

# Cluster-Based Algorithms for Relational Join

The New Computation Model

More Efficient Joins Via Replication

Optimum Strategies for Special Cases

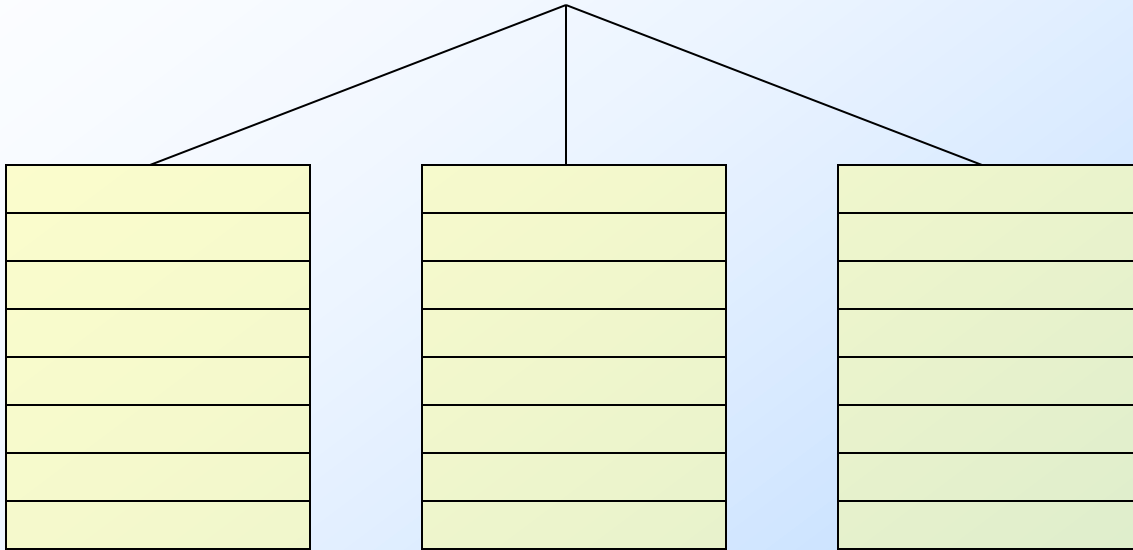
*Joint work with Foto Afrati*

# Next-Generation File Systems

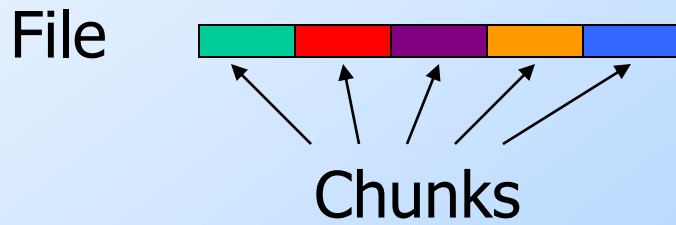
- ◆ Files are very large.
- ◆ They are divided into *chunks*.
  - ◆ Perhaps 16MB to a chunk.
- ◆ Chunks are replicated at several *compute-nodes*.
- ◆ A *master* (possibly replicated) keeps track of all locations of all chunks.

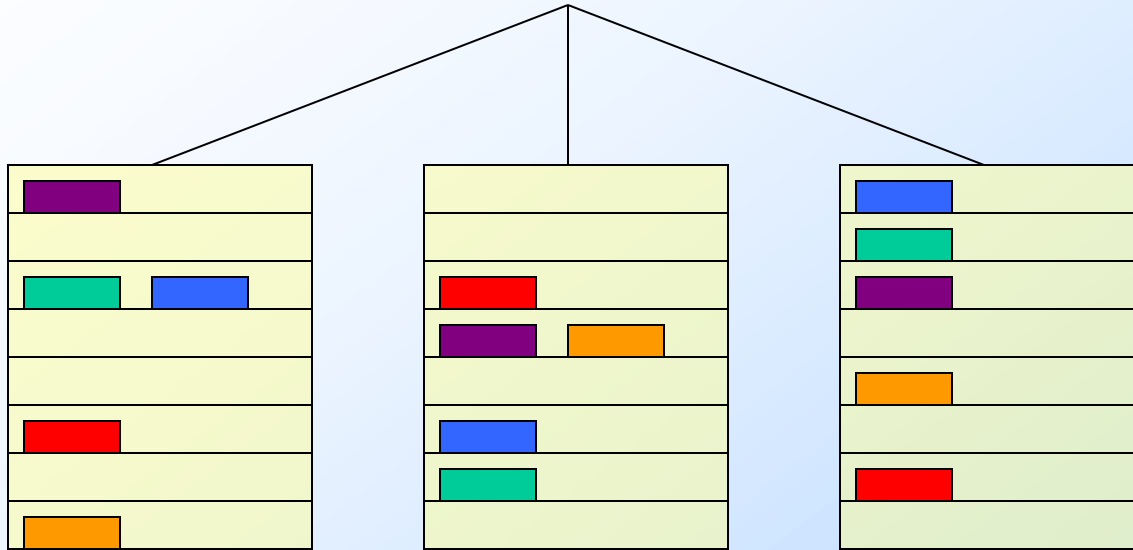
# Compute Nodes

- ◆ Organized into racks.
- ◆ Intra-rack connection typically gigabit speed.
- ◆ Inter-rack connection faster by a small factor.



Racks of Compute Nodes





3-way replication of files, with copies on different racks.

# Implementations

- ◆ *GFS* (Google File System – proprietary).
- ◆ *HDFS* (Hadoop Distributed File System – open source).
- ◆ *KFS* (Kosmix File System).
- ◆ *Clustera* (Implementation at U. Wisconsin).

# Moving Up the Computation Stack

- ◆ *BigTable* (Google retrieval system for objects).
- ◆ *Hadoop* (Open-source implementation of Google's map-reduce).
- ◆ *PIG* (Yahoo! implementation of relational algebra/SQL on top of Hadoop).

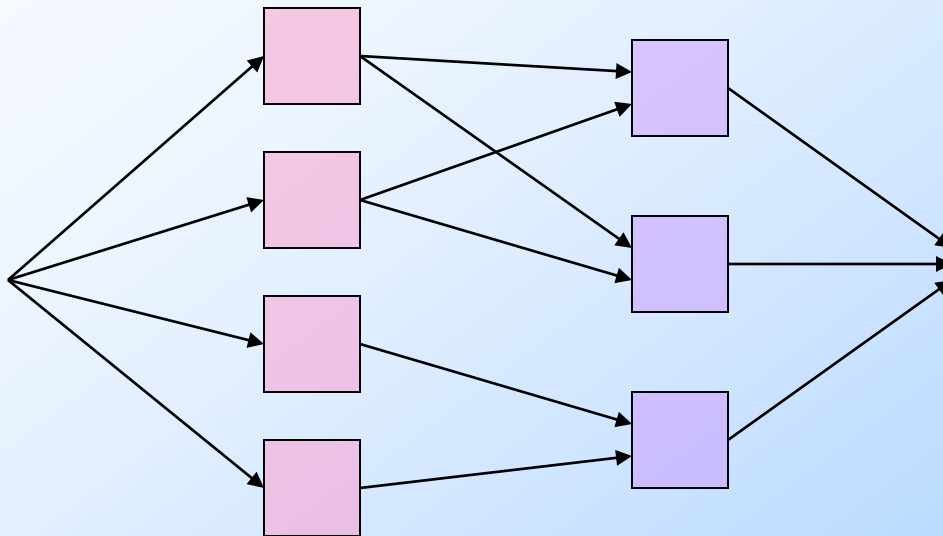
# Reading and Writing Files

- ◆ Order of elements in a file doesn't matter.
  - ◆ **Example:** A file could be a collection of documents; order within a doc matters, but not the order of docs.
- ◆ Parallel reading, writing OK.

# Algorithms

- ◆ *Algorithm* = acyclic collection of processes.
  - ◆ Arc means the process at the tail feeds some data to the head.

# Example Algorithm



# Communication Cost

- ◆ *Communication cost* = sum of sizes of inputs to all processes of an algorithm.
- ◆ Also, *elapsed* communication cost: the maximum over all paths through the acyclic graph of the sizes of the inputs to each of the processes on that path.

# Why Not Count Output Size?

- ◆ Outputs of one process are inputs to at least one other, or are algorithm output.
- ◆ Algorithm outputs tend to be small because users can't make use of too much information.

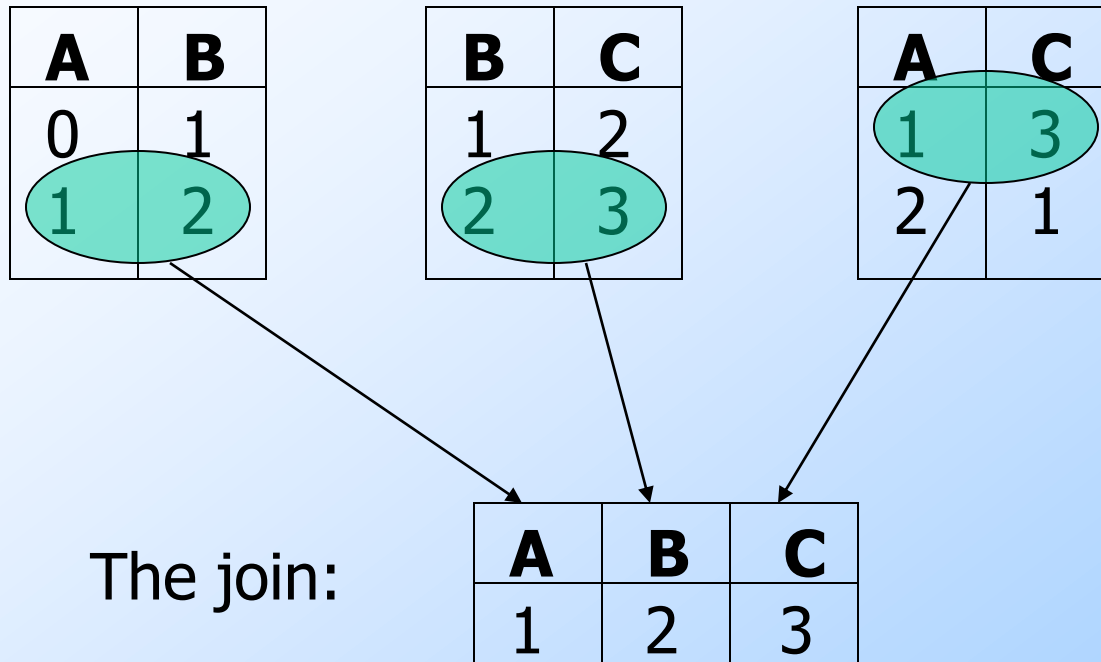
# Why Not Count Processing Time?

- ◆ In the types of applications we discuss, work by a process is usually proportional to input size.
- ◆ And it occurs in main memory, so we can do a lot in the time it takes to get your input over a gigabit line.

# Natural Join of Relations

- ◆ **Given:** a collection of relations, each with attributes labeling their columns.
- ◆ **Find:** Those tuples over all the attributes such that when restricted to the attributes of any relation  $R$ , that tuple is in  $R$ .

# Example: Natural Join



# Map-Reduce Algorithms

- ◆ *Map* processes send inputs to key-value pairs.
  - ◆ “keys” are not necessarily unique.
- ◆ Outputs of Map processes are sorted by key, and each key is assigned to one *Reduce* process.
- ◆ Reduce processes combine values associated with a key.

# Joining by Map-Reduce

- ◆ Suppose we want to compute  $R(A,B) \text{ JOIN } S(B,C)$ , using  $k$  compute nodes.
- ◆  $R$  and  $S$  are each stored in a chunked file.

# Joining by Map-Reduce – (2)

- ◆ Use a hash function  $h$  from B-values to  $k$  buckets.
- ◆ Many Map processes take chunks from  $R$  and  $S$ , and send:
  - ◆ Tuple  $R(a,b)$  to Reduce process  $h(b)$ .
  - ◆ Tuple  $S(b,c)$  to Reduce process  $h(b)$ .

# Joining by Map-Reduce – (3)

- ◆ If  $R(a,b)$  joins with  $S(b,c)$ , then both tuples are sent to Reduce process  $h(b)$ .
- ◆ Thus, their join  $(a,b,c)$  will be produced there and shipped to the output file.

# 3-Way Join

- ◆ Consider a chain of three relations:  
 $R(A, B) \text{ JOIN } S(B, C) \text{ JOIN } T(C, D)$
- ◆ **Example:** R, S, and T are “friends” relations.
- ◆ We could join any two by the 2-way map-reduce algorithm shown, then join the third with the resulting relation.
- ◆ But intermediate joins are large.

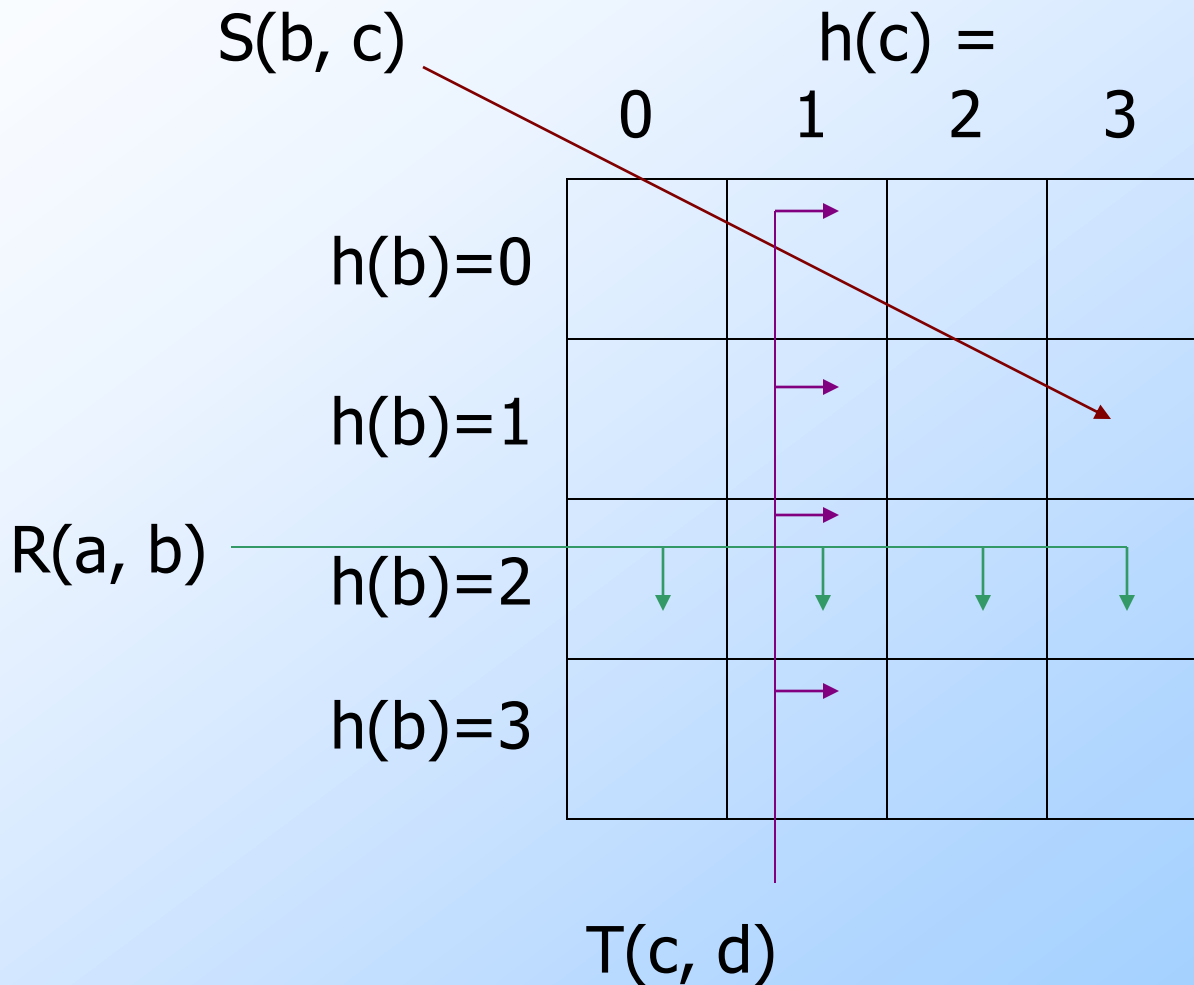
## 3-Way Join – (2)

- ◆ An alternative is to divide the work among  $k = m^2$  Reduce processes.
- ◆ Hash both B and C to  $m$  values.
- ◆ A Reduce process corresponds to a hashed B-value and a hashed C-value.

## 3-Way Join – (3)

- ◆ Each S-tuple  $S(b,c)$  is sent to one Reduce process:  $(h(b), h(c))$ .
- ◆ But each tuple  $R(a,b)$  must be sent to  $m$  Reduce processes  $(h(b), x)$ .
- ◆ And each tuple  $T(c,d)$  must be sent to  $m$  Reduce processes  $(y, h(c))$ .

Example:  $m = 4$ ;  $k = 16$ .



## 3-Way Join – (4)

- ◆ Thus, any joining tuples  $R(a,b)$ ,  $S(b,c)$ , and  $T(c,d)$  will be joined at the Reduce process  $(h(b), h(c))$ .
- ◆ **Communication cost:**  $s + mr + mt$ .
  - ◆ **Convention:** Lower-case letter is the size of the relation whose name is the corresponding upper-case letter.
    - **Example:**  $r$  is the size of  $R$ .

# Comparison of Methods

- ◆ Suppose for simplicity that:
  - ◆ Relations  $R$ ,  $S$ , and  $T$  have the same size  $r$ .
  - ◆ The probability of two tuples joining is  $p$ .
- ◆ The 3-way join has cost  $r(2m+1)$ .
- ◆ Two two-way joins have a cost of:
  - ◆  $3r$  to read the relations, plus
  - ◆  $pr^2$  to read the join of the first two.
  - ◆ **Total** =  $r(3+pr)$ .

## Comparison – (2)

- ◆ 3-way beats 2-way if  $2m+1 < 3+pr$ .
- ◆  $pr$  is the multiplicity of each join.
  - ◆ Thus, the 3-way chain-join is useful when the multiplicity is high.
- ◆ **Example:** relations are “friends”;  $pr$  is about 300.  $m^2 = k$  can be 20,000.
- ◆ **Example:** relations are Web links;  $pr$  is about 15.  $m^2 = k$  can be 64.

# Some Questions

- ◆ When we discussed the 3-way chain-join, we used attributes B and C for the *map-key* (index for the Reduce processes).
- ◆ Why not include A and/or D?
- ◆ Why use the same number of buckets for B and C?

# Share Variables

- ◆ For the general problem, we use a *share variable* for each attribute.
  - ◆ The number of buckets into which values of that attribute are hashed.
- ◆ **Convention:** The share variable for an attribute is the corresponding lower-case letter.
  - ◆ **Example:** the share variable for attribute A is always *a*.

# Share Variables – (2)

- ◆ The product of all the share variables is  $k$ , the number of Reduce processes.
- ◆ The communication cost of a multiway join is the sum of the size of each relation times the product of the share variables for the attributes that *do not* appear in the schema of that relation.

# Example: Minimizing Cost

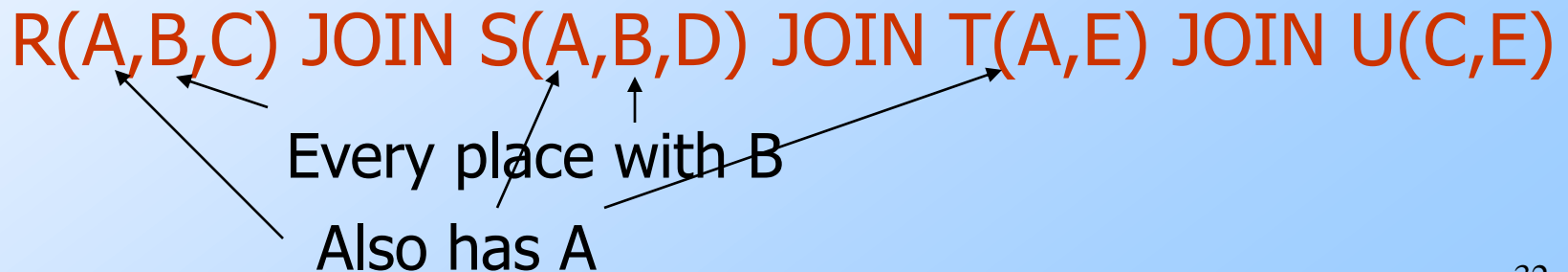
- ◆ Consider the cyclic join  
 $R(A, B) \text{ JOIN } S(B, C) \text{ JOIN } T(A, C)$
- ◆ Cost function is  $rc + sa + tb$ .
- ◆ Construct the Lagrangean:  
 $rc + sa + tb - \lambda(abc - k)$
- ◆ Take derivative wrt each share variable, then multiply by that variable.
  - ◆ Result is 0 at minimum.

# Example – Continued

- ◆  $d/da$  of  $rc + sa + tb - \lambda(abc - k)$  is  $s - \lambda bc$ .
- ◆ Multiply by  $a$  and set to 0:  $sa - \lambda abc = 0$ .
- ◆ **Note:**  $abc = k : sa = \lambda k$ .
- ◆ Similarly,  $d/db$  and  $d/dc$  give:  
 $sa = tb = rc = \lambda k$ .
- ◆ **Solution:**  $a = (krt / s^2)^{1/3}$ ;  $b = (krs / t^2)^{1/3}$ ;  
 $c = (kst / r^2)^{1/3}$ ;
- ◆ Cost =  $rc + sa + tb = 3(krst)^{1/3}$ .

# Dominated Attributes

- ◆ Certain attributes can't be in the map-key.
- ◆ A *dominates* B if every relation of the join with B also has A.
- ◆ Example:



## Example – (2)

R(A,B,C) JOIN S(A,B,D) JOIN T(A,E) JOIN U(C,E)

- ◆ Cost expression:  
 $rde + sce + tbcd + uabd$
- ◆ Since  $b$  appears wherever  $a$  does, if there were a minimum-cost solution with  $b > 1$ , we could replace  $b$  by 1 and  $a$  by  $ab$ , and the cost would *lower*.

# Dominated Attributes – Continued

- ◆ Thus, we do not put any dominated attribute in the map-key.
- ◆ This rule explains why, in the discussion of the chain join

$R(A, B) \text{ JOIN } S(B, C) \text{ JOIN } T(C, D)$

we did not put A or D in the map key.

# Solving the General Case

- ◆ Unfortunately, there are more complex cases than dominated attributes, where the equations derived from the Lagrangean imply a positive sum of several terms = 0.
  - ◆ We can fix, generalizing dominated attributes, but we have to branch on which attribute needs to be eliminated from the map-key.

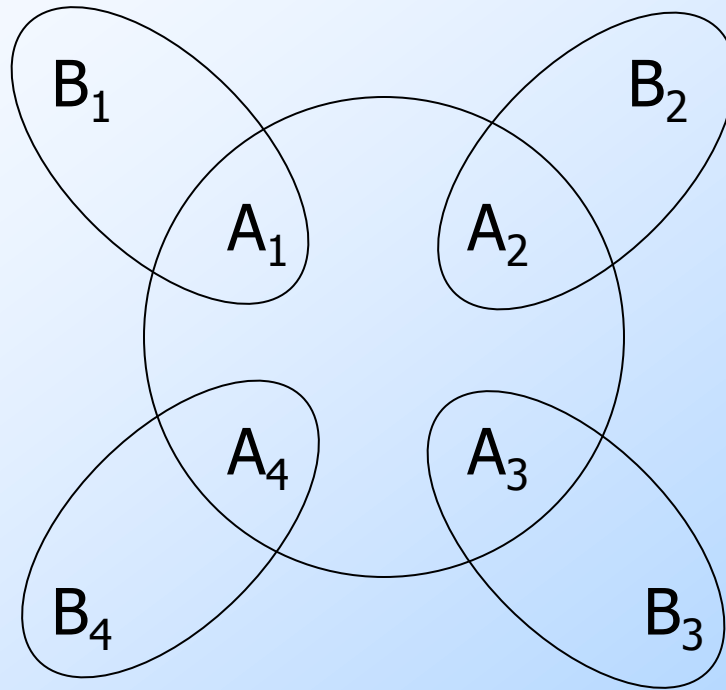
# Solving – (2)

- ◆ Solutions not in integers:
  - ◆ Drop an attribute with a share  $< 1$  from the map-key and re-solve.
  - ◆ Round other nonintegers, and treat  $k$  as a suggestion, since the product of the integers may not be  $k$ .

# Special Case: Star Joins

- ◆ A *star join* combines a large *fact table*  $F(A_1, A_2, \dots, A_n)$  with tiny *dimension tables*  $D_1(A_1, B_1), D_2(A_2, B_2), \dots, D_n(A_n, B_n)$ .
  - ◆ There may be other attributes not shown, each belonging to only one relation.
- ◆ **Example:** Facts = sales; dimensions tell about buyer, product, etc.

# Star-Join Pattern



# Star Joins – (2)

- ◆ Map-key = the  $A$ 's.
  - ◆  $B$ 's are dominated.
- ◆ **Solution:**  $d_i a_i = \lambda k$  for all  $i$ .
  - ◆ That is, the shares are inversely proportional to the dimension-table sizes.

# Cool Application of Star Join Result

- ◆ Fact/dimension tables are often used for analytics.
  - ◆ **Example**: fact table is all sales records; dimension tables give info about customers, products, suppliers, etc.
- ◆ **Aster Data approach**: partition fact table among nodes permanently; replicate needed pieces of dimension tables.

# Star-Join Application – (2)

- ◆ Our solution lets you partition the fact table to  $k$  nodes.
- ◆ Replication of tuples in the dimension tables is minimized.

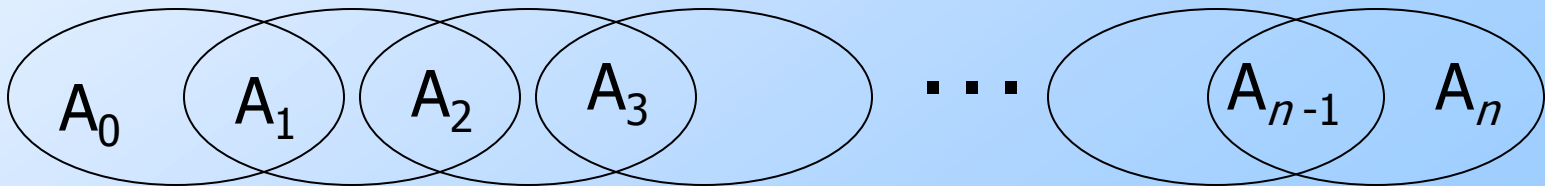
# Chain Joins

◆ A chain join has the form

$R(A_0, A_1) \text{ JOIN } R(A_1, A_2) \text{ JOIN } \dots \text{ JOIN } R(A_{n-1}, A_n)$

◆ Other attributes may appear, but only in one relation.

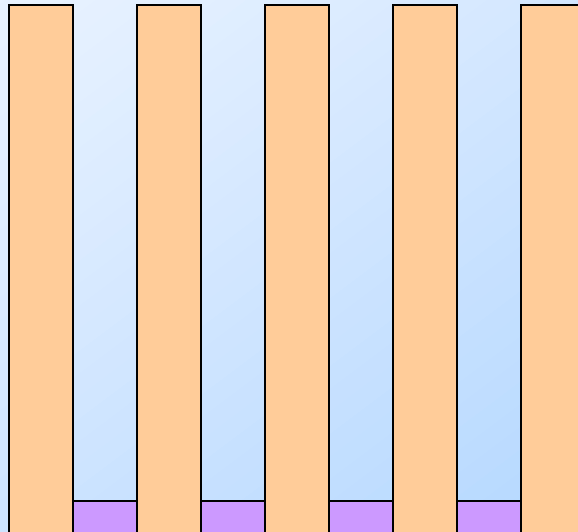
◆  $A_0$  and  $A_n$  are dominated; other attributes are in the map-key.



# Special Case: All Relations Have the Same Size

- ◆ Illustrates strange behavior.
  - ◆ Even and odd  $n$  have very different distributions of the share variables.
- ◆ **Even**  $n$  :  $a_2 = a_4 = \dots = a_{n-2} = 1$ ;  
 $a_1 = a_3 = \dots = a_{n-1} = k^{2/n}$

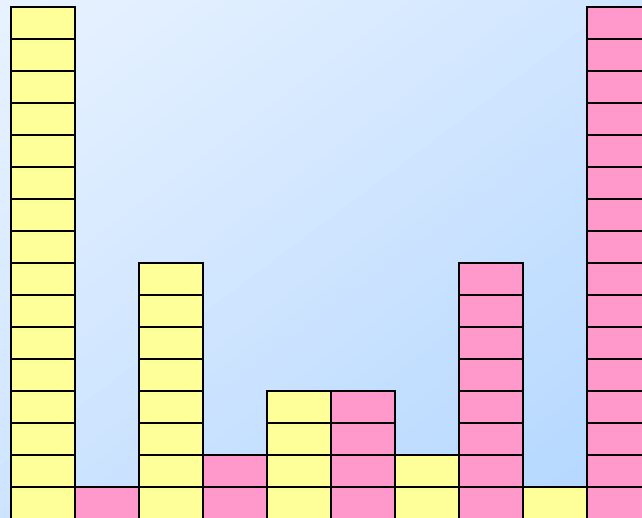
# Pattern for Even $n$



# Odd $n$ , Equal Relation Sizes

- ◆ Even  $a$ 's grow exponentially.
  - ◆ That is,  $a_4 = a_2^2$ ;  $a_6 = a_2^3$ ;  $a_8 = a_2^4, \dots$
- ◆ The odd  $a$ 's form the inverse sequence.
  - ◆ That is,  $a_1 = a_{n-1}$ ;  $a_3 = a_{n-3}$ ;  $a_5 = a_{n-5}; \dots$

# Pattern for Odd $n$



# Summary

1. Multiway joins can be computed by replicating tuples and distributing them to many compute nodes.
2. Minimizing communication requires us to solve a nonlinear optimization.
3. Method wins for star queries and queries on high-fanout graphs.
4. Exact solution for chain and star queries.