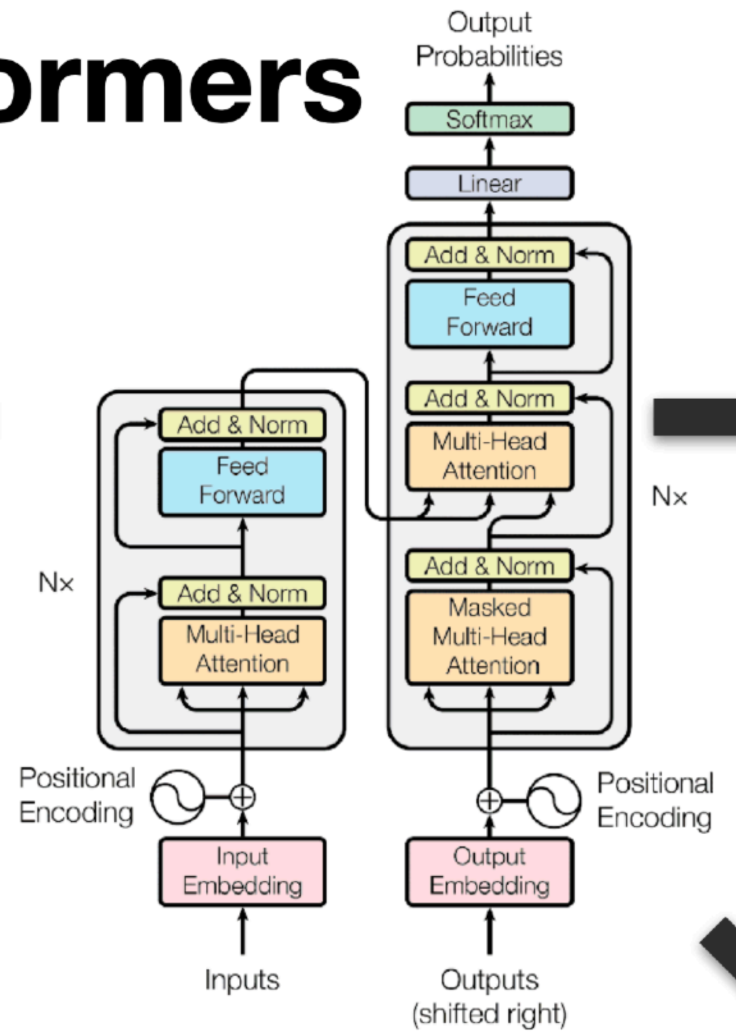
COS 597A

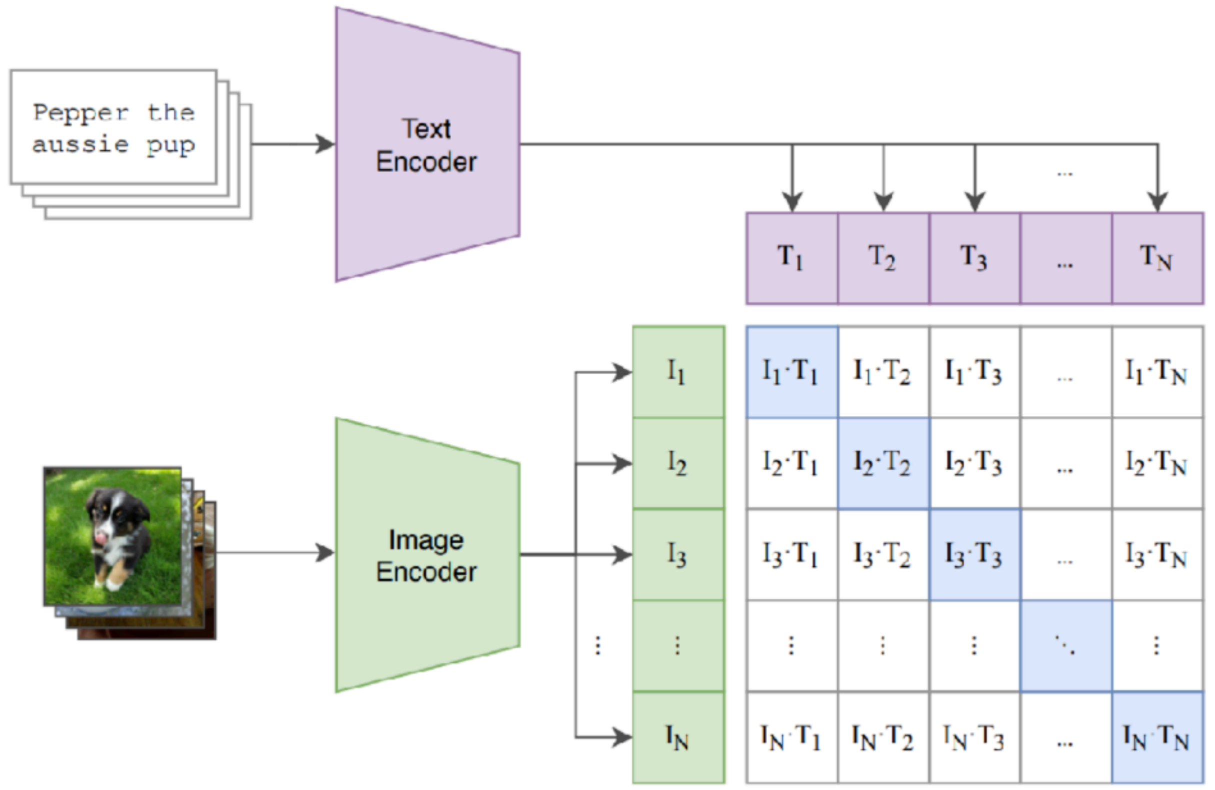
L1: Overview - Systems for LLM Training and Inference

Fall 2025

Transformer models as universal architecture

Transformers





Natural Language Supervision for Vision (CLIP)

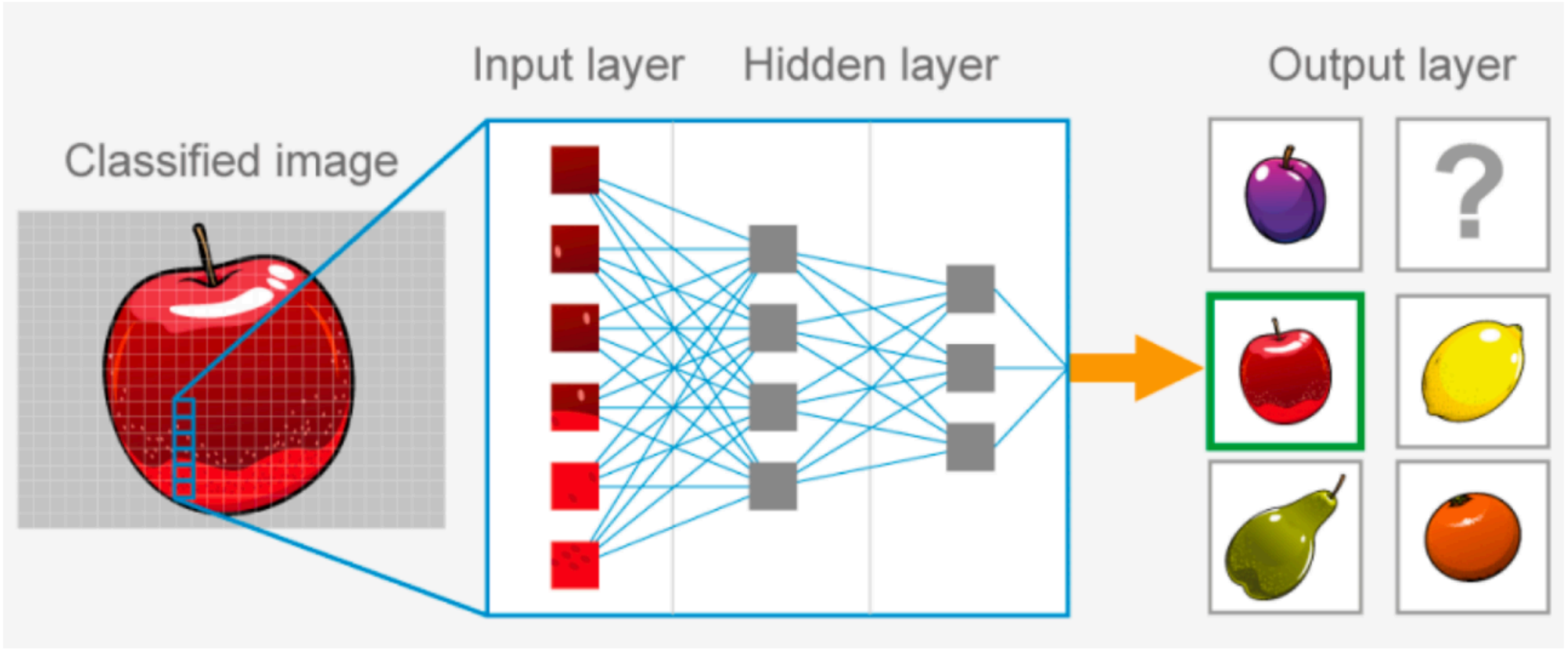
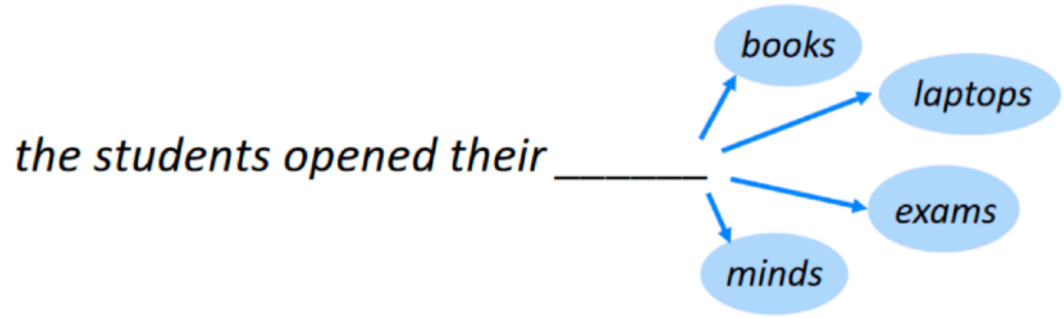


Image Classification (Vision Transformer, Swin-Transformer)



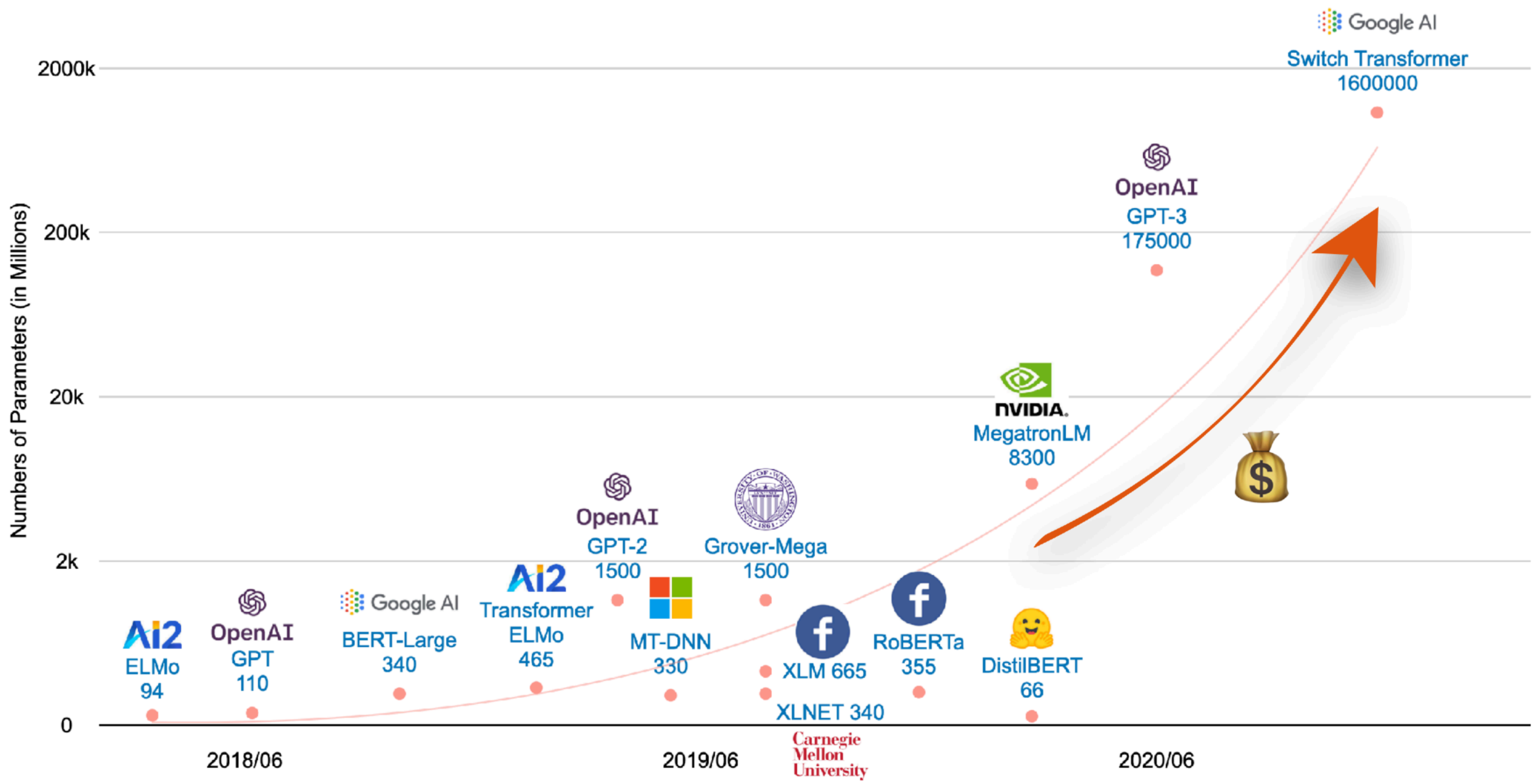
Language Model (BERT, T5, GPT3/4)

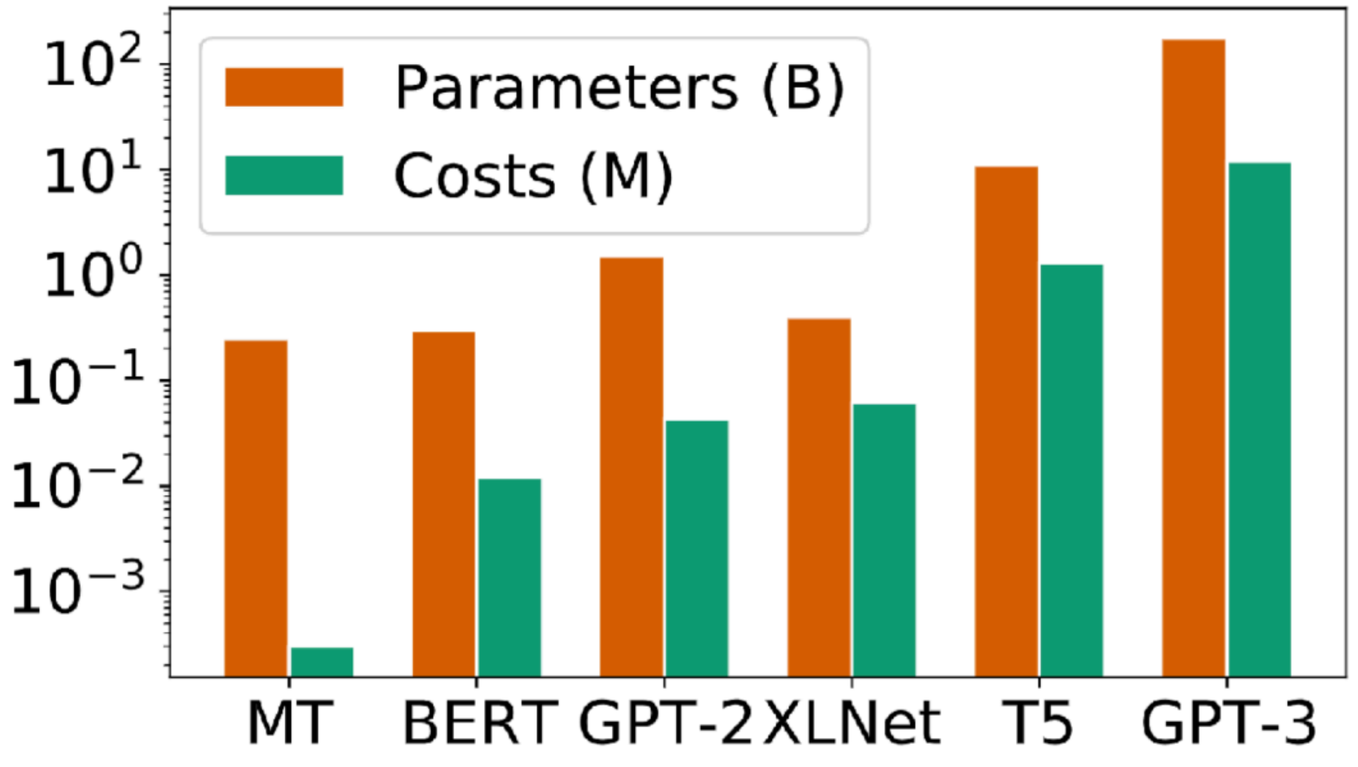
Text to Image (Stable Diffision)

Speech Recognition (wav2vec, HuBERT)

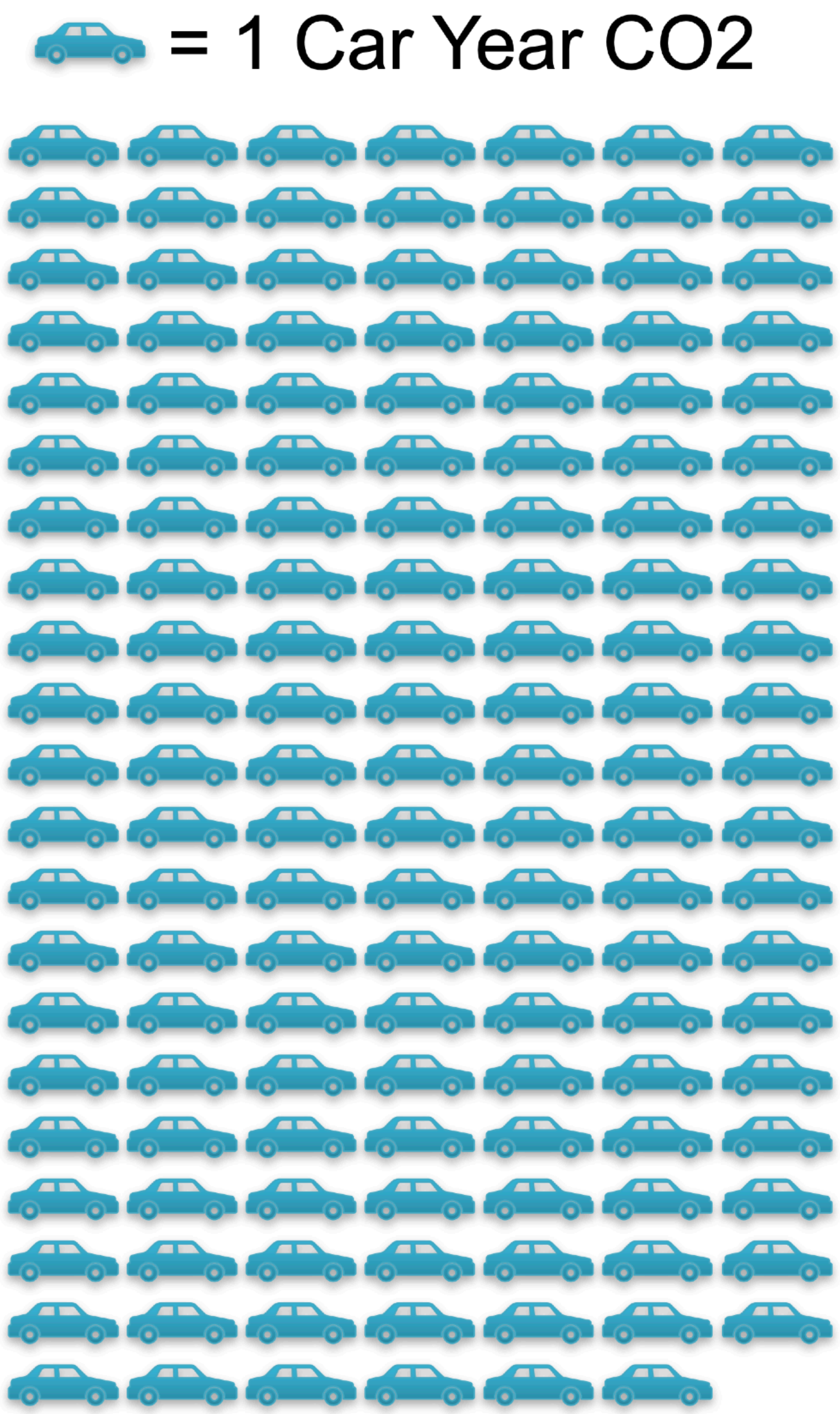
Slide credit: Lei Li

Why we need systems optimization?
LLMs are expensive





Carbon footprint:
Training GPT3 =
driving a car for
146 years!

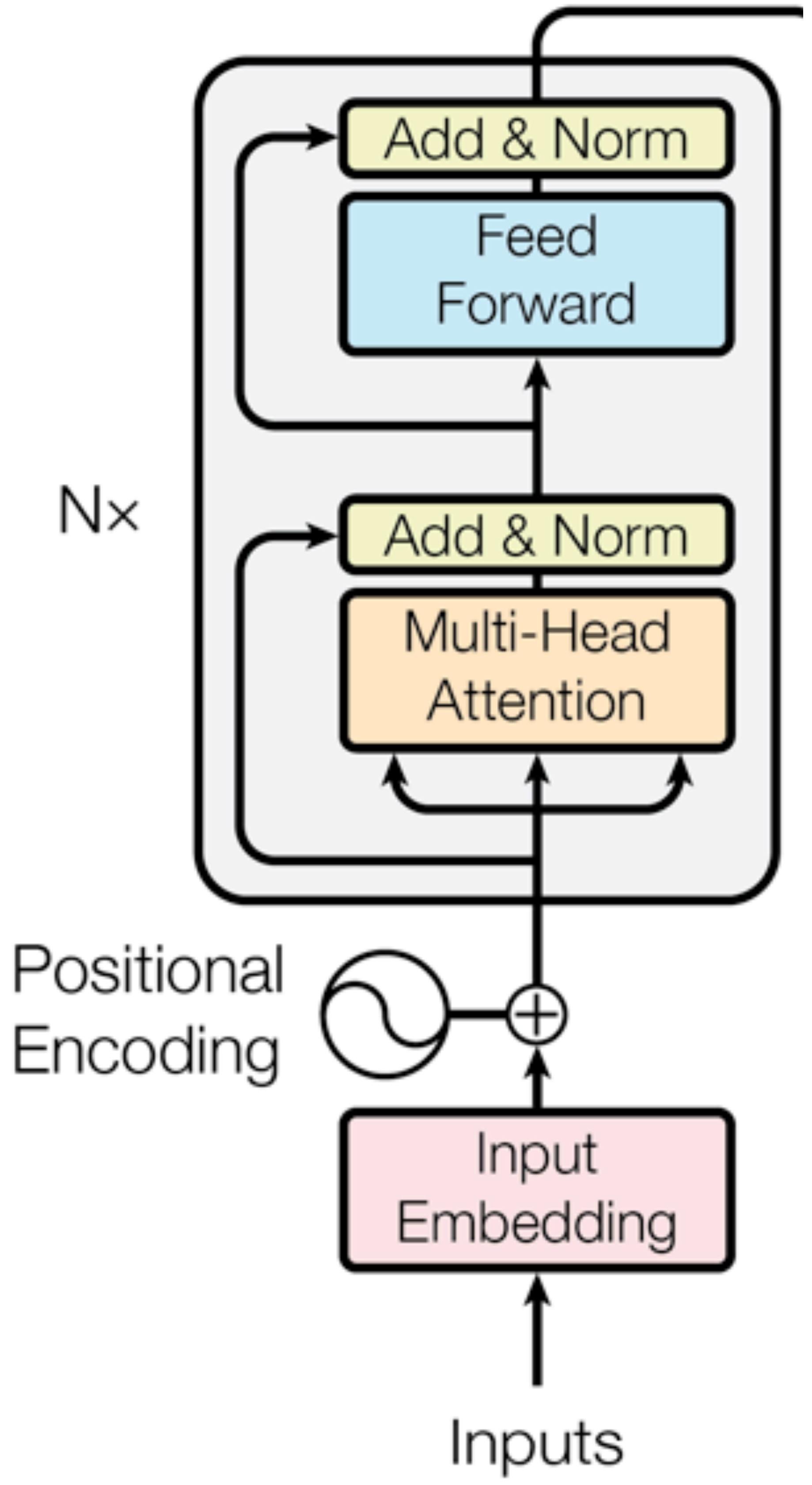


Slide credit: Lei Li

Overview

- (All) Transformer math you need to know
- Training systems
- Inference systems

Transformer architecture



From the bottom to the top:

- Input embedding
- Positional encoding
- A stack of Transformer encoder layers

Transformer encoder is a stack of N layers, which consists of two sub-layers:

- Multi-head attention layer
- Feed-forward layer

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1} \longrightarrow \mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$$

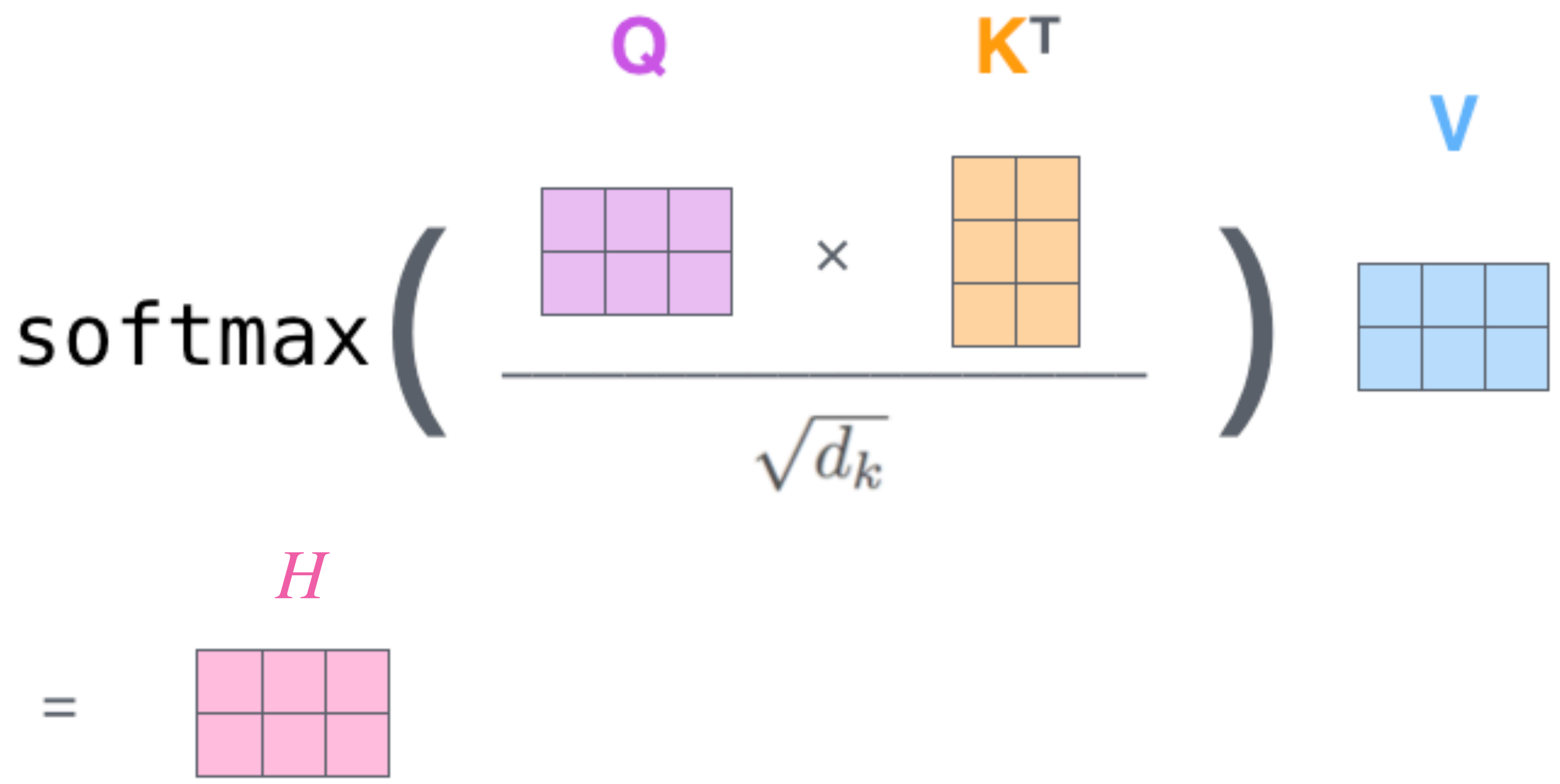
Self-attention

$X \in \mathbb{R}^{n \times d_1}$ (n = input length)

$Q = XW^Q \qquad K = XW^K \qquad V = XW^V$

$$W^Q \in \mathbb{R}^{d_1 \times d_q}, W^K \in \mathbb{R}^{d_1 \times d_k}, W^V \in \mathbb{R}^{d_1 \times d_v}$$

$$\boxed{\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\overbrace{Q}^{n \times d_q}\overbrace{K^T}^{d_k \times n}}{\sqrt{d_k}}\right)\overbrace{V}^{n \times d_v}}$$

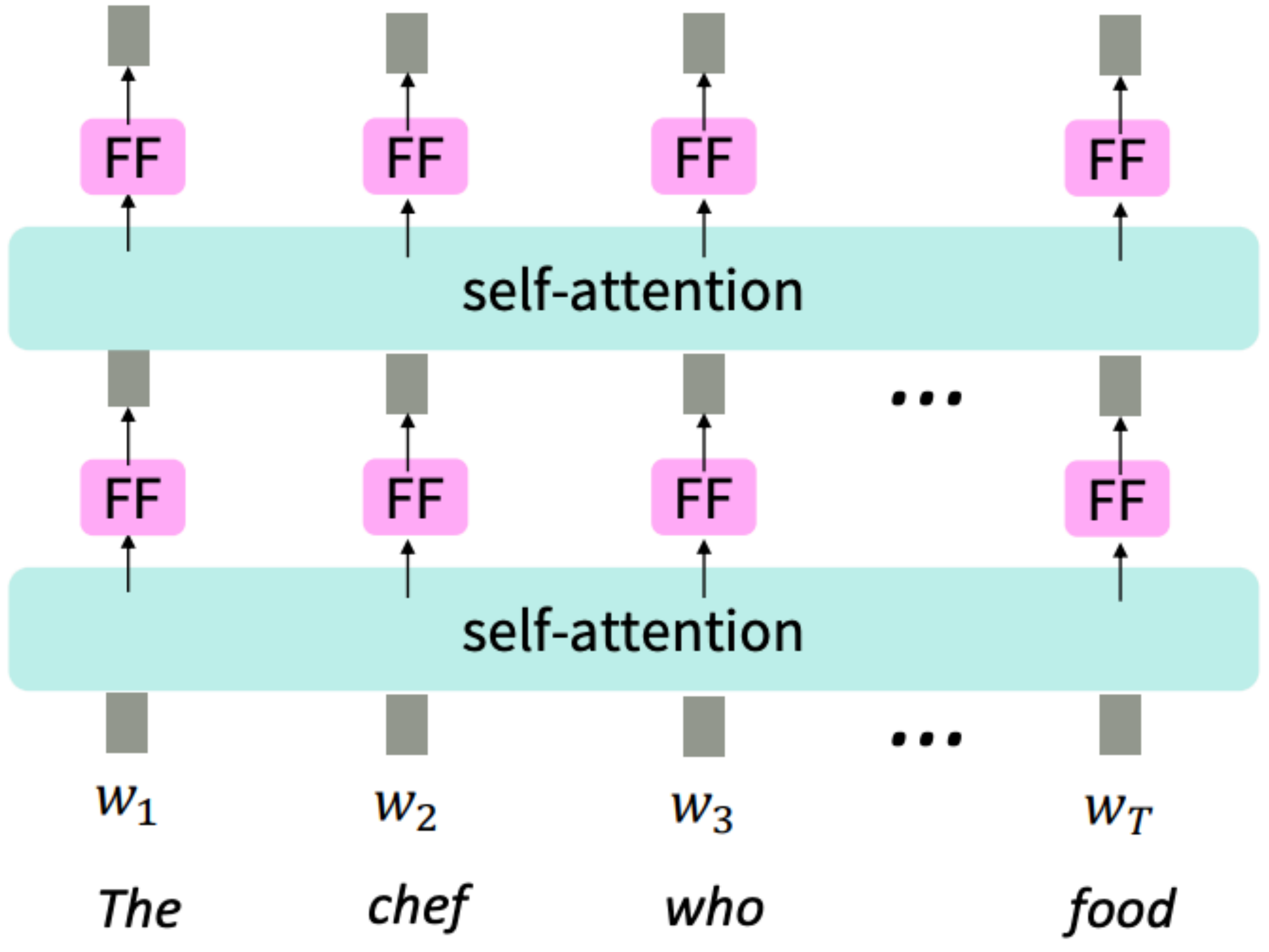


https://jalammar.github.io/illustrated-transformer/

Feed-forward Network (MLP)

- There are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors

- Simple fix: add a feed-forward network to post-process each output vector



$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W_2} + \mathbf{b}_2$$

$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

In practice, they use $d_{ff} = 4d$

This is actually where the majority of the compute and parameters go!

Residual connection & layer normalization

Add & Norm: $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

Residual connections (He et al., 2016)

Instead of $X^{(i)} = \mathrm{Layer}(X^{(i-1)})$ (i represents the layer)

We let $X^{(i)} = X^{(i-1)} + \mathrm{Layer}(X^{(i-1)})$, so we only need to learn "the residual" from the previous layer

Gradient through the residual connection is 1 - good for propagating information through layers

Residual connection unlocks deep networks!

Residual connection & layer normalization

Add & Norm: $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

Layer normalization (Ba et al., 2016) helps train model faster

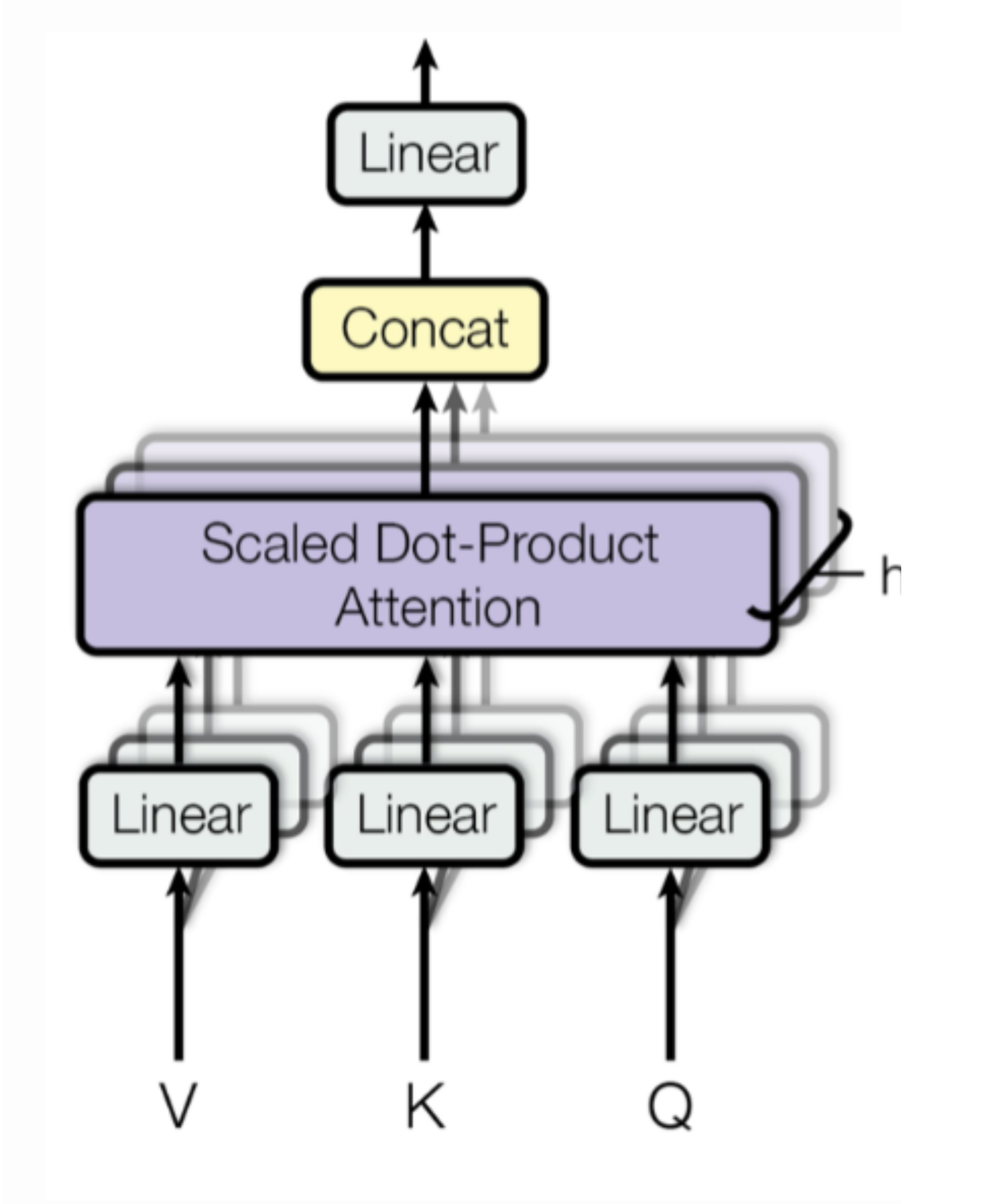
Idea: normalize the hidden vector values to unit mean and stand deviation within each layer

$$\boxed{y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta}$$

$\gamma, \beta \in \mathbb{R}^d$ are learnable parameters

LayerNorm is crucial for stable training with deep networks

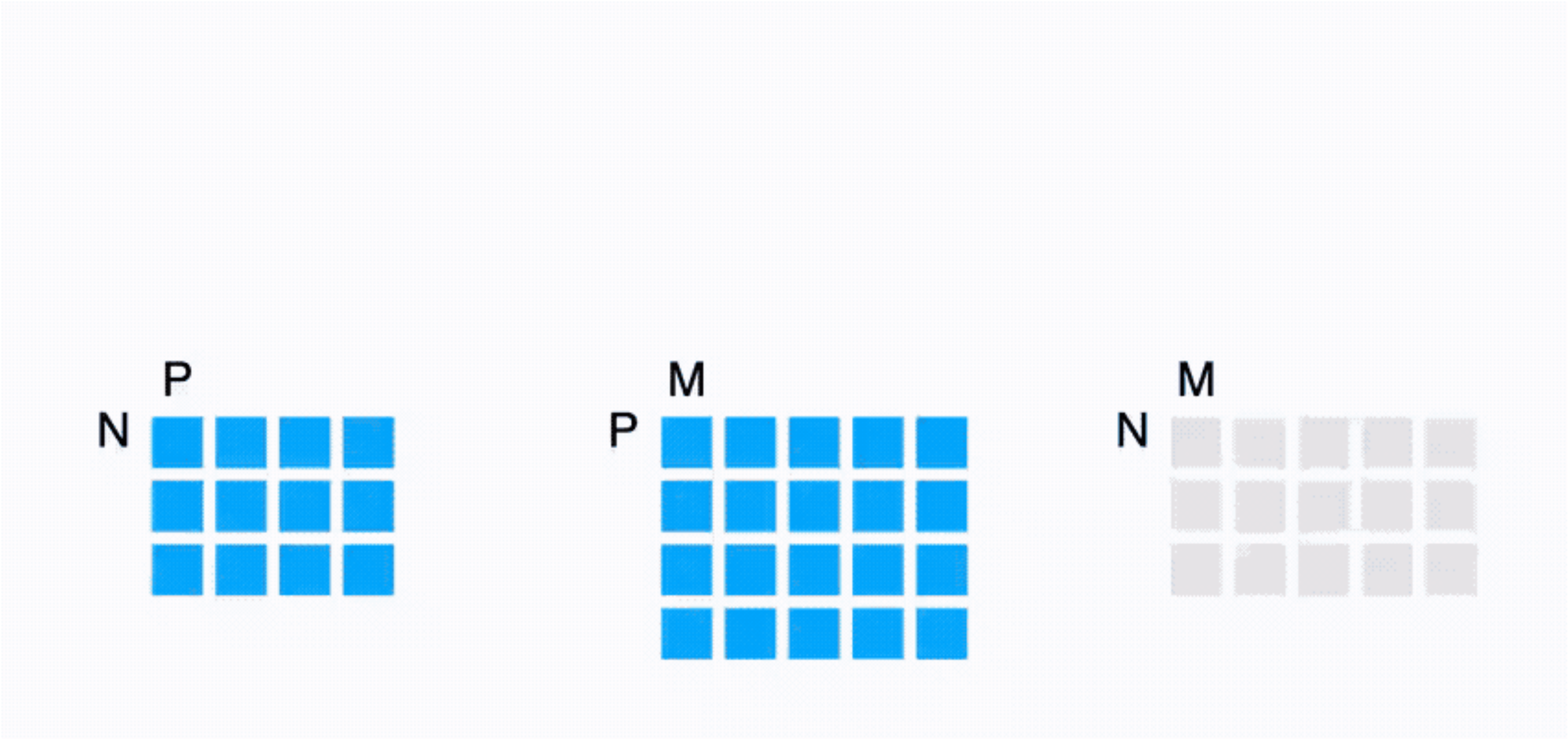
Transformer Math: Parameters



- Embed dim D, number of layers L, vocab size V, context length N
- MHA layer: $4D^2$ ($3D^2$ for QKV projection, D^2 for output projection)
- FFN layer: $8D^2$
  FFN($\mathbf{x}_i$) $= W_2\phi(W_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2$
  $\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}$, $\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}$
  Typically $d_{ff} = 4d$
- Embedding and LM head: DV each
- Total: $12LD^2 + DV$ if shared embedding, $12LD^2 + 2DV$ if not shared

Transformer Math: Matrix Multiply

- Matmul size (N x P) @ (P x M) has 2MNP FLOPS



- Per input vector, each weight matrix size N x P has 2NP FLOPS
- Forward pass FLOPS per input vector: 2 x no. (non-embedding) params

https://jax-ml.github.io/scaling-book/transformers/

Transformer Math: Forward and Backward FLOPS

- Forward: $A = W\ X$

- Backward:
  Weight gradient $dW = dA\ X^T$
  Activation gradient $dX = W^T\ dA$
  Twice the FLOPS of forward

- Forward + backward FLOPS per input vector: 6 x no. (non-embedding) params

Transformer Math: Attention FLOPS

- Forward, for input length N: $4N^2D$ ($2N^2D$ for QK^T, $2N^2D$ for PV)
  N: context length, D: model dimension
  Per input vector: $4ND$

- Backward: 2x forward

- Forward + backward FLOPS per input vector: 12LND
  L: number of layers

	S1	S1	EOT	S2	S2
S1	1	0	0	0	0
S1	1	1	0	0	0
EOT	1	1	1	0	0
S2	1	1	1	1	0
S2	1	1	1	1	1

- For causal attention: only compute half the entries -> 6LND
  Convention varies on how to count
  (should it be 12LND, 6LND, or even 8LND or 7LND due to recompute in the backward pass?)

Transformer Math: Total FLOPS

- Forward + backward FLOPS per input vector:
  6 x no. (non-embedding) params + 12LND

  L layers, context length N, model dim D

- Typically approximated as: 6 x no. (non-embedding) params
  when context length is not too long

- Total FLOPS when trained on T tokens: ≈ 6 x no. (non-embedding) params x no. tokens

Hardware

NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935GB/s	2,039GB/s
Max Thermal Design Power (TDP)	300W	400W***
Multi-Instance GPU	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 10GB
Form Factor	PCIe dual-slot air cooled or single-slot liquid cooled	SXM
Interconnect	NVIDIA® NVLink® Bridge for 2 GPUs: 600GB/s ** PCIe Gen4: 64GB/s	NVLink: 600GB/s PCIe Gen4: 64GB/s
Server Options	Partner and NVIDIA-Certified Systems™ with 1-8 GPUs	NVIDIA HGX™ A100-Partner and NVIDIA-Certified Systems with 4,8, or 16 GPUs NVIDIA DGX™ A100 with 8 GPUs

* With sparsity

Technical Specifications

	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)
Multi-Instance GPUs	Up to 7 MIGs @ 10GB each
Form Factor	SXM
Interconnect	**NVIDIA NVLink™:** 900GB/s **PCIe Gen5:** 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs NVIDIA DGX H100 with 8 GPUs
NVIDIA Enterprise	Add-on

*With sparsity

BF16 Dense:
989 TFLOPS

How long does it take to train a model?

- Model FLOPS utilization (MFU)
  MFU = Model FLOPS per sec / theoretical max TFLOPS per sec

- Typical MFU: 30-50% (e.g. 300-500 TFLOPS/sec per H100)

- Why: memory-bandwidth bound operations, communication, power-throttling

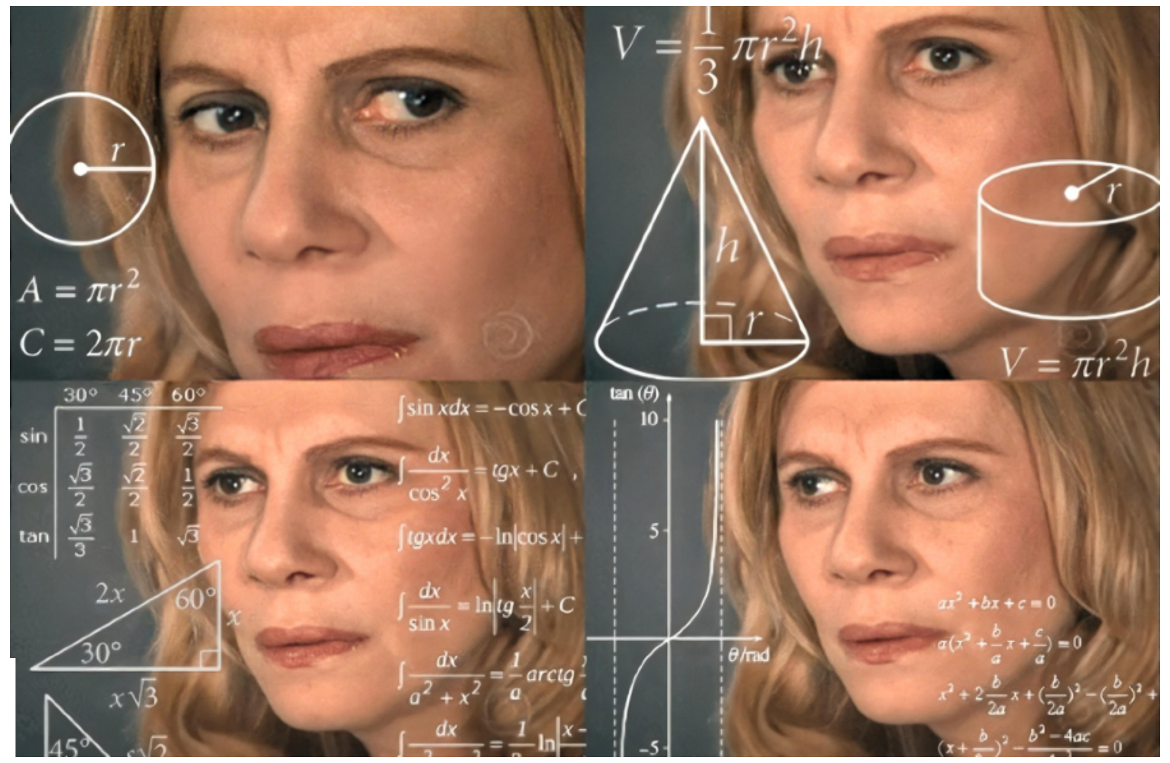
- E.g., how many H100 hours does it take to train a X billion model to Y trillion tokens?
  $\approx 6 * 10^9$X * 10^{12}Y FLOPS / ($400 * 10^{12}$ * 3600 sec)

How long does it take to train a Llama?

- How many H100 hours does it take to train a X billion model to Y trillion tokens?
  $\approx 6 * 10^9 X * 10^{12} Y$ FLOPS / $(400 * 10^{12} * 3600$ sec)

- E.g., Llama-3: 405B, 15T tokens

  ≈ 6 * 405e9 * 15e12 / (400e12 * 3600)
  = 25.3M hours = 66 days on 16K H100



	Training Time (GPU hours)	Training Power Consumption (W)	Training Location-Based Greenhouse Gas Emissions (tons CO2eq)	Training Market-Based Greenhouse Gas Emissions (tons CO2eq)
Llama 3.1 8B	1.46M	700	420	0
Llama 3.1 70B	7.0M	700	2,040	0
Llama 3.1 405B	30.84M	700	8,930	0
Total	39.3M		11,390	0

Overview

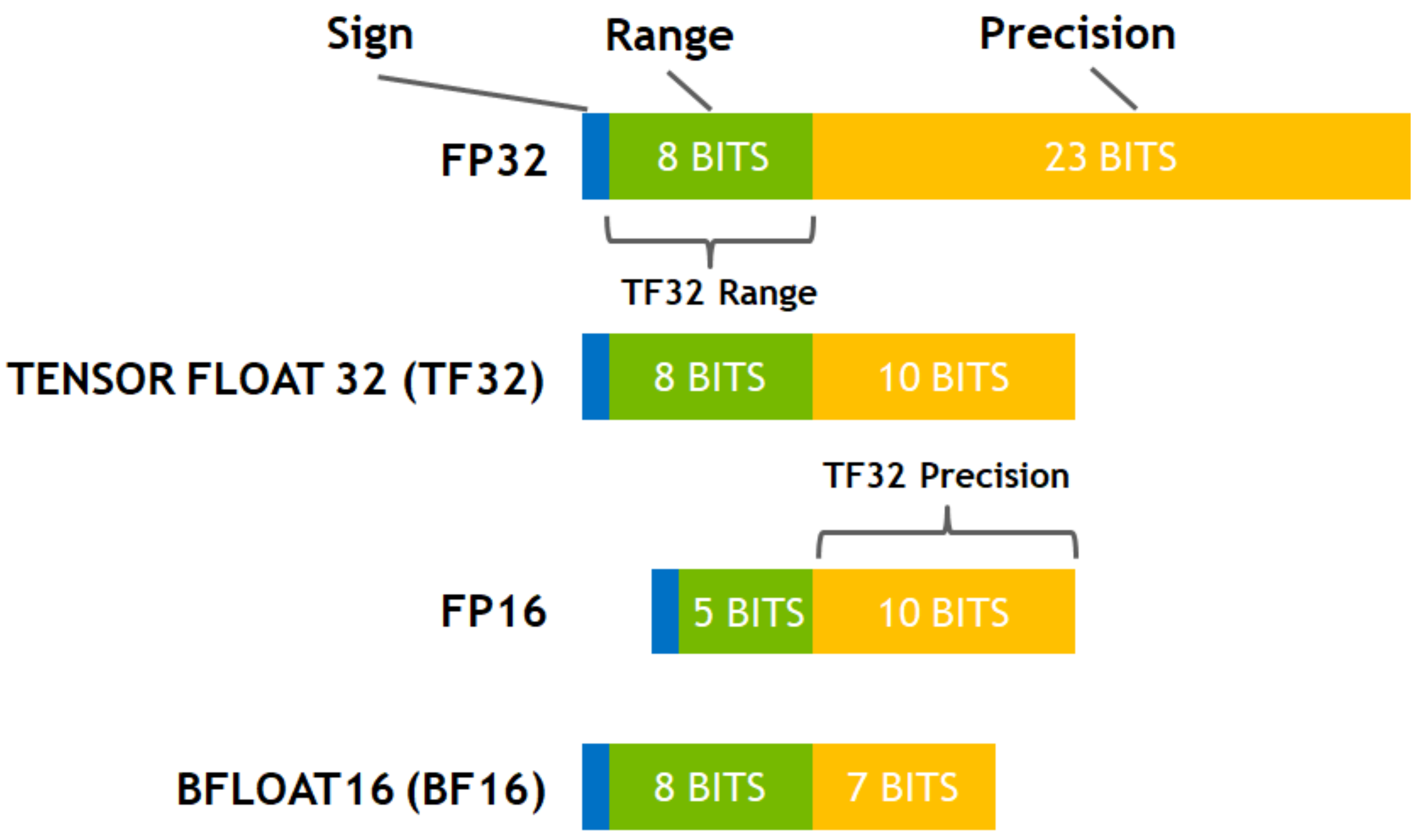
- (All) Transformer math you need to know
- **Training systems**
- Inference systems

Training: Mixed Precision

Technical Specifications	
	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)
Multi-Instance GPUs	Up to 7 MIGs @ 10GB each
Form Factor	SXM
Interconnect	**NVIDIA NVLink™:** 900GB/s **PCIe Gen5:** 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs NVIDIA DGX H100 with 8 GPUs
NVIDIA Enterprise	Add-on

*With sparsity

BF16 Dense:
989 TFLOPS



BF16/FP16 FLOPS are much higher than FP32!

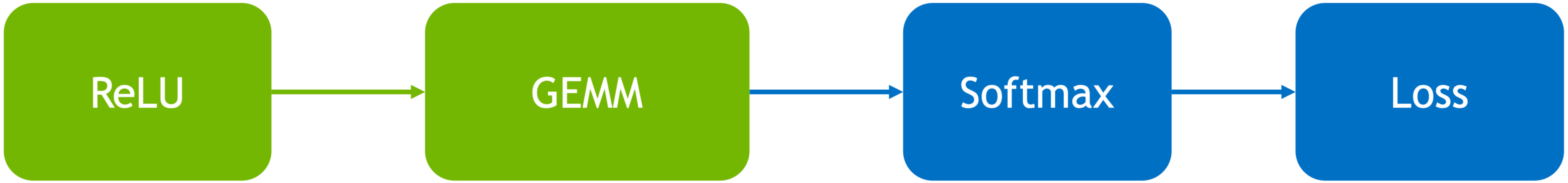
Training: Automatic Mixed Precision

FP16

- GEMMs + Convolutions can use Tensor Cores
- Most pointwise ops (e.g. add, multiply): 1/2X memory storage for intermediates, 2X memory throughput

FP32

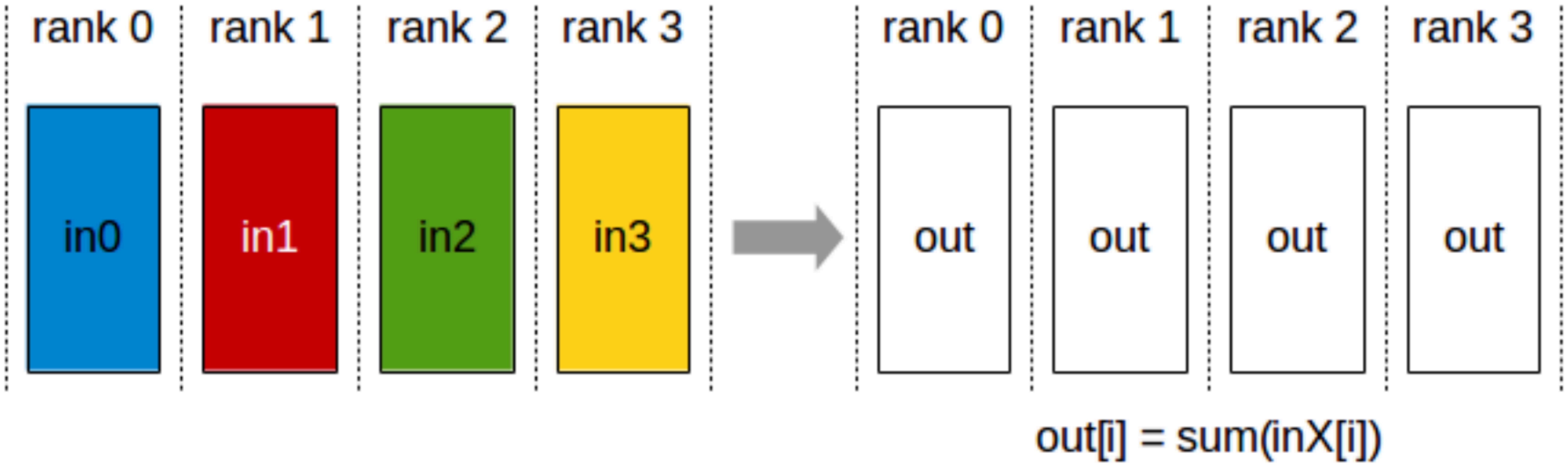
- Weight updates benefit from precision
- Loss functions (often reductions) benefit from precision and range
- Softmax, norms, some other ops benefit from precision and range



Done automatically if you use PyTorch AMP

Distributed Training: Communication AllReduce

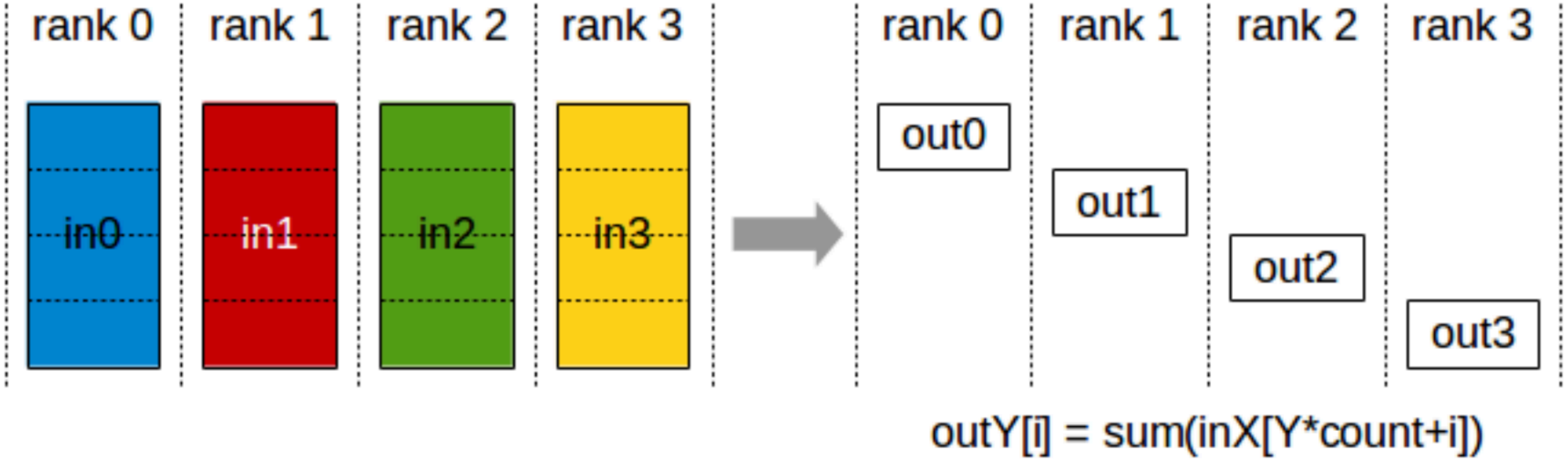
- Compute reduction (sum, min, max) across devices and writing the result in the receive buffers of every rank.



Slide credit: Lei Li

Distributed Training: Communication
ReduceScatter

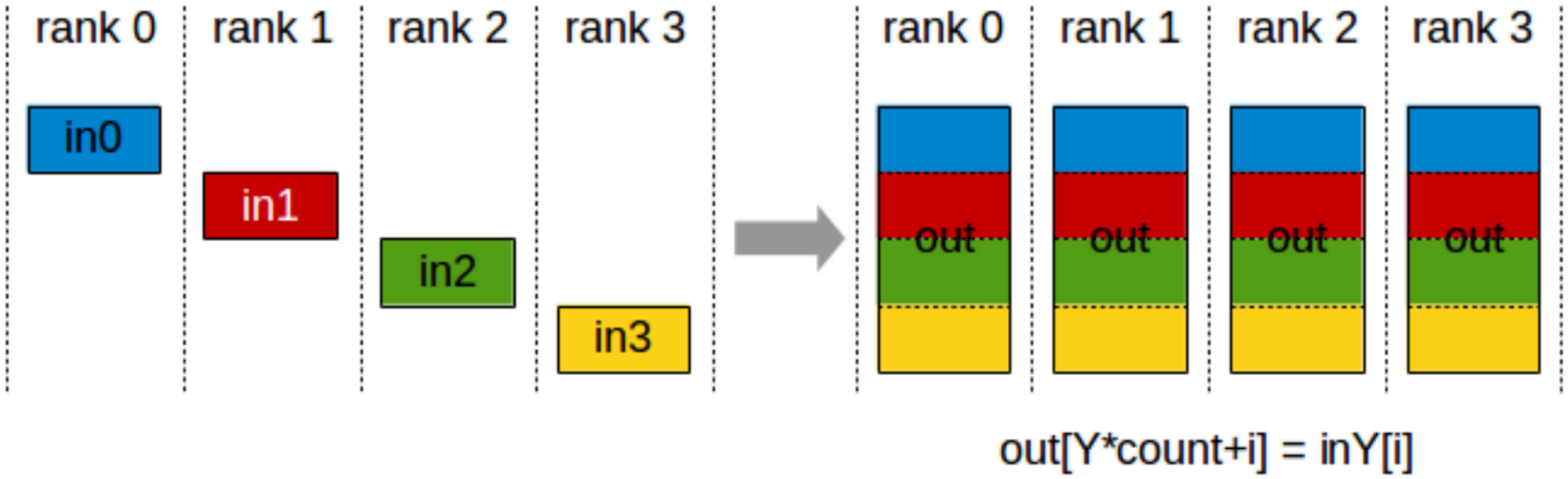
- Compute reduction (sum, min, max) and writing parts of results scattered in ranks



Slide credit: Lei Li

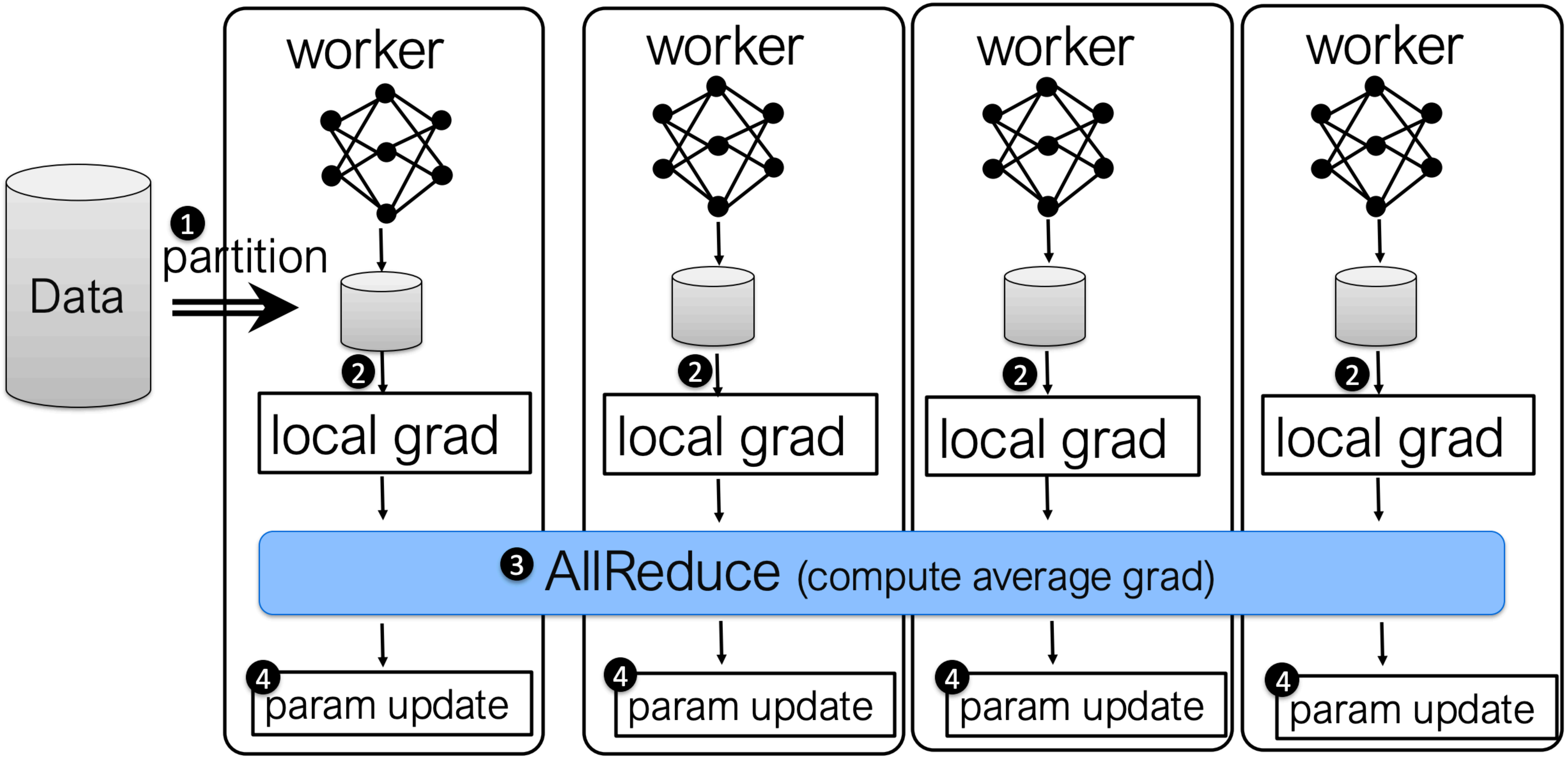
Distributed Training: Communication
AllGather

- gathers N values from k ranks into an output of size k*N, and distributes that result to all ranks (devices).



Slide credit: Lei Li

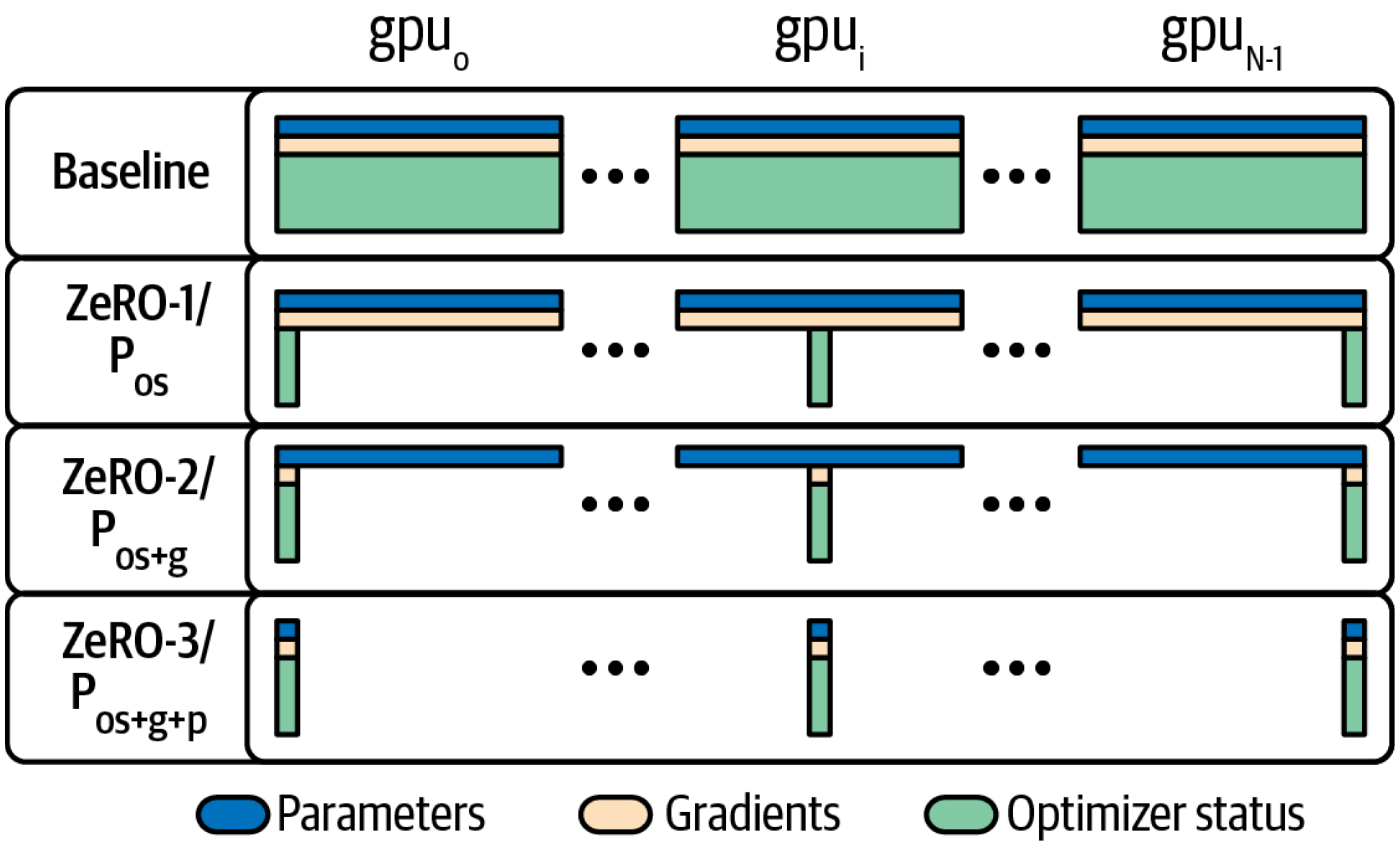
Distributed Training: Data Parallel



PyTorch calls this Distributed Data Parallel (DDP)

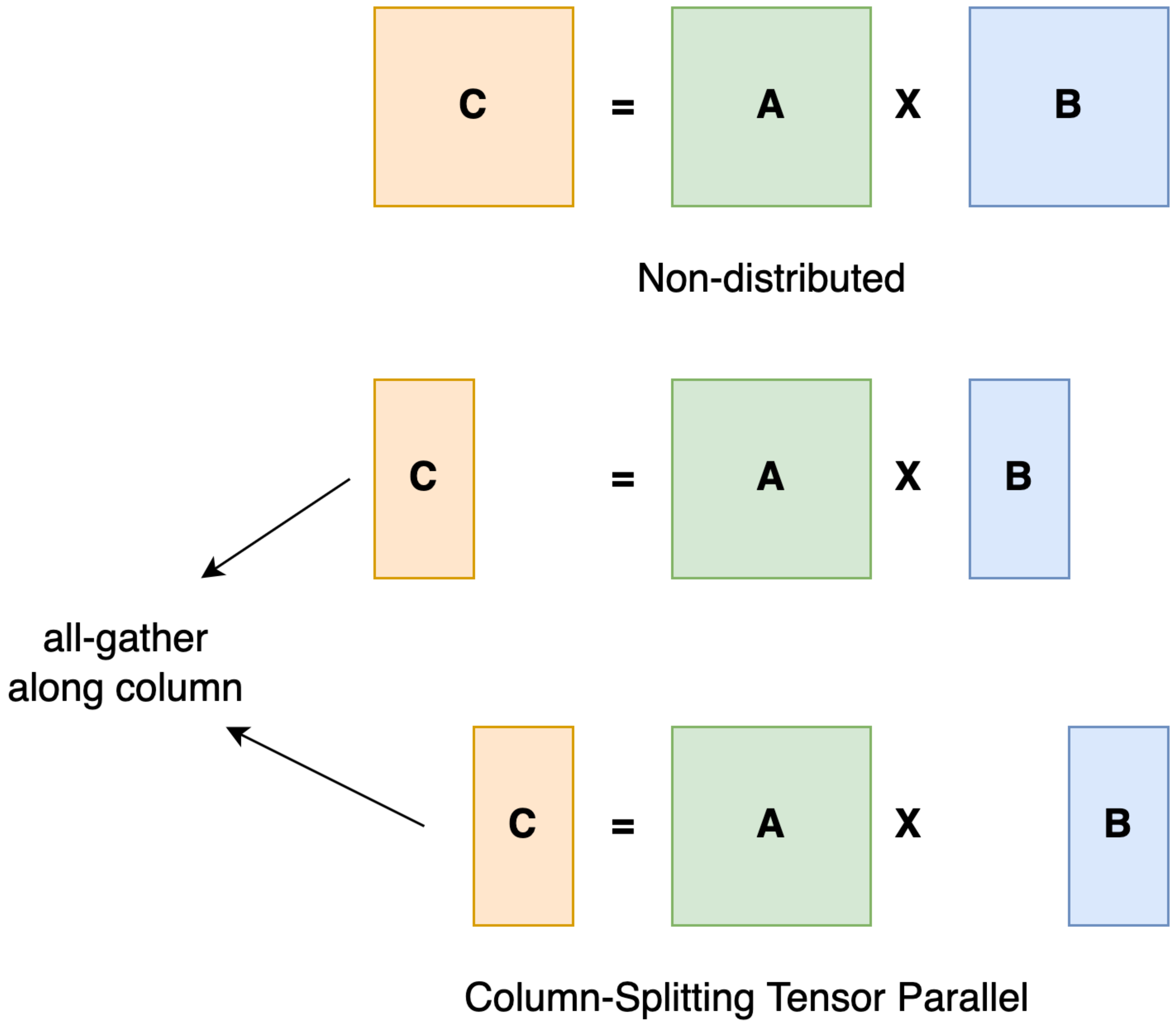
Slide credit: Lei Li

Distributed Training:
Zero Redundancy Optimizer (ZeRO)

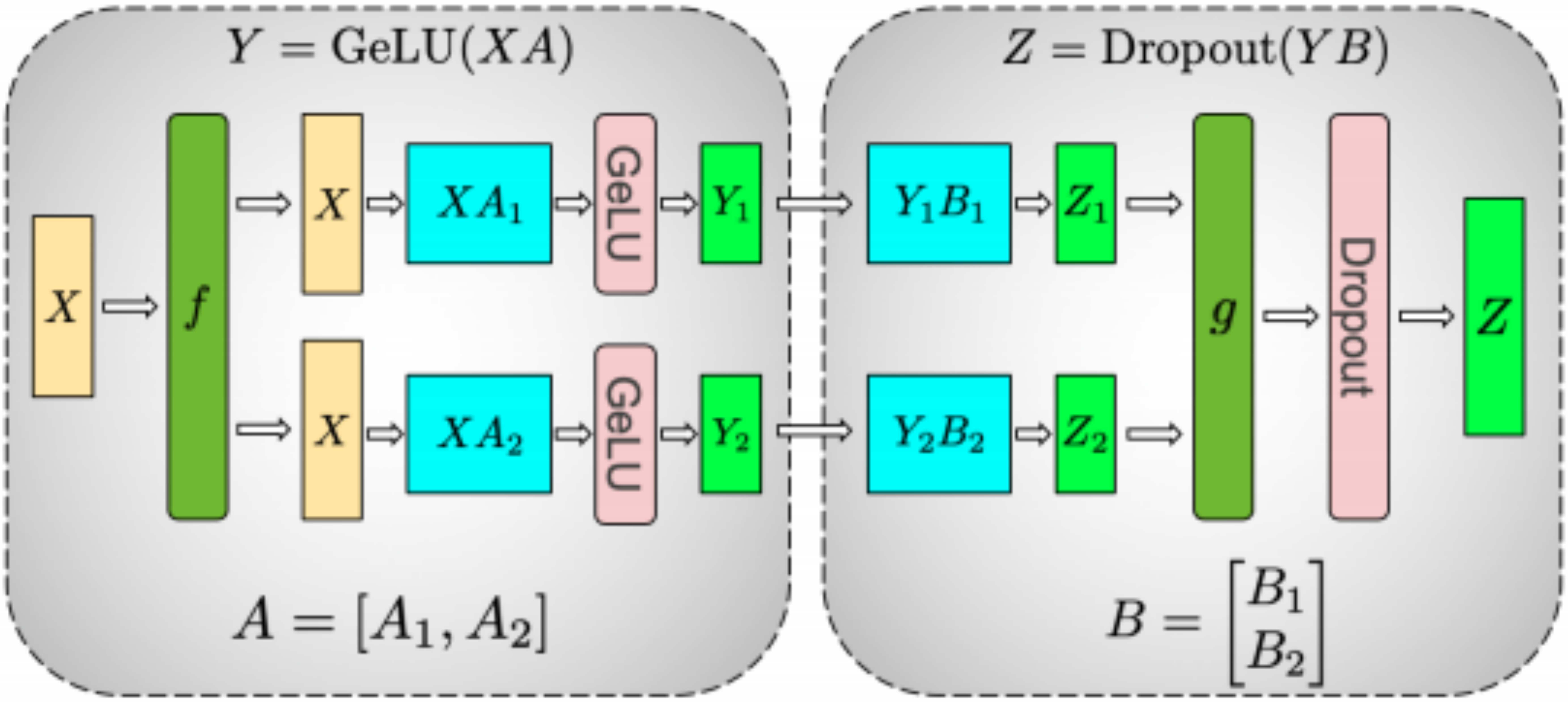


PyTorch: Fully Sharded Data Parallel (FSDP)

Distributed Training: Tensor Parallel



Distributed Training: Tensor Parallel



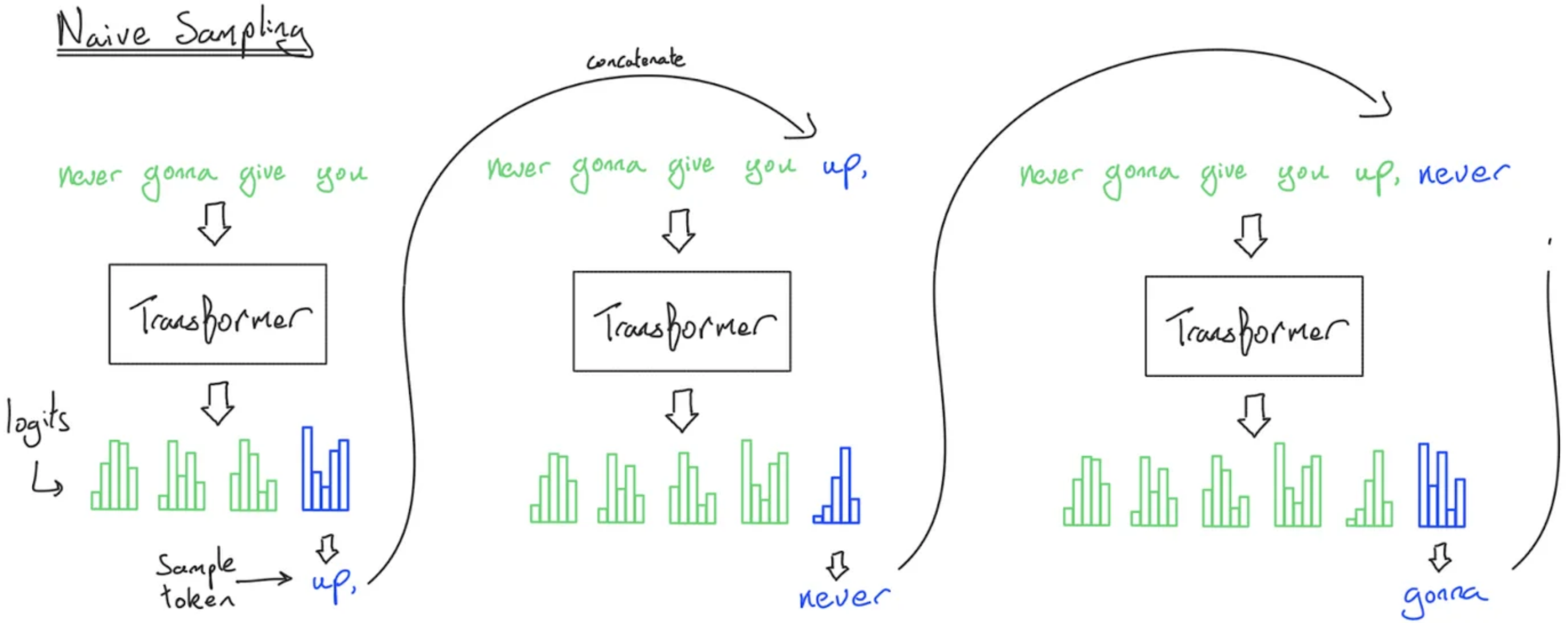
(a) MLP

Shoeybi et al. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Overview

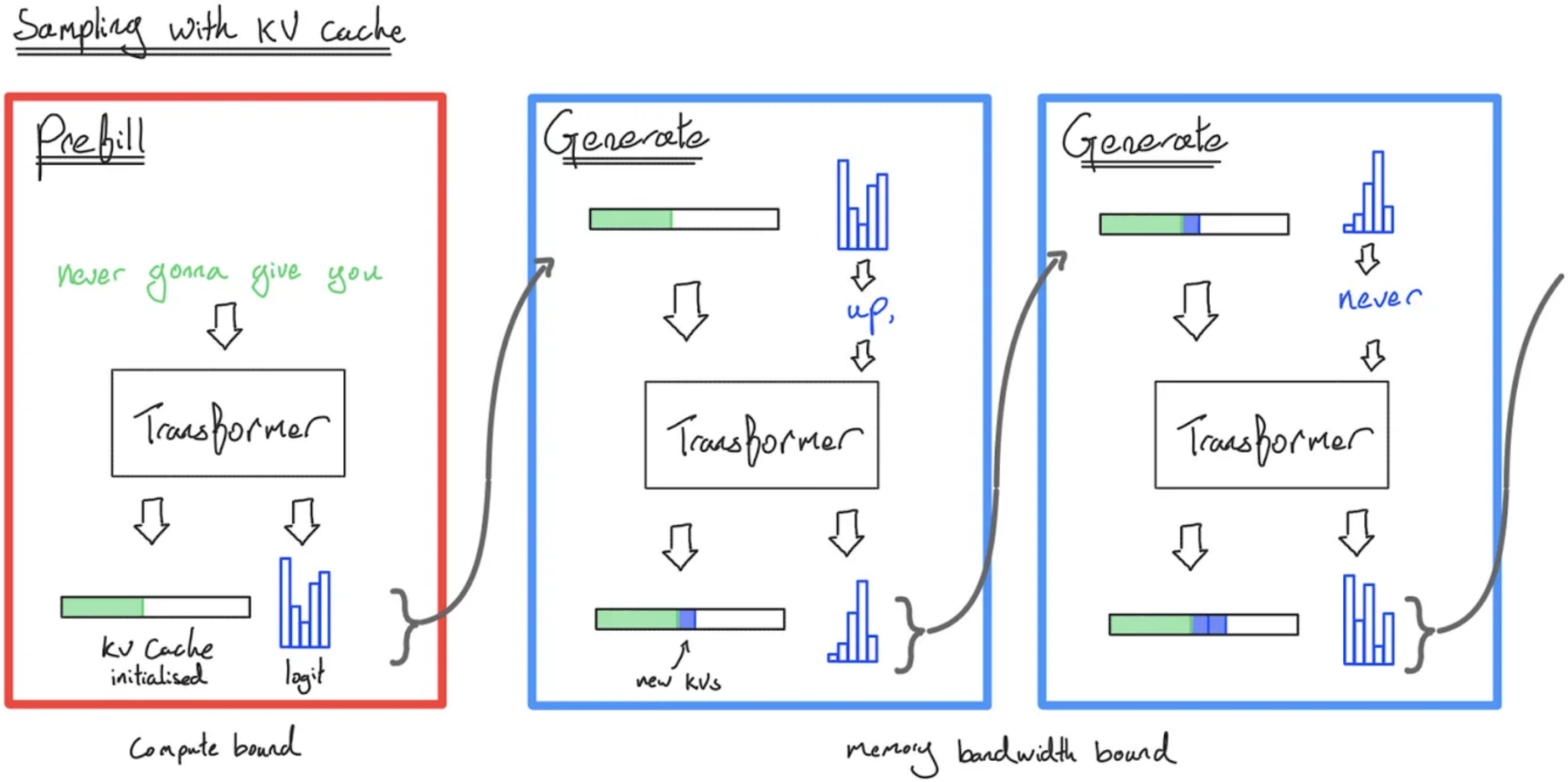
- (All) Transformer math you need to know
- Training systems
- **Inference systems**

Inference: KV caching



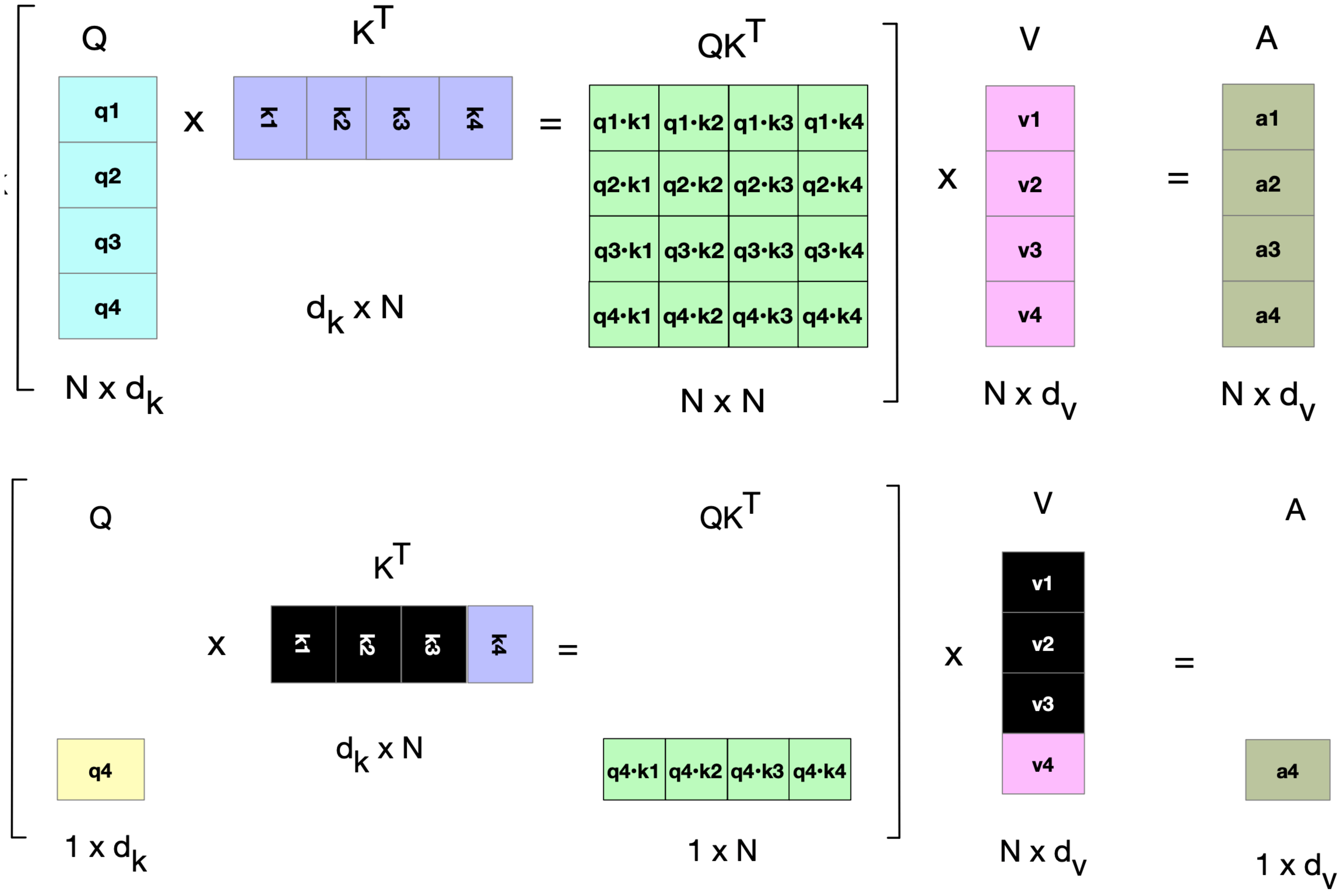
https://jax-ml.github.io/scaling-book/inference/

Inference: KV caching



https://jax-ml.github.io/scaling-book/inference/

Inference: KV caching



Slide credit: Dan Jurafsky

Inference Math

- Decoding is memory-bandwidth bound: Need to load Params + KV cache to memory
- Model bandwidth utilization (MBU) = No. of bytes loaded per sec / Theoretical mem bw
- Typical MBU: 30-70%

gpt-fast, A100 (max 2TB/s):

Model	Technique	Tokens/Second	Memory Bandwidth (GB/s)
Llama-2-7B	Base	104.9	1397.31
	8-bit	155.58	1069.20
	4-bit (G=32)	196.80	862.69
Llama-2-70B	Base	OOM	
	8-bit	19.13	1322.58
	4-bit (G=32)	25.25	1097.66
Llama-3.1-8B	Base	93.89	1410.76
	8-bit	137.64	1030.89
Llama-3.1-70B	Base	OOM	
	8-bit	18.04	1253.78

gpt-fast: https://github.com/pytorch-labs/gpt-fast

Inference: Reducing KV cache with Grouped Query Attention

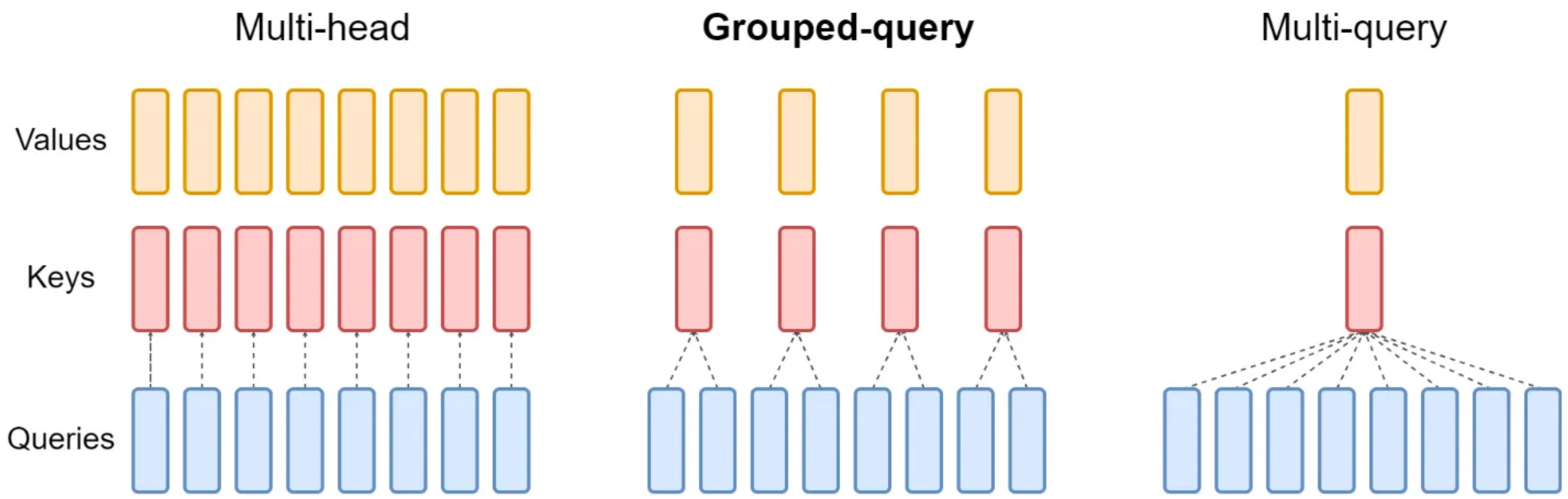
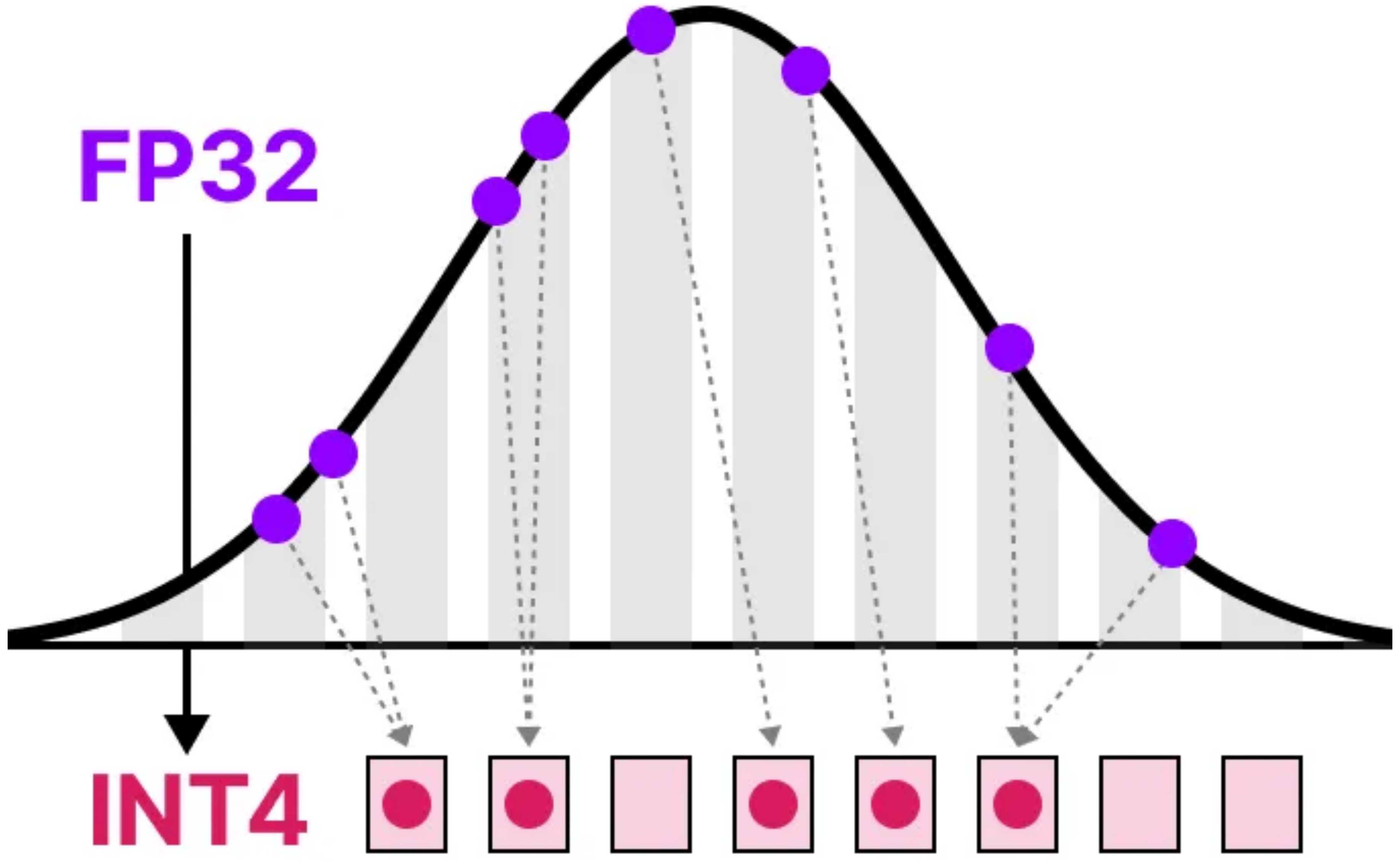


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

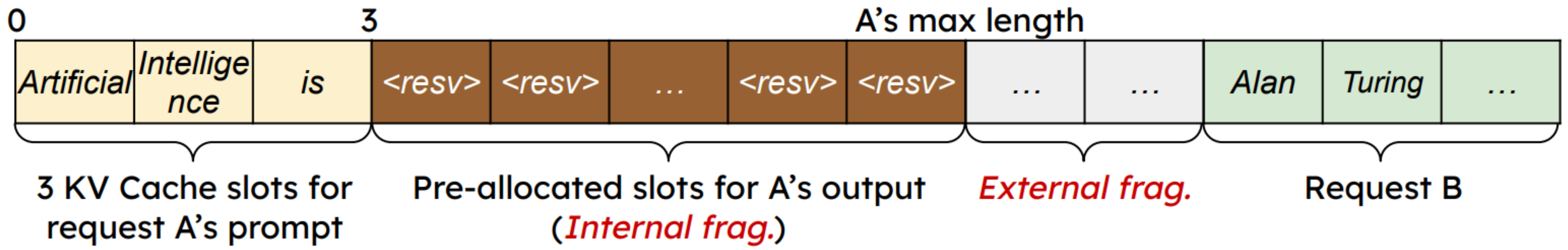
Ainslie et al. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Inference: Weight & KV Cache Quantization



- Reduce no. bytes loaded, at the cost of slightly lower model quality

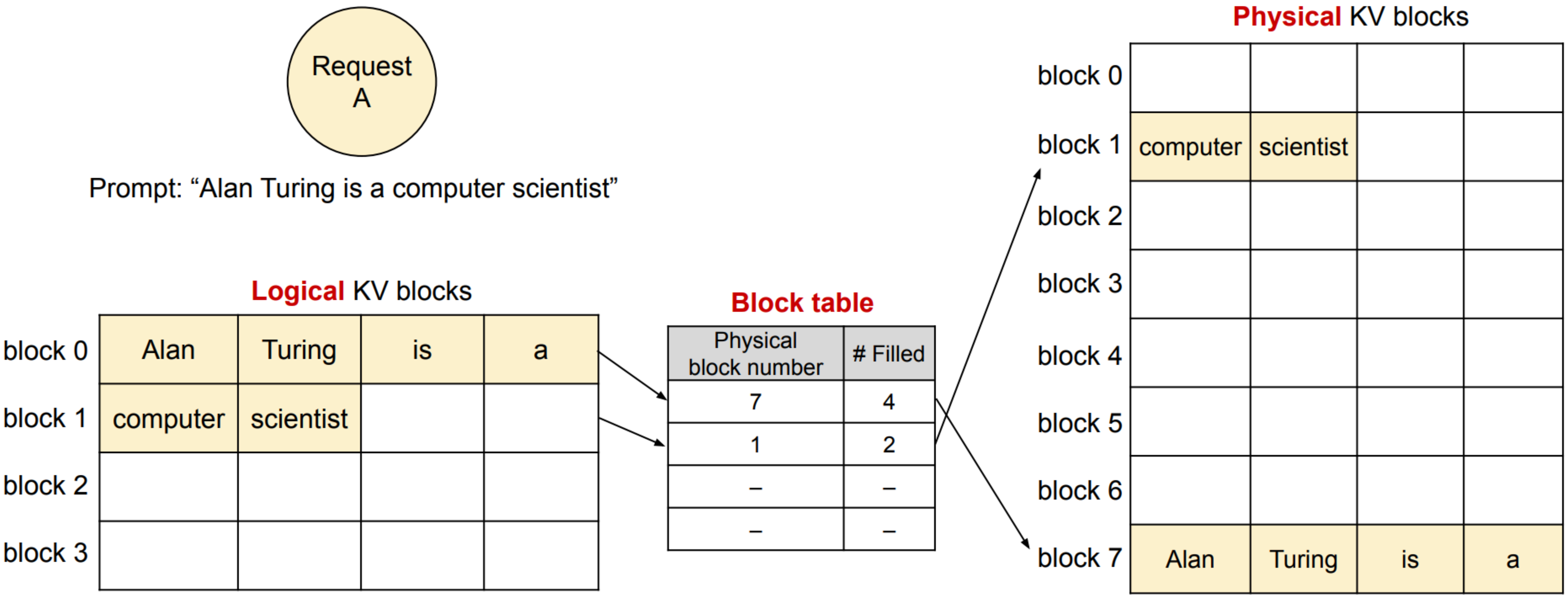
Inference: Challenge with KV cache memory management



- **Pre-allocates contiguous** memory to the request’s max length
- **Memory fragmentation**:
  Internal fragmentation due to unknown output length
  External fragmentation due to non-uniform per-request max length

Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention

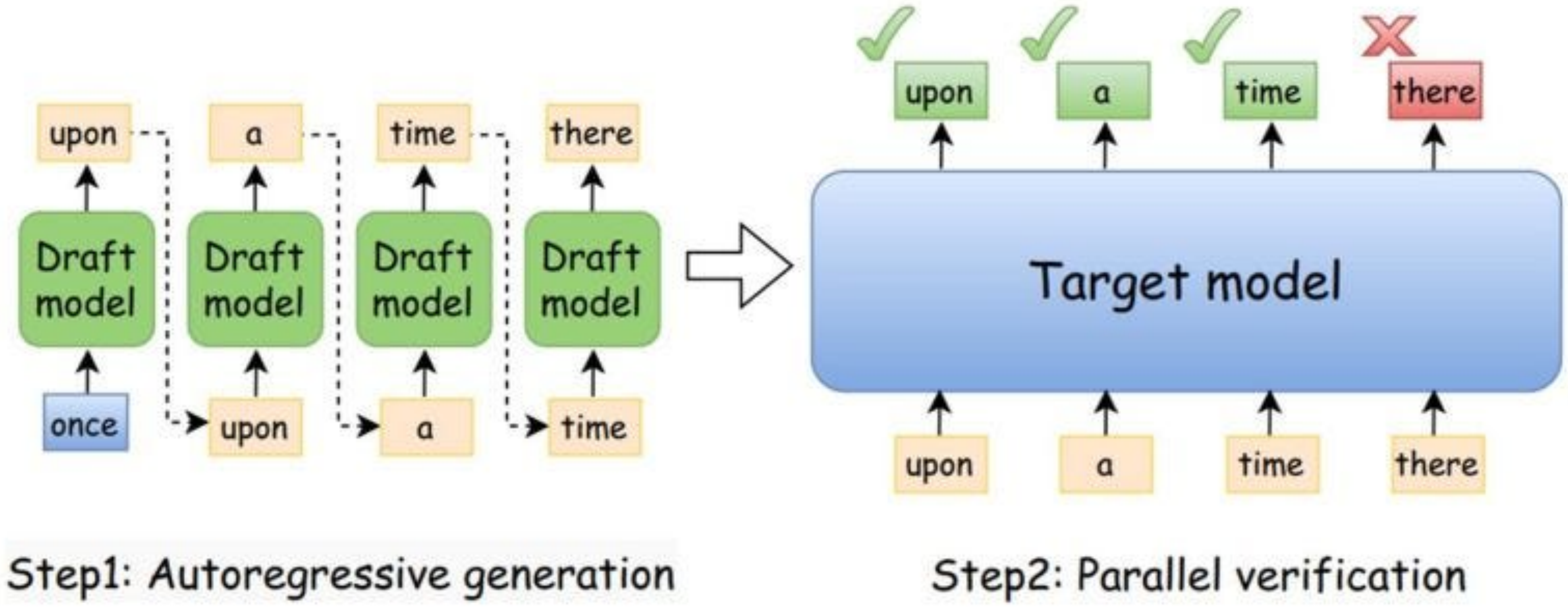
Inference: Memory management with Paged KV



Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention

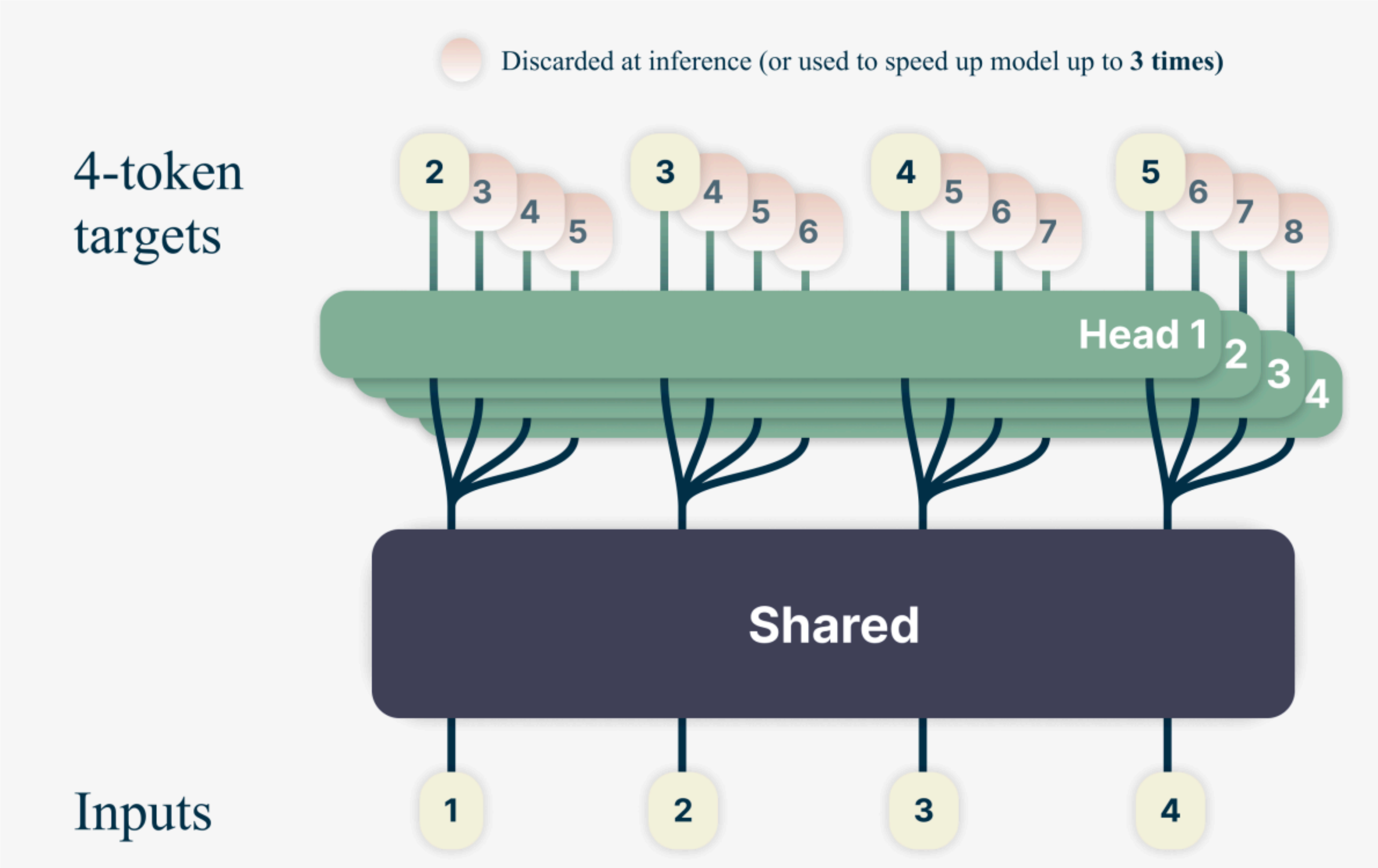
Inference: Speculative Decoding

- Use a small Draft model to predict what the Target model will output
- Verify with Target model multiple tokens in parallel (≈ same cost as processing 1 token)



Leviathan et al. Fast Inference from Transformers via Speculative Decoding

Multi-token Prediction



Gloeckle et al. Better & Faster Large Language Models via Multi-token Prediction

Concluding Thoughts

- Efficient training & inference for LLM is a wide open field
- Intersection of model architecture, systems, and algorithms
- Trend: close co-design of model, inference system, and hardware