Data Parallelism in LLM Training

Spring 2026

Lecturer: Yuedong (Steven) Xu

Fudan University

ydxu@fudan.edu.cn

Disclaimer

Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blogposts, research talks, tutorial videos, and other materials shared by the research community. We sincerely appreciate their efforts and assistance, and try our best to cite the sources of the materials used in this course.

Distributed LLM Training: Outline

- **Transformer: A Quick Overview**
- Data Parallelism
  - Parameter-Server
  - All-Reduce
  - Memory Optimization

Transformer Overview

- Transformer everywhere
  - Basic object in Transformer library: ***pipeline()*** which connects a model with its necessary preprocessing and postprocessing steps

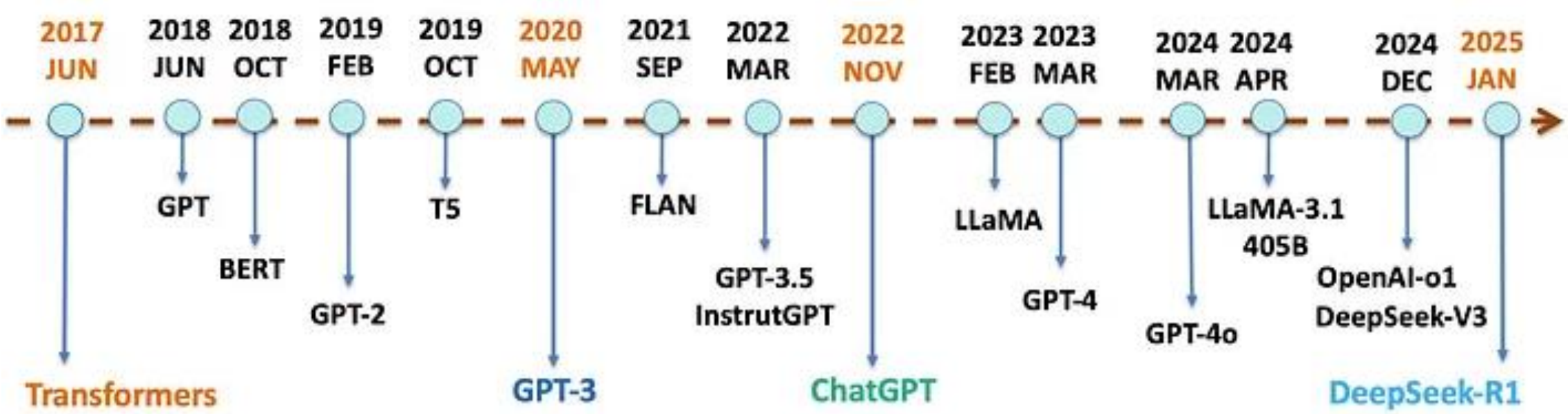
```
from transformers import pipeline
classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a machine learning system course my whole life.")

[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

  - Other pipelines
    - *text-generation, text-classification, summarization, translation, feature-extraction*
    - *image-to-text, image-classification, object-detection*
    - *automatic-speech-recognition, text-to-speech, audio-classification*

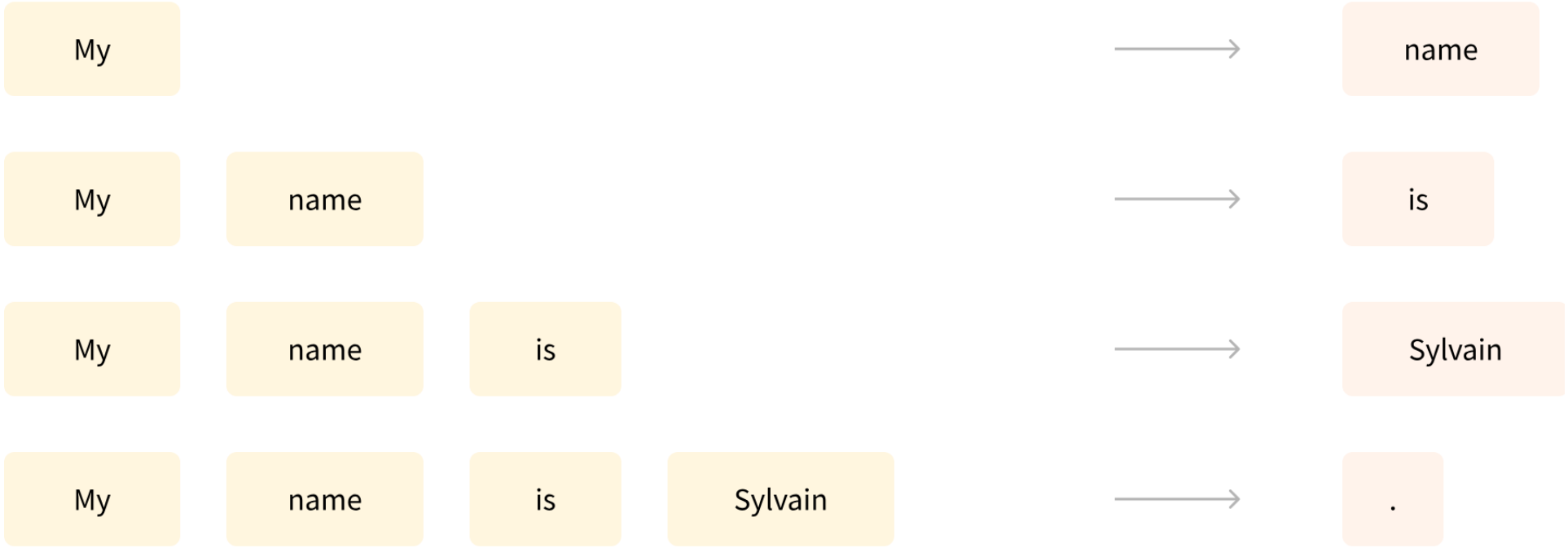
Transformer Overview

- A brief history of Transformer models



Transformer Overview

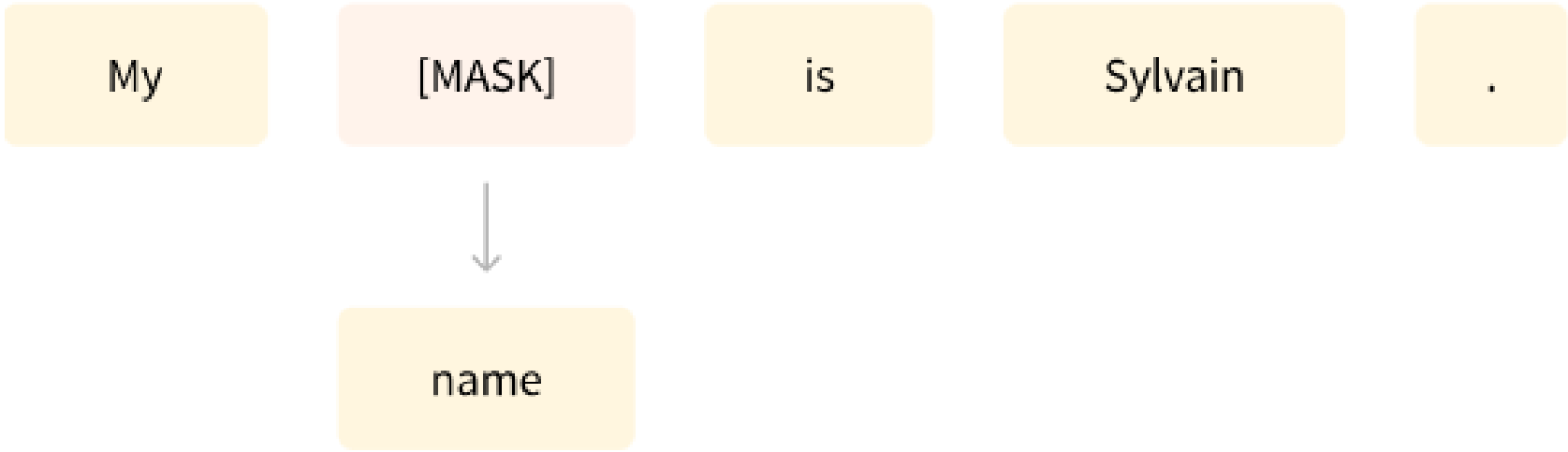
- Causal language model



Predicting the next word in a sentence having read a number of previous words

Transformer Overview

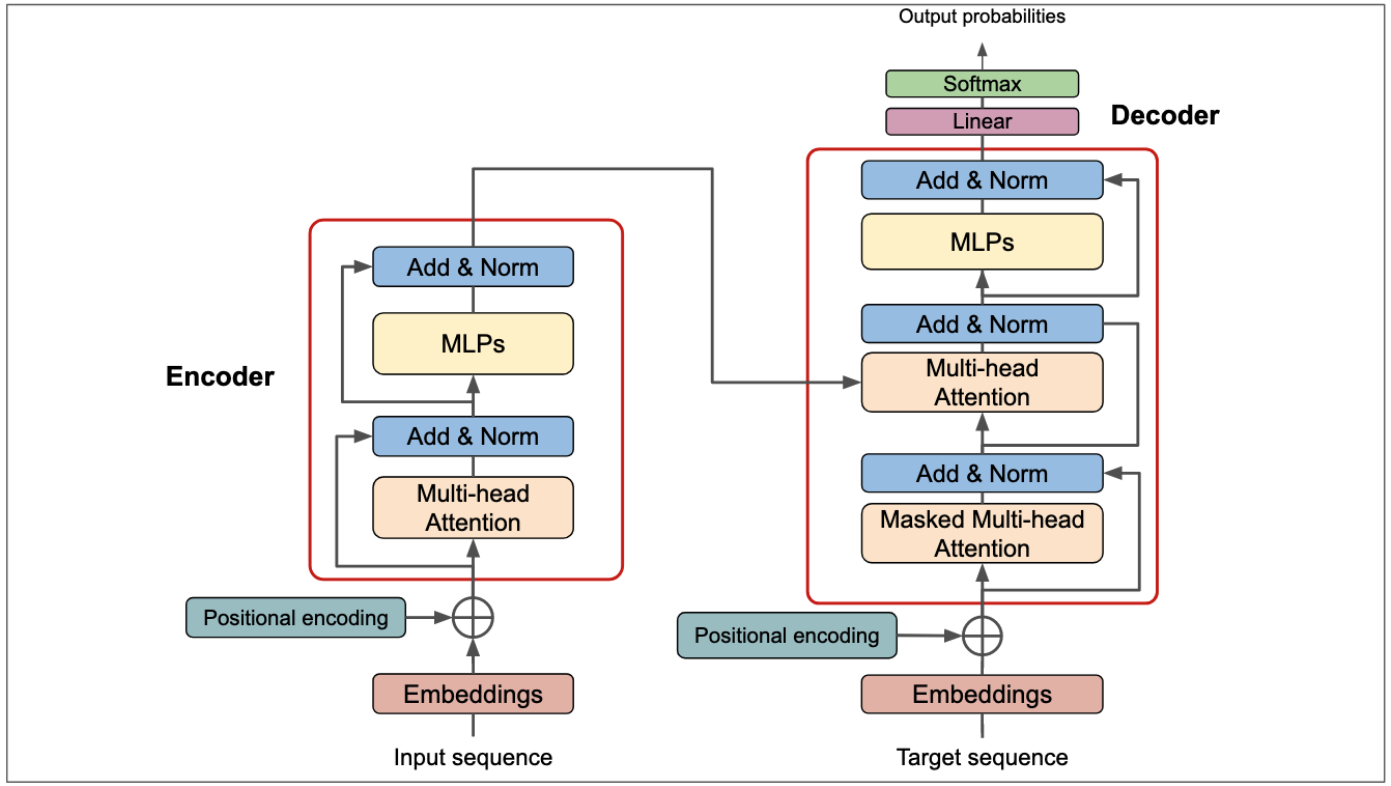
- Non-causal language model



Masked language modeling in which the model predicts a masked word in the sentence

Transformer Overview

- Transformer architecture



A minimalist description of Transformer *

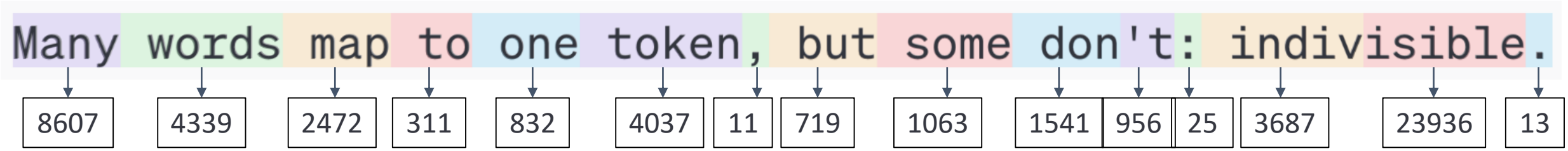
Encoder (left): receiving an input and building a representation of it (its features), i.e. acquire understanding from the input.

Decoder (right): utilizing the encoder's representation along with other inputs to generate a target sequence, i.e. generating outputs.

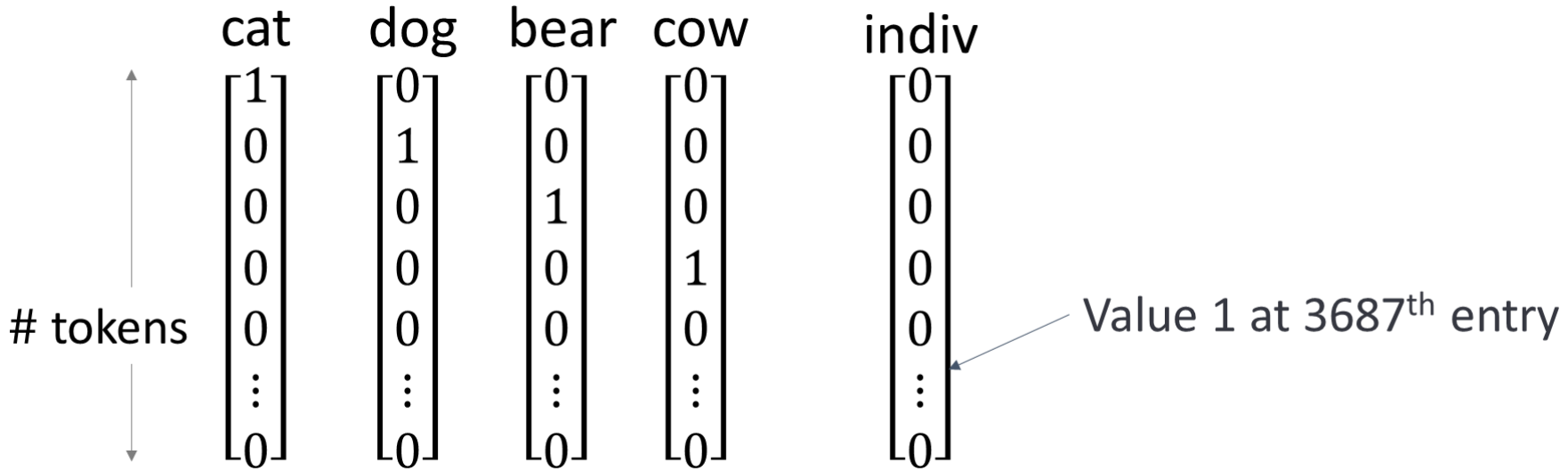
* https://re-cinq.com/blog/llm-architectures

Transformer Overview

- Sequence to tokens

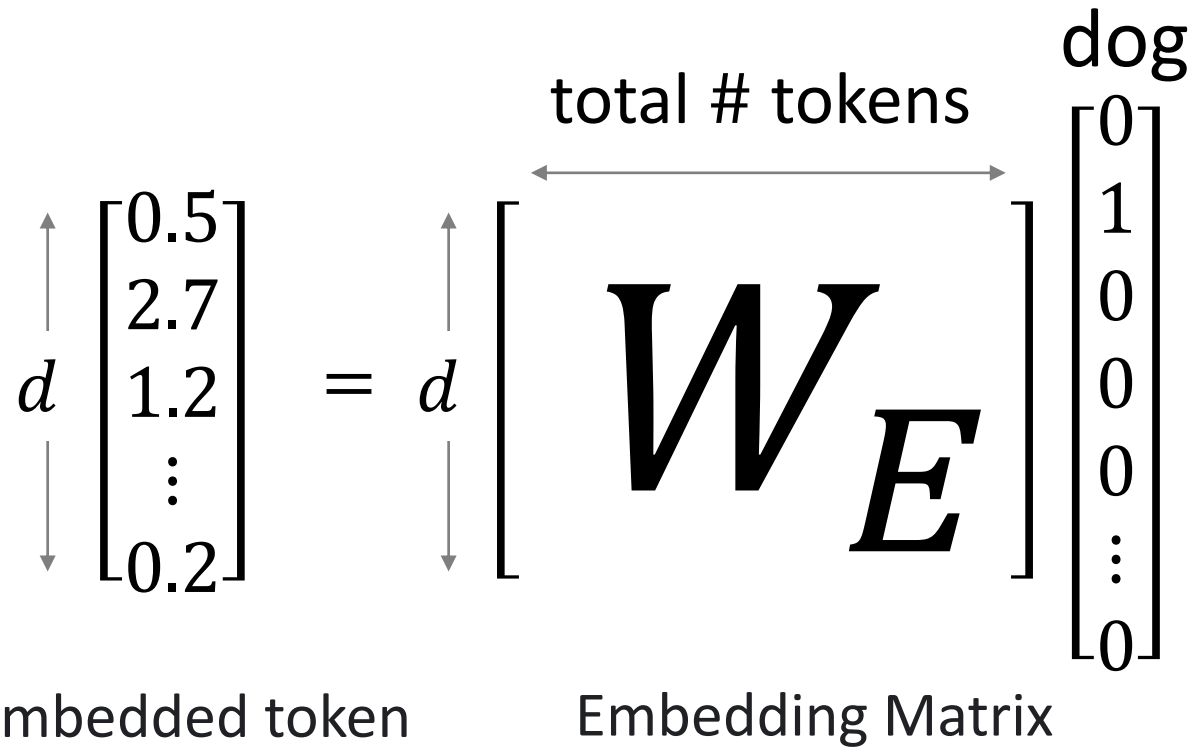


One-hot encoding



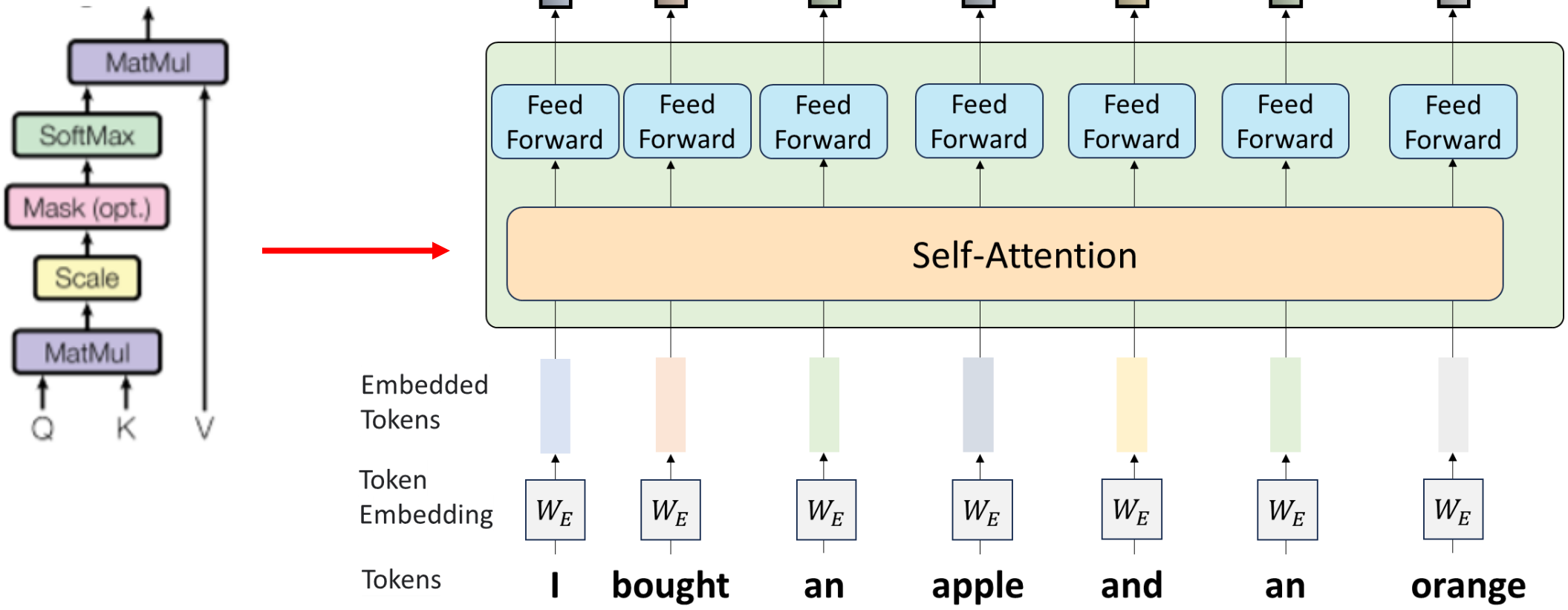
Transformer Overview

- Token embedding
  - Compressing a high dimensional vector into a lower one
  - Multiplying one-hot encoded vector by embedding matrix (index lookup)



Transformer Overview

- Attention computing



Transformer Overview

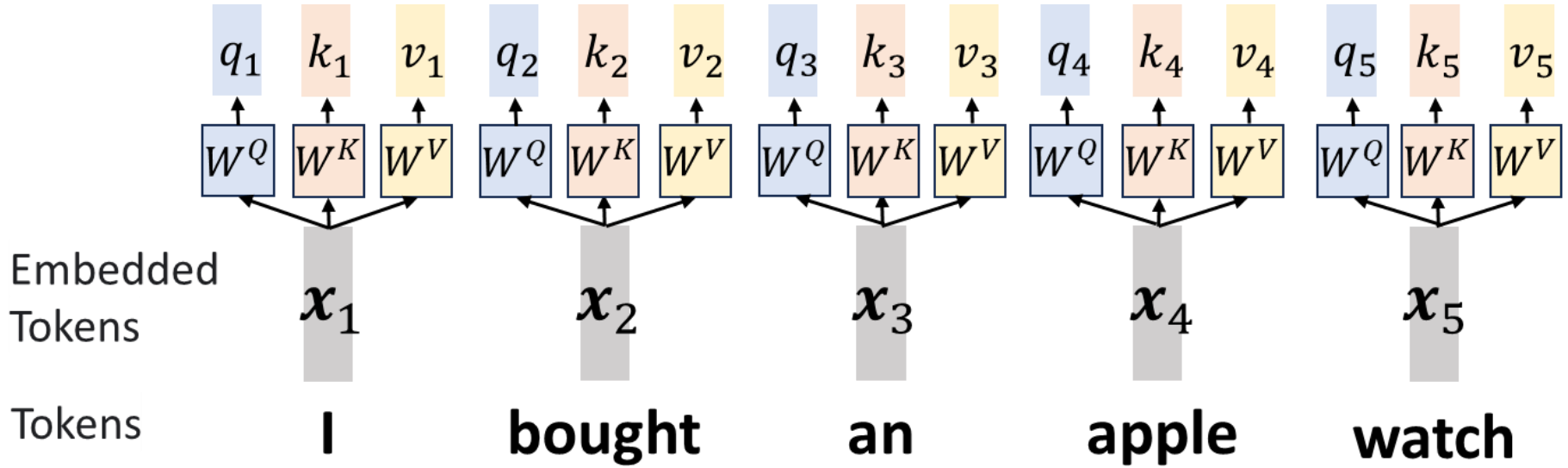
- Attention computing

$W^Q \in R^{d \times d_k}$

$W^K \in R^{d \times d_k}$

$W^V \in R^{d \times d_v}$

$$Q = \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} W^Q \qquad K = \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} W^K \qquad V = \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} W^V$$



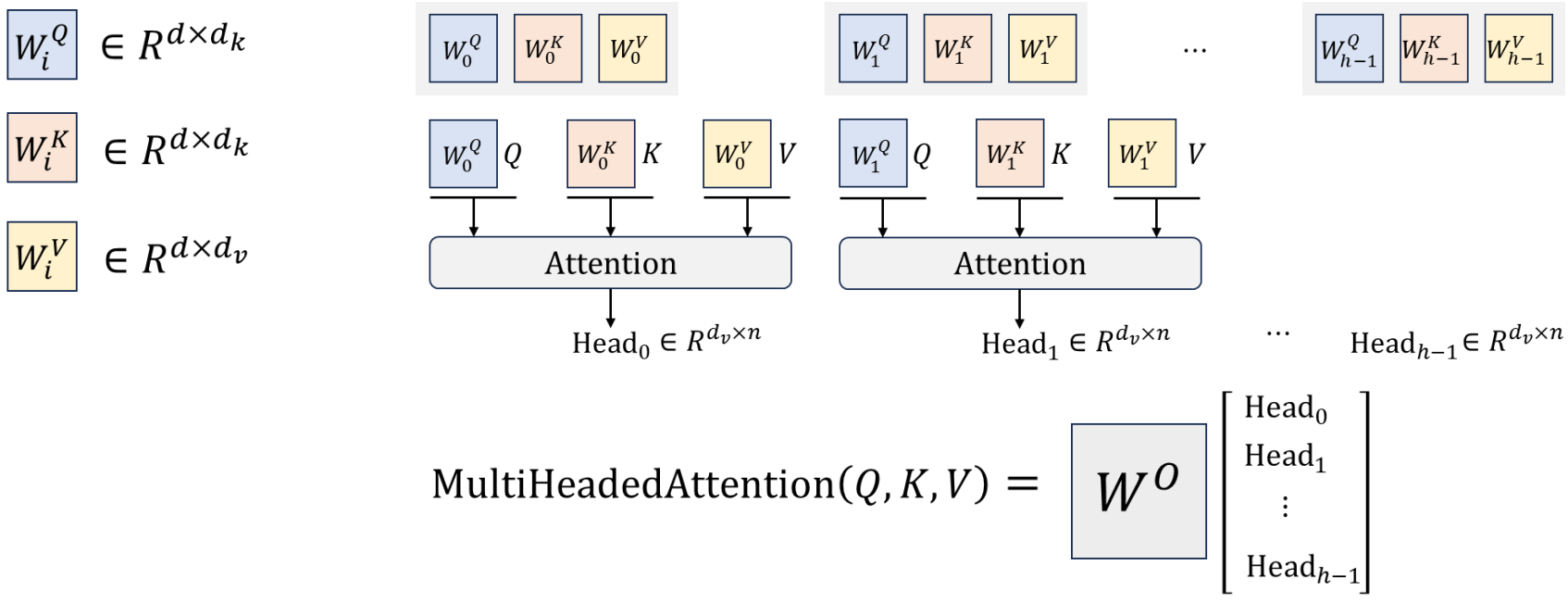
* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

Transformer Overview

- Attention computing
  - Single head ($d_k = d$)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)V$$

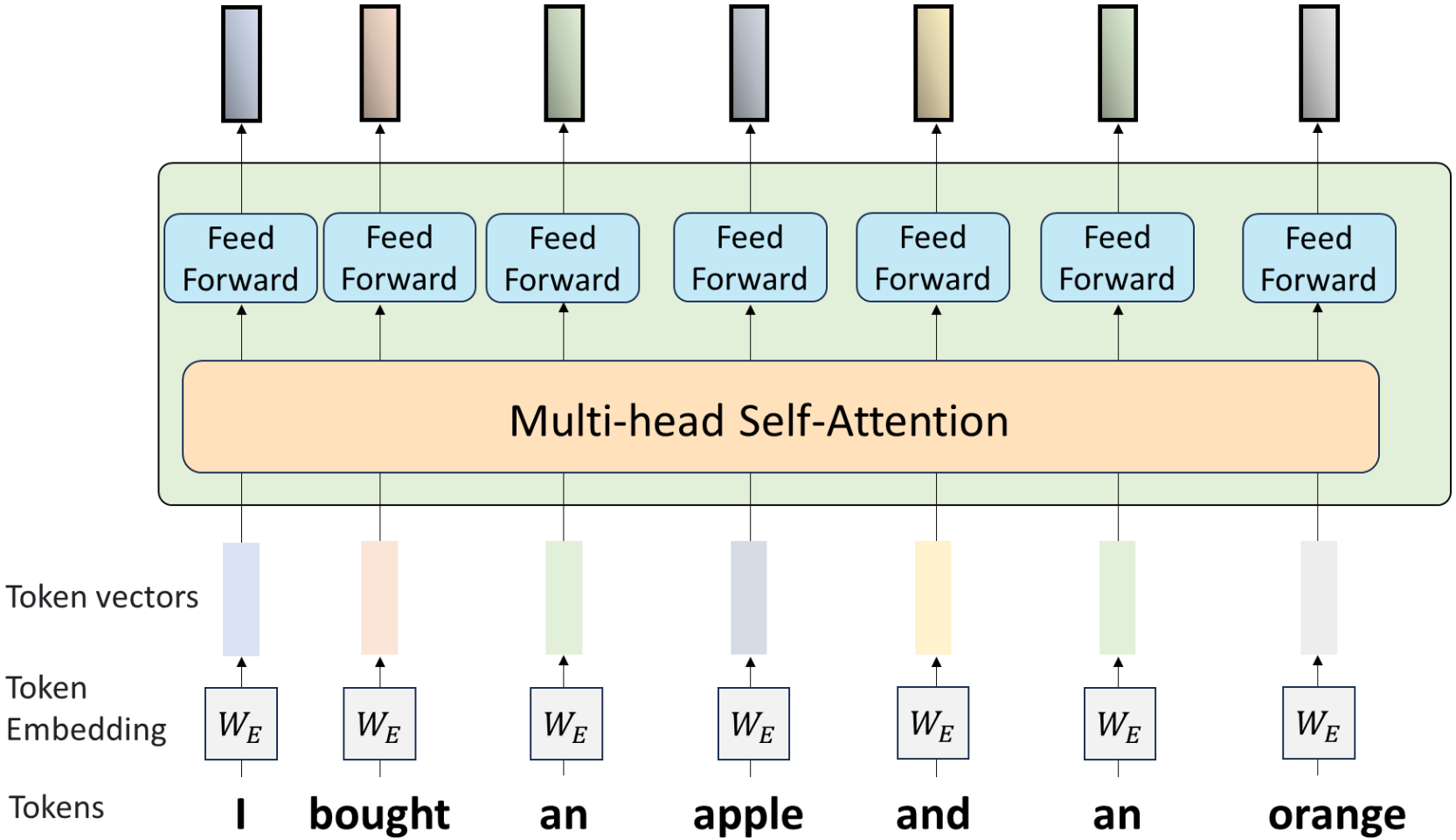
  - Multi-head ($d_k = d/h$)



$$\text{MultiHeadedAttention}(Q, K, V) = W^O \begin{bmatrix} \text{Head}_0 \\ \text{Head}_1 \\ \vdots \\ \text{Head}_{h-1} \end{bmatrix}$$

Transformer Overview

- FFN computing

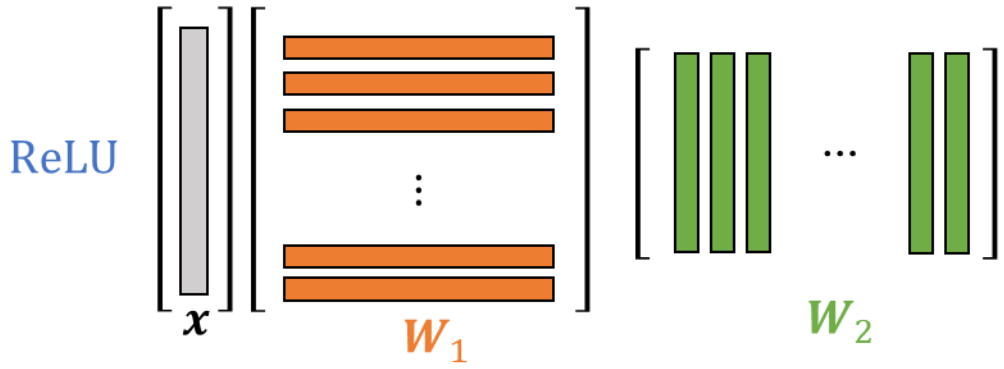


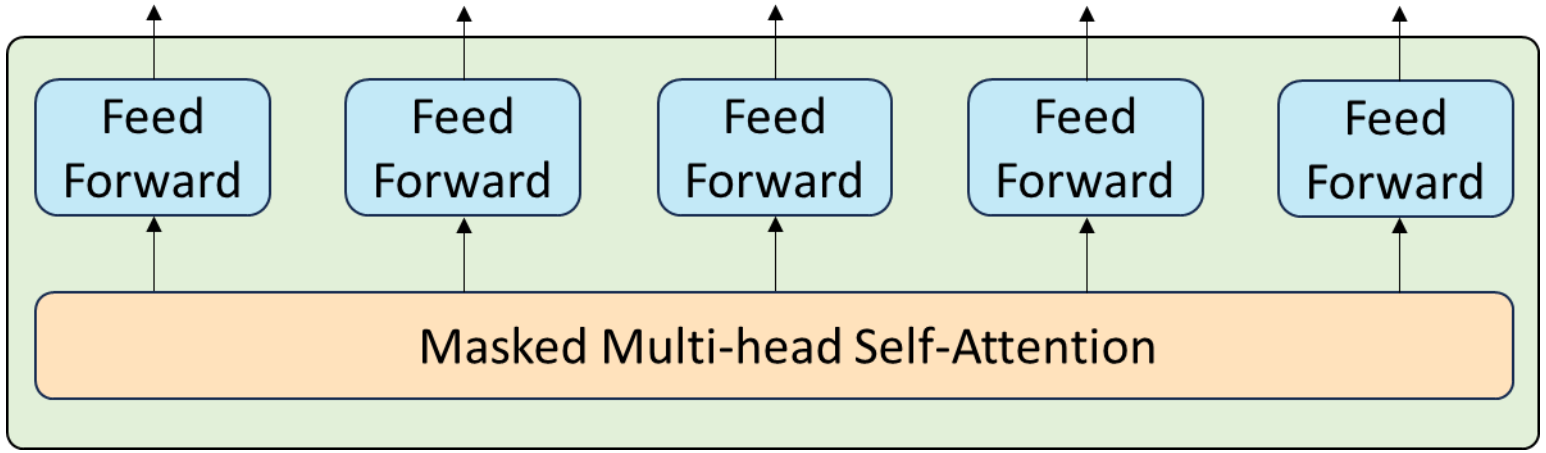
* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

Transformer Overview

- FFN computing

$$FFN(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

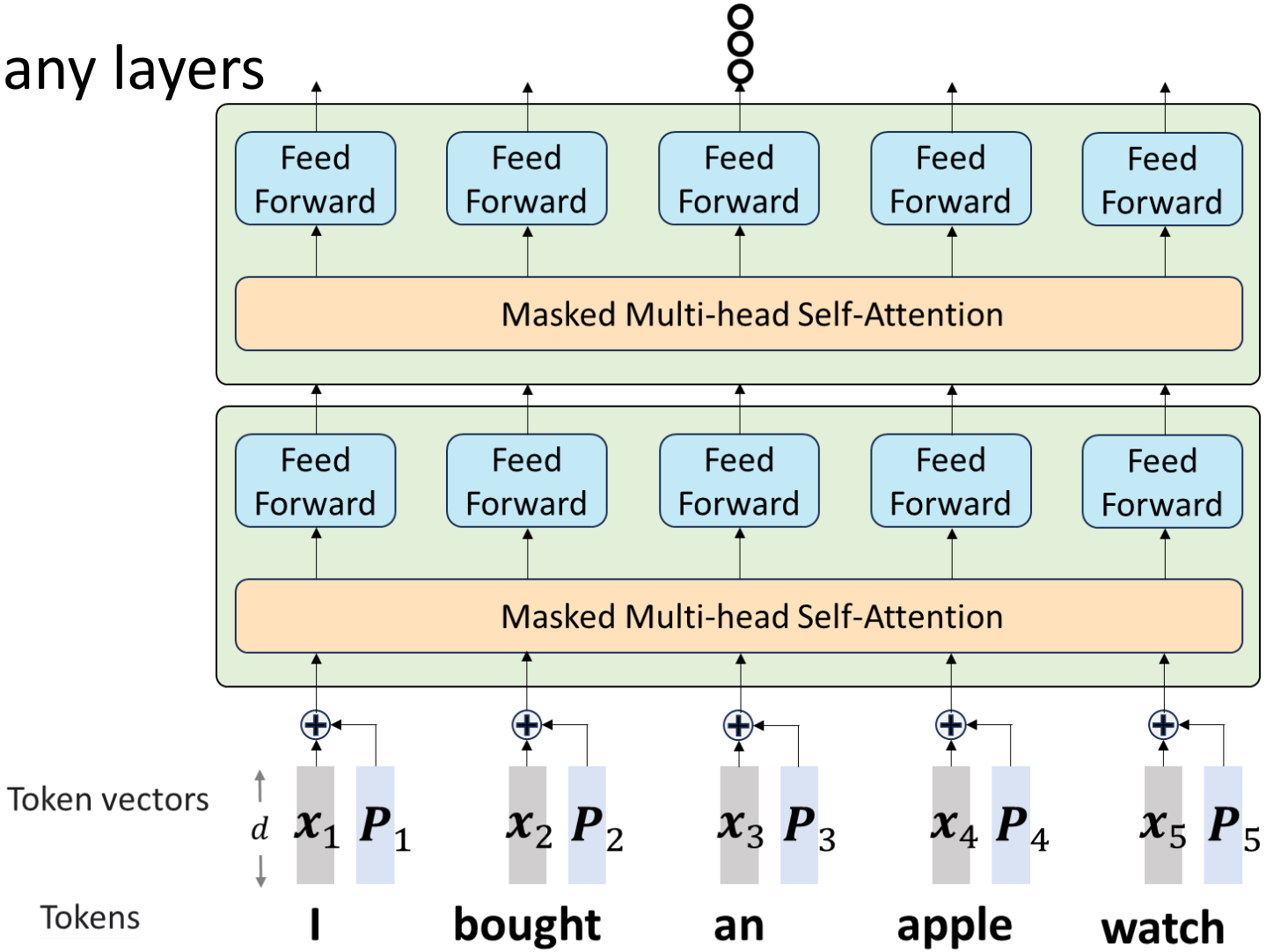




* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

Transformer Overview

- Stacking many layers



* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

Transformer Overview

- Encoder versus decoder

Encoder:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Decoder:

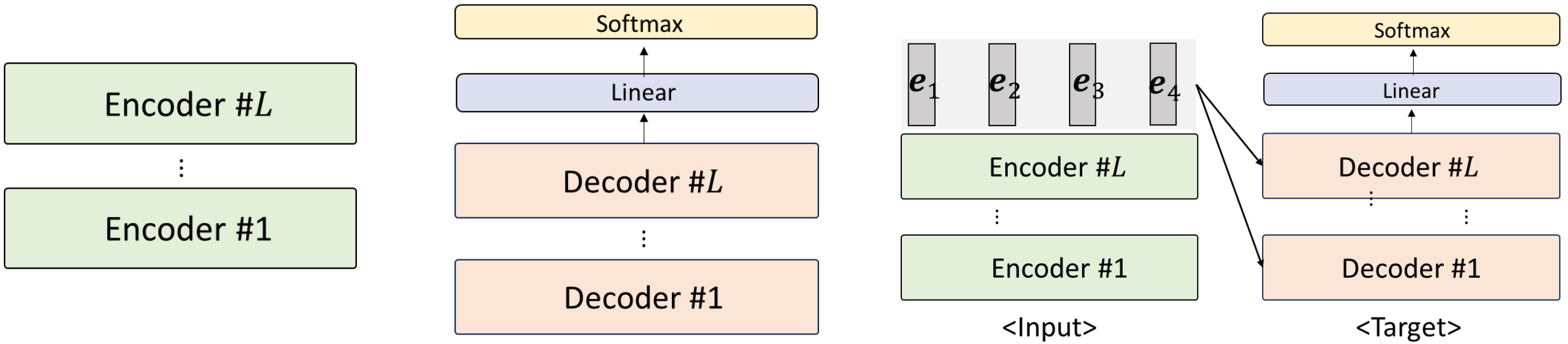
$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

Transformer Overview

- Encoder-only, decoder-only and encoder-decoder



Encoder-only

Decoder-only

Encoder-decoder

* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

Transformer Overview

- Output
  - Linear projection

$$\text{Logits} = x \cdot W^T + b$$

  - Softmax to probability distribution

$$P(w_i) = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

  - Decoding/Sampling to predict a word

* LayerNorm and residual omitted; Slides from Prof. Kai-Bin Huang

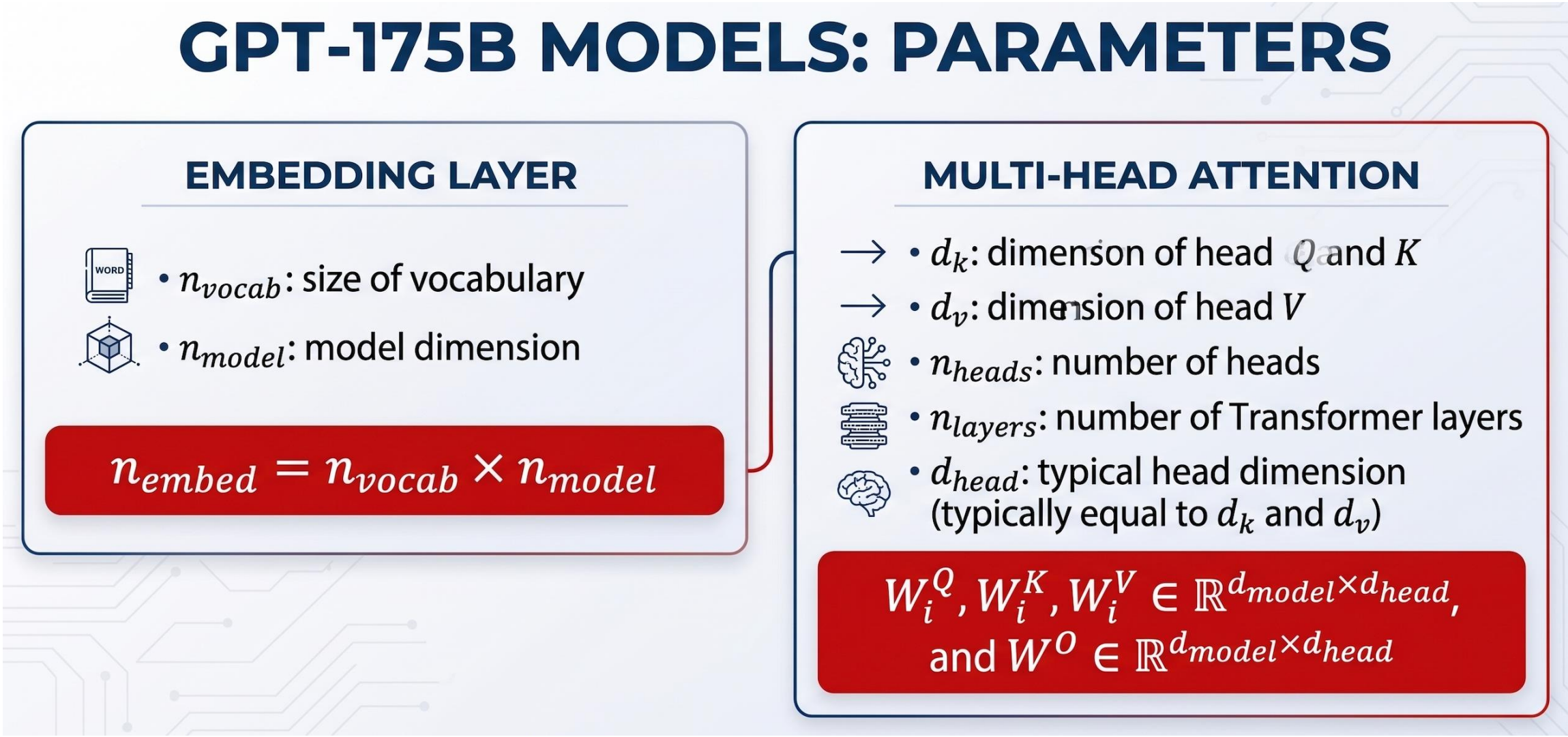
Distributed LLM Training: Outline

- What to memorize
  - Computation flow instead of functionalities of different modules
  - Computational cost and memory consumption

- What to know beforehand
  - Block matrix multiplication
  - Some basic software, hardware and networking knowledge

Why do we need “distributed training”?

Quantitative Analysis

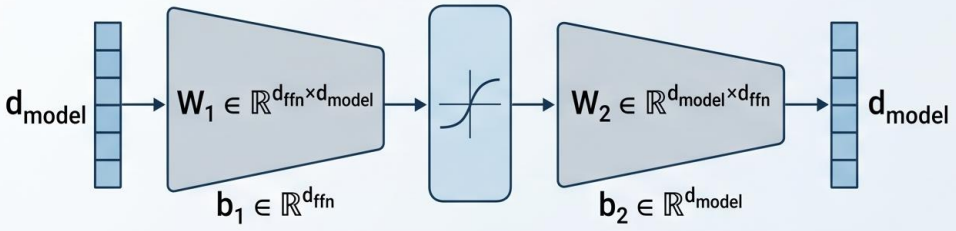


Quantitative Analysis

GPT Models: Parameter Counting

Multi-Layer Perceptron (MLP)

Parameter Breakdown



d_{ffn}: Hidden Dimension of Fully Connected Layer

Parameters: $W_1 \in \mathbb{R}^{d_{ffn} \times d_{model}}$, $b_1 \in$ in $\mathbb{R}^{d_{morel}}$, $b_2 \in \mathbb{R}^{d_{model}}$

$$n_{MLP} = 2d_{ffn} \times d_{model} + d_{ffn} + d_{model}$$

Simplified Total Parameter Count

(Excluding Biases)

$$n_{total} = n_{vocab} d_{model} + n_{layers} (4d^2_{model} + 2d_{ffn} d_{model})$$

*This formula counts key parameter matrices and input token embeddings, excluding all biases and positional embeddings.

Simplified Total Parameter Count

(Excluding Biases)

(Variable d_{model} is uofl hidden dimensiout for consistency)

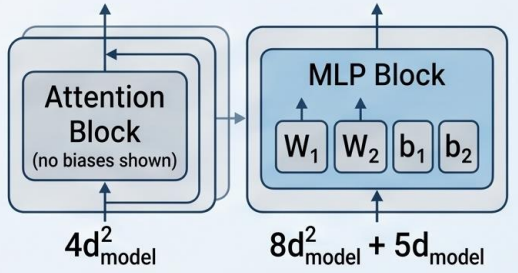
$$n_{total} = n_{vocab} d_{model} + n_{layers} (4d^2_{model} + 2d_{ffn} d_{model})$$

*This formula counts key parameter matrices and input token embeddings, excluding all biases and positional embeddings.

Parameter Estimation for Standard GPT Models

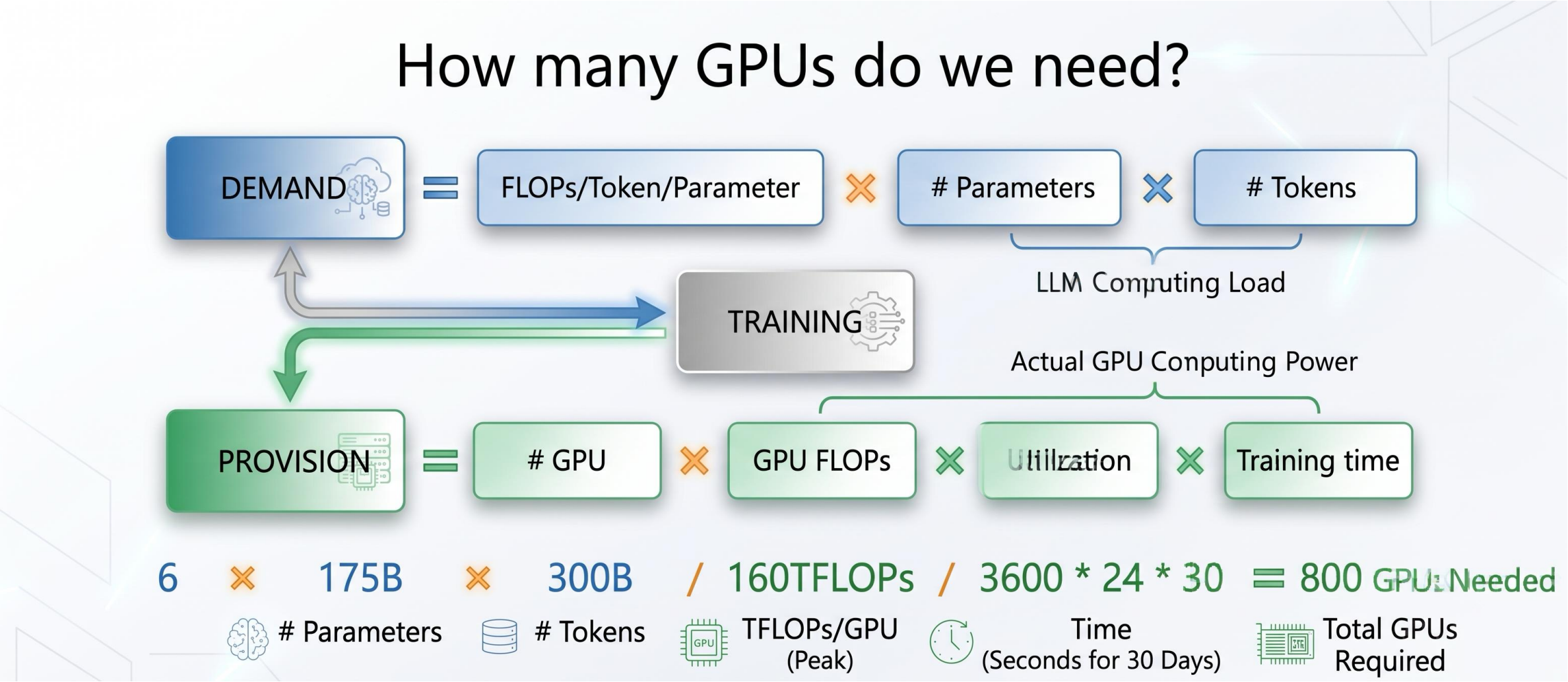
- $d_{model} = d_{head} \times n_{heads}$
- $d_{ffn} = 4\, d_{model}$

Note: Today's largest models often use different ratios.



$$n_{total} = n_{vocab} d_{model} + n_{layers} (12d^2_{model} + 5d_{model})$$

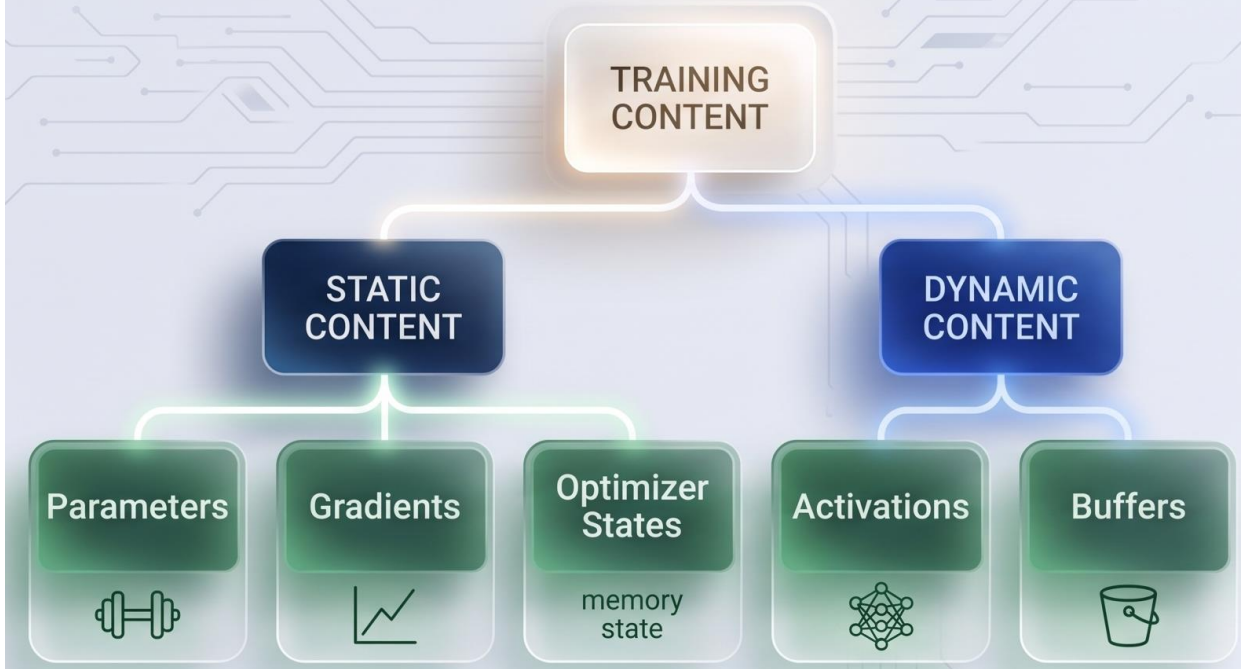
Quantitative Analysis



《Language Models are Few-Shot Learners》
《Efficient large-scale language model training on GPU clusters using megatron-LM》

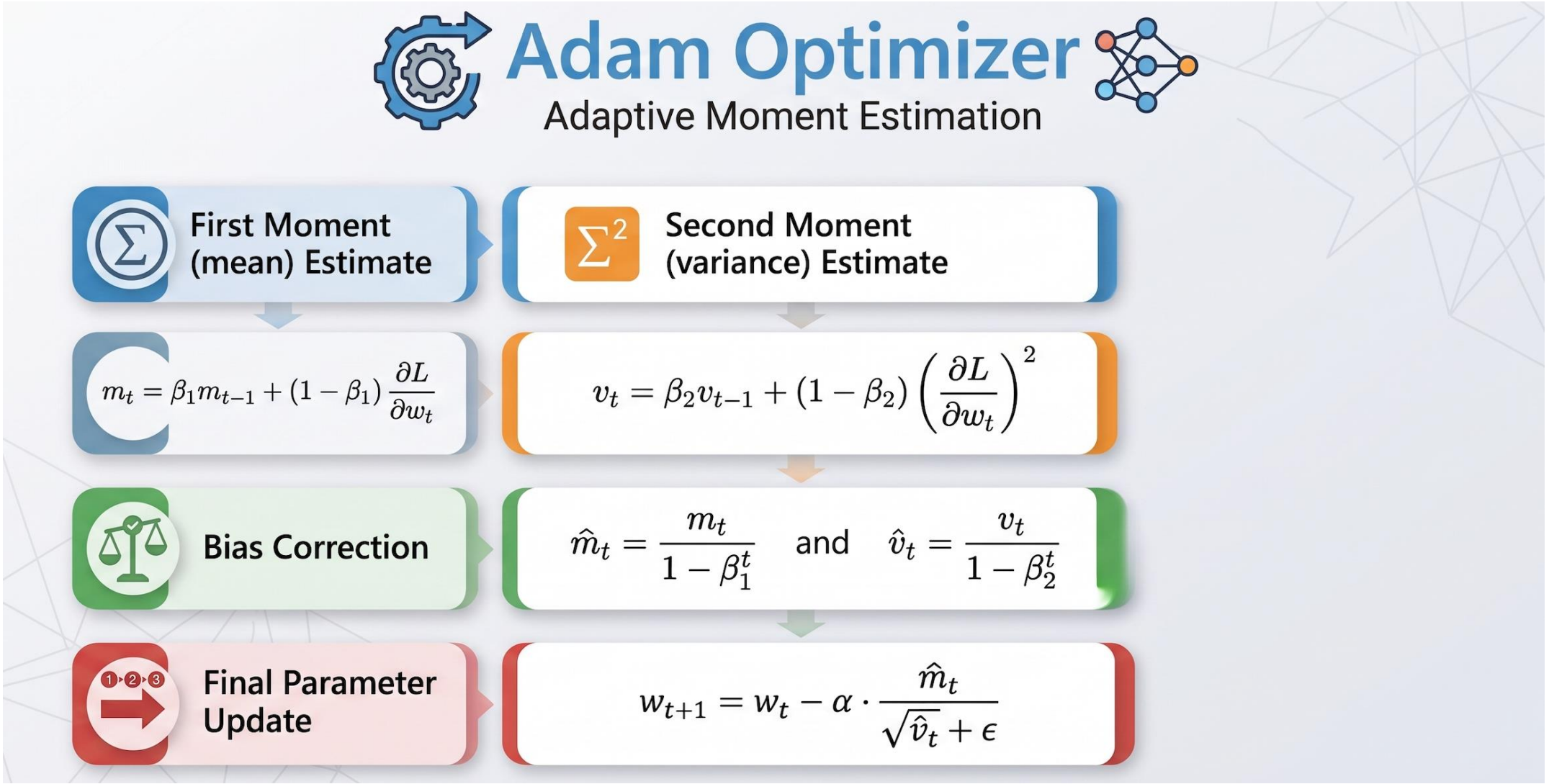
Quantitative Analysis

- GPU HBM Content During Training



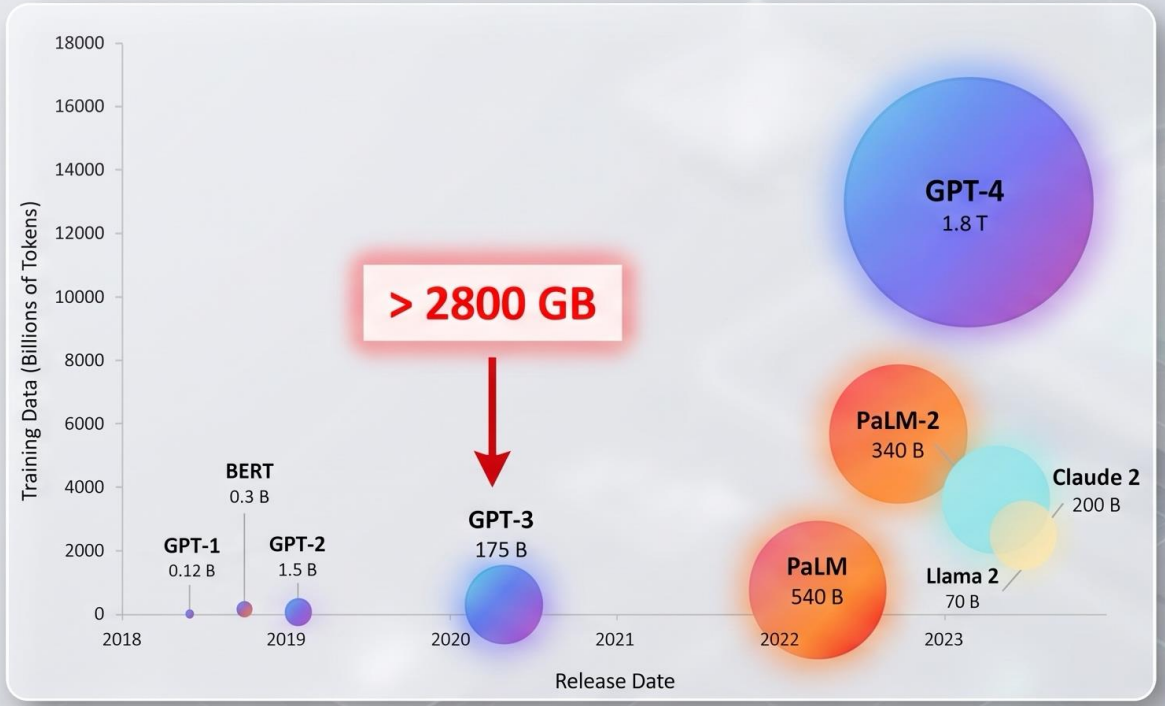
- An example of GPT-3 175B
  - **OPTIMIZER STATES**
    - 32-bit Parameter **700 GB**
    - Adam Moment **700 GB**
    - Adam Variance **700 GB**
  - 16-bit Parameter **350 GB**
  - 16-bit Gradient **350 GB**
  - Activations (depending on batch size)
  - Buffer and Fragmentation

Quantitative Analysis



Quantitative Analysis

- **GPT-175B models: storage**



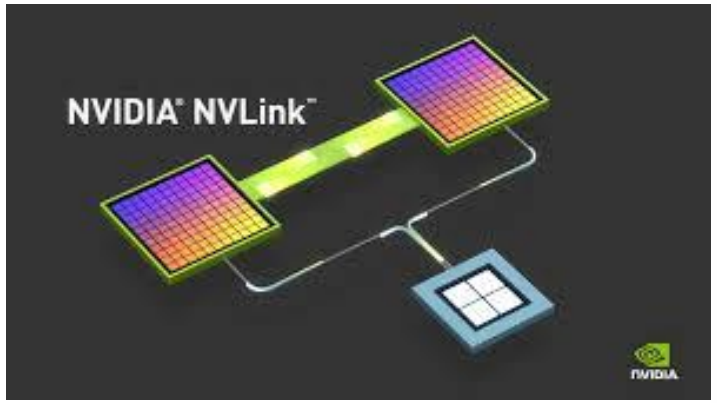
Models become larger and larger

GPU	Memory	Type
Tesla P100	16GB	HBM2
Tesla V100	32GB	HBM2
Tesla A100	40/80GB	HBM2E
Tesla A800	80GB	HBM2E
Tesla H100	80GB	HBM3
Tesla H800	80GB	HBM3

GPU HBM size remains small

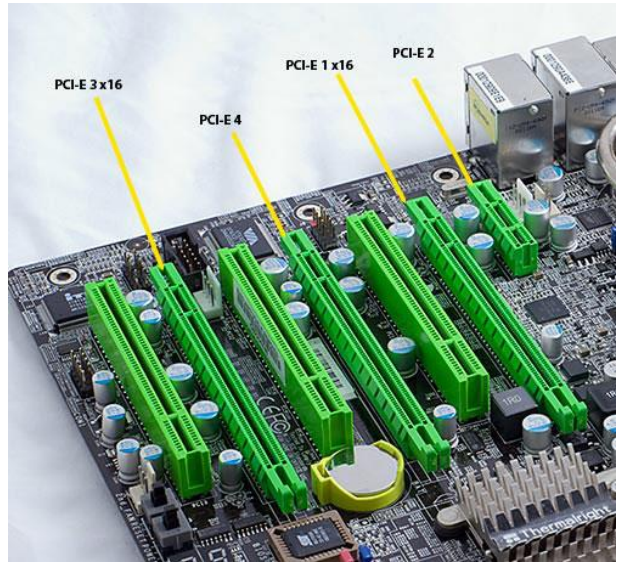
Distributed Training

- Multiple GPUs compute collaboratively
  - Data Parallel, Pipeline Parallel, Tensor Parallel, Expert Parallel, Sequence Parallel, Context Parallel
- GPU interconnection



NVLink 5.0 (e.g. 1.8TB/s)

RDMA (RoCE/InfiniBand, e.g. 400Gbps)



PCIe 5.0 (e.g. 128GB/s, 16lanes)

Distributed Training

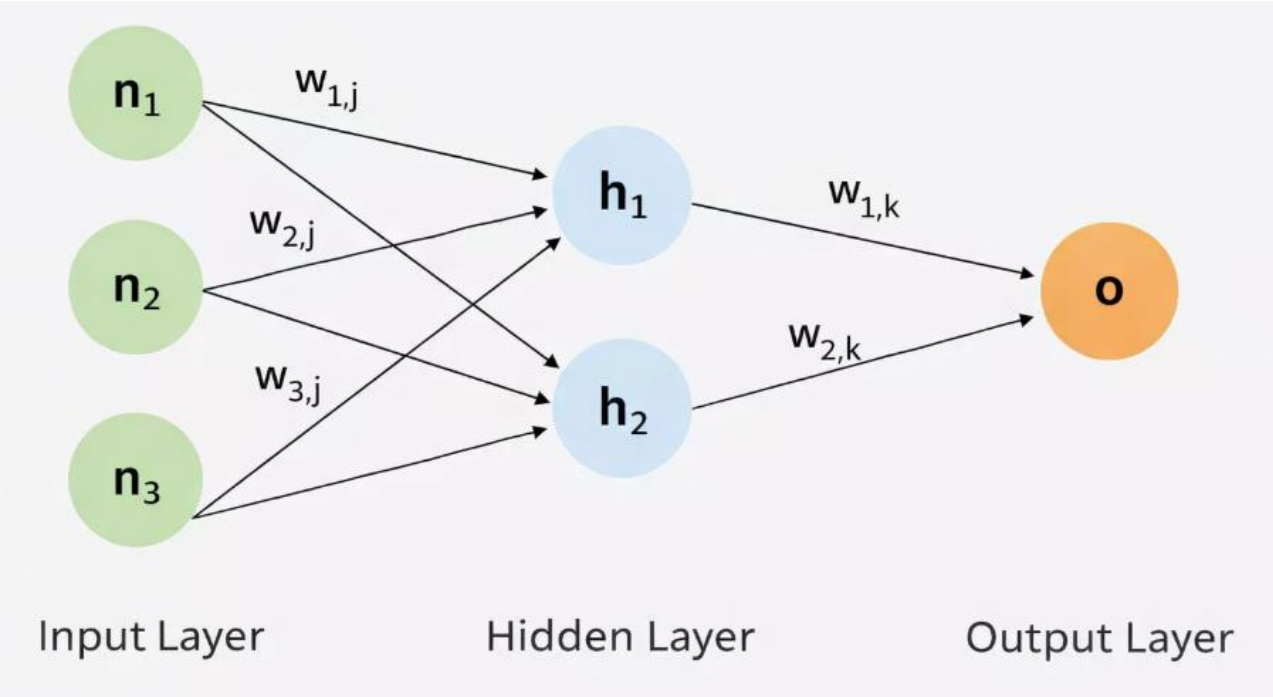
- Importance of Communication
  - Traffic volume
    - 16-bit gradient of GPT3-175B: 350GB needs to be transmitted in every iteration
    - Even more data communication if model is partitioned
  - Imbalanced technology upgrading
    - Ampere A100 (FP16 312TFLOPS, released in 2020) → Blackwell B300 (FP8 72PFLOPS, 2025) → Rubin (2026)
    - NVLink 3.0 (600GB/s, 2020) → NVLink 4.0 (900GB/s, 2022) → NVLink 5.0 (1.8TB/s, 2025) → NVLink 6.0 (3.6TB/s, 2026)

Distributed LLM Training: Outline

- Transformer: A Quick Overview
- Data Parallelism
  - **Parameter-Server**
  - All-Reduce
  - Memory Optimization

DNN Training

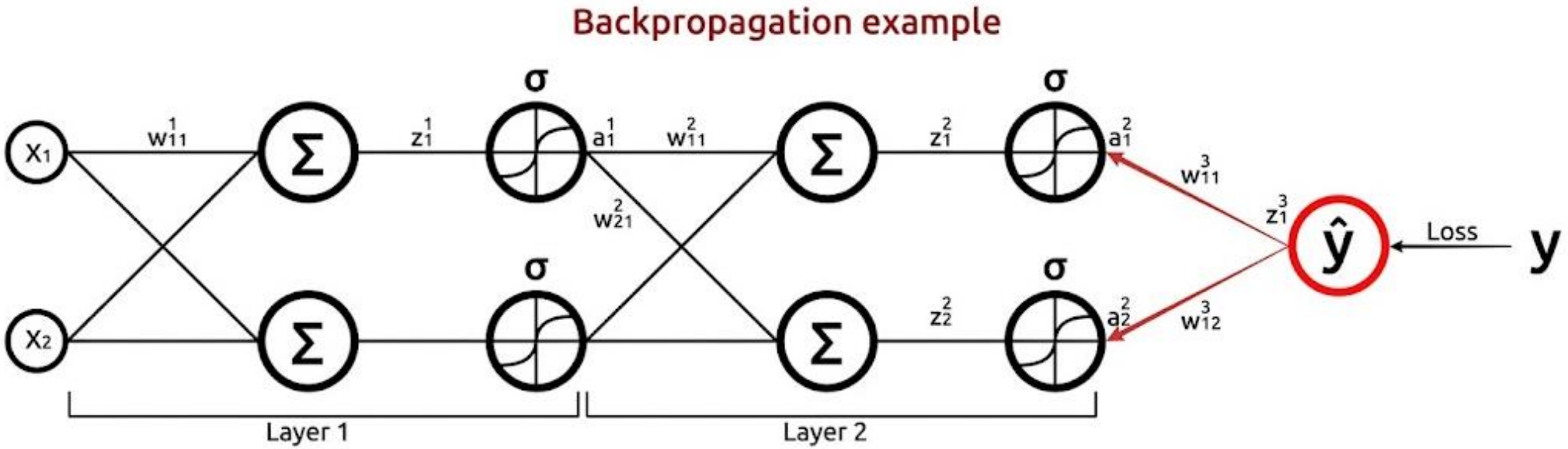
- Forward propagation



Forward propagation

DNN Training

- Backward propagation

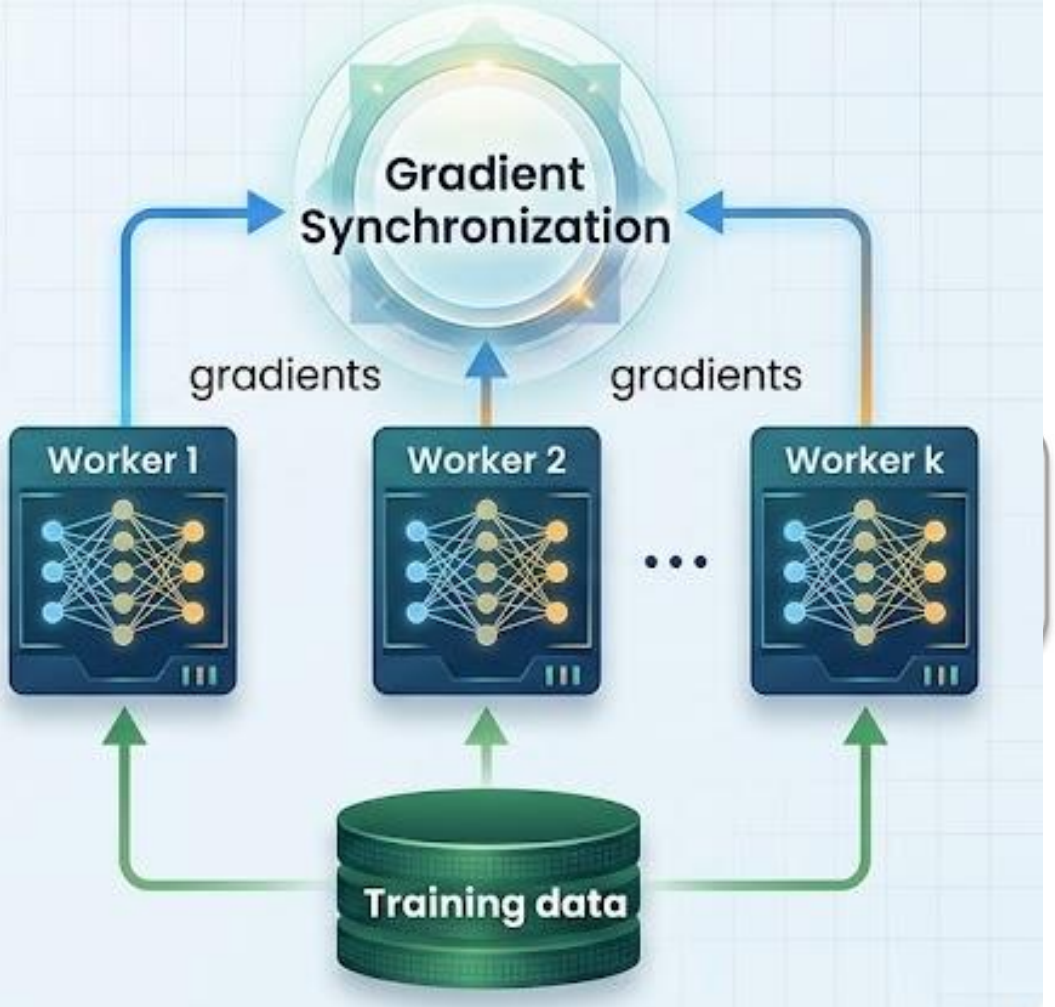


$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial z_1^3}\left[\frac{\partial z_1^3}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial a_1^1} + \frac{\partial z_1^3}{\partial a_2^2}\frac{\partial a_2^2}{\partial z_2^2}\frac{\partial z_2^2}{\partial a_1^1}\right]\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_{11}^1}$$

Distributed LLM Training: Outline

Data Parallelism

- Distribute data to workers
- Each worker work independently
- Synchronize gradients via different approaches
- Repeat the above procedures until model convergence
- Aggregate compute power



Parameter Server Architecture

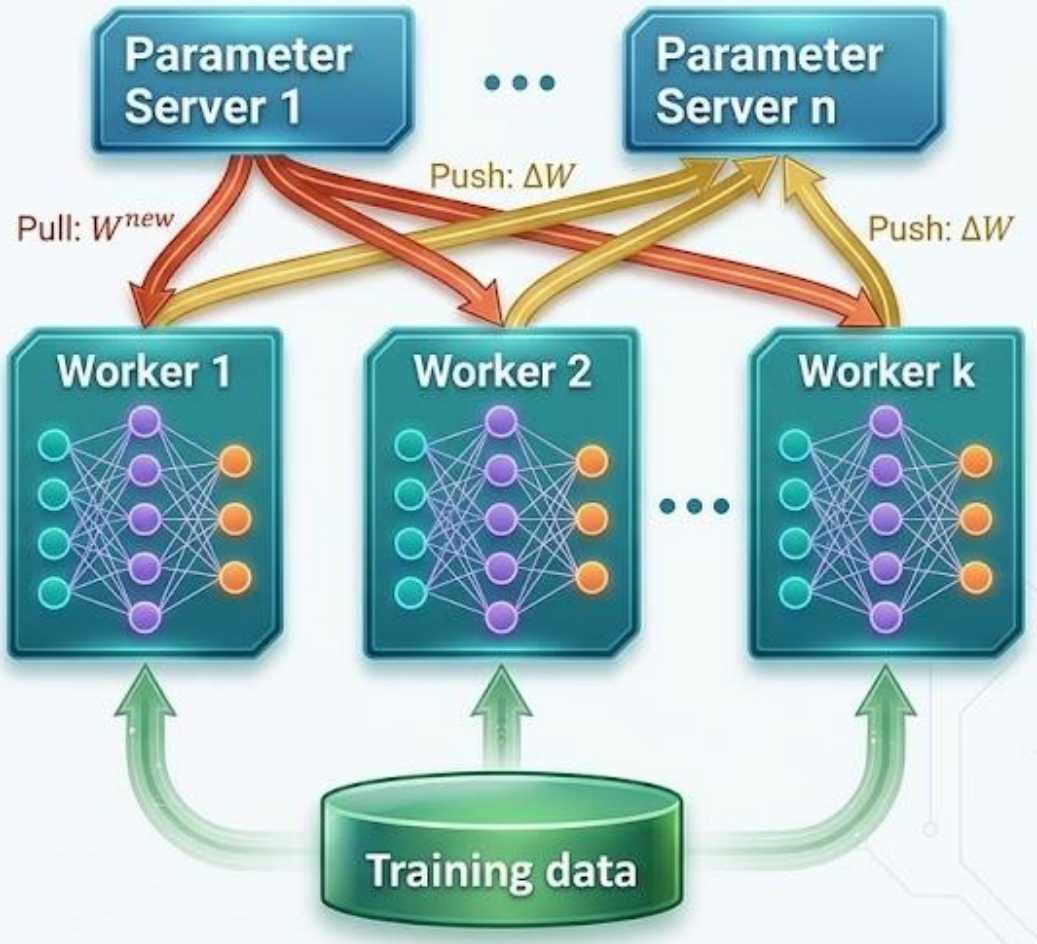
Update: $W^{new} = W - \gamma \Delta W$

5. Pull: W^{new}

3. Push: ΔW

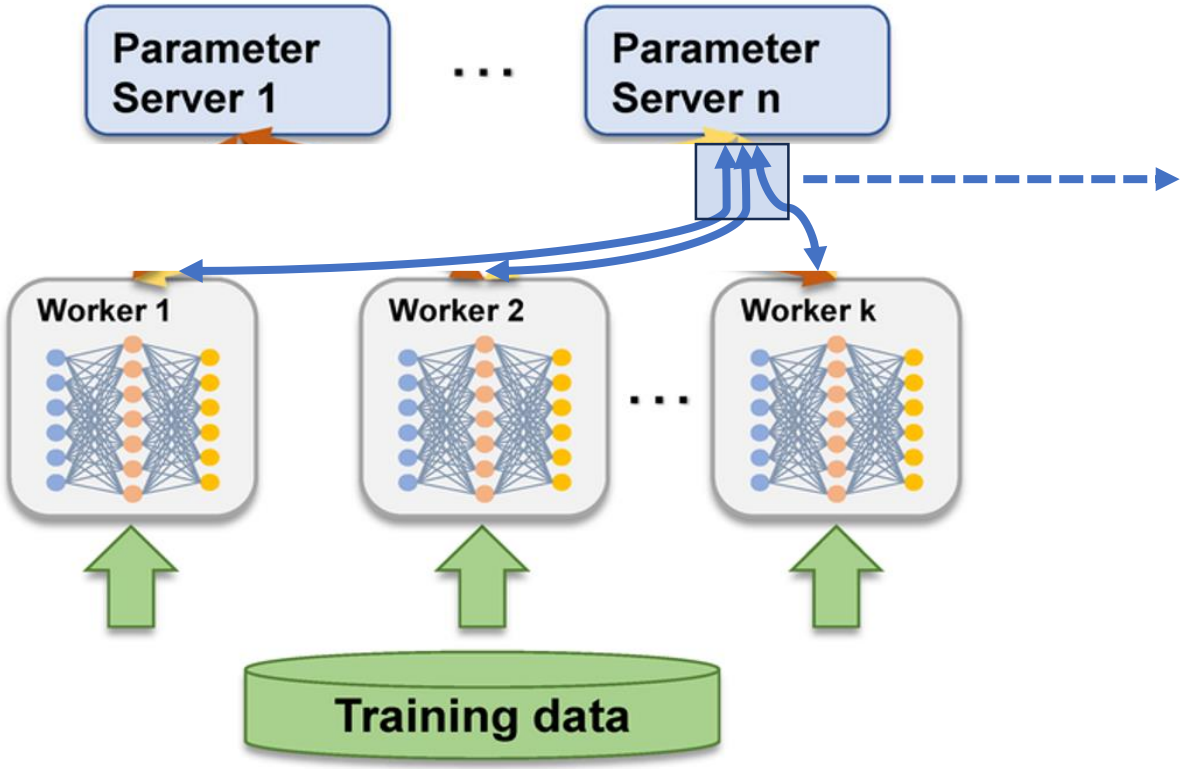
2. back propagation

1. forward propagation



Distributed Parameter Server Architecture: Augmenting Bandwidth

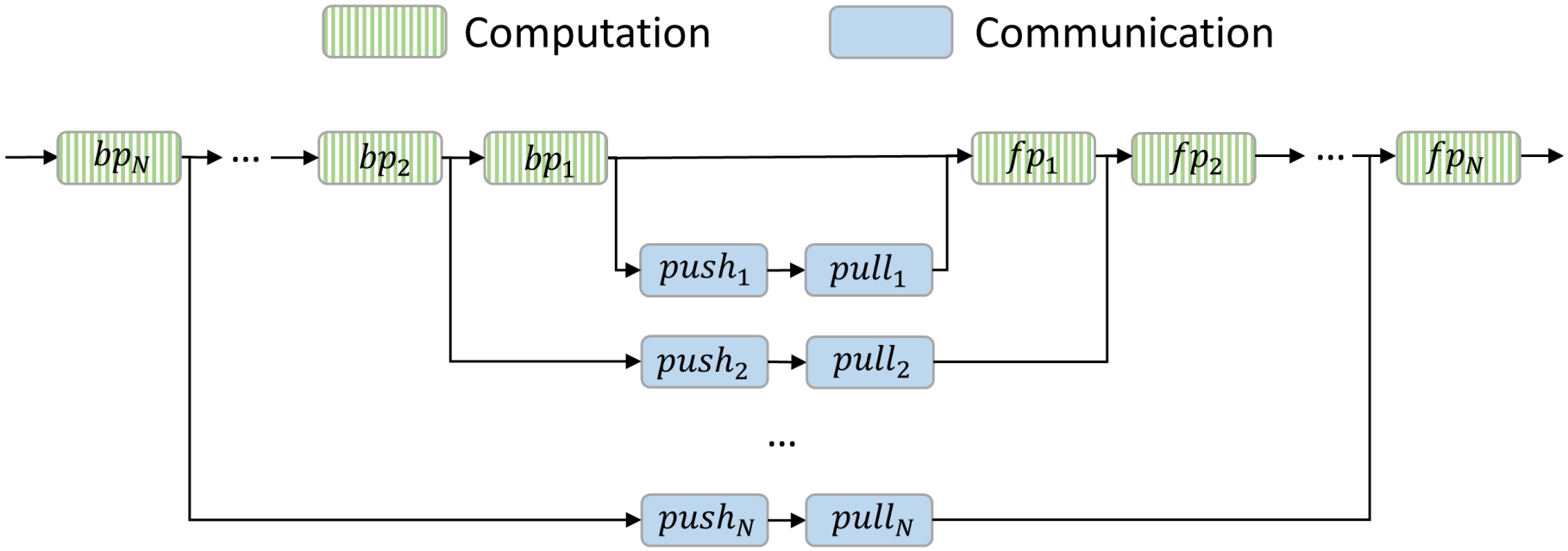
Parameter Server Architecture



- Shared bottleneck
  - Communication throttles computation
  - Reduced bandwidth efficiency due to multi-flow competition
  - Difficult to overlap comp. and comm., push and pull

Network Bottleneck

Optimizing PS Architecture via Scheduling

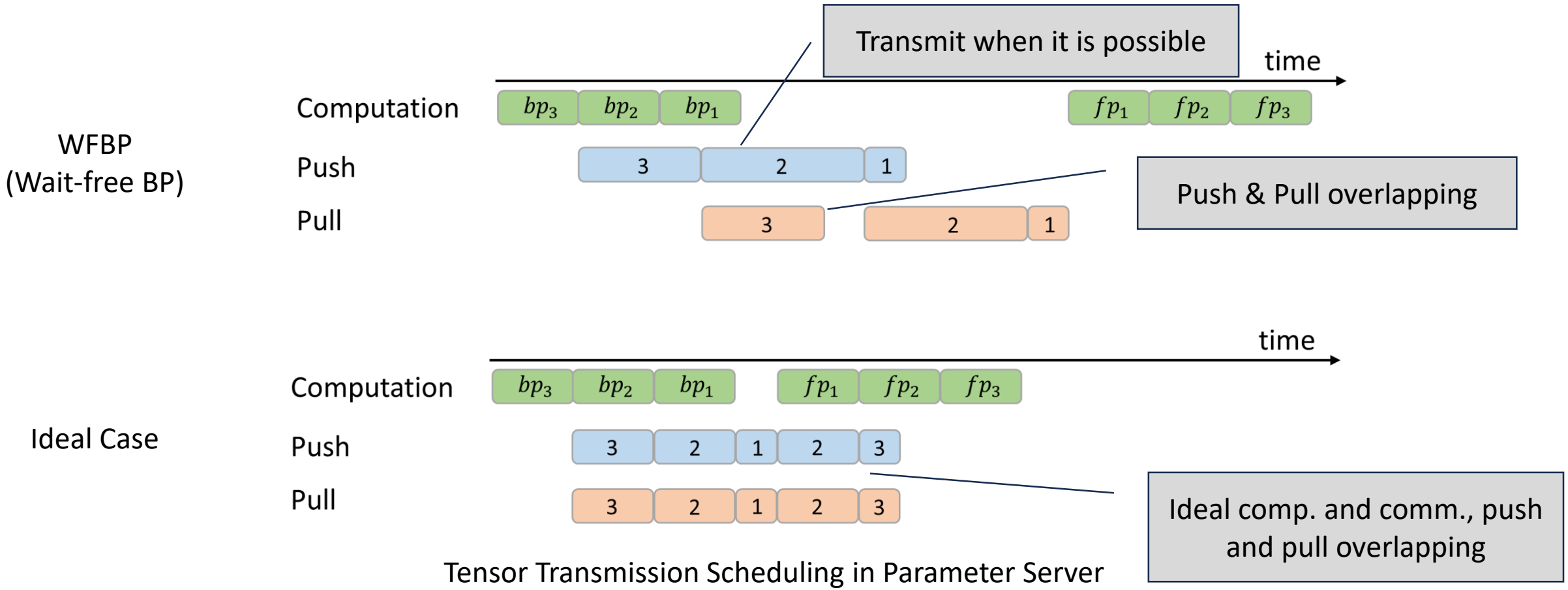


Computation order: $bp_N \rightarrow bp_{N-1} \rightarrow \ldots \rightarrow bp_2 \rightarrow bp_1 \rightarrow fp_1 \rightarrow fp_2 \rightarrow \ldots \rightarrow fp_N$

Data availability order: $gradient_N \rightarrow gradient_{N-1} \rightarrow \ldots \rightarrow gradient_2 \rightarrow gradient_1$

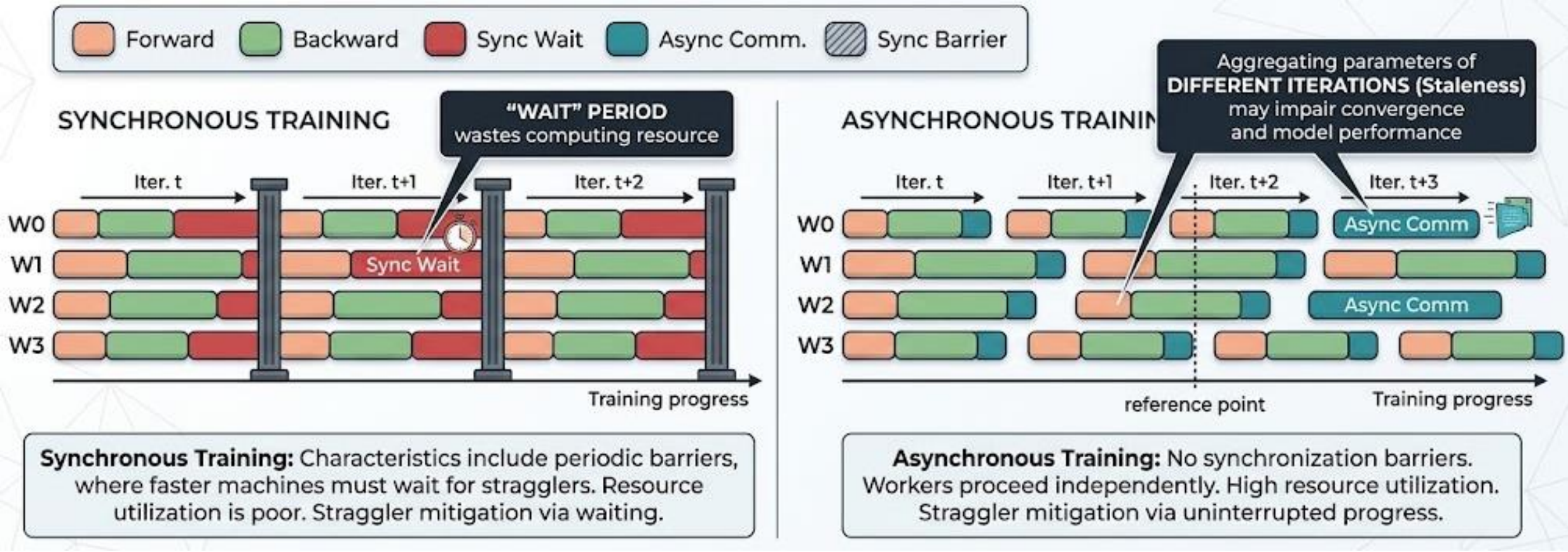
Layer-wise Computation and Communication for a DNN with three layers

Optimizing PS Architecture via Scheduling



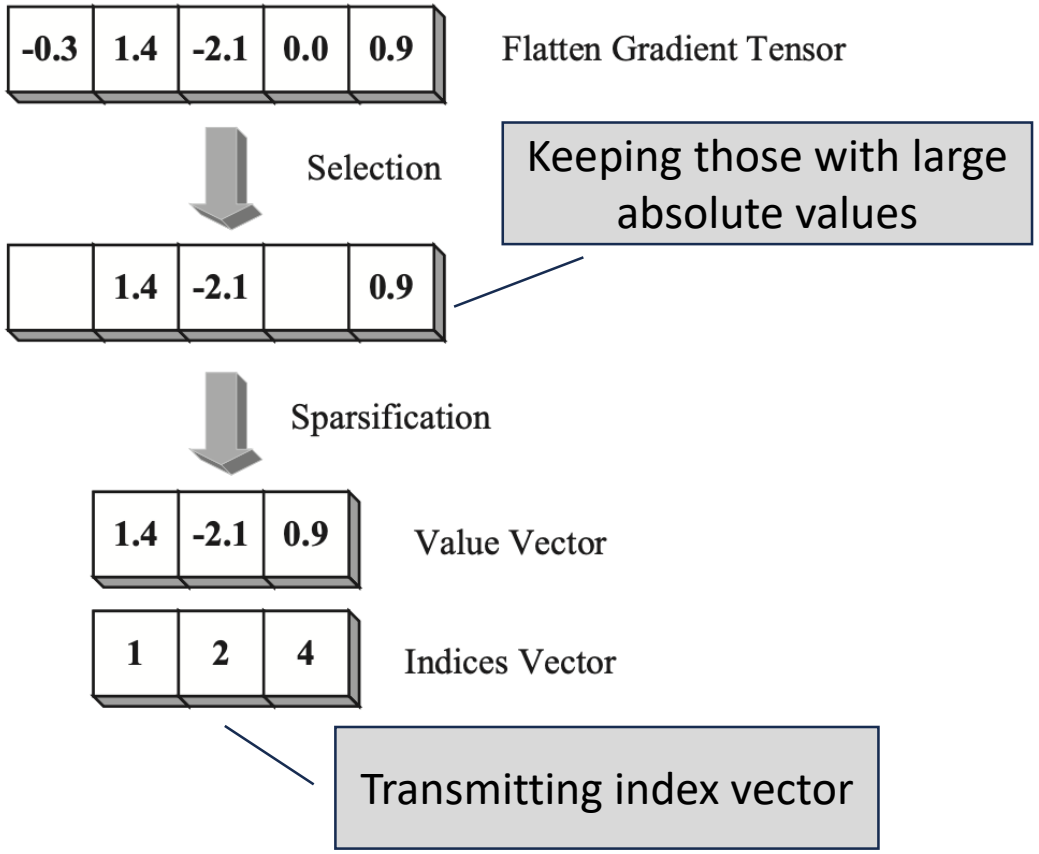
Tensor Transmission Scheduling in Parameter Server

Optimizing PS Architecture via Asynchronism

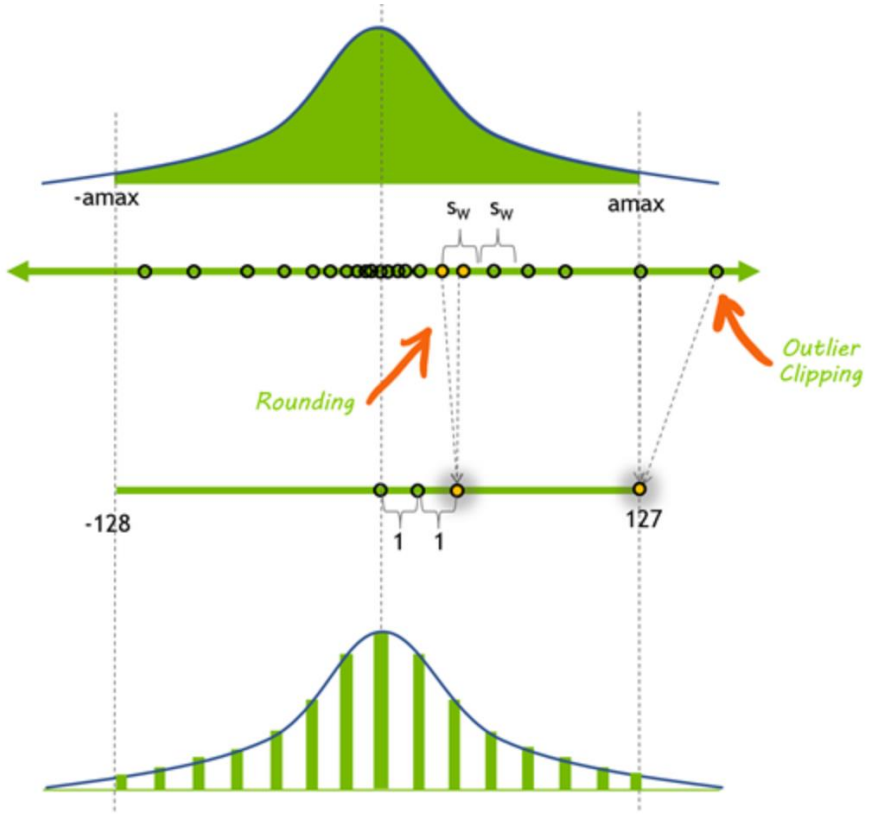


Asynchronous Training: Mitigating Stragglers

Optimizing PS Architecture via Compression



Sparsification: Reducing # of parameters

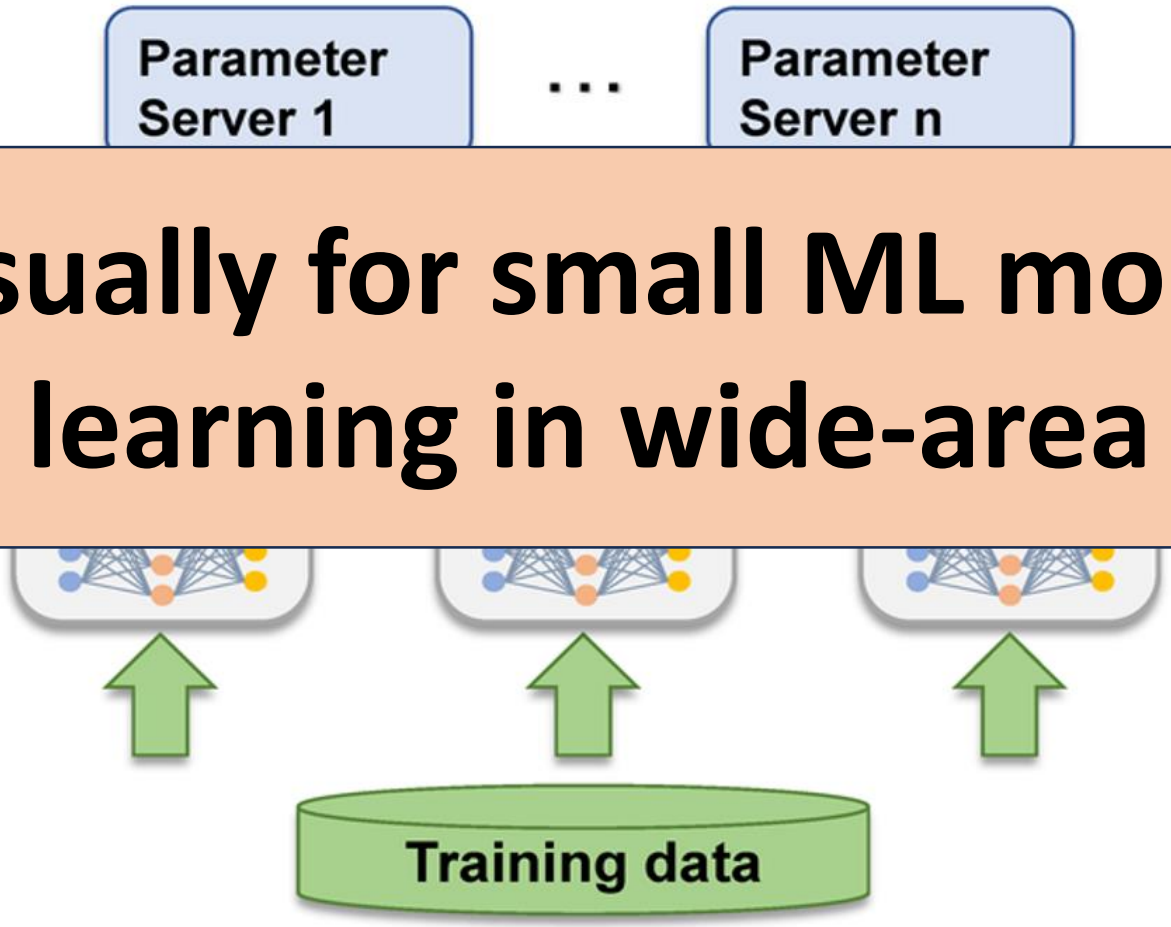


Mapping gradients into low-precision ones w/wo the consideration of value distribution

Sparsification and Quantization

《Communication Compression Techniques in Distributed Deep Learning: A Survey》

Parameter Server Architecture



Simple, usually for small ML models and distributed learning in wide-area networks

Thanks!