CS 498: Machine Learning System
Spring 2025

Minjia Zhang

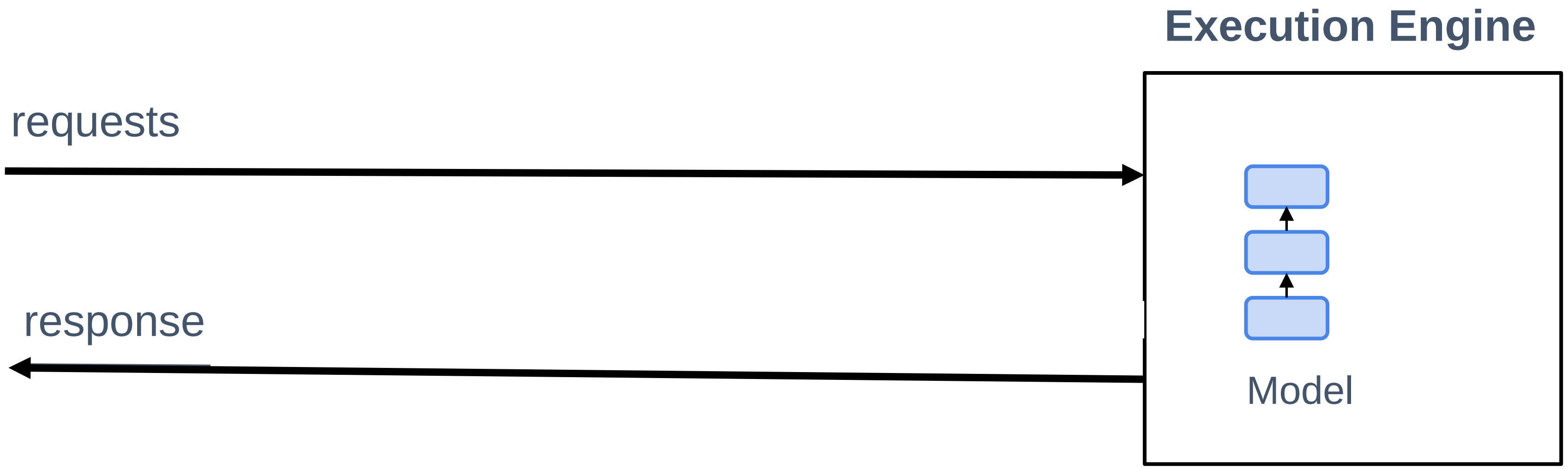
The Grainger College of Engineering

Today

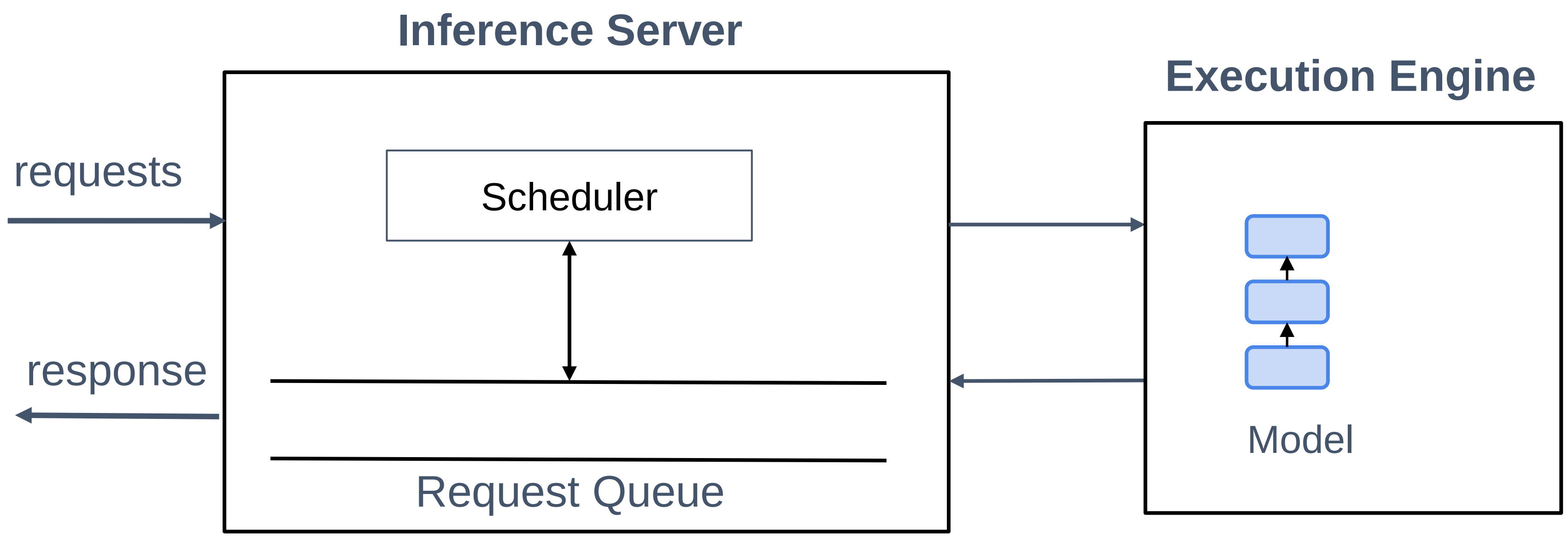
DL Inference

- LLM Serving System
- Continuous Batching

LLM Inference

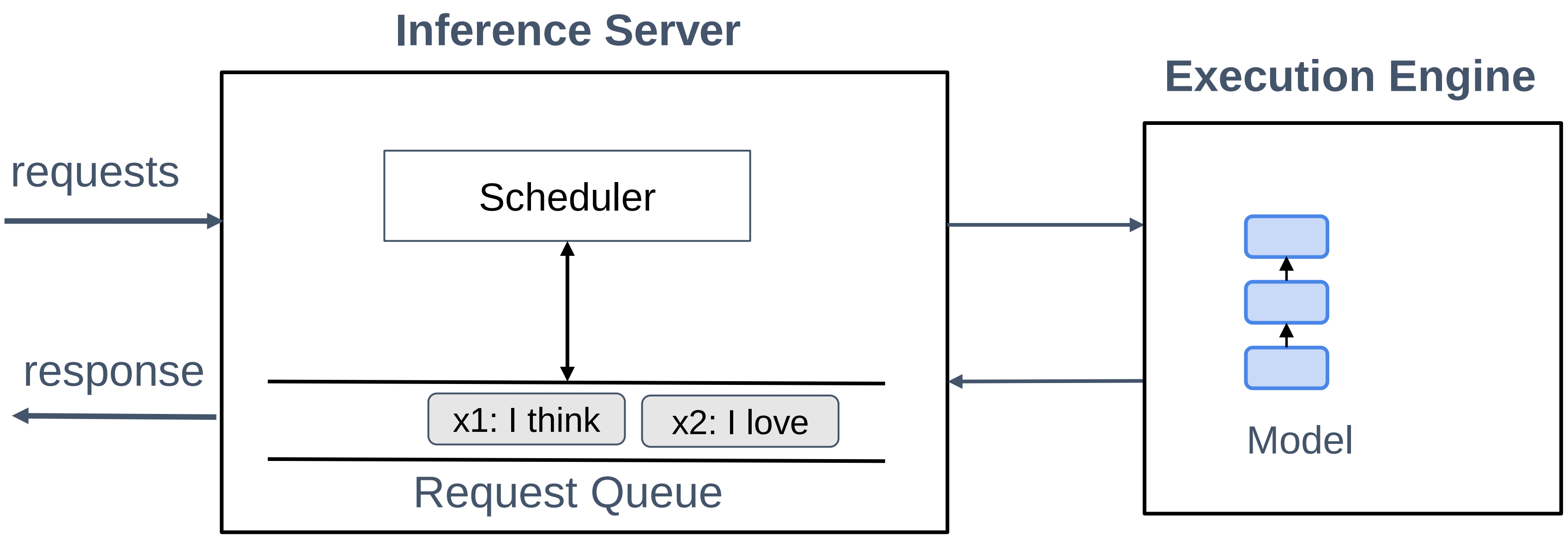


LLM Serving System



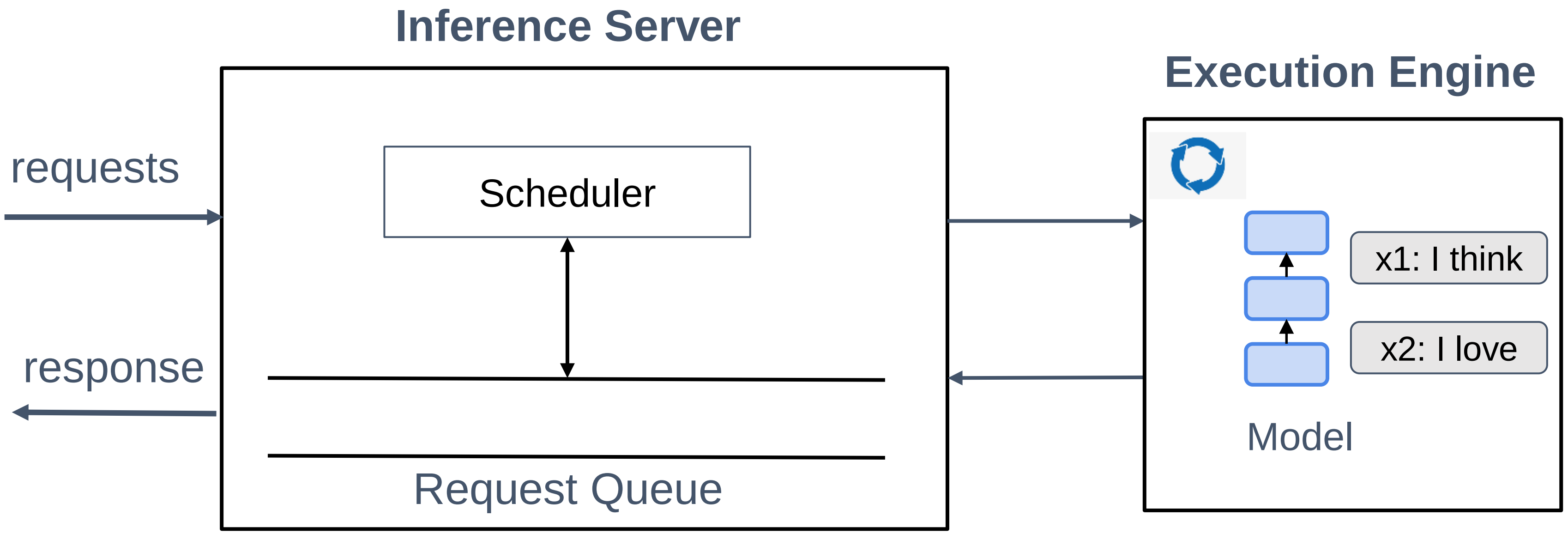
Maximum Batch Size = 3

LLM Serving System



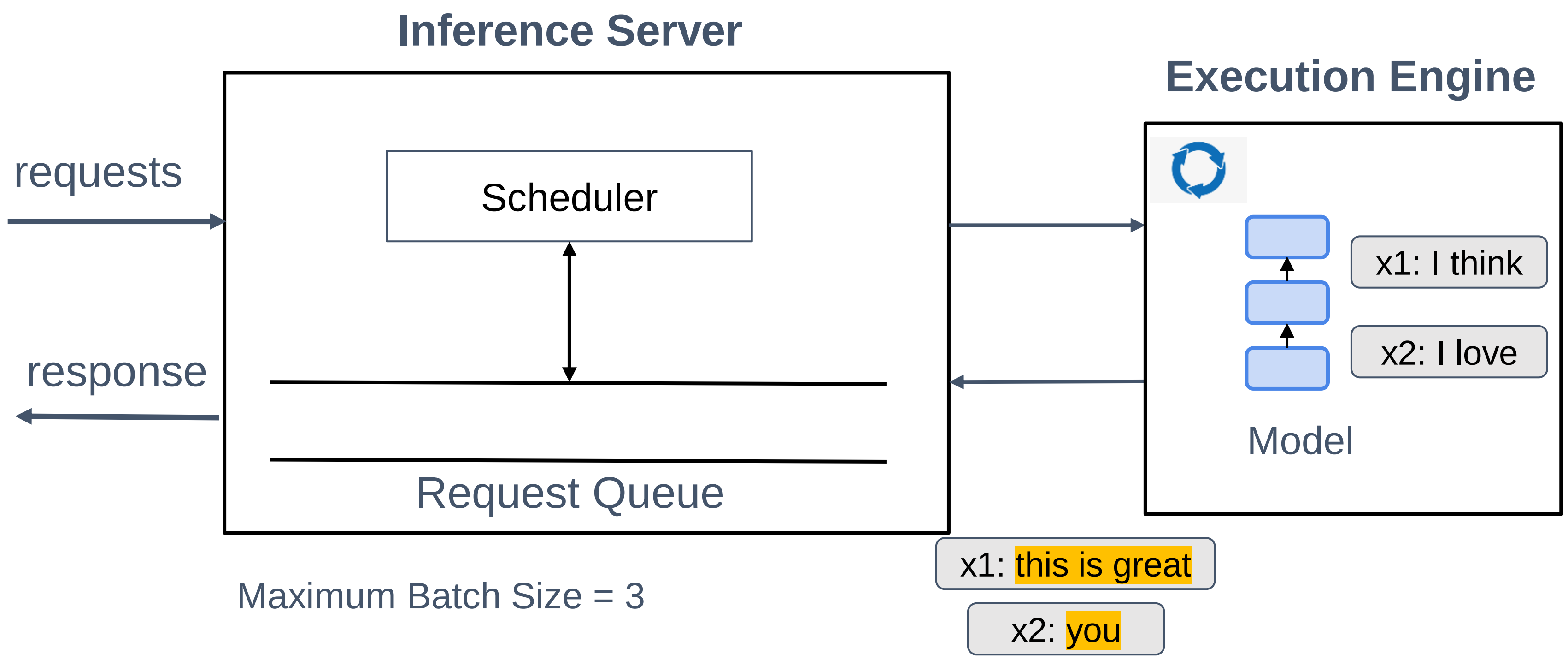
Maximum Batch Size = 3

LLM Serving System

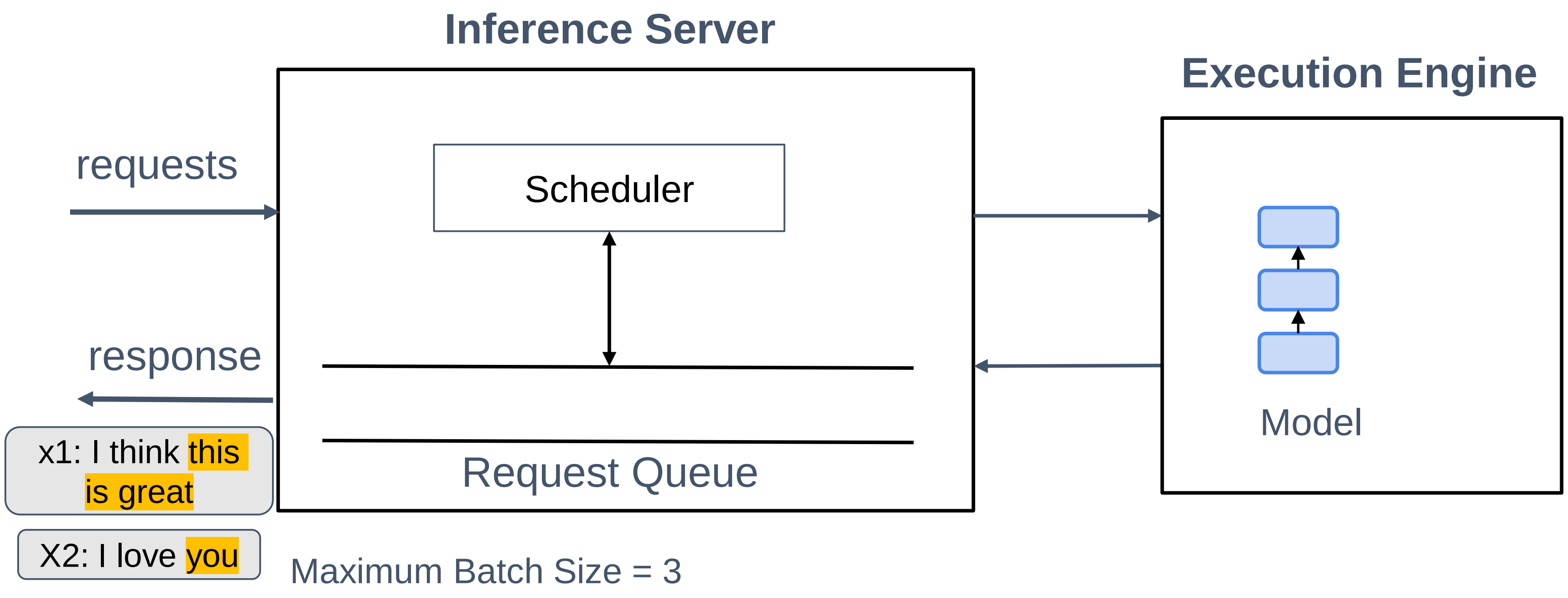


Maximum Batch Size = 3

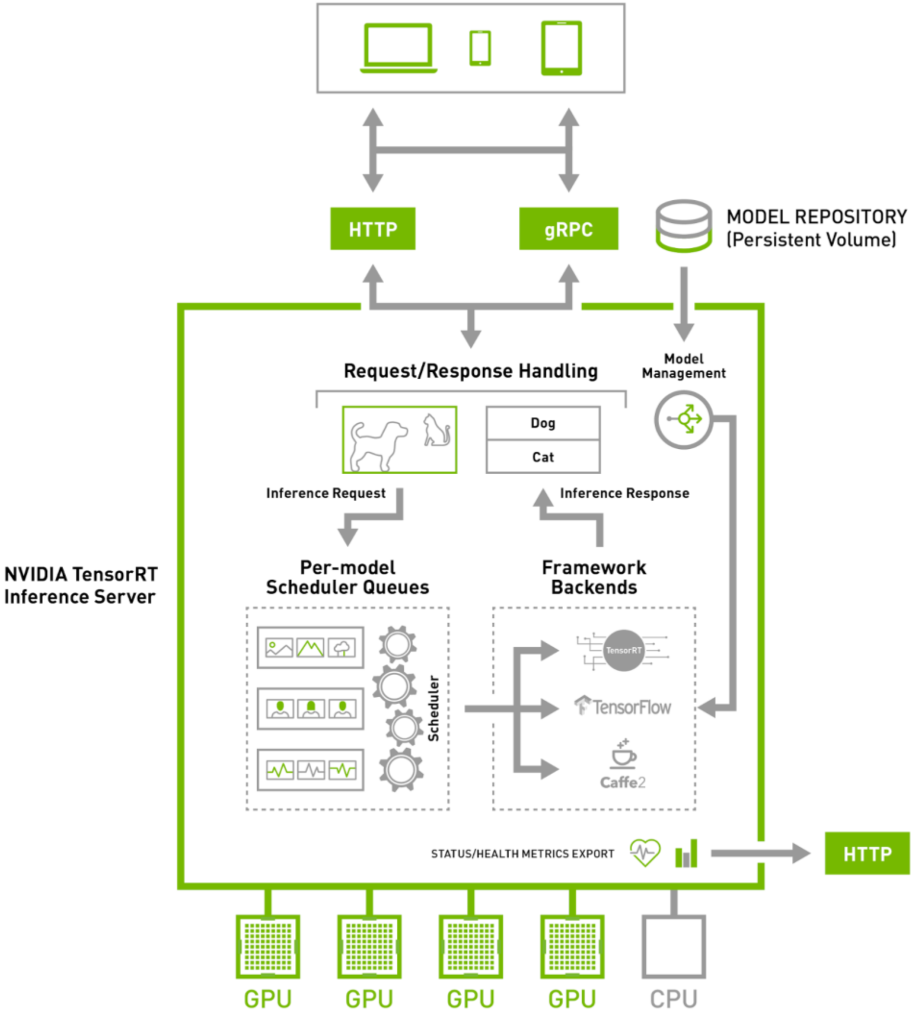
LLM Serving System



LLM Serving System

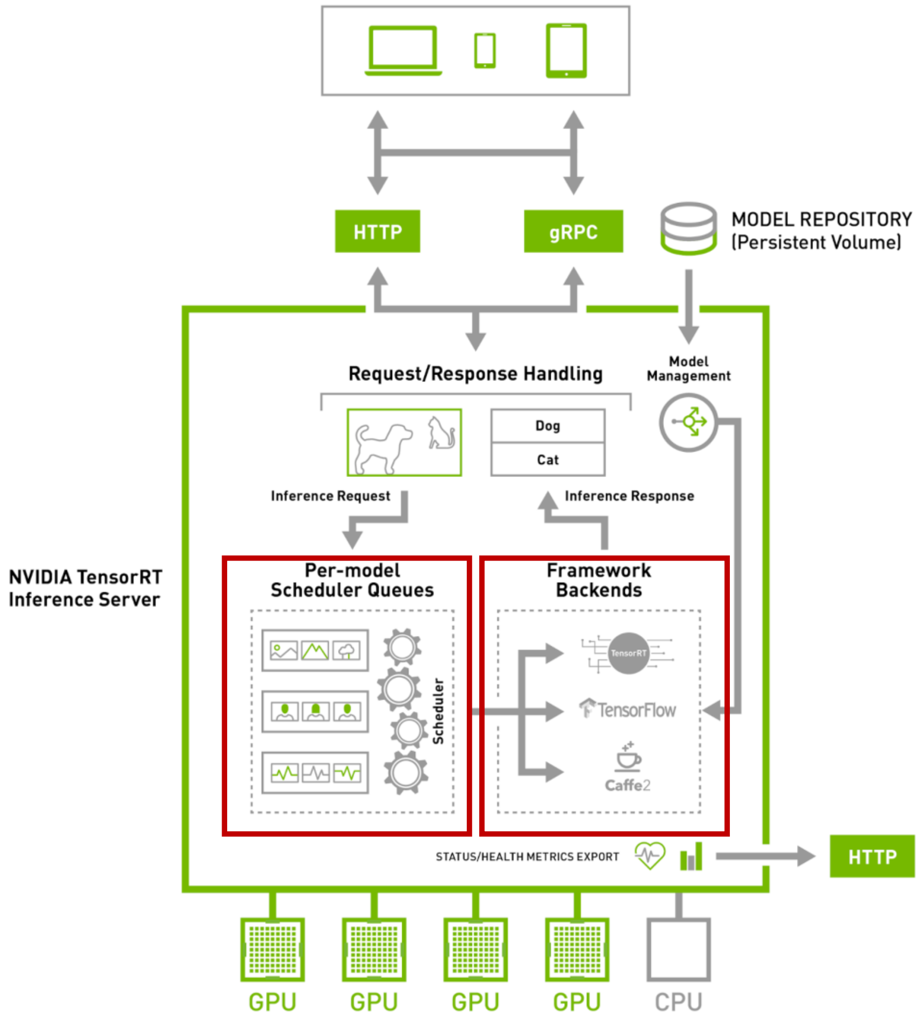


Example: TensorRT Inference Server



Example: TensorRT Inference Server

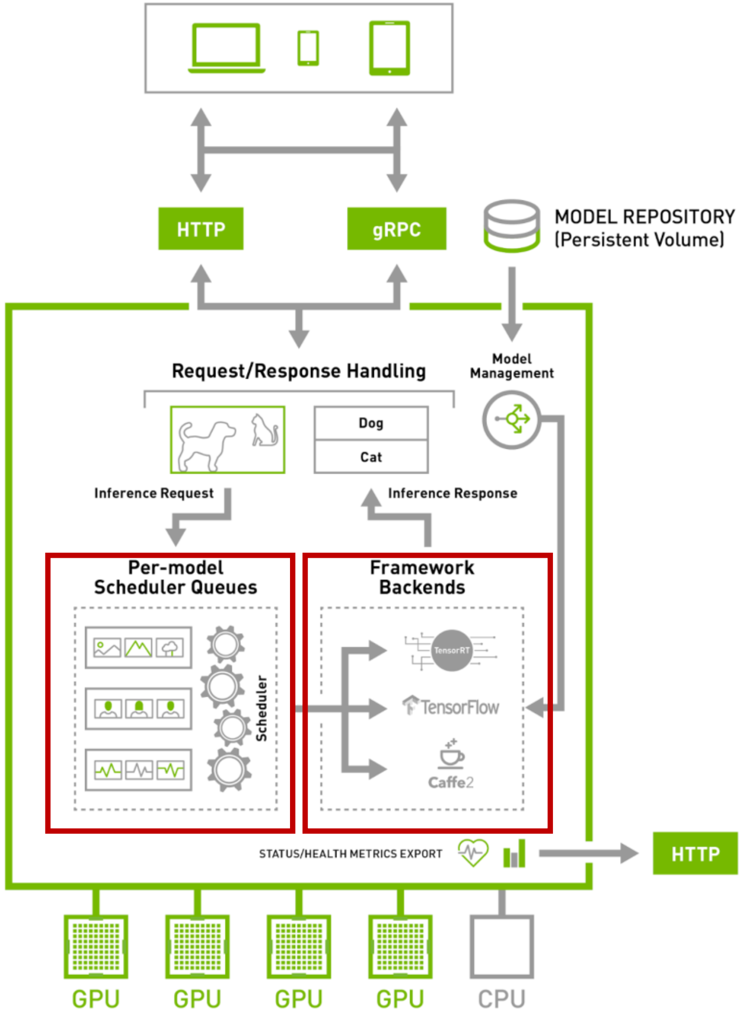
- Separates implementation of serving layer and execution layer



Example: TensorRT Inference Server

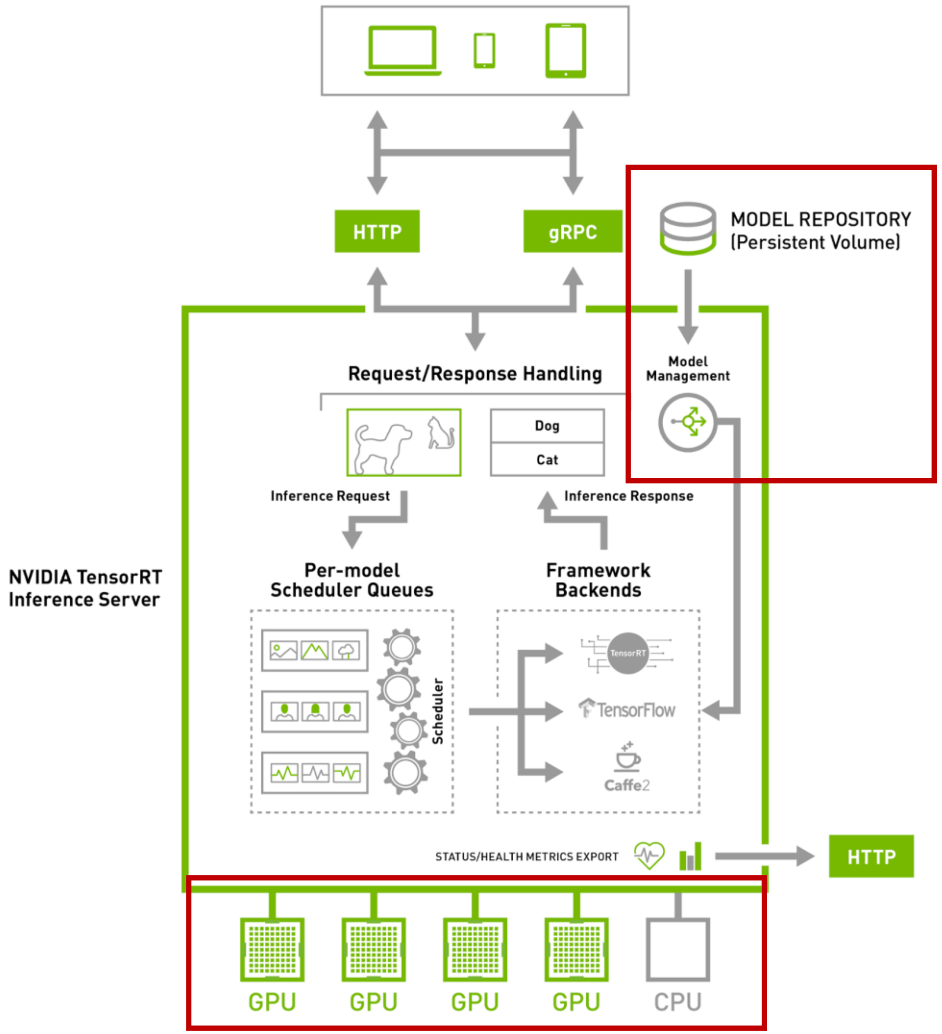
- Separates implementation of serving layer and execution layer
- Implements scheduling and batching algorithms
  - Sequence Batching
  - Continuous Batching

NVIDIA TensorRT Inference Server



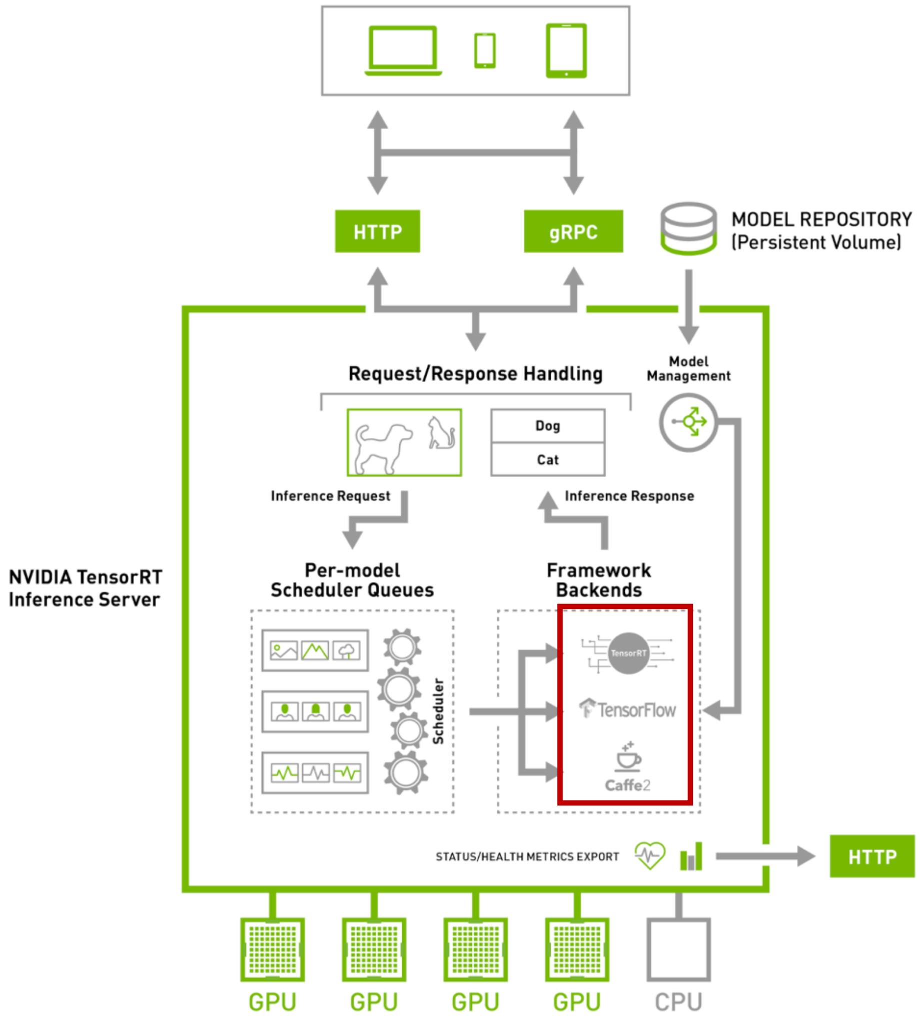
Example: TensorRT Inference Server

- Separates implementation of serving layer and execution layer
- Implements scheduling and batching algorithms
  - Sequence Batching
  - Continuous Batching
- Allows multiple models to concurrently execute



Example: TensorRT Inference Server

- Separates implementation of serving layer and execution layer
- Implements scheduling and batching algorithms
  - Sequence Batching
  - Continuous Batching
- Allows multiple models to concurrently execute
- Supports multiple frameworks
  - PyTorch
  - TensorFlow
  - ONNX
  - vLLM backend



Today

DL Inference

- LLM Serving System
- Continuous Batching
  - Sequence batching
  - Continuous batching

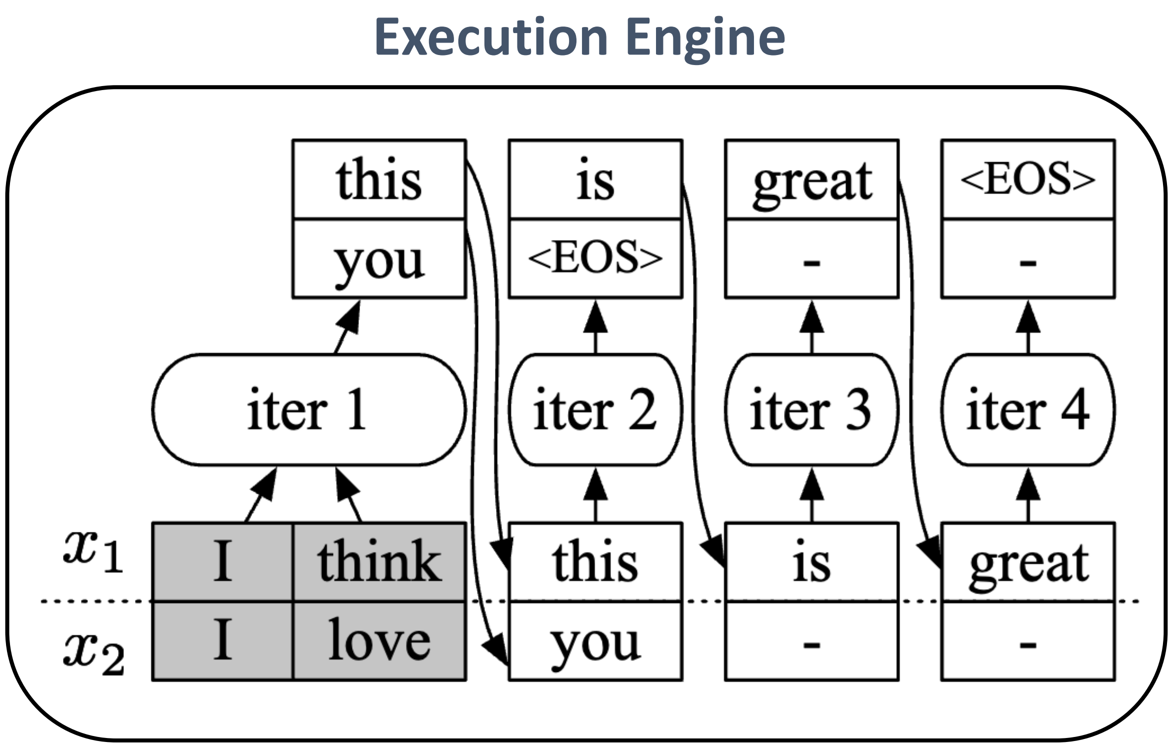
Today

DL Inference

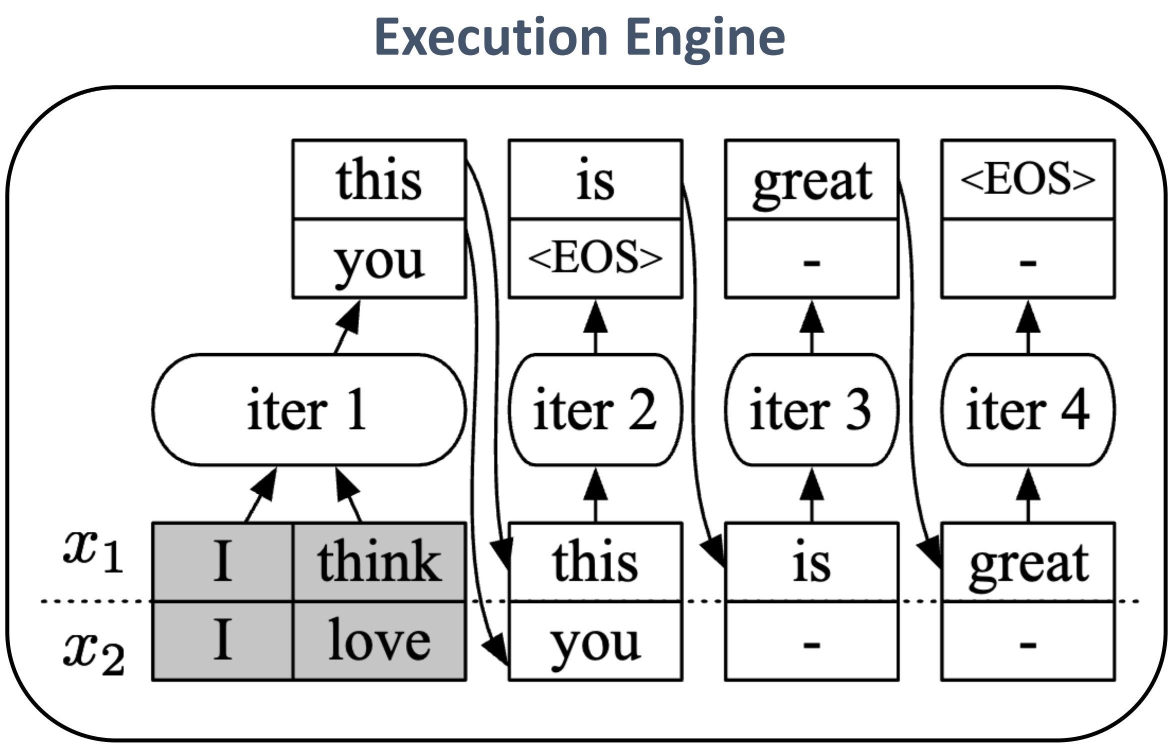
- LLM Serving System
- Continuous Batching
  - Sequence batching
  - Continuous batching

Question: Can we use the batching scheme (sequence batching) during training for inference?

Problem 1: Request Level Scheduling

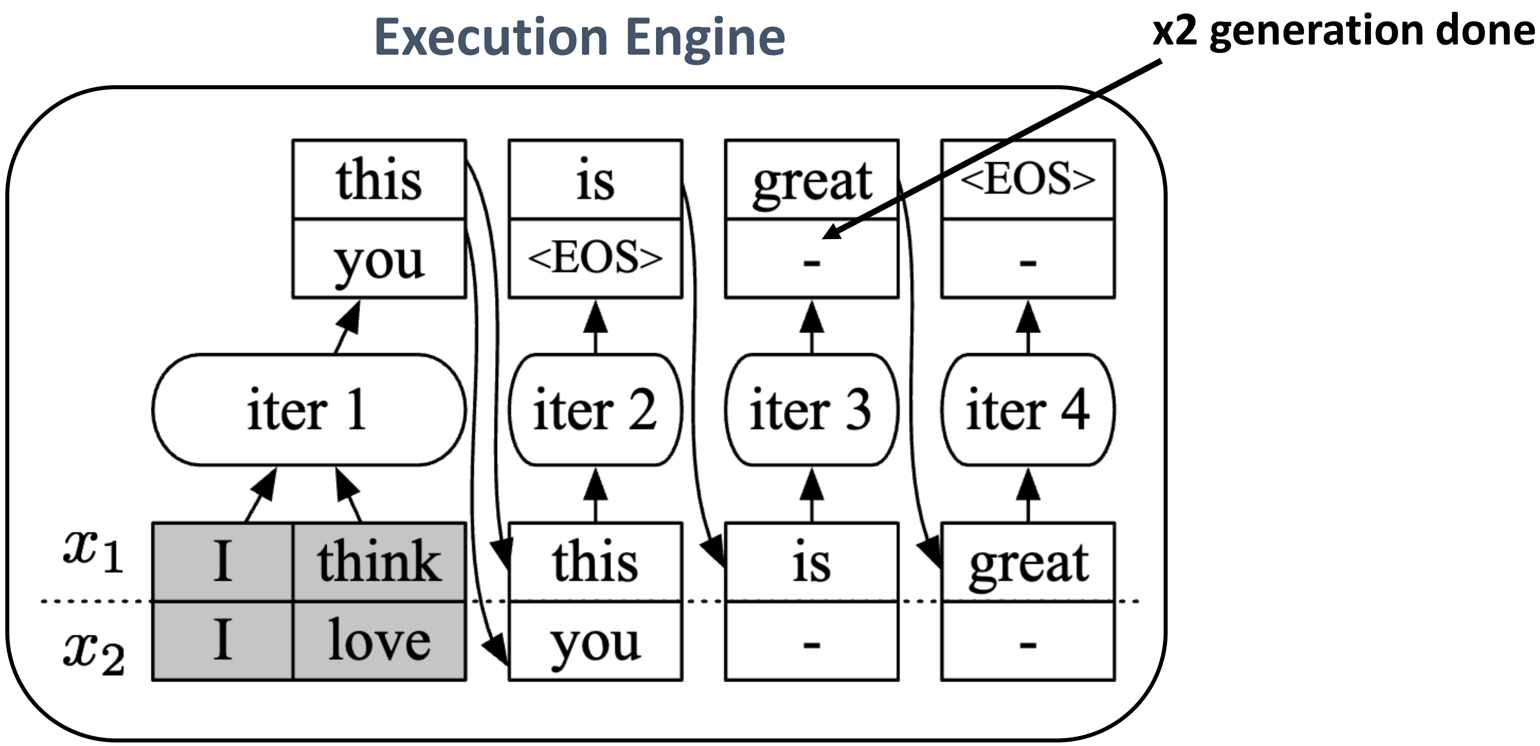


Problem 1: Request Level Scheduling



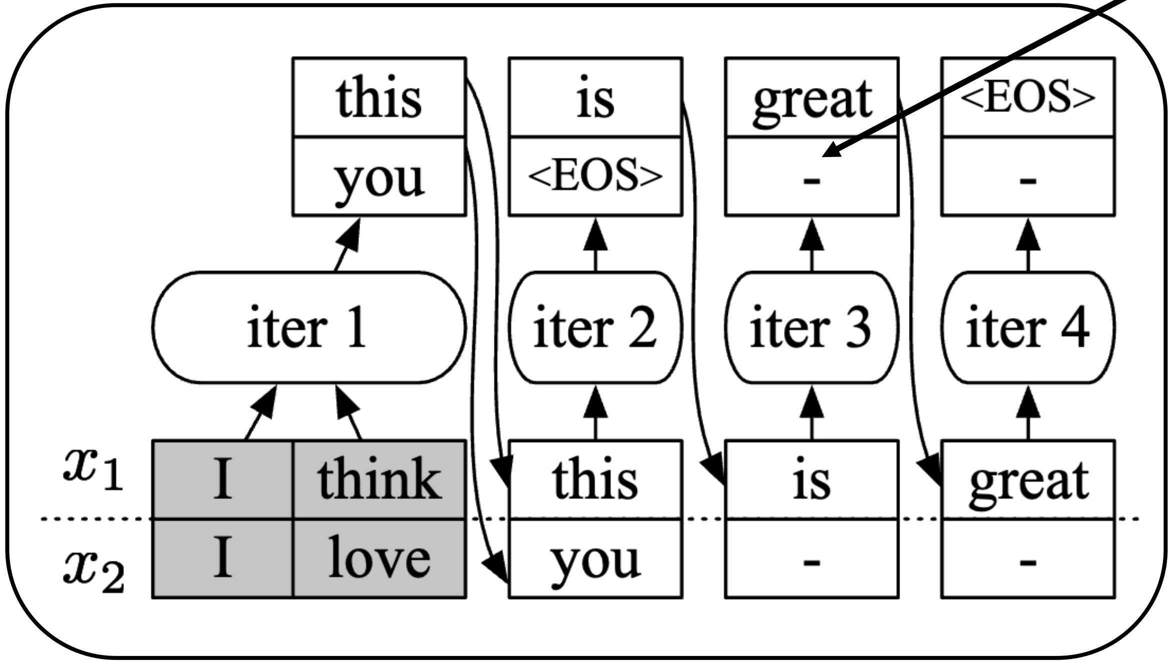
Do you see any problem?

Problem 1: Request Level Scheduling



Problem 1: Request Level Scheduling

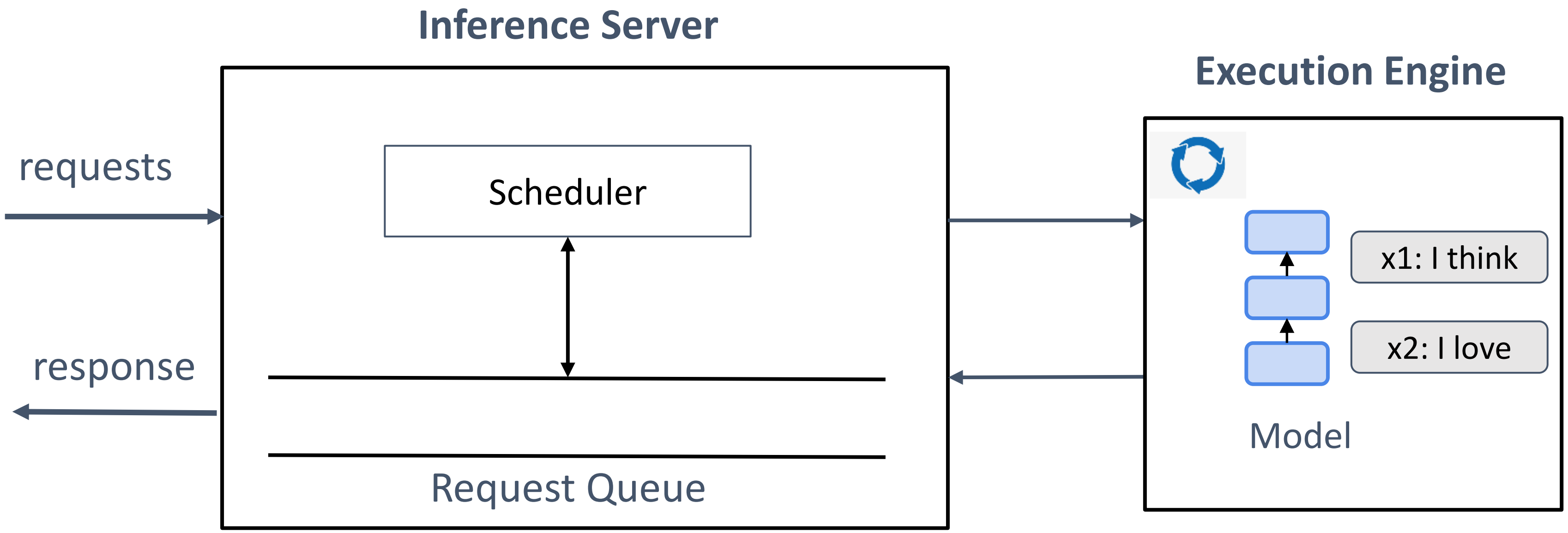
Execution Engine



x2 generation done

Early finished requests cannot return to the Inference Server → Latency increase

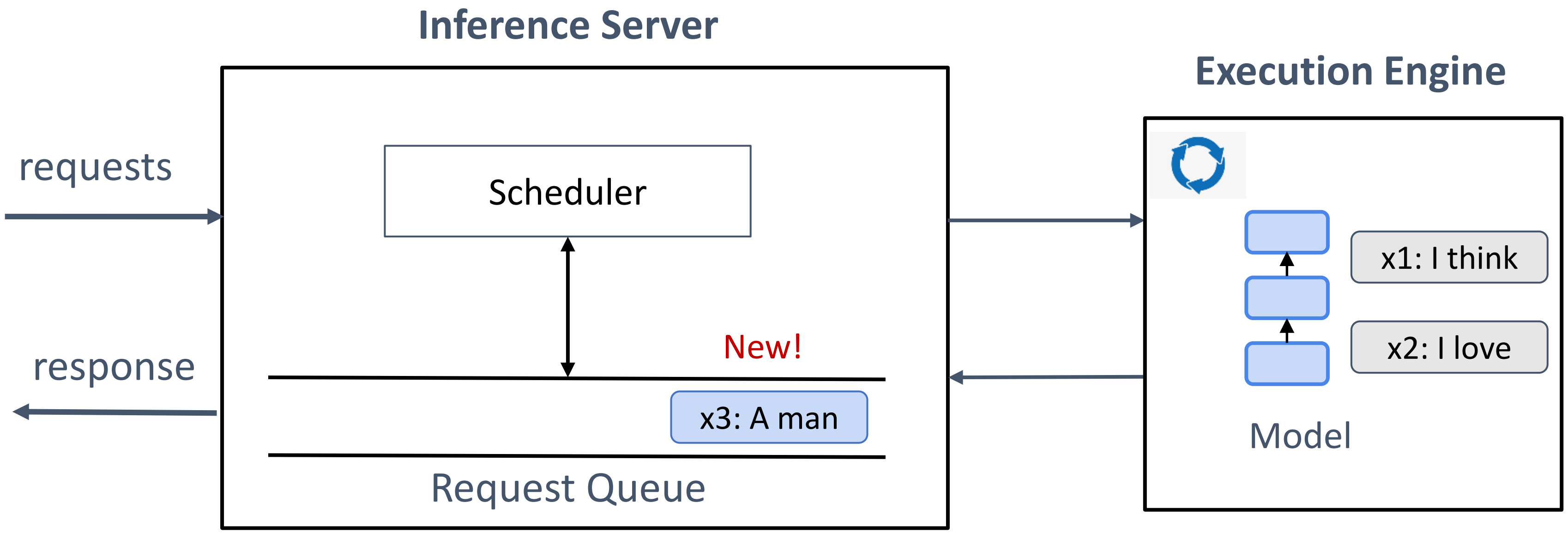
Problem 1: Request Level Scheduling



Maximum Batch Size = 3

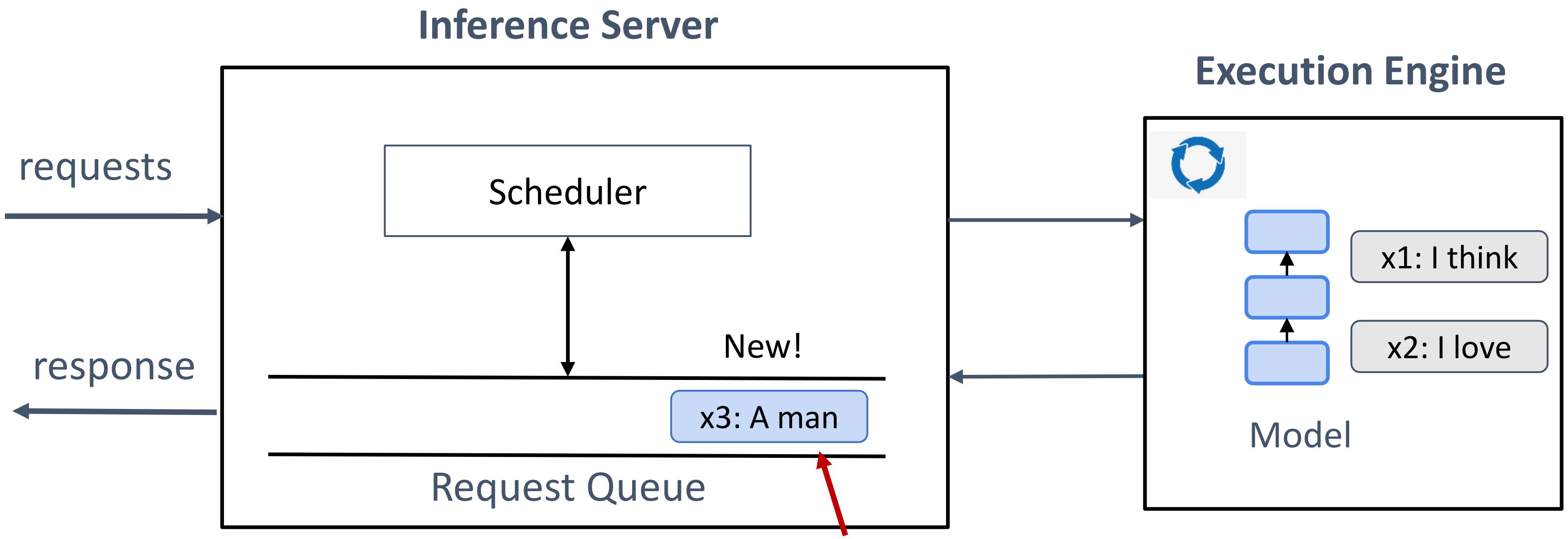
Started processing x1 and x2

Problem 1: Request Level Scheduling



Maximum Batch Size = 3

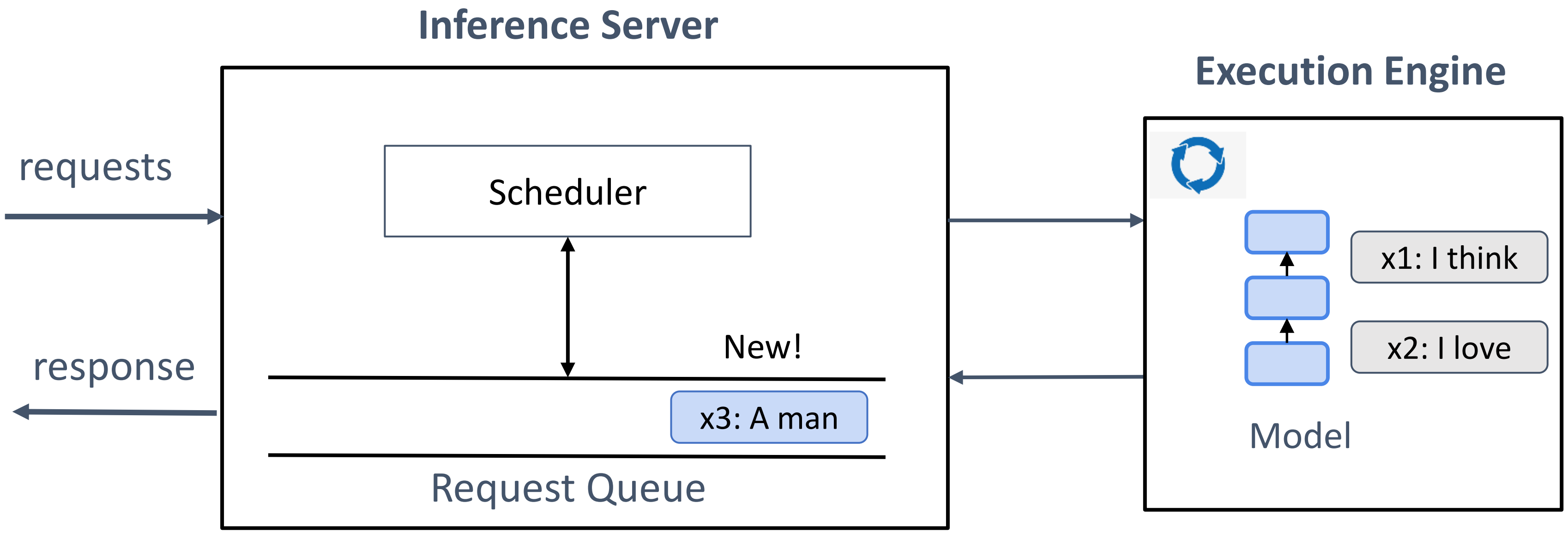
Problem 1: Request Level Scheduling



Maximum Batch Size = 3

Late join requests need to wait until engine finishes execution
→ Latency Increase

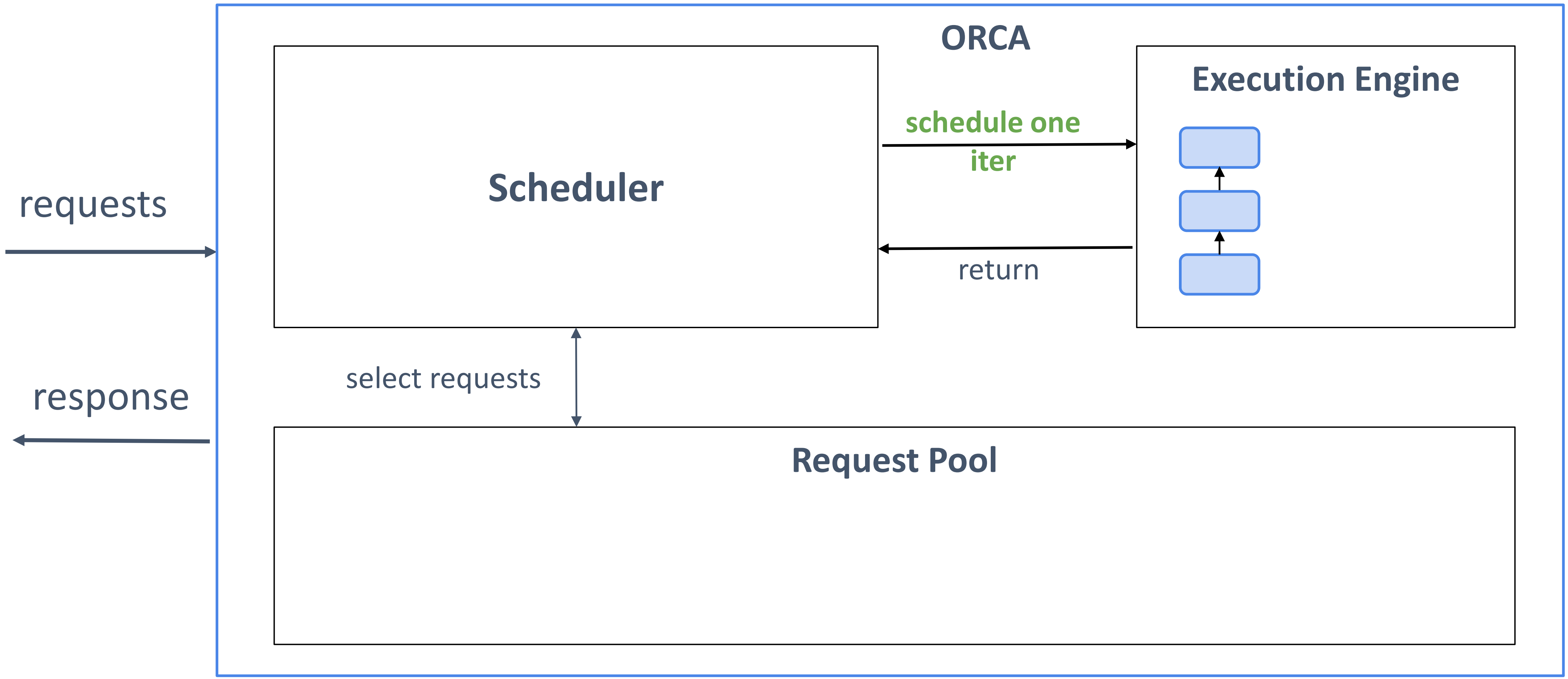
Problem 1: Request Level Scheduling



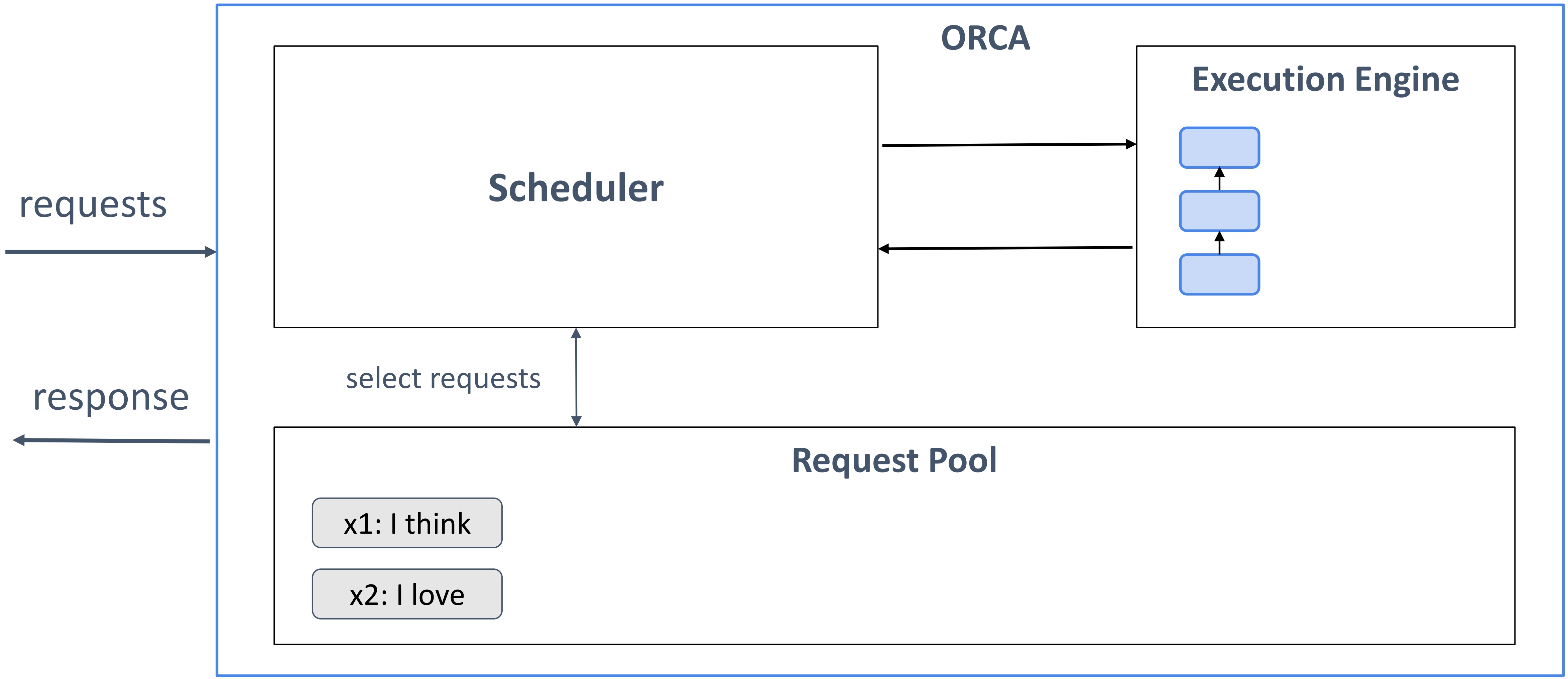
Maximum Batch Size = 3

Question: How can we avoid redundant computation and ensure late-arriving requests to be processed more promptly?

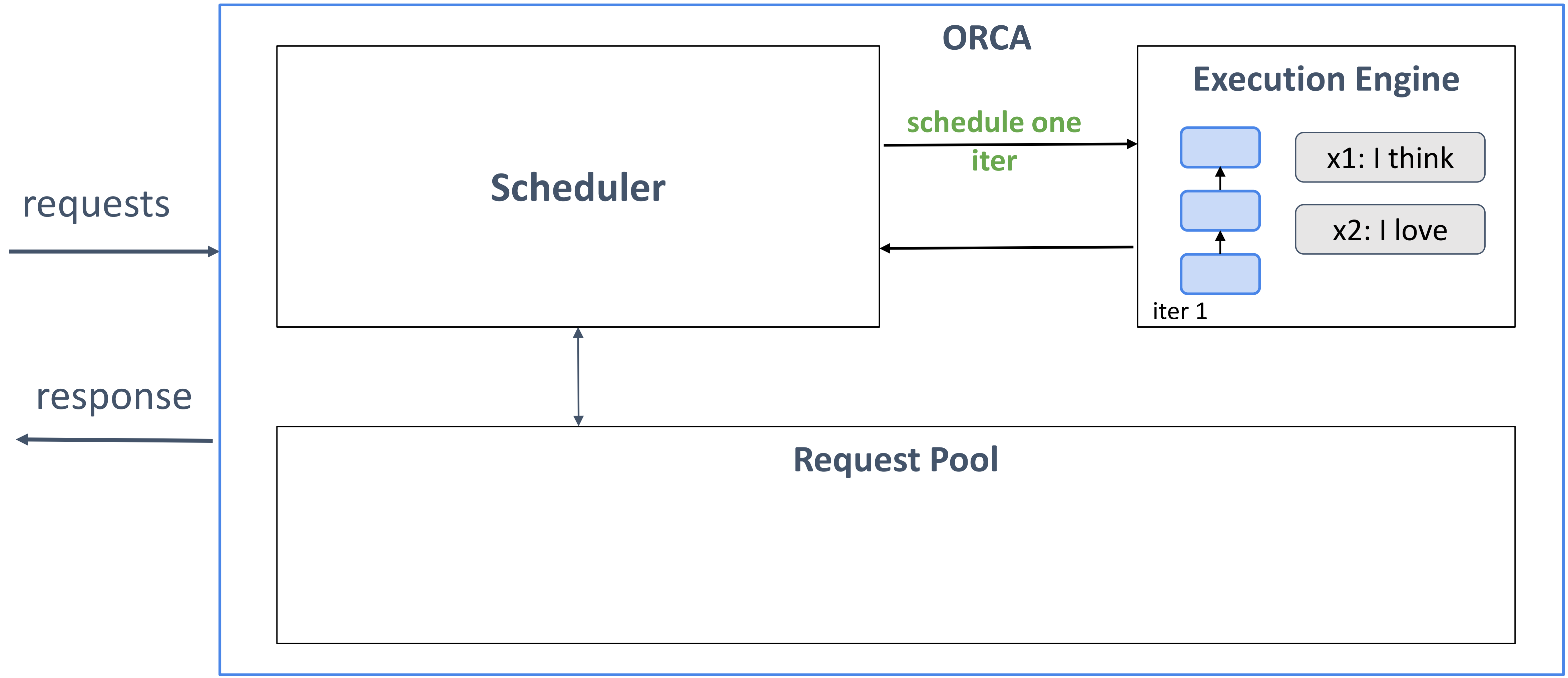
Solution 1: Iteration Level Scheduling



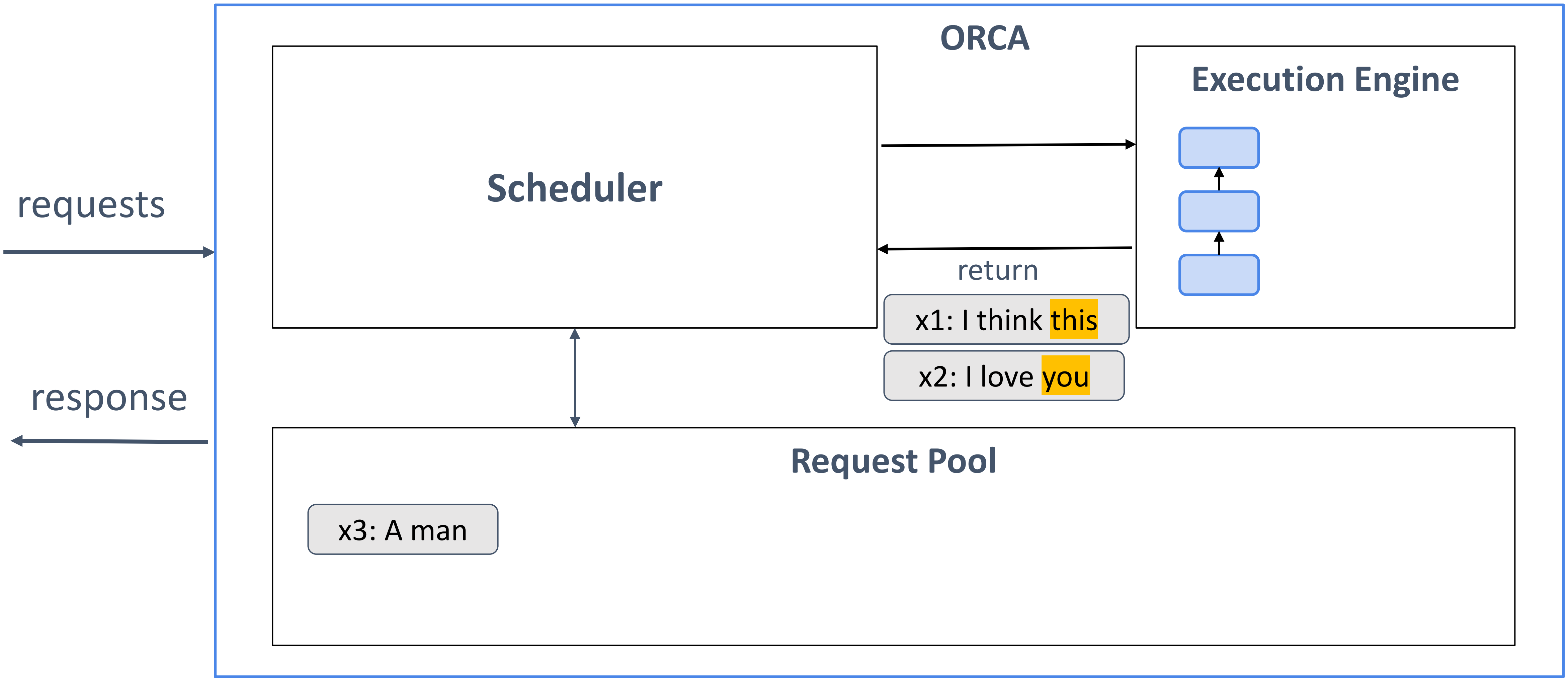
Solution 1: Iteration Level Scheduling



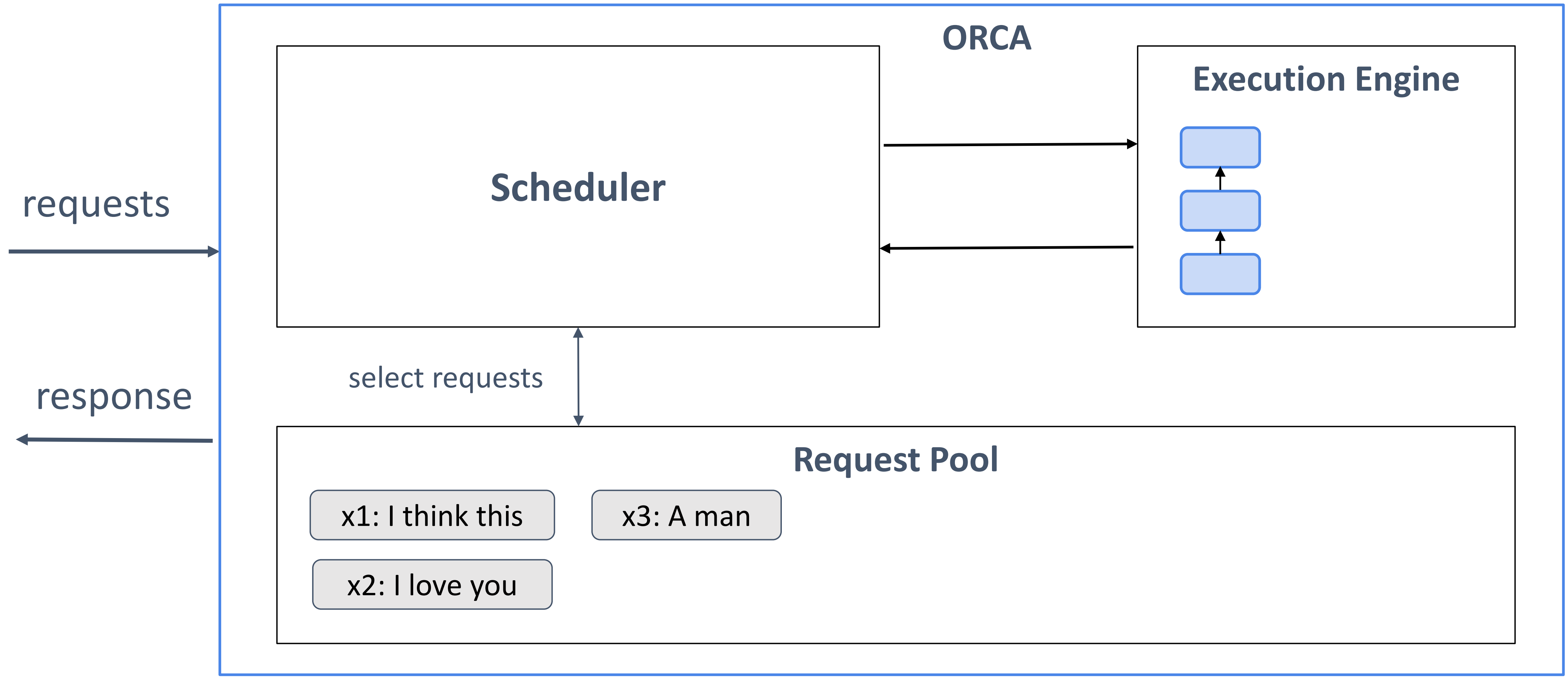
Solution 1: Iteration Level Scheduling



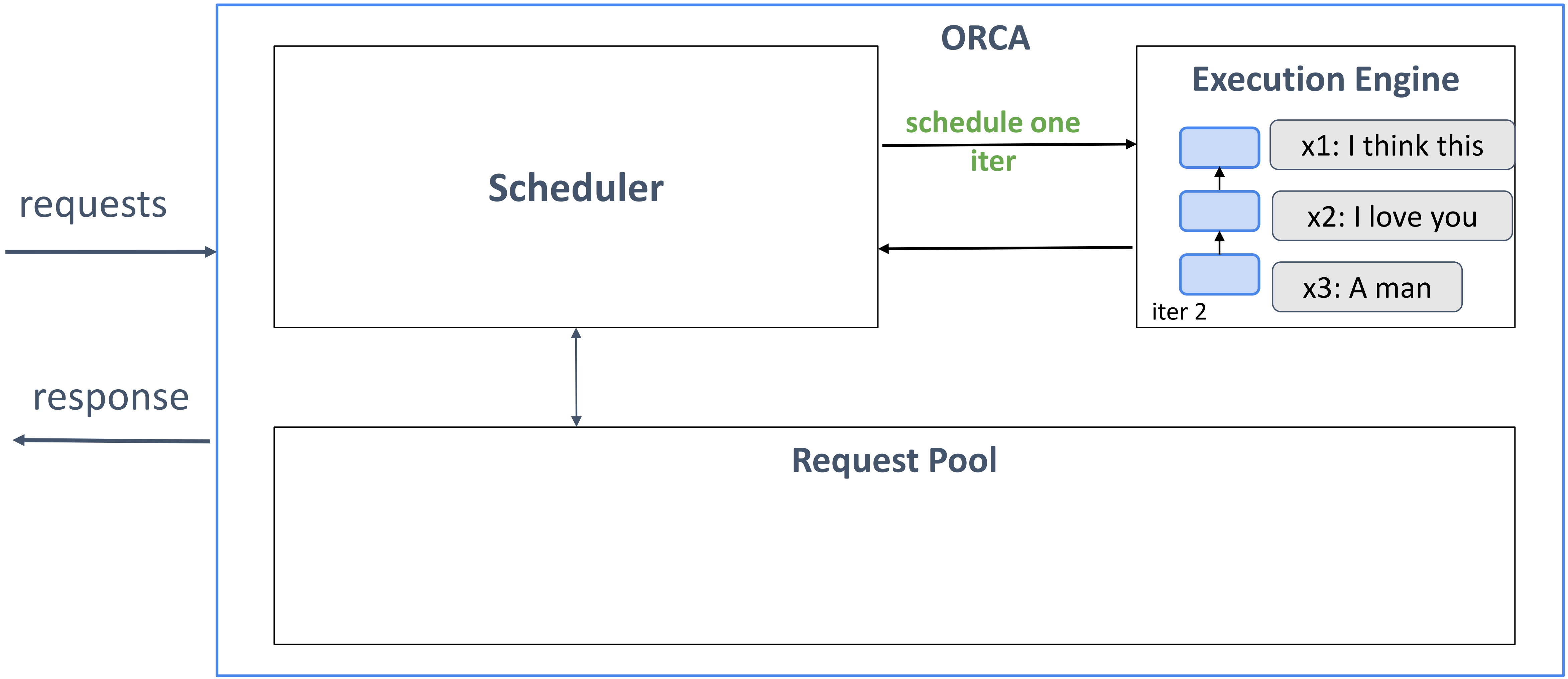
Solution 1: Iteration Level Scheduling



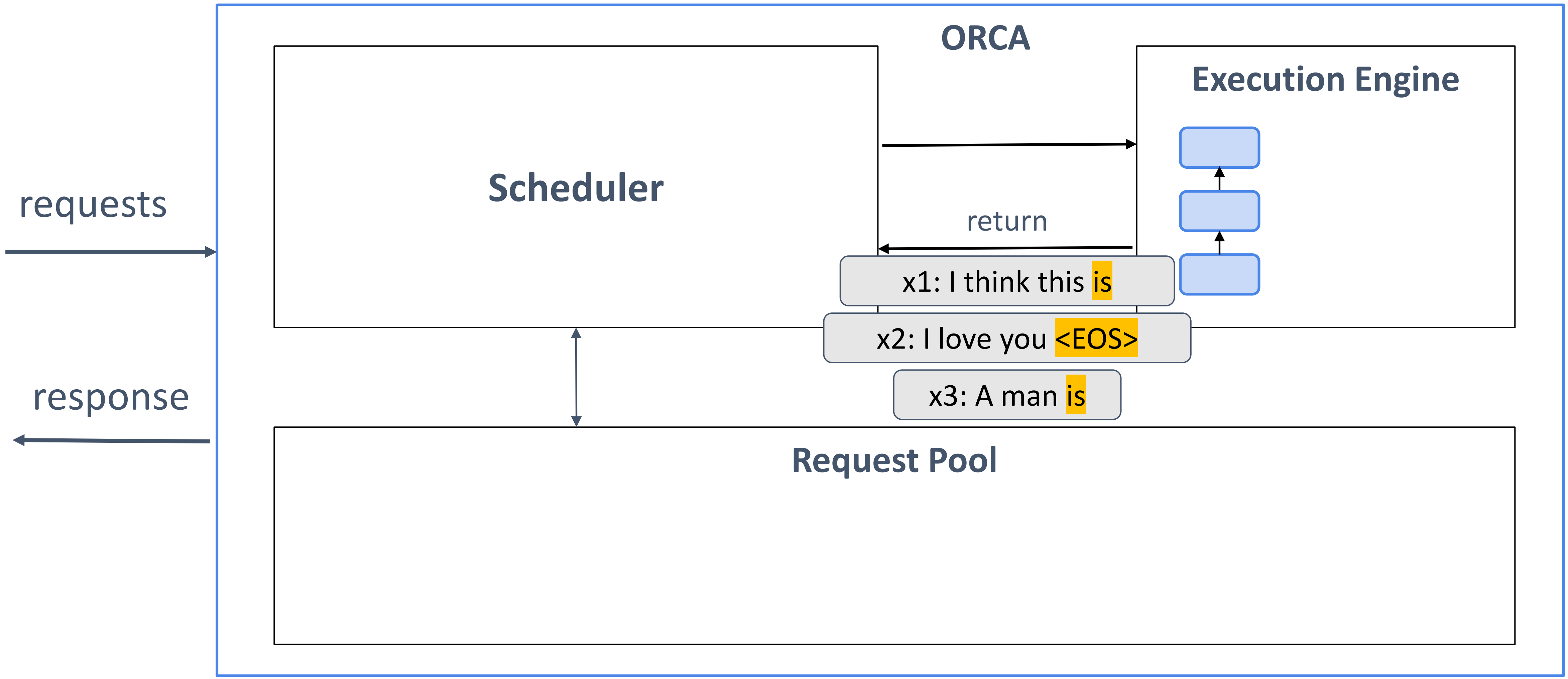
Solution 1: Iteration Level Scheduling



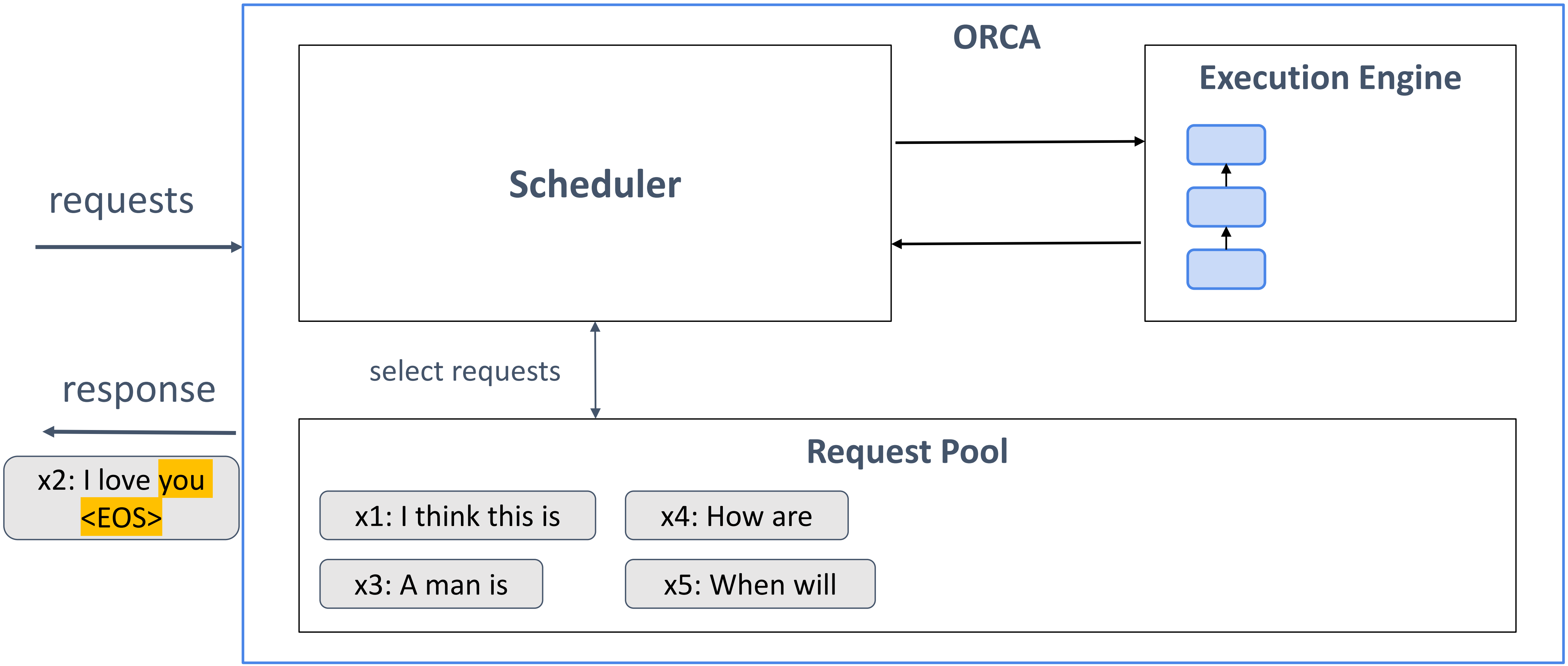
Solution 1: Iteration Level Scheduling



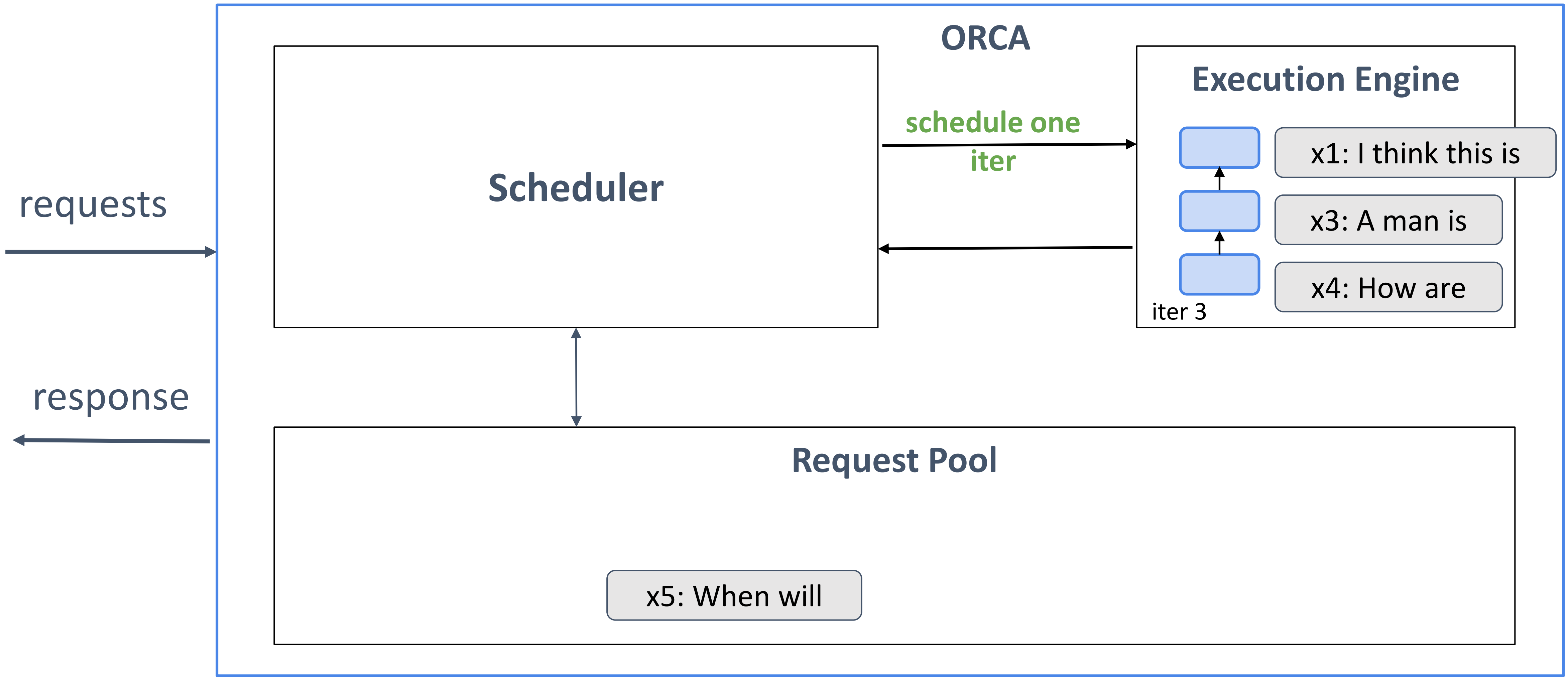
Solution 1: Iteration Level Scheduling



Solution 1: Iteration Level Scheduling



Solution 1: Iteration Level Scheduling



Solution 1: Iteration Level Scheduling

Iteration Level Scheduling handles early finished requests and late joining requests

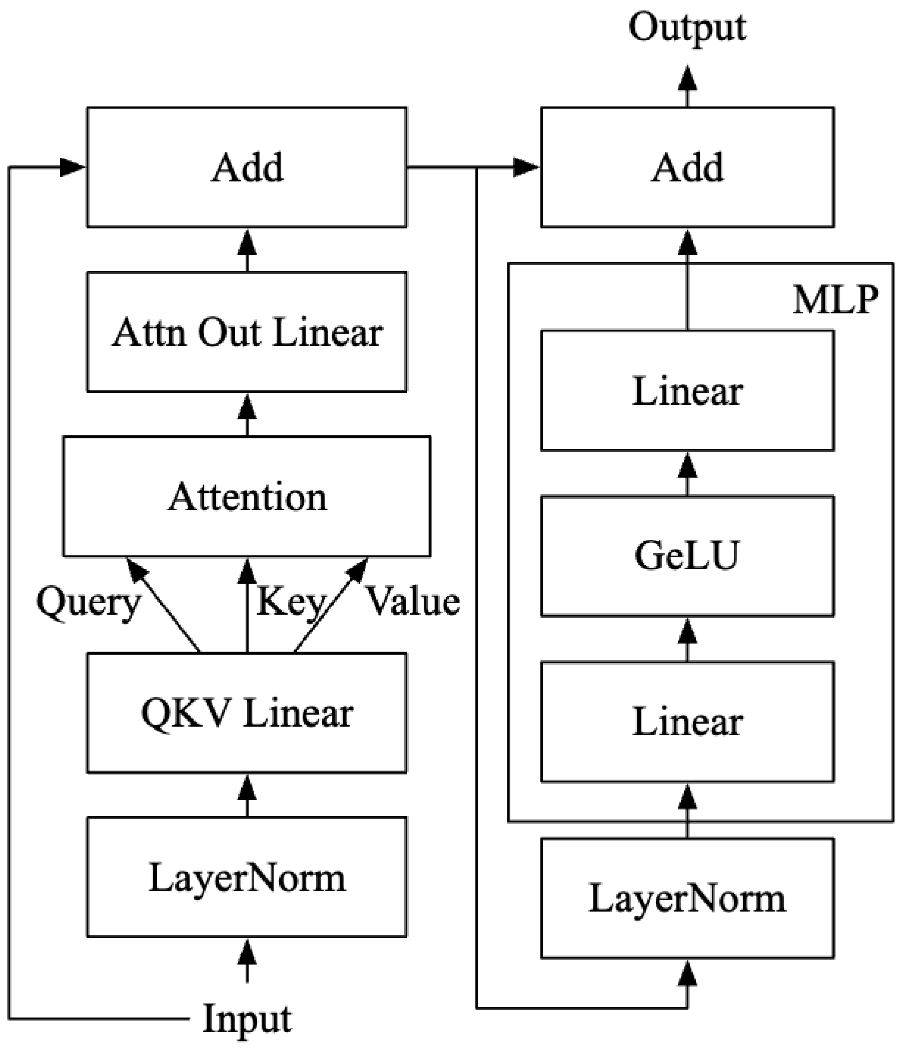
Problem 2: How to Batch Requests?

Let's assume Batch Size B = 1

Input Dimension: [L x H] (L=sequence length, H=hidden dim.)

Attention Operation:

1. **QK^T** : [LxH] x [HxL] → [L x L]
2. **P = softmax(QK^T)** : [L x L]
3. **O = PV** : [LxL] x [LxH] → [L x H]



Problem 2: How to Batch Requests?

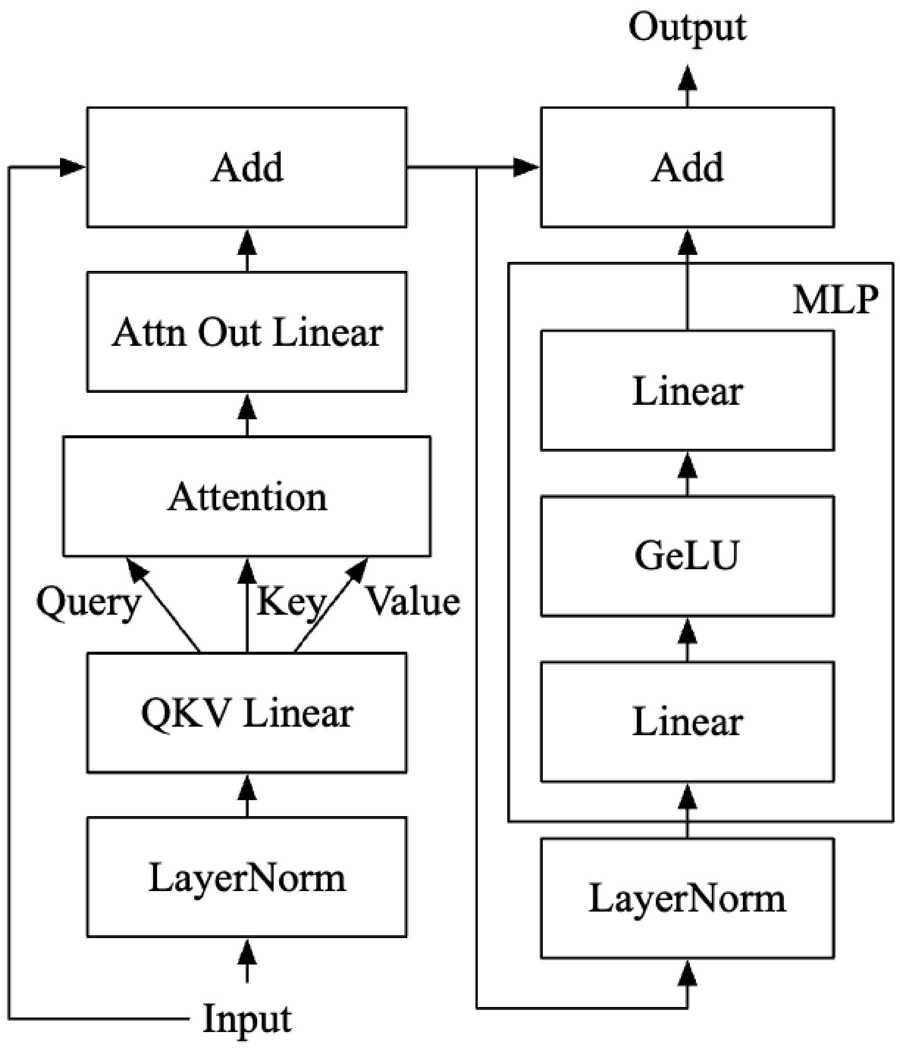
Let's assume Batch Size B = 1

Input Dimension: [L x H] (L=sequence length, H=hidden dim.)

Attention Operation:

1. **QK^T** : [LxH] x [HxL] → [L x L]
2. **P = softmax(QK^T)** : [L x L]
3. **O = PV** : [LxL] x [LxH] → [L x H]

With Batch Size B, **QK^T** will be **[B x L x L]**



Problem 2: How to Batch Requests?

Let's assume Batch Size B = 1

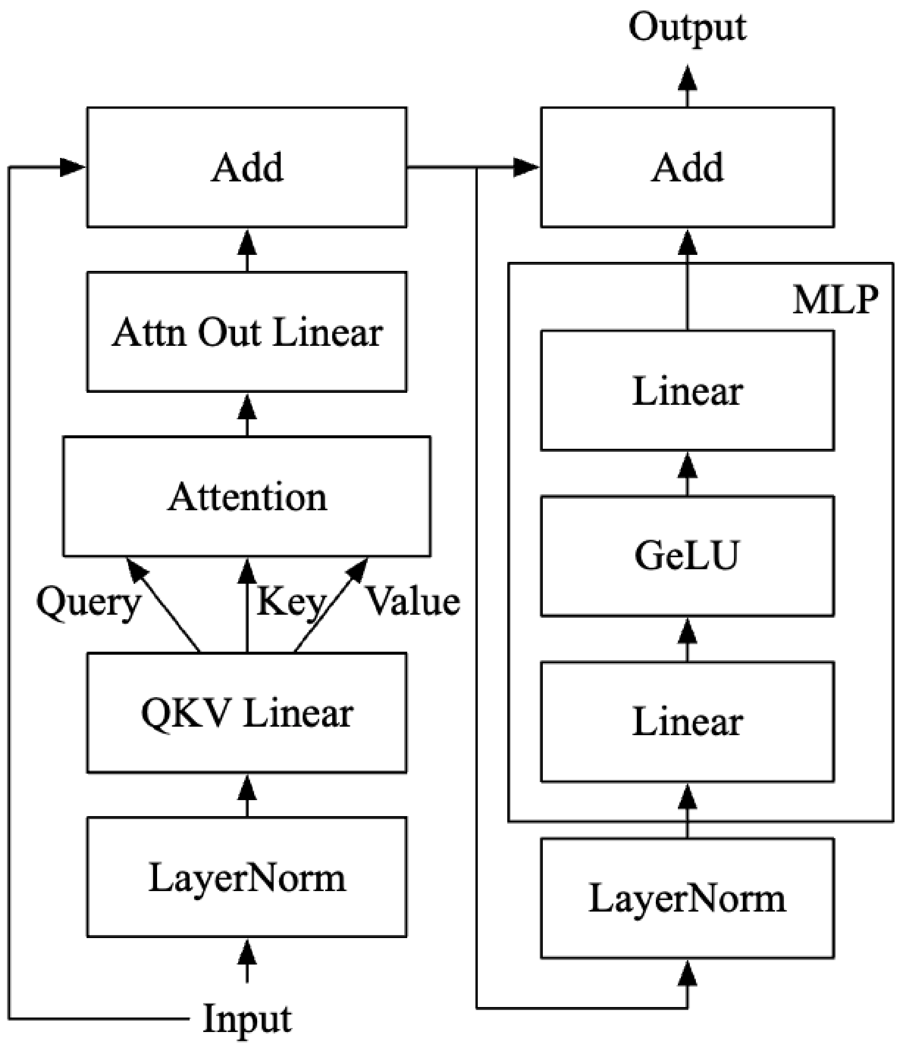
Input Dimension: [L x H] (L=sequence length, H=hidden dim.)

Attention Operation:

1. **QK^T** : [LxH] x [HxL] → [L x L]
2. **P = softmax(QK^T)** : [L x L]
3. **O = PV** : [LxL] x [LxH] → [L x H]

With Batch Size B, **QK^T** will be **[B x L x L]**

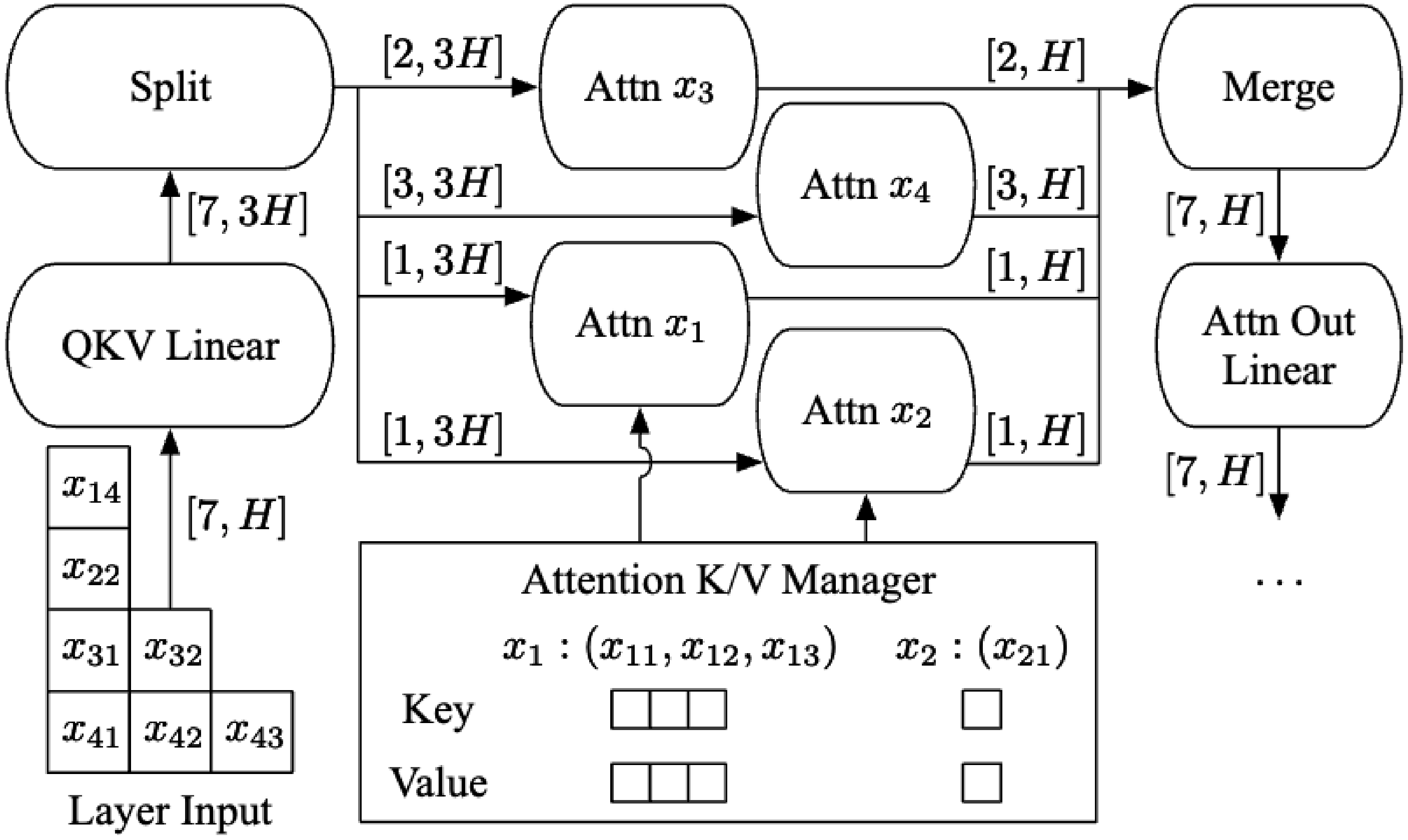
With different sequence lengths, QK^T cannot be easily computed



Solution 2: Selective Batching

Only **Attention operation** does not work with batching tensors with diff. L_i

Batch for other ops. (Layer Norm, GeLU, etc.)



Solution 2: Selective Batching

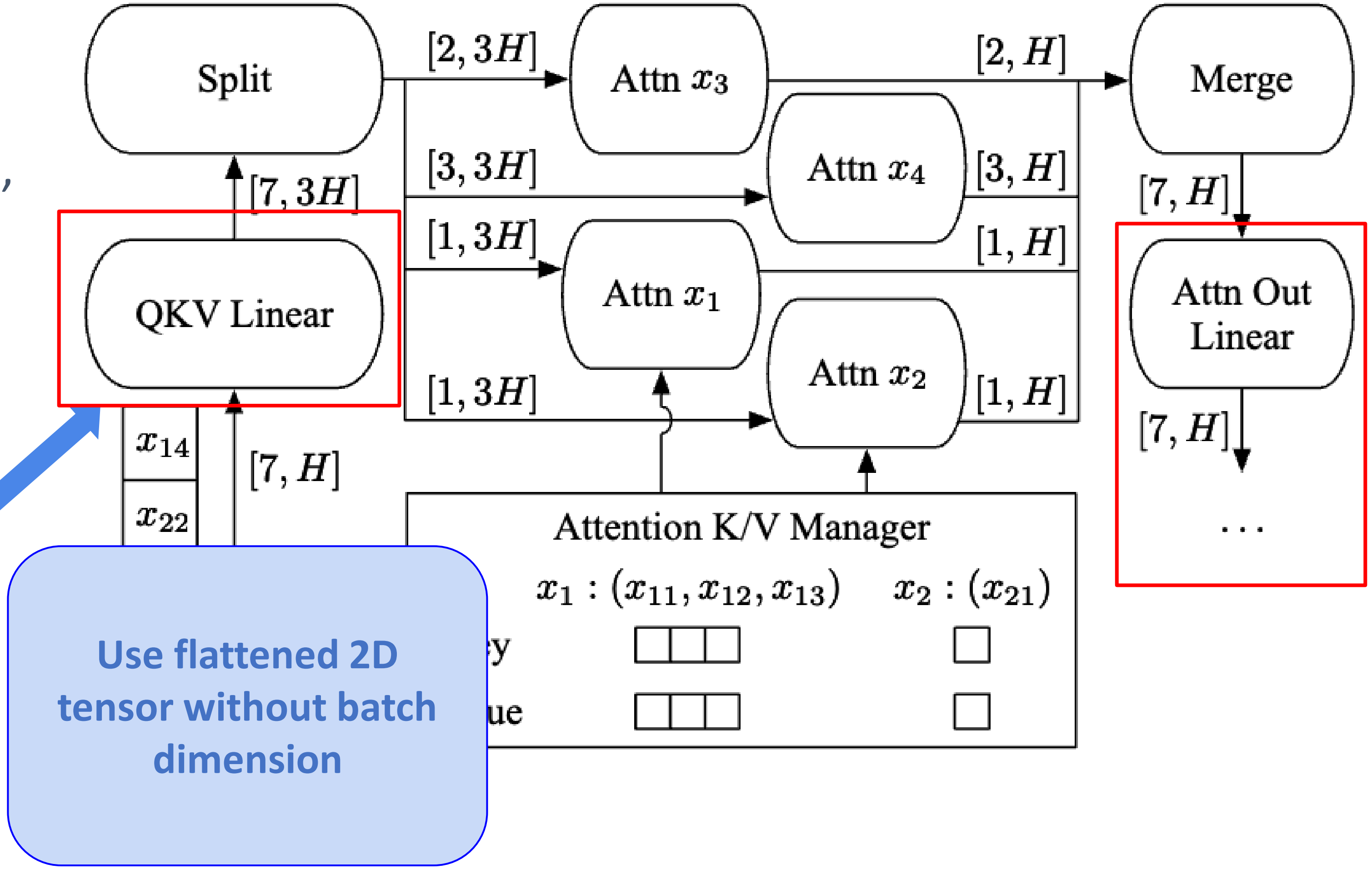
Only **Attention operation** does not work with batching tensors with diff. L_i

Batch for other ops. (Layer Norm, GeLU, etc.)

Coalesce $[L_i, H]$ tensor to $[\Sigma L_i, H]$ for batching

x1: [1,H]
x2: [1,H]
x3: [2,H]
x4: [3,H]

[7,H] tensor



Solution 2: Selective Batching

Split, process each request and merge tensors

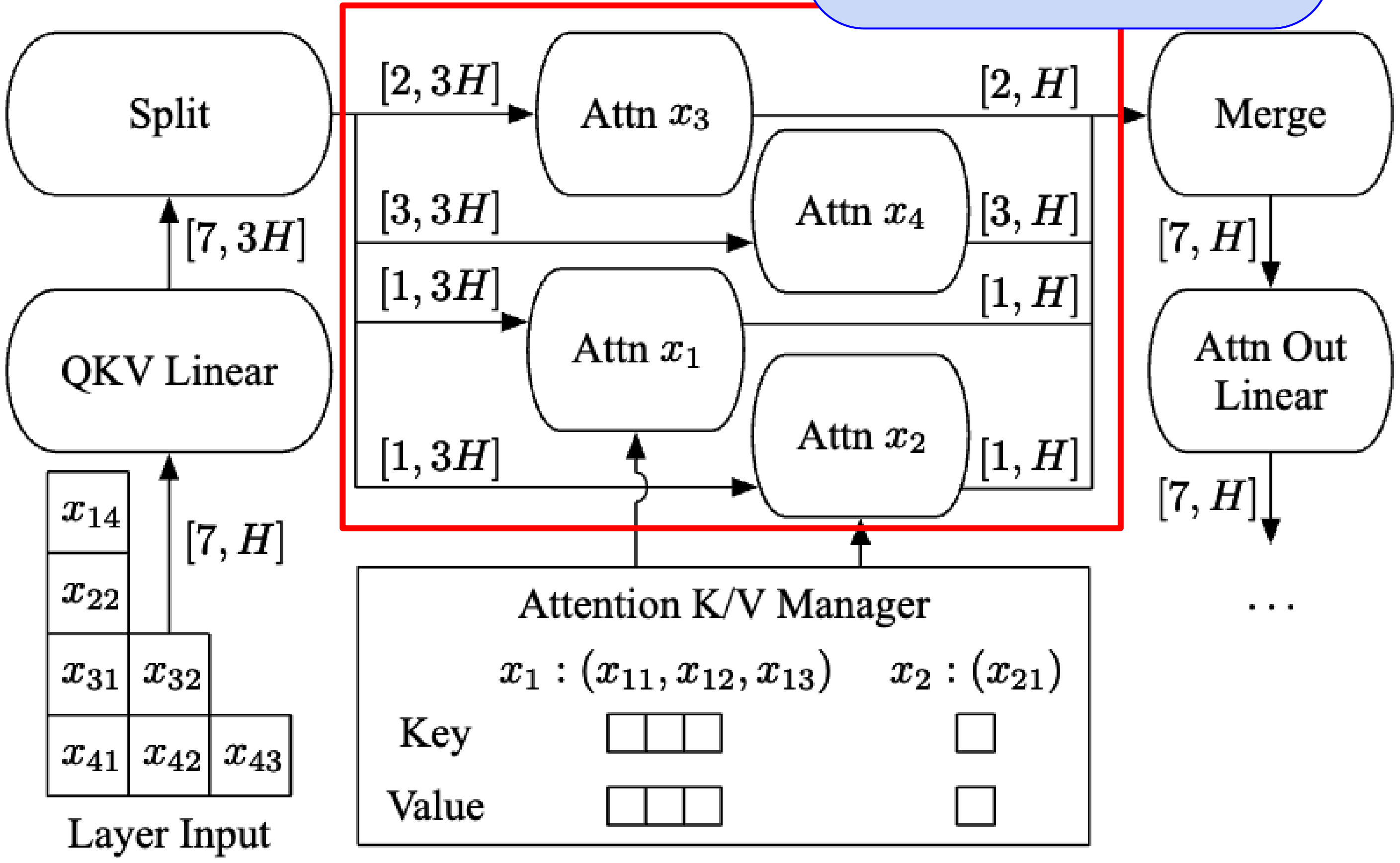
Only **Attention operation** does not work with batching tensors with diff. L_i

Batch for other ops. (Layer Norm, GeLU, etc.)

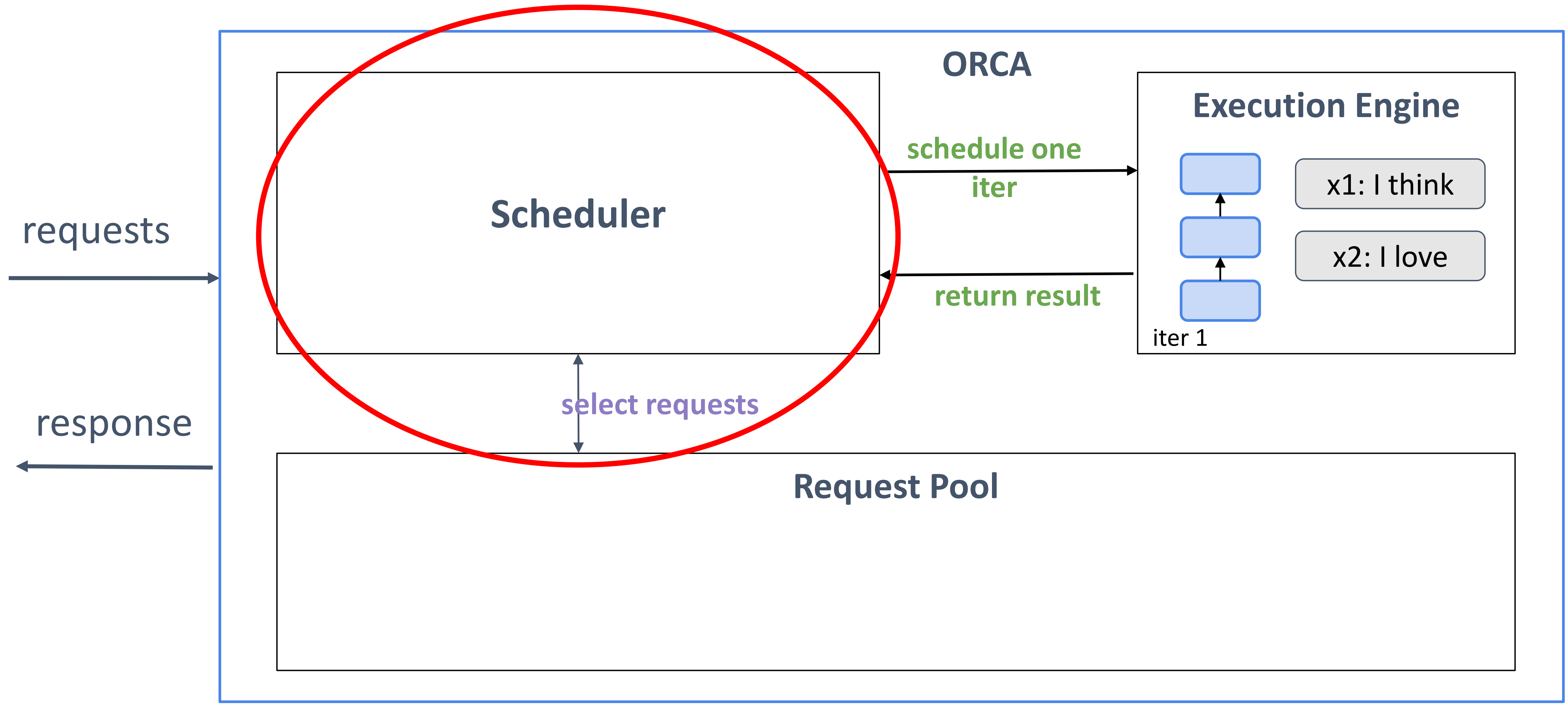
Coalesce $[L_i, H]$ tensor to $[\Sigma L_i, H]$ for batching

x1: [1,H]
x2: [1,H]
x3: [2,H]
x4: [3,H]

[7,H] tensor



LLM Inference Scheduler



LLM Inference Scheduler

- Enforces iteration-level first-come-first-served (FCFS) property
- Maximum batch size → Throughput vs. Latency control knob
- Reserves max_tokens memory slots per request
- …

Throughput Experiments

- Question: When would continuous batching provide more benefits than sequence batching?

Throughput Experiments

- Hypothesis
  - Continuous batching performs better the more variance there is in sequence lengths
- Frameworks
- Setup – hardware/model
- Setup – data
- Results

Throughput Experiments: Frameworks

- Static batching
  - HuggingFace
  - NVIDIA FasterTransformer
- Continuous batching
  - HuggingFace text-generation-inference (TGI)
  - Ray Serve
  - vLLM

https://www.anyscale.com/blog/continuous-batching-llm-inference

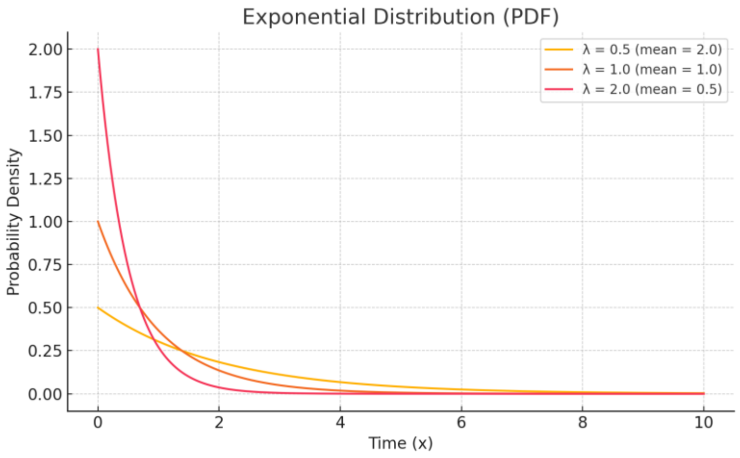
Throughput Experiments: Hardware/model

- NVIDIA A100-40GB
- Meta's OPT-13B
  - dtype = float16 → 26GB for parameters
- No tensor parallelism

https://www.anyscale.com/blog/continuous-batching-llm-inference

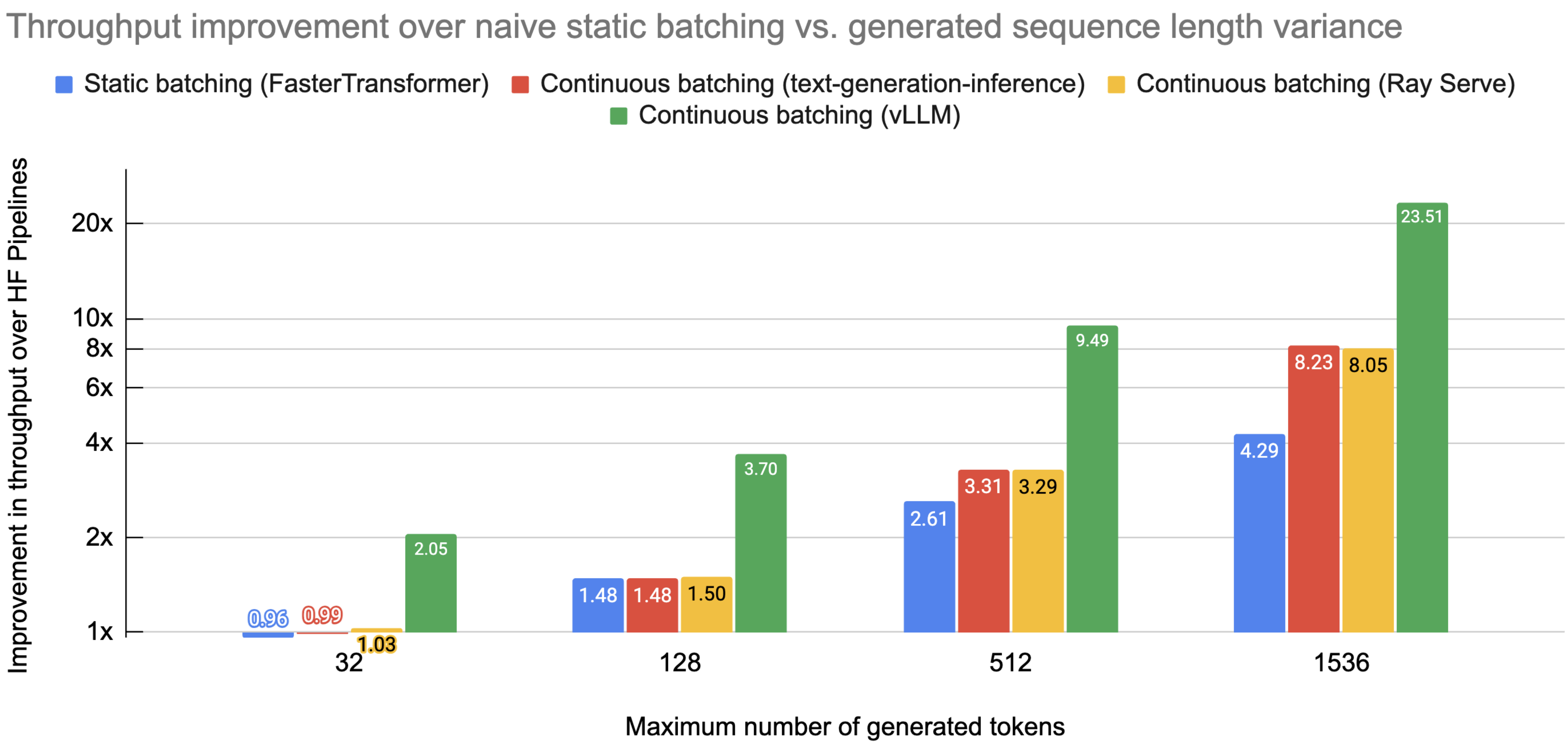
Throughput Experiments: Data

- Hypothesis
  - Continuous batching performs better the more variance there is in sequence lengths
- How to test?
  - Generate 1000 prompts each with 512 input tokens
  - Generate predetermined output length for each prompt, following an exponential distribution
  - Configure model to ignore EOS token



https://www.anyscale.com/blog/continuous-batching-llm-inference

Throughput Improvement from Continuous Batching



https://www.anyscale.com/blog/continuous-batching-llm-inference

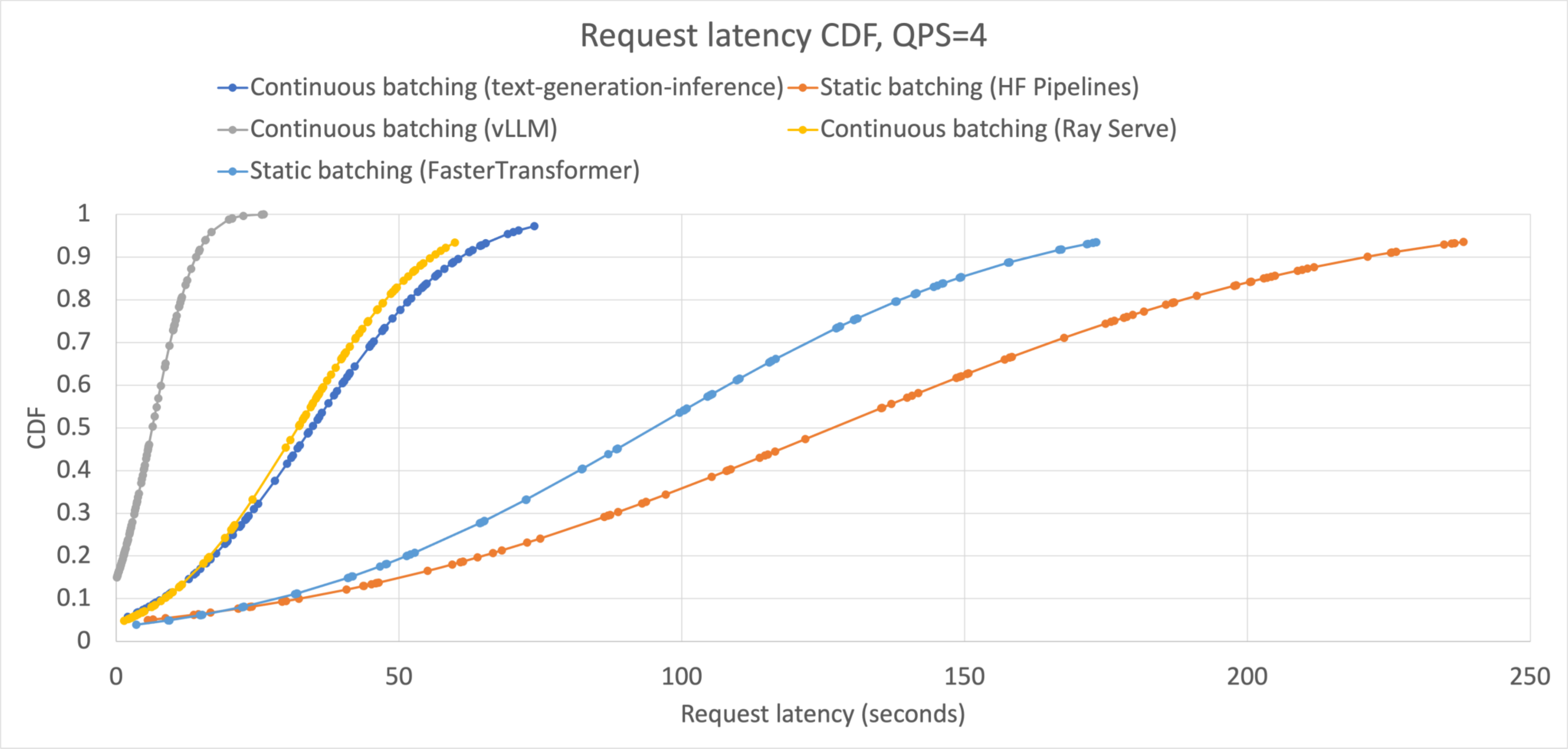
How does vLLM beat TGI

- vLLM uses PagedAttention – extra batch size space

Contiguous Memory

Non-Contiguous Memory

E2E Latency Experiments: Results

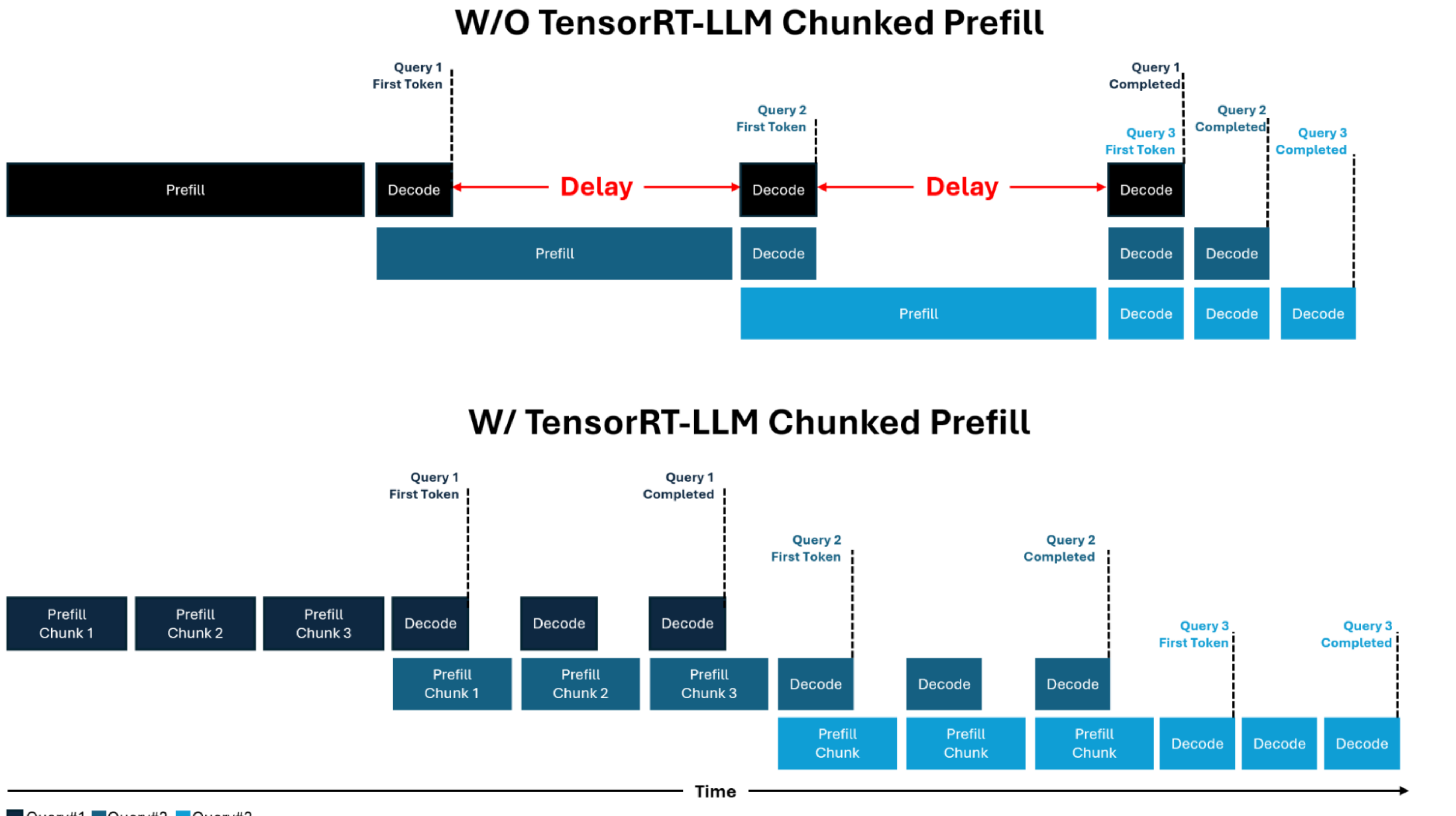


https://www.anyscale.com/blog/continuous-batching-llm-inference

Summary: Continuous Batching

- Continuous batching handles early-finished and late-arrived requests more efficiently
- Fills GPU capacity after each token generation
- As variance in sequence length increases, continuous batching increases GPU utilization

LLM Inference Scheduler: Chunked Prefill

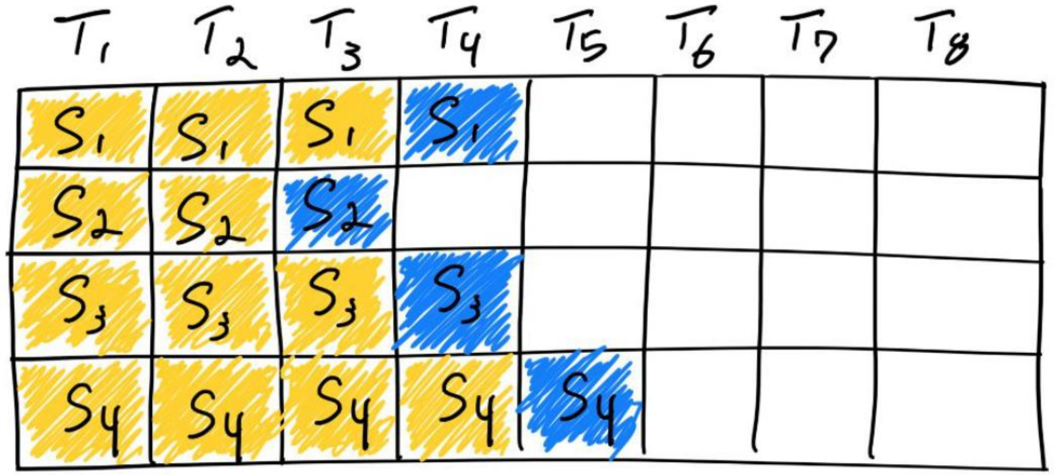


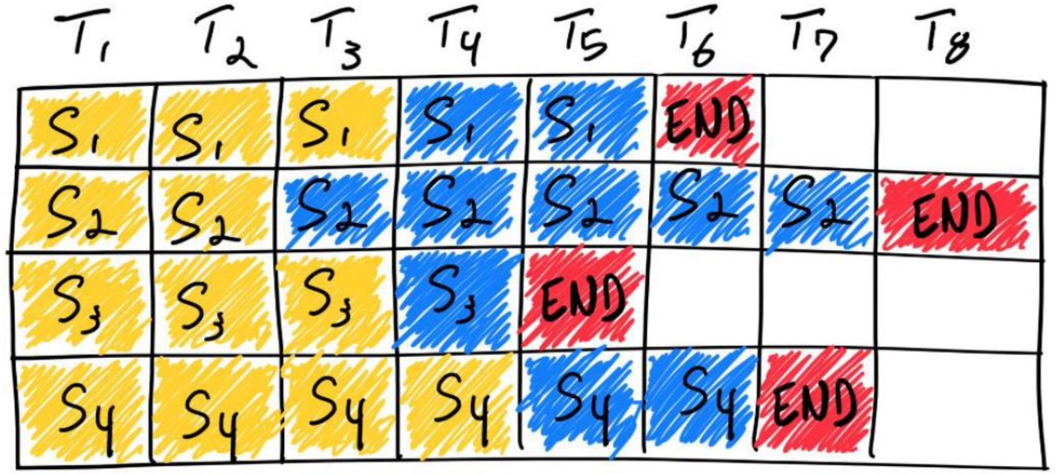
Visual for illustration purposes only. In production environments multiple Prefills are processed in parallel (not depicted). With TensorRT-LLM Inflight Batching queries are evicted once completed and new queries are added (not depictec).

Questions?

Sequence Batching (Static Batching)

- Batching multiple sequences on GPU, aka “static batching”
- Problem: GPU utilization drops as sequences complete





Legend:

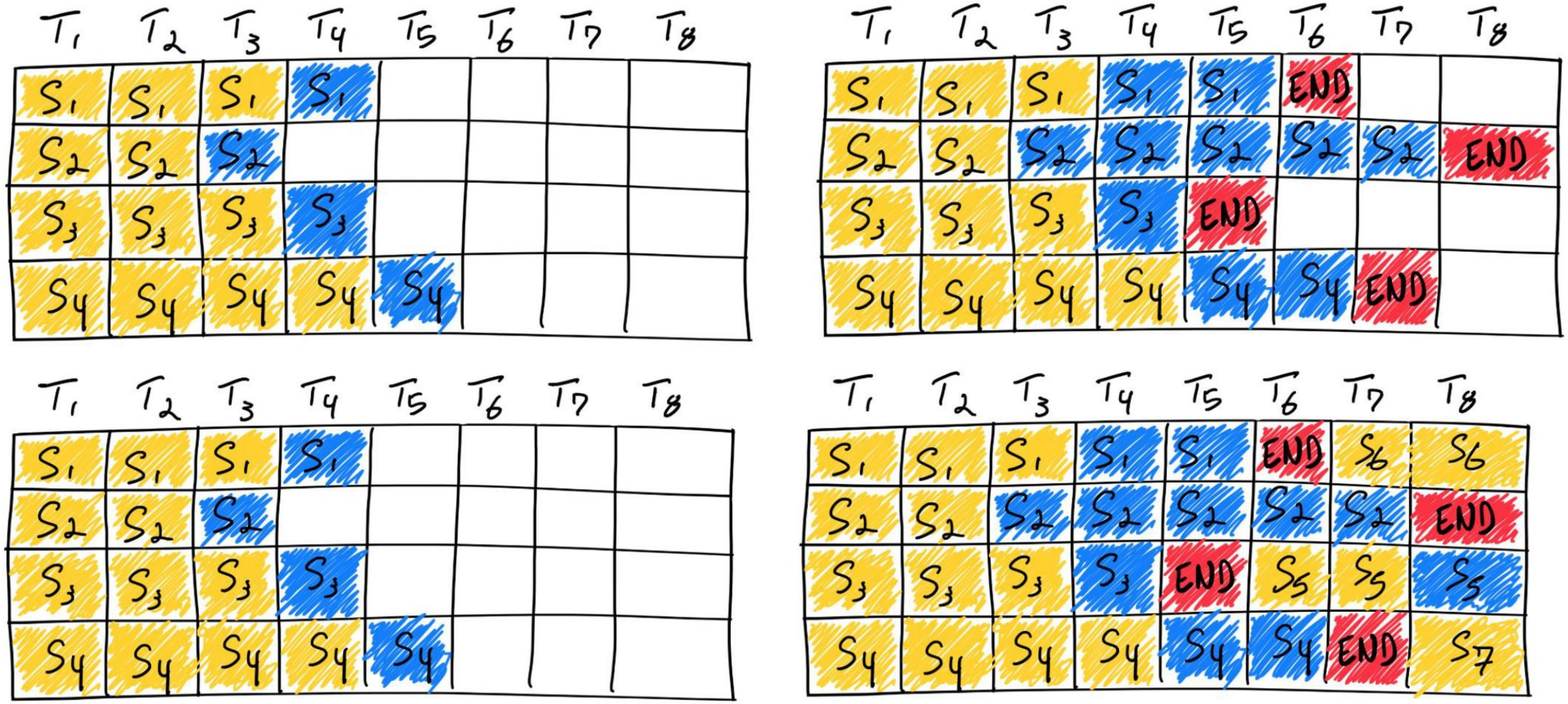
- Yellow: prompt token
- Blue: generated token
- Red: end-of-sequence token

Continuous Batching

Top: static batching
Bottom: continuous batching

Legend:

- Yellow: prompt token
- Blue: generated token
- Red: end-of-sequence token



Throughput Experiments: Results

Throughput (token/s) vs. variance in generated sequence lengths	**Generation limit** (higher limit implies higher variance in output sequence lengths)			
	max 32 tokens	max 128 tokens	max 512 tokens	max 1536 tokens
Static batching (HF Pipelines)	2988	972	214	81
Static batching (FasterTransformer)	2869	1441	558	346
Continuous batching (Ray Serve)	3090	1460	703	650
Continuous batching (text-generation-inference)	2948	1442	707	665
Continuous batching (vLLM)	6121	3592	2029	1898

https://www.anyscale.com/blog/continuous-batching-llm-inference