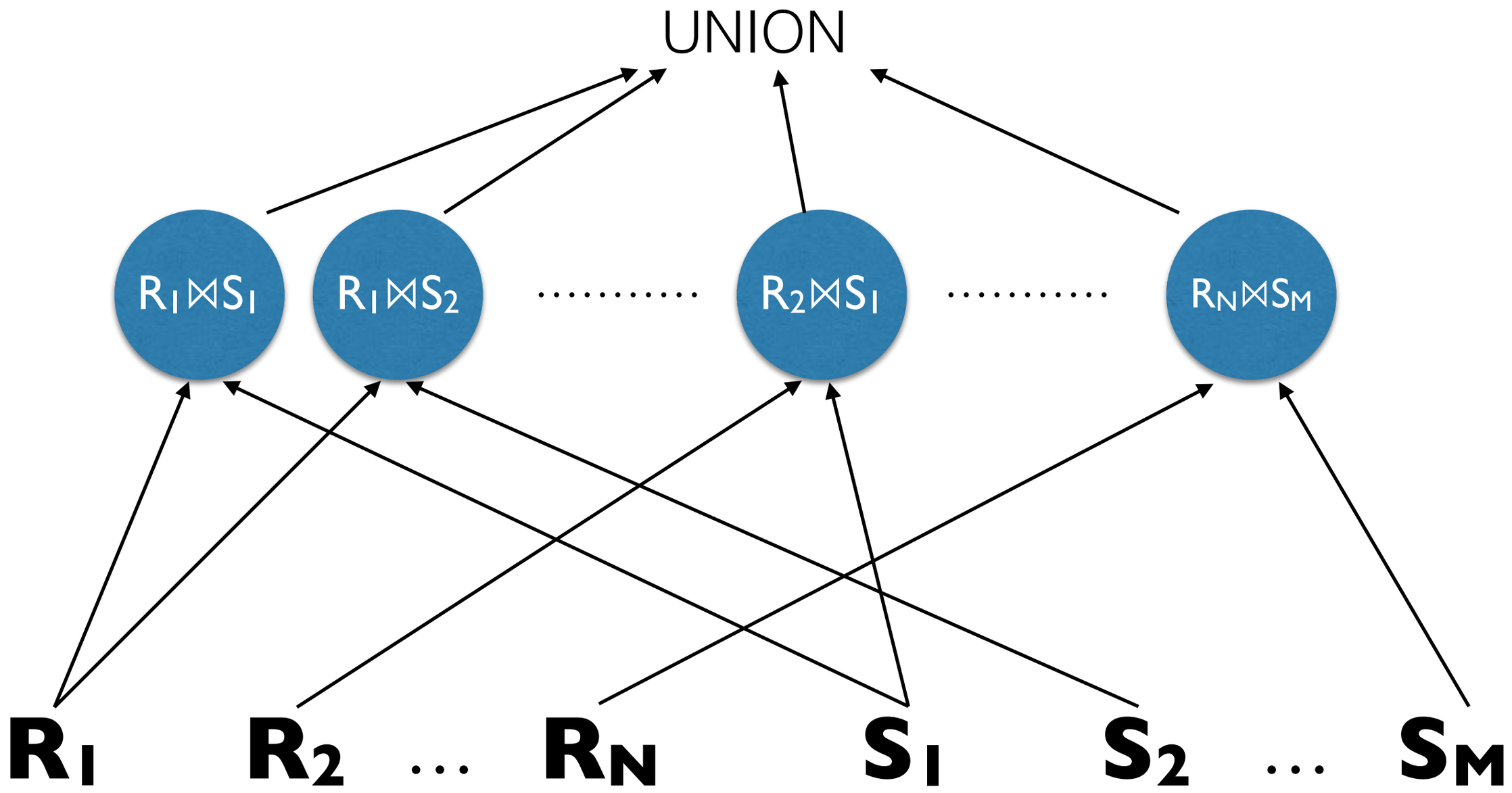
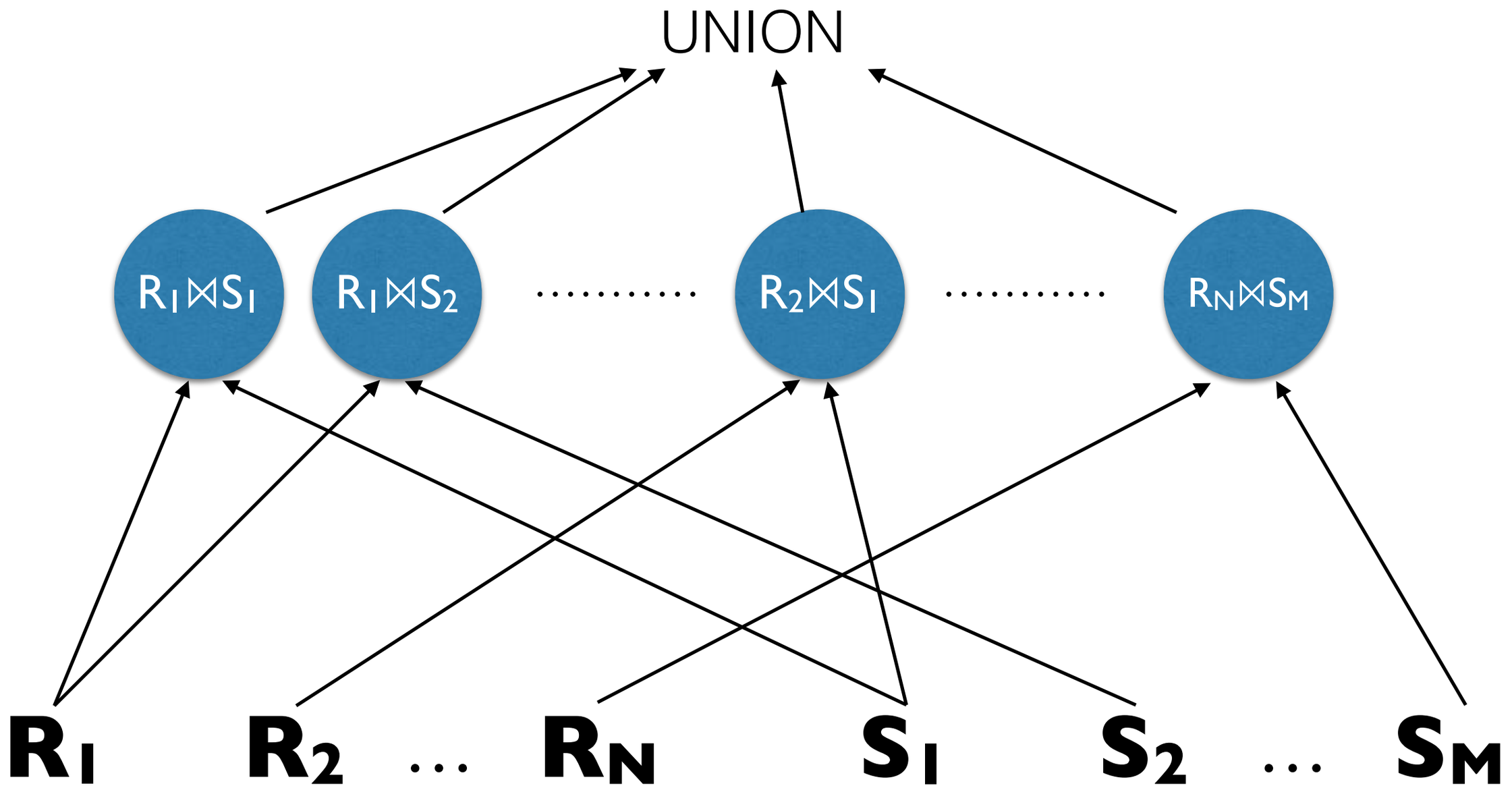
Semi-Joins and Bloom Join

Databases: The Complete Book
Ch 20

Practical Concerns



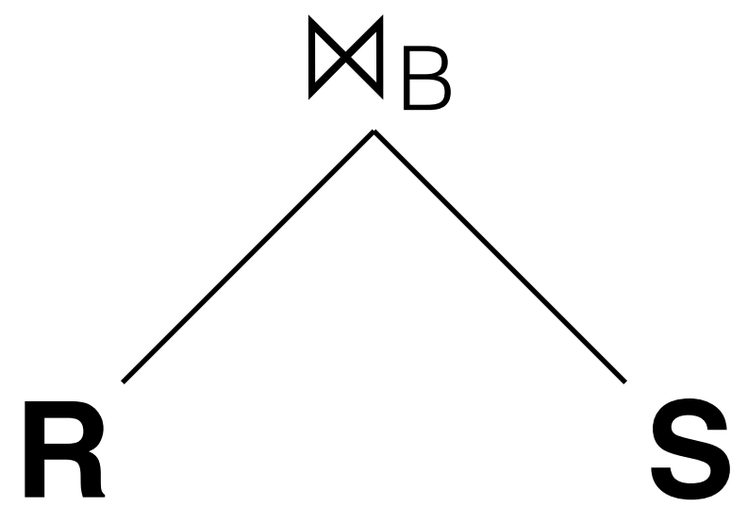
Practical Concerns



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

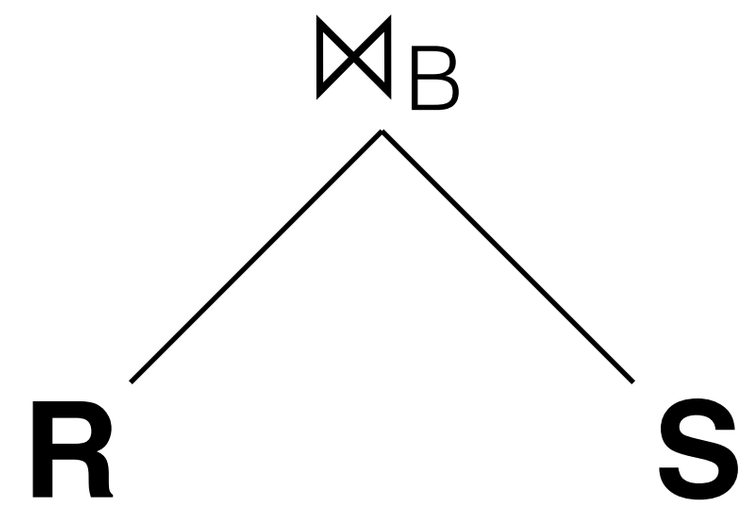
Distributing the Work

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



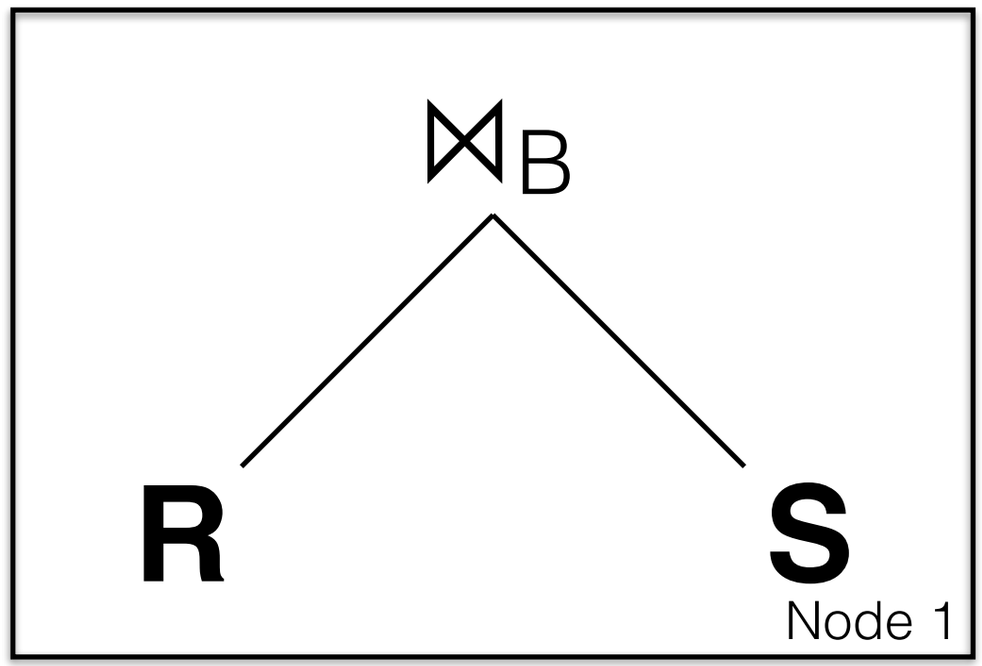
Distributing the Work

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

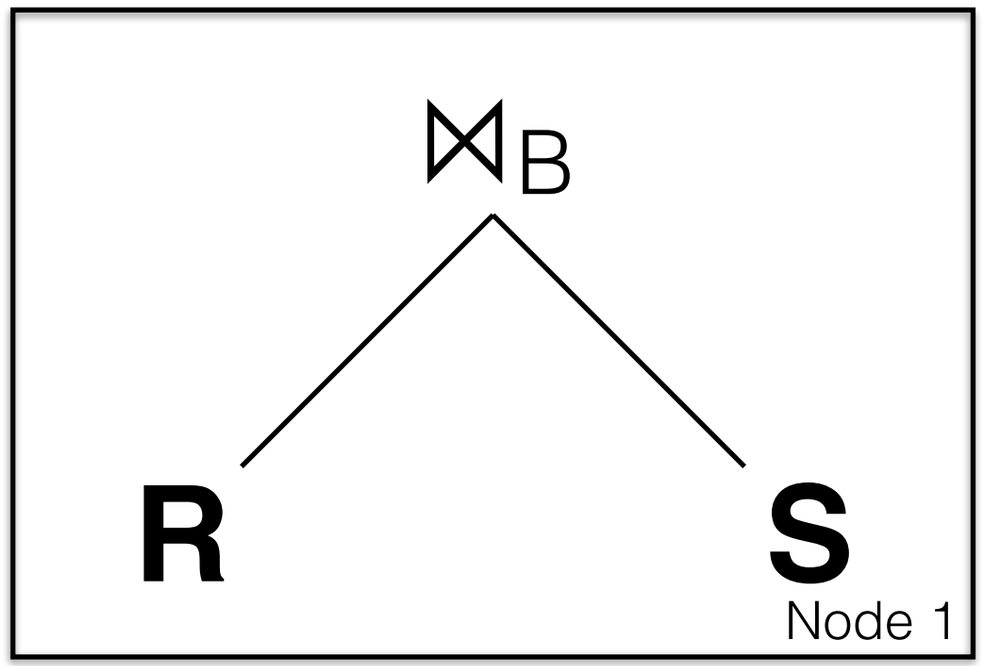


R and S may be any RA expression…

Distributing the Work

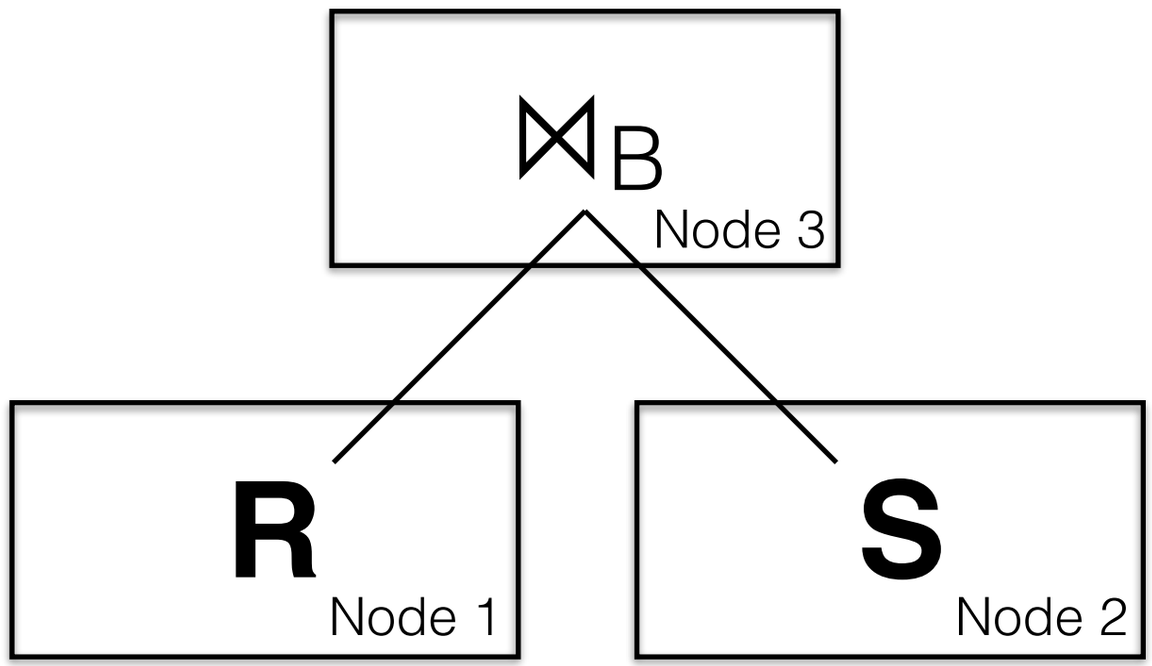


Distributing the Work

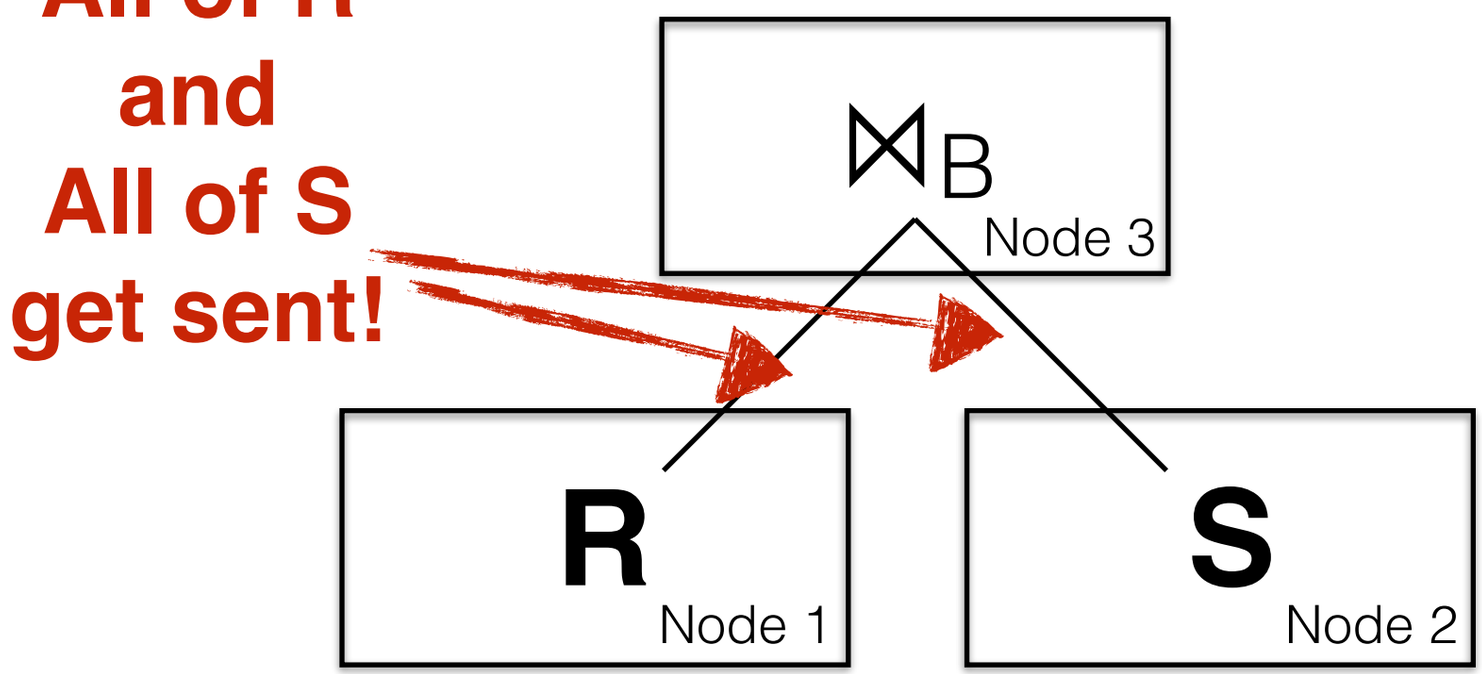


No Parallelism!

Distributing the Work

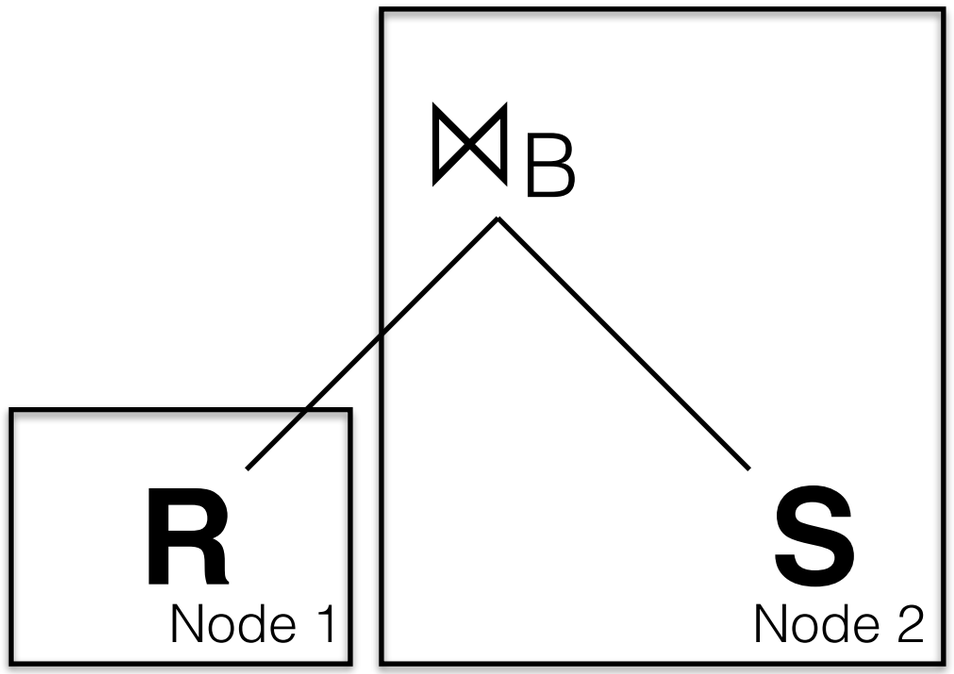


Distributing the Work

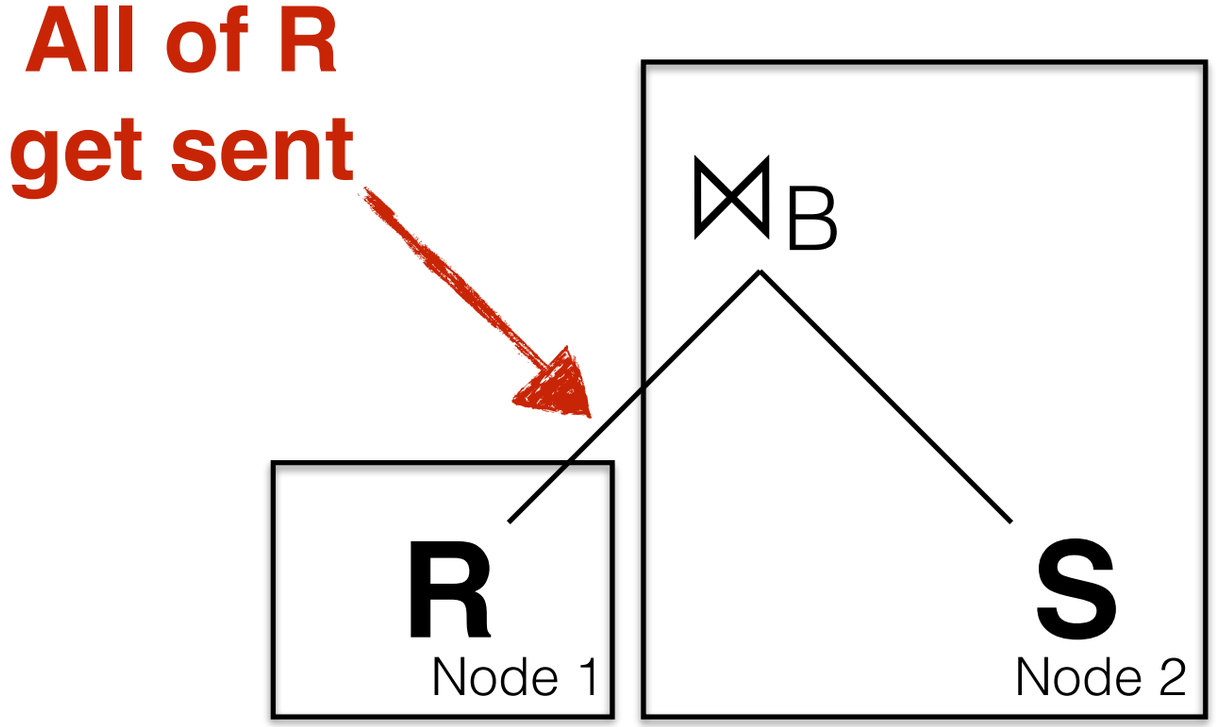


Lots of Data Transfer!

Distributing the Work



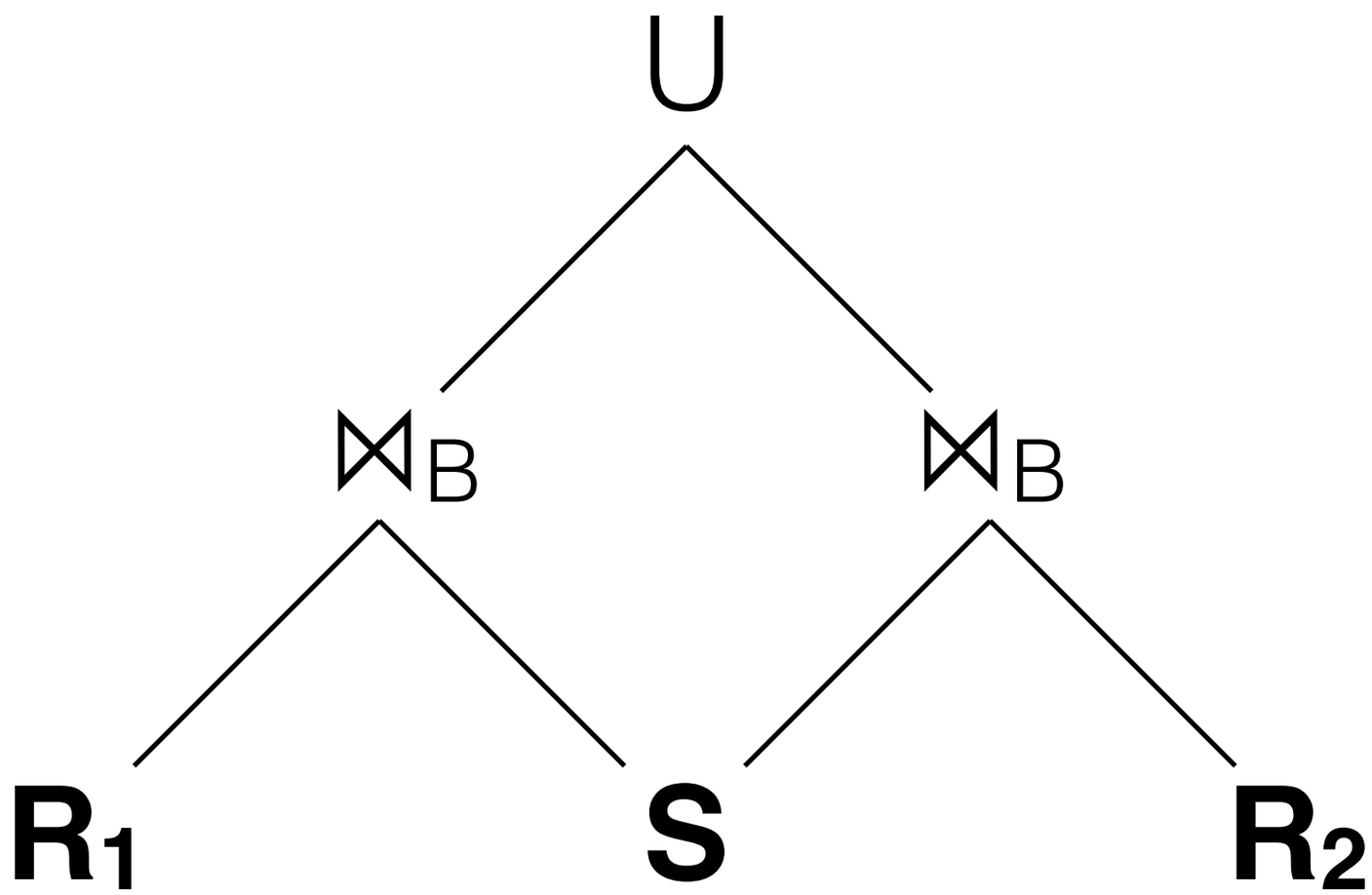
Distributing the Work



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

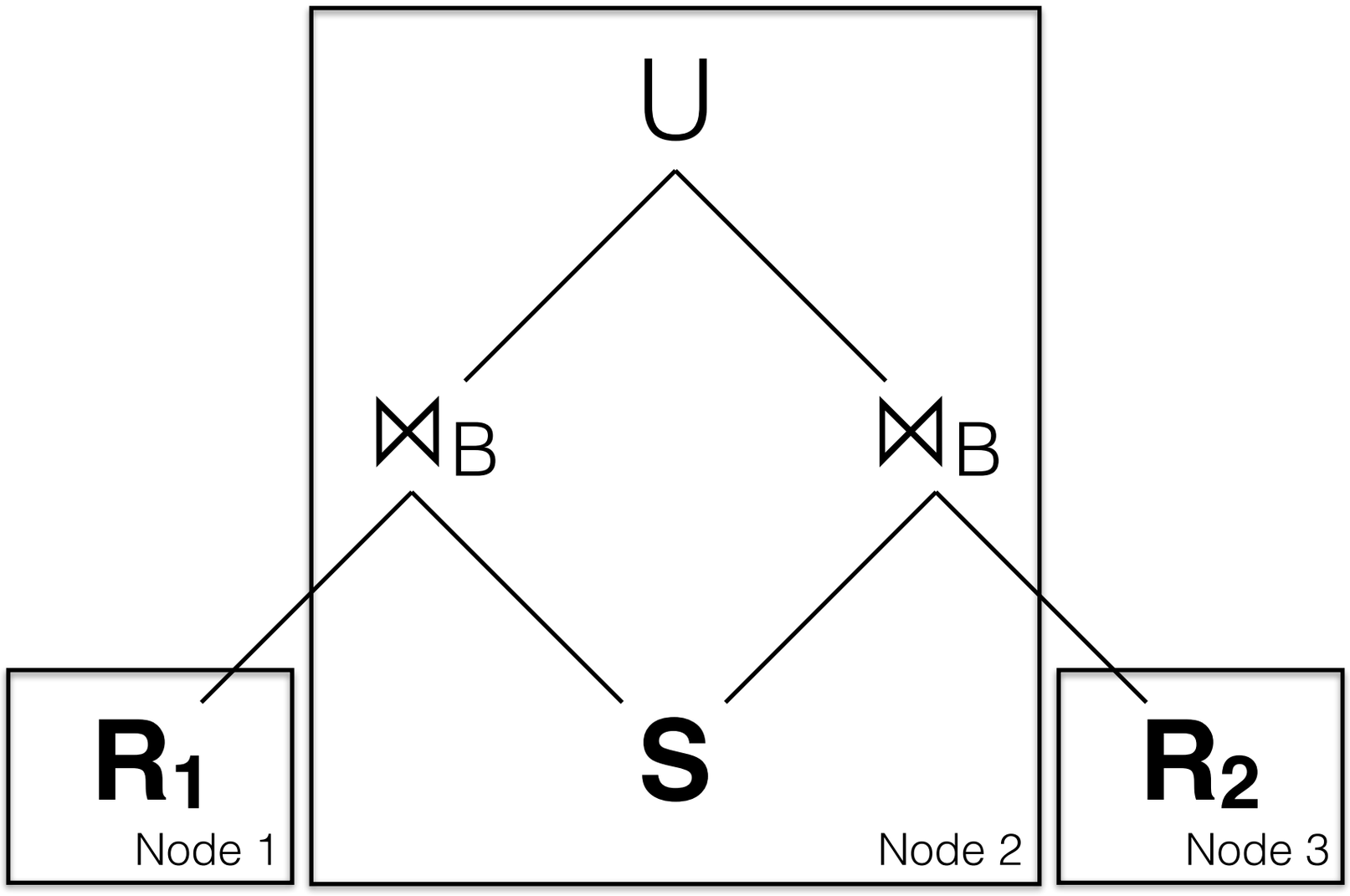
Distributing the Work

What can we do if R is partitioned?



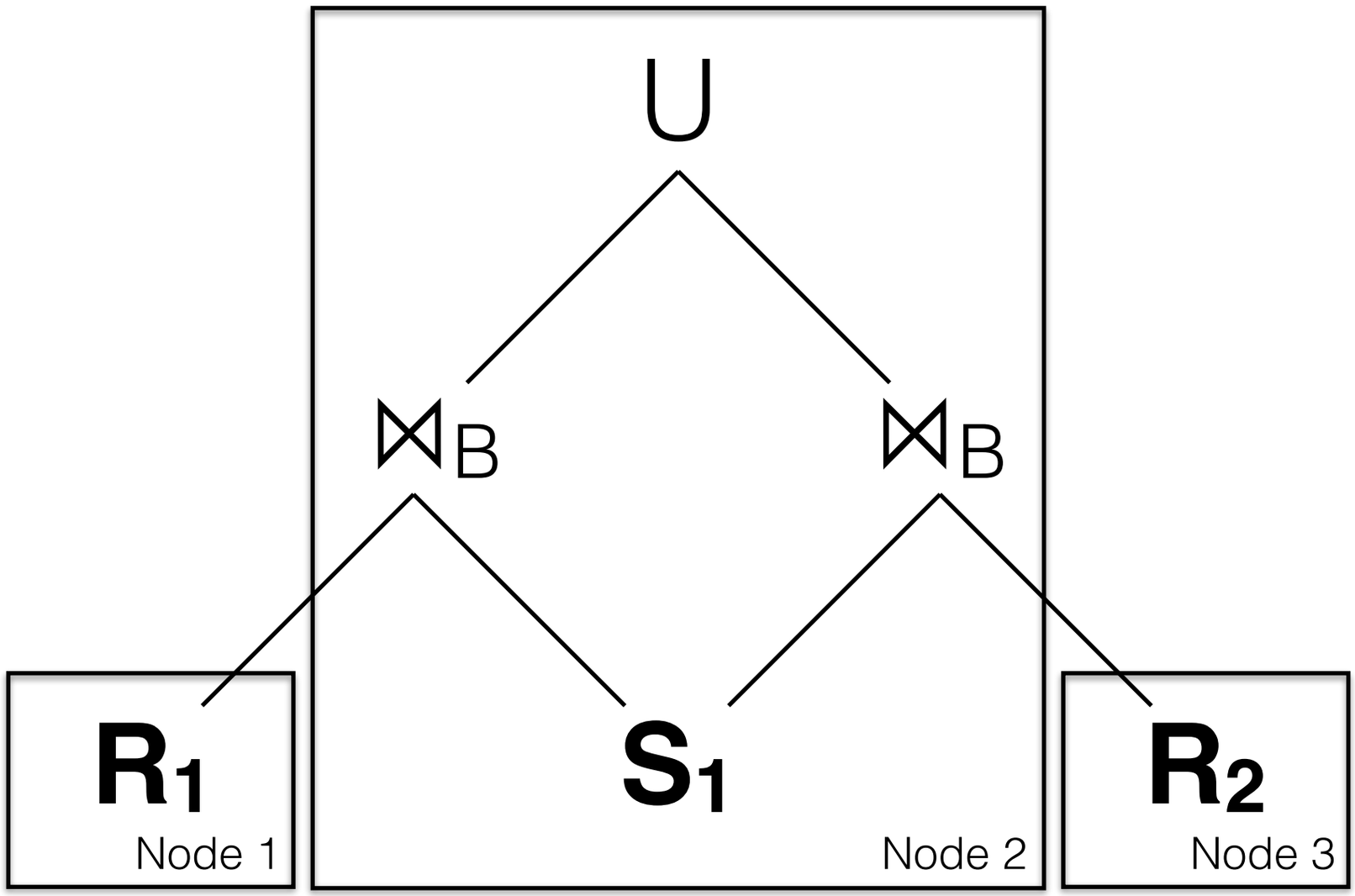
Distributing the Work

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



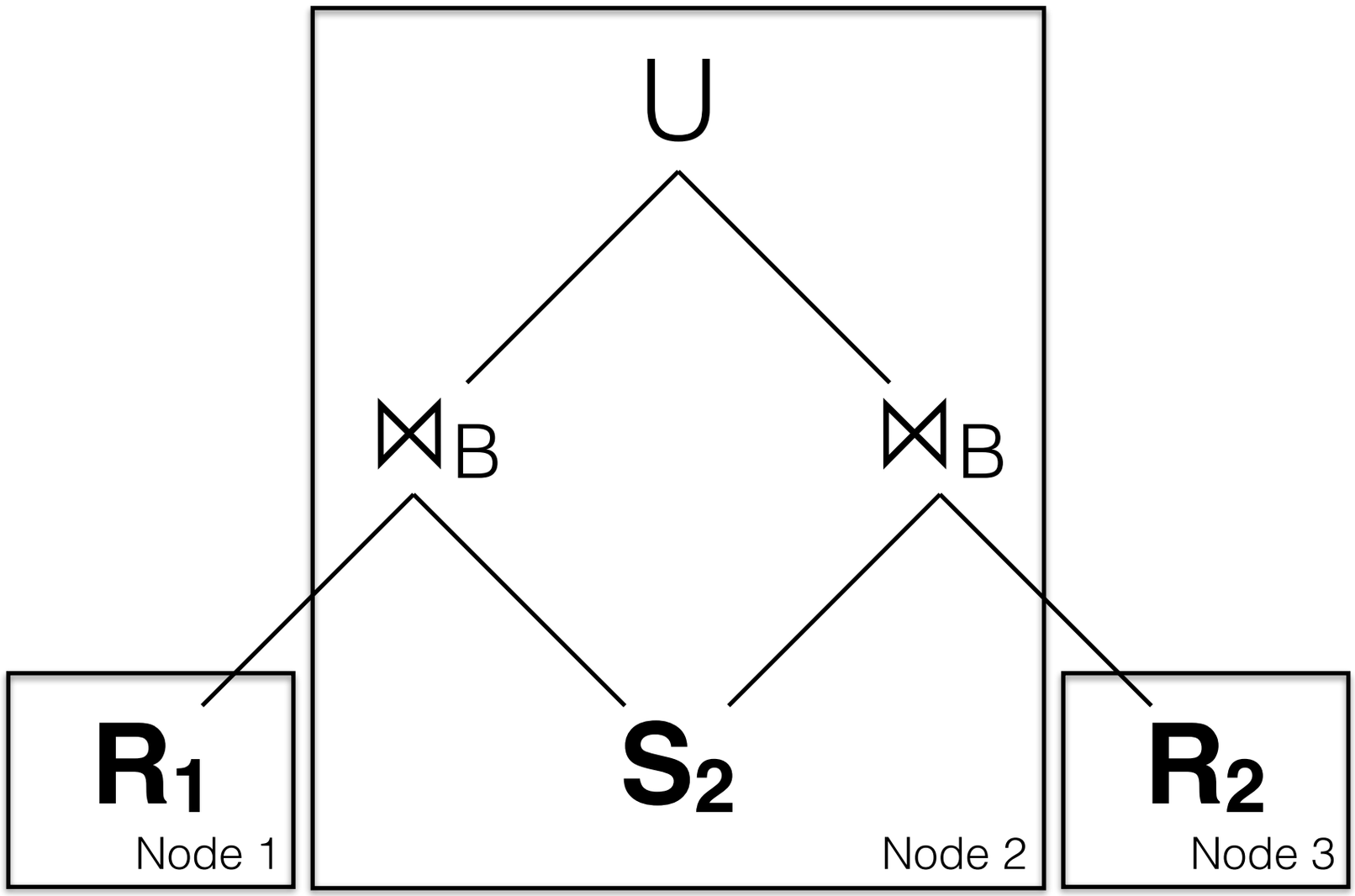
Distributing the Work

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



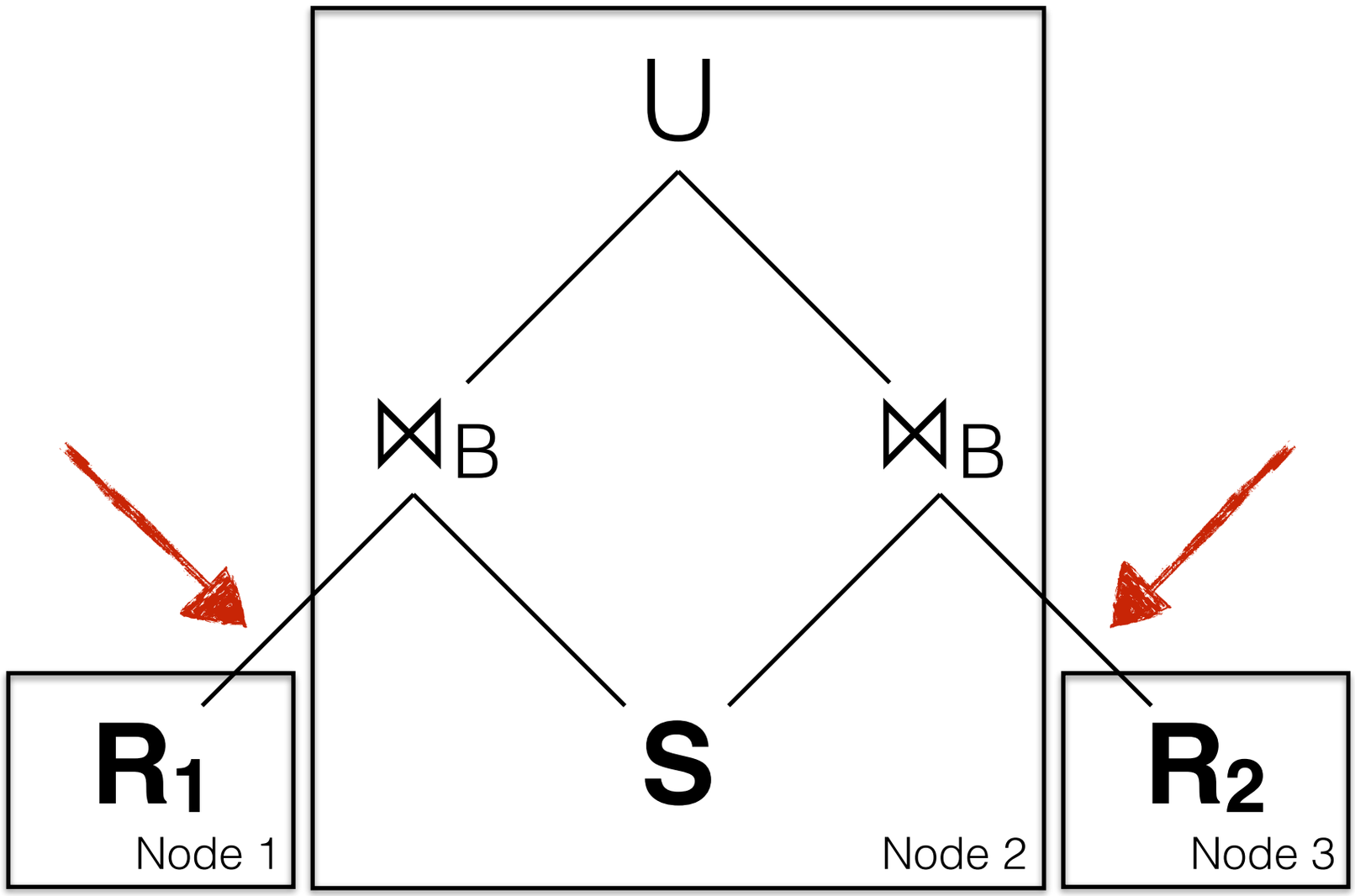
Distributing the Work

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



Distributing the Work

…and neatly captures the data transfer issue.



Distributing the Work

So let's use it:

S_i joins with $R_1, R_2, \ldots, R_N$ <u>locally</u>.

Distributing the Work

So let's use it:

S_i joins with $R_1, R_2, \ldots, R_N$ <u>locally</u>.

Goal: Minimize amount of data sent from R_k to S_i

Distributing the Work

So let's use it:

S_i joins with $R_1, R_2, \ldots, R_N$ locally.

Goal: Minimize amount of data sent from R_k to S_i

Solution 1: Use the partitioning strategy
(like last lecture)

Distributing the Work

So let's use it:

S_i joins with $R_1, R_2, \ldots, R_N$ <u>locally</u>.

Goal: Minimize amount of data sent from R_k to S_i

Solution 1: Use the partitioning strategy
(like last lecture)

Solution 2: "Hints" to figure out what R_k should send

Sending Hints

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

The naive approach…

Node 1

R_k

Node 2

S_i

Sending Hints

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

The naive approach…

Node 1

R_k

Send me R_k

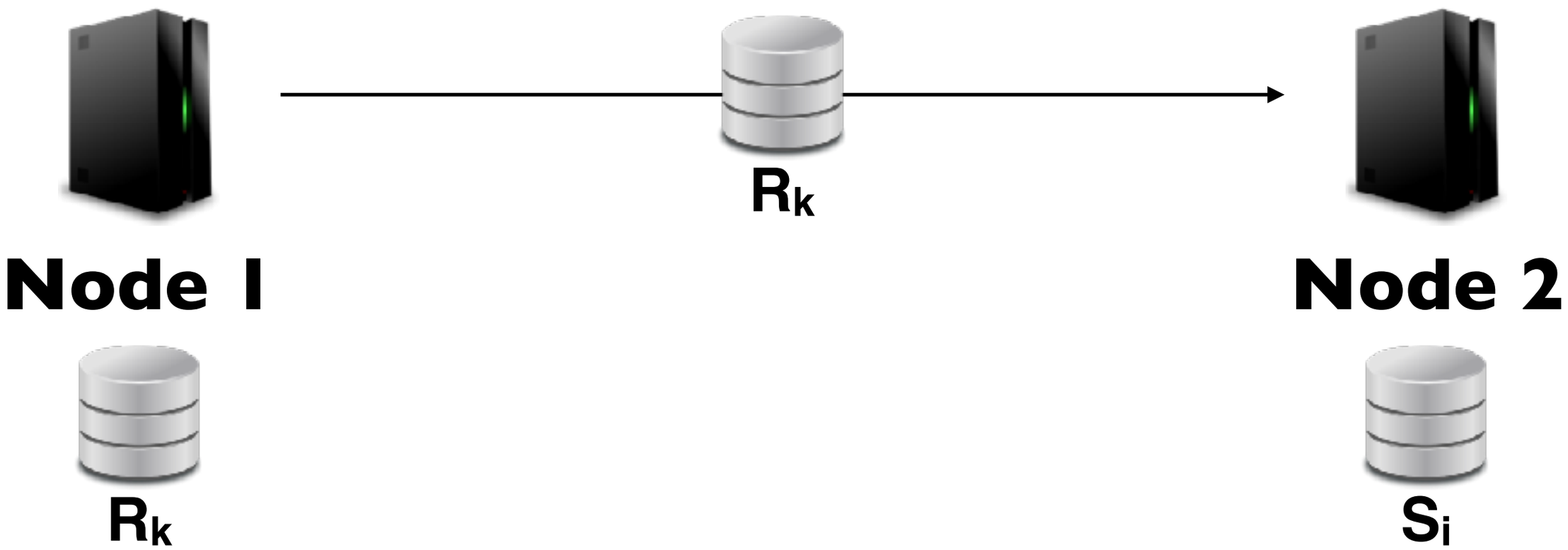
Node 2

S_i

Sending Hints

$R_k \bowtie_B S_i$

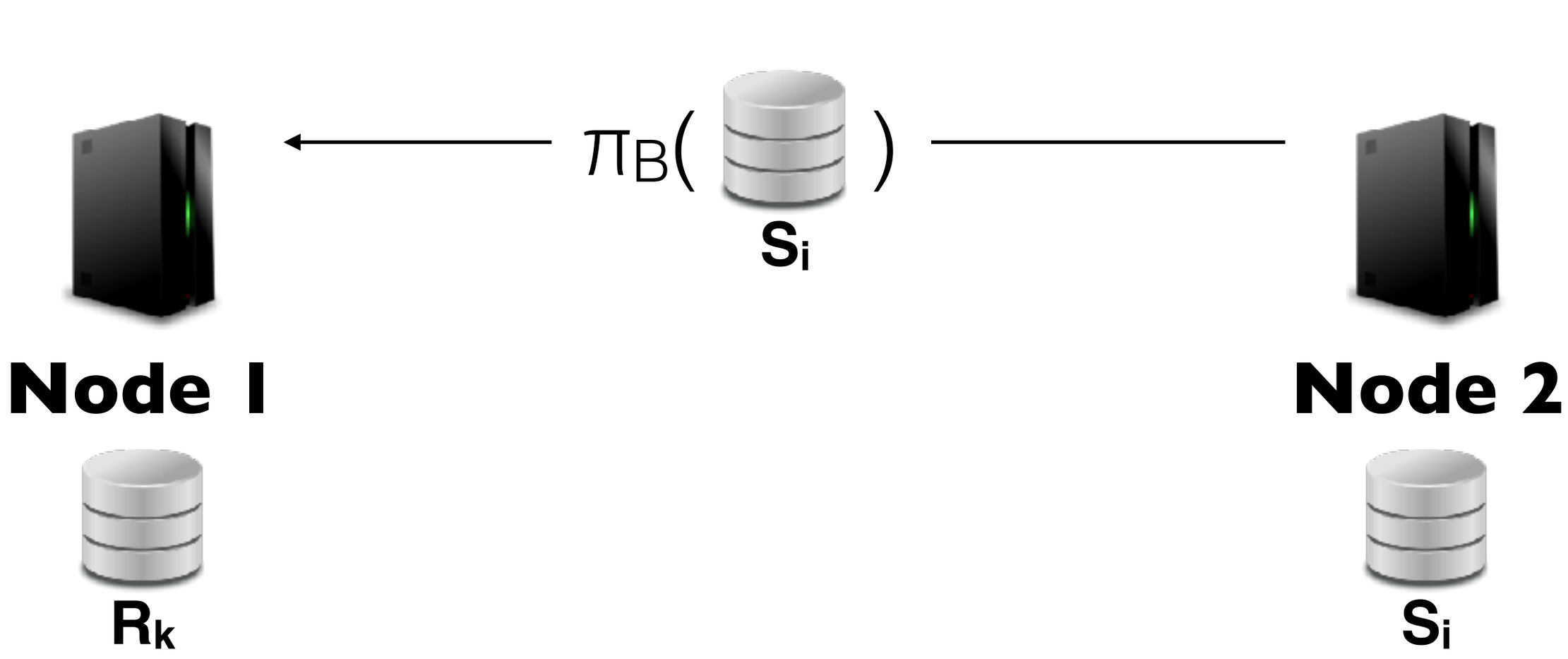
The naive approach…



Sending Hints

$R_k \bowtie_B S_i$

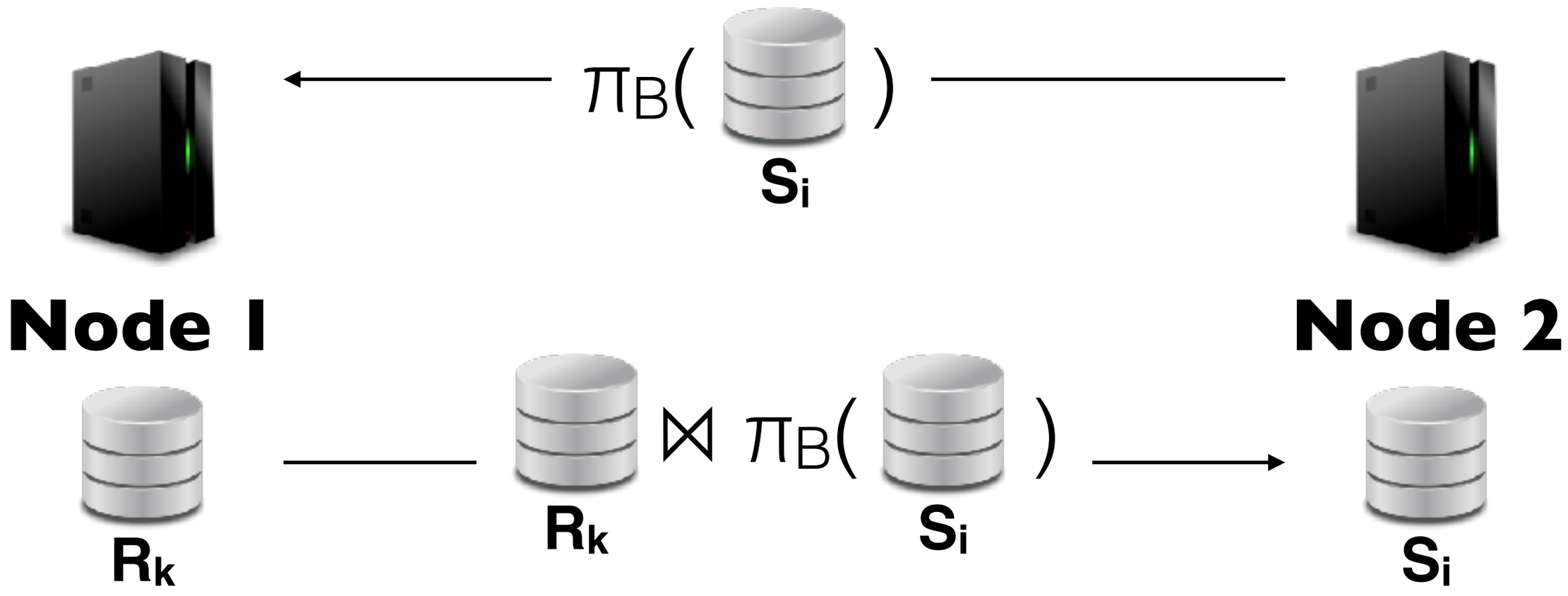
The smarter approach…



Sending Hints

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

The smarter approach…



Sending Hints

$R_k \bowtie_B S_i$

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>

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 semi-join.

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

$R_k \bowtie_B S_i$

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>

Send me data with a parity bit of '0'

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

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

Send me data with a parity bit of '0'

Node 2

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

Sending Hints

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

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>

Sending Hints

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

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>

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

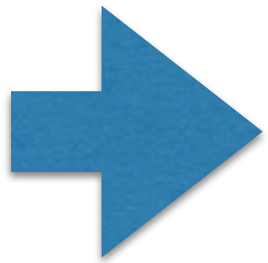
Can we summarize the parity bits?

Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filters

Alice
Bob
Carol
Dave

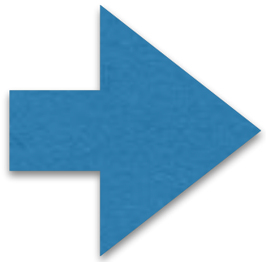


Bloom Filter

Bloom Filters

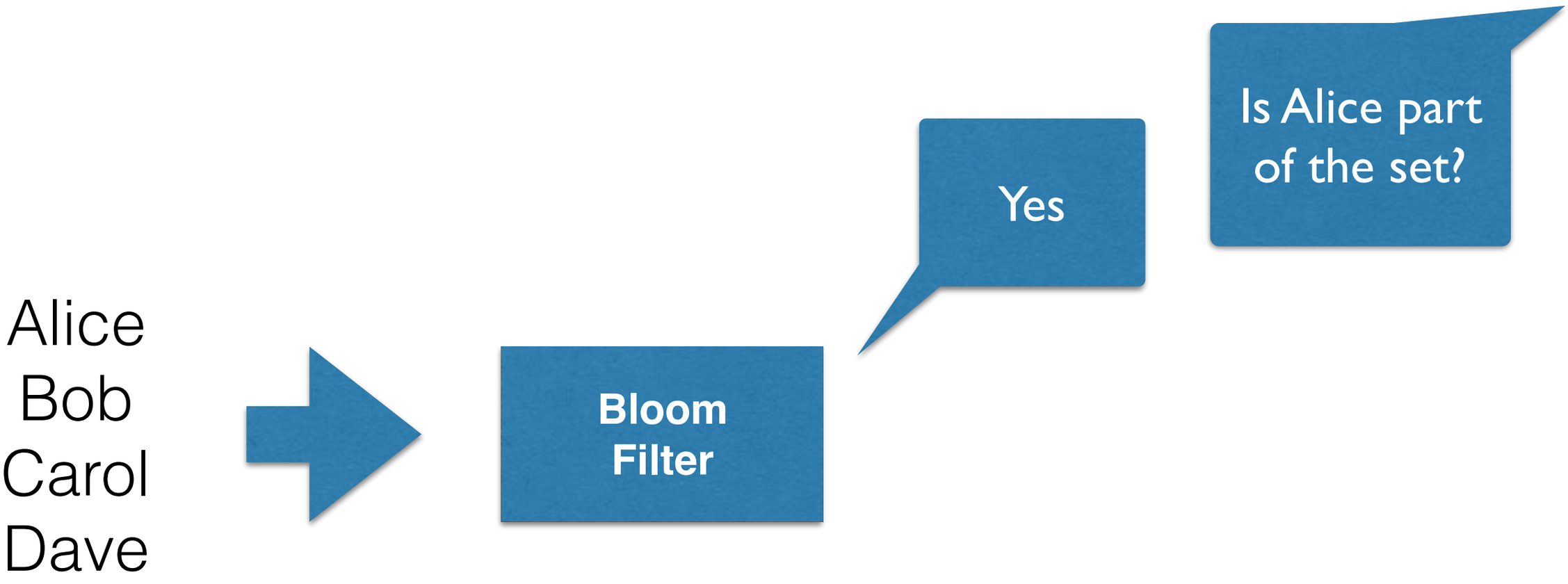
Is Alice part of the set?

Alice
Bob
Carol
Dave



Bloom Filter

Bloom Filters



Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter

Is Alice part of the set?

Yes

Is Eve part of the set?

Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter

Is Alice part of the set?

Yes

No

Is Eve part of the set?

Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter

Is Alice part of the set?

Yes

Is Eve part of the set?

No

Is Fred part of the set?

Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter

Is Alice part of the set?

Yes

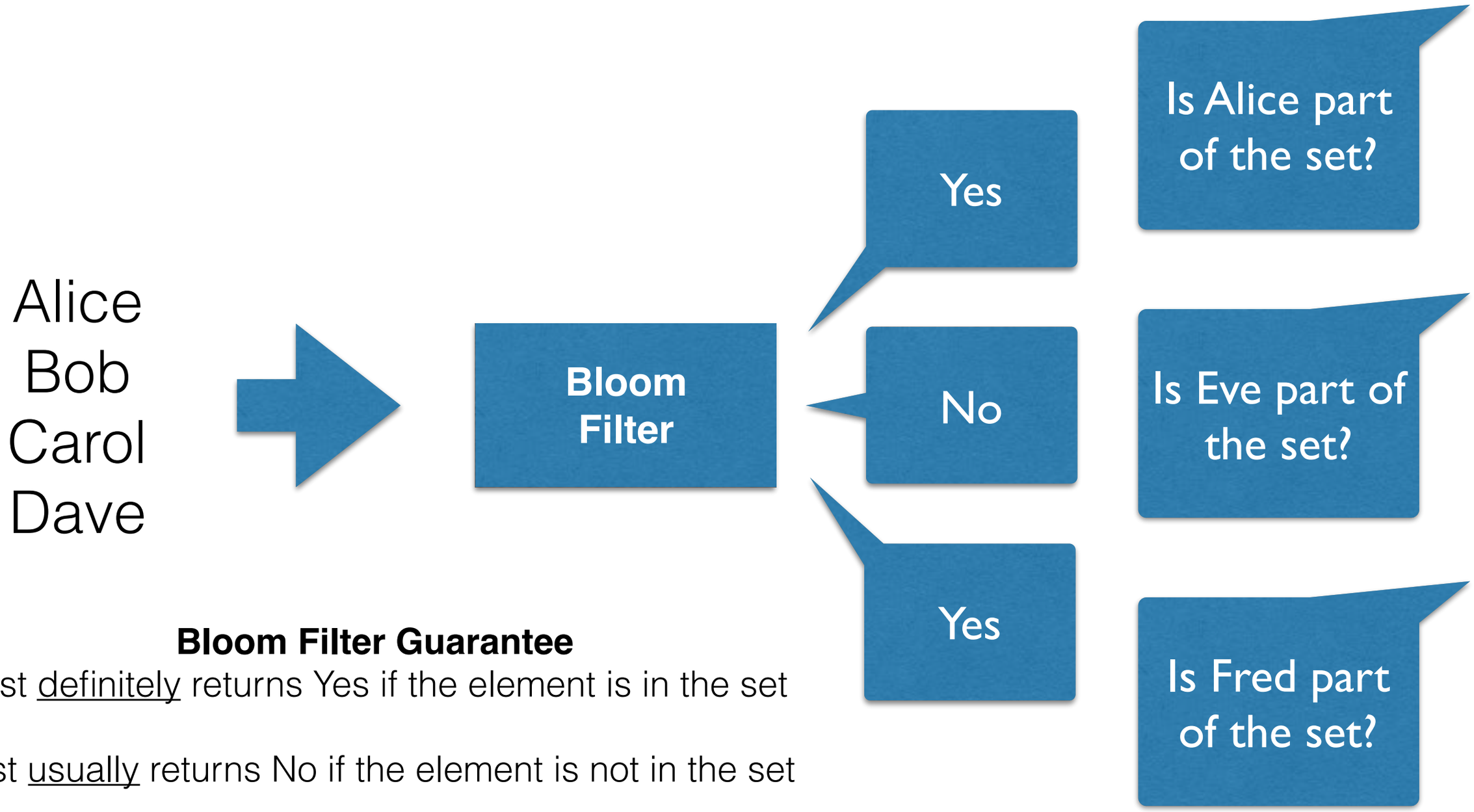
Is Eve part of the set?

No

Is Fred part of the set?

Yes

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

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

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?

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

(Key 4 & S) = 01001100

Bloom Filters

Key 1 00I0I0I0

Key 2 0I0I0II0

Key 3 I0000II0

Key 4 0I00II00

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

(Key 4 & S) = 01001100

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 ✗

(Key 4 & S) = 01001100

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 ✗

(Key 4 & S) = 01001100 ✓

False Positive

Sending Hints

$R_k \bowtie_B S_i$

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

$R_k \bowtie_B S_i$

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

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

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>

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>

<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 bloom-join.