CSE 444: Database Internals

Section 4:

Query Optimizer

Plan for Today

- Problem 1A, 1B: Estimating cost of a plan
  - You try to compute the cost for 5 mins
  - We go over the solution together

- Problem 2: Sellinger Optimizer
  - We will do it together

1. Estimating Cost of a given plan

Student (sid, name, age, address)
Book(bid, title, author)
Checkout(sid, bid, date)

Query:

```
SELECT S.name
FROM Student S, Book B, Checkout C
WHERE S.sid = C.sid
AND B.bid = C.bid
AND B.author = 'Olden Fames'
AND S.age > 12
AND S.age < 20
```

S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

Assumptions

- Student: S, Book: B, Checkout: C

- Sid, bid foreign key in C referencing S and B resp.
- There are 10,000 Student records stored on 1,000 pages.
- There are 50,000 Book records stored on 5,000 pages.
- There are 300,000 Checkout records stored on 15,000 pages.
- There are 500 different authors.
- Student ages range from 7 to 24.

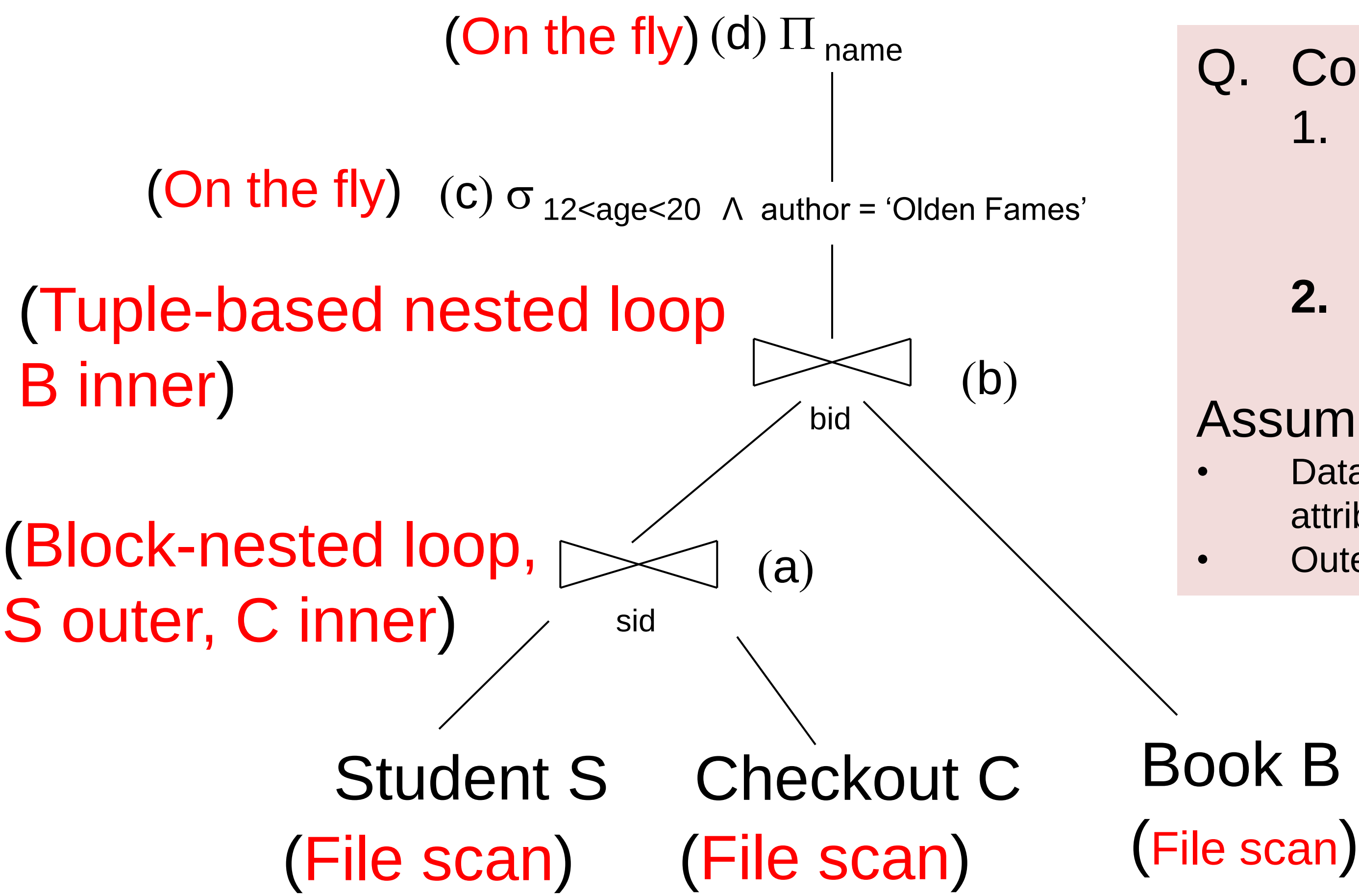
S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author)
C(<u>sid,bid</u>,date)

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

Physical Query Plan – 1A



Q. Compute
1. the cost and cardinality in steps (a) to (d)
2. **the total cost**

Assumptions:
- Data is not sorted on any attributes
- Outer relation fits in memory

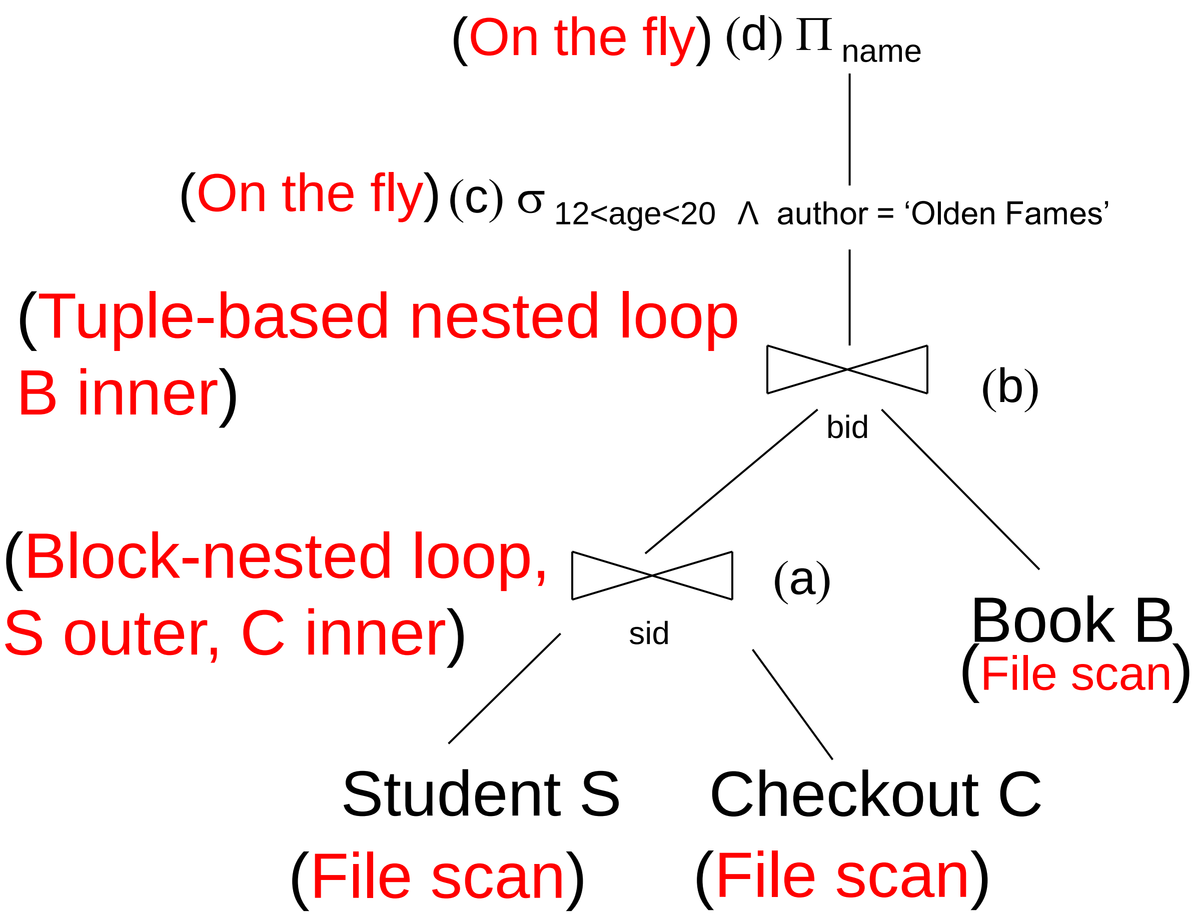
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

Solution – 1A



(a) B(S) + B(S) * B(C)
= 1000 + 1000 * 15000
= 15,001,000

cardinality = 300,000
(foreign key join, output pipelined to next join)
Also, applying the formula, join size=T(S) * T(C)/max (V(S, sid), V(C, sid))
= T(S) since V(S, sid) > = V(C, sid) and T(S) = V(S, sid)

(b) T(S ⋈ C) * B(B)
= 300,000 * 5,000 = 15 * 10^8
cardinality = 300,000
(foreign key join, don't need scanning for outer relation)

(c, d) cost 0 (on the fly)
Cardinality:
300,000 * 1/500 * 7/18
= 234 (approx)
(assuming uniformity and independence)

Total cost = 1,515,001,000
Final cardinality = 234 (approx)

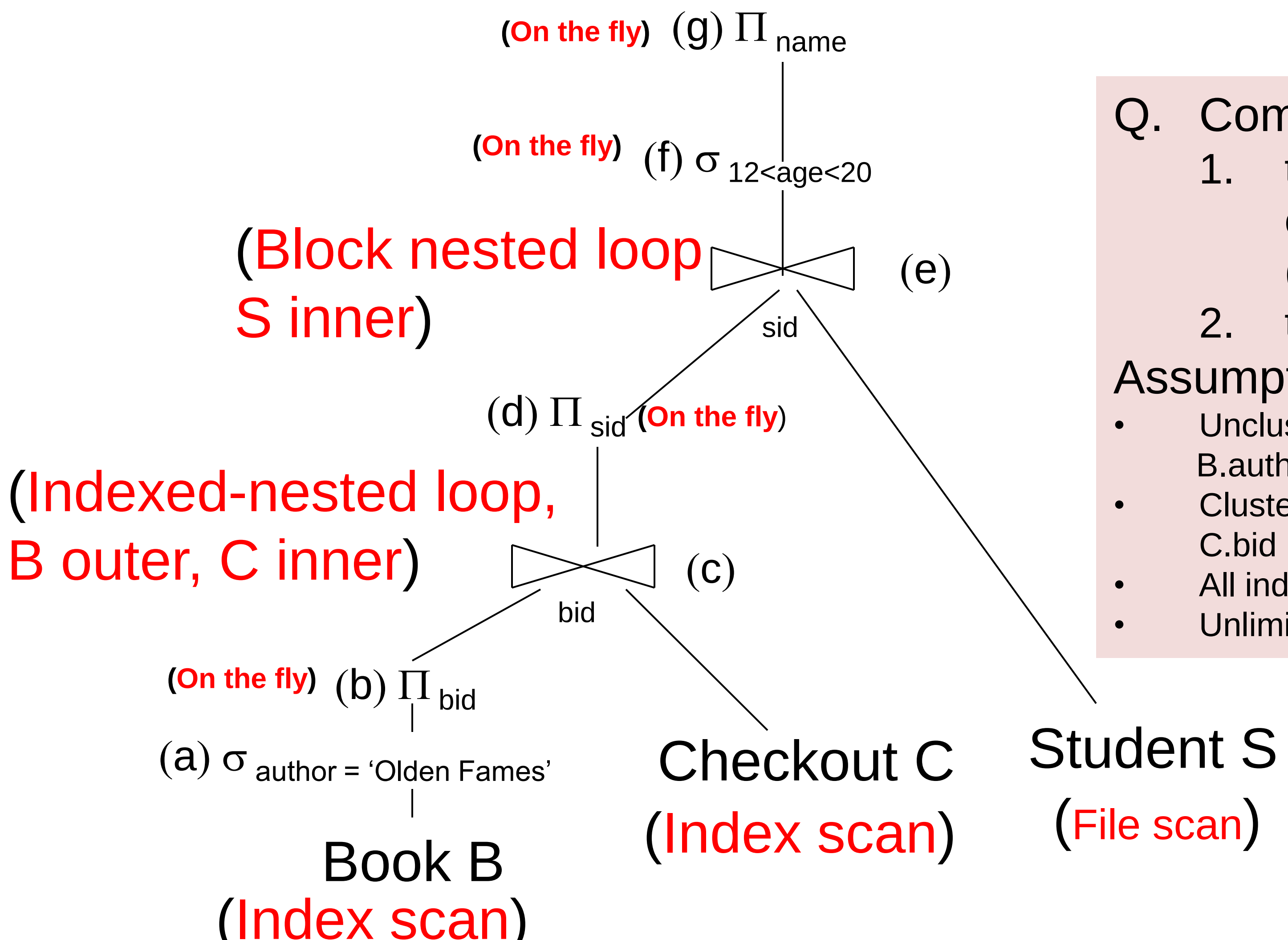
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

Physical Query Plan – 1B



Q. Compute
1. the cost and cardinality in steps (a) to (g)
2. the total cost

Assumptions:
- Unclustered B+tree index on B.author
- Clustered B+tree index on C.bid
- All index pages are in memory
- Unlimited memory

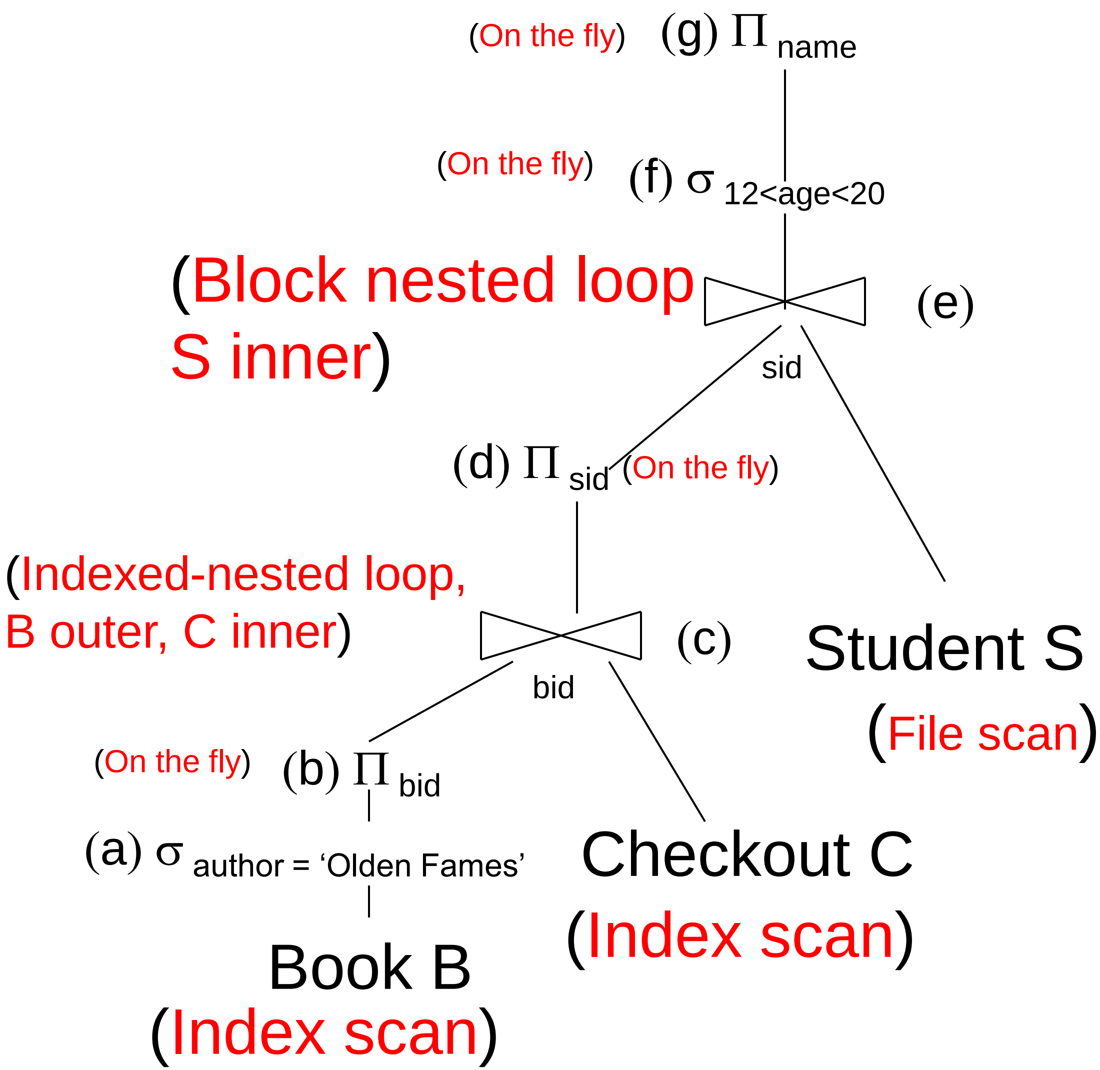
S(<u>sid</u>,name,age,addr)
B(<u>bid</u>,title,author): Un. B+ on author
C(<u>sid,bid</u>,date): Cl. B+ on bid

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

Solution – 1B



(a) **T(B) / V(B, author)**

cost = 50,000/500 = 100 (unclustered)

cardinality = 100

(b) Cost 0, cardinality 100

(c)

i. one index lookup per outer B tuple
ii. 1 book has 6 checkouts (uniformity)
iii. # C tuples per page = T(C)/B(C) = 20
iv. 6 tuples fit in at most 2 consecutive pages (clustered) – could assume 1 page as well

Cost <= 100 * 2= 200

cardinality = 100 * 6 = 600

(= 100 * T(C)/ MAX(100, V(C, bid)) assuming V(C, bid) = V(B, bid) = T(B) = 50,000)

(d) Cost 0, cardinality 600

(e) Outer relation is already in memory, need to scan S relation

cost B(S) = 1000

Cardinality = 600

(f) Cost = 0

Cardinality = 600 * 7/18 = 234 (approx)

(d) Cost 0, cardinality 234

Total cost = 1300 (compare with 1,515,001,000 in 1A!)

Final cardinality = 234 (approx) (same as 1A!)

2. Sellinger Optimization Example

Sailors (sid, sname, srating, age)
Boats(bid, bname, color)
Reserves(sid, bid, date, rname)

Query:

```
SELECT S.sid, R.rname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid
AND B.bid = R.bid
AND B.color = red
```

Example is from the Ramakrishnan book

S (sid, sname, srating, age)
B (bid, bname, color)
R (sid, bid, date, rname)

Available Indexes

- Sailors: S, Boats: B, Reserves: R
- Sid, bid foreign key in R referencing S and B resp.
- Sailors
  - Unclustered B+ tree index on sid
  - Unclustered hash index on sid
- Boats
  - Unclustered B+ tree index on color
  - Unclustered hash index on color
- Reserves
  - Unclustered B+ tree on sid
  - Clustered B+ tree on bid

S (sid, sname, srating, age): B+tree - sid, hash index - sid
B (bid, bname, color) : B+tree - color, hash index - color
R (sid, bid, date, rname) : B+tree - sid, **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

First Pass

- Where to start?
  - How to access each relation, assuming it would be the first relation being read
  - File scan is also available!
- Sailors?
  - No selection matching an index, use File Scan (no overhead)
- Reserves?
  - Same as Sailors
- Boats?
  - Hash index on color, matches B.color = red
  - B+ tree also matches the predicate, but hash index is cheaper
    - B+ tree would be cheaper for range queries

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Second Pass

- What next?
  - For each of the plan in Pass 1 taken as outer, consider joining another relation as inner
- What are the combinations? How many new options?

Outer	Inner	OPTION 1	OPTION 2	OPTION 3
R (file scan)	**B**	(B+-color)	(hash color)	(File scan)
R (file scan)	**S**	(B+-sid)	(hash sid)	,,
S (file scan)	**B**	(B+-color)	(hash color)	,,
S (file scan)	**R**	(B+-sid)	(Cl. B+ bid)	,,
B (hash index)	**R**	(B+-sid)	(Cl. B+ bid	,,
B (hash index)	**S**	(B+-sid)	(hash sid)	,,

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Second Pass

- Which outer-inner combinations can be discarded?
  - B, S and S, B: Cartesian product!

Outer	Inner	OPTION 1	OPTION 2	OPTION 3
R (file scan)	**B**	(B+-color)	(hash color)	(File scan)
R (file scan)	**S**	(B+-sid)	(hash sid)	,,
~~S (file scan)~~	~~**B**~~	~~(B+-color)~~	~~(hash color)~~	~~,,~~
S (file scan)	**R**	(B+-sid)	(Cl. B+ bid)	,,
~~B (hash index)~~	~~**S**~~	~~(B+-sid)~~	~~(hash sid)~~	~~,,~~
B (hash index)	**R**	(B+-sid)	(Cl. B+ bid):	,,

OPTION 3 is not shown on next slide, expected to be more expensive

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Outer	Inner	OPTION 1	OPTION 2
R (file scan)	**S**	**(B+-sid)** Slower than hash-index (need Sailor tuples matching S.sid = value, where value comes from an outer R tuple)	**(hash sid):** likely to be faster **2A. Index nested loop join** **2B Sort Merge based join:** (no index is sorted on sid, need to sort, output sorted by sid, retained if cheaper)
R (file scan)	**B**	**(B+-color)** Not useful	**(hash color)** Consider all methods, select those tuples where B.color = red using the color index (note: no index on bid)
S (file scan)	**R**	**(B+-sid)** Consider all methods	**(Cl. B+ bid)** Not useful
B (hash index)	**R**	**(B+-sid)** Not useful	**(Cl. B+ bid)** **2A. Index nested loop join** (no H. I. on bid) **2B. Sort-merge join** (clustered, index sorted on bid, produces outputs in sorted order by bid, retained if cheaper)

Keep the least cost plan between
- (R, S) and (S, R)
- (R, B) and (B, R)

S (sid, sname, srating, age): 1. B+tree - sid, 2. hash index - sid
B (bid, bname, color) : 1. B+tree - color, 2. hash index - color
R (sid, bid, date, rname) : 1. B+tree - sid, 2. **Clustered** B+tree - bid

SELECT S.sid, R.rname
WHERE S.sid = R.sid
B.bid = R.bid, B.color = red

Third Pass

- Join with the third relation
- For each option retained in Pass 2, join with the third relation
- E.g.
  - Boats (B+tree on color) – sort-merged-join – Reserves (B+tree on bid)
  - Join the result with Sailors (B+ tree on sid) using sort-merge-join
    - Need to sort (B join R) by sid, was sorted on bid before
    - Outputs tuples sorted by sid
    - Not useful here, but will be useful if we had GROUP BY on sid
    - In general, a higher cost "**interesting**" plans may be retained (e.g. sort operator at root, grouping attribute in group by query later, join attriute in a later join)

Tomorrow, Lecture 12

- Pseudocode for Sellinger Optimization as a dynamic programming
- Complexity of the algorithm