Algorithmic Aspects of Parallel Query Processing

Paris Koutris – U. of Wisconsin

Semih Salihoglu – U. of Waterloo

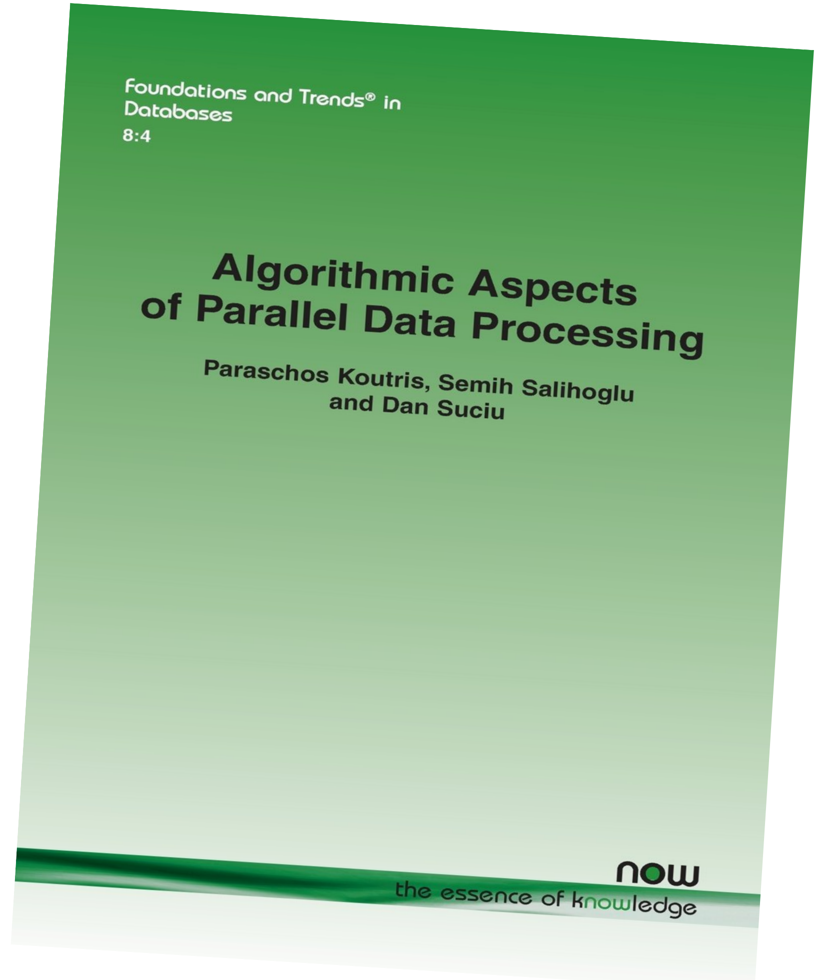
Dan Suciu – U. of Washington

Motivation

- Most modern data analytics tools process data on a cluster:
  - Spark, Dremel, Redshift, Myria, Hive, Impala, Scope, Flink, etc
- **Reason**: use sufficiently many nodes to avoid having to spill intermediate results to disk
- **Consequence**: data is processed on clusters with x10-x1000 nodes

This tutorial: basic data processing algorithms on a large cluster

Tutorial based on this survey



https://tinyurl.com/y99w99b4

Free until June 18
(create account)

Outline

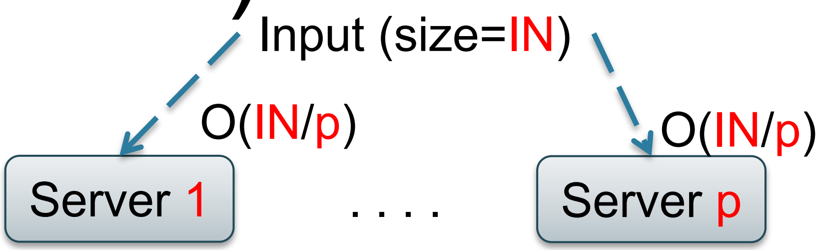
- Models of parallel computation (Dan)
- Two-way joins (Paris)
- Multi-way joins (Paris+Semih)
- Sorting & Matrix multiplication (Paris+Semih)
- Conclusion (Dan)

Models

- Abstract model to analyze algorithms

- **M**assively **P**arallel **C**ommunication (MPC) (simplified BSP model [VALIANT '90])
  - Cluster of nodes (=servers, =processors)
  - Computation = several rounds
  - Each round = processing + communication
- Shared-nothing architecture

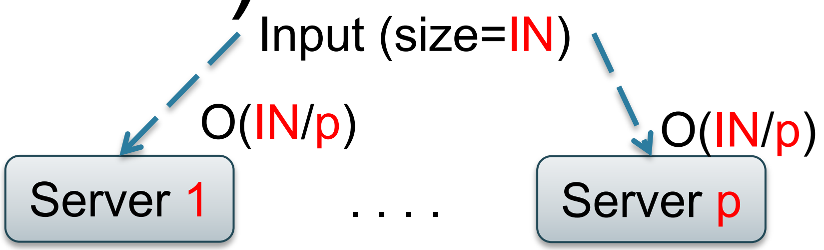
Massively Parallel Communication Model (MPC)



Input data = size IN

Number of servers = p

Massively Parallel Communication Model (MPC)



Input data = size IN

Number of servers = p

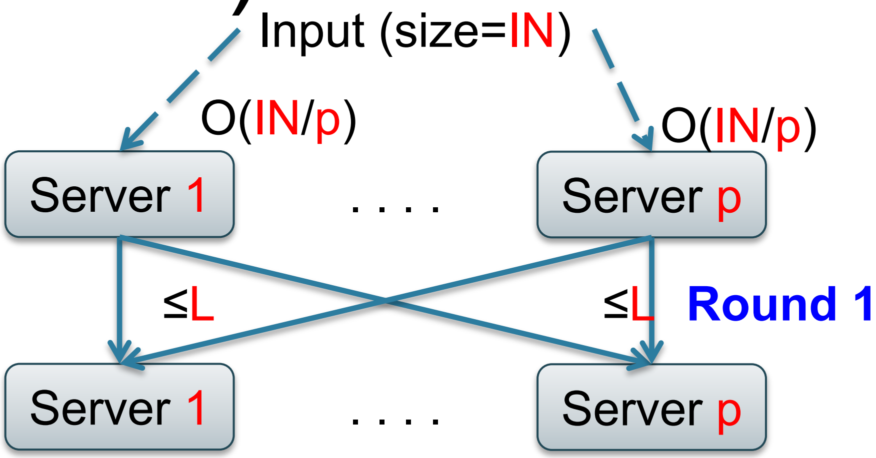
One round = Compute & communicate

Massively Parallel Communication Model (MPC)

Input data = size IN

Number of servers = p

One round = Compute & communicate

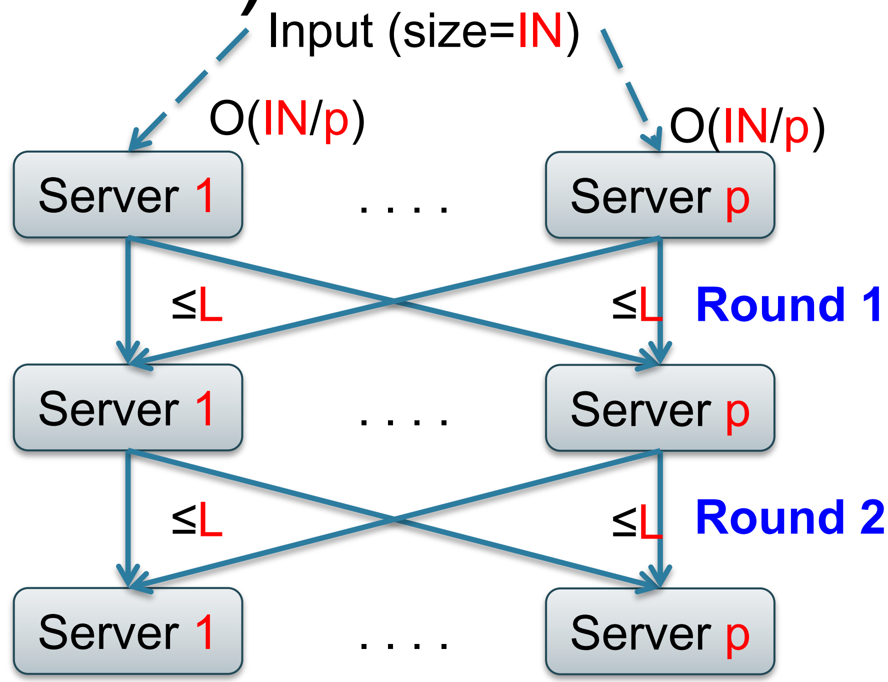


Massively Parallel Communication Model (MPC)

Input data = size IN

Number of servers = p

One round = Compute & communicate

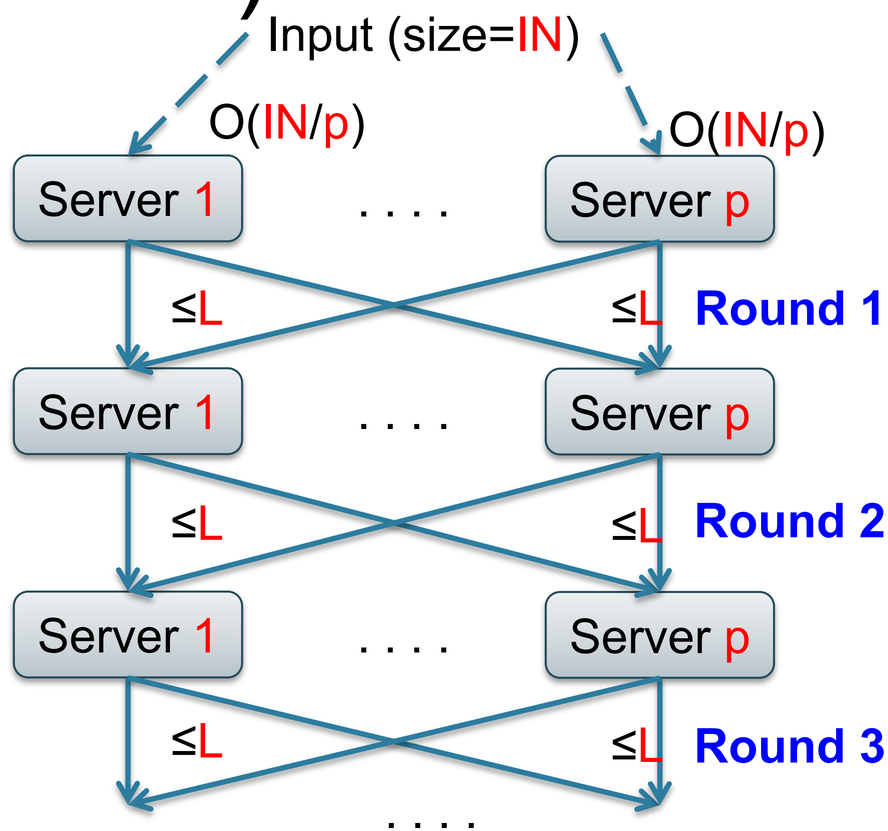


Massively Parallel Communication Model (MPC)

Input data = size IN

Number of servers = p

One round = Compute & communicate



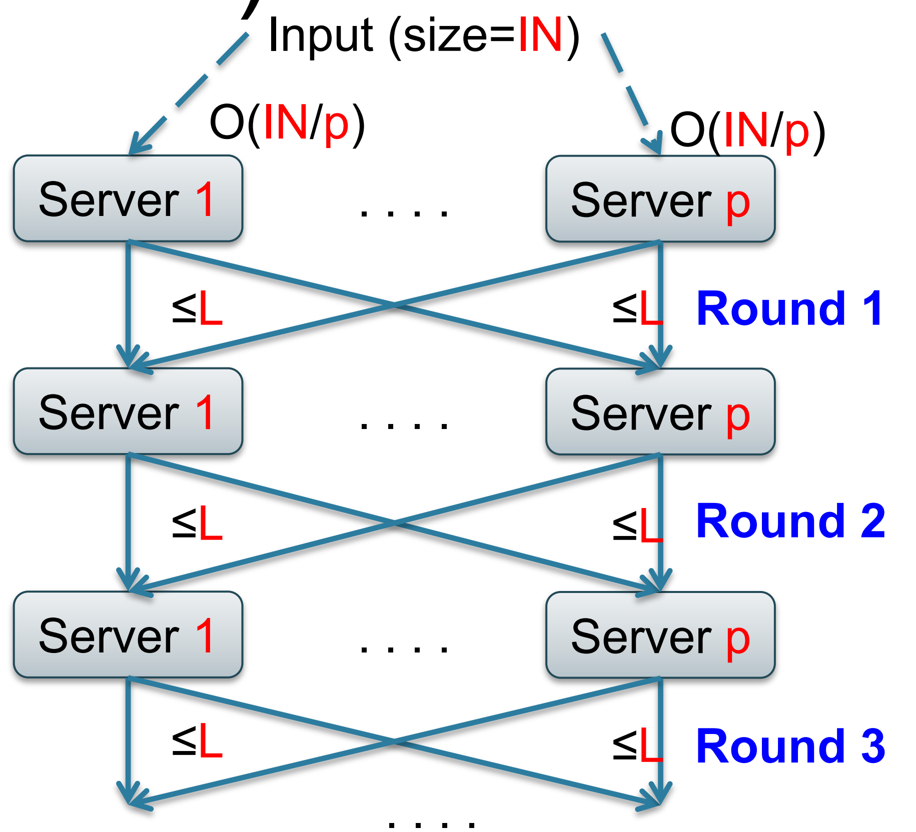
Massively Parallel Communication Model (MPC)

Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds



Massively Parallel Communication Model (MPC)

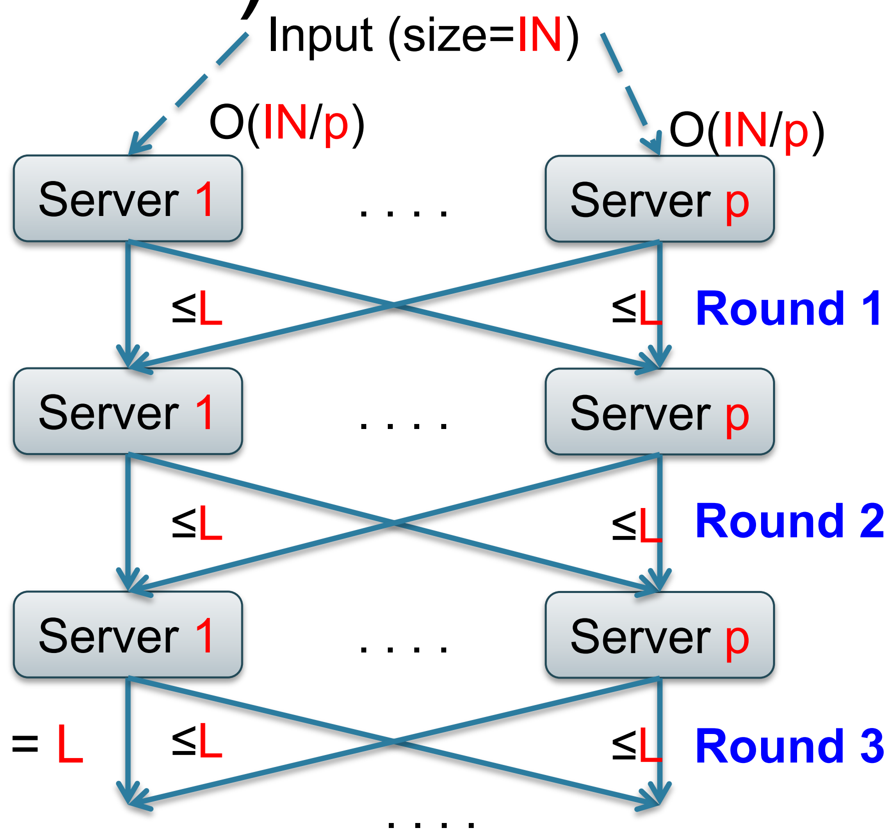
Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L



Massively Parallel Communication Model (MPC)

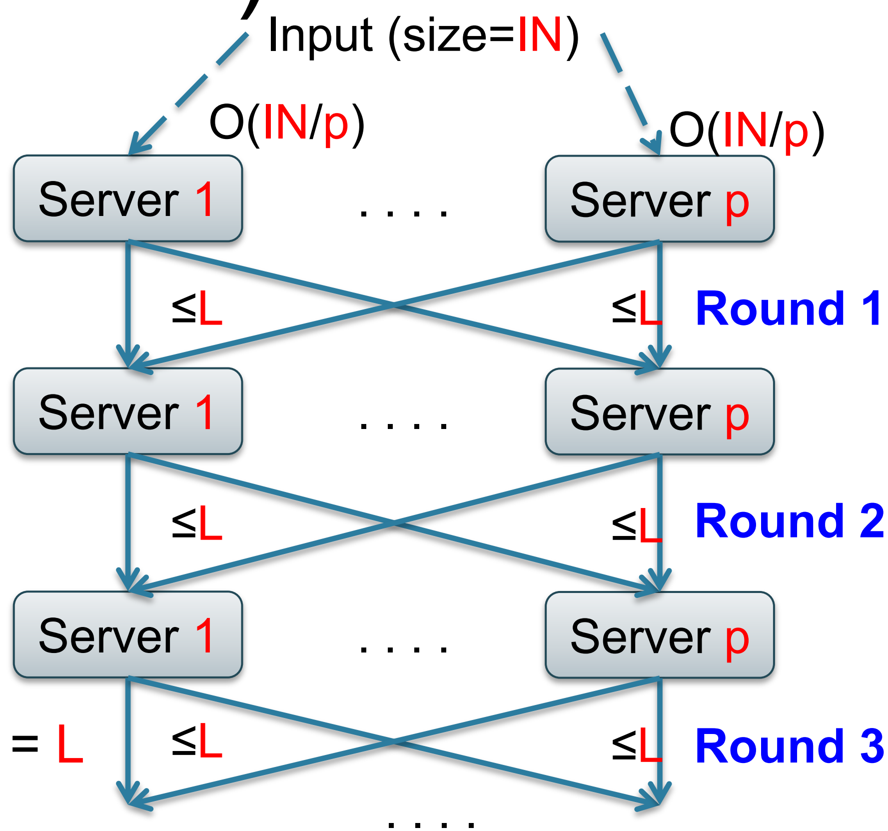
Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L



Cost:

Load L

Rounds r

Massively Parallel Communication Model (MPC)

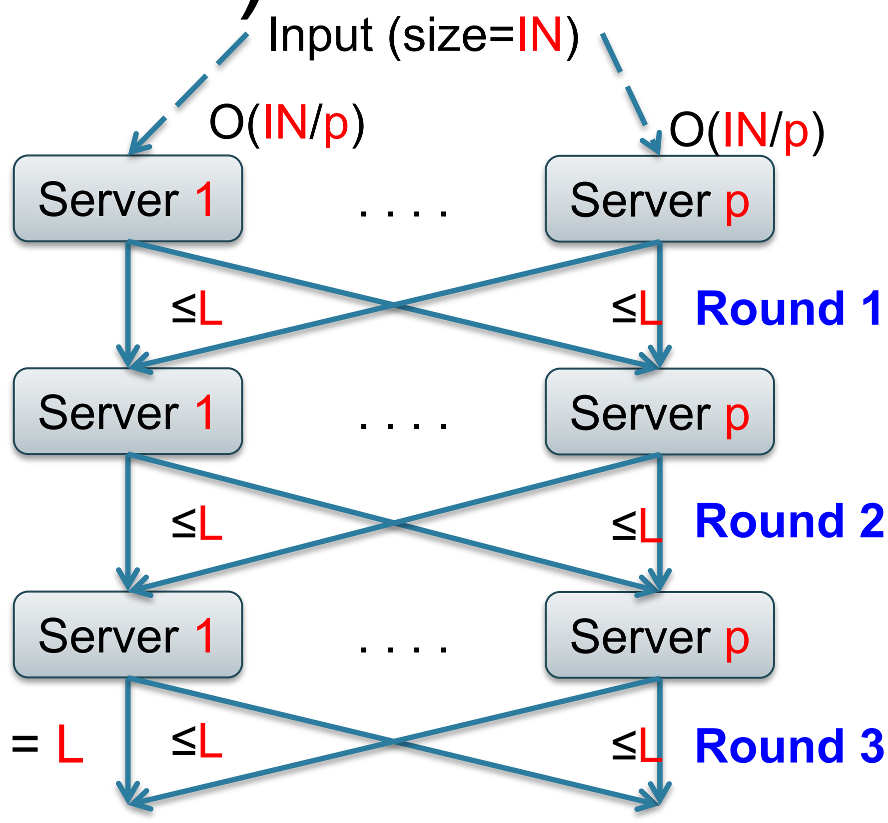
Input data = size IN

Number of servers = p

One round = Compute & communicate

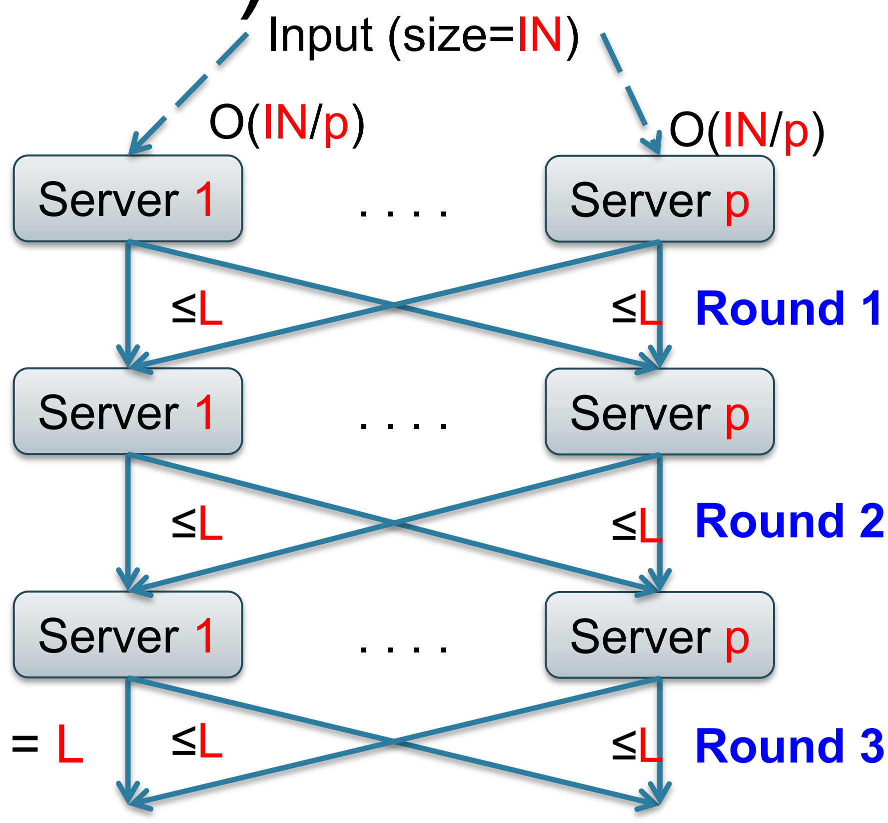
Algorithm = Several rounds

Max communication load / round / server = L



Cost:				
Load L				
Rounds r				

Massively Parallel Communication Model (MPC)



Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L

Cost:			Naïve 1	
Load L			L = IN	
Rounds r			1	

Massively Parallel Communication Model (MPC)

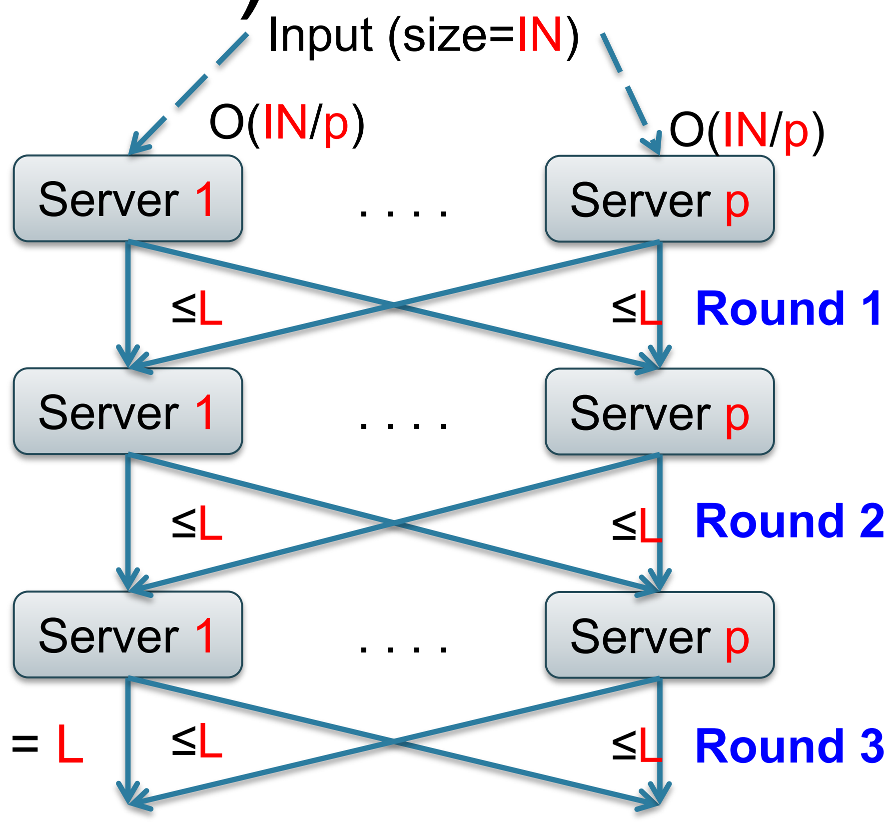
Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L



Cost:			Naïve 1	Naïve 2
Load L			L = IN	L = IN/p
Rounds r			1	p

Massively Parallel Communication Model (MPC)

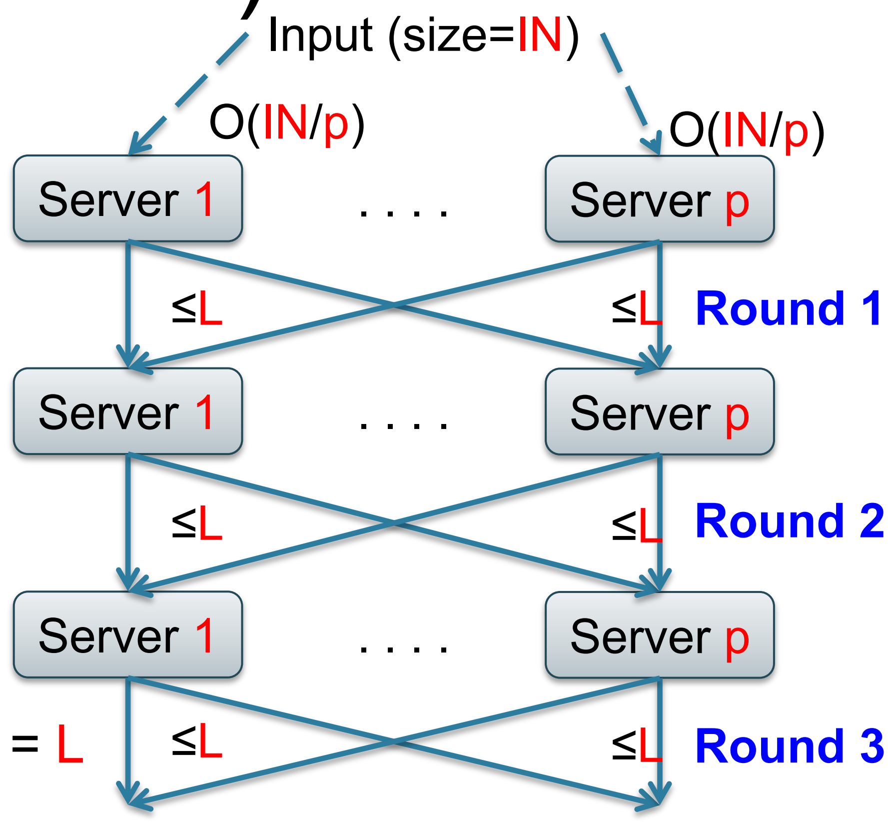
Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L



Cost:	Ideal		Naïve 1	Naïve 2
Load L	L = IN/p		L = IN	L = IN/p
Rounds r	1		1	p

Massively Parallel Communication Model (MPC)

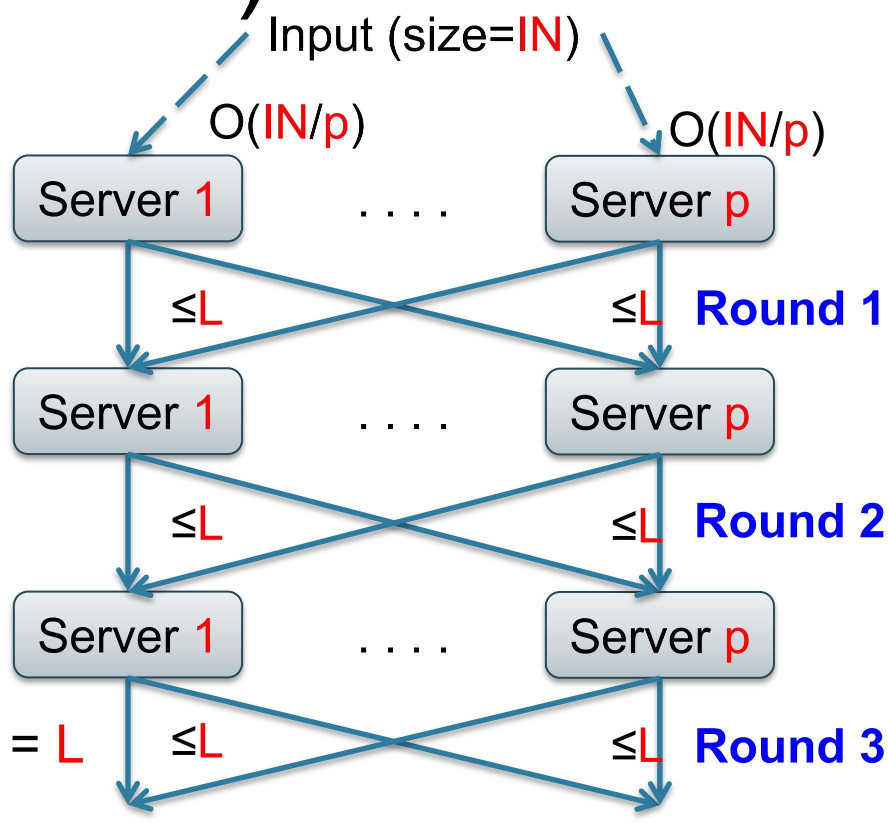
Input data = size IN

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L



Cost:	Ideal	Practical $\varepsilon \in (0,1)$	Naïve 1	Naïve 2
Load L	L = IN/p	$L = IN/p^{1-\varepsilon}$	L = IN	L = IN/p
Rounds r	1	O(1)	1	p

Discussion: Traditional Models

- Circuits ≈ *oblivious* MPC
  - Circuit-size = $p \times r$, Depth = r, Fan-in = L
- PRAM: shared-memory, p processors
  - Brent's theorem: $T_p = O(\text{Circuit-size}/p + \text{Depth})$
- BSP [VALIANT '90]: shared-nothing
  - detailed communication cost
  - MPC removes those details

Summary of the Model

- MPC
  - shared nothing
  - All-to-all communication

- Two cost parameters:
  - L (load) = max communication at each server
  - r (number of rounds)

Outline

- Models of parallel computation (Dan)
- Two-way joins (Paris)
- Multi-way joins (Paris+Semih)
- Sorting & Matrix multiplication (Paris+Semih)
- Conclusion (Dan)

2-way Joins

Join(x,y,z) = R(x,y) ⋈ S(y,z)

SELECT *
FROM R , S
WHERE R.y = S.y ;

We will see 2 types of techniques for a parallel join:

- hash-based join
- sort-based join

Parallel Hash Join

Join(x,y,z) = R(x,y) ⋈ S(y,z)

- IN = |R| + |S|
- p servers

R

x	y
a	b
a	c
b	c

S

y	z
b	d
b	e
c	e

Choose a *hash function* h that maps values from the domain to one of the p servers

Round #1 communication: each server

- sends record R(x,y) to server h(y)
- sends record S(y,z) to server h(y)

Round #1 computation: each server

- computes the join R(x,y) ⋈ S(y,z) of the local instances

A Simple Analysis w/o Skew

How far away is the maximum load L from the expected load IN/p ?

Suppose that every value of y appears at most once in the database (**no skew**). Then:

$$Pr[L \geq (1+\delta)\frac{\text{IN}}{p}] \leq pe^{-\frac{\delta^2 \text{IN}}{3p}}$$

In other words, for large enough input, with high probability we have load:

$$L = O(IN/p)$$

A Simple Analysis with Skew

Suppose now that every value of y appears exactly d times in the database. Then:

$$Pr[L \geq (1+\delta)\frac{\text{IN}}{p}] \leq pe^{-\frac{\delta^2 \text{IN}}{3pd}}$$

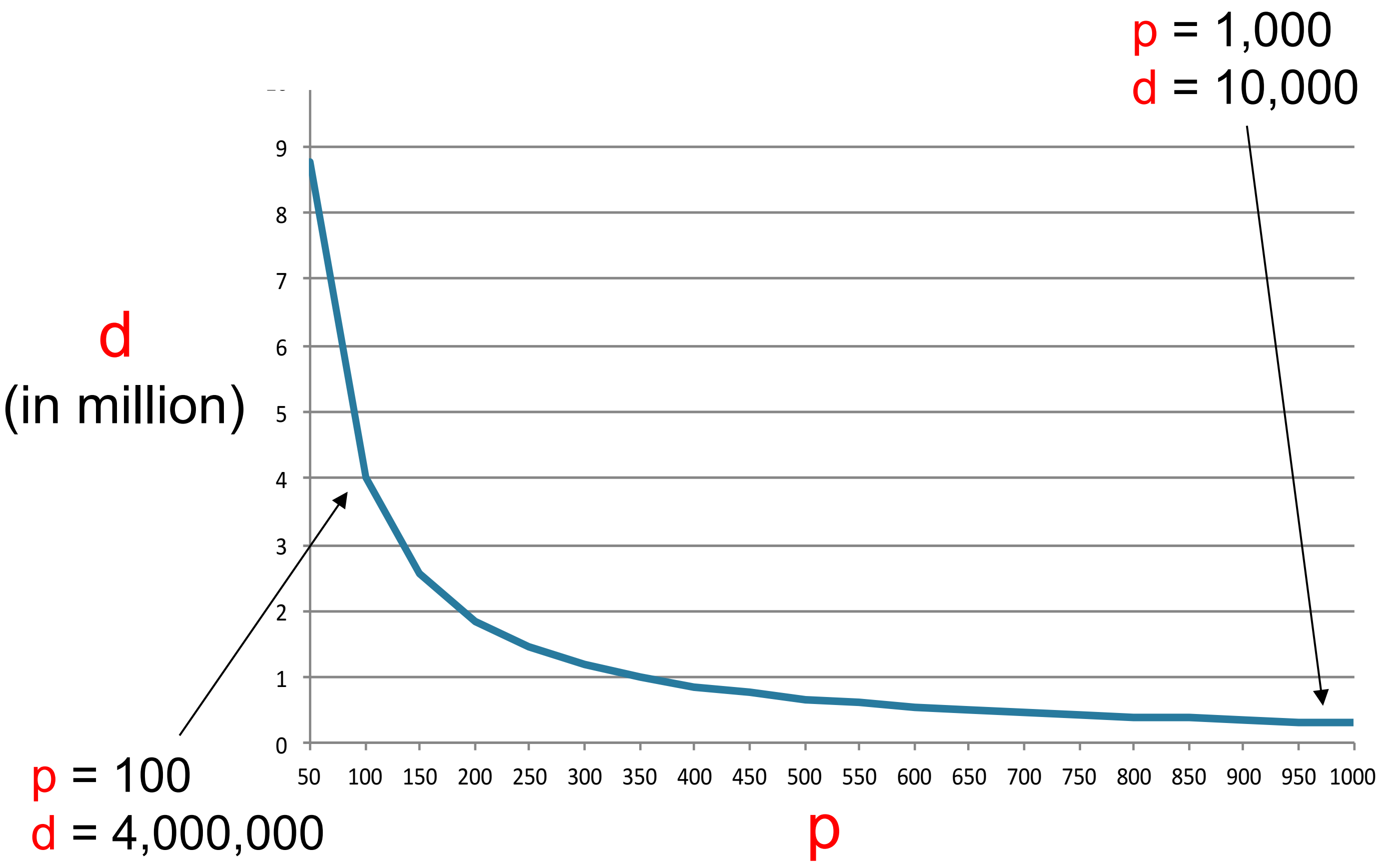
The exponent has now an additional factor d

$d << IN/p$ | $d = \Theta(IN/p)$ | $d >> IN/p$

$L = O(IN/p)$ | $L = O(IN \log(p) / p)$ | *as large as* $L = IN$

The Effect of Skew

- IN = 100 billion tuples
- at most 30% over the expected load IN/p with probability 95%



As the number p of servers grows, it is more likely that we observe the effects of skew!

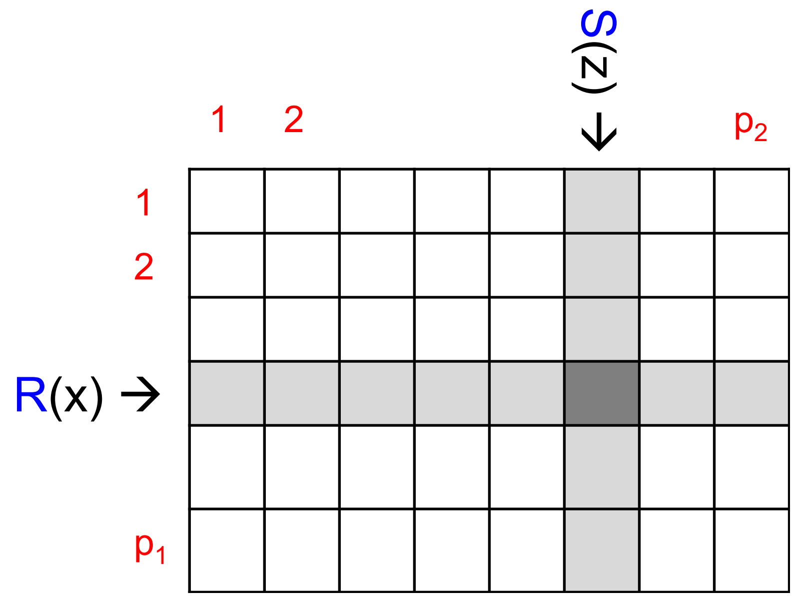
Skew in the Extreme

- In the extreme, all tuples in R and all tuples in S have the same value for attribute y
- Parallel hash-join will incur a load $L = IN$ in this case
- We can do better by observing that the join then degenerates to a Cartesian product:

$$\text{Product}(x,z) = R(x) \bowtie S(z)$$

SELECT *
FROM R , S ;

Cartesian Product



Product(x,z) = R(x) ⋈ S(z)

- Choose **shares** p_1, p_2 s.t. $p = p_1 \times p_2$
- Arrange servers in a $p_1 \times p_2$ rectangle

Round #1 communication: each server
- sends a tuple from R to a random row
- sends a tuple from S to a random column

Round #1 computation: each server
- computes the Cartesian product locally

$$L = 2\sqrt{\frac{|R||S|}{p}}$$

Optimal choice for p_1, p_2: $|R| / p_1 = |S| / p_2$

- OUT = |R| * |S|
- The above 1-round algorithm is optimal for the load
- When |R| << |S|, the algorithm *broadcasts* R and partitions only S

Parallel Join for Arbitrary Skew

- For inputs with arbitrary skew, we need to combine the 2 techniques: parallel hash join + Cartesian product
- Key concept: **heavy hitter**

Heavy hitter: any value of the join attribute y that occurs at least IN/p times in R or S

A value that is not heavy hitter is called **light hitter**

Parallel Join for Arbitrary Skew

Algorithm

1. Run the parallel hash join for the light hitter values

   load $L = O(IN/p)$

2. For each heavy hitter $\mathbf{b_i}$ compute the Cartesian product of the *subquery* $R(x, b_i) \bowtie S(b_i, z)$ using p_i exclusive servers

By choosing the p_i appropriately such that their sum is p, we can get:

$$L = O\left(\sqrt{\frac{\text{OUT}}{p}} + \frac{\text{IN}}{p}\right)$$

Parallel Sort Join

Algorithm [HU ET AL. '17]

1. Union the two relations R,S
2. Parallel sort the result using the value of the join attribute as sort key
3. We distinguish two cases for a value of y:
   - if all tuples with value **b** are in the same server, the join can be computed locally
   - for values that cross multiple servers, we apply the Cartesian product algorithm

$$L = O\left(\sqrt{\frac{\text{OUT}}{p}} + \frac{\text{IN}}{p}\right)$$

2-way Joins in Practice

- Parallel Hash Join [SparkSQL, Hive, Myria, Impala, …]
  - most commonly used join algorithm
- Broadcast Join [Hive, Impala, SparkSQL]
  - if one relation is much smaller than the other relation, broadcast it to every server
- Parallel Sort Join [SparkSQL]

Note: the choice of the local join algorithm is independent of the parallel algorithm!

Outline

- Models of parallel computation (Dan)
- Two-way joins (Paris)
- Multi-way joins (Paris+Semih)
- Sorting & Matrix multiplication (Paris+Semih)
- Conclusion (Dan)

From 2-way to Multiway Joins

The *triangle query*

- $\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$
- $|R| = |S| = |T| = N$ tuples
- $IN = 3\ N$

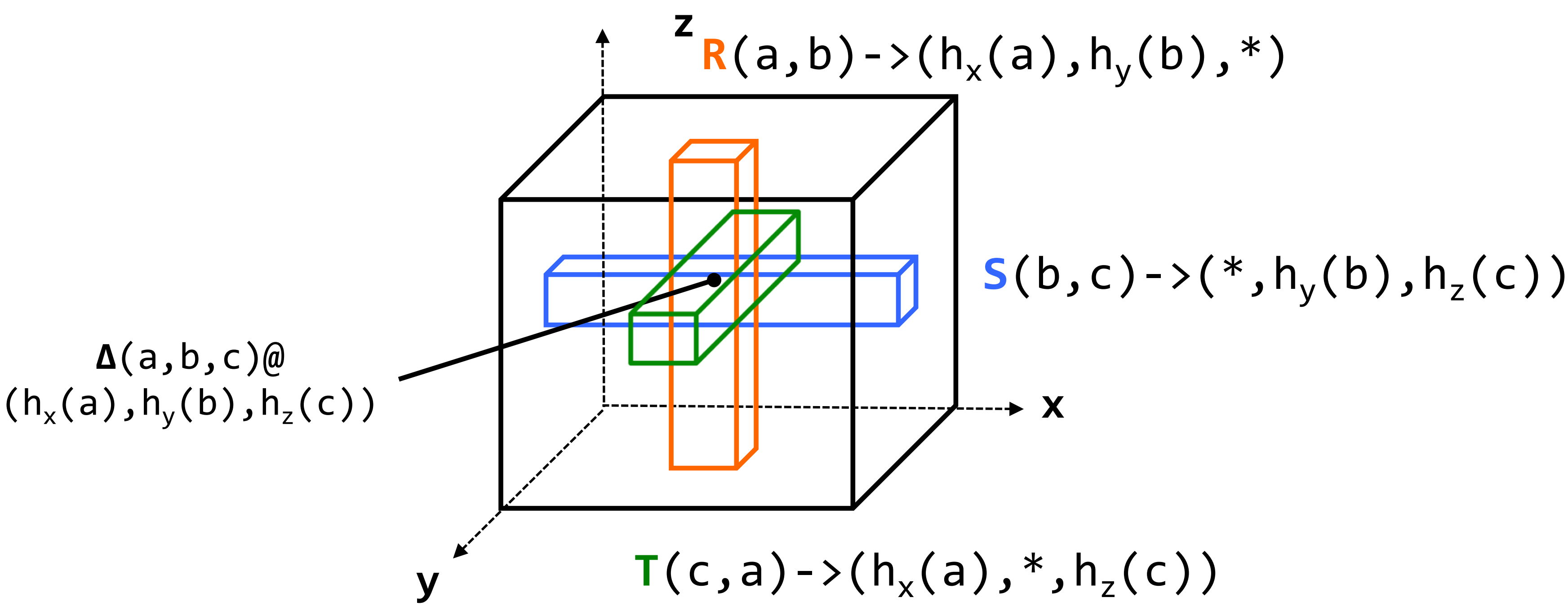
The triangle query can be computed in one round!

- Algorithm introduced by [AFRATI AND ULLMAN '10]
  - Named later **Shares Algorithm**
  - For MapReduce
- Analyzed/optimized [BEAME ET AL. '13,'14]
  - **HyperCube Algorithm**
  - For the MPC model

Triangles in One Round

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

- Place servers in a $p^{1/3} \times p^{1/3} \times p^{1/3}$ cube
- Each server is identified by a coordinate (i, j, k)
- Choose 3 random, independent hash functions h_x, h_y, h_z



Analysis for Triangles

- $\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$
- $|R| = |S| = |T| = N$ tuples

> **Theorem** The HyperCube algorithm computes triangles with load $L = O(N/p^{2/3})$ w.h.p. on any input database **without skew**

Can we compute triangles with $L = N/p$?

- **No!** [BEAME ET AL. '13]
- In fact, any 1-round algorithm has load $L = \Omega(N/p^{2/3})$, even on inputs without skew

Multiway Joins

$\mathbf{Q}(x_1, x_2, \ldots, x_k) = S_1(\mathbf{x_1}) \bowtie S_2(\mathbf{x_2}) \bowtie \ldots \bowtie S_l(\mathbf{x_l})$

shares

<u>HyperCube Algorithm</u>

- organize the p servers in a $p_1 \times p_2 \times \ldots \times p_k$ hypercube
- choose k independent hash functions

Round #1 communication: send $S_j(x_{j1}, x_{j2}, \ldots)$ to ***all*** servers whose coordinates agree with $h_{j1}(x_{j1}), h_{j2}(x_{j2}), \ldots$

Round #1 computation: compute **Q** locally on every server

How do we choose the *shares* so that we minimize L?

Choosing the Shares

$\mathbf{Q}(x_1, x_2, \ldots, x_k) = S_1(\mathbf{x_1}) \bowtie S_2(\mathbf{x_2}) \bowtie \ldots \bowtie S_l(\mathbf{x_l})$

- the shares must satisfy $\prod_i p_i \leq p$
- #tuples a server receives from S_j = $\dfrac{|S_j|}{\prod_{i:x_i \in S_j} p_i}$
- To optimize load, we *minimize*

$$\max_j \frac{|S_j|}{\prod_{i:x_i \in S_j} p_i}$$

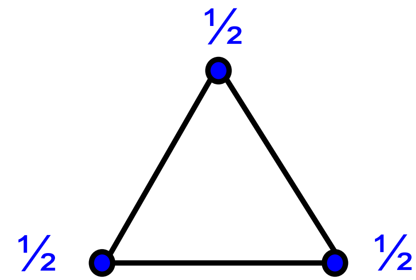
Refresher: Covers & Packings

$\mathbf{Q}(x_1, x_2, \ldots, x_k) = S_1(\mathbf{x_1}) \bowtie S_2(\mathbf{x_2}) \bowtie \ldots \bowtie S_l(\mathbf{x_l})$

Fractional vertex cover

weights $v_1, v_2, \ldots\ v_k \geq 0$ s.t.

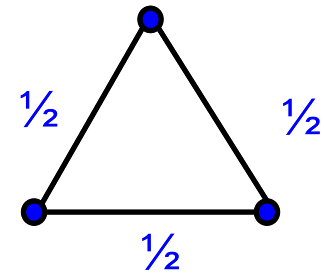
for all S_j: $\sum_{i: x_i \in S_j} v_i \geq 1$



Fractional edge packing

weights $u_1, u_2, \ldots\ u_l \geq 0$ s.t.

for all x_i: $\sum_{j: x_i \in S_j} u_j \leq 1$



$$min_{\boldsymbol{v}} \Sigma_i v_i = max_{\boldsymbol{u}} \Sigma_j u_j = \boldsymbol{\tau}^*$$

Optimal Load

$\mathbf{Q}(x_1, x_2, \ldots, x_k) = S_1(\mathbf{x_1}) \bowtie S_2(\mathbf{x_2}) \bowtie \ldots \bowtie S_l(\mathbf{x_l})$

[Beame'14]

Theorem The HyperCube algorithm computes a join query *in one round* with load

$$L = \max_{\text{edge packing } \mathbf{u}} \left(\frac{\prod_{j=1}^{t} |S_j|^{u_j}}{p} \right)^{1/\sum_j u_j}$$

The load achieved is **optimal**

When $|S_1| = |S_2| = \ldots = N$ then $L = N / p^{1/\tau^*}$

* The load analysis works only for data **without skew**

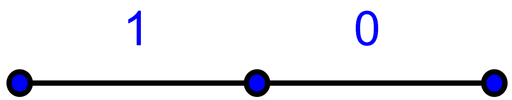
Example - Equal Size

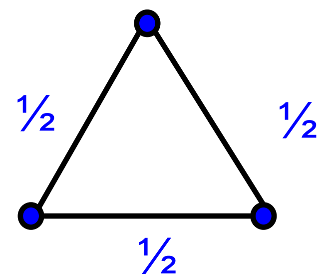
When all relations have equal size (N),
then the optimal load formula becomes $L = N / p^{1/\tau^*}$

R(x,y) ⋈ S(y,z)

R(x,y) ⋈ S(y,z) ⋈ T(z,x)

τ* = 1





τ* = 3/2

L = N / p

$L = N / p^{3/2}$

Example - Unequal Size

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

Edge packing u_R, u_S, u_T	Maximum load L	Max when	HyperCube Shares
1/2, 1/2, 1/2	$(\lvert R\rvert\,\lvert S\rvert\,\lvert T\rvert)^{1/3} / p^{2/3}$		
1, 0, 0	$\lvert R\rvert / p$		
0, 1, 0	$\lvert S\rvert / p$		
0, 0, 1	$\lvert T\rvert / p$		
0, 0, 0	0		

L = max of these values

$$L = \max_{\mathbf{u}} \left(\frac{|R|^{u_R} |S|^{u_S} |T|^{u_T}}{p} \right)^{1/(u_R+u_S+u_T)}$$

Example - Unequal Size

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

Edge packing u_R, u_S, u_T	Maximum load L	Max when	HyperCube Shares
1/2, 1/2, 1/2	$(\lvert R\rvert\,\lvert S\rvert\,\lvert T\rvert)^{1/3} / p^{2/3}$	$\lvert R\rvert \approx \lvert S\rvert \approx \lvert T\rvert$	$p_x, p_y, p_z > 1$
1, 0, 0	$\lvert R\rvert / p$		
0, 1, 0	$\lvert S\rvert / p$		
0, 0, 1	$\lvert T\rvert / p$		
0, 0, 0	0		

L = max of these values

$$L = \max_{\mathbf{u}} \left(\frac{|R|^{u_R} |S|^{u_S} |T|^{u_T}}{p} \right)^{1/(u_R+u_S+u_T)}$$

Example - Unequal Size

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

Edge packing u_R, u_S, u_T	Maximum load L	Max when	HyperCube Shares
1/2, 1/2, 1/2	$(\lvert R\rvert \lvert S\rvert \lvert T\rvert)^{1/3} / p^{2/3}$	$\lvert R\rvert \approx \lvert S\rvert \approx \lvert T\rvert$	$p_x, p_y, p_z > 1$
1, 0, 0	$\lvert R\rvert / p$	$\frac{\lvert R\rvert}{p} \geq \sqrt{\frac{\lvert S\rvert\lvert T\rvert}{p}}$	$p_z = 1$
0, 1, 0	$\lvert S\rvert / p$		
0, 0, 1	$\lvert T\rvert / p$		
0, 0, 0	0		

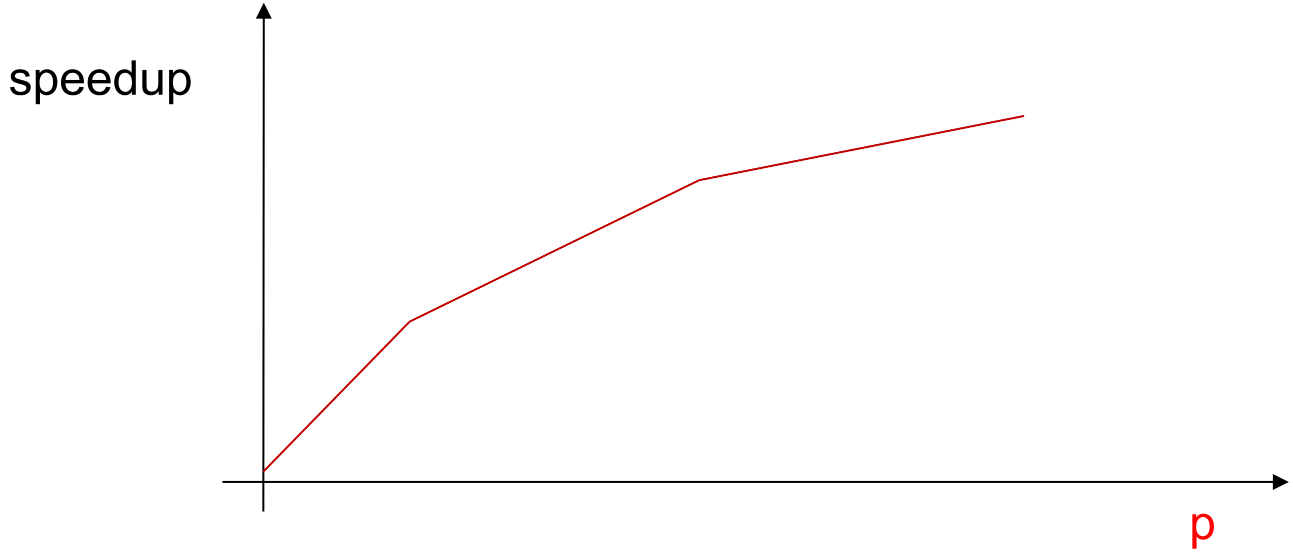
↑

L = max of these values

$$L = \max_{\mathbf{u}} \left(\frac{|R|^{u_R} |S|^{u_S} |T|^{u_T}}{p} \right)^{1/(u_R+u_S+u_T)}$$

HyperCube Speedup

- Given by $1/p^{\sum u_i}$, better than $1/p^{1/\tau^*}$
- As p increases, speedup degrades to $1/p^{1/\tau^*}$



Skew Matters

- Skewed data significantly degrades the performance in distributed query processing
- Skewed values must be treated specially!

- State of the art in large scale distributed systems: DIY ☹

The SkewHC algorithm

$\mathbf{Q}(x_1, x_2, \ldots, x_k) = S_1(\mathbf{x_1}) \bowtie S_2(\mathbf{x_2}) \bowtie \ldots \bowtie S_l(\mathbf{x_l})$

$|S_1| = |S_2| = \ldots = N$

Def. A value is a *heavy hitter* if it occurs > N/p times

Def. Fix $\boldsymbol{x} \subseteq \{x_1, \ldots, x_k\}$. The *residual query* $\mathbf{Q_x}$ is obtained from $\mathbf{Q}$ by removing the variables $\mathbf{x}$ and the empty atoms

Algorithm: in parallel, for every combination of heavy/light, compute the residual query for that combination

Theorem Let $\psi^*(\boldsymbol{Q}) = max_{\boldsymbol{x}} \tau^*(\boldsymbol{Q}_{\boldsymbol{x}})$

- load of the algorithm is $L = O(N/p^{1/\psi^*})$
- any algorithm needs load $L = \Omega(N/p^{1/\psi^*})$

Example

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

Heavy hitter = a value that occurs at least N/p times
Each attribute has at most p heavy hitters

x	y	z	Residual query	τ^*	L	$p_1 \times p_2 \times p_3$
light	light	light	$R(x,y) \bowtie S(y,z) \bowtie T(z,x)$	3/2	$N/p^{2/3}$	$p^{1/3} \times p^{1/3} \times p^{1/3}$
. . . .			. . .			

Example

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

Heavy hitter = a value that occurs at least N/p times
Each attribute has at most p heavy hitters

x	y	z	Residual query	τ^*	L	$p_1 \times p_2 \times p_3$
light	light	light	$R(x,y) \bowtie S(y,z) \bowtie T(z,x)$	3/2	$N/p^{2/3}$	$p^{1/3} \times p^{1/3} \times p^{1/3}$
light	light	heavy	$R(x,y) \bowtie S(y) \bowtie T(x)$	2	$N/p^{1/2}$	$p^{1/2} \times p^{1/2} \times 1$
. . . .			. . .			

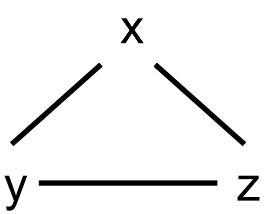
Example

$\Delta(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

Heavy hitter = a value that occurs at least N/p times
Each attribute has at most p heavy hitters

x	y	z	Residual query	τ^*	L	$p_1 \times p_2 \times p_3$
light	light	light	$R(x,y) \bowtie S(y,z) \bowtie T(z,x)$	3/2	$N/p^{2/3}$	$p^{1/3} \times p^{1/3} \times p^{1/3}$
light	light	heavy	$R(x,y) \bowtie S(y) \bowtie T(x)$	2	$N/p^{1/2}$	$p^{1/2} \times p^{1/2} \times 1$
light	heavy	heavy	$R(x) \bowtie T(x)$	1	N/p	$p \times 1 \times 1$
. . . .			. . .			

Summary

Query	No Skew 1 Round (τ^*)		Skew 1 Round (ψ^*)
	$\tau^* = 3/2$ $IN/p^{2/3}$	$\leq$	$\psi^* = 2$ $IN/p^{1/2}$
x — y — z	$\tau^* = 1$ IN/p	$\leq$	$\psi^* = 2$ $IN/p^{1/2}$
general	$IN/p^{1/\tau^*}$	$\leq$	$IN/p^{1/\psi^*}$

What about multiple rounds?

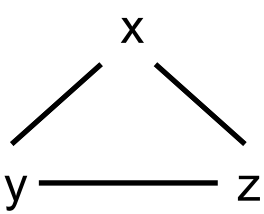
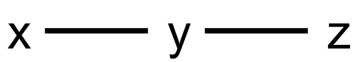
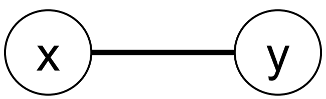
Multiple Rounds

- Queries are typically executed in multiple rounds

```
SELECT cKey, month, sum(price)
FROM Orders, Customers
GROUP BY cKey, month
```

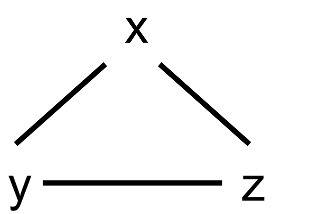
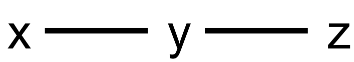
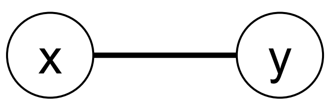
- High-level Question:
  - What is the computational power of rounds?
- Upshot: Few results & many open questions for CQs

1-round vs Multi-round

Query	No Skew 1 Round (τ^*)		Skew 1 Round (ψ^*)
	$\tau^* = 3/2$ $IN/p^{2/3}$	≤	$\psi^* = 2$ $IN/p^{1/2}$
	$\tau^* = 1$ IN/p	≤	$\psi^* = 2$ $IN/p^{1/2}$
 R(x),S(x, y),T(y)	$\tau^* = 2$ $IN/p^{1/2}$	≤	$\psi^* = 2$ $IN/p^{1/2}$

No Skew, Multi Rounds => Easy

1-round vs Multi-round

Query	No Skew Multi-Round		No Skew 1 Round (τ^*)		Skew 1 Round (ψ^*)
	IN/p	≤	$\rho^* =$ $\tau^* = 3/2$ IN/$p^{2/3}$	≤	$\psi^* = 2$ IN/$p^{1/2}$
	IN/p	≤	$\tau^* = 1$ IN/p	≤	$\rho^* =$ $\psi^* = 2$ IN/$p^{1/2}$
 R(x),S(x, y),T(y)	$\rho^* = 1$ IN/p	≤	$\tau^* = 2$ IN/$p^{1/2}$	≤	$\psi^* = 2$ IN/$p^{1/2}$

No Skew, Multi Rounds => Easy

Skew, Multi Round

AGM bound ρ^*

$1 \leq \rho^*, \ \tau^* \ \leq \psi^*$

Tight for only some queries

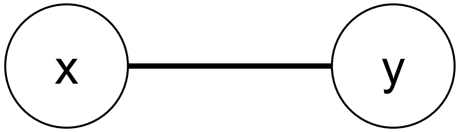
General queries open

Background: AGM Bound

Given Q: $R_1 \bowtie \ldots \bowtie R_n$, what's the max size of |OUT|?

Assume $|R_i|$ are equal. Let $\vec{e} = (e_1 \ldots e_n)$ be a fractional edge cover:

Then: $|OUT| \leq IN^{|\vec{e}|}$



R(x) S(x, y) T(y)

1 0 1

$\rho^* = 1$

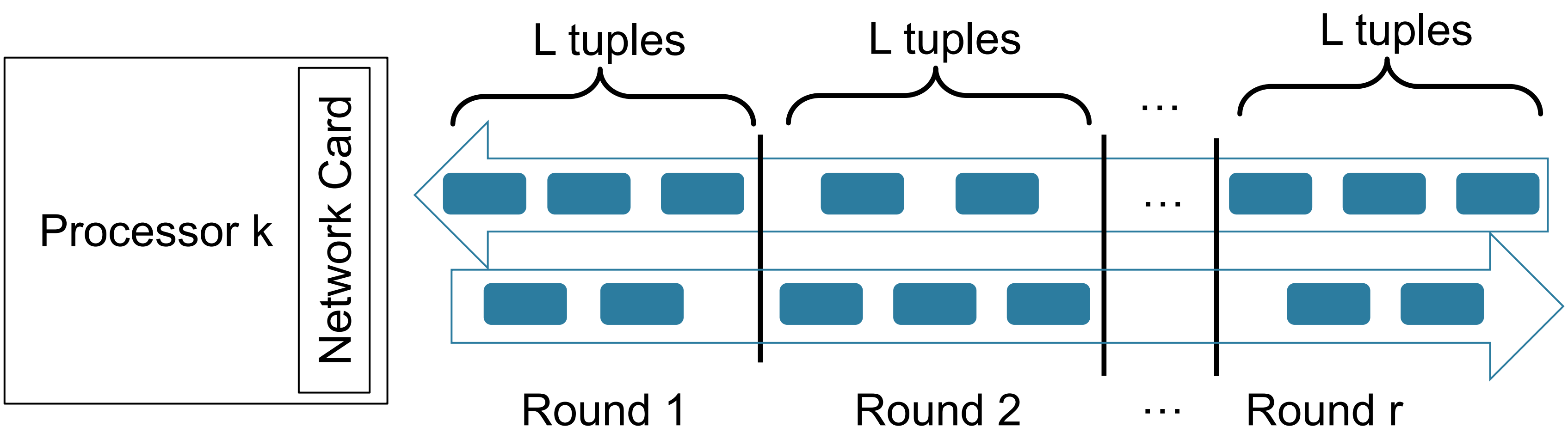
$|OUT| \leq IN^2$

ρ^* : weight of minimum fractional edge cover

$|OUT| \leq IN^{\rho^*}$

Multi-round Communication LB

Simple counting argument!



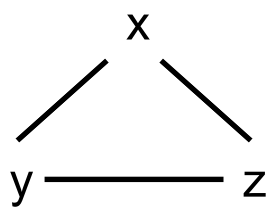
with rL tuples, we can output $(rL)^{\rho^*}$ tuples:

So: $p(rL)^{\rho^*} \geq IN^{\rho^*} \Rightarrow L \geq O(IN/(r\, p^{1/\rho^*}))$

If $r = O(1)$, then $L \geq O(IN/p^{1/\rho^*})$

Extreme No Skew Multi-Round: Easy Case

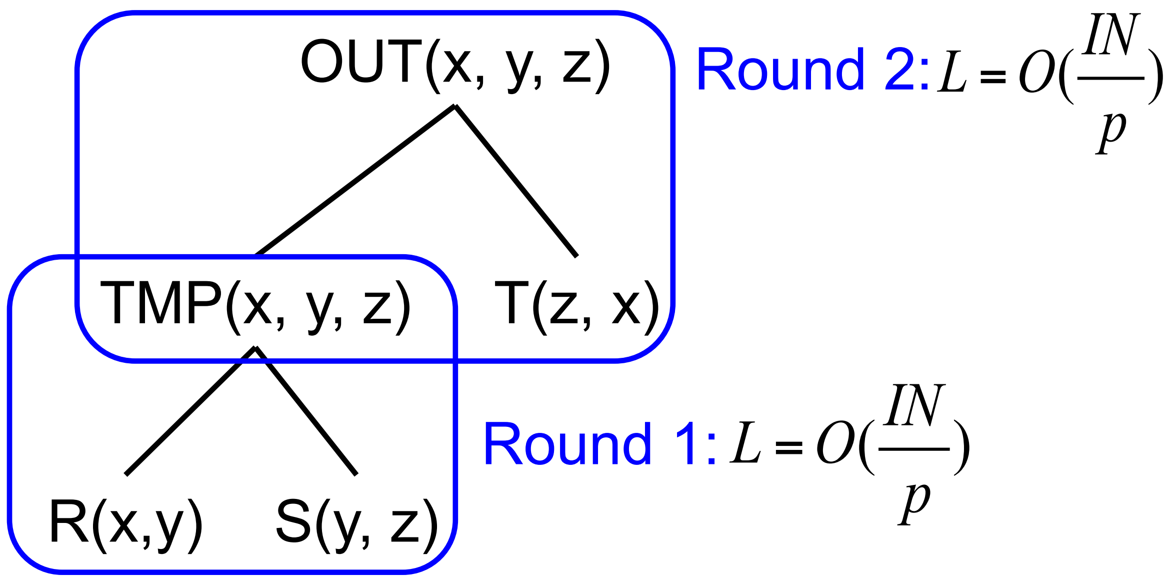
- Iterative binary joins at each round
- Ex:



R(x, y)	
1	2
2	4
…	…
n	9

S(y, z)	
1	n
2	7
…	…
n	3

T(z, x)	
1	5
2	6
…	…
n	1

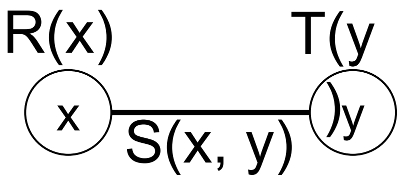


- Any Q, with r = n-1 and optimal L

Intermediate relation sizes do not grow with binary joins in the case of extreme no-skew

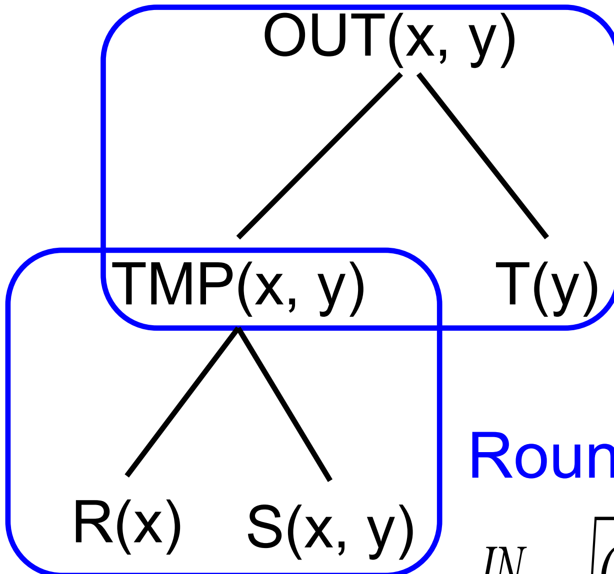
Skew Multi-Round: Hard Case (1)

- Iterative binary joins *if Q decomposes into semijoins*
- Ex 1:



$\rho^* = 1$

$\psi^* = 2$



Round 2: $L = O(\frac{IN}{p})$

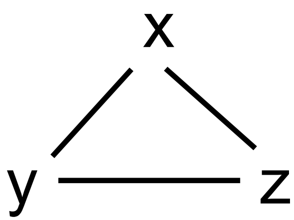
Round 1:

$$L = O(\frac{IN}{p} + \sqrt{\frac{OUT}{p}}) = O(\frac{IN}{p} + \sqrt{\frac{IN}{p}}) = O(\frac{IN}{p})$$

Semijoins remove potential outputs each round without growing intermediate relations

Skew Multi-Round: Hard Case (2)

- Ex 2:

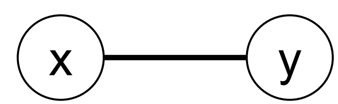


$\rho^* = 3/2$

- Decompose into "semijoin residual queries"

Heavy-Light + Semijoins Alg

$\deg(h) \overset{?}{<} IN/p^{1/3}$

Light values	$O(p^{1/3})$ many Heavy values (e.g. z = 3)
R^L(x, y) **S^L(y, z)** **T^L(z, x)**	**R(x, y)** **S(y, 3)** **T(3, x)**
	q(z=3): 
1 round HC on p machines	2 round semijoin on $p^{2/3}$ machines
L=O($IN/p^{2/3}$)	L=O($IN/p^{2/3}$)

r: 2, L=O($IN/p^{2/3}$) => worst-case optimal

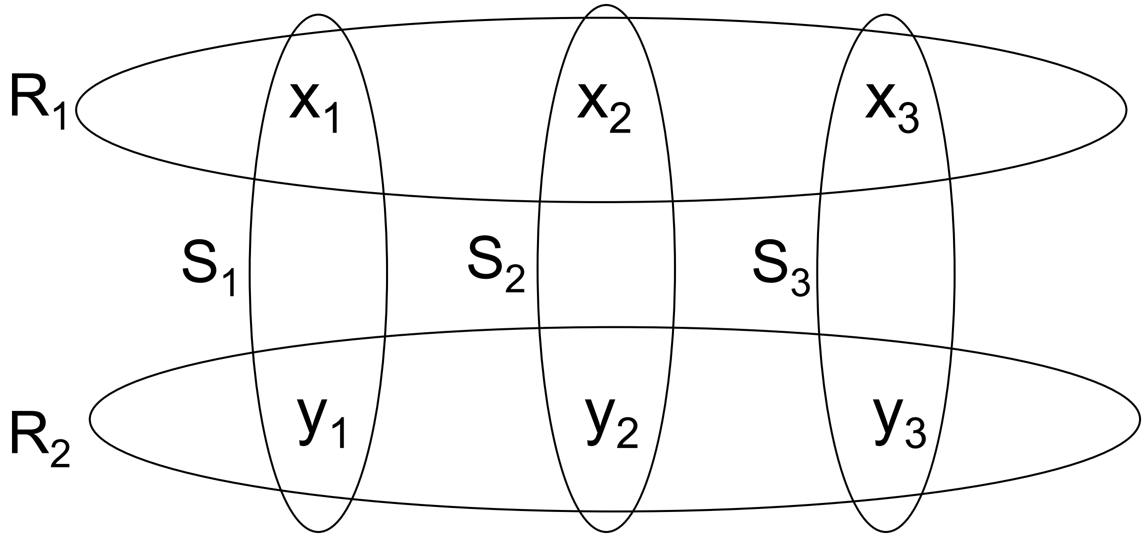
Limitation of HL+Semijoins

Heavy Light+Semijoins: decomposes Q into ``residual semijoin queries'' *(recursively)* based on degrees

But only when arities ≤ 2 or several special cases

Open: $L=O(IN/p^{1/\rho^*})$ for general queries?

Example Difficult Query



$\rho^* = 2$

$\psi^* = 3$

Open: $L=O(IN/p^{1/2})$ in $O(1)$ rounds?

Scalability Limitation of $L=IN/p^{1/\rho^*}$ (1)

- ρ^* can be very high for even small-size queries

$$A_0 \overset{R_1}{\text{———}} A_1 \overset{R_2}{\text{———}} \cdots \overset{R_{20}}{\text{———}} A_{20}$$

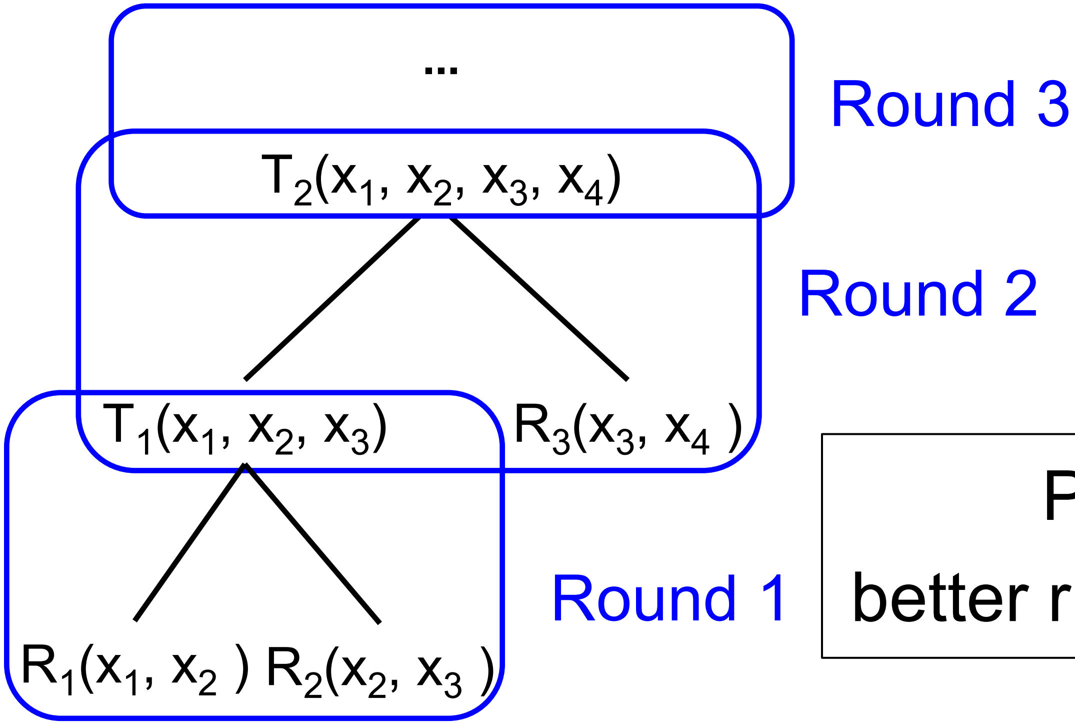
$$\rho^* = 10$$

- HC or Heavy-Light+Semijoin give poor scalability

2x speedup requires 1024x more processors

Scalability Limitation of L=IN/p^{1/ρ^*} (2)

- Iterative BJ might generate a lot of intermediate data



Problem: $|T_i| > p|IN|$

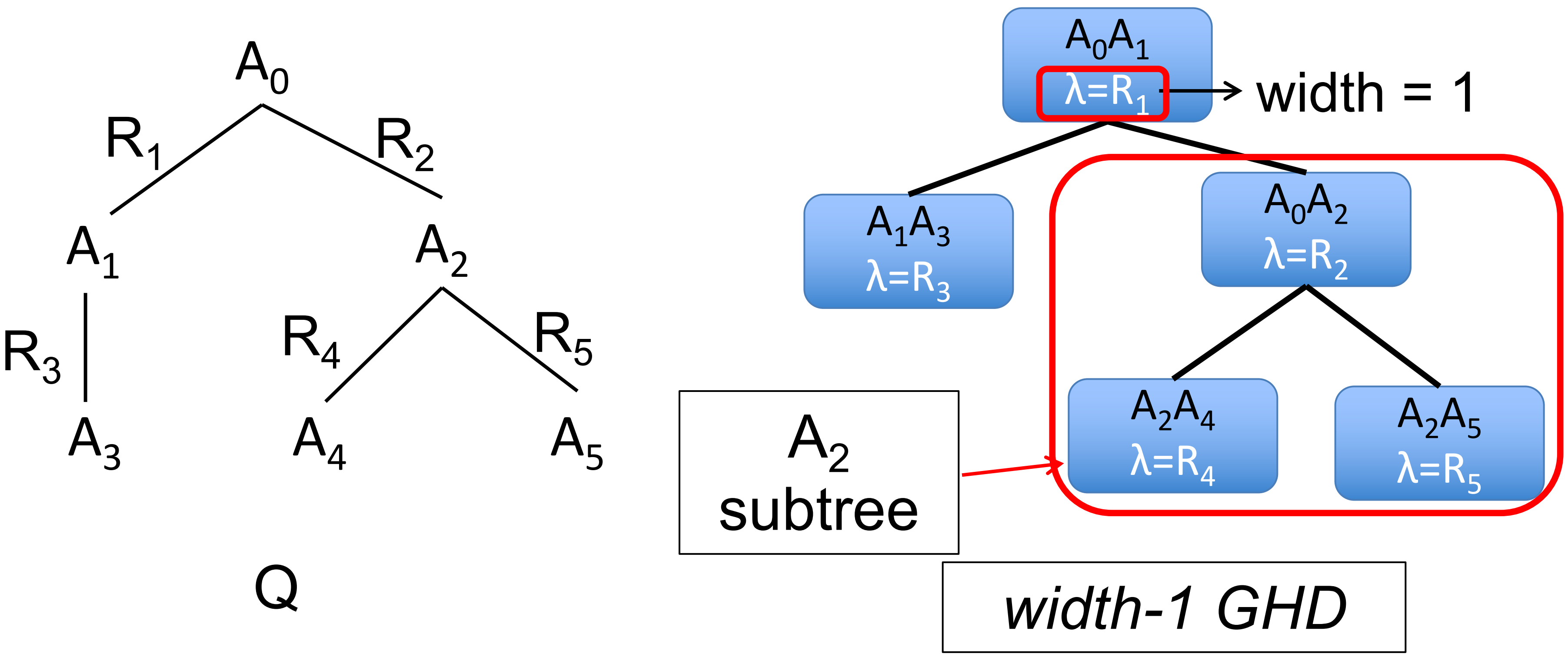
better run 1 round & replicate IN

Upshot: Semijoins can help if OUT is small

Yannakakis Algorithm For Acyclic Queries

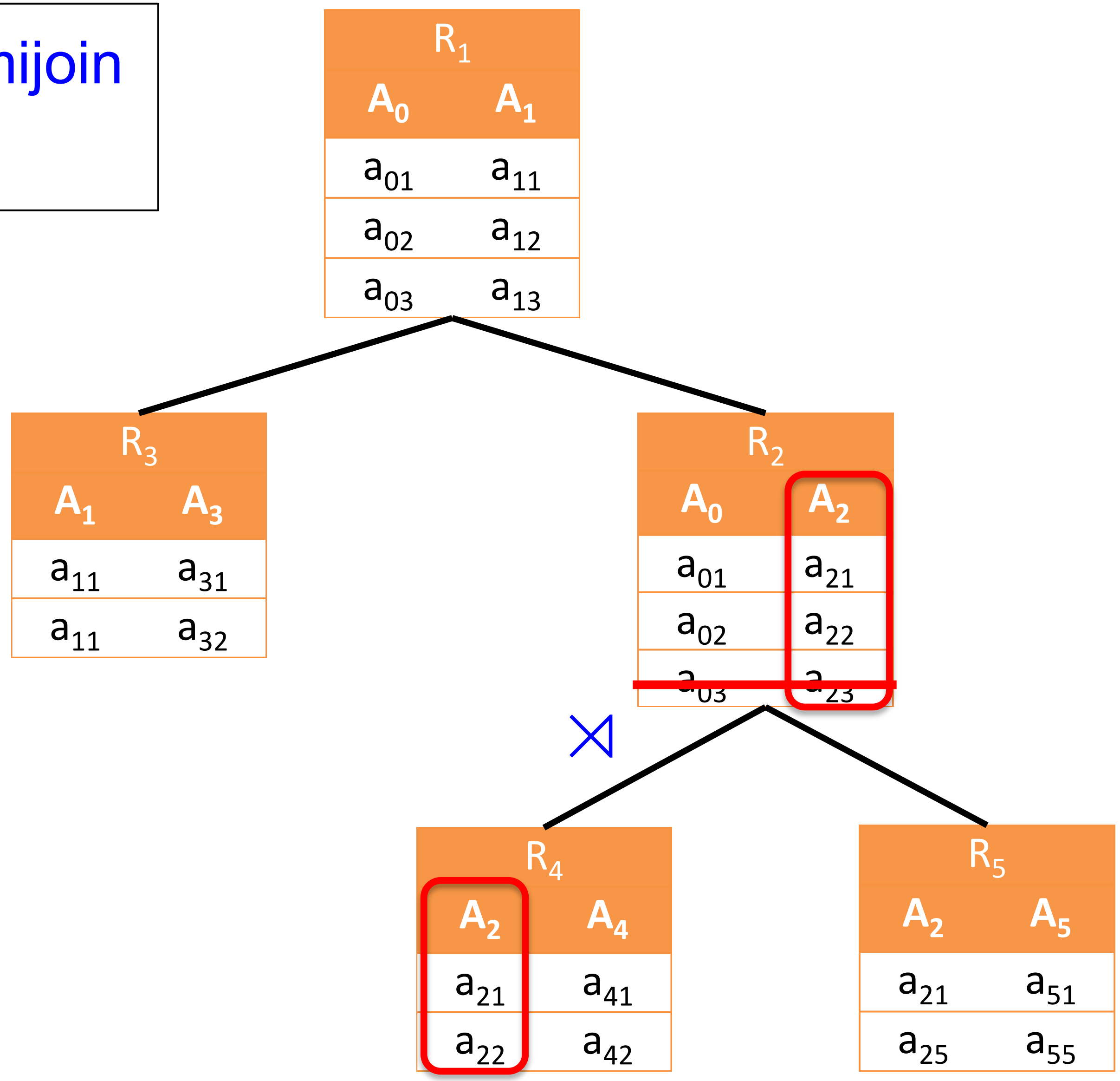
Input: Acyclic query Q: R1 ⋈ R2 ⋈ … ⋈ Rn &

a width-1 *Generalized Hypertree Decomposition* of Q



Yannakakis Algorithm For Acyclic Queries

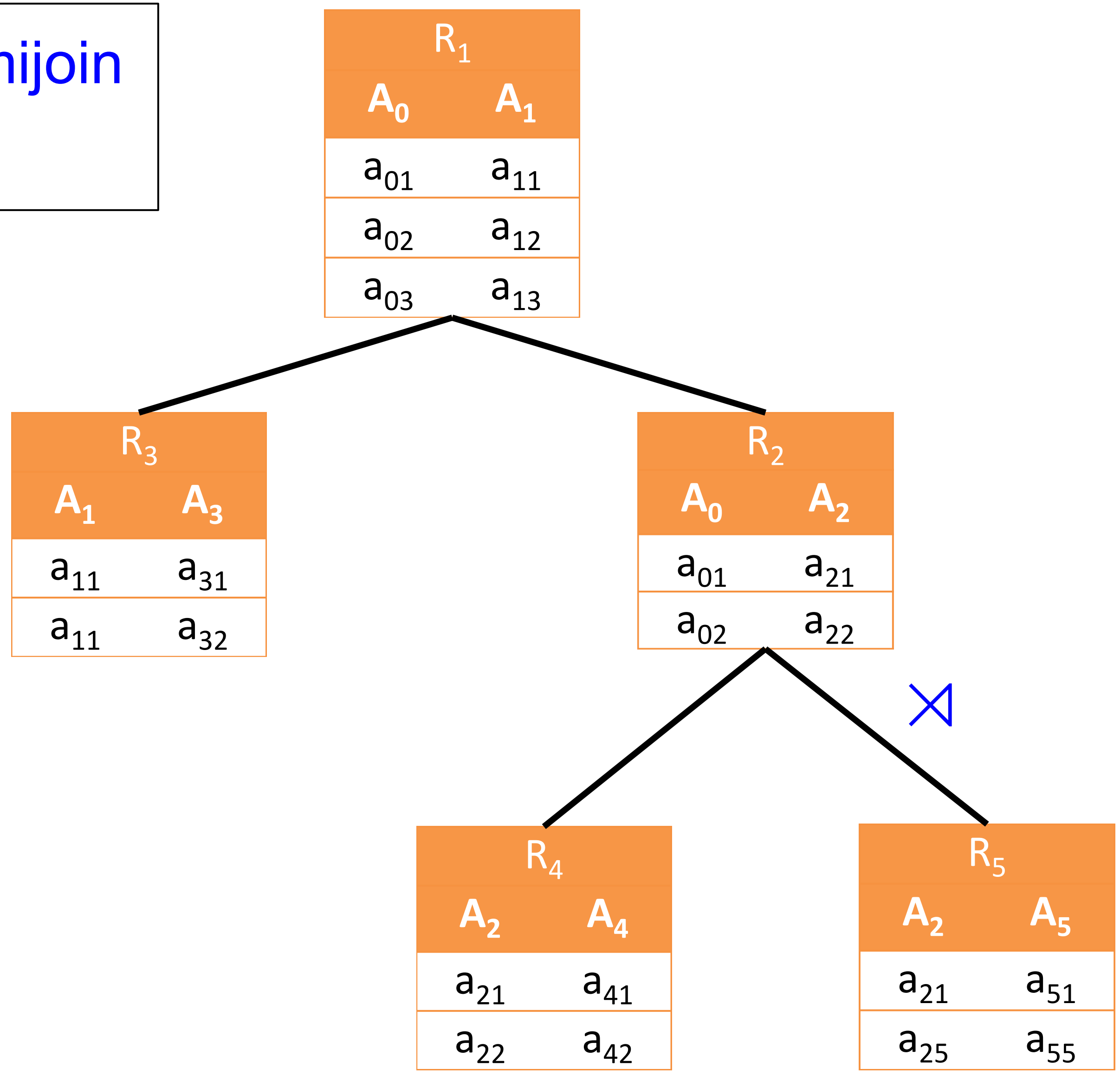
Upward Semijoin Phase



width-1 GHD

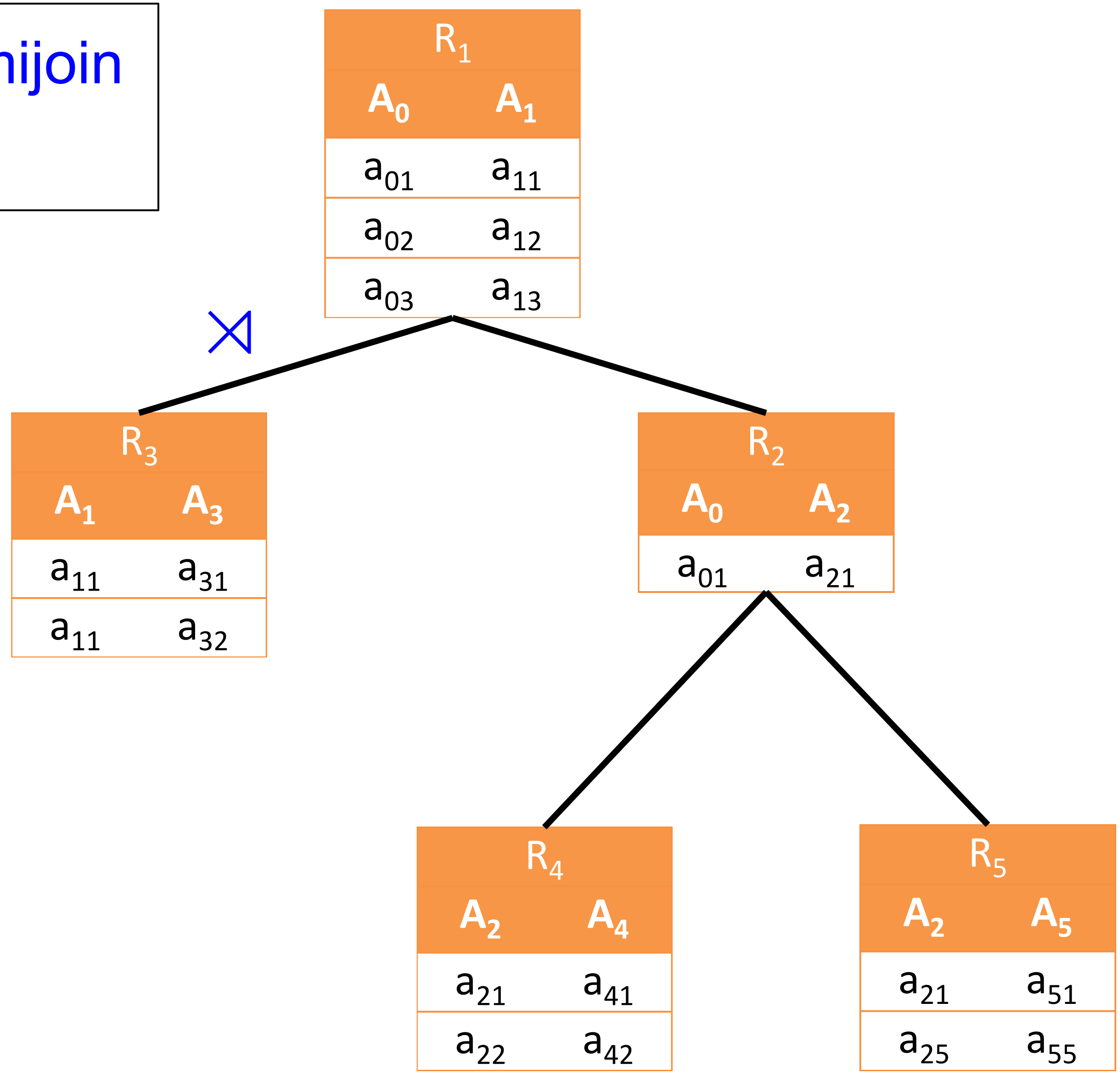
Yannakakis Algorithm For Acyclic Queries

Upward Semijoin Phase



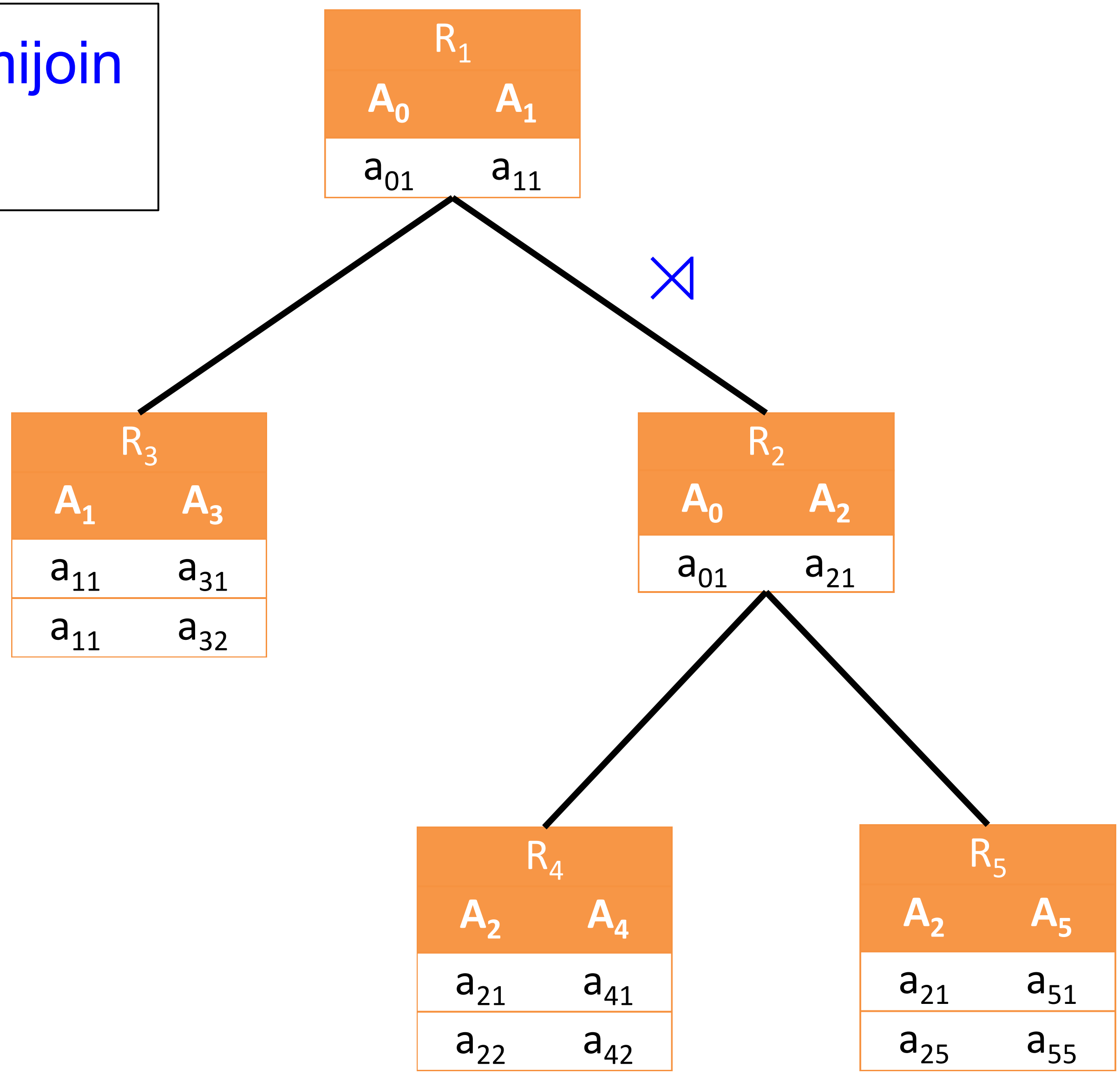
Yannakakis Algorithm For Acyclic Queries

Upward Semijoin Phase



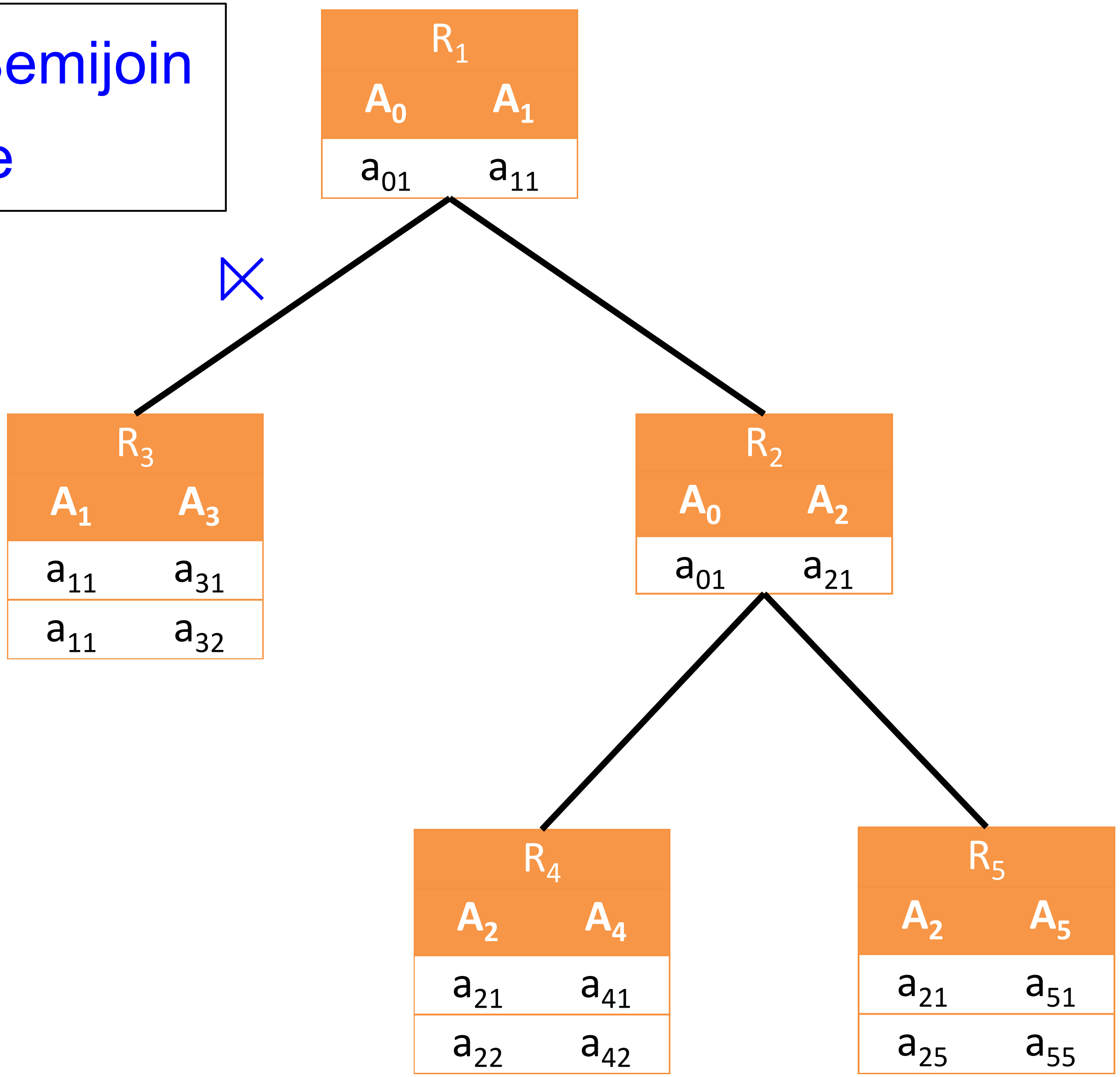
Yannakakis Algorithm For Acyclic Queries

Upward Semijoin Phase



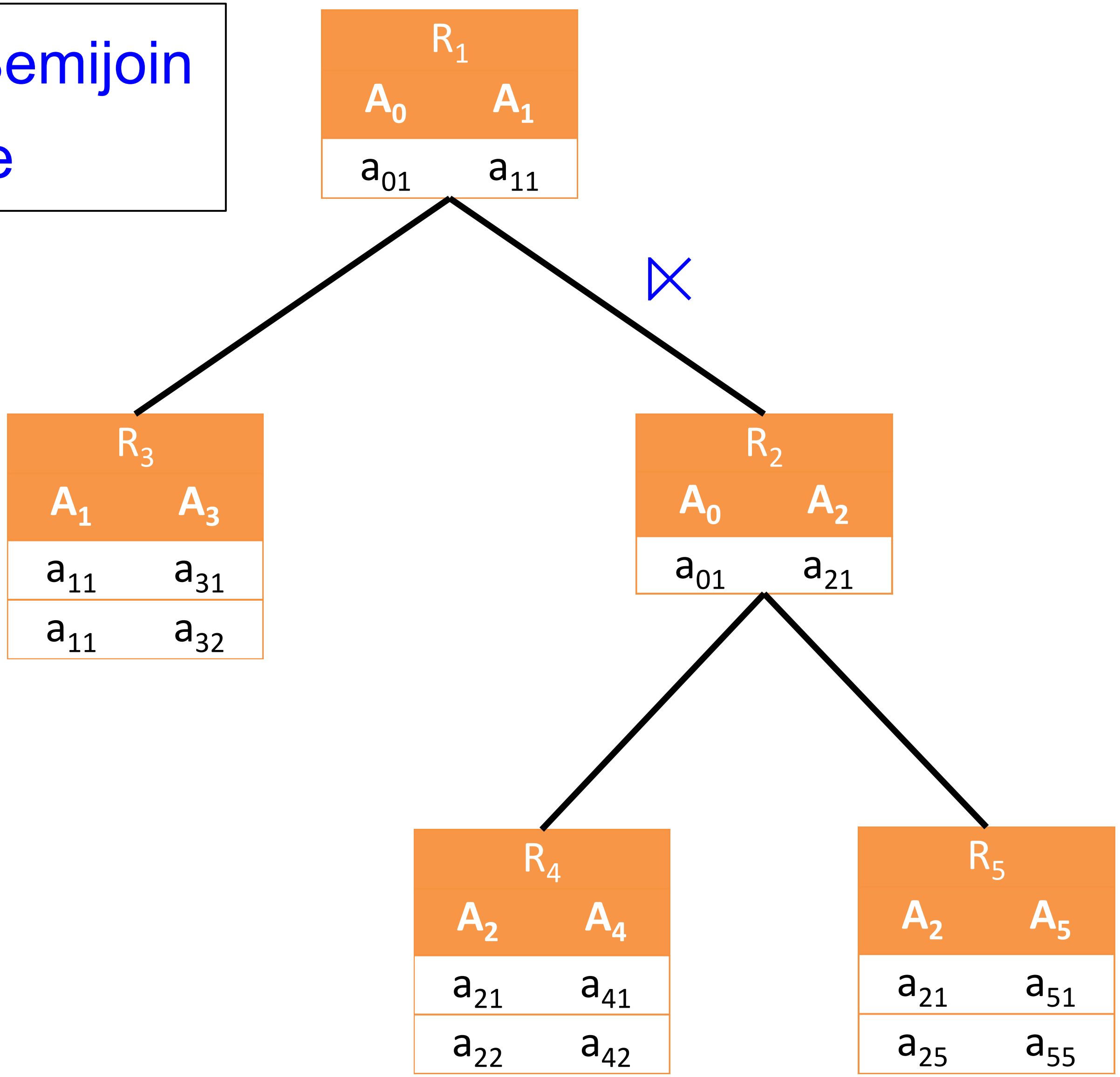
Yannakakis Algorithm For Acyclic Queries

Downward Semijoin Phase



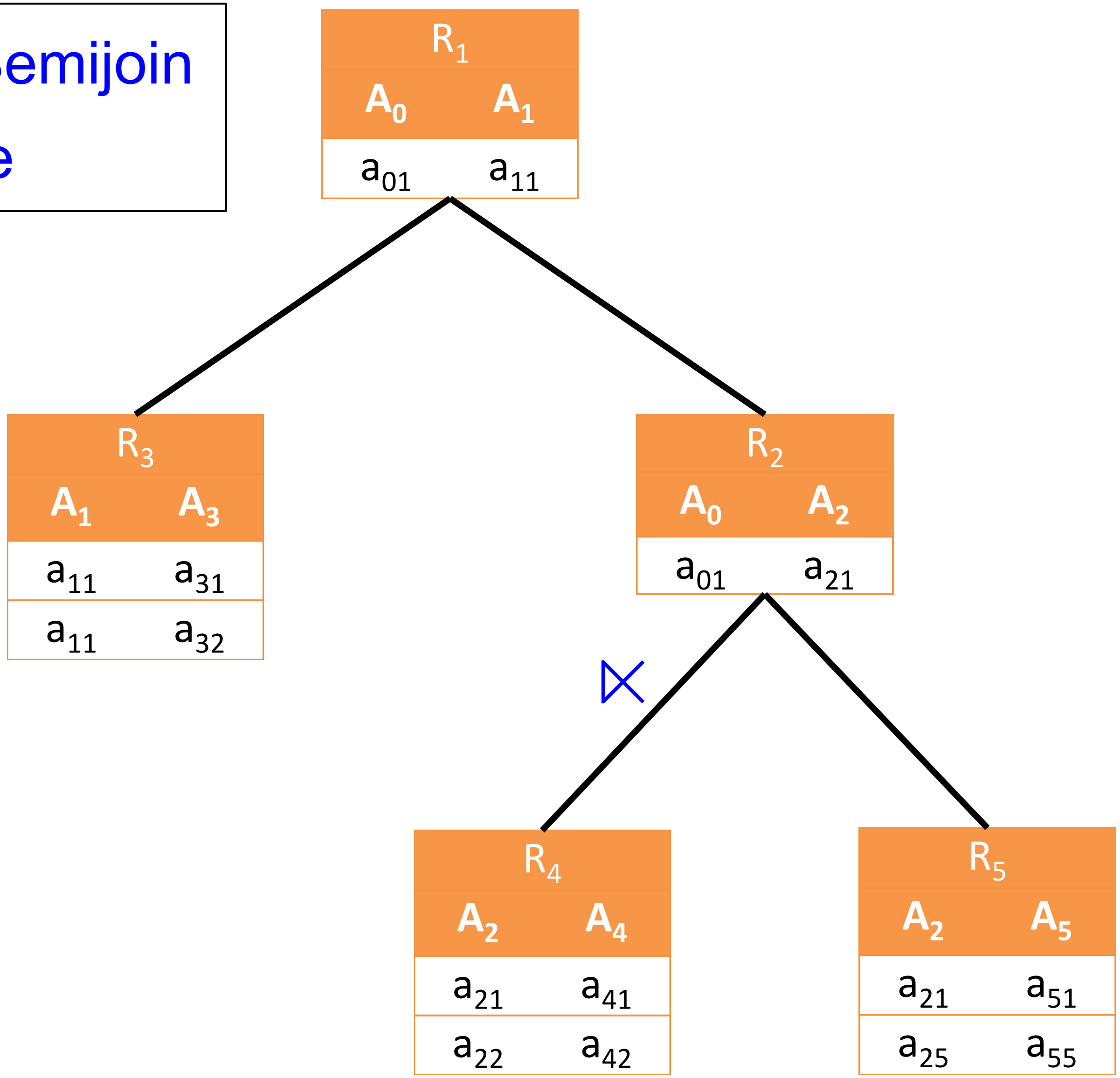
Yannakakis Algorithm For Acyclic Queries

Downward Semijoin Phase



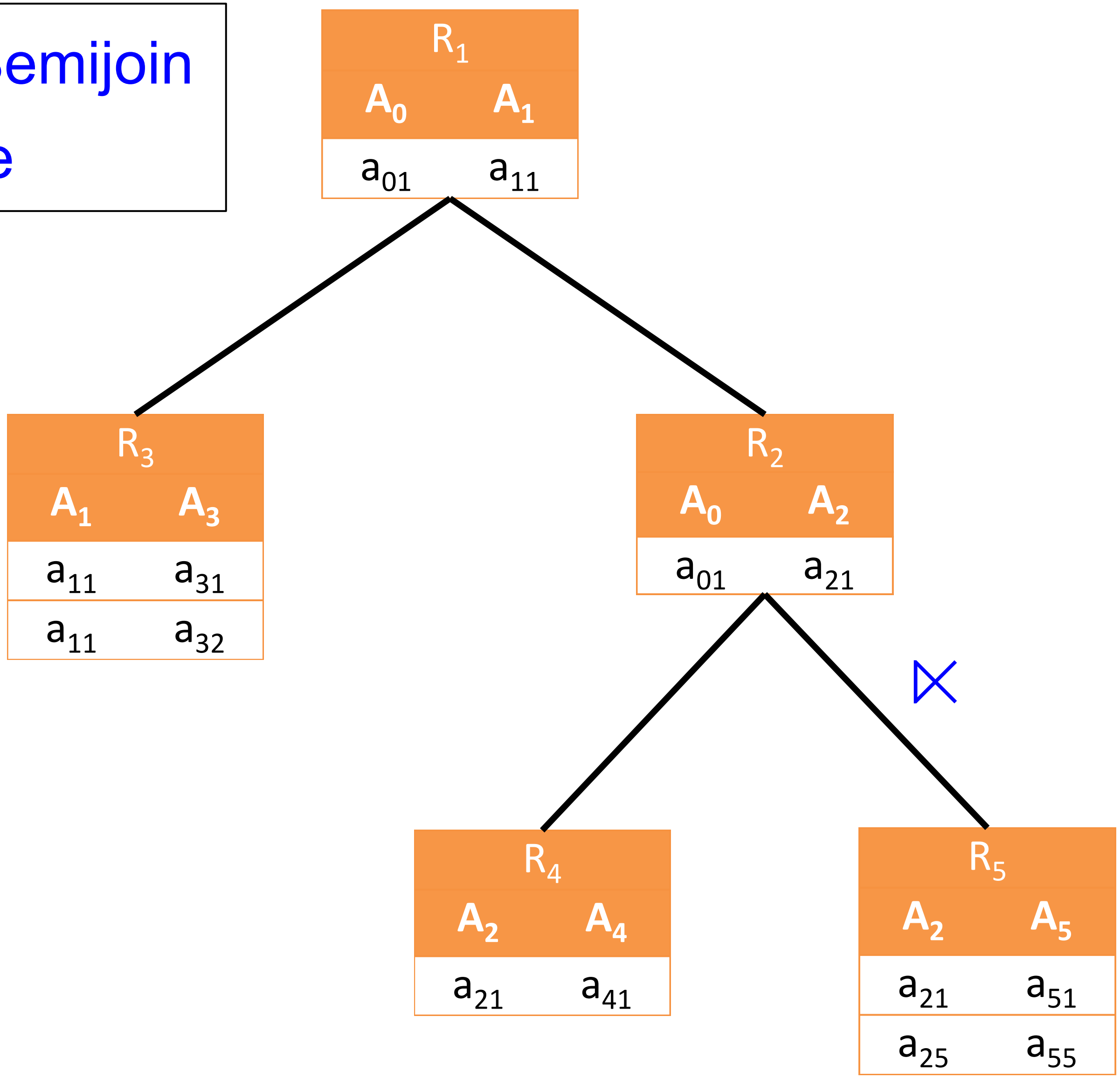
Yannakakis Algorithm For Acyclic Queries

Downward Semijoin Phase



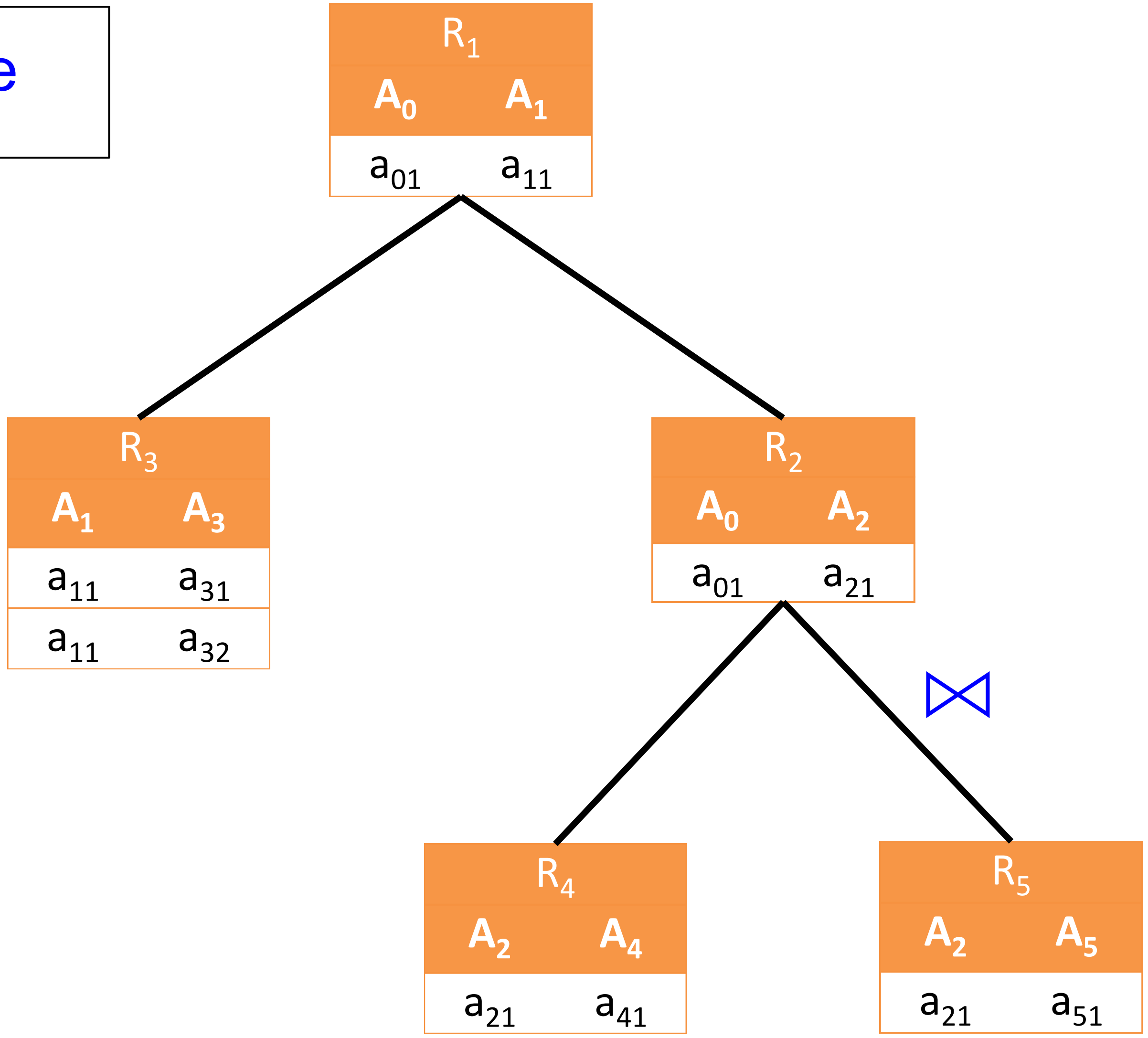
Yannakakis Algorithm For Acyclic Queries

Downward Semijoin Phase



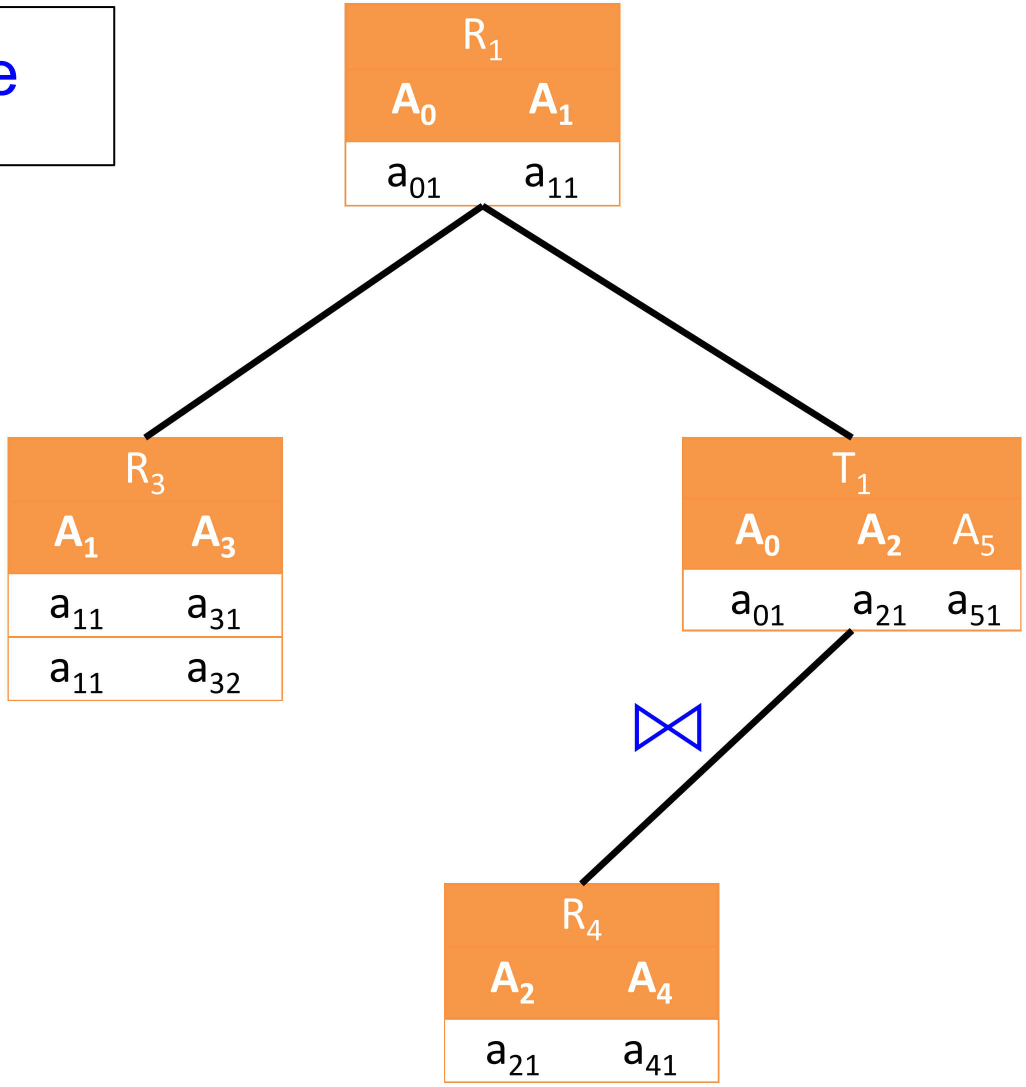
Yannakakis Algorithm For Acyclic Queries

Join Phase



Yannakakis Algorithm For Acyclic Queries

Join Phase



Yannakakis Algorithm For Acyclic Queries

Join Phase

R_1

A_0	A_1
a_{01}	a_{11}

R_3

A_1	A_3
a_{11}	a_{31}
a_{11}	a_{32}

$\bowtie$

T_2

A_0	A_2	A_4	A_5
a_{01}	a_{21}	a_{41}	a_{51}

Yannakakis Algorithm For Acyclic Queries

Join Phase

T_3				
A_0	A_1	A_2	A_4	A_5
a_{01}	a_{11}	a_{21}	a_{41}	a_{51}

$\bowtie$

R_3	
A_1	A_3
a_{11}	a_{31}
a_{11}	a_{32}

Yannakakis Algorithm For Acyclic Queries

OUT					
A_0	A_1	A_2	A_3	A_4	A_5
a_{01}	a_{11}	a_{21}	a_{31}	a_{41}	a_{51}
a_{01}	a_{11}	a_{21}	a_{32}	a_{41}	a_{51}

O(n) semijoins + O(n) joins

Serial Run-Time: O(IN+OUT)

because $|T_i| \leq$ OUT

GYM: Distributed Yannakakis on Acyclic Q

- Naively distribute semijoins&joins separate rounds:

$$r = O(n) \qquad L = O\left(\frac{IN + OUT}{p}\right)$$

- Linear scalability: 2x processors, 2x speed up

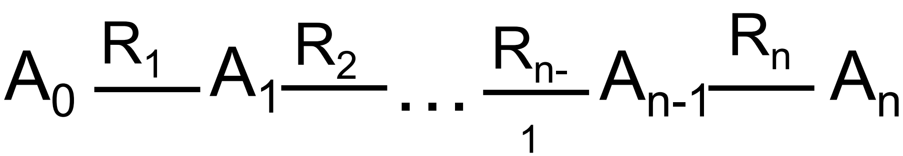
	GYM r=O(n)		HL + Semijoins r = O(1), arity 2
OUT< p^{1-1/ρ^*}IN	L = (IN +OUT)/p	≤	L = IN/p^{1/ρ^*}

Larger p allows for larger OUT

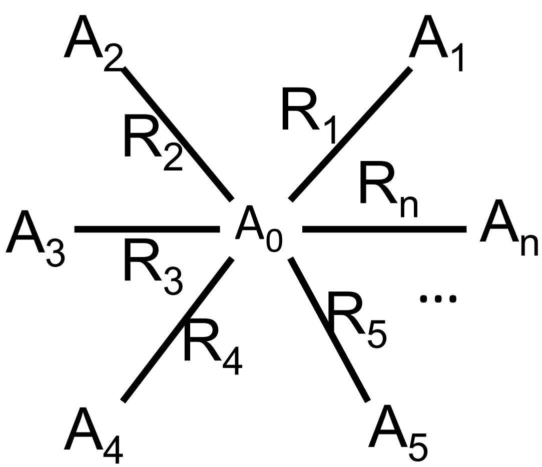
GYM in O(d) Rounds

- Acyclic queries have w-1 GHDs w/ different depths

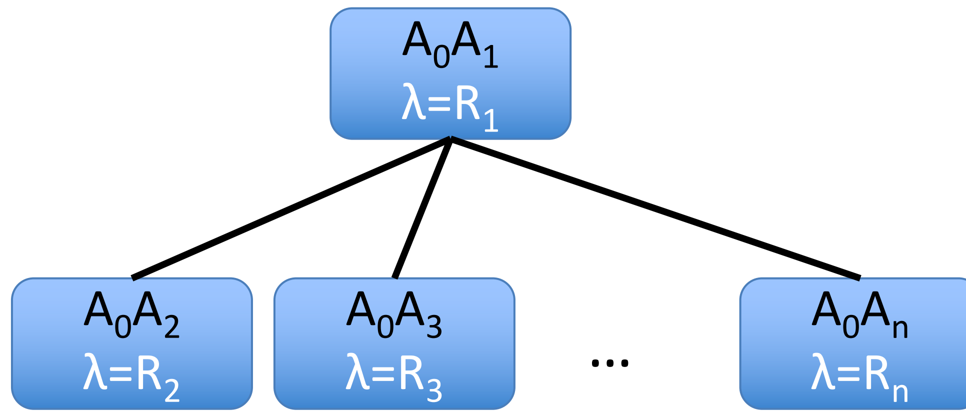
Path-n



Star-n



Lowest depth w-1 GHD:
$d = \Theta(n)$

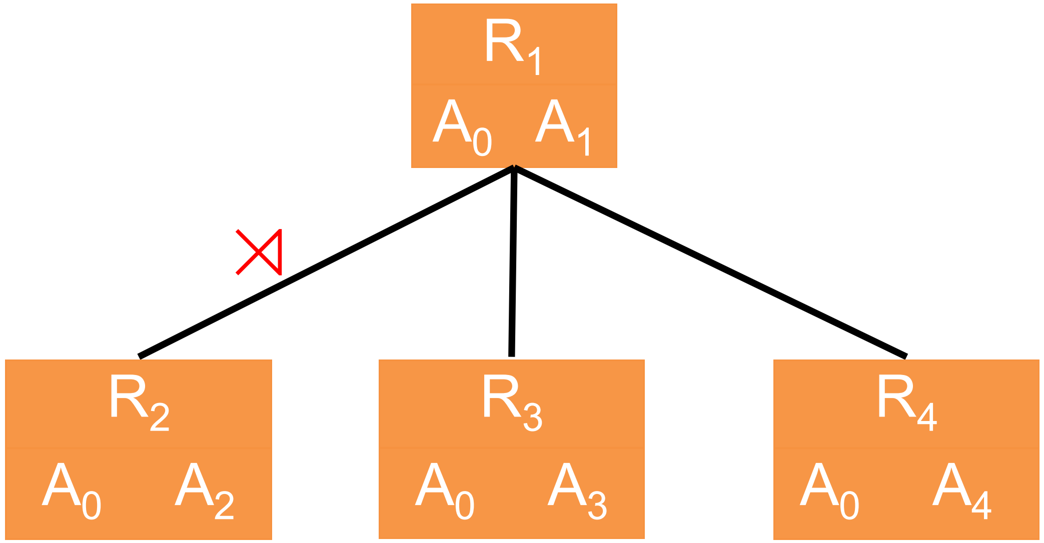


d=1

Can optimize GYM to run in r = O(d) and same L
w/ parallel joins and semijoins

Ex: Vanilla GYM

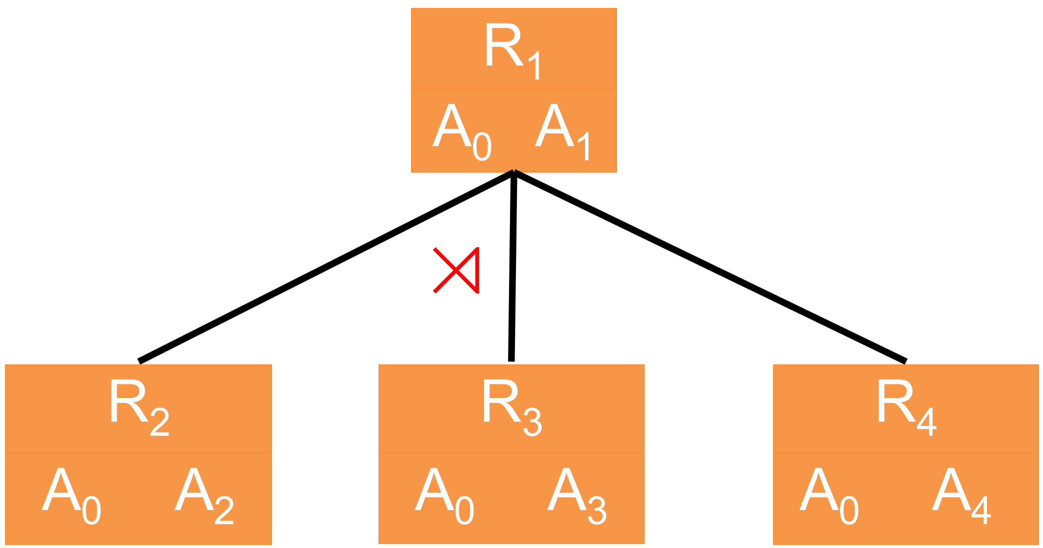
Upward Semijoin Phase



Round 1

Ex: Vanilla GYM

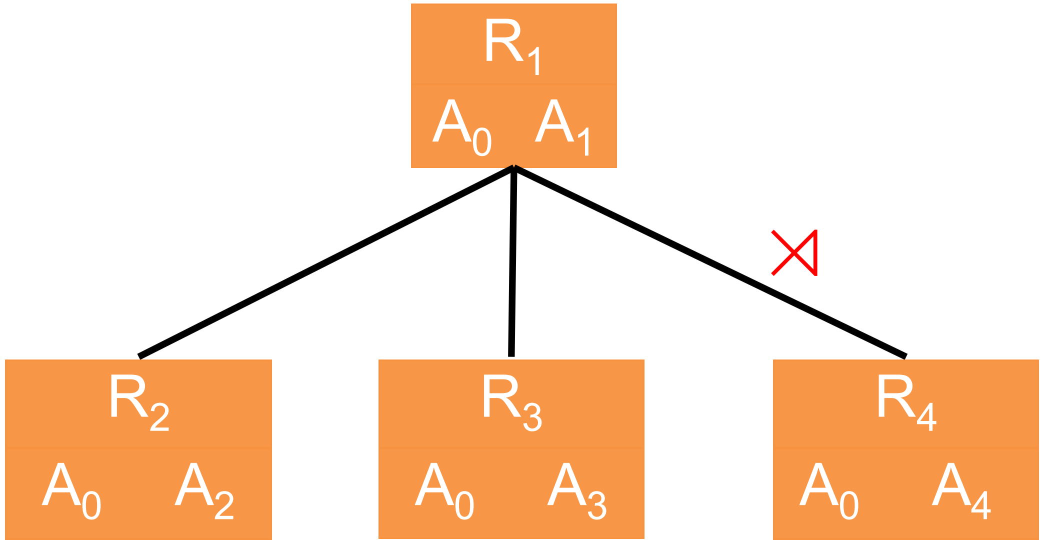
Upward Semijoin Phase



Round 2

Ex: Vanilla GYM

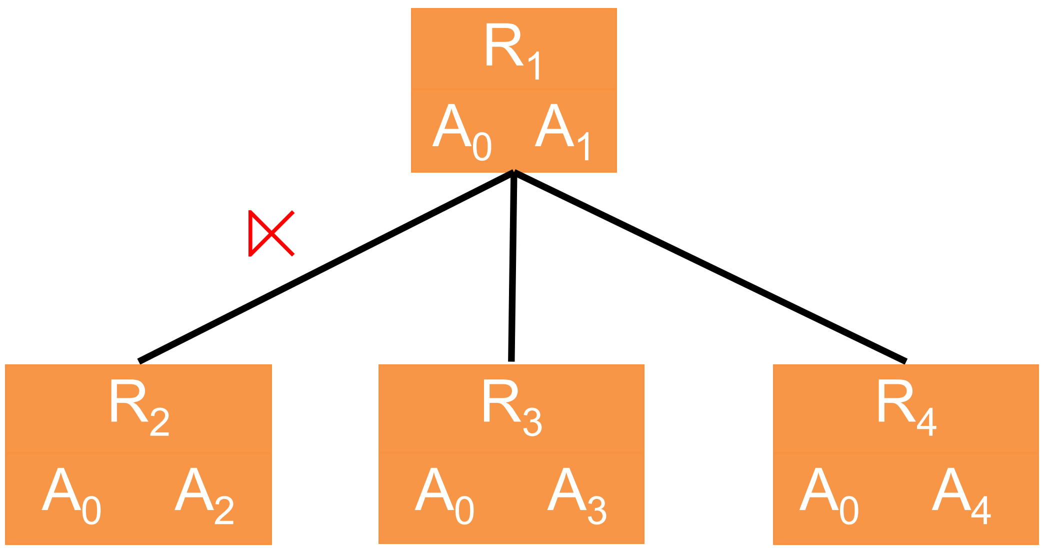
Upward Semijoin Phase



Round 3

Ex: Vanilla GYM

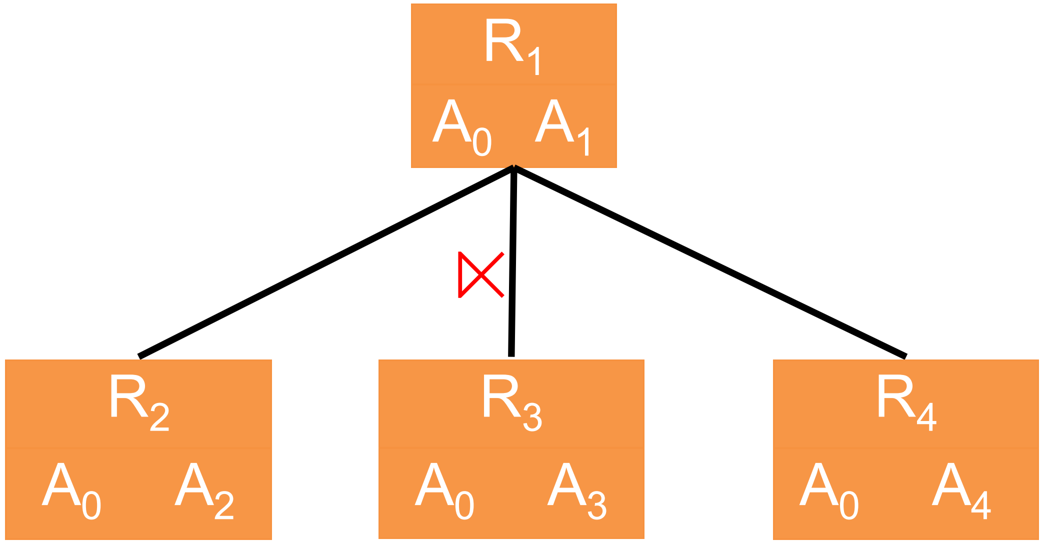
Downward Semijoin Phase



Round 4

Ex: Vanilla GYM

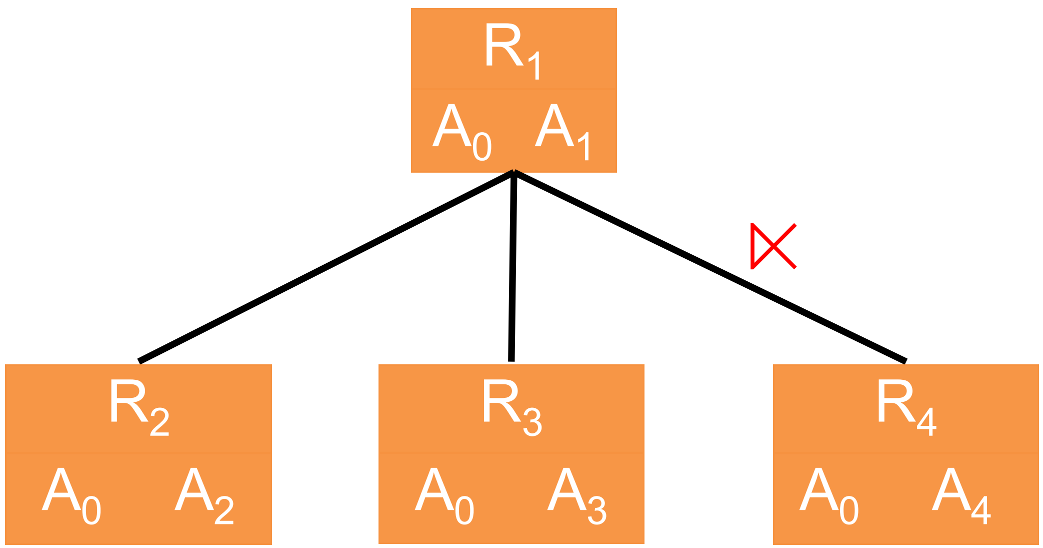
Downward Semijoin Phase



Round 5

Ex: Vanilla GYM

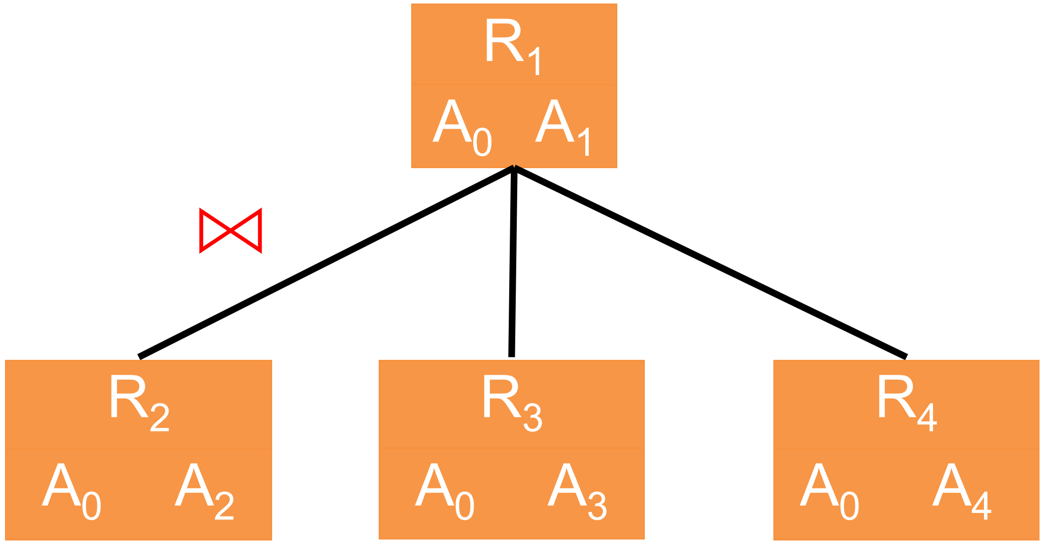
Downward Semijoin Phase



Round 6

Ex: Vanilla GYM

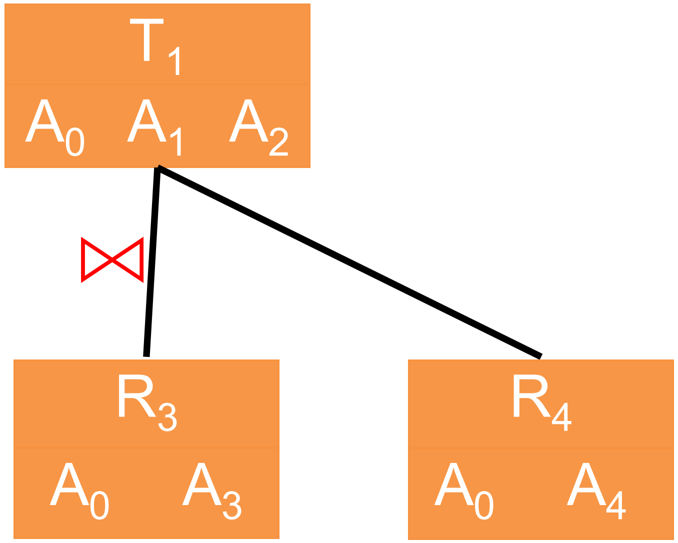
Join Phase



Round 7

Ex: Vanilla GYM

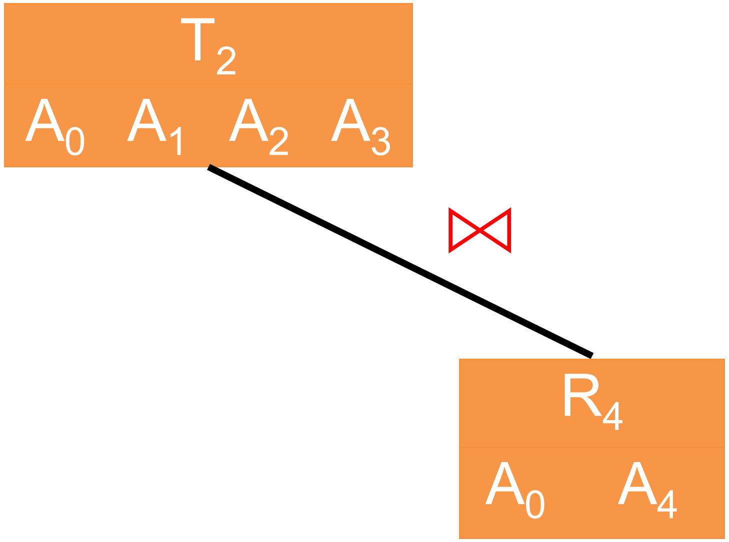
Join Phase



Round 8

Ex: Vanilla GYM

Join Phase



Round 9

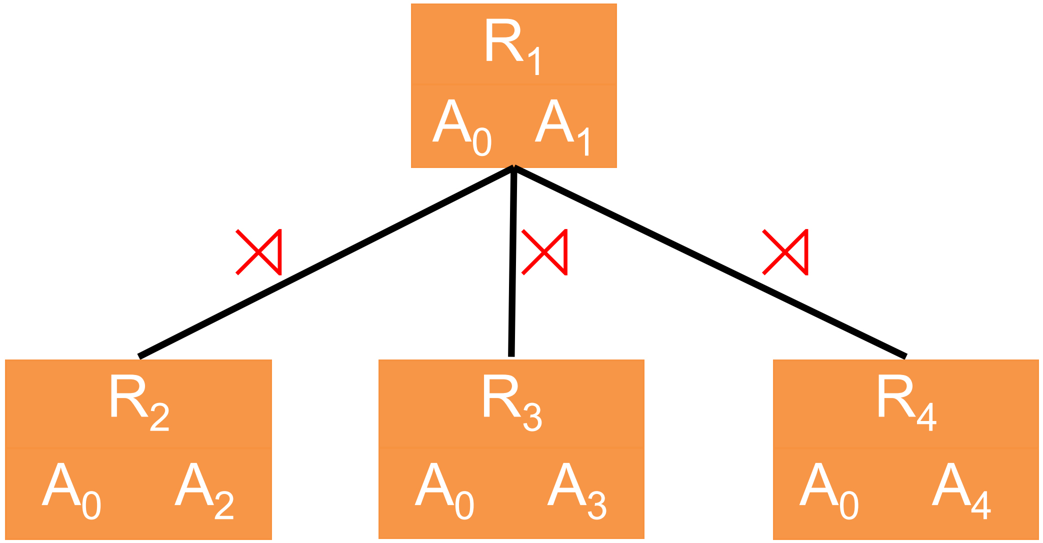
Ex: Vanilla GYM

$$r = 9$$

$$L = O\left(\frac{IN + OUT}{p}\right)$$

Ex: Optimized GYM

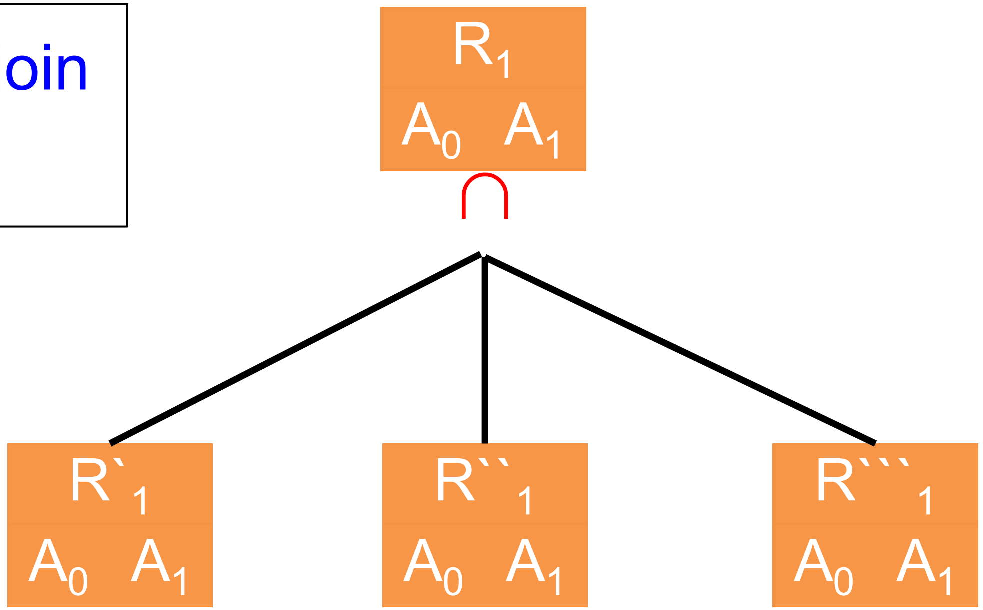
Upward Semijoin Phase



Round 1

Ex: Optimized GYM

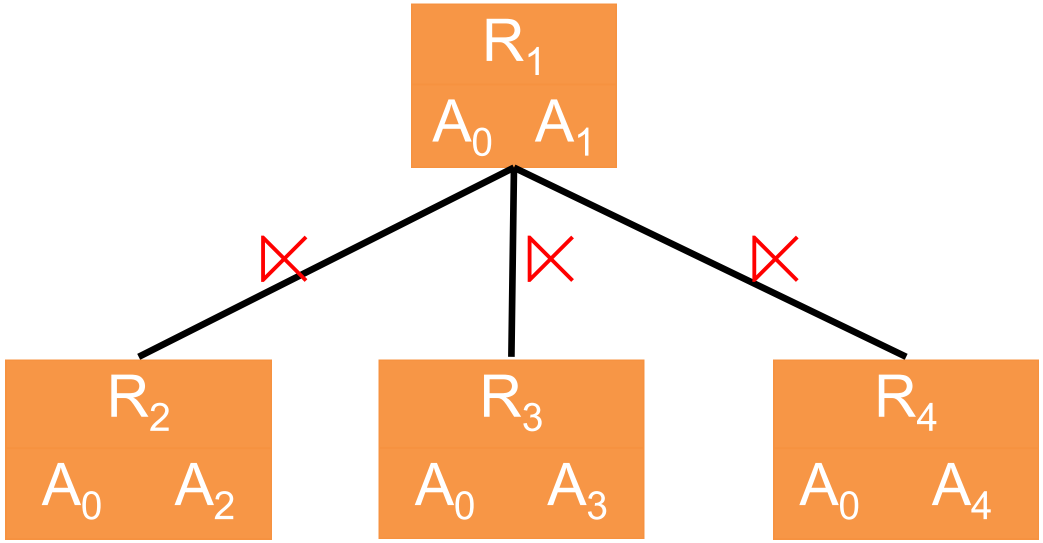
Upward Semijoin Phase



Round 2

Ex: Optimized GYM

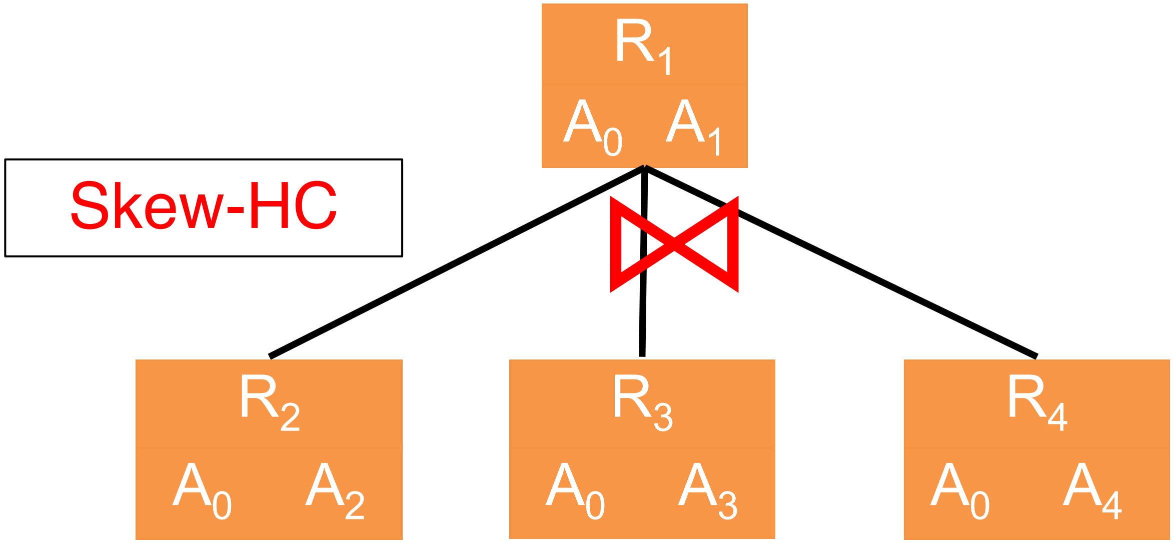
Downward Semijoin Phase



Round 3

Ex: Optimized GYM

Join Phase



Round 4

Ex: Optimized GYM

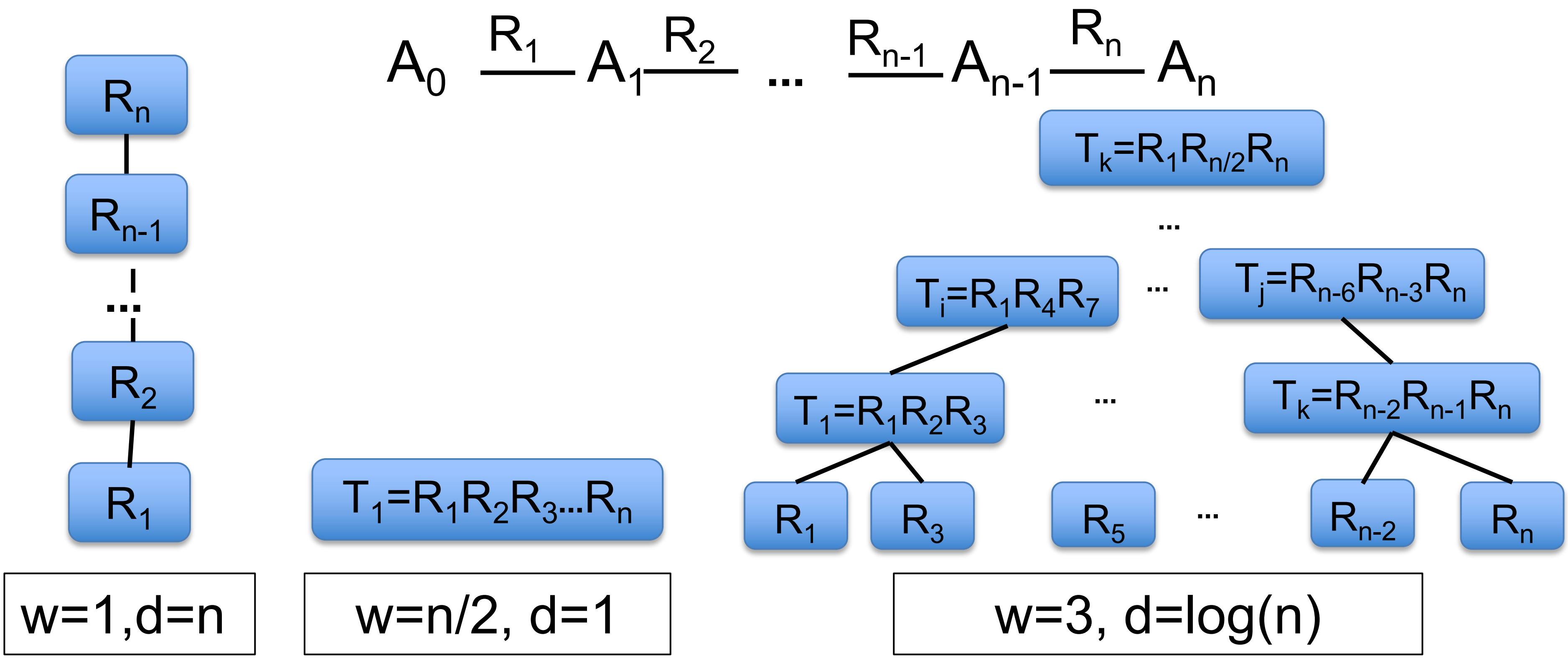
$r = 4$

$$L = O\left(\frac{IN + OUT}{p}\right)$$

OUT				
A_0	A_1	A_2	A_3	A_4

Generalizing To Any Query and GHD

Any Q with width-w, depth-d GHD can run in:
$r=O(d)$, $L=O((IN^w + OUT)/p)$



Can tradeoff r and L by constructing GHDs
with different w and d

Main Takeaways

1. High-degree residual query decompositions + semijoins:

   $L=O(IN/p^{1/\rho^*})$ (for some Qs)

2. Yannakakis style processing improves L if OUT is small

3. Depth & width of decompositions allow r & L tradeoffs

Multi-round Multiway Joins In Practice

- Most Systems: Iterative Binary Join Plans
- Tributary Join [Chu et al. SIGMOD '15]
- Subgraph Queries:
  - BiGJoin [Ammar et al., VLDB '18]
  - SEED [Lai et al., VLDB Journal, '17]
  - TwinTwigJoin [Lai et al., VLDB '16]
  - PSgL [Shao et al., SIGMOD '14]

Outline

- Models of parallel computation (Dan)
- Two-way joins (Paris)
- Multi-way joins (Paris+Semih)
- Sorting & Matrix multiplication (Paris+Semih)
- Conclusion (Dan)

Sorting

- Sorting is a fundamental operation in data processing
  - join computation (parallel merge join)
  - similarity joins
  - aggregation/grouping
- input size: N items

The PSRS algorithm

Parallel **S**ort by **R**egular **S**ample (PSRS)

- find $p-1$ values, called *splitters*

  $$-\infty = y_0 < y_1 < \cdots < y_{p-1} < y_p = +\infty$$

- each server gets assigned one of the p intervals
- partition the data such that all items in the same interval are sent to the same server
- each server sorts the items locally

The PSRS algorithm

How does PSRS find the splitters?

- each server sorts its data locally, and computes the p-1 local splitters (called the *regular sample*) that partition uniformly the local data
- each server broadcasts its regular sample
- the final p splitters are computed by sorting the union of the local regular samples, and choosing every p-th item.

The PSRS algorithm: analysis

- PSRS achieves a load of $L = O(N/p)$, assuming that the number of servers p << $N^{1/3}$
- Modern implementations of PSRS replace the local sorting by *sampling* to improve performance

What happens for larger values of p?

Cole’s Algorithm

- designed for the PRAM model
- works for arbitrary number of processors p

sorts in time $O(N / p \log(N))$

- does not naturally extend to BSP or MPC, since the processors access the shared memory in patterns that are costly to convert into messages

Goodrich's Algorithm

- designed for BSP (hence can be adapted to MPC)
- works for arbitrary number of processors p

with load $L = N/p$, it runs in $O(\log_L(N))$ rounds

- the algorithm is very complex!

Lower Bounds on Sorting

Theorem The minimum number of rounds needed by any MPC algorithm to sort N items is $\Omega(\log_L N)$

Theorem The minimum communication needed by any MPC algorithm to sort N items is $\Omega(N \log_L N)$

- The lower bounds are independent of the number of servers p
- Having more processors doesn't improve communication or synchronization!

Sorting in Practice

- None of the optimal parallel algorithms are used in practice
- In real-world instances of sorting, parallelism is coarse-grained ($p << N$)
- Typical method: find the splitters, partition, and then sort locally

Year	Winner	Time	p and Memory/Processor
2016	Tencent Sort	134s	512 (512GB)
2015	FuxiSort	377s	3134 (96GB) + 243 (128GB)
2014	TritonSort	1378s	186 (244GB)
2014	Apache Spark	1406s	207 (244GB)
2013	Hadoop	4328s	2100 (64GB)
2011	TritonSort	8274s	52 (24GB)

Conventional Square Matrix Multiplication

n × n matrices:

A

3.2	6.7	…	7.4
4.3	2.7	…	8.7
…	…	…	…
-1.1	0.3	…	1.4

B

5.8	0.1	…	2.2
6.1	3.8	…	1.6
…	…	…	…
0.8	2.5	…	0.4

=

C

8.8	0.5	…	1.4
1.5	1.0	…	2.5
…	…	…	…
4.4	3.0	…	5.6

- Focus on Conventional Algs that do all n^3 products
  - i.e.: Strassen-like algs are not allowed
- Reflects practice & o.w. LBs are trivial
- Stateless MPC: Procs keep O(L) elements (memory)
- Analyze other parameters, p, r, C=prL

SQL Query of Matrix Multiplication

```
select A.i,B.k,sum(A.v*B.v)
from   A, B
where  A.j = B.j
group by A.i, B.k
```

Aggregation Part

Join Part

A (rows i, columns j):

3.2	6.7	…	7.4
4.3	2.7	…	8.7
…	…	…	…
-1.1	0.3	…	1.4

× B (rows j, columns k):

5.8	0.1	…	2.2
6.1	3.8	…	1.6
…	…	…	…
0.8	2.5	…	0.4

= C (rows i, columns k):

8.8	0.5	…	1.4
1.5	1.0	…	2.5
…	…	…	…
4.4	3.0	…	5.6

A

i	j	v
1	1	3.2
…	…	…
n	n	1.4

B

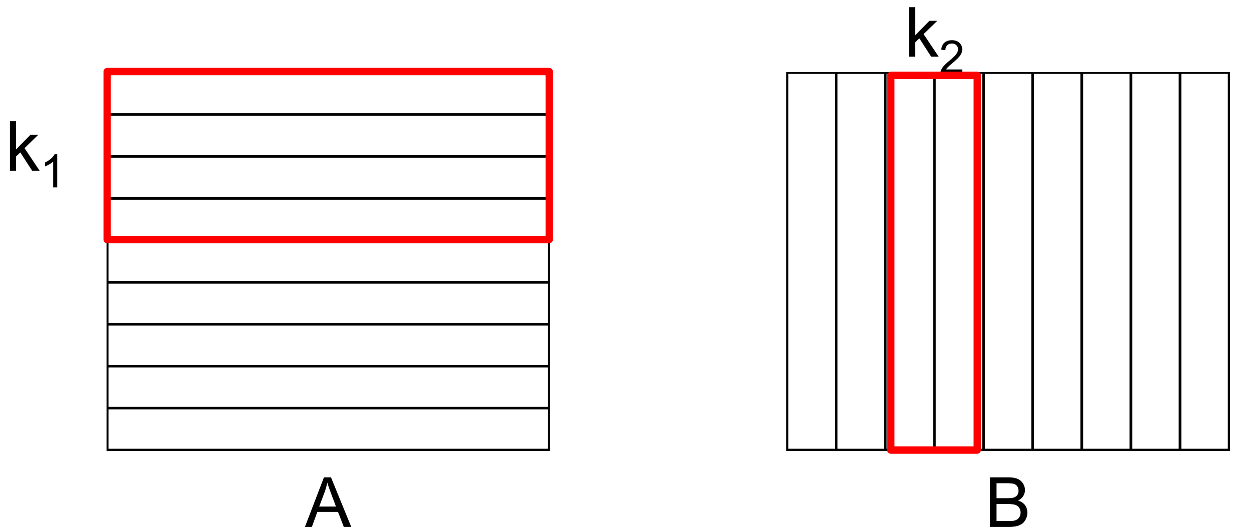
j	k	v
1	1	5.8
…	…	…
n	n	2.6

C

i	k	sum
1	1	8.8
…	…	…
n	n	5.6

1-round Algorithm (1)

- Need entire rows & cols: $2n \leq L \leq n^2$



- Suppose $L = 2tn$ (so each proc. can store 2t rows and cols)

Q: # rows k_1 & # cols k_2 should a processor get?

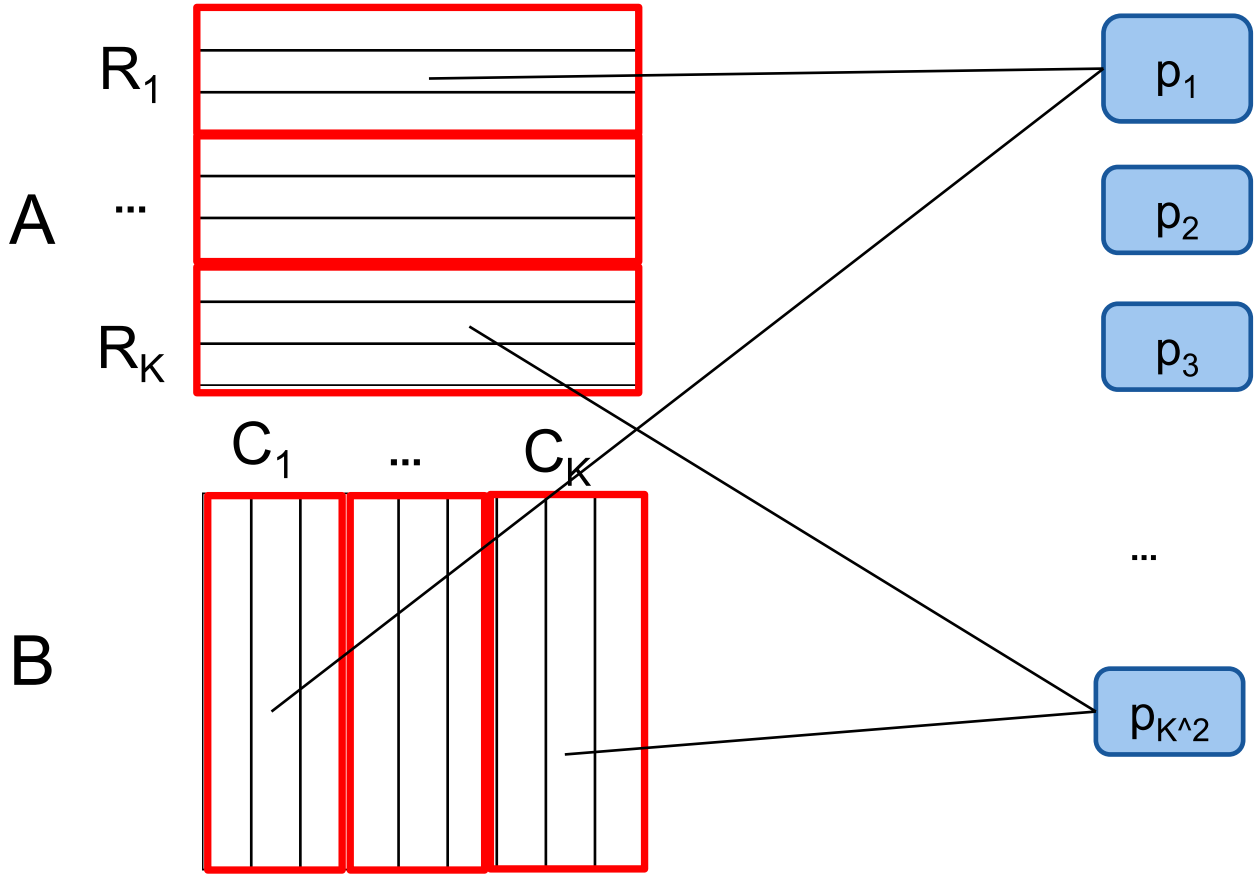
A: $k_1=k_2= t$ can do $k_1k_2n = t^2n$ products

Ex:t=3, n=75 => 9*75 = 675 prods/proc

[McKellar & Coffman, CACM, 1969], [Afrati et. al. VLDB, 2013]

1-round Algorithm (2)

L = 2tn: Divide into K=n/t rectangular groups



$1\ R_i$ & $1\ C_j$ per processor

$L = 2tn,\ C = K^2L \Rightarrow C = O(n^4/L)$

[McKellar & Coffman, CACM, 1969], [Afrati et. al. VLDB, 2013]

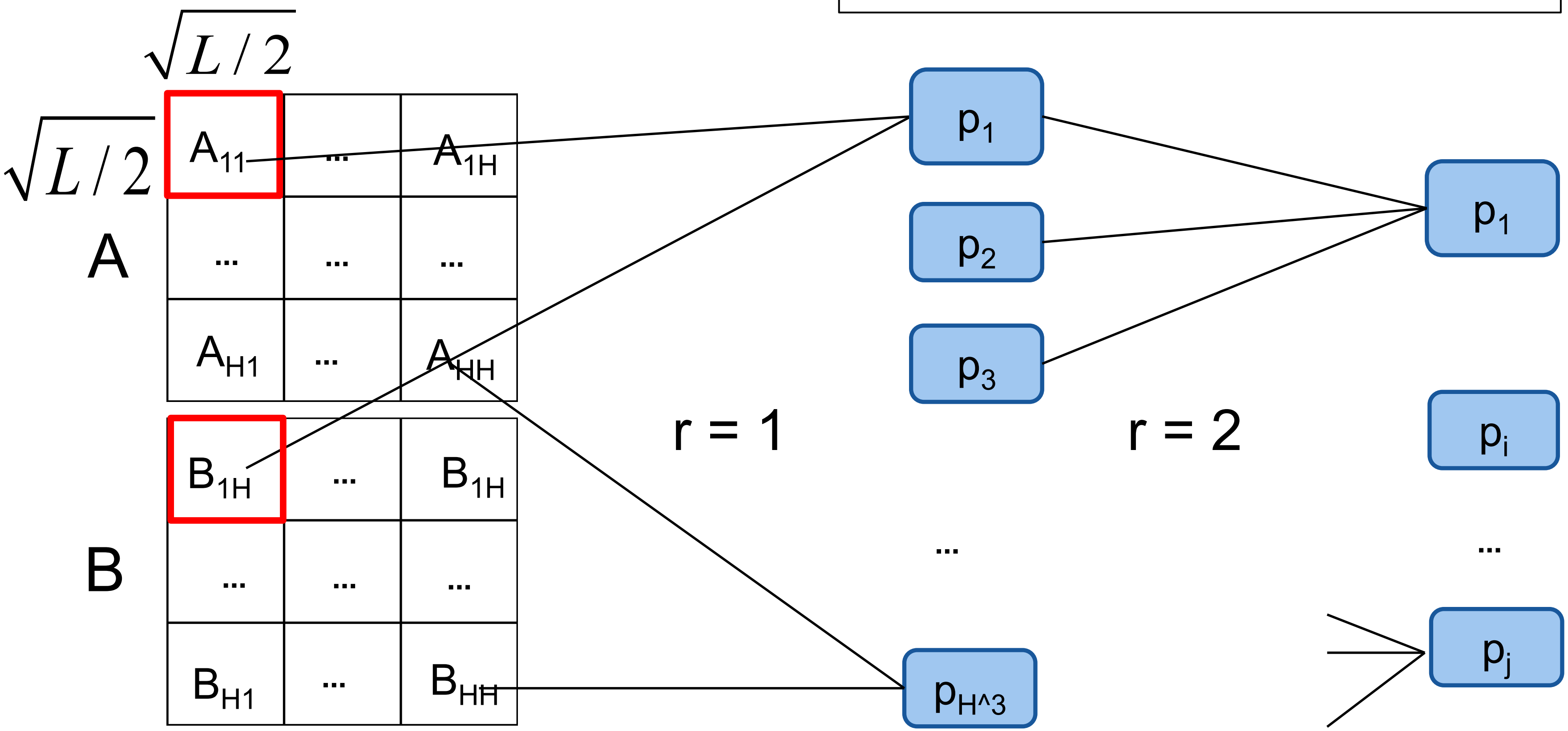
2-round Algorithm (1)

- Can do partial products & aggregate in separate rounds

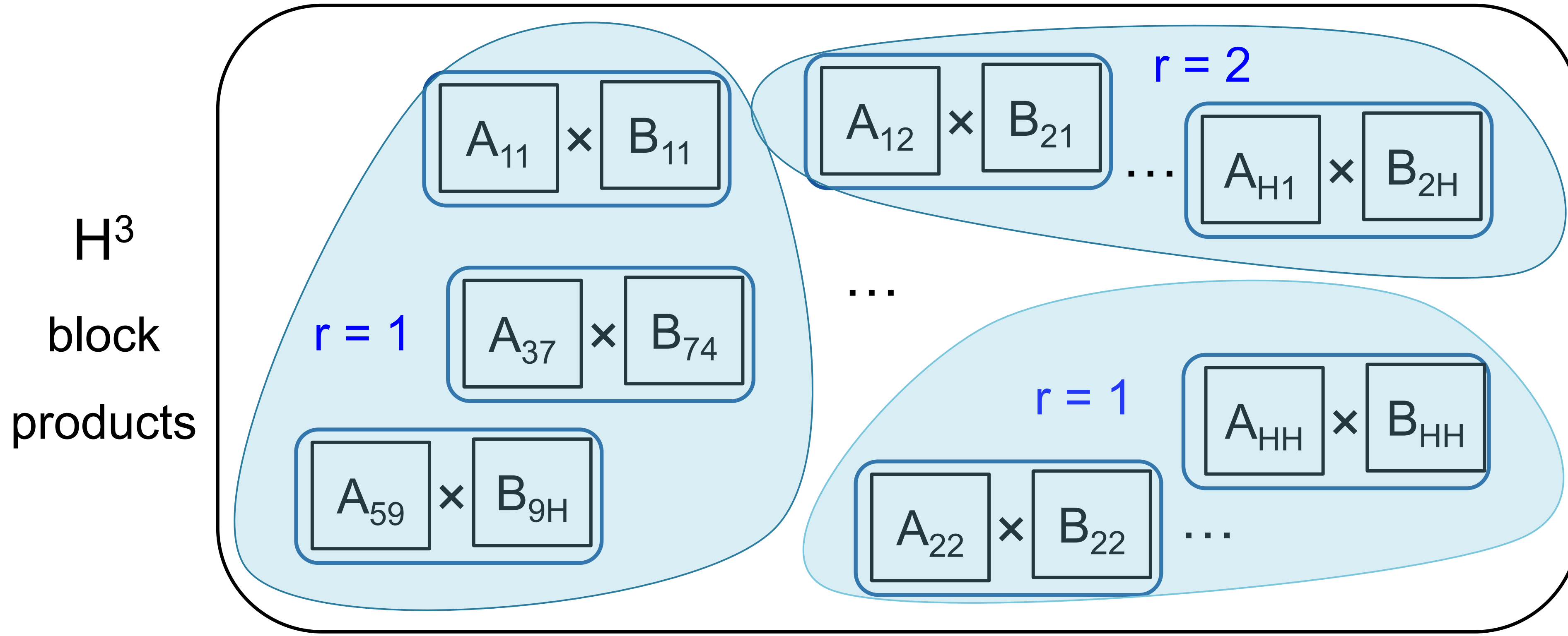
$$H = \frac{n}{\sqrt{L/2}} = \frac{n}{\sqrt{tn}}$$

prods/proc: $(tn)^{3/2}$

$(3*75)^{3/2}$=3375 vs 675



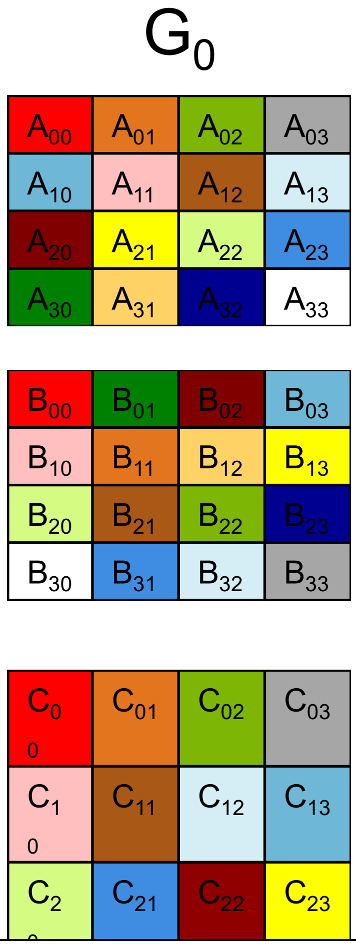
Generalizing to > 2 Rounds



- Group into H groups of H^2 s.t.
  G_z: $A_{i,j} \times B_{j,k}$ s.t. $j = (i + k + z) \bmod H$
- Each r: mult. as many block products as possible
  - w/ 1 proc doing 1 block product (+ partial aggregation)

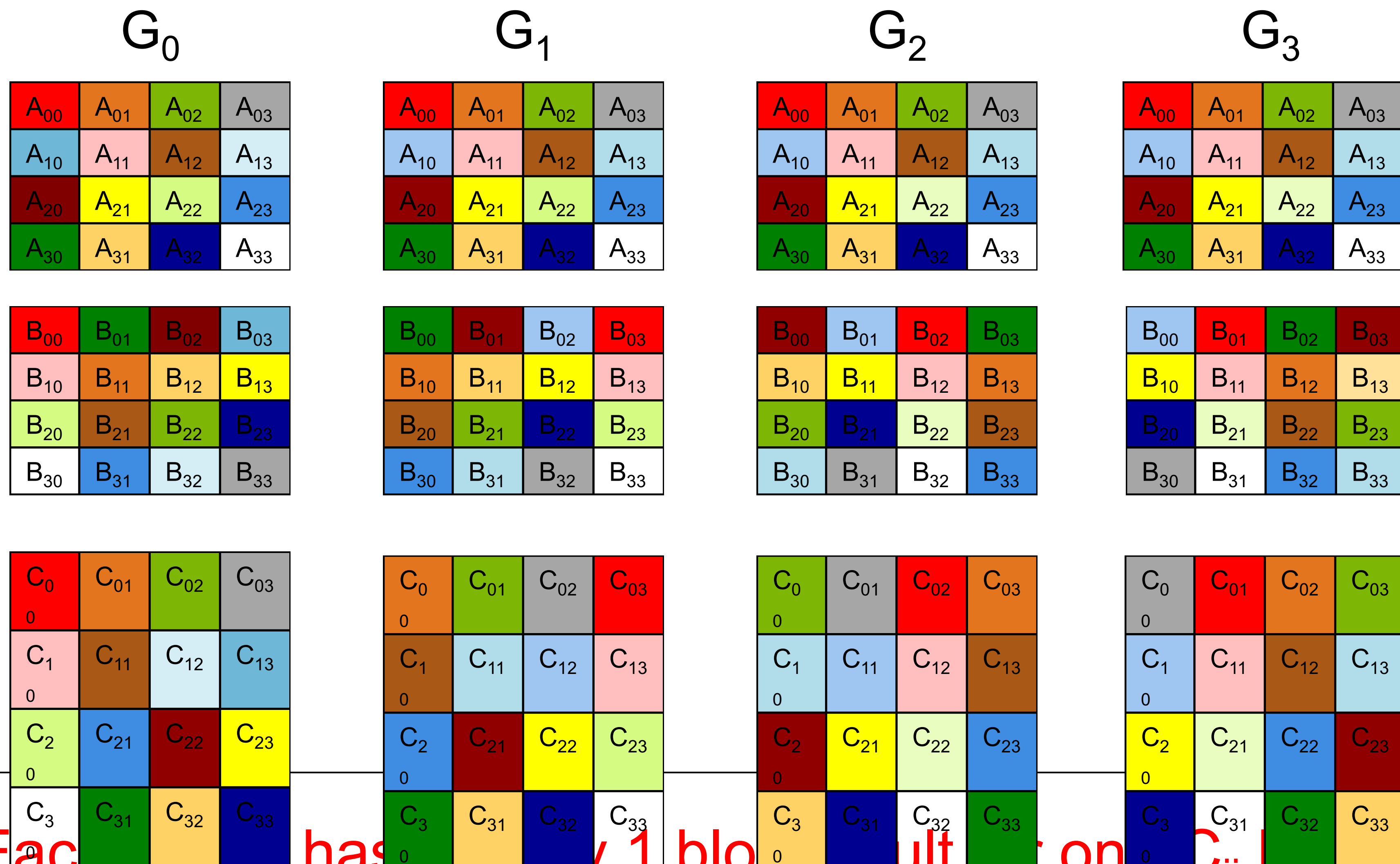
[McColl & Tiskin, Algoritmica, 1999], [Pietracaprina et. al., ICS, 2012]

Example Groups



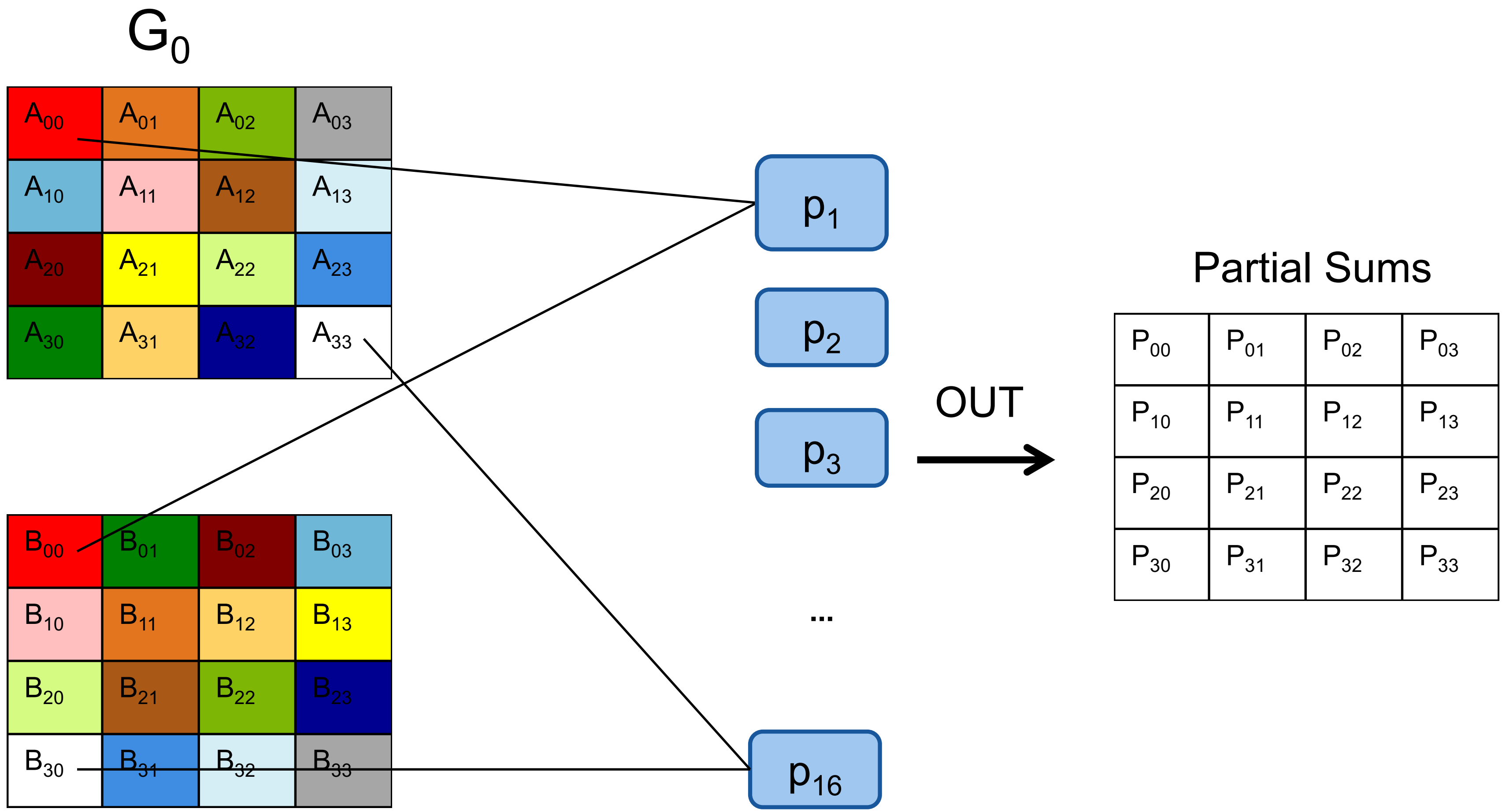
Each group has exactly 1 block-mult for one C_{ij} block

Example Groups



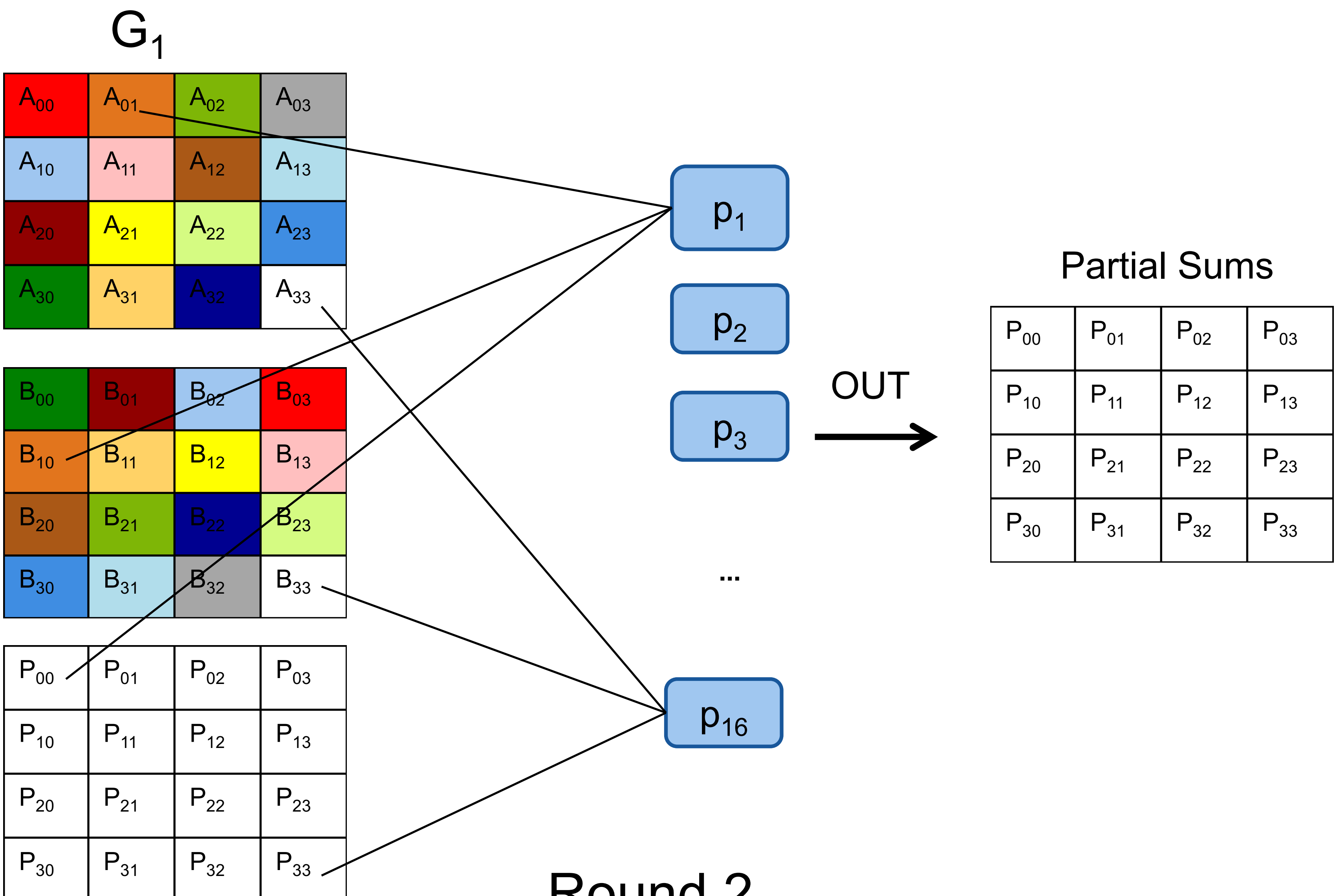
Each group has exactly 1 block mult for one C_{ij} block

Example 1: $p=H^2$ ($H=4$, $p=16$)



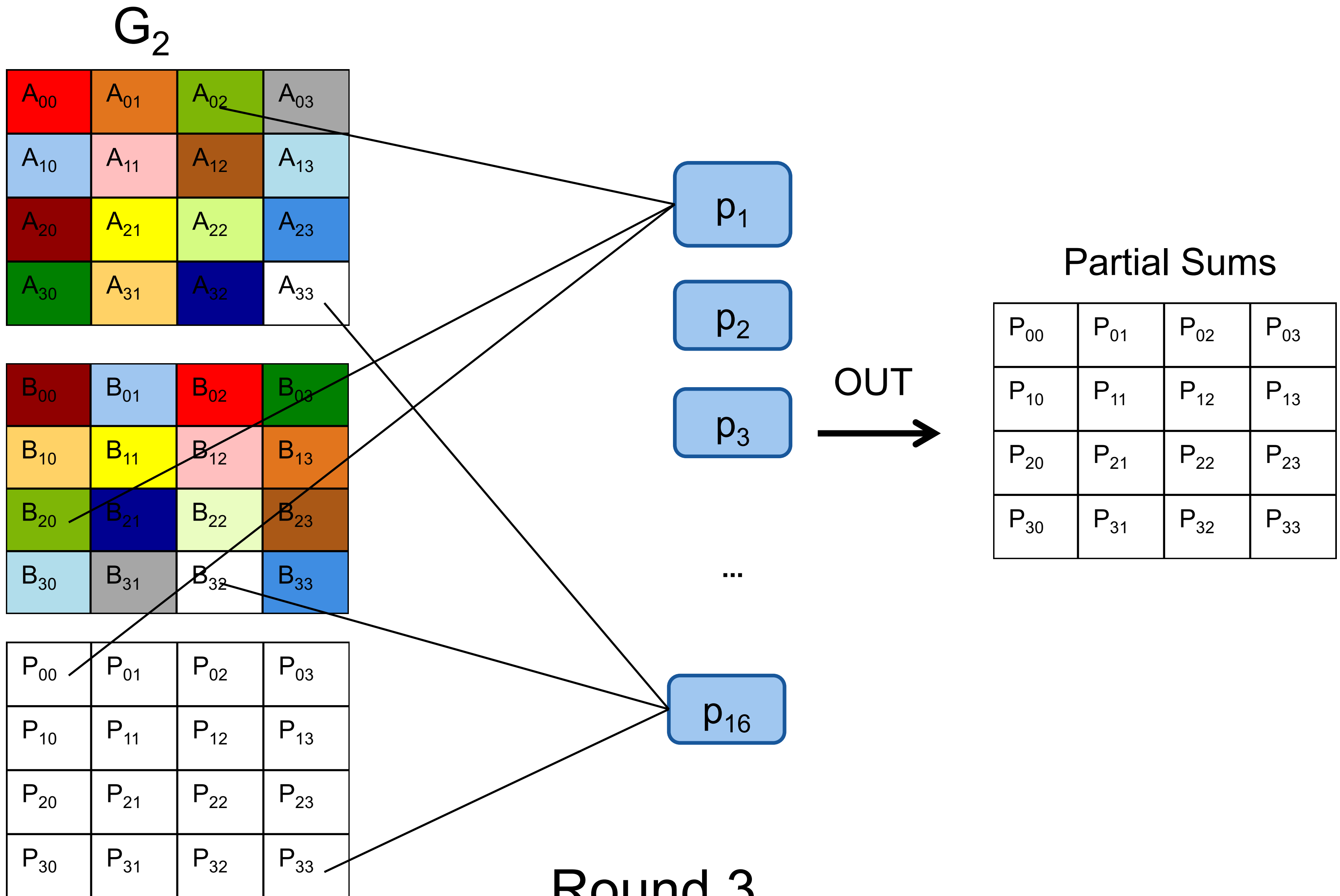
Round 1

Example 1: $p=H^2$ ($H=4$, $p=16$)



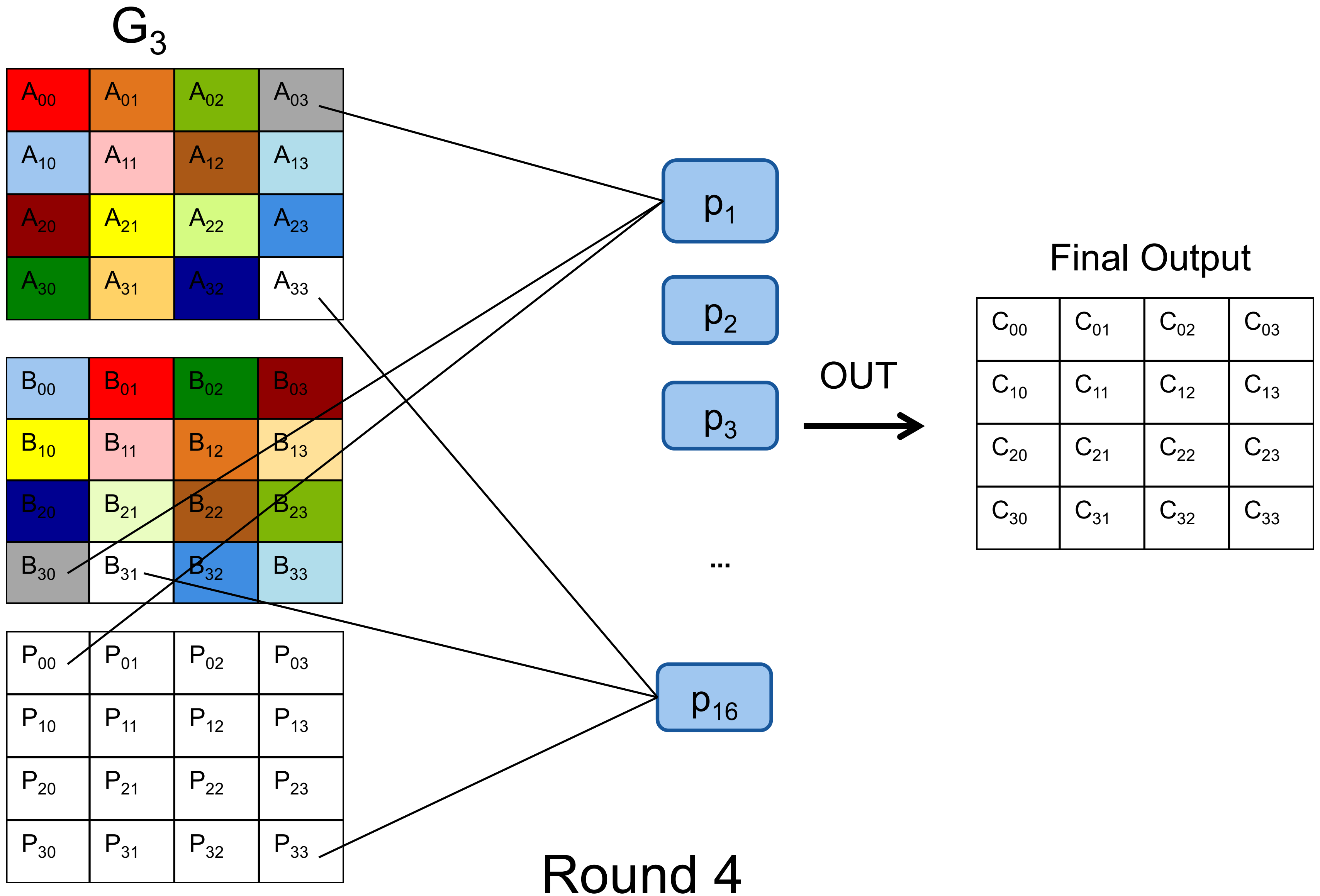
Round 2

Example 1: $p=H^2$ ($H=4$, $p=16$)



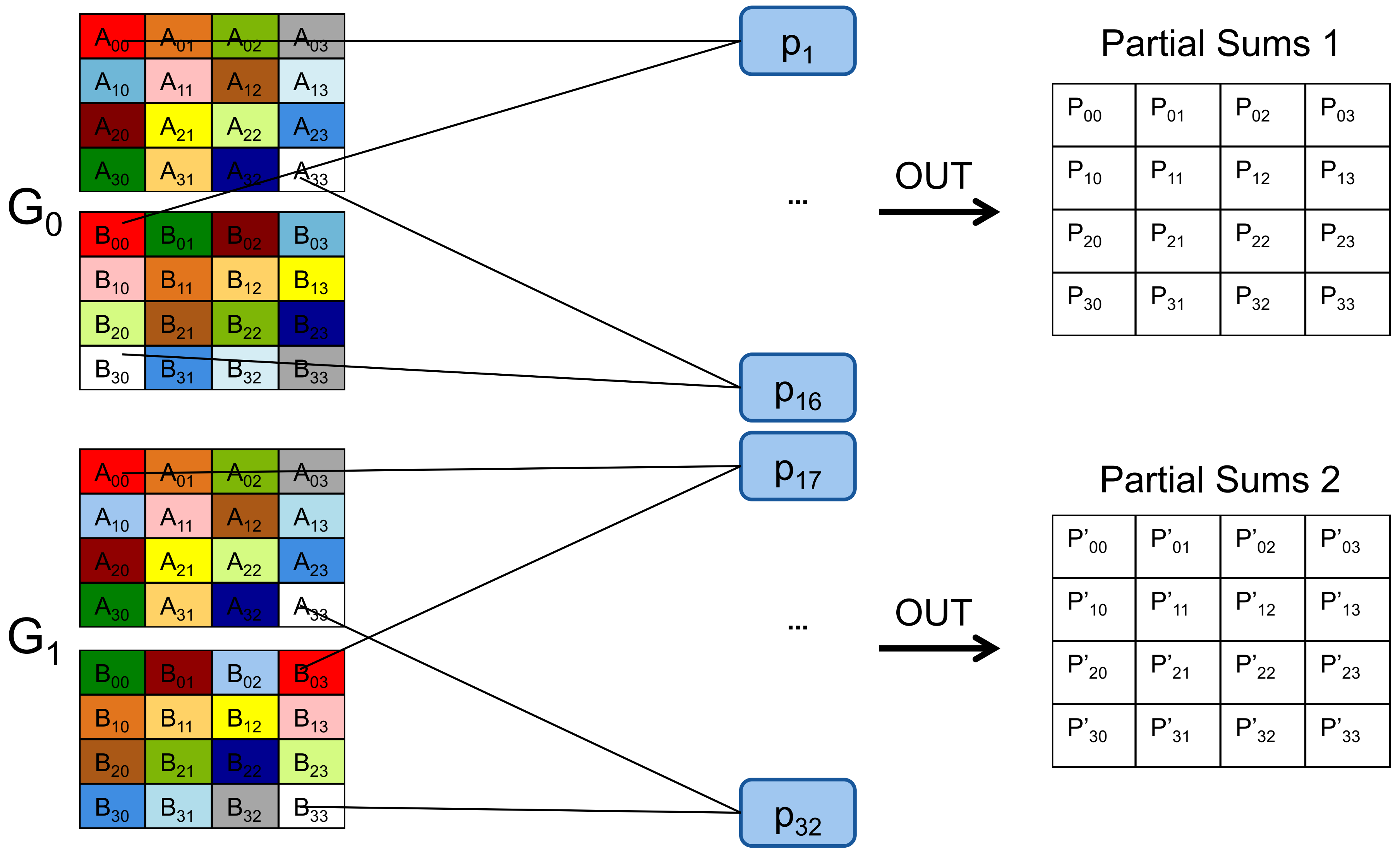
Round 3

Example 1: $p=H^2$ (H=4, p=16)



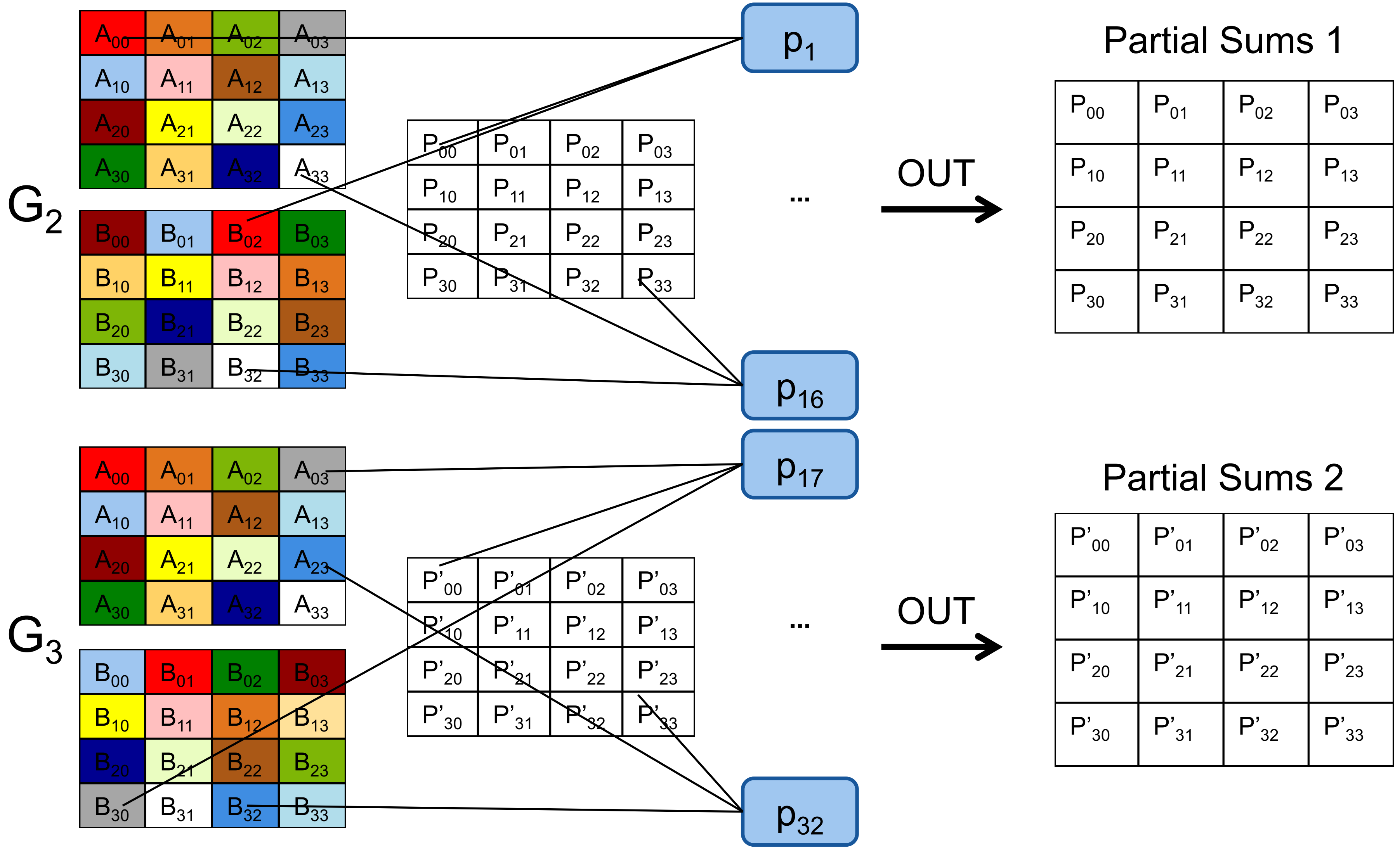
Round 4

Example 2: $p=2H^2$ ($H=4$, $p=32$)



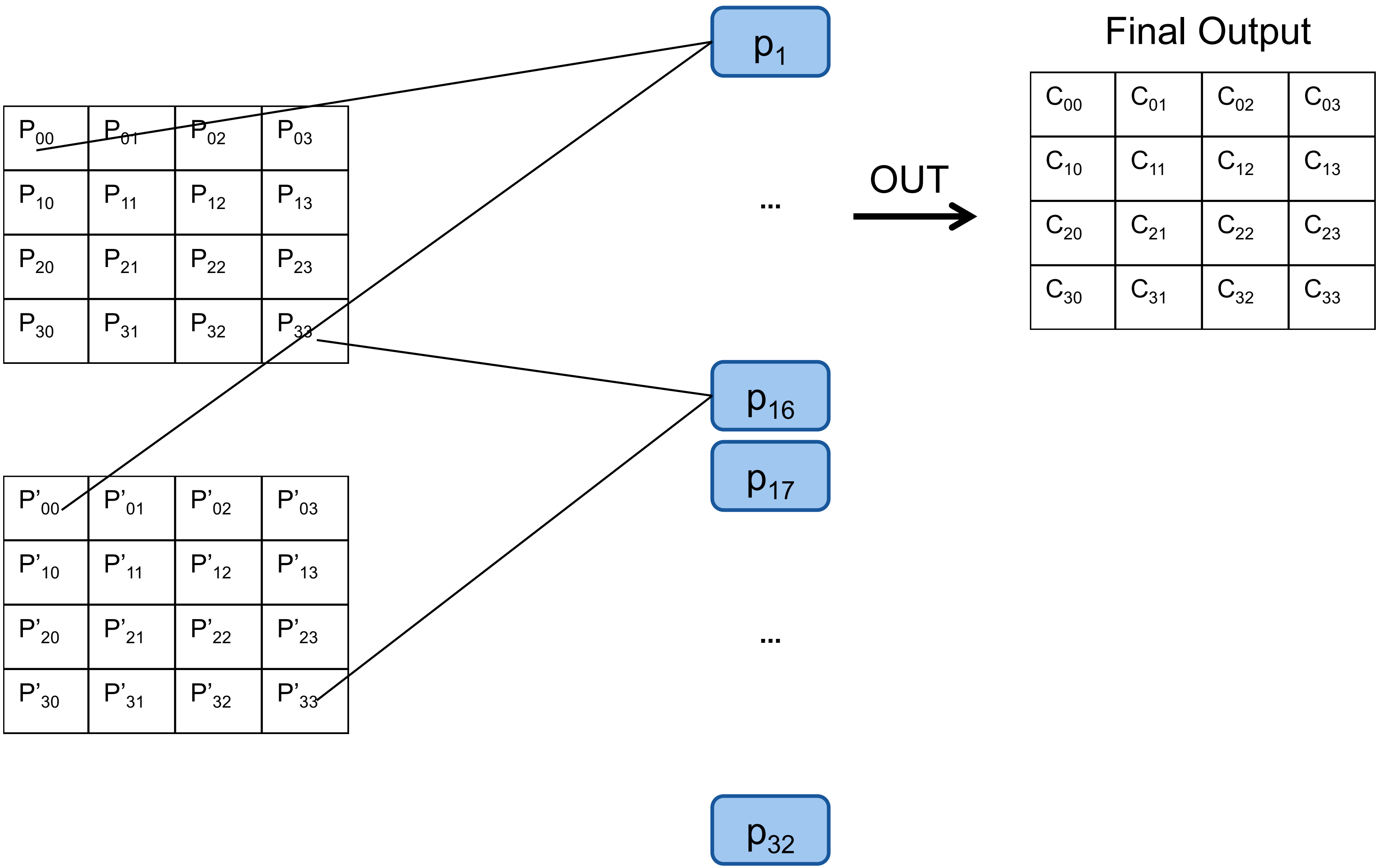
Round 1

Example 2: $p=2H^2$ ($H=4$, $p=32$)



Round 2

Example 2: $p=2H^2$ ($H=4$, $p=32$)



Round 3

Cost Analysis

	Communication	Rounds
Rectangle-Block	$O(n^4/L)$	1
Square-Block	$rpL=O(n^3/L^{1/2})$	$O(n^3/(pL^{3/2}) + \log_L n)$

Generalizes 2D and 3D algorithms.

Tight both for C and r for a given L!

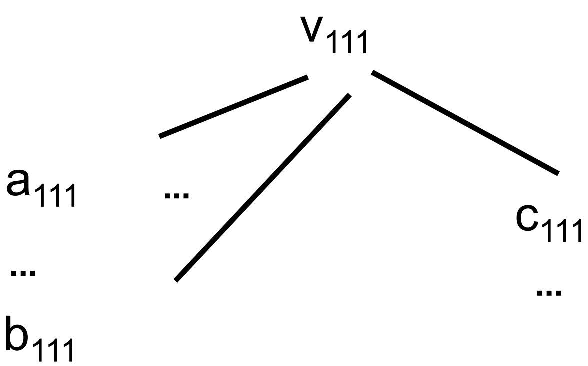
- Sub-linear Scalability: $pL > n^3/L^{1/2} \Rightarrow L > n^2/p^{2/3} \Rightarrow L > IN/p^{2/3}$

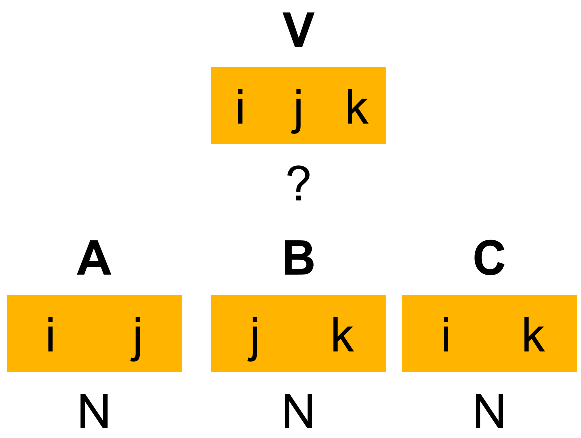
Next: B/c of connection to △ query and ρ^* of △ is 3/2

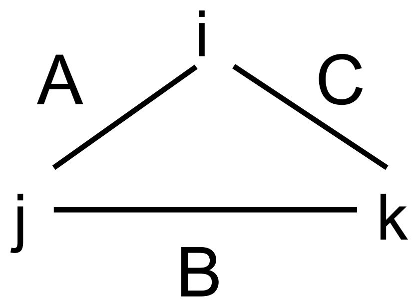
Round-Independent LB on C (and L) (1)

Suppose a proc. has N a_{ij}, N b_{jk} and contributes to N c_{ik} elements

Q: How many elementary multiplications can it perform?



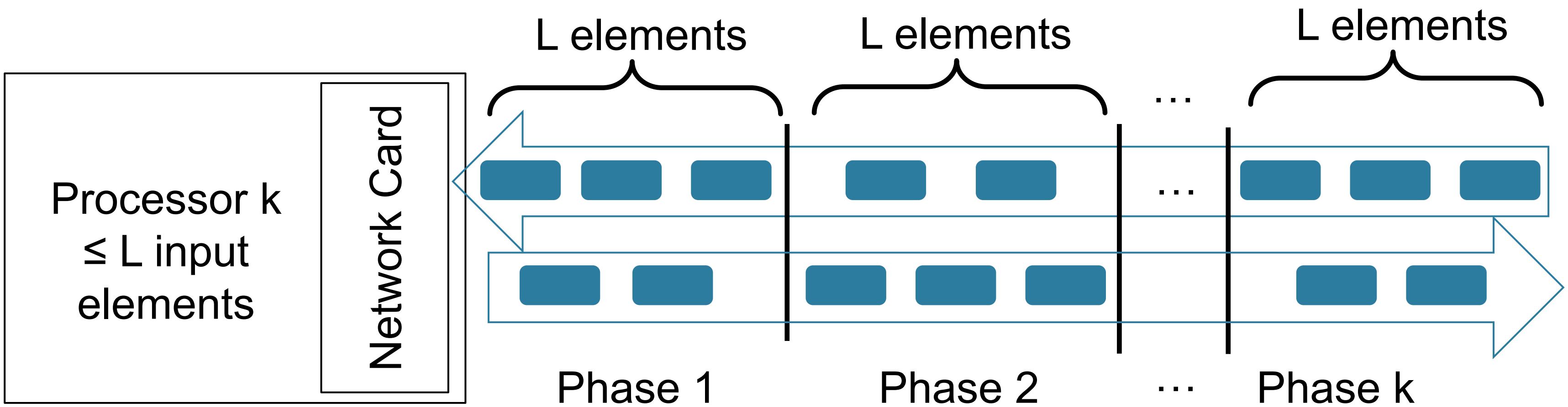




A: AGM => ρ^*= 3/2, so $O(N^{3/2})$

[Irony, Toledo, & Tiskin, JPDC, 2004], [Hong & Kung, STOC, 1981]

Round-Independent LB on C (and L) (2)



With L communication, a proc can do $O(L^{3/2})$ products.

n^3 products must be performed.

=> $C = \Omega(n^3/L^{1/2})$ (irrespective of r)

Lower Bounds on Rounds

- max(LB for Join, LB for GroupBy-and-Aggr)
- LB for Join:

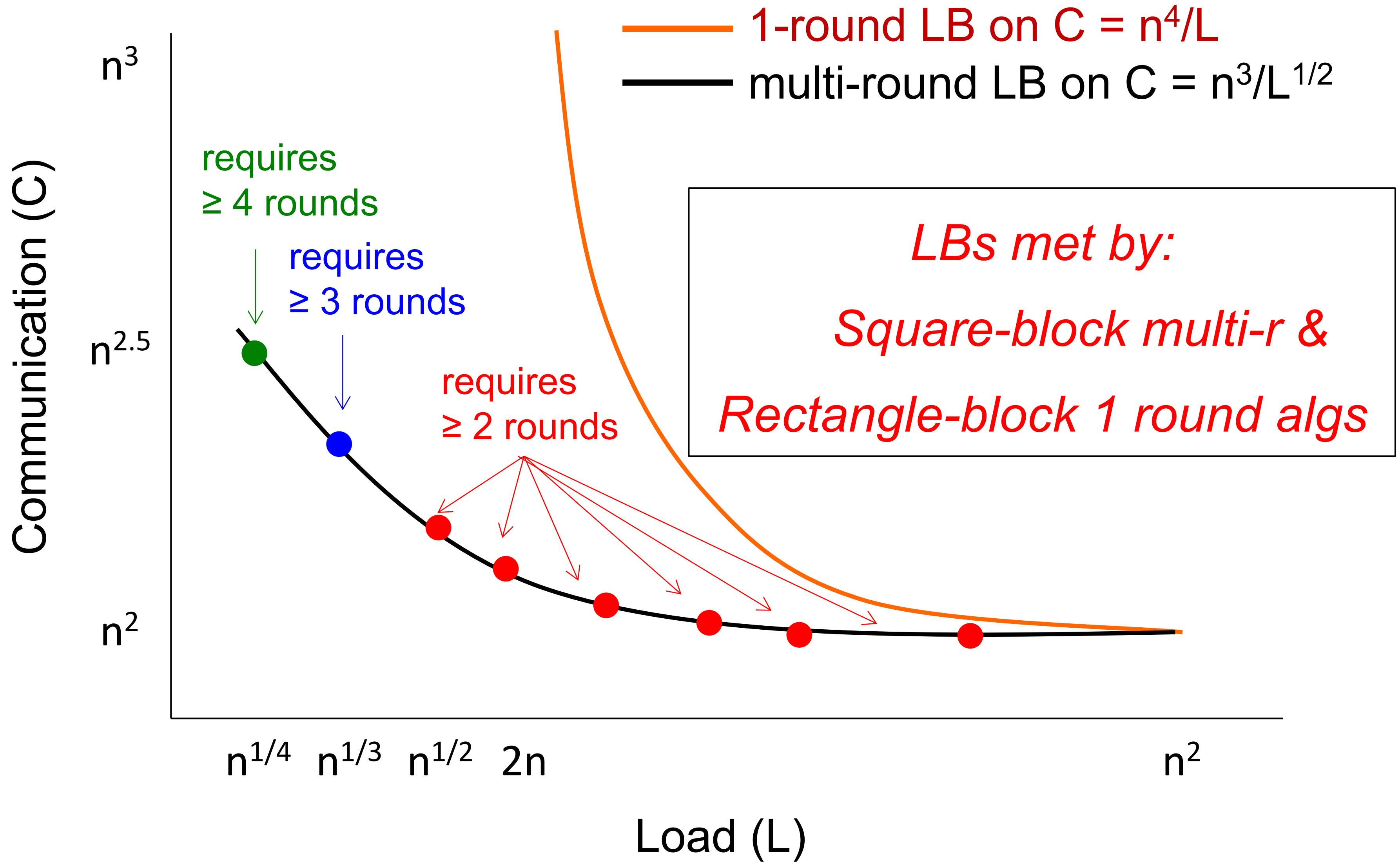
  $C = r \times p \times L > n^3/L^{1/2} \Rightarrow r = \Omega(n^3/(pL^{3/2}))$

- LB for GroupBy-and-Aggr: $r = \Omega(\log_L n)$

$$r = \Omega(\max(n^3/(pL^{3/2}), \log_L n))$$

[Irony, Toledo, & Tiskin, JPDC, 2004], [Goodrich, SICOMP, 1999]

Summary



1-round LB on C = n4/L
multi-round LB on C = n3/L1/2
n3
n2.5
n2
Communication (C)
requires
≥ 4 rounds
requires
≥ 3 rounds
requires
≥ 2 rounds
LBs met by:
Square-block multi-r &
Rectangle-block 1 round algs
n1/4 n1/3 n1/2 2n
n2
Load (L)

Other Results

- Non-Square MM
- Sparse square and non-square MM
- Strassen-like MM
- Cholesky, LU, QR decompositions, eigenvalue and singular value decompositions

Outline

- Models of parallel computation (Dan)
- Two-way joins (Paris)
- Multi-way joins (Paris+Semih)
- Sorting & Matrix multiplication (Paris+Semih)
- Conclusion (Dan)

Summary

Algorithms for:
- Joins: 2-way, multi-way
- Sorting
- Matrix multiplication

Goal: minimize total runtime
- Minimize communication cost
- Minimize number of rounds

Main Takeaways

Joins:

- Skew-free data
  - Optimal communication related to the fractional edge packing number **τ***
- Skewed data
  - Optimal communication related to the fractional edge covering number **ρ***

Total communication >> input data

Main Takeaways

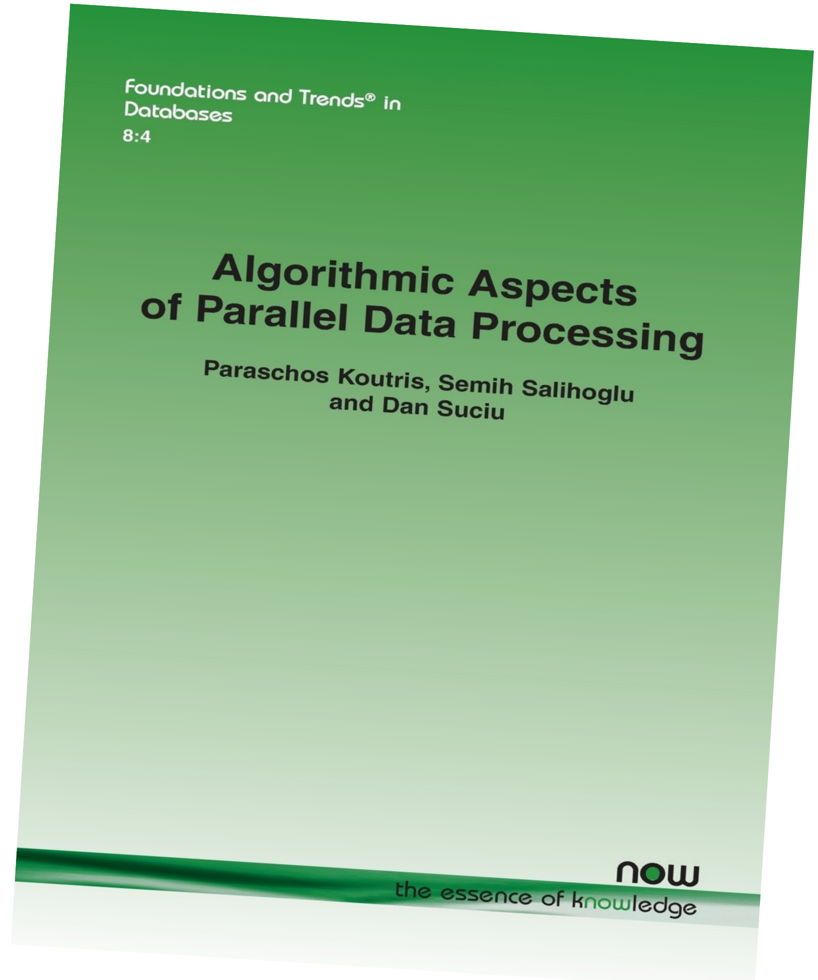
Sorting and matrix multiplication

- No skew
- Total communication = O(IN)
- When p << IN, algorithms are simple
- When p ≈ IN, algorithms are complex

Open Questions

- **Sorting-based** optimal multi-join algorithm
- Optimal communication cost f(IN,**OUT**)
- Lower bound for multi-rounds (**w/o skew**)

Questions?



https://tinyurl.com/y99w99b4

Free until June 18
(create account)