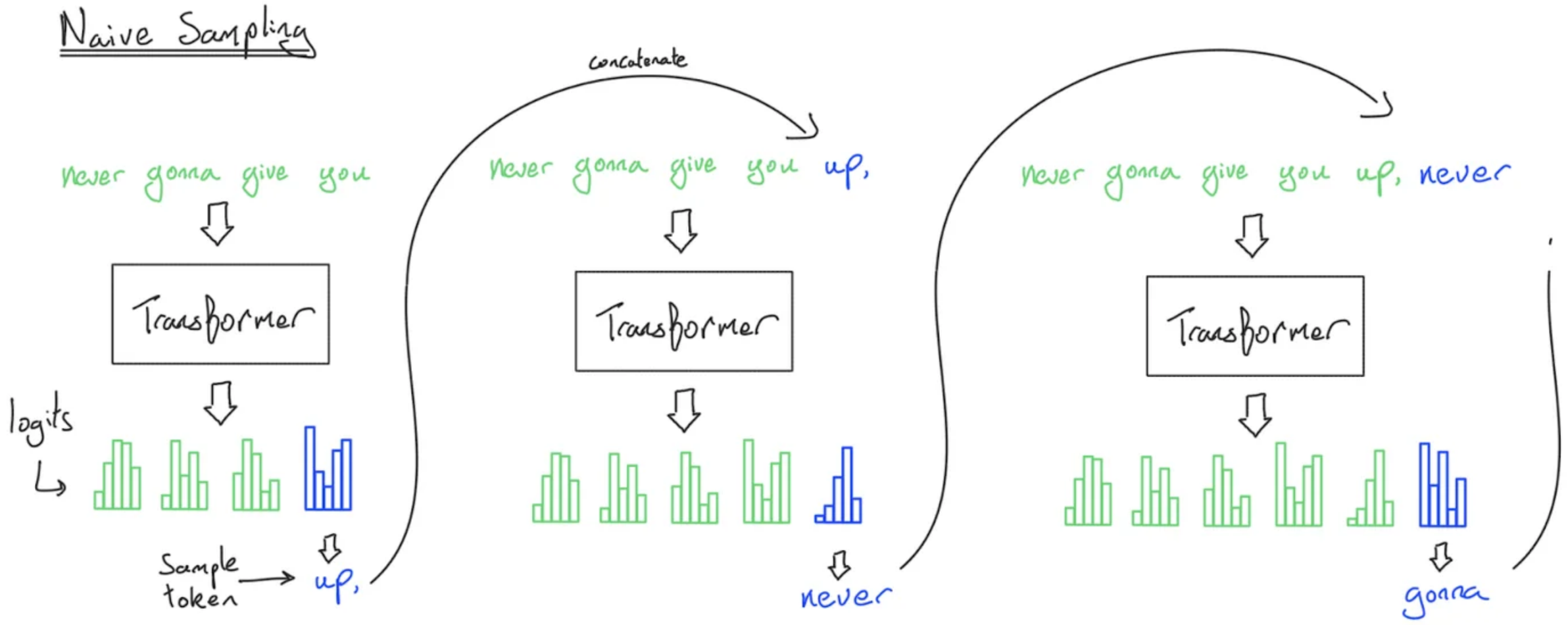
COS 484

L16: Systems for LLM Inference

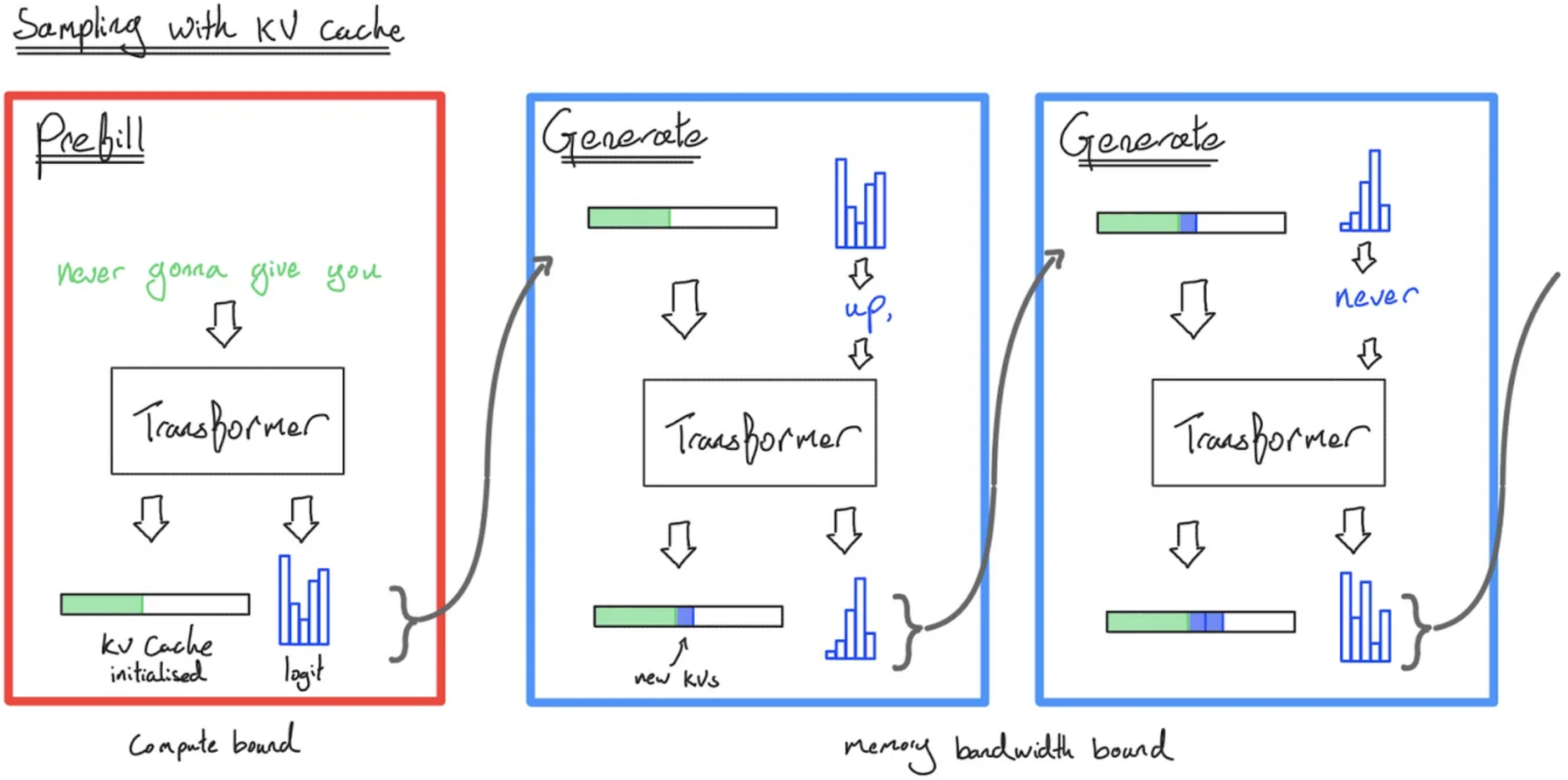
Spring 2026

Inference: KV caching



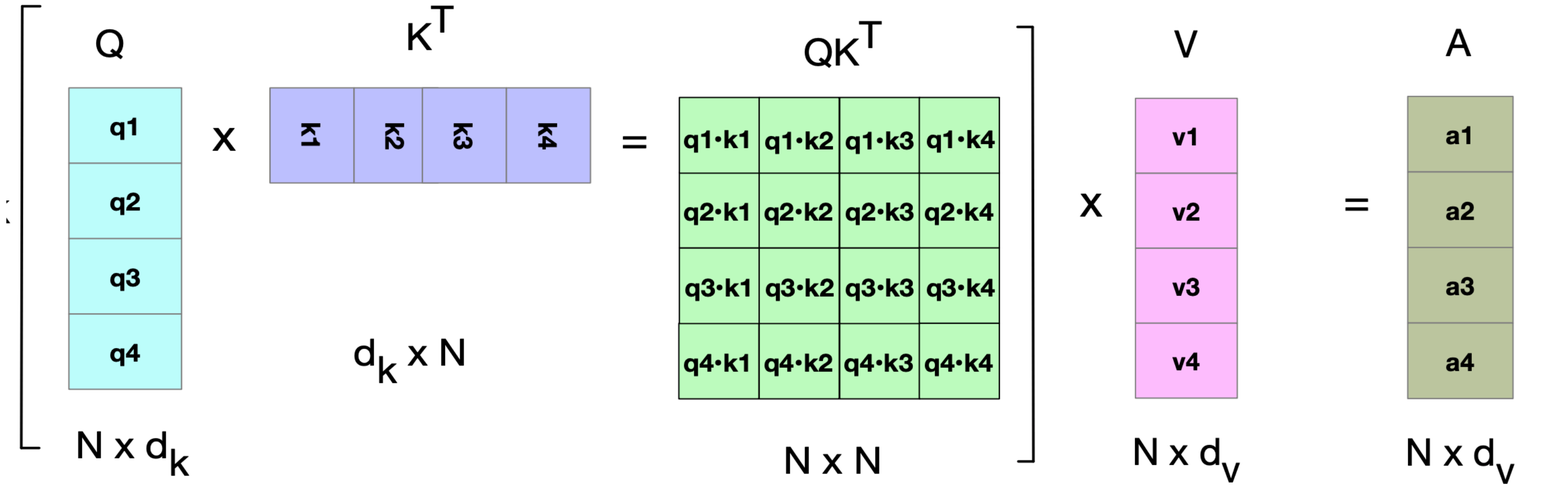
https://jax-ml.github.io/scaling-book/inference/

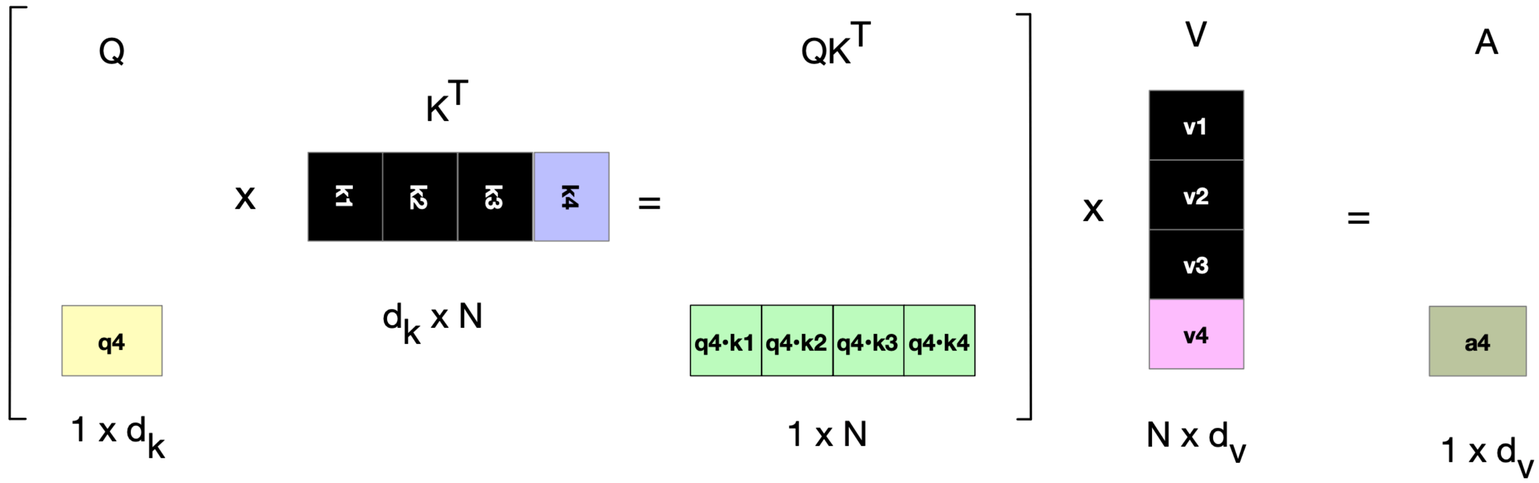
Inference: KV caching



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

Hardware

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

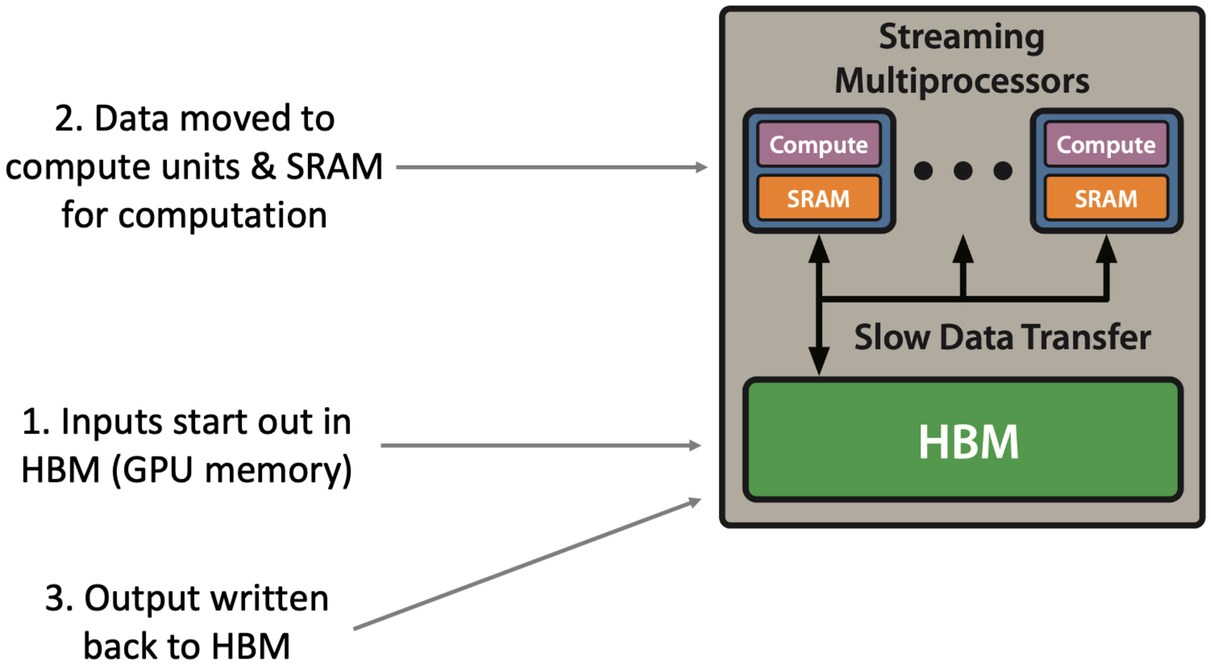
	A100 80GB PCIe	A100 80GB SXM
FP64	**9.7 TFLOPS**	
FP64 Tensor Core	**19.5 TFLOPS**	
FP32	**19.5 TFLOPS**	
Tensor Float 32 (TF32)	**156 TFLOPS | 312 TFLOPS***	
BFLOAT16 Tensor Core	**312 TFLOPS | 624 TFLOPS***	
FP16 Tensor Core	**312 TFLOPS | 624 TFLOPS***	
INT8 Tensor Core	**624 TOPS | 1248 TOPS***	
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**

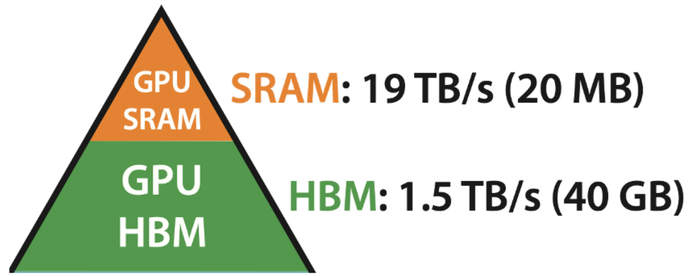
* With sparsity

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

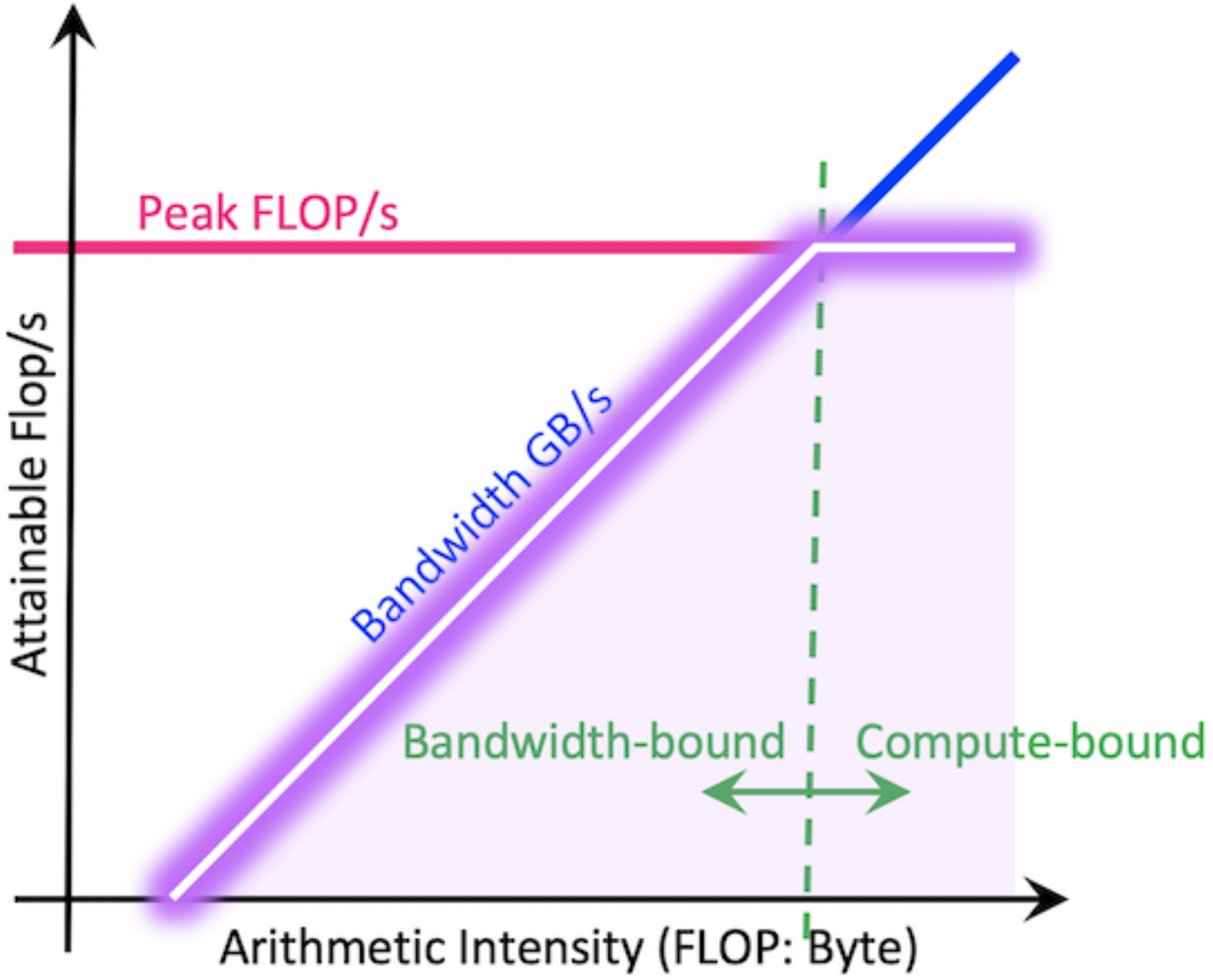
*With sparsity

Background: GPU Compute model

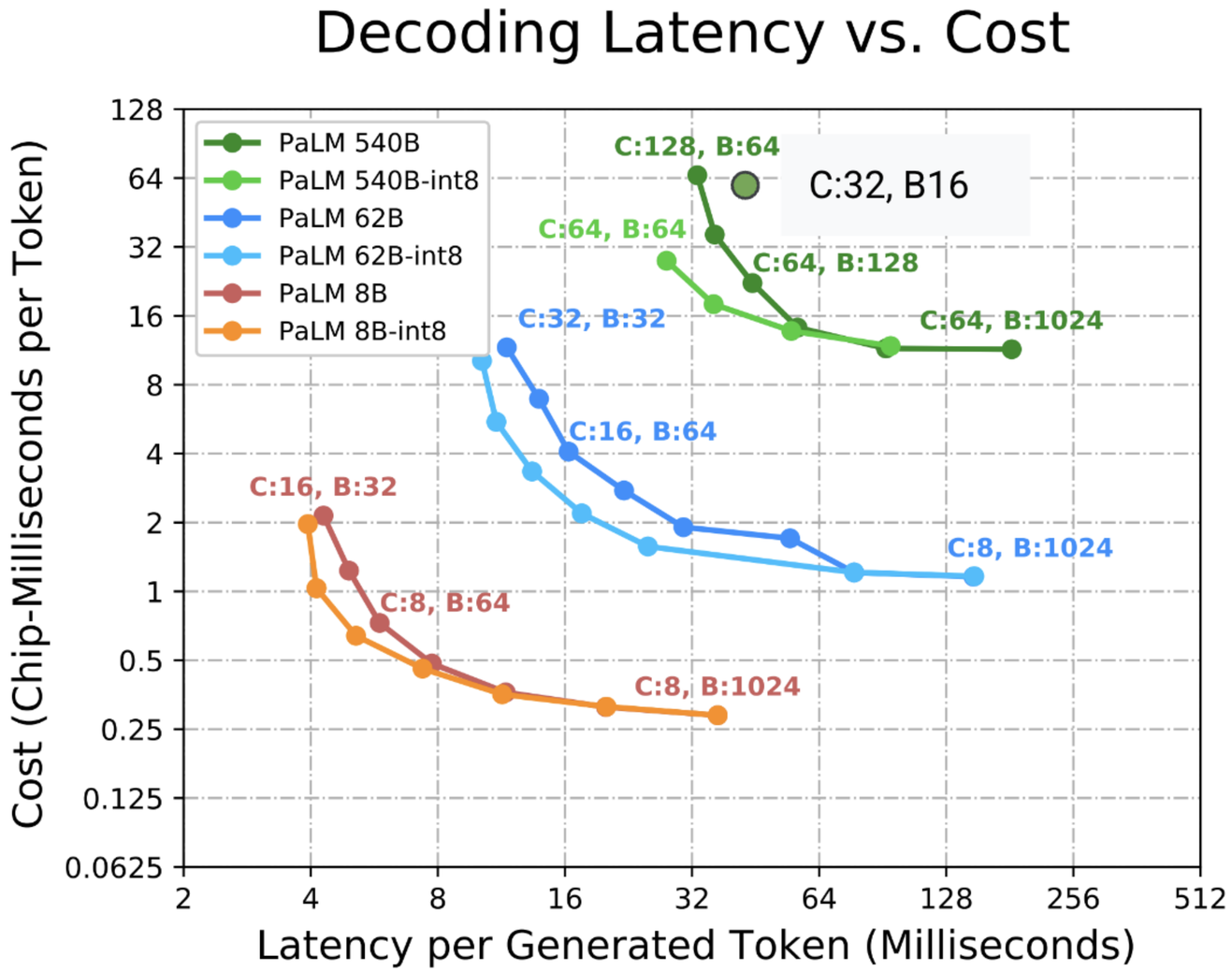




Roofline model



Throughput and Latency Tradeoffs



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

Reducing KV cache with Grouped Query Attention

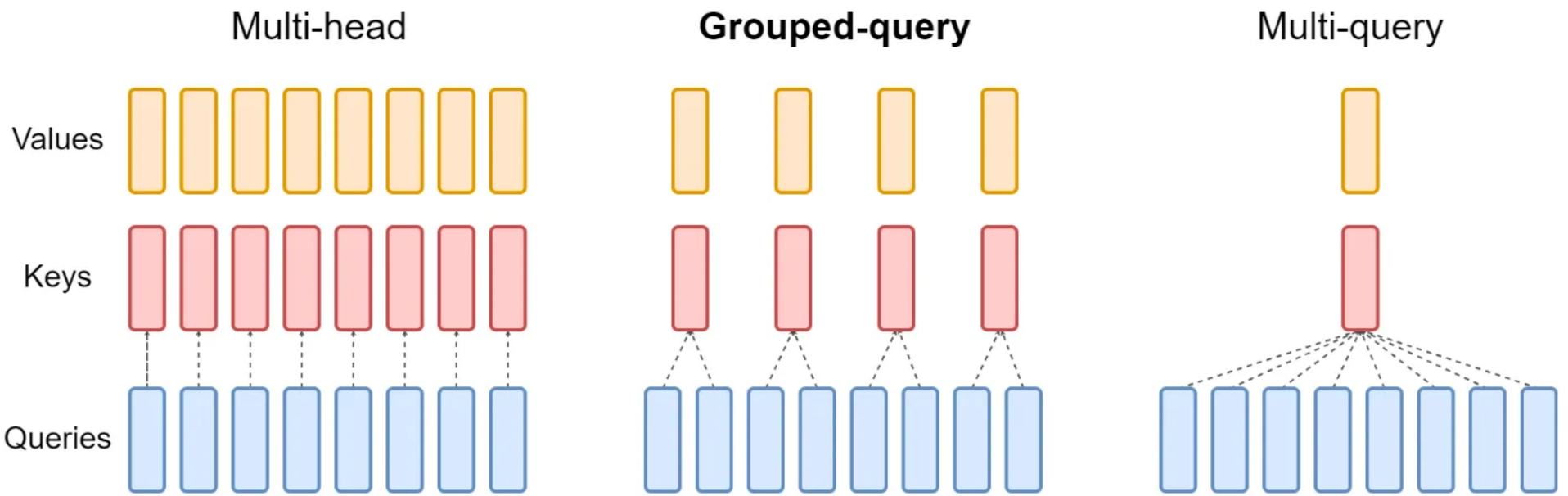


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between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

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

Reducing KV cache with Local (Sliding Window) Attention

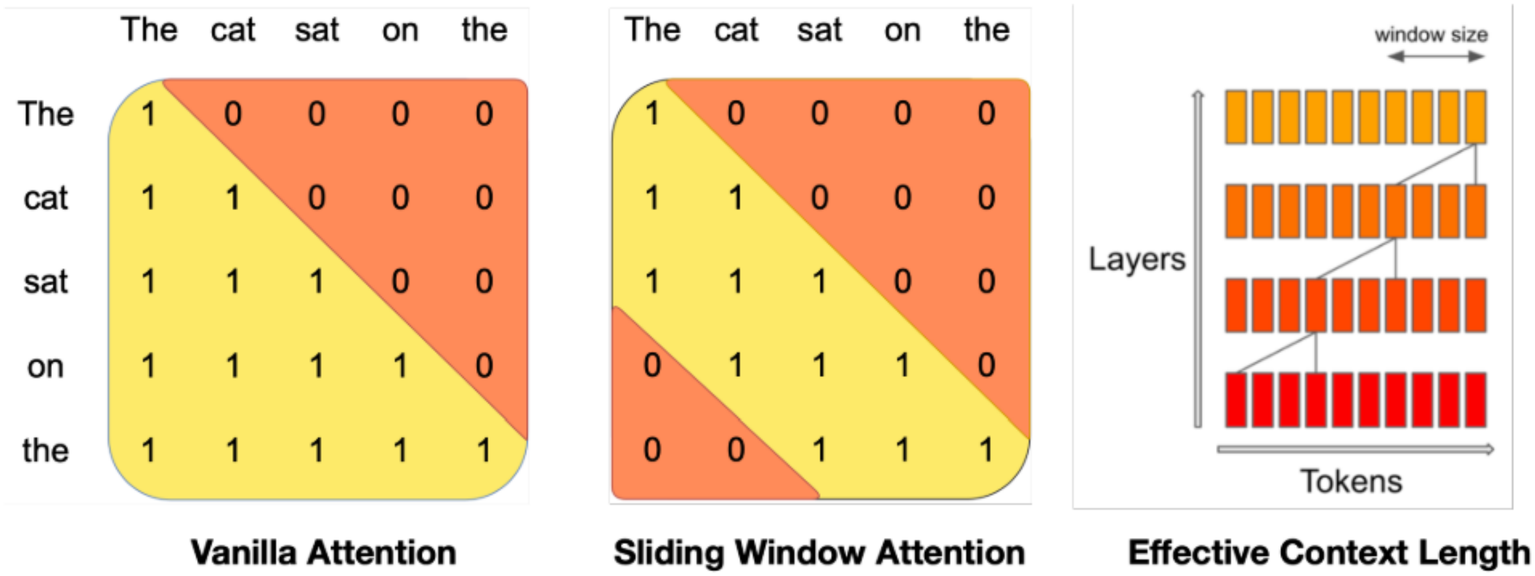


Figure 1: Sliding Window Attention. The number of operations in vanilla attention is quadratic in the sequence length, and the memory increases linearly with the number of tokens. At inference time, this incurs higher latency and smaller throughput due to reduced cache availability. To alleviate this issue, we use sliding window attention: each token can attend to at most W tokens from the previous layer (here, $W = 3$). Note that tokens outside the sliding window still influence next word prediction. At each attention layer, information can move forward by W tokens. Hence, after k attention layers, information can move forward by up to $k \times W$ tokens.

Mistral 7B. https://arxiv.org/abs/2310.06825

Sharing KVs across layers

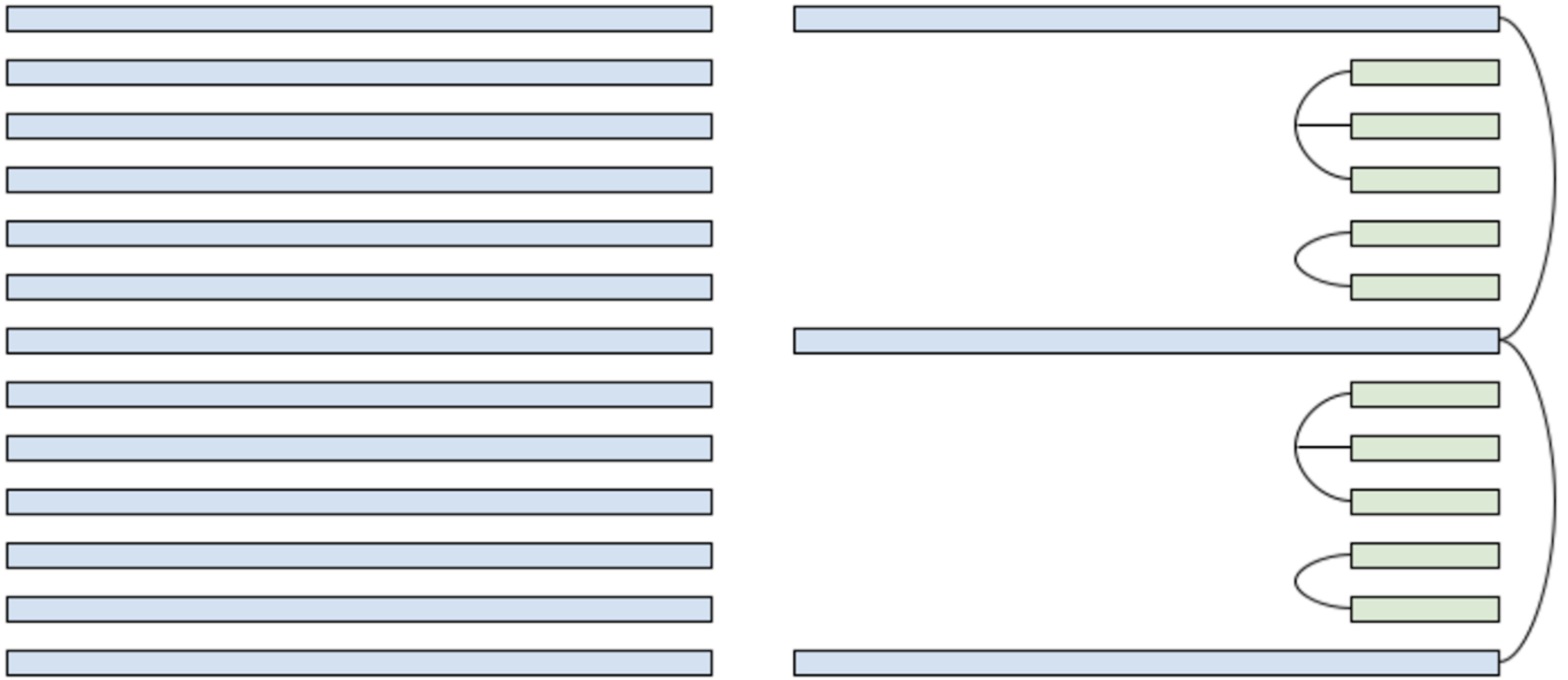
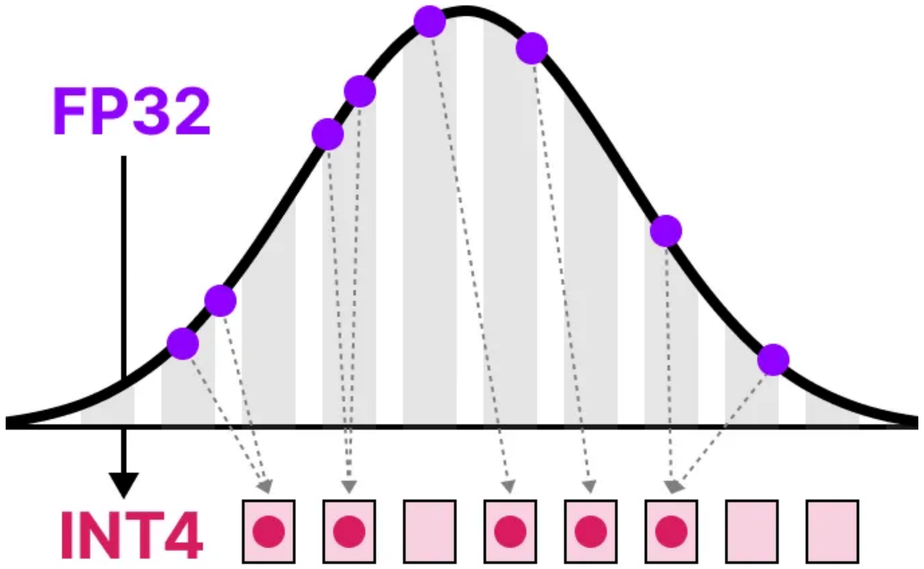


Figure 1. Left: Standard transformer design where every attention is global attention. Right: The attention design in our production model. Blue boxes indicate global attention, green boxes indicate local attention, and curves indicate KV-sharing. For global attention layers, we share KV across multiple non-adjacent layers. This illustration depicts only a subset of the layers in the full model.

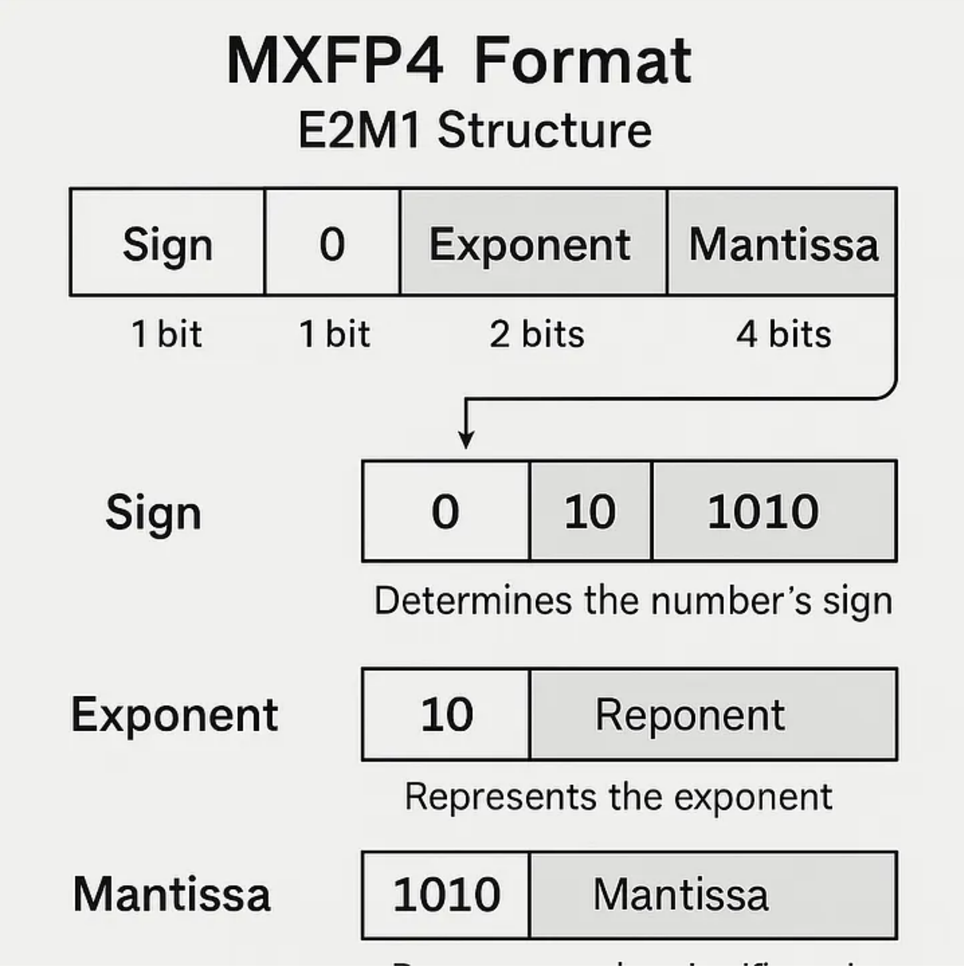
Character AI. https://blog.character.ai/optimizing-ai-inference-at-character-ai/

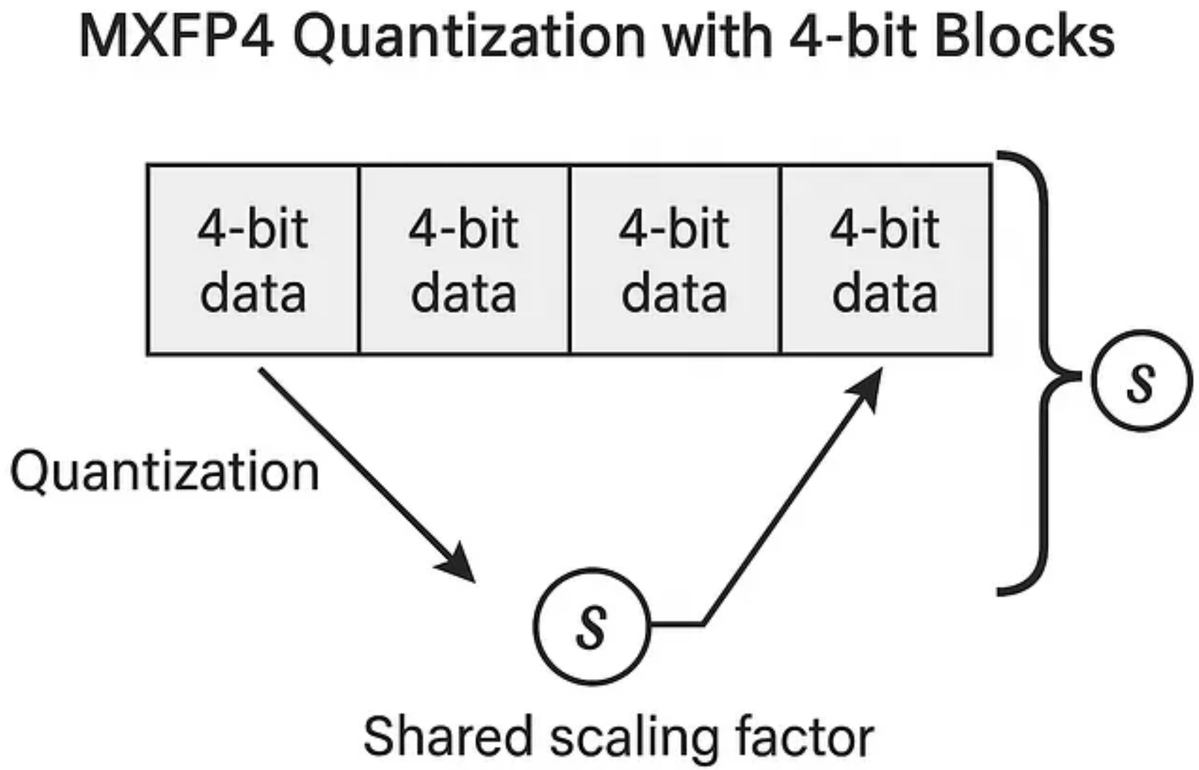
Inference: Weight & KV Cache Quantization



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

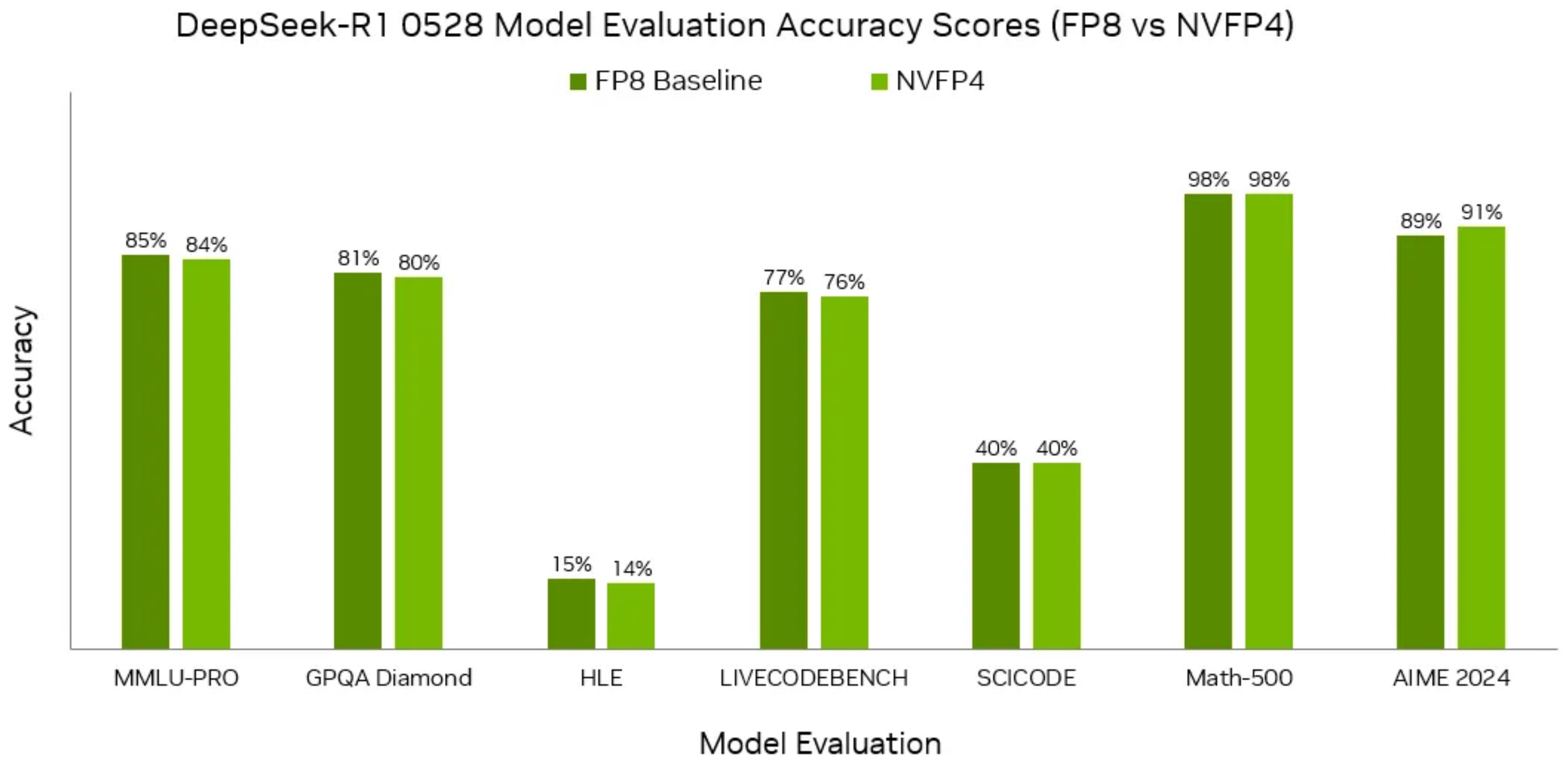
Numerical Formats: Microscaling Low Precision





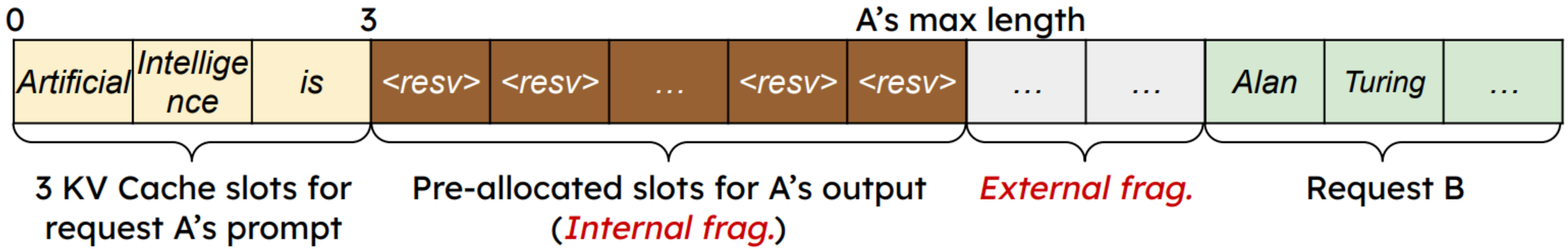
https://huggingface.co/blog/RakshitAralimatti/learn-ai-with-me

Numerical Formats: Low Precision and Quality



https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/

Inference: Challenge with KV cache memory management

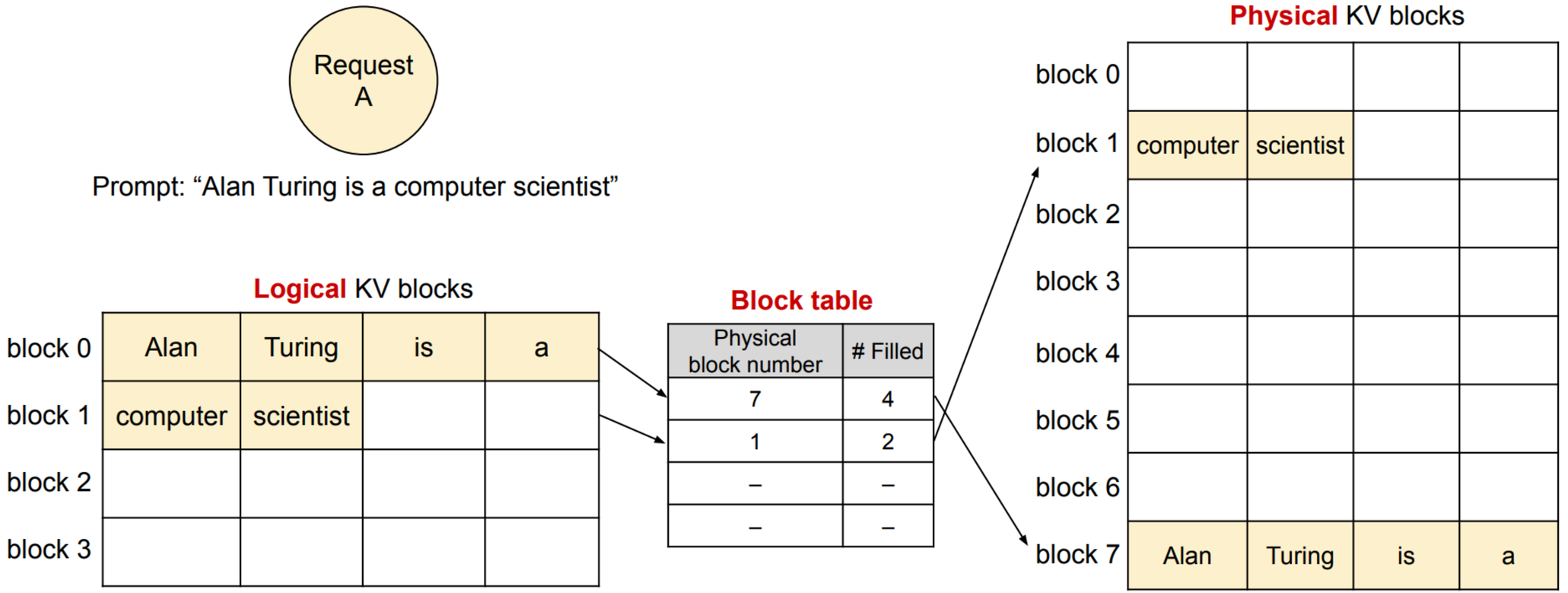


- **Pre-allocates contiguous** memory to the request's max length

- **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

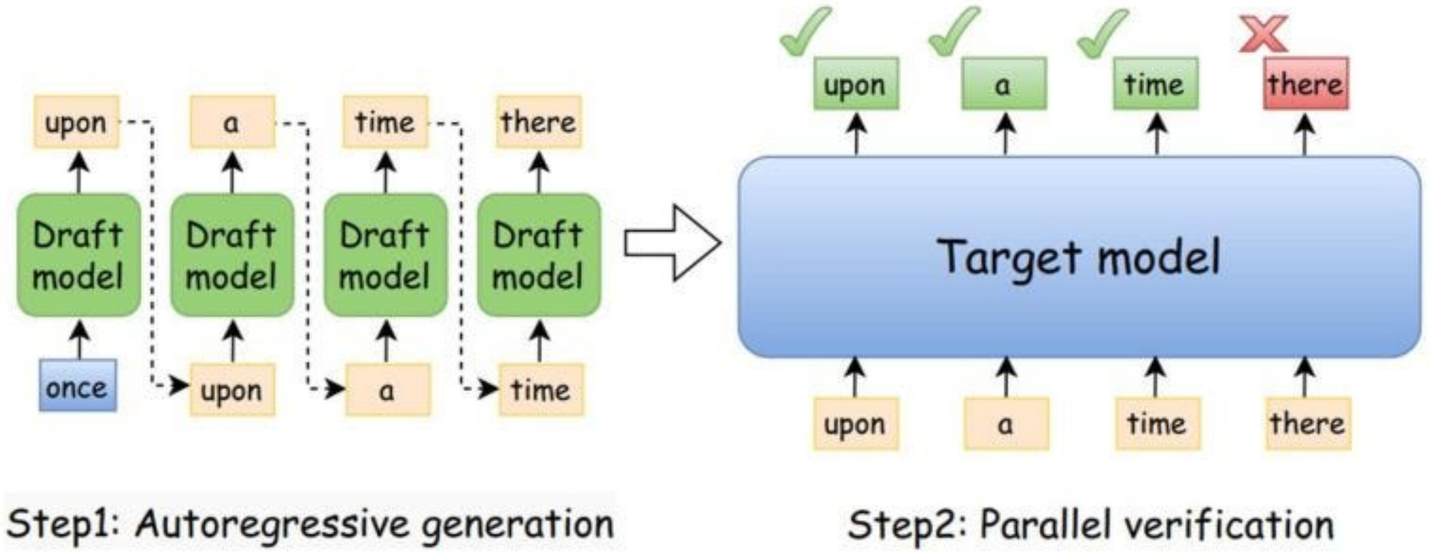


Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention

Inference: Speculative Decoding

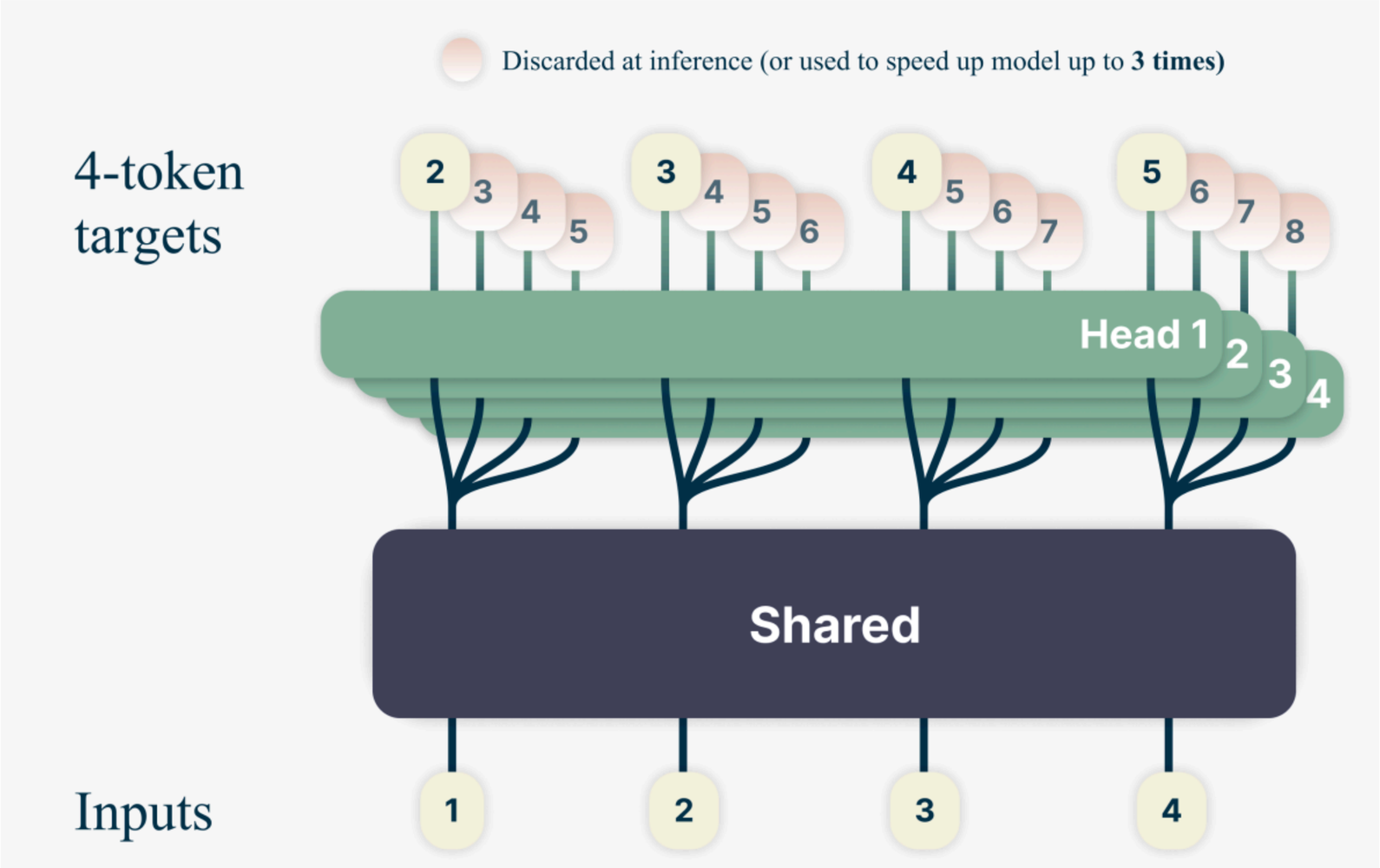
- Use a small Draft model to predict what the Target model will output

- Verify with Target model multiple tokens in parallel (≈ same cost as processing 1 token)



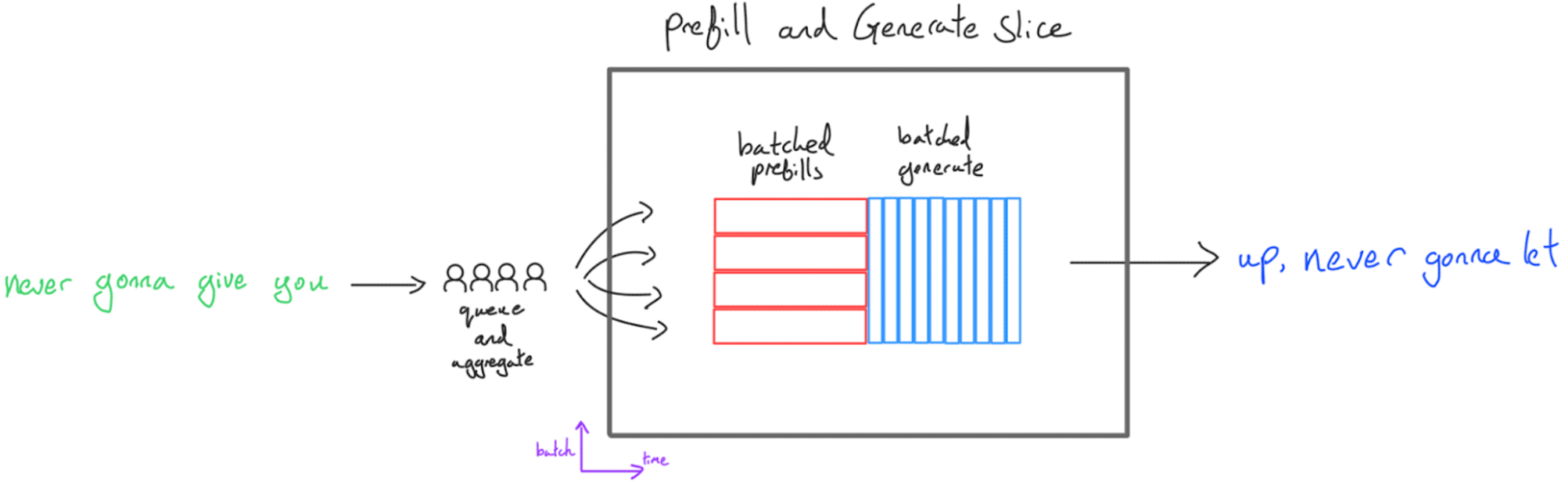
Leviathan et al. Fast Inference from Transformers via Speculative Decoding

Multi-token Prediction



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

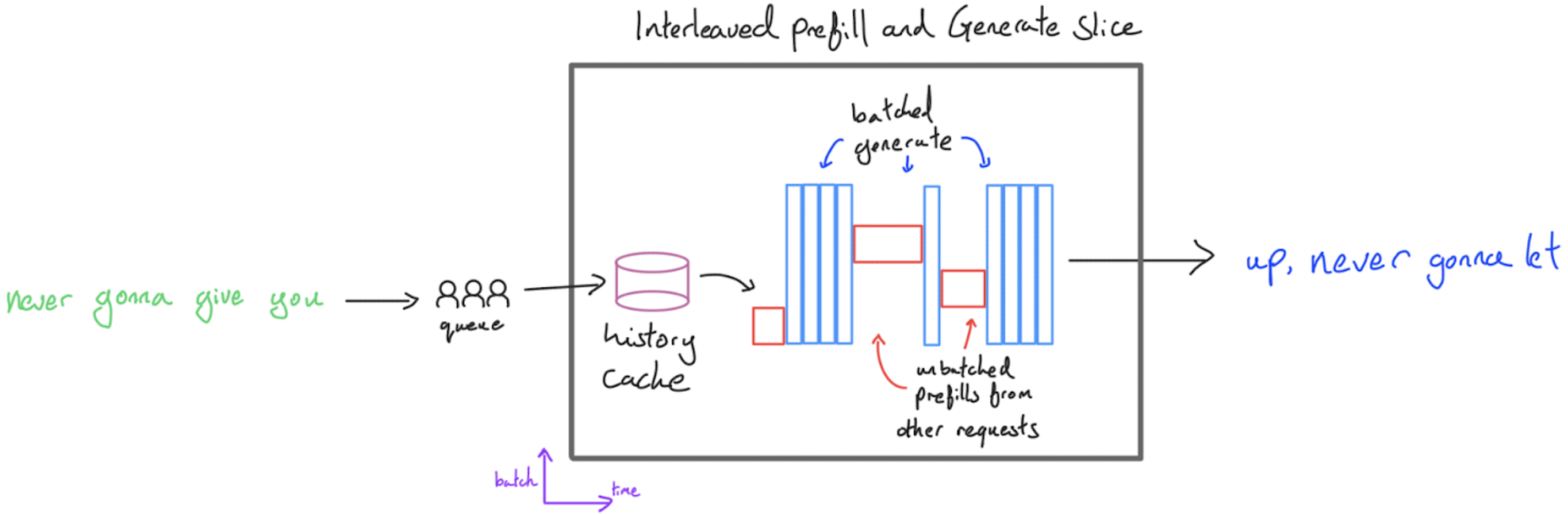
Inference: How to Schedule Requests



Naive batching: prefill and generate

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

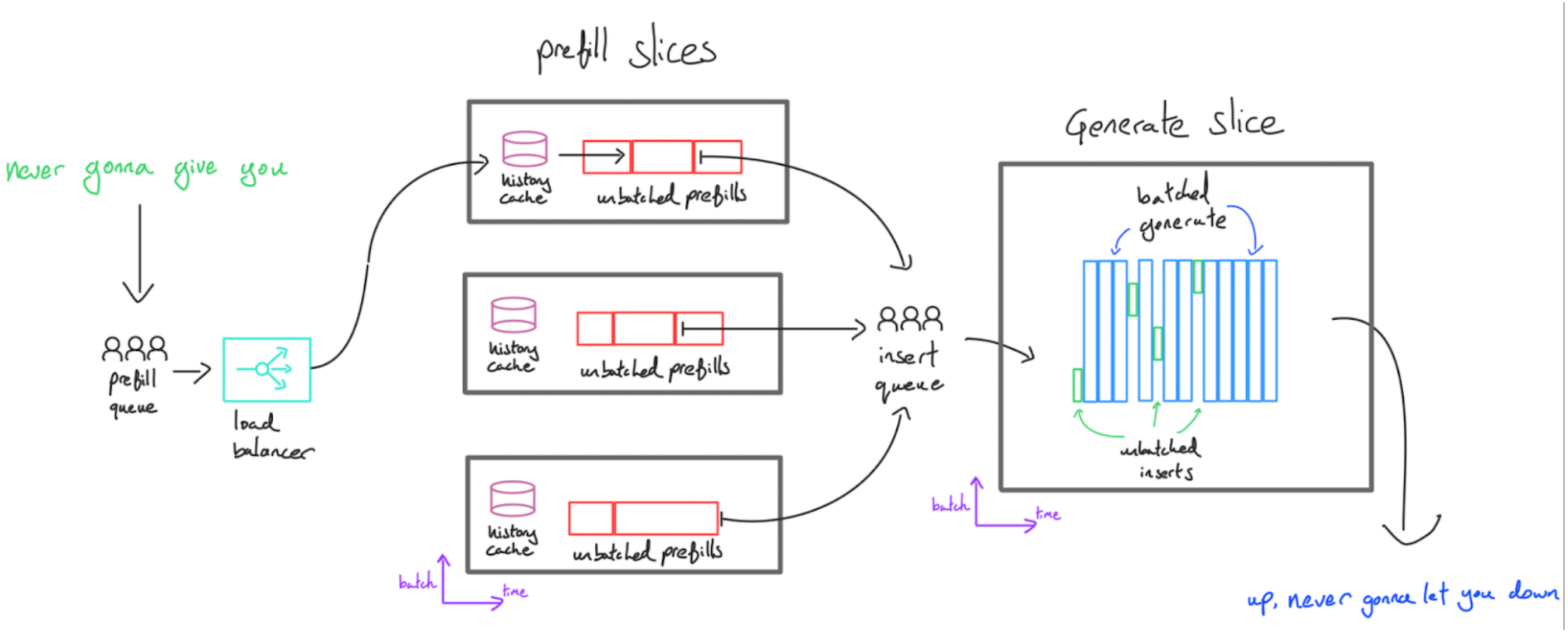
Inference: Interleaved batching



Prefill at batch=1 and generate at larger batch

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

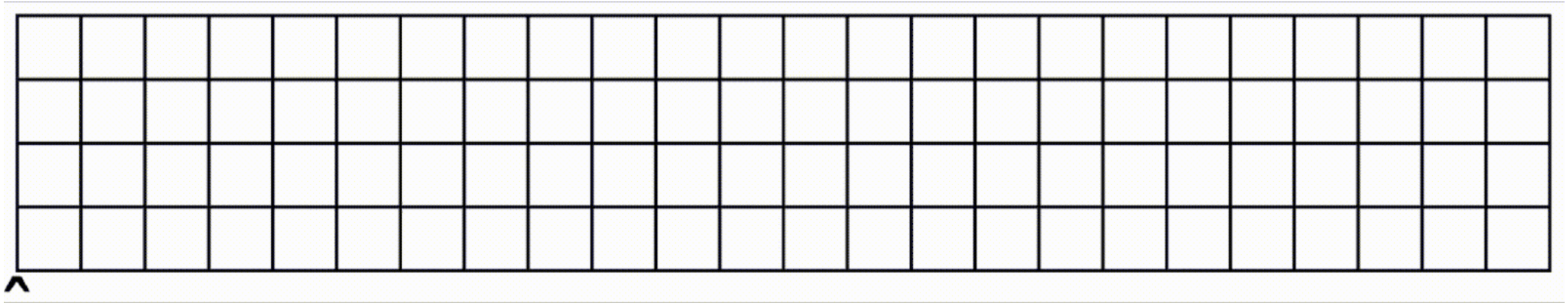
Disaggregated Prefill & Generate (Decode)



Different chips to do prefill and generate

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

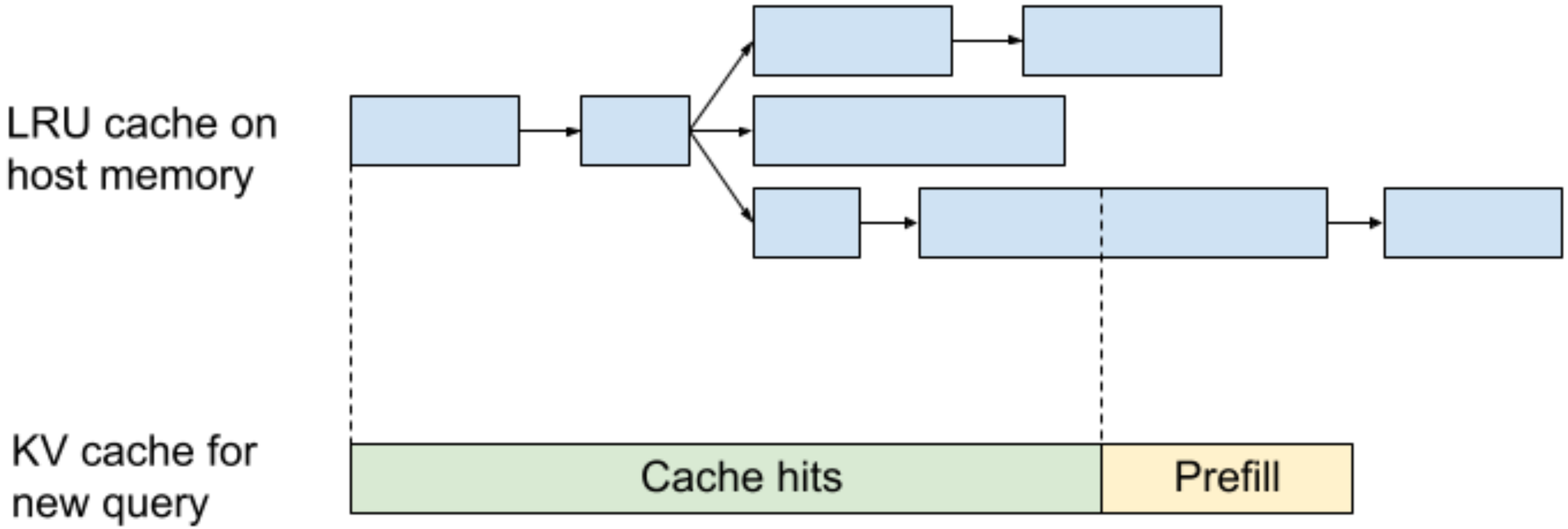
Continuous batching



Prefill: variable context lengths and inserts results into a KV buffer
Generate: takes in the KV cache, and performs the generation step for all currently active requests.

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

Prefix caching



Critical for multi-turn applications (chatbot, coding)

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

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