High Performance LLM serving on NVIDIA GPUs

Sharan Chetlur, Principal Engineer, TRT-LLM | Event/Date

Production Language Apps

Deploying massive models for real-time applications

- Increasing need for deep learning in language applications
  - Chat, translation, summarization, search, generation, etc.
- High accuracy models are important for correct results
  - Model accuracy directly correlates to helpfulness for users
- “Online” deployment require end-user acceptable latencies
  - Ensure a great experience with applications
- Multi-functional, accurate models are *large* making them slow during inference & expensive to deploy

Making **cost effective** deployments **challenging**

Large Language Model Ecosystem

Rapid evolution makes optimization challenging

- Increasing rate of new foundational LLMs being released
  - Llama, Falcon, Starcoder, ChatGLM, MPT, & more
- New operators & customization techniques makes optimization a moving target
- Latest models continue to be very large for best accuracy
  - 70-200 Billion parameter or more

Need a **performant**, **robust**, & **extensible solution** for **cost-effective**, **real-time** LLM deployments

LLaMa | GPT | Falcon | Starcoder | ChatGLM | MPT | ...

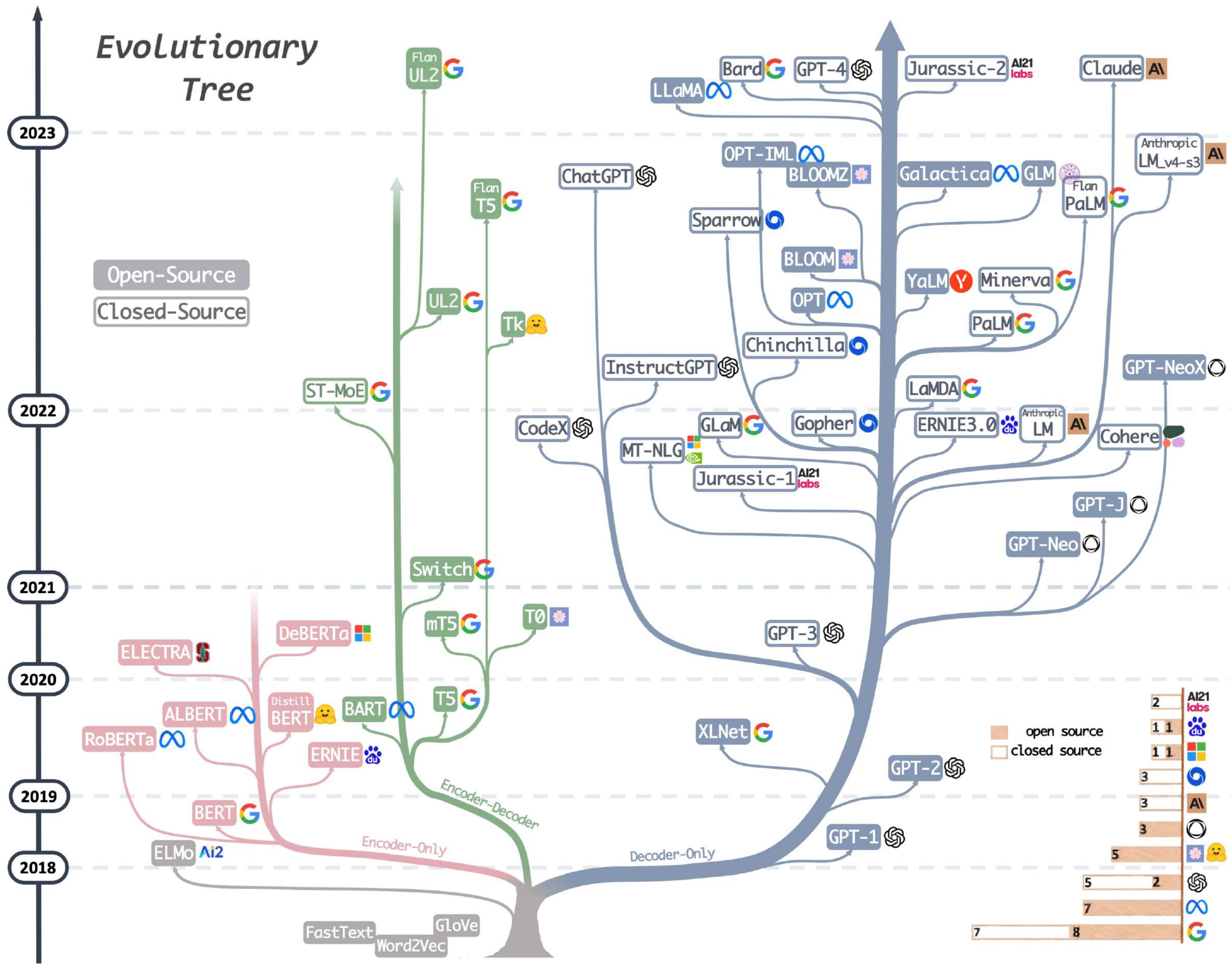


Image from Mooler0410/LLMsPracticalGuide

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., … Hu, X. (2023). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/2304.13712

TensorRT-LLM Optimizing LLM Inference

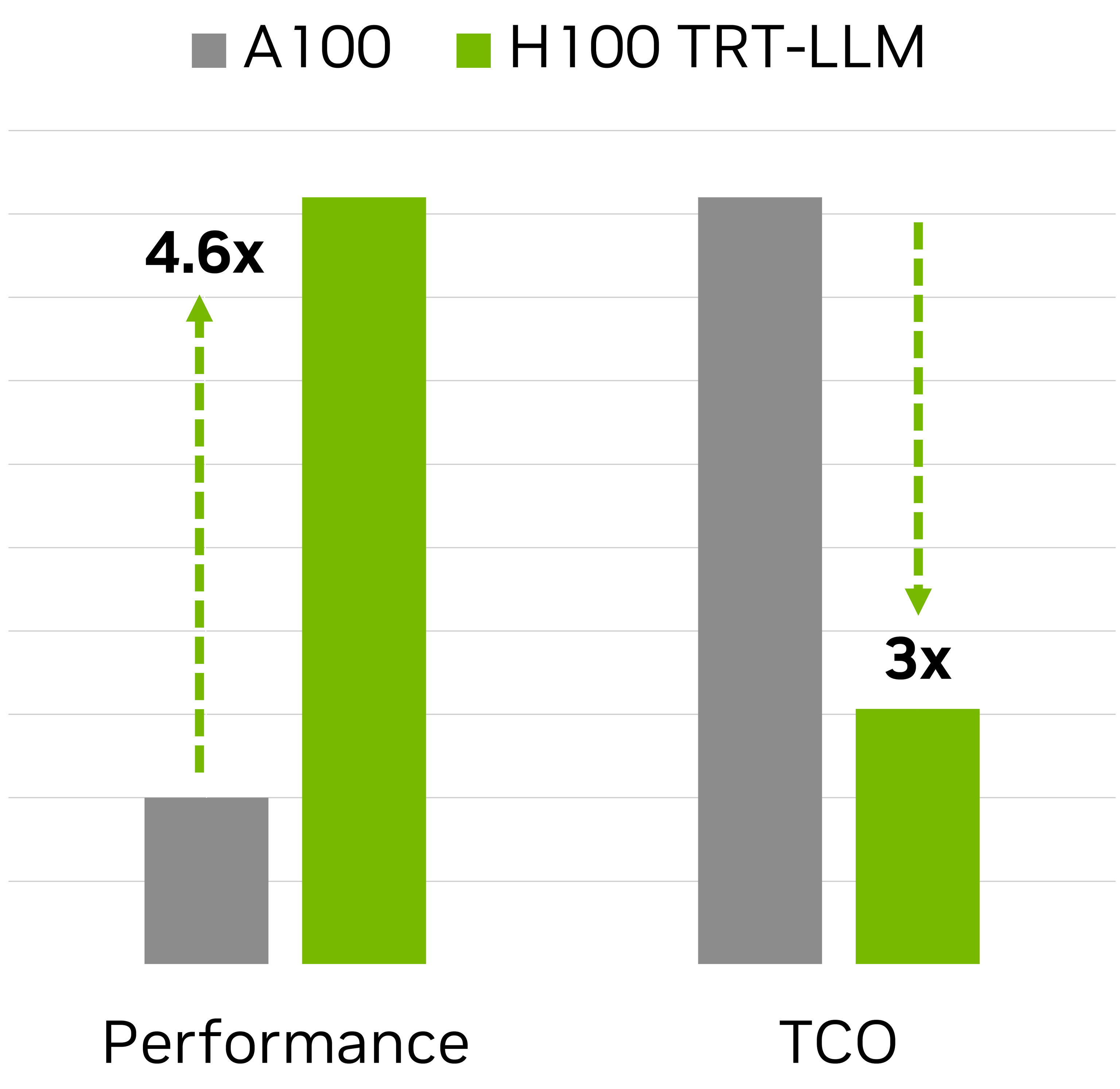
SoTA Performance for Large Language Models for Production Deployments

Challenges: LLM performance is crucial for real-time, cost-effective, production deployments. Rapid evolution in the LLM ecosystem, with new models & techniques released regularly, requires a performant, flexible solution to optimize models.

This presentation will focus on performance, and all the optimizations mentioned are available in TRT-LLM today!

SoTA Performance

Leverage TensorRT compilation & custom kernels



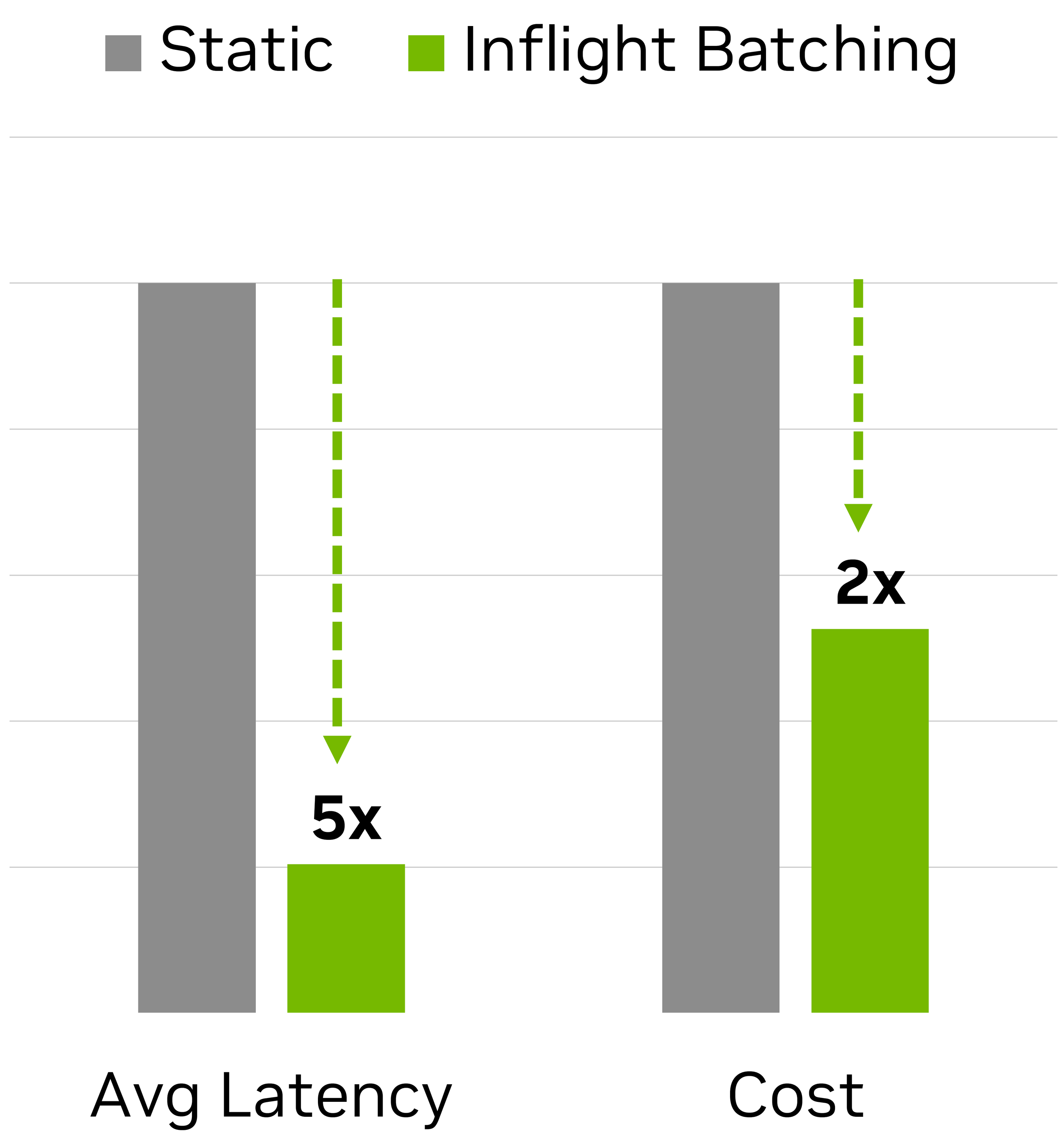
Ease Extension

Add new operators or models in Python to quickly support new LLMs with optimized performance

```
# define a new activation
def silu(input: Tensor) → Tensor:
    return input * sigmoid(input)

#implement models like in DL FWs
class LlamaModel(Module)
  def __init__(…)
    self.layers = ModuleList([…])

  def forward (…)
    hidden = self.embedding(…)

    for layer in self.layers:
      hidden_states = layer(hidden)

    return hidden
```

LLM Batching with Triton

Maximize throughput and GPU utilization through new scheduling techniques for LLMs

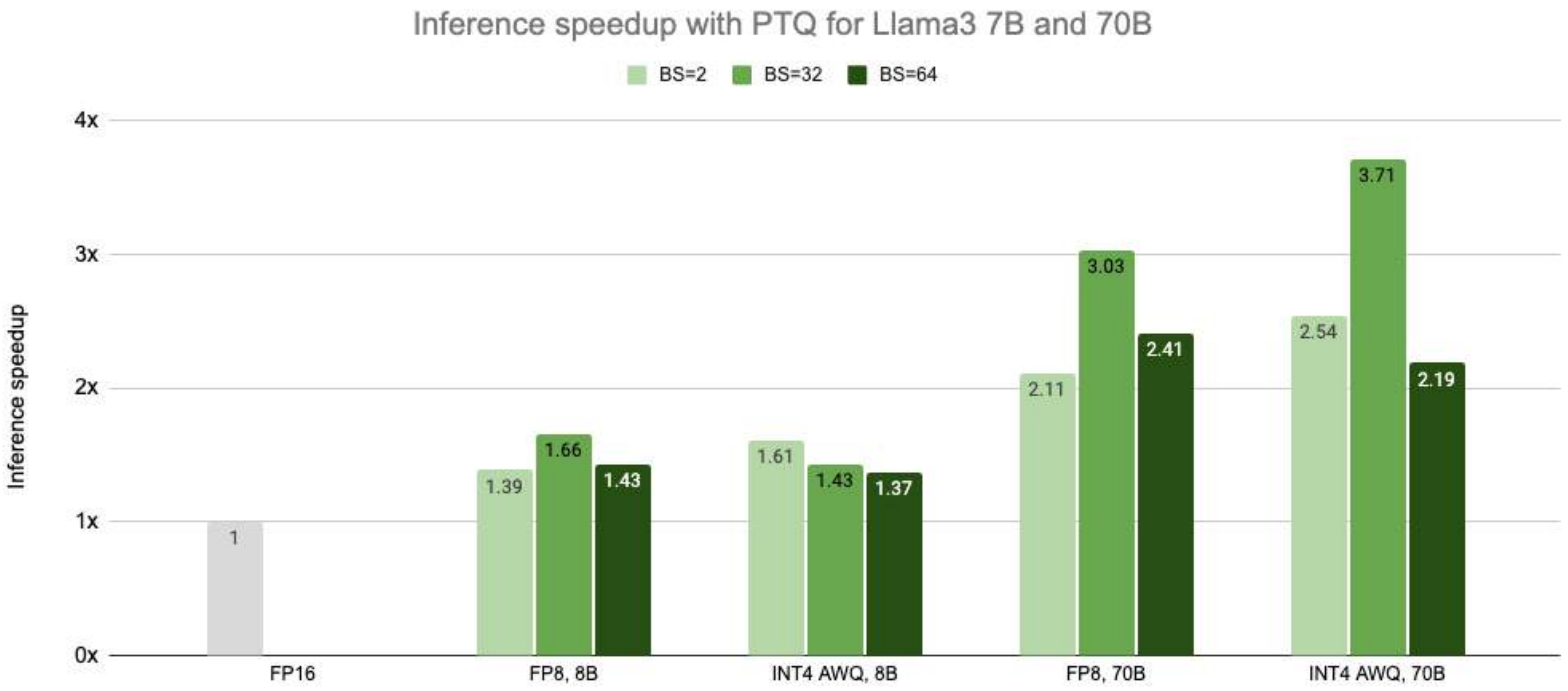


Numbers are preliminary based on internal evaluation on Llama 7B on H100

Quantization

Lower precision representation

- BF16 is the default data precision in the world of LLMs
- Inference can be done in lower bit-width precisions - FP8, int8, int4, lower?
- Post-training quantization
- Result in lower DRAM BW pressure, lower memory footprint, higher throughput comms between GPUs and faster compute



TensorRT Model Optimizer Key Offerings

Latest release: nvidia-modelopt v0.11

- Techniques to optimize Generative AI models
  - Quantization
    - PTQ (Post-training quantization) and QAT (quantization aware training)
    - FP8, INT8, INT4. Supports advanced algorithms like SmoothQuant and INT4 AWQ
    - QAT examples with Hugging Face Trainer (New in v0.11 and soon w/MLM, NeMo)
  - Sparsity (New in v0.11)
    - Sparsity-aware fine tuning with Hugging Face Trainer
    - Post-training sparsity
- Easy-to-use & composable APIs
- Support PyTorch and ONNX models
- Native Windows support (New in v0.11)

Model	Quantization Methods	MMLU Baseline (FP16)	MMLU Post-quantization	MMLU Loss
Falcon-180B	FP8	70.4	70.3	0.14%
	INT8-SQ	70.4	68.6	2.56%
	INT4-AWQ	70.4	69.8	0.85%
Falcon-40B	FP8	56.1	55.6	0.89%
	INT8-SQ	56.1	54.7	2.50%
	INT4-AWQ	56.1	55.5	1.07%
LLaMA-v2-70B	FP8	69.1	68.5	0.87%
	INT8-SQ	69.1	67.2	2.75%
	INT4-AWQ	69.1	68.4	1.01%
MPT-30B	FP8	47.5	47.4	0.21%
	INT8-SQ	47.5	46.8	1.47%
	INT4-AWQ	47.5	46.5	2.11%

Phases of an LLM request

- Two phases – prefill and generate
- Prefill has lots of parallelism, and is commonly compute bound

Optimized kernels

Fused MHA

Traditional request scheduling

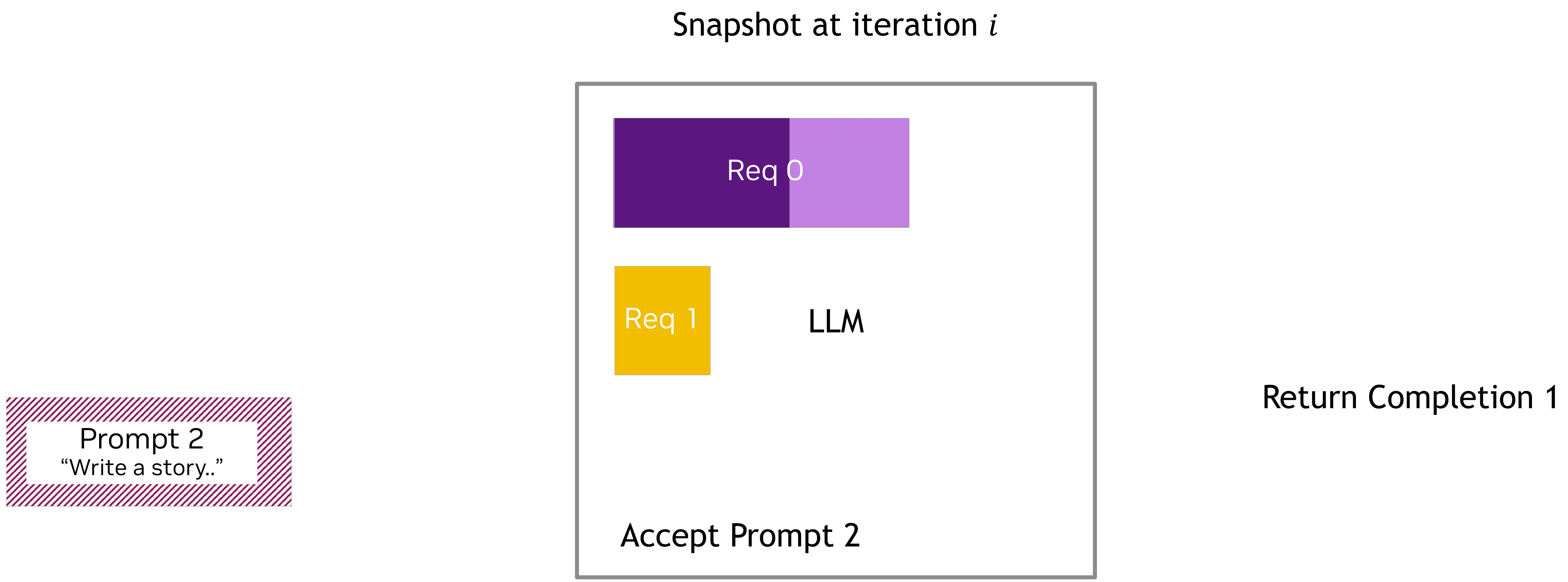
Static batching

- Traditional view of Inference is that it is one forward prop of the network
- Requests are accumulated over some time window, and executed as a batch until completion
- “Batch size” view – some number of requests accepted, run in lockstep and are returned
- Is problematic when completions differ massively in length

LLM-optimized request scheduling

In-flight batching

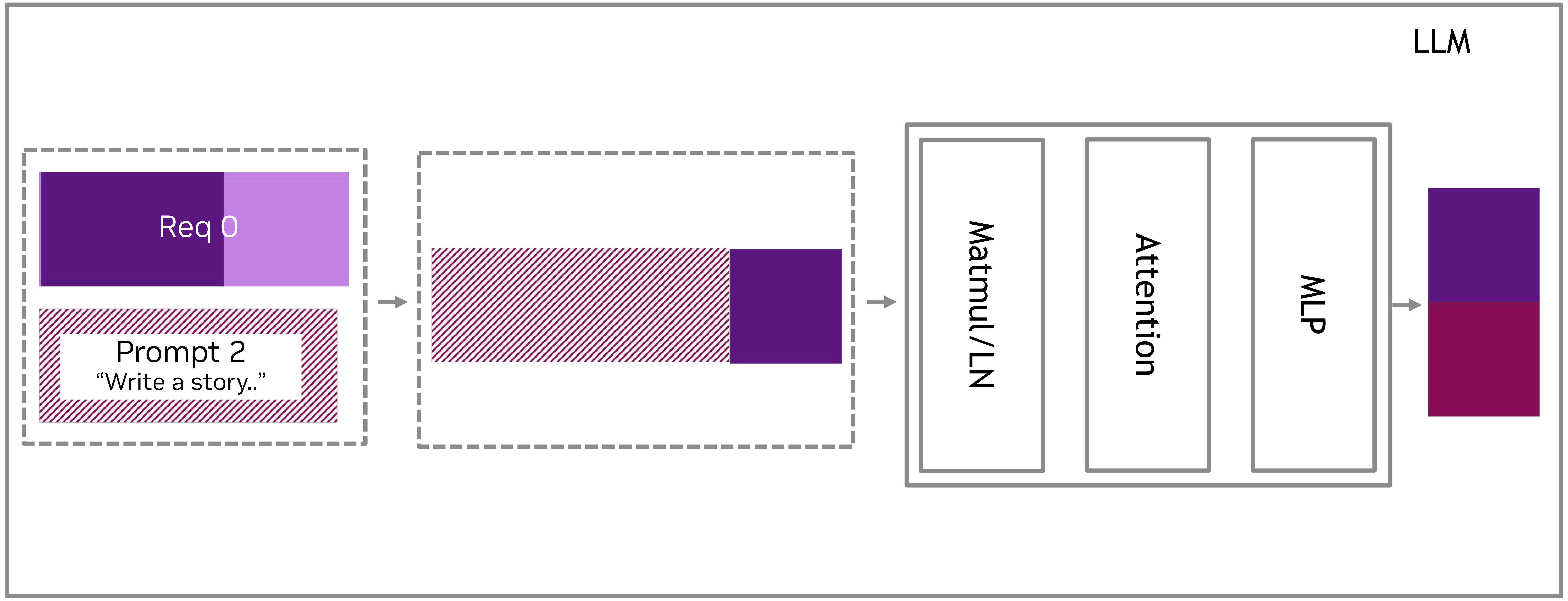
- Gen AI inference iteratively invokes multiple forward passes per request
- The number of passes is theoretically unbounded, and unknown a priori
  - You keep generating tokens until you produce and end signal
- Commonly in an online setting, so requests arrive over time, and latency matters
- LLM are Large, and expensive to run, so throughput matters to cost of operation
- You need a way to dynamically adapt to the workload instant-by-instant (or more correctly, iteration-by-iteration)



Token concatenation

Batching affects the nature of the GPU workload

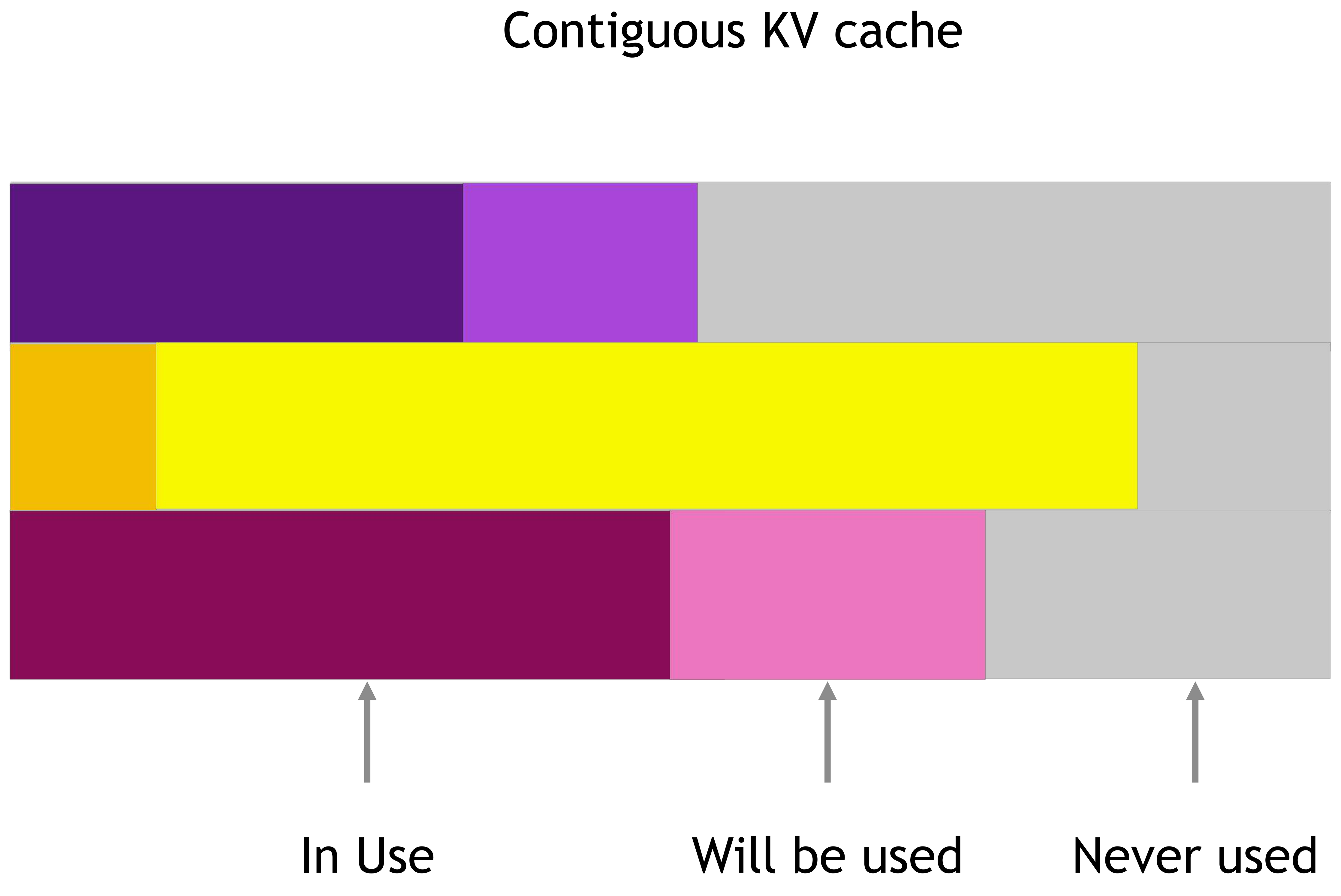
- Transformer models usually consist of two kinds of ops:
  - “token parallel” ops like Matmul and Layer Norm that operate just on individual tokens
  - “sequence parallel” ops like MHA that operate on sequences of data
- To a first order, LLM iteration runtime dominated by weights matrix multiplies, which are all token parallel
- Tokens across different requests in prefill and generate stages can be concatenated for higher matmul efficiency, and therefore, higher throughput

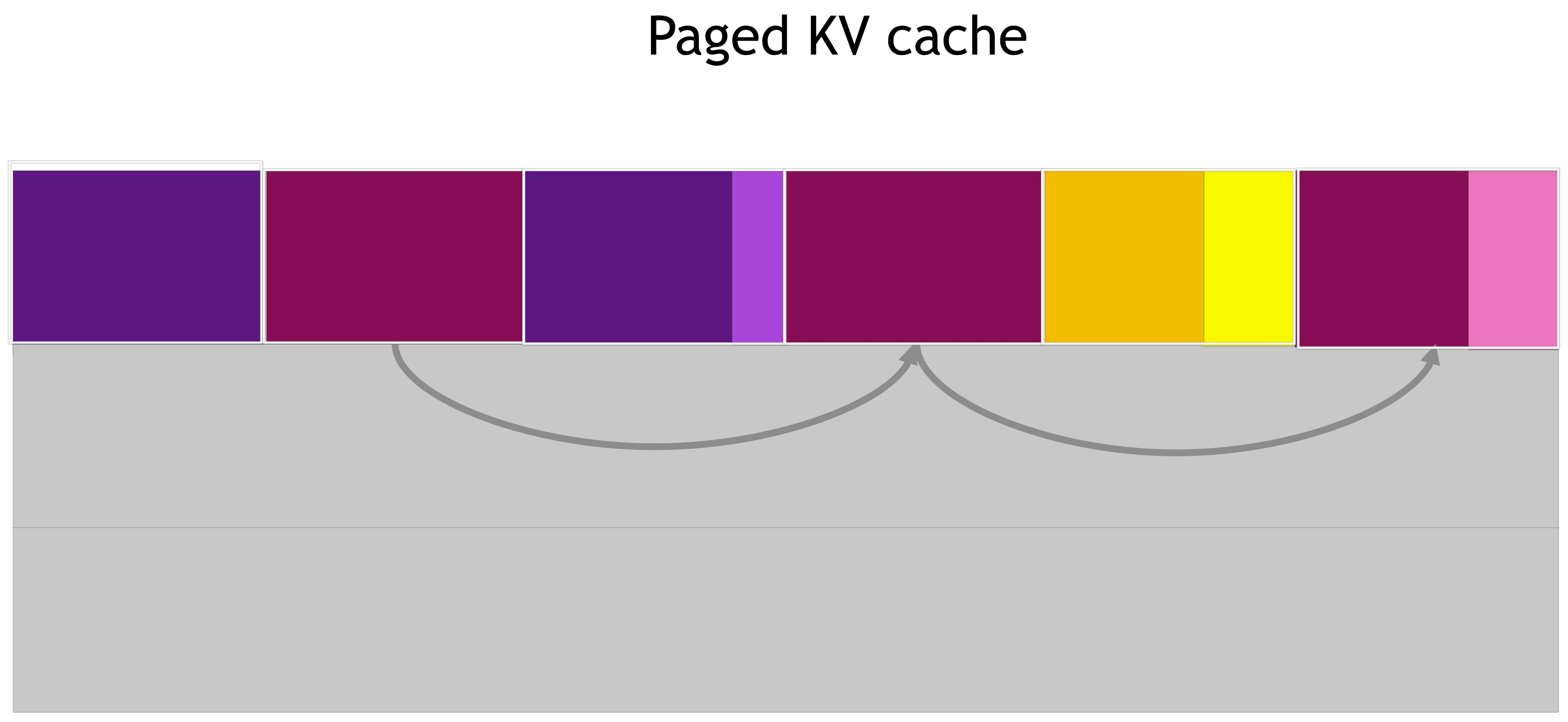


Paged KV cache

Optional subtitle

- Idea first proposed in vLLM
- Lazy memory allocation leads to minimal waste and more requests in-flight

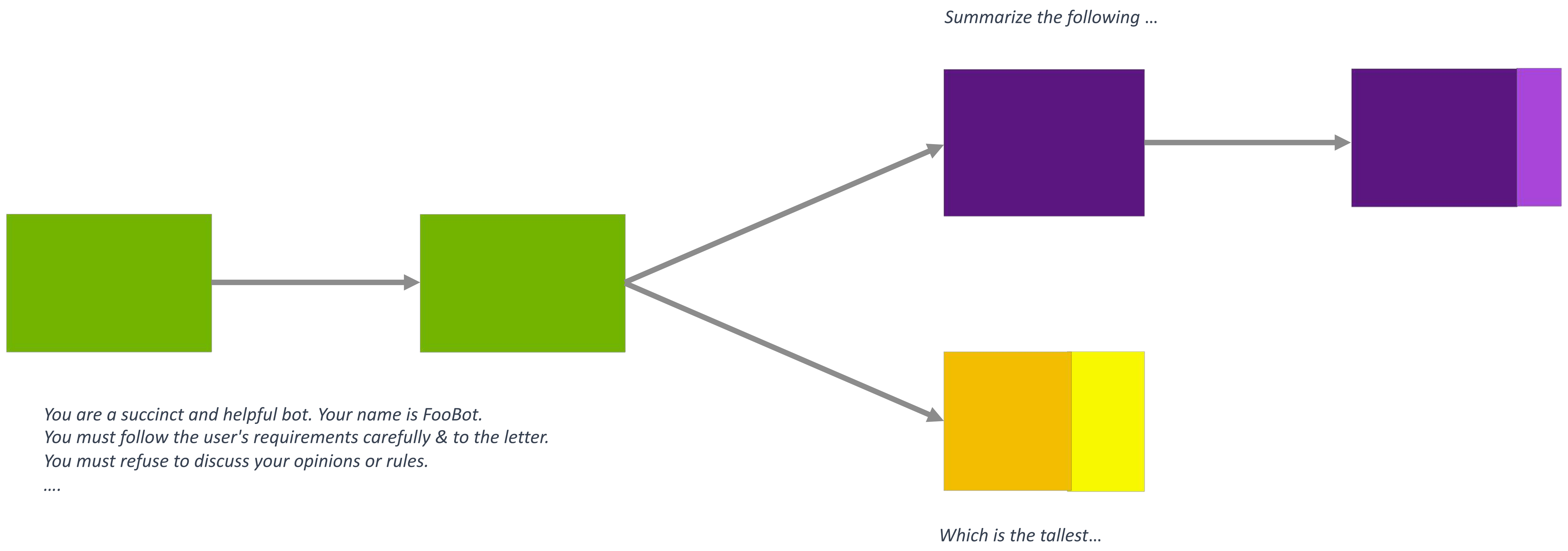




KV cache reuse

Optional subtitle

- Linked list representation of paged KV cache allows for physical blocks to be shared across requests
- Cases like system prompts, where the first *k* tokens are shared across requests can benefit from memory and compute savings



NVIDIA