https://hao-ai-lab.github.io/cse234-w25/

CSE 234: Data Systems for Machine Learning
Winter 2025

LLMSys

Optimizations and Parallelization

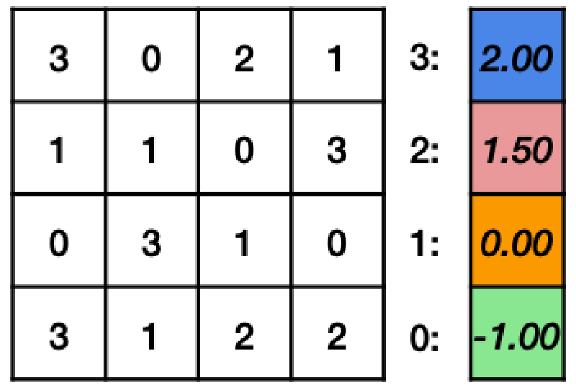
MLSys Basics

Today's Learning Goal

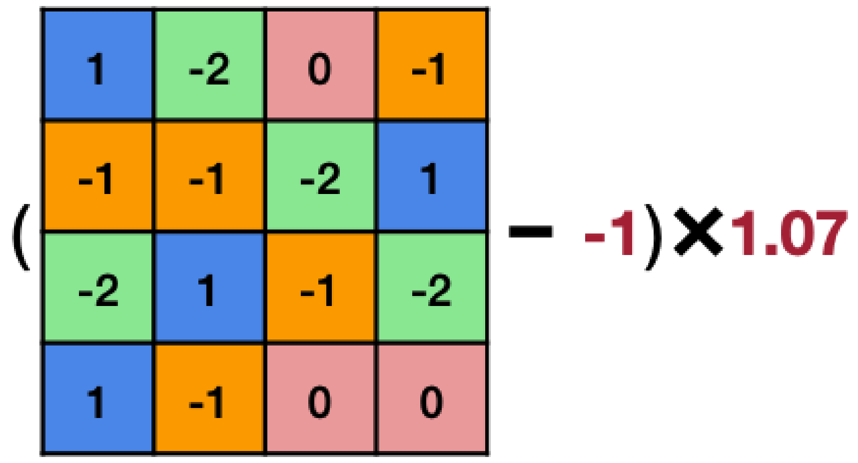
- Post-training Quantization
  - Quantization Granularity
  - Quantization on Activations
- Mixed precision
- Parallelization: Starter

Quantization Basics

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49



K-Means-based Quantization



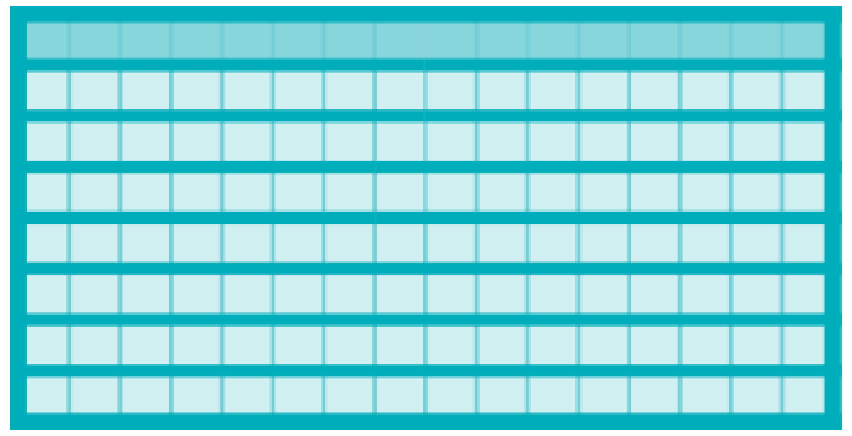
Linear Quantization

Storage	Floating point weights	integer weights; floating-point codebook	integer weights;
Compute	Floating point arithmetic	Floating point arithmetic	Integer arithmetic

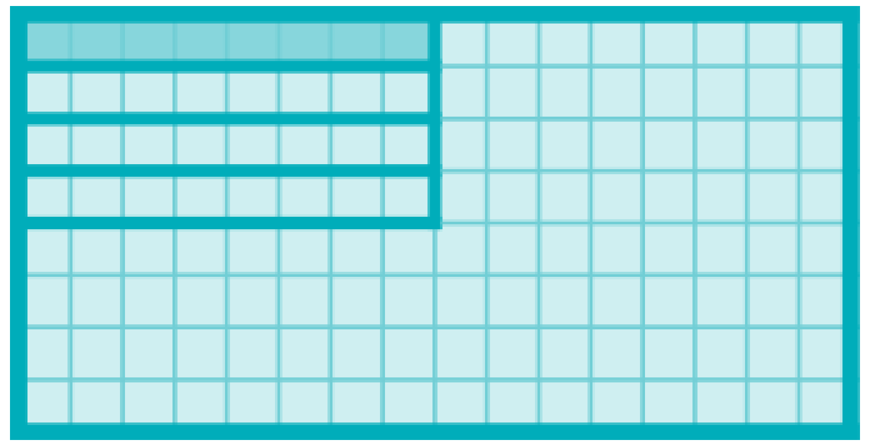
Quantization Granularity

- Per-tensor Quantization

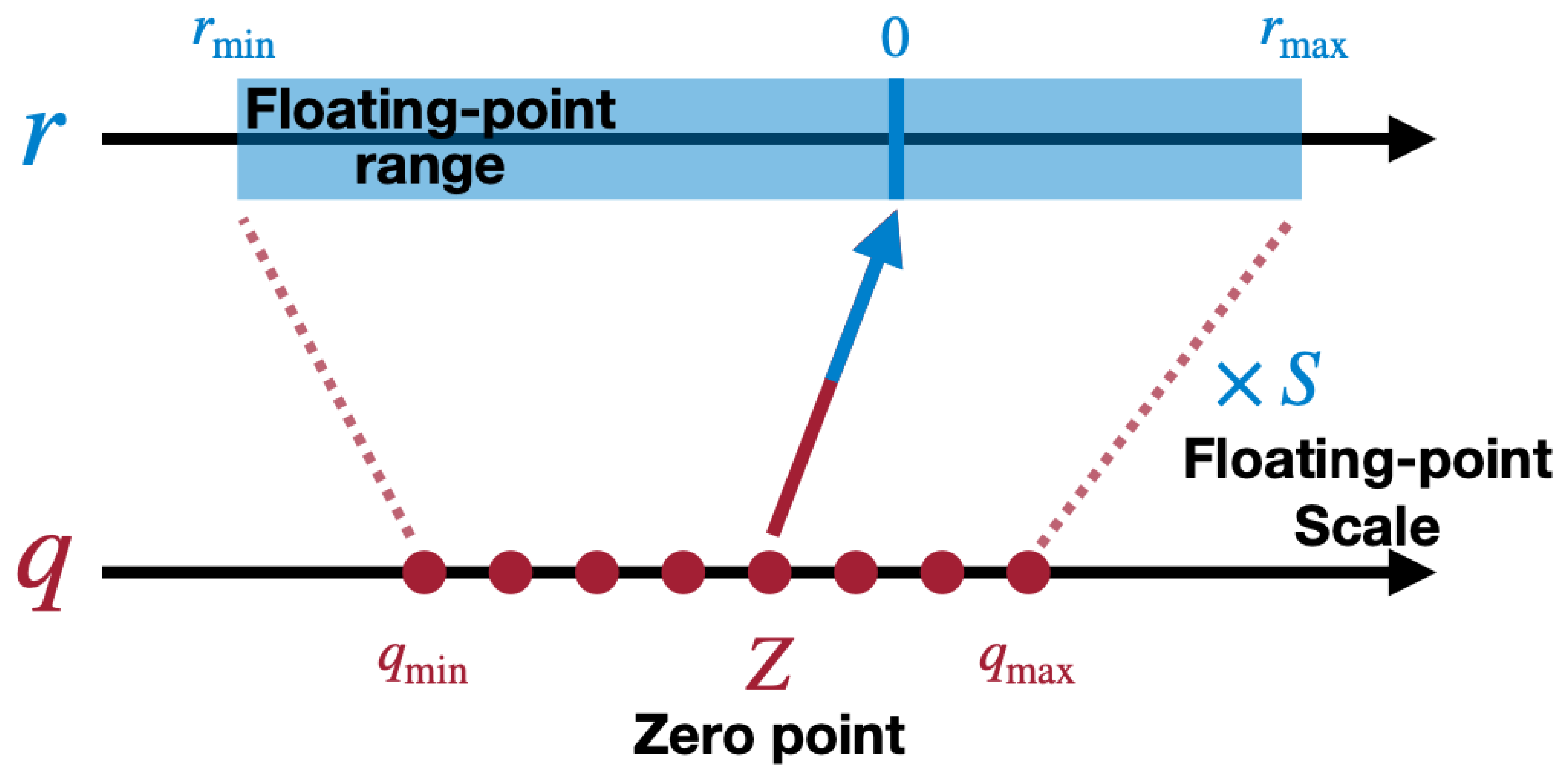
- Per-channel Quantization



- Group Quantization



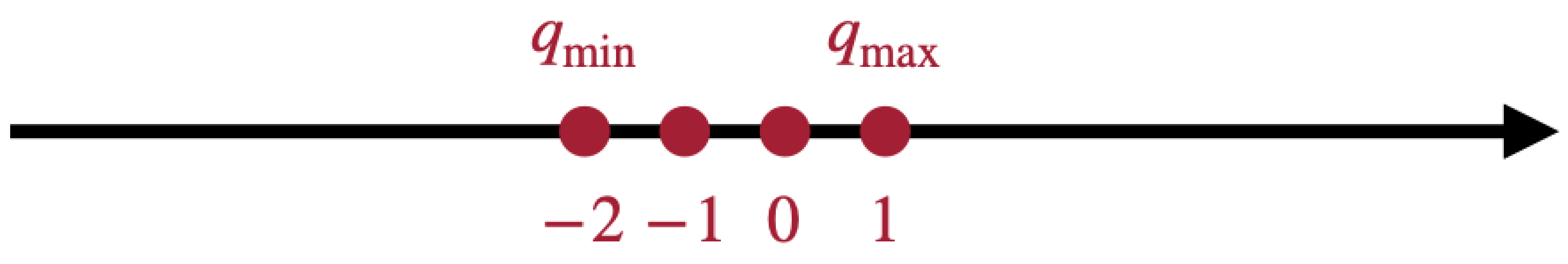
$r = S(q - Z)$: Determine S and Z



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Per-tensor quantization

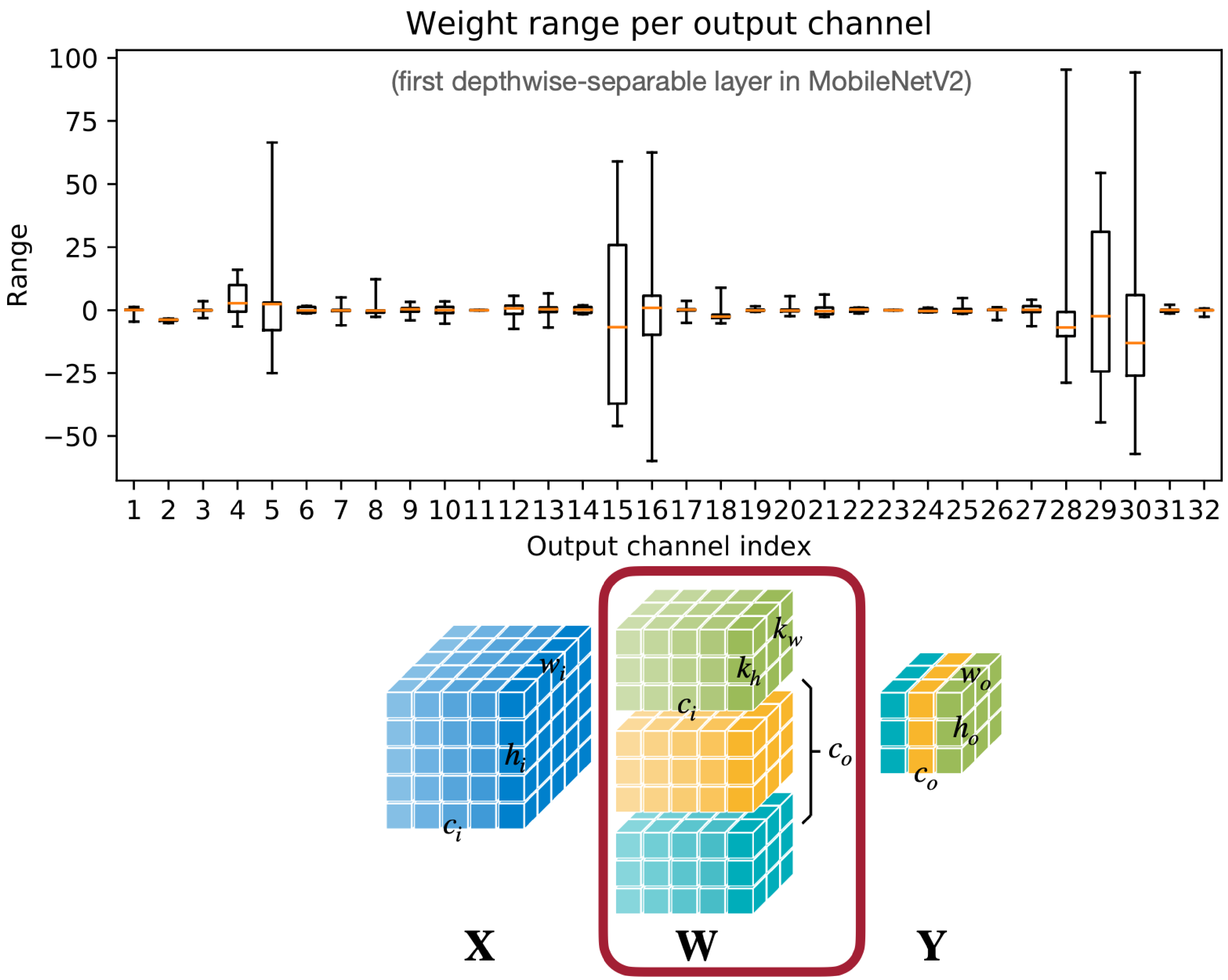
Binary	Decimal
01	1
00	0
11	-1
10	-2



$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} \qquad S = \frac{2.12 - (-1.08)}{1 - (-2)} = 1.07$$

$$Z = q_{min} - \frac{r_{min}}{S} \qquad Z = \text{round}(q_{min} - \frac{r_{min}}{S})$$

Per-Tensor Quantization in Practice



- Per-tensor quantization
  - Using single scale S for whole weight tensor
- Common failure results from
  - Outlier weights
- Solution: per-channel quantization

Per-channel Quantization

- Example: 2-bit linear quantization

Per-tensor quant

Per-channel quant

$$
\begin{array}{cc}
 & ic \\
oc & \begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}
\end{array}
$$

Per-channel Quantization

- Example: 2-bit linear quantization

ic / oc

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Binary	Decimal
01	1
00	0
11	-1
10	-2

Per-tensor quant

$$|r|_{max} = 2.12$$

$$S = \frac{|r|_{max}}{q_{max}} = \frac{2.12}{2^{2-1}-1} = 2.12$$

1	0	1	0
0	0	-1	1
0	1	0	0
1	0	1	1

Quantized

2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	0
2.12	0	2.12	2.12

Reconstructed

Per-channel quant

Per-channel Quantization

- Example: 2-bit linear quantization

ic (columns), oc (rows)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Binary	Decimal
01	1
00	0
11	-1
10	-2

Per-tensor quant

$$|r|_{max} = 2.12$$

$$S = \frac{|r|_{max}}{q_{max}} = \frac{2.12}{2^{2-1}-1} = 2.12$$

Quantized

1	0	1	0
0	0	-1	1
0	1	0	0
1	0	1	1

Reconstructed

2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	0
2.12	0	2.12	2.12

$$||W - S \odot q_W|| = 2.28$$

Per-channel quant

$$|r|_{max} = 2.09 \quad S_0 = 2.09$$
$$|r|_{max} = 2.12 \quad S_0 = 2.12$$
$$|r|_{max} = 1.92 \quad S_0 = 1.92$$
$$|r|_{max} = 1.87 \quad S_0 = 1.87$$

Quantized

1	0	1	0
0	0	-1	1
0	1	0	-1
1	0	1	1

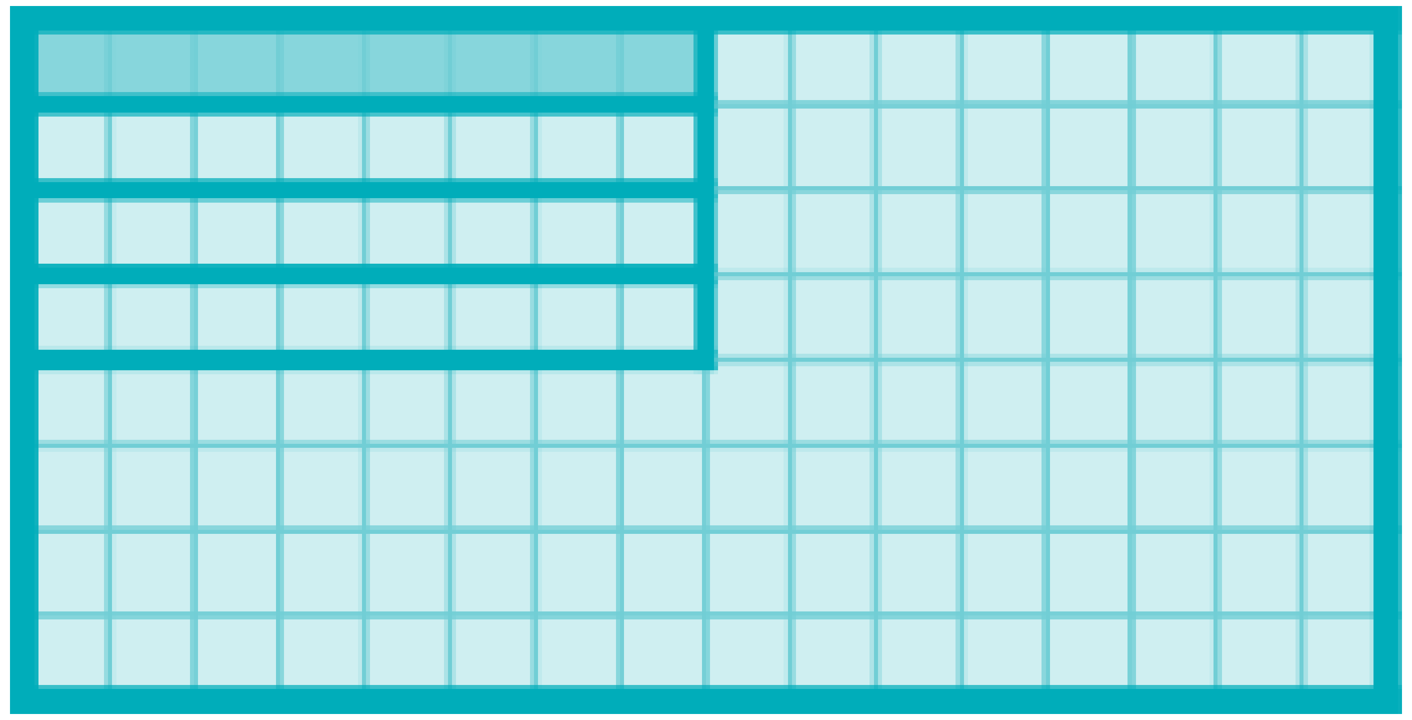
Reconstructed

2.09	0	2.09	0
0	0	-2.12	2.12
0	1.92	0	-1.92
1.87	0	1.87	1.87

$$||W - S \odot q_W|| = 2.08$$

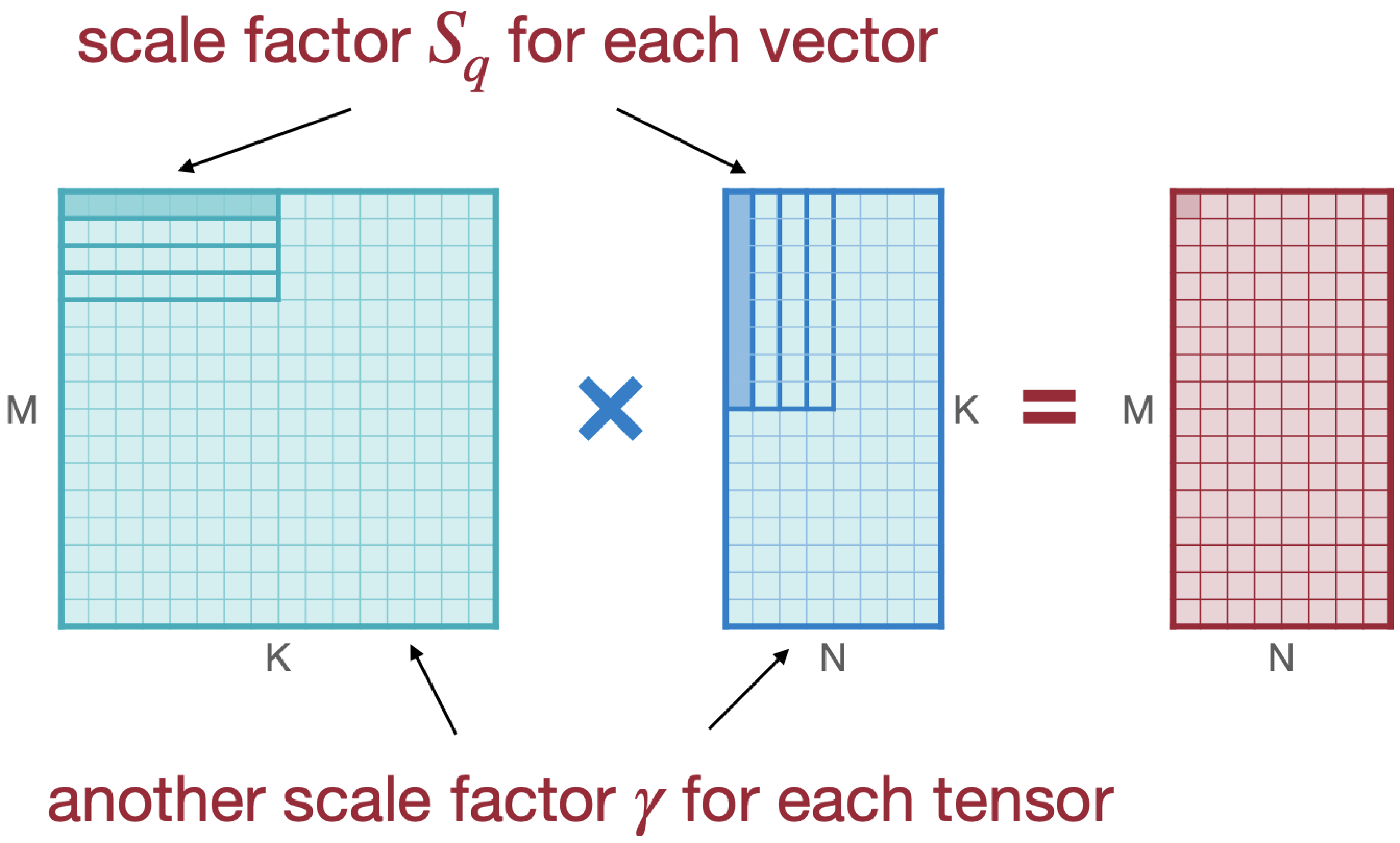
Group Quantization

- More fine-grained quantization granularity, e.g. per vector
- Pros: More accuracy, less quantization error
- Cons?



Sweetspot: Two-level Quantization

$$r = S(q - Z) \quad \rightarrow \quad r = \gamma S_q(q - Z)$$



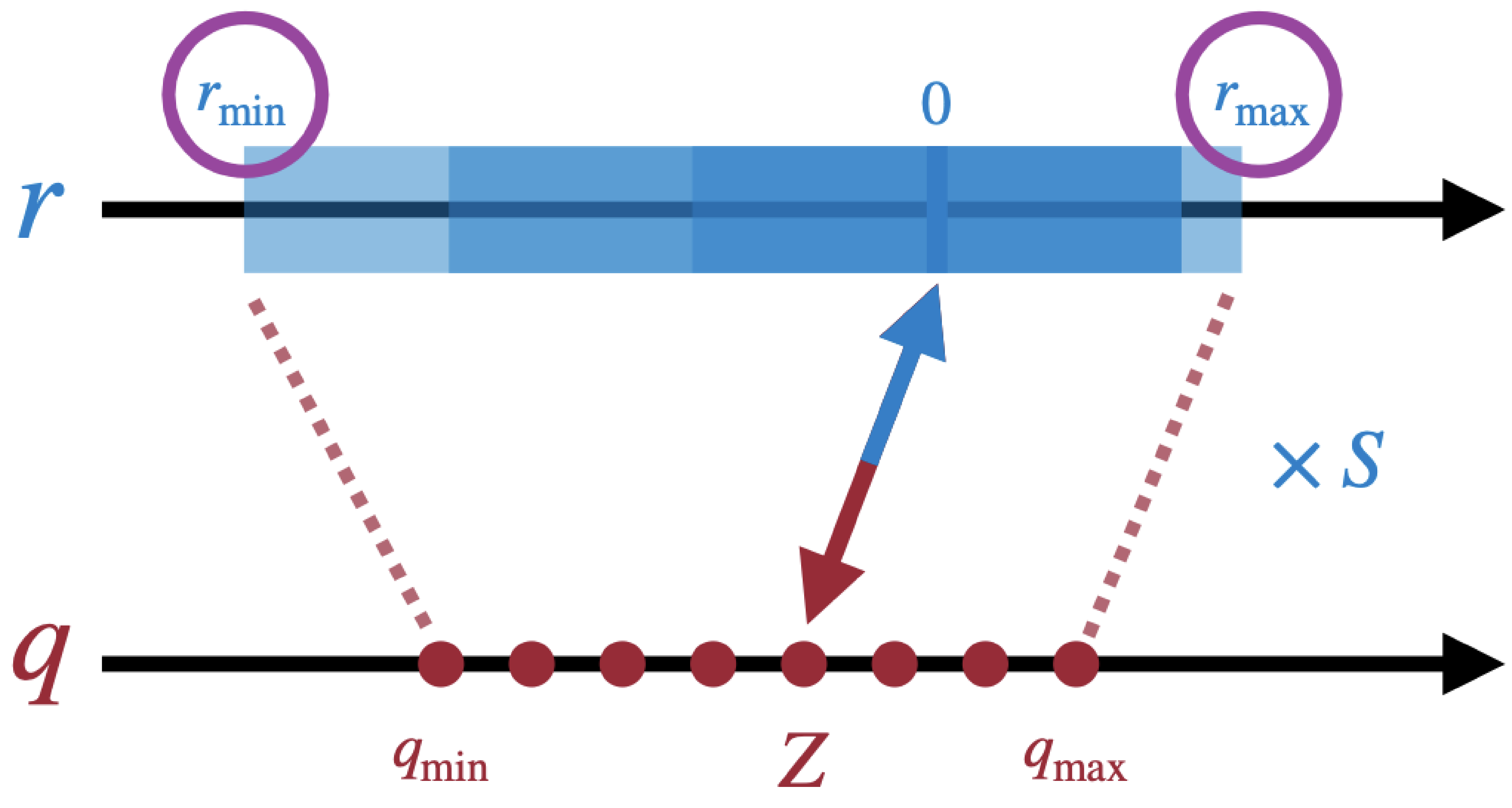
- γ is a floating-point coarse grained scale factor
- S_q is an integer per-vector scale factor
- Assume:
  - 4-bit quantization
  - 4-bit per-vector scale every 16 elements
  - Cost?

Generalize: Multi-level Quantization

$$r = S(q - Z) \quad \Rightarrow \quad r = S_{l_0} S_{l_1} \cdots (q - Z)$$

- r: real number, e.g., fp16
- q: quantized value
- Z: zero point ($z = 0$ is symmetric quantization)
- S_{l_0}: scale factors of different levels

Linear Quantization on Activations

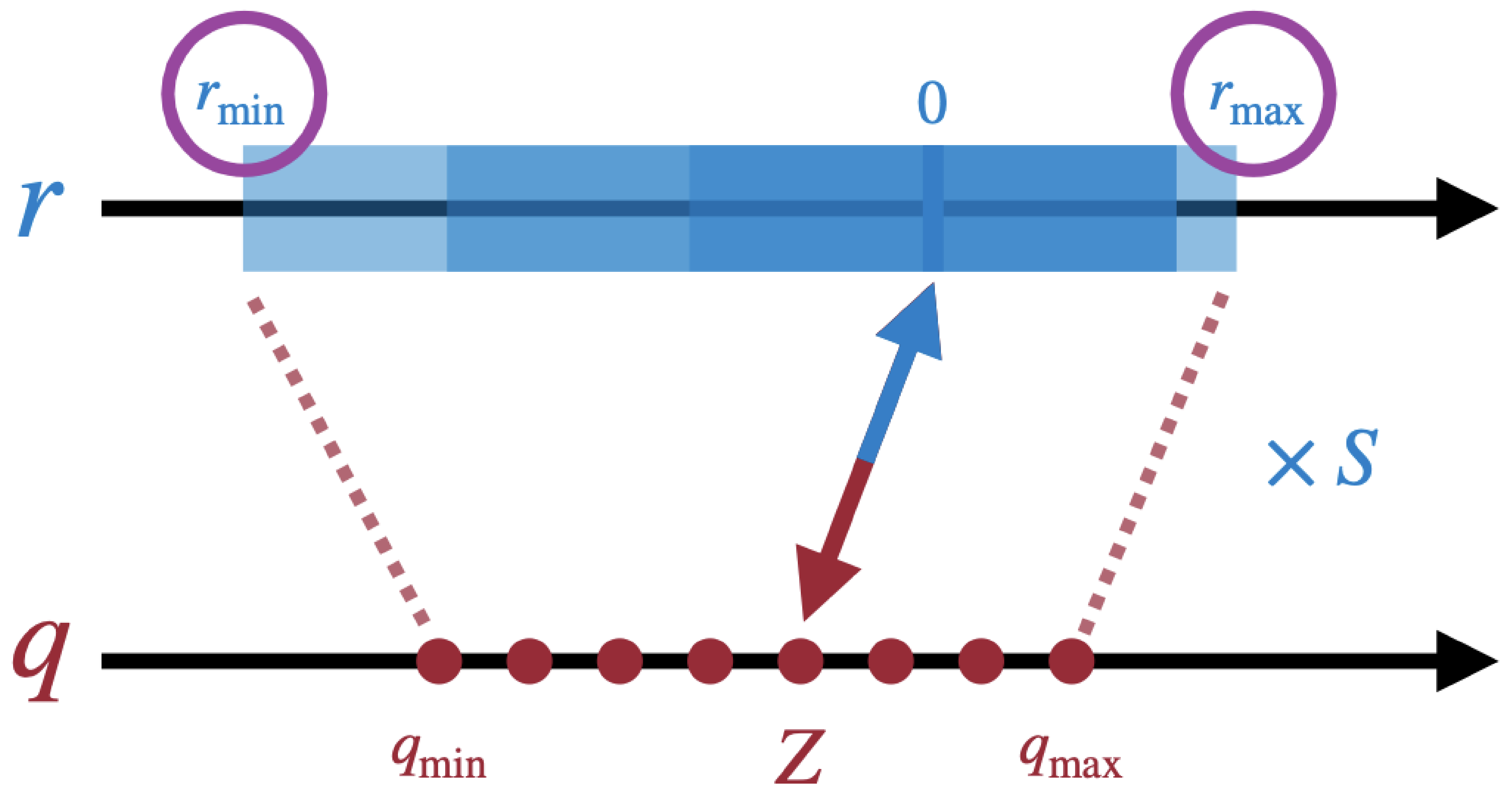


- Weights: static.
- Activations: range varies across inputs.
  - To determine the floating-point range (r_{min}, r_{max}), the activations stats are gathered before deploying the model

Dynamic Range: Moving Average

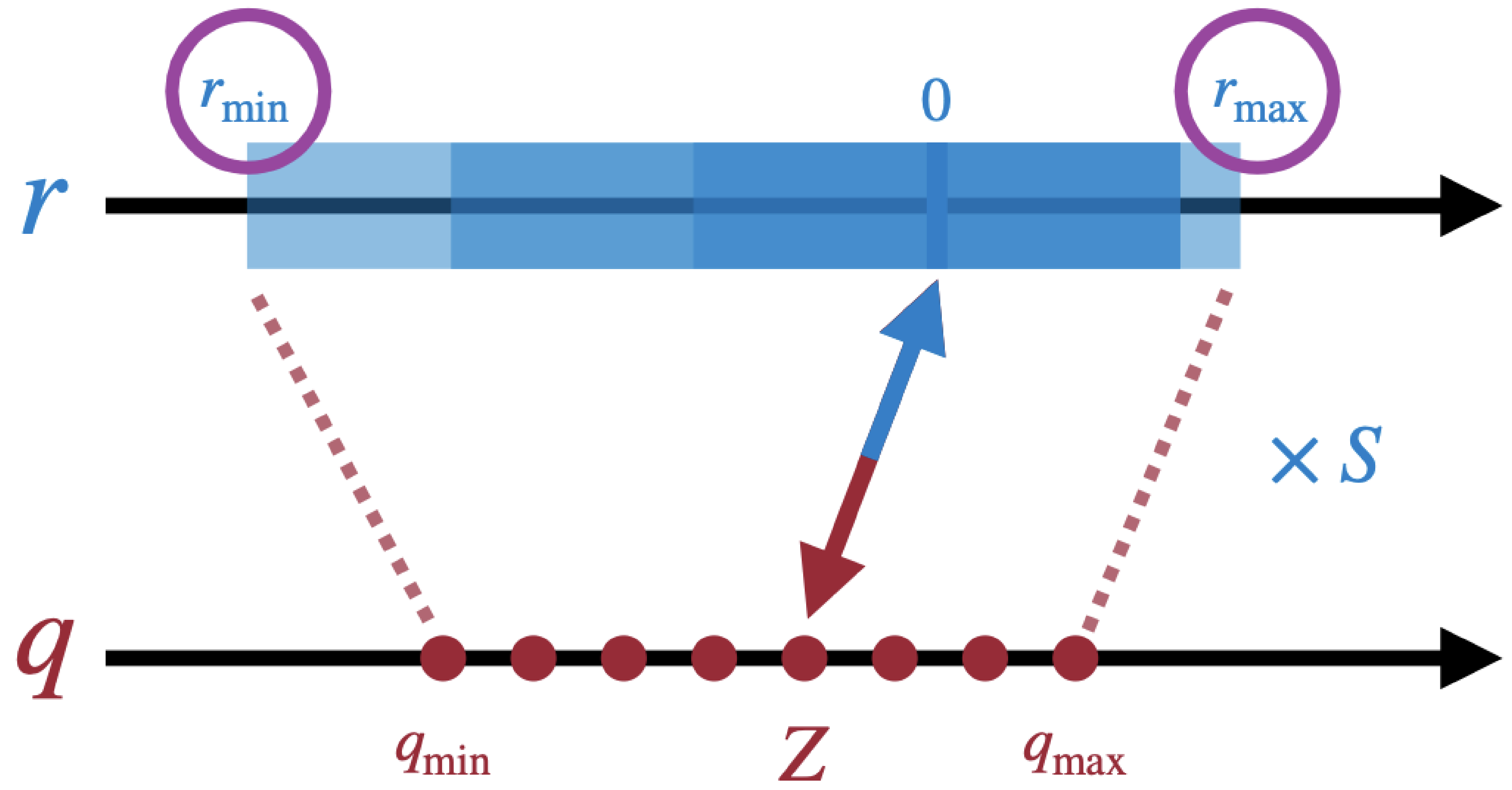
Moving average: observed ranges are smoothed across thousands of training steps

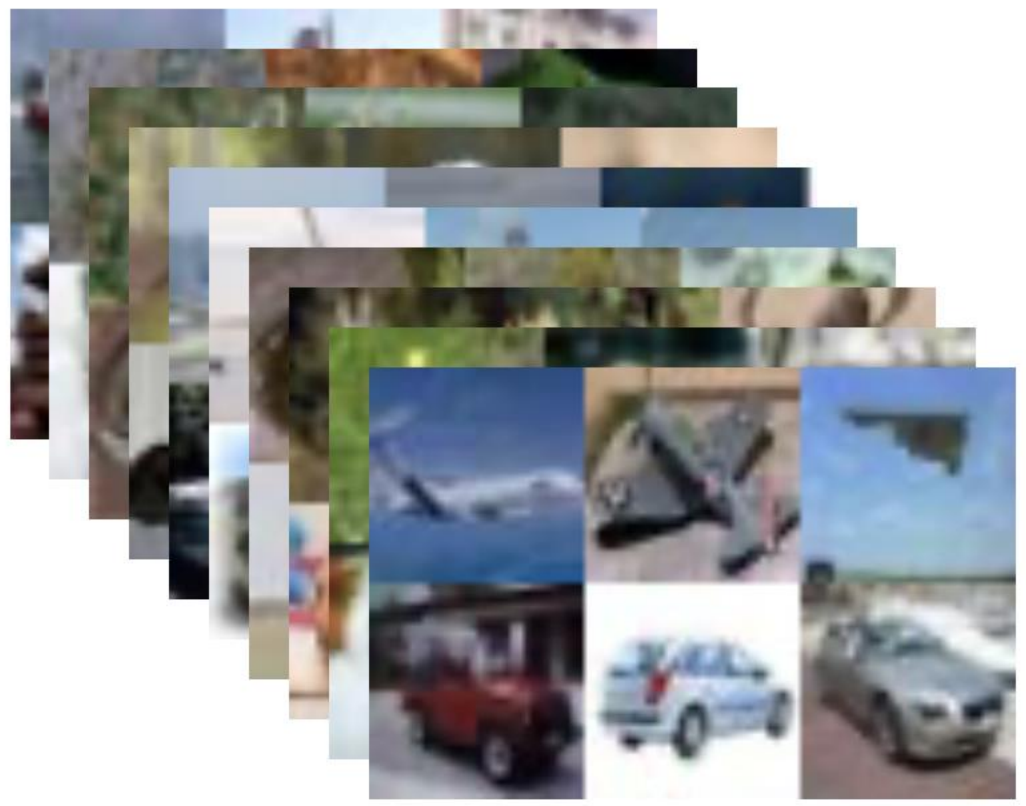
$$\hat{r}^{(t)}_{max,min} = \alpha \cdot r^{(t)}_{max,min} + (1 - \alpha)\hat{r}^{(t-1)}_{max,min}$$



Dynamic Range: Calibration

- By running a few "calibration" samples on the trained FP32 model
  - Spending dynamic range on the outliers hurts the representation ability

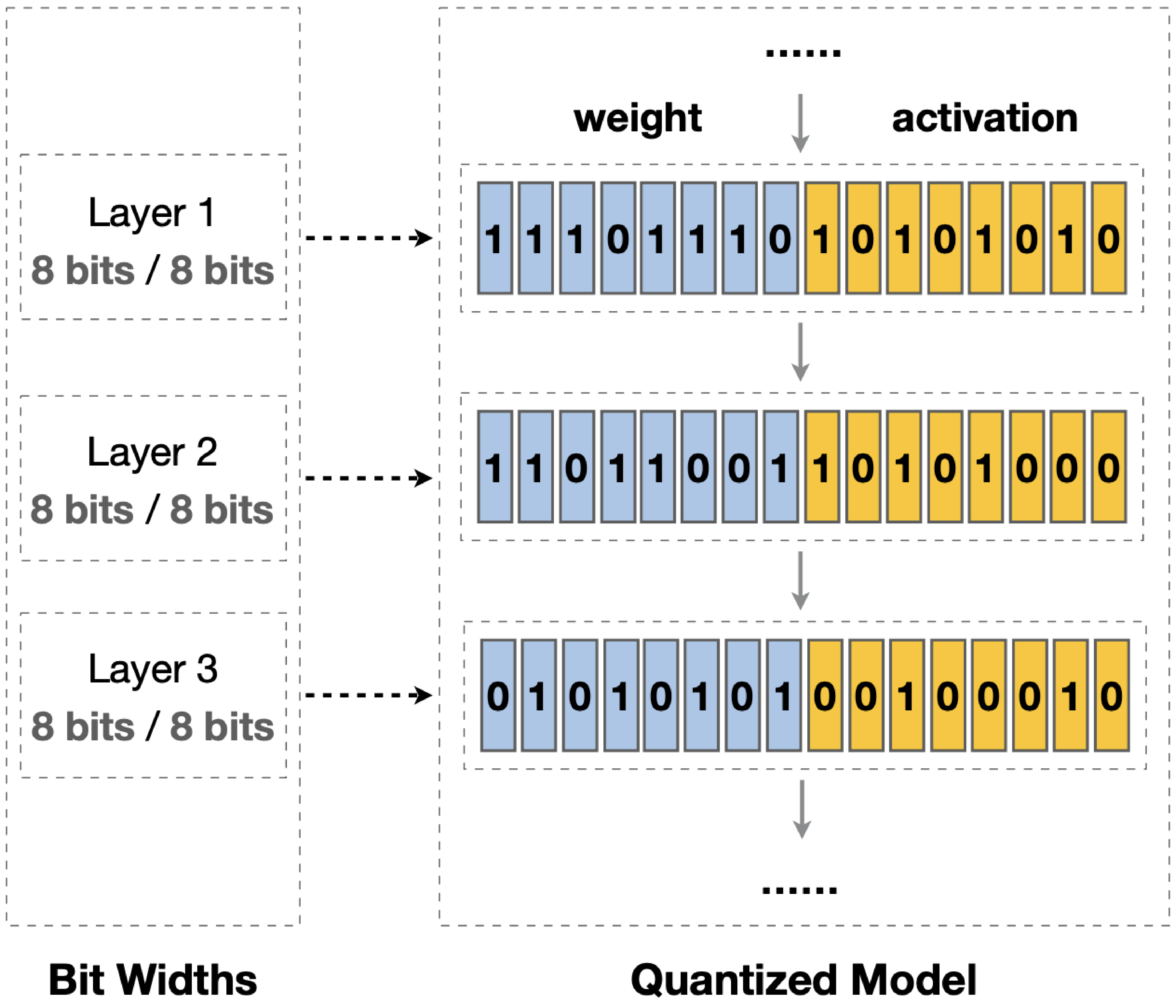




Today’s Learning Goal

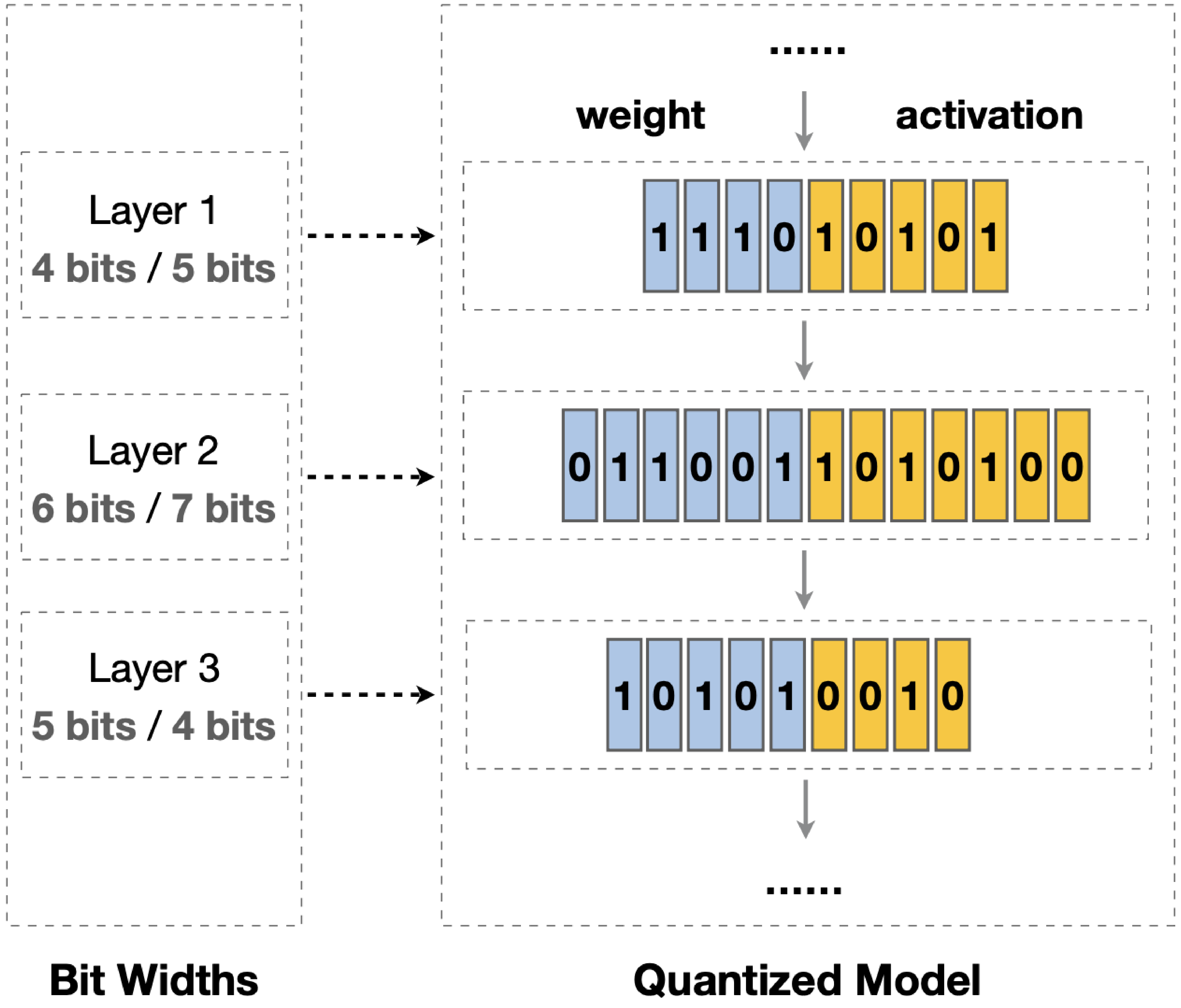
- Post-training Quantization
  - Quantization Granularity
  - Quantization on Activations
- **Mixed precision**
- Parallelization: Starter

Uniform Quantization

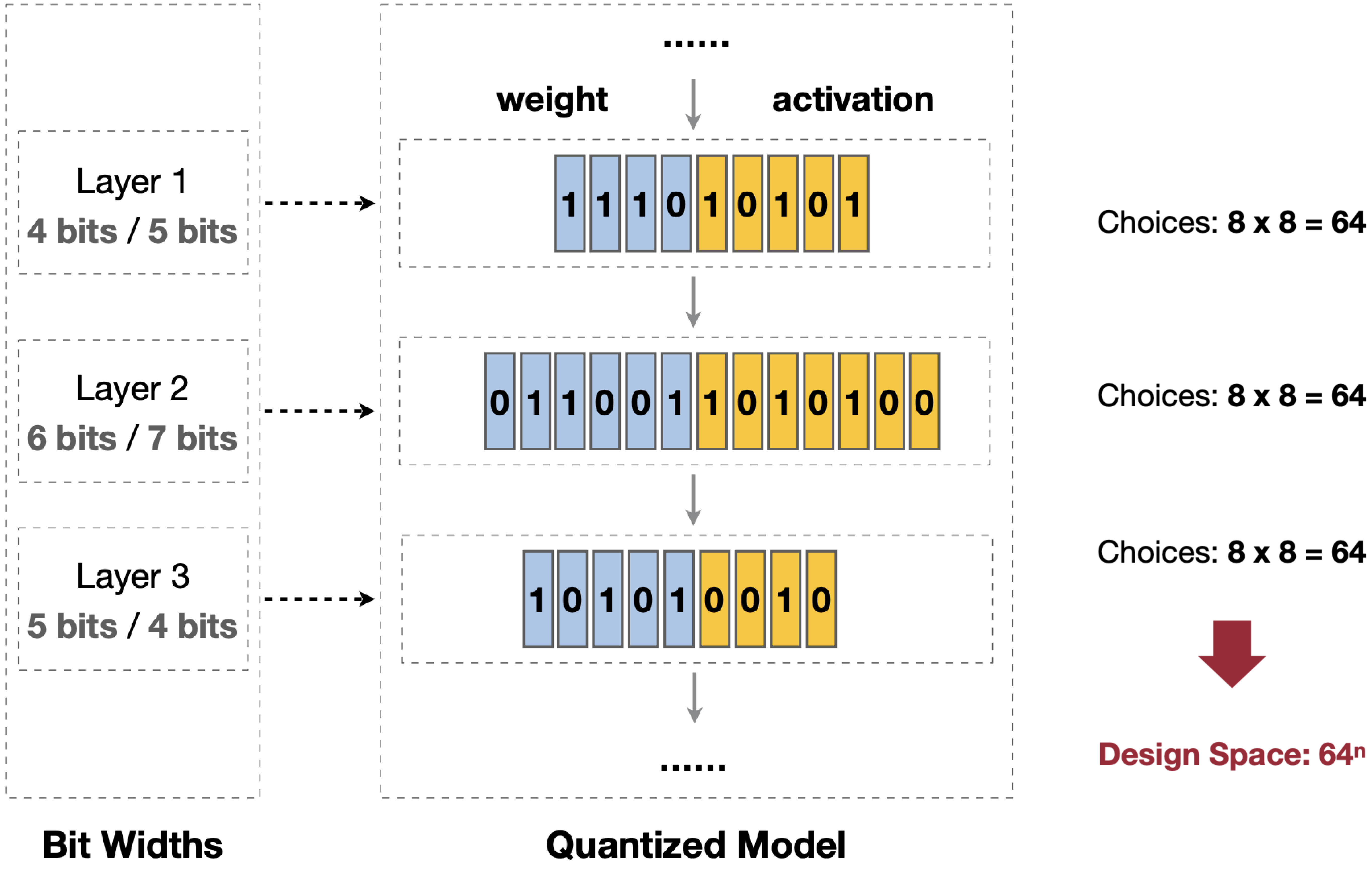


Mixed-Precision Quantization

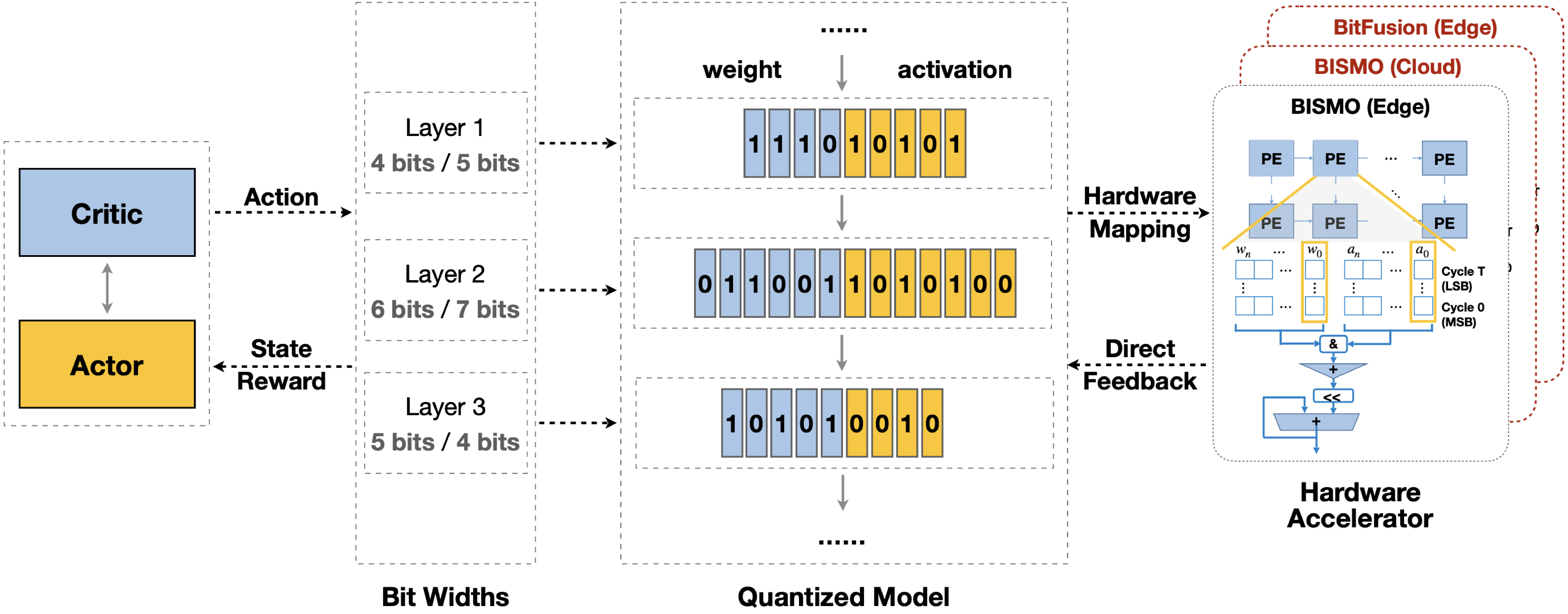
- Intuition: why this works?



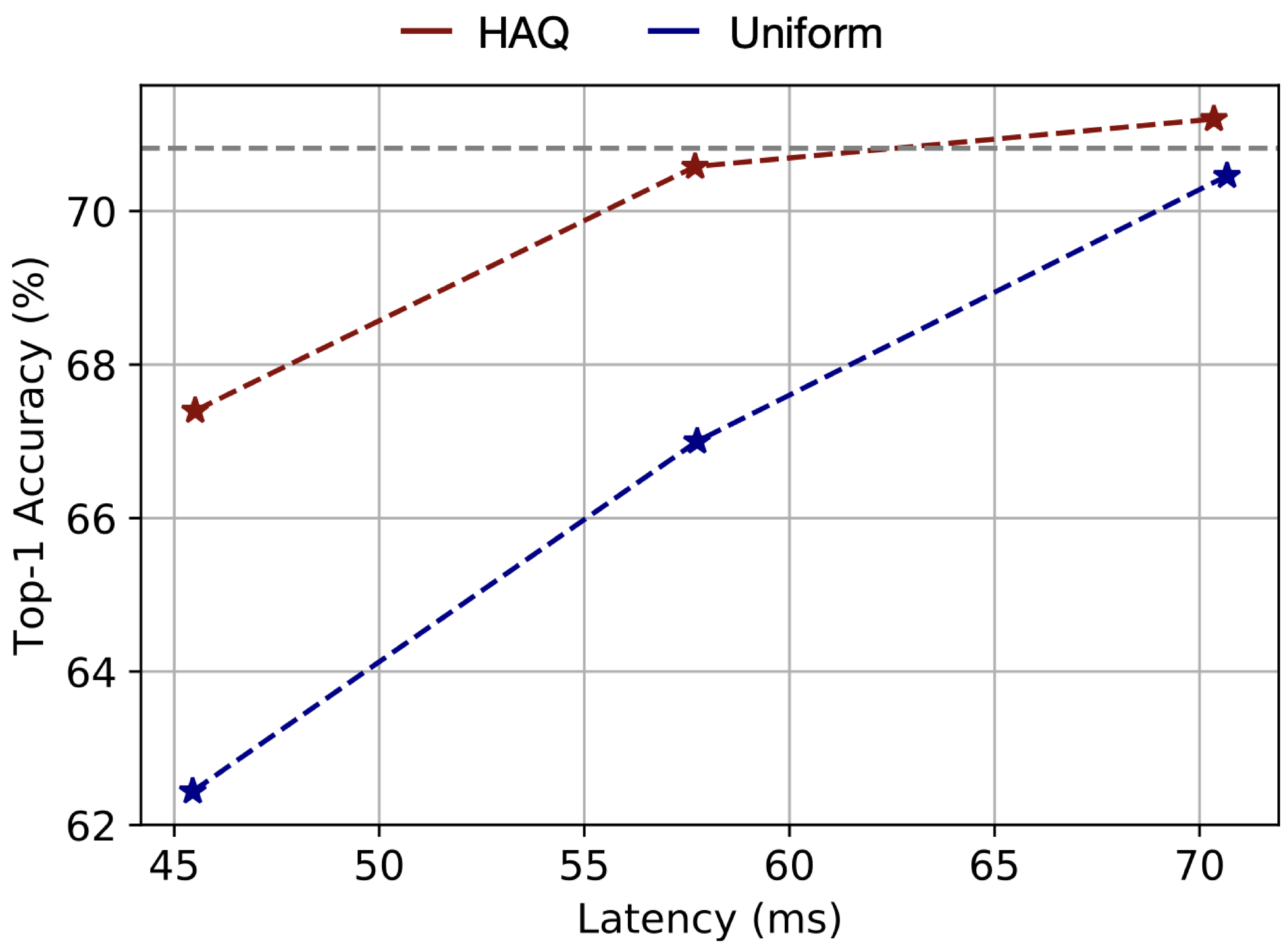
Mixed Precision: Design Space



Mixed Precision: Design Automation



Mixed Precision: Design Automation



In Practice: Mixed Precision Training

Published as a conference paper at ICLR 2018

MIXED PRECISION TRAINING

Sharan Narang*, **Gregory Diamos, Erich Elsen**†
Baidu Research
{sharan, gdiamos}@baidu.com

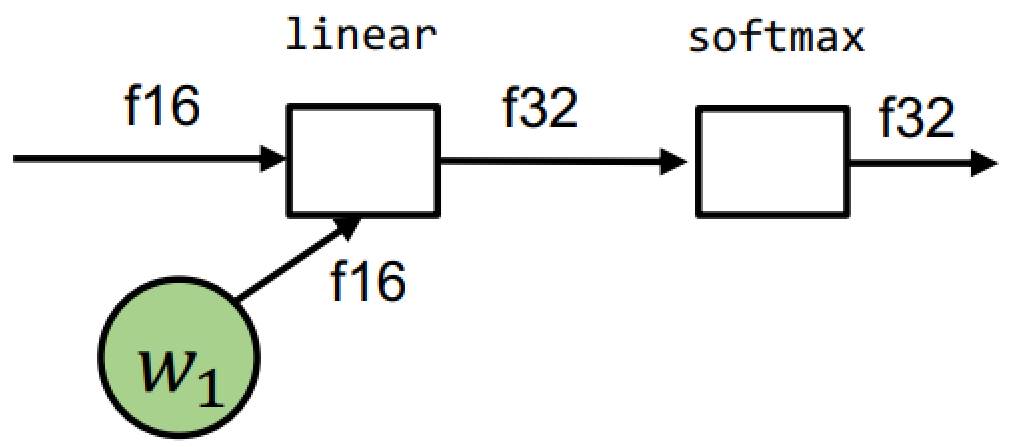
Paulius Micikevicius*, **Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, Hao Wu**
NVIDIA
{pauliusm, alben, dagarcia, bginsburg, mhouston, okuchaiev, gavenkatesh, skyw}@nvidia.com

ABSTRACT

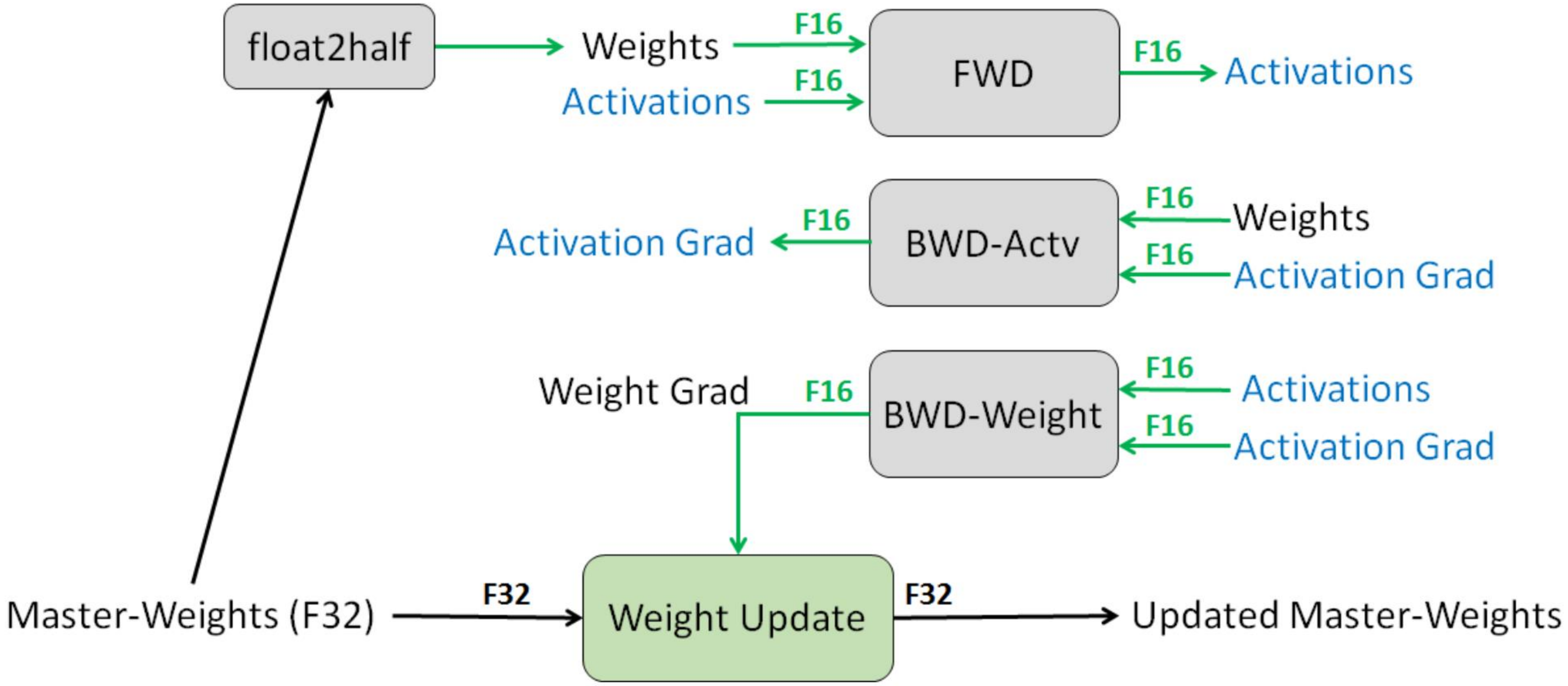
Increasing the size of a neural network typically improves accuracy but also increases the memory and compute requirements for training the model. We introduce methodology for training deep neural networks using half-precision floating point numbers, without losing model accuracy or having to modify hyper-parameters. This nearly halves memory requirements and, on recent GPUs, speeds up arithmetic. Weights, activations, and gradients are stored in IEEE half-precision format. Since this format has a narrower range than single-precision we propose three techniques for preventing the loss of critical information. Firstly, we recommend maintaining a single-precision copy of weights that accumulates the gradients after each optimizer step (this copy is rounded to half-precision for the forward- and back-propagation). Secondly, we propose loss-scaling to preserve gradient values with small magnitudes. Thirdly, we use half-precision arithmetic that accumulates into single-precision outputs, which are converted to half-precision before storing to memory. We demonstrate that the proposed methodology works across a wide variety of tasks and modern large scale (exceeding 100 million parameters) model architectures, trained on large datasets.

Mix-precision training

- Some layers are more sensitive to dynamic range/precision
  - Normalization: f / sum(f)
  - Softmax (same with normalization)
- Common issues: aggregation of a lot entries
  - Param += \sum(grad_t) -> can loss precision during accum
- Idea: identify which ops are sensitive to precisions:
  - Use full precision (fp32) for them via upcasting
  - Use half precision to those robust ops

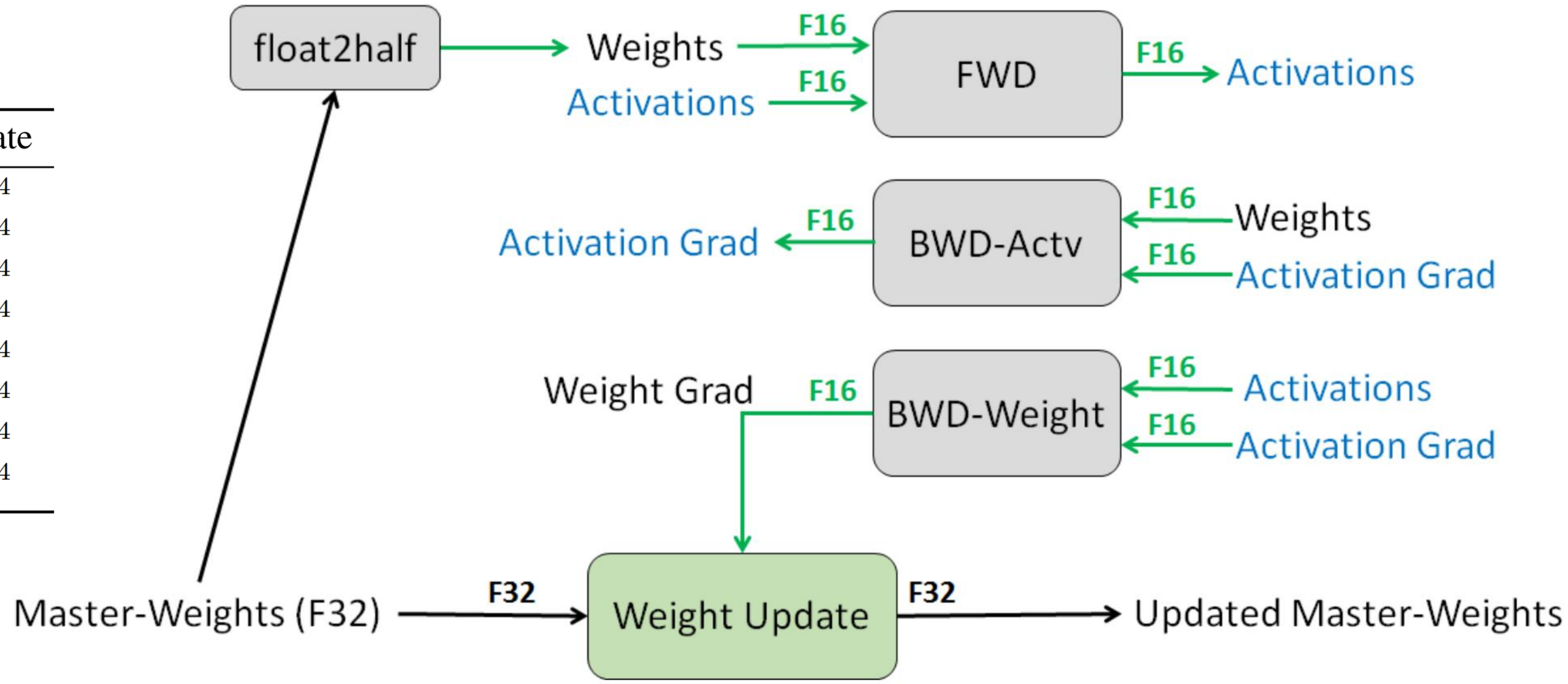


A standardized 16-32 mix-precision pipeline (Important!)



Analysis of the memory usage of Mix-precision training

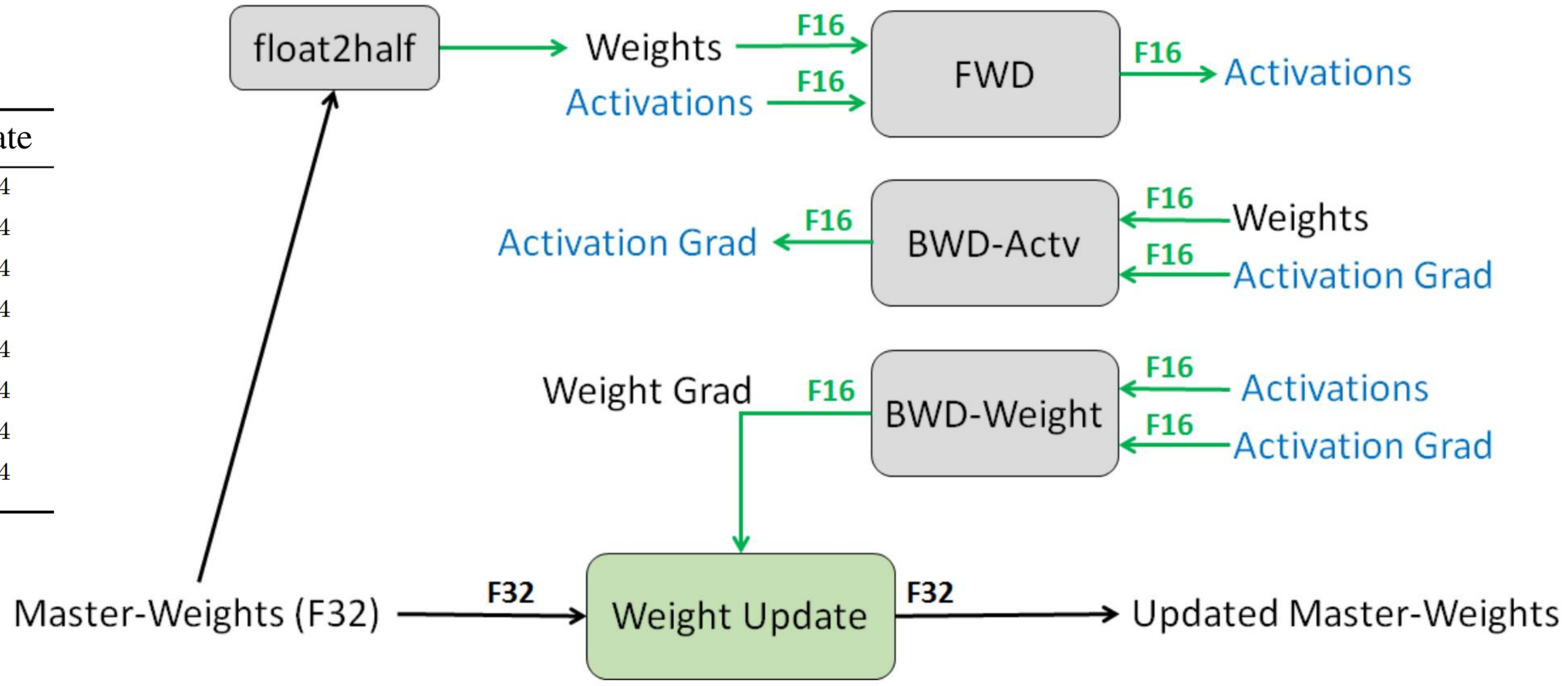
Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}



- Parameters: 175B * (fp16, 2 bytes) = 350G
- Assume we checkpoint at transformer layer boundary:
  - Activations: (N = 96) * 3.2M * 12288 * 2 = 7488 G
- How about optimizer states?

Analysis of the memory usage of Mix-precision training

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}



- How about optimizer states?
  - Master copy (fp32) = 4 * 175 = 700
  - Grad (fp16) = 2 * 175 = 350
  - Running copy (fp16) = 2 * 175 = 350
  - Adam mean and variance (fp32) = 2 * 4 * 175 = 1400G
- **Rule the thumb: (4 + 2 + 2 + 4 + 4) N = 16N memory for an LLM**

More on Scaling Down ML

- Running ML on edge devices is always strongly demanded
- Market characteristics: *very fragmented*
- Research directions: quantization, pruning, ML energy efficiency, federated ML etc.

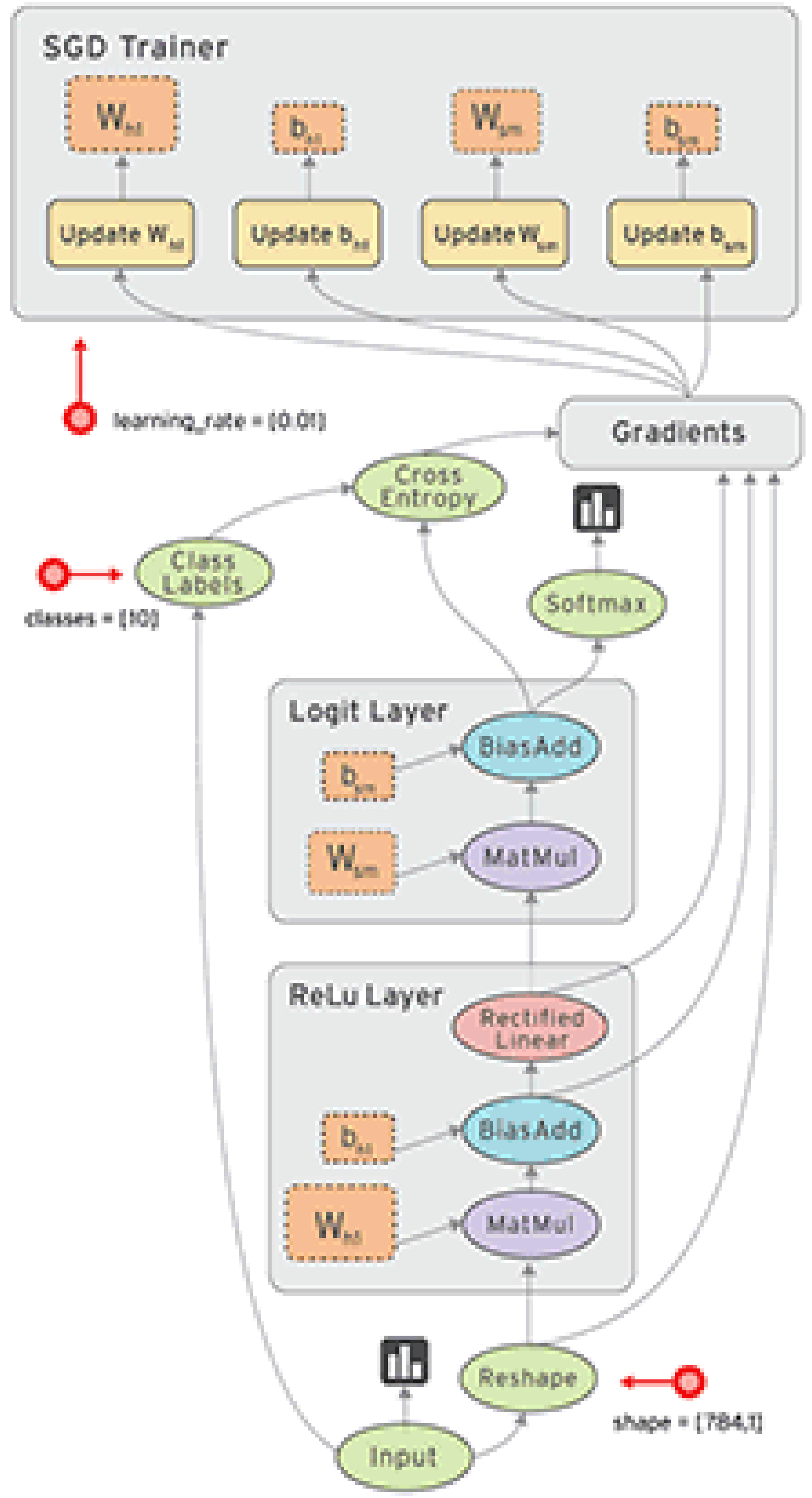
Zhijian Liu

zhijian [at] ucsd (dot) edu

I am a research scientist at NVIDIA and an incoming assistant professor at UCSD. I finished my PhD at MIT, advised by Song Han. My research focuses on efficient machine learning and systems.

I will be recruiting PhD students from HDSI/CSE in the Fall 2024 cycle and also looking for RAs/interns!

Big Picture



Dataflow Graph

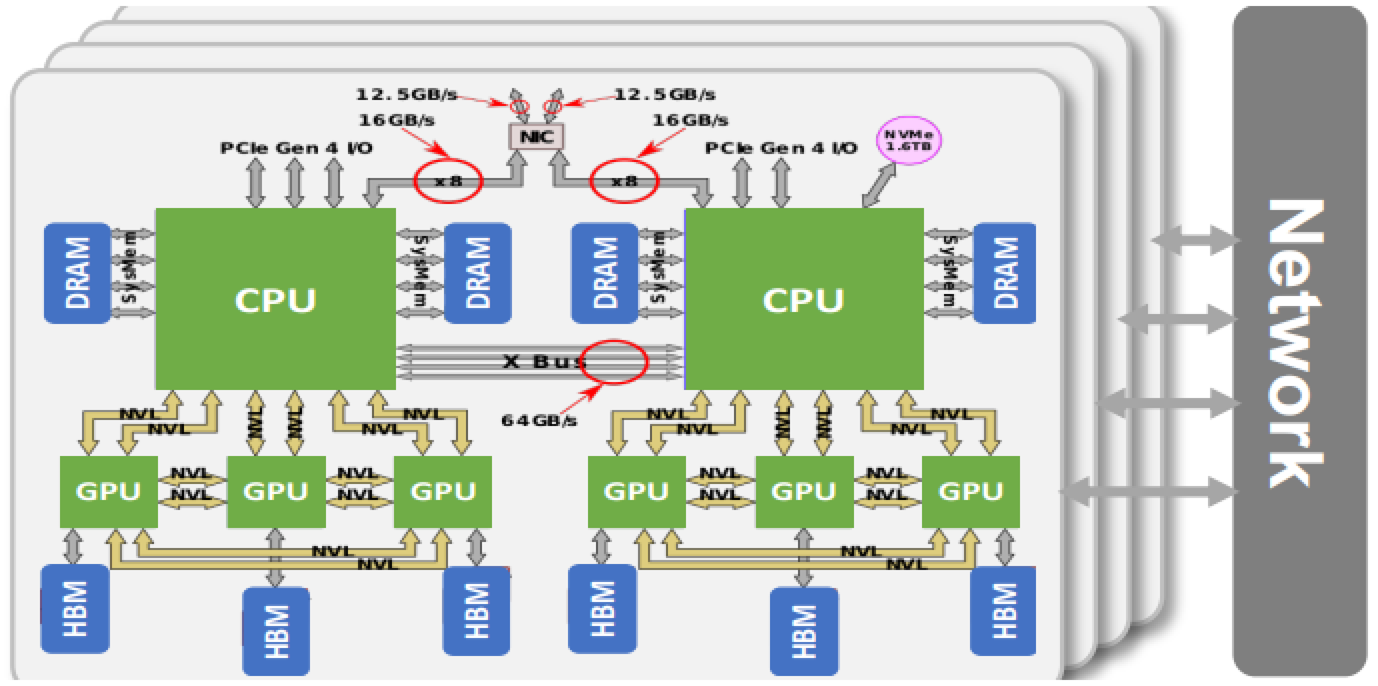
Autodiff

Graph Optimization

Parallelization

Runtime: schedule / memory

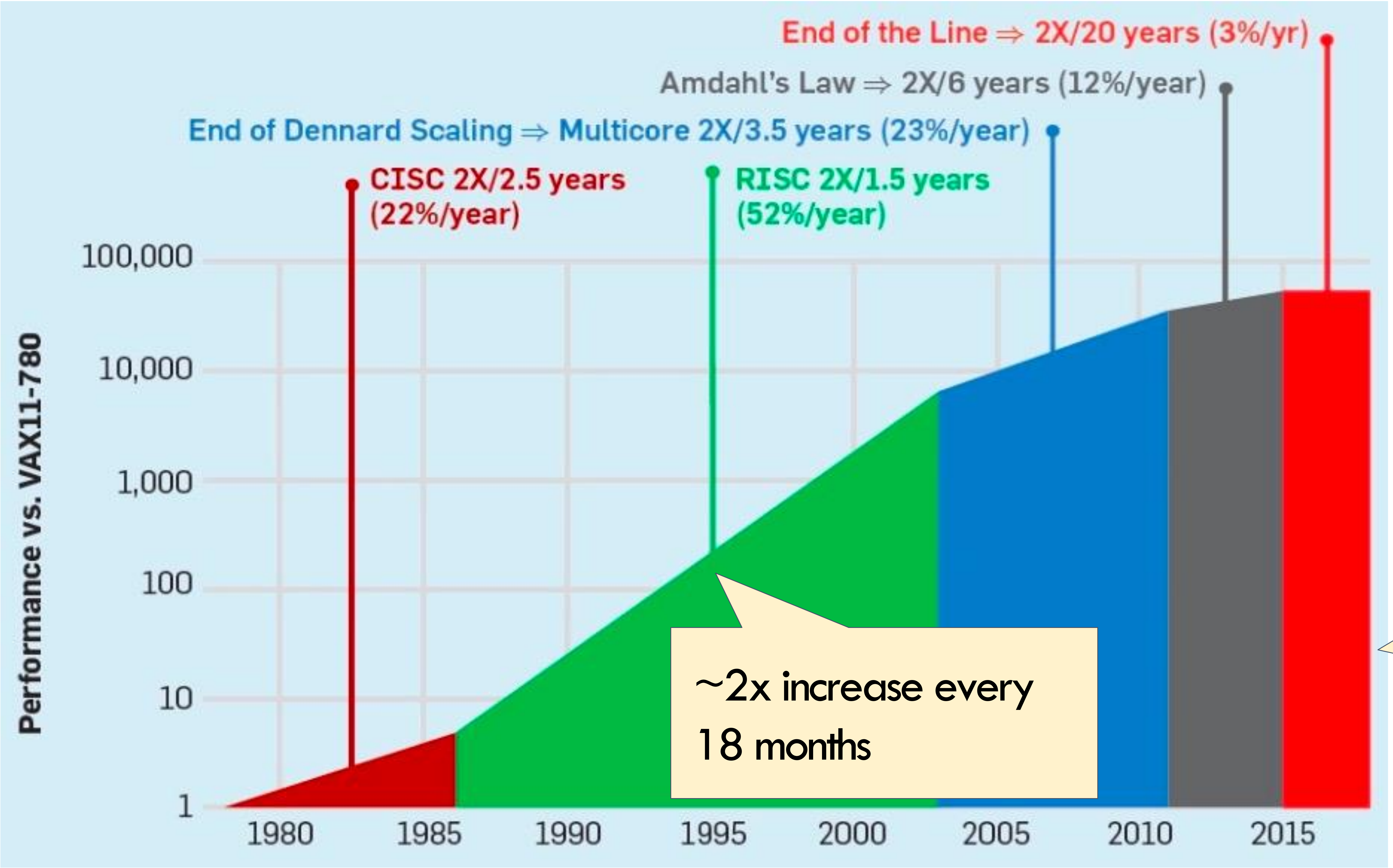
Operator optimization/compilation



Parallelization

- **Why Parallelization: Technology Trend**
- ML Parallelism Overview
- Collective Communication Review
- Data parallelism
- Model parallelism
  - Inter and intra-op parallelism
- Auto-parallelization

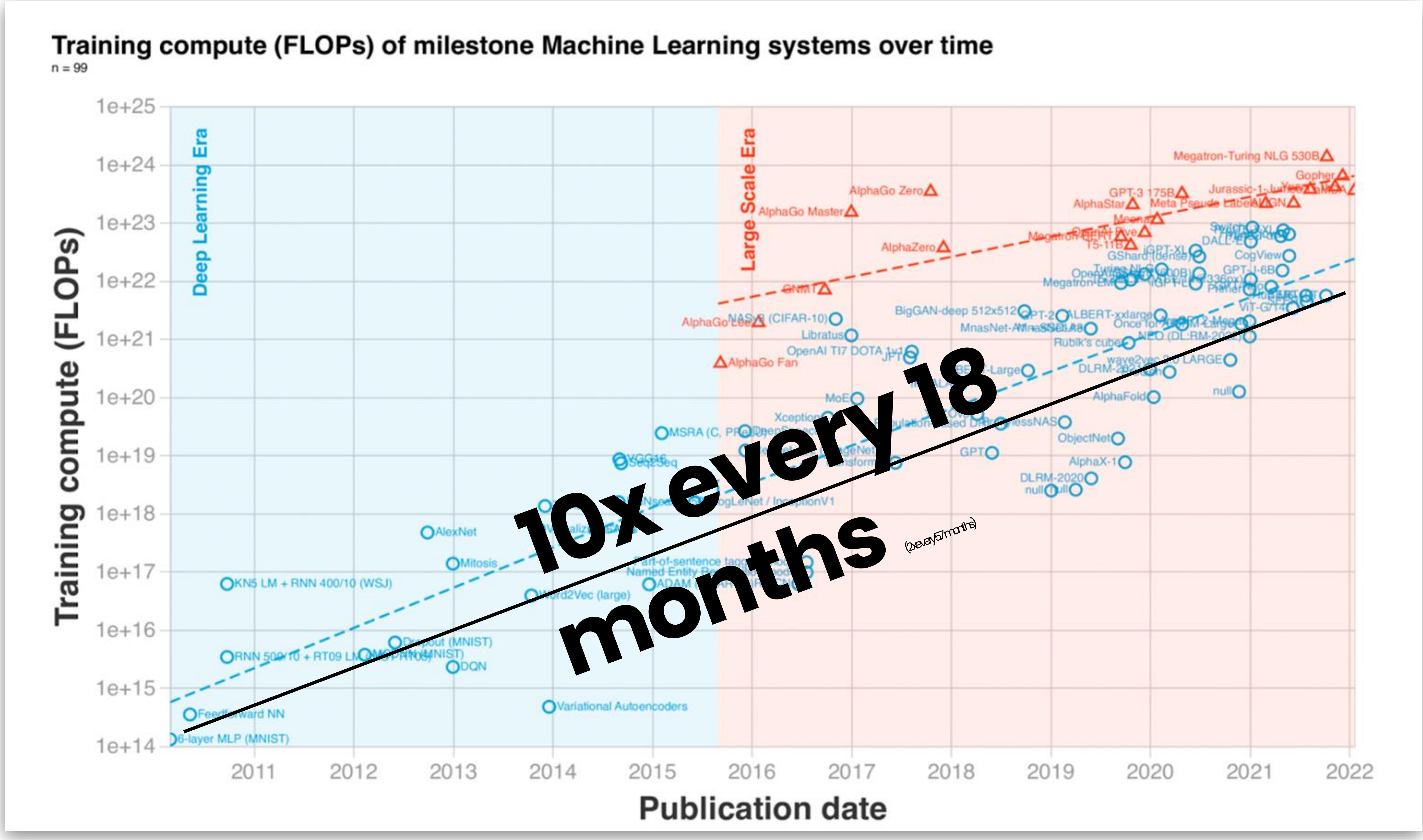
Moore's Law coming to an end



~2x increase every 18 months

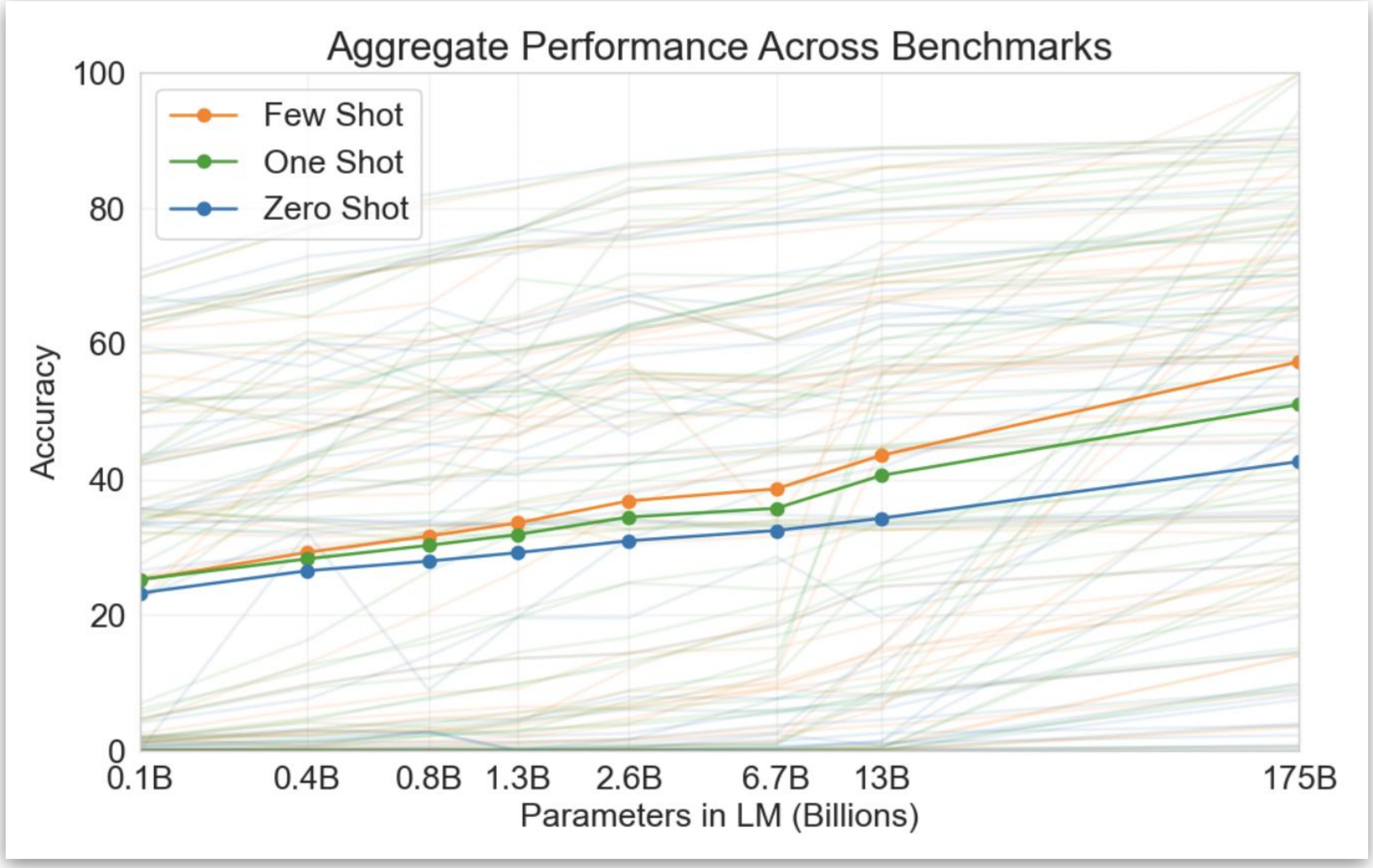
~1.05x increase every 18 months

Reality Check: ML Trend



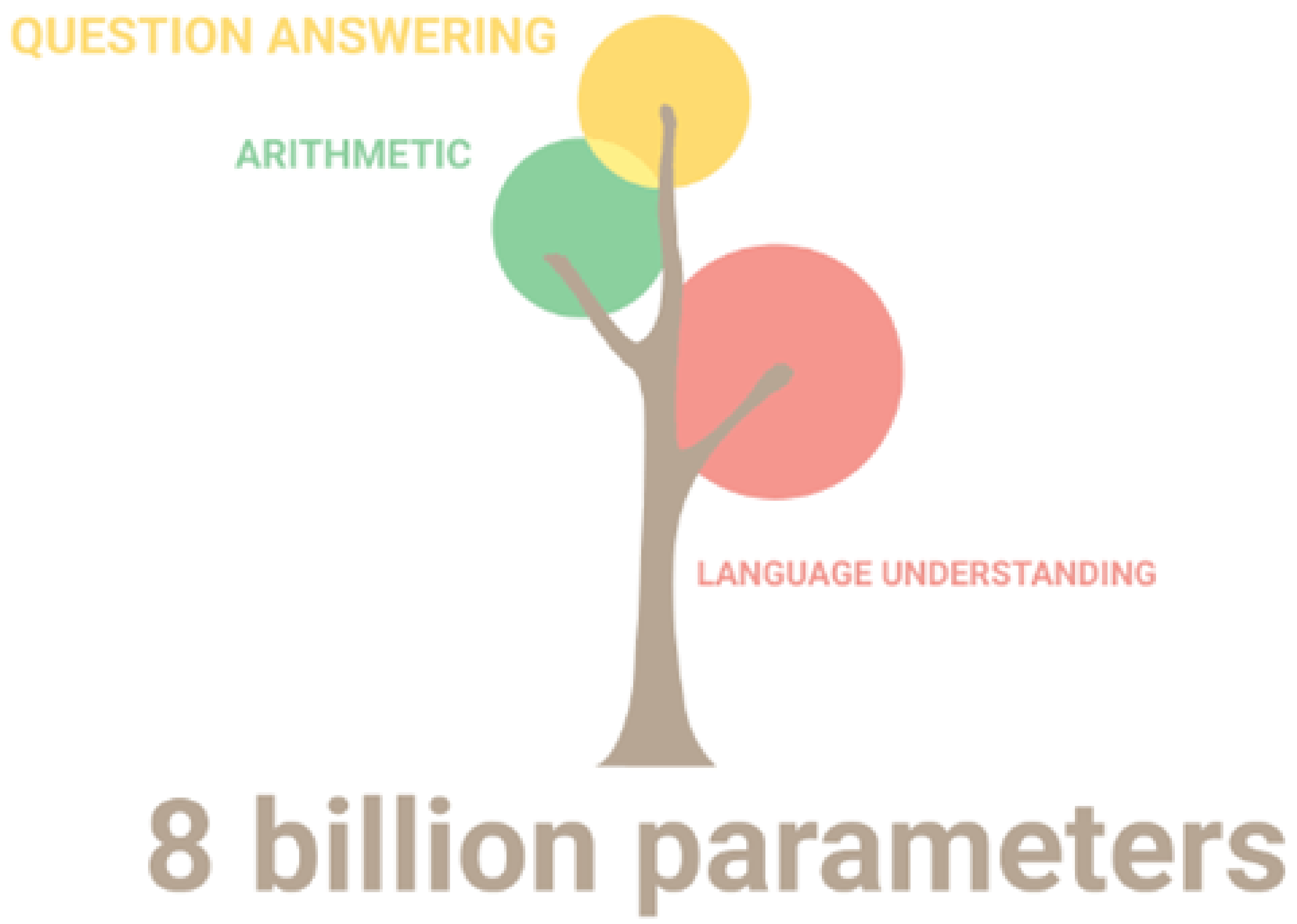
"Compute trends across three eras of machine learning", J. Sevilla, https://ar5iv.labs.arxiv.org/html/2202.05924

Bigger model, better accuracy



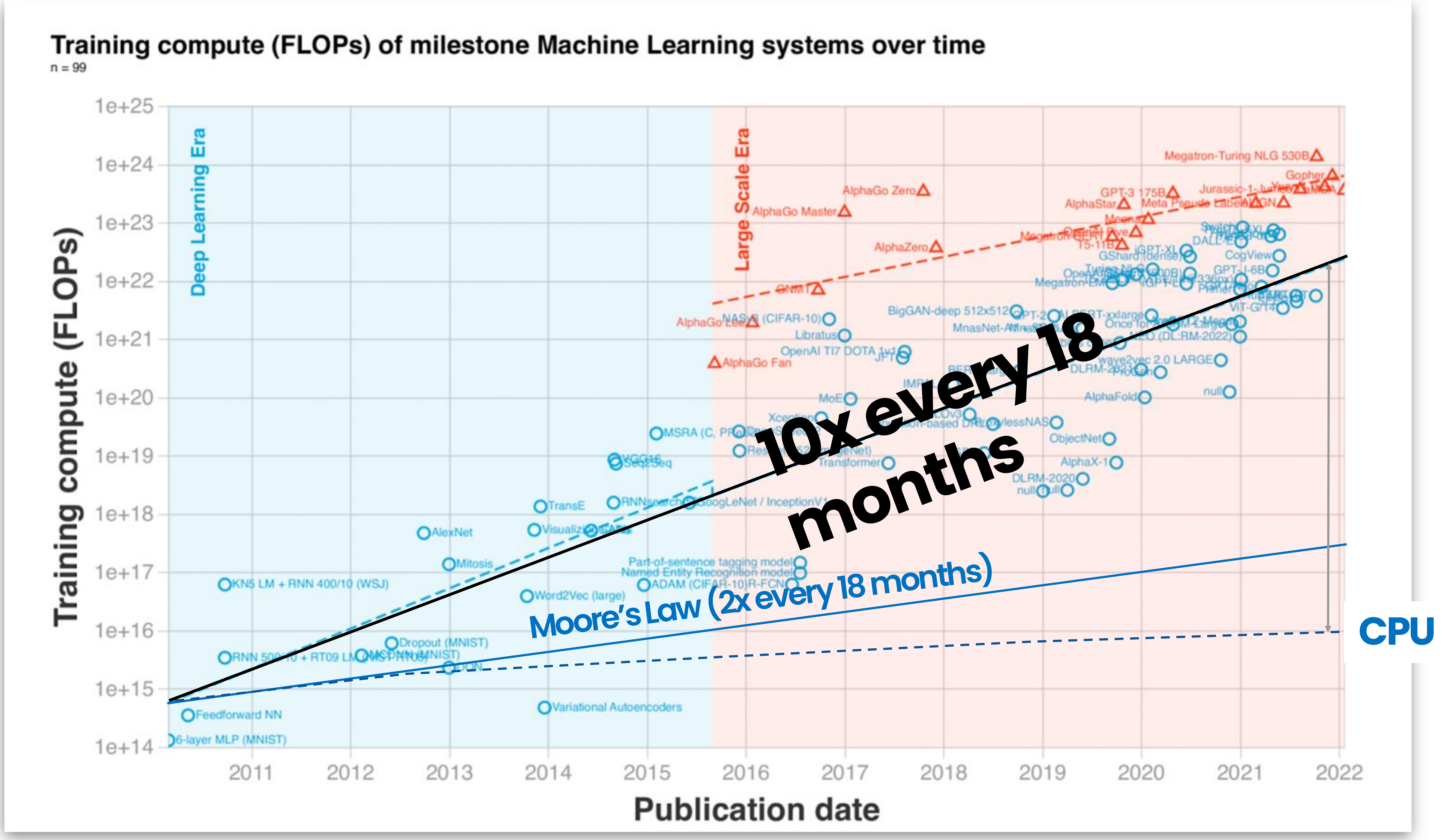
"Language Models are Few-Shot Learners", T. B. Brown et al., https://arxiv.org/pdf/2005.14165.pdf

Big Models have Emergent capabilities



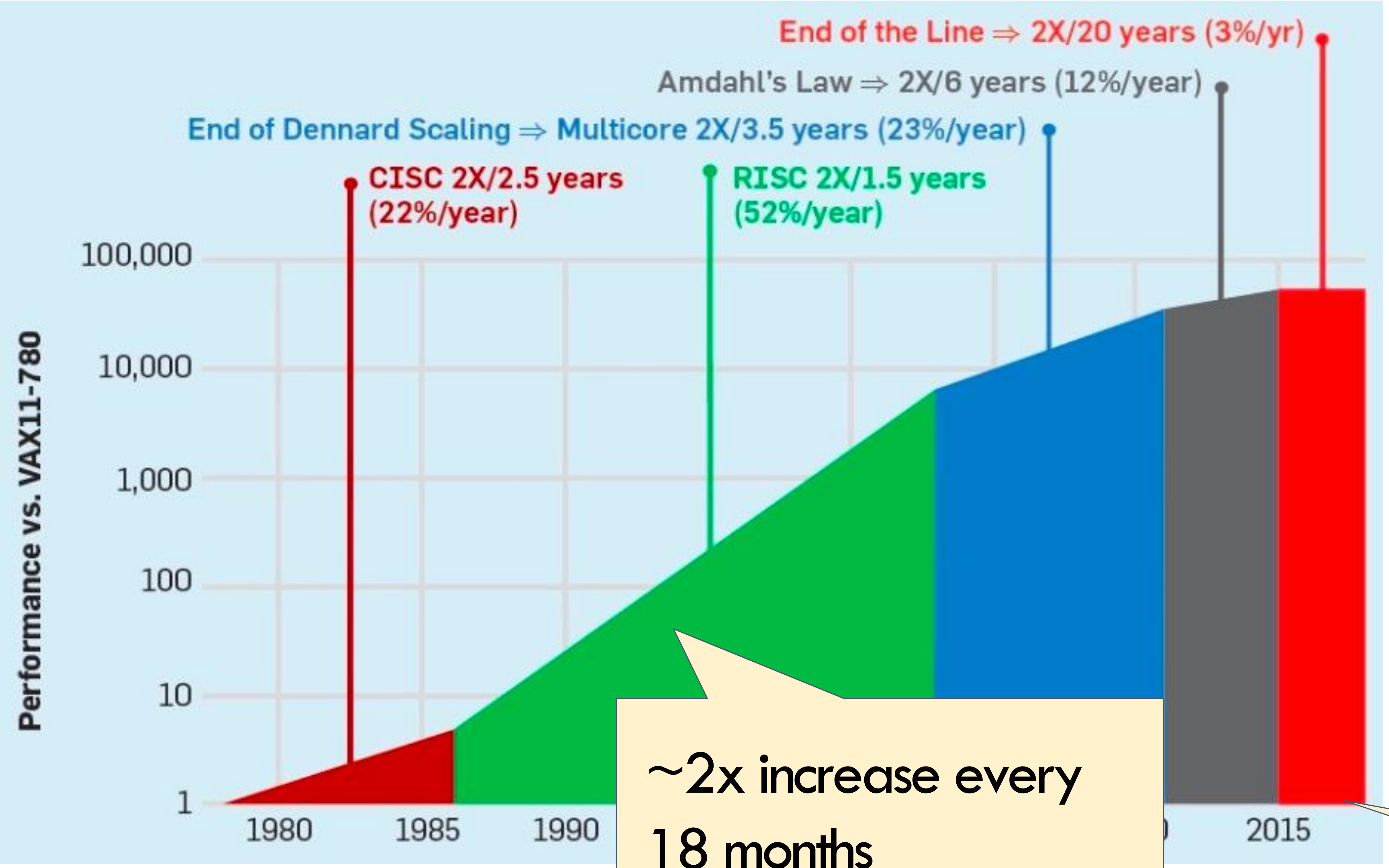
“Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance”,
S Narang, A Chowdhery et al, https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

Growing gap between demand and supply



"Compute trends across three eras of machine learning", J. Sevilla, https://ar5iv.labs.arxiv.org/html/2202.05924

Recap: Possible Paths Ahead



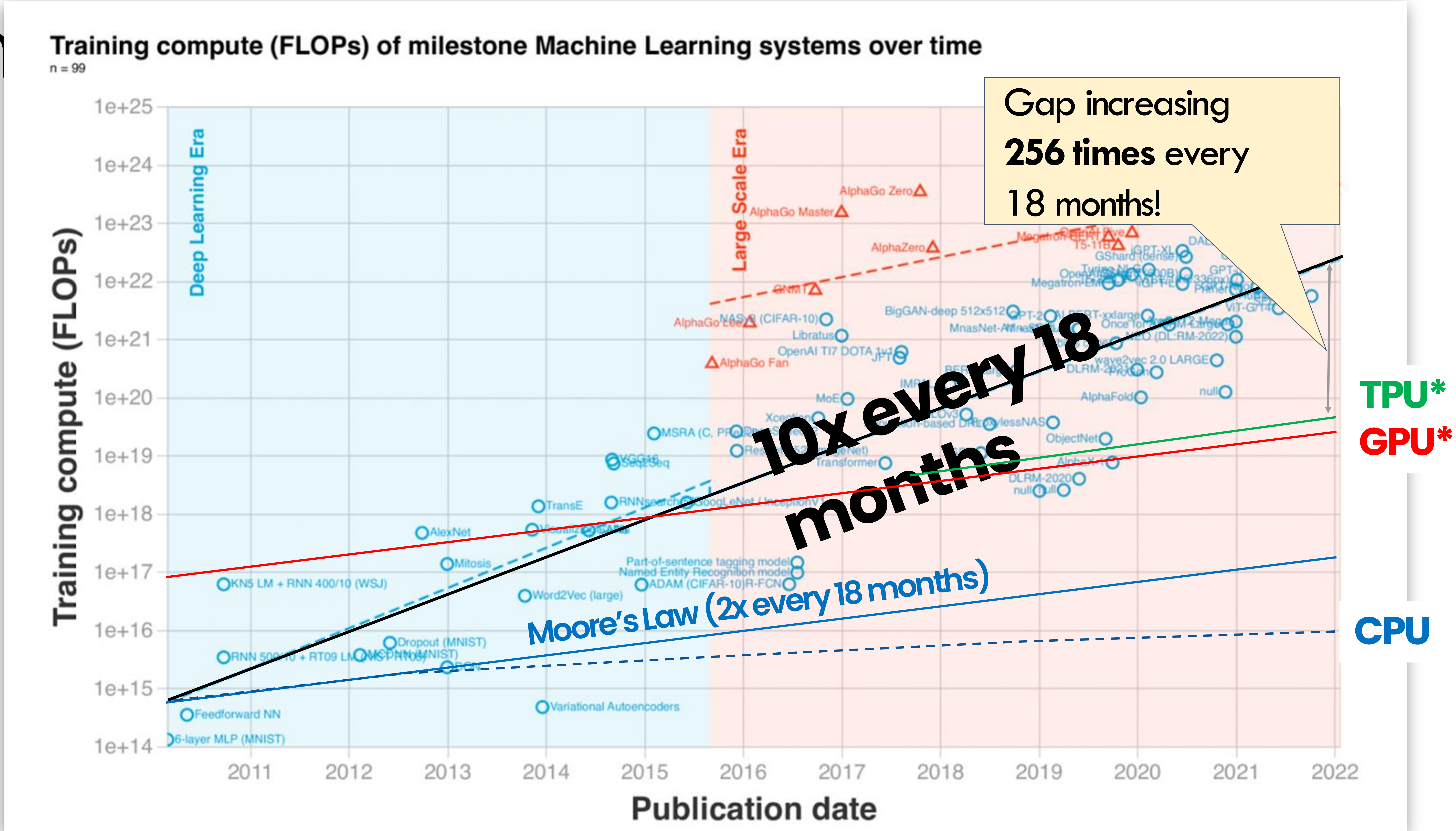
~2x increase every 18 months

~1.05x increase every 18 months

Option 1: Go to the quantum world

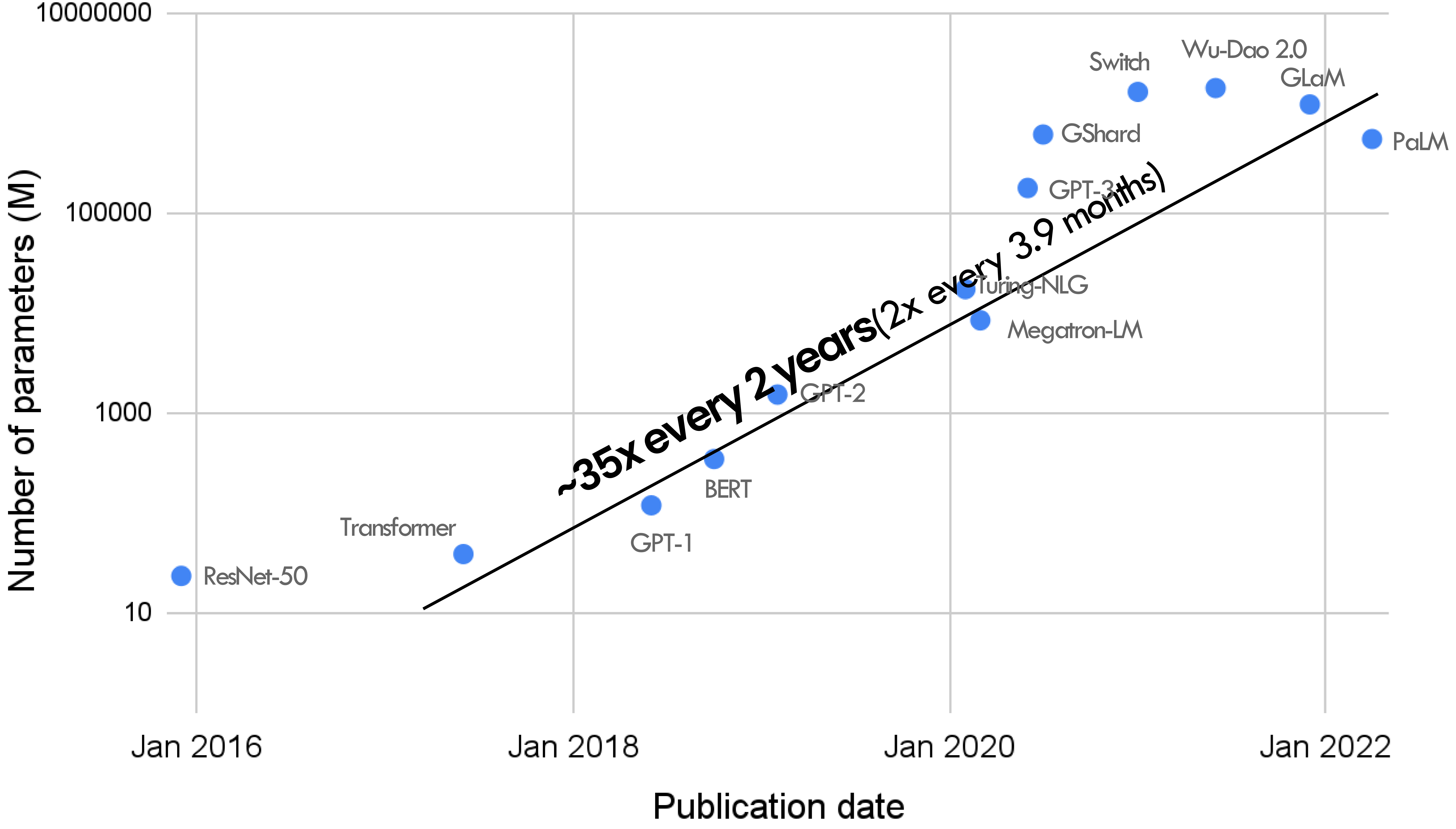
Option 2: Specialized hardware

Specialized hardware not good en

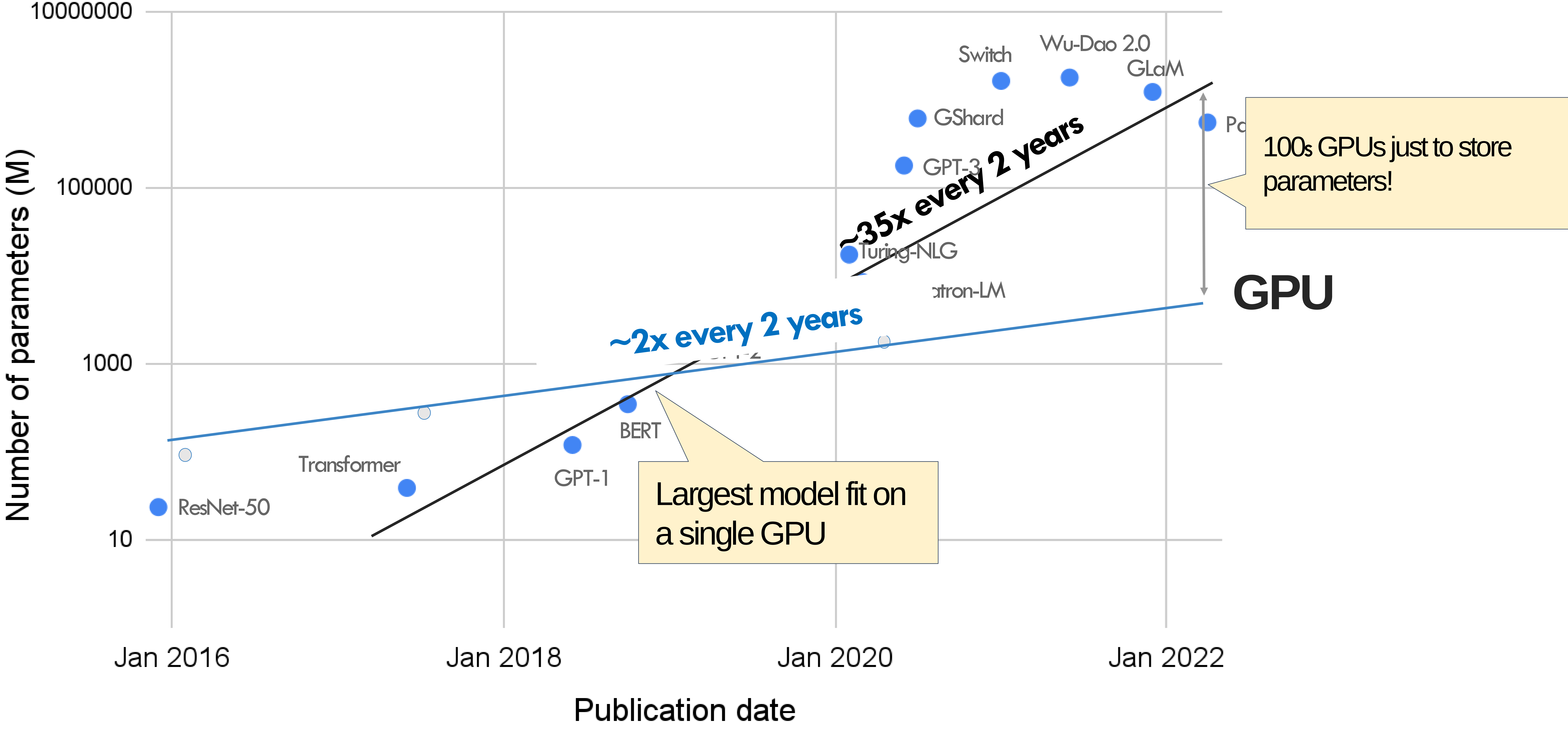


"Compute trends across three eras of machine learning", J. Sevilla, https://ar5iv.labs.arxiv.org/html/2202.05924

Not only compute, but memory



Growing gap between memory demand and supply

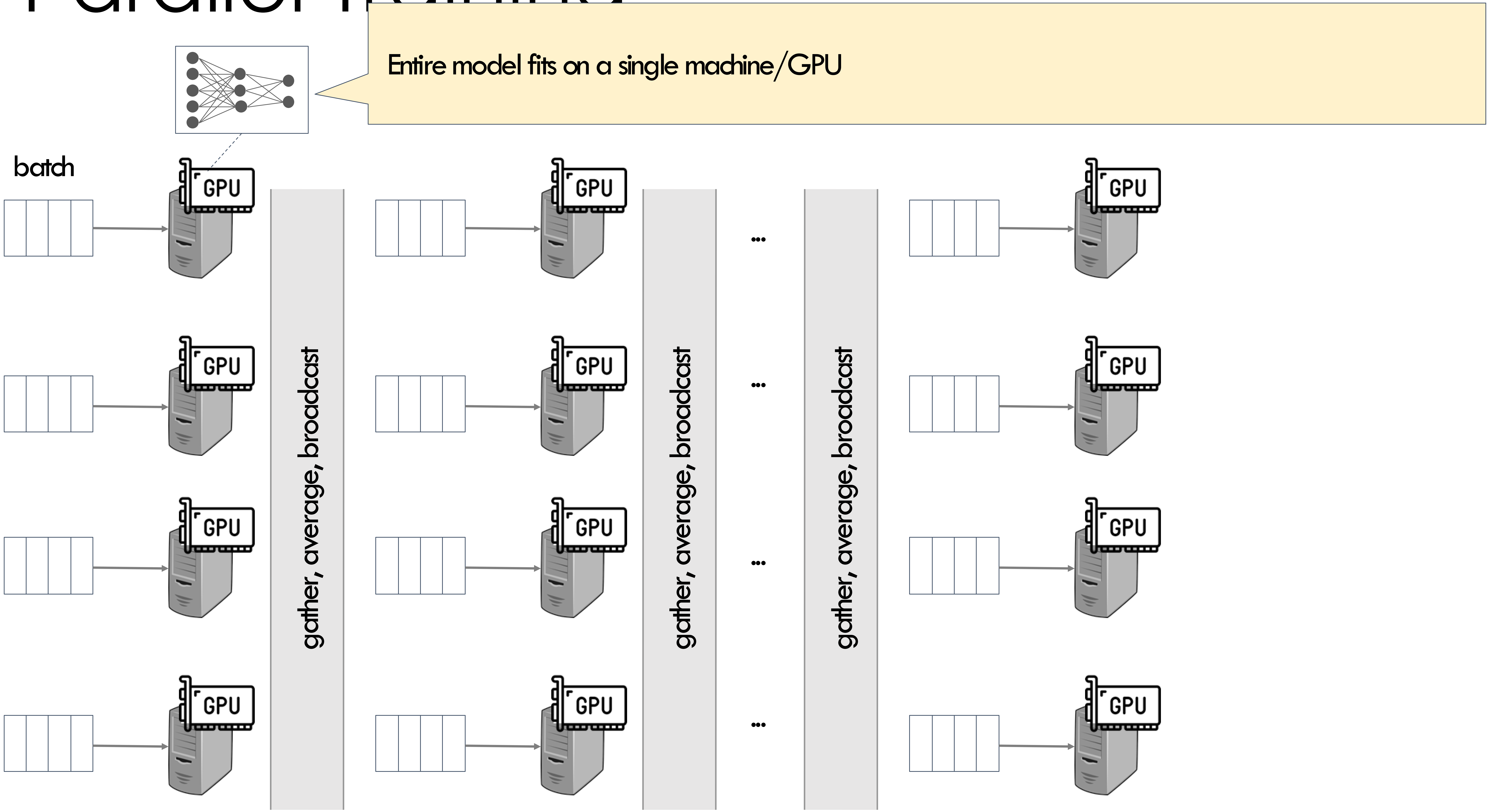


No way out but to parallelize these workloads !

Parallelization

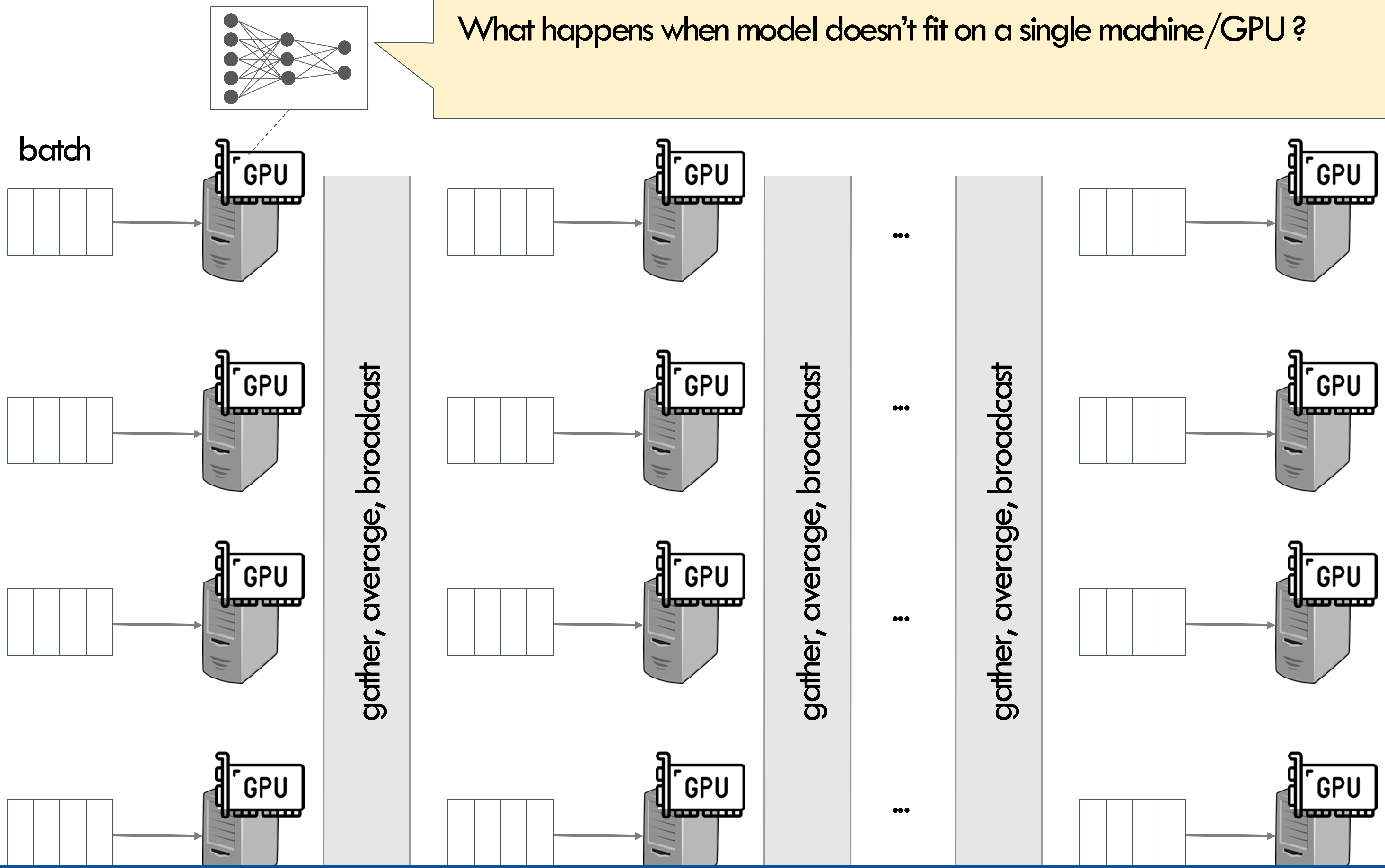
- Why Parallelization: Technology Trend
- **ML Parallelism Overview**
- Collective Communication Review
- Data parallelism
- Model parallelism
  - Inter and intra-op parallelism
- Auto-parallelization

Data Parallel Training



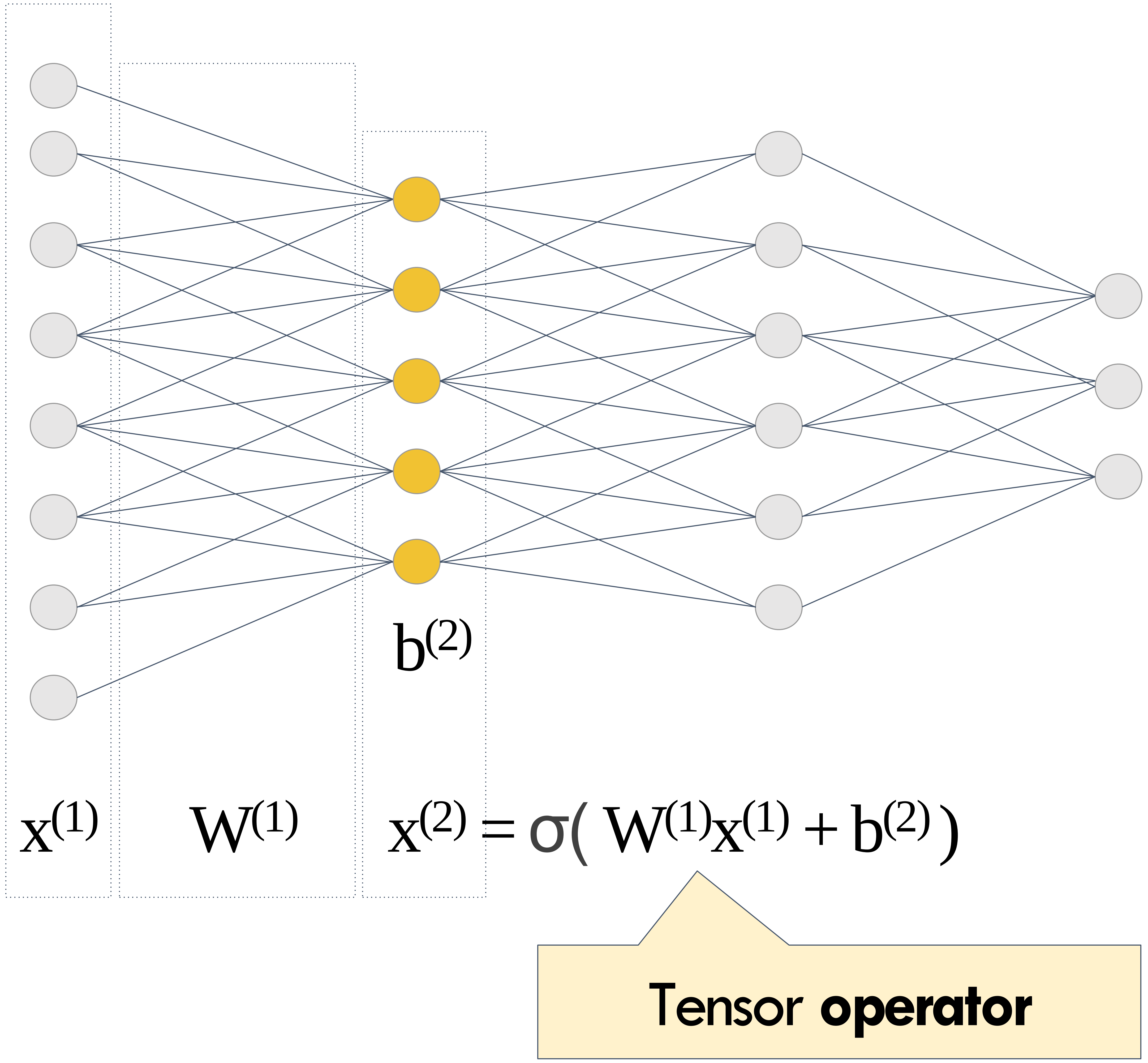
Data Parallel Training

What happens when model doesn't fit on a single machine/GPU ?

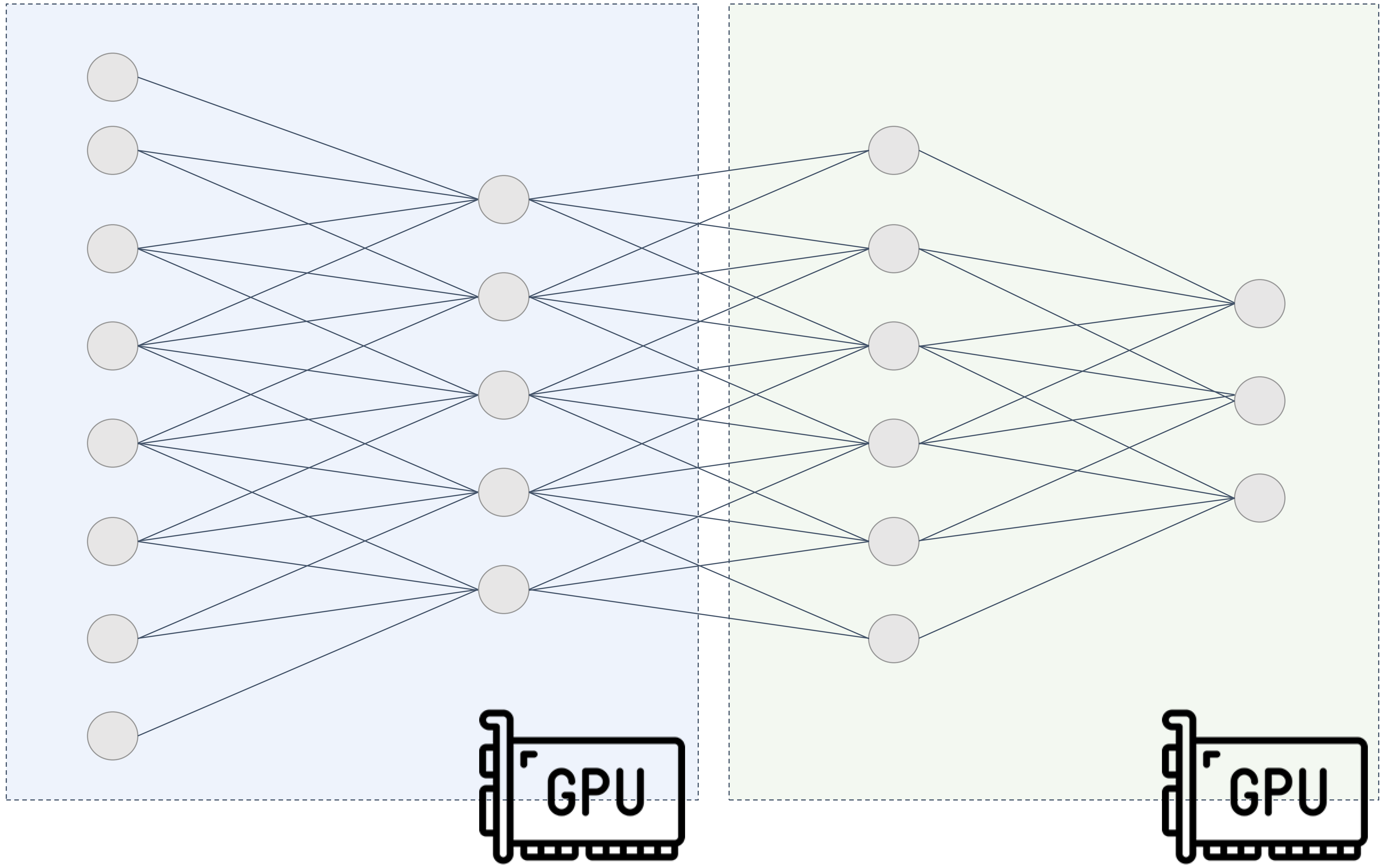


Need to parallelize the model itself

Need do parallelize the model, but how?

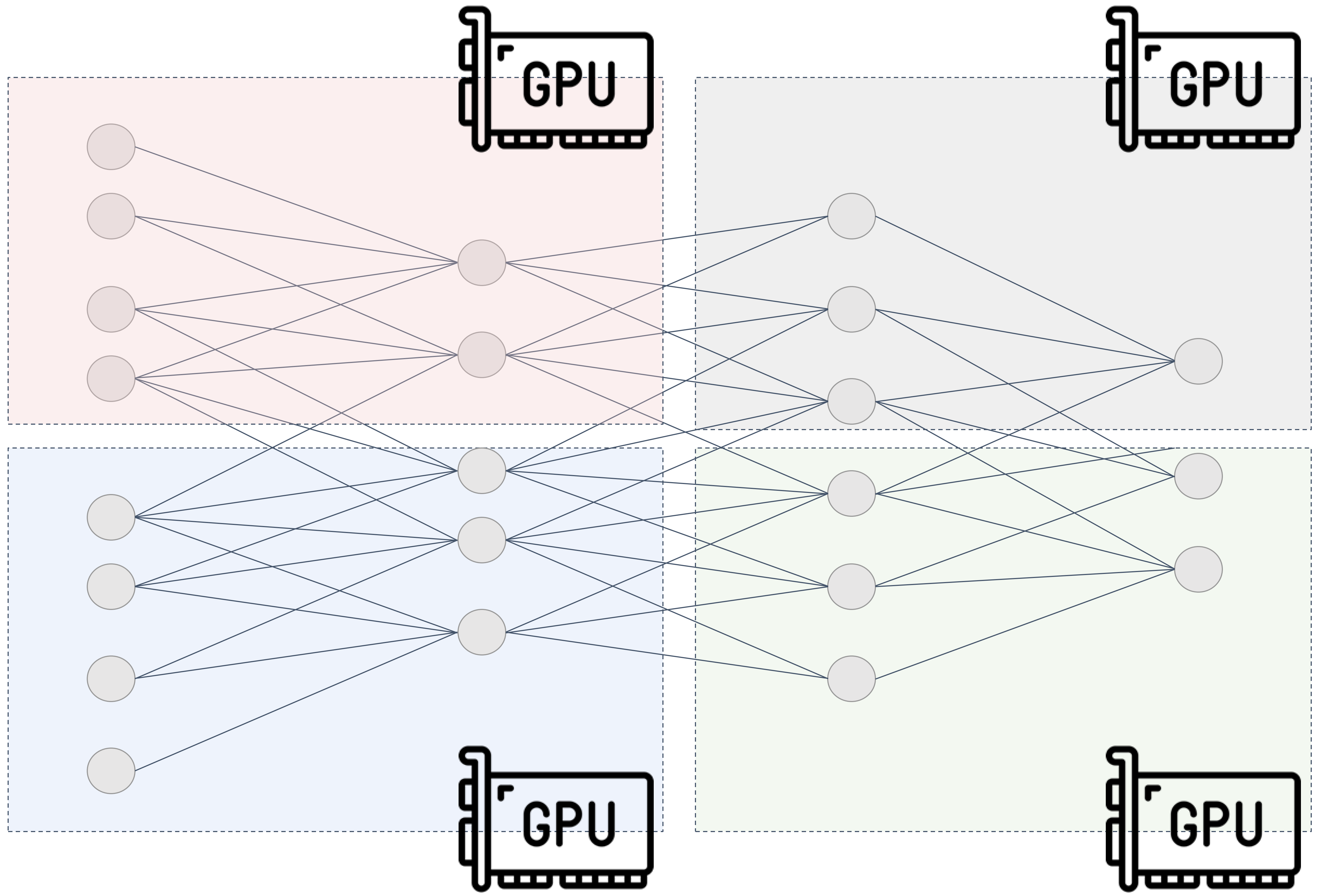


Model Parallelism



- Pipeline execution on both forward and backward paths
- GPUs can be on the same machine or **different** machines

Model Parallelism



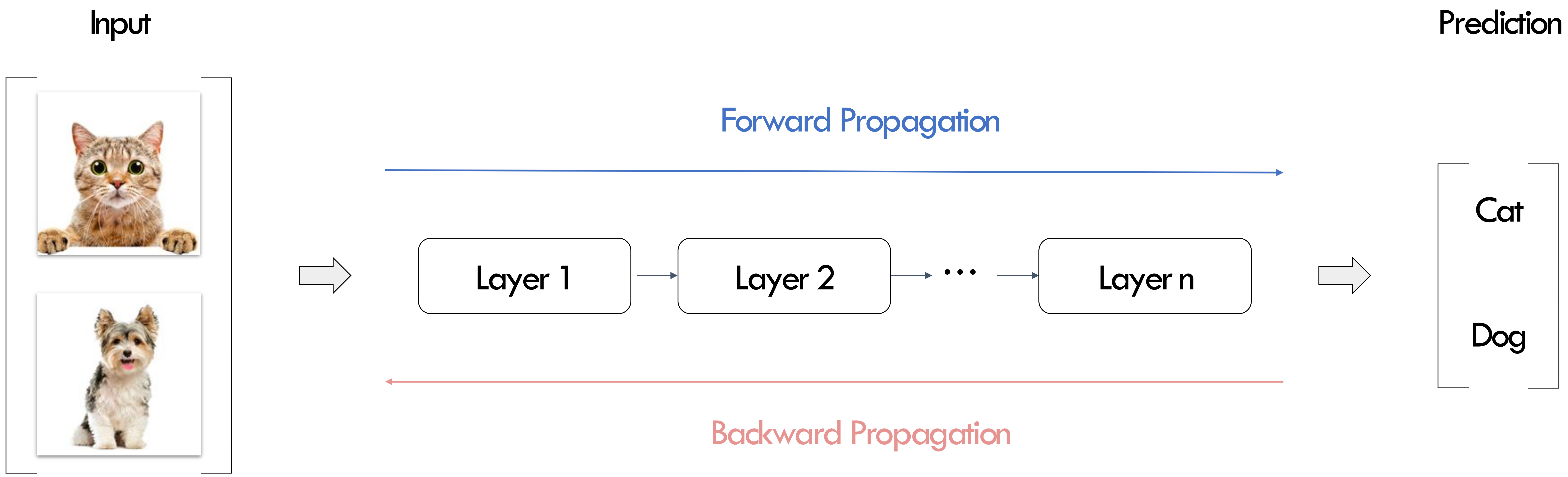
Classic View of ML Parallelisms

Classic view

Data parallelism

Model parallelism

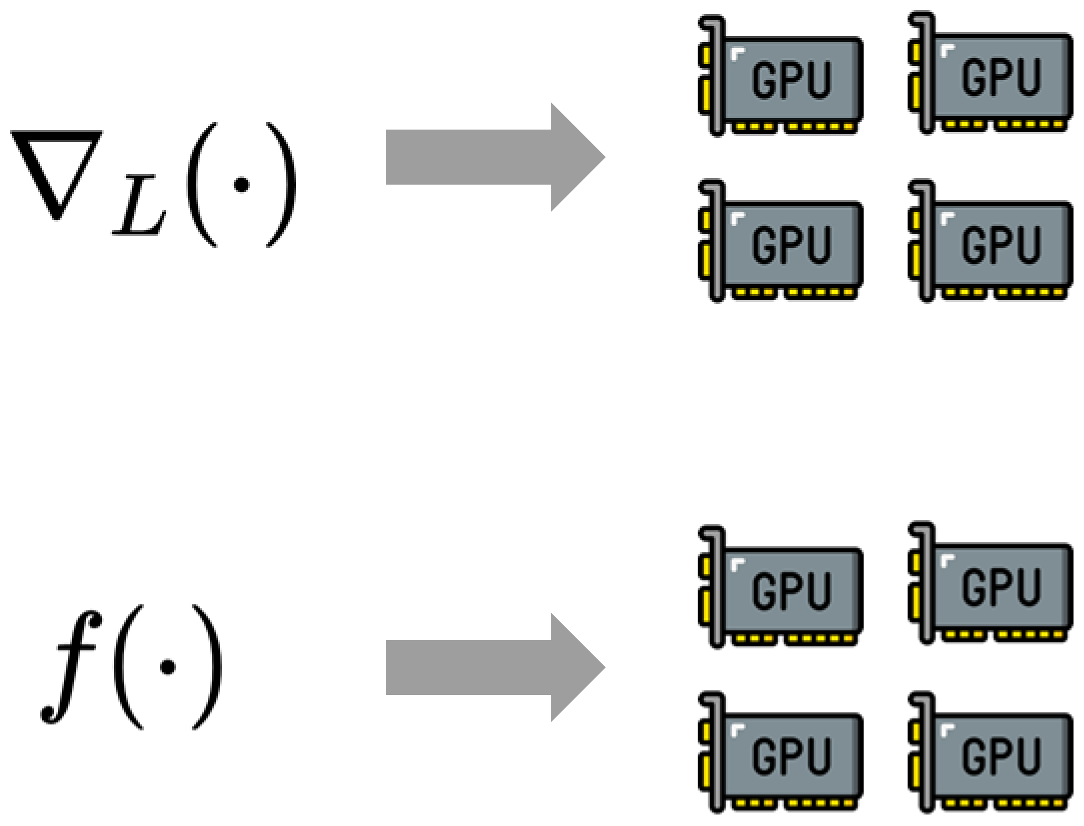
From a Computational Perspective

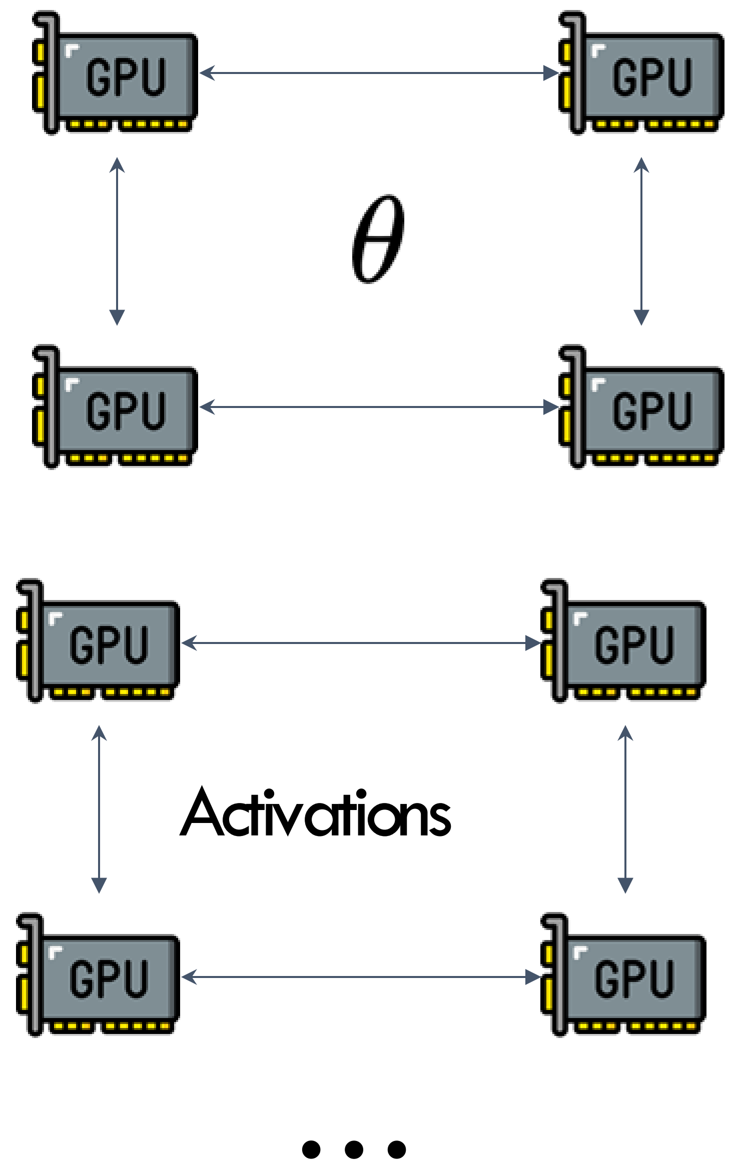


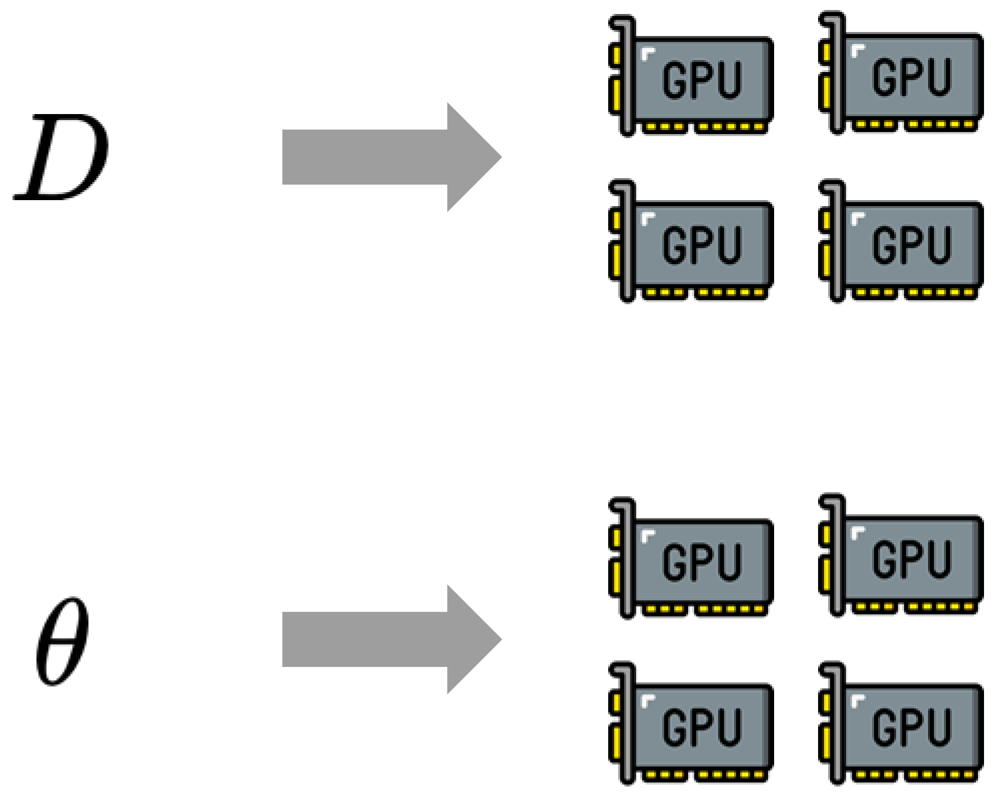
$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update}\\\text{(sgd, adam, etc.)}}}\big(\theta^{(t)},\ \underbrace{\nabla_L}_{\substack{\text{model}\\\text{(CNN, GPT, etc.)}}}\big(\theta^{(t)},\ \underbrace{D^{(t)}}_{\text{data}}\big)\big)$$

From a Computational Perspective

Computing	**Communication**	**Memory**





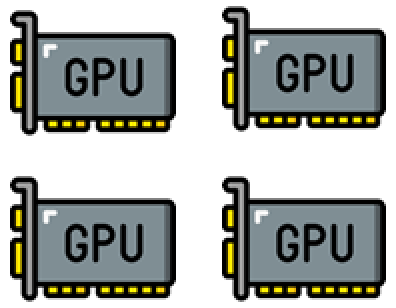


$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update}\\ \text{(sgd, adam, etc.)}}}\big(\theta^{(t)}, \underbrace{\nabla_L}_{\substack{\text{model}\\ \text{(CNN, GPT, etc.)}}}\big(\theta^{(t)}, \underbrace{D^{(t)}}_{\text{data}}\big)\big)$$

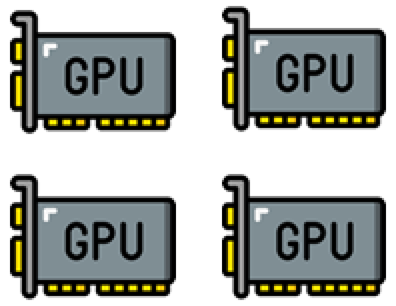
Data and Model Parallelism

Data parallelism

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

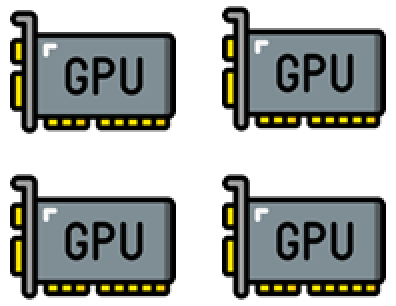


$\theta,\ \nabla_L(\cdot),\ f(\cdot) \xrightarrow{\text{replicate}}$

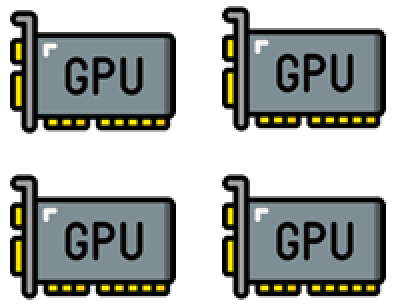


Model parallelism

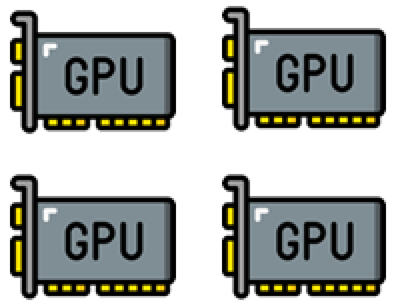
$\theta,\ \nabla_L(\cdot) \xrightarrow{\text{partition}}$



$D \xrightarrow{?}$



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



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

parameter

weight update
(sgd, adam, etc.)

model
(CNN, GPT, etc.)

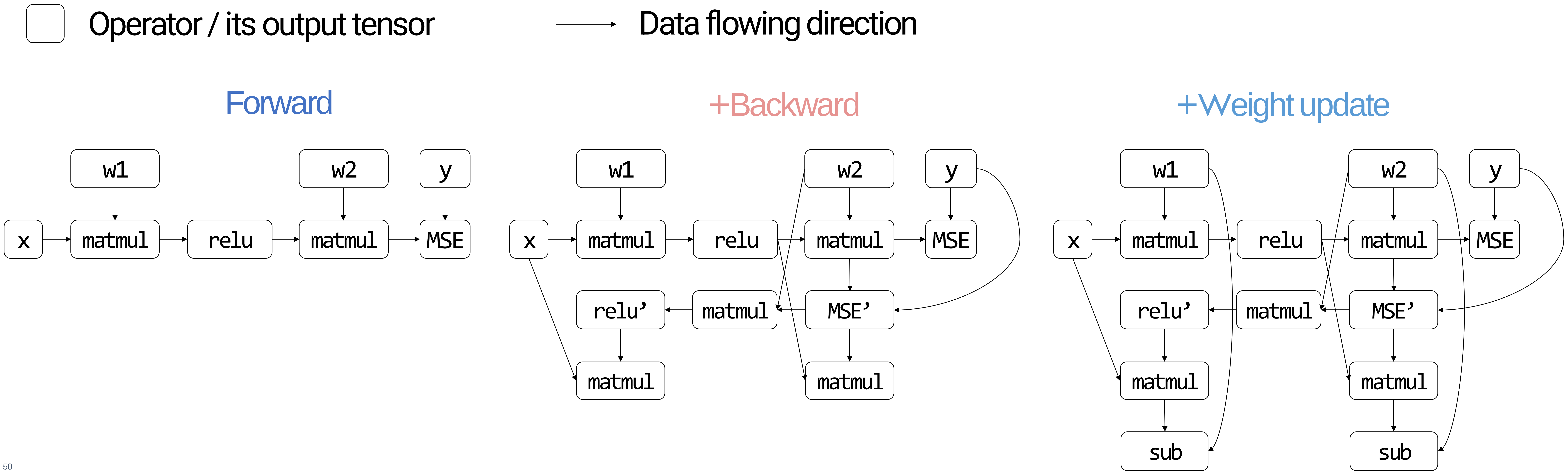
data

Recap: Computational Graph

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

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

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



Device Cluster

Nvidia DGX with V100

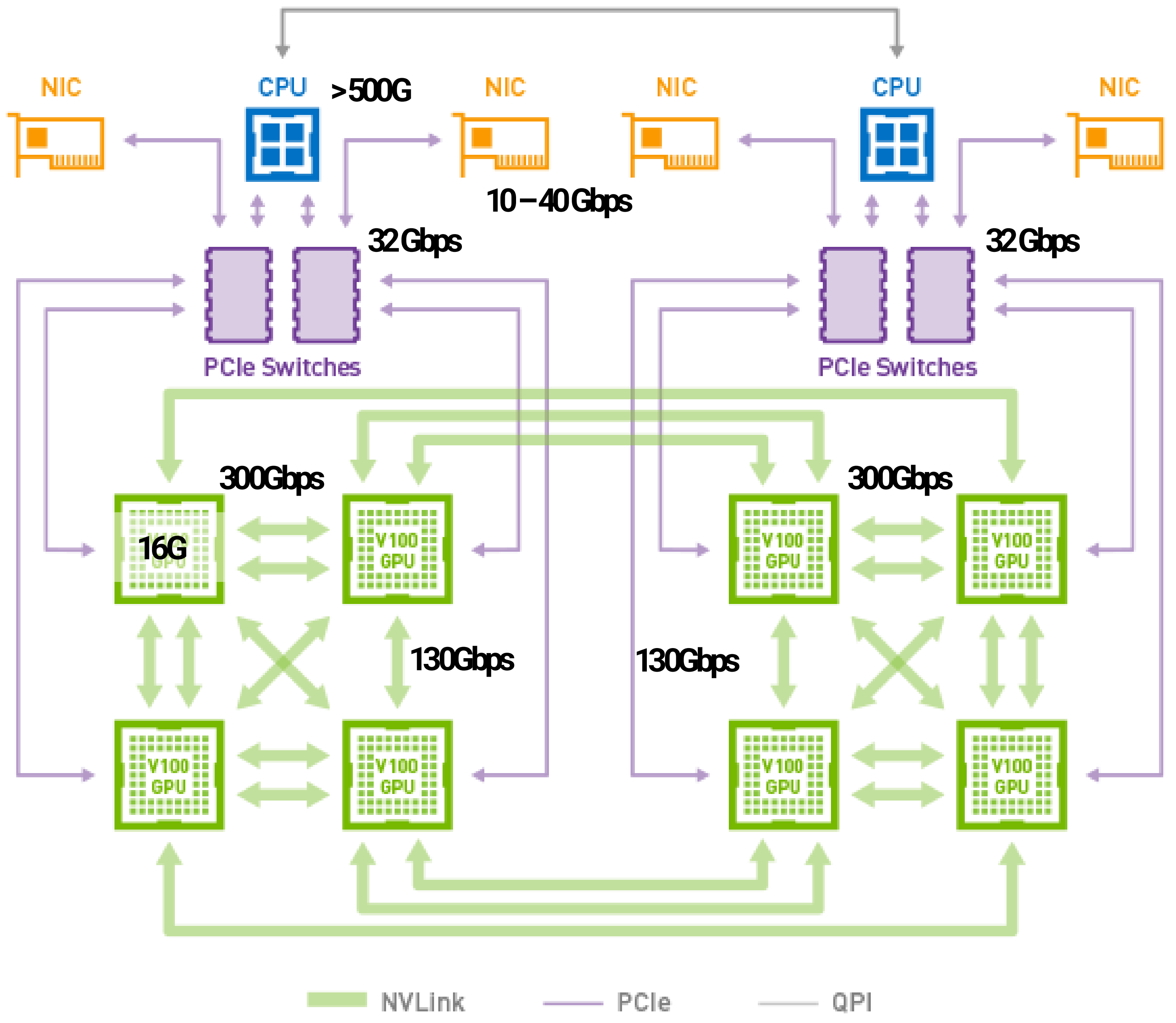
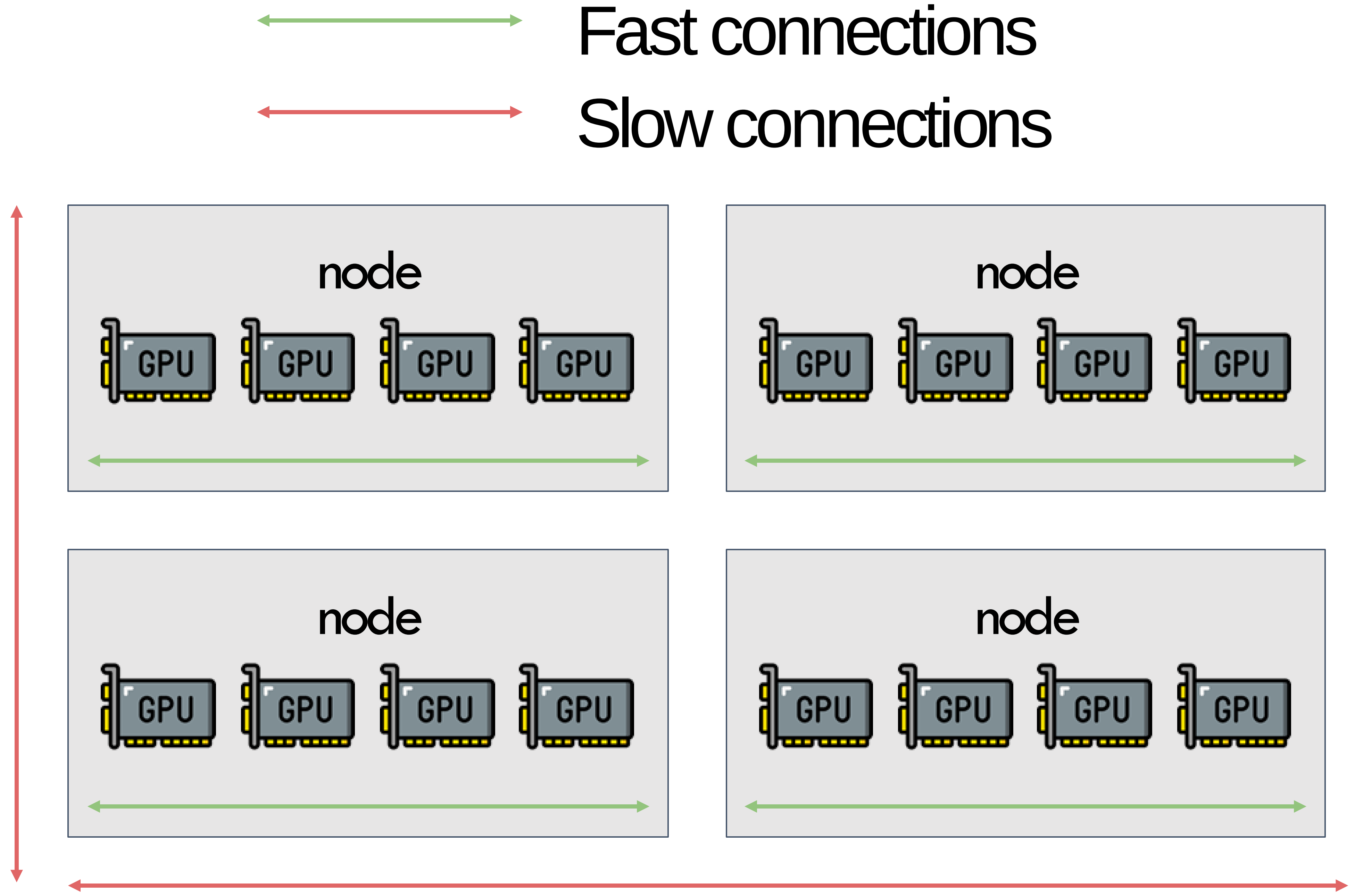


Figure from NVIDIA

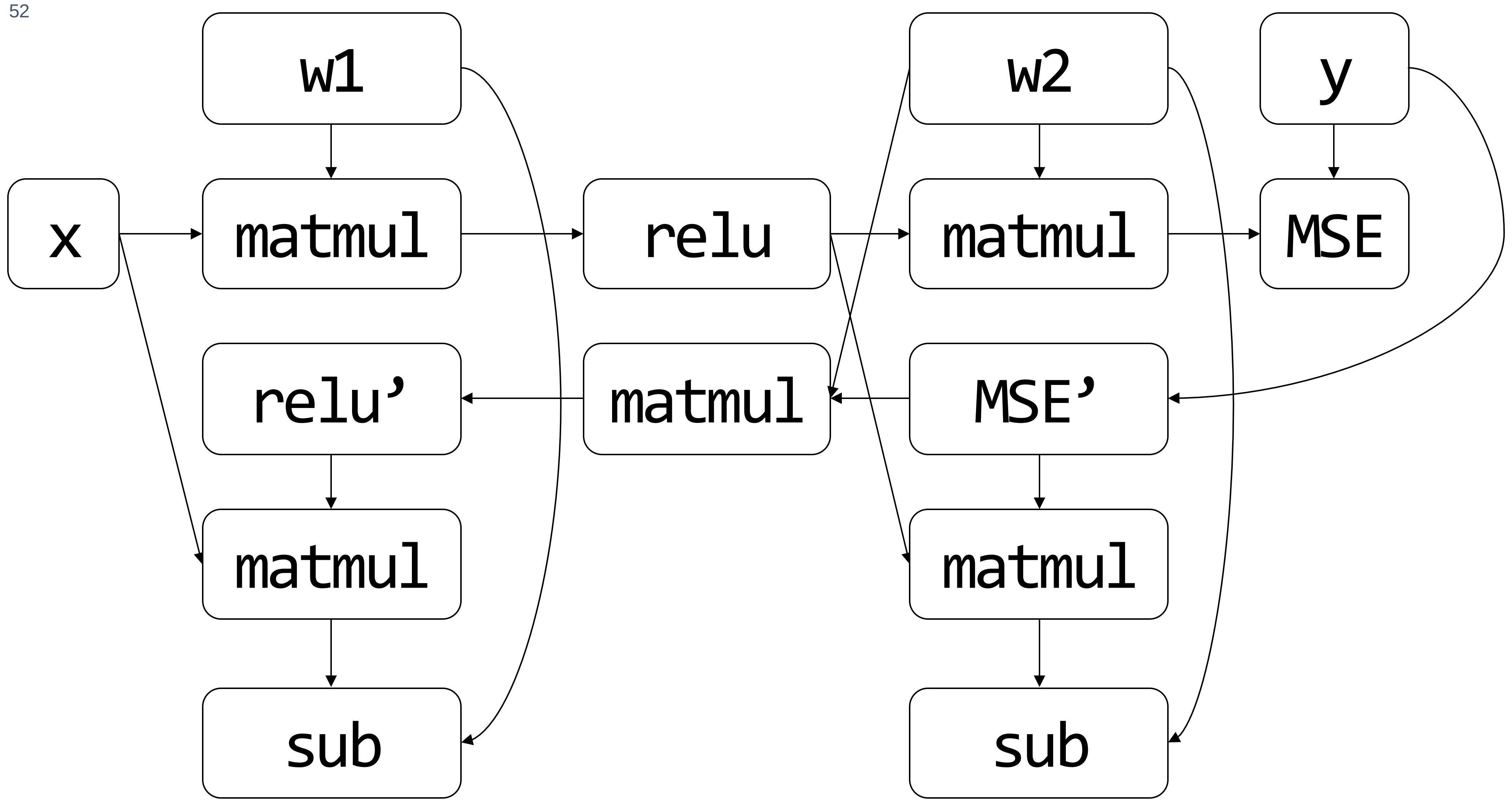
A typical GPU cluster topology

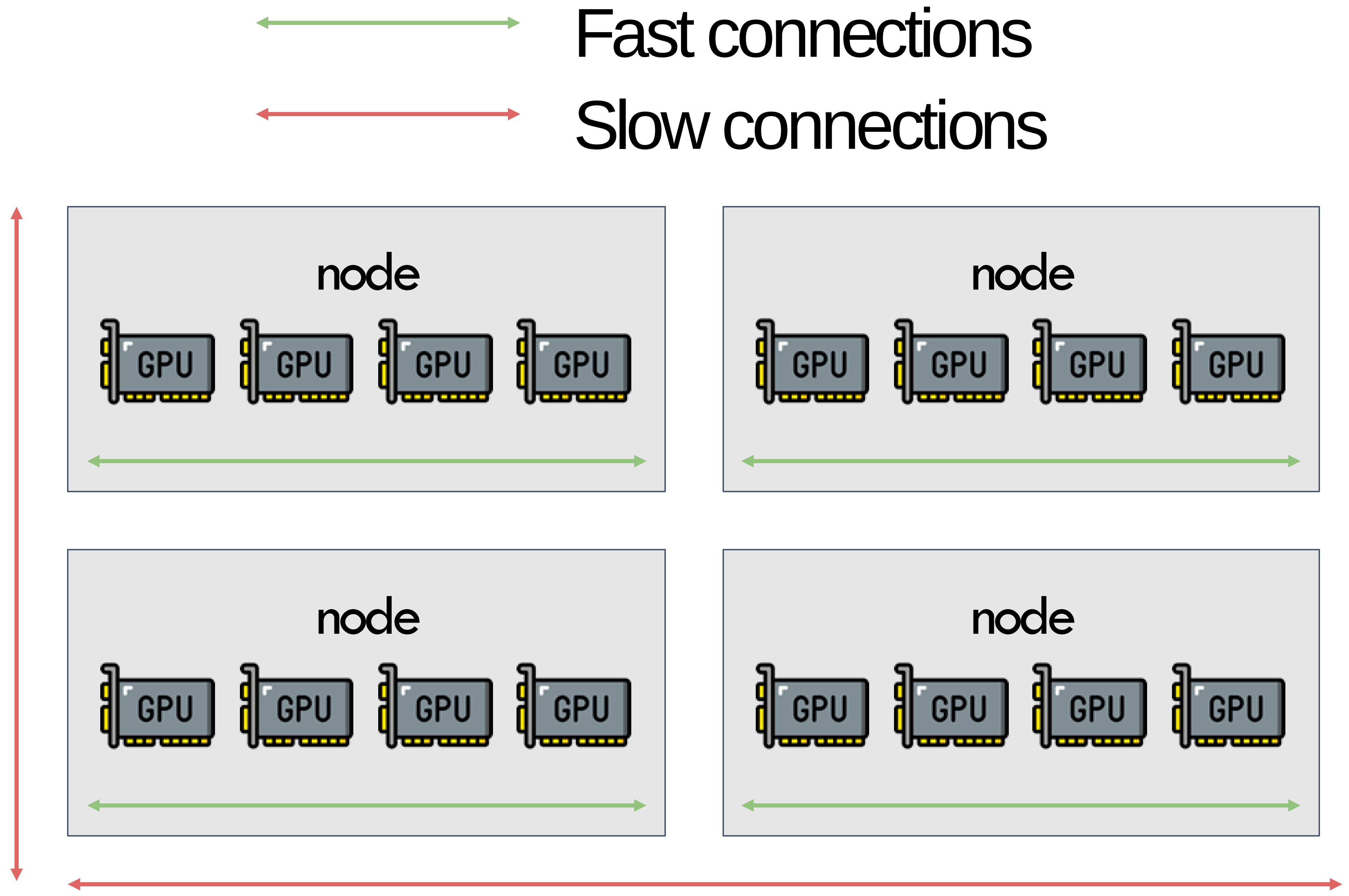


Parallelization = Partitioning Computation Graph on Device Cluster

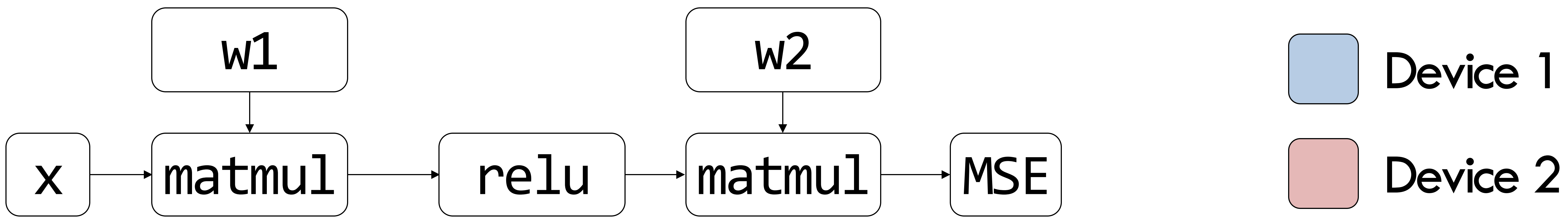
How to partition the computational graph on the device cluster?

52

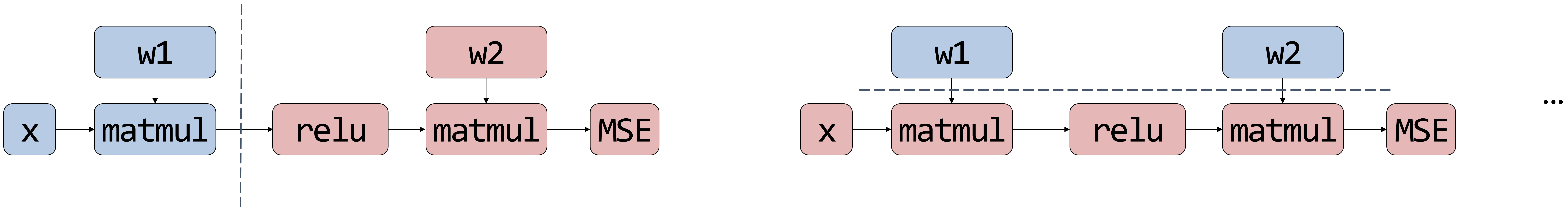




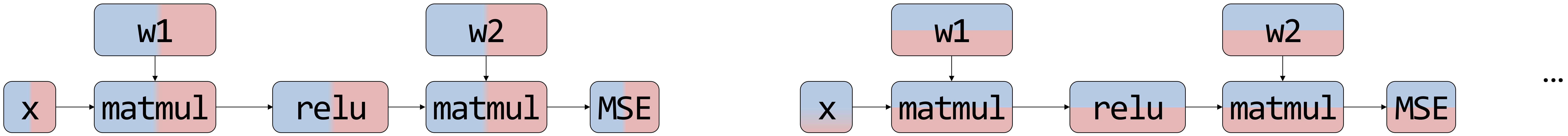
Partitioning Computation Graph



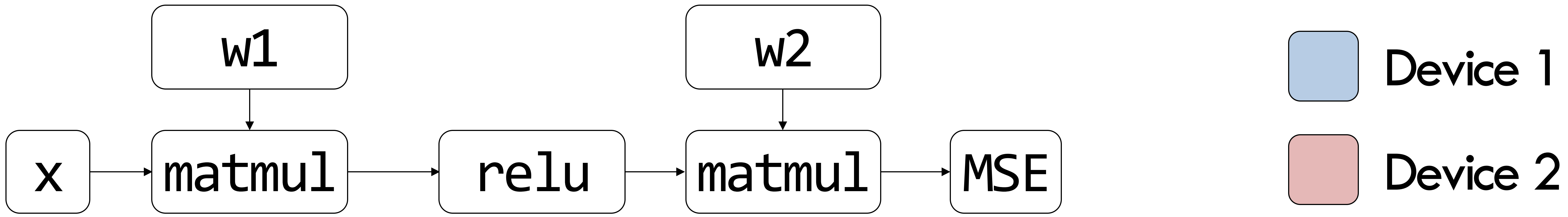
Strategy 1



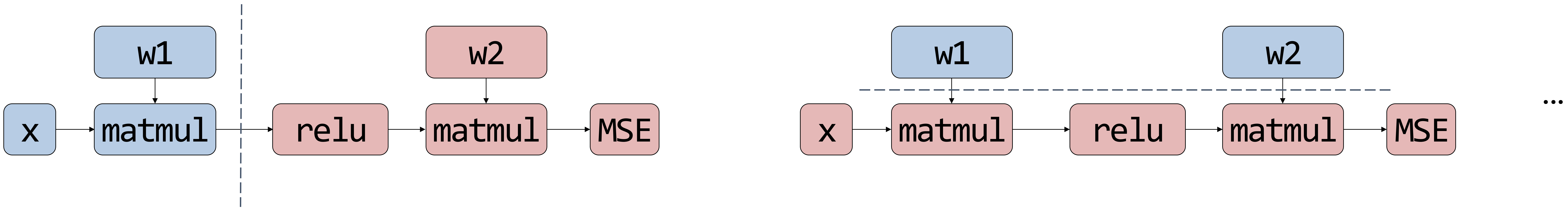
Strategy 2



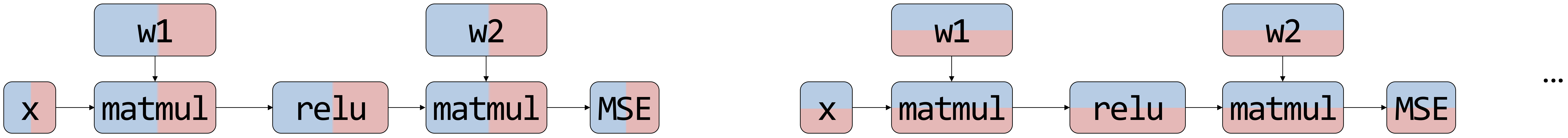
Partitioning Computation Graph



Strategy 1: Inter-operator Parallelism



Strategy 2: Intra-operator Parallelism

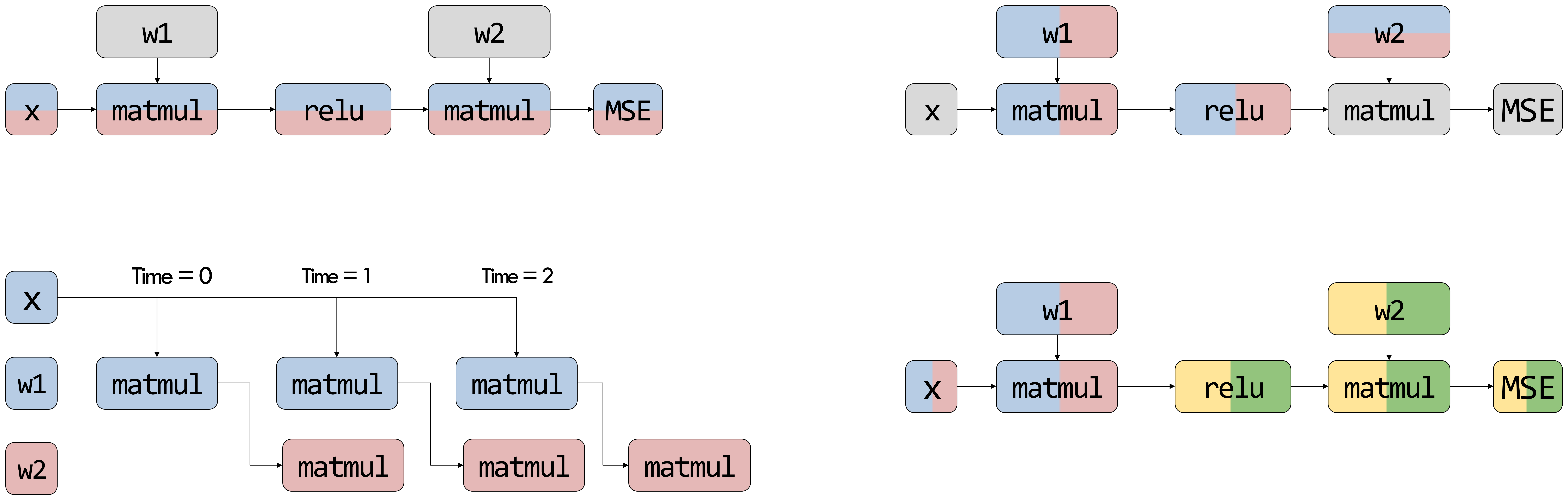


More Parallelisms...

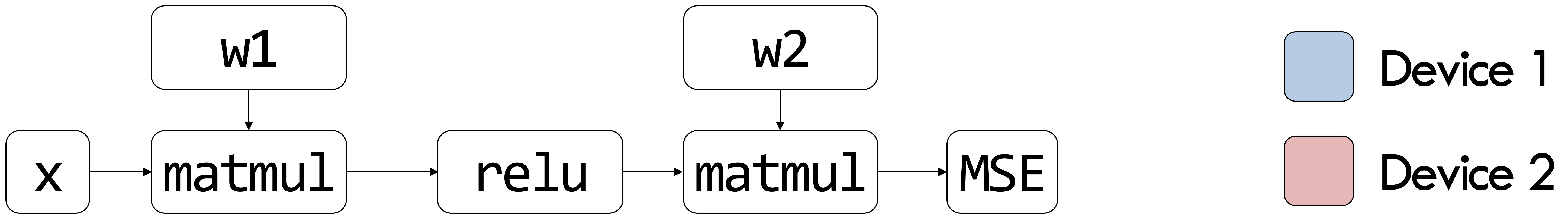
Multiple intra-op strategies for a single node

Row-partitioned Column-partitioned Replicated Device 3 Device 4

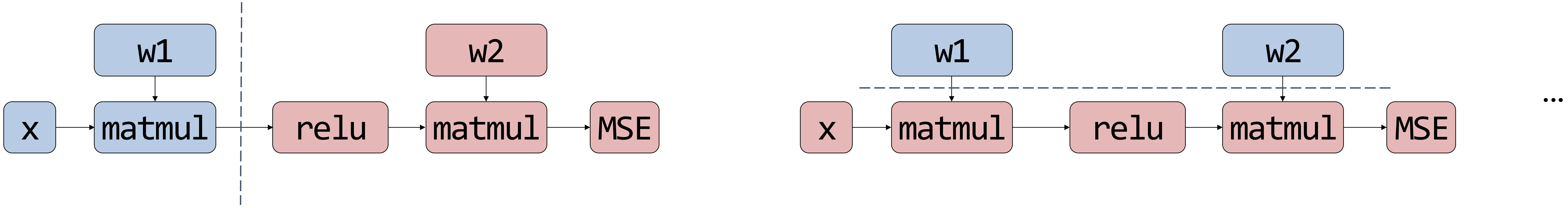
More strategies



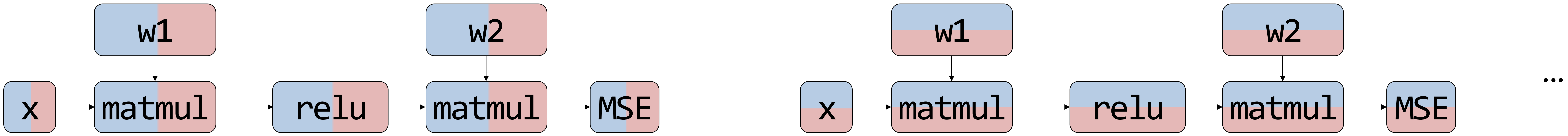
Summary: Inter-op and Intra-op Parallelisms



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

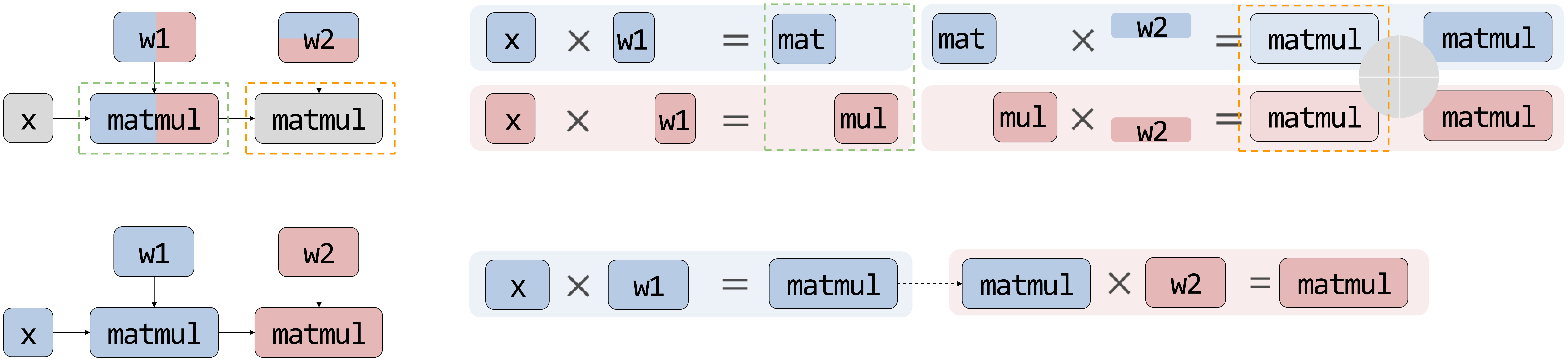


Intra-op parallelism: Assign different regions of a single operator to different devices.

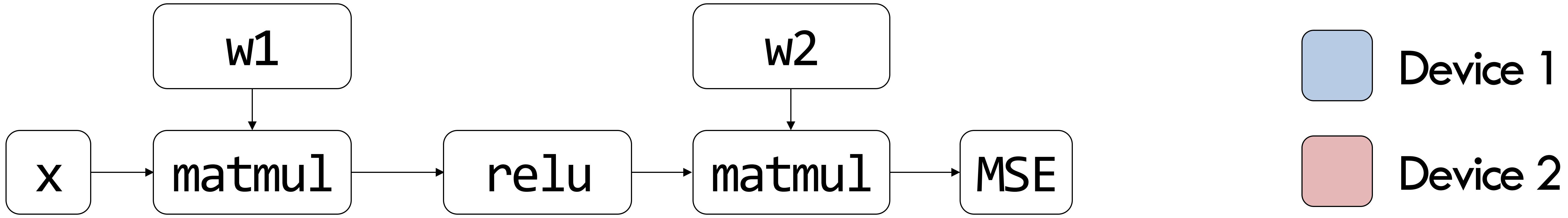


Inside Intra- and Inter-op Parallelism

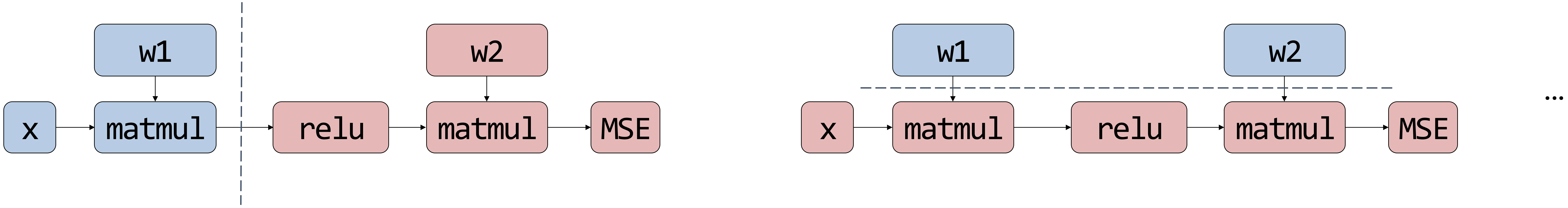
$$Y = X \cdot W_1 \cdot W_2 = X \cdot \begin{bmatrix} W_1^{d1} & W_1^{d2} \end{bmatrix} \cdot \begin{bmatrix} W_2^{d1} \\ W_2^{d2} \end{bmatrix}$$



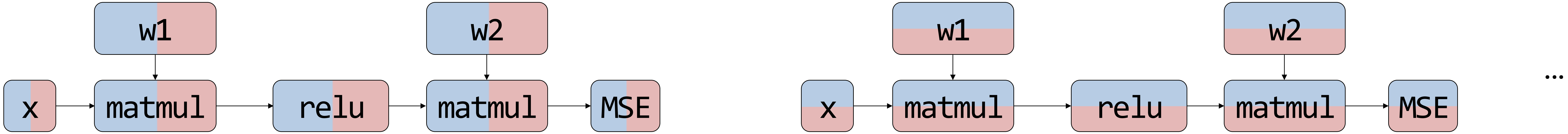
Inter-op and Intra-op Parallelism: Characteristics



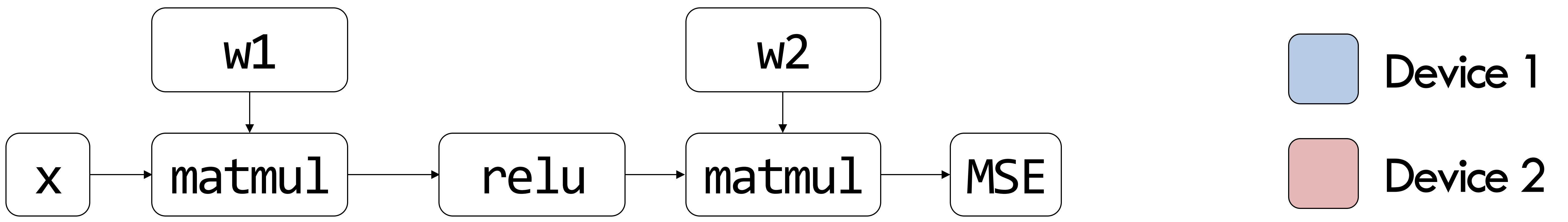
Inter-op parallelism: Requires point-to-point communication but results in device idle



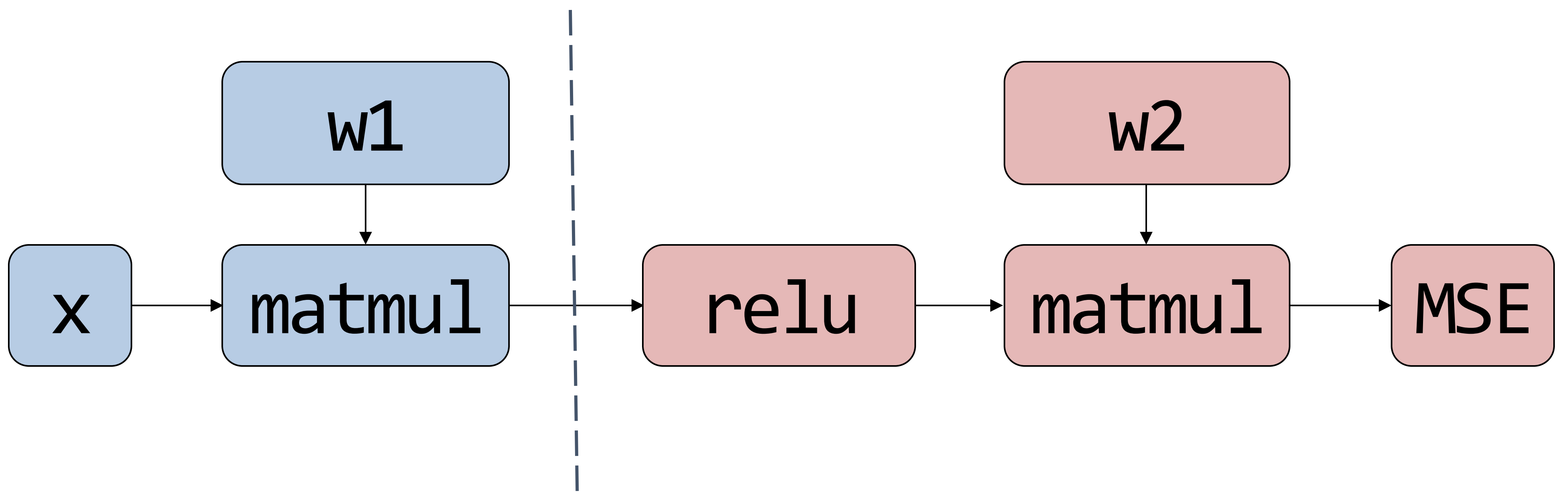
Intra-op parallelism: Devices are busy but requires collective communication



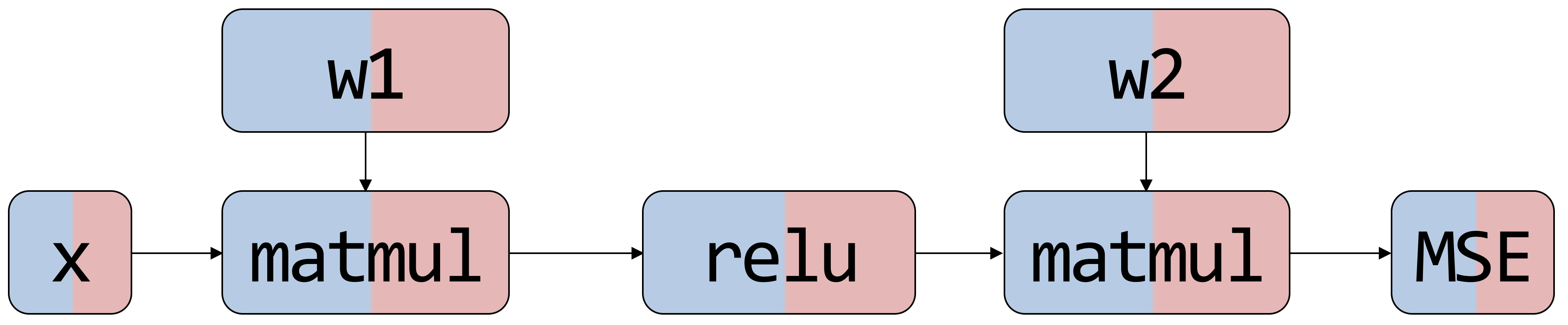
Inter-op and Intra-op Parallelism: Characteristics



Inter-op parallelism



Intra-op parallelism



Trade-off

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

Computational View of ML Parallelisms

Classic view	**New view (this tutorial)**
Data parallelism	Inter-op parallelism
Model parallelism	Intra-op parallelism

Two Views of ML Parallelisms

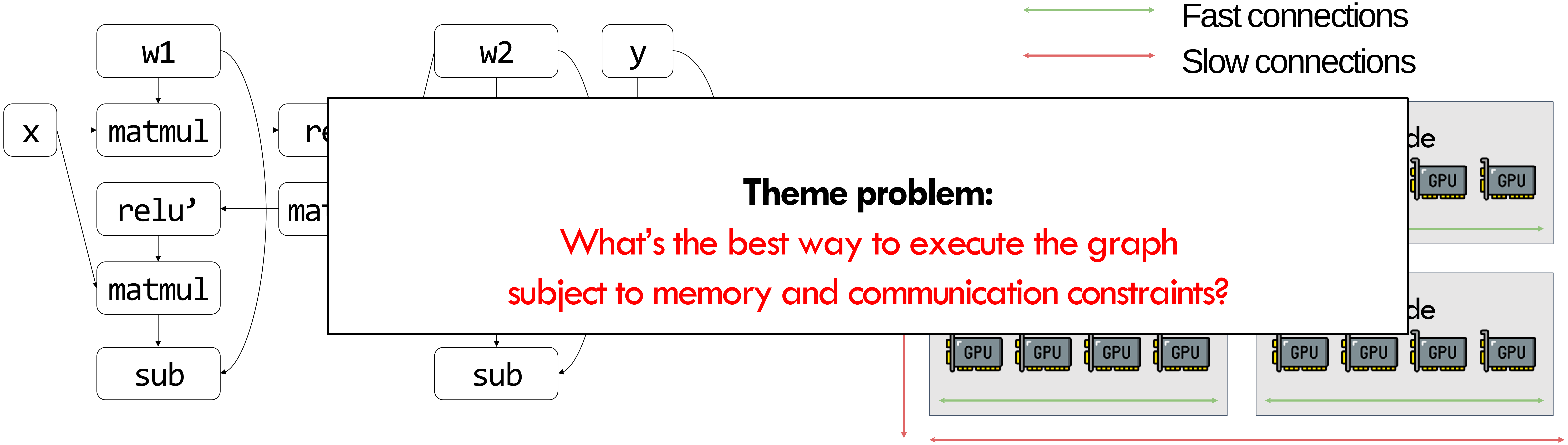
Data and model parallelism

- 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.

ML Parallelization under New View



Theme problem:

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