CS3223 Tutorial 4

Gary Lim

Admin

- **Lab 4 (Group) Report** is due on Monday, 21 Feb
- **Midterms** is on the **4th of March** (Friday) **10 to 11:30am**. More details will be provided later!
  - Venue: MPSH2

Agenda

Chapter Review

Tutorial Questions

Q & A

Chapter Review

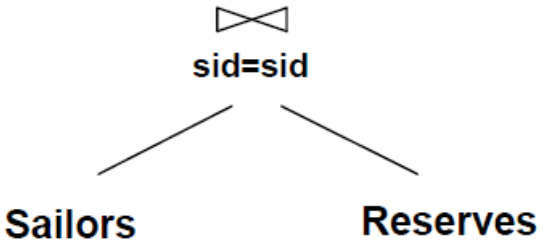
Join Algorithms

- Iterative-based
  - Simple Nested Loop
  - Page-Nested Loop
  - Block-Nested Loop
- Sort-Merge Join
- GRACE Hash-Join
- Index Nested Loop Join

Join Algorithms

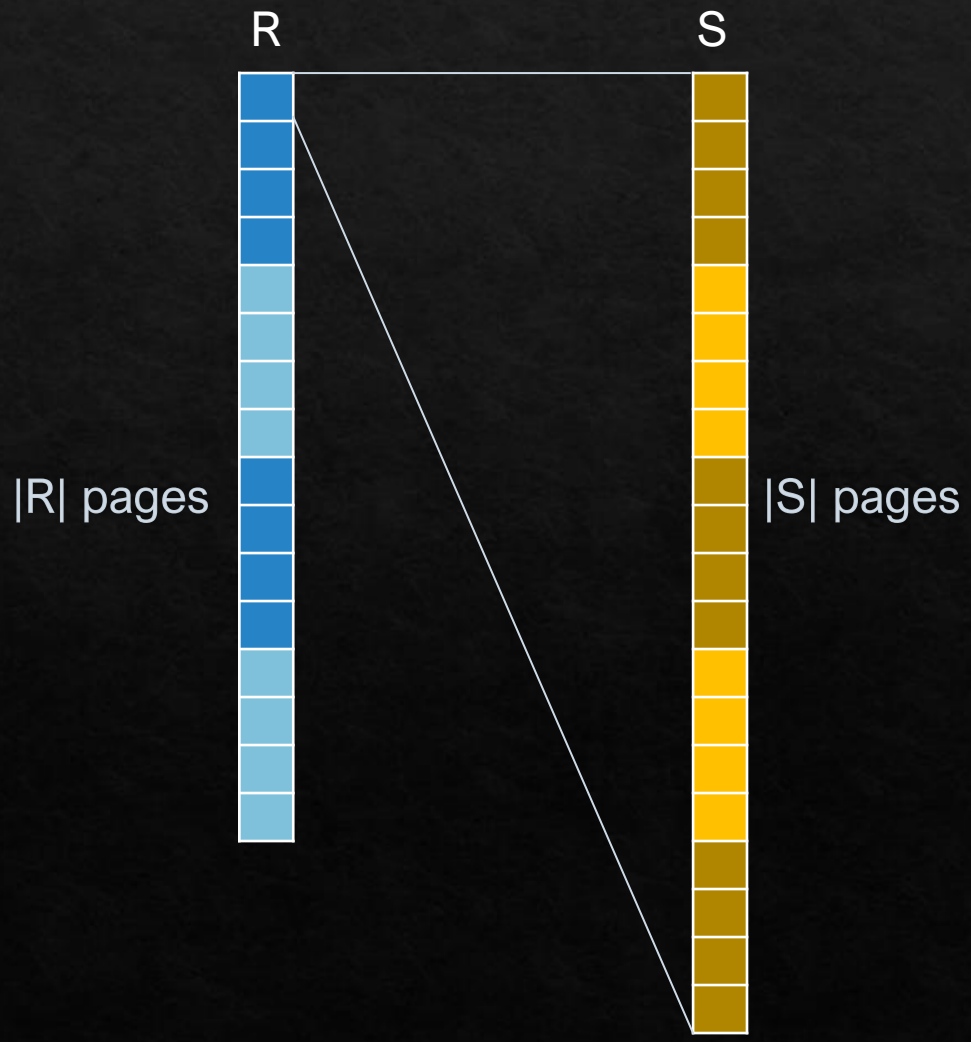
- Frequently used operation in Relational Databases (join on some foreign key)
- Costly, have to scan entire relations
- Join Selectivity (ρ):
  - $||R \bowtie S|| = \rho \times ||R|| \times ||S||$

```
SELECT  *
FROM    Reserves R, Sailors S
WHERE   R.sid=S.sid
```



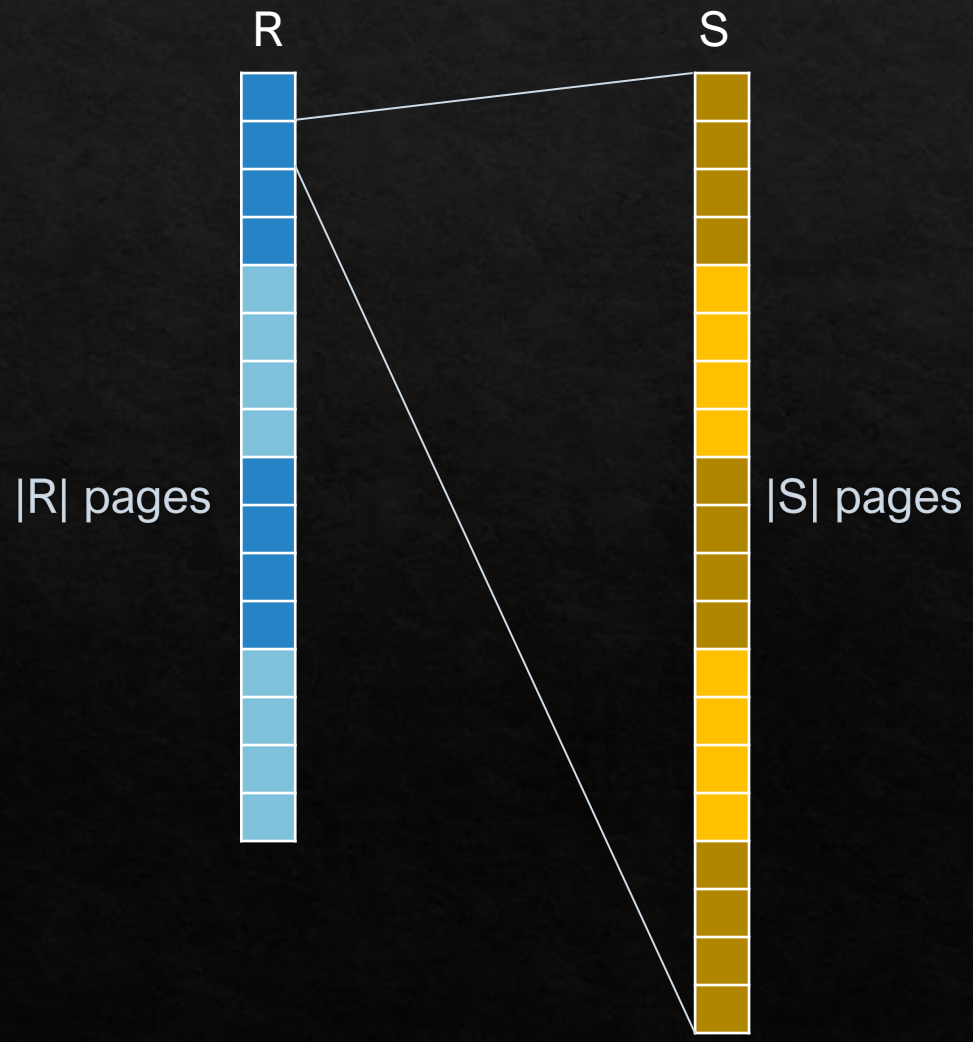
Simple Nested Loop Join

```
foreach tuple r in R do
    foreach tuple s in S do
        if r.sid == s.sid then add <r, s> to result
```



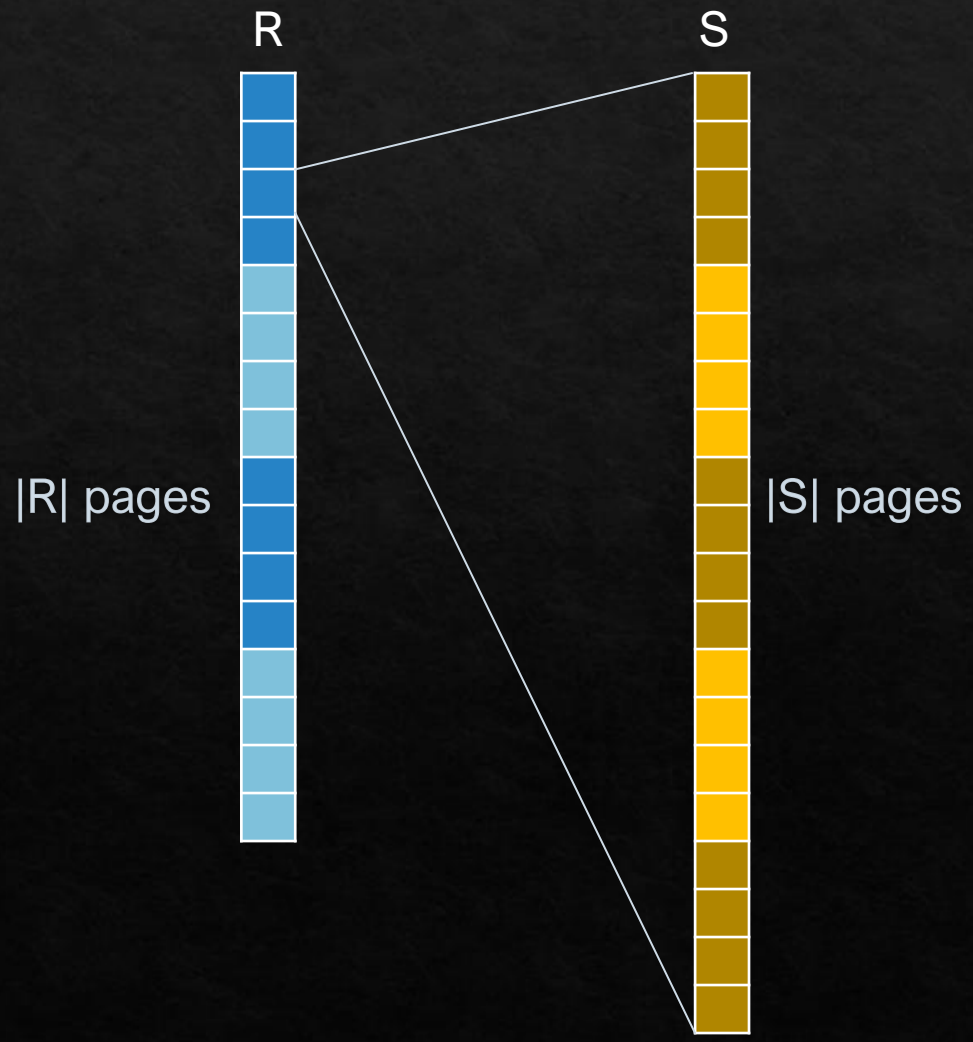
Simple Nested Loop Join

```
foreach tuple r in R do
    foreach tuple s in S do
        if r.sid == s.sid then add <r, s> to result
```



Simple Nested Loop Join

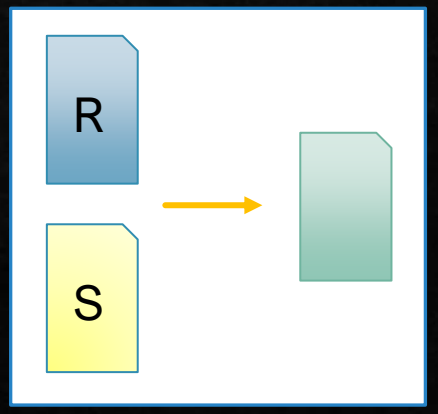
```
foreach tuple r in R do
    foreach tuple s in S do
        if r.sid == s.sid then add <r, s> to result
```

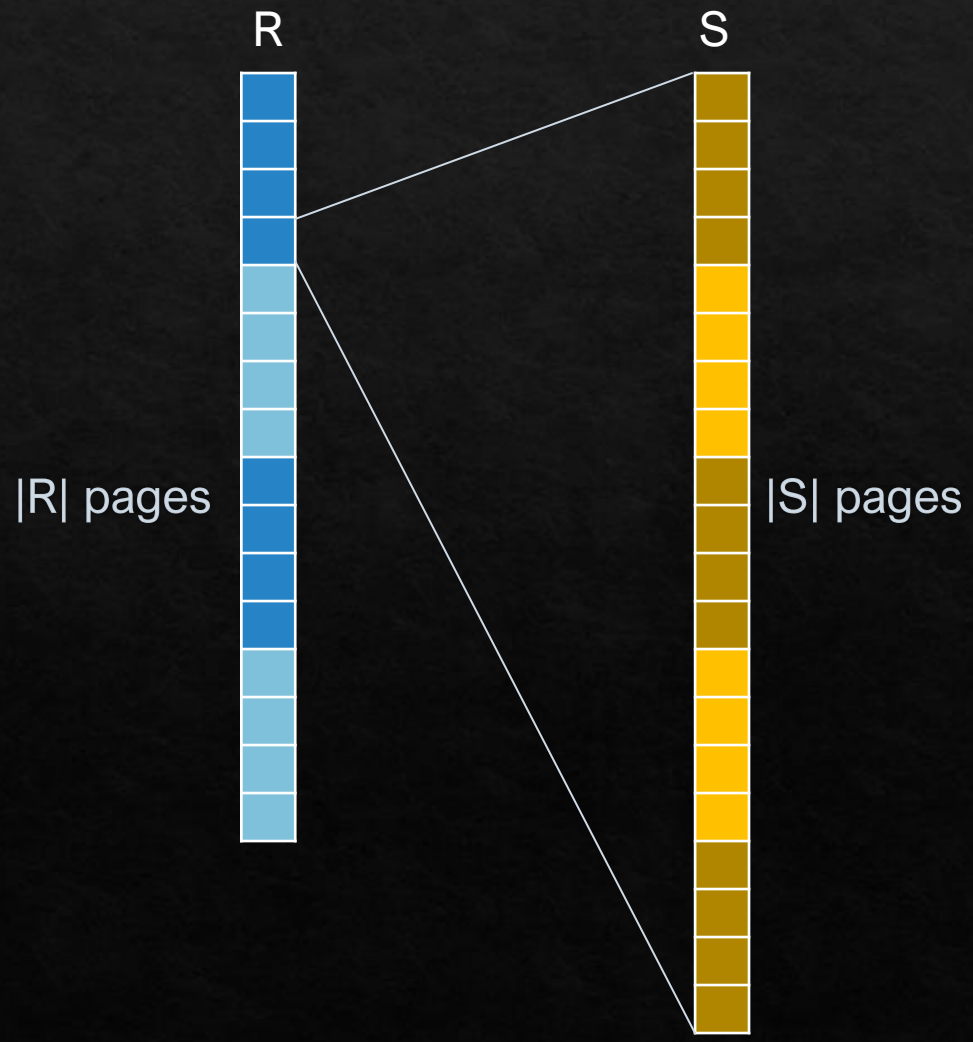


Simple Nested Loop Join

```
foreach tuple r in R do
    foreach tuple s in S do
        if r.sid == s.sid then add <r, s> to result
```

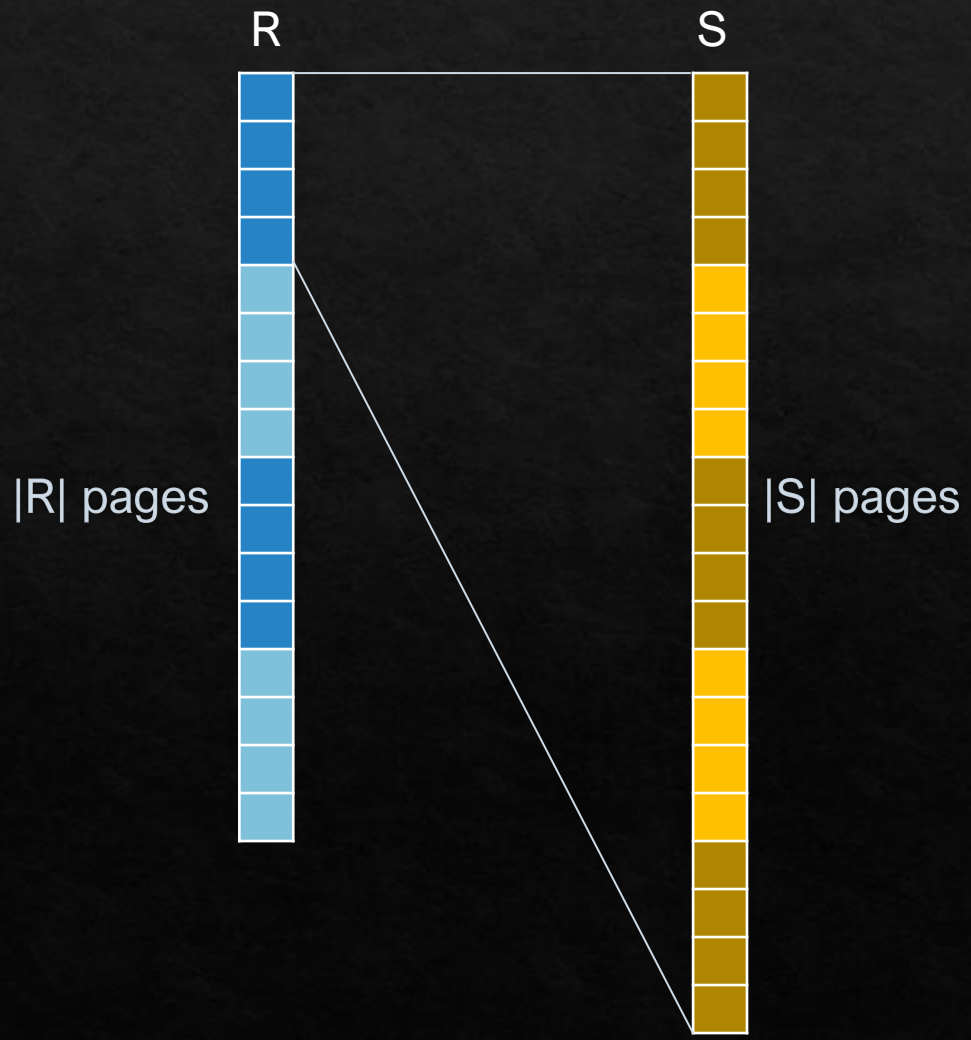
Cost: |R| + ||R|| * |S|
Memory: Only need 3 pages





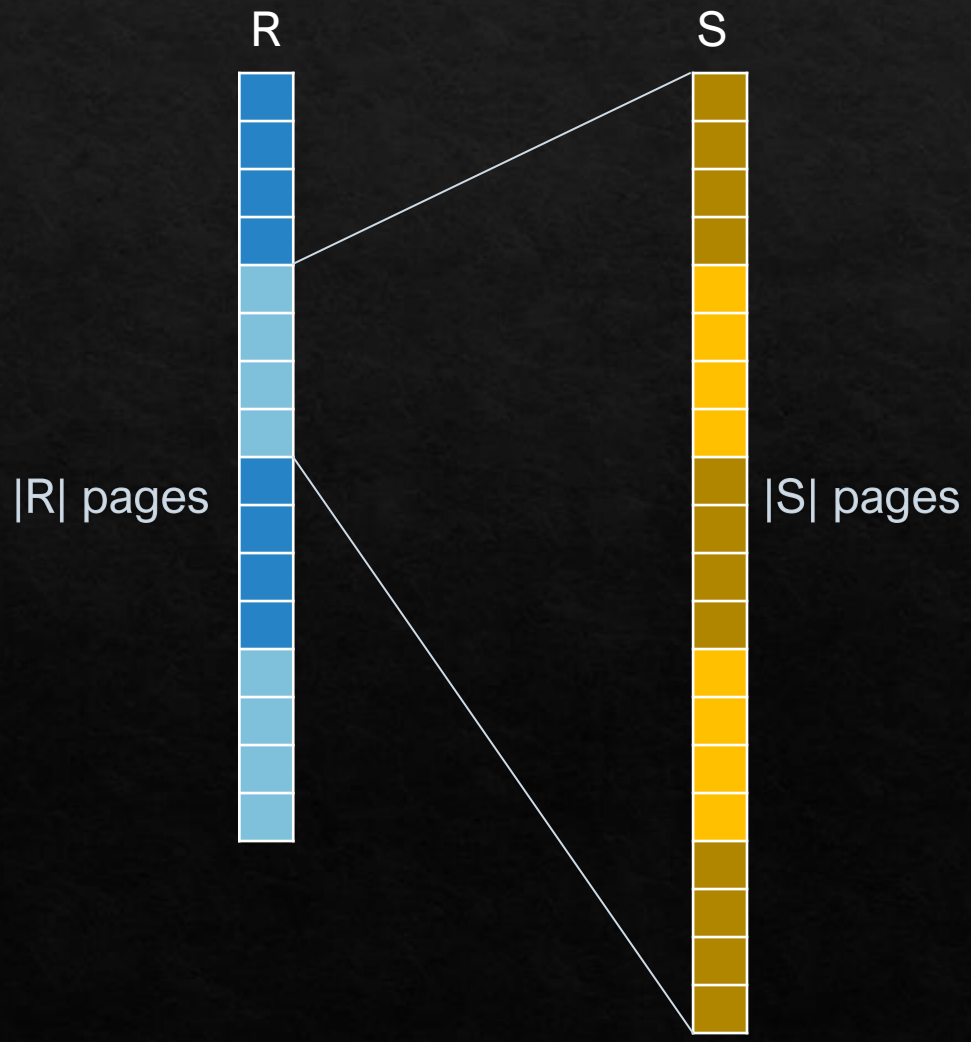
Page Nested Loop Join

```
for each page P_R of R do
    for each page P_S of S do
        foreach tuple r in P_R do
            foreach tuple s in P_S do
                if r.sid == s.sid then add <r, s> to result
```



Page Nested Loop Join

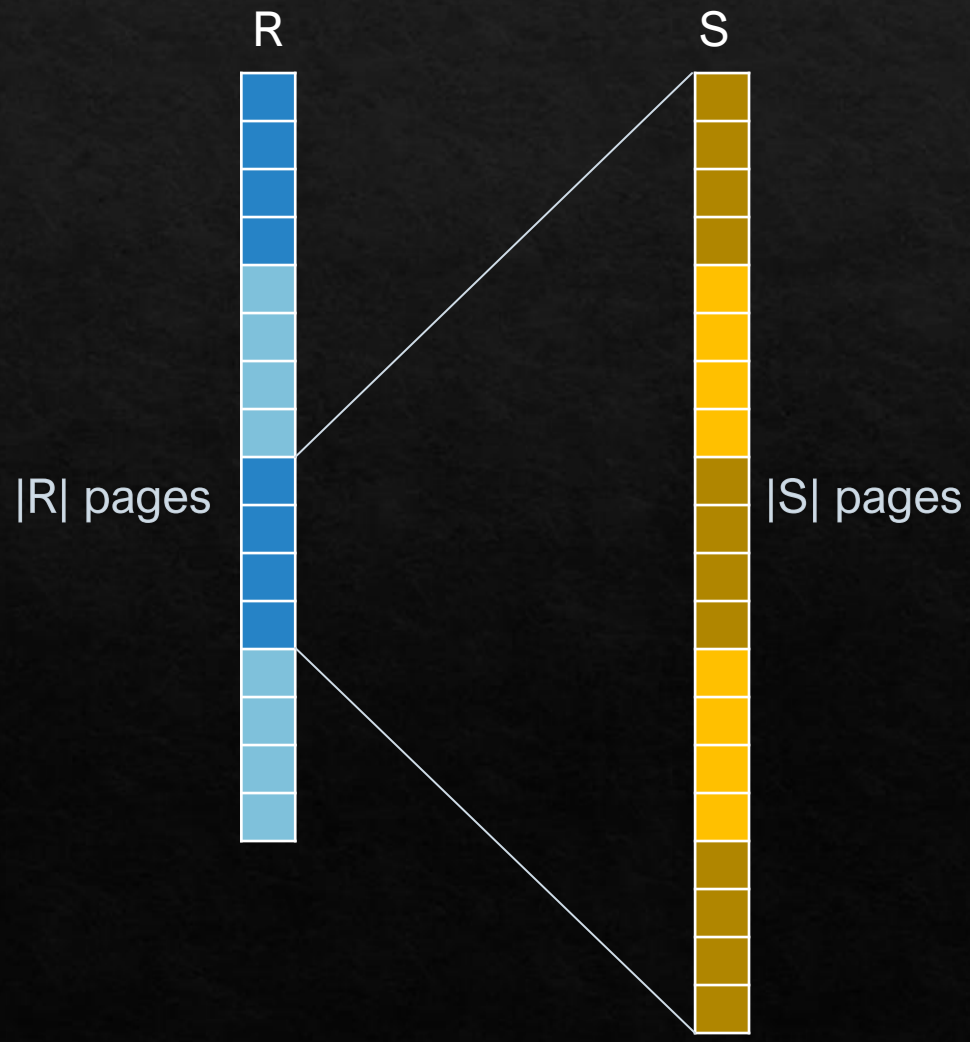
```
for each page P_R of R do
    for each page P_S of S do
        foreach tuple r in P_R do
            foreach tuple s in P_S do
                if r.sid == s.sid then add <r, s> to result
```



Page Nested Loop Join

```
for each page P_R of R do
    for each page P_S of S do
        foreach tuple r in P_R do
            foreach tuple s in P_S do
                if r.sid == s.sid then add <r, s> to result
```

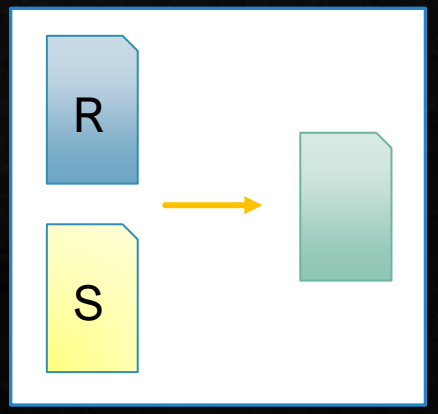
Cost: ?
Memory: ?

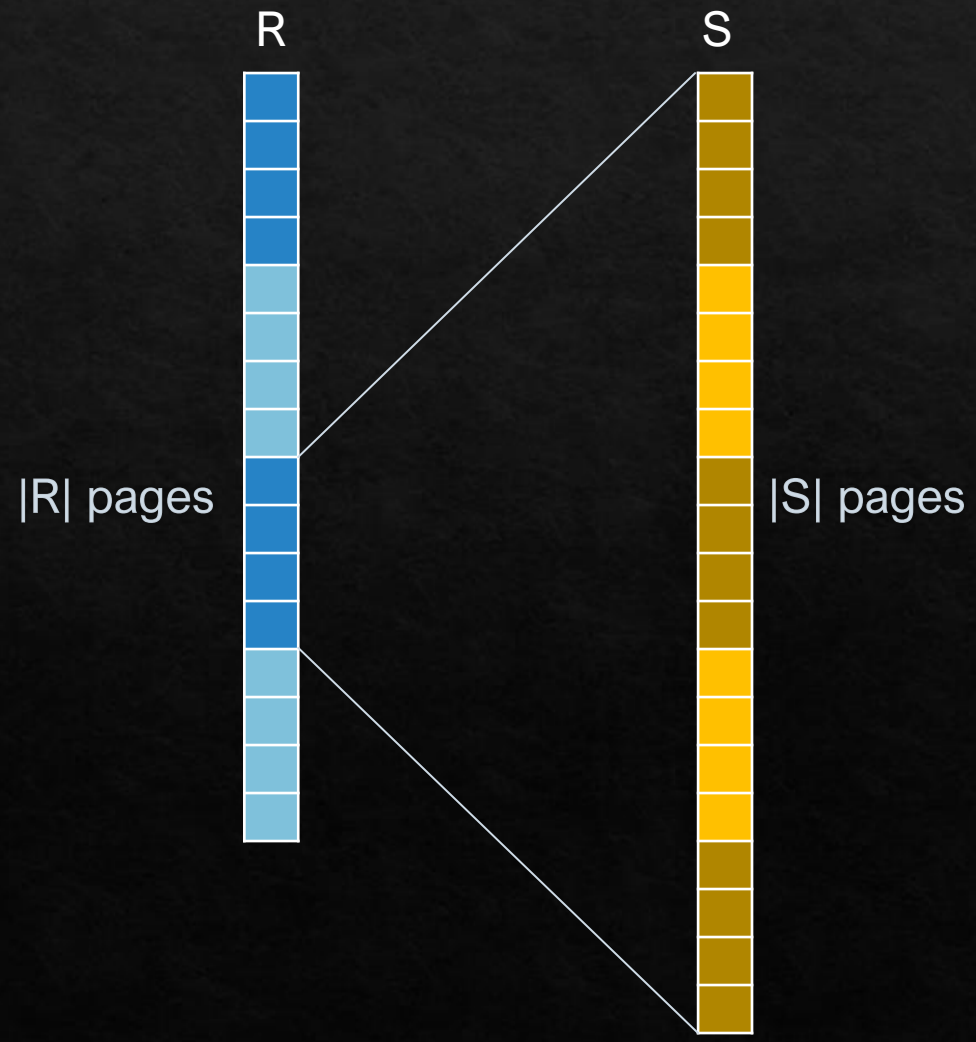


Page Nested Loop Join

for each page P_R of R do
    for each page P_S of S do
        foreach tuple r in P_R do
            foreach tuple s in P_S do
                if r.sid == s.sid then add <r, s> to result

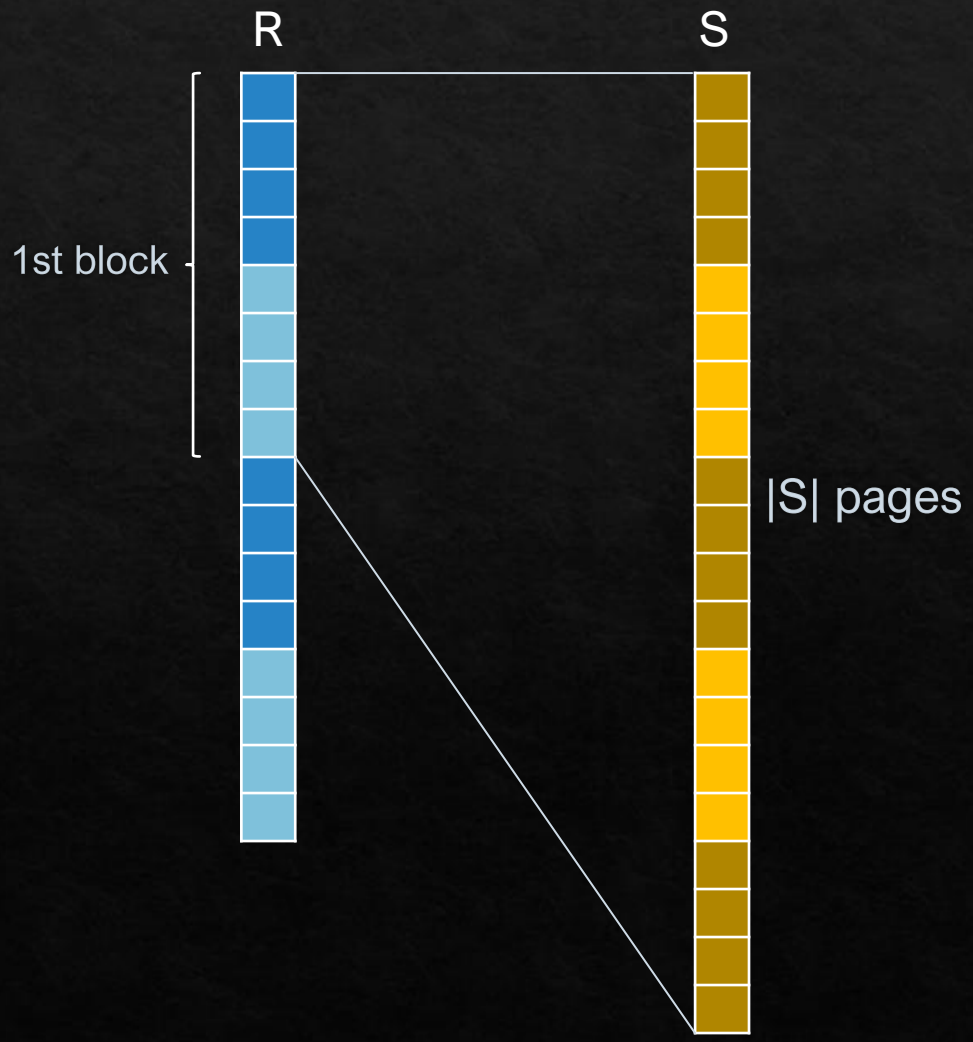
Cost: |R| + |R| * |S|
Memory: Only need 3 pages





Block Nested Loop Join

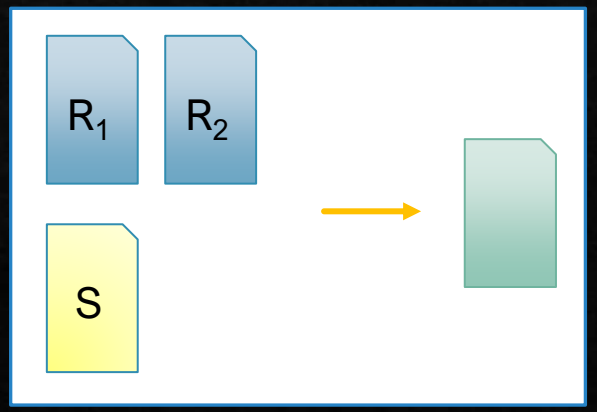
for each **block** B_R of R do
  for each page P_S of S do
    foreach tuple r in B_R do
      foreach tuple s in P_S do
        if r.sid == s.sid then add <r, s> to result



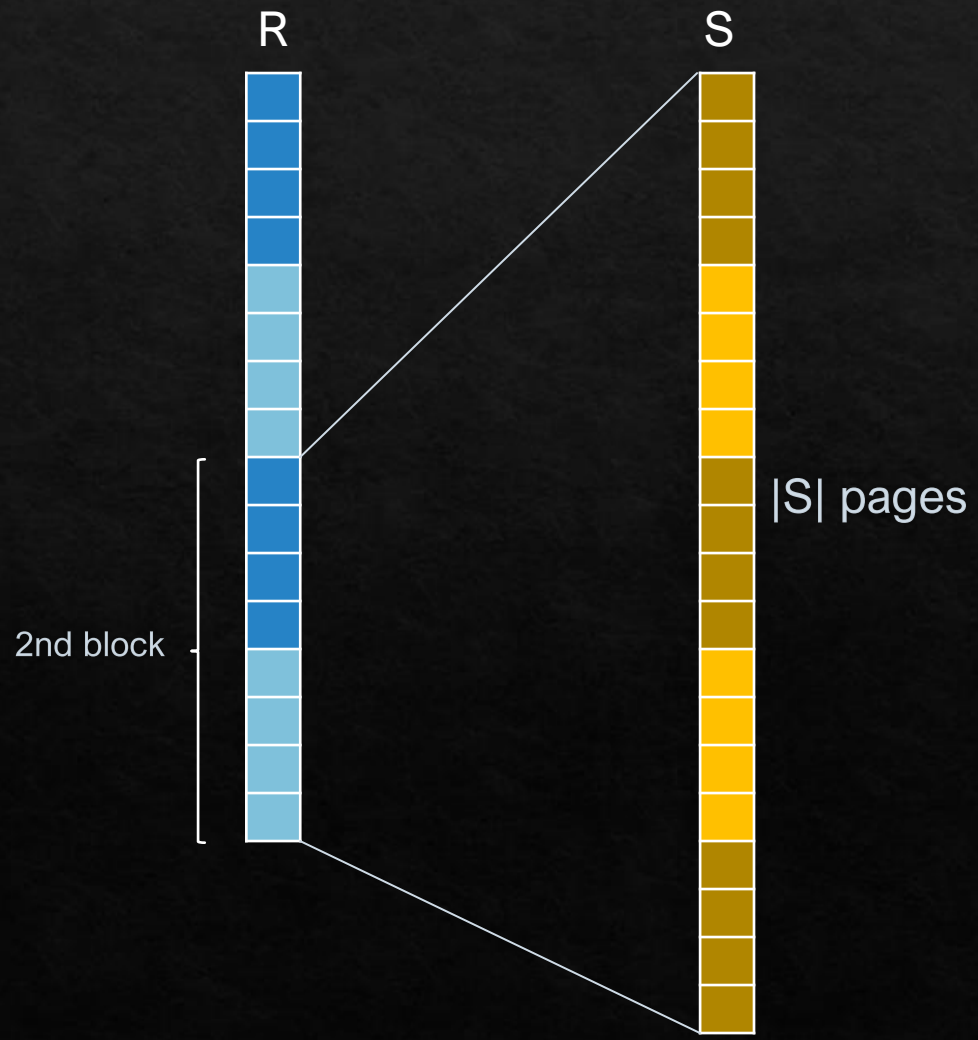
Block Nested Loop Join

for each **block** B_R of R do
  for each page P_S of S do
    foreach tuple r in B_R do
      foreach tuple s in P_S do
        if r.sid == s.sid then add <r, s> to result

Cost: |R| + #outer blocks * |S|
= |R| + [|R| / (block size)] * |S|



If we allocate 1 buffer page for S and 1 for output, block size = B-2

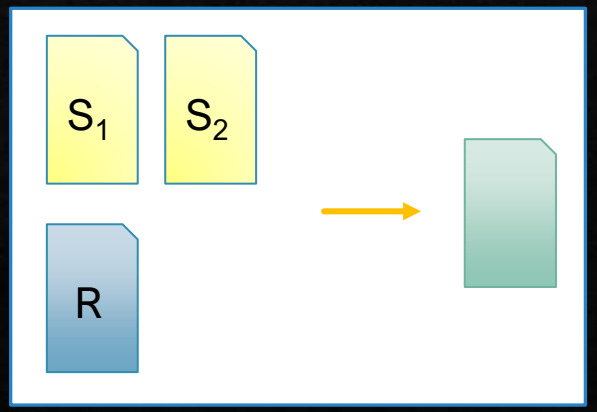


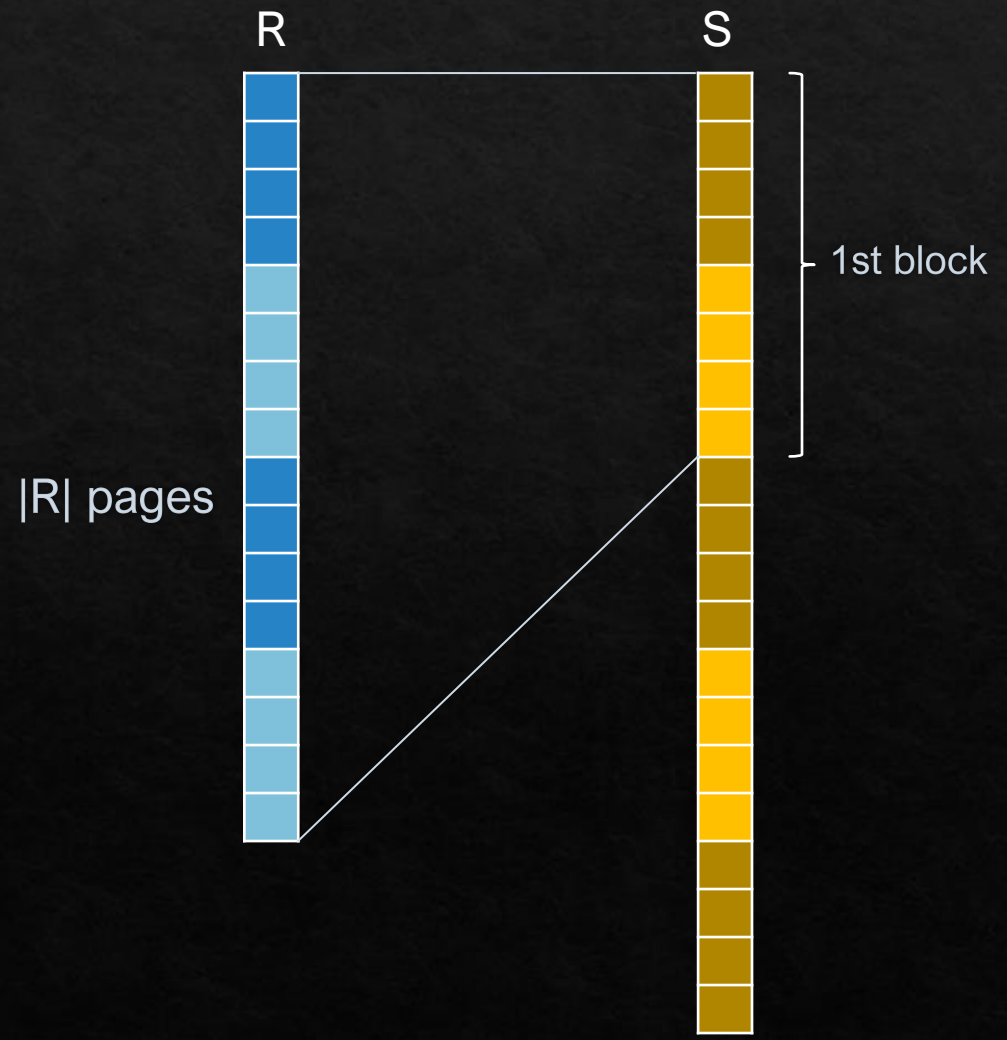
Block Nested Loop Join

for each **block** B_S of S do
  for each page P_R of R do
    foreach tuple s in B_S do
      foreach tuple r in P_R do
        if r.sid == s.sid then add <r, s> to result

We can also use S as the outer relation

Cost: |S| + #outer blocks * |R|
= |S| + [|S| / (block size)] * |R|



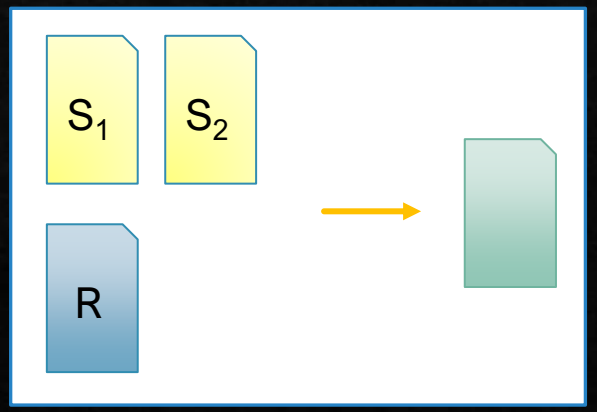


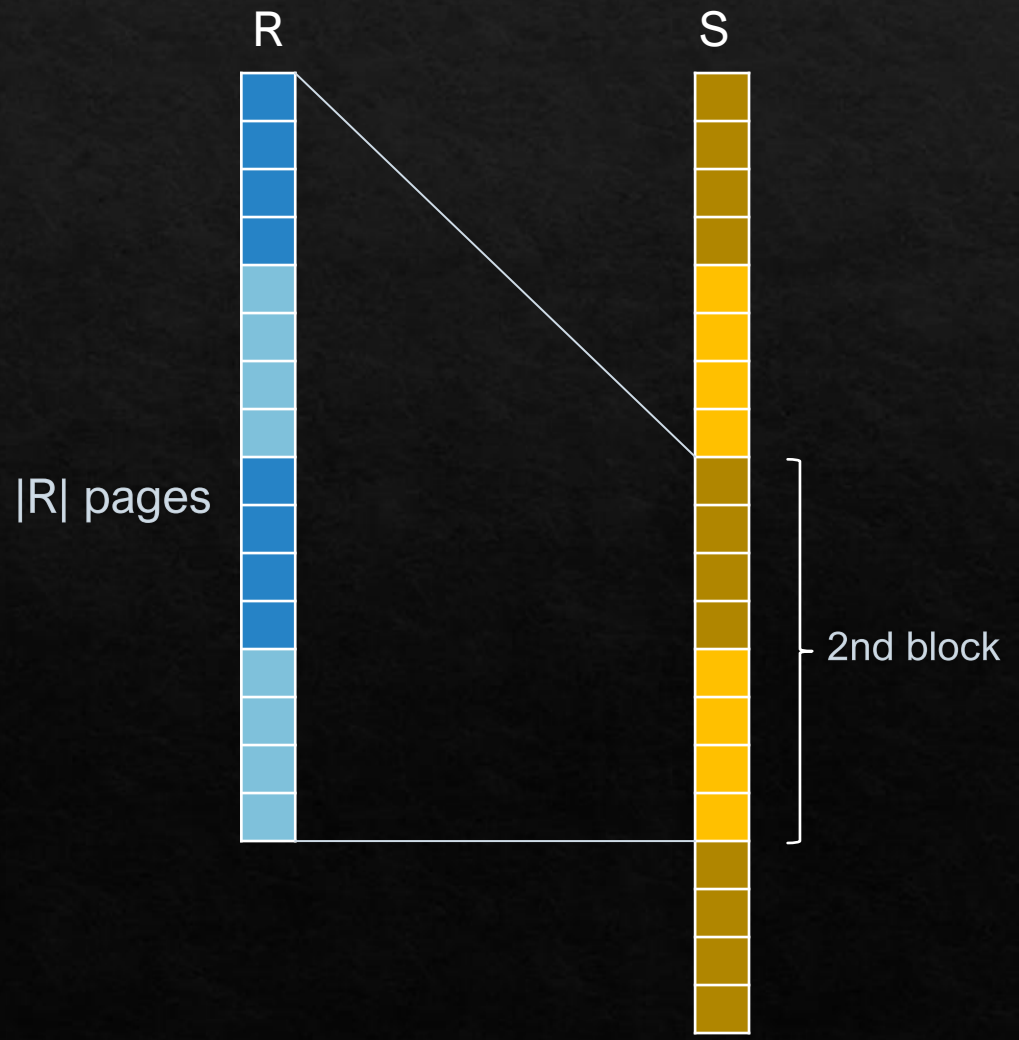
Block Nested Loop Join

We can also use S as the outer relation

```
for each block B_S of S do
    for each page P_R of R do
        foreach tuple s in B_S do
            foreach tuple r in P_R do
                if r.sid == s.sid then add <r, s> to result
```

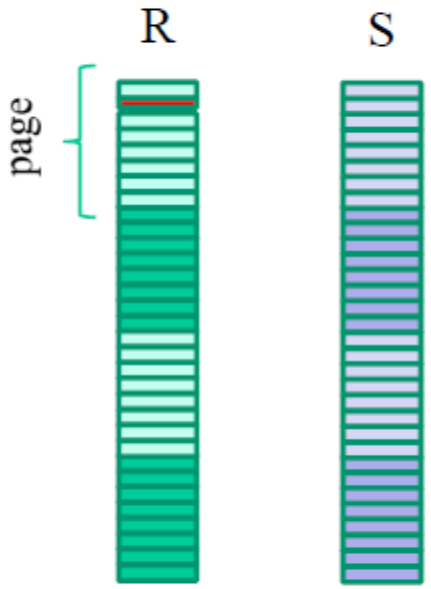
Cost: |S| + #outer blocks * |R|
= |S| + ⌈ |S| / (block size) ⌉ * |R|

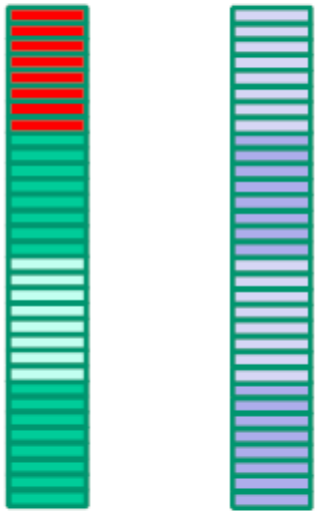


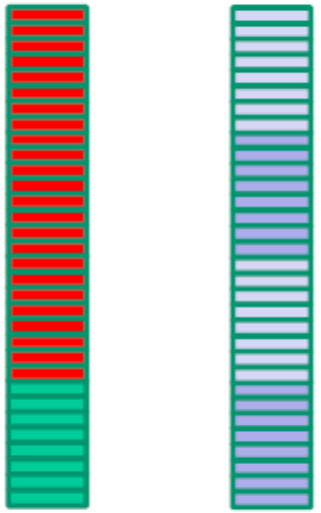


- Motivation: How to better exploit buffer space to minimize number of I/Os?

1. Utilize more buffer space
2. Fewer scans of inner relation







For each R tuple, scan S
(memory size = 3 pages)
Number of iterations of S: ||R||

For each page of R tuples,
scan S (memory = 3 pages)
Number of iterations: |R|

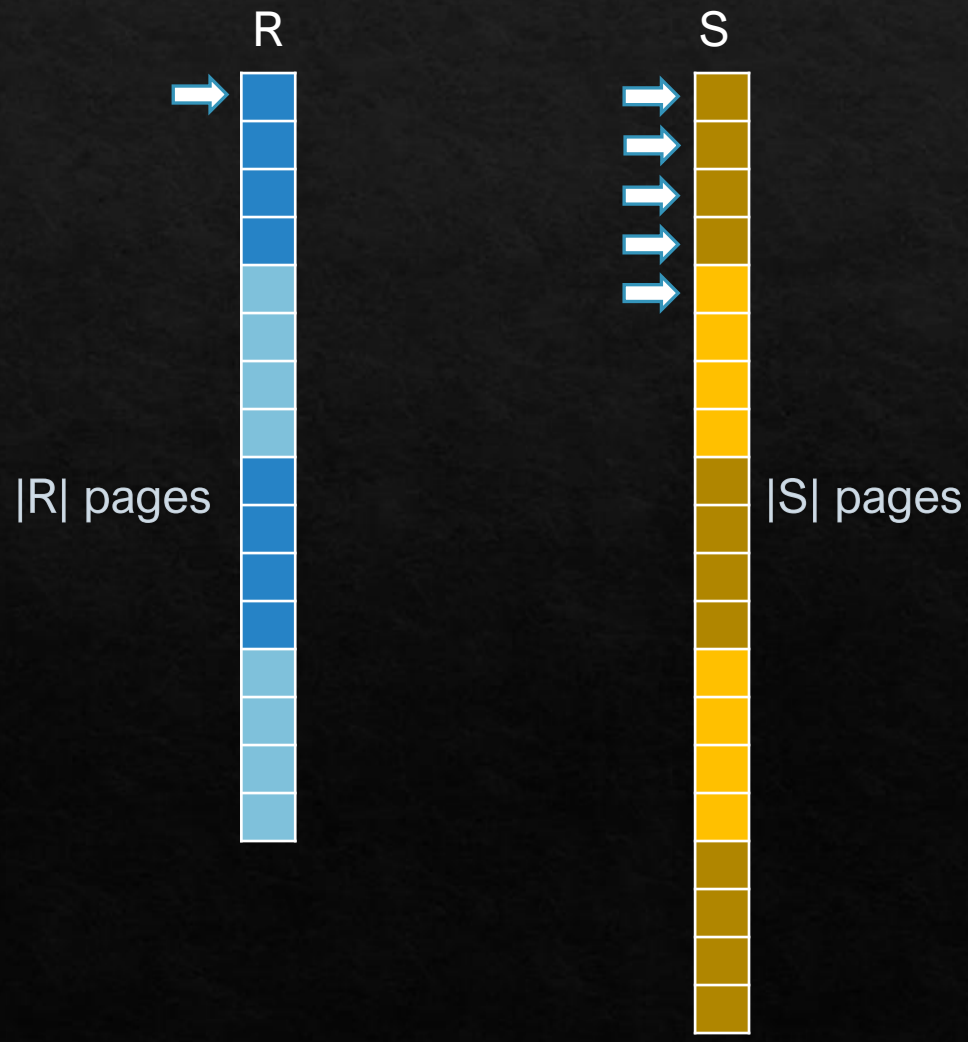
For each block of R tuples,
scan S (memory > 3 pages)
block size (B): buffer size – 2
Number of iterations: |R|/B

Sort-Merge Join

1. Sort R on join column(s)
    1. $2|R| * (\lfloor \log_{B-1}(\lceil |R| / B \rceil) \rfloor + 1)$
2. Sort S on join column(s)
    1. $2|S| * (\lfloor \log_{B-1}(\lceil |S| / B \rceil) \rfloor + 1)$

Sort-Merge Join

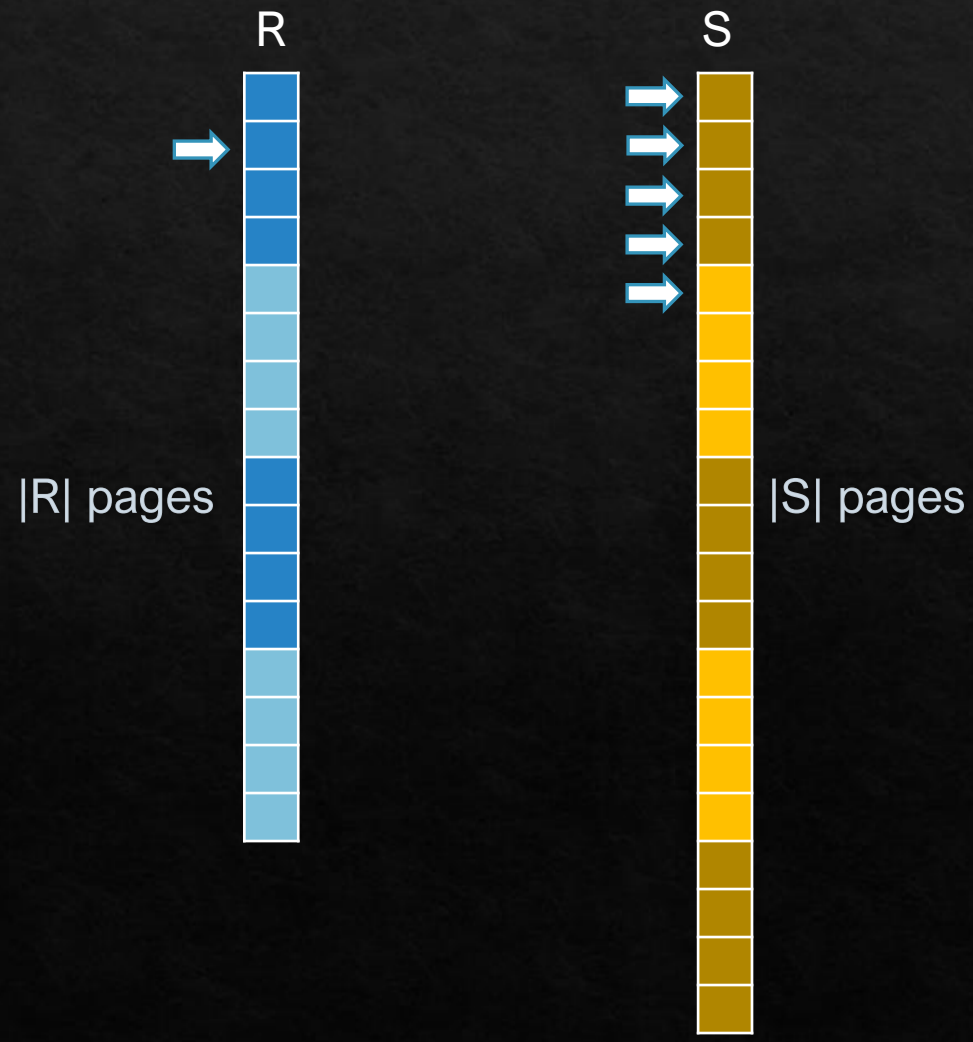
Merging with multiple matches



Sort-Merge Join

Merging with multiple matches

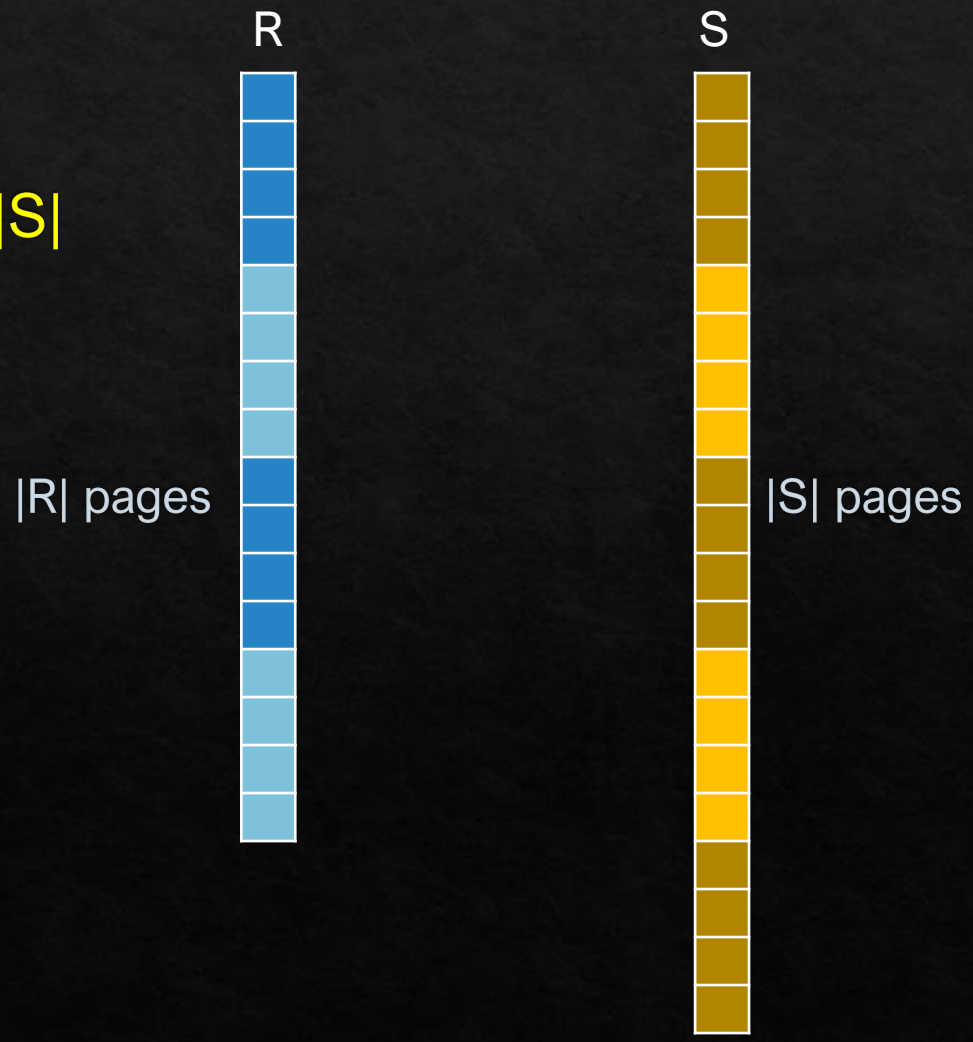
- May have to scan S partition multiple times due to multiple matches of R



Sort-Merge Join

Merging Cost

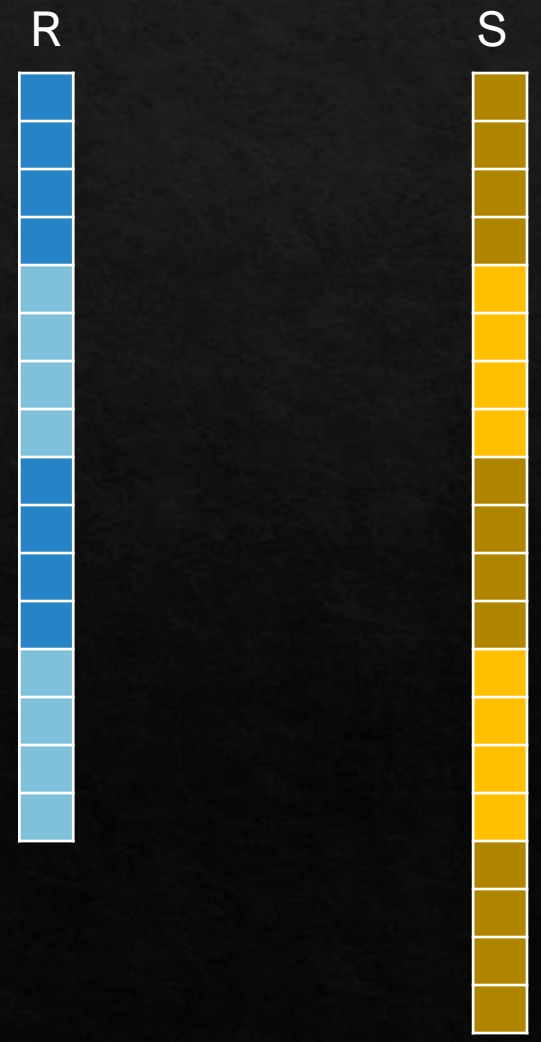
1. Each S partition scanned at most once: |R| + |S|



Sort-Merge Join

Merging Cost

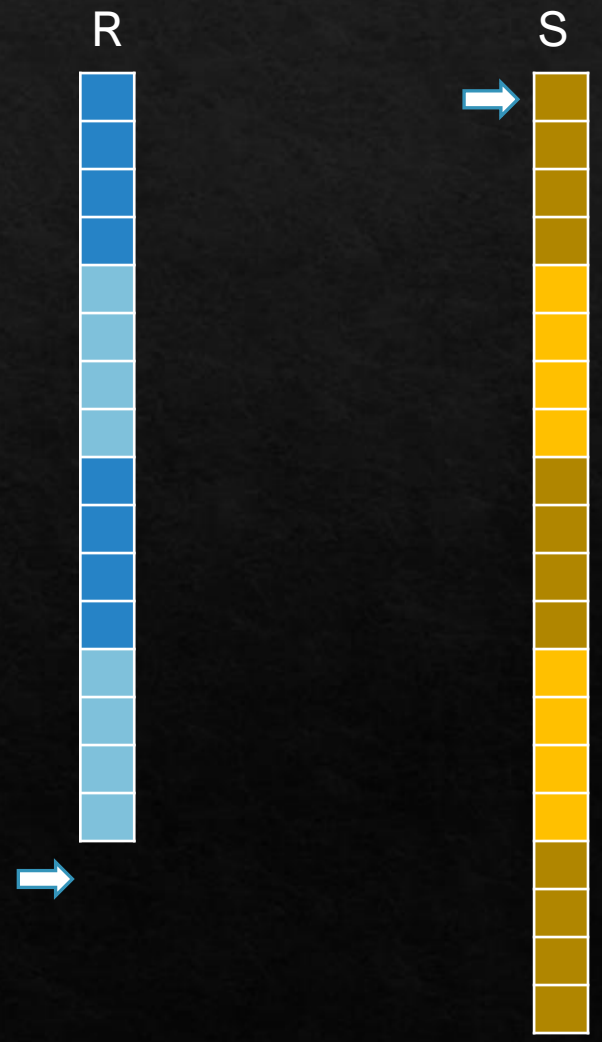
1. Each S partition scanned at most once: |R| + |S|
2. Best case: |R| (or |S|)
    1. When does this occur?



Sort-Merge Join

Merging Cost

1. Each S partition scanned at most once: |R| + |S|
2. Best case: |R| (or |S|) + 1
    1. Largest value in R is smaller than smallest value in S



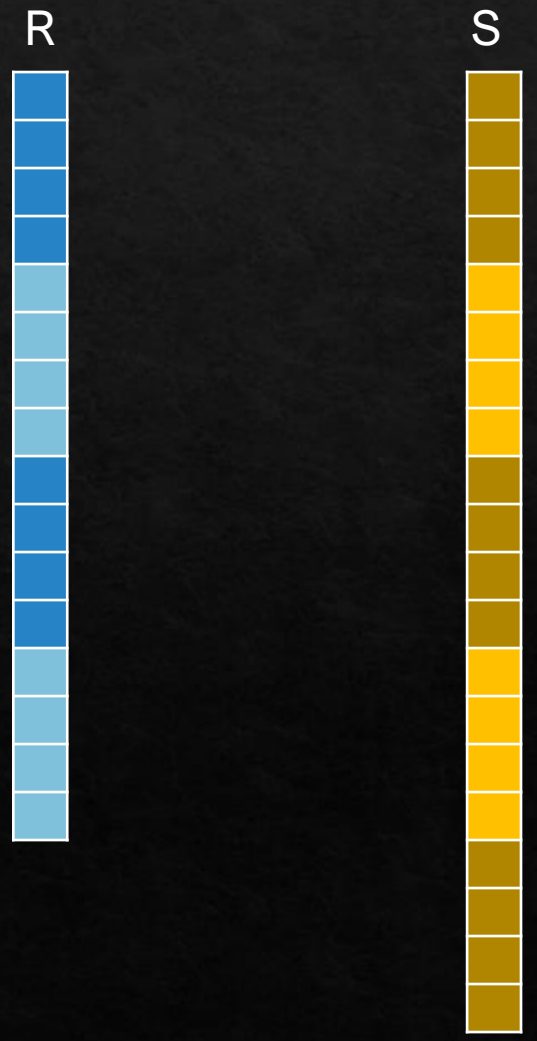
Sort-Merge Join

Merging Cost

1. Each S partition scanned at most once: |R| + |S|
2. Best case: |R| (or |S|) + 1
    1. Largest value in R is smaller than smallest value in S
3. Worst case: |R| + |R| * |S|
    1. Each tuple in R matches every tuple in S

Total Cost

1. Merging Cost + $2|R| * (\lceil \log_{B-1}(\lceil |R| / B \rceil) \rceil + 1)$
   $+ 2|S| * (\lceil \log_{B-1}(\lceil |S| / B \rceil) \rceil + 1)$



Hash Join

Two Phases

- Partition Phase
  - Partition R on h()
  - Partition S on h()
- Join Phase
  - Read one partition of R
  - Hash the entries in this partition using $h_2()$
  - Scan the corresponding partition of S and match entries using $h_2()$

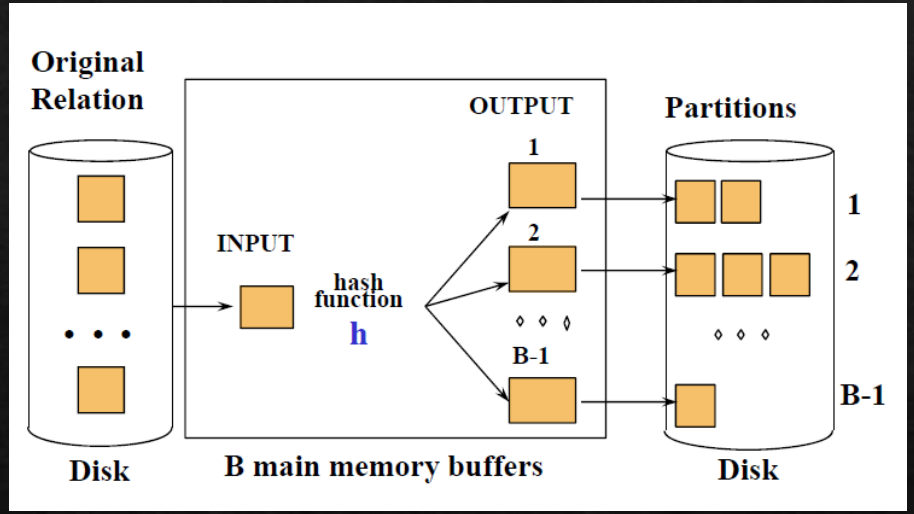
Example if h(x) = x mod 3,
Entries 6, 9, 12 will be in the same partition

To distinguish 6, 9, 12 apply
$h_2(x) = x \bmod 5$
6 → 1
9 → 4
12 → 2

Hash Join

Two Phases

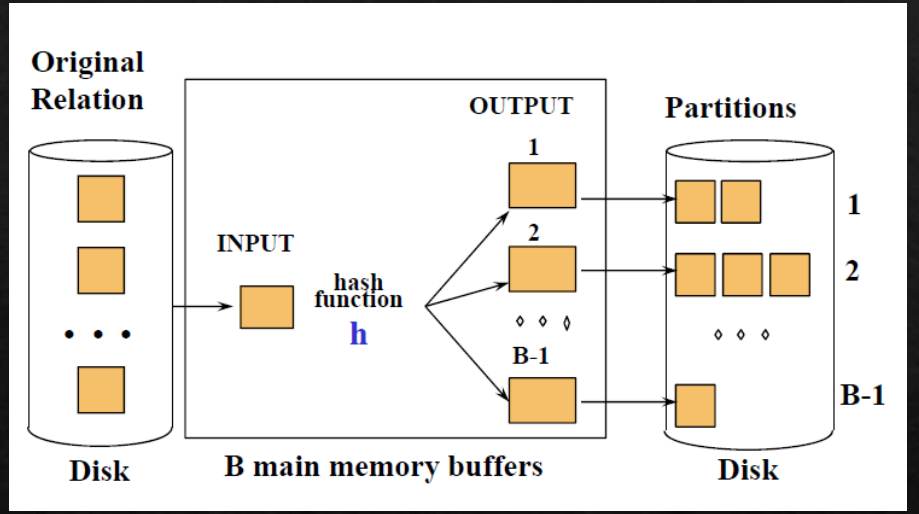
- Partition Phase.
  - Partition R on h(). **Cost = 2|R|**
  - Partition S on h(). **Cost = 2|S|**
- Join Phase
  - Read one partition of R
  - Hash the entries in this partition using $h_2()$
  - Scan the corresponding partition of S and match entries using $h_2()$

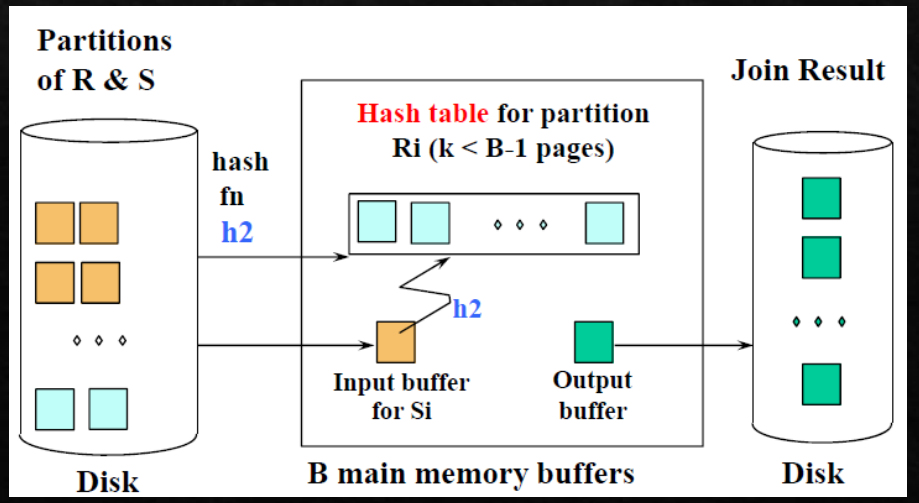


Hash Join

Two Phases

- Partition Phase.
  - Partition R on h(). **Cost = 2|R|**
  - Partition S on h(). **Cost = 2|S|**
- Join Phase.
  - Read one partition of R
  - Hash the entries in this partition using $h_2()$
  - Scan the corresponding partition of S and match entries using $h_2()$
  - **Cost = |R| + |S|**



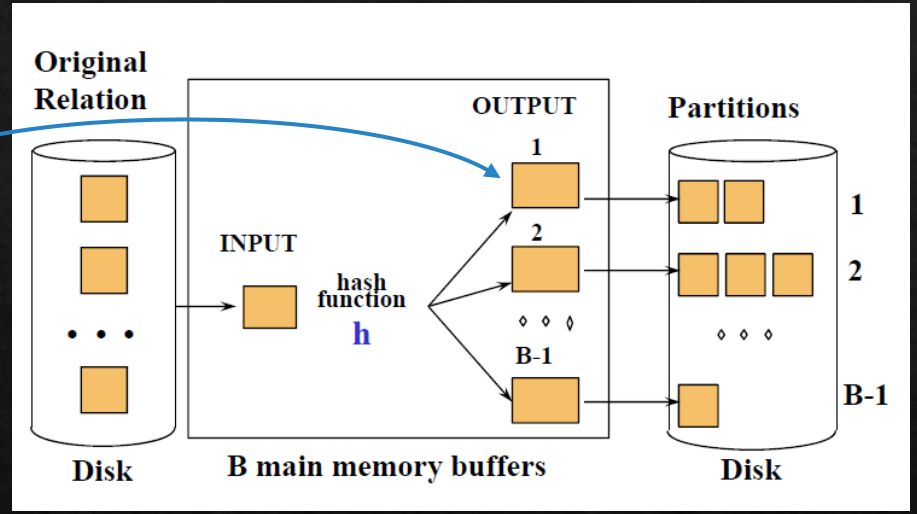


Hash Join

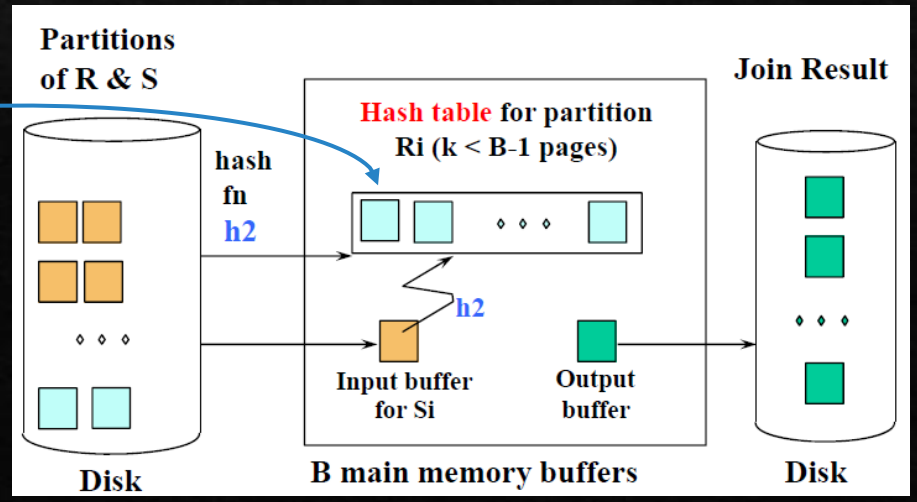
Two Phases

- Partition Phase.
  - Partition R on h(). **Cost = 2|R|**
  - Partition S on h(). **Cost = 2|S|**
- Join Phase.
  - Read one partition of R
  - Hash the entries in this partition using $h_2()$
  - Scan the corresponding partition of S and match entries using $h_2()$
  - **Cost = |R| + |S|**

#partitions <= B-1



#pages per partition <= B-2



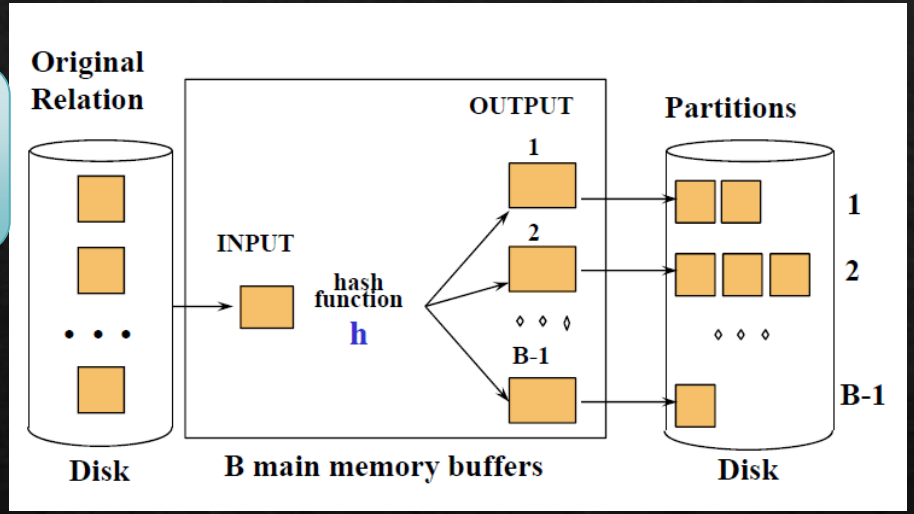
Hash Join

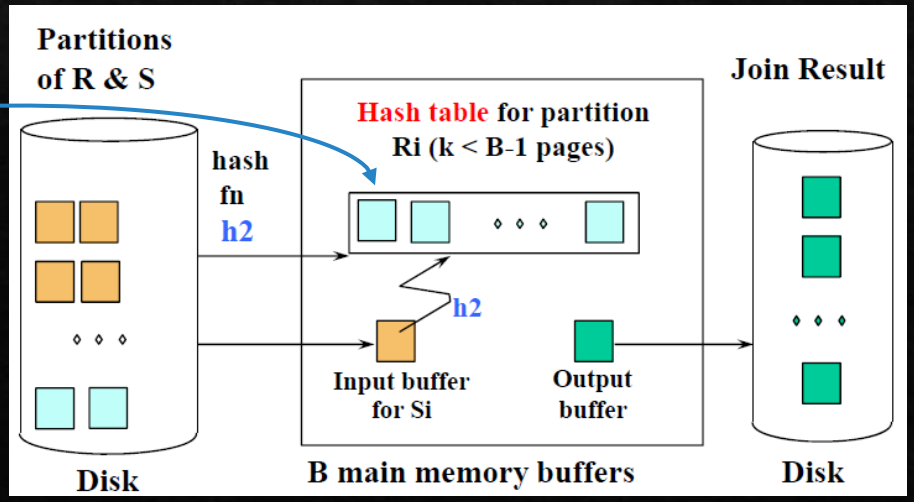
(Size of rel) / (#partition) <= size of partition
M / (B-1) <= B-2
B >= sqrt(M)

Two Phases

- Partition Phase.
  - Partition R on h(). **Cost = 2|R|**
  - Partition S on h(). **Cost = 2|S|**
- Join Phase.
  - Read one partition of R
  - Hash the entries in this partition using $h_2()$
  - Scan the corresponding partition of S and match entries using $h_2()$
  - **Cost = |R| + |S|**

If B < sqrt(M), perform recursive partitioning. Each addition round of partitioning costs 2|R| + 2|S|





Index Nested Loop Join

foreach tuple r in R do
    search index of S on sid using $S_{search\text{-}key}$ = r.sid
        for each matching key
            retrieve s; add (r, s) to result

Cost: |R| + ||R|| * (cost of finding matching S tuples)

Cost of finding matching S tuples

B^+ tree (format 2)

- Traverse index + cost of fetching S tuples from data page
  - Clustered Index: 1 I/O (because all matching tuples are usually found in the same page)
  - Unclustered Index: 1 I/O per match

Hash Index (format 2)

- 1.2 (average #overflow pages) + cost of fetching S tuples (1 I/O each)

Summary

Method	Cost
Simple Nested Loop Join	|R| + ||R|| * |S|
Page Nested Loop Join	|R| + |R| * |S|
Block Nested Loop Join	|R| + ⌈ |R| / (block size) ⌉ * |S|
Index Nested Loop Join	|R| + ||R|| * (cost of finding matching S tuples)
Sort Merge Join	Merging Cost + 2|R| * (⌈ logB-1(⌈|R| / B⌉) ⌉ + 1) + 2|S| * (⌈ logB-1(⌈|S| / B⌉) ⌉ + 1) Merging Cost: |R| + |S| (if at most one match per item)
GRACE Hash Join	Partition Phase: 2|R|+ 2|S| (per round) Join Phase: |R| + |S|

Tutorial Questions

Q1

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

What is the total number of **random I/Os** and **sequential I/Os**?

Q1

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

Q1

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block

Q1

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block
- Number of matching S tuples per R tuple = (#tuples of S) / (#distinct values of S.B) = 100,000/40,000 = 2.5

Q1

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block
- Number of matching S tuples per R tuple = (#tuples of S) / (#distinct values of S.B) = 100,000/40,000 = 2.5
- ||R|| * 2.5 random I/O (since B+ tree index on S.B is non-clustered)

Q1

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block
- Number of matching S tuples per R tuple = (#tuples of S) / (#distinct values of S.B) = 100,000/40,000 = 2.5
- ||R|| * 2.5 random I/O (since B+ tree index on S.B is non-clustered)
- No need anymore I/O for the last step since we already have the tuples we need for joining

Q1

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
            For each of these tuples s
                If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block
- Number of matching S tuples per R tuple = (#tuples of S) / (#distinct values of S.B) = 100,000/40,000 = 2.5
- ||R|| * 2.5 random I/O (since B+ tree index on S.B is non-clustered, and no additional I/O to retrieve nodes)
- Total = (|R| + ||R|| * 2.5) random I/O

  = (1000 + 500,000 * 2.5)

  = 1,251,000 random I/Os
- Total access time = **1,251,000 t_r**

Q2

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on C for S to retrieve all the tuples s of S such that s.C = r.C
            For each of these tuples s
                If s.B = r.B, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block

Q2

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on C for S to retrieve all the tuples s of S such that s.C = r.C
            For each of these tuples s
                If s.B = r.B, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block
- Number of matching S tuples per R tuple = (#tuples of S) / (#distinct values of S.C) = 100,000/40 = 2500 tuples
- But since this is clustered index on C, it will only take |S|/(#distinct values of S.C) = 1000 / 40 = 25 I/O (each value of S.C span 25 blocks)
- ||R|| * (1 random I/O + 24 sequential I/O)
  - Since B+ tree is clustered

Q2

```
SELECT *
FROM R, S
WHERE (R.B=S.B) AND (R.C=S.C)
```

```
For every block B of R, retrieve using the clustered index on A for R
    For every tuple r of B
        Use the index on C for S to retrieve all the tuples s of S such that s.C = r.C
            For each of these tuples s
                If s.B = r.B, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
```

- R(A,B,C) and S(B,C,D,E)
- Relation R
  - Clustered, unique B+ tree index on A (in memory)
- Relation S
  - Non-clustered, non-unique B+ tree index on B (in memory)
  - Clustered, non-unique B+ tree index on C (in memory)
- All R.B are found in S.B
- All R.C are found in S.C
- Tuples of S that agree on attribute C are stored sequentially on adjacent blocks on disk (if they cannot fit into one block)
- 500 tuples of R per block
- 500,000 tuples of R in total
- 100 tuples of S per block
- 100,000 tuples of S in total
- S.B has 40,000 distinct values, S.C has 40 distinct values

#blocks (#pages):
|R| = 500,000 / 500 = 1000
|S| = 100,000 / 100 = 1000

What is the total number of **random I/Os** and **sequential I/Os**?

Let t_r = time for random I/O, t_s = time for sequential I/O

- |R| random I/O to retrieve each R block
- Number of matching S tuples per R tuple = (#tuples of S) / (#distinct values of S.C) = 100,000/40 = 2500
- But since this is clustered index on C, it will only take |S|/(#distinct values of S.C) = 1000 / 40 = 25 I/O
- ||R|| * (1 random I/O + 24 sequential I/O)
  - Since B+ tree is clustered
- Total = (|R| + ||R|| * 1) random I/O + (||R|| * 24) seq I/O
  = (1000 + 500,000) rand I/O + (500,000 * 24) seq I/O
  = 501,000 random I/O + 12,000,000 sequential I/O
- Total access time = **501,000 t_r + 12,000,000 t_s**

Q2

- Q1: **1,251,000 t_r**
- Q2: **501,000 t_r + 12,000,000 t_s**

- **Q1** has less overall I/Os than **Q2**
- But Q2 has more sequential I/O and less random I/O than Q1

- This holds **1,251,000 t_r < 501,000 t_r + 12,000,000 t_s**
  - If $t_r < 16\ t_s$

Q3

- Ignore cost of writing out result
- 3-level B+ trees, format 2
- R contains 1,000,000 tuples and has 20 tuples per page.
- S contains 2,000,000 tuples and has 40 tuples per page.
- S.b is the primary key for S.
- R.a is a foreign key that references S.b
- 100 buffer pages are available (inclusive of input/output buffers)

Q3

- If neither relation has any indexes built on it
  - What is the cost of joining R and S using block nested loop join?

Q3

- If neither relation has any indexes built on it
  - What is the cost of joining R and S using block nested loop join?

  ||R|| = 1,000,000; |R| = 50,000

  ||S|| = 2,000,000; |S| = 50,000
  - 100 buffer pages available: 1 for inner relation, 1 for output, block size = 98
  - Since |R| = |S|, no difference which relation is used as the outer relation.

  Cost = |R| + #blocks * |S|

  = |R| + ⌈ |R| / (block size) ⌉ * |S|

  = 50,000 + ⌈ 50,000 / 98 ⌉ * 50,000

  = **25,600,000 I/O**

Q3

- If neither relation has any indexes built on it
  - What is the cost of joining R and S using a sort-merge join algorithm? What would the cost be if both relations are already sorted on the join attributes?

Q3

- If neither relation has any indexes built on it
  - What is the cost of joining R and S using a sort-merge join algorithm? What would the cost be if both relations are already sorted on the join attributes?

||R|| = 1,000,000; |R| = 50,000

||S|| = 2,000,000; |S| = 50,000

Sorting cost = $2|R| * (\lceil \log_{B-1}(\lceil |R| / B \rceil) \rceil + 1) + 2|S| * (\lceil \log_{B-1}(\lceil |S| / B \rceil) \rceil + 1)$

= 600,000 I/O

Merging cost = |R| + |S| = 100,000 I/O (also the cost if already sorted)

Total cost = **700,000 I/O**

Q3

- If neither relation has any indexes built on it
  - What is the cost of joining R and S using a hash join algorithm?

Q3

- If neither relation has any indexes built on it
  - What is the cost of joining R and S using a hash join algorithm?

||R|| = 1,000,000; |R| = 50,000

||S|| = 2,000,000; |S| = 50,000

  - Since B = 100 < sqrt(50,000), we need recursive partitioning
  - In the second pass, there will be 50,000/99 = 506 pages per partition
  - Now, B >= sqrt(506)

Partitioning cost = 2 * (2|R| + 2|S|)= 400,000 I/O

Joining cost = |R| + |S| = 100,000 I/O

Total cost = **500,000 I/O**

Recall:
B >= sqrt(|R|) to partition in one pass

If B < sqrt(M), perform recursive partitioning. Each addition round of partitioning costs 2|R| + 2|S|

Q3

- If a clustered B+-tree exists on S.b, would nested-index algorithm be preferred? What are the worst and best case scenarios?

  ||R|| = 1,000,000; |R| = 50,000

  ||S|| = 2,000,000; |S| = 50,000

  - Format 2, 3-level B+ tree, (we do not assume it is in memory this time)

Q3

- If a clustered B+-tree exists on S.b, would nested-index algorithm be preferred? What are the worst and best case scenarios?

  ||R|| = 1,000,000; |R| = 50,000

  ||S|| = 2,000,000; |S| = 50,000

  - Format 2, 3-level B+ tree, (we do not assume it is in memory this time)
  - Worst case (none of the S records can be buffered)
    - |R| + ||R|| * (3 + 1) = **4,050,000 I/O**

Q3

- If a clustered B+-tree exists on S.b, would nested-index algorithm be preferred? What are the worst and best case scenarios?

  ||R|| = 1,000,000; |R| = 50,000

  ||S|| = 2,000,000; |S| = 50,000

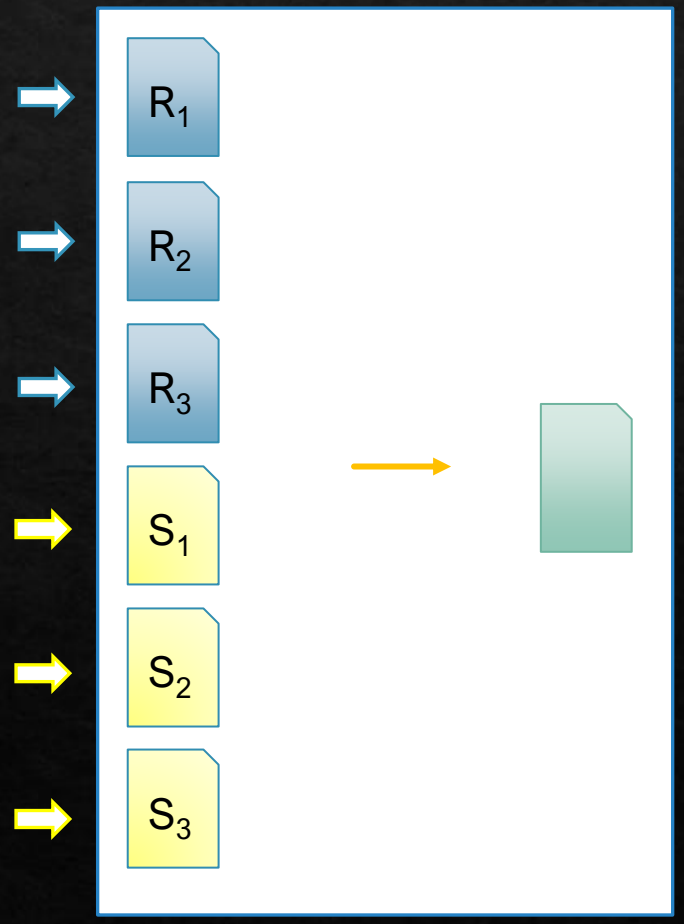
  - Format 2, 3-level B+ tree, (we do not assume it is in memory this time)
  - Worst case (none of the S records can be buffered)
    - |R| + ||R|| * (3 + 1) = **4,050,000 I/O**
  - Best case (only one value for R.a, so only fetch S.b value once and keep it in buffer to join all R tuples):
    - |R| + (3 + 1) = **50,004 I/O**

Q4

The sort-merge join can be refined by combining the **merging phases** in the **sorting** of R and S with the merging required for the join. In other words, when sorting R (and S), we do not merge the runs into one single sorted run; instead, the join is performed on a set of sorted runs of R and S, rather than on a single sorted run of R and S. Describe and discuss a sort-merge join with this refinement.

Q4

- Generate floor((B-1)/2) number of sorted runs for R and S each
- Allocate 1 buffer page for each sorted run, and 1 buffer page for output
- All matching tuples between R and S will be seen at the same time and written to output if found
- Save cost from reading/writing out intermediate sorted runs



Summary

Method	Cost
Simple Nested Loop Join	|R| + ||R|| * |S|
Page Nested Loop Join	|R| + |R| * |S|
Block Nested Loop Join	|R| + ⌈ |R| / (block size) ⌉ * |S|
Index Nested Loop Join	|R| + ||R|| * (cost of finding matching S tuples)
Sort Merge Join	Merging Cost + 2|R| * (⌈ logB-1(⌈|R| / B⌉) ⌉ + 1) + 2|S| * (⌈ logB-1(⌈|S| / B⌉) ⌉ + 1) Merging Cost: |R| + |S| (if at most one match per item)
GRACE Hash Join	Partition Phase: 2|R|+ 2|S| (per round) Join Phase: |R| + |S|

Extra Question

Consider two relations R and S, with 1000 and 500 blocks, respectively, and memory M = 101. Calculate the total number of disk I/O's needed if there are only **two distinct values** in the **join column**, each appearing in half the tuples of R and half the tuples of S.

- Perform **two** nested-loop joins of 500 and 250 blocks, respectively, using 101 blocks of memory.

  might need a ceiling here

  - takes 250 + 500*250/100 = 1500 disk I/O's, so **two** of them takes 3000.
- Cost of sorting the two relations
  - four disk I/O (2 passes of Read+Write) per block of the relations, which is 6000
- The total disk I/O cost is 9000 I/O

This solution assumes you know there are duplicates; otherwise, the cost can be much higher.