Introduction to Database Systems
I2DBS – Spring 2023

- Week 8:
- Indexes

Jorge-Arnulfo Quiané-Ruiz

Readings:

PDBM 12

Information

- **Homework 2:**
  - We will provide feedback and solution ASAP
- **Exercise 8 is online**
  - It is about impact of indexes
  - = Topic for today (and partly next week)
- **Homework 3**
  - It will be online early next week (or before)
  - Yet, you have 2 working weeks to solve it: deadline on April 21st
  - Topic = DDL + Normalization + Indexing (+ SQL)
    - Rather extensive … but you need to learn all of this!
    - Exercise 8 is very useful as preparation!

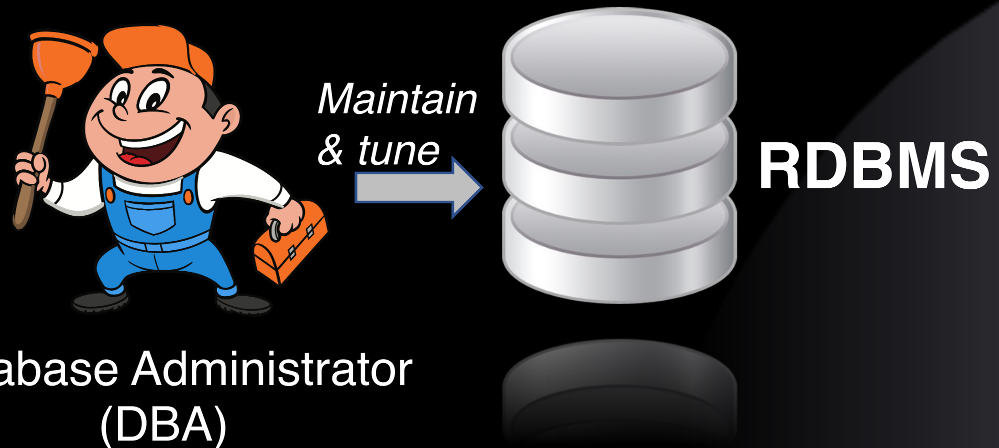
Profile of the Week

IT University of Copenhagen

Edward M. McCreight

Co-Inventor of B-Tree Data Structure

- **1940:** USA
- **1968:** co-invented B-Tree
  "A Space-Economical Trie Storage Structure"
- **1969:** PhD in Computer Sciencefrom Carnegie Mellon
- **Institutions:** Boeing, Xerox, Adobe



Database Administrator
(DBA)

Indexes

Readings:

PDBM 12

Initial Case Study

- **Which of these queries should run faster? How much faster?**

```
(1) SELECT COUNT(*) FROM movie
     WHERE year=1948;

(2) SELECT COUNT(*) FROM movie
      WHERE year=1920 or year=1924 or year=1928 or year=1932
         or year=1936 or year=1940 or year=1944 or year=1948;
```

(Q1) If ran several times, would the first run be slower? Why?

(Q2) If so, how big would be that difference?

(Q3) Would an index improve performance?

Seeing Whether an Index is Used

- **EXPLAIN ANALYZE can be used to show PostgreSQL's query plan**

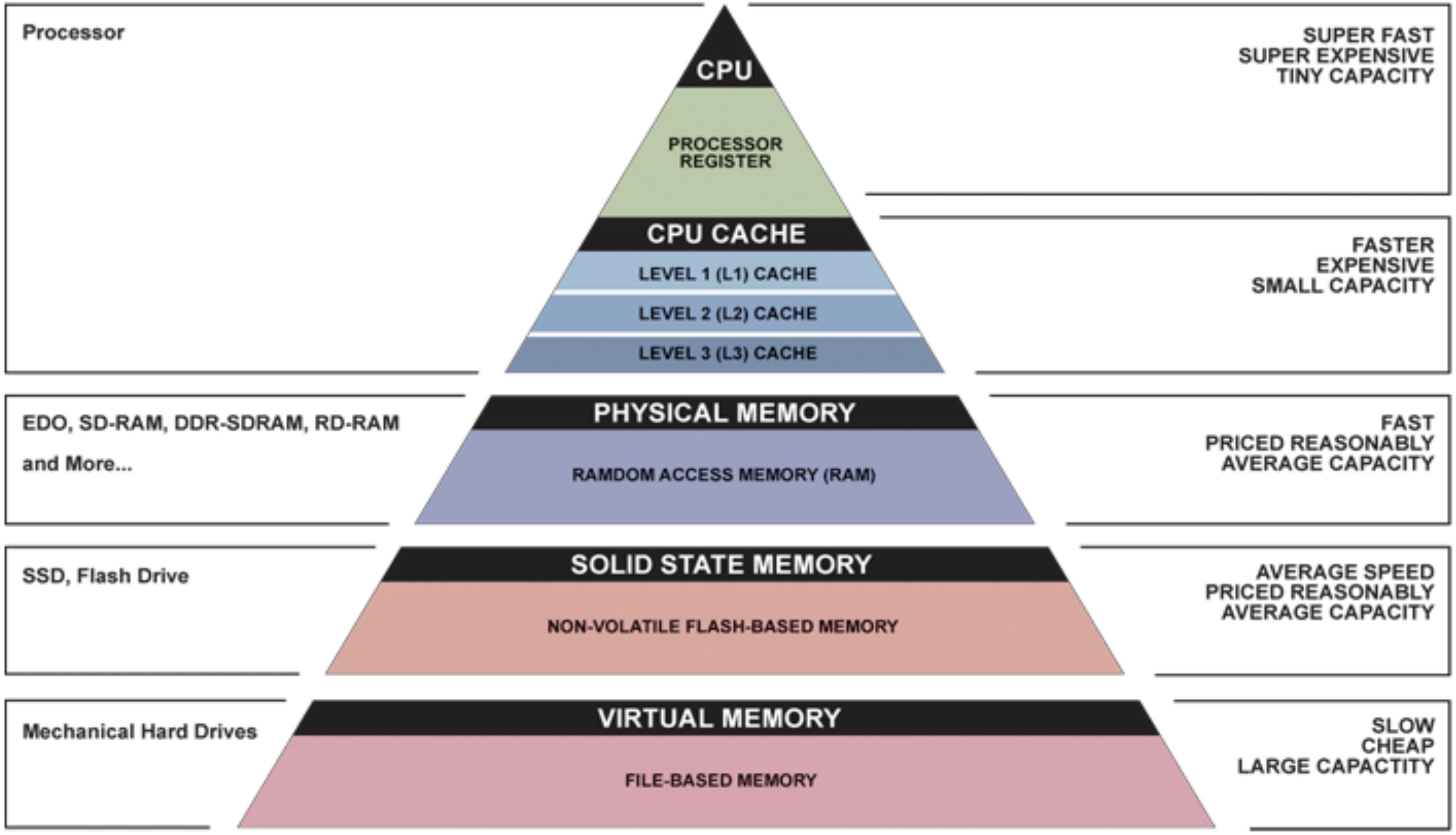
(1) EXPLAIN ANALYZE

```
SELECT COUNT(*) FROM movie
WHERE year=1948;
```

(2) EXPLAIN ANALYZE

```
SELECT COUNT(*) FROM movie
WHERE year=1920 or year=1924 or year=1928 or year=1932
    or year=1936 or year=1940 or year=1944 or year=1948;
```

Disk Recap

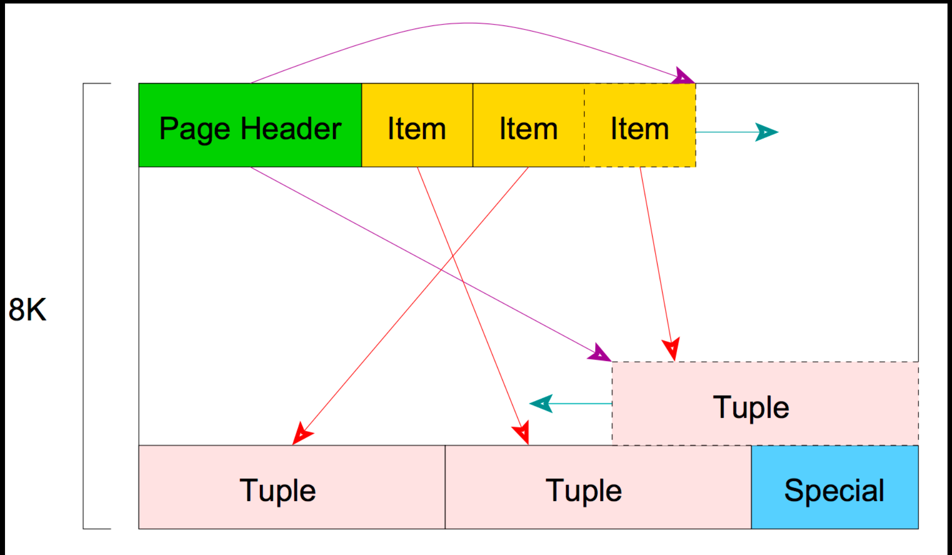


▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Impact of Disk-Based Storage

- **Unit of disk reads: KBs**
  - We cannot read a few bytes from disk!
  - Traditional DBMS: 8KB / 16KB
  - Linux: 128KB
  - How many employee records fit in one disk read?

- **Historically:**
  Avoid reading in random order
  - Penalty 1: Random reads costly (but SSDs!)
  - Penalty 2: Have to read the same page often!



Random or Sequential?

- **Example:**
  - Table = 2B rows = 1 TB
  - Memory = 256 GB ⟹ only small part of the table fits in RAM!
  - Disk read = 128 KB ⟹ 8M sequential reads vs 2B random reads!
  - Cost of reading records in sequential or random order?

- **Costs:**
  - HDD:
    - Sequential = ~8M SRs = 1.5 hours
    - Random = ~2B RRs = 114 days
  - SSD:
    - Sequential = ~8M SRs = 7 minutes
    - Random = ~2B RRs = 30 hours

	ms/IO	IOs	msec	sec	min	hours	days
HDD	0,642	8.000.000	5136000	5136	86	1,43	0,06
	4,930	2.000.000.000	9860000000	9860000	164.333	2.738,89	114,12
SSD	0,056	8.000.000	448000	448	7	0,12	0,01
	0,055	2.000.000.000	110000000	110000	1.833	30,56	1,27

Full Table Scans

- **When a DBMS sees a query of the form:**

```
SELECT *
FROM R
WHERE <condition>
```

It reads through all the tuples of R and report those tuples that satisfy the condition.

Selective Queries

- **Consider the query from before:**

```
SELECT *
FROM R
WHERE <condition>
```

- If we have to report 80% of the tuples in R, it makes sense to do a full table scan.
- On the other hand, if the query is very selective, and returns just a small percentage of the tuples, we might hope to do better.

Point Queries

- **Consider a selection query with a single equality in the condition:**

```
SELECT *
FROM person
WHERE birthdate='1975-02-06';
```

- **This is a point query: We look for a single value of birthdate.**
  - We may still return > 1 record!

- **Point queries are easy if data is sorted by the attribute used in the condition.**
  - How? What algorithms would work?

Range Queries

- **Consider a selection query of the form:**

```
SELECT *
FROM person
WHERE birthdate BETWEEN '1975-02-01' and '1975-02-28';
```

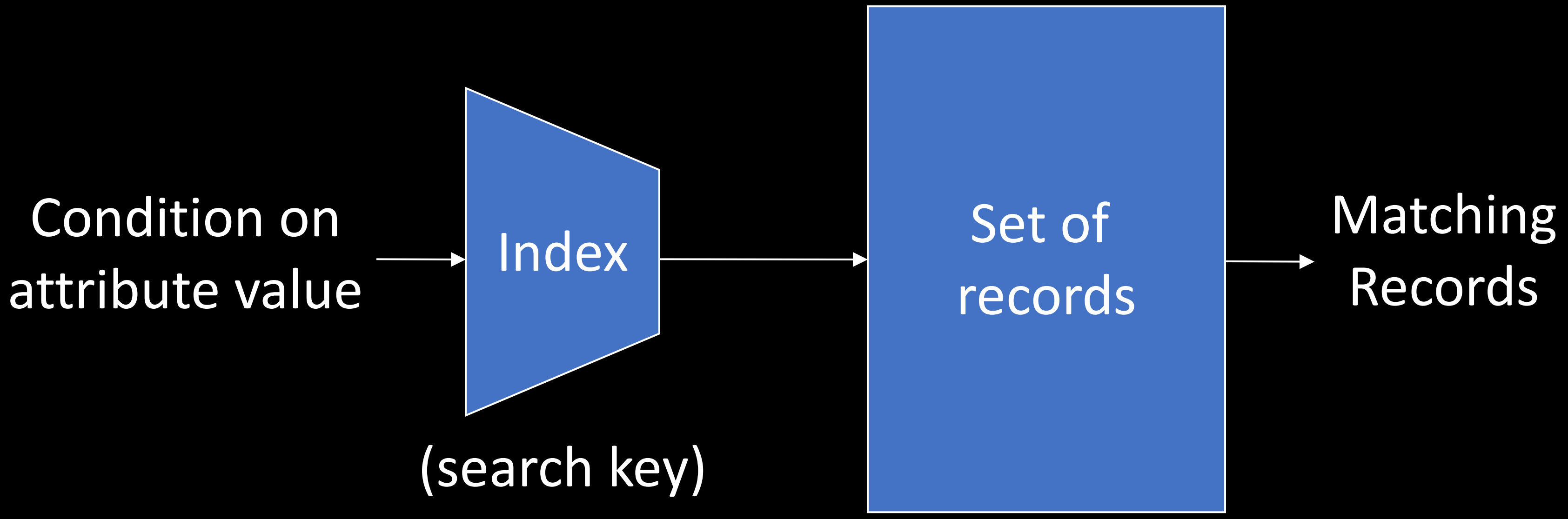
- **This is a range query: we look for a range of values of birthdate.**
- **Range queries are also easy if data is sorted by the right attribute.**
  - But often not as selective as point queries.

Indexes

- **To speed up queries the DBMS may build an index on the birthdate attribute.**
- **A database index is similar to an index in the back of a book:**
  - For every piece of data you might be interested in (e.g., the attribute value 1975-02-06), the index says where to find the row with the actual data!
  - The index itself is organized such that one can quickly do the lookup.
- **Looking for information in a relation with the help of an index is called an index scan (range) or index lookup (point)**

Indexing

- **An index is a data structure that supports efficient access to data**
  - In databases, indexes are also stored on disk



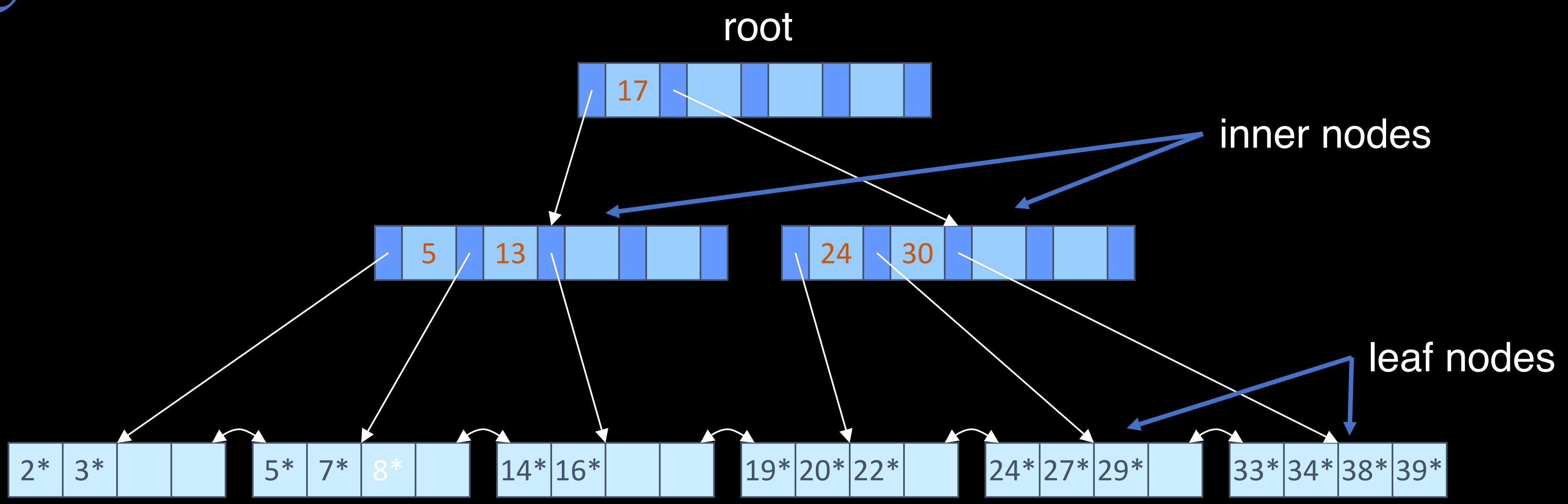
Two Techniques

- **Two basic techniques dominate in modern DBMSs:**
  - Hashing: Use a fixed transformation algorithm to convert the attribute value into a database address
  - Tree search: A dynamic search structure is built that guides the search for a given attribute value to the proper database location
- **Hashing supports equality queries only.**
  - Typically used dynamically for large-scale joins
  - Rarely available to developers
- **Tree search is more versatile and accessible**

B+ Trees

- **The most common index type**
  … in relational systems
- **Supports equality and range queries**
- **Dynamic structure**
  - Adapts to insertions and deletions
  - Maintains a balanced tree

A Sample B+-tree



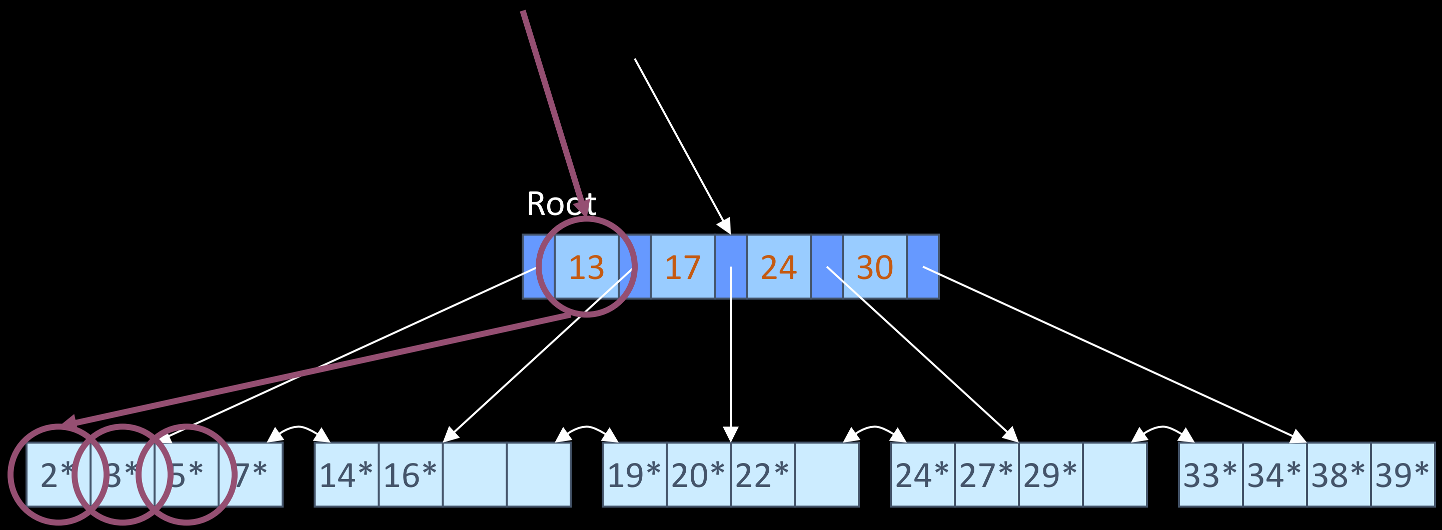
Each inner/leaf node is one disk page

Nodes have a minimum & maximum capacity

- **X* represents (search key, pointer list) pairs (X, [address of tuple X1, ...])**
  - Unique index: Only one entry in list
- **Key values are sorted: K1 ≤ ... ≤ Kd (d is maximum capacity or order of node)**
  - For any two adjacent key values Ki, Ki+1 the pointer Pi points to a node covering all values in the interval [Ki, Ki+1)

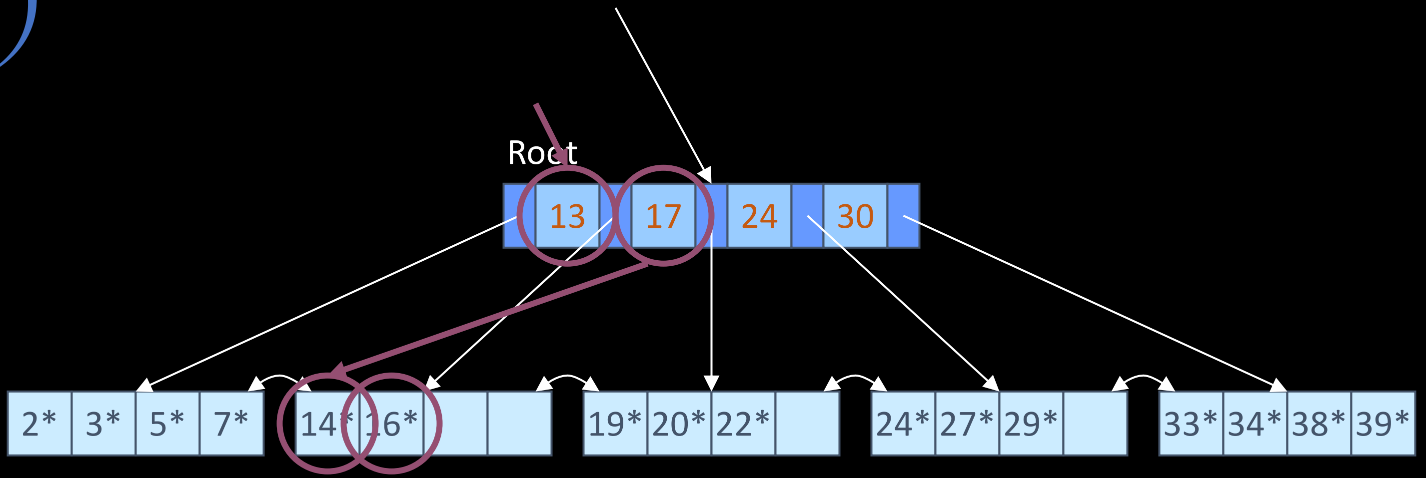
Searching

- Begin at the root
- Comparisons guide the search to the appropriate leaf
- Ex: Find 5*

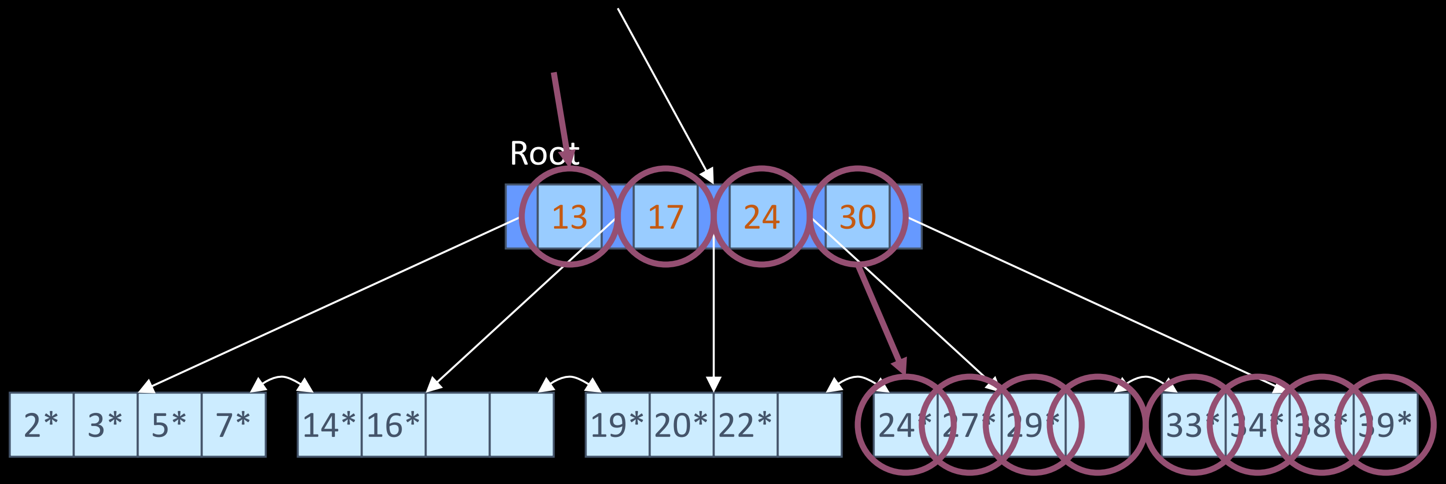


Searching (II)

- Ex: Find 15*

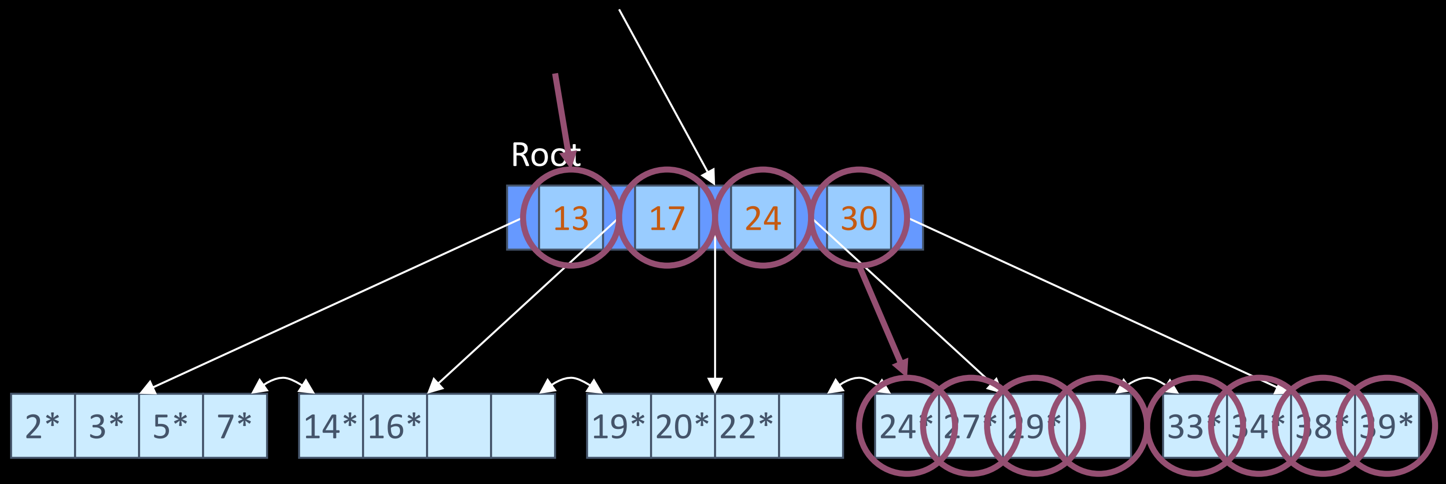


- Ex: Find all records >=24*



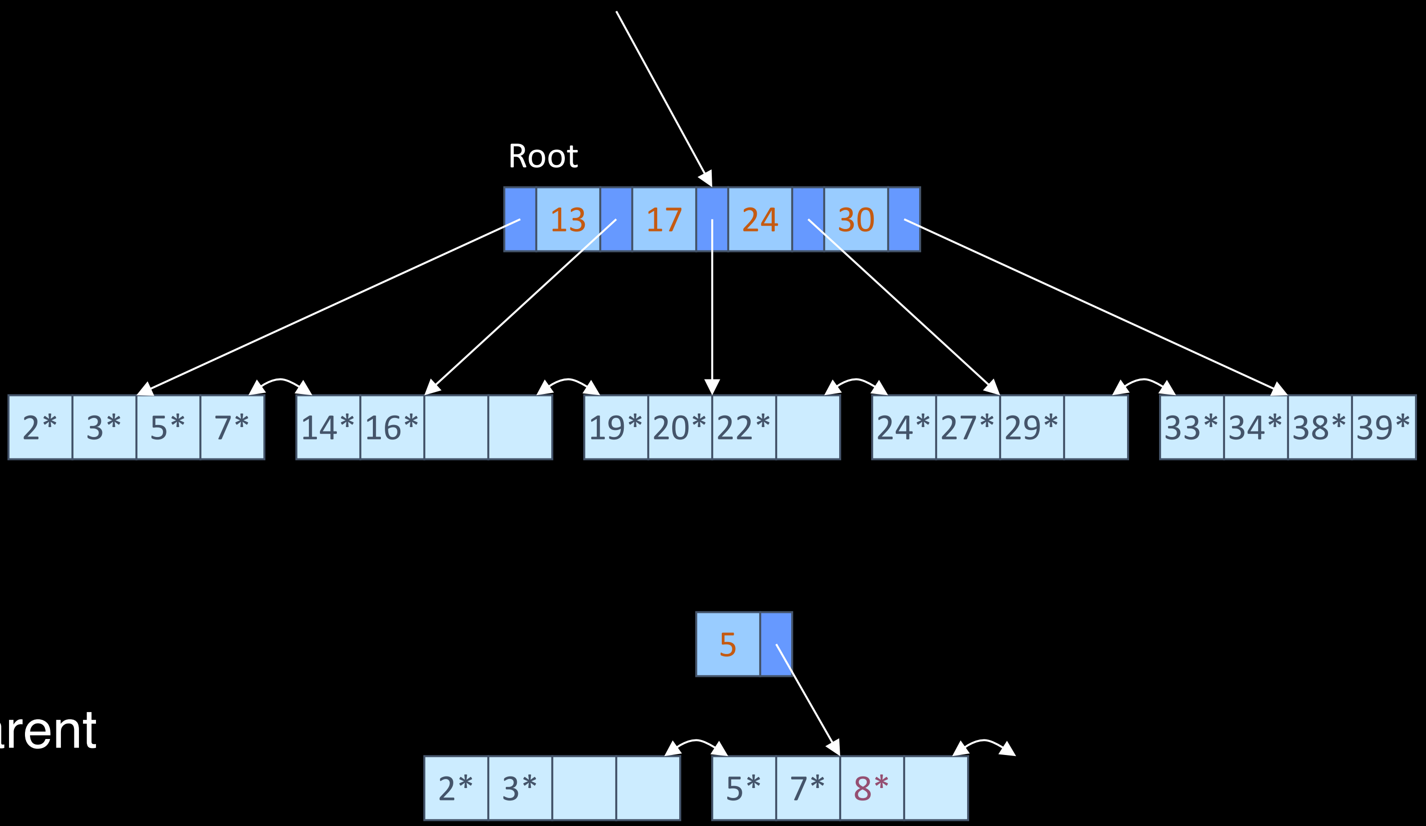
Intra-Node Searching

- We have used scans
- B+-tree nodes have hundred(s) of key values
- Use binary search!



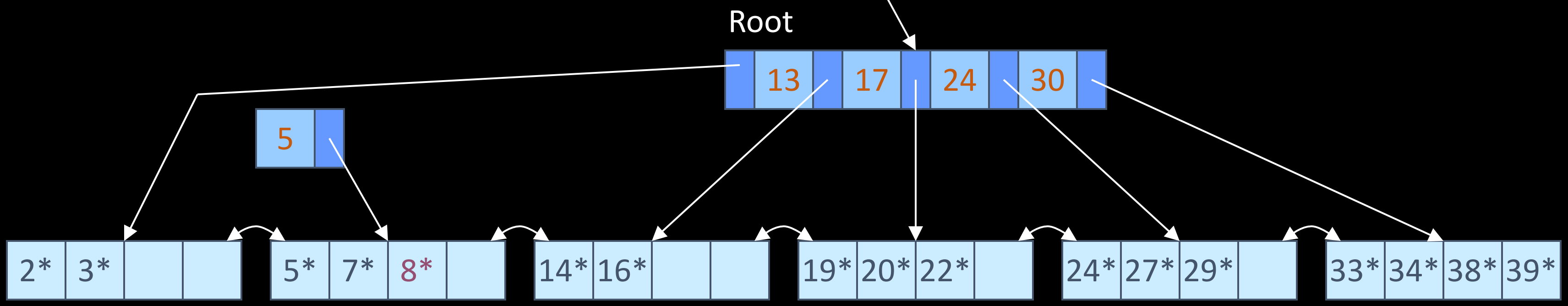
Inserting

- **Leaf is full, must therefore split**
- **Root is full, must therefore split**

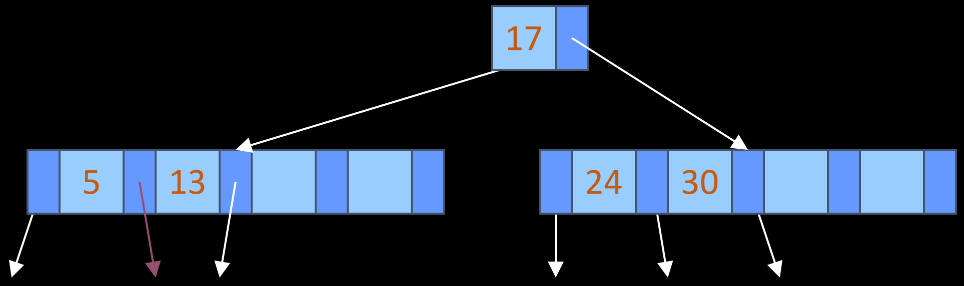


- **Inserting 8* Example**
  - First split the leaf
  - Copy middle search key to the parent

Inserting 8* Example (cont.)

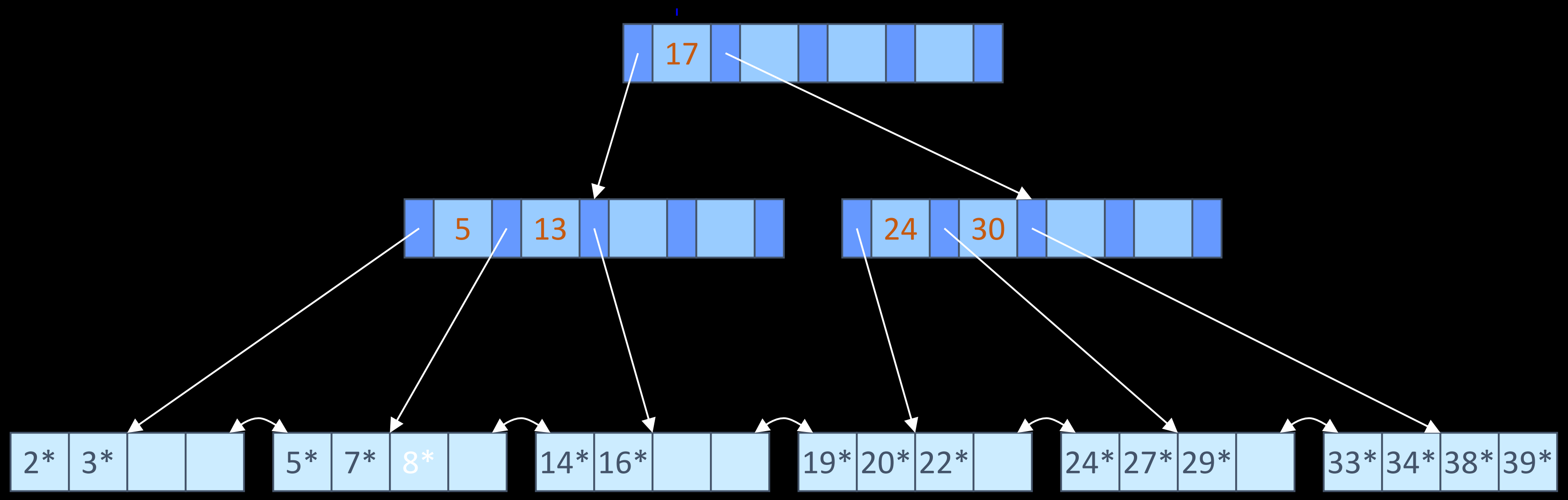


- **Then split the root**
- **Move the middle search key into the new root**



After Inserting 8*

- **Trees grow wider, then higher**



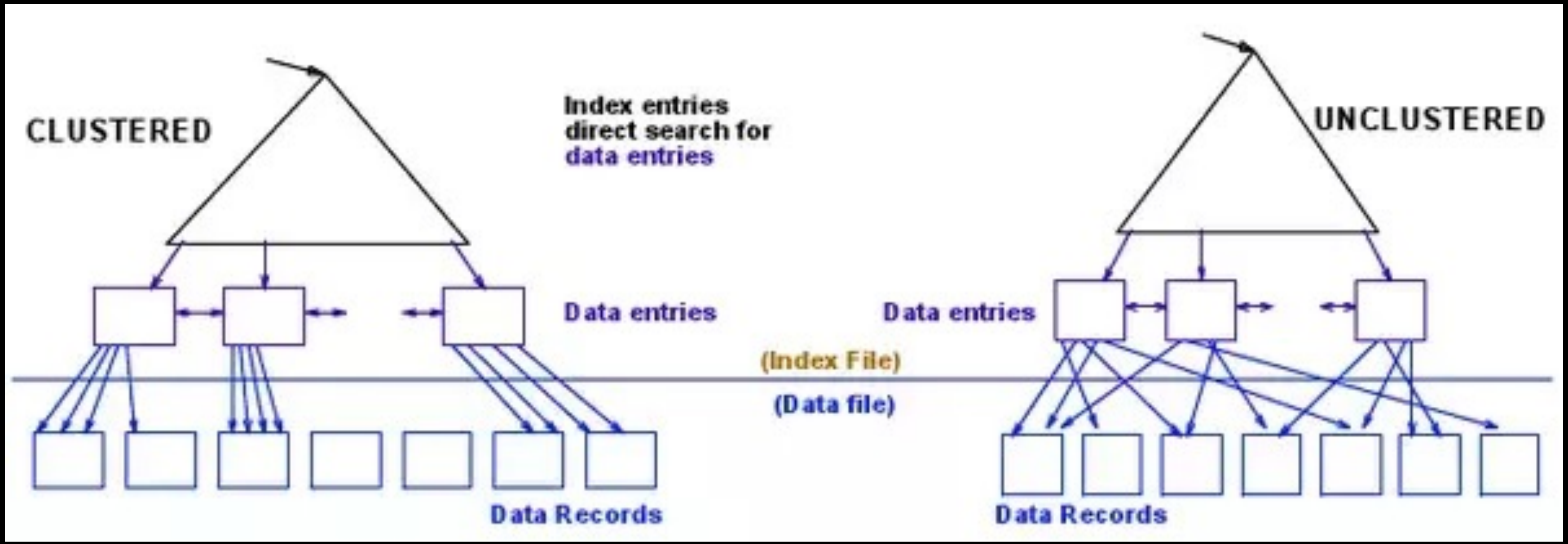
Storage Capacity – Some Statistics

- **A typical tree:**
  - Order: 1000 (~= 16 KB per page / 16 b per entry)
  - Utilization: 67% (usual numbers in real life)
  - Fanout: 670
- **Capacity**
  - Root: 670 records
  - Two levels: 6702 = 448.900 records
  - Three levels: 6703 = 300.763.000 records
  - Four levels: 6704 = 201.511.210.000 records
- **Top levels may fit in memory**
  - Level 1 = 1 page = 16 KB
  - Level 2 = 670 pages = 11 MB
  - Level 3 = 448.900 pages = 7 GB

Index Jargon

- Indexes vs. Indices
- Search key vs. Primary key vs. Candidate key
- Unique index vs. Non-unique index
- Primary index vs. Secondary index
- Dense index vs. Sparse index
- Clustered index vs. Unclustered index

Clustered vs. Unclustered Index



https://qph.fs.quoracdn.net/main-qimg-a578a13aeca2112760f11b2084f00065.webp

Clustered Indexes

- **If the tuples of a relation are stored sorted according to some attribute, an index on this attribute is called clustered.**
  - Clustered indexes make point and range queries on the key very efficient
  - Why? Sequential reads + As few reads as possible!

- **Many DBMSs automatically build a clustered index on the primary key of each relation.**
  - PostgreSQL has limited clustering support (later!)

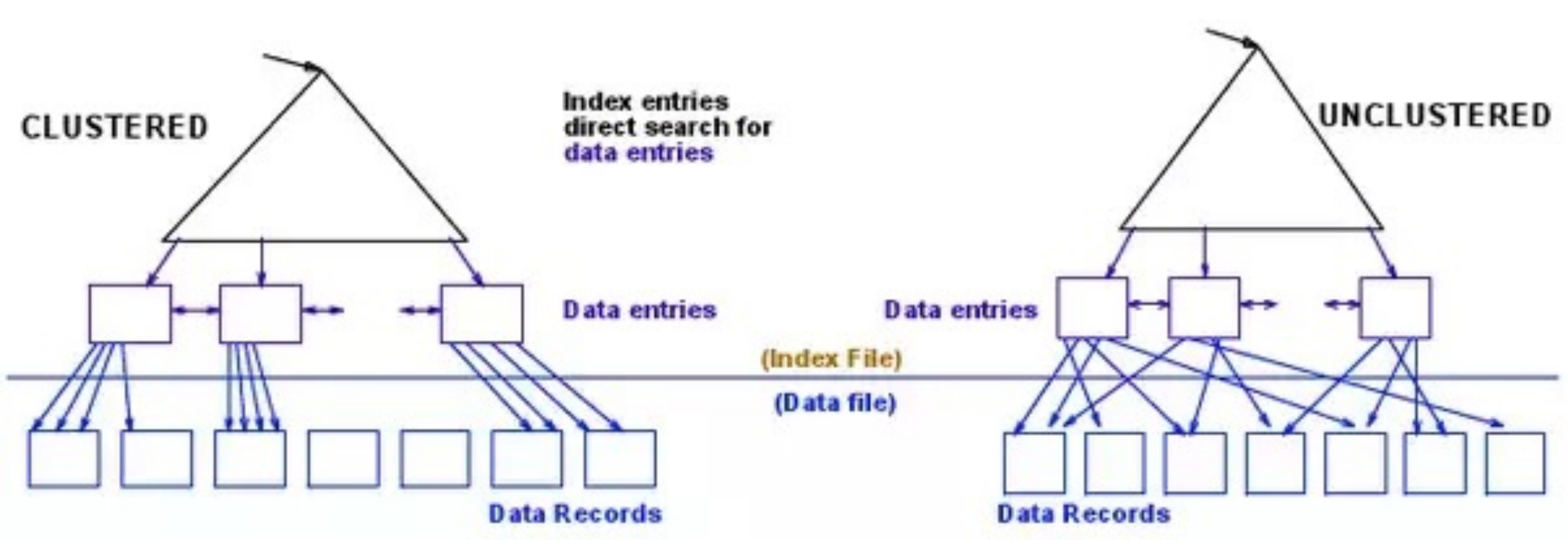
- **A clustered index is sometimes referred to as a primary index.**
  - Can there be more than one clustered? Why?

Unclustered Indexes

- **It is possible to create further indexes on a relation. Typical syntax:**
  - CREATE INDEX myIndex ON involved(actorId);
- **The unclustered indexes are sometimes called non-clustered or secondary indexes.**
- **Unclustered indexes:**
  - Make most point queries more efficient.
  - Make some (narrow) range queries more efficient.

Clustered vs. Unclustered Index

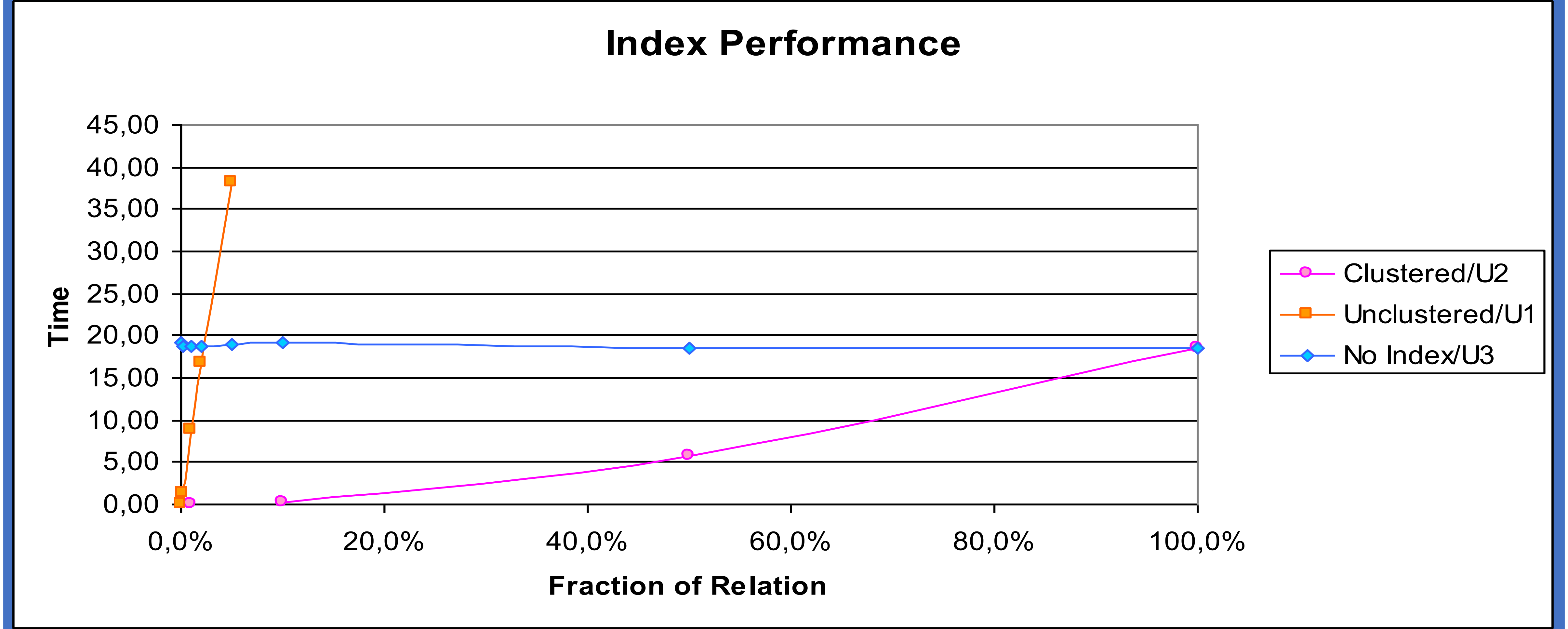
- **To retrieve M records, where M is small:**
  - Clustered: Probably one disk read
  - Unclustered: Probably M random disk reads
- **To retrieve M records, where M is large:**
  - Clustered: Probably M/records_per_page sequential disk reads
  - Unclustered: Up to M random disk reads
- **We still need to read the index itself – same for both!**



Index Scan vs Full Table Scan

- **Point and range queries on the attribute(s) of the clustered index are almost always best performed using an index scan.**
- **Non-clustered indexes should only be used with high selectivity queries.**
  - Old rule of thumb: a secondary index scan is faster than a full table scan for queries returning less than 1% of a relation.
  - New rule of thumb?

Impact of Clustering on Performance



Covering Index

- **An index that contains ALL attributes used in a query is called covering**
  - Resulting query plans are index-only

```
SELECT COUNT(*) FROM movie WHERE year=1948;
CREATE INDEX movieyear ON movie(year);

SELECT name FROM person WHERE height=170;
CREATE INDEX phn ON person(height, name);
```

- **The data from the relation is not needed = no disk reads required to retrieve tuples**
  - Should a covering index be clustered or unclustered?
  - What is what matters then?

Clustered vs. Covering Index

- **To retrieve M records, where M is small:**
  - Clustered: Probably one disk read
  - Covering: Definitely need 0 disk reads
- **To retrieve M records, where M is large:**
  - Clustered: Probably M/records_per_page sequential disk reads
  - Covering: Definitely need 0 disk reads
- **We still need to read the index itself – same for both!**

Practice: Types of Indexes

- **For each of the following queries:**
  - Which index would give the best plan?
  - Would the index be covering?
  - Would you prefer clustered or unclustered index?
  - Based on these queries, if you could choose, which index should be clustered?

(Q1) SELECT * … WHERE birthdate = ‘20-02-2002’

(Q2) SELECT * … WHERE height < 170

(Q3) SELECT * … WHERE ID = 4564

(Q4) SELECT AVG(birthdate) …

Multi-Attribute Indexes

- **Defining an index on several attributes:**

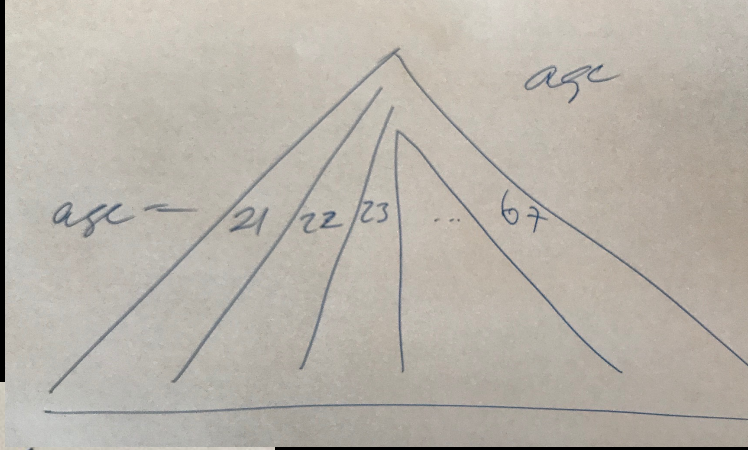
```
CREATE INDEX myIndex
ON person (height,birthdate);
```

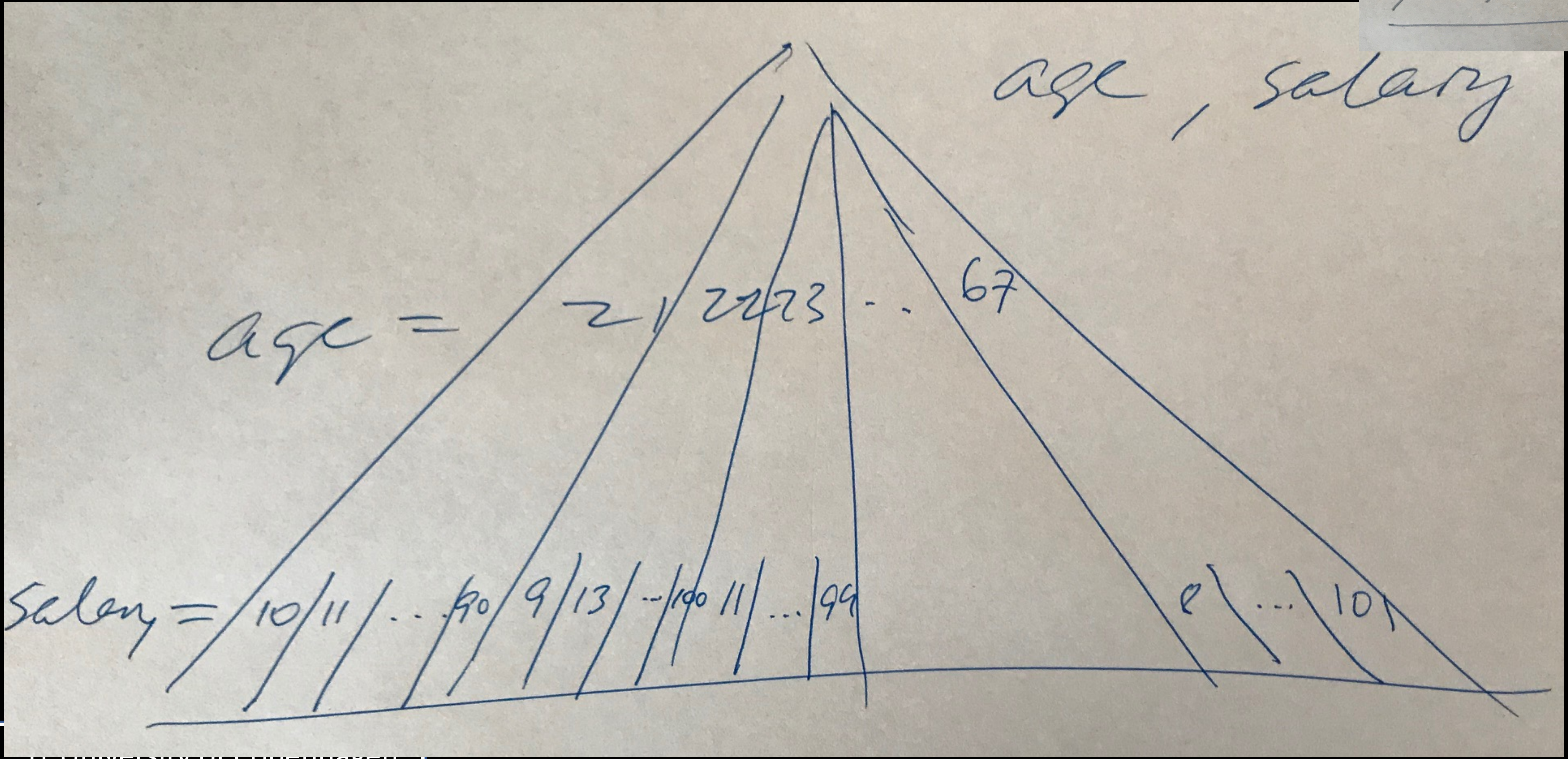
- **Speeds up point/range queries such as:**

```
SELECT *
FROM person
WHERE height=170 and birthdate<'1945-08-08'
```

- **An index on several attributes usually gives index for any prefix of these attributes, due to lexicographic sorting.**

Lexicographic Sorting?





Problem Session

- **Which point and range queries are ”easy” (equality on a prefix, range on one) when the relation is indexed with this two-attribute index?**

```
CREATE INDEX myIndex
ON person (height, birthdate);
```

(Q1) A range query on height?

(Q2) A range query on birthdate?

(Q3) A point query on birthdate?

(Q4) A point query on birthdate combined with a range query on height?

(Q5) A point query on height combined with a range query on birthdate?

Choosing to Use an Index

- **The choice of whether to use an index is made by the DBMS for every instance of a query**
  - May depend on query parameters
  - You do not need to take indexes into account when writing queries

- **Estimating selectivity is done using statistics**
  - In PostgreSQL, statistics are gathered by executing statements such as ANALYZE involved

Choosing Columns

- **Candidates for index search keys**
  - Columns in WHERE clauses
  - Columns in GROUP BY clauses
  - Columns in ORDER BY clause
- **Columns that are rarely candidates**
  - Large columns (too much space)
  - Frequently updated columns (too much maintenance)
  - Columns in SELECT clauses (not used to find tuples)
    - ... but see covering indices!

What Speaks Against Indexing?

- **Space usage:**
  - Similar to size of indexed columns (plus pointer)
  - Most space for leaves, less for tree nodes
  - Not really important!
- **Time usage for keeping indexes updated under data insertion/change:**
  - Depends on the index architecture
  - This is important!

Other Impact of Indexes

The DBMS may use indexes in other situations than a simple point or range query.

- **Some joins can be executed using a modest number of index lookups**
  - May be faster than looking at all data
  - But hash-based joins are usually fastest (next lecture!)
- **Some queries may be executed by only looking at the information in the index**
  - Index only query execution plan (”covering index”)
  - May need to read much less data.
- **Consistency (checking keys and foreign keys)**

Practice: Problem Session

- **What would be good indexes for this query?**

```
SELECT firstNames
FROM person
WHERE gender='m'
  AND firstnames LIKE 'Maria%’;
```

Index types

- **Common:**
  - B-trees (point queries, range queries)
  - Hash tables (only point queries on the whole search key, but somewhat faster)
  - Bitmap indexes (good for ”dense” sets)
- **More exotic:**
  - Full text indexes (substring searches)
  - Spatial indexes (proximity search, 2D range search, multimedia, …)
  - … and way more!

B+-tree Implementations in Some Major DBMS

Postgres
- Cannot specify a clustered index!
- Manual CLUSTER command!
- Is an index on SERIAL clustered?

SQL Server
- Table stored in clustered index
- Primary keys can be unclustered
- Indexes maintained dynamically

DB2
- Table stored in clustered index
- Explicit command for index reorganization

Oracle
- No clustered index until 10g
- Index organized table (unique/clustered)
- Indexes maintained dynamically

MySQL
- Primary key is clustered
- Table stored in clustered index
- Indexes maintained dynamically

Takeaways

- **Large databases need to be equipped with suitable indexes**
  - Need understanding of what indexes might help a given set of queries
  - ‘Key’ distinction: Clustered vs Unclustered
  - A detailed understanding of various index types is beyond the scope of this course

What is next?

Next Lecture:

- Understanding Query Processing