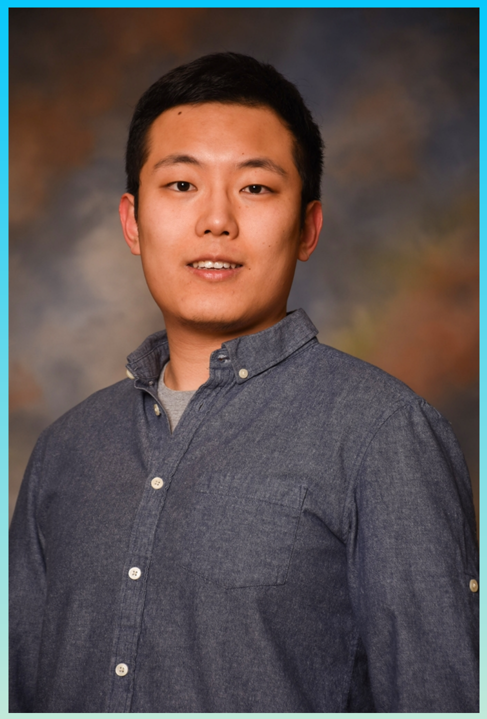
Serving DNNs like Clockwork

Performance Predictability from the Bottom Up


Arpan Gujarati


Safya Alzayat


Wei Hao


Antoine Kaufman


Jonathan Mace

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS


Reza Karimi


Ymir Vigfusson

EMORY UNIVERSITY

Serving DNNs like Clockwork

Performance Predictability from the Bottom Up

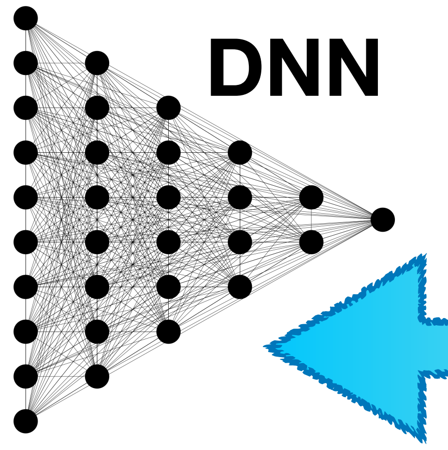
Serving DNNs like Clockwork

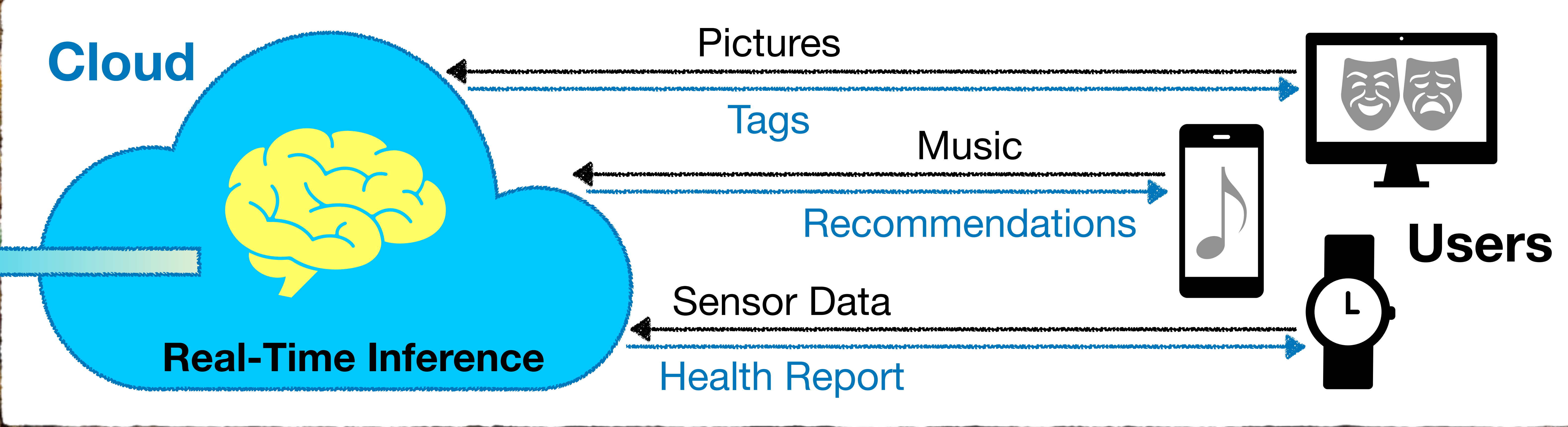
Performance Predictability from the Bottom Up

Serving DNNs like <u>Clockwork</u>

Performance Predictability from the Bottom Up

DNN inference has a very predictable execution time!





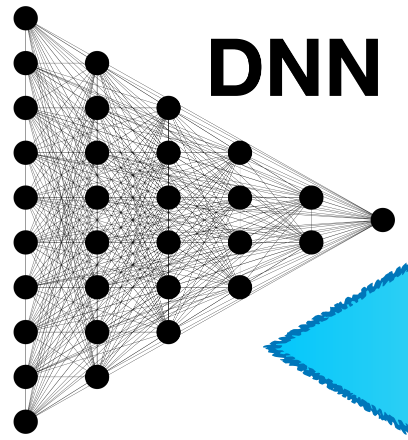
<u>Clockwork</u>

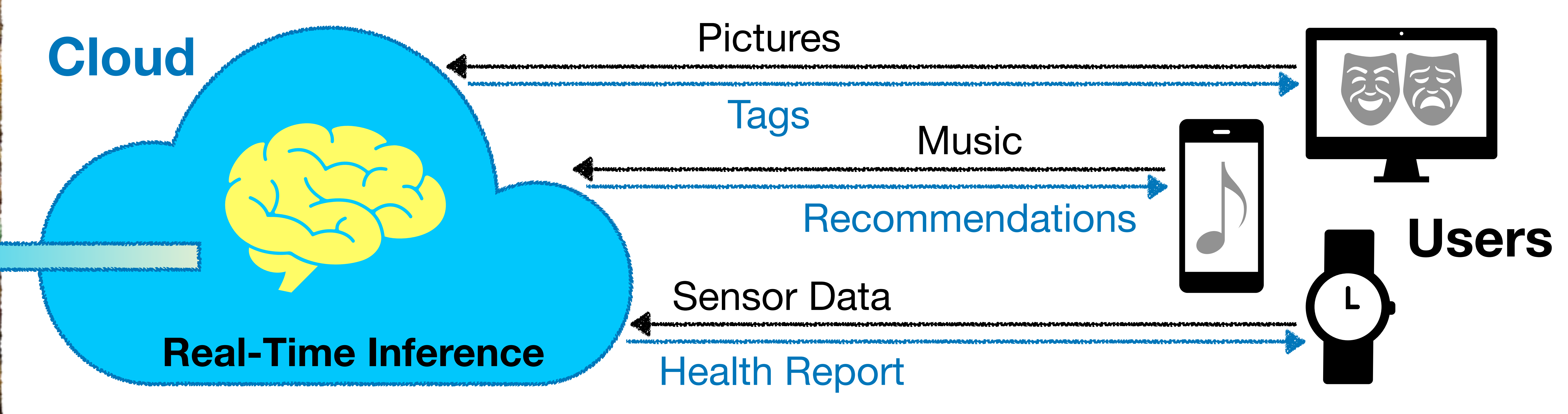
End-to-end predictable DNN serving platform for the Cloud

Serving DNNs like Clockwork

Performance Predictability from the Bottom Up

DNN inference has a very predictable execution time!





Clockwork

End-to-end predictable DNN serving platform for the Cloud

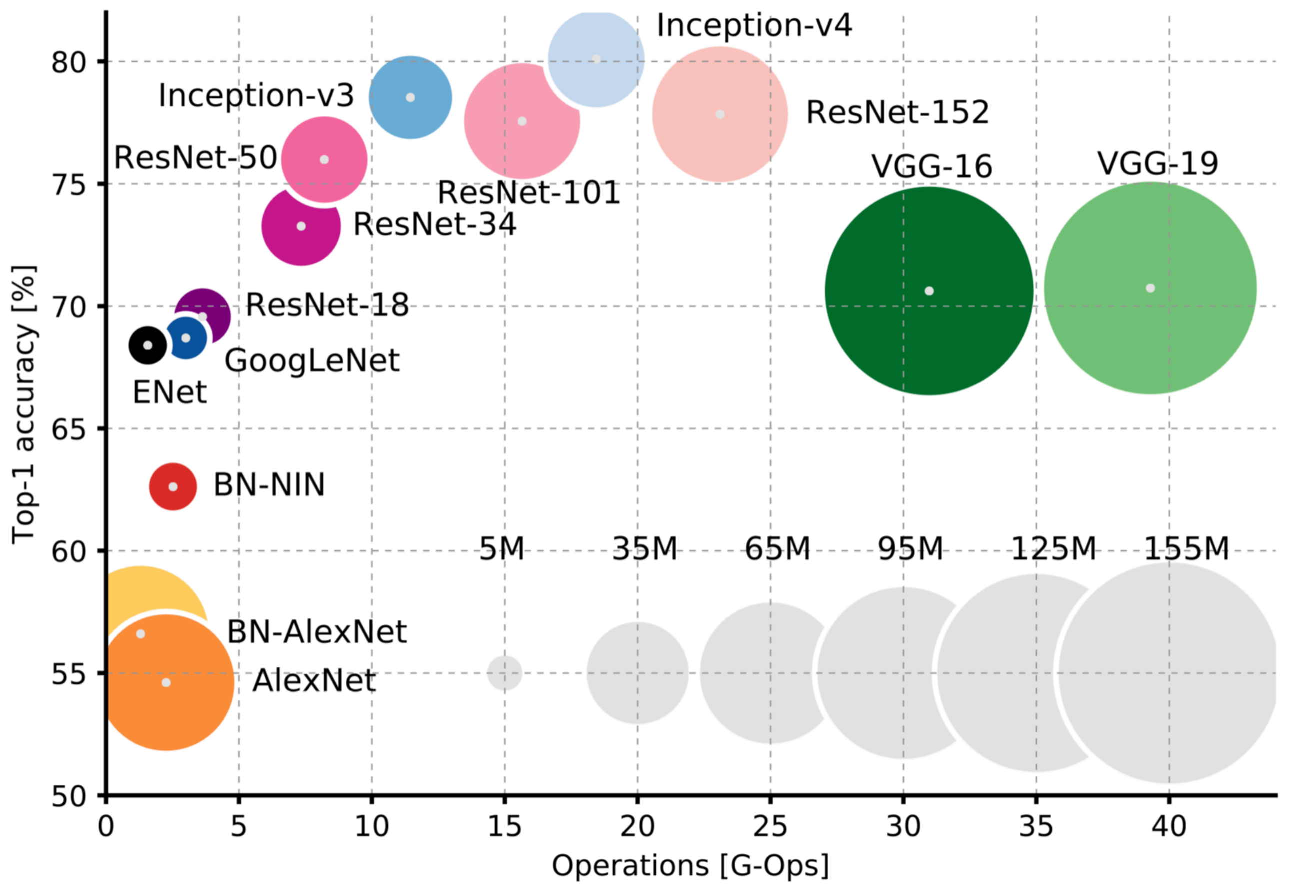
- ✓ Supports 1000s of models concurrently per GPU
- ✓ Mitigates tail latency, supporting tight latency SLOs (10—100 ms)
- ✓ Close to ideal goodput under overload, contention, and bursts

Background

Inference Serving at the Cloud Scale is Difficult

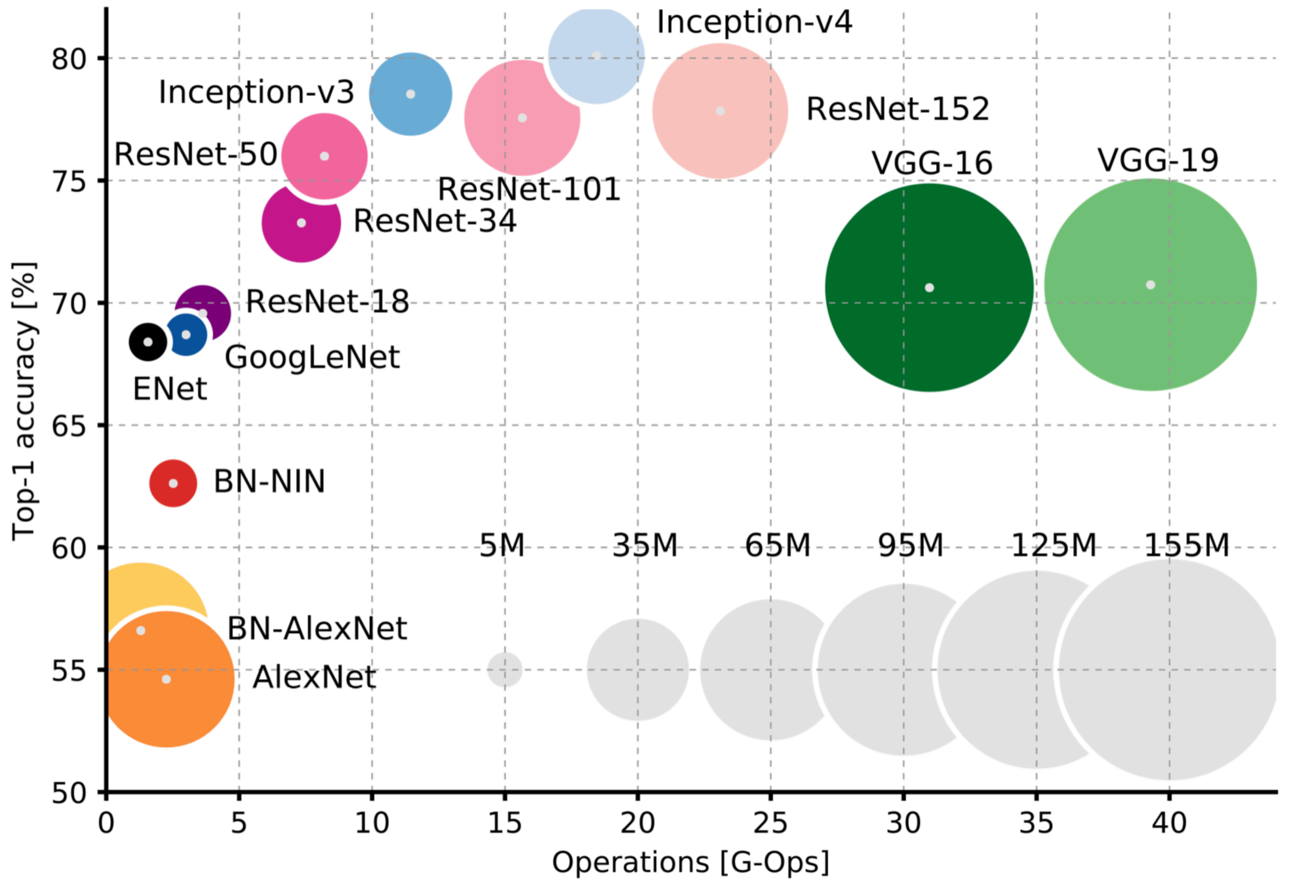
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

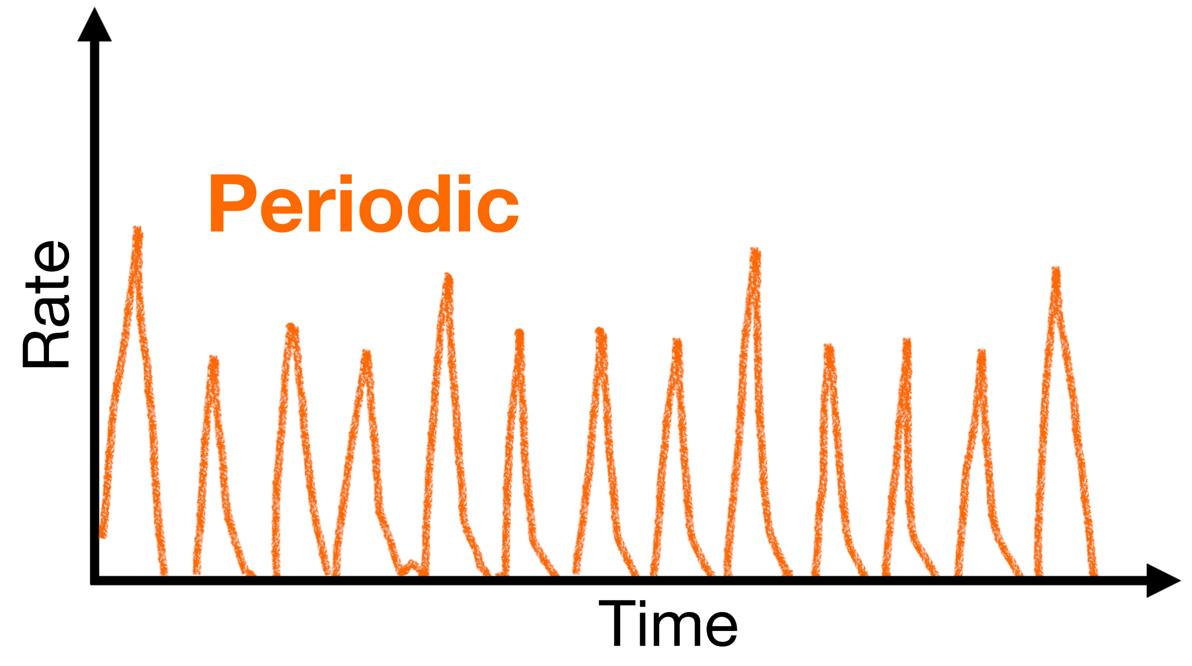


Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

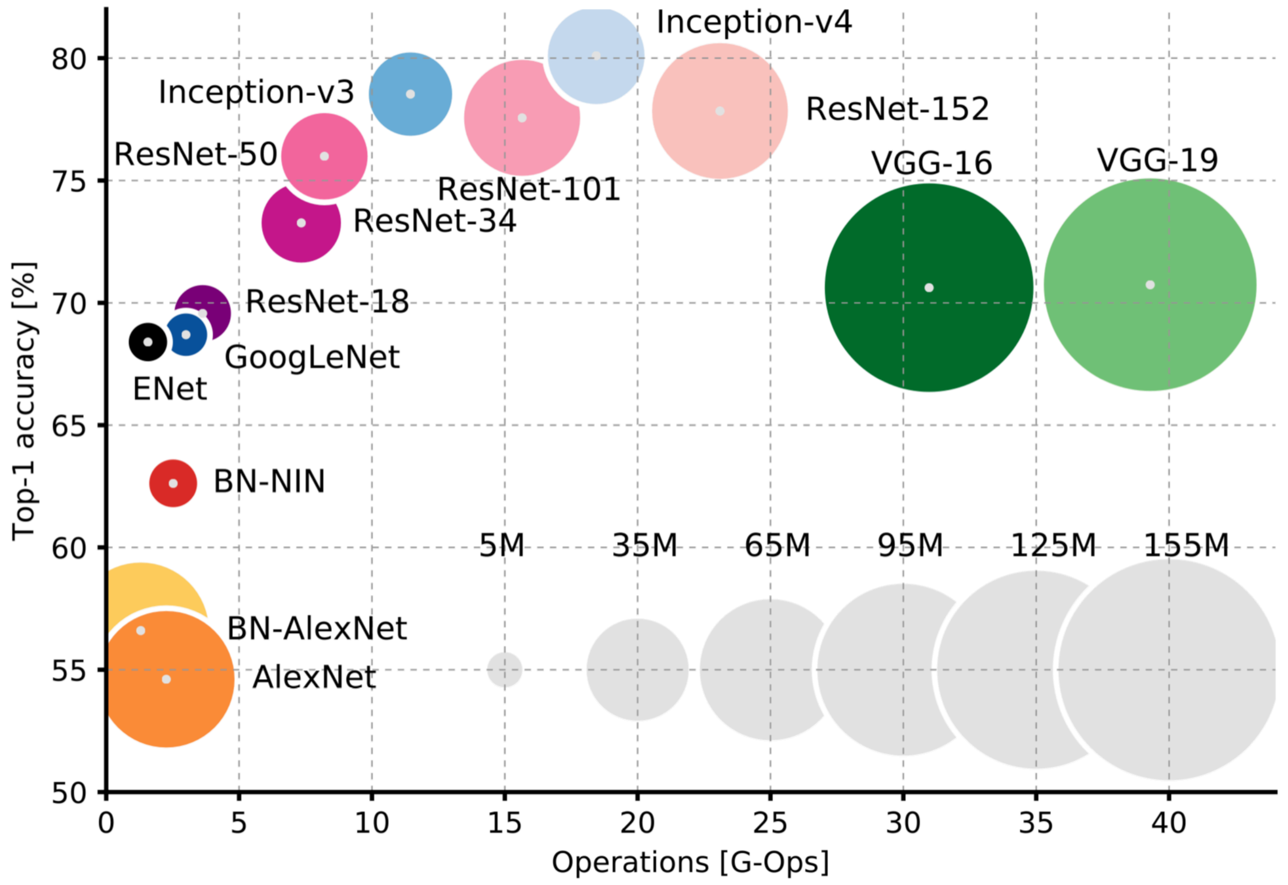


Requests arrive at different rates and regularity

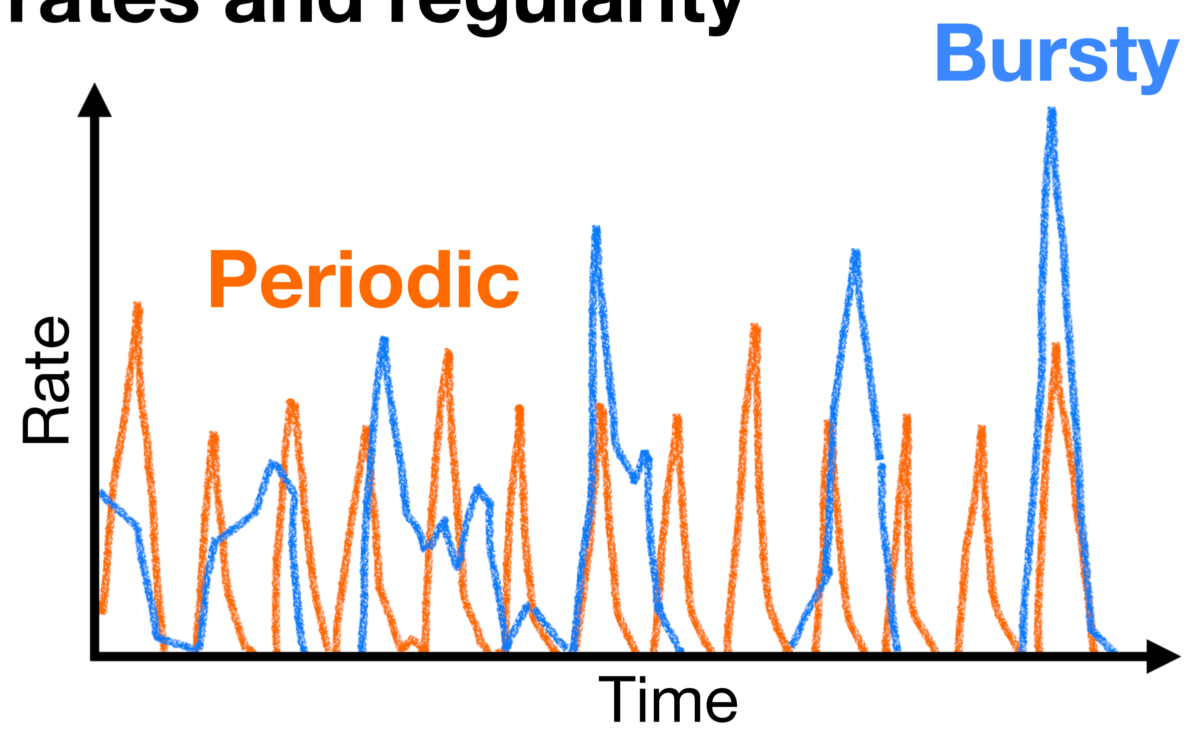


Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

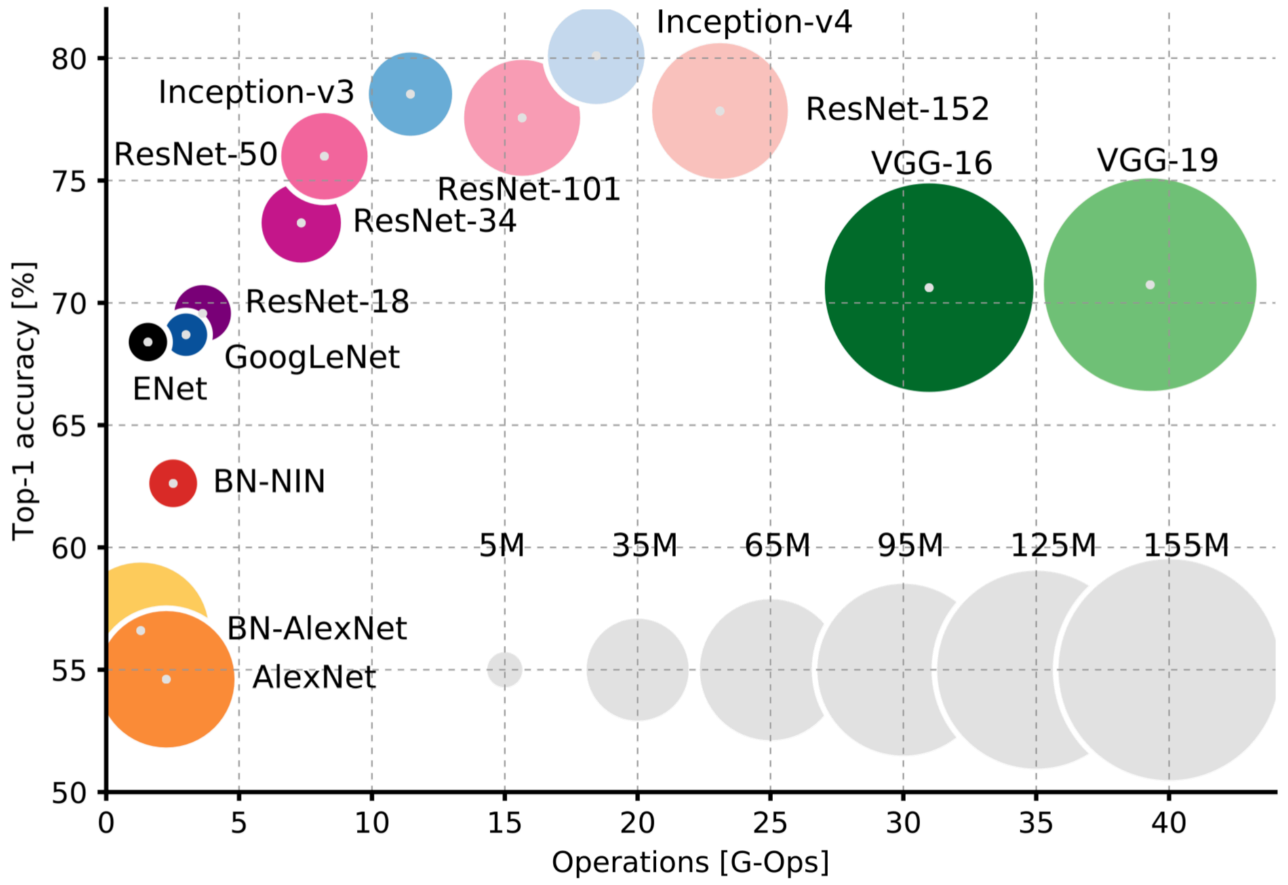


Requests arrive at different rates and regularity

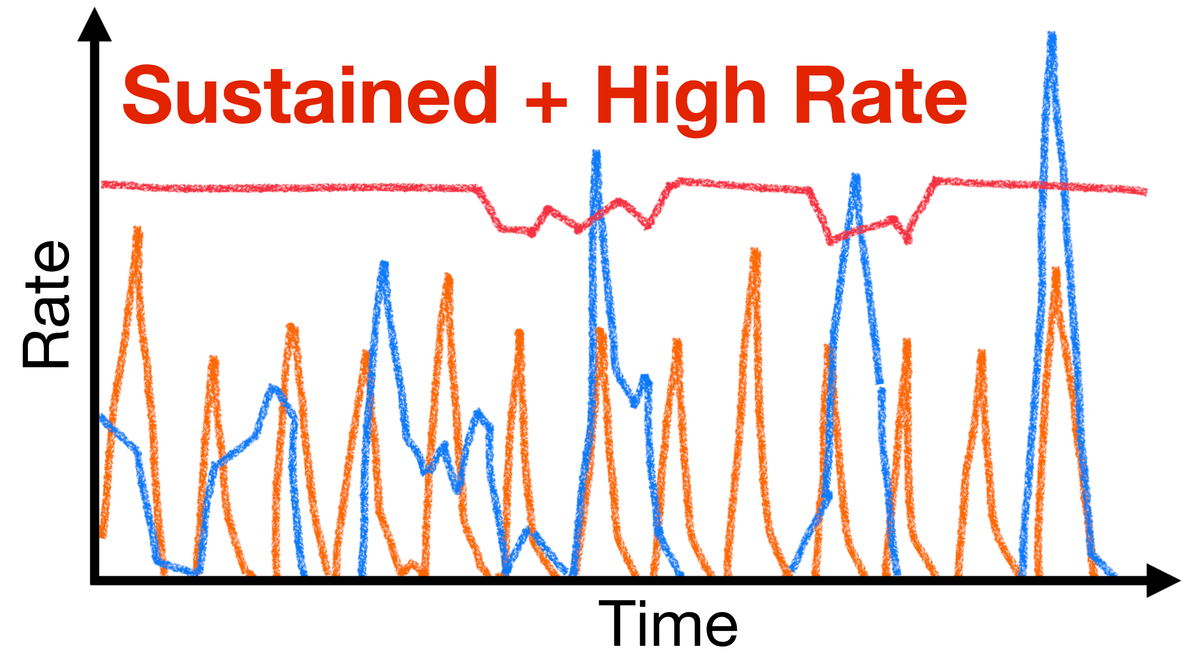


Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

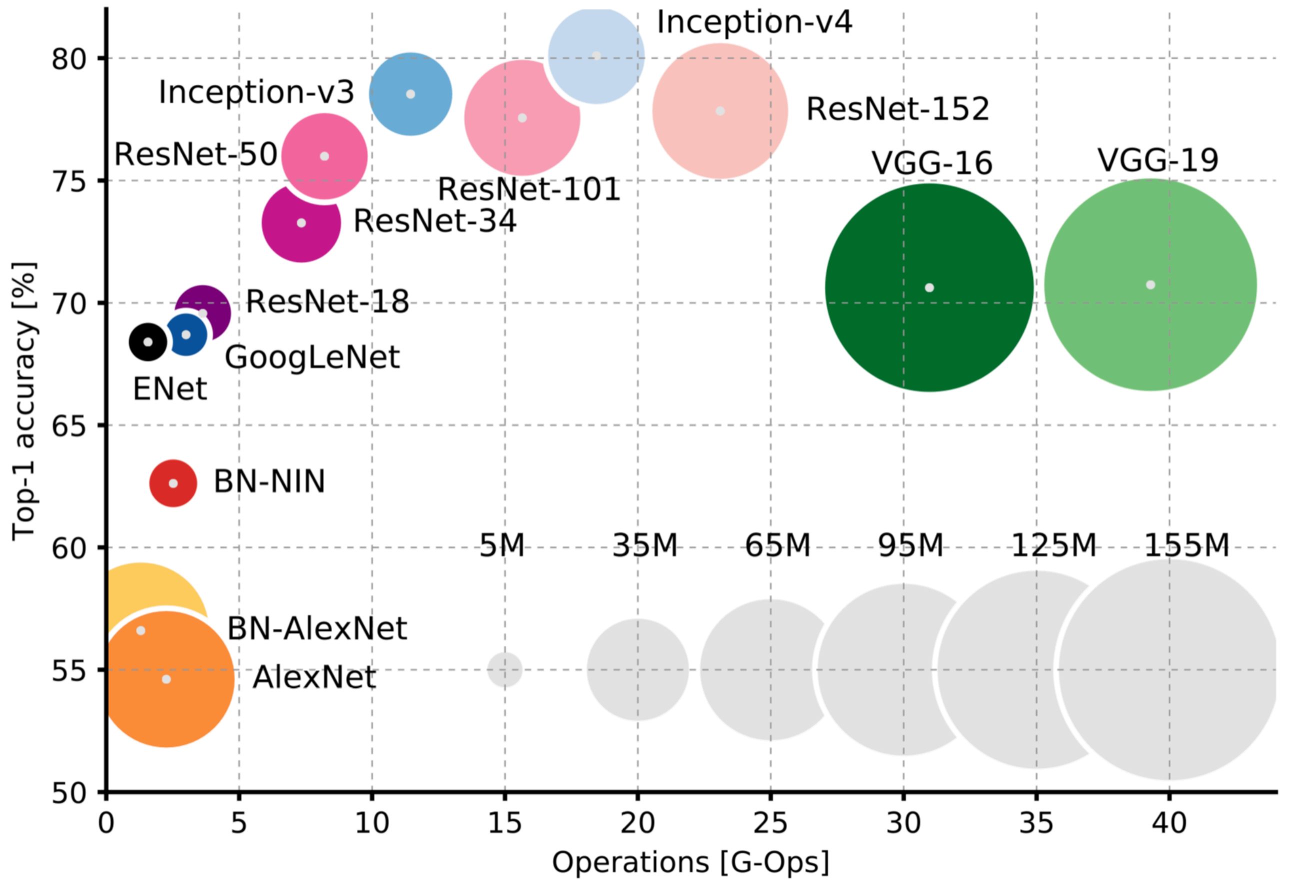


Requests arrive at different rates and regularity

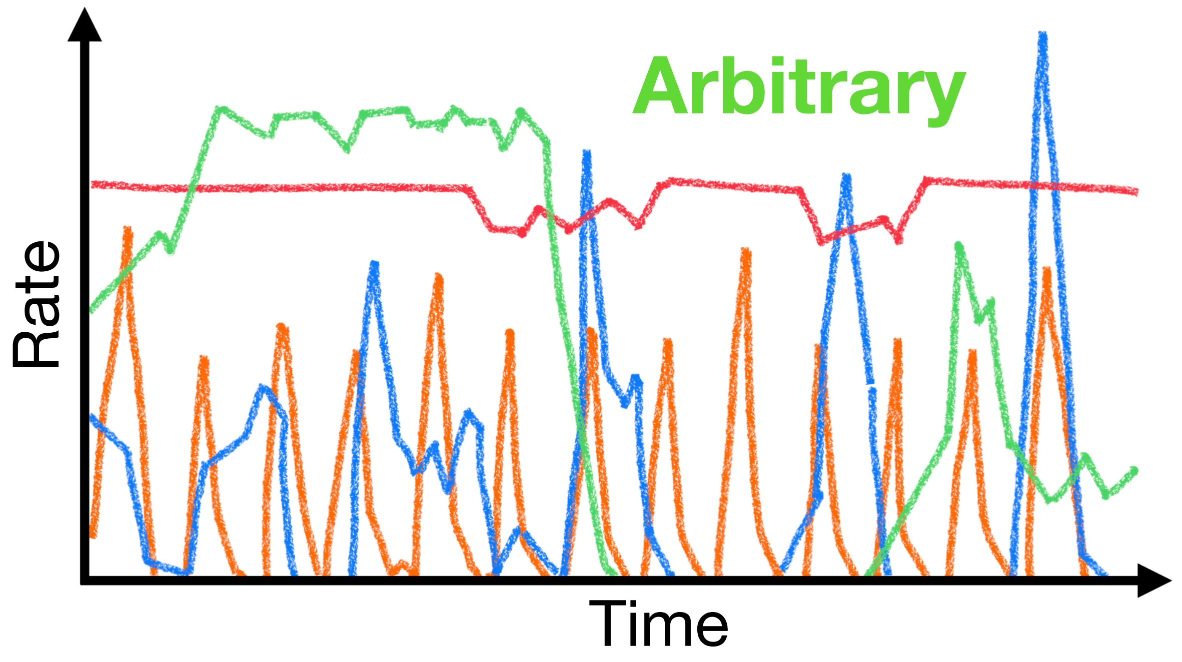


Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

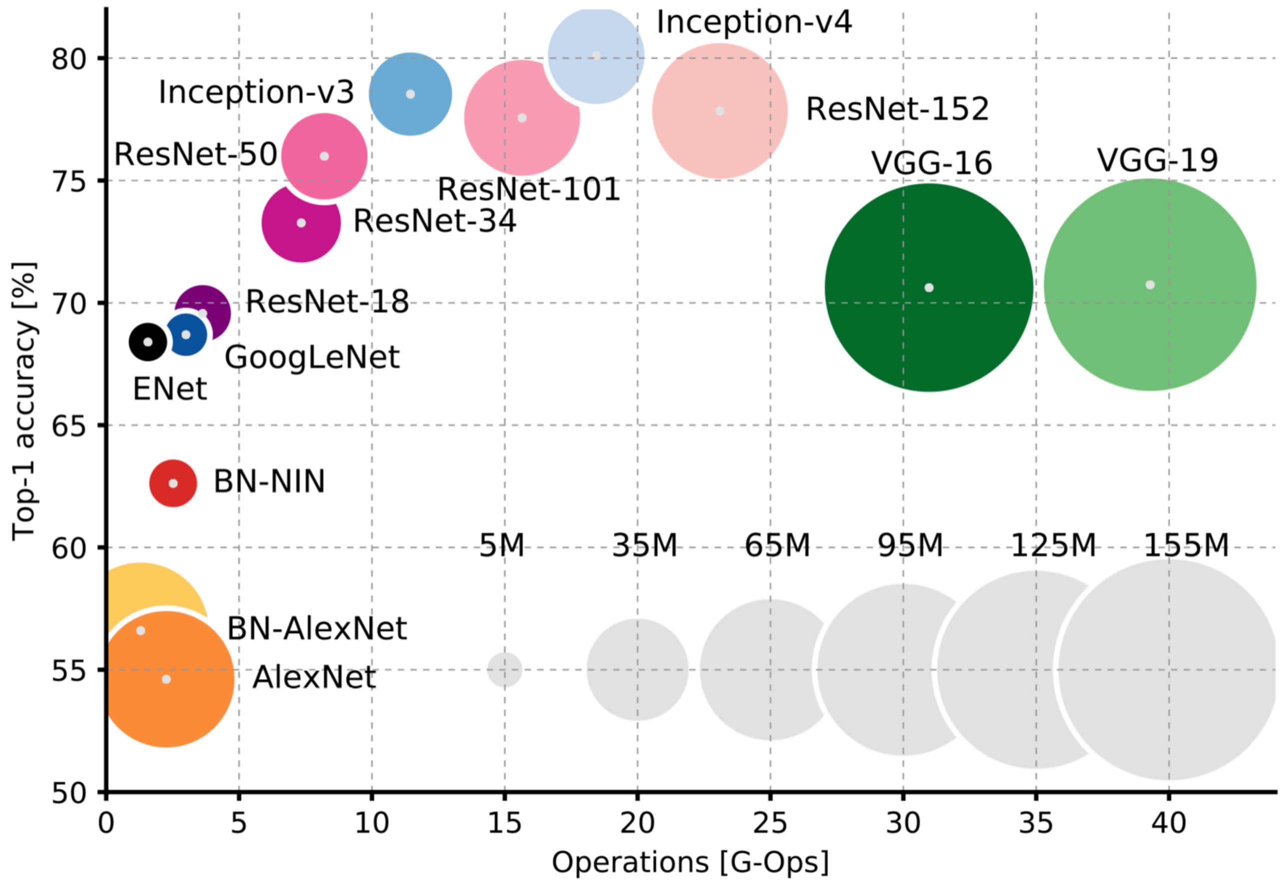


Requests arrive at different rates and regularity

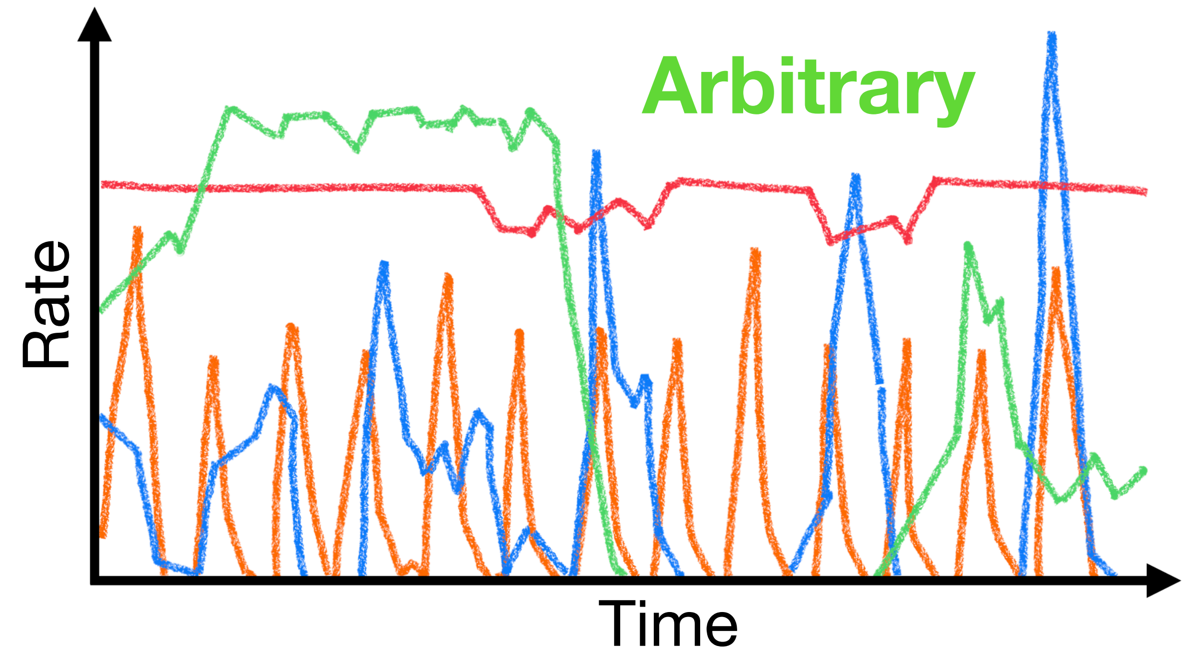


Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements



Requests arrive at different rates and regularity



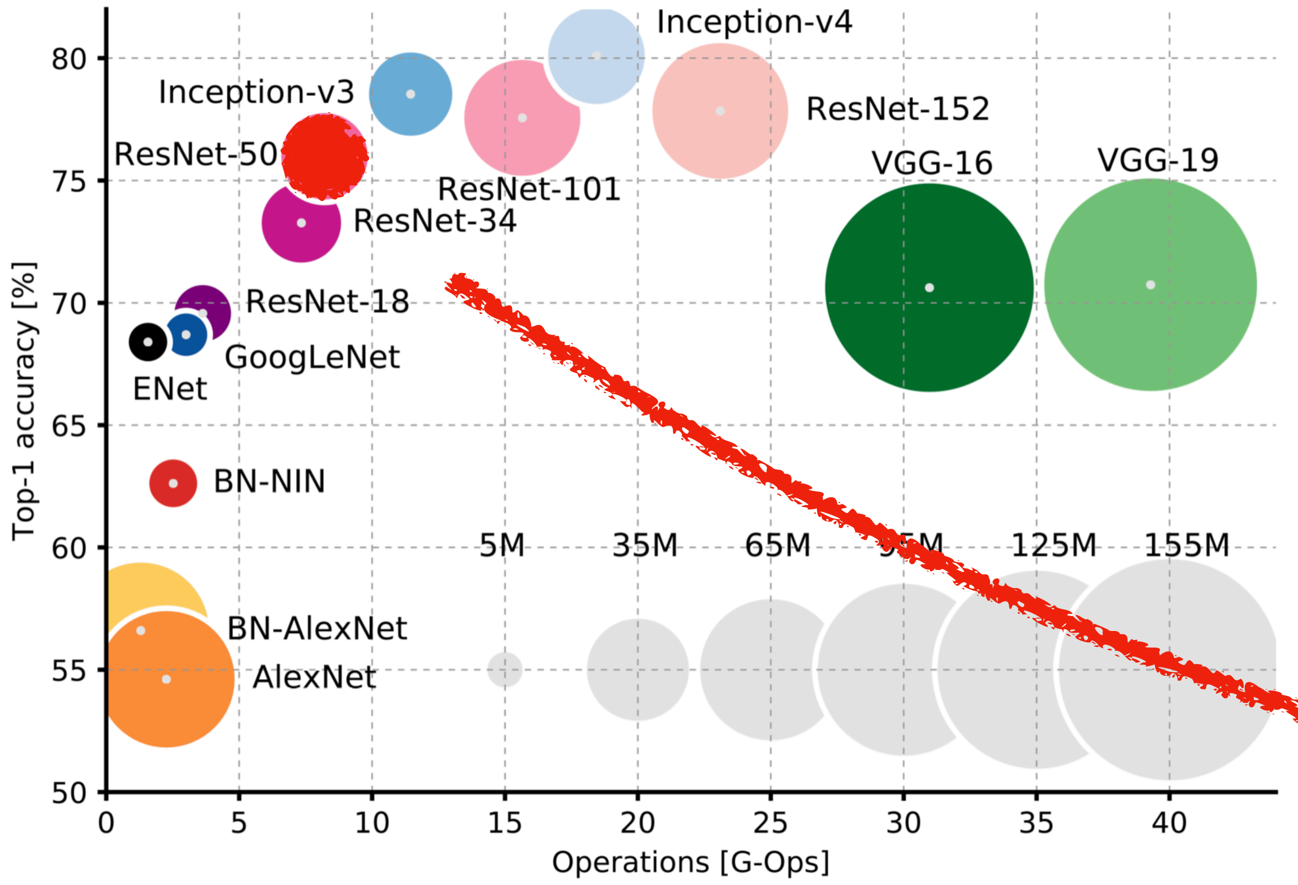
Each request has an inherent deadline

Latency SLOs

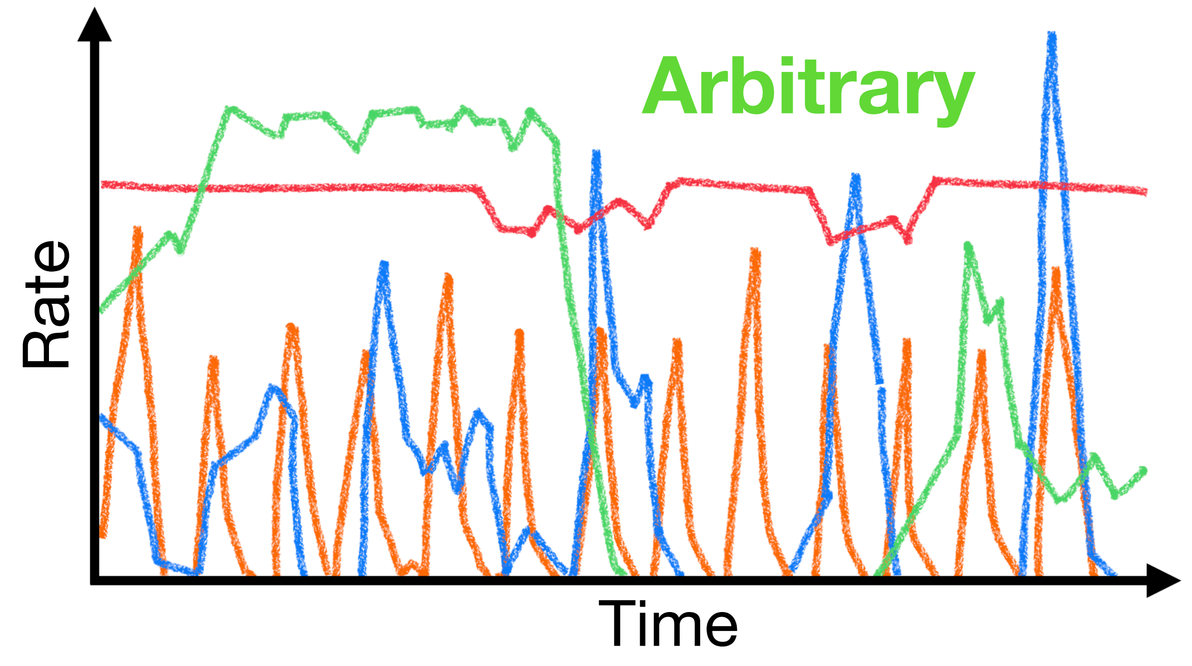
(e.g., 100ms)

Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements



Requests arrive at different rates and regularity

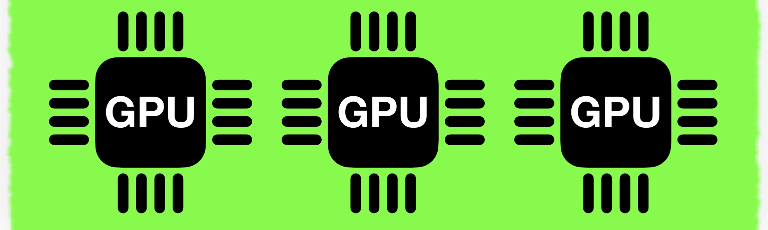


Each request has an inherent deadline

Latency SLOs

(e.g., 100ms)

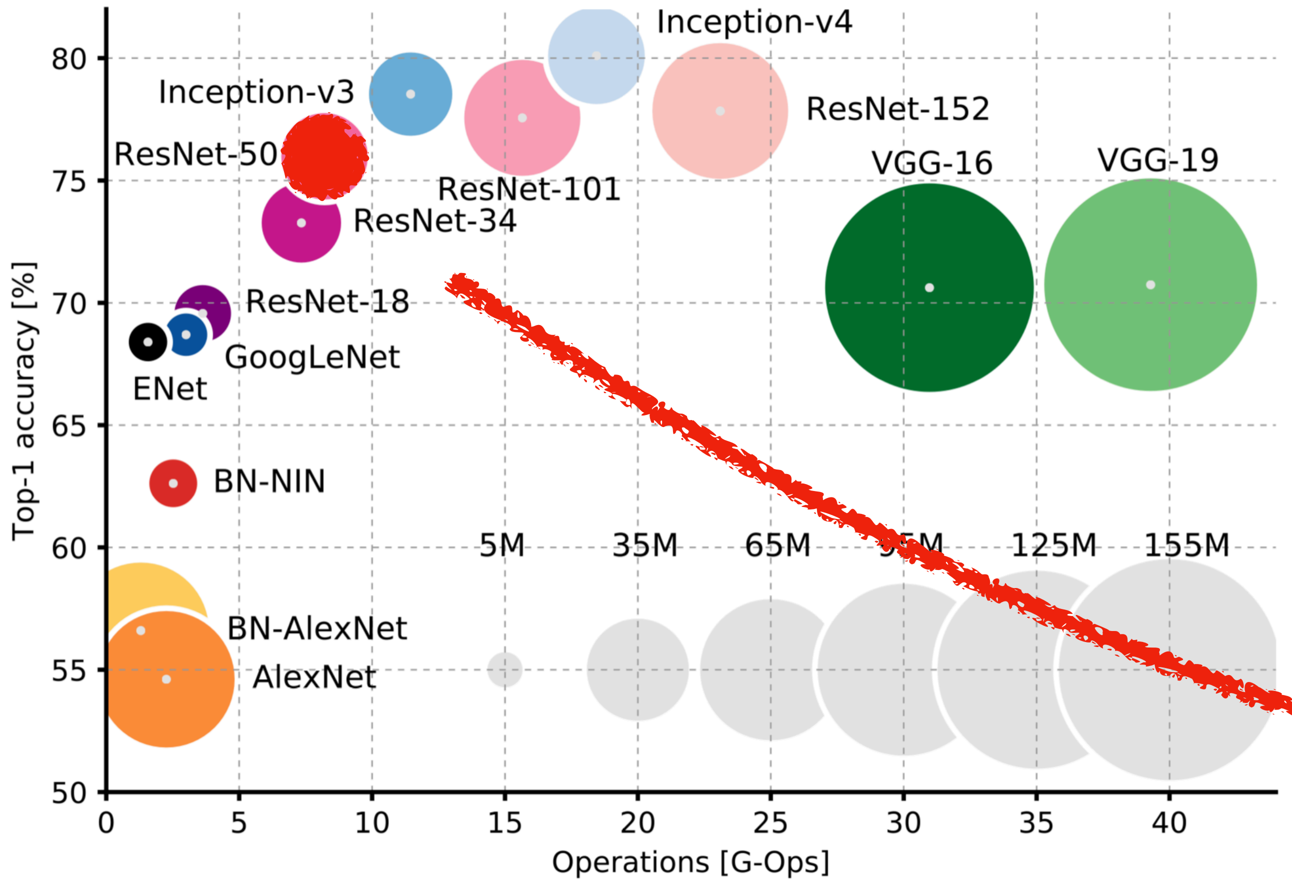
HW accelerators are necessary!



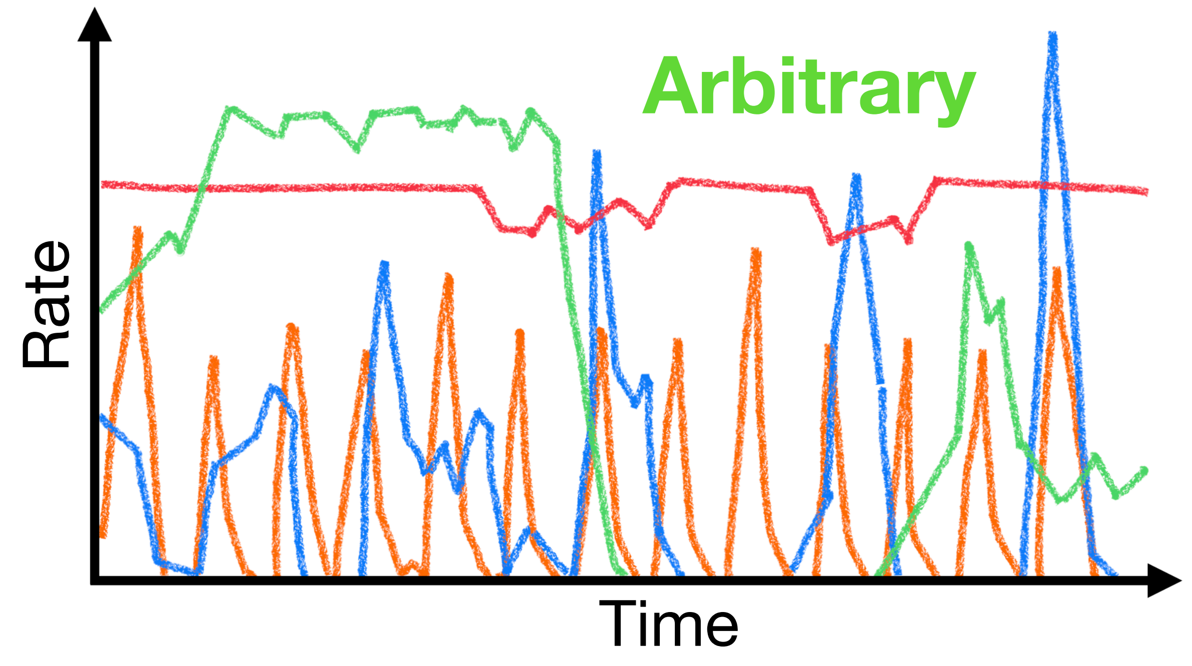
ResNet-50	Latency	Throughput
CPU	175 ms	6 req/s
GPU	2.8 ms	350 req/s

Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements



Requests arrive at different rates and regularity

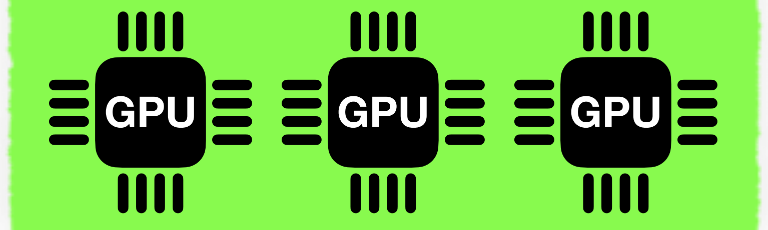


Each request has an inherent deadline

Latency SLOs

(e.g., 100ms)

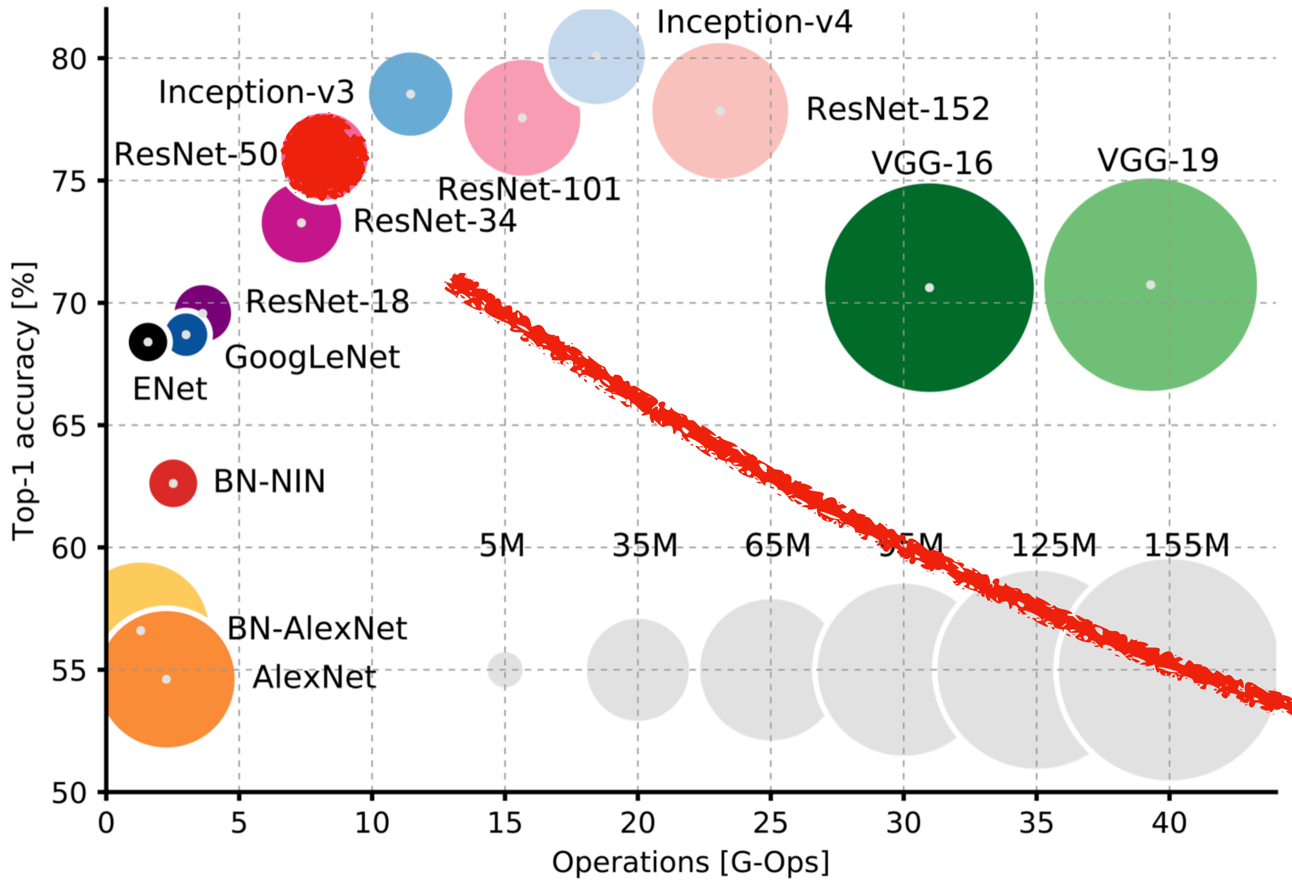
HW accelerators are necessary!



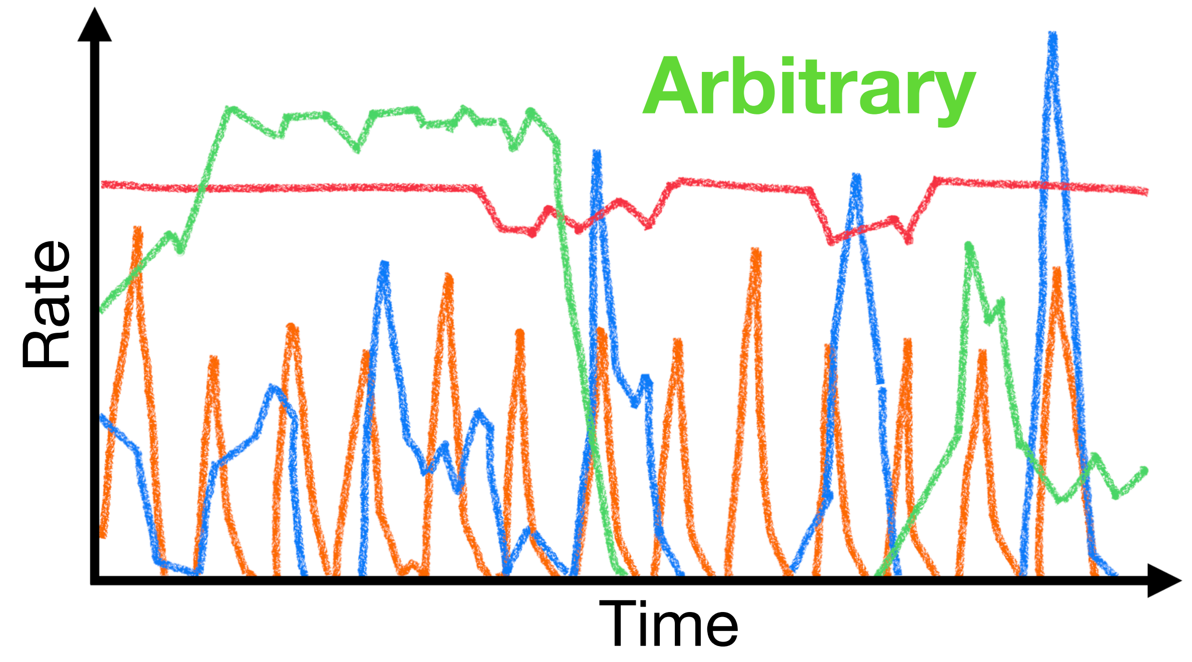
ResNet-50	Latency	Throughput	Cost
CPU	175 ms	6 req/s	$
GPU	2.8 ms	350 req/s	$$$

Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements



Requests arrive at different rates and regularity

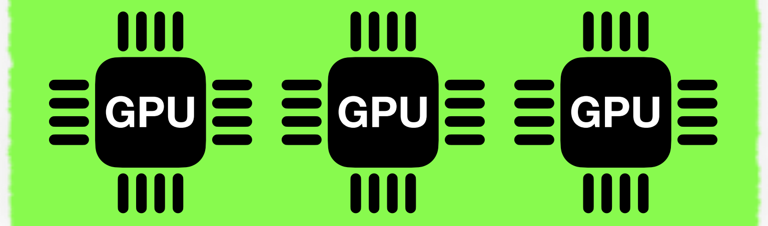


Each request has an inherent deadline

Latency SLOs

(e.g., 100ms)

HW accelerators are necessary!



ResNet-50	Latency	Throughput	Cost
CPU	175 ms	6 req/s	$
GPU	2.8 ms	350 req/s	$$$

Problem

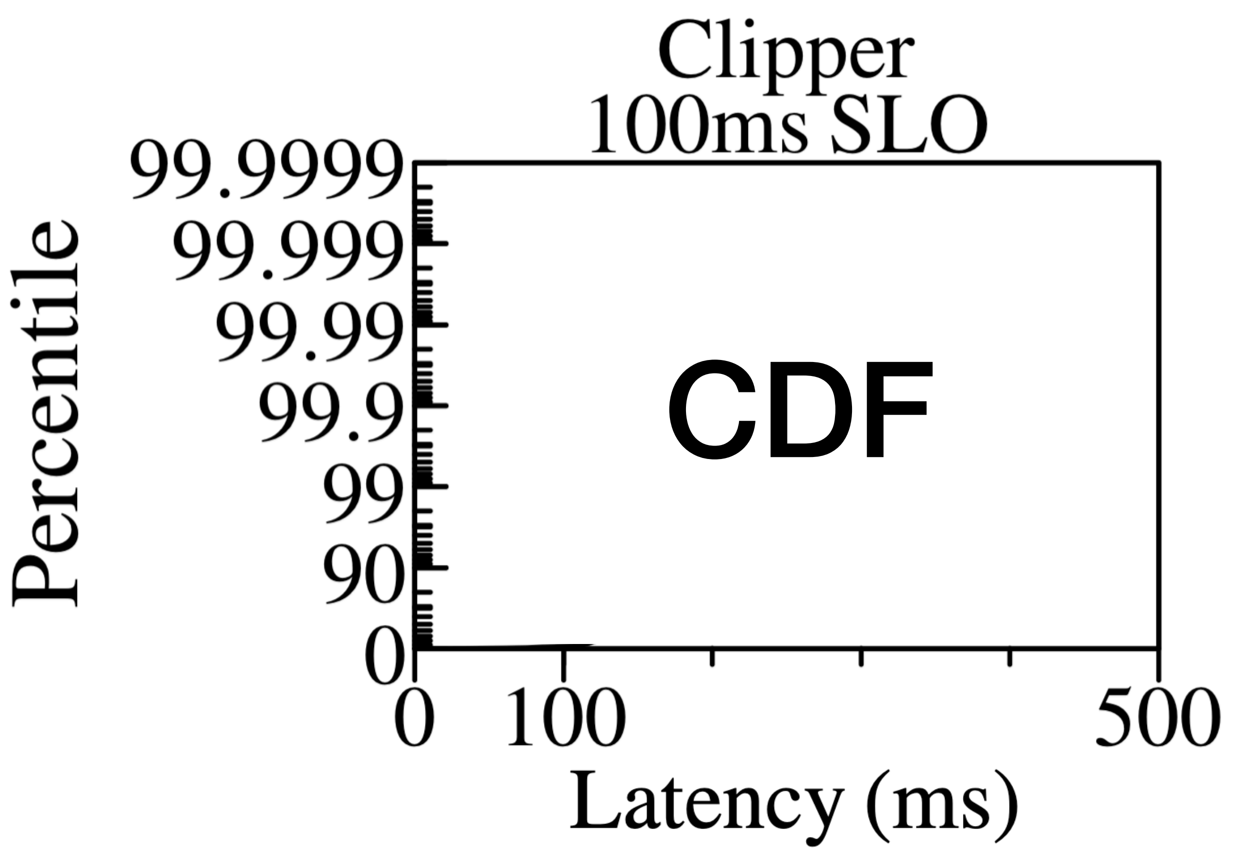
How can cloud providers efficiently share resources while meeting SLOs?

Existing Systems Incur Very High Tail Latency

Existing Systems Incur Very High Tail Latency

Inference latency

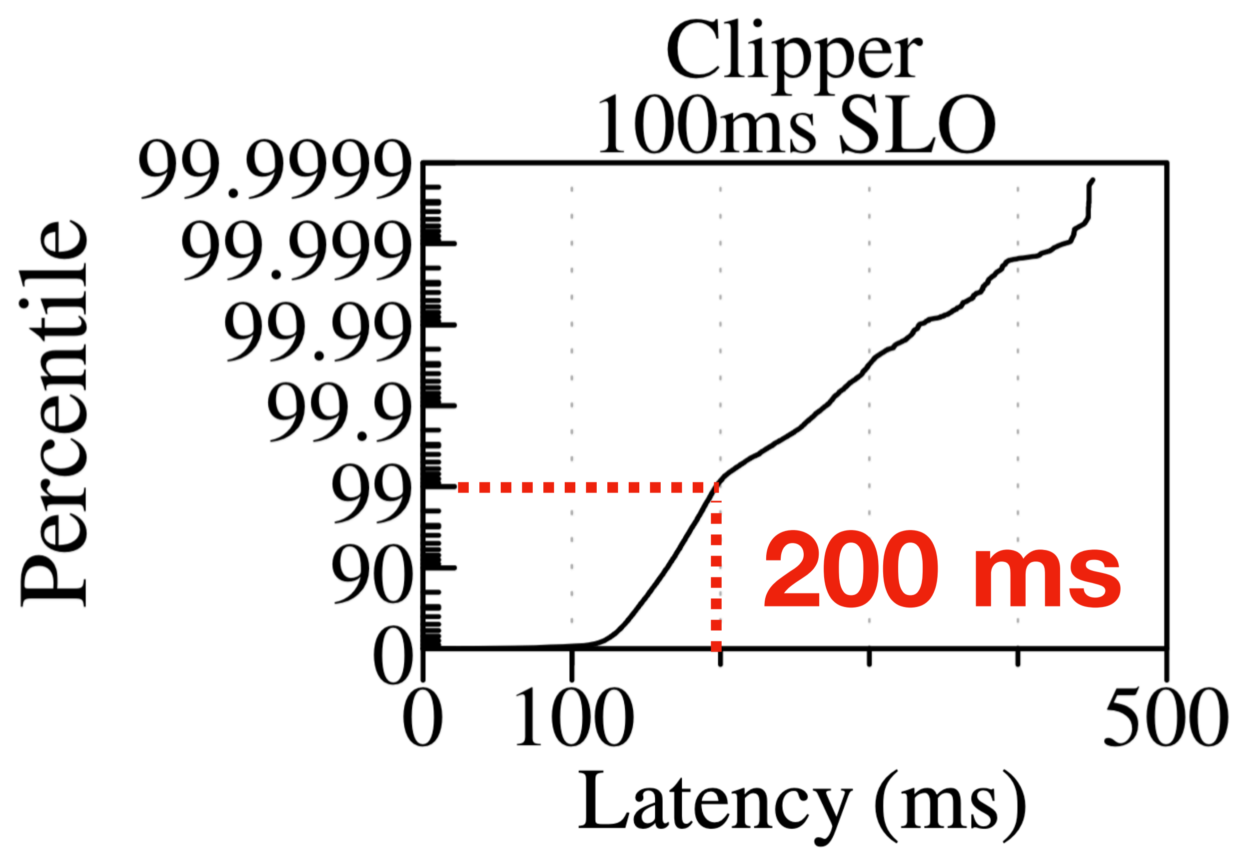
- **15 trained ResNet50**
- **Single GPU worker**
- **16 concurrent requests per model**



Existing Systems Incur Very High Tail Latency

Inference latency

- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

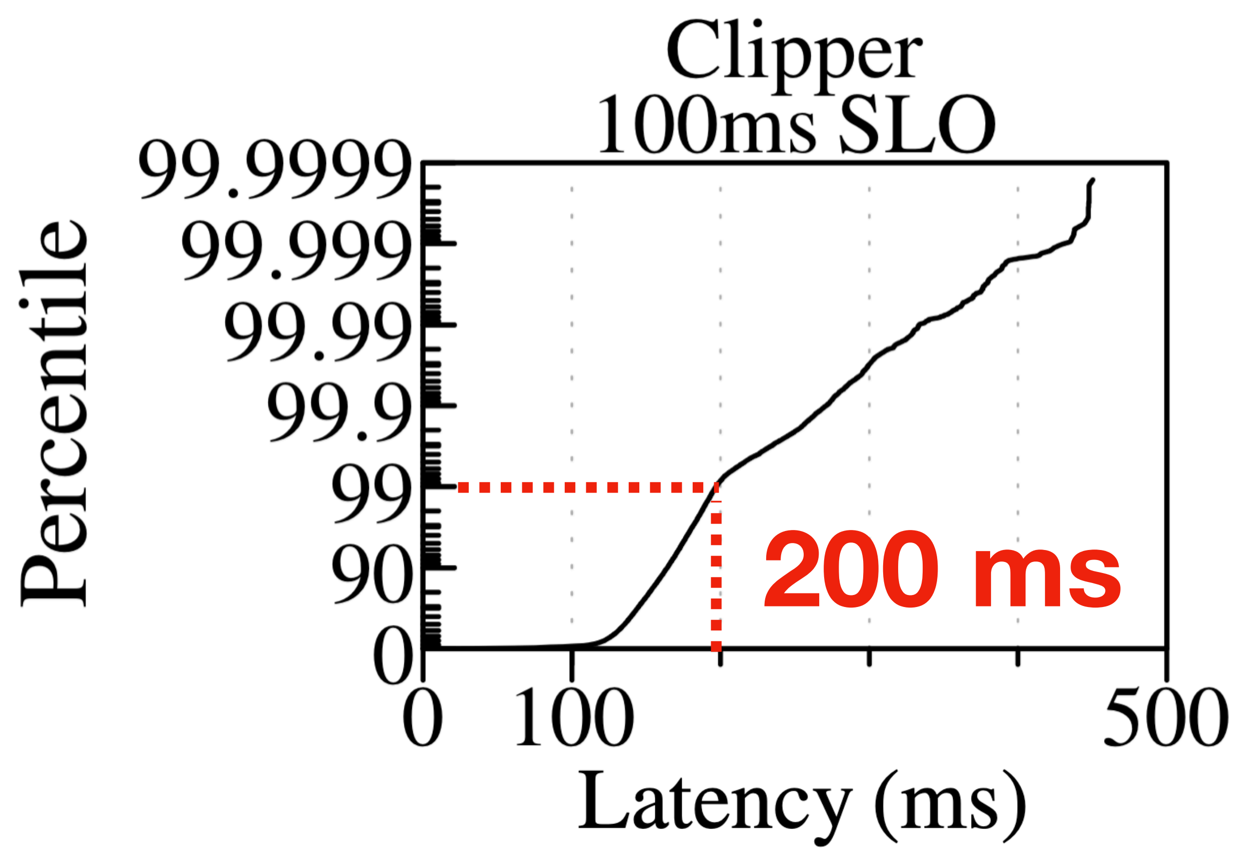


Tail latency >> SLO

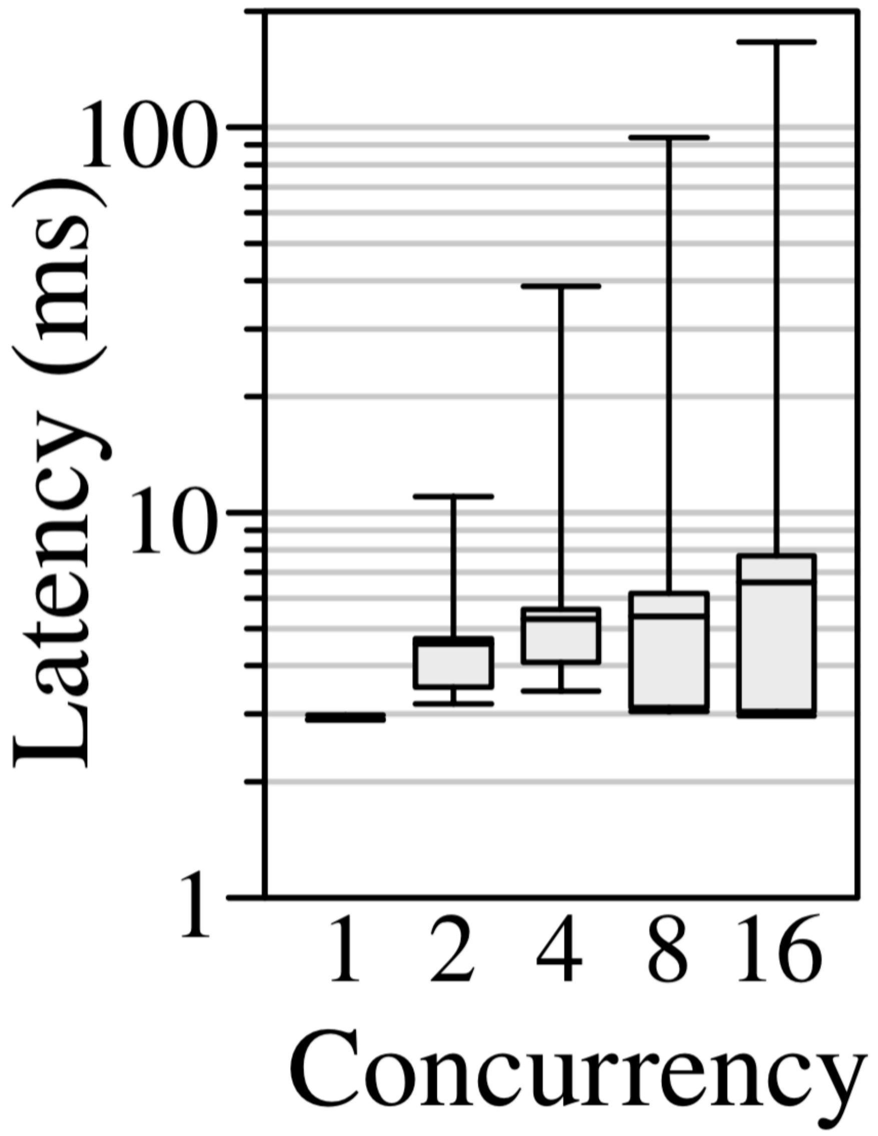
Existing Systems Incur Very High Tail Latency

Inference latency

- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model



Tail latency >> SLO

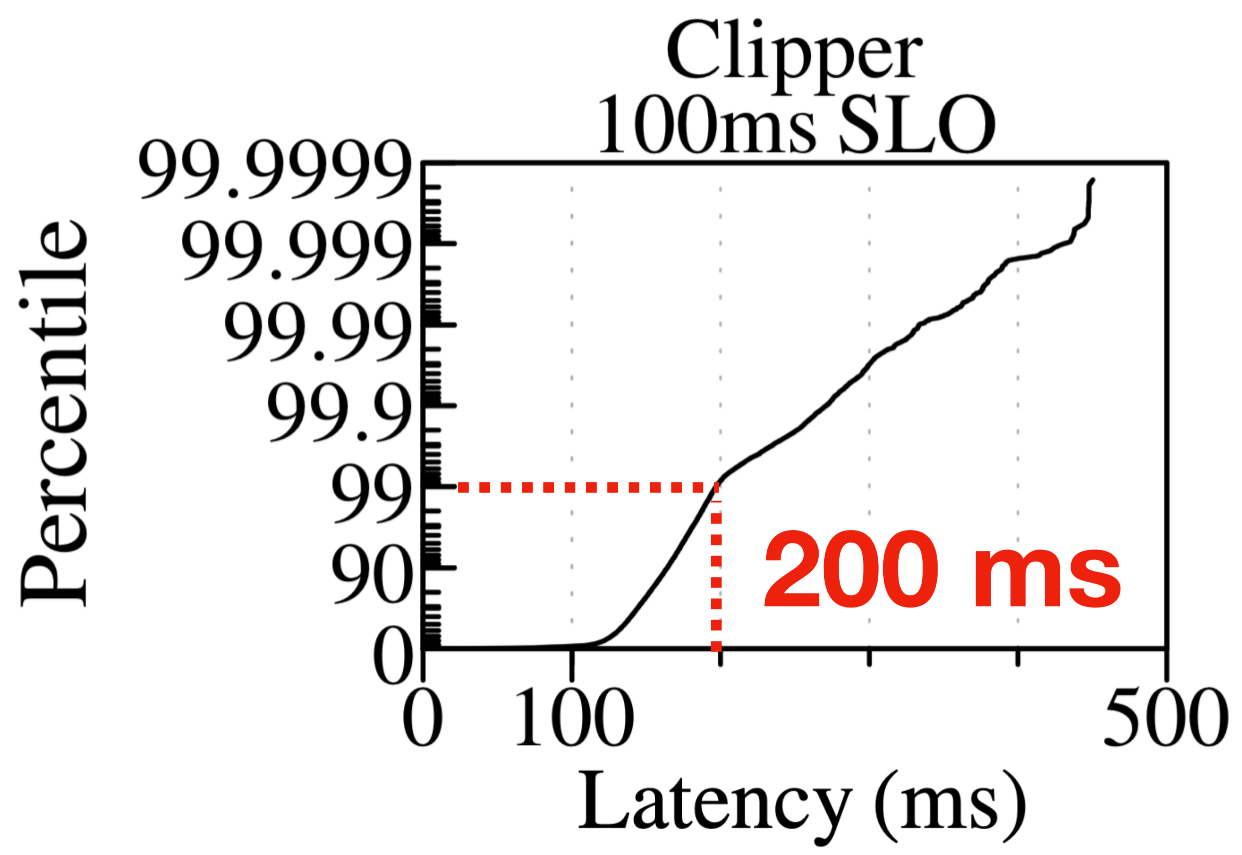


Concurrent DNN inference over GPU

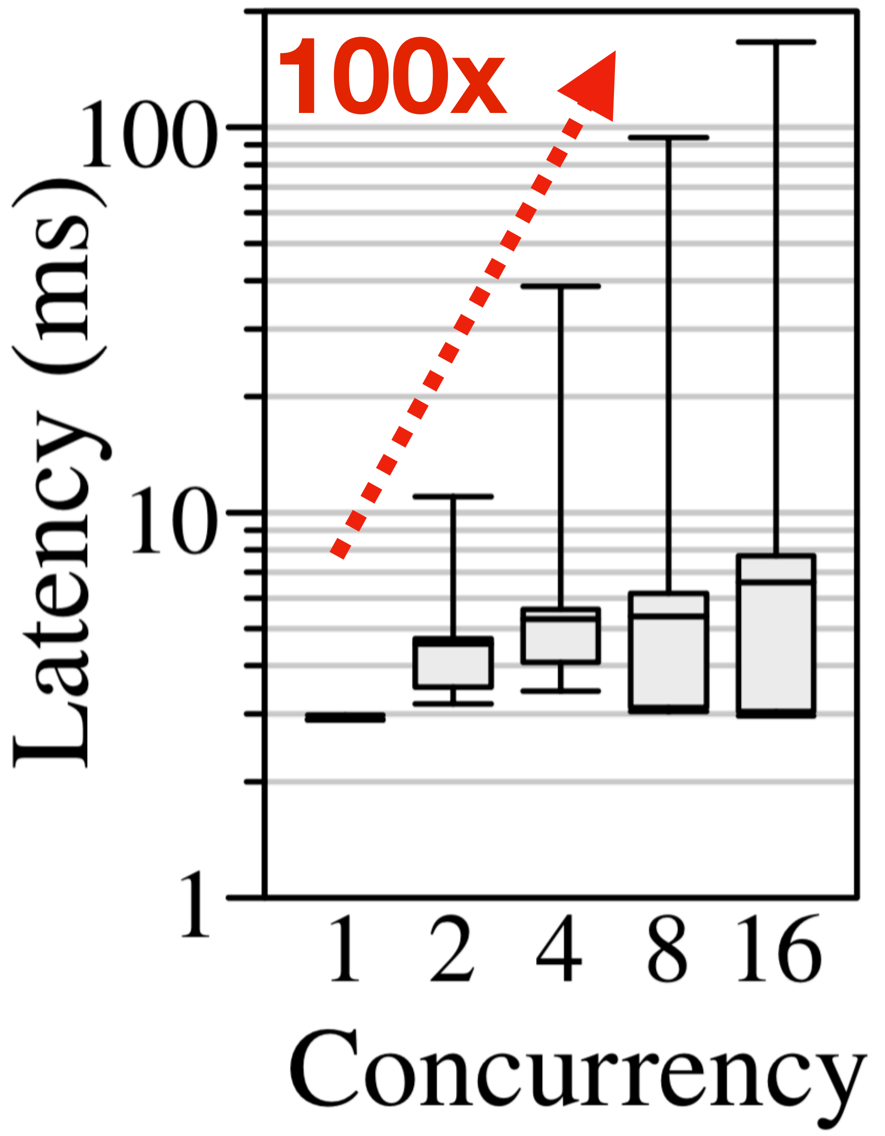
Existing Systems Incur Very High Tail Latency

Inference latency

- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model



Tail latency >> SLO



Concurrent DNN inference over GPU

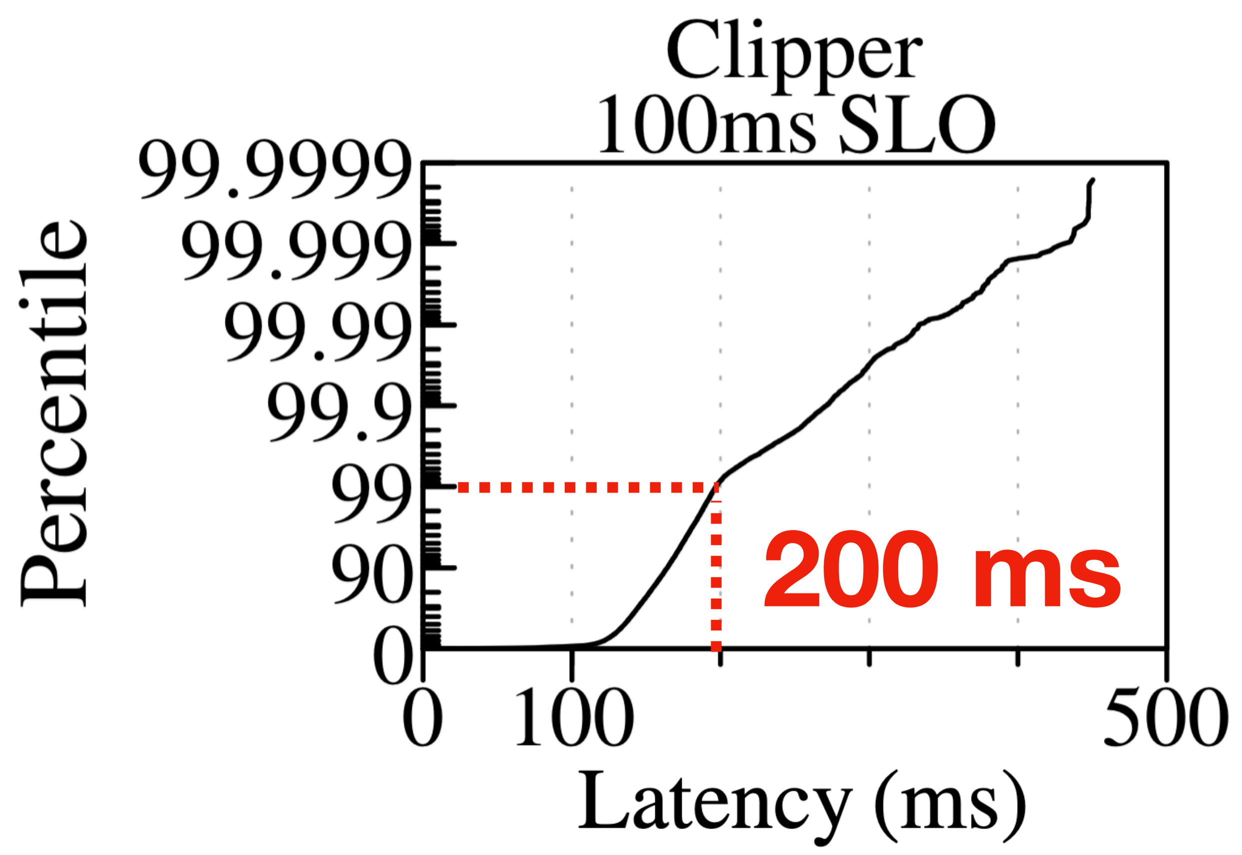
High variance in latency

Throughput gains only 25%

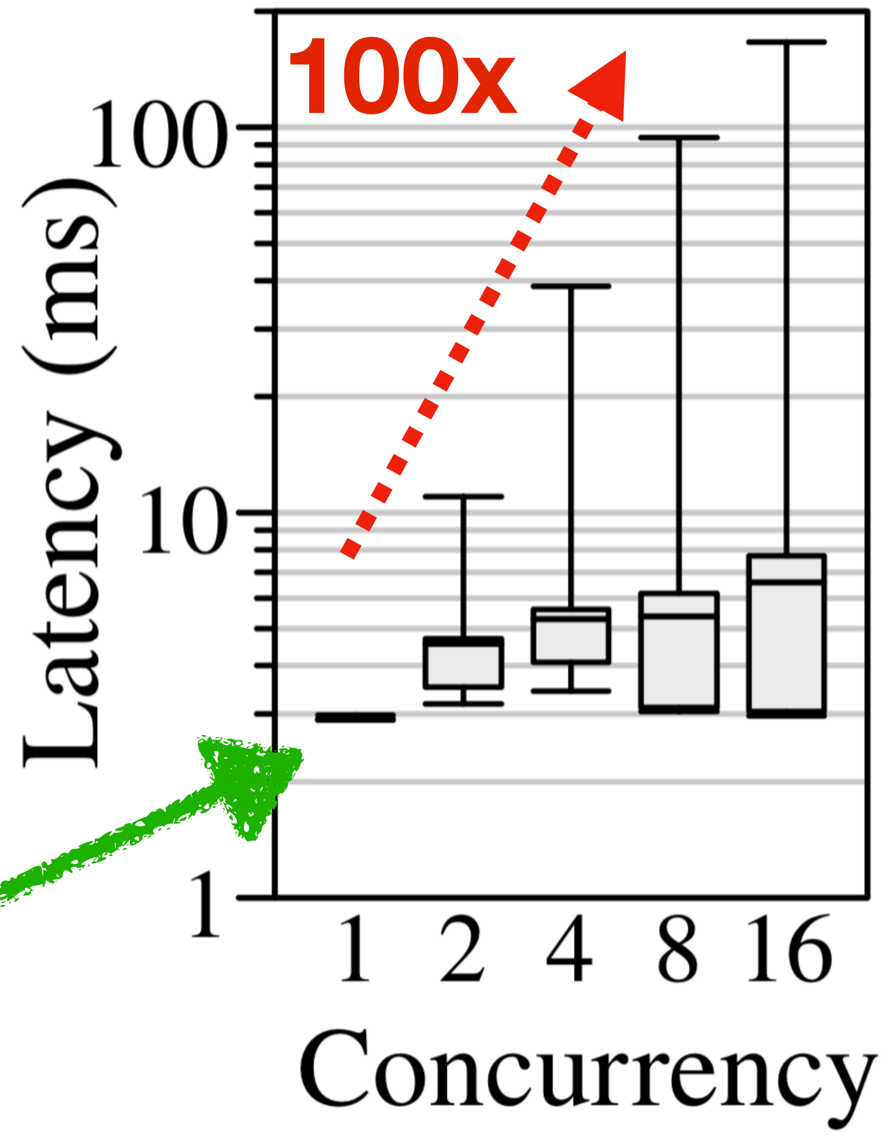
Existing Systems Incur Very High Tail Latency

Inference latency

- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model



Tail latency >> SLO



Concurrent DNN inference over GPU

High variance in latency

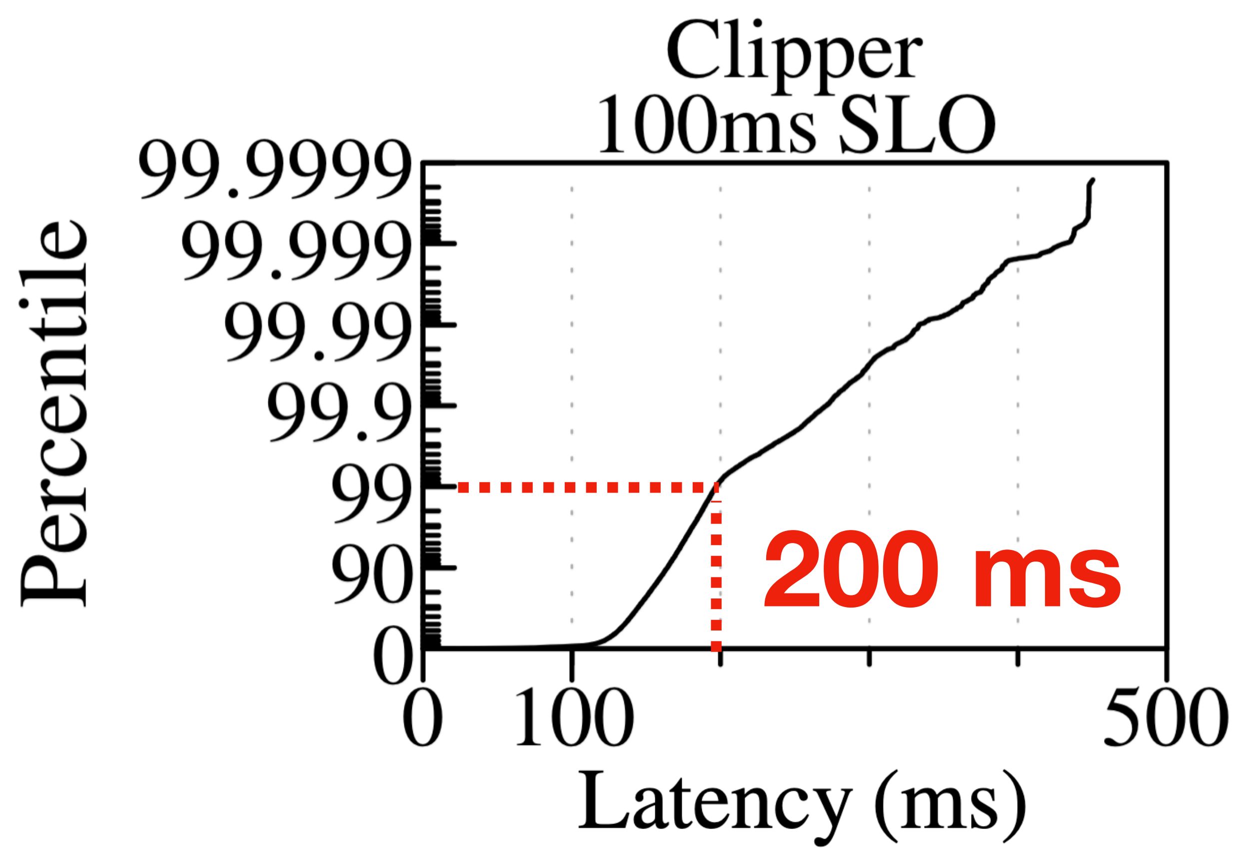
Throughput gains only 25%

Single-thread latency is extremely predictable

Existing Systems Incur Very High Tail Latency

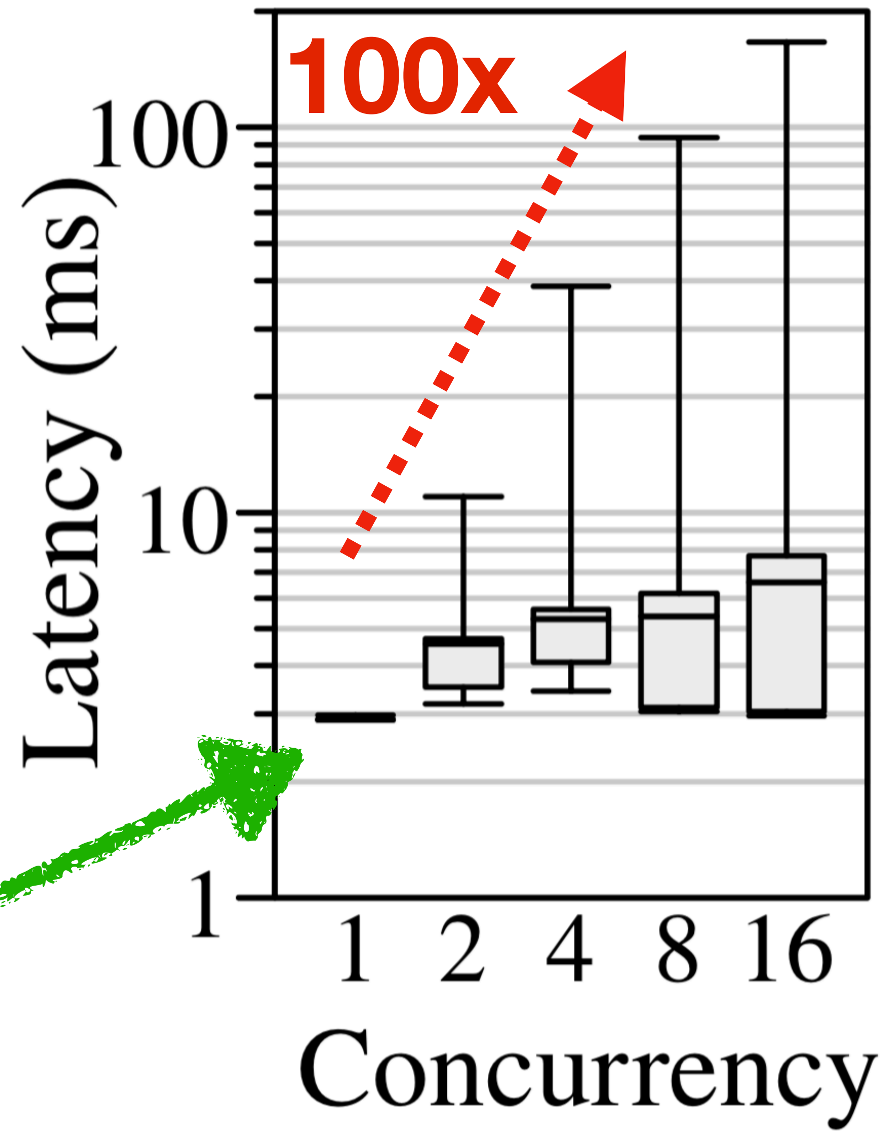
Inference latency

- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model



Tail latency >> SLO

Preserves DNN predictability at every stage of model serving



Concurrent DNN inference over GPU

High variance in latency

Throughput gains only 25%

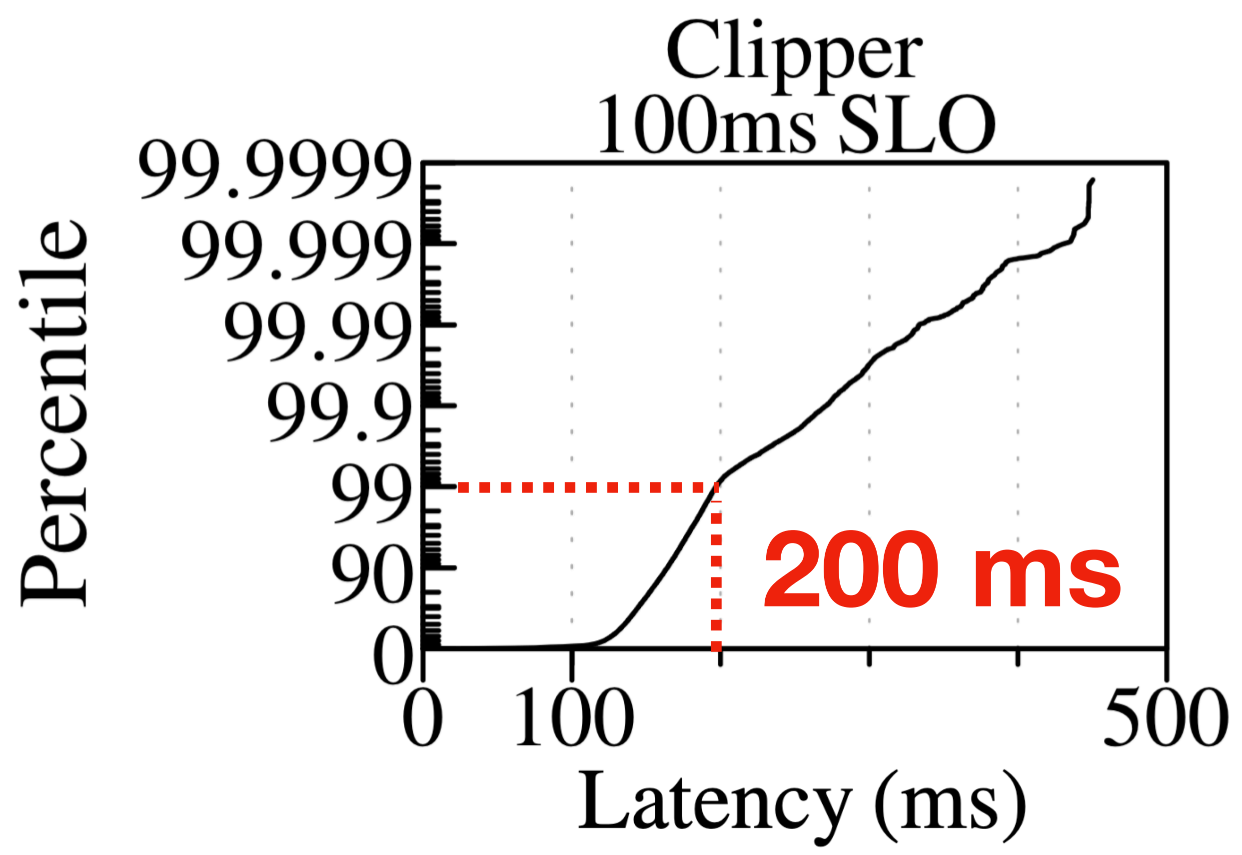
Clockwork adopts a contrasting approach!

Single-thread latency is extremely predictable

Existing Systems Incur Very High Tail Latency

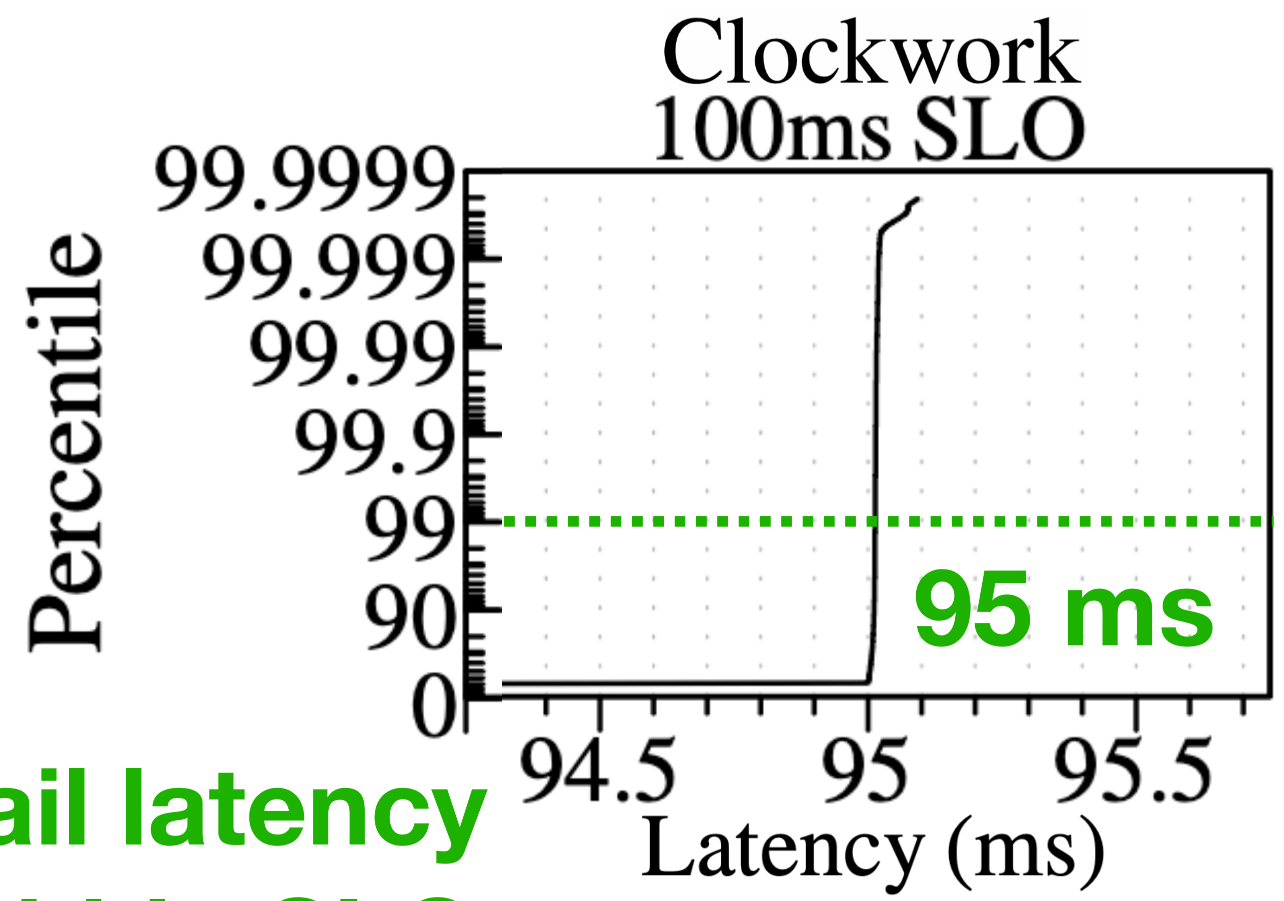
Inference latency

- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

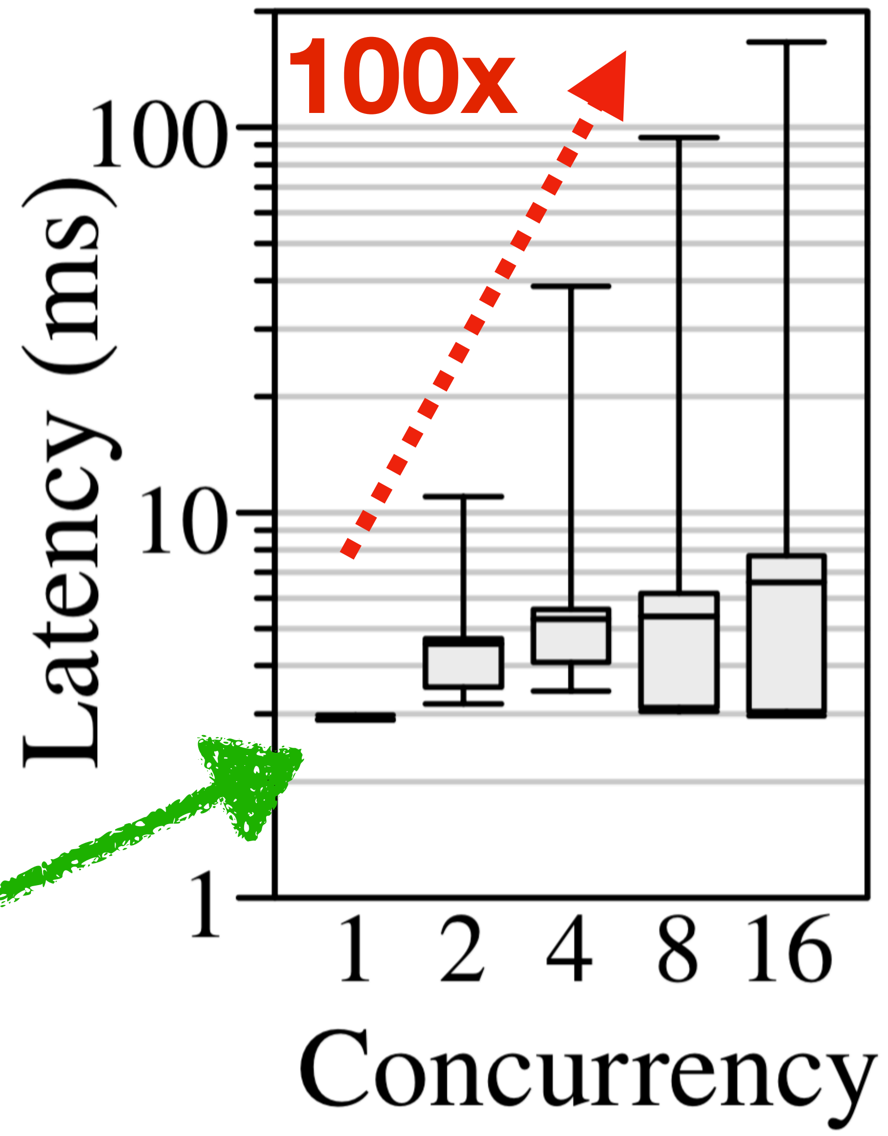


Tail latency >> SLO

Preserves DNN predictability at every stage of model serving



Tail latency within SLO



Concurrent DNN inference over GPU

High variance in latency

Throughput gains only 25%

Clockwork adopts a contrasting approach!

Single-thread latency is extremely predictable

How does Clockwork Achieve End-to-End Predictability?

Design Principles

Goal: 1000s of models, many users, limited resources

Design Principles

Goal: 1000s of models, many users, limited resources

Maximize sharing

Design Principles

Goal: 1000s of models, many users, limited resources

Maximize sharing

1. Predictable worker with no choices

Design Principles

Goal: 1000s of models, many users, limited resources

Maximize sharing

1. Predictable worker with no choices

2. Consolidating choices at a central controller

Design Principles

Goal: 1000s of models, many users, limited resources

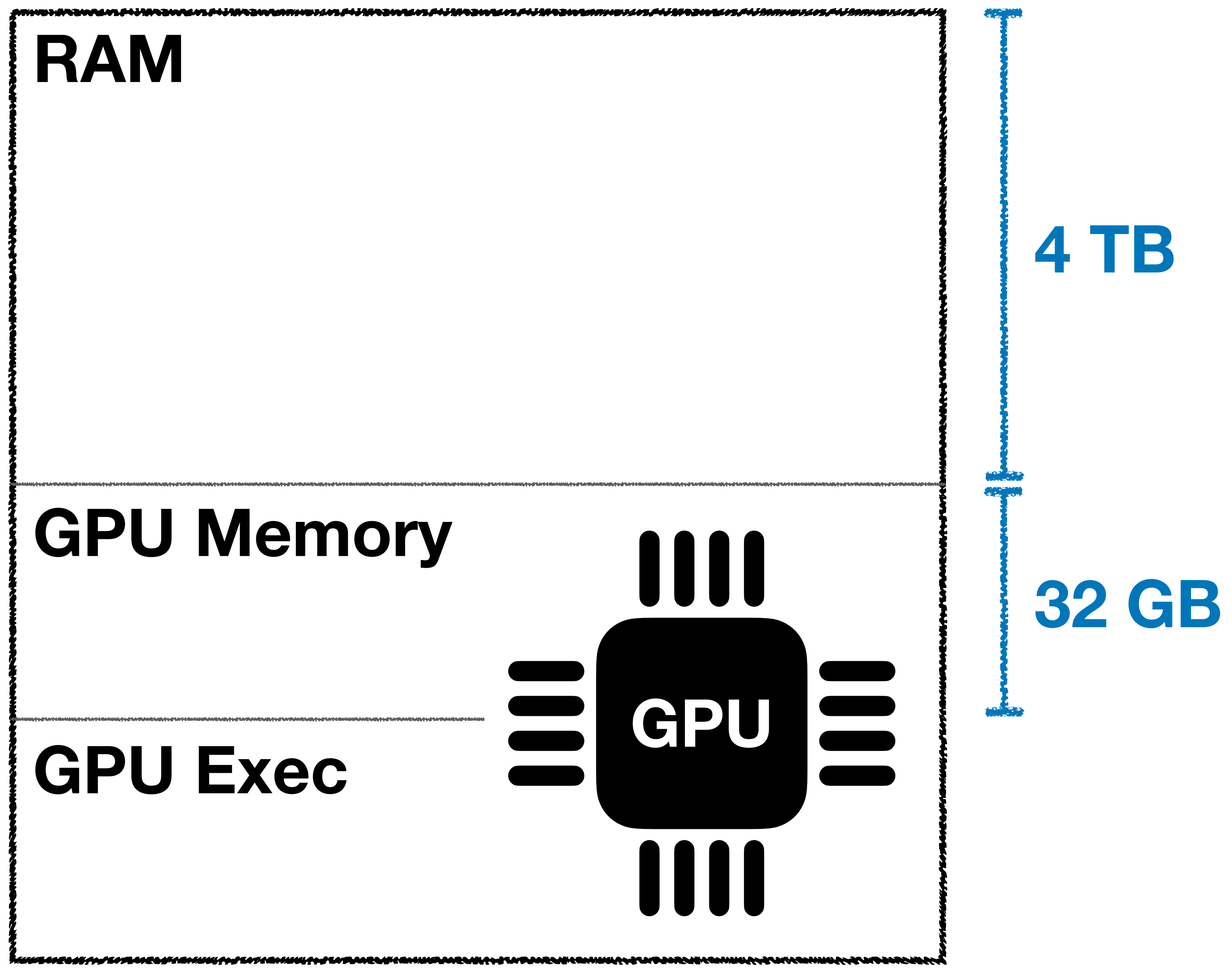
Maximize sharing

1. Predictable worker with no choices

2. Consolidating choices at a central controller

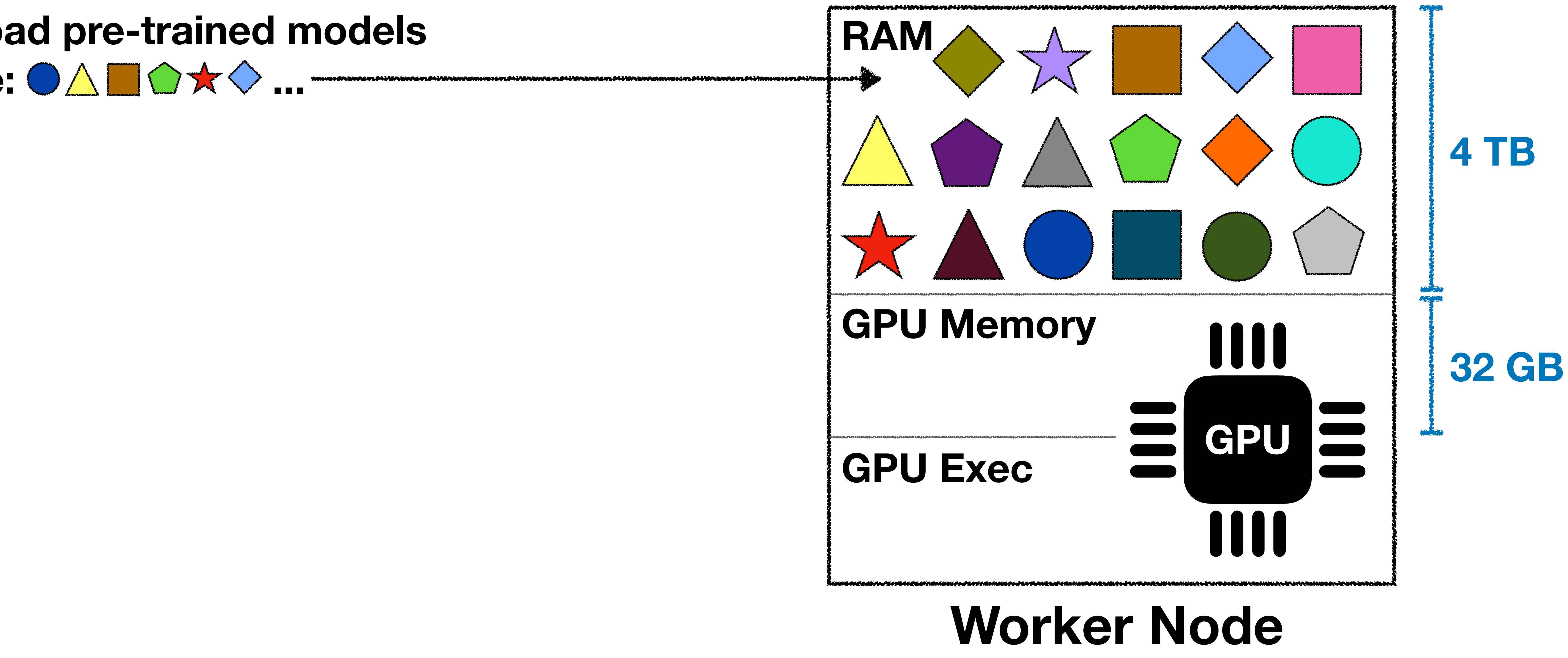
3. Deadline-aware scheduling for SLO compliance

Designing a Predictable Worker (1/2)

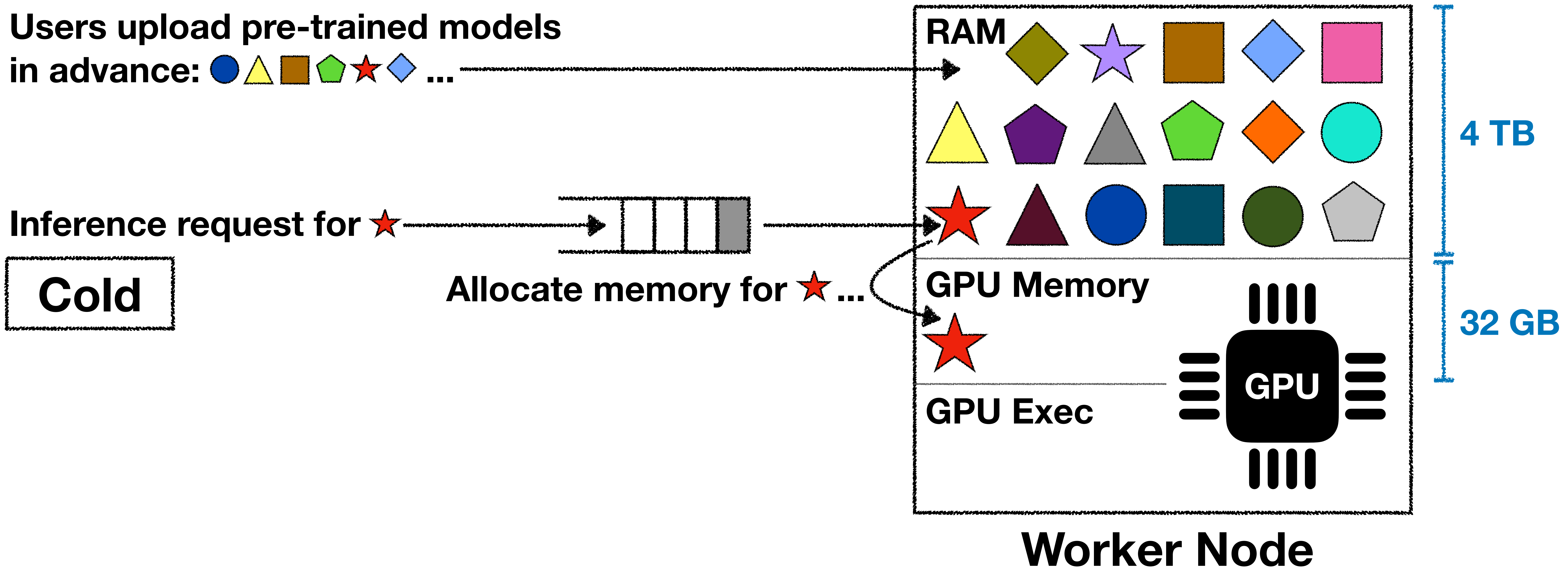


Worker Node

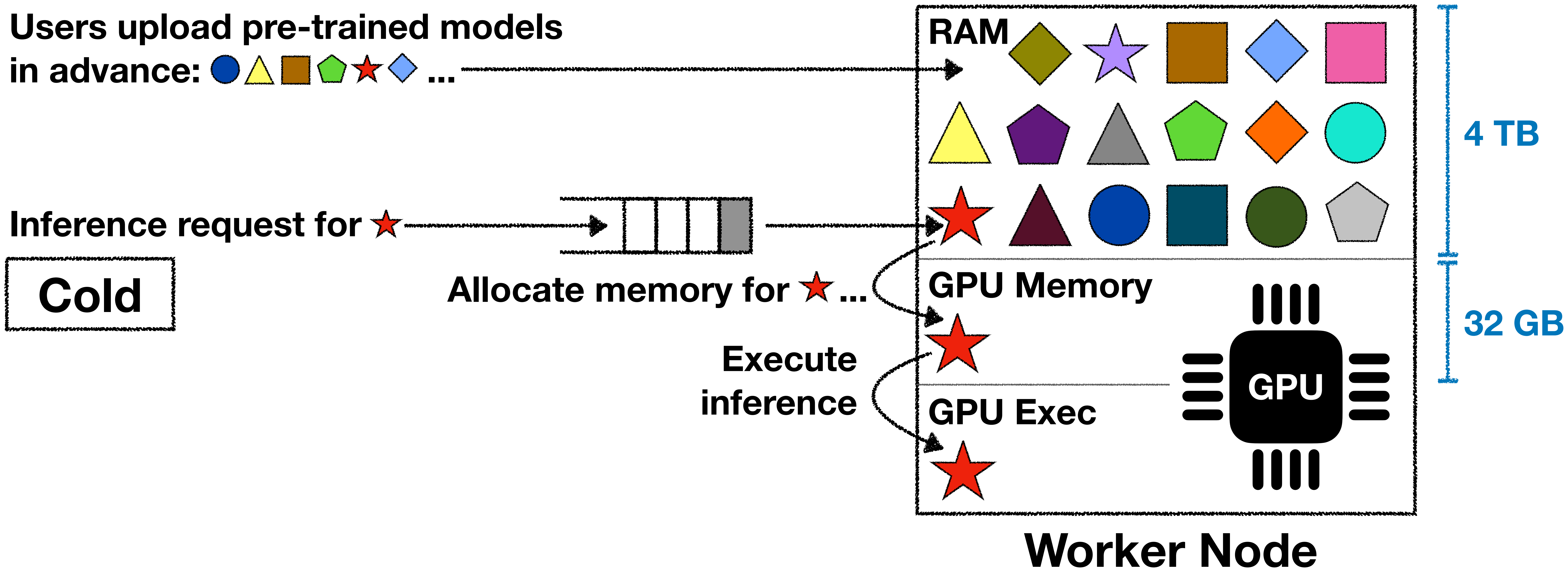
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: 

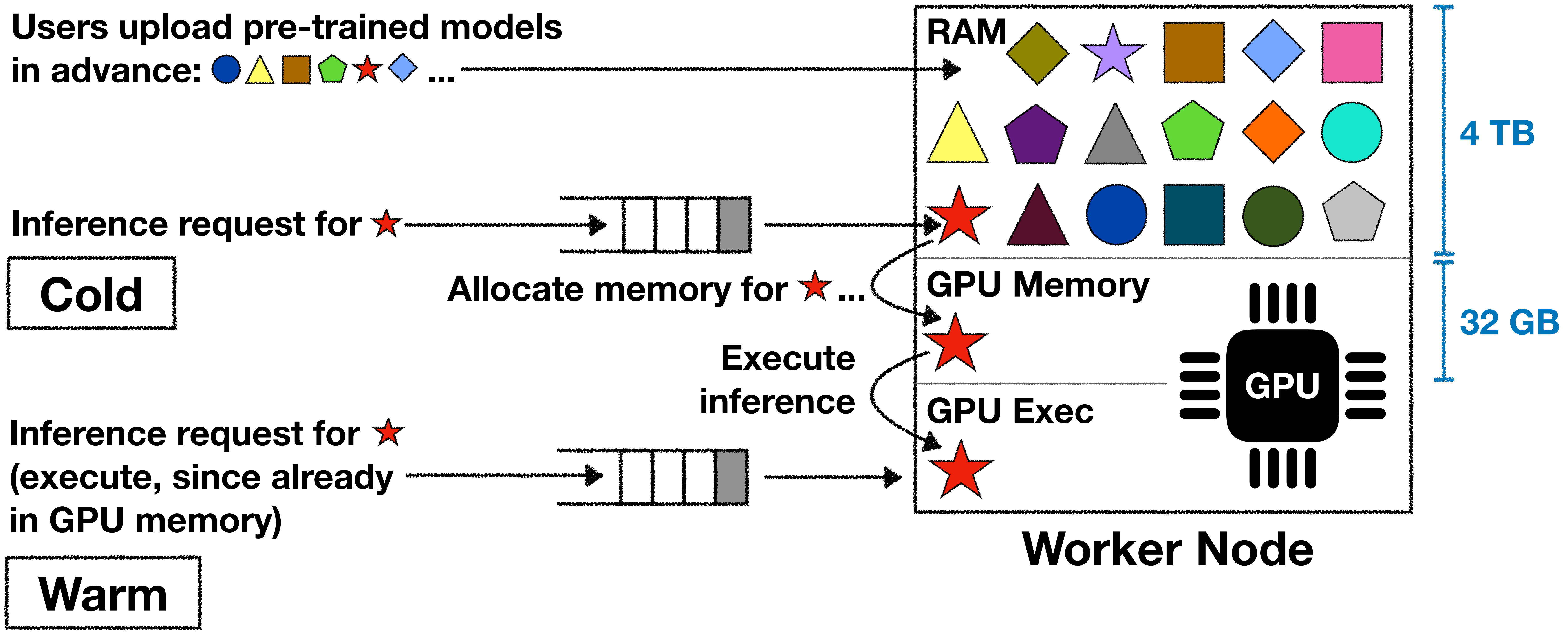
Designing a Predictable Worker (1/2)



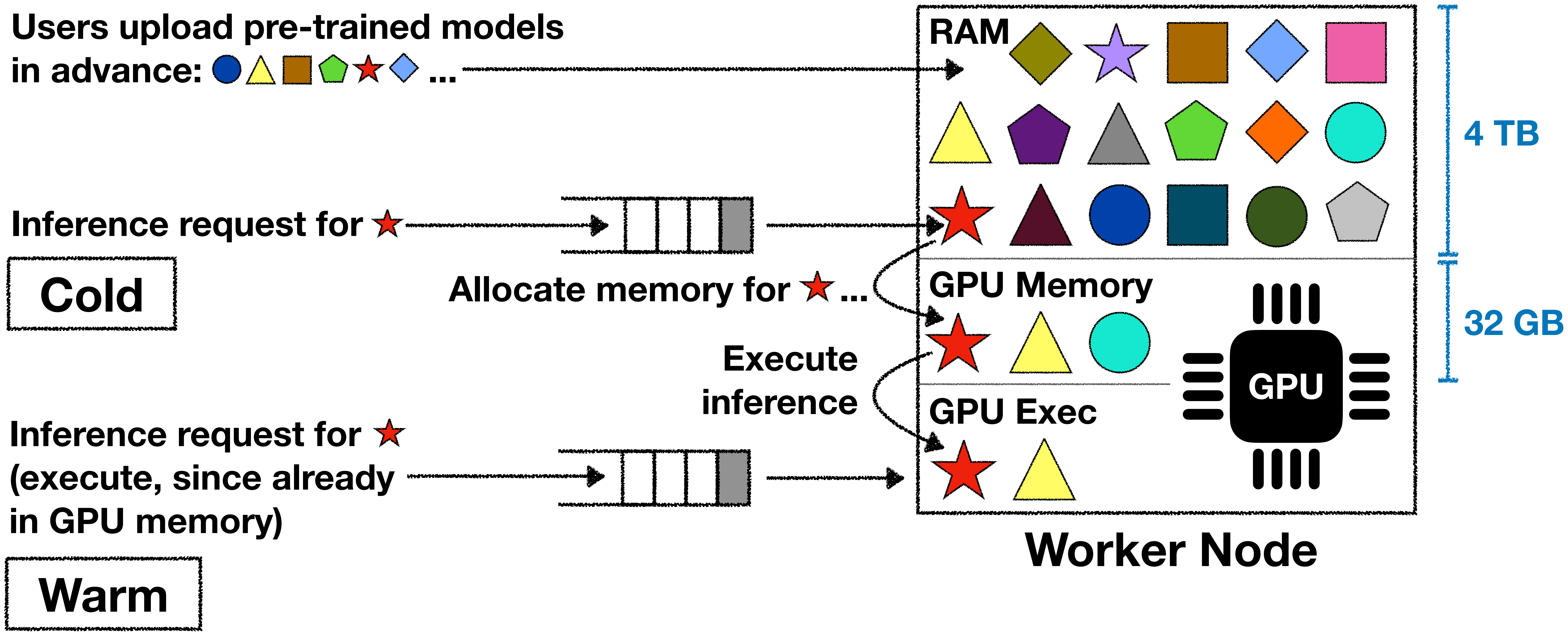
Designing a Predictable Worker (1/2)



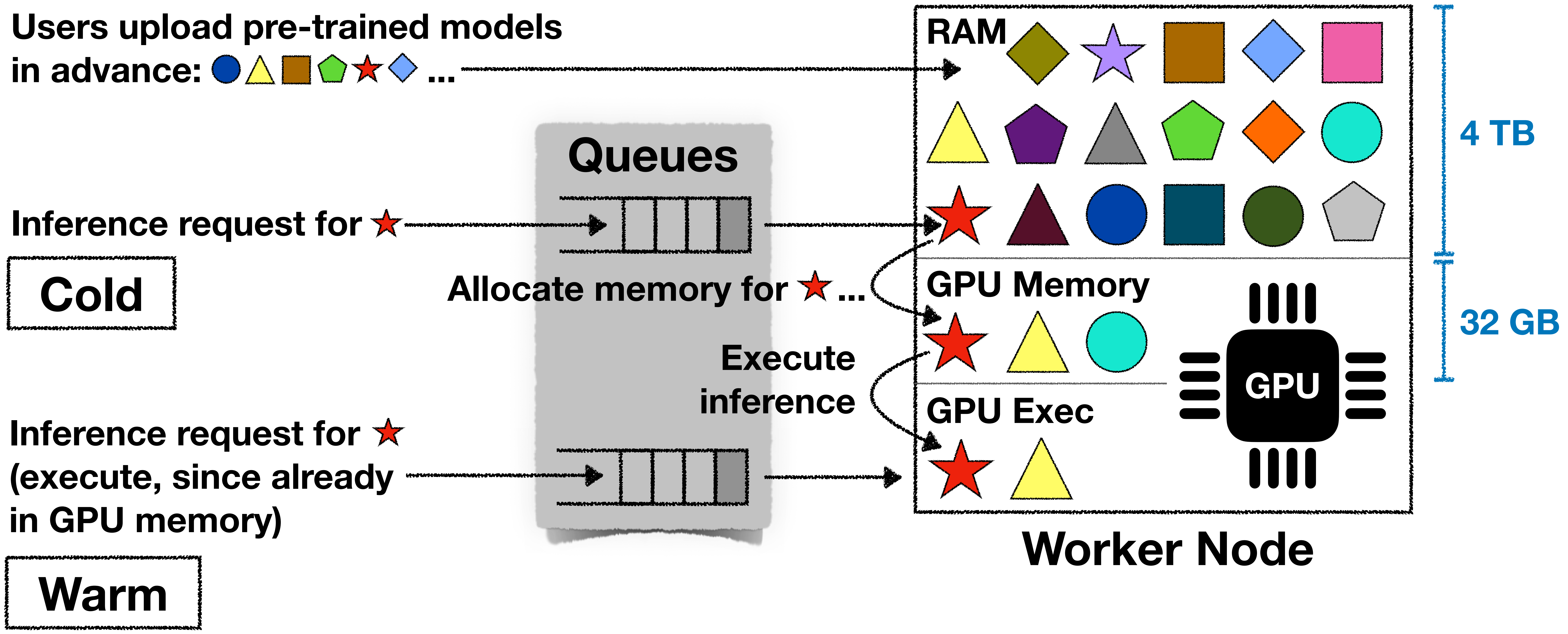
Designing a Predictable Worker (1/2)



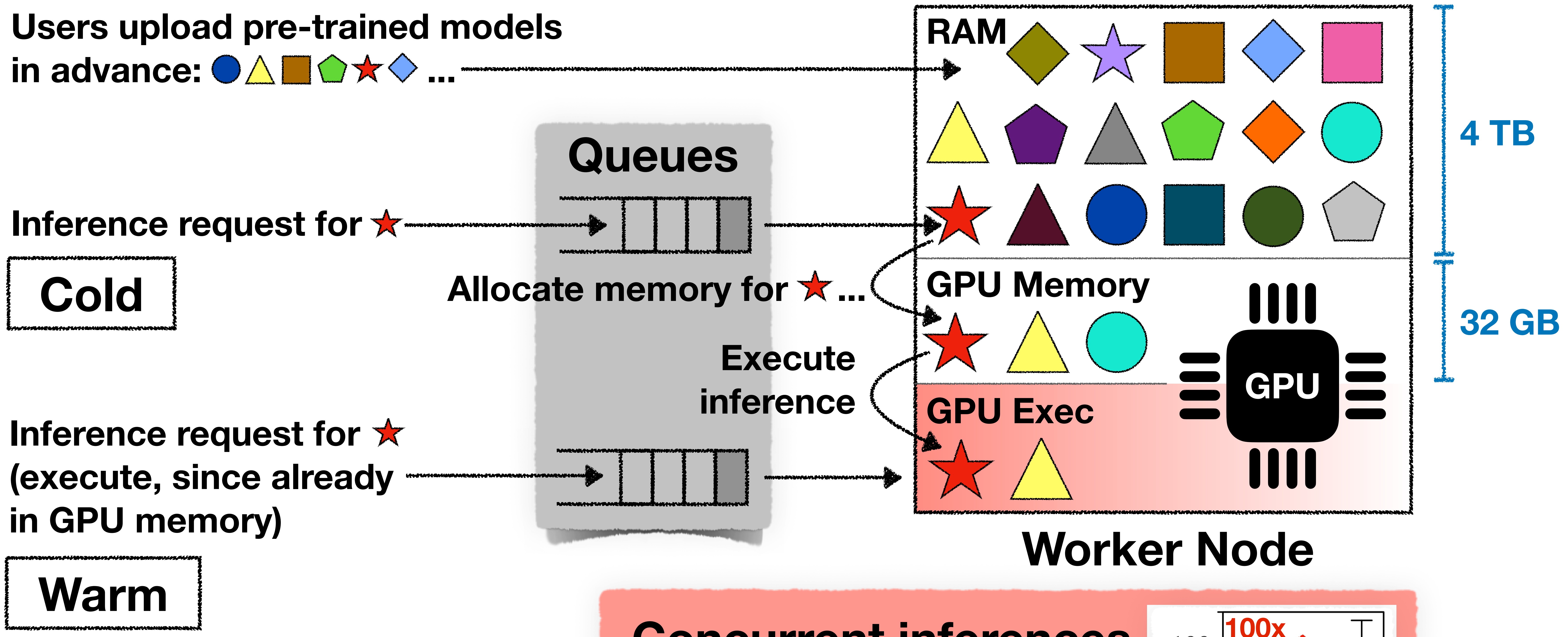
Designing a Predictable Worker (1/2)



Designing a Predictable Worker (1/2)



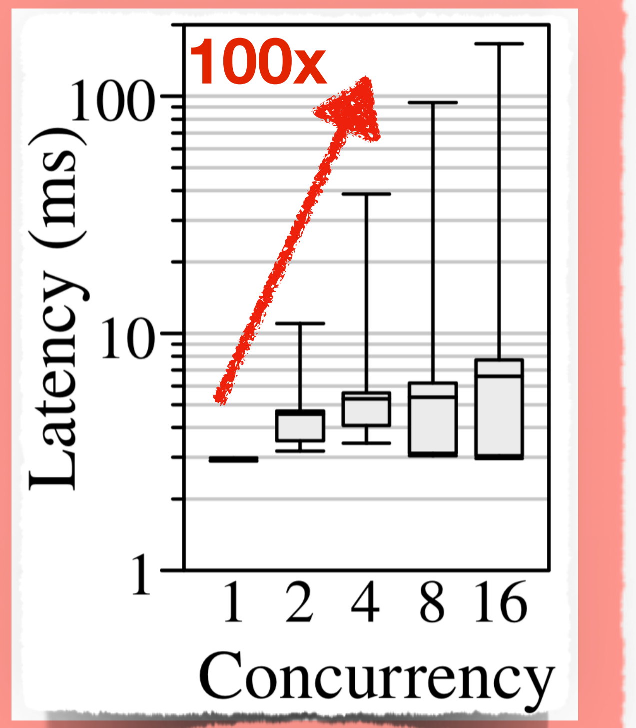
Designing a Predictable Worker (1/2)



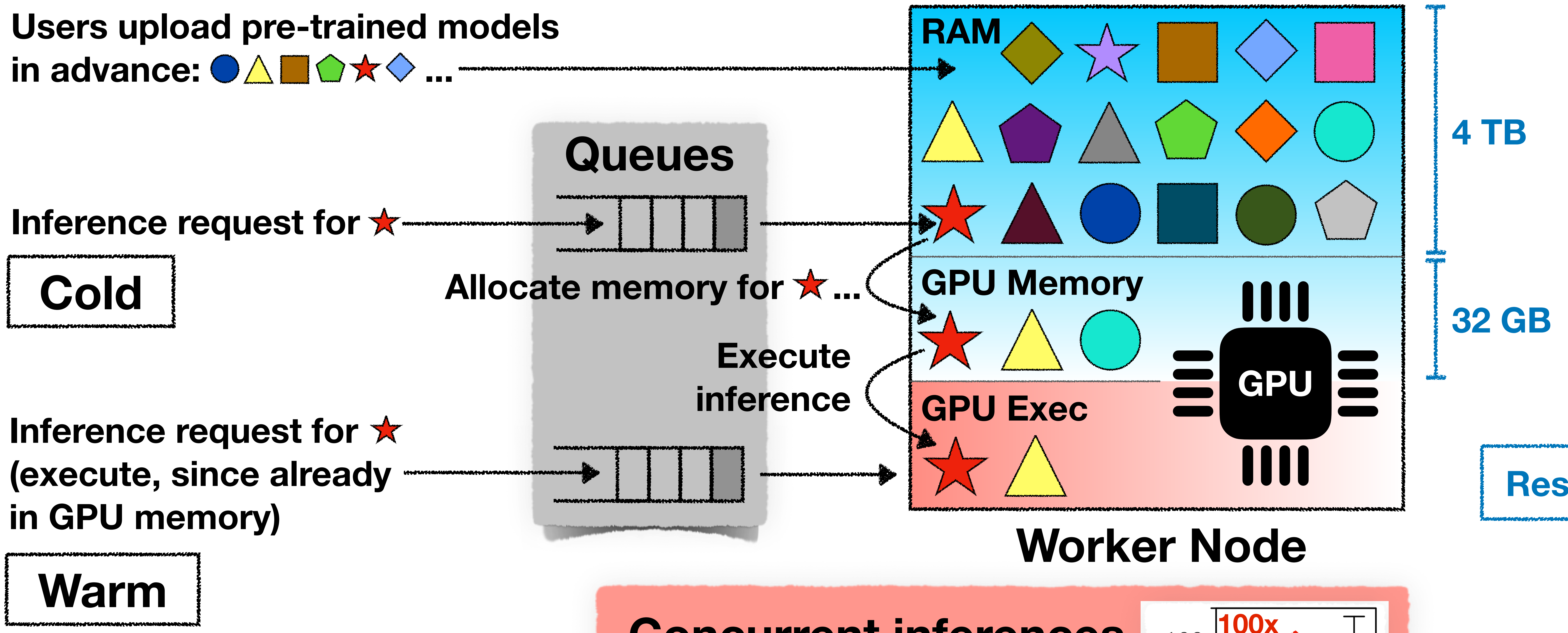
Concurrent inferences

+ Proprietary & undocumented policies

→ Unpredictable response times



Designing a Predictable Worker (1/2)



Managed memory can be unpredictable

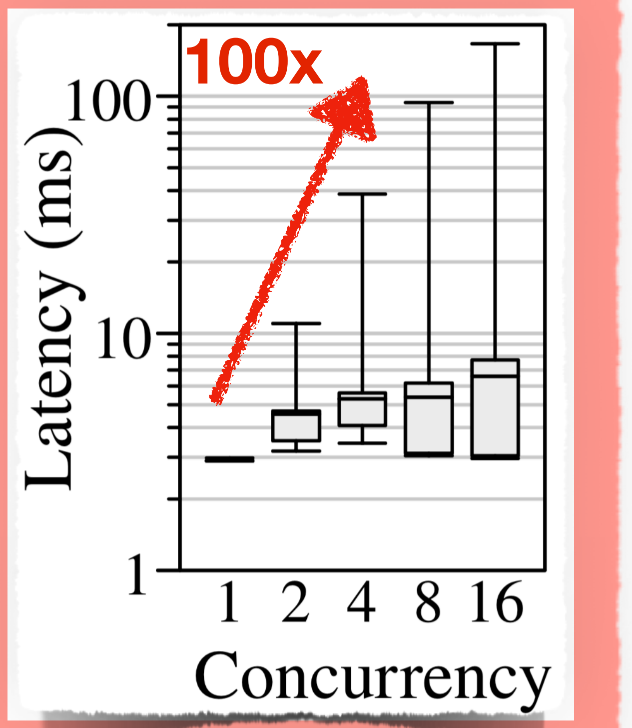
- GPU memory (cache) hits & misses

ResNet-50 — Hit: 2.3 ms | Miss: 10.6 ms

Concurrent inferences

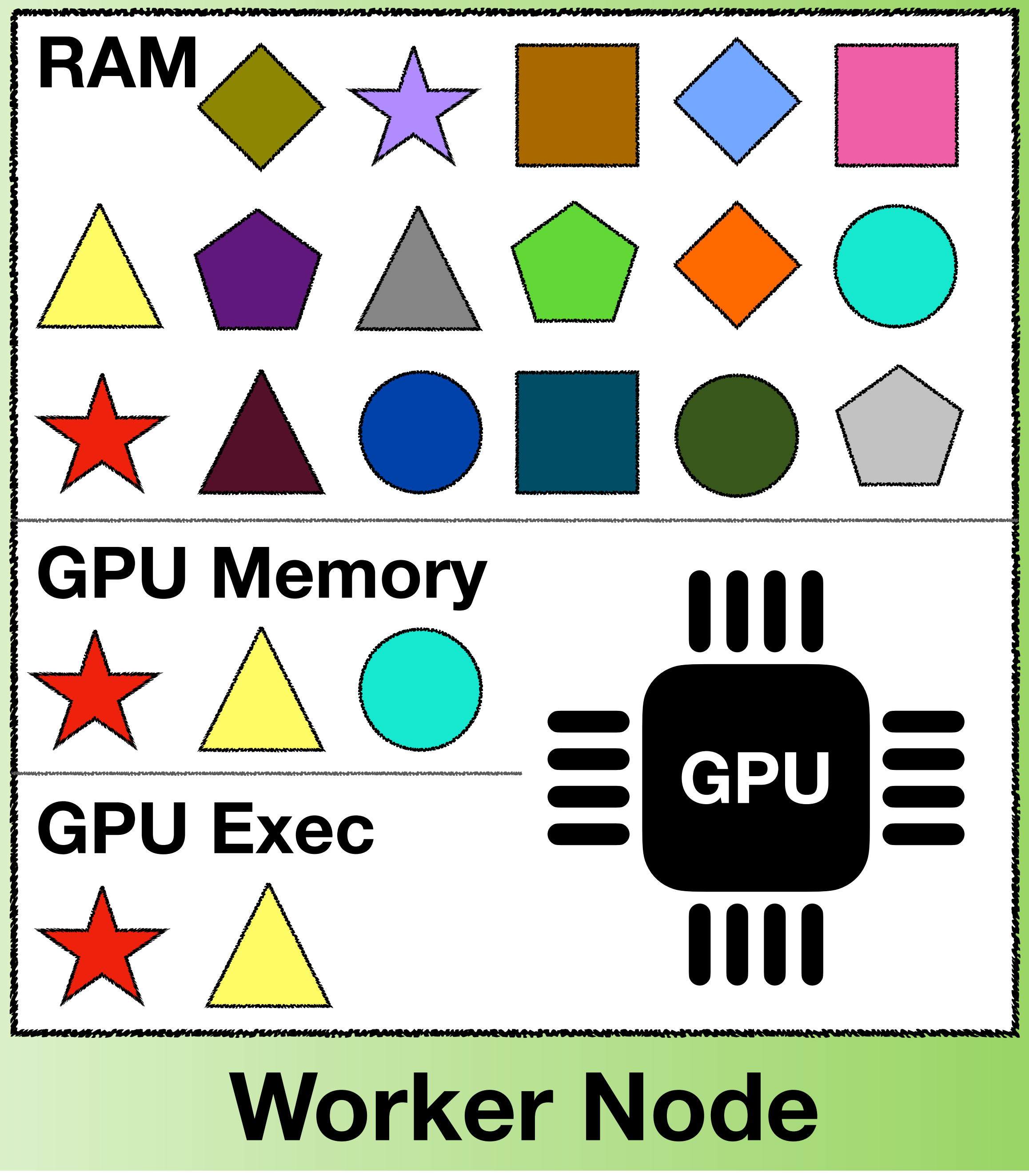
+ Proprietary & undocumented policies

→ Unpredictable response times

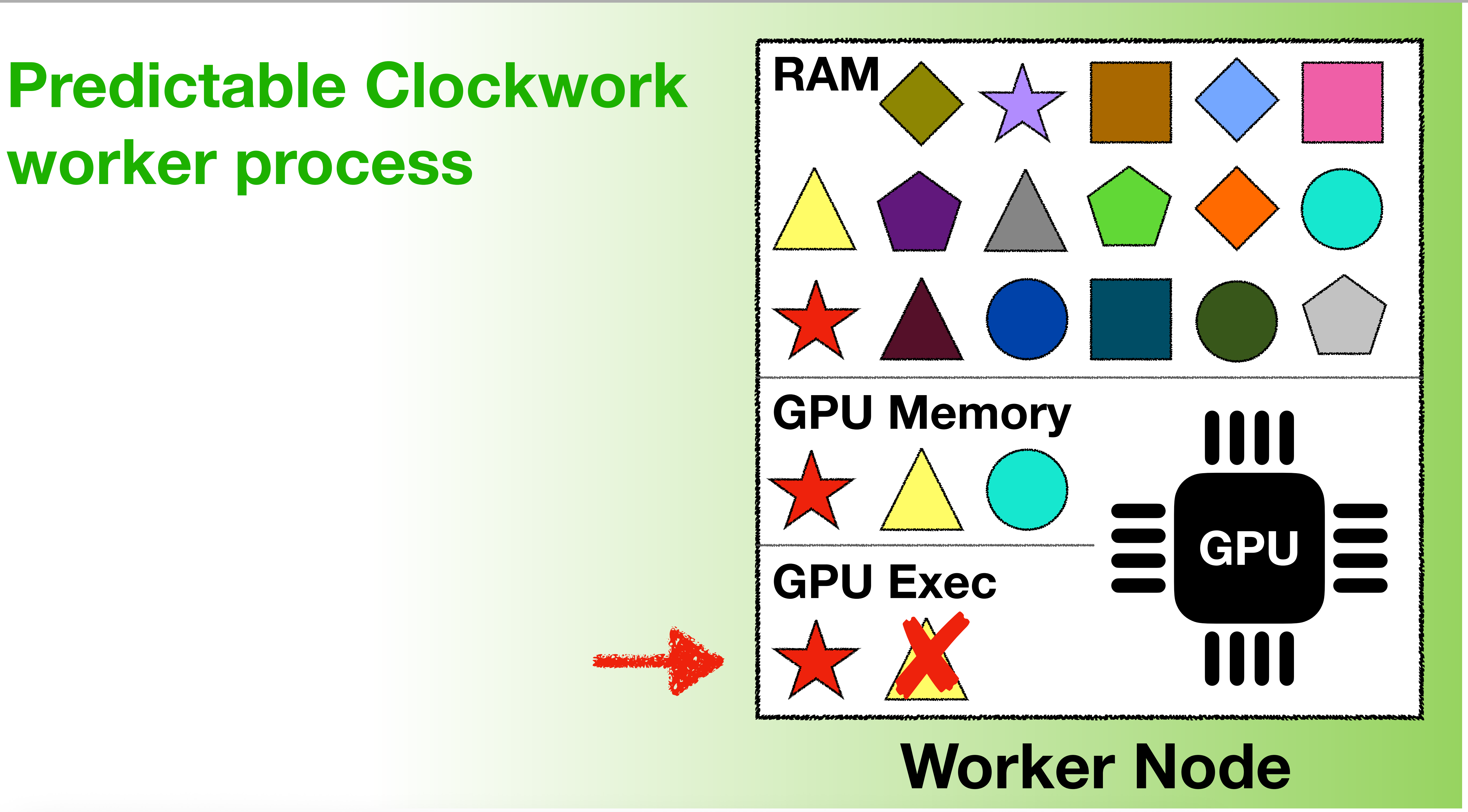


Designing a Predictable Worker (2/2)

Predictable Clockwork worker process



Designing a Predictable Worker (2/2)



Concurrent inferences

+ Proprietary & undocumented policies

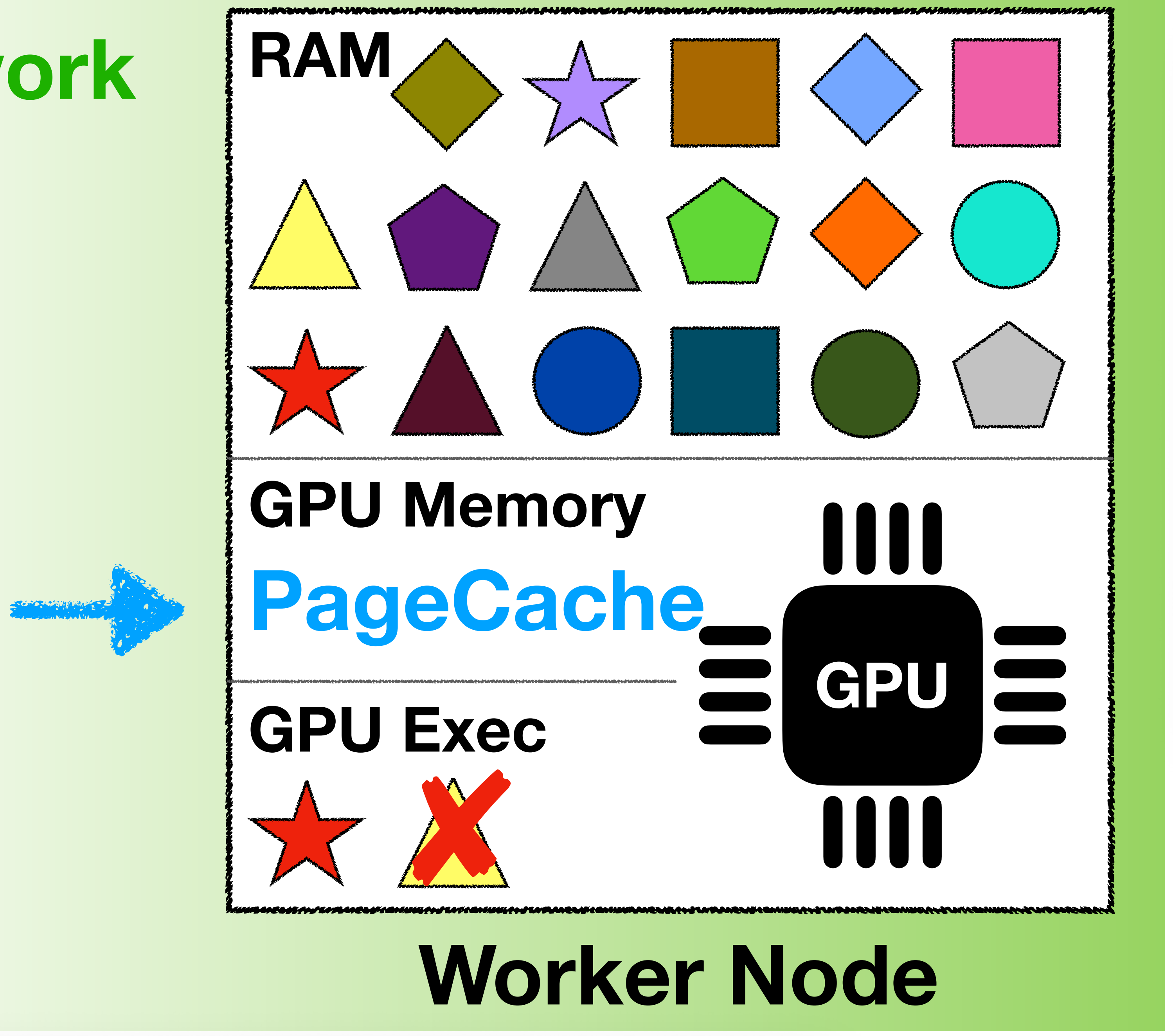
→ Unpredictable response times

Solution

Execute inference one at a time

Designing a Predictable Worker (2/2)

Predictable Clockwork worker process



Managed memory can be unpredictable

Solution

Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Concurrent inferences

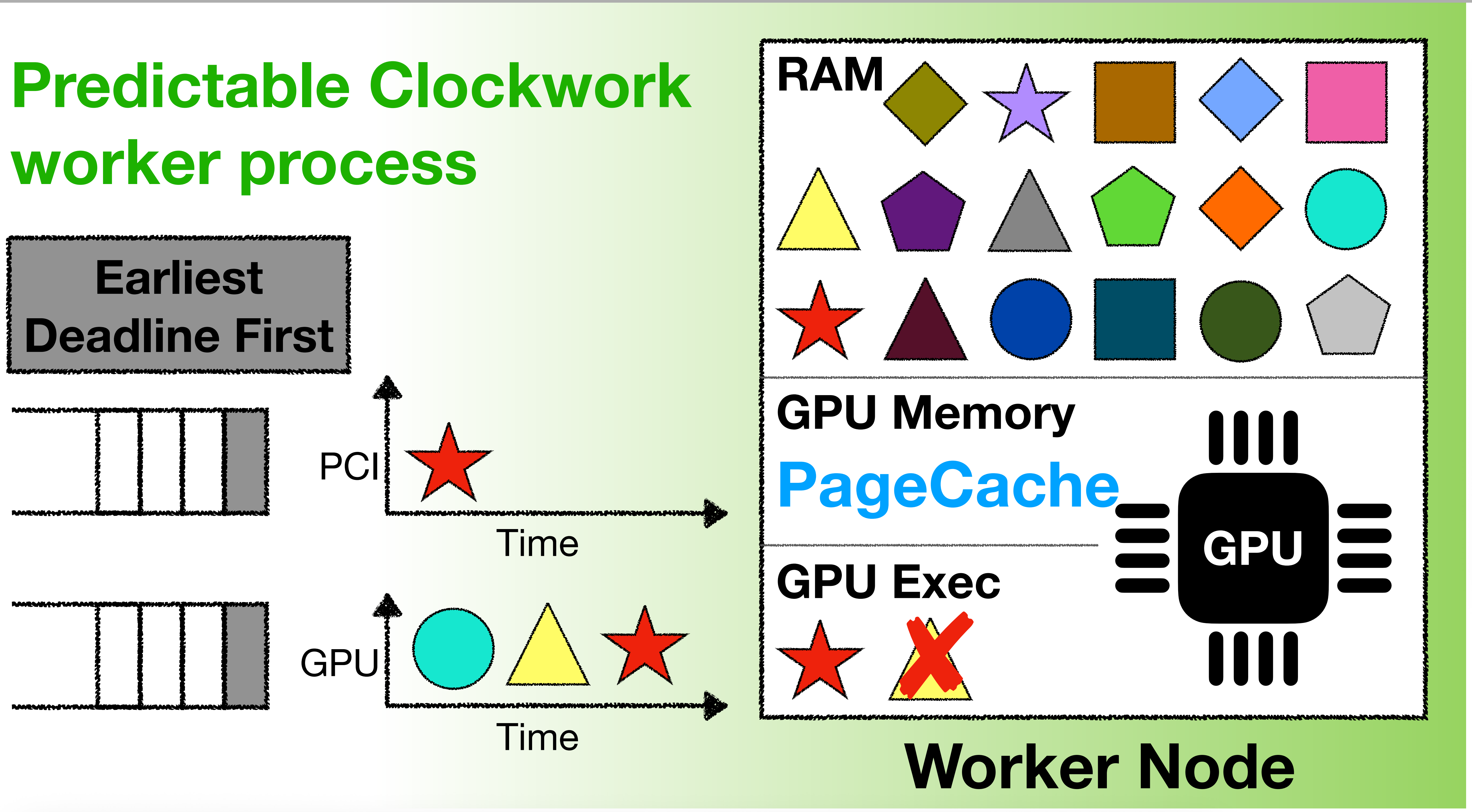
+ Proprietary & undocumented policies

→ Unpredictable response times

Solution

Execute inference one at a time

Designing a Predictable Worker (2/2)



Managed memory can be unpredictable

Solution

Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Concurrent inferences

+ Proprietary & undocumented policies

→ Unpredictable response times

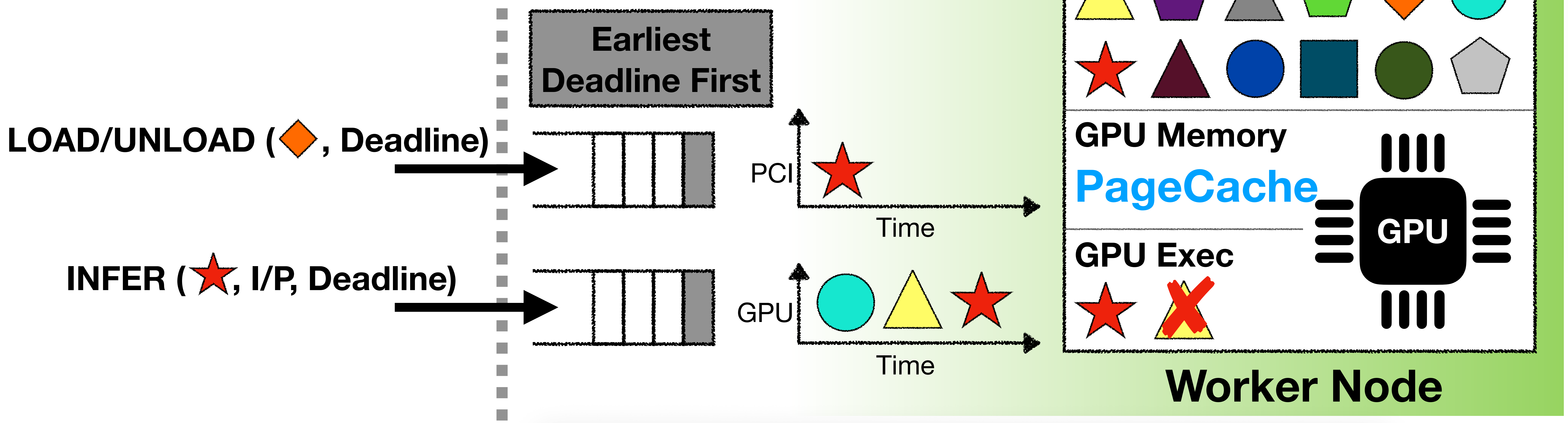
Solution

Execute inference one at a time

Designing a Predictable Worker (2/2)

Choices outsourced via action APIs

Predictable Clockwork worker process



Managed memory can be unpredictable

Solution

Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Concurrent inferences

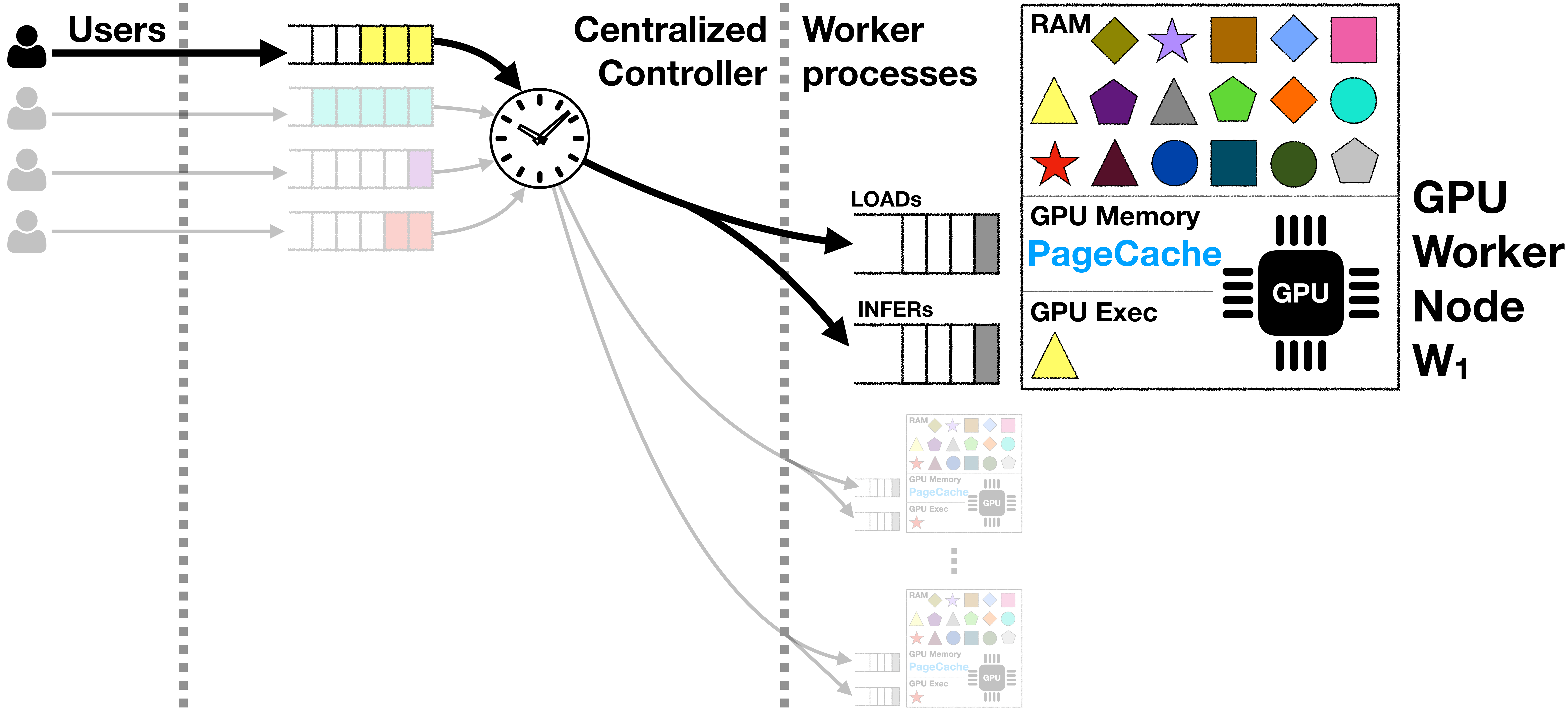
+ Proprietary & undocumented policies

→ Unpredictable response times

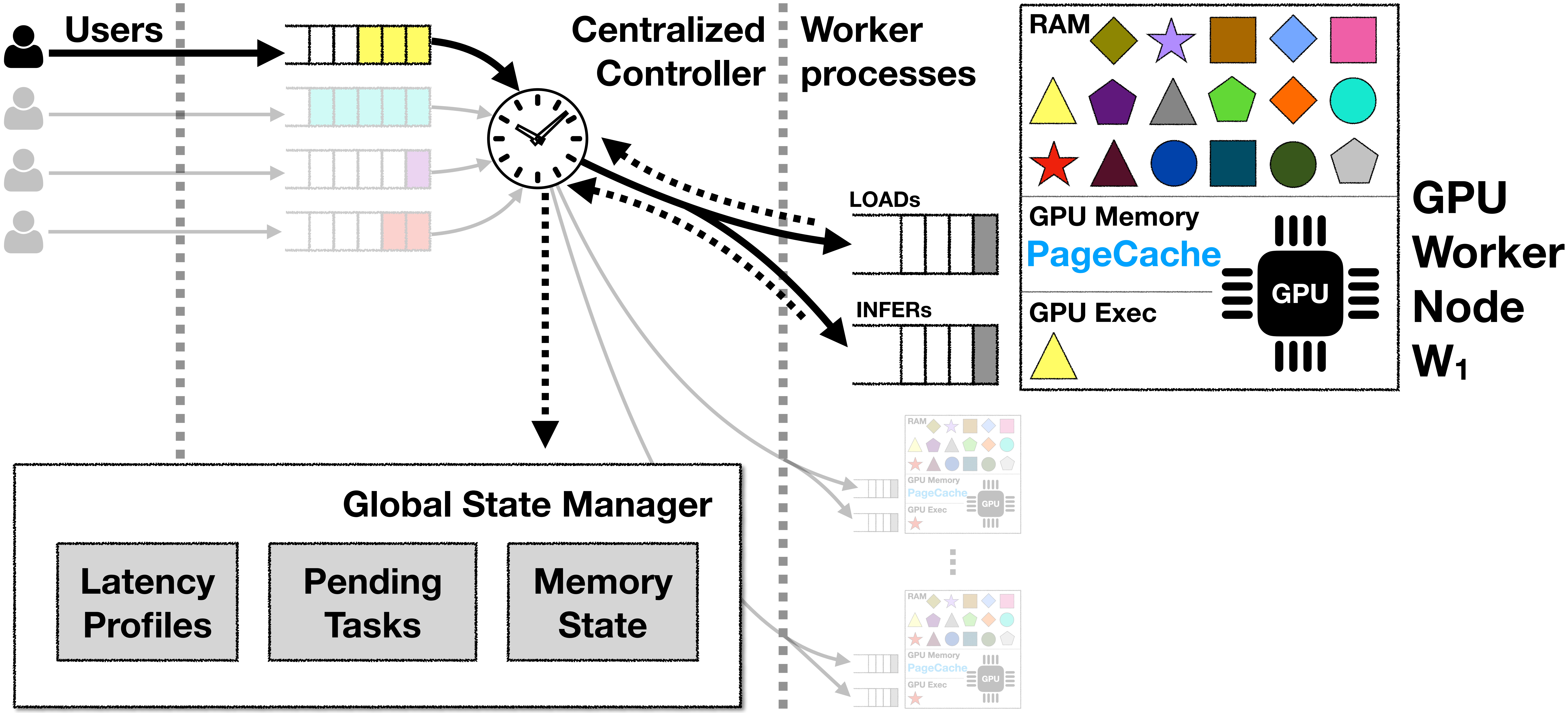
Solution

Execute inference one at a time

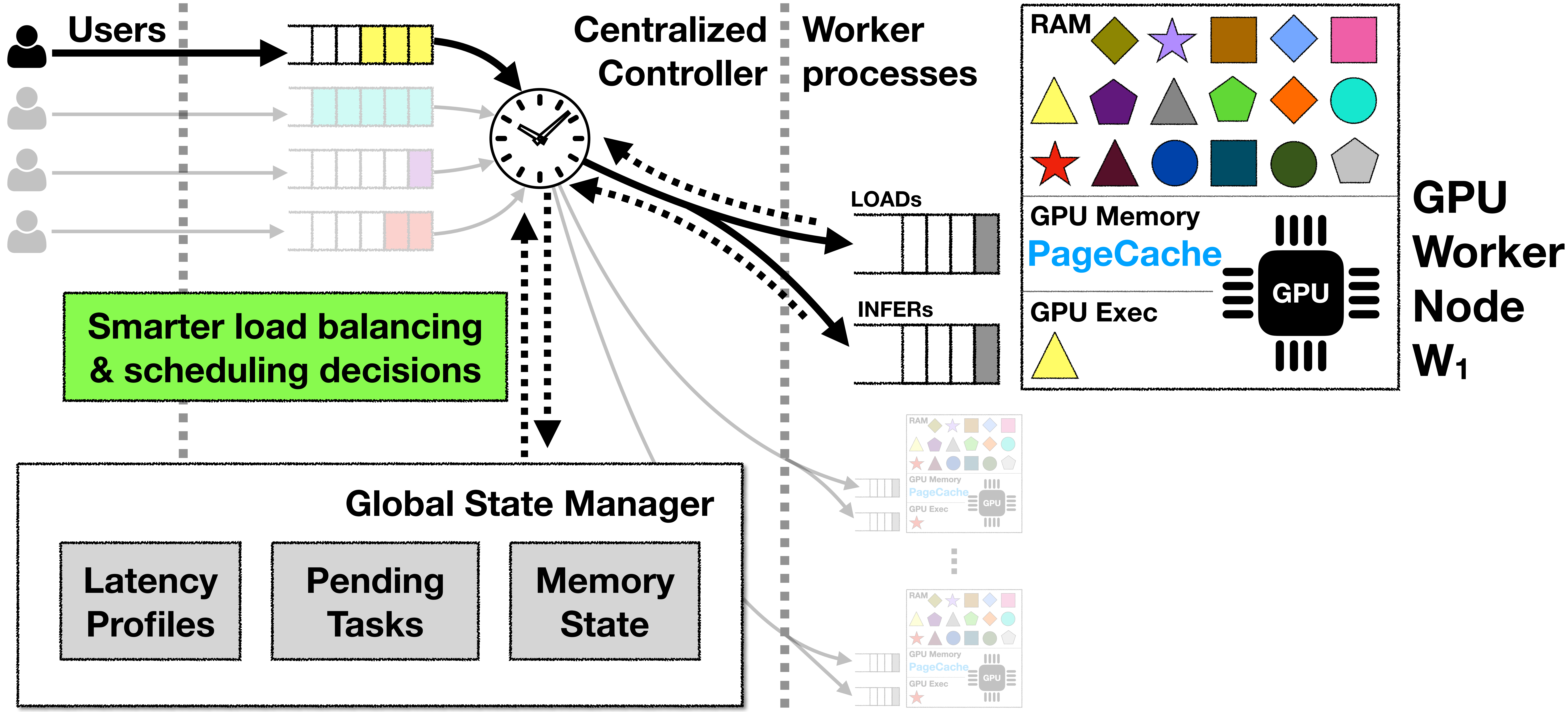
Consolidating Choices



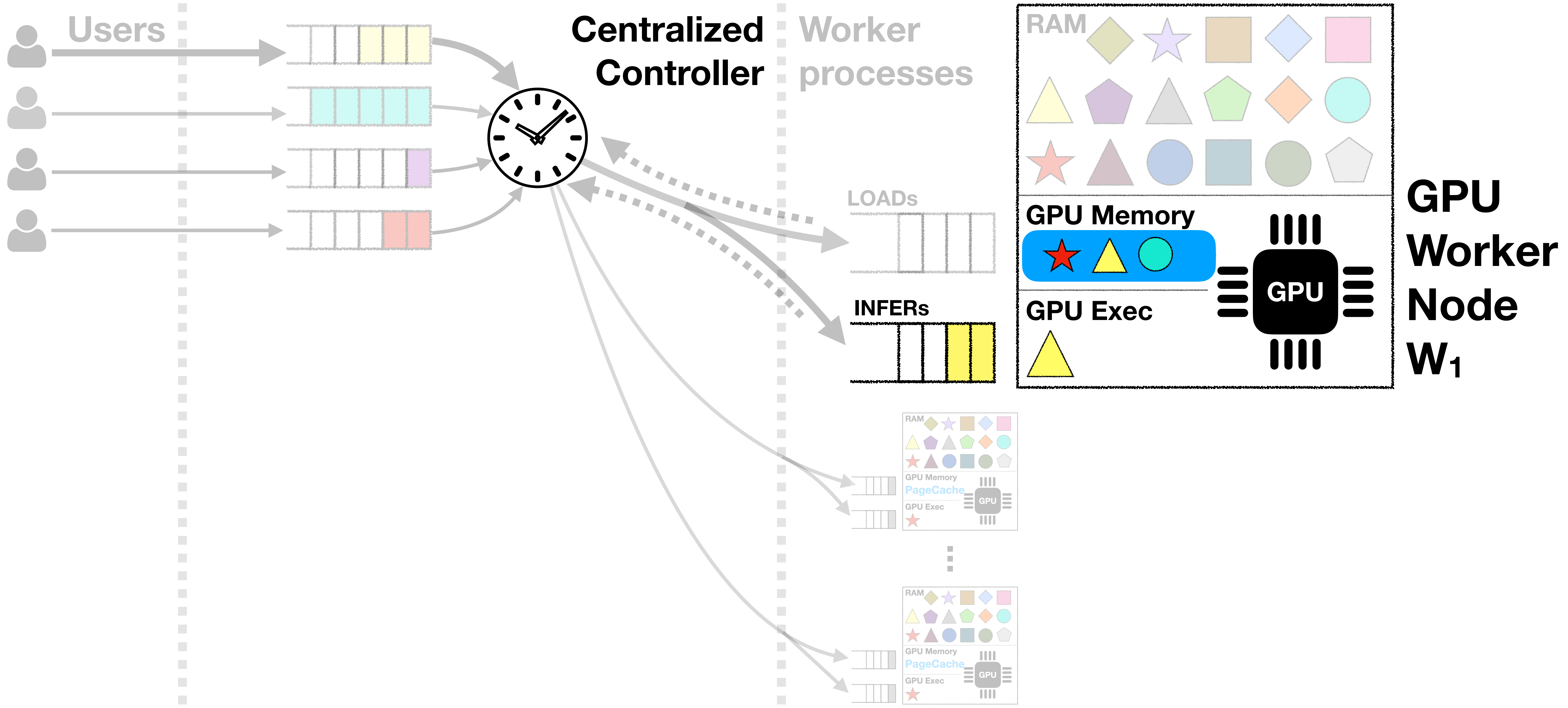
Consolidating Choices



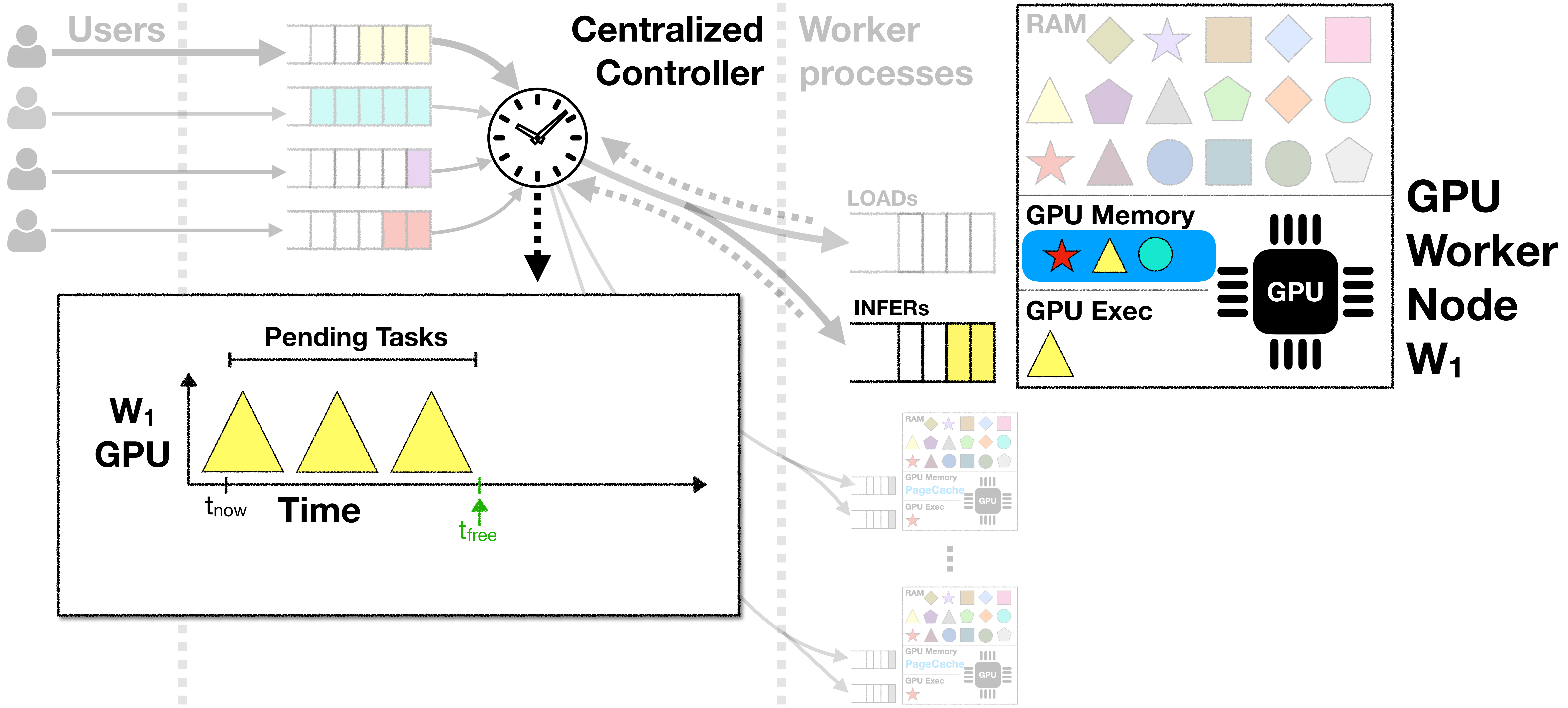
Consolidating Choices



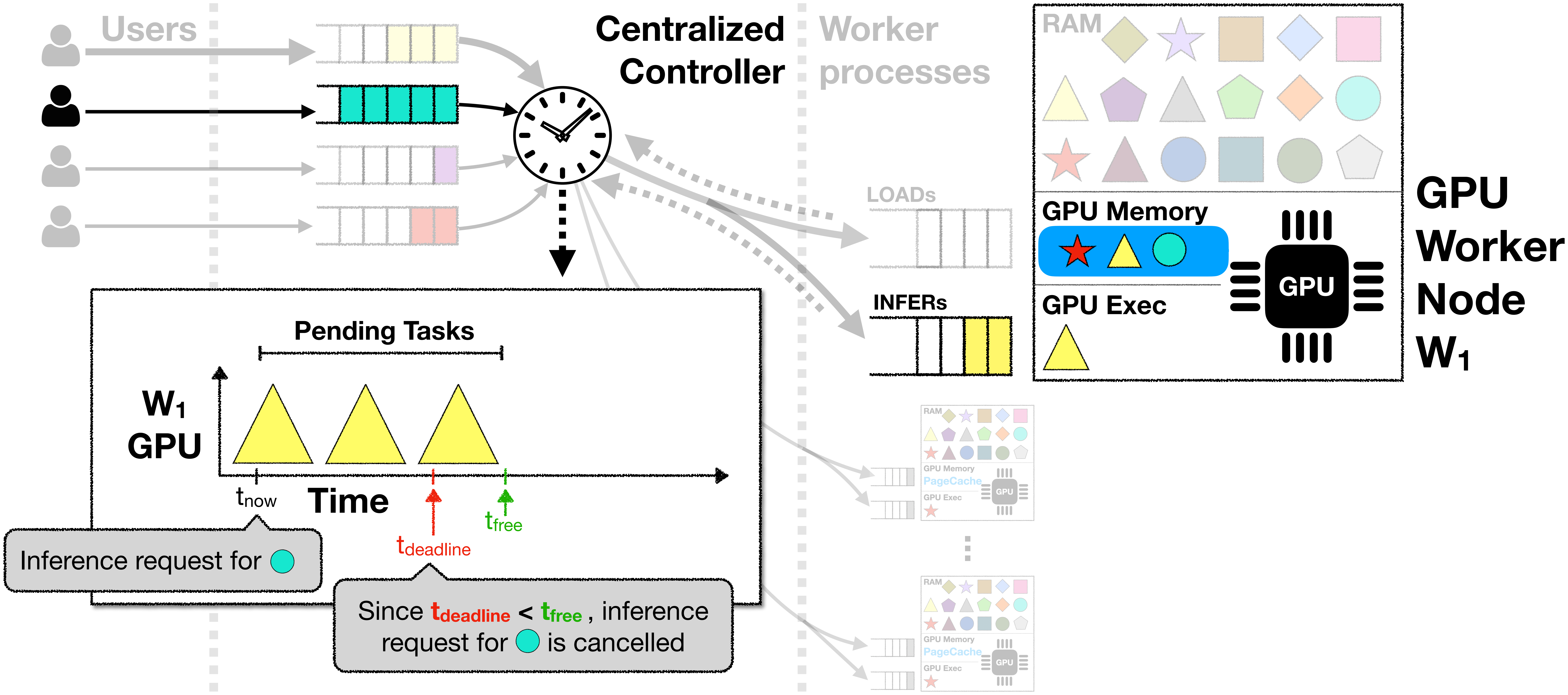
SLO-aware Scheduling



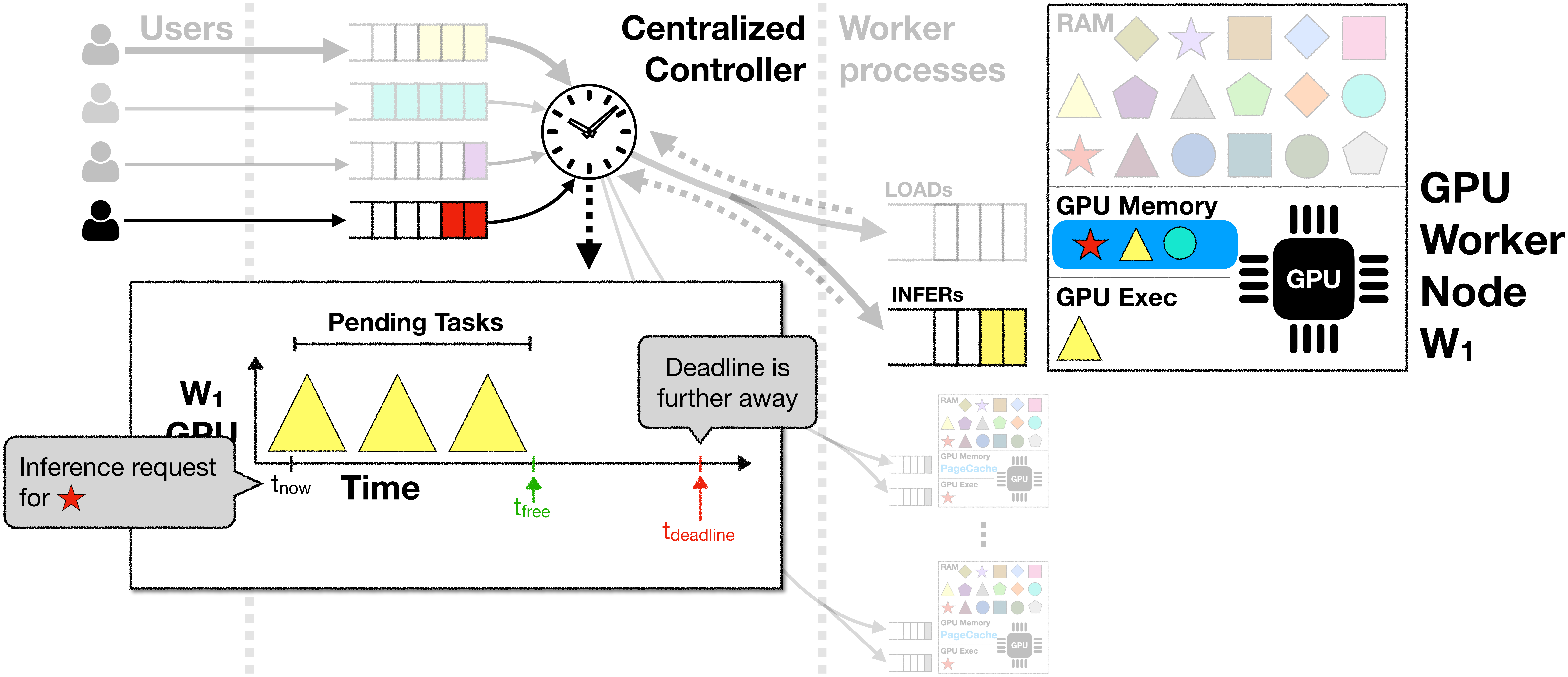
SLO-aware Scheduling



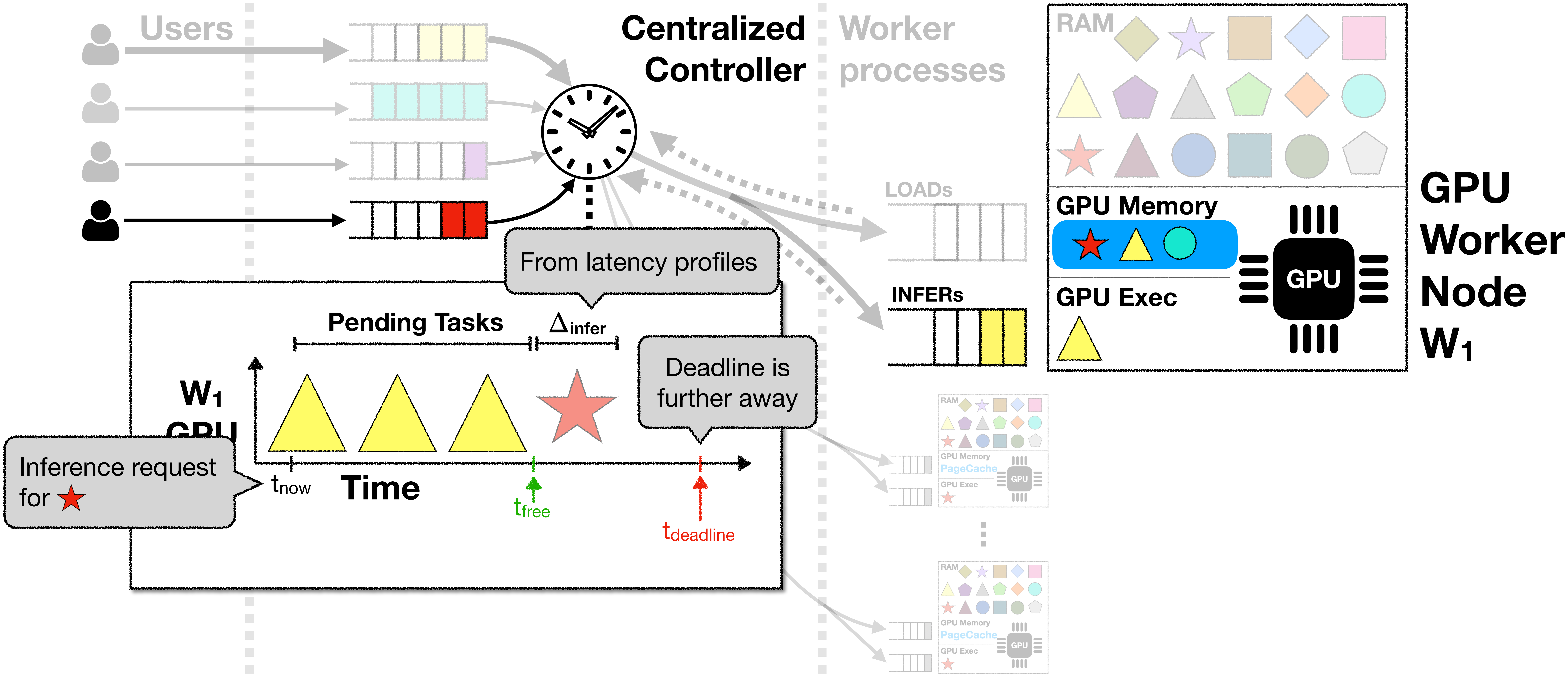
SLO-aware Scheduling



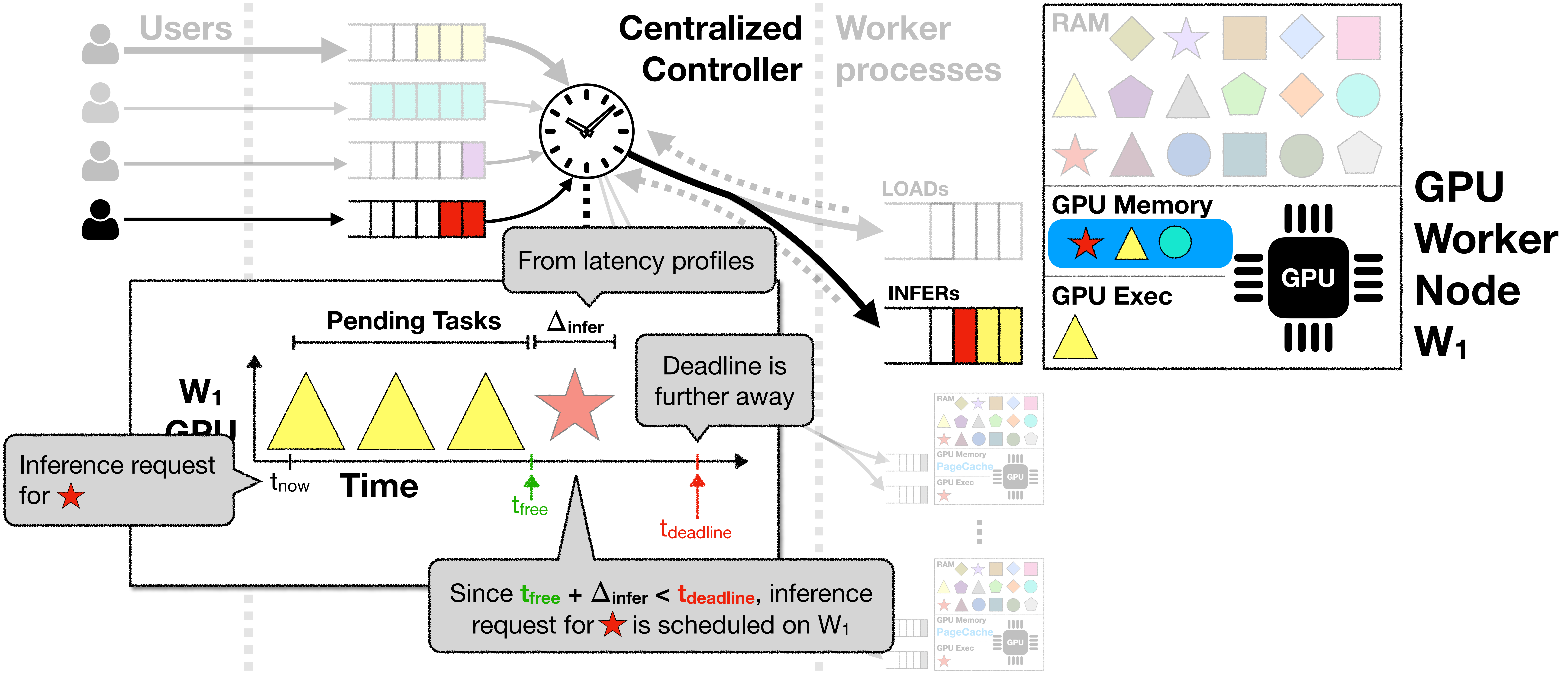
SLO-aware Scheduling



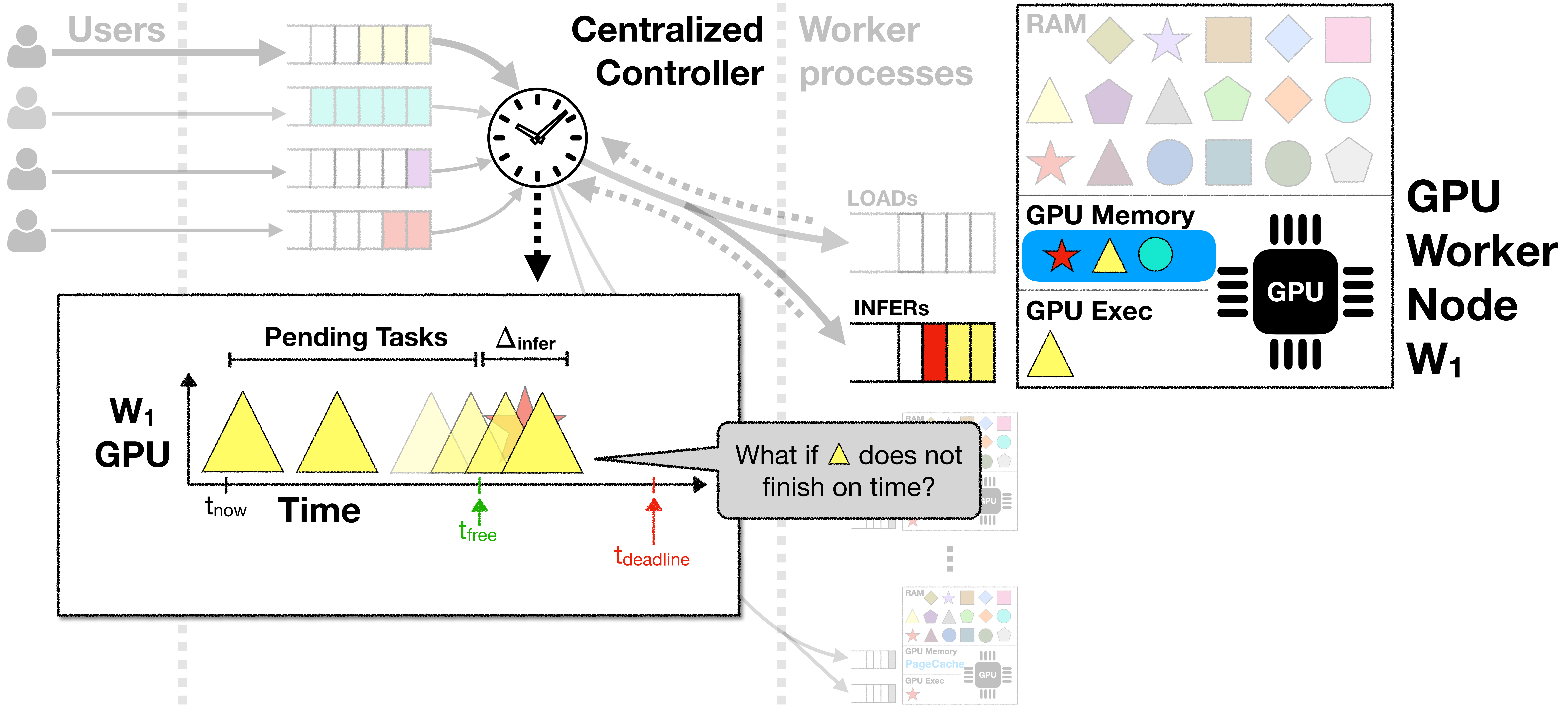
SLO-aware Scheduling



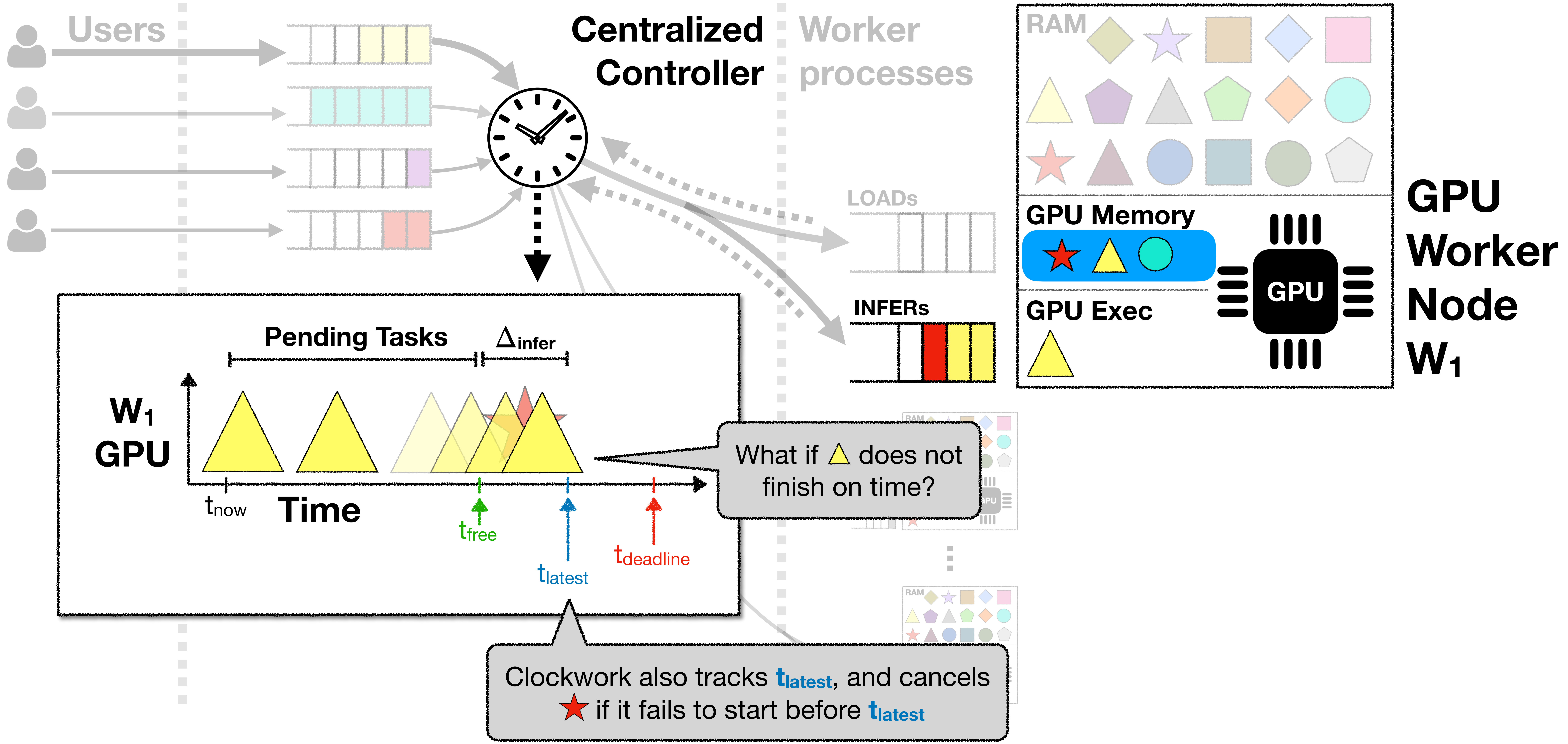
SLO-aware Scheduling



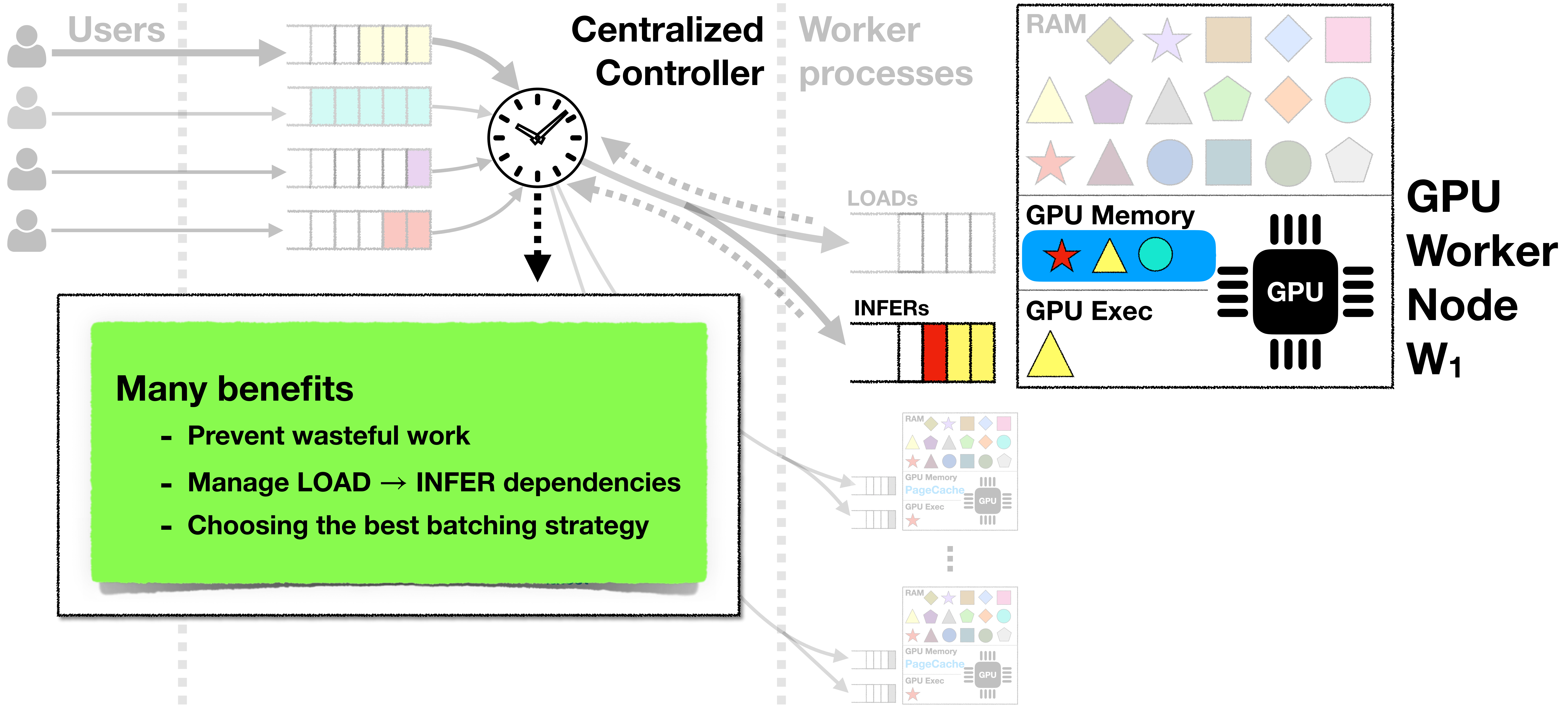
SLO-aware Scheduling



SLO-aware Scheduling



SLO-aware Scheduling



Many benefits

- Prevent wasteful work
- Manage LOAD → INFER dependencies
- Choosing the best batching strategy

Evaluation

Questions

Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?

Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients
from batch requests without latency SLOs?

Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?

Are Clockwork workers predictable?

Does consolidating choice help achieve end-to-end predictability?

Can Clockwork controller Scale?

Workloads from production traces

Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?

This talk

Are Clockwork workers predictable?

Does consolidating choice help achieve end-to-end predictability?

Can Clockwork controller Scale?

Workloads from production traces

Experiment Setup

12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory + 1 Controller + 1 Client

Experiment Setup

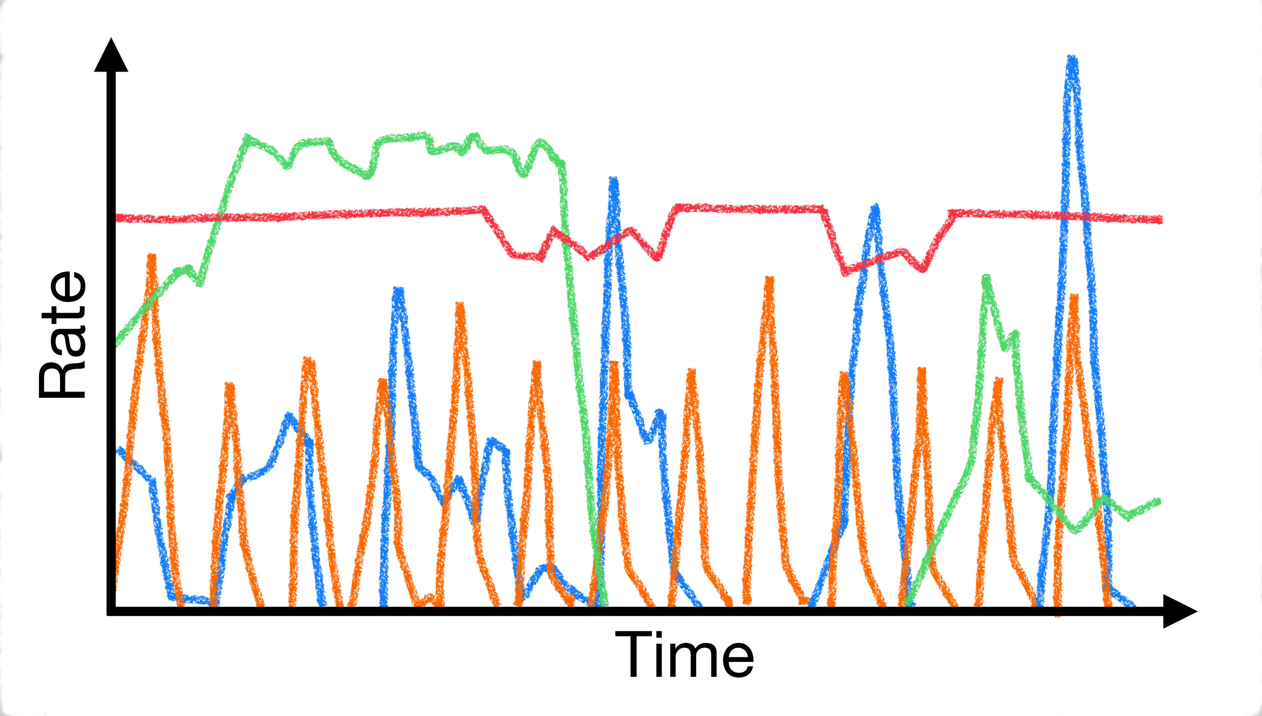
12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory **+** **1 Controller** **+** **1 Client**

Microsoft's Azure Functions

Shahrad et al. *"Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider."* USENIX ATC 2020

46,000 functions, 2 weeks

- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads



Workload

Experiment Setup

12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory **+** **1 Controller** **+** **1 Client**

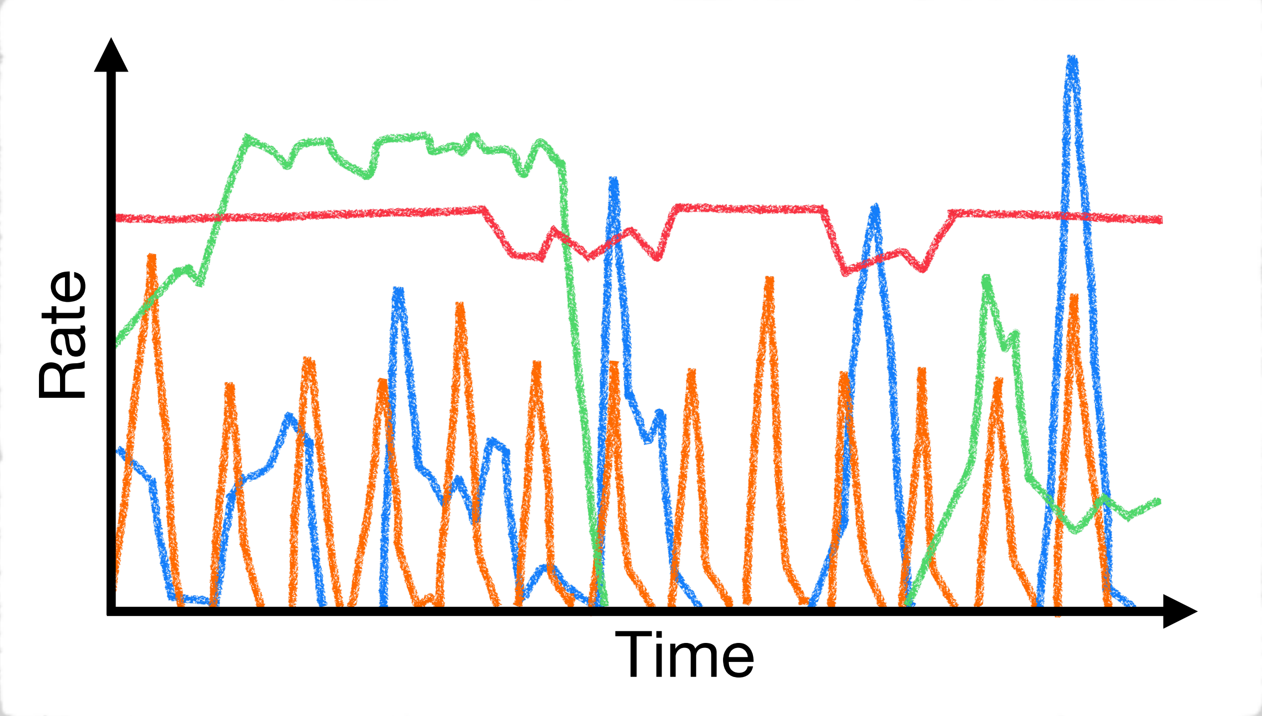
Workload

Microsoft's Azure Functions

Shahrad et al. *"Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider."* USENIX ATC 2020

46,000 functions, 2 weeks

- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads



4026 model instances

- Saturates 768 GB RAM
- 61 different model architectures
- ResNet, DenseNet, Inception, etc.

Are Clockwork Workers Predictable?

Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

Overpredictions → **Idle resources**

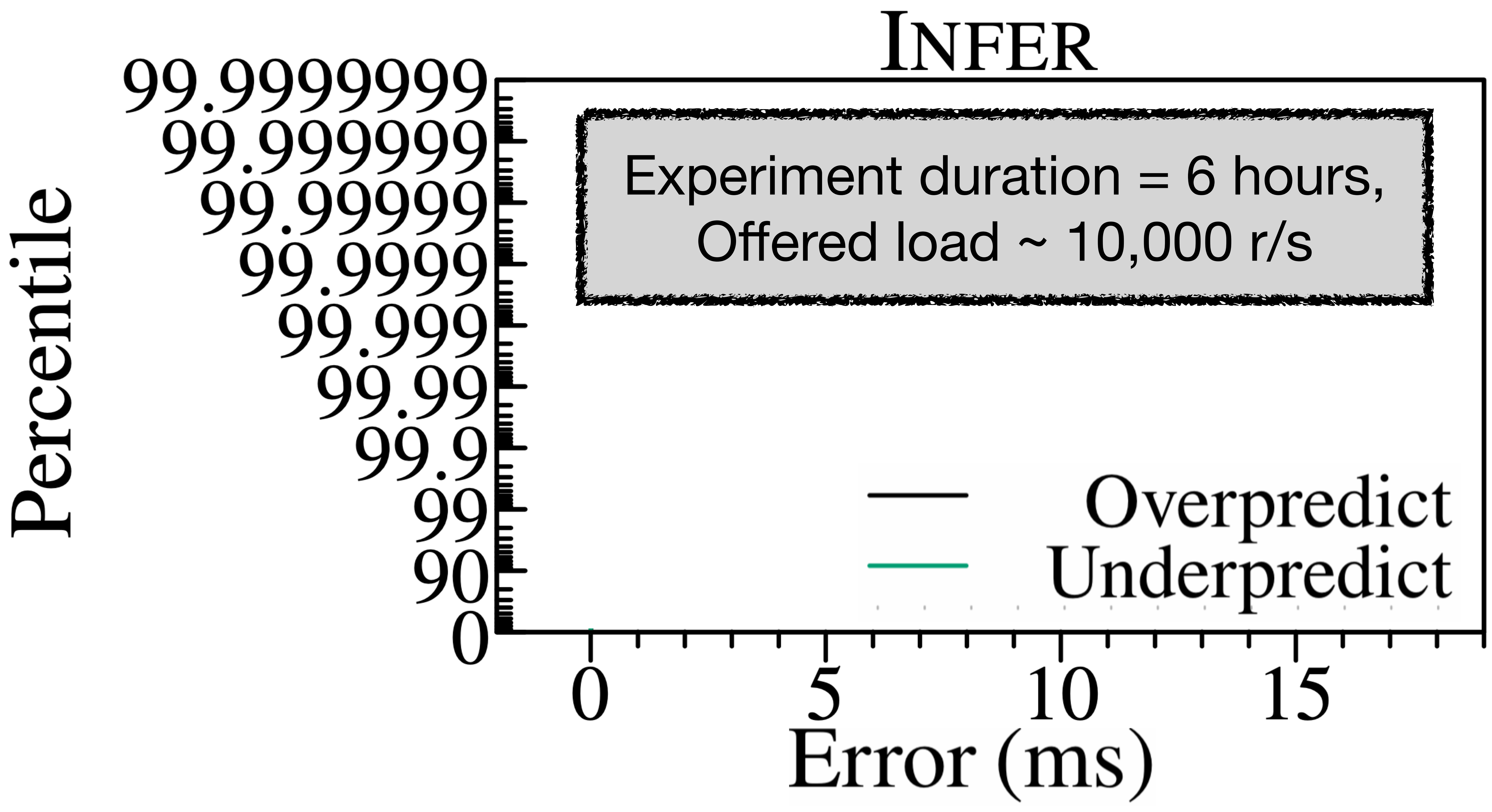
Underpredictions → **SLO violations**

Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

Overpredictions → Idle resources

Underpredictions → SLO violations

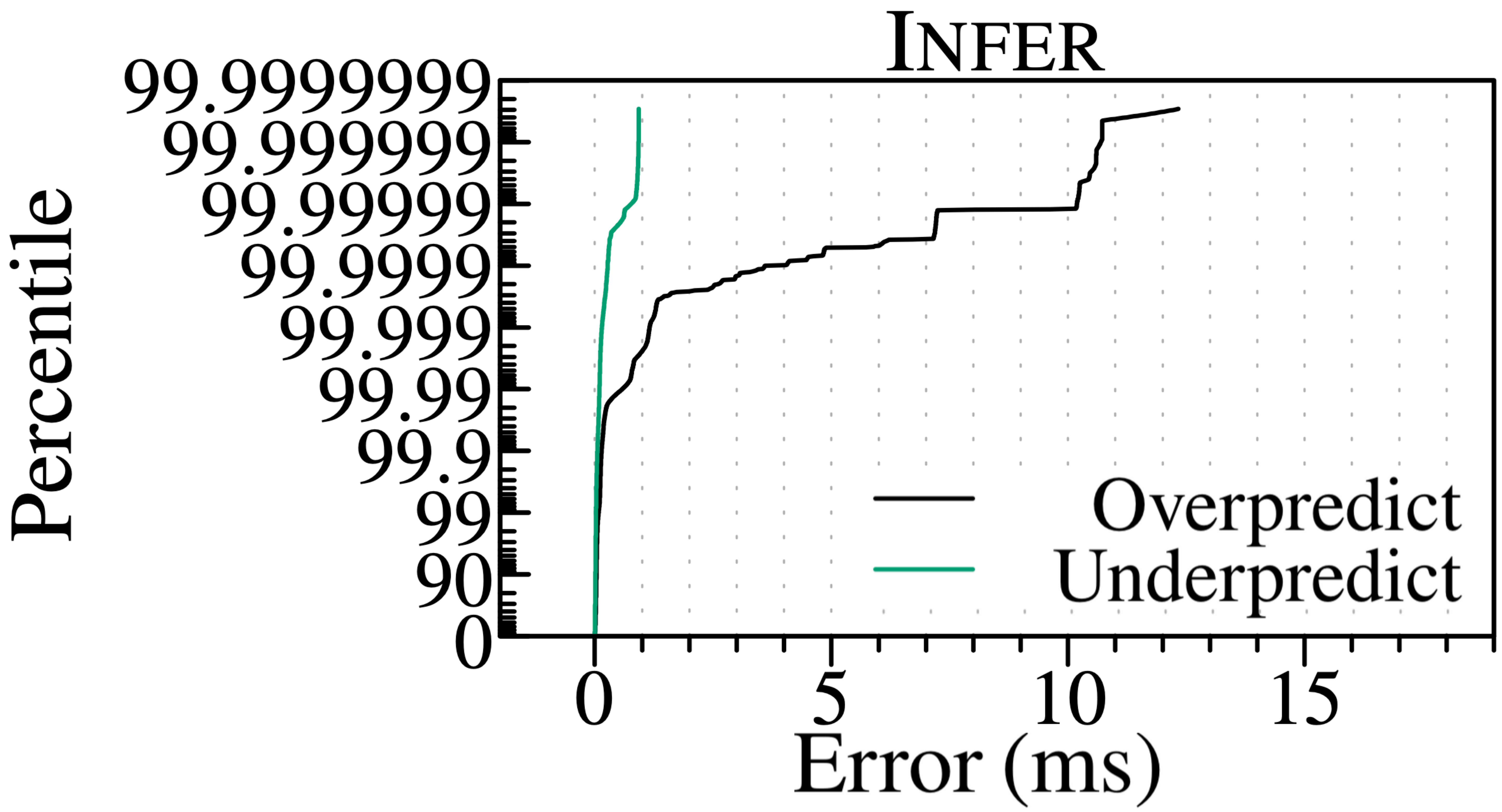


Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

Overpredictions → Idle resources

Underpredictions → SLO violations

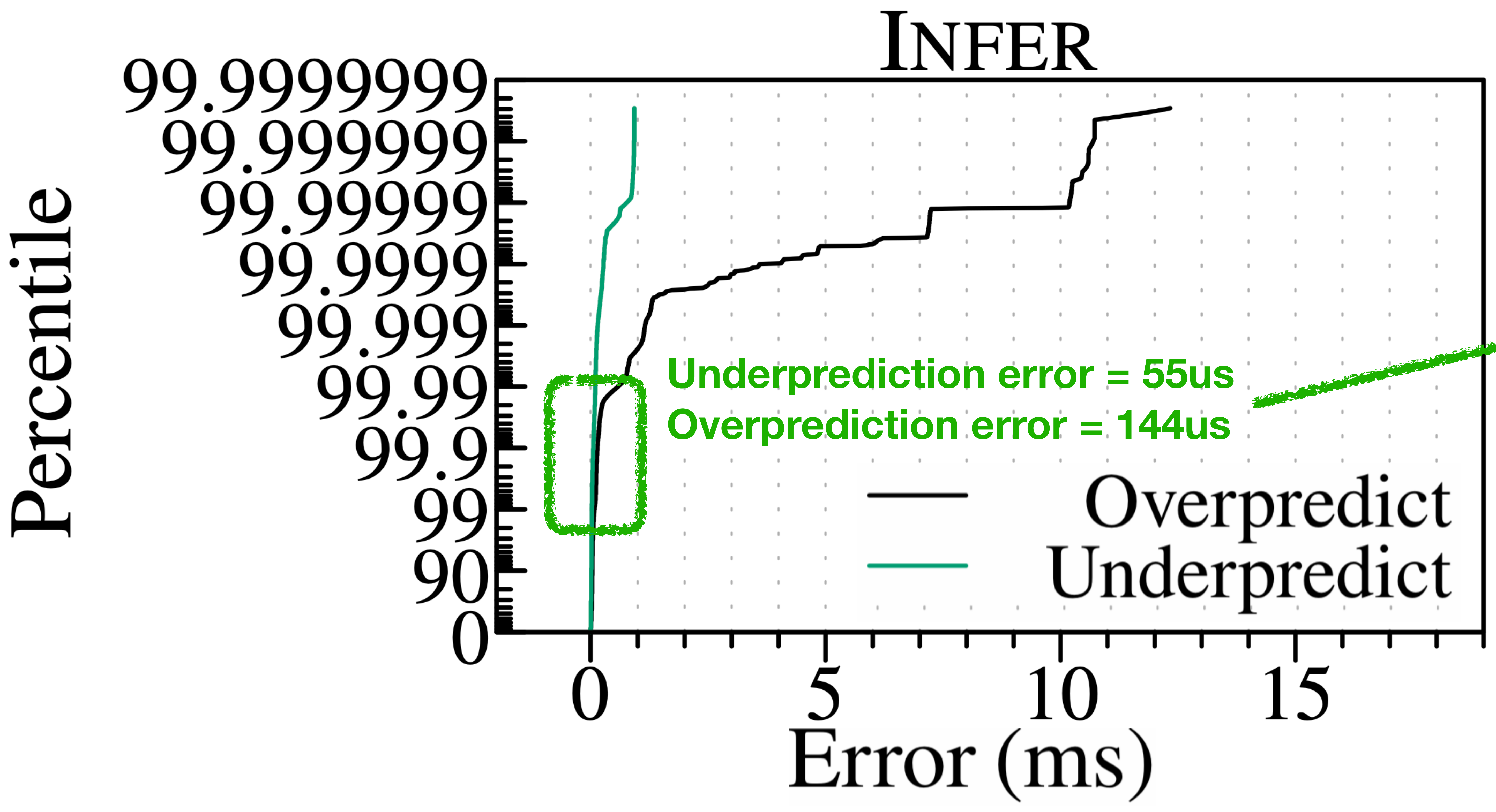


Clockwork consistently overpredicts more than its underpredicts

Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

Overpredictions → Idle resources
Underpredictions → SLO violations



Clockwork consistently overpredicts more than its underpredicts

Errors are significant only in extremely rare cases

Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

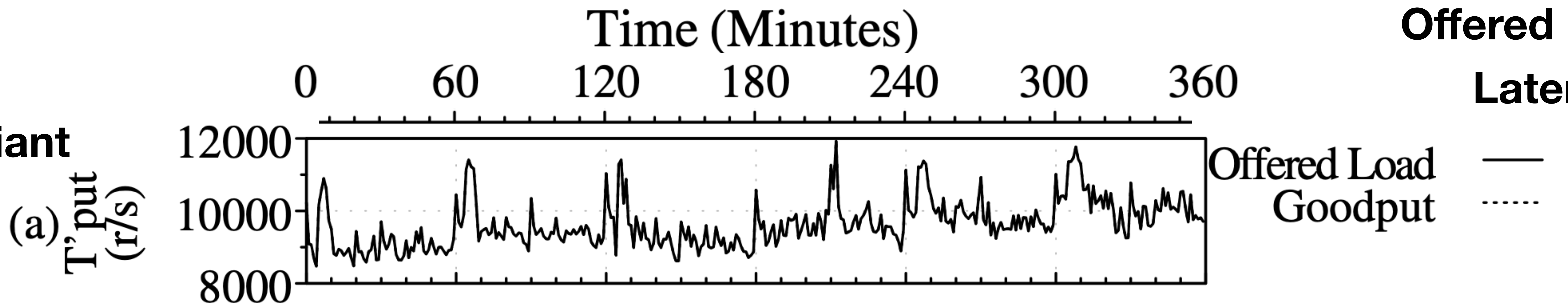
Latency SLO = 100 ms deadline for each request

Does Consolidating Choice Help?

Goodput = SLO compliant throughput

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request



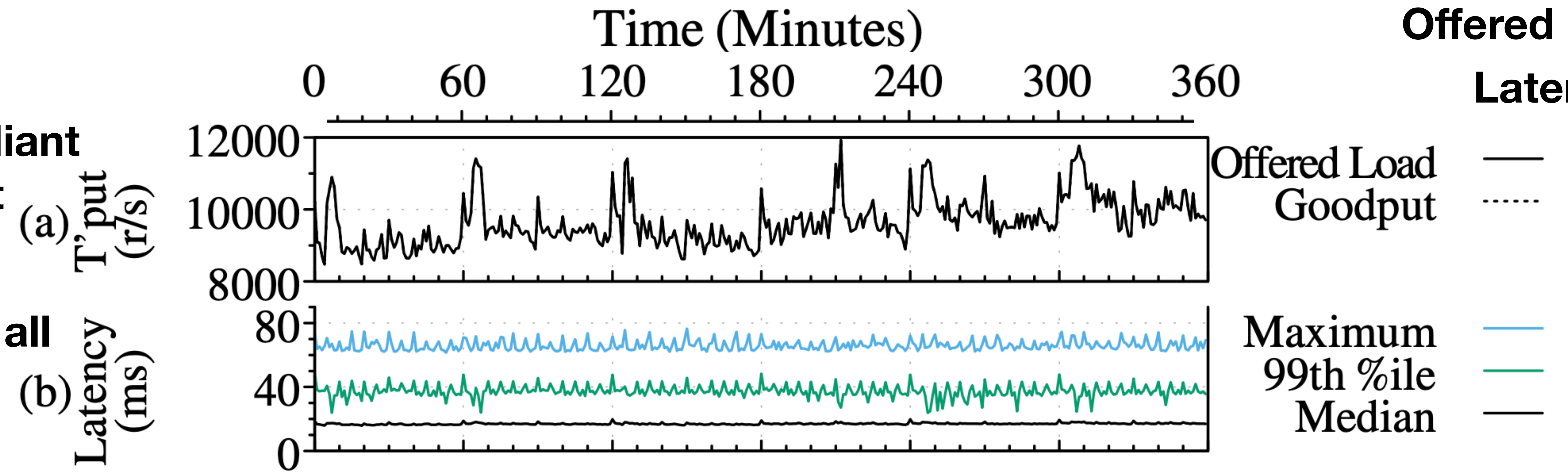
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request

Goodput = SLO compliant throughput

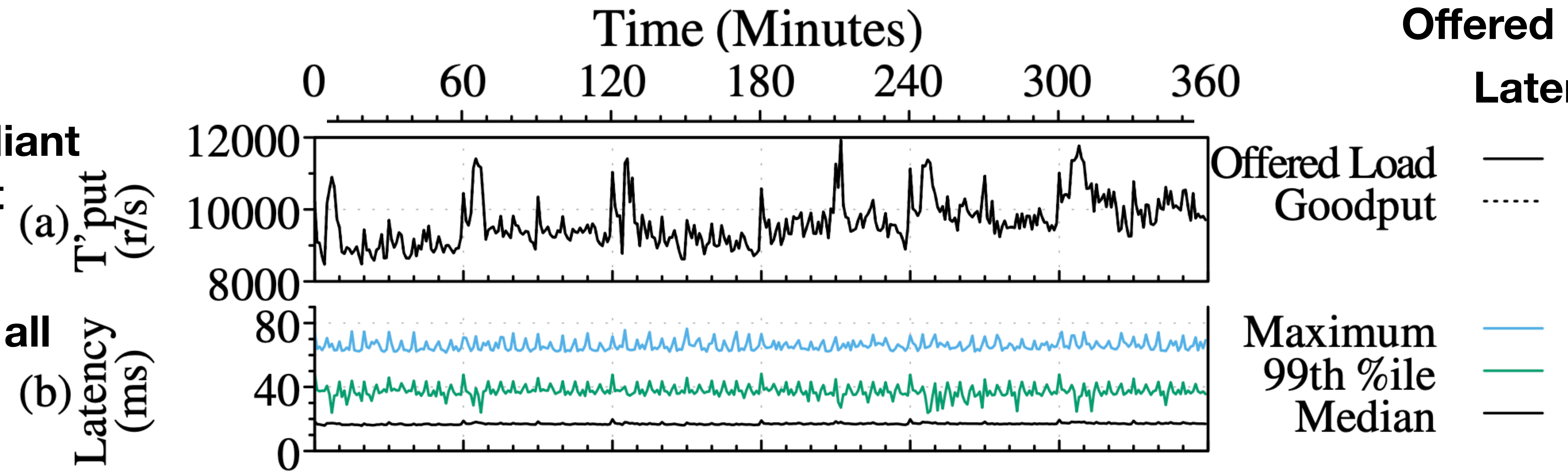
Latency of all completed requests



Does Consolidating Choice Help?

Goodput = SLO compliant throughput

Latency of all completed requests



Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork

- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

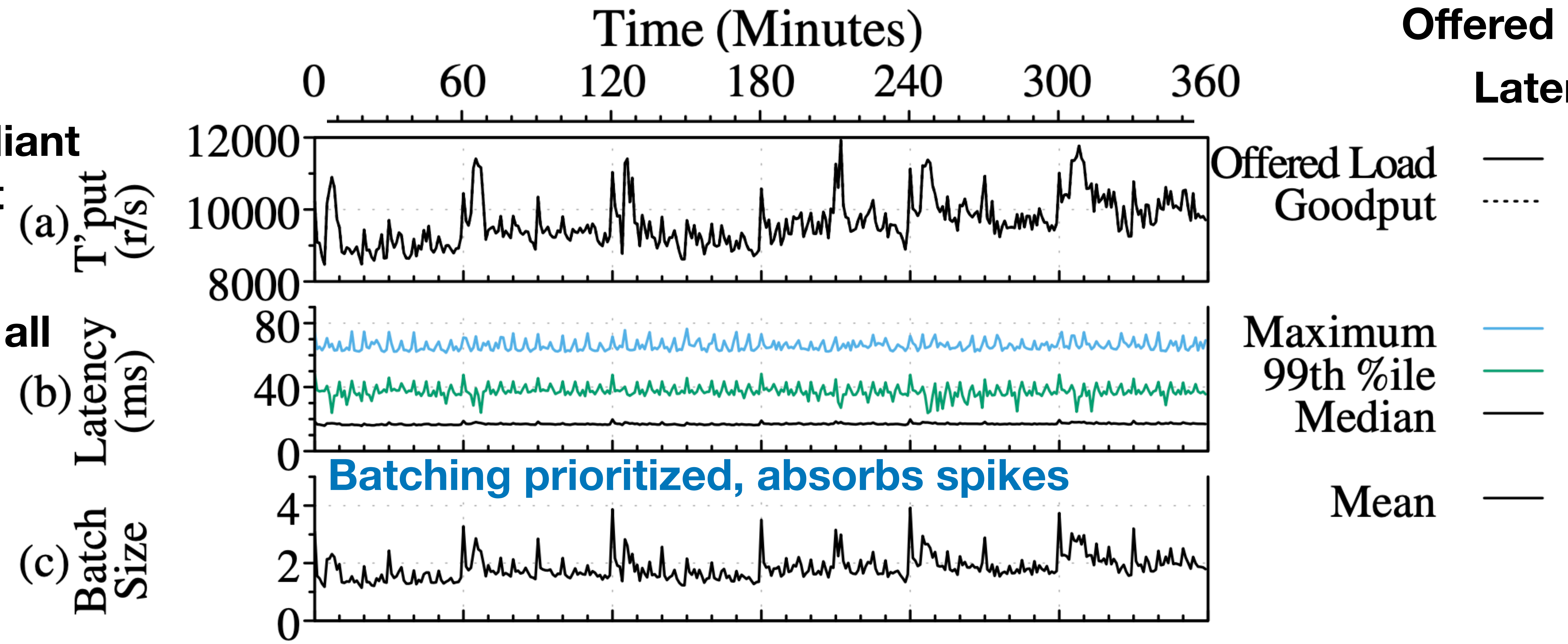
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request

Goodput = SLO compliant throughput

Latency of all completed requests



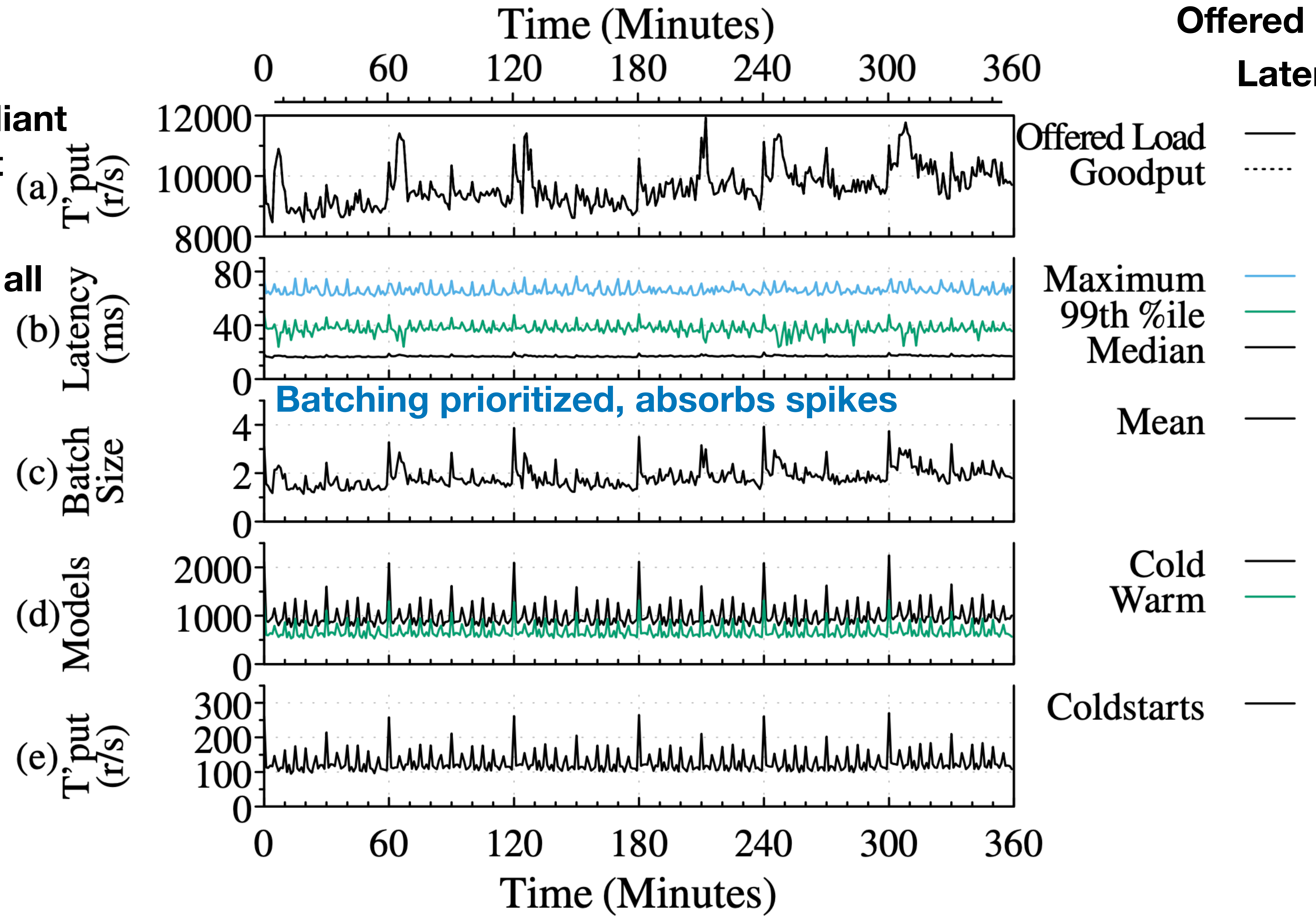
The workload is successfully scheduled by Clockwork

- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

Does Consolidating Choice Help?

Goodput = SLO compliant throughput

Latency of all completed requests



Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request

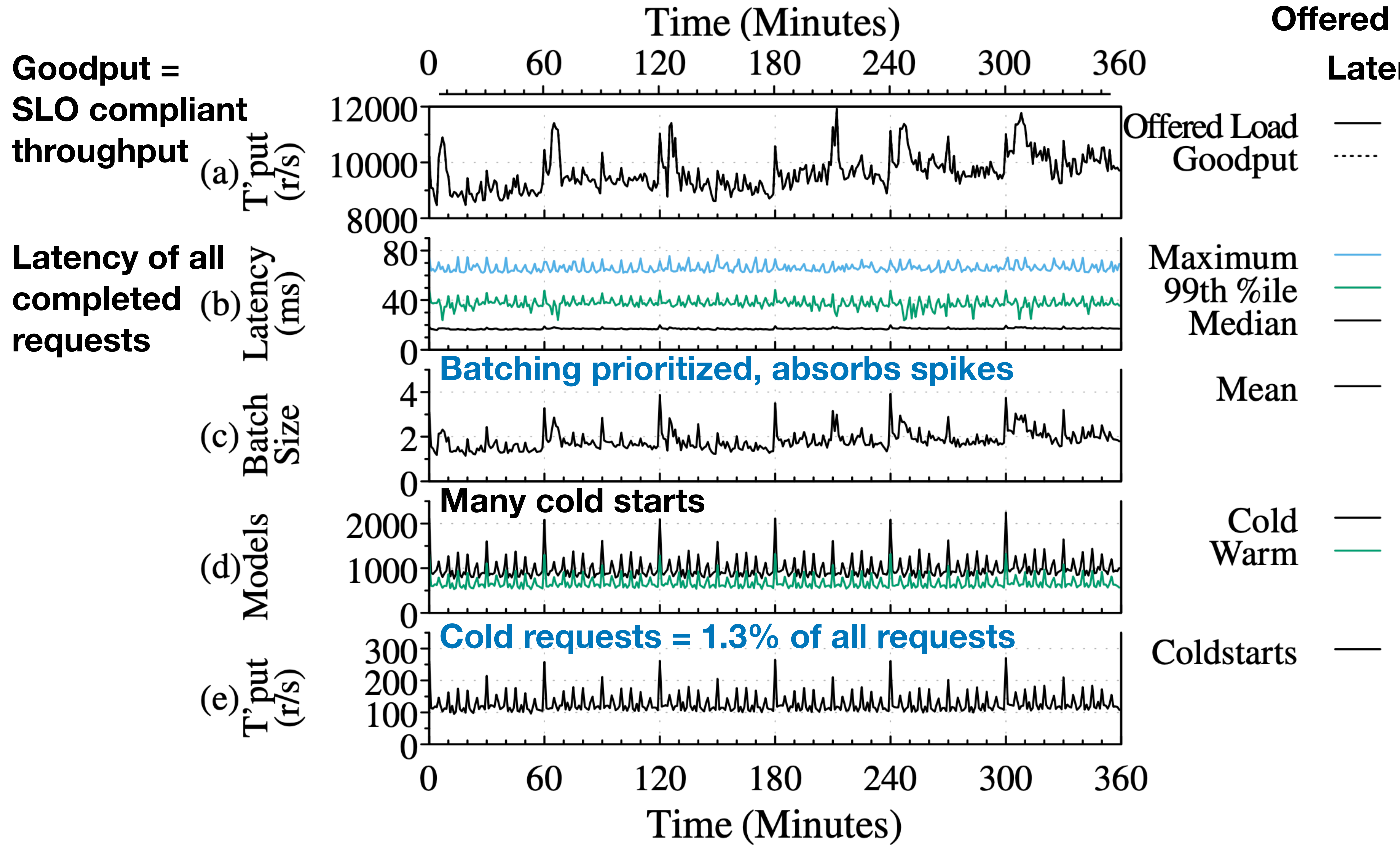
The workload is successfully scheduled by Clockwork

- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request



The workload is successfully scheduled by Clockwork

- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

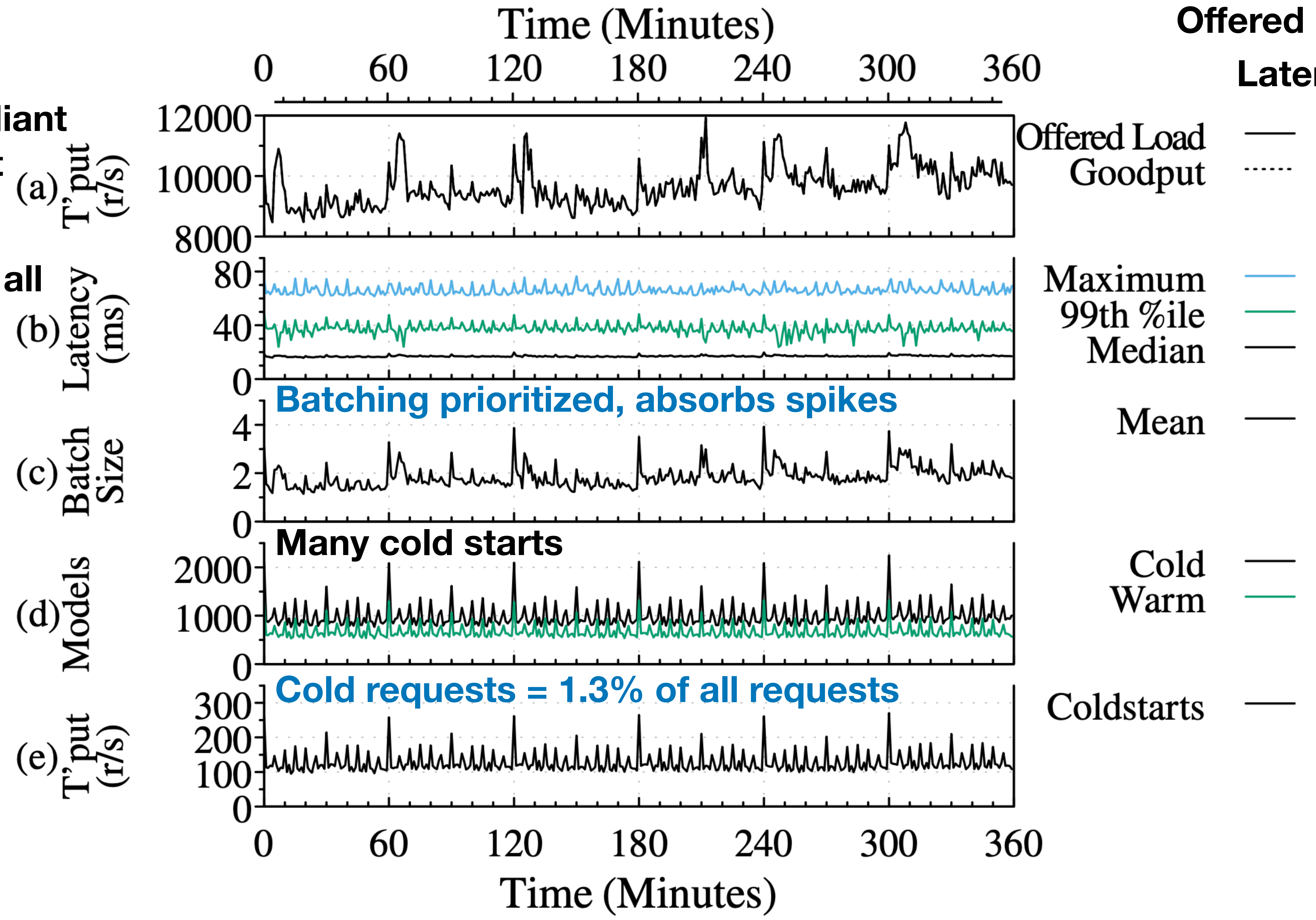
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s

Latency SLO = 100 ms deadline for each request

Goodput = SLO compliant throughput

Latency of all completed requests



The workload is successfully scheduled by Clockwork

- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

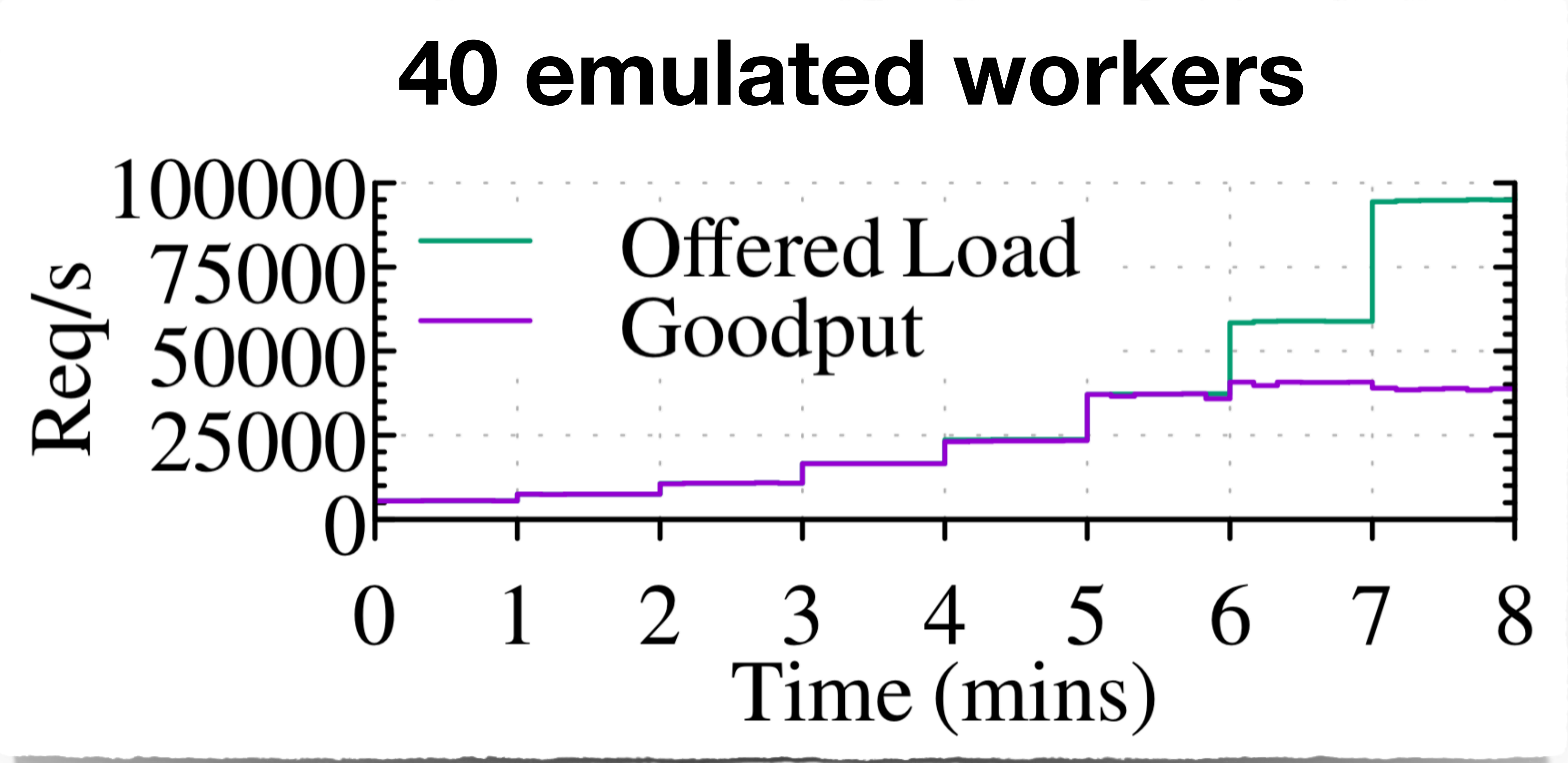
Does Clockwork Controller Scale?

Does Clockwork Controller Scale?

Methodology

- **Replace GPU workers with emulated workers**
- **From the controller's vantage point, nothing changes**
- **Measure the peak goodput as we vary #workers**

Does Clockwork Controller Scale?

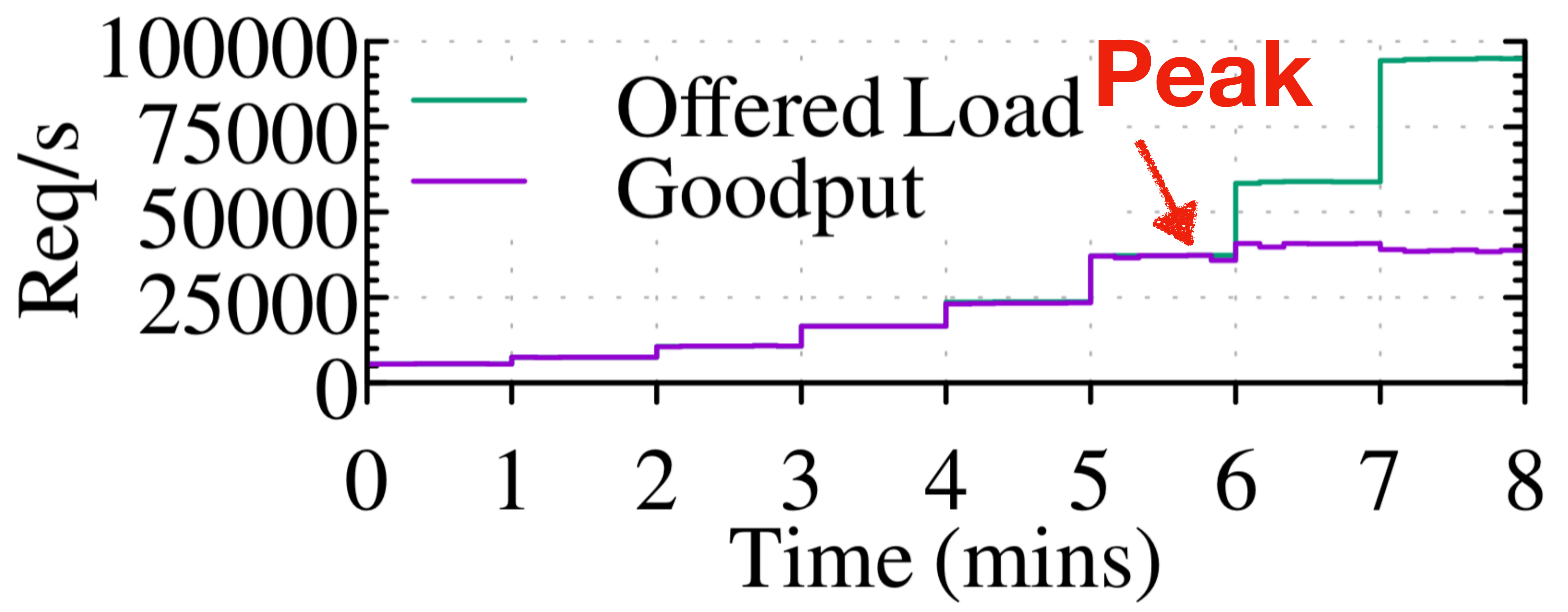


Methodology

- Replace GPU workers with emulated workers
- From the controller's vantage point, nothing changes
- Measure the peak goodput as we vary #workers

Does Clockwork Controller Scale?

40 emulated workers



Methodology

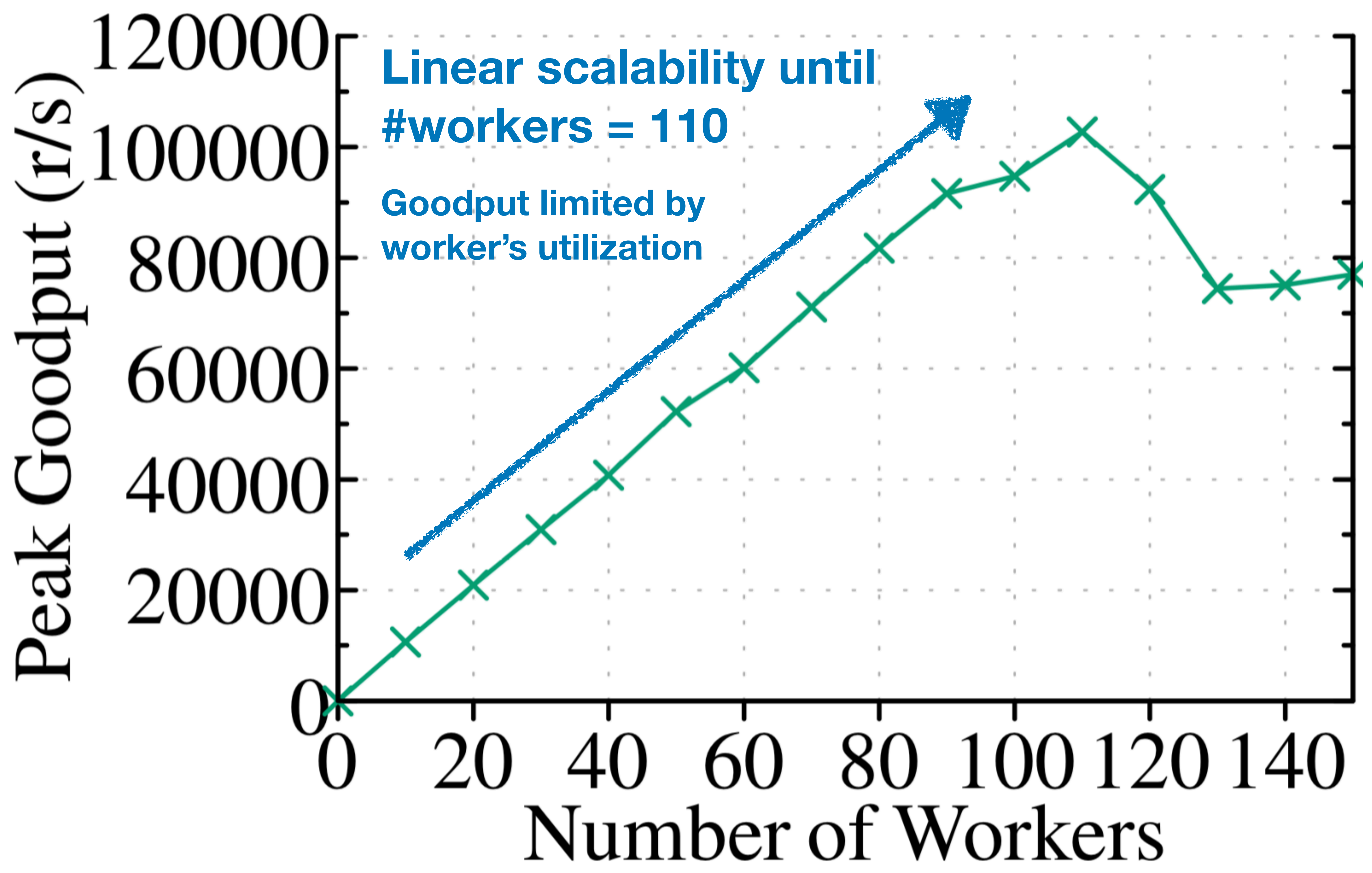
- Replace GPU workers with emulated workers
- From the controller's vantage point, nothing changes
- Measure the peak goodput as we vary #workers

Does Clockwork Controller Scale?

Methodology

- **Replace GPU workers with emulated workers**
- **From the controller's vantage point, nothing changes**
- **Measure the peak goodput as we vary #workers**

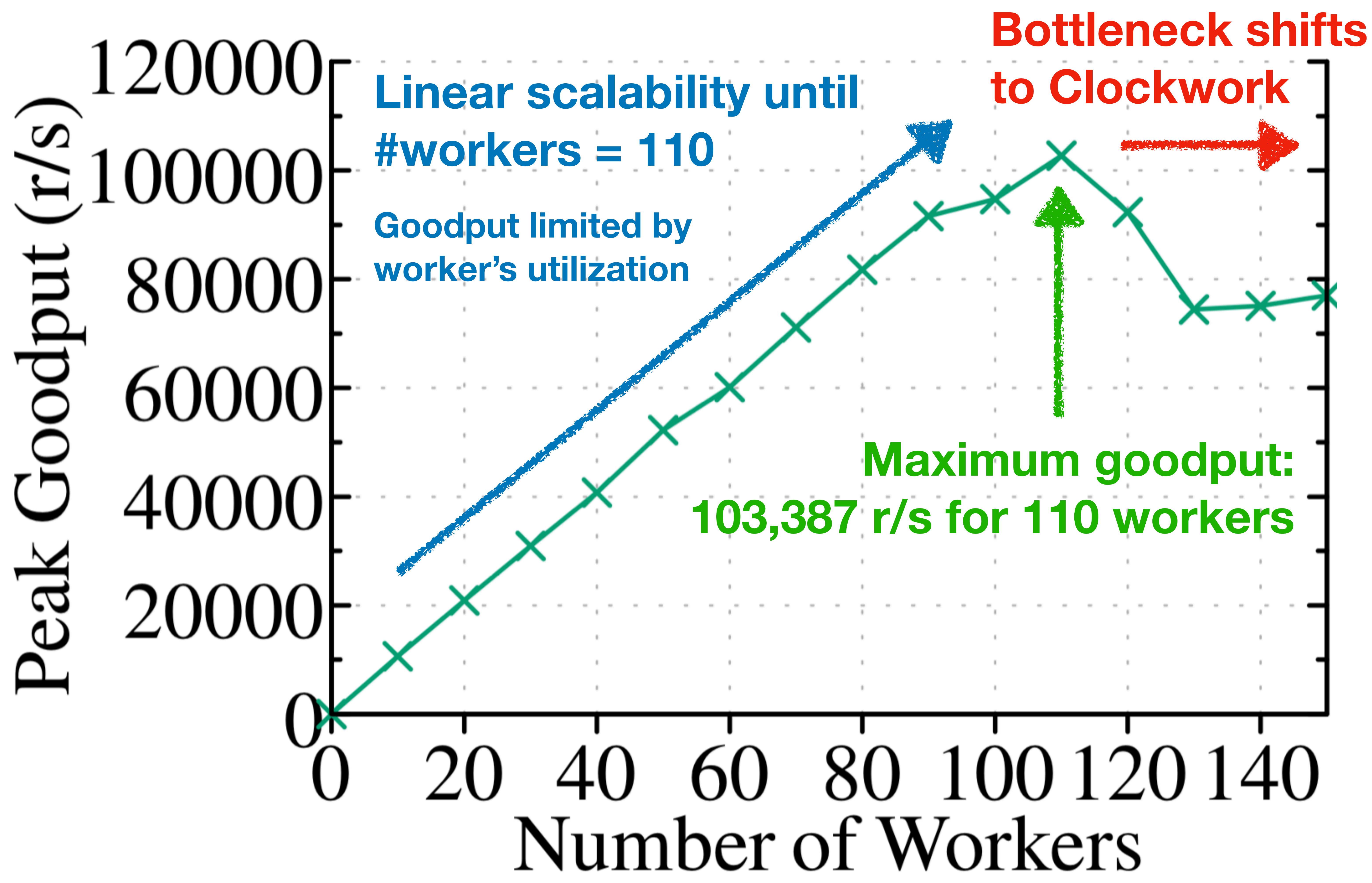
Does Clockwork Controller Scale?



Methodology

- Replace GPU workers with emulated workers
- From the controller's vantage point, nothing changes
- Measure the peak goodput as we vary #workers

Does Clockwork Controller Scale?



Methodology

- Replace GPU workers with emulated workers
- From the controller's vantage point, nothing changes
- Measure the peak goodput as we vary #workers

Summary

Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability

- **Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice**

Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability

- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform

- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller

Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability

- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform

- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller

Outperforms state-of-the-art DNN serving platforms

- Efficiently fulfills aggressive tail-latency SLOs
- Supports 1000s of DNN models with varying workload characteristics concurrently on each GPU

Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability

- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform

- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller

Outperforms state-of-the-art DNN serving platforms

- Efficiently fulfills aggressive tail-latency SLOs
- Supports 1000s of DNN models with varying workload characteristics concurrently on each GPU

https://gitlab.mpi-sws.org/cld/ml/clockwork

ARTIFACT EVALUATED
usenix ASSOCIATION
AVAILABLE

ARTIFACT EVALUATED
usenix ASSOCIATION
FUNCTIONAL

ARTIFACT EVALUATED
usenix ASSOCIATION
REPRODUCED