Data Organization on Files

Davood Rafiei

Heap Files

- Rows appended to end of file as they are inserted
  - Hence the file is unordered
- Deleted rows create gaps in file
  - File must be periodically compacted to recover space

Transcript Stored as a Heap File

666666	MGT123	F1994	4.0
123456	CS305	S1996	4.0
987654	CS305	F1995	2.0

page 0

717171	CS315	S1997	4.0
666666	EE101	S1998	3.0
765432	MAT123	S1996	2.0
515151	EE101	F1995	3.0

page 1

234567	CS305	S1999	4.0
878787	MGT123	S1996	3.0

page 2

Heap File - Performance

- Assume file contains *F* pages
- *Searching for a row*:
  - Access path is scan
  - Avg. *F/*2 page transfers if row exists
  - *F* page transfers if row does not already exist
- *Deleting a row*:
  - Access path is scan
  - Avg. *F/*2+1 page transfers if row exists
  - *F* page transfers if row does not exist

Heap File - Performance

– Organization inefficient when a subset of rows is requested: *F* pages must be read

```
SELECT  T.Course, T.Grade
FROM  Transcript T                    -- equality search
WHERE  T.StudId = 123456

SELECT  T.StudId, T.CrsCode
FROM  Transcript T                    -- range search
WHERE  T.Grade BETWEEN 2.0 AND 4.0
```

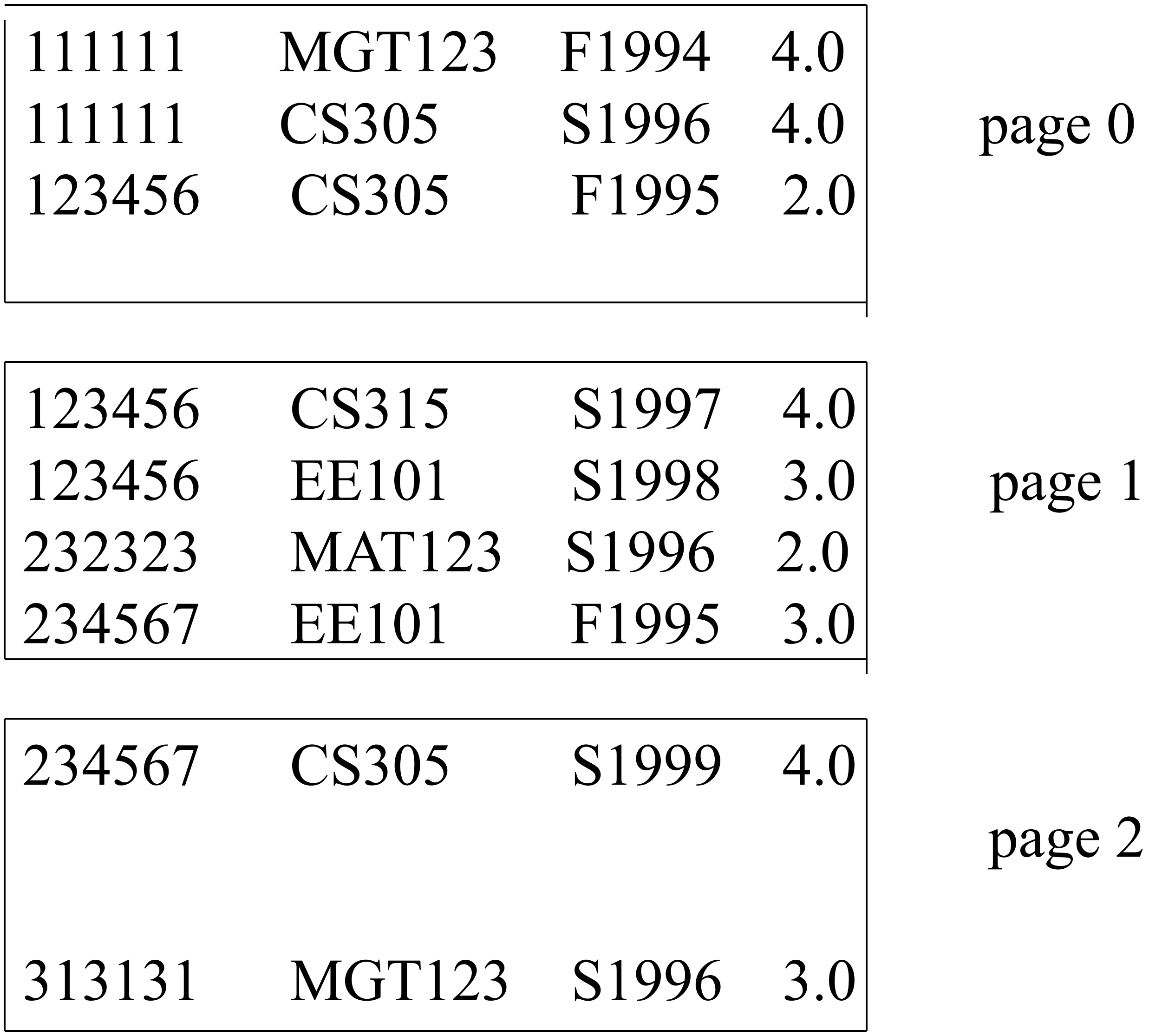
Sorted File

- Rows are sorted based on some attribute(s)
  - Access path is binary search
  - Equality or range query based on that attribute has cost log_2F to retrieve page containing first row
  - Successive rows are in same (or successive) page(s) and cache hits are likely
  - By storing all pages on the same track, seek time can be minimized
- Example – Transcript sorted on *StudId* :

```
SELECT  T.Course, T.Grade
FROM  Transcript T
WHERE  T.StudId = 123456
```

```
SELECT  T.Course, T.Grade
FROM  Transcript T
WHERE T.StudId BETWEEN
          111111 AND 199999
```

Transcript Stored as a Sorted File



page 0

111111	MGT123	F1994	4.0
111111	CS305	S1996	4.0
123456	CS305	F1995	2.0

page 1

123456	CS315	S1997	4.0
123456	EE101	S1998	3.0
232323	MAT123	S1996	2.0
234567	EE101	F1995	3.0

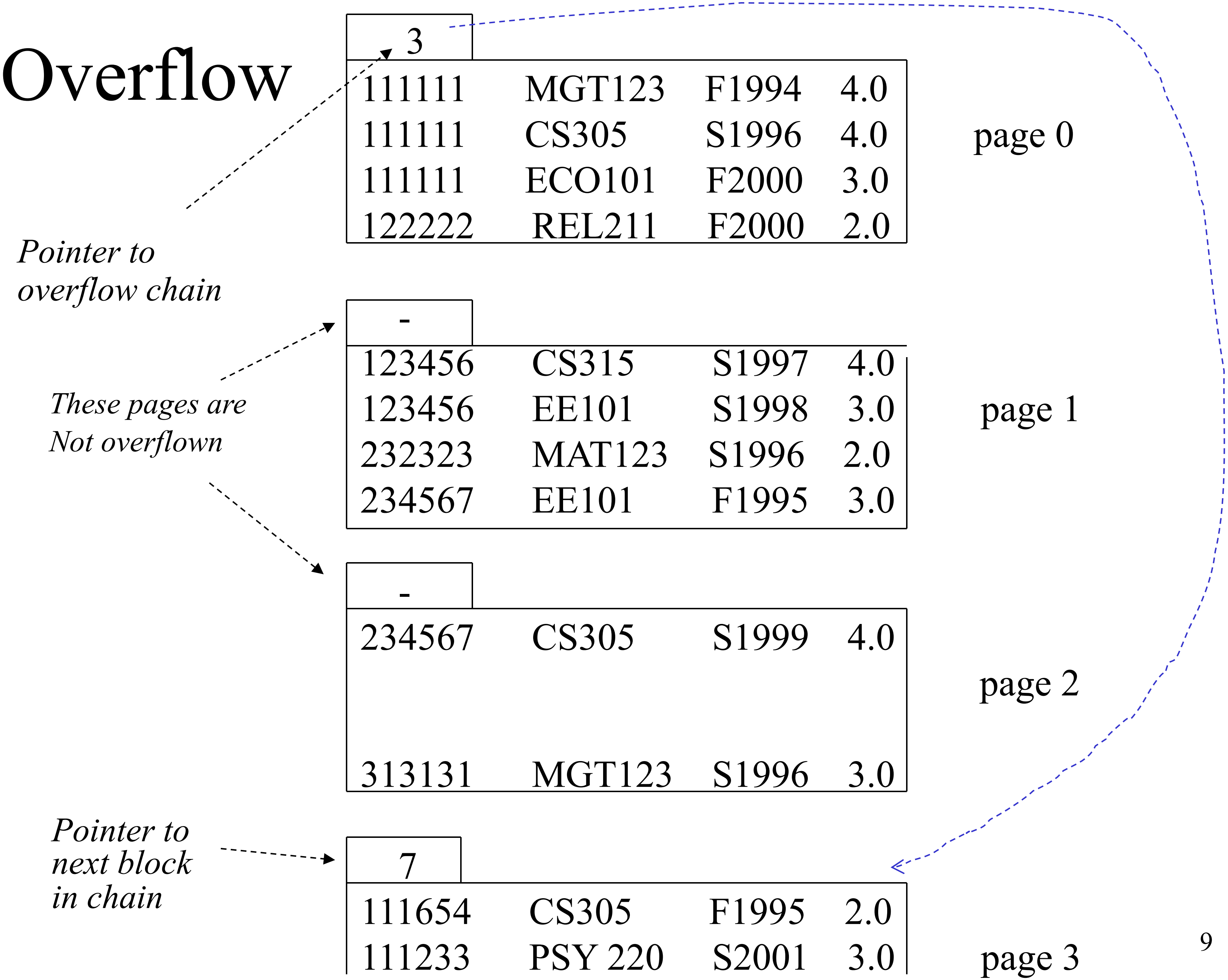
page 2

234567	CS305	S1999	4.0
313131	MGT123	S1996	3.0

Maintaining Sorted Order

- **Problem**: After the correct position for an insert has been determined, inserting the row requires (on average) $F/2$ reads and $F/2$ writes (because shifting is necessary to make space)
- **Partial Solution 1**: Leave empty space in each page: *fillfactor*
- **Partial Solution 2**: Use *overflow pages* (*chains*).
  - Disadvantages:
    - Successive pages no longer stored contiguously
    - Overflow chain not sorted, hence cost no longer $log_2 F$

Overflow



Pointer to overflow chain

These pages are Not overflown

Pointer to next block in chain

3

111111	MGT123	F1994	4.0
111111	CS305	S1996	4.0
111111	ECO101	F2000	3.0
122222	REL211	F2000	2.0

page 0

-

123456	CS315	S1997	4.0
123456	EE101	S1998	3.0
232323	MAT123	S1996	2.0
234567	EE101	F1995	3.0

page 1

-

234567	CS305	S1999	4.0
313131	MGT123	S1996	3.0

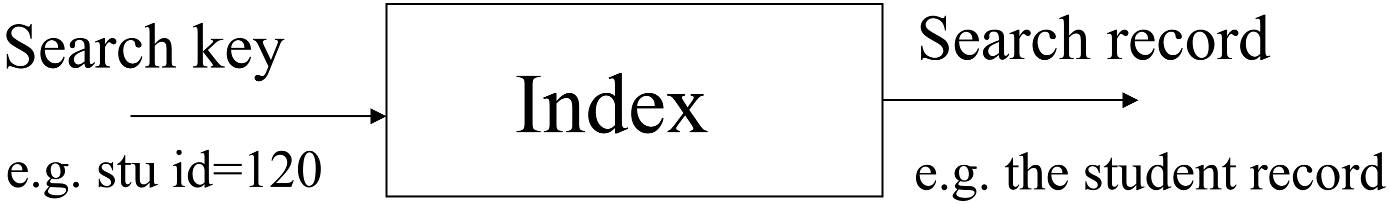
page 2

7

111654	CS305	F1995	2.0
111233	PSY 220	S2001	3.0

page 3

Index Files



Search key

e.g. stu id=120

Index

Search record

e.g. the student record

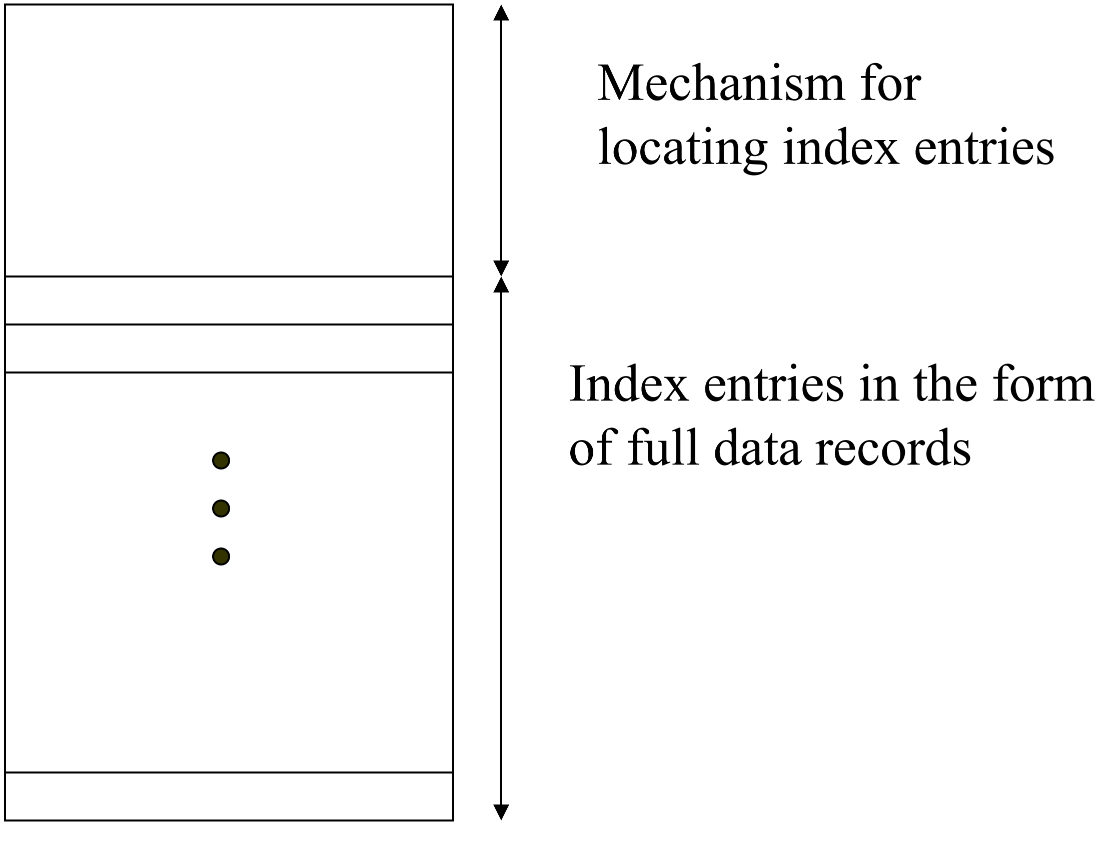
- Mechanism for efficiently locating row(s) without having to scan entire table
- Don’t confuse candidate key with search key:
  - Candidate key: *set* of attributes; *guarantees* uniqueness
  - Search key: *sequence* of attributes; *does not guarantee* uniqueness –just used for search

Index Properties

- *Index entries*
  - Full record vs key and a pointer
  - Integrated vs separate
  - Clustered vs unclustered
  - Dense vs sparse

Integrated Storage Structure

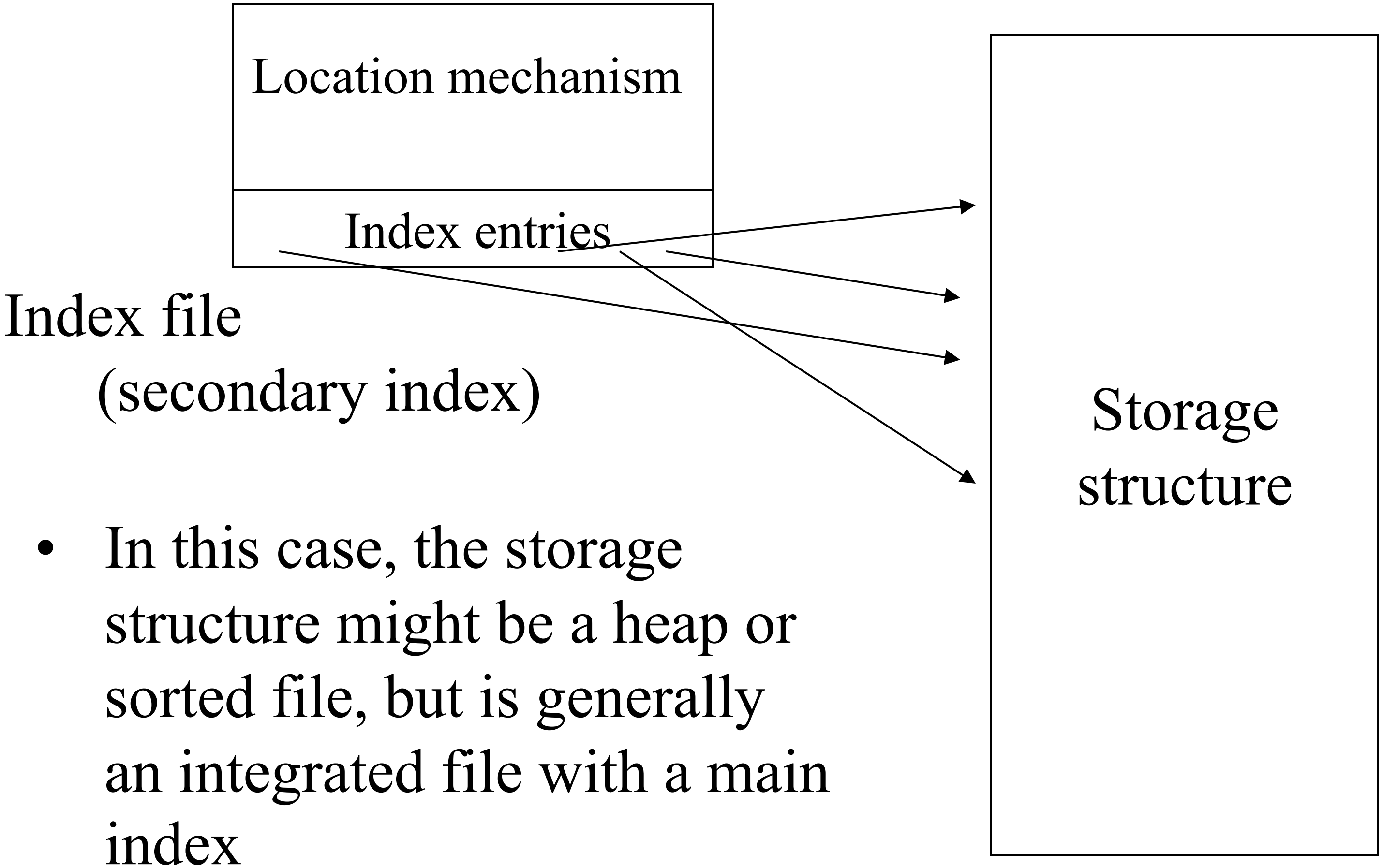
Contains table and (main) index



Mechanism for locating index entries
Index entries in the form of full data records

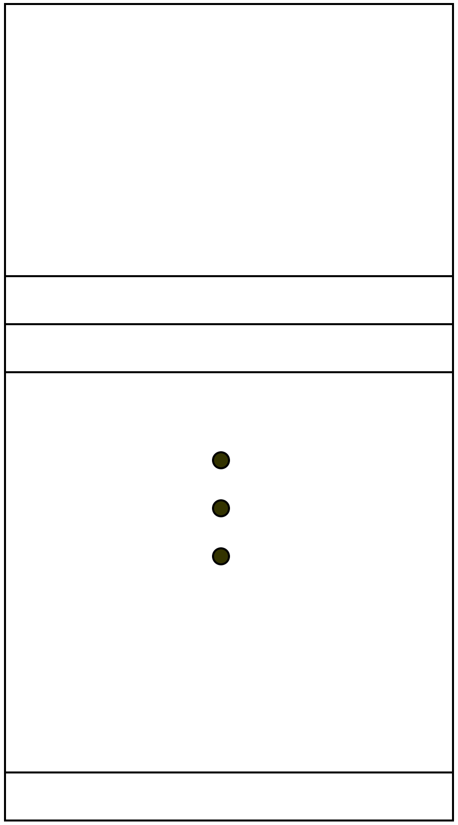
Data file

Index File With Separate Storage Structure



- In this case, the storage structure might be a heap or sorted file, but is generally an integrated file with a main index

Clustered Integrated Index

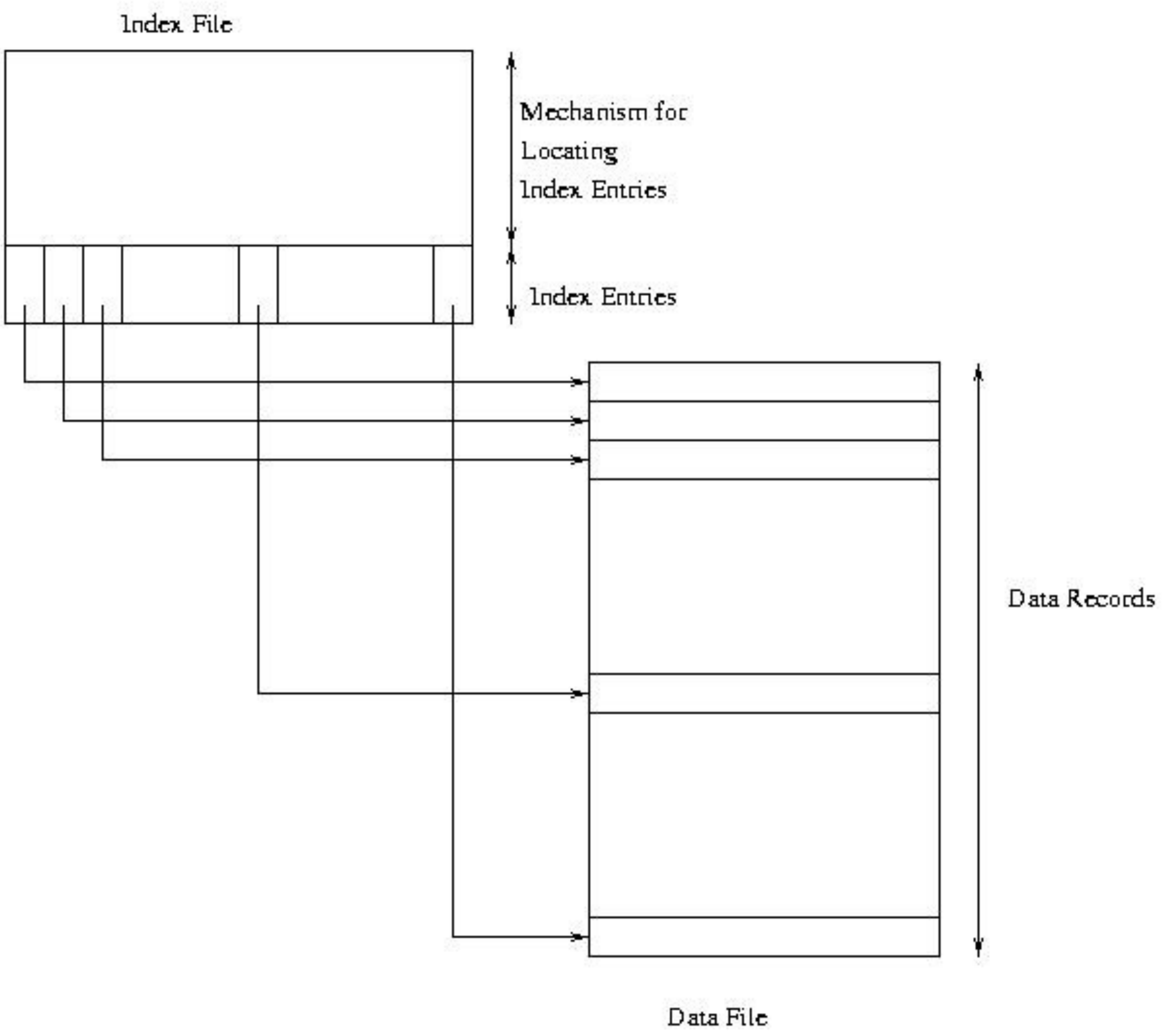


Mechanism for locating index entries

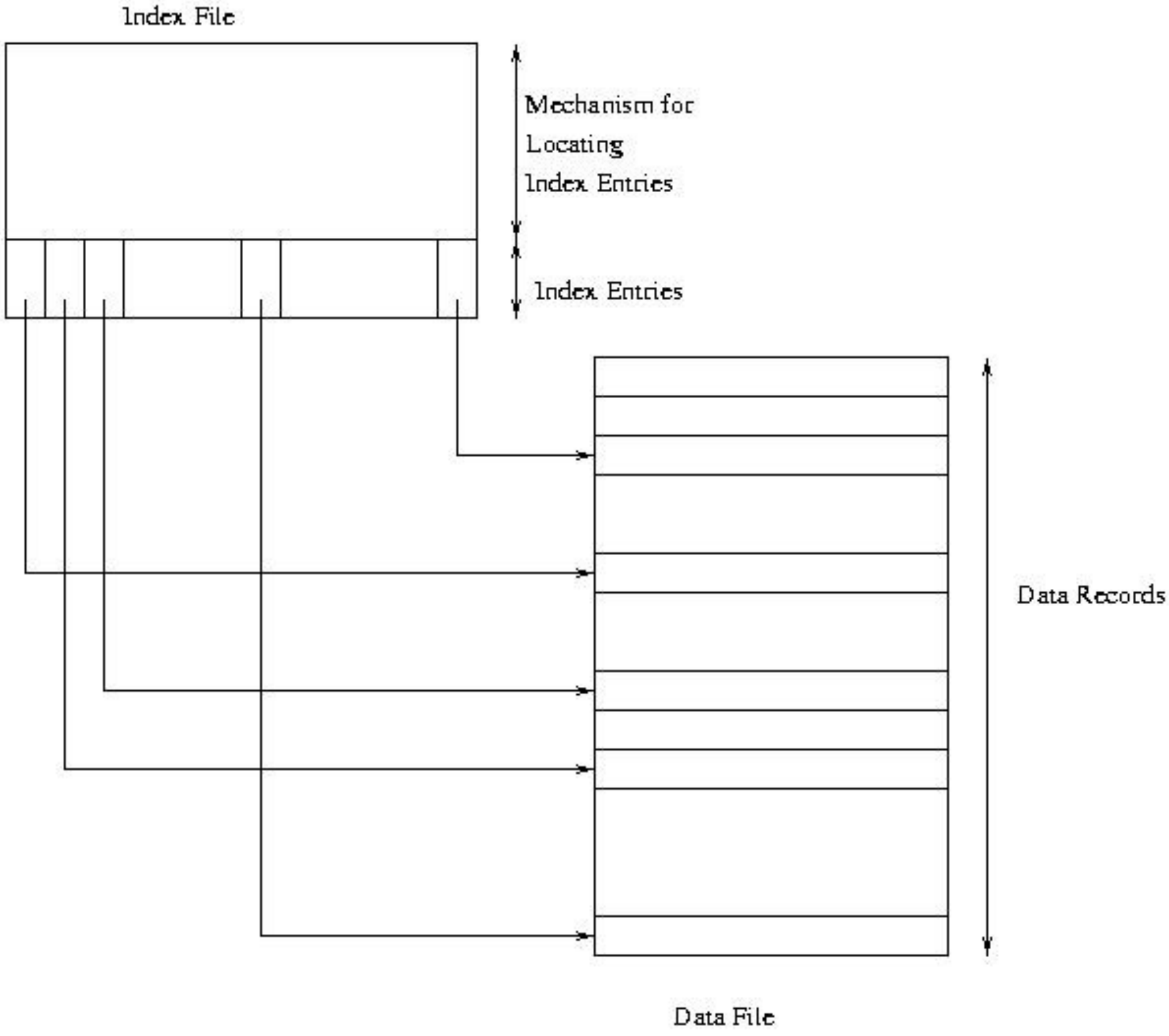
Index entries in the form of full data records

Data file

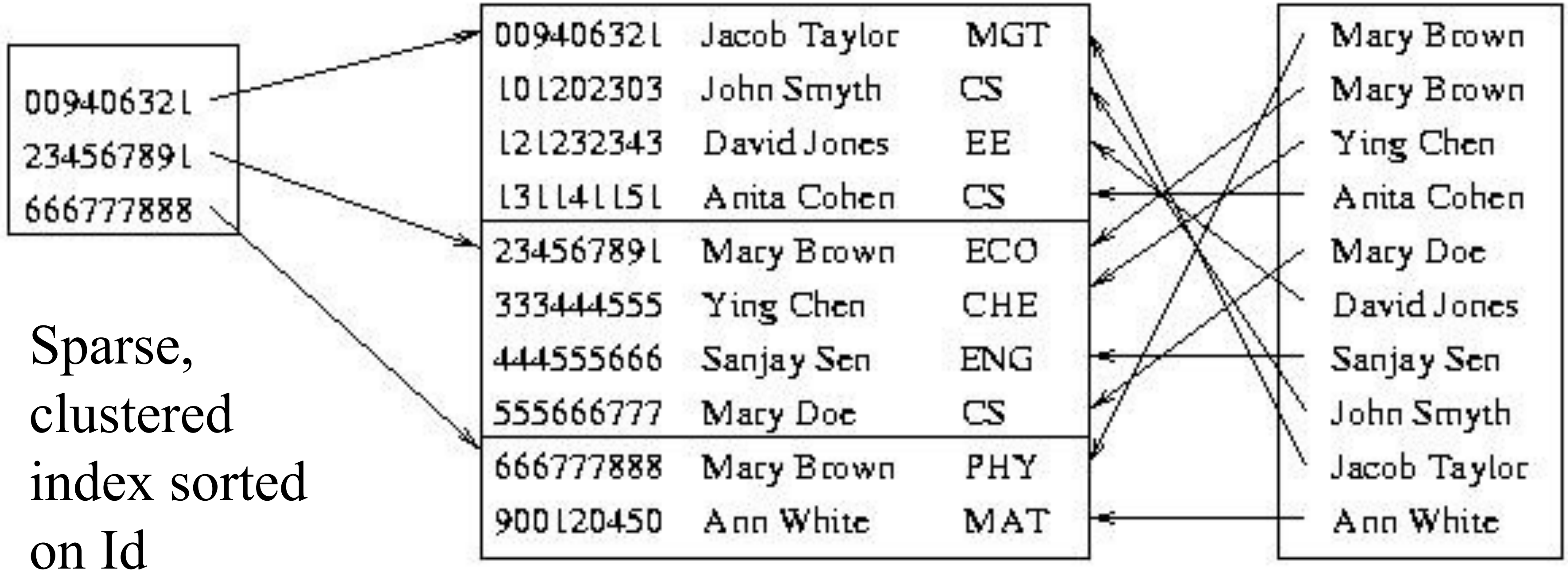
Clustered Separate Index



Unclustered Separate Index



Sparse Vs. Dense Index



Id	Name	Dept
009406321	Jacob Taylor	MGT
101202303	John Smyth	CS
121232343	David Jones	EE
131141151	Anita Cohen	CS
234567891	Mary Brown	ECO
333444555	Ying Chen	CHE
444555666	Sanjay Sen	ENG
555666777	Mary Doe	CS
666777888	Mary Brown	PHY
900120450	Ann White	MAT

009406321
234567891
666777888

Sparse, clustered index sorted on Id

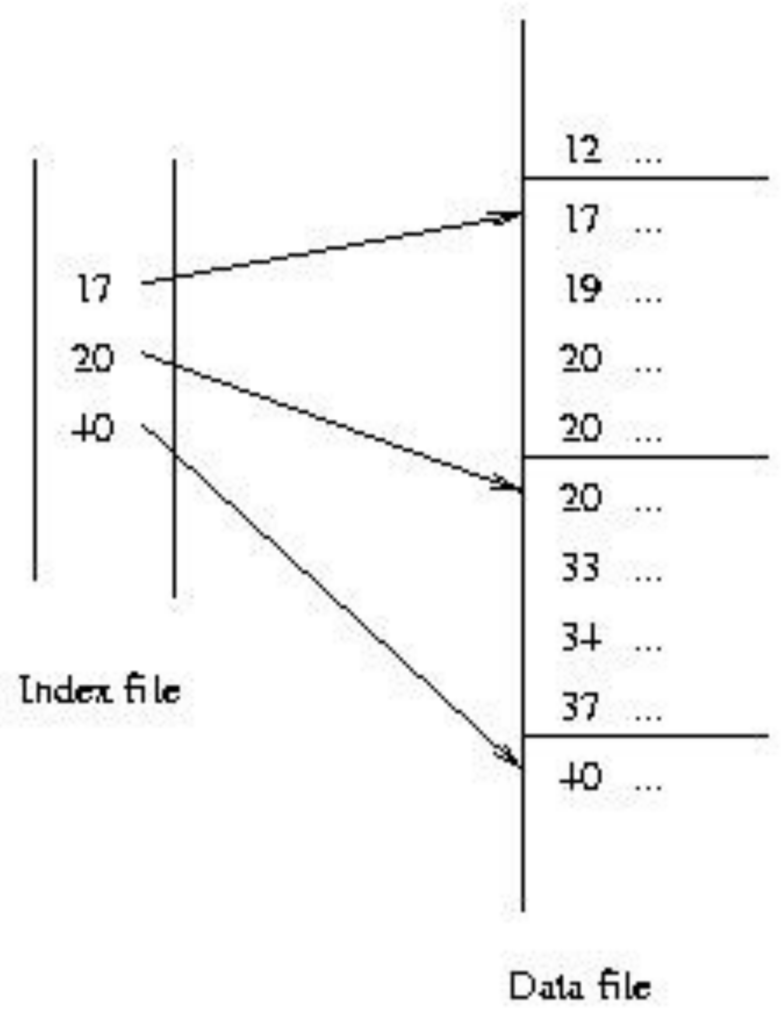
data file sorted on Id

Mary Brown
Mary Brown
Ying Chen
Anita Cohen
Mary Doe
David Jones
Sanjay Sen
John Smyth
Jacob Taylor
Ann White

Dense, unclustered index sorted on Name

Sparse Index

Search key should be candidate key of data file.



17
20
40
Index file
12 ...
17 ...
19 ...
20 ...
20 ...
20 ...
33 ...
34 ...
37 ...
40 ...
Data file