Where We Are

Machine Learning Systems	2012 - Now
Big Data	2010 - Now
Cloud	2000 - 2016
Foundations of Data Systems	1980 - 2000

Logistics

- Exam date:
  - Final Exam date (tentative): **Friday, March 22, 8 - 11 am, PT**
  - **Decision: In-person Exam**
- Next week:
  - TA will hold multiple hours of Exam review
  - Pay attention to Piazza announcement about scheduling
  - Make sure you attend and get important secret sauces ☺
  - TAs and I will all be available for OH by appointment to help you on exam and wrapping the course!

ML System history

- ML Systems evolve as more and more ML components (models/optimization algorithms) are unified

Ad-hoc: diverse model family, optimization algos, and data

Opt algo: iterative-convergent

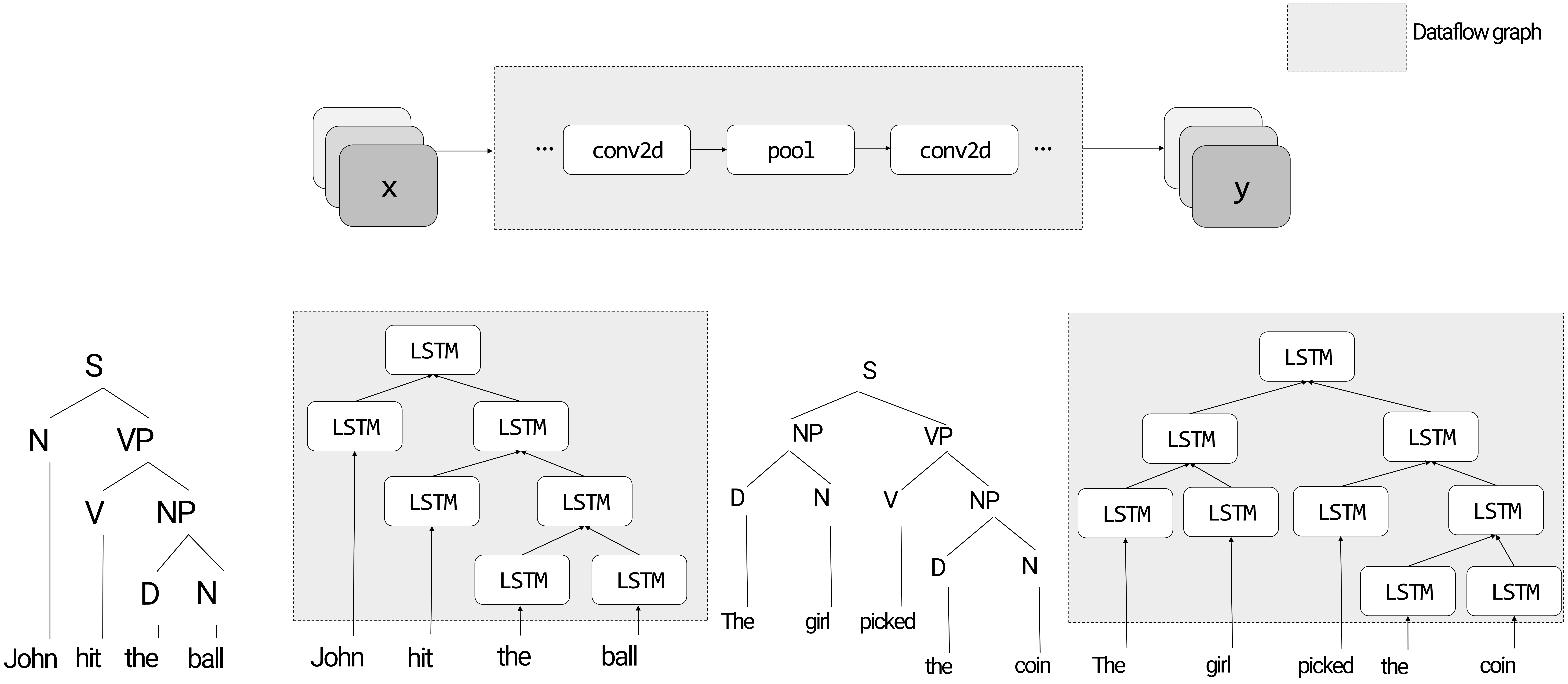
Model family: neural nets

Model: CNNs/transformers/GNNs

LLMs: transformer decoders

Today: NN, data flow graph, and data parallelism

Static Models vs. Dynamic Models



Static vs. Dynamic Dataflow Graphs

- Static Dataflow graphs
  - Define once, execute many times
  - Execution: Once defined, all following computation will **follow** the defined computation
  - Advantages
    - No extra effort for batching optimization, because it can be by nature batched
    - It is always easy to handle a static computational dataflow graphs in all aspects, because of its fixed structure
      - Node placement, distributed runtime, memory management, etc.
    - Benefit the developers

Static vs. Dynamic Dataflow Graphs

- Can we handle dynamic dataflow graphs?
  - Difficulty in expressing complex flow-control logic
  - Complexity of the computation graph implementation
  - Difficulty in debugging

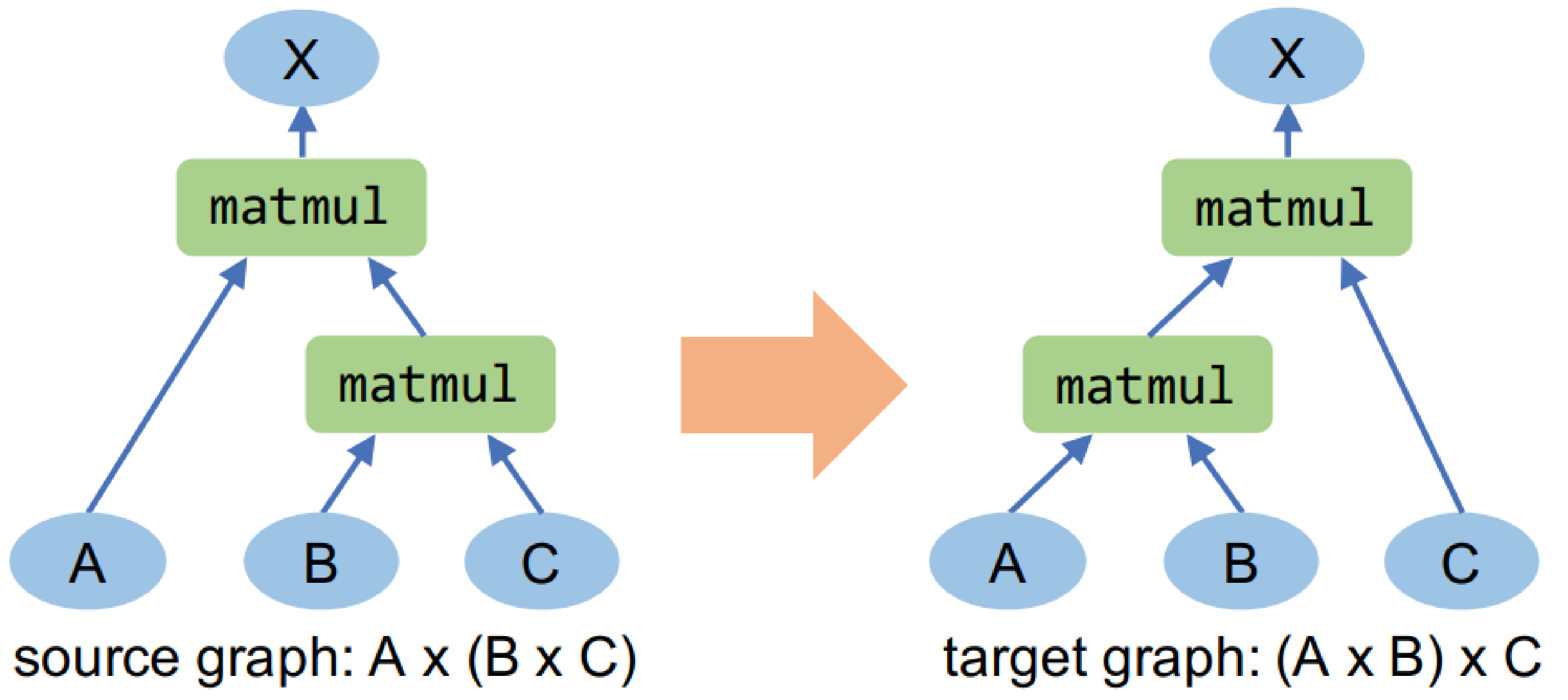
How to Handle Dynamic Dataflow Graph?

- In general two ways:
  - Imperative: do not requiring contracting the entire graph before execution
  - Other symbolic representation on top of dataflow graph
    - vertex-centric representation

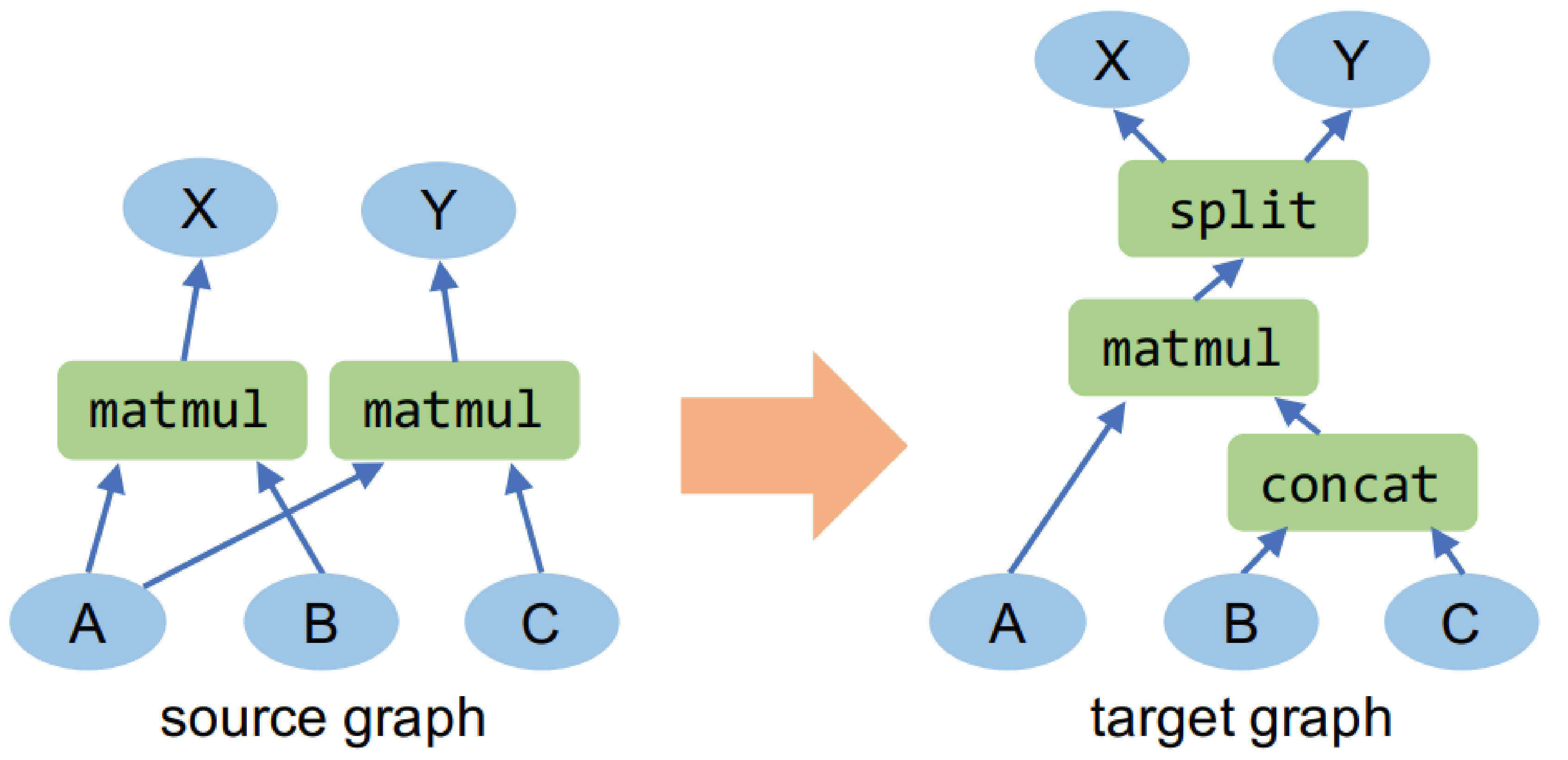
Questions

- Is **CNN training** static or dynamic graph?
- Is **CNN inference** static or dynamic graph?
- Is **GPT-3 (transformers decoder)** training static graph or dynamic?
- Is **GPT-3 inference with batch size = 1** static or dynamic graph
- Is **GPT-3 serving** static or dynamic graph

Advanced Topic: DL Dataflow Graph Optimization



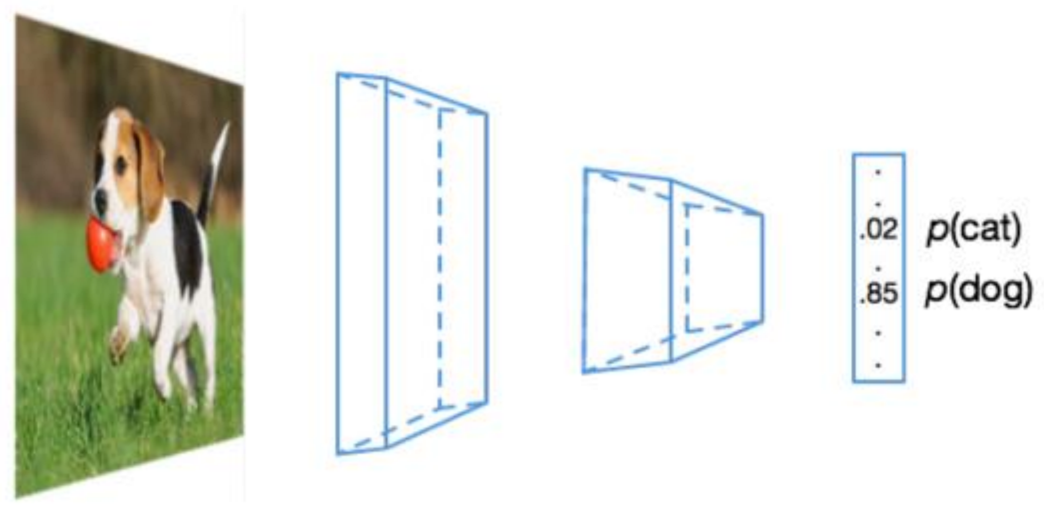
(a) Associativity of matrix multiplication.



(b) Fusing two matrix multiplications using concatenation and split.

Advanced Topic: DL Graph Compilation

ML Models



High-level IR Optimizations and Transformations

Tensor Operator Level Optimization

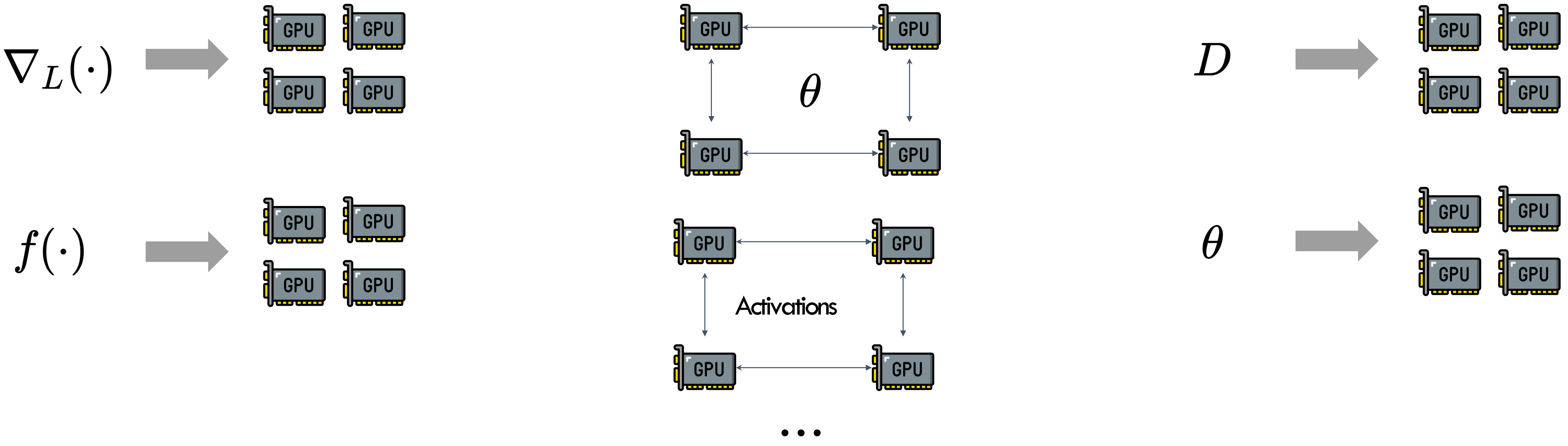
Direct code generation

Where We Are

- Deep Learning as Dataflow Graphs
- Auto-differentiation Libraries
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- **DL Parallelism**

DL Parallelization: 3 Core Problems

Computing | **Communication** | **Memory**



$$\theta^{(t+1)} = f\left(\theta^{(t)}, \nabla_L\left(\theta^{(t)}, D^{(t)}\right)\right)$$

parameter

weight update (sgd, adam, etc.)

model (CNN, GPT, etc.)

data

Two Views of ML Parallelisms

Classic view

Data parallelism

Model parallelism

New view

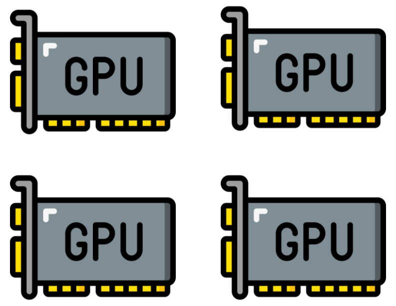
Inter-op parallelism

Intra-op parallelism

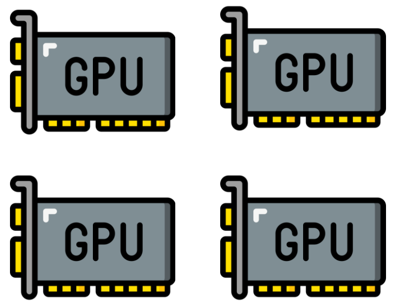
Data and Model Parallelism

Data parallelism

D partition

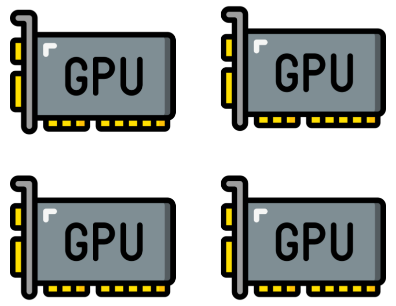


$\theta,\ \nabla_L(\cdot),\ f(\cdot)$ replicate

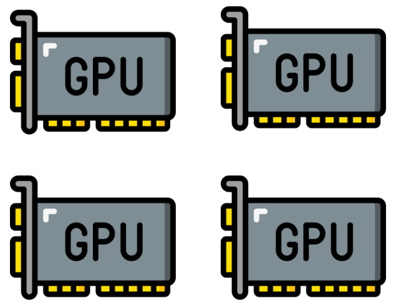


Model parallelism

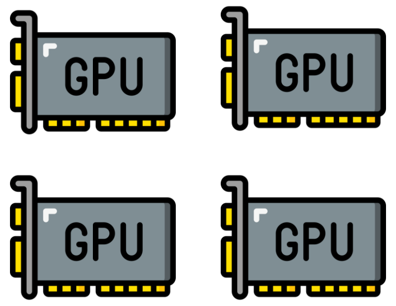
$\theta,\ \nabla_L(\cdot)$ partition



D ?

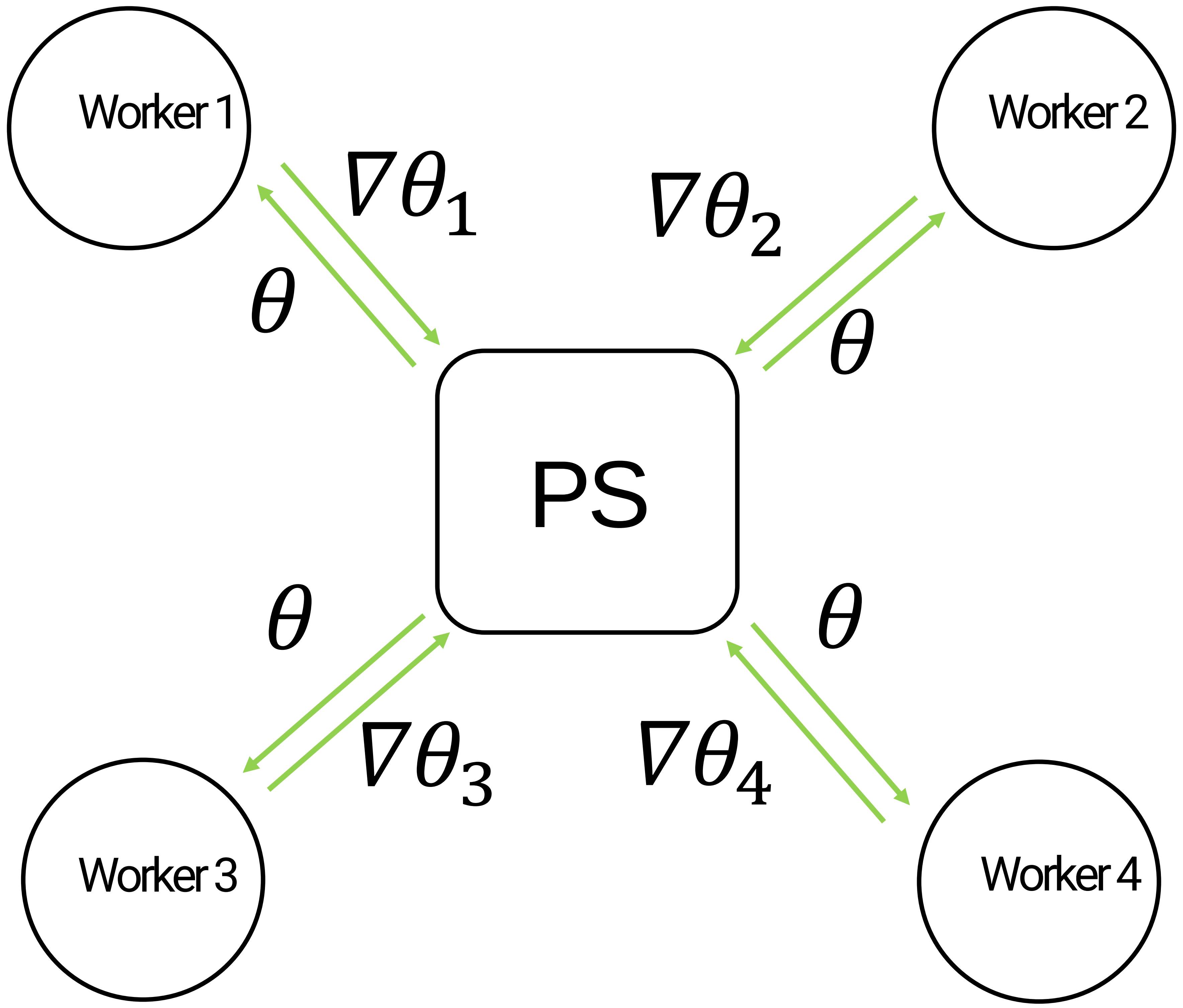


$f(\cdot)$?



$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update}\\ \text{(sgd, adam, etc.)}}}\Big(\theta^{(t)},\ \underbrace{\nabla_L}_{\substack{\text{model}\\ \text{(CNN, GPT, etc.)}}}\big(\theta^{(t)},\ \underbrace{D^{(t)}}_{\text{data}}\big)\Big)$$

PS Implements Data Parallelism



In total P workers

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

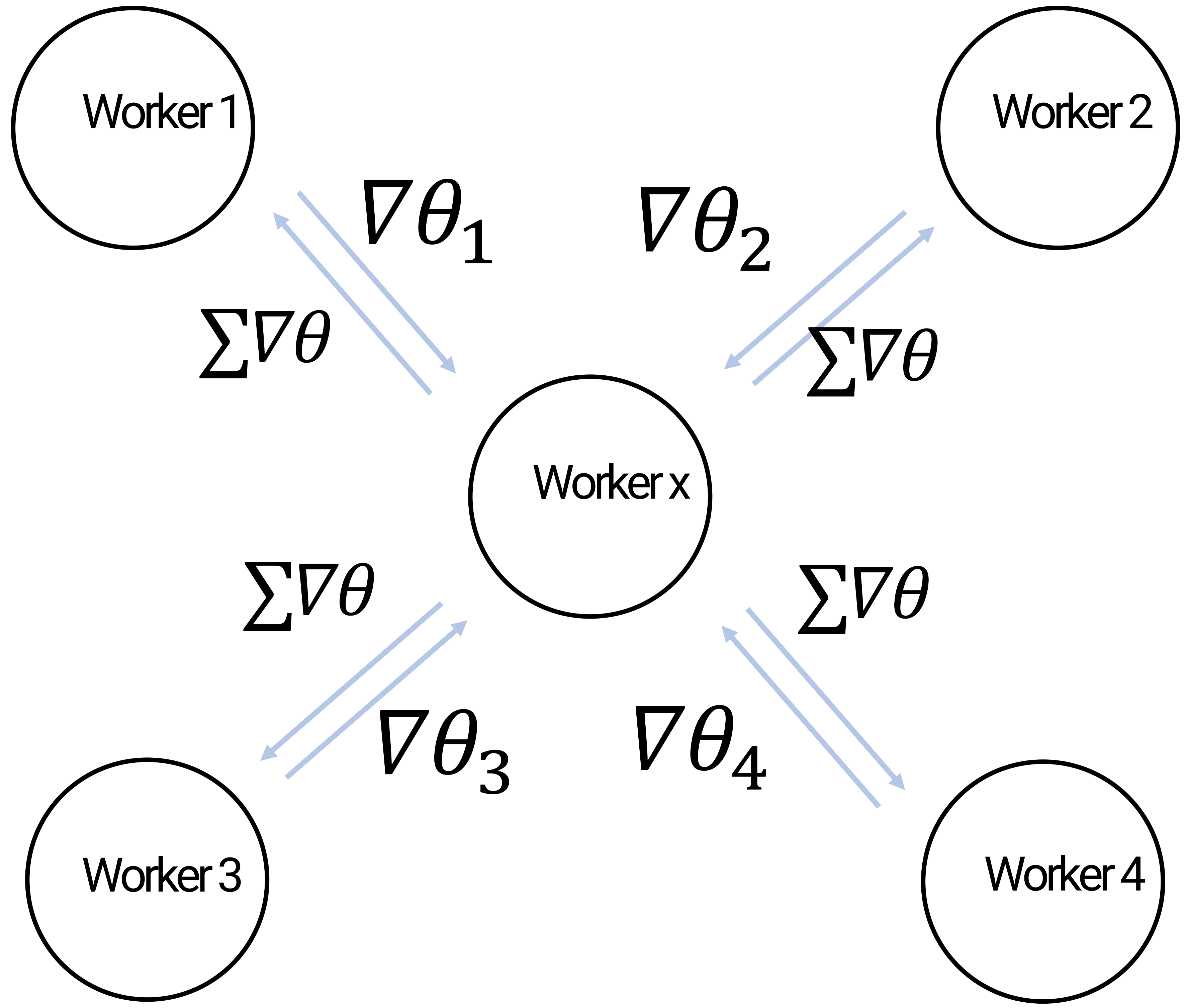
PS collects, aggregates, and applies the gradients, and then broadcast the parameters back to workers

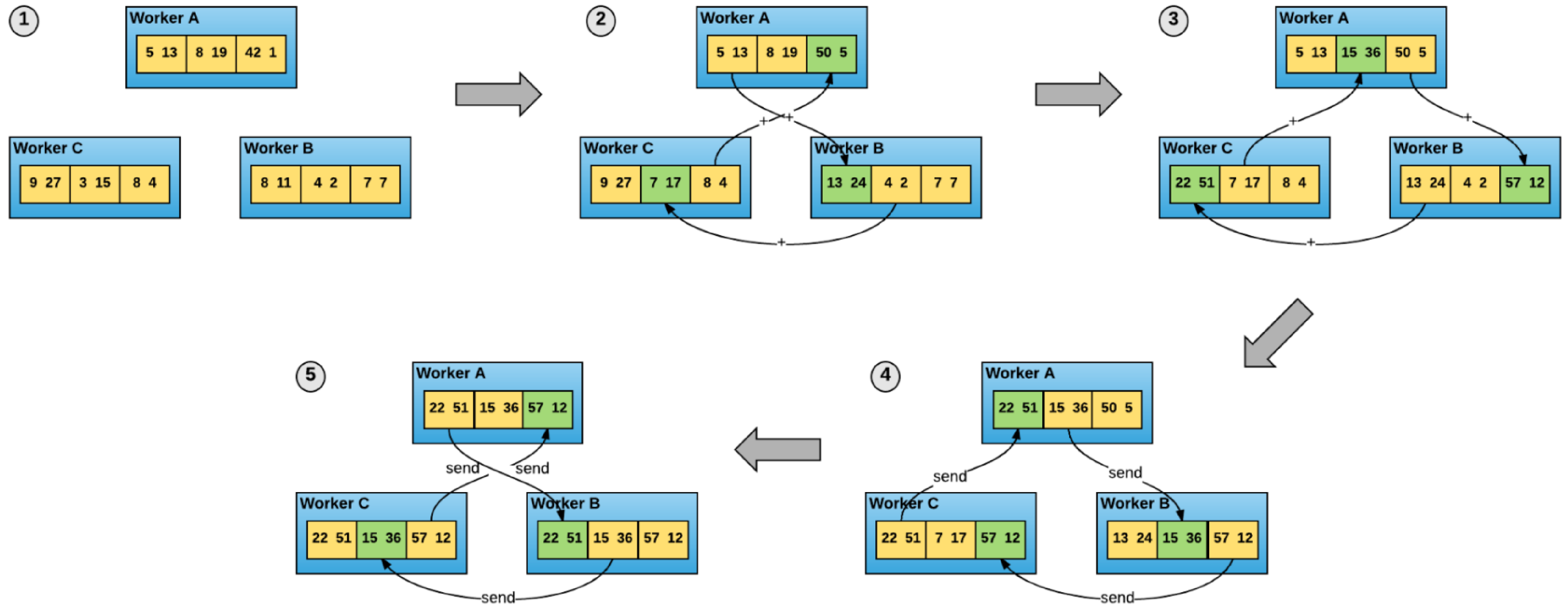
Happening locally on each worker

Data partition p

Representee Systems: Poseidon, GeePS, BytePS, etc.

AllReduce Can Also Handle Data Parallelism Comm



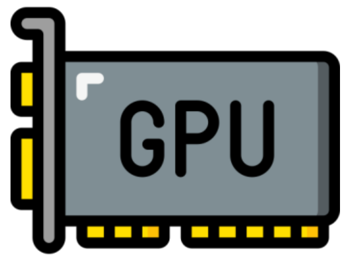


Representee Systems: Horovod, Torch.DDP

Big Model: The Core Computational Challenge

BERT
340M parameters (680 MB)

<



Device Memory
16 - 40 GB

<<

GPT-3
175B parameters (350 GB)

How to train and serve big models?

Model Parallelism

Two Views of ML Parallelisms

Data and model parallelism

- Two pillars: **data** and **model**.
- ✅ “Data parallelism” is general and precise.
- ❓ “Model parallelism” is vague.
- ❓ The view creates ambiguity for methods that neither partitions data nor the model computation.

New: Inter-op and Intra-op parallelism.

- Two pillars: **computational graph** and **device cluster**
- ✅ This view is based on their computing characteristics.
- ✅ This view facilitates the development of new parallelism methods.

Device Cluster

Nvidia DGX with V100

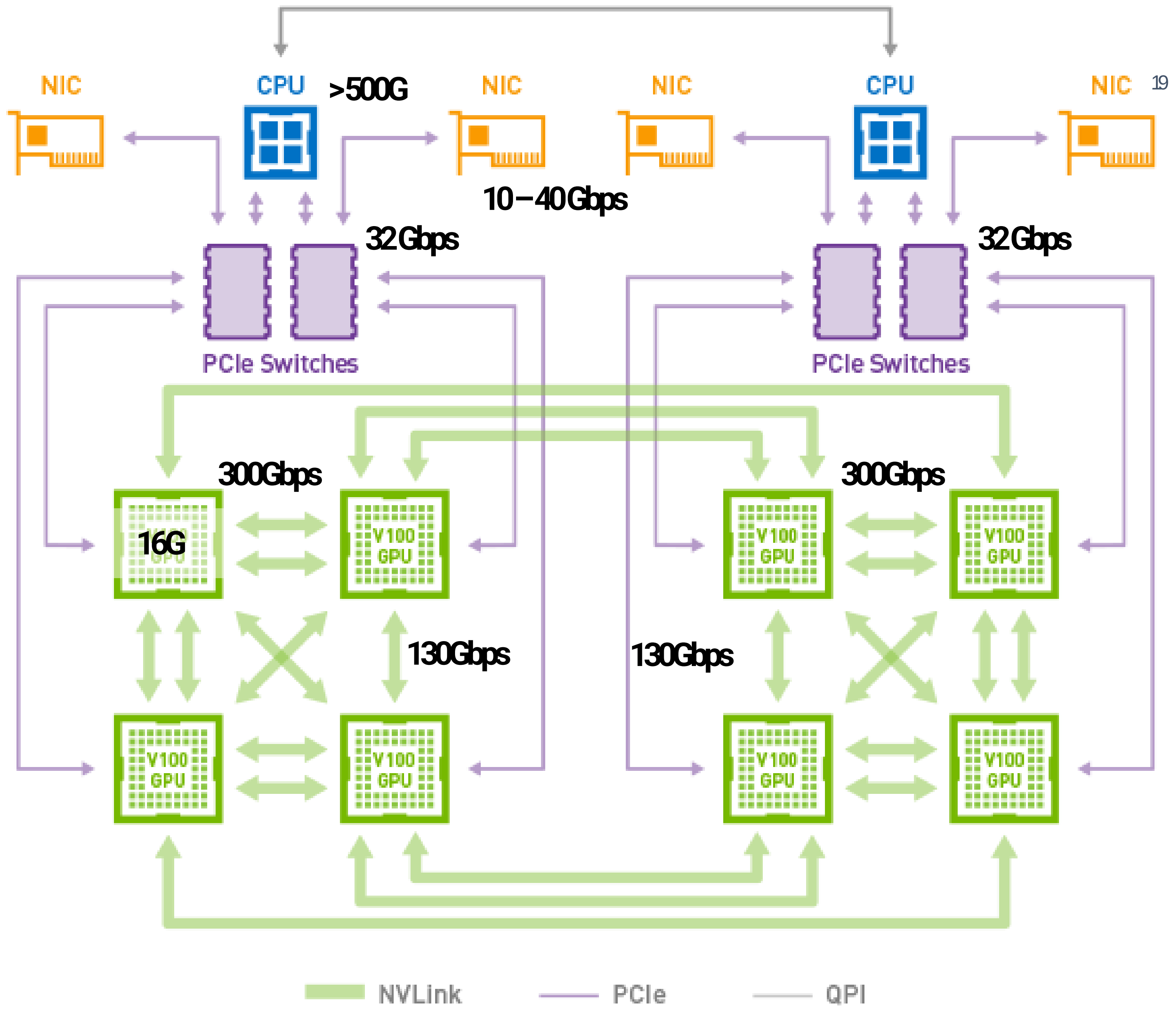
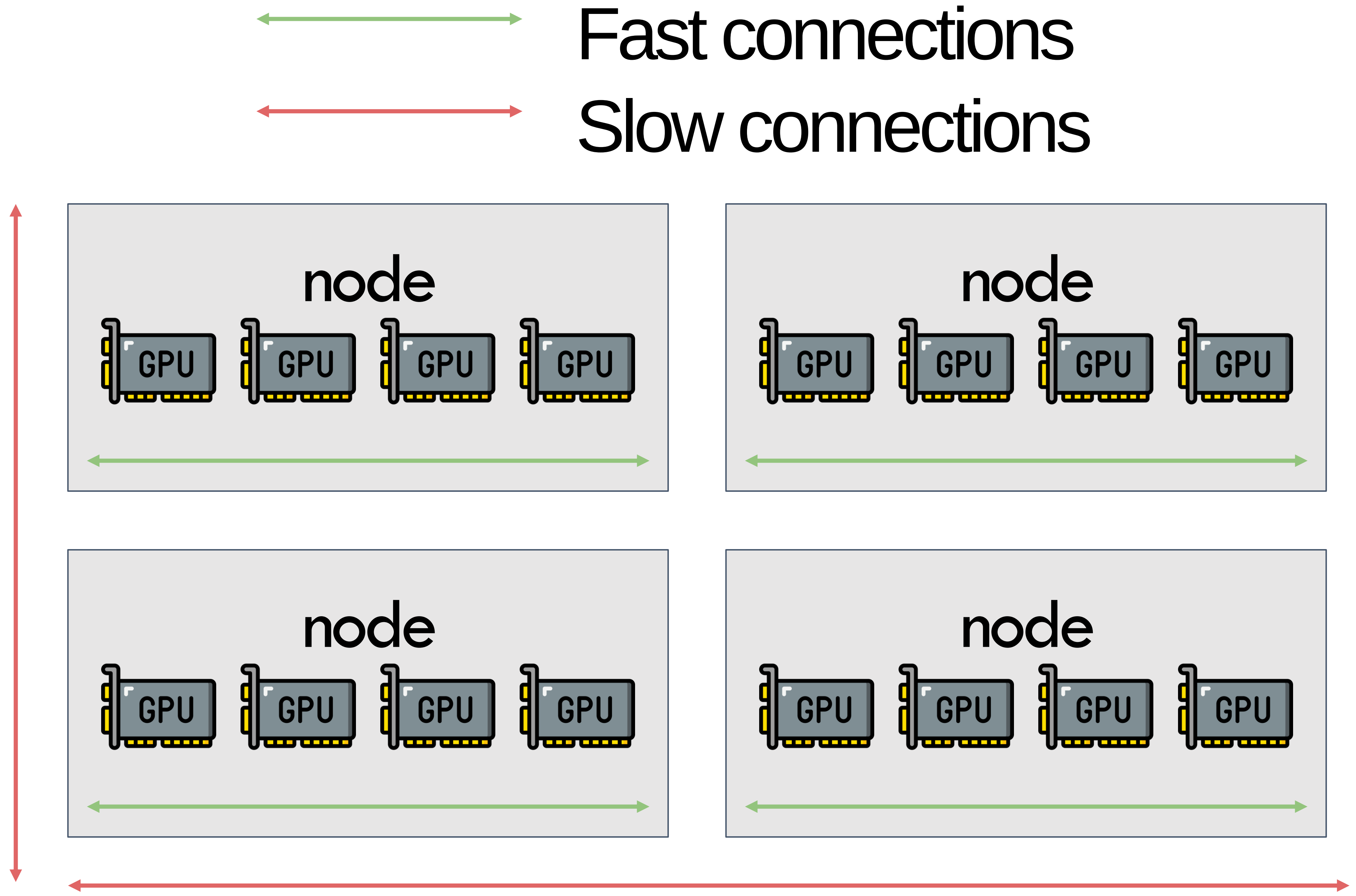


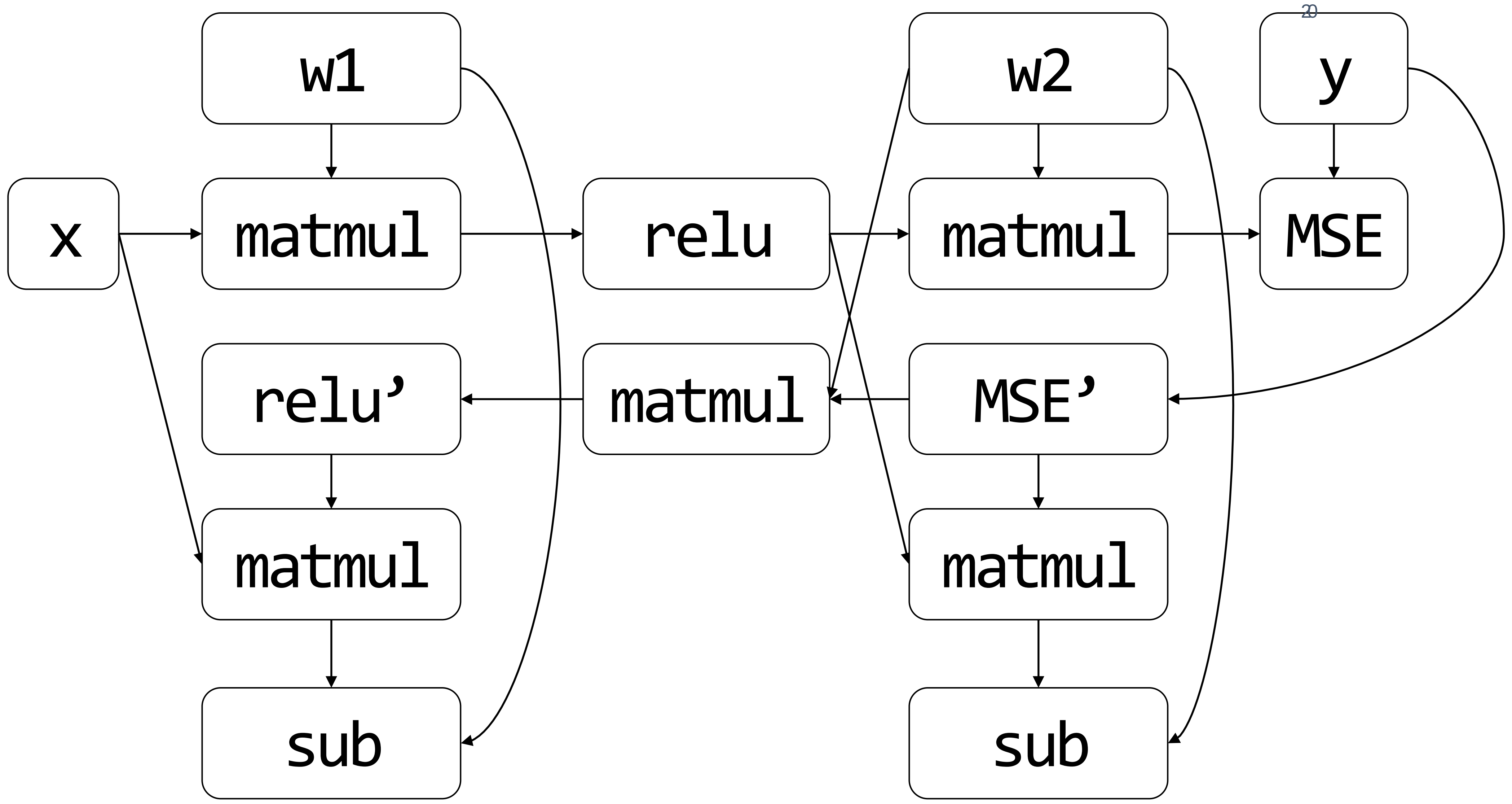
Figure from NVIDIA

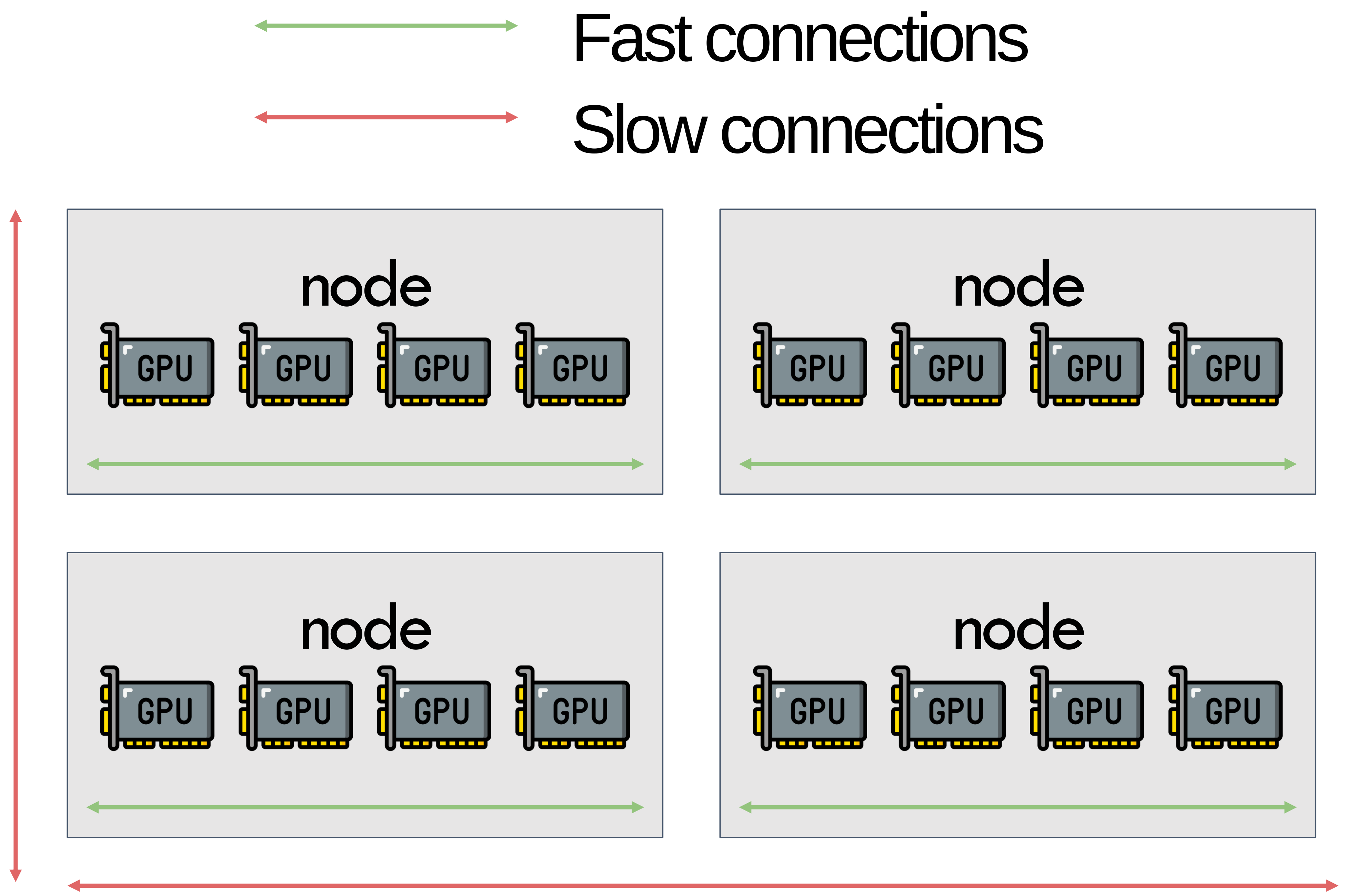
A typical GPU cluster topology



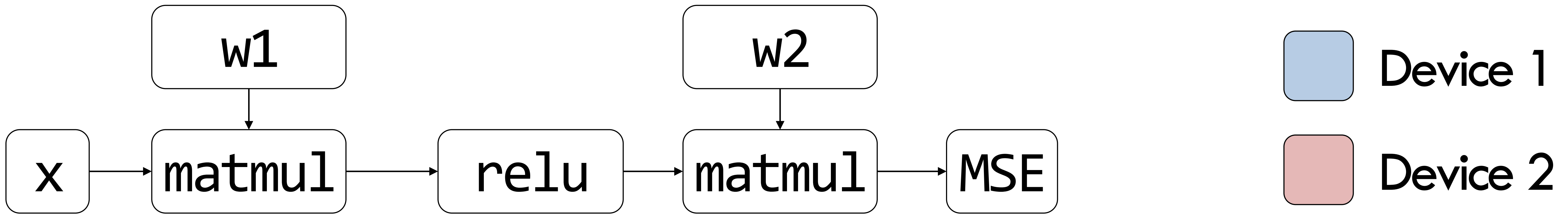
Partitioning Computation Graph on Device Cluster

How to partition the computational graph on the device cluster?

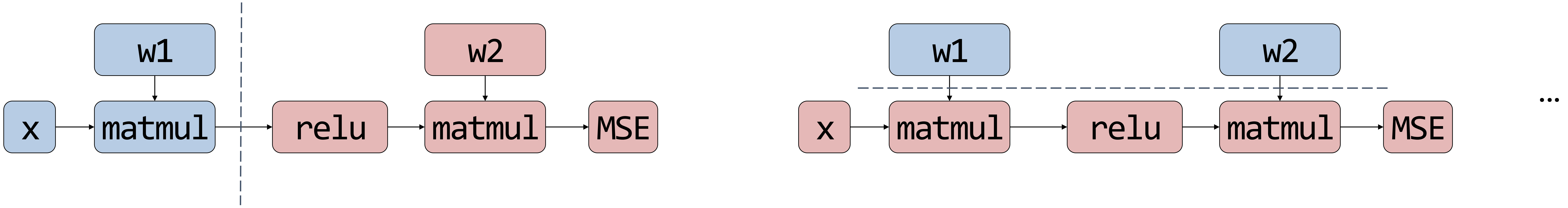




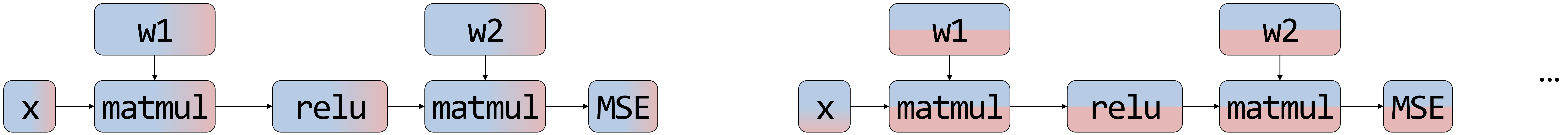
Partitioning Computation Graph



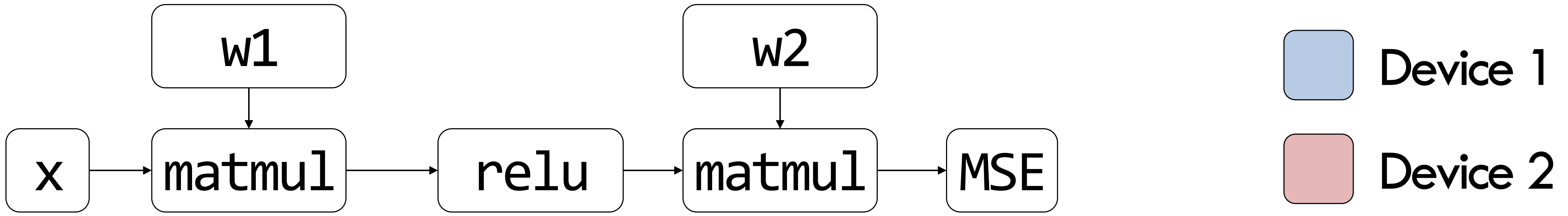
Strategy 1



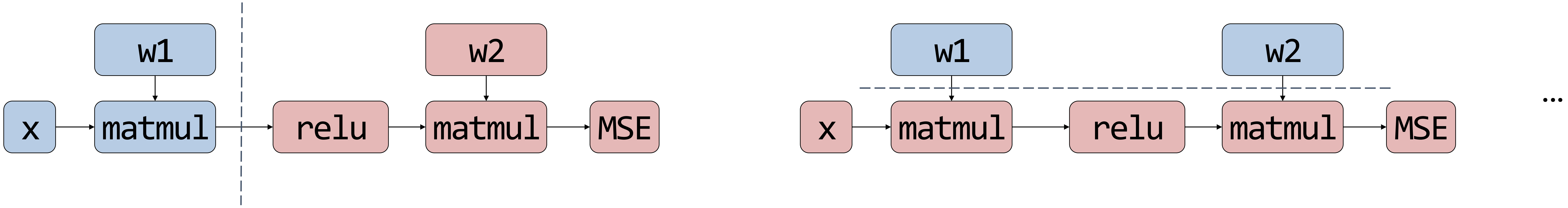
Strategy 2



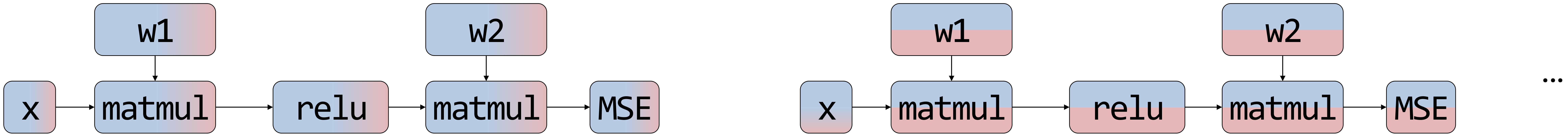
Partitioning Computation Graph



Strategy 1: Inter-operator Parallelism



Strategy 2: Intra-operator Parallelism

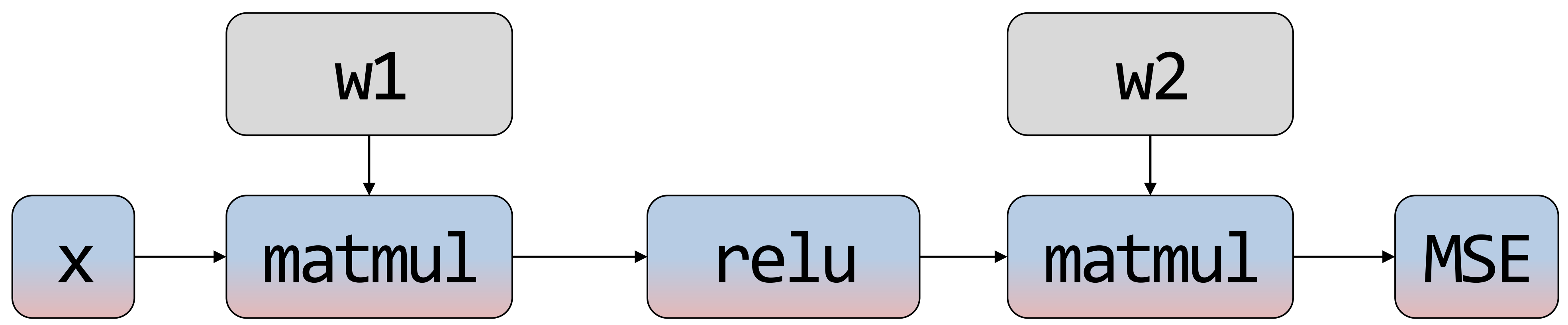


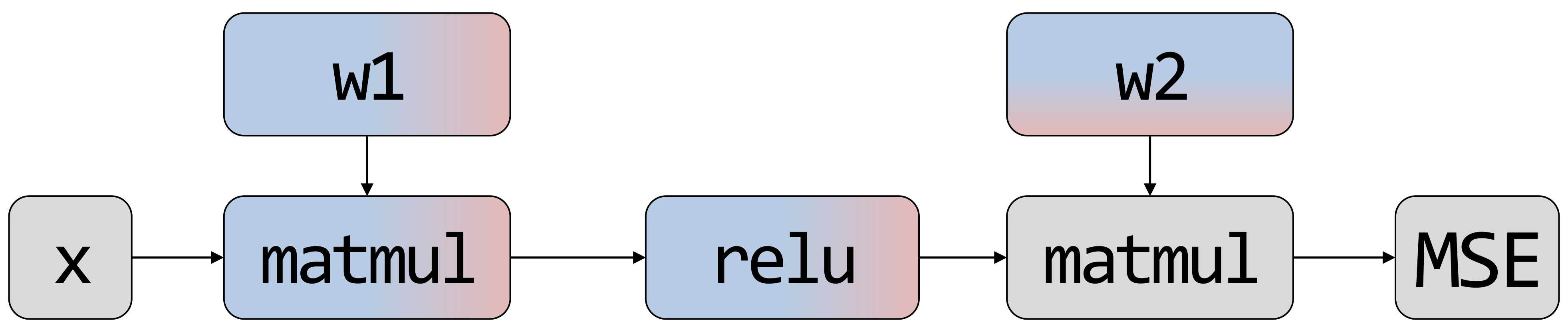
More Parallelisms...

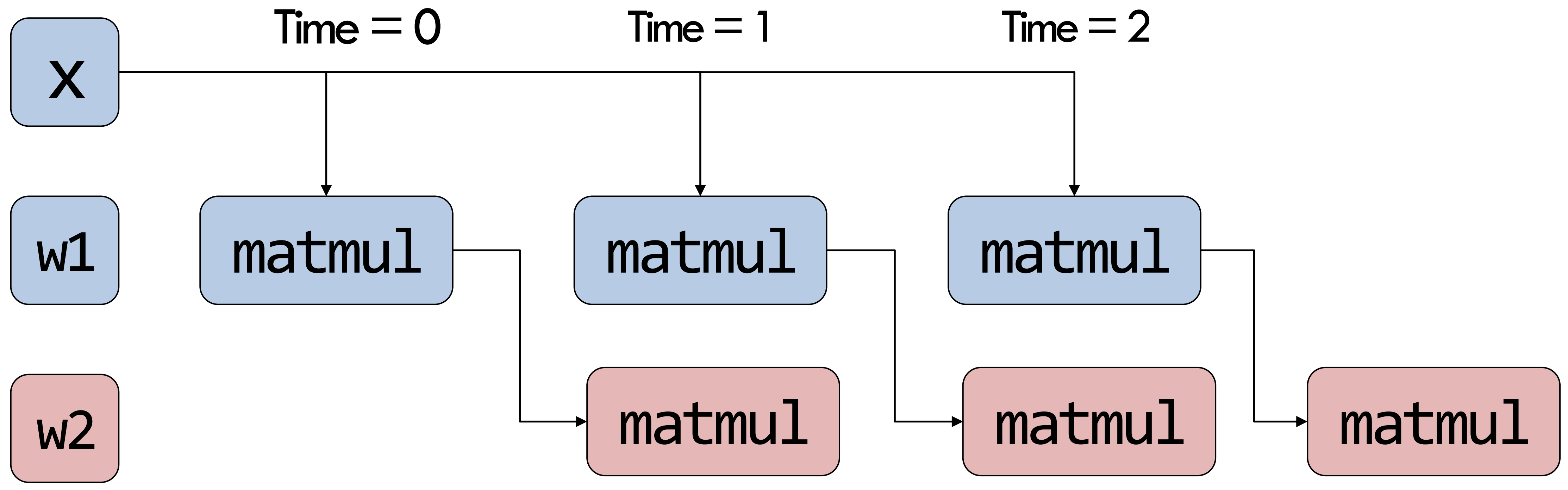
Multiple intra-op strategies for a single node

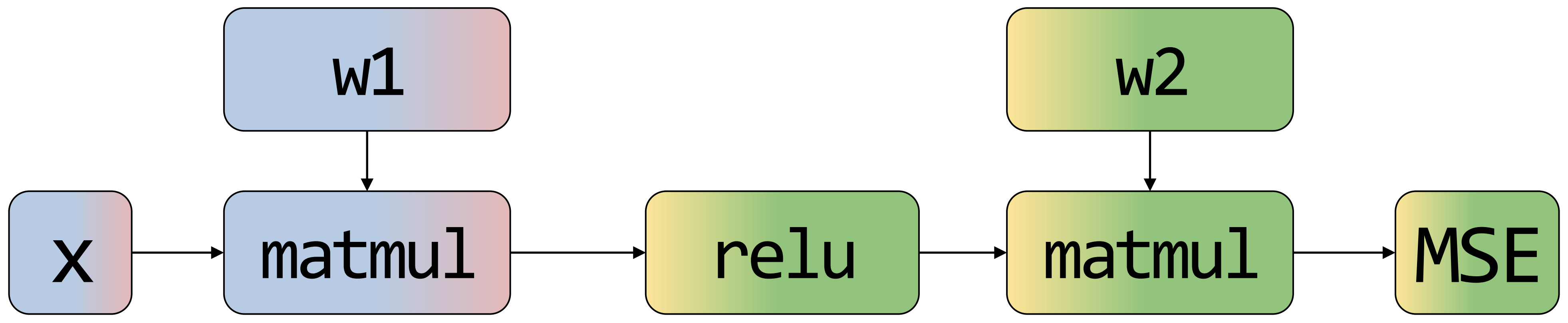
Row-partitioned　Column-partitioned　Replicated　Device 3　Device 4

More strategies

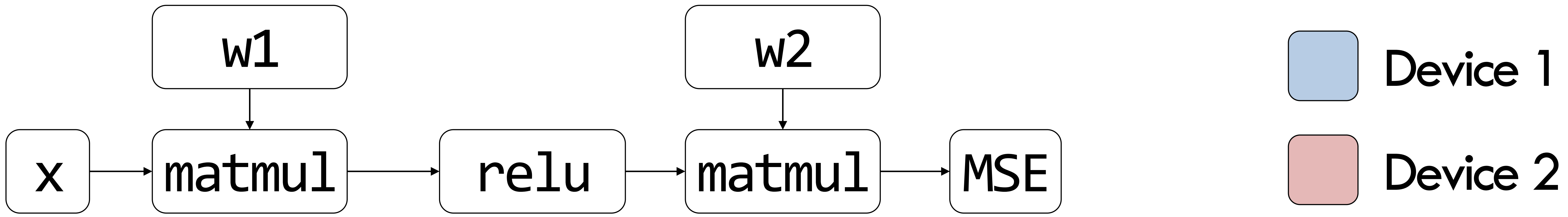




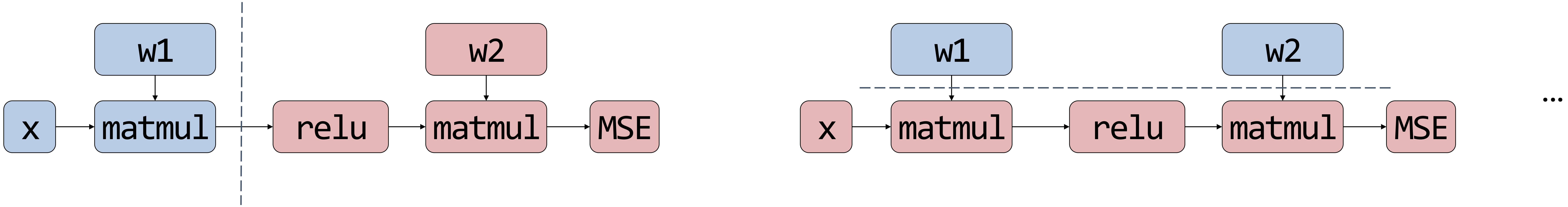




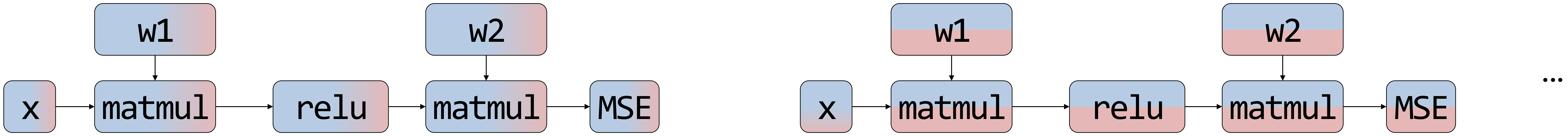
Summary: Inter-op and Intra-op Parallelisms



Inter-op parallelism: Assign different operators to different devices.

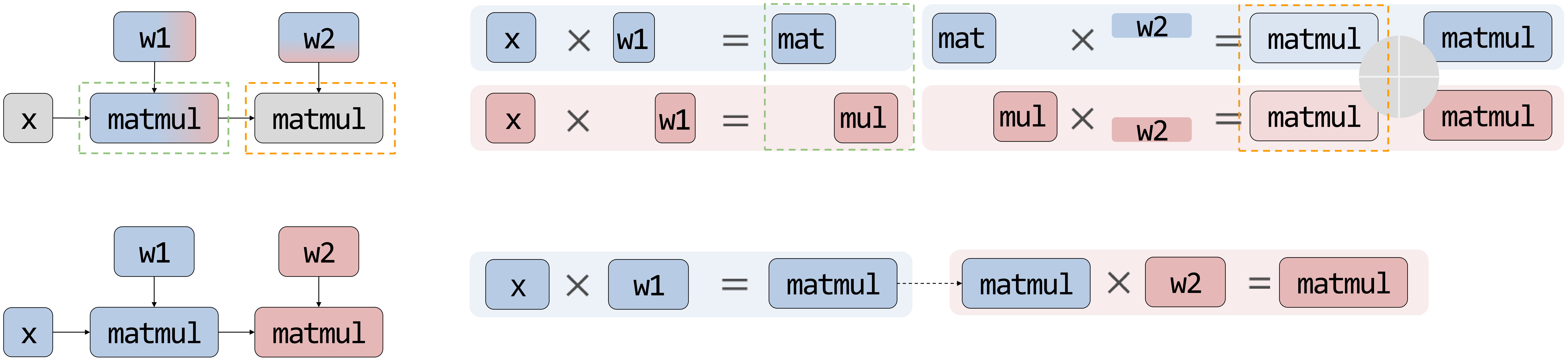


Intra-op parallelism: Assign different regions of a single operator to different devices.

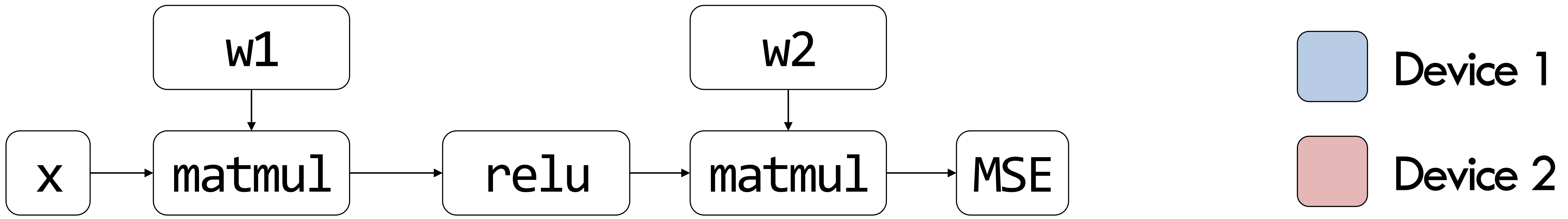


Inside Intra- and Inter-op Parallelism

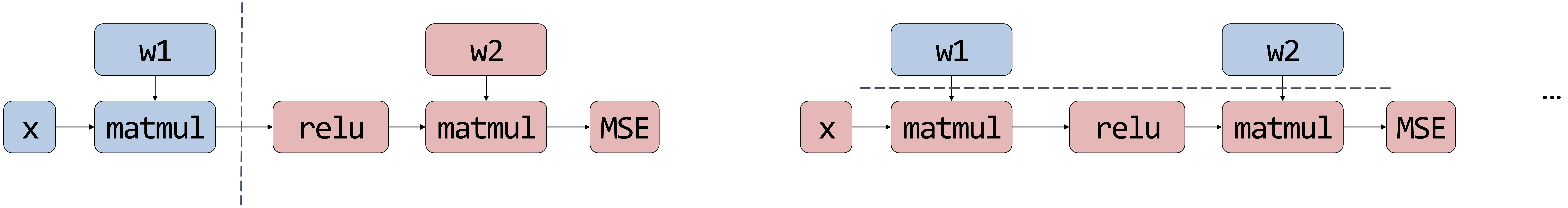
$$Y = X \cdot W_1 \cdot W_2 = X \cdot \begin{bmatrix} W_1^{d1} & W_1^{d2} \end{bmatrix} \cdot \begin{bmatrix} W_2^{d1} \\ W_2^{d2} \end{bmatrix}$$



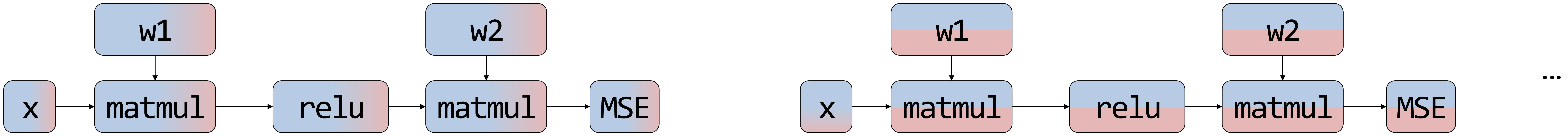
Inter-op and Intra-op Parallelism: Characteristics



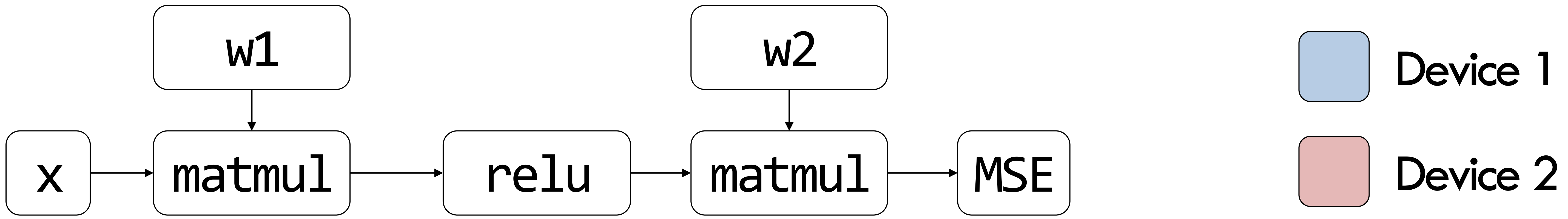
Inter-op parallelism: Requires point-to-point communication but results in device idle



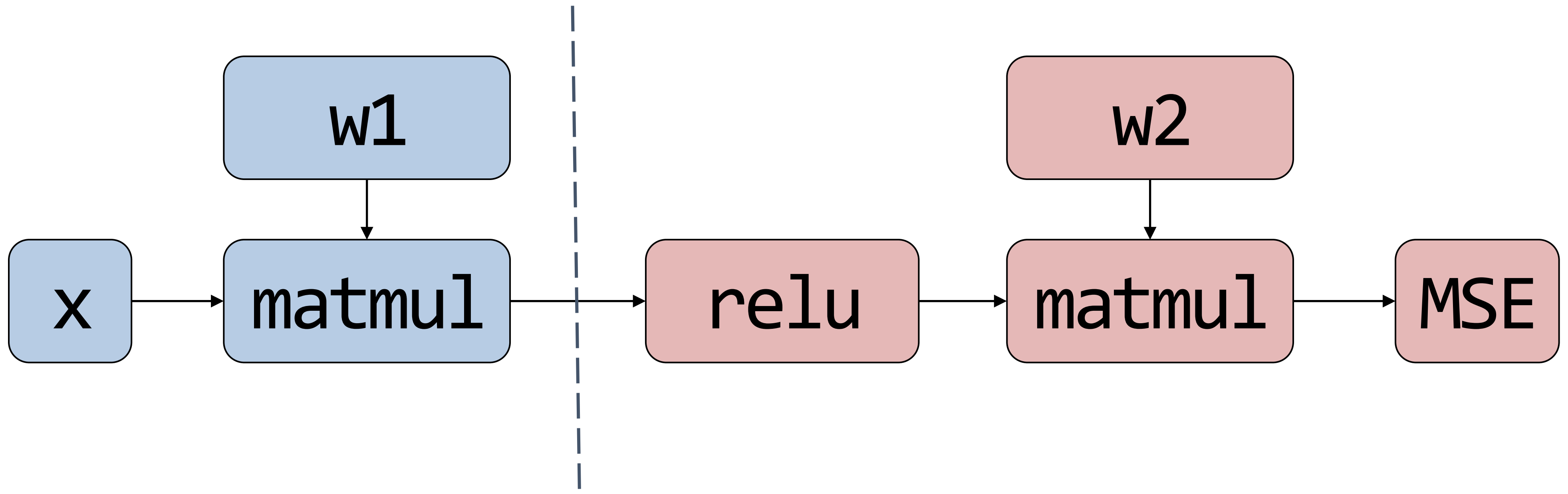
Intra-op parallelism: Devices are busy but requires collective communication



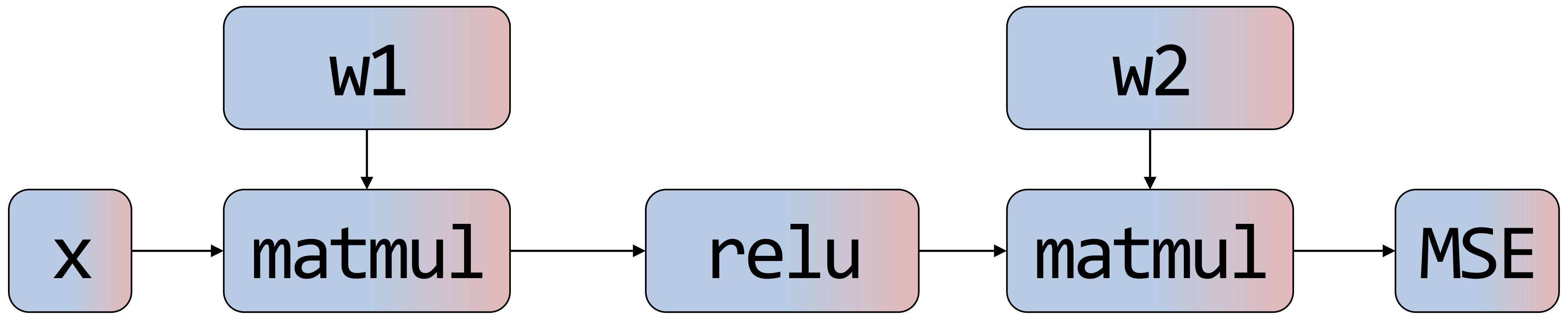
Inter-op and Intra-op Parallelism: Characteristics



Inter-op parallelism



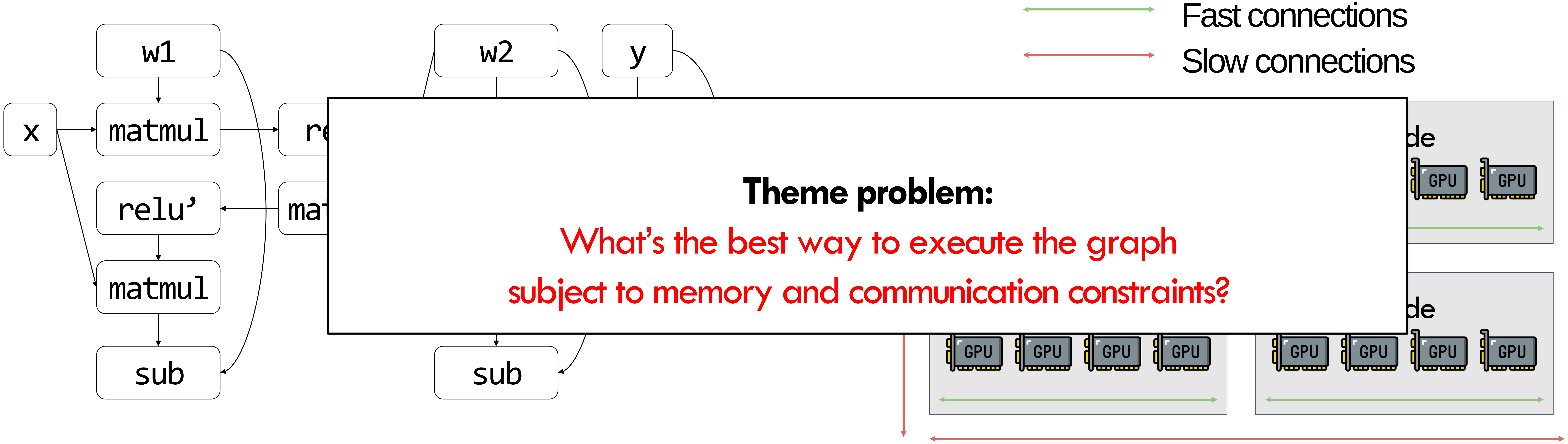
Intra-op parallelism



Trade-off

	Inter-operator Parallelism	**Intra-operator Parallelism**
Communication	Less	More
Device Idle Time	More	Less

ML Parallelization under New View



Theme problem:

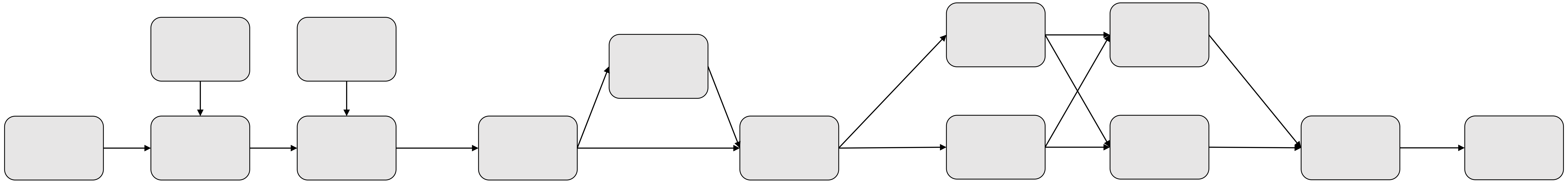
What's the best way to execute the graph subject to memory and communication constraints?

Where We Are

- Deep Learning as Dataflow Graphs
- Auto-differentiation Libraries
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- DL Parallelism
  - **Inter-op parallelism**
  - Intra-op parallelism

Computational Graph (Neural Networks) → Stages

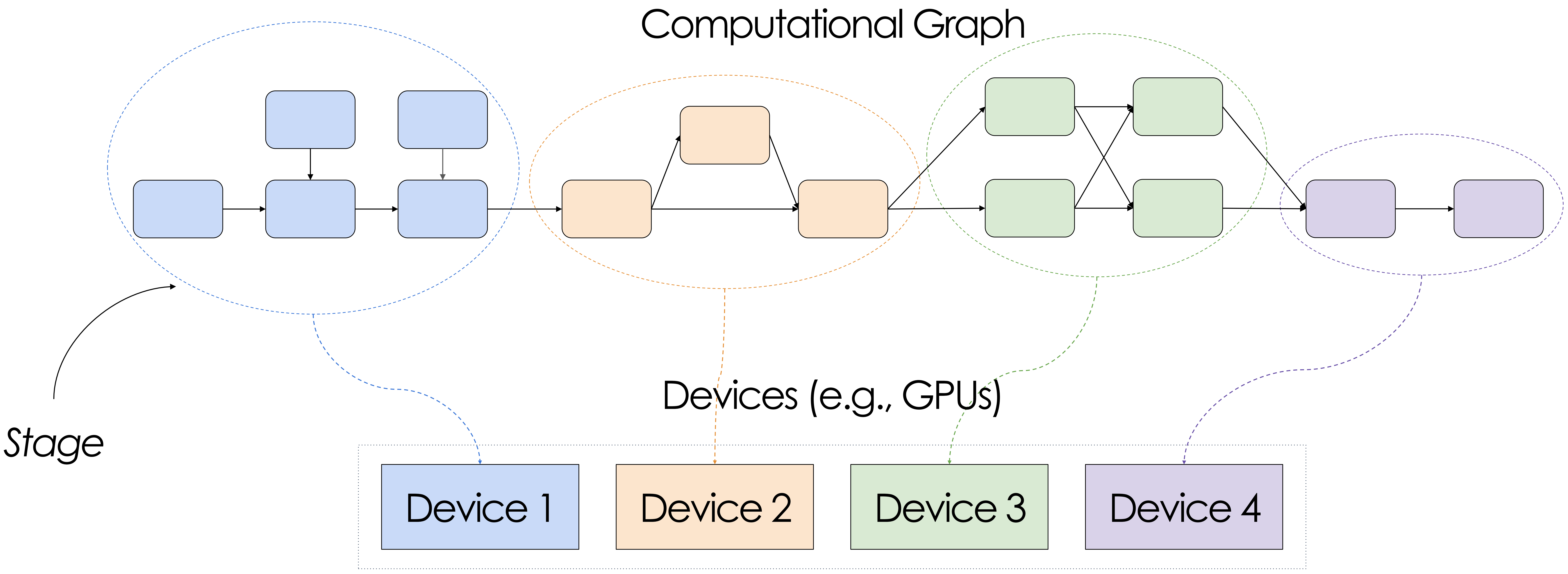
Computational Graph



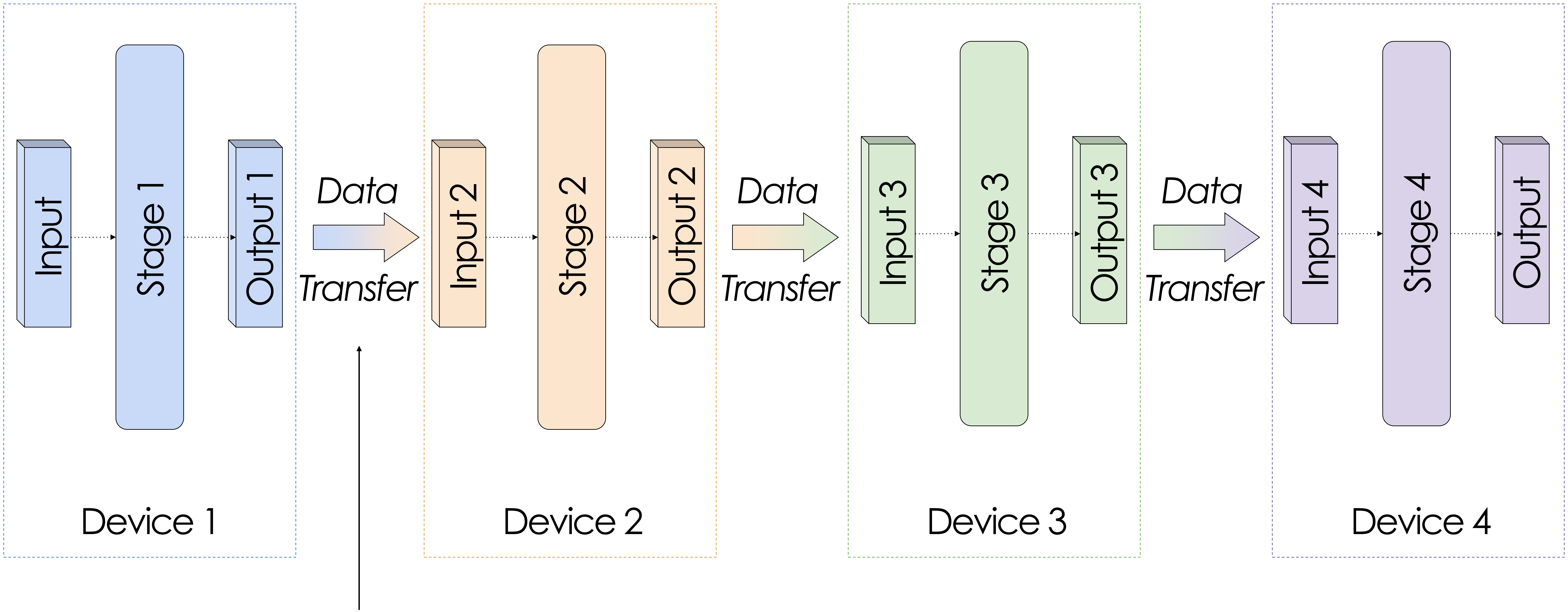
Devices (e.g., GPUs)

Device 1 | Device 2 | Device 3 | Device 4

Computational Graph (Neural Networks) → Stages

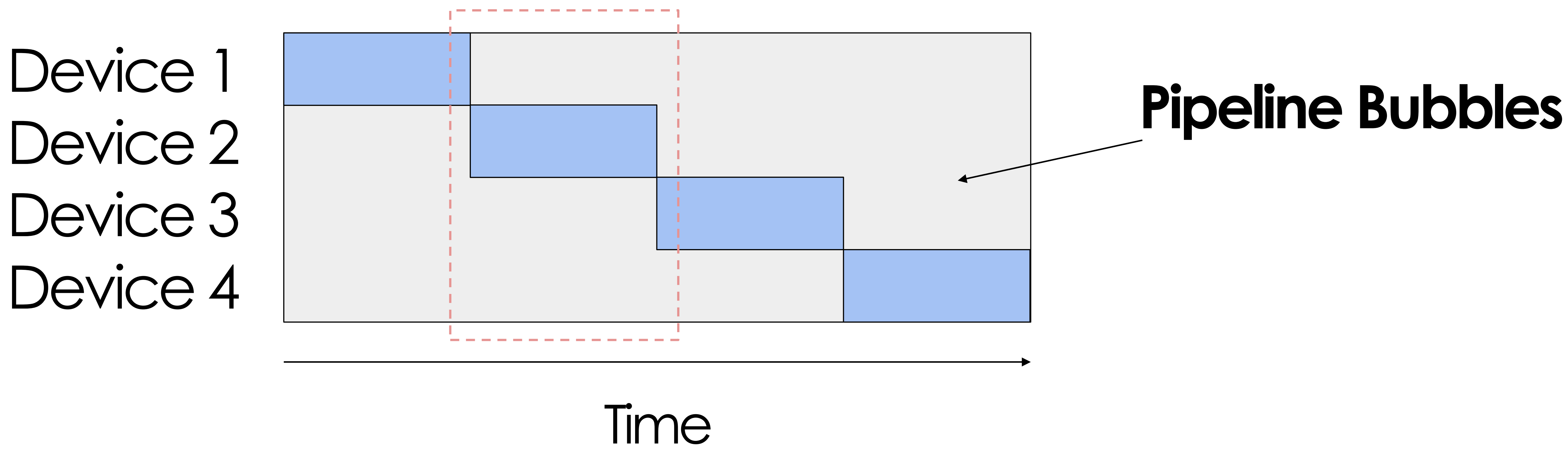


Execution & Data Movement



Note: The time spent on data transfer is typically **small,** since we only communicates stage outputs at stage boundaries between two stages.

Timeline: Visualization of Inter-Operator Parallelism

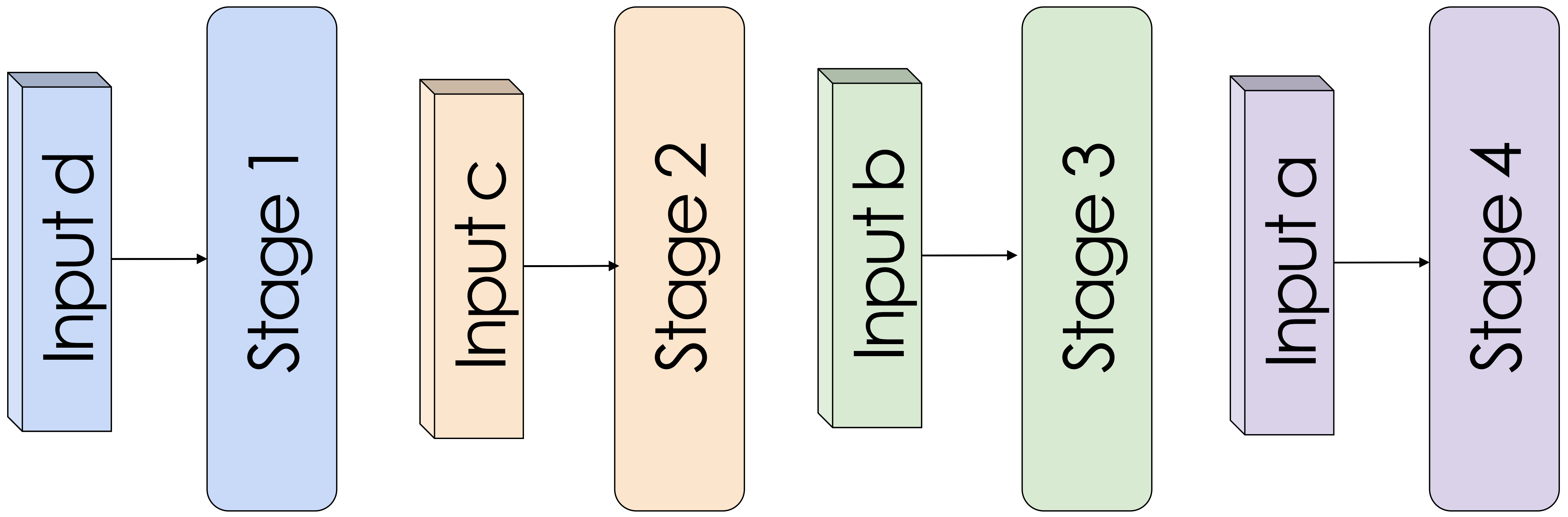


- Gray area ( ) indicates devices being idle (a.k.a. Pipeline bubbles).

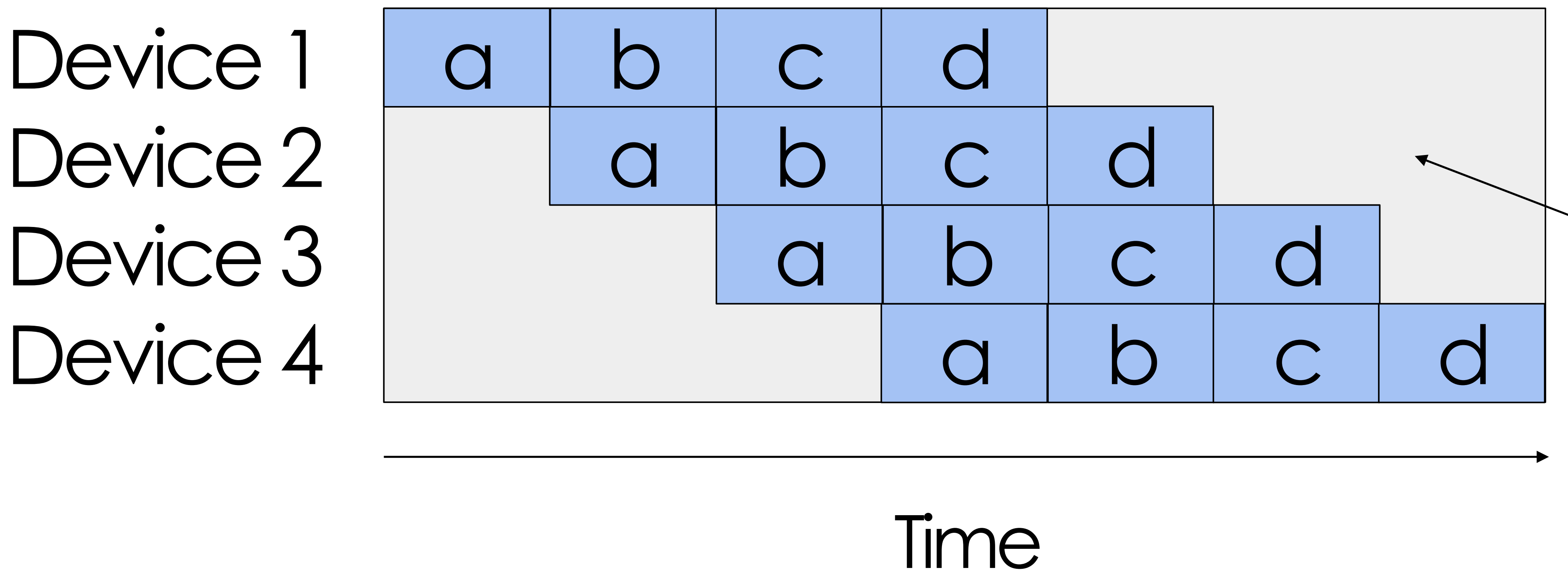
- Only 1 device activated at a time.

- **Pipeline bubble percentage** = bubble_area / total_area
  = (D - 1) / D, assuming D devices.

Reduce Pipeline Bubbles via Pipelining Inputs

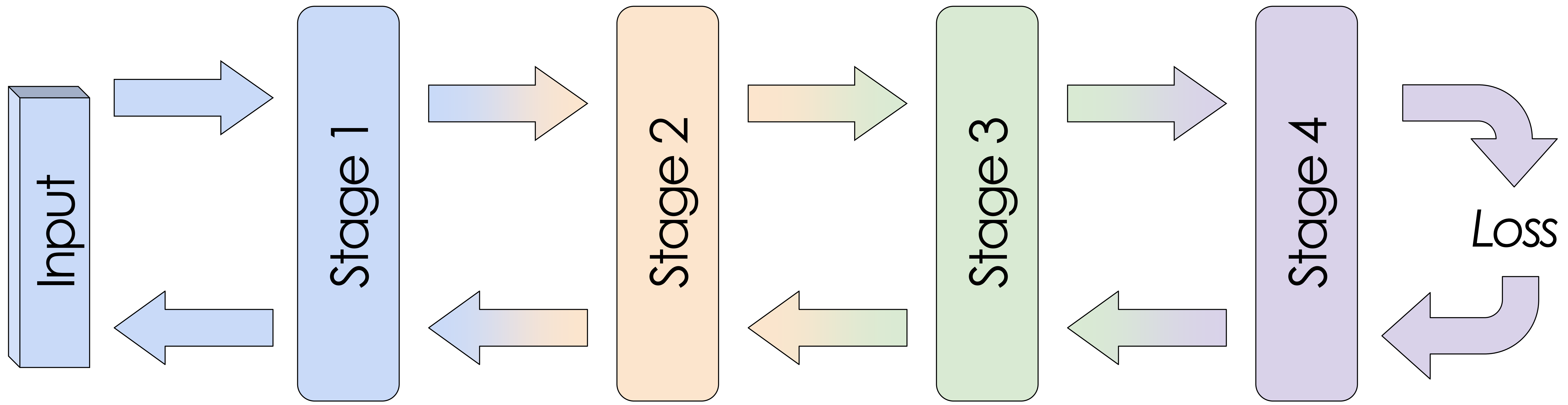


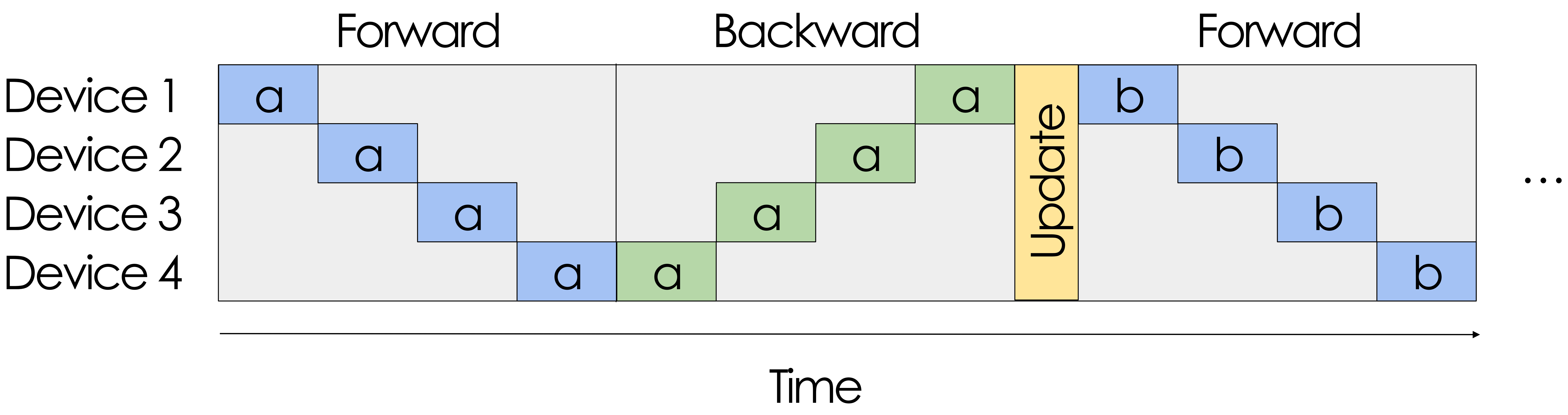
Used in inference.



Pipeline bubbles percentage
= (D - 1) / (D - 1 + N)
with D devices and N inputs.

Training: Forward & Backward Dependency





How to Reduce Pipeline Bubbles for Training?

- Synchronous Pipeline Parallel Algorithms
  - GPipe
  - 1F1B
  - Interleaved 1F1B
  - TeraPipe
  - Chimera
- Asynchronous Pipeline Parallel Algorithms
  - AMPNet
  - Pipedream/Pipedream-2BW

How to Reduce Pipeline Bubbles for Training?

- Synchronous Pipeline Parallel Algorithms
  - **GPipe**
  - **1F1B**
  - Interleaved 1F1B
  - TeraPipe
  - Chimera
- Asynchronous Pipeline Parallel Algorithms
  - AMPNet
  - Pipedream/Pipedream-2BW

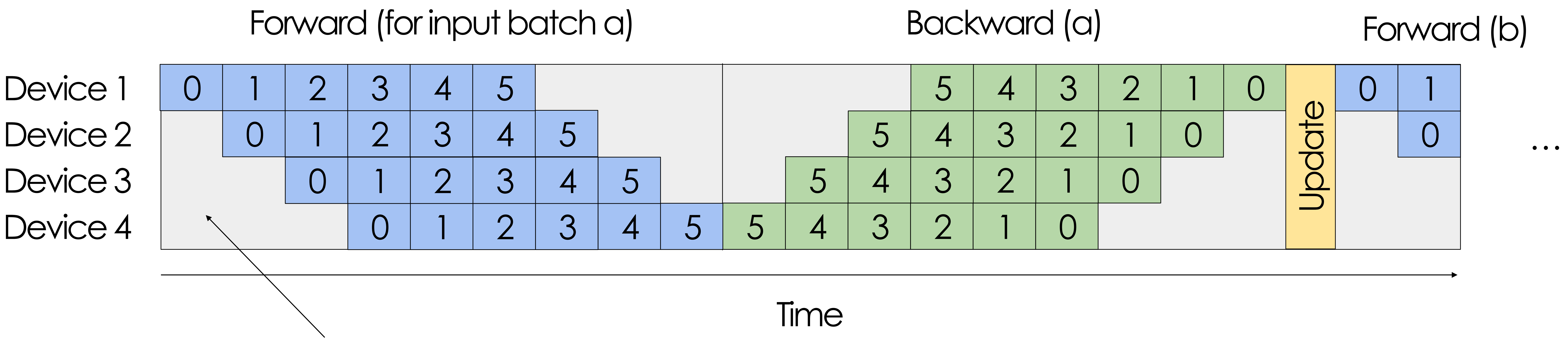
GPipe

Idea: Partition the input batch into multiple "*micro-batches*". Pipeline the micro-batches.

Accumulate the gradients of the micro-batches:

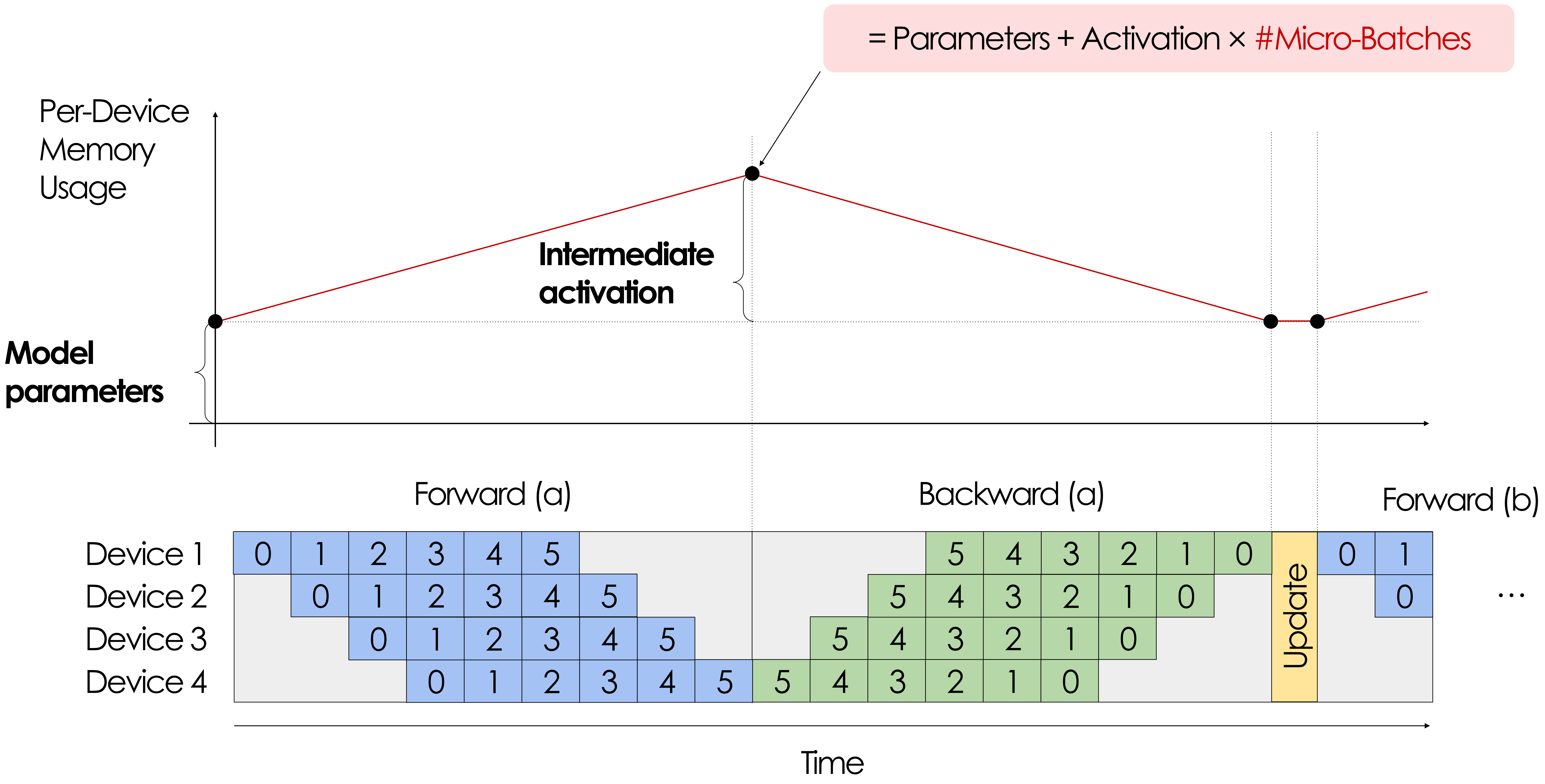
$$\nabla L_\theta(x) = \frac{1}{N}\sum_{i=1}^{N} \nabla L_\theta(x_i)$$

Example: Slice each input batch into 6 micro-batches:

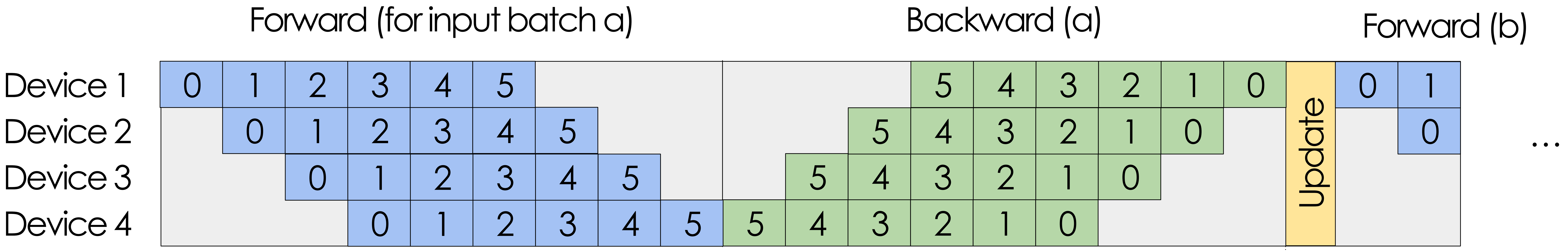


Pipeline bubbles percentage = (D - 1) / (D - 1 + N)
with D devices and N micro-batches.

GPipe: Memory Usage

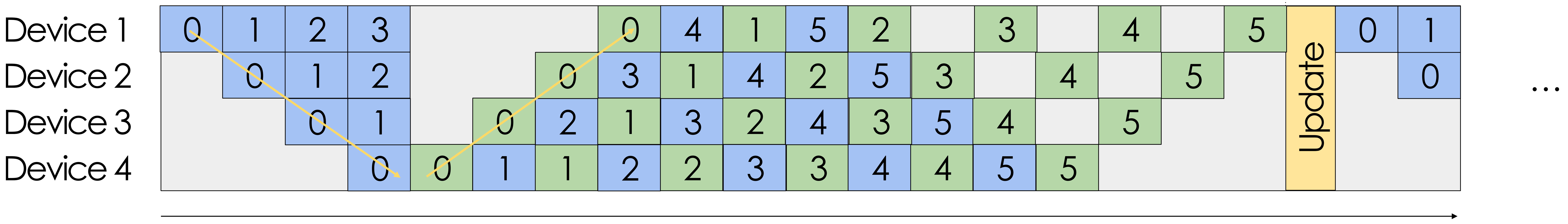


GPipe Schedule:



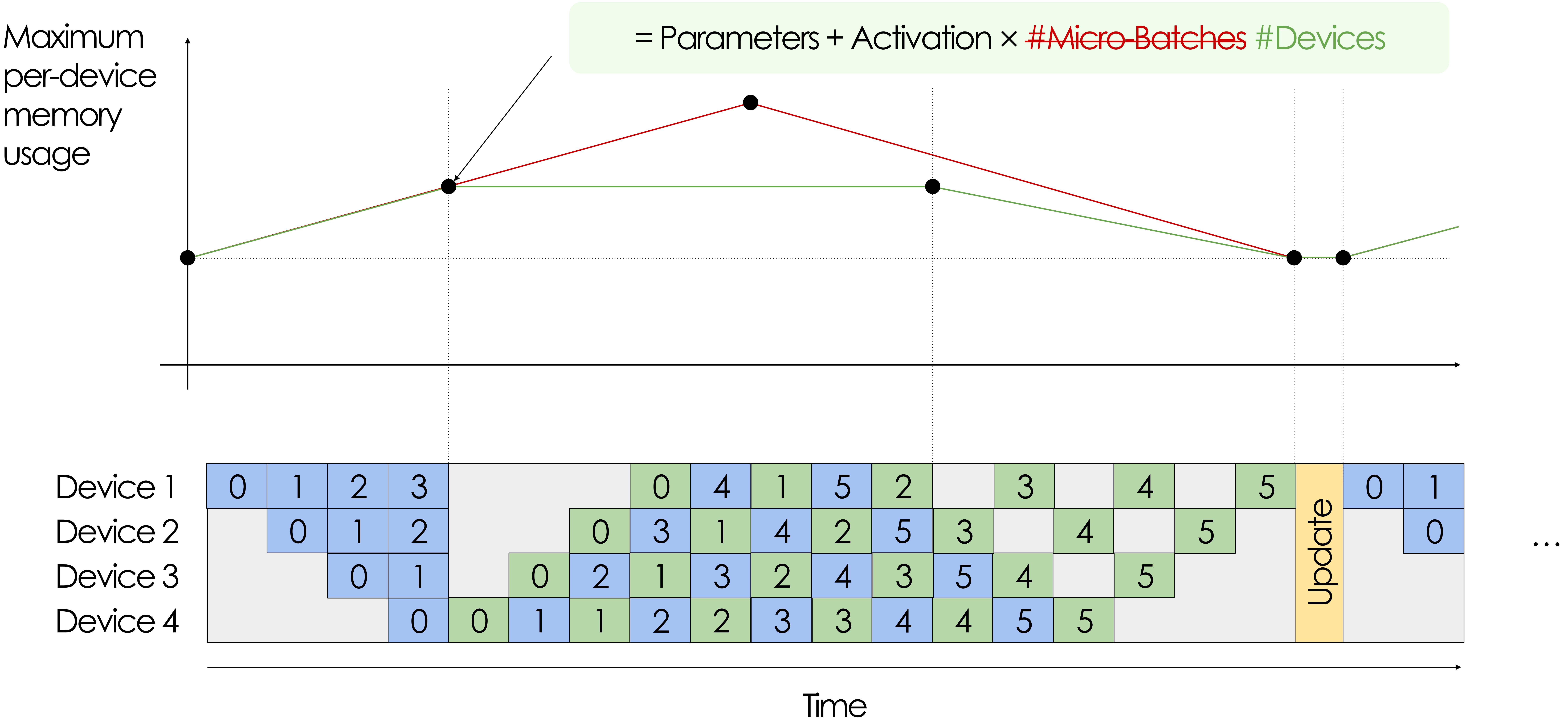
1F1B (1 Forward 1 Backward) Schedule:

Same Latency



Perform backward as early as possible

1F1B Memory Usage

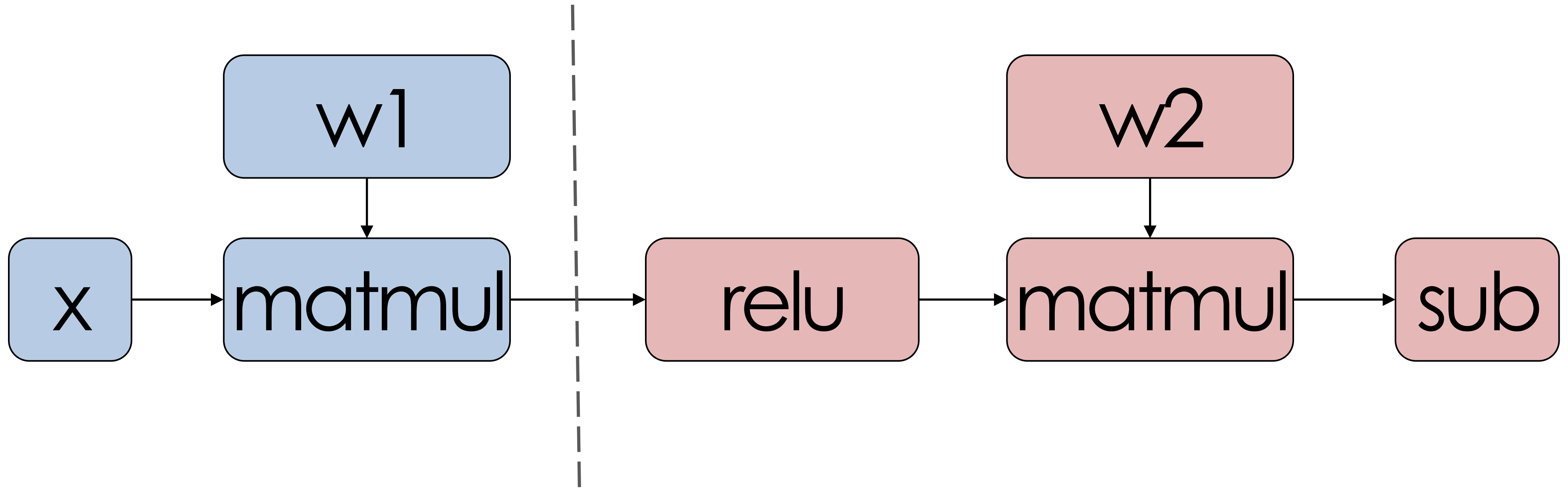


Where We Are

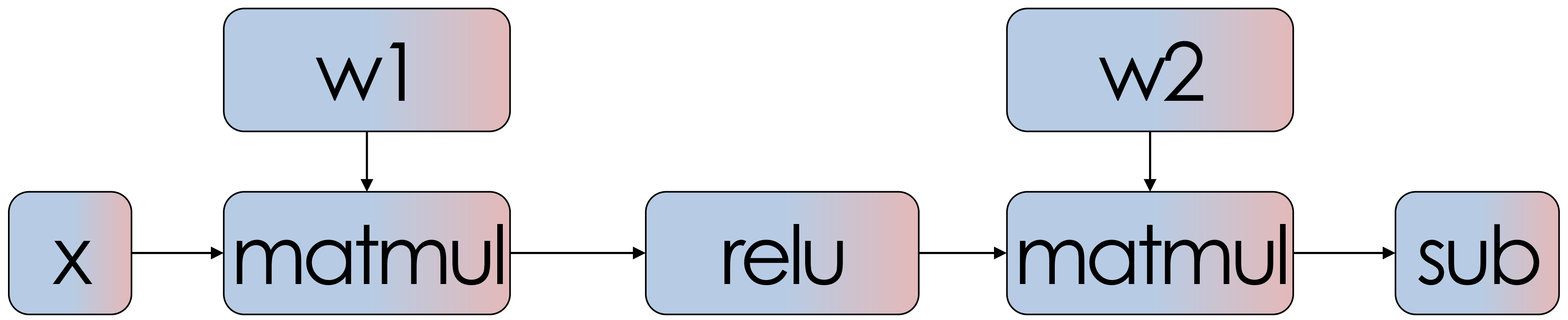
- Deep Learning as Dataflow Graphs
- Auto-differentiation Libraries
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- DL Parallelism
  - Inter-op parallelism
  - **Intra-op parallelism**

Recap: Intra-op and Inter-op

Strategy 1: Inter-operator Parallelism



Strategy 2: Intra-operator Parallelism



This section:

How to parallelize an **operator** ?

How to parallelize a **graph** ?

Parallelize One Operator

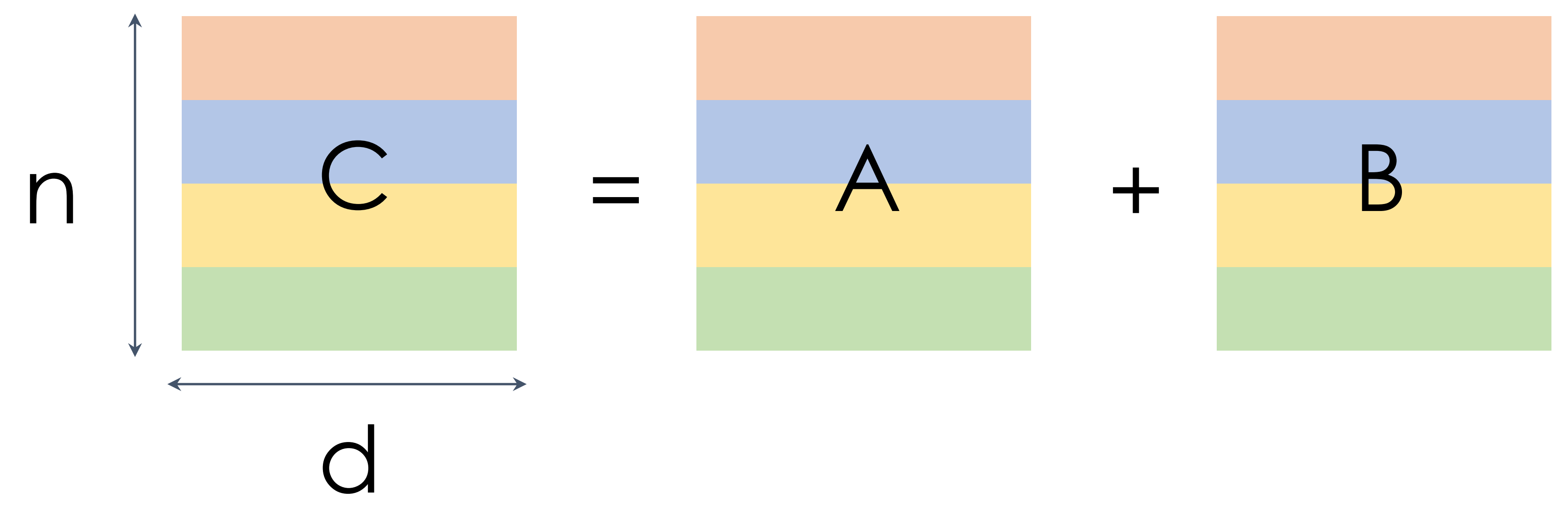
Element-wise operators

```
for n in range(0, N):
  for d in range(0, D):
    C[n,d] = A[n,d] + B[n,d]
```

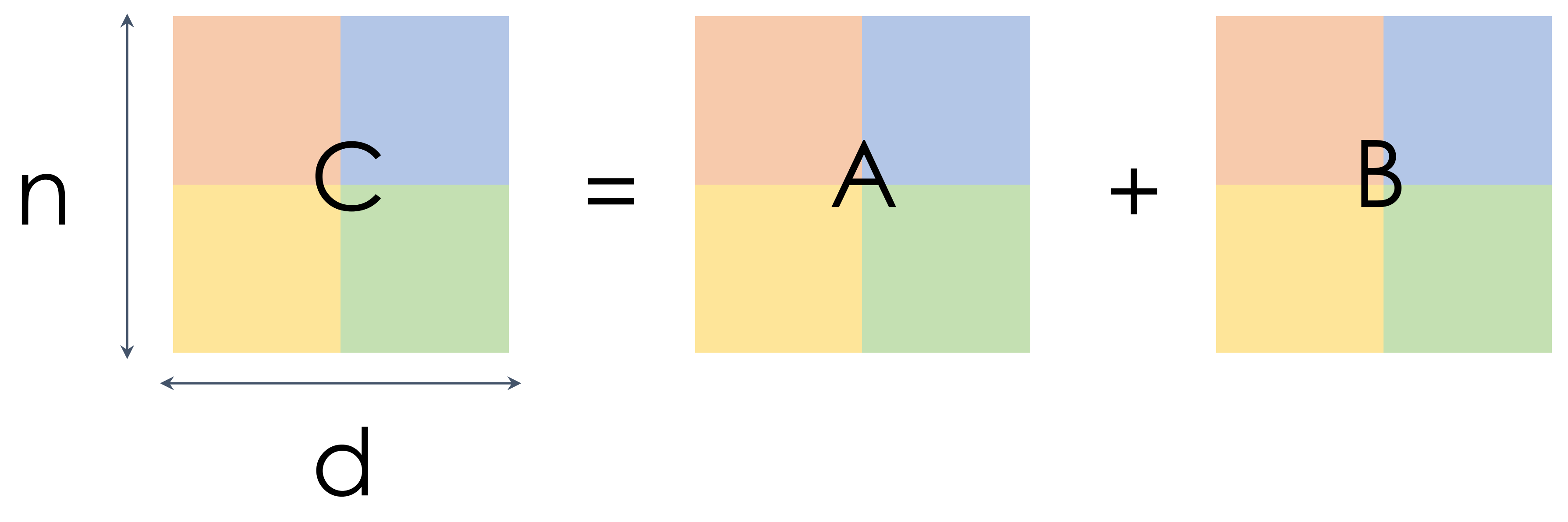
No dependency on the two for-loops.
Can arbitrarily split the for-loops on different devices.

device 1 device 2 device 3 device 4

Parallelize loop n



Parallelize both loop n and loop d



a lot of other variants

…

Parallelize One Operator

Matrix multiplication

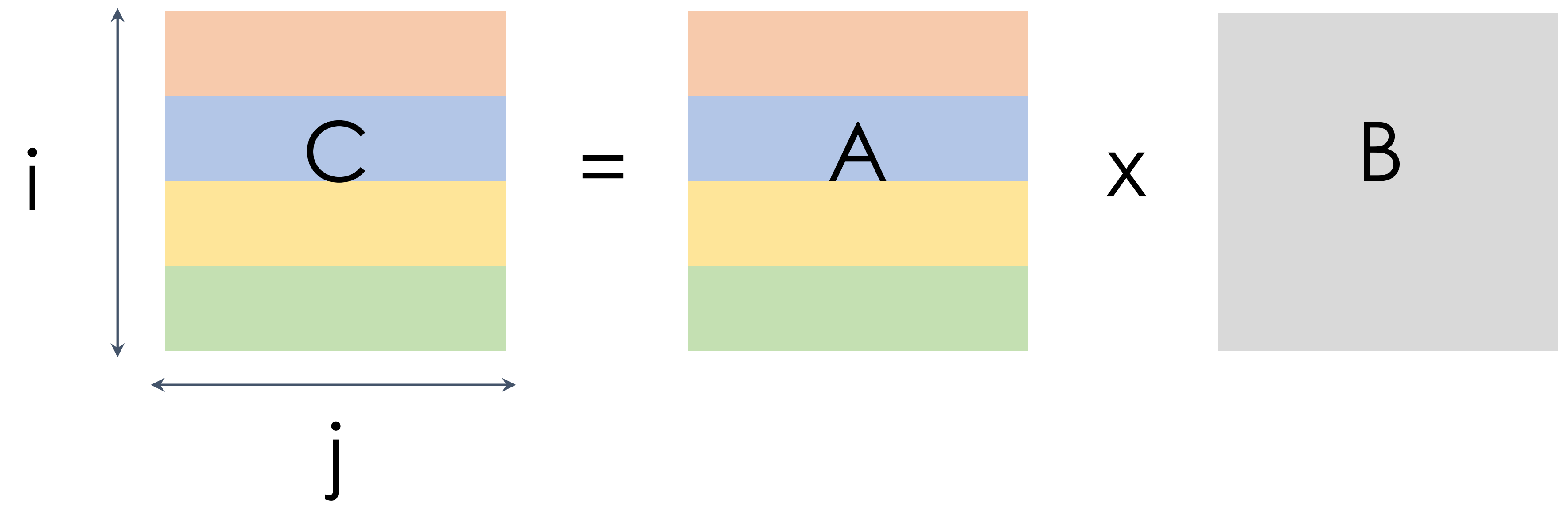
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

No dependency on the two spatial for-loops.
Can arbitrarily split the for-loops on different devices.

Accumulation on this reduction loop.
Have to accumulate partial results if we split this for-loop

device 1 | device 2 | device 3 | device 4 | replicated

Parallelize loop i



$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} \times B$$

Parallelize One Operator

Matrix multiplication

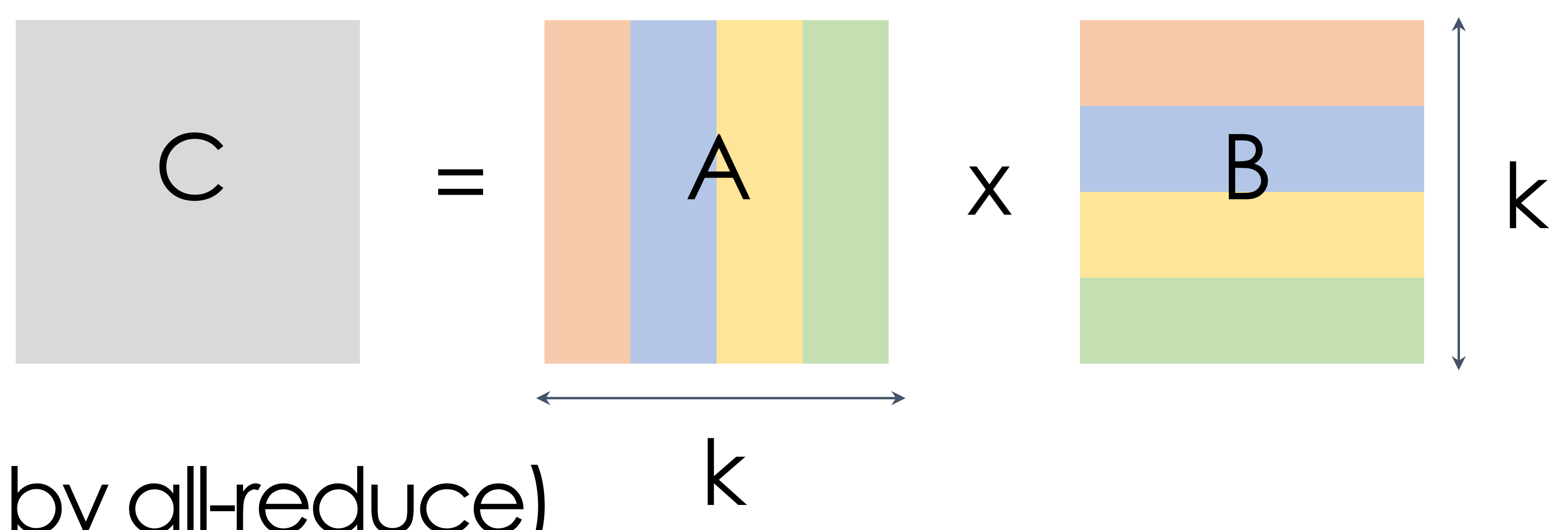
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

No dependency on the two spatial for-loops.
Can arbitrarily split the for-loops on different devices.

Accumulation on this reduction loop.
Have to accumulate partial results if we split this for-loop

device 1 device 2 device 3 device 4 replicated

Parallelize loop k



(got by all-reduce)

$$C = [A_1 \;\; A_2 \;\; A_3 \;\; A_4]\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4$$

Parallelize One Operator

Matrix multiplication

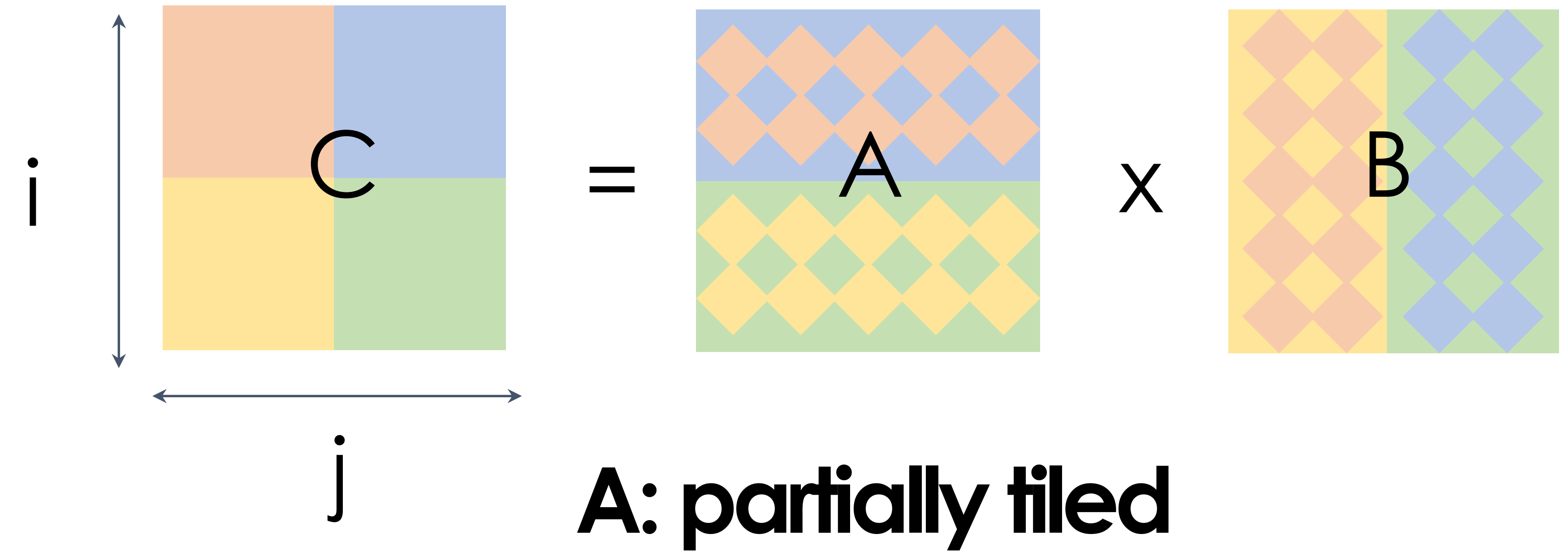
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

No dependency on the two spatial for-loops.
Can arbitrarily split the for-loops on different devices.

Accumulation on this reduction loop.
Have to accumulate partial results if we split this for-loop

device 1 device 2 device 3 device 4

Parallelize loop i and j

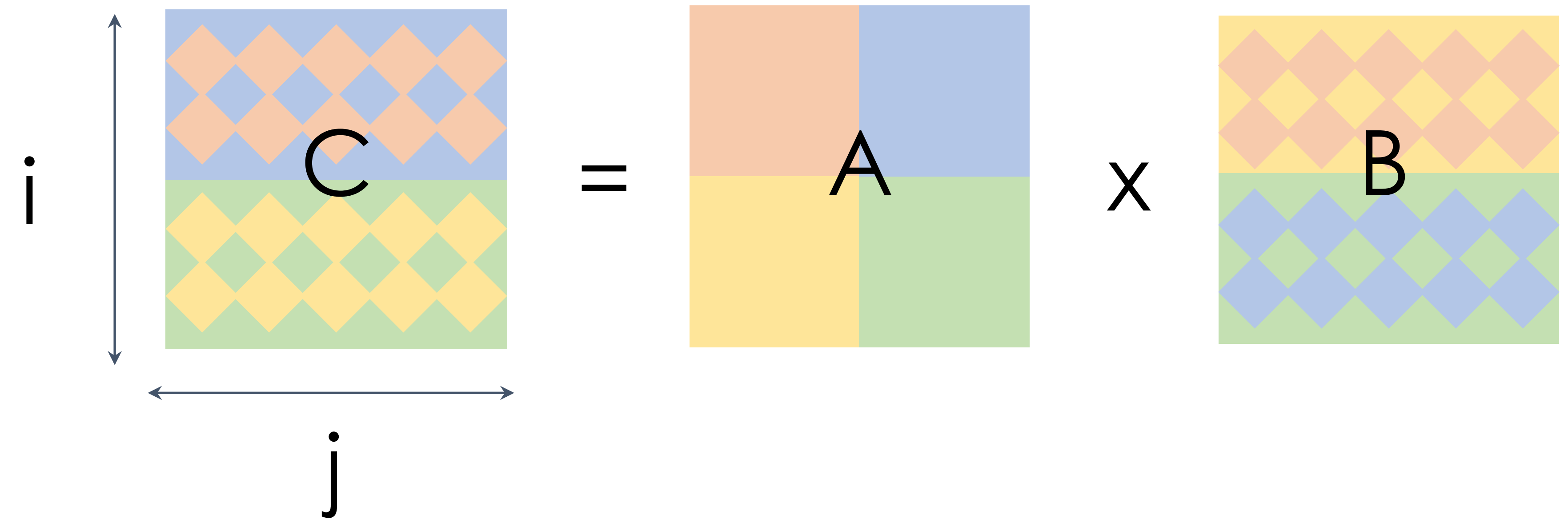


A: partially tiled

Device 1 and 2 hold a replicated tile

Device 3 and 4 hold a replicated tile

Parallelize loop i and k



C: got by all-reduce

a lot of
other variants
…

Parallelize One Operator

2D Convolution

Simple spatial loops. Can be arbitrarily split.

Stencil computation loops. Splitting these requires careful boundary handling.

Reduction loop. Need to accumulate partial results.

Reduction loops. But usually too small (<= 5) for parallelization.

```
for n in range(0, N):
 for co in range(0, CO):
  for h in range(0, H):
   for w in range(0, W):
    for ci in range(0, CI):
     for kh in range(0, KH):
      for kw in range(0, KW):
       C[n,co,h,w] += A[n,co,h+kh,w+kw] x B[kh,kw,co,ci]
```

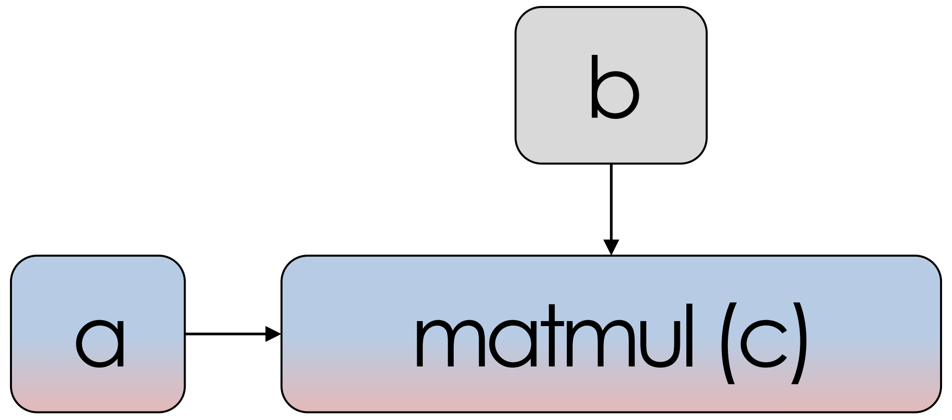
Simple case: Parallelize loop n, co, ci, then the parallelization strategies are almost the same as matmul's.

Complicated case: Parallelize loop h and w

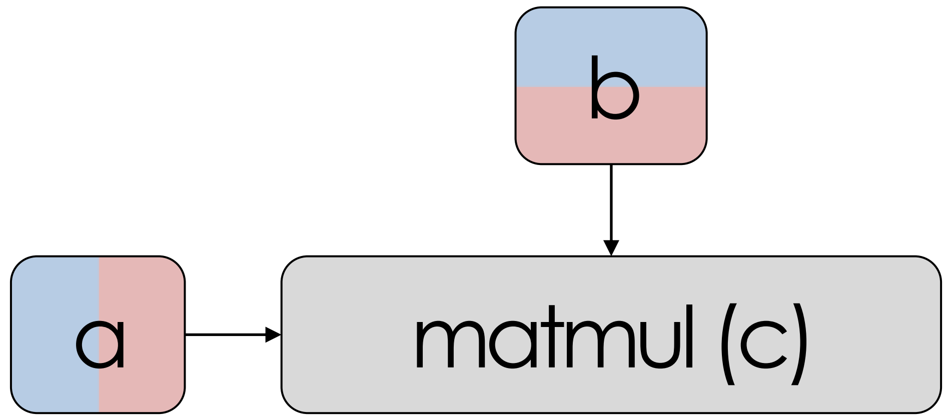
Data Parallelism as A Case of Intra-op Parallelism

Replicated Row-partitioned Column-partitioned

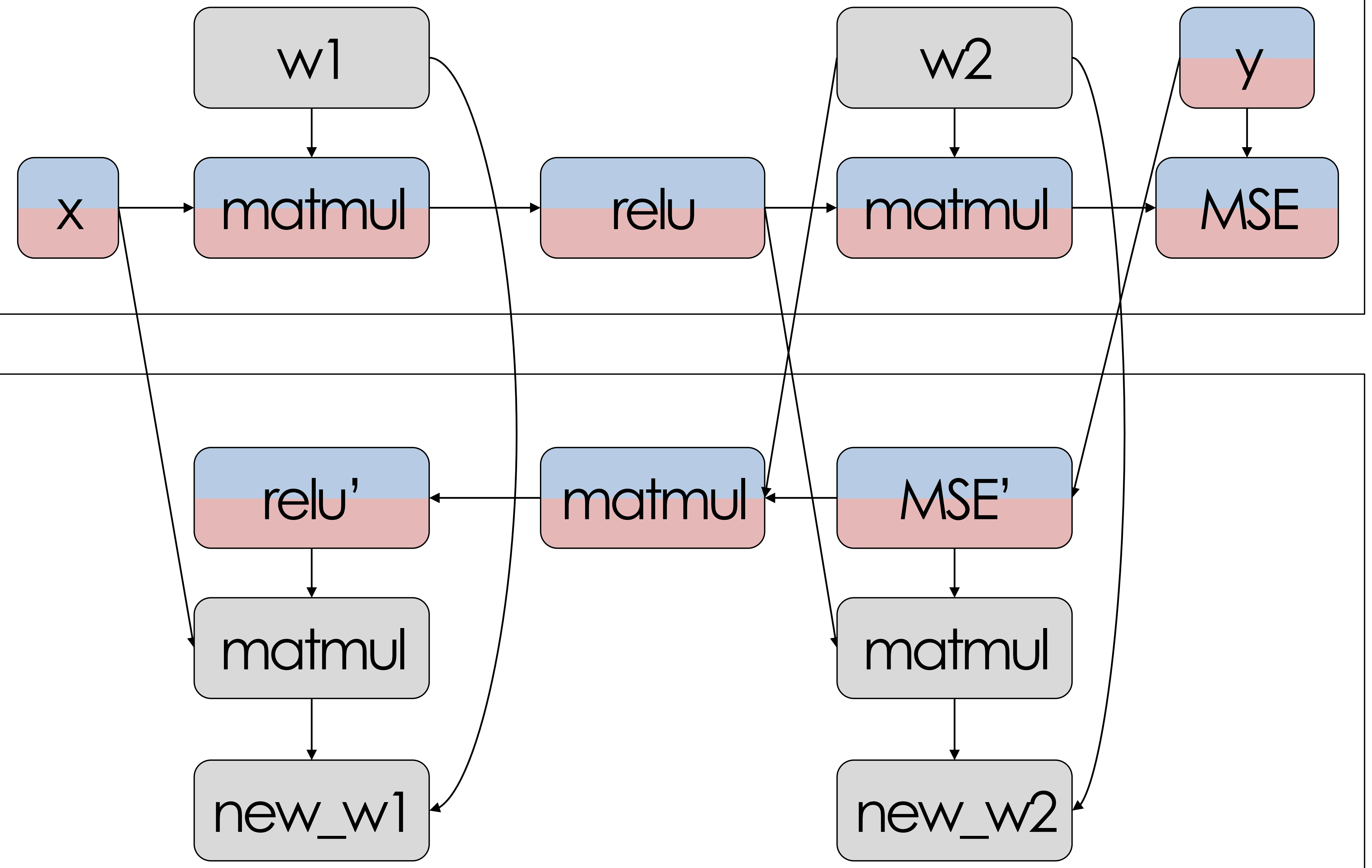
Matmul Parallelization Type 1
communication cost = 0



Matmul Parallelization Type 2
communication cost = all-reduce(c)



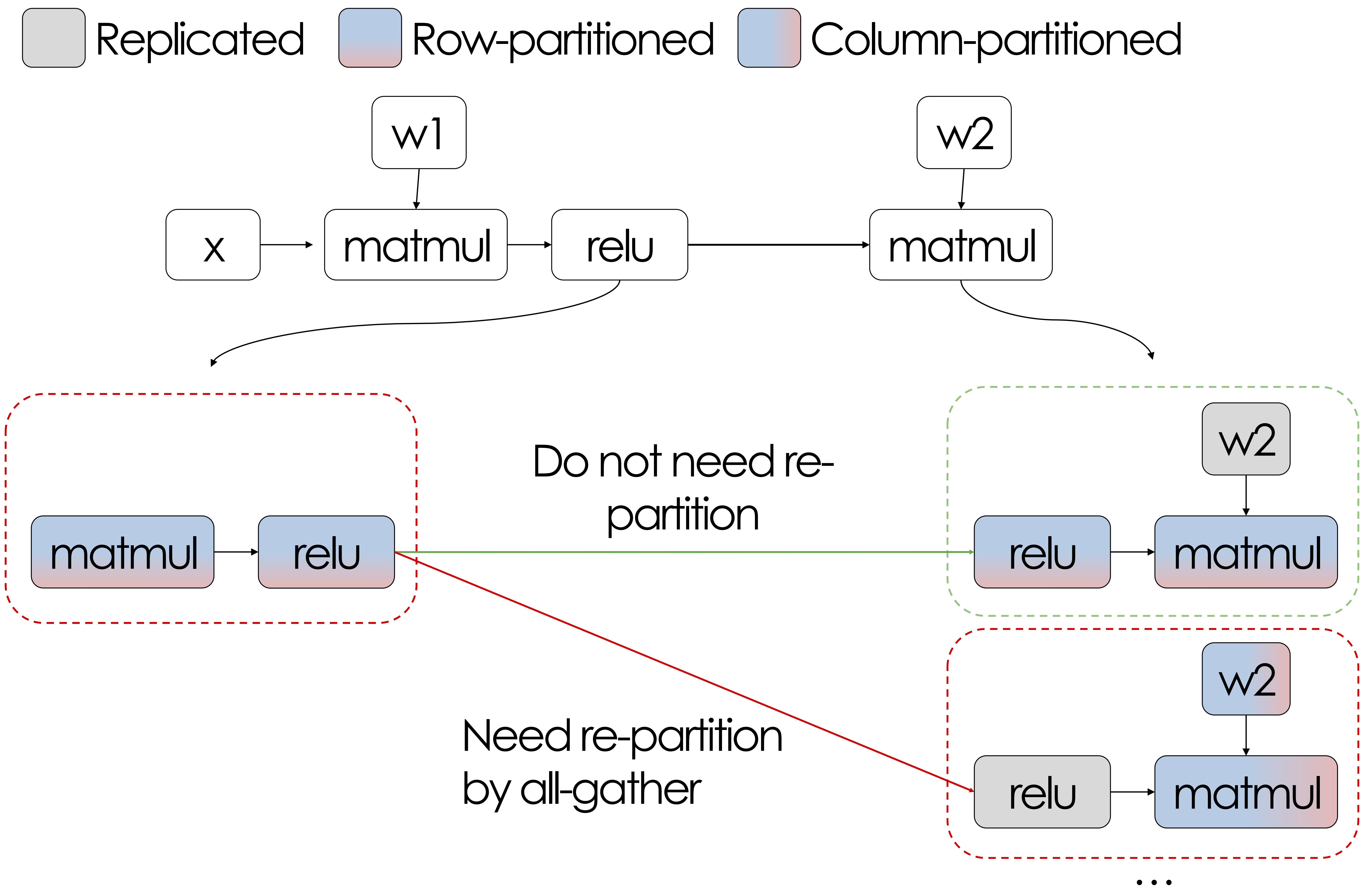
Forward Pass
Two "Type 1" matmuls: no communication



Backward Pass
One "Type 1" matmul: no communication
Two "Type 2" matmuls: require all-reduce

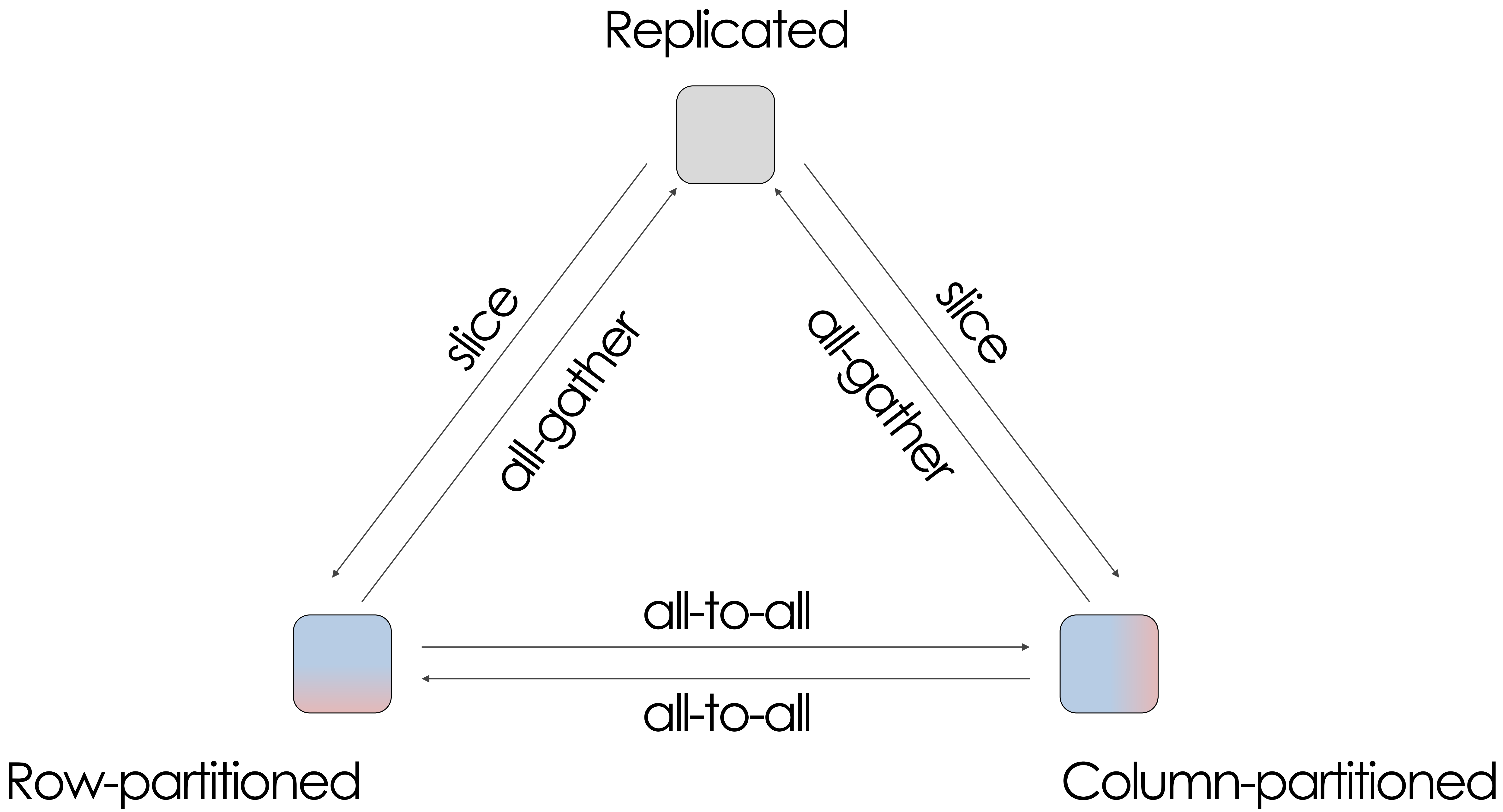
Re-partition Communication Cost

Different operators' parallelization strategies require different partition format of the same tensor



Re-partition Communication Cost

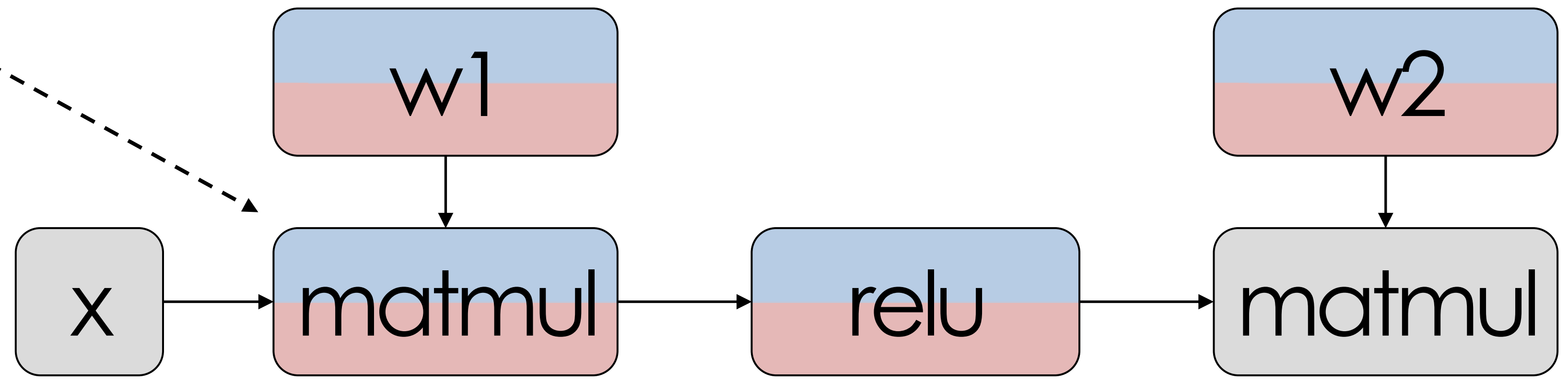
Different operators' parallelization strategies require different partition format of the same tensor



Parallelize All Operators in a Graph

Problem

Pick a parallel strategy of each operator



Minimize Node costs (computation + communication) + Edge costs (re-partition communication)

Solution

Manual design
Randomized search
Dynamic programming
Integer linear programming

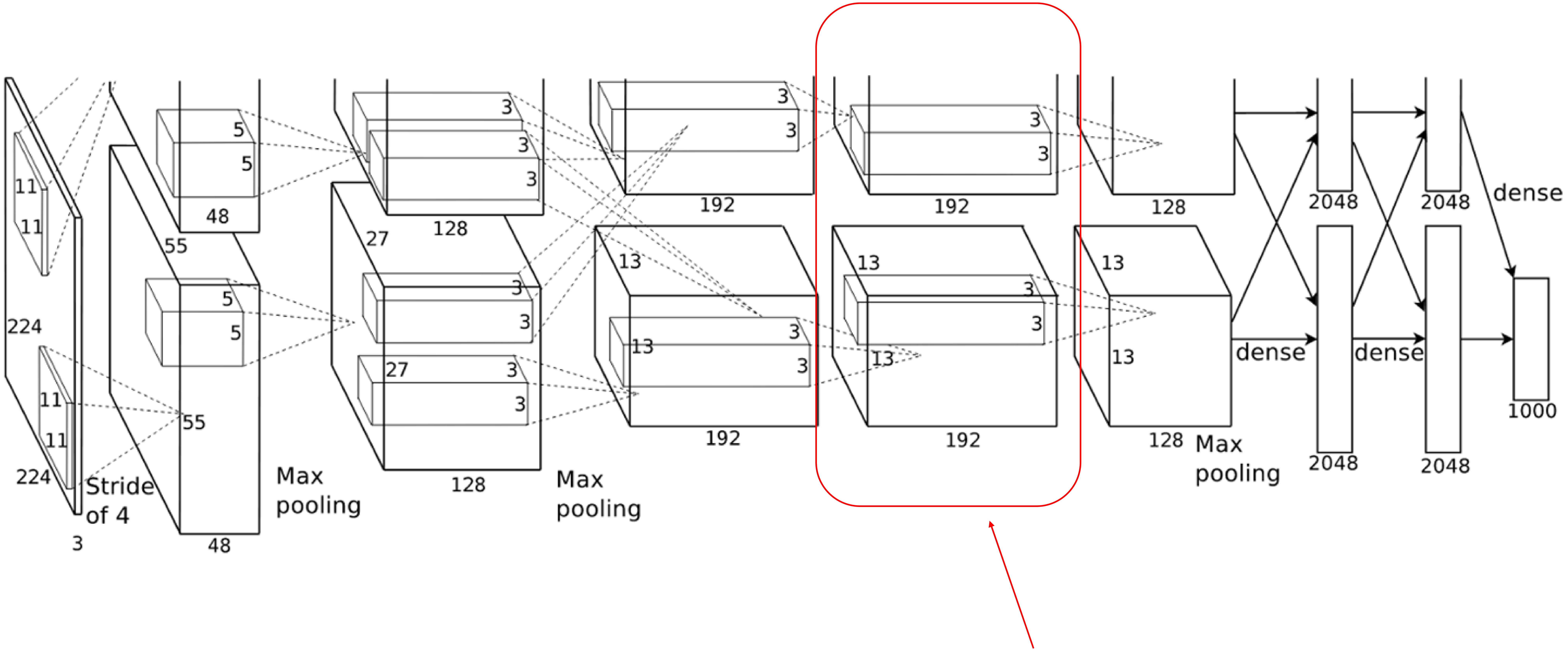
Important Projects

- Model-specific Intra-op Parallel Strategies
  - **AlexNet**
  - **Megatron-LM**
  - GShard MoE

- Systems for Intra-op Parallelism
  - ZeRO
  - Mesh-Tensorflow
  - GSPMD
  - Tofu
  - FlexFlow

AlexNet

Result: increase top-1 accuracy by 1.7%



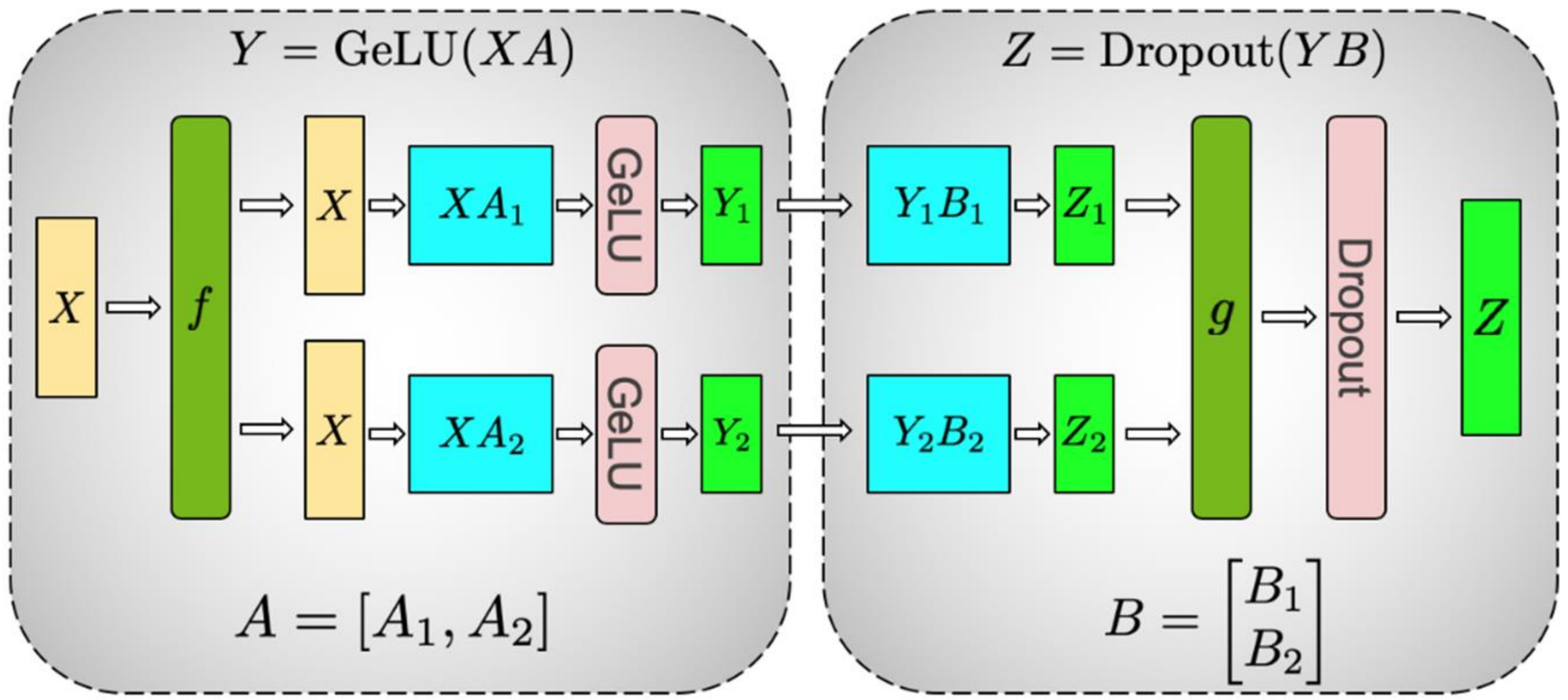
Assign a group convolution layer to 2 GPUs

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional

Megaton-LM

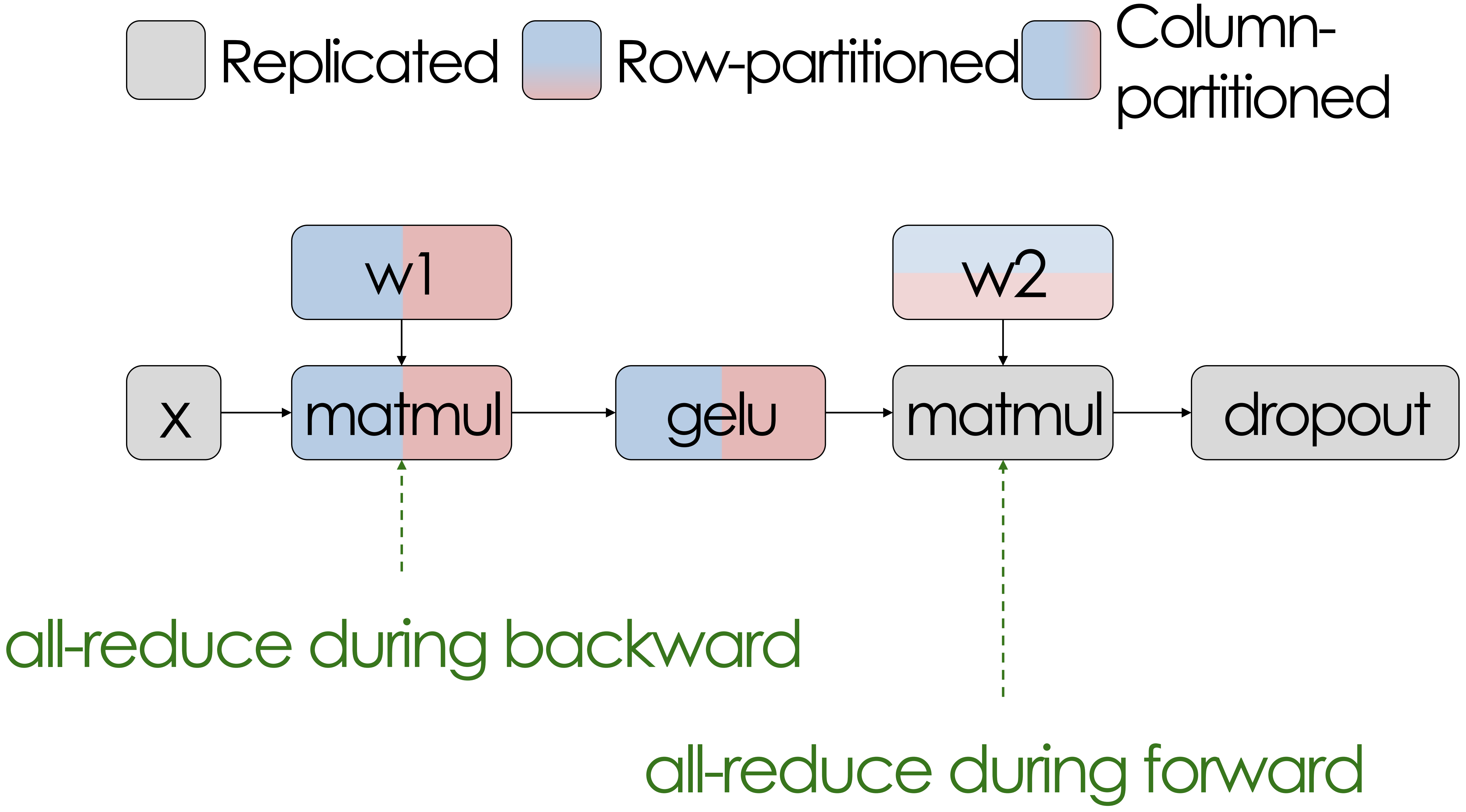
Result: a large language model with 8.3B parameters that outperforms SOTA results

Figure 3 from the paper:
How to partition the MLP in the transformer.

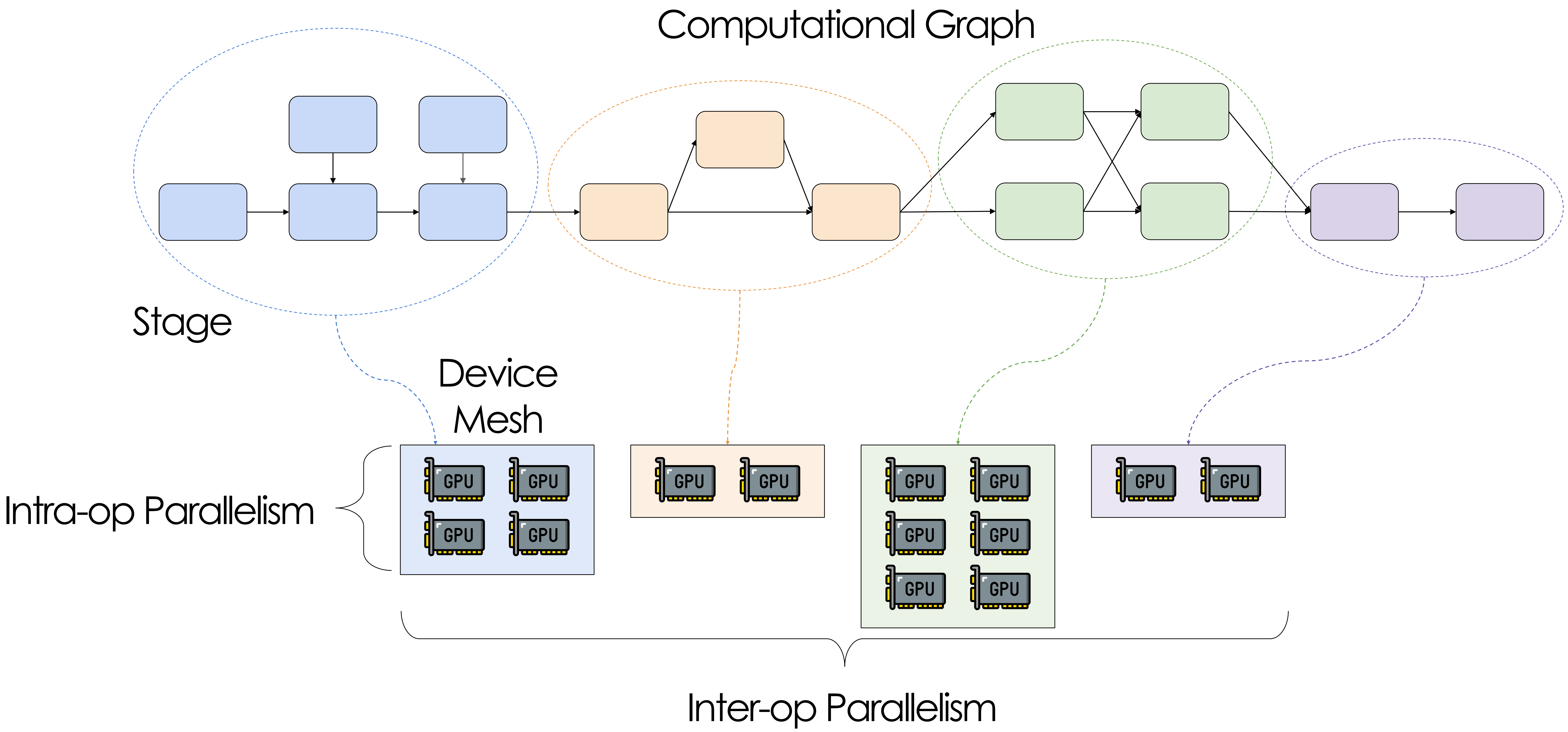


(a) MLP

Illustrated with the notations in this tutorial



Combine Intra-op Parallelism and Inter-op Parallelism



Intra-operator Parallelism Summary

- We can parallelize a single operator by exploiting its internal parallelism
- To do this for a whole computational graph, we need to choose strategies for all nodes in the graph to minimize the communication cost
- Intra-op and inter-op can be combined