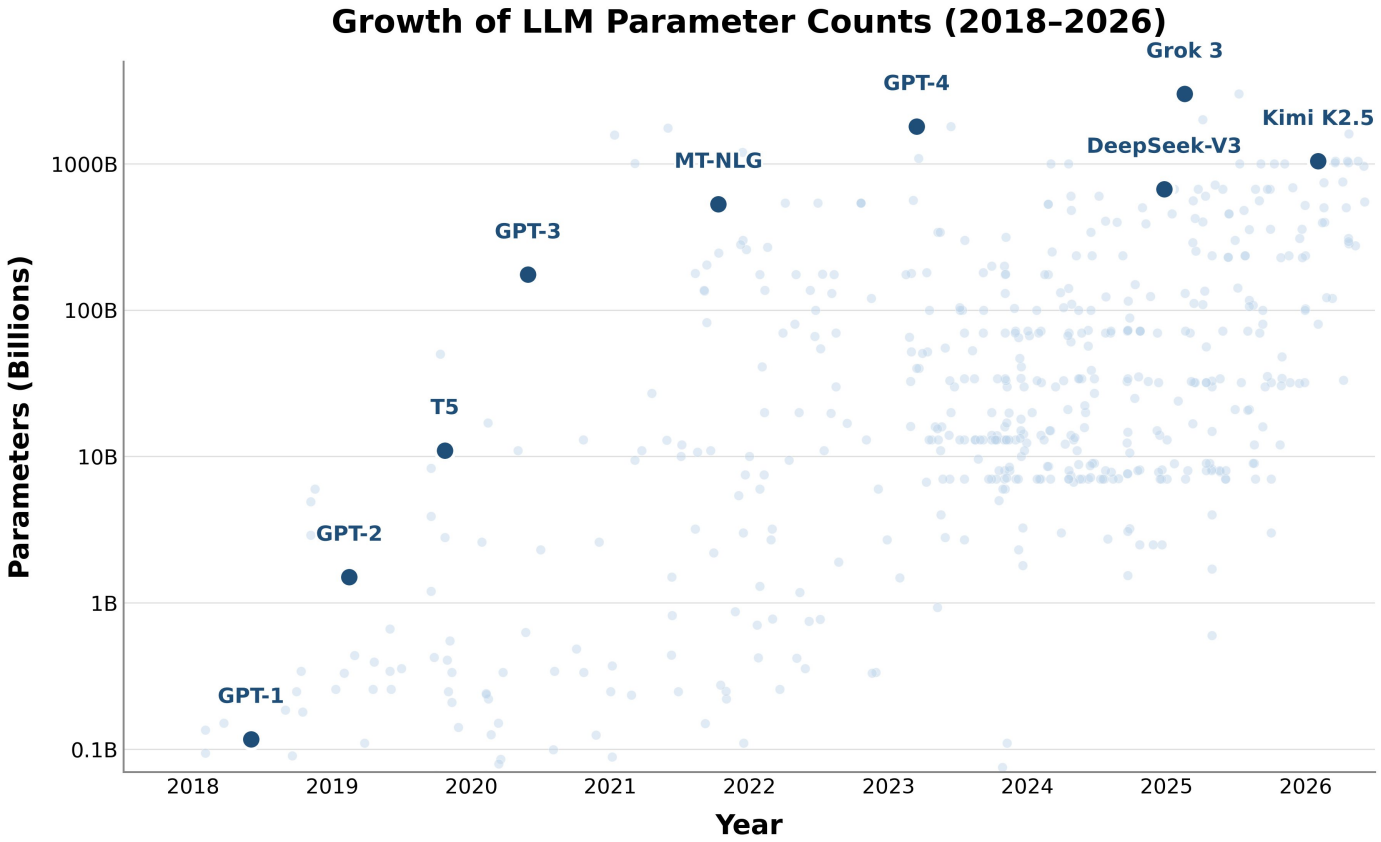
Revisiting Pipeline Parallelism for LLM Serving

Soonjae Hwang Jeongseob Ahn

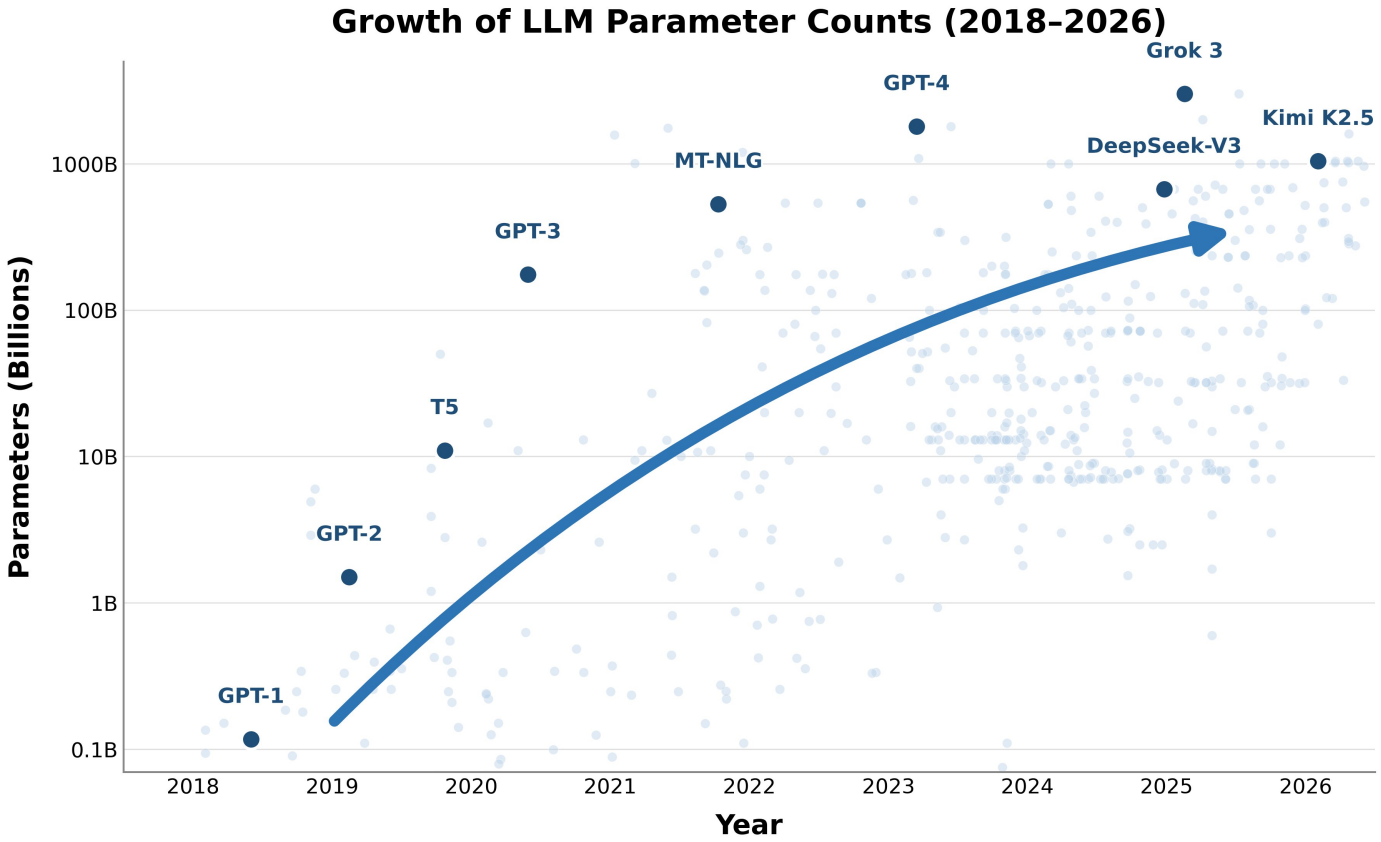


Scaling LLMs Strains Inference Systems



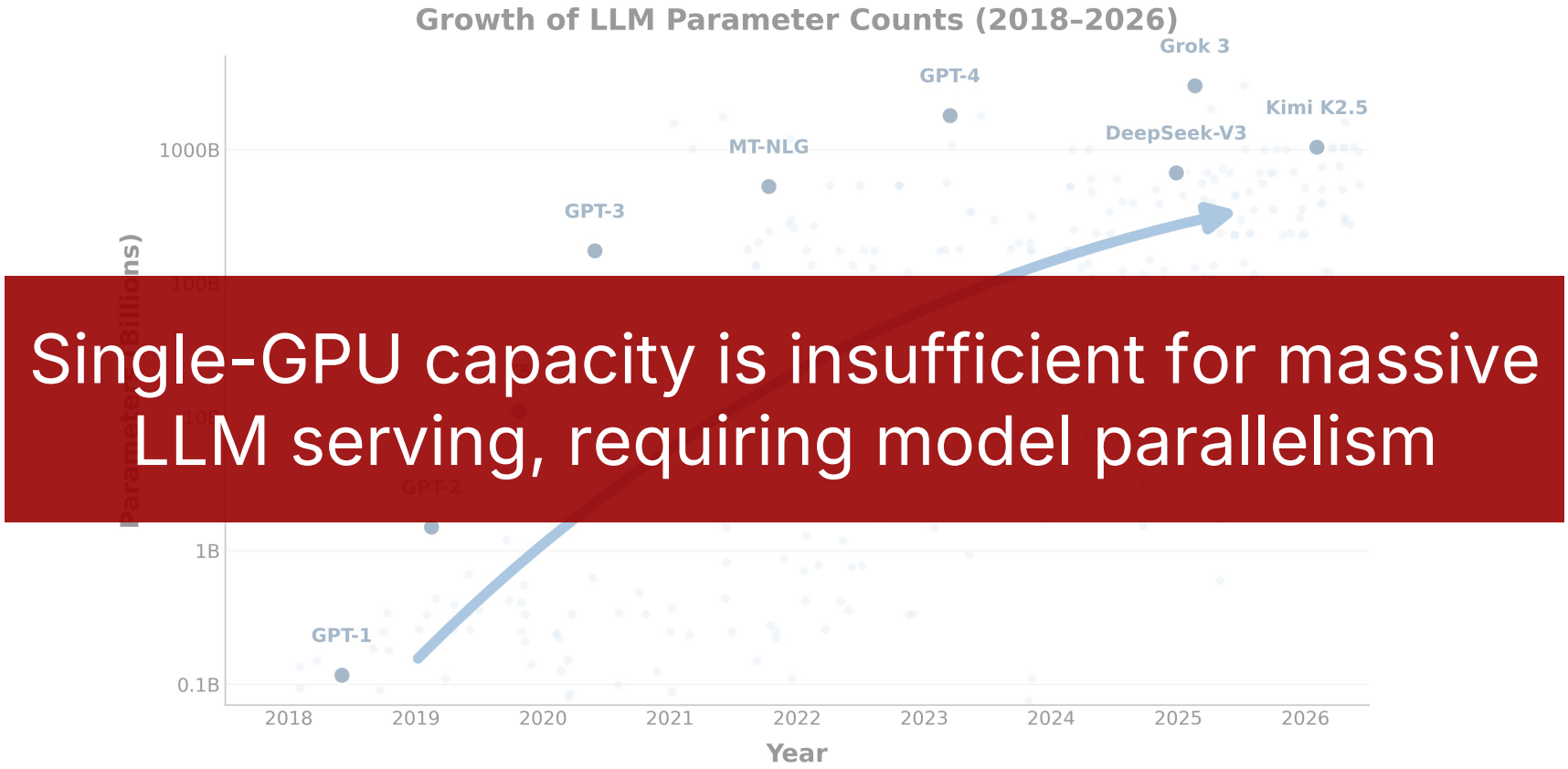
Source: Epoch AI, "Data on AI Models" (epoch.ai/data/ai-models), accessed Jun. 2026. CC-BY.

Scaling LLMs Strains Inference Systems



Source: Epoch AI, "Data on AI Models" (epoch.ai/data/ai-models), accessed Jun. 2026. CC-BY.

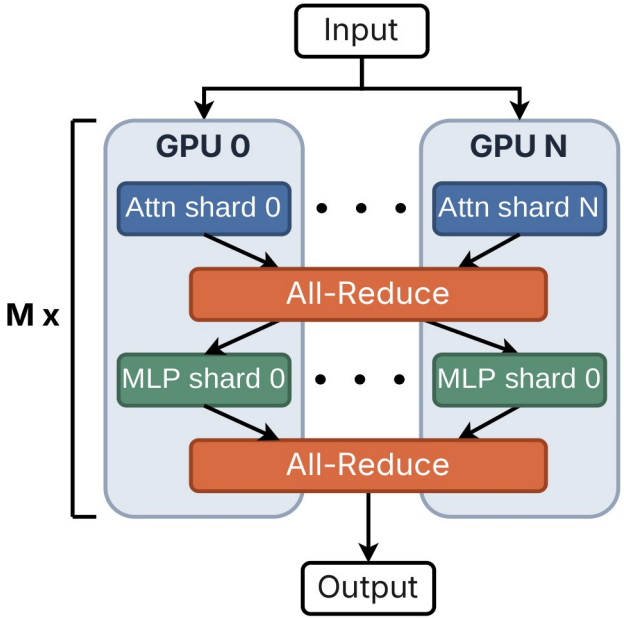
Scaling LLMs Strains Inference Systems



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

Source: Epoch AI, "Data on AI Models" (epoch.ai/data/ai-models), accessed Jun. 2026. CC-BY.

Background: Model Parallelism (TP vs. PP)

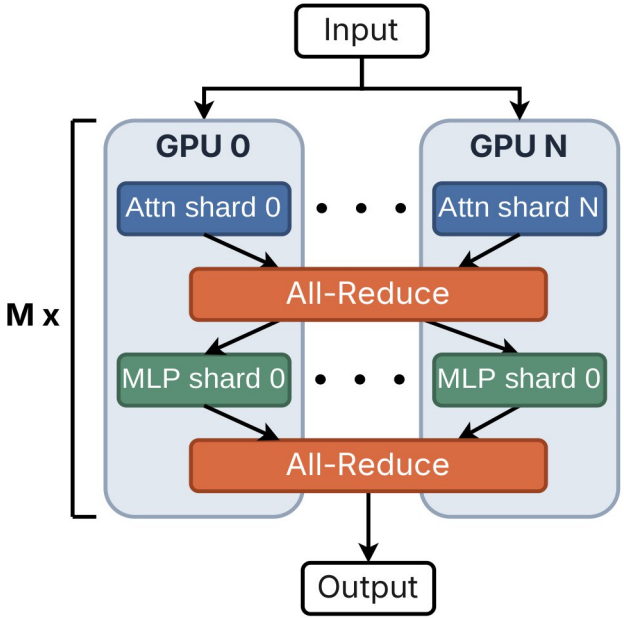


Tensor Parallelism (TP)

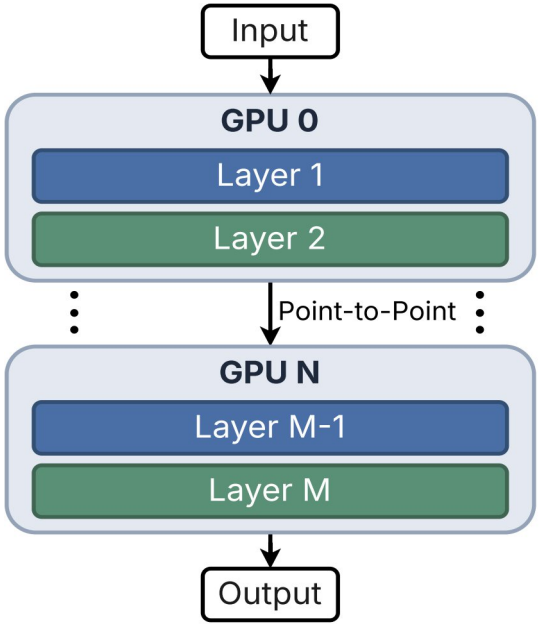
⏱ [Low latency]

⚠ High-BW needed (e.g., NVLink)

Background: Model Parallelism (TP vs. PP)



Tensor Parallelism (TP)



Pipeline Parallelism (PP)

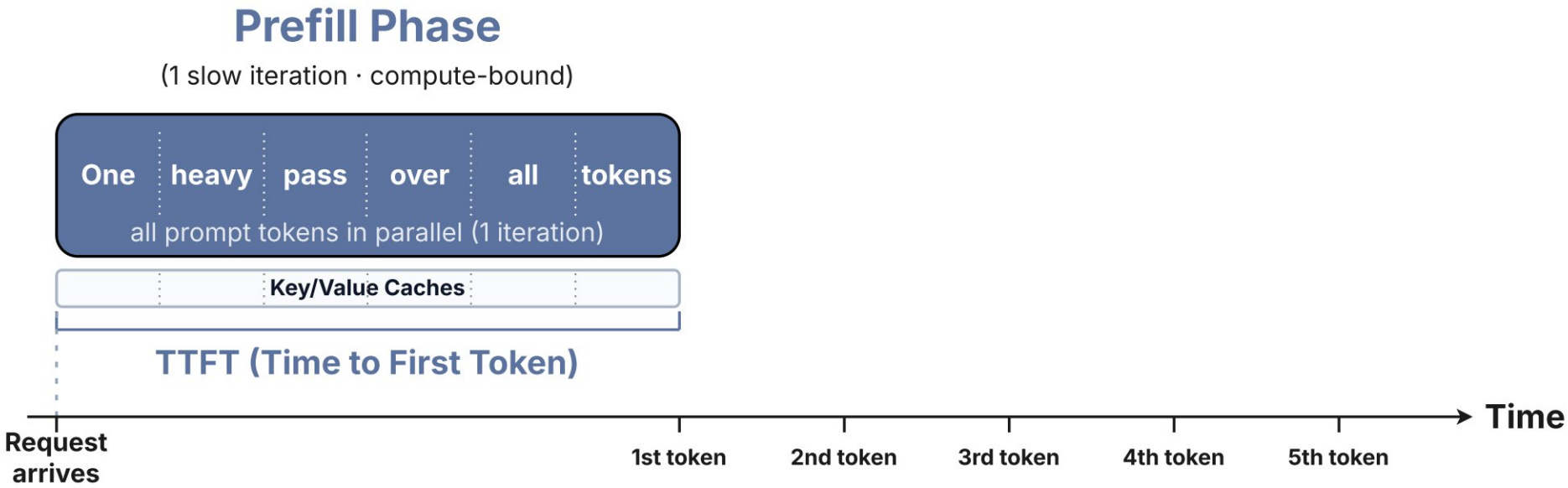
⏱ [Low latency]

⚠ High-BW needed (e.g., NVLink)

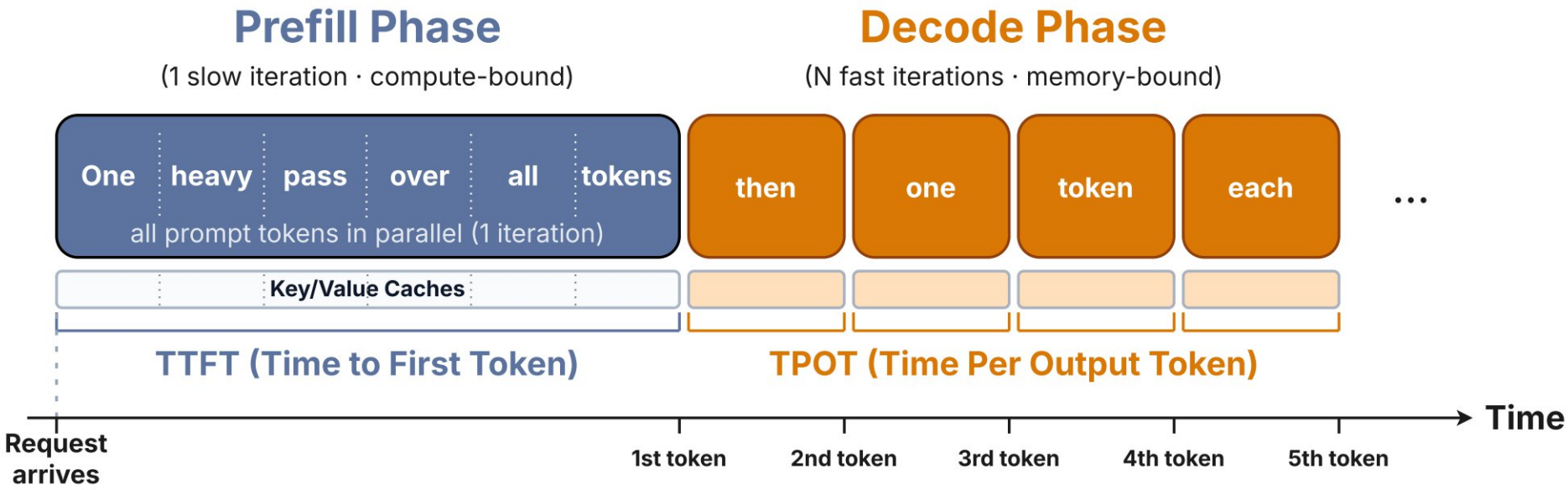
🚀 [High throughput]

✓ Low-BW friendly (e.g., PCIe)

Background: LLM Inference



Background: LLM Inference



Limitations of Throughput as a Serving Metric

Offline Batch Inference

Throughput
= Completed Requests / Time = 10 req/s

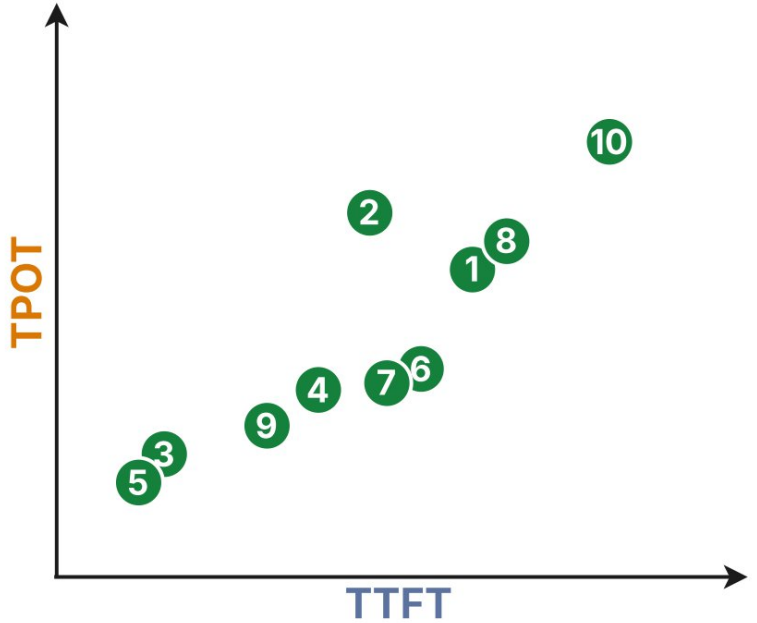


= One request = SLO box

Limitations of Throughput as a Serving Metric

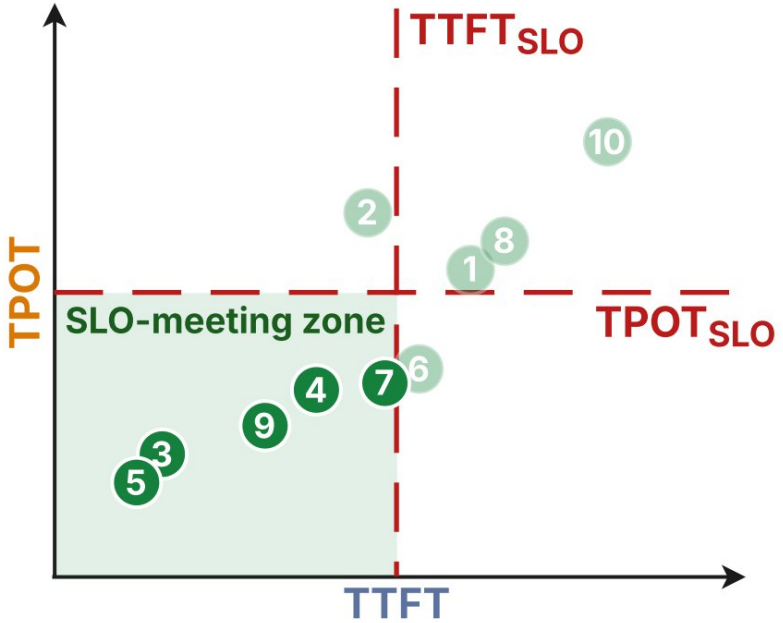
Offline Batch Inference

Throughput
= Completed Requests / Time = **10 req/s**



Online Interactive Serving

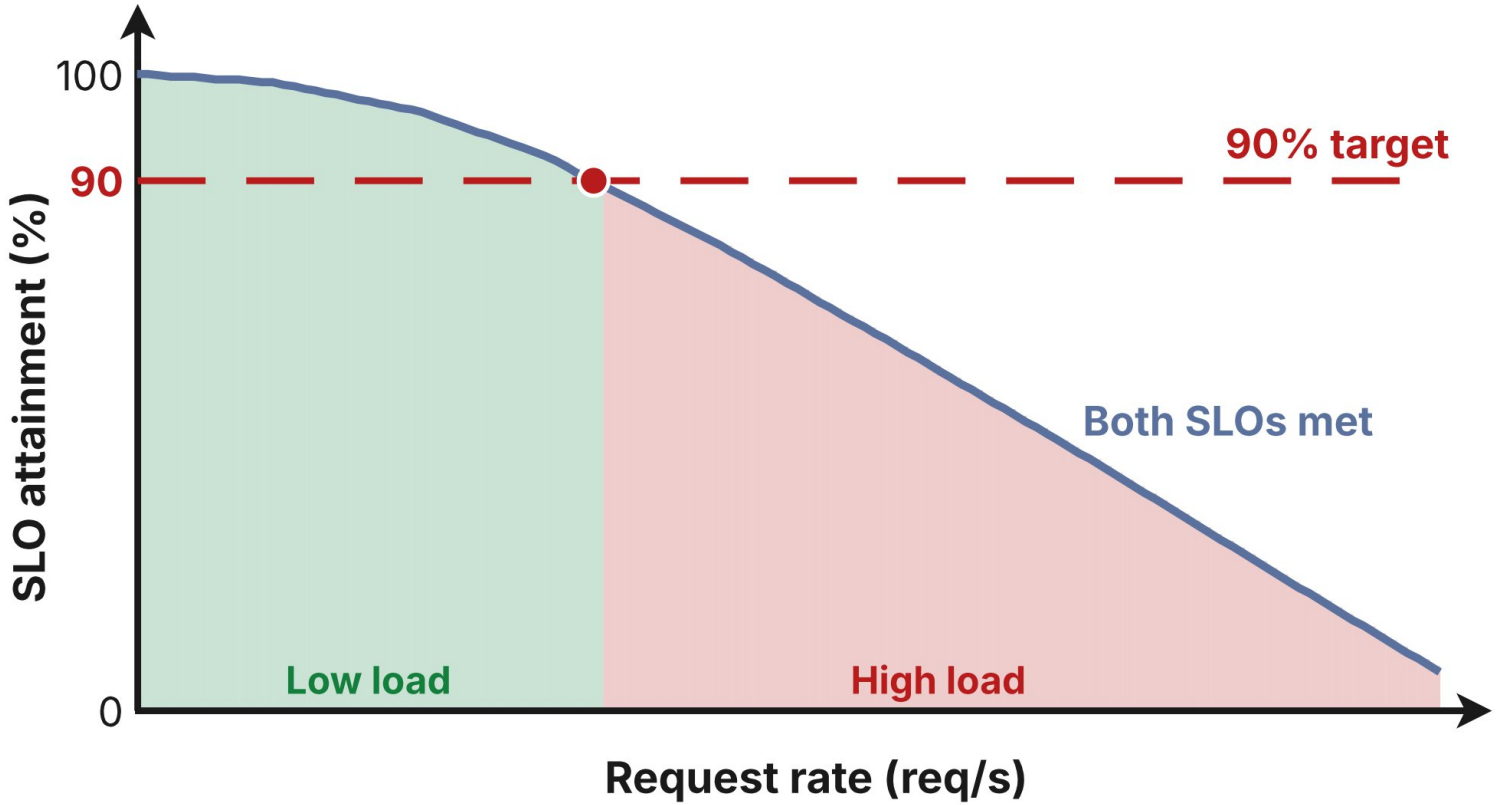
Effective Throughput
= SLO-meeting Requests / Time = **5 req/s**



i = One request = SLO box

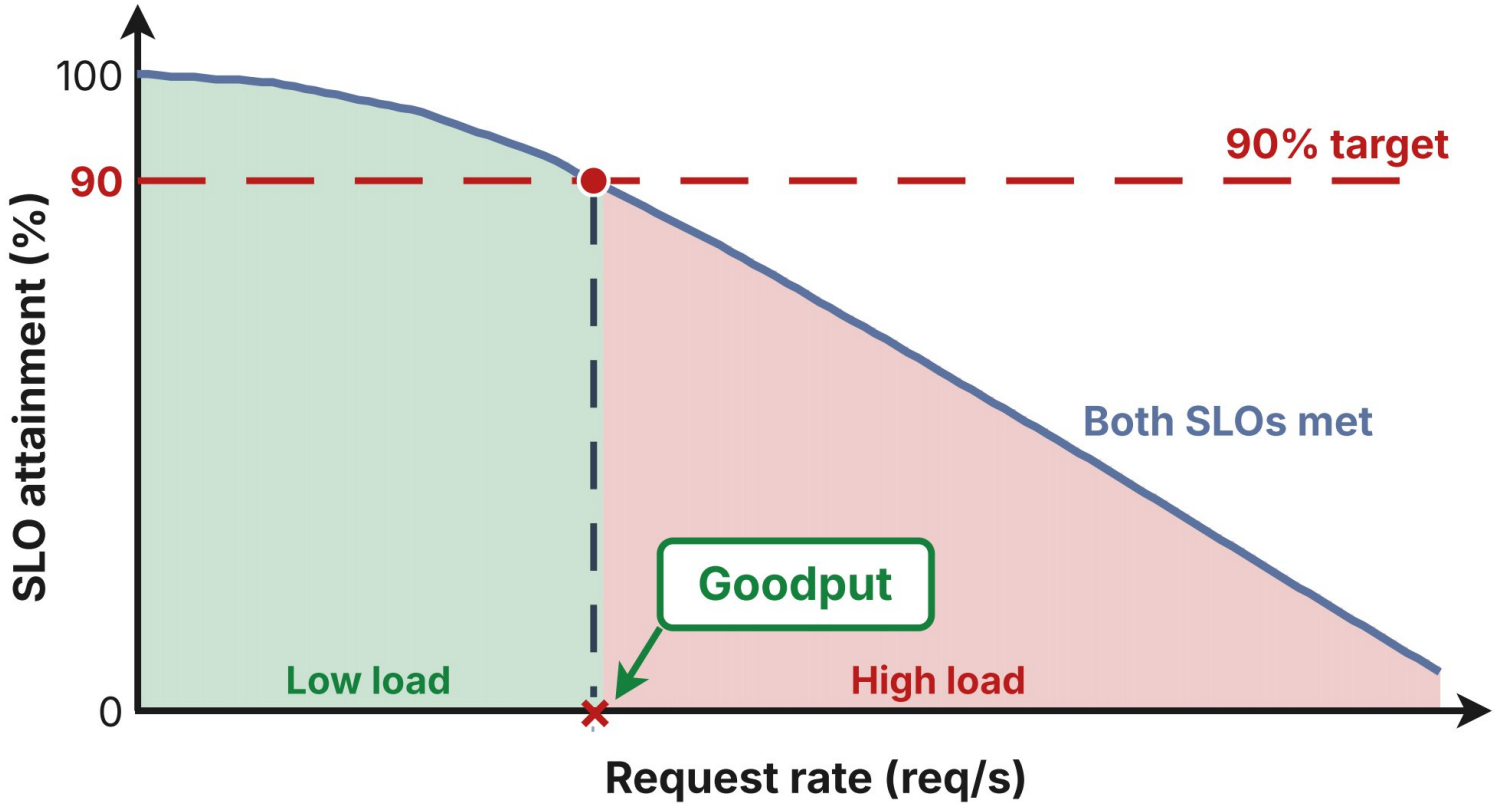
From Throughput to Goodput

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



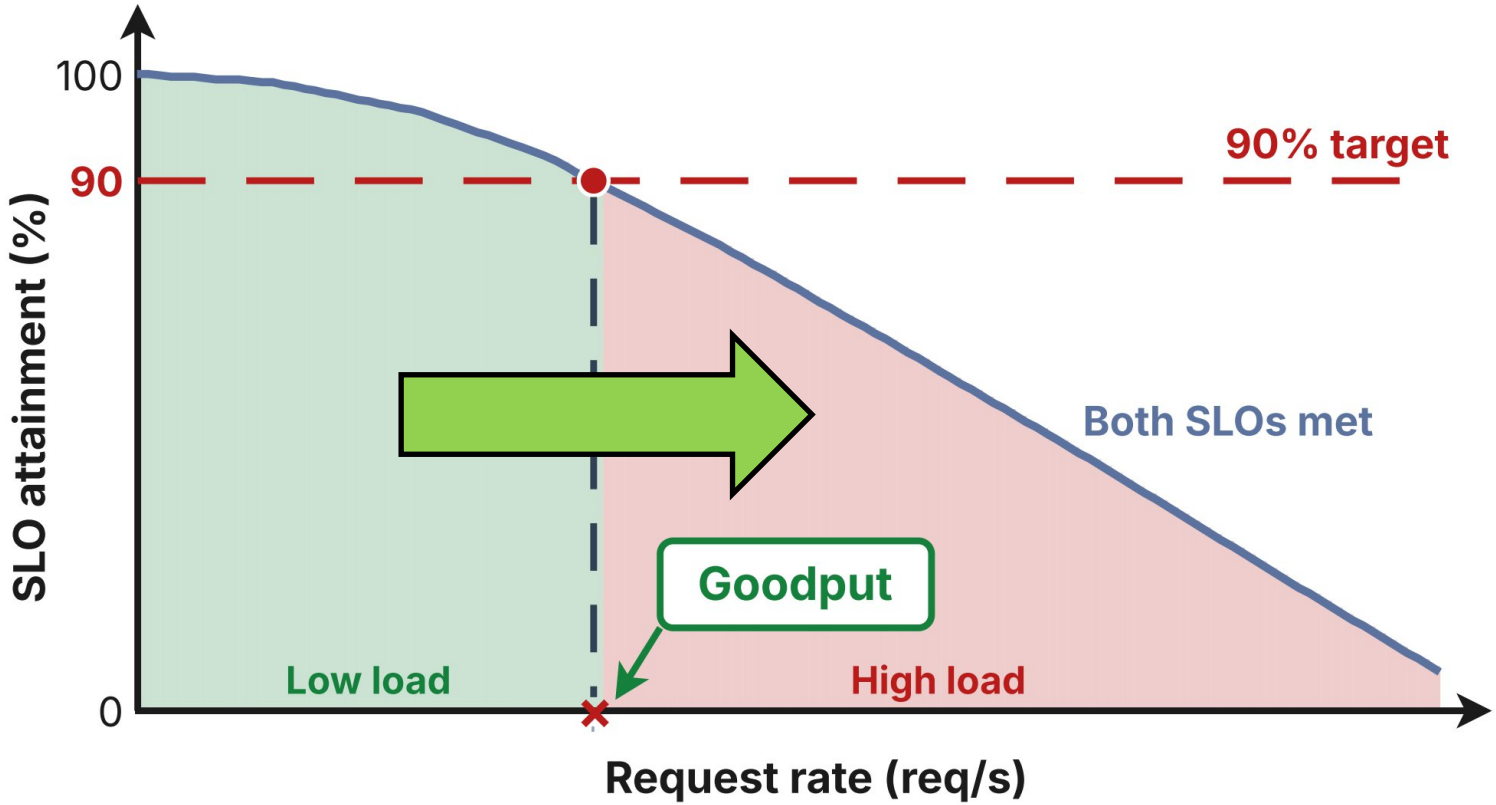
From Throughput to Goodput

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



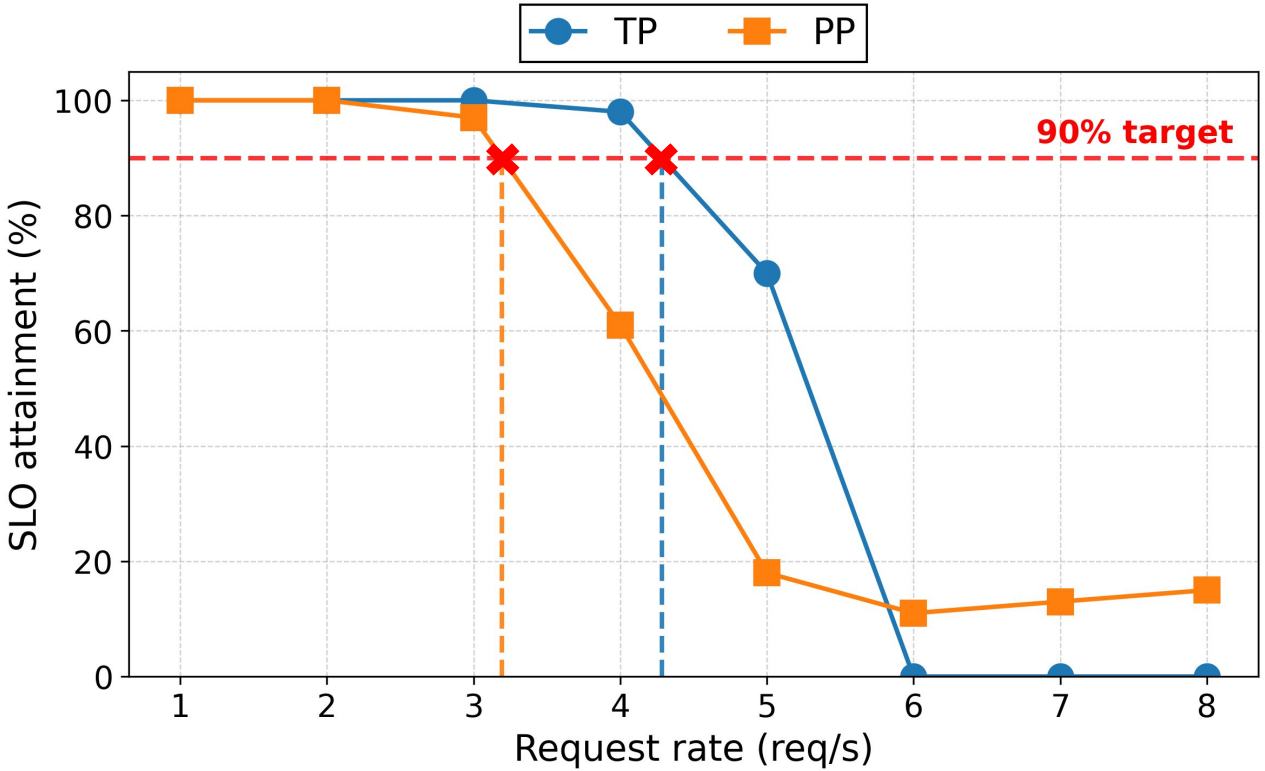
From Throughput to Goodput

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



Goodput Degradation of PP over PCIe

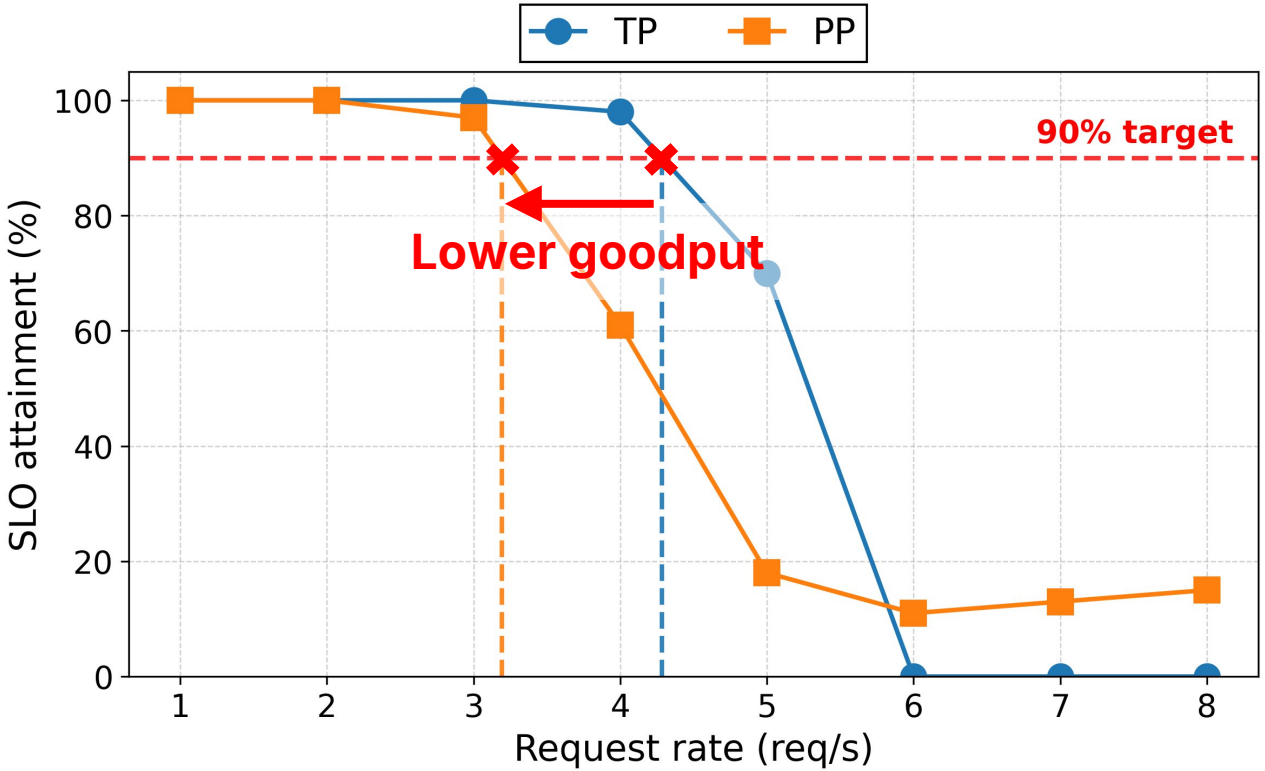
PP fails to match TP's goodput despite its communication advantages



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

Goodput Degradation of PP over PCIe

PP fails to match TP's goodput despite its communication advantages



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

Why does PP suffer from low goodput even in a PCIe environment?

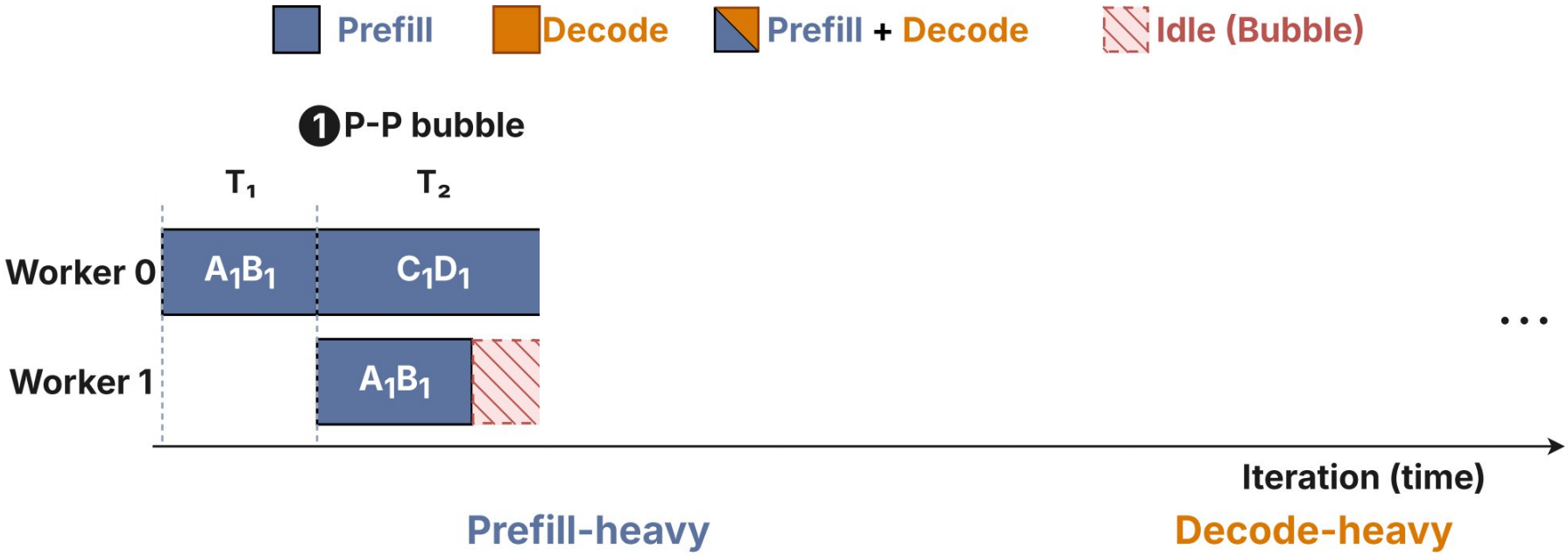
Why does PP suffer from low goodput even in a PCIe environment?



Pipeline Bubble

Challenge: Pipeline Bubbles

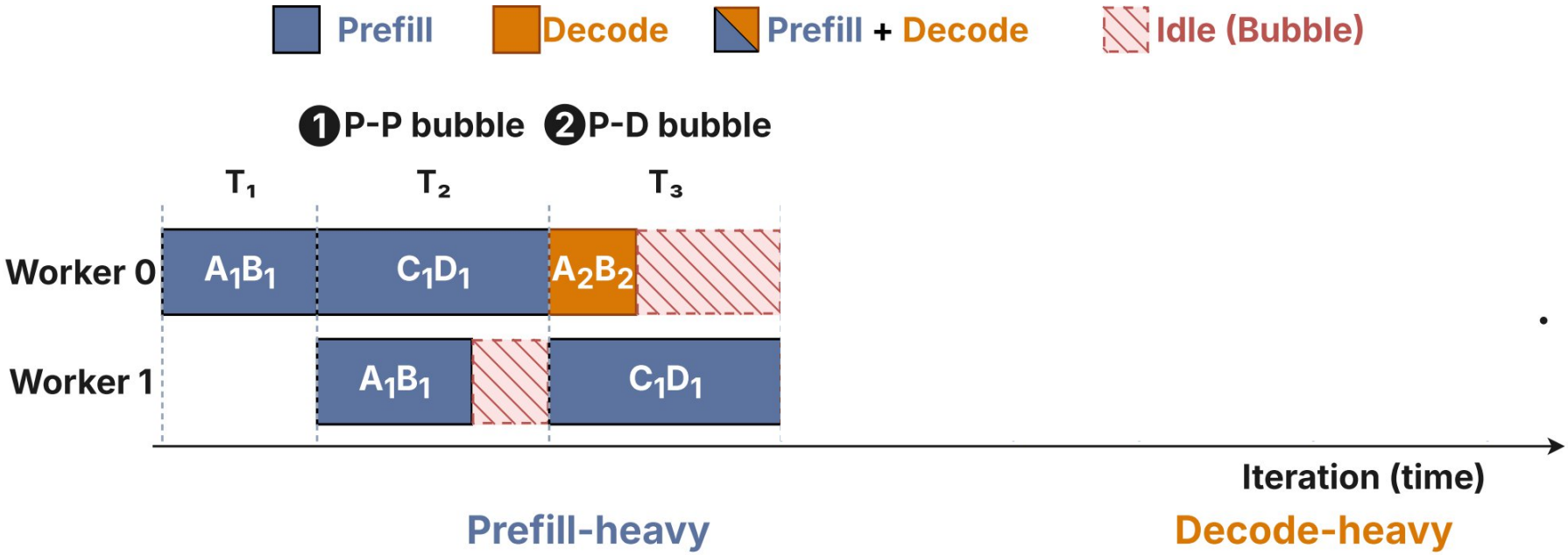
Diverse request patterns trigger severe execution imbalances



A1B1: two requests A and B in the same batch and the subscript indicates the iteration number

Challenge: Pipeline Bubbles

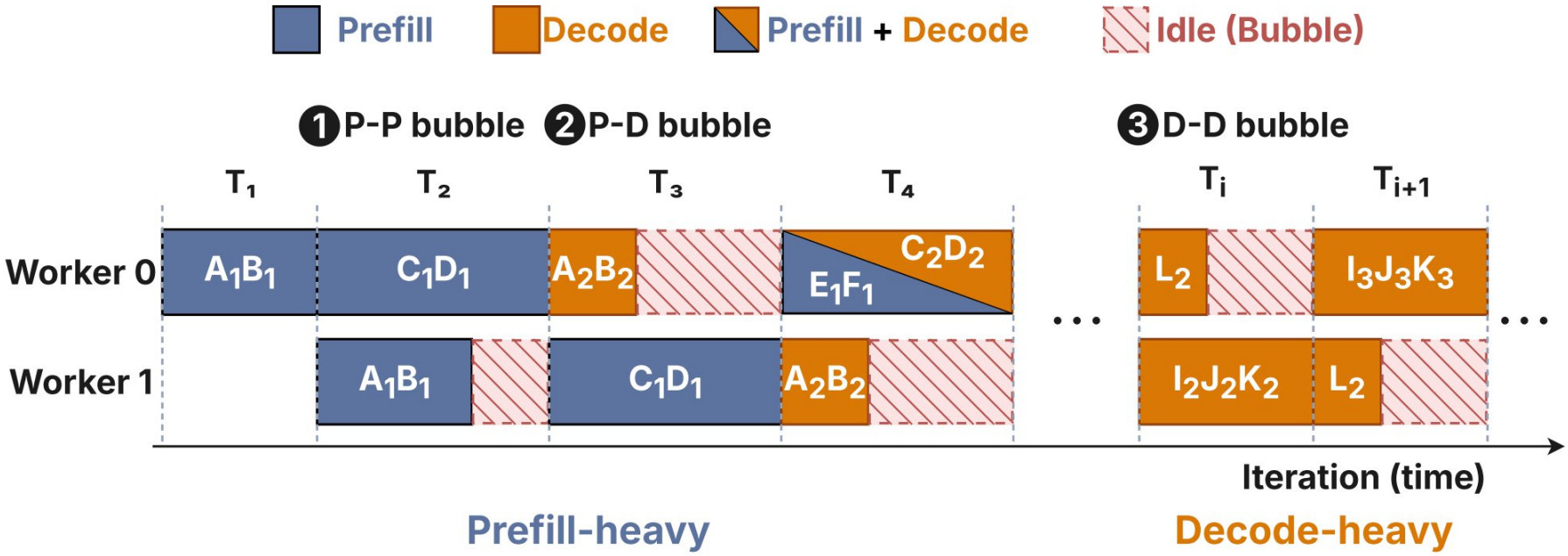
Diverse request patterns trigger severe execution imbalances



A1B1: two requests A and B in the same batch and the subscript indicates the iteration number

Challenge: Pipeline Bubbles

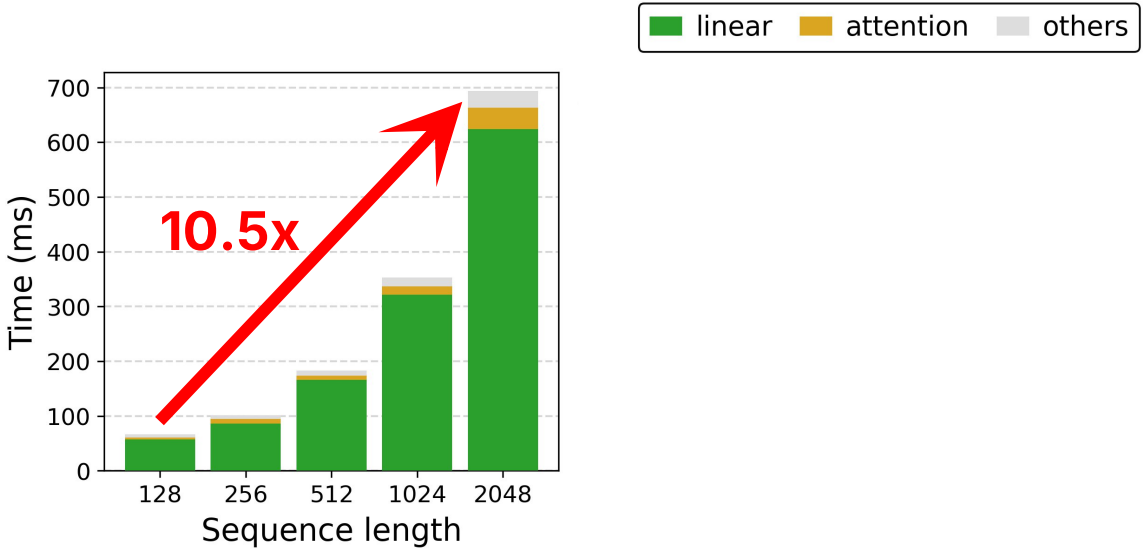
Diverse request patterns trigger severe execution imbalances



A1B1: two requests A and B in the same batch and the subscript indicates the iteration number

Root Cause of Pipeline Bubbles: Prefill Phase

Prefill latency scales drastically with input sequence length

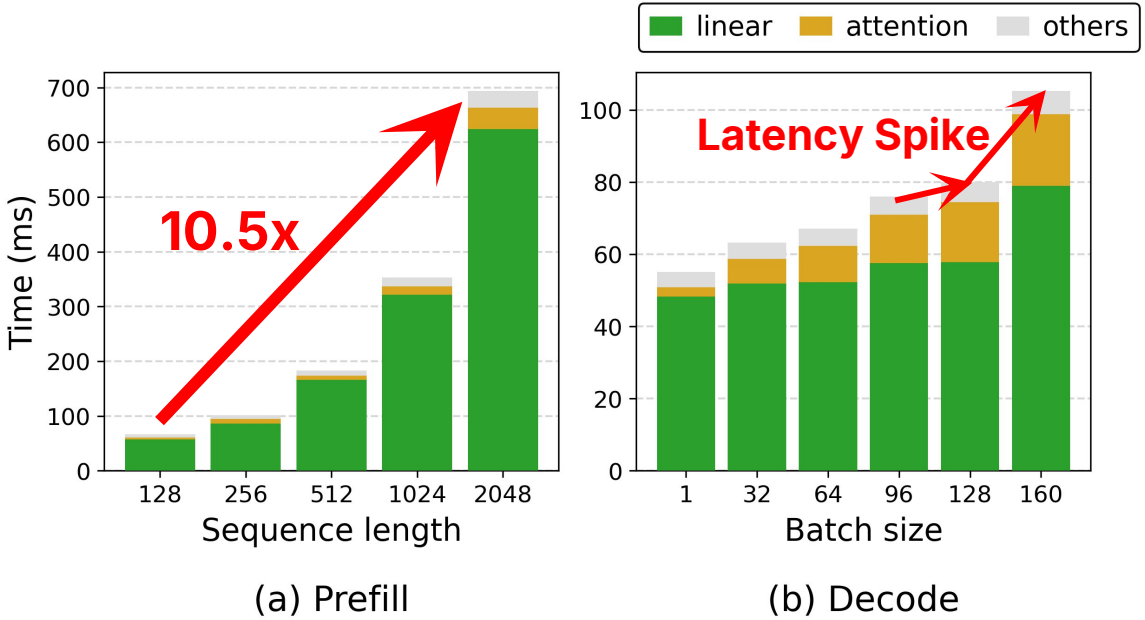


(a) Prefill

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

Root Cause of Pipeline Bubbles: Decode Phase

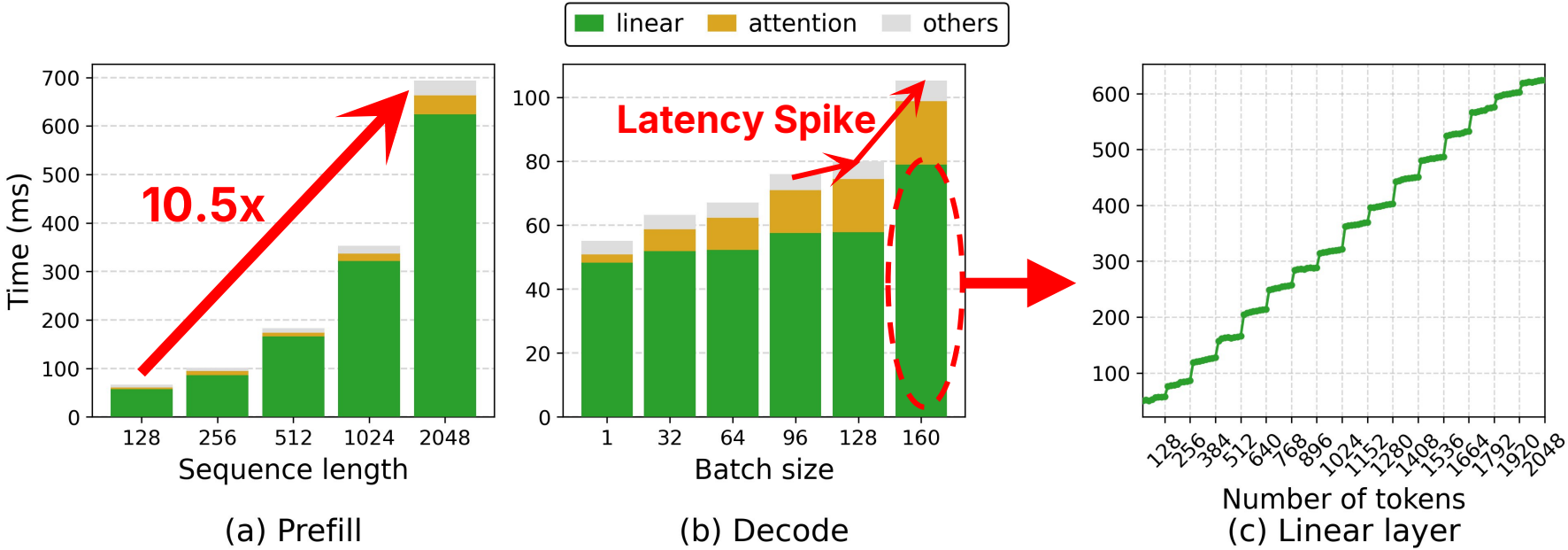
Decode latency grows with batch size and jumps sharply beyond 128



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

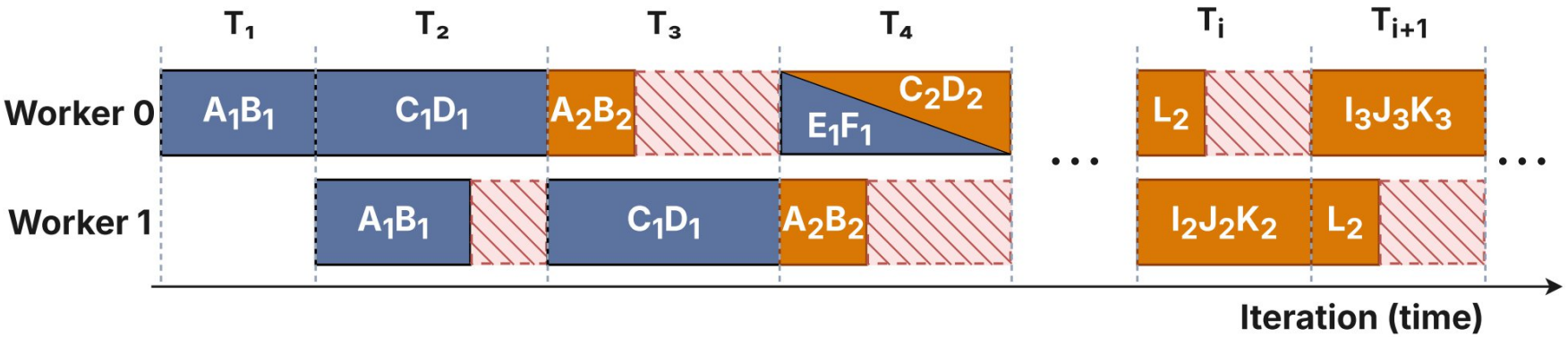
Root Cause of Pipeline Bubbles: Decode Phase

Caused by GPU kernel tiling at 128-token boundaries



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

Our Approach: Chunking & Rebalancing

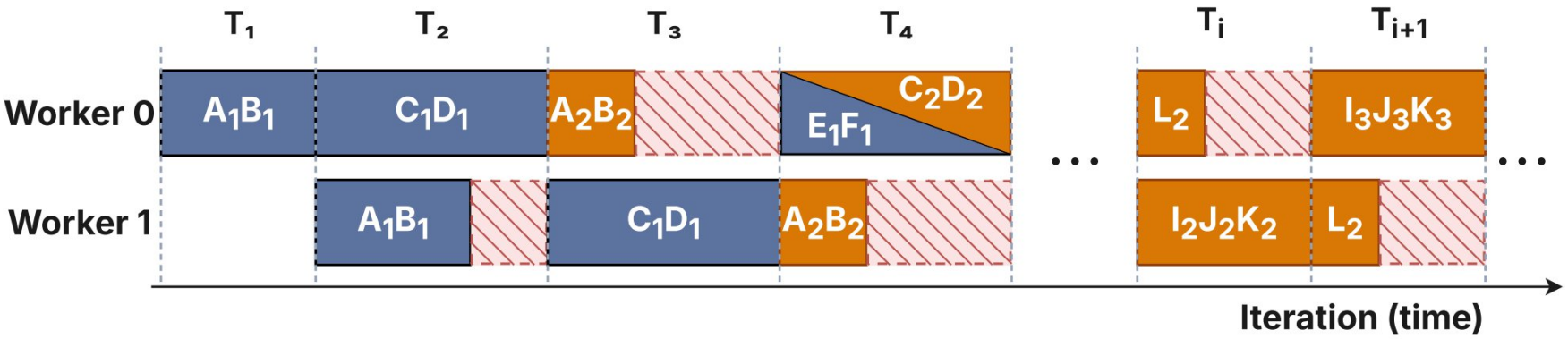


Prefill-heavy

↓

Dynamic Chunked Prefill

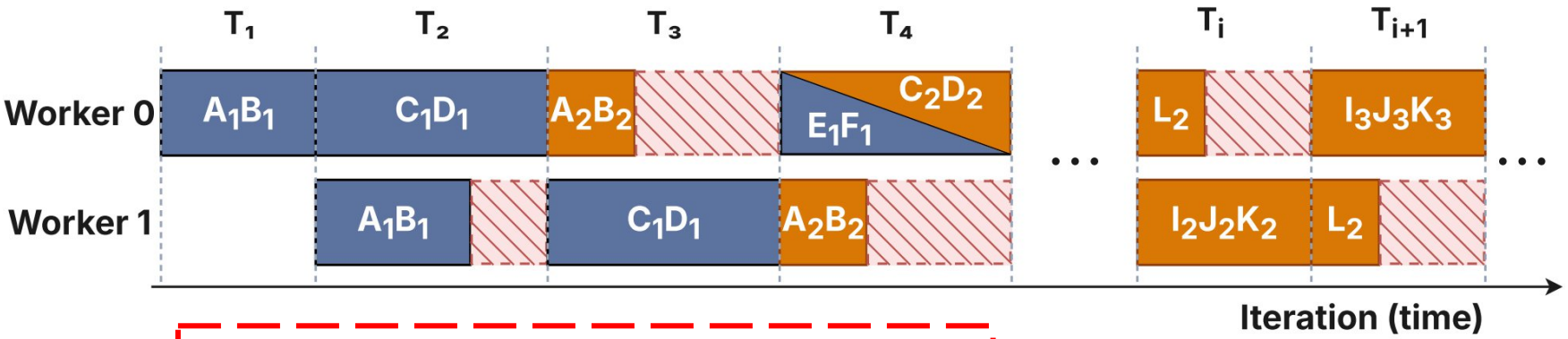
Our Approach: Chunking & Rebalancing



Prefill-heavy → **Dynamic Chunked Prefill**

Decode-heavy → **Delay Scheduling**

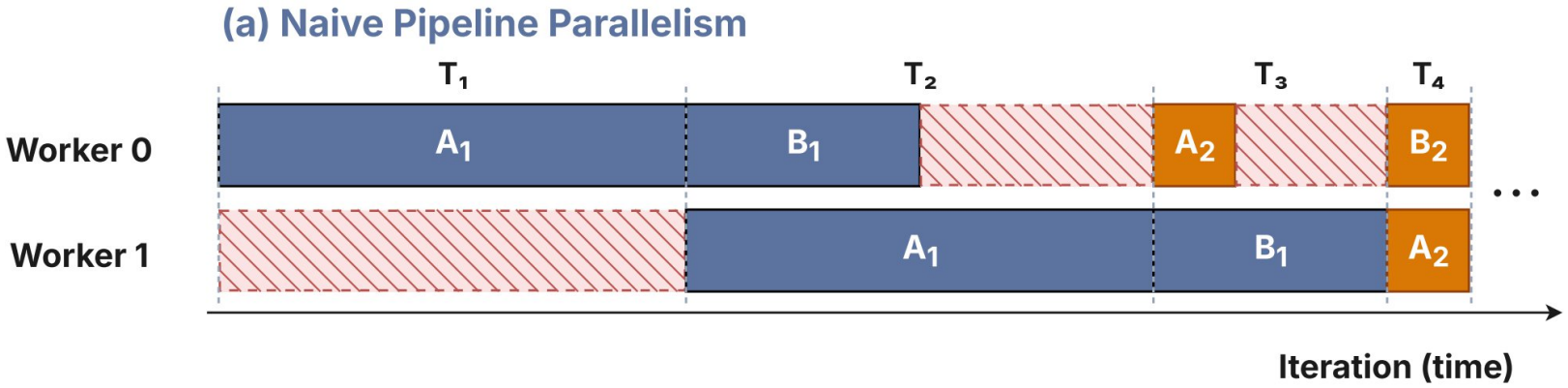
Our Approach: Chunking & Rebalancing



Prefill-heavy → **Dynamic Chunked Prefill**

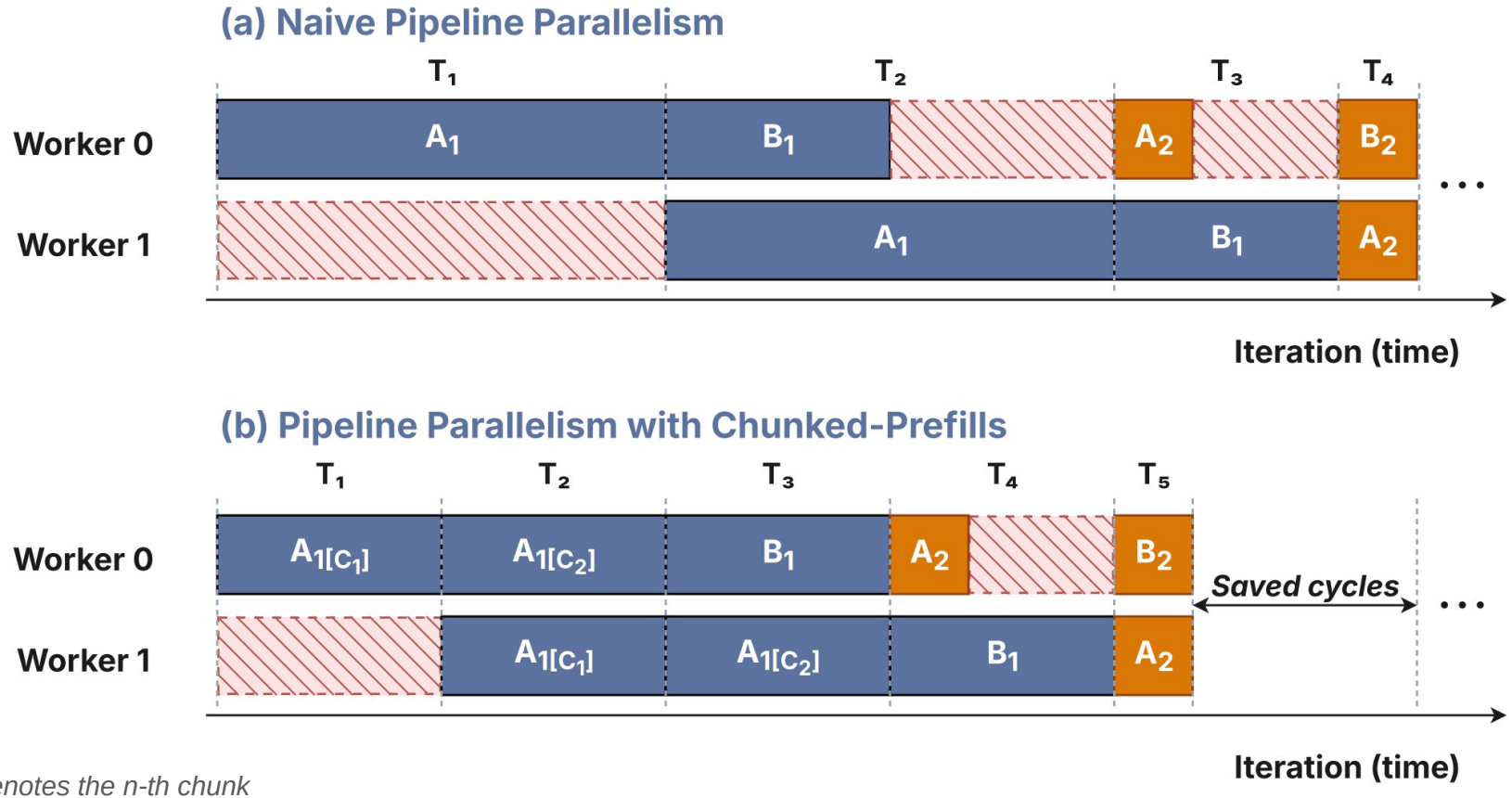
Decode-heavy → **Delay Scheduling**

Chunked Prefill in Pipeline Parallelism



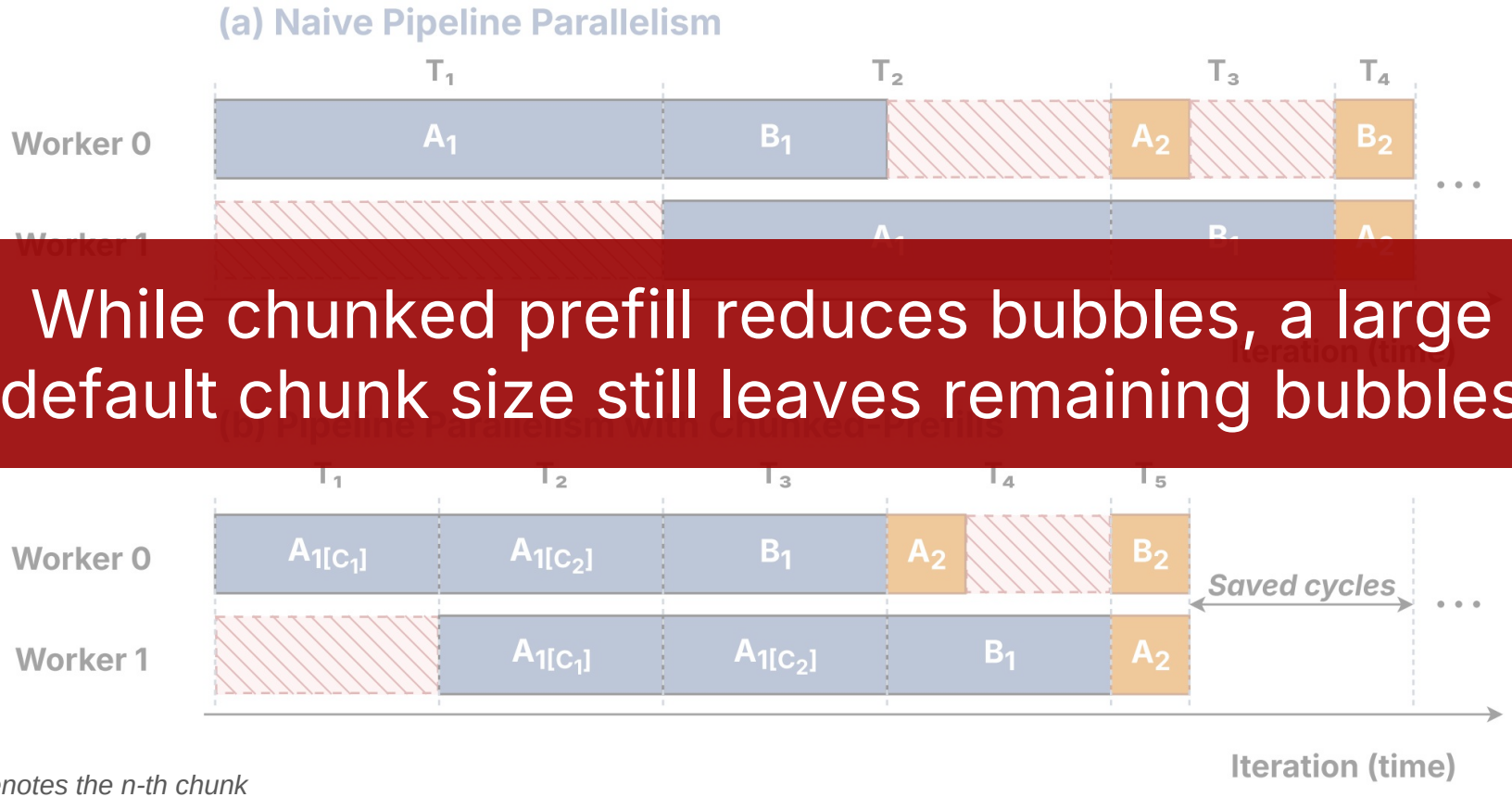
[C_n] denotes the n-th chunk

Chunked Prefill in Pipeline Parallelism



[C_n] denotes the n-th chunk

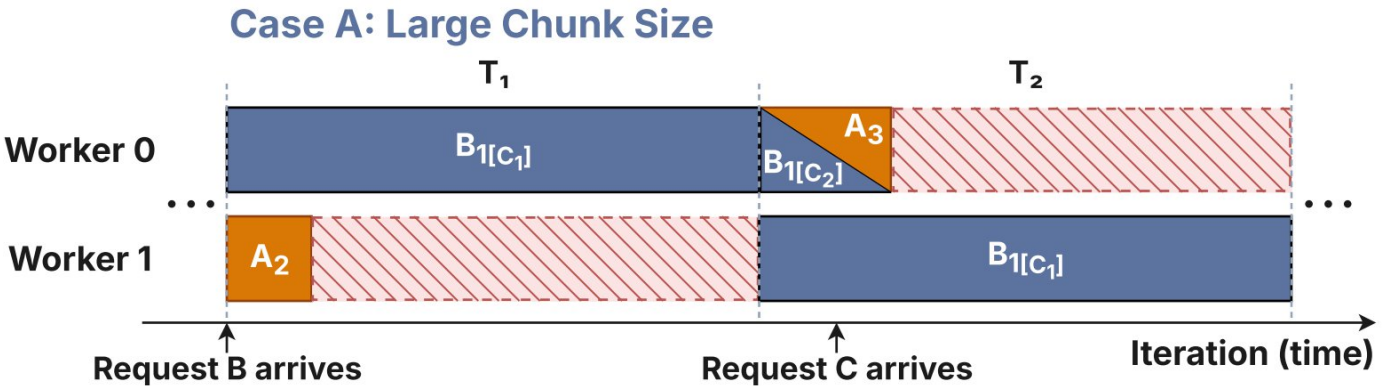
Chunked Prefill in Pipeline Parallelism



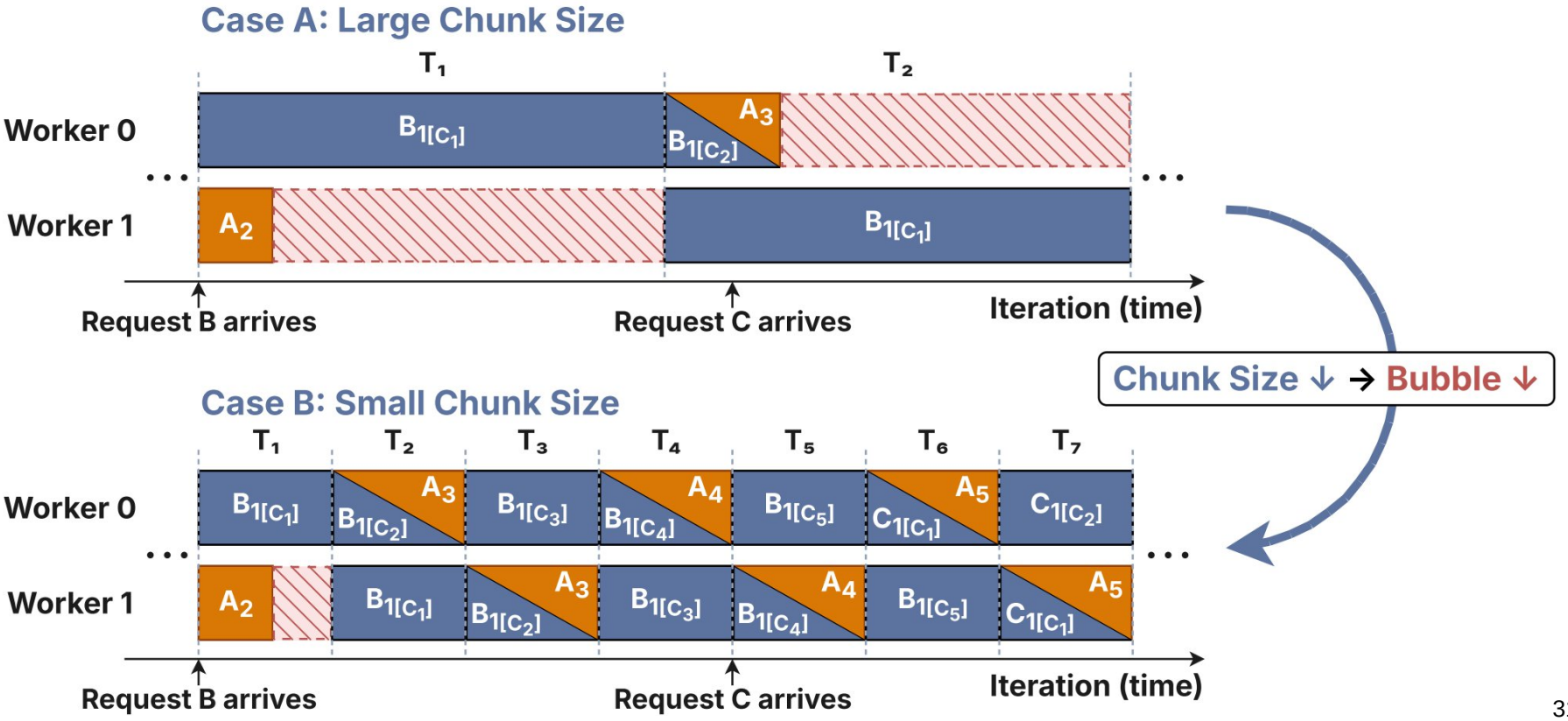
While chunked prefill reduces bubbles, a large default chunk size still leaves remaining bubbles

[C_n] denotes the n-th chunk

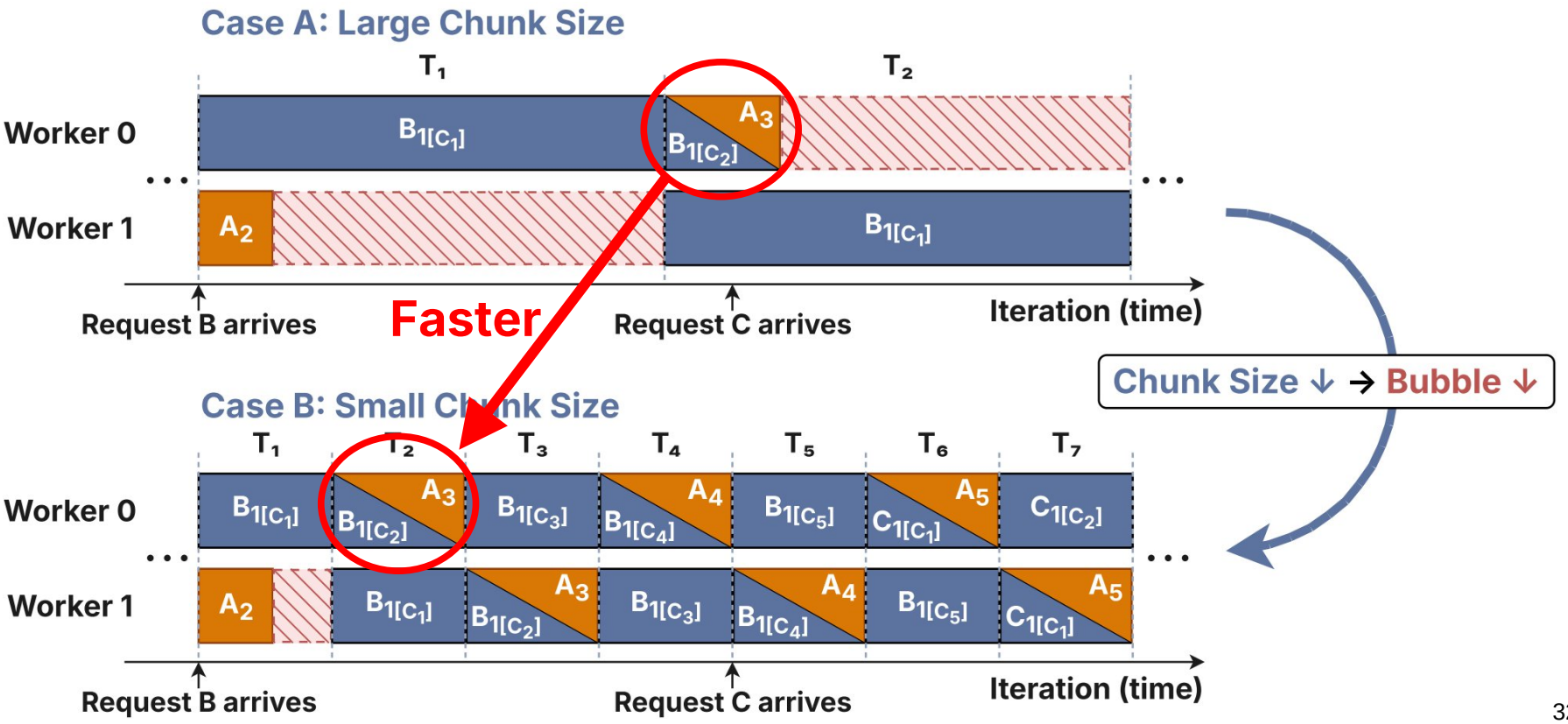
Observation #1: Chunk Size Under Low Load



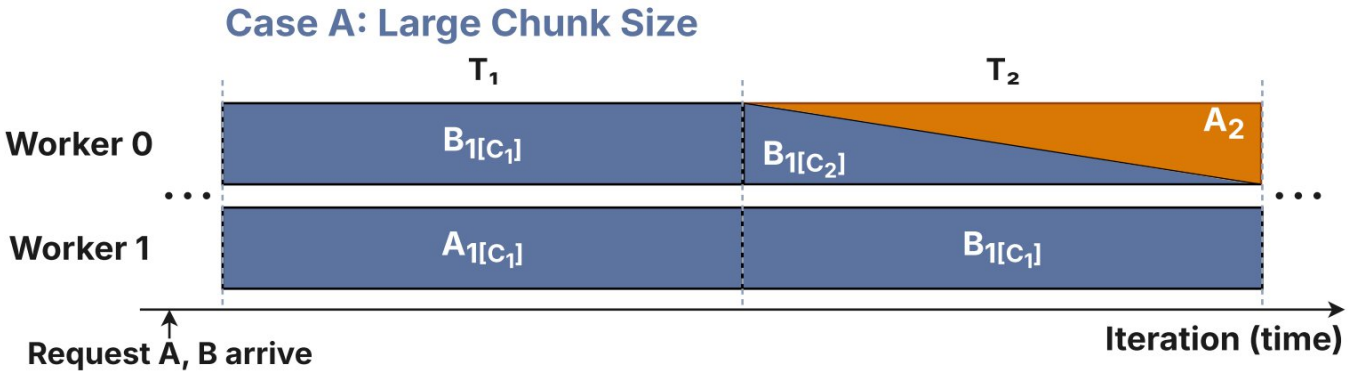
Observation #1: Chunk Size Under Low Load



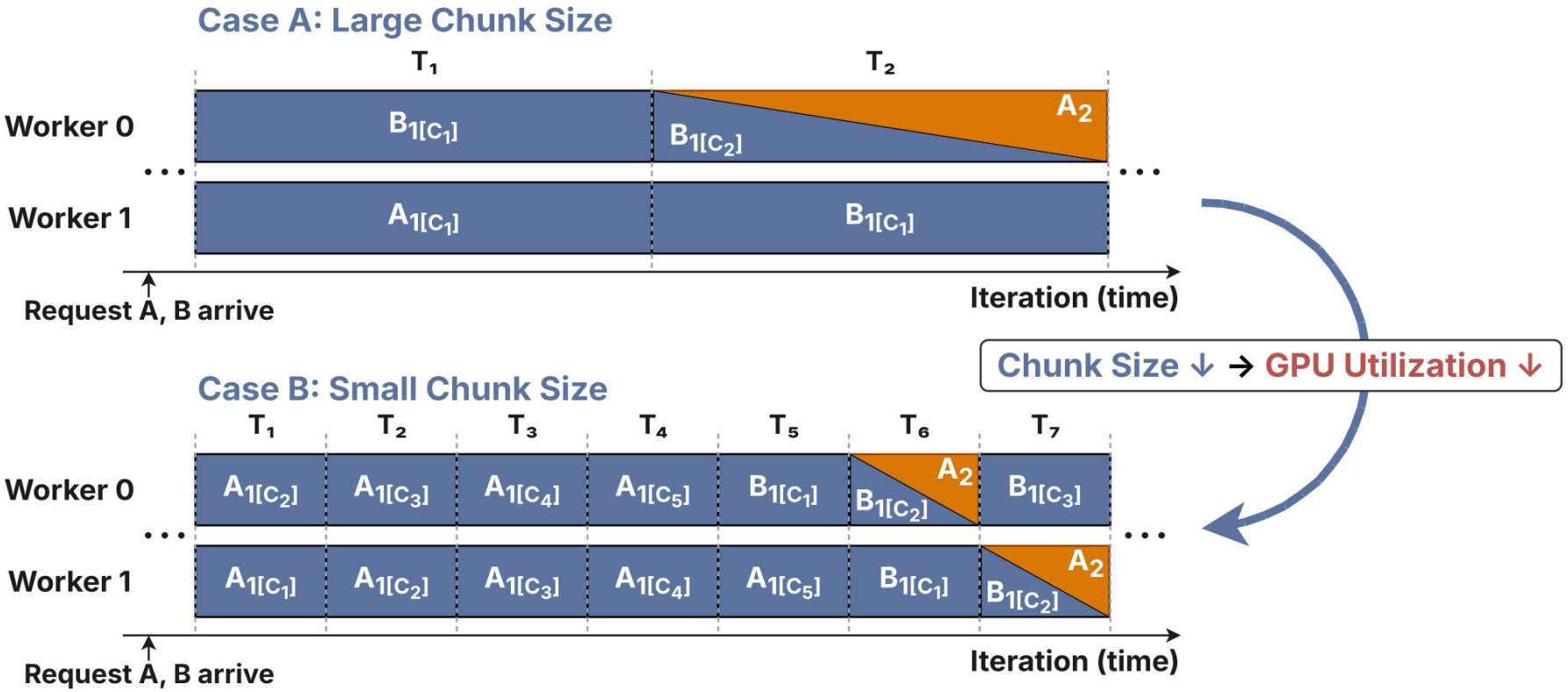
Observation #1: Chunk Size Under Low Load



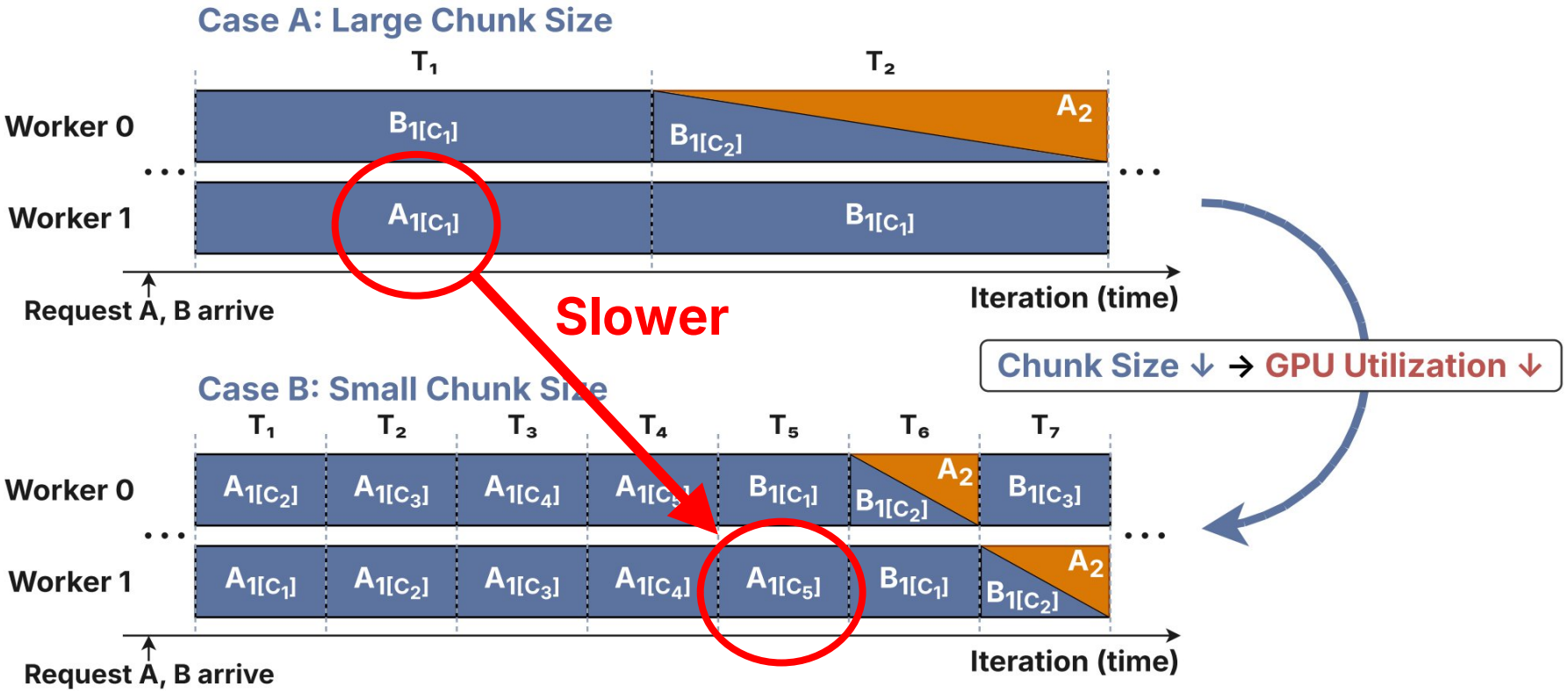
Observation #2: Chunk Size Under High Load



Observation #2: Chunk Size Under High Load



Observation #2: Chunk Size Under High Load



Let’s adjust the chunk size dynamically based on request load 💡

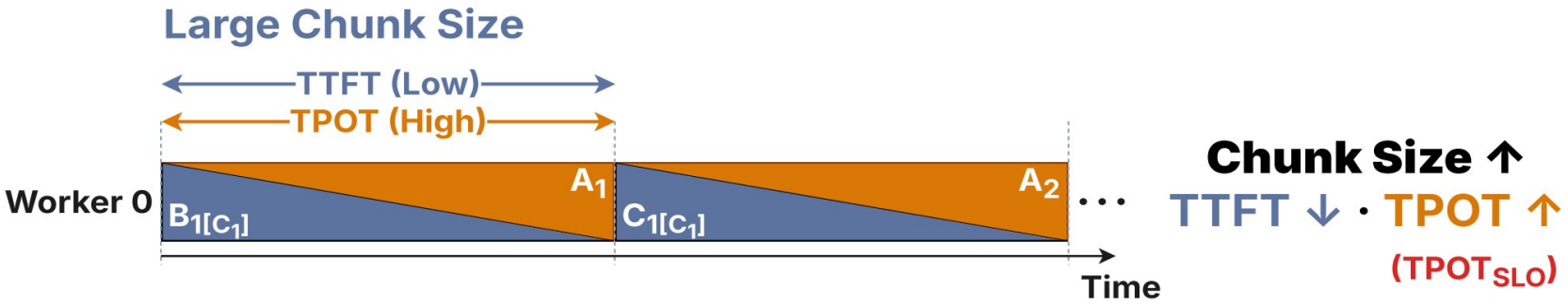
Let’s adjust the chunk size dynamically based on request load 💡

But how?

SLO-Driven Chunk Size Adaptation

Deriving upper and lower bounds based on SLOs

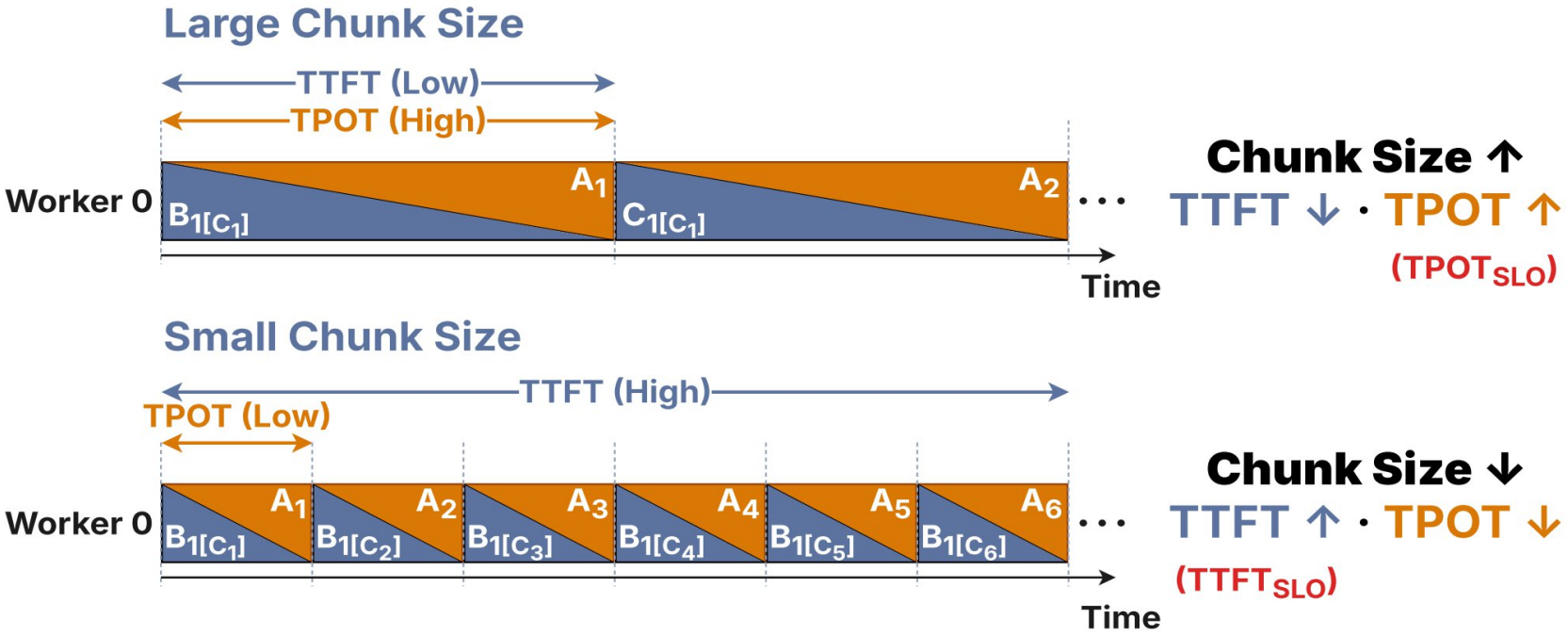
→Prioritize the lower-bound for bubble minimization



SLO-Driven Chunk Size Adaptation

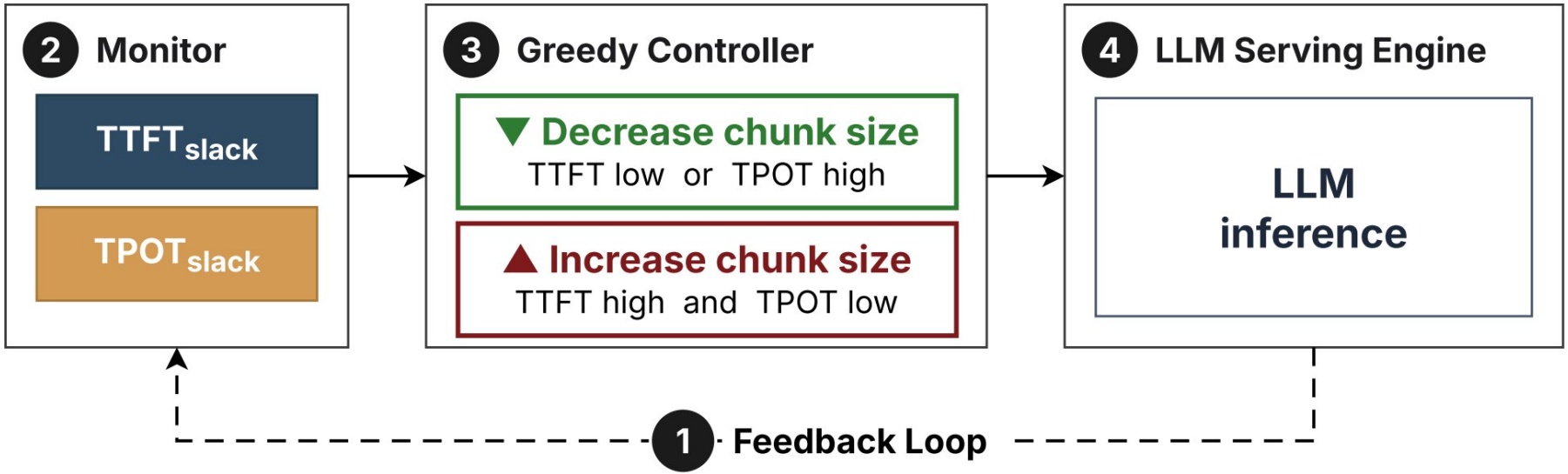
Deriving upper and lower bounds based on SLOs

→Prioritize the lower-bound for bubble minimization



Greedy Dynamic Chunked Prefill

Dynamically adjusts chunk size via real-time TTFT/TPOT feedback



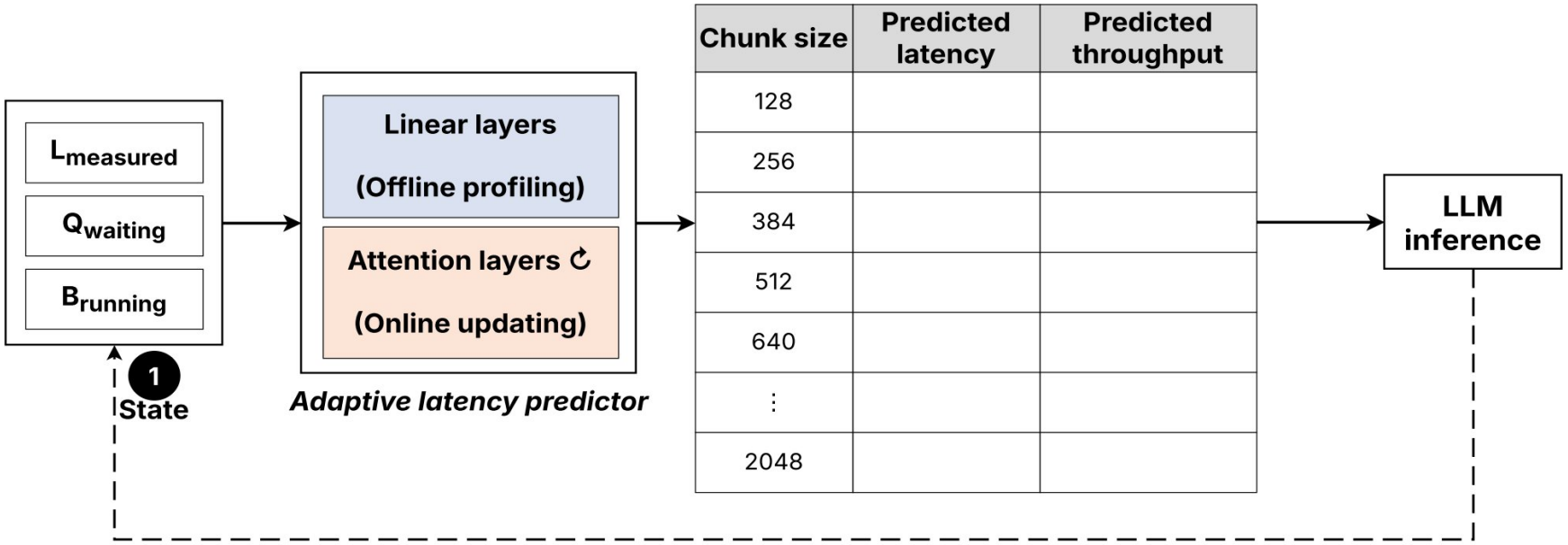
Why Reactive Feedback is Insufficient

- *Reactive Step-wise Lag*
- *Inaccurate Feedback*

Predictive Dynamic Chunked Prefill

Retrieve real-time system states

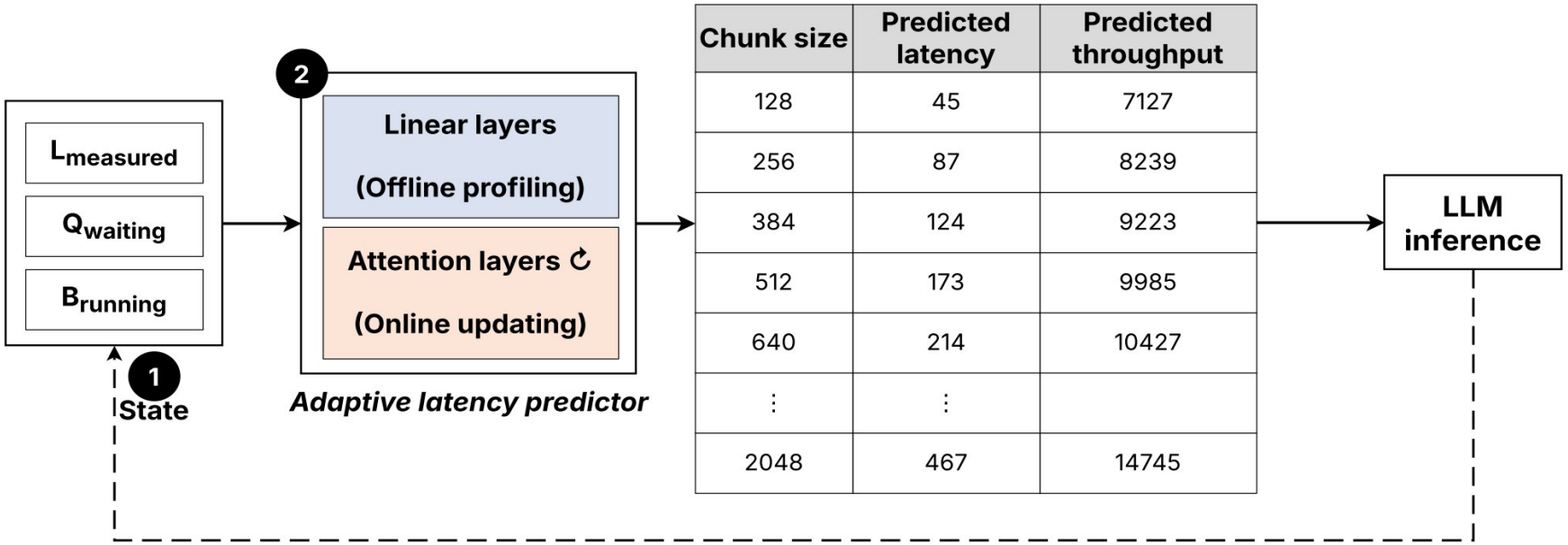
- $L_{measured}$ (Measured latency), $Q_{waiting}$ (Waiting queue), $B_{running}$ (Running batch)



Predictive Dynamic Chunked Prefill

Predict latency and throughput

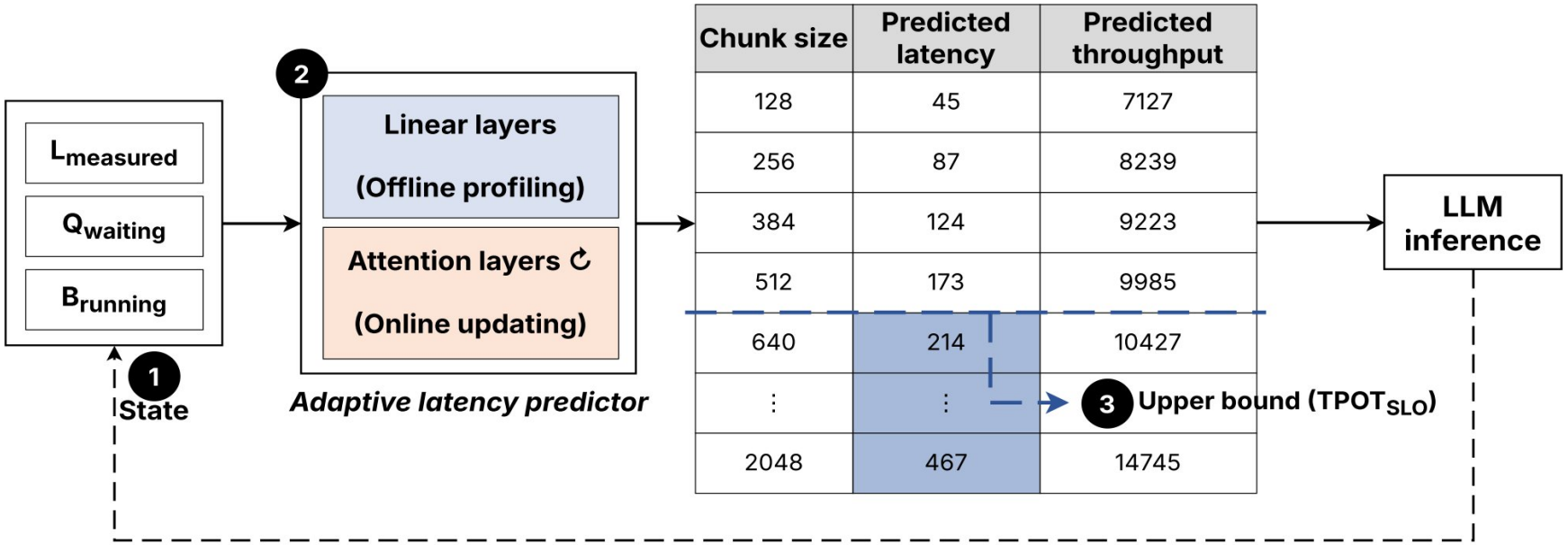
- Linear (Offline): few chunk size options
- Attention (Online): diverse batch combinations



Chunk size	Predicted latency	Predicted throughput
128	45	7127
256	87	8239
384	124	9223
512	173	9985
640	214	10427
⋮	⋮	
2048	467	14745

Predictive Dynamic Chunked Prefill

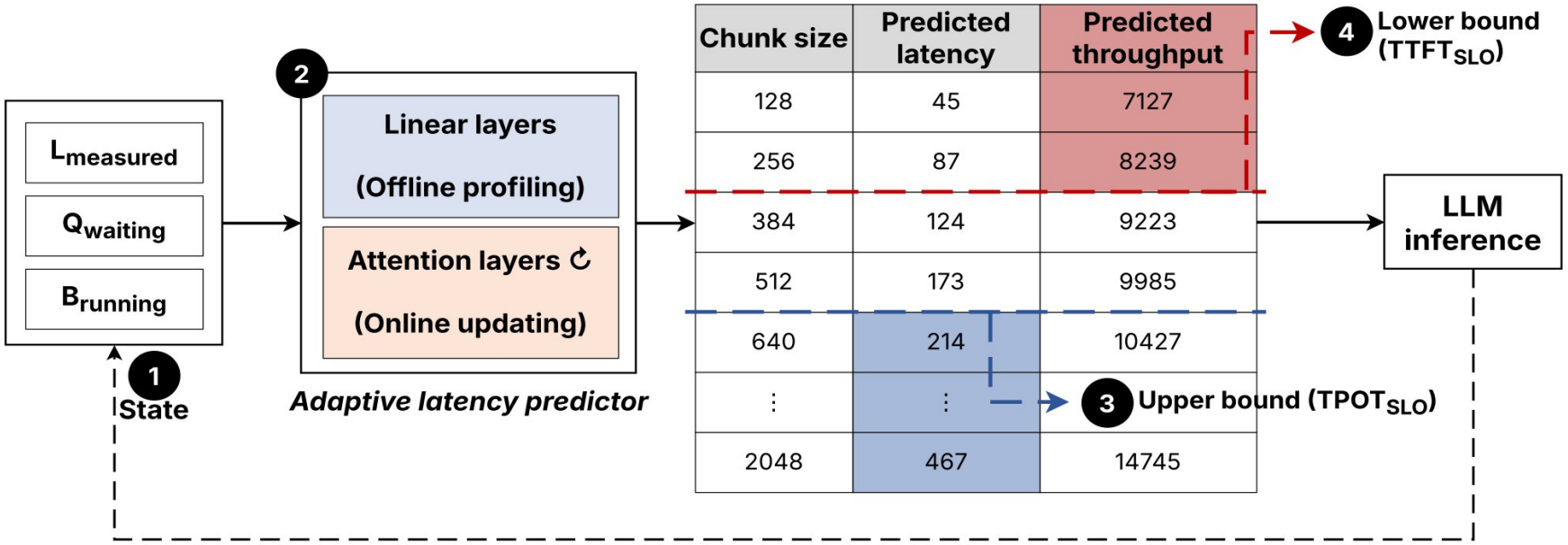
Upper Bound: Limited by $TPOT_{SLO}$ on predicted latency



Predictive Dynamic Chunked Prefill

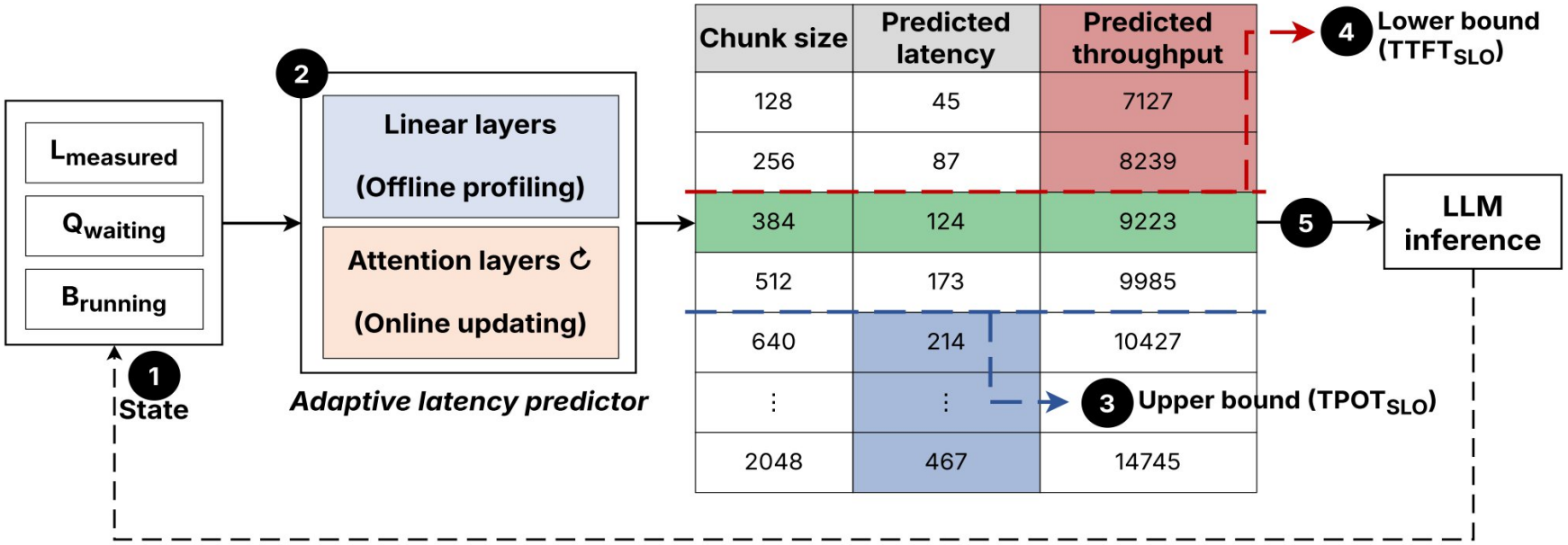
Upper Bound: Limited by $TPOT_{SLO}$ on predicted latency

Lower Bound: Filtered by $TTFT_{SLO}$ on predicted throughput

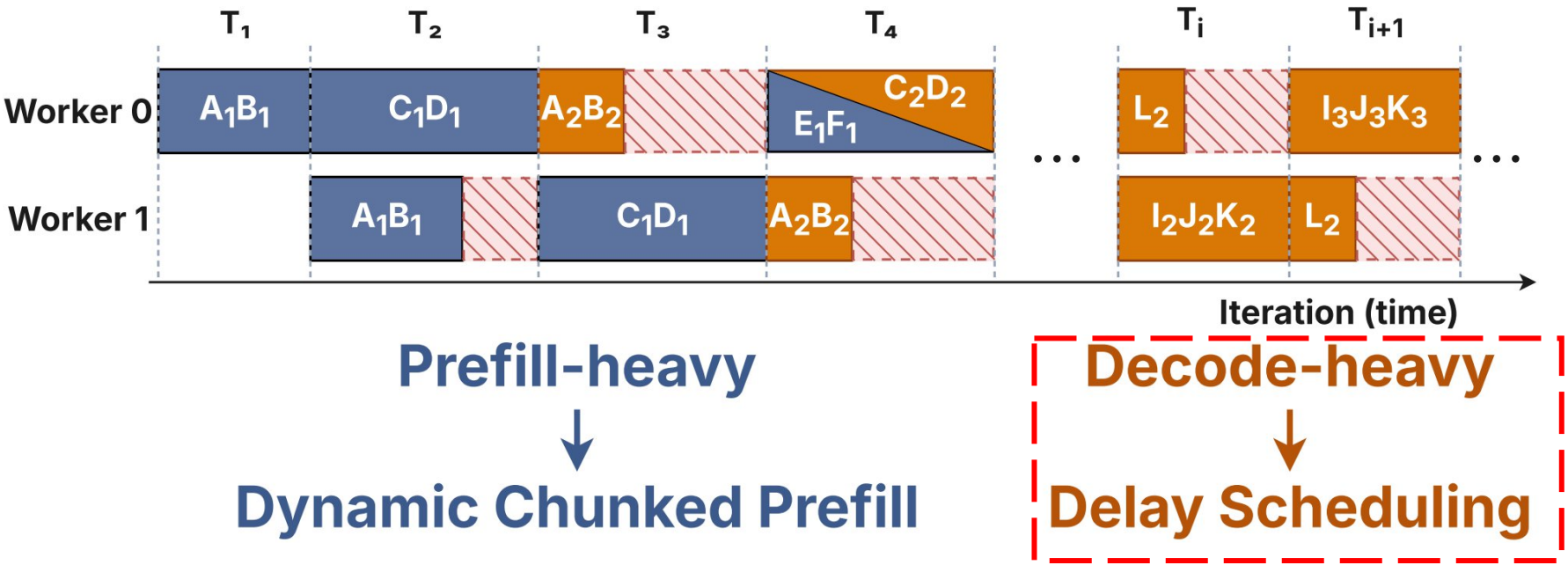


Predictive Dynamic Chunked Prefill

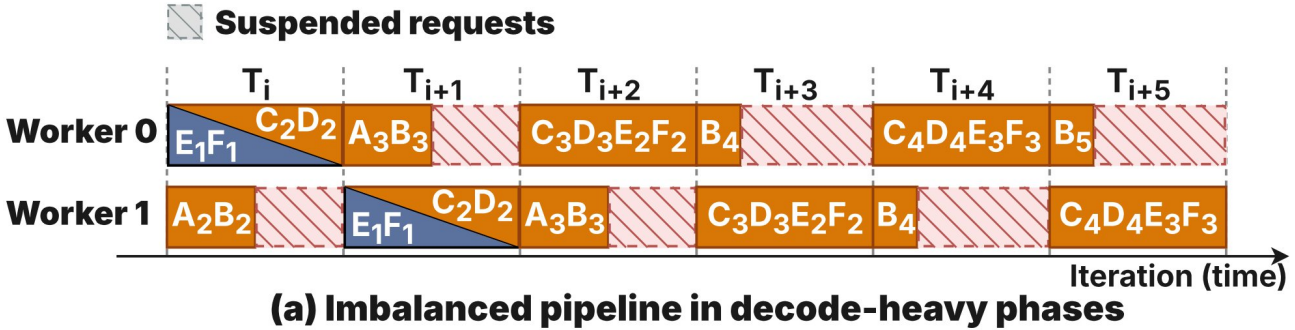
Prioritizing the Lower Bound to minimize pipeline bubbles



Our Approach: Chunking & Rebalancing



Delay Scheduling

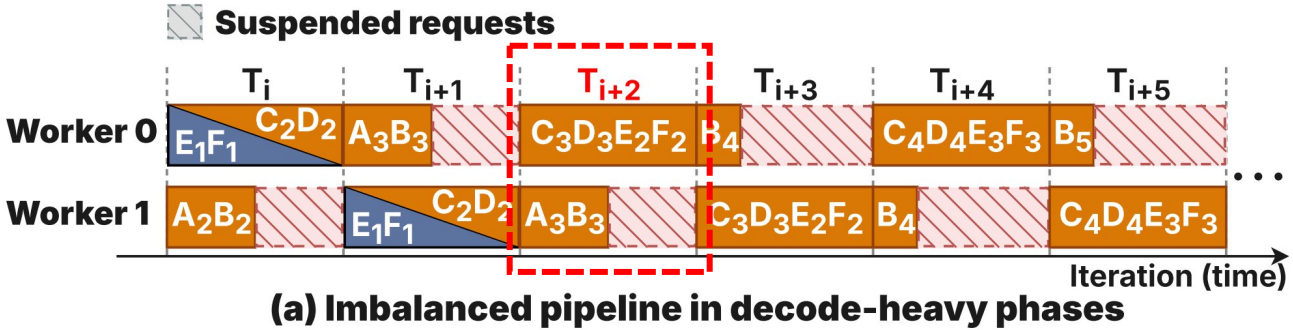


(a) Imbalanced pipeline in decode-heavy phases



(b) Balanced pipeline with delay scheduling

Delay Scheduling



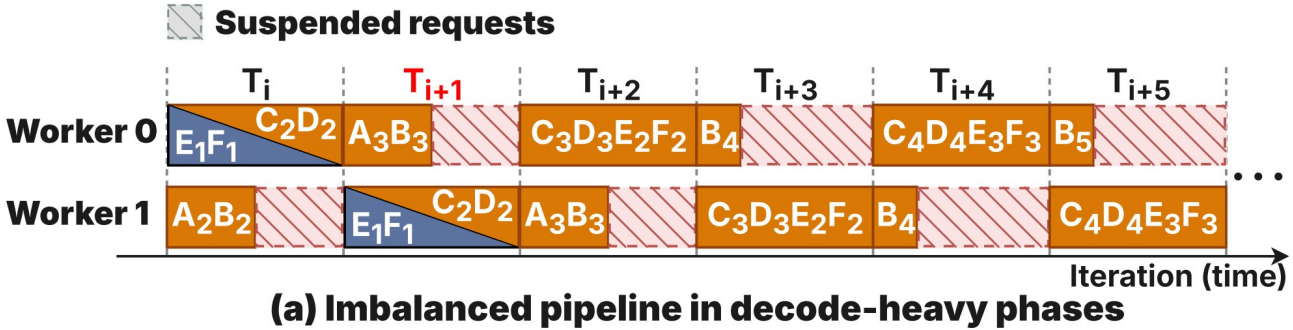
(a) Imbalanced pipeline in decode-heavy phases



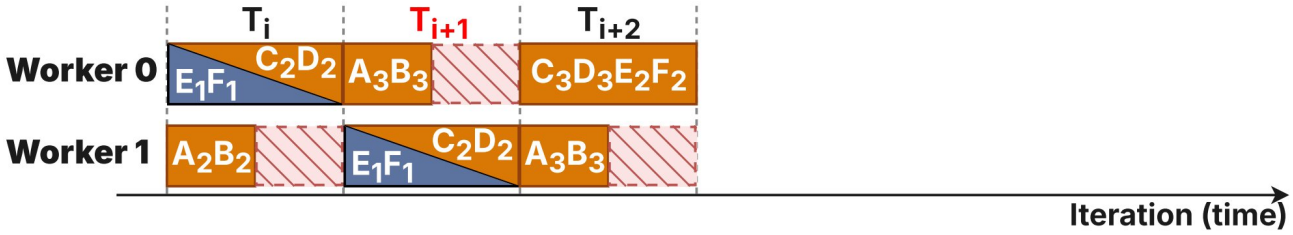
(b) Balanced pipeline with delay scheduling

Delay Scheduling

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



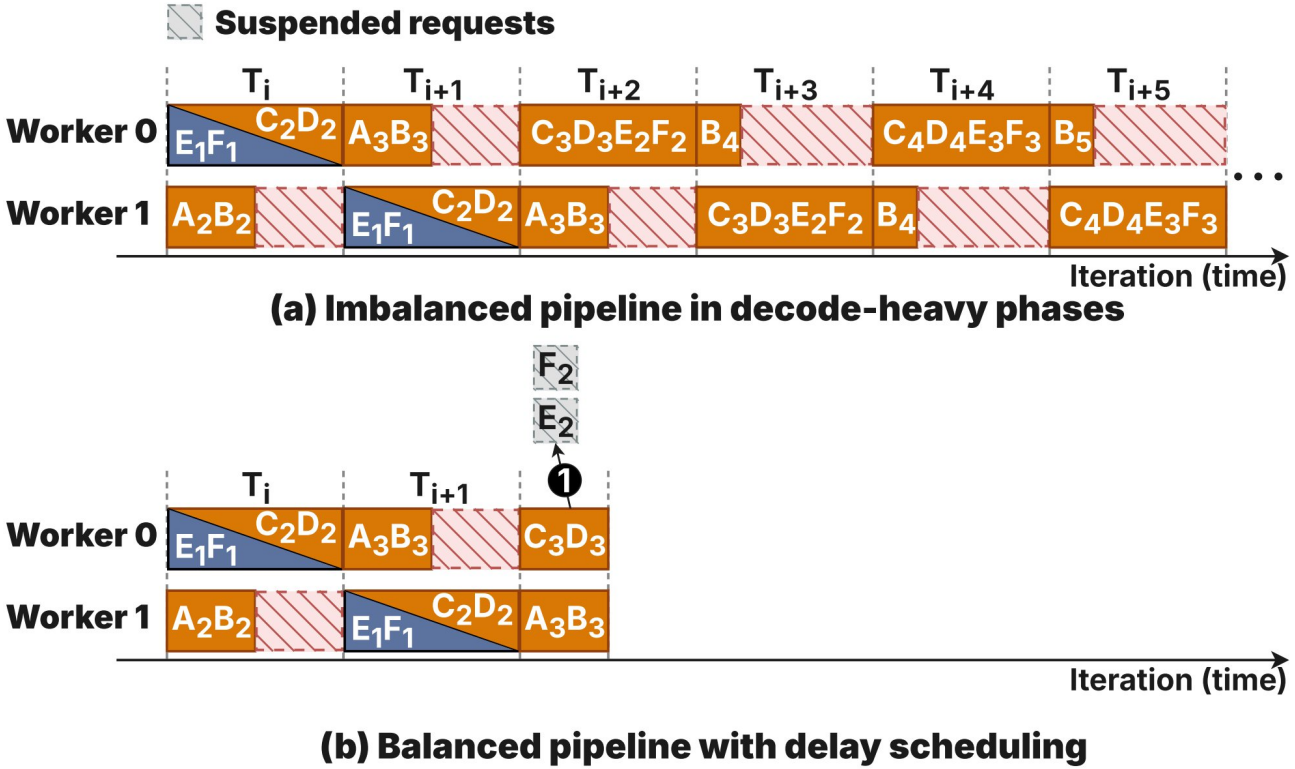
(a) Imbalanced pipeline in decode-heavy phases



(b) Balanced pipeline with delay scheduling

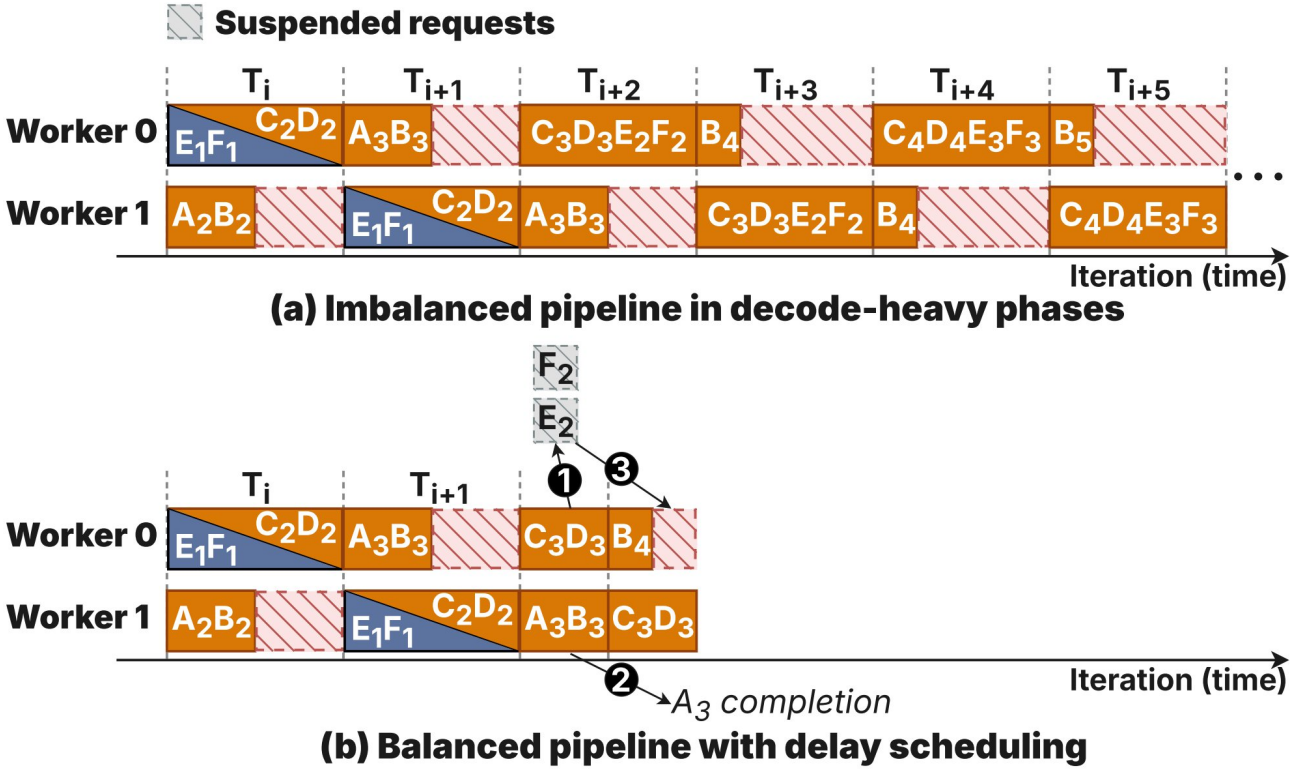
Delay Scheduling

Tiling boundary spike suspends Requests E and F



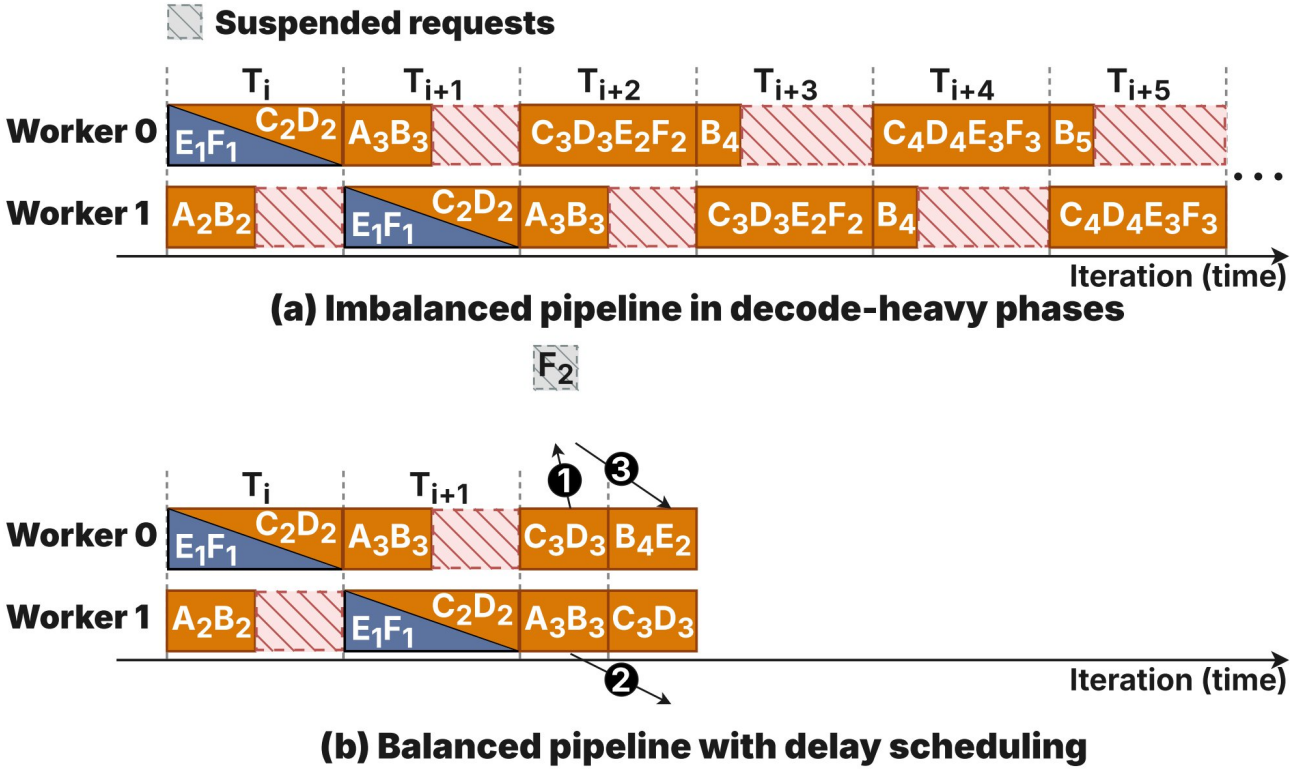
Delay Scheduling

Resumes suspended Request E as Request A completes



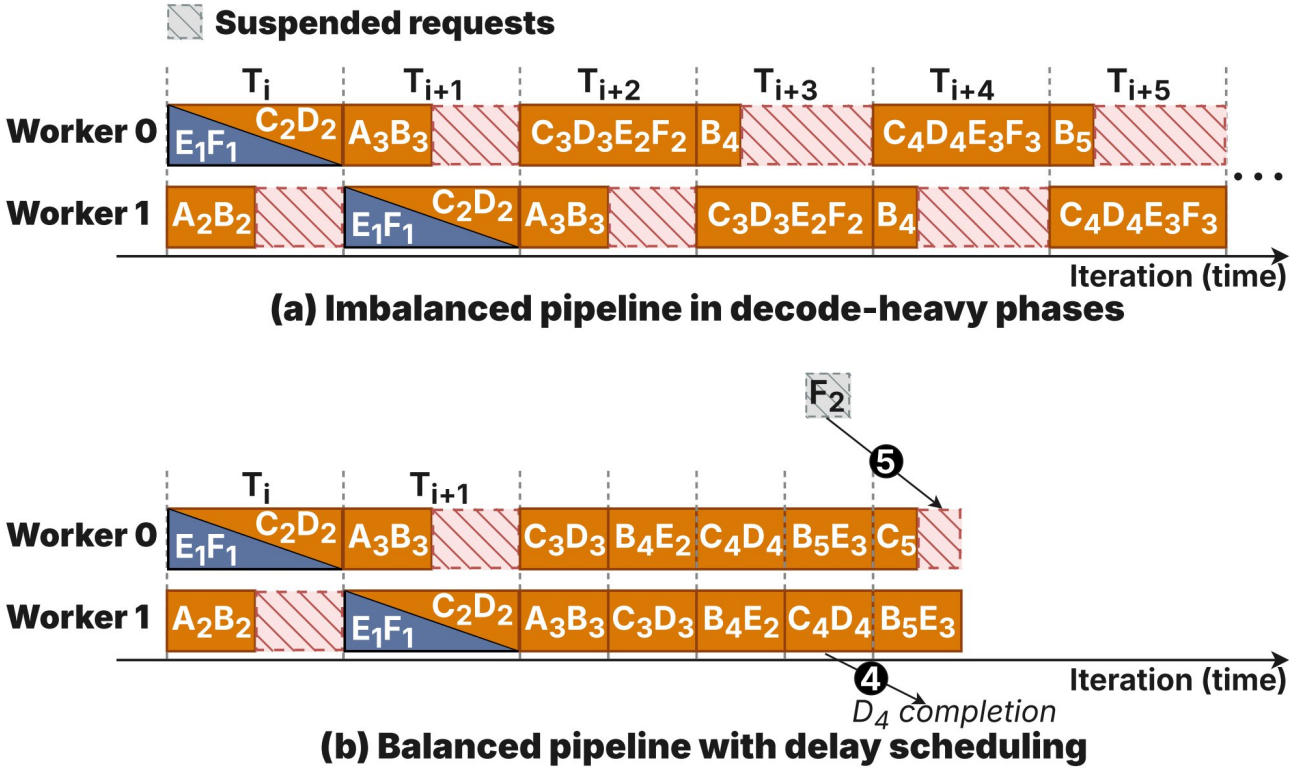
Delay Scheduling

Resumes suspended Request E as Request A completes



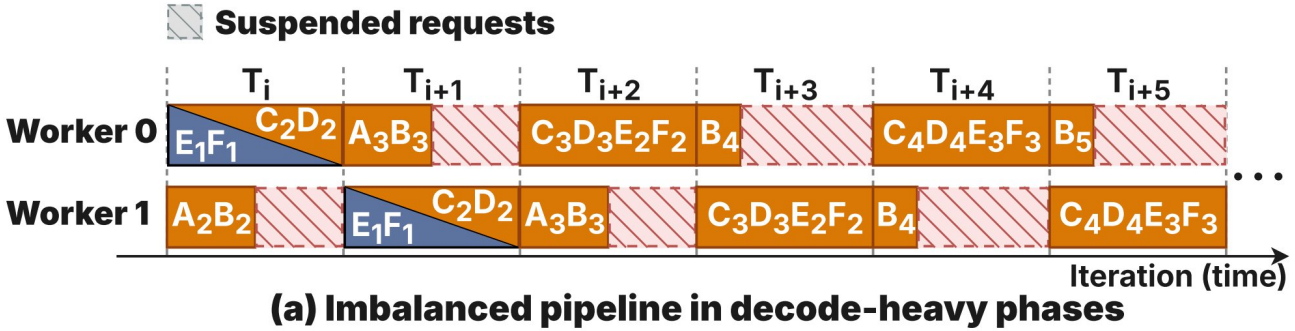
Delay Scheduling

Sequentially resumes Request F as request D completes

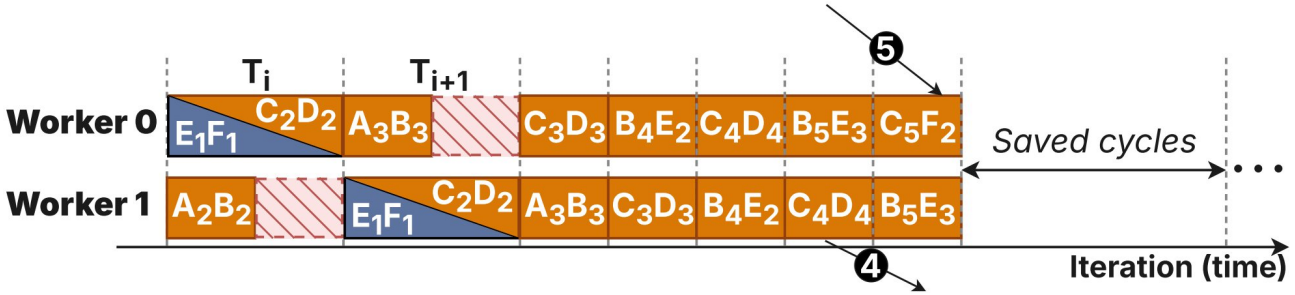


Delay Scheduling

Sequentially resumes Request F as request D completes



(a) Imbalanced pipeline in decode-heavy phases



(b) Balanced pipeline with delay scheduling

Evaluation Setup

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

Workloads

- **Models:** Qwen2.5 32B and 14B
- **Datasets:**

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

Metrics

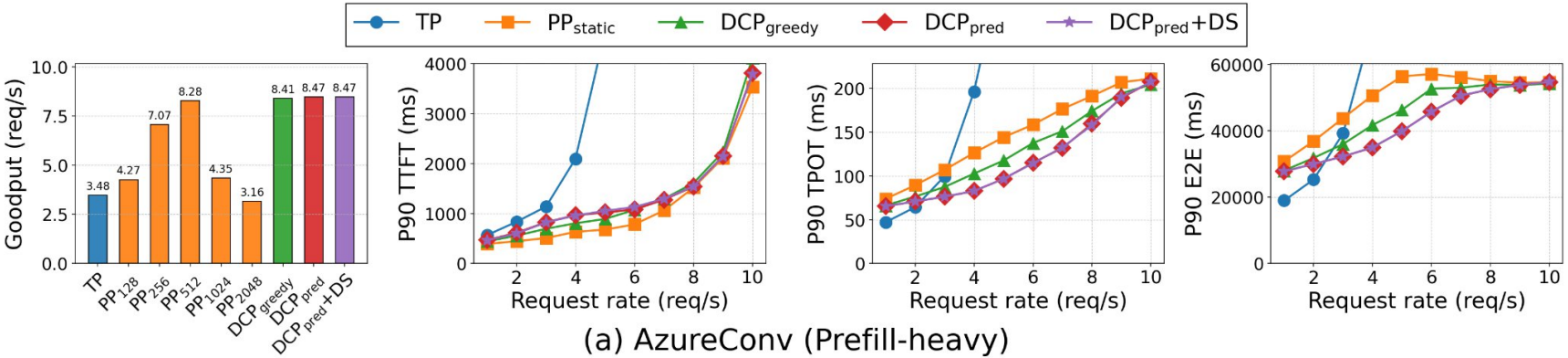
- Goodput, TTFT (SLO: 2,000ms), TPOT (SLO: 200ms), E2E

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

Evaluation: Goodput & Latency

Prefill-heavy: DCP dramatically reduces TPOT and E2E latency

→Eliminates prefill-induced bubbles by dynamically chunking inputs

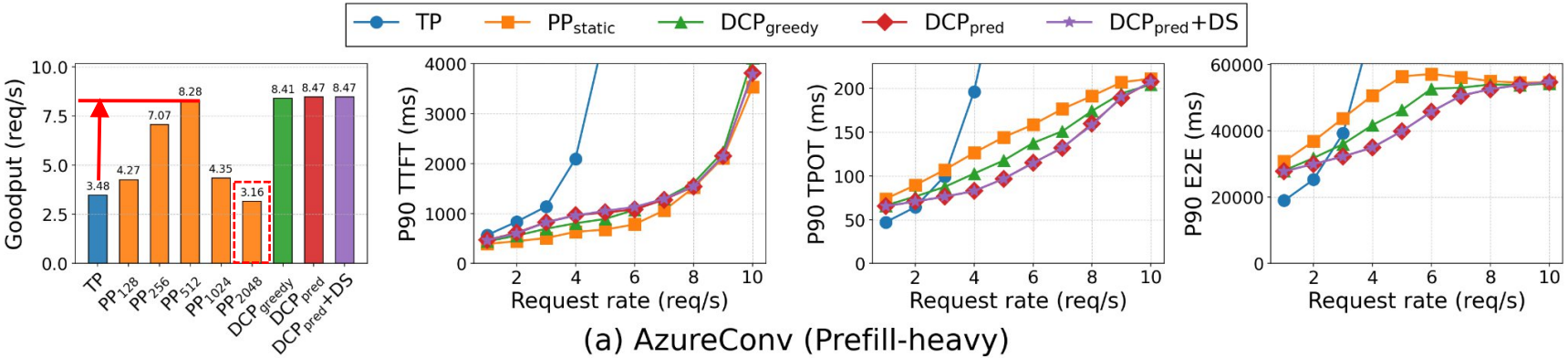


(a) AzureConv (Prefill-heavy)

Evaluation: Goodput & Latency

Prefill-heavy: DCP dramatically reduces TPOT and E2E latency

→Eliminates prefill-induced bubbles by dynamically chunking inputs

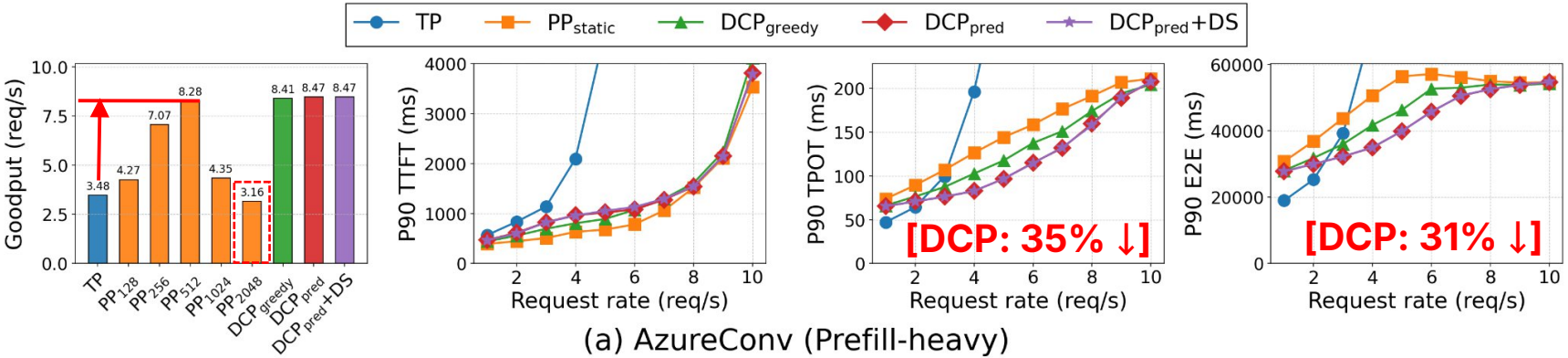


(a) AzureConv (Prefill-heavy)

Evaluation: Goodput & Latency

Prefill-heavy: DCP dramatically reduces TPOT and E2E latency

→Eliminates prefill-induced bubbles by dynamically chunking inputs

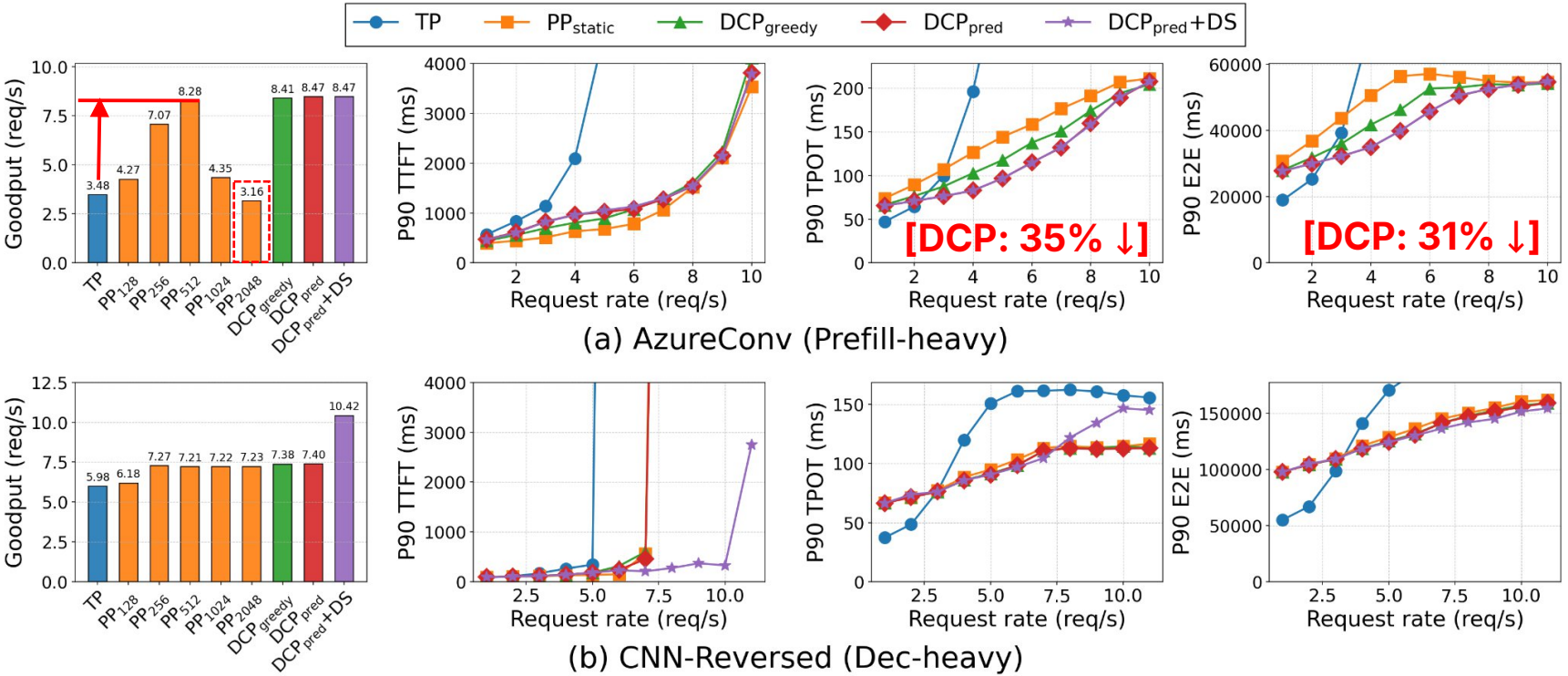


(a) AzureConv (Prefill-heavy)

Evaluation: Goodput & Latency

Dec-heavy: DS significantly improves goodput

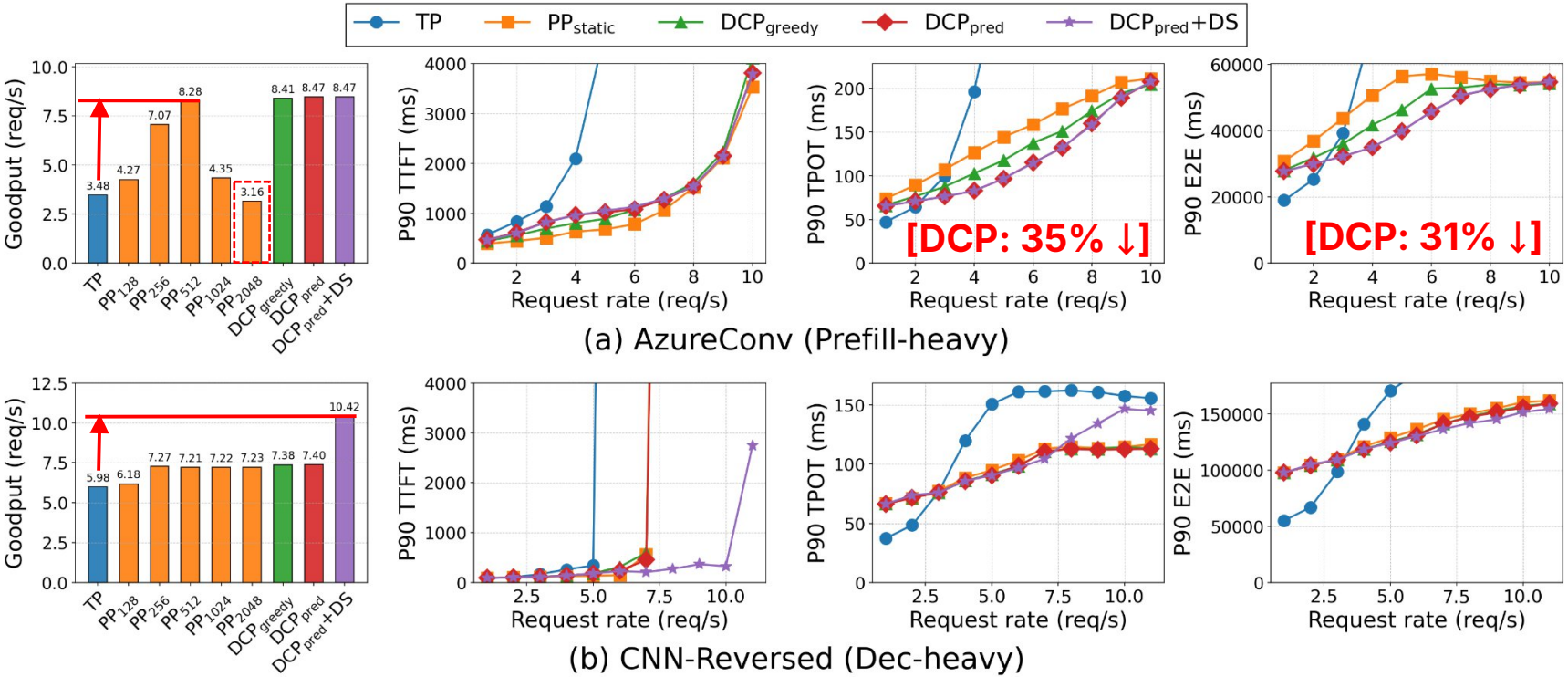
→Skips KV allocation for delayed requests & reduces bubbles for faster E2E



Evaluation: Goodput & Latency

Dec-heavy: DS significantly improves goodput

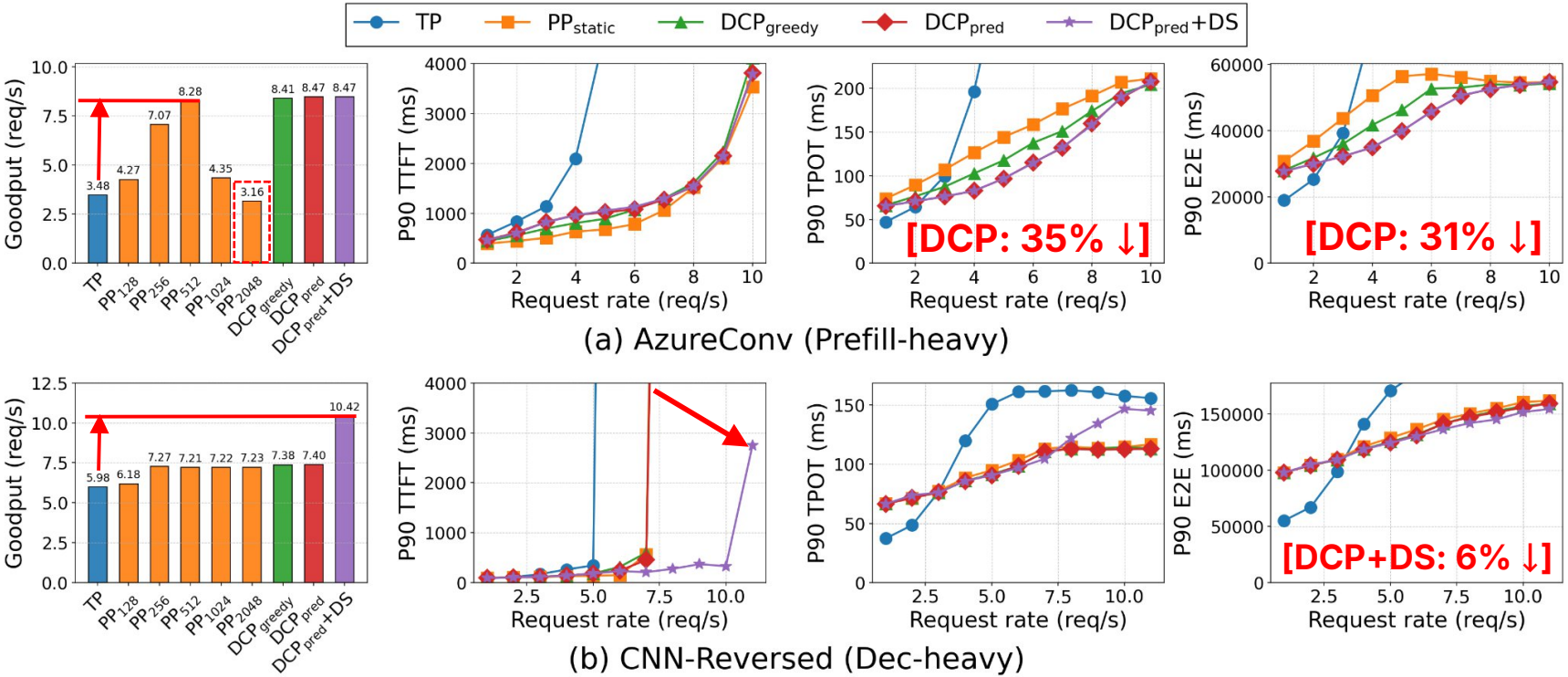
→Skips KV allocation for delayed requests & reduces bubbles for faster E2E



Evaluation: Goodput & Latency

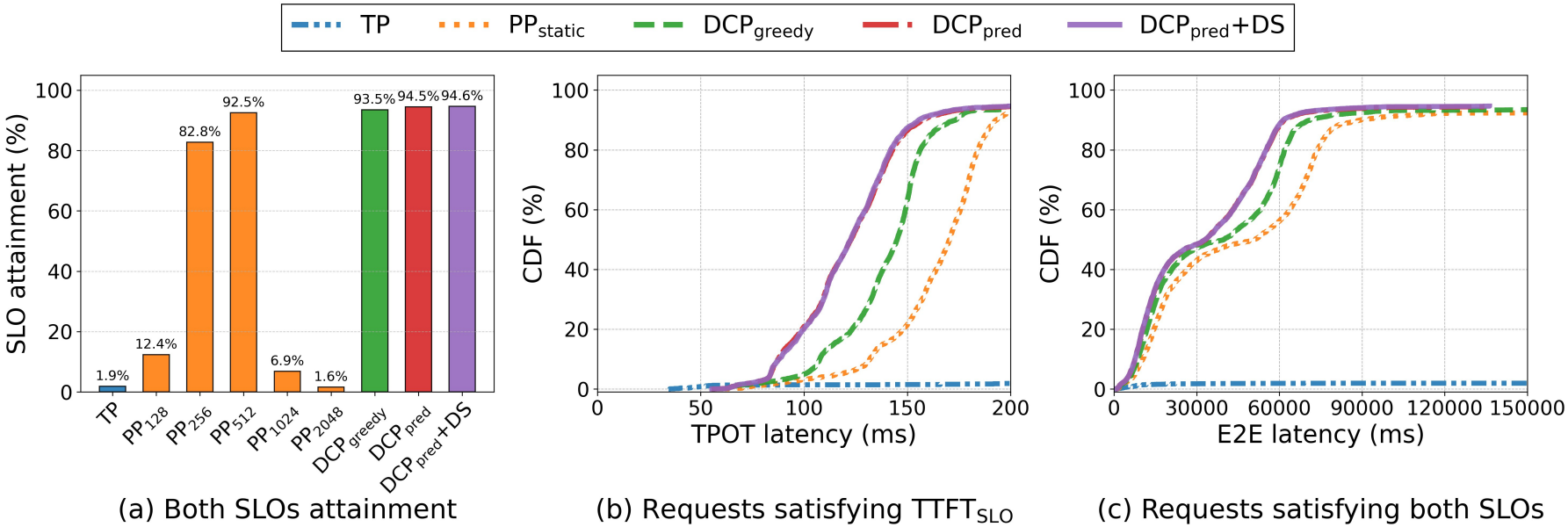
Dec-heavy: DS significantly improves goodput

→Skips KV allocation for delayed requests & reduces bubbles for faster E2E



Evaluation: Real Workload Replay

Significantly Improves request completion within the same latency limit



(a) Both SLOs attainment

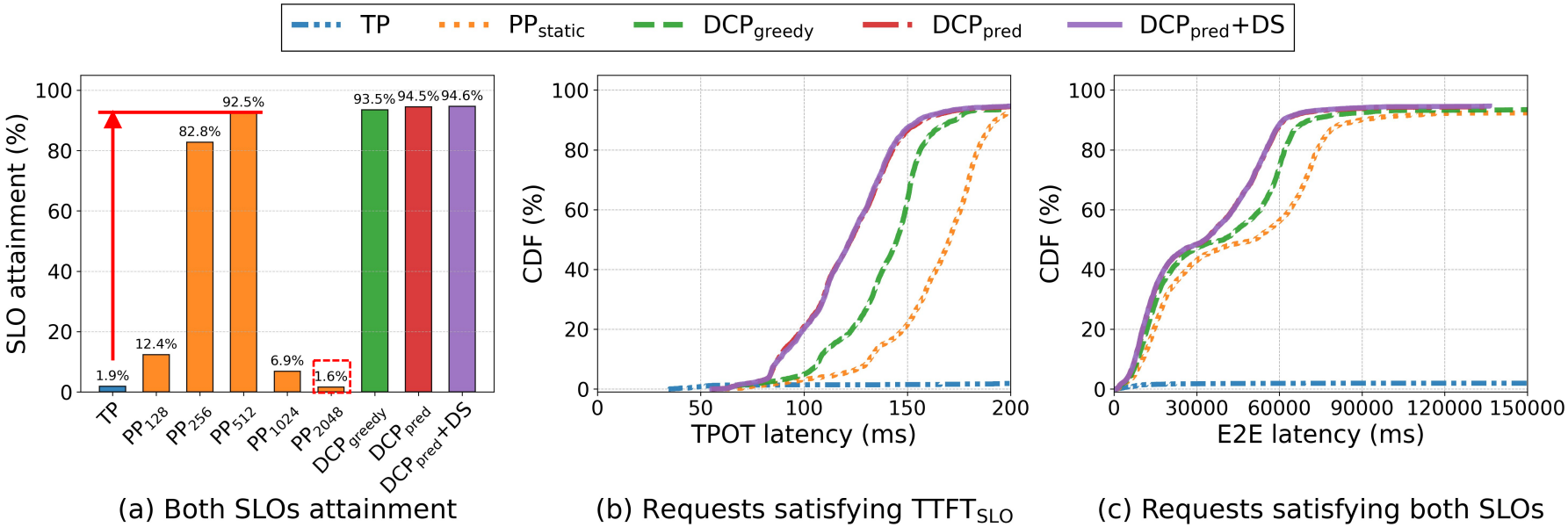
(b) Requests satisfying $TTFT_{SLO}$

(c) Requests satisfying both SLOs

AzureConv (real trace replay)

Evaluation: Real Workload Replay

Significantly Improves request completion within the same latency limit



(a) Both SLOs attainment

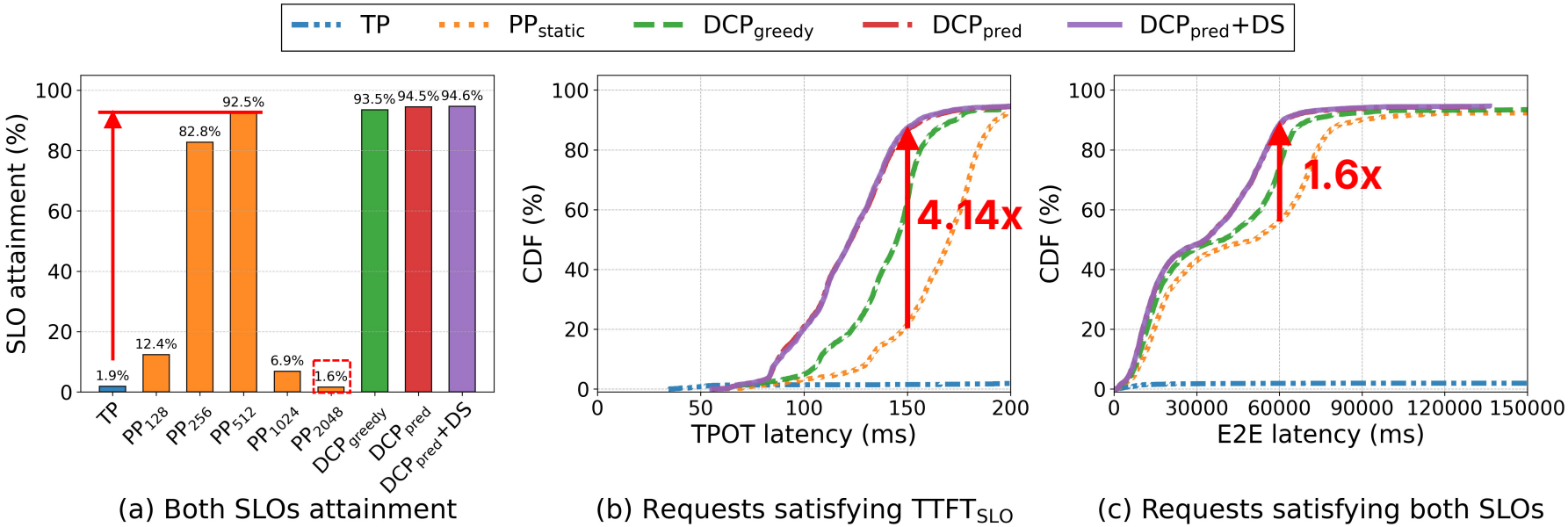
(b) Requests satisfying $TTFT_{SLO}$

(c) Requests satisfying both SLOs

AzureConv (real trace replay)

Evaluation: Real Workload Replay

Significantly Improves request completion within the same latency limit



(a) Both SLOs attainment

(b) Requests satisfying $TTFT_{SLO}$

(c) Requests satisfying both SLOs

AzureConv (real trace replay)

Summary

Problem

- Severe pipeline imbalances under online workloads

Solution

- Dynamic chunked prefill uses SLOs to eliminate prefill-induced bubbles
- Delay scheduling rebalances decode workloads across workers

Result

- Maximizes goodput over TP while reducing TPOT by 35% and E2E latency by 31% vs. static PP
- Practical alternative to TP for PCIe-interconnected systems (commodity GPU servers, NPUs)

📧 sjhwang@csl.korea.ac.kr | 💻 github.com/Sys-KU/FastPP