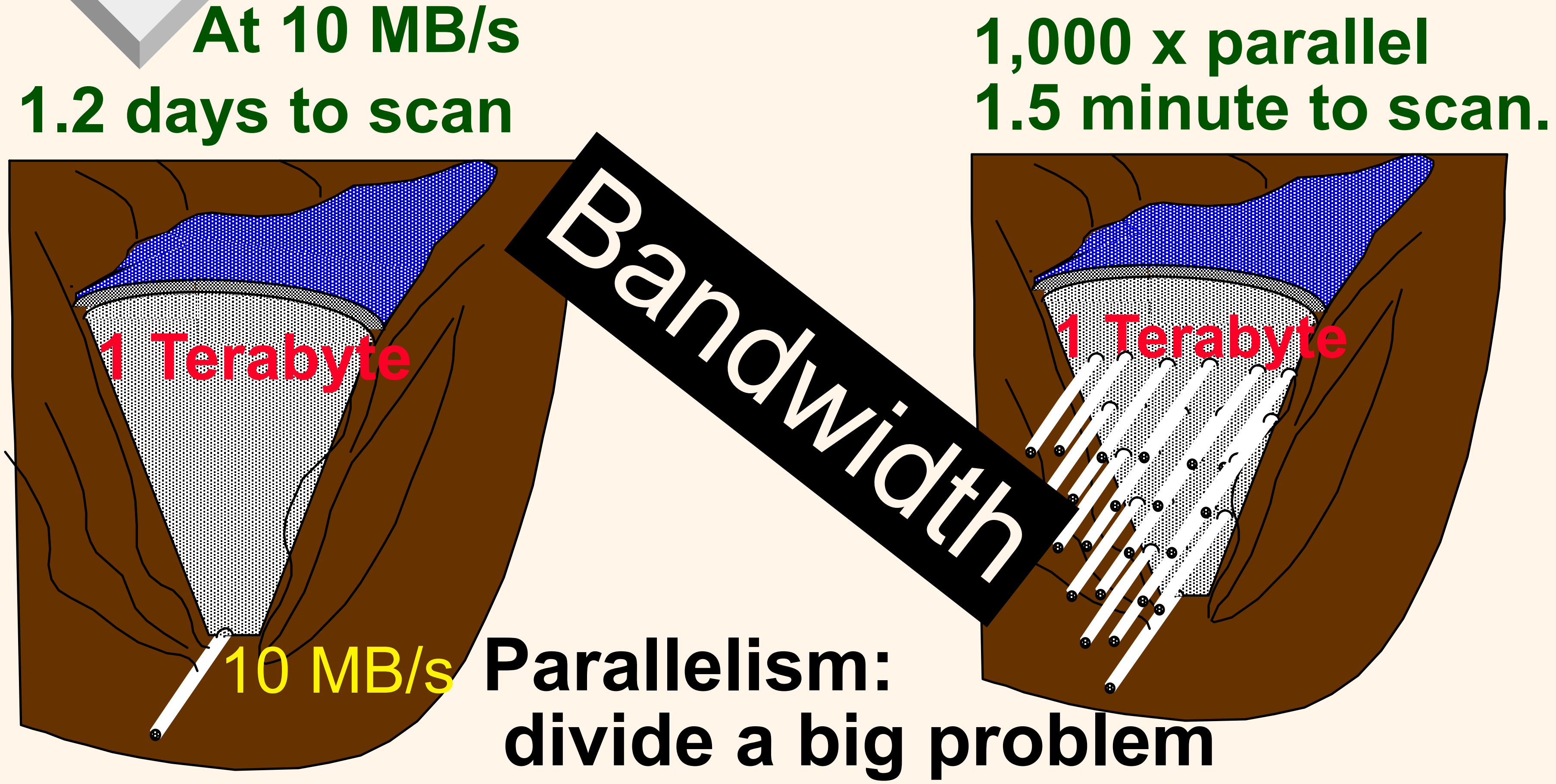
Parallel DBMS

Chapter 22, Sections 22.1–22.6

Why Parallel Access To Data?



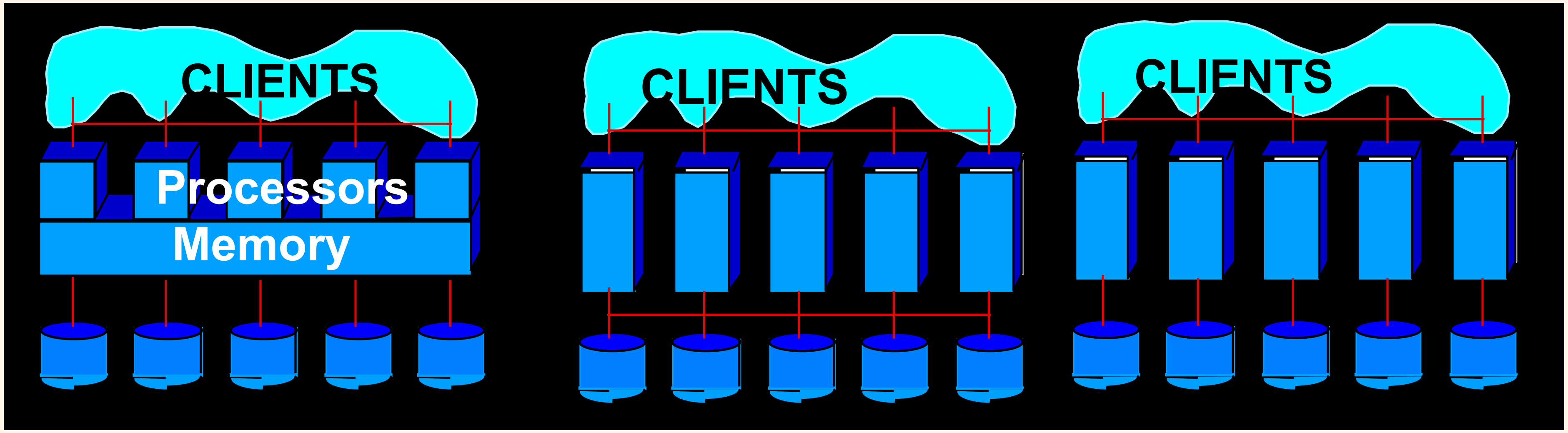
Parallelism:
divide a big problem into many smaller ones to be solved in parallel.

Parallel DBMS: Introduction

- Centralized DBMSs assume that
  - Processing of individual transactions is sequential
  - Control and/or data are maintained at one single site
- These assumptions have been relaxed in recent decades:
  - **Parallel DBMSs:**
    - Use of parallel evaluation techniques; parallelization of various operations such as data loading, index construction, and query evaluations.
    - Data may still be centralized; distribution dictated solely by performance considerations
  - **Distributed DBMSs:**
    - Use of both control and data distribution; data and control are dispersed and stored across several sites
    - Data distribution also dictated by considerations of increased availability and local ownership

Architectures for Parallel Databases

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



Easy to program
Expensive to build
Difficult to scaleup

Sequent, SGI, Sun

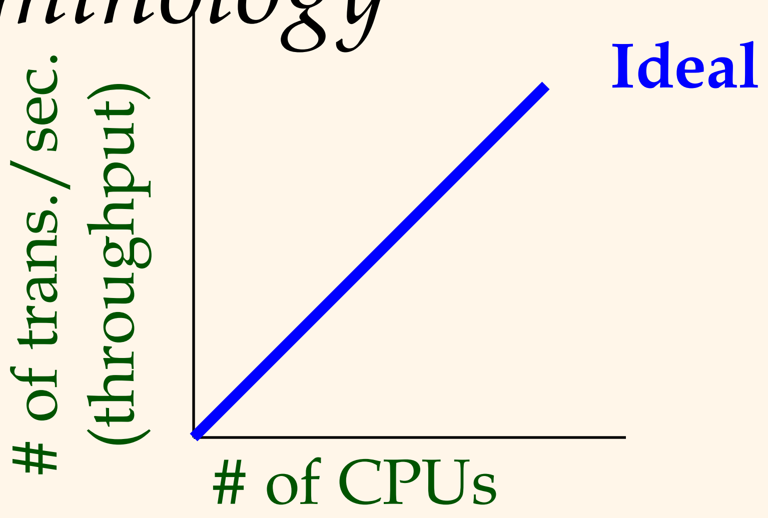
VMScluster, Sysplex

Hard to program
Cheap to build
Easy to scaleup

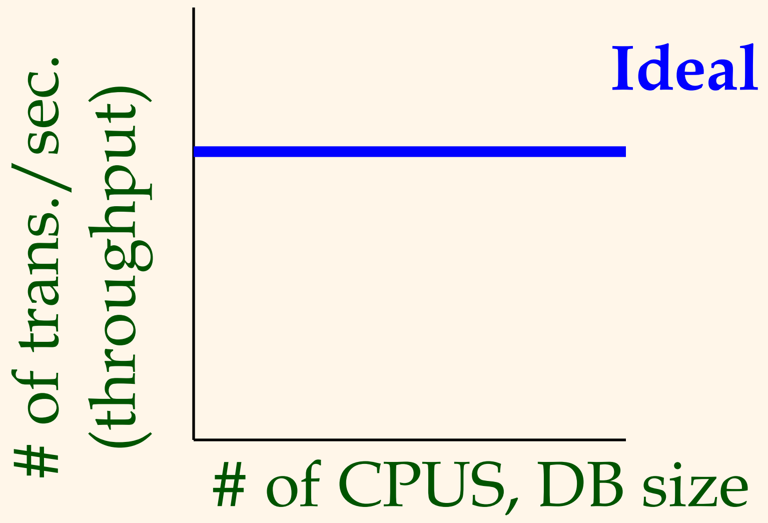
Tandem, Teradata, SP2

Some Parallelism Terminology

- Speed-Up
  - For a given amount of data, more resources (CPUs) means proportionally more transactions processed per second.

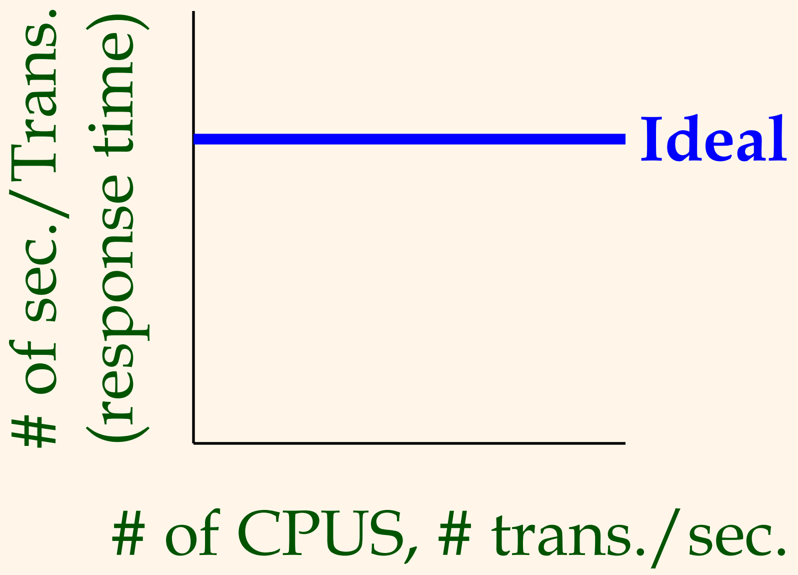


- Scale-Up with DB size
  - If resources increased in proportion to increase in data size, # of trans./sec. remains constant.



Some Parallelism Terminology

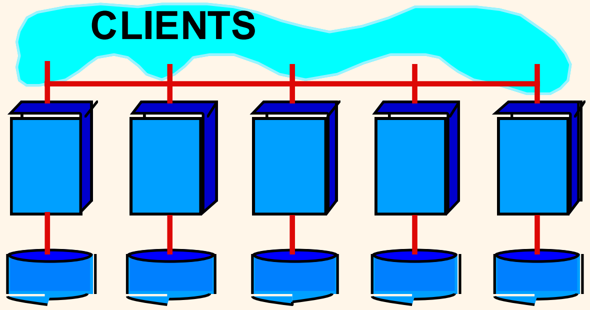
- Scale-Up
  - If resources increased in proportion to increase in # of trans./sec., response time remains constant.



What Systems Work Which Way ?
(as of 9/1995)

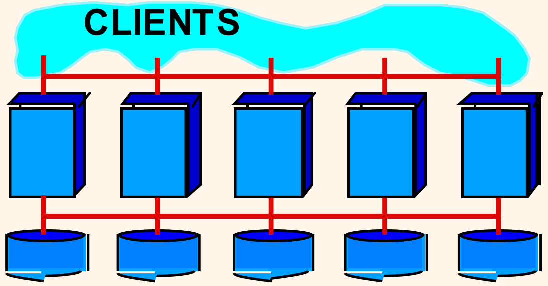
Shared Nothing

Teradata:	400 nodes
Tandem:	110 nodes
IBM / SP2 / DB2:	128 nodes
Informix/SP2	48 nodes
ATT & Sybase	? nodes



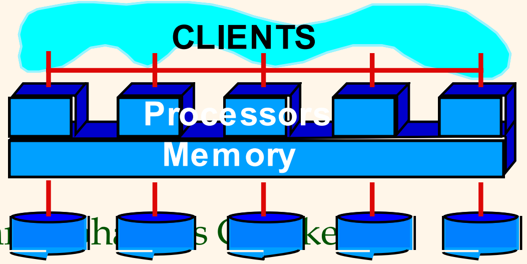
Shared Disk

Oracle	170 nodes
DEC Rdb	24 nodes



Shared Memory

Informix	9 nodes
RedBrick	? nodes

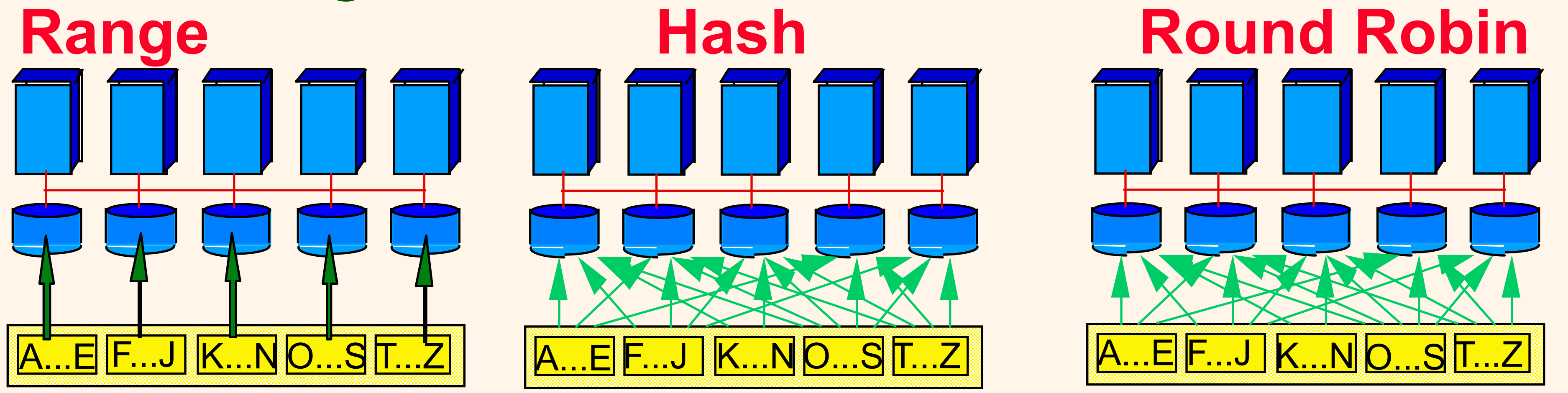


Different Types of DBMS Parallelisms

- 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 parallelism

Automatic Data Partitioning

Partitioning a table:



Good for equijoins, range queries, Selections, group-by

Good for equijoins

Good to spread load

Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning

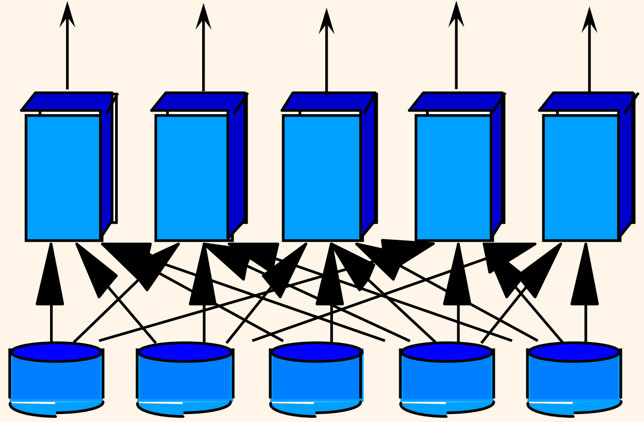
Parallelizing Existing Code for Evaluating a Relational Operator

- How to readily parallelize existing code to enable sequential evaluation of a relational operator?
  - Idea: use parallel data streams
  - Details:
    - MERGE streams from different disks or the output of other operators to provide input streams for an operator
    - SPLIT output of an operator to parallelize subsequent processing
- A **<u>parallel evaluation plan</u>** is a dataflow network of relational, merge, and split operators.
  - Merge and split should have buffering capabilities
  - They should regulate the output of relational operators

Parallel Scanning and Bulk Loading

- Scanning
  - Pages of a relation are read in parallel, and, if the relation is partitioned across many disks, retrieved tuples are merged.
  - Selection of tuples matching some condition may not require all sites if range or hash partitioning is used.
- Bulk loading:
  - Indexes can be built at each partition.
  - Sorting of data entries required for building the indexes during bulk loading can be done in parallel.

Parallel Sorting



- Simple idea:
  - Let each CPU sorts the portion of the relation located on its local disk;
  - then, merge the sorted sets of tuples
- Better idea:
  - First scan in parallel and redistribute the relation by range-partitioning all tuples; then each processor sorts its tuples:
    - The CPU collects tuples until memory is full
    - It sorts the in-memory tuples and writes out a run, until all incoming tuples have been written to sorted runs on disk
    - The runs on disk are then merged to get the sorted version of the portion of the relation assigned to the CPU
    - Retrieve the entire sorted relation by visiting the CPUs in the range-order to scan the tuples.
  - Problem: how to avoid ***skew*** in range-partition!
  - Solution: "sample" the data at start and sort the sample to determine partition points (splitting vector).

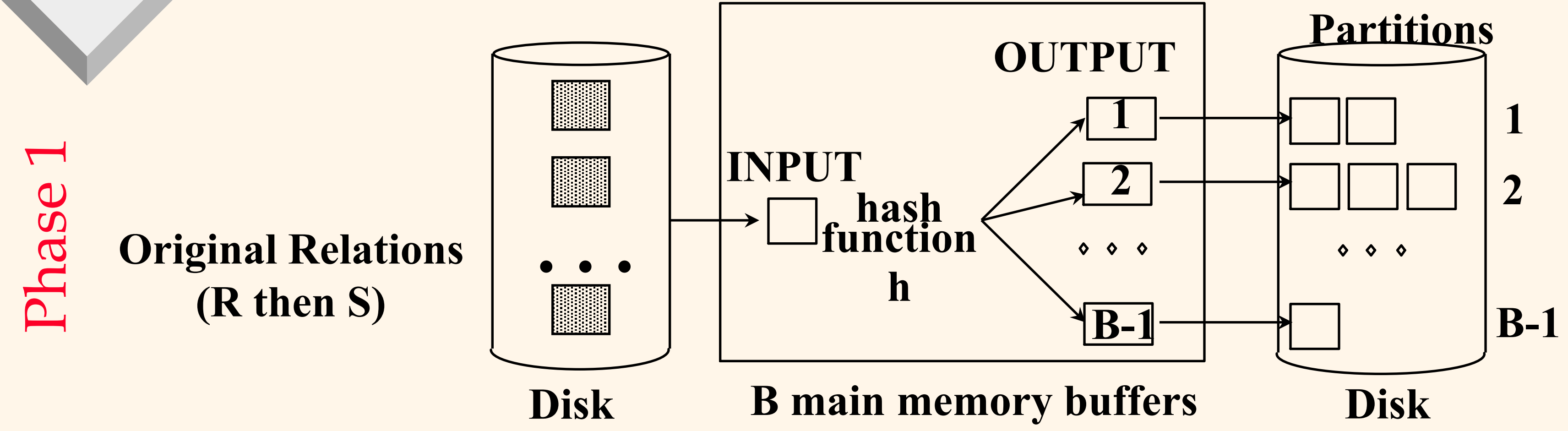
Parallel Joins: Range-Partition

- Assumptions: *A* and *B* are initially distributed across many processors, and *k* processors are availoable.
- Algorithm to join relations A and B on attribute age:
  1. At each processor where there are subsets of *A* and/or *B*, divide the range of *age* into *k* disjoint subranges and place partition *A* and *B* tuples into partitions corresponding to the subranges.
  2. Assign each partition to a processor to carry out a local join. Each processor joins the *A* and *B* tuples assigned to it.
  3. Tuples scanned from each processor are split, with a split operator, into output streams, depending on how many processors are available for parallel joins.
  4. Each join process receives input streams of *A* and *B* tuples from several processors, with merge operators merging all inputs streams from *A* and *B*, respectively.

Parallel Joins: Hash-Partitions

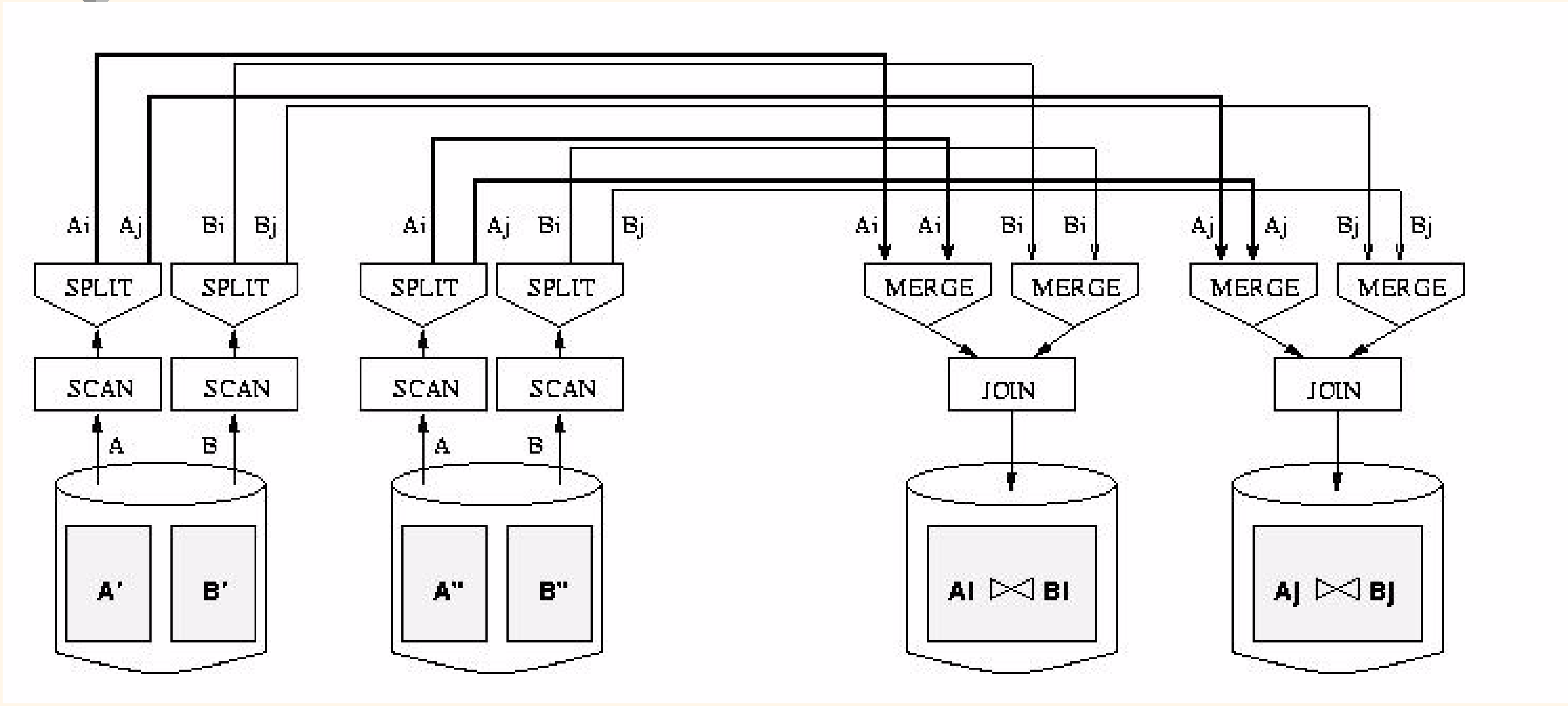
- Algorithm for hash-partition: Step 1 on previous slide changes to:
  1. At each processor where there are subsets of *A* and/or *B*, all local tuples are retrieved and hashed on the *age* attribute into *k* disjoint partitions, with the same hash function h used at all sites.
- Using range partitioning leads to a parallel version of Sort-Merge Join.
- Using hash partitioning leads to a parallel version of Hash Join.

Parallel Hash Join



- In first phase, 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



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

Improved Parallel Hash Join

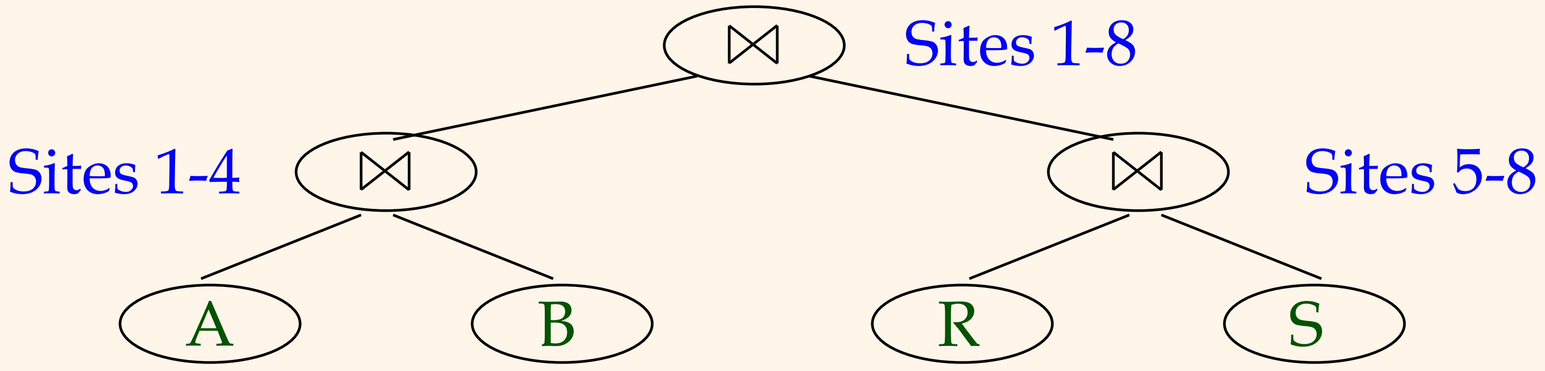
- Assumptions:
  - *A* and *B* are very large, which leads to the size of each partition still being too large, which in turns leads to high local cost for processing the "smaller" joins.
  - *k* partitions, *n* processors and *k*=*n*.
- Idea: Execute all smaller joins of *Ai* and *Bi*, *i=1,…,k*, one after the other, with each join executed in parallel using all processors.

Improved Parallel Hash Join (Cont'd)

- Algorithm:
  1. At each processor, apply hash function *h1* to partition *A* and *B* into partitions *i=1,…,k* . Suppose |*A*|<|*B*|. Then choose *k* such sum of all *k* partitions of *A* fits into aggregate memory of all *n* processors.
  2. For *i=1,…,k* , process join of *i-th* partitions of *A* and *B* by doing this at every site:
     1. Apply 2nd hash function *h2* to all *Ai* tuples to determine where they should be joined send *t* to site *h2(t)*.
     2. Add in-coming Ai tuples to an in-memory hash table.
     3. Apply *h2* to all *Bi* tuples to determine where they should be joined send *t* to site *h2(t)*.
     4. Probe the in-memory hash table with in-coming *Bi* tuples.

Parallel Query Optimization

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



Parallel Query Optimization (Cont'd)

- Common approach: 2 phases
  - Pick best sequential plan (System R algorithm)
  - Pick degree of parallelism based on current system parameters.
- "Bind" operators to processors
  - Take query tree, "decorate" as in previous picture.

Parallel DBMS Summary

- Parallelism natural to query processing:
  - Both pipeline and partition parallelism!
- Shared-Nothing vs. Shared-Mem
  - Shared-disk too, but less standard
  - Shared-mem easy, costly. Doesn't scaleup.
  - Shared-nothing cheap, scales well, harder to implement.
- Intra-op, Inter-op, & Inter-query parallelism all possible.

Parallel DBMS Summary (Cont'd)

- Data layout choices important!
- Most DB operations can be done with partition-parallelism
  - Sort.
  - Sort-merge join, hash-join.
- Complex plans.
  - Allow for pipeline-parallelism, but sorts, hashes block the pipeline.
  - Partition parallelism achieved via bushy trees.

Parallel DBMS Summary (Cont'd)

- Hardest part of the equation: optimization.
  - 2-phase optimization simplest, but can be ineffective.
  - More complex schemes still at the research stage.
- We haven't said anything about transactions, logging.
  - Easy in shared-memory architecture.
  - Takes some care in shared-nothing.