Data Processing: Sort-based and Hash-based Joins and Parallelism

ACS, Yongluan Zhou

What should we learn today?

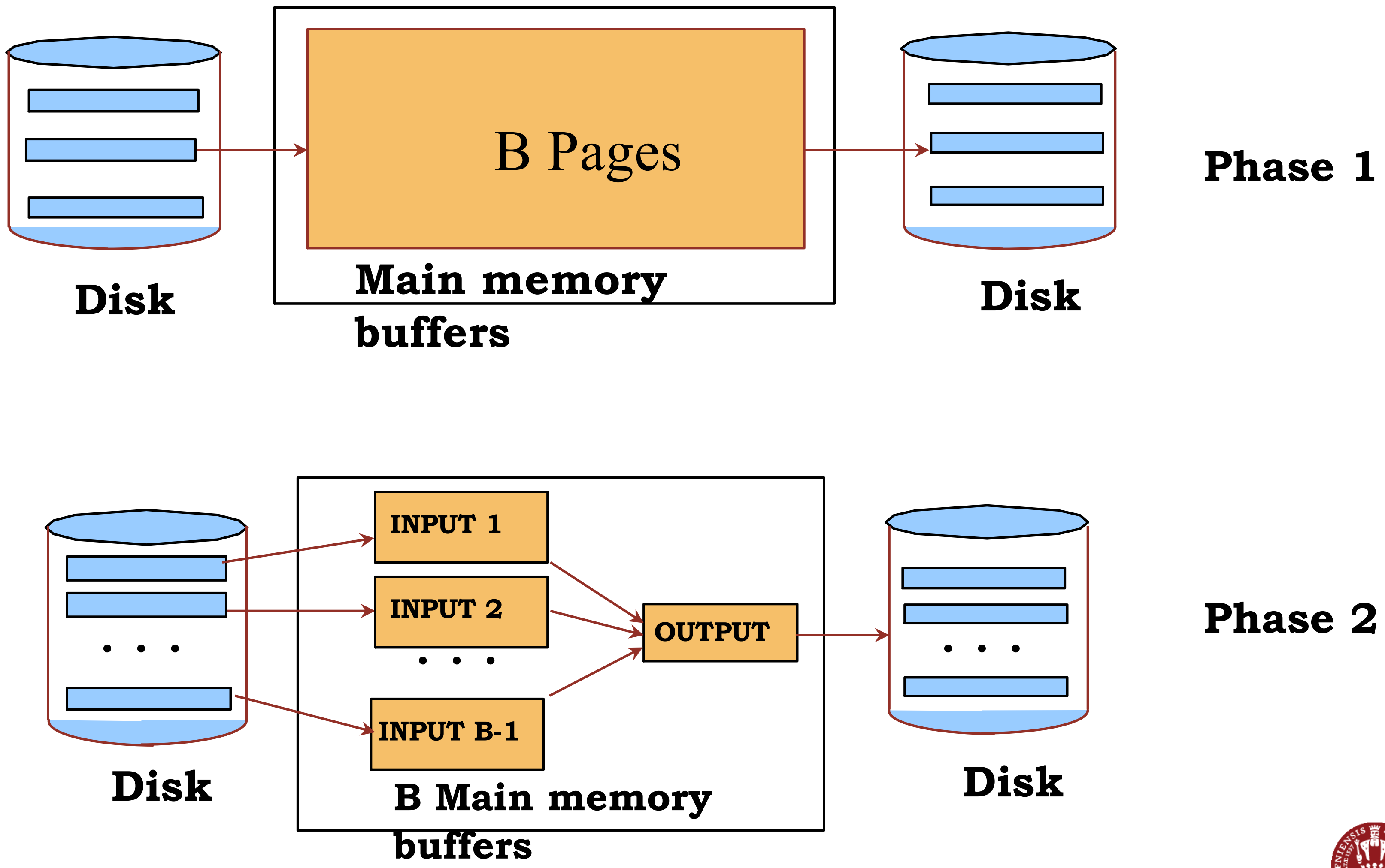
- **Sort-based and hash-based** join algorithms.
- Identify the main metrics in parallel data processing, namely **speed-up and scale-up**
- Describe different models of parallelism (**partition, pipelined**) and architectures for parallel data processing (**shared-memory, shared-disk, shared-nothing**)
- Explain different **data partitioning** strategies as well as their advantages and disadvantages
- Apply data partitioning to achieve parallelism in data processing operators
- Explain the main algorithms for parallel processing, in particular **parallel scan, parallel sort, and parallel joins**

Do-it-yourself-recap: Sort-Merge and Hash-Based Algorithms

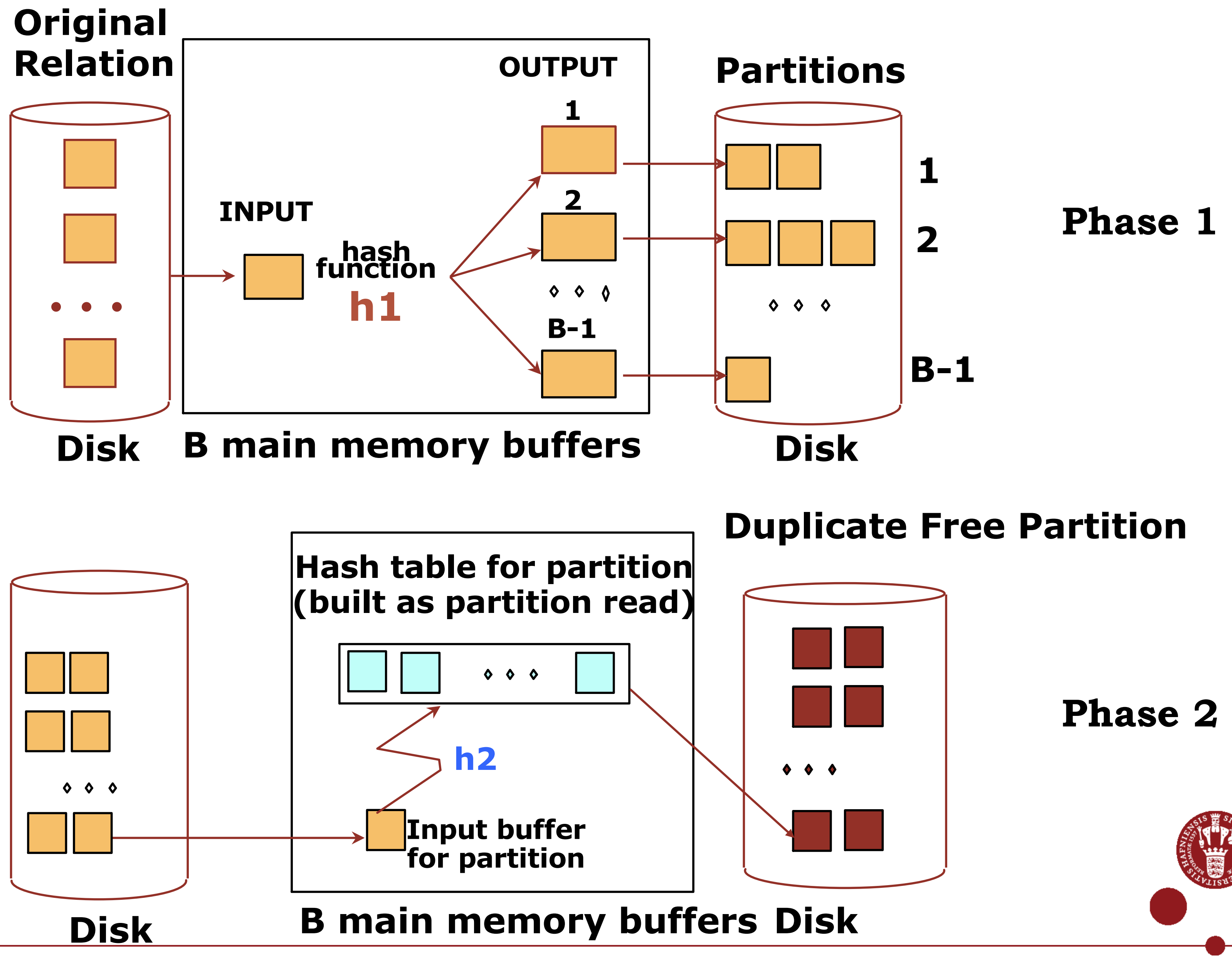
- One relation R
- Sort-merge duplicate elimination
  - How did the algorithm work?
  - Is merge enough or is sorting always needed?
- Hash-based duplicate elimination
  - How did the algorithm work?
  - Why did you need two different hash functions?

Source: Ramakrishnan & Gehrke (partial)

Sort-Merge Algorithm Phases



Hash-Based Algorithm Phases



Sort-Merge Join

1. Sort R on join attr(s)
2. Sort S on join attr(s)
3. Scan sorted-R and sorted-S in tandem, to find matches

Example:

```
SELECT  *
FROM    Reserves R1, Sailors S1
WHERE   R1.sid=S1.sid
```

Q: What if all the sid in the two tables are identical, and memory can only hold 2 records from each table at the same time

sid	sname	rating	age
22	dustin	7	45.0
22	yuppy	9	35.0
22	lubber	8	55.5
22	guppy	5	35.0
22	rusty	10	35.0

sid	bid	day	rname
22	103	12/4/96	guppy
22	103	11/3/96	yuppy
22	101	10/10/96	dustin
22	102	10/12/96	lubber
22	101	10/11/96	lubber
22	103	11/12/96	dustin

Cost of Sort-Merge Join

- Cost: Sort R + Sort S + (|R|+|S|)
  - But in the worst case, last term could be |R|*|S| *(very unlikely!)*
  - Q: what is worst case?

Suppose B = 35 buffer pages (|R|=1000, |S|=500):

- Both R and S can be sorted in 2 passes
- Total join cost = 4*1000 + 4*500 + (1000 + 500) = 7500

Suppose B = 300 buffer pages (|R|=1000, |S|=500):

- Again, both R and S sorted in 2 passes
- Total join cost = 7500

Block-Nested-Loop cost = 2,200 ... 15,000

Other Considerations …

- An important refinement:

  Do the join during the final merging pass of sort !

  - If have enough memory, can do:

    1. Read R and write out sorted runs
    2. Read S and write out sorted runs
    3. Merge R-runs and S-runs, and find $R \bowtie S$ matches

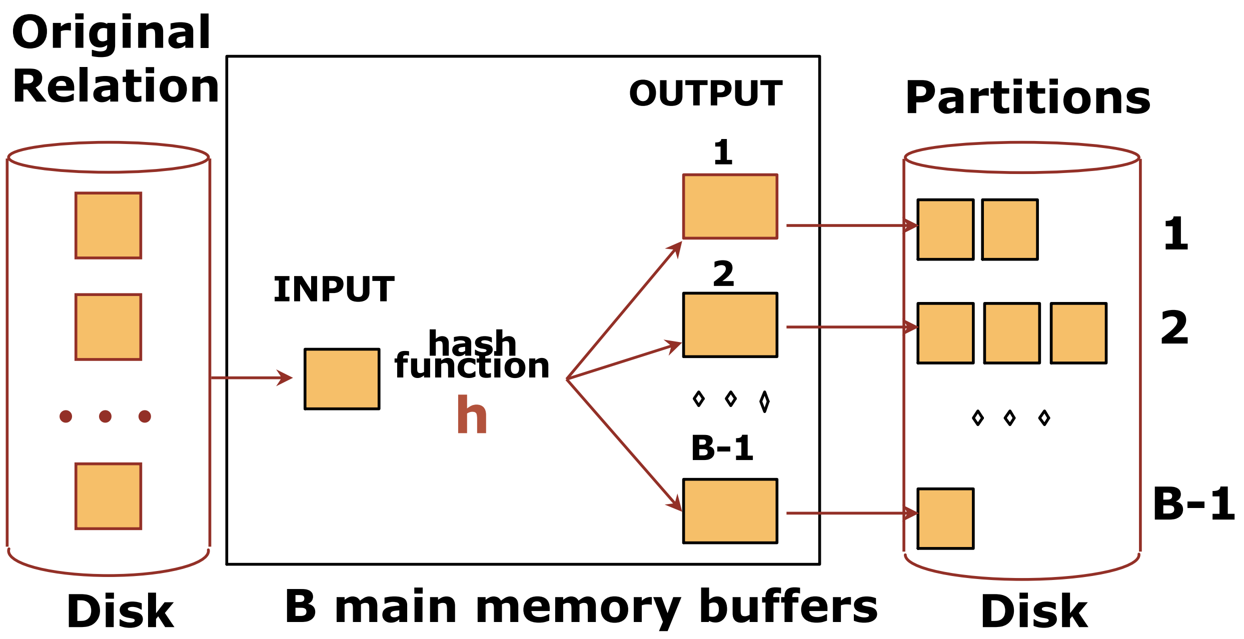
  Cost = 3*|R| + 3*|S|

  Q: how much memory is "enough"

- Sort-merge join an especially good choice if:
  - one or both inputs are already sorted on join attribute(s)
  - output is required to be sorted on join attribute(s)

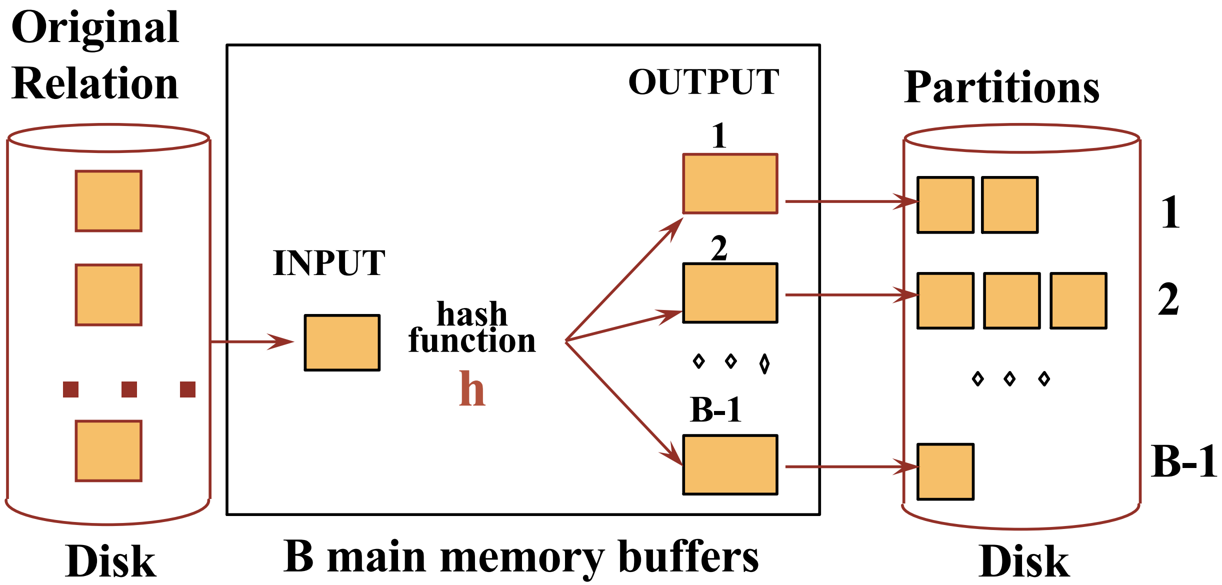
GRACE Hash Join

- Partition both relations using hash fn h: R tuples in partition i will only match S tuples in partition i.
- R join S = R_1 join S_1 U R_2 join S_2 U … U R_{B-1} join S_{B-1}

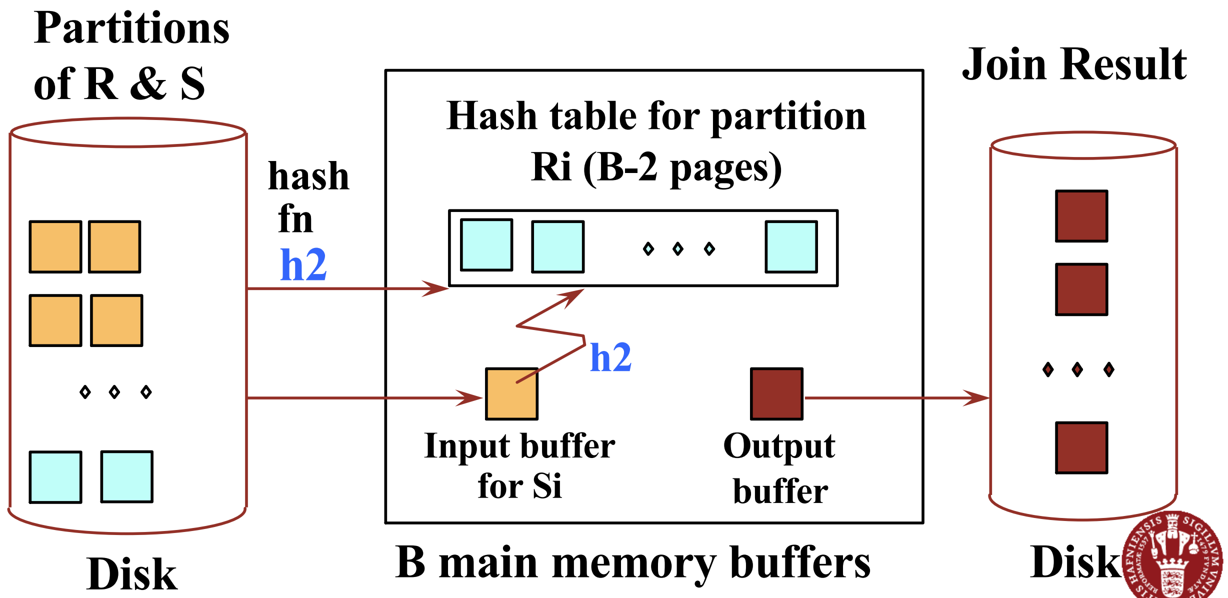


Grace Hash Join

- Partitioning phase: read+write both relations
  $\Rightarrow$ 2(|R|+|S|) I/Os



- Matching phase: read both relations
  $\Rightarrow$ |R|+|S| I/Os



- If memory is enough, 2-pass hash join cost = 3(|R|+|S|)

Q: how much memory is "enough"?

2-Pass Hash Join vs. 2-Pass Sort-Merge Join

- Given “enough” memory, both have cost of 3 (M + N)

- Benefits of hash join
  - Superior if relation sizes differ greatly
  - **Refinement**: hybrid hash join allows for dynamically adjusting to smaller relation fitting in main memory
  - Highly parallelizable

- Benefits of sort-merge join
  - Less sensitive to data skew
  - Result is sorted

Relational Operators

- We now study implementation alternatives

- Select
- Project
- Join
- Set operations (union, intersect, except)
- Aggregation

Set Operations

- Intersection and cross-product special cases of join.
- Union (Distinct) and Except are similar; we'll do union.
- **Sorting based approach to union:**
  - Sort both relations (on combination of all attributes).
  - Scan sorted relations and merge them.
  - Alternative: Merge runs from Pass 0 for both relations.
- **Hash based approach to union:**
  - Partition R and S using hash function h.
  - For each S-partition, build in-memory hash table (using h2), scan corresponding R-partition and add tuples to output while discarding duplicates.

Relational Operators

- We now study implementation alternatives

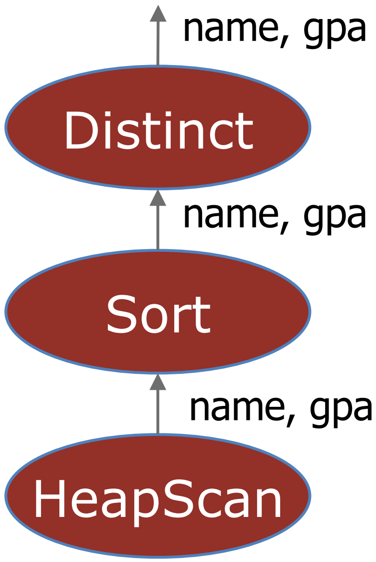
- Select
- Project
- Join
- Set operations (union, intersect, except)
- Aggregation

Homework! ☺

Query Execution Framework

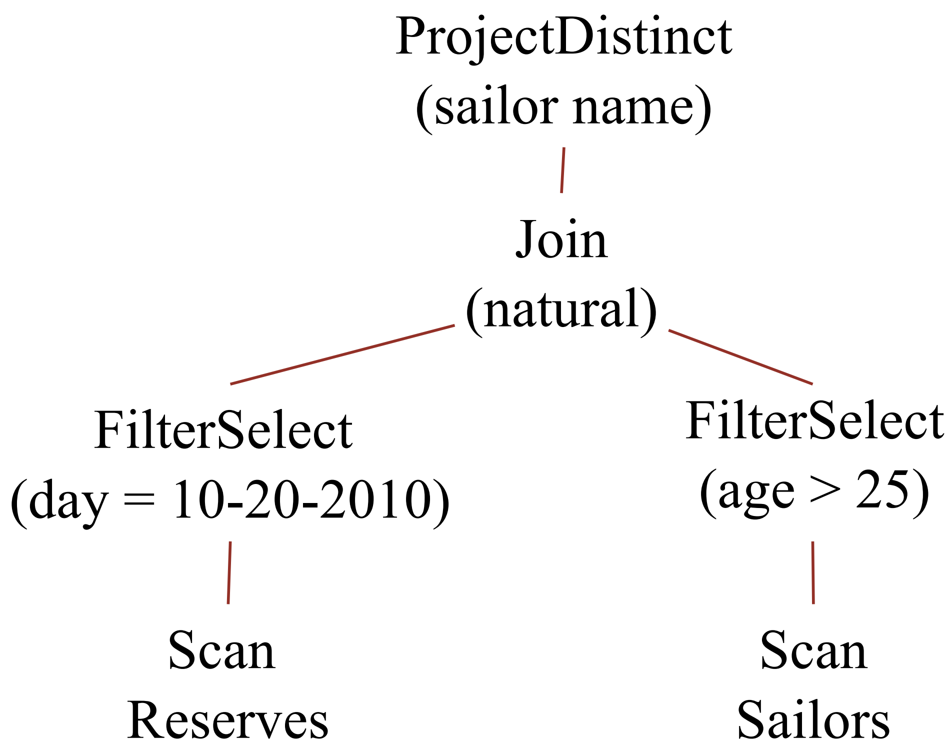
```
SELECT DISTINCT name, gpa
FROM Students
```

One possible query execution plan:



Design Goals for Operator Interface

- Must be able to compose several operators together into operator tree
- Must coordinate how data is passed between operators
- Should only buffer necessary data in main memory



Pull Model

- User requests one tuple at a time from top-level operator
- Operators calculate next tuple by requesting tuples from their input operators

ONC-Interface (iterator)

- **Open**
  - Initializes the operator
- **Next**
  - Calculates the next tuple and returns it to caller
- **Close**
  - Cleans up and closes operator

Example: Basic Selection (FilterSelection)

```
FilterSelection {
        Iterator input;
        Predicate P;
        …
        public void open() { input.open(); }

        public Tuple next() {
                Tuple result = null;
                do {
                        result = input.next();
                } while ( !P(result) )
                return result;
        }

        public void close() { input.close(); }
}
```

Parallel Data Processing

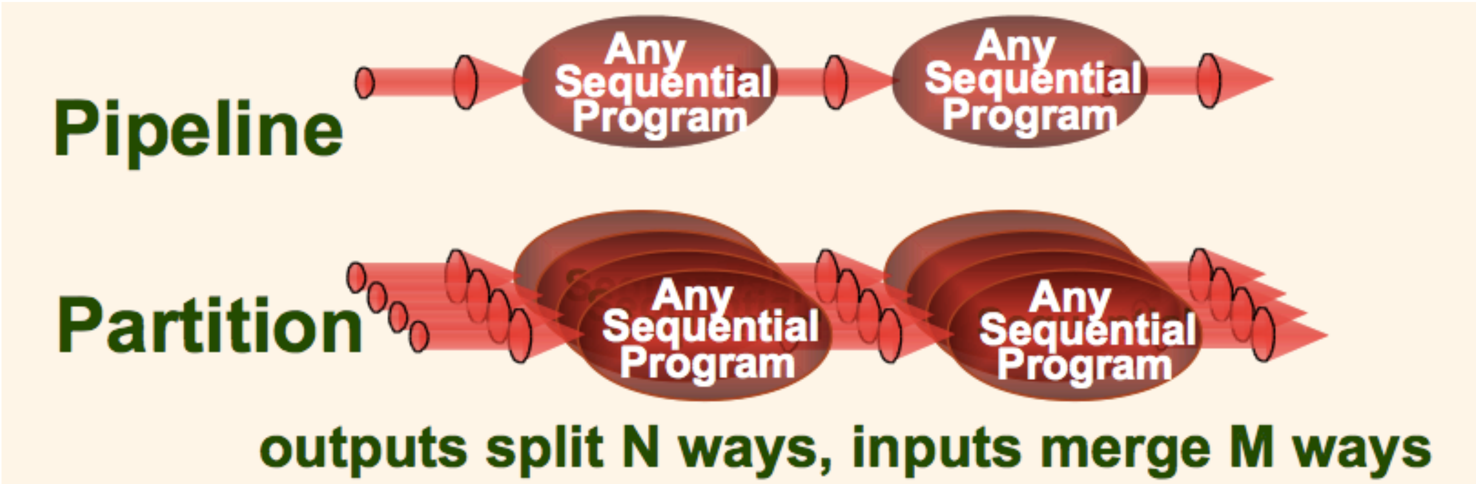
Why Parallel Access to Data?

- **Take 1 TB of data**
  - At 10MB/s → 1.2 days to scan
  - 1000x parallel → 1.5 minute to scan
- **Bandwidth!**
- **Parallelism**
  - Divide big problem into many smaller ones to be solved in parallel
  - Take advantage of natural independence among portions of processing

Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

Parallel Databases: Intro

- Parallelism is natural to data processing
  - *Pipeline parallelism:* many machines each doing one step in a multi-step process.
  - *Partition parallelism:* many machines doing the same thing to different pieces of data.
  - **Both are natural in database systems!**



Source: Ramakrishnan & Gehrke (partial),
Joe Hellerstein, Berkeley (partial)

Database Systems: The || Success Story

- Database systems are the most successful application of parallelism
  - Teradata, Greenplum, Vertica, many others
  - Every major DBMS vendor has some || server
  - Key in **analytics**, Big Data, major market growth

Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

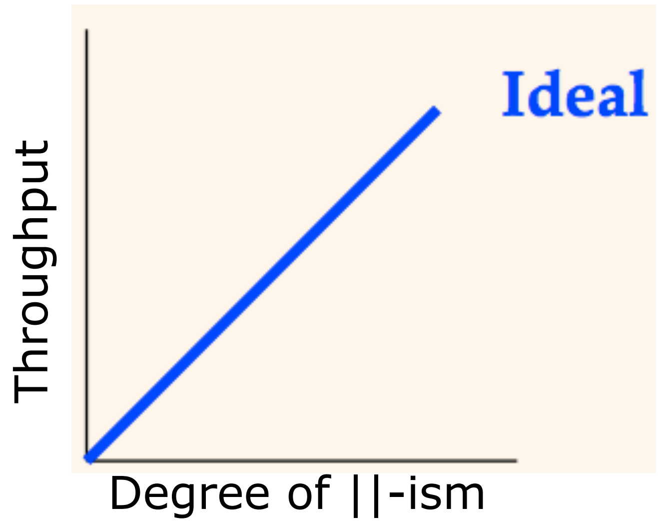
Reasons for success

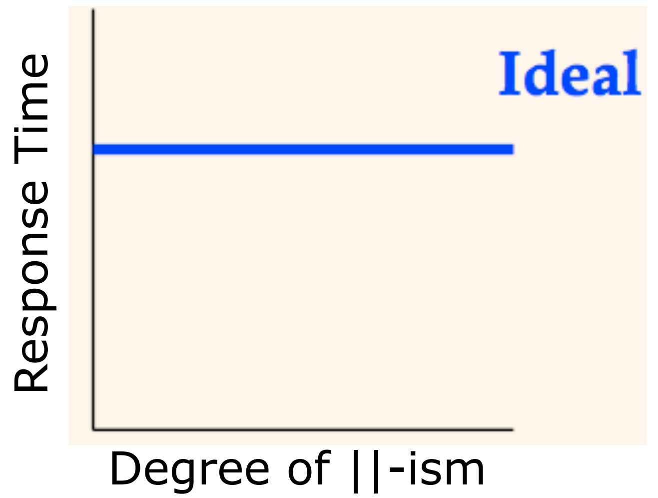
- **A close set of operators with well-defined semantics**
  - Fully optimized implementations
  - Automatic optimizer
- **Bulk-processing** (= partition ||-ism). Natural pipelining.
- **Inexpensive hardware** can do the trick!
- Users/app-programmers **don't need to think in ||**

Some || Terminology

- Speed-Up
  - More resources means proportionally less time for processing a given amount of data
  - Can we have super-linear speed-up?

- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant

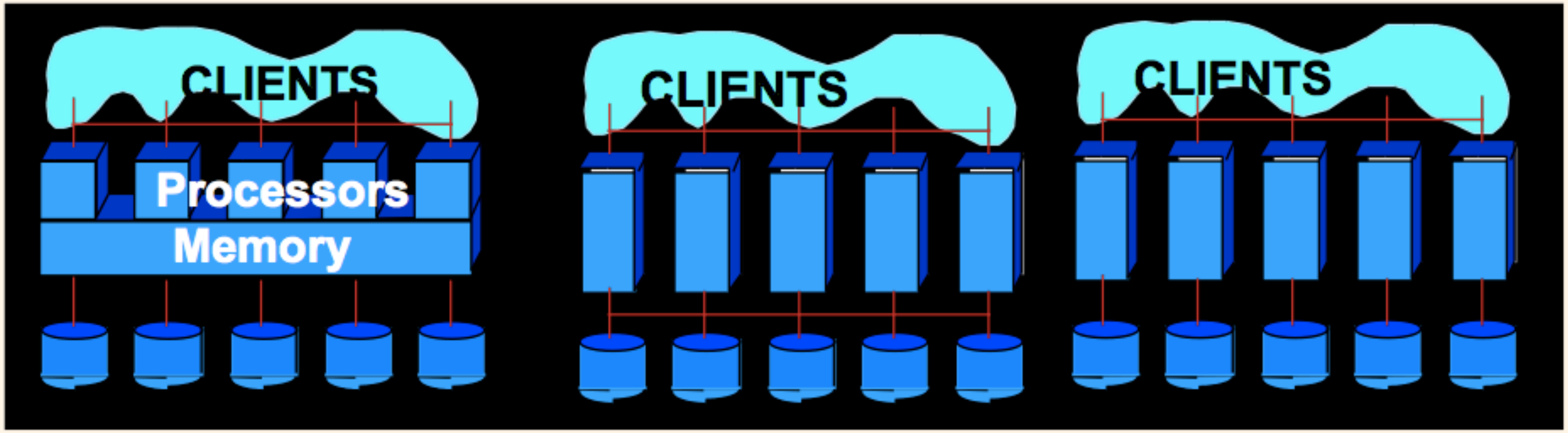




Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

Architectural Issue: Shared What?

Shared Memory (SMP) | Shared Disk | Shared Nothing (network)



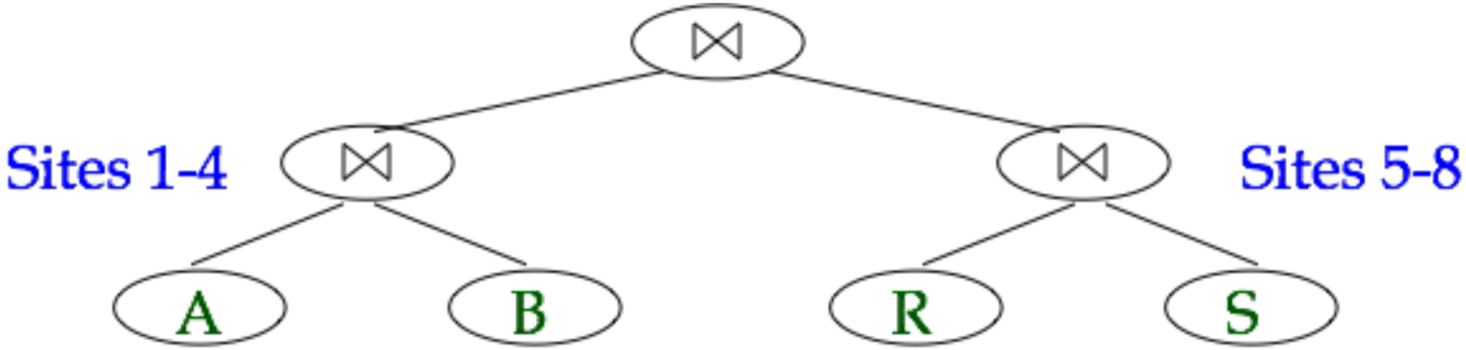
Easy to program
Expensive to build
Difficult to scaleup

Hard to program
Cheap to build
Easy to scaleup

Source: Ramakrishnan & Gehrke (partial),
Joe Hellerstein, Berkeley (partial)

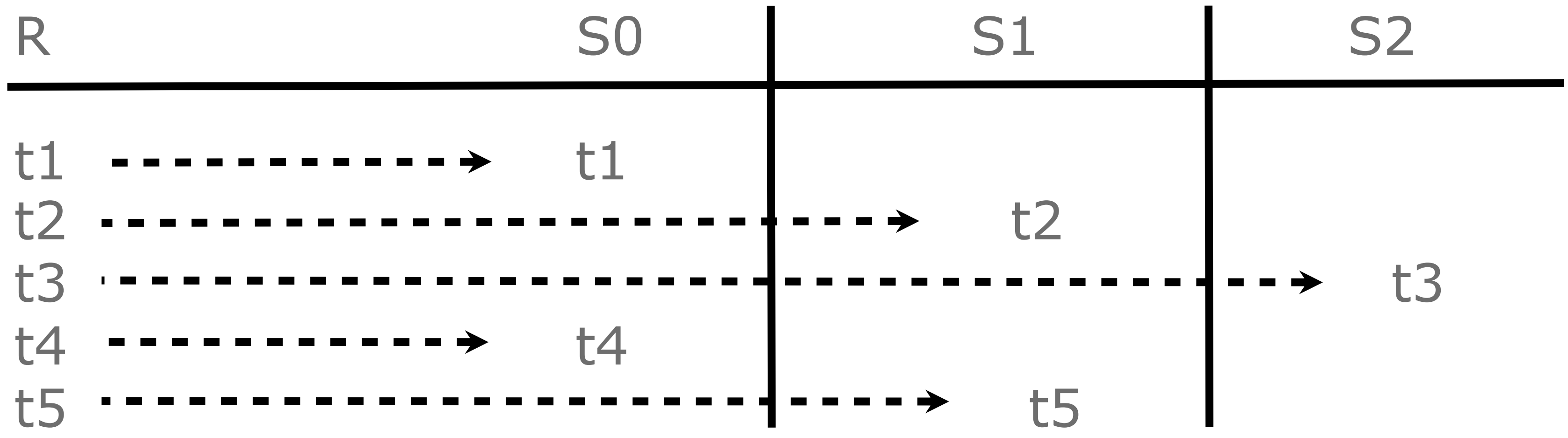
Different Types of Database System ||-ism

- Intra-operator parallelism
  - get all machines working to compute a given operation (scan, sort, join)
- Inter-operator parallelism
  - each operator may run concurrently on a different site (exploits pipelining)
- Inter-query parallelism
  - different queries run on different sites
- We'll focus on intra-operator ||-ism

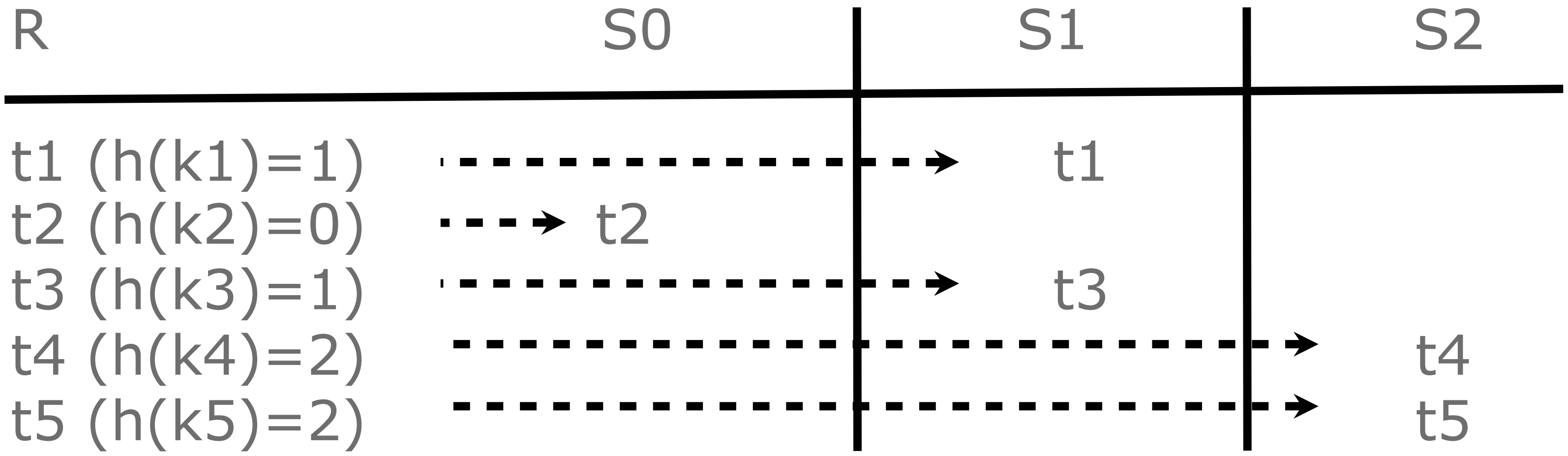


Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

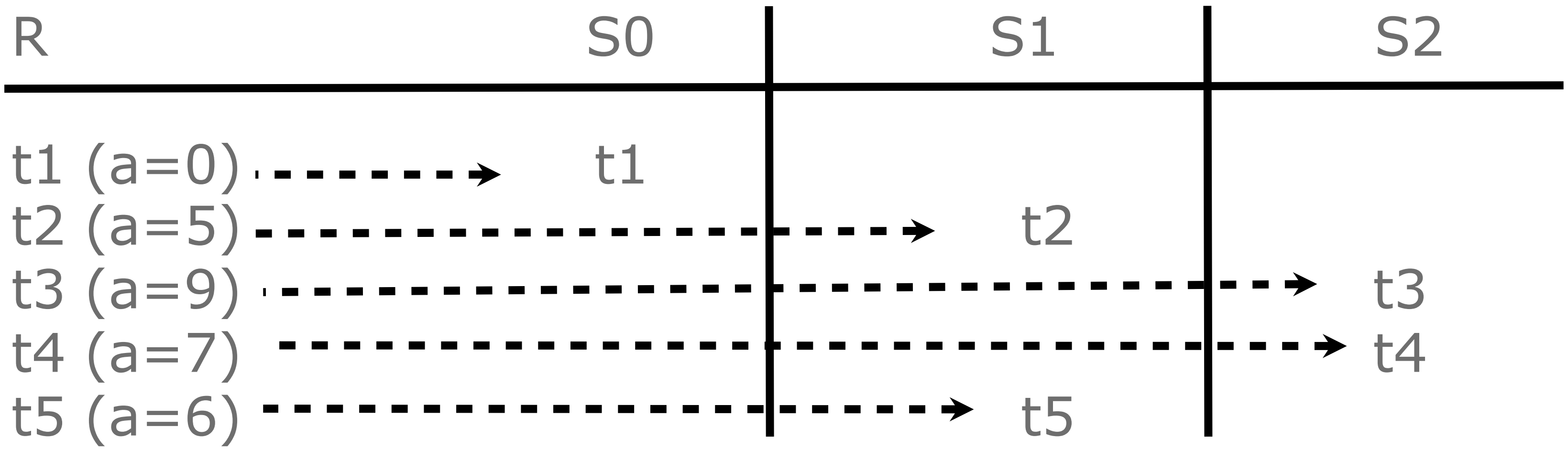
Round Robin



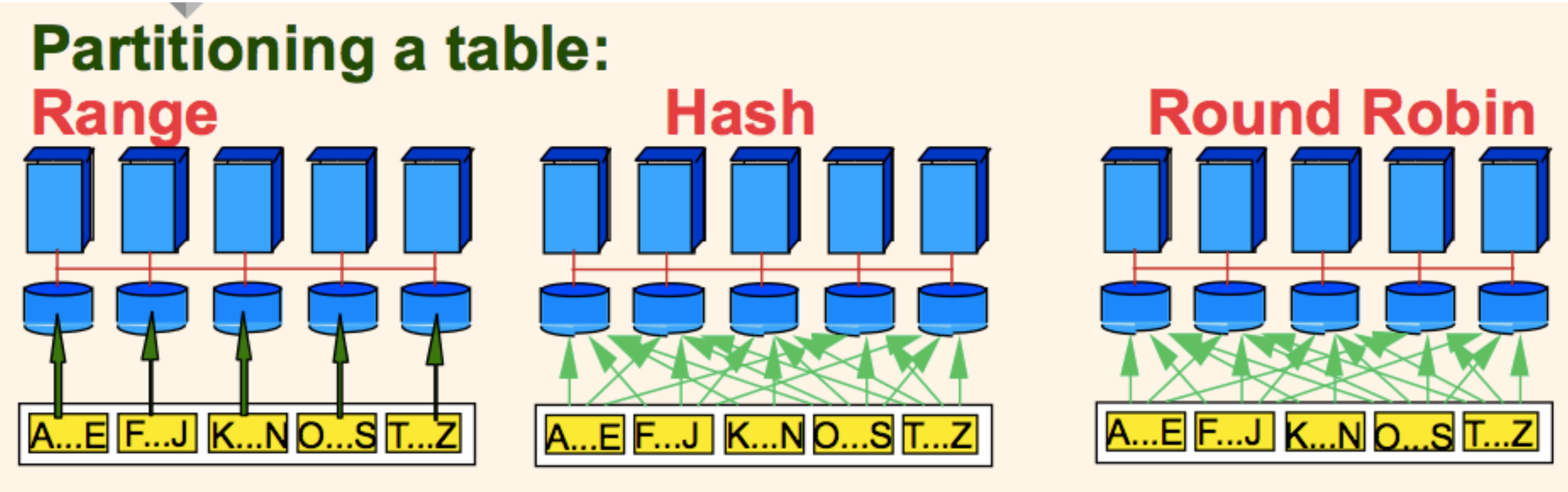
Hash Partitioning



Range Partitioning



Discussion: Automatic Data Partitioning



- What type of queries is each method good or bad for?
  - Type of queries
- Do they suffer from data skewness?
  - and hence load imbalance?

Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

Handling Skew

- For range partitioning, sample load on disks.
  - Cool hot disks by making range smaller
- For hash partitioning,
  - Cool hot disks by making more buckets than #nodes, and then mapping some buckets on hot disks to others
- During query processing
  - Use hashing and assume uniform
  - If range partitioning, sample data and use histogram to level the bulk
  - SMP/River scheme: work queue used to balance load

Source: Joe Hellerstein, Berkeley (partial)

Questions so far?

Parallel Scans

- Scan in parallel, and merge.
- Selection may not require range or hash partitioning.
- Indexes can be built at each partition.

Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

Discussion: Hashing-Sorting Duality

- Sort-based algorithms
  - Work independently on chunks
  - Then merge
- Hash-based algorithms
  - Partition
  - Then work independently on buckets

- Is there a partition-based parallel sorting approach?
- Is there a merge-based parallel sorting approach?
- How would it work?

Parallelizing Sort

- **Why?**
  - DISTINCT, GROUP BY, ORDER BY, sort-merge join, index build
- **Phases:**
  - 1. Redistribute data to all nodes by range partitioning
  - 2. Sort data in parallel
  - 3. Read the entire sorted relation by visiting the nodes in order
- **Notes:**
  - phase 1 requires repartitioning 1-1/n of the data! High bandwidth network required.
  - phase 2&3 totally local processing
  - *linear* speedup, scaleup!

Source: Joe Hellerstein, Berkeley (partial)

Parallel Joins

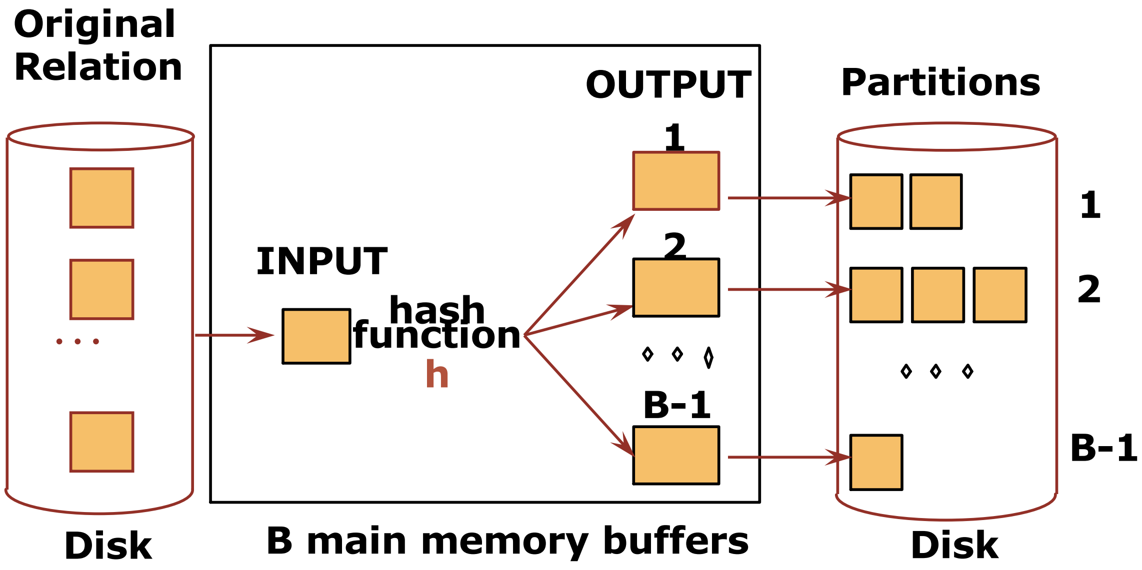
- Nested loop:
  - Each outer tuple must be compared with each inner tuple that might join.
  - Easy for hash/range partitioning on join cols or replicating the inner table, hard otherwise!

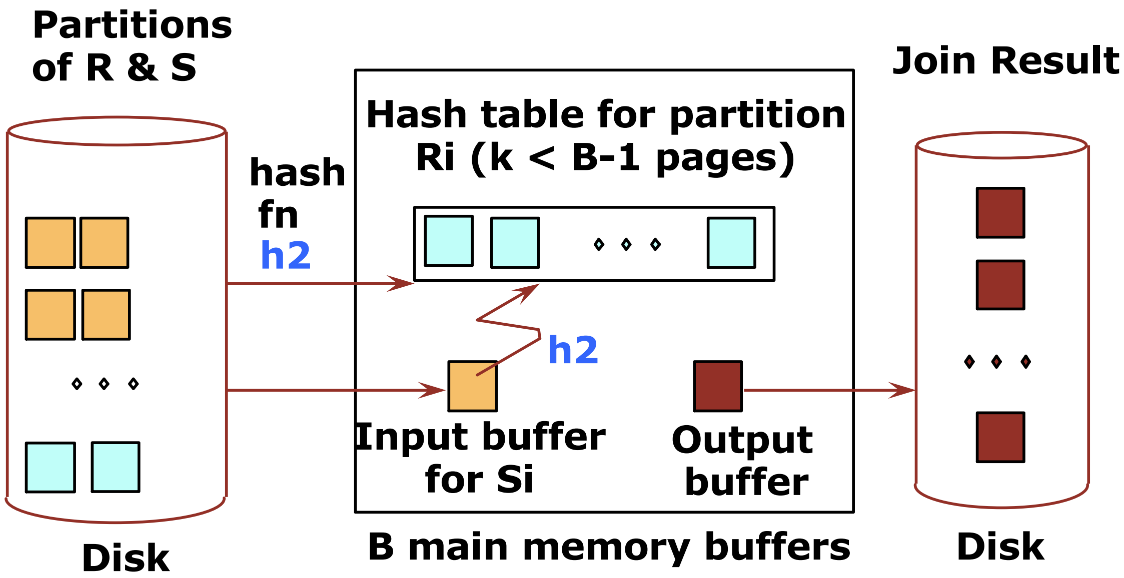
- Sort-Merge (or plain Merge-Join):
  - Sorting produce range-partitioned data
    - R and S should use the same partitioning function
    - skews of two relations
      - Goal: $|R_i|+|S_i|$ are the same for all i
  - Local merging of the partitioned tables

Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

Parallelizing Hash Join?

- Partition both relations using hash fn h: R tuples in partition i will only match S tuples in partition i.
- Read in a partition of R, hash it using h2 (<> h!). Scan matching partition of S, search for matches.

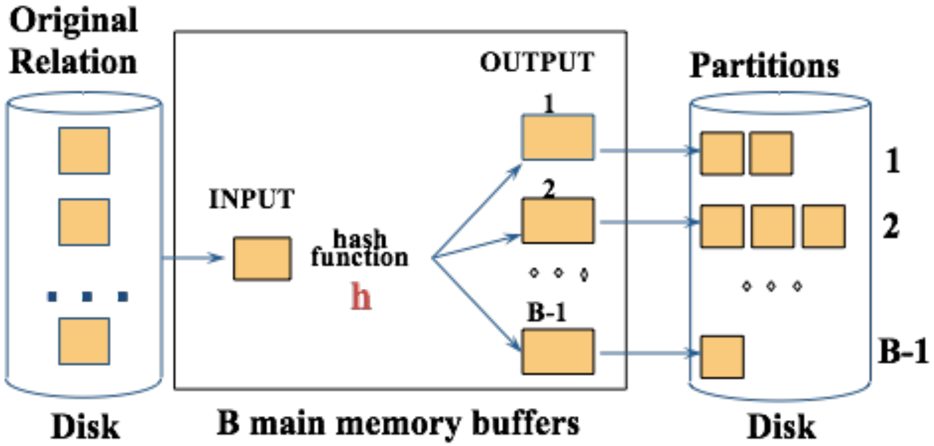




Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

Parallel Hash Join

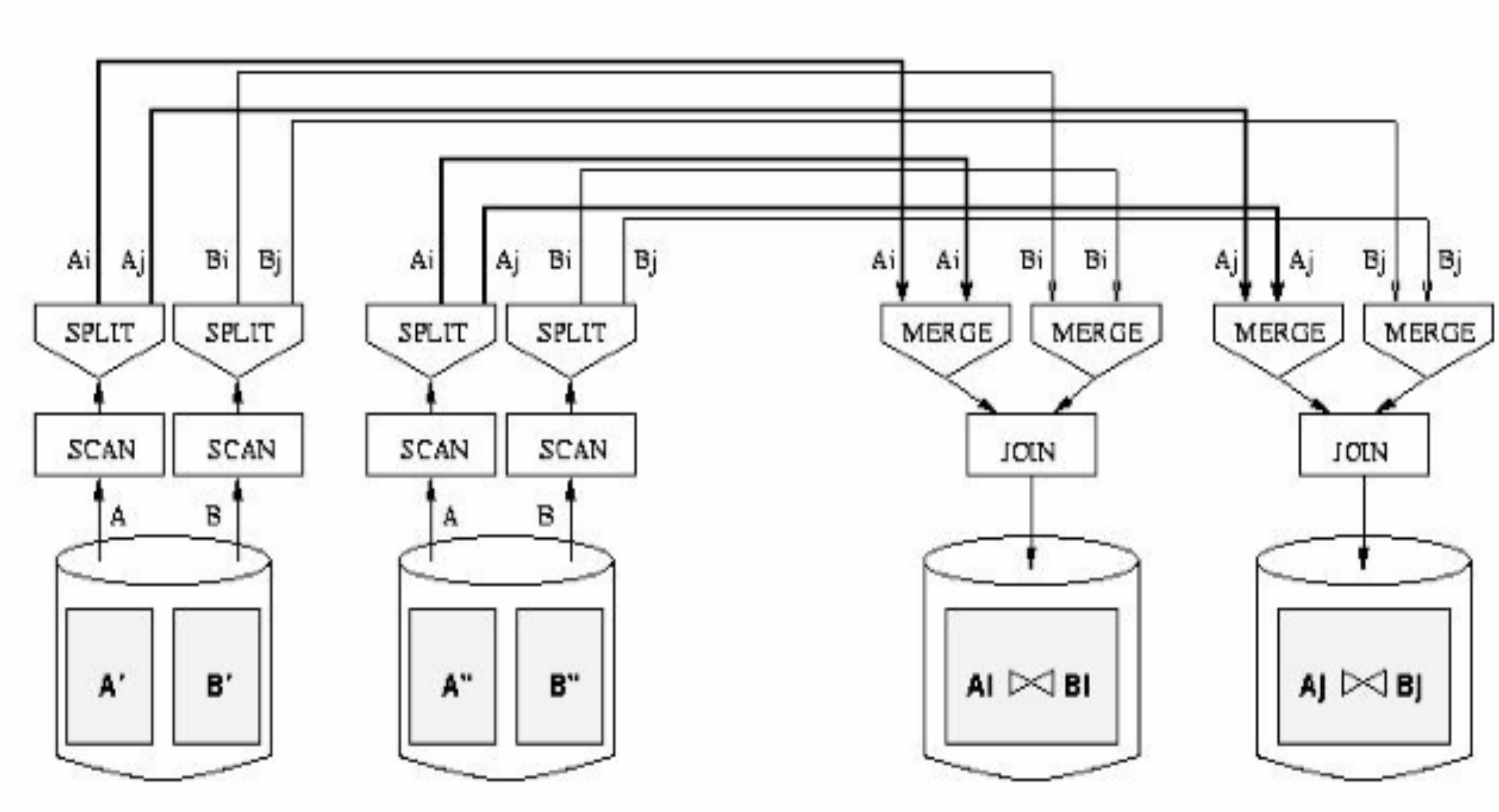
Phase 1



- In Phase 1, partitions get distributed to different sites:
  - A good hash function automatically distributes work evenly!
- Do second phase at each site.
- Almost always the winner for equi-join.

Dataflow Network for Parallel Join

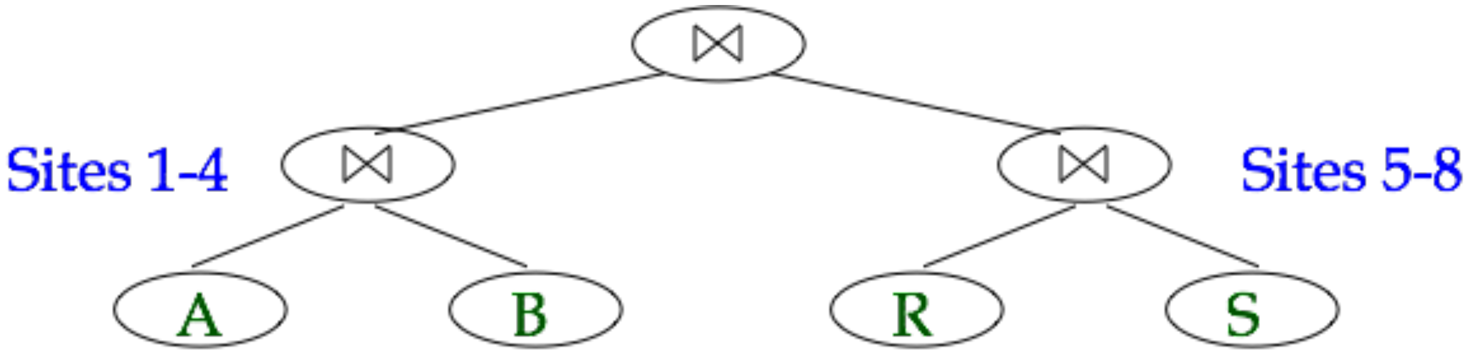
Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)



- Good use of split/merge makes it easier to build parallel versions of sequential join code.

Complex Parallel Query Plans

- Complex Queries: Inter-Operator parallelism
  - Pipelining between operators:
    - note that sort and phase 1 of hash-join block the pipeline!!



Source: Ramakrishnan & Gehrke (partial), Joe Hellerstein, Berkeley (partial)

What should we learn today?

- Identify the main metrics in parallel data processing, namely **speed-up and scale-up**
- Describe different models of parallelism (**partition, pipelined**) and architectures for parallel data processing (**shared-memory, shared-disk, shared-nothing**)
- Explain different **data partitioning** strategies as well as their advantages and disadvantages
- Apply data partitioning to achieve parallelism in data processing operators
- Explain the main algorithms for parallel processing, in particular **parallel scan, parallel sort, and parallel joins**