HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds

Authors: Chiheng Lou, Sheng Qi , Chao Jin, Dapeng Nie, Haoran Yang,

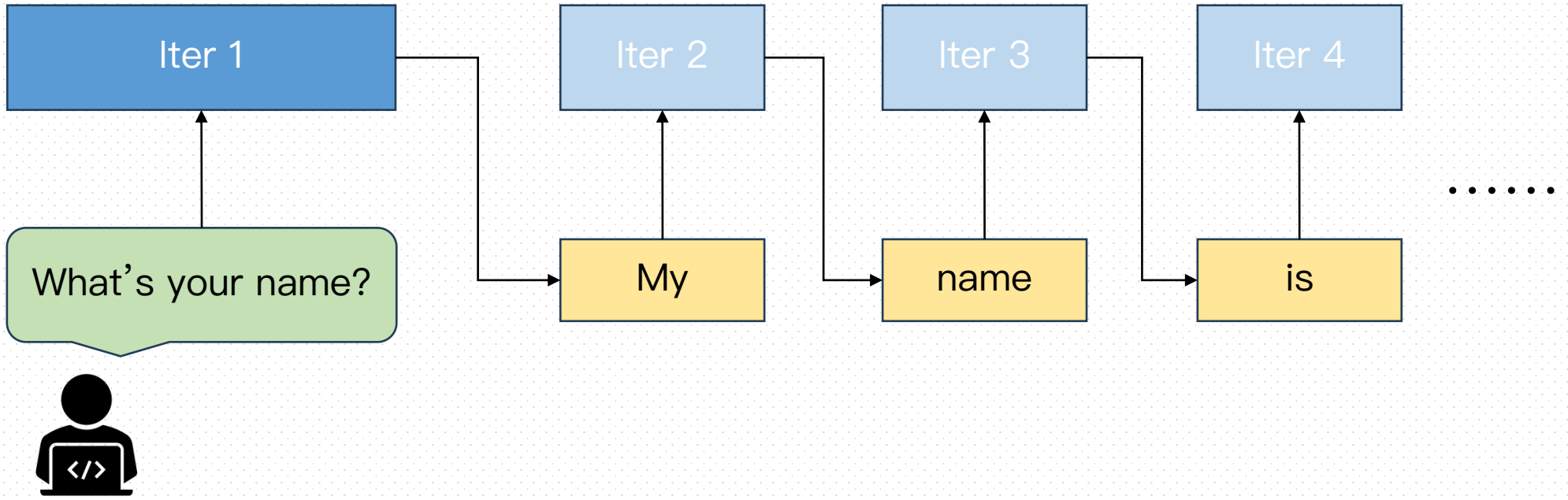
Yu Ding, Xuanzhe Liu, Xin Jin

Presented by **Jiyang Wang**

2025—11–18

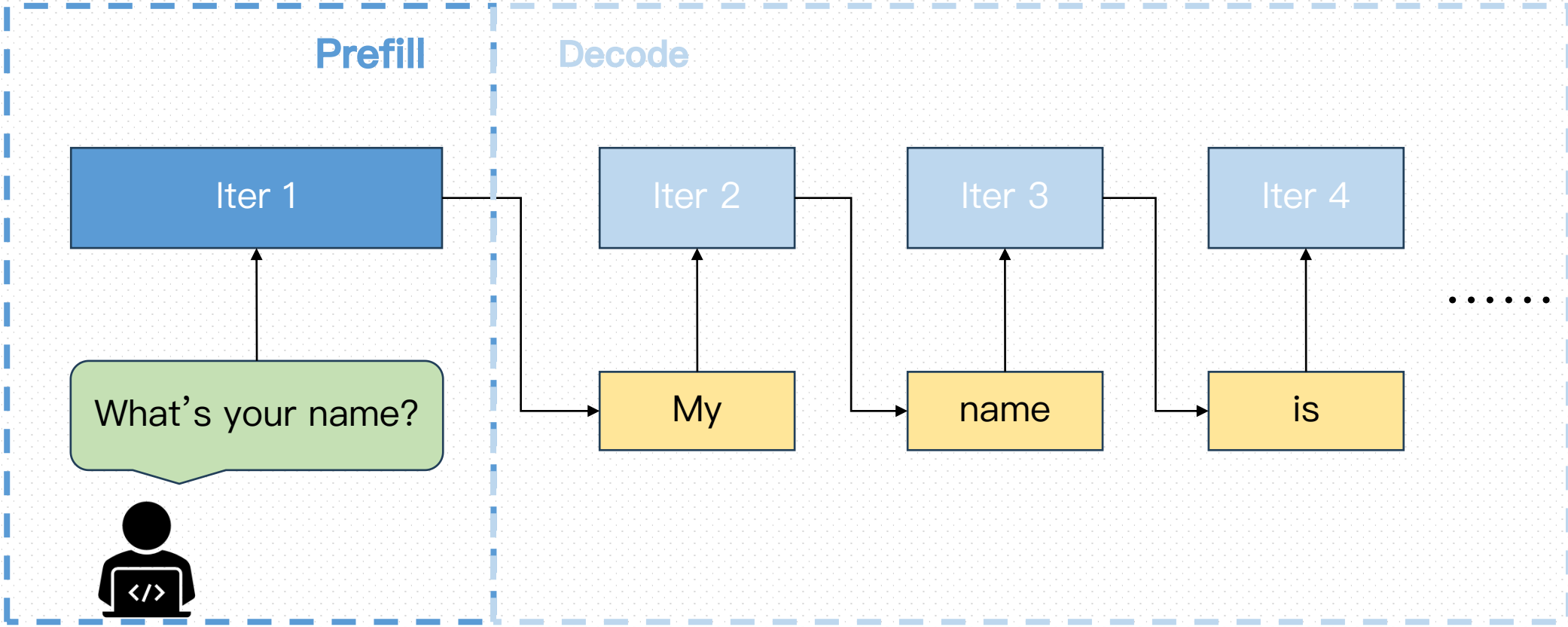
Background: LLM Inference

❑Autoregressive manner



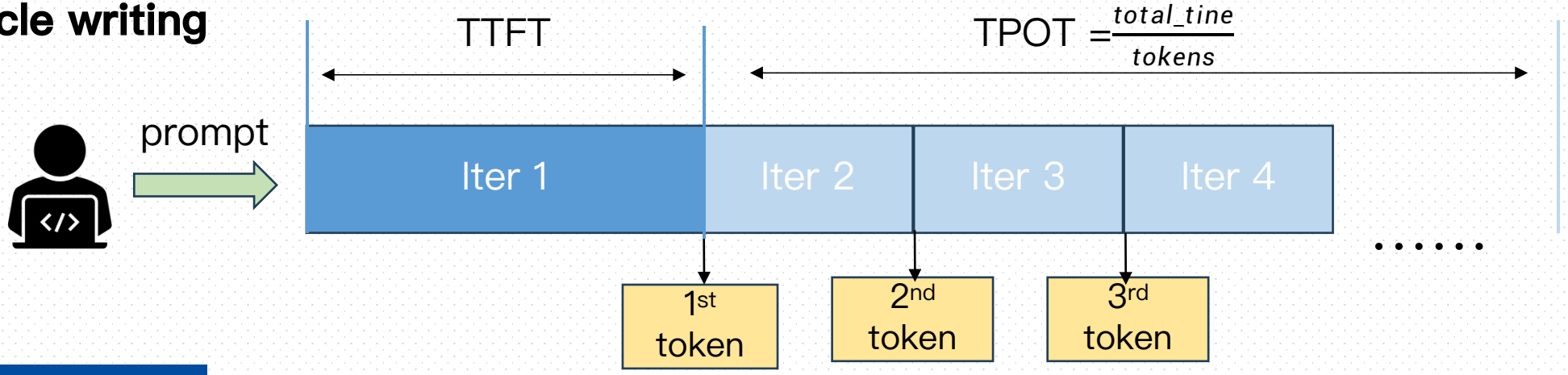
Background: LLM Inference

- Autoregressive manner
- Prefill & decode



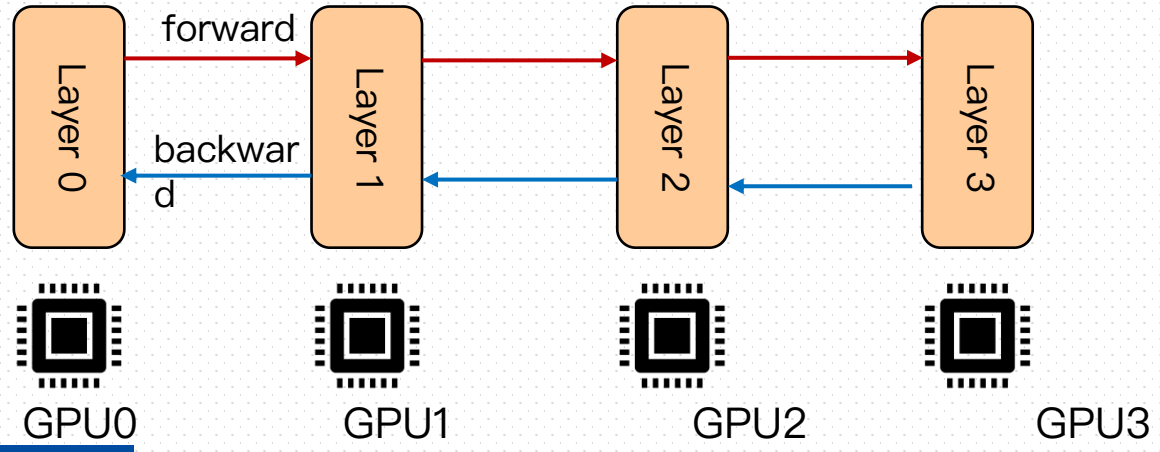
Background: LLM Inference

- Autoregressive manner
- Prefill & decode
- Service level objectives (SLOs)
  - Time to first token (TTFT)
    - Real-time chatbot
  - Time per output token (TPOT)
    - Article writing

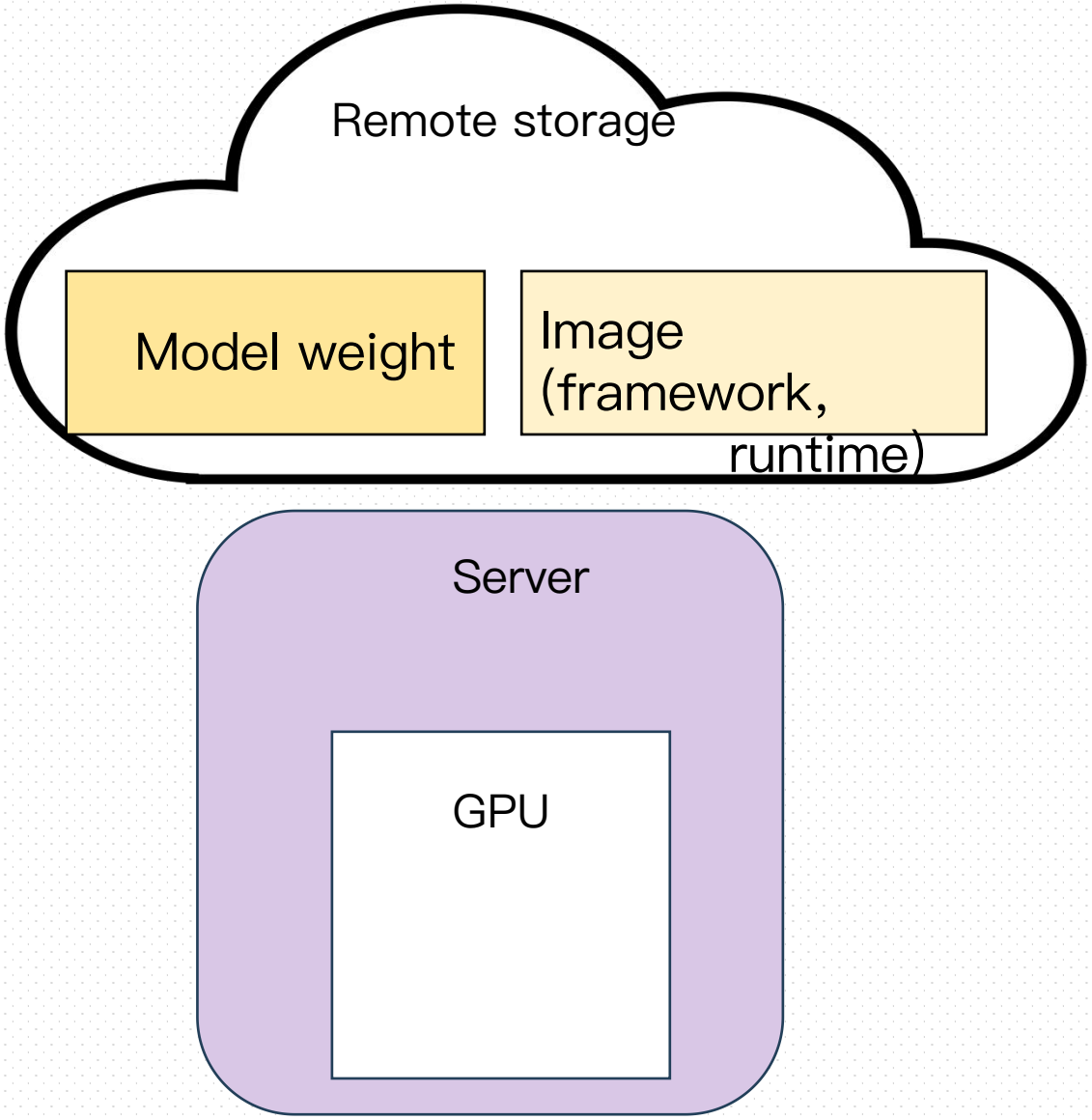


Background: LLM

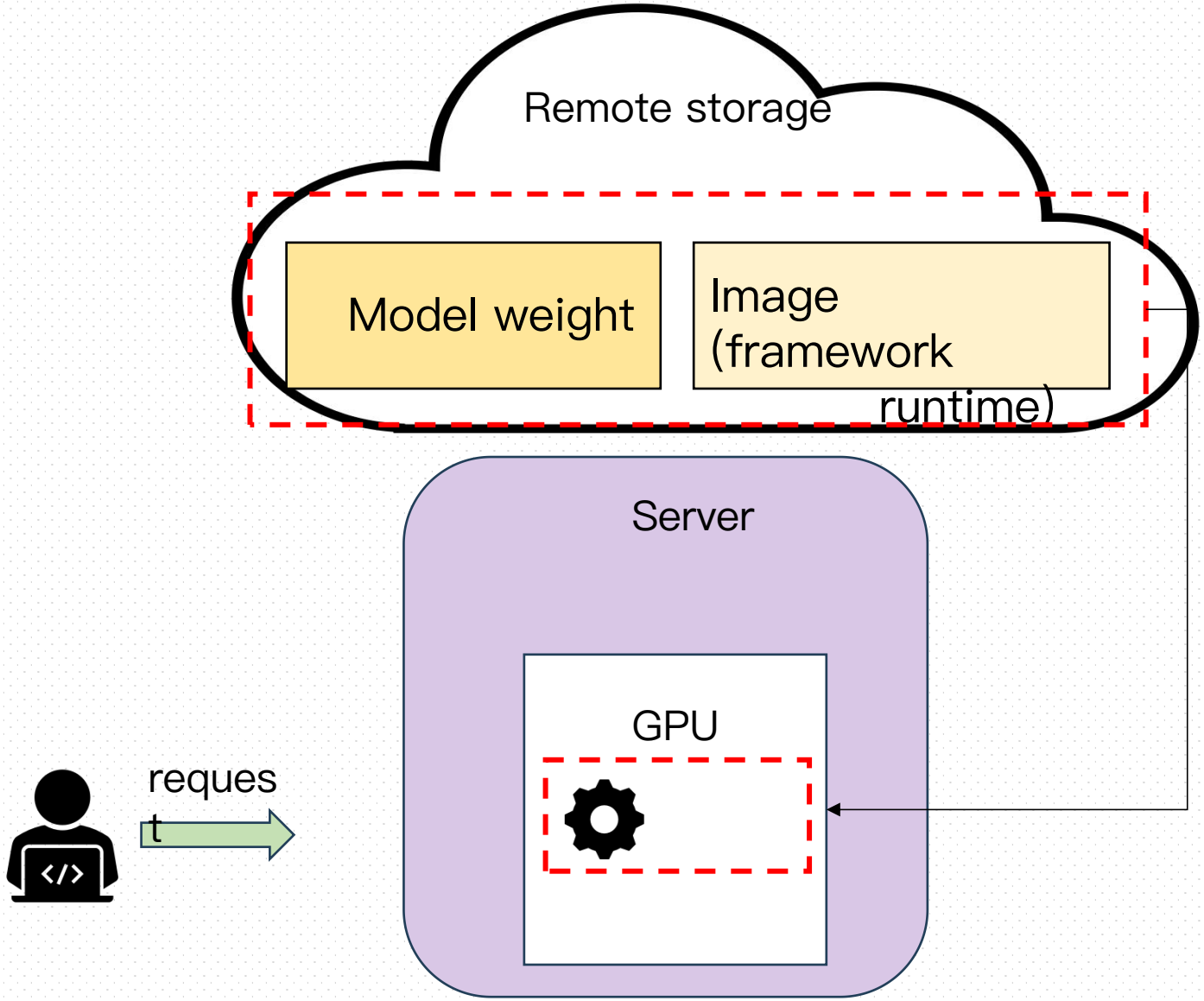
- Autoregressive manner
- Prefill & decode
- Service level objectives (SLOs)
  - Time to first token (TTFT)
  - Time per output token (TPOT)
- Pipeline parallelism in training



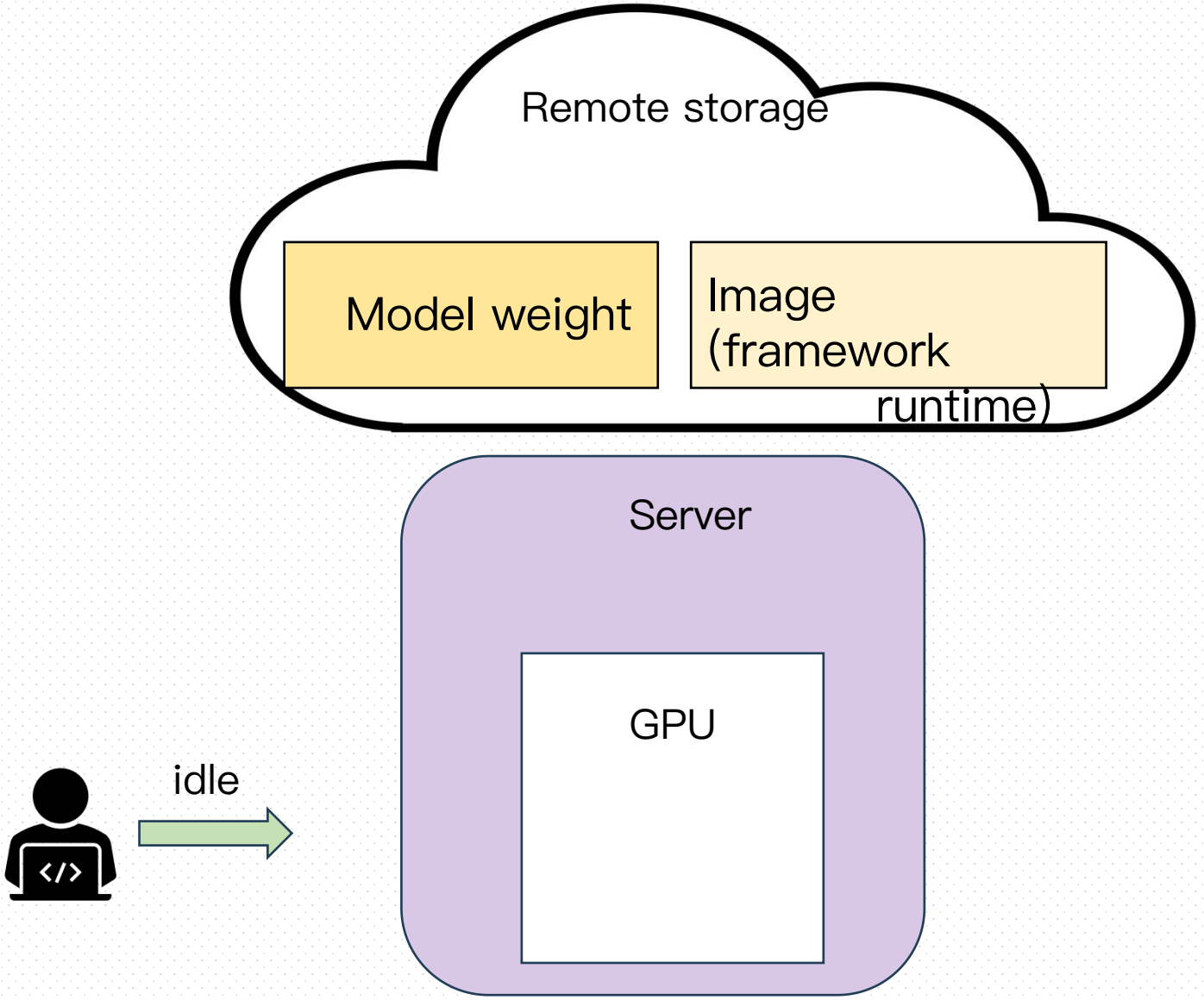
Background: Serverless LLM Serving



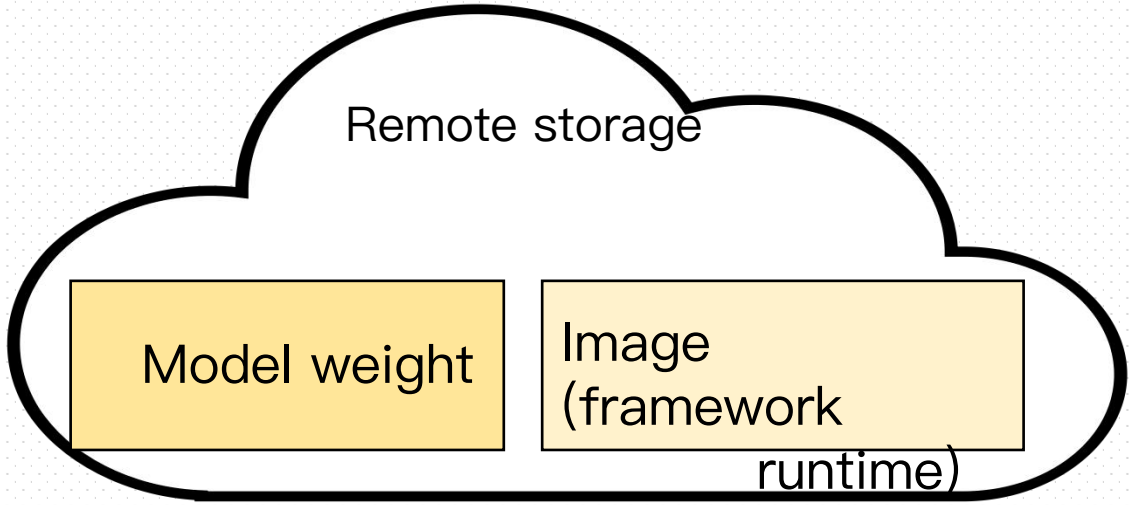
Background: Serverless LLM Serving



Background: Serverless LLM Serving

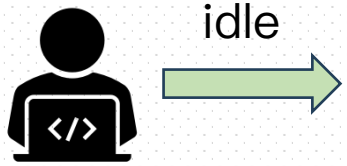


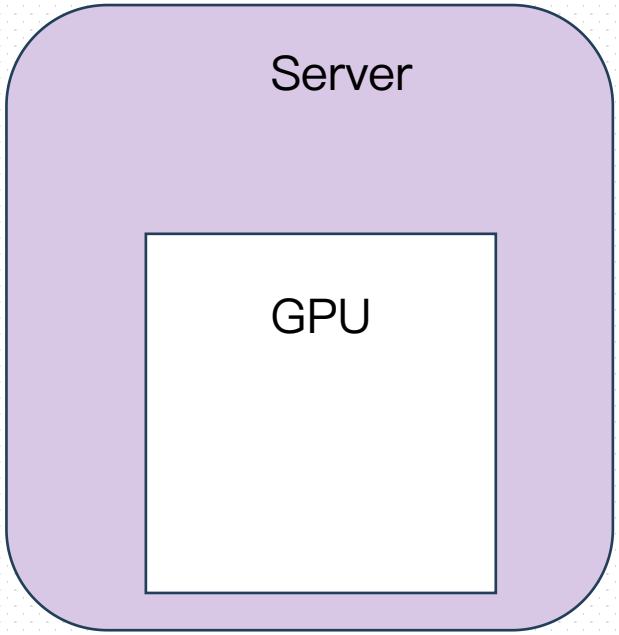
Background: Serverless LLM Serving



For user : Pay-per-use billing

For production : long-tail models





Background: Serverless LLM Serving

- **Network bandwidth constraints**
  - **Sharing among multiple instances**

Background: Serverless LLM Serving

- **Network bandwidth constraints**
  - **Sharing among multiple instances**
  - **Cost-efficiency requirement**

User
- GPU capabilities is important
- Prioritize cost saving

Provider Minimize cost per GPU

Cost of AWS EC2 instance (L40S GPU)

Instance	Mem.(GB)	Band.(Gbps)	#GPU	Cost($/h)	Cost/GPU($/h)
g6e.xlarge	32	up to 20	1	1.861	1.861
g6e.2xlarge	64	up to 20	1	2.24208	2.24208
g6e.4xlarge	128	20	1	3.00424	3.00424
g6e.8xlarge	256	25	1	4.52856	4.52856
g6e.16xlarge	512	35	1	7.57719	7.57719
g6e.12xlarge	384	100	4	10.49264	2.62316
g6e.24xlarge	768	200	4	15.06559	3.76640
g6e.48xlarge	1536	400	8	30.13118	3.76640

~120 %

~400 %

Background: Serverless LLM Serving

❑Network bandwidth constraints

❖Sharing among multiple instances

❖Cost-efficiency requirement

User
- GPU capabilities is important
- Prioritize cost saving

Provider Minimize cost per GPU

Cost of AWS EC2 instance (L40S GPU)

Instance	Mem.(GB)	Band.(Gbps)	#GPU	Cost($/h)	Cost/GPU($/h)
g6e.xlarge	32	up to 20	1	1.861	1.861
g6e.2xlarge	64	up to 20	1	2.24208	2.24208
g6e.4xlarge	128	20	1	3.00424	3.00424
g6e.8xlarge	256	25	1	4.52856	4.52856
g6e.16xlarge	512	35	1	7.57719	7.57719
g6e.12xlarge	384	100	4	10.49264	2.62316
g6e.24xlarge	768	200	4	15.06559	3.76640
g6e.48xlarge	1536	400	8	30.13118	3.76640

~120 %

~400 %

☹ Cold start latency is long

☺ Reduce CPU, memory and network resources

Background: Serverless LLM Serving

- **Network bandwidth constraints**
  - **Sharing among multiple instances**
  - **Cost-efficiency requirement**

User
- GPU capabilities is important
- Prioritize cost saving

Provider Minimize cost per GPU

Cost of AWS EC2 instance (L40S GPU)

Instance	Mem.(GB)	Band.(Gbps)	#GPU	Cost($/h)	Cost/GPU($/h)
g6e.xlarge	32	up to 20	1	1.861	1.861
g6e.2xlarge	64	up to 20	1	2.24208	2.24208
g6e.4xlarge	128	20	1	3.00424	3.00424
g6e.8xlarge	256	25	1	4.52856	4.52856
g6e.16xlarge	512	35	1	7.57719	7.57719
g6e.12xlarge	384	100	4	10.49264	2.62316
g6e.24xlarge	768	200	4	15.06559	3.76640
g6e.48xlarge	1536	400	8	30.13118	3.76640

~120 %

~400 %

☹ Cold start latency is long

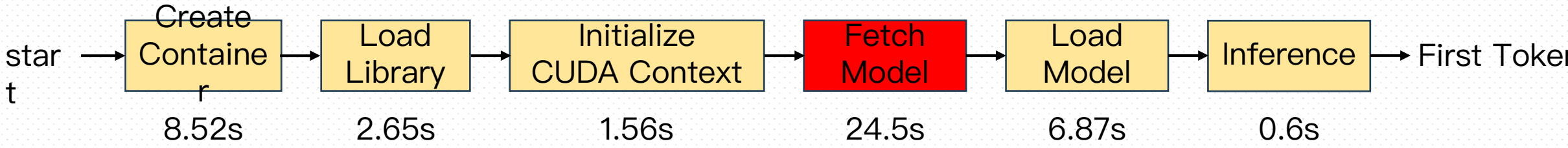
☺ Reduce CPU, memory and network resources

Trade–off between cold start latency & cost benefits

Background: Serverless LLM Serving

- **Network bandwidth constraints**
- **Breakdown of cold starts**

Llama2–7B on A10 GPU using vLLM



- **Model fetching :** From remote storage to local memory
- **Model loading :** Transfer weights to GPU; CUDA graph & KV cache init

~2s

Observation & Opportunities

Overlapping!

- Model fetching can be parallelism
  - Each worker only host a part of model

Observation & Opportunities

Overlapping!

- Model fetching can be parallelism
  - Each worker only host a part of model
- Model loading and library loading use different resource
  - The two phases can be parallelism

Observation & Opportunities

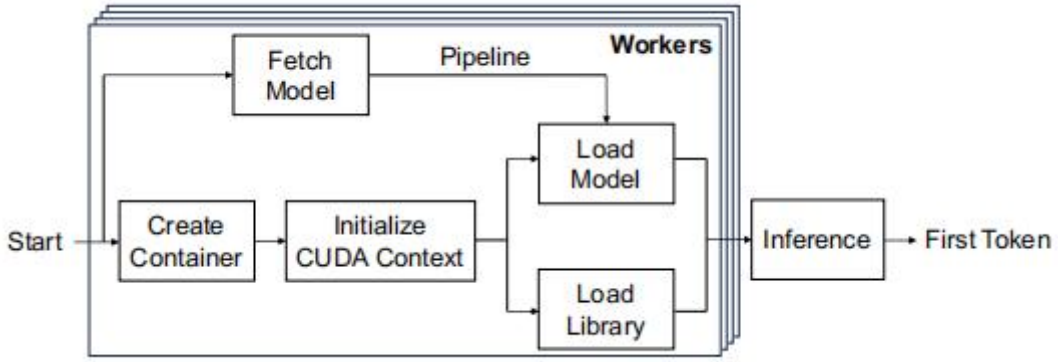
Overlapping!

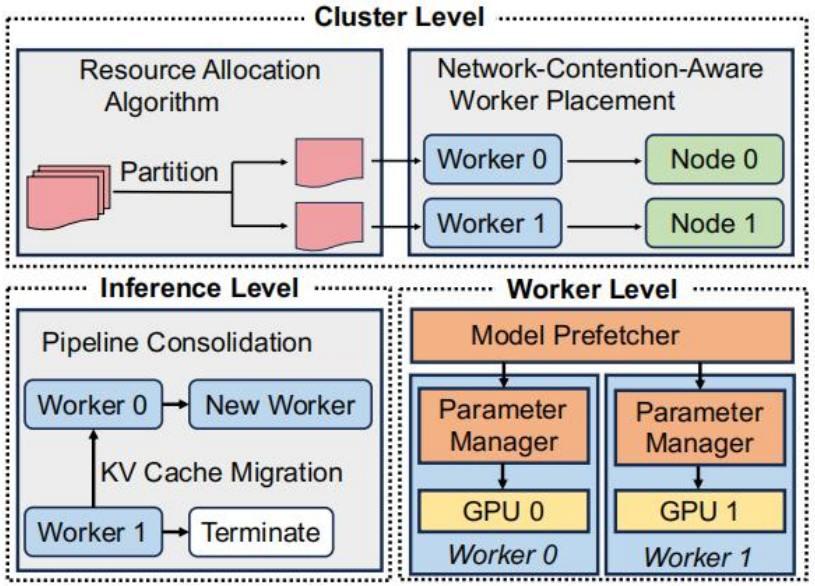
- Model fetching can be parallelism
  - Each worker only host a part of model
- Model loading and library loading use different resource
  - The two phases can be parallelism
- Model fetching and loading can be pipelined
  - At tensor granularity to hide overhead

Observation & Opportunities

Overlapping!

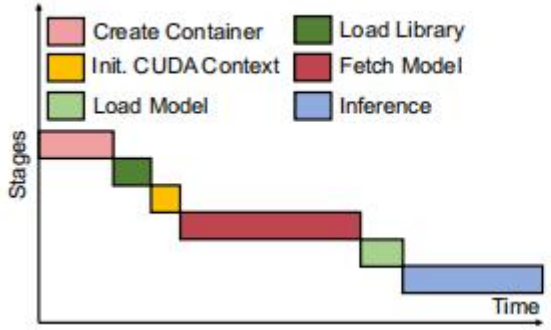
- Model fetching can be parallelism in different worker
- Parallel Model loading and library loading inside same worker
- Model fetching and loading pipelined at tensor granularity



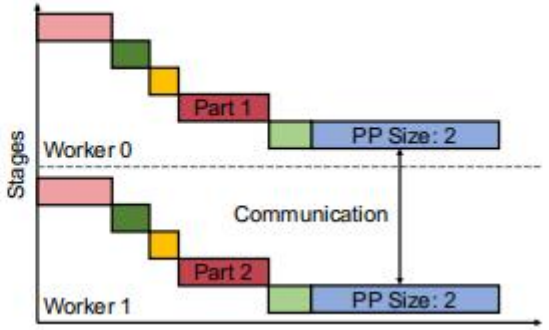


HydraServe: Cluster-Level

- **Cluster-level overall design**
  - **Multiple workers on different servers**
  - **Each worker fetches a part of model**
  - **Workers exchange intermediate results during inference**



Normal cold start



Cold start with 2 workers

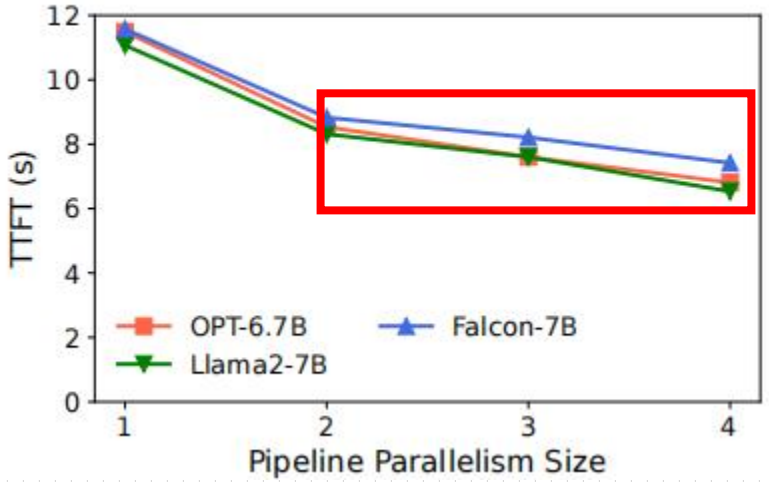
HydraServe: Cluster-Level

❑Trade-off analysis

Setup : 4 servers (A10, 188GB memory, 16 Gbps network bandwidth)

❖Larger parallelism sizes reduce model fetching time

➢Other stage also matters

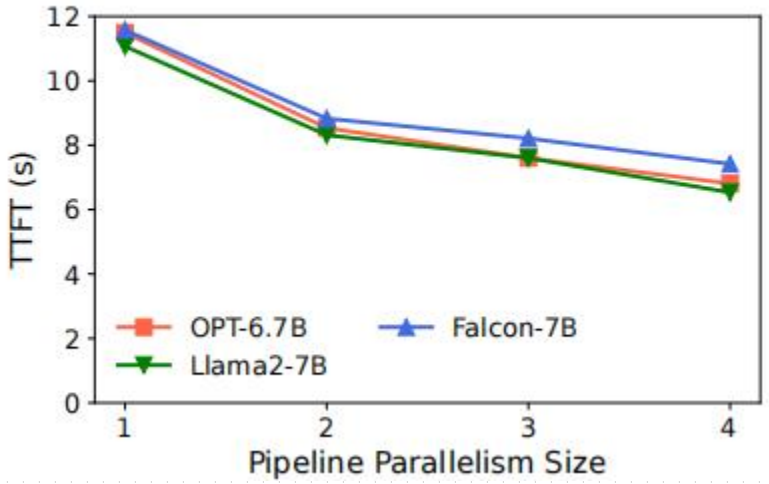


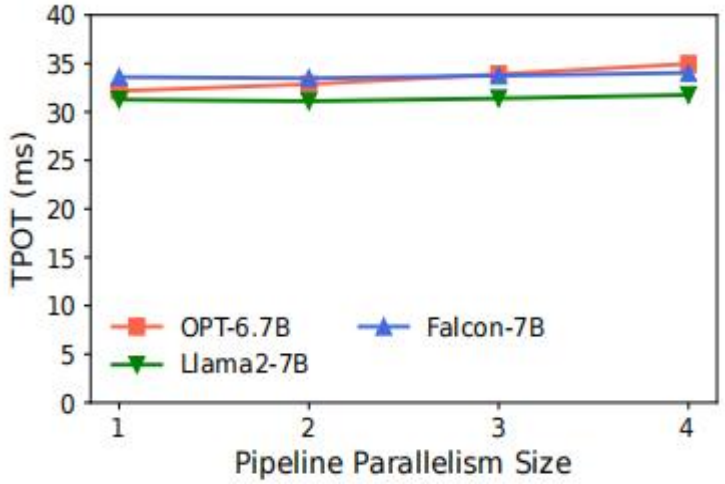
HydraServe: Cluster-Level

❑Trade-off analysis

Setup : 4 servers (A10, 188GB memory, 16 Gbps network bandwidth)

- ❖Larger parallelism sizes reduce model fetching time
- ❖Pipeline parallelism has a modest impact on TPOT
  - ➢8 KB of inter-layer results per token in Llama2-7B





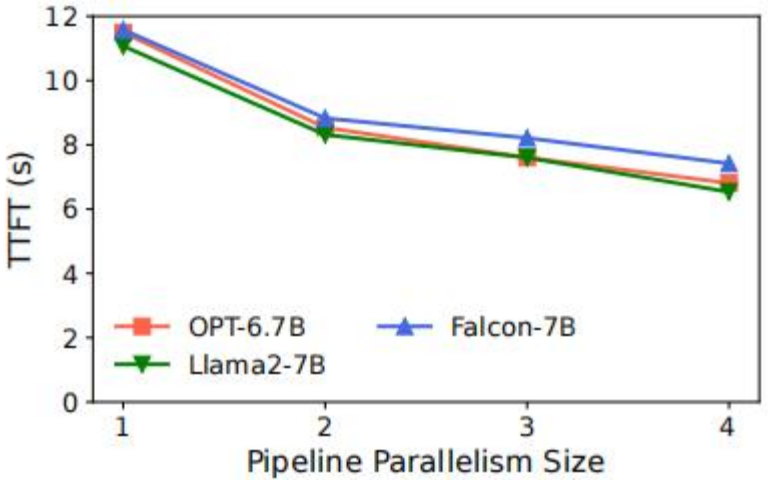
HydraServe: Cluster-Level

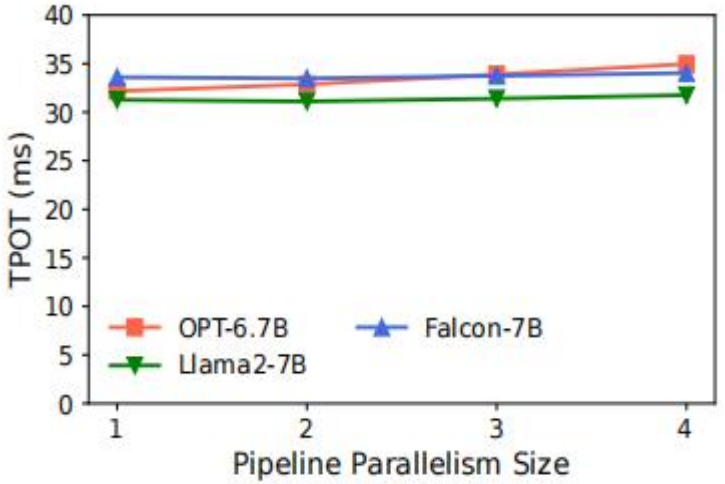
❑Trade-off analysis

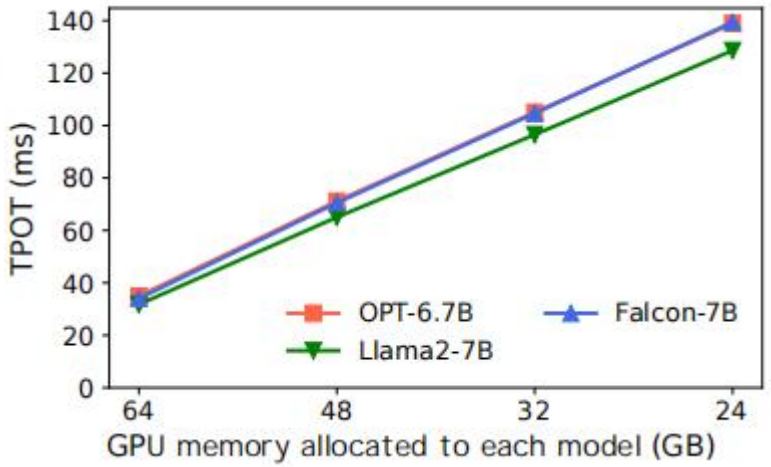
Setup : 4 servers (A10, 188GB memory, 16 Gbps network bandwidth)

- ❖Larger parallelism sizes reduce model fetching time
- ❖Pipeline parallelism has a modest impact on TPOT
- ❖Worker colocation leads to longer TPOT

pipeline parallelism size = 4







HydraServe: Cluster-Level

- Algorithm design
  - Target
    - Satisfy TTFT & TPOT
    - Minimum GPU sharing
  - Search space
    - Pipeline parallelism size (s)
      - When s > 4, yield little improvement

HydraServe: Cluster-Level

❑Algorithm design

❖Target

- ➢Satisfy TTFT & TPOT
- ➢Minimum GPU sharing

❖Search space

- ➢Pipeline parallelism size ($1 \leq s \leq 4$)
- ➢GPU memory to each worker
  - The same as non-parallelized (M)
  - The minimal memory required ($\frac{M}{s}$)

HydraServe: Cluster-Level

❑Algorithm design

❖Target

- ➢Satisfy TTFT & TPOT
- ➢Minimum GPU sharing

❖Search space

- ➢Pipeline parallelism size ($1 \leq s \leq 4$)
- ➢GPU memory to each worker
  - The same as non-parallelized (M)
  - The minimal memory required ($\frac{M}{s}$)
- ➢Number of full-memory workers (w)

HydraServe: Cluster-Level

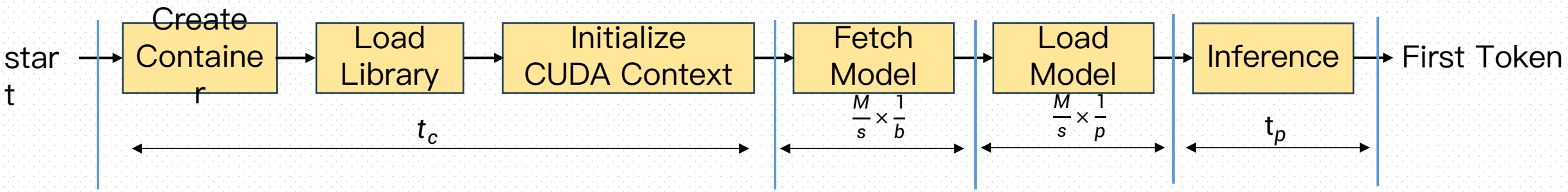
❑Algorithm design

❖Search space

- ➢Pipeline parallelism size (s)
- ➢Number of full-memory workers (w)

❖TTFT estimates

- ➢1 worker



HydraServe: Cluster-Level

❑Algorithm design

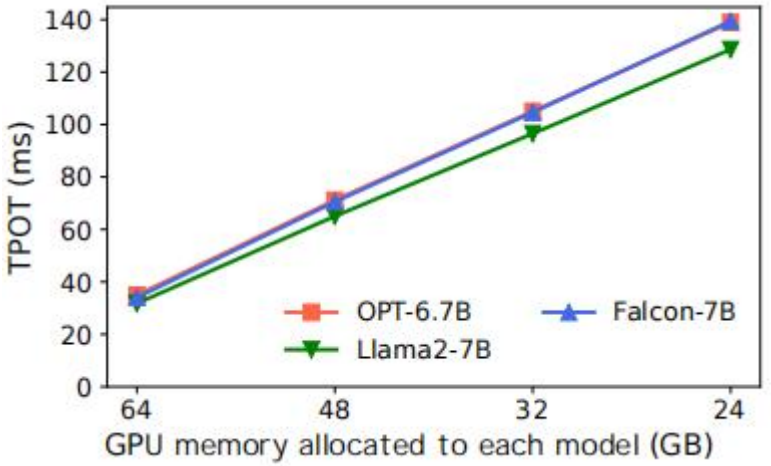
❖Search space

- ➢Pipeline parallelism size (s)
- ➢Number of full-memory workers (w)

❖TTFT estimates

- ➢1 worker
- ➢Inference time for each s workers
  - Data transmission : t_n
  - Prefill
    - ✓ Full-memory workers :$t_p \times \frac{1}{s}$
    - ✓ Low-memory workers : t_p

pipeline parallelism size = 4



HydraServe: Cluster-Level

❑Algorithm design

❖Search space

- ➢Pipeline parallelism size (s)
- ➢Number of full-memory workers (w)

❖TTFT estimates

- ➢1 worker
- ➢Inference time for each s workers
  - Data transmission : t_n
  - Prefill
    - ✓ Full-memory workers :$t_p \times \frac{1}{s}$
    - ✓ Low-memory workers : t_p

Total inference time is :

$$t_p \times \left(s - w + \frac{w}{s}\right) + t_n \times s$$

HydraServe: Cluster-Level

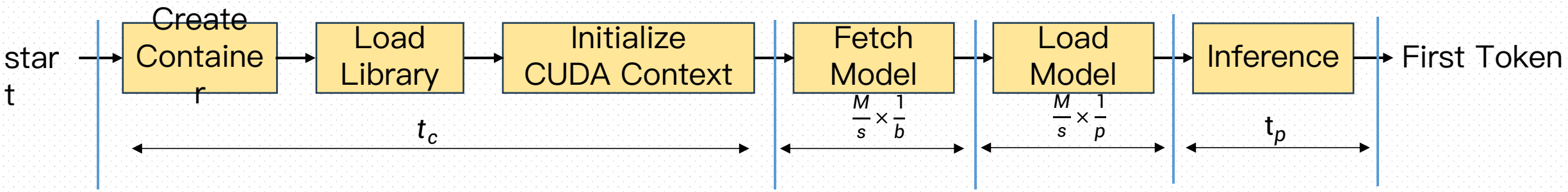
❑Algorithm design

❖Search space

- ➢Pipeline parallelism size (s)
- ➢Number of full-memory workers (w)

❖TTFT estimates

- ➢1 worker



HydraServe: Cluster-Level

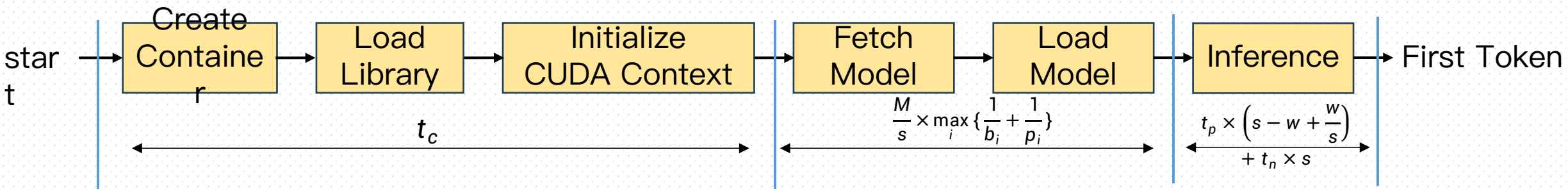
❑Algorithm design

❖Search space

- ➢ Pipeline parallelism size (s)
- ➢ Number of full-memory workers (w)

❖TTFT estimates

- ➢ s workers



HydraServe: Cluster-Level

❑Algorithm design

❖Search space

- ➢ Pipeline parallelism size (s)
- ➢ Number of full-memory workers (w)

❖TTFT estimates

$$t_c + \frac{M}{s} \times \max_i \{\frac{1}{b_i} + \frac{1}{p_i}\} + t_p \times \left(s - w + \frac{w}{s}\right) + t_n \times s$$

HydraServe: Cluster-Level

❑Algorithm design

❖Search space

- ➢ Pipeline parallelism size (s)
- ➢ Number of full-memory workers (w)

❖TTFT estimates

$$t_c + \frac{M}{s} \times \max_i \{\frac{1}{b_i} + \frac{1}{p_i}\} + t_p \times \left(s - w + \frac{w}{s}\right) + t_n \times s$$

❖TPOT estimates

$$t_d \times \left(s - w + \frac{w}{s}\right) + t_n \times s$$

HydraServe: Cluster-Level

- **Algorithm design**
  - **Search space**
    - **Pipeline parallelism size (s)**
    - **Number of full-memory workers (w)**
  - **TTFT estimates**
  - **TPOT estimates**
  - **Server selection**
    - **Priority smallest** $\max_i \{\frac{1}{b_i} + \frac{1}{p_i}\}$

Algorithm 1 Resource Allocation Algorithm

Input: time cost of container creation and runtime initialization t_c, data transmission t_n, prefill t_p, and decoding t_d; model size M; GPU server network bandwidth b_i and PCIe bandwidth p_i; user specified requirements SLO_{TTFT} and SLO_{TPOT}.
Output: pipeline parallelism size s, #full-memory workers w, and selected GPU servers g.

$S \leftarrow \emptyset$
for $s \in \{1, 2, \cdots, 4\}$ **do**
  for $w \in \{0, 1, \cdots, s\}$ **do**
    $i_1, i_2, \cdots, i_k \leftarrow$ Servers that fit a model of size M.
    $j_1, j_2, \cdots, j_l \leftarrow$ Servers that fit a model of size M/s.
    $j'_1, \cdots, j'_\mu \leftarrow \text{MergeSort}((j_1, \cdots, j_l), (i_{w+1}, \cdots, i_k))$
    $g \leftarrow (i_1, i_2 \cdots, i_w, j'_1, \cdots, j'_{s-w})$
    $\text{max\_ratio} \leftarrow \max_{x \in g} \left(\frac{1}{b_x} + \frac{1}{p_x}\right)$
    $\text{TTFT} \leftarrow t_c + \frac{M}{s} \times \text{max\_ratio} + t_p \times (s - w + \frac{w}{s}) + t_n \times s$
    $\text{TPOT} \leftarrow t_d \times (s - w + \frac{w}{s}) + t_n \times s$
    if $\text{TTFT} \leq \text{SLO}_{\text{TTFT}}$ **and** $\text{TPOT} \leq \text{SLO}_{\text{TPOT}}$ **then**
      $S \leftarrow S \cup \{(s, w, g)\}$
if S is $\emptyset$ **then**
  return $(1, 1, (i_1))$ ▷ Use single worker if no solution
else
  $c \leftarrow$ Scheme that incurs minimal GPU sharing from S
  return c

HydraServe: Cluster-Level

- **Network-Contention**
  - **Model fetching & Inference (small intermediate results)**
    - ☺ Prioritize inference packets

HydraServe: Cluster-Level

- Network-Contention
  - Model fetching & Inference (small intermediate results)
  - Workers fetch models on the same GPU

HydraServe: Cluster-Level

❑Network-Contention

❖Model fetching & Inference (small intermediate results)

❖Workers fetch models on the same GPU

A GPU server with N workers comes a new one

- ➢ **Bandwidth estimates :** $\frac{B}{N+1}$
- ➢ **Worker_i time to fetch the pending model (S_i) :** $T_{rest} = S_i \div \frac{B}{N+1}$

HydraServe: Cluster-Level

❑Network-Contention

❖Model fetching & Inference (small intermediate results)

❖Workers fetch models on the same GPU

A GPU server with *N* workers comes a new one

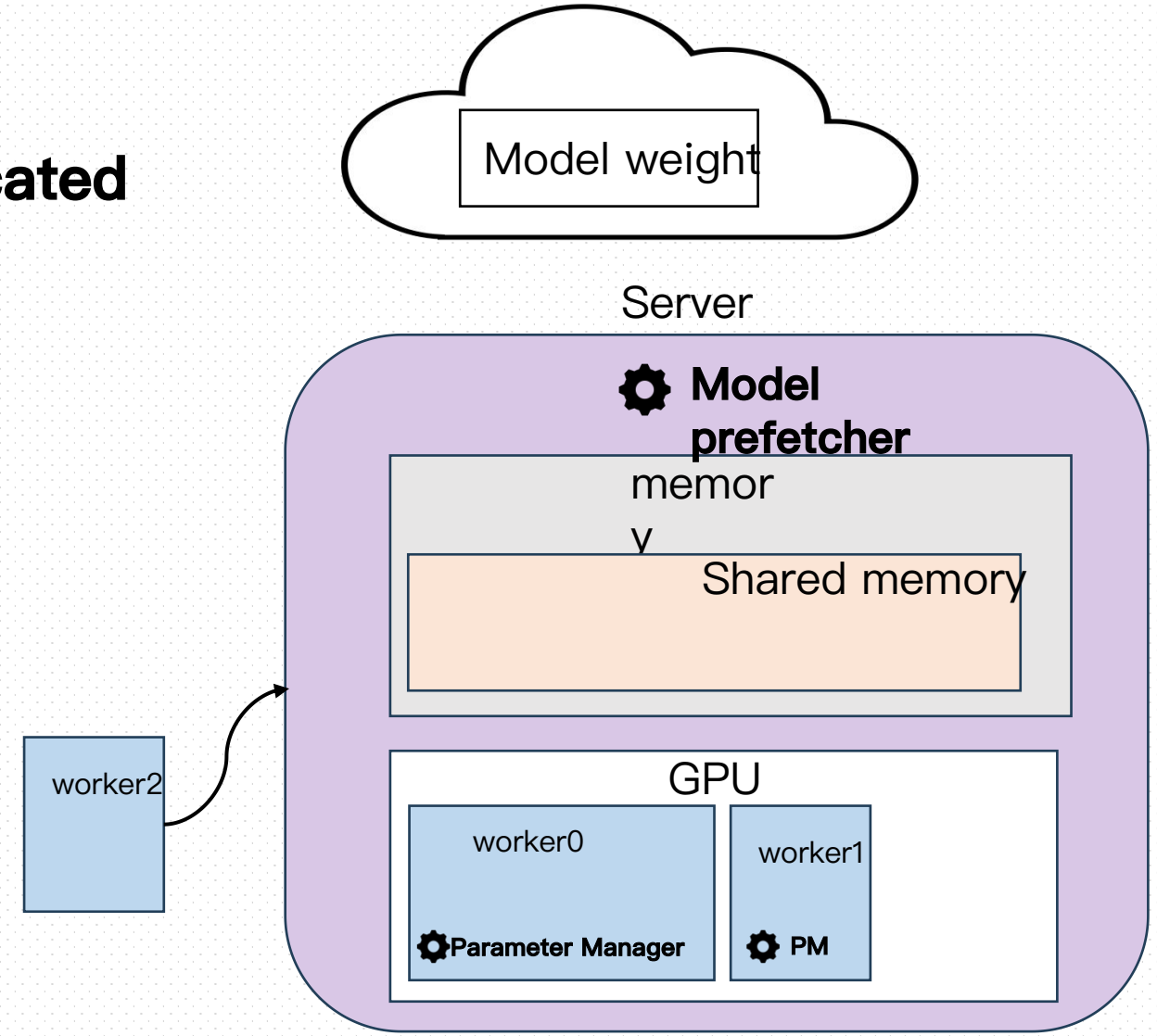
- **Bandwidth estimates :** $\frac{B}{N+1}$
- **Worker_i time to fetch the pending model (S_i) :** $T_{rest} = S_i \div \frac{B}{N+1}$
- S_i **estimates :** $S_i' = S_i - \frac{B}{N} \times (T_{curr} - T')$

Start or completion of a cold start time

We should meet : $T_{rest} \leq Deadline_i - T_{curr}$

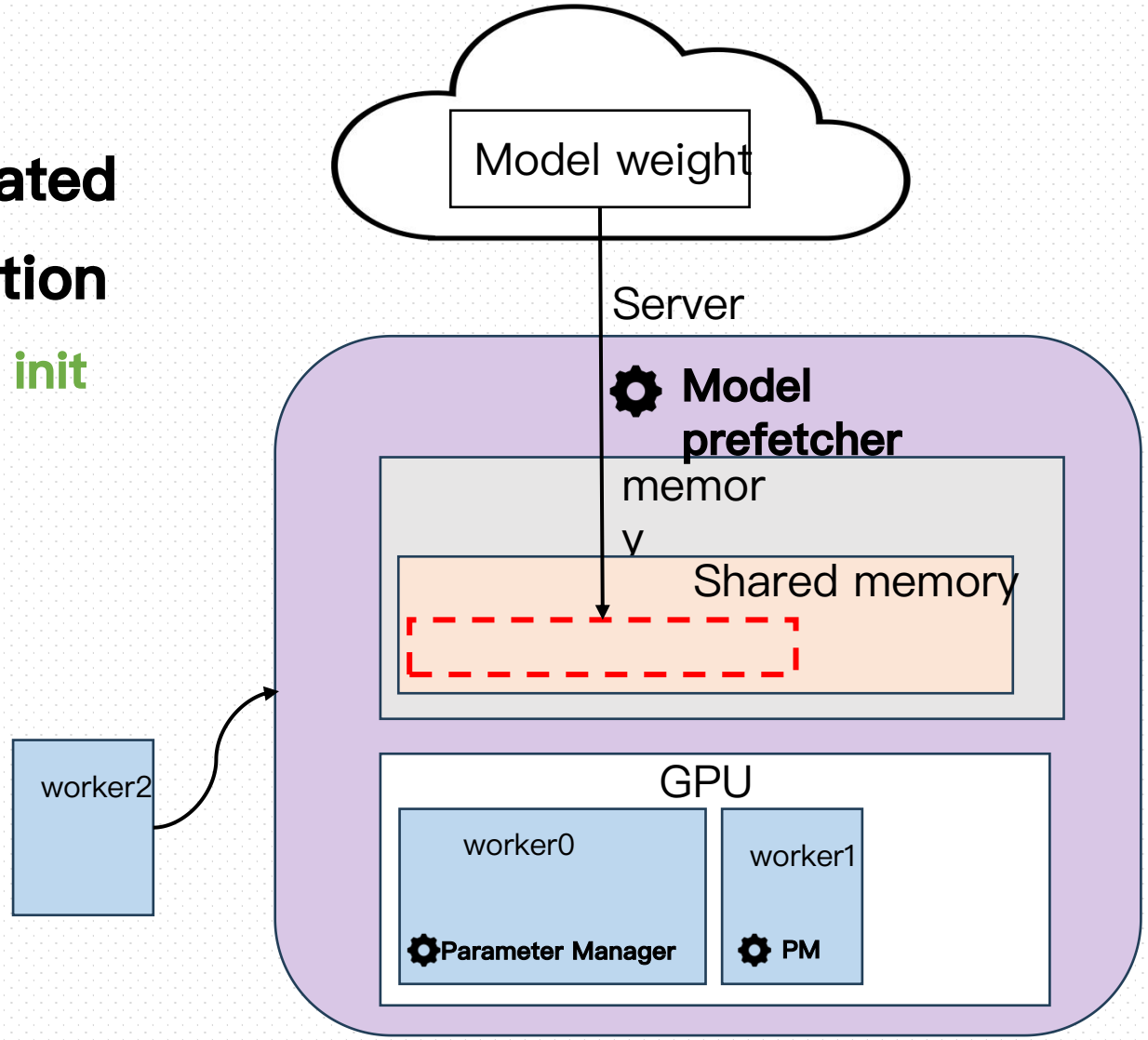
HydraServe: Worker-Level

- **Model prefetching**
  - **Shared memory is pre-allocated**
- **Parameter Manager**
  - **An individual thread**



HydraServe: Worker-Level

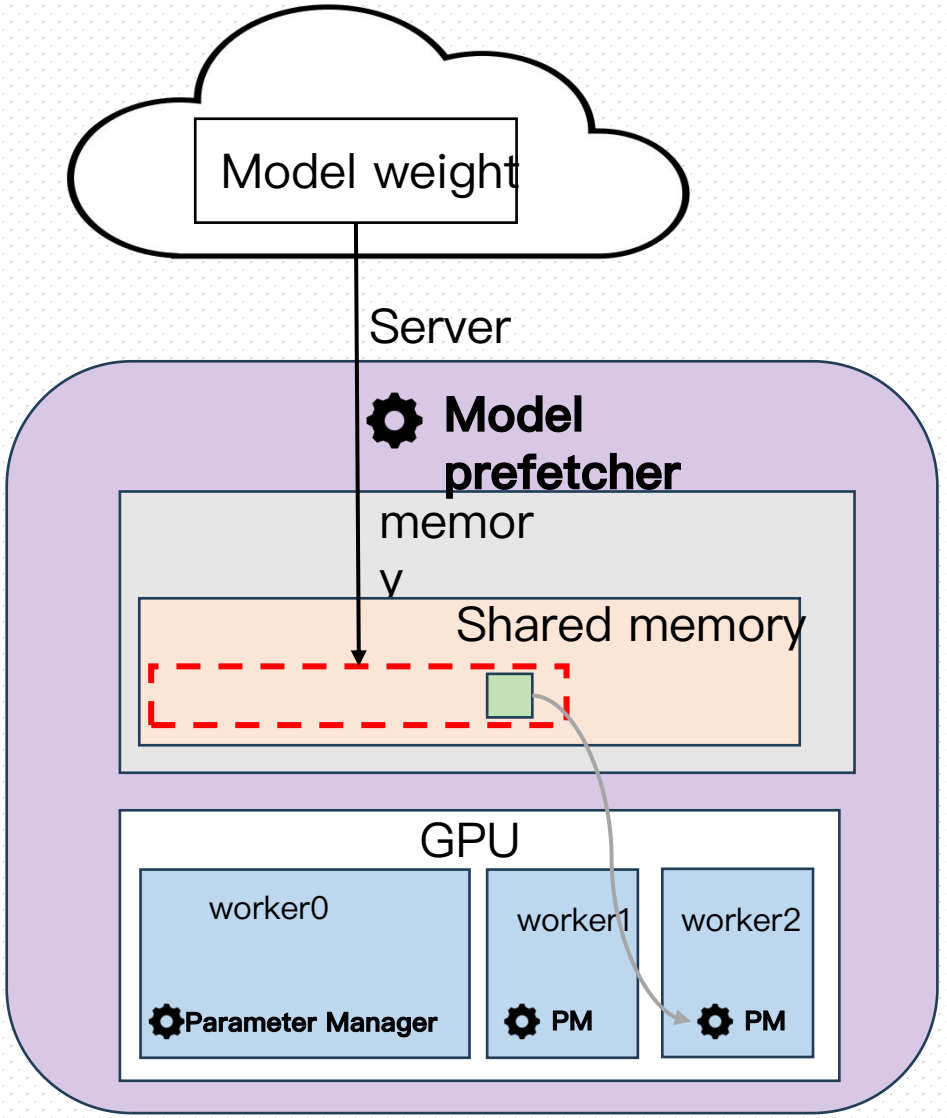
- Model prefetching
  - Shared memory is pre-allocated
  - Starts before container creation
    - Overlap container and runtime init
- Parameter Manager
  - An individual thread



HydraServe: Worker-Level

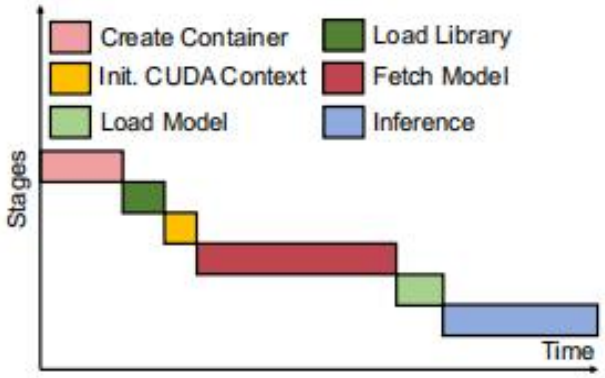
- **Model prefetching**
  - **Shared memory is pre-allocated**
  - **Starts before container creation**
    - Container and runtime init are overlapped
- **Parameter Manager**
  - **An individual thread**
  - **Container first init it**
  - **Streaming manner with zero-copy**

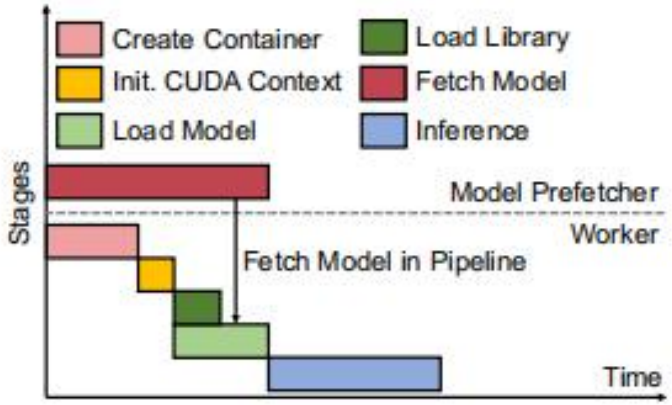
Model prefetch and load can be pipelined

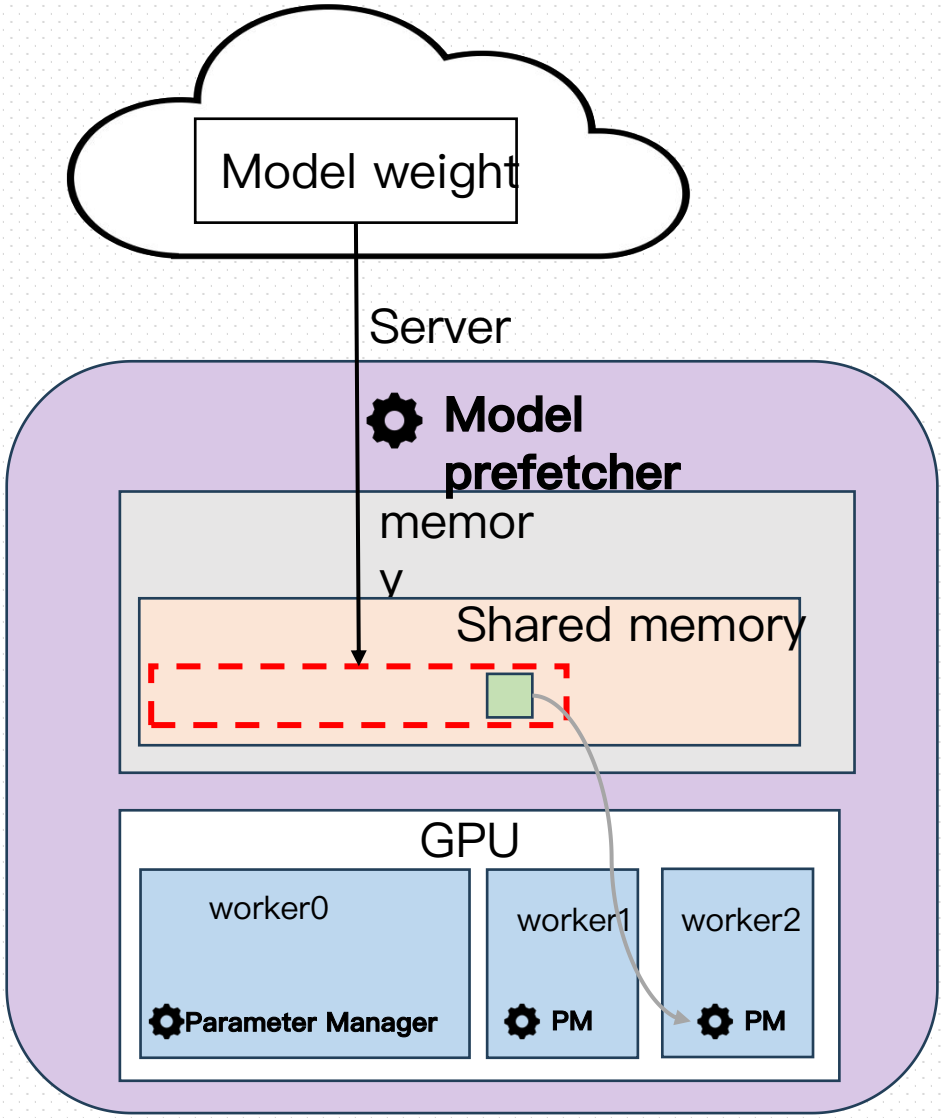


HydraServe: Worker-Level

- ❑Model prefetching
- ❑Parameter Manager
- ❑Prioritize CUDA context init
  - ❖Overlap library and model load







HydraServe: Inference-Level

❑Worker Scaling

Parameter manager continue to load the remind part

❖Scaling down

- ➢ Only one worker fetched unloaded model
- ➢ Default mechanism

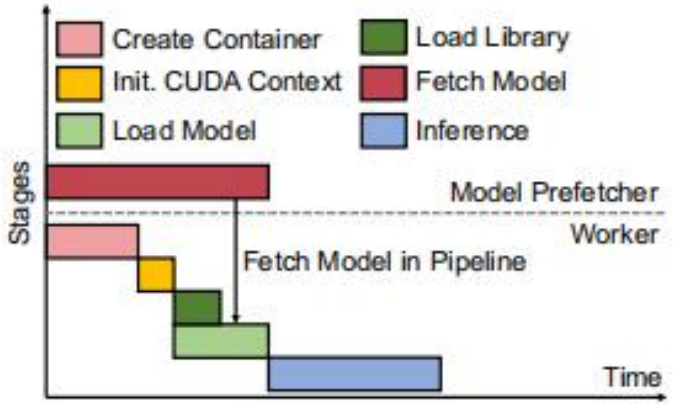
❖Scaling up

- ➢ All workers become individual serving endpoint
- ➢ Tackle load spikes
  - Use sliding window strategy to predict

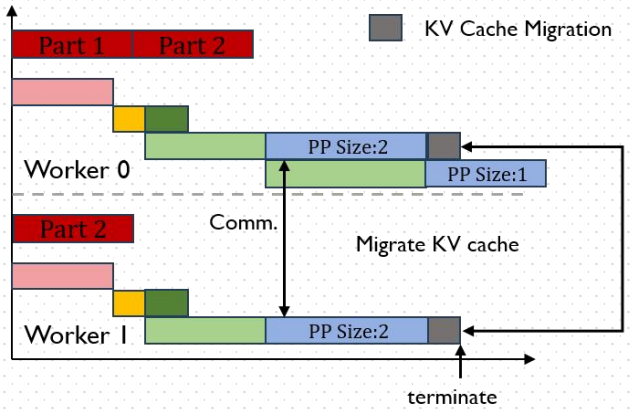
HydraServe: Inference-Level

❑Worker Scaling

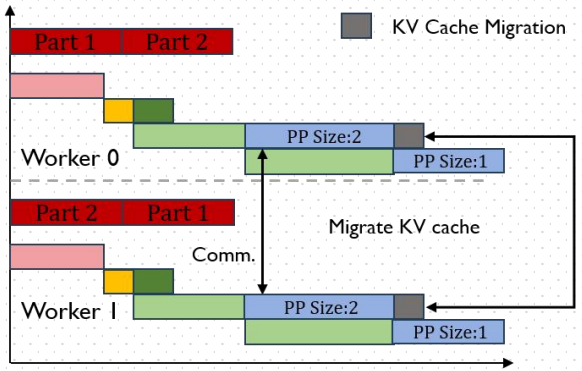
Origin: 1 worker



Scaling down



Scaling up

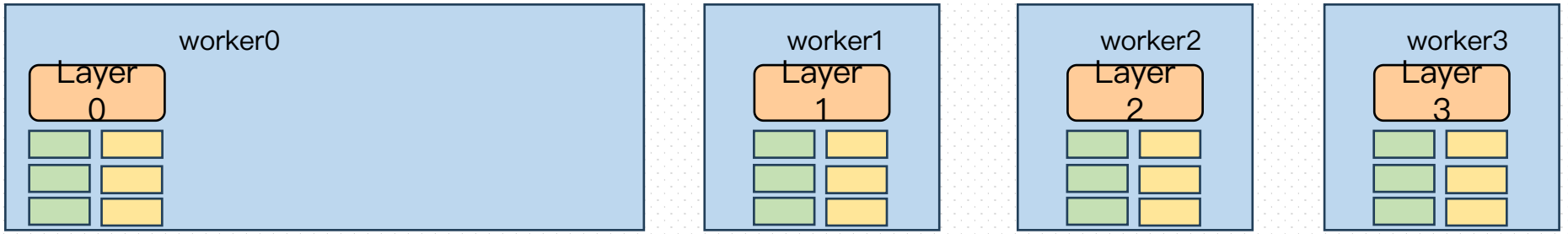


HydraServe: Inference-Level

- Worker Scaling
- KV cache migration
  - Stop scheduling & wait for on-the-fly batches
  - Gather the blocks
  - Place the block at different layers

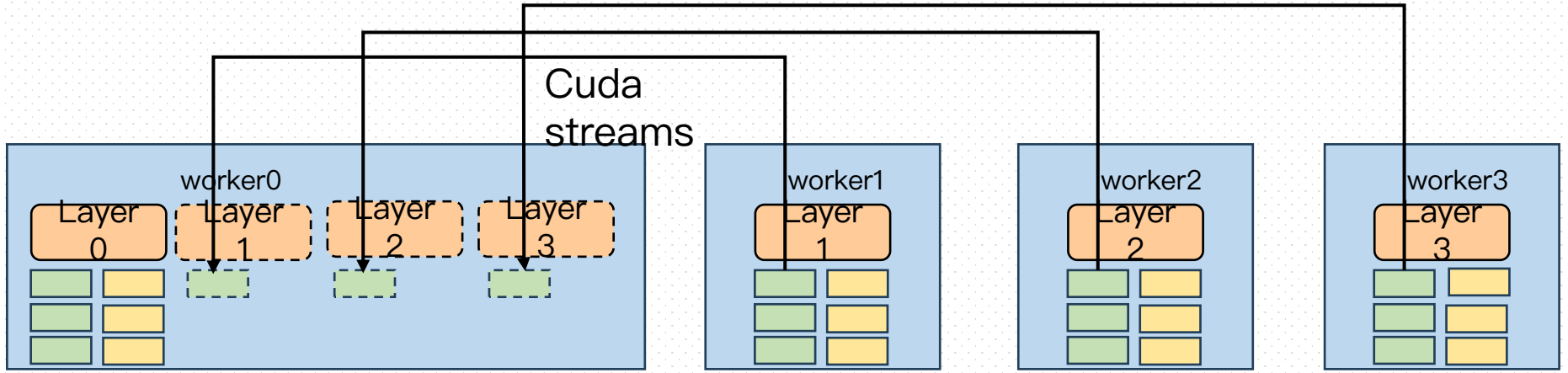
HydraServe: Inference-Level

- Worker Scaling
- KV cache migration
  - Stop scheduling & wait for on-the-fly batches
  - Gather the blocks
  - Place the block at different layers



HydraServe: Inference-Level

- Worker Scaling
- KV cache migration
  - Stop scheduling & wait for on-the-fly batches
  - Gather the blocks
  - Place the block at different layers



Implementation

- Instance startup optimizations in vLLM
  - Postpone KV Cache Allocation on CPU
  - Skip Online Memory Profiling
  - Direct GPU Tensor Usage (Zero-Copy)

Evaluation : Latency & Pipeline Consolidation

- Testbed
  - A cluster with 4 A10 servers & 4V100 servers
    - A10 server contains single A10 GPU and 188GiB memory
    - V100 server contains 4 V100 GPUs snd 368GiB memory
    - Network bandwidth per server is 16Gps

Evaluation : Latency & Pipeline Consolidation

❑Testbed

❖A cluster with 4 A10 servers & 4V100 servers

- ➢A10 server contains single A10 GPU and 188GiB memory
- ➢V100 server contains 4 V100 GPUs snd 368GiB memory
- ➢Network bandwidth per server is 16Gps
- ➢Remote storage has sufficient network capacity

❑Baselines

❖Servrless vLLM

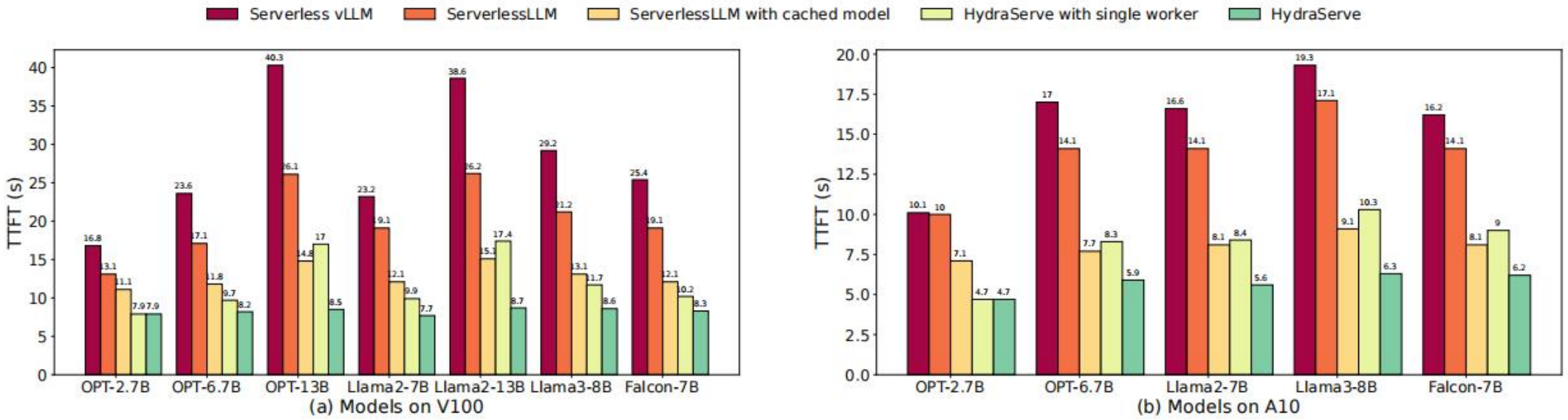
- ➢Iterates through all servers to select a available GPU

❖ServerlessLLM[1]

[1] ServerlessLLM: Low-Latency Serverless Inference for Large Language Models

Evaluation : Cold start Latency

❑Latency



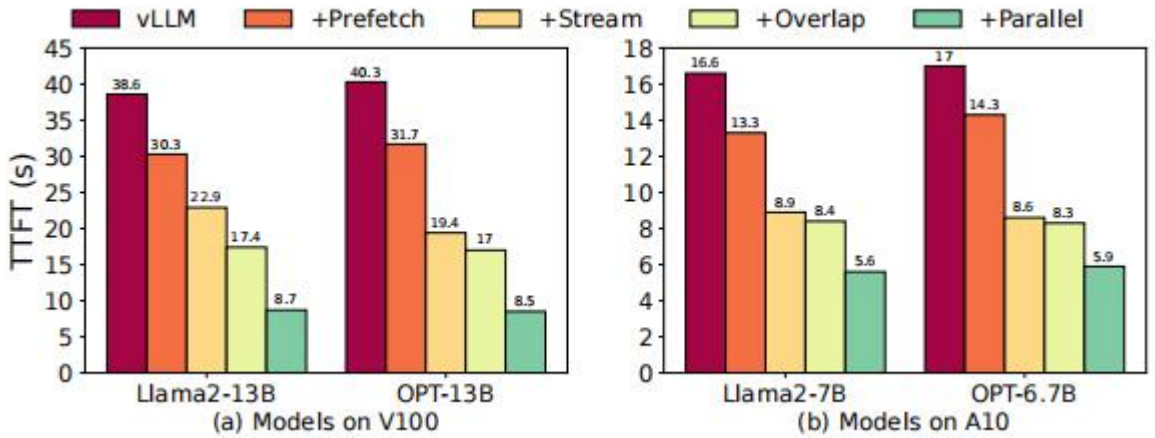
HydraServe reduces cold start latency by 2.1x–4.7x compared to serverless vLLM

1.7x–3.1x compared to serverlessLLM

Evaluation : Cold start Latency

❑Breakdown

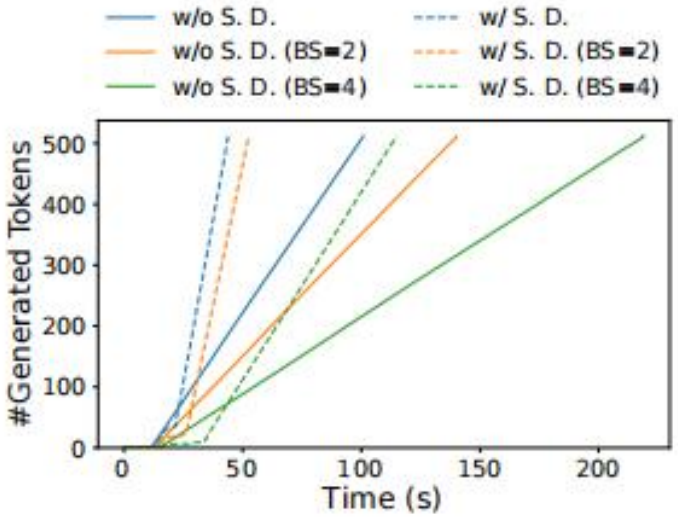
- ❖Prefetch : model prefetcher
- ❖Stream : streaming loading & implementation optimize
- ❖Overlap : overlap library & model loading
- ❖Parallel : parallelize the cold start



Evaluation : Pipeline Consolidation

Llama2-13B on V100, input length is 512 tokens

❑Scaling down



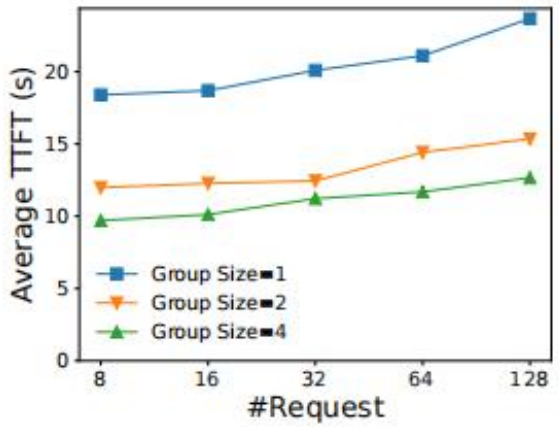
❖Reduce end-to-end generation time by 1. 61×—1.70×

❖Maintain almost same inference speed during early start

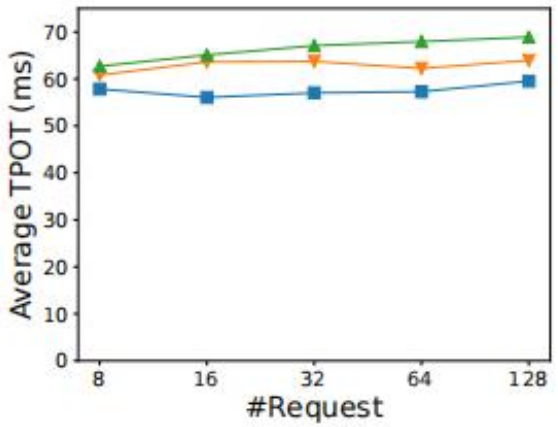
Evaluation : Pipeline Consolidation

Llama2-13B on V100, input length is 512 tokens

❑Scaling up



(a) Average TTFT of different loads.



(b) Average TPOT of different loads.

Maximum batch size : 8

❖128 concurrent requests, reduce average TTFT by 1. 87×

❖Average TPOT only increase by 1. 08×—1.19×

Evaluation : End-to-End

❑Testbed

- ❖A cluster with 2 A10 servers
  - ➢4 A10 GPUs , 752GiB memory , 64Gbps network bandwidth
- ❖A cluster with 4 V100 servers
  - ➢4 V100 GPUs , 368GiB memory , 16Gbps network bandwidth
- ❖Remote storage has sufficient network capacity

❑Baselines

- ❖Servrless vLLM
  - ➢Iterates through all servers to select a available GPU
- ❖ServerlessLLM[1]

[1] ServerlessLLM: Low-Latency Serverless Inference for Large Language Models

Evaluation : End-to-End

❑Workload

❖SLO based on warm requests (1024 input tokens, 8 batch sizes)

Model	Model Size	GPU Card	TTFT	TPOT
Llama2-7B	12.5GB	A10	1.5s	42ms
Llama2-13B	24.2GB	V100	2.4s	58ms

TTFT and TPOT of warm request

Application	TTFT	TPOT	Dataset
Chatbot Llama2-7B	7.5s	200ms	ShareGPT
Chatbot Llama2-13	12s	200ms	ShareGPT
Code Completion Llama2-7B	7.5s	84ms	HumanEval
Code Completion Llama2-13B	12s	116ms	HumanEval
Summarization Llama2-7B	15s	84ms	LongBench
Summarization Llama2-13B	24s	116ms	LongBench

SLOs summary

Evaluation : End-to-End

❑Workload

❖SLO based on warm requests (1024 input tokens, 8 batch sizes)

Model	Model Size	GPU Card	TTFT	TPOT
Llama2-7B	12.5GB	A10	1.5s	42ms
Llama2-13B	24.2GB	V100	2.4s	58ms

TTFT and TPOT of warm request

Application	TTFT	TPOT	Dataset
Chatbot Llama2-7B	7.5s	200ms	ShareGPT
Chatbot Llama2-13	12s	200ms	ShareGPT
Code Completion Llama2-7B	7.5s	84ms	HumanEval
Code Completion Llama2-13B	12s	116ms	HumanEval
Summarization Llama2-7B	15s	84ms	LongBench
Summarization Llama2-13B	24s	116ms	LongBench

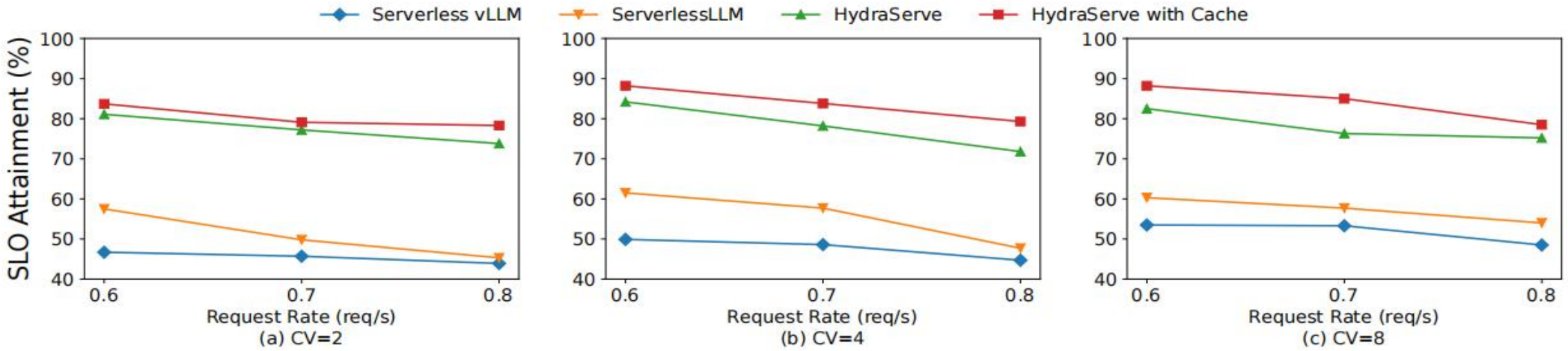
SLOs summary

❖Workloads

- **Microsoft Azure Function Trace**
  - **Models are mapped, round-robin approach**

Evaluation : End-to-End

❑Change the CV



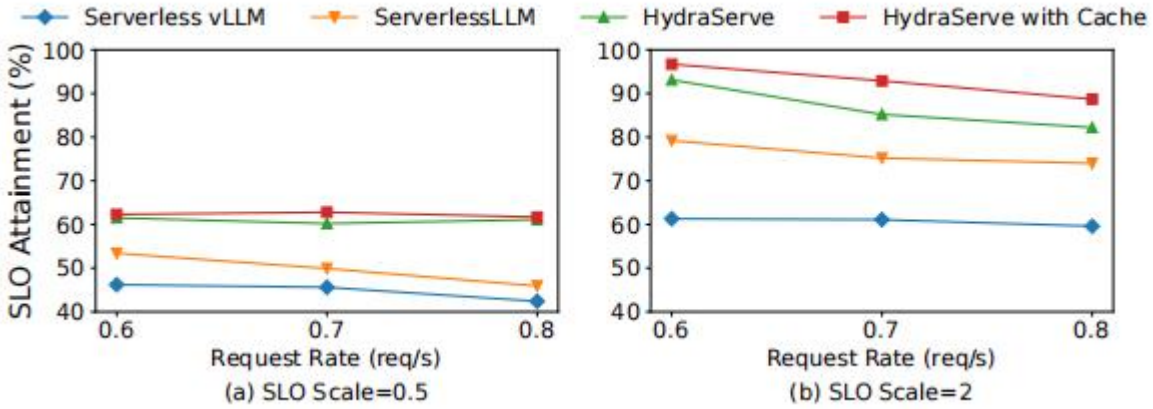
❖Achieve 1.43×—1.74× higher TTFT SLO during bursty requests

❖Caching further improve 1.11×

Evaluation : End-to-End

❑Change the SLO scales

CV fixed at 8



❖Tight SLO : Meet ~63% TTFT

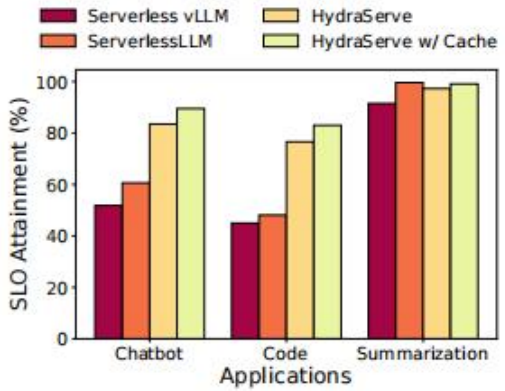
❖Loose SLO : Achieve 1.38×—1.52× improvement

1.49×—1.58× improvement with caching

Evaluation : End-to-End

❑Application analysis

CV = 8, RPS = 0.6

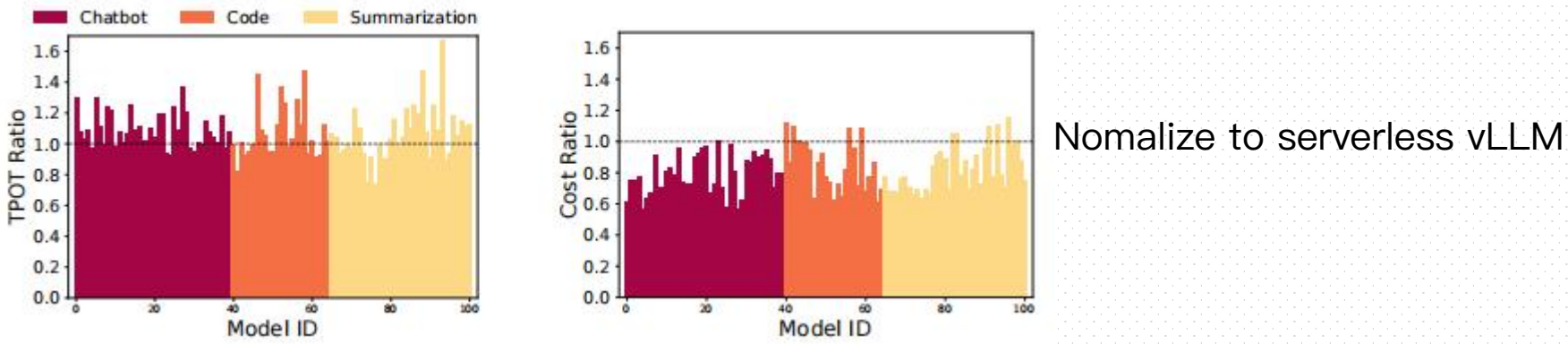


- ❖Chatbot & Code completion: 1. 61×—1.70× improvement in TTFT
- ❖Code completion has lower TTFT SLO attainment
  - ➢Workers keep alive for shorter time, cause more cold starts

Evaluation : End-to-End

❑TPOT and resource usage penalties

CV = 8, RPS = 0.6



Nomalize to serverless vLLM

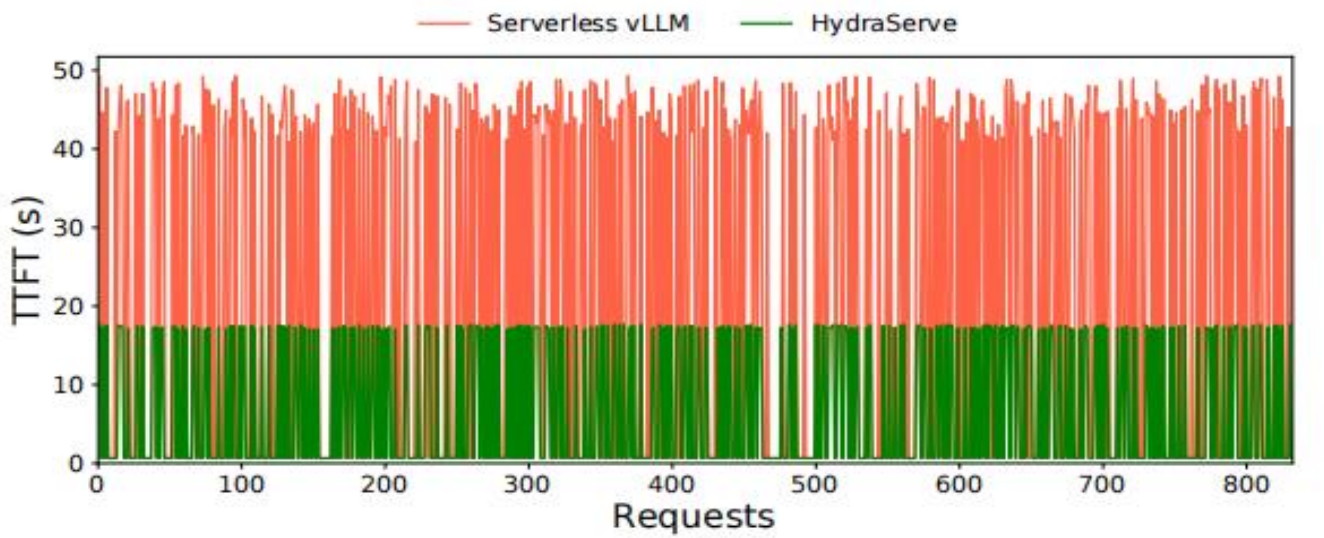
❖TPOT: A 1.06× average increase

❖Cost : HydraServe consumes lower in most case

- ➢ Pipeline groups merge is quickly
- ➢ Fast worker startup reduce GPU usage

Evaluation : Brownfield

Llama2-7B on NVIDIA A10 GPUs with 24GB GPU memory



❖HydraServe achieves an average 2.6× reduction in cold-start TTFT

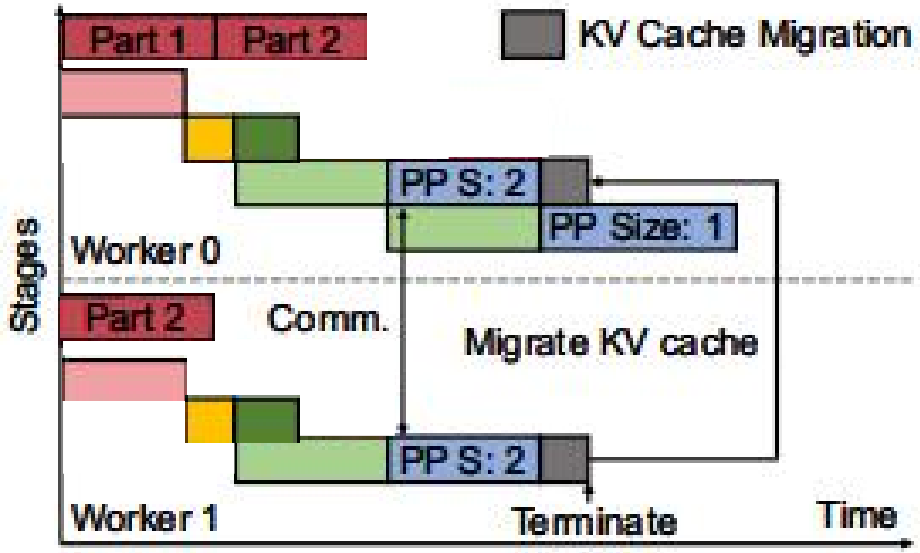
Conclusion

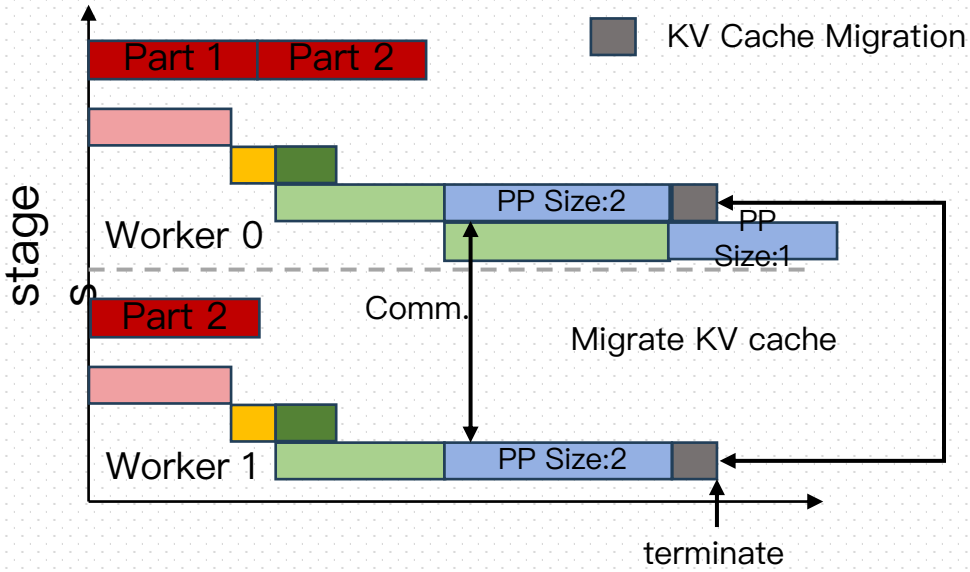
- **Pros:**
  - Use pipeline parallelism to reduce fetching time
  - Overlap load model and load library
  - Consolidate pipeline to reduce overhead

- **Cons:**
  - Remote storage needs sufficient bandwidth
  - Due to lack of NVLink, evaluated models can reside in a GPU

Q&A

Background: Out-of-order execution





HydraServe: Inference-Level

❑Worker Scaling

❖Scaling down