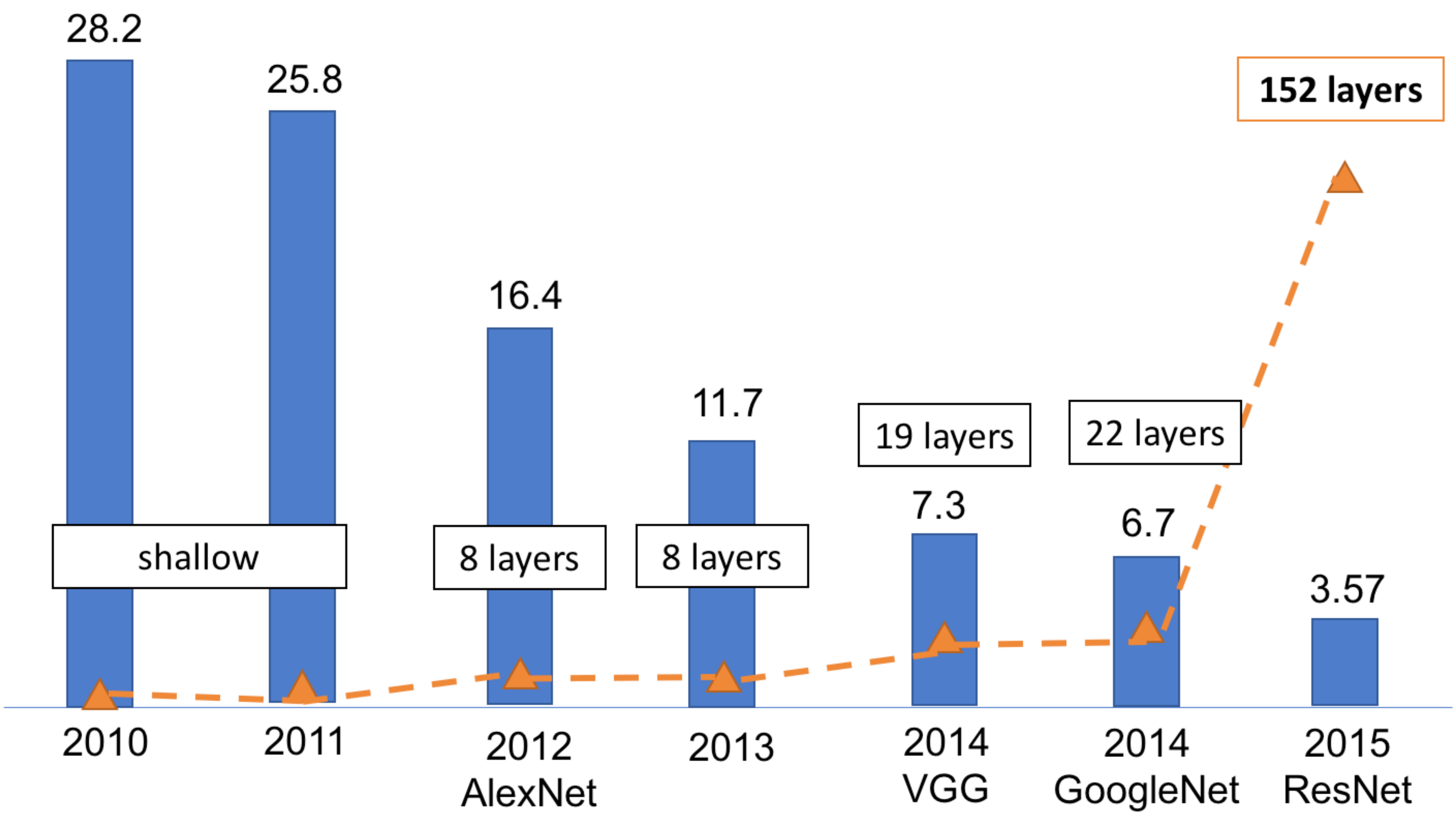
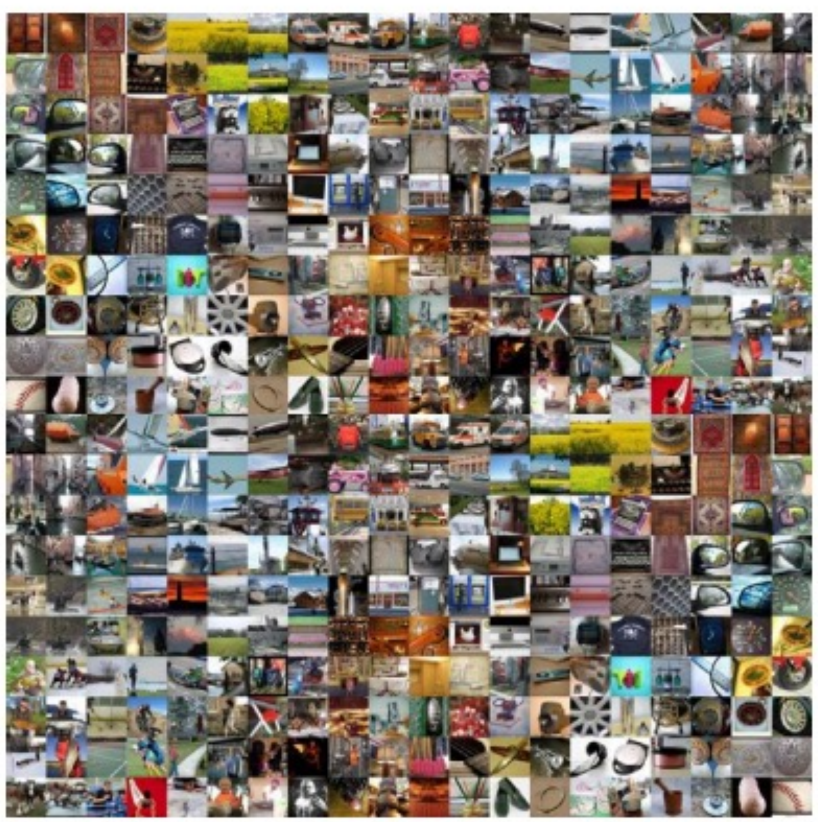
Software for (Deep) Machine Learning

Performance on ImageNet

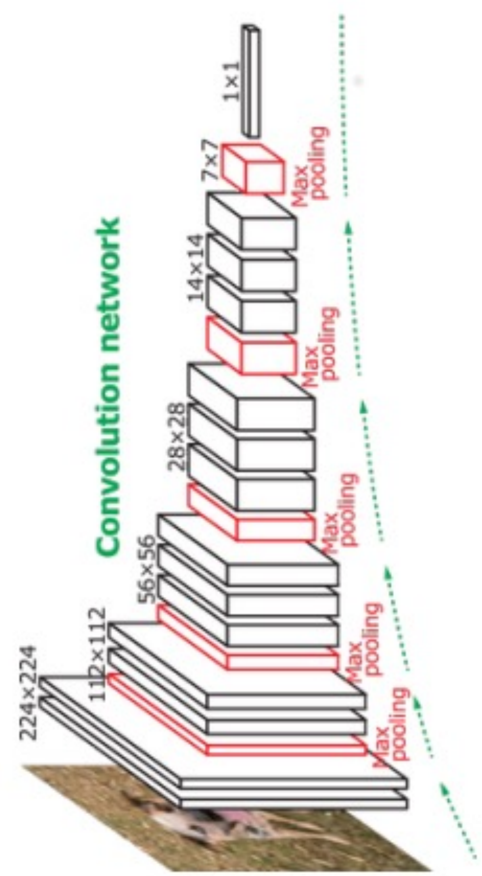


Scalability

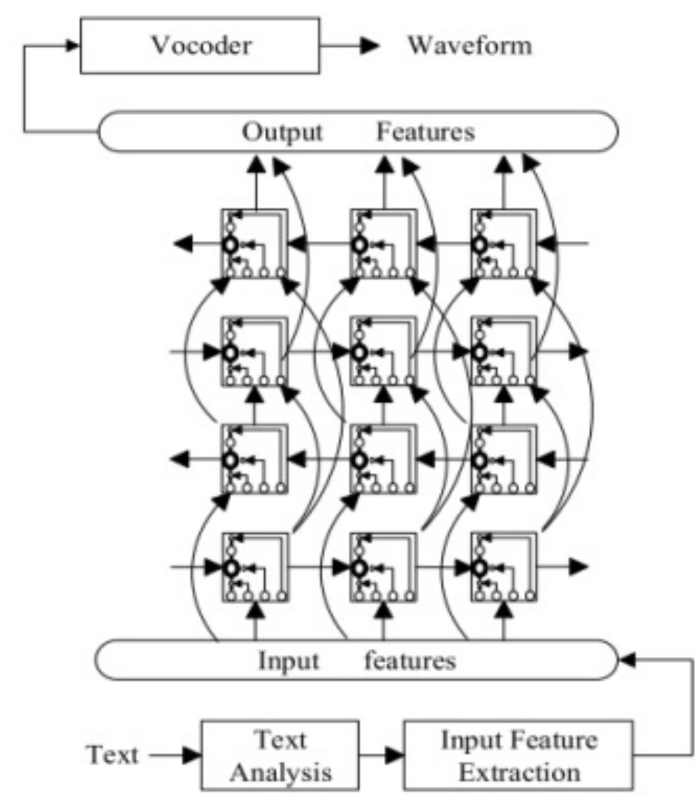
Big Data



Big Models

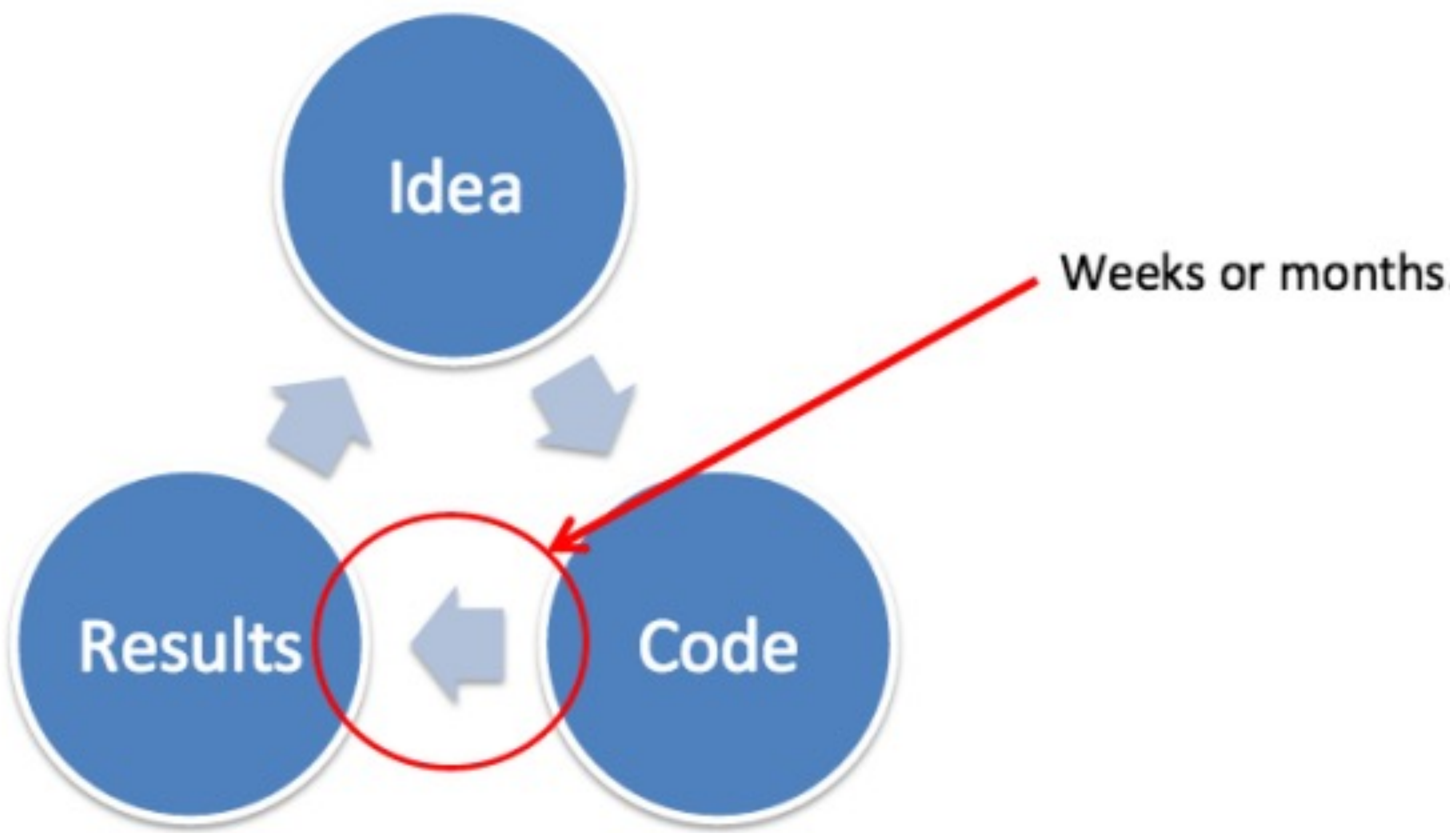


Complex Models



Motivation

- Training models is slow.



Why?

- Too many computations

```
while not end():
    data = get_data_sample()
    model.update(data)
```

$$\min_x \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x^{(k-1)})$$

Why?

- Too many computations

usually iterate over all the available training data

```
while not end():
    data = get_data_sample()
    model.update(data)
```

$$\min_x \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x^{(k-1)})$$

Why?

- Data is getting bigger

MNIST
60K 20x20

IMAGENET
~14M multiple sizes

Why?

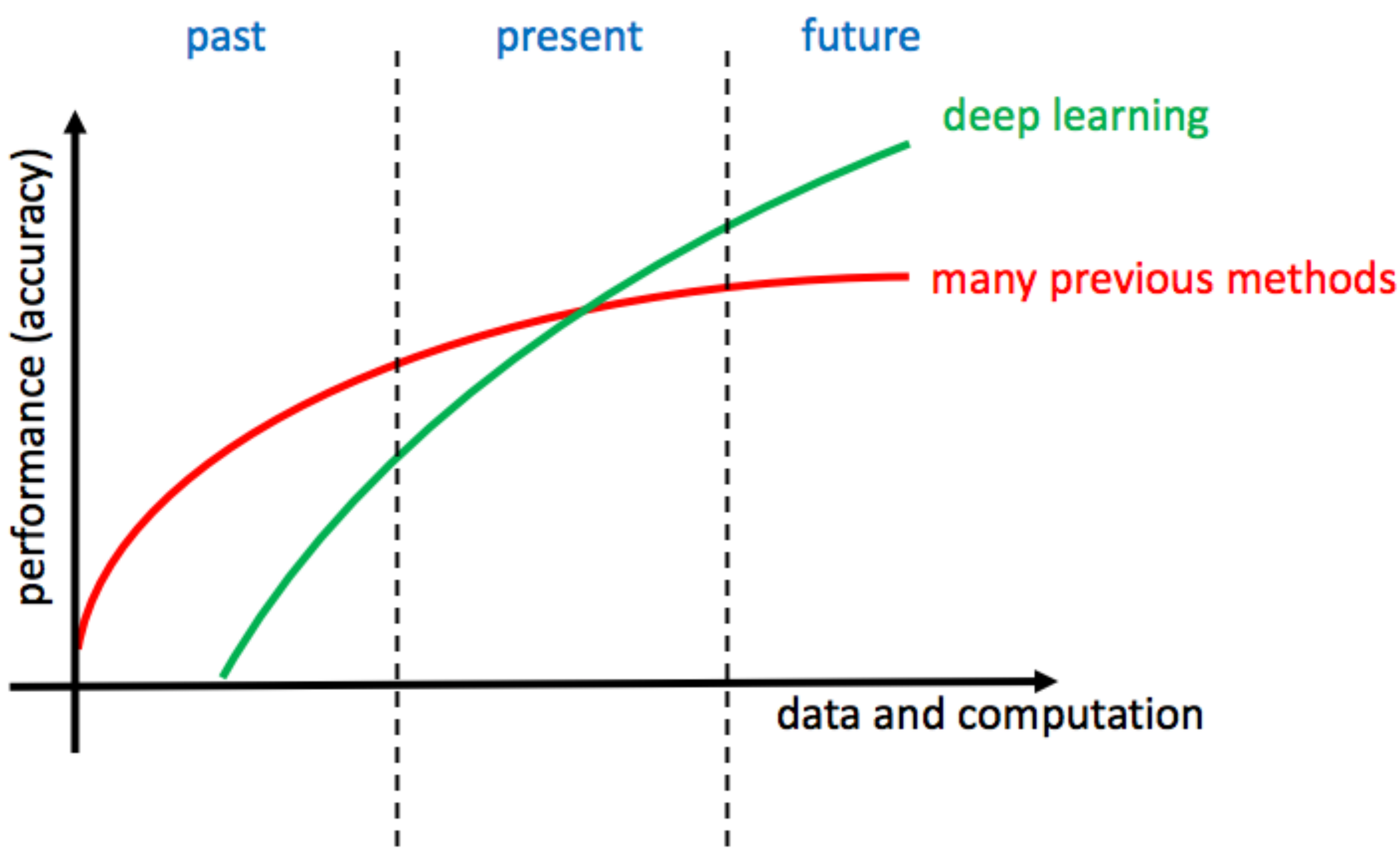
- Too many computations

```
while not end():
    data = get_data_sample()
    model.update(data)
```

might be slow

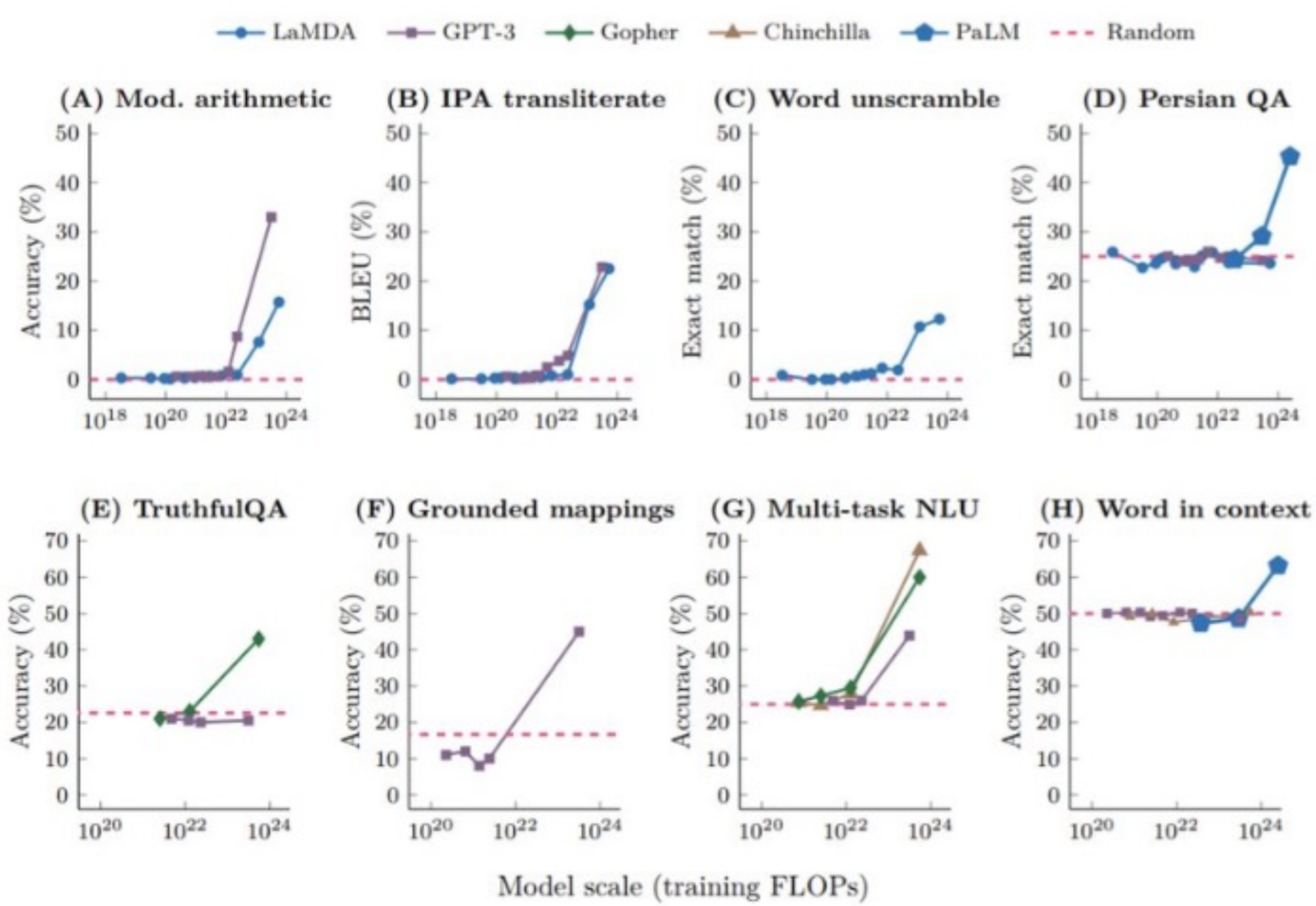
Why

- Larger models statistically work better



Why

- Larger models statistically work better



Solution 1: Data Parallelism

Data Parallelism

Parallel updates

parallelize/distribute data

```
while not end():
    data = get_data_sample()
    model.update(data)
```

duplicated

distributed

Data Parallelism

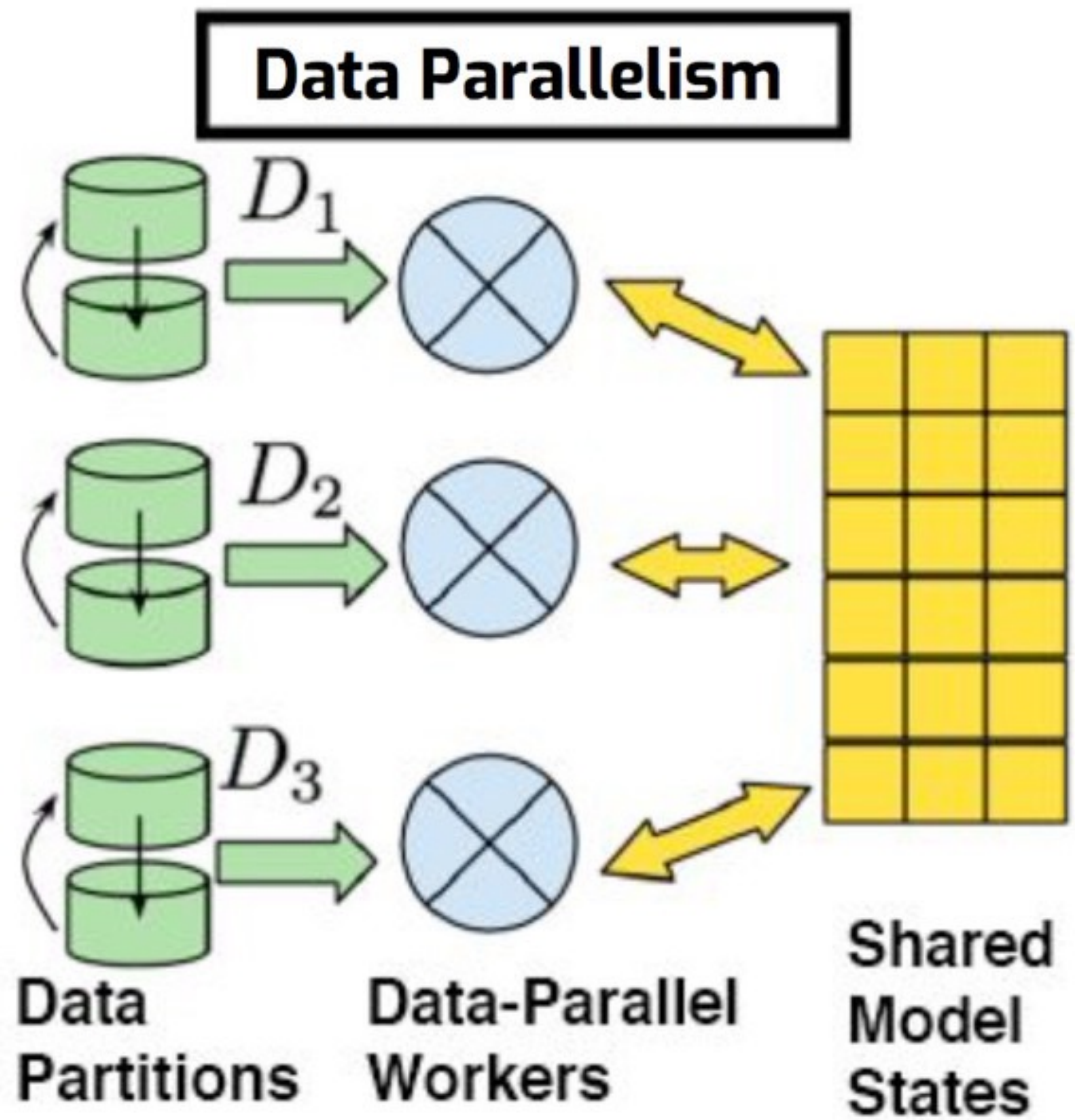
parallelize/distribute data

```
while not end():
    data = get_data_sample()
    model.update(data)
```

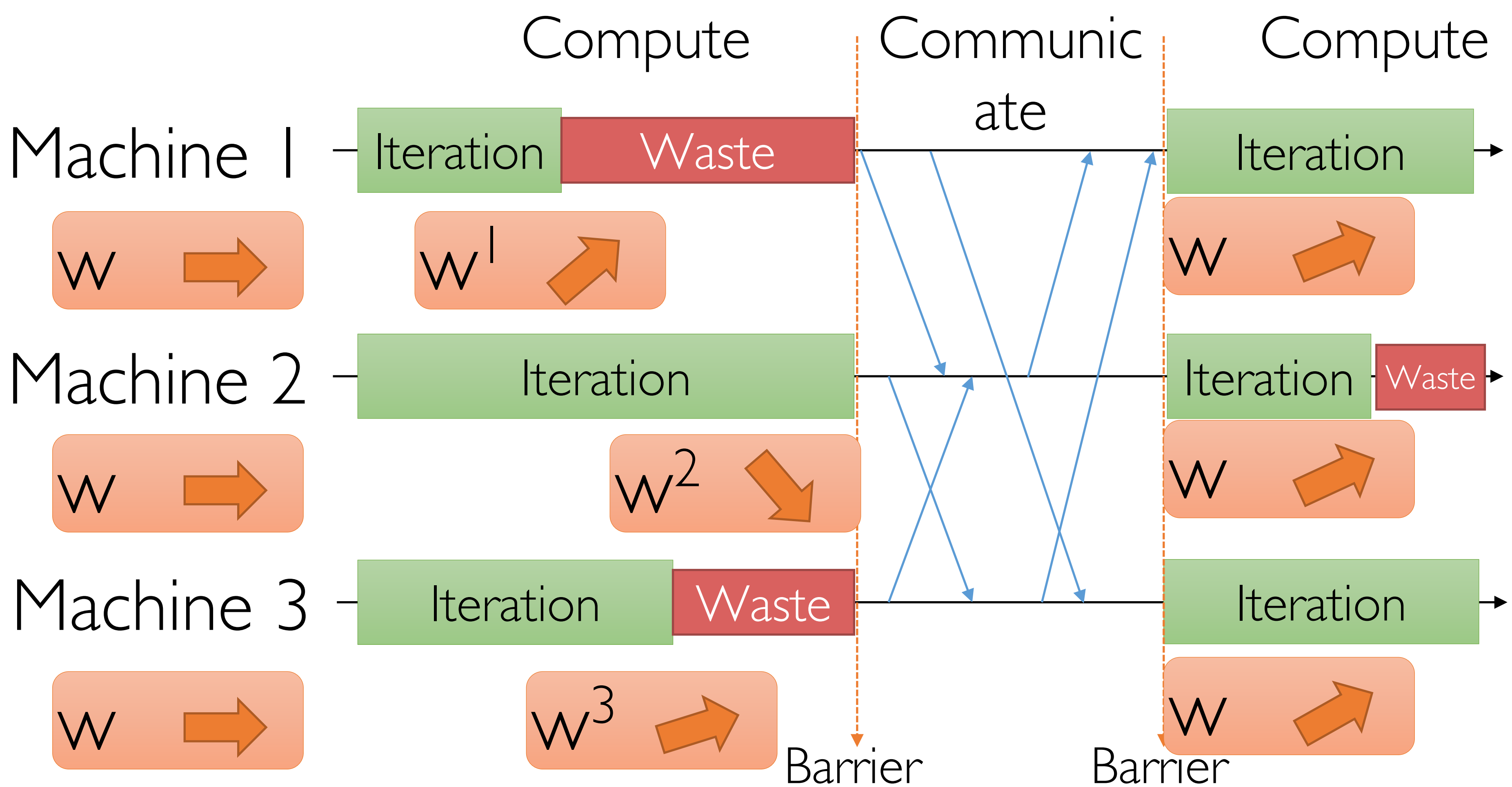
$$\min_x \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x^{(k-1)})$$

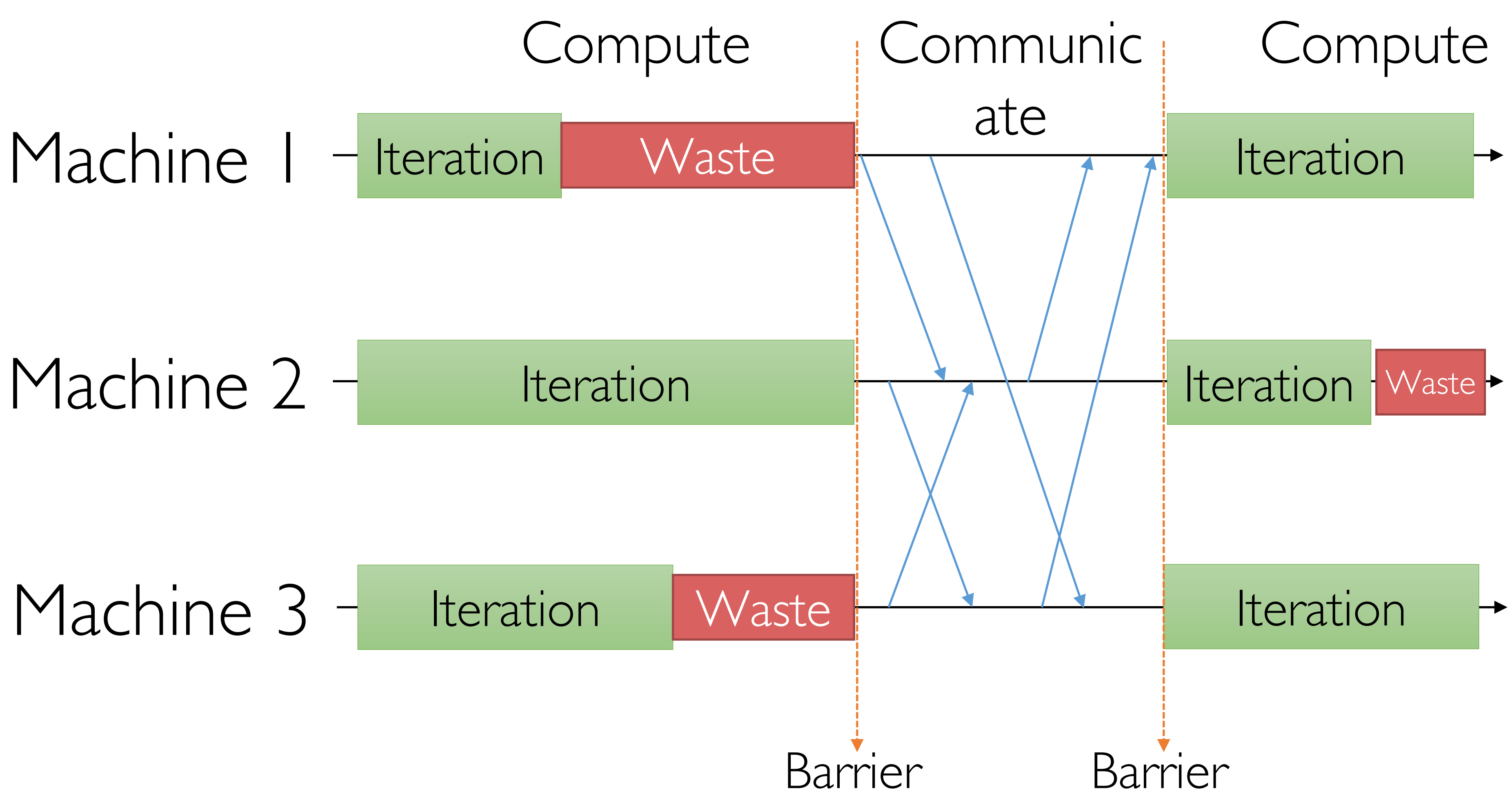
Data Parallelism



Bulk Synchronous Parallel (BSP) Execution

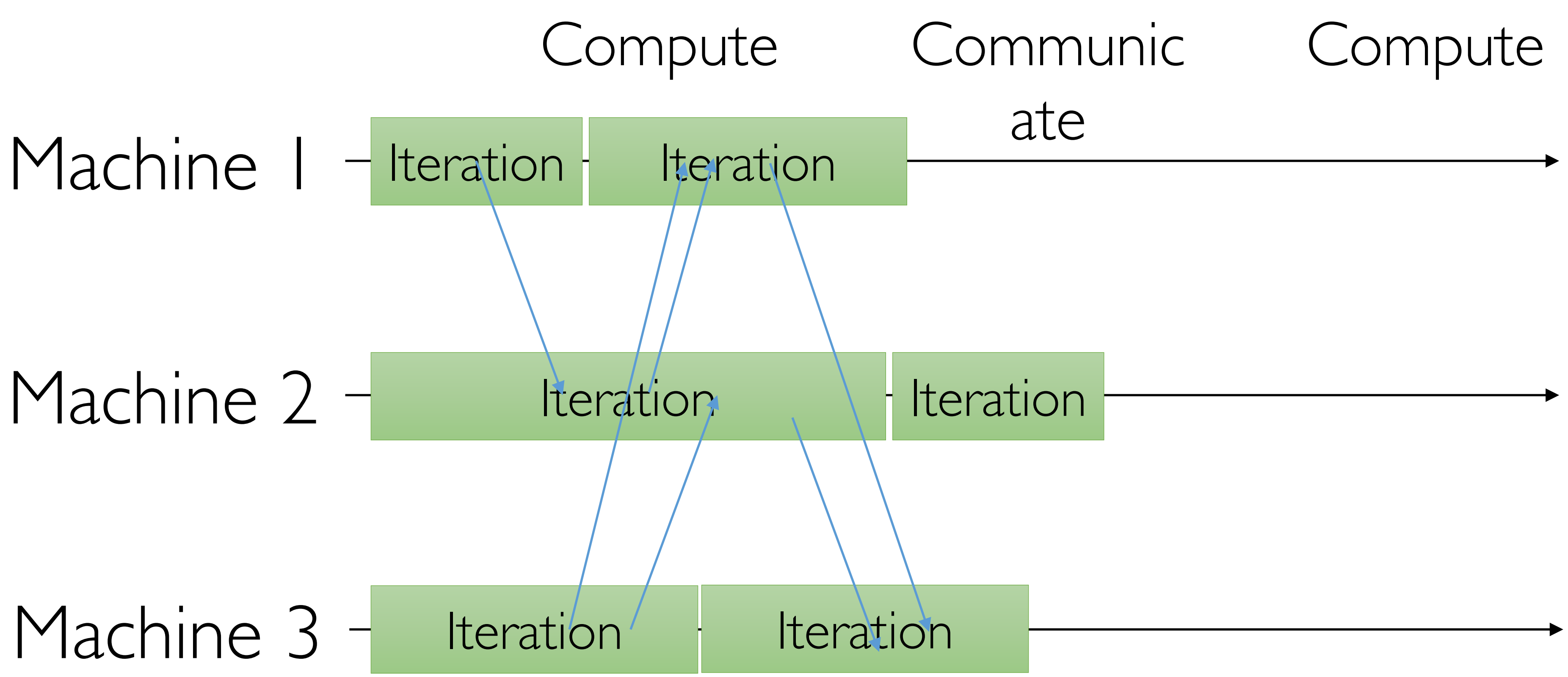


Asynchronous Execution



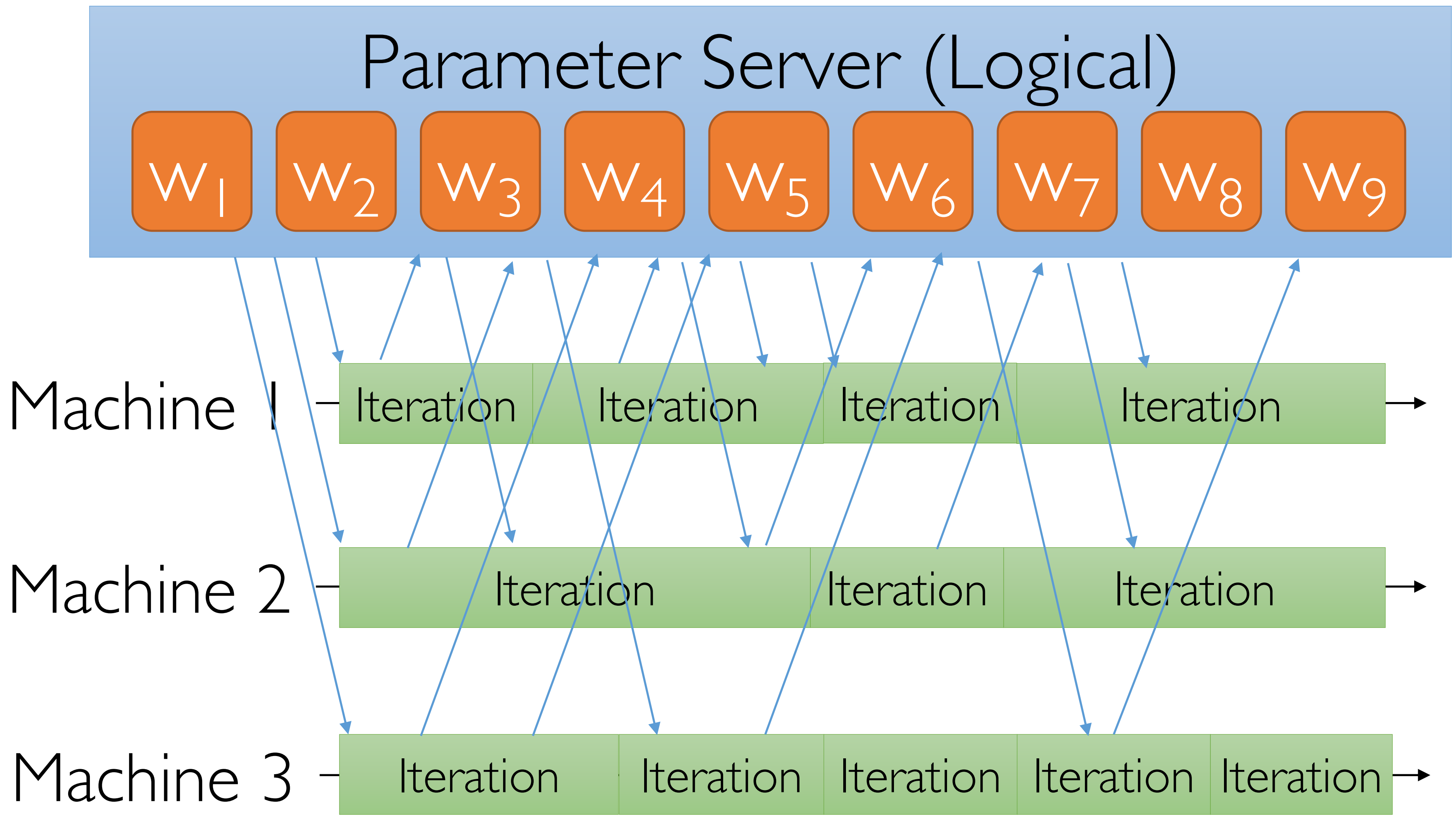
Enable more frequent coordination on parameter values

Asynchronous Execution

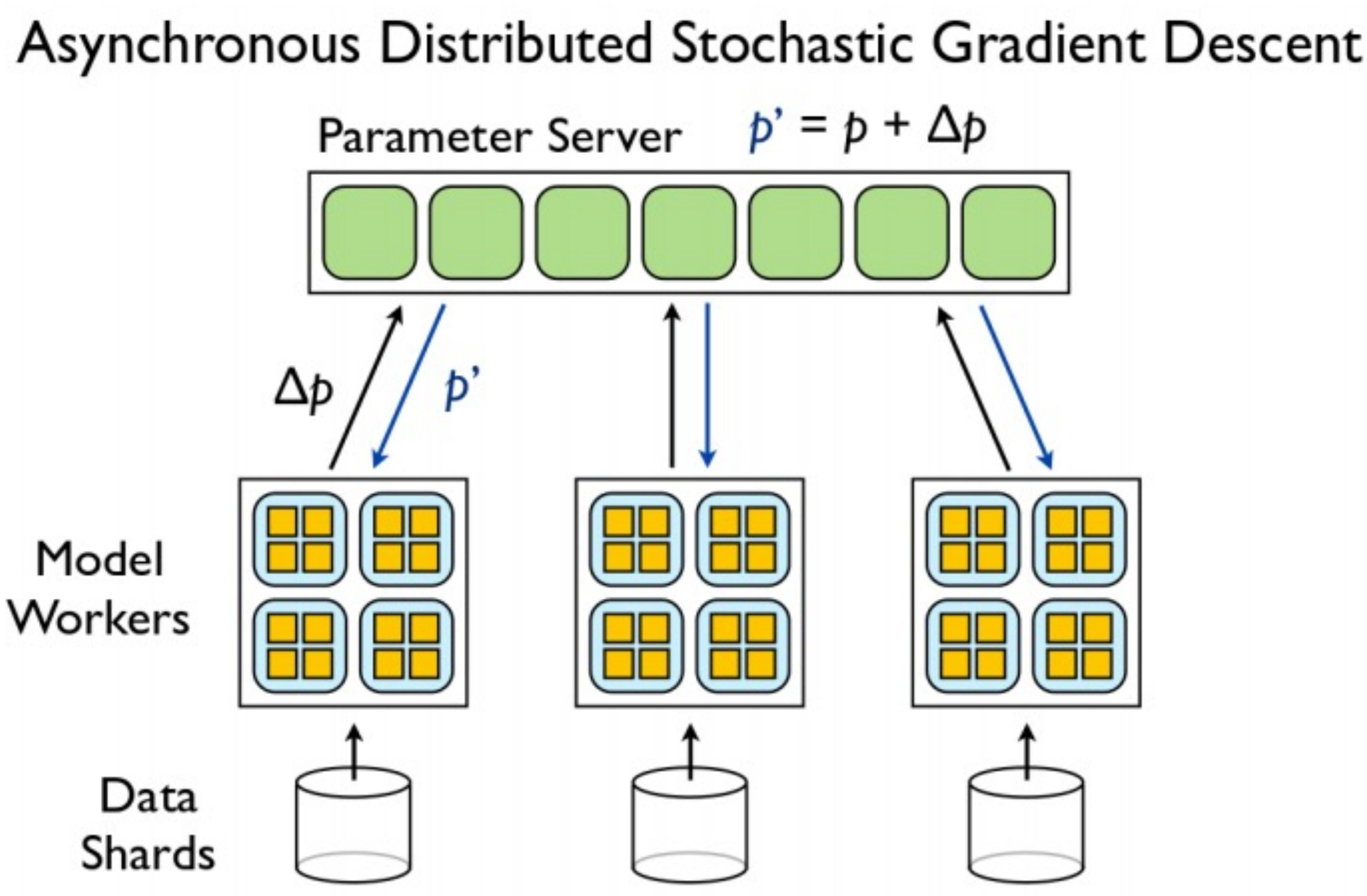


Enable more frequent coordination on parameter values

Asynchronous Execution

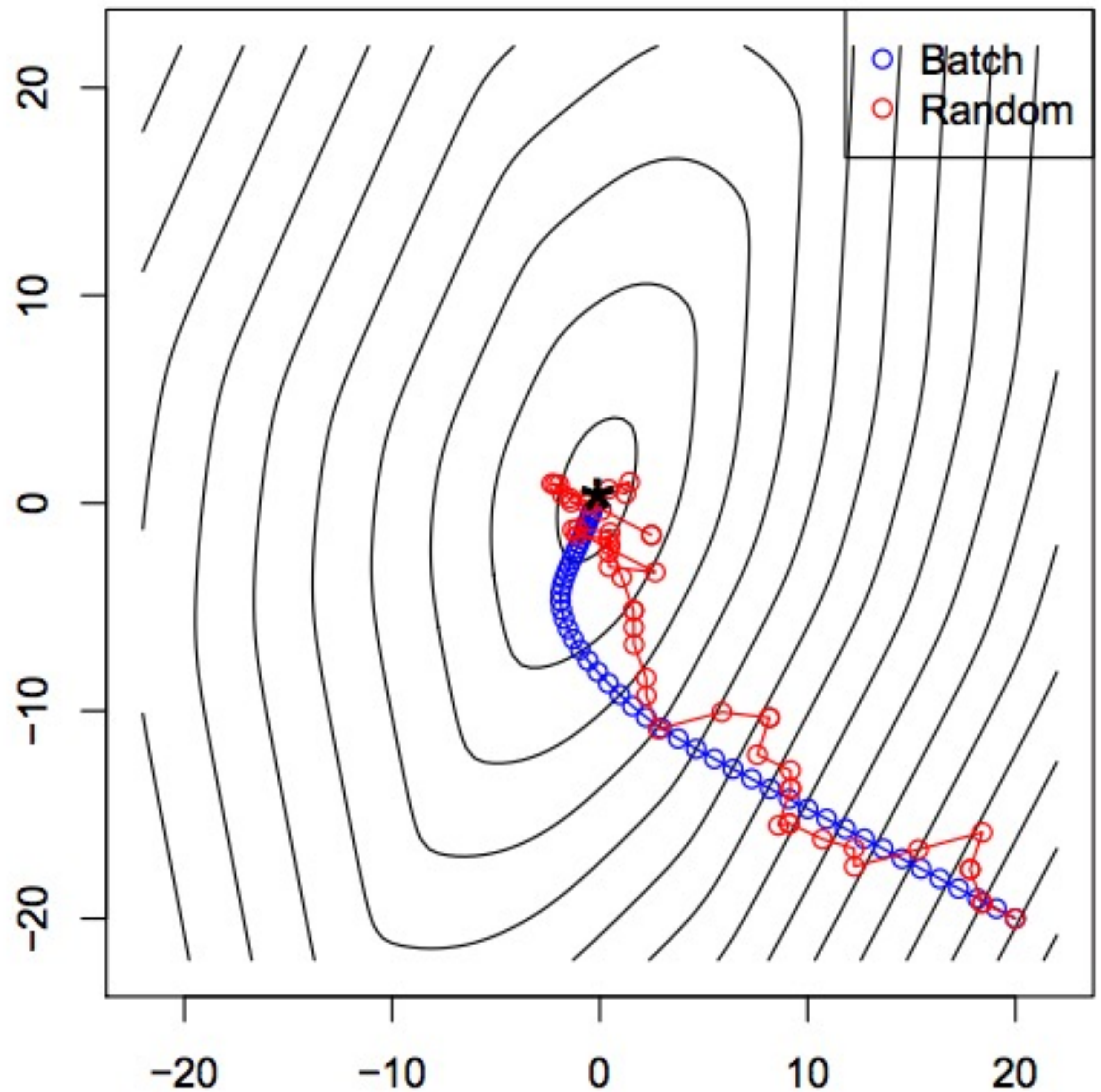


Data Parallelism



Pros and Cons

- Pros
  - Model agnostic
  - Horizontal scaling
- Cons
  - Very communication heavy
  - Noisy (async version)



Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

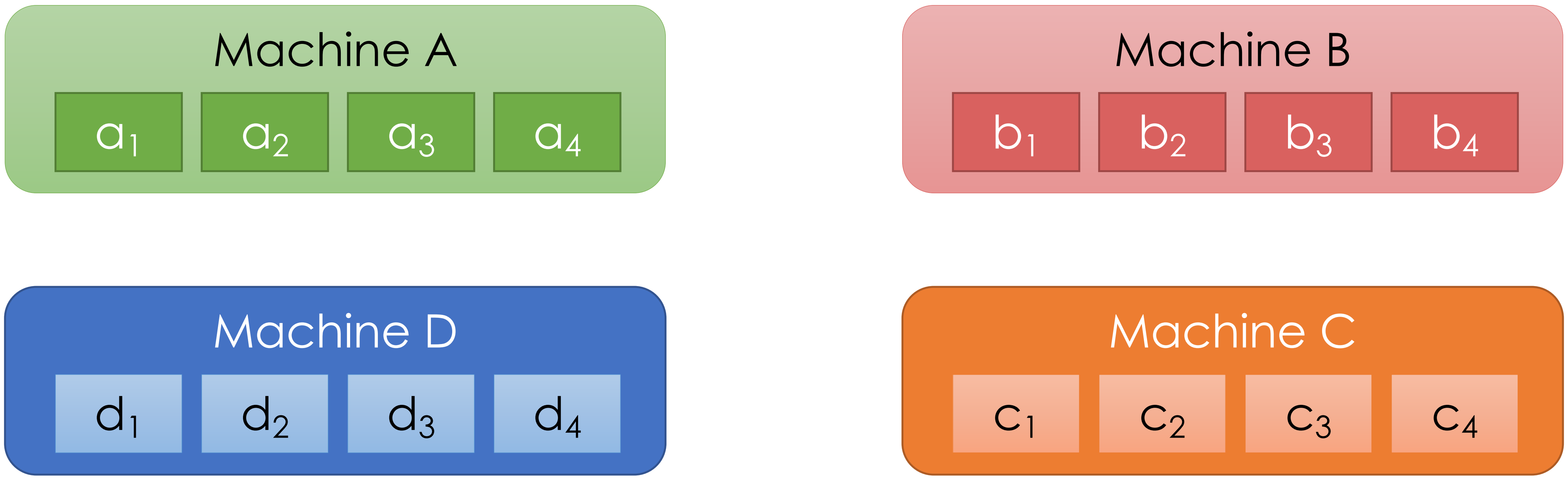
Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

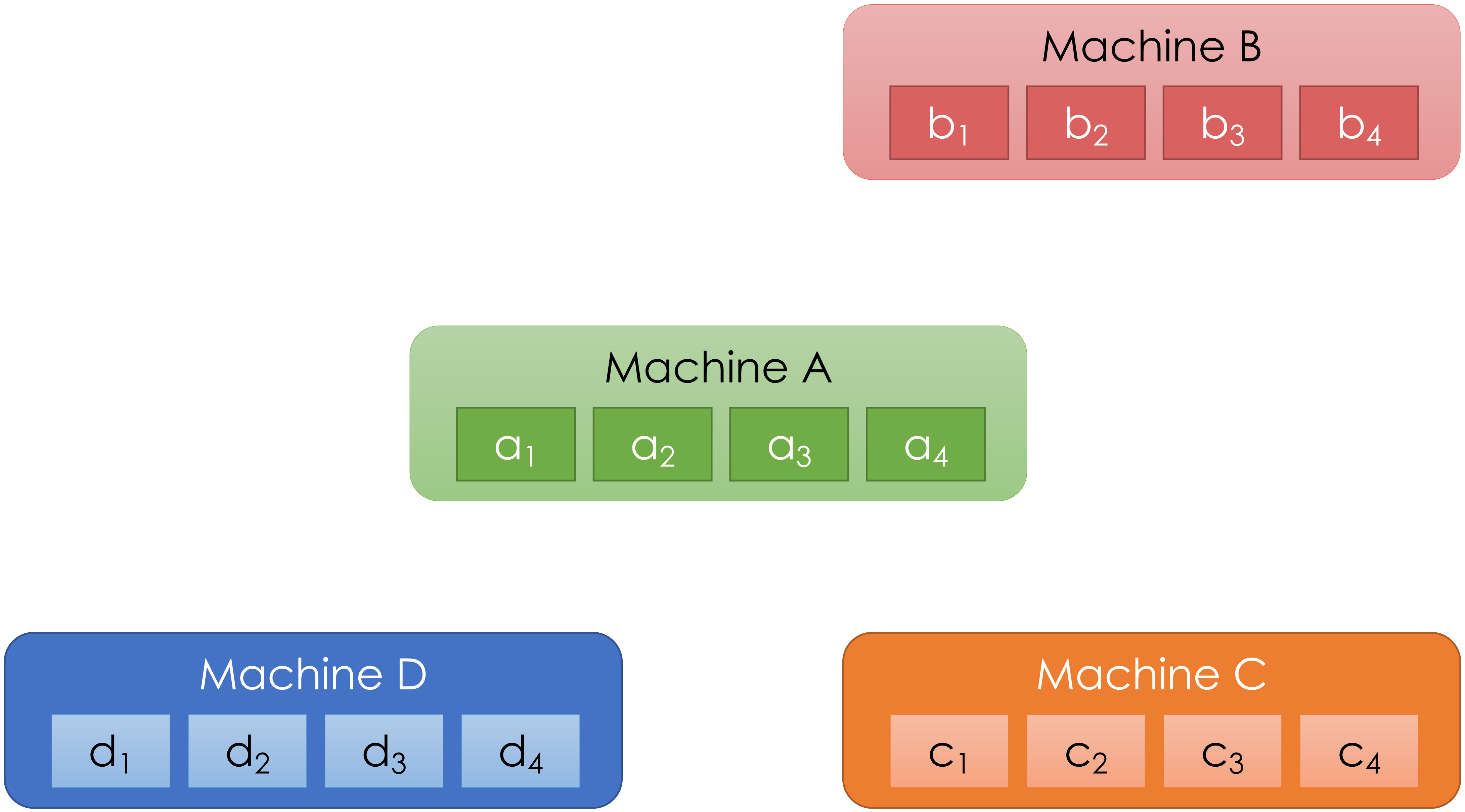
All Reduce

Mechanism to sum and distribute data across machines.

- Used to sum and distribute the gradient



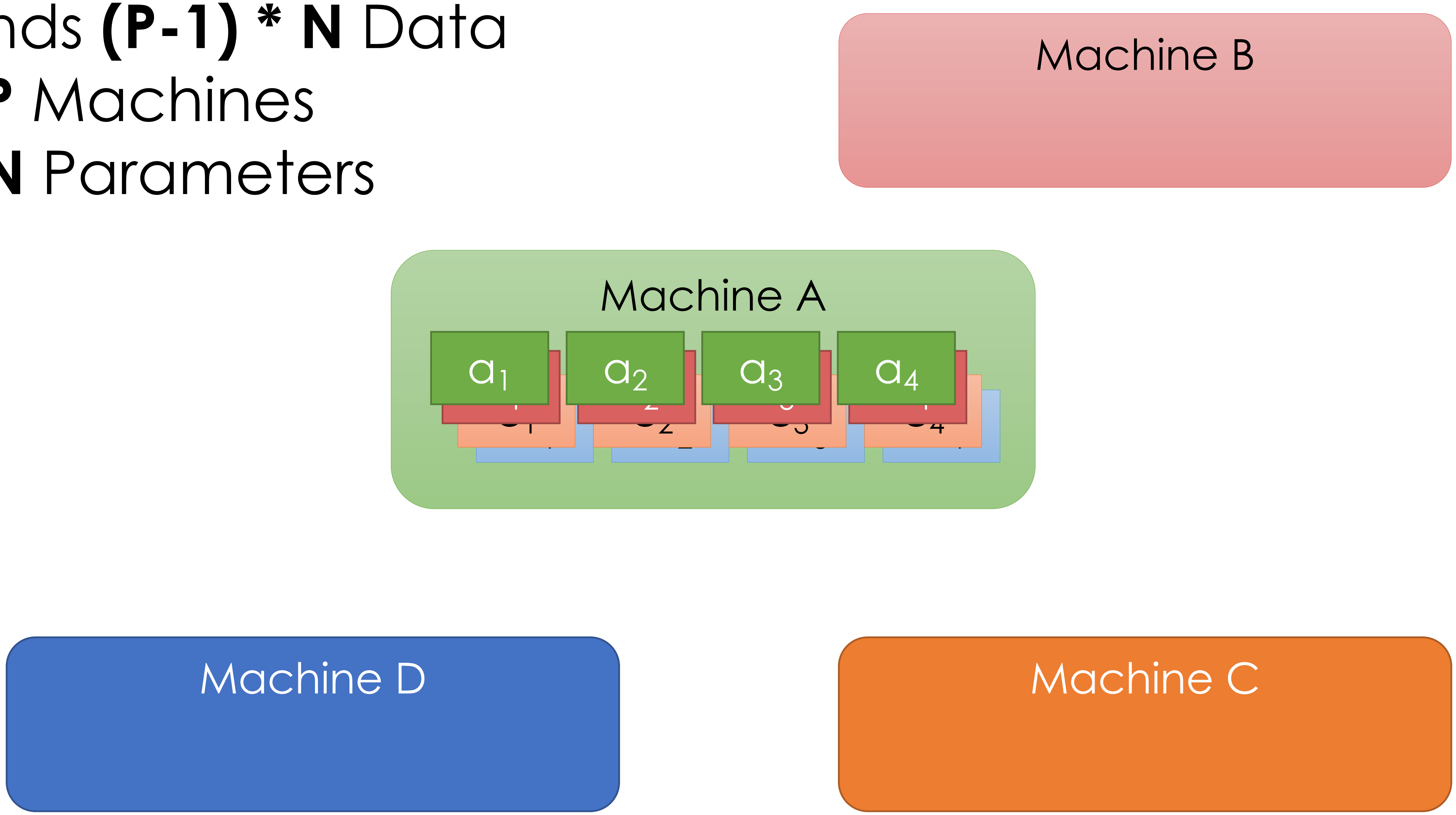
Single Master All-Reduce



Single Master All-Reduce

Sends **(P-1) * N** Data

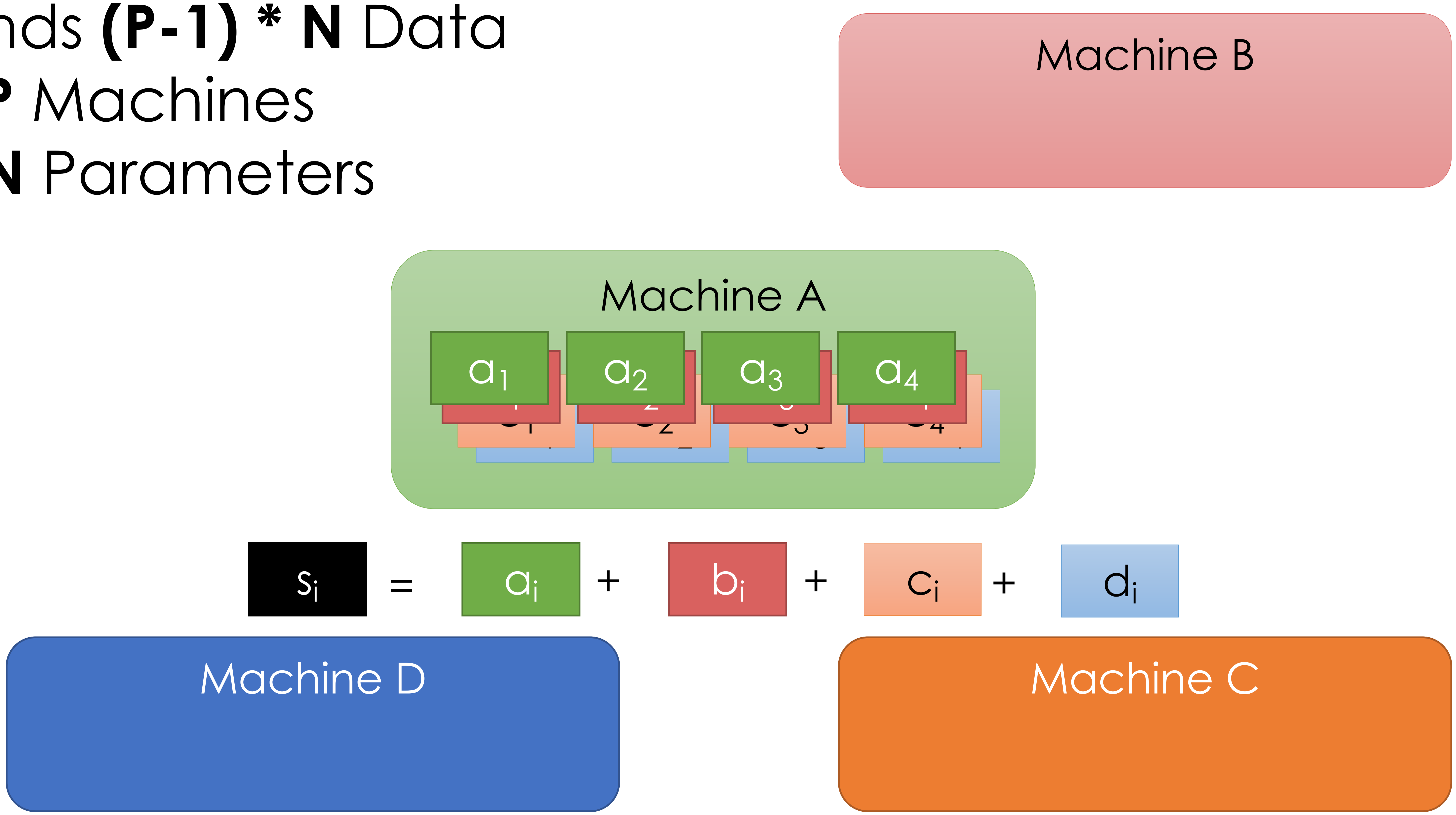
- ➢ **P** Machines
- ➢ **N** Parameters



Single Master All-Reduce

Sends **(P-1) * N** Data

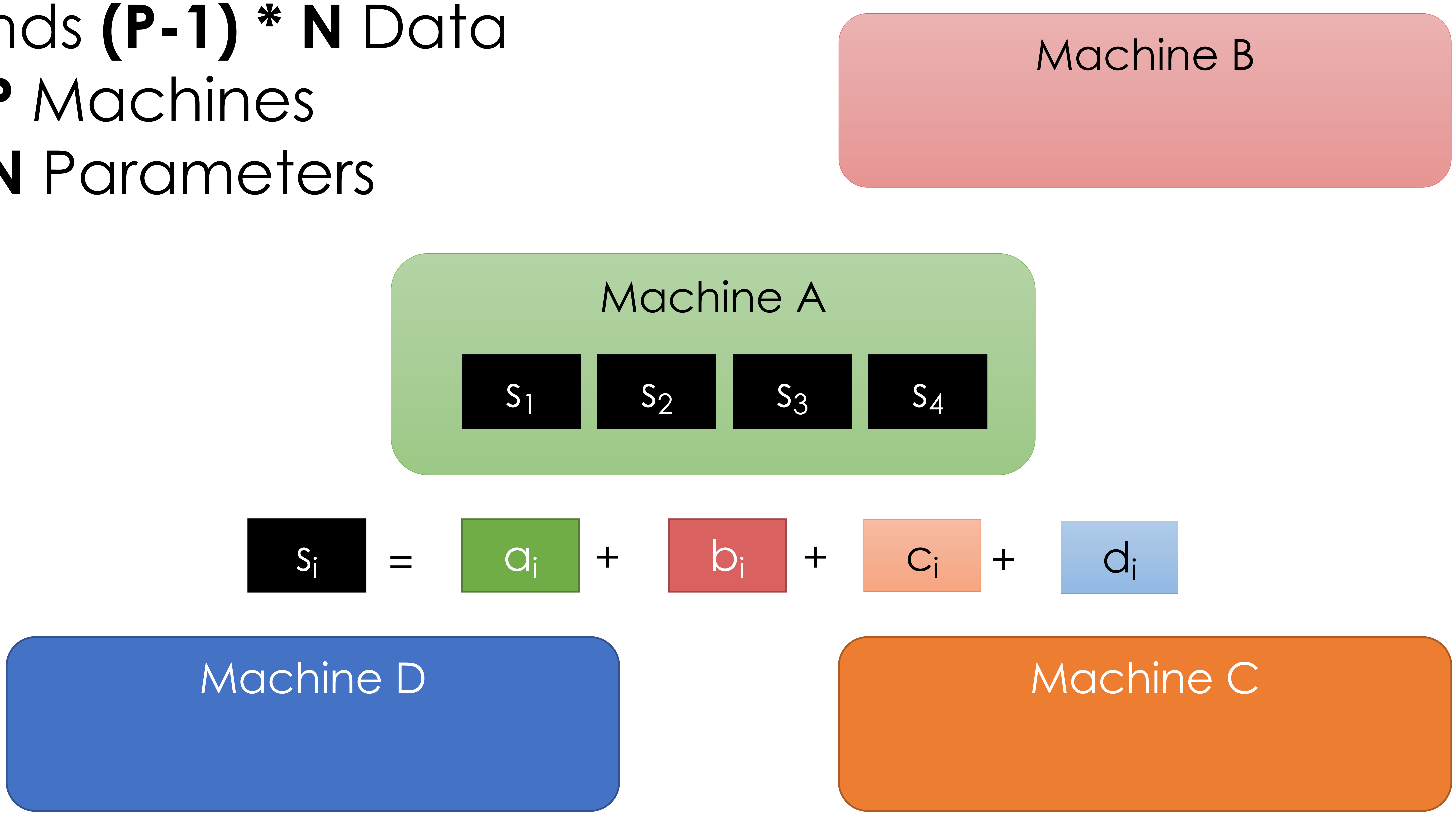
- **P** Machines
- **N** Parameters



Single Master All-Reduce

Sends **(P-1) * N** Data

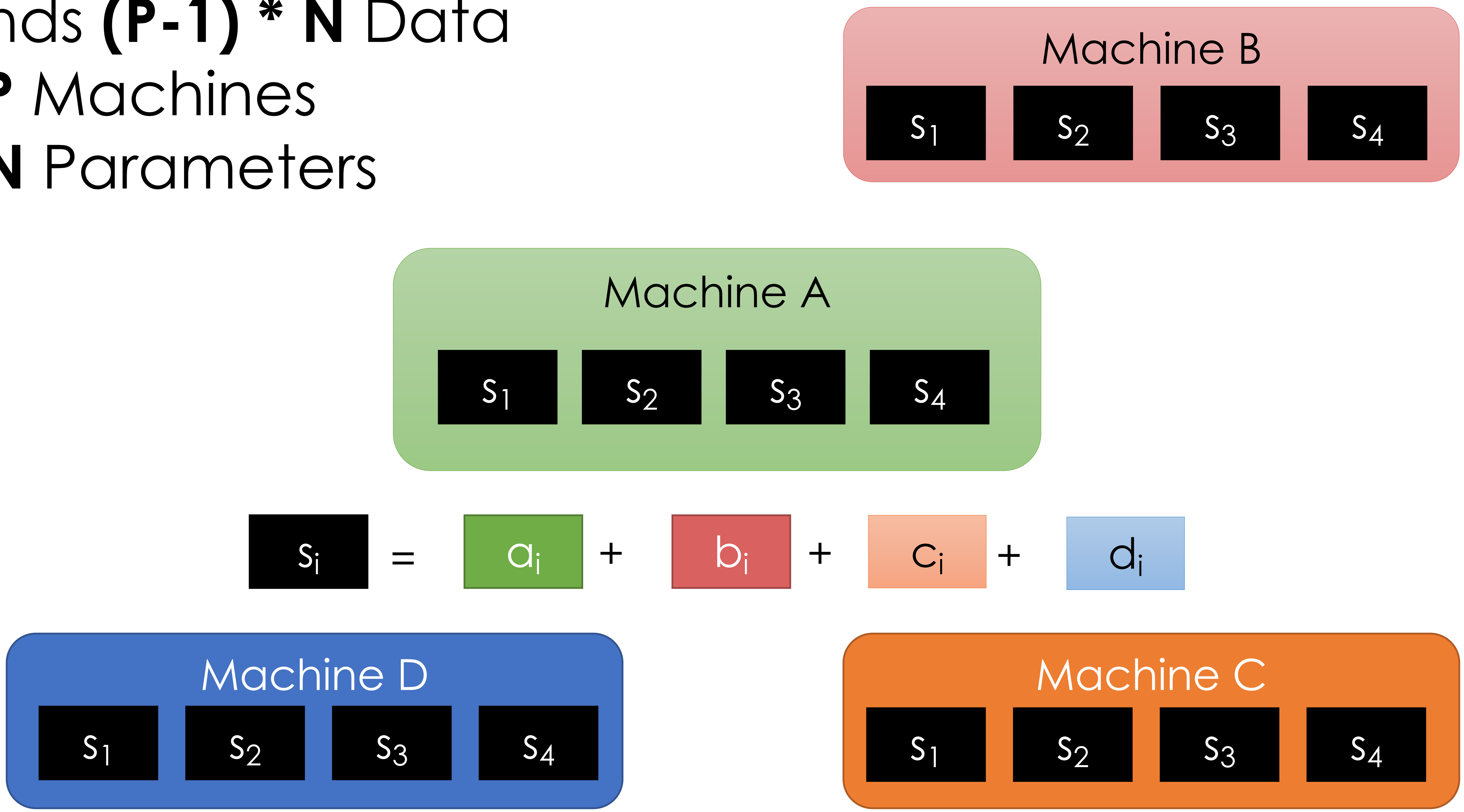
- **P** Machines
- **N** Parameters



Single Master All-Reduce

Sends **(P-1)** *2 *** N** Data

- ➢ **P** Machines
- ➢ **N** Parameters



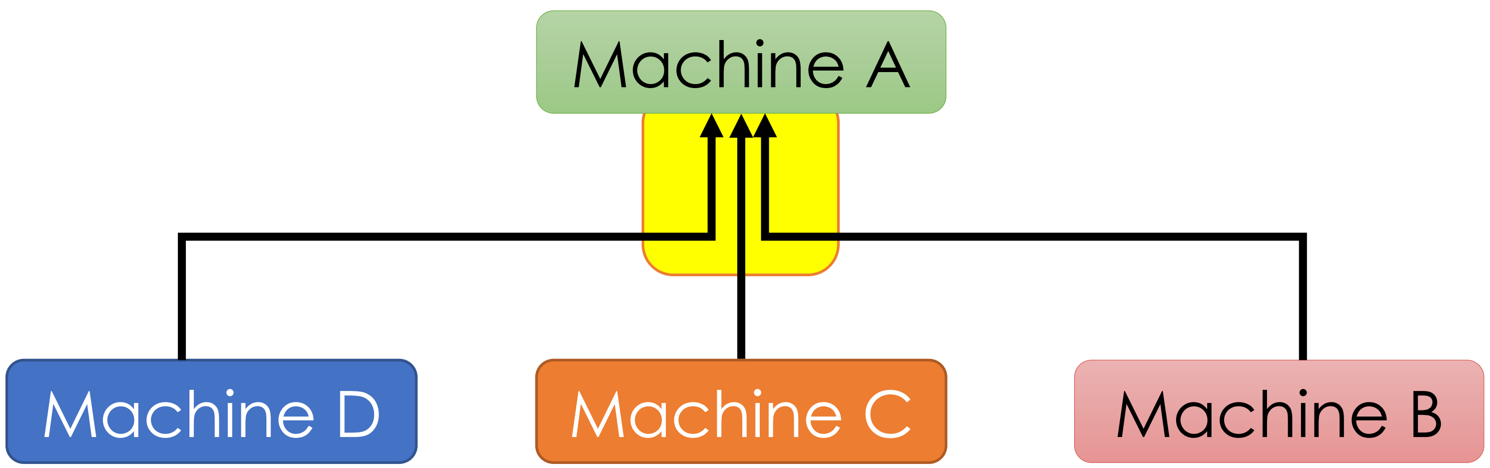
Single Master All-Reduce

*2

Sends **(P-1)** ***** **N** Data

➢ **P** Machines

➢ **N** Parameters

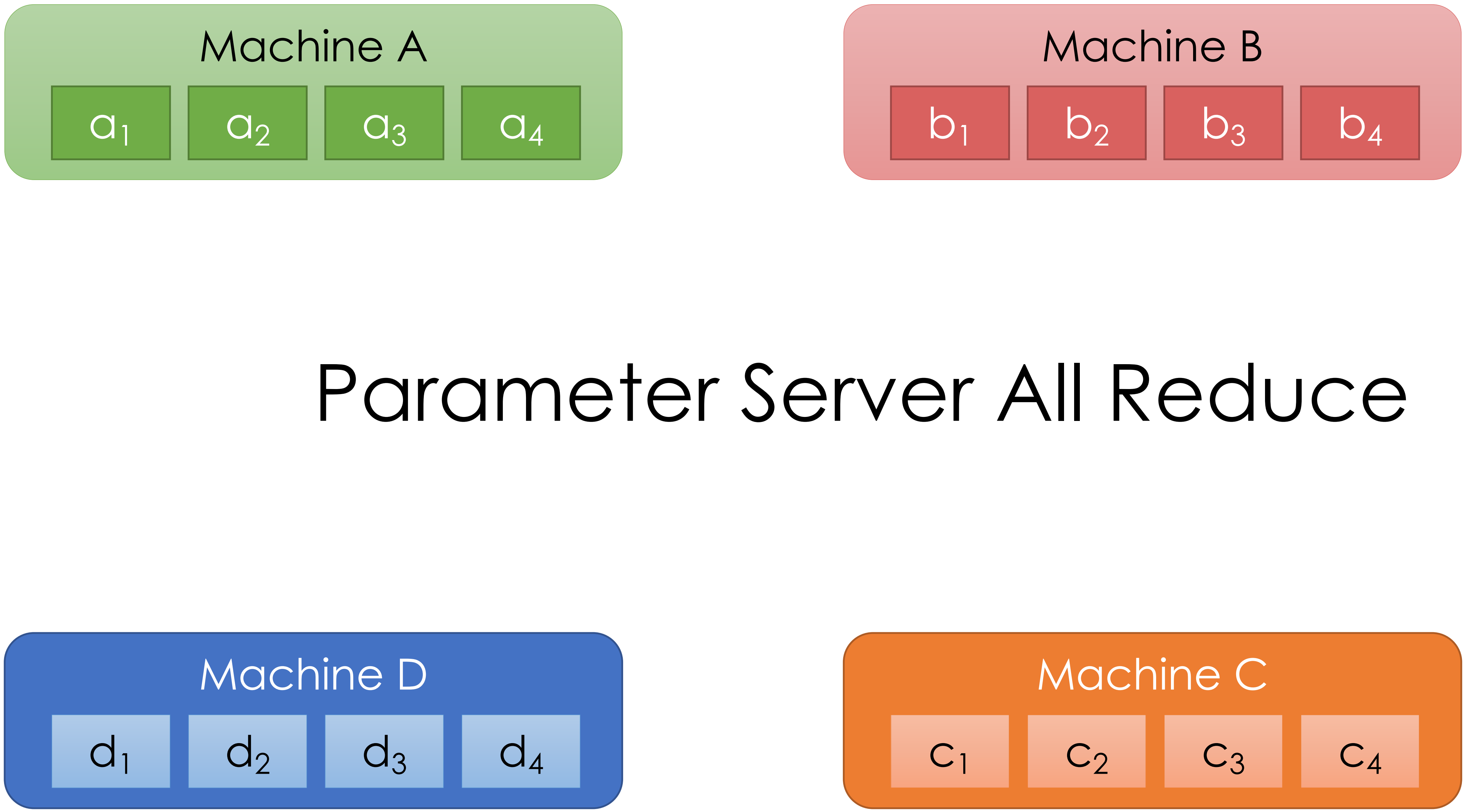


Issues?

➢ High **fan-in** on Machine A

➢ **(P-1) * N Bandwidth** for Machine A

Parameter Server All Reduce



Send each entry to parameter server for that entry.

- Key 1 → A
- Key 2 → B
- Key 3 → C
- Key 4 → D

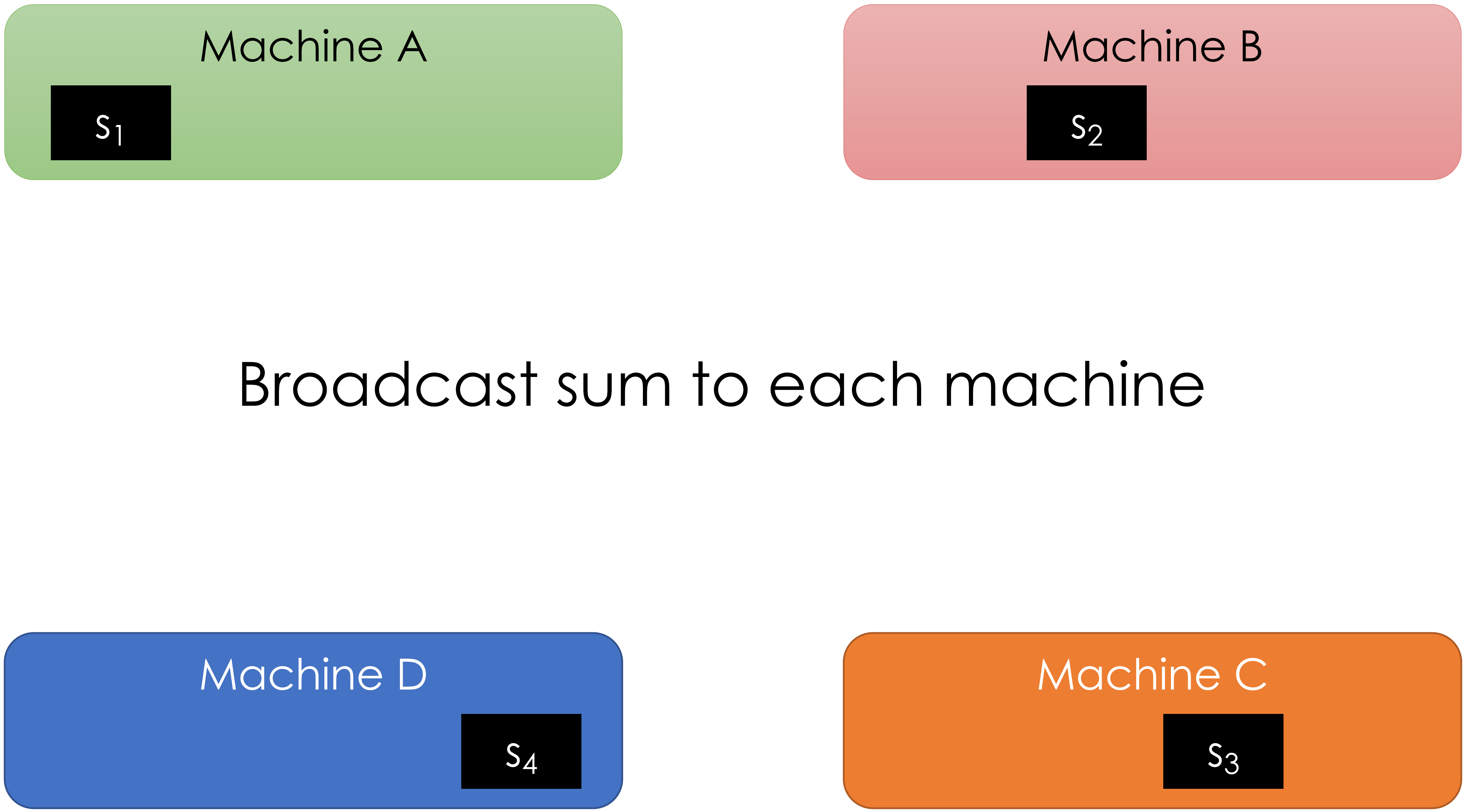
Each machine sends N/P data to all other machines.

P * (P-1) * N/P = (P-1) * N

- **P** Machines
- **N** Parameters

Compute local sum on each machine

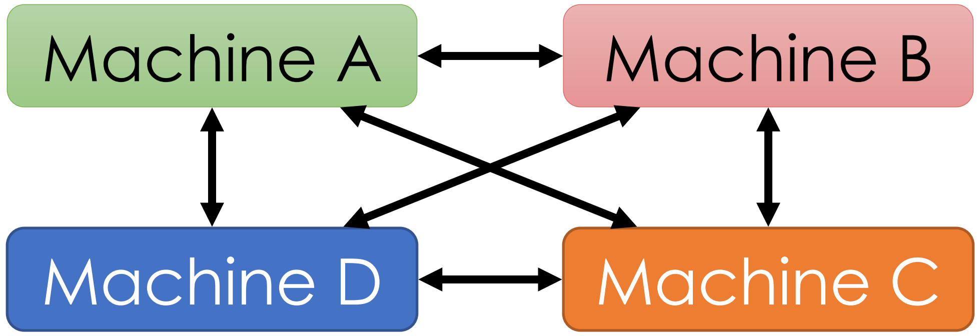
$$s_i = a_i + b_i + c_i + d_i$$


Machine A
s_1
Machine B
s_2
Broadcast sum to each machine
Machine D
s_4
Machine C
s_3

Broadcast sum to each machine

Parameter Server All-Reduce

➢ Same amount of data transmitted as before



➢ Same **high fan-in** (P-1)

➢ **Reduced** Inbound Bandwidth = (P-1)N/P

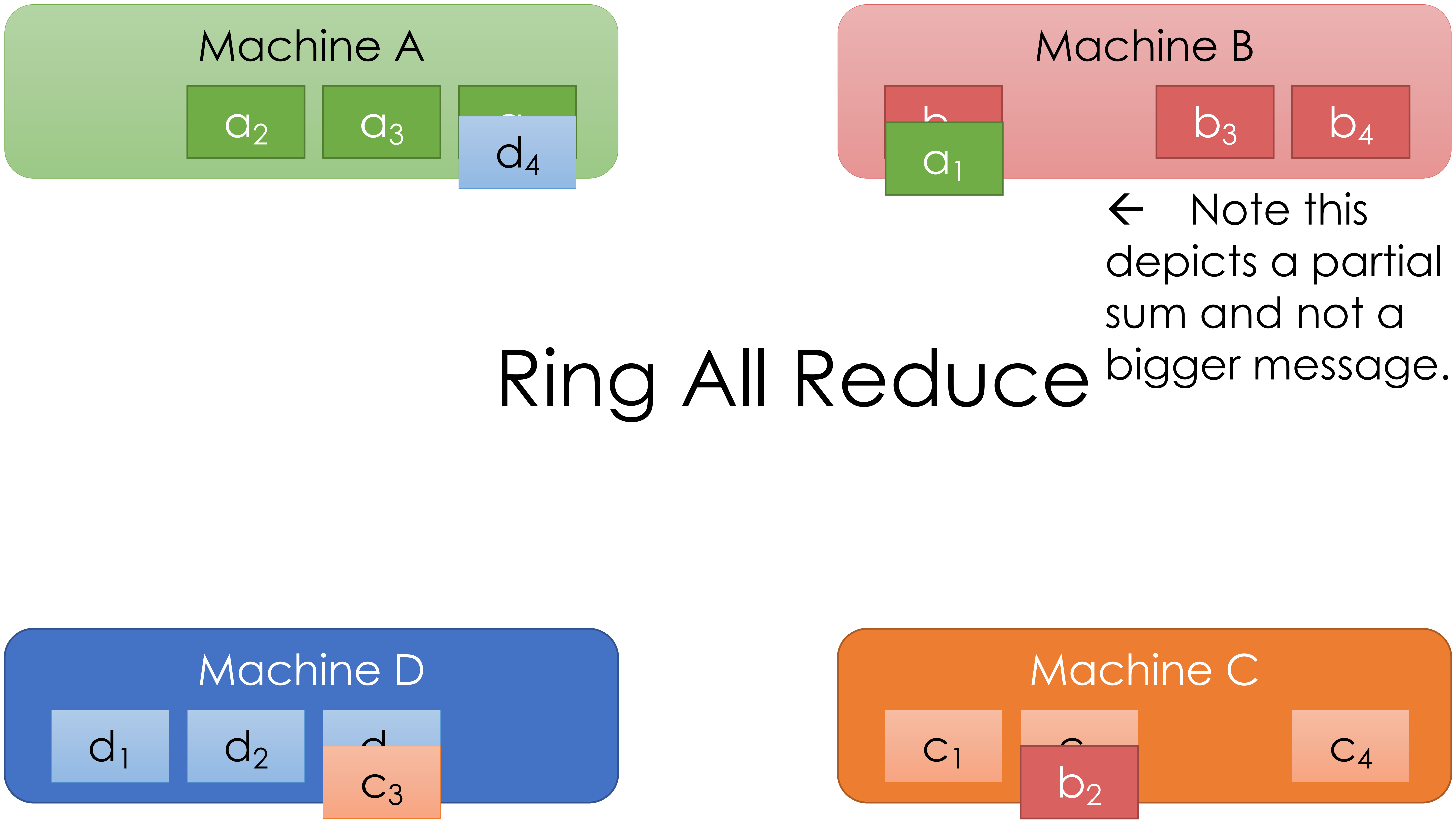
  ➢ Previously (P-1)*N

Ring All Reduce

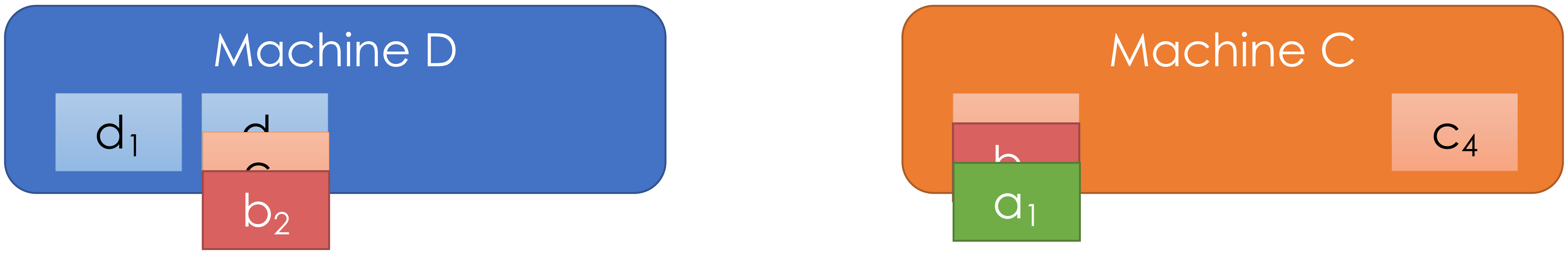
Send messages in a ring using to reduce fan-in.

Ring All Reduce

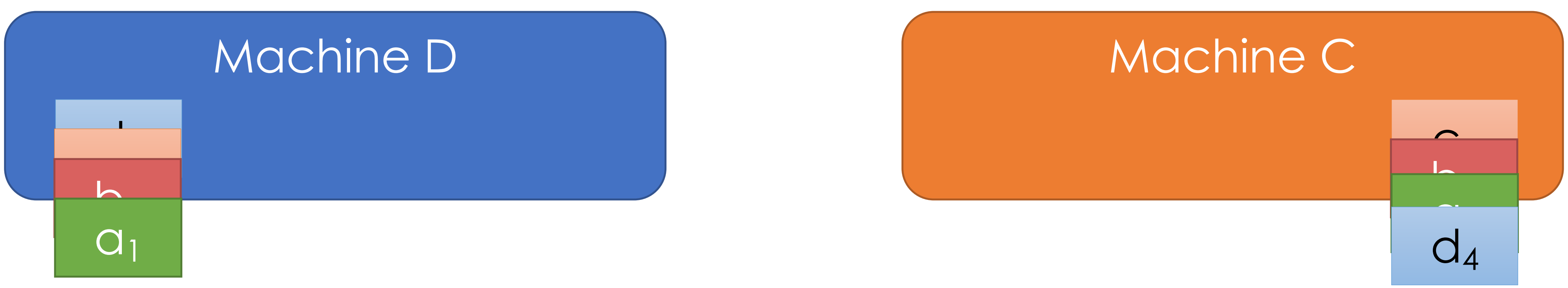
← Note this depicts a partial sum and not a bigger message.



Ring All Reduce



Ring All Reduce



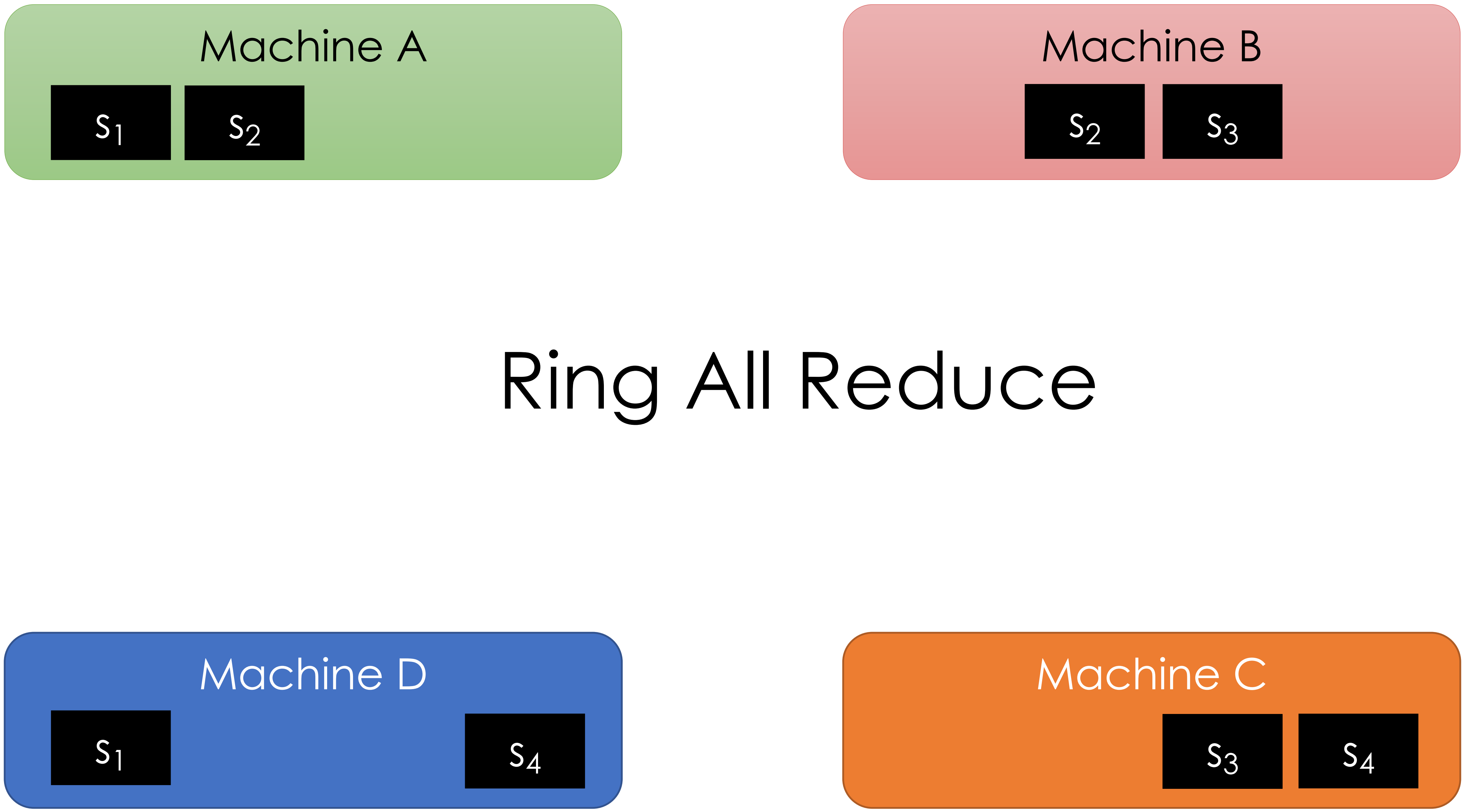
Each machine sends N/P data to next machine each of (p-1) rounds:

(P-1) * P * N/P = (P-1) * N

- **Bandwidth** per round:
  - **P (N/P) = N** (doesn't depend on P)
- **Fan-in Per Round:**
  - 1 (doesn't depend on P)

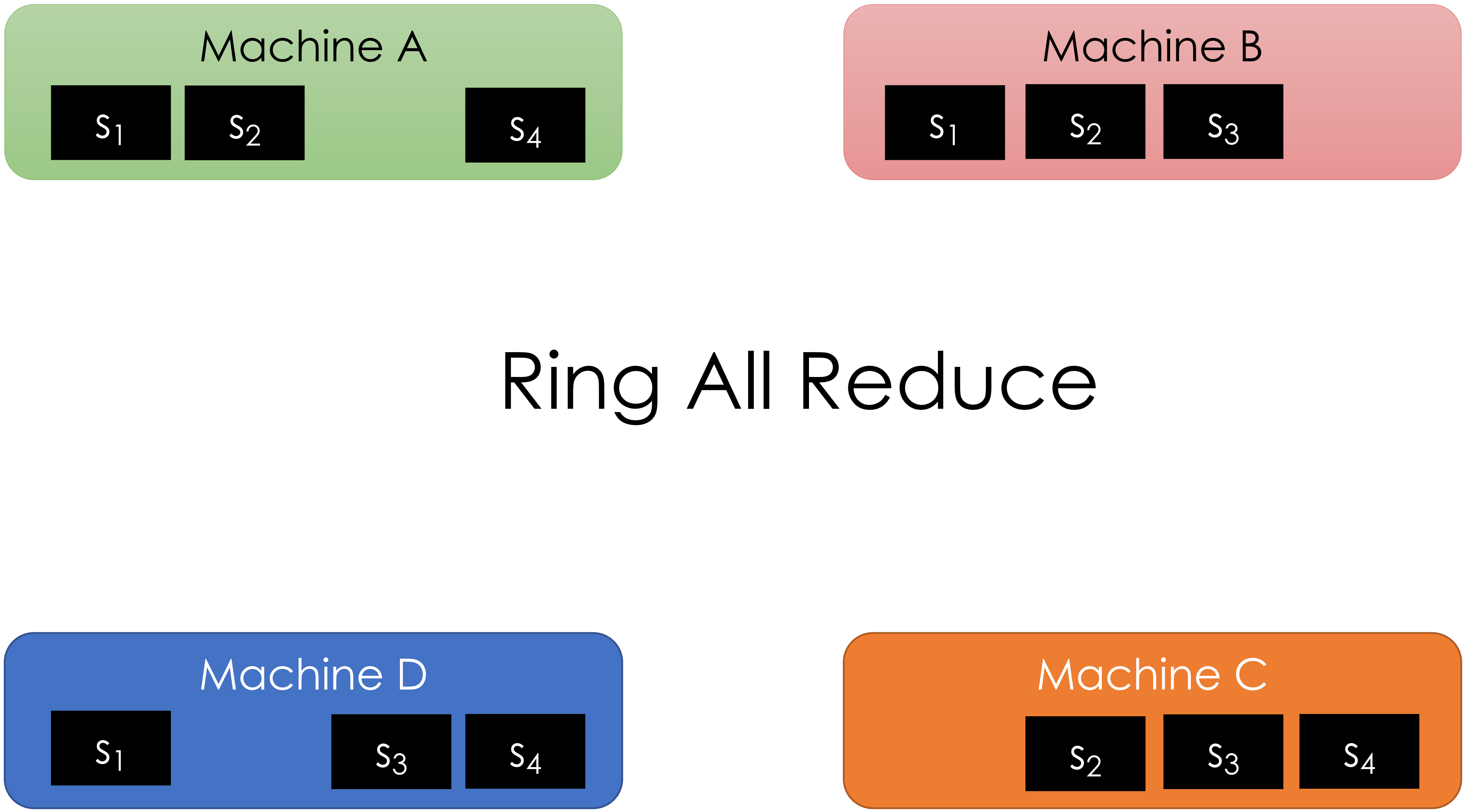
Ring All Reduce

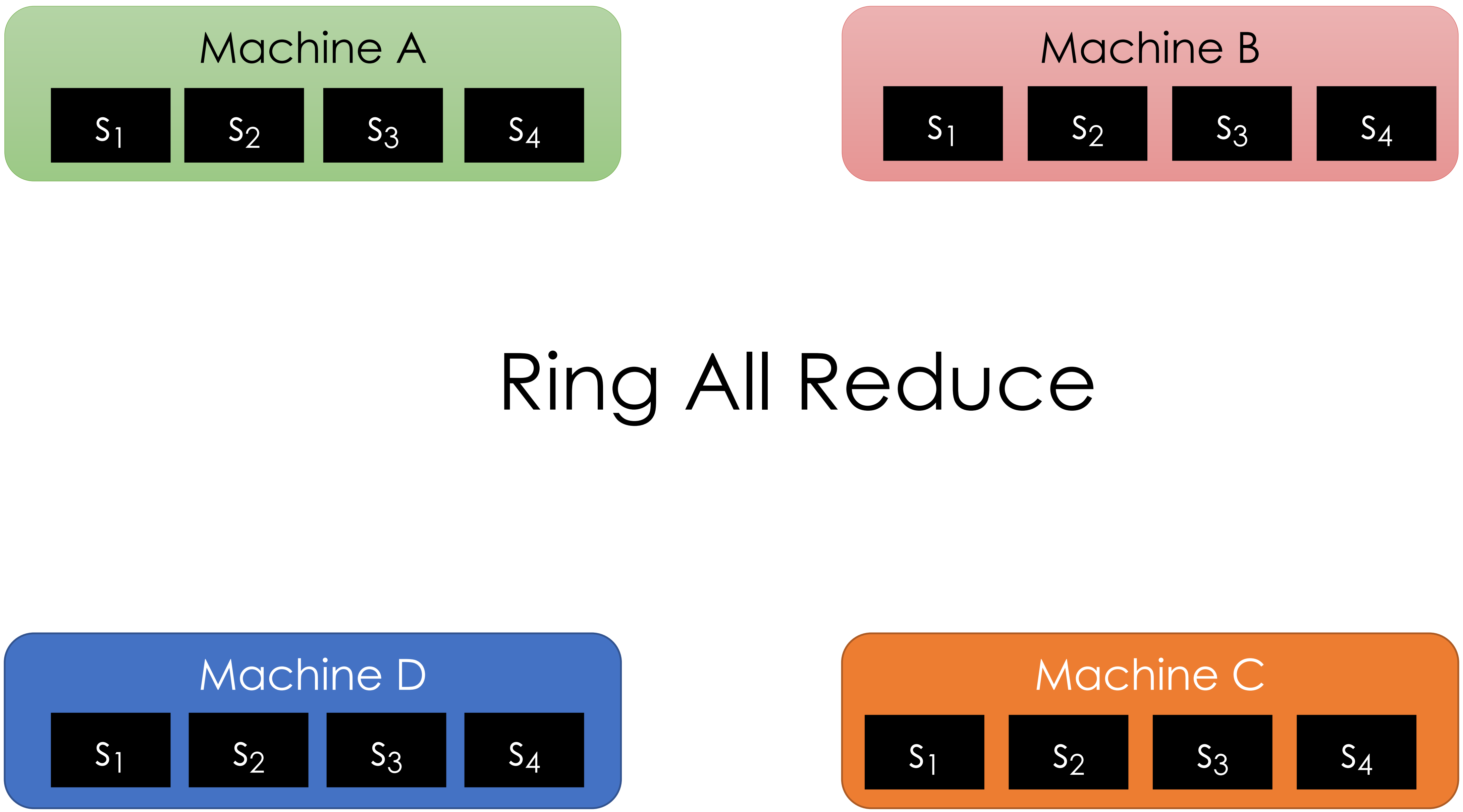
Broadcast stage repeats process sending messages forw



Machine A
S1
S2
Machine B
S2
S3
Ring All Reduce
Machine D
S1
S4
Machine C
S3
S4

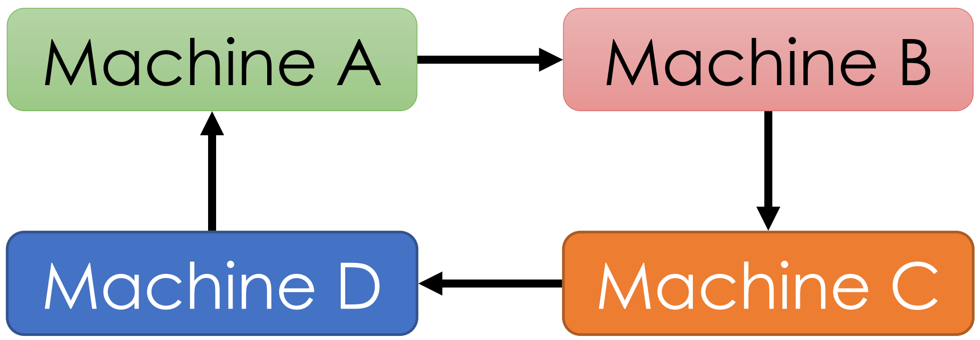
Ring All Reduce




Machine A
S1 S2 S3 S4
Machine B
S1 S2 S3 S4
Ring All Reduce
Machine D
S1 S2 S3 S4
Machine C
S1 S2 S3 S4

Ring All-Reduce

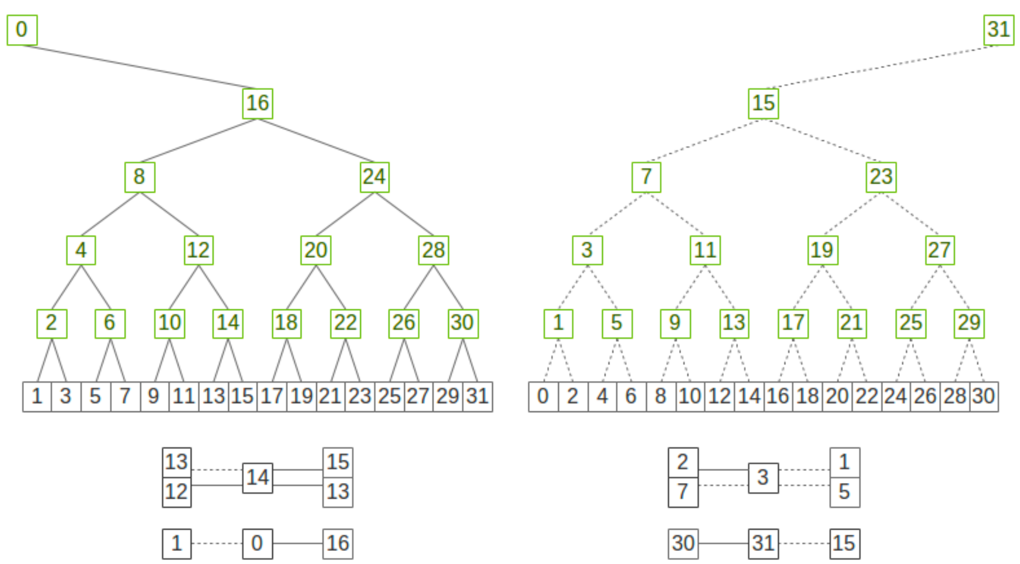
- ➢ Simplified communication topology with low fan-in

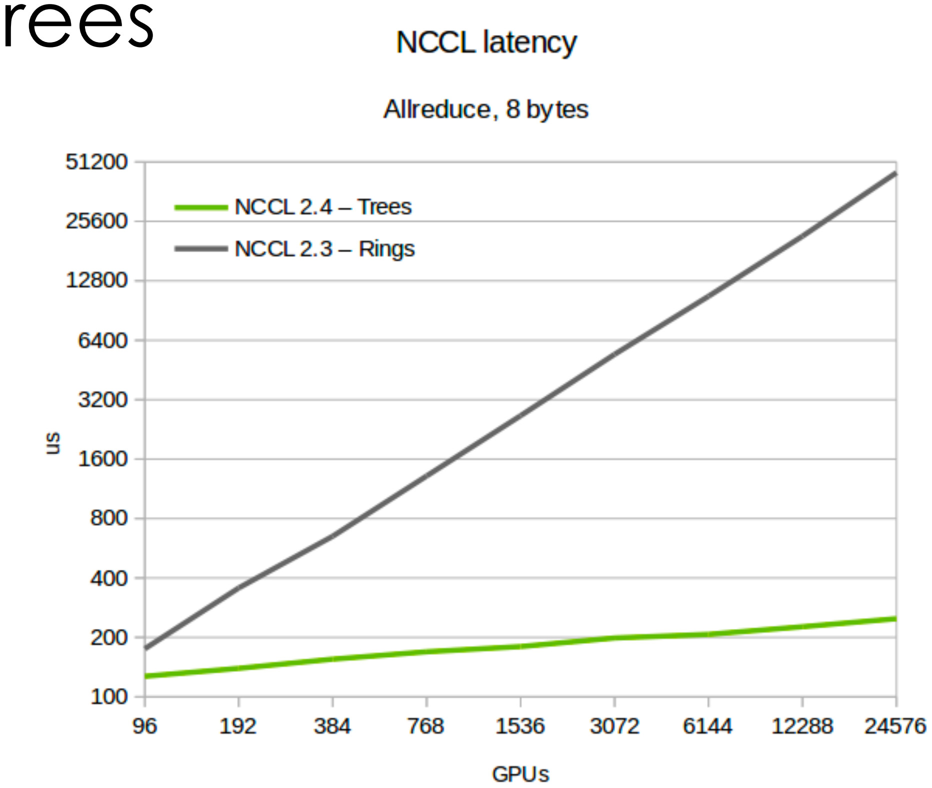


- ➢ Overall communication
  - ➢ Same total communication: **2*(P-1)*N**
  - ➢ **Bandwidth** per round (N) doesn't depend on P
  - ➢ **Fan-in** is constant (doesn't depend on P)
- ➢ **Issue:** Number of communication rounds (P-1)

Double Binary Tree All-Reduce

➢ Two overlaid binary reduction trees





➢ Double the fan-in → Log(p) rounds of communication
- ➢ Currently used on Summit super-computer and latest NCCL

https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/