CS-300: Data-Intensive Systems

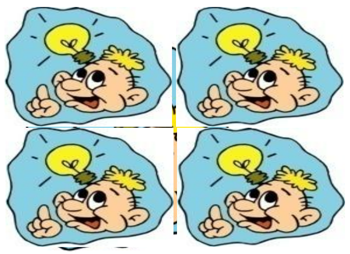
Storage, Files, and Indexing

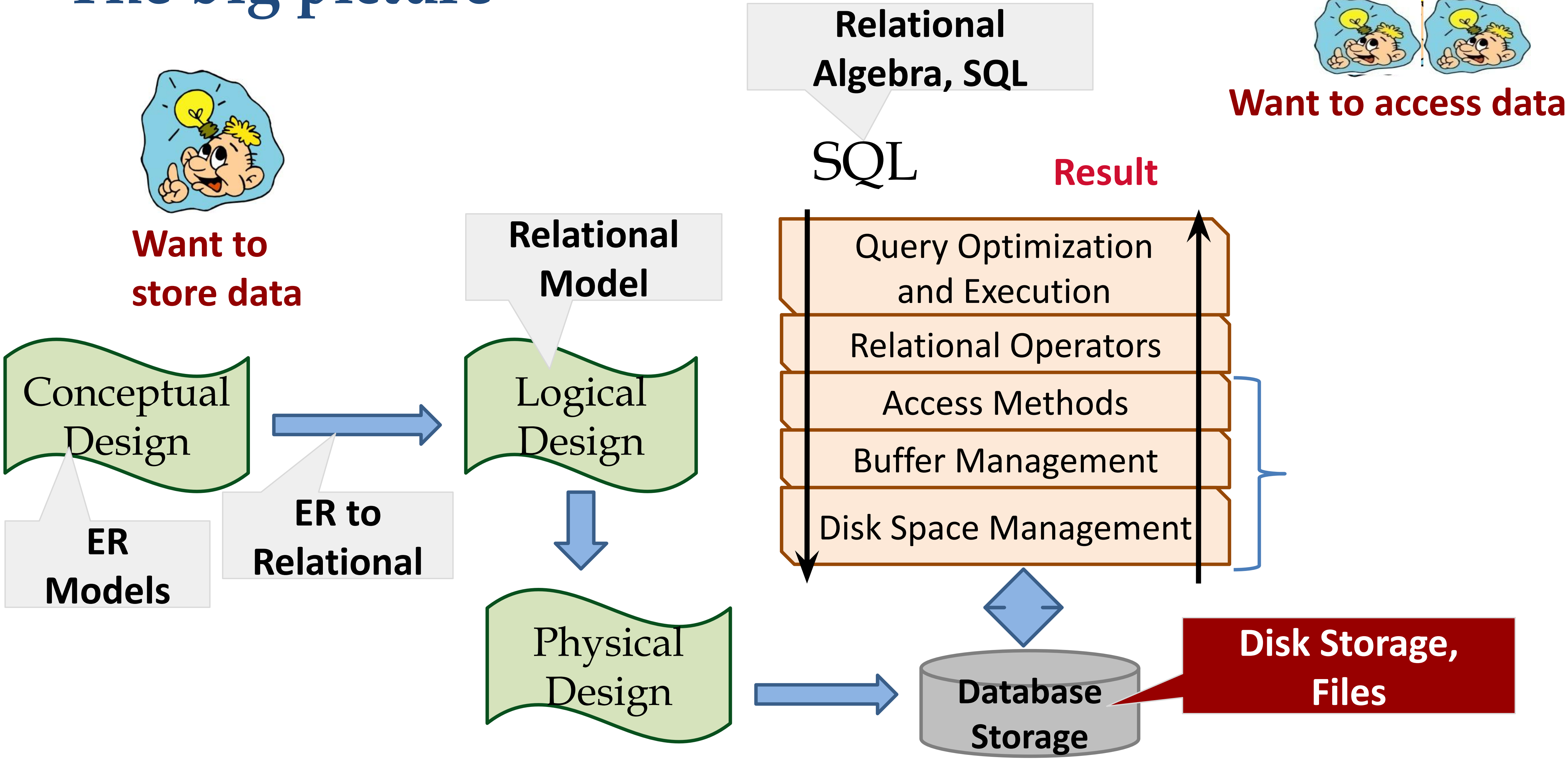
(Chapters 13.1-13.4 14.1 14.2)

Prof. Anastasia Ailamaki, Prof. Sanidhya Kashyap

EPFL

The big picture





File and access layer

Database as a **“file of records”**

- The database is stored as a collection of *files*
- Files are maintained by the underlying OS

- Operations:
  - Create/delete files
  - Insert/delete/modify record
  - Retrieve one particular record (*point access)* Specified using *record id*
  - Retrieve range of records (*range access)* Satisfying some conditions
  - Retrieve all records (*scan*)

File and access methods layer

- **File Organization**
  - How is data organized in files?

- **Indexing**
  - How to make data access efficient?

- **Storage**
  - How is data physically stored on disk?

File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

File organization

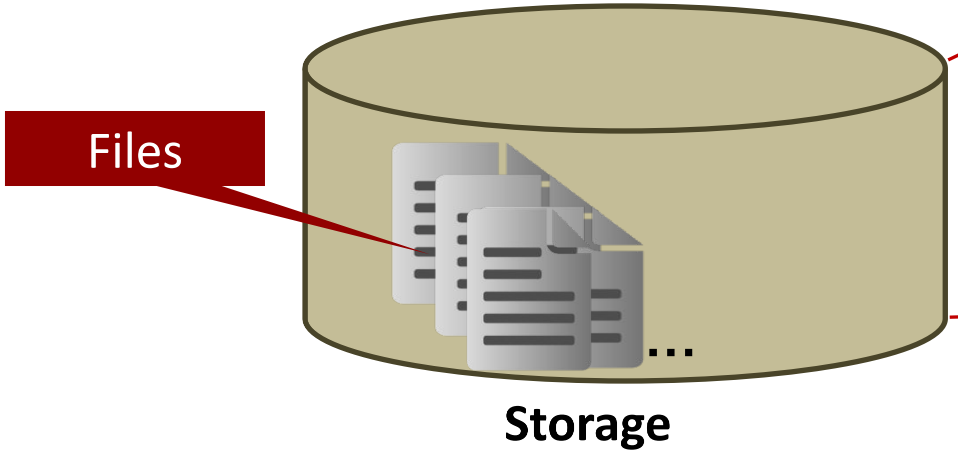
instructor

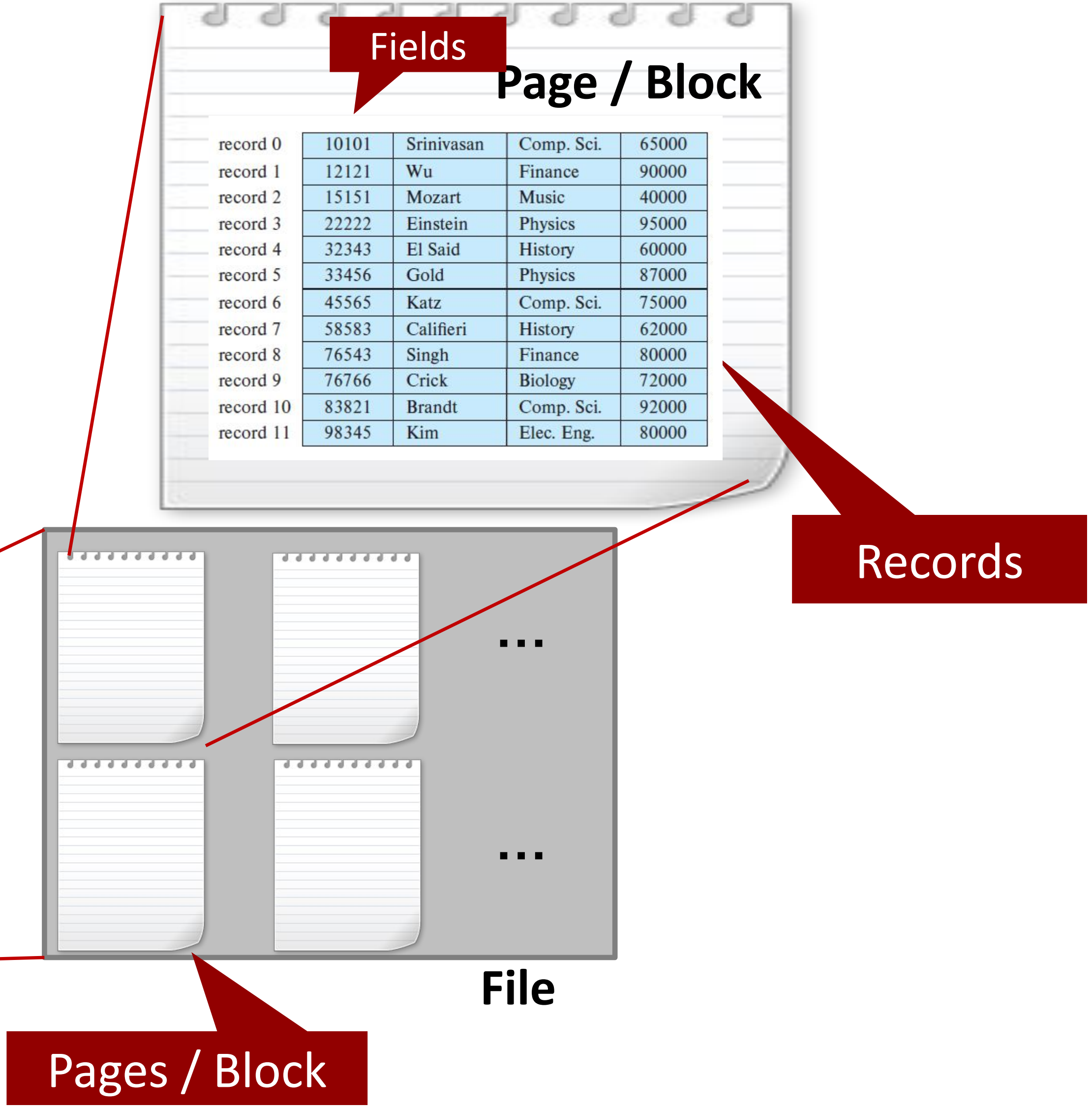
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Each file is made of pages

Each pages is made of *records*

A record is a sequence of fields



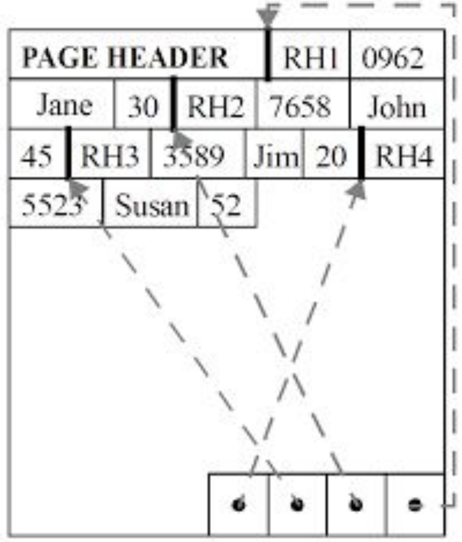


Page format (N-ary storage model)

- Nowadays: Row store
- Page = collection of slots

RH1	0962	Jane	30
RH2	7658	John	45
RH3	3589	Jim	20
RH4	5523	Susan	52

- Each slot stores one record
  - Record identifier: <page_id, slot_number>
  - Option 2: <uniq> -> <page_id, slot_number>



- Page format should support
  - Fast searching, insertion, deletion
- *Page format* depends on record format
  - **Fixed-Length**
  - **Variable-Length**

Record formats: fixed length

```
type instructor = record
        ID varchar (5);
        name varchar(20);
        dept_name varchar (20);
        salary numeric (8,2);
    end
```

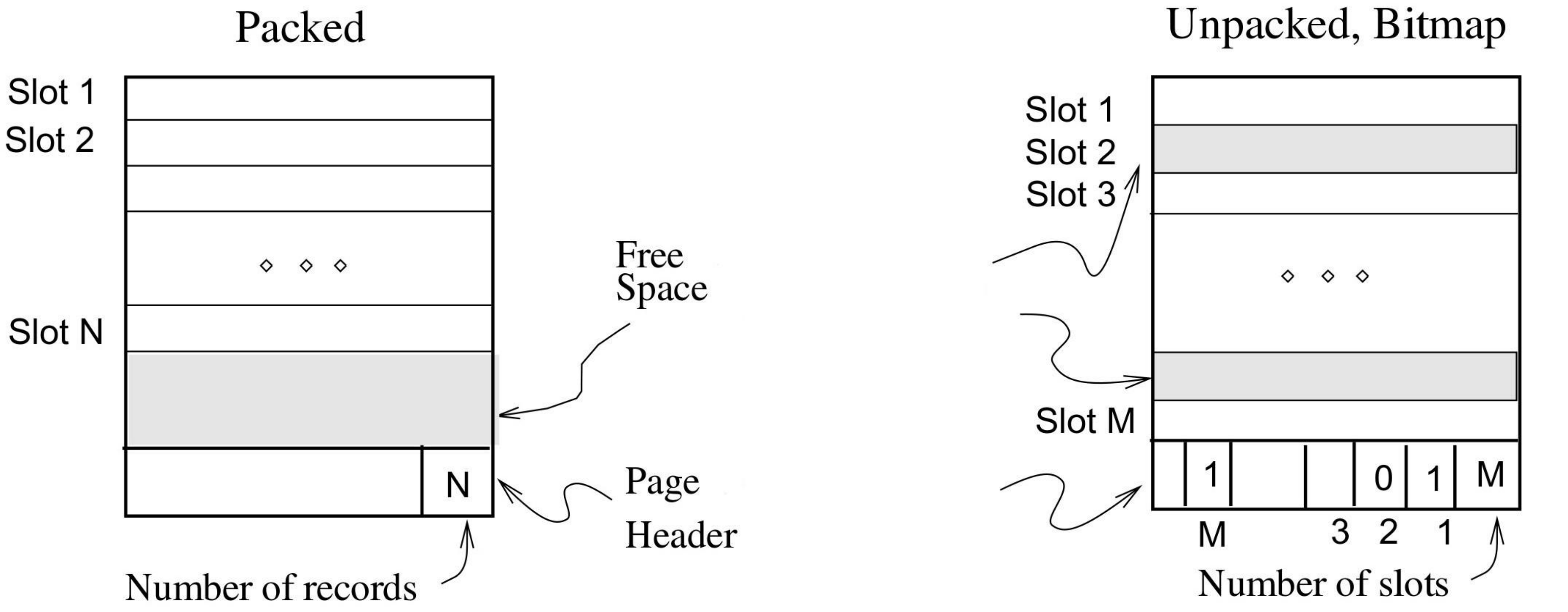
- Schema is stored in ***system catalog***
  - Number of fields is fixed for all records of a table
  - Domain is fixed for all records of a table

- Each field has fixed length
- Finding i^{th} field is done via arithmetic

Simple: 53 bytes each!

	5 bytes	20 bytes each (eventual padding)		8 bytes
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Page format: fixed-length records



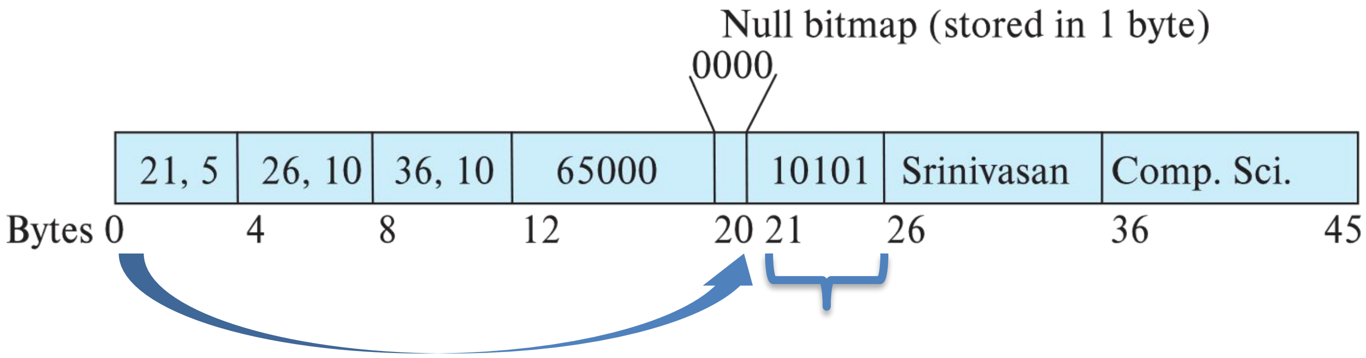
- <u>Record id</u> = *<page id, slot #>*
- In the *packed* case, moving records for free space management changes *rid* (maybe unacceptable)

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)

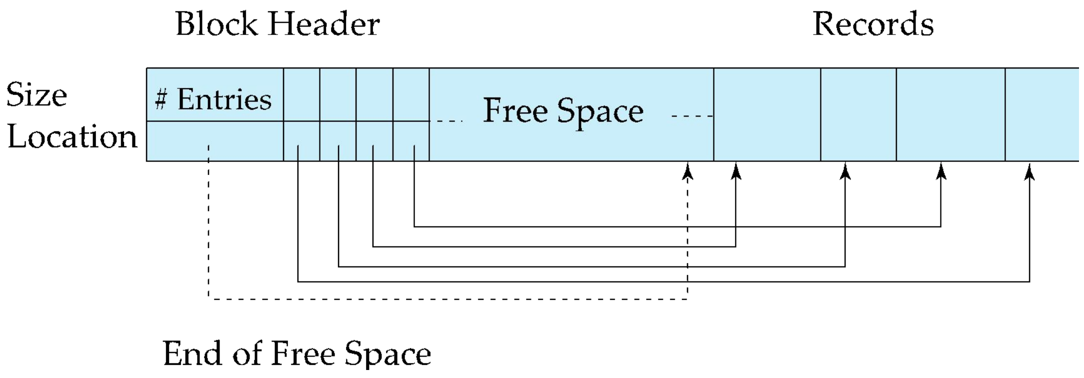
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000



Variable-Length Records: Slotted Page Structure

- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them



Variable-length records: Issues

- If a field grows and no longer fits?
  - Shift all subsequent fields

- If record no longer fits in page?
  - Move a record to another page after modification

- What if record size > page size?
  - SQL Server record size = 8KB
  - DB2 record size = page size

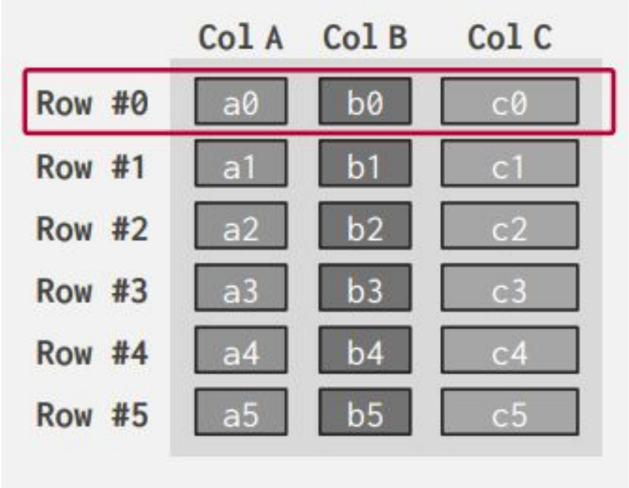
Row store (N-ary storage model): Summary

Advantages

- Fast inserts, updates, and deletes.
- Good for queries that need the entire tuple (OLTP)
- Can use index-oriented physical storage for clustering.

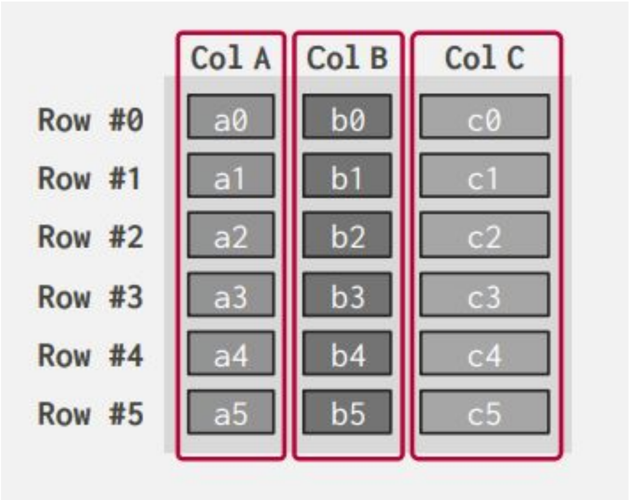
Disadvantages

- Not good for scanning large portions of the table and/or a subset of the attributes
- Not ideal for compression because of multiple value domains within a single page



What about storing an entire column contiguously in a block of data?

Decomposition Storage Model (DSM)



Columnar Representation

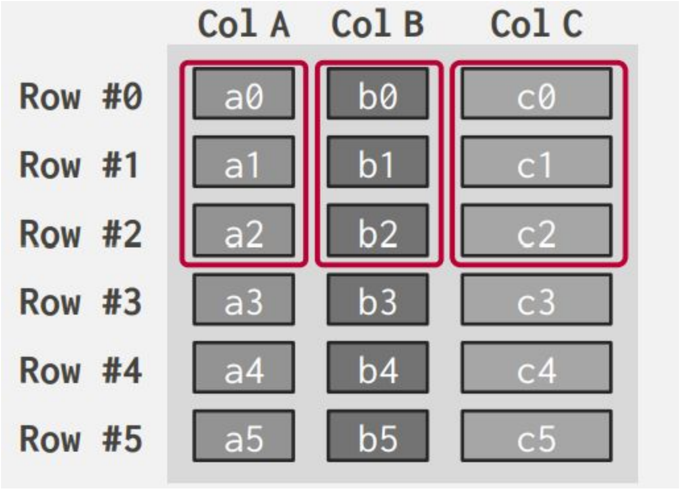
- Benefits:
  - Reduced IO if only some attributes are accessed
  - Improved CPU cache performance
  - Improved compression
  - **Vector processing** on modern CPU architectures
- Drawbacks
  - Cost of tuple reconstruction from columnar representation
  - Cost of tuple deletion and update
  - Cost of decompression
- Columnar representation found to be more efficient for decision support than row-oriented representation
- Traditional row-oriented representation preferable for transaction processing
- Some databases support both representations
  - Called **hybrid row/column stores**

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Partition Attributes Across (PAX)

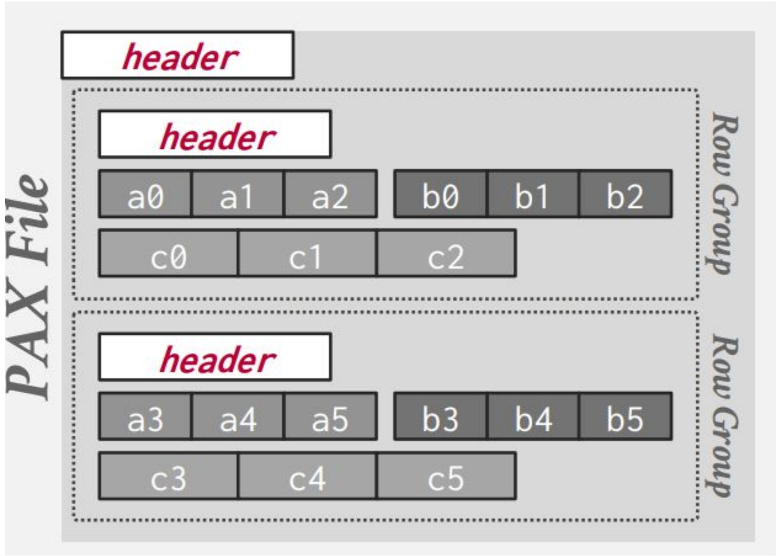
Horizontally partition rows into groups

Then vertically partition their attributes into columns



Global header contains directory with the offsets to the file's row groups

Each row group contains its own meta-data header about its contents



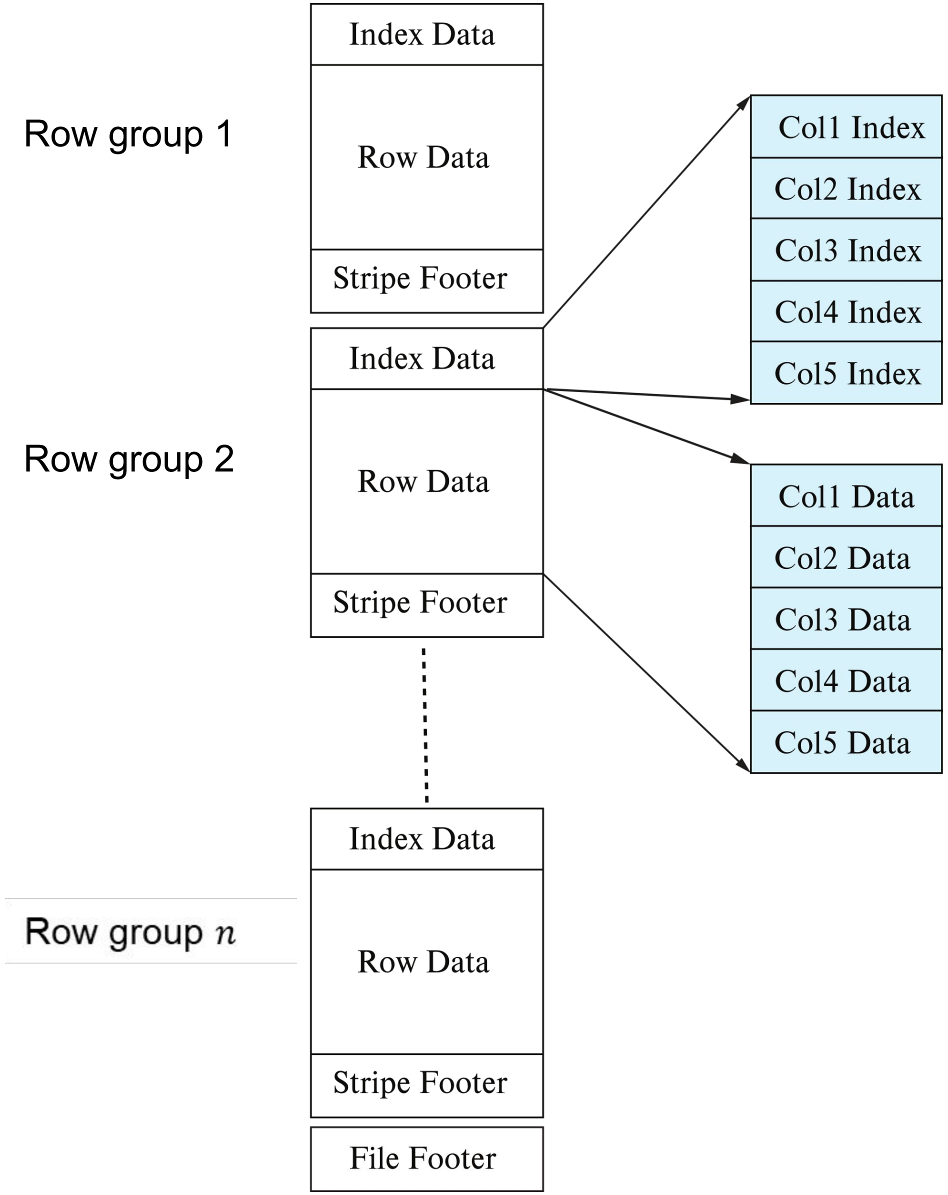
VLDB 2001 paper by Ailamaki et al.

First adopted by Oracle 9i

Now used at Google, Snowflake, Microsoft, etc.

File Representation based on PAX

- Optimized Row Columnar (ORC) and Parquet: file formats with columnar storage inside file
- Very popular for big-data applications
- ORC file format shown on right:



File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

Alternative file organizations

Many alternatives exist, *each good for some situations, and not so good in others:*

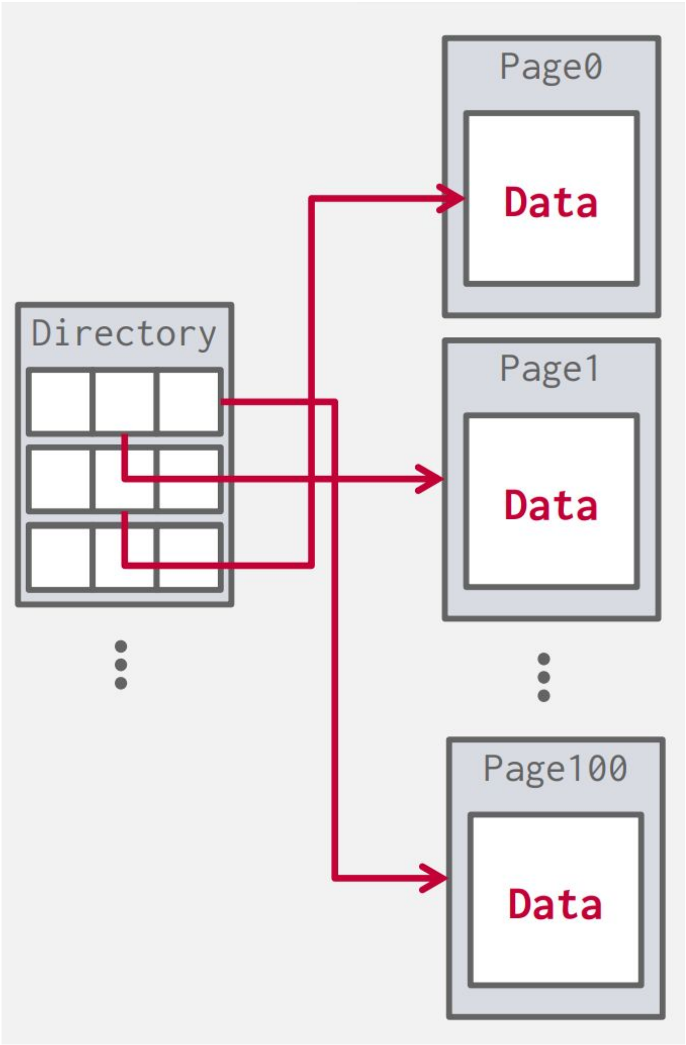
- Heap files: Suitable when typical access is a file scan retrieving all records
- Sorted (Sequential) Files: Best for retrieval in some order, or for retrieving a *range* of records
- Index File Organizations: (will cover shortly..)

Heap (unordered) File Organization

- Simplest file structure
  - Contains records in no particular order
  - Need to be able to scan, search based on rid

- The DBMS can locate a page on disk given by using
  1. Linked List: Header page holds pointers to a list of free pages and a list of data pages
  2. Page Directory: DBMS maintains special pages that track locations of data pages along with the amount of free space on each page

- As file grows and shrinks, disk pages are allocated and de-allocated
  - Need to manage free space



Sorted (Sequential) File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Heap file vs. sorted file

- Which is better?
  - Let us design a cost model to find out

- Simplified cost model:
  - Based only on IO cost
    - IO is the dominating cost
    - Ignore CPU and other overheads
    - Ignore effect of prefetching and sequential access
  - Consider only average case

More assumptions…

- Single record insert and delete
- Equality search: exactly one match (e.g., search on key)
  - Question: what if more or fewer?

- Heap Files:
  - Insert always appends at the end of the file

- Sorted Files:
  - Files compacted after deletions
  - Search done on file-ordering attribute

Cost of operations (in # of I/O’s)

B: Number of data pages

	Heap File	Sorted File	notes…
Scan all records	B	B	
Equality Search	0.5B	$\log_2$ B	assumes exactly one match!
Range Search	B	($\log_2$ B) + (#match pages)	
Insert	2	($\log_2$B) + 2*(B/2)	must R & W
Delete	0.5B + 1	($\log_2$B) + 2*(B/2)	must R & W

File indexing and organization

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

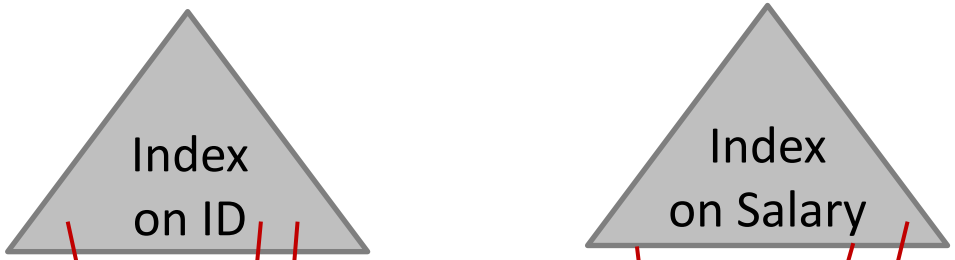
Indexing

- Instructor with ID = 22222?
- Instructor with ID = 33456?
- Instructor with ID = 83821?

- Updates?

- Salary >= 90K?

- Indexing
  - Multiple efficient access paths



instructor

	ID	*name*	*dept_name*	*salary*
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Sorted File on ID

Indexing

- Sometimes, we want to retrieve records by specifying the *values in one or more fields*, e.g.,
  - Find all students in the "CS" department
  - Find all students with a gpa > 3

- An **index** on a file speeds up selections on the *search key fields* for the index
  - The *search key* is NOT necessarily a *key* (e.g., no need to be unique)

- Index and keys
  - Any subset of the fields in a relation can be the search key for an index on the relation
  - Can have multiple indices on any number of fields

Index search conditions

- **Search condition =**

<search key, comparison operator>

Examples…

(1) Condition: Department = "CS"

- Search key: Department
- Comparison operator: equality (=)

(2) Condition: GPA > 3

- Search key: GPA
- Comparison operator: greater-than (>)

File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

- Index Classification
  - Clustered/Unclustered
  - Sparse/Dense
- Types of Indexes
  - Primary
  - Clustering
  - Secondary Key
  - Secondary Non-Key
- Indexing techniques
  - Hash vs tree
- Choosing search key

Index classification

- Clustered vs Unclustered
- Dense vs Sparse
- Indexing field
  - Key
  - Non-Key
- Physical ordering of the file
  - Ordered on indexing field
  - Not ordered on indexing field

2

2

2 X 2 = 4

		Physical Ordering on Indexing Field	
		Ordered	Not Ordered
Indexing Field	Key	Primary Index	Secondary Index (Key)
	Non Key	Clustering Index	Secondary Index (Non Key)

Some of the material for this topic is taken from "Database Systems", Elmasri, Navathe.

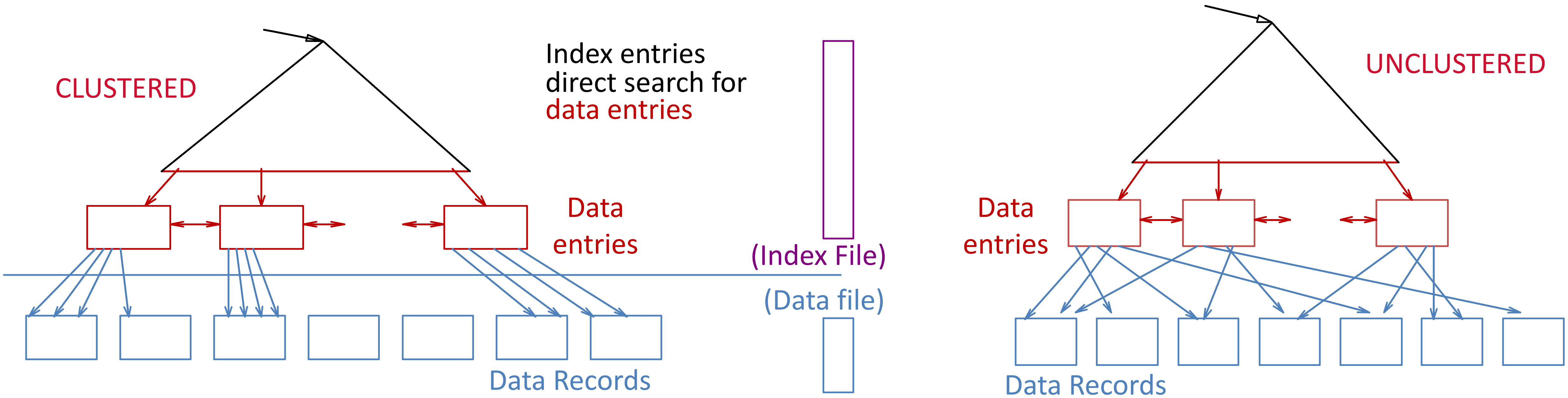
File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

- Index Classification
  - Clustered/Unclustered
  - Sparse/Dense
- Types of Indexes
  - Primary
  - Clustering
  - Secondary Key
  - Secondary Non-Key
- Indexing techniques
  - Hash vs tree
- Choosing search key

Index classification: Clustering

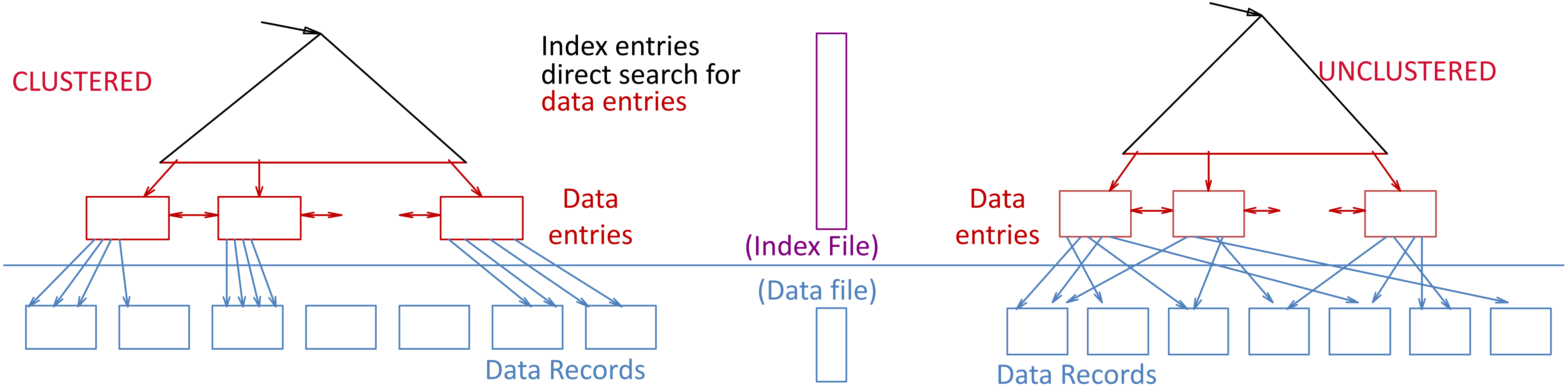
- Clustered vs. unclustered: If the order of the data records is similar to the order of the index data entries, then the index is ***clustered***



Clustered vs unclustered index

Assuming Alternative 2 for data entries and data records stored in a Heap file:

- To build clustered index, first sort the Heap file
  - Keeping some free space on each page for future inserts
- Overflow pages may be needed for inserts
  - The order of data records is similar—but not identical to—the sort order



Clustered vs unclustered index

- Cost of retrieving records found in range scan:
  - Clustered: cost = # pages in file w/matching records
  - Unclustered: cost ≈ # of matching index <u>data entries</u>

- What are the tradeoffs?
  - Clustered Pros:
    - Efficient for range searches
  - Clustered Cons:
    - Expensive to maintain (on the fly or sloppy with reorganization)

Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation

10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Dense Index Files (Cont.)

- Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

Sparse Index Files

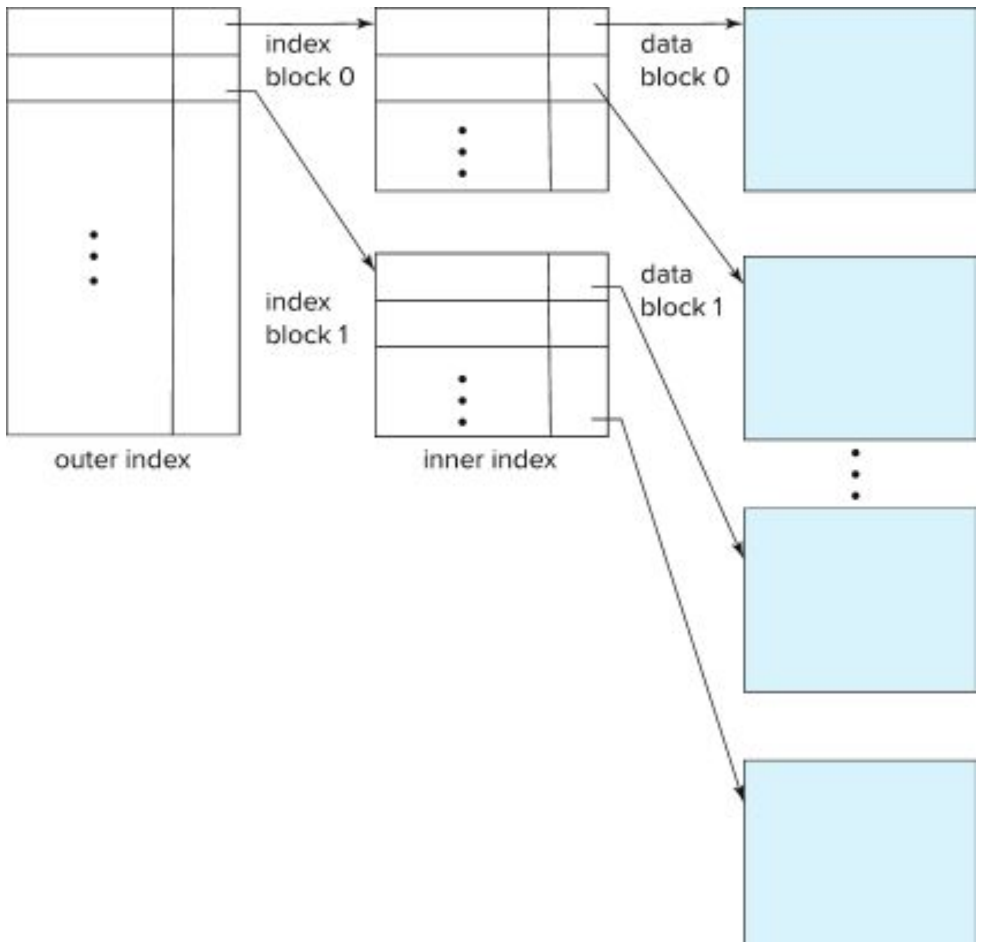
10101
32343
76766

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points

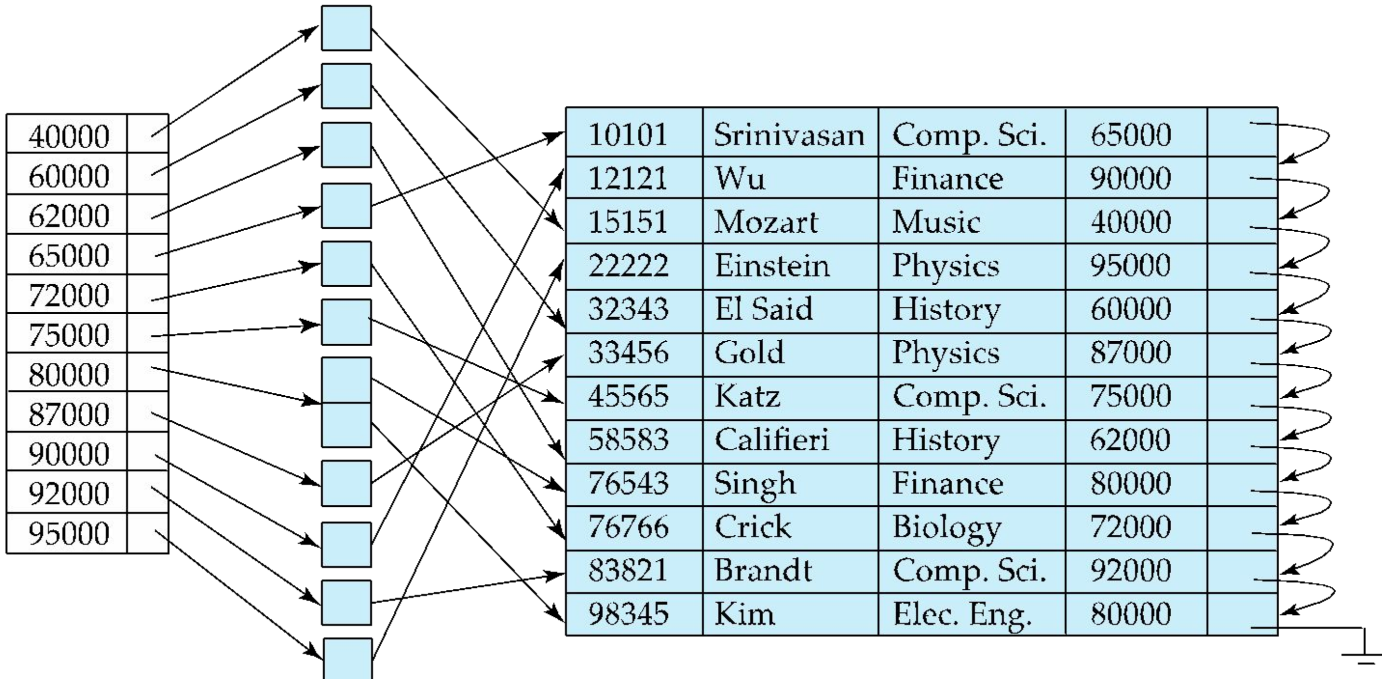
Multilevel Index

- If index does not fit in memory, access becomes expensive.
- Solution: treat index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of the basic index
  - inner index – the basic index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.



Secondary Index Example

- Secondary index on salary field of instructor
- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value.
- Secondary indices on unsorted columns have to be dense



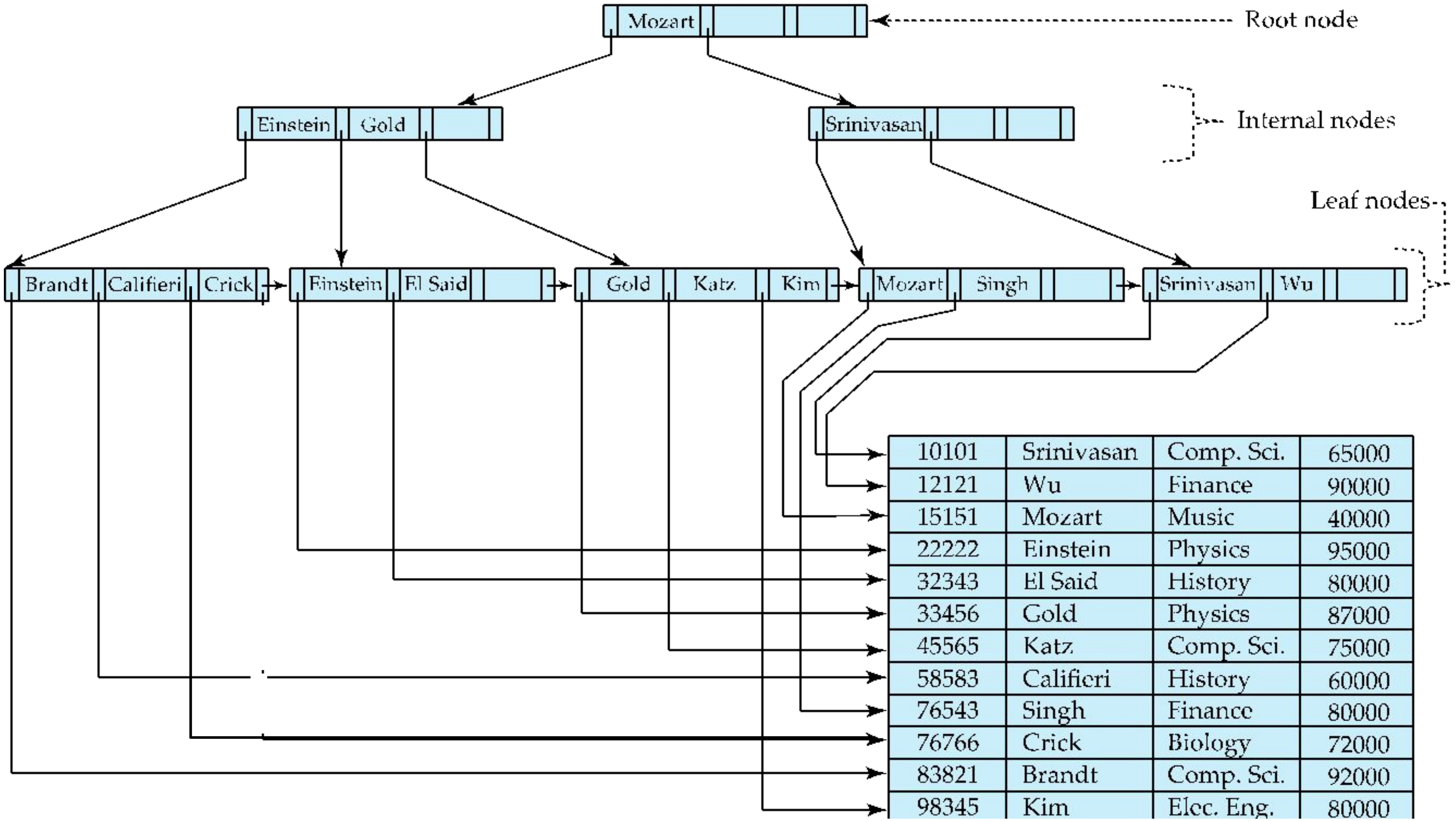
B$^+$-Tree Index Files

- B$^+$-tree is a rooted tree satisfying the following properties:
  - All paths from root to leaf are of the same length
  - Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
  - A leaf node has between $\lceil (n–1)/2 \rceil$ and n–1 values
  - Special cases:
    - If the root is not a leaf, it has at least 2 children.
    - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values

  - General structure

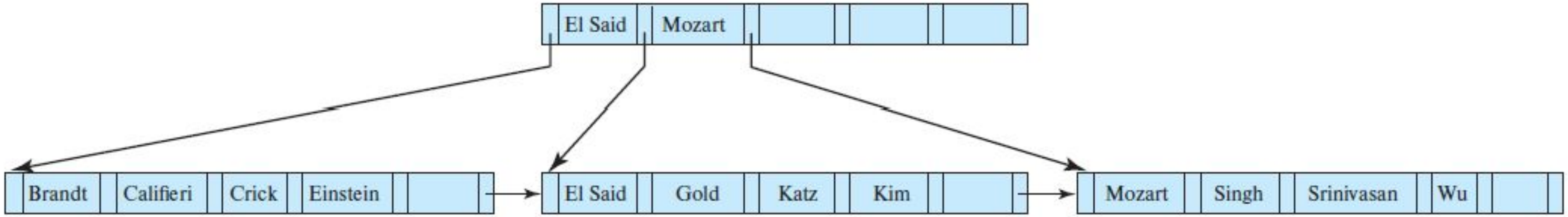
P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

Example of a B$^+$-Tree (n = 4)



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Example of a B$^+$-Tree ($n = 6$)



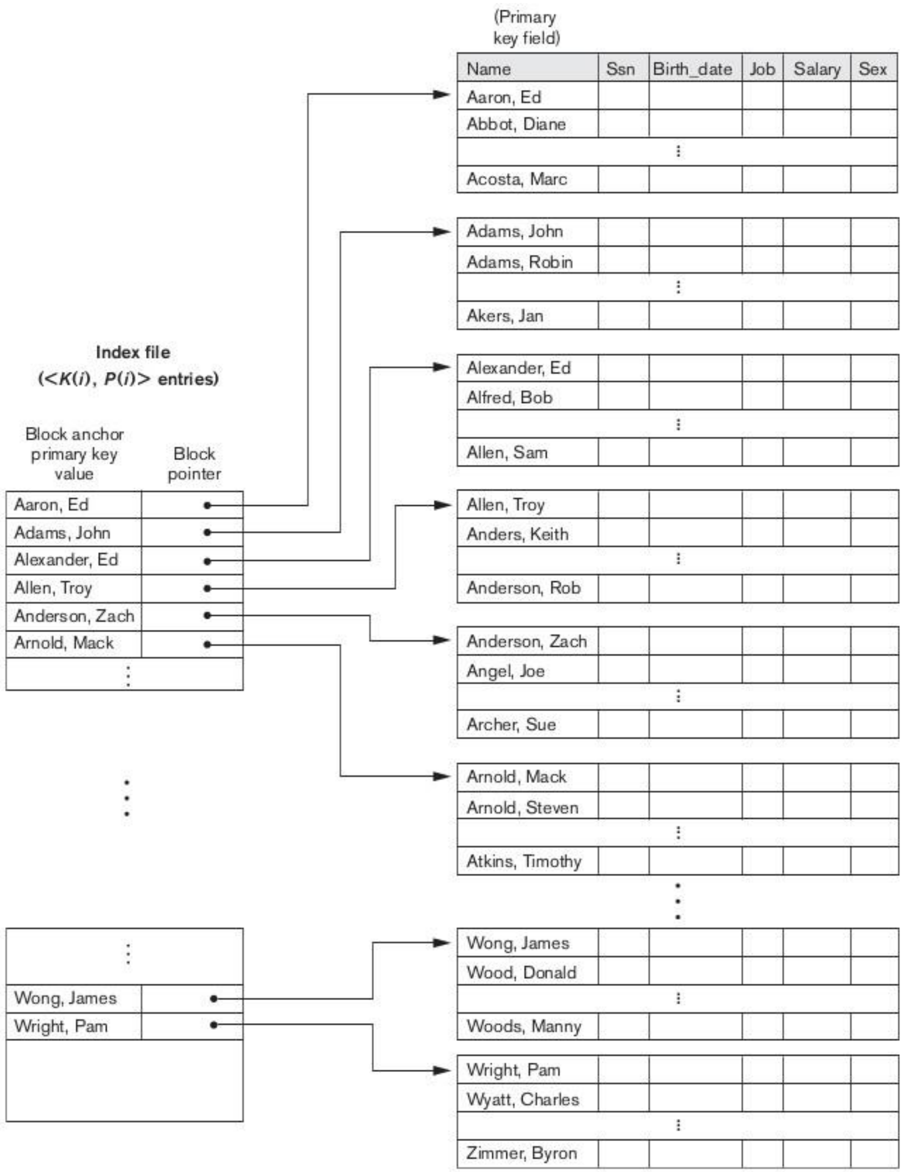
File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

- Index Classification
  - Clustered/Unclustered
  - Sparse/Dense
- Types of Indexes
  - Primary
  - Clustering
  - Secondary Key
  - Secondary Non-Key
- Indexing techniques
  - Hash vs tree
- Choosing search key

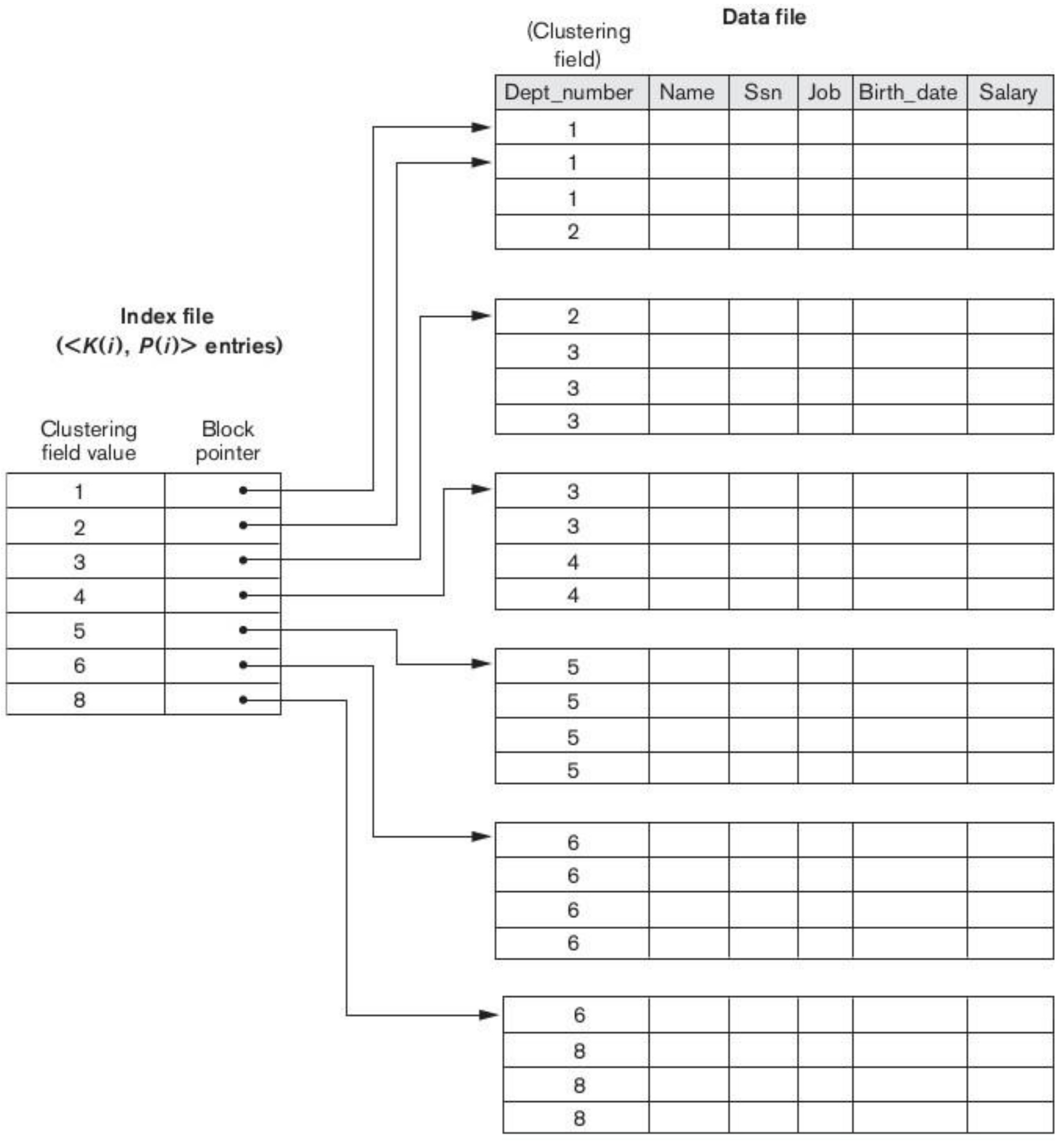
Primary index

- Indexing Field = Key
- File is physically sorted on indexing field
- One index entry *per block*
- Index pointers can be block pointers (anchors)
- Sparse Index



Clustered index

- Indexing Field = Non-Key
- File is physically sorted on indexing field
- One index entry *per distinct value*
- Index pointer is block pointer to first block with the value
- Sparse Index



Index classification: Summary

Type of Index	Indexing Field	File physically sorted on indexing field?	Index Entries	Index Pointers	Sparse or Dense?
Primary	Key	Yes	One per block	Block anchor	Sparse
Clustering	Non-Key	Yes	One per value	Block pointer	Sparse
Secondary Key	Key	No	One per record	Record pointer	Dense
Secondary Non-Key	Non-Key	No	One per record/ value	Record pointer/ Variable length/ indirection	Dense

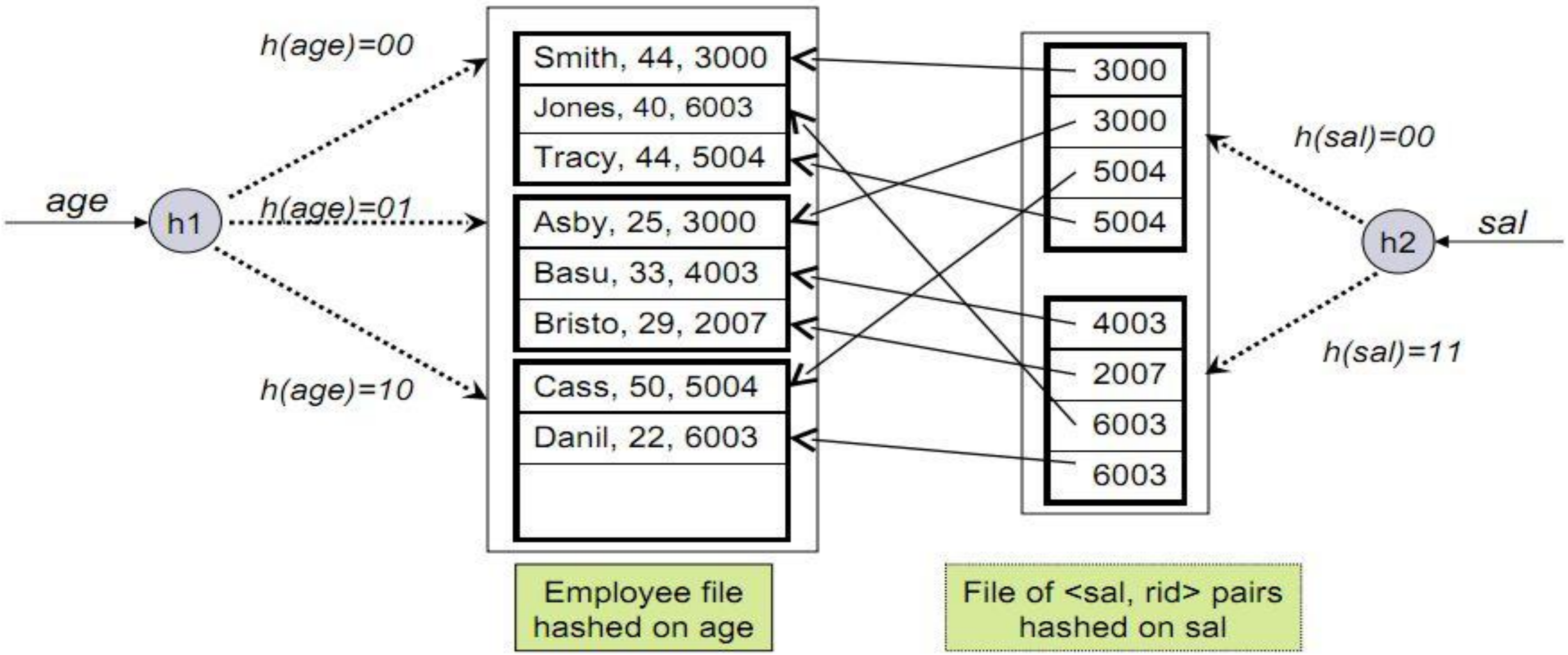
File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

- Index Classification
  - Clustered/Unclustered
  - Sparse/Dense
- Types of Indexes
  - Primary
  - Clustering
  - Secondary Key
  - Secondary Non-Key
- Indexing techniques
  - Hash vs tree
- Choosing search key

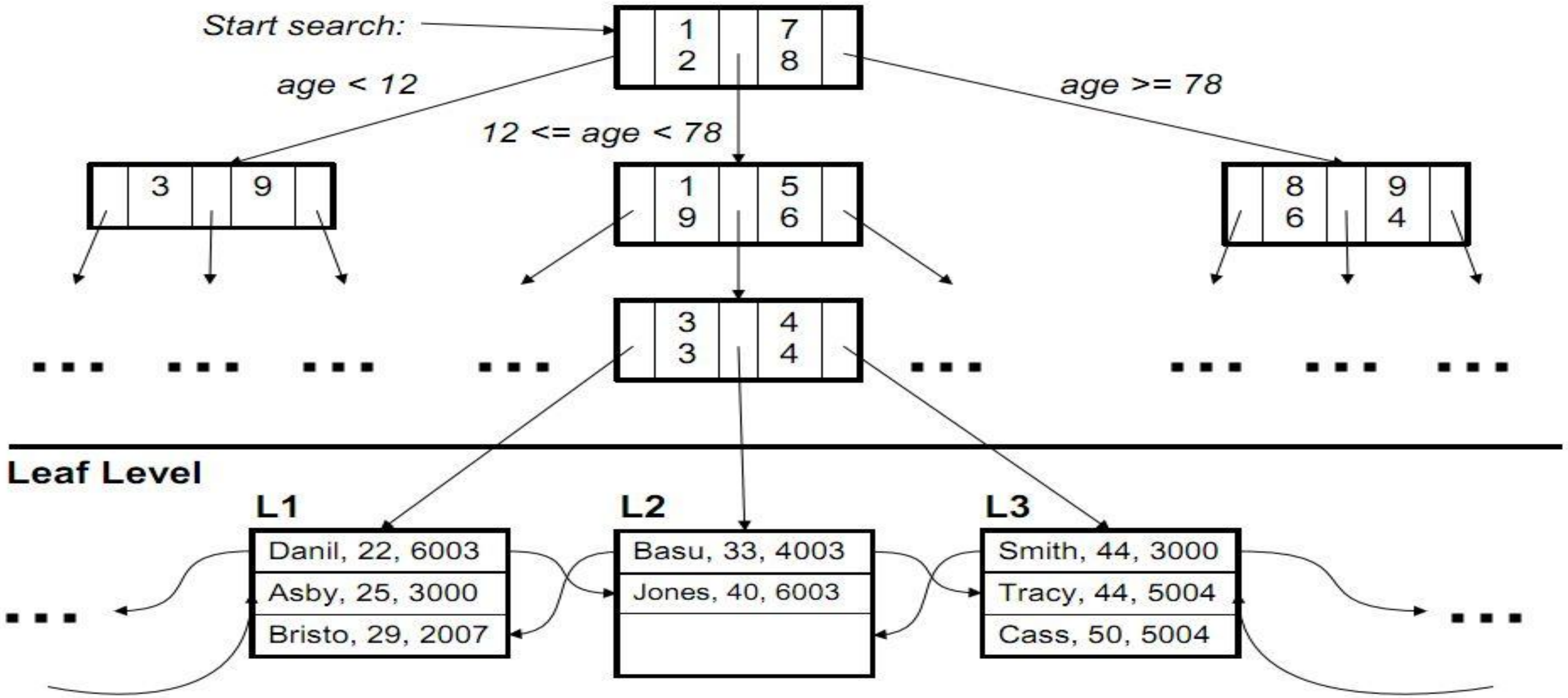
Hash-based index

- Good for **equality** selections
  - File = a collection of *buckets*
    - Bucket = *primary* page plus 0 or more *overflow* pages
  - *Hash function* **h**: **h**(*r.search_key*) = bucket for record *r*



Tree-based index

- Good for **range** selections
  - Leaves contain data entries sorted by search key value
  - B+ tree: all root->leaf paths have equal length *(height)*



Indexing decisions are driven by queries

```
Select E.dno
From Employees E
Where E.age > 40
```

- Would you build a b+tree index or hash index?
  - Hint: it’s not an equality query

- Would you build a clustered index?

- When would a B+tree be suboptimal?
  - Hint: think about selectivity

File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

- Index Classification
  - Clustered/Unclustered
  - Sparse/Dense
- Types of Indexes
  - Primary
  - Clustering
  - Secondary Key
  - Secondary Non-Key
- Indexing techniques
  - Hash vs tree
- Choosing search key

Choosing a search key

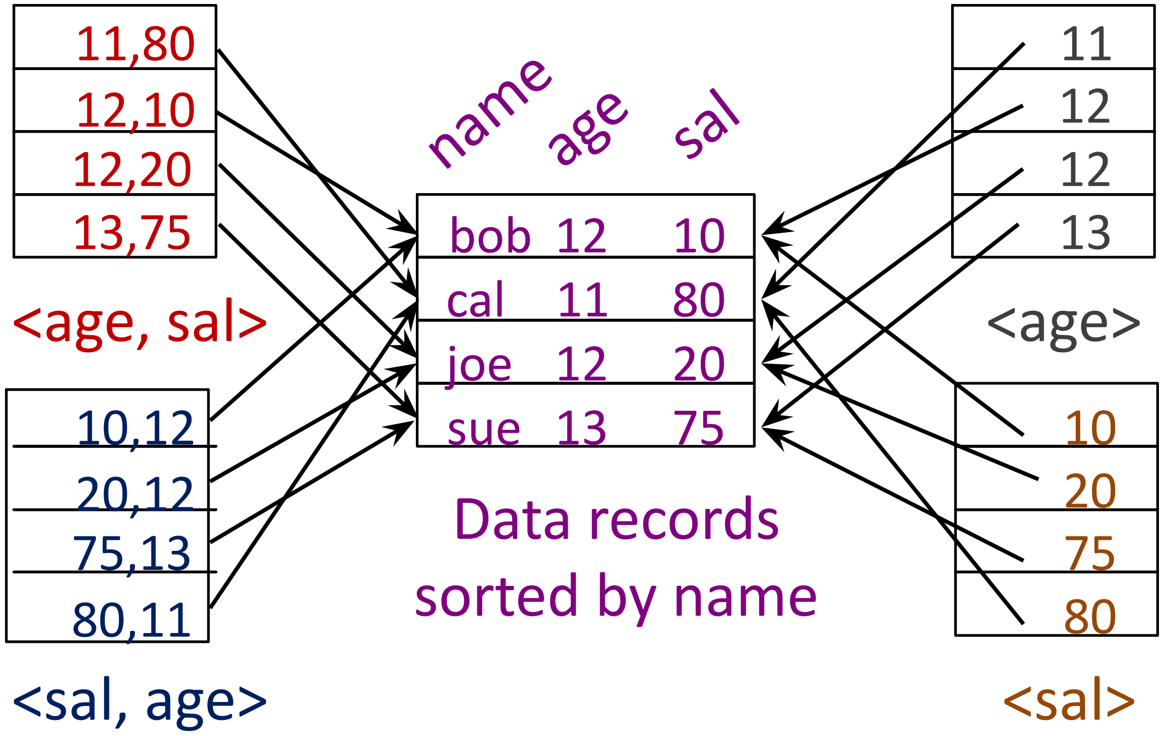
```
Select sID
From Student
Where sName = 'Mary' And GPA > 3.9
```

- Can build an index on sname
  - What index would this be? Hash or tree?
- Can build index on GPA
  - What index would this be? Hash or tree?
- How about <sname, gpa> together?

Composite search key

- Search on field *combination*.
  - Equality query: Every field value is equal to a constant. E.g. wrt <sal,age> index:
    - age=12 and sal =75
  - Range query: Some field value is not a constant. E.g.:
    - age =12; or age=12 and sal > 20
- Data entries in index sorted by search key for range queries
  - Lexicographic order.

Examples of composite key indexes using lexicographic order



Composite search key: Tradeoffs

```
Select AVG(E.sal)
From   Employees E
Where  E.age = 25
     AND E.sal BETWEEN 3000 AND 5000
```

- Do we build index on <age,sal>, <sal,age>?
- What index would this be? Hash or tree?
- Do we really need data file?
  - “index-only evaluation” is possible

File organization and indexing

- File Organization
- Heap & Sorted Files
- Indexing
- Meta-data
  - System Catalog

System catalogs

- For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each view:
  - view name and definition
- Plus stats, authorization, buffer pool size, etc.

Catalogs are themselves stored as relations!

System catalog in Oracle

```
> desc user_tables
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TABLE_NAME                                NOT NULL VARCHAR2(128)
 TABLESPACE_NAME                                    VARCHAR2(30)
 CLUSTER_NAME                                       VARCHAR2(128)
 IOT_NAME                                           VARCHAR2(128)
 STATUS                                             VARCHAR2(8)
 PCT_FREE                                           NUMBER
 PCT_USED                                           NUMBER
 ...

> desc user_tab_columns
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 TABLE_NAME                                NOT NULL VARCHAR2(128)
 COLUMN_NAME                               NOT NULL VARCHAR2(128)
 DATA_TYPE                                          VARCHAR2(128)
 DATA_TYPE_MOD                                      VARCHAR2(3)
 DATA_TYPE_OWNER                                    VARCHAR2(128)
 DATA_LENGTH                               NOT NULL NUMBER
 DATA_PRECISION                                     NUMBER
 DATA_SCALE                                         NUMBER
 NULLABLE                                           VARCHAR2(1)
 ...
```

System catalog in Oracle

```
> select * from student;

SID        NAME                        AGE
---------- -------------------- ----------
A1234      John                         23
B1         Xavier                       24
A21341     Scott                        21
C12948291  Benjamin                     25
1948       Ben                          29

> select table_name, num_rows, blocks, avg_row_len from user_tables;

TABLE_NAME             NUM_ROWS     BLOCKS AVG_ROW_LEN
-------------------- ---------- ---------- -----------
STUDENT                       5          5          20

> select table_name, column_name, data_type, data_length from user_tab_columns;

TABLE_NAME COLUMN_NAME DATA_TYPE  DATA_LENGTH
---------- ----------- ---------- -----------
STUDENT    SID         CHAR                10
STUDENT    NAME        VARCHAR2            32
STUDENT    AGE         NUMBER              22
```

Summary

- Database organized as a collection of files
  - Several file organizations (heap, sorted, …) with tradeoffs
- Files are a collection of pages
  - Several page layouts (NSM, DSM, …) with tradeoffs
- Pages contain a collection of records
  - Several record formats (fixed, variable length…) with tradeoffs
- Index is a quick way to find records
  - Several index types with tradeoffs

One size does not fit all!