Swift and Trustworthy Large-Scale GPU Simulation with Fine-Grained Error Modeling and Hierarchical Clustering

Euijun Chung, Seonjin Na, Sung Ha Kang, Hyesoon Kim

Georgia Institute of Technology

2025 IEEE/ACM International Symposium on Microarchitecture

GPU microarchitecture simulation

Cycle-level simulations enable **fast validation** of new (micro)architecture designs.

Architectural design changes

Cycle-level GPU Simulator

IPC
Cache Hit Rate
Num of Instrs
Power usage
…

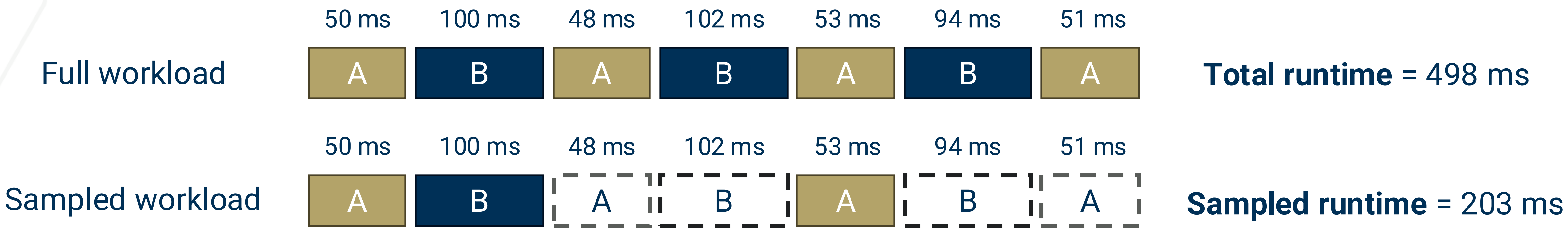
Problem: Cycle-level simulators are too slow!

✓ A 1-second workload on a real GPU can take **several days** on a simulator.

- For trace-based simulators, **trace size** grows along with **workload size**.

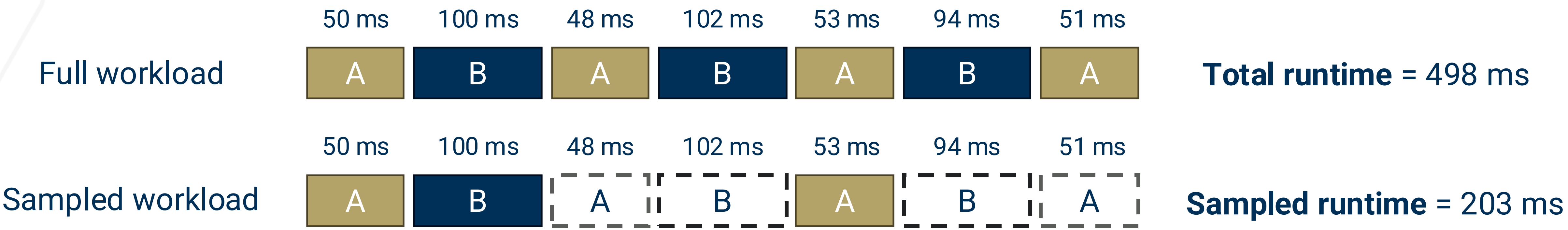
Kernel-level sampling for GPU workloads

- **Kernel-level sampling**: reducing workload size by sampling important kernels.
- **Idea**: Instead of running the full workload, **skip** the repeating kernels.
  - **Pros:** Simulation acceleration, reduced trace size / **Cons:** Simulation accuracy



Kernel-level sampling for GPU workloads

- **Kernel-level sampling**: reducing workload size by sampling important kernels.
- **Idea**: Instead of running the full workload, **skip** the repeating kernels.
  - **Pros:** Simulation acceleration, reduced trace size / **Cons:** Simulation accuracy

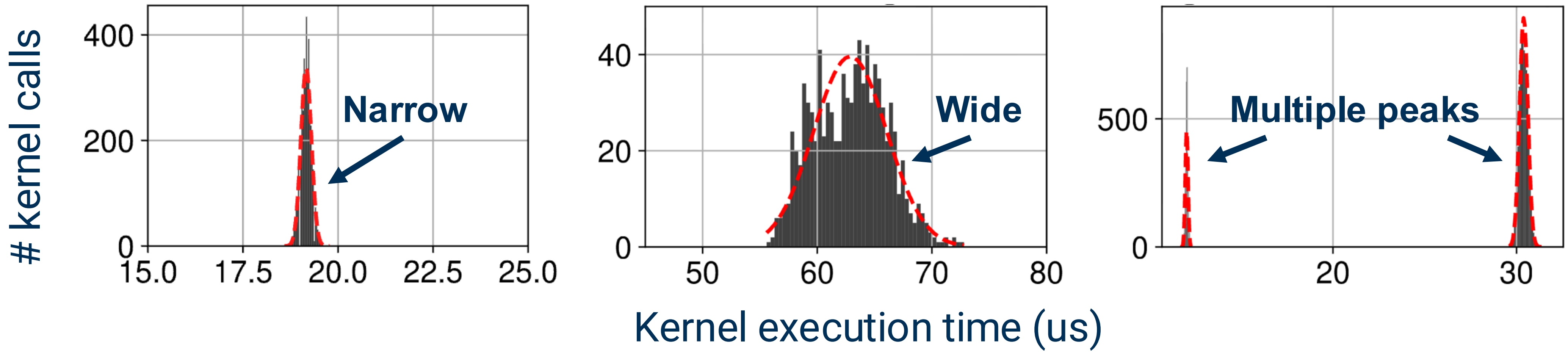


Tradeoff on **speedup** and **accuracy**:
More kernel samples make the sampled simulation **longer** but **accurate.**

GPU Kernels' execution time distributions

Observation: Identical GPU kernels show huge variation across invocations.

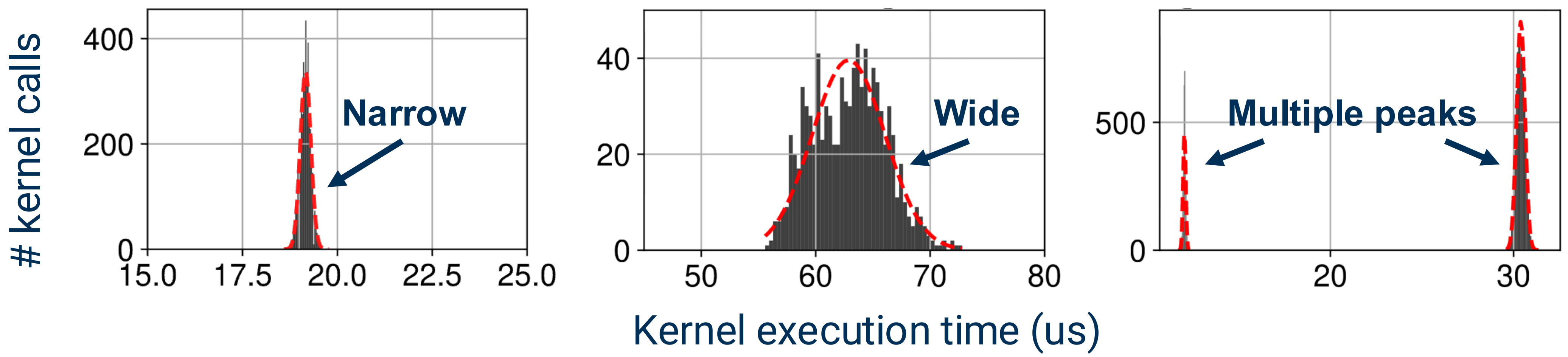
Idea: Leverage **kernel exe. time distributions** as a key signature to sample kernels.



GPU Kernels' execution time distributions

Observation: Identical GPU kernels show huge variation across invocations.

Idea: Leverage **kernel exe. time distributions** as a key signature to sample kernels.

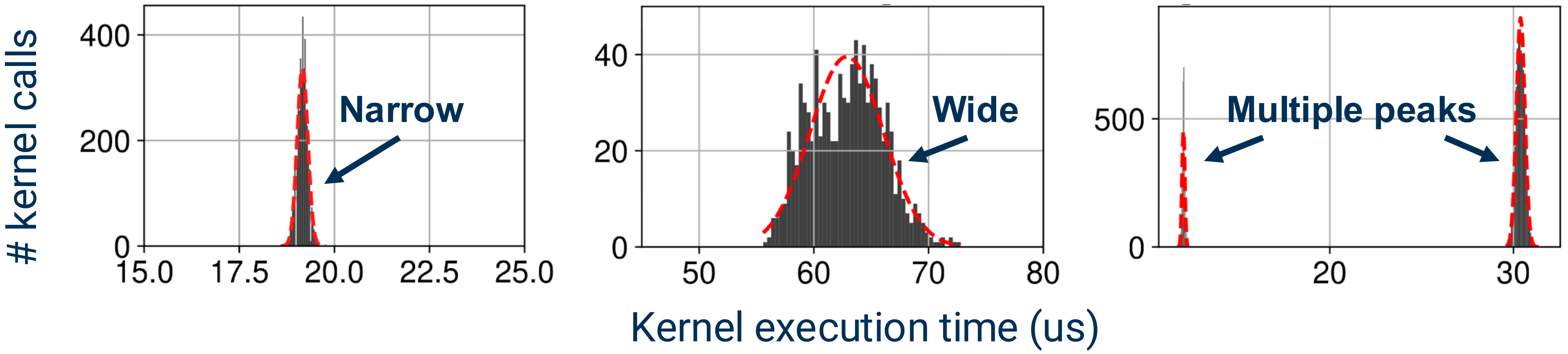


- **Narrow:** constant exe. time → **less** samples
- **Wide:** variable performance → **more** samples
- **Multiple:** kernel in multiple contexts → **separate** peaks into clusters *then* sample

GPU Kernels' execution time distributions

Observation: Identical GPU kernels show huge variation across invocations.

Idea: Leverage **kernel exe. time distributions** as a key signature to sample kernels.

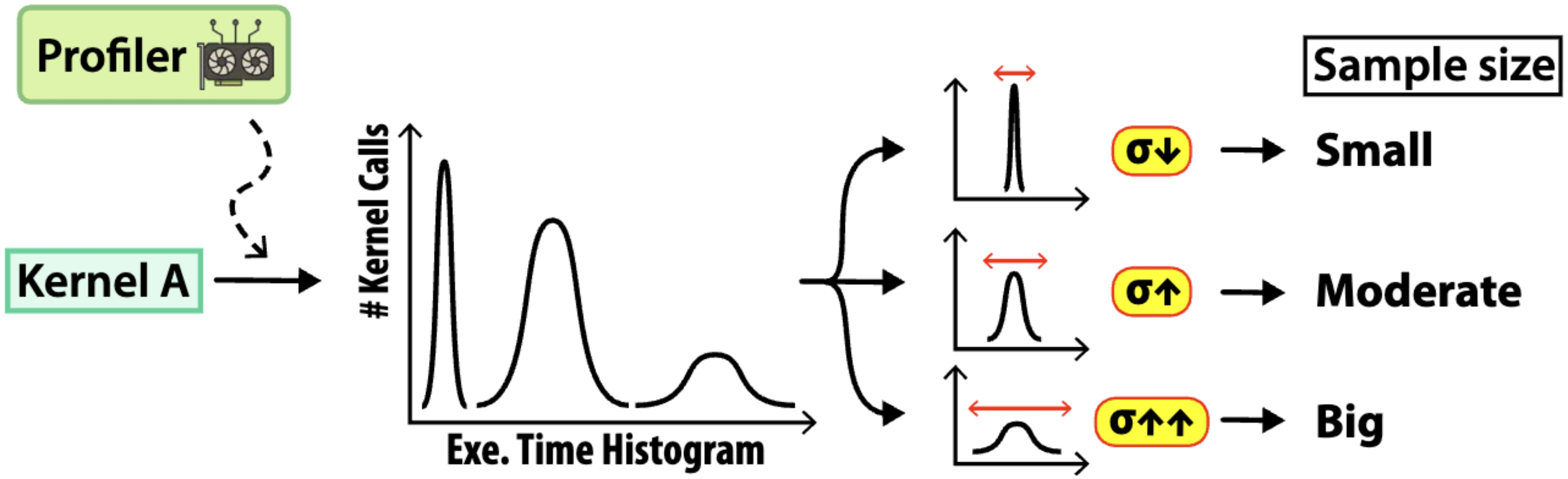


Question 1: Based on their distribution, **how many kernels** to sample?
Question 2: How to **maximize the speedup** while **the error is minimal?**

Determining the sample size

Question 1: Based on their distribution, **how many kernels** to sample?

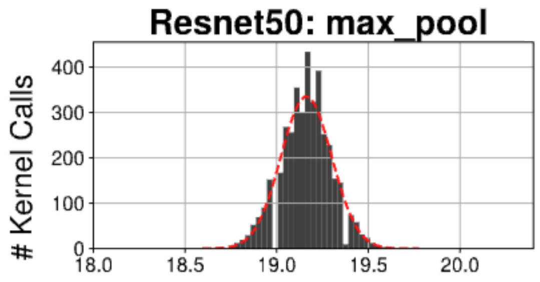
***Solution*: Statistical approach** based on kernel profiles.



Adaptive sample size → **speedup is maximized** while **sampling error is minimal.**

Applying the Central Limit Theorem (CLT)

Central Limit Theorem: **The mean of samples** will *always* follow a **Gaussian distribution** as the sample size $m \to \infty$.

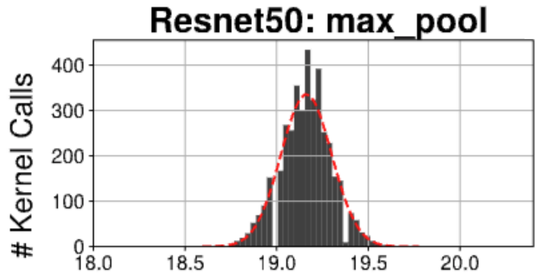


m kernel samples

Average kernel execution time follows a Gaussian distribution $\boldsymbol{\bar{X} \sim N(\mu, \sigma^2/m)}$.

Applying the Central Limit Theorem (CLT)

Central Limit Theorem: **The mean of samples** will *always* follow a **Gaussian distribution** as the sample size $m \rightarrow \infty$.



m kernel samples

Average kernel execution time follows a Gaussian distribution $\bar{X} \sim N(\mu, \sigma^2/m)$.

We **analytically calculate** the relationship between the sample size (m) and the error (e).

- The **minimum number of samples** to ensure the error bound ϵ:

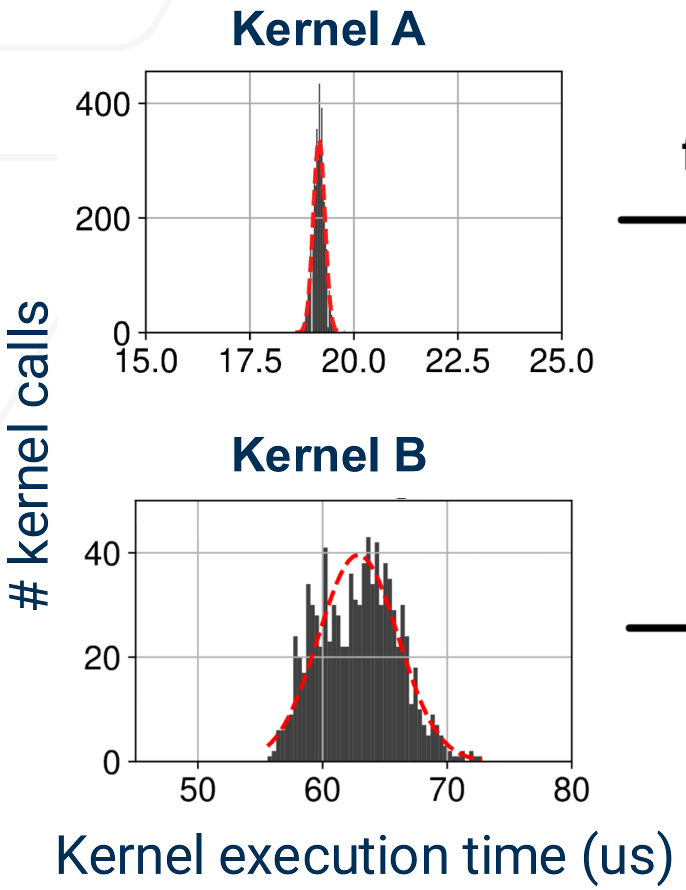
$$e = \left| \frac{|C|\bar{X} - |C|\mu}{|C|\mu} \right| = \left| \frac{\mu \pm \frac{z_{1-\alpha/2}\sigma}{\sqrt{m}} - \mu}{\mu} \right| = \frac{z_{1-\alpha/2}\sigma}{\mu\sqrt{m}} \leq \epsilon$$

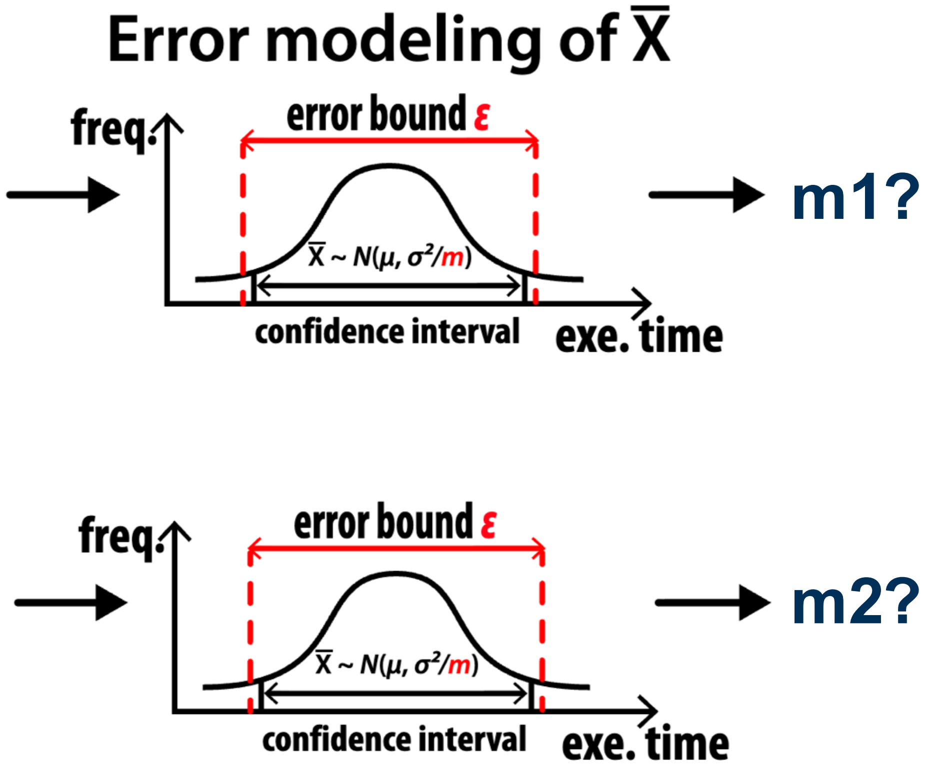
Error bound (e.g. 5%)

Assuming Gaussian distribution

STEM: Statistical Error Model for kernel sampling

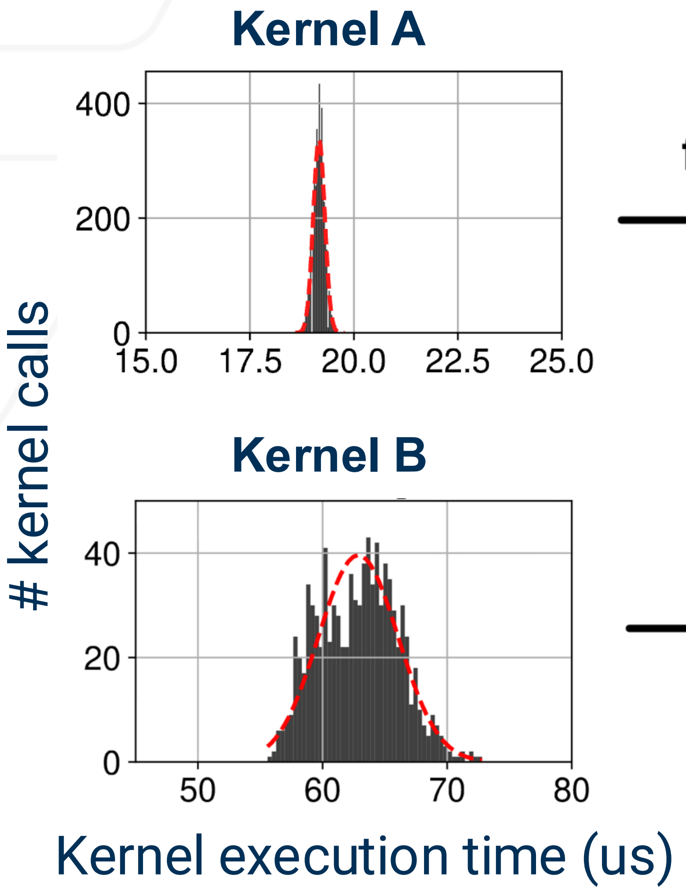
Optimizing for Multiple Kernels: minimize **sim. time** while the **total error is bounded.**

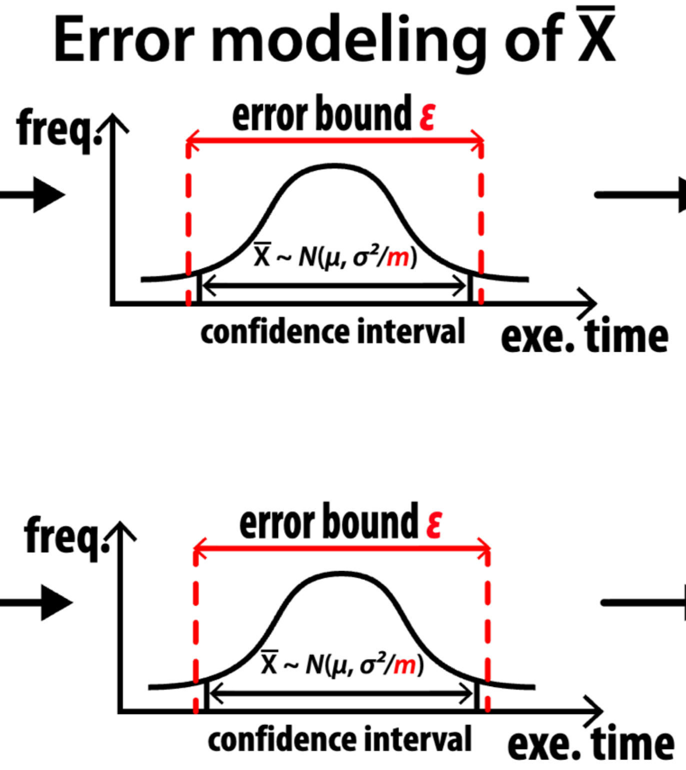




STEM: Statistical Error Model for kernel sampling

Optimizing for Multiple Kernels: minimize **sim. time** while the **total error is bounded.**





m1?

m2?

$$
\begin{aligned}
\underset{m_i}{\text{minimize}} \quad & \tau = \sum_i m_i \mu_i \\
\text{subject to} \quad & \sum_i N_i^2 \frac{\sigma_i^2}{m_i} \le \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i \mu_i \right)^2 \\
\text{and} \quad & m_i > 0 \ \text{ for } \forall i \in \{0, \ldots, k-1\}.
\end{aligned}
$$

STEM: Statistical Error Model for kernel sampling

Optimizing for Multiple Kernels: minimize **sim. time** while the **total error is bounded.**

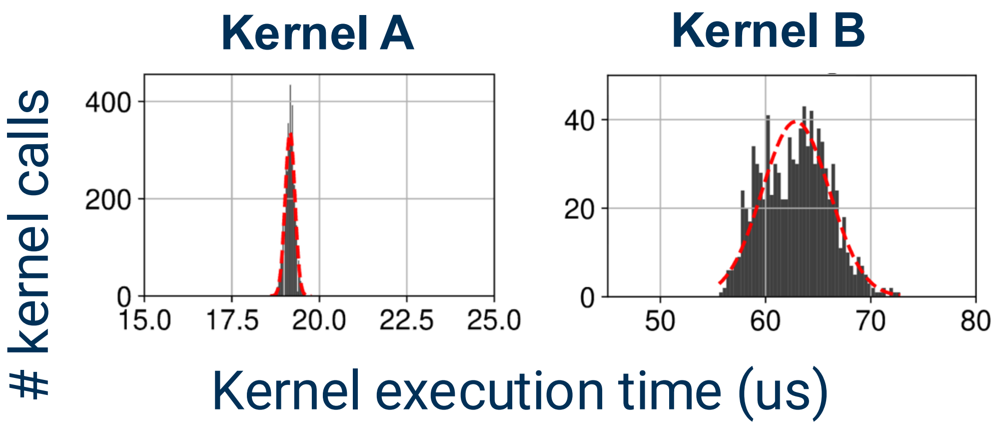
$$\begin{aligned} &\underset{m_i}{\text{minimize}} && \tau = \sum_i m_i \mu_i \\ &\text{subject to} && \sum_i N_i^2 \frac{\sigma_i^2}{m_i} \leq \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i \mu_i \right)^2 \\ &\text{and} && m_i > 0 \ \text{for} \ \forall i \in \{0, \dots, k-1\}. \end{aligned}$$

→

KKT Solver

$$\boxed{\begin{gathered}\text{minimize } \mathbf{m}: \\ \nabla \mathcal{L}(\mathbf{m}^*; \lambda) = \mathbf{0}\end{gathered}}$$

→



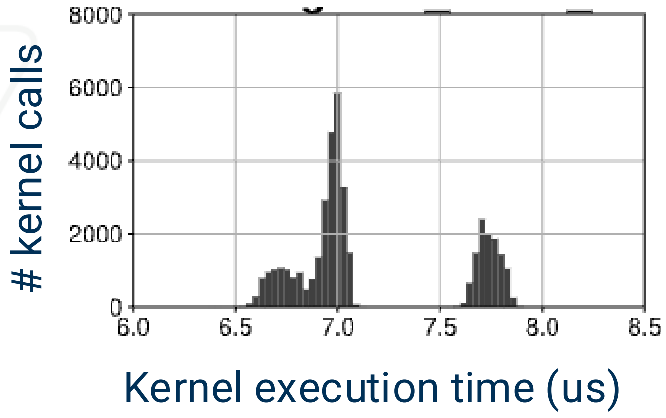
STEM's output:
- **Kernel A:** 10 samples
- **Kernel B:** 50 samples

Georgia Tech

Optimizing STEM for runtime-heterogeneous kernels

Problem: Some kernels favor **splitting before sampling with STEM.**

Goal: Distinguish each peak into **separate clusters** before sampling.



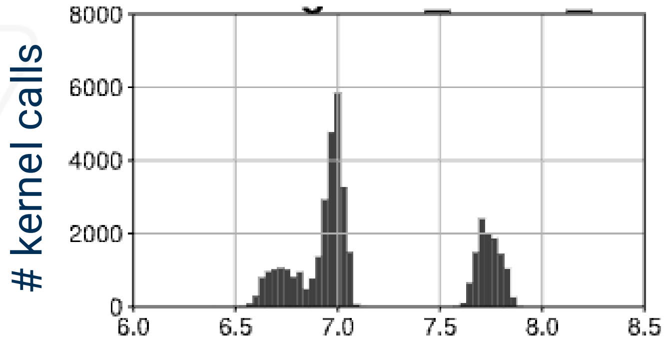
STEM

Sample **100** kernels

Optimizing STEM for runtime-heterogeneous kernels

Problem: Some kernels favor **splitting before sampling with STEM.**

Goal: Distinguish each peak into **separate clusters** before sampling.

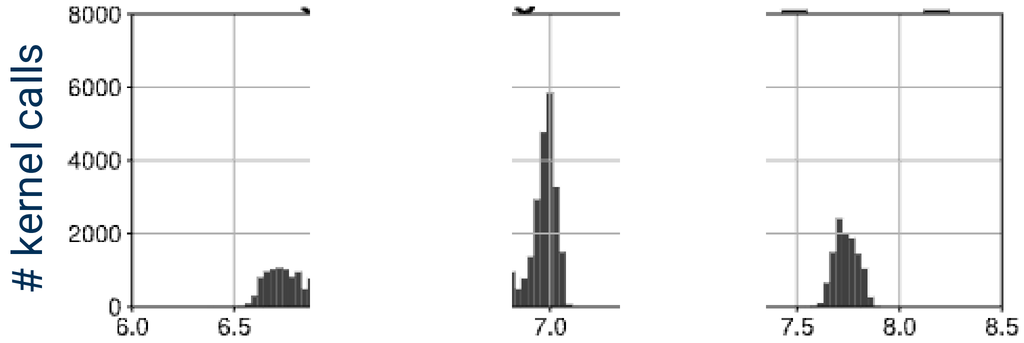


Kernel execution time (us)

STEM

Sample **100** kernels

Clustering



Kernel execution time (us)

STEM | STEM | STEM

15 kernels | **10** kernels | **10** kernels

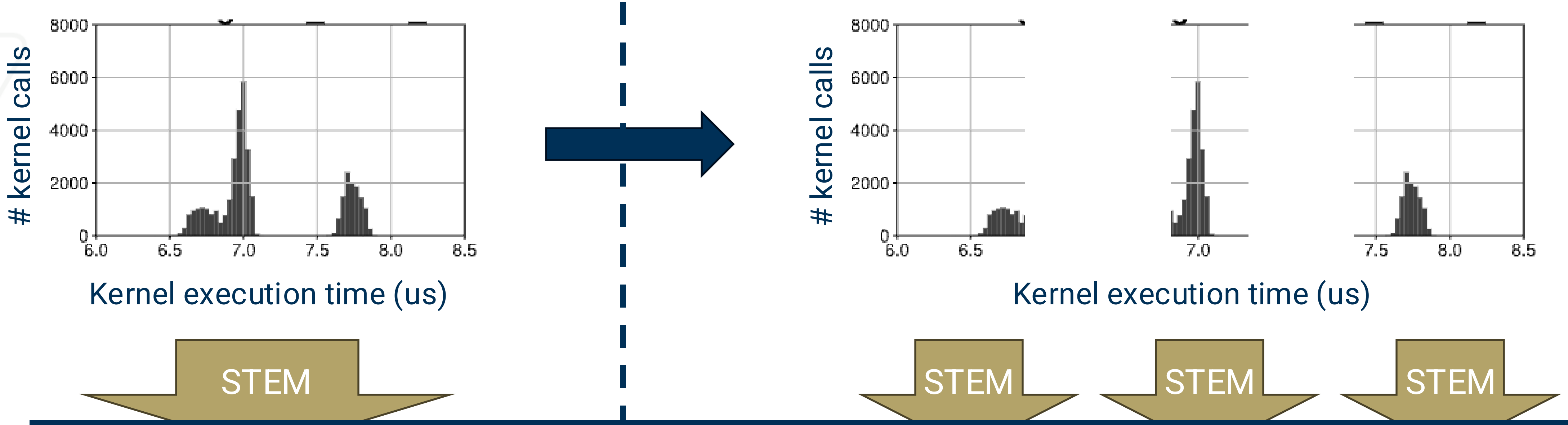
→ **35** kernels in total!

Georgia Tech

Optimizing STEM for runtime-heterogeneous kernels

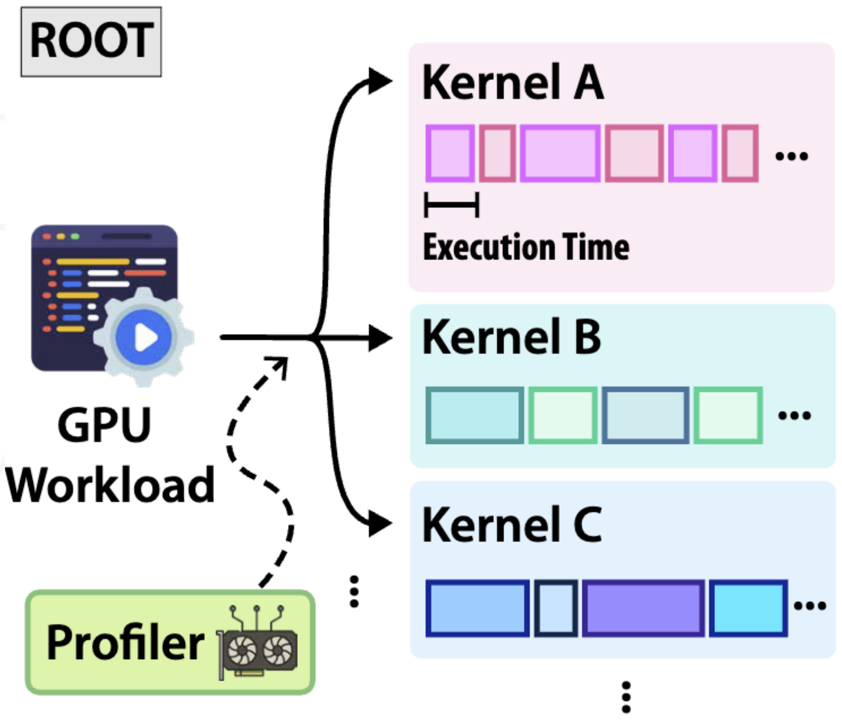
Problem: Some kernels favor **splitting before sampling with STEM.**

Goal: Distinguish each peak into **separate clusters** before sampling.

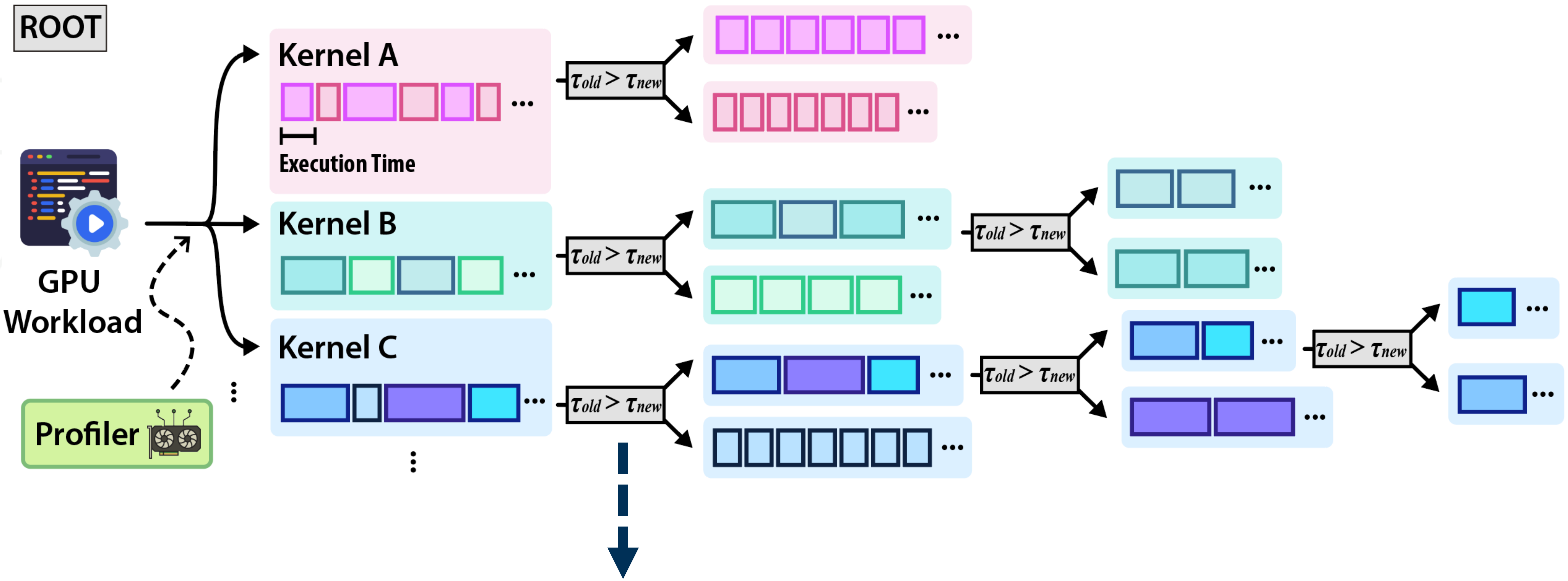


Question 1: The optimal number of subclusters is unknown.
Question 2: How to **optimize** clustering for sampling?

ROOT: Fine-grained hierarchical kernel clustering

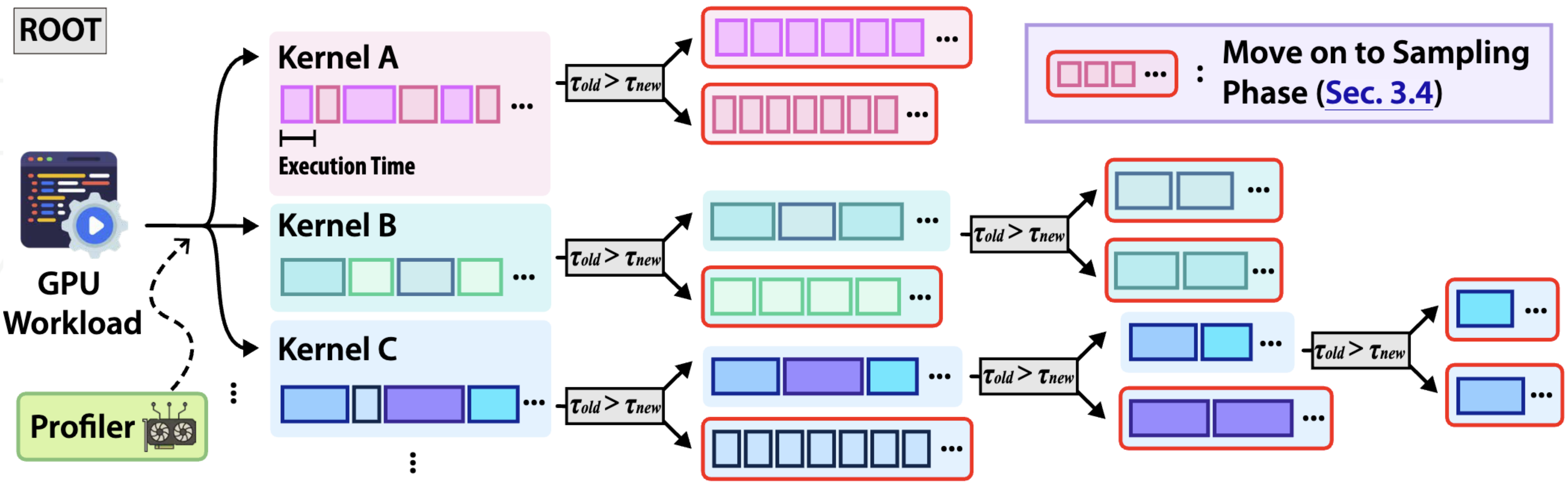


Hierarchical clustering of ROOT



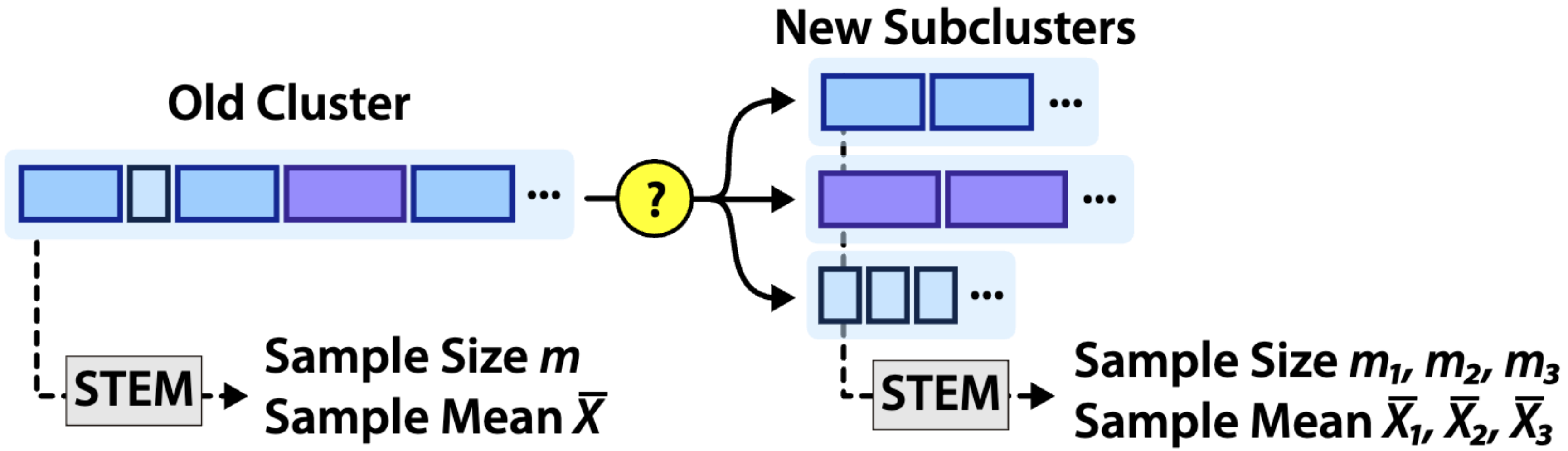
At every step: Check whether the splitting helps reducing sim. time

Hierarchical clustering of ROOT



Deriving the ROOT

ROOT leverages **STEM** to estimate whether splitting will help on kernel sampling.



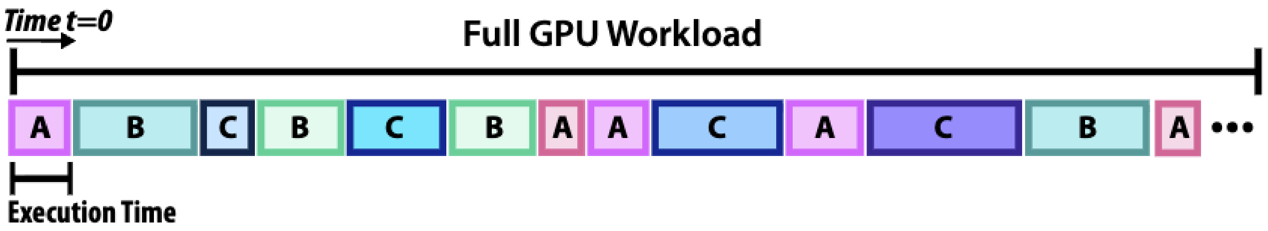
Compare the simulation time (τ)*: $\tau_{old} = m\bar{X}$

$$\tau_{new} = \sum_i m_i \bar{X}_i$$

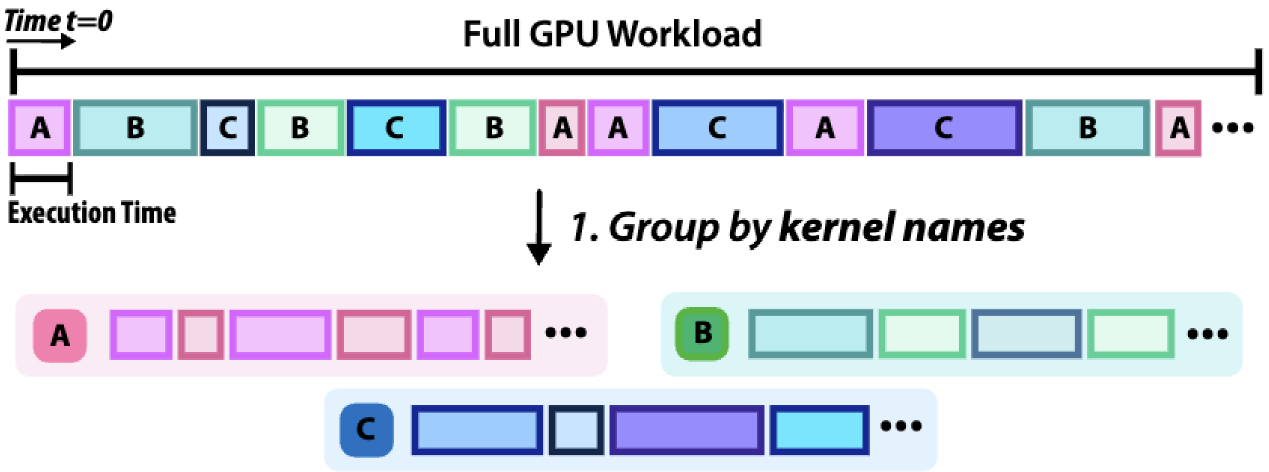
→ If $\tau_{\text{old}} > \tau_{new}$,
we can save simulation time.

* We use kernel latency as a heuristic for estimating the simulation time.

Summary of STEM + ROOT

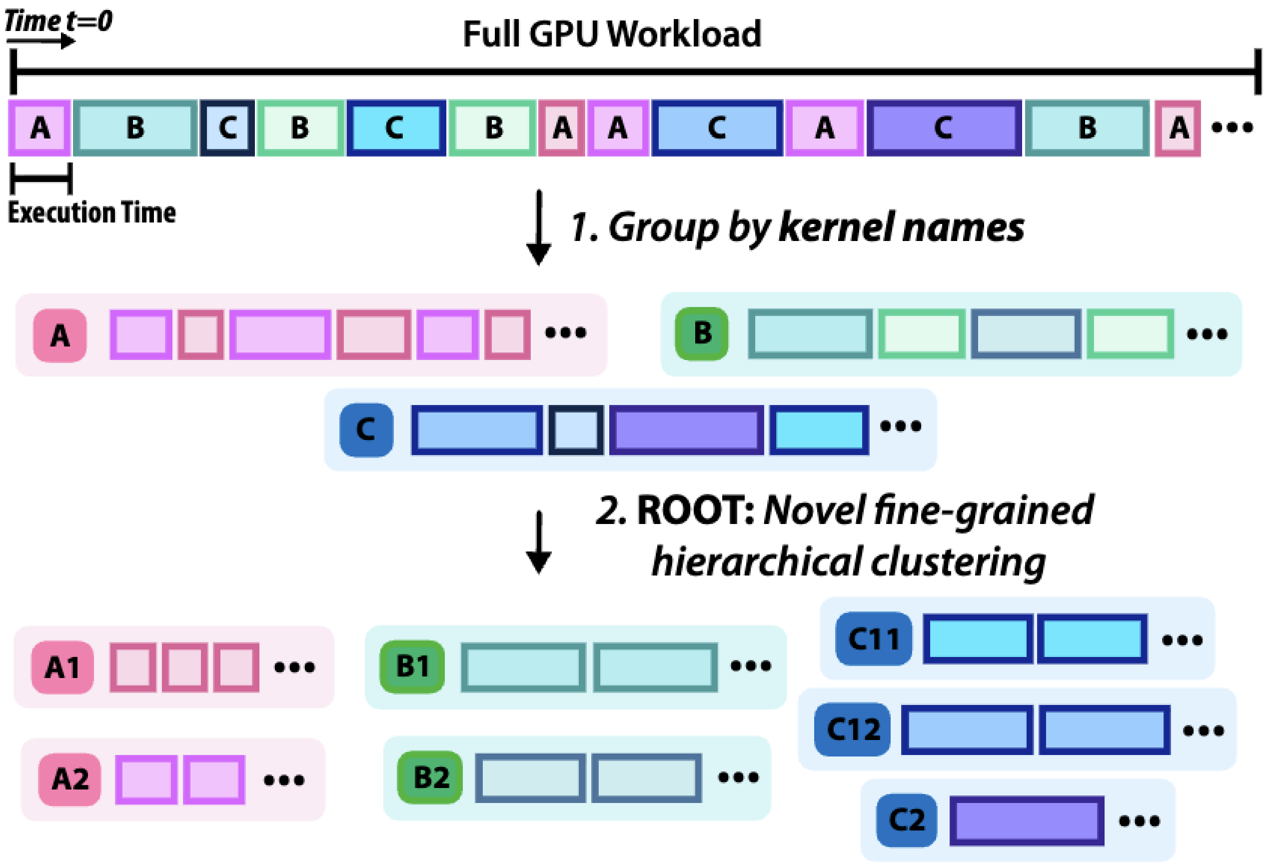


Summary of STEM + ROOT



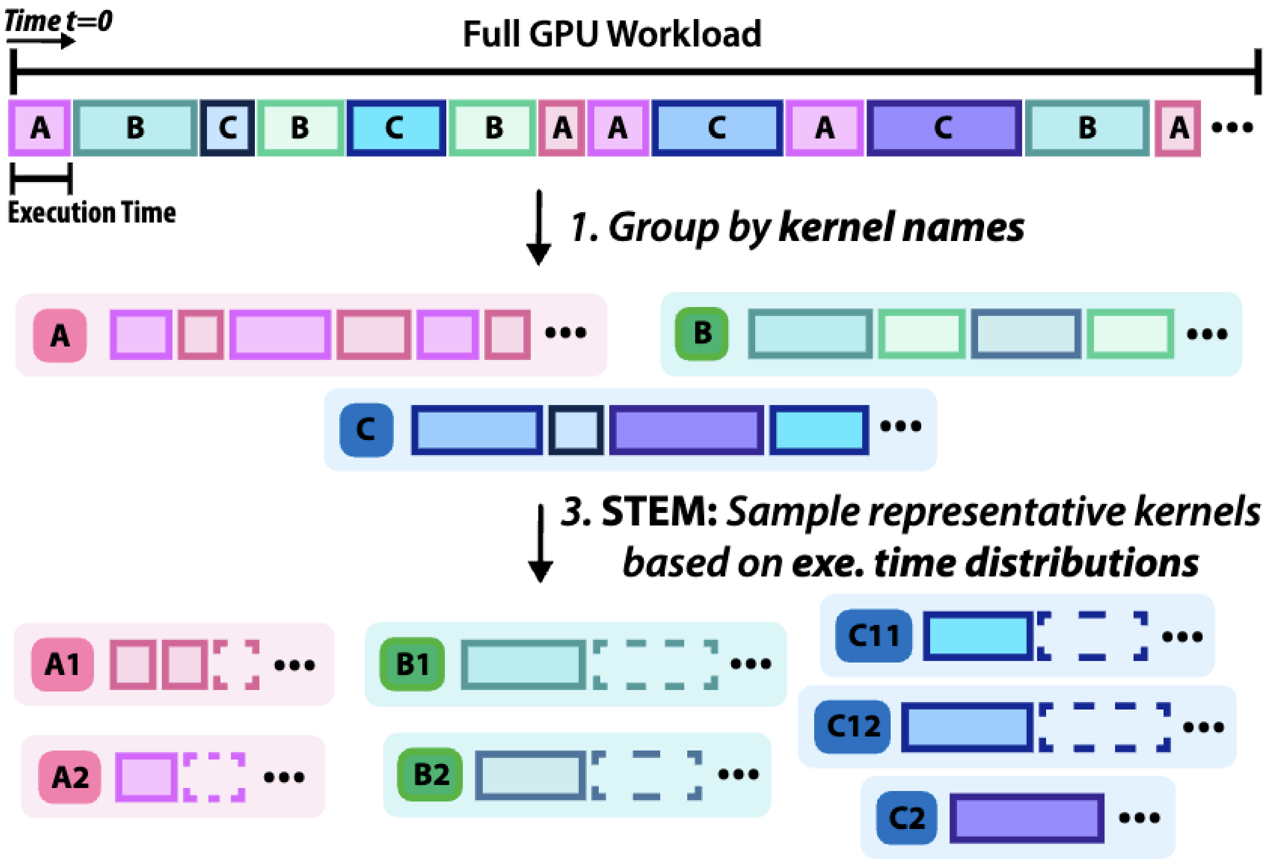
1. Group kernels by kernel names.

Summary of STEM + ROOT



2. ROOT **additionally separates** runtime-heterogenous kernels into different groups.

Summary of STEM + ROOT



3. STEM selects the **optimal sample size** of each group for the best speedup and accuracy.

Evaluation of STEM+ROOT

Evaluated GPU workloads:

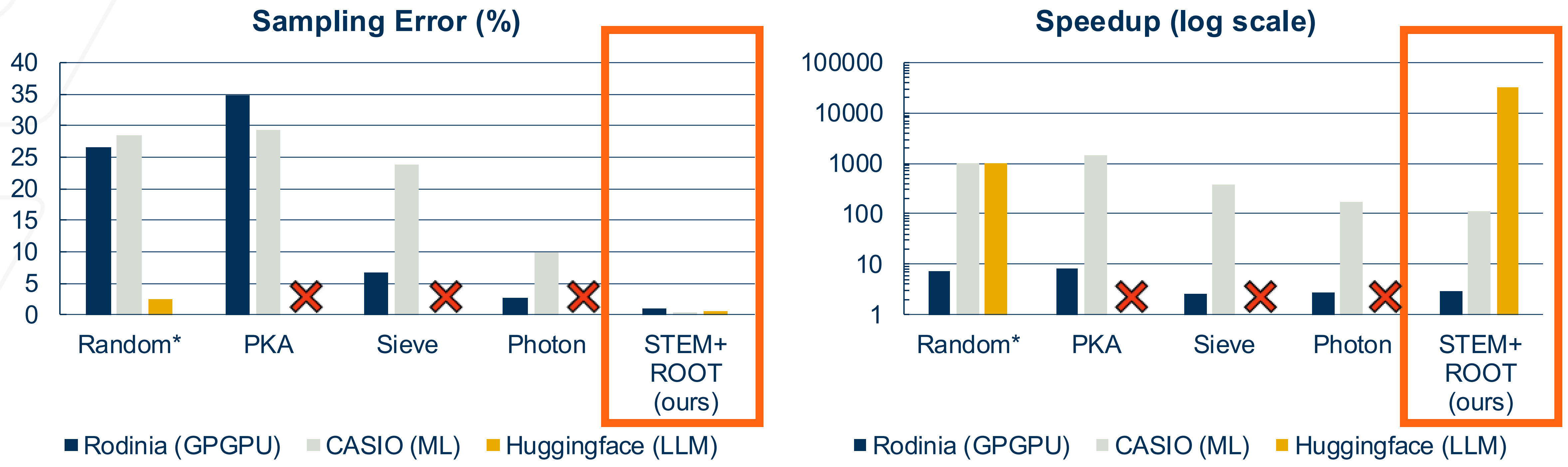
- Rodinia* (GPGPU workloads)
- Casio** (ML workloads)
- Huggingface (Large-scale LLM/ML workloads)

Baseline methods:

- Random sampling
- PKA [MICRO ‘20]
- Sieve [ISPASS ‘23]
- Photon [MICRO ‘23]

* S. Che et al., Rodinia: A benchmark suite for heterogeneous computing, 2009
** M. Davies et al., A Journey of a 1,000 Kernels Begins with a Single Step: A Retrospective of Deep Learning on GPUs, ASPLOS 2024

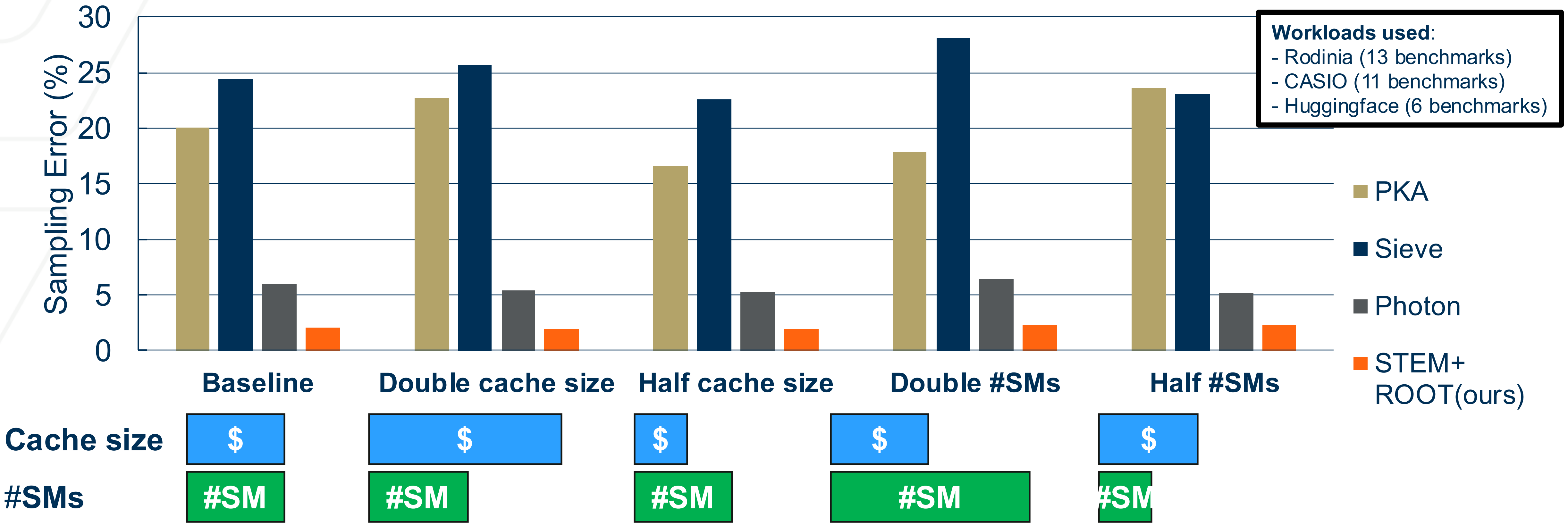
Speedup & Error validation on real HWs



❌: Infeasible due to significant profiling or sampling process overhead

- STEM+ROOT achieves **significantly lower sampling error with comparable speedup.**

Speedup & Error validation on cycle-level simulators



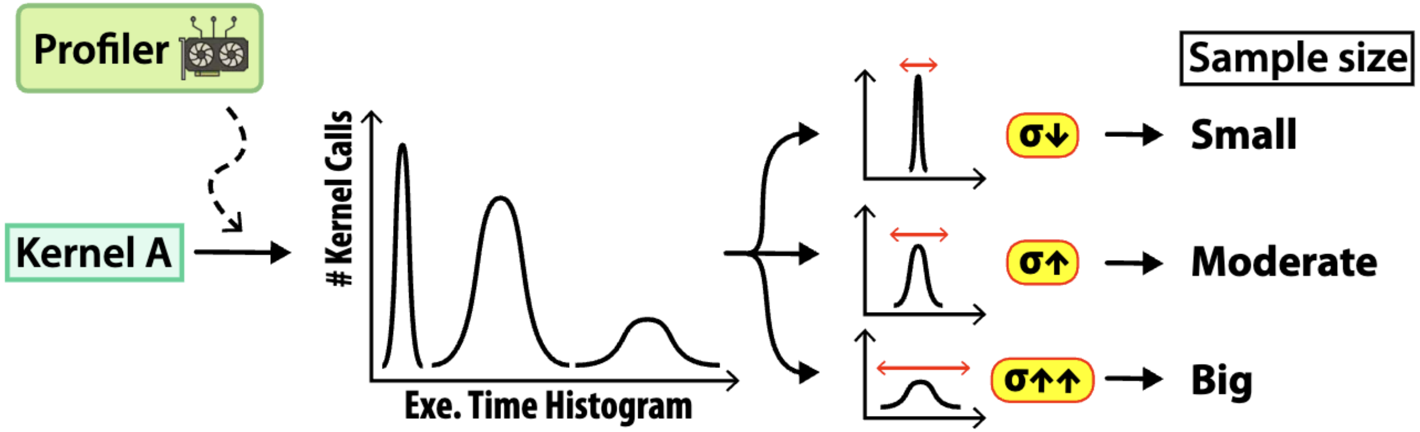
- Kernel's exe. time distribution reveals useful information about its characteristics.
- **Adaptive sample size** stays robust under HW (compute/memory) changes.

More details & evaluation results in our paper!

- Mathematical **modeling and proofs** on statistical sampling
- **Sensitivity analysis** on changing the error bound
- Evaluating STEM on a GPU with **kernel profiles from a different GPU**
- Evaluation on **microarchitecture metrics** (Cache hit rate, # instrs, etc.)
- Workload **profiling overhead** comparison for sampling
- and more.

Conclusion

- **Problem**: Tradeoff between speedup and accuracy in sampling for simulations.
- **Idea**: Leverage **kernels exe. time distribution** to select representative kernels.



- **STEM**: Optimal sample size selection with bounded sampling error.
- **ROOT**: Hierarchical clustering for distinguishing **runtime-heterogeneous kernels**.
- **Result**: **Fast**, **accurate** and **scalable** kernel sampling for large-scale GPU workloads
  - Evaluated 30 GPU benchmarks, STEM+ROOT achieves <1% error with high speedup.

Thank you!

Questions?

- Presenter: Euijun Chung (euijun@gatech.edu)

Link to the paper

Georgia Tech

Backup slides

Kernel-level sampling for GPU workloads

	50 ms	100 ms	48 ms	102 ms	53 ms	94 ms	51 ms	
Full workload	A	B	A	B	A	B	A	**Total runtime** = 498 ms

	50 ms	100 ms
Sampled workload	A	B

Sampled runtime = 150 ms

→ **Total runtime estimation** = 50ms * $\boxed{4}$ + 100ms * $\boxed{3}$ = 500 ms

Sampled 1 out of 4 A kernels

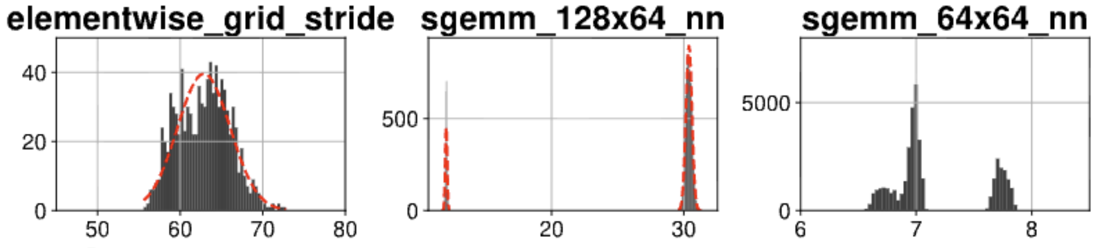
Sampled 1 out of 3 B kernels

- **Speedup** over full simulation $\approx \frac{498}{150} = 3.32$
- **Sampling error** $= \frac{|500-498|}{498} \times 100(\%) = 0.4\%$

Can we make the kernel sampling **fast** and **accurate** by leveraging the characteristics of **large-scale GPU workloads?**

STEM: Statistical Error Model for kernel sampling

Question: What if we are sampling kernels from multiple clusters at the same time?



Sample m1, m2, m3 kernels from each cluster

Optimization problem:

$$\underset{m_i}{\text{minimize}} \quad \tau = \sum_i m_i \mu_i$$

$$\text{subject to} \quad \sum_i N_i^2 \frac{\sigma_i^2}{m_i} \le \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i \mu_i \right)^2$$

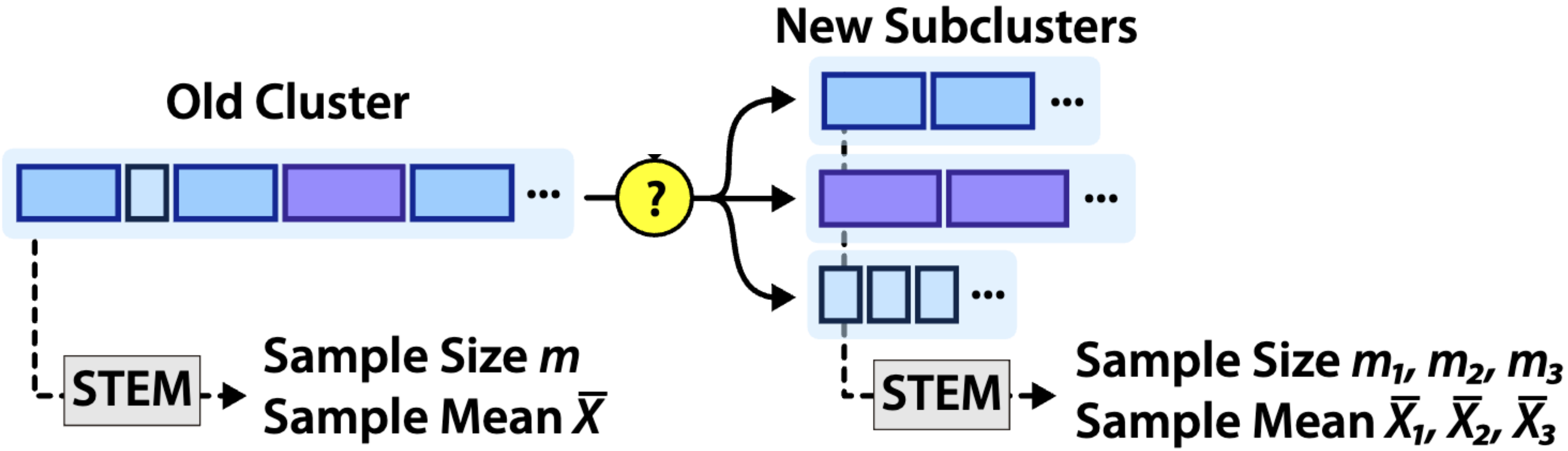
$$\text{and} \quad m_i > 0 \ \text{ for } \ \forall i \in \{0, \ldots, k-1\}.$$

Solution (Using KKT Conditions):

$$m_i = \left\lceil \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \right\rceil \text{ for } \forall i \in \{0, \ldots, k-1\}$$

$$a_i \equiv \mu_i,\ b_i \equiv N_i^2 \sigma_i^2, \text{ and } c \equiv \left(\epsilon \textstyle\sum_i N_i \mu_i / z_{1-\alpha/2}\right)^2$$

Deriving the ROOT

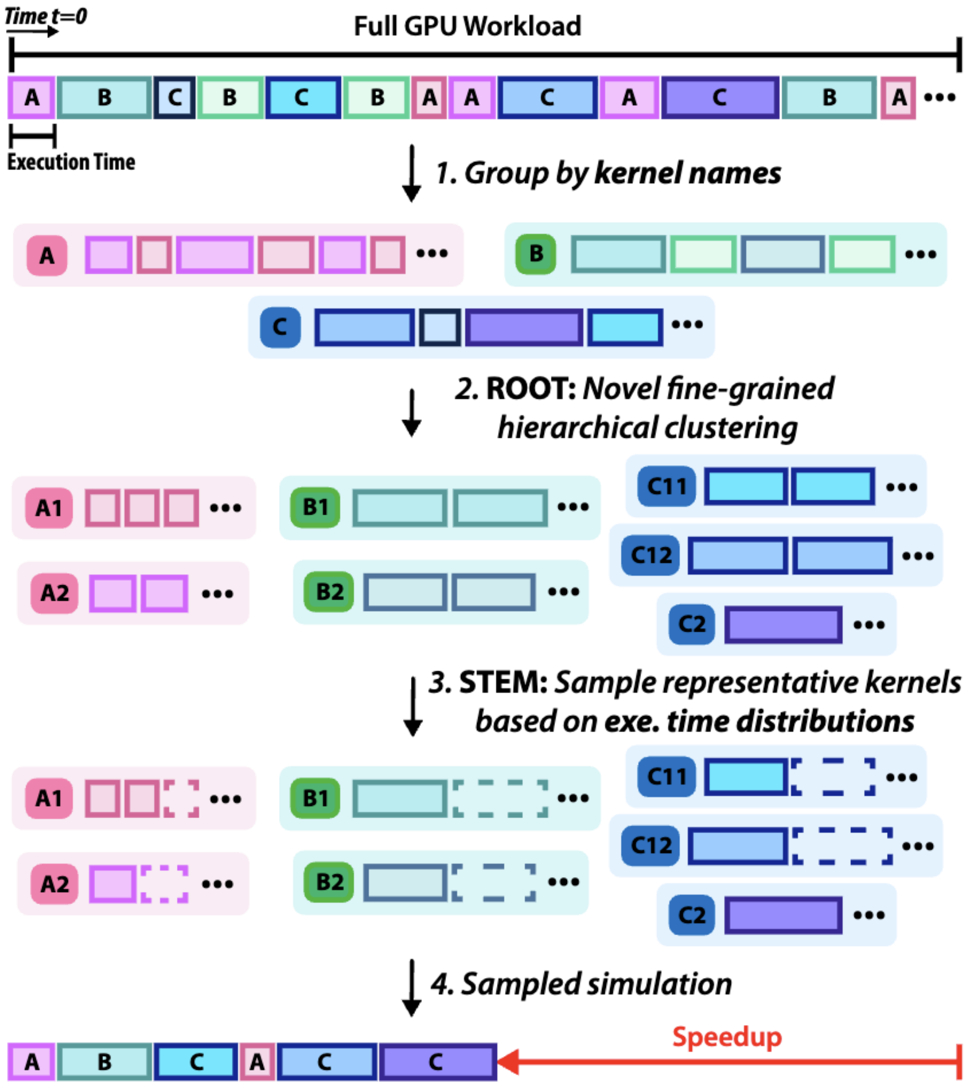


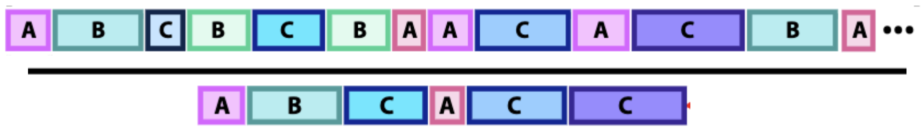
Compare the speedup:

$$\tau_{old} = m\bar{X} = \lceil (z_{1-\alpha/2}\sigma/\mu\epsilon)^2 \rceil \cdot \bar{X}$$

$$\tau_{new} = \sum_i m_i \bar{X}_i = \sum_i \left\lceil \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \right\rceil \cdot \bar{X}_i$$

Kernel-level sampling of GPU workloads



✓ Speedup = 

✓ Sampling error is minimal (bounded).

Georgia Tech

Baseline kernel sampling methods for GPU workloads

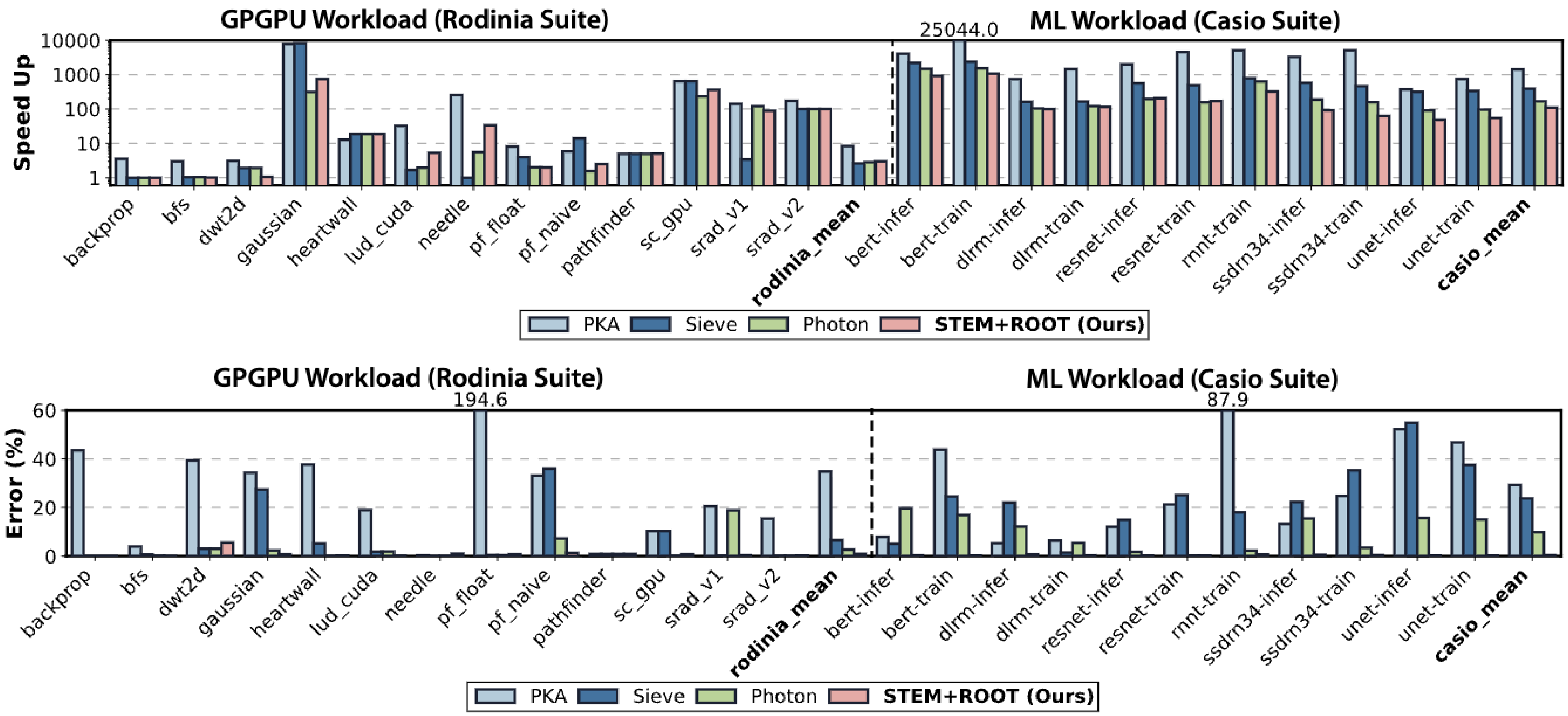
Sampling Methods	PKA [2]	Sieve [24]	Photon [21]	STEM+ROOT (ours)
Kernel signature	12 instr. level metrics	Kernel name & Num. of instrs	GPU Basic Block Vector (BBV)	Kernel name & Exe. time distribution
Clustering	k-means	Hand-tuned, based on CoV (σ/μ)	Find a kernel with similar BBV and #warps (95% threshold)	Fine-grained hierarchical (ROOT)
Kernel sample size	Single per cluster, first chronological	Single per cluster, first chronological		Adaptive sampling with statistically determined sample size (STEM)
Profiling granularity	Instr. count and statistics *per warp*	Instr. count *per warp*	Basic block count *per warp*	Execution time *per kernel*
Scalability for large-scale workloads	Very low	Low	Low	High

Limitations on previous works:

- PKA, Sieve, and Photon all rely on **static code-level analysis**, which fail to capture runtime heterogeneity of GPU kernels
- PKA and Sieve rely on **heavy profiling** of instr-level metrics
- Photon's BBV comparions between kernels involve $O(N^2d)$ **computations**.
  - N = Number of kernels, d = BBV dimension

Georgia Tech

Speedup & Error validation



Baseline methods: PKA [Micro '20], Sieve [ISPASS '23], Photon [MICRO '23]

Evaluations on Microarchitectural metrics

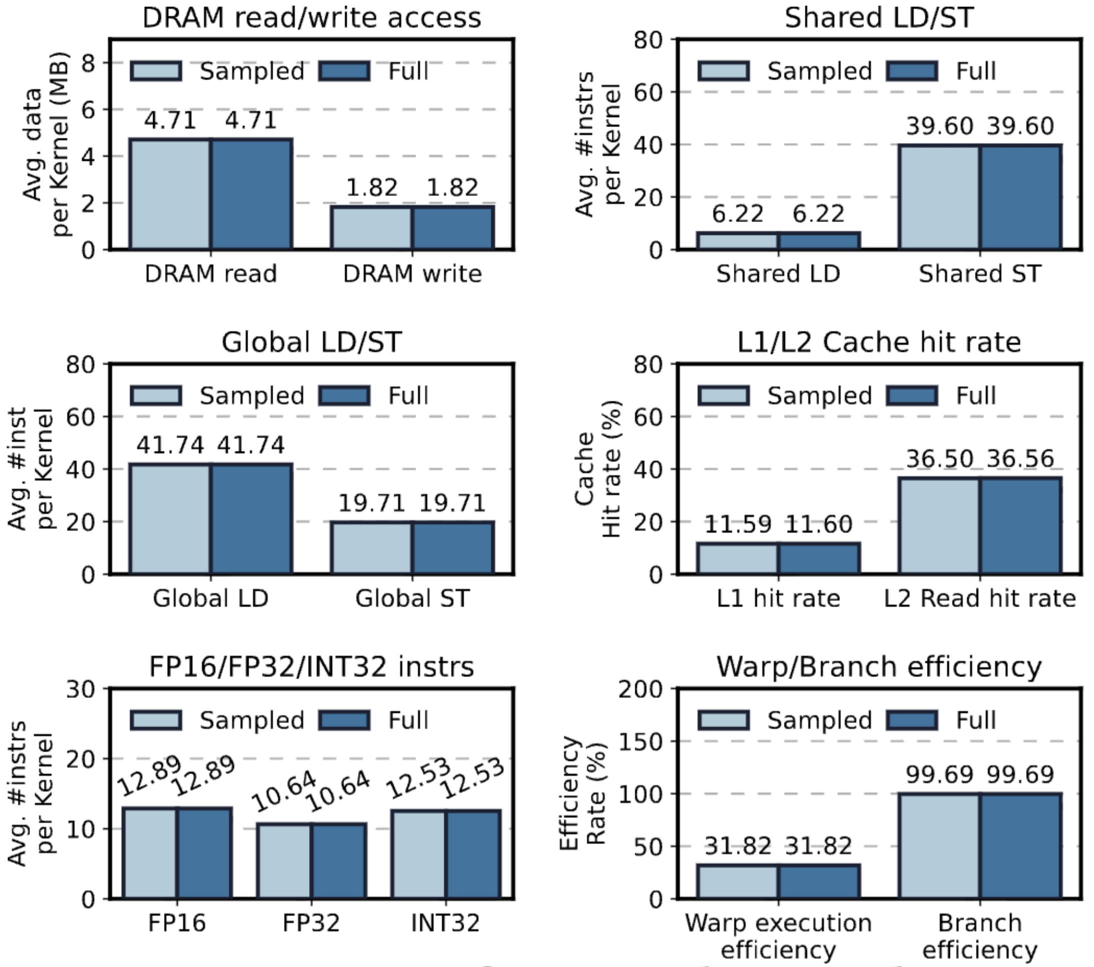


Figure 14: Comparison of microarchitectural metrics between the full workload and the sampled workload. We used the `bert_infer` workload of the CASIO benchmark suite.

Profiling overhead

Sampling methods	Profiler used, metrics collected	Rodinia (GPGPU)	CASIO (ML)	Huggingface (LLM & ML)
PKA [2]	NCU, collecting 12 metrics	35.57×	3704.23×	N/A
Sieve [24]	NVBit, collecting num. of instrs	94.14×	293.58×	N/A
Photon [21]	NVBit, collecting & processing BBVs	12.81×	38.58×	N/A
STEM (ours)	NSYS, collecting kernel exe. time	**1.54×**	**5.53×**	**1.33×**

Using **execution time** as a key parameter gives a huge improvement in **scalability**