LLM Inference Serving Systems

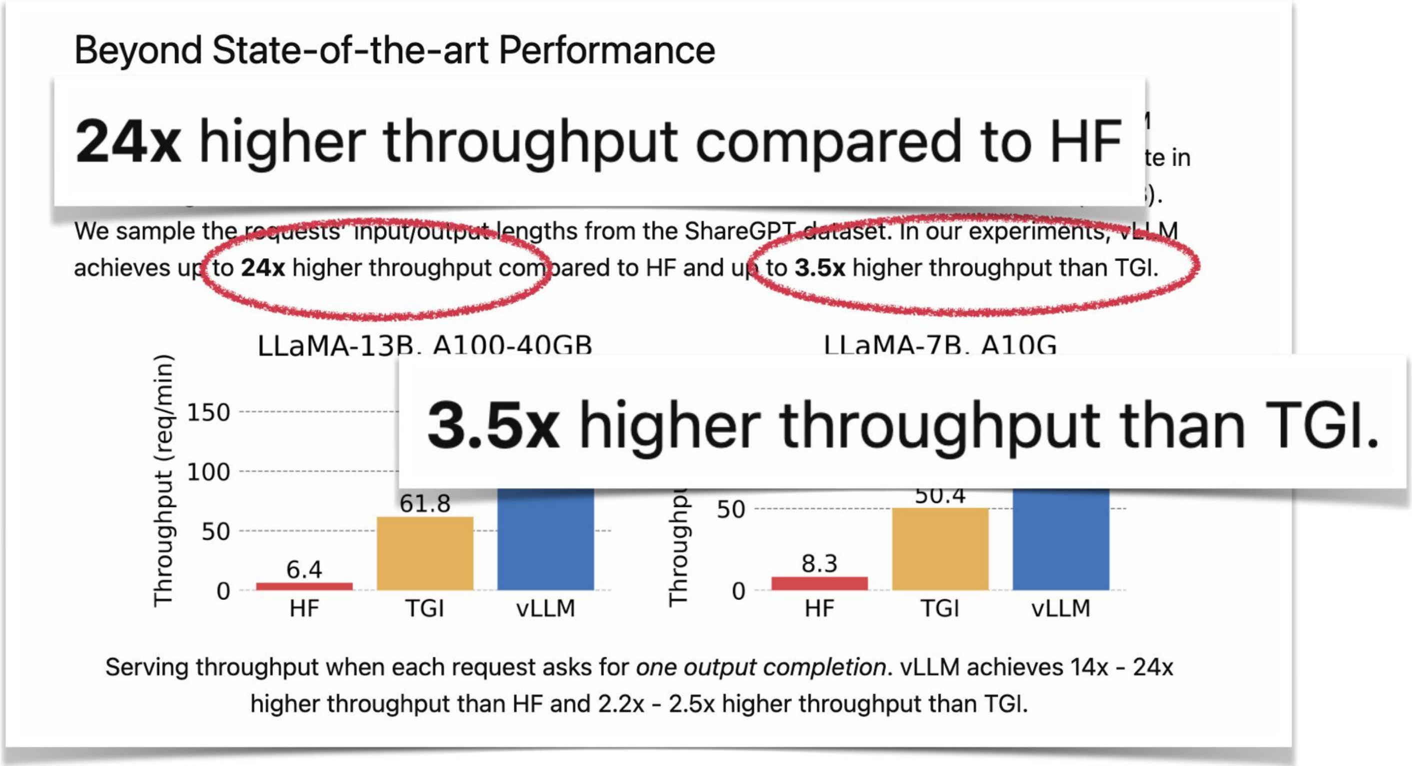
Arvind Krishnamurthy

Material adapted from slides by Hao Zhang (UCSD) and Amey Agrawal (GTech)

Lecture Outline

- Requirements of LLM Serving Systems
- Interleaved execution in Sarathi Serve
- Disaggregation in DistServe

LLM Systems Today Optimize **Throughput**



Motivation: Applications have Diverse SLO

• TTFT

Time to first token

Initial response time

• TPOT

Time per output token

Average time between two subsequent generated tokens

Chatbot

Fast initial response

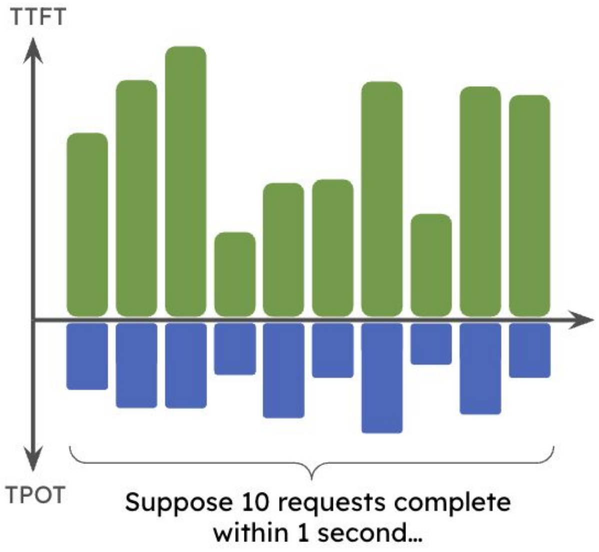
Human reading speed (P99 latency = 250ms)

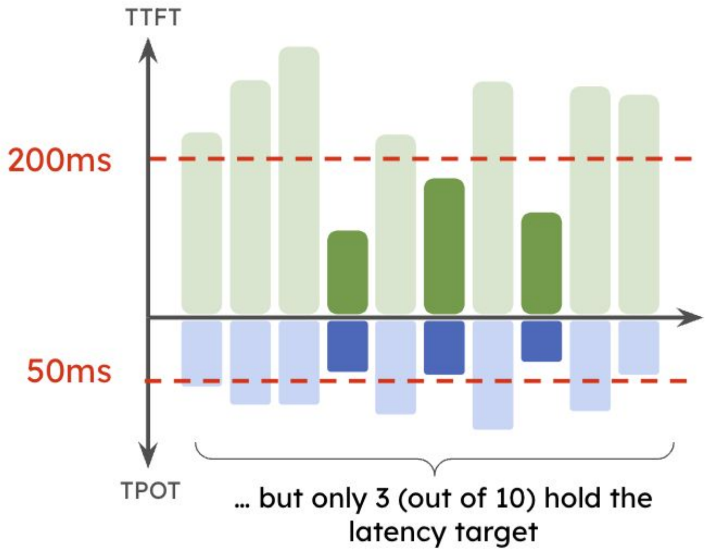
Summarization

User can tolerate longer initial response

Data output generation (P99 latency = 35ms)

High Throughput ≠ High Goodput





Throughput = 10 rps
= completed request / time

under SLO criteria

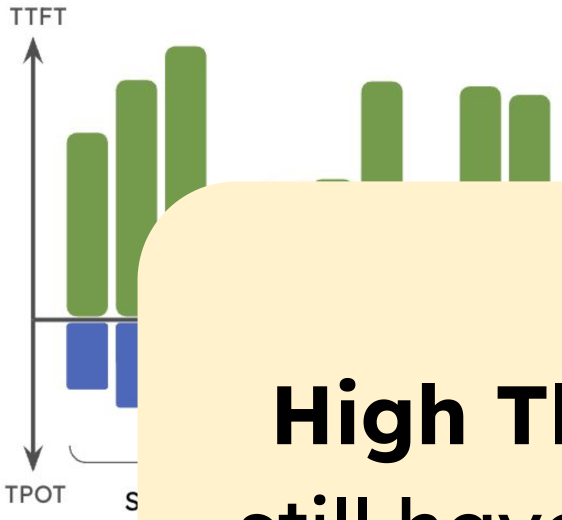
Goodput = 3 rps 🤯
= completed request **within SLO** / time

High Throughput System

...

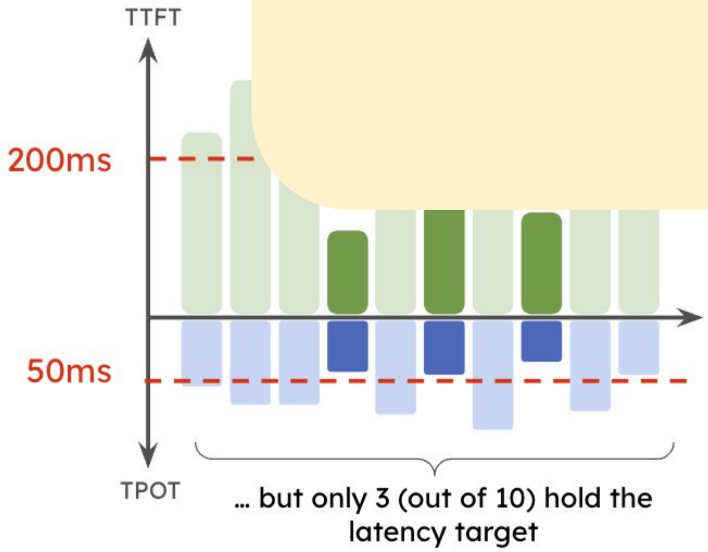
can have
Low Goodput!

High Throughput ≠ High Goodput



High
put

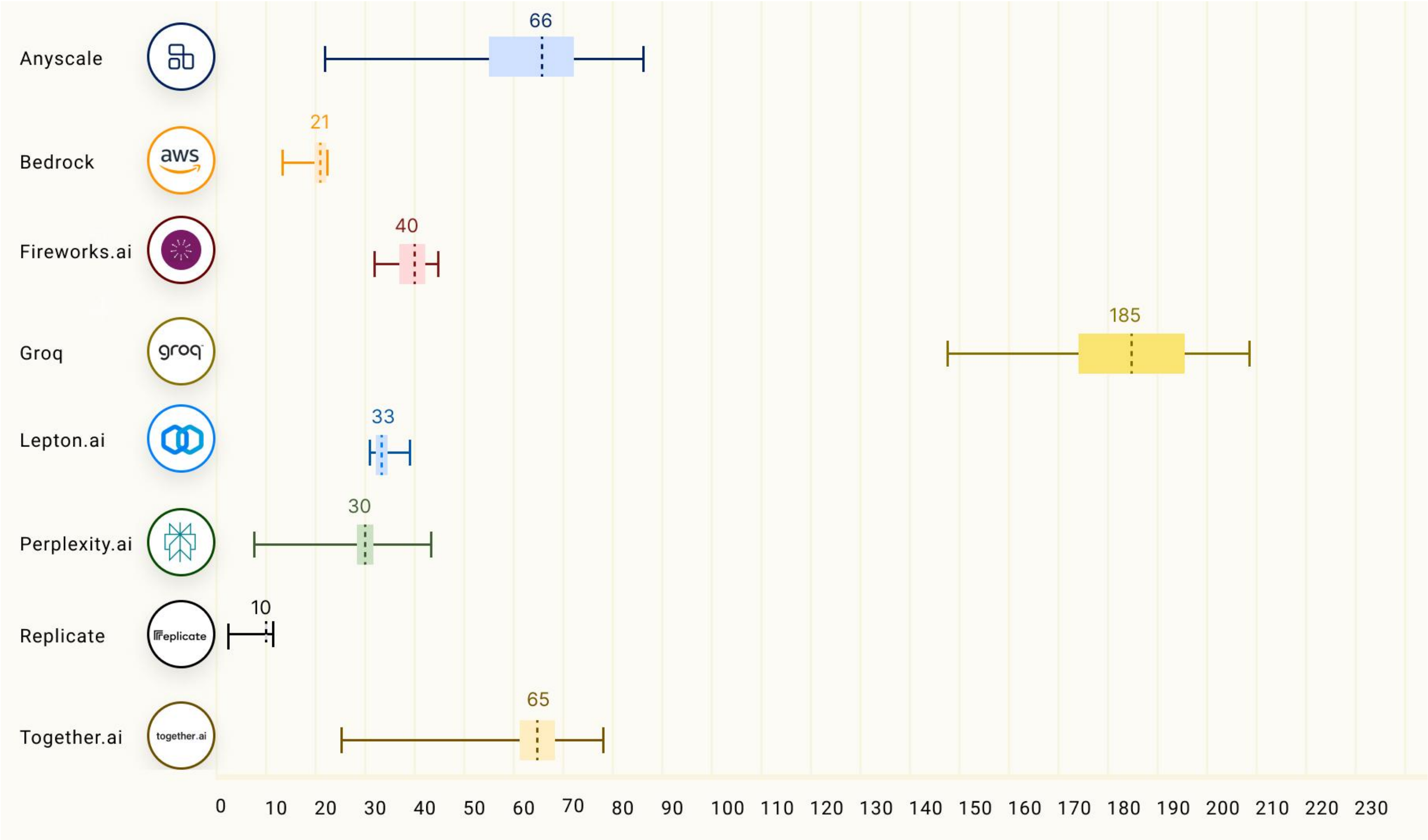
High Throughput can still have **Low Goodput** ⇒ Poor UX 💔



dput!

Goodput = 3 rps
= completed request **within SLO** / time

LLM Performance leaderboard (tokens/sec)



Taming Throughput-Latency Tradeoff in LLM Inference with **Sarathi-Serve**

Amey Agrawal[1], Nitin Kedia[2], Ashish Panwar[2], Jayashree Mohan[2], Nipun Kwatra[2], Bhargav Gulavani[2], Alexey Tumanov[1], Ramachandran Ramjee[2]

[1]Georgia Institute of Technology, [2]Microsoft Research India

Can we maintain low latency with high throughput?

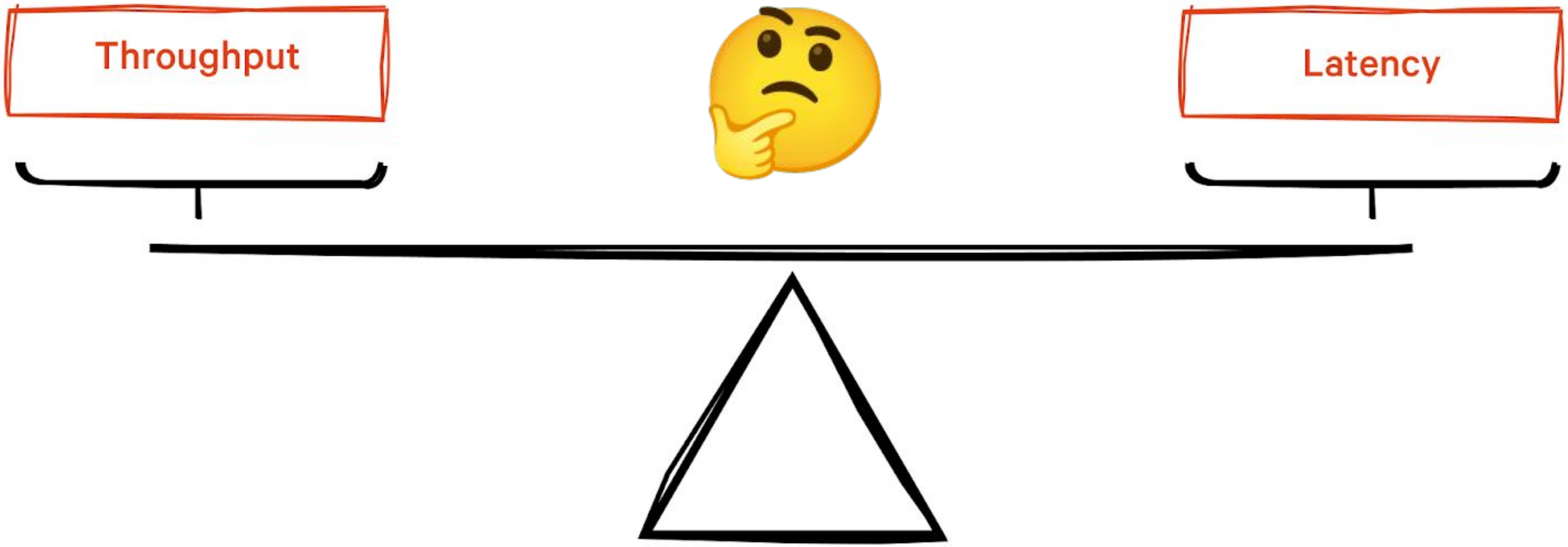
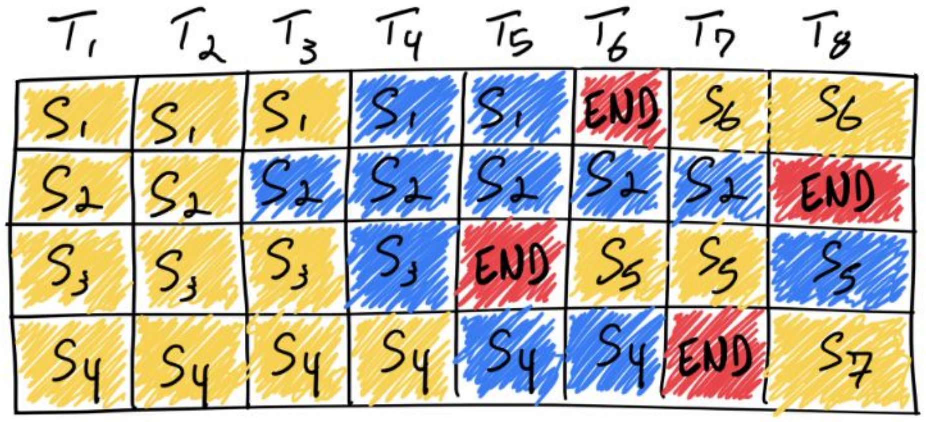


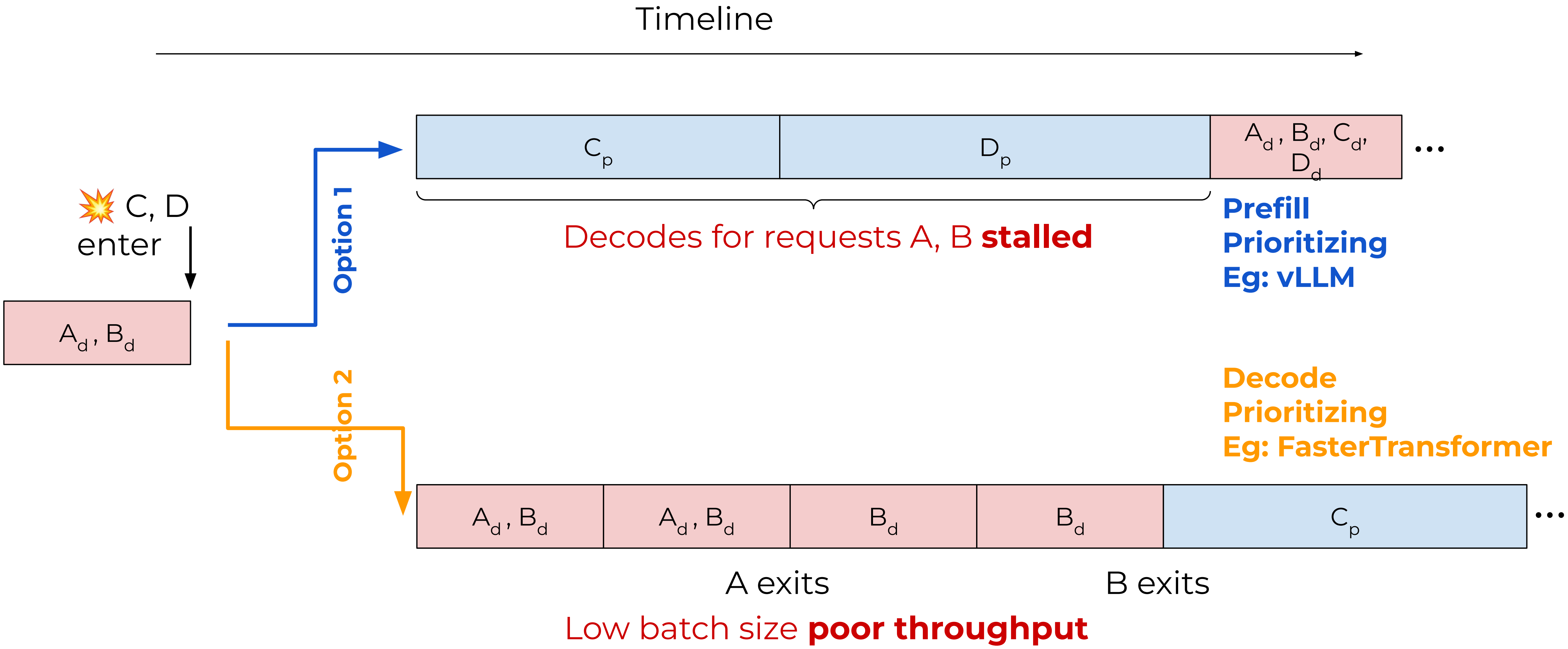
Image credits: Redpanda

Background: Continuous Batching

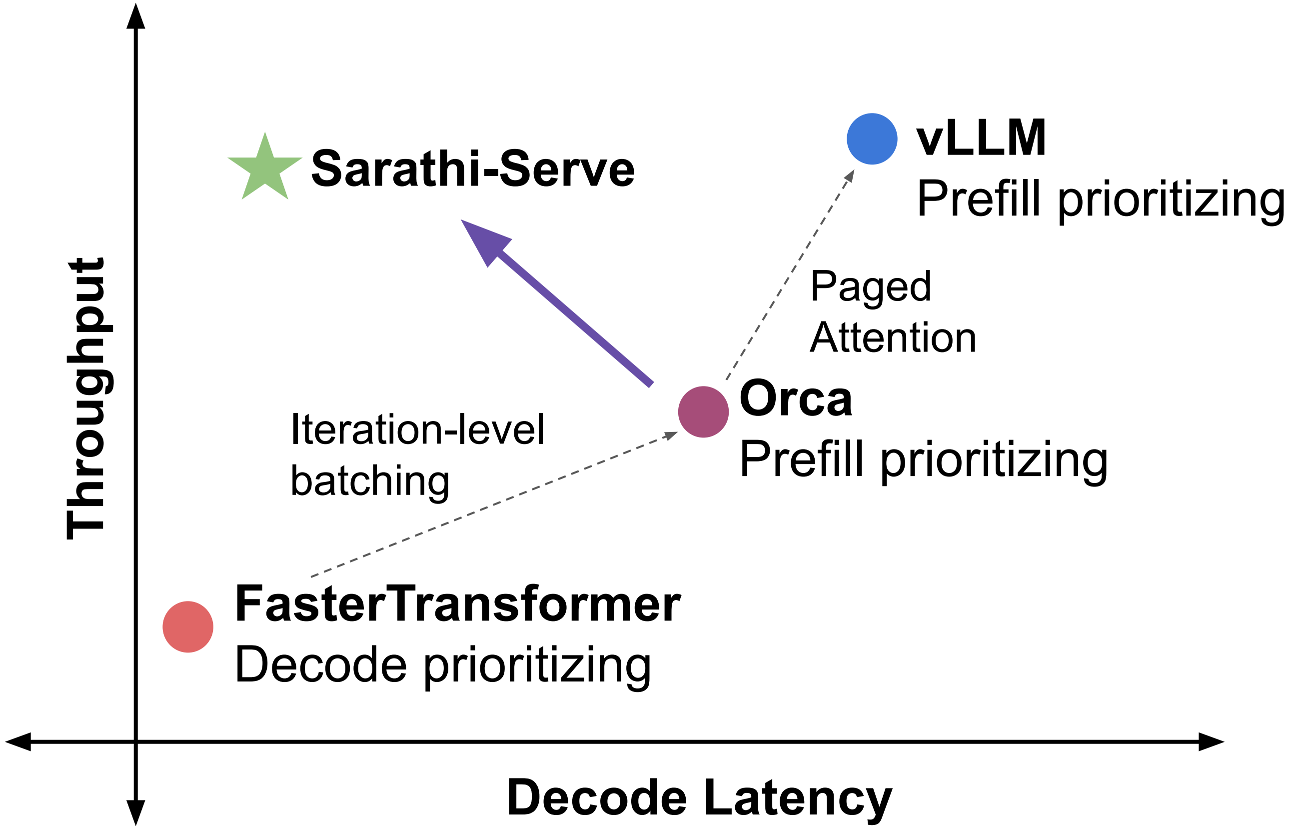
- Integrate new requests as old requests rotate off
- Need to perform prefill for new request
  - Results in a stall for existing requests
- Two queues in the system: prefill requests waiting to be integrated; ongoing decode requests



🔎 The Prefill-Decode Scheduling Conundrum



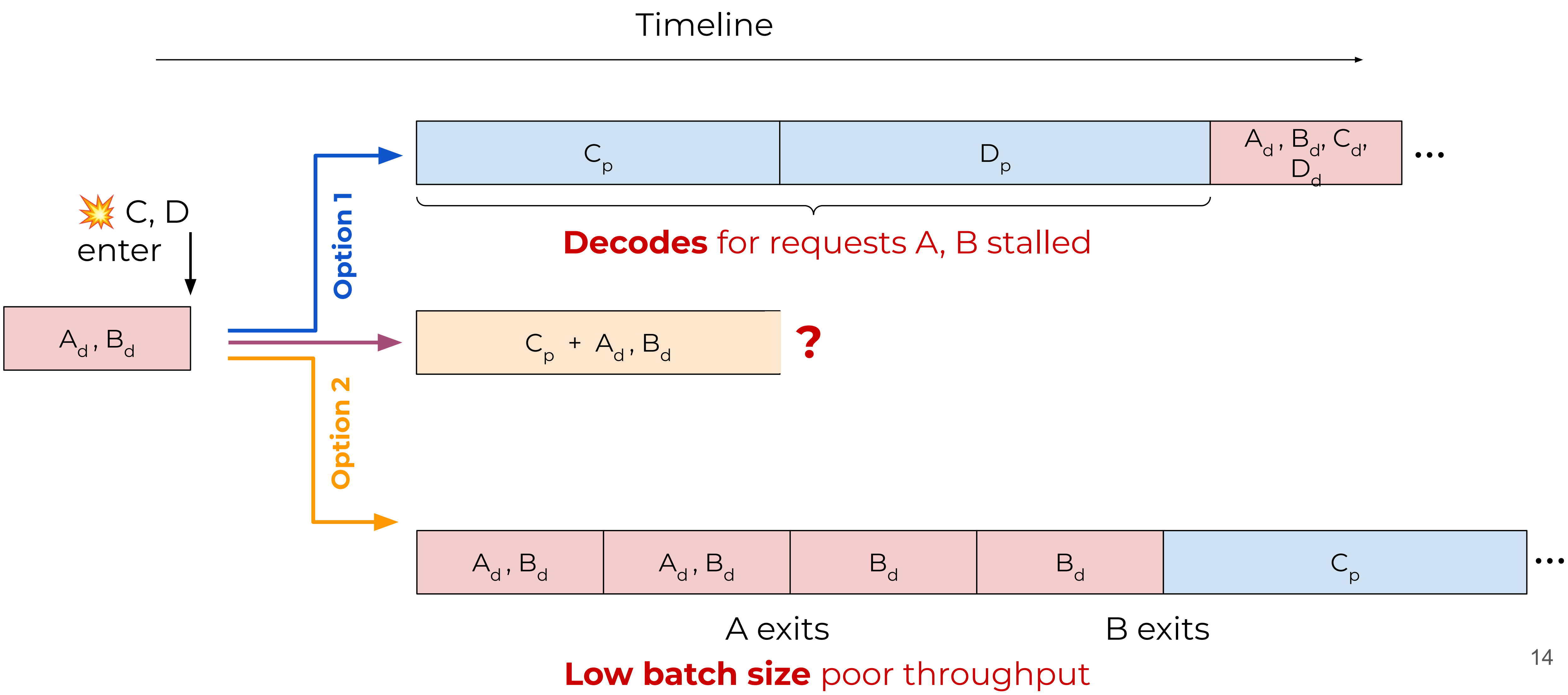
⚖️ The Latency-Throughput Tradeoff



Existing batching policies make a harsh latency-throughput tradeoff ❗

How can we achieve both high throughput and low-latency? 🤔

🔎 The Prefill-Decode Scheduling Conundrum



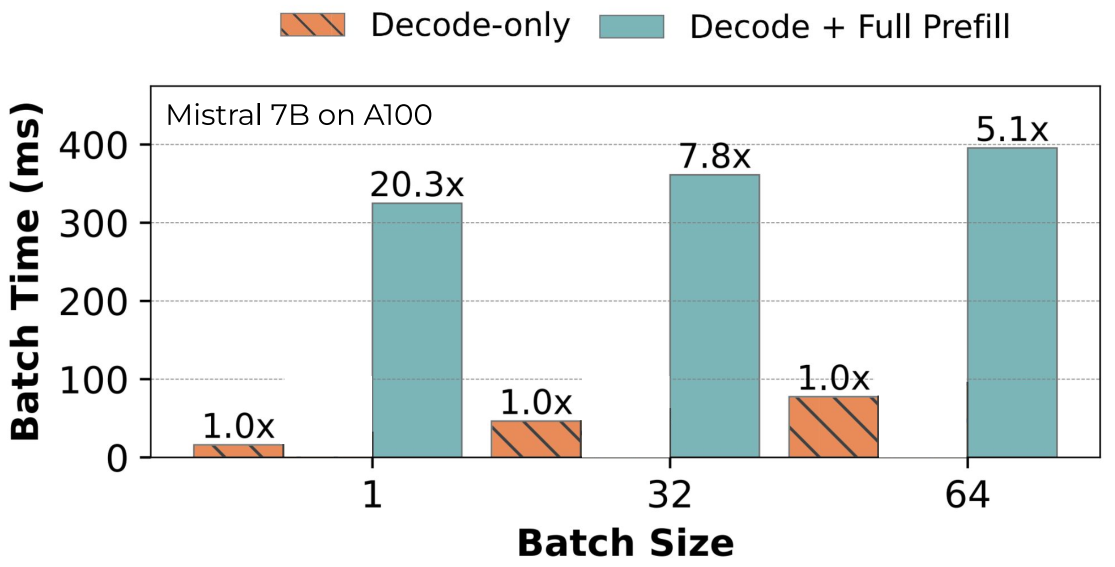
🔎 Mixed Batching

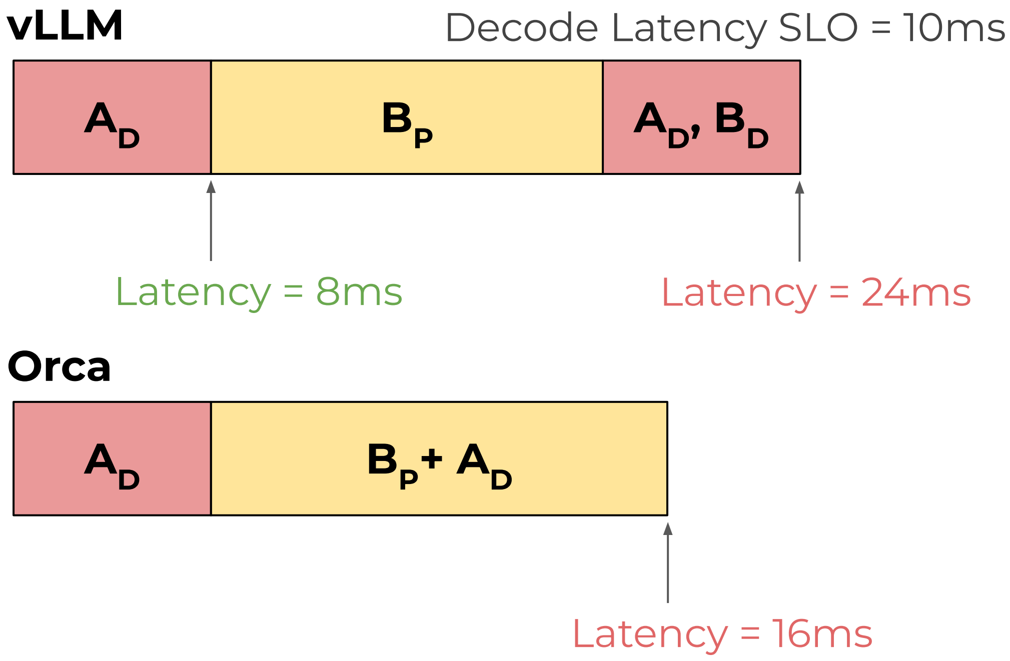
Idea

🧑‍🏭 Fused computation of prefill and decodes

Challenge

😭 Naively combining prefill and decode operations leads to increase in latency

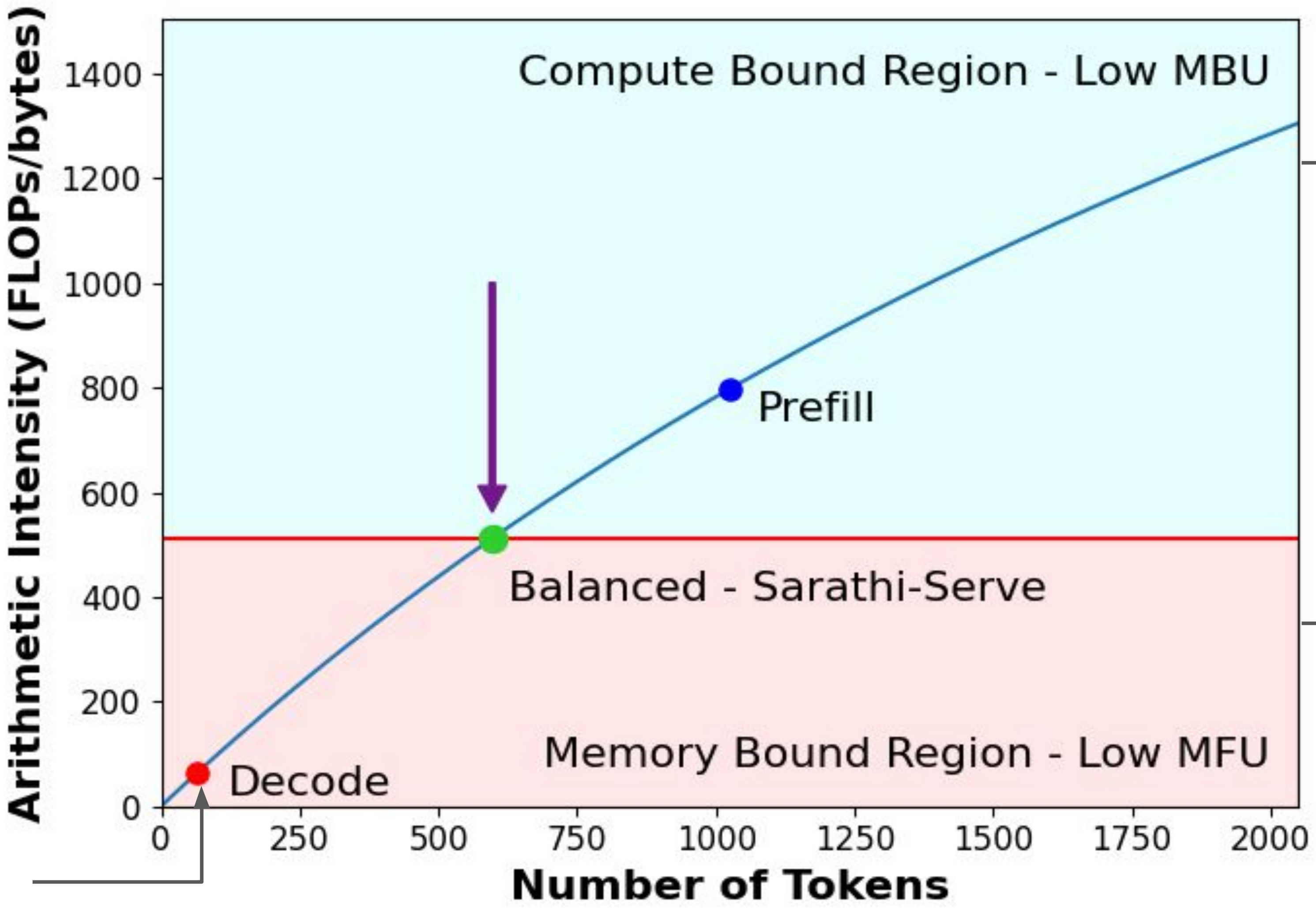




Key Insight

Prefill computation can be done at a marginal cost with careful batching 💡

🔎 Observation: Arithmetic Intensity Slack



Latency dominated by compute

Grows linearly with num tokens

Latency dominated by weight fetch time

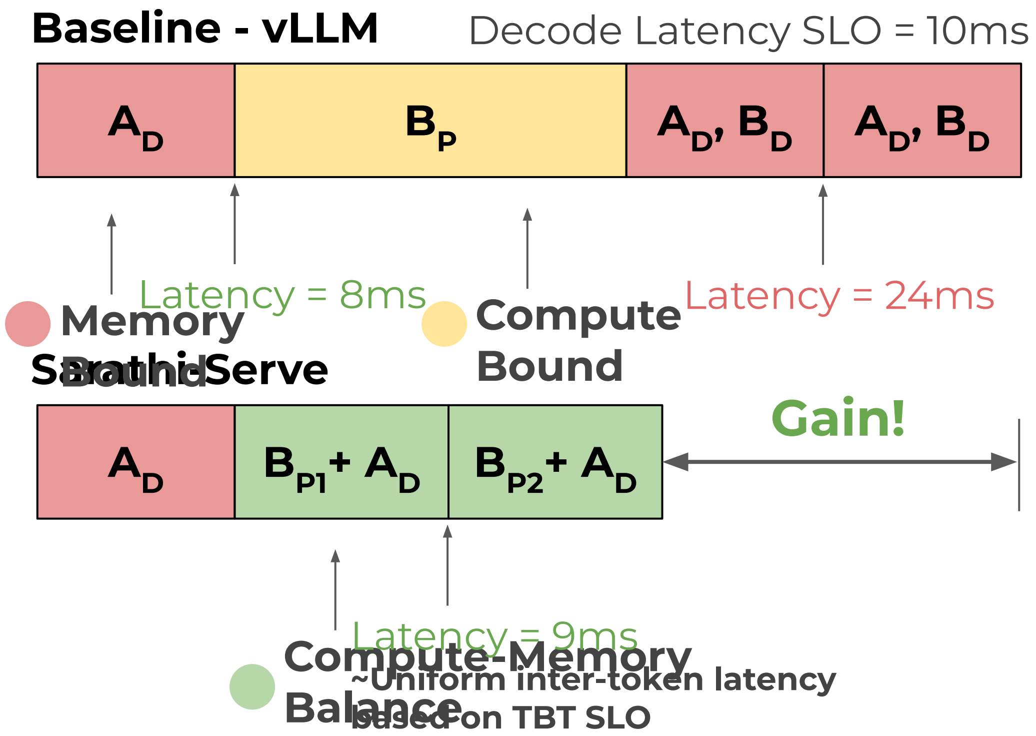
Independent of num tokens

Constrained due to memory overhead in decode phase

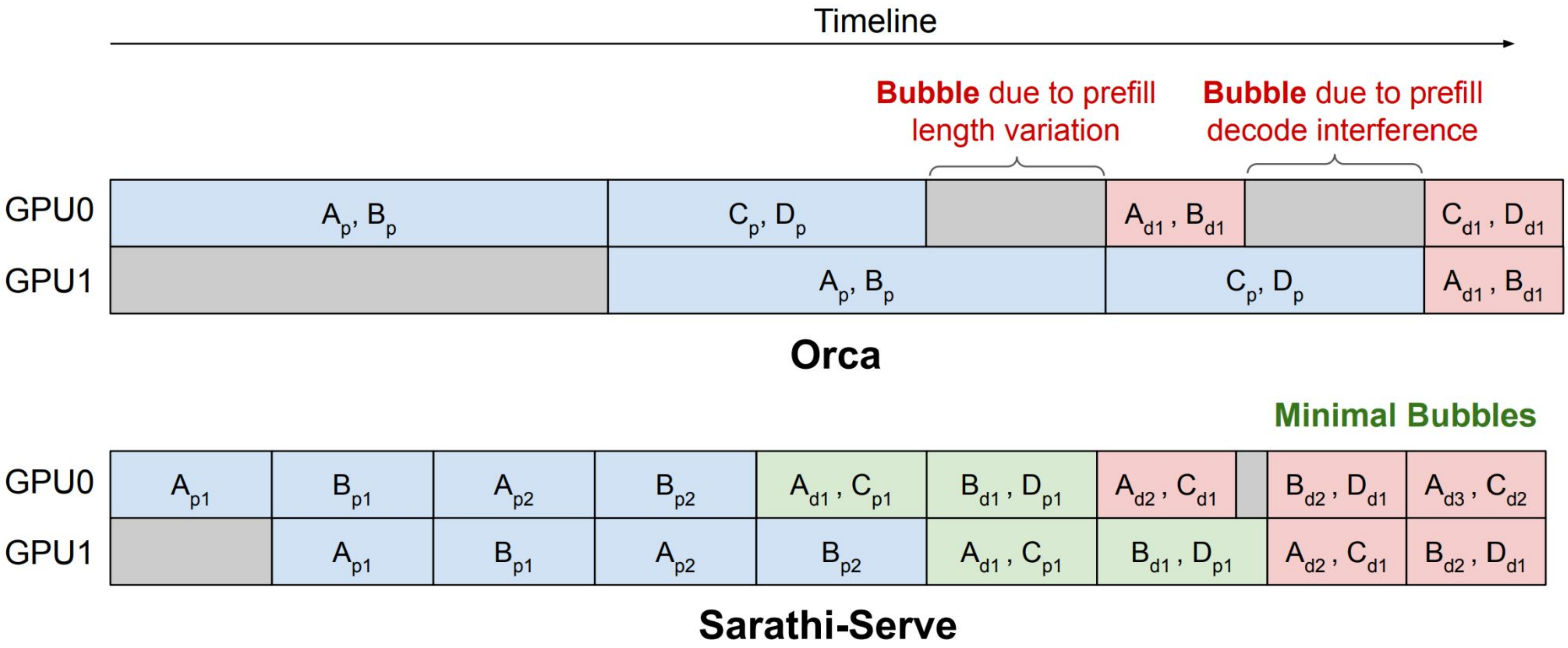
🧠 Stall-free Batching

Key Idea

🔪 Split large prefills into smaller chunks – just enough to consume the leftover compute budget in decode batches



Avoid Pipeline Bubbles



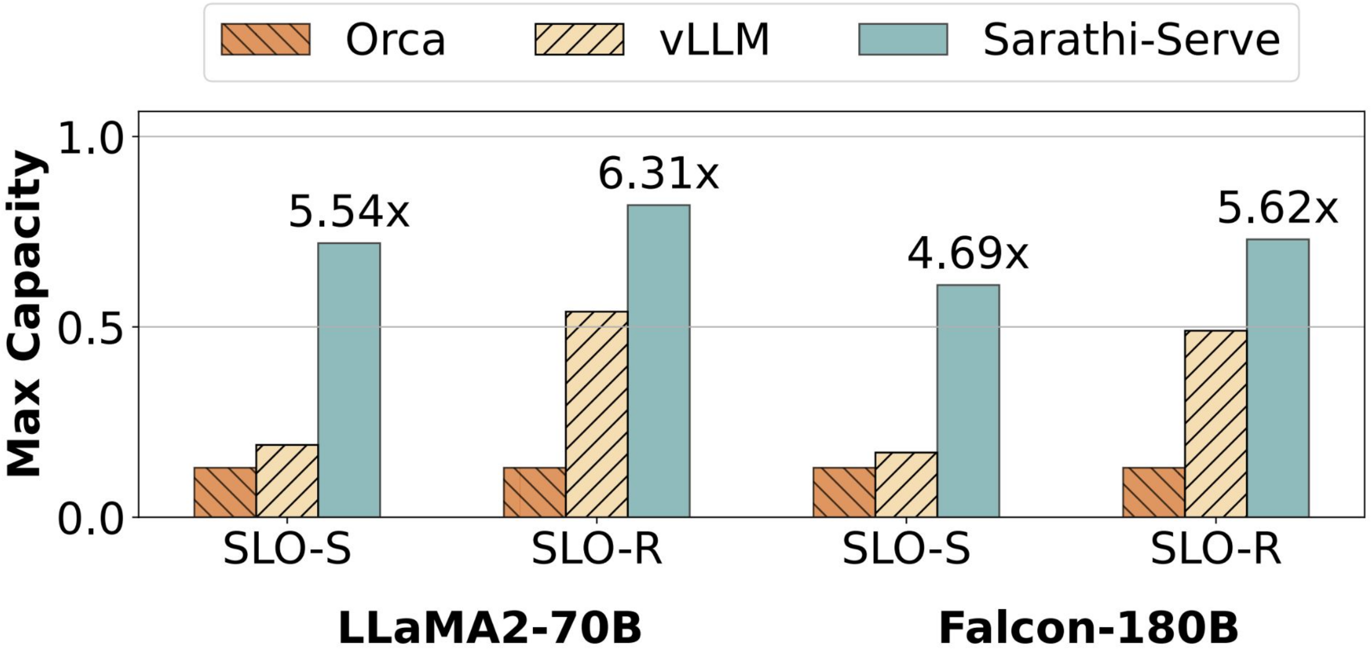
Evaluations 🧮

Serving Capacity under SLOs

adapt using different chunk sizes

Setup

ShareGPT4 trace on on A100 GPUs with strict (S) and relaxed (R) latency SLOs



5-6x higher capacity 🎊

🗒️ Summary

😨 **Problem:** State-of-the-art systems sacrifice decode latency to achieve higher throughput

💡 **Key Insight** - Low arithmetic intensity of decodes allows for adding compute intensive prefills with negligible decode latency cost

🧮 **Key Results** - We achieve optimality in both latency and throughput simultaneously leading up to 6x higher capacity under SLO constraints

🎉 **Industry Adoption** - Available in all major serving frameworks and more.

vLLM CentML anyscale

Paper GitHub Use in vLLM

Course Logistics

- Assignment 2 out soon
  - Performance modeling of LLM prefill and decode
  - Use AMD clusters for the assignment
  - Read document on AMD cluster usage
- Information on project logistics will be released next week

Throughput is Not All You Need

Maximizing **Goodput** in LLM Serving using **Prefill-Decode Disaggregation**

Hao Zhang @ Hao AI Lab & vLLM Team

DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving
Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, Hao Zhang

Prefill and Decode have Distinct Characteristics

- ## Prefill

Compute-bound

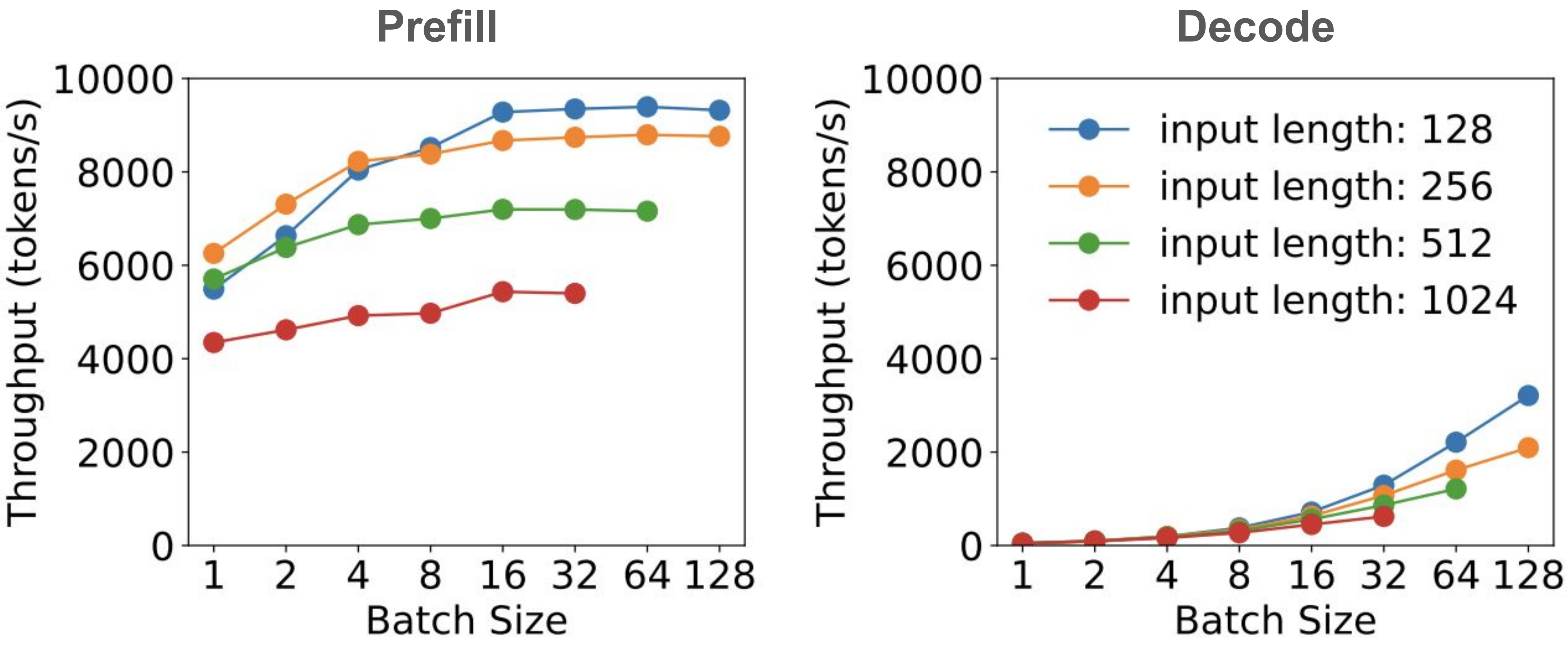
One prefill saturates compute. Batching across context tokens.

- ## Decode

Memory-bound

Must batch many requests together to saturate compute

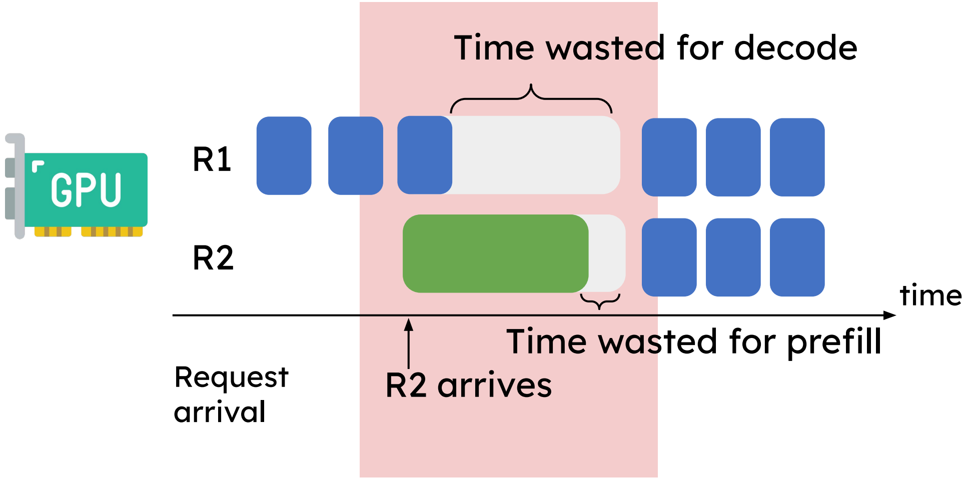
Prefill and Decode have Distinct Characteristics



Continuous Batching Causes Interference

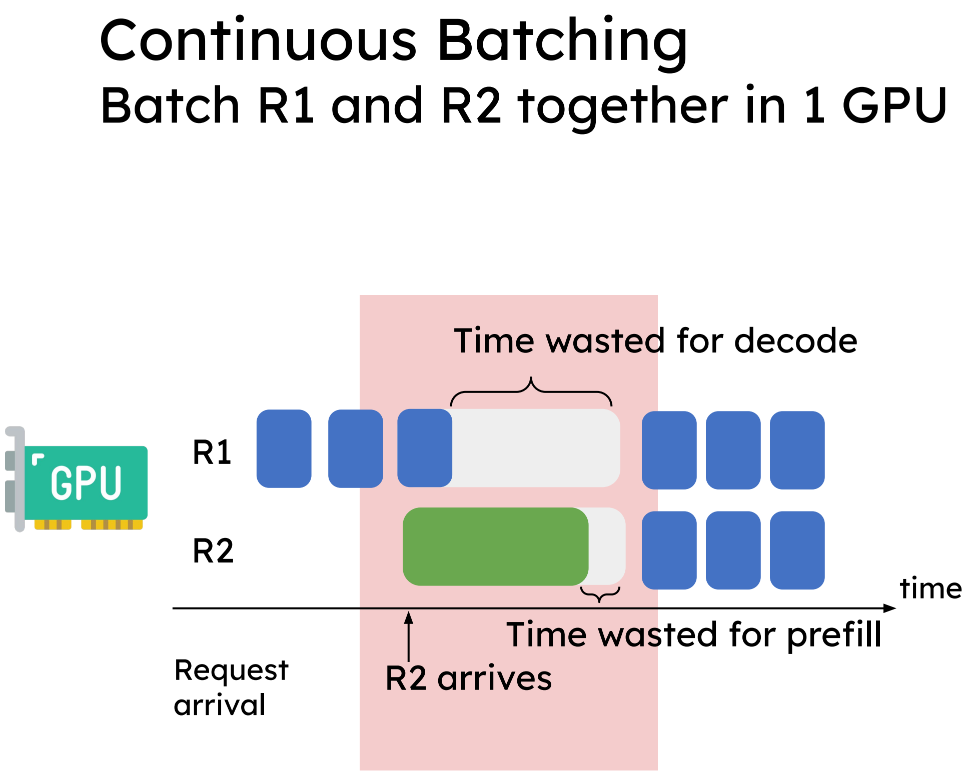
Continuous Batching

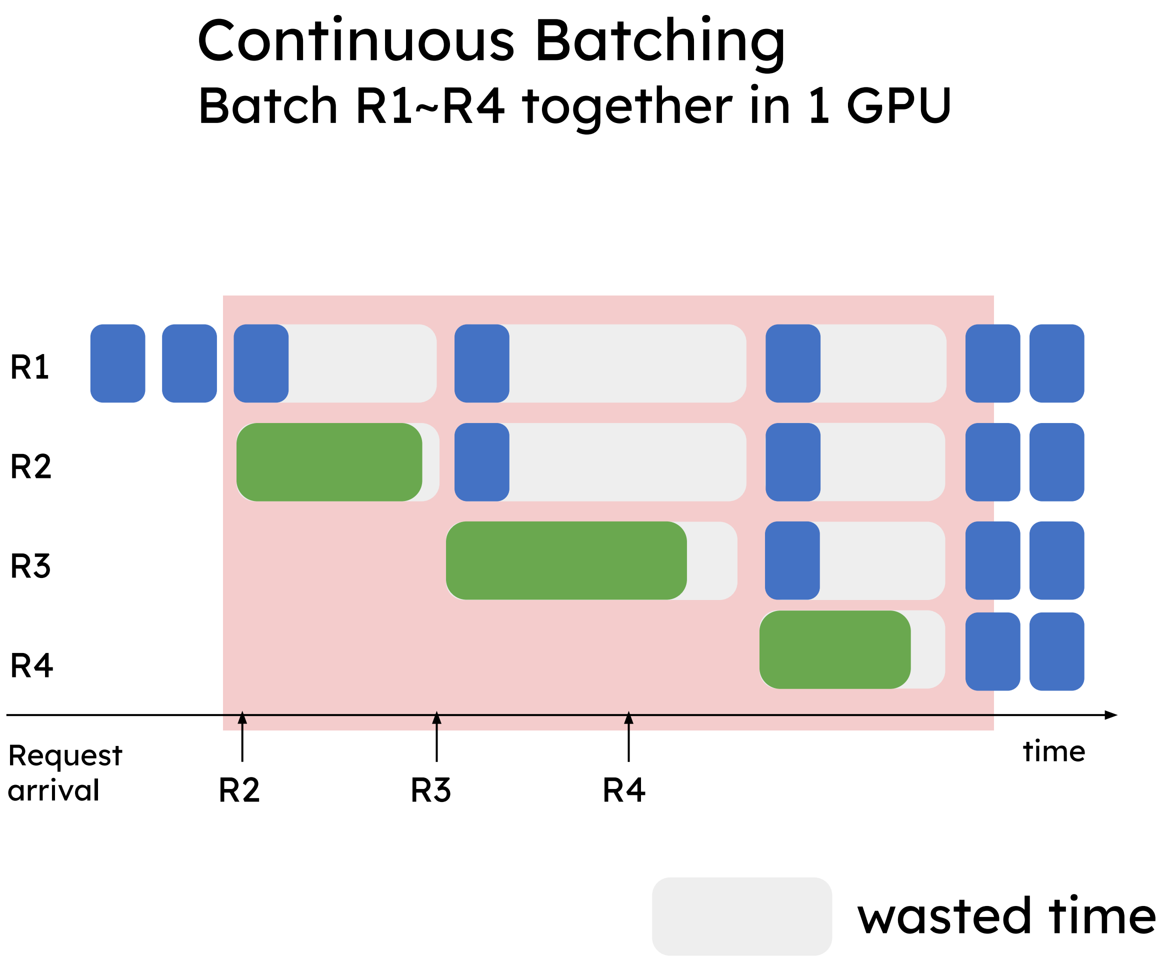
Batch R1 and R2 together in 1 GPU



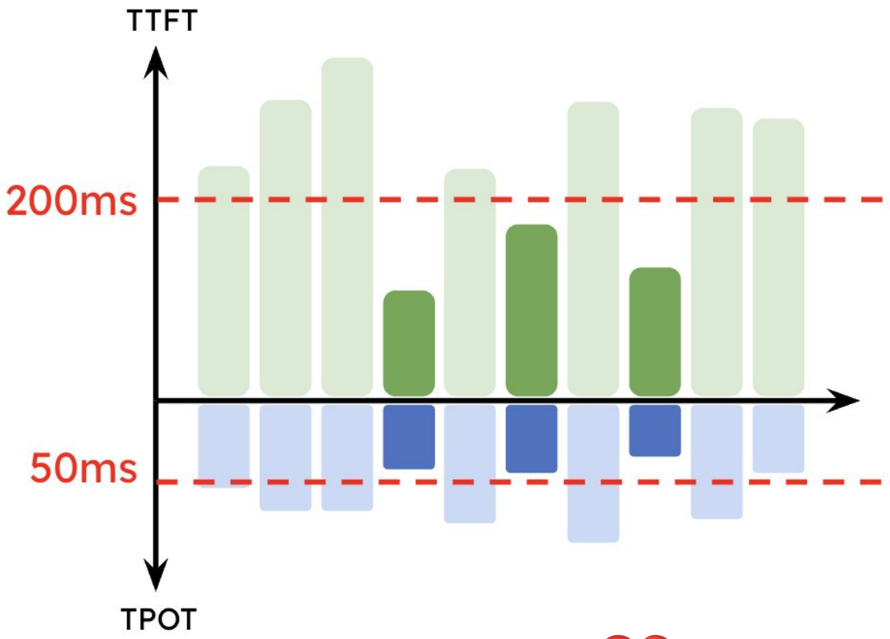
wasted time

Continuous Batching Causes Interference



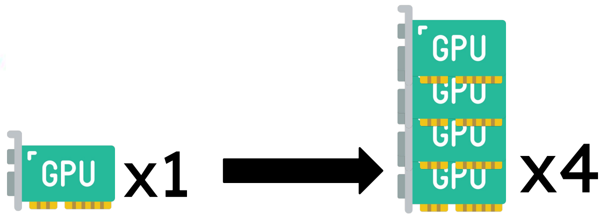


Colocation → Overprovision Resource to meet SLO



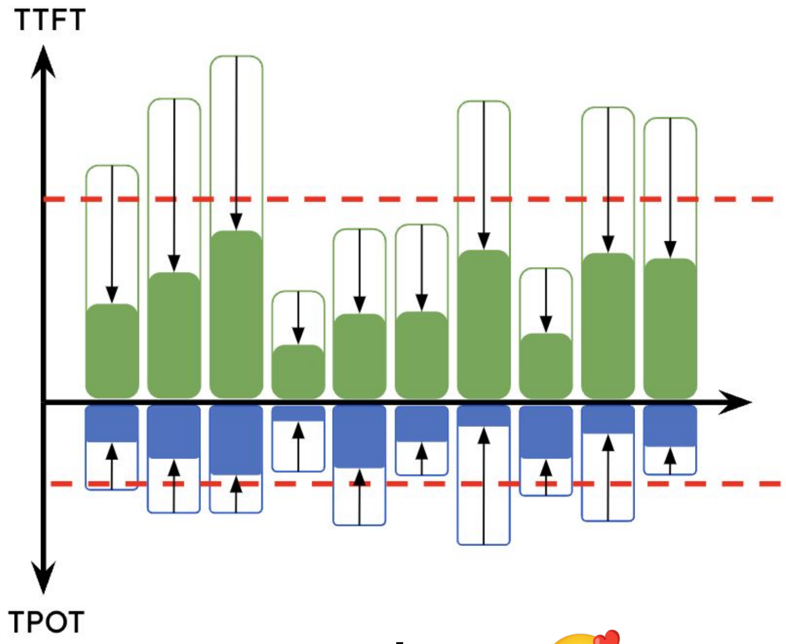
Poor UX 💔

lower cost 💰



add more GPU

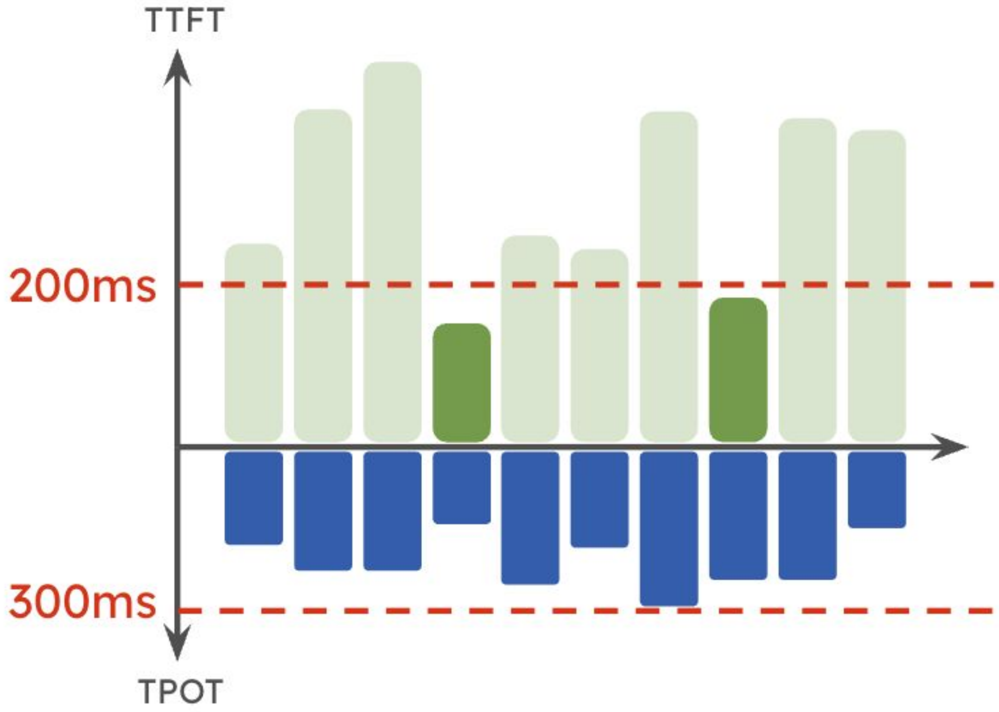
⚖️



Good UX 🥰

Higher cost 💰💰💰💰

Colocation → Coupled Parallelism

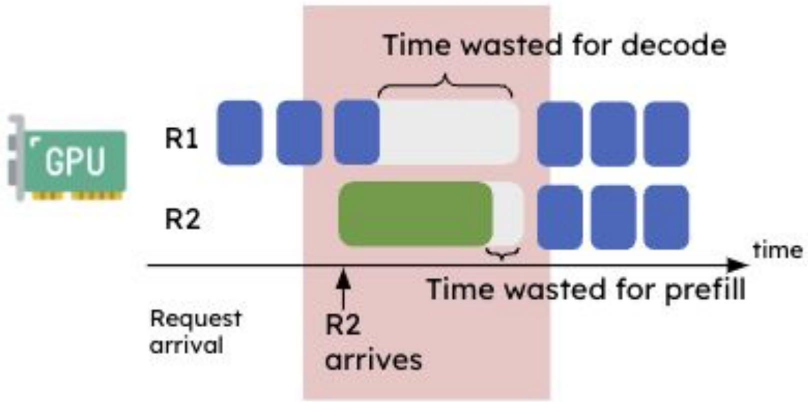


TTFT tight, TPOT loose

	PP	TP
• Prefill	😐	❤️
• Decode	❤️	😐

Prefill and Decode have different preferences

Summary: Problems caused by Colocation

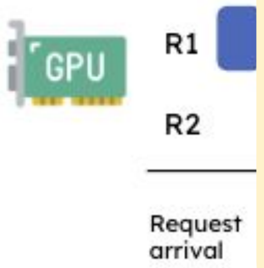


Continuous Batching
Cause Interference

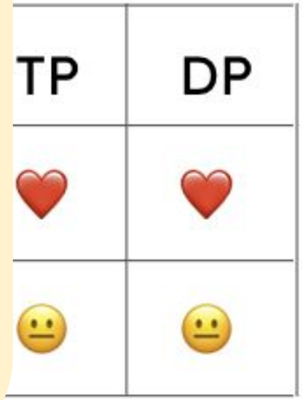
	PP	TP	DP
• Prefill	😐	❤️	❤️
• Decode	❤️	😐	😐

Coupled Parallelism Strategy

Summary: Problems caused by Colocation



Is there a better way to achieve better **Goodput per GPU?**

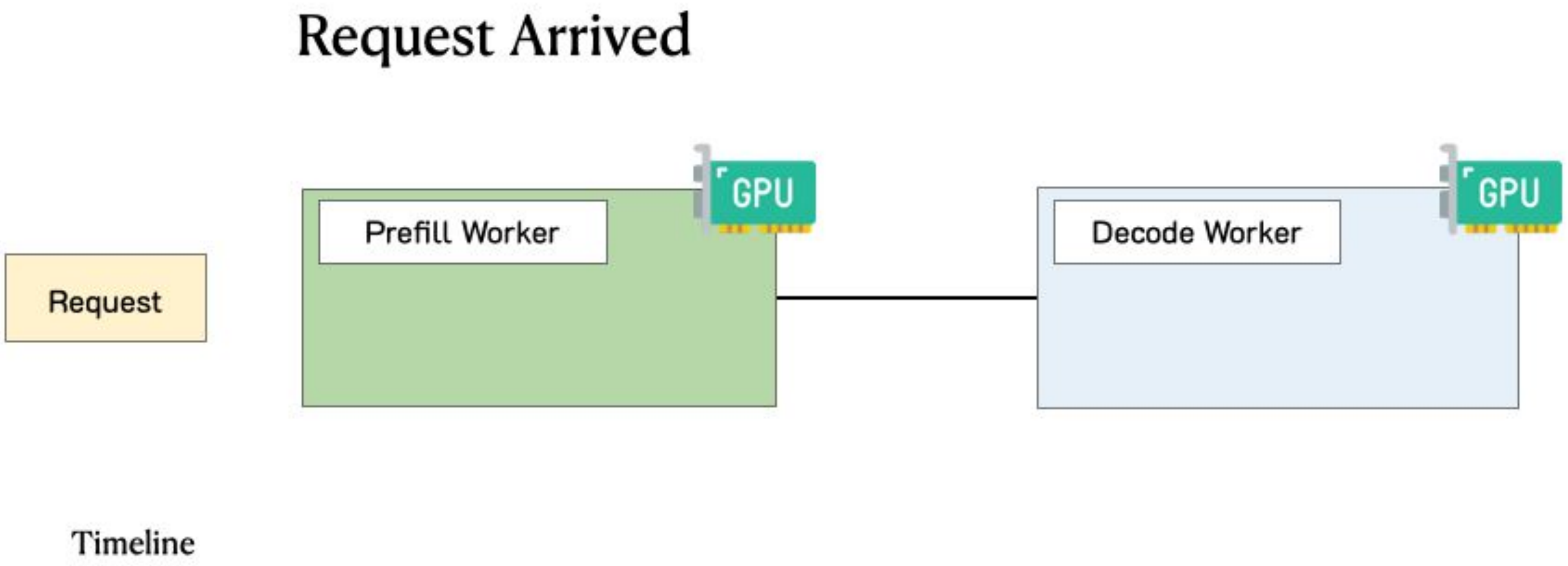


Continuous Batching
Cause Interference

Coupled Parallelism Strategy

DistServe: Disaggregating Prefill and Decode

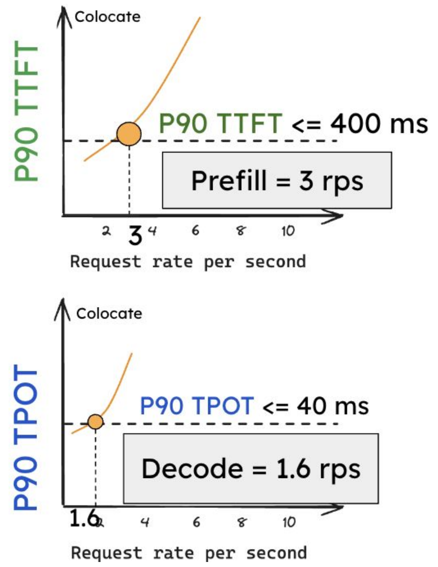
Disaggregation is a technique that

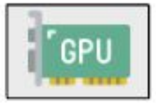


Disaggregation achieves better goodput

Colocate

1 GPU for both Prefill and Decode





Max System goodput

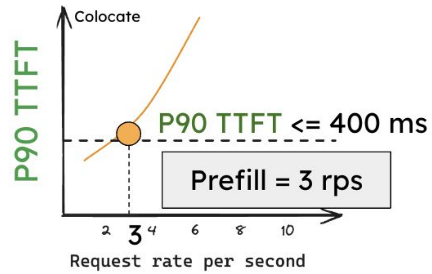
= Min(Prefill, Decode)

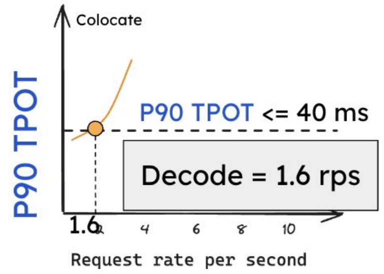
= 1.6 rps / GPU 😐

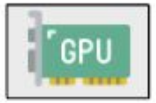
Disaggregation achieves better goodput

Colocate

1 GPU for both Prefill and Decode







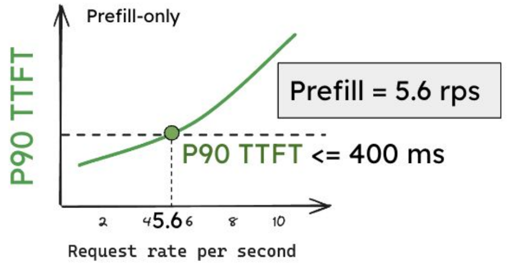
Max System goodput

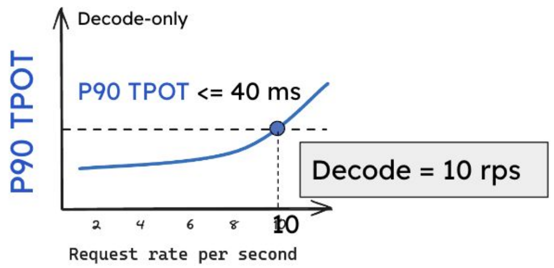
= Min(Prefill, Decode)

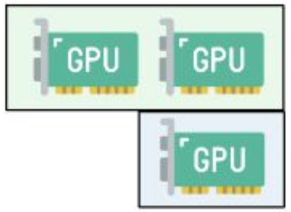
= 1.6 rps / GPU 😐

Disaggregate (2P1D)

2 GPU for Prefill + 1 GPU for Decode







Disaggregate (2P1D) goodput

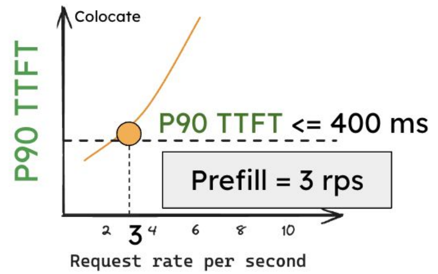
= Min (5.6 x **2**, 10) rps / 3 GPU

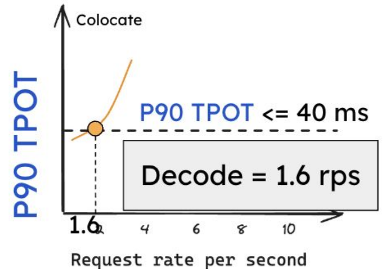
= 3.3 rps / GPU 😎

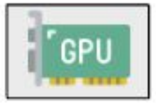
Disaggregation achieves better goodput

Colocate

1 GPU for both Prefill and Decode







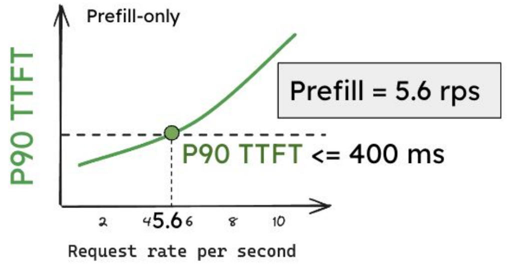
Max System goodput

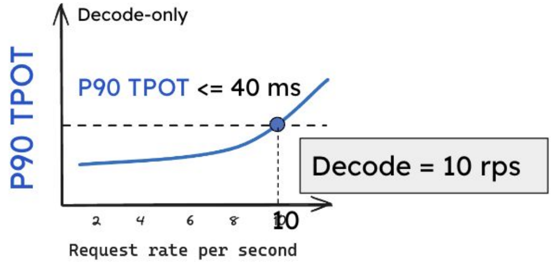
= Min(Prefill, Decode)

= 1.6 rps / GPU

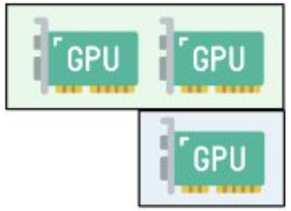
Disaggregate (2P1D)

2 GPU for Prefill + 1 GPU for Decode





Simple Disaggregation achieves **2x** goodput (per GPU)



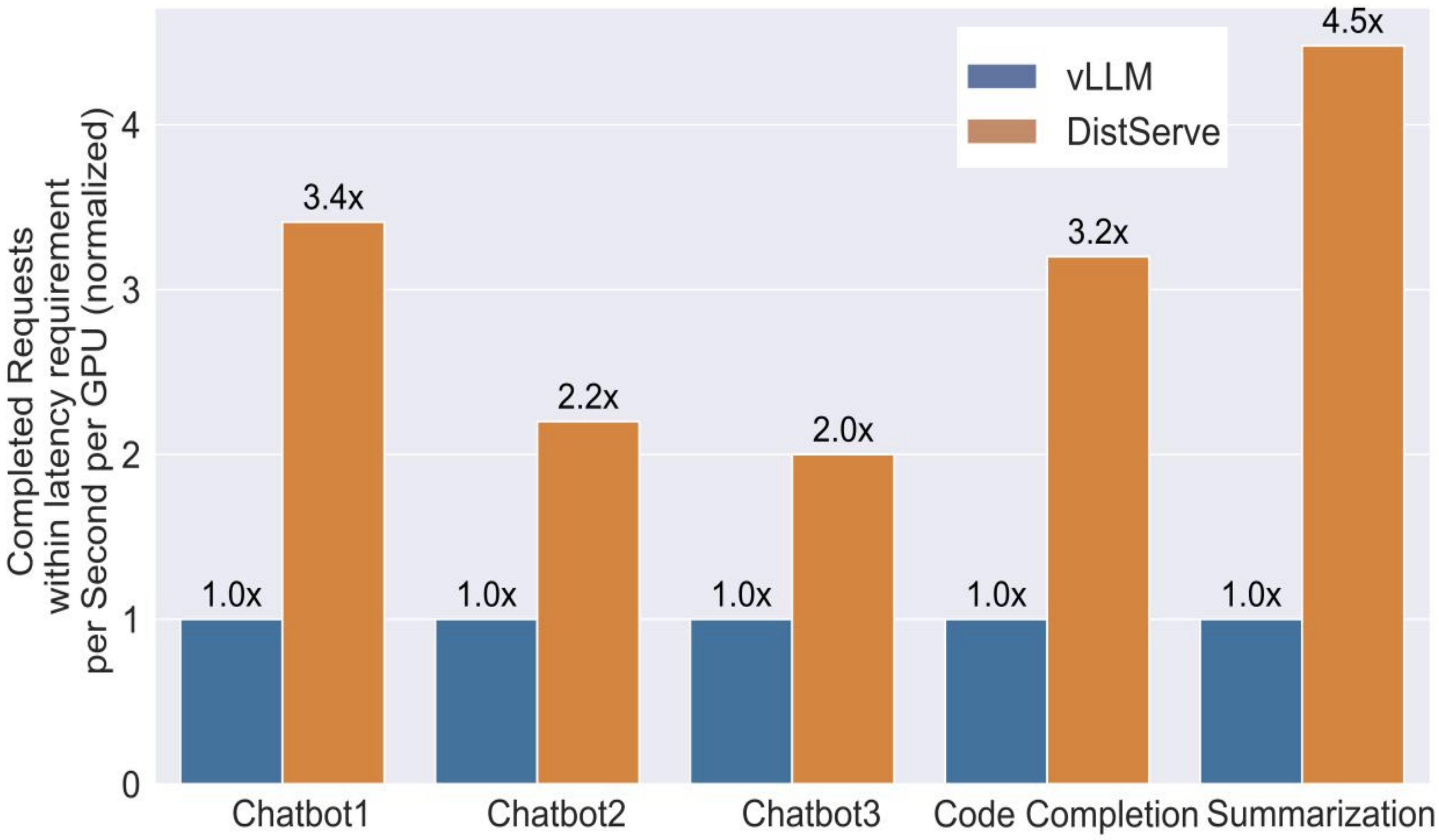
Disaggregate (2P1D) goodput

= Min (5.6 x **2**, 10) rps / 3 GPU

= 3.3 rps / GPU

What are issues or potential downsides with disaggregation?

Evaluation



Achieves 2.0x - 4.48x compared to vanilla vLLM

- Chatbot: 2.0 - 3.4x
- Code Completion: 3.2x
- Summarization: 4.5x

DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving
Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, Hao Zhang

Summary

- **Goodput** instead of Throughput
- **Disaggregation** is effective to optimize **goodput**!
- **DistServe** achieves 2.0x - 4.48x compared to vLLM
- Integrating into vLLM
- Already adopted by companies like anyscale