Queueing Predictions and LLMs Challenges and Open Problems

Part 1

Michael Mitzenmacher

Part 2

Rana Shahout

Harvard University

Survey Article That Goes With The Talk

Computer Science > Artificial Intelligence

[Submitted on 10 Mar 2025]

Queueing, Predictions, and LLMs: Challenges and Open Problems

Michael Mitzenmacher, Rana Shahout

Queueing systems present many opportunities for applying machine-learning predictions, such as estimated service times, to improve system performance. This integration raises numerous open questions about how predictions can be effectively leveraged to improve scheduling decisions. Recent studies explore queues with predicted service times, typically aiming to minimize job time in the system. We review these works, highlight the effectiveness of predictions, and present open questions on queue performance. We then move to consider an important practical example of using predictions in scheduling, namely Large Language Model (LLM) systems, which presents novel scheduling challenges and highlights the potential for predictions to improve performance. In particular, we consider LLMs performing inference. Inference requests (jobs) in LLM systems are inherently complex; they have variable inference times, dynamic memory footprints that are constrained by key-value (KV) store memory limitations, and multiple possible preemption approaches that affect performance differently. We provide background on the important aspects of scheduling in LLM systems, and introduce new models and open problems that arise from them. We argue that there are significant opportunities for applying insights and analysis from queueing theory to scheduling in LLM systems.

Subjects: **Artificial Intelligence (cs.AI)**; Data Structures and Algorithms (cs.DS)
Cite as: arXiv:2503.07545 **[cs.AI]**
(or arXiv:2503.07545v1 **[cs.AI]** for this version)
https://doi.org/10.48550/arXiv.2503.07545

Submission history

From: Rana Shahout [view email]
[v1] Mon, 10 Mar 2025 17:12:47 UTC (574 KB)

Simple Scheduling Example

- Suppose I have *m* short jobs requiring service time *s*, and *n* long jobs requiring service time *t.* I'm interested in average time in the system.
- I don't know which jobs are which, and schedule randomly.
  - Expected time is:
- I know which jobs are which, and schedule short jobs first.
  - Expected time is:

Simple Scheduling Example

- Suppose I have *m* short jobs requiring service time *s*, and *n* long jobs requiring service time *t.* I'm interested in average time in the system.
- I know which jobs are which, and schedule short jobs first.
  - Expected time is: $\frac{(m^2-m)s+(n^2-n)t+mns}{2(n+m)}$
- I don't know which jobs are which, and schedule randomly.
  - Expected time is: $\frac{(m+n)\big((m-1)s+(n-1)t\big)+mt+ns}{2(n+m)}$

Prediction Question

- What if we have predictions:
  - Short jobs are predicted long incorrectly with probability p
  - Long jobs are predicted short incorrectly with probability q

Prediction Question

- What if we have predictions:
  - Short jobs are predicted long incorrectly with probability *p*
  - Long jobs are predicted short incorrectly with probability *q*
- Expression is a little ugly. Nice homework exercise though.

Prediction Question

- What if we have predictions:
  - Short jobs are predicted long incorrectly with probability *p*
  - Long jobs are predicted short incorrectly with probability *q*
- Expression is a little ugly. Nice homework exercise though.
- Asymptotic gap from optimal
  - Predictions: $\frac{nm(t-s)(p+q)}{2(n+m)}$
  - Random: $\frac{nm(t-s)}{2(n+m)}$

Motivating Example: Search

Given a sorted array of integers A[1...n], and a query **q** check if **q** is in the array.

2	4	7	11	16	22	37	38	44	88	89	93	95	96	97	98	99

Motivating Example: Search

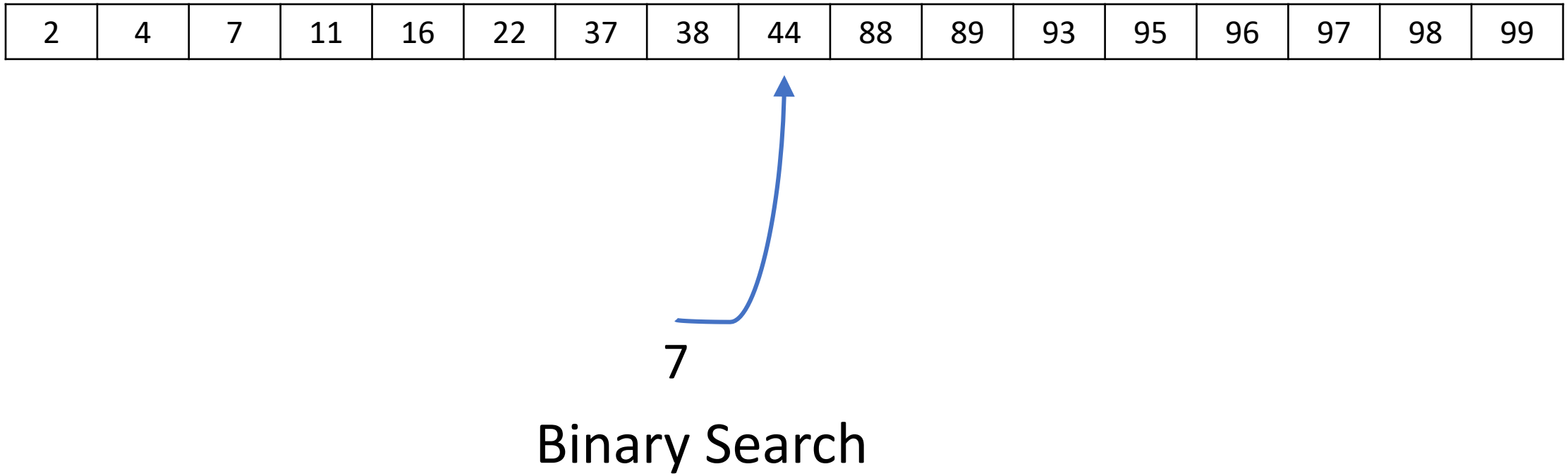
Given a sorted array of integers A[1...n], and a query **q** check if **q** is in the array.

2	4	7	11	16	22	37	38	44	88	89	93	95	96	97	98	99

7

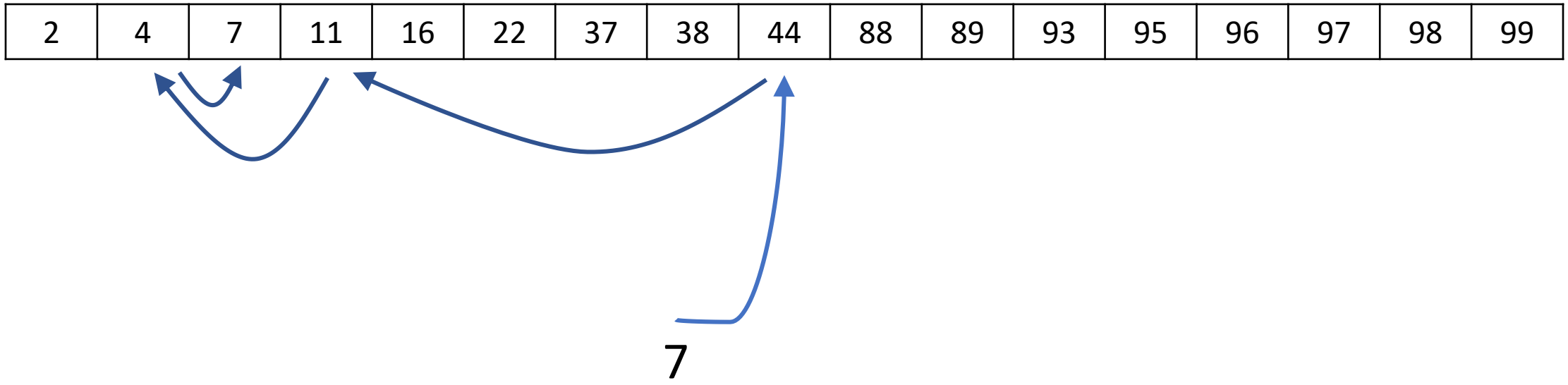
Motivating Example: Search

Given a sorted array of integers A[1...n], and a query **q** check if **q** is in the array.



Motivating Example: Search

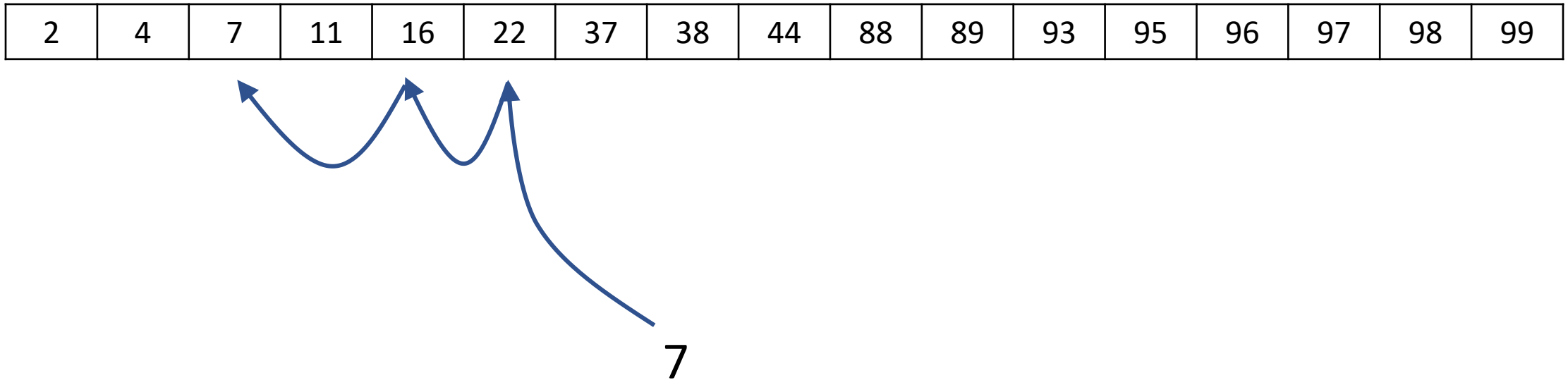
Given a sorted array of integers A[1...n], and a query **q** check if **q** is in the array.



Binary Search

Motivating Example: Search

Given a sorted array of integers A[1...n], and a query **q** check if **q** is in the array.



Predict where **q** appears; use doubling binary search.

Search Costs

- Binary search: $O(\log n)$
- Prediction-based search: $O(\log |\text{prediction error}|)$
  - Plus time to do the prediction.
- Robust: In the worst case, prediction-based search is also $O(\log n)$
  - Not “worse” than binary search (at least symptotically)
- Consistent: In the best case (and even “near-best-case”), prediction-based search is constant-time.
  - Essentially optimal with perfect information.

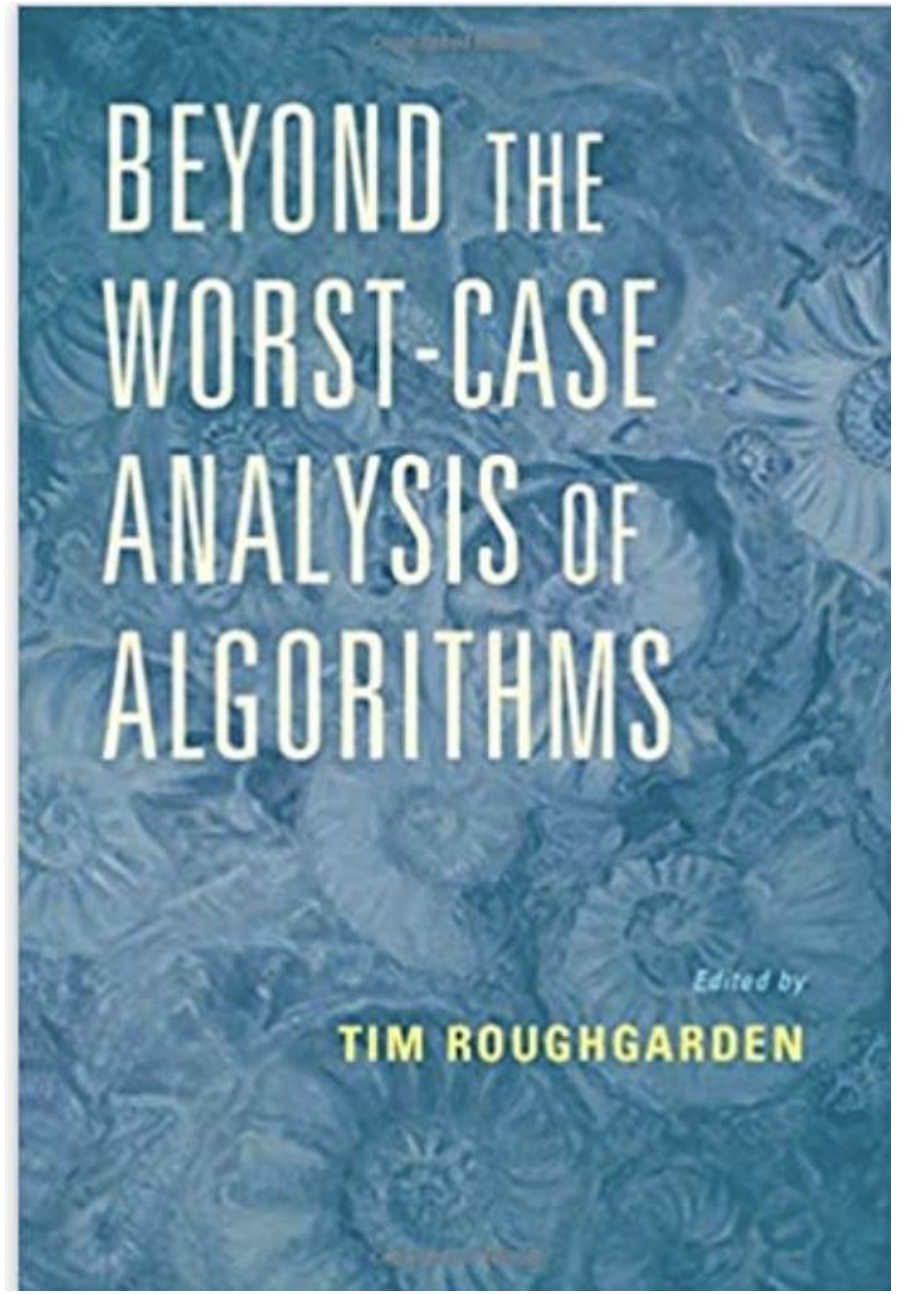
Search Costs

- Binary search: $O(\log n)$
- Prediction-based search: $O(\log |\text{prediction error}|)$
  - Plus time to do the prediction.
- Robust: In the worst case, prediction-based search is also $O(\log n)$
- Consistent: In the best case (and even "near-best-case"), prediction-based search is constant-time.

Machine Learning Can Improve Traditional Algorithms

Motivation

- “Traditional” algorithms based on worst-case analysis.
  - Very important analysis paradigm.
  - Has moved theory and practice forward dramatically.
- But “worst-case” isn’t everything.
- Theory has tried to move *beyond worst-case analysis*
  - Random/perturbed/limited inputs
  - Approximation algorithms/heuristics
- Machine learning offers possibilities for *new algorithmic paradigms*



Survey Article: Mitzenmacher and Vassilvitskii
Algorithms with Predictions, https://arxiv.org/abs/2006.09123

Algorithms with Predictions

'07 '09 '10 '17 '18 '19 '20 '21 '22 '23 '24 '25

Newest first

256 papers

Online Metric Matching: Beyond the Worst Case	Yang, Yu	arXiv '25	allocation / matching, online
With a Little Help From My Friends: Exploiting Probability Distribution Advice in Algorithm Design	Canonne, Chen, Mestre	arXiv '25	allocation / matching, online, prophet inequality
Robust Gittins for Stochastic Scheduling	Moseley, Newman, Pruhs, Zhou	arXiv '25	approximation, online, scheduling
Faster Global Minimum Cut with Predictions	Moseley, Niaparast, Singh	arXiv '25	graph problems, running time
Learning-Augmented Frequent Directions	Aamand, Chen, Gollapudi, Silwal, Wu	arXiv '25	streaming
Efficiently Solving Discounted MDPs with Predictions on Transition Matrices	Lixing Lyu, Jiashuo Jiang, Wang Chi Cheung	arXiv '25	MDP, running time, sample complexity
Mechanisms for Selling an Item Among a Strategic Bidder and a Profiled Agent	Caragiannis, Kalantzis	arXiv '25	auctions, game theory / mechanism design
Incremental Approximate Single-Source Shortest Paths with Predictions	McCauley, Moseley, Niaparast, Niaparast, Singh	arXiv '25	dynamic / data structure, graph problems, shortest path
Utilitarian Distortion with Predictions	Filos-Ratsikas, Kalantzis, Voudouris	arXiv '25	game theory / mechanism design
Pareto-Optimality, Smoothness, and Stochasticity in Learning-Augmented One-Max-Search	Benomar, Croissant, Perchet, Angelopoulos	arXiv '25	online, search
Algorithms with Calibrated Machine Learning Predictions	Shen, Vitercik, Wikum	arXiv '25	online, rent-or-buy, scheduling
Polynomial Time Learning-	Bampis, Escoffier,	arXiv '25	approximation, auctions, graph problems, routing / TSP, scheduling

dynamic / data structure
online
running time
approximation
streaming
game theory / mechanism design
differential privacy
survey
prior/related work

allocation / matching
auctions
caching / paging
clustering
computational geometry
convex body chasing
covering
distributed algorithms
dynamic pricing
experimental evaluation

Using Machine Learning: Predictions

- Machine learning can be used to provide predictions
- Given *good* predictions, can we get better than worst-case algorithms.
  - Question: what should we allow when predictions are bad?
- Would like **provable** statements.
  - Because I'm theoretically inclined.
  - If my predictions are X-good, my algorithm's performance will be Y.
- We call this algorithms with predictions or learning-augmented algorithms.

Queues

Standard Queueing Model

- Arrival Process
  - Usually Poisson arrivals; exponentially distributed service times
- Service Time
  - For theory, exponential service times are nice
  - In practice, some kind of heavy-tailed distribution
  - In simple models, service time is *unknown*
- Service Discipline
  - First-In First-Out (FIFO), = First-Come First-Served
  - Others possible
  - Preemption

Main Result for Standard Queues

- For M/M/1 queues
  - FIFO queues
  - Poisson arrivals of rate $\lambda < 1$
  - Exponential service times with mean 1
- The steady state expected time in the system is

$$1/(1-\lambda)$$

Known Service Times

- Can do better if service times are known.
  - Shortest job first. (Non-preemptive.)
  - Shortest remaining processing time. (Preemptive.)
  - Preemptive shortest job first. (Preemptive.)
- All of these have known formulae for Poisson arrivals, general service time distributions.

Predicted Service Times

- Suppose we have an ML algorithm that can predict service times. Then we can consider policies:
  - Shortest predicted job first (SPJF).
  - Shortest predicted remaining service time (SPRPT).
  - Preemptive shortest predicted job first (PSPJF).
- A natural probabilistic model:
  - Service distribution given by density function $g(x,y)$, representing density of jobs with true service time x and predicted service time y.

Shortest Job First Queues

- Let *S* be service distribution, f_s the density.
- Analysis uses that once a job arrives, can ignore jobs with higher service times that arrive.
- So only important quantities for a job with service time *x* are:

$$\rho = \lambda \int_{t=0}^{\infty} t f_s(t) \mathrm{dt}$$ Rate of inflow of work

$$\rho_x = \lambda \int_{t=0}^{x} t f_s(t) \mathrm{dt}$$ Rate of inflow of work for jobs with service $\leq x$

- Expected waiting time in queue for job of service time *x* given by

$$E[W(x)] = \frac{\rho E[S^2]}{2E[S]](1-\rho_x)^2}$$

Shortest Predicted Job First Queues

- Let *S* be service distribution, f_s the density, *g*(*x*,*y*) joint service-prediction density, f_p the density of predicted service times.
- Analysis uses that once a job arrives, can ignore jobs with higher predicted service times that arrive.
- So only important quantities for a job with service time *x* are:

$$\rho = \lambda \int_{t=0}^{\infty} t f_s(t) \text{dt}$$

Rate of inflow of work

$$\rho'_y = \lambda \int_{t=0}^{y} \int_{x=0}^{\infty} x g(x,t) \text{ dx dt}$$

Rate of inflow of work for jobs with predicted service $\leq y$

- Expected waiting time in queue for job of predicted time *y* given by

$$E[W'(y)] = \frac{\rho E[S^2]}{2E[S](1-\rho'_y)^2}$$

Price of Misprediction

- A competitive ratio formulation: price of misprediction
  - What is the (worst-case) ratio of the expected total waiting time using the ML algorithm compared to having perfect information?
  - Not vs OPT, but vs algorithm
- End result: by integrating over service/predicted service times, get a "simple" formula for price of misprediction for shortest job first.

$$\frac{\int_{y=0}^{\infty} \frac{f_p(y)}{(1-\rho' y^2)}\, dy}{\int_{x=0}^{\infty} \frac{f_s(x)}{(1-\rho_x{}^2)}\, dx}$$

SOAP : A Key Analysis Method

- These policies can be analyzed directly
  - But there's a useful general methodology, that is useful for scheduling with predictions
- SOAP: Schedule Ordered by Age-based Priority
  - Ziv Scully, Mor Harchol-Balter, Alan Scheller-Wolf
- M/G/1 queues: can calculate expected time in system in equilibrium
  - Job priority determined by a "rank" (lowest rank wins)
  - Rank can depend on a job type (known, fixed on entry) and the amount of service received.
    - Example: SRPT– a job's type is its service time s, received service a, rank would be $s - a$.
    - Example: SPRPT– a job's type is its service and predicted time (s,z), received service a, rank would be $z - a$.

Scheduler

- More generally, instead of thinking of policies, think of a *scheduler* decdides what job gets served at any point in time.
- SOAP allows analysis of a broader class of interesting schedulers.
  - Schedulers based on suitable rank functions.

Test Examples

Service times are exponentially distributed, mean 1
Prediction times for a job of true size *x* is exponentially distributed with mean *x*

λ	SRPT Eqns	SRPT Sim	SPRPT Eqns	SPRPT Sim	FIFO Eqns
0.5	1.4254	1.4251	1.6531	1.6588	2.00
0.6	1.6041	1.6039	1.9305	1.9397	2.50
0.7	1.8746	1.8757	2.3539	2.3684	3.33
0.8	2.3528	2.3519	3.1168	3.1376	5.00
0.9	3.5521	3.5486	5.04808	5.0973	10.00
0.95	5.5410	5.5466	8.3221	8.4075	20.00
0.98	10.4947	10.5003	16.6239	16.7852	50.00
0.99	17.6269	17.6130	28.7302	28.7847	100.00

λ	SJF Eqns	SJF Sims	SPJF Eqns	SPJF Sims	FIFO Eqns
0.5	1.7127	1.7128	1.7948	1.7949	2.00
0.6	1.9625	1.9625	2.1086	2.1087	2.50
0.7	2.3122	2.3121	2.5726	2.5730	3.33
0.8	2.8822	2.8828	3.3758	3.3760	5.00
0.9	4.1969	4.1987	5.3610	5.3609	10.00
0.95	6.2640	6.2701	8.6537	8.6541	20.00
0.98	11.2849	11.2734	16.9502	16.9782	50.00
0.99	18.4507	18.4237	29.0536	29.1162	100.00

Equations and simulations match.
Significant improvements over no information, especially under high loads.

Open Questions

- Better/more realistic models of predictions?
  - Models of point predictions.
  - Predictions as distributions, instead of point values.
  - Predictions that degrade over time.
- Extensions of SOAP?
  - Rank cannot depend on queue state, such as number of jobs in the queue.
- Gittins-optimal policies?

Weak Predictions: Single Bit

- What if you could only get a “single bit” hint?
  - Is the job “short” or “long”?
  - Bigger than some threshold, or smaller?
  - Put it in front, or in back?
- A natural model
  - May be an easier prediction problem
  - May be easier to implement (no need to remember times)
  - May be limited in communication

Single Bit Model

- Use a threshold: above a threshold is big, below is small.
- Can consider perfect single bit hints, or hints from predictions.
- Equations derived in terms of arrival rate, service distribution, threshold, prediction scheme. (See paper: Queues with Small Advice.)
  - Specific example: exponential model -- job of true size *x* has predicted size exponentially distributed with mean *x*.
- Here we choose optimal threshold from experiments.

Theoretical Results

- Analysis follows standard methods… but model leads to fun integrals.
- Expected sojourn time for exponential service time, non-preemptive, exponential predictions involves modified Bessel functions of the first and second kind.

$$S_{e,n,e} = \frac{\lambda(1-\lambda(1-2\sqrt{T}K_1(2\sqrt{T})))}{(1-\lambda)(1-\lambda(1-2TK_2(2\sqrt{T})))} + 1.$$

- Expected sojourn time for Weibull service time, non-preemptive, exponential predictions involves the Meijer G-function.

$$S_{w,n,e} = \frac{3\lambda\left(1-\lambda\left(1-\sqrt{\frac{T}{2\pi}}G_{0,3}^{3,0}\left(\frac{T}{2} \mid -\frac{1}{2},0,\frac{1}{2}\right)\right)\right)}{(1-\lambda)\left(1-\lambda\left(1-\sqrt{\frac{T^3}{2\pi}}G_{0,3}^{3,0}\left(\frac{T}{2} \mid -\frac{3}{2},0,\frac{1}{2}\right)\right)\right)} + 1.$$

Results for Single Bit Predictions

λ	FIFO	THRESHOLD NO PREEMPT	THRESHOLD PREEMPT	SRPT	PREDICTION NO PREEMPT	PREDICTION PREEMPT	SPRPT
0.50	2.000	1.783	1.564	1.425	1.850	1.698	1.659
0.60	2.500	2.089	1.814	1.604	2.209	2.013	1.940
0.70	3.333	2.542	2.203	1.875	2.761	2.517	2.369
0.80	5.000	3.329	2.910	2.355	3.757	3.451	3.143
0.90	10.00	5.278	4.755	3.552	6.366	5.960	5.097
0.95	20.00	8.535	7.914	5.532	10.848	10.372	8.424
0.98	50.00	16.495	15.735	10.436	22.418	21.909	16.696

Table 1. Results for exponentially distributed service times. Prediction results are using exponential predictions.

Prediction times for a job of true size *x* is exponentially distributed with mean *x*
For threshold schemes, consider if prediction is above or below threshold

Results for Single Bit Predictions

	FIFO	THRESHOLD NO PREEMPT	THRESHOLD PREEMPT	SRPT	PREDICTION NO PREEMPT	PREDICTION PREEMPT	SPRPT
0.50	4.000	3.012	1.608	1.411	3.155	1.736	1.940
0.60	5.500	3.676	1.867	1.574	3.918	2.062	2.280
0.70	8.000	4.565	2.258	1.813	4.983	2.568	2.750
0.80	13.00	5.955	2.951	2.217	6.721	3.481	3.519
0.90	29.00	8.940	4.649	3.154	10.630	5.790	5.224
0.95	58.00	13.223	7.448	4.517	16.546	9.846	7.788
0.98	148.0	22.451	15.194	7.666	29.346	20.918	13.404

Table 2. Results for Weibull distributed service times. Prediction results are using exponential predictions.

Prediction times for a job of true size *x* is exponentially distributed with mean *x*
For threshold schemes, consider if prediction is above or below threshold

Open Problems

- Better/more realistic prediction models?
- Can we optimize predictions in this setting?
  - Predicting a short job as long is worse than predicting a long job as short.
- Tail behaviors when using 2 or more classes?
  - See Chen and Dong: Scheduling with service-time information: The power of two priority classes

Obvious Shortcoming for SRPT

- When a prediction is smaller than a job size, predicted remaining service time will be 0.
- Very bad if a big job is predicted small.
- We should ***do something*** for jobs that sit at the front of the queue with predicted remaining service time 0.
- Examined in: Uniform Bounds for Scheduling with Job Size Estimates, Scully, Grosof, Mitzenmacher

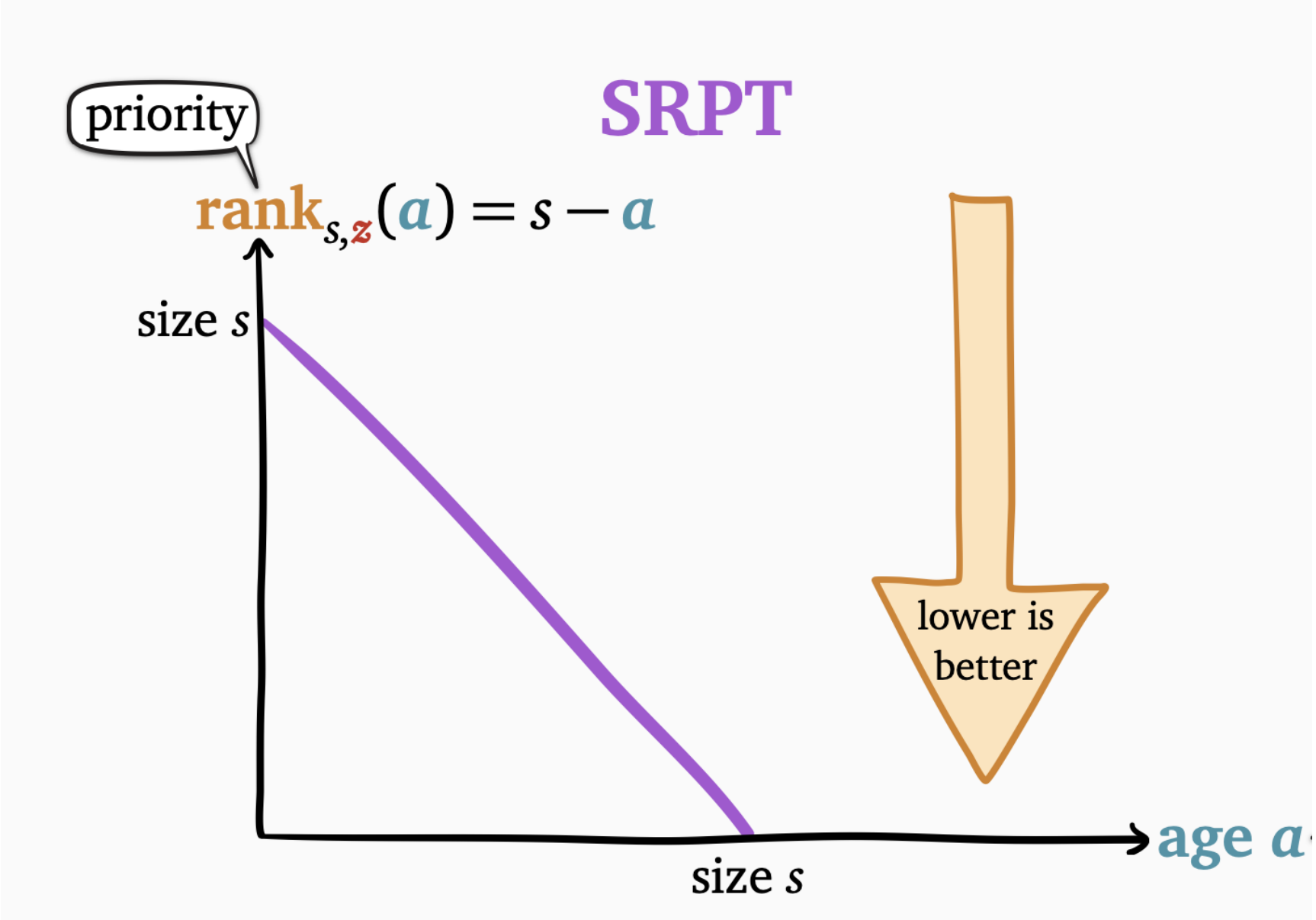
What We Want From a Scheduling Policy

- Consistent : Near optimal when predictions are good.
- Robust : Not too much worse than not using predictions when predictions are bad.
- Graceful degradation (smoothness) : performance degrades gracefully as predictions get worse.

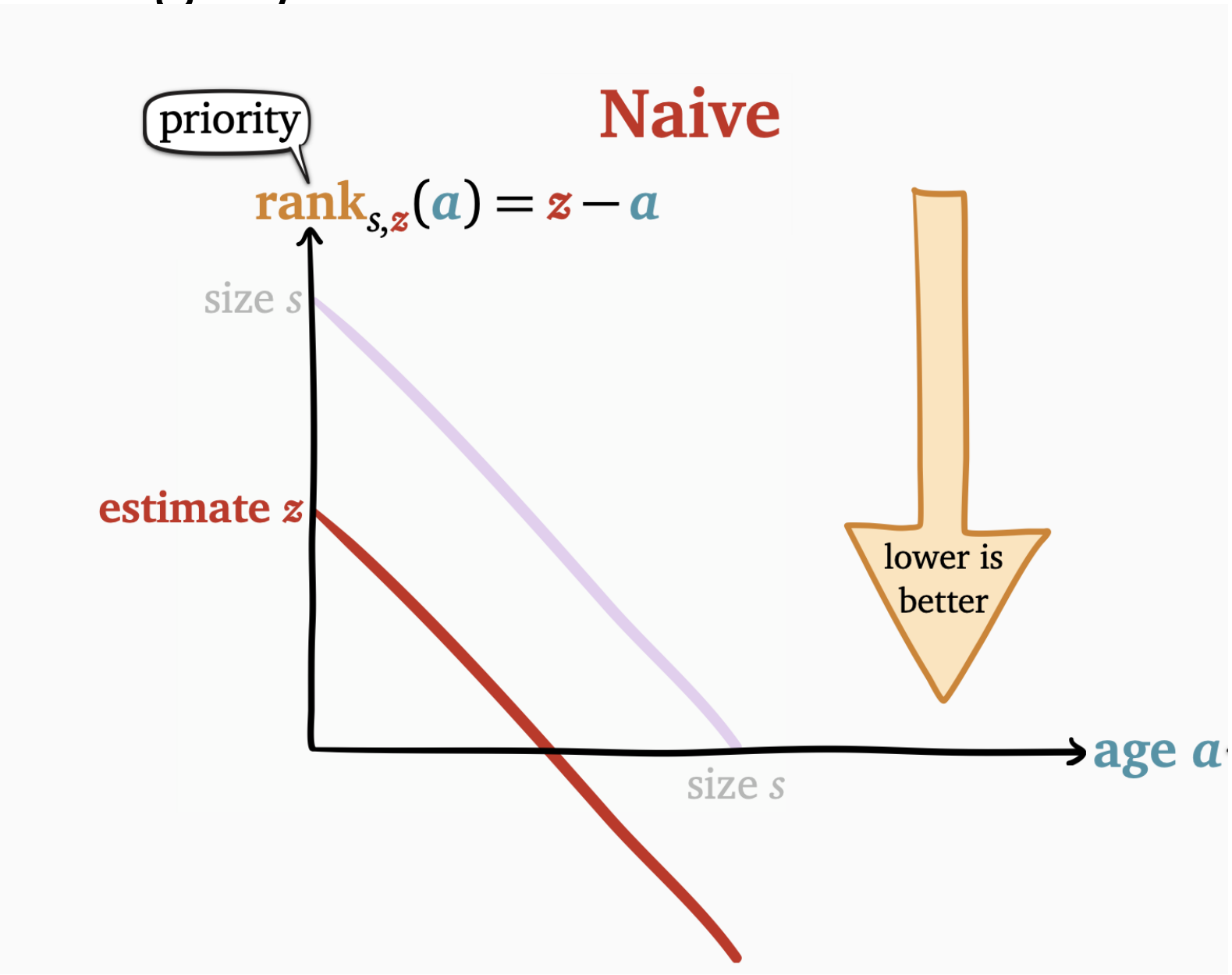
M/G/1 Setting with Predictions

- Performing well with arbitrary errors is too much to ask for.
- Model: (β, α)-bounded noise [β for "below", α for "above"]
  - True size is *s*, then estimate *z* governed by joint distribution on pairs (*s*,*z*), with $z \in [\beta s, \alpha s]$.
- We find a policy P that is
  - C-consistent : $E[T_P]/E[T_{SRPT}] \to C$ as $\beta, \alpha \to 1$ $(C = 1)$
  - G-graceful : $E[T_P]/E[T_{SRPT}] \leq G\,\alpha/\beta$ for all β, α $(G = 3.5)$
- Show constant robustness, $E[T_P]/E[T_{SRPT}] \leq R$ for all β, α, is not possible.

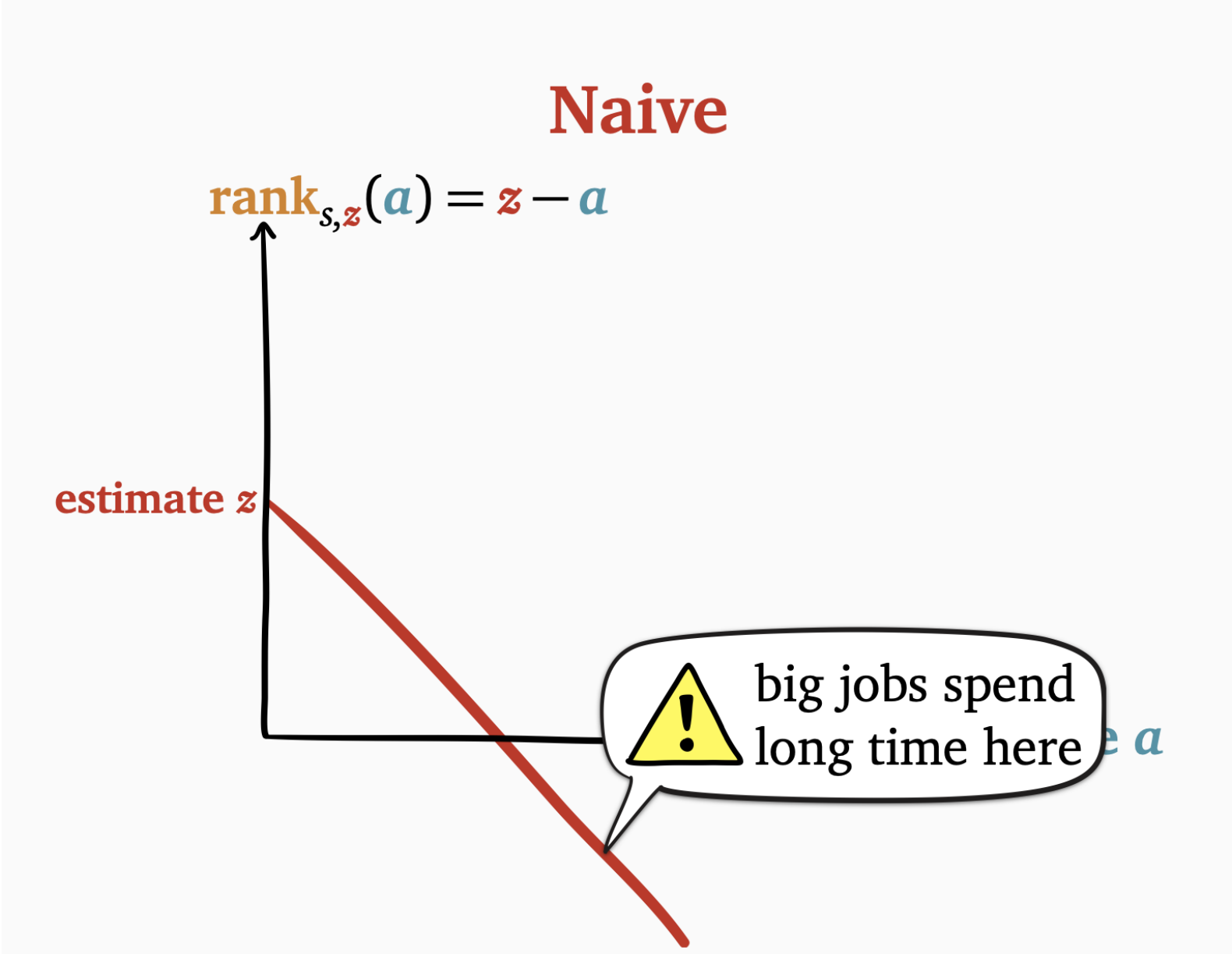
Scheduling by Rank



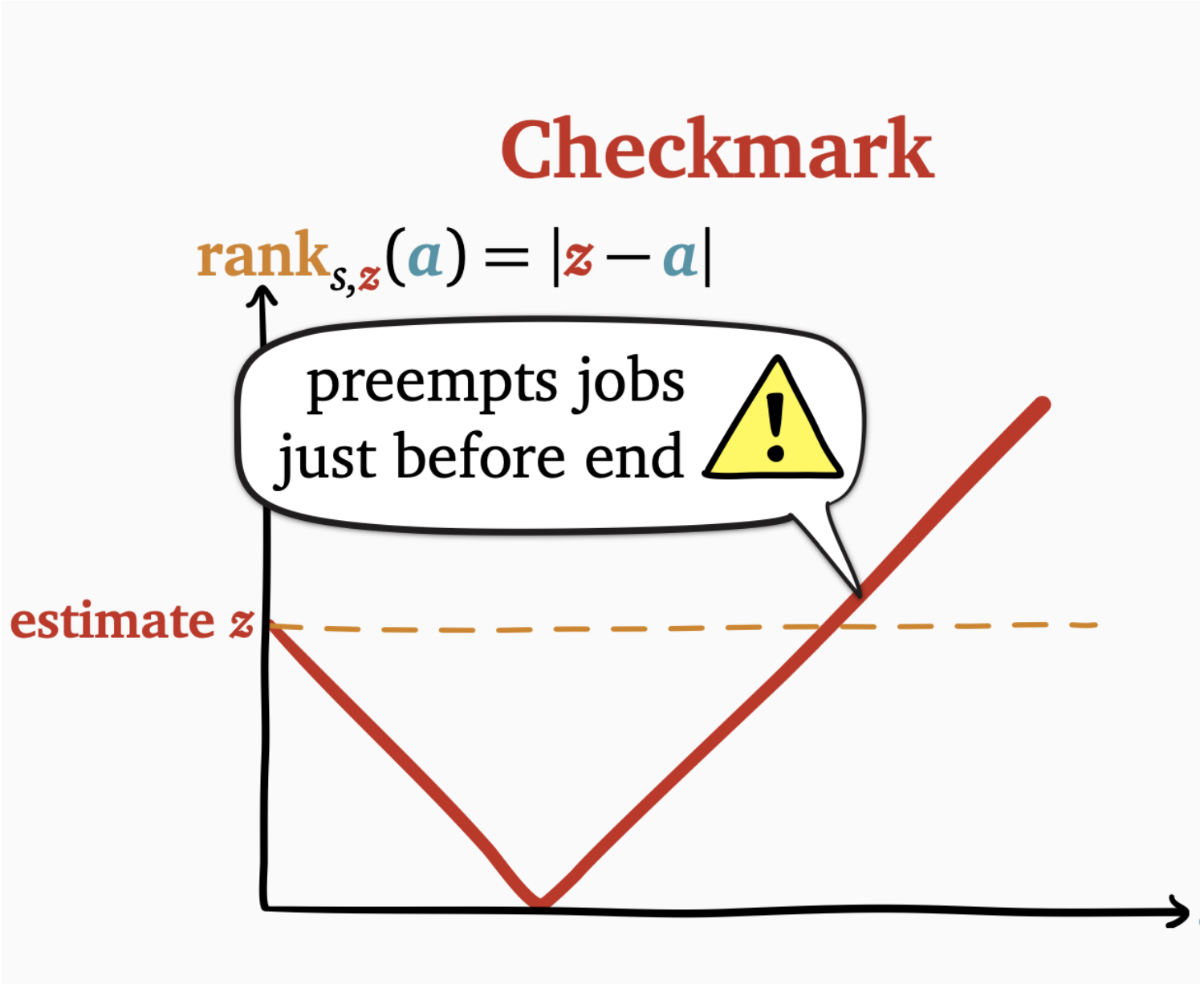
Scheduling by Rank



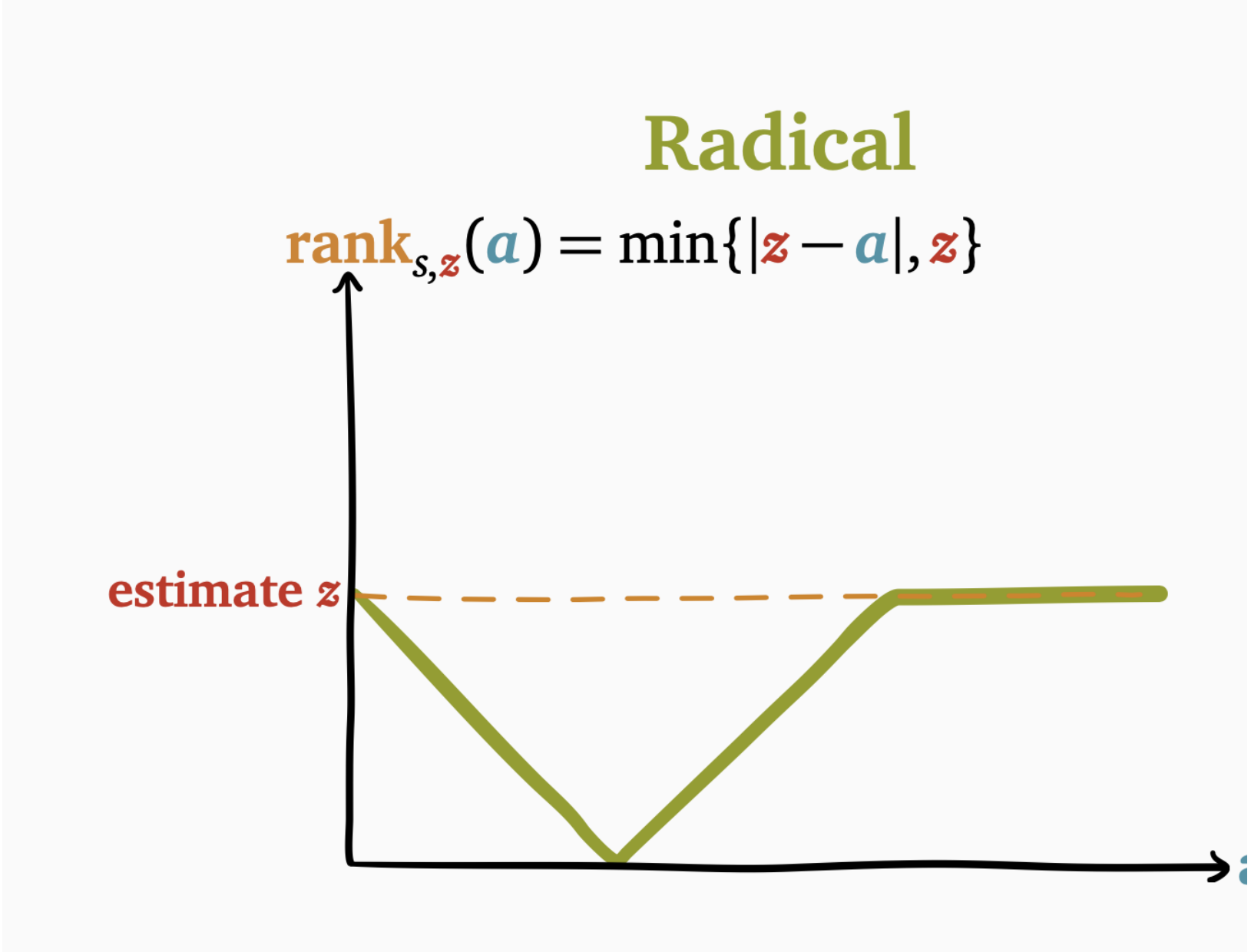
Scheduling by Rank



Scheduling by Rank



Scheduling by Rank



Related Work

- Azar, Leonardi, Touitou consider the online setting (not queueing theoretic).
  - Job size predictions are within a factor μ of actual size.
  - Call their algorithms distortion-oblivious (μ does not need to be known in advance).
  - Get O(μ log μ) competitive-ratio where μ is the maximum distortion.

Open Questions

- Better policies/bounds for graceful degradation and consistency, in this setting?
- Weaker restrictions on the predictions for such results?
- Bounds for other desirable criteria, such as tail behaviors?
- Could there be advantages to going beyond rank-based scheduling?

Multiple Servers

- Very few results for using predictions in multiple server settings!
- Mostly wide open with open questions.
  - See survey for more discussion.
- One analyzable system: Power of 2 choices with 1-bit predictions.
  - Power of 2 choices: use fluid limit analysis to analyze limiting "infinite server" system where each job picks shortest of 2 (or d) queues.
  - With 1-bit predictions, state space is "small" enough can write limiting differential equations (and calculate them) in some cases.

Open Questions

- Analyze power of 2 choices more generally with predictions?
- Most any multiple server question with predictions remains open….

Predictions, with Costs

- Algorithms with predictions area is still “new”.
- In theoretical works, common that predictions come for free
  - No discussion of costs of prediction
- For scheduling, this seems unnatural.
  - *Predicting a job’s service time may itself take some server time*…
  - Interesting extreme case: introducing predictions could overload a heavily loaded system.
    - Predictions may not be helpful, and actively harmful.

Prediction Cost Models

- External Cost Model
  - Predictions provided from some external source
    - Does not affect service times
    - But has some cost, e.g., fixed cost per prediction
      - Some jobs may not use a prediction
    - Expected cost per job = response time + prediction cost
- Server Time Cost Model
  - Predictions require some amount of service time, e.g., fixed time
  - Predictions must also be scheduled
    - Expected cost per job = response time

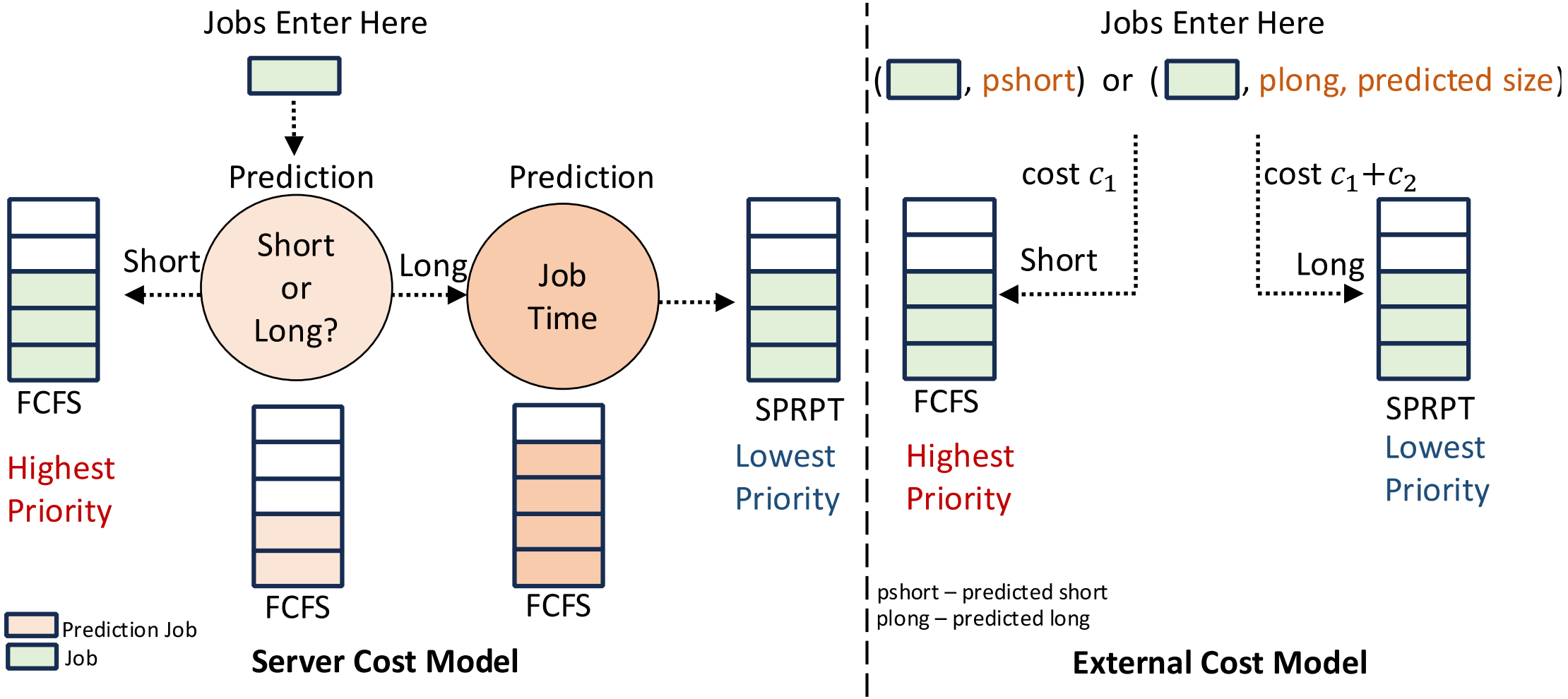
New Scheduling Strategies with Costs

- Can re-analyze M/G/1 scheduling schemes like SPRPT, 1-bit predictions with costs
- But also consider new settings
- Suppose predictions have different costs
  - 1-bit predictions are cheap
  - Service time predictions are more expensive
- SkipPredict: 1-bit predictions for all jobs, service time predictions only for predicted long jobs.

SkipPredict, Server Time Model

- A job arrives at a server….
  - It needs a 1-bit prediction (short or long), joins 1-bit prediction queue.
    - 2nd highest priority.
- If predicted short, it joins a queue for short jobs.
  - These jobs have highest priority.
  - Served FIFO.
- If predicted long, joins a queue for a second prediction.
  - Where its service time is predicted.
- Predicted long jobs served by SPRPT.
  - Lowest priority.

SkipPredict

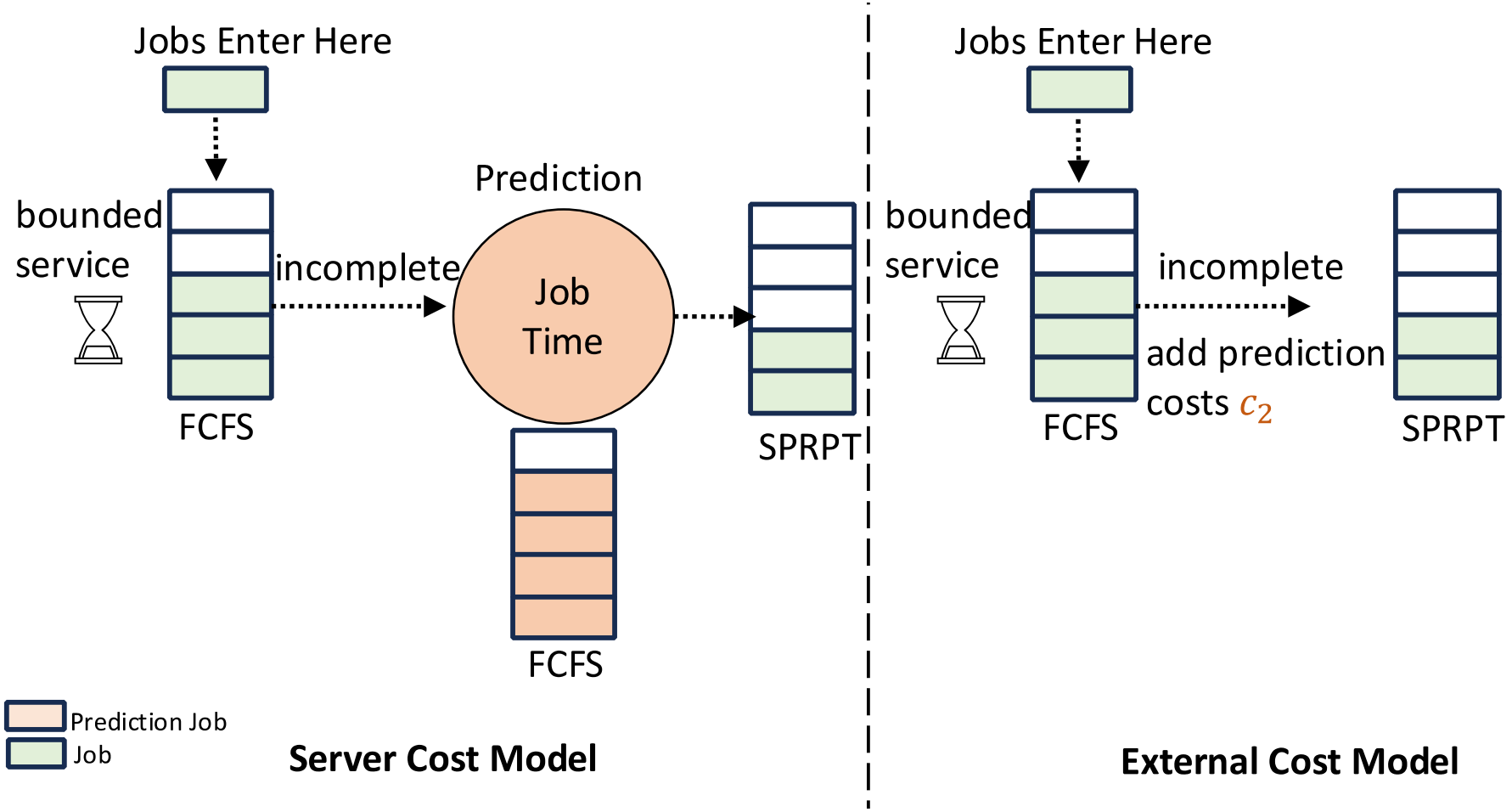


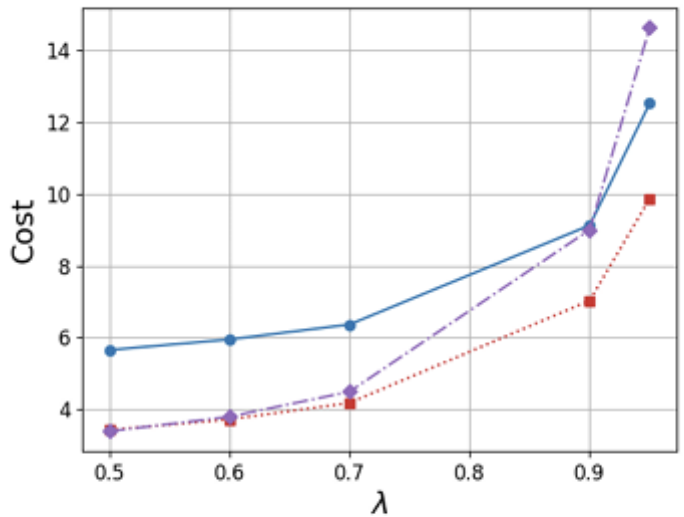
DelayPredict

- Short predictions for everyone may get expensive.
  - Suppose more expensive predictions are not much more expensive!
- So what if we do not use short predictions.
- Use predictions after a job has run a long time.
- Instead of 1-bit predictions, run all jobs for time *L* (FIFO) under high priority, then move to low priority, predict service time, and used predicted service time for SPRPT.

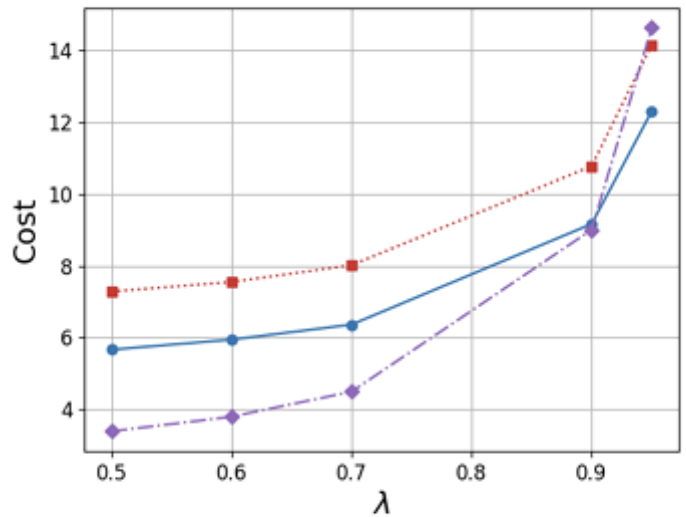
DelayPredict

Instead of 1-bit predictions, run all jobs for time L, then predict if still going

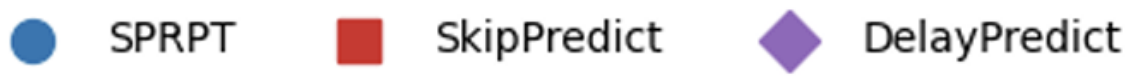




(a) large cost gap, $c_1 = 0.5, c_2 = 4$



(b) small cost gap, $c_1 = 3.5, c_2 = 4$



Cost Analysis

- SkipPredict and DelayPredict are SOAP-analyzable schemes
  - Priority is determined by job's type (service time and prediction outcomes) and time in service.
- Best scheme may depend on costs, load
  - But both SkipPredict and DelayPredict can be better than SPRPT when prediction costs are included.

Open questions

- Other models of costs for predictions?
- Methods to dynamically choose when to pay the cost of prediction?
  - Based on current load?
  - “Gittins”-style policies, choose when to predict?
  - Features that determine whether to predict in real settings?
- Dedicated prediction servers, vs. prediction+jobs on same server?
- Other economic models for predictions?

Conclusion of Part 1….

- The theory of scheduling with predictions is historically understudied, and many interesting (and basic?) open problems seem available.
- Recent progress in methods (SOAP) is helpful; hopefully more analysis tools can be added.
- Predictions seem quite powerful and potentially very useful in practice.
  - So we'll look at a very timely area where there seem to be a lot of practical papers – LLM scheduling.
  - Question: Can queueing theory help?
  - Come back for Part 2….

Overview

Part 1

Queueing and Predictions

Cost of predictions

Open questions

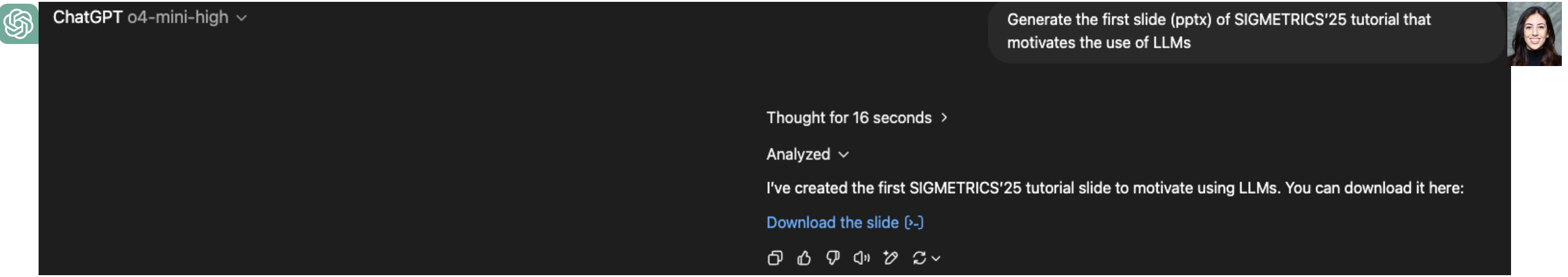
Part 2

LLM systems as an application of queueing with predictions

Three settings of LLM systems

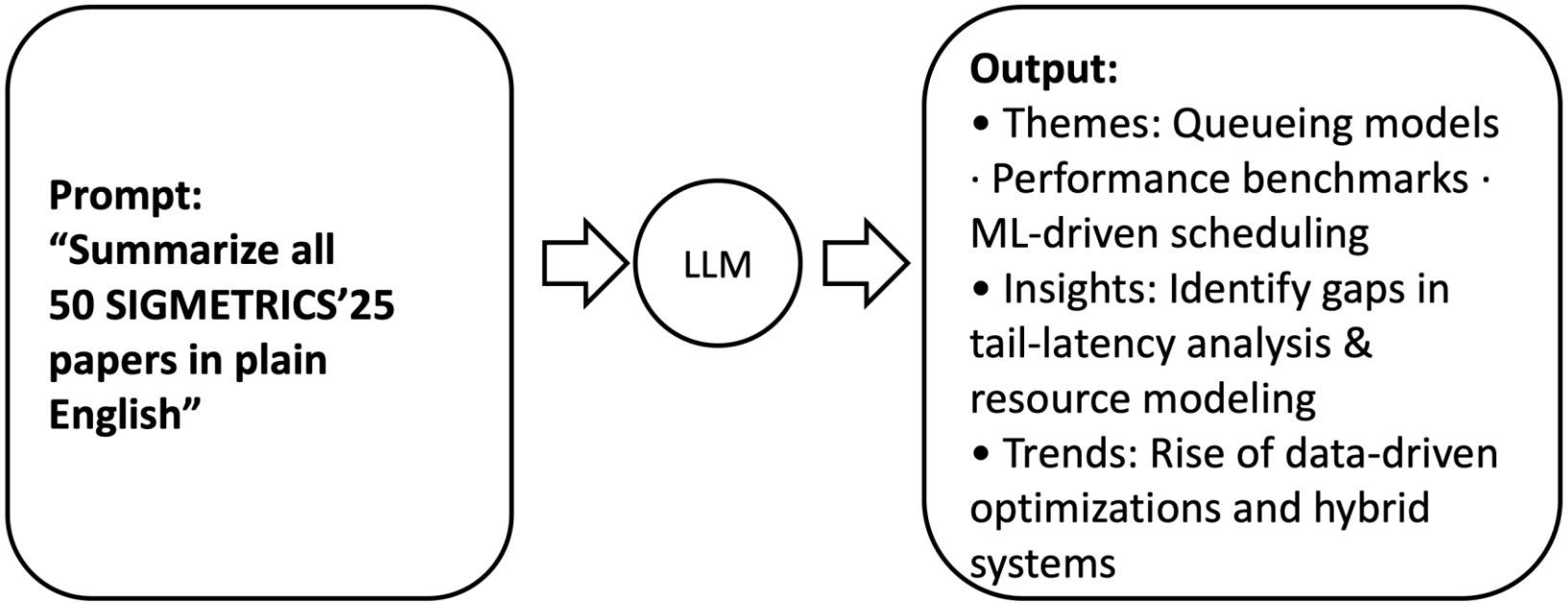
Open questions

Large Language Models (LLMs) are everywhere



The generated slide*:

Motivation – Why Bring LLMs into SIGMETRICS?



* This is the only slide generated by LLM in this talk

Many Users

- ChatGPT has over 400 million weekly active users.
- The platform also sees over 1 billion messages sent per day.
- There are many AI LLMs

Survey: 52% of U.S. adults now use AI large language models like ChatGPT

By Elon University News Bureau, staff | March 12, 2025

LLMs are slow

and expensive..

- LLMs run on high-end GPUs such as NVIDIA A100
- Each GPU can only serve a handful of requests per second

Users are not patient

"Users dropped off before the response arrived."
In interactive apps, every 100ms matters - especially on mobile

"Why is my LLM bill so high?"
Longer response times = long GPU time = more $$$

"My AI assistant takes 17 seconds to respond.."

Scheduling for latency optimization

- Choose which requests to serve when resources are constrained.
- Order pending work (e.g. SRPT, priority queues, FIFO) to minimize wait.
- Proven to cut average and tail latency in datacenters, OS and cloud services.
- LLM systems add new twists, with challenges that differ from traditional queueing systems.
- Goal of this part: present these challenges and where new research opportunities lie.

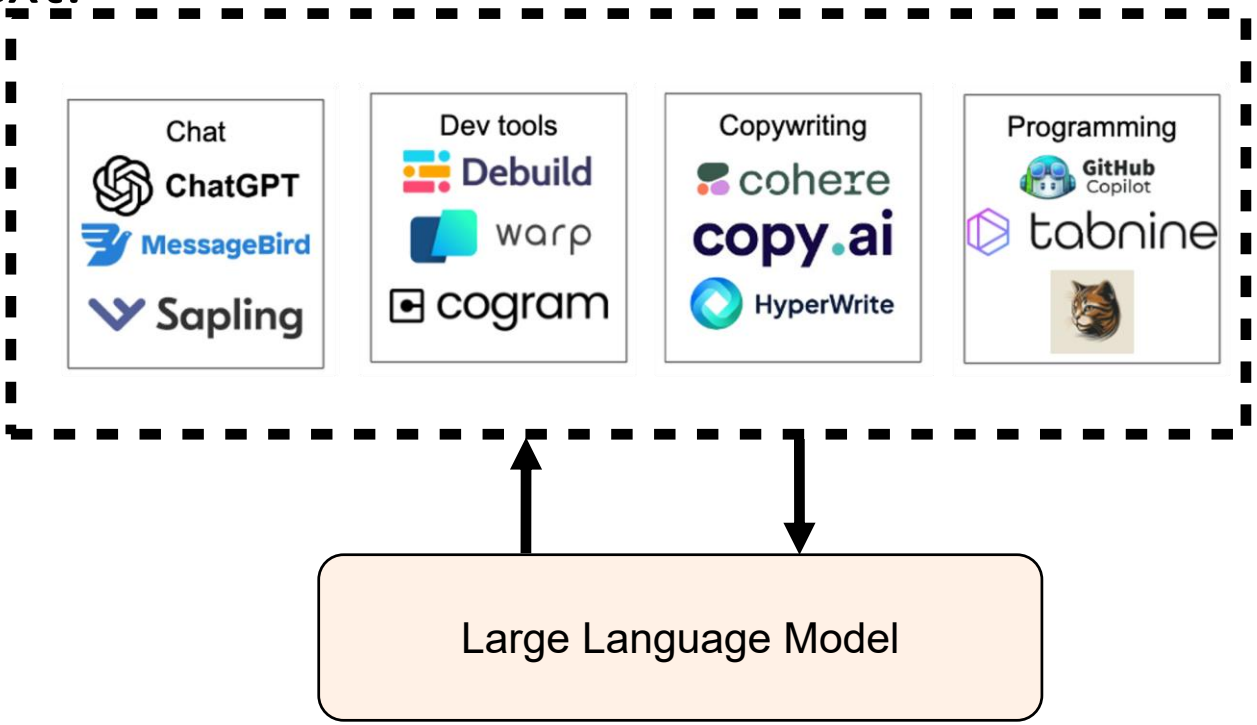
Agenda

- What are LLMs and how do LLM systems work?
- Three settings, challenges, and open questions:
  - Scheduling for LLM serving
  - Scheduling in compound AI systems
  - Scheduling for LLM reasoning tasks

What are LLMs and how do LLM systems work?

Large Language Model (LLM)

- A type of deep neural network designed to generate human-like text.
- Trained on large amounts of data to capture language patterns, grammar, and even context.



Three Phases in the Life of an LLM

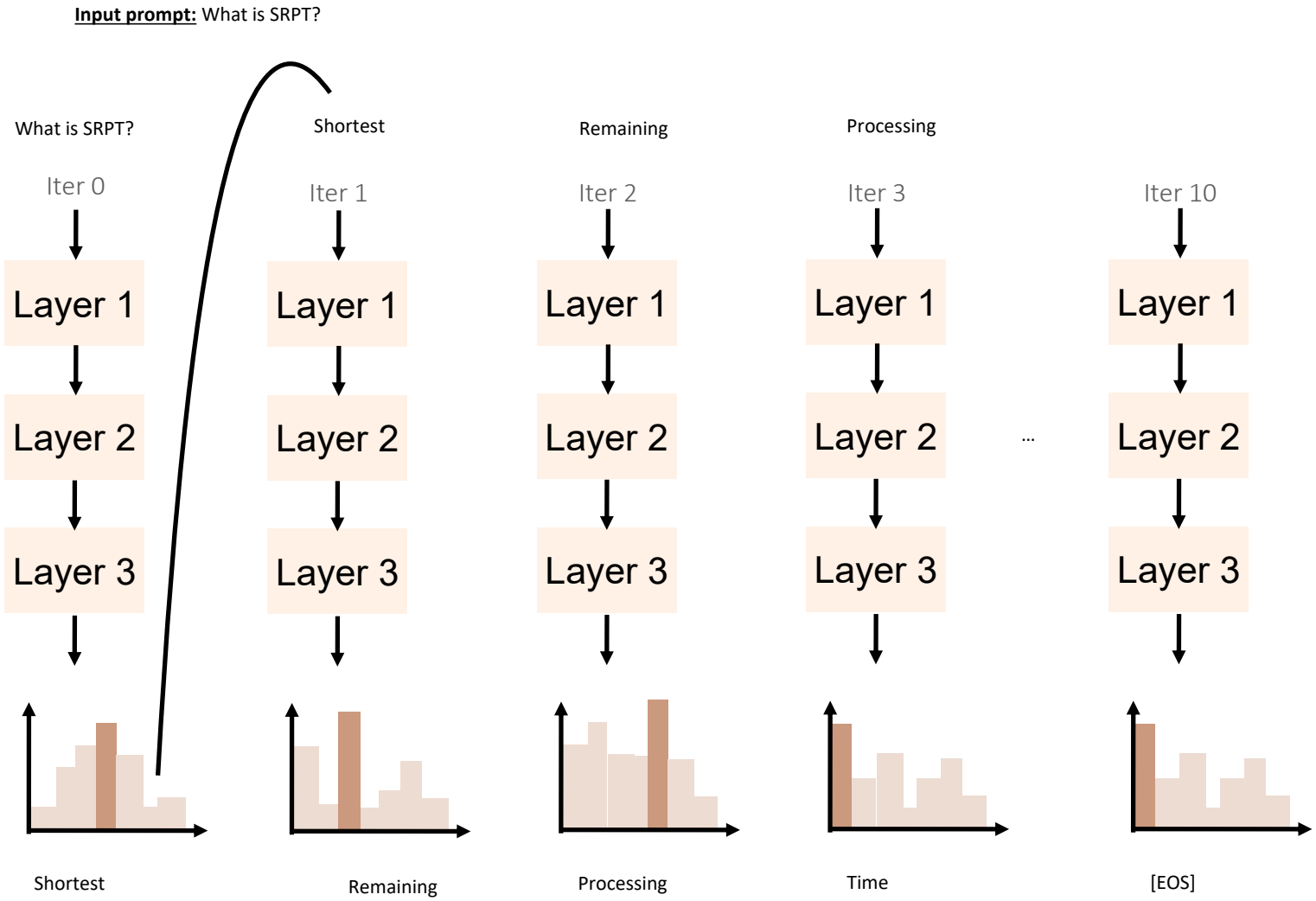
- **Training**: offline, resource-intensive where models learn from large datasets.
- **Post-training**: Improves the model's responses using additional techniques like fine-tuning and feedback.
- **Inference**: usually online, where pre-trained models respond to user requests.

Today's focus: Efficiently serving LLM requests during inference.

Examples: ChatGPT.

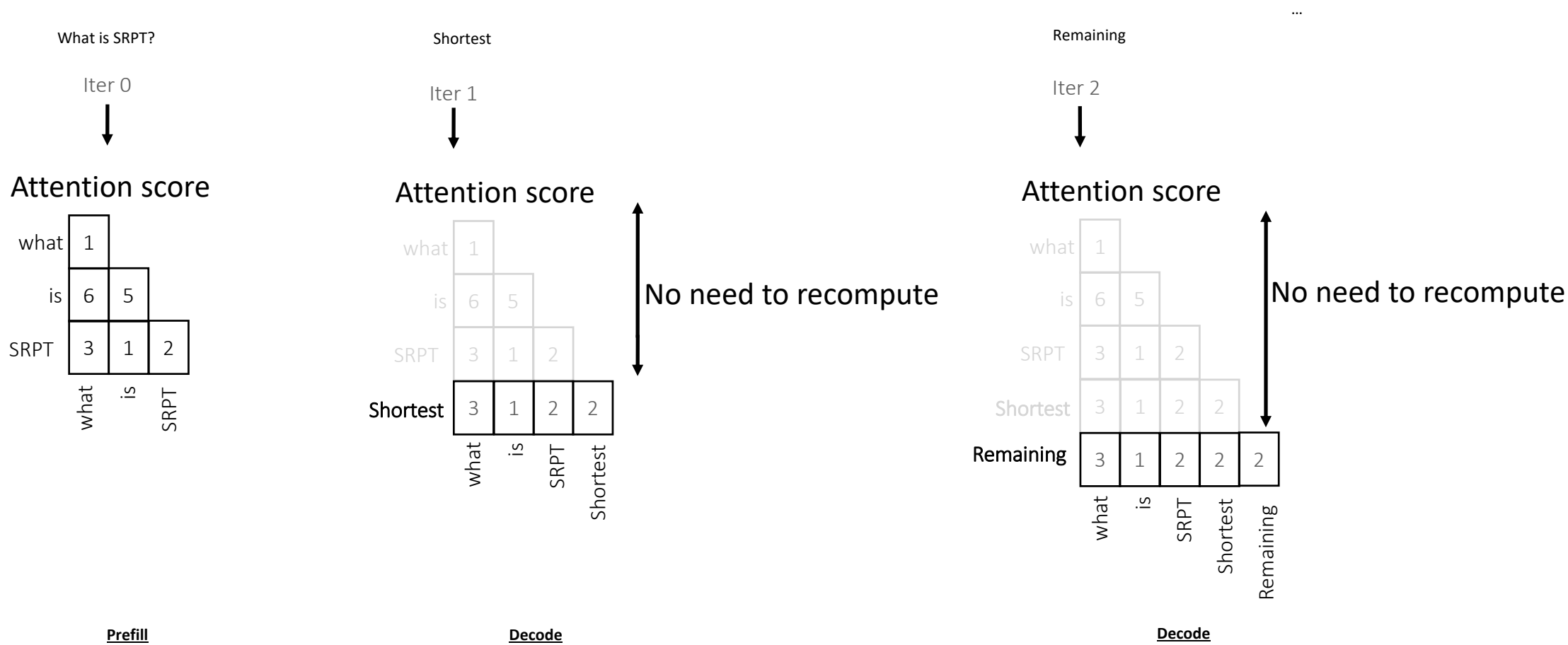
How Inference in LLM Works

Autoregressive Decoding



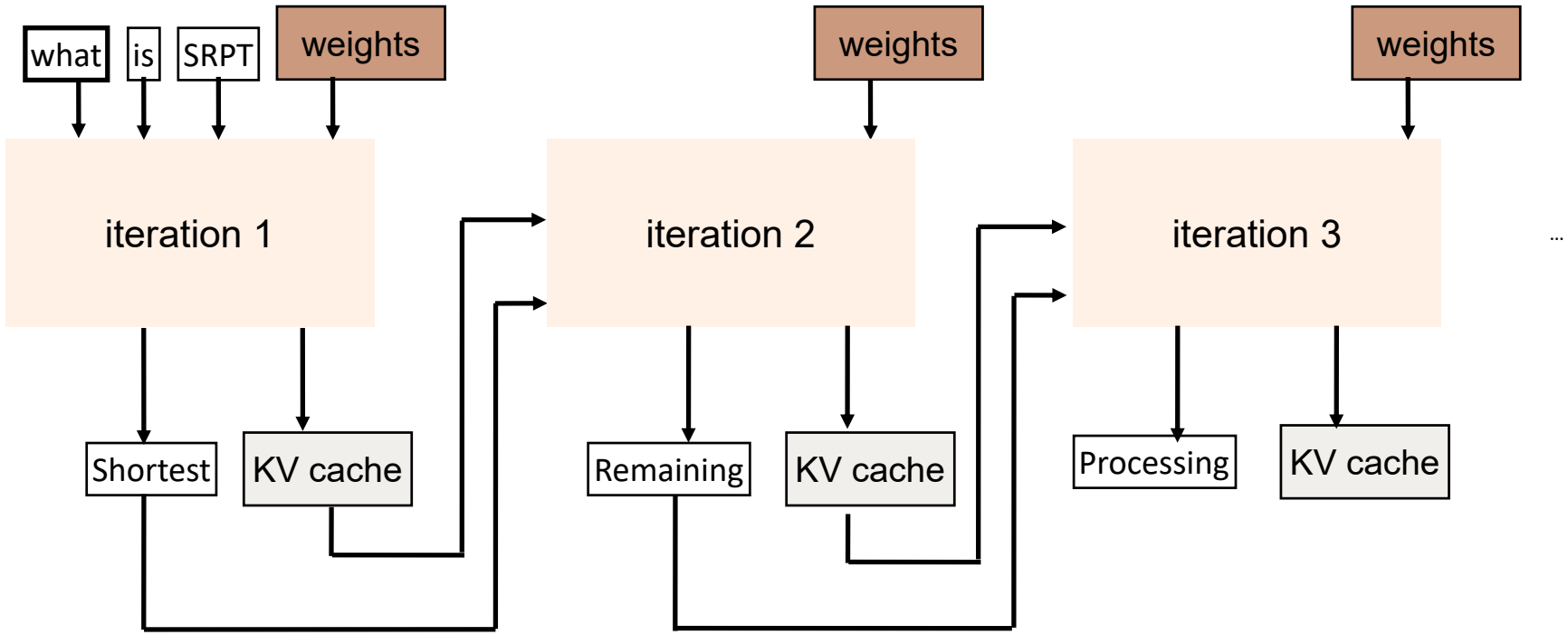
How Inference in LLM Works

Autoregressive Decoding



Key-Value (KV) Cache: What Get Stored

- Save attention (keys and values) for the following iterations to avoid re-computation.
- Grows with the number of tokens generated.
- Stored in GPU memory, can consume gigabytes per request.
- Memory is limited (40G/80G), storing KV cache for many requests becomes a bottleneck.


what
is
SRPT
weights
weights
weights
iteration 1
iteration 2
iteration 3
...
Shortest
KV cache
Remaining
KV cache
Processing
KV cache

How Big is the KV Cache

- For a transformer model, the KV cache memory size **per request** is approximately:

$$Memory \approx 2 \times L \times H \times S \times T \times dtype\_size$$

#layer #heads head size #tokens

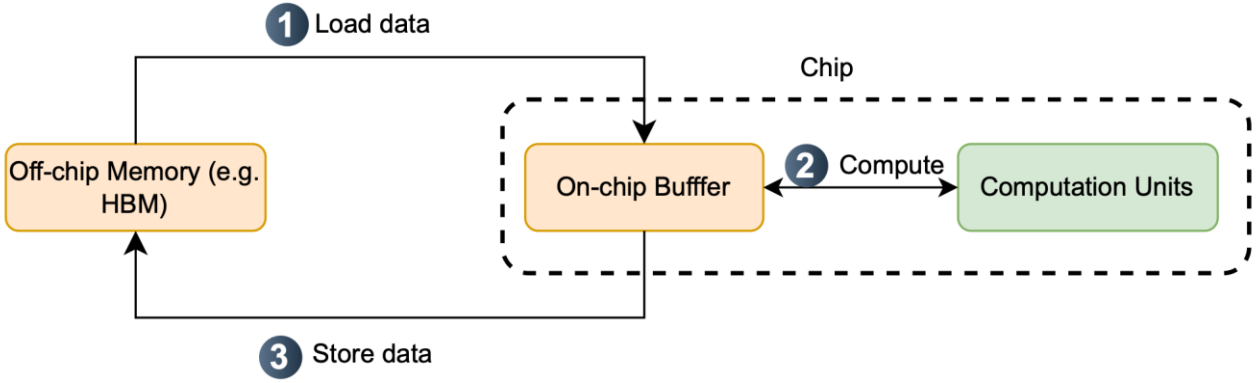
- **Example: f**or LLaMA-13B, FP16, sequence length 2,048:

$$Memory \approx 2 \times 32 \times 40 \times 128 \times 2048 \times 2\,bytes \approx 3.2\,GB$$

Execution on Hardware

1. Data movement: model weights and KV cache are transferred from GPU memory (e.g., HBM) to faster on-chip buffers (e.g., SRAM).
2. Computation**:** Each token is processed by matrix multiplications in the transformer layers using on-chip processing units.

3. Write back: The selected token and updated KV entries are written back to GPU memory.



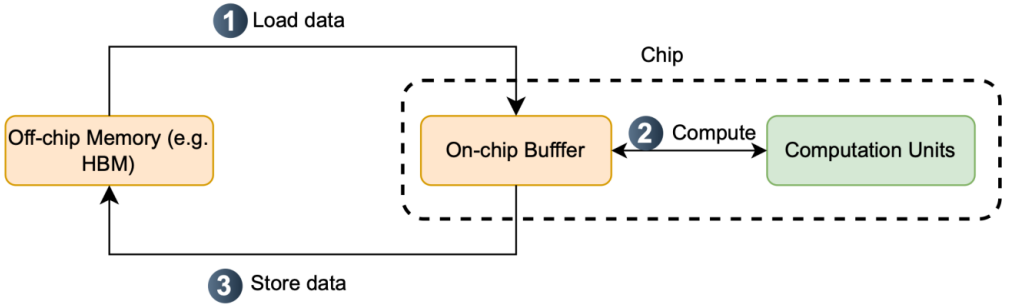
Memory-bound vs. Compute-bound in Inference

- Each token generation involves:
  Loading model weights and KV cache from HBM
  Computing next token using transformer layers
  Storing updated KV cache back to memory
- When token generation is slow due to step (2) → **Compute-bound**
- When step (1) or (3) becomes the bottleneck → **Memory-bandwidth-bound**

Question

Is the prefill phase compute-bound or memory-bound?
Is the decode phase compute-bound or memory-bound?

Answer



- Prefill phase (processes the input prompt):
  All tokens are processed in parallel through all layers
  High compute utilization → typically **compute-bound**
- Decode phase (generates one token at a time):
  One token is generated per step, requiring KV cache reads/writes
  Low compute utilization, dominated by memory access → **memory-bandwidth-bound**

Open Question

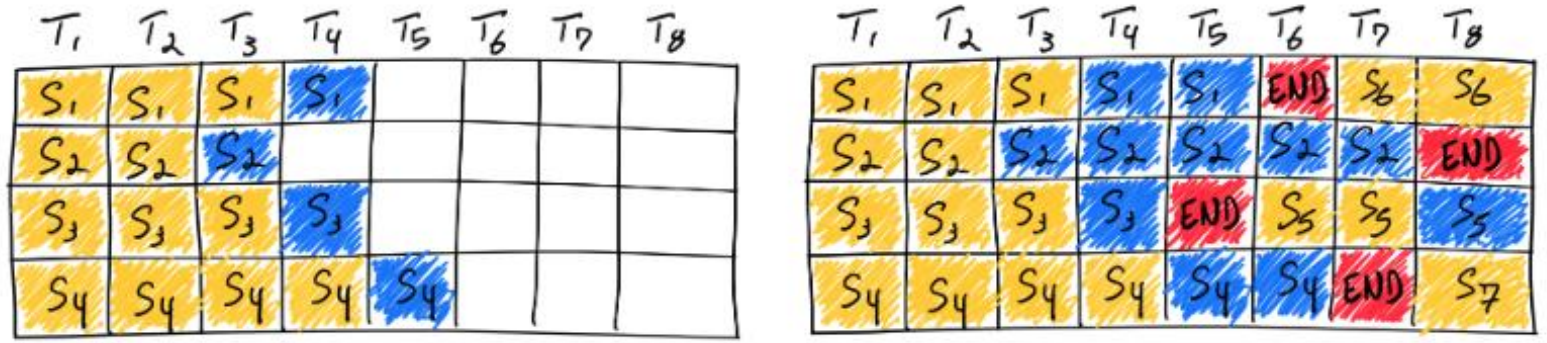
- How should we rank requests when prefill and decode phases differ in size?
- Example: two requests with the same total size
  - One: short prefill, long decode
  - Other: long prefill, short decode
- The “better” request may depend on hardware characteristics (compute vs. memory limits)

Batching

- Batching multiple requests helps to reduce the bottleneck by:
  - maximizes compute utilization by processing multiple prompts simultaneously.
  - loading model weights once and reusing them for multiple inputs.
- This improves **compute utilization** and **amortizes the cost** of loading model weights

Continuous Batching

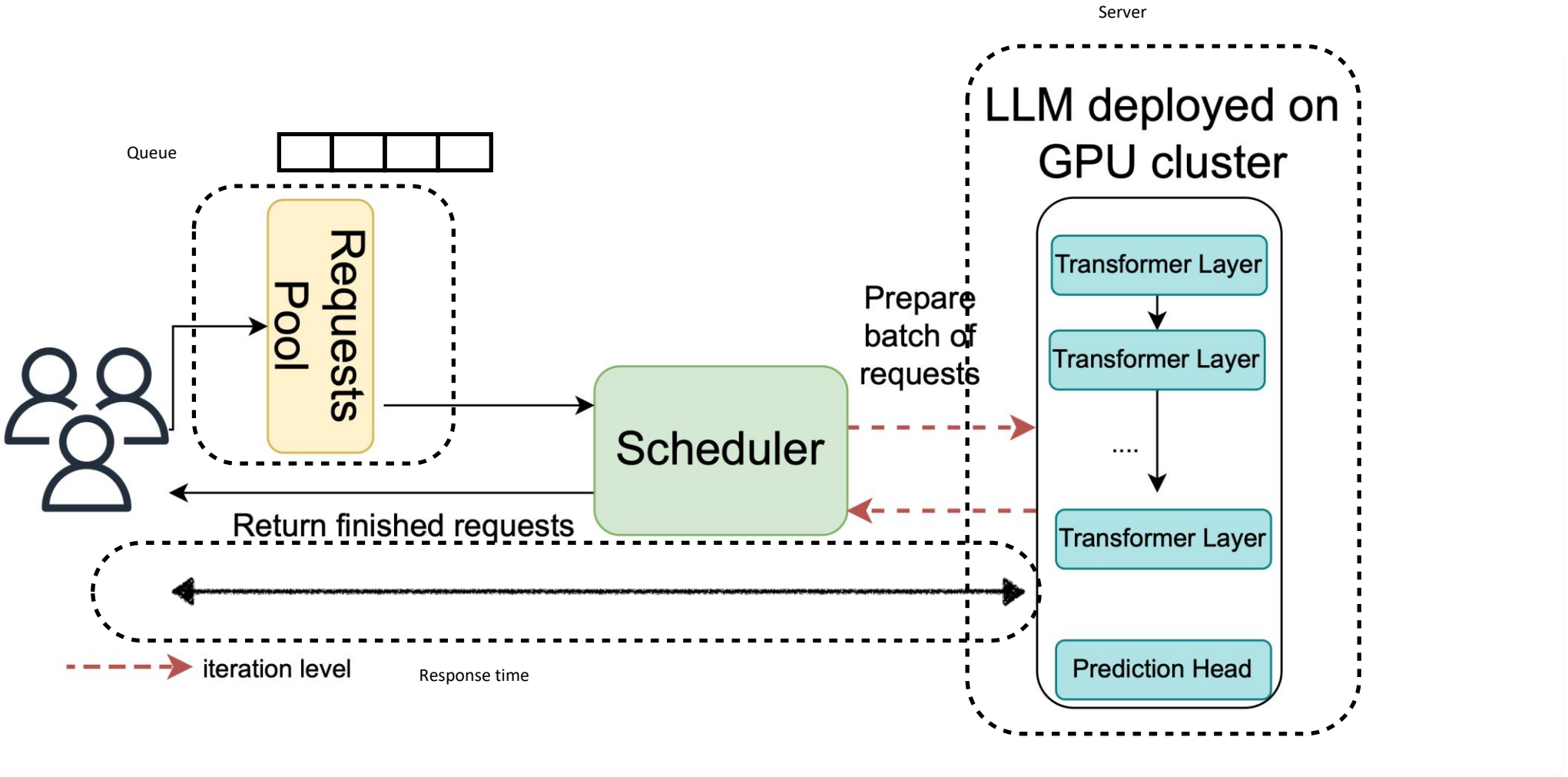
- Continuous batching dynamically adds new requests to the batch as slots free up.
- Modern systems (e.g., vLLM) implement continuous batching with token-level granularity



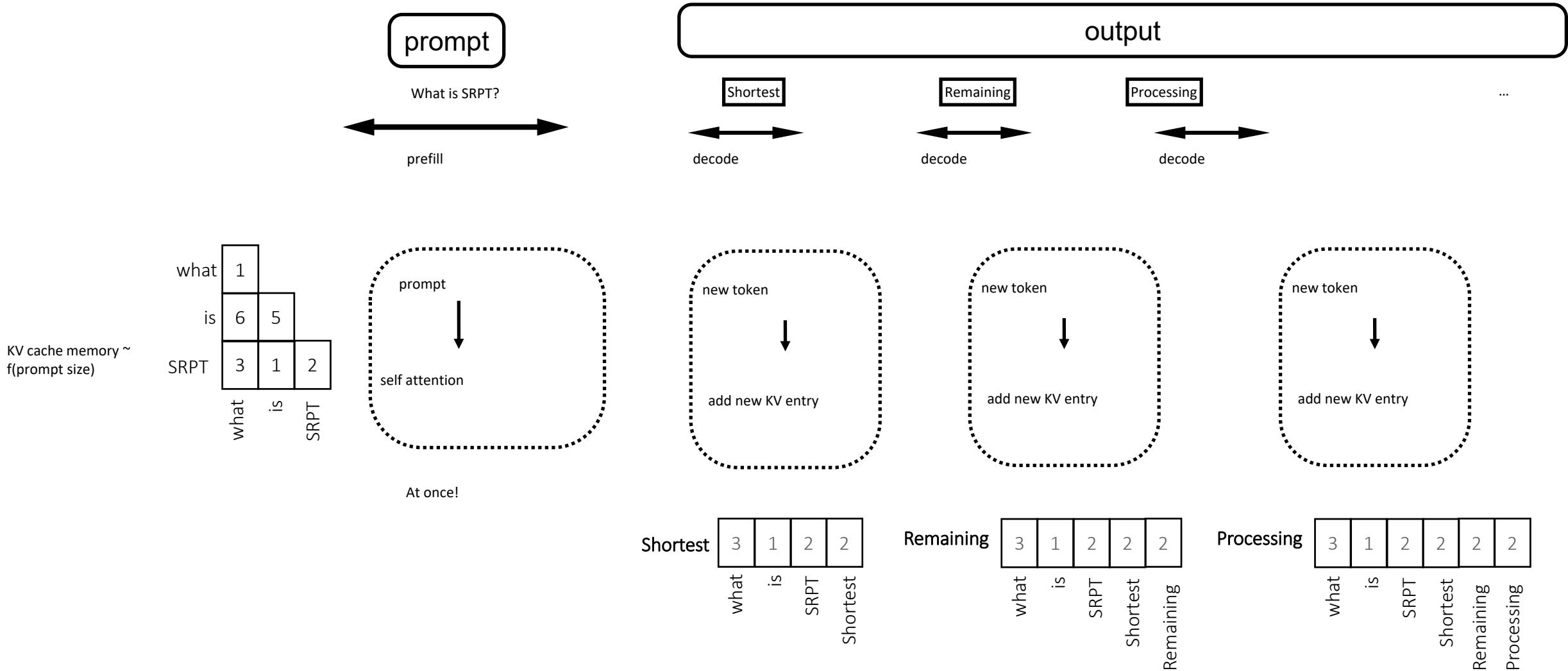
Completing seven sequences using continuous batching. Left shows the batch after a single iteration, right shows the batch after several iterations. Once a sequence emits an end-of-sequence token, we insert a new sequence in its place (i.e. sequences S5, S6, and S7). This achieves higher GPU utilization since the GPU does not wait for all sequences to complete before starting a new one.

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

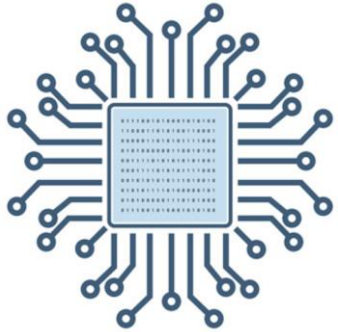
System Perspective



The Job Perspective

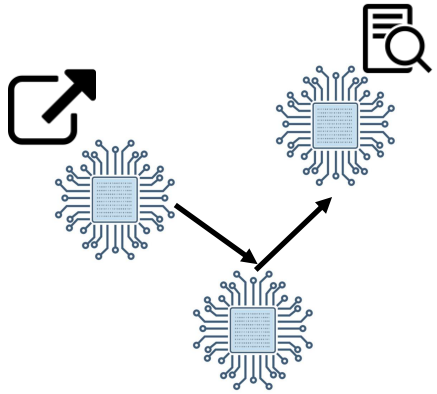


Three LLM System Settings



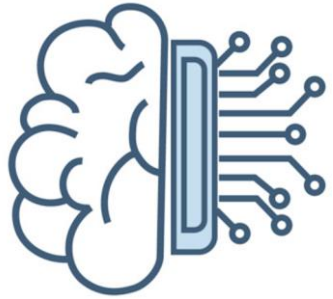
Single LLM Systems

A single LLM deployment on one or more GPUs.



Compound AI Systems

Integrate multiple components (e.g., external APIs, retrieval systems).



LLM Reasoning Systems

Include multi-stage reasoning (e.g., chain-of-thought) for complex responses.

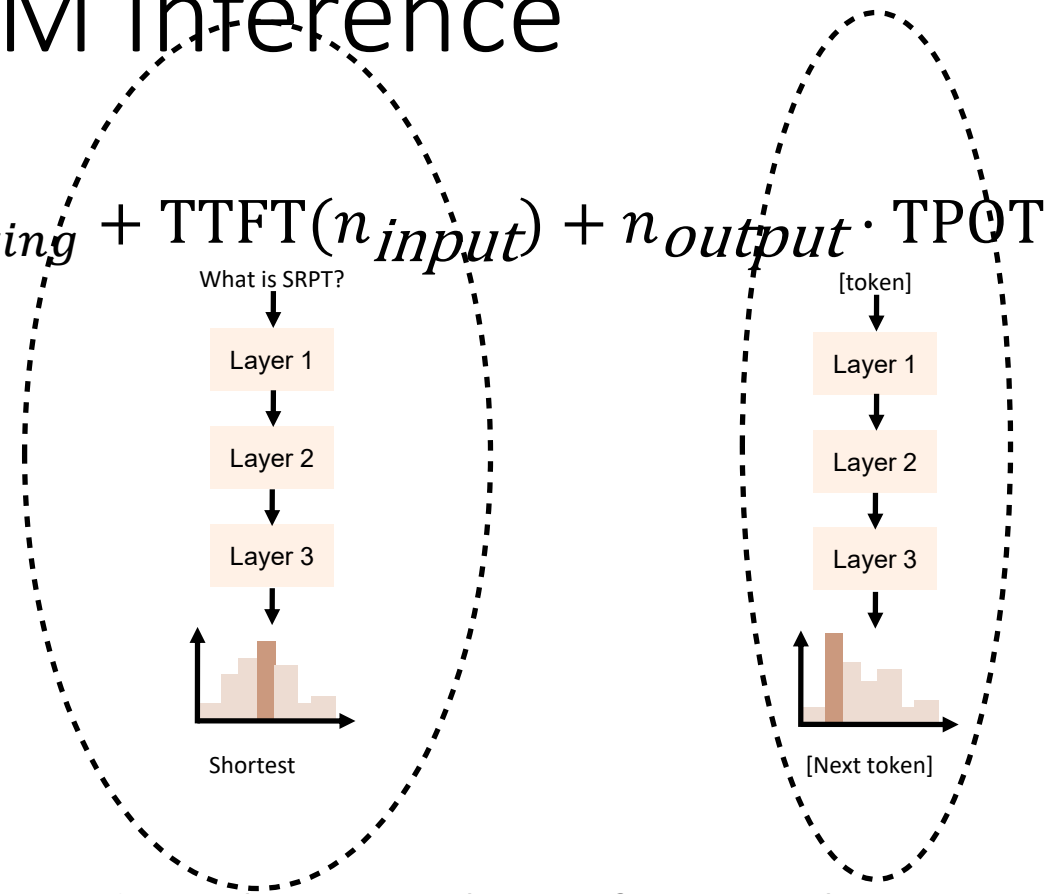
Key Metrics in LLM Systems

- Compute Cost
  Depends on model size (parameter count) and response length (due to autoregressive decoding)
- Latency: Affected by model size and output length
  - Time-to-first-token
  - End-to-end response time
- Throughput
  Tokens generated per second
- Accuracy
  Larger models often yield better responses, but accuracy gains vary by task
- Energy Usage
  Total energy consumed (kWh) and associated carbon emissions

Single LLM Systems

Latency in LLM Inference

$$t_{waiting} + \mathrm{TTFT}(n_{input}) + n_{output} \cdot \mathrm{TPOT}$$



- Smaller models can reduce latency, but often at the cost of accuracy.
- Our approach focuses on system-level optimizations to minimize latency without compromising model accuracy.

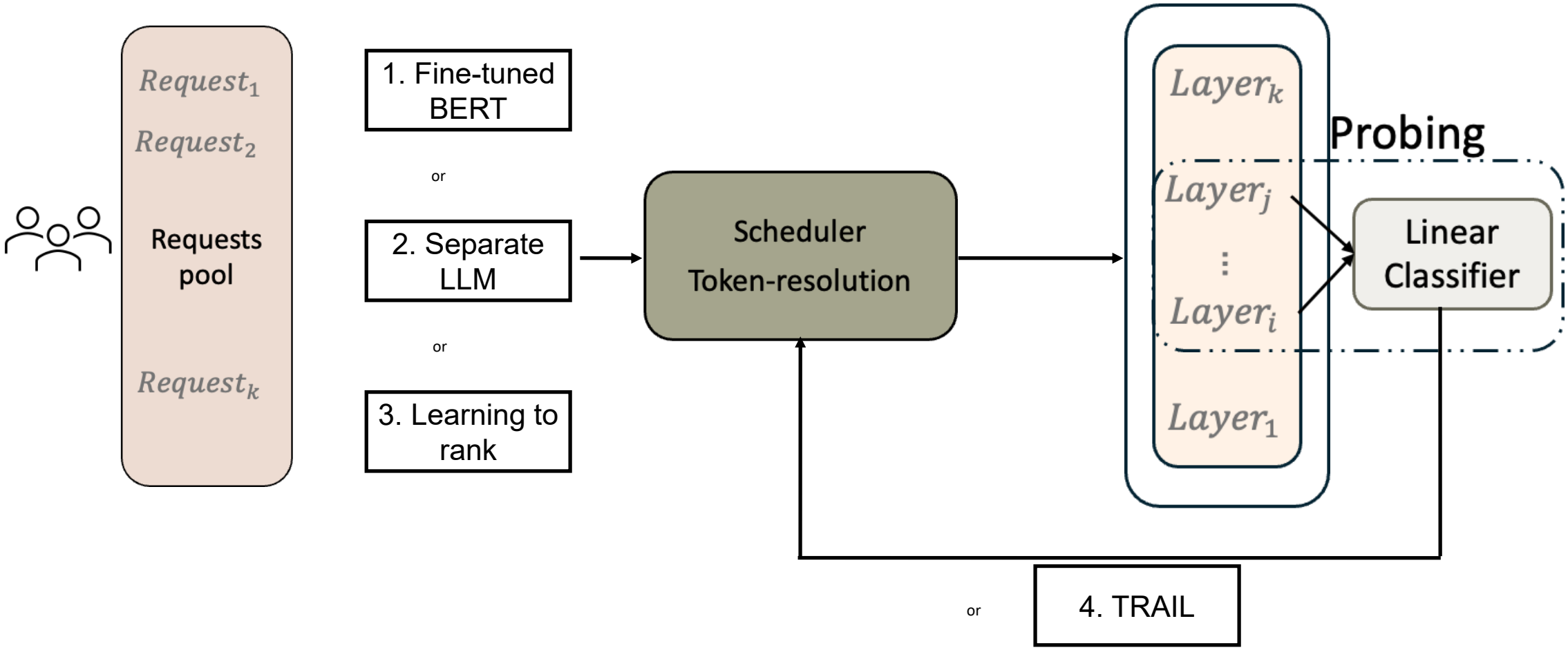
Size Prediction Challenge

- Size prediction is critical for size-based scheduling
- Output length are challenge to predict due to autoregressive decoding

Predicting request output length

Existing approaches

Method	Comments
1. Fine-tuned BERT (S^3, Jin et al.)	Struggles with high-variance outputs
2. Separate LLM (Zheng et al.)	Adds compute and resource overhead
3. Learning to Rank (LTR, Fu et al.)	Ignores prompt size (prefill cost)
4. TRAIL (Shahout et al.)	Requires refinement during decoding

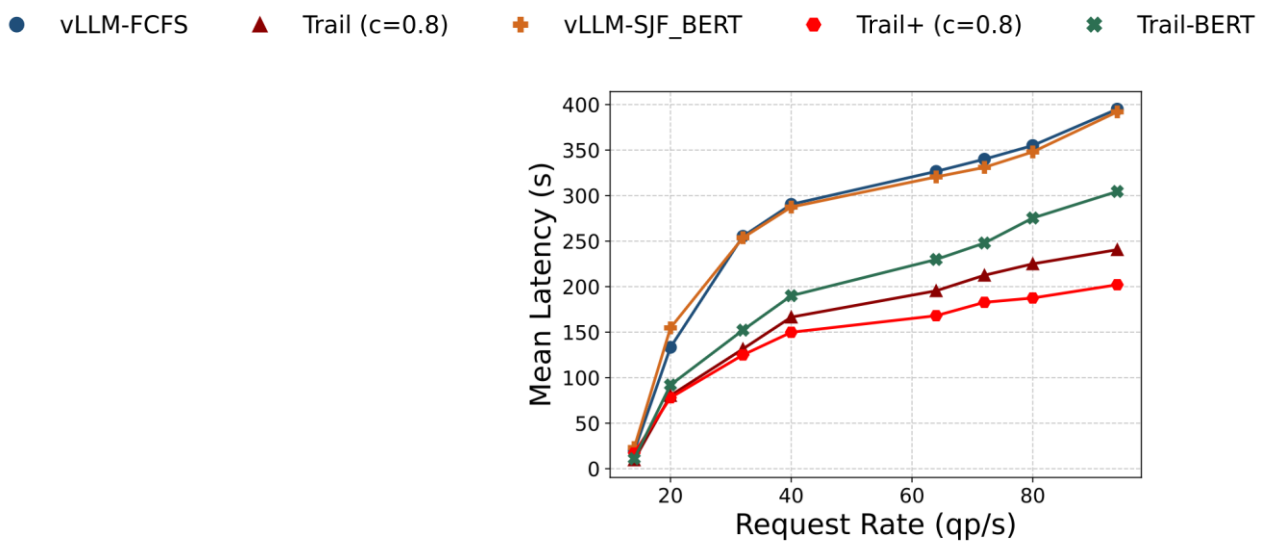

$Request_1$
$Request_2$
Requests pool
$Request_k$
1. Fine-tuned BERT
or
2. Separate LLM
or
3. Learning to rank
Scheduler
Token-resolution
$Layer_k$
Probing
$Layer_j$
$Layer_i$
$Layer_1$
Linear Classifier
or
4. TRAIL

Scheduling Benefits

Table 4: Performance of sequence scheduling with different response length perception module.

	Throughput (samples/s) ↑	Improvement ↑	Tokens/batch ↓
Vanilla	1.22		377
Ground Truth Preditor	2.52	+107%	201
Pooling + MLP	1.96	+61%	216
[LEN]-token Fine-tune	2.10	+72%	210
Perception Only*	1.40	+15%	328
Instruction Tunning (mean)	1.77	+45%	211
Instruction Tunning (max)	**2.27**	**+86%**	**208**

Separate LLM



Trail

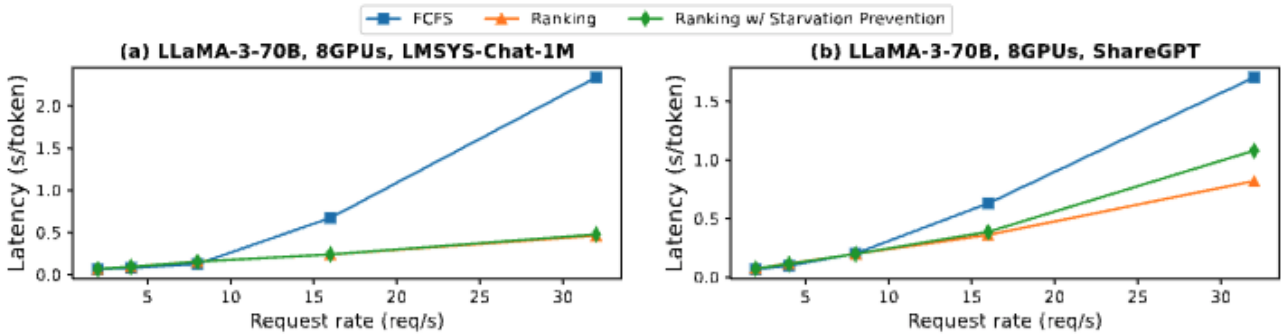
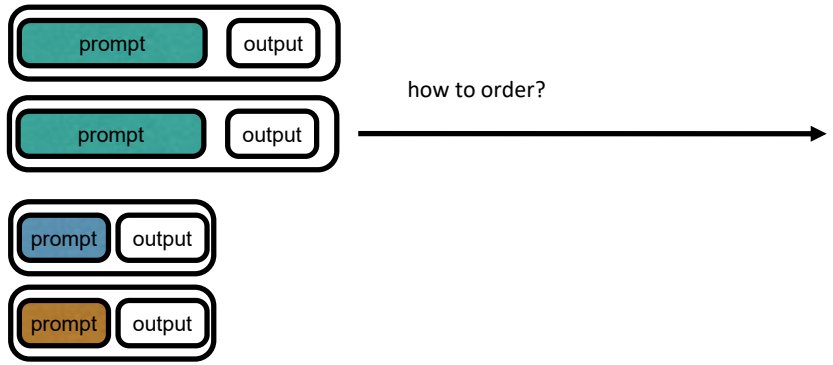


Figure 5: Influence of starvation prevention on latency

Learning to Rank

Open Question ? : Prompt Sharing in Scheduling

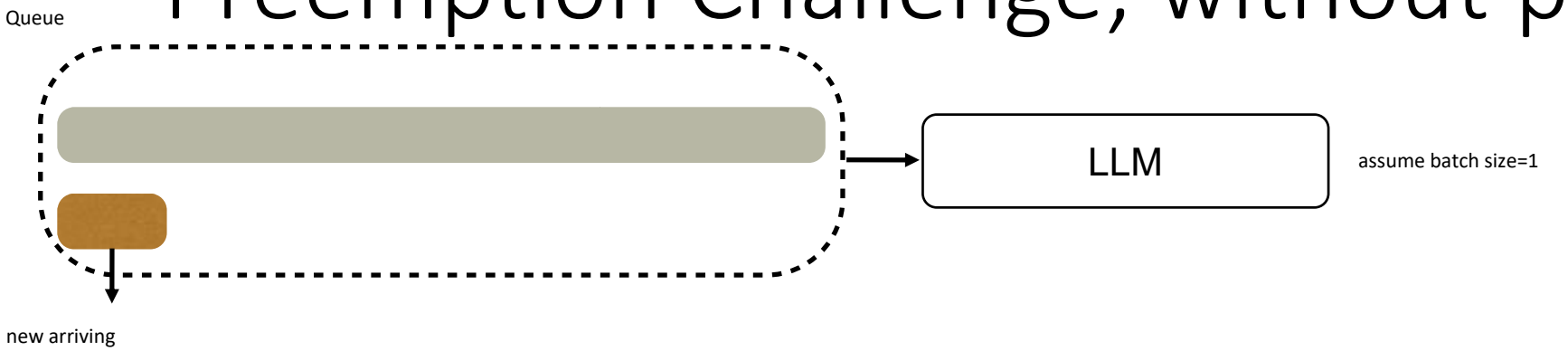
- Prompt sharing: when multiple requests contain overlapping input segments, enabling the system to reuse KV computations.
- How should prompt sharing be incorporated into scheduling decisions? In scenarios where requests share similar prompts, what strategies can be employed to batch such requests effectively while avoiding delays for standalone small requests?

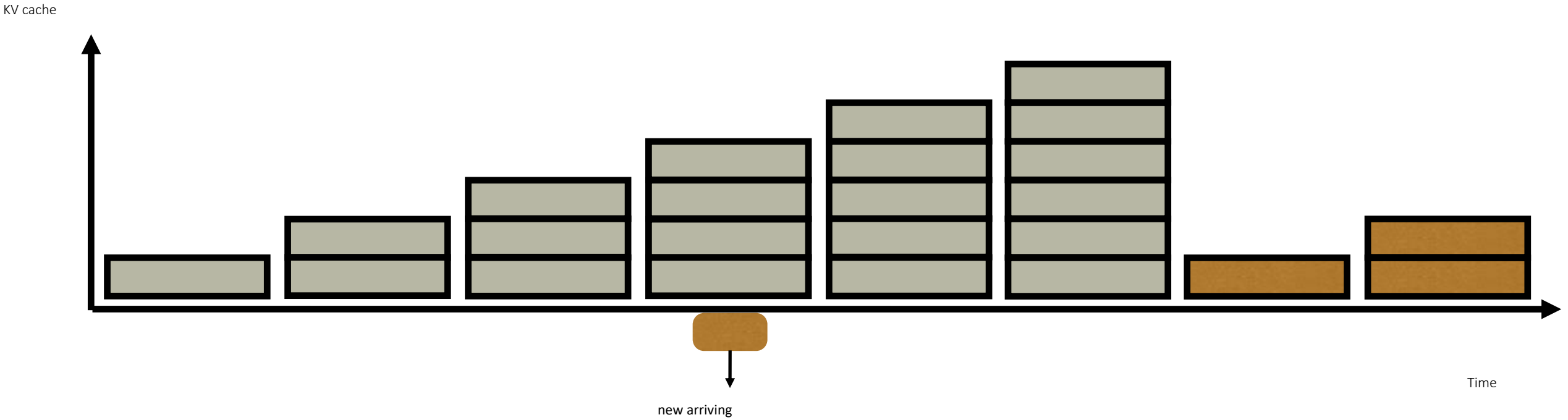


Question

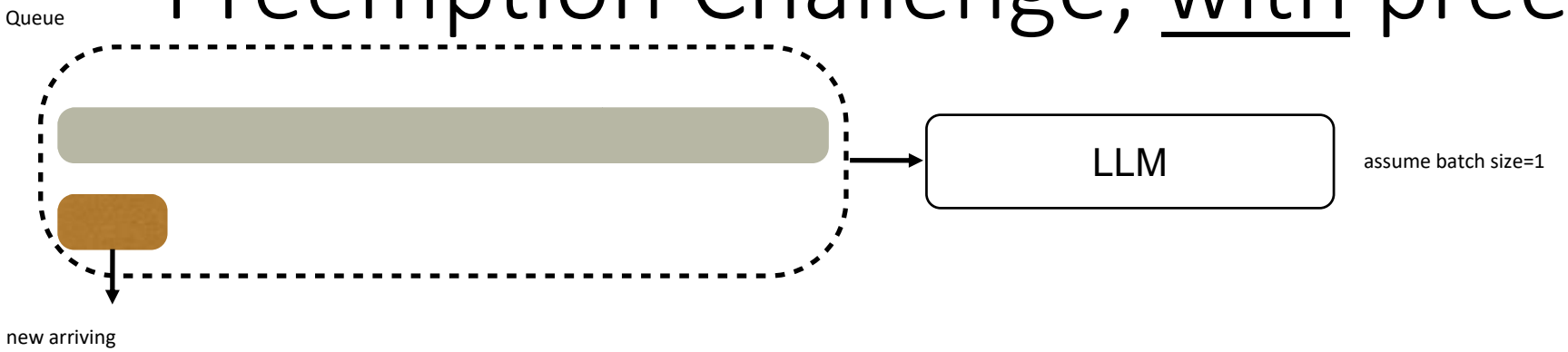
- In preemptive scheduling (like SRPT), a running job can be interrupted (paused or removed) to allow a higher-priority (shorter) job to run.
- Do you think preemptive scheduling is straightforward to implement in LLM inference systems?

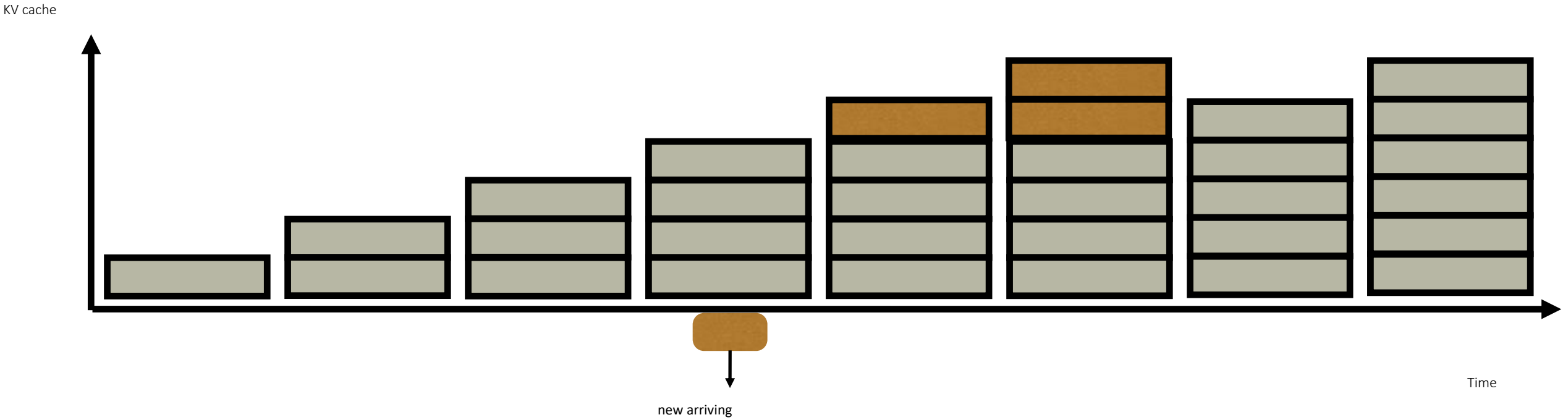
Preemption Challenge; without preemption





Preemption Challenge; with preemption





in LLM systems, preemption introduces KV memory overhead

- In LLMs, pausing a request is costly because KV memory must be preserved, swapped, or recomputed.
  - a challenge absent in traditional queueing systems.
- There are several works address the preemption challenge

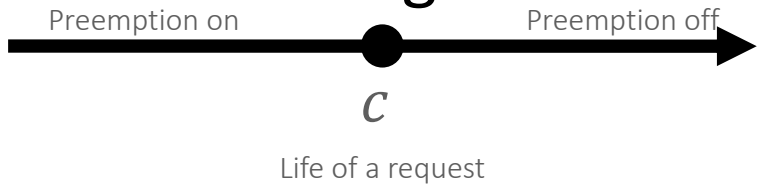
Existing works tackle preemption

- FastServe* uses a Multi-Level Feedback Queue (MLFQ). It maintains multiple queues with different service thresholds.
- When a job exceeds its current level's threshold, it is preempted and moved to a lower-priority queue.
- Proposes proactive GPU memory management by swapping KV cache between GPU and CPU. However, swapping incurs hidden costs:
  - limited PCIe bandwidth.
  - Fragmented KV cache causes multiple kernel launches and sync overhead.
  - And requests may stall during swap operations.

* Fast Distributed Inference Serving for Large Language Models. Wu et al.

Existing works tackle preemption

- Preempting late-stage requests is wasteful due to large KV cache already in memory.
- TRAIL* disables preemption once a request reaches a certain age threshold $c \cdot (\textit{predicted size})$
  - Age = the amount of time the job has been served.
  - Young requests (early stage): preemption is cheap (in memory) → allowed.
  - Old requests (late stage): preemption is costly → avoided.
  - Analyzed with SOAP framework in M/G/1 setting.



* Don’t Stop Me Now: Embedding Based Scheduling for LLMs. Shahout et al. ICLR 2025

Open Question

- The cost of preemption varies many factors like: model size, batch size, available memory, distribution of request sizes.

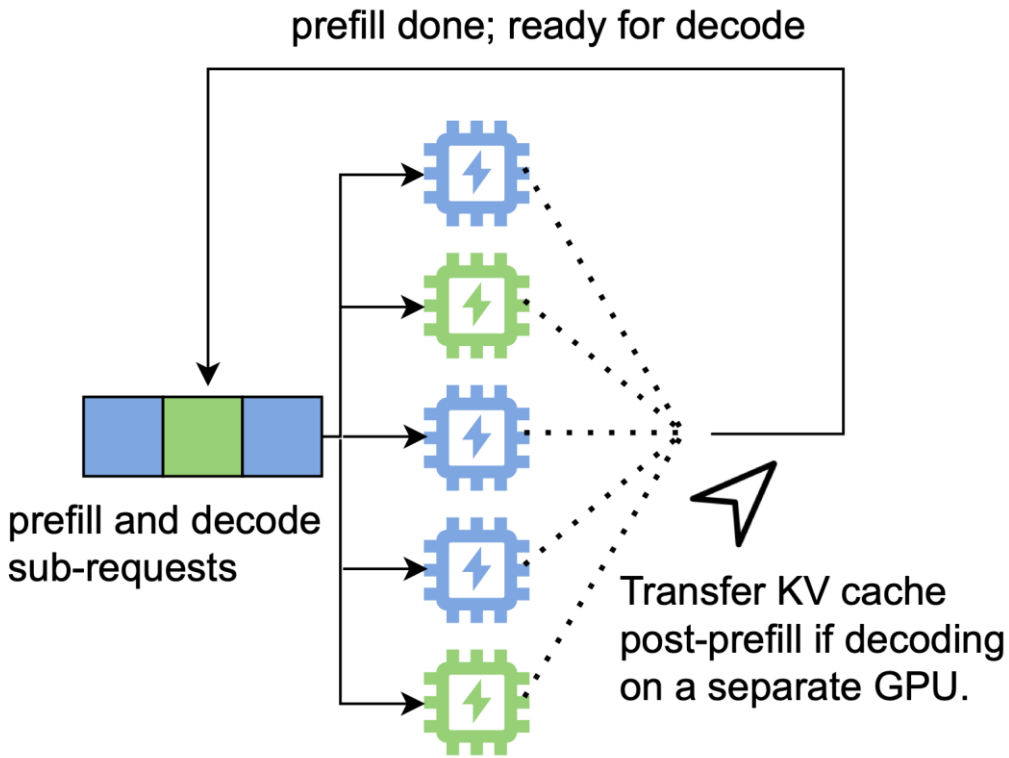
How can we dynamically tune preemption thresholds based on system state and request characteristics?

GPU Resource Allocation

- How should GPU resources be orchestrated to balance prefill and decode workloads?
- Decode-optimized GPUs may benefit from larger memory capacity to hold many KV caches, even with modest compute
- Prefill-optimized GPUs may prioritize high compute throughput, requiring less memory but faster execution

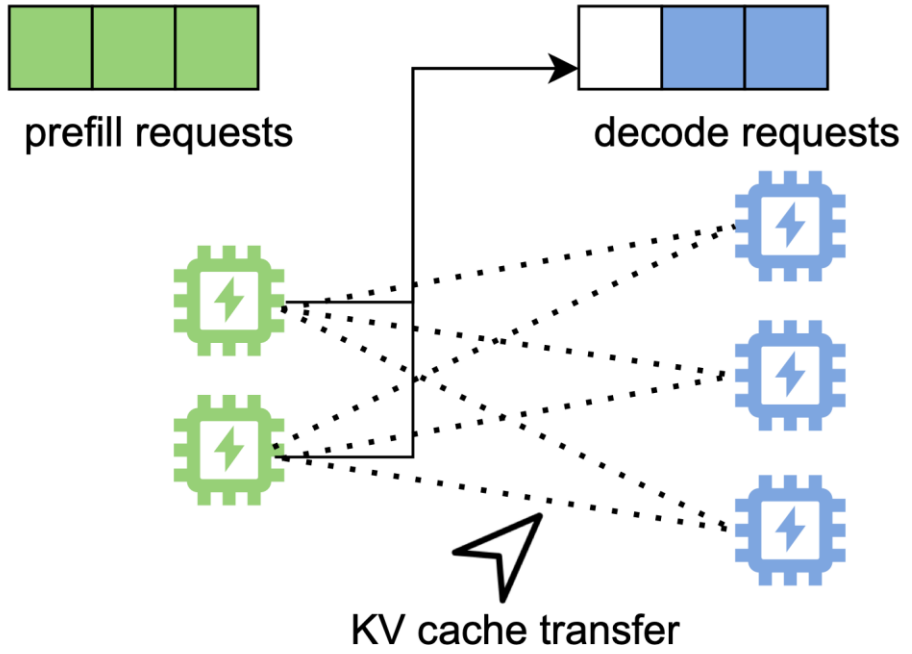
Pooled GPUs approach

- In a pooled setup, each GPU can handle both prefill and decode phases (improves flexibility and utilization).
- After prefill completes, a request may:
  - Continue decoding on the same GPU, or
  - Re-enter the queue and wait for another GPU to pick up the decode phase



Dedicated GPUs approach

- The GPU pool is partitioned into two groups:
  - One handles only prefill
  - The other handles only decode
- This resembles a tandem queueing system, but with added complexity:
  - GPUs may differ in speed (compute vs. memory-optimized)
  - KV cache transfer between prefill and decode GPUs introduces overhead



Splitwise: Efficient Generative LLM Inference Using Phase Splitting, Patel et al. ISCA 2024

Open Question

- How can scheduling policies coordinate prefill and decode across GPUs with varying compute and memory capabilities?
- Can queueing models, particularly tandem queues, help optimize scheduling in the dedicated GPU setup?
- Can a group of lower-capacity GPUs match or exceed the performance of a single high-end GPU?
  - Requires modeling of KV cache transfer overhead, interconnect bandwidth, and power consumption.
  - Theoretical and empirical analysis needed to guide when to scale up vs. scale out

Compound AI Systems

Compound AI systems

- So far, we've focused on scheduling for a single LLM.
- AI systems today are increasingly compound: they integrate multiple LLMs and external tools.
  - Augmented LLMs — external tool use.
  - Information retrieval — RAG.
- As compound systems grow, efficient scheduling and coordination across modules becomes a central challenge.

Augmented LLMs

- Extend standard LLMs by integrating external tools
  Examples: calculators for math, image generators for visual output, virtual machines for code execution
- API call durations can range from milliseconds to several seconds
- KV cache may remain allocated during API calls, depending on how the system manages the request
- Different augmentation types require handling strategies
  A single strategy is unlikely to perform well across all cases

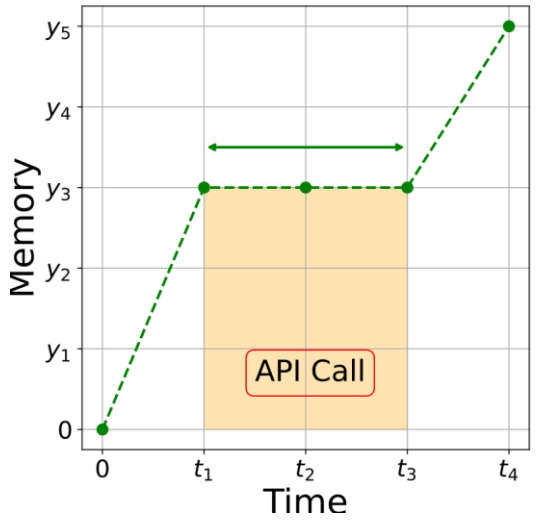
Handling Requests with API Calls: Three Strategies

1. **Preserve**: keeps the KV cache in memory while waiting for an API response. Memory wasteful during the call.
2. **Discard and Recompute**: discards and re- computes the KV matrices from the start once the API returns. Re-computation overhead
3. **Swap**: offloads the KV cache to the CPU to free up memory, and reloads it when the API returns.

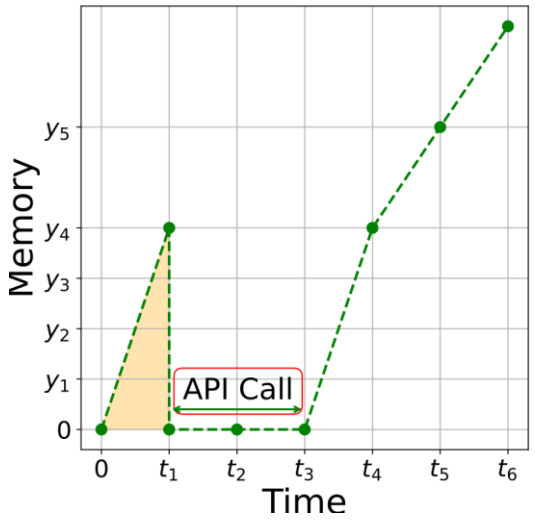
Pausing running requests and managing data transfer between CPU and GPU memory

Memory consumption of the handling strategies

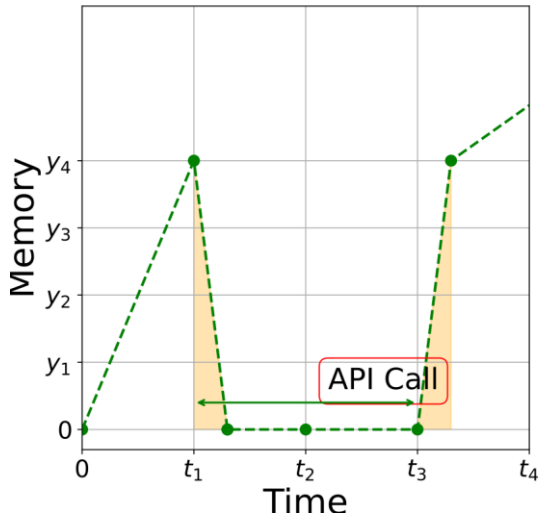
Memory consumption over time for a request with one API call. The highlighted area represents memory waste for one request.



Preserve

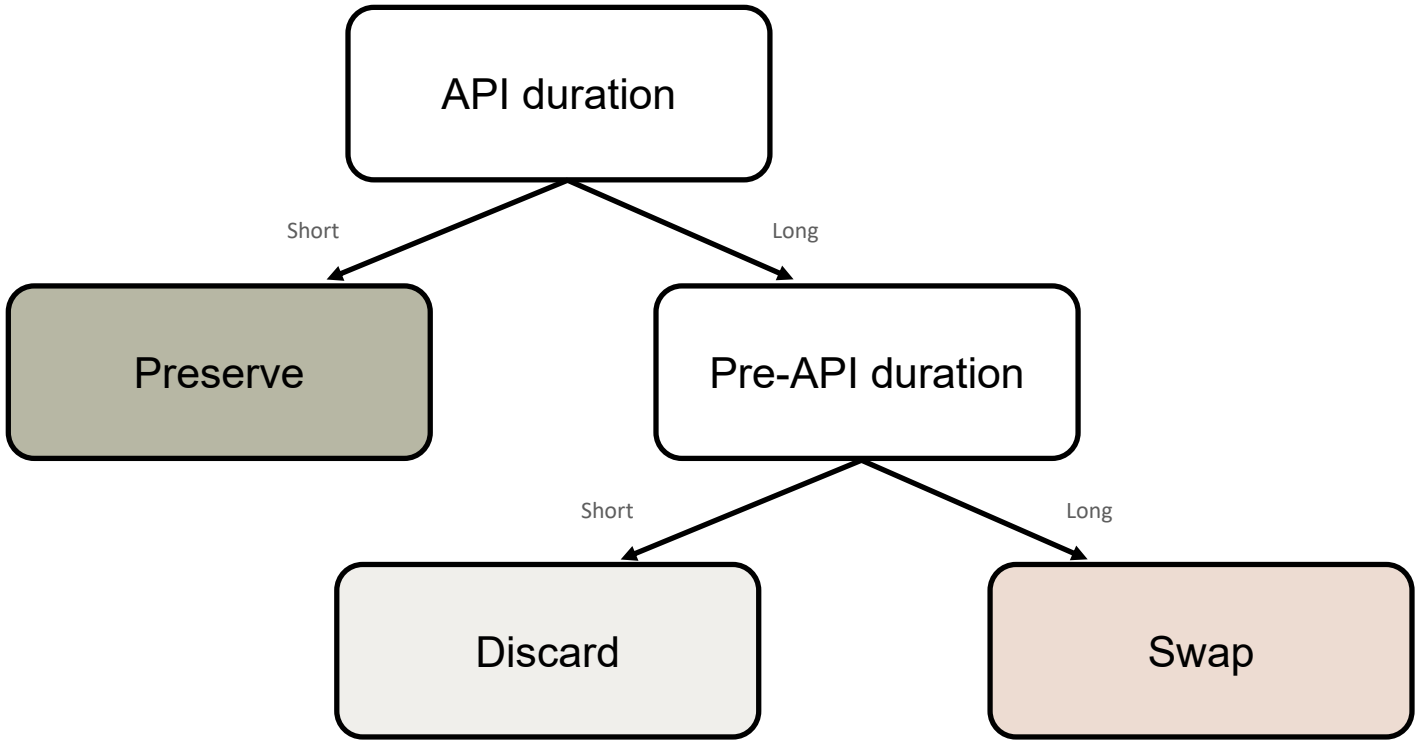


Discard and Recompute



Swap

No one strategy for all API calls



InferCept: Efficient Intercept Support for Augmented Large Language Model Inference. Abhyankar et al. ICML 2024

Traditional Scheduling don't work with API

- Example

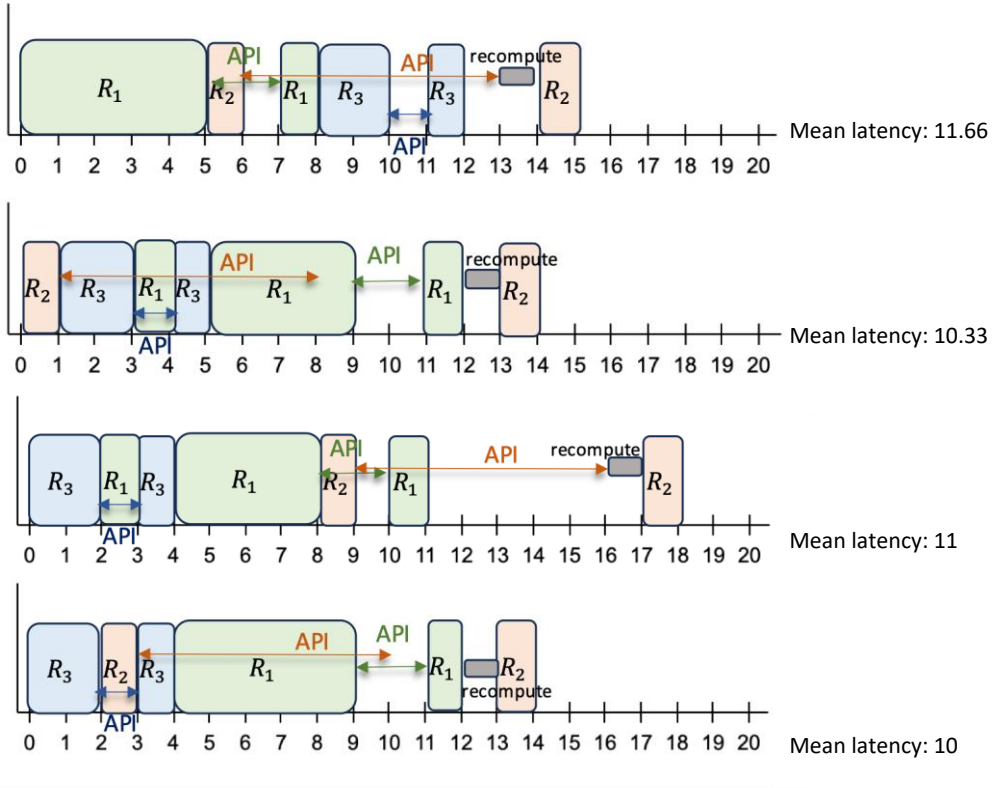
	R_1	R_2	R_3
Total length	6	2	3
API start after	5	1	2
API duration	2	7	1
Memory action	Preserve	Discard	Swap

first come first serve

Shortest jobs first

Shortest (size+API duration) first

Optimal?



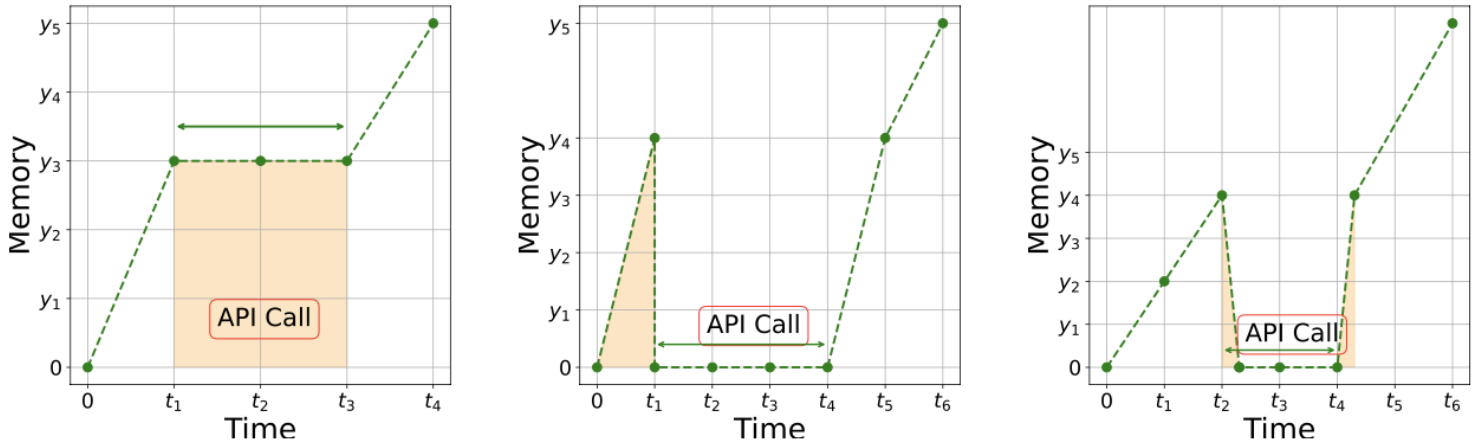
MARS: Memory-Aware Scheduling for Augmented LLMs*

- First system to go beyond FIFO for scheduling augmented LLMs
- Proposes a predictive, memory-aware scheduler that jointly handles:
  - API strategy selection
  - Request prioritization
- Integrates scheduling and API handling
  - Determines optimal handling strategy (Preserve, Discard, Swap) before process the request
  - Rank requests based on memory consumption over time.

* Fast Inference for Augmented Large Language Models. Shahout et al. 2024

Step 1

- So we need to predict or estimate: API duration and the length of the pre-API output
- For each request, we pick the strategy with the minimal KV cache memory waste (yellow area and impact on the system)

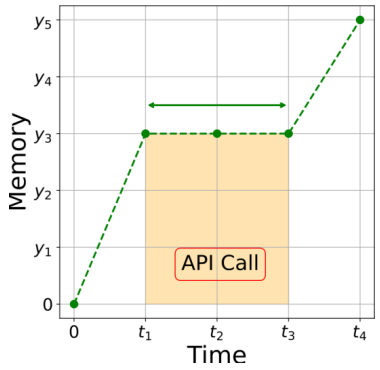


(a) Preserve **(b)** Discard **(c)** Swap

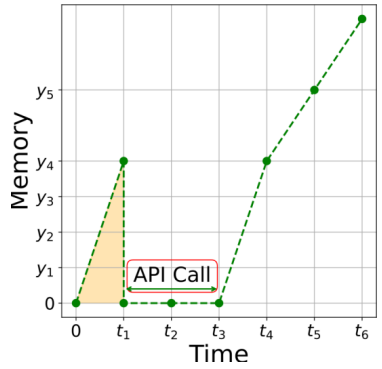
Step 2

- Consider the handling strategy in the scheduling ranking

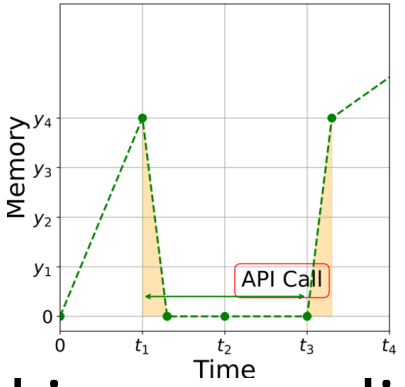
Preserve



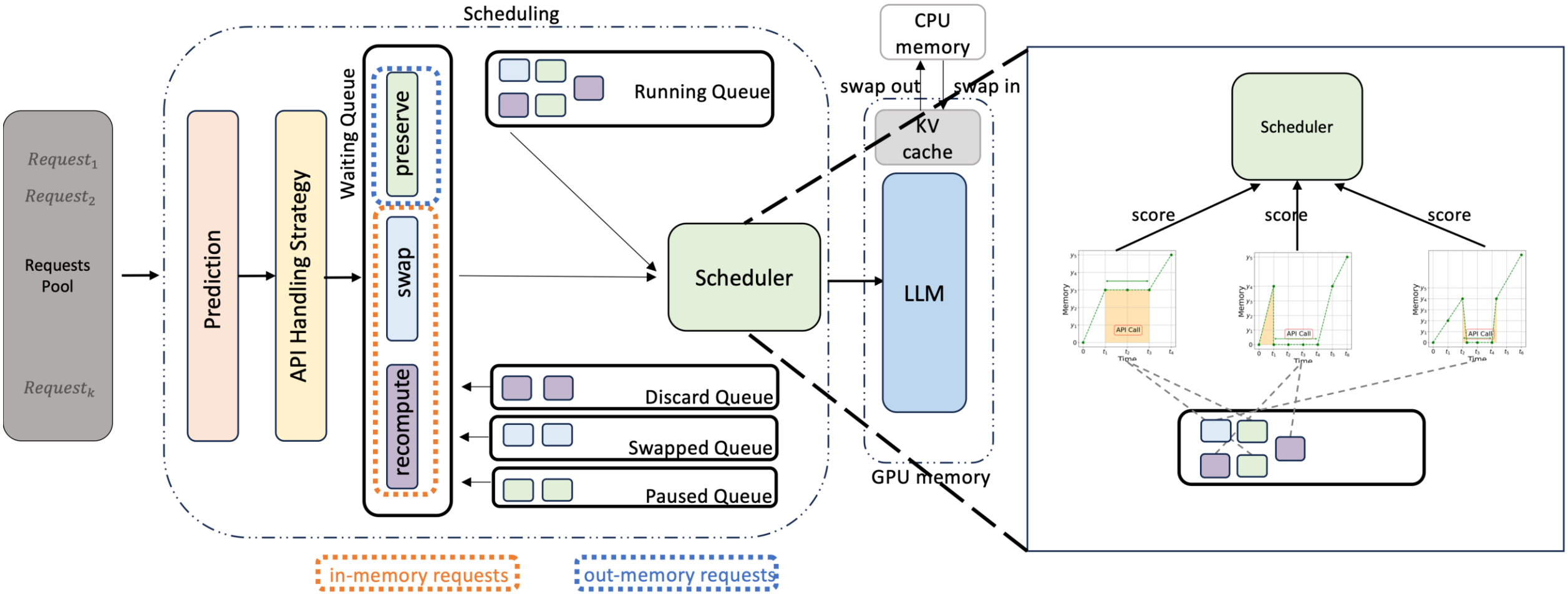
Discard



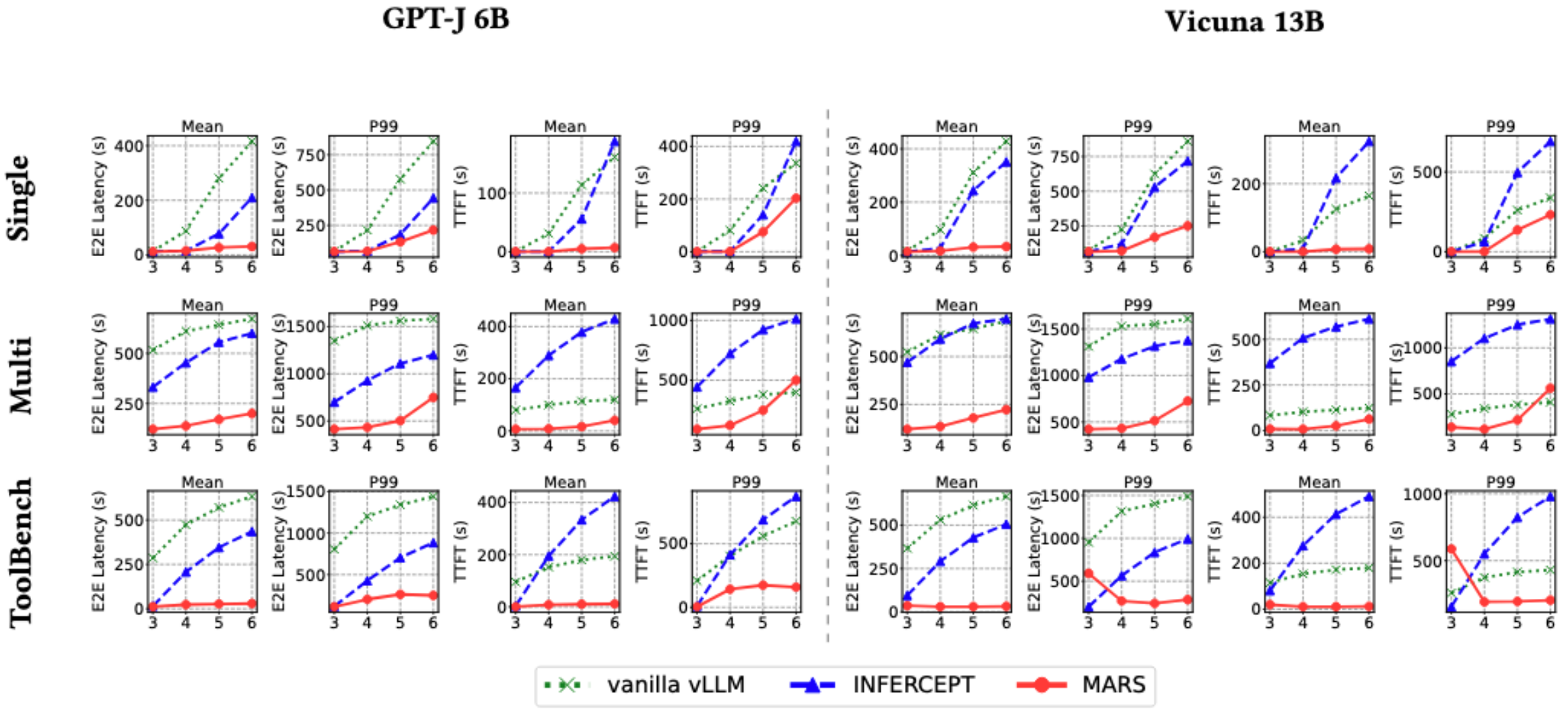
Swap



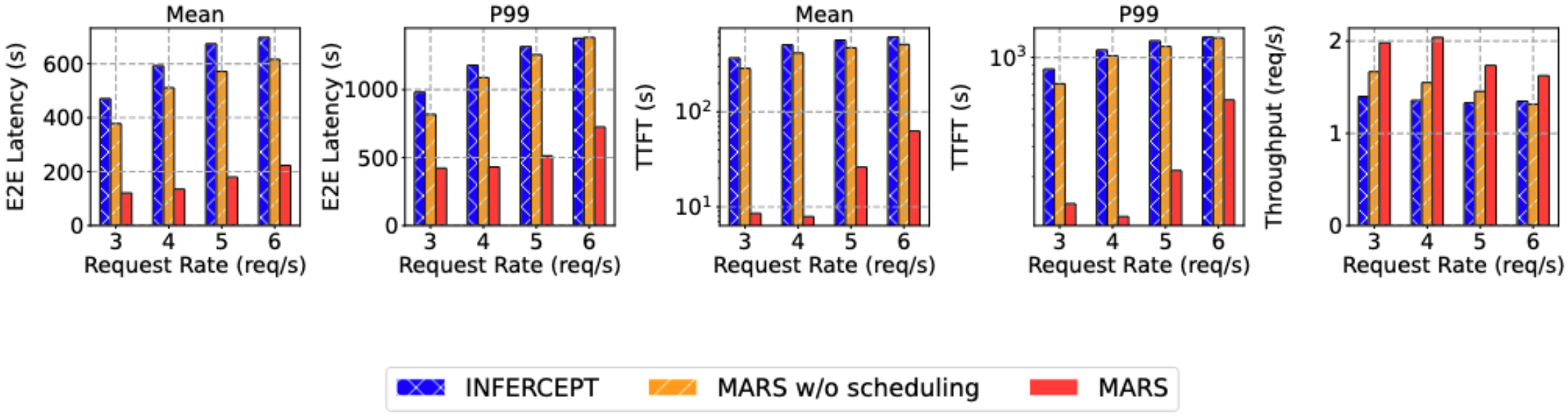
- Check: without API calls, this is the same as ranking according to request length.


Scheduling
$Request_1$
$Request_2$
Requests Pool
$Request_k$
Prediction
API Handling Strategy
Waiting Queue
preserve
swap
recompute
Running Queue
Scheduler
Discard Queue
Swapped Queue
Paused Queue
CPU memory
swap out
swap in
KV cache
LLM
GPU memory
in-memory requests
out-memory requests
Scheduler
score
score
score
Memory
Time
API Call

MARS achieves lower latency and faster time to first token (TTFT)



MARS breakdown



Open Question

- What theoretical guarantees can we establish for scheduling API-augmented requests?
  Can we develop simplified abstractions with provable bounds to guide practical scheduler design?

Multiple LLMs Available

- Compound AI systems often include LLMs of different sizes.
- Model size affects runtime, cost, and output quality:
  - Smaller models are sufficient for simple queries.
  - Larger models provide better reasoning for complex prompts, but with higher cost and latency.
- Always using large models leads to unnecessary cost, latency, and resource load.
- Prediction beyond size: In this setting, we aim to predict answer quality, not just output length.
- Implication: Enables routing based on query complexity and size.

Multiple LLMs Needed

- Many compound AI systems require multiple LLMs working together
- There are multiple settings:
  - One setting: each LLM handles a specific subtask, and a coordinator aggregates the outputs
    challenge: assign subtasks to heterogeneous GPUs to maximize parallelism and minimize coordination overhead
    Emphasizes parallel, independent subtasks
  - Another setting: each stage refines the output of the previous one
    challenge: manage stage dependencies and balance pipeline execution with synchronization
    Focuses on ordered, interdependent computation

Open Question

- How can we design effective scheduling algorithms for multi-LLM systems
  both for the two settings?
- Can predictions go beyond output size, for example, predicting the execution path (which LLMs will be needed)?
- How can such predictions guide scheduling decisions in multi-LLM systems?

Reasoning Systems

Scheduling in LLM Reasoning Systems

- Modern LLMs can handle complex reasoning tasks
  - Examples: math, code generation, legal analysis
- They do this by exploring multiple reasoning paths and selecting the best one
- This requires inference-time algorithms that go beyond standard generation
- Reasoning Process:
  - Expansion: Generate candidate reasoning paths
  - Aggregation: Combine or select from candidates to produce the final answer

Open Research Challenges

- Ranking ≠ token length: True “request size” includes the number of reasoning steps and tokens per step
- Scheduling requires adaptive predictions:
  - Predict reasoning complexity (e.g., easy/moderate/hard)
  - Estimate token count per path
- Update predictions and scheduling decisions as intermediate outputs arrive
- Dynamic resource allocation:
  - Allocate more compute to promising branches
  - Terminate unpromising ones early

Queueing, Predictions, and LLMs: Challenges and Open Problems

Michael Mitzenmacher[1] and Rana Shahout[1]

[1]Harvard University, USA

Abstract

Queueing systems present many opportunities for applying machine-learning predictions, such as estimated service times, to improve system performance. This integration raises numerous open questions about how predictions can be effectively leveraged to improve scheduling decisions. Recent studies explore queues with predicted service times, typically aiming to minimize job time in the system. We review these works, highlight the effectiveness of predictions, and present open questions on queue performance.

We then move to consider an important practical example of using predictions in scheduling, namely Large Language Model (LLM) systems, which presents novel scheduling challenges and highlights the potential for predictions to improve performance. In particular, we consider LLMs performing inference. Inference requests (jobs) in LLM systems are inherently complex; they have variable inference times, dynamic memory footprints that are constrained by key-value (KV) store memory limitations, and multiple possible preemption approaches that affect performance differently. We provide background on the important aspects of scheduling in LLM systems, and introduce new models and open problems that arise from them. We argue that there are significant opportunities for applying insights and analysis from queueing theory to scheduling in LLM systems.

1 Introduction

In this paper, we survey recent work on using predictions in queueing systems, as well as recent work on the specific setting of scheduling in Large Language Model (LLM) systems, where predictions seem both useful and natural. We focus on presenting several open questions for consideration. Our purpose is to highlight the work in these areas, and encourage researchers to tackle the many interesting problems raised by systems that make use of predictions.[1]

To introduce the queueing theoretic problems, let us consider a standard queue, such as an M/G/1 queue – where jobs arrive to a single-server queue, according to a Poisson arrival process