COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 16 EXERCISES
QUERY PROCESSING: EXPRESSION EVALUATION

EXERCISE 1

The Student relation consists of 10,000 tuples sorted on student id.

Each student has 5 attributes, each 20 bytes, so the tuple size is 100 bytes.

The page size is 1,000 bytes so, $bf_{Student} = \lfloor 1000/100 \rfloor = 10$.

Therefore, $B_{Student} = \lceil 10000/10 \rceil = 1{,}000$ pages.

The buffer size M is 100 pages and there are 5,000 different student names.

There is no index.

We want to evaluate the query:

```
select distinct name
from Student;
```

```
select distinct name
from Student;
```

EXERCISE 1 (cont'd)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1,000 pages
Each attribute: 20 bytes
Page size: 1,000 bytes
M pages: 100

a) Projection using external sorting

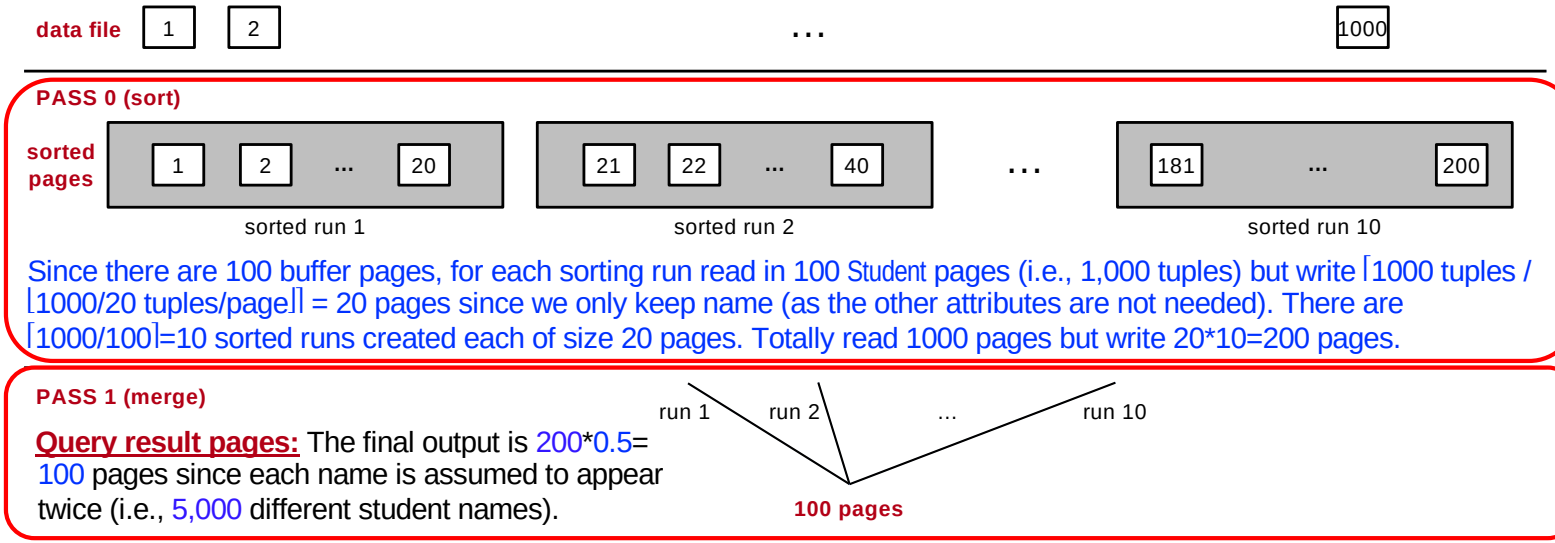
Pass 0: Read 100 pages containing 1000 tuples *for each sorted run* but write 20 pages creating total 10 sorted runs of 20 pages each.

Sort page I/O cost: 1000 pages read + 10*20 pages written = 1200

Pass 1: Read and merge 10 runs (200 pages) using a 10-way merge.

Merge page I/O cost: 200

Projection page I/O cost: 1200 + 200 = 1400



```
select distinct name
from Student;
```

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1,000 pages
Each attribute: 20 bytes
Page size: 1,000 bytes
M pages: 100

EXERCISE 1 (cont'd)

b) Projection using hashing (using 20 partitions)

Read the file page-by-page and assign each tuple to a partition.

For each tuple, keep only the name attribute. Thus, read 1,000 pages, but only write back $\lceil 10000 \text{ tuples} / \lfloor 1000/20 \text{ tuples/page} \rfloor \rceil = 200$ pages.

Partitioning page I/O cost: 1000 pages read + 200 pages written = 1200

At the next step, we load each partition into the buffer, build the in-memory hash table and perform duplicate elimination within the in-memory hash table partition.

Duplicate elimination page I/O cost: 200 (to read the 20 partitions)

Projection page I/O cost: 1200 + 200 = 1400

Query result pages: The final output is 200*0.5=100 pages since each name is assumed to appear twice (i.e., 5,000 different student names).

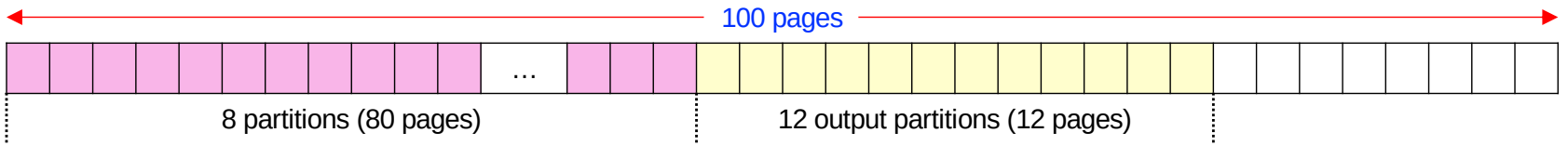
```
select distinct name
from Student;
```

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1,000 pages
Each attribute: 20 bytes
Page size: 1,000 bytes
M pages: 100

EXERCISE 1 (cont'd)

Optimization

Since there are 200 pages written and 20 partitions, the average partition size is 200 / 20 = 10 pages.



Given the large buffer, we can keep 8 full partitions in the buffer (i.e., 8 x 10 = 80 pages). This leaves 20 pages for the output buffers of which 1 is assigned to each of the remaining 12 partitions that are not kept in the buffer (i.e., 12 buffer pages are assigned as output pages).

This avoids writing and reading again the 8 partitions (i.e., 80 pages).

Projection page I/O cost: 1400 - 80 * 2 = 1240

(Compared to 1400 page I/Os.)

EXERCISE 2

Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400
ratings: 10
boats: 100

Sailor(sailorId, sName, rating, age) Reserves(sailorId, boatId, rDate)

For the Sailor relation, each tuple is 50 bytes, a page can hold 80 tuples and there are 500 full pages. For the Reserves relation, each tuple is 40 bytes, a page can hold 100 tuples and there are 1,000 full pages. There are 10 different sailor ratings and 100 different boats. Assume that sailors are distributed uniformly over the 10 ratings and reservations are distributed uniformly over the 100 boats.

Our goal is to process the query:

```
select sName
from Sailor natural join Reserves
where boatId=30
        and rating>5;
```

Sailor tuples: 500*80 = 40,000
Reserves tuples: 1,000*100 = 100,000

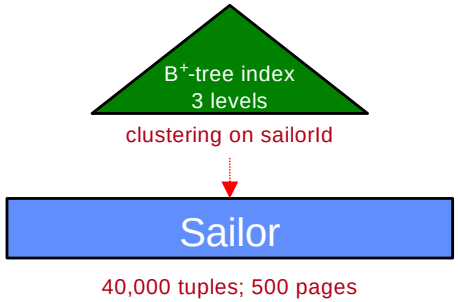
Some useful statistics

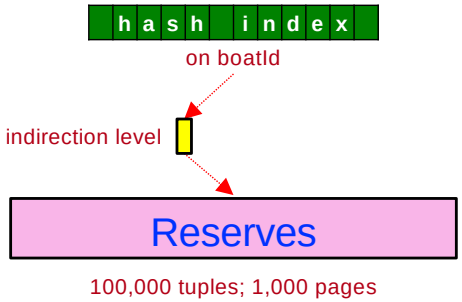
- On average, each sailor has (100,000/40,000) = 2.5 reservations.
- On average, each boat has (100,000/100) = 1,000 reservations.
- On average, for each rating there are (40,000/10) = 4,000 sailors.

EXERCISE 2 (CONT'D)

Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400
ratings: 10
boats: 100

The Sailor relation contains a clustering B^+-tree index with 3 levels on sailorId and the Reserves relation contains a hash index on boatId. Both the B^+-tree and hash index can fit 400 index entries per page. For non-clustering indexes, each pointer leads to a different page.






```
select sName
from Sailor natural join Reserves
where boatId=30
        and rating>5;
```

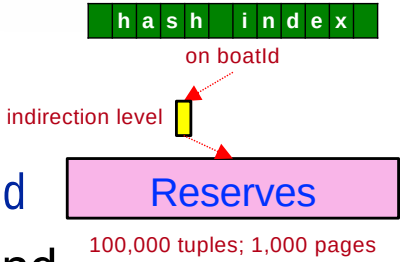
Some useful statistics

- On average, each sailor has 2.5 reservations.
- On average, each boat has 1,000 reservations.
- On average, for each rating there are 4,000 sailors.

Estimate the minimum page I/O cost of processing the query using a fully pipelined execution method (i.e., do not materialize anything except the query result).

```
select sName
from Sailor natural join Reserves
where boatId=30
        and rating>5;
```

EXERCISE 2 (CONT'D)



Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400
ratings: 10
boats: 100

a) Cost to evaluate $\sigma_{boatId=30}$

Strategy 1: use hash index on Reserves.boatId

Boat id 30 is hashed requiring 1 page I/O and $\lceil 100{,}000 / 100 \rceil = 1000$ index entries are returned (since the 100,000 reservations are uniformly distributed over the 100 boats).

The 1000 index entries occupy $\lceil 1000 / 400 \rceil = 3$ pages $\Rightarrow$ 3 page I/Os.

For each index entry, the corresponding Reserves tuple is retrieved requiring 1000 page I/Os (one for each index entry).

Page I/O cost: 1 + 3 + 1000 = 1004

Strategy 2: do linear search on Reserves

Page I/O cost: 1000

We only need to keep the sailorId for each tuple.

b) Cost to evaluate $\sigma_{rating>5}$

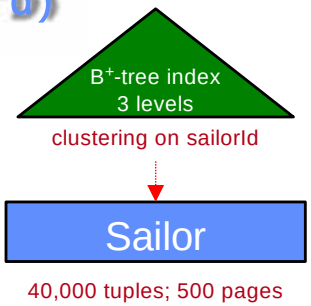
Strategy: do on-the-fly after join

Page I/O cost: 0

```
select sName
from Sailor natural join Reserves
where boatId=30
        and rating>5;
```

EXERCISE 2 (CONT'D)

Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400
ratings: 10
boats: 100



c) Cost to evaluate Sailor ⋈ Reserves

Strategy: use indexed nested-loop using sailorId index on Sailor

For each of the 1000 sailor ids in the result of the selection $\sigma_{boatId=30}$, we use the B$^+$-tree index on sailorId to find the corresponding Sailor tuple.

☞ For each tuple, we check *on-the-fly* if it meets the condition rating>5.

Page I/O cost: 1000 * (3 levels + 1 for record retrieval) = 4000

d) Cost to evaluate π_{sName}

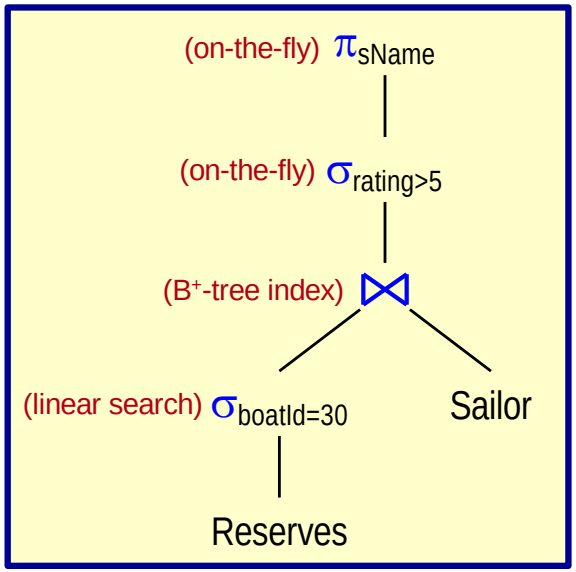
Strategy: do *on-the-fly* after join

Page I/O cost: 0

Query page I/O cost:

1000 + 0 + 4000 + 0 = 5000

Execution plan



EXERCISE 3

The Sailor relation consists of 40,000 tuples sorted on sailor id.

Each sailor has 4 attributes, each 10 bytes, so the tuple size is 40 bytes.

The page size is 800 bytes so $bf_{Sailor} = \lfloor 800 / 40 \rfloor = 20$.

Therefore, $B_{Sailor} = \lceil 40000 / 20 \rceil = 2000$ pages.

The buffer size M is 100 pages and 5% of sailor names are the same.

There is no index.

We want to evaluate the query:

```
select distinct sName
from Sailor;
```

```
select distinct sName
from Sailor;
```

Sailor tuples: 40,000
bf_{Sailor}: 20 tuples/page
B_{Sailor}: 2,000 pages
Page size: 800 bytes
M pages: 100

EXERCISE 3 (CONT'D)

a) Projection using external sorting

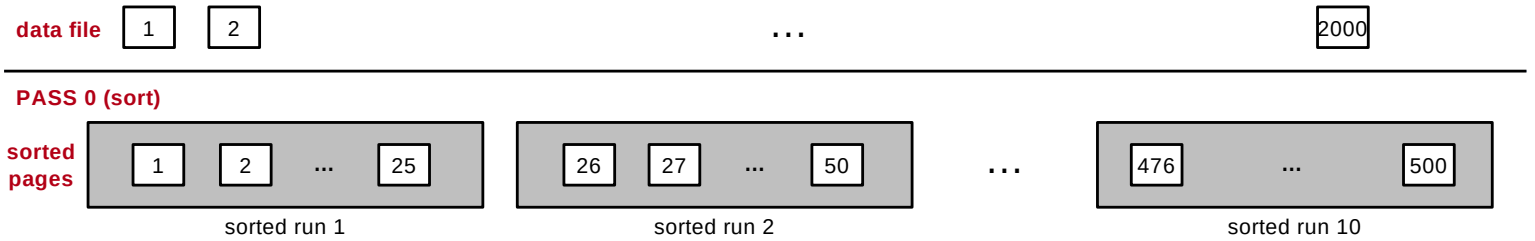
Pass 0: Read 100 pages containing 2000 tuples *for each sorted run* but write 25 pages creating total 20 sorted runs of 25 pages.

Sort page I/O cost: 2000 pages read + 20 * 25 pages written = 2500

Pass 1: Read and merge the 500 pages using a 20-way merge.

Merge page I/O cost: 500

Projection page I/O cost: 2500 + 500 = 3000



Since there are 100 buffer pages, for each sorting run read in 100 Sailor pages but write ⌈2000 tuples / ⌊800/10 tuples/page⌋⌉ = 25 pages since we only keep name (as the other attributes are not needed). There are ⌈2000/100⌉=20 sorted runs created each of size 25 pages. Totally read 2000 pages but write 20*25=500 pages.

PASS 1 (merge)

Result pages: The final output is 500*0.95 = 475 pages since it is assumed that 5% of sailor names are the same.

```
select distinct sName
from Sailor;
```

Sailor tuples: 40,000
bf_{Sailor}: 20 tuples/page
B_{Sailor}: 2,000 pages
Page size: 800 bytes
M pages: 100

EXERCISE 3 (CONT'D)

b) Projection using hashing (using 40 partitions; no optimization)

Read the file page-by-page and assign each tuple to a partition.

For each tuple, keep only the name attribute (i.e., read 2,000 pages, but only write back $\lceil 40{,}000 / \lfloor 800 / 10 \rfloor \rceil = 500$ pages).

Partitioning page I/O cost: 2000 pages read + 500 pages written = 2500

At the next step, load each partition into the buffer, build the in-memory hash table and perform duplicate elimination within the in-memory hash table partition.

Duplicate elimination page I/O cost: 500 (to read the 40 partitions)

Projection page I/O cost: 2500 + 500 = 3000

Query result pages: The final output is 500*.95 = 475 pages since it is assumed that 5% of sailor names are the same.

EXERCISE 4

Student(sId, name, deptId, address) EnrollsIn(courseId, *sId*, semester, grade)

The Student relation contains 10,000 tuples in 1,000 pages and the EnrollsIn relation contains 50,000 tuples in 5,000 pages. There are 25 different departments and 1,000 different courses. All attributes have the same length.

Our goal is to process the query:

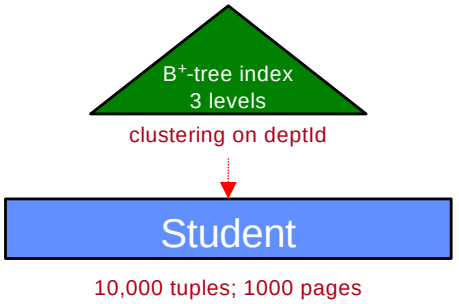
```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
        and deptId='COMP';
```

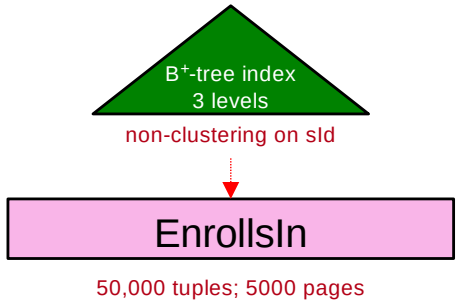
Some useful statistics

- On average, a student enrolls in (50,000/10,000) = 5 courses.
- On average, a department has (10,000/25) = 400 students.
- On average, each course has an enrollment of (50,000/1,000) = 50 students.

EXERCISE 4 (cont'd)

The Student relation contains a clustering index on deptId and the EnrollsIn relation contains a non-clustering index on sId. Each available index is a B^+-tree with 3 levels. For non-clustering indexes, each pointer leads to a different page.






```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
        and deptId='COMP';
```

Some useful statistics

- On average, a student enrolls in 5 courses.
- On average, a department has 400 students.
- On average, each course has an enrollment of 50 students.

Estimate the minimum page I/O cost of processing the query using a fully pipelined execution method (i.e., do not materialize anything except the query result).

```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
        and deptId='COMP';
```

EXERCISE 4 (cont'd)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5
departments: 25
courses: 1000

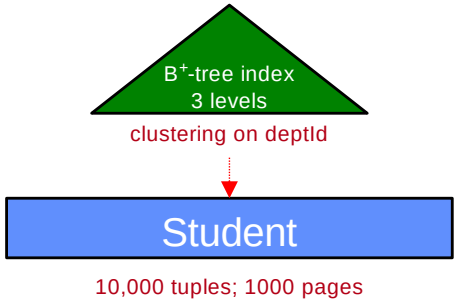
a) Cost to evaluate $\sigma_{courseId='COMP3311'}$

Strategy: do on-the-fly after join

Page I/O cost: 0

b) Cost to evaluate $\sigma_{deptId='COMP'}$

Strategy: use B^+-tree index on Student.deptId



The selection reads 3 index pages using the clustering index on deptId.

Since the file is ordered on deptId, there are 25 different departments and students are distributed uniformly over departments, $\lceil 1000 / 25 \rceil$ = 40 pages of Student tuples are read.

These 40 pages contain (40 pages * 10 tuples/page) = 400 student tuples.

We only need to keep the student ids and the name for each tuple.

Page I/O cost: 3 + 40 = 43

```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
        and deptId='COMP';
```

EXERCISE 4 (cont'd)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5
departments: 25
courses: 1000

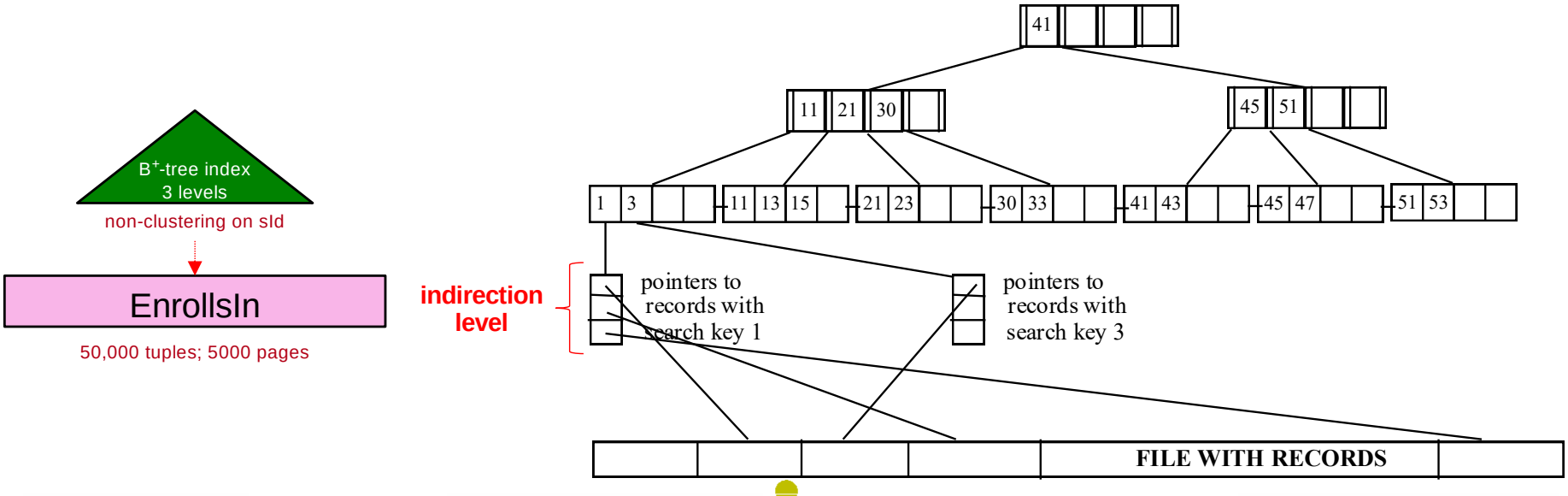
c) Cost to evaluate Student ⋈ EnrollsIn

Strategy: use indexed nested-loop using sId index on EnrollsIn

For each of the 400 students in the selection result, we use the sId index on EnrollsIn to find the corresponding EnrollsIn tuples.

For each sId (student) we need to retrieve 5/2 = 2.5 EnrollsIn tuples, on average.

Since the sId index on EnrollsIn is non-clustering, we first need to retrieve the pointers to the EnrollsIn records as shown in the figure below.



```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
        and deptId='COMP';
```

EXERCISE 4 (cont'd)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5
departments: 25
courses: 1000

Assuming the 2.5 record pointers per student all fit on a page (which is highly likely), we need 400 page I/Os to retrieve the record pointers (one for each student).

For each EnrollsIn tuple retrieved, we check *on-the-fly* if it meets the condition courseId='COMP3311'.

Page I/O cost: 400*(3+1+ 2.5) = 2600

d) Cost to evaluate π_{name}

Strategy: do *on-the-fly* after join

Page I/O cost: 0

Query page I/O cost:

0 + 43 + 2600 + 0 = 2643

Execution plan