THE UNIVERSITY OF MELBOURNE

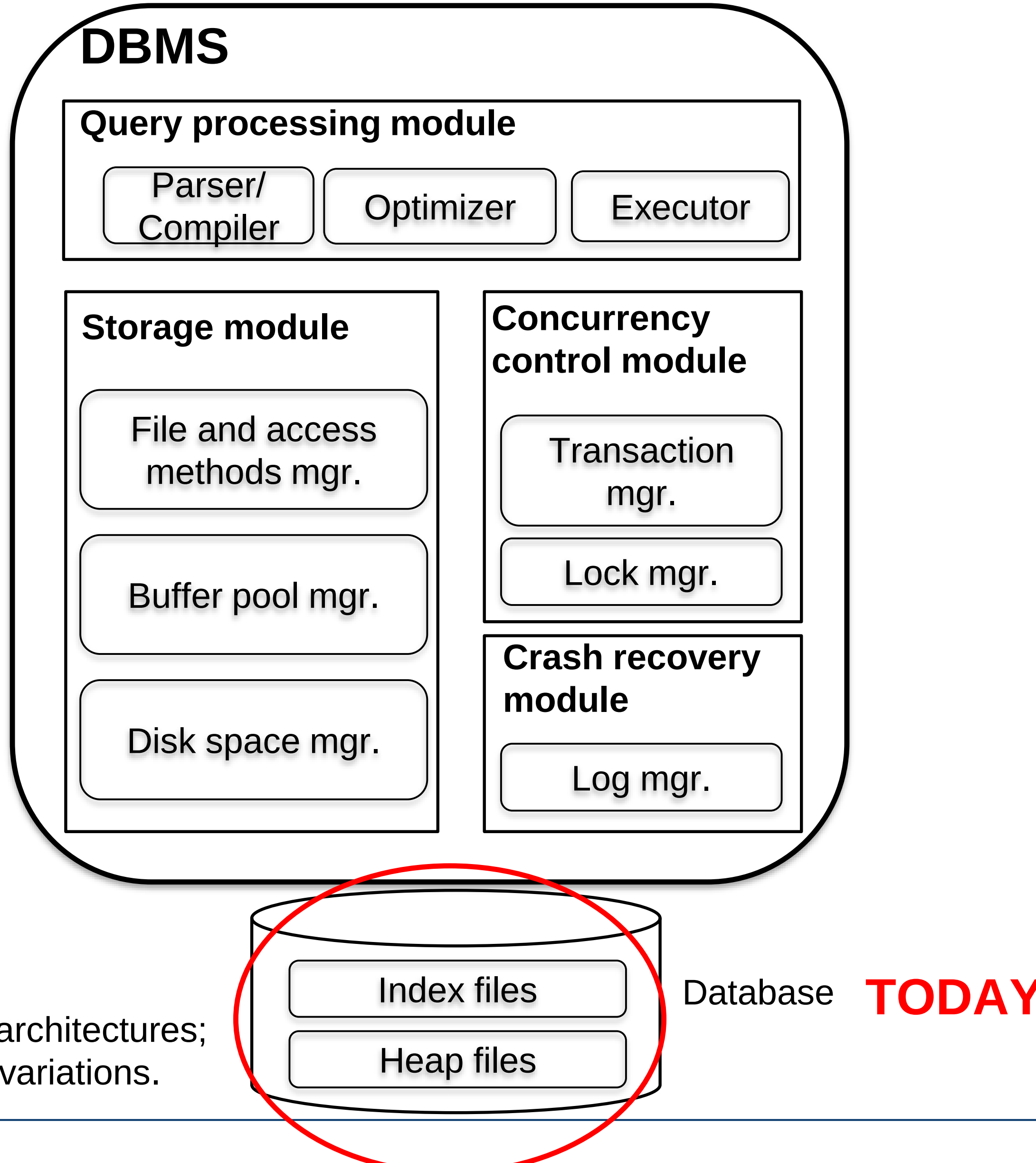
INFO20003 Database Systems

Dr Renata Borovica-Gajic

Lecture 10

Storage and Indexing

Remember this? Components of a DBMS



This is one of several possible architectures; each system has its own slight variations.

Coverage

- File organization (Heap & sorted files)
- Index files & indexes
- Index classification

Readings: Chapter 8, Ramakrishnan & Gehrke, Database Systems

Files

- FILE: A collection of pages, each containing a collection of records.
- Must support:
  - insert/delete/modify record
  - read a particular record (specified using *record id*)
  - scan all records (possibly with some conditions on the records to be retrieved)

Alternative File Organizations

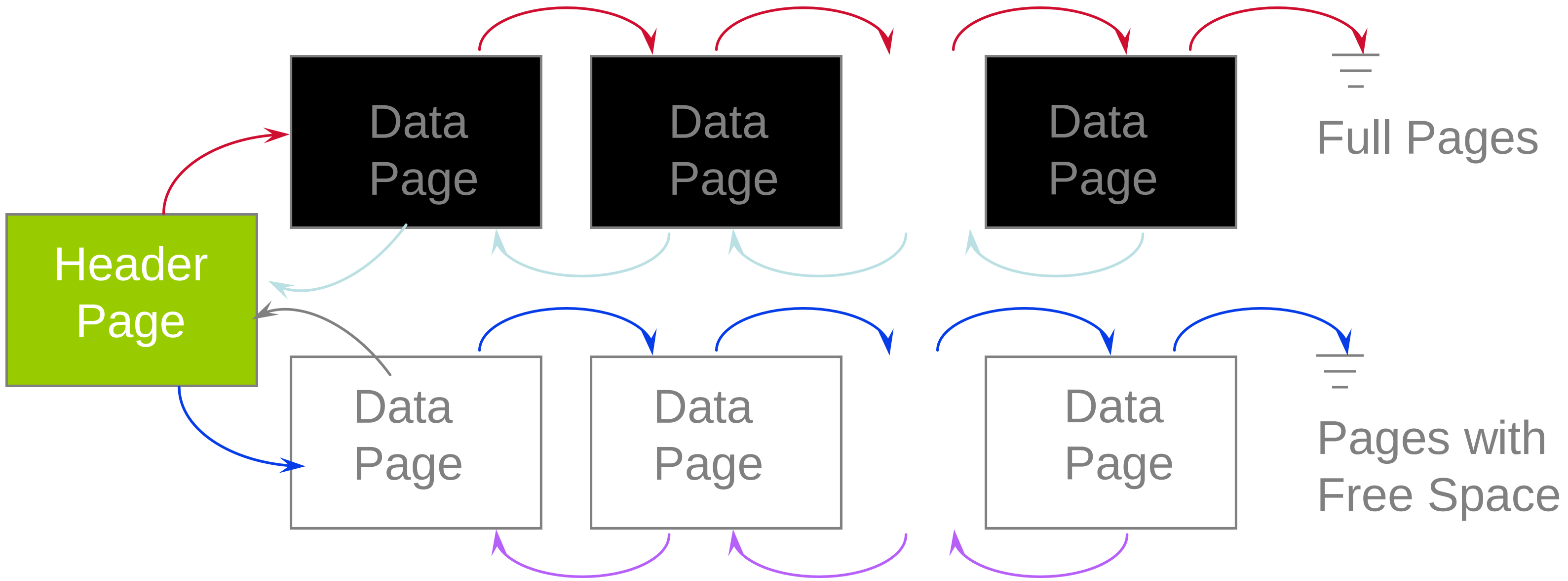
Many alternatives exist, *each good for some situations, and not so good in others:*

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

Heap (Unordered) Files

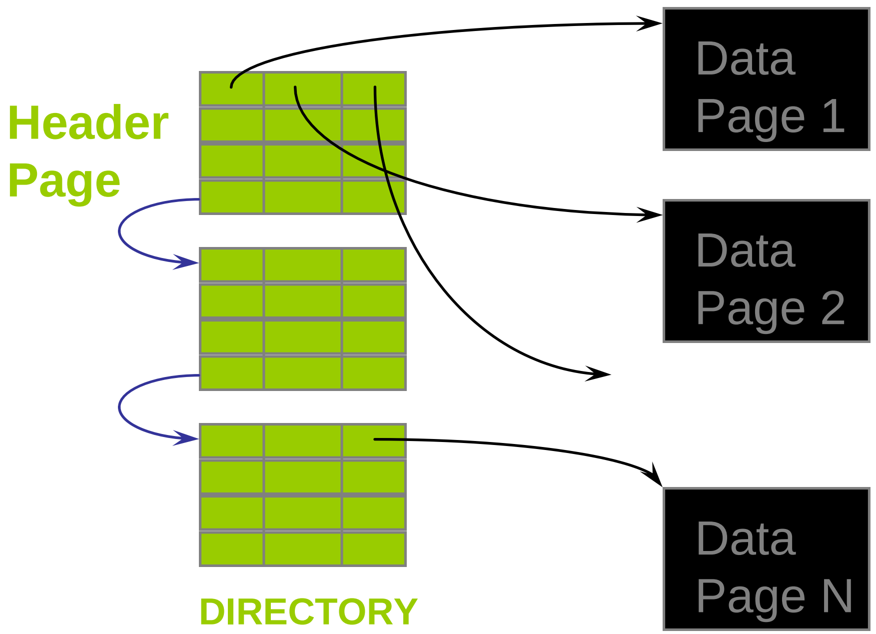
- Simplest file structure
  - contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.

Heap File Implemented Using Lists



- The header page id and Heap file name must be stored someplace.
- Each page contains 2 'pointers' plus data.

Heap File Using a Page Directory



- The entry for a page can include the number of free bytes on the page.
- The directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages*!

Heap files vs. Sorted files

- **Quick (imprecise) cost model: # of disk I/O's**
  - For simplicity, ignore:
    - CPU costs
    - Gains from pre-fetching and sequential access
  - Average-case analysis; based on several simplistic assumptions.

** Good enough to show the overall trends!*

Some Assumptions in the Analysis

- 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 to end of 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*

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!*

Attribute catalog

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

File Organization & Indexing

- File organization (Heap & sorted files)

- Index files & indexes

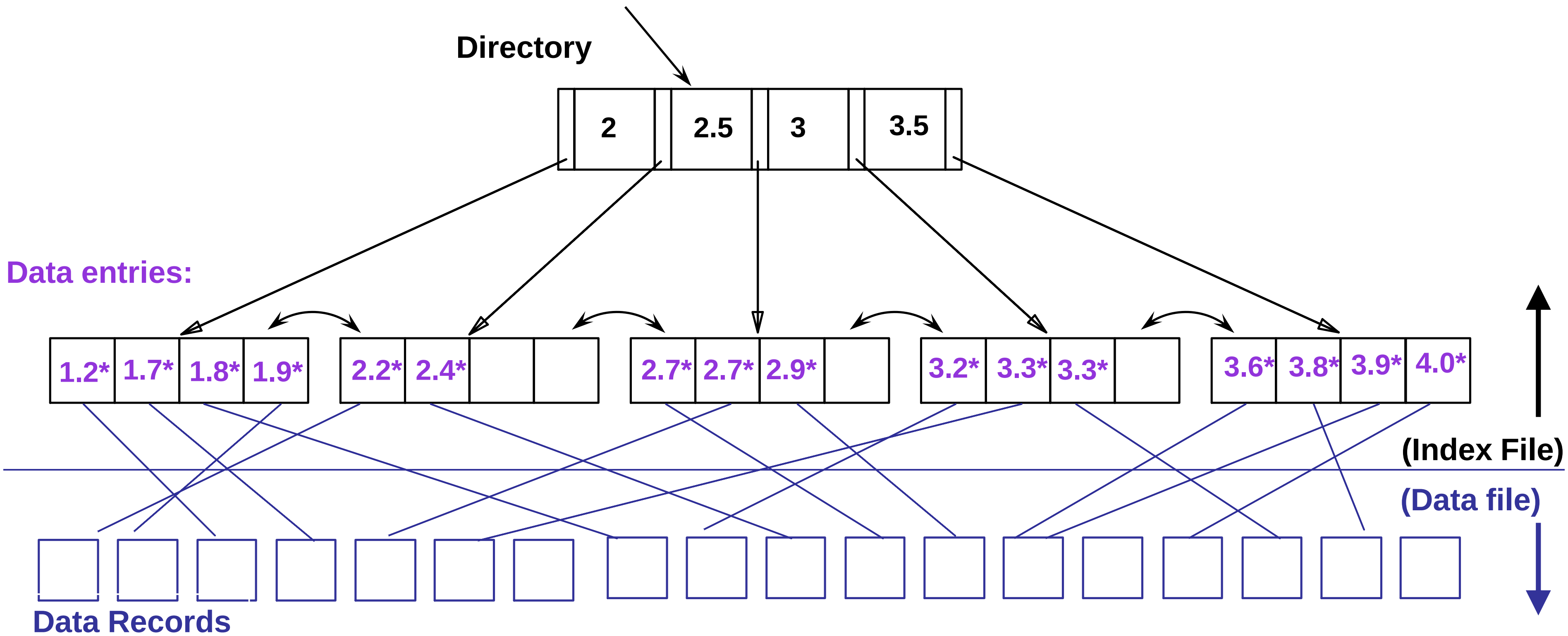
- Index classification

Indexes

- 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.
  - –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* (e.g., doesn't have to be unique).

Example: Simple Index on GPA



An index contains a collection of **data entries**, and supports efficient retrieval of **records** matching a given **search condition**

Index Search Conditions

- Search condition =

  <search key, comparison operator>

Examples:

(1) Condition: Department = “CS”

- Search key: “CS”
- Comparison operator: equality (=)

(2) Condition: GPA > 3

- Search key: 3
- Comparison operator: greater-than (>)

Index Classification

- Representation of data entries in index
  - i.e., what is at the bottom of the index?
  - 3 alternatives here
- Clustered vs. Unclustered
- Primary vs. Secondary
- Dense vs. Sparse
- Single Key vs. Composite
- Indexing technique
  - Tree-based, hash-based, other

Alternatives for Data Entry **k*** in Index

1. Actual data record (with key value **k**)
2. <**k**, rid of matching data record>
3. <**k**, list of rids of matching data records>

- Choice is orthogonal to the indexing technique.
  - Examples of indexing techniques: B+ trees, hash-based structures, R trees, …
  - Typically, index contains auxiliary info that directs searches to the desired data entries
- Can have multiple (different) indexes per file.
  - E.g. file sorted on *age*, with a hash index on *name* and a B+tree index on *salary*.

Alternatives for Data Entries (Contd.)

Alternative 1:

Actual data record (with key value **k**)

- If this is used, index structure is a file organization for data records (like Heap files or sorted files).
- At most one index on a given collection of data records can use Alternative 1.
- This alternative saves pointer lookups but can be expensive to maintain with insertions and deletions.

Alternatives for Data Entries (Contd.)

Alternative 2

<**k**, rid of matching data record>

and Alternative 3

<**k**, list of rids of matching data records>

- Easier to maintain than Alternative 1.
- If more than one index is required on a given file, at most one index can use Alternative 1; rest must use Alternatives 2 or 3.
- Alternative 3 more compact than Alternative 2, but leads to *variable sized data entries* even if search keys are of fixed length.
- Even worse, for large rid lists the data entry would have to span multiple pages!

File Organization & Indexing

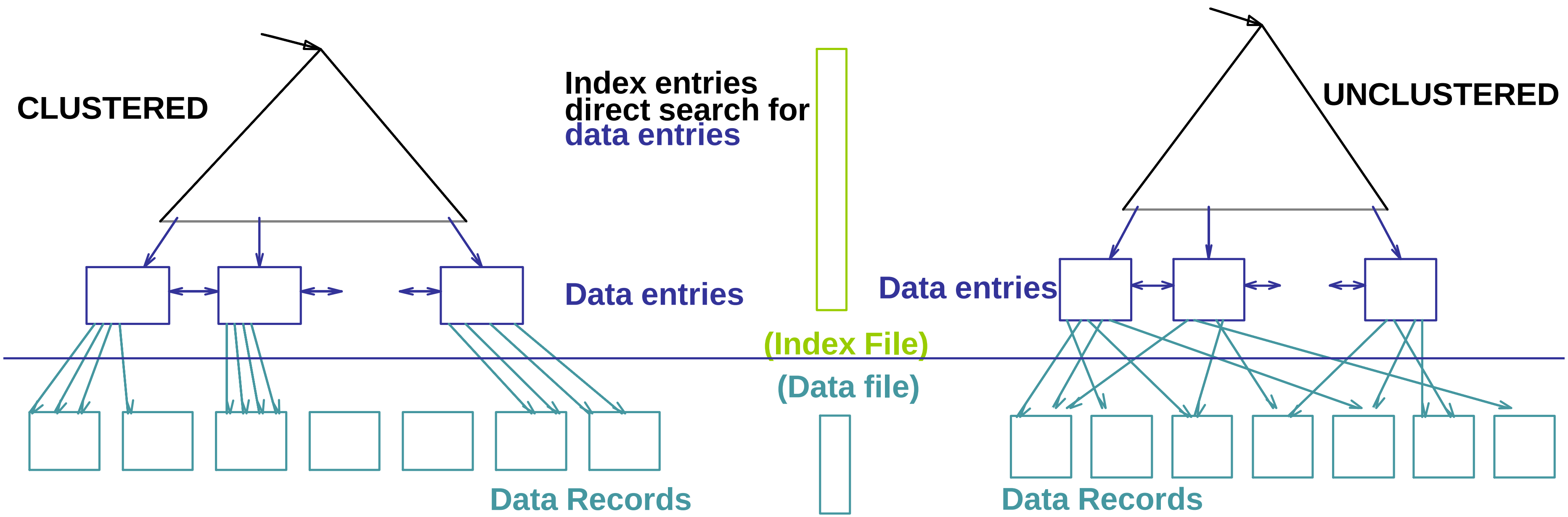
- File organization (Heap & sorted files)
- Index files & indexes
- Index classification

Index Classification

- Representation of data entries in index
  - –i.e., what is at the bottom of the index?
  - –3 alternatives here
- Clustered vs. Unclustered
- Primary vs. Secondary
- Dense vs. Sparse
- Single Key vs. Composite
- Indexing technique
  - –Tree-based, hash-based, other

Index Classification - clustering

- *Clustered vs. unclustered*: If order of data records is the same as, or `close to', order of index data entries, then called *clustered index*.

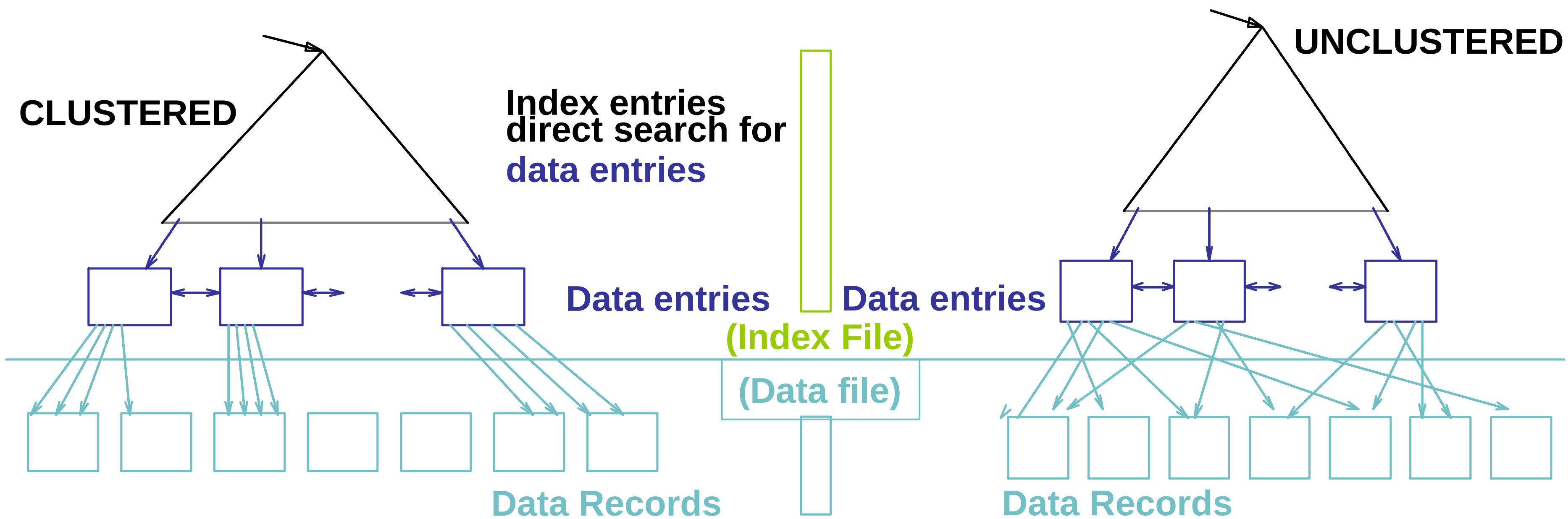


Index Classification - clustering

- A file can have a clustered index on at most one search key.
- Cost of retrieving data records through index varies *greatly* based on whether index is clustered!
- Note: Alternative 1 implies clustered, *but not vice-versa*.

Clustered vs. Unclustered Index

- Suppose that Alternative (2) is used for data entries, and that the data records are stored in a Heap file.
  - To build clustered index, first sort the Heap file (with some free space on each page for future inserts).
  - Overflow pages may be needed for inserts. (Thus, order of data recs is `close to', 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 data entries
- What are the tradeoffs?
  - Clustered Pros:
    - Efficient for range searches
    - May be able to do some types of compression
  - Clustered Cons:
    - Expensive to maintain (on the fly or sloppy with reorganization)

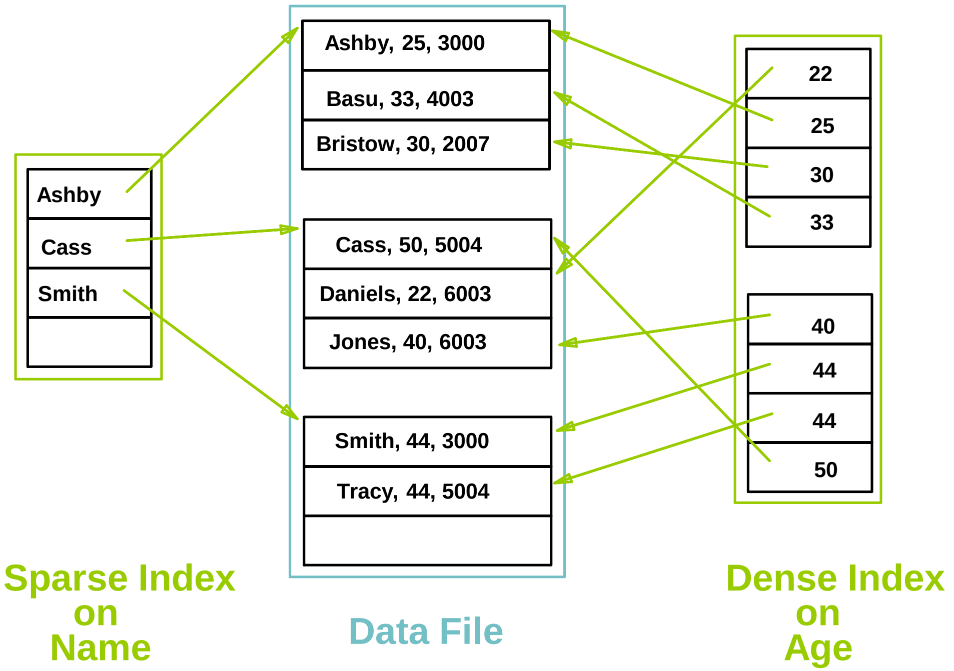
Primary vs. Secondary Index

- *Primary*: index key includes the file's primary key
- *Secondary*: any other index

- Sometimes confused with Alt. 1 vs. Alt. 2/3
- Primary index never contains duplicates
- Secondary index may contain duplicates
  - If index key contains a candidate key, no duplicates => unique index

Dense vs. Sparse Index

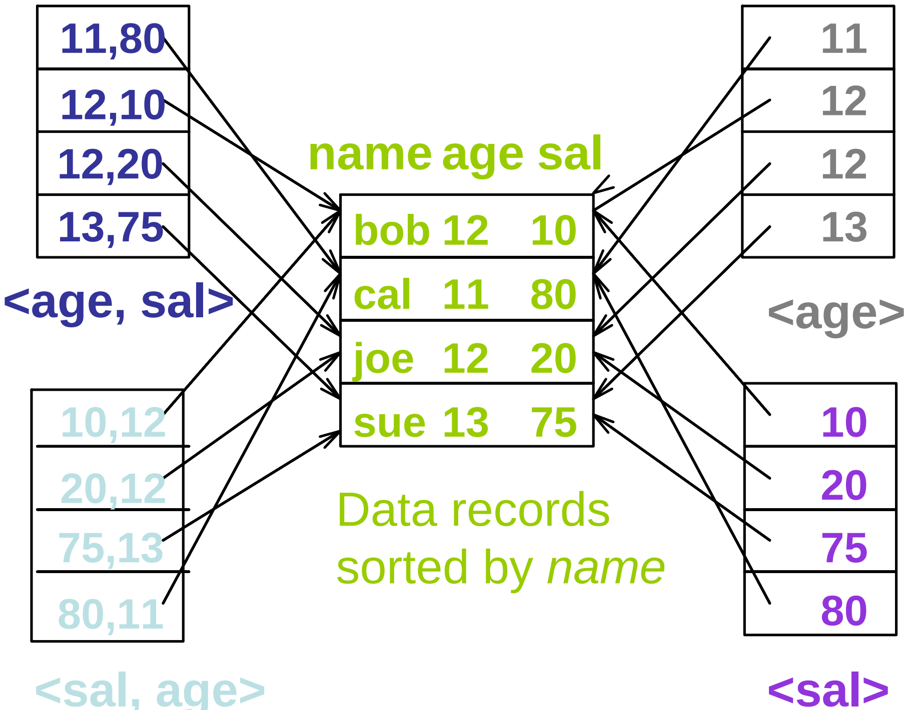
- *Dense:* at least one data entry per key value
- *Sparse:* an entry per data page in file
  - –Every sparse index is clustered!
  - –Sparse indexes are smaller; however, some useful optimizations are based on dense indexes.
  - –Alternative 1 always leads to dense index.



Composite Search Keys

- Search on *combination* of fields.
  - –Equality query: Every field is equal to a constant value. 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.



Tree vs. Hash-based index

- 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 in which record *r* belongs.
- Tree-based index
  - Good for range selections.
    - Hierarchical structure (Tree) directs searches
    - Leaves contain data entries sorted by search key value
    - B+ tree: all root->leaf paths have equal length *(height)*

Summary

- Catalog relations store information about relations, indexes and views.
- Many alternative file organizations exist, each appropriate in some situation.
- If selection queries are frequent, sorting the file or building an *index* is important.
- Index is a collection of data entries plus a way to quickly find entries with given key values.
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)

Summary (Cont.)

- Data entries in index can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
  - Choice orthogonal to *indexing structure (i.e. tree, hash, etc.)*.
- Usually have several indexes on a given file of data records, each with a different search key.
- Indexes can be classified as
  - clustered vs. unclustered
  - Primary vs. secondary
  - etc.
- Differences have important consequences for utility/performance.

What's examinable

- Describe alternative file organizations
- What is an index, when do we use them
- Index classification

Next Lecture

- Query processing part 1
  - Selection and projection (execution, costs)
  - Let’s demystify how DBMS perform work