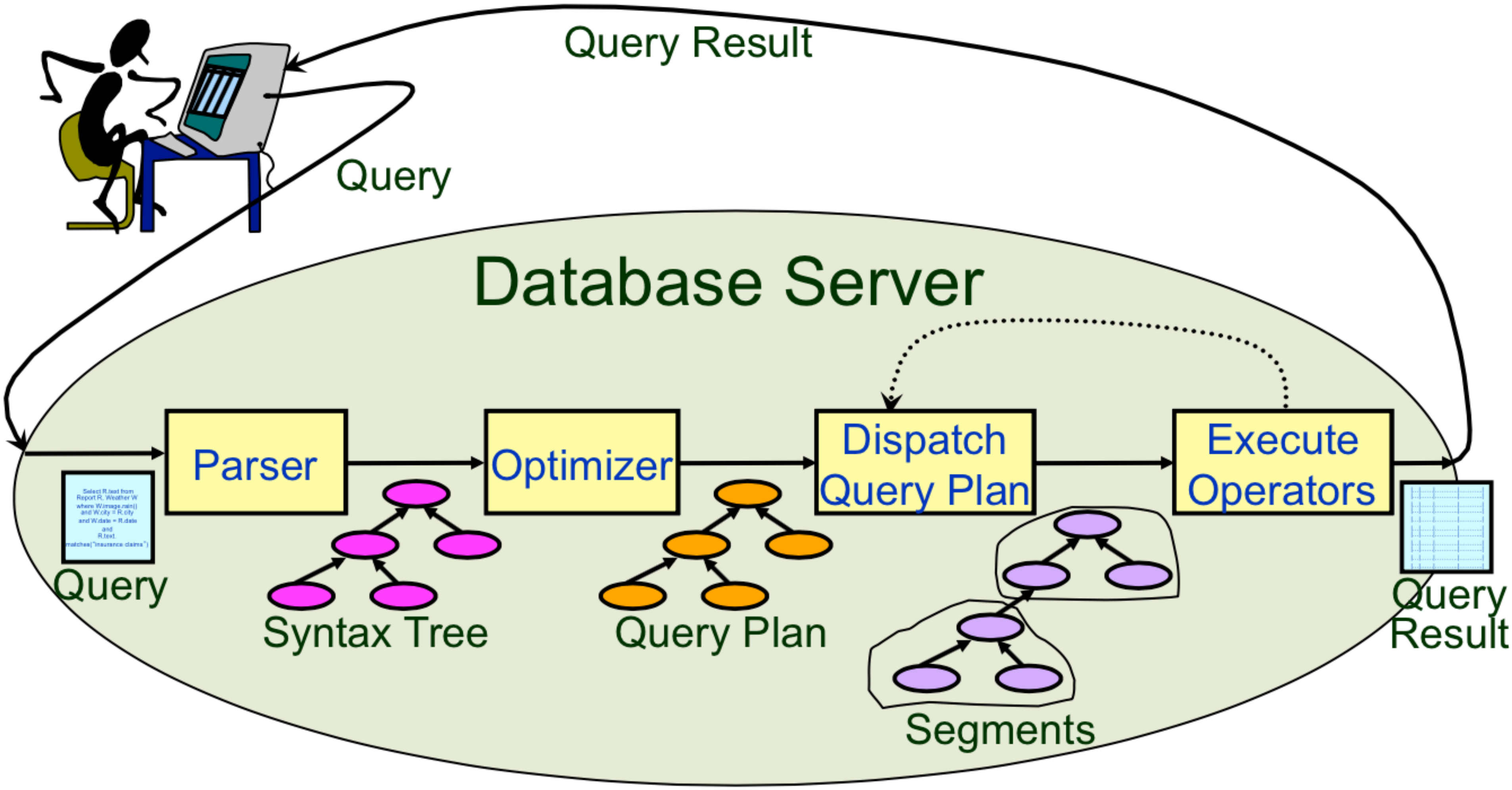
Advanced Join Strategies

Chapter 12 and 14

Query Execution Life-Cycle



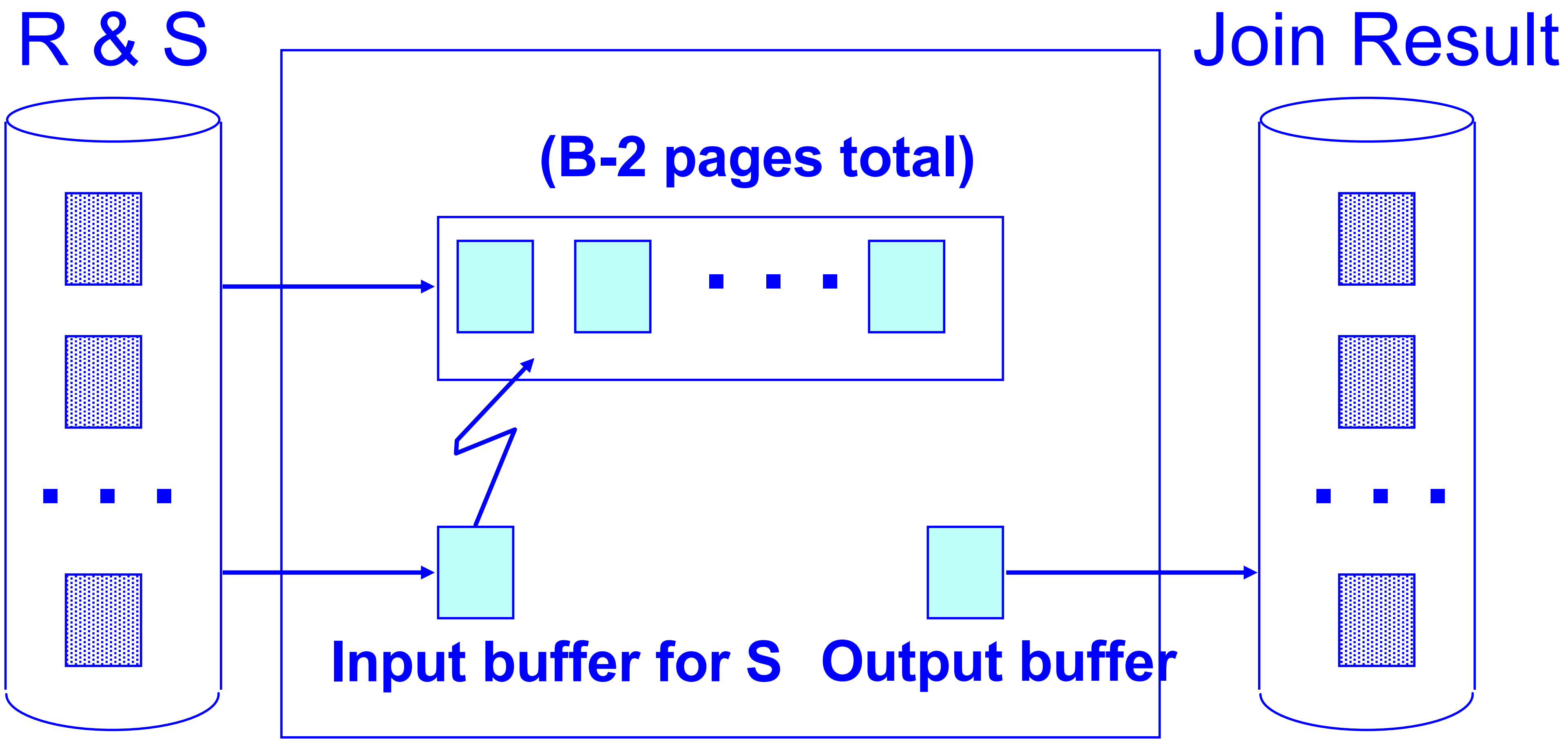
Join strategies seen so far

- Simple nested loops
- Page nested loops
- Block nested loops

- Today:
  - Take advantage of hashing
  - Take advantage of sorting

Block Nested Loops

Can we exploit available memory?
Suppose we have B buffer pages available.



Block Nested Loops

- Can we exploit available memory?
- Suppose we have B buffer pages available.
- Use B-2 pages to hold a block of **outer** R.

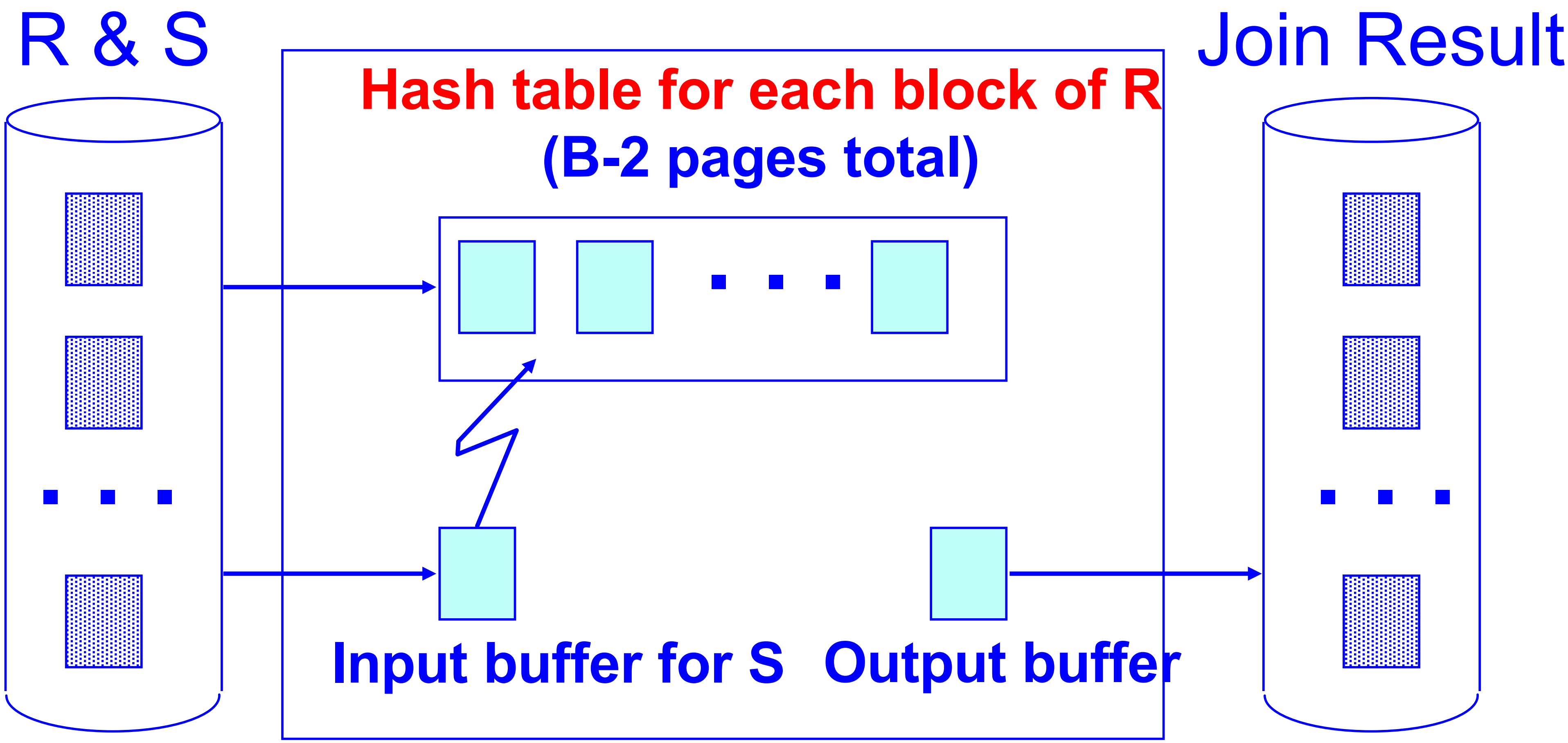
```
foreach block of B-2 pages of R do
    foreach page of S do
        for all matching in-memory tuples r in R and s in S:
            add <r, s> to result
```

Can we speed up the CPU cost?

- I/O Cost: $|R| + |S| * \left\lceil \frac{|R|}{B-2} \right\rceil$

Block Nested Loops – CPU cost

Use an in-memory hash table for R to speed up equality searches on common attributes with S.



Index Nested Loops

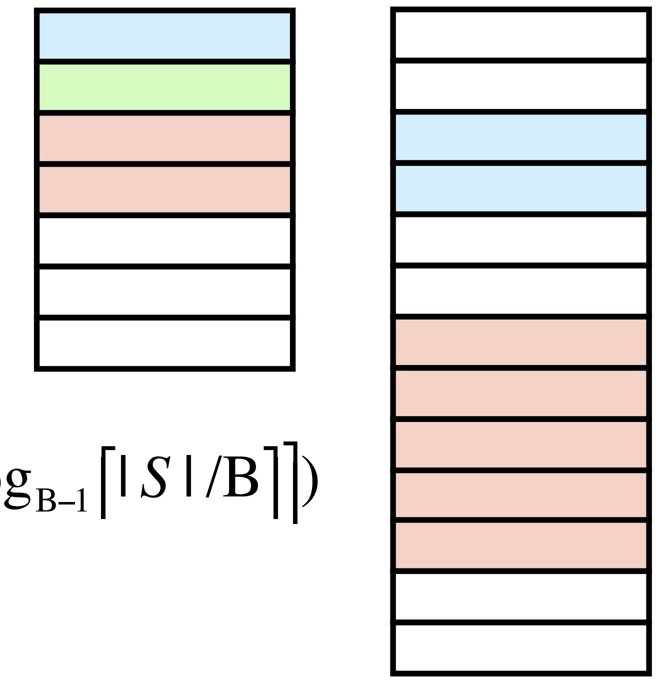
foreach tuple r in R do
    Probe Index on S.sid
        foreach matching tuple s, add <r, s> to result

- Let's say there is an index available on S.
- Use index on join attribute of the inner relation
  - Cost: |R| + (||R|| * cost of finding matching S tuples)
- Cost of finding matching S tuples per R tuple
  - Index cost: typically, 1-2 I/Os for hash index
  - Index cost: typically, 2-4 I/Os for B+ tree.
  - Record retrieval cost
    - Clustered index: one I/O (typical) *for all S tuples per R tuple*
    - Unclustered Index: Up to one I/O *per matching S tuple*.

Sort-Merge Join

1. Sort R on the join attribute (if necessary)
2. Sort S on the join attribute (if necessary)
3. Merge

Sorted R Sorted S



Note: formula for Sort can vary, depending on sorting method used.

- Sort: $2.|R|.(1+\lceil \log_{B-1}\lceil |R|/B \rceil \rceil) + 2.|S|.(1+\lceil \log_{B-1}\lceil |S|/B \rceil \rceil)$
- Merge Cost: (|R|+|S|)
- Merge Worst Case: **(|R|*|S|)**
  - When?
  - Backups needed if #duplicates in S exceeds buffer size

CYU: SM, Page NL, Block NL

- |R| = 128 |S| = 64 B = 8
- Find the cost of R ⋈ S
- Sort-Merge (use two-way merge sort)
  - Sort R: $2 * 128 (\log_2 128 + 1) = 2048$
  - Sort S: $2 * 64 (\log_2 64 + 1) = 896$
  - Merge: 128 + 64 = 192
  - TOTAL: 3136
- Page NL
  - Scan outer: 64
  - Join: 64 * 128 = 8192
  - TOTAL: 8256

In NL, which rel. is the outer?

- Block NL
  - Scan outer: 64
  - Join: $\lceil 64/6 \rceil * 128 = 1408$
  - TOTAL: 1472

CYU: SM, Page NL, Block NL

- $|R| = 128$ $|S| = 64$ $B = 8$
- Find the cost of $R \bowtie S$
- Sort-Merge (use all the buffers for sort)
  - Sort S: $2*64*(1+\lceil \log_7 \lceil 64/8 \rceil \rceil)= 384$
  - Sort R: $2*128*(1+\lceil \log_7 \lceil 128/8 \rceil \rceil)= 768$
  - Merge (input): $64+128 = 192$
  - TOTAL: 1344
- Page NL
  - Scan outer: 64
  - Join: $64 * 128 = 8192$
  - TOTAL: 8256

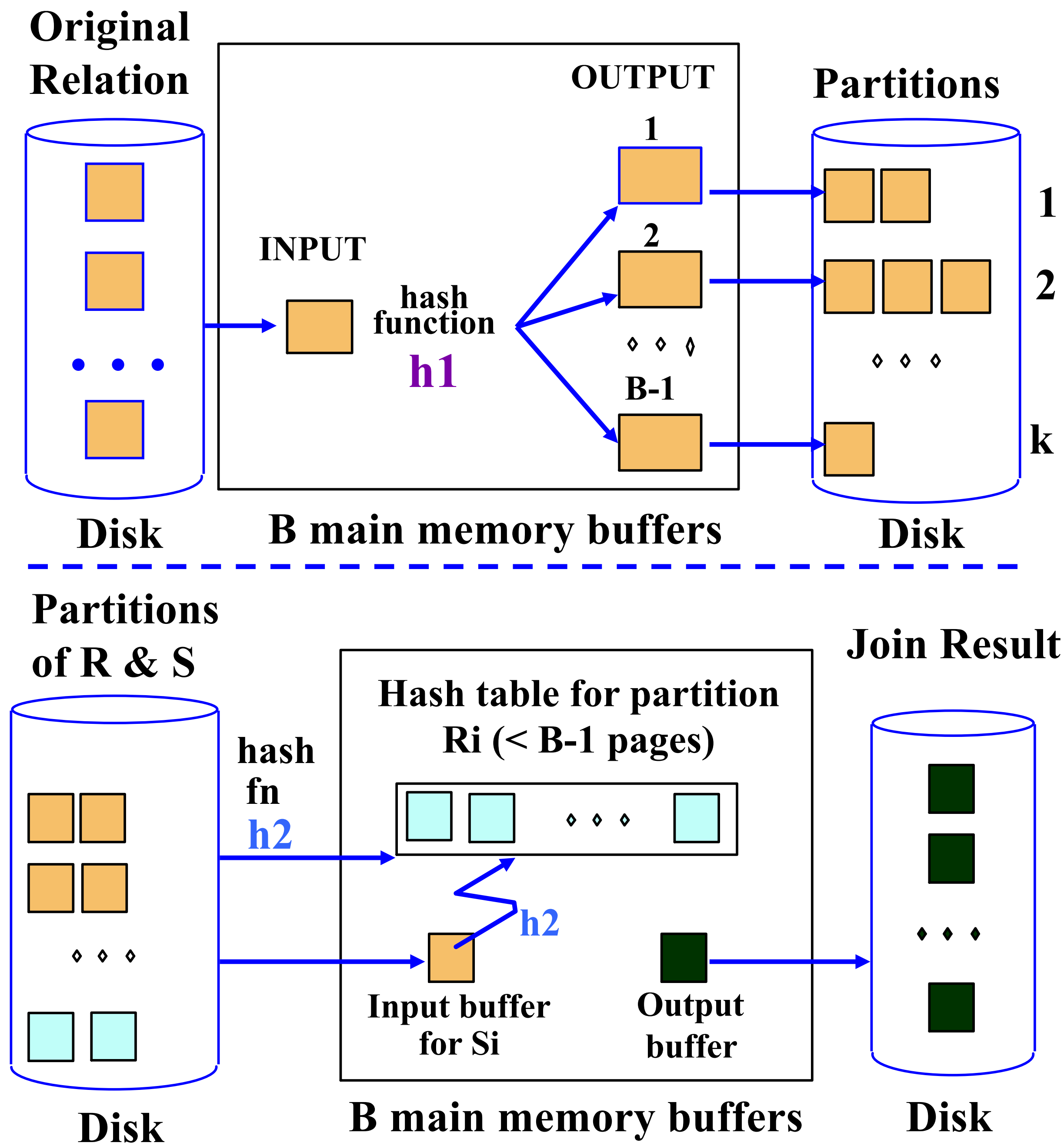
$\log_7 8=1.07$
$\log_7 8=1.42$

- Block NL
  - Scan outer: 64
  - Join: $\lceil 64/6 \rceil * 128 = 1408$
  - TOTAL: 1472

Grace Hash Join

- Step 1: (Build) Hash both relations, R and S, on the join attribute, producing *k* disk-based partitions
  - Guarantees that R tuples can only join with S tuples in the same partition
- Step 2: (Probe) Read in complete partition of (smaller relation) R, and scan S partition for matches .Use a different hash function h2

- **k = B-1 (# of buckets)**
- h1 ≠ h2



Cost Analysis: Grace Hash Join

- Partition phase, read+write both relations; $2(|R|+|S|)$, assuming one partition round is enough.
- In probe phase,
  - read each partition of relation R once:
    - For that, read the corresponding partition of S once
  - Total cost for all partitions: $|R|+|S|$ I/Os.
  - *Assumes each R partition fits in memory in probe phase*
- This is *Grace hash join. 3(|R| + |S|) I/Os.*
- The purpose of h2: *reducing CPU costs*

Cost Analysis: Grace Hash Join

- Grace Hash Join can sometimes require additional partitioning rounds (use additional hash functions on both R and S). The goal is to make all the partitions of the smaller of R or S fit entirely in B-2 buffers.

Sort-Merge in 3(|R|+|S|) I/Os

- Sort-merge can be optimized to 3(|R|+|S|) I/Os
  - When larger relation |S| ≤ B
  - In other words, sufficient memory to do internal sort.

- **Rationale:**
  - Since R fits in memory, 2*|R| to sort R.
  - Since S fits in memory, 2*|S| to sort S.
  - Once both are sorted, assuming no backups are required, merge would take an additional |R| + |S| read cost.

Sort-Merge in 3(|R|+|S|) I/Os

- Sort-merge can be optimized to 3(|R|+|S|) I/Os
  - When larger relation $|S| \leq B^2$
  - In other words, sufficient memory for only square root of the # of pages in the larger relation.
- **Rationale:**
  - Let's say |R| is 400, |S| is 10,000, B is 100 pages.
  - Produce runs of 196 pages long (2*M). Let's call it 200 page long runs (approx).
  - We end up with 2 runs of R, and 50 runs of S.
  - Read in one page from each run into memory (52 pages total needed), merge and on-the-fly apply join condition, producing the output (one output page needed).

Join Algorithms

- Hash-Join vs. Sort-Merge Cost:
  - Hash join costs 3(|R|+|S|) I/Os
    - if each partition of the smaller relatoin fits in memory
  - Sort-merge can be optimized to 3(|R|+|S|) I/Os
    - When larger relation $|S| \leq B^2$ (details omitted)
  - Memory requirements (rough):
    - Sort-merge: Larger relation $|S| \leq B^2$
    - Hash join: Smaller relation $|R| \leq B^2$
  - Hash Join superior if relation sizes differ greatly.
  - Hash Join is highly parallelizable.
  - Hash join poor if partitioning is skewed
  - Sort-Merge better if relations already sorted

General Join Conditions

- Equalities over several attributes e.g., *R.sid=S.sid* AND *R.rname=S.sname*:
  - Block Nested Loop
    - That will always work.
  - Index Nested Loop:
    - Index on *<sid, sname>* or *sid* or *sname*.
  - Sort-merge join:
    - sort the Reserves table on <sid, rname>; and
    - the Sailors table on <sid, sname>
  - Hash join:
    - hash the Reserves table on <sid, rname>; and
    - the Sailors table on <sid, sname>

Inequality conditions

- Inequality conditions (e.g., *R.rname < S.sname*):
  - Block Nested Loop: That should still work.
  - Sort-Merge and Hash Join not applicable
  - Index nested-loop, need B+ tree index.
    - Hash index will not help.

Announcements

- Optional Exercises:
  - 12.1 (1-4), 12.3, 12.5
  - 13.1, 13.3
  - 14.1 (2, 3, 4, 6, 7, 8, 9, 10), 14