Where We Are

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

Logistics

- Exam date and location:
  - **Friday, March 22, 8 - 11 am, PT,** PCYNH 122
  - **Exam Review: next Tuesday**
- If you have scheduling conflicts
  - Reach out to instructor team asap to coordinate
- In-person vs. online
  - In-person is easier
  - In-person is fairer
  - We were advised against online by senior faculty

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

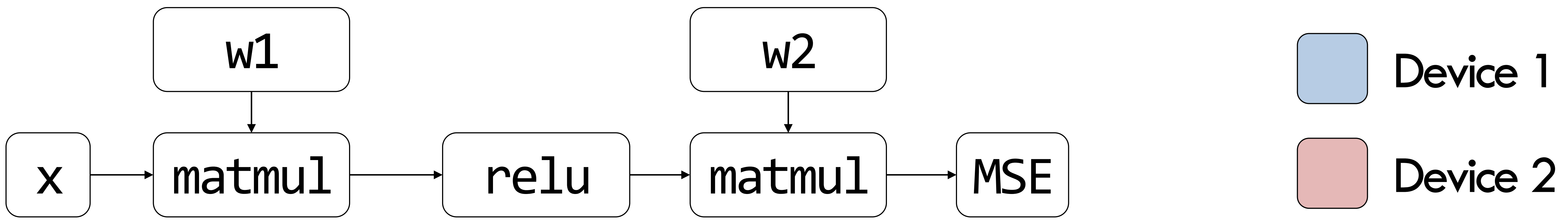
Model family: neural nets

Model: CNNs/transformers/GNNs

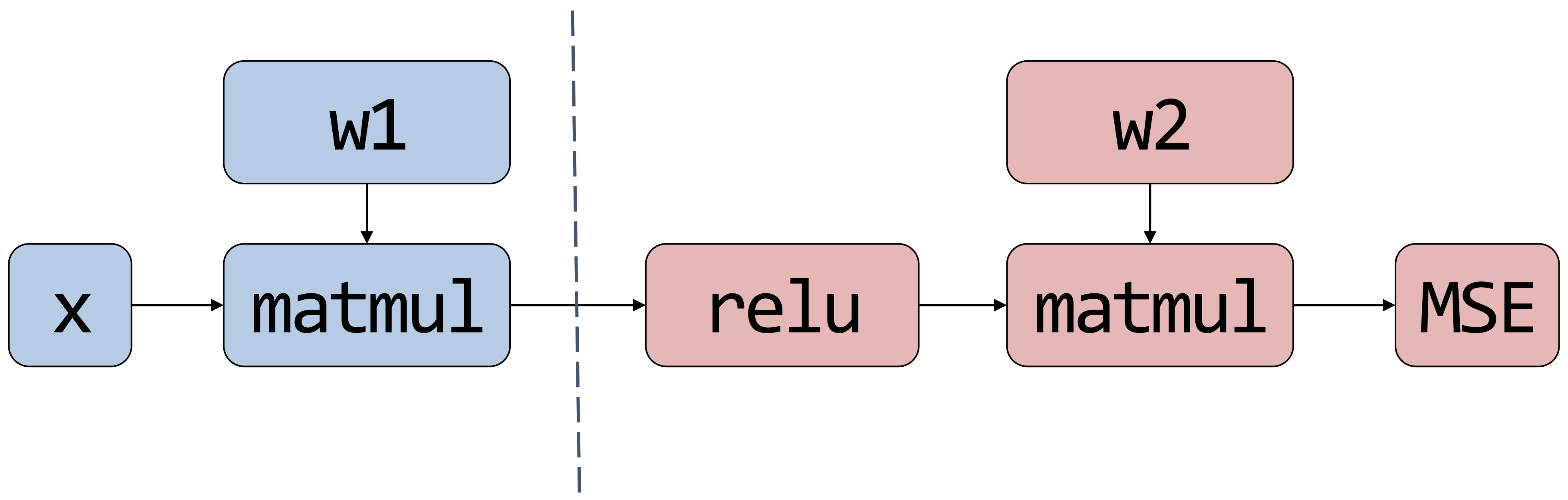
LLMs: transformer decoders

Today: ML Parallelism, and transformers

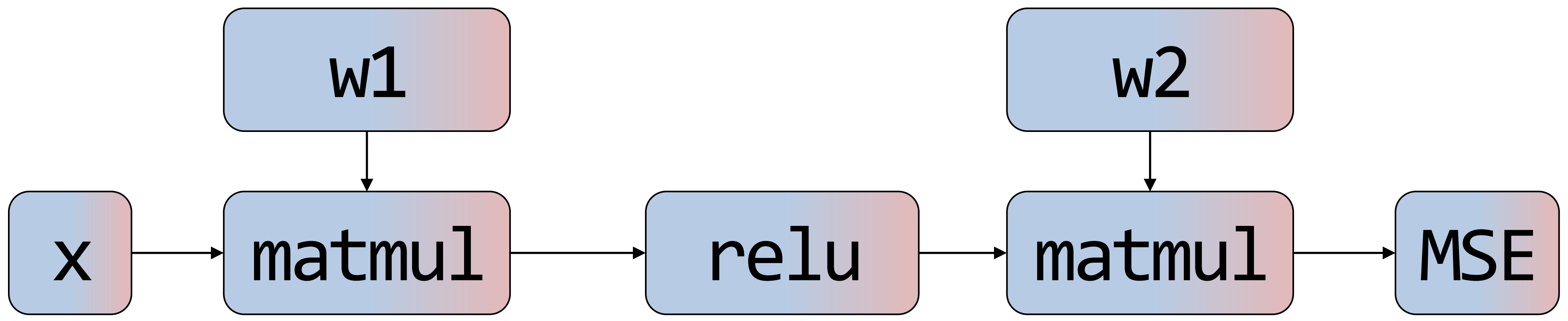
Inter-op and Intra-op Parallelism: Characteristics



Inter-op parallelism



Intra-op parallelism



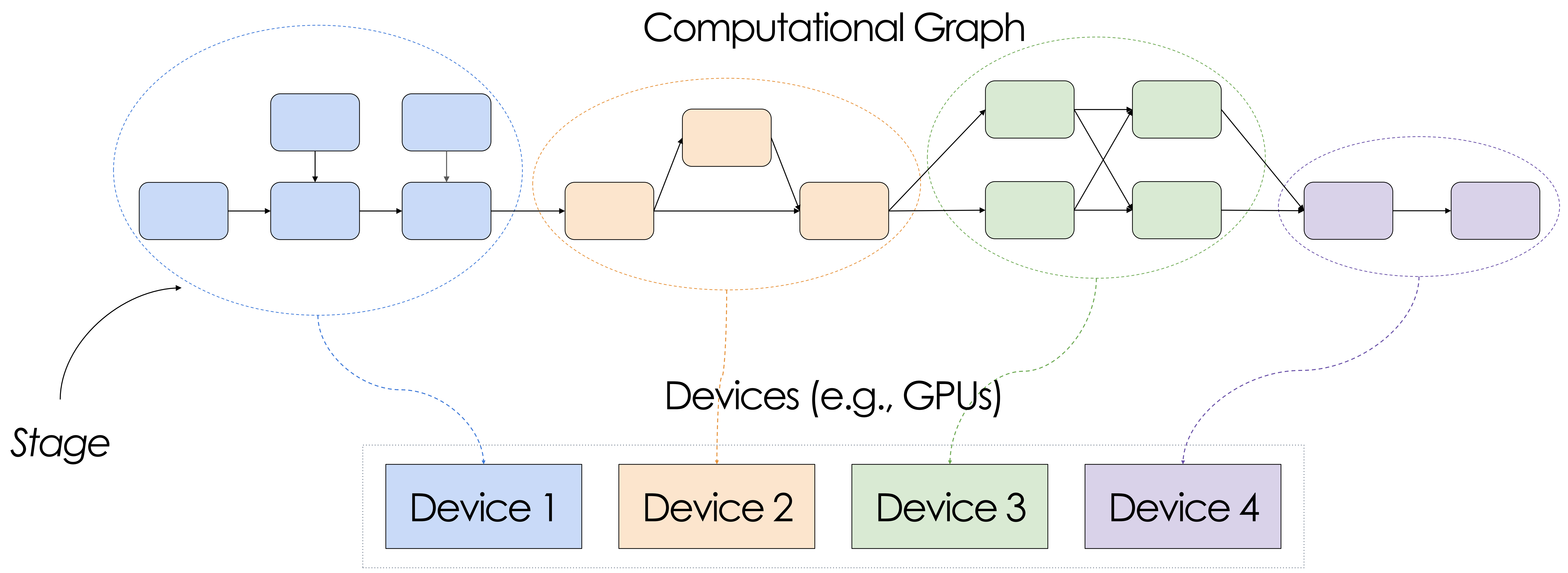
Trade-off

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

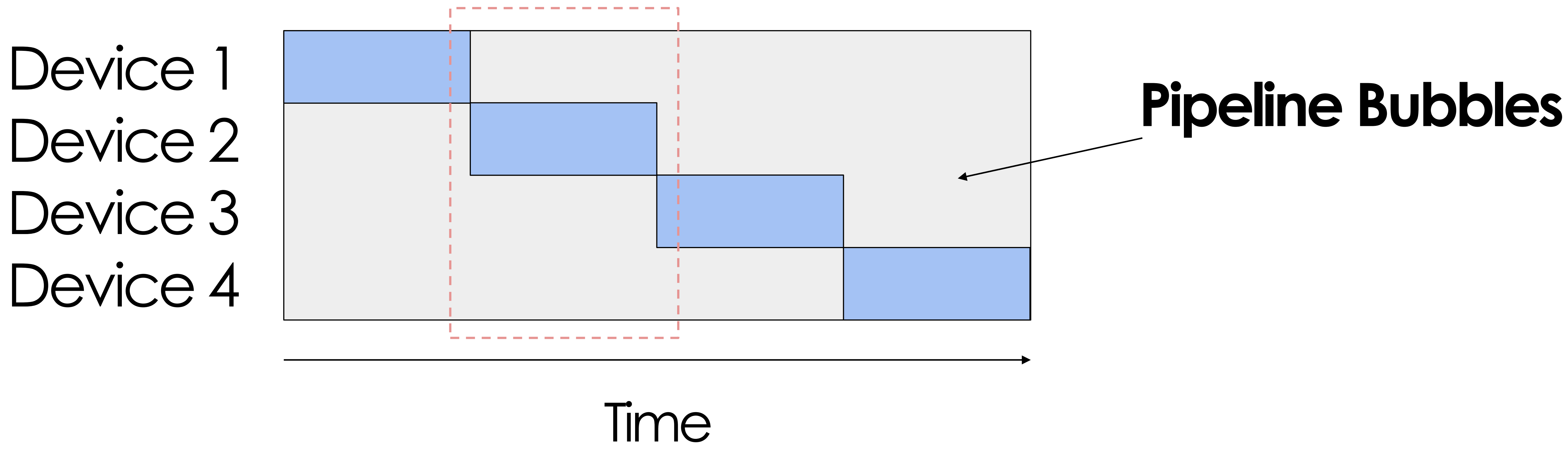
Where We Are

- 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



Timeline: Visualization of Inter-Operator Parallelism



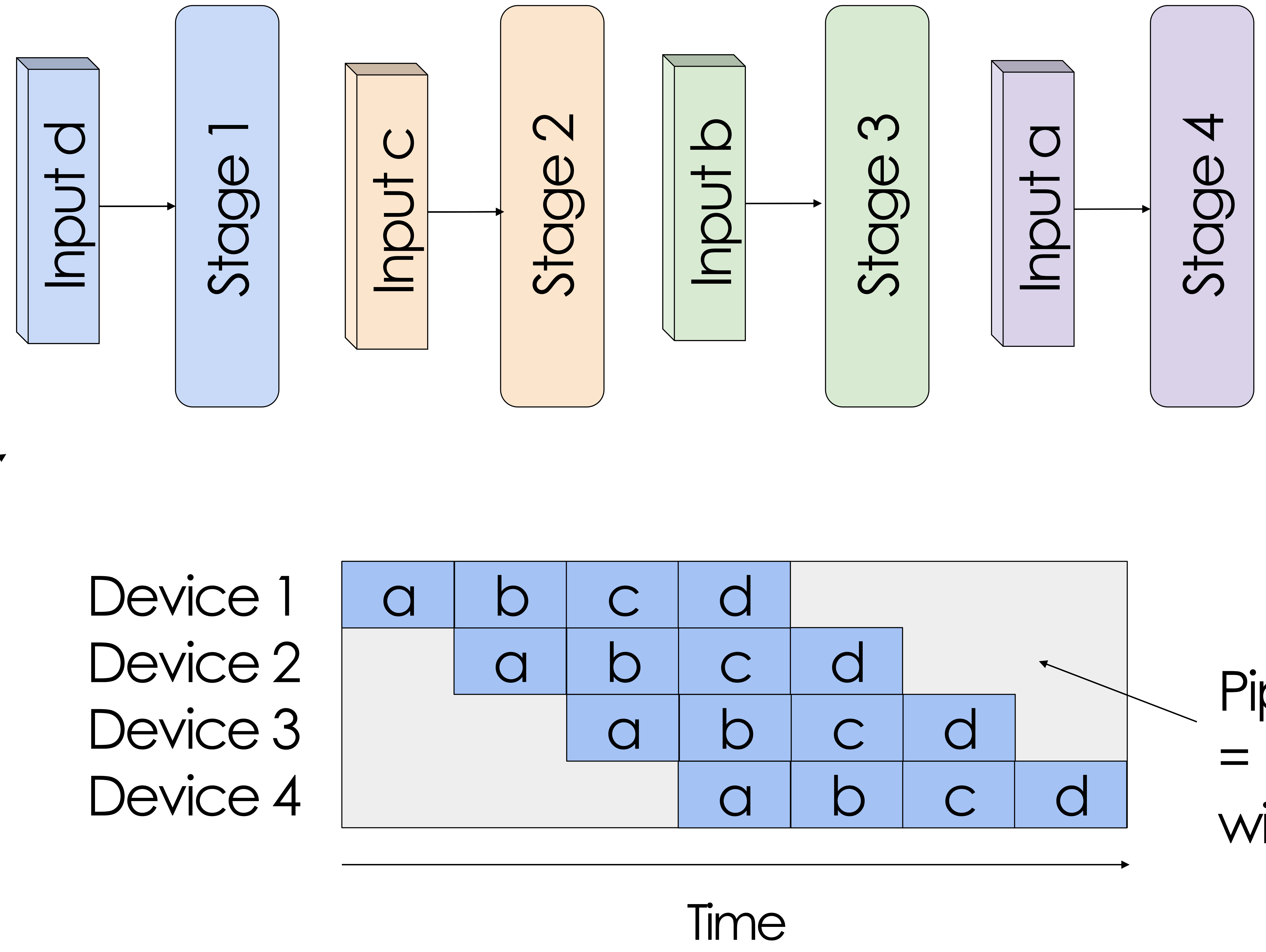
- Gray area ( ) indicates devices being idle (a.k.a. Pipeline bubbles).

- Only 1 device activated at a time.

- **Pipeline bubble percentage** = bubble_area / total_area

  = (D - 1) / D, assuming D devices.

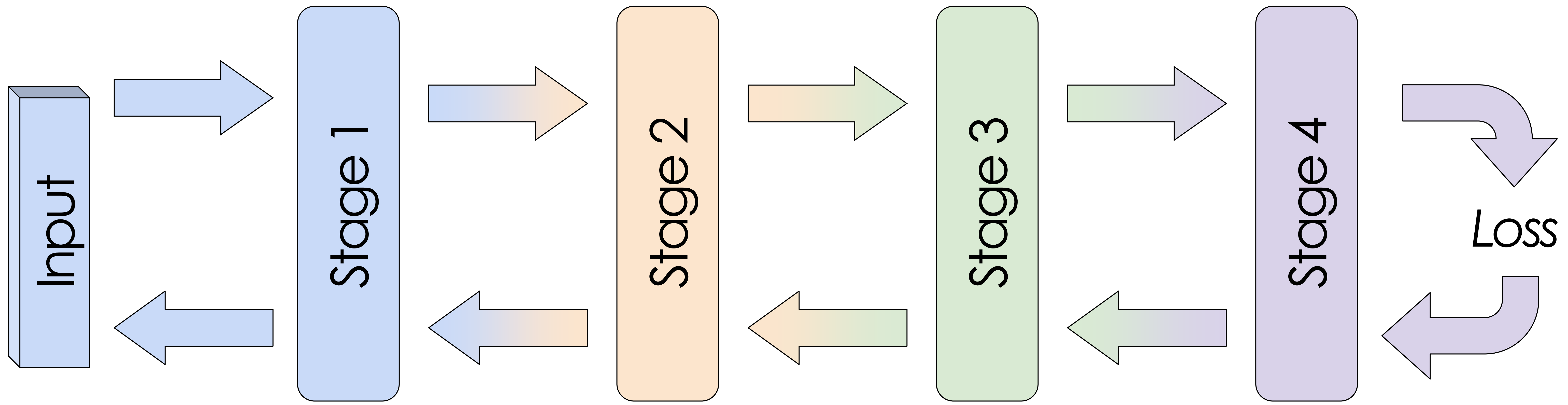
Reduce Pipeline Bubbles via Pipelining Inputs

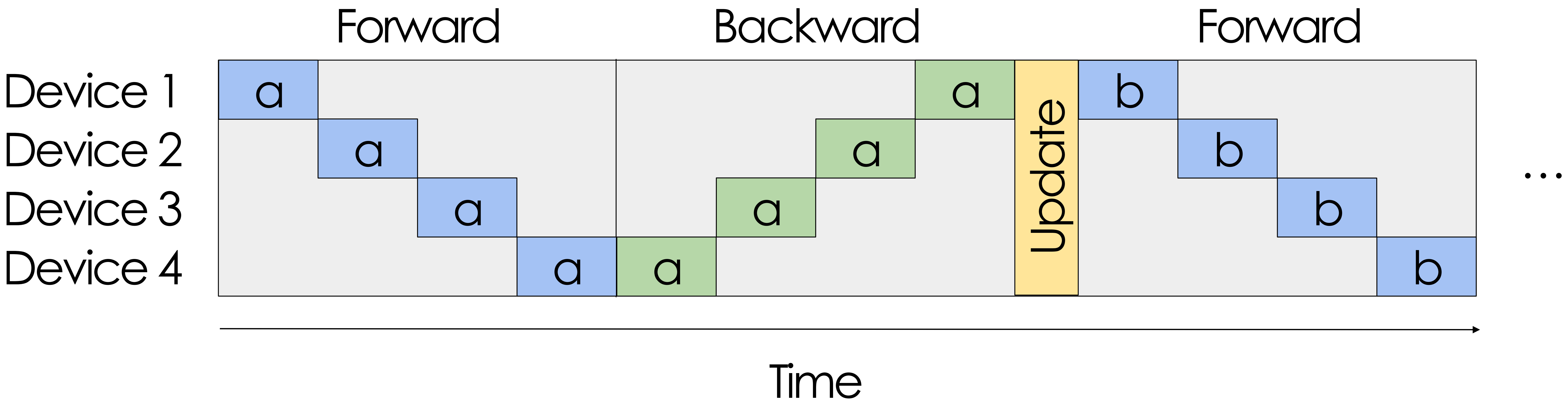


Used in inference.

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

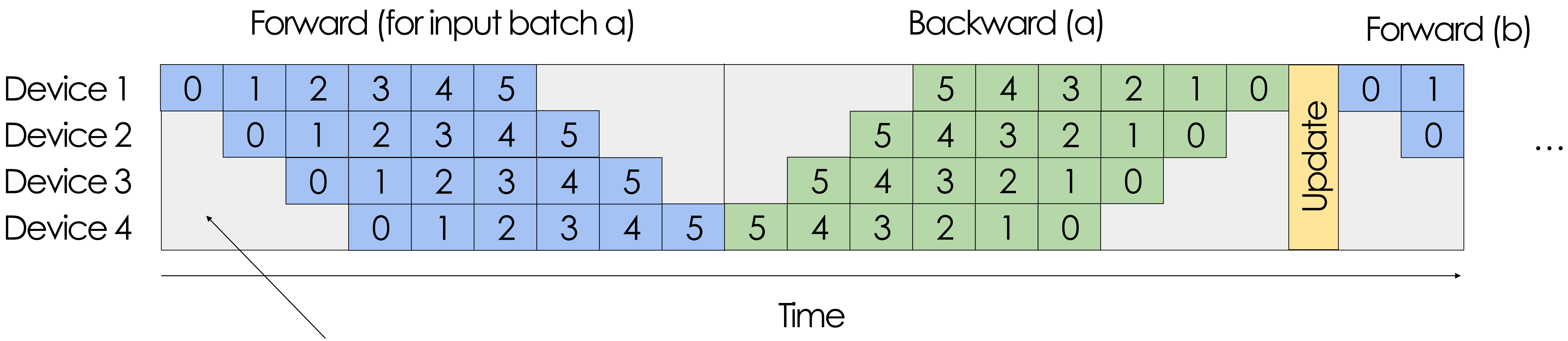
GPipe

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

Accumulate the gradients of the micro-batches:

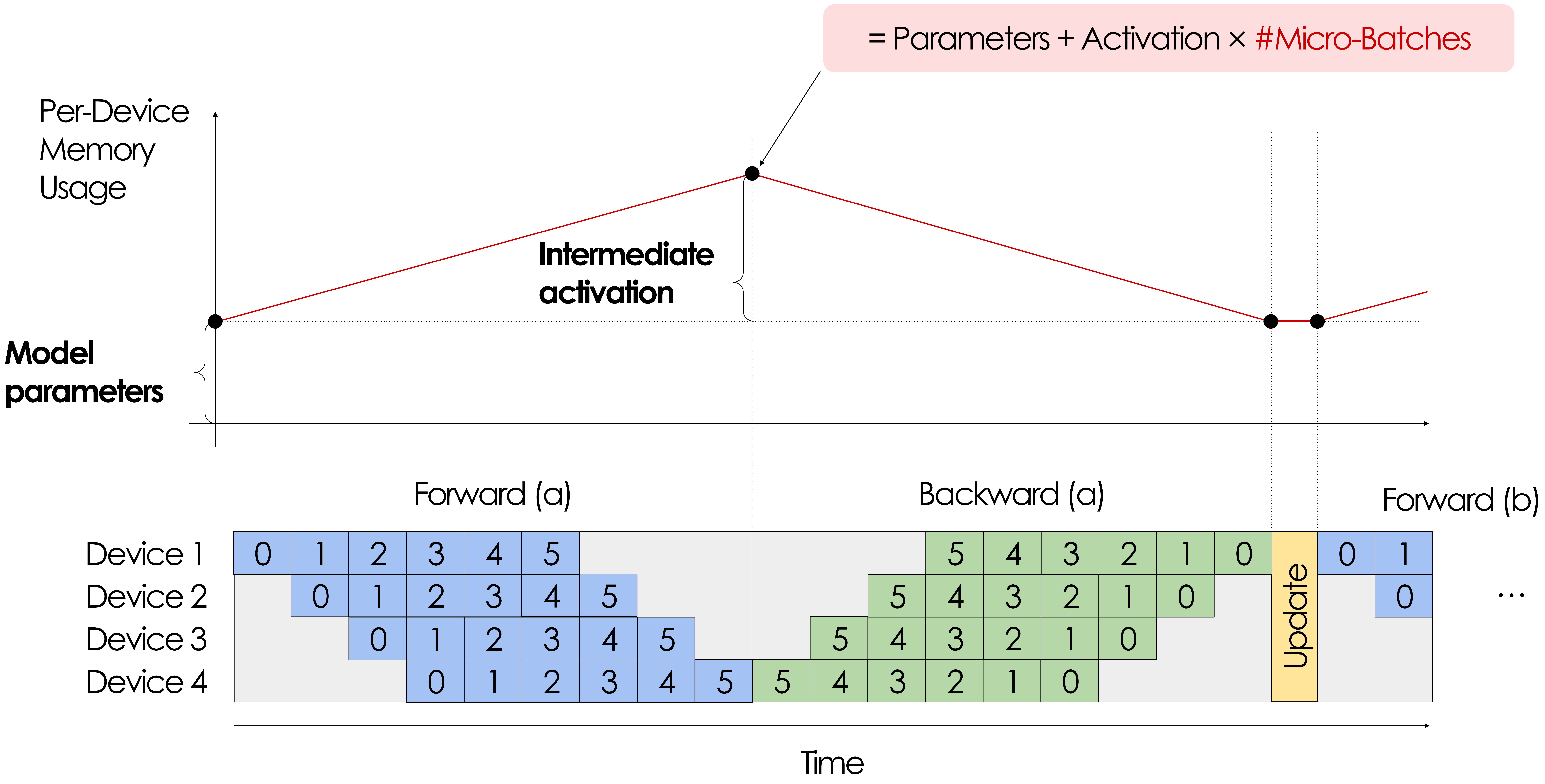
$$\nabla L_\theta(x) = \frac{1}{N} \sum_{i=1}^{N} \nabla L_\theta(x_i)$$

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

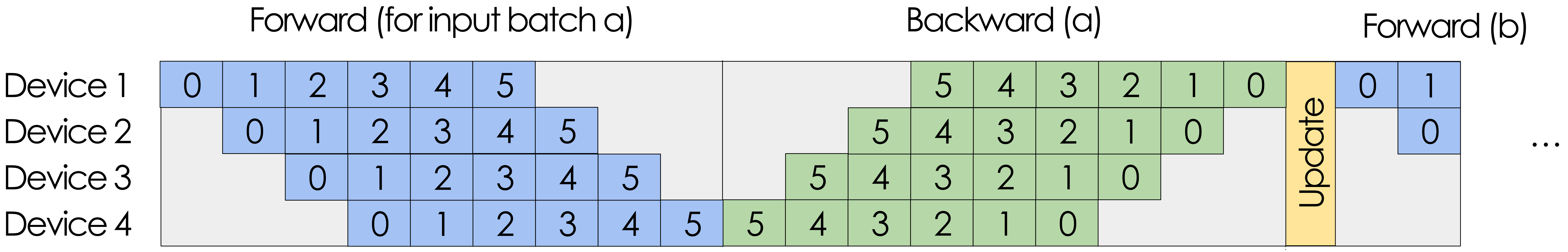


Pipeline bubbles percentage = (D - 1) / (D - 1 + N)
with D devices and N micro-batches.

GPipe: Memory Usage

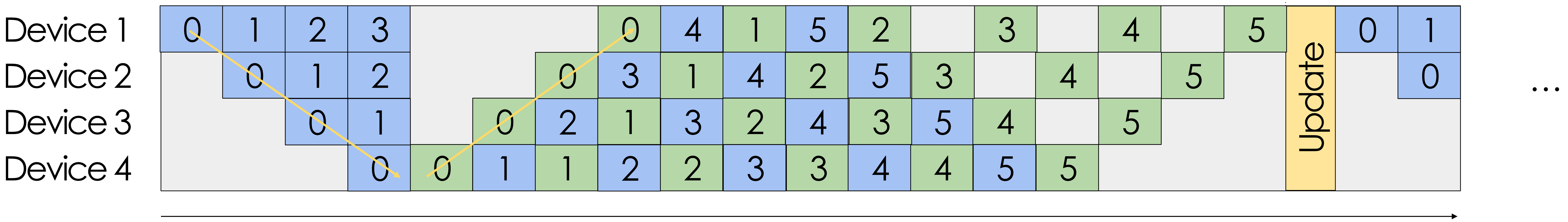


GPipe Schedule:



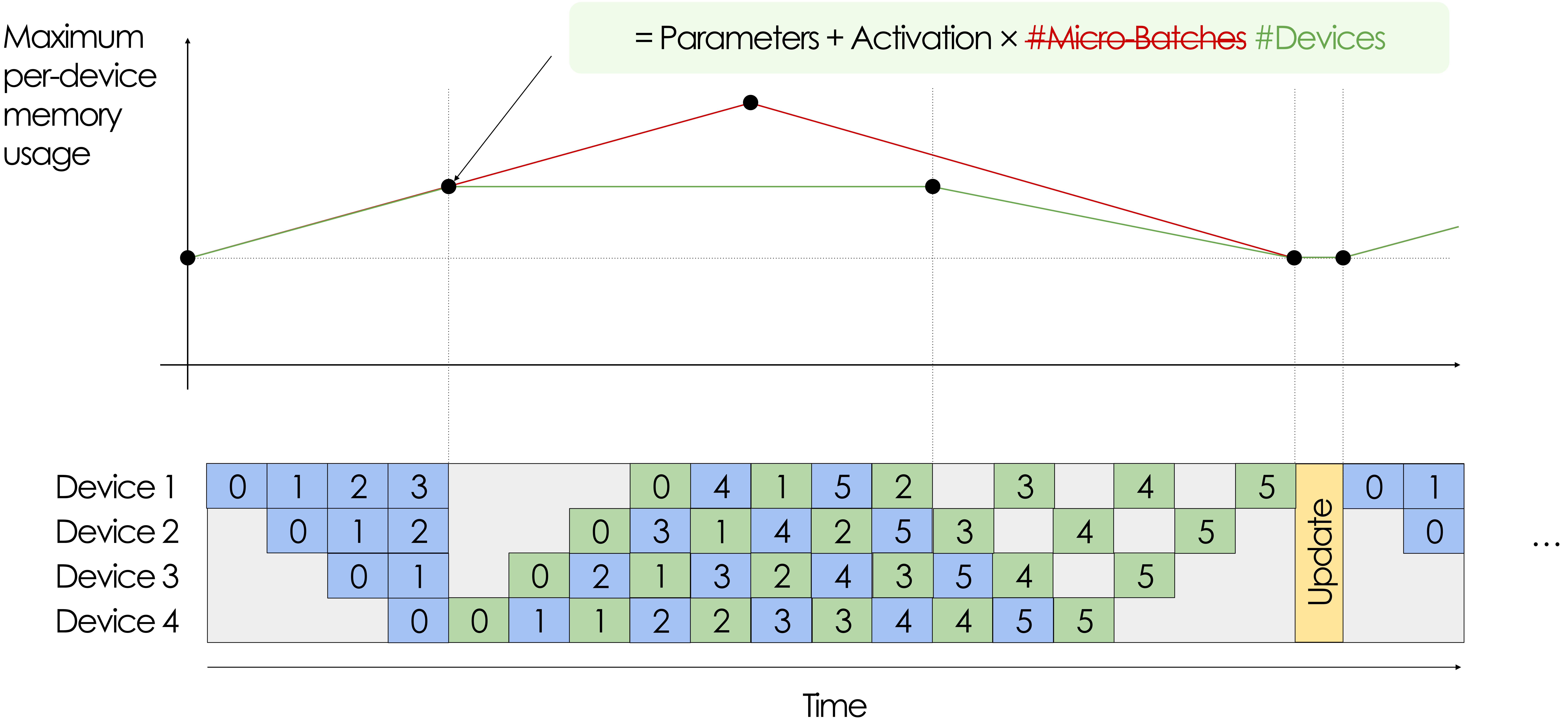
Same Latency

1F1B (1 Forward 1 Backward) Schedule:



Perform backward as early as possible

1F1B Memory Usage

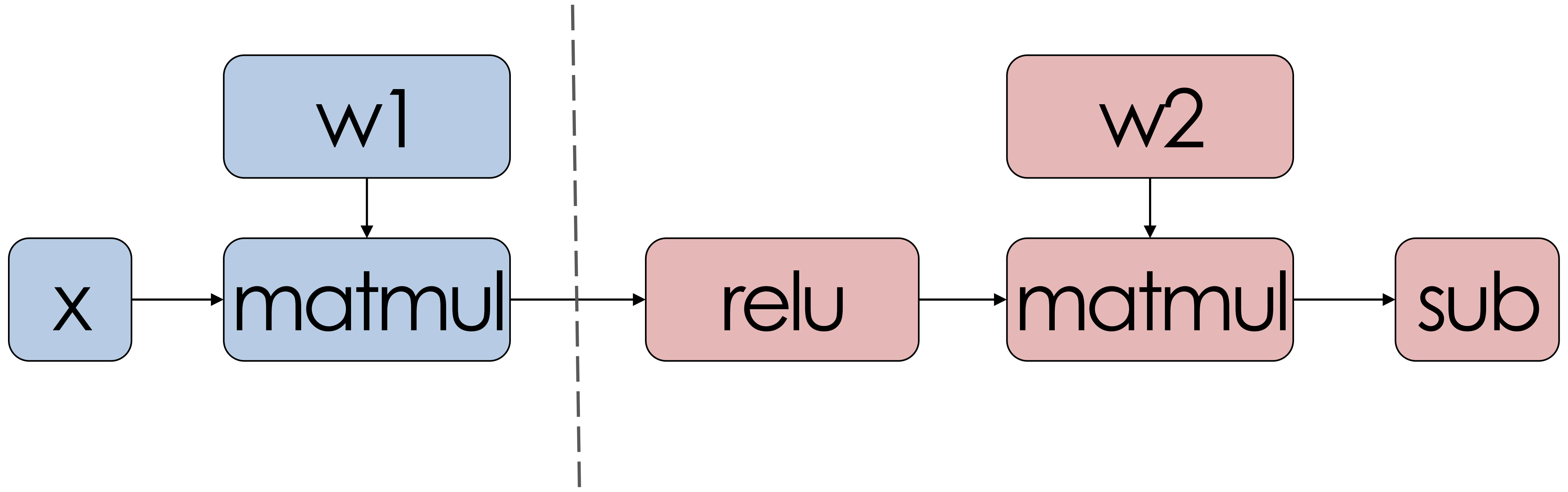


Where We Are

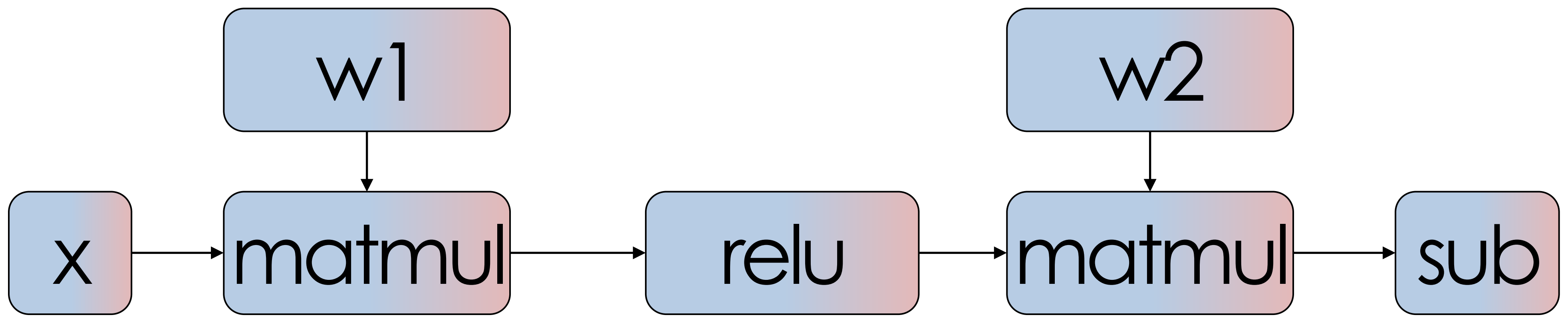
- Deep Learning as Dataflow Graphs
- Auto-differentiation Libraries
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- DL Parallelism
  - Inter-op parallelism
  - **Intra-op parallelism**

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

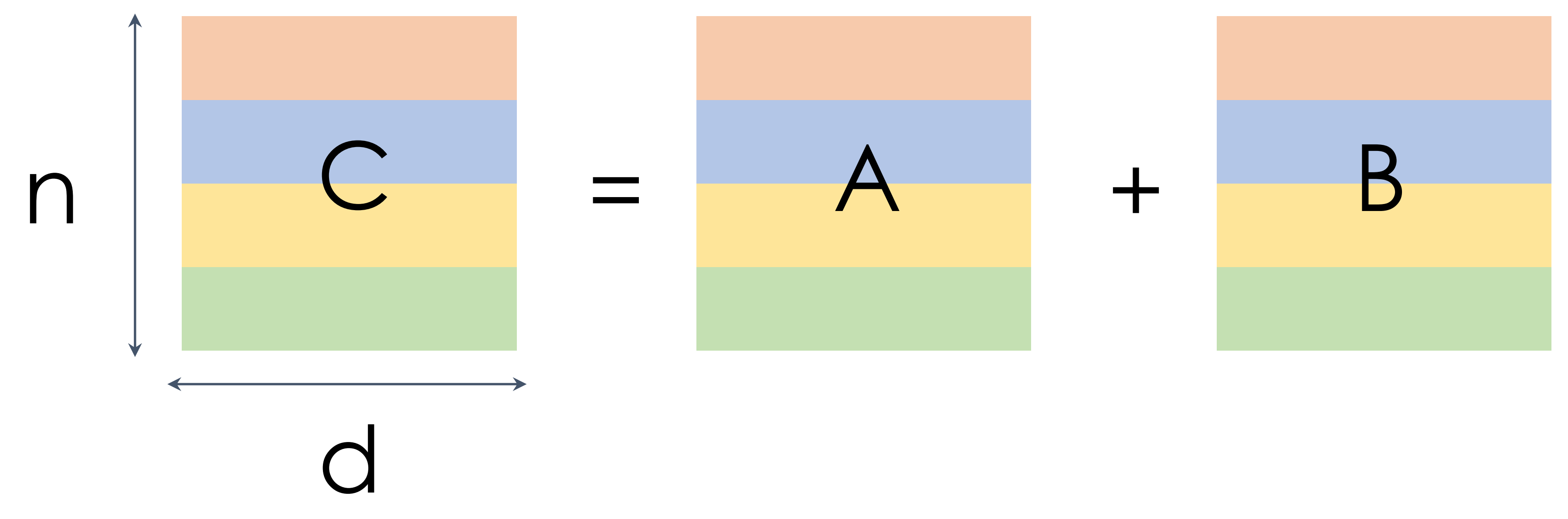
Element-wise operators

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

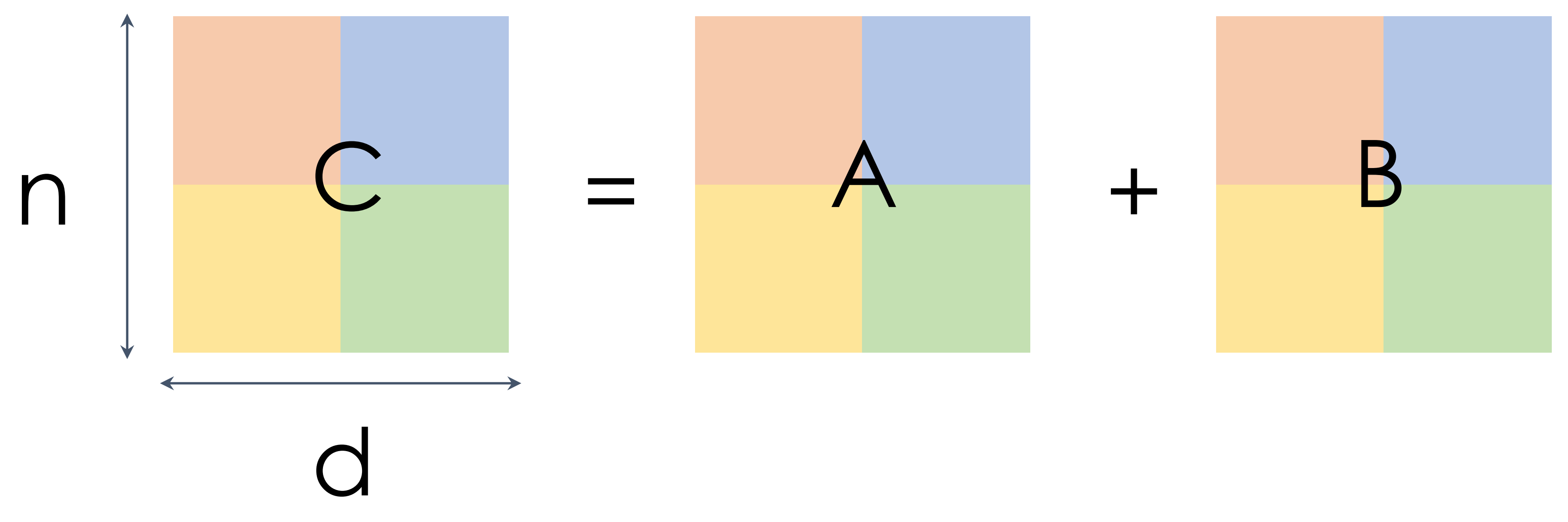
No dependency on the two for-loops.
Can arbitrarily split the for-loops on different devices.

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

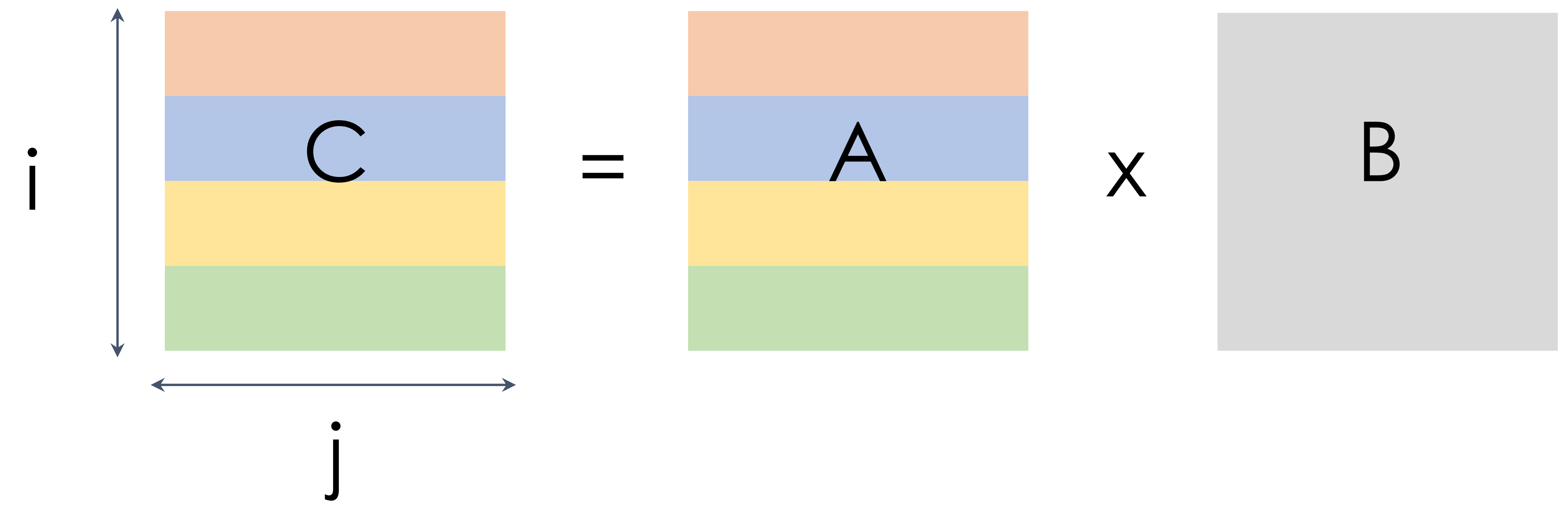
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

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

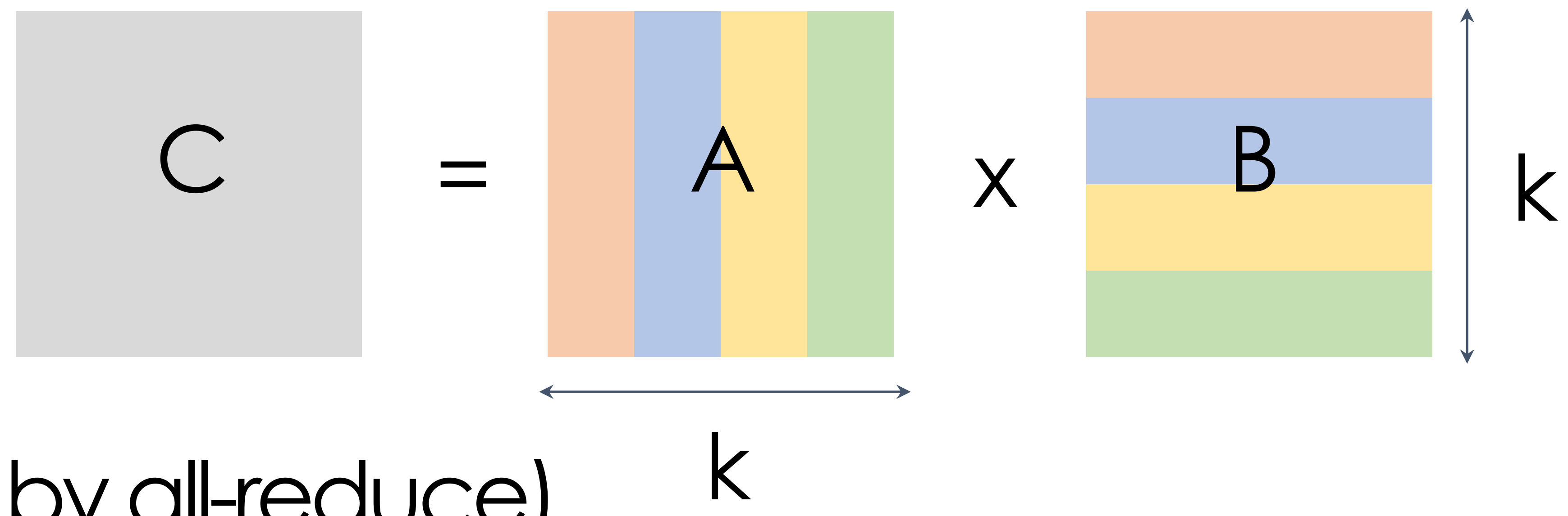
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

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

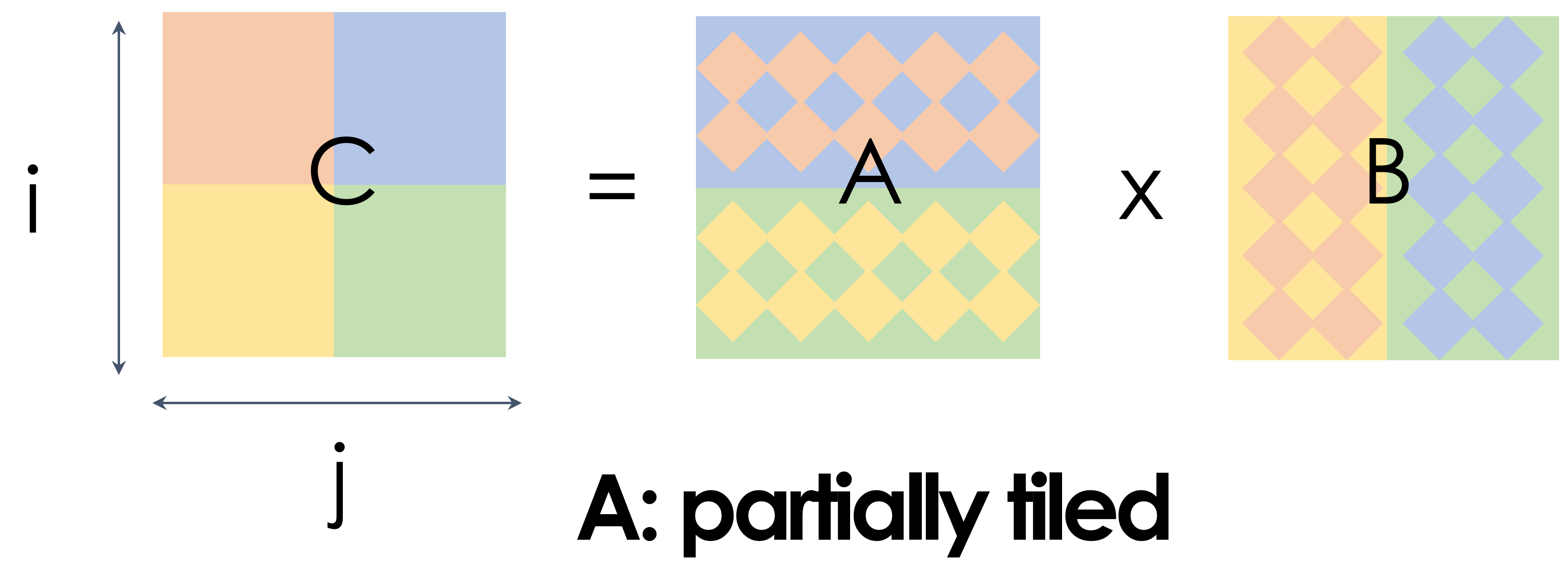
```
for i in range(0, N):
  for j in range(0, M):
    for k in range(0, K):
      C[i,j] = C[i,j] + A[i,k] x B[k,j]
```

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

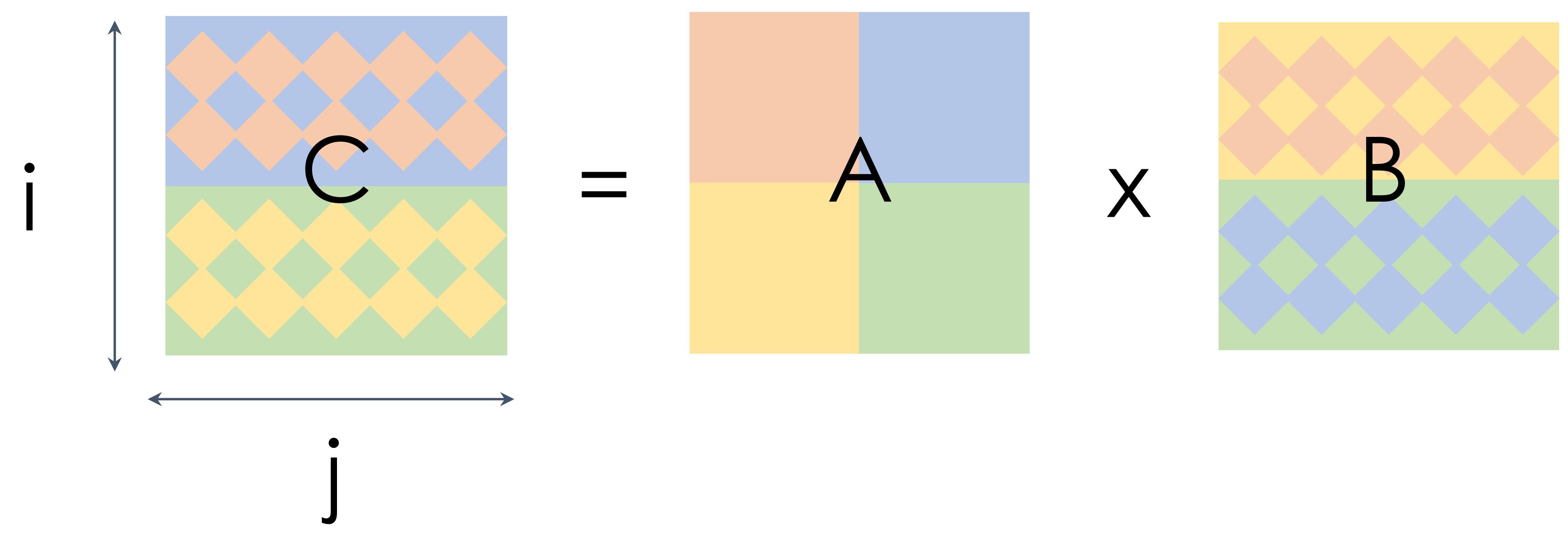


A: partially tiled

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

```
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 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.

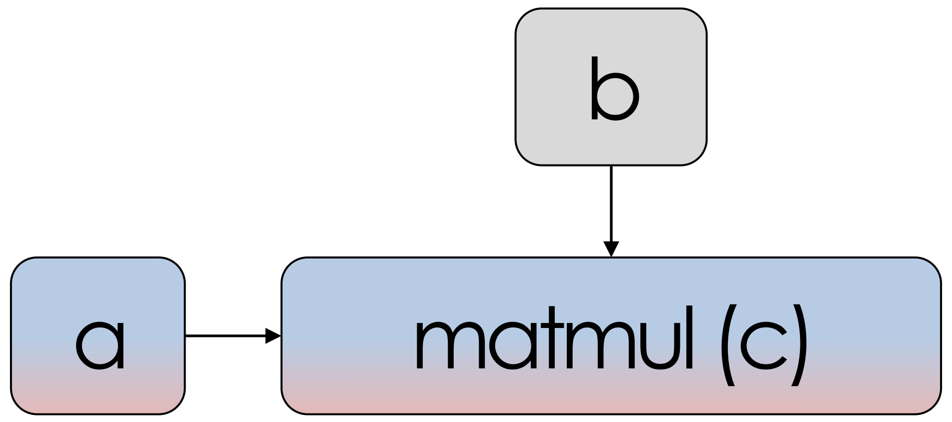
Simple case: Parallelize loop n, co, ci, then the parallelization strategies are almost the same as matmul's.

Complicated case: Parallelize loop h and w

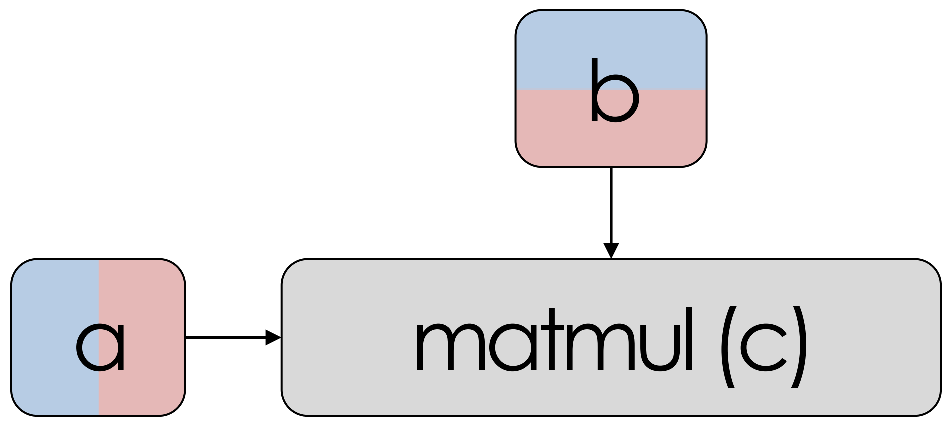
Data Parallelism as A Case of Intra-op Parallelism

Replicated Row-partitioned Column-partitioned

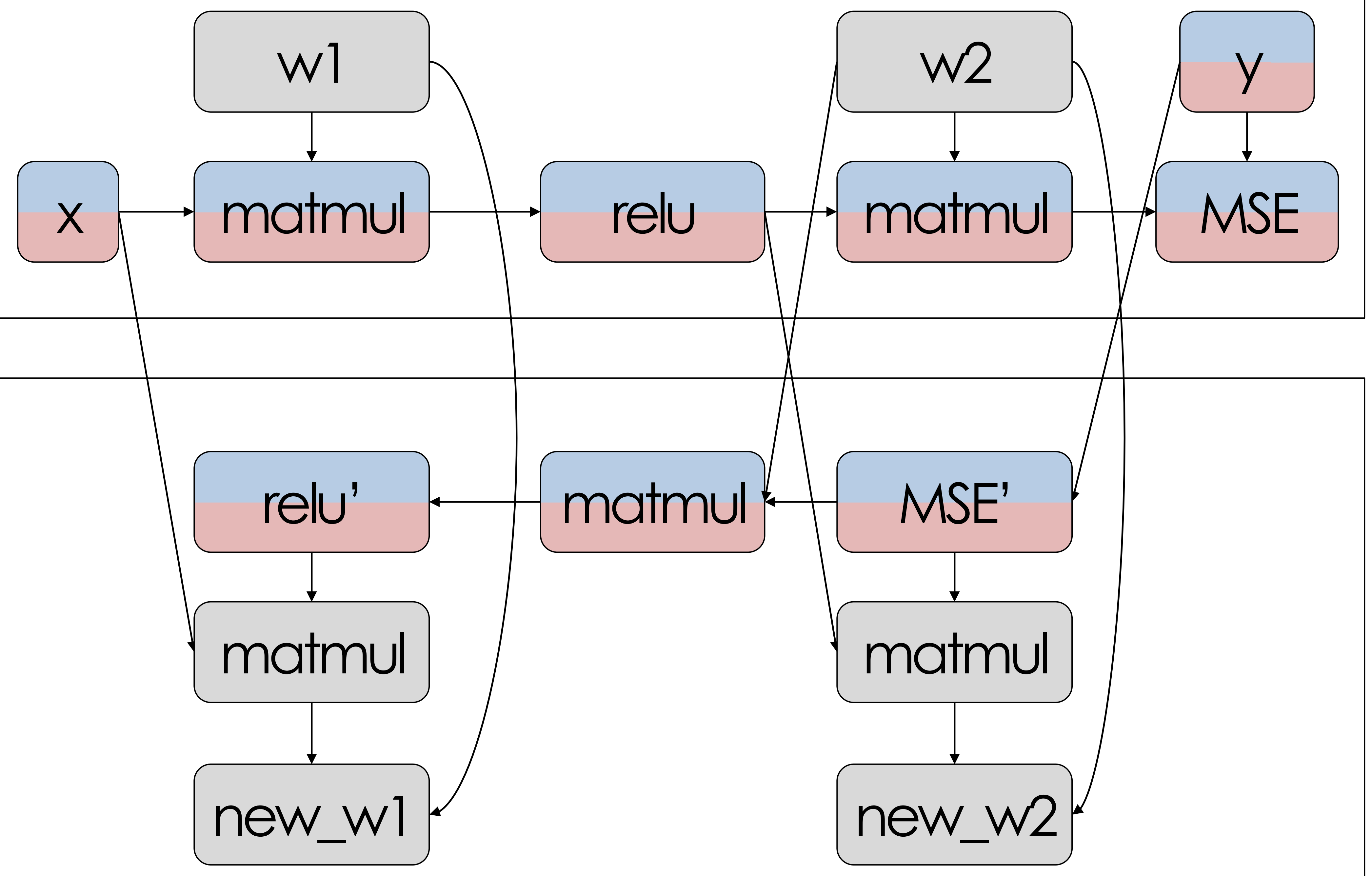
Matmul Parallelization Type 1
communication cost = 0



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



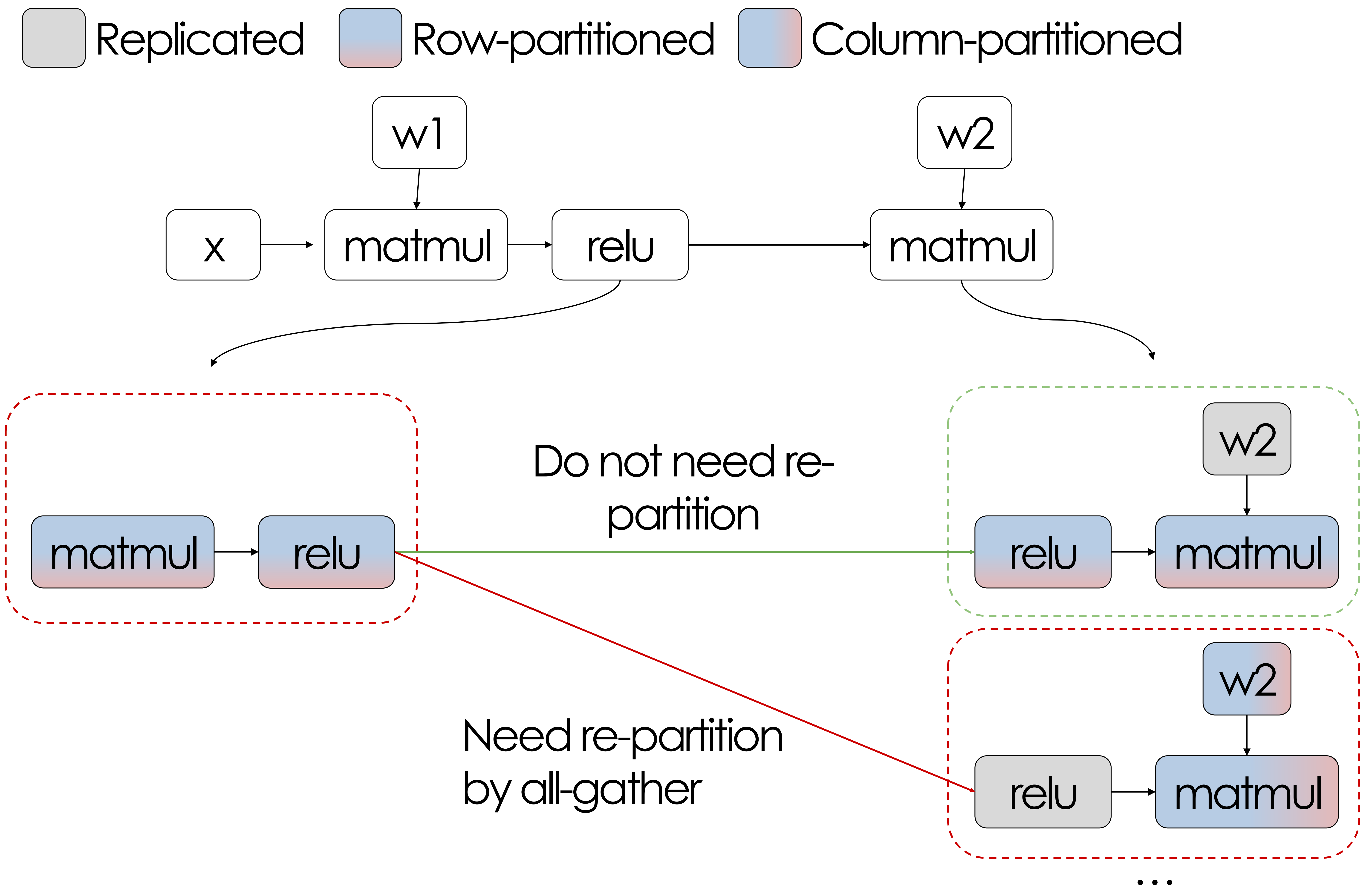
Forward Pass
Two "Type 1" matmuls: no communication



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

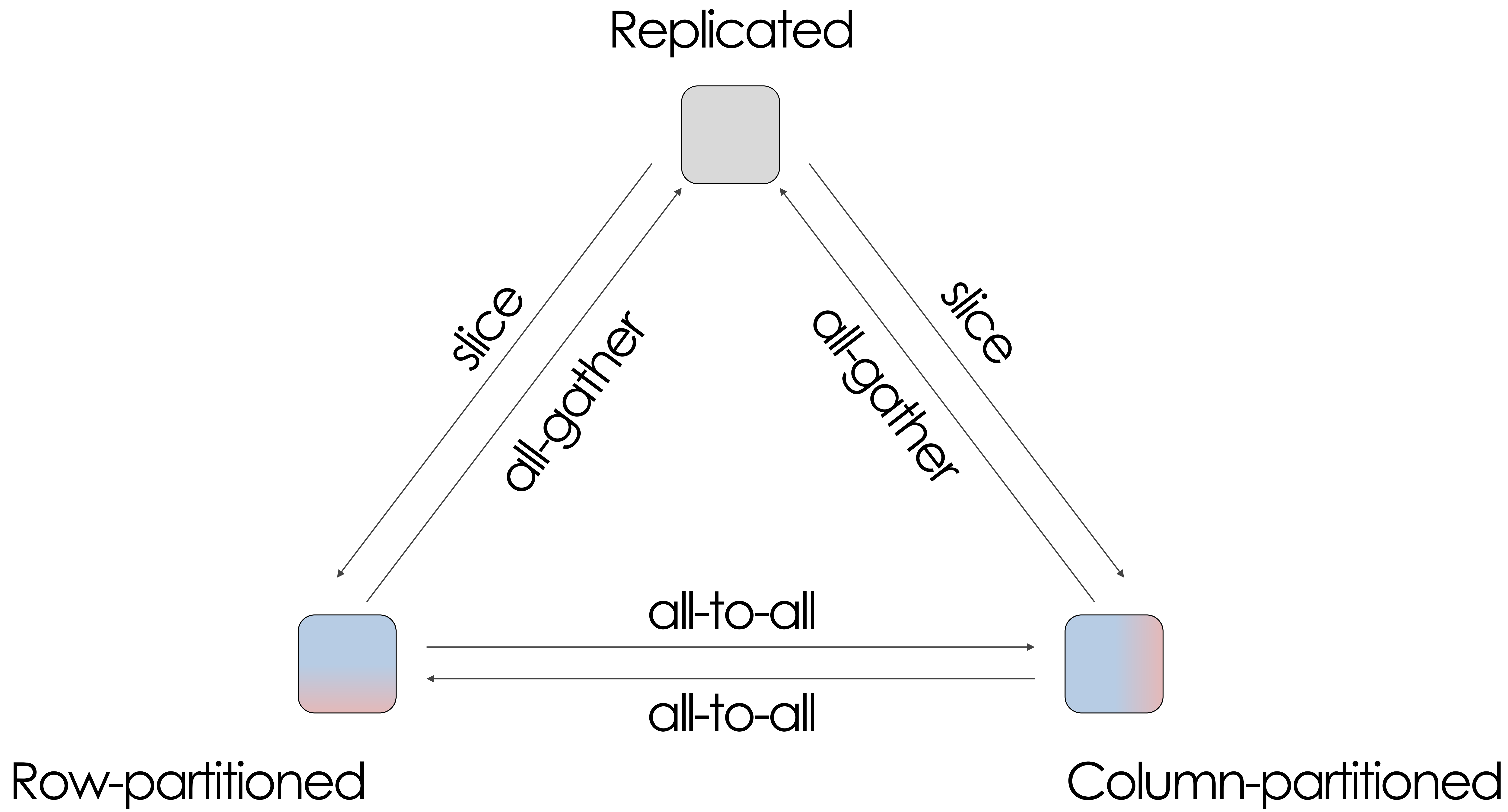
Re-partition Communication Cost

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



Re-partition Communication Cost

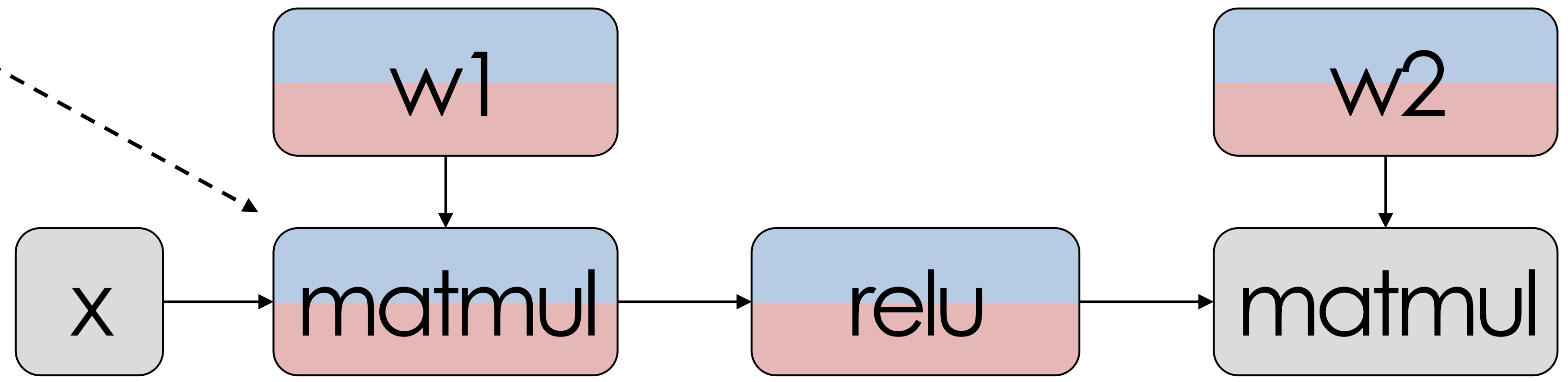
Different operators' parallelization strategies require different partition format of the same tensor



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

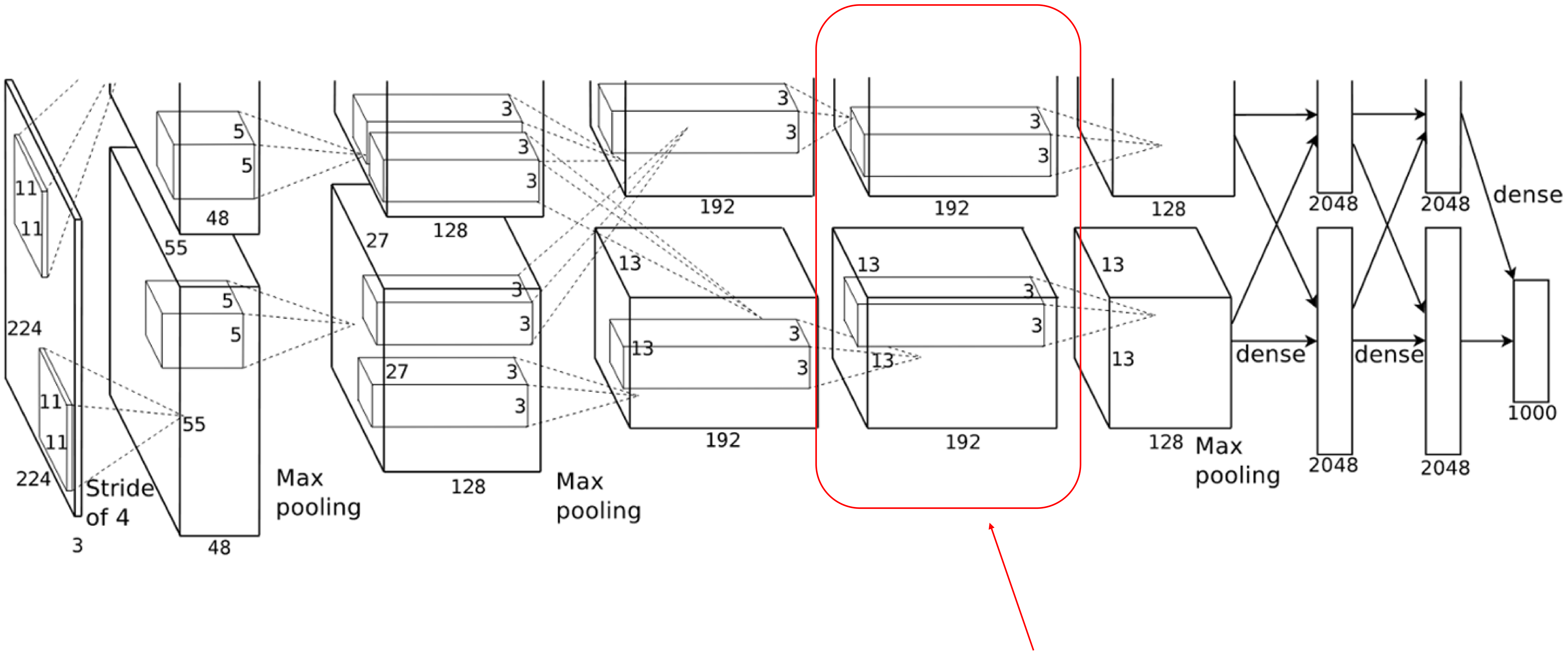
Important Projects

- 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%

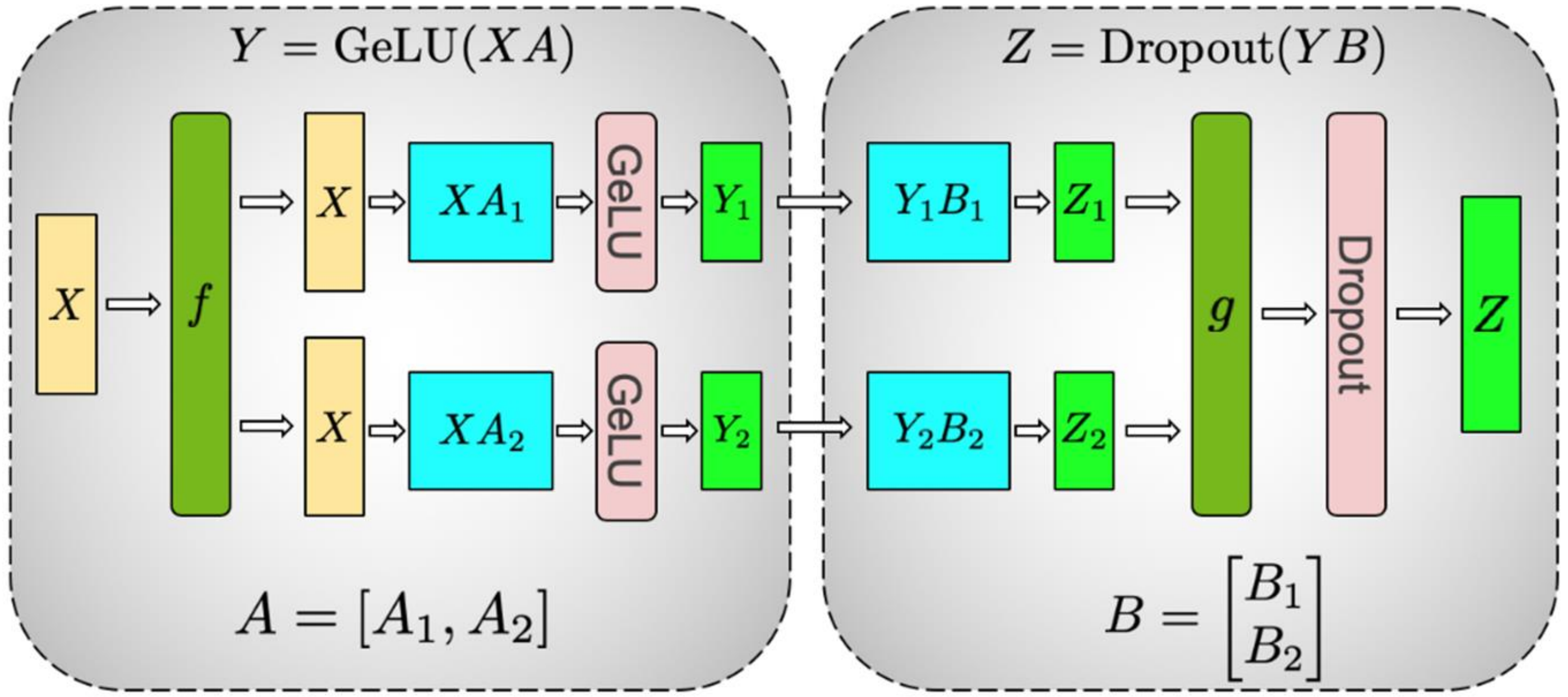


Assign a group convolution layer to 2 GPUs

Megaton-LM

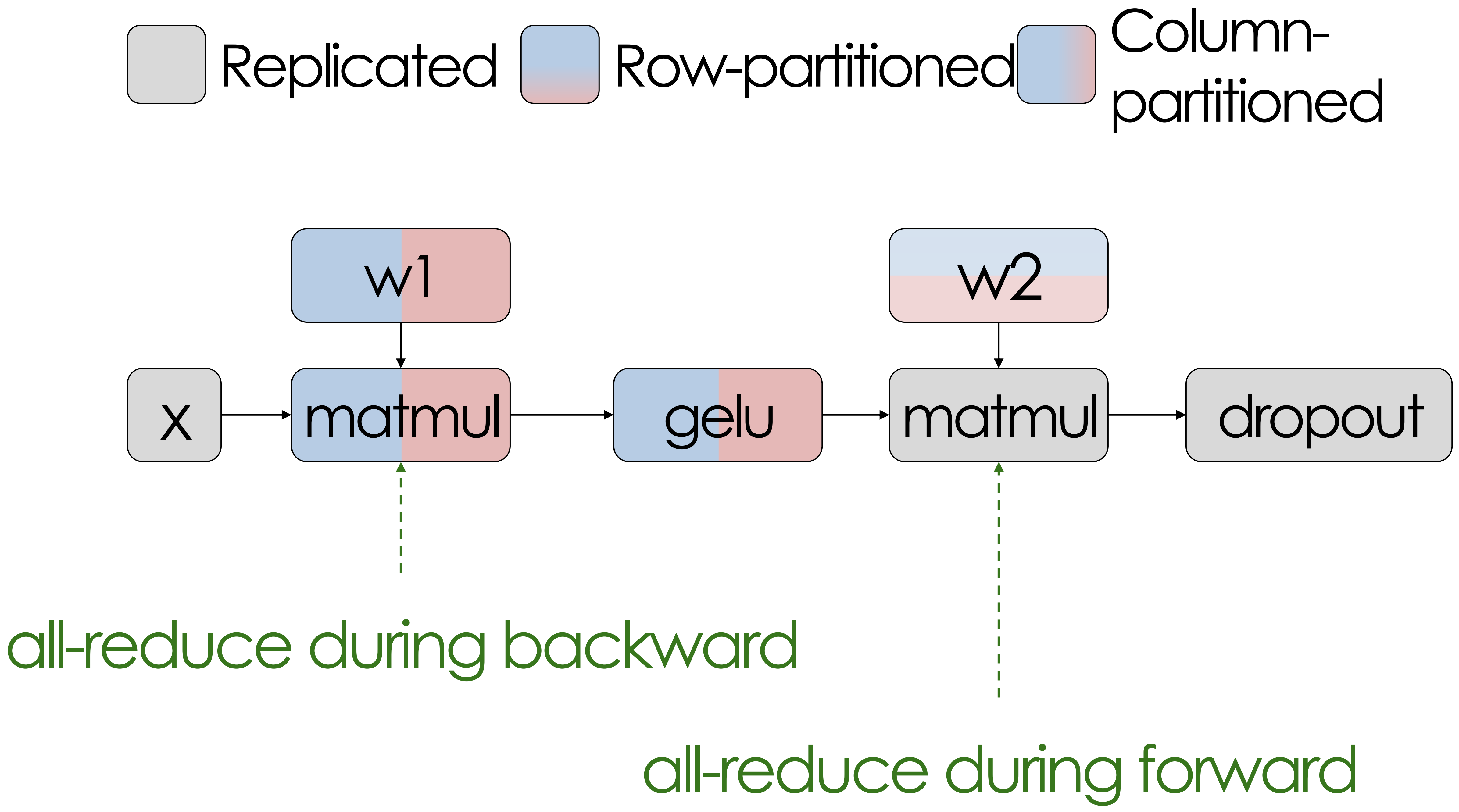
Result: a large language model with 8.3B parameters that outperforms SOTA results

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



(a) MLP

Illustrated with the notations in this tutorial

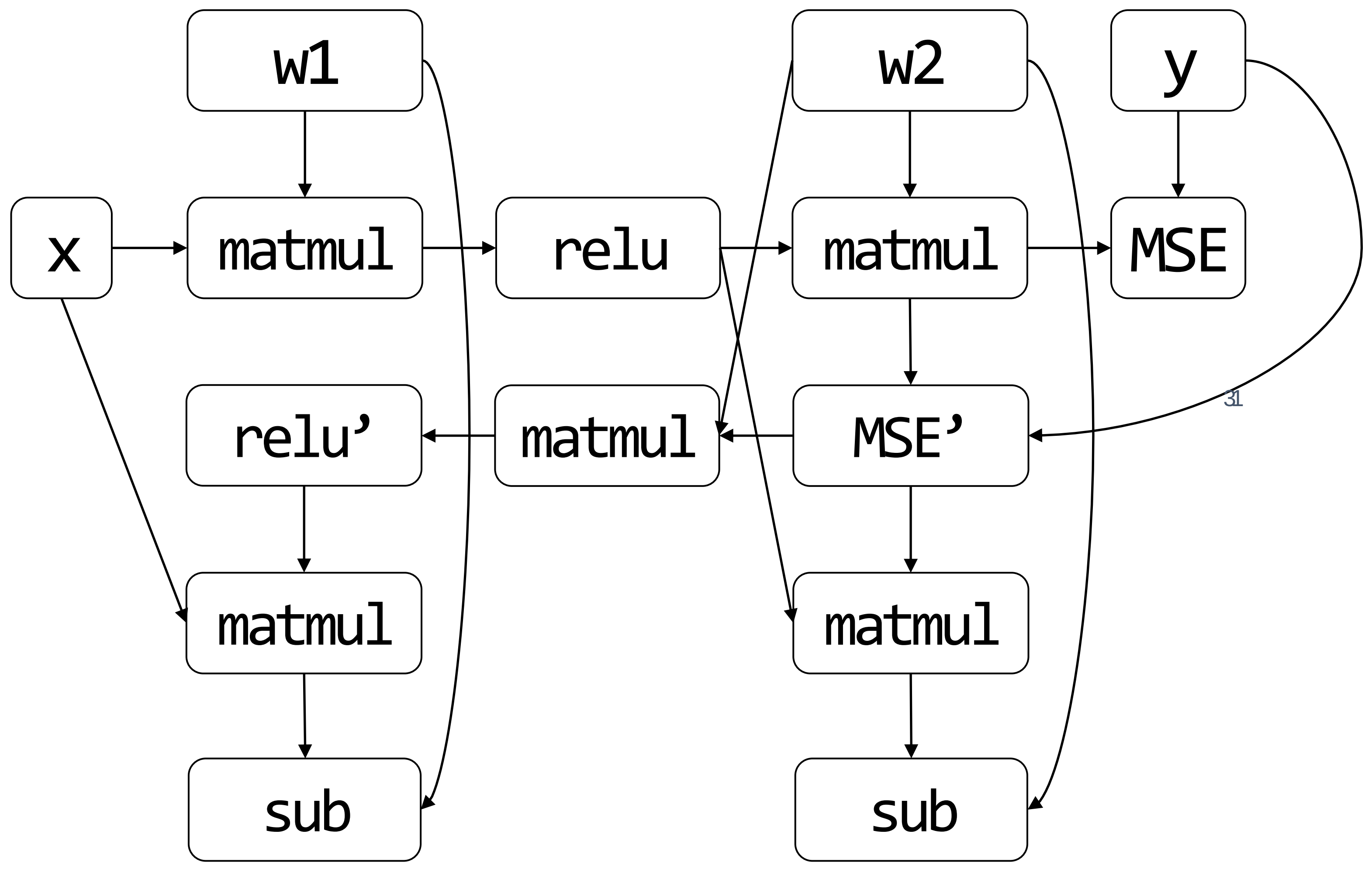


Intra-operator Parallelism Summary

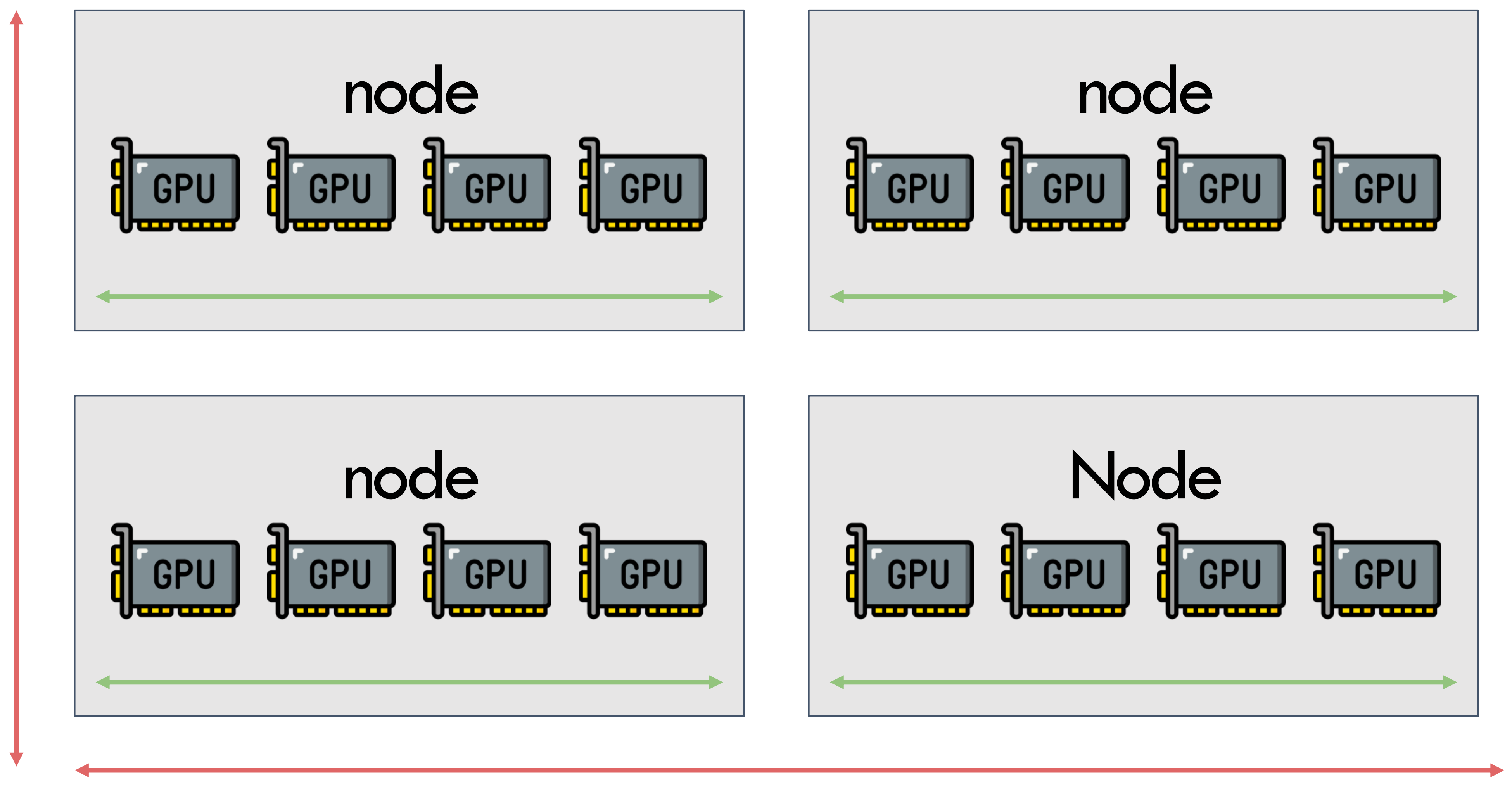
- We can parallelize a single operator by exploiting its internal parallelism
- To do this for a whole computational graph, we need to choose strategies for all nodes in the graph to minimize the communication cost
- Intra-op and inter-op can be combined

Advanced Topic: Auto-parallelization

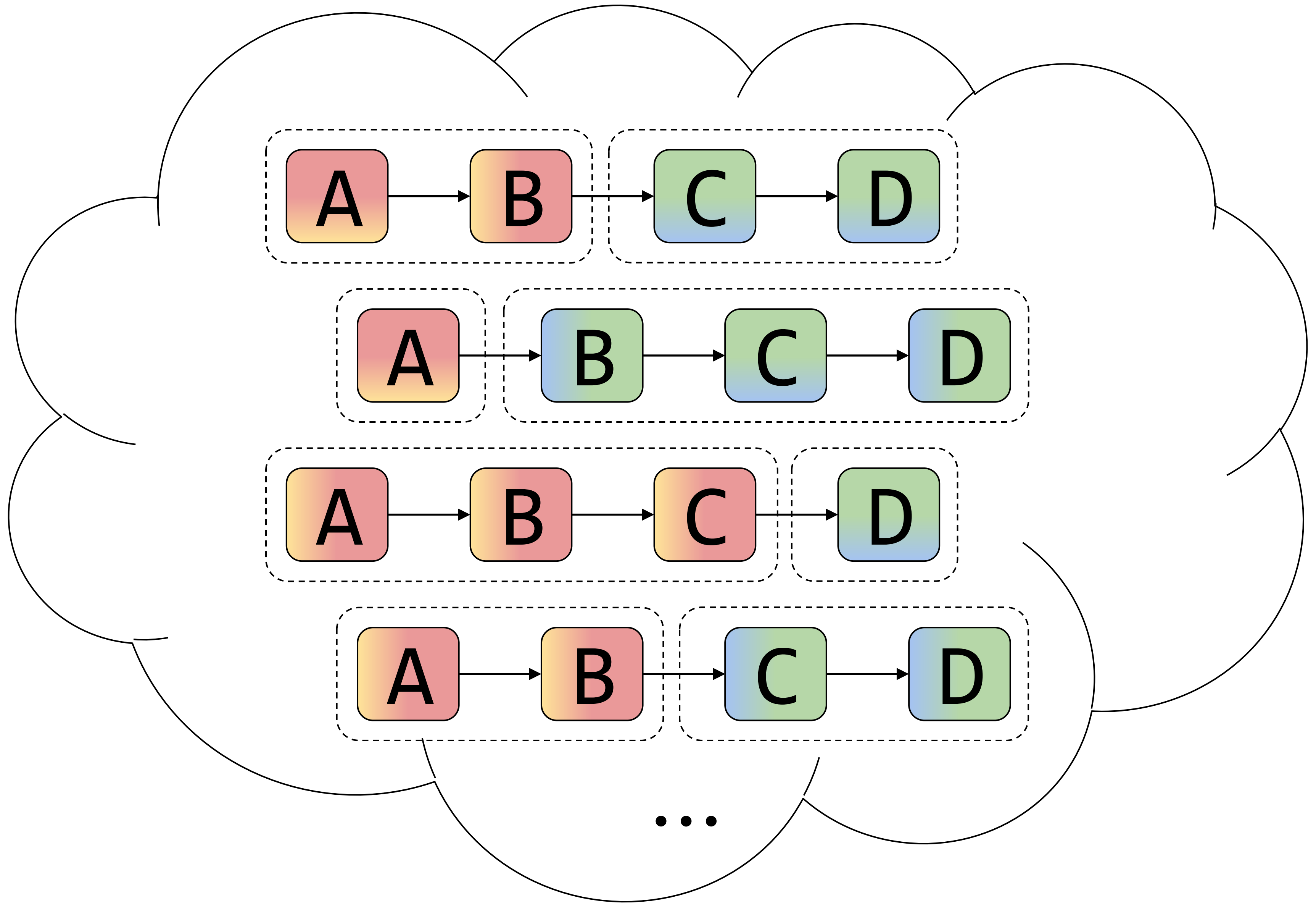
Model



Cluster



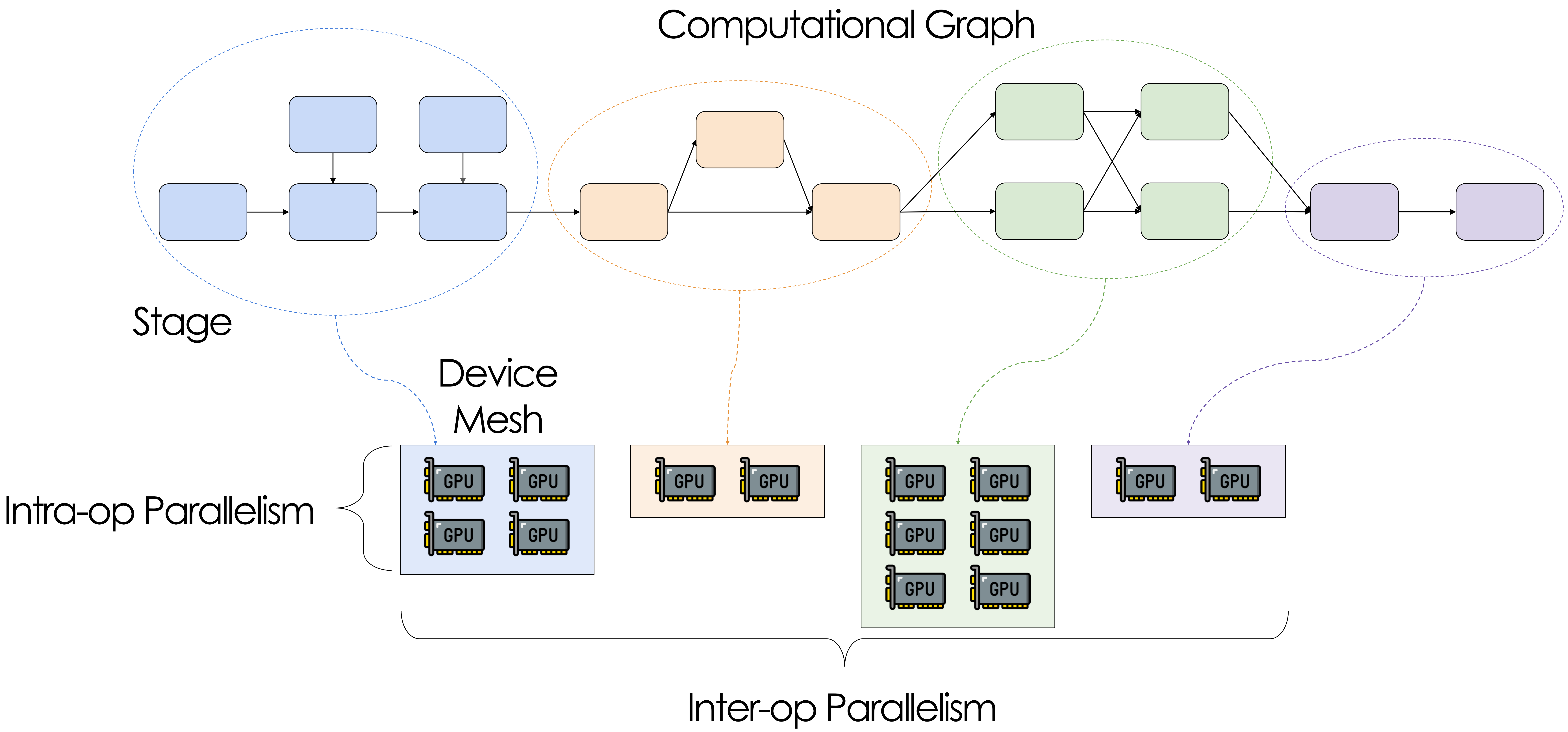
Strategy



Auto-parallelization: Problem

$$\max_{\text{strategy}} \ \text{Performance}(\text{Model, Cluster})$$
$$s.t. \ \text{strategy} \in \text{Inter-op} \cup \text{Intra-op}$$

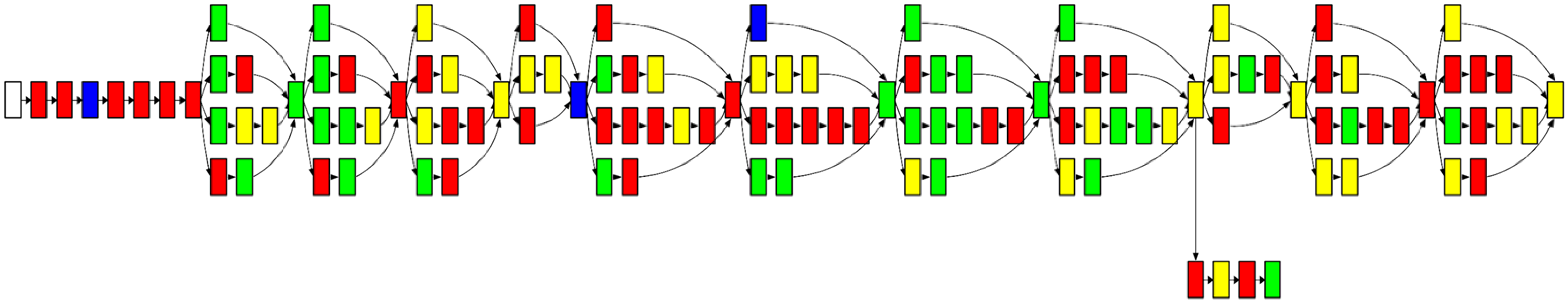
Combine Intra-op Parallelism and Inter-op Parallelism



The Search Space is Huge

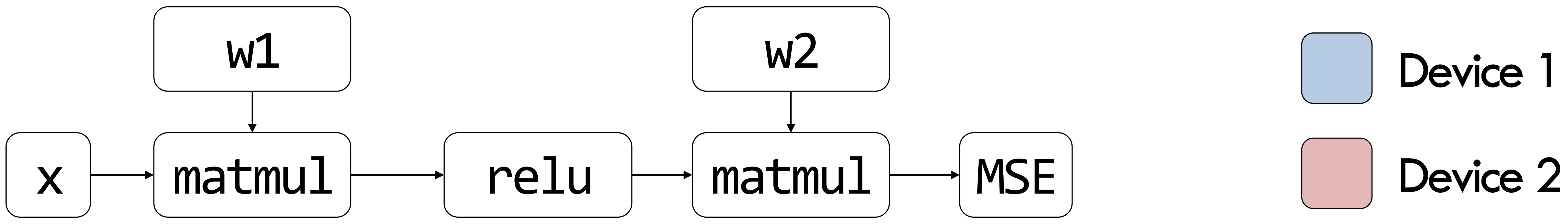
#ops in a real model (nodes to color)	#op types (type of nodes)	#devices on a cluster (available colors)
100 - 10K	**80 - 200+**	**10s - 1000s**

One Inefficient Way: Search (by Google)

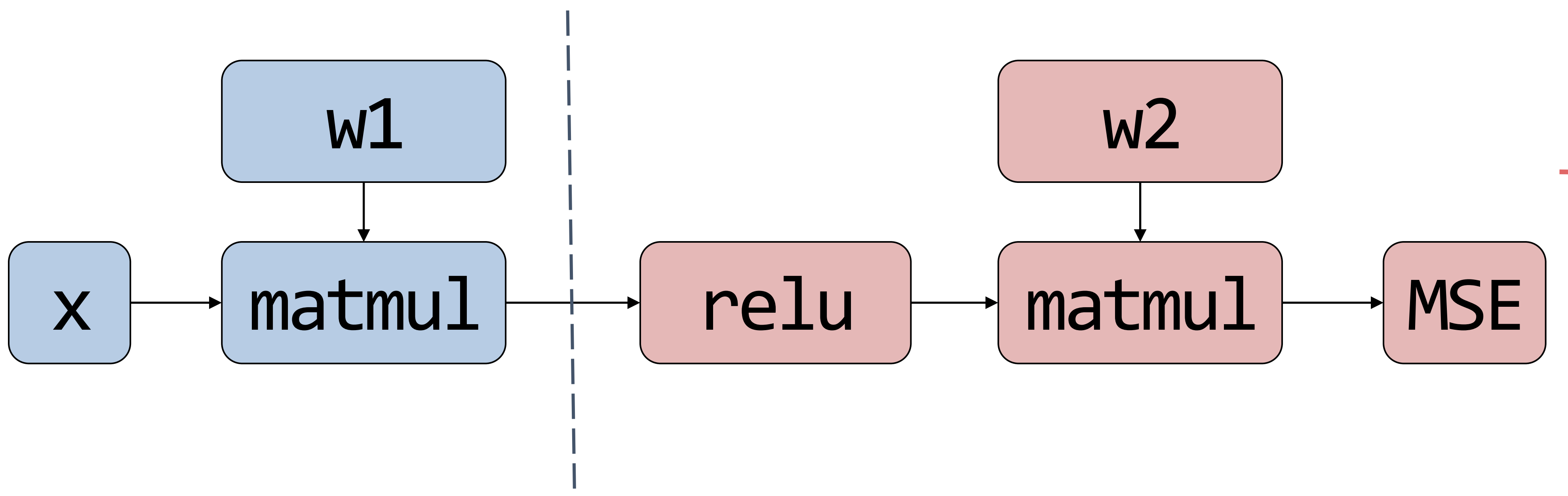


Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**	2	25.24	22.88	11.22	**4.60**	0.0%
			4	23.41	24.52	10.65	**3.85**	19.0%

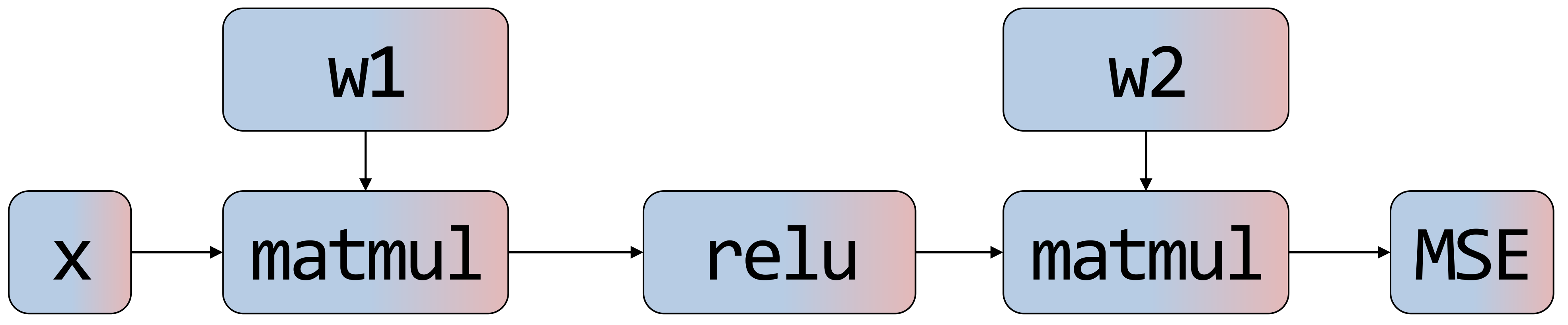
High-level Idea to avoid exhausted search

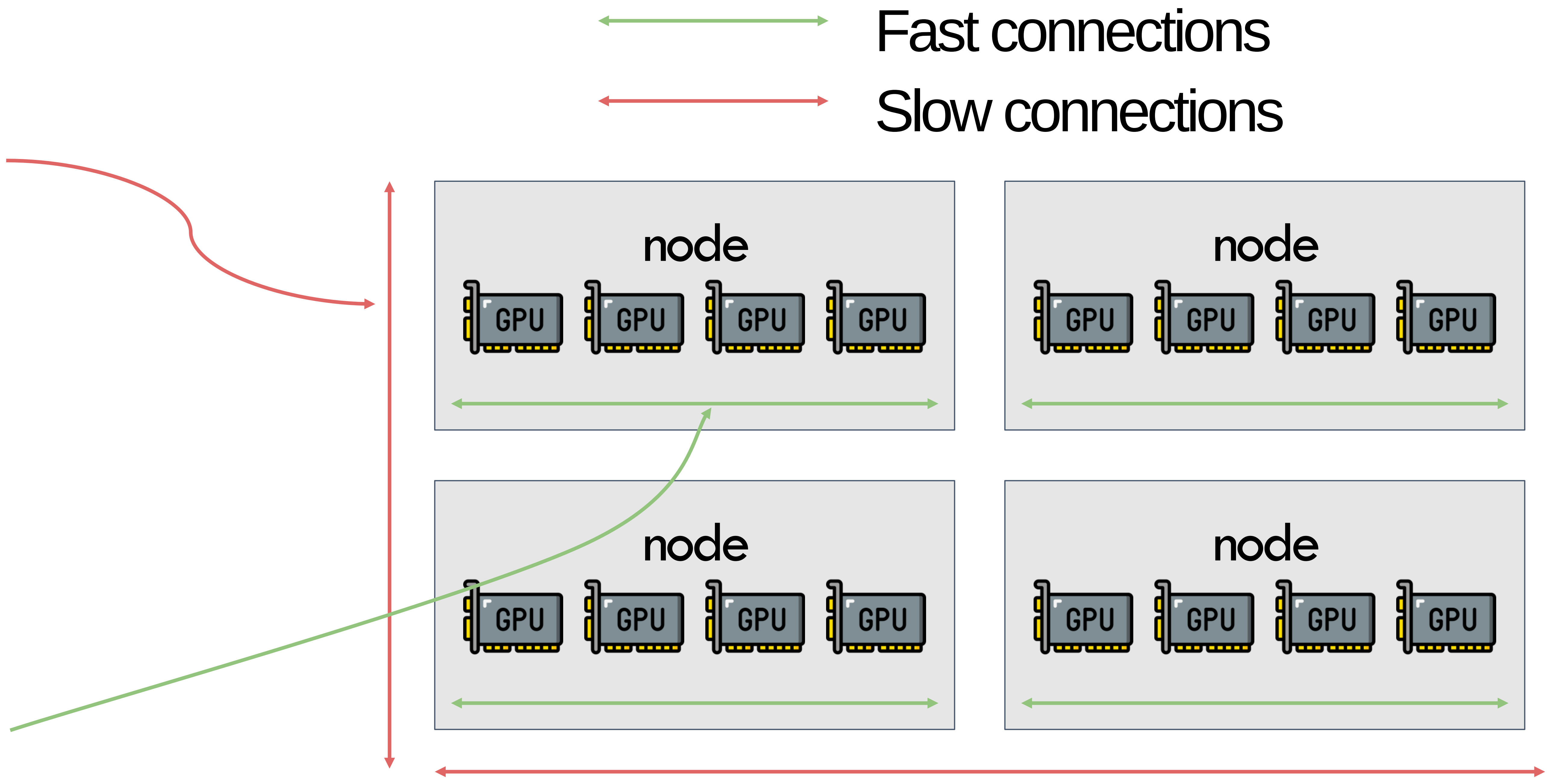


Inter-op parallelism

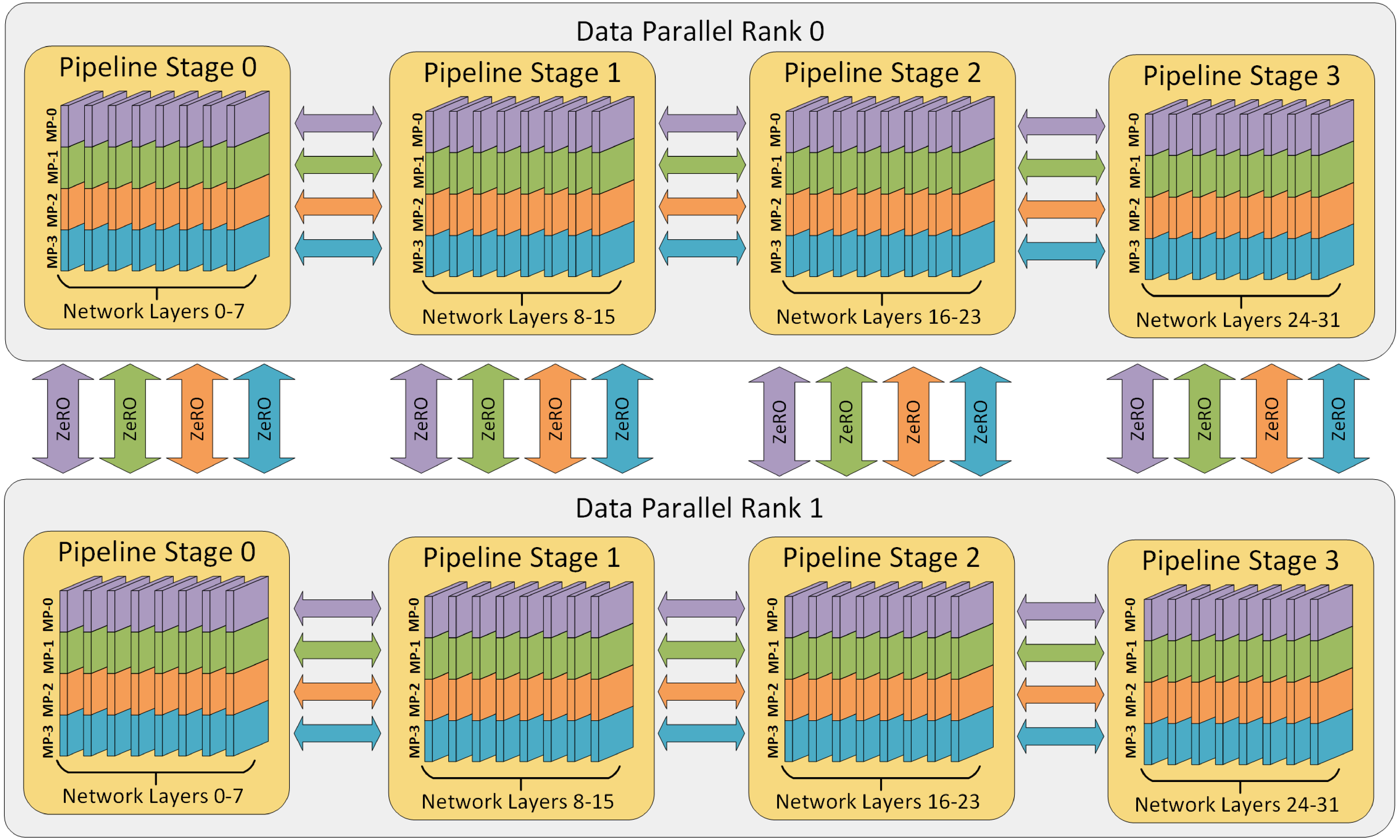


Intra-op parallelism





How GPT-3/4 Training solves this: Massively Parallel



Problems We haven’t covered in ML Systems

- Graph optimization and compilation
- Machine learning for systems
- GPU Architectures
- TinyML: ML on edge devices
- Federated learning
- Transformer-specific optimizations
- ML/LLM Serving
  - Continuous batching
  - Paged attention
  - Speculative decoding

My Upcoming Teaching Schedule

- Spring 24
  - DSC 291: ML Systems
- Fall 24
  - Chill
- Winter 25
  - CSE/DSC 234: ML Systems (DSC 291 will be lifted to this one)
- Spring 25
  - DSC 204A (This course again)

Thank you!
(remember to submit course eval for your own/my/TA's/benefits!)