Parallel DBs

April 25, 2017

Why Scale Up?

Scan of 1 PB at 300MB/s (SATA r2 Limit)

~1 Hour ~3.5 Seconds

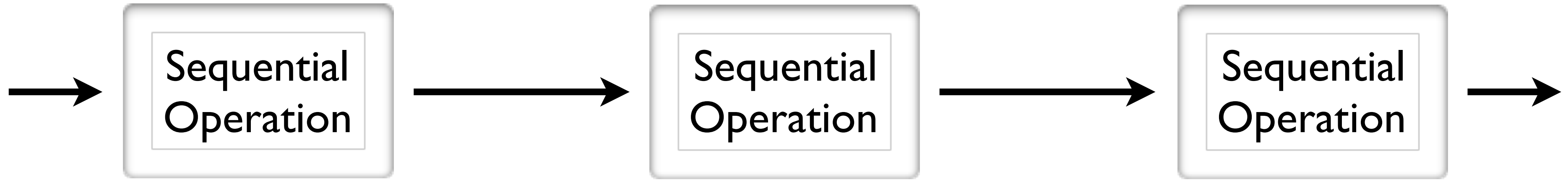
Data Parallelism

Replication

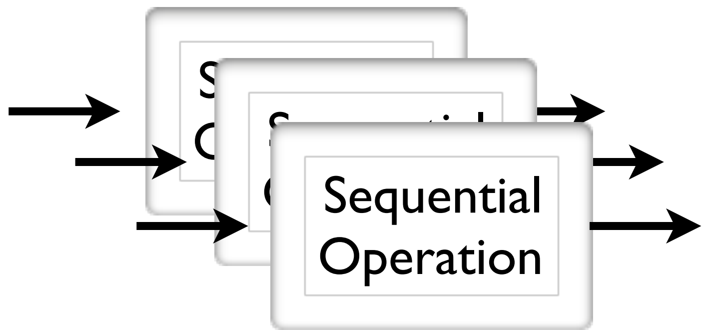
Partitioning

Operator Parallelism

- Pipeline Parallelism: A task breaks down into stages; each machine processes one stage.



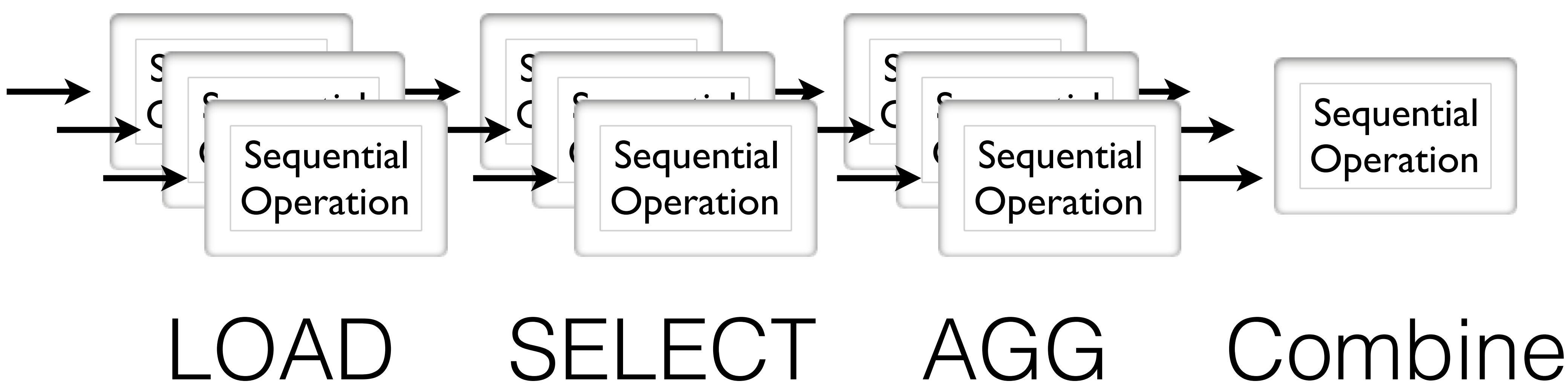
- Partition Parallelism: Many machines doing the same thing to different pieces of data.



Types of Parallelism

- Both types of parallelism are natural in a database management system.

```
SELECT SUM(…) FROM Table WHERE …
```

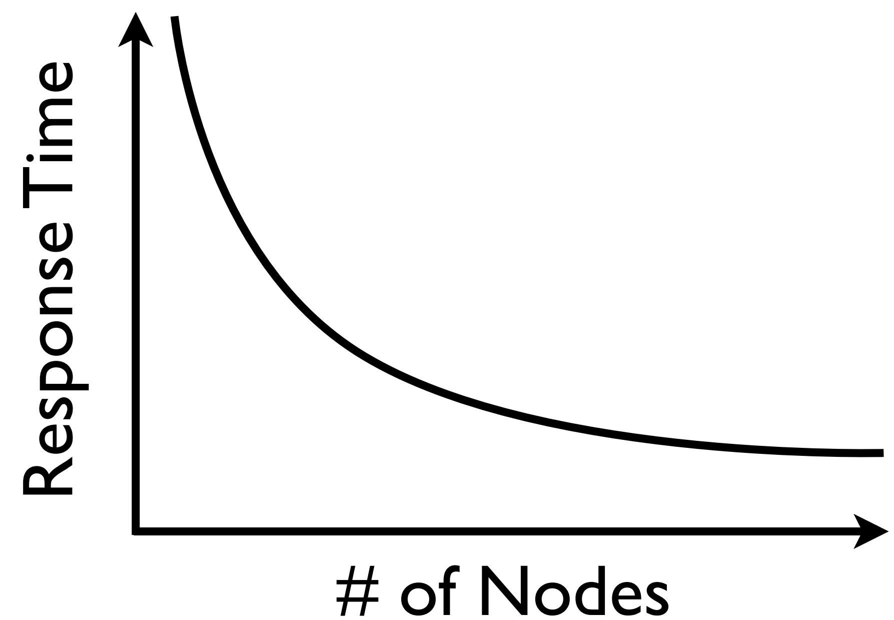


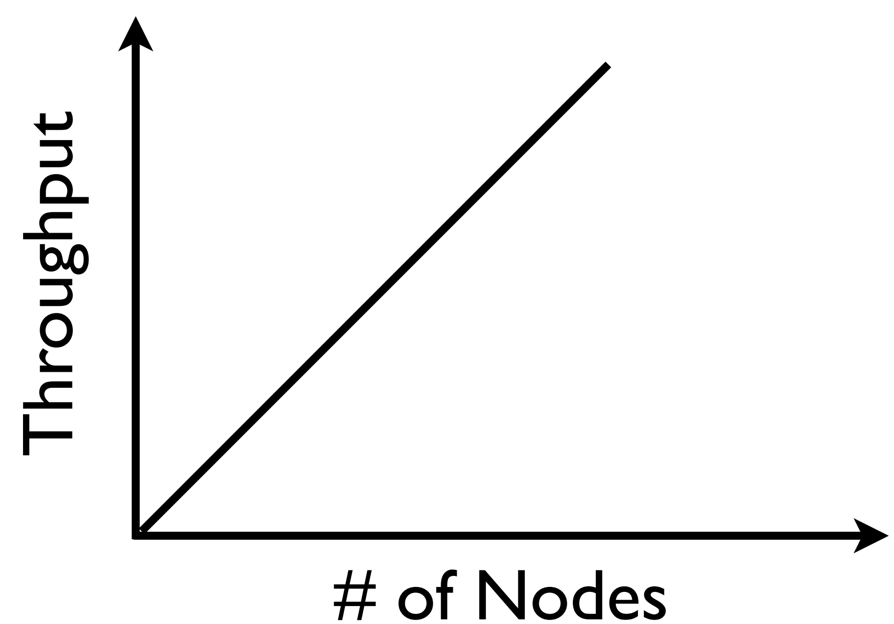
DBMSes: The First || Success Story

- Every major DBMS vendor has a || version.
- Reasons for success:
  - Bulk Processing (Partition ||-ism).
  - Natural Pipelining in RA plan.
  - <u>Users don't need to think in ||</u>.

Types of Speedup

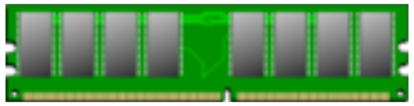
- Speed-up ||-ism
  - More resources = proportionally less time spent.
- Scale-up ||-ism
  - More resources = proportionally more data processed.





Parallelism Models

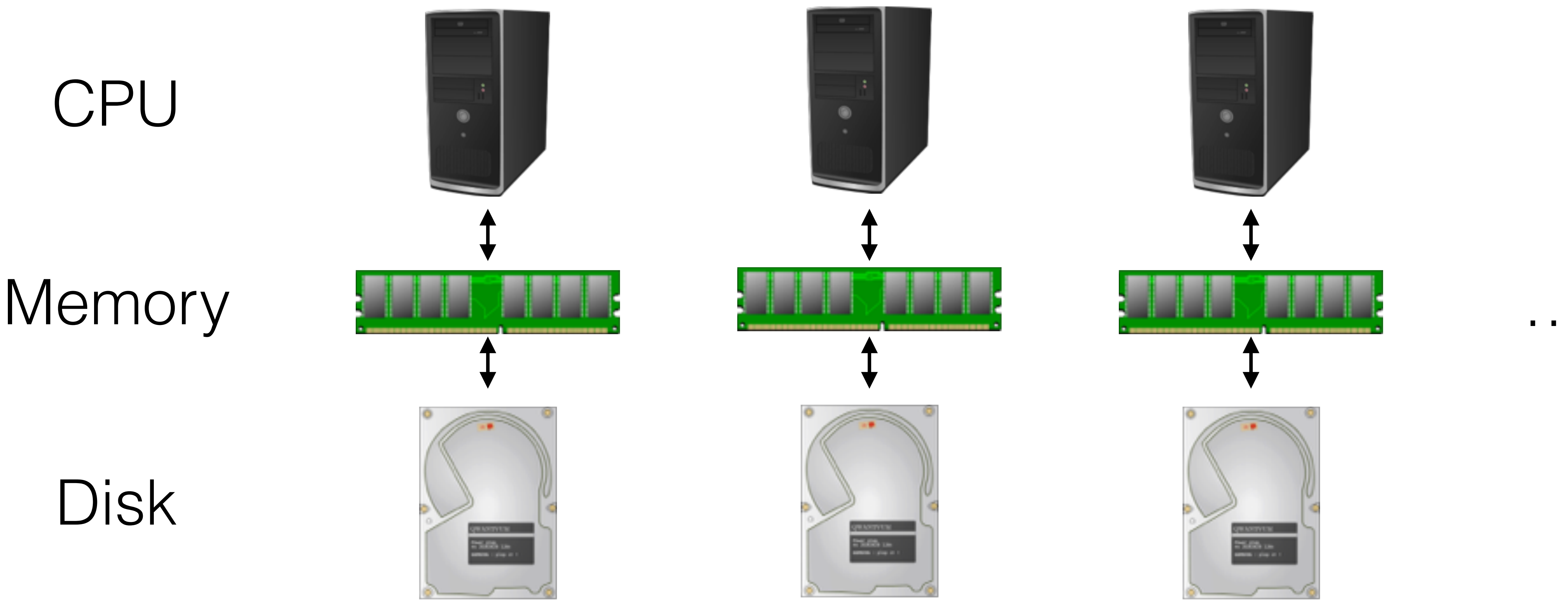
CPU



Memory

Disk

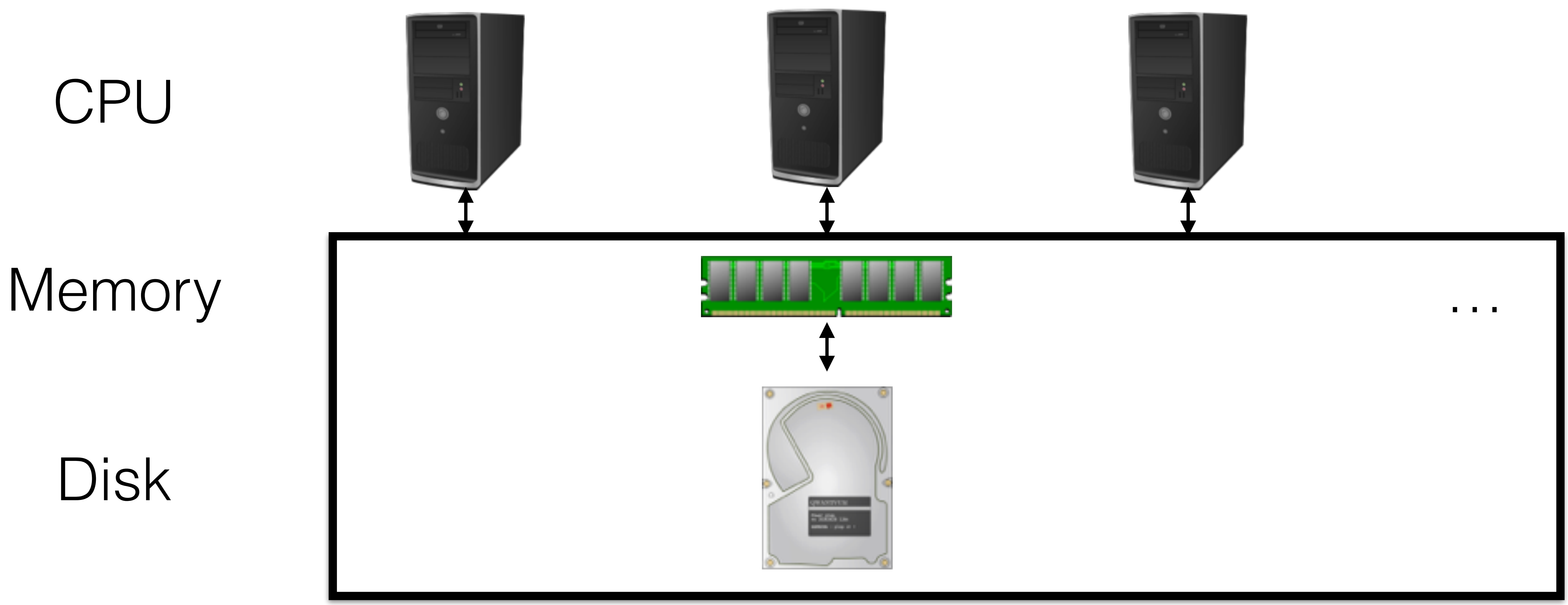
Parallelism Models



How do the nodes communicate?

Parallelism Models

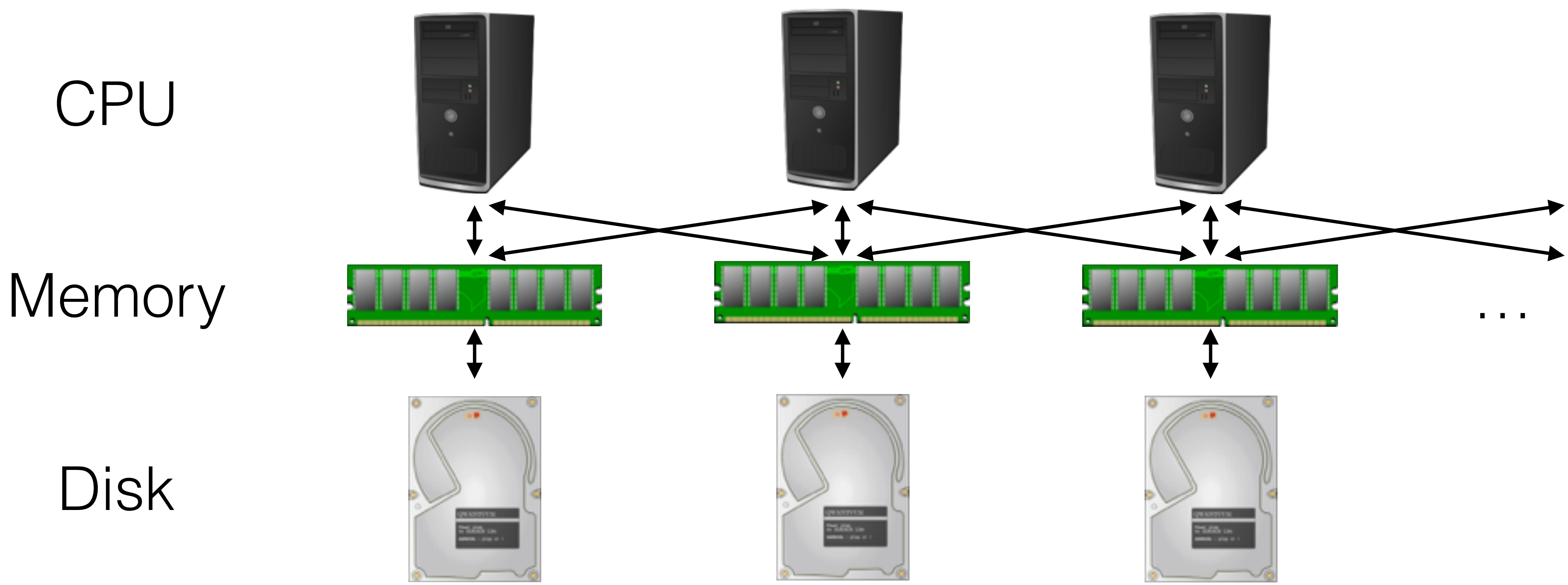
Option 1: “Shared Memory” available to all CPUs



e.g., a Multi-Core/Multi-CPU System

Parallelism Models

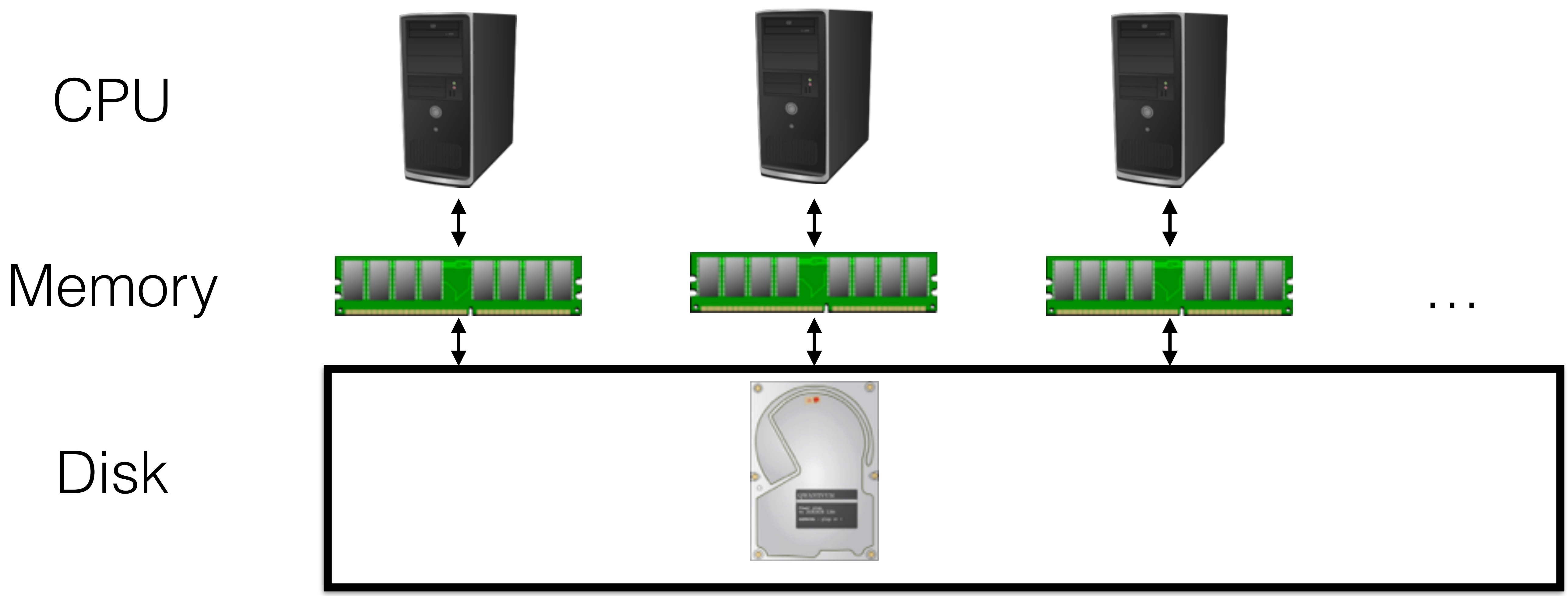
Option 2: Non-Uniform Memory Access.



Used by most AMD servers

Parallelism Models

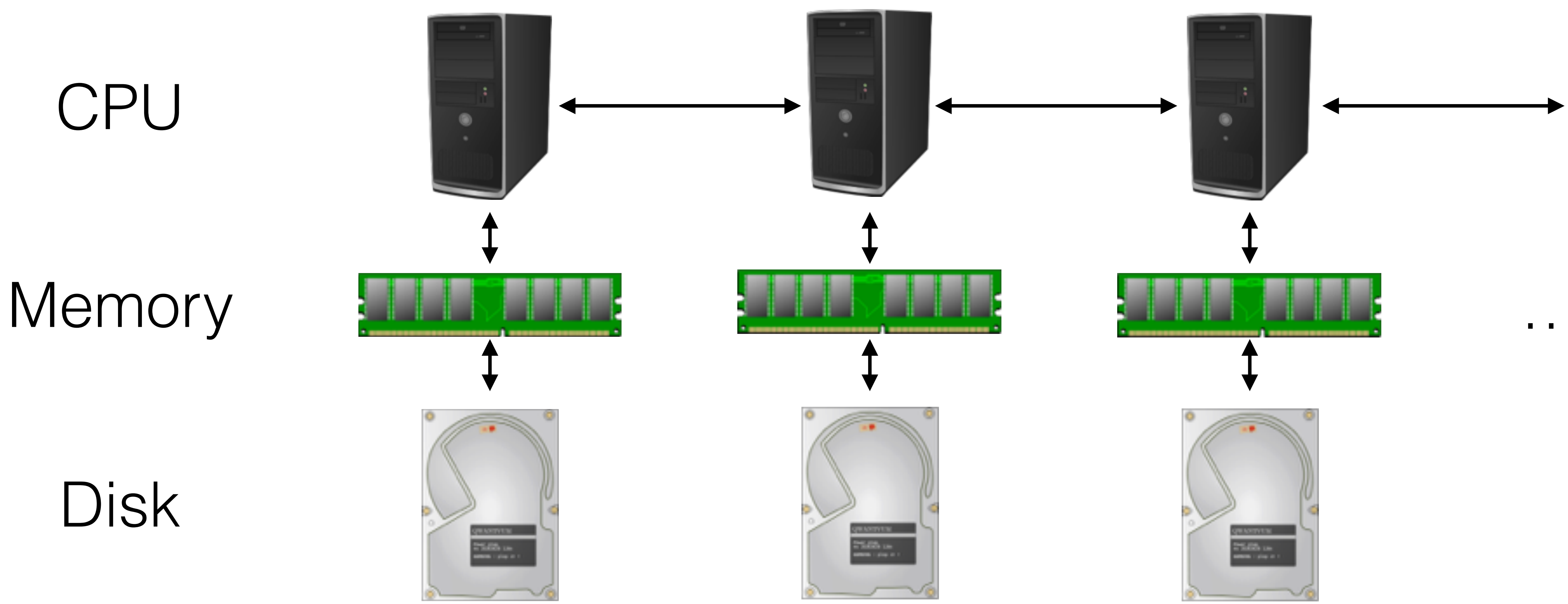
Option 3: “Shared Disk” available to all CPUs



Each node interacts with a “disk” on the network.

Parallelism Models

Option 4: "Shared Nothing" in which all communication is explicit.



Examples include MPP, Map/Reduce. Often used as basis for other abstractions.

Parallelizing

OLAP - Parallel Queries

OLTP - Parallel Updates

Parallelizing

OLAP - Parallel Queries

OLTP - Parallel Updates

Parallelism & Distribution

- *Distribute* the Data
  - Redundancy
  - Faster access
- *Parallelize* the Computation
  - Scale up (compute faster)
  - Scale out (bigger data)

Operator Parallelism

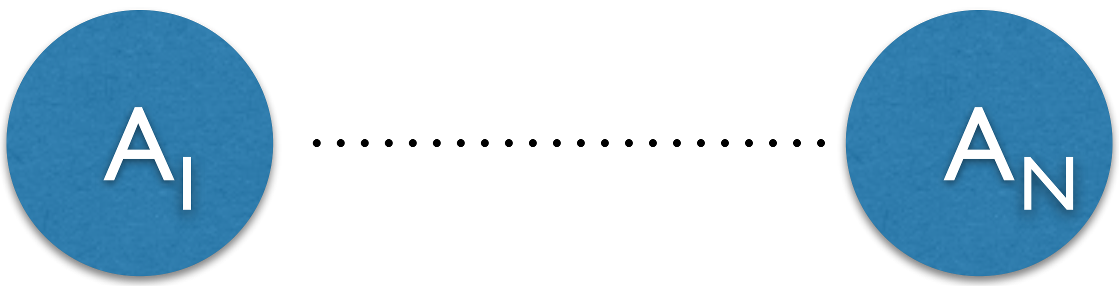
- **General Concept**: Break task into individual units of computation.
- **Challenge**: How much data does each unit of computation need?
- **Challenge**: How much data *transfer* is needed to allow the unit of computation?

Same challenges arise in Multicore, CUDA programming.

Parallel Data Flow

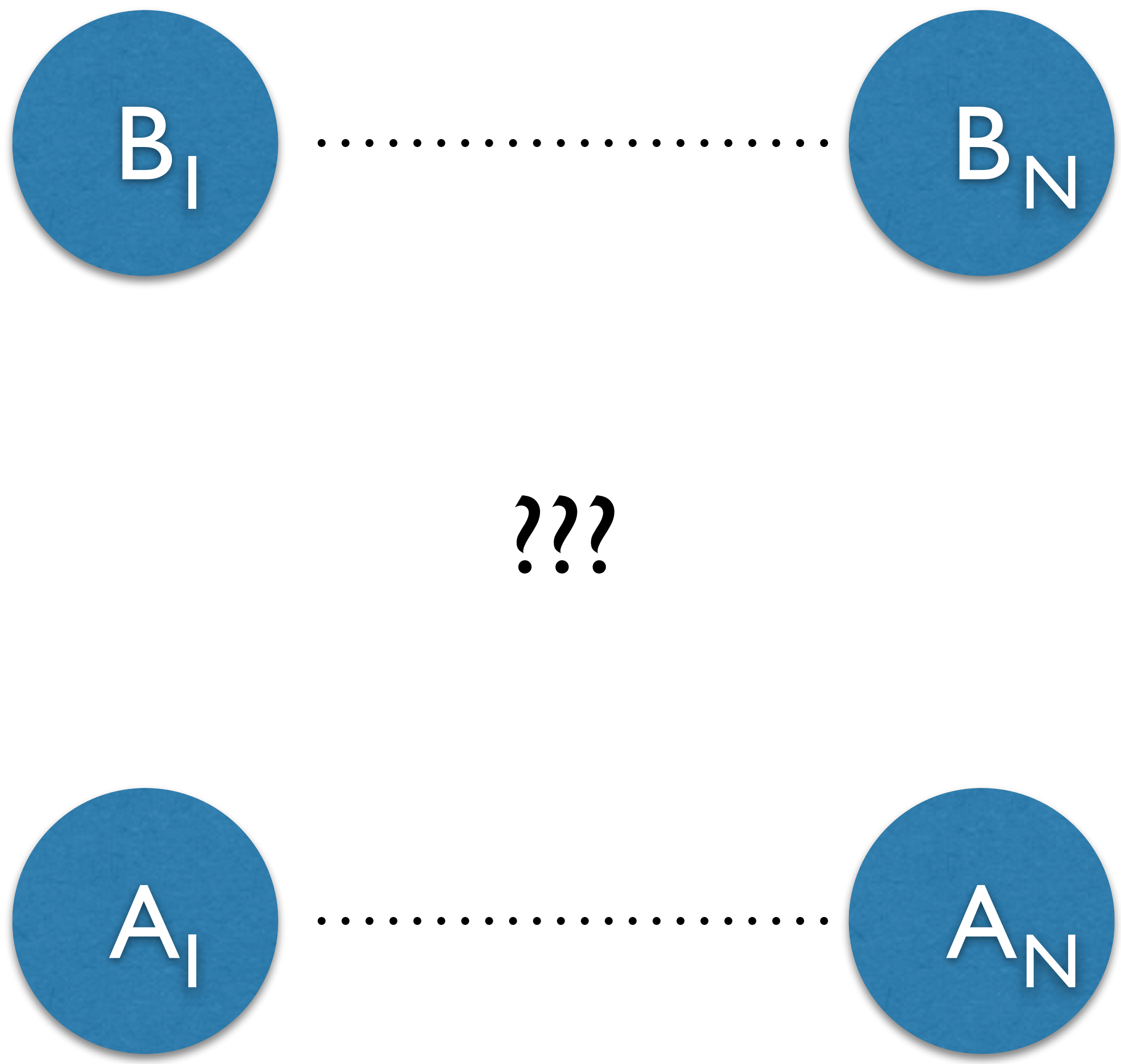
No Parallelism

Parallel Data Flow



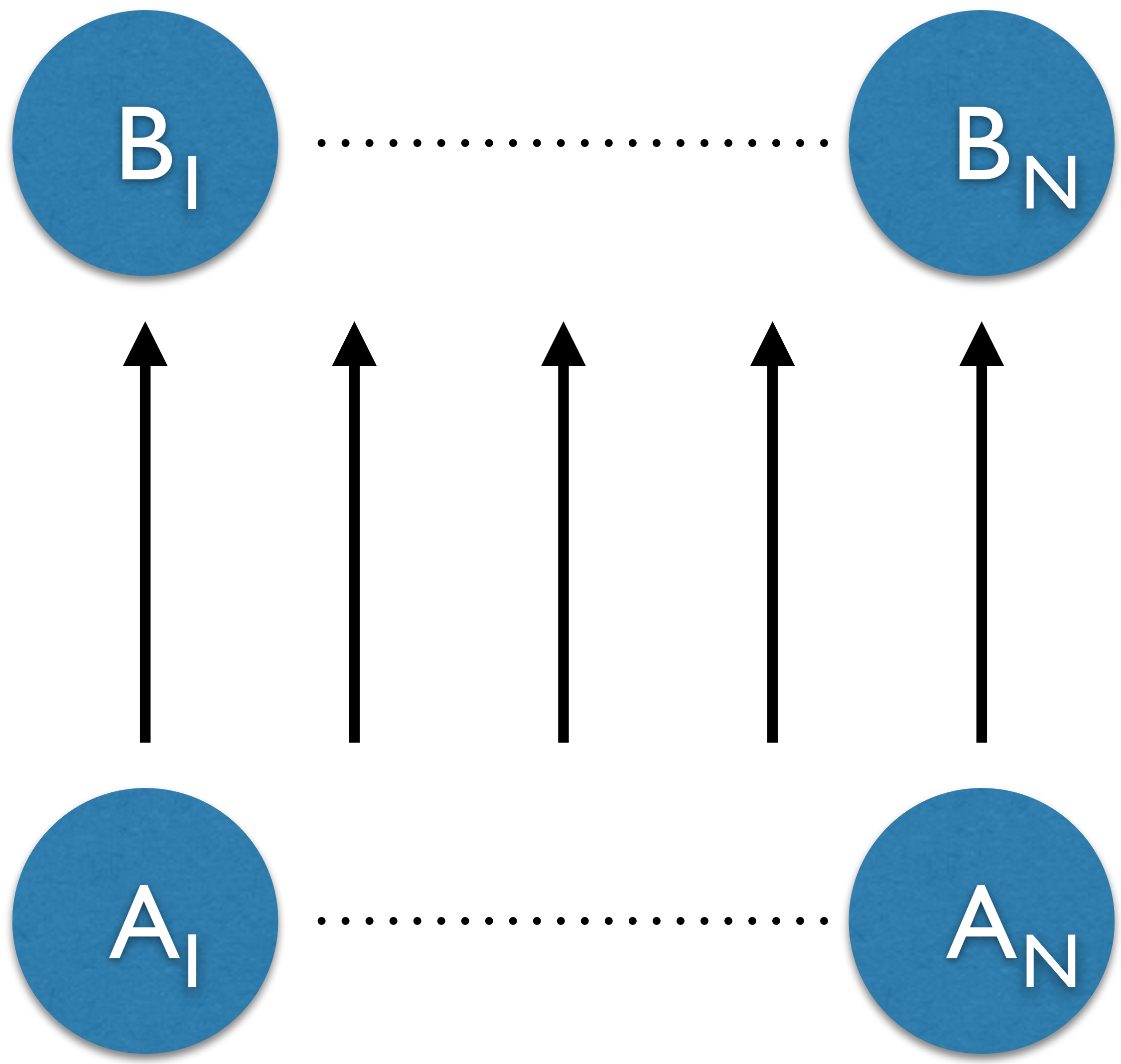
N-Way Parallelism

Parallel Data Flow



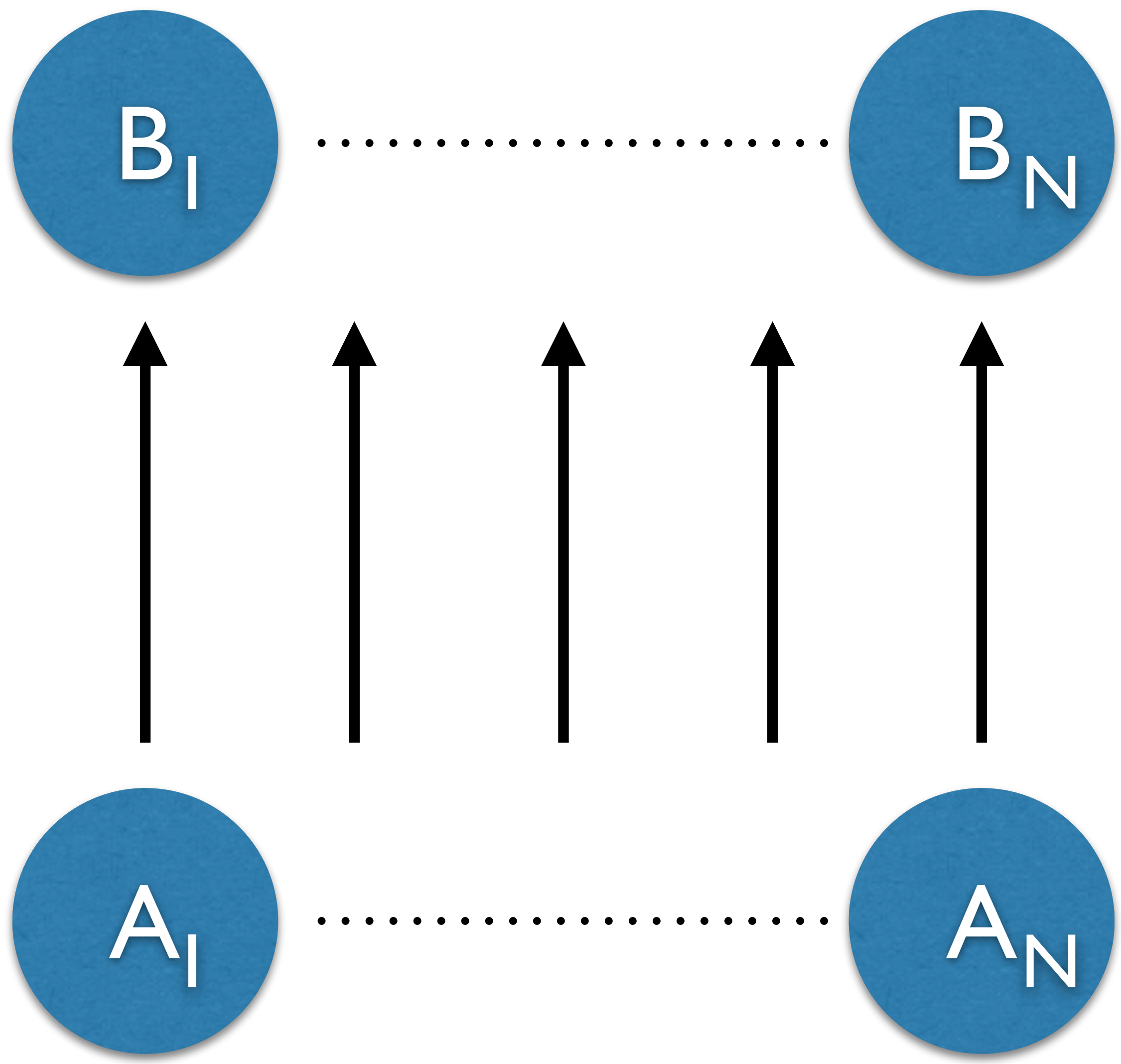
Chaining Parallel Operators

Parallel Data Flow



One-to-One Data Flow ("Map")

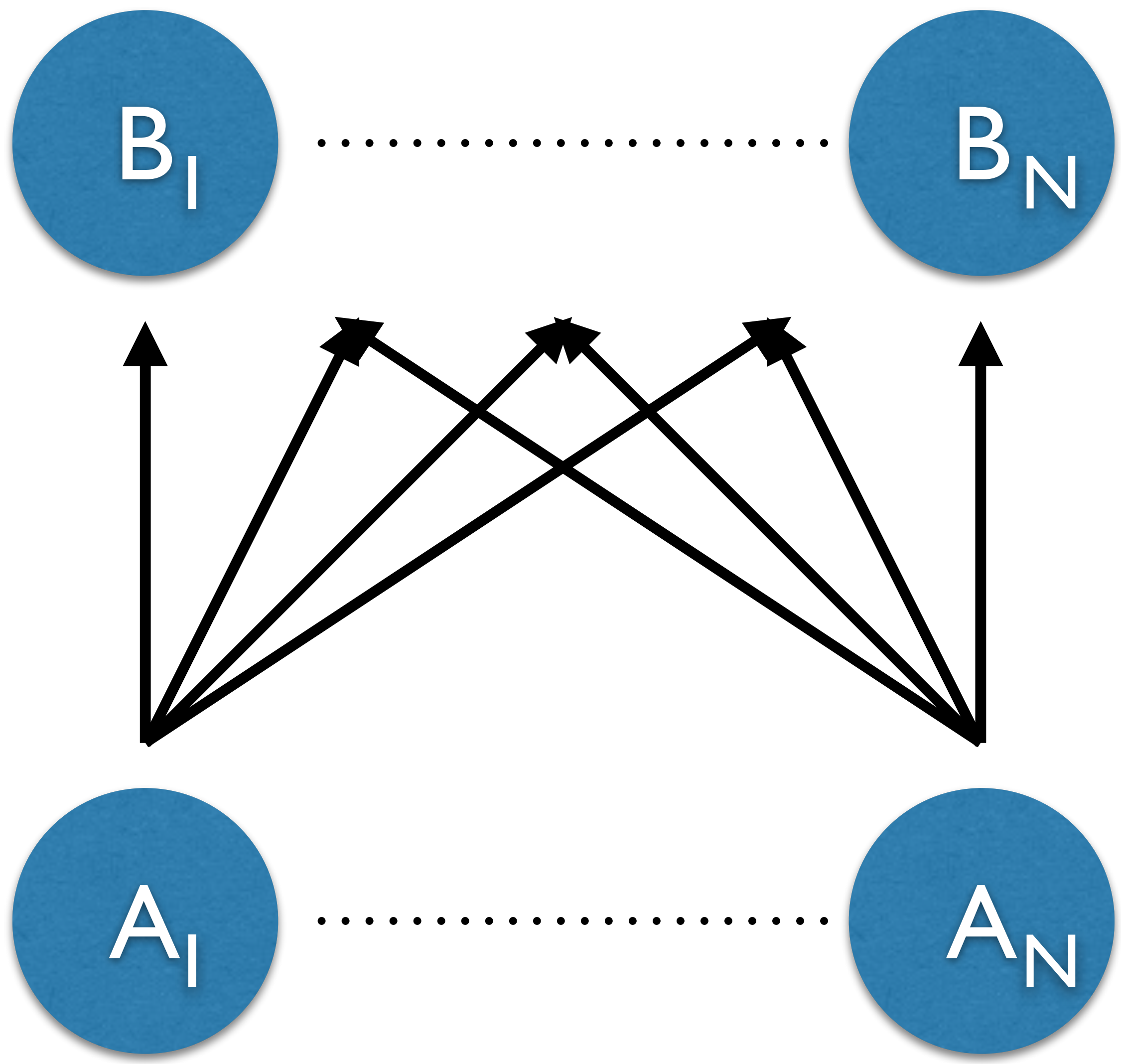
Parallel Data Flow



One-to-One Data Flow

Parallel Data Flow

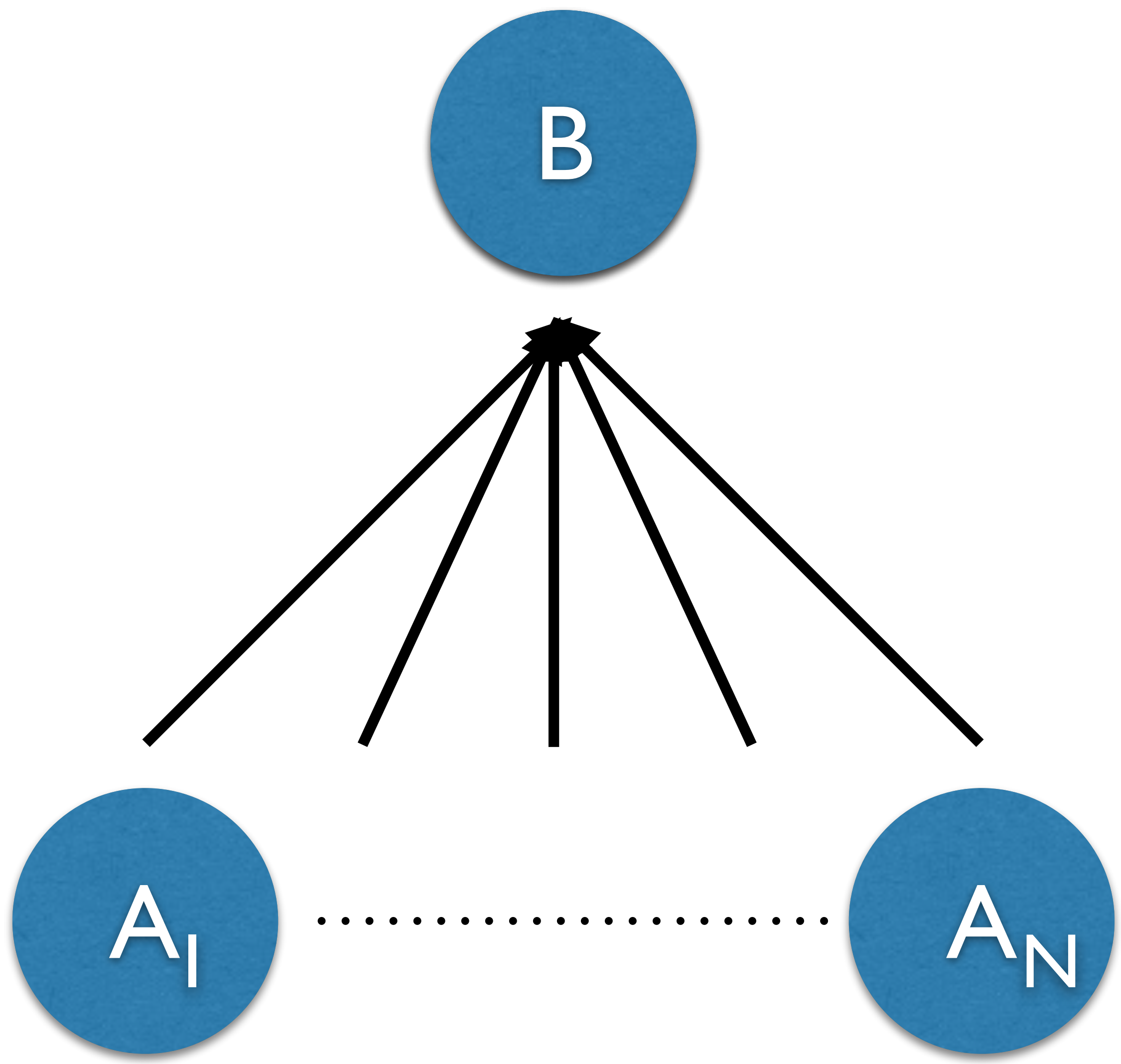
<u>Extreme 1</u>
All-to-All
All nodes send all records to all downstream nodes



<u>Extreme 2</u>
Partition
Each record goes to exactly one downstream node

Many-to-Many Data Flow

Parallel Data Flow



Many-to-One Data Flow (“Reduce/Fold”)

Parallel Operators

Select **Project** **Union (bag)**

What is a logical “unit of computation”?

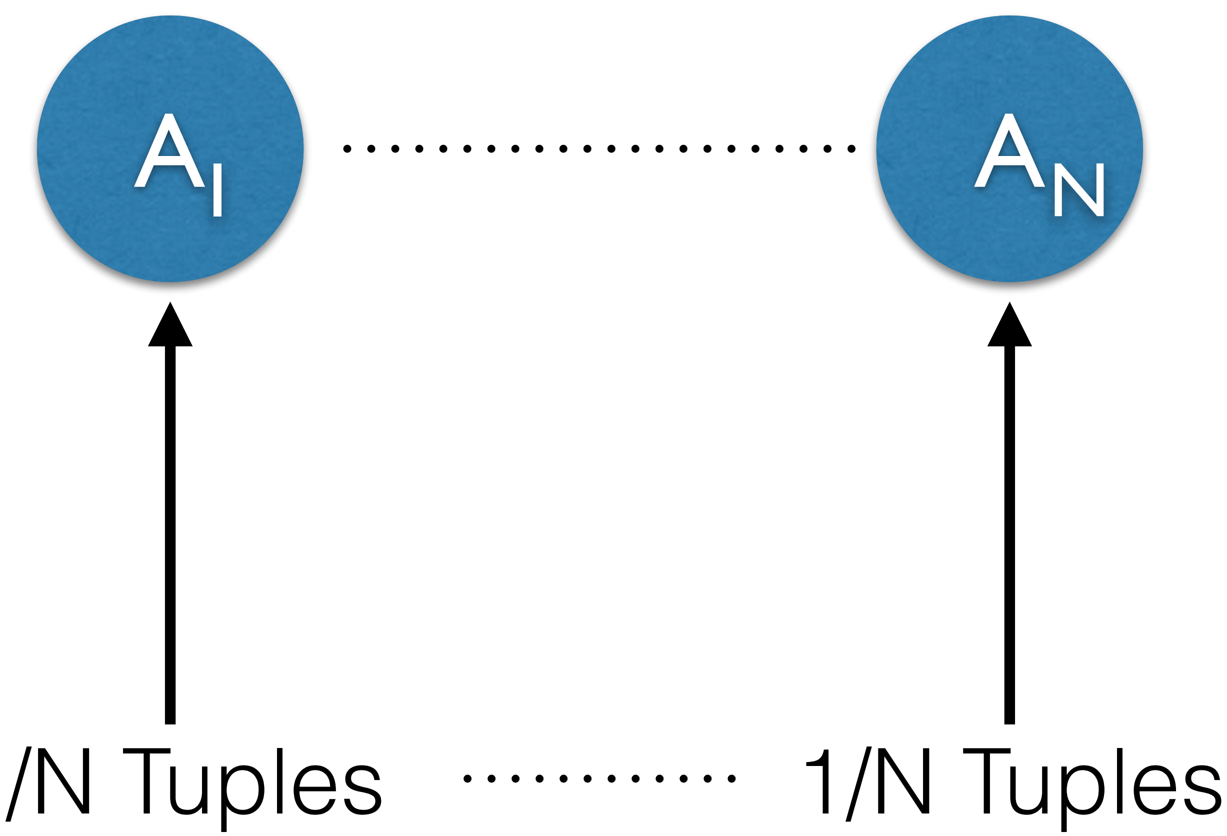
(1 tuple)

Is there a data dependency between units?

(no)

Parallel Operators

Select **Project** **Union (bag)**



Parallel Joins

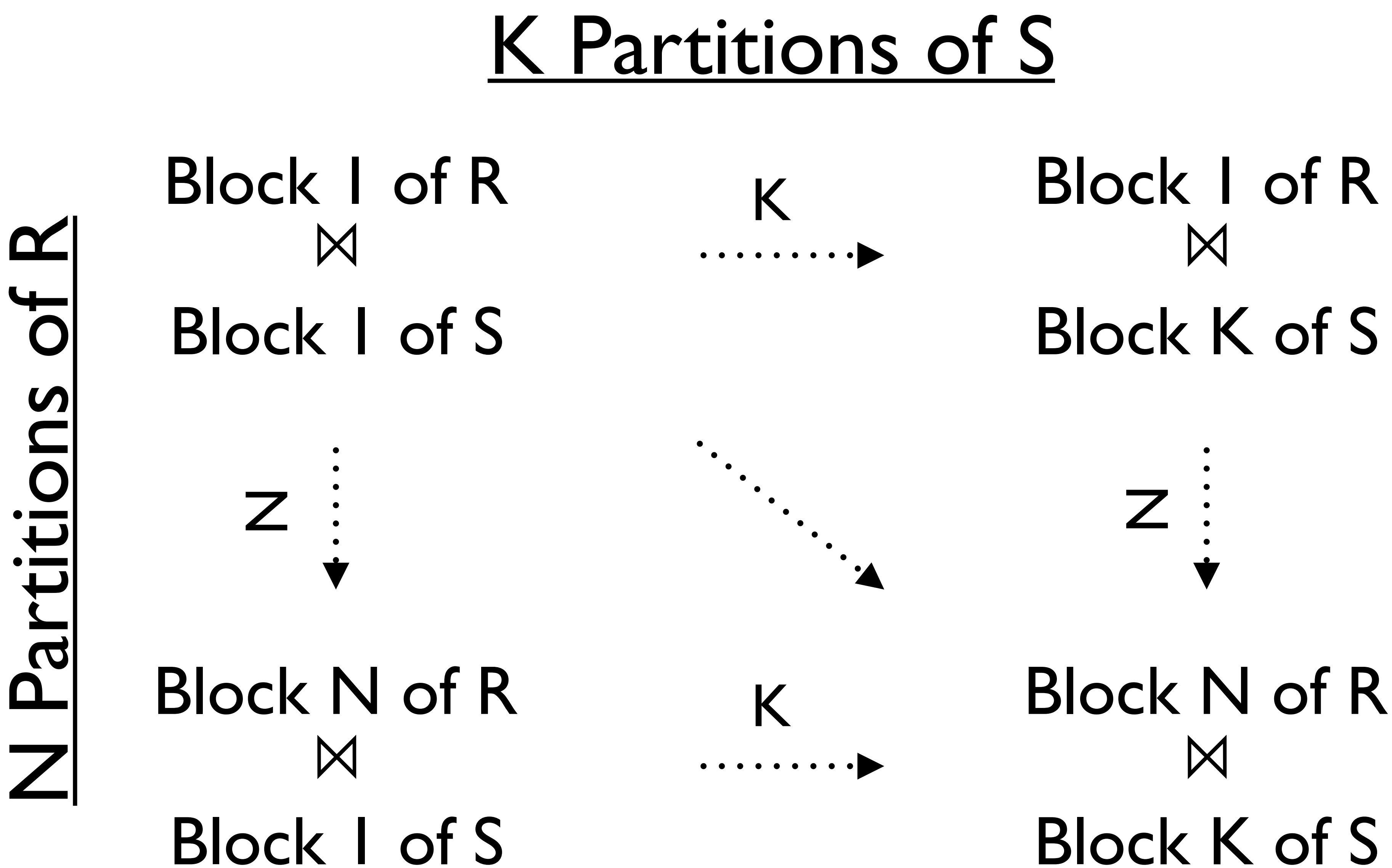
```
FOR i IN 1 to N
   FOR j IN 1 to K
      JOIN(Block i of R,
           Block j of S)
```

Partition

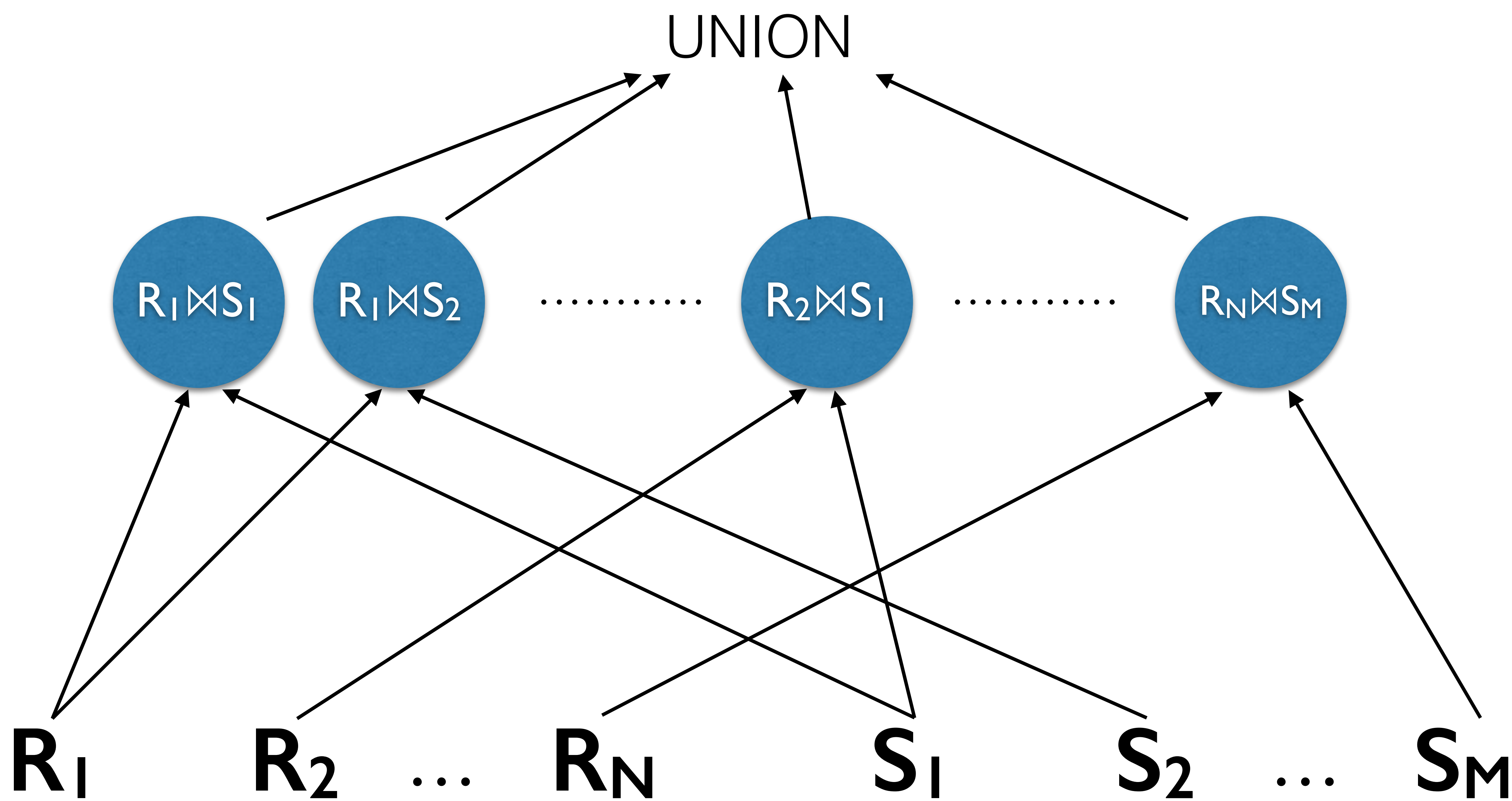
Partition

One Unit of Computation

Parallel Joins



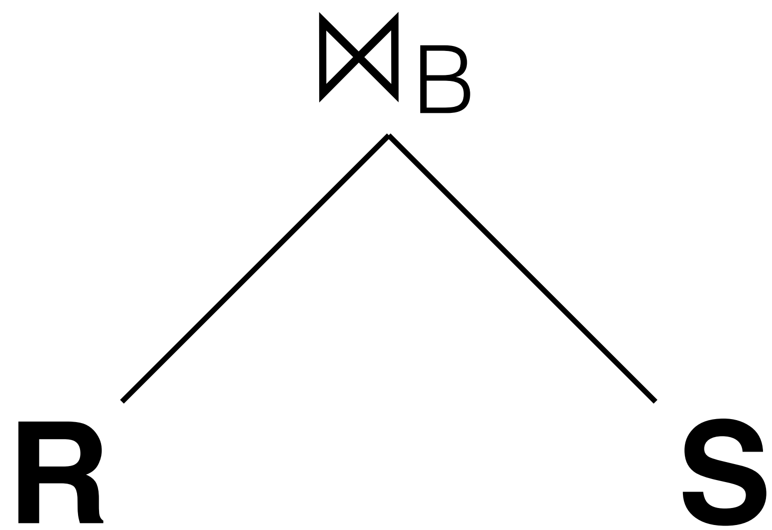
Practical Concerns



Where does the computation happen?
How does the data get there?

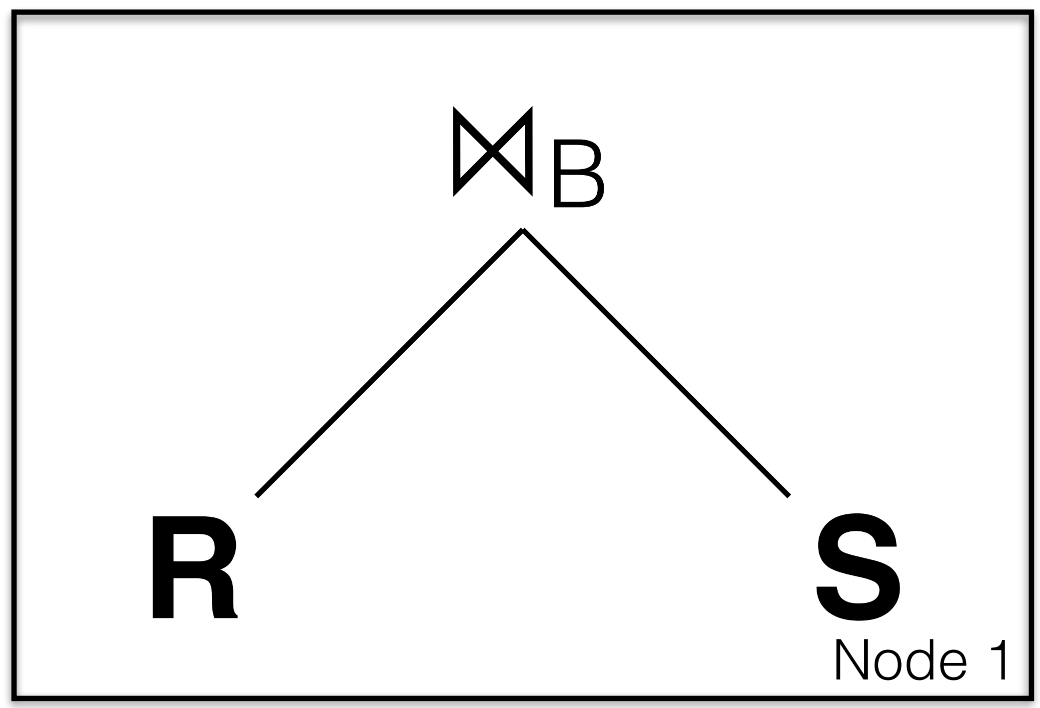
Distributing the Work

Let's start simple… what can we do with no partitions?



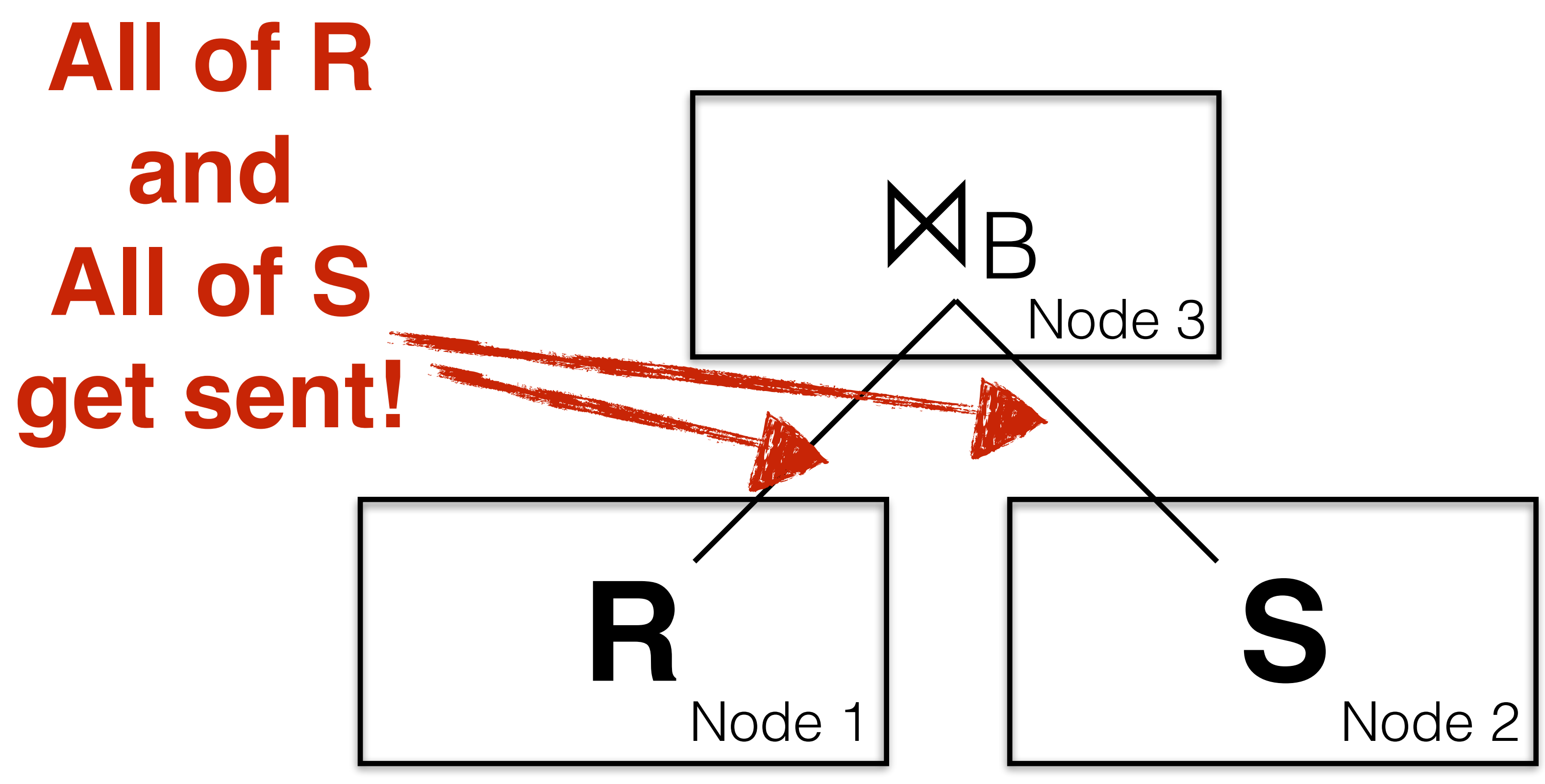
R and S may be any RA expression…

Distributing the Work



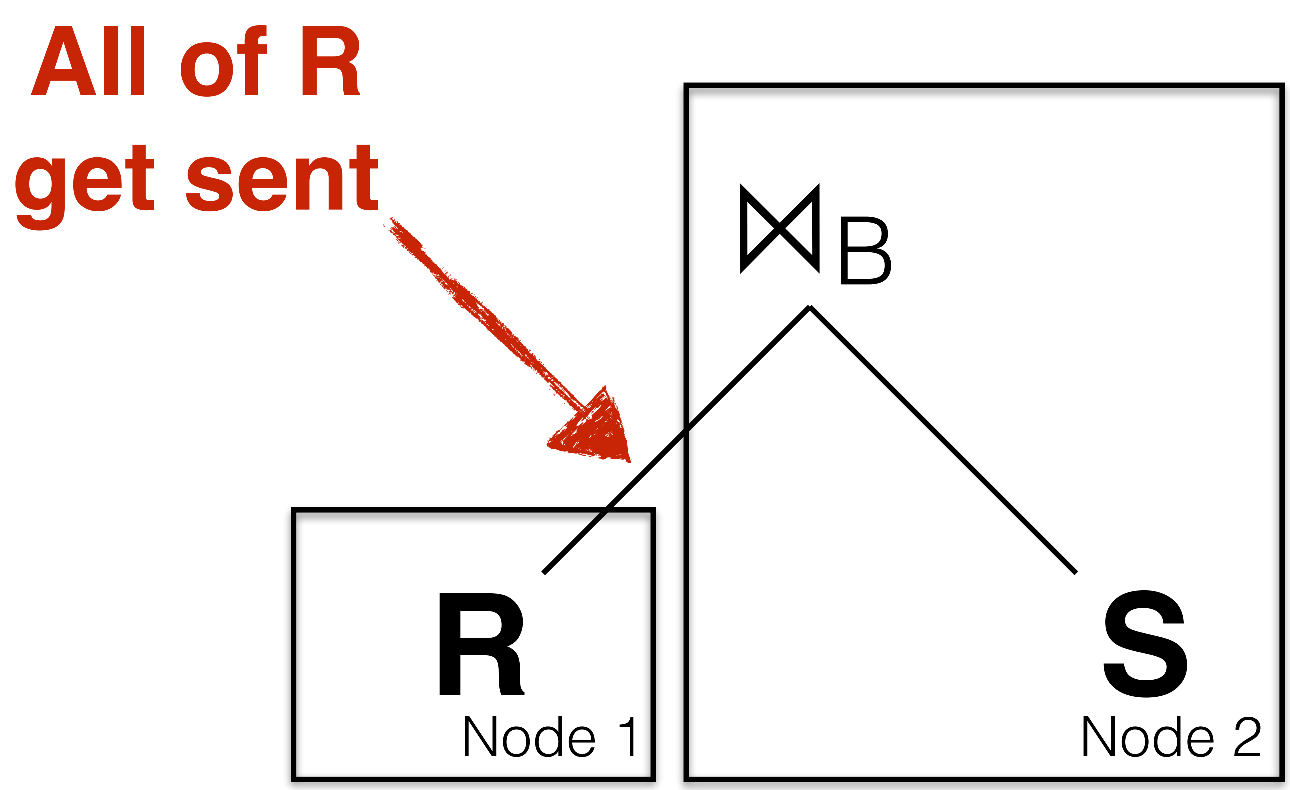
No Parallelism!

Distributing the Work



Lots of Data Transfer!

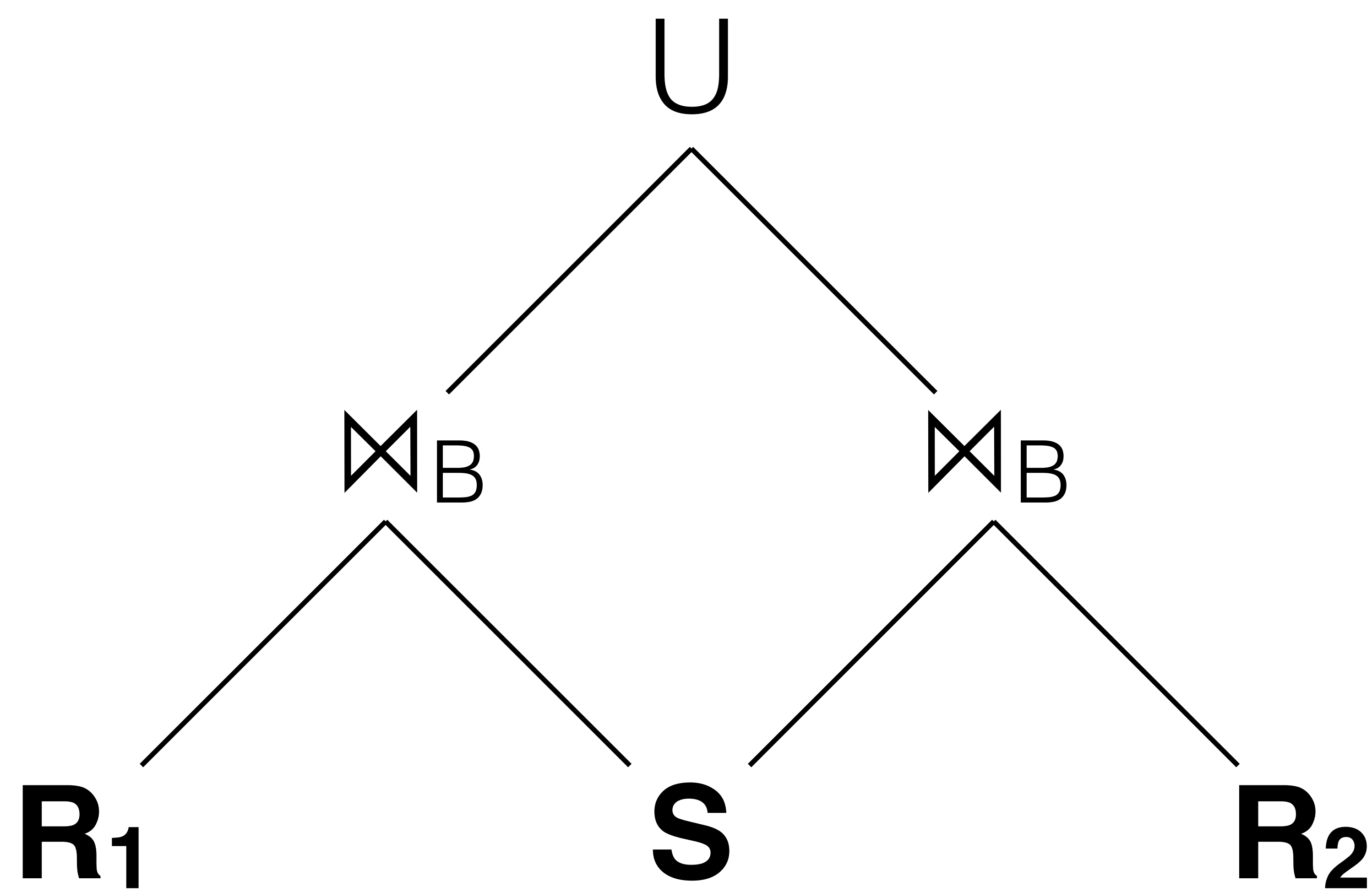
Distributing the Work



Better! We can guess whether R or S is smaller.

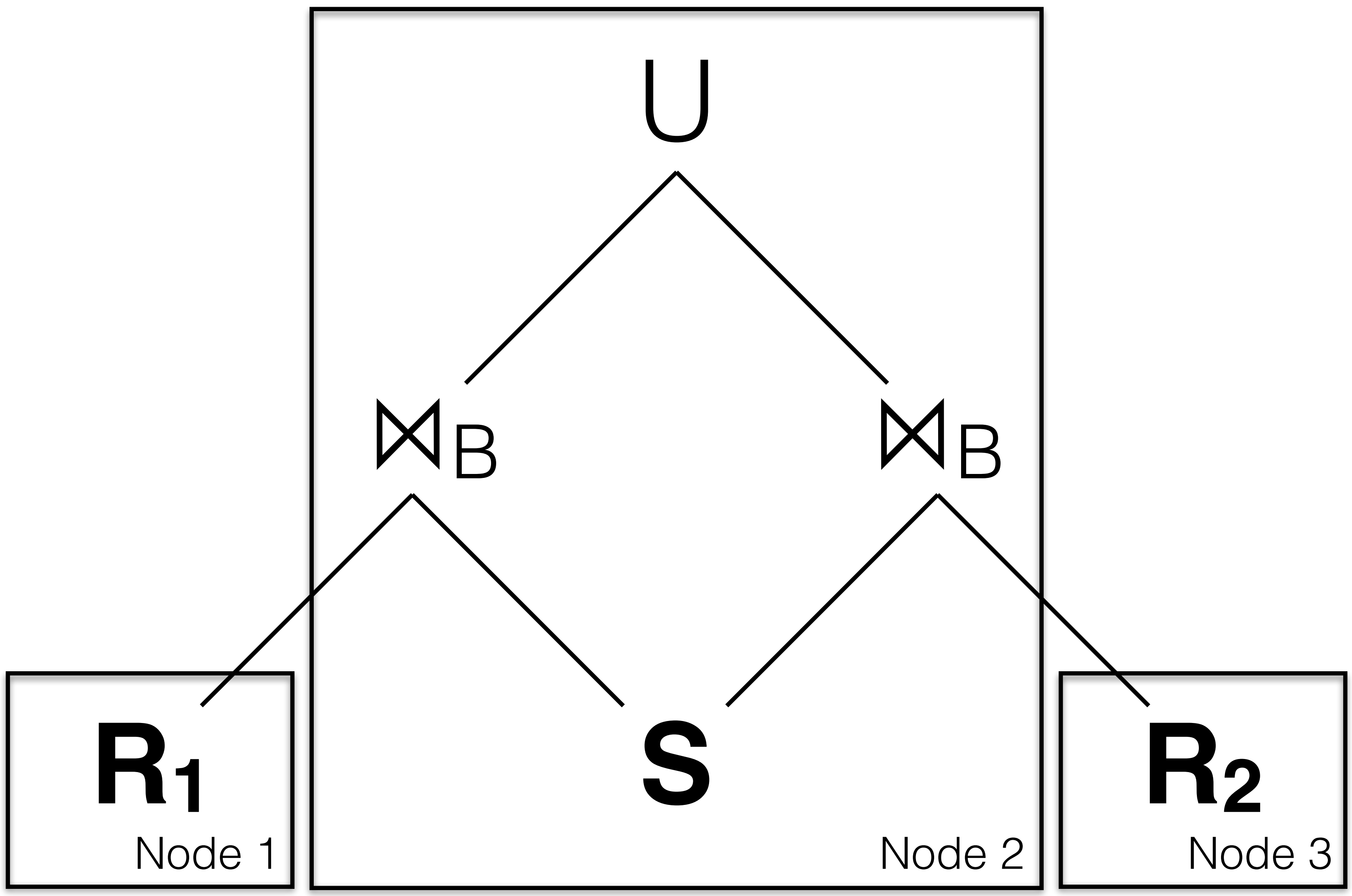
Distributing the Work

What can we do if R is partitioned?



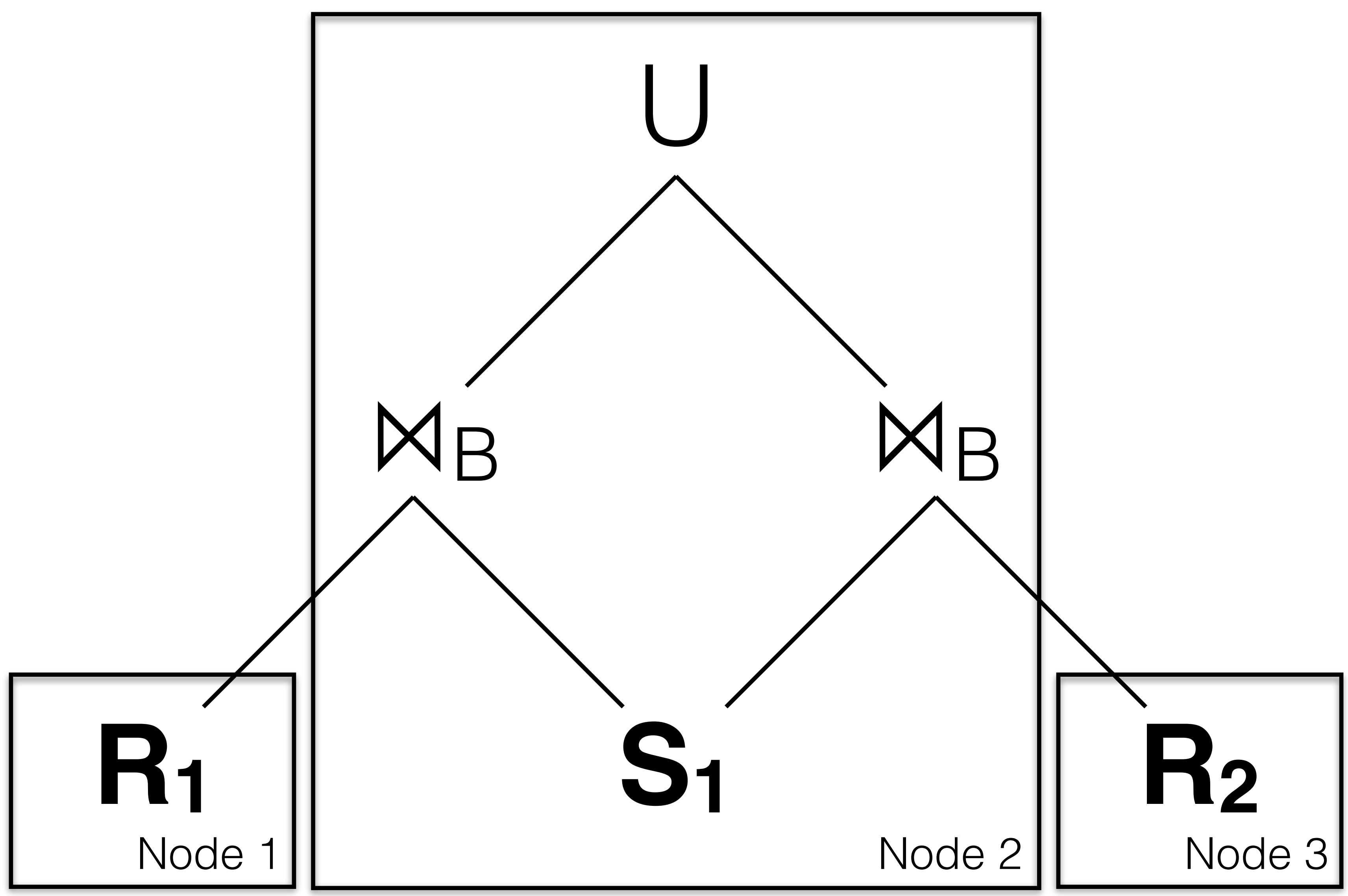
Distributing the Work

There are lots of partitioning strategies, but this one is interesting….



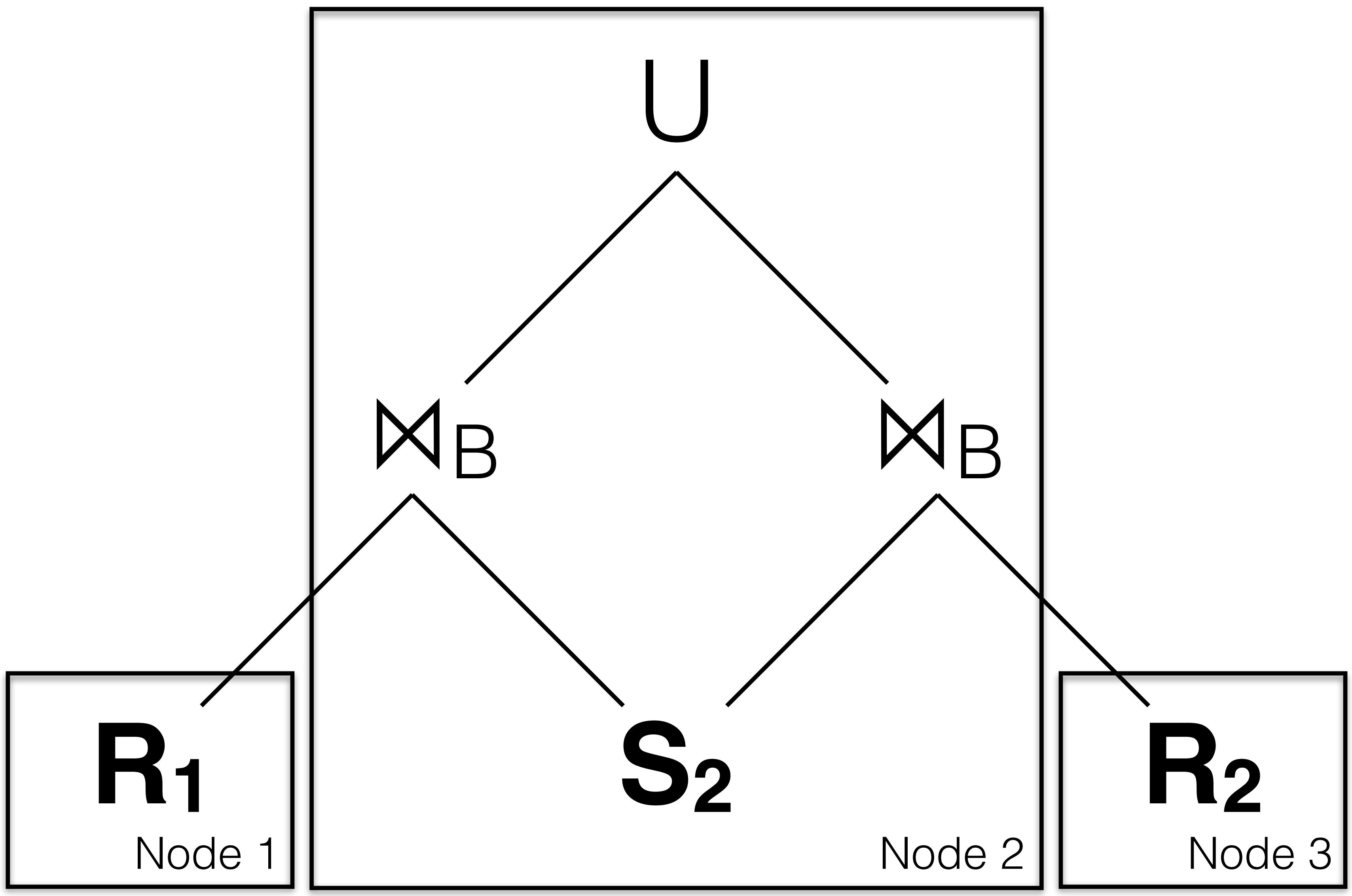
Distributing the Work

… it can be used as a model for partitioning S…



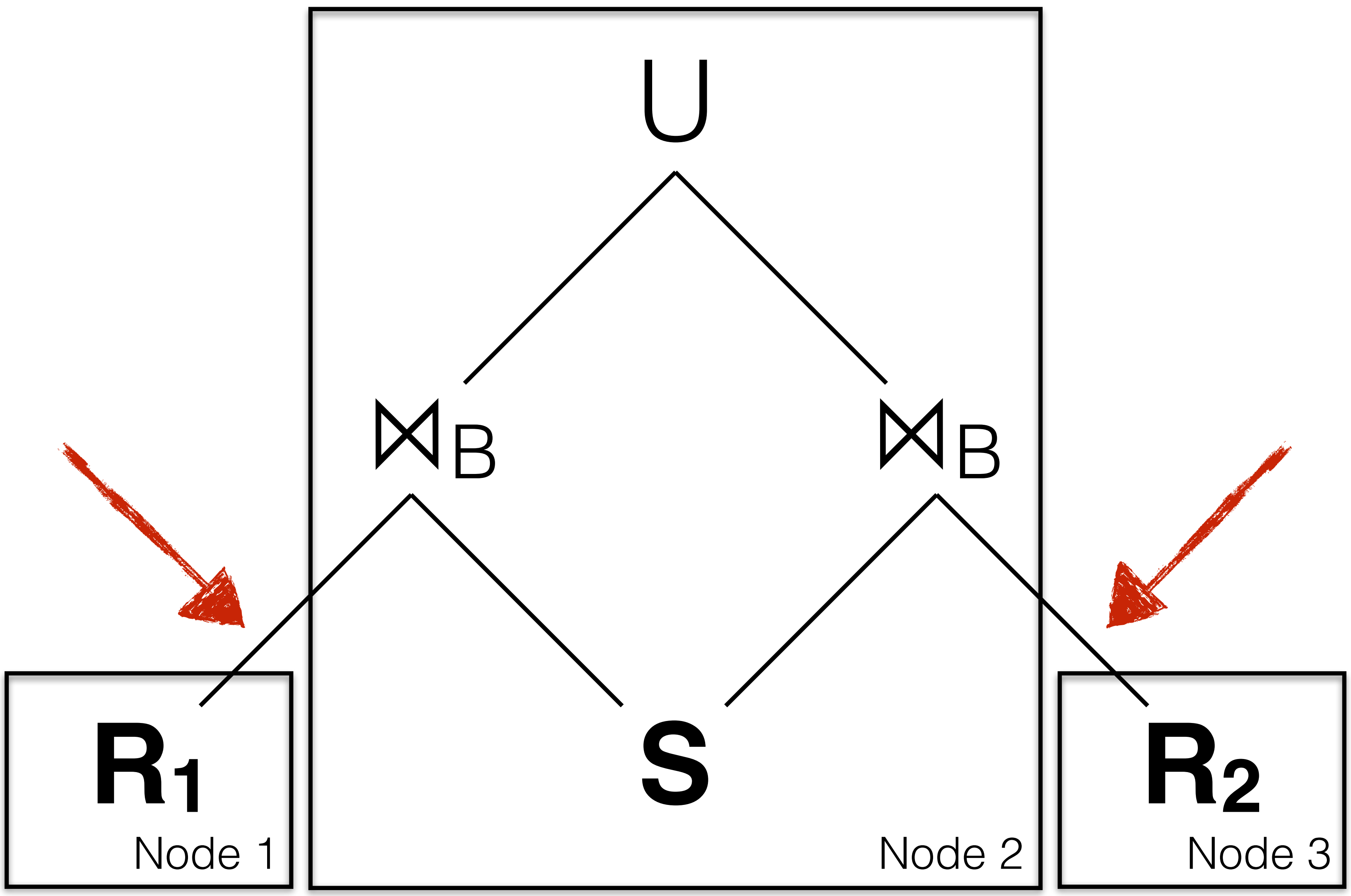
Distributing the Work

… it can be used as a model for partitioning S…

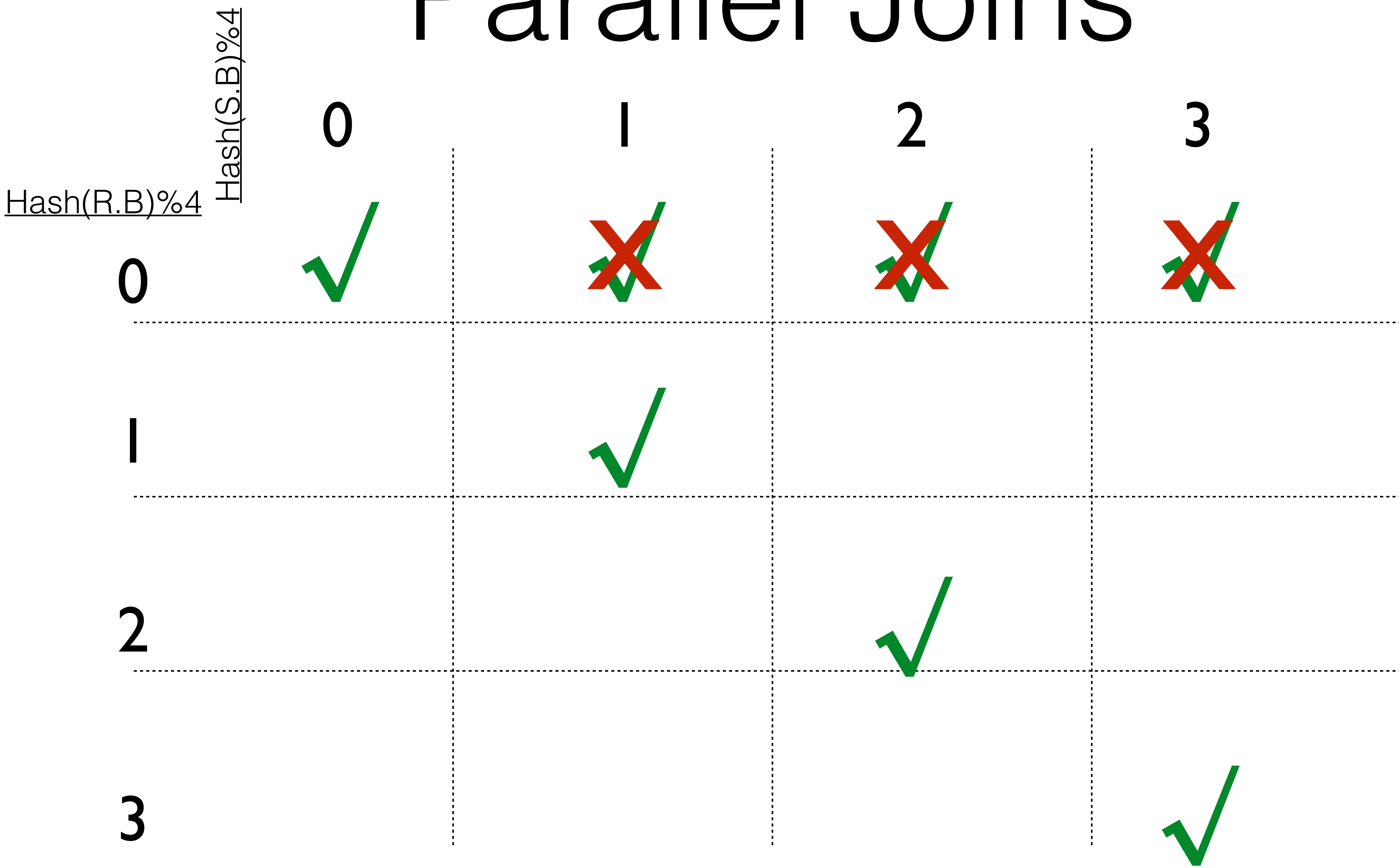


Distributing the Work

…and neatly captures the data transfer issue.



Parallel Joins



R $\bowtie_B$ **S**: Which Partitions of S Join w/ Bucket 0 of R?

Parallel Joins

R.B \ S.B	B<25	25≤B<50	50≤B<75	75≤B
B<25	✓	✓	✓	✓
25≤B<50	✗	✓	✓	✓
50≤B<75	✗	✗	✓	✓
75≤B	✗	✗	✗	✓

R $\bowtie_{R.B < S.B}$ **S**: Which Partitions of S Can Produce Output?

Can we further reduce the amount of data sent?

Sending Hints

$$R_k \bowtie_B S_i$$

The naive approach…

Node 1

R_k

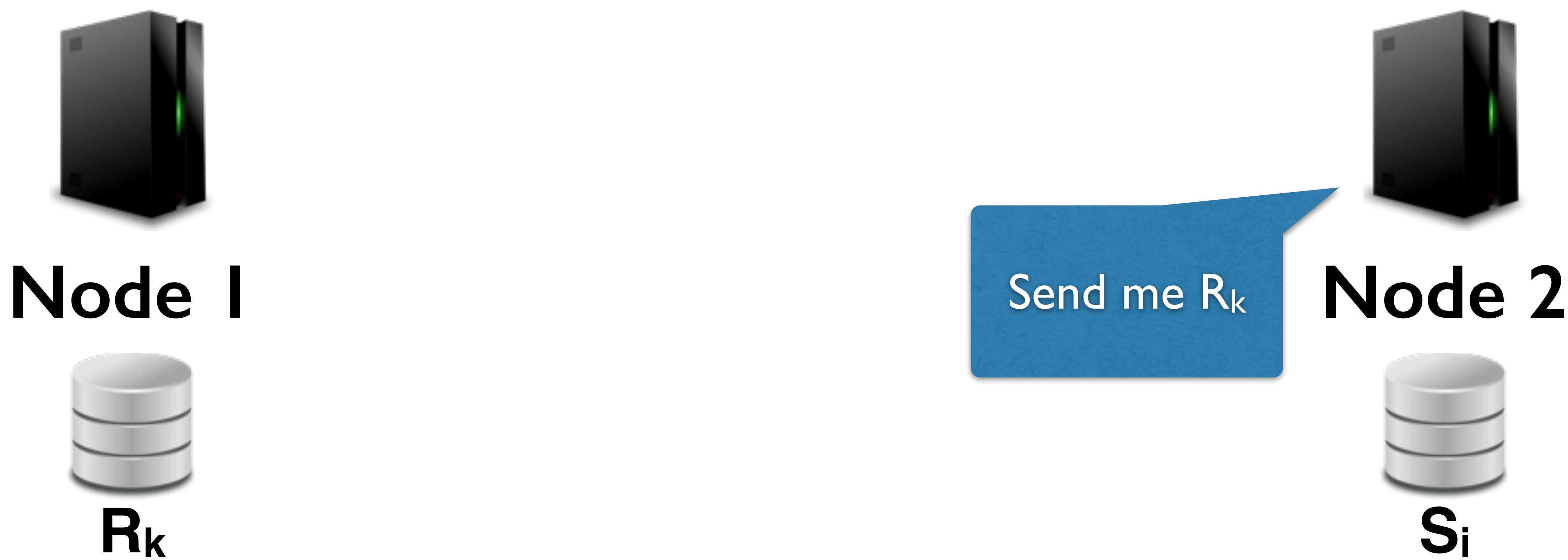
Node 2

S_i

Sending Hints

$R_k \bowtie_B S_i$

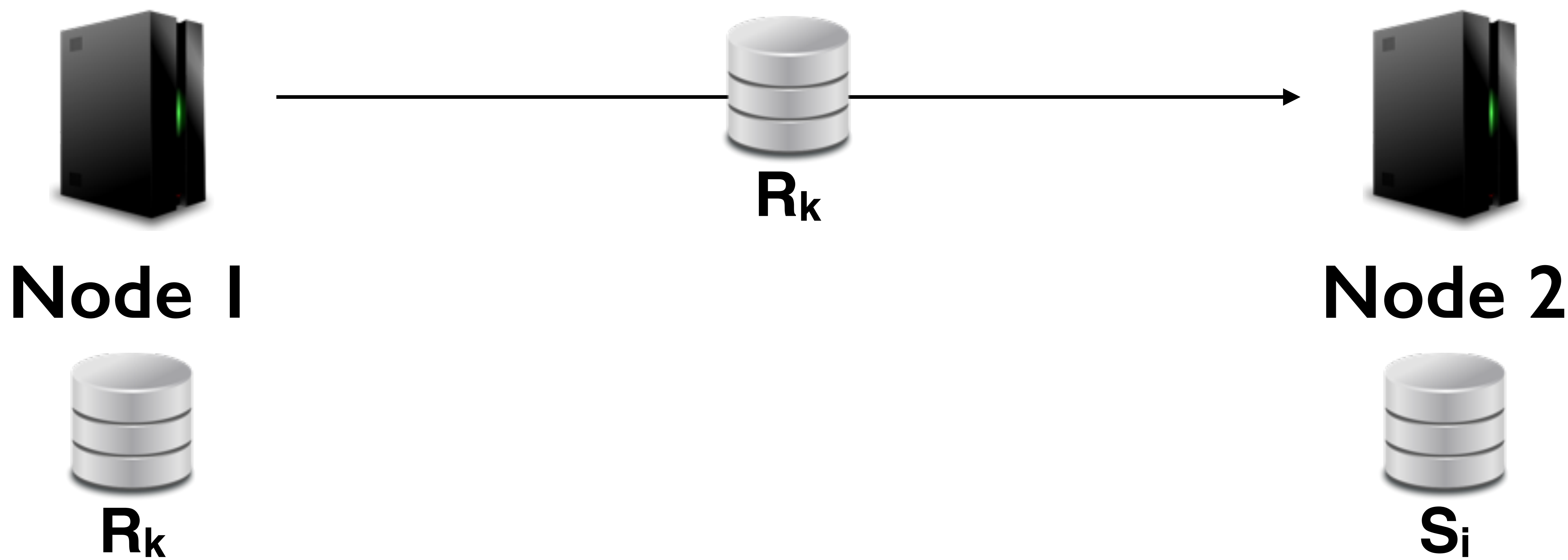
The naive approach…



Sending Hints

$\mathbf{R_k} \bowtie_B \mathbf{S_i}$

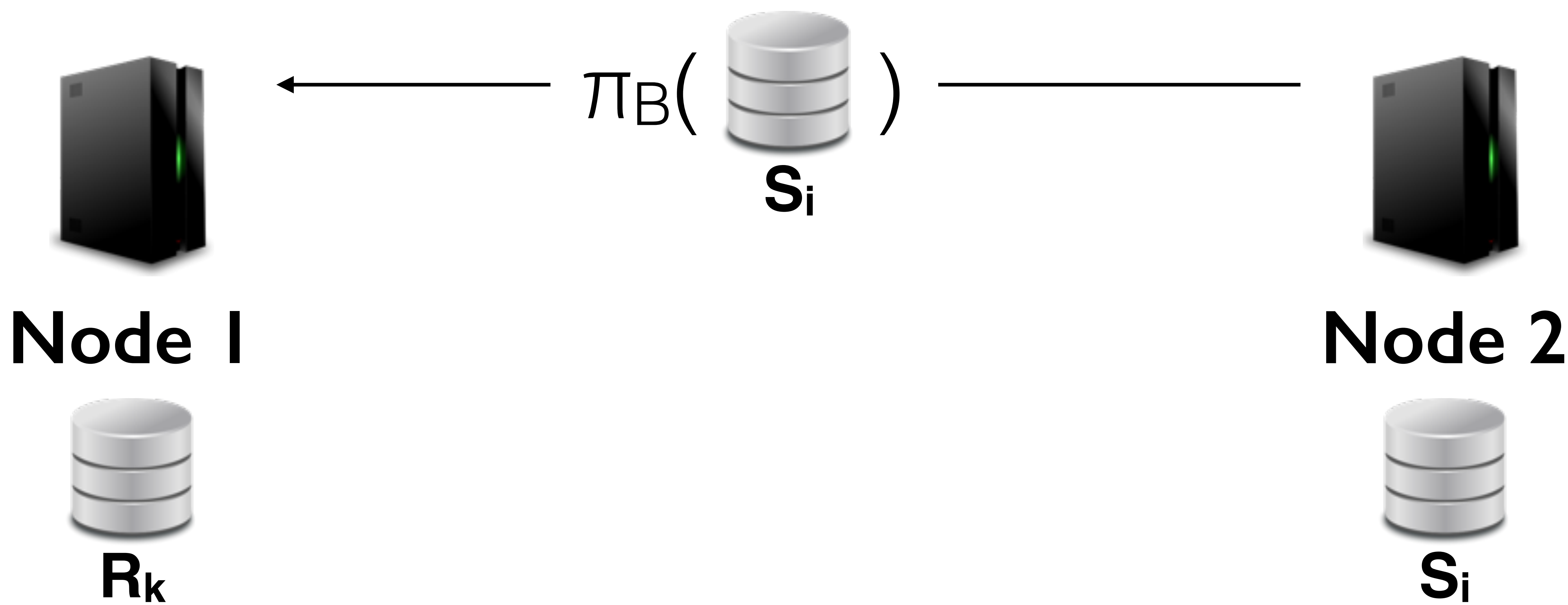
The naive approach…



Sending Hints

$$\mathbf{R_k} \bowtie_B \mathbf{S_i}$$

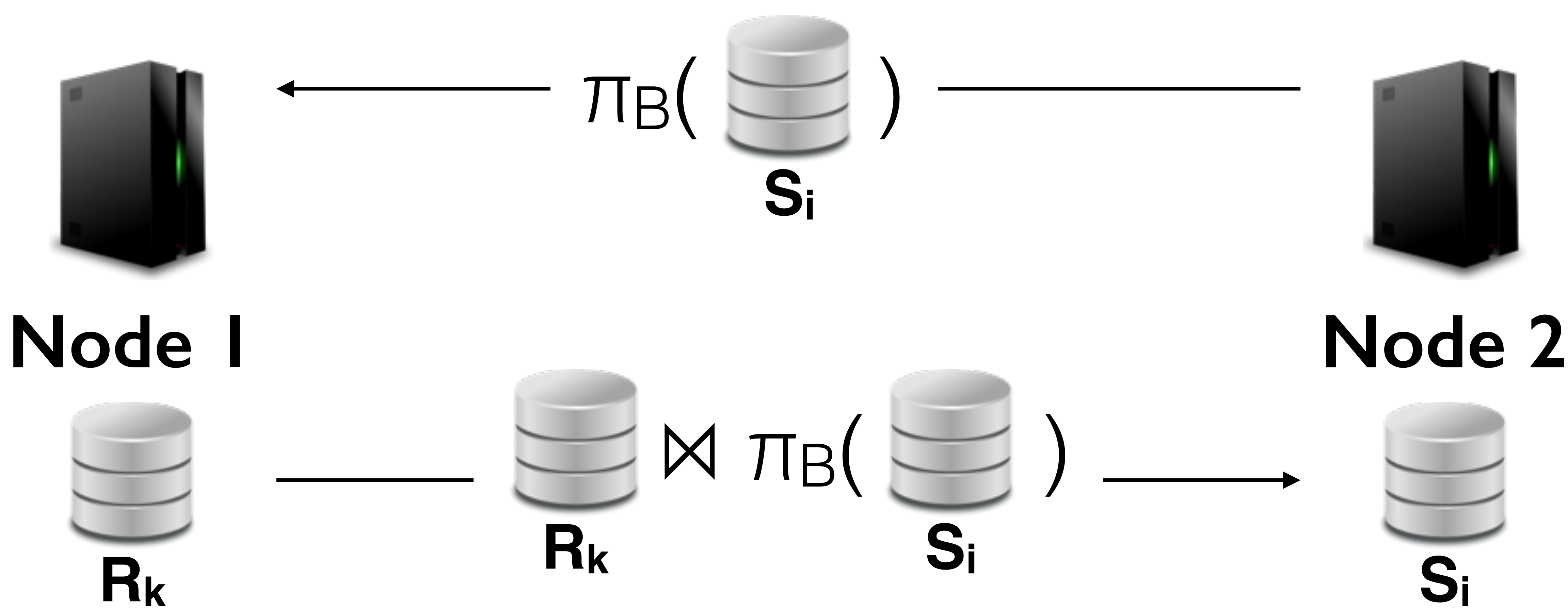
The smarter approach…



Sending Hints

$R_k \bowtie_B S_i$

The smarter approach…



Sending Hints

$$\mathbf{R_k} \bowtie_B \mathbf{S_i}$$

The smarter approach…

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Node 2

<2,X>
<3,Y>
<6,Y>

Sending Hints

$$\mathbf{R_k} \bowtie_B \mathbf{S_i}$$

The smarter approach…

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Node 2

<2,X>
<3,Y>
<6,Y>

Send me rows with a 'B' of 2,3, or 6

Sending Hints

$\mathbf{R_k} \bowtie_B \mathbf{S_i}$

The smarter approach…

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

<2,B>
<2,C>
<3,D>

Send me rows with a 'B' of 2,3, or 6

Node 2

<2,X>
<3,Y>
<6,Y>

This is called a <u>semi-join</u>.

Sending Hints

Now Node 1 sends as little data as possible…

… but Node 2 needs to send a lot of data.

Can we do better?

Sending Hints

$R_k \bowtie_B S_i$

Strategy 1: Parity Bits

Node 1

<1,A> 1
<2,B> 0
<2,C> 0
<3,D> 1
<4,E> 0

Node 2

0 <2,X>
0 <6,Y>

Sending Hints

$\mathbf{R_k} \bowtie_B \mathbf{S_i}$

Strategy 1: Parity Bits

Node 1

<1,A> 1
<2,B> 0
<2,C> 0
<3,D> 1
<4,E> 0

Node 2

0 <2,X>
0 <6,Y>

Send me data with a parity bit of ‘0’

Sending Hints

$R_k \bowtie_B S_i$

Strategy 1: Parity Bit

Node 1 sending too much is ok!
(Node 2 still needs to compute $\bowtie_B$)

Node 1

<1,A> 1
<2,B> 0
<2,C> 0
<3,D> 1
<4,E> 0

<2,B>
<2,C>
<4,E>

Send me data with a parity bit of '0'

Node 2

0 <2,X>
0 <6,Y>

Problem: <u>One</u> parity bit is too little

Sending Hints

$\mathbf{R_k} \bowtie_B \mathbf{S_i}$

Strategy 1: Parity Bit

Node 1

<1,A> 1
<2,B> 0
<2,C> 0
<3,D> 1
<4,E> 0

Node 2

0 <2,X>
1 <3,Y>
0 <6,Y>

Problem: <u>One</u> parity bit is too little

Sending Hints

$R_k \bowtie_B S_i$

Strategy 2: Parity Bits

Node 1

<1,A> 01
<2,B> 10
<2,C> 10
<3,D> 11
<4,E> 00

<2,B>
<2,C>
<3,D>

Send me data with parity bits 10 or 11

Node 2

10 <2,X>
11 <3,Y>
10 <6,Y>

Problem: Almost as much data as π_B

Sending Hints

Can we summarize the parity bits?

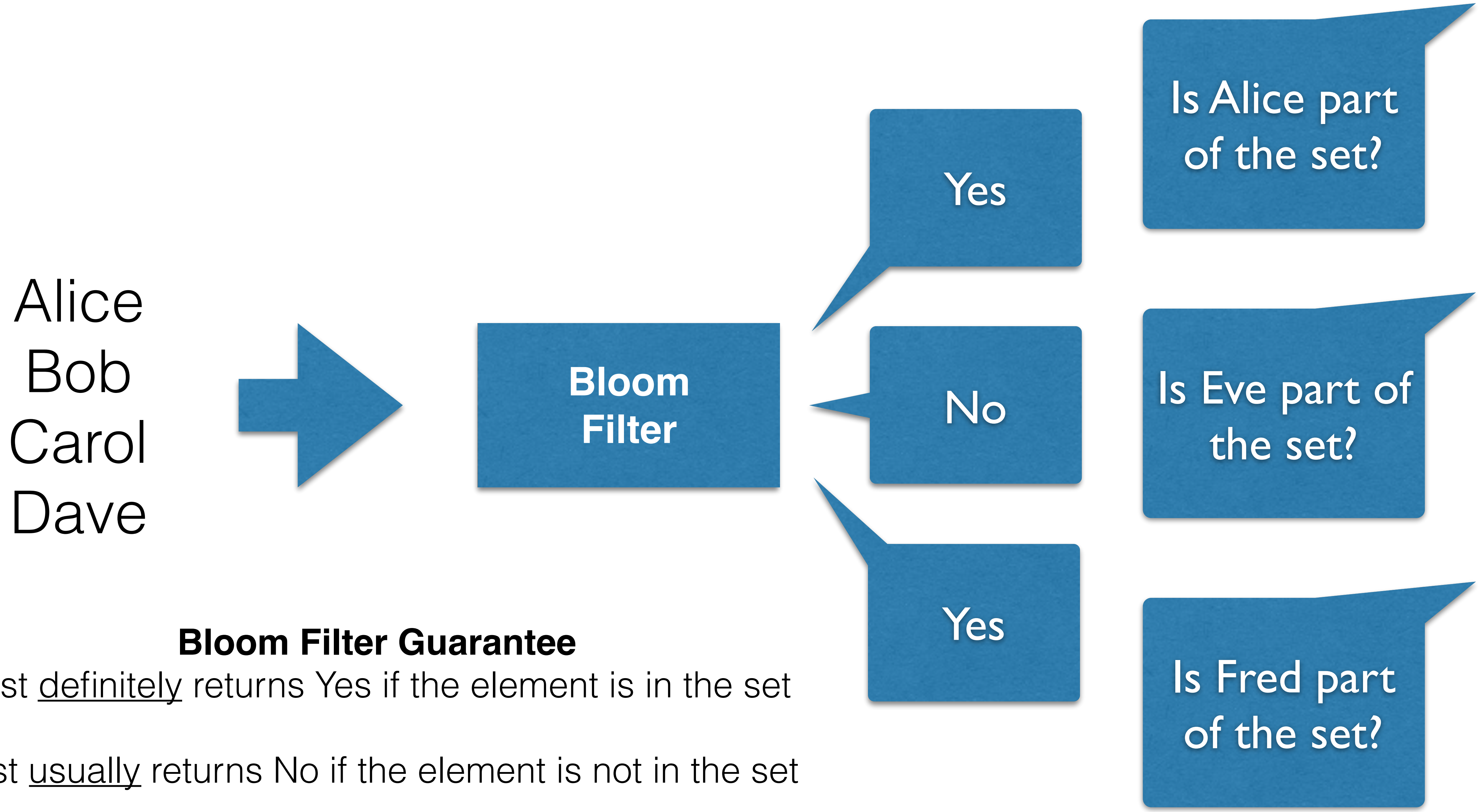
Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filters



Bloom Filter Guarantee

Test definitely returns Yes if the element is in the set

Test usually returns No if the element is not in the set

Bloom Filters

A Bloom Filter is a bit vector

M - # of bits in the bit vector

K - # of hash functions

For ONE key (or record):
  For i between 0 and K:
    bitvector[$hash_i$ (key) % M] = 1

Each bit vector has ~K bits set

Bloom Filters

Key 1 00101010

Key 2 01010110

Key 3 10000110

Key 4 01001100

Filters are combined by Bitwise-OR

e.g. (Key 1 | Key 2)

= 01111110

How do we test for inclusion?

(Key & Filter) == Key?

(Key 1 & S) = 00101010 √

(Key 3 & S) = 00000110 X

(Key 4 & S) = 01001100 √

False Positive

Sending Hints

$\mathbf{R_k} \ltimes_B \mathbf{S_i}$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Node 2

<2,X>
<3,Y>
<6,Y>

Sending Hints

$$\mathbf{R_k} \bowtie_B \mathbf{S_i}$$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Send me rows with a 'B' in the bloom filter summarizing the set {2,3,6}

Node 2

<2,X>
<3,Y>
<6,Y>

Sending Hints

$R_k \bowtie_B S_i$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

<2,B>
<2,C>
<3,D>
<4,E>

Send me rows with a ‘B’ in the bloom filter summarizing the set {2,3,6}

Node 2

<2,X>
<3,Y>
<6,Y>

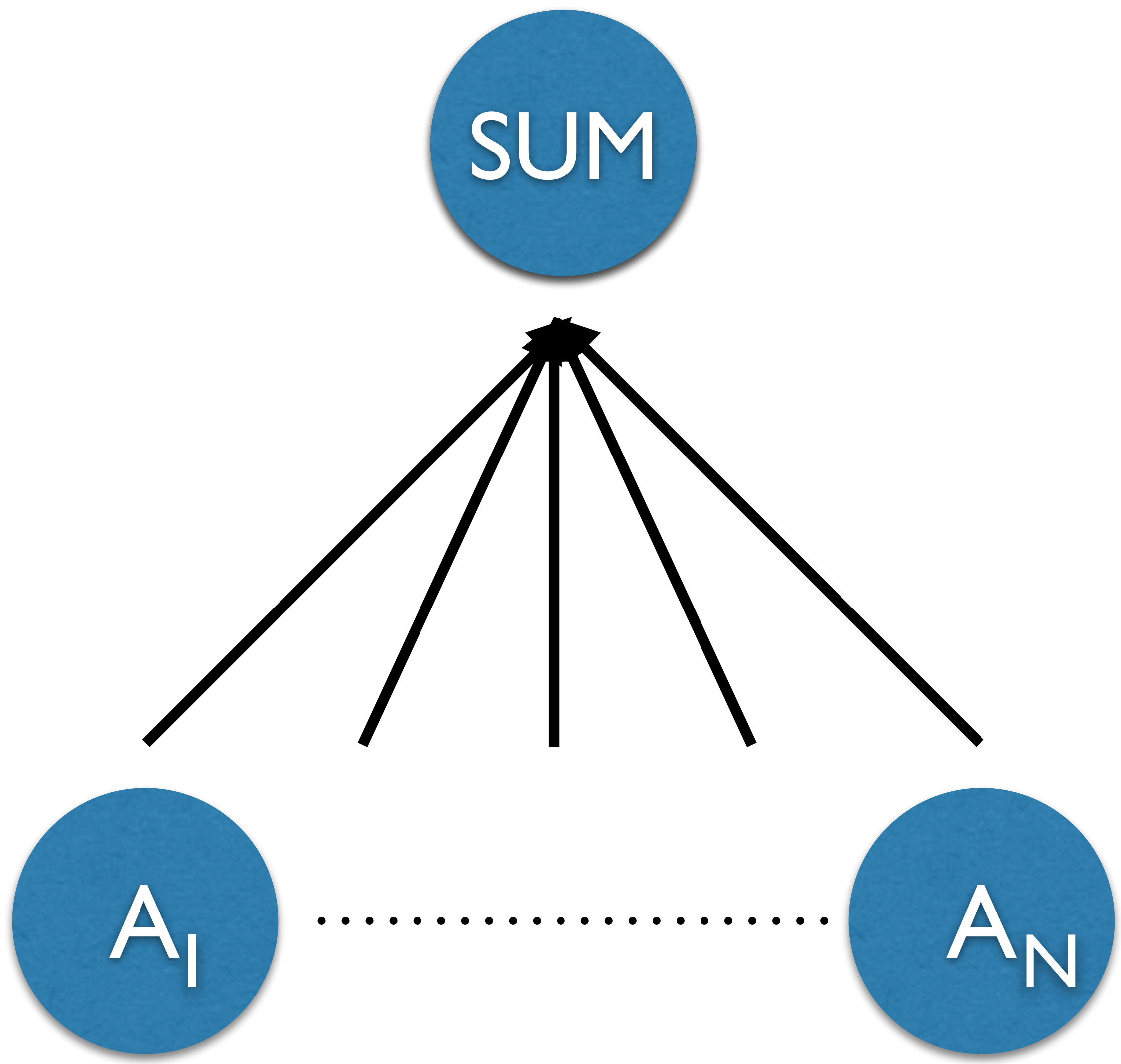
This is called a <u>bloom-join</u>.

Parallel Aggregates

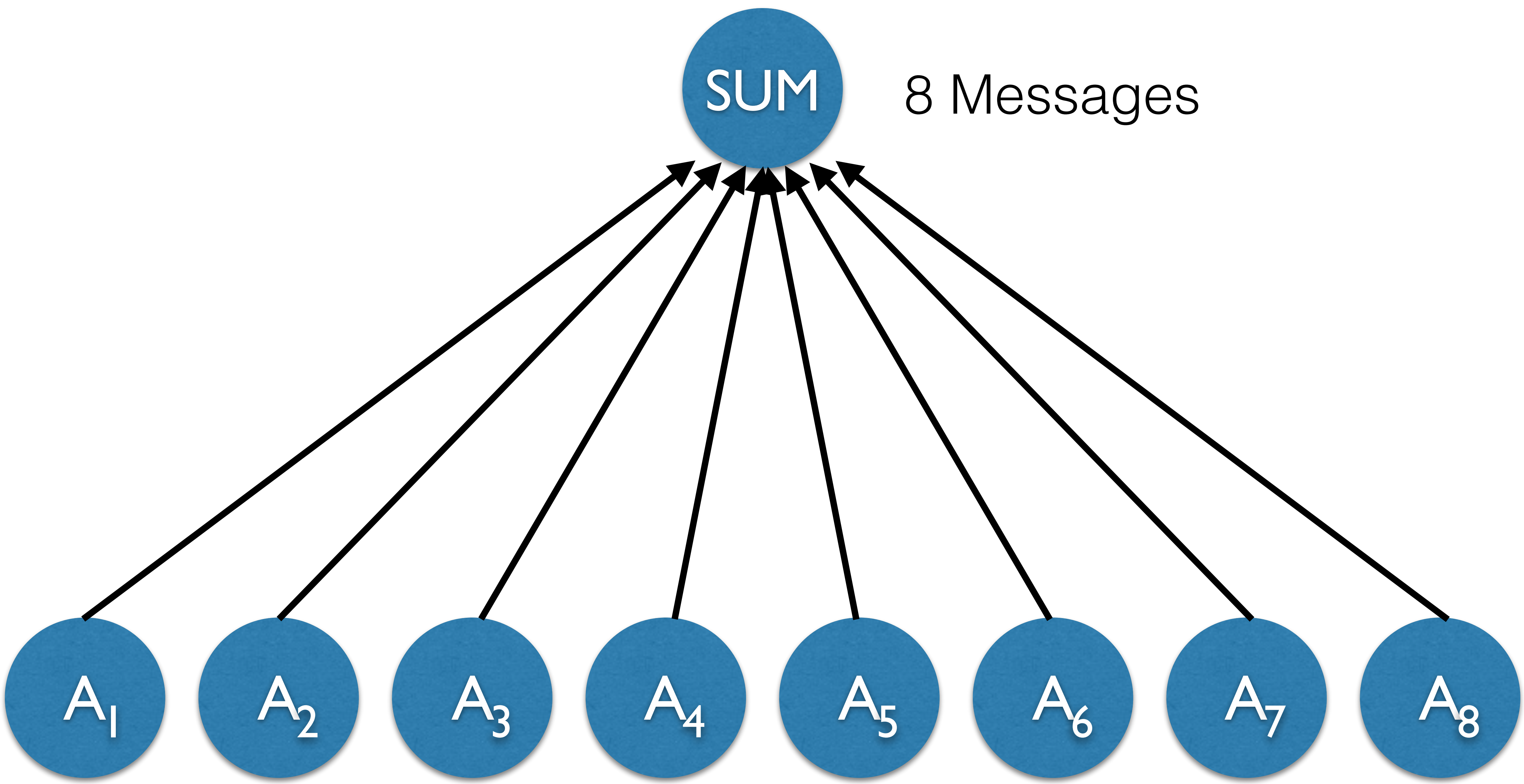
Algebraic: Bounded-size intermediate state
(Sum, Count, Avg, Min, Max)

Holistic: Unbounded-size intermediate state
(Median, Mode/Top-K Count, Count-Distinct;
Not Distribution-Friendly)

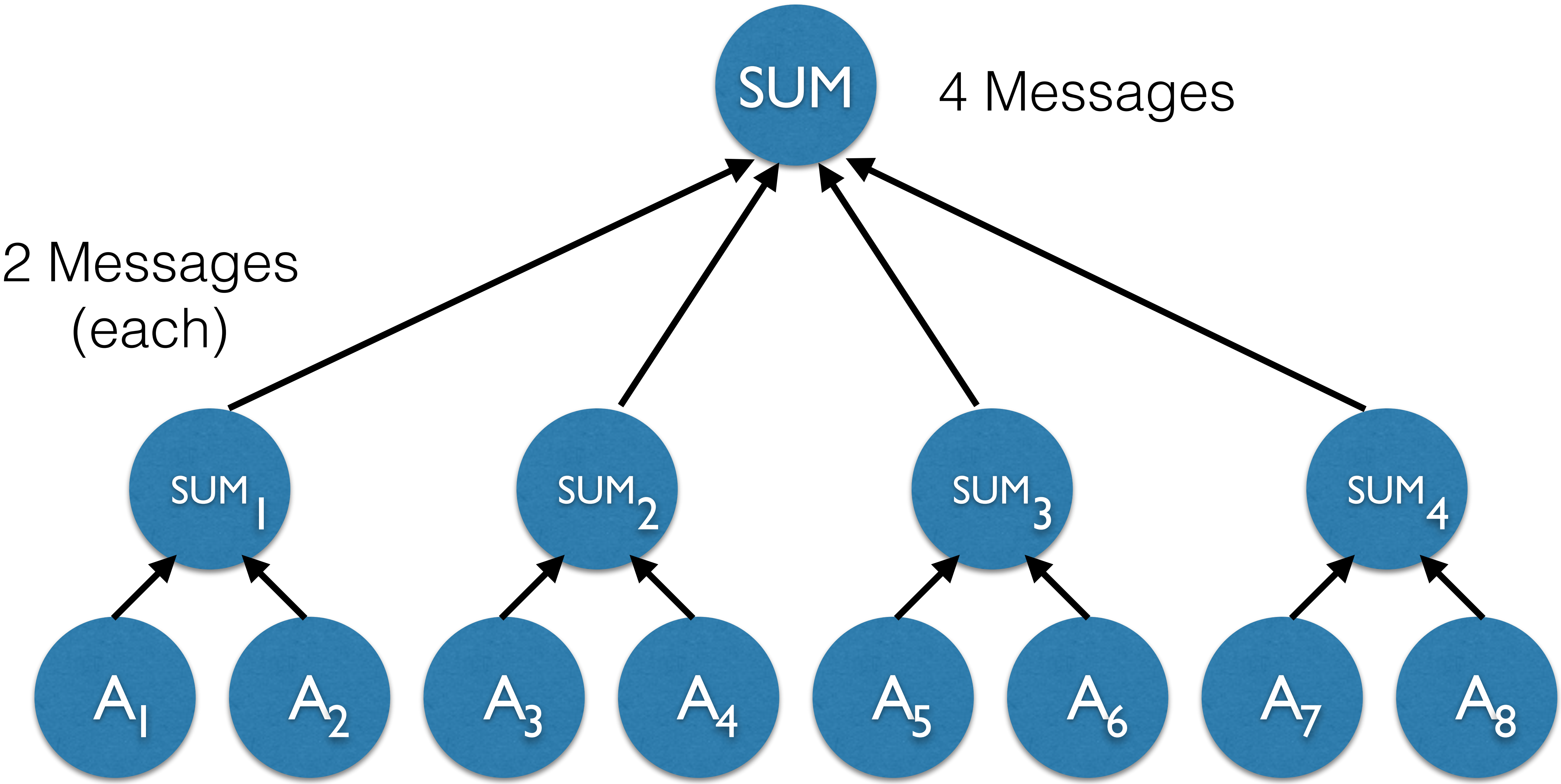
Fan-In Aggregation



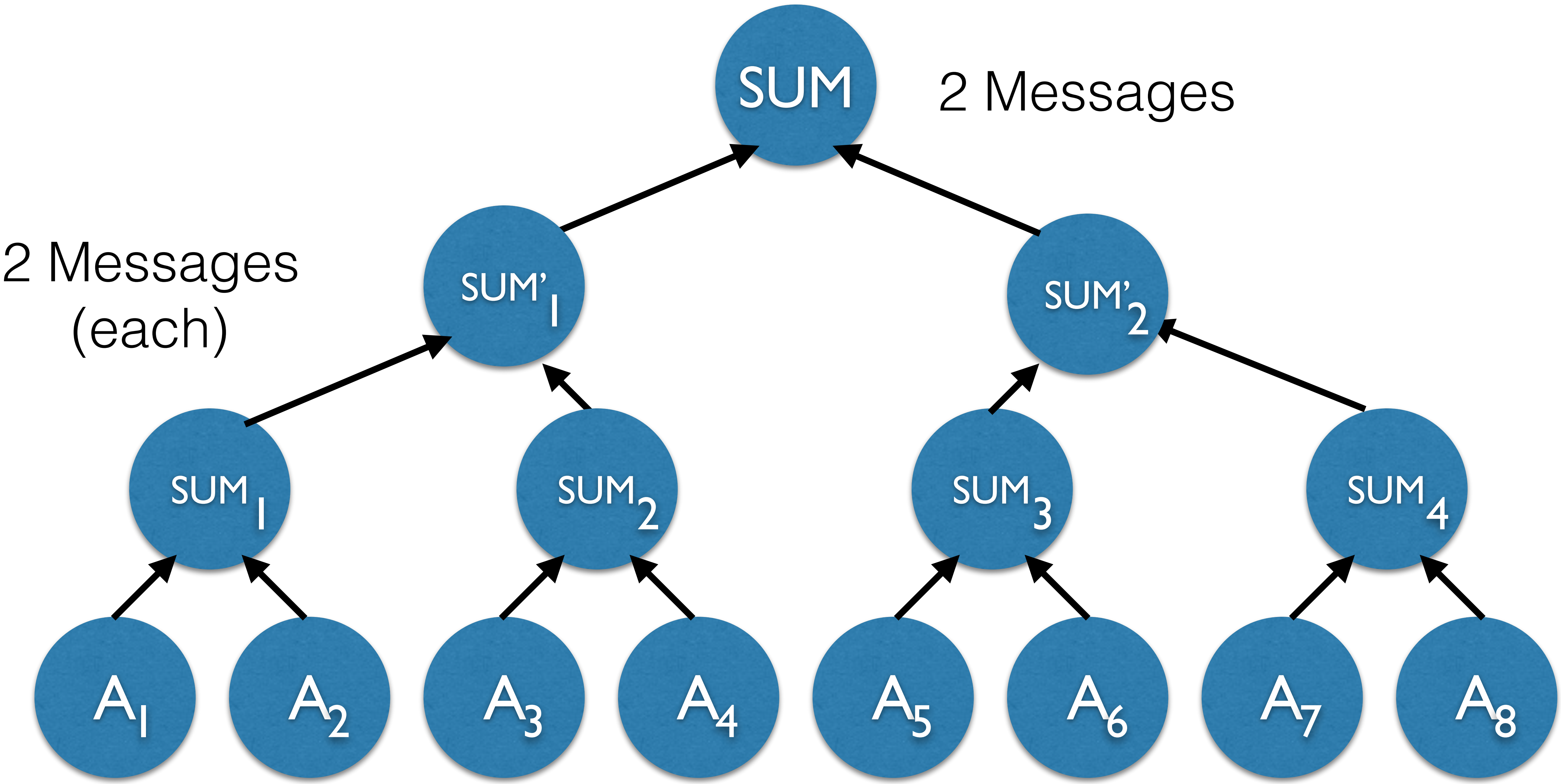
Fan-In Aggregation



Fan-In Aggregation



Fan-In Aggregation



Fan-In Aggregation

If Each Node Performs K Units of Work…
(K Messages)
How Many Rounds of Computation Are Needed?

$Log_K(N)$

Fan-In Aggregation Components

Combine(Intermediate_1, …, Intermediate_N)
= Intermediate

$$<SUM_1, COUNT_1> \otimes \ldots \otimes <SUM_N, COUNT_N>$$
$$= <SUM_1+\ldots+SUM_N, COUNT_1+\ldots+COUNT_N>$$

Compute(Intermediate) = Aggregate

`Compute(<SUM, COUNT>) = SUM / COUNT`