Colocating ML **Inference** and **Training** with ***Fast GPU Memory Handover***

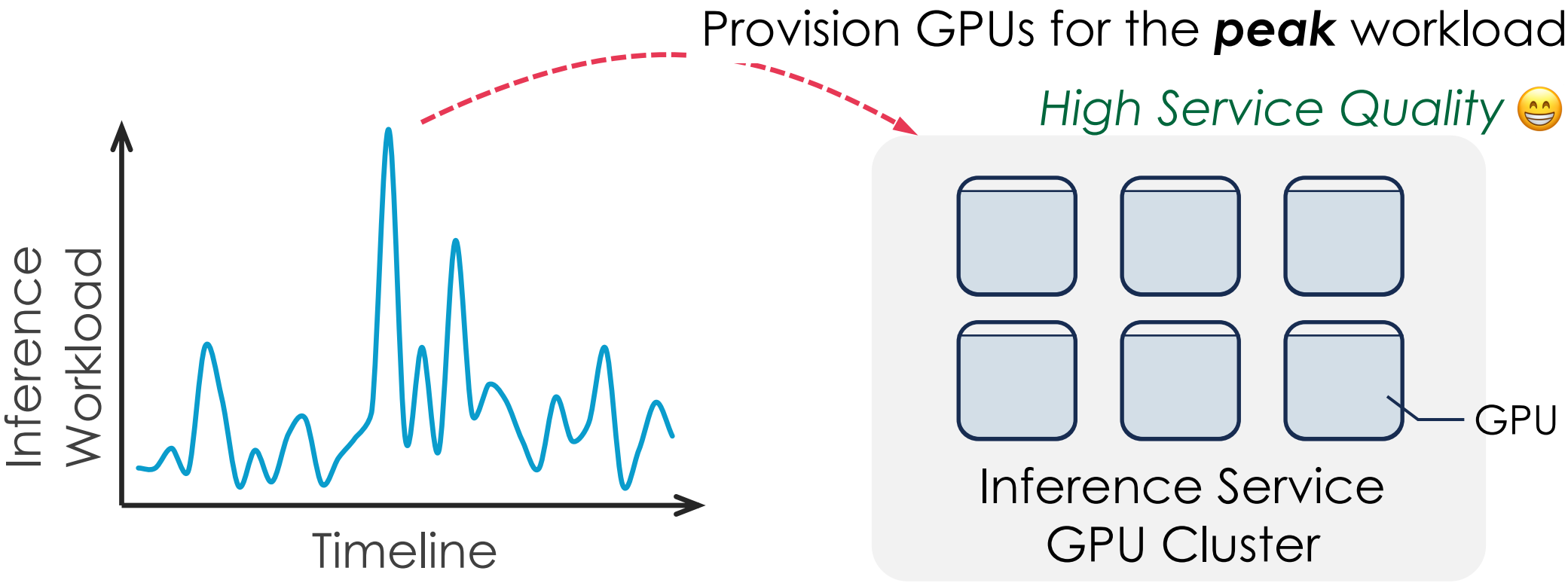
Jiali Wang, Yankui Wang, **Mingcong Han**, Rong Chen

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

SHANGHAI JIAO TONG UNIVERSITY
上海交通大學

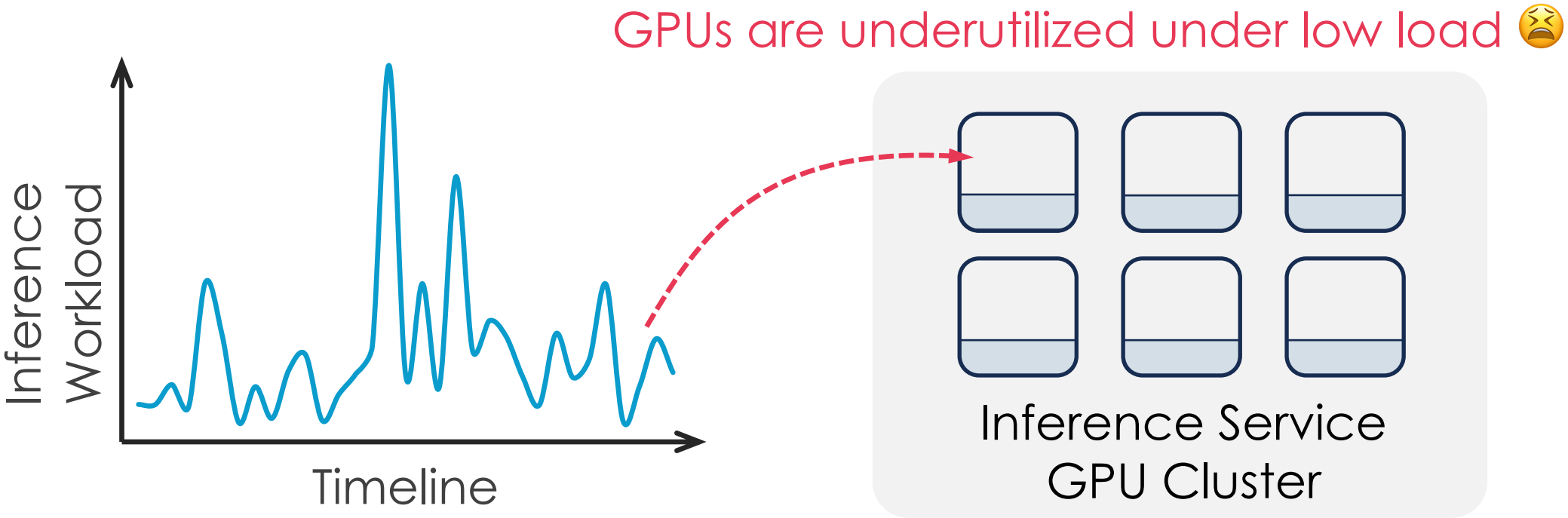
GPU underutilization in inference services

- Inference workloads are naturally **fluctuating**



GPU underutilization in inference services

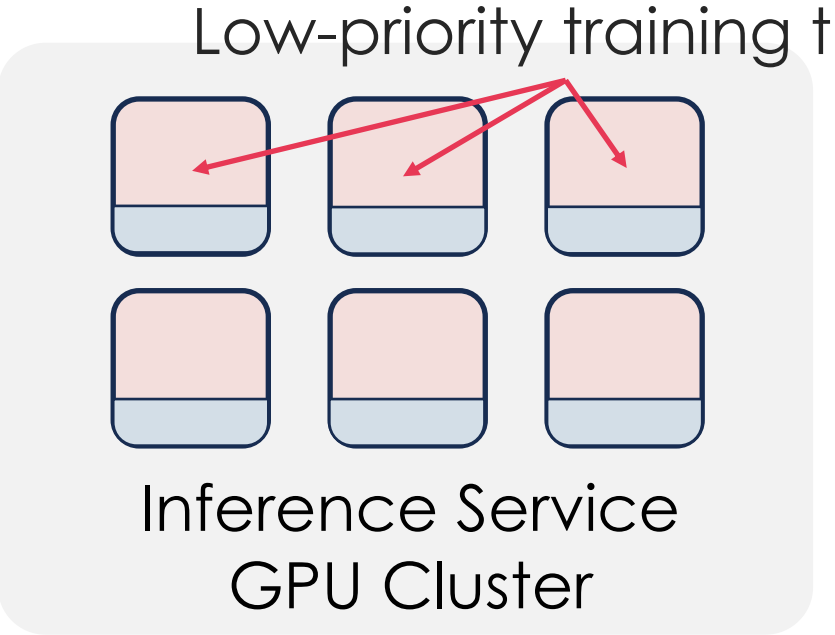
- Inference workloads are naturally **fluctuating**
- Over-provisioning GPUs is wasting valuable GPU cycles!



Eradicating GPU over-provision via Colocation

- Colocating ***latency-sensitive ML inference*** with ***throughput-intensive ML training***
- Inference and training tasks execute concurrently

GOAL: fully saturating GPUs, without compromising inference SLO and training throughput !!! 🤩



Eradicating GPU over-provision via Colocation

- Colocating ***latency-sensitive ML inference*** with ***throughput-***

Oops! GPU cannot hold both inference and training

Numerous Inference Models + Training States > GPU Memory

without compromising inference

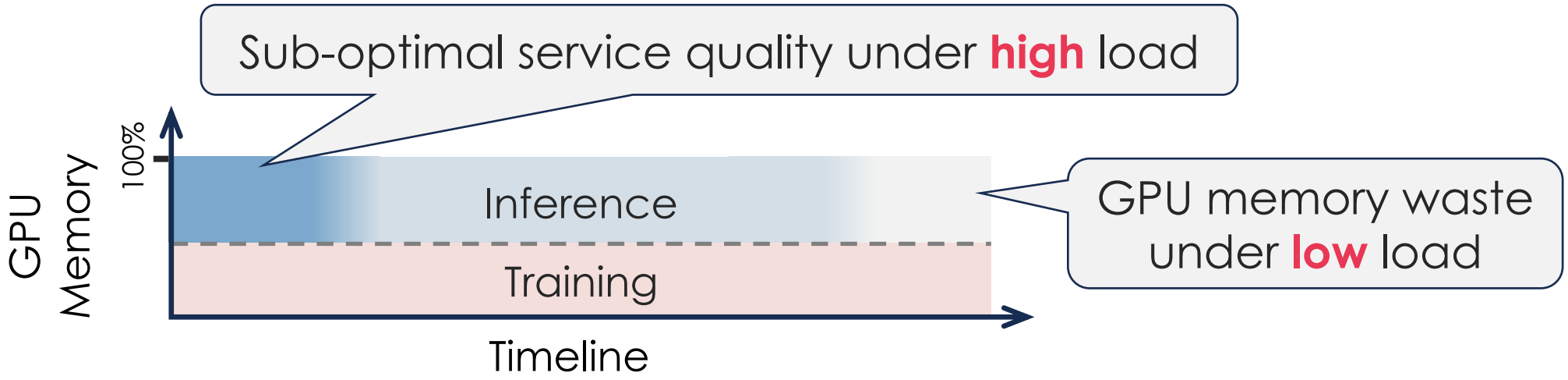
SLO and training throughput !!! 🤩



Inference Service
GPU Cluster

Existing approach: **Static Memory Partition**

- Statically partitioning GPU memory for inference and training tasks

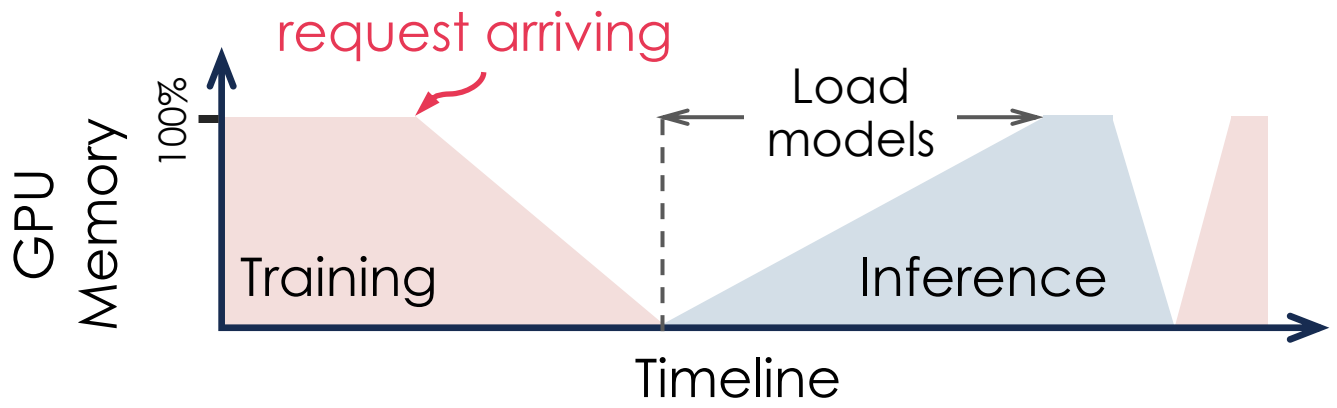


Inadaptability to dynamic workloads

⇒ mismatched between requirement and allocation

Existing approach: **Dynamic Memory Swap**

Software Task Switch



Context Switching overhead

⇒ *Inference cold starts, training starvation*

Hardware Unified Memory

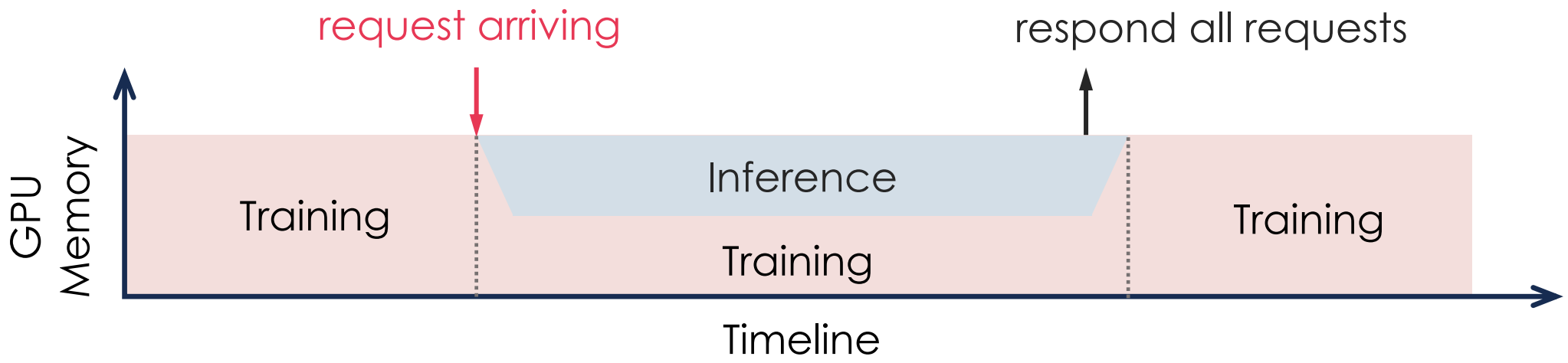


Substantial memory swaps

⇒ *degraded performance for both task*

Our approach: **Dynamic Memory Transfer**

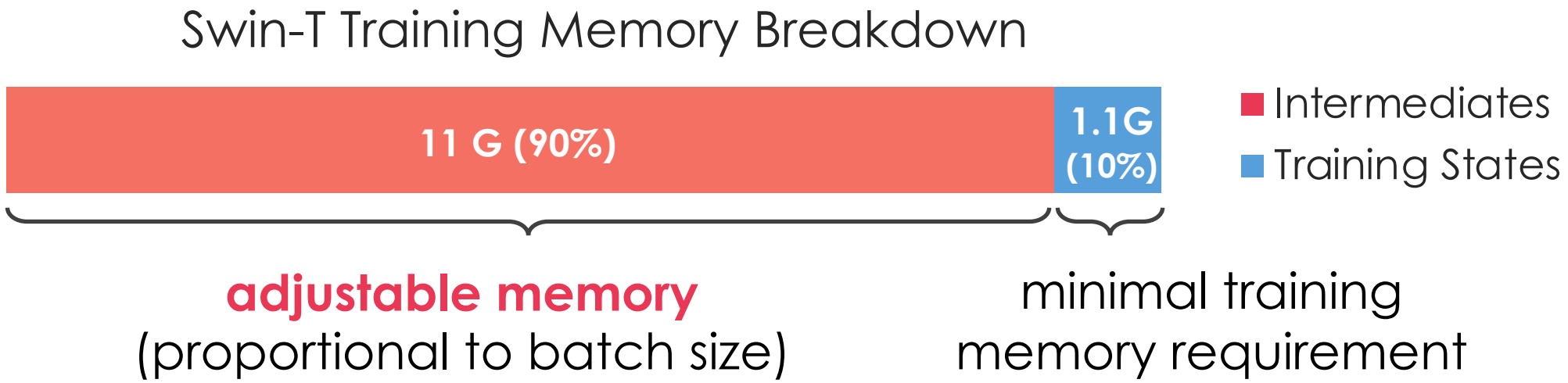
- Transferring memory between training and inference
  - Adapt to dynamic workload
  - Prioritizing inference tasks to use GPU memory



How to keep the training task executing when part of its memory is transferred?

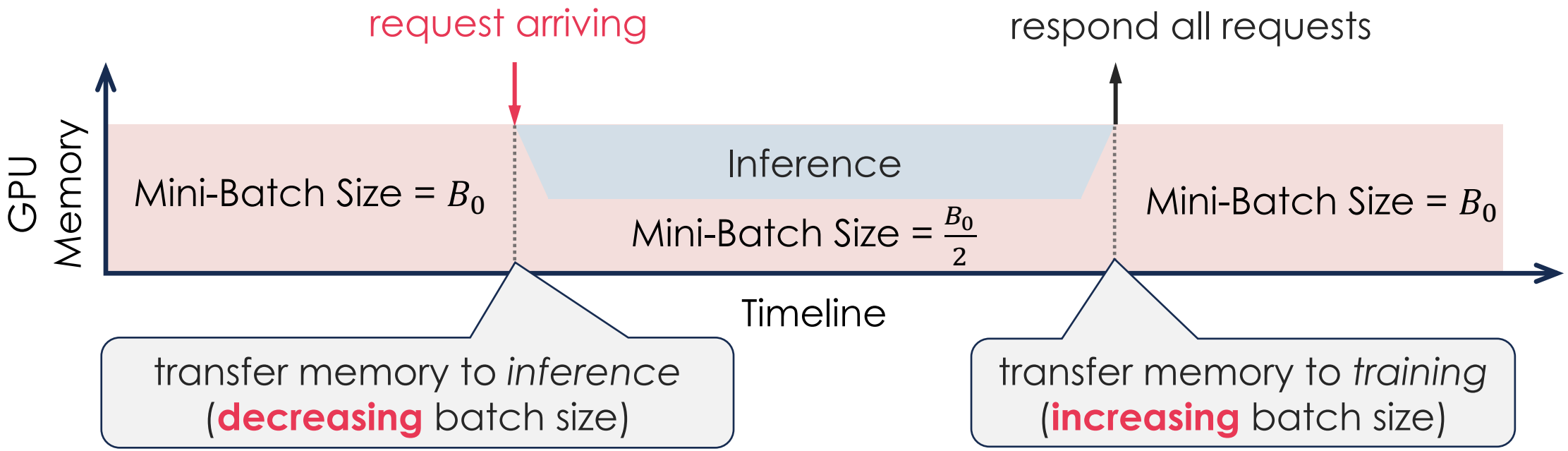
Key Insight: **Training Elasticity**

- Training task is **elastic** and **reconfigurable**
- Large training batch can be divided into mini-batches



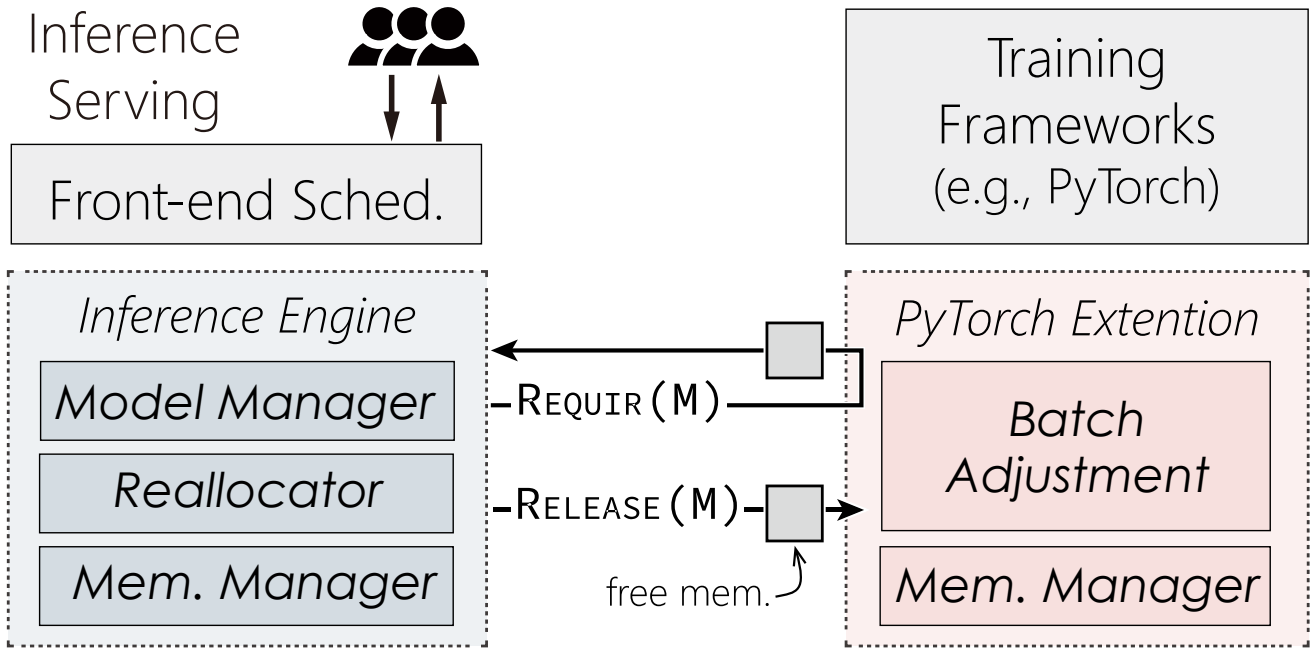
Key Insight: **Training Elasticity**

- Transferring memory between training and inference by reconfiguring training tasks (i.e., changing batch size)



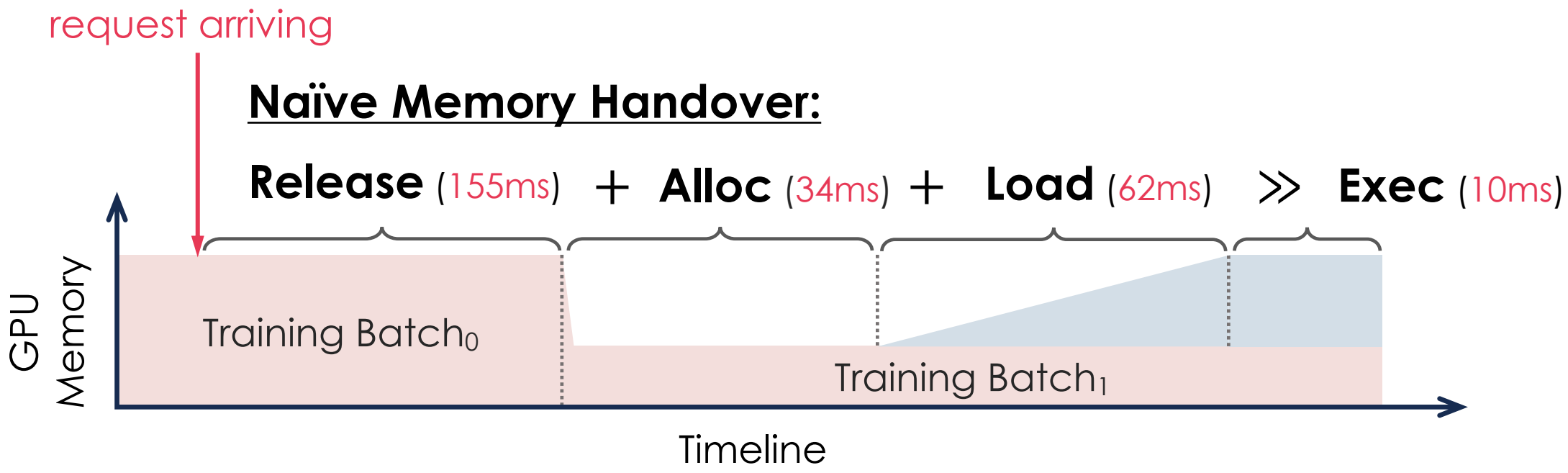
SIRIUS: ML Inference & Training Colocation System

Architecture of **SIRIUS**



Challenge: Fast Memory Handover

- Transferring training memory to inference takes a long time



Violating Inference SLO! 😫

Challenge: Fast Memory Handover

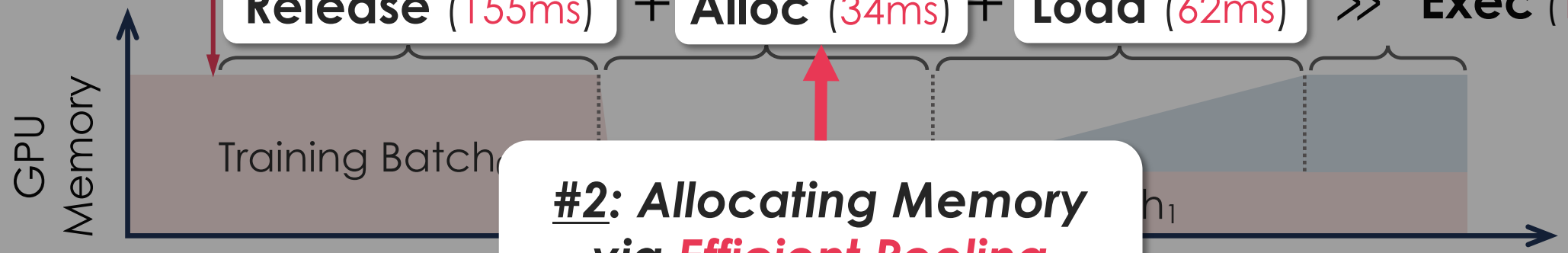
- T… mory to i… me

#1*: Releasing Training Task Memory *Instantly

***#3*: *SLO-aware* Allocation — Avoiding Cold Starts**

Naïve Memory Handover:

Release (155ms) + **Alloc** (34ms) + **Load** (62ms) ≫ **Exec** (10ms)



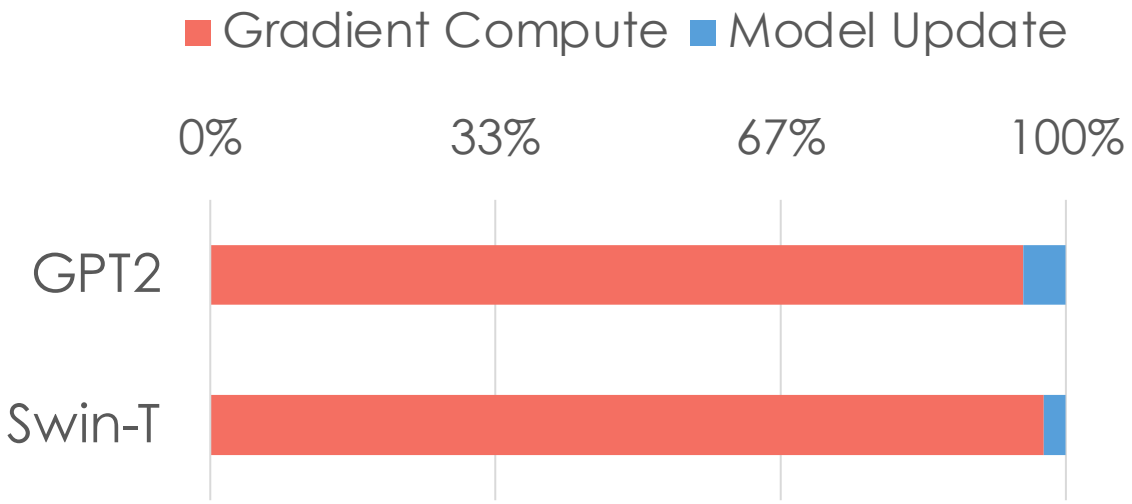
#2*: Allocating Memory via *Efficient Pooling

Violating Inference SLO! 😫

#1: Releasing Training Task Memory Instantly

Design Goal: Releasing the memory without completing the current training batch

Observation: time-consuming gradient compute NOT changes training state, atomic model update is short

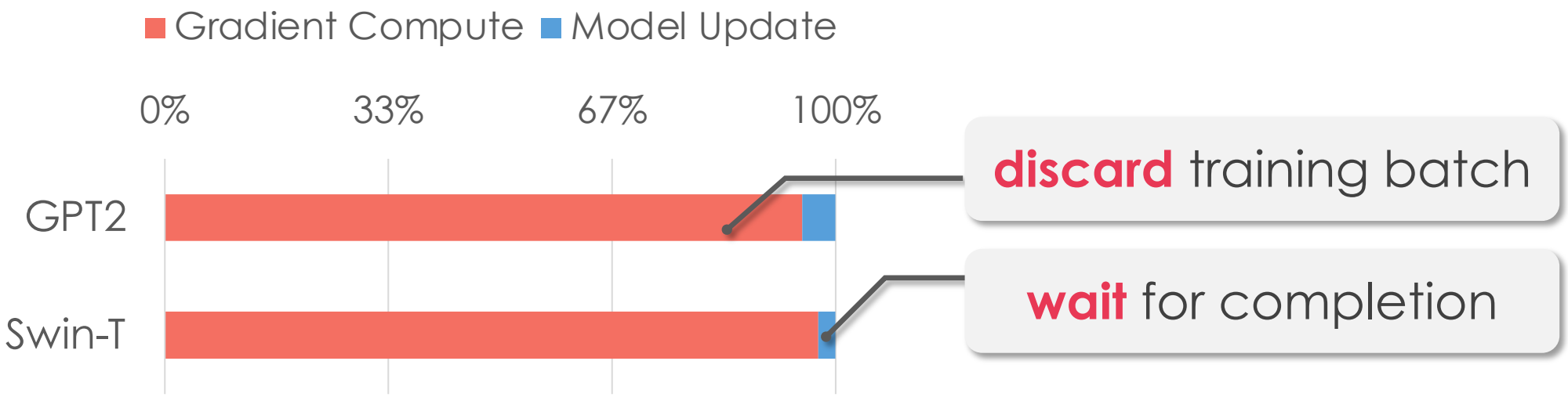



```
def Train(model,dataset):
  for batch in dataset:
    ComputeGrad(batch)
    UpdateModel(model)
```

#1: Releasing Training Task Memory Instantly

Our Approach

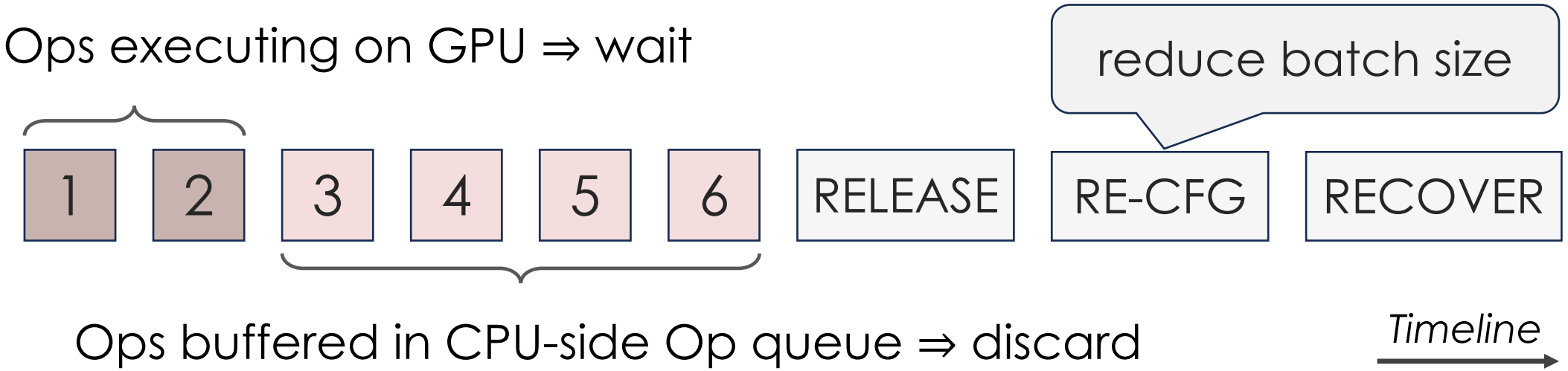
- *Gradient compute*: interrupt and discard the batch (<2ms)
- *Model update* : wait for completion (<10ms)



How to discard executing training batch?

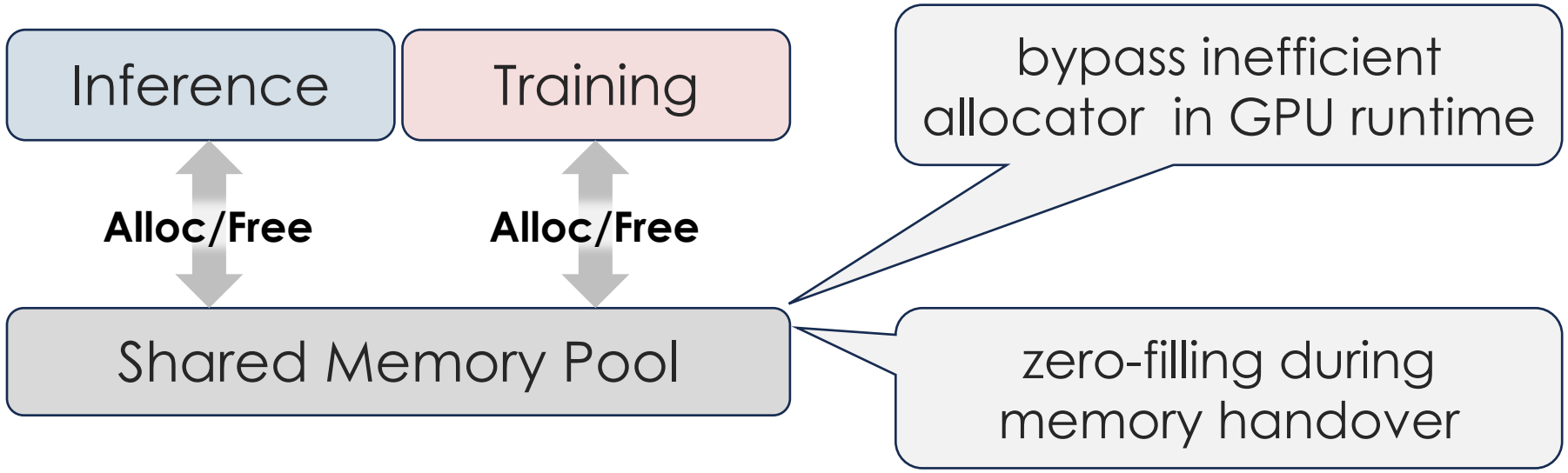
Problem: Training Ops on GPU cannot be cancelled

Solution: Limited the #Ops on GPU by buffering the asynchronous Ops in a CPU-side queue



#2: Allocating Memory via Efficient Pooling

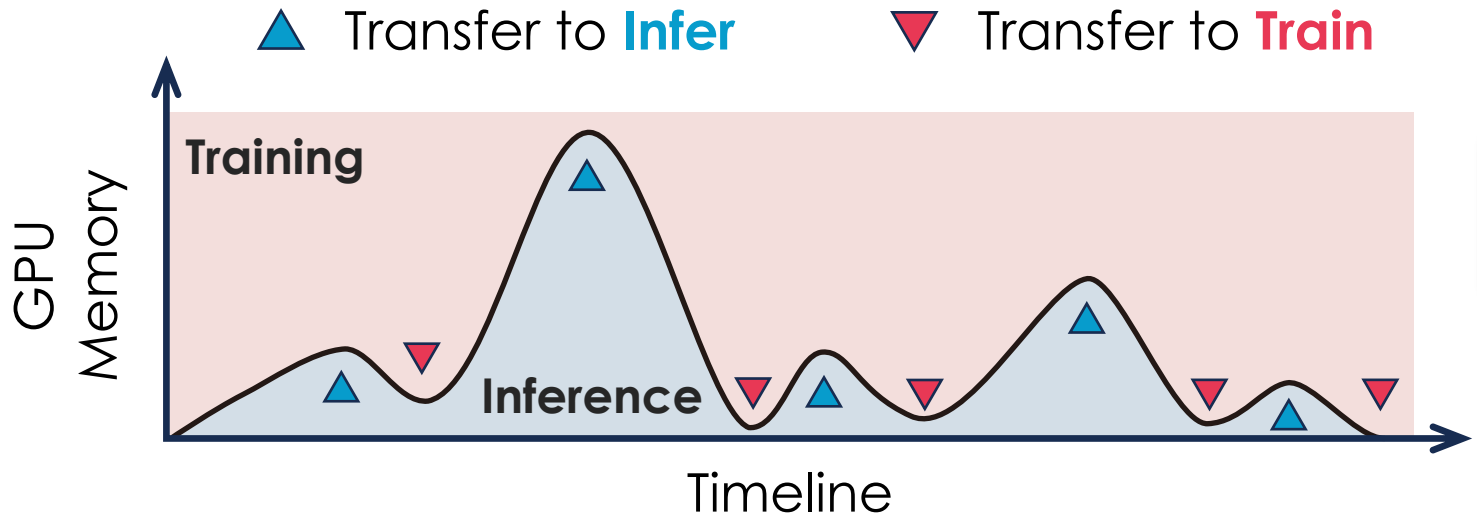
- A shared memory pool is necessary to speed up memory handover



How to bypass the memory pool in PyTorch? Check solutions in our paper !

#3: SLO-aware Allocation — Avoiding Cold Starts

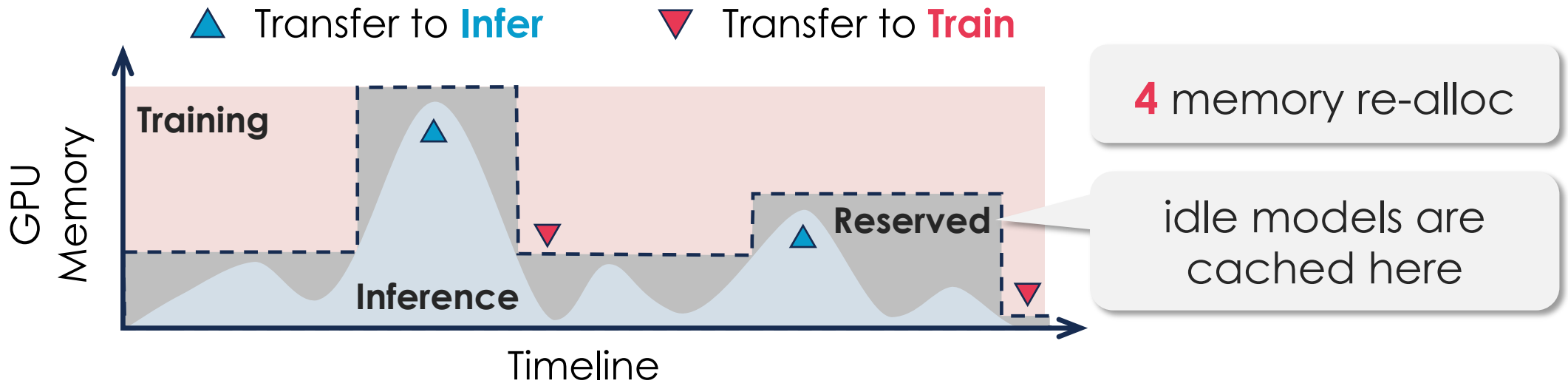
- Fluctuating workload leads to memory allocation thrashing
  - Leading to more inference model loading
  - hurting training throughput



10 memory re-alloc between infer/train

#3: SLO-aware Allocation — Avoiding Cold Starts

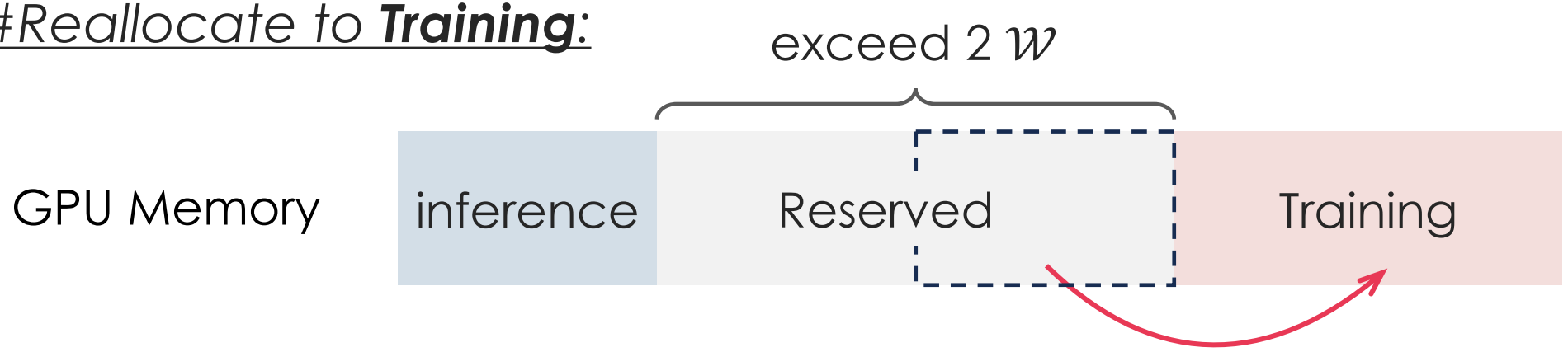
- Fluctuating workload leads to memory allocation thrashing
- Reserving little memory for inference to mitigate this



#3: SLO-aware Allocation — Avoiding Cold Starts

- Fluctuating workload leads to memory allocation thrashing
- Reserving little memory for inference to mitigate this
- Watermark $\mathcal{W}$ controls the allocation granularity

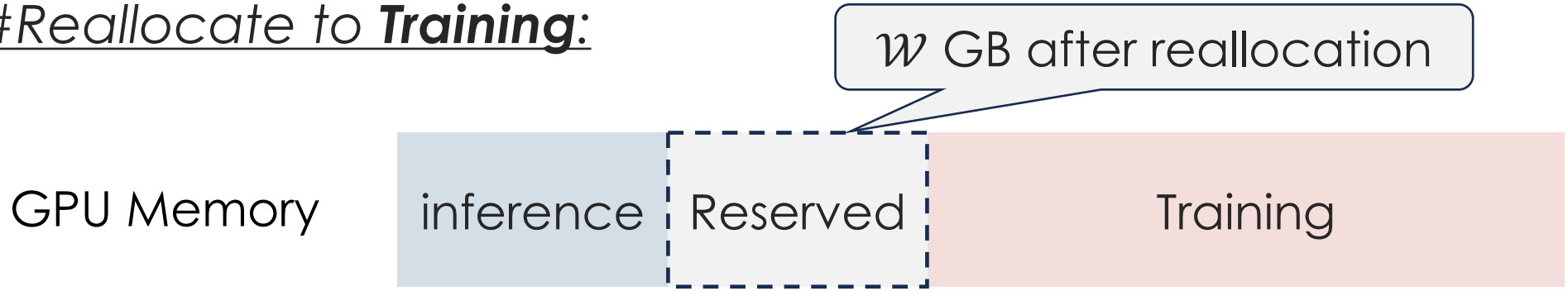
*#Reallocate to **Training**:*



#3: SLO-aware Allocation — Avoiding Cold Starts

- Fluctuating workload leads to memory allocation thrashing
- Reserving little memory for inference to mitigate this
- Watermark $\mathcal{W}$ controls the allocation granularity

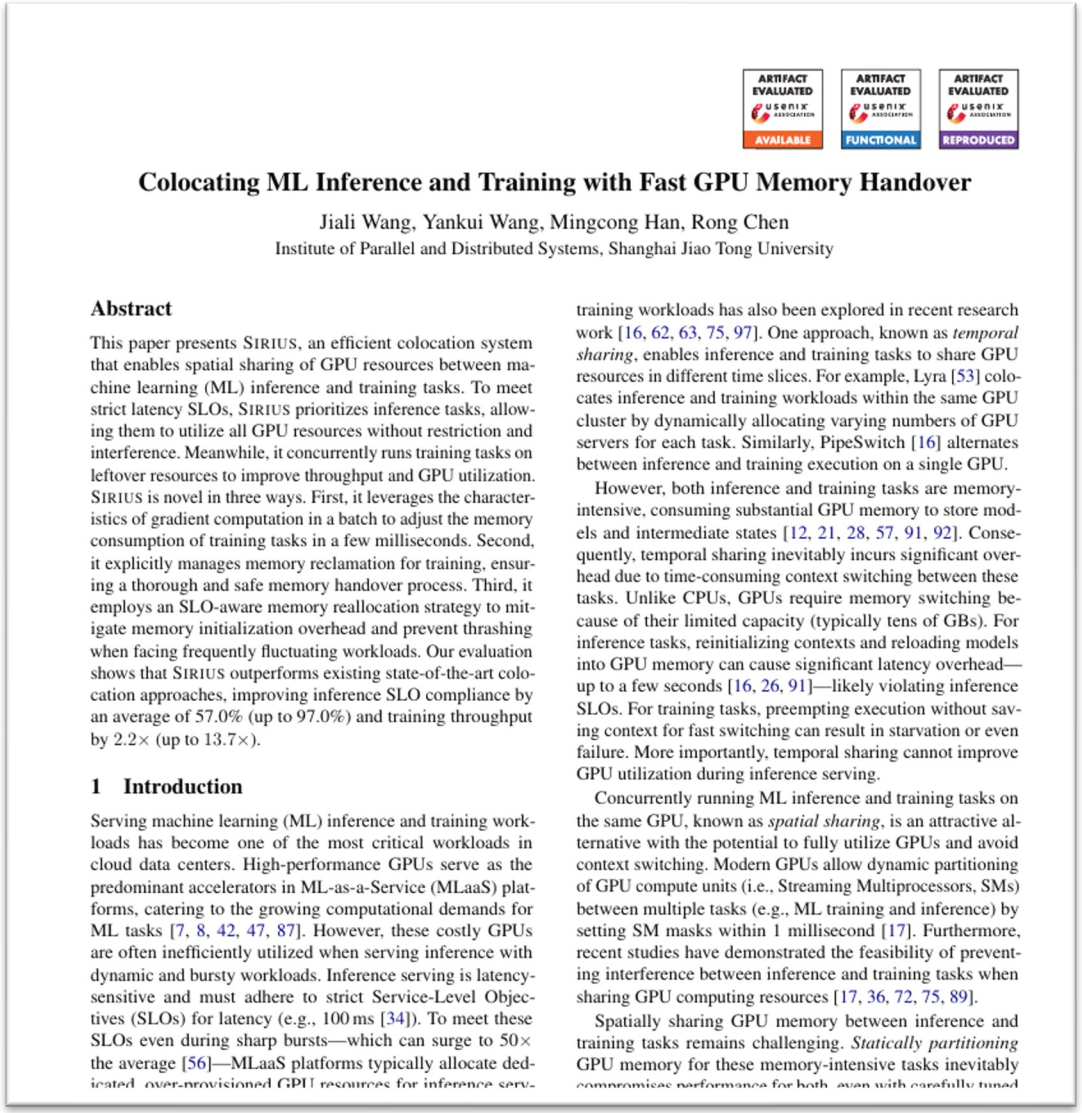
*#Reallocate to **Training**:*



*How to determine the watermark? Check **queueing theory model** in our paper!*

Techniques Not Covered

- Colocating inference and training on **Multiple GPUs**
- Colocating **LLM workloads**
- **Pipelining** model load and inference
- **Avoiding interference** on GPU computation resources



ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Colocating ML Inference and Training with Fast GPU Memory Handover

Jiali Wang, Yankui Wang, Mingcong Han, Rong Chen
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University

Abstract

This paper presents SIRIUS, an efficient colocation system that enables spatial sharing of GPU resources between machine learning (ML) inference and training tasks. To meet strict latency SLOs, SIRIUS prioritizes inference tasks, allowing them to utilize all GPU resources without restriction and interference. Meanwhile, it concurrently runs training tasks on leftover resources to improve throughput and GPU utilization. SIRIUS is novel in three ways. First, it leverages the characteristics of gradient computation in a batch to adjust the memory consumption of training tasks in a few milliseconds. Second, it explicitly manages memory reclamation for training, ensuring a thorough and safe memory handover process. Third, it employs an SLO-aware memory reallocation strategy to mitigate memory initialization overhead and prevent thrashing when facing frequently fluctuating workloads. Our evaluation shows that SIRIUS outperforms existing state-of-the-art colocation approaches, improving inference SLO compliance by an average of 57.0% (up to 97.0%) and training throughput by 2.2× (up to 13.7×).

1 Introduction

Serving machine learning (ML) inference and training workloads has become one of the most critical workloads in cloud data centers. High-performance GPUs serve as the predominant accelerators in ML-as-a-Service (MLaaS) platforms, catering to the growing computational demands for ML tasks [7, 8, 42, 47, 87]. However, these costly GPUs are often inefficiently utilized when serving inference with dynamic and bursty workloads. Inference serving is latency-sensitive and must adhere to strict Service-Level Objectives (SLOs) for latency (e.g., 100 ms [34]). To meet these SLOs even during sharp bursts—which can surge to 50× the average [56]—MLaaS platforms typically allocate dedicated, over-provisioned GPU resources for inference serv-

training workloads has also been explored in recent research work [16, 62, 63, 75, 97]. One approach, known as *temporal sharing*, enables inference and training tasks to share GPU resources in different time slices. For example, Lyra [53] colocates inference and training workloads within the same GPU cluster by dynamically allocating varying numbers of GPU servers for each task. Similarly, PipeSwitch [16] alternates between inference and training execution on a single GPU.

However, both inference and training tasks are memory-intensive, consuming substantial GPU memory to store models and intermediate states [12, 21, 28, 57, 91, 92]. Consequently, temporal sharing inevitably incurs significant overhead due to time-consuming context switching between these tasks. Unlike CPUs, GPUs require memory switching because of their limited capacity (typically tens of GBs). For inference tasks, reinitializing contexts and reloading models into GPU memory can cause significant latency overhead—up to a few seconds [16, 26, 91]—likely violating inference SLOs. For training tasks, preempting execution without saving context for fast switching can result in starvation or even failure. More importantly, temporal sharing cannot improve GPU utilization during inference serving.

Concurrently running ML inference and training tasks on the same GPU, known as *spatial sharing*, is an attractive alternative with the potential to fully utilize GPUs and avoid context switching. Modern GPUs allow dynamic partitioning of GPU compute units (i.e., Streaming Multiprocessors, SMs) between multiple tasks (e.g., ML training and inference) by setting SM masks within 1 millisecond [17]. Furthermore, recent studies have demonstrated the feasibility of preventing interference between inference and training tasks when sharing GPU computing resources [17, 36, 72, 75, 89].

Spatially sharing GPU memory between inference and training tasks remains challenging. *Statically partitioning* GPU memory for these memory-intensive tasks inevitably compromises performance for both, even with carefully tuned

Evaluation

- Inference Workload Traces
  - **Generated traces**: LIGHT, HEAVY, BURST, SKEWED
  - **Real-world traces**: Microsoft Azure Functions (MAF)
- Comparing Targets
  - **SP-50** (Static Partition: Infer 50%, Train 50%)
  - **SP-75** (Static Partition: Infer 75%, Train 25%)
  - **Task Switch**
  - **NVIDIA UM+MPS**

Performance on a single GPU (1xV100)

- Workload: **56 DNN instances** (6 different models, infer) + **Swin-T** (train)



Static Partition

Low **SLO compliance** (SP-50 40%, SP-75 86%)

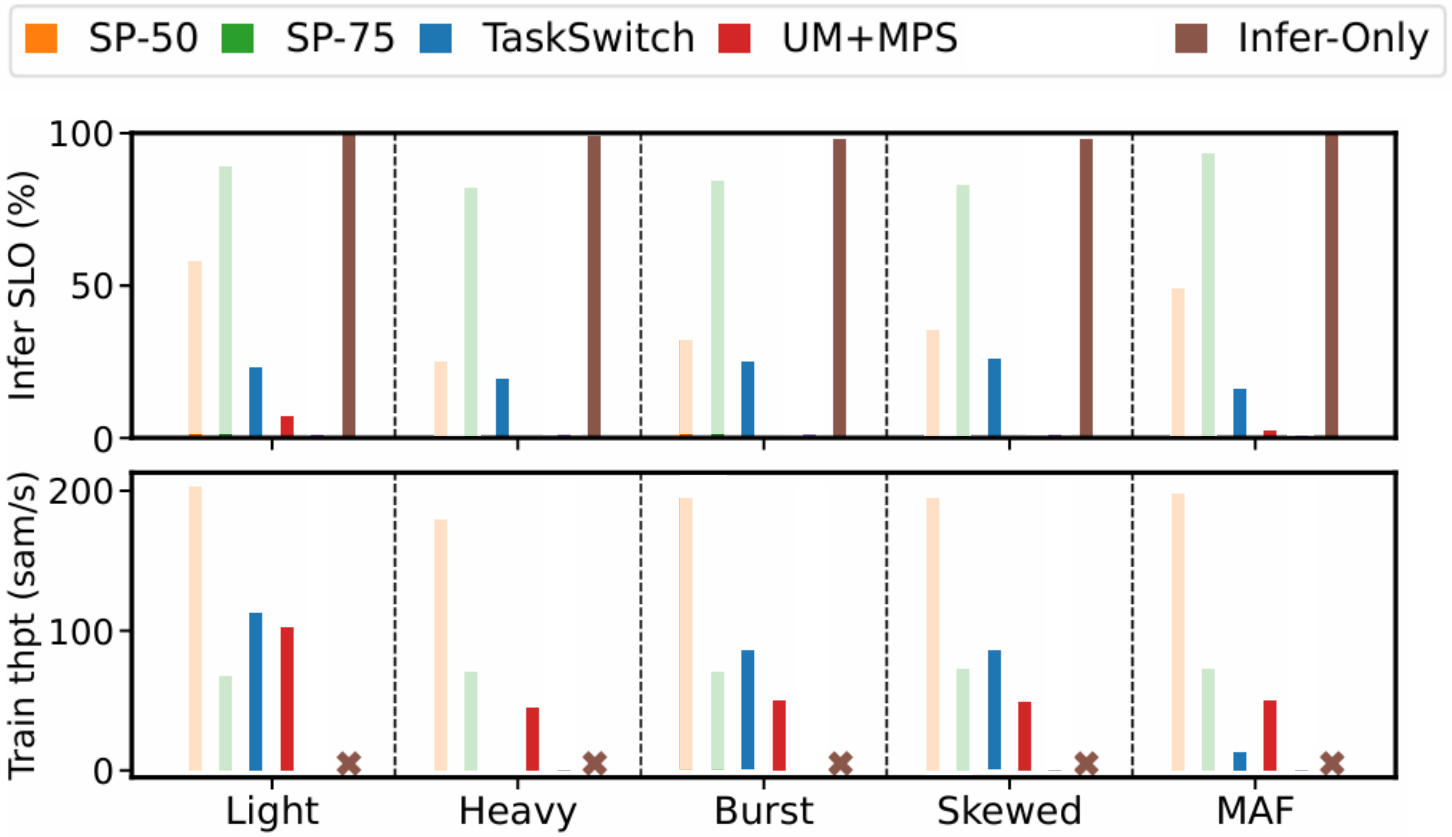
😫

Tradeoff **training thpt** (SP-75 2.8 × lower than SP-50)

SLO = 4x standalone inference time

Performance on a single GPU (1xV100)

- Workload: **56 DNN instances** (6 different models, infer) + **Swin-T** (train)



Task Switch & UM

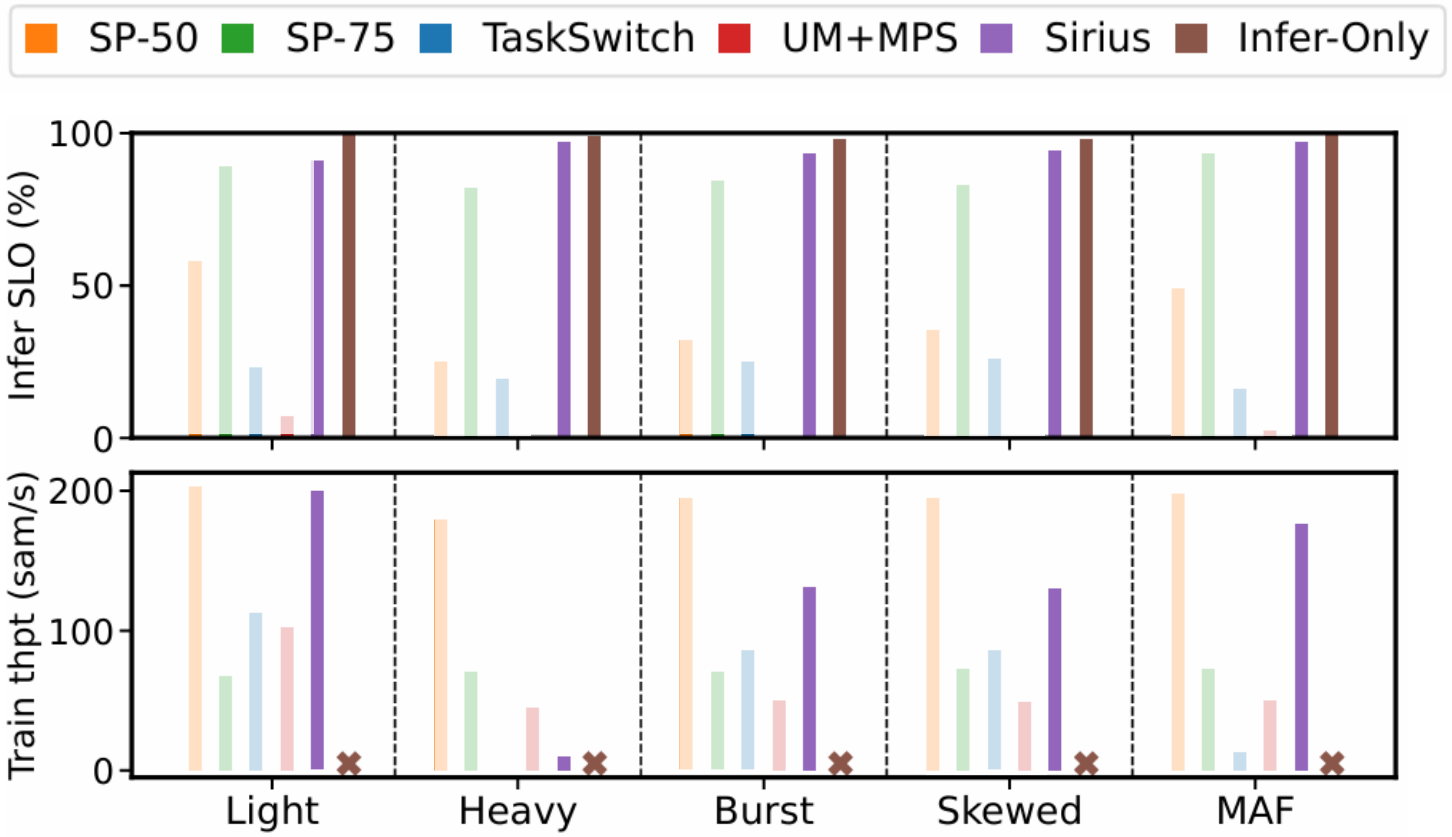
Low **SLO compliance**
(TaskSwitch 22%, UM 2%)

Low **training thpt**

SLO = 4x standalone inference time

Performance on a single GPU (1xV100)

- Workload: **56 DNN instances** (6 different models, infer) + **Swin-T** (train)



Sirius

Up to **98%** of SLO compliance of infer-only

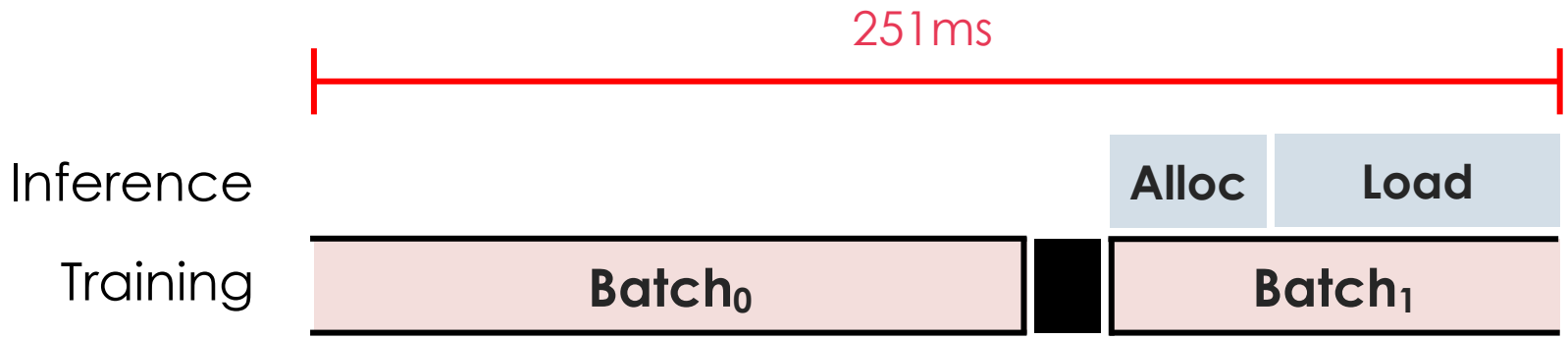
Improve **SLO compliance** by an average of **57%**

Improve **training thpt** by an average of **2.2×**

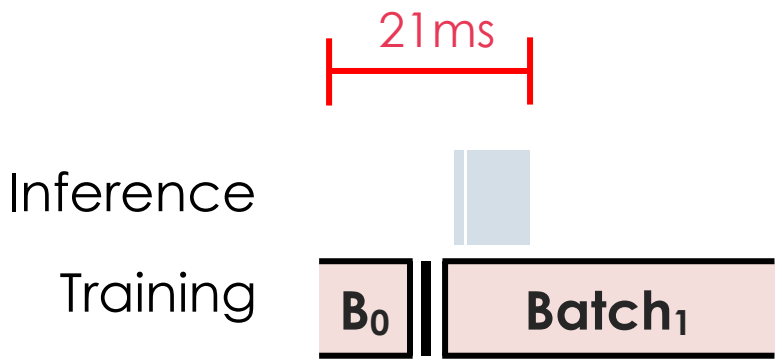
SLO = 4x standalone inference time

How Fast is Memory Handover (1xV100)?

Naïve Memory Handover:



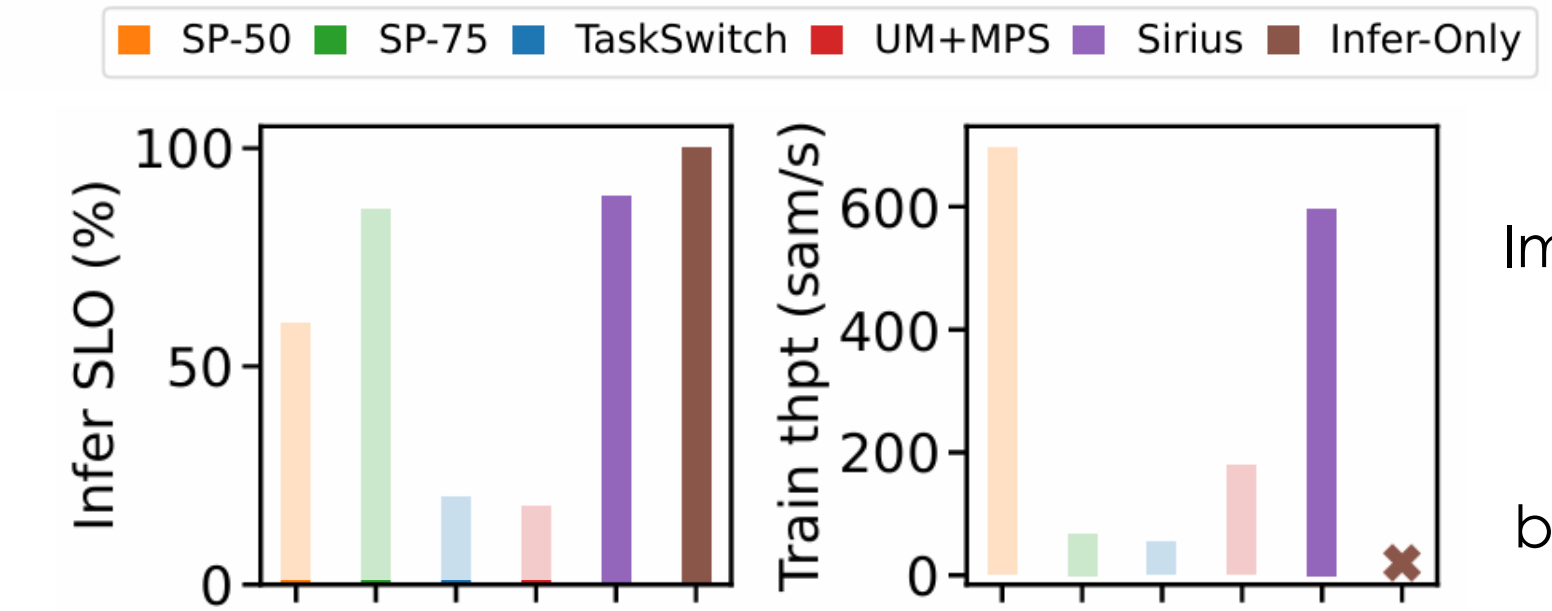
Fast Memory Handover:



Sirius's memory handover takes less than 25ms (11.9 × improvement)

Colocation on Multiple GPUs (4xV100)

- Workload: **56 DNN instances** (infer) + **Swin-T** (train) using LIGHT trace



Sirius

Improve **SLO compliance** by an average of **43%**

Improve **training thpt** by an average of **6.1×**

Conclusion



- **SIRIUS** — a colocation system that leverages training elasticity to dynamically share GPU memory
  - Saturating GPUs, without compromising inference SLOs
- Future work
  - Colocation on multi-GPU/multi-node
- SIRIUS is available at https://github.com/SiriusInfTra/Sirius