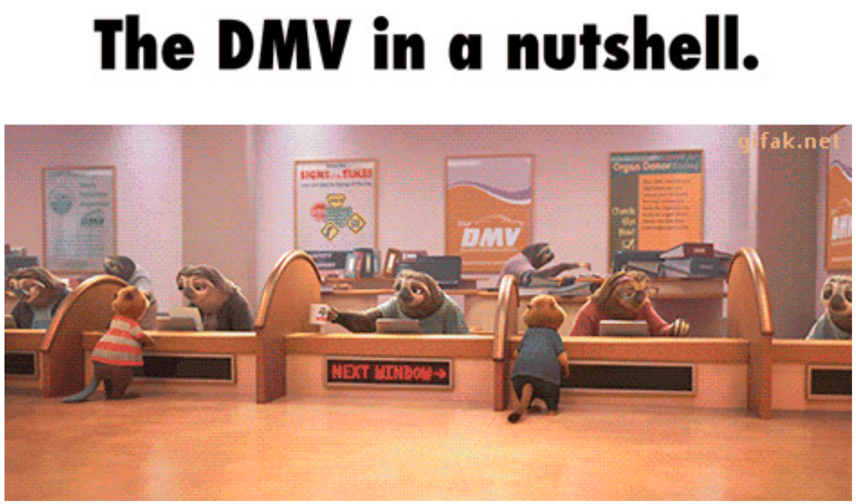


Performance of Microprocessors and Memory improving faster than disks and tapes

The Memory Hierarchy

Reading from disk is many times slower than from memory

- Memory access is as fast as Chuck Norris,
- whereas disk access is slower than the line at the DMV



VS

Why Not Store Everything in Main Memory?

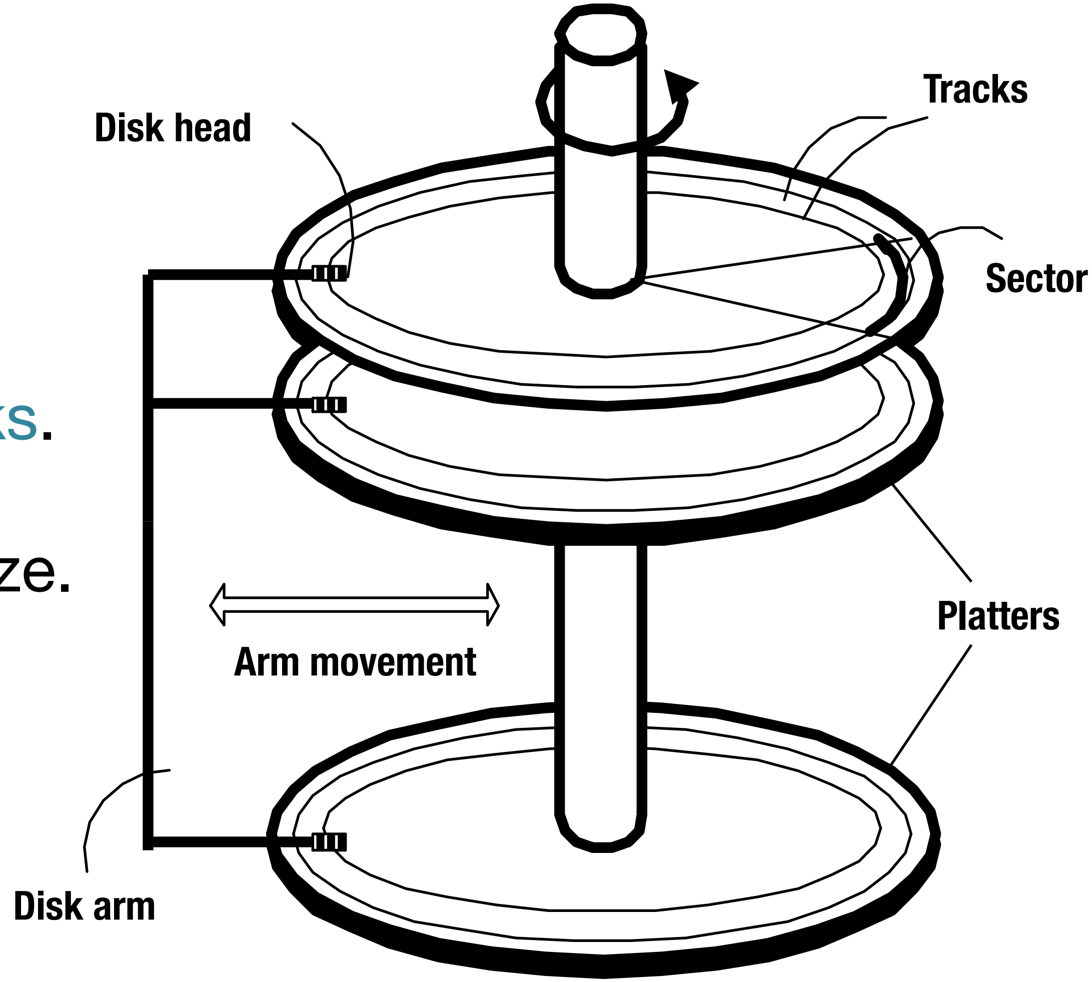
- Too expensive: RAM costs 100-1000x Disk per GB
- Main memory is volatile: Want data to persist between runs
- Typical storage hierarchy:
  - Main memory (RAM) for currently used data
  - Disk for the main database (secondary storage)
    - Non-volatile storage
  - Tapes for archiving older versions of data (tertiary storage)
    - Sequential access devices

Disks

Set of tracks with same diameter called a cylinder.

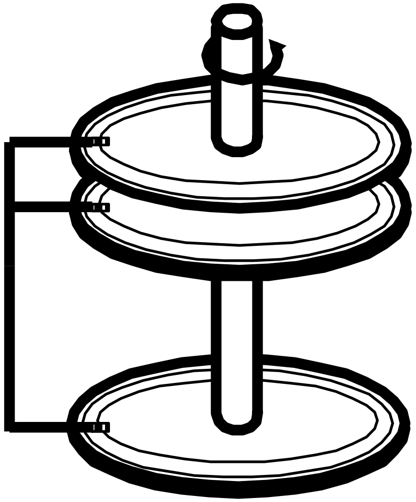
Data stored in blocks. Size of block is a multiple of sector size.

Only one disk head reads or writes at a time.



Performance Implications

- Data must be in memory for DMBS to use.
- Unit of transfer is a block. Whole block must be transferred. Reading or writing a disk block is called an I/O.
- Disk geometry affects access time (hard drives)
    - Seek time: time to move disk head to appropriate track
    - Rotational delay: time waiting for block to move under disk head
    - Transfer time: time to read or write block once head is positioned



Arranging Blocks on Disk

Access time = seek time + rotational delay + transfer time

- GOAL: Minimize seek time and rotational delay
- 'Next' block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Arranging blocks so they are read and written sequentially is important to reducing time spent doing disk I/O

Comparison

- Data must be brought into memory to be read/written
- Typically: Memory size << Disk size
- Why?
  - Cost: memory (100x), SSD(10x), HD (1x)
  - Performance:
    - Seq I/O: memory (50x) vs. SSD (5x) vs. HD(1x)
    - Random I/O: memory (50x) vs. SSD (2-5x) vs. HD(0.01x)

Hard drives much cheaper, but also slower. Arrange in a hierarchy to get the best speed at the lowest cost.

Disk Properties

- Comparison with main memory
  - non-volatile (disks) vs. volatile (main memory)
  - Both random-access
  - Unit of access: one or more blocks (disks) vs. a byte or word
  - Speed: very slow (disk) vs. fast (memory)
- Another kind of drive: solid state
  - Cost/MB: 10x over hard drives currently.
  - Can provide lower latencies and higher throughput than hard drives
  - Access times independent of block placement
- Hybrid non-volatile storage:
  - Use a combination of solid state drives and hard drives

Disk Performance

- Traditional disks
  - Sequential I/O faster than random I/O
- Solid-state drives becoming popular
  - Same speed: sequential or random I/O (Layout not relevant)
  - 10x higher cost, but lower latencies
- Most databases today still use traditional disks, but could change over time
- Hybrid drives that use solid state drives as a cache also becoming common

Pages and Records

- Databases store data on disks
- Unit of information for disks is a page
  - Physical abstraction
  - Page size is typically 4KB to 16KB
- How are tables stored in pages?
  - A table is stored as a *file of records*
  - Records are logical units
  - Records stored in pages
  - The term "file of records" here simply means a named *set* of pages, not necessarily an OS file
- Typically, one page has multiple records
- One table (file) will typically require multiple pages

Check Your Understanding

- Why do databases need disks? Why not just use main memory?
- Which has the lowest per GB cost?
  - Main memory
  - Solid state drives
  - Magnetic drives
- Of the above, for best speed, non-volatile storage, and highest capacity, which of the above would you use?

Operations on File - Example

- Employees (Name, Age, Salary)
- Operations
  - Scan: Fetch all employees from disk
  - Equality Search: age = 21
  - Range Selection: age >= 18 AND age < 65
  - Insert a record
  - Delete a record

How do we organize the file
to accomplish these tasks efficiently?

File Organization

- Each record in a file has a unique Record ID (RID), which is sufficient to locate the record on the disk
  - e.g. an RID may be (page#, offset, length)
- Some methods of arranging file of records on disk
  - Heap (random order) files: No particular order defined for records
  - Sorted Files: Records sorted based on one or more attributes
  - Indexes: Organize records using trees or hashing

Indexes

- A data structure that organizes data records on disk to optimize certain operations.
- Speed up selections on the search key field of the index (denoted k)
  - Any subset of the fields of a relation can be the search key for an index on the relation
  - Search key is not the same as key (minimal set of fields that uniquely identify a record in a relation)
- An index file contains a collection of data entries k* for each search key value k
  - k* should allow us to get to the record contents

Data Entries k*

- A data entry k* must give us a way to get to the data for k:
- Alternative 1: Data entry k* is an actual record (with search key k)
  - Index File == File of Records
- Alternative 2: Data entry k* is (k, rid) pair, where rid refers to a record with search key k
  - Actual data records stored in a different file
- Alternative 3: Data entry k* is (k, rid-list) pair, where rid-list refers to list of records with search key k

How many indexes can use Alternative 1?

Choosing Among Alternatives

- Only one index can use alternative 1
    - Otherwise, you would get redundancy
    - Other indexes can be Alternative 2 or 3
- Alternative 3 is more compact than alternative 2, but leads to variable-length index entries

Check Your Understanding

Can search key k be a composite value,
e.g. pair of values or attributes?

Can a file (relation) have more than one index?

Can a file (relation) have more than one index
using data entry alternative 1?

Index Design Space

Organization Structure for k*

- Hash-based

  (+) Equality search
- Tree-based

  (+) Range, equality search
  - B+Tree (dynamic)
  - ISAM (static)

Data Entry (k*) Contents

1. Actual Data record

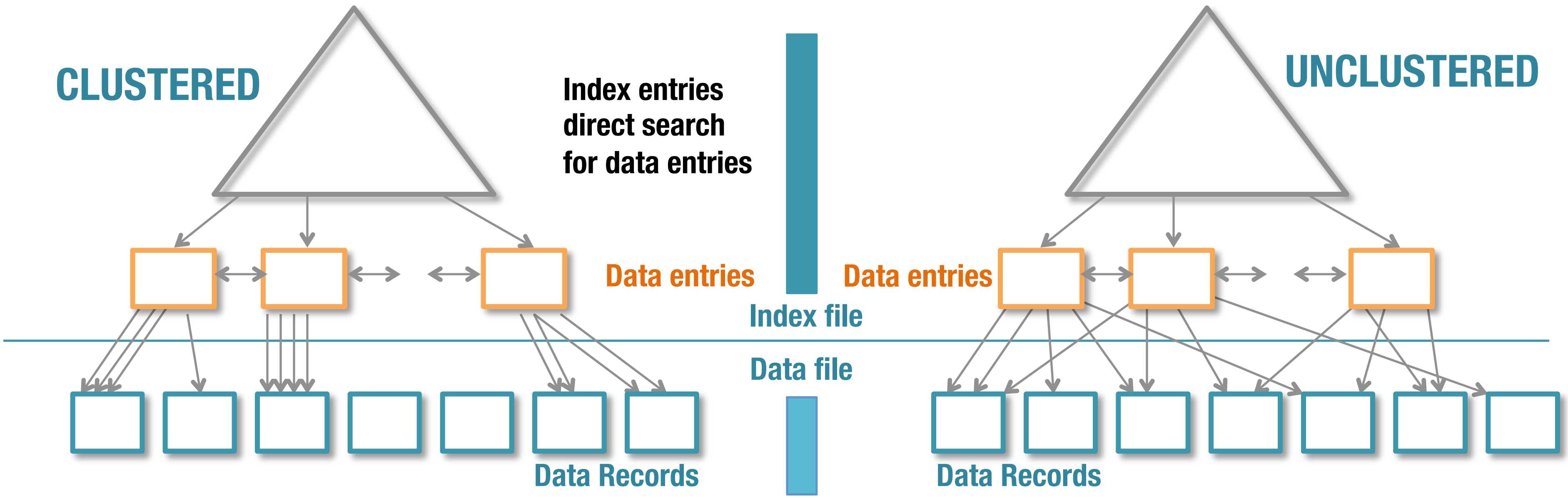
   index = file
2. <k, rid>

   actual records in a diff file
3. <k, list of rids>

Indexing Terminology

- Primary vs. secondary index: If search key contains primary key, then called primary index
  - Otherwise, called secondary index
- Clustered vs. unclustered index: If order of data records is the same as, or 'close to', order of data entries, then called clustered index
  - Alternative 1 implies clustered index. A file that uses Alternative 1 is also called a *clustered file*
  - But not every clustered index uses Alternative 1
- A file can be clustered on at most one search key

Clustered vs. Unclustered Index



- Suppose: data records in heap file, index with Alt. 2
- To build clustered index, first sort the Heap file

Suppose you have an index on Age,
and you want to do a range selection (e.g., Age >= 18)
Which index would you prefer? Why?

Cost of a Search Query

- Cost of Page I/O >> memory read
- # of pages read more important than # of records read
- Searching a heap (unordered)
    - All pages must be read

Search on Unclustered Index

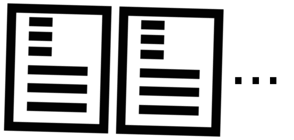
- Let's say you want to **find records** of U. of Michigan **students who are 18 years old**.
- Unclustered index on age
- 25% of students are 18 years old
- Each page can hold 10 records
- 100,000 records
- How many pages are you likely to read?
  - Close to 100%, i.e. 10,000 pages?
  - Close to 25%, i.e. 2,500 pages?

Search on Clustered Index

- Let's say you want to **find records** of U. of Michigan **students who are 18 years old**.
- **Clustered index** on age
- 25% of students are 18 years old
- Each page can hold 10 records
- 100,000 records
- What percent of pages are you likely to read, taking advantage of the index?
  - Close to 100%, i.e. 10,000 pages?
  - Close to 25%, i.e. 2,500 pages?

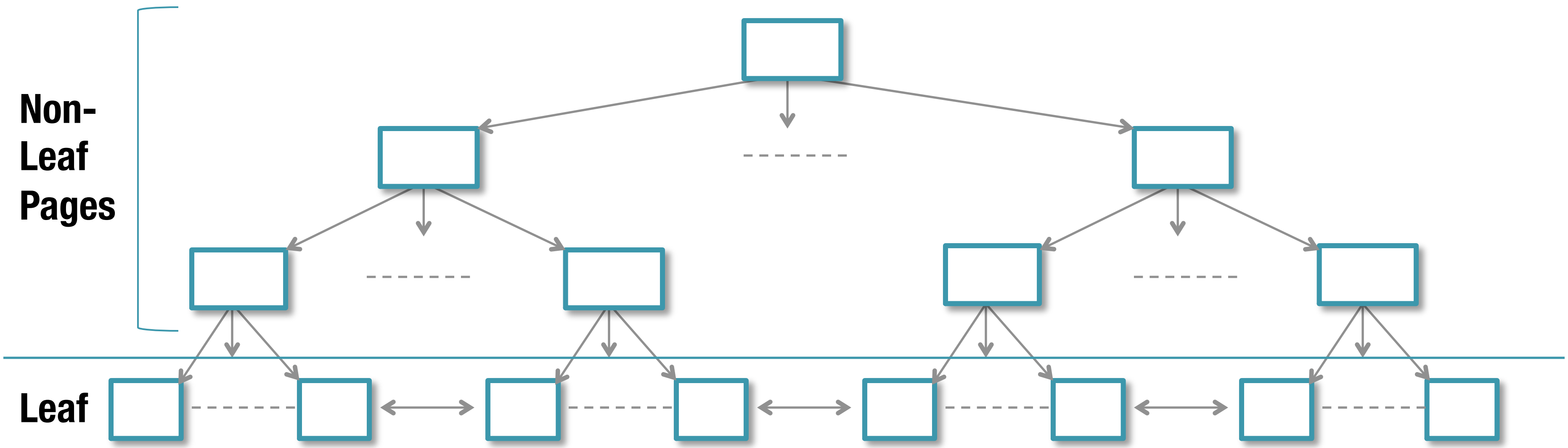
Search on Clustered Index

- In practice, pages are often only around 2/3rd full because deleting records will leave holes in each page over time

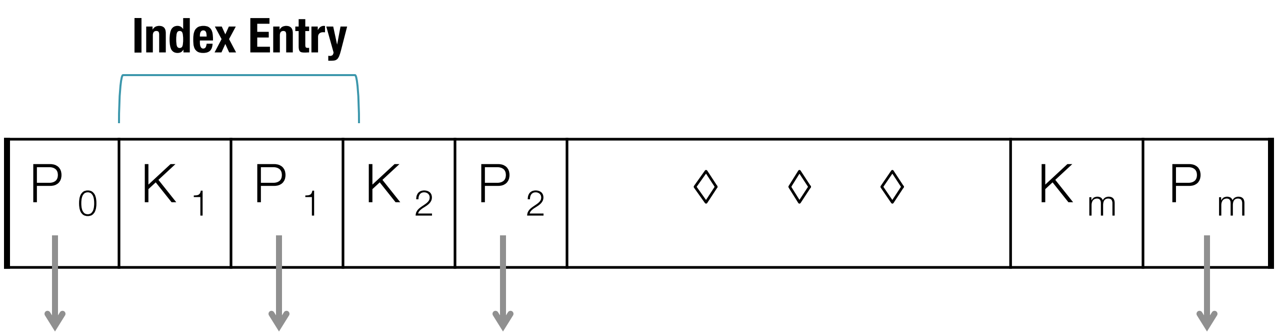


- Thus, the file will occupy around 15,000 pages = 10,000 * 3/2
- So, in the last problem, number of pages fetched will be approximately:
    - 3,750 pages (25% of 15,000 pages)

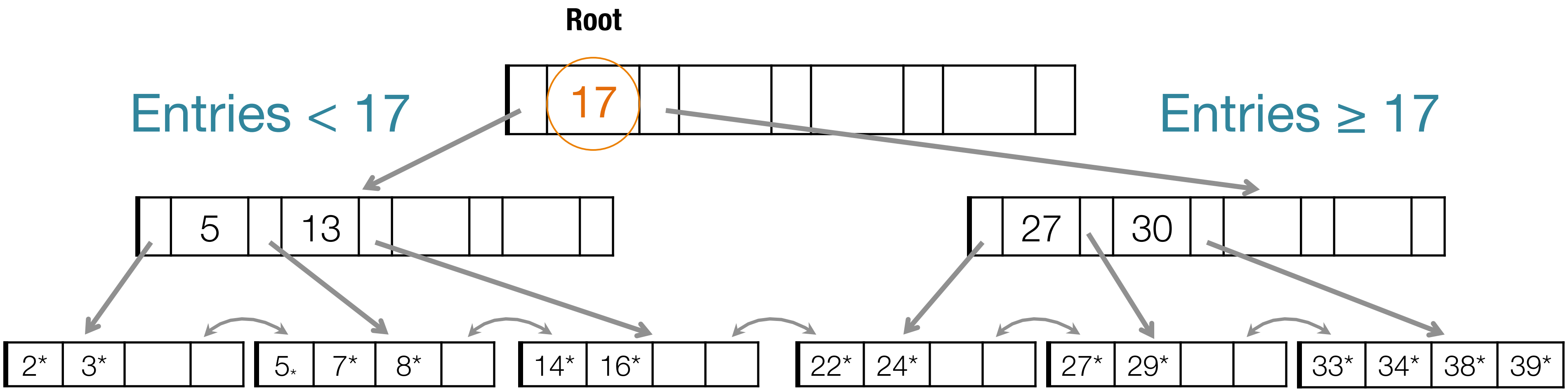
B+ Tree Indexes



- Leaf pages contain data entries, and are chained (prev & next)
- Non-leaf pages contain index entries and direct searches:



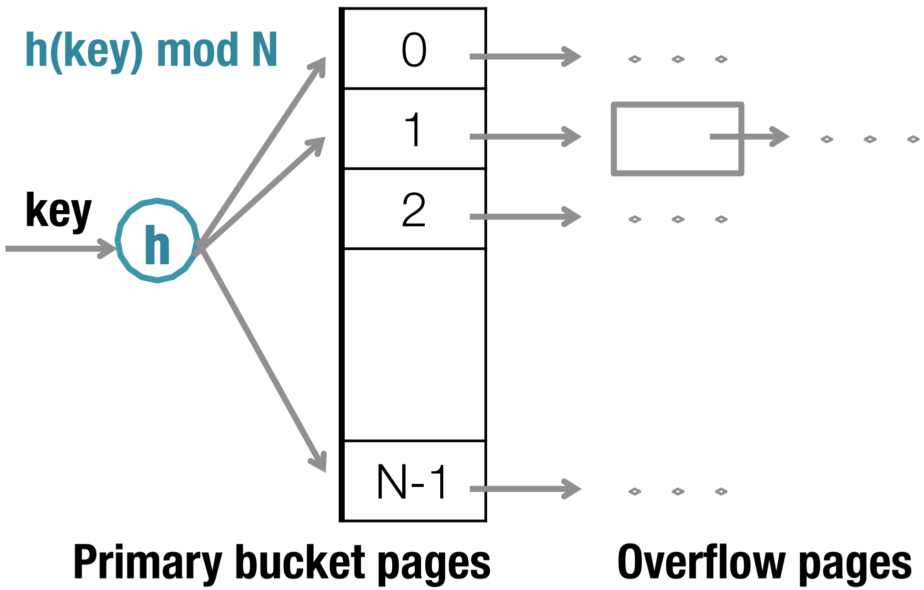
Example B+ Tree



- Find 29*? 28*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it
  Need to adjust parent sometimes
  - And change sometimes bubbles up the tree

Hash-Based Indexes

- Good for equality selections
  - Index is a collection of buckets. Bucket = **primary** page plus zero or more **overflow** pages
  - Hashing function **h**: **h**(*r*) = bucket in which record *r* belongs
    h looks at the search key fields of *r*



Buckets contain:
- If Alternative (1) is used → data rec
- Alternative 2 → <key, rid>
- Alternative 3 → <key, rid-list> pairs

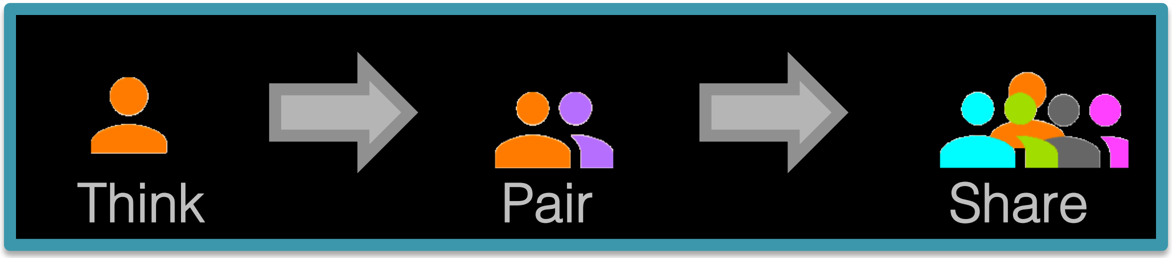
Comparing File Organizations & Indexes Example

- `Employees(Name, Age, Salary)`
- 10 records/page and 10,000,000 records
- Assume that people are uniformly distributed between ages 1 to 100
- Operations:
  - Fetch the names of employees:
    - Scan: all employees
    - Equality Search: age = 21
    - Range Selection: age >= 18 AND age < 65
  - Insert a record
  - Delete a record

Analysis of I/O Cost

Assumptions

- For simplicity, ignore CPU cost
- Important Factors for I/O Cost:
  - How many pages (approx) read/written?
  - Are I/Os sequential or non-sequential?
- With your neighbor, do this for:
  - HEAP and Clustered index on Age
  - All 5 ops: SCAN, EQUALITY, RANGE, INS, DEL



Heap File

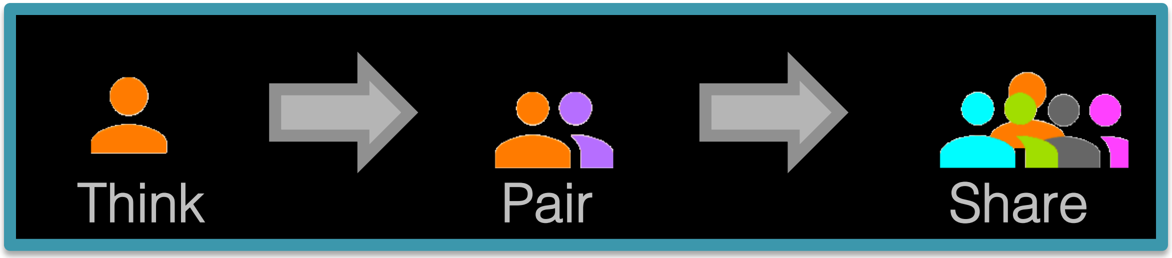
- Scan:
  - Read all pages in files sequentially
- Equality Search (age = 21):
  - Read all pages in files sequentially
  - worst case
- Range Selection on age:
  - Read all pages in file sequentially
- Insert:
  - Read and write last page or write into a new page (2 page I/Os)
- Delete:
  - Searching cost (expensive) + rewrite the page with the record

Clustered Index (on Age)

- Scan:
  - Approx. 1.5 x heap file due to 2/3rd page occupancy (sequential)
- Equality Search (age = 18):
  - Traverse height of tree, read records page (non-sequential)
  - If multiple tuples qualify, read them sequentially
- Range Selection:
  - Traverse height of tree (non-sequential)
  - Scan leaf records (sequential)
- Insert:
  - Traverse height of tree (non-sequential) + write
- Delete:
  - Traverse height of tree (non-sequential) + write

Analysis of I/O Cost

- With your neighbor, do this for:
  - Heap-with-Unclustered-**Tree**-Index (on Age)
  - Heap-with-Unclustered-**Hash**-Index (on Age)
  - All 5 ops: SCAN, EQUALITY, RANGE, INS, DEL



Heap File w/ Unclustered Tree Index (on Age)

- Scan:
- Equality Search:
- Range Selection:

Midterm Review on Thursday

- Insert:
- Delete:

Heap File w/ Unclustered Hash Index (on Age)

- Scan:
- Equality Search:
- Range Selection:
- Insert:
- Delete:

Midterm Review on Thursday

Cost estimation

	scan	eq	range	ins	del
Heap	B	B/2	B	2	Search+1
sorted	B	$\log_2 B$	<- +m	Search+B	Search+B
Clust.	1.5B	h	<- +m	Search+1	Search+1
u-tree	~B	1+h’	<- +m’	Search+2	Search+2
u-hash	~B	~2	B	Search+2	Search+2

Cost estimation - big-O notation:

	scan	eq	range	ins	del
Heap	B	B	B	2	B
sorted	B	$\log_2 B$	$\log_2 B$	B	B
Clust.	B	$\log_F B$	$\log_F B$	$\log_F B$	$\log_F B$
u-tree	B	$\log_F B$	$\log_F B$	$\log_F B$	$\log_F B$
u-hash	B	1	B	1	1