Parallel DBs

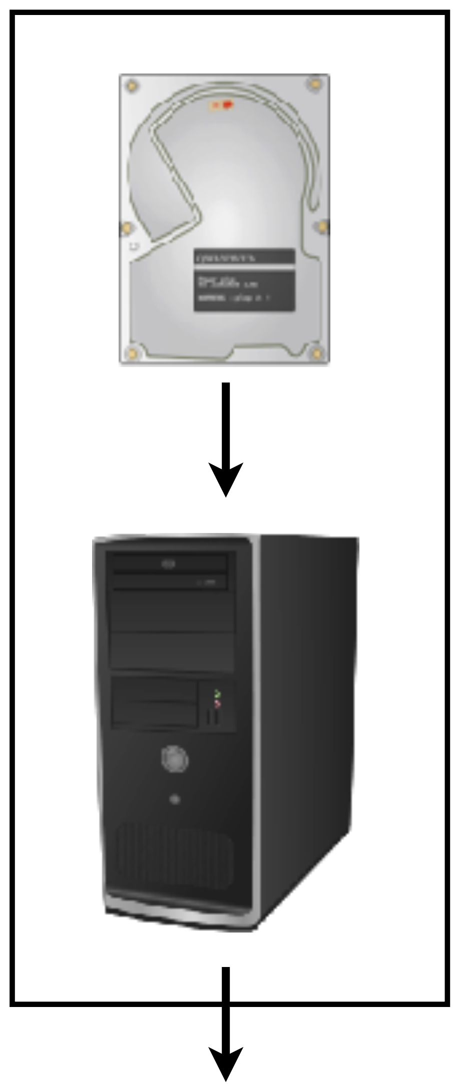
April 23, 2018

Why Scale?

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

Why Scale Up?

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



~1 Hour

Why Scale Up?

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

~1 Hour

~3.5 Seconds

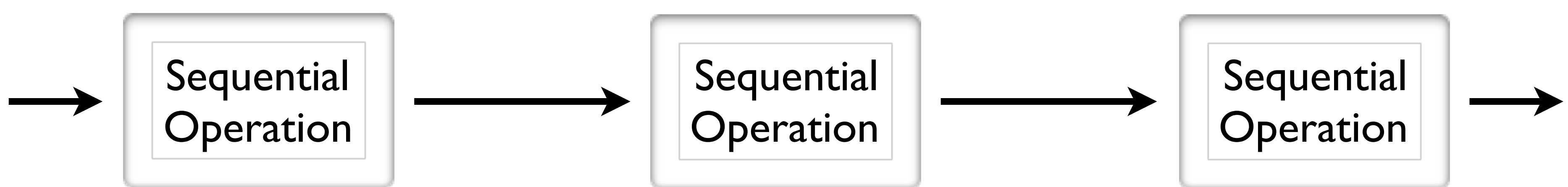
Data Parallelism

Replication

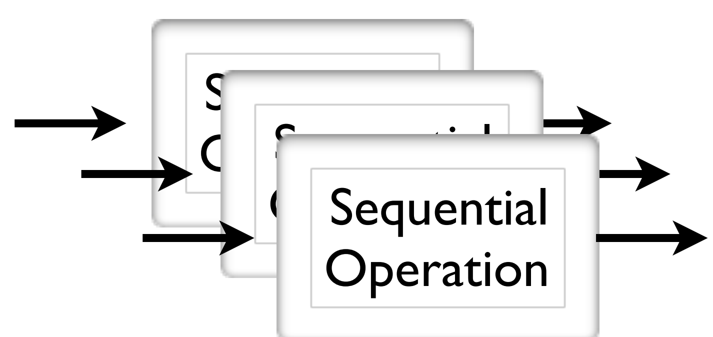
Partitioning

Operator Parallelism

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



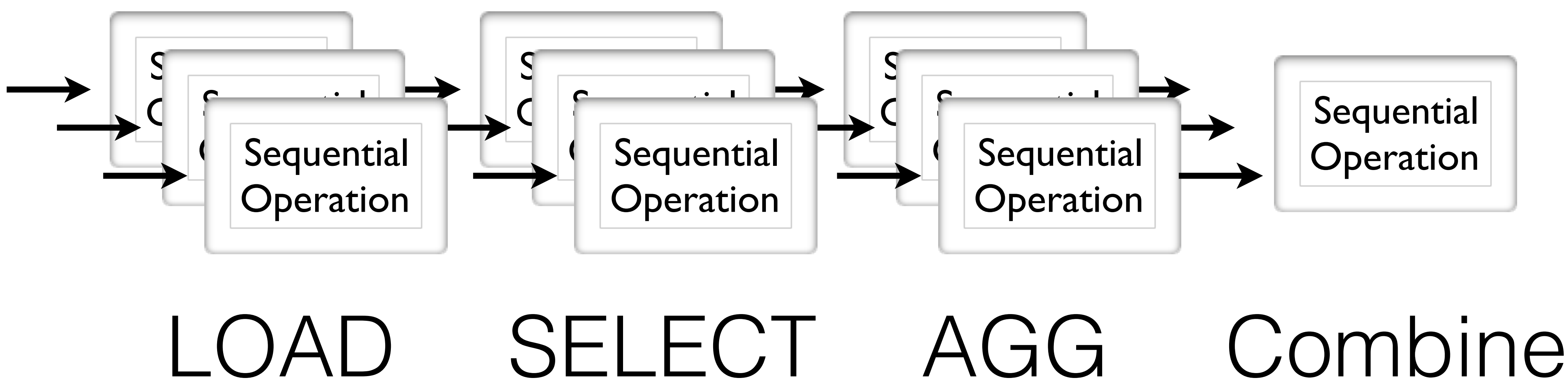
- Partition Parallelism: Many machines doing the same thing to different pieces of data.



Types of Parallelism

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

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

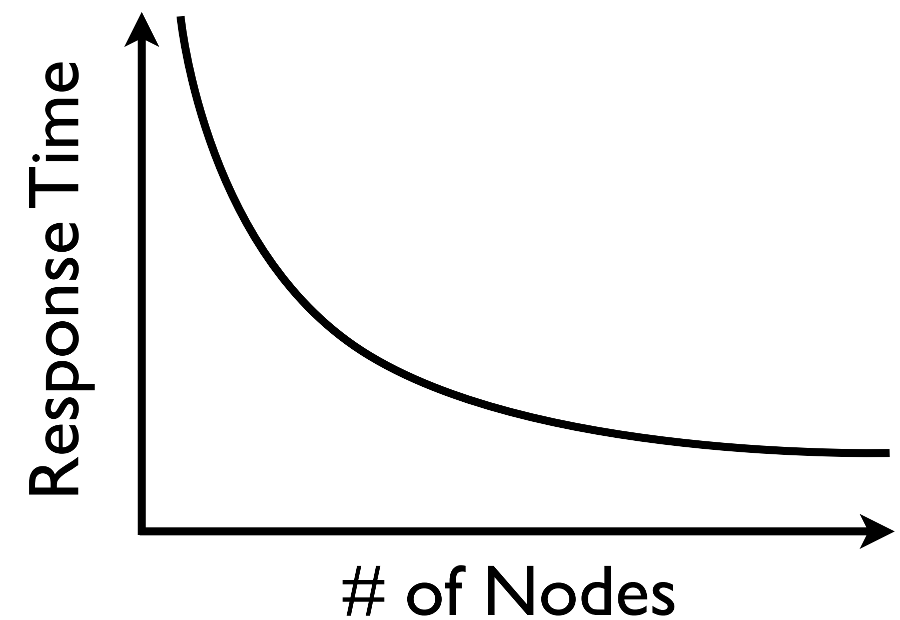


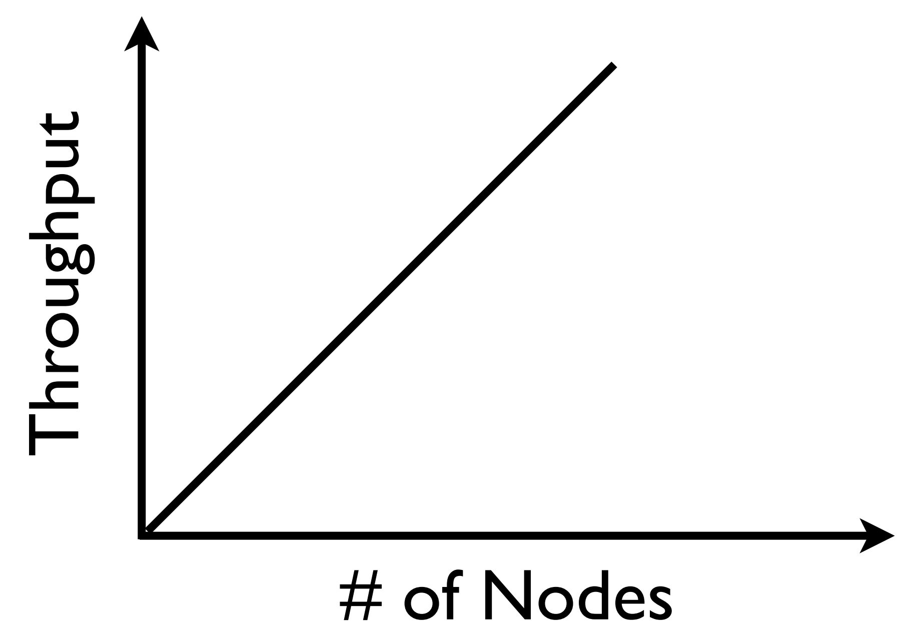
DBMSes: The First || Success Story

- 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

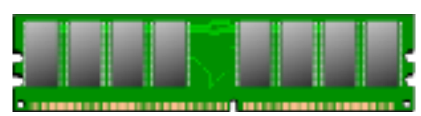
- Speed-up ||-ism
  - More resources = proportionally less time spent.
- Scale-up ||-ism
  - More resources = proportionally more data processed.





Parallelism Models

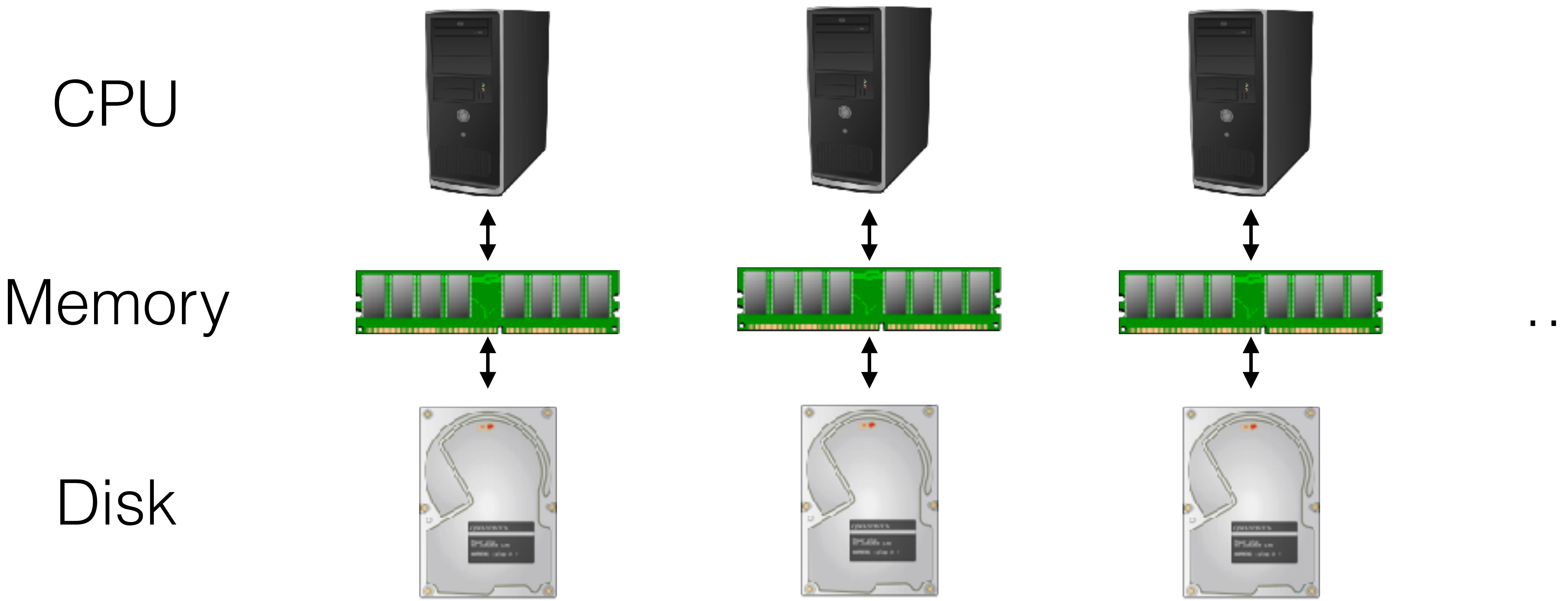
CPU



Memory

Disk

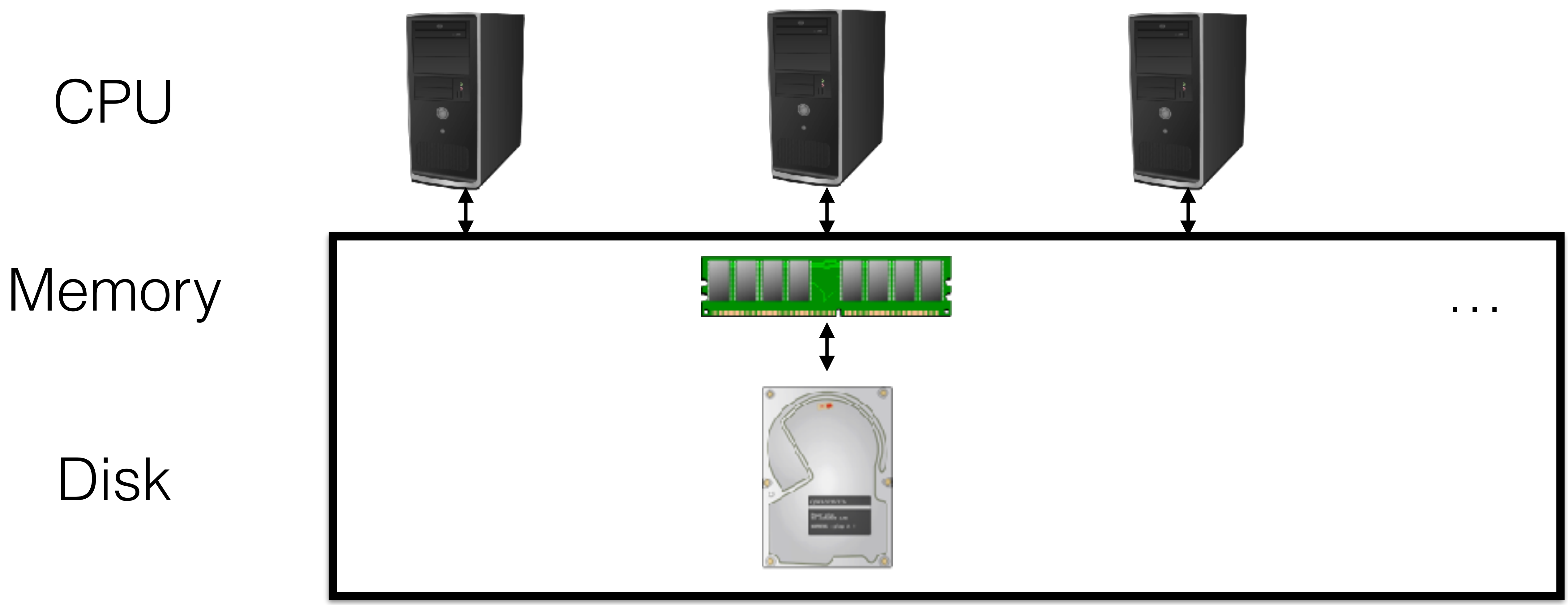
Parallelism Models



How do the nodes communicate?

Parallelism Models

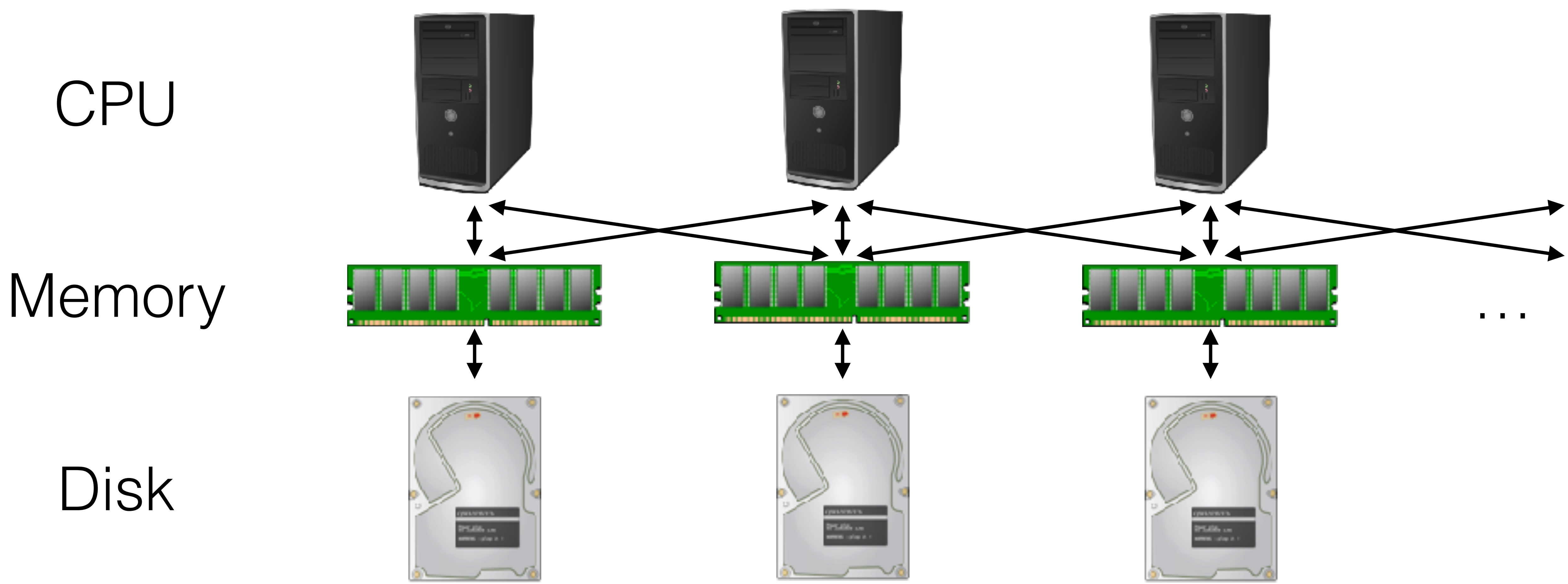
Option 1: “Shared Memory” available to all CPUs



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

Parallelism Models

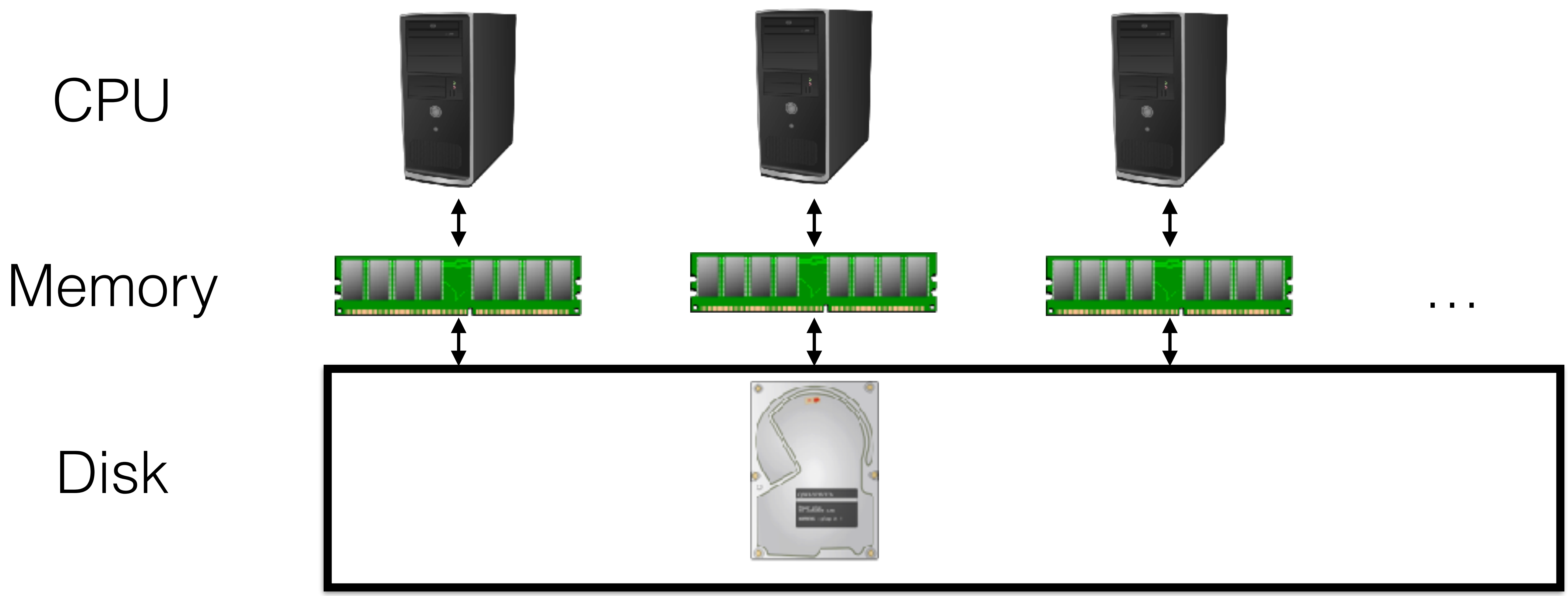
Option 2: Non-Uniform Memory Access.



Used by most AMD servers

Parallelism Models

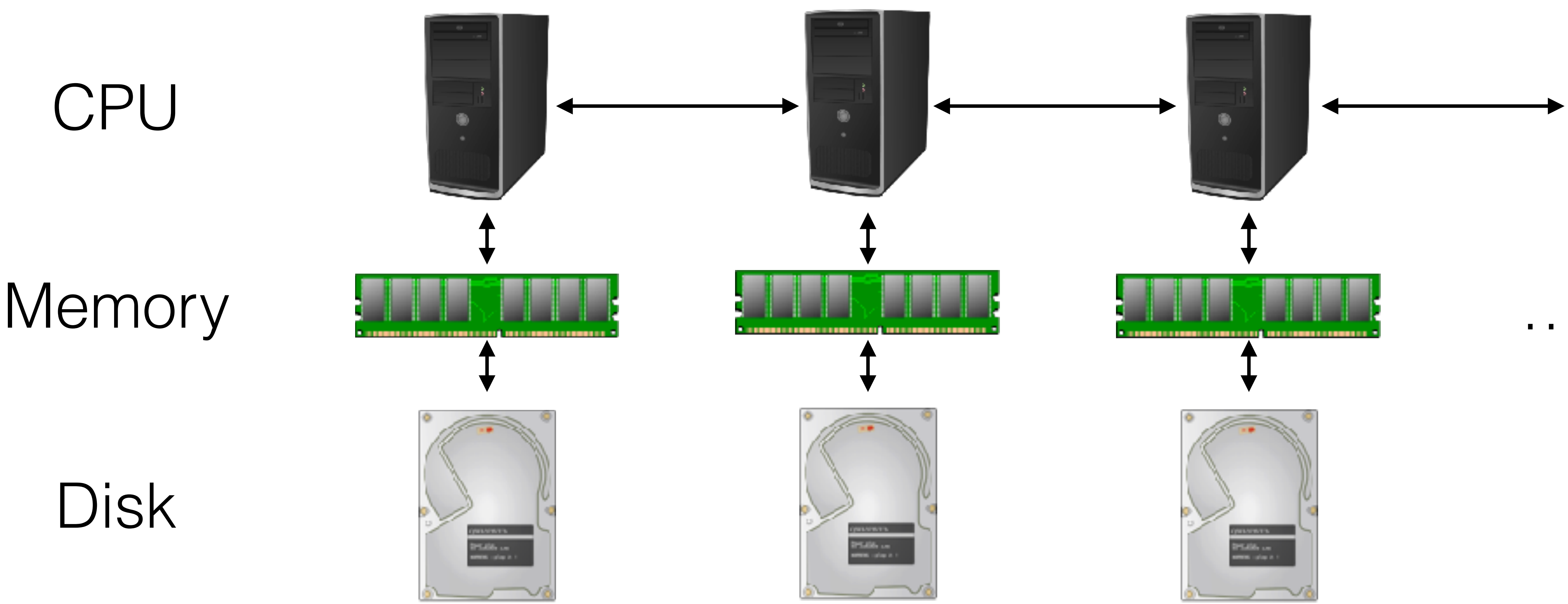
Option 3: "Shared Disk" available to all CPUs



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.

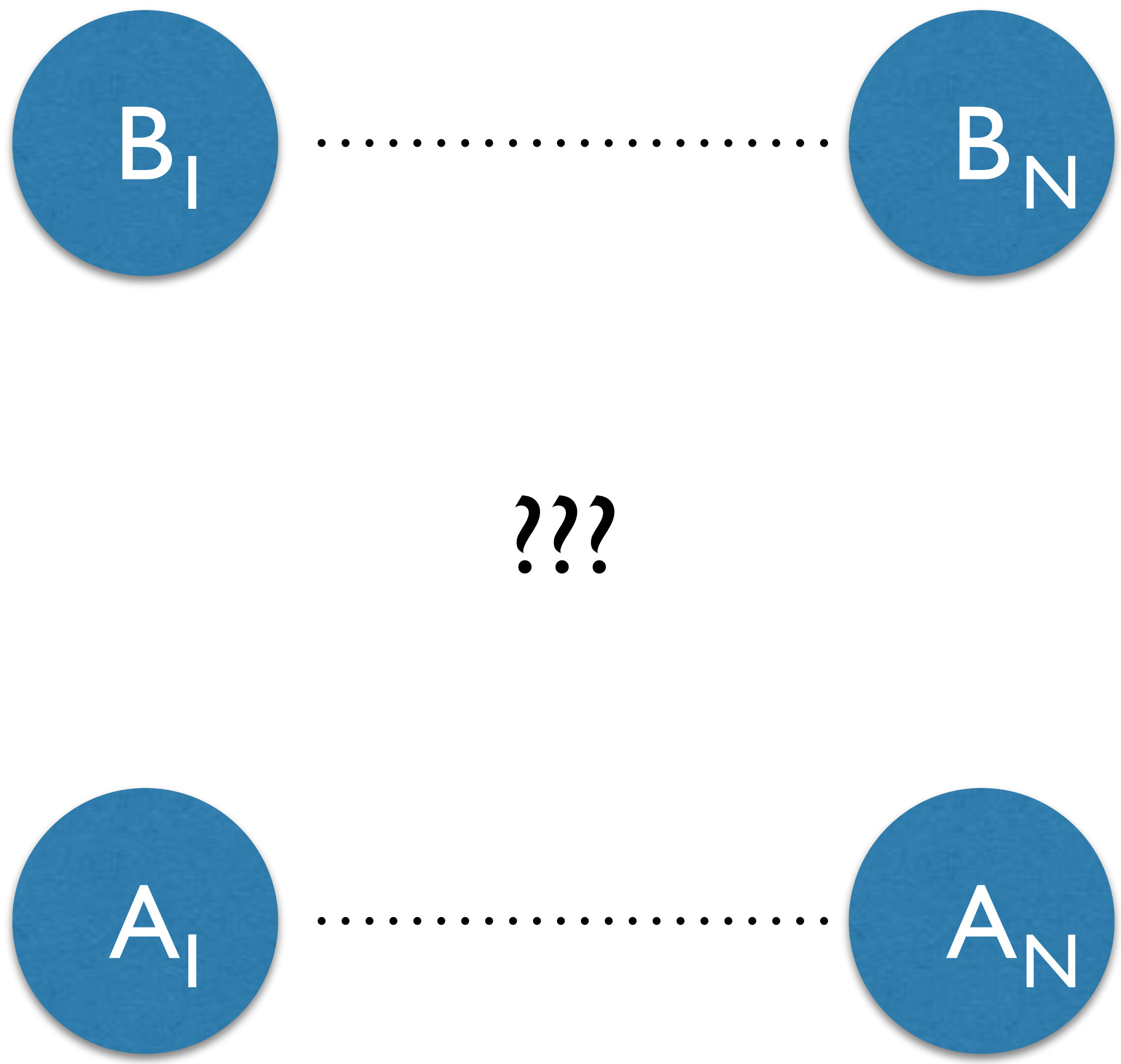
Parallel Data Flow

No Parallelism

Parallel Data Flow

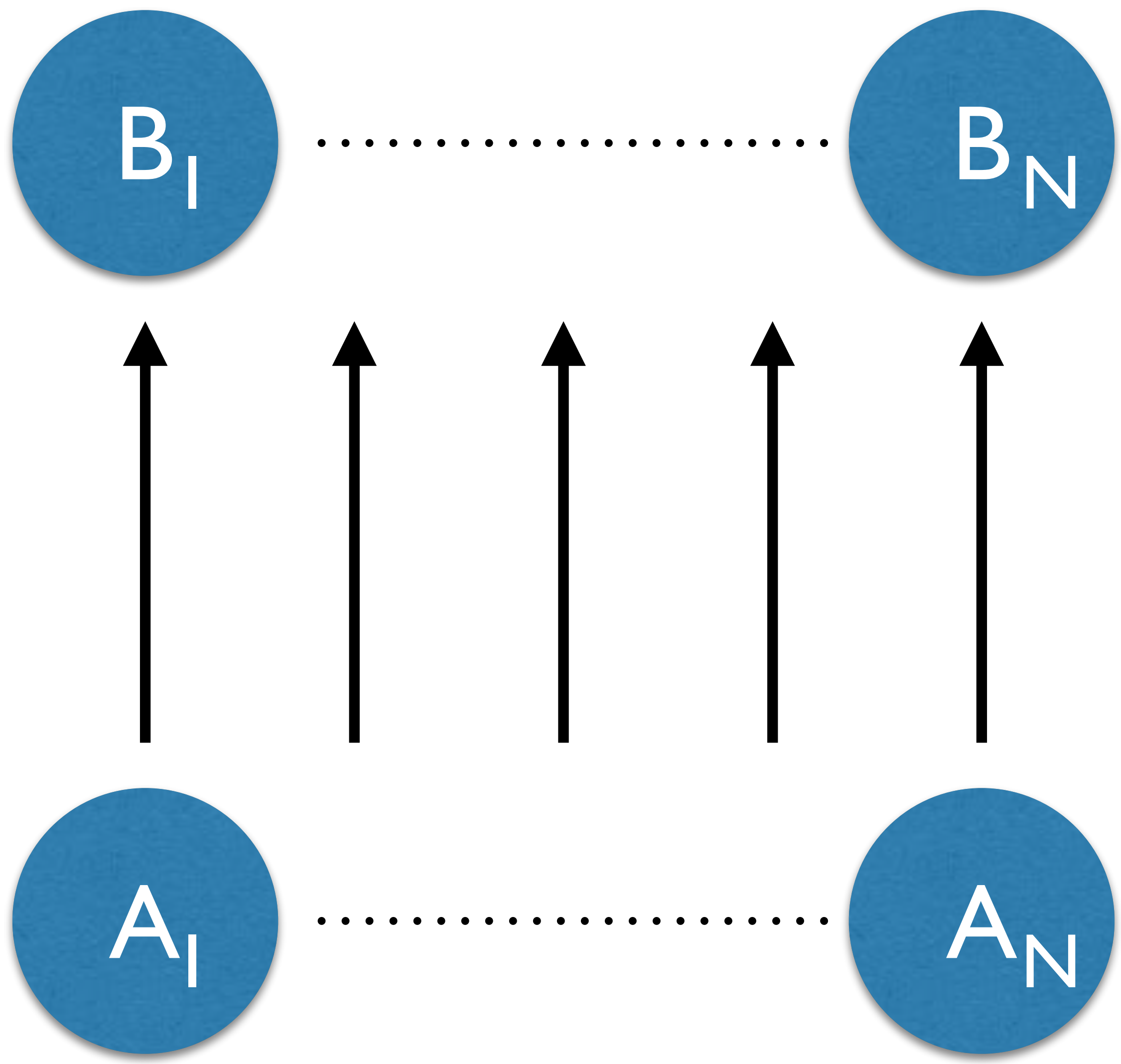
N-Way Parallelism

Parallel Data Flow



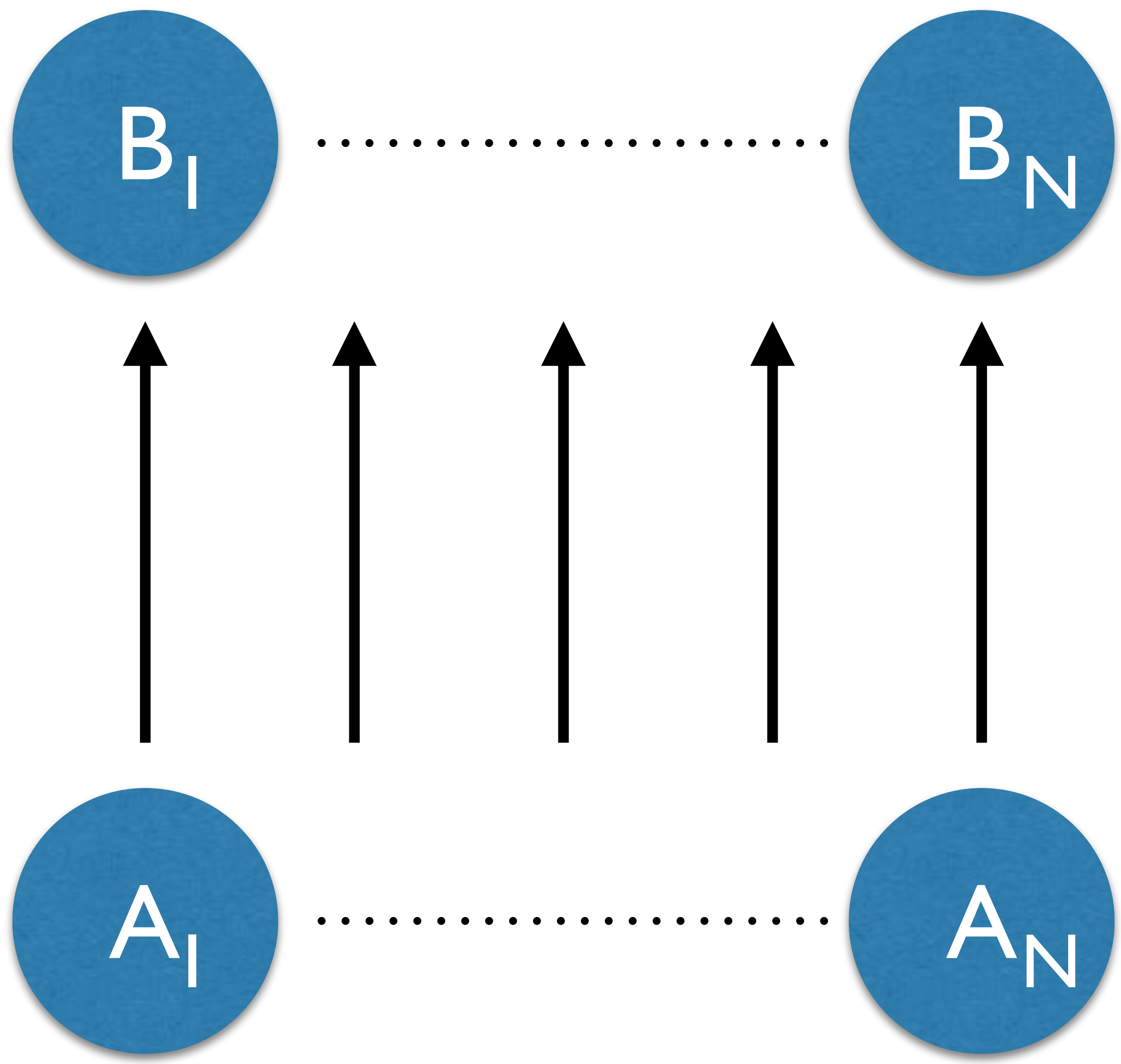
Chaining Parallel Operators

Parallel Data Flow



One-to-One Data Flow (“Map”)

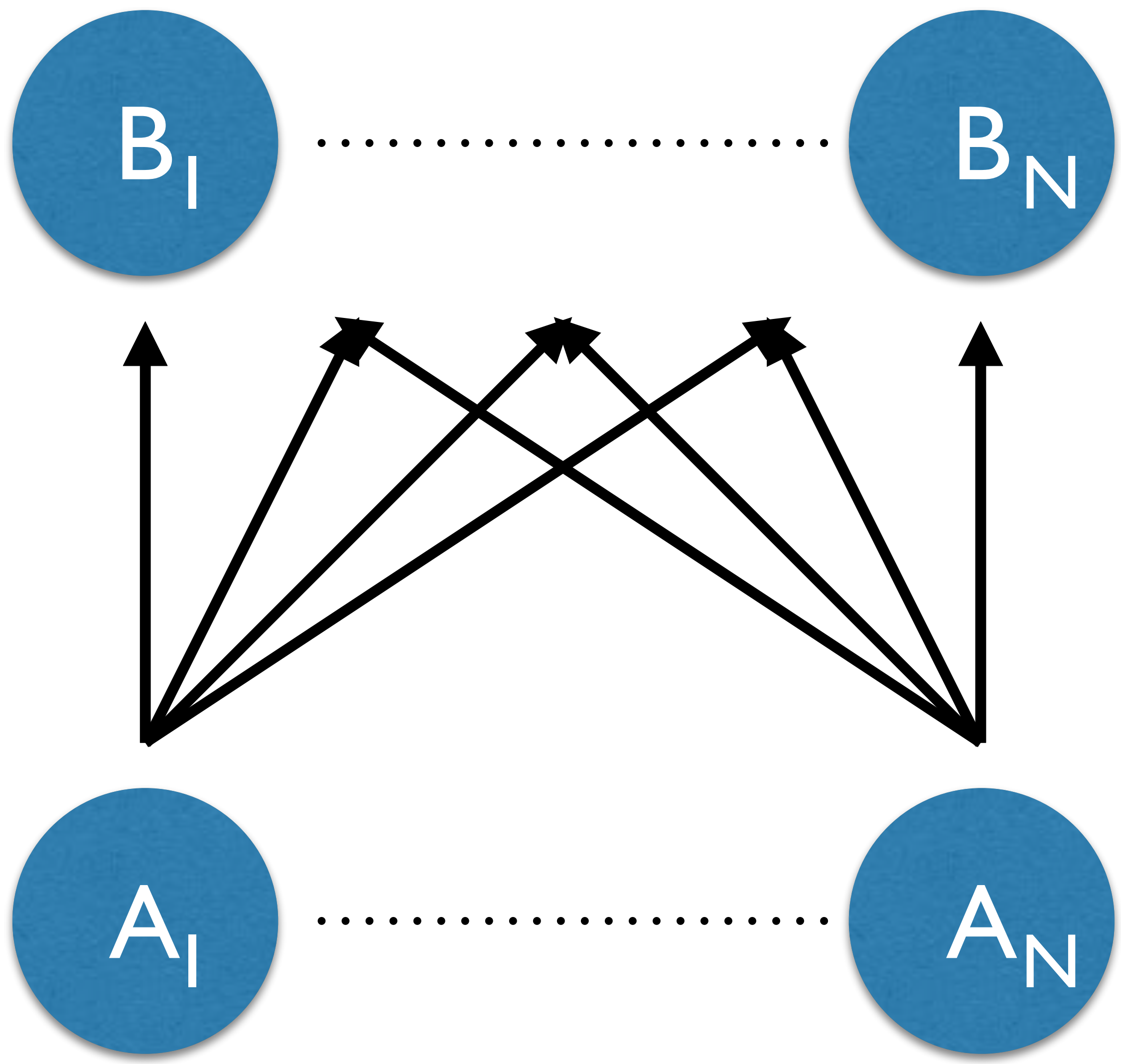
Parallel Data Flow



One-to-One Data Flow

Parallel Data Flow

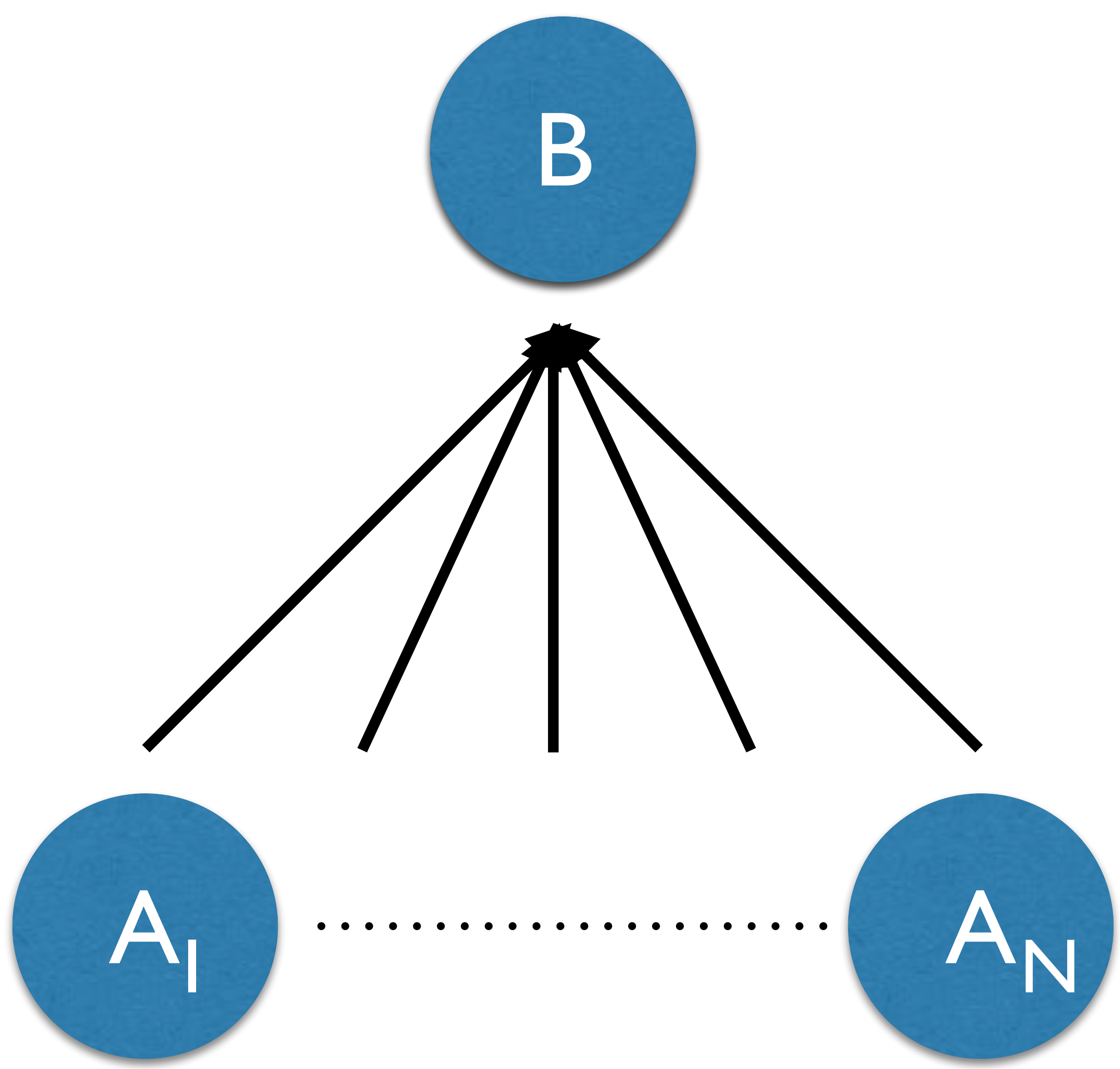
<u>Extreme 1</u>
All-to-All
All nodes send all records to all downstream nodes



Many-to-Many Data Flow

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

Parallel Data Flow



Many-to-One Data Flow (“Reduce/Fold”)

Parallel Operators

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

What is a logical "unit of computation"?

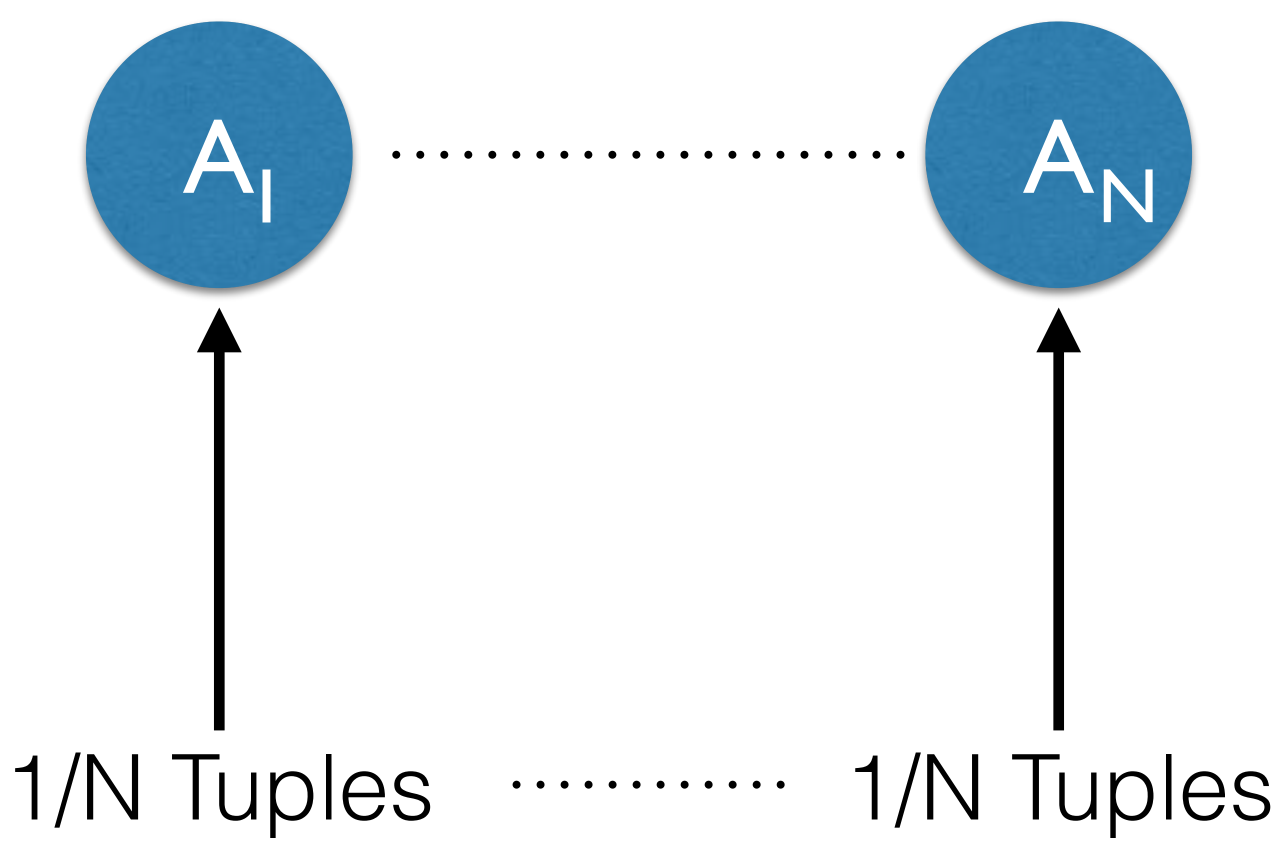
(1 tuple)

Is there a data dependency between units?

(no)

Parallel Operators

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



Parallel Joins

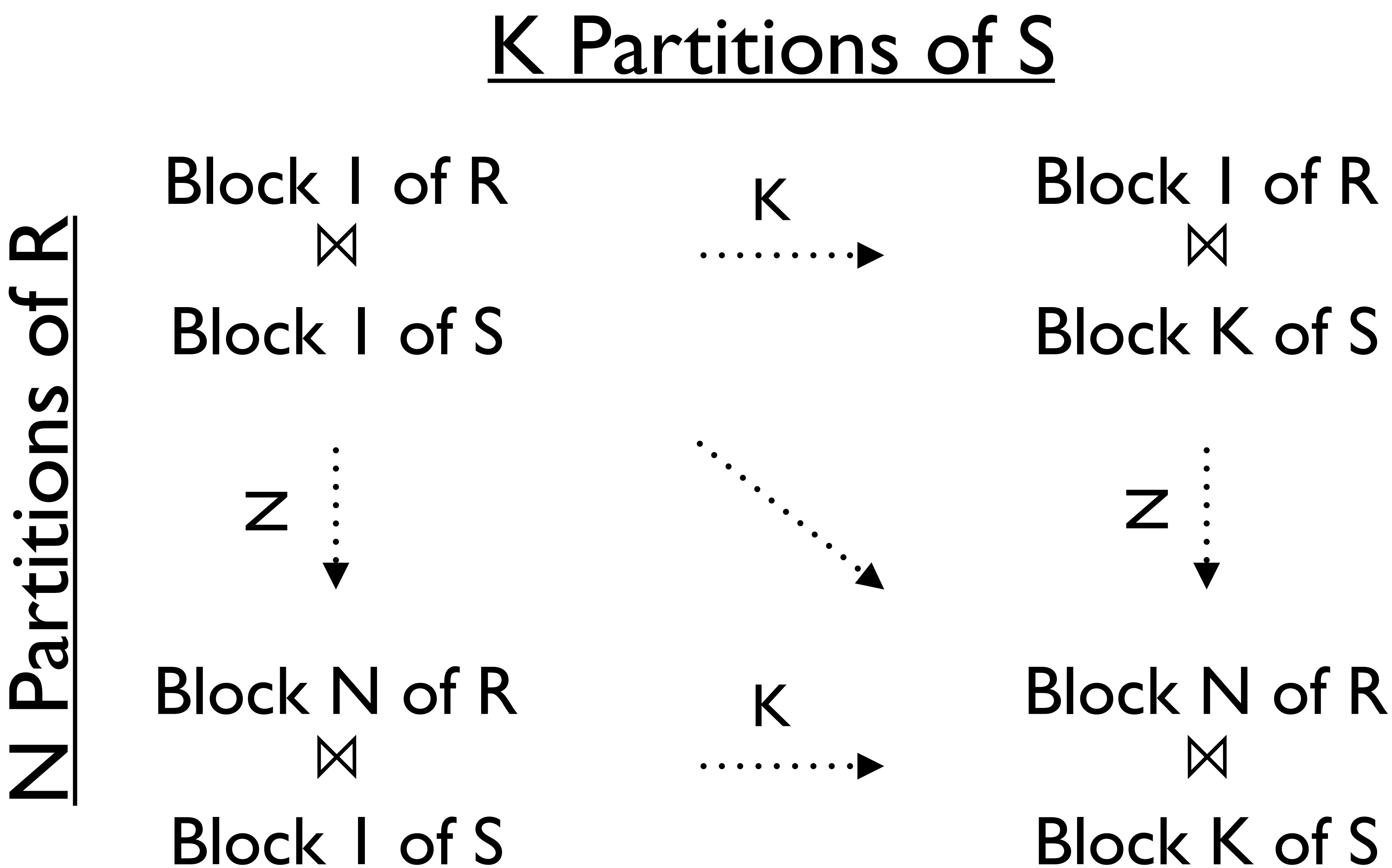
```
FOR i IN 1 to N
  FOR j IN 1 to K
    JOIN(Block i of R,
         Block j of S)
```

Partition

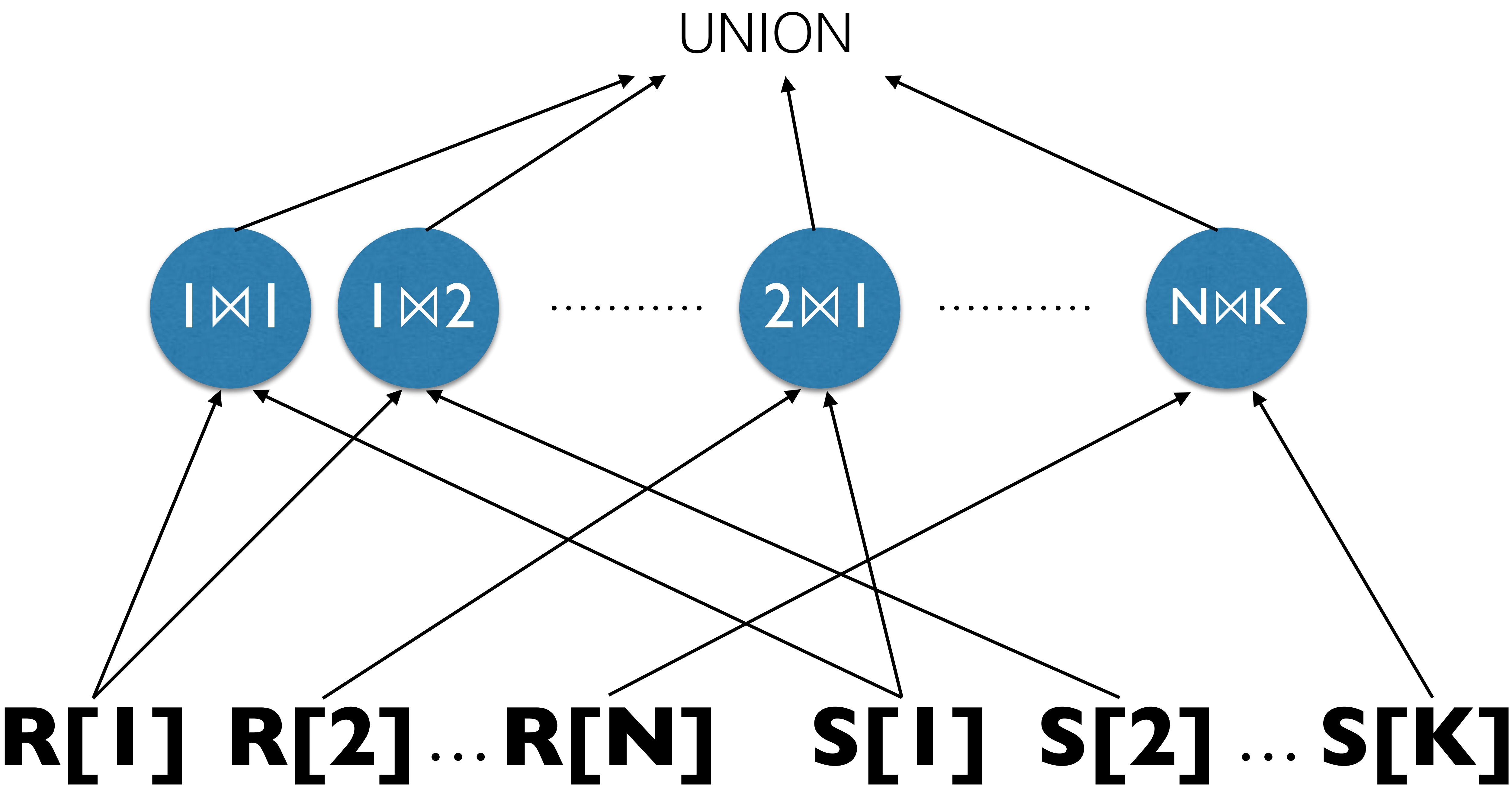
Partition

One Unit of Computation

Parallel Joins



Parallel Joins



Parallel Joins

How much data needs to be transferred?

How many “units of computation” do we create?

Parallel Joins

What if we partitioned “intelligently”?

Parallel Joins

Hash(R.B)%4 \ Hash(S.B)%4	0	1	2	3
0	✓	✗	✗	✗
1		✓		
2			✓	
3				✓

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	✗	✗	✗	✓

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

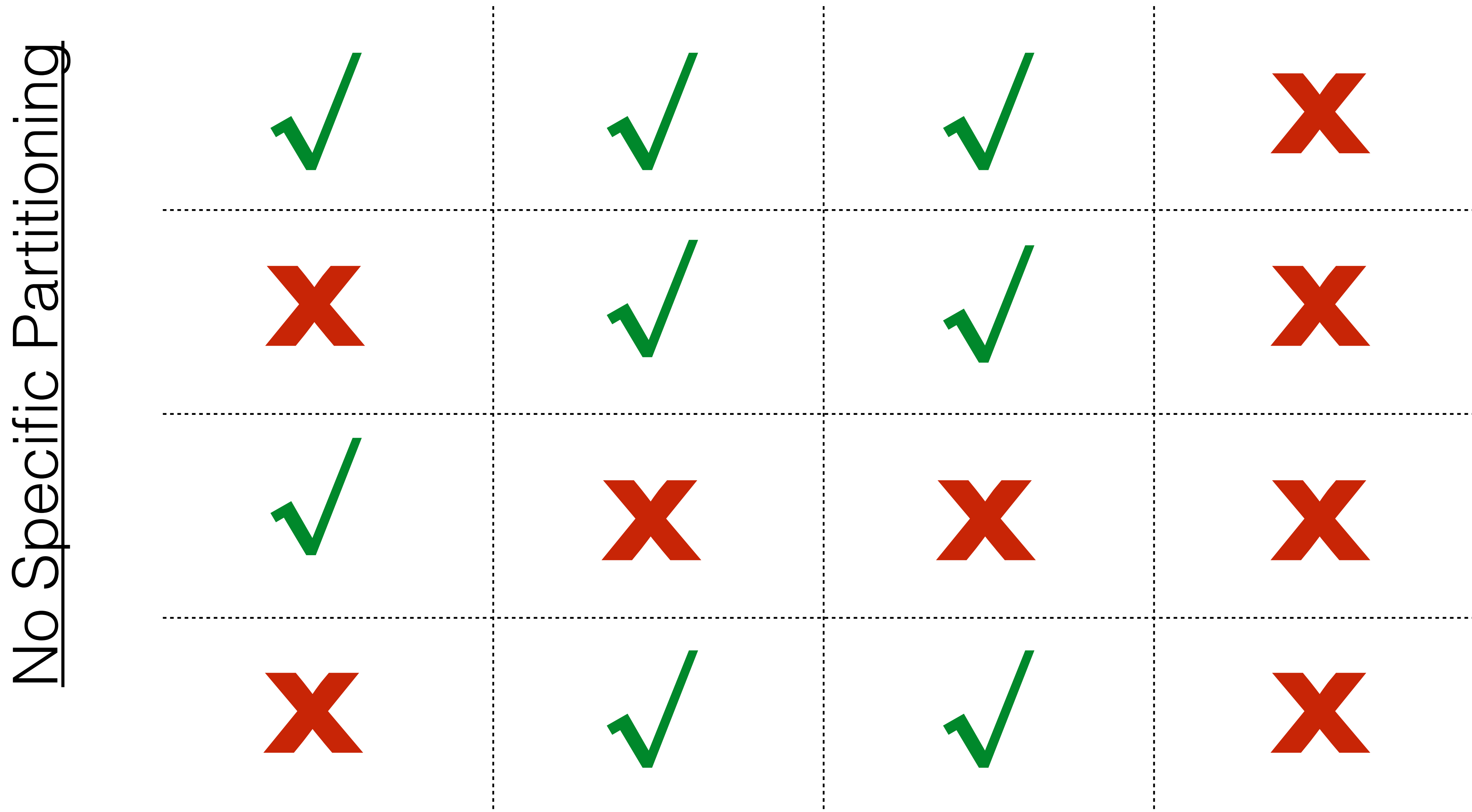
Parallel Joins

Use partitioning to eliminate
units of computation

Exactly the same idea as External Hash Join
(Called Theta Join for Inequalities)

Bloom Join

No Specific Partitioning



What if the join is highly selective… Can we detect which tuples are useful?

Bloom Join

Goal: Summarize which tuples are useful for the join?

False positives: OK
False negatives: NOT OK

Bloom Join

Strategy 1: Parity Bit

R $\bowtie$ S

R	parity	parity	S
<1,...>	1	0	<2,...>
<2,...>	0		
<3,...>	1		
<4,...>	0		

Node 1

Send me data w/ parity bit 0

Node 2

Bloom Join

Strategy 1: Parity Bit

R		⋈		S
<1,...>	1	<2,...>	0	<2,...>
<2,...>	0	<4,...>	1	<3,...>
<3,...>	1			
<4,...>	0			

Node 1

Send me data w/ parity bit 0 or 1

Node 2

Bloom Join

Strategy 2: Multiple Parity Bits

<u>R</u>	⋈	<u>S</u>
<1,...> 01		10 <2,...>
<2,...> 10		11 <3,...>
<3,...> 11		
<4,...> 00		

Node 1

Send me data w/ parity bits 10, 11

Node 2

What's the problem with this?

A Simplified Bloom Join

Key 1	00101010
Key 2	01010110
Key 3	10000110
Key 4	01001100

How do we summarize?

Bitwise OR

e.g. (Key 1 | Key 2)

= 01111110

How do we test for inclusion?

(Key & Summary) == Key?

(Key 1 & S) = 00101010 ✓

(Key 3 & S) = 00000110 ✗

(Key 4 & S) = 01001100 ✓

False Positive

Bloom Filters

Generating a bit vector for a key:

M - # of bits in the bit vector

K - # of hash functions

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

Each bit vector has ~K bits set

Bloom Filters

Probability that 1 bit is set by 1 hash fn

$1/m$

Bloom Filters

Probability that 1 bit is not set by 1 hash fn

$$1 - \ 1/m$$

Bloom Filters

Probability that 1 bit is not set by k hash fns

$$(1 - 1/m)^k$$

Bloom Filters

Probability that 1 bit is not set by k hash fns
for n records

$$(1 - 1/m)^{k\,n}$$

So for an arbitrary record, what is the probability
that all of its bits will be set?

Bloom Filters

Probability that 1 bit is set by k hash fns
for n records

$$1 - (1 - 1/m)^{k\,n}$$

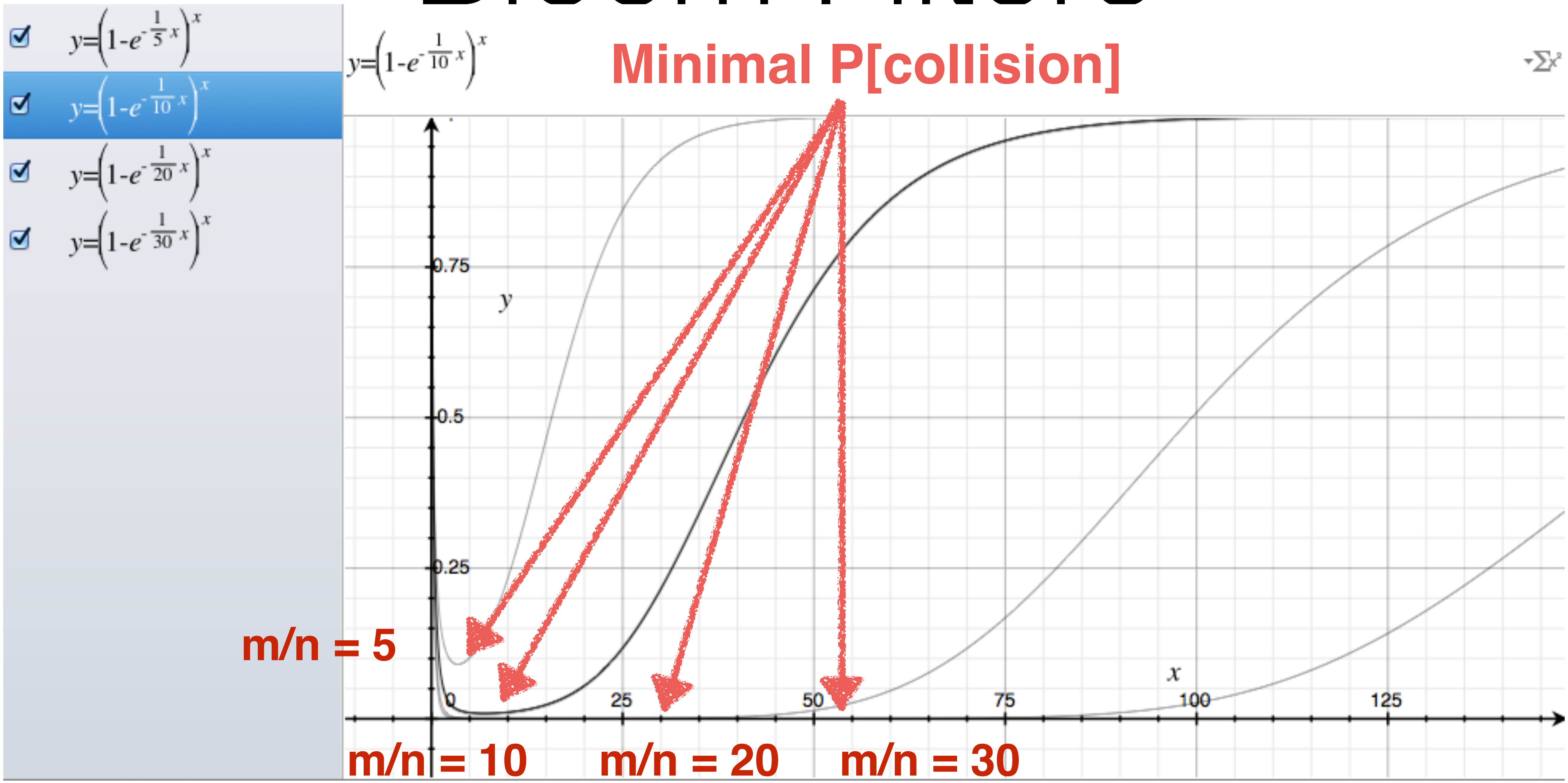
Bloom Filters

Probability that all k bits are set by k hash fns for n records

$$\approx \left(1 - \left(1 - 1/m \right)^{kn} \right)^{k}$$

$$\approx \left(1 - e^{-kn/m}\right)^{k}$$

Bloom Filters



Minimal P[collision] is at k ≈ c · m/n

Bloom Filters

$$k \approx c \cdot m/n$$

$$\frac{m}{k} \approx cn$$

m is linearly related to n (for a fixed k)

Bloom Join

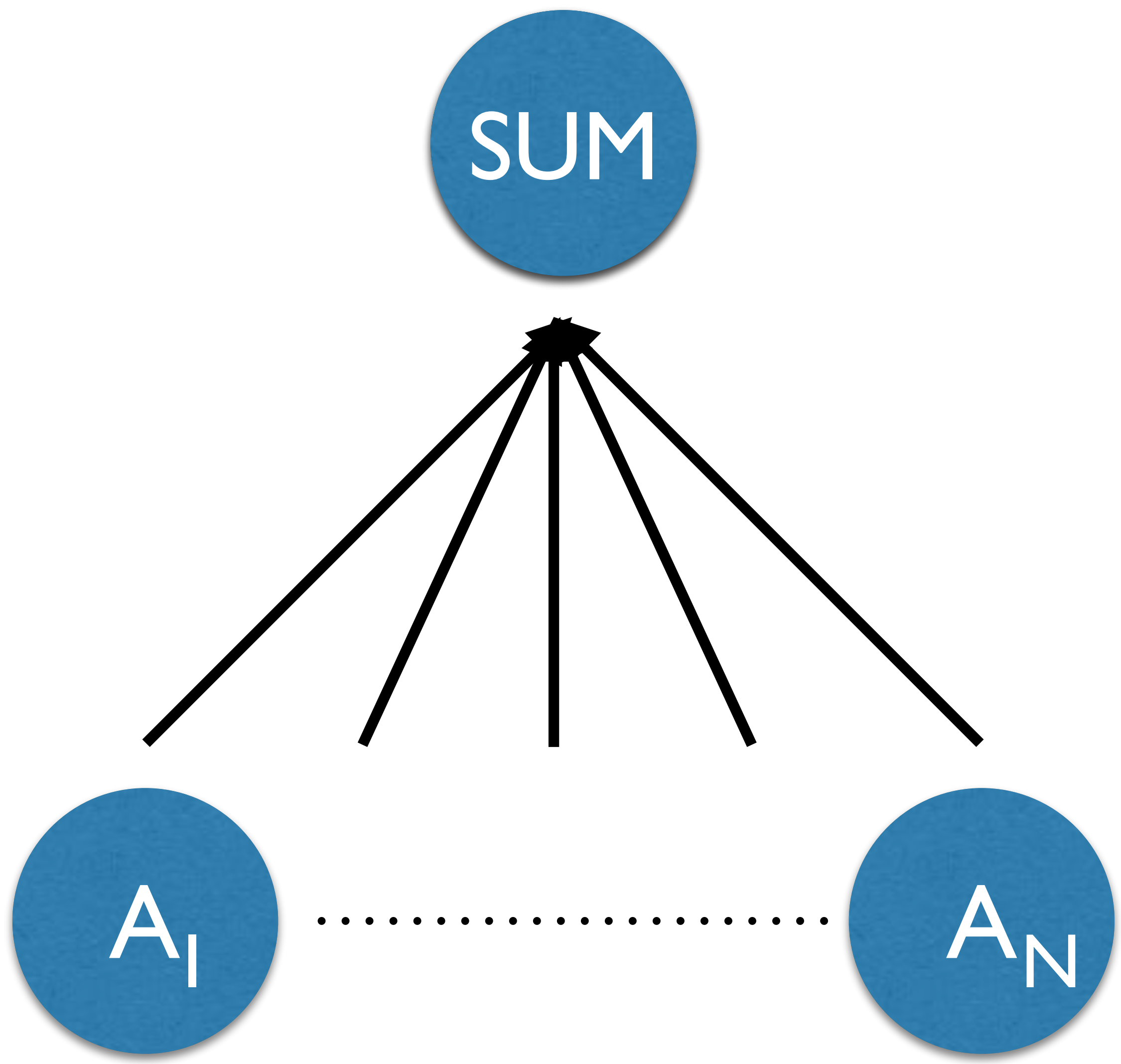
- Node 2 Computes Bloom Filter for Local Records
- Node 2 Sends Bloom Filter to Node 1
- Node 1 Matches Local Records Against Bloom Filter
- Node 1 Sends Matched Records to Node 2
    - Superset of “useful” records
- Node 2 Performs Join Locally

Parallel Aggregates

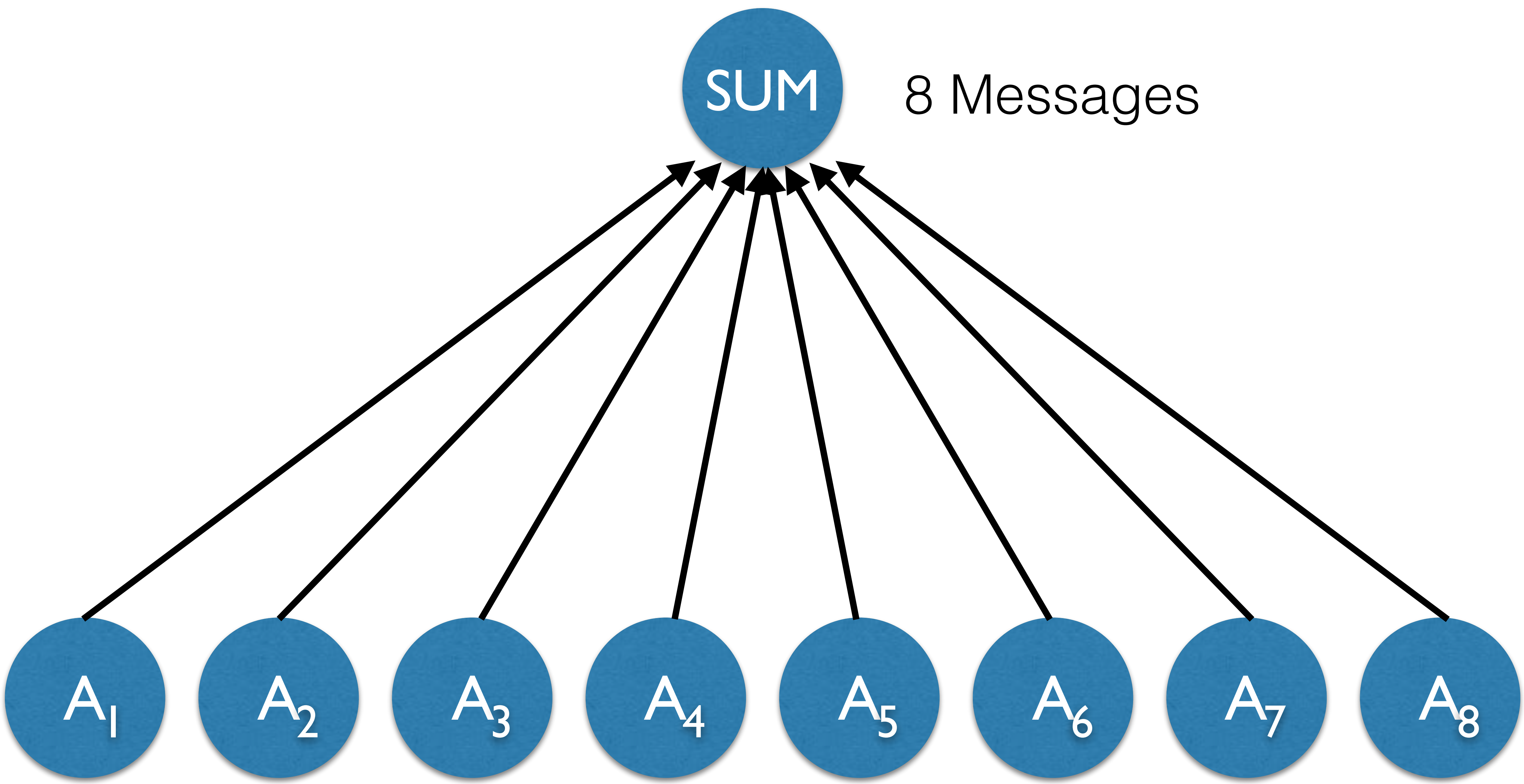
Algebraic: Bounded-size intermediate state
(Sum, Count, Avg, Min, Max)

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

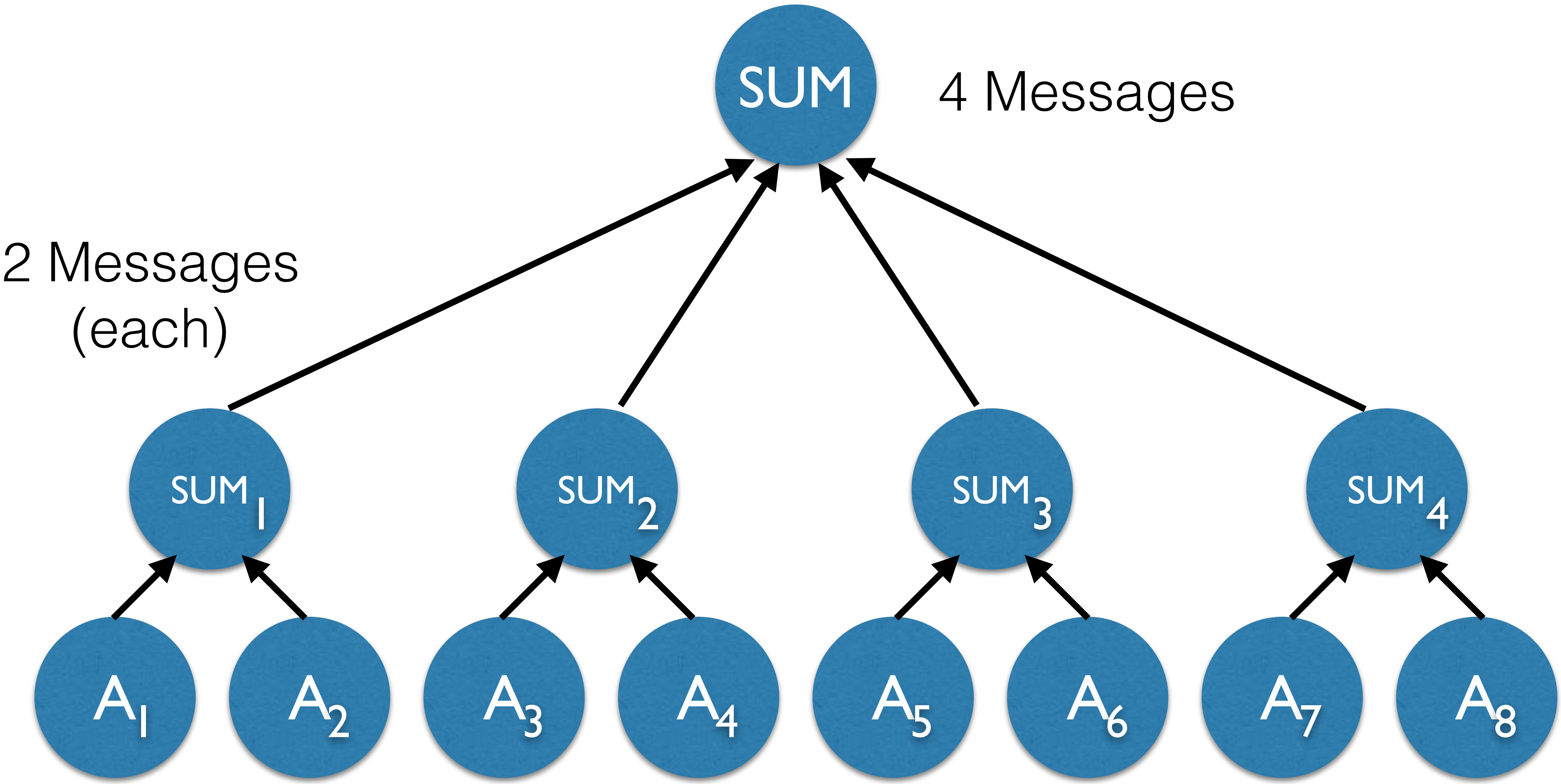
Fan-In Aggregation



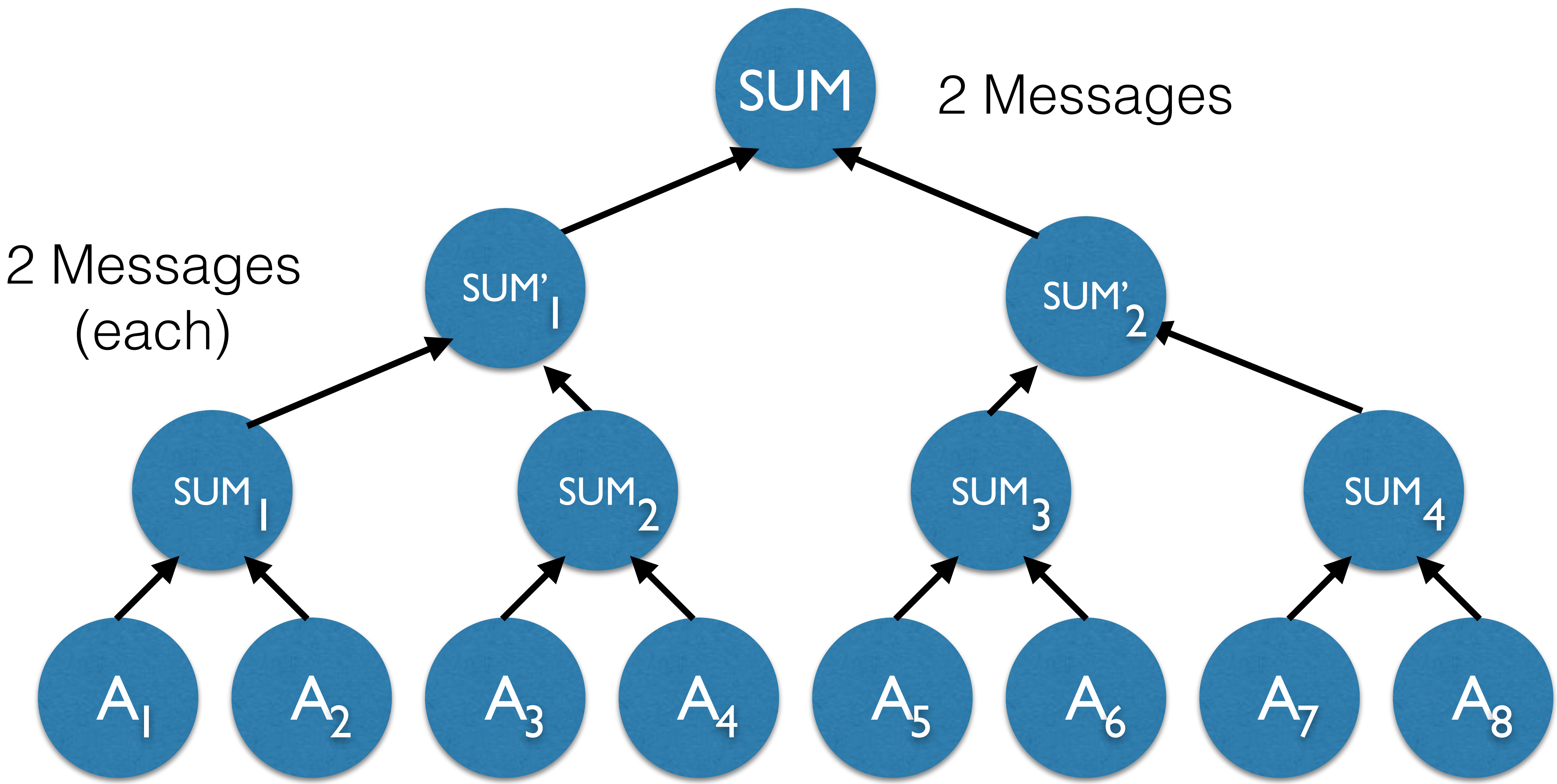
Fan-In Aggregation



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

$$\langle \mathtt{SUM}_1, \mathtt{COUNT}_1 \rangle \otimes \ldots \otimes \langle \mathtt{SUM}_N, \mathtt{COUNT}_N \rangle$$
$$= \langle \mathtt{SUM}_1 + \ldots + \mathtt{SUM}_N, \mathtt{COUNT}_1 + \ldots + \mathtt{COUNT}_N \rangle$$

Compute(Intermediate) = Aggregate

$$\mathtt{Compute}(\langle \mathtt{SUM}, \mathtt{COUNT} \rangle) = \mathtt{SUM} \;/\; \mathtt{COUNT}$$