Parallel Databases

R&G Chapter 22

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Creating Parallelism

- Division of Labor:
  - Stages of Execution: Pipeline Parralellism
  - Parts of Data: Partition Parallelism
- Benefits: Speed Up vs Scale-Up
- Architectures:
  - Shared Memory, Shared Disk, Shared Nothing

Creating Parallelism

- Inter-Operator Parallelism
  - (Mostly) Easy: Partition operators between nodes.
- Inter-Query Parallelism
  - Hard: Need to deal with xacts (next lecture)
- Intra-Operator Parallelism
  - Our focus for today.

Partitioning the Data

- Which machine does a data value end up at?
- Range Partitioning:
  - Split data up into N ranges.
- Hash Partitioning
  - Split data up based on hash value.
- Round Robin
  - Value 1 to machine 1, 2 to machine 2, etc…

Parallel Sorts

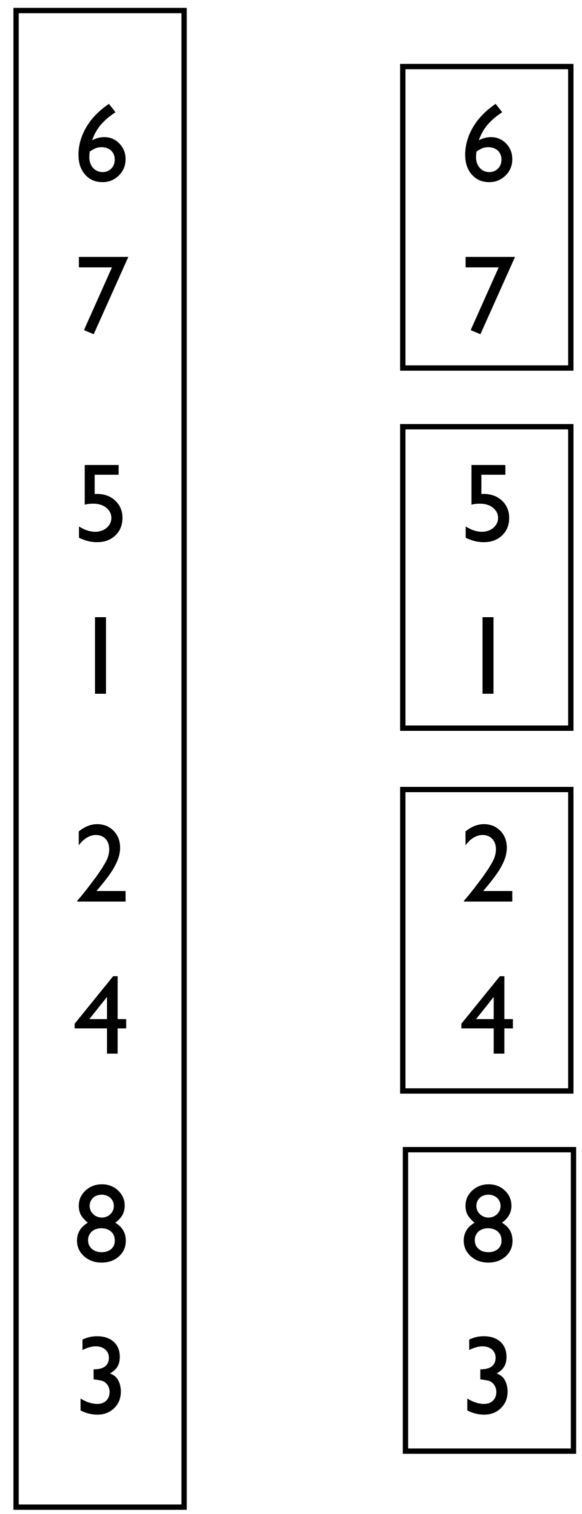
- Record as of March 2013: TritonSort (UCSD)
  - 0.725 TB/Min using 52 nodes
    - 1 Trillion Records in ~2.5 min
  - http://sortbenchmark.org
  - http://sortbenchmark.org/2011_06_tritonsort.pdf

Parallel Sorts

6
7
5
1
2
4
8
3

Parallel Sorts

Partition



Parallel Sorts

Partition

Local Sort

6	6
7	7
5	1
1	5
2	2
4	4
8	3
3	8

Parallel Sorts

Partition

Local Sort

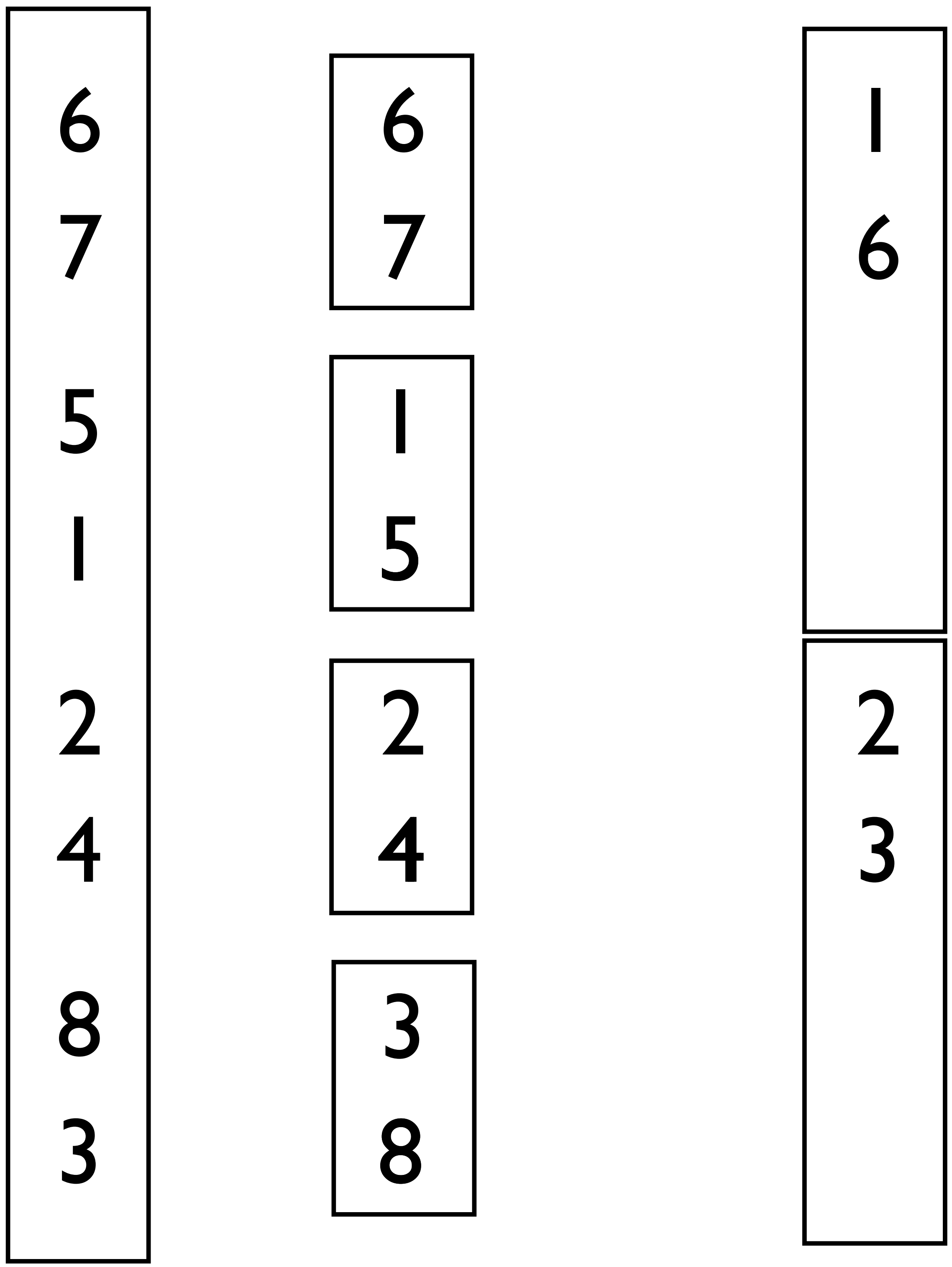
6	6
7	7
5	1
1	5
2	2
4	4
8	3
3	8

Parallel Sorts

Partition

Local Sort

Merge Sort

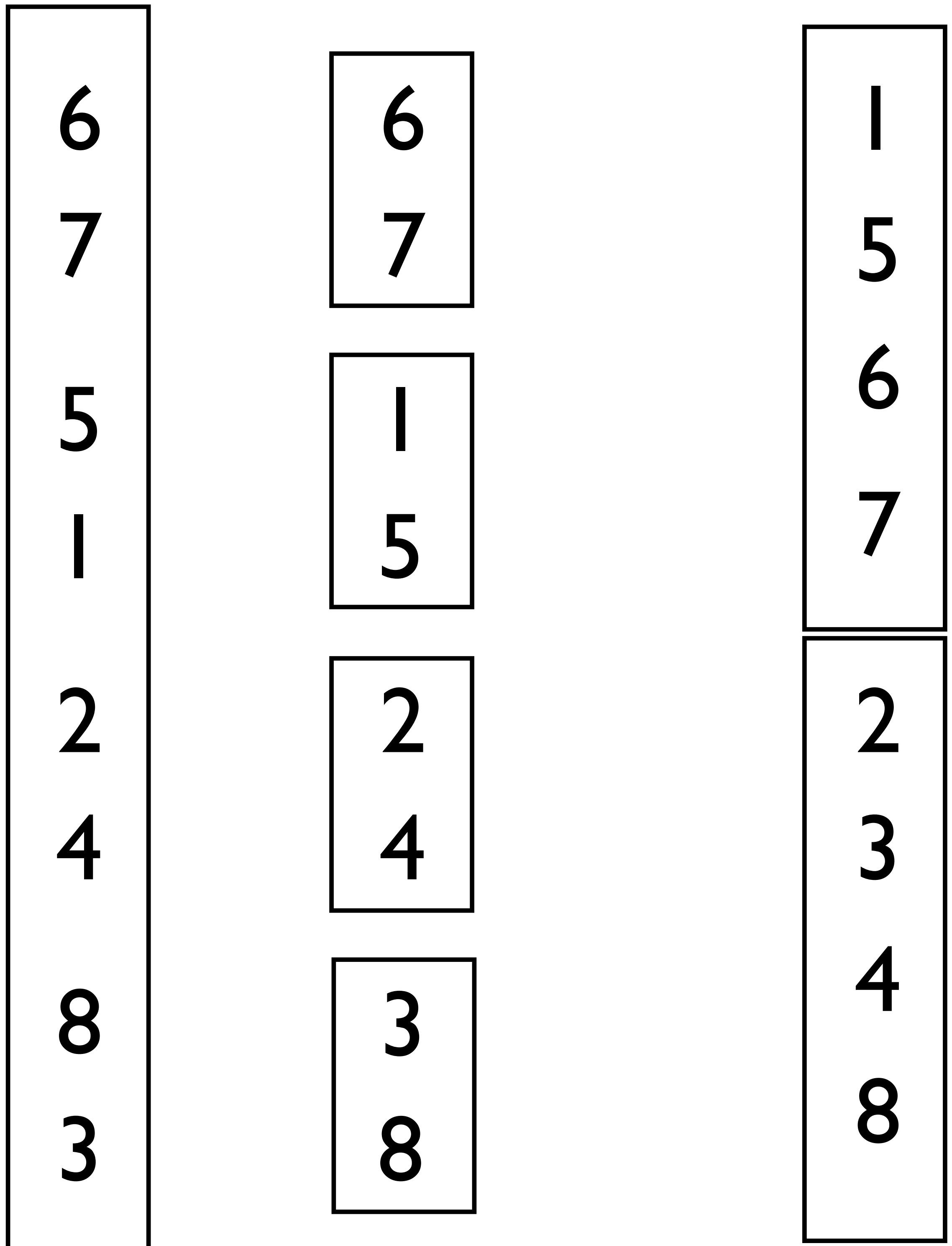


Parallel Sorts

Partition

Local Sort

Merge Sort

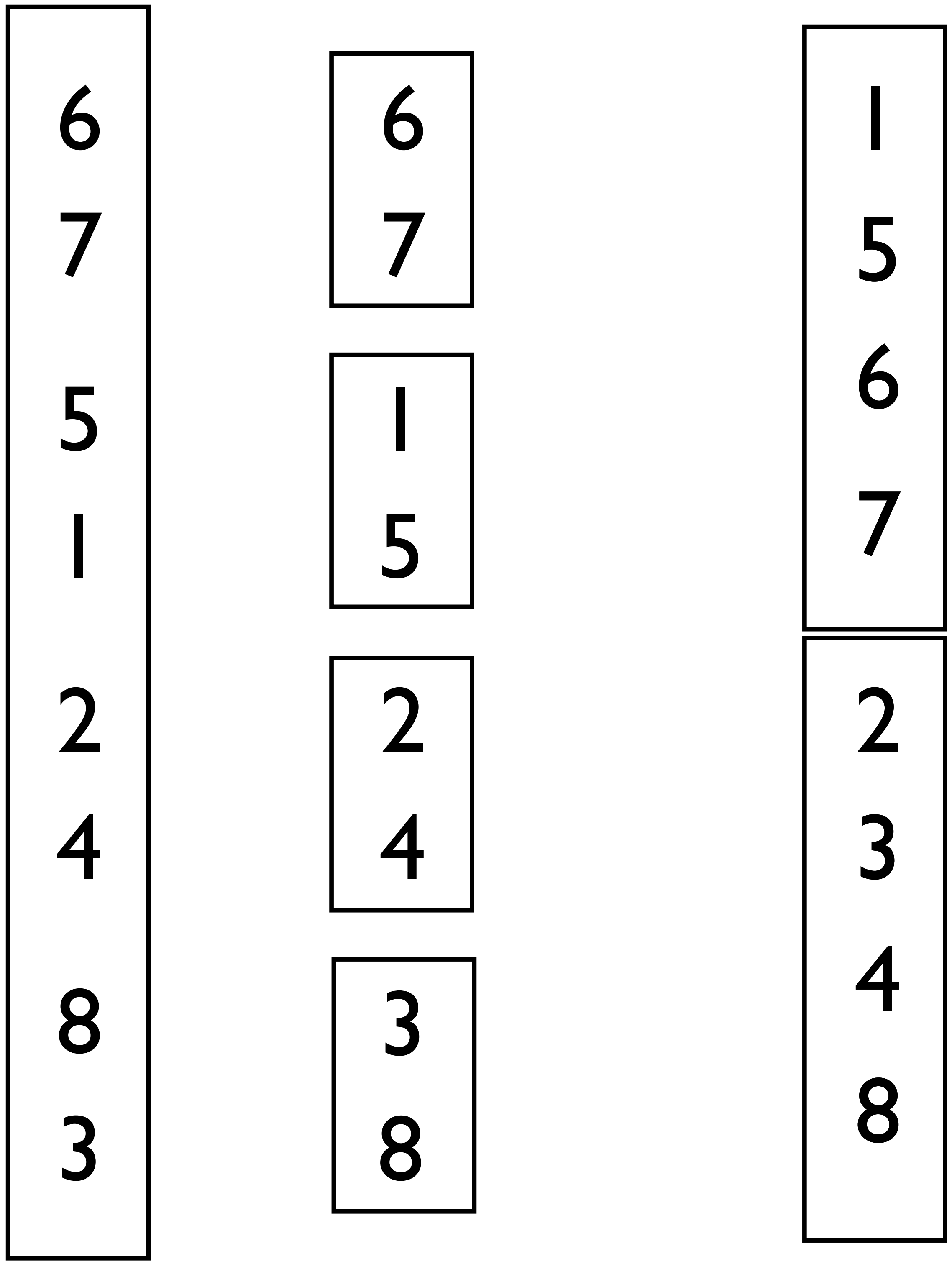


Parallel Sorts

Partition

Local Sort

Merge Sort



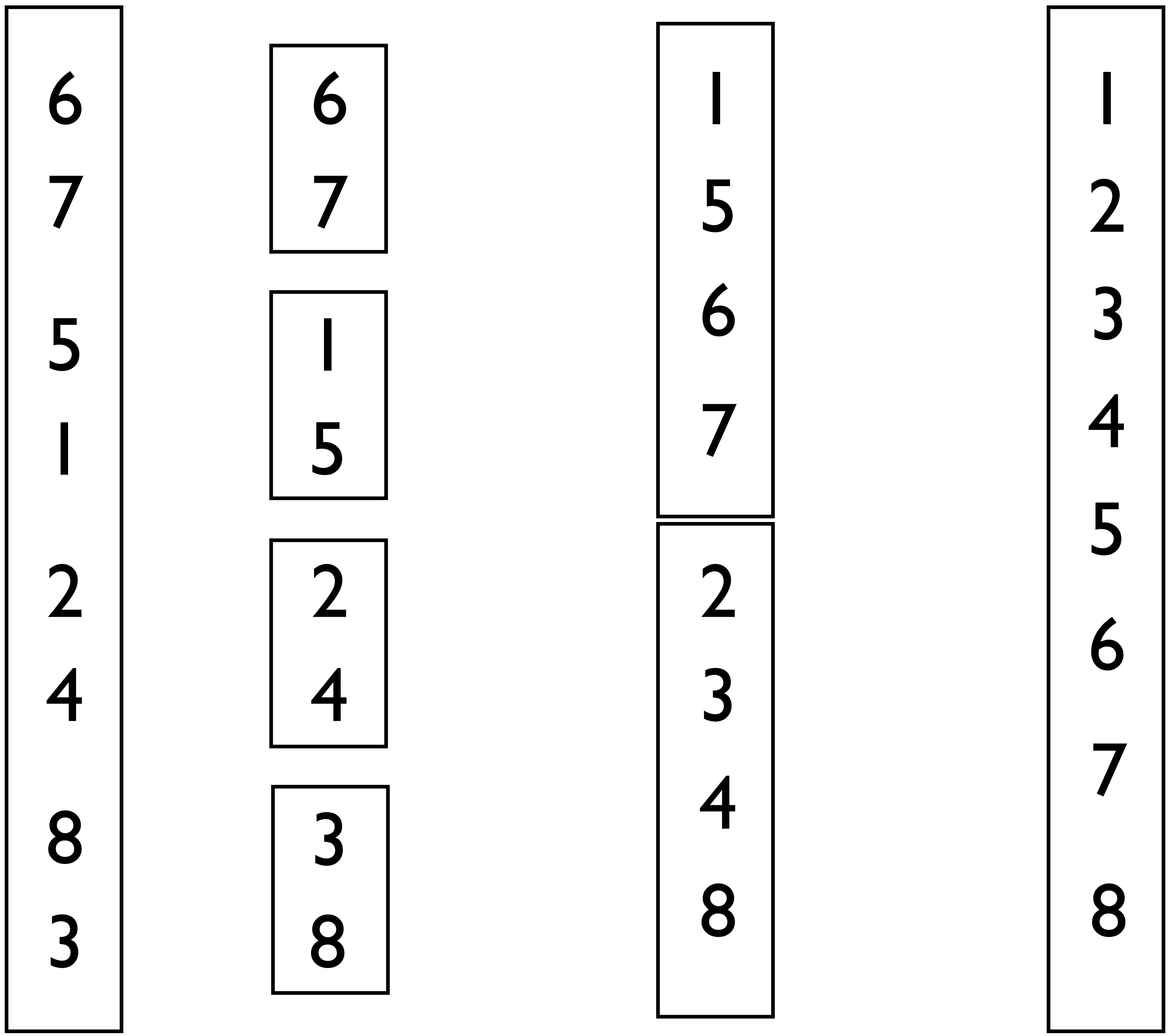
Parallel Sorts

Partition

Local Sort

Merge Sort

Recur



Parallel Sorts

- General Strategy
  - Scan in parallel
  - Range partition to other nodes
  - As tuples come in, sort locally
- Problem: Skew!
  - How do we pick partition boundaries?

Parallel Scans

- Scan the data separately on each machine
- Merge the results together
  - E.g., compute SUM(X) on each machine
  - SUM all computed SUM(X)
- May not need all machines to participate.
- We can build an index at each site.

Parallel Aggregates

- Aggregate Functions:
  - Distributive: F(A,B,C,D) = F(F(A,B),F(C,D))
  - Algebraic: F(A,B,C,D) = G(H(A,B),H(C,D))
  - Holistic: Can't decompose F.
- What are some examples of each?
- How do we exploit this for ||ism?

Parallel Group By

- Compute partial aggregates locally for each group.
- Use hash group-by columns to partition groups.
- Send partial aggregate to a site for each group.
- Each group's site merges partial aggregates.

Parallel Joins-NLJ

- Analogous to Block-NLJ
- One site computes each block-pair.

image credit: openclipart.org

Parallel Joins-NLJ

- Analogous to Block-NLJ
- One site computes each block-pair.

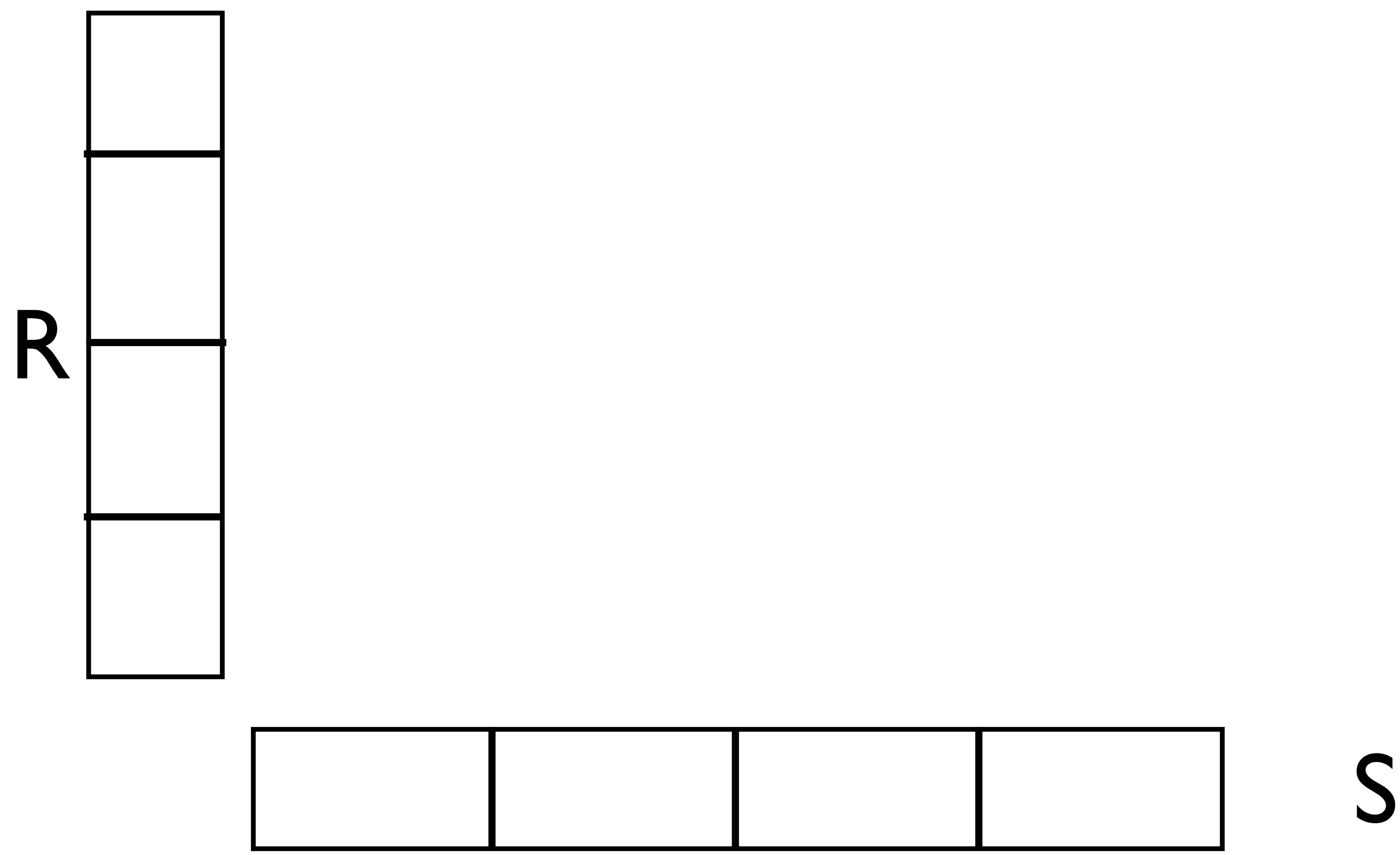


image credit: openclipart.org

Parallel Joins-NLJ

- Analogous to Block-NLJ
- One site computes each block-pair.

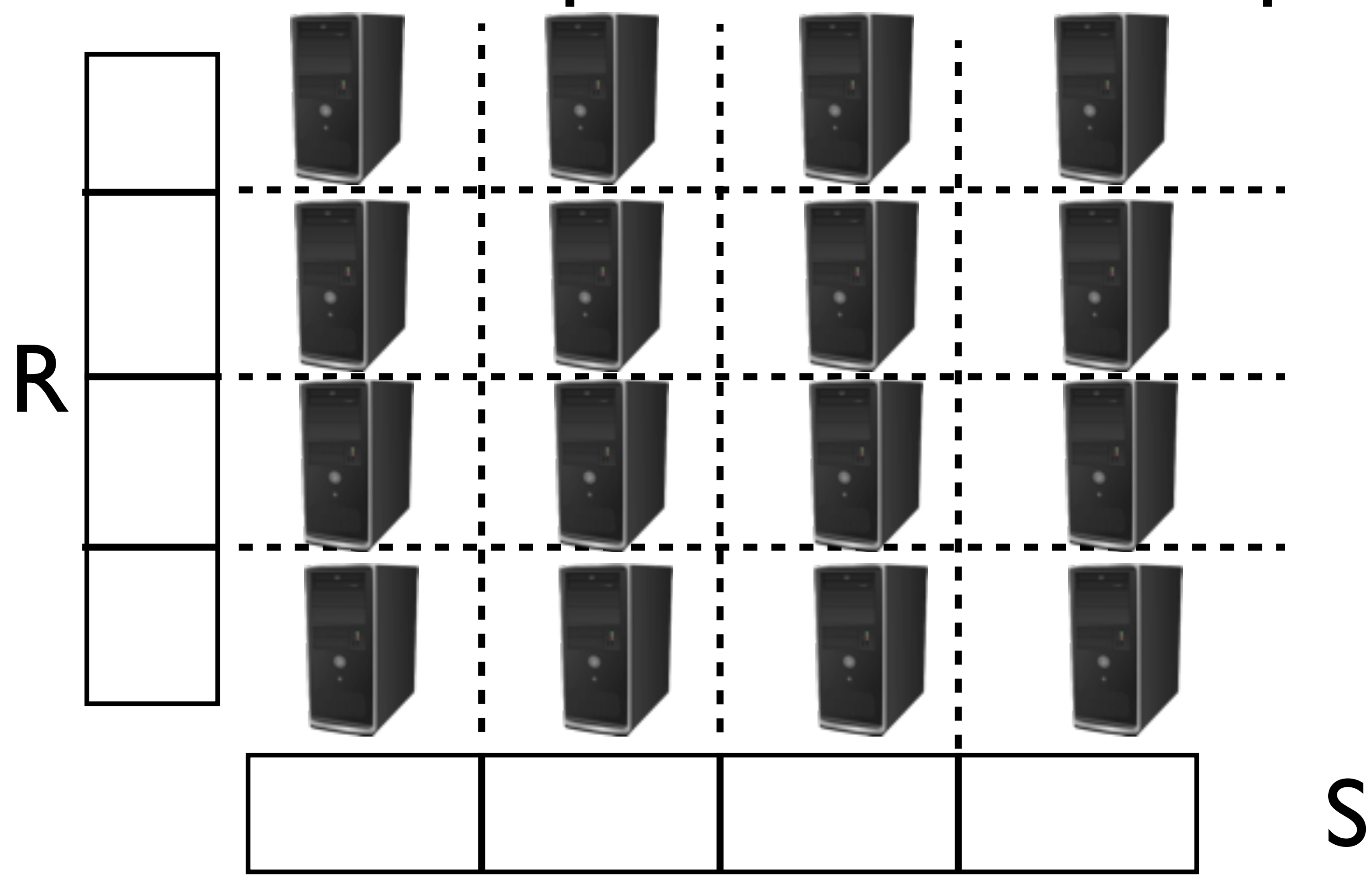


image credit: openclipart.org

Parallel Joins-NLJ

- **Optimization**: Skip blocks that are guaranteed to be empty

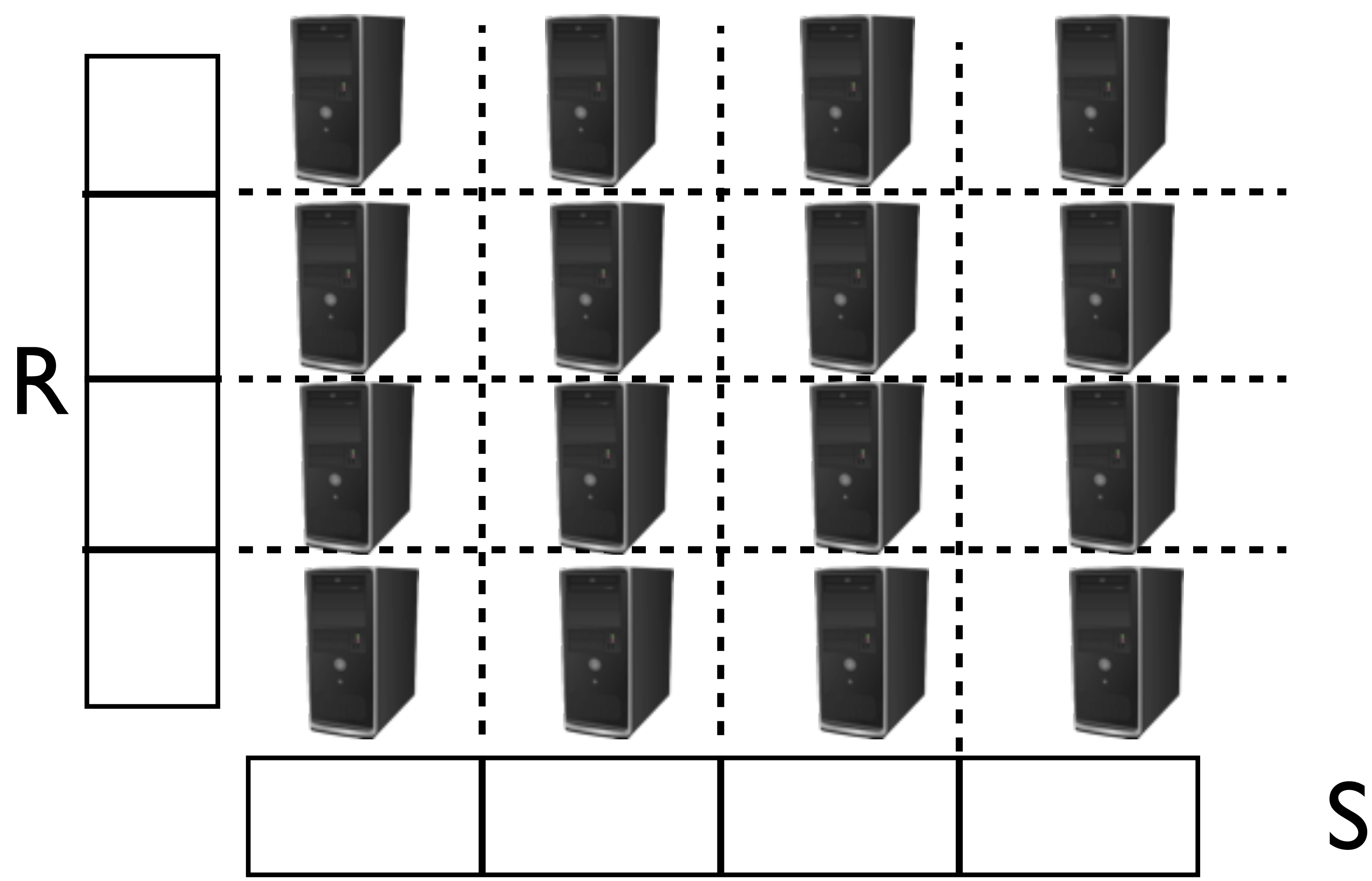
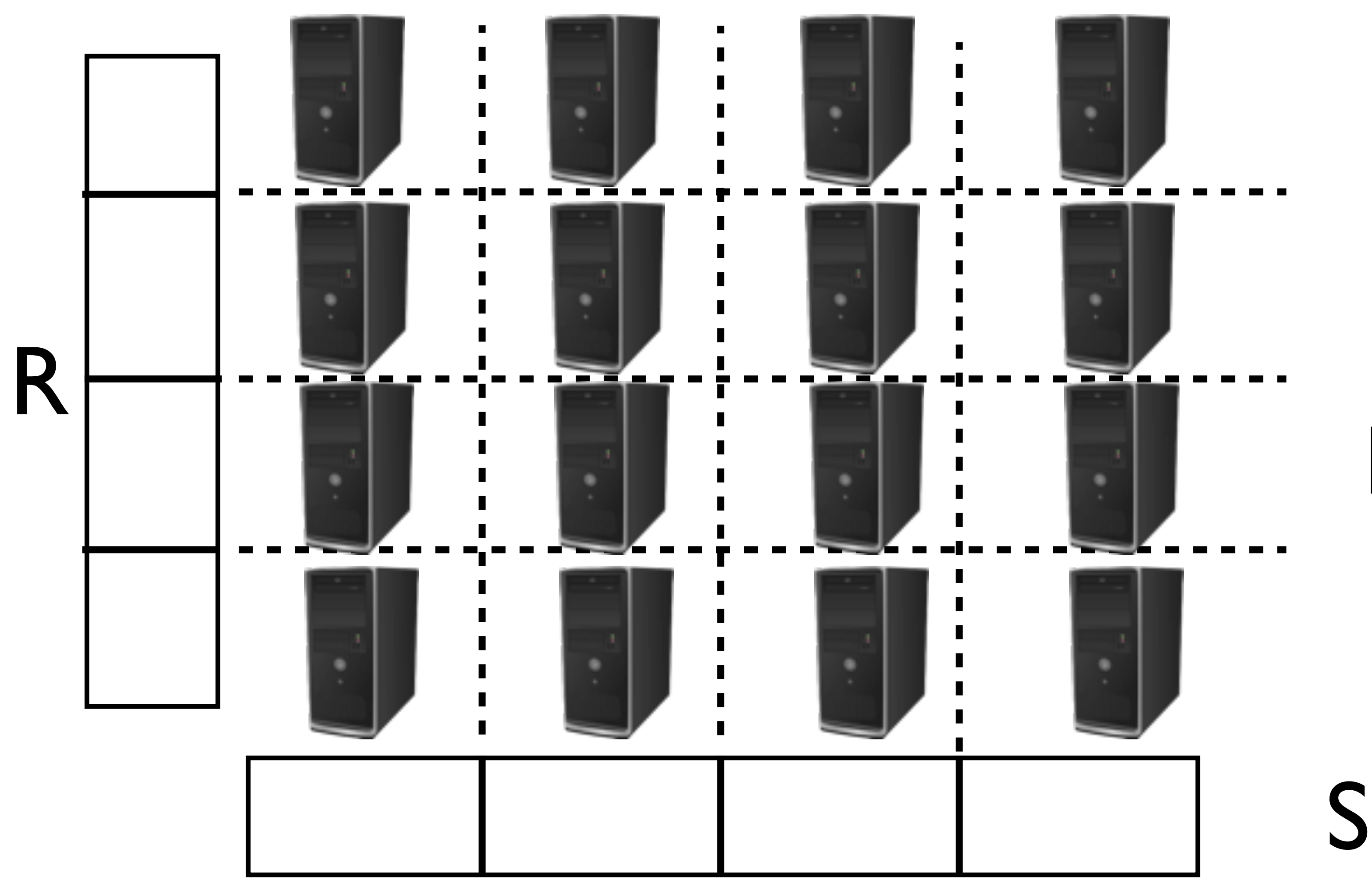


image credit: openclipart.org

Parallel Joins-NLJ

- **Optimization**: Skip blocks that are guaranteed to be empty




```
SELECT …
FROM R,S
WHERE R.A>S.B
```

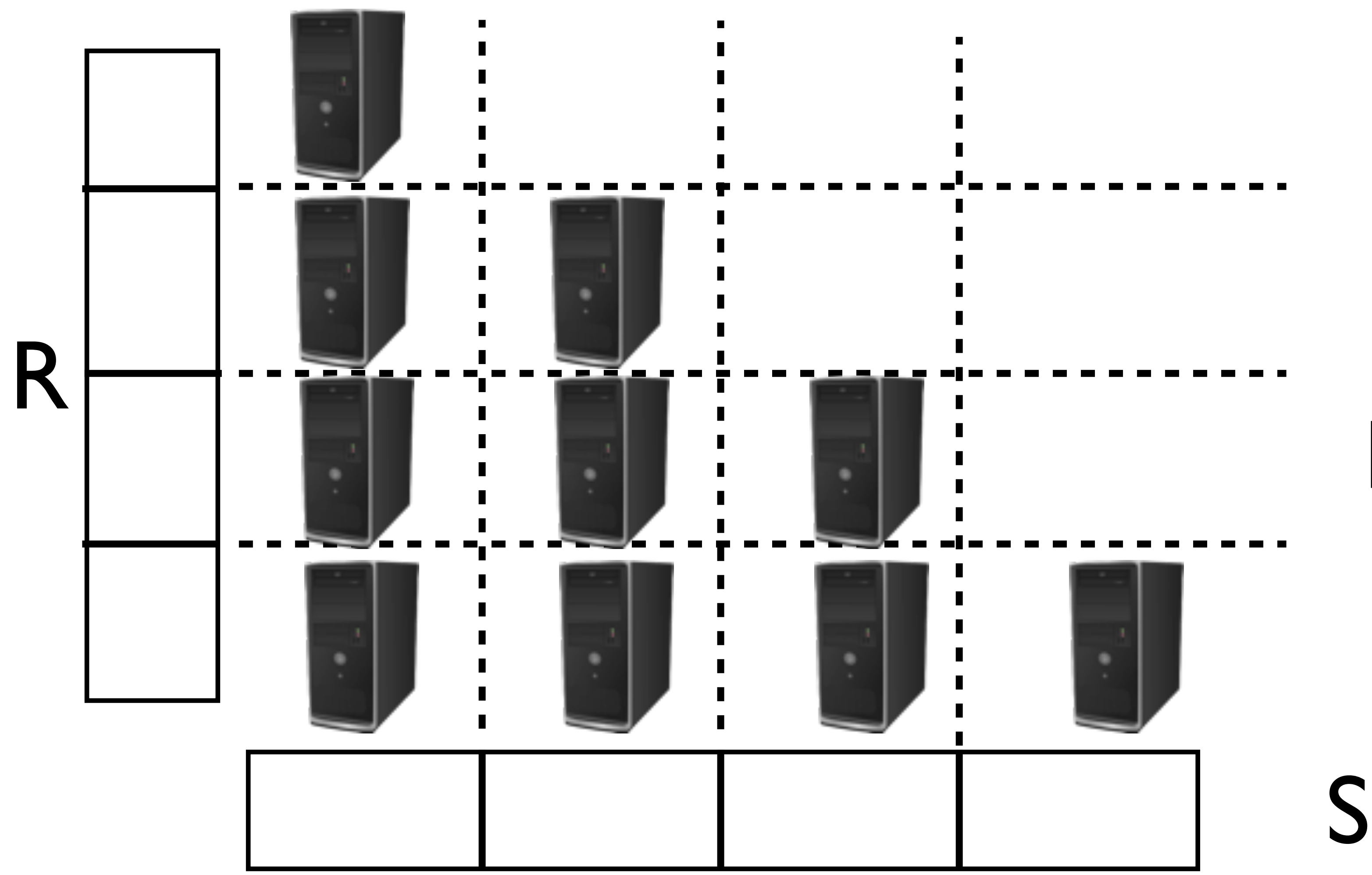
R: Range on A

S: Range on B

image credit: openclipart.org

Parallel Joins-NLJ

- **Optimization**: Skip blocks that are guaranteed to be empty




```
SELECT …
FROM R,S
WHERE R.A>S.B
```

R: Range on A

S: Range on B

image credit: openclipart.org

Parallel Joins-NLJ

- **Limitation**: Requires a large amount of network communication.
  - Why does this happen?
  - Why is this a problem?
  - What can be done about it?

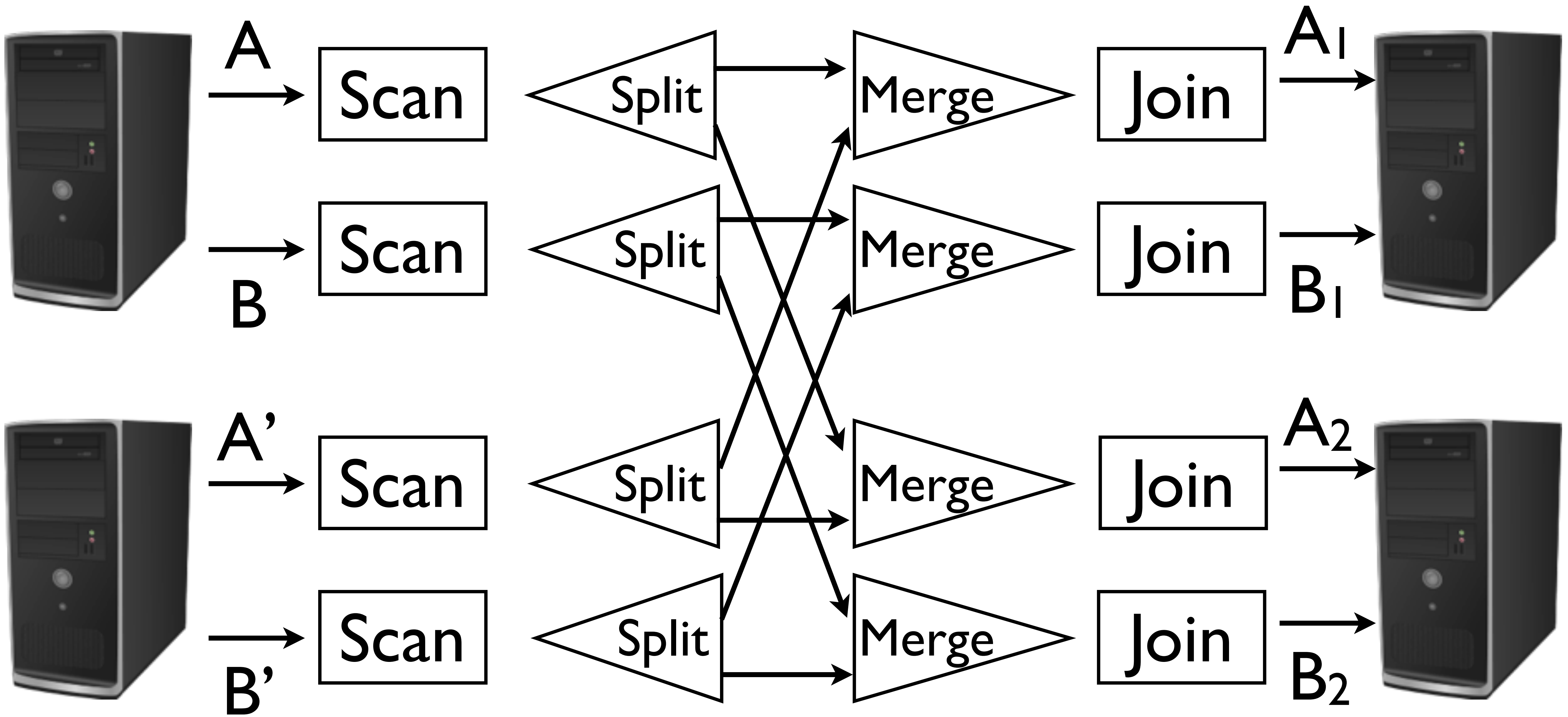
Parallel Joins-Merge Join

- Case 1: Identical Range Partitioning
  - Devolves to Single-Site Sort/Merge
- Case 2: Different Partitioning Schemes
  - Sorting gives range partitions! (what about skew?)
- Case 3: Similar Range Partitioning
  - How would you handle this case?

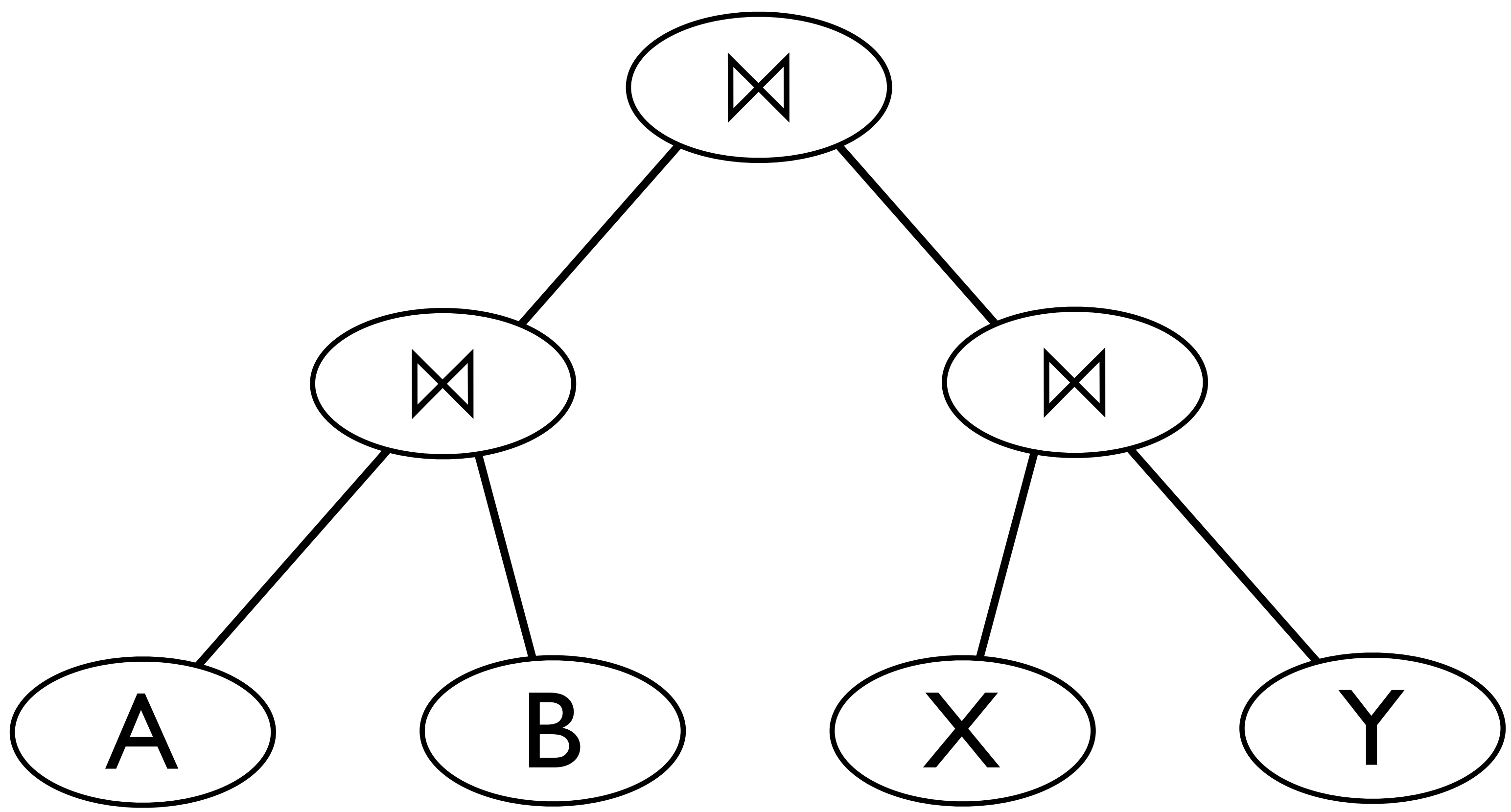
Parallel Joins-Hash

- Phase 1: Partition/Split the data by hash
  - Each partition goes to a different site.
  - (should produce roughly equivalent workloads)
- Phase 2: Join
  - Perform Joins locally
- Almost always the fastest for Equi-Join
  - Common algorithm for equi-joins in Map/Reduce.

Parallel Joins-Hash



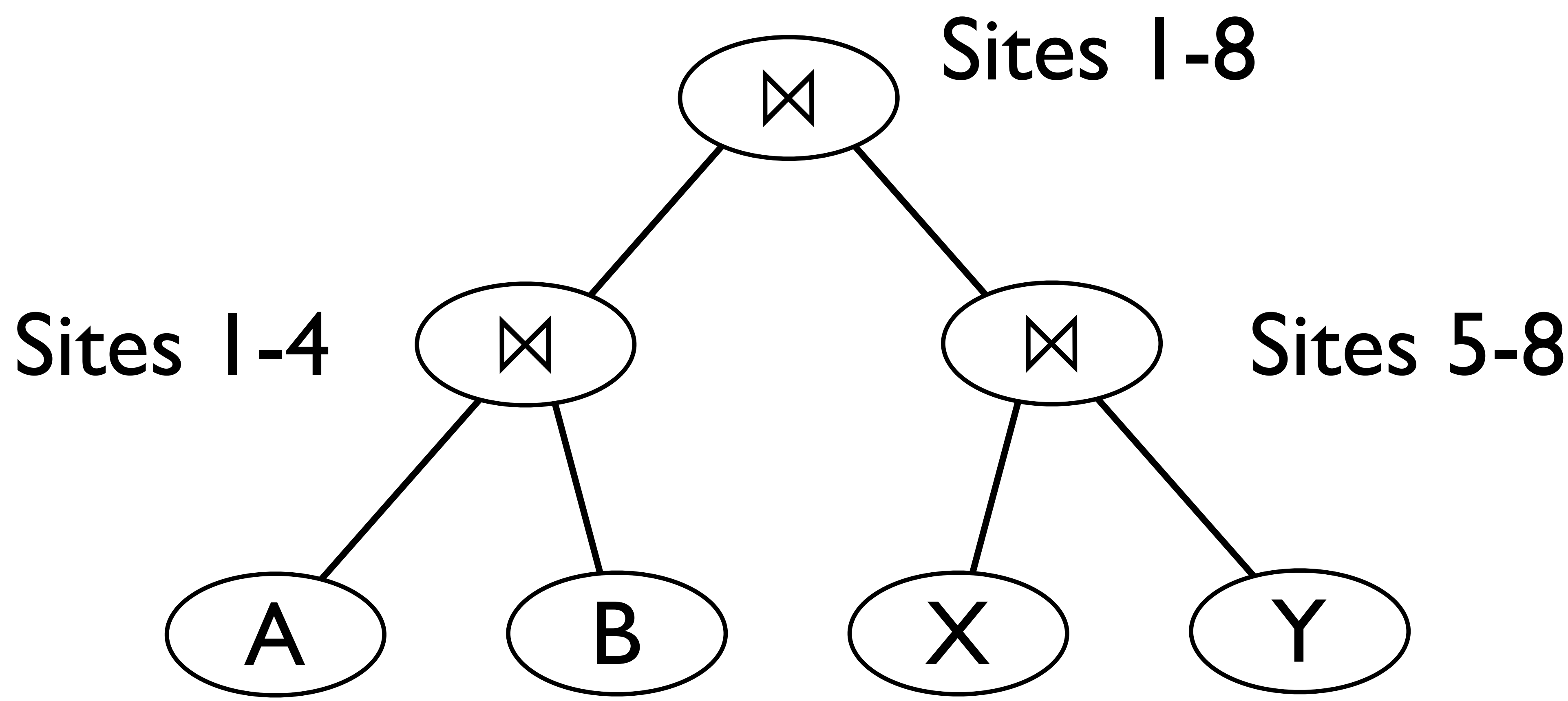
Multi-Stage Query Plans



The Partitioning Phase (Phase 1) of Hash Join is Blocking!

How do we get Operator-Level Parallelism?

Multi-Stage Query Plans



The Partitioning Phase (Phase 1) of Hash Join is Blocking!

How do we get Operator-Level Parallelism?

Optimizing || Query Plans

- Building a fast parallel query execution engine is (relatively) easy.
- Optimizing queries for ||ism is much harder.
  - Even more variables to consider.
  - Optimizer complexity is high!
  - Lots of research being done in this area.

Optimizing || Query Plans

- Common Approach: 2 Phase Optimizer
  - Phase 1: Pick best sequential plan.
  - Phase 2: Assign operators to sites.
    - “Decorate” query tree as above.
- What’s wrong with this?

Example

- SELECT * FROM Officers WHERE Name < “Spock”
- ‘Officers’ partitioned on name with secondary index on name as well.
- Sites 1-5, all < ‘Spock’ : Table Scan Best Access Path
- Site 6: ‘Scott’ through ‘Uhura’ : Index Scan!