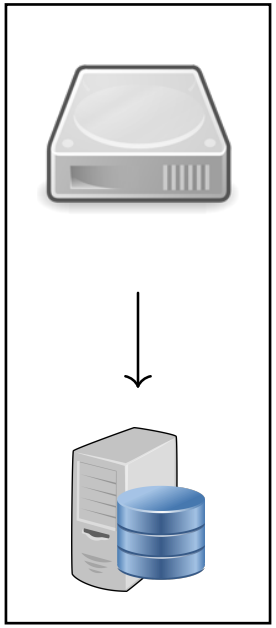
Parallelizing Queries

CSE 4/562: Database Systems | Lecture 14

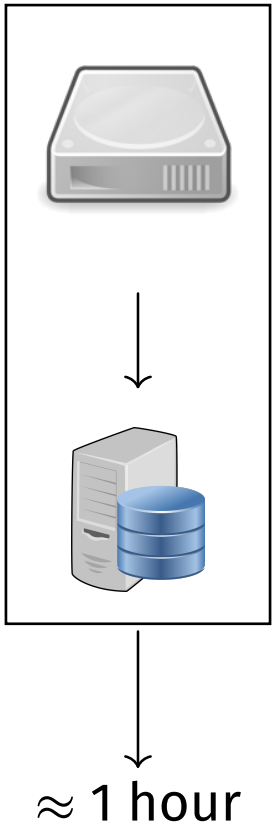
DB. Sys.: T.C.B.: Ch. 20.1-20.4

Why Scale?

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

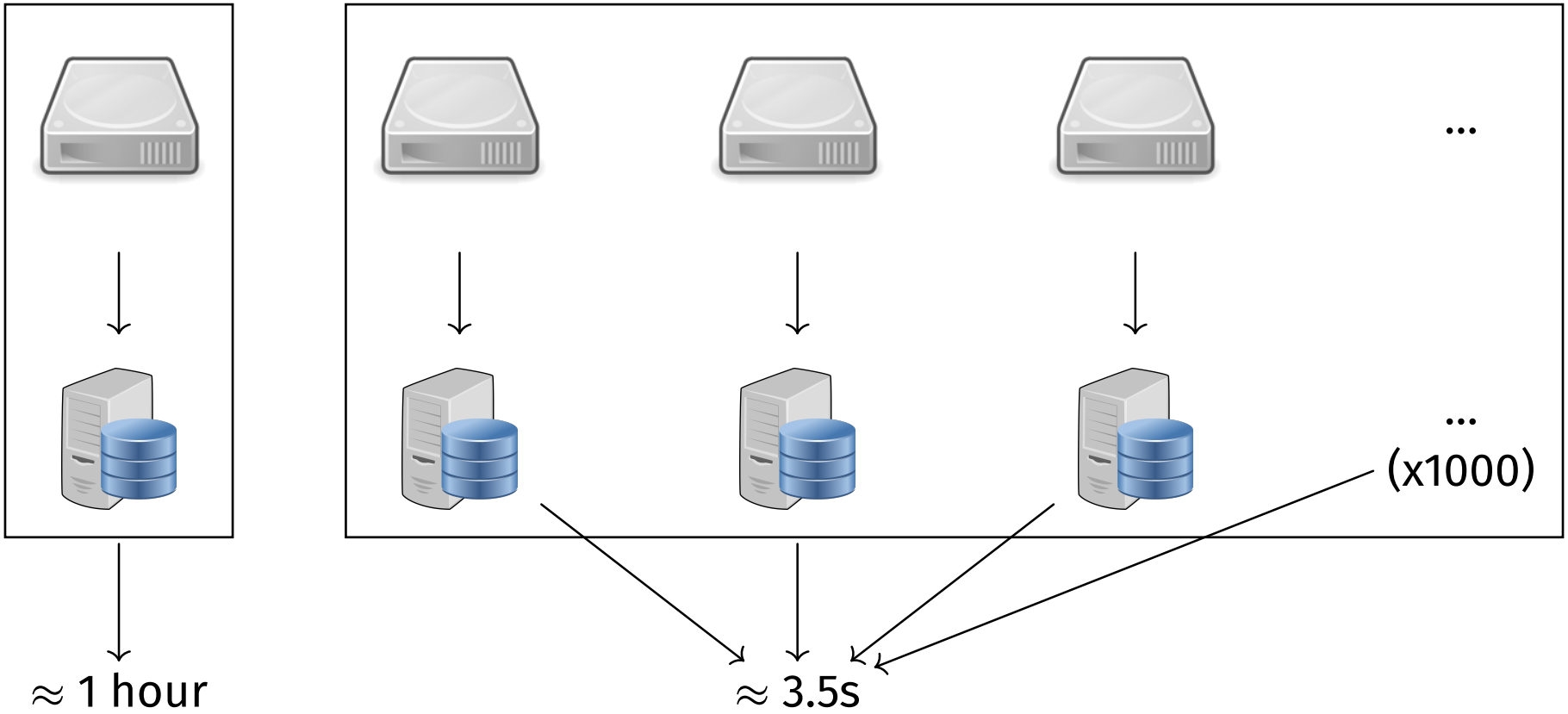


Scan 1 PB at 300MB/s (SATA r2)



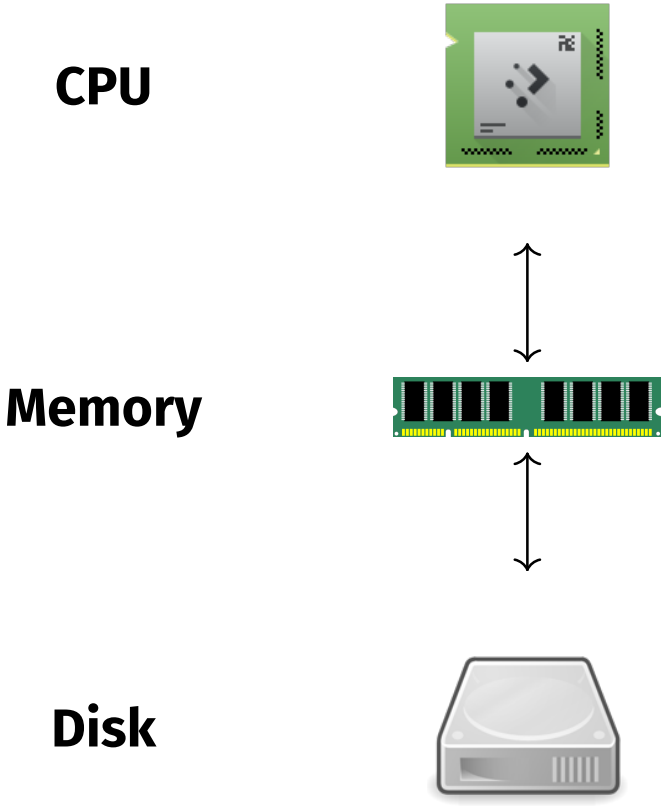
Advantages of Parallelization

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

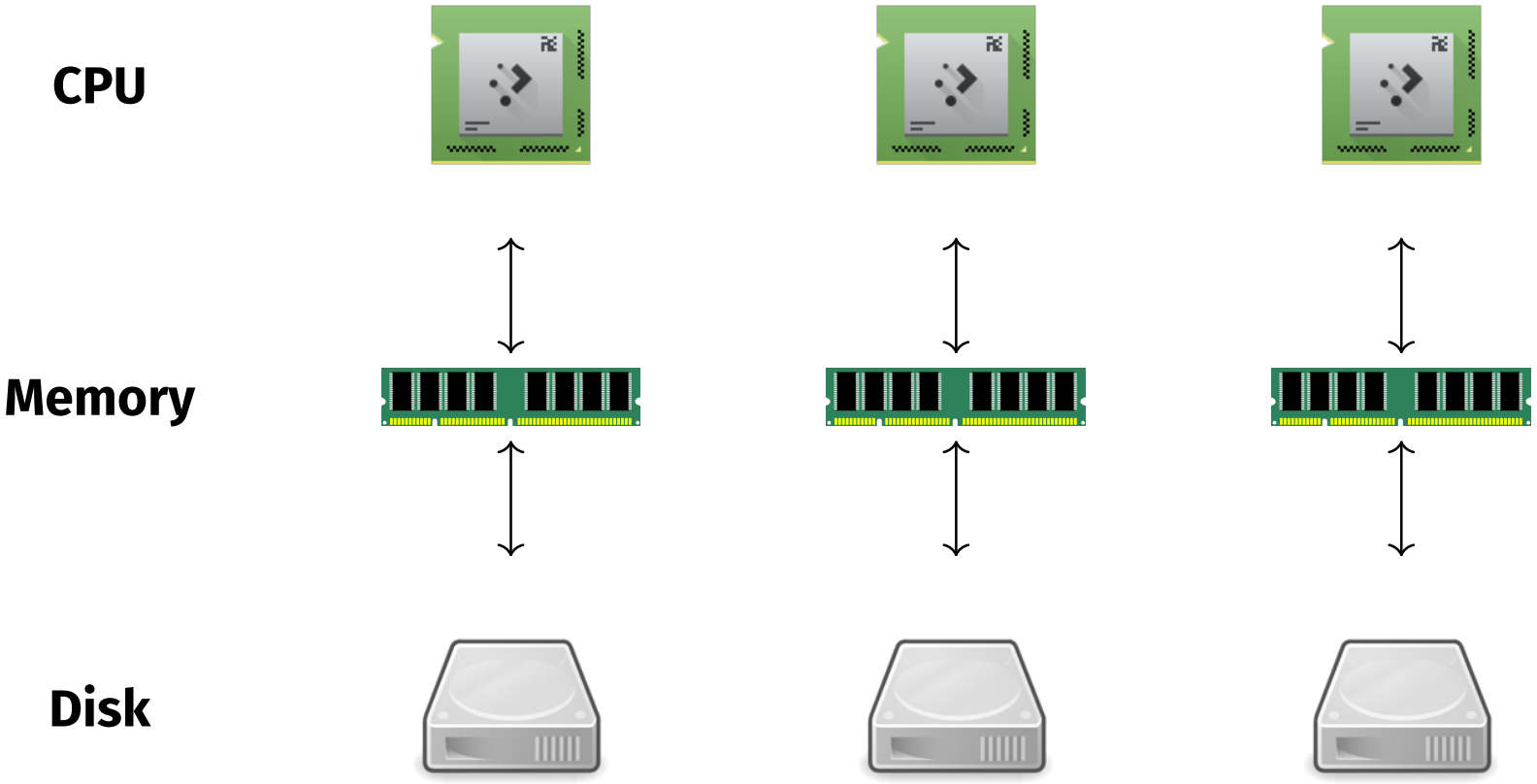


Communication Models

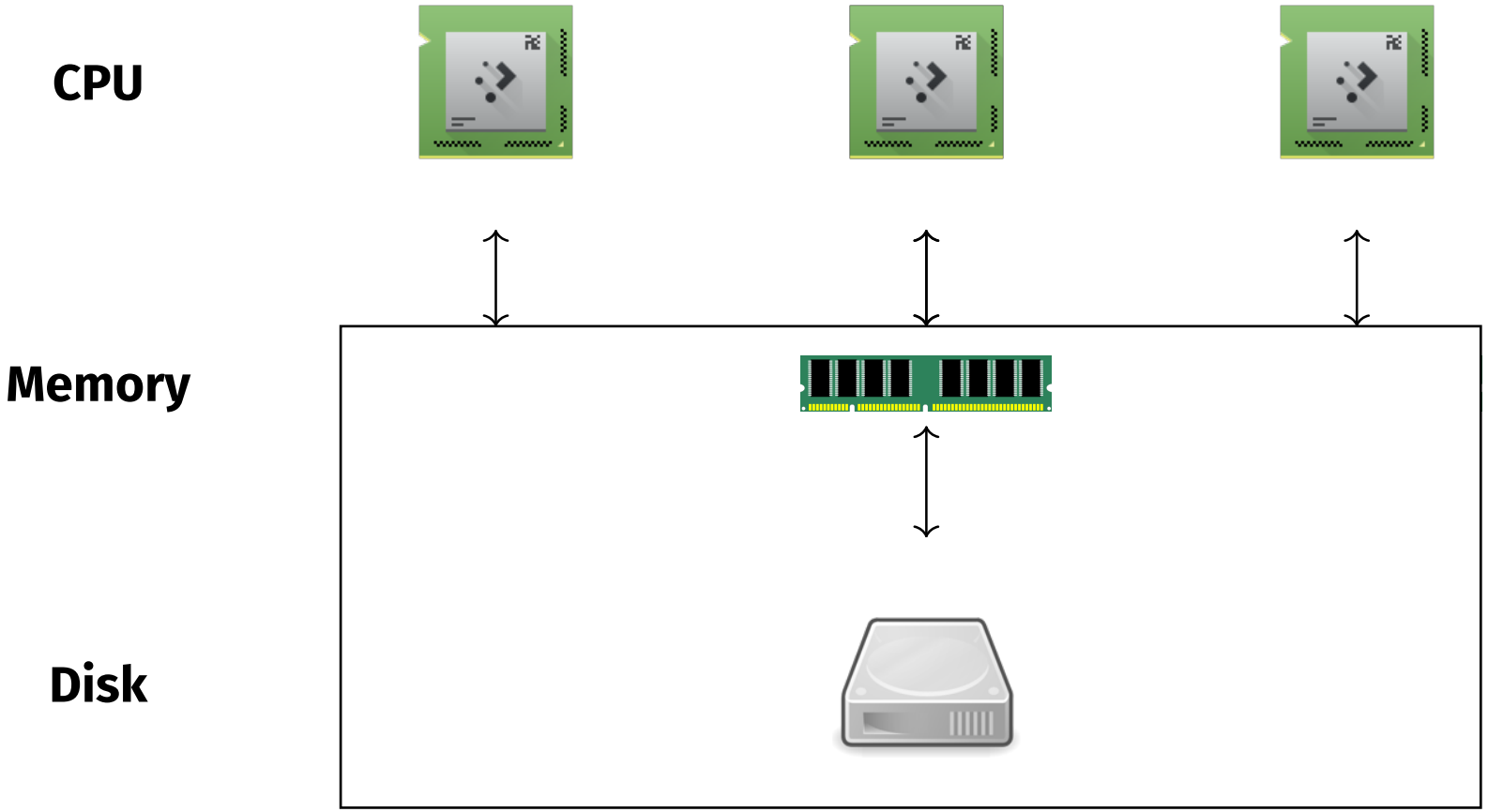
No Parallelism



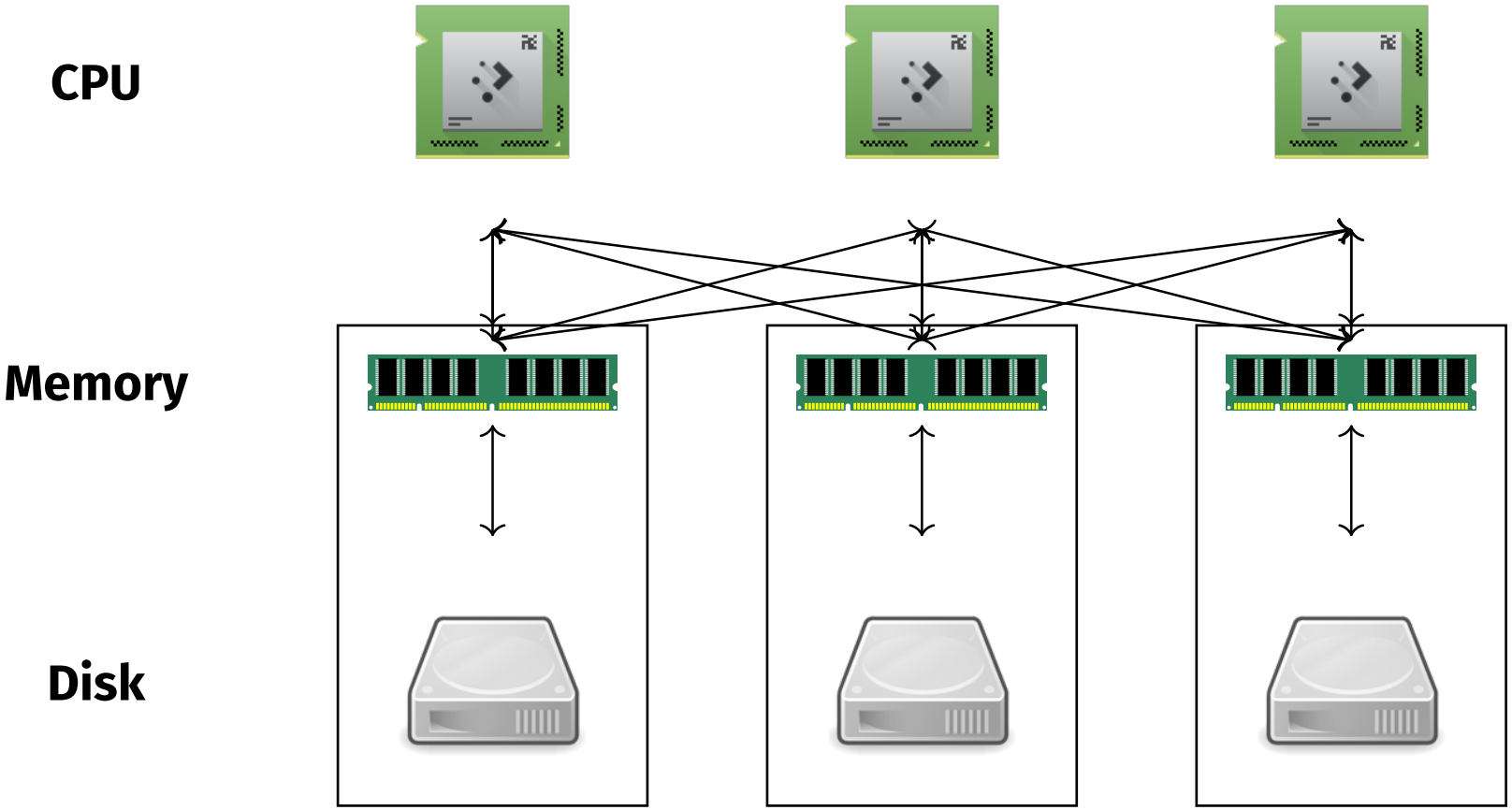
No Parallelism



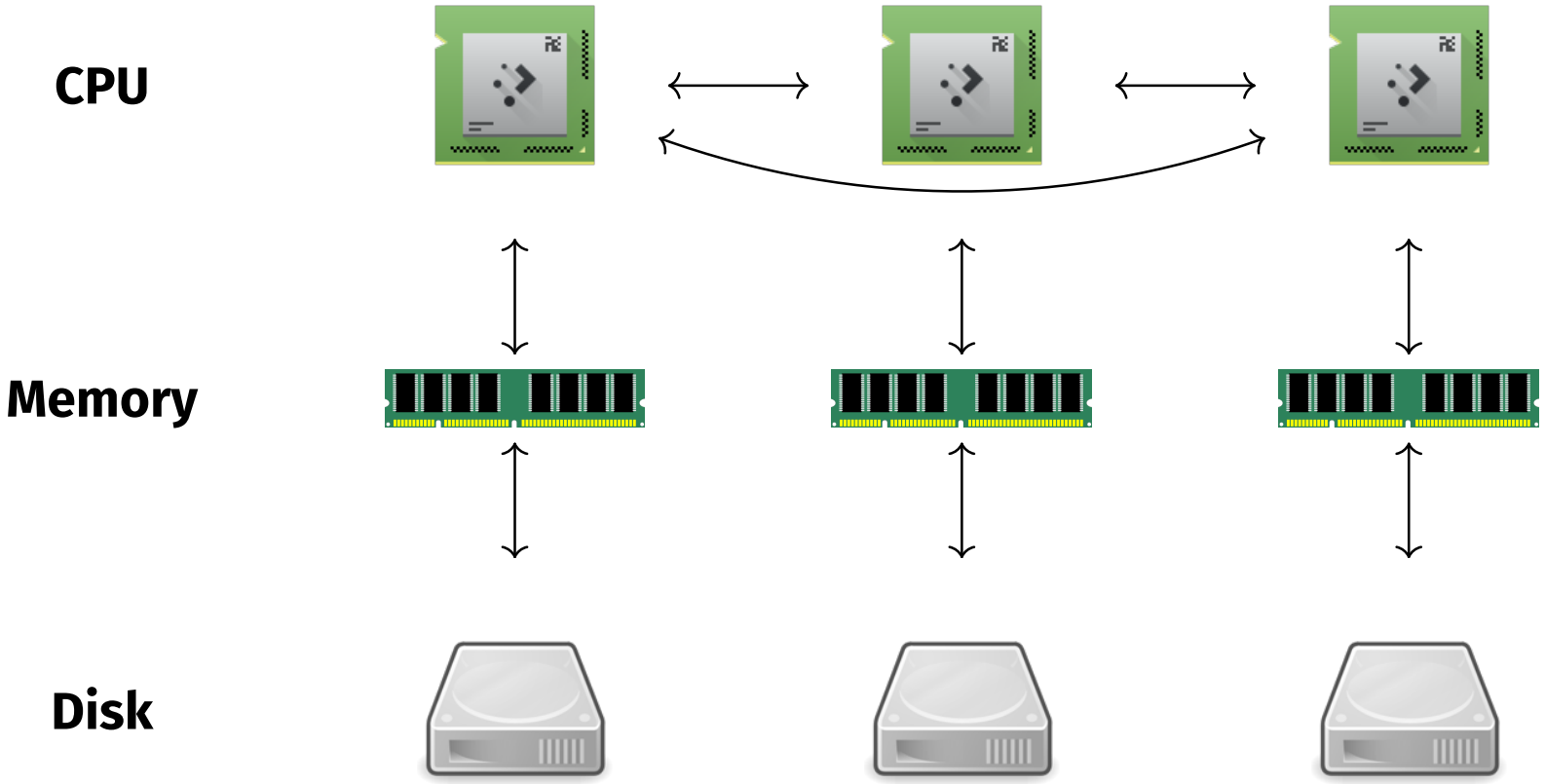
Shared Memory



Non-Uniform Memory Access



Shared Nothing / Message Passing



Shared Memory

- S3
- RAM/Modern OSes

NUMA

- AMD CPUs
- Hadoop

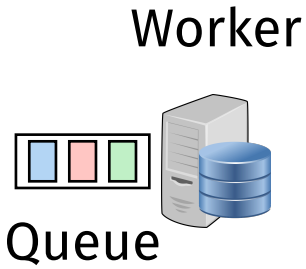
Shared Nothing

- MPP
- Apache Spark

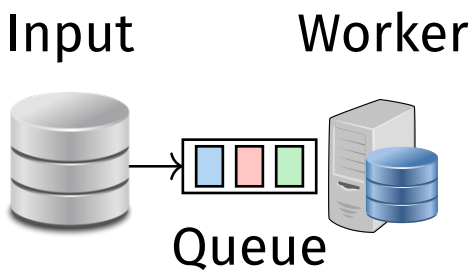
Today: Shared Nothing

Worker

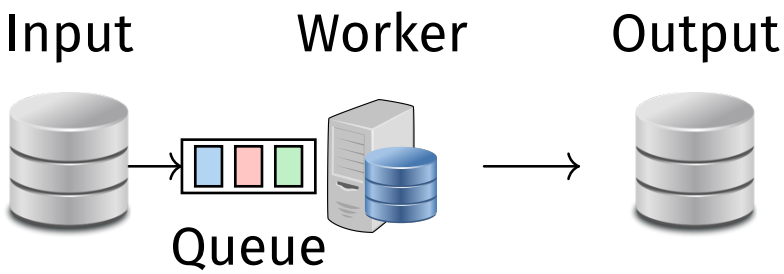
The Basic Model



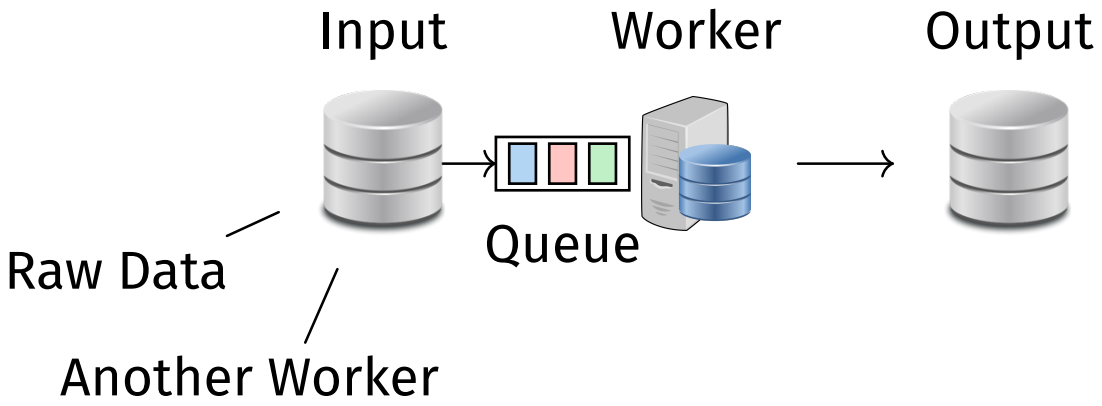
The Basic Model



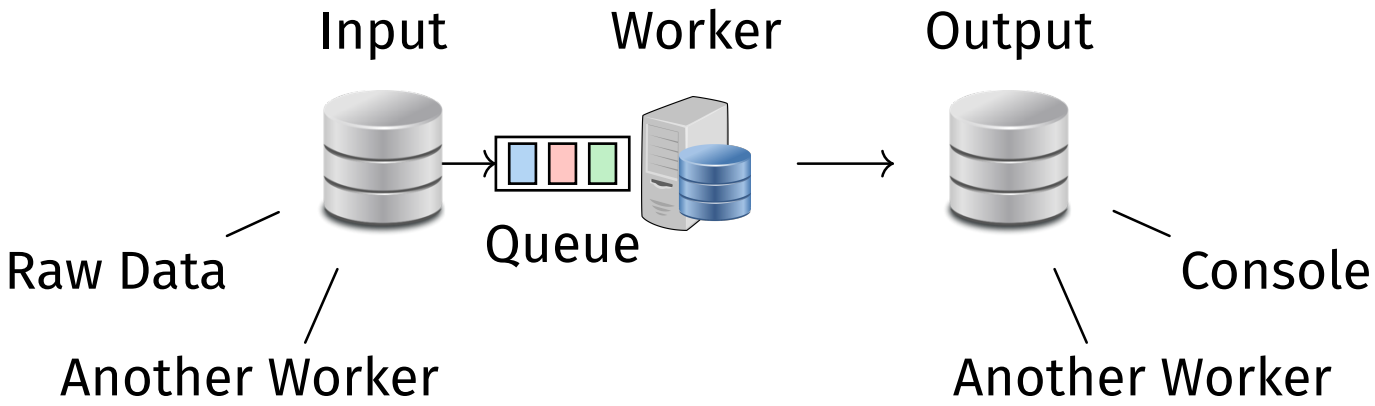
The Basic Model



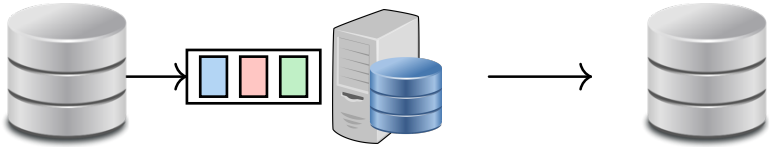
The Basic Model



The Basic Model

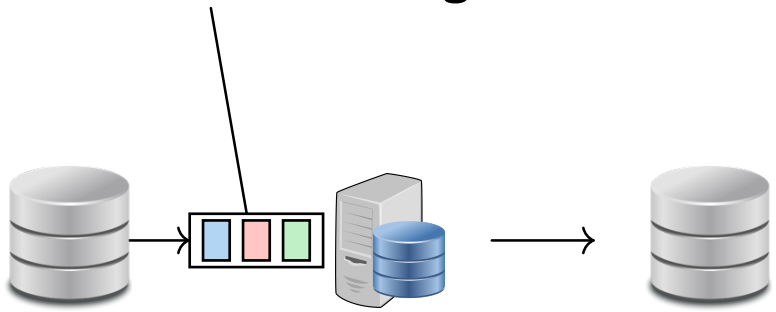


Potential Slowdowns



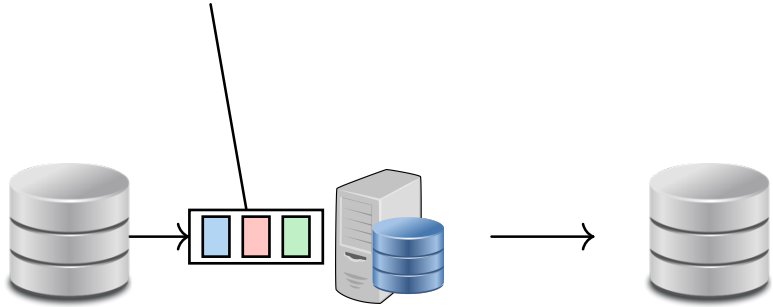
Potential Slowdowns

Data doesn’t arrive fast enough.



Potential Slowdowns

Data doesn’t arrive fast enough.



Worker processes data slower than it arrives.

Putting Workers Together

Image credit: Wikipedia - Public Domain Image - Photographer Unknown

Putting Workers Together

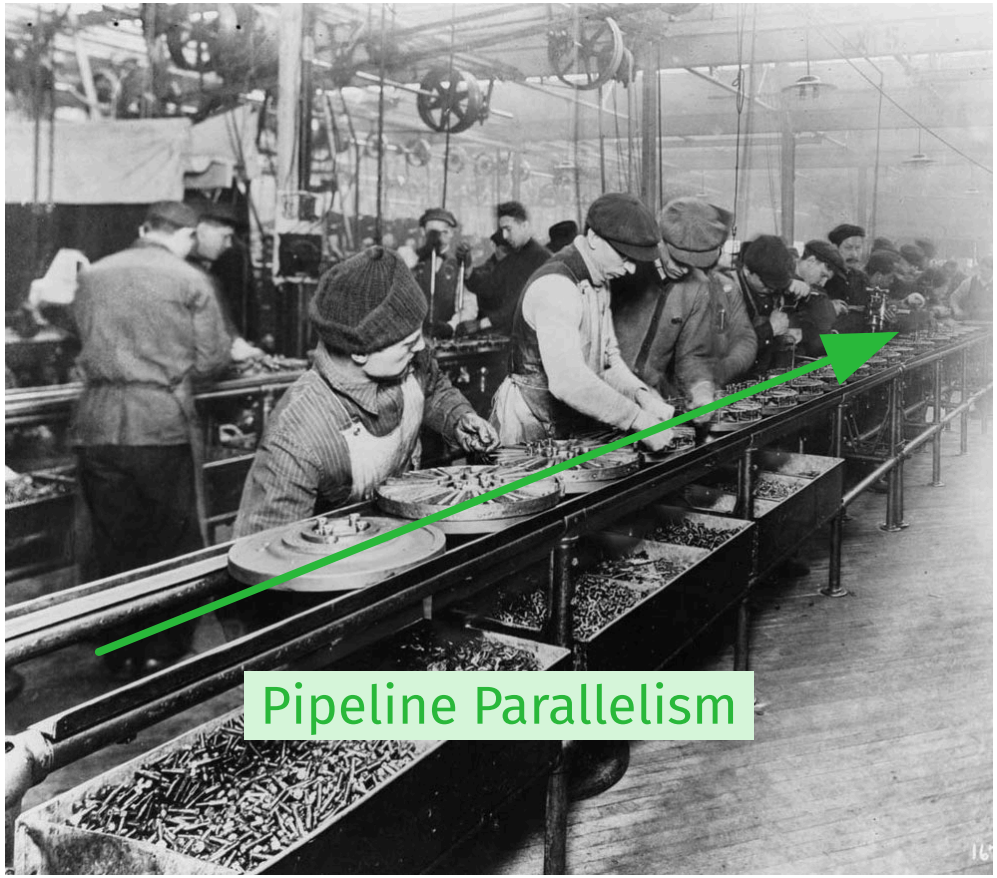


Image credit: Wikipedia - Public Domain Image - Photographer Unknown

Putting Workers Together

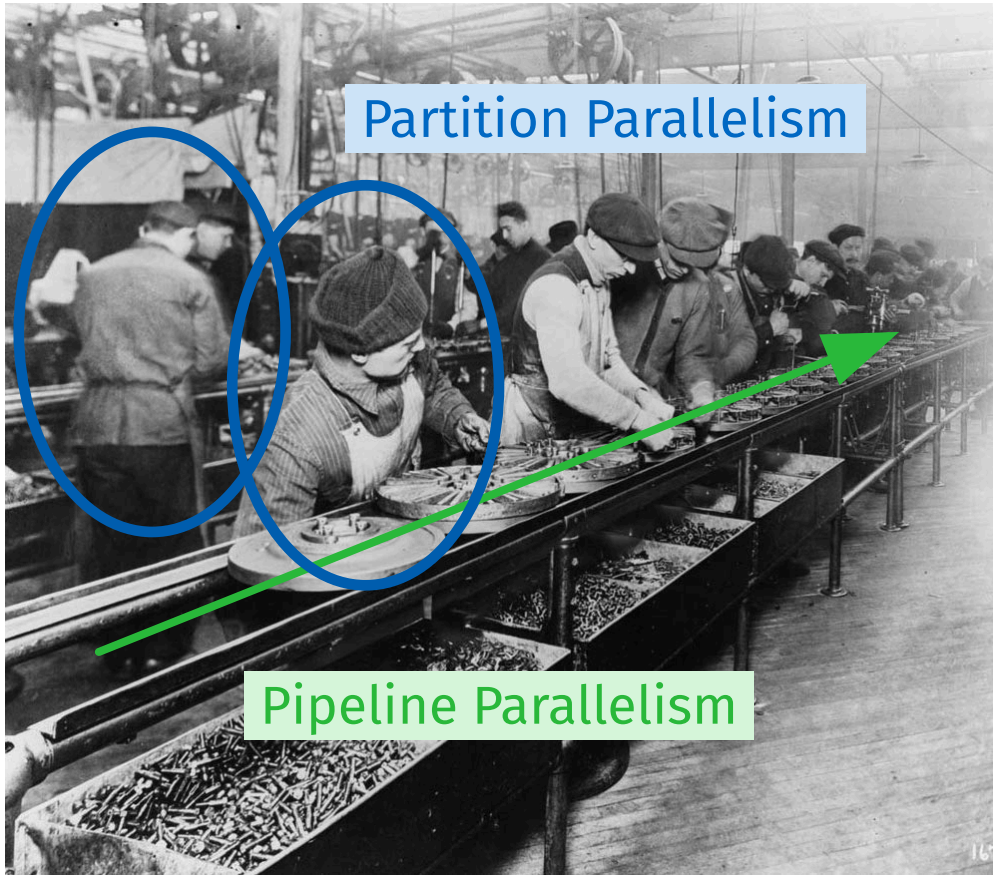
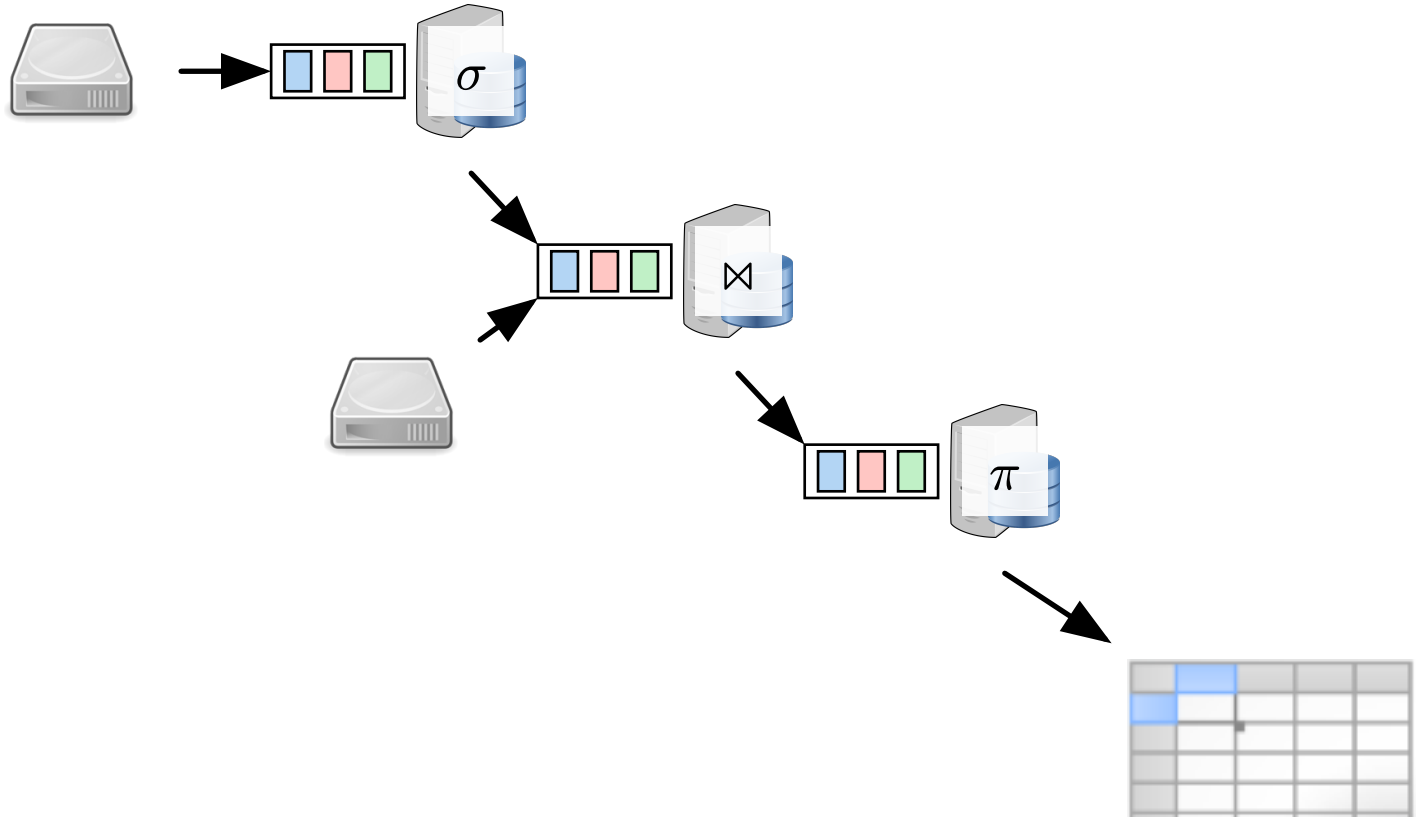
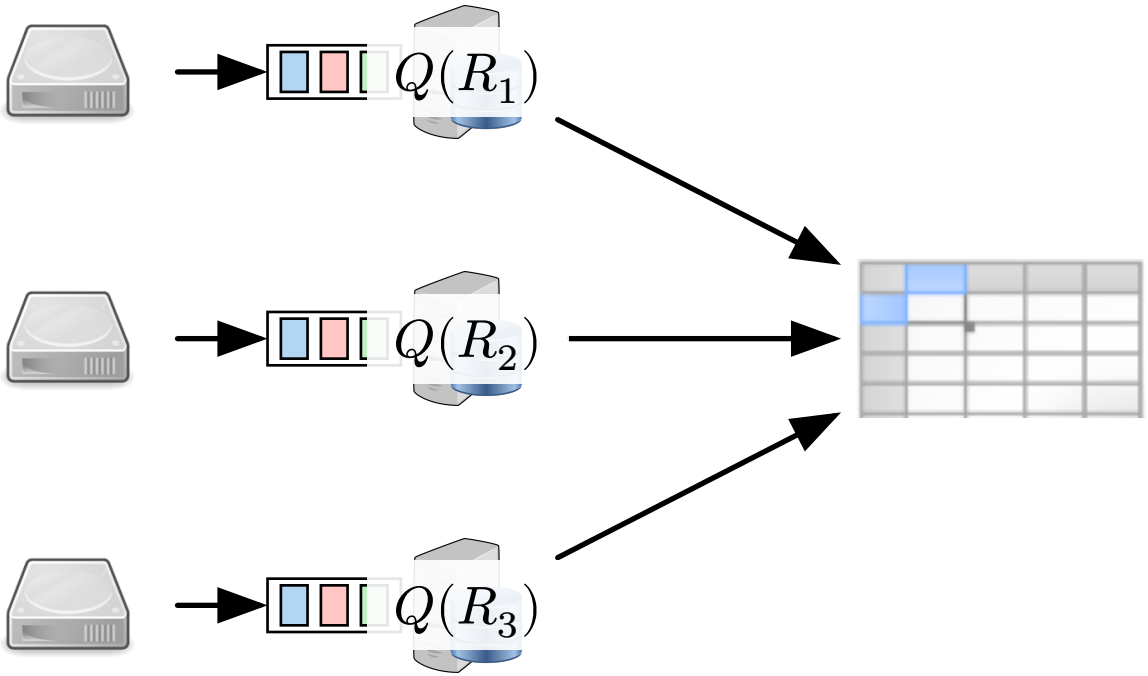


Image credit: Wikipedia - Public Domain Image - Photographer Unknown

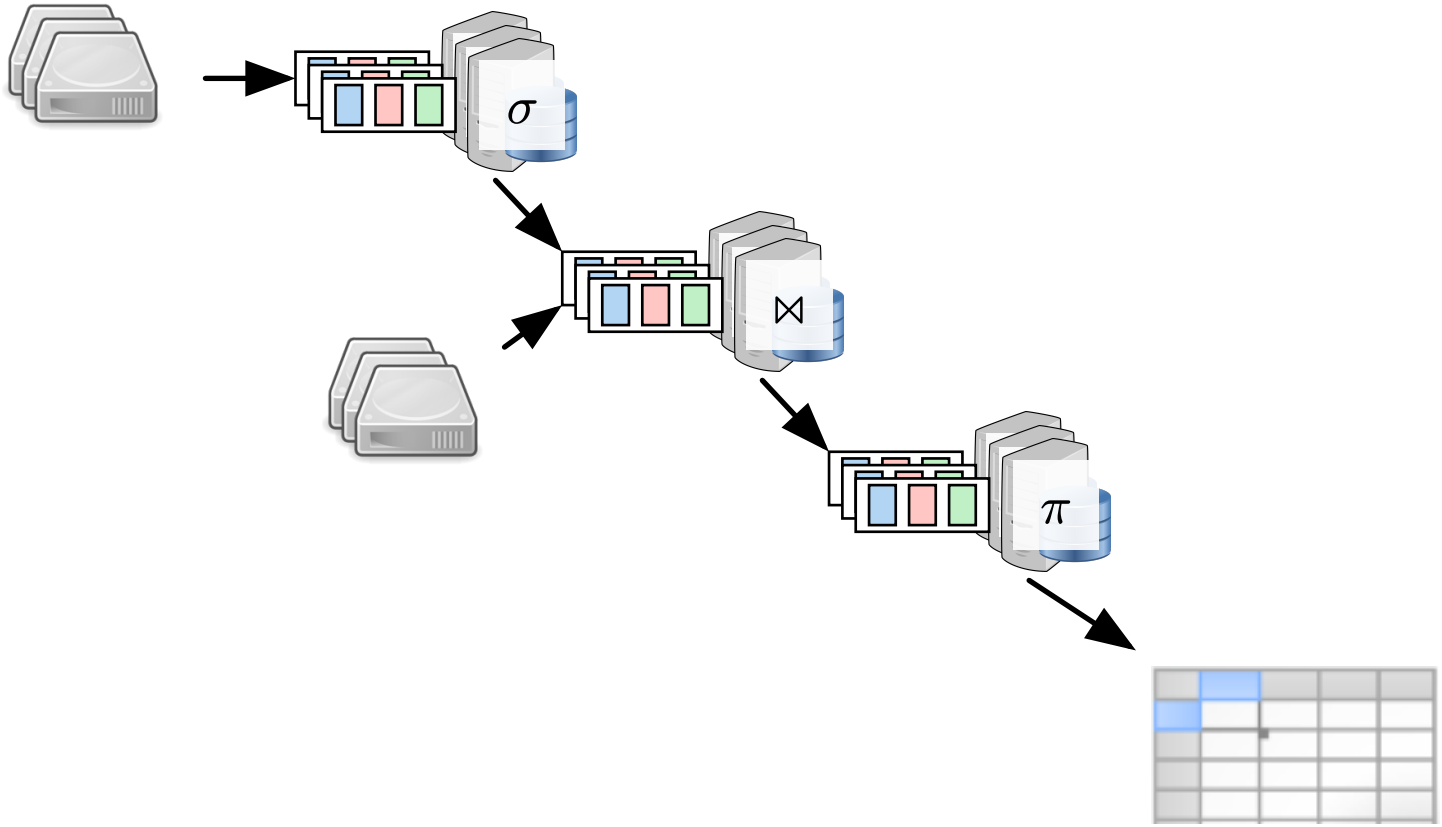
Pipeline Parallelism



Partition Parallelism



Together



Distributing Data

Replication

Partitioning (Sharding)

Partitioning Data

How do we pick what goes into each partition?

- Arbitrary
- Range
- Hash

Partitioning Data

How do we pick what goes into each partition?

- Arbitrary
- Range
- Hash

look familiar?

Partitioning Data

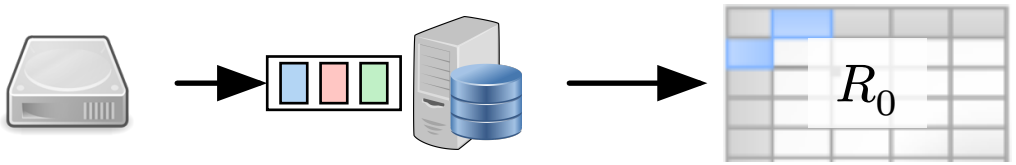
How do we pick what goes into each partition?

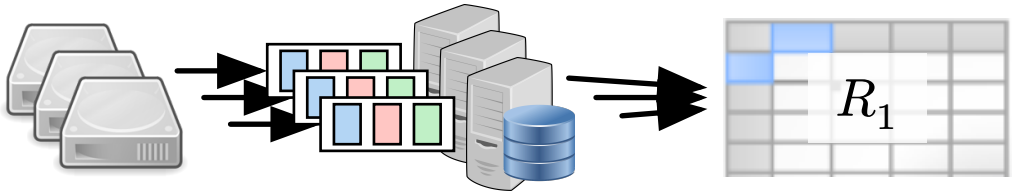
- Arbitrary
- Range
- Hash

look familiar?

How do we parallelize?

Equivalence





$$R_0 \stackrel{?}{=} R_1$$

Equivalence

Can we run each worker independently on one partition?

$$Q(R_0 \cup ... \cup R_N) \stackrel{?}{=} Q(R_0) \cup ... \cup Q(R_N)$$

Equivalence - The Easy Cases

Selection

- $\sigma(R_0 \cup ... \cup R_N) =$

Projection

- $\pi(R_0 \cup ... \cup R_N) =$

Union

- $(R_0 \cup ... \cup R_N) =$

Equivalence - The Easy Cases

Selection

- $\sigma(R_0 \cup ... \cup R_N) = \sigma(R_0) \cup ... \cup \sigma(R_N)$

Projection

- $\pi(R_0 \cup ... \cup R_N) =$

Union

- $(R_0 \cup ... \cup R_N) =$

Equivalence - The Easy Cases

Selection

- $\sigma(R_0 \cup ... \cup R_N) = \sigma(R_0) \cup ... \cup \sigma(R_N)$

Projection

- $\pi(R_0 \cup ... \cup R_N) = \pi(R_0) \cup ... \cup \pi(R_N)$

Union

- $(R_0 \cup ... \cup R_N) =$

Equivalence - The Easy Cases

Selection

- $\sigma(R_0 \cup ... \cup R_N) = \sigma(R_0) \cup ... \cup \sigma(R_N)$

Projection

- $\pi(R_0 \cup ... \cup R_N) = \pi(R_0) \cup ... \cup \pi(R_N)$

Union

- $(R_0 \cup ... \cup R_N) = (R_0) \cup ... \cup (R_N)$

Aggregation

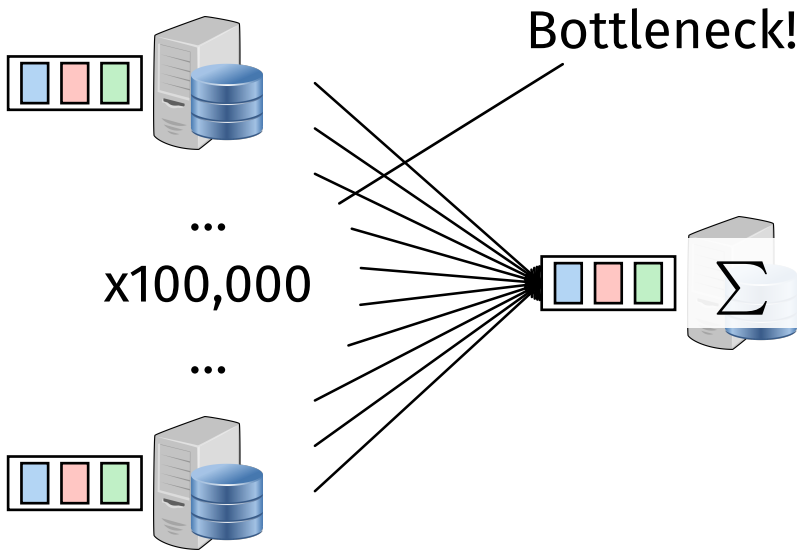
Single Row Aggregation

- $\sum(R_0 \cup ... \cup R_N) =$

Single Row Aggregation

- $\sum(R_0 \cup ... \cup R_N) =$???

Distributed Aggregation



Algebraic Aggregates

- `Init()`: The “default” value of the aggregate
- `Accumulate(agg, value)`: Incorporate a new value
- `Finalize(agg) -> value`: Post-process the aggregate

Algebraic Aggregates

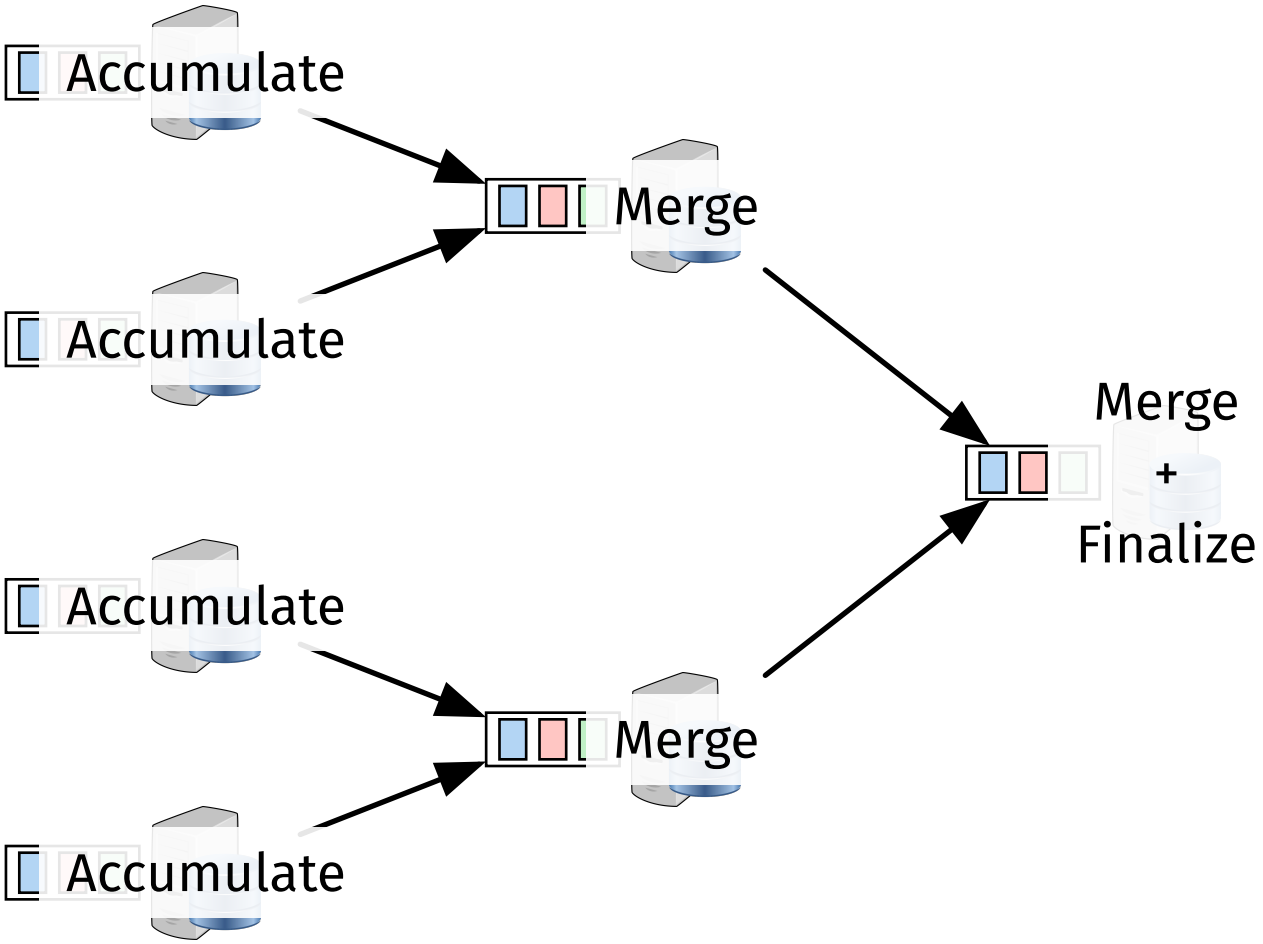
- `Init()`: The “default” value of the aggregate
- `Accumulate(agg, value)`: Incorporate a new value
- `Finalize(agg) -> value`: Post-process the aggregate
- `Merge(agg, agg)`: Combine independent intermediate aggregates

Single Row Aggregation

- $\sum(R_0 \cup ... \cup R_N) = \sum(R_0) + ... + \sum(R_N)$

Final “Merge” step only processes N tuples

Fold/Merge


Accumulate
Accumulate
Merge
Accumulate
Accumulate
Merge
Merge
+
Finalize

Multi-Row Aggregation

- **Arbitrary Partitioning**: As single-row
- **Hash/Range Partitioning**: Aggregate groups do not cross partitions

Joins

$$(R_1 \cup R_2) \bowtie (S_1 \cup S_2)$$

$$(R_1 \cup R_2) \bowtie (S_1 \cup S_2)$$

$$(R_1 \bowtie (S_1 \cup S_2)) \cup (R_2 \bowtie (S_1 \cup S_2))$$

$$(R_1 \cup R_2) \bowtie (S_1 \cup S_2)$$

$$(R_1 \bowtie (S_1 \cup S_2)) \cup (R_2 \bowtie (S_1 \cup S_2))$$

$$(R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup (R_2 \bowtie (S_1 \cup S_2))$$

$$(R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup (R_2 \bowtie S_1) \cup (R_2 \bowtie S_2)$$

$$(R_1 \cup ... \cup R_N) \bowtie (S_1 \cup ... \cup S_K)$$

$$=$$

$$(R_1 \bowtie S_1) \cup ... \cup (R_1 \bowtie S_K)$$

$$... \cup ... \cup ...$$

$$(R_N \bowtie S_1) \cup ... \cup (R_N \bowtie S_K)$$

	S_0	S_1	S_2	S_3	S_4
R_0	$R_0 \bowtie S_0$	$R_0 \bowtie S_1$	$R_0 \bowtie S_2$	$R_0 \bowtie S_3$	$R_0 \bowtie S_4$
R_1	$R_1 \bowtie S_0$	$R_1 \bowtie S_1$	$R_1 \bowtie S_2$	$R_1 \bowtie S_3$	$R_1 \bowtie S_4$
R_2	$R_2 \bowtie S_0$	$R_2 \bowtie S_1$	$R_2 \bowtie S_2$	$R_2 \bowtie S_3$	$R_2 \bowtie S_4$
R_3	$R_3 \bowtie S_0$	$R_3 \bowtie S_1$	$R_3 \bowtie S_2$	$R_3 \bowtie S_3$	$R_3 \bowtie S_4$
R_4	$R_4 \bowtie S_0$	$R_4 \bowtie S_1$	$R_4 \bowtie S_2$	$R_4 \bowtie S_3$	$R_4 \bowtie S_4$

	S_0	S_1	S_2	S_3	S_4
R_0	$R_0 \bowtie S_0$	$R_0 \bowtie S_1$	$R_0 \bowtie S_2$	$R_0 \bowtie S_3$	$R_0 \bowtie S_4$
R_1	$R_1 \bowtie S_0$	$R_1 \bowtie S_1$	$R_1 \bowtie S_2$	$R_1 \bowtie S_3$	$R_1 \bowtie S_4$
R_2	$R_2 \bowtie S_0$	$R_2 \bowtie S_1$	$R_2 \bowtie S_2$	$R_2 \bowtie S_3$	$R_2 \bowtie S_4$
R_3	$R_3 \bowtie S_0$	$R_3 \bowtie S_1$	$R_3 \bowtie S_2$	$R_3 \bowtie S_3$	$R_3 \bowtie S_4$
R_4	$R_4 \bowtie S_0$	$R_4 \bowtie S_1$	$R_4 \bowtie S_2$	$R_4 \bowtie S_3$	$R_4 \bowtie S_4$

N workers gets us $\sqrt{N}$ scaling

Partitioning Data

How do we pick what goes into each partition?

- Arbitrary
- Range
- Hash

Hash Partitioning

For $R \underset{A}{\bowtie} S$:

- $R_i \leftarrow \sigma_{\text{hash}(A)=i}(R)$
- $S_i \leftarrow \sigma_{\text{hash}(A)=i}(S)$

Distributed Joins

	S_0	S_1	S_2	S_3	S_4
R_0	$R_0 \bowtie S_0$	$R_0 \bowtie S_1$	$R_0 \bowtie S_2$	$R_0 \bowtie S_3$	$R_0 \bowtie S_4$
R_1	$R_1 \bowtie S_0$	$R_1 \bowtie S_1$	$R_1 \bowtie S_2$	$R_1 \bowtie S_3$	$R_1 \bowtie S_4$
R_2	$R_2 \bowtie S_0$	$R_2 \bowtie S_1$	$R_2 \bowtie S_2$	$R_2 \bowtie S_3$	$R_2 \bowtie S_4$
R_3	$R_3 \bowtie S_0$	$R_3 \bowtie S_1$	$R_3 \bowtie S_2$	$R_3 \bowtie S_3$	$R_3 \bowtie S_4$
R_4	$R_4 \bowtie S_0$	$R_4 \bowtie S_1$	$R_4 \bowtie S_2$	$R_4 \bowtie S_3$	$R_4 \bowtie S_4$

Distributed Joins

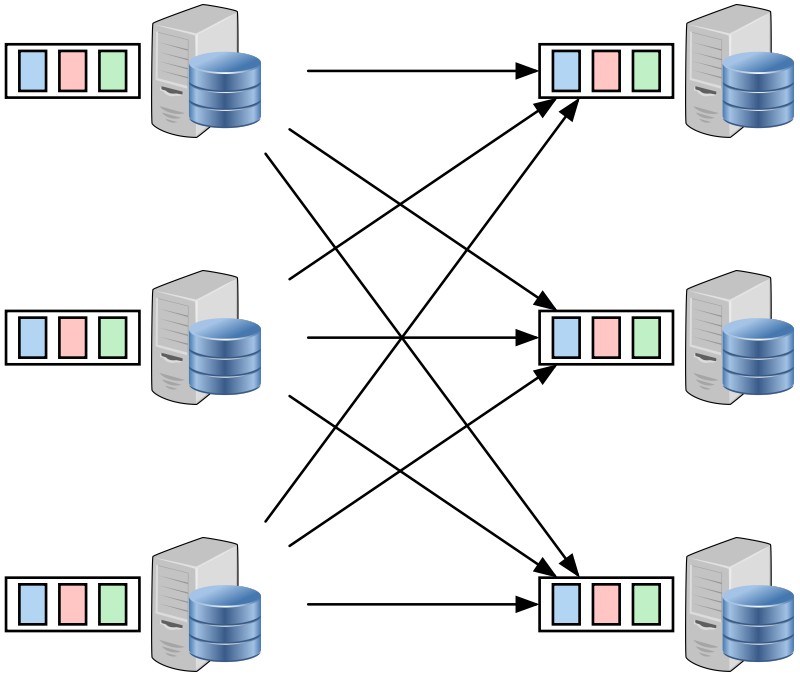
	S_0	S_1	S_2	S_3	S_4
R_0	$R_0 \bowtie S_0$	$R_0 \bowtie S_1$	$R_0 \bowtie S_2$	$R_0 \bowtie S_3$	$R_0 \bowtie S_4$
R_1	$R_1 \bowtie S_0$	$R_1 \bowtie S_1$	$R_1 \bowtie S_2$	$R_1 \bowtie S_3$	$R_1 \bowtie S_4$
R_2	$R_2 \bowtie S_0$	$R_2 \bowtie S_1$	$R_2 \bowtie S_2$	$R_2 \bowtie S_3$	$R_2 \bowtie S_4$
R_3	$R_3 \bowtie S_0$	$R_3 \bowtie S_1$	$R_3 \bowtie S_2$	$R_3 \bowtie S_3$	$R_3 \bowtie S_4$
R_4	$R_4 \bowtie S_0$	$R_4 \bowtie S_1$	$R_4 \bowtie S_2$	$R_4 \bowtie S_3$	$R_4 \bowtie S_4$

	S_0	S_1	S_2	S_3	S_4
R_0	$R_0 \bowtie S_0$	$R_0 \bowtie S_1$	$R_0 \bowtie S_2$	$R_0 \bowtie S_3$	$R_0 \bowtie S_4$
R_1	$R_1 \bowtie S_0$	$R_1 \bowtie S_1$	$R_1 \bowtie S_2$	$R_1 \bowtie S_3$	$R_1 \bowtie S_4$
R_2	$R_2 \bowtie S_0$	$R_2 \bowtie S_1$	$R_2 \bowtie S_2$	$R_2 \bowtie S_3$	$R_2 \bowtie S_4$
R_3	$R_3 \bowtie S_0$	$R_3 \bowtie S_1$	$R_3 \bowtie S_2$	$R_3 \bowtie S_3$	$R_3 \bowtie S_4$
R_4	$R_4 \bowtie S_0$	$R_4 \bowtie S_1$	$R_4 \bowtie S_2$	$R_4 \bowtie S_3$	$R_4 \bowtie S_4$

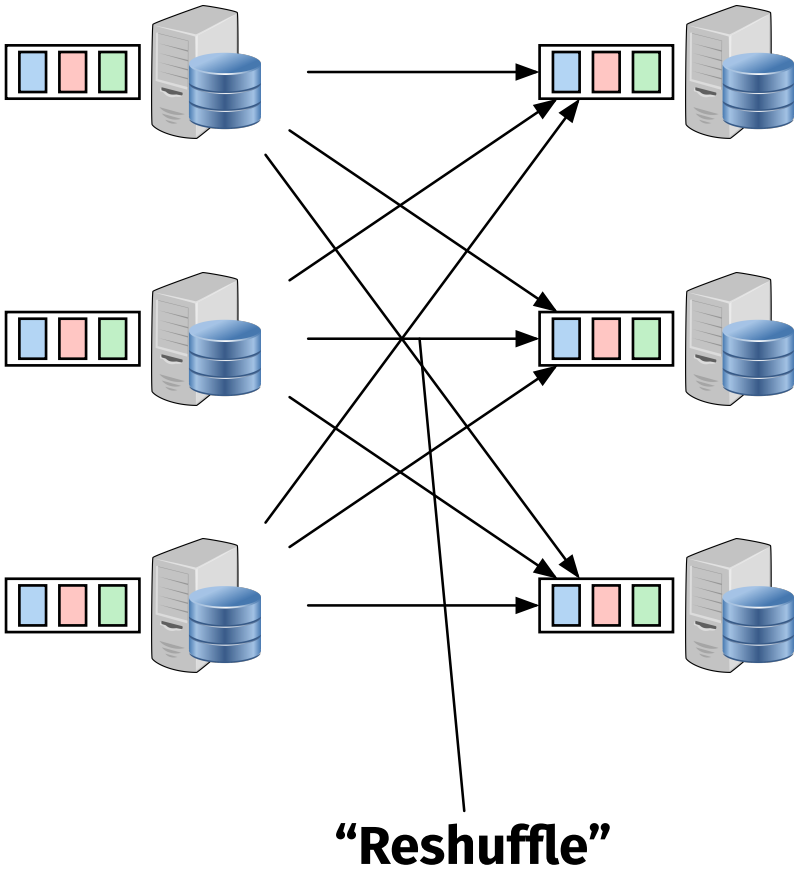
Hash gets us N scaling for N workers

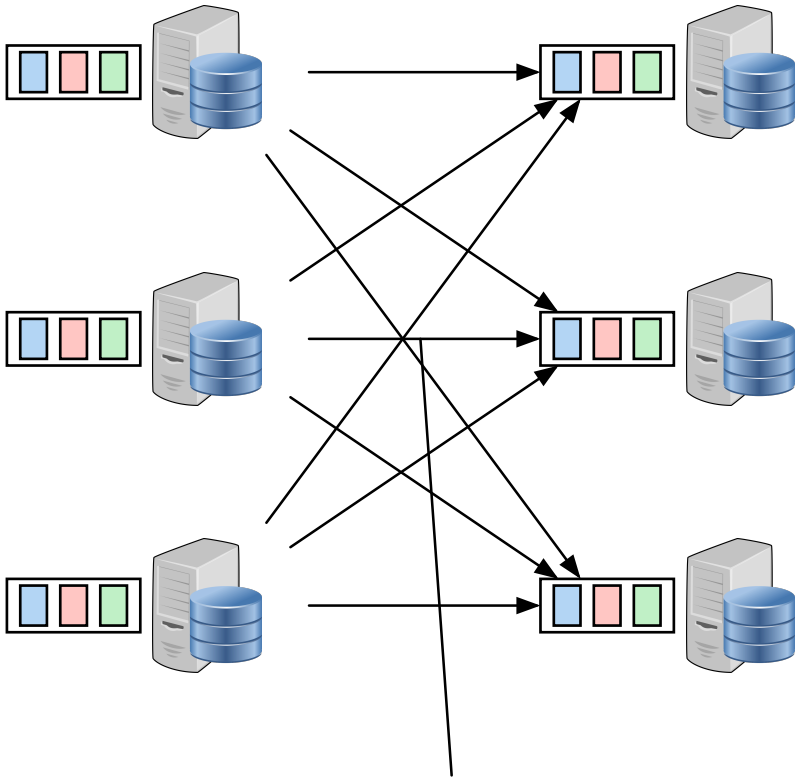
What if the partitions aren’t aligned so nicely?

Distributed Joins



Distributed Joins





“Reshuffle”

(Each tuple goes to exactly one place)

Reshuffling

- R_i is initially located at **X**
- S_i is initially located at **Y**

Then...

1. Copy S_i from **Y** to **X**
2. Compute $R_i \bowtie S_i$ at **X**

Can we reduce network use more?

Distributed Joins

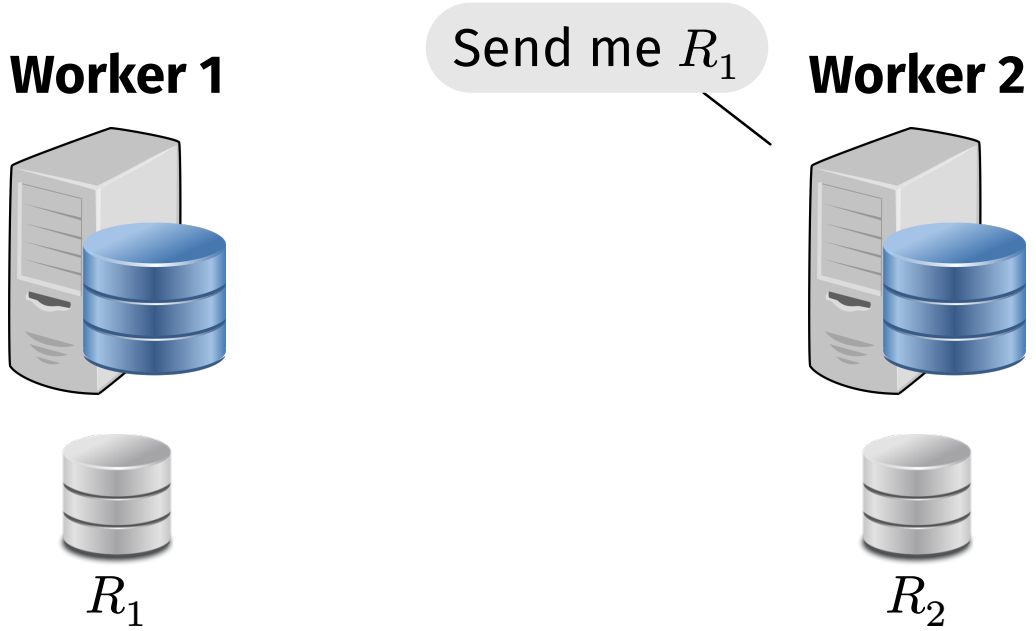
Worker 1

R_1

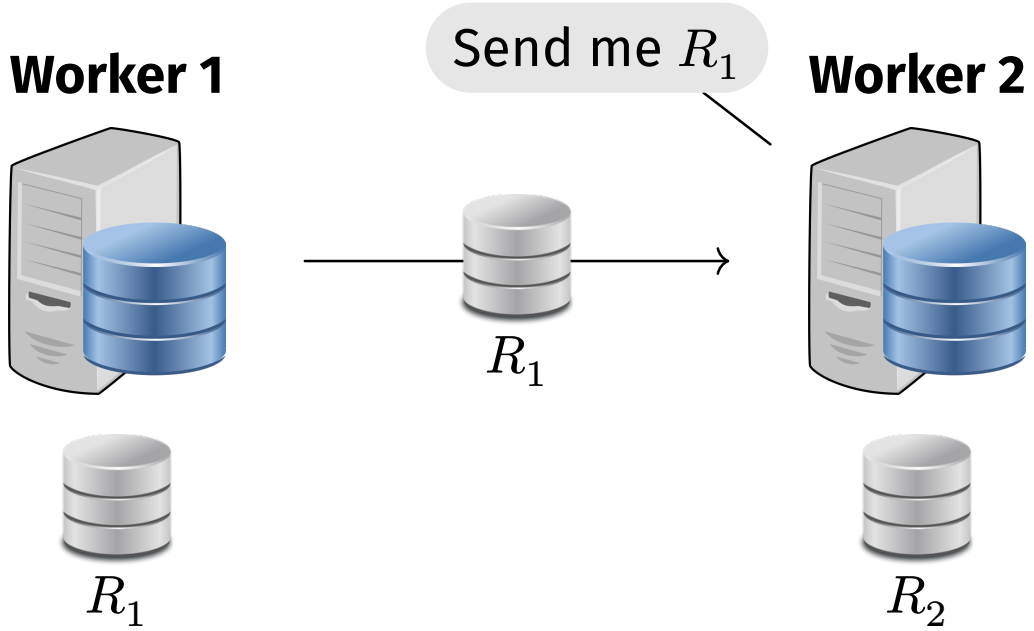
Worker 2

R_2

Distributed Joins



Distributed Joins



Distributed Joins

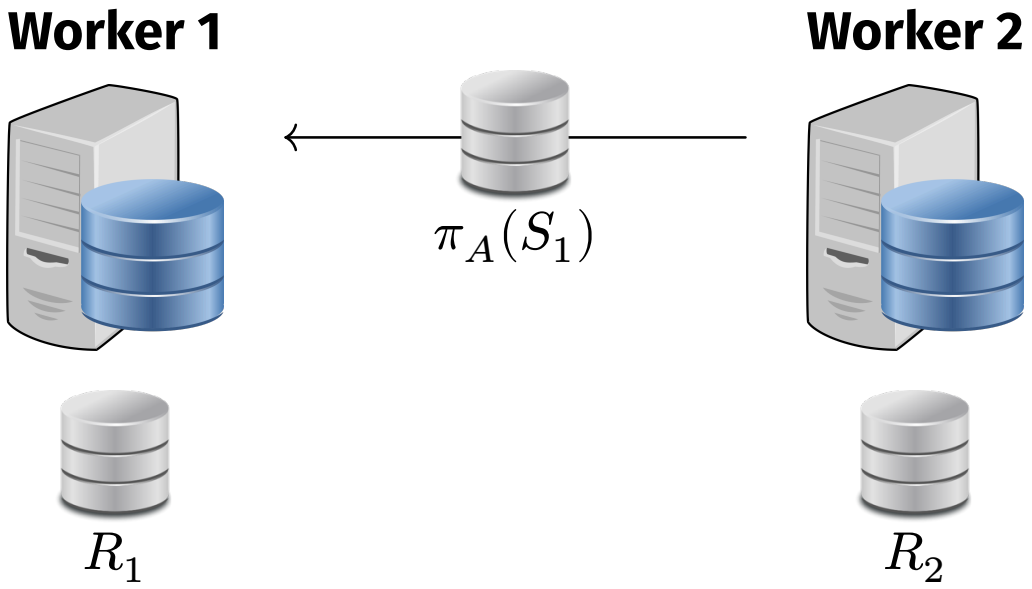
Worker 1

R_1

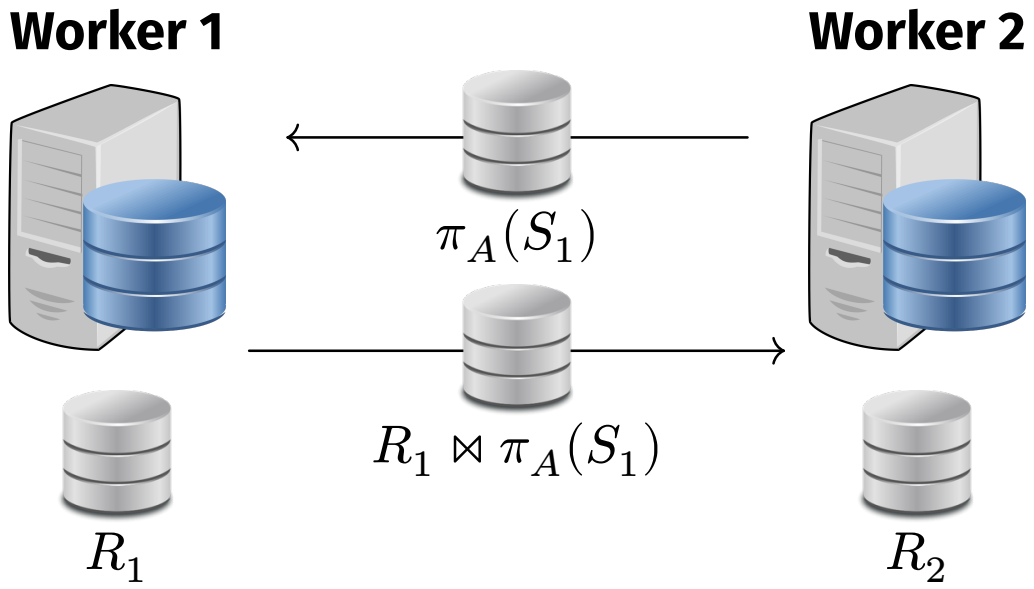
Worker 2

R_2

Distributed Joins



Distributed Joins



Semi-join

Worker 1

<M, 1>
<N, 2>
<O, 2>
<P, 3>
<Q, 4>

Worker 2

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

Semi-join

Worker 1

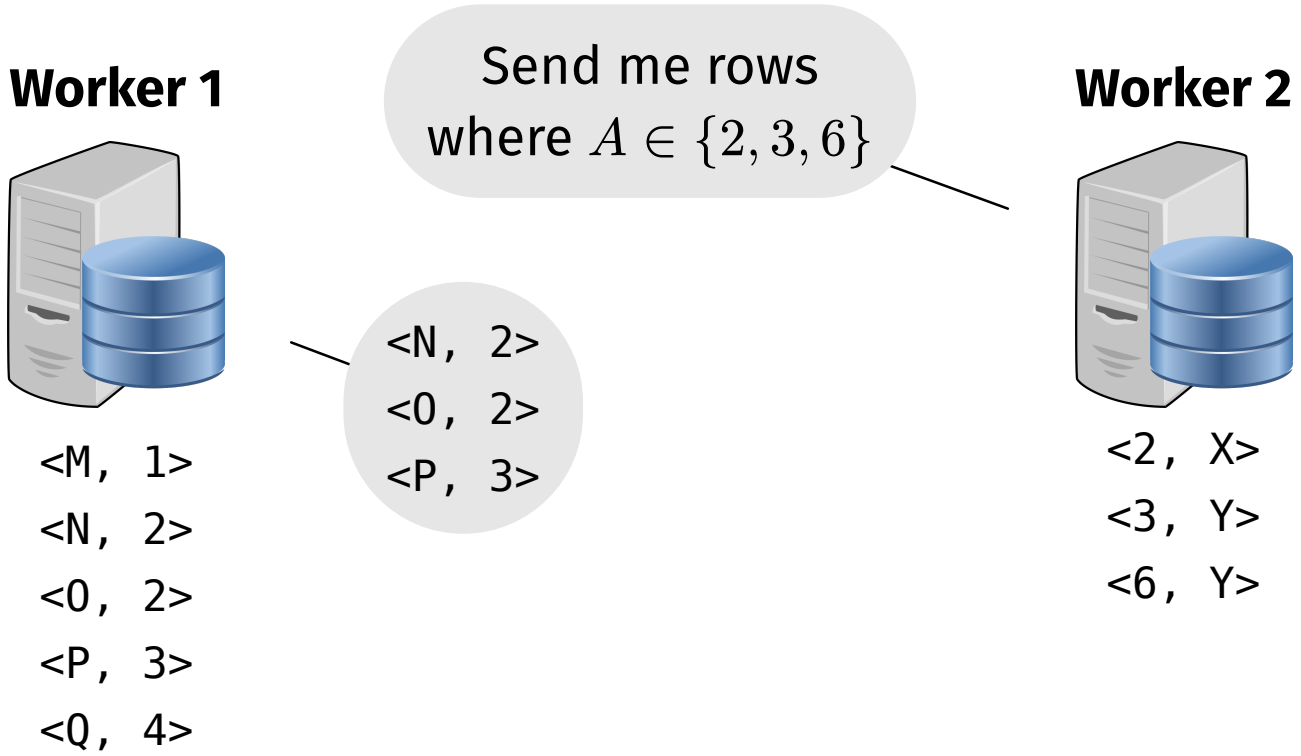
```
<M, 1>
<N, 2>
<O, 2>
<P, 3>
<Q, 4>
```

Send me rows where $A \in \{2, 3, 6\}$

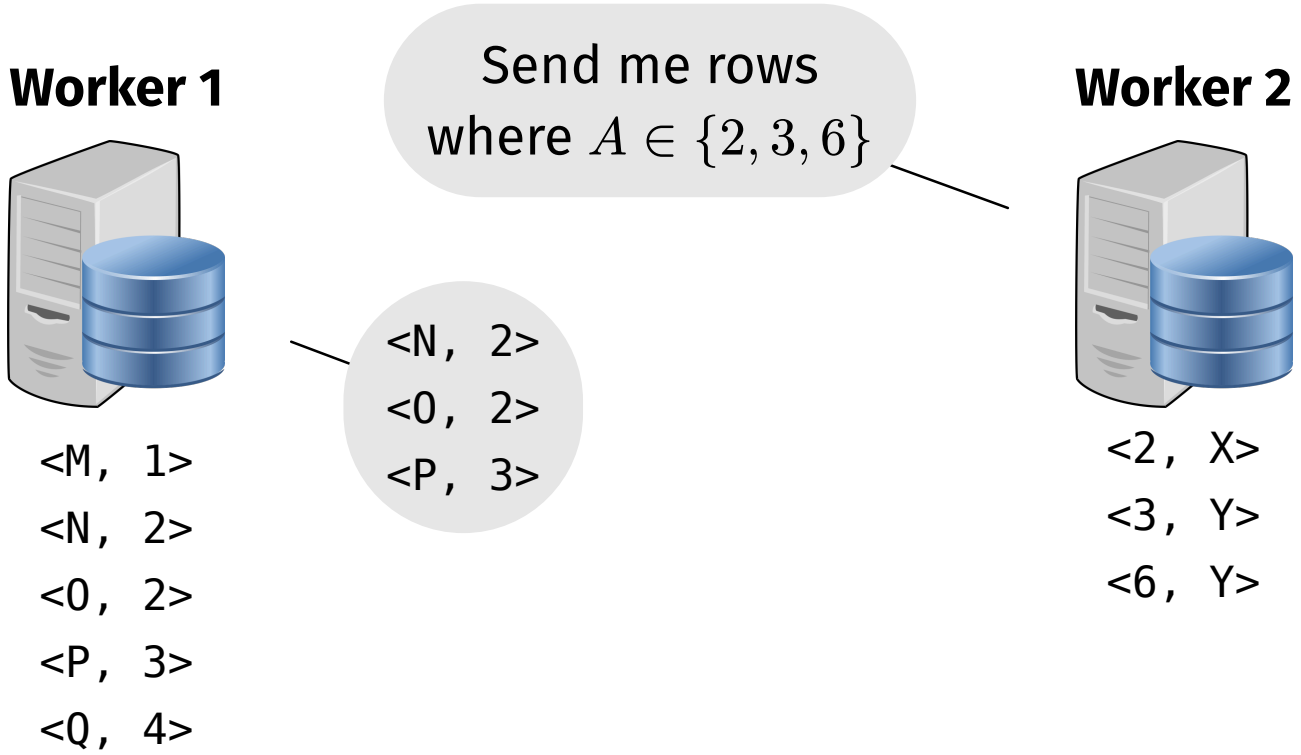
Worker 2

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

Semi-join


Worker 1
Send me rows where $A \in \{2, 3, 6\}$
Worker 2
<N, 2>
<O, 2>
<P, 3>
<M, 1>
<N, 2>
<O, 2>
<P, 3>
<Q, 4>
<2, X>
<3, Y>
<6, Y>

Semi-join



This is called a **semi-join**

Problem: Worker 2 is sending a lot of data

Problem: Worker 2 is sending a lot of data

Idea: Compress $\pi_A(S_1)$ using a bloom filter

Bloom Join

Worker 1

<M, 1>
<N, 2>
<O, 2>
<P, 3>
<Q, 4>

Worker 2

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

Bloom Join

Worker 1

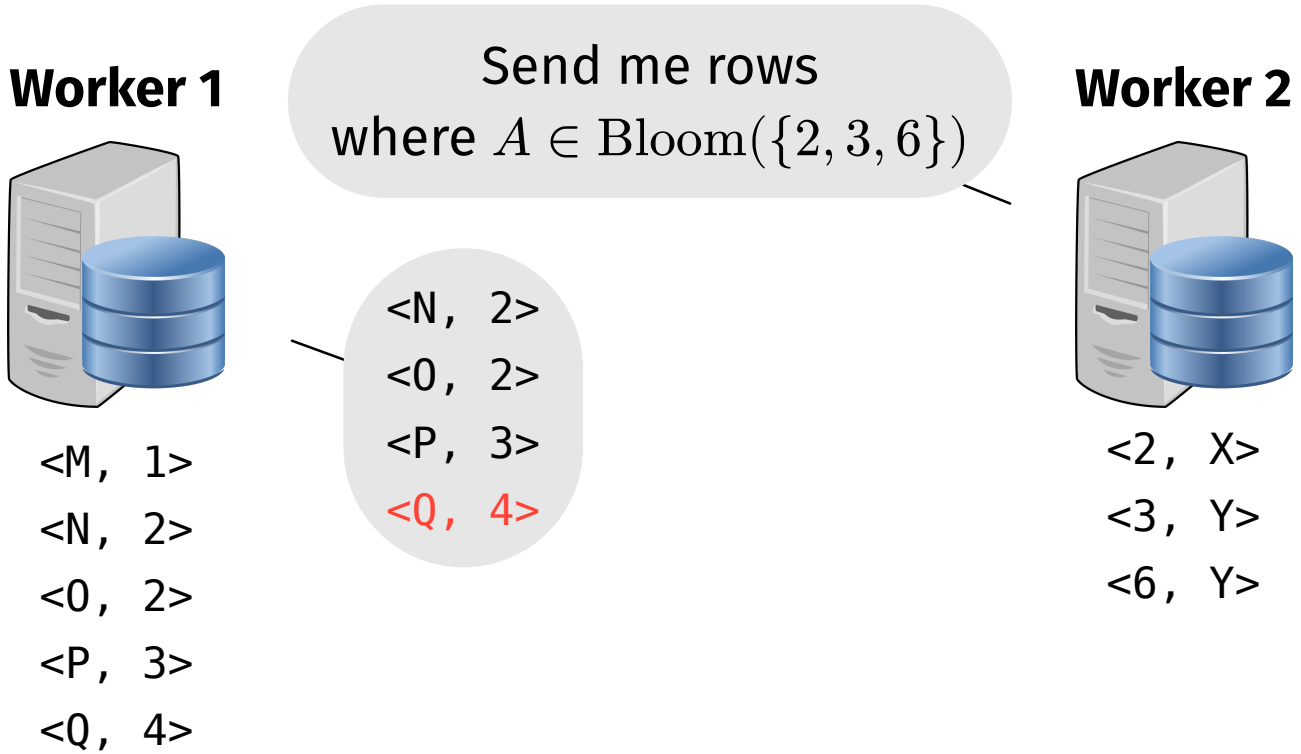
```
<M, 1>
<N, 2>
<O, 2>
<P, 3>
<Q, 4>
```

Send me rows
where $A \in \text{Bloom}(\{2, 3, 6\})$

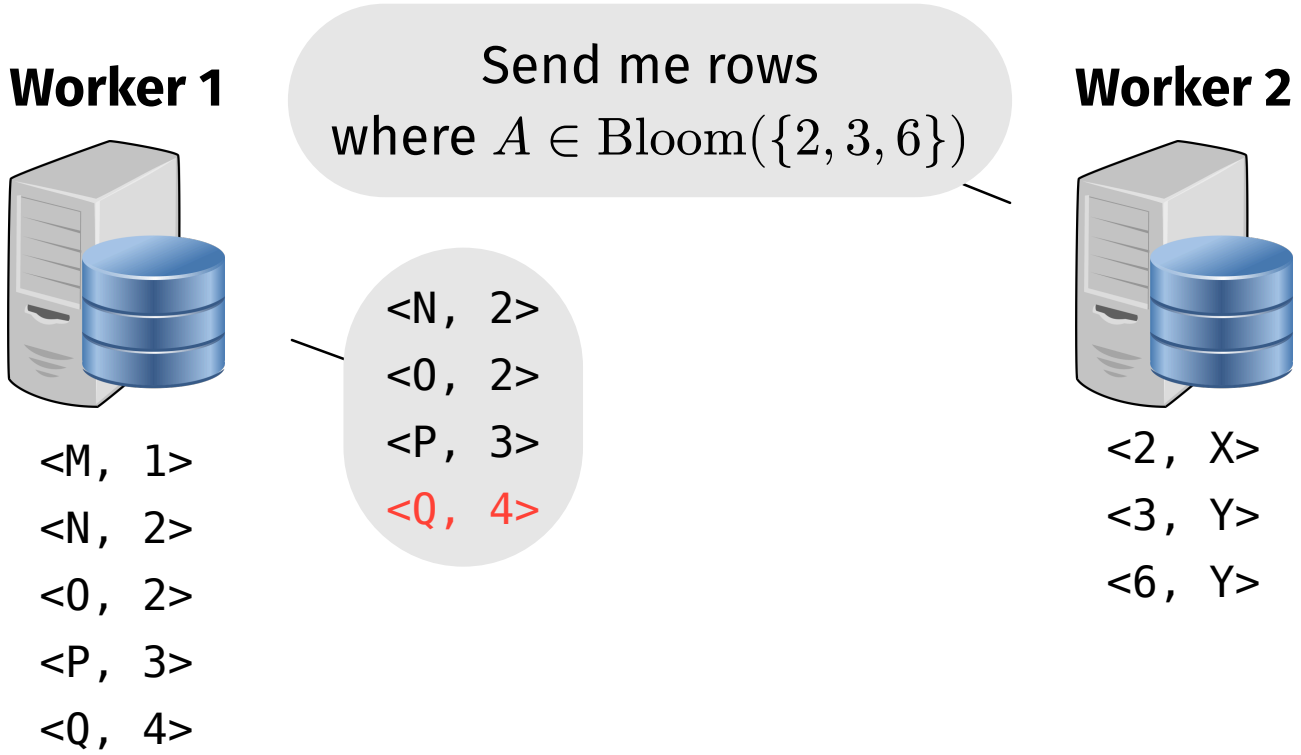
Worker 2

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

Bloom Join


Worker 1
Send me rows
where $A \in \text{Bloom}(\{2, 3, 6\})$
Worker 2
<N, 2>
<O, 2>
<P, 3>
<Q, 4>
<M, 1>
<N, 2>
<O, 2>
<P, 3>
<Q, 4>
<2, X>
<3, Y>
<6, Y>

Bloom Join



This is called a **Bloom Join**

Updates

- Deadline for checkpoint 3 extended by one week
- No class on Thursday (video lecture to be released soon)