# CSEP 544: Lecture 09

## Advanced Query Processing
## Parallel Query Evaluation

# Outline

- Optimal Sequential Algorithms
- Semijoin Reduction
- Optimal Parallel Algorithms

# Semijoin: Review of Basics

$$R \ltimes_C S = \Pi_{A1,\ldots,An} (R \bowtie_C S)$$

Where $A_1, \ldots, A_n$ are the attributes in R

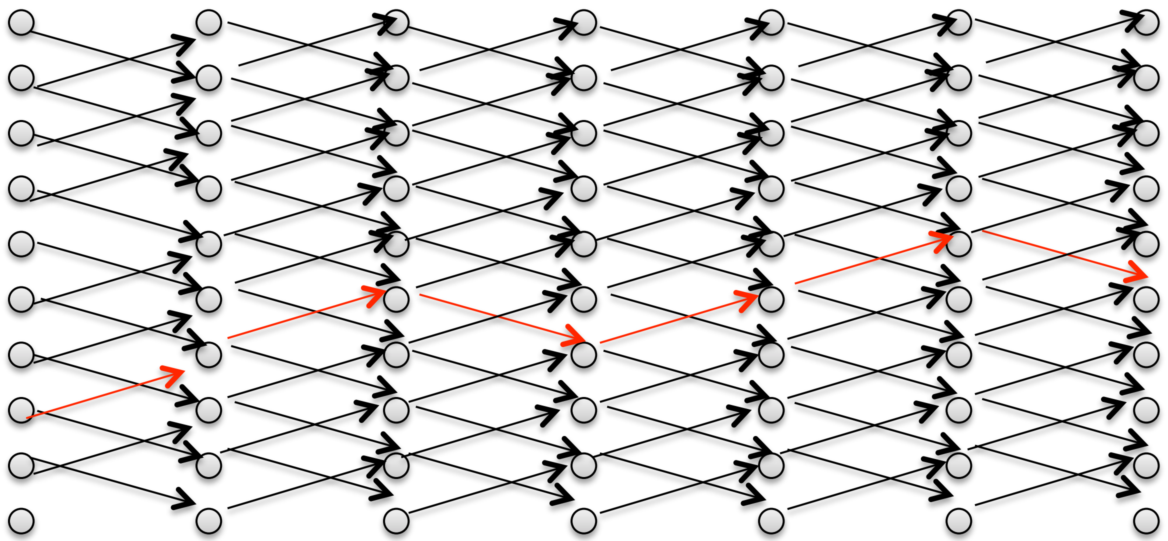
Formally, R ⋉C S means this: retain from R only those tuples that have some matching tuple in S

Duplicates in R are preserved

Duplicates in S don't matter

# Application

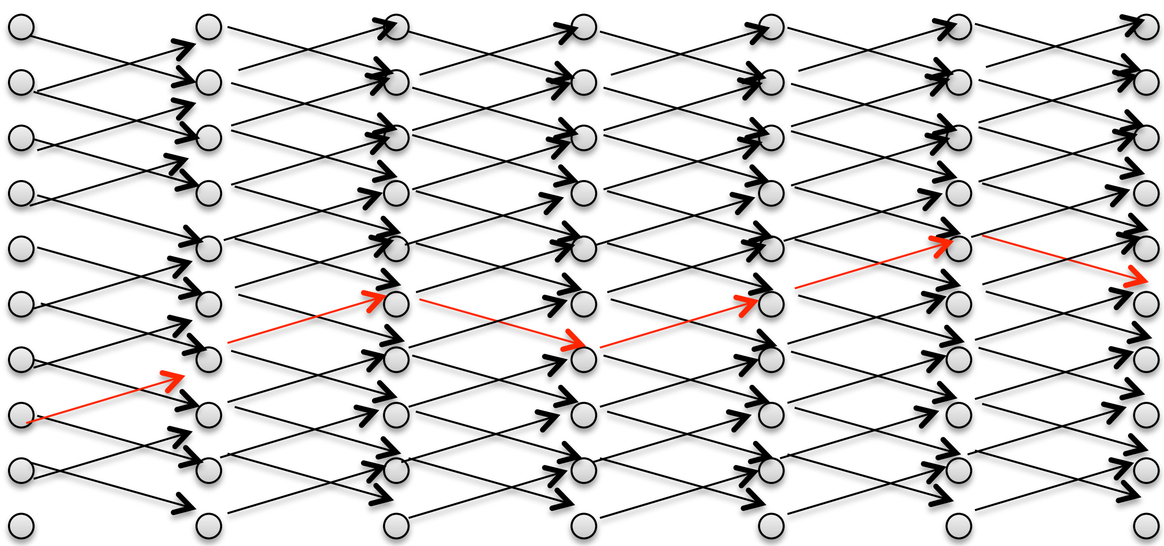
$Q(x,y,z,u,v,w) = R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)$



Intermediate relations of size up to $m^5$…
... But final answer can be as small as 0  (or 1...)

# Application

$Q(x,y,z,u,v,w) = R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)$



Solution: semi-join reduction

/* Forwards: */

$S := S \ltimes R$

$T := T \ltimes S$

$K := K \ltimes T$

$L := L \ltimes K$

/* then backwards: */

$K := K \ltimes L$

$T := T \ltimes K$

$S := S \ltimes T$

$R := R \ltimes S$

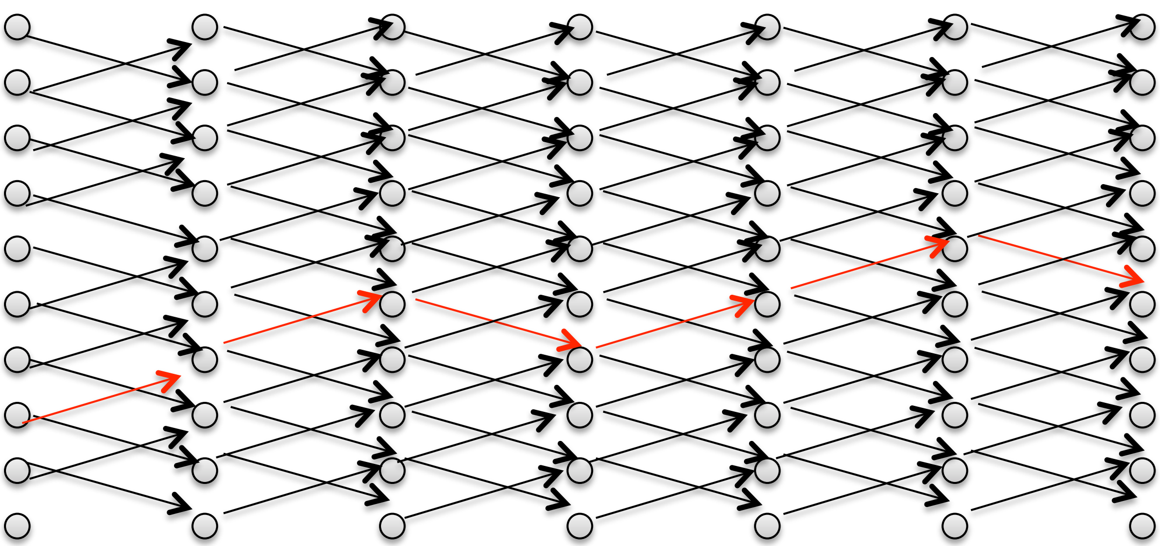
Intermediate relations of size up to $m^5$…

... But final answer can be as small as 0 (or 1...)

# Application

$Q(x,y,z,u,v,w) = R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)$

Solution: semi-join reduction



/* Forwards: */

$S := S \ltimes R$

$T := T \ltimes S$

$K := K \ltimes T$

$L := L \ltimes K$

/* then backwards: */

$K := K \ltimes L$

$T := T \ltimes K$

$S := S \ltimes T$

$R := R \ltimes S$

Intermediate relations of size up to $m^5$…

... But final answer can be as small as 0 (or 1...)

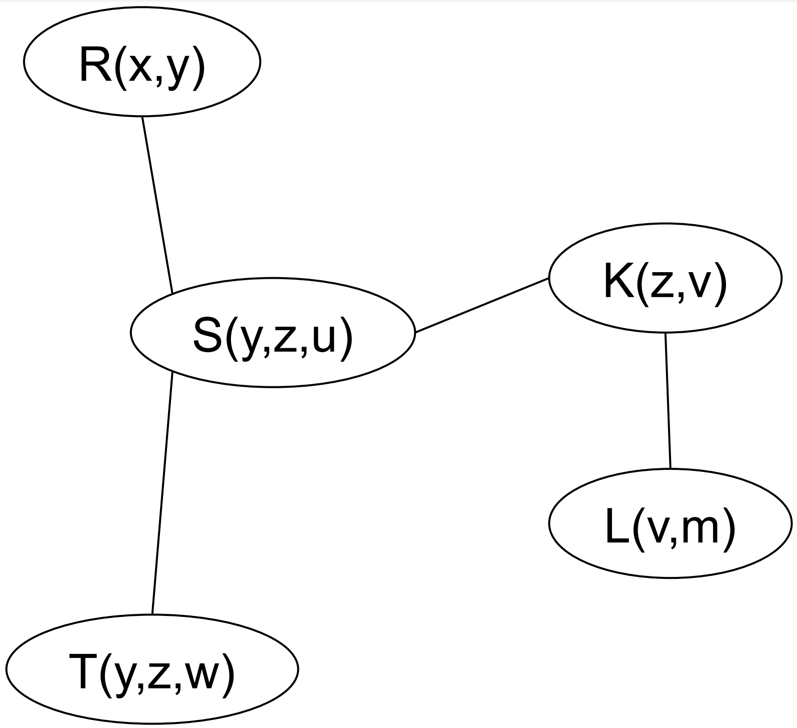
Next, compute the query in time O(|Input| + |Output|)

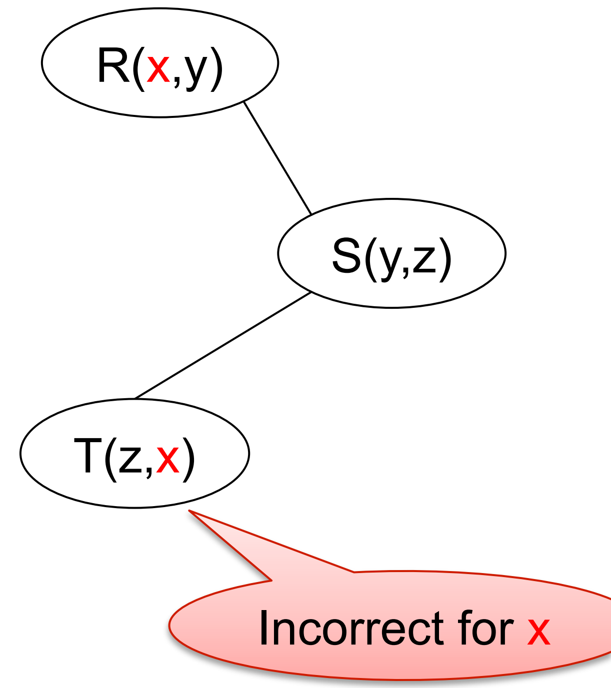
# Acyclic Queries

- A query is *acyclic* if its relations can be placed in a tree such that the set of nodes that contain any variable form a connected set

- Yannakakis' algorithm (from the 80's): any acyclic query can be computed in time: O(|Input| + |Output|)

Acyclic query:
Q(x,y,z,u,v,w,m) = R(x,y),S(y,z,u),T(y,zw),K(z,v),L(v,m)




O(n + |Output|)

Cyclic query:
Q(x,y,z) = R(x,y),S(y,z),T(z,x)

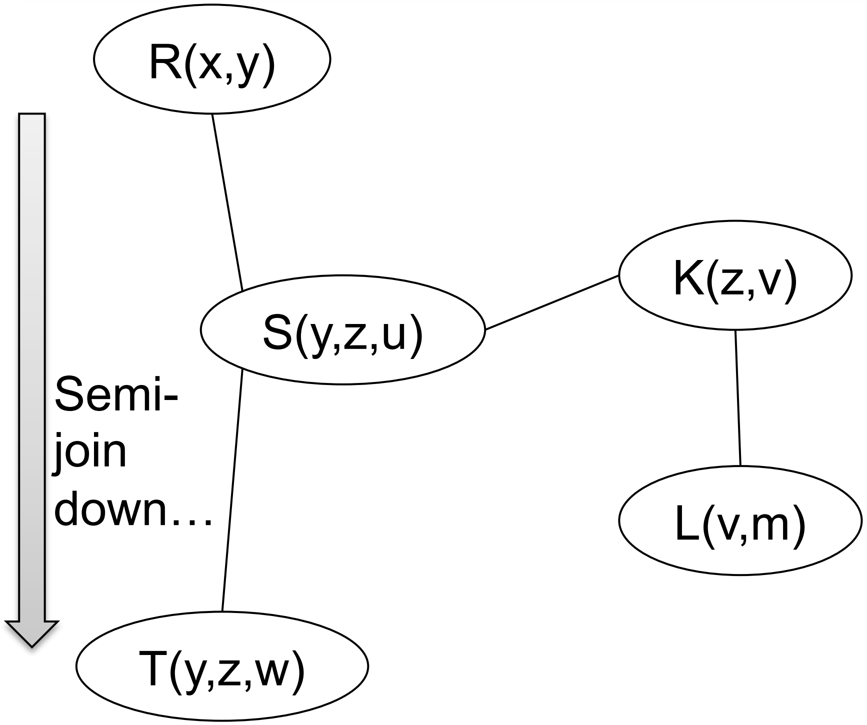
# Yannakakis' Algorithm

- Step 1: semi-join reduction
  - Two sweeps: up and down

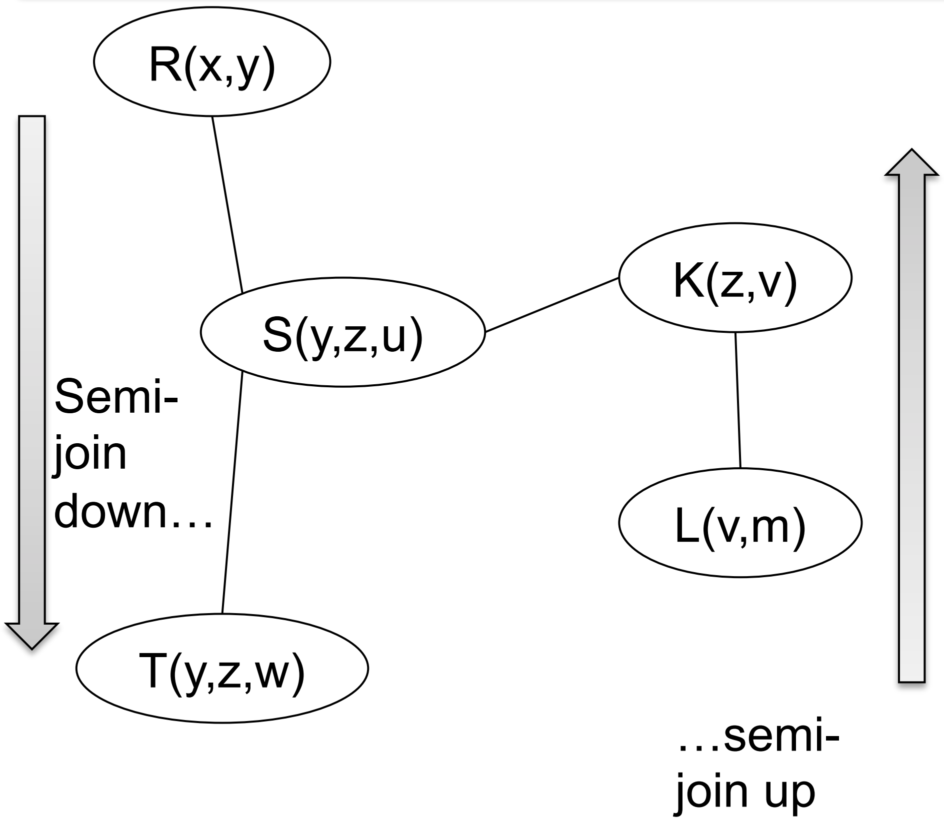
- Step 2: use the tree as query query plan, compute bottom up

- Note: this works also for queries with projections and/or aggregates (in class…)

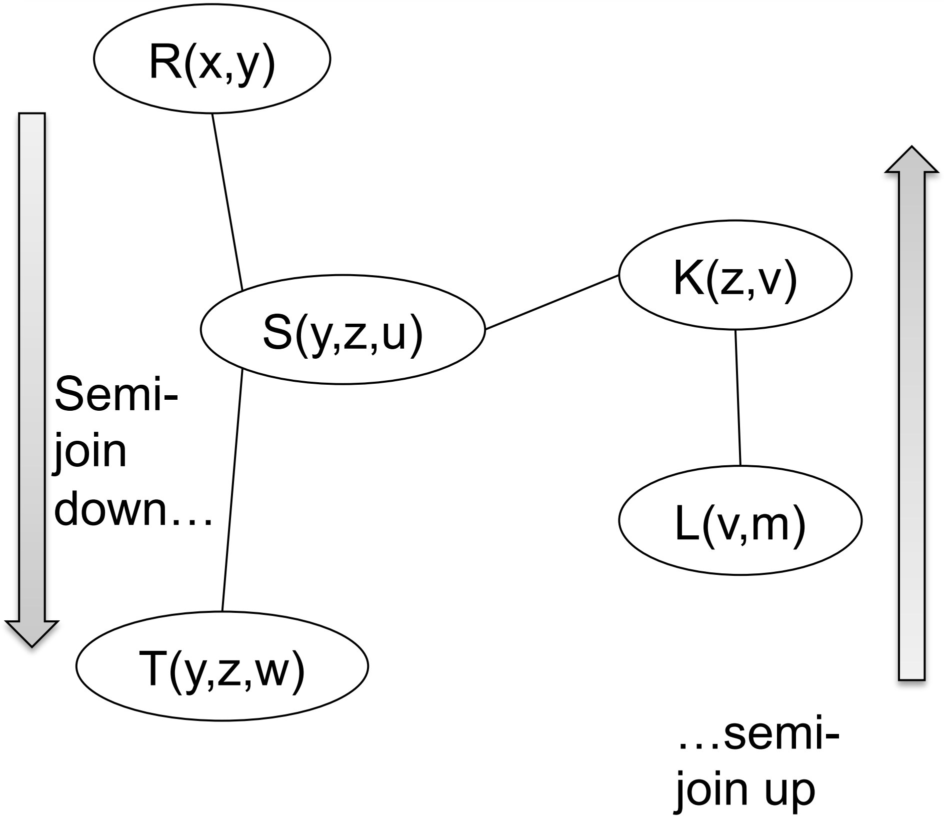
Acyclic query:
Q(x,y,z,u,v,w,m) = R(x,y),S(y,z,u),T(y,zw),K(z,v),L(v,m)



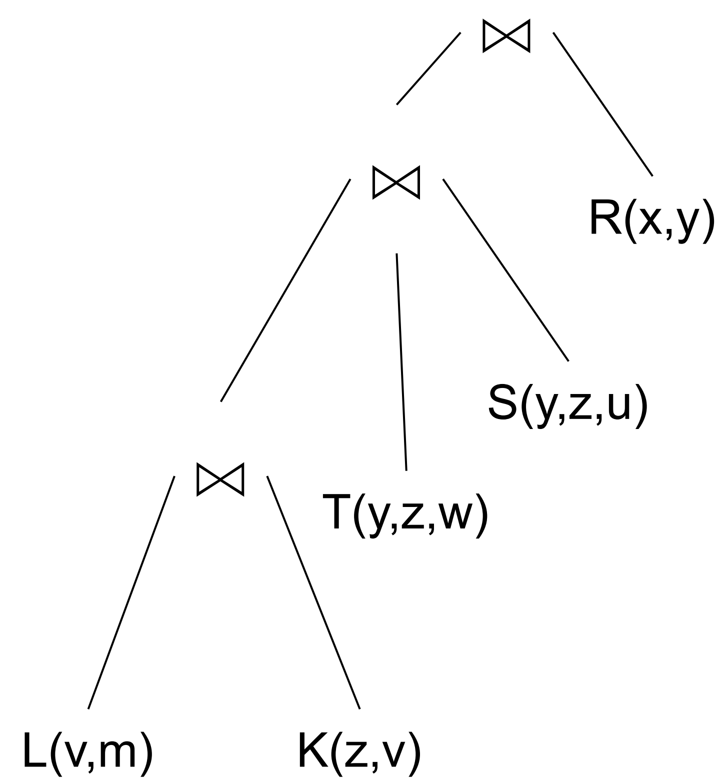
Acyclic query:
Q(x,y,z,u,v,w,m) = R(x,y),S(y,z,u),T(y,zw),K(z,v),L(v,m)



Acyclic query:
$Q(x,y,z,u,v,w,m) = R(x,y),S(y,z,u),T(y,zw),K(z,v),L(v,m)$

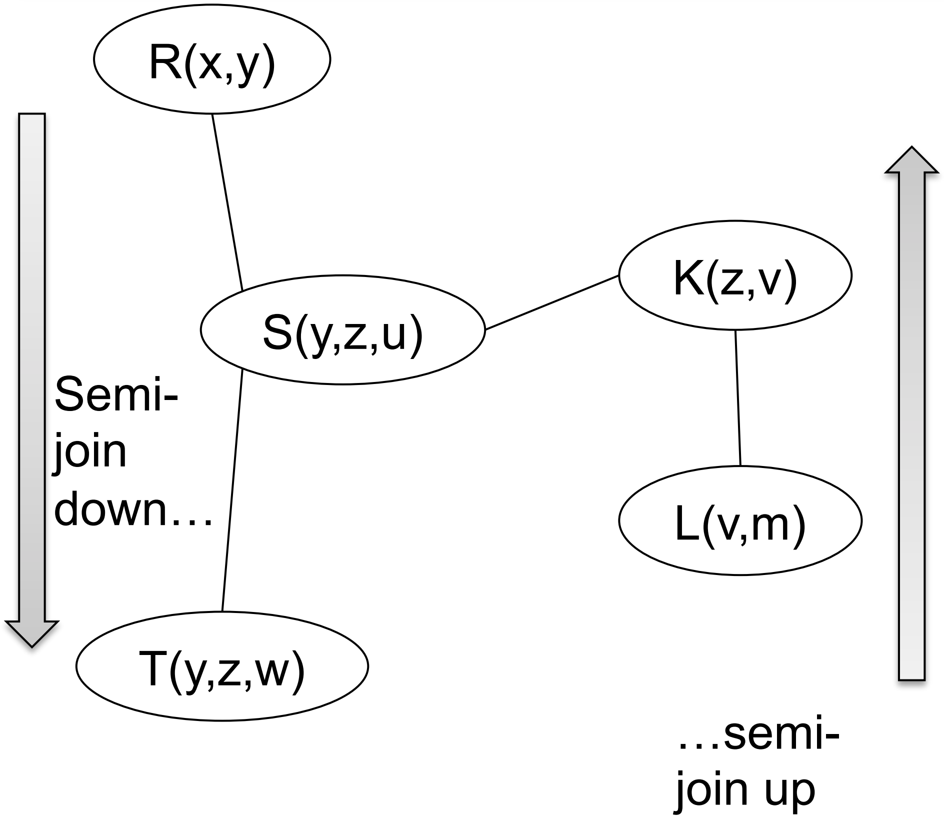



Query plan

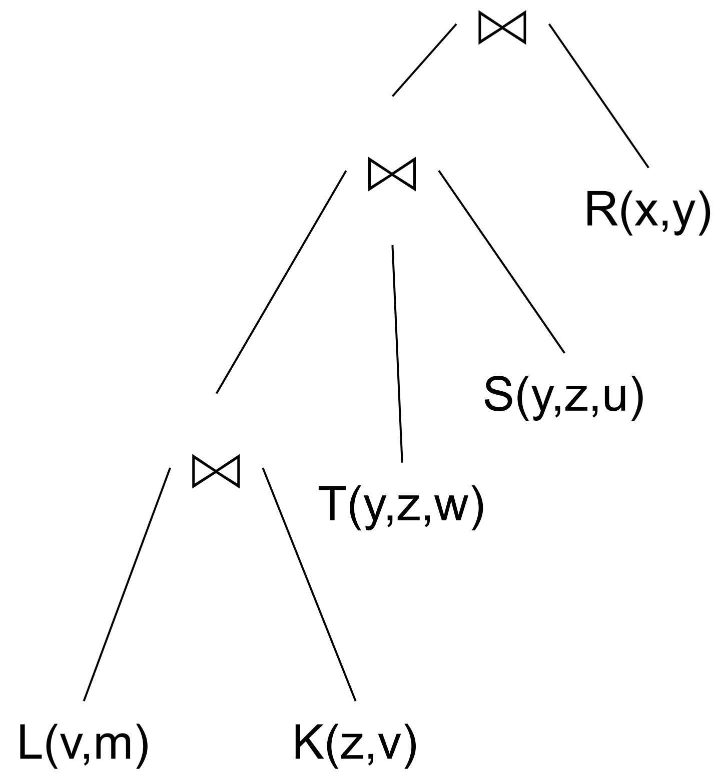



O(n + |Output|)

Acyclic query:
Q(x,y,z,u,v,w,m) = R(x,y),S(y,z,u),T(y,zw),K(z,v),L(v,m)




Query plan




O(n + |Output|)

Question: change the query plan to return only variables x,w,m
Q(x,w,m) = R(x,y),S(y,z,u),T(y,zw),K(z,v),L(v,m)

# Cyclic Queries:

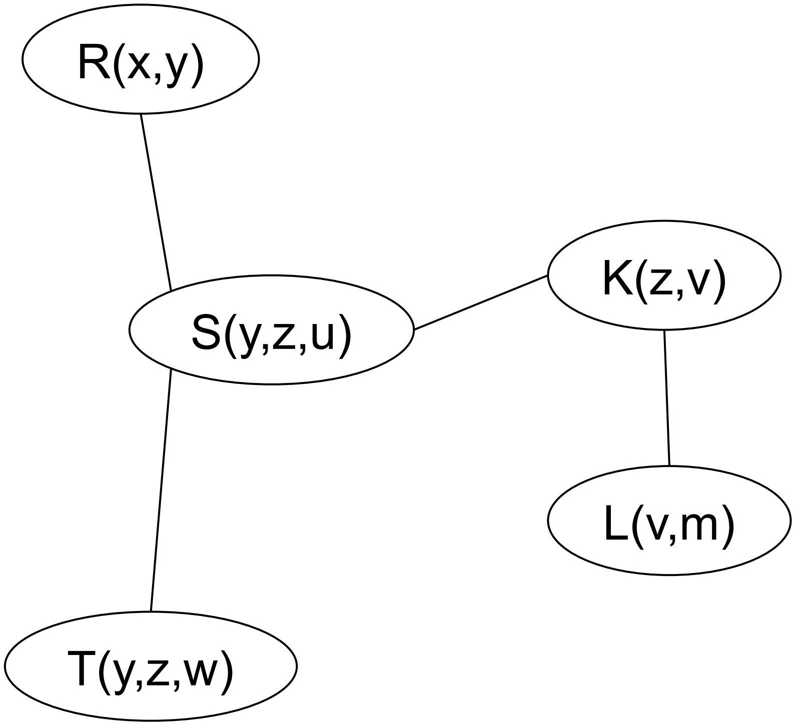
1. Acyclic queries: computable in time $O( |input| + |output| )$ [Yannakakis'81]

2. Arbitrary queries: computable in time $O( |input|^{\text{fractional-tree-width}} + |output| )$

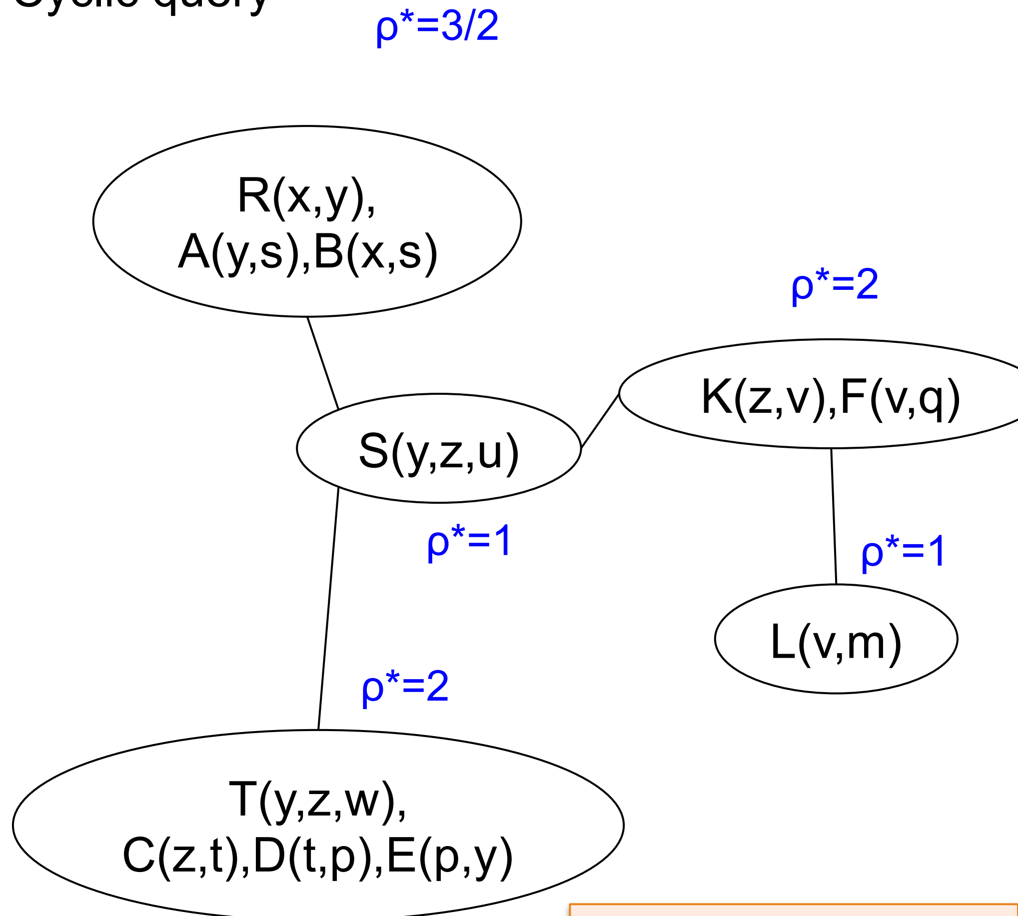
The fractional tree-width:

- Tree nodes may contain multiple relations
- FTW is the largest AGM bound of any node

Acyclic query




Cyclic query




$O(n + |\text{Output}|)$

ftw = max(ρ*) = 2

In class:
how do we
compute this query

Optimal edge cover = ρ*
AGM bound = $m^{\rho^*}$
Generic-join algorithm, time = $O(m^{\rho^*})$

$O(|\text{Input}|^2 + |\text{Output}|)$

# Outline

- Optimal Sequential Algorithms
- Semijoin Reduction
- Optimal Parallel Algorithms

# Parallel Models

- Shared-Nothing: p servers
- MapReduce: p reducers

- Notice: in MR we can choose p, but, as we shall see, it's better if p is the number of physical servers

# Overview

Computes a full conjunctive query in one round, by partial replication.

- Replication appears in [Ganguli'92]
- Shares Algorithm: [Afrati&Ullman'10]
  - For MapReduce
- HyperCube Algorithm [Beame'13,'14]
  - Same as in Shares
  - But different optimization/analysis

# The Triangle Query

- Input: three tables
  R(X, Y), S(Y, Z), T(Z, X)

  |R| = |S| = |T| = m tuples

- Output: compute all triangles
  Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)

**T**

| Z | X |
|---|---|
| Fred | Alice |
| | Jim |
| | Jim |
| | Alice |

**S**

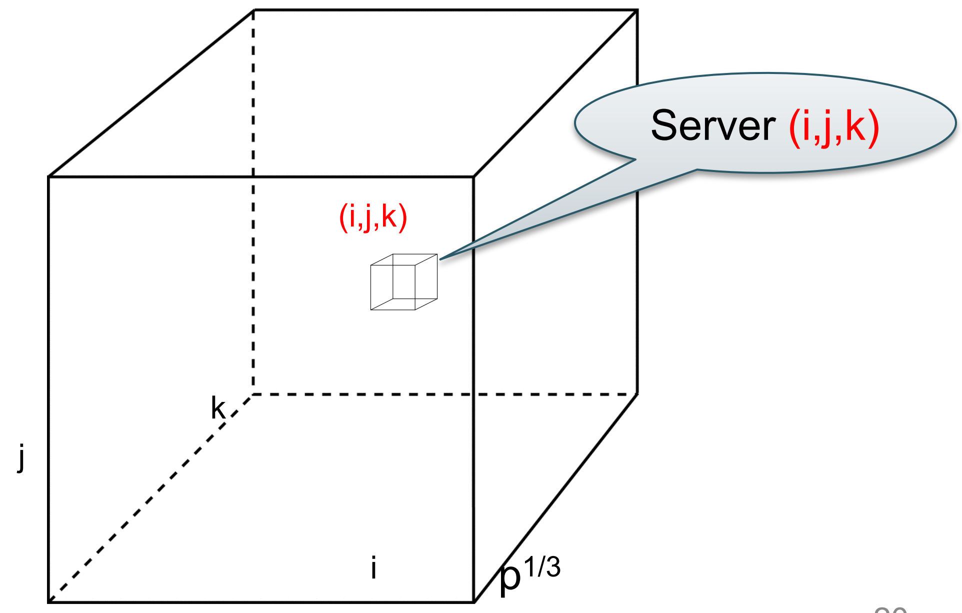
| Y | Z |
|---|---|
| Fred | Alice |
| | Jim |
| | Jim |
| | Alice |

**R**

| X | Y |
|---|---|
| Fred | Alice |
| Jack | Jim |
| Fred | Jim |
| Carol | Alice |
| … | |

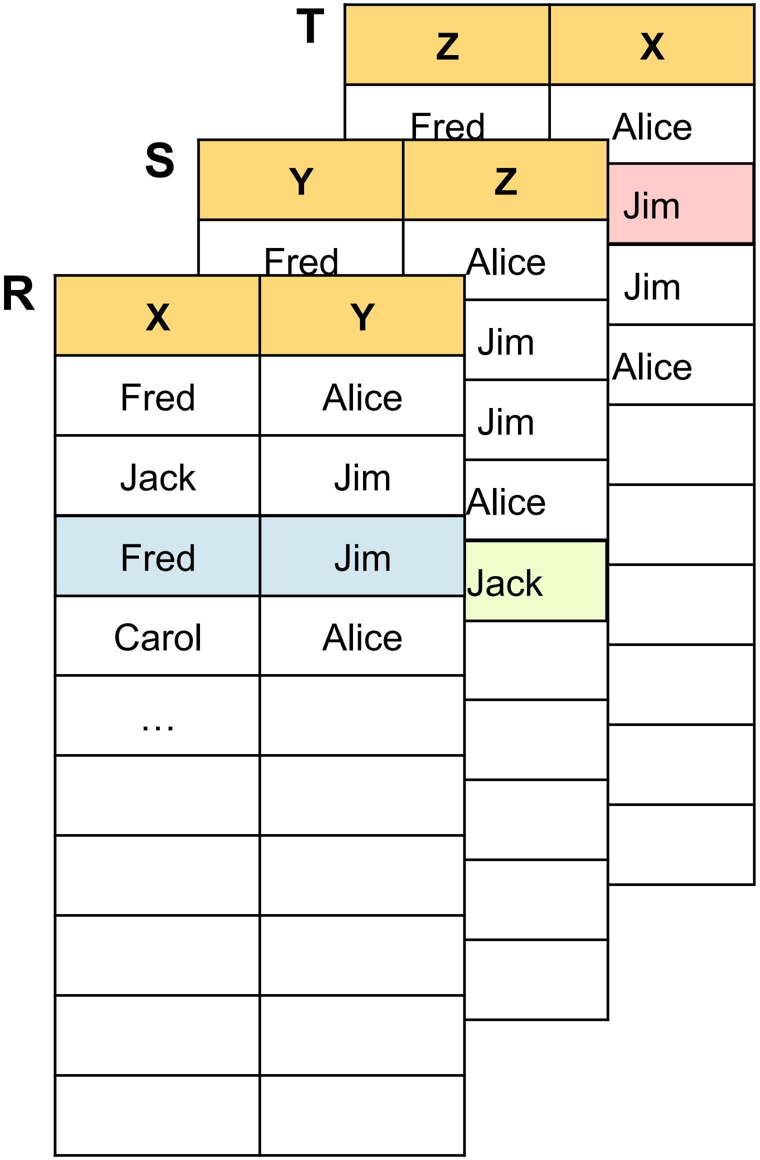
# Triangles in One Round

- Place servers in a cube $p = p^{1/3} \times p^{1/3} \times p^{1/3}$
- Each server identified by (i,j,k)



Triangles(x,y,z) = R(x,y), S(y,z), T(z,x)    |R| = |S| = |T| = m tuples

# Triangles in One Round

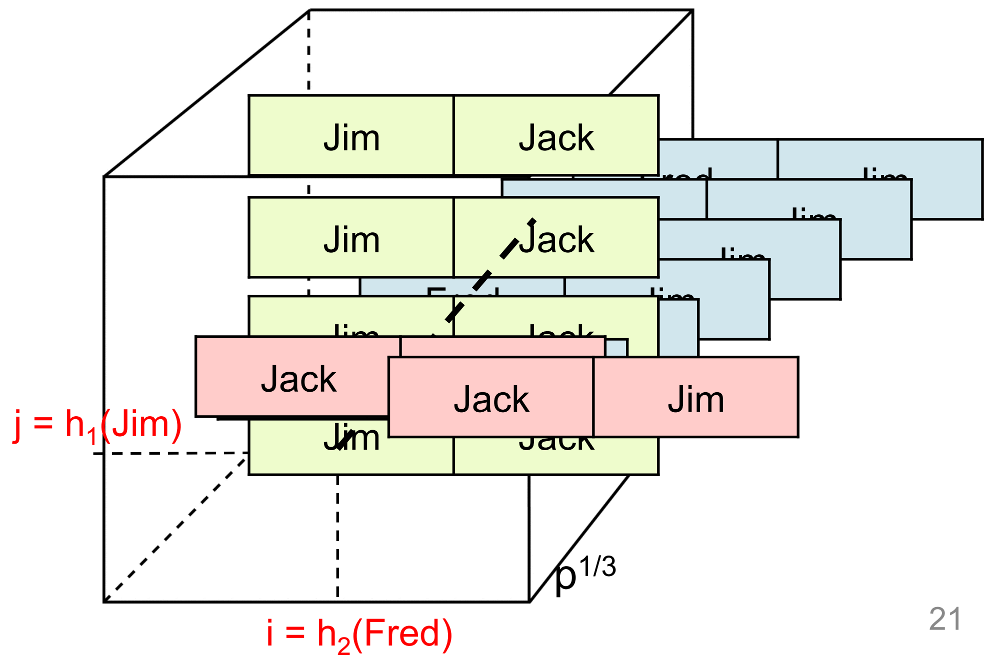



**Round 1**:

Send R(x,y) to all servers ($h_1(x)$,$h_2(y)$,*)
Send S(y,z) to all servers (*, $h_2(y)$, $h_3(z)$)
Send T(z,x) to all servers ($h_1(x)$, *, $h_3(z)$)

**Output**:

compute locally $R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

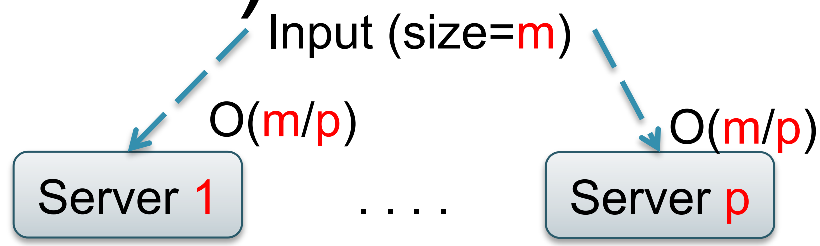


# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size m

Number of servers = p



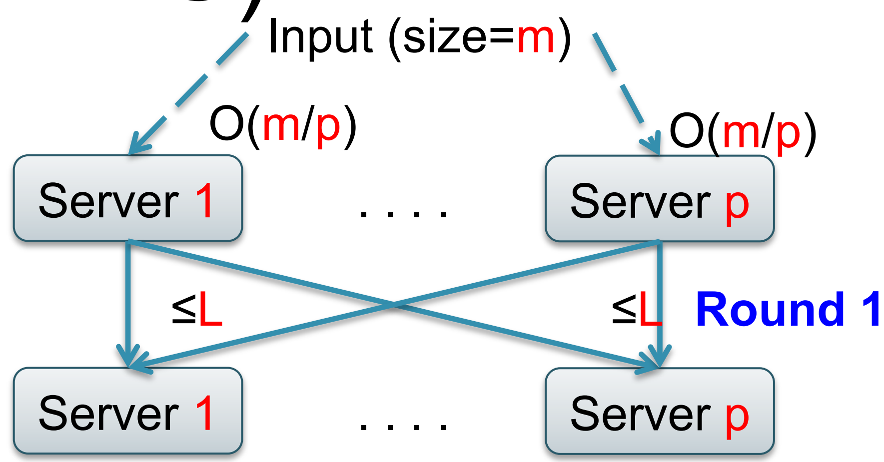
# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size m

Number of servers = p

One round = Compute & communicate



# Massively Parallel Communication Model (MPC)

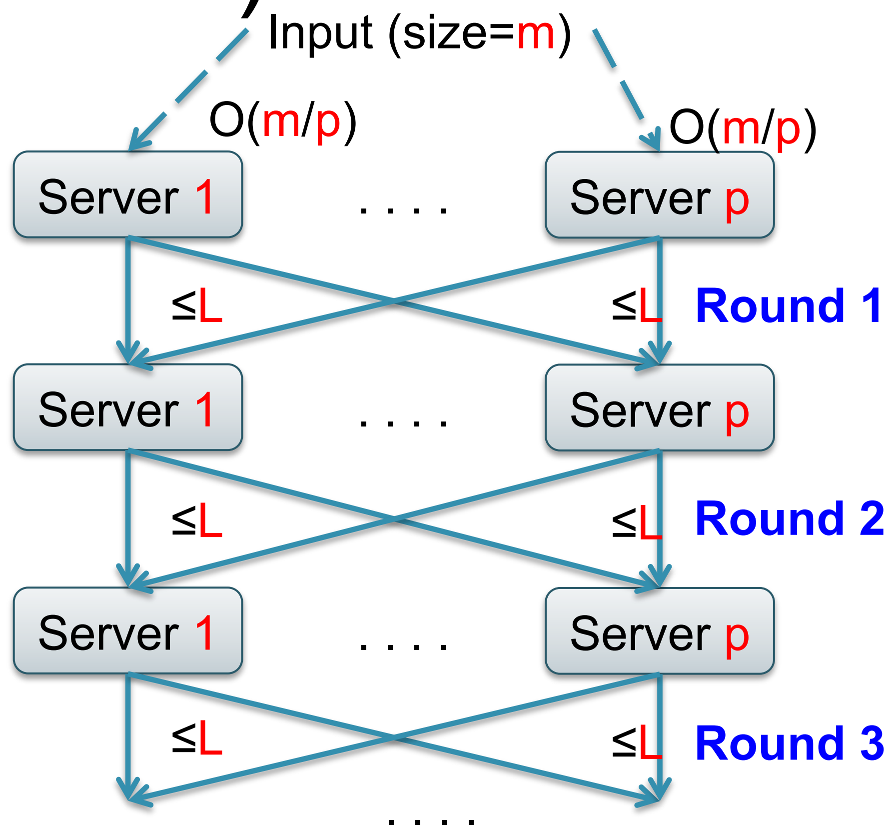
Extends BSP [Valiant]

Input data = size m

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds



# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

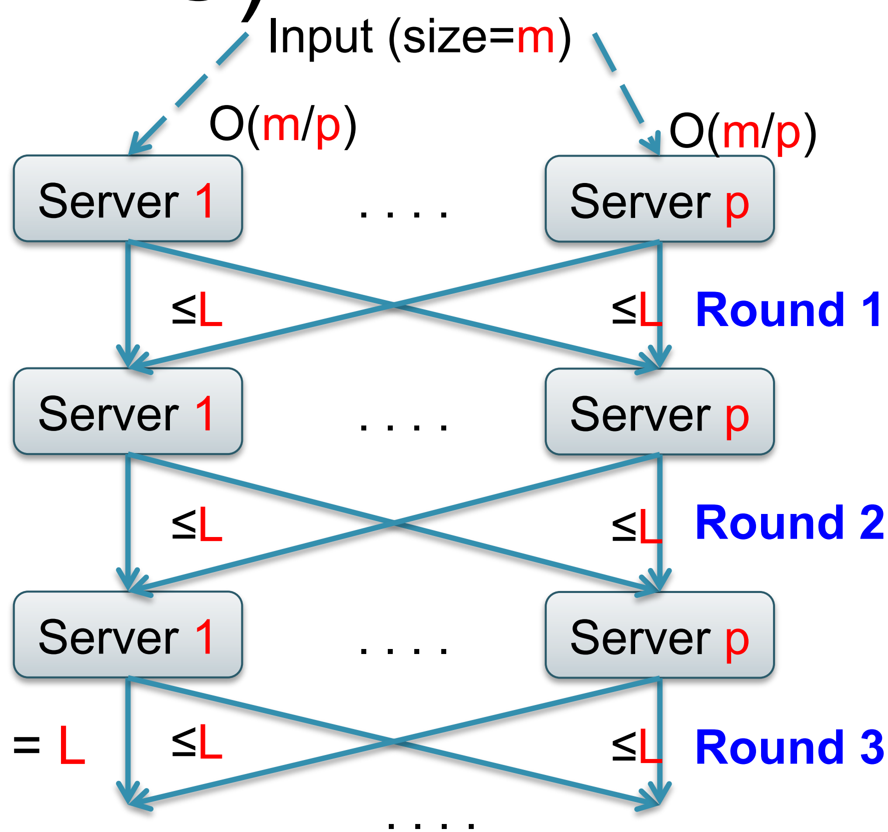
Input data = size m

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L



# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

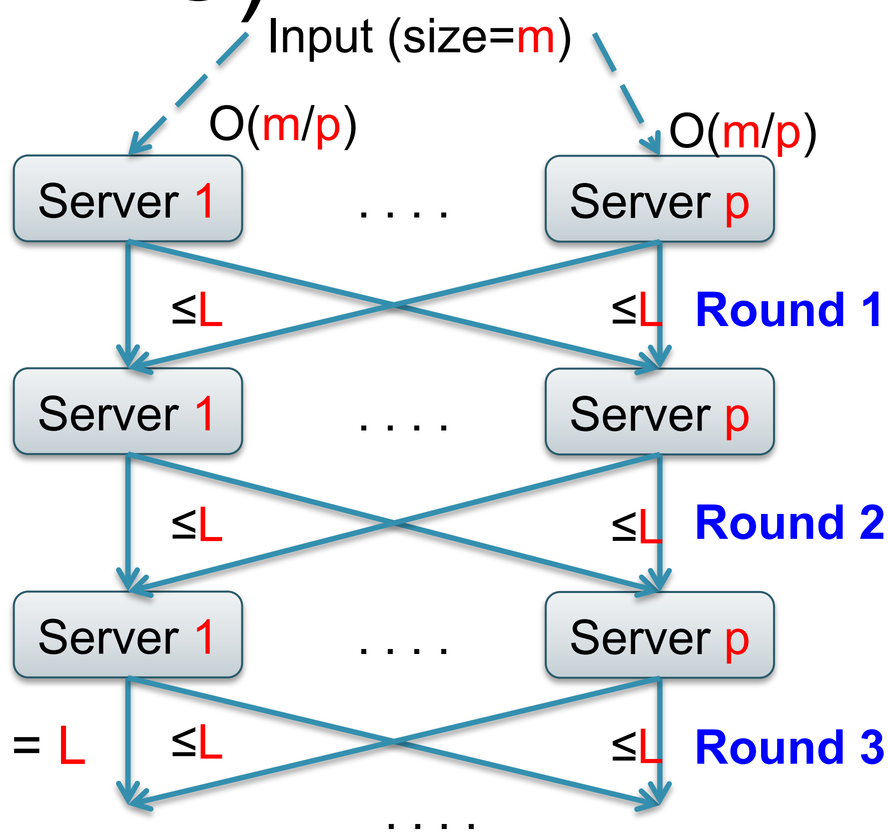
Input data = size m

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)

Extends BSP [Valiant]

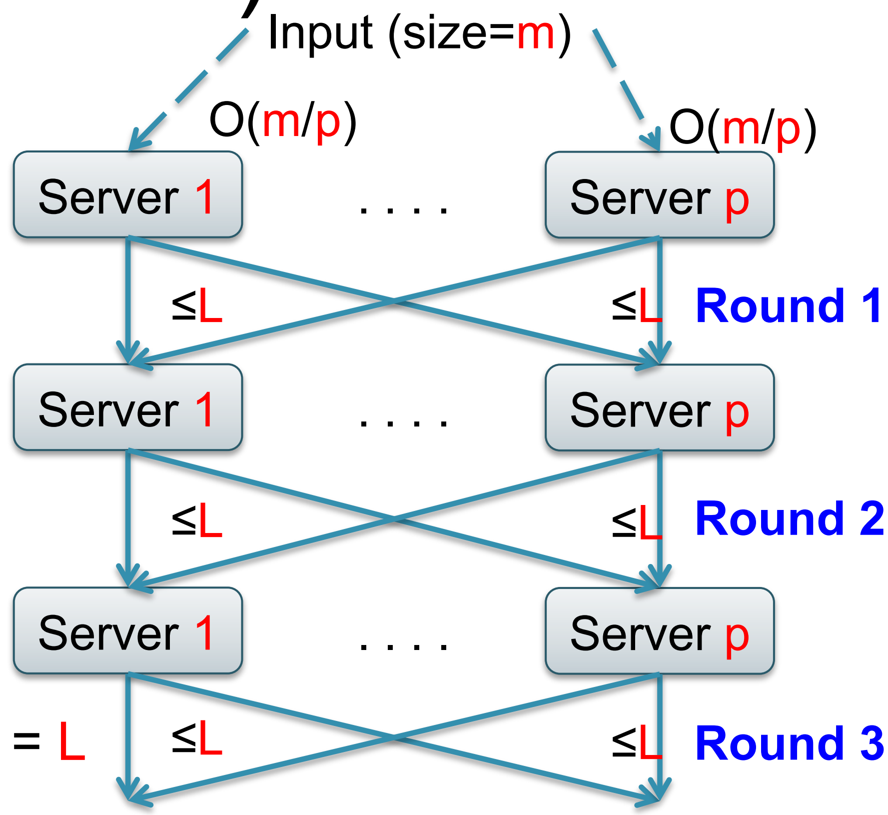
Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$




| **Cost:** | | | Naïve 1 | |
|---|---|---|---|---|
| Load $L$ | | | $L = m$ | |
| Rounds $r$ | | | 1 | |

# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

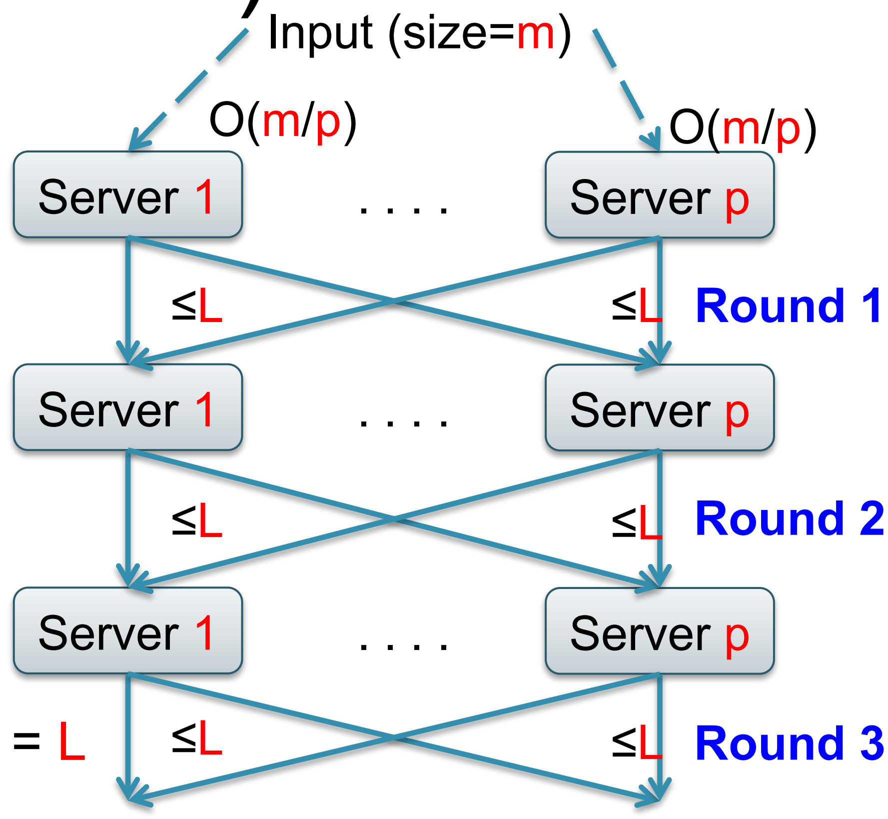
Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$




| Cost: | | | Naïve 1 | Naïve 2 |
|---|---|---|---|---|
| Load $L$ | | | $L = m$ | $L = m/p$ |
| Rounds $r$ | | | 1 | $p$ |

# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

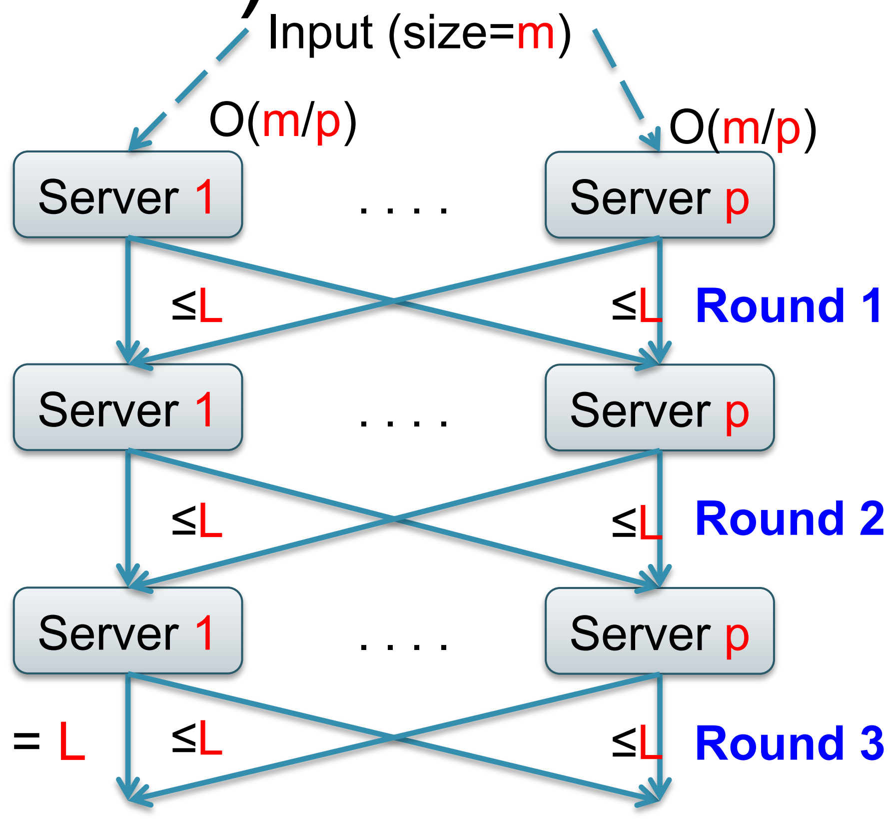
Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$




| **Cost:** | Ideal | | Naïve 1 | Naïve 2 |
|---|---|---|---|---|
| Load $L$ | $L = m/p$ | | $L = m$ | $L = m/p$ |
| Rounds $r$ | 1 | | 1 | $p$ |

# Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

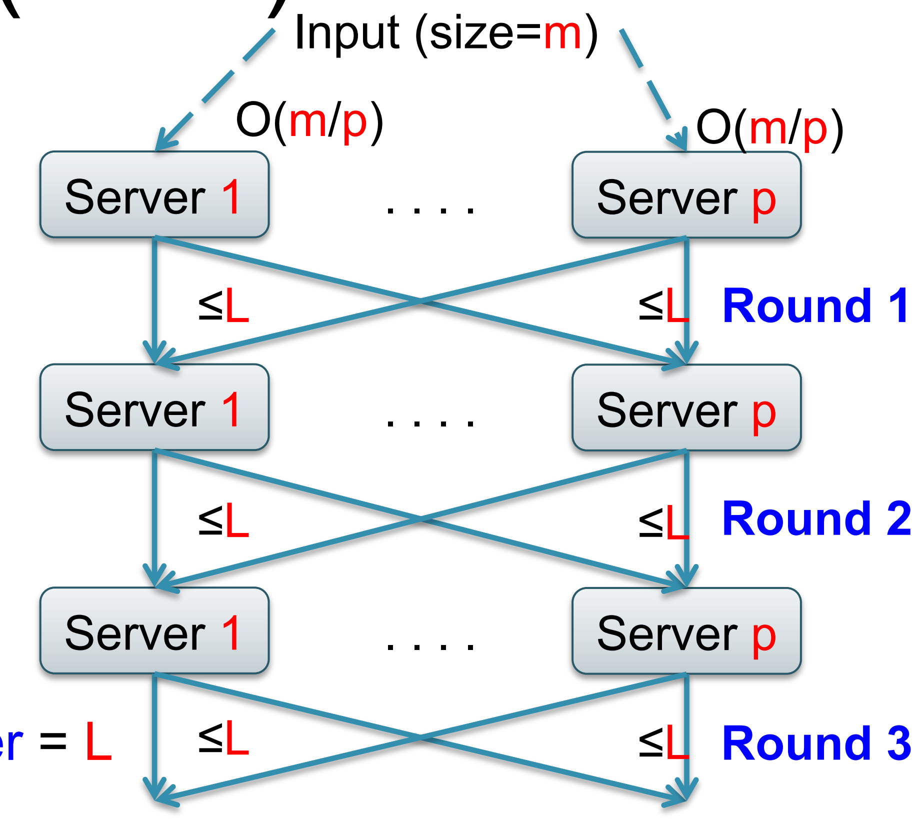
Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$

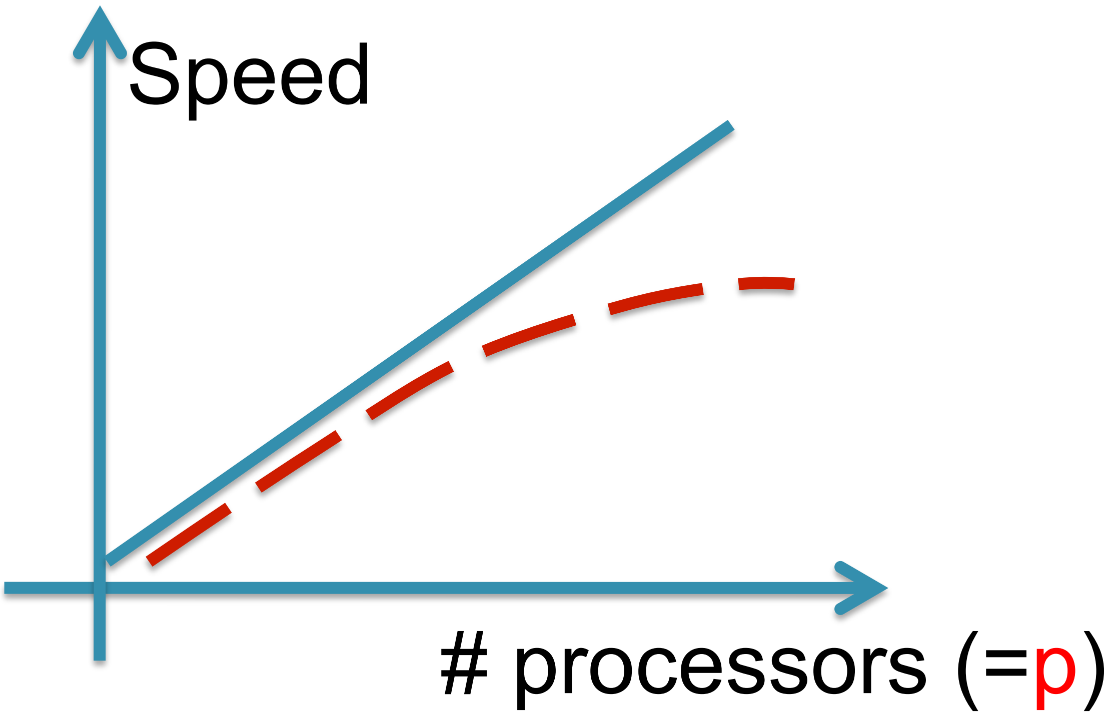



| **Cost:** | Ideal | Practical $\varepsilon \in (0,1)$ | Naïve 1 | Naïve 2 |
|---|---|---|---|---|
| Load $L$ | $L = m/p$ | $L = m/p^{1-\varepsilon}$ | $L = m$ | $L = m/p$ |
| Rounds $r$ | 1 | $O(1)$ | 1 | $p$ |

# Data Complexity of the Load

- Query $Q$ is fixed blue
- Input: data $m$, servers $p$ red
- Load: $L = f(m, p)$
- We discuss one round only

# Speedup




A load of $L = m/p$ corresponds to linear speedup

A load of $L = m/p^{1-\varepsilon}$ corresponds to sub-linear speedup

# Example: Join(x,y,z) = R(x,y), S(y,z)

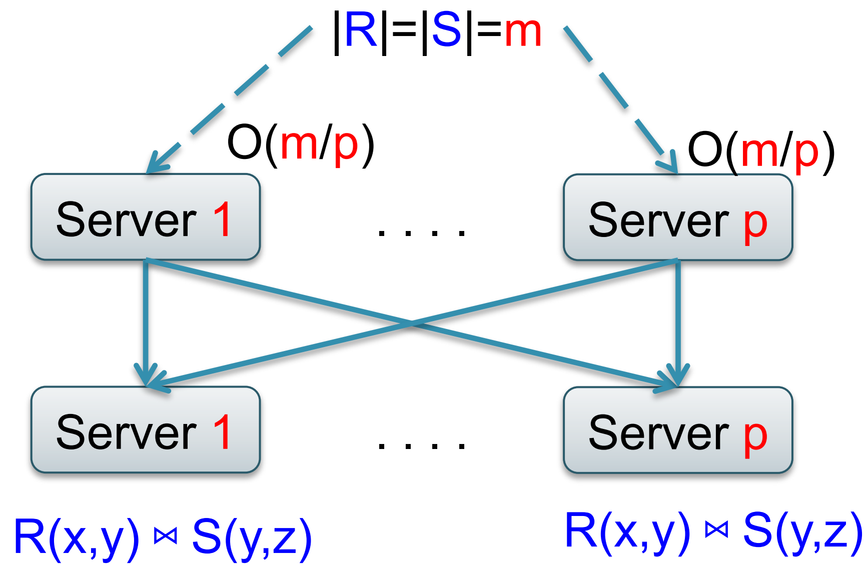
**R**

| x | y |
|---|---|
| a | b |
| a | c |
| b | c |

$\bowtie$

**S**

| y | z |
|---|---|
| b | d |
| b | e |
| c | e |




**Input**: R, S

- Uniformly partitioned

**Round 1**: each server

- Sends record R(x,y) to server h(y) mod p
- Sends record S(y,z) to server h(y) mod p

**Output**: each server

- local join R(x,y) ⋈ S(y,z)

Assuming no skew

L = O(m/p) w.h.p.

Trianges(x,y,z) = R(x,y), S(y,z), T(z,x) |R| = |S| = |T| = m tuples

# Example: Triangles

**Round 1**:

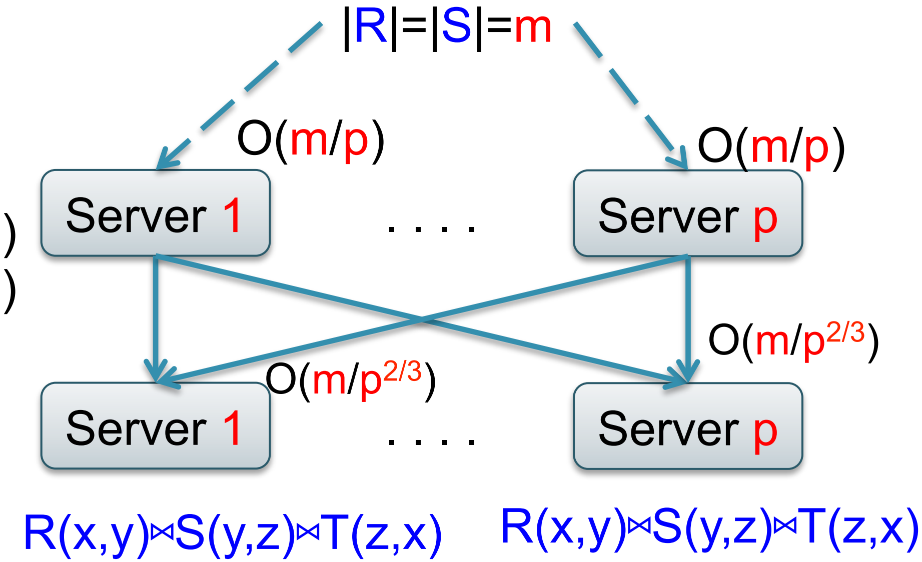
Send R(x,y) to all servers ($h_1$(x),$h_2$(y),*)

Send S(y,z) to all servers (*, $h_2$(y), $h_3$(z))

Send T(z,x) to all servers ($h_1$(x), *, $h_3$(z))

**Output**:

compute locally R(x,y) ⋈ S(y,z) ⋈ T(z,x)

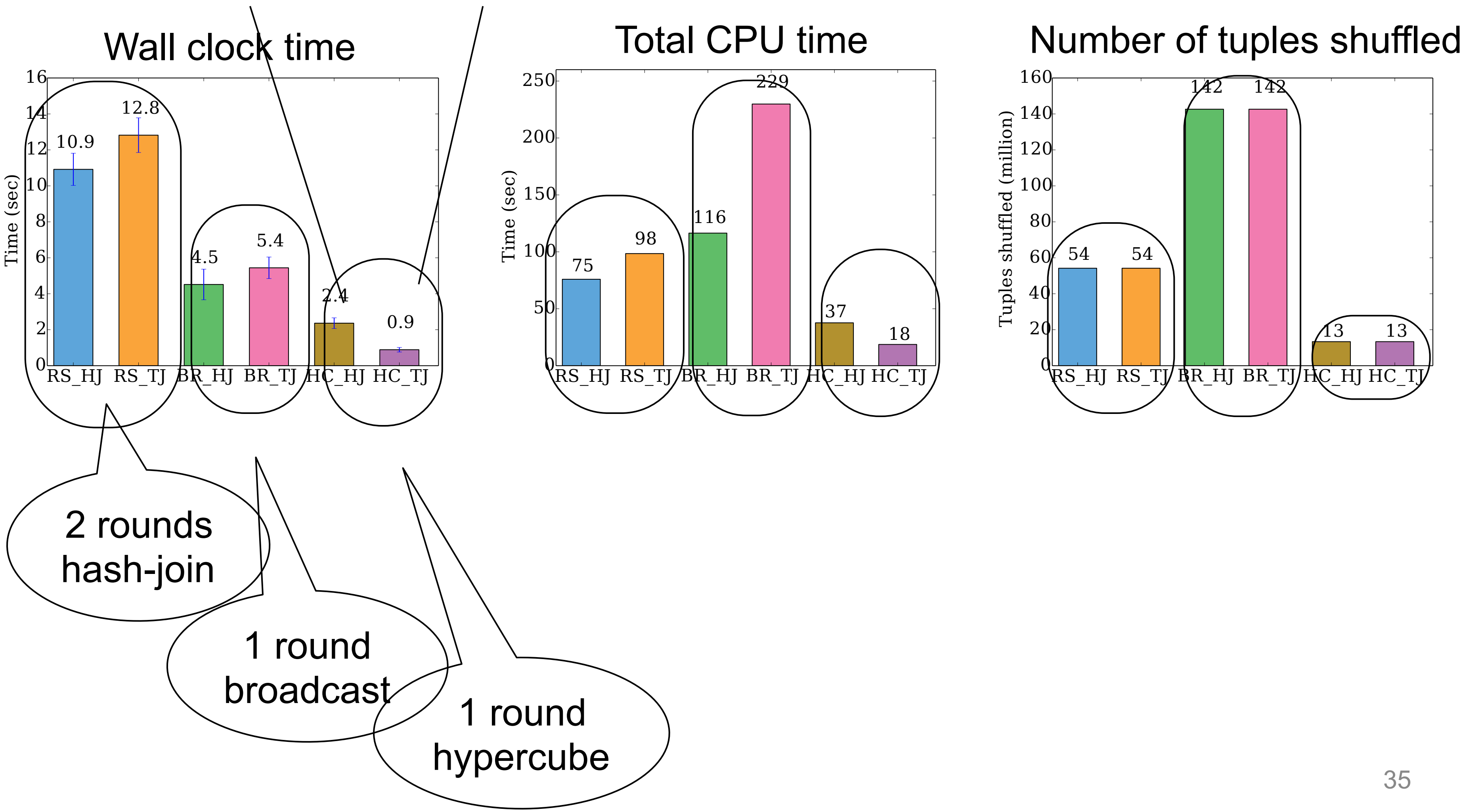



**Theorem** If data has "no skew", then HyperCube computes Triangles with load/server L = O($m/p^{2/3}$) w.h.p.

Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)    |R| = |S| = |T| = 1.1M

# 1.1M triples of Twitter data → 220k triangles;   p=64

**local 1 or 2-step hash-join; local 1-step Leapfrog Trie-join (a.k.a. Generic-Join)**



Trianges(x,y,z) = R(x,y), S(y,z), T(z,x) |R| = |S| = |T| = 1.1M

1.1M triples of Twitter data → 220k triangles; p=64

| shuffle | tuples sent | producer skew | consumer skew |
|---|---|---|---|
| R(x, y) ->h(y) | 1,114,289 | 1 | 1.35 |
| S(y, z) ->h(y) | 1,114,289 | 1 | 1.72 |
| RS(x, y, z) ->h(z) | 50,862,578 | 20.8 | 1 |
| T(z, x) ->h(z) | 1,114,289 | 1 | 1.01 |
| Total | 54,205,445 | N.A. | N.A. |

Table 2: Load balance with regular shuffles in query Q1

| shuffles | tuples sent | producer skew | consumer skew |
|---|---|---|---|
| HCS R(x, y) | 4,457,156 | 1 | 1.05 |
| HCS S(y, z) | 4,457,156 | 1 | 1.05 |
| HCS T(z, x) | 4,457,156 | 1 | 1.05 |
| Total | 13,371,468 | N.A. | N.A. |

Table 3: Load balance with HyperCube shuffles in query Q1

# HperCube Algorithm for Full CQ

$$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_\mathbf{1}), \mathbf{S_2}(\bar{\mathbf{x}}_\mathbf{2}), \ldots, \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$$

- Write: $p = p_1 * p_2 * \ldots * p_k$

  $p_i$ = a "*share*"

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

  $h_{j1}(x_{j1}), \quad h_{j2}(x_{j2}), \quad \ldots$

  independent hash functions

- **Output:** compute $Q$ locally

# Computing Shares $p_1, p_2, \ldots, p_k$

$$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_\mathbf{1}), \mathbf{S_2}(\bar{\mathbf{x}}_\mathbf{2}), \ldots, \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$$

Load/server from $S_j$ : $L_j = m_j / (p_{j1} * p_{j2} * \ldots)$

Optimization problem: find $p_1 * p_2 * \ldots * p_k = p$

[Beame'13] linear opt: Minimize $\max_j L_j$

# Vertex Cover / Edge Packing

Hyper-graph: nodes $x_1, \ldots, x_k$, edges $S_1, \ldots, S_l$

**Def**. A *fractional vertex cover*: k numbers $v_1, v_2, \ldots v_k \geq 0$ such that for every hyperedge $S_j$: $\Sigma_i v_i \geq 1$

**Def**. A *fractional edge packing*: l numbers $u_1, u_2, \ldots u_l \geq 0$ such that for every node $x_i$: $\Sigma_j u_j \leq 1$

$$\Sigma_i v_i \geq \Sigma_j u_j$$

$$\min_{v1,\ldots vk} \Sigma_i v_i = \max_{u1,\ldots ul} \Sigma_j u_j = \tau^*$$

# Optimal L

$$\mathbf{Q(x_1, \ldots, x_k) = S_1(\bar{x}_1), S_2(\bar{x}_2), \ldots, S_\ell(\bar{x}_\ell)}$$

Suppose $m_1 = \ldots = m_l = m$

Fractional vertex packing:

**Thm**. For every $v_1, \ldots v_k$,
there exists an algorithm for $Q$
with $L \leq m / p^{1/\tau}$ w.h.p.
on skew-fee instances,
where $\tau = \Sigma_i v_i$

$\geq$

Fractional edge cover:

**Thm**. For every $u_1, \ldots u_l$,
any algorithm for $Q$
has $L \geq m / p^{1/\tau}$
even on skew-fee instances,
where $\tau = \Sigma_j u_j$

**Proof**: use shares $p_i = p^{v_i / \tau}$

**Proof**: uses Friedgut's inequality

Optimal load: $L = m / p^{1/\tau^*}$ Optimal speedup: $1/p^{1/\tau^*}$

# Example

$Q(x,y,z) = R(x,y), S(y,z), T(z,x)$

Suppose $m_R = m_S = m_T = m$

Fractional vertex packing:

$$
\begin{array}{llll}
\min(v_R + v_S + v_T) & & & \\
 & & & \\
R: & v_x + v_y & & \geq 1 \\
S: & \quad\;\; v_y + v_S & & \geq 1 \\
T: & v_x + \quad\;\; v_z & & \geq 1
\end{array}
$$

$\boxed{\geq}$

Fractional edge cover:

$$
\begin{array}{llll}
\max(w_R + w_S + w_T) & & & \\
 & & & \\
x: & w_R \quad\quad + w_T & & \leq 1 \\
y: & w_R + w_S & & \leq 1 \\
z: & \quad\quad w_S + w_T & & \leq 1
\end{array}
$$

Optimal load: $L = m / p^{2/3}$ Optimal speedup: $1/p^{2/3}$

# Lessons So Far

- MPC model: cost = communication load + rounds
- **HyperCube**: rounds=1, $L = m/p^{1/\tau^*}$ Sub-linear speedup
  Note: it only shuffles data! Still need to compute Q locally.
- Strong optimality guarantee: any algorithm with
  better load $m/p^s$ reports only $1/p^{s\times\tau^*-1}$ fraction of answers.

  Parallelism gets harder as p increases!
- Total communication = $p\times L = m \times p^{1-1/\tau^*}$

  MapReduce model is wrong! It encourages many reducers p

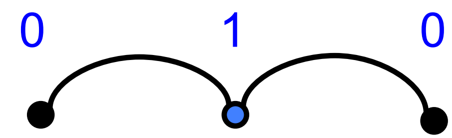
# Skew Matters

- If the database is skewed, the query becomes provably harder. We want to optimize for the common case (skew-free) and treat skew separately

- Parallel processing different from sequential processing, were worst-case optimal algorithms (LFTJ, generic-join) are for arbitrary instances, skewed or not.

# Skew Matters

- Join(x,y,z) = R(x,y),S(y,z)




$\tau^* = 1$

L = m/p

- Suppose R, S are skewed, e.g. single value y
- The query becomes a cartesian product!

Product(x,z) = R(x),S(z)


$\tau^* = 2$

$L = m/p^{1/2}$

Lets examine skew…

# All You Need to Know About Skew

Hash-partition a bag of $m$ data values to $p$ bins

**Fact 1** Expected size of any **one** fixed bin is $m/p$

**Fact 2** Say that database is *skewed* if some value has degree > $m/p$. Then **some** bin has load > $m/p$

**Fact 3** Conversely, if the database is *skew-free* then max size of **all** bins = $O(m/p)$ w.h.p.

Hiding log $p$ factors

| | |
|---|---|
| Join: | if $\forall$ degree < $m/p$ then $L = O(m/p)$ w.h.p |
| Triangles: | if $\forall$ degree < $m/p^{1/3}$ then $L = O(m/p^{2/3})$ w.h.p |
| In general: | if $\forall$ degree < $m/p^{v_i/\tau^*}$ then $L = O(m/p^{1/\tau^*})$ w.h.p |

# AGM Bound + Optimal Alg.

$Q(x,y,z) = R(x,y), S(y,z), T(z,x)$

Hypergraph = variables + relations

(Generalized) fractional vertex packing:

$\max(v_R + v_S + v_T)$

Log base m

$$\begin{array}{llcl}
R: & v_x + v_y & & \le \log|R| \\
S: & & v_y + v_S & \le \log|S| \\
T: & v_x + & v_z & \le \log|T|
\end{array}$$

(Generalized) fractional edge cover:

$\min(w_R \log|R| + w_S \log|S| + w_T \log|T|)$

$$\begin{array}{llll}
x: & w_R & + w_T & \ge 1 \\
y: & w_R + w_S & & \ge 1 \\
z: & w_S & + w_T & \ge 1
\end{array}$$

**Thm**. For any feasible $v_R, v_S, v_T$

$\log|Q| \ge$ objective

$|Q| \ge m^{vx} \times m^{vy} \times m^{vz}$

$\le$

**Thm**. For any feasible $w_R$, $w_S$, $w_T$:

$\log|Q| \le$ objective

$|Q| \le |R|^{wR} \times |S|^{wS} \times |T|^{wT}$

Proof "Free" instance

$R(x,y) = [m^{vx}] \times [m^{vy}]$

$S(y,z) = [m^{vy}] \times [m^{vz}]$

$T(z,x) = [m^{vx}] \times [m^{vz}]$

Proof. (use Friedgut's inequality)

Optimal edge cover = $\rho^*$

AGM bound = $m^{\rho^*}$

Generic-join algorithm, time = $O(m^{\rho^*})$

# The AGM Inequality

$$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_1), \mathbf{S_2}(\bar{\mathbf{x}}_2), \ldots, \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$$

$m_1 = \ldots = m_l = m$

**Fact**. Any MPC algorithm satisfies $r \times L \geq m / p^{1/\rho^*}$
where $r$ = # rounds, $L$ = load/server

Informal proof:

- AGM lower bound: there exists db s.t. $|Q| = m^{\rho^*}$
- Each server receives subset $|S_i| \leq r \times L$
- AGM upper bound: server reports $\leq (r \times L)^{\rho^*}$ answers
- All $p$ servers report $\leq p \times (r \times L)^{\rho^*}$ answers, $= m^{\rho^*}$

# Lessons so Far

- Skew affects communication dramatically
  - E.g. Join from linear $m/p$ to $m/p^{1/2}$

- Analysis differs:

  Tight

  - w/o skew: $L = m\ /\ p^{1/\tau^*}$ fractional vertex cover
  - w/ skew: $L \geq m\ /\ p^{1/\rho^*}$ fractional edge cover

  Not tight

- Focus on skew-free databases.
  Handle skewed values as a residual query.

# Statistics

- So far: all relations have same size $m$
- In reality, we know their sizes = $m_1$, $m_2$, …

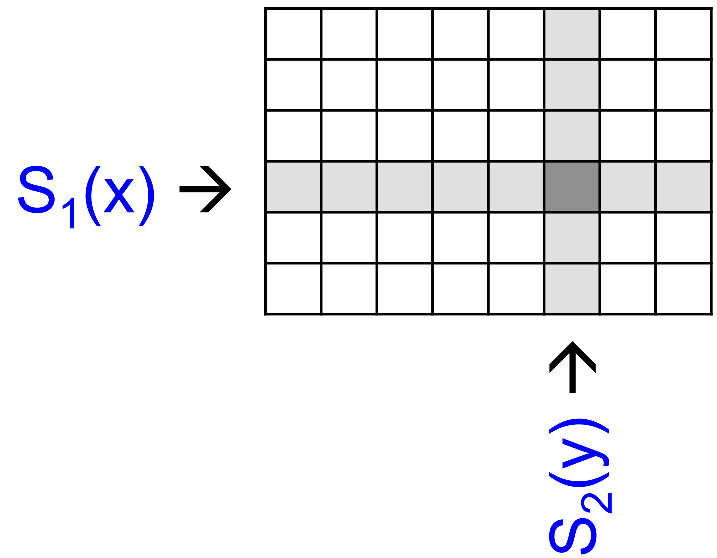
**Q1**: What is the optimal choice of shares?

**Q2**: What is the optimal load $L$?

Will answer **Q2**, giving closed formula for $L$.
Will answer **Q1** indirectly, by showing that HyperCube takes advantage of statistics.

# L for Cross Product

2-way product $Q(x,y) = S_1(x) \times S_2(y)$ $|S_1|=m_1$, $|S_2| = m_2$




Shares $p = p_1 \times p_2$

$L = \max(m_1 / p_1 ,\ m_2 / p_2)$

Minimal L:

$$\mathbf{L} = \left(\frac{\mathbf{m_1 \cdot m_2}}{\mathbf{p}}\right)^{\frac{1}{2}}$$

t-way product: $Q(x_1,\ldots,x_u) = S_1(x_1)\times \ldots \times S_t(x_t)$:

$$\mathbf{L} = \left(\frac{\mathbf{m_1 \cdot m_2 \cdots m_t}}{\mathbf{p}}\right)^{\frac{1}{t}}$$

# L for arbitrary Query Q

$$\mathbf{Q}(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \mathbf{S_1}(\bar{\mathbf{x}}_1), \mathbf{S_2}(\bar{\mathbf{x}}_2), \ldots, \mathbf{S}_\ell(\bar{\mathbf{x}}_\ell)$$

Relations sizes= $m_1$, $m_2$, … Then, for any 1-round algorithm

**Fact** For any integral edge packing $S_{j1}$, $S_{j2}$, …, $S_{jt}$:

$$L \geq \left(\frac{\mathbf{m_{j_1}} \cdot \mathbf{m_{j_2}} \cdots \mathbf{m_{j_t}}}{\mathbf{p}}\right)^{\frac{1}{t}}$$

t = size of packing

**Theorem**: [Beame'14]

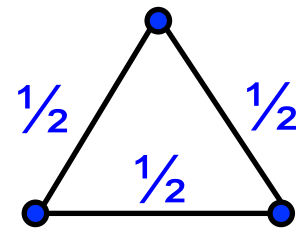
(1) For any fractional edge packing $u_1$, …, $u_l$

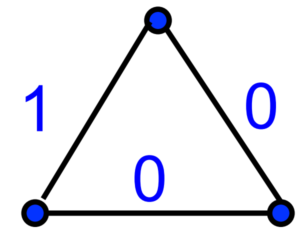
$$L \geq \mathbf{L}(\mathbf{u}) = \left(\frac{\mathbf{m_1}^{u_1} \cdot \mathbf{m_2}^{u_2} \cdots \mathbf{m}_\ell^{u_\ell}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_1}+\mathbf{u_2}+\cdots+\mathbf{u}_\ell}}$$

(2) The optimal load of the HyperCube algorithm is $\max_{\mathbf{u}} L(\mathbf{u})$

# Example

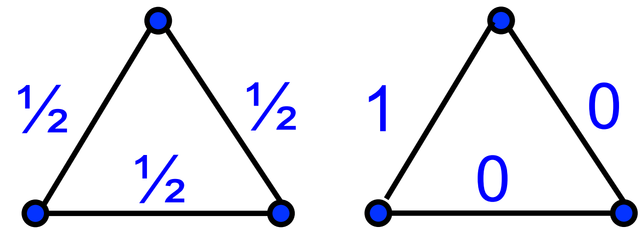
Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)




| Edge packing $u_1, u_2, u_3$ | $\left(\frac{m_1^{u_1} \cdot m_2^{u_2} \cdot m_3^{u_3}}{p}\right)^{\frac{1}{u_1+u_2+u_3}}$ |
|---|---|
| | |
| | |
| | |
| | |

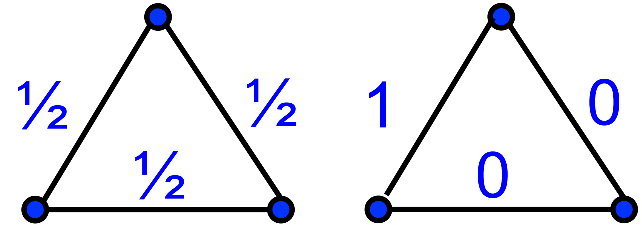
# Example




Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)

| Edge packing $u_1, u_2, u_3$ | $\left(\frac{\mathbf{m_1}^{\mathbf{u_1}} \cdot \mathbf{m_2}^{\mathbf{u_2}} \cdot \mathbf{m_3}^{\mathbf{u_3}}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_1}+\mathbf{u_2}+\mathbf{u_3}}}$ |
|---|---|
| 1/2, 1/2, 1/2 | $(m_1\ m_2\ m_3)^{1/3} / p^{2/3}$ |
| | |
| | |
| | |

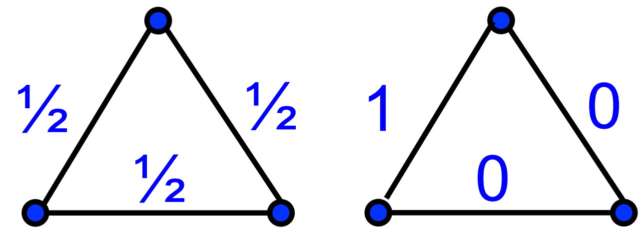
# Example




Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)

| Edge packing $u_1, u_2, u_3$ | $\left(\dfrac{\mathbf{m_1}^{\mathbf{u_1}} \cdot \mathbf{m_2}^{\mathbf{u_2}} \cdot \mathbf{m_3}^{\mathbf{u_3}}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_1}+\mathbf{u_2}+\mathbf{u_3}}}$ |
|---|---|
| 1/2, 1/2, 1/2 | $(m_1\, m_2\, m_3)^{1/3} / p^{2/3}$ |
| 1, 0, 0 | $m_1 / p$ |
| | |
| | |

# Example

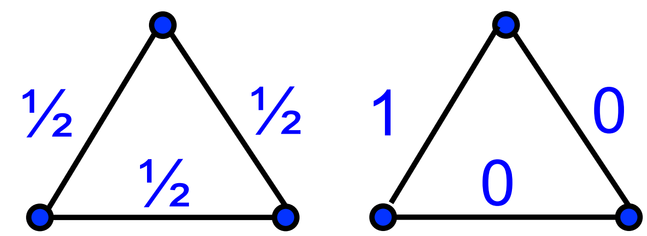



Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)

| Edge packing $u_1, u_2, u_3$ | $\left(\dfrac{m_1^{u_1} \cdot m_2^{u_2} \cdot m_3^{u_3}}{p}\right)^{\frac{1}{u_1+u_2+u_3}}$ |
|---|---|
| 1/2, 1/2, 1/2 | $(m_1\, m_2\, m_3)^{1/3} / p^{2/3}$ |
| 1, 0, 0 | $m_1 / p$ |
| 0, 1, 0 | $m_2 / p$ |
| 0, 0, 1 | $m_3 / p$ |

L = the largest of these four values.

# Example




Trianges(x,y,z) = R(x,y), S(y,z), T(z,x)

| Edge packing $u_1, u_2, u_3$ | $\left(\dfrac{\mathbf{m_1}^{\mathbf{u_1}} \cdot \mathbf{m_2}^{\mathbf{u_2}} \cdot \mathbf{m_3}^{\mathbf{u_3}}}{\mathbf{p}}\right)^{\frac{1}{\mathbf{u_1}+\mathbf{u_2}+\mathbf{u_3}}}$ |
|---|---|
| 1/2, 1/2, 1/2 | $(m_1\ m_2\ m_3)^{1/3} / p^{2/3}$ |
| 1, 0, 0 | $m_1 / p$ |
| 0, 1, 0 | $m_2 / p$ |
| 0, 0, 1 | $m_3 / p$ |

L = the largest of these four values.

Assuming $m_1 > m_2 , m_3$

- When p is small, then $L = m_1 / p$.
- When p is large, then $L = (m_1\ m_2\ m_3)^{1/3} / p^{2/3}$

# Discussion

$$\mathbf{L} = \max_{\mathbf{u}} \left( \frac{\mathbf{m_1}^{u_1} \cdot \mathbf{m_2}^{u_2} \cdots \mathbf{m}_\ell^{u_\ell}}{\mathbf{p}} \right)^{\frac{1}{\mathbf{u_1}+\mathbf{u_2}+\cdots+\mathbf{u}_\ell}}$$

Speedup

**Fact 1** L = [geometric-mean of $m_1$,$m_2$,..] / $p^{1/\Sigma u_j}$

**Fact 2** As p increases, speedup degrades.

$1/p^{1/\Sigma u_j} \rightarrow 1/p^{1/\tau^*}$

**Fact 3** . If $m_j < m_k/p$ , then $u_j = 0$.

Intuitively: broadcast the small relations $S_j$

# Summary

- Traditional query plans are sub-optimal for complex, cyclic queries
- Real data/applications: keys or UDF’s make traditional plans suboptimal even for acyclic queries
- Novel algorithm are now emerging, especially for large-scale data analytics