Appeared in USENIX OSDI 2026

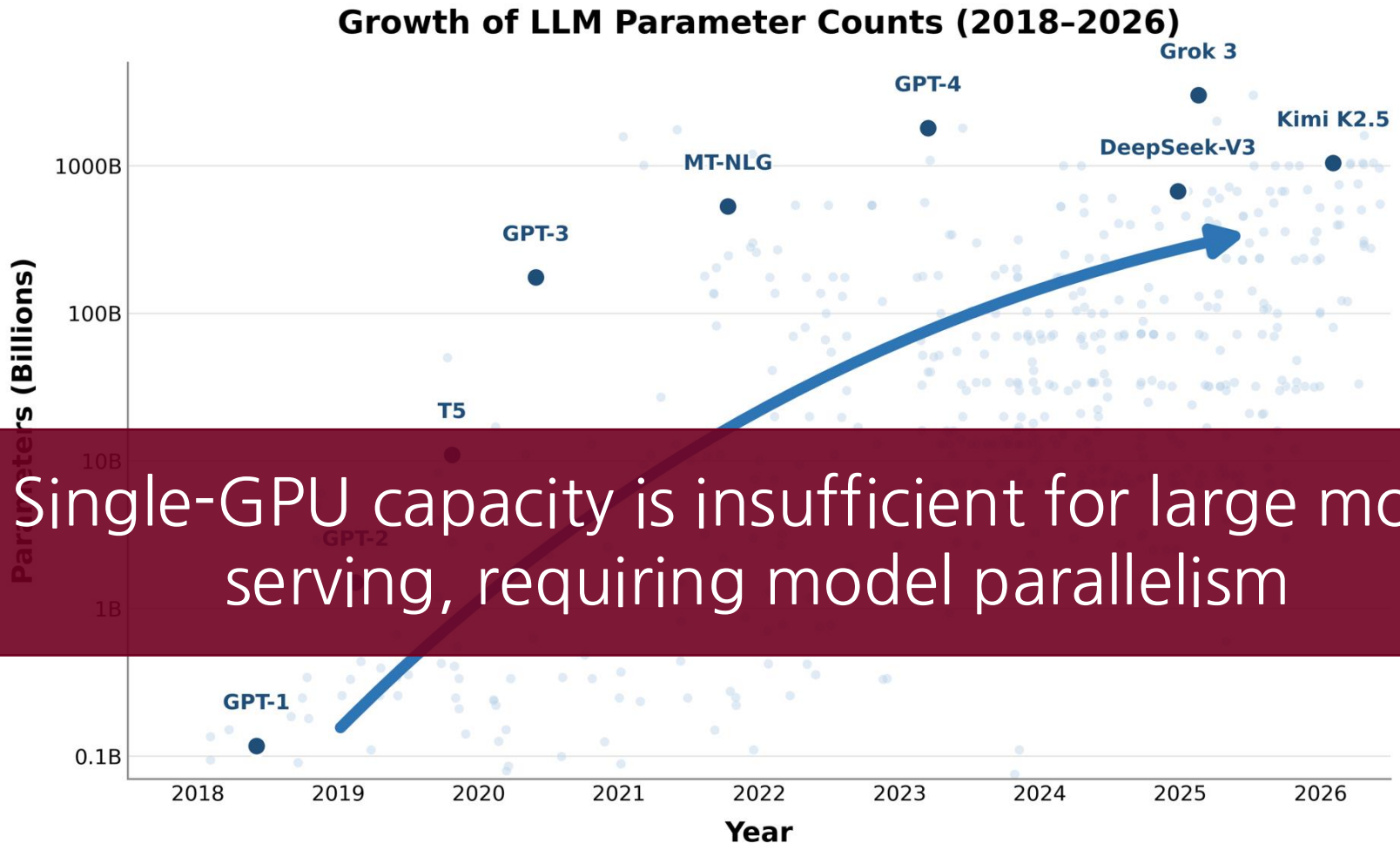
Revisiting Pipeline Parallelism for LLM Serving

Soonjae Hwang
황순재

Jeongseob Ahn
안정섭

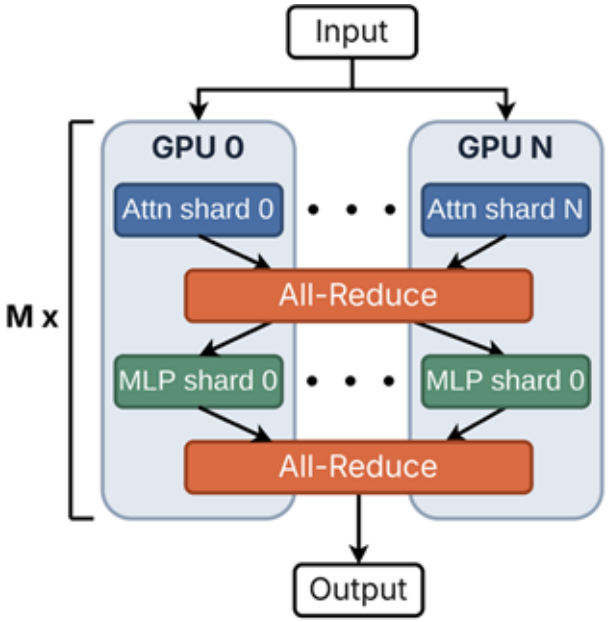
KOREA UNIVERSITY

Scaling LLMs strains inference systems



Single-GPU capacity is insufficient for large model serving, requiring model parallelism

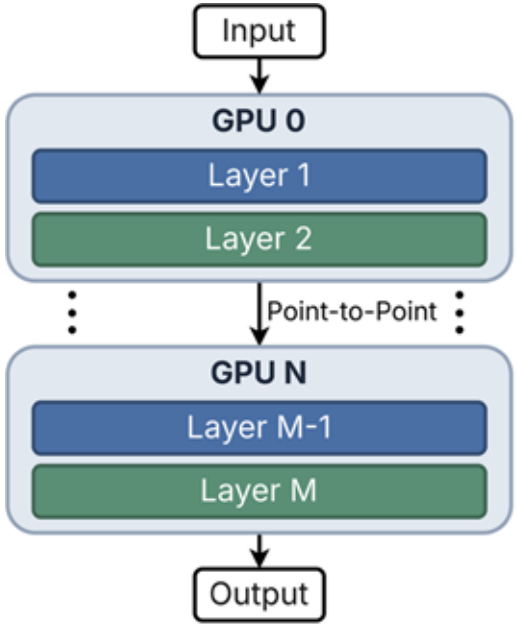
Tensor parallelism vs. pipeline parallelism



Tensor Parallelism (TP)

⏱ **[Low latency]**

⚠ **High-BW needed (e.g., NVLink)**



Pipeline Parallelism (PP)

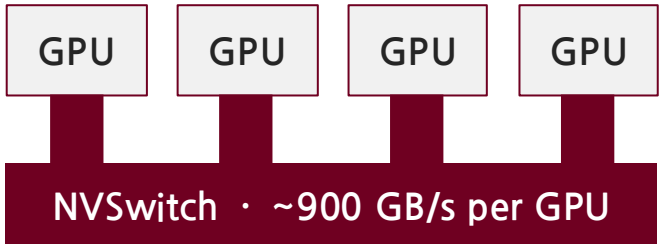
🚀 **[High throughput]**

✓ **Low-BW friendly (e.g., PCIe)**

Not every accelerator has an NVLink-class fabric

High-end GPU system (DGX / HGX)

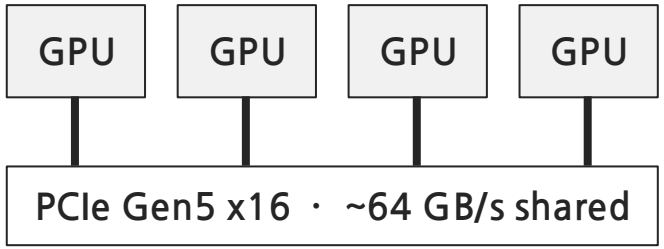
NVLink + NVSwitch, all-to-all



TP across 8 GPUs runs at full speed

Commodity PCIe GPU server

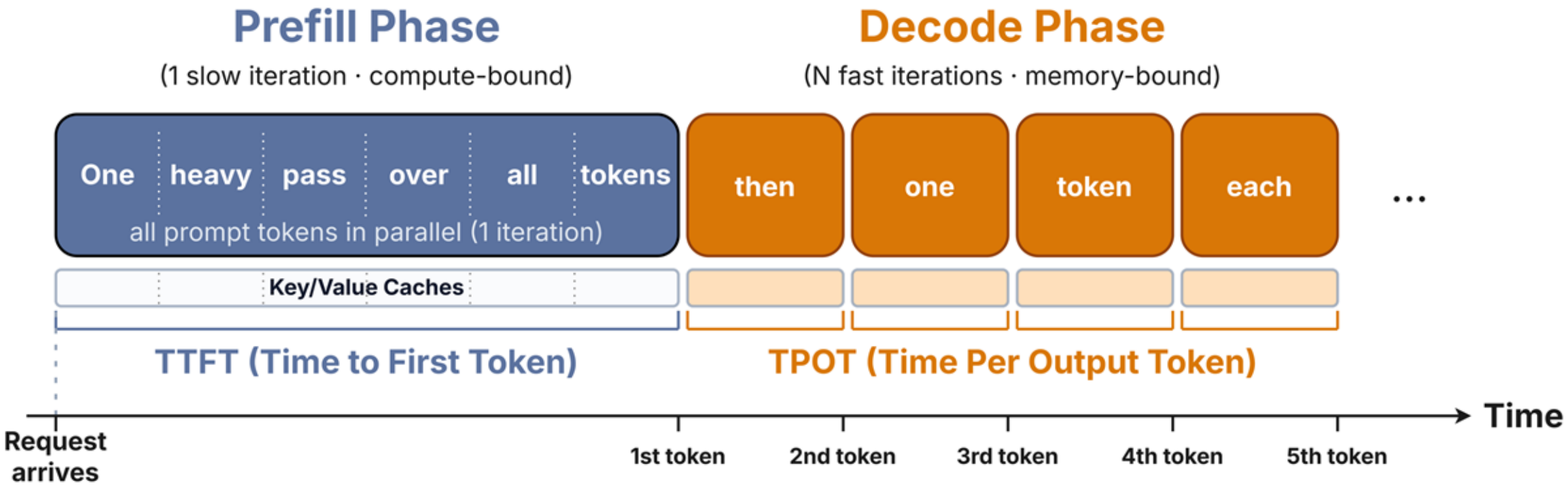
GPUs behind PCIe switch and CPU



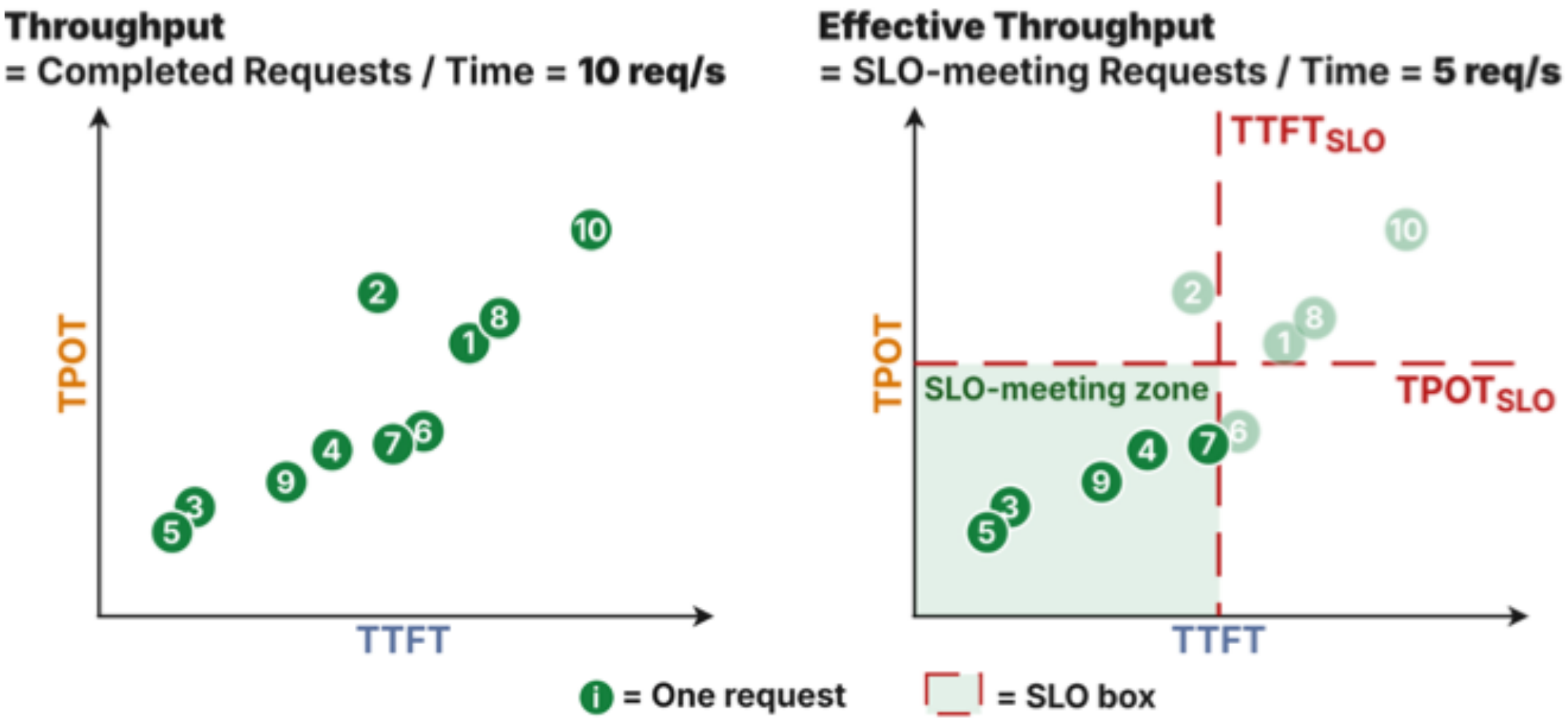
TP all-reduce becomes the bottleneck

- TP assumes NVLink-class bandwidth
  - PCIe-based GPUs and NPU systems break the assumption

LLM Inference: Prefill + Decode

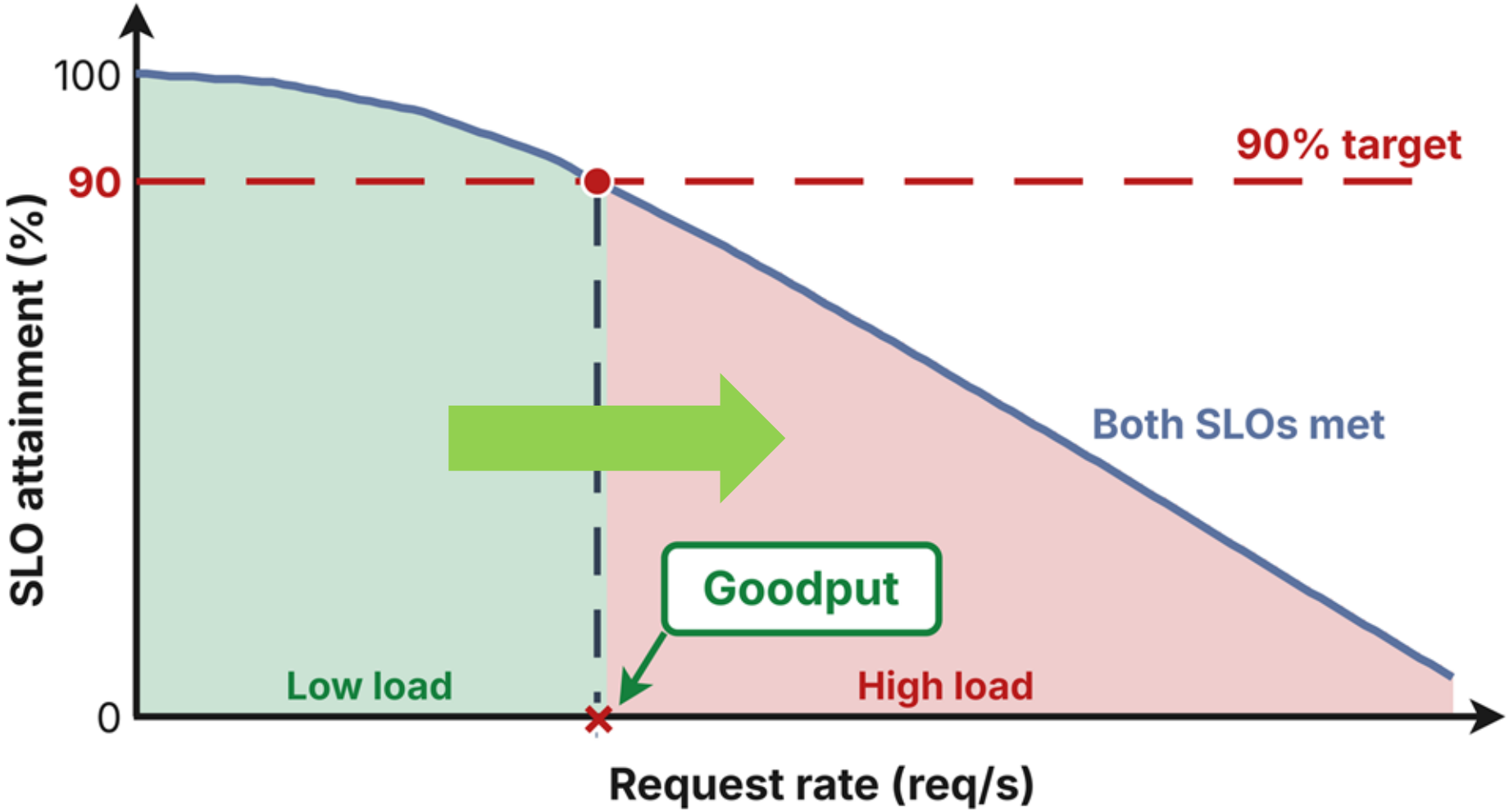


Offline (batch) inference vs. Online serving



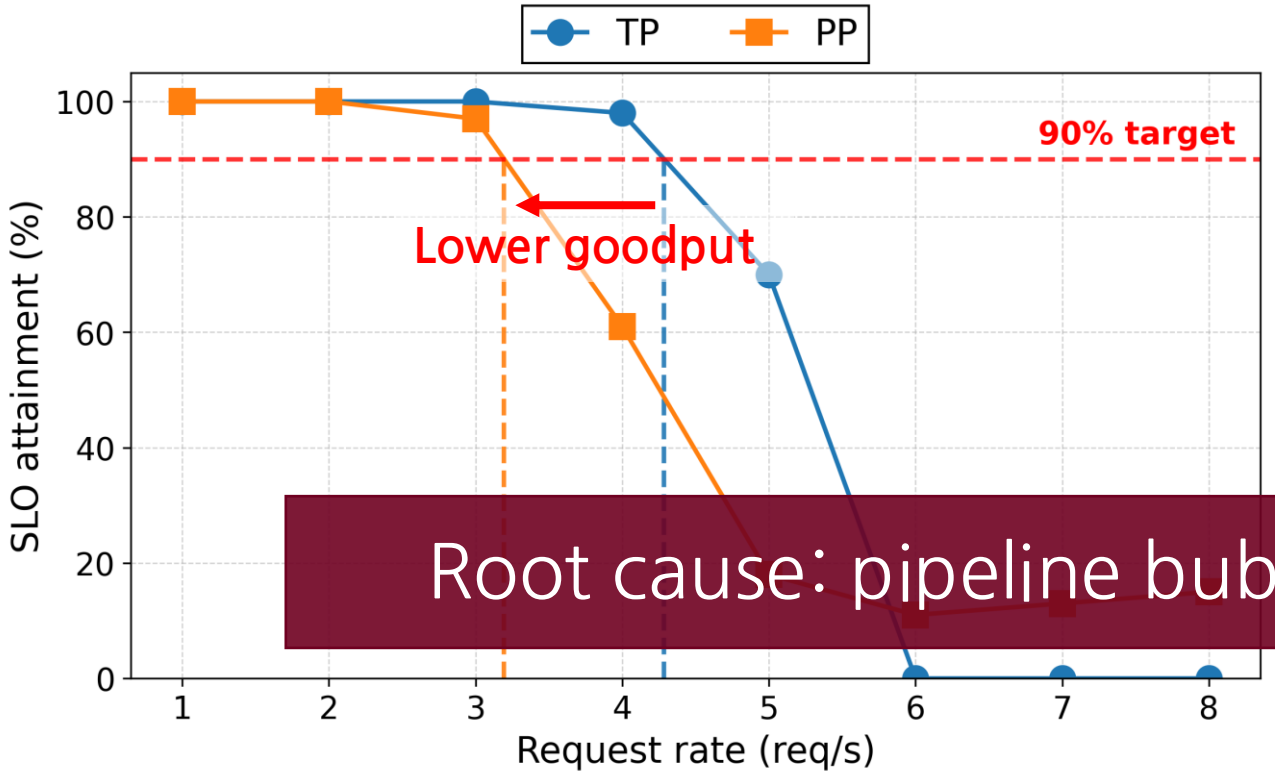
From Throughput to Goodput

Max request rate with ≥90% SLO attainment for TTFT & TPOT



Pipeline parallelism doesn't scale well

- PP shows lower goodput than TP even in PCIe environments

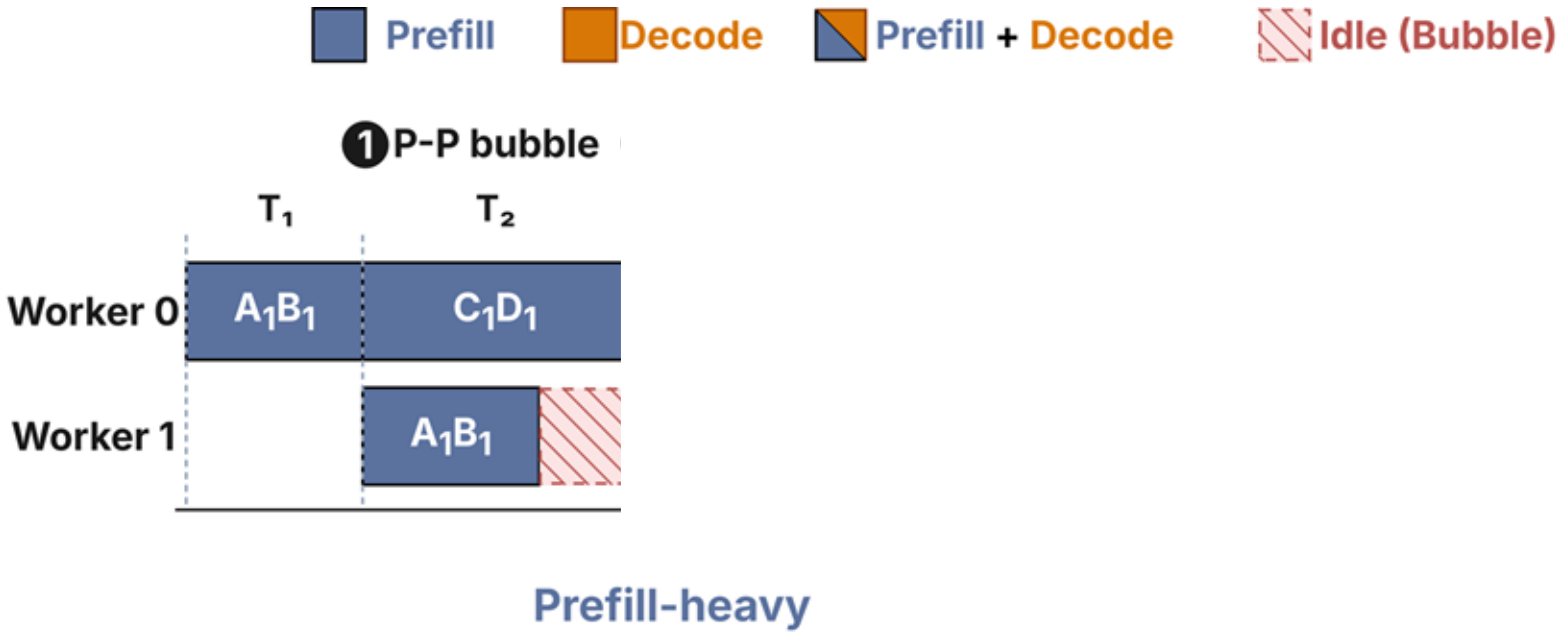


- Model: Qwen2.5-32B
- GPU: 4x A100 PCIe 4.0
- Dataset: CNN-DailyMail
  (Avg. input & output length: 883 / 65)

Root cause: pipeline bubbles

Pipeline bubbles in online LLM serving

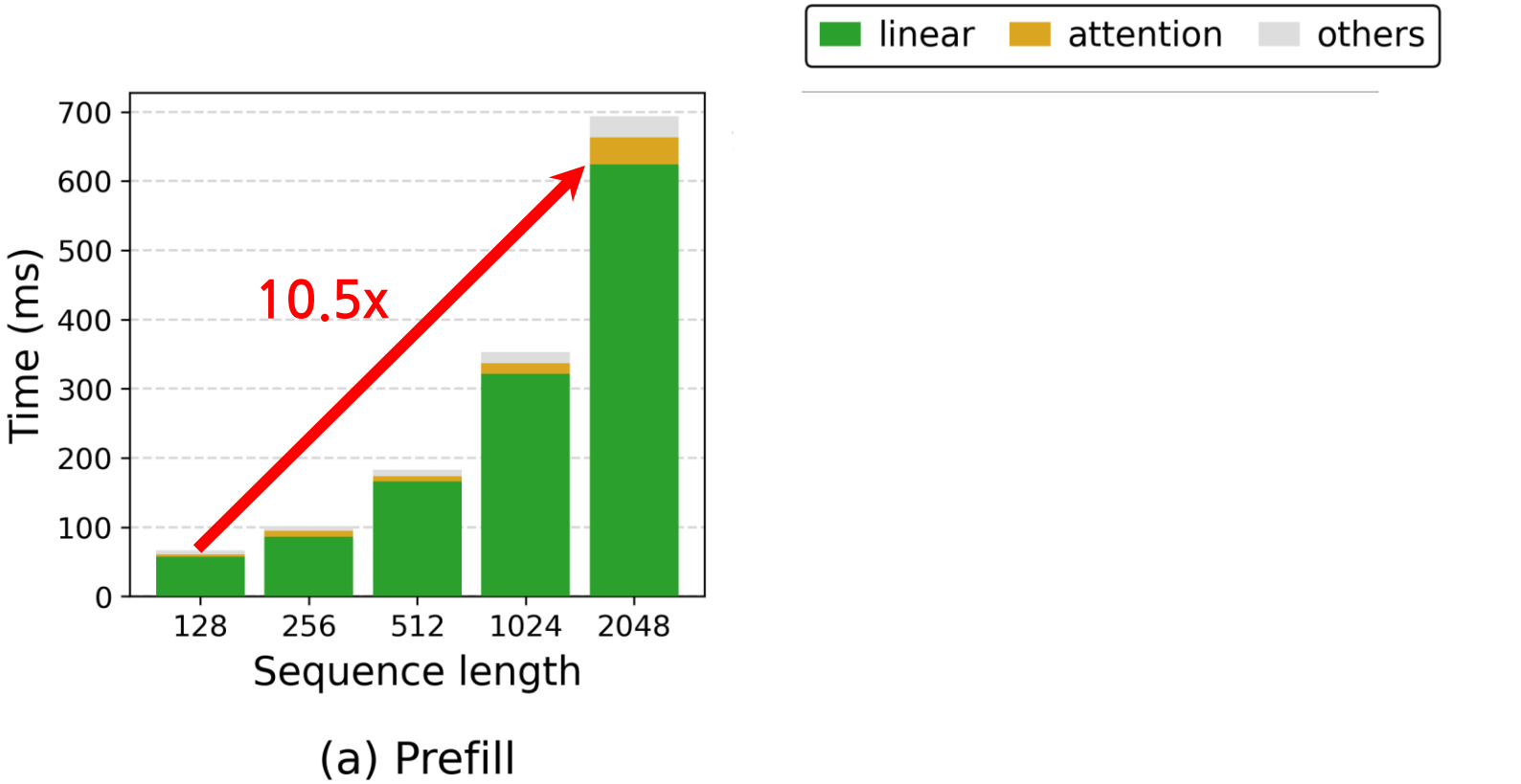
- Diverse user reqeust patterns lead to severe pipeline imbalances



A_1B_1: two requests A and B in the same batch and the subscript indicates the iteration number

Pipeline bubbles in **prefill** phases

- Prefill latency scales drastically with input sequence length

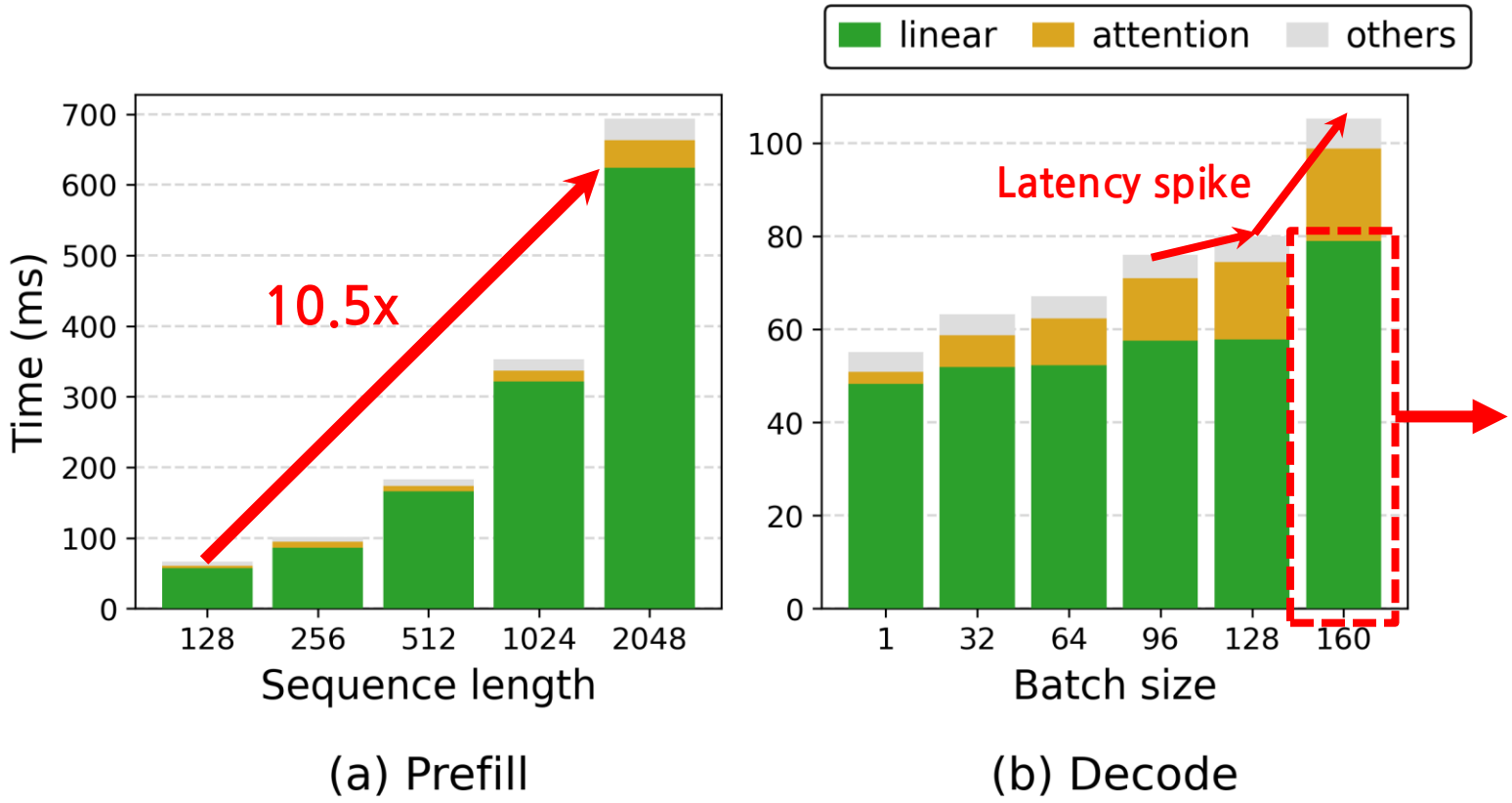


(a) Prefill

- Model: Qwen2.5-32B
- GPU: A100 40GB

Pipeline bubbles in **decode** phases

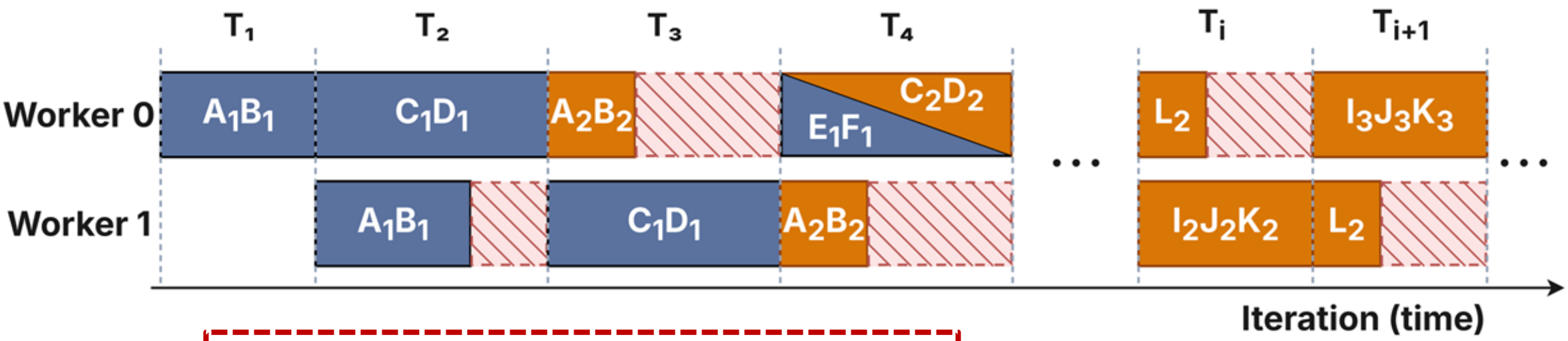
- Decode latency grows with batch size and jumps sharply beyond 128



- Model: Qwen2.5-32B
- GPU: A100 40GB

Caused by GPU kernel tiling at 128-token boundaries

Our approach: chunking & rebalancing



Prefill-heavy

↓

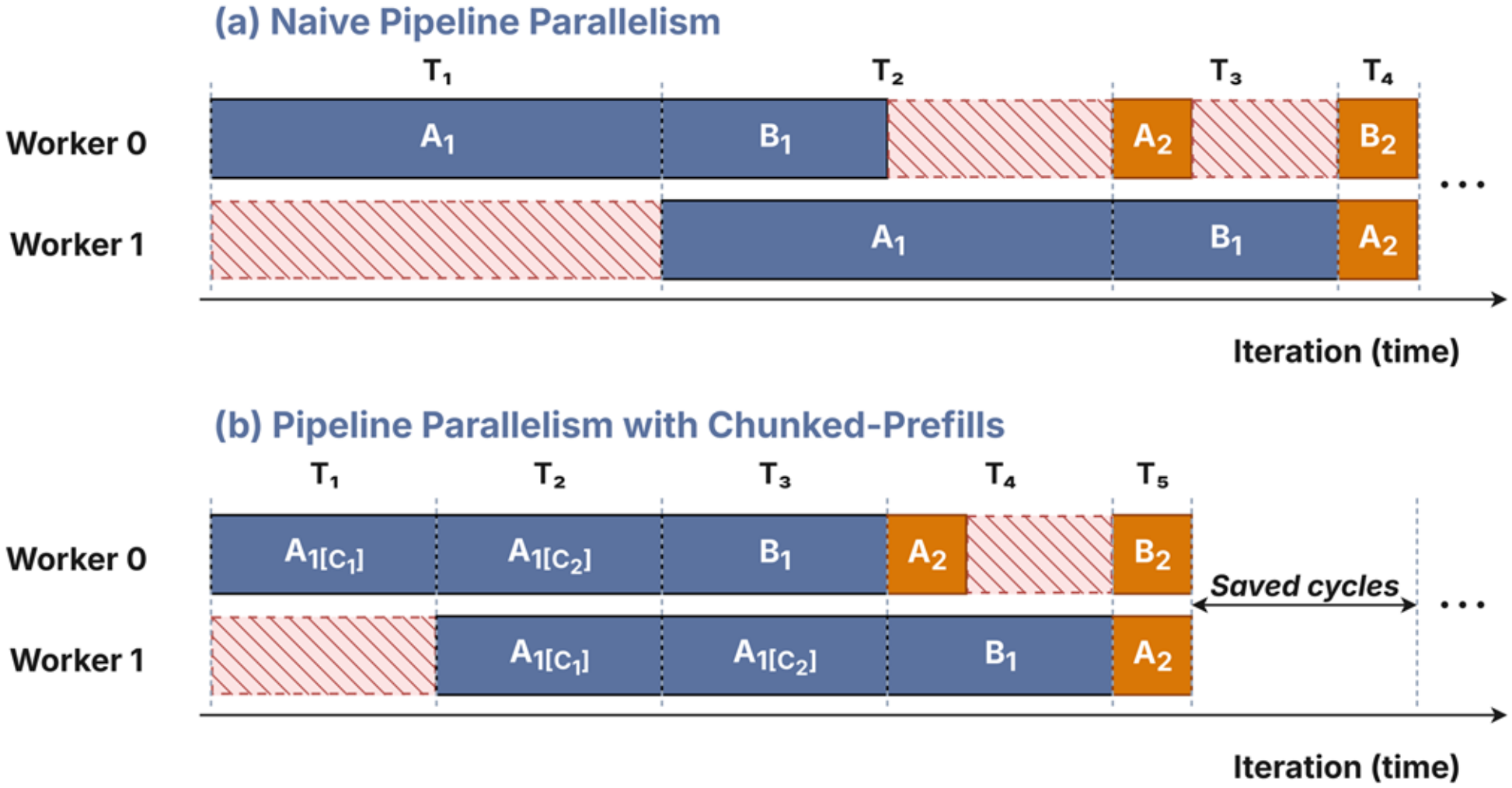
Dynamic Chunked Prefill

Decode-heavy

↓

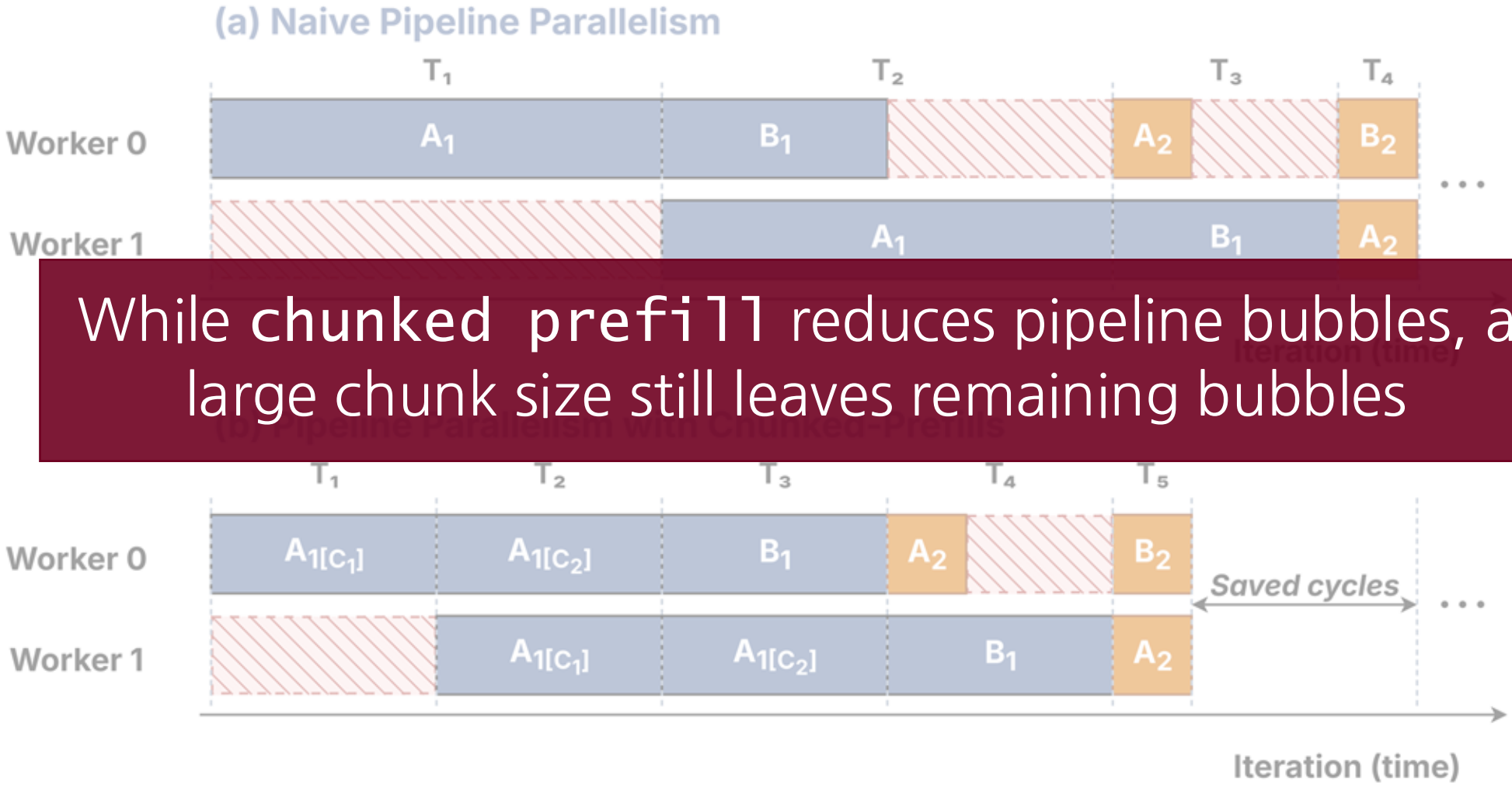
Delay Scheduling

Chunked Prefill in pipeline parallelism



[C_n] denotes the n-th chunk

Chunked Prefill in pipeline parallelism

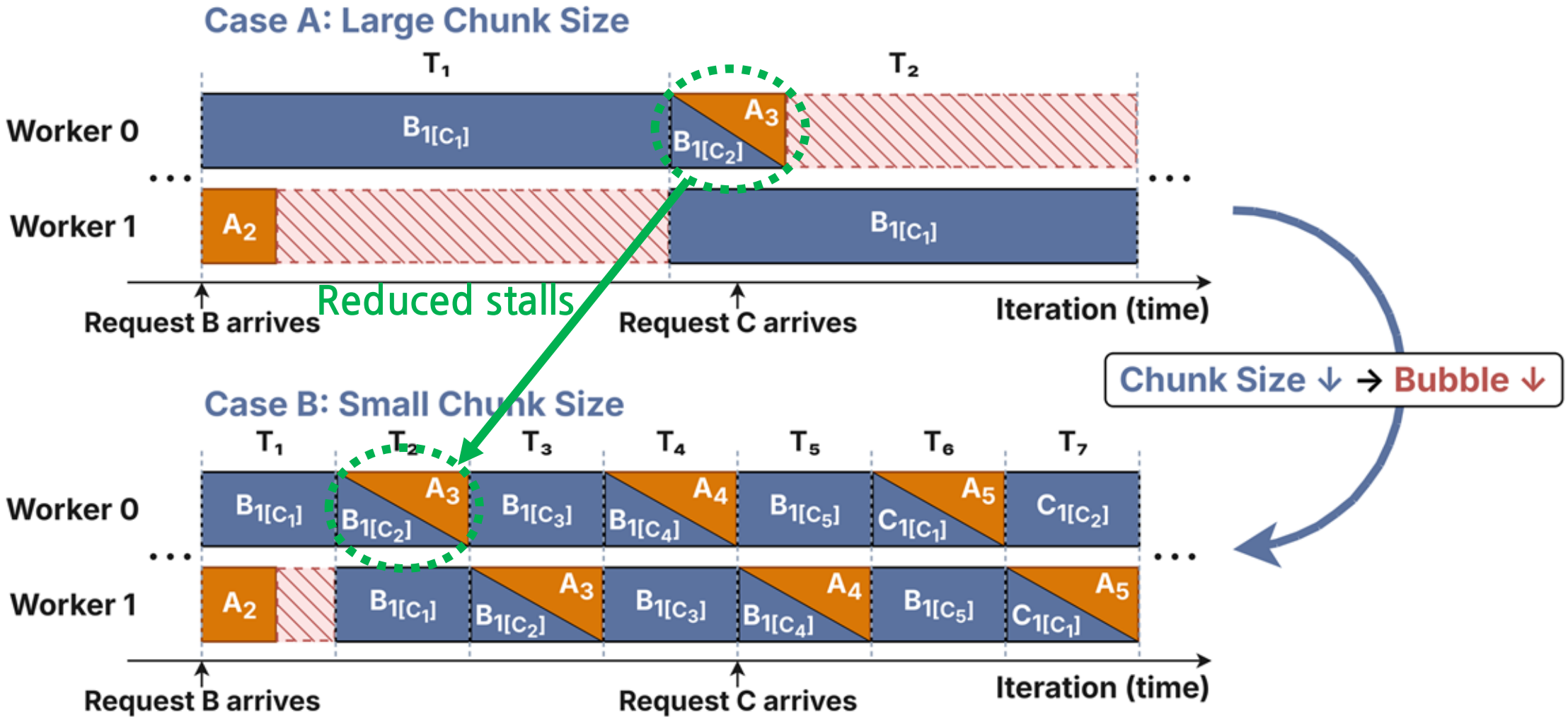


While **chunked prefill** reduces pipeline bubbles, a large chunk size still leaves remaining bubbles

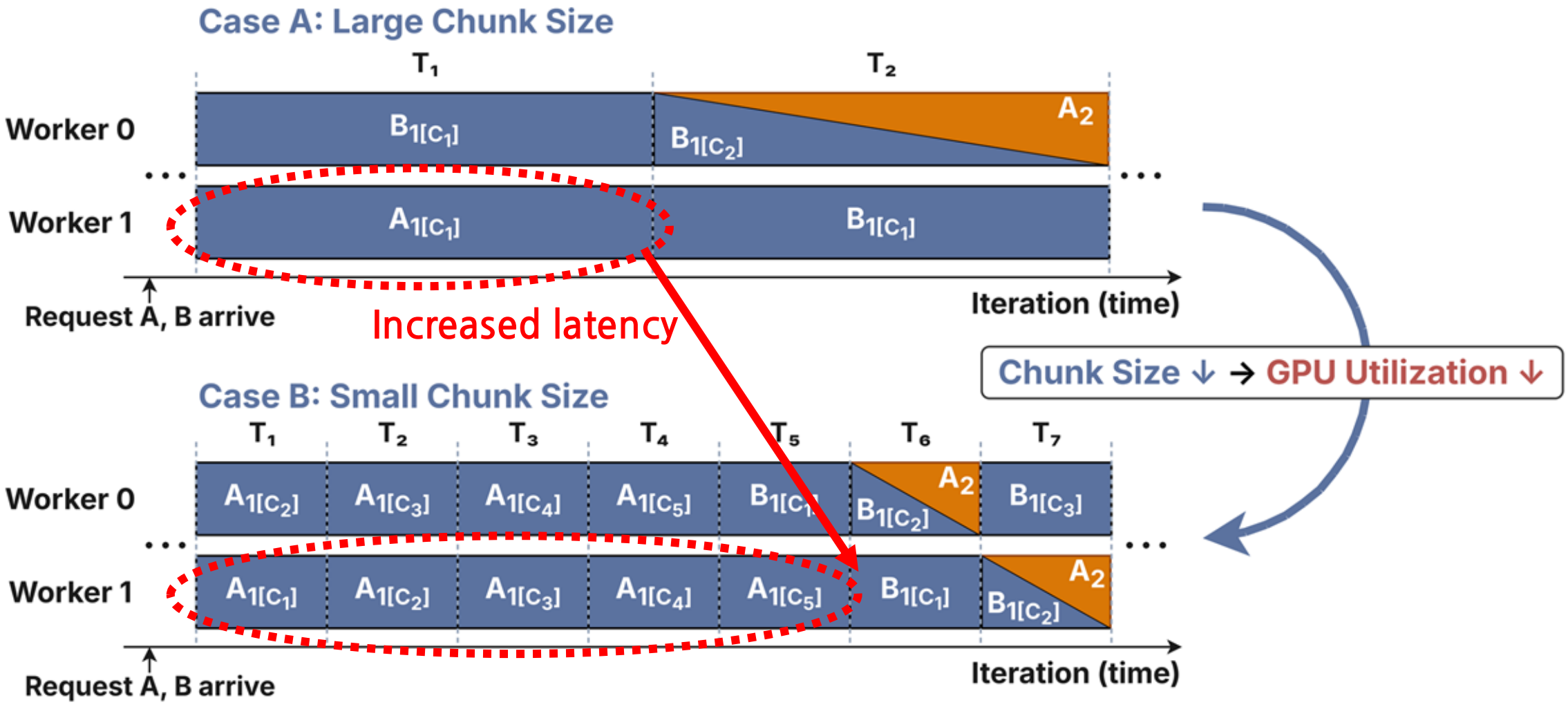
[C_n] denotes the n-th chunk

Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve (OSDI 2024)

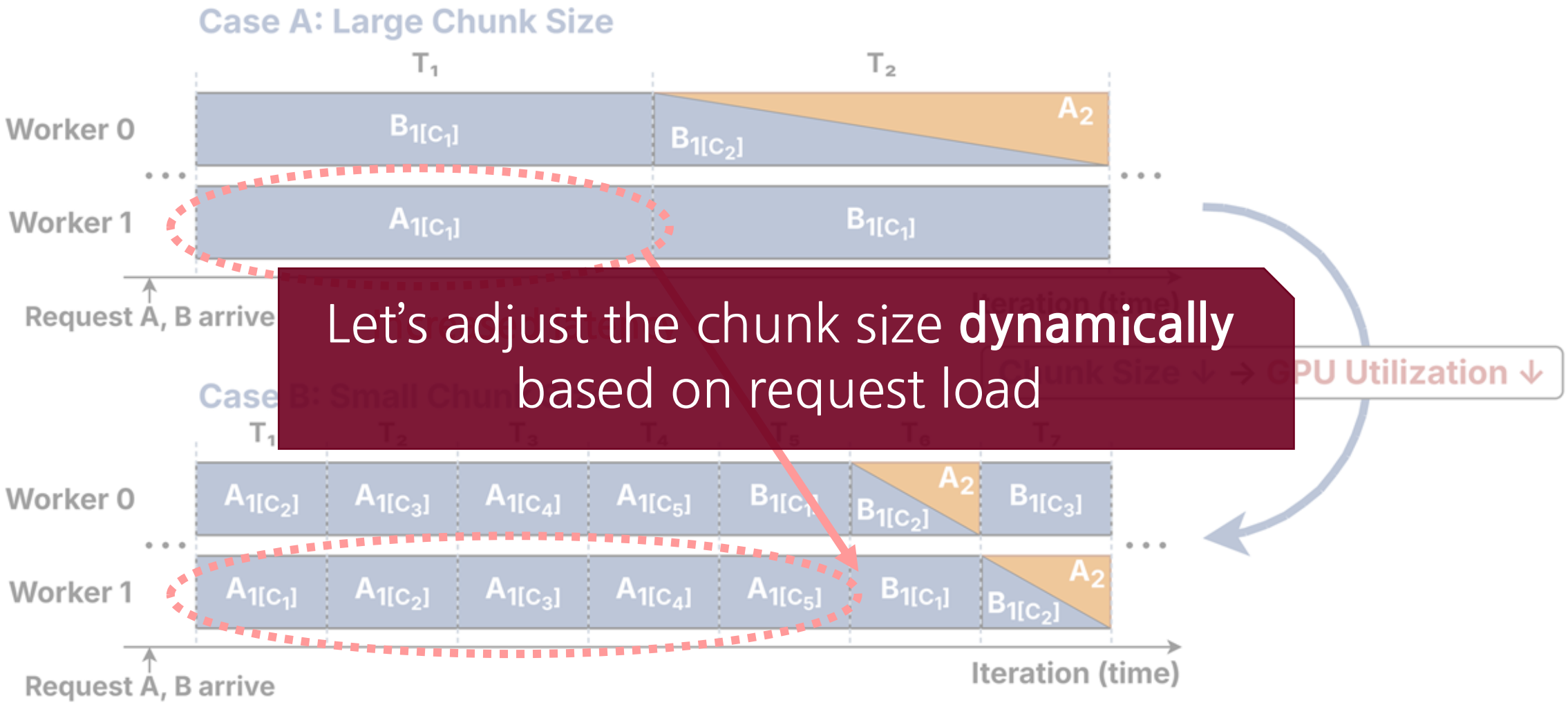
Key observation #1: chunk size under low load



Key observation #1: chunk size under high load

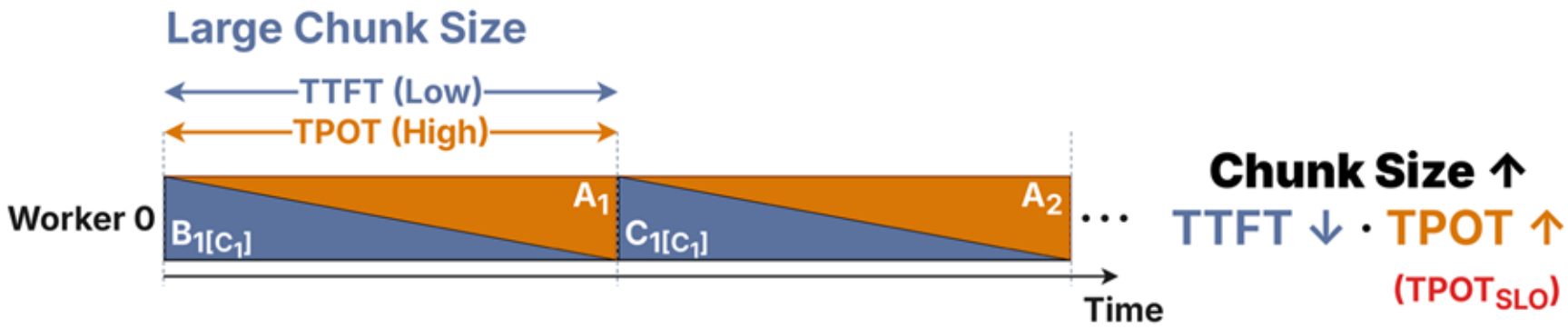


Key observation #1: chunk size under high load



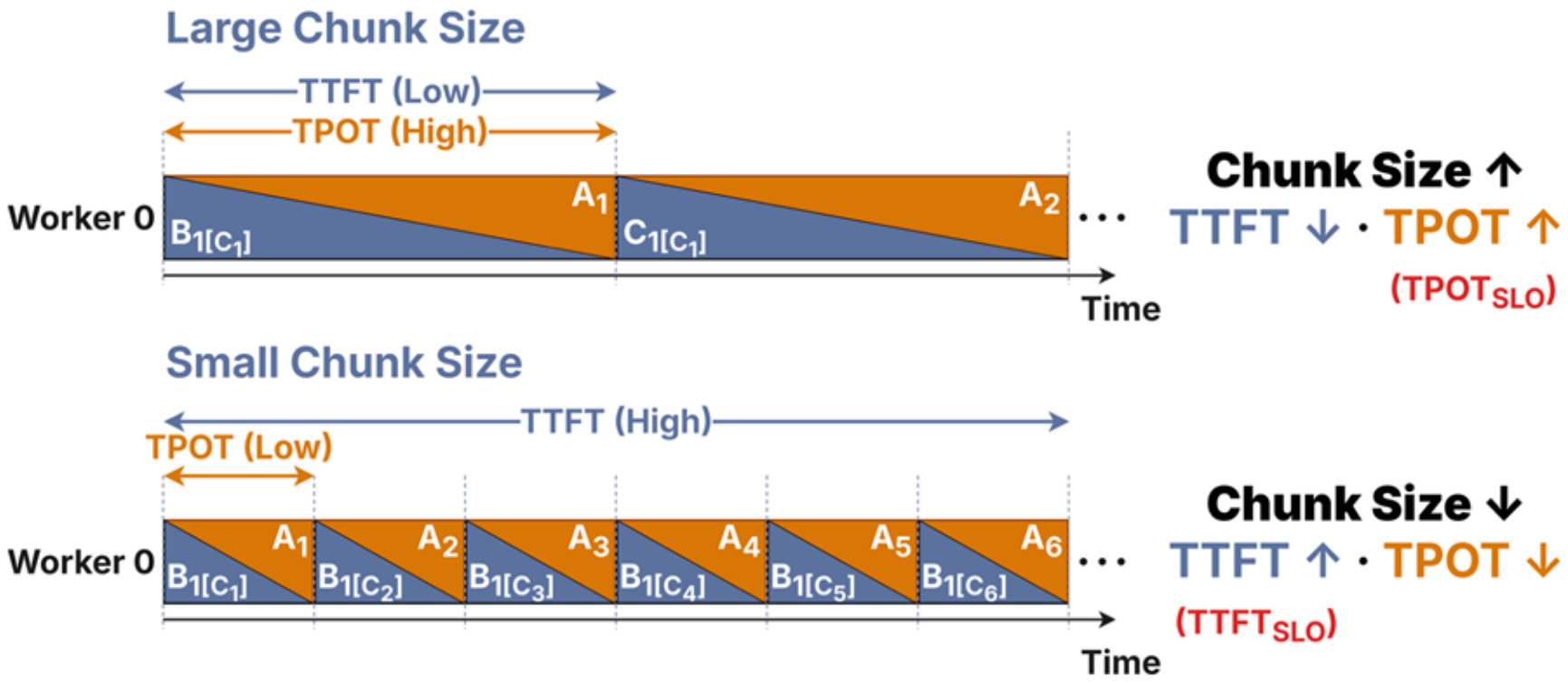
SLO-driven chunk size adapation

- Deriving upper and lower chunk sizes based on given SLOs
  - Prioritize the lower-bound for reducing bubbles



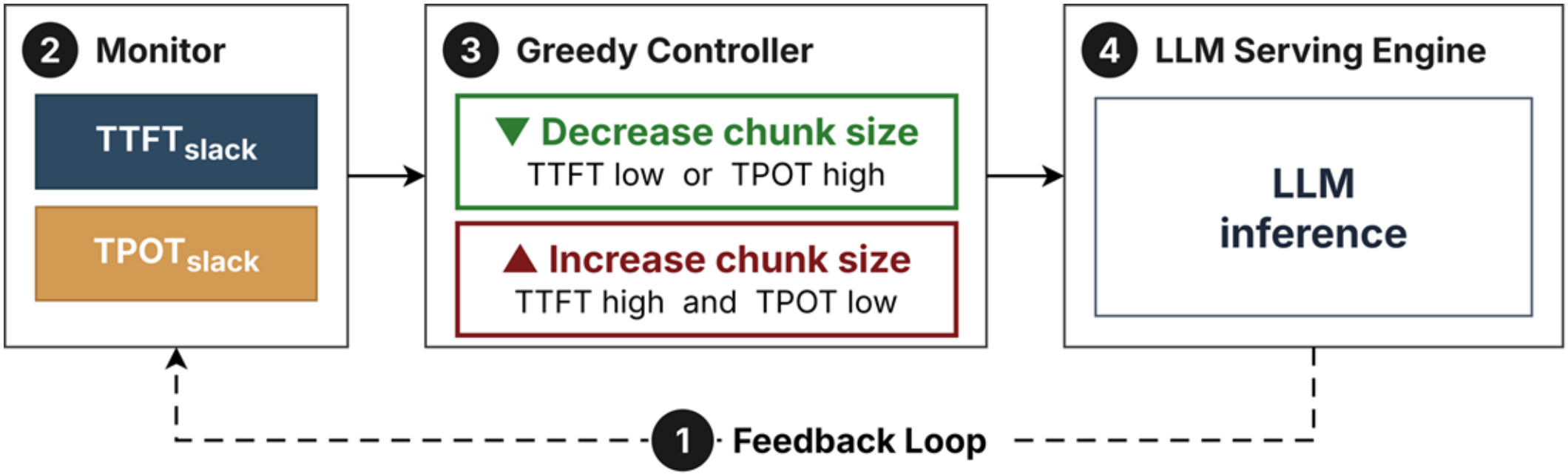
SLO-driven chunk size adapation

- Deriving upper and lower chunk sizes based on given SLOs
  - Prioritize the lower-bound for reducing bubbles



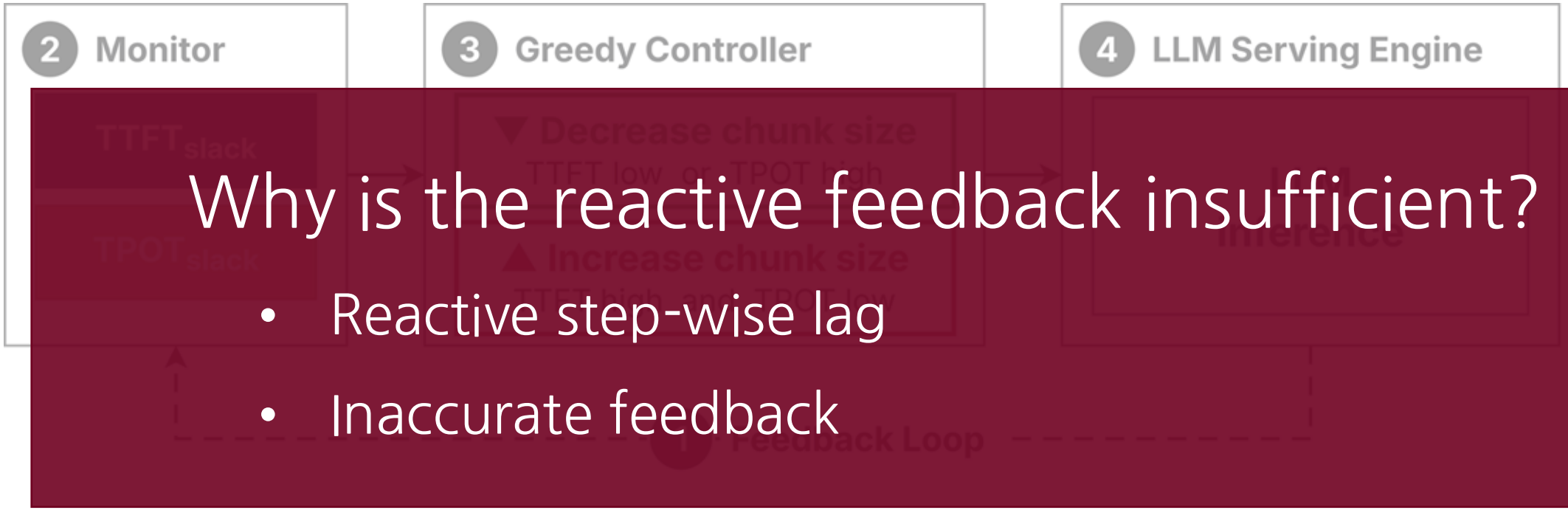
(1) Greedy approach: dynamic chunked prefill

- Adjusts chunk size via real-time TTFT & TPOT feedback



(1) Greedy approach: dynamic chunked prefill

- Adjusts chunk size via real-time TTFT & TPOT feedback

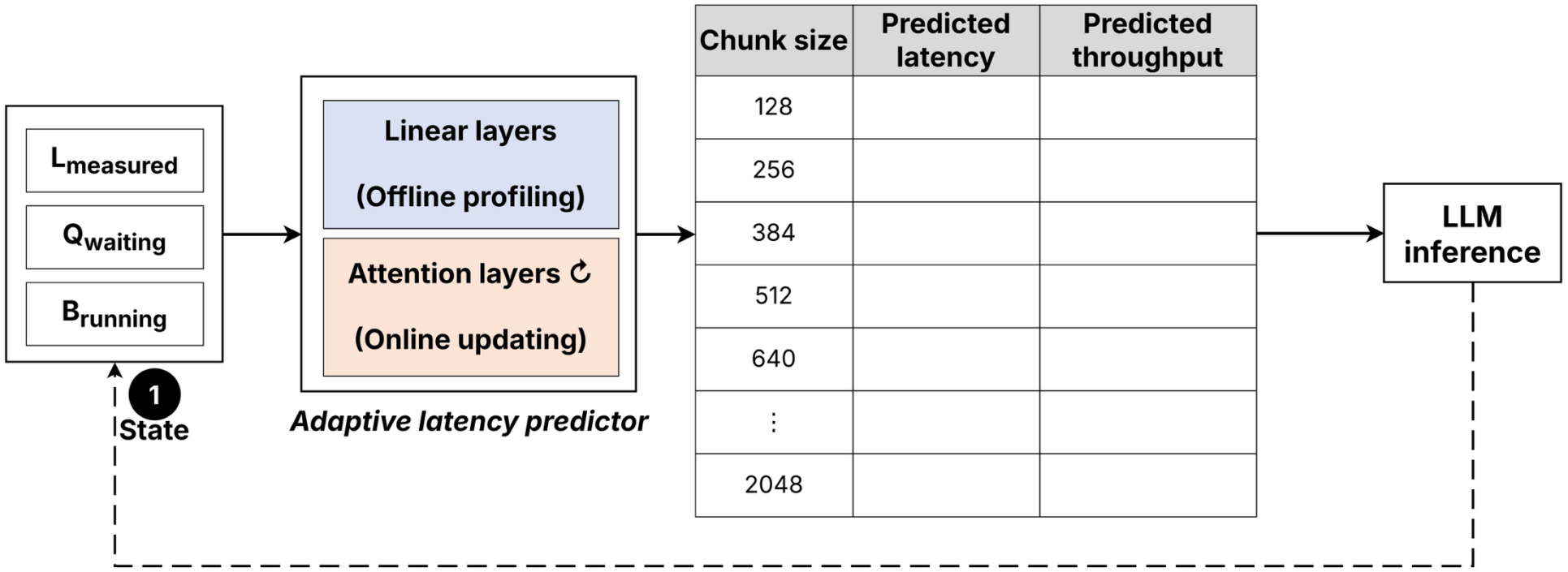


Why is the reactive feedback insufficient?

- Reactive step-wise lag
- Inaccurate feedback

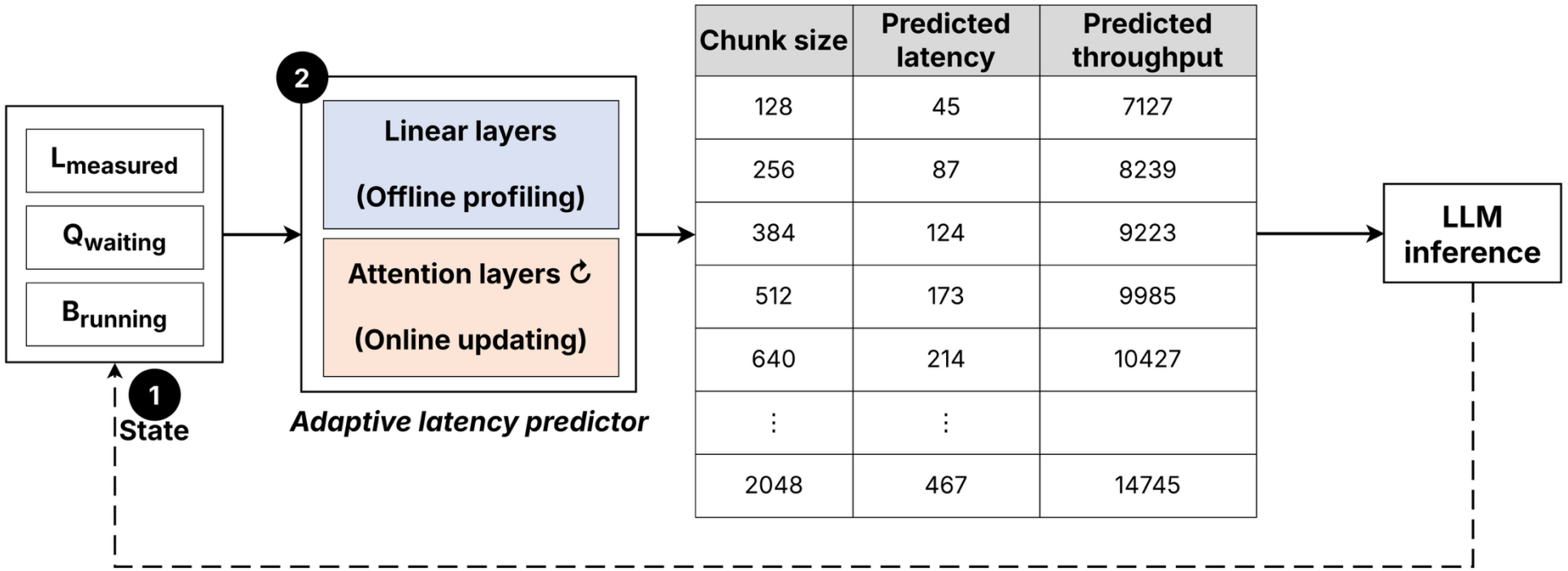
(2) predictive approach: dynamic chunked prefill

- Retrieve real-time system states
  - Measured latency ($L_{measured}$), Waiting queue($Q_{waiting}$), Running batch ($B_{running}$)



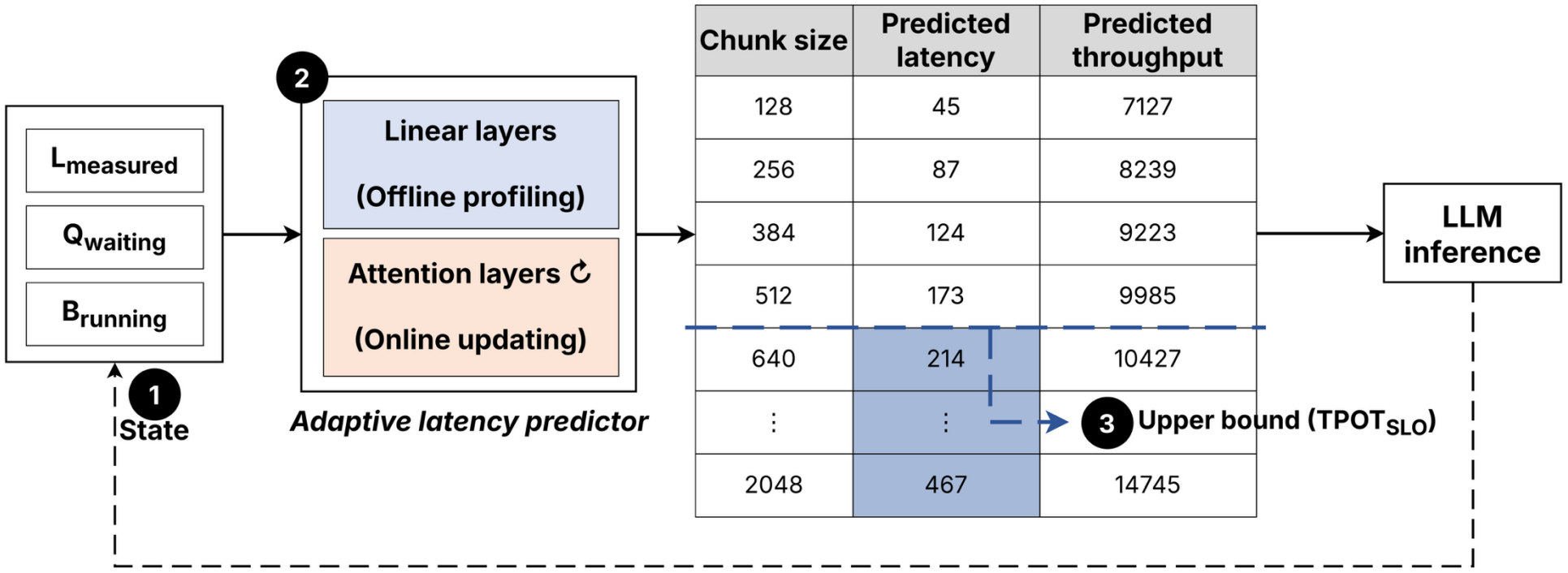
(2) predictive approach: dynamic chunked prefill

- Predict latency and throughput for each iteration
  - Linear layers (offline): measure few chunk size in prior
  - Attention layers (online): update this portion based on measured latency errors



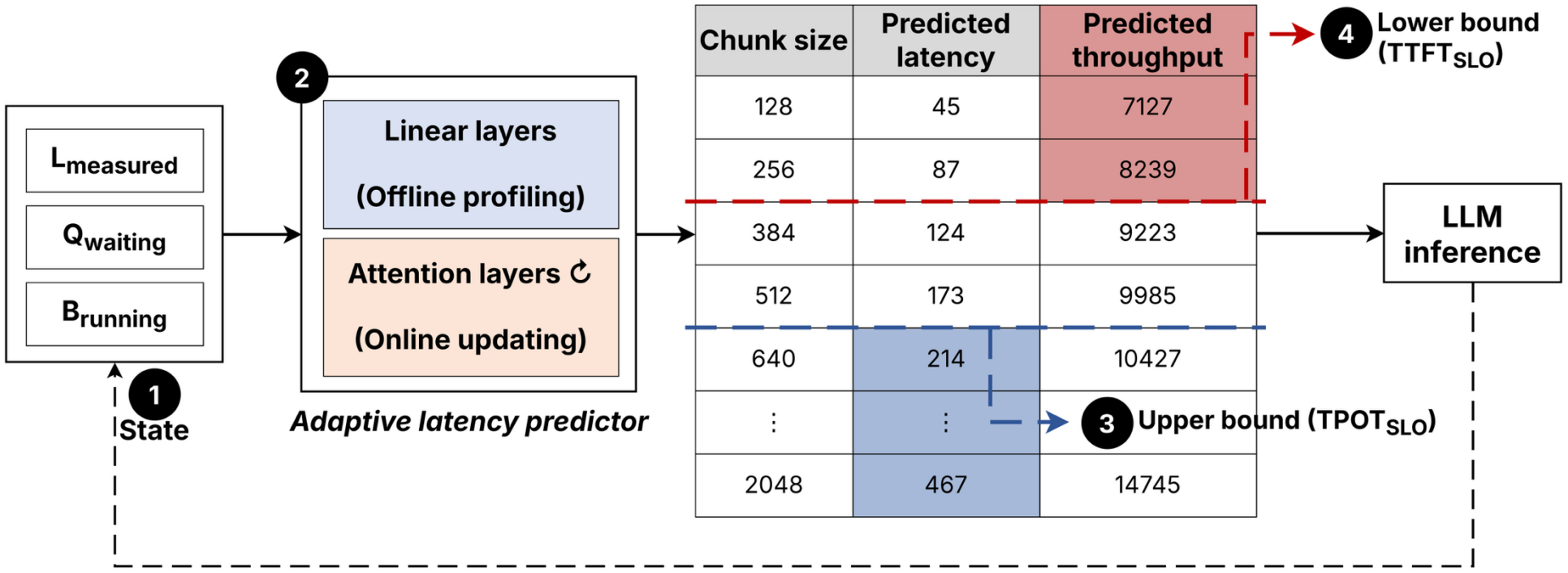
(2) predictive approach: dynamic chunked prefill

- Upper bound: limited by $TPOT_{SLO}$ on predicted latency



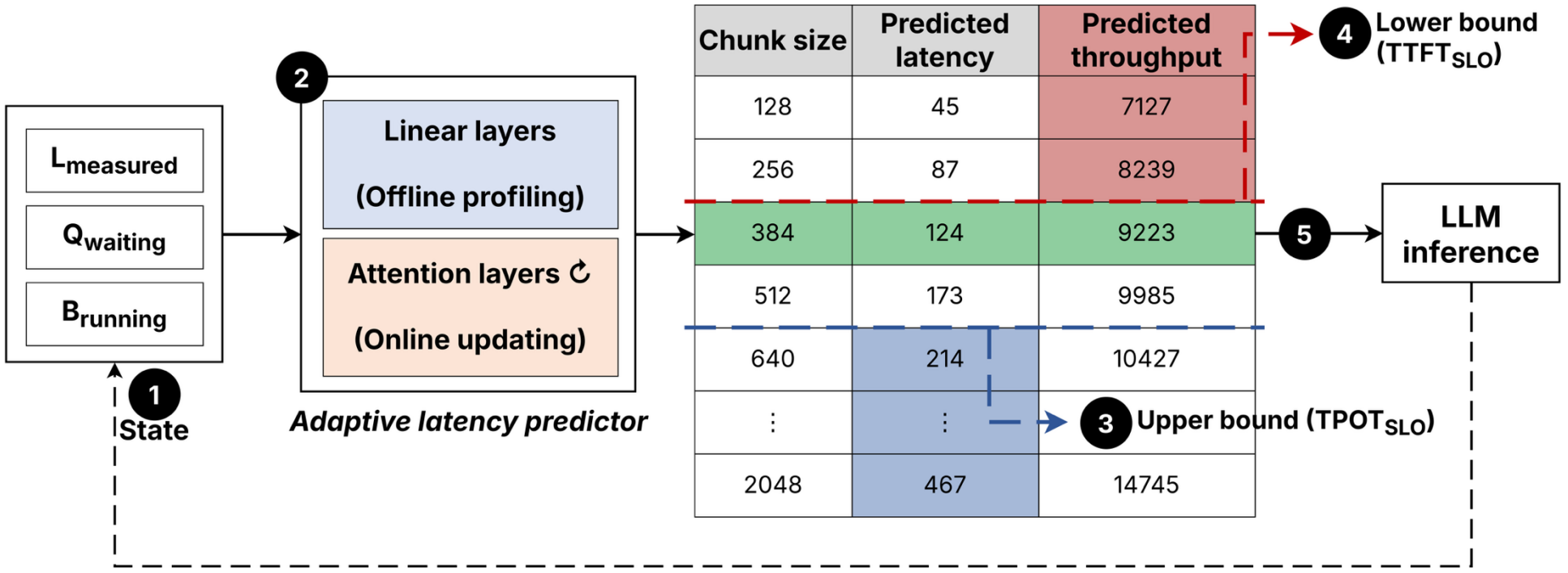
(2) predictive approach: dynamic chunked prefill

- Upper bound: limited by $TPOT_{SLO}$ on predicted latency
- Lower bound: filtered by $TTFT_{SLO}$ on predicted throughput

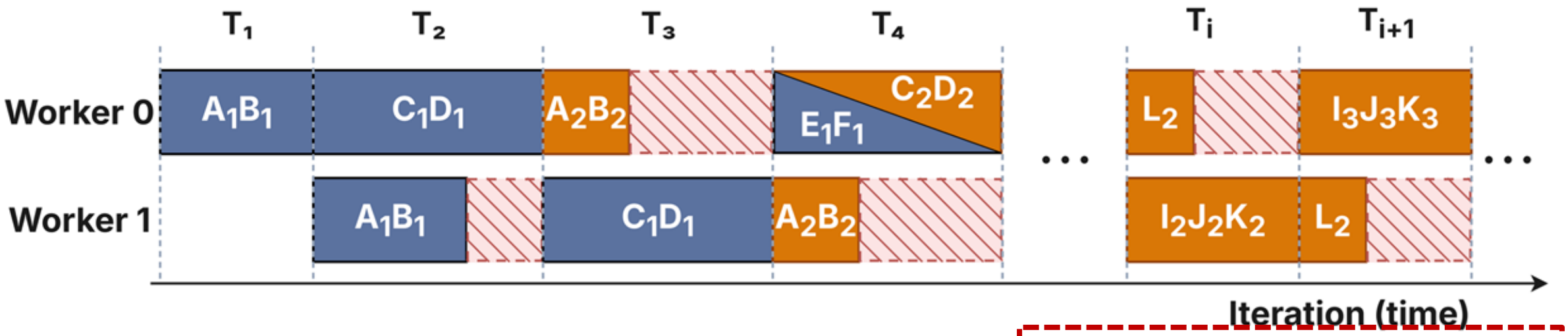


(2) predictive approach: dynamic chunked prefill

- Prioritizing the lower bound to minimize pipeline bubbles



Our approach: chunking & rebalancing



Prefill-heavy

↓

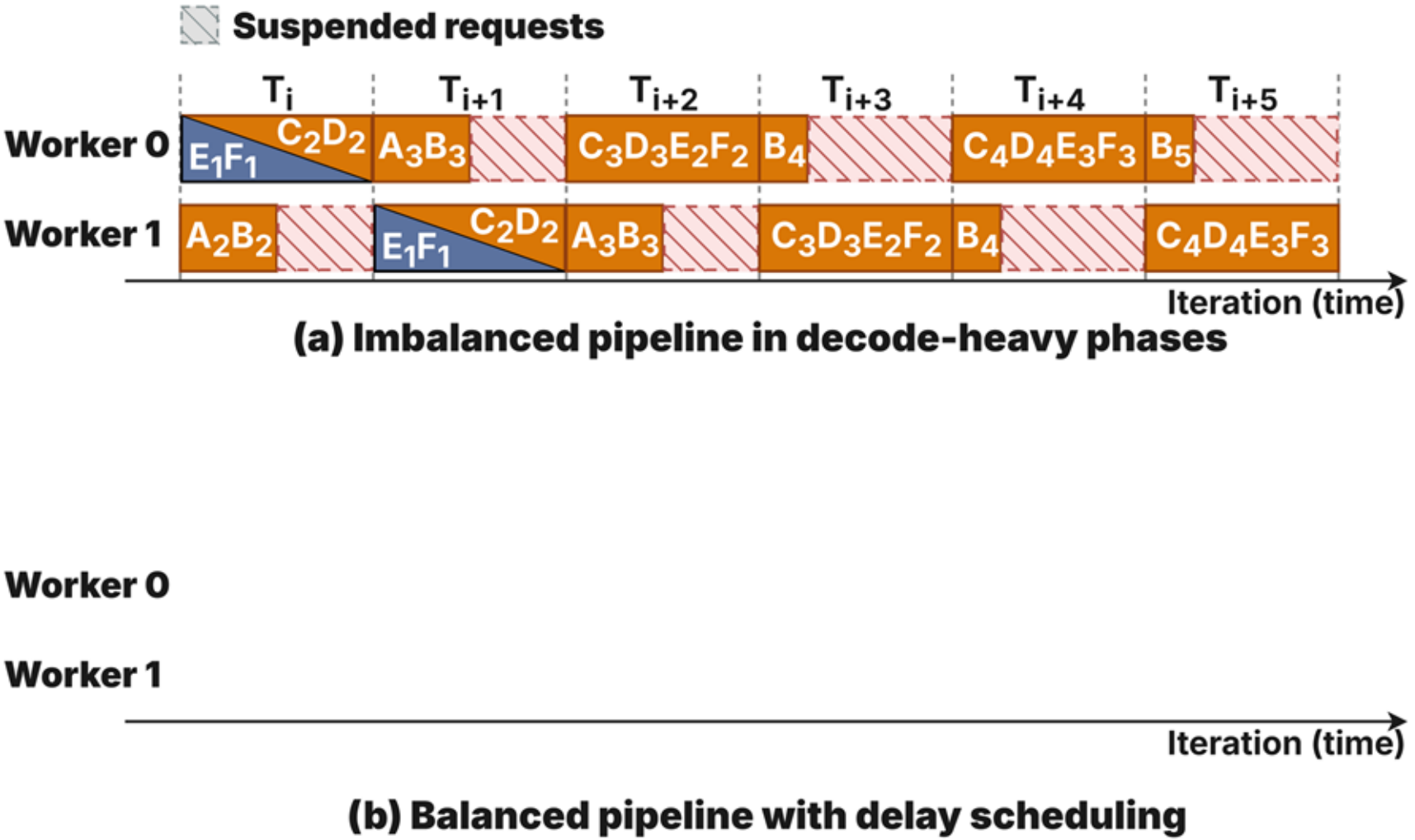
Dynamic Chunked Prefill

Decode-heavy

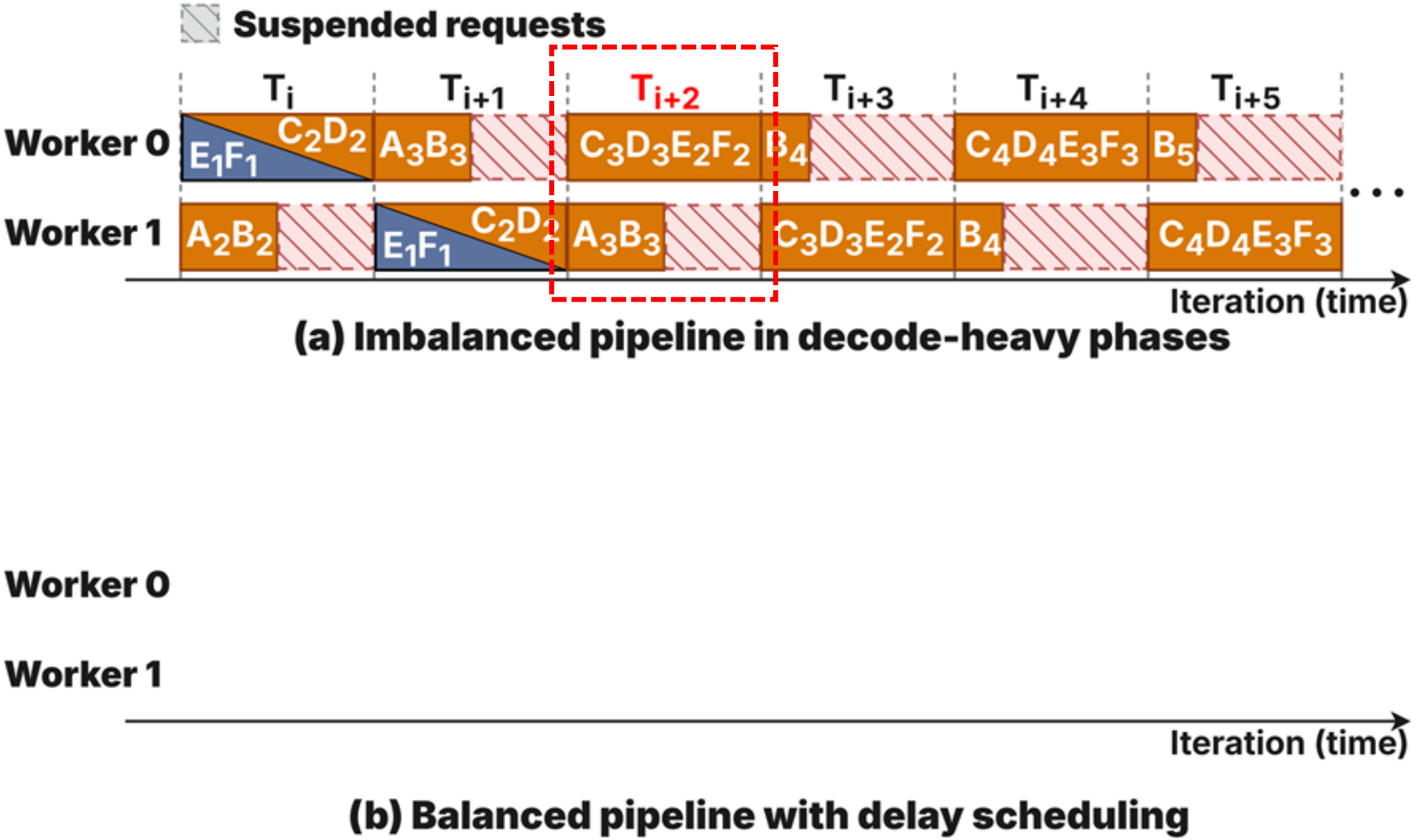
↓

Delay Scheduling

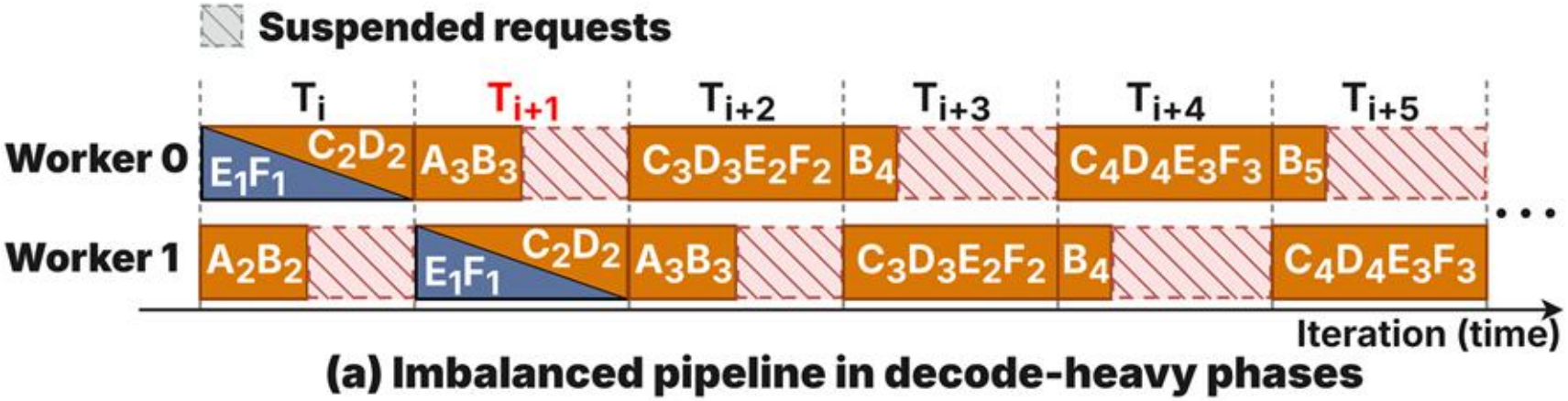
Delay scheduling for pipeline rebalancing



Delay scheduling for pipeline rebalancing

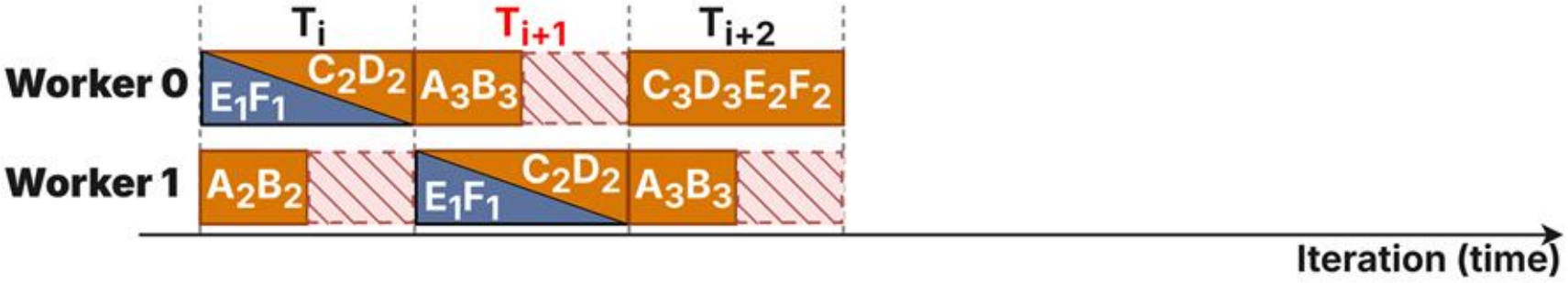


Delay scheduling for pipeline rebalancing



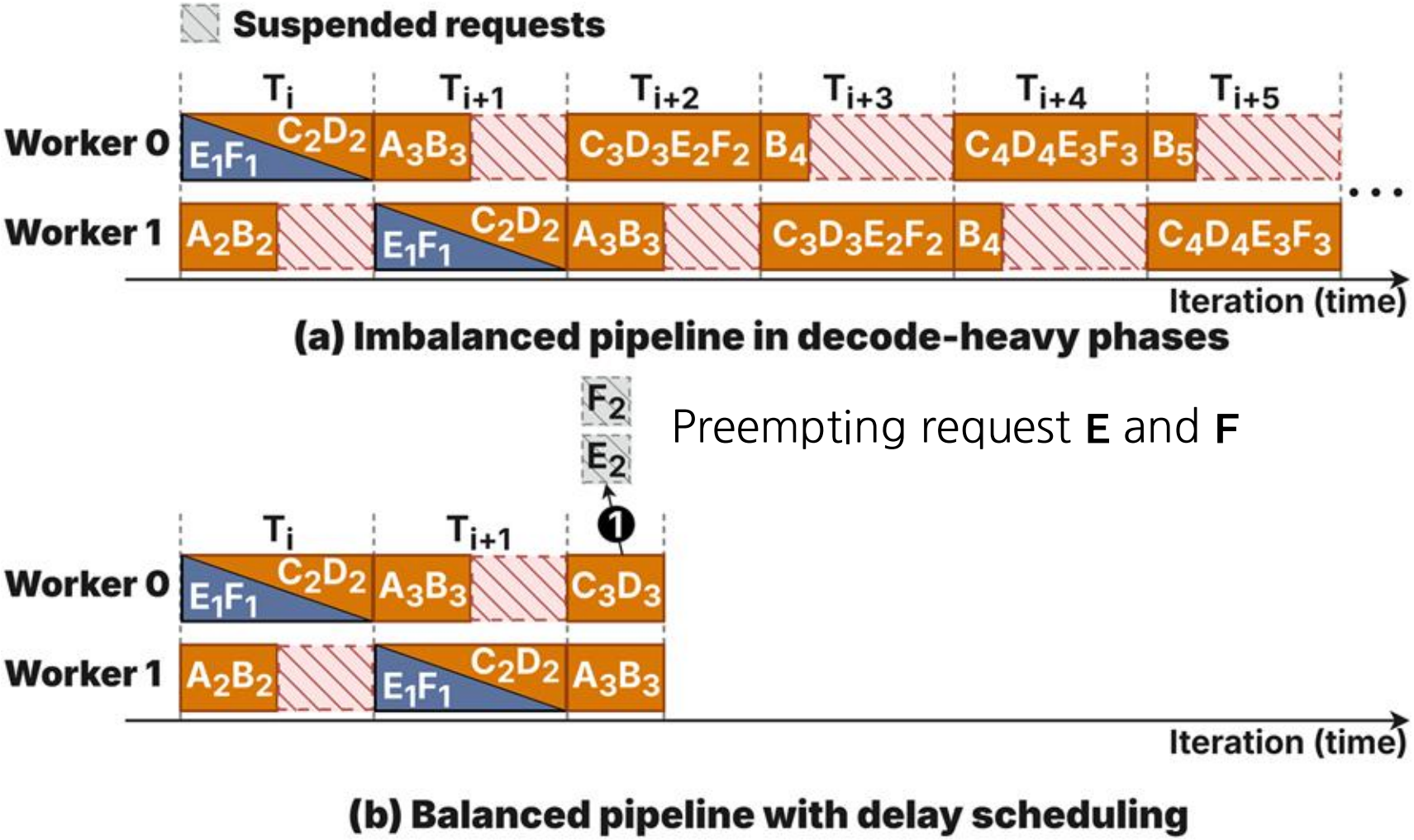
(a) Imbalanced pipeline in decode-heavy phases

Execution remains identical to the naive baseline up to T_{i+1}

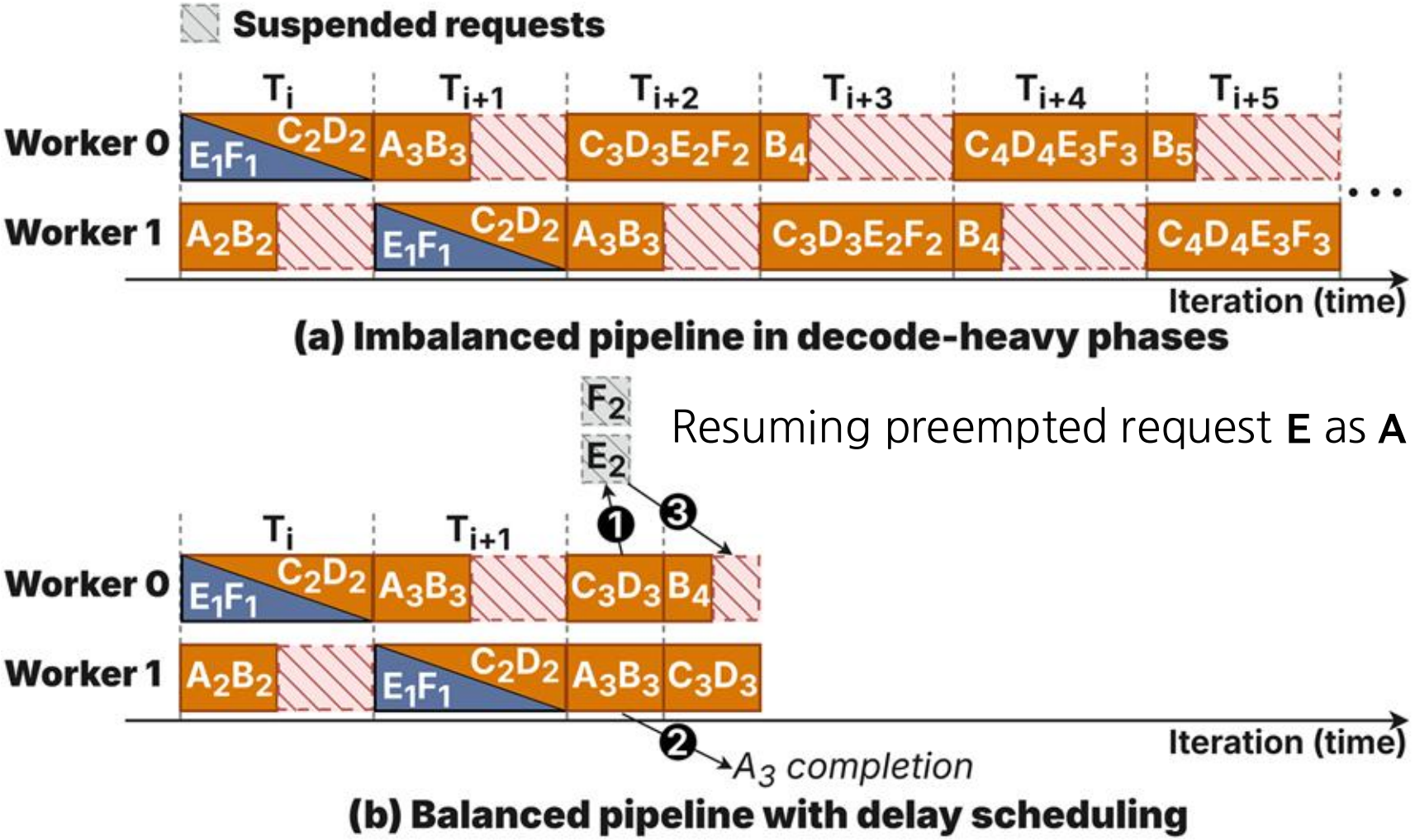


(b) Balanced pipeline with delay scheduling

Delay scheduling for pipeline rebalancing

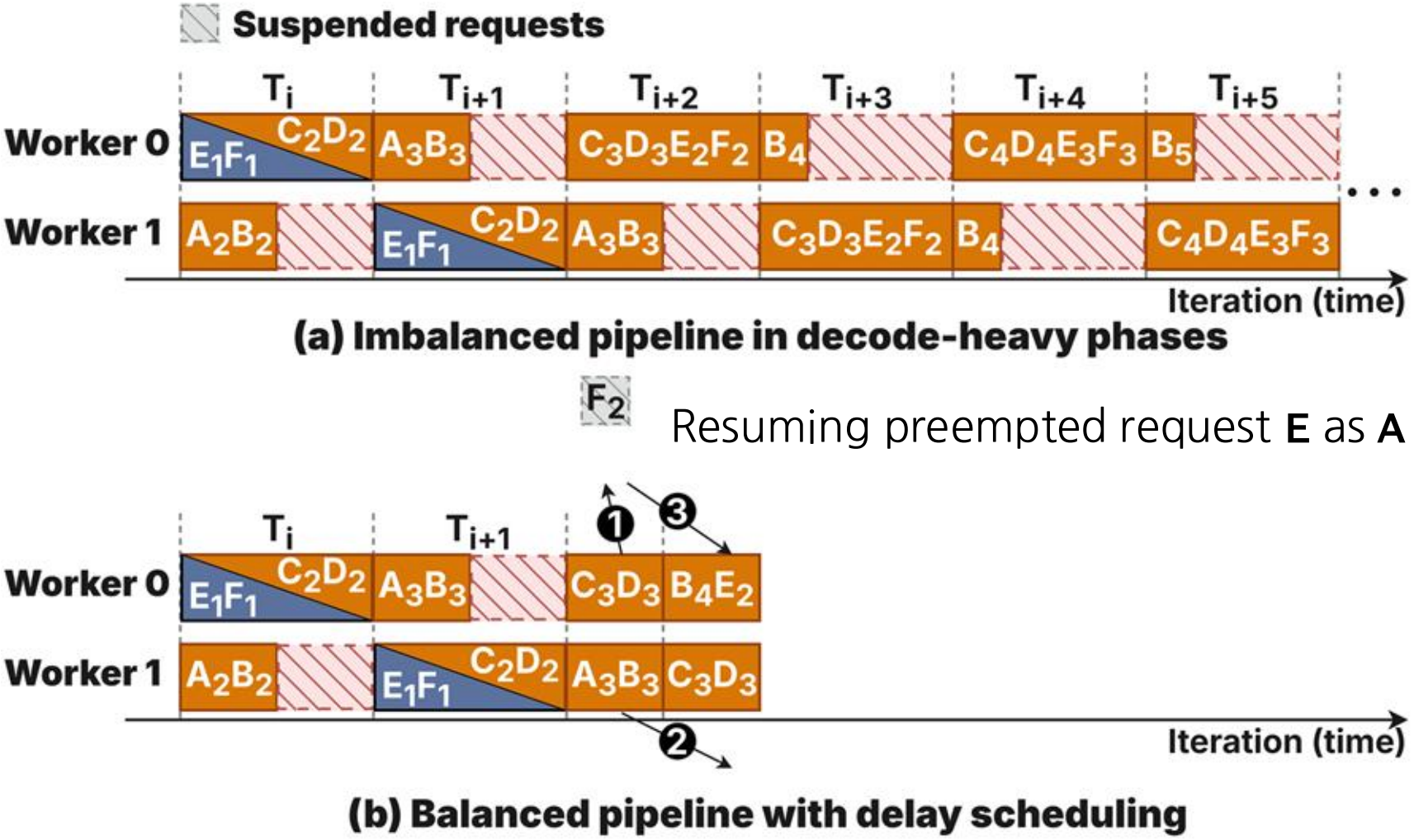


Delay scheduling for pipeline rebalancing

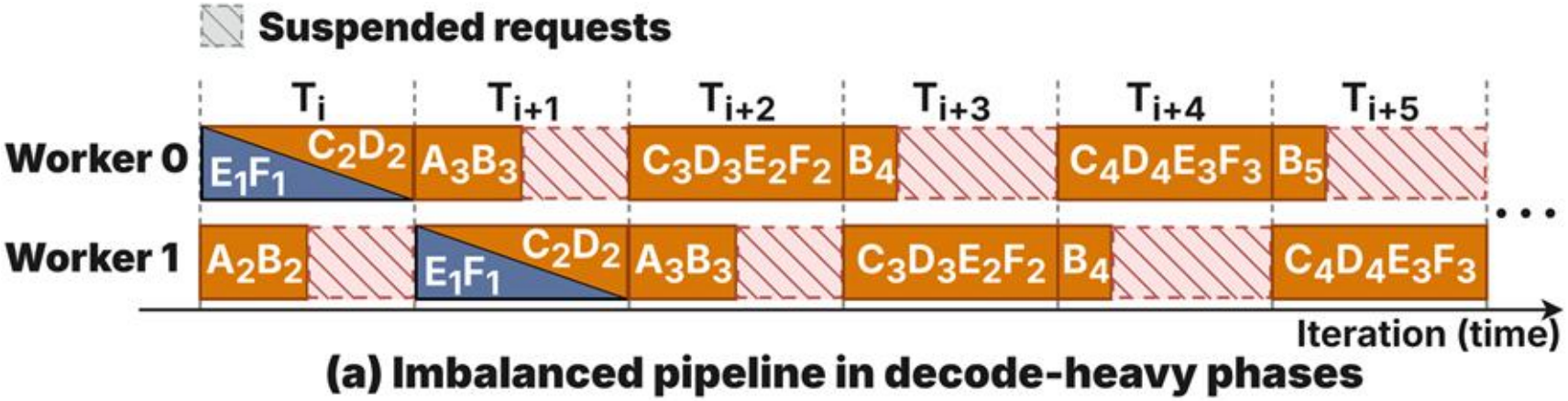


Resuming preempted request **E** as **A** completes

Delay scheduling for pipeline rebalancing

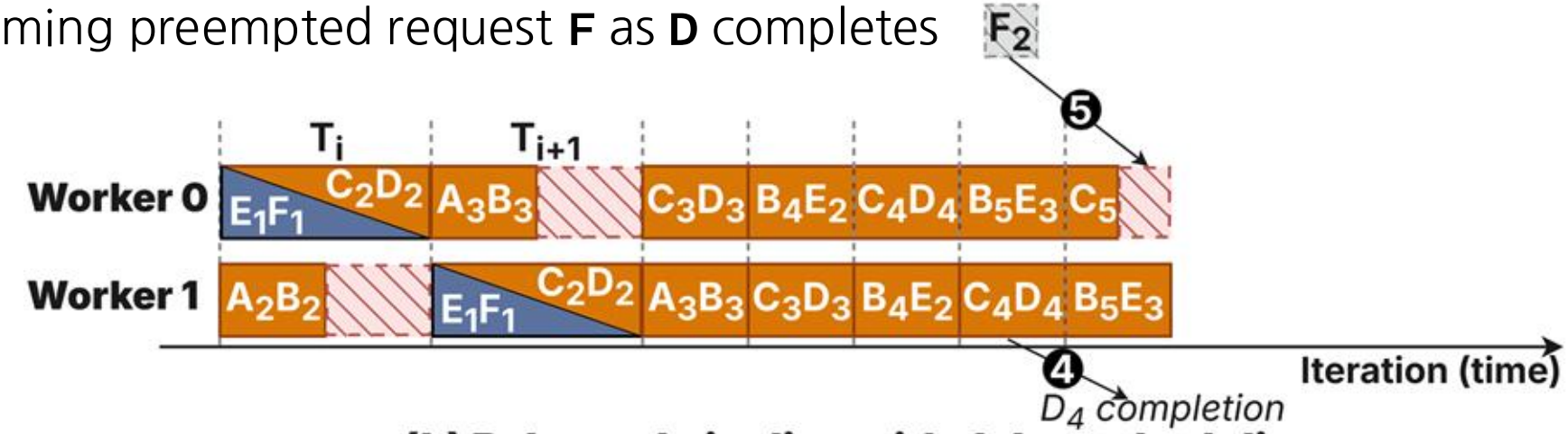


Delay scheduling for pipeline rebalancing



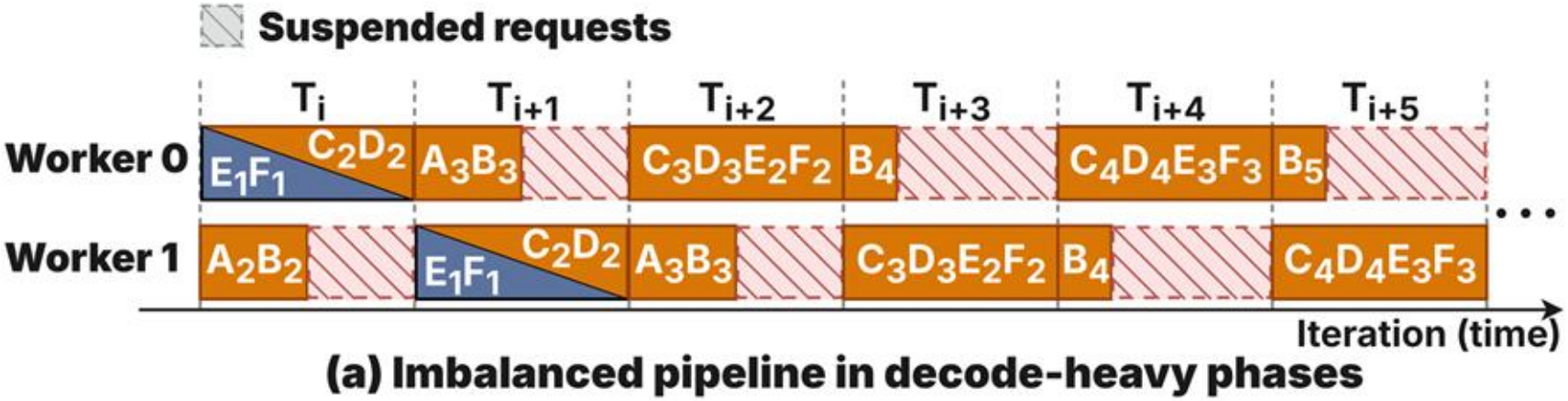
(a) Imbalanced pipeline in decode-heavy phases

Resuming preempted request **F** as **D** completes



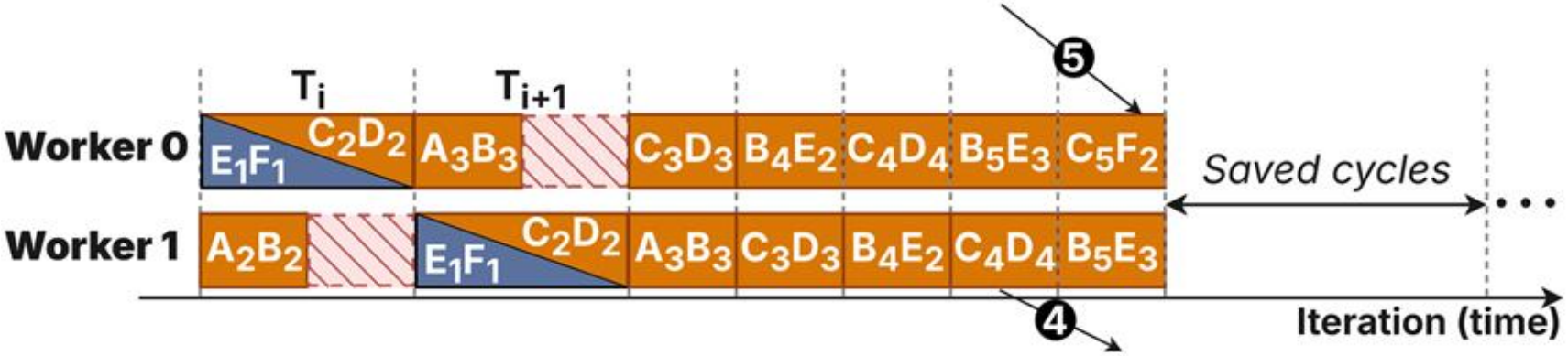
(b) Balanced pipeline with delay scheduling

Delay scheduling for pipeline rebalancing



(a) Imbalanced pipeline in decode-heavy phases

Resuming preempted request **F** as **D** completes



(b) Balanced pipeline with delay scheduling

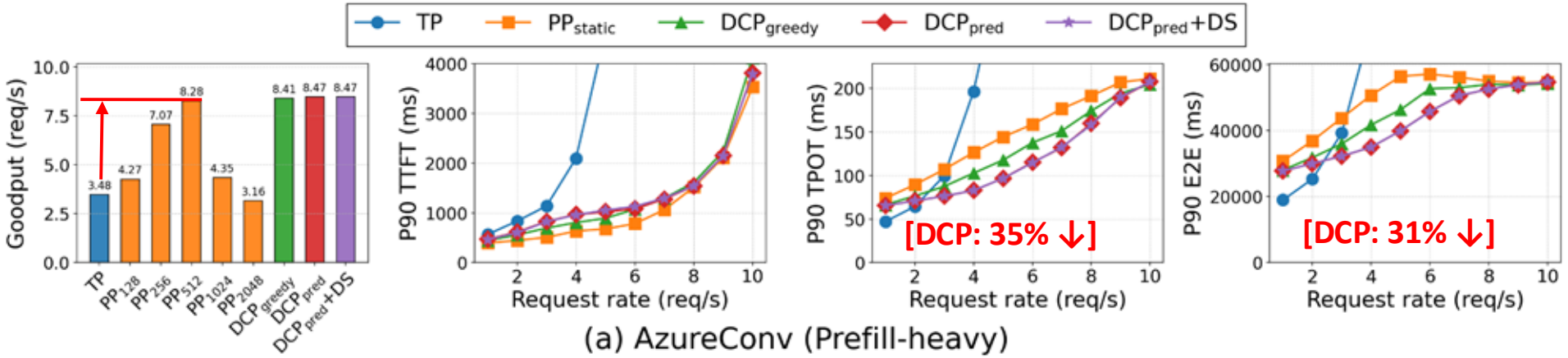
Evaluation setup

- Hardware: 4x NVIDIA A100 (40GB) PCIe 4.0, AMD EPYC 9754 CPU
- Software: SGLang v0.4.1, PyTorch v2.5.1, CUDA 12.4
- Models: Qwen 2.5-32B (FP16) (Qwen 2.5-14B is also evaluated in the paper)
- Comparisons: TP, PP, DCP (Dynamic Chunked Prefill) and DS (Delay Scheduling)
- Metrics: Goodput, TTFT (SLO: 2,000ms), TPOT (SLO: 200ms), E2E (end-to-end) latency
- Workloads: Azure Conv, ShareGPT, CNN, CNN-Reversed

Trace	Type	Avg. In/Out	Arrival Pattern
AzureConv	Prefill-heavy	1,161 / 212	Poisson, Real trace replay
ShareGPT	Prefill-heavy	226 / 199	Poisson
CNN	Prefill-heavy	883 / 65	Poisson
CNN-Reversed	Decode-heavy	65 / 883	Poisson

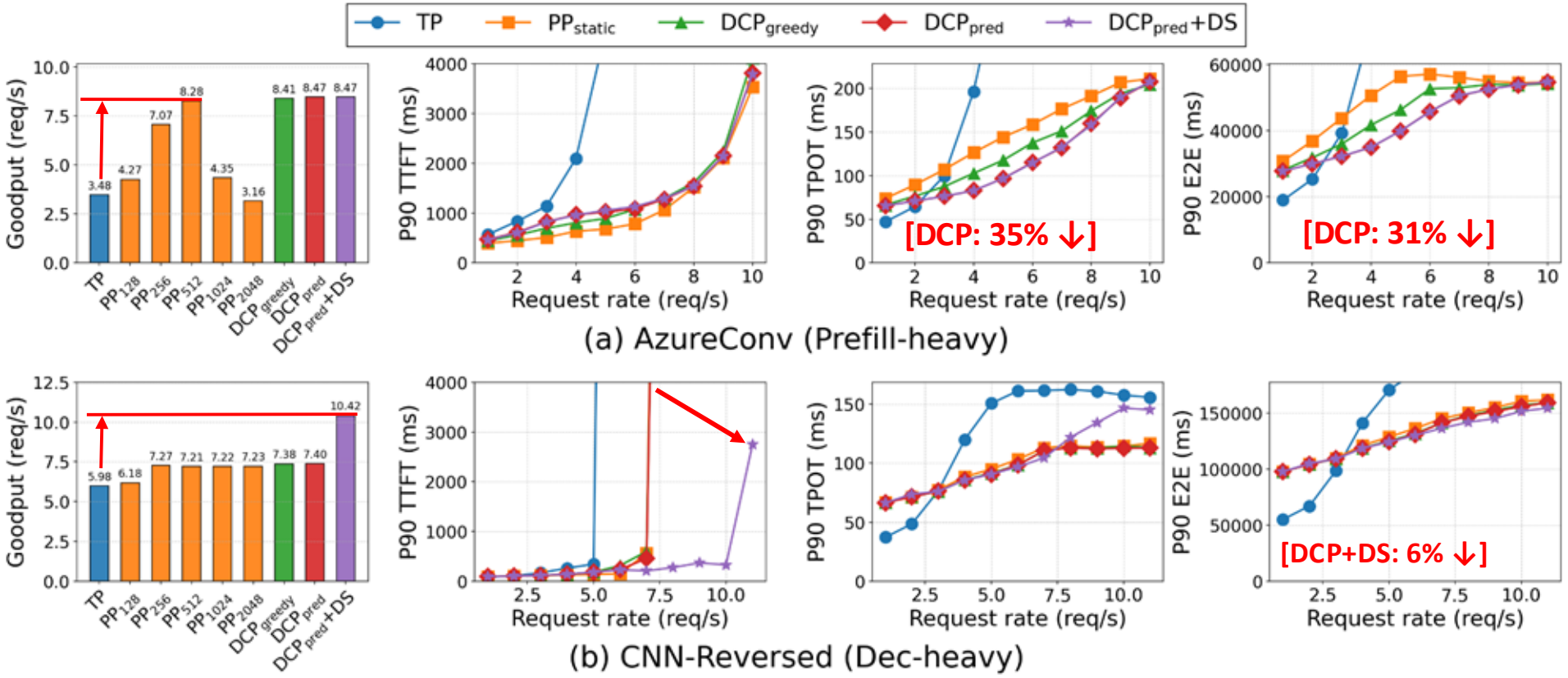
Source: MLCommons, "MLPerf Inference: Datacenter Benchmark Suite" (mlcommons.org/benchmarks/inference-datacenter) for standard online serving SLO targets (TTFT: 2,000ms, TPOT: 200ms)

Prefill-heavy: goodput & latency



- **Azure Conversation (Prefill-heavy):** DCP dramatically reduces TPOT and E2E latency
  - Eliminates prefill-induced bubbles by dynamically chunking inputs

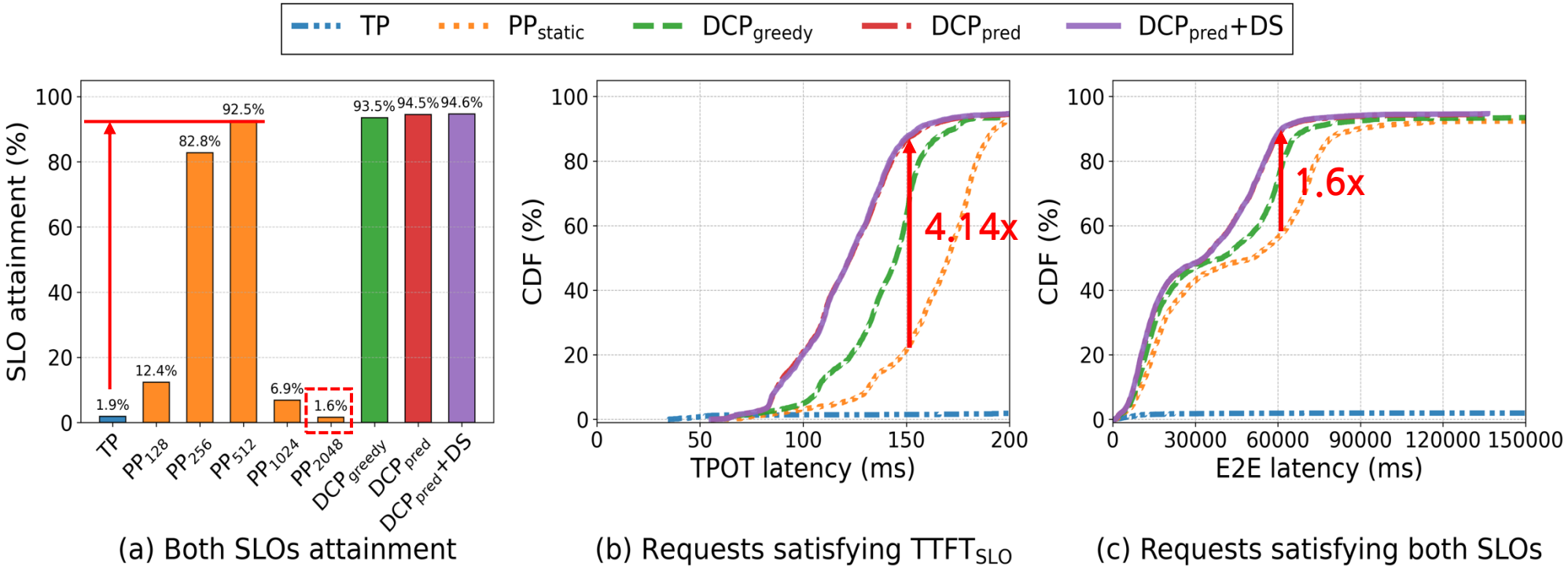
Decode-heavy: goodput & latency



- **CNN-Reversed (Dec-heavy)**: DS significantly improves goodput
  - Skips KV allocation for delayed requests & reduces bubbles for faster E2E

Replaying Azure Conversation trace

- Significantly improves request completion time within the same latency budget



(a) Both SLOs attainment

(b) Requests satisfying $TTFT_{SLO}$

(c) Requests satisfying both SLOs

Summary

- Problem
  - Naive pipeline parallelism is insufficient for online interactive LLM serving
- Our approaches
  - Dynamic chunked prefill minimizes prefill-induced pipeline bubbles
  - Delay scheduling rebalances decode computations across workers
- Results
  - Alleviating inefficiency of pipeline parallelism for LLM serving
  - Demonstrating fast LLM serving on commodity multi-GPU instances with PP
    - PCIe-based multi-GPU instances (NVIDIA and AMD) and NPUs
  - https://github.com/Sys-KU/FastPP