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**
  - TAs and I are still debating between classroom or canvas
    - Will update you by this Wed (3/13)
  - All Multiple choice questions
- Course Evaluation
  - It is important for both me, yourself, and TAs
  - Please participate to get your extra credits ☺

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

Recap: Parameter Server

- Pros?
  - General: Abstract iterative-convergent algo
  - Relax Consistency: stale synchronous
  - Nice interface like map-reduce
- Cons?
  - Extension to GPUs?
  - Strong assumption on communication bottleneck

The Second Unification: Neural Networks

Imagenet classification with deep convolutional neural networks [PDF] neurips.cc

A Krizhevsky, I Sutskever… - Advances in neural …, 2012 - proceedings.neurips.cc

We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the …

Save Cite Cited by 126745 Related articles All 102 versions

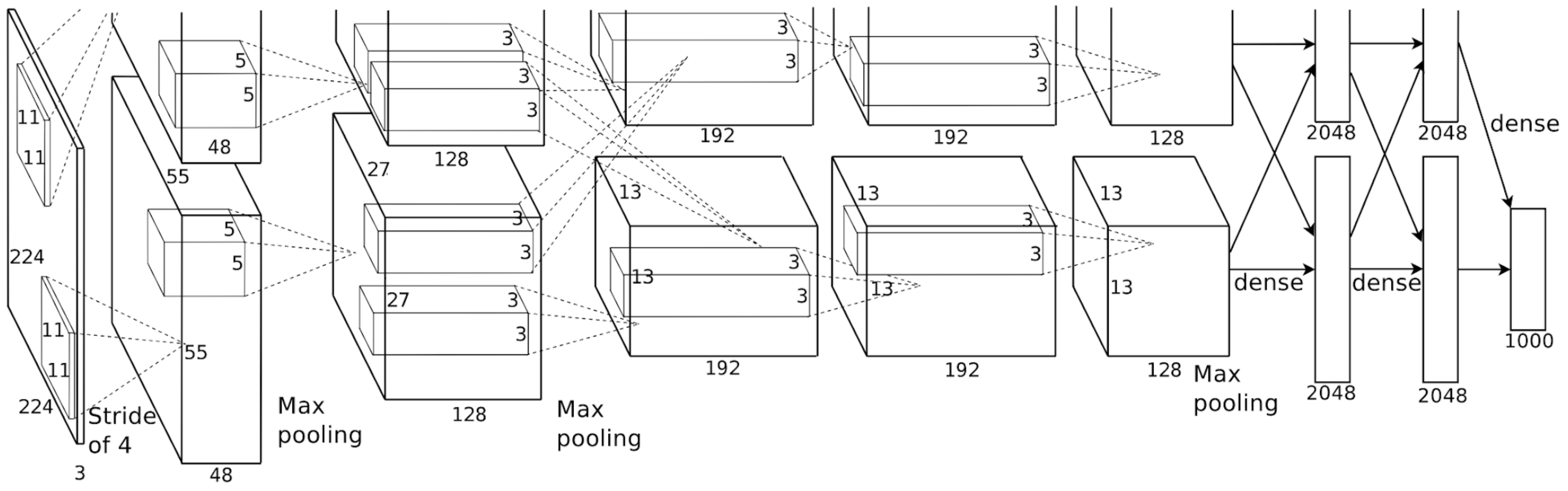


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure from AlexNet

[Krizhevsky et al., NeurIPS 2012], [Krizhevsky et al., preprint, 2014]

Why DL Emerged and Succeeded(but failed before)?

- It beats the previous state-of-the-art method by 10 points
  - Every year we see 1 point improvement in the past 10 years
- It scales with the size of data
  - Train with the entire ImageNet data
- It is simple: optimized by 1st-order method SGD
- Its computation pattern aligns with hardware (GPU/accelerators)

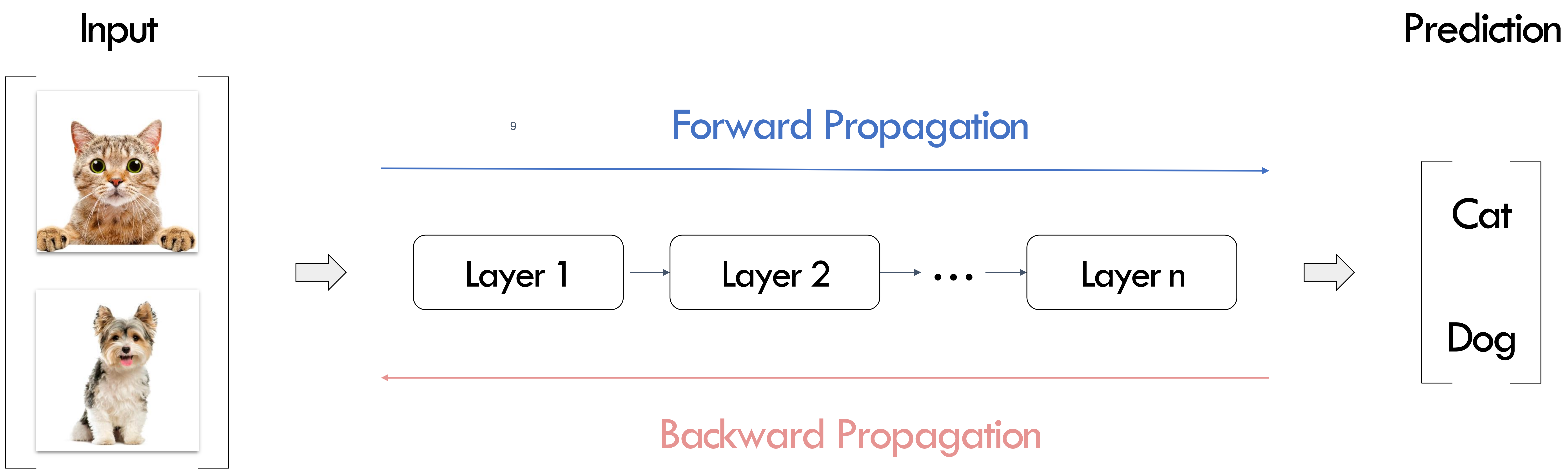
Deep learning Characteristics

- Iterative-convergent?
  - Yes: SGD
- Model still diverse?
  - No: much less diverse than the entire spectrum of ML
  - Yes: Still many flavors of NNs and needs sufficiently expressive lib to program various architectures
- Compute very intensive?
  - Yes: GPU becomes a must
- Model very large?
  - No: It starts with a relatively small model (2012)
  - Yes: It becomes large when people discover the transformer architecture
- Existing data systems to program NNs?
  - Map-reduce: not for iterative-convergence
  - Spark: op lib is very corase grained and not for neural network ops
  - P: programming model offers too many flexibility which renders it not so helpful

Outline

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

Background: DL Computation



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

parameter

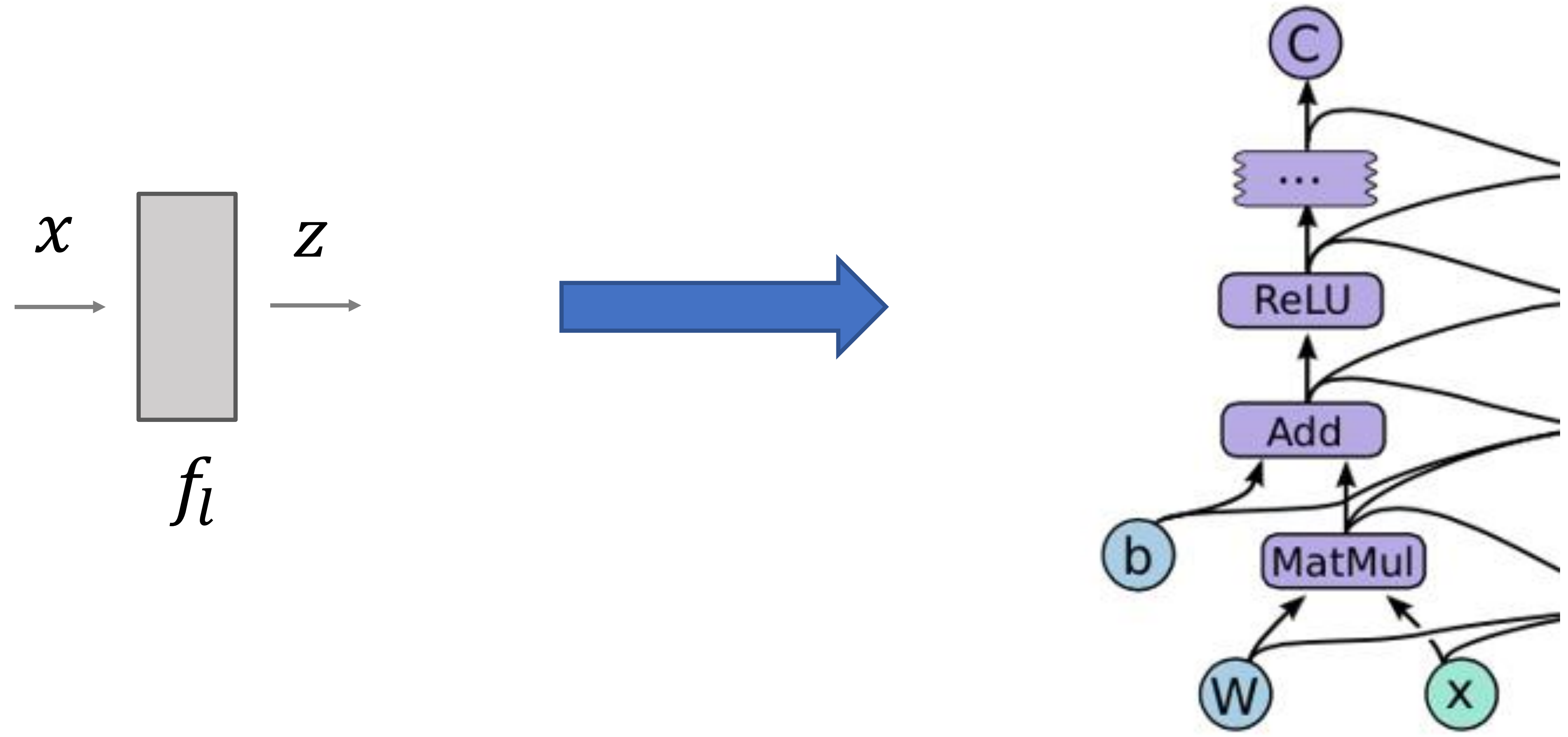
weight update
(sgd, adam, etc.)

model
(CNN, GPT, etc.)

data

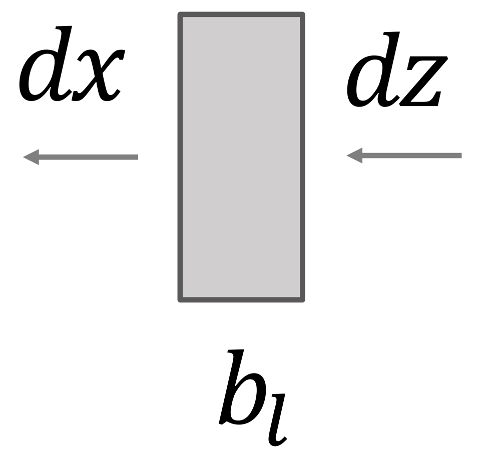
A Computational Layer in DL: forward

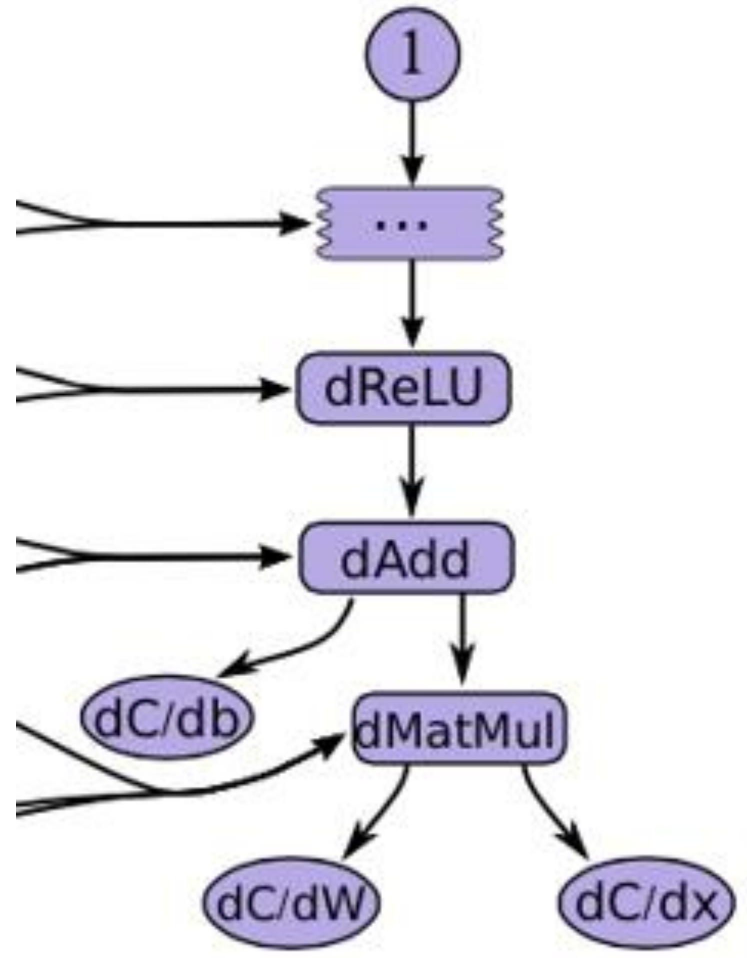
- A layer in a neural network is composed of a few finer computational operations
  - Consider: $\mathrm{z} = \mathrm{f_l}(\mathrm{x})$: $y = Wx + b, z = ReLU(y)$



A Computational Layer in DL: backward

- Denote the backward pass through a layer l as b_l
  - b_l derives the gradients of the input $x(\mathrm{d}x)$,given the gradient of z as $\mathrm{d}z$, as well as the gradients of the parameters W, b
  - $\mathrm{d}x$ will be the backward input of its previous layer $l - 1$
  - Backward pass can be thought as a backward dataflow where the gradient flow through the layer





A Layer as a Dataflow Graph

- Give the forward computation flow, gradients can be computed by auto differentiation
  - Automatically derive the backward gradient flow graph from the forward dataflow graph

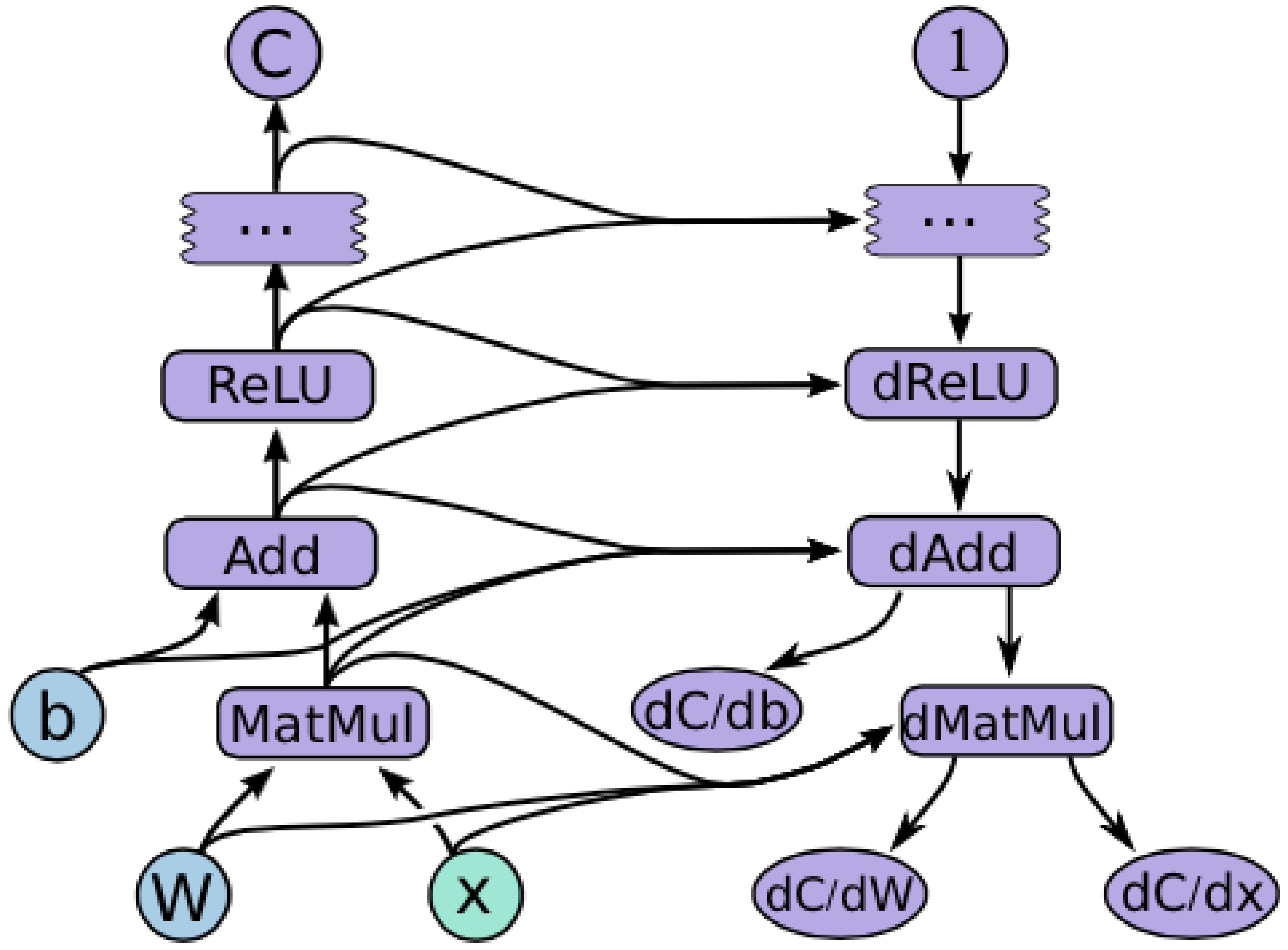
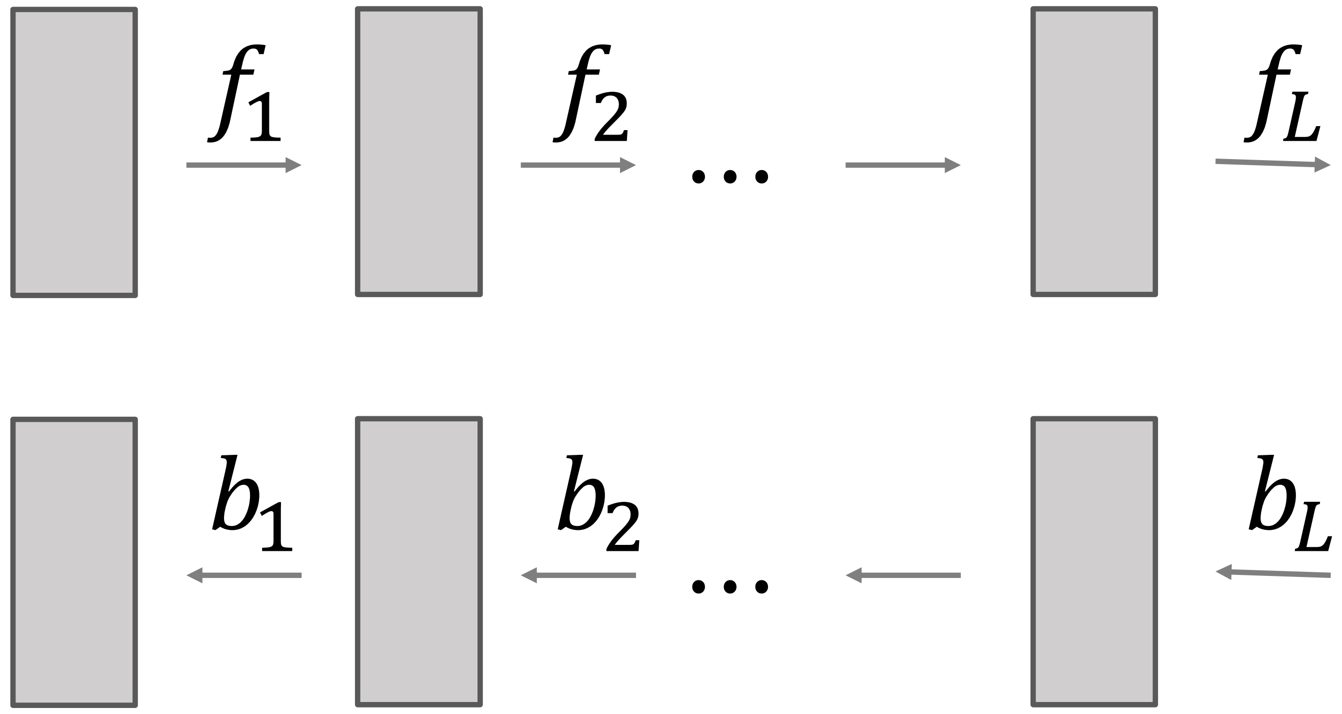


Photo from TensorFlow website

Combining Weight Update

- Gradients can be computed by auto differentiation
  - Automatically derive the gradient flow graph from the forward dataflow graph



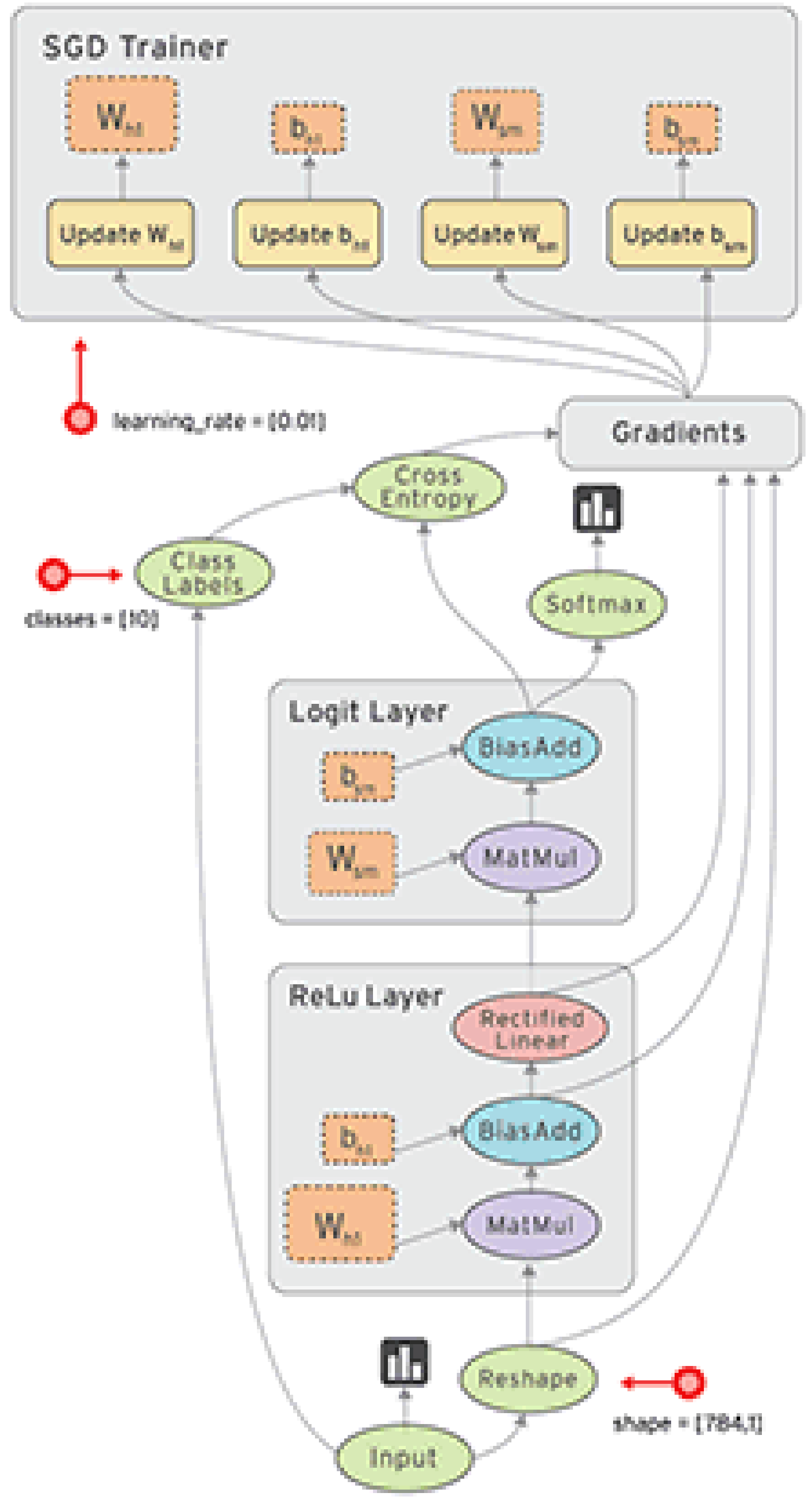


Photo from TensorFlow website

Practice

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

$$L = \mathrm{MSE}(w_2 \cdot \mathrm{ReLU}(w_1 x), y) \quad \theta = \{w_1, w_2\}, D = \{(x, y)\}$$

$$f(\theta, \nabla_L) = \theta - \nabla_L$$

Forward	Backward	Weight update
$L(\cdot)$	$\nabla_L(\cdot)$	$f(\cdot)$

Practice

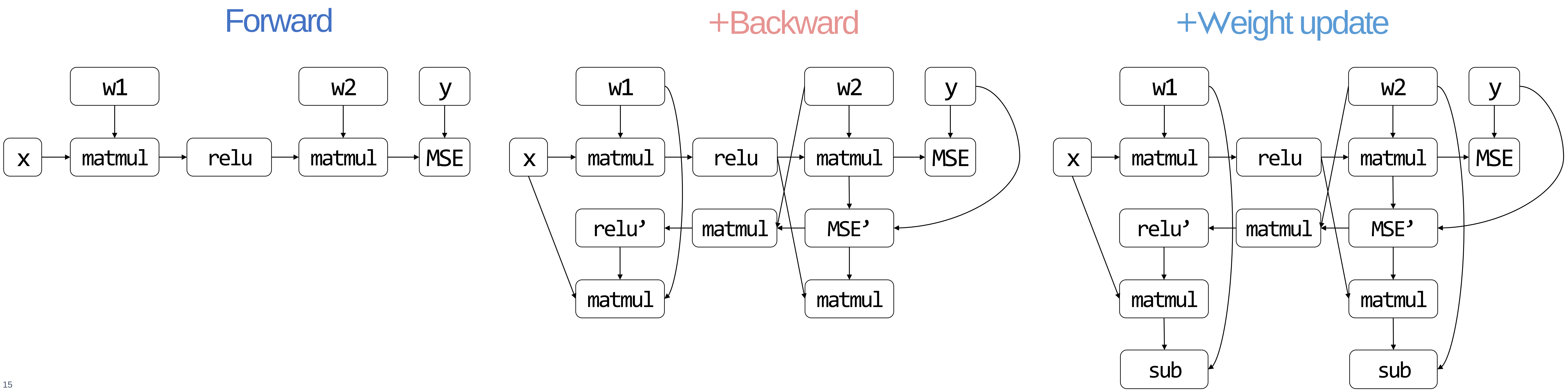
$$\theta^{(t+1)} = f\left(\theta^{(t)}, \nabla_L\left(\theta^{(t)}, D^{(t)}\right)\right)$$

$$L = \mathrm{MSE}(w_2 \cdot \mathrm{ReLU}(w_1 x), y) \quad \theta = \{w_1, w_2\}, D = \{(x, y)\}$$

$$f(\theta, \nabla_L) = \theta - \nabla_L$$

Operator / its output tensor

Data flowing direction



Dataflow Graph Programming Model Today

- Define a neural network
  - Define operations and layers: fully-connected? Convolution? Recurrent?
  - Define the data I/O: read what data from where?
  - Define a loss function/optimization objective: L2 loss? Softmax? Ranking Loss?
  - Define an optimization algorithm: SGD? Momentum SGD? etc
- Auto-differential Libraries will then take over
  - Connect operations, data I/O, loss functions and trainer.
  - Build forward dataflow graph and backward gradient flow graphs.
  - Perform training and apply updates

Discussion:
Compare this vs. Spark, parameter server, MapReduce?

Outline

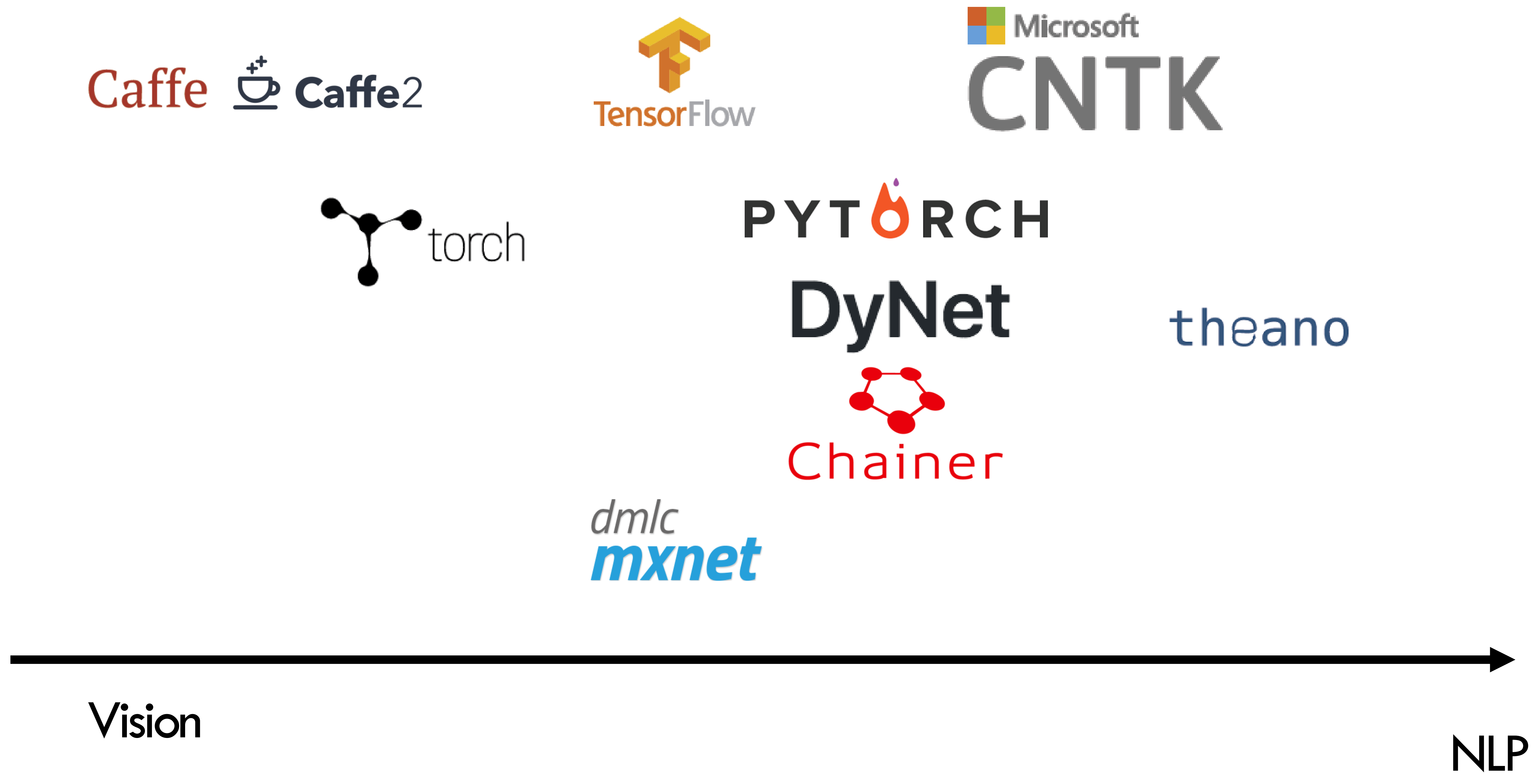
- Deep Learning as Dataflow Graphs
- **Auto-differentiable Libraries**
  - Symbolic vs. Imperative
  - Static vs. Dynamic
- DL Parallelism

Auto-differential Libraries

- Auto-differential Library automatically derives the gradients following the back-propagation rule.
- A lot of auto-differentiation libraries have been developed:

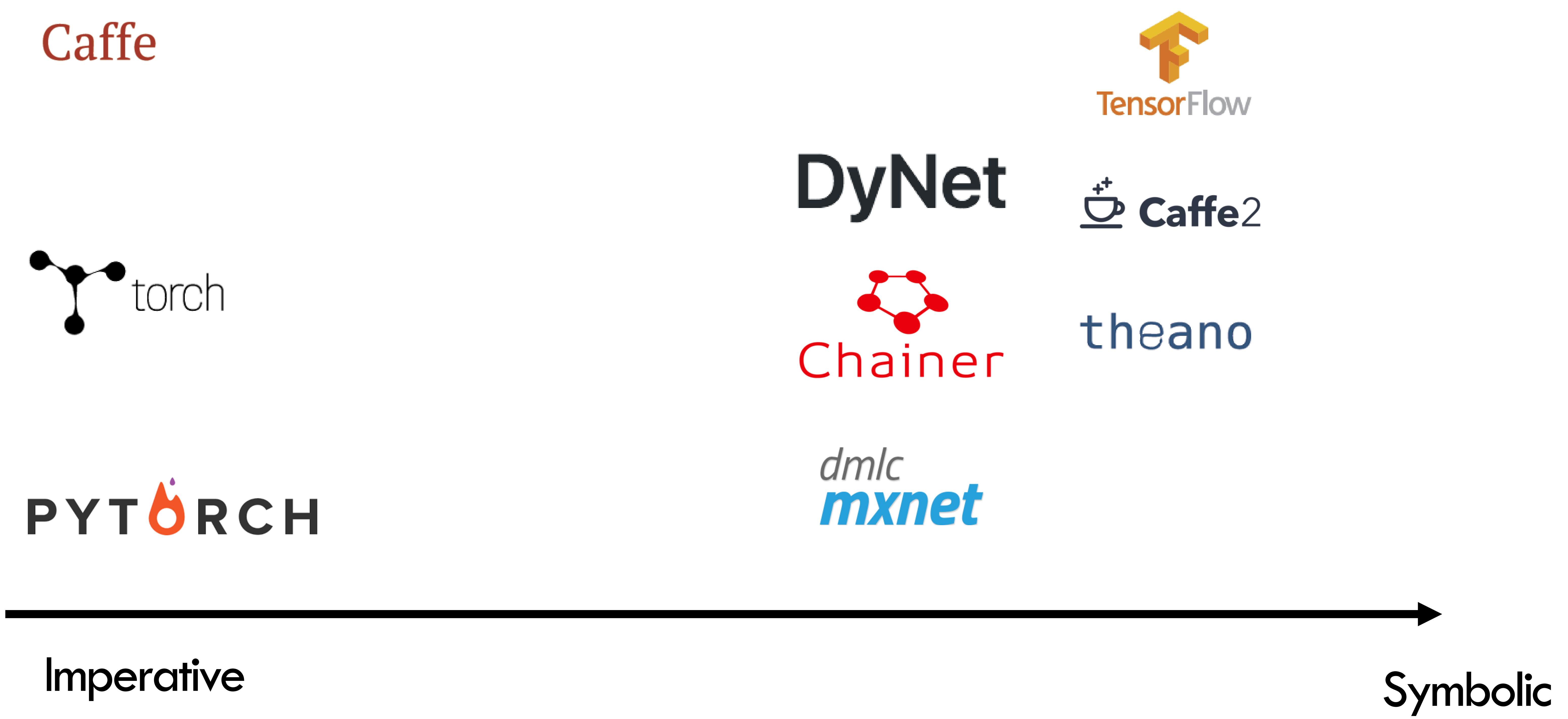
Deep Learning Toolkits

- They are roughly adopted by different domains



Symbolic vs. Imperative

- They are also designed differently
  - Symbolic v.s. imperative programming



Symbolic vs. Imperative

- Symbolic vs. imperative programming
  - Symbolic: write symbols to assemble the networks first, evaluate later
  - Imperative: immediate evaluation

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + Constant(1)
# compiles the function
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```

Symbolic

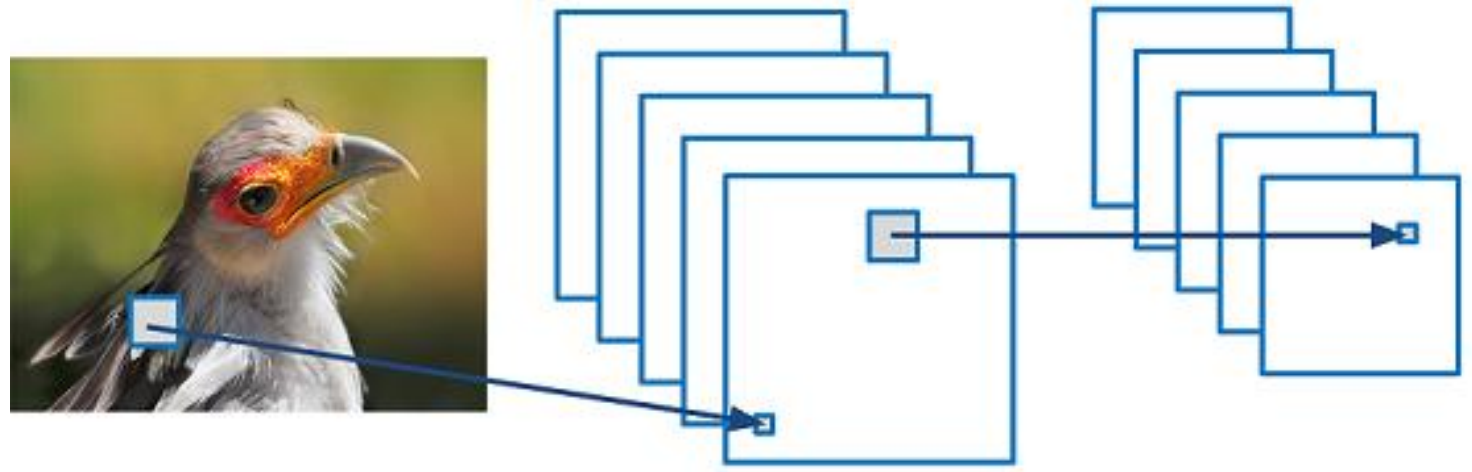
```
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

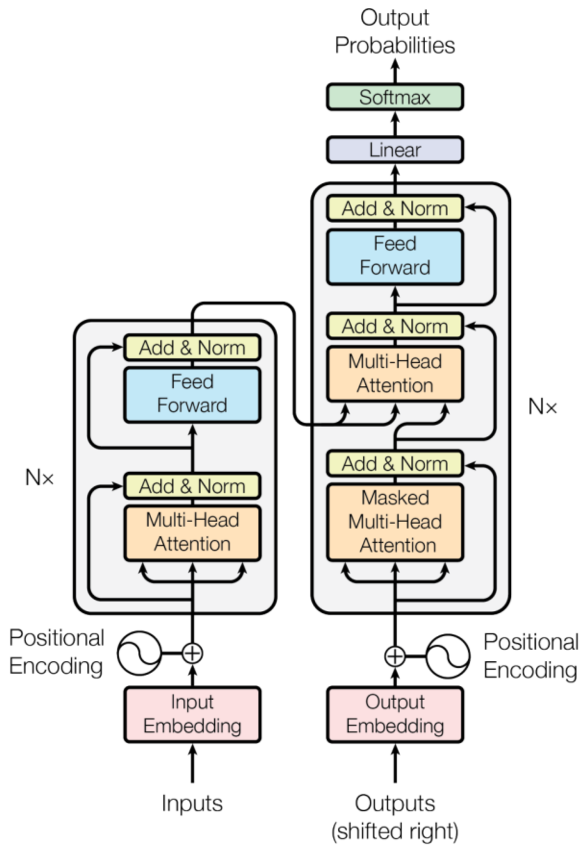
Imperative

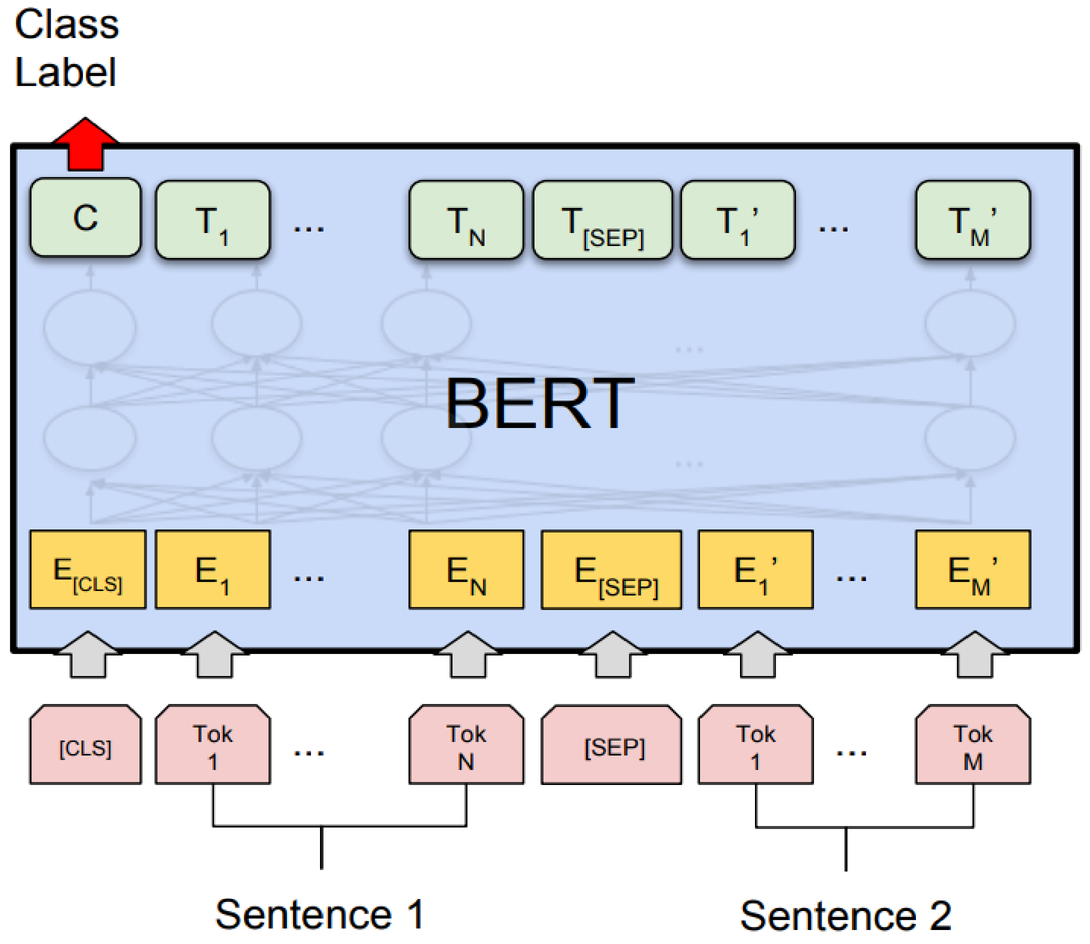
Symbolic vs. Imperative

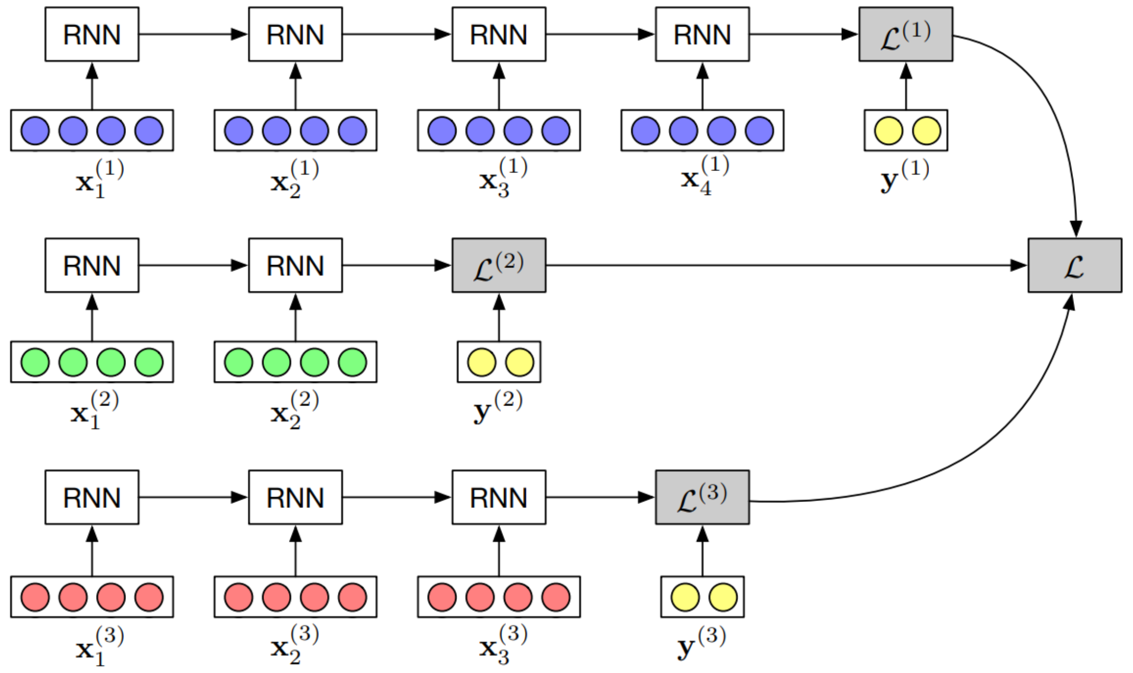
- Symbolic
  - Good
    - easy to optimize (e.g. distributed, batching, parallelization) for developers
    - More efficient
  - Bad
    - The way of programming might be counter-intuitive
    - Hard to debug for user programs
    - Less flexible: you need to write symbols before actually doing anything
- Imperative:
  - Good
    - More flexible: write one line, evaluate one line (that's why we all like Python)
    - Easy to program and easy to debug: because it matches the way we use C++ or python
  - Bad
    - Less efficient
    - More difficult to optimize

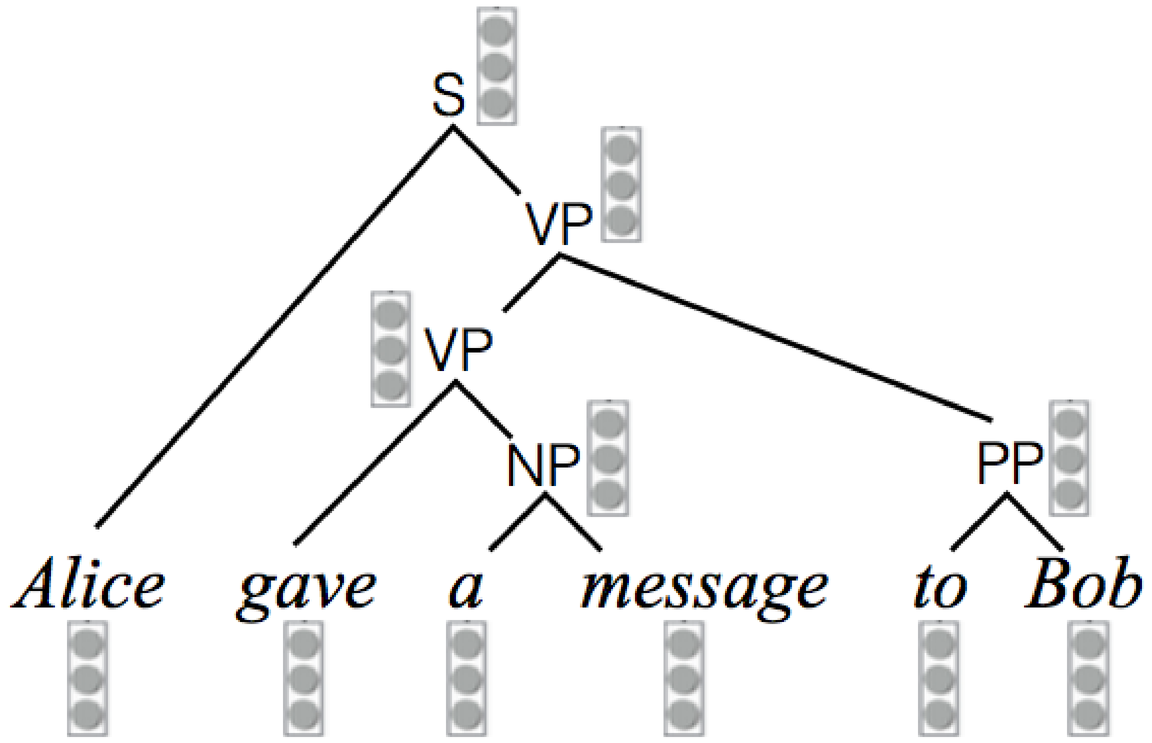
Are All Models expressive in Dataflow Graph?

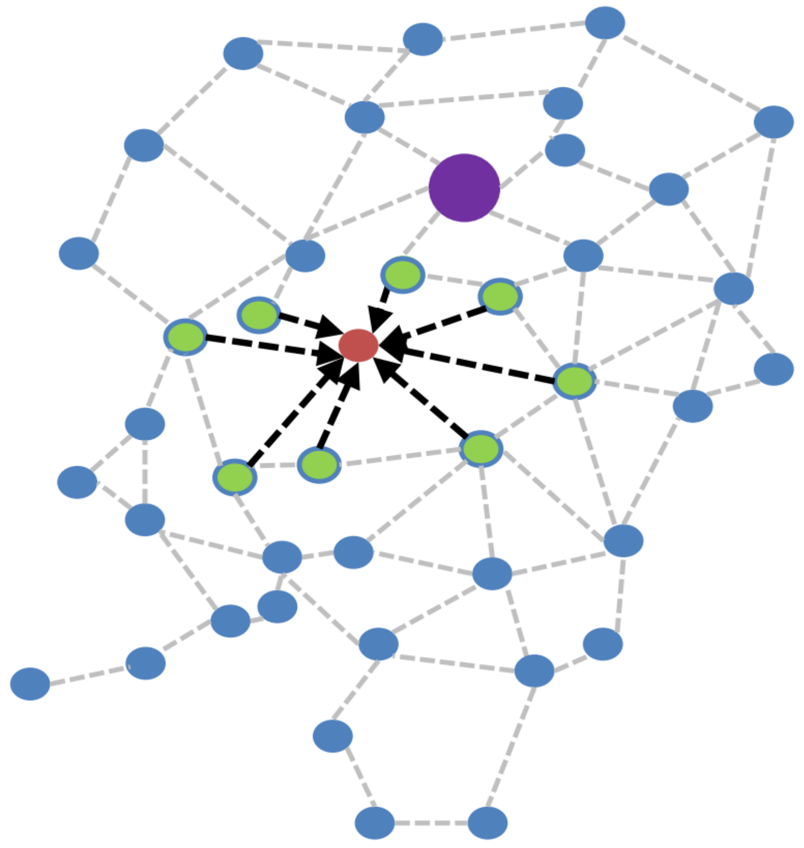




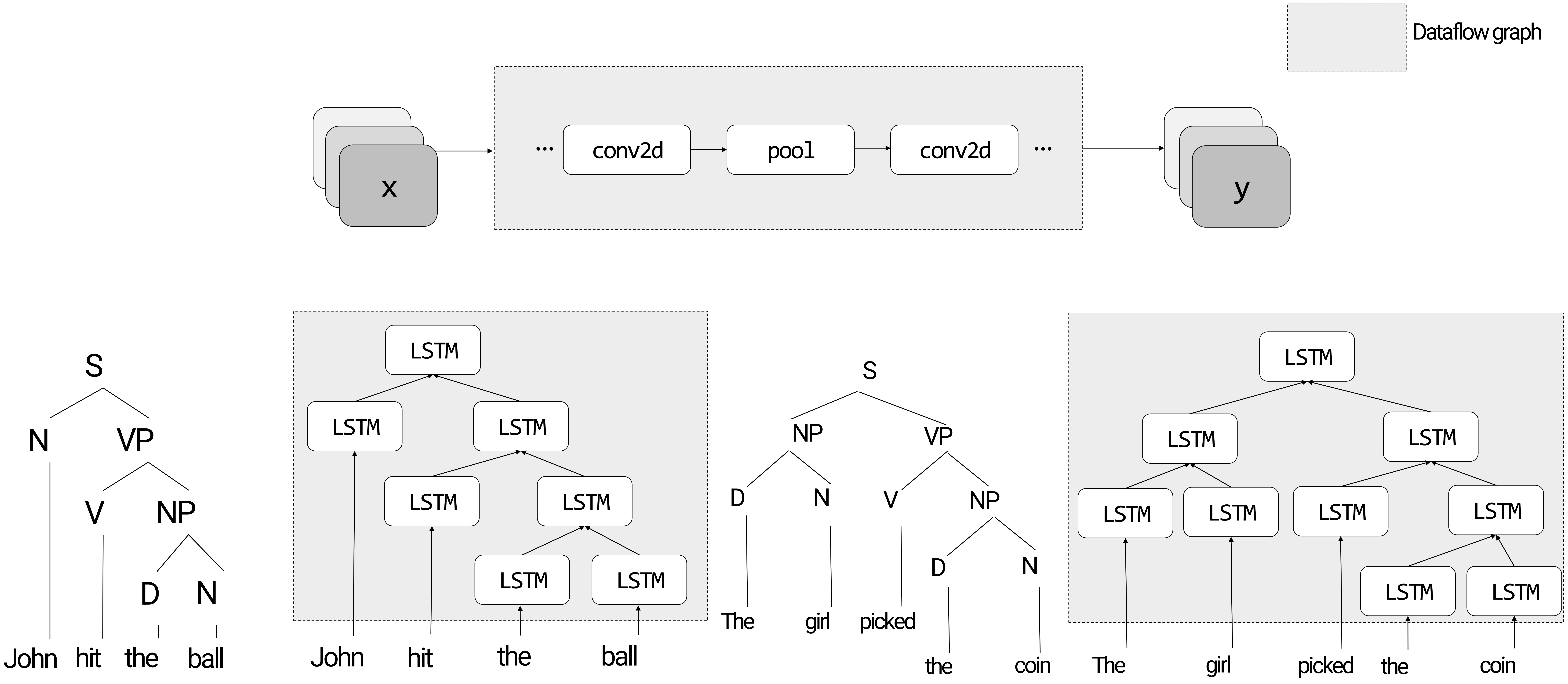








Static Models vs. Dynamic Models



Static vs. Dynamic Dataflow Graphs

- Static Dataflow graphs
  - Define once, execute many times
    - For example: convolutional neural networks
  - 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

- Dynamic Dataflow graphs
  - When do we need?
    - In all cases that static dataflow graphs do not work well
  - Variably sized inputs
  - Variably structured inputs
  - Nontrivial inference algorithms
  - Variably structured outputs
  - Etc.

Static vs. Dynamic Dataflow Graphs

- Can we handle dynamic dataflow graphs? Using static methods (or declaration) will have a lot of problems
  - Difficulty in expressing complex flow-control logic
  - Complexity of the computation graph implementation
  - Difficulty in debugging

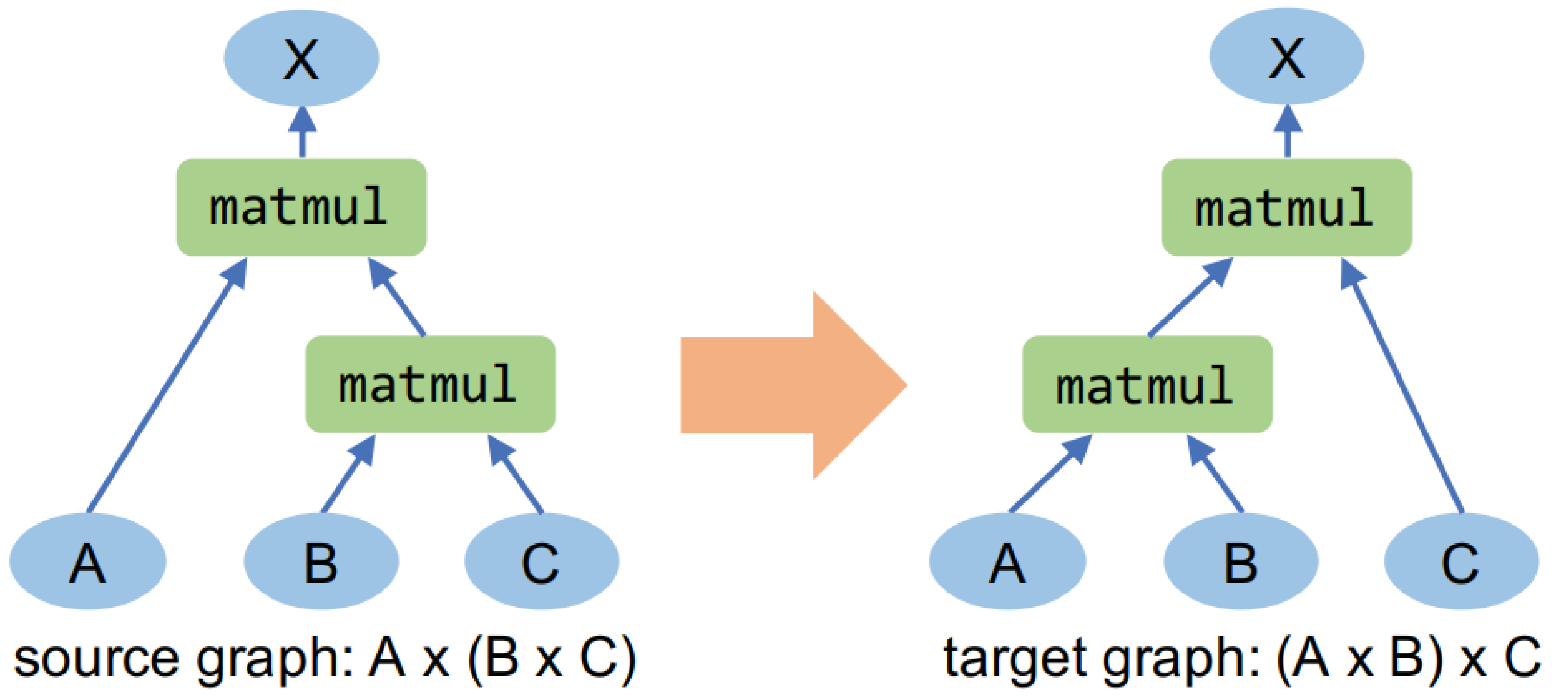
Questions

- 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

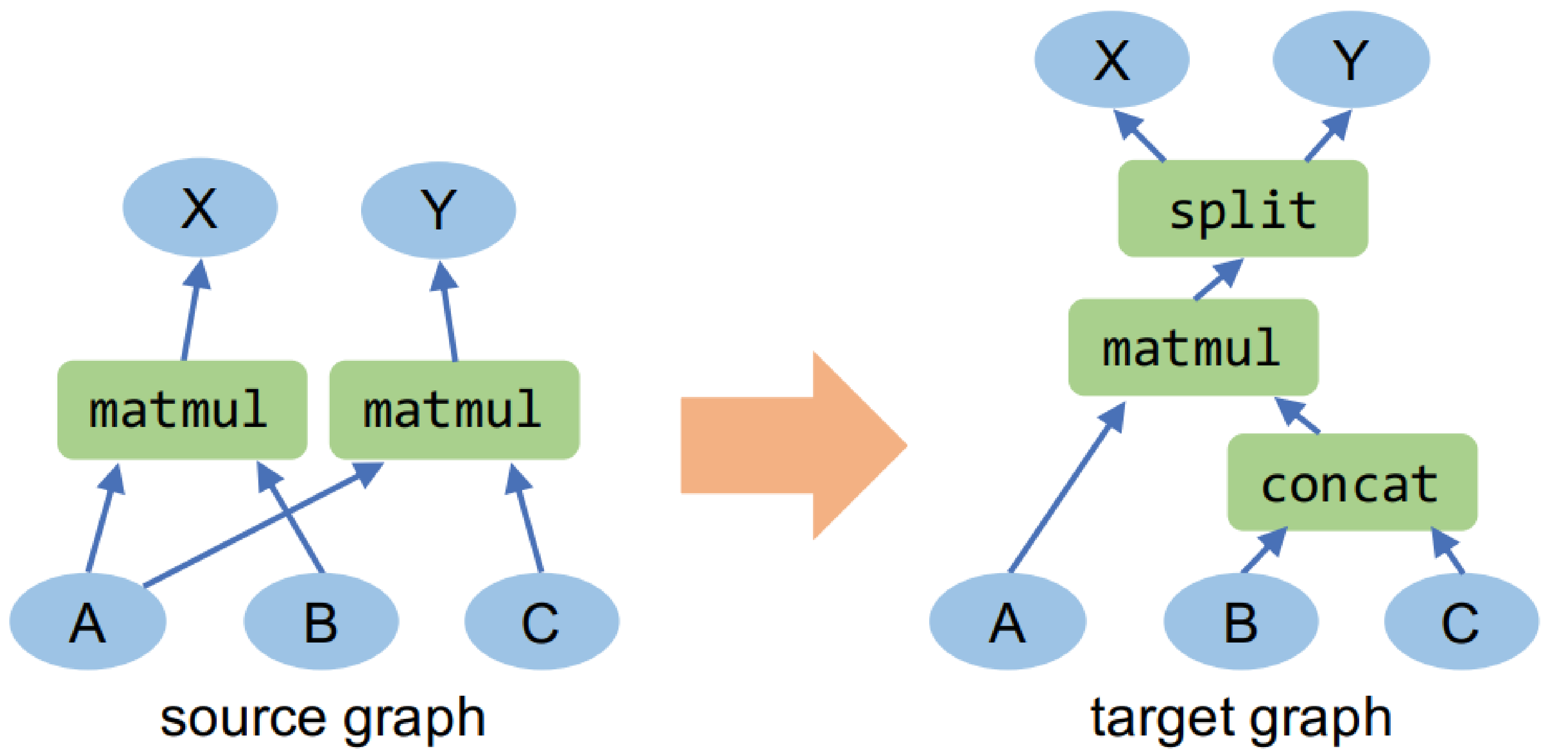
How to Handle Dynamic Dataflow Graph?

- In general two ways:
  - Imperative: do not requiring contracting the entire graph before execution
  - Use other representation: vertex-centric representation

Advanced Topic: DL Dataflow Graph Optimization



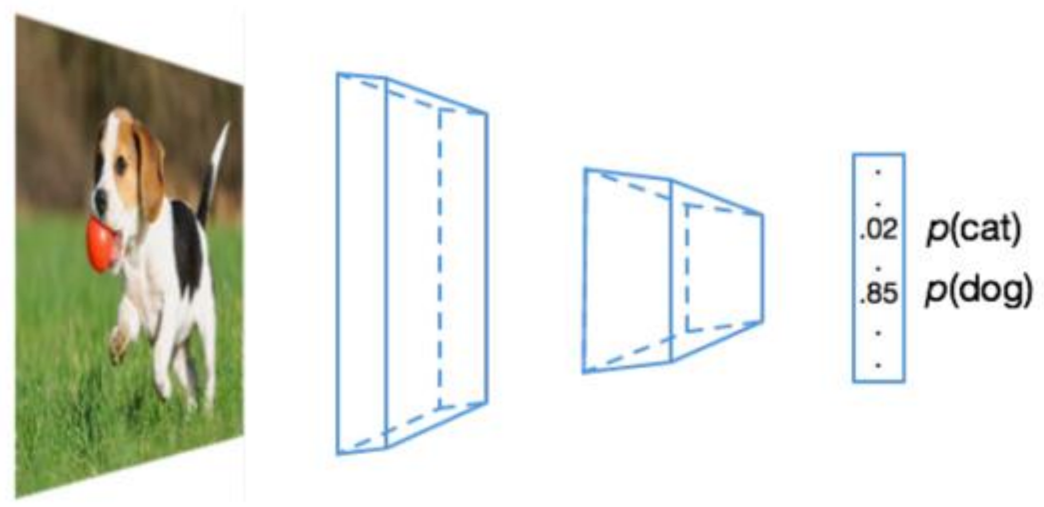
(a) Associativity of matrix multiplication.



(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

Dataflow Graph Summary

- Dataflow graphs seems to be a dominant choice for representing deep learning models
  - What’s good for dataflow graphs
    - Good for **static** workflows: define once, run for arbitrary batches/data
    - Programming convenience: **easy to program once you get used to it**.
    - Easy to parallelize/batching for a fixed graph
    - Easy to optimize: a lot of off-the-shelf optimization techniques for graph
  - What‘s bad for dataflow graphs
    - Not good for **dynamic** workflows: need to define a graph for every training sample -> overheads
    - Hard to program dynamic neural networks: how can you define dynamic graphs using a language for static graphs? (e.g. LSTM, tree-LSTM).
    - Not easy for debugging.
    - Difficult to parallelize/batching across multiple graphs: every graph is different, no natural batching.

Where We Are

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

Recap: DL Computation

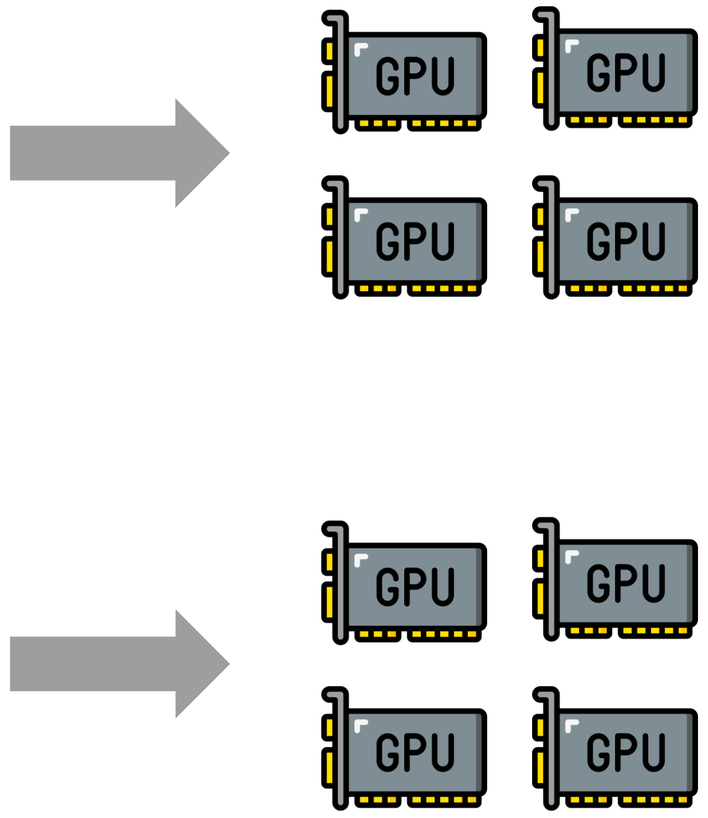
$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update} \\ \text{(sgd, adam, etc.)}}}\Big(\theta^{(t)},\ \underbrace{\nabla_L}_{\substack{\text{model} \\ \text{(CNN, GPT, etc.)}}}\big(\theta^{(t)},\ \underbrace{D^{(t)}}_{\text{data}}\big)\Big)$$

DL Parallelization: 3 Core Problems

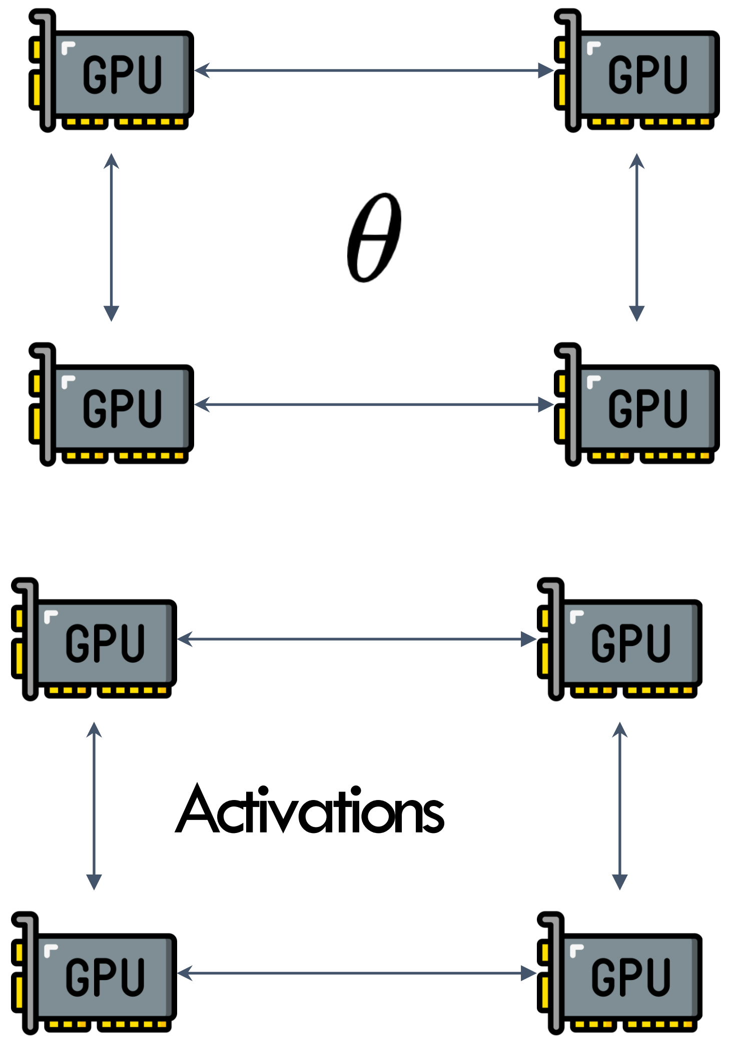
Computing

$\nabla_L(\cdot)$

$f(\cdot)$



Communication

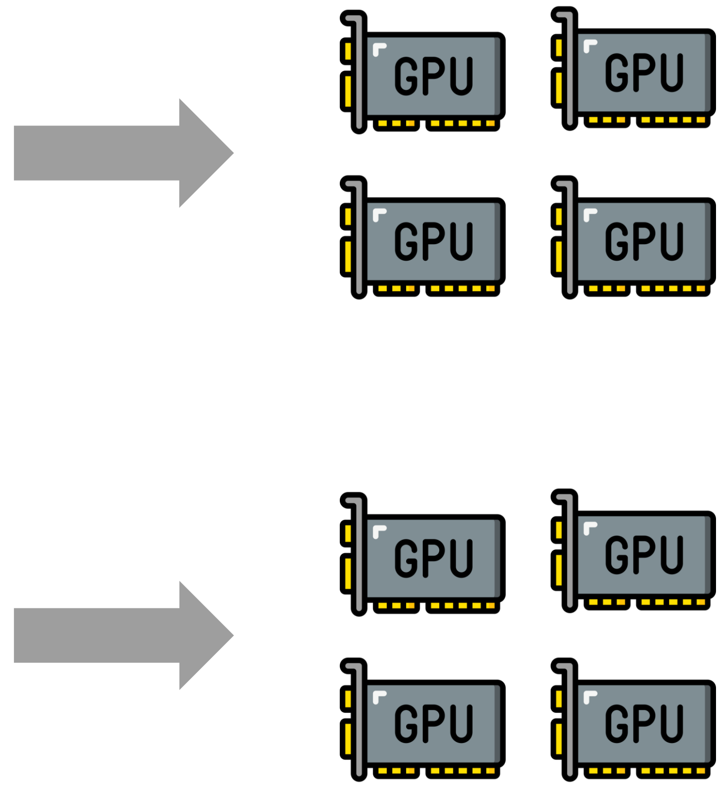


...

Memory

D

θ



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

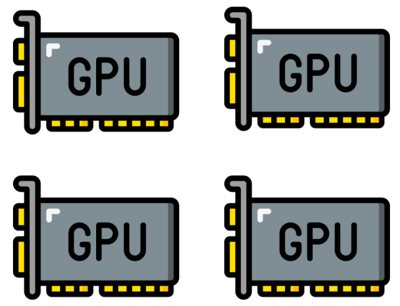
Two Views of ML Parallelisms

Classic view	**New view**
Data parallelism	Inter-op parallelism
Model parallelism	Intra-op parallelism

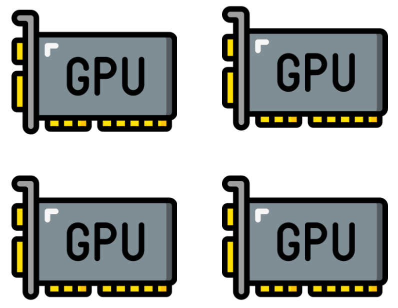
Data and Model Parallelism

Data parallelism

$D \xrightarrow{\text{partition}}$

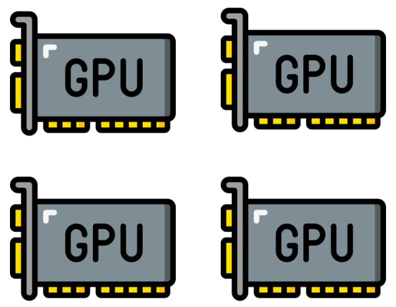


$\theta,\ \nabla_L(\cdot),\ f(\cdot) \xrightarrow{\text{replicate}}$

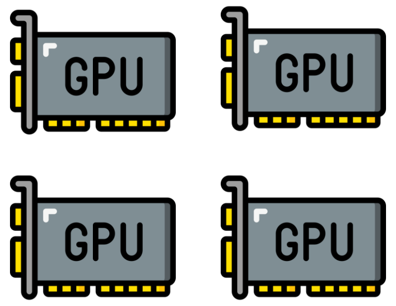


Model parallelism

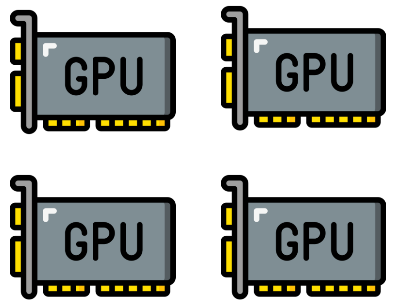
$\theta,\ \nabla_L(\cdot) \xrightarrow{\text{partition}}$



$D \xrightarrow{?}$

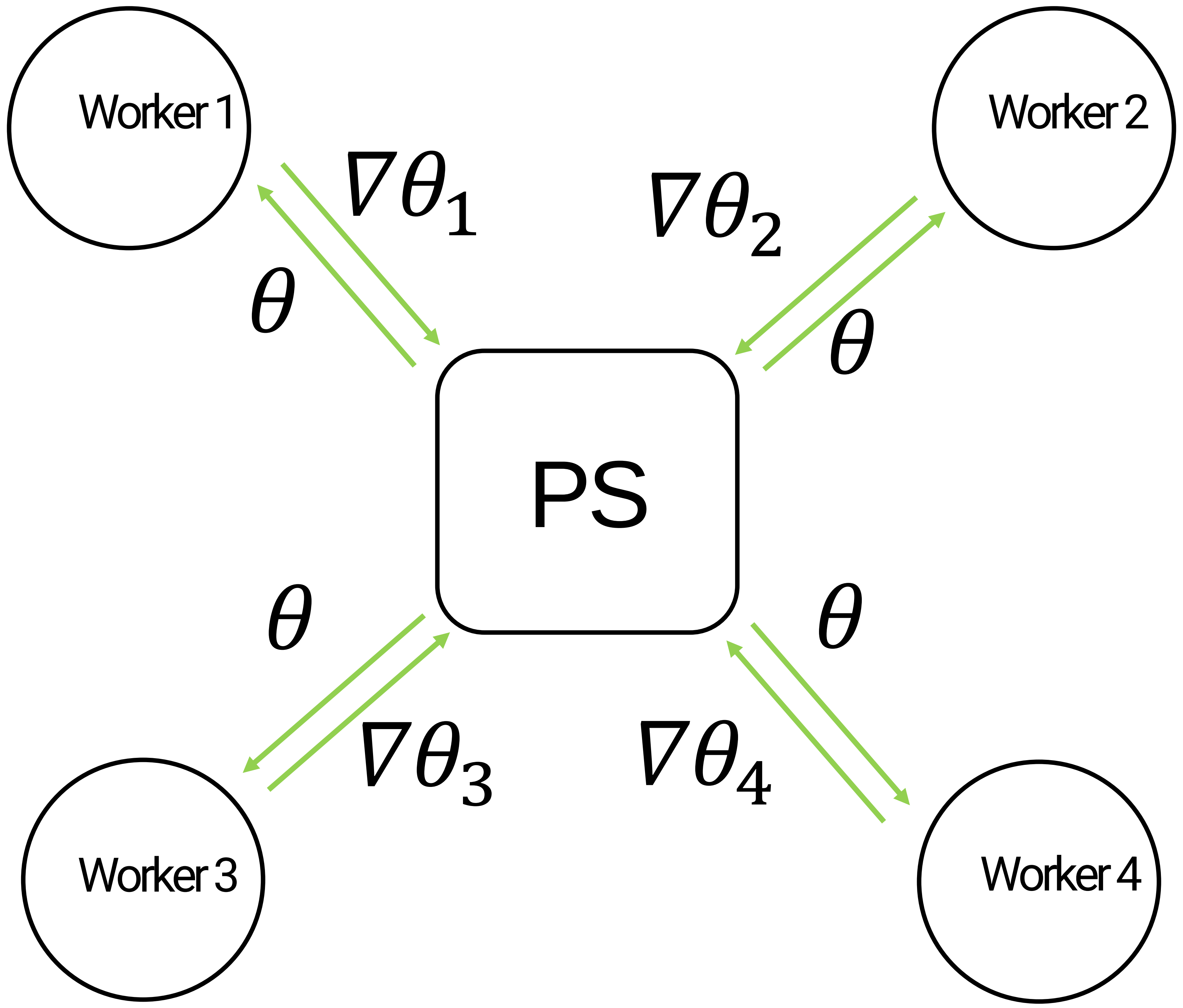


$f(\cdot) \xrightarrow{?}$



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

PS Implements Data Parallelism



In total P workers

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

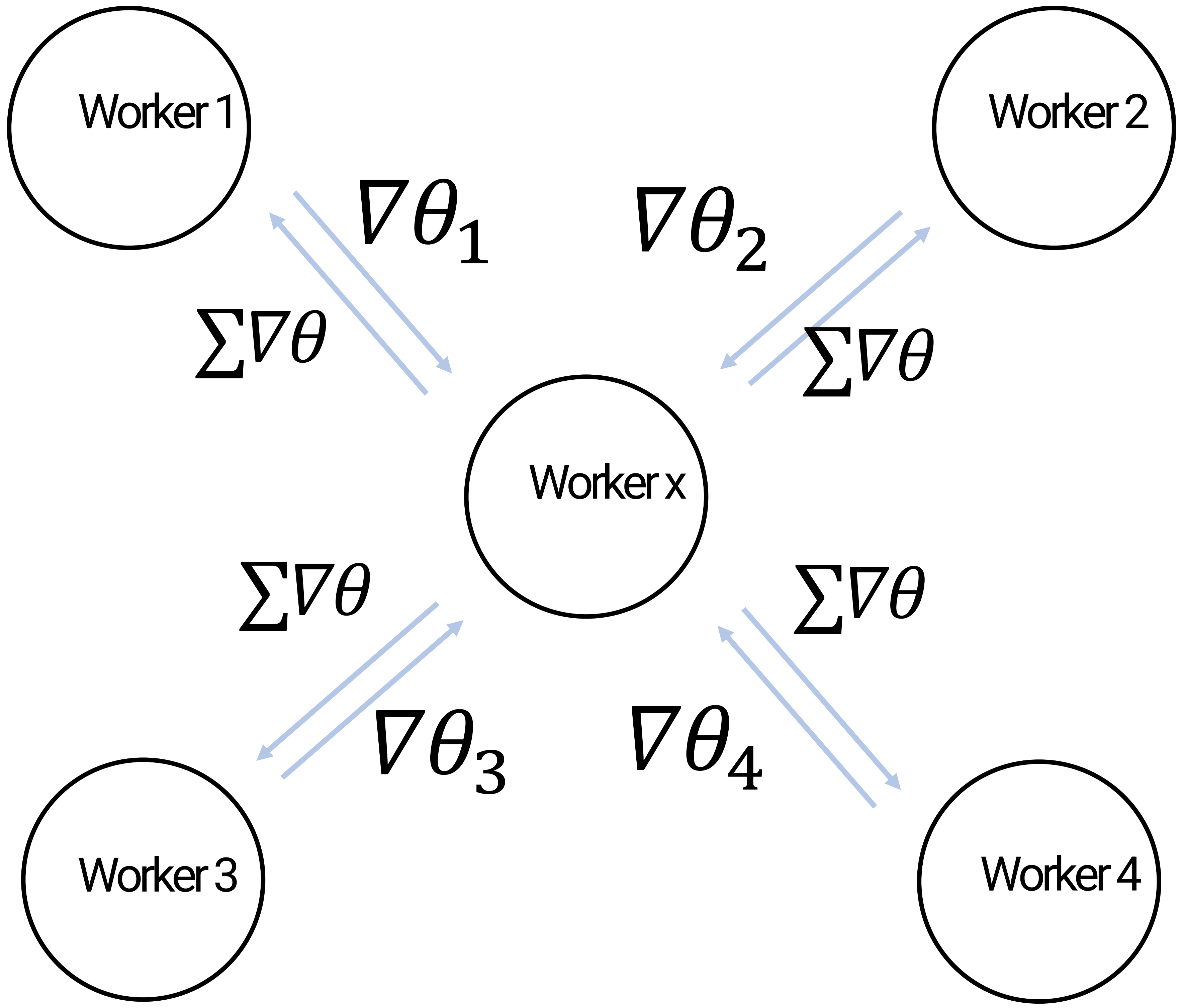
PS collects, aggregates, and applies the gradients, and then broadcast the parameters back to workers

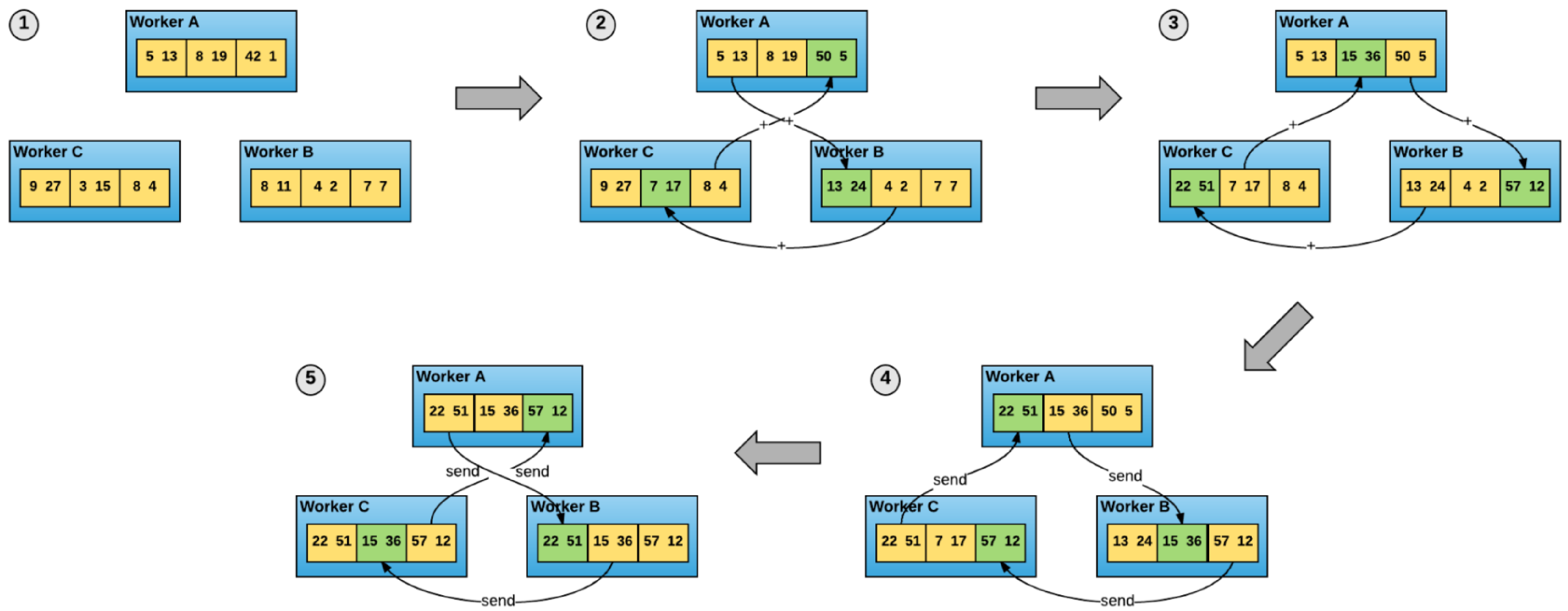
Happening locally on each worker

Data partition p

Representee Systems: Poseidon, GeePS, BytePS, etc.

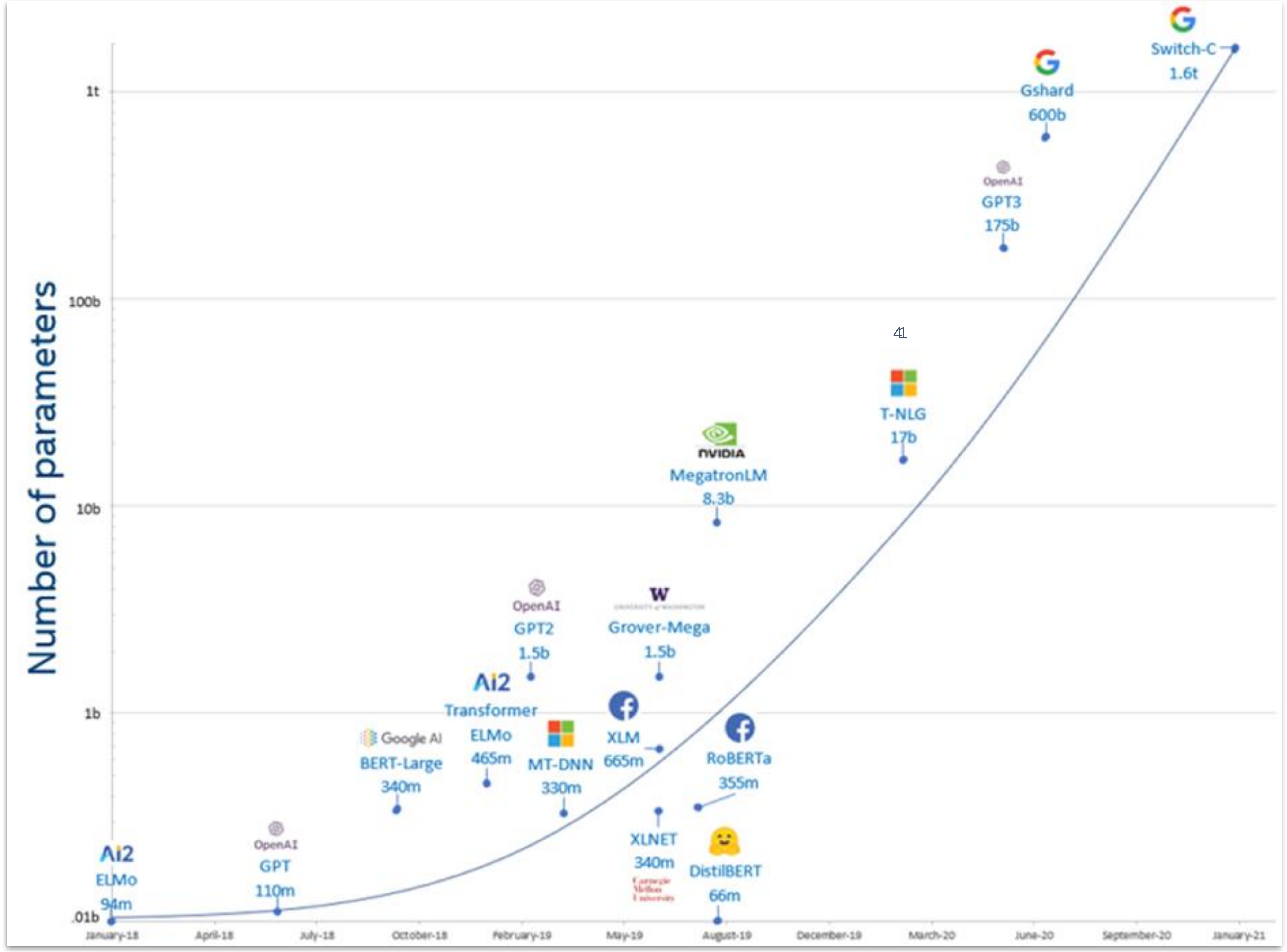
AllReduce Can Also Handle Data Parallelism Comm

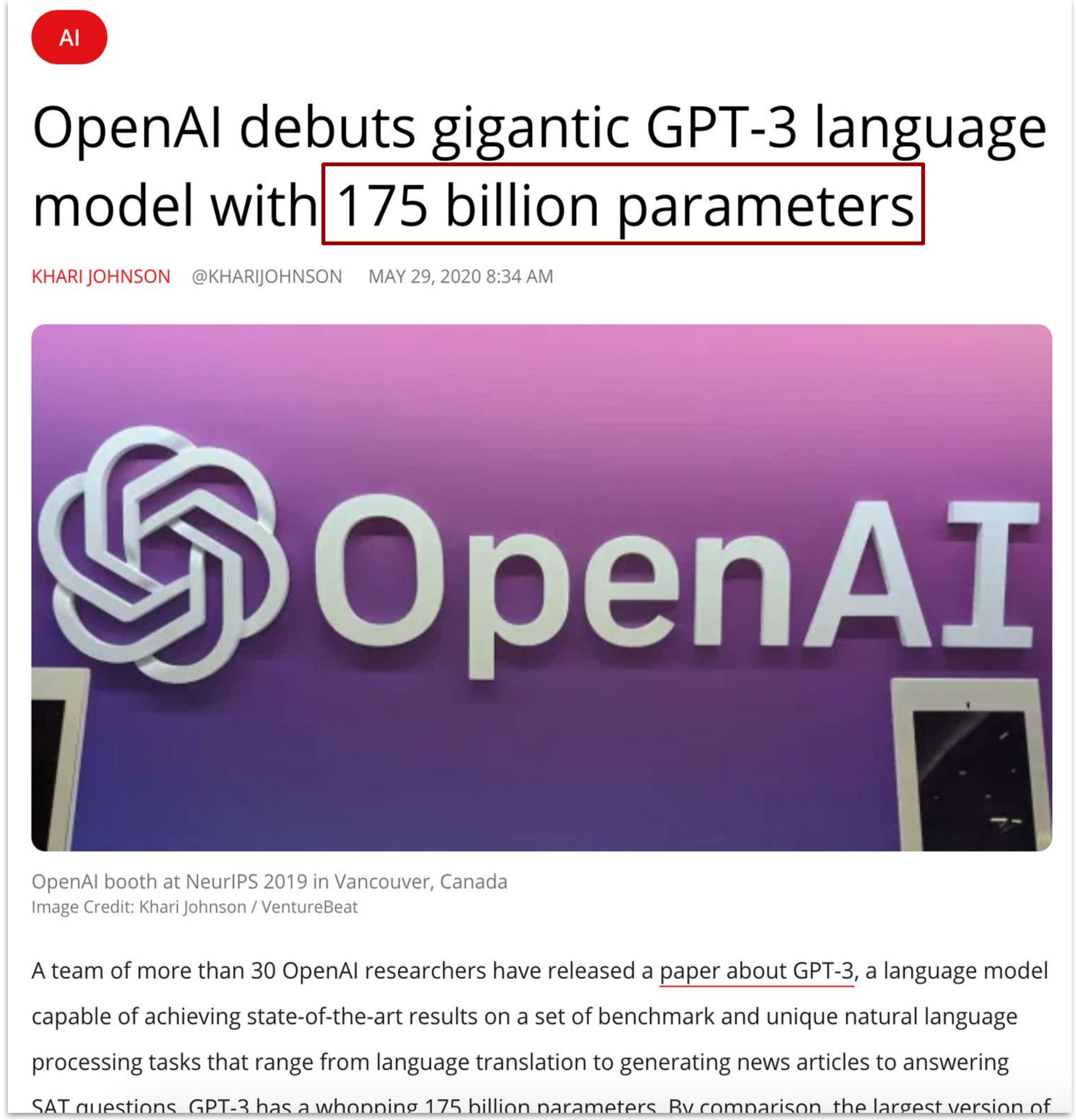




Representee Systems: Horovod, Torch.DDP

Big Models Become Prominent




AI

OpenAI debuts gigantic GPT-3 language model with 175 billion parameters

KHARI JOHNSON @KHARIJOHNSON MAY 29, 2020 8:34 AM

OpenAI booth at NeurIPS 2019 in Vancouver, Canada
Image Credit: Khari Johnson / VentureBeat

A team of more than 30 OpenAI researchers have released a paper about GPT-3, a language model capable of achieving state-of-the-art results on a set of benchmark and unique natural language processing tasks that range from language translation to generating news articles to answering SAT questions. GPT-3 has a whopping 175 billion parameters. By comparison, the largest version of

RESEARCH

Democratizing access to large-scale language models with OPT-175B

May 3, 2022

a BigScience initiative

BLOOM

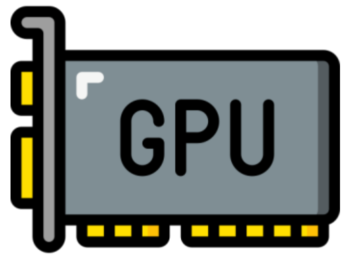
176B params · 59 languages · Open-access

Image source: towardsdatascience.com

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

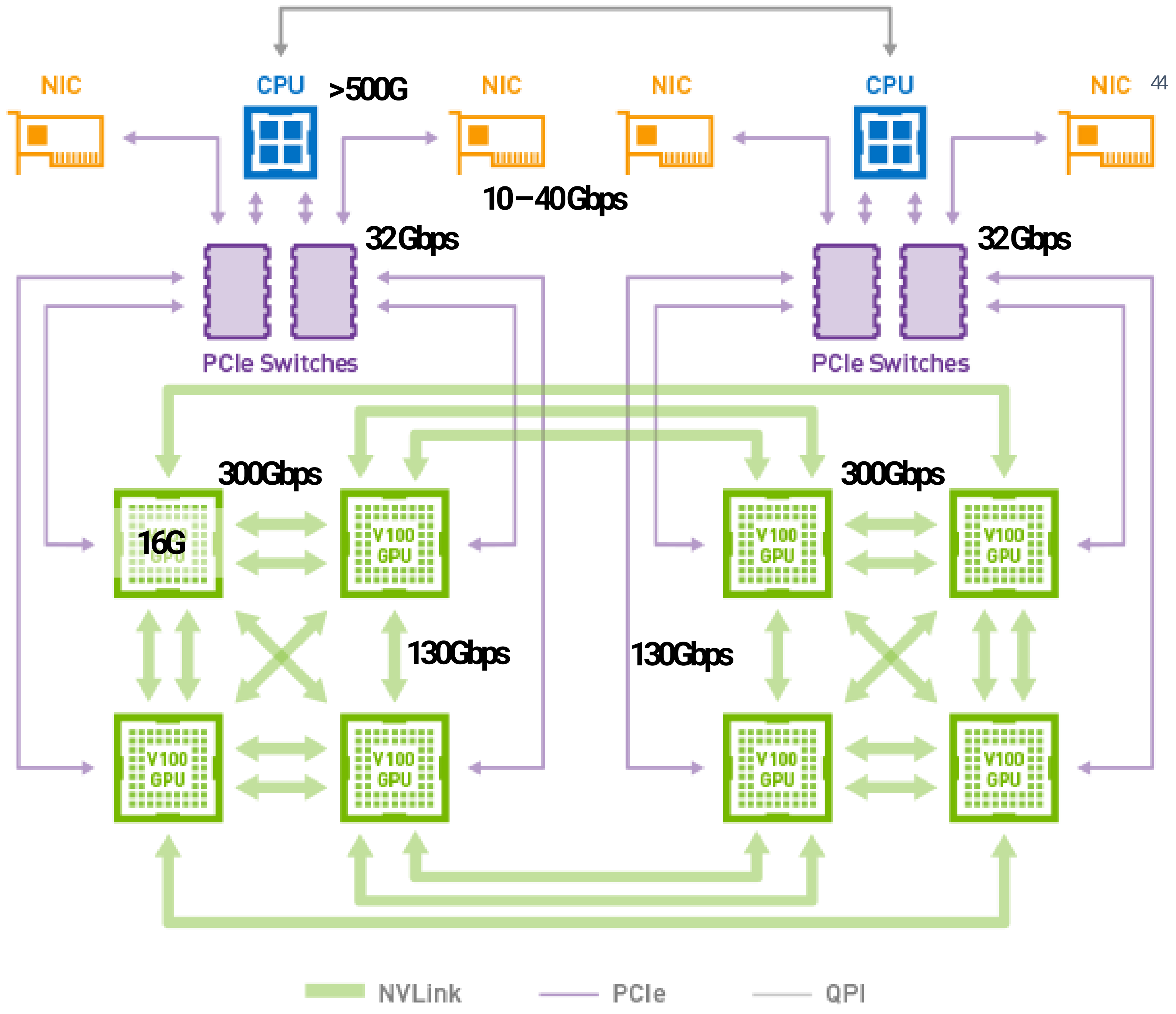
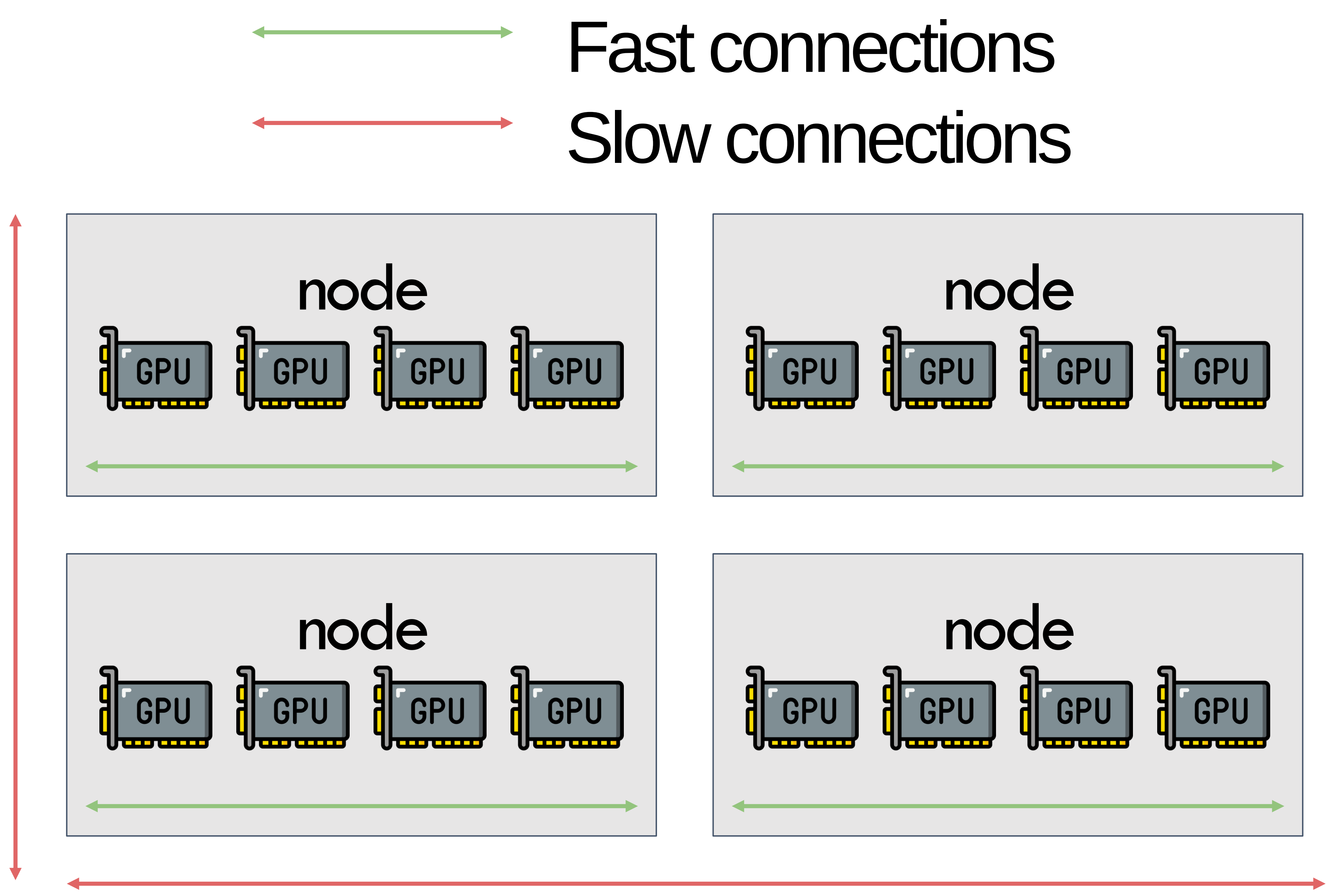


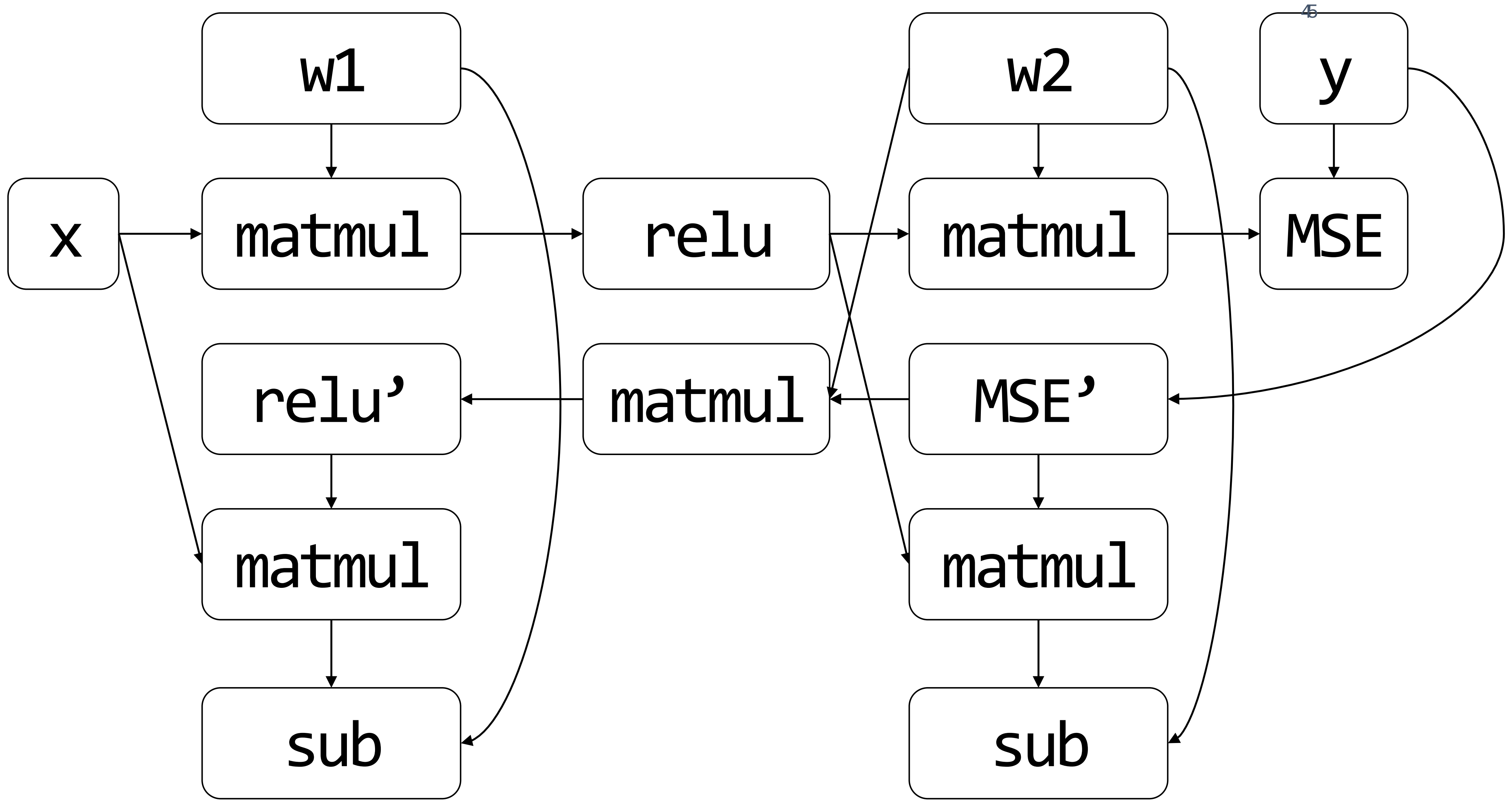
Figure from NVIDIA

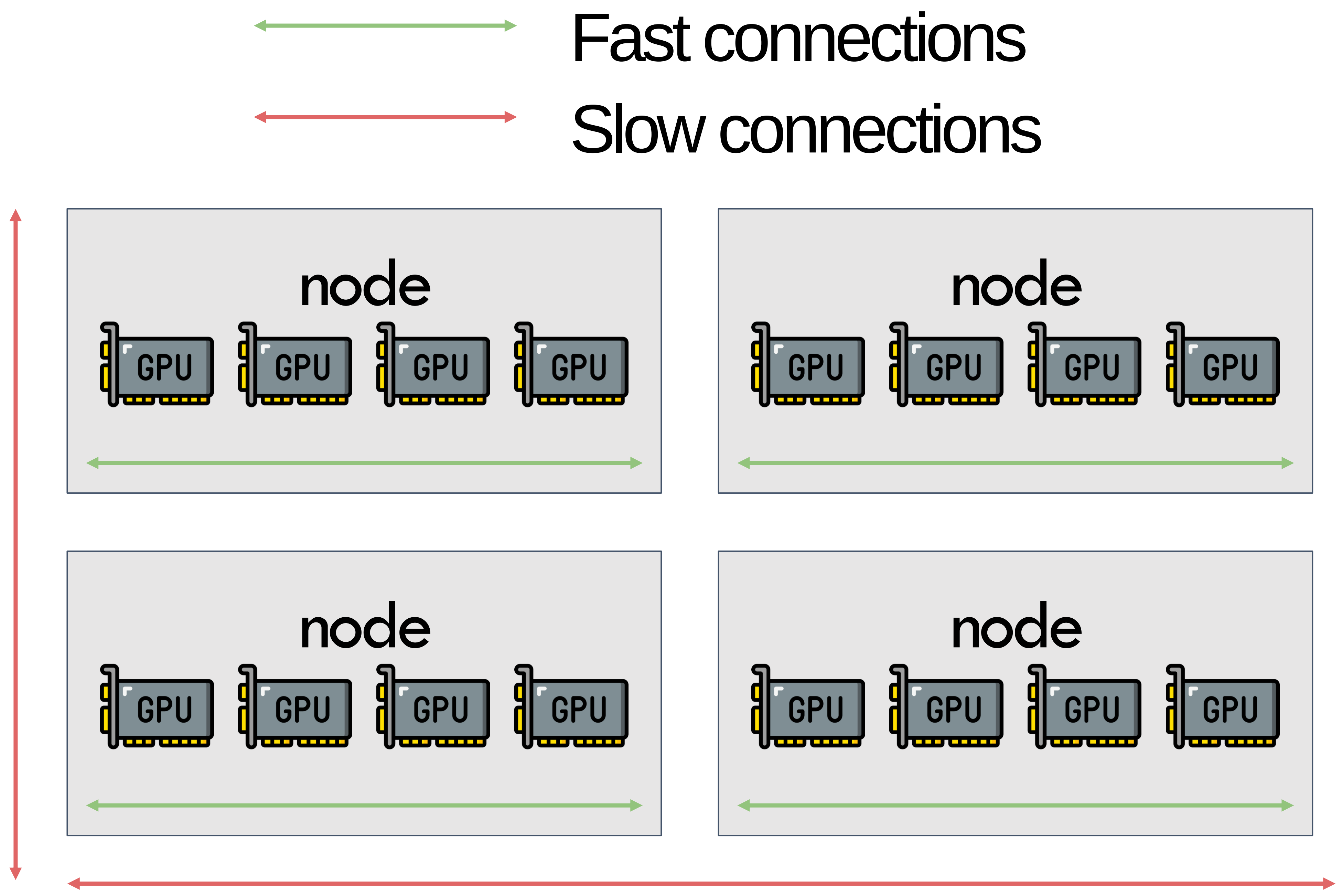
A typical GPU cluster topology



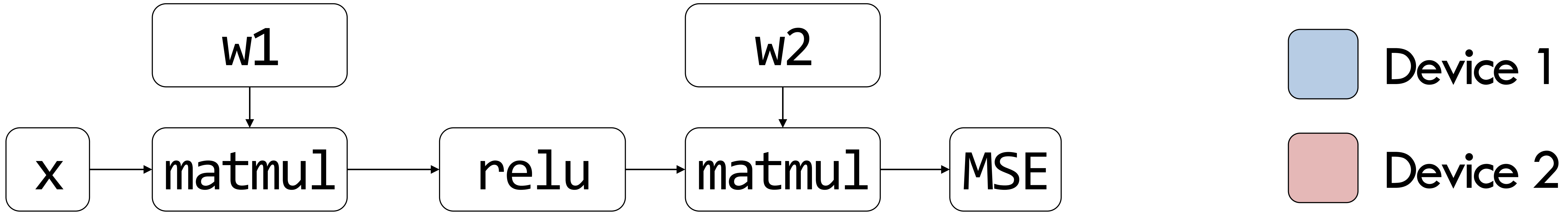
Partitioning Computation Graph on Device Cluster

How to partition the computational graph on the device cluster?

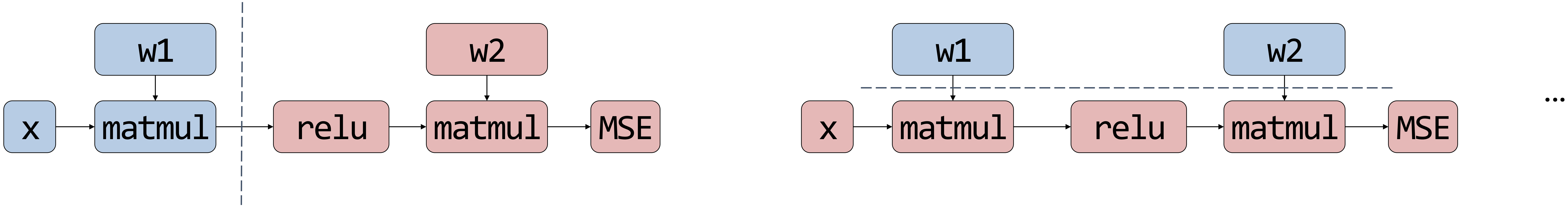




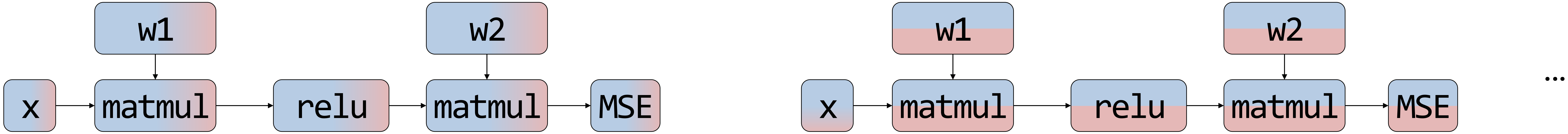
Partitioning Computation Graph



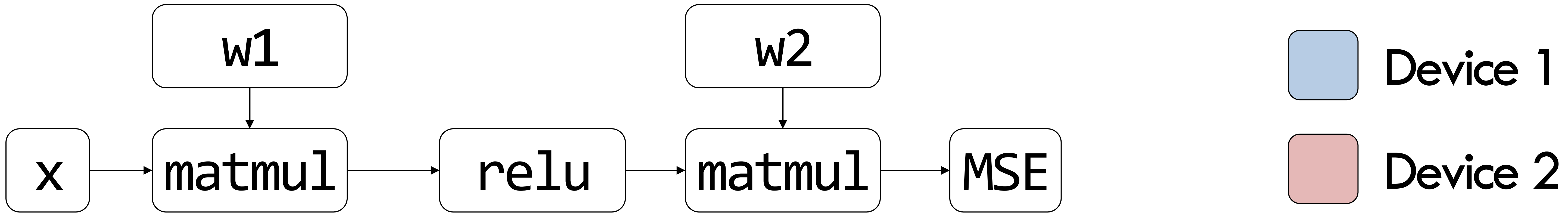
Strategy 1



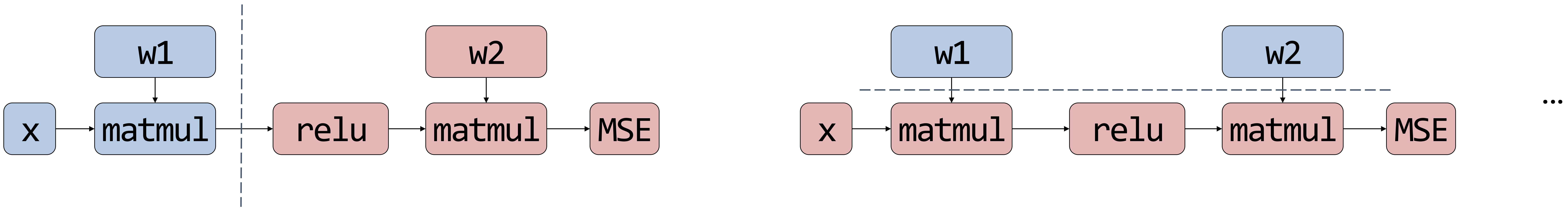
Strategy 2



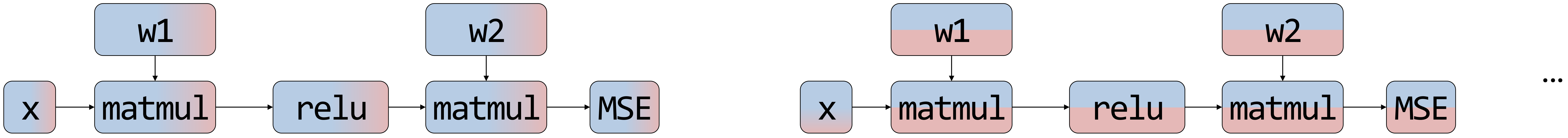
Partitioning Computation Graph



Strategy 1: Inter-operator Parallelism



Strategy 2: Intra-operator Parallelism

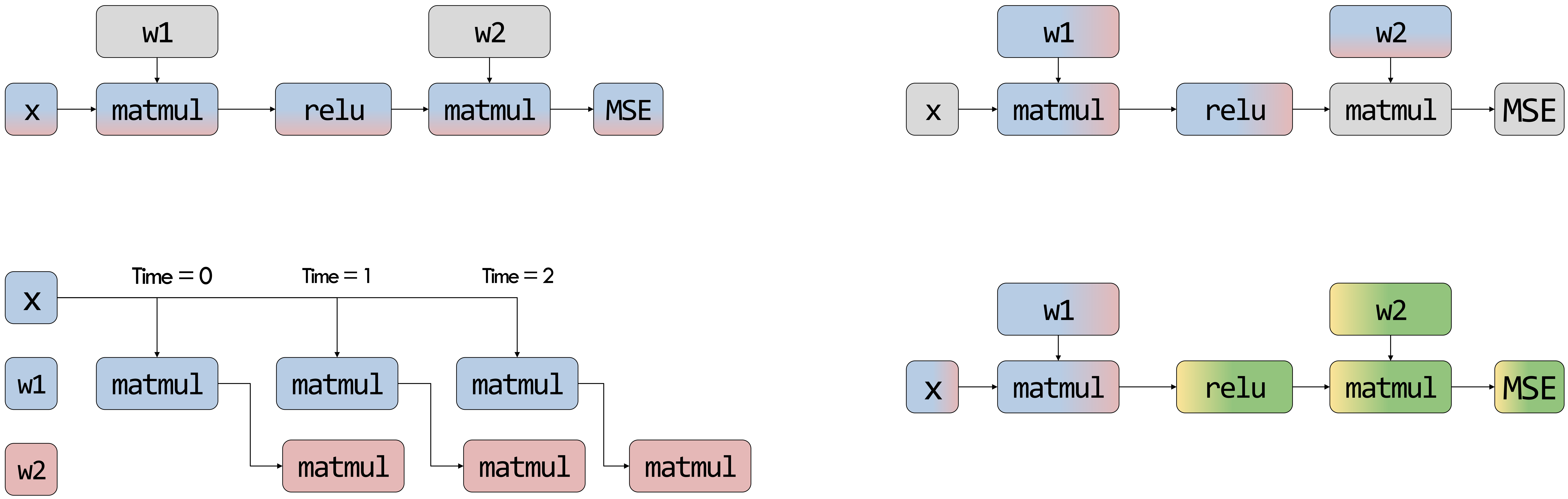


More Parallelisms…

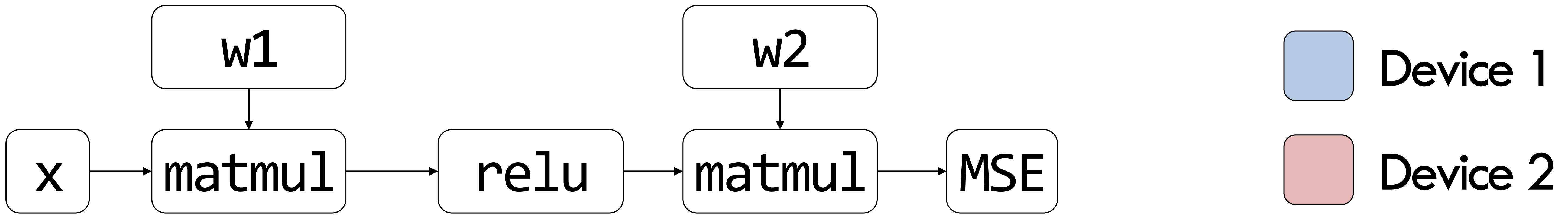
Multiple intra-op strategies for a single node

Row-partitioned   Column-partitioned   Replicated   Device 3   Device 4

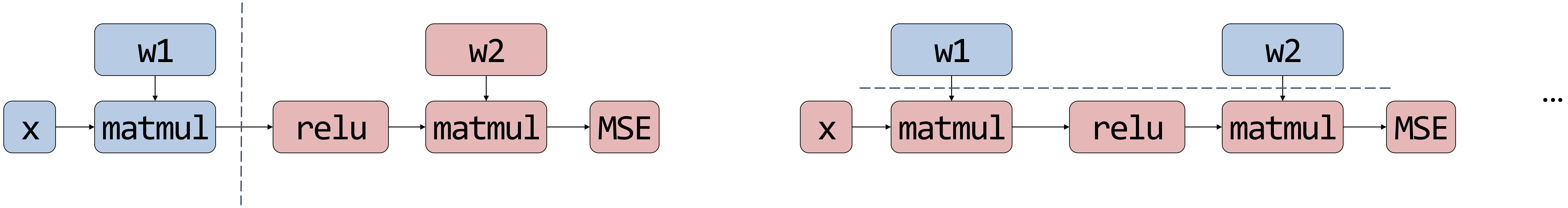
More strategies



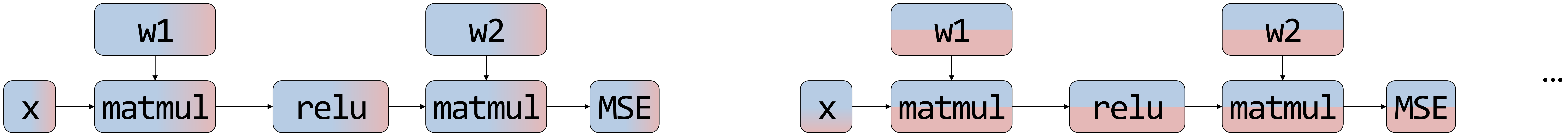
Summary: Inter-op and Intra-op Parallelisms



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

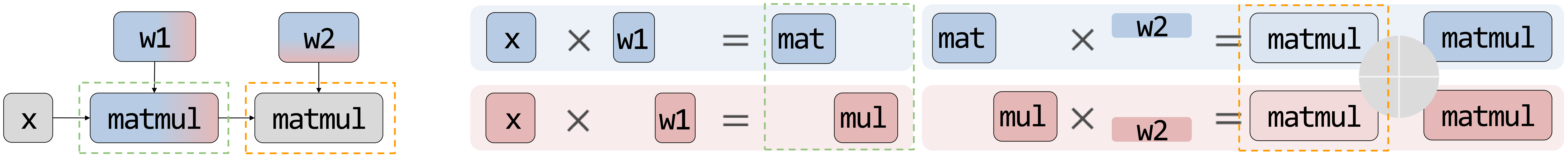


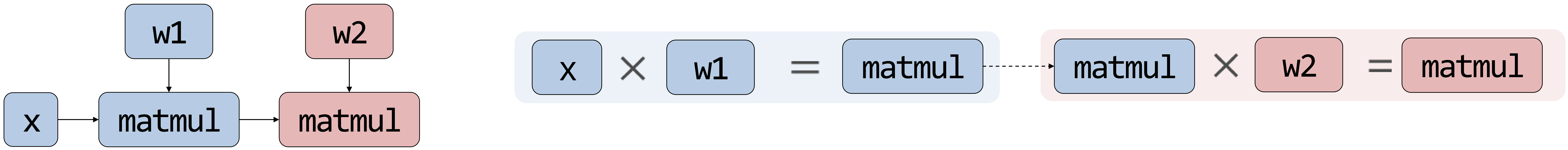
Intra-op parallelism: Assign different regions of a single operator to different devices.



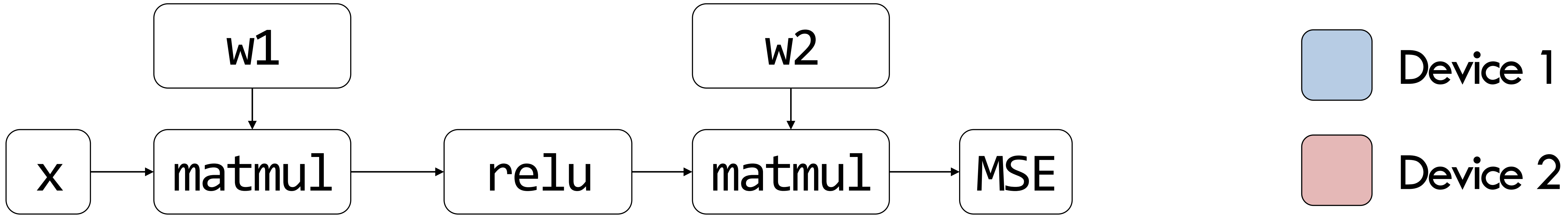
Inside Intra- and Inter-op Parallelism

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

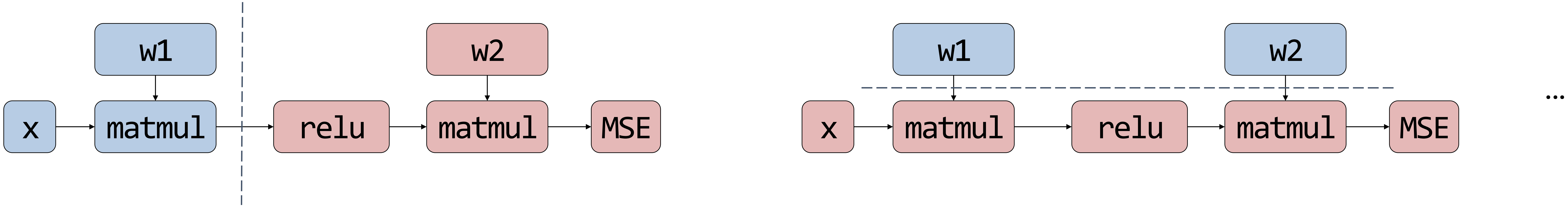




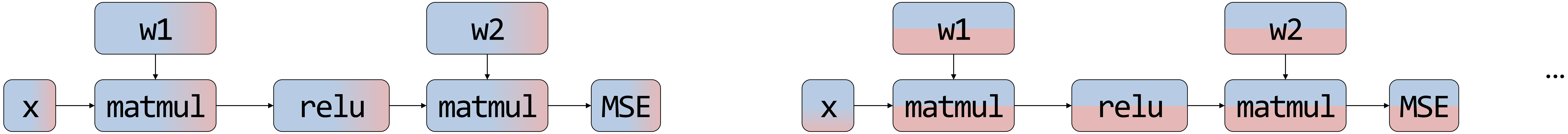
Inter-op and Intra-op Parallelism: Characteristics



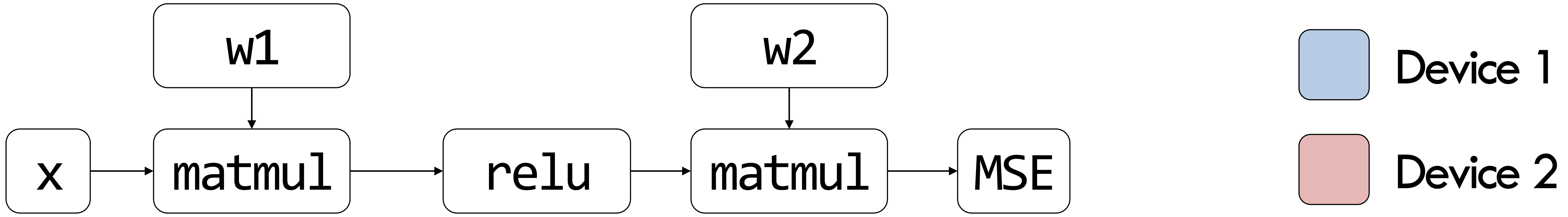
Inter-op parallelism: Requires point-to-point communication but results in device idle



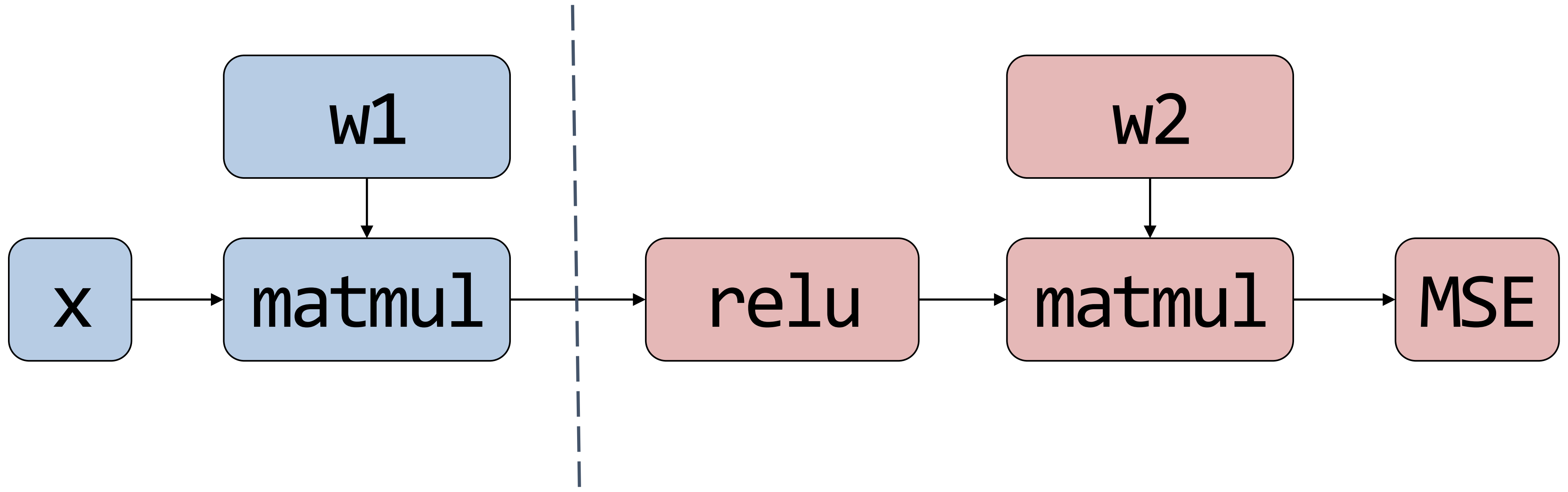
Intra-op parallelism: Devices are busy but requires collective communication



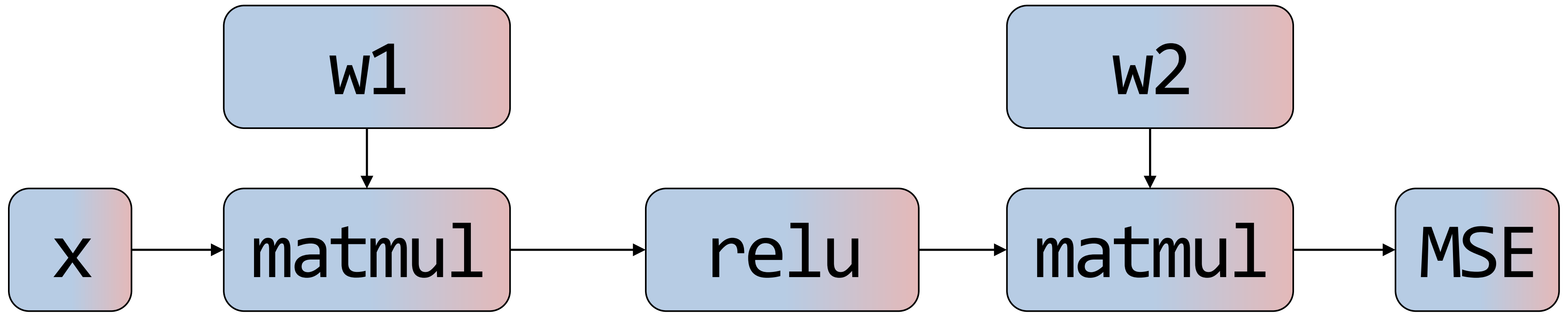
Inter-op and Intra-op Parallelism: Characteristics



Inter-op parallelism



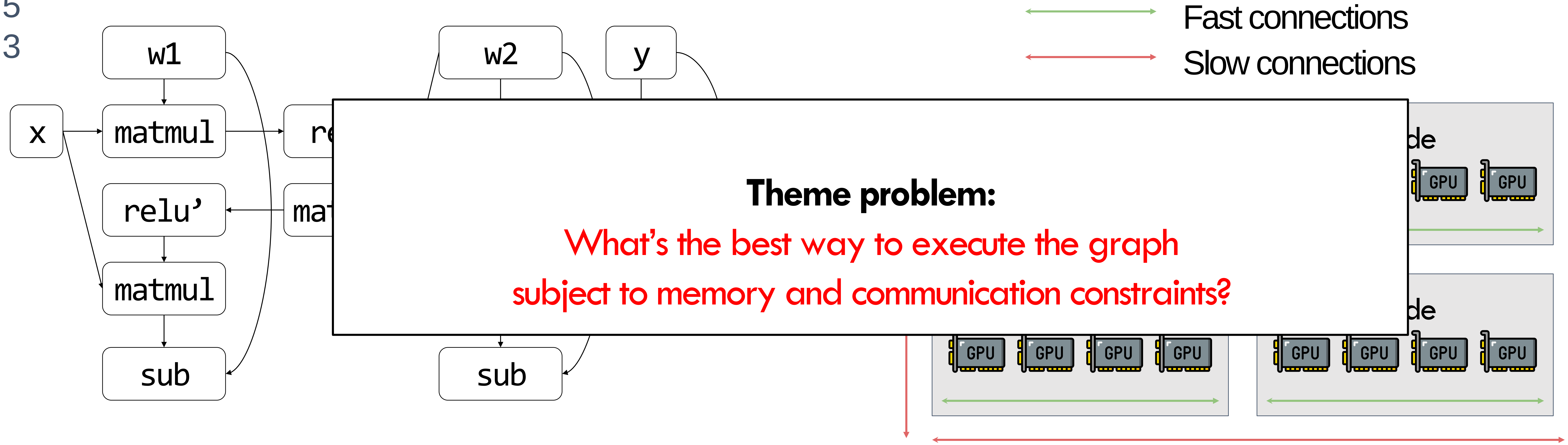
Intra-op parallelism



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?