Transparent spatial sharing of multiple and heterogeneous accelerators

Manos Pavlidakis
Computer Science Department, University of Crete, Greece

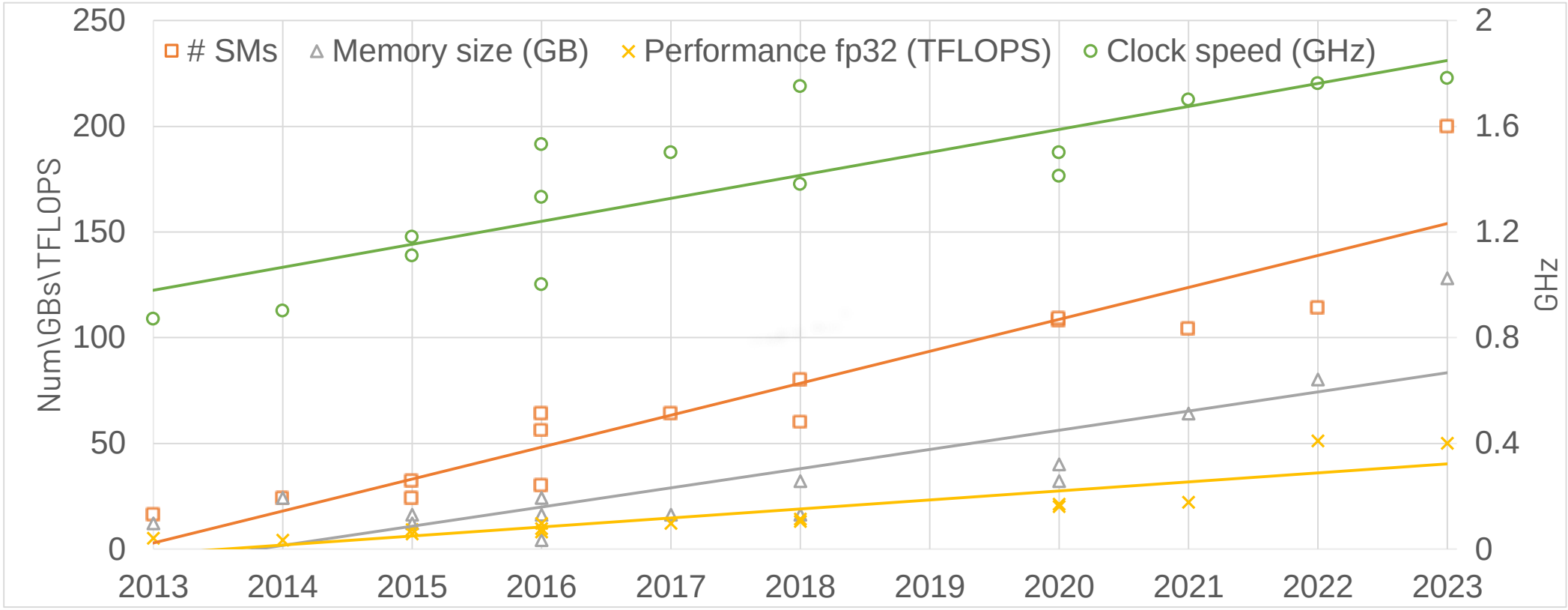
Advisor: Professor Angelos Bilas

PhD defense

* UC Berkeley RISElab, AI and Memory Wall

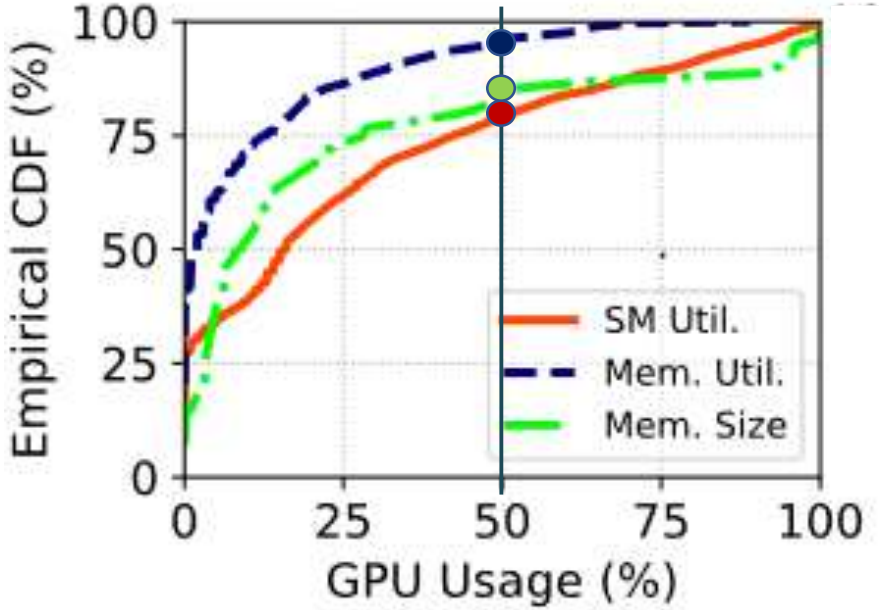
Single accelerator resources increase

- Every 2 years:
  - SMs: 1.9x, Memory capacity: 1.7x
  - Performance fp32: 1.6x, Clock speed: 1.2x, and Performance int8: 3x *



Today applications fail to utilize a single large accelerator [1,2,3]

- **Only 20%** of jobs use > 50% **SMs** of a single GPU
- **Only 4%** of jobs use > 50% **memory bw utilization**
- **Only 15%** of jobs use > 50% of the available **memory size**



*Average SM and Memory utilization of various jobs using a **single** GPU [2]*

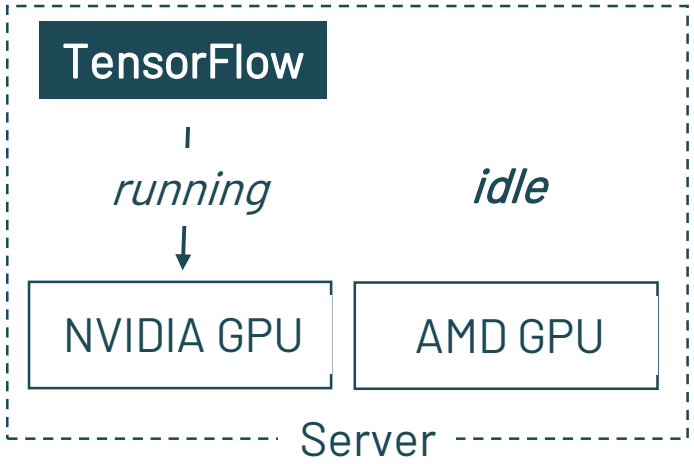
[1] NSDI'22, MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters, **Alibaba** Production **Cluster**
[2] HPCA' 22, AI-Enabling Workloads on Large-Scale GPU-Accelerated System: Characterization, Opportunities, and Implications, **MIT Supercloud**
[3] Arxiv'17, Workload Analysis of BLUE WATERS, NCSA **Petascale-level supercomputer**

Sources of accelerator under-utilization

1. **Lack** of resource **adaptation** to **dynamic** application **load**
   - Elastic sharing: one app uses a varying number of accelerators at runtime

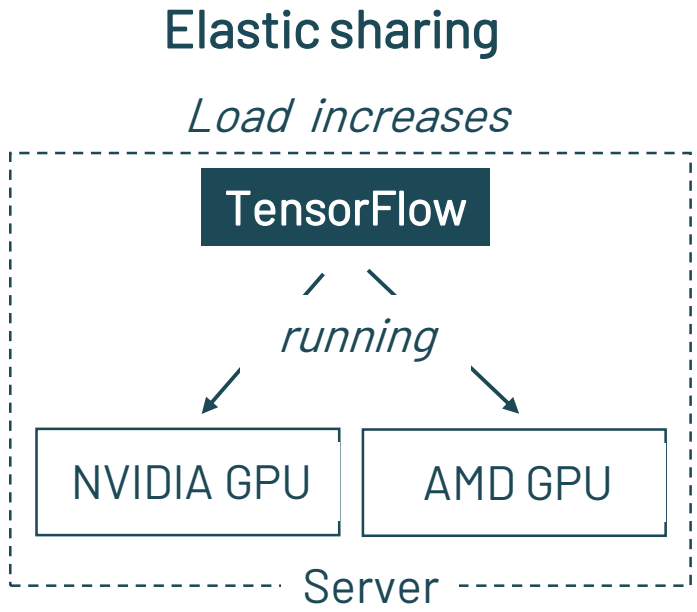
2. **Lack** of efficient and safe accelerator **multi-tenancy**
   - Spatial sharing: Multiple apps/tenants run on one accelerator in parallel

Elastic sharing



Sources of accelerator under-utilization

1. **Lack** of resource **adaptation** to **dynamic** application **load**
   - Elastic sharing: one app uses a varying number of accelerators at runtime

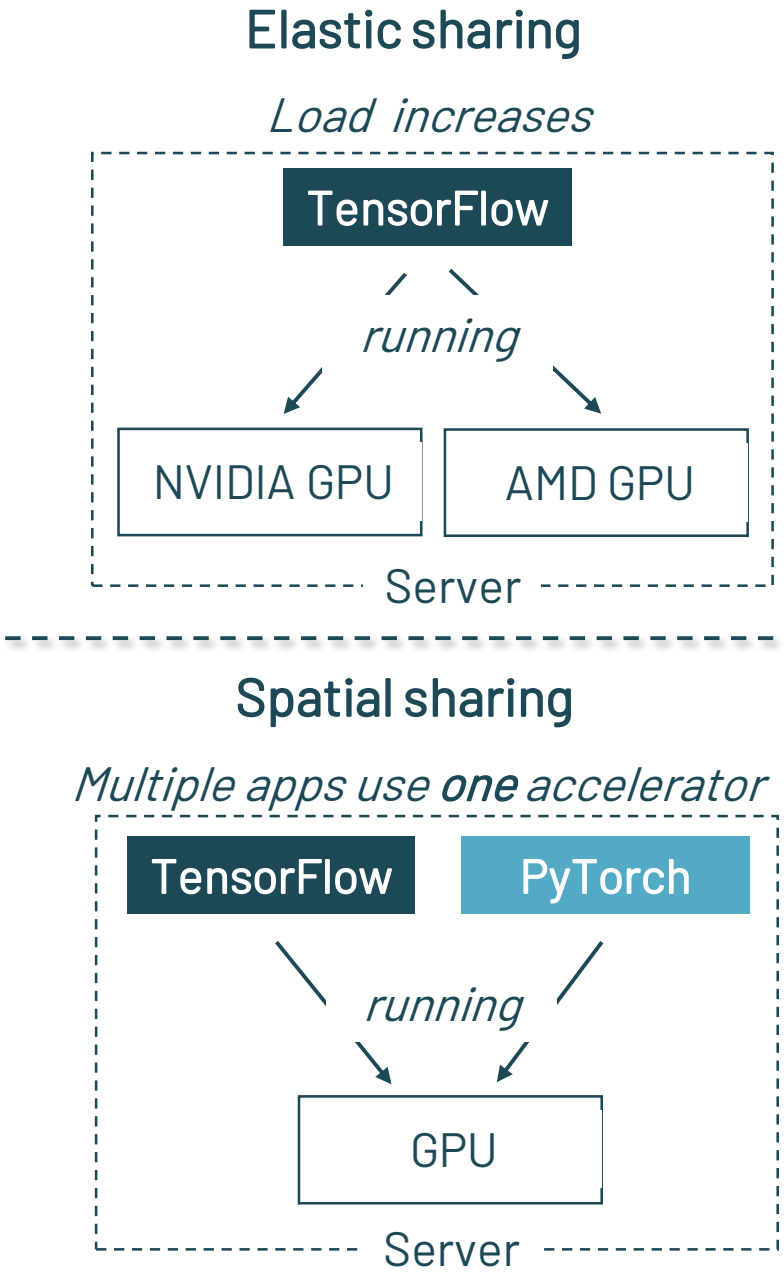


2. **Lack** of efficient and safe accelerator **multi-tenancy**
   - Spatial sharing: Multiple apps/tenants run on one accelerator in parallel

Sources of accelerator under-utilization

1. **Lack** of resource **adaptation** to **dynamic** application **load**
   - Elastic sharing: one app uses a varying number of accelerators at runtime

2. **Lack** of efficient and safe accelerator **multi-tenancy**
   - Spatial sharing: Multiple apps/tenants run on one accelerator in parallel



Lack of adaptation to dynamic application load

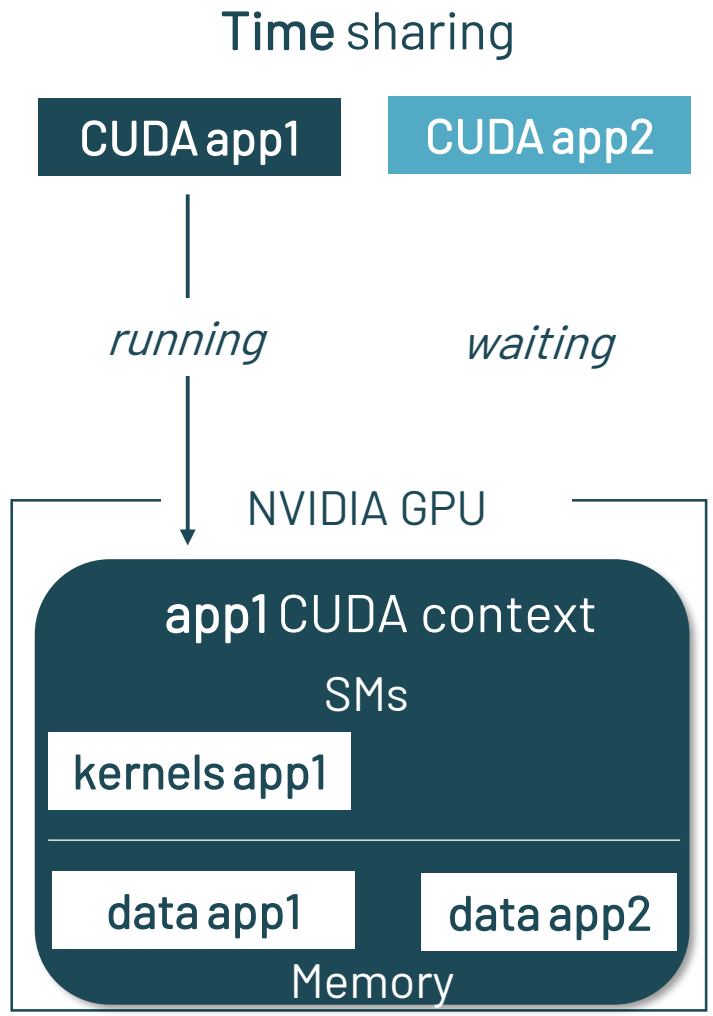
- Apps **once** at their **launch** time **select statically** accelerator **type**, **number**, and **set**
  - Existing **programming models** (PMs) can **not** perform **dynamic** decisions
  - **Accelerator-specific** PMs: A **CUDA** app can choose **NVIDIA** GPU**1** and GPU**3**
  - **Unified** PMs: A **SYCL** app can choose NVIDIA **GPU1** and Intel **FPGA2**
  - **Unified** PMs only **hide** the accelerator **type** different from accelerator-specific
- **Static decisions** lead to **accelerator under-utilization**
  - Apps have **variable resource demands** during execution [4,5] → e.g. Num. of accelerators
  - Existing solution: **Over-provisioning** to avoid performance degradation **but** leads to **idleness**

✓ **Elastic sharing** using a **common runtime process** between apps and accelerators

[4] SoCC'19, DCUDA: Dynamic GPU Scheduling with Live Migration Support
[5] SoCC'22, MISO: exploiting multi-instance GPU capability on multi-tenant GPU clusters

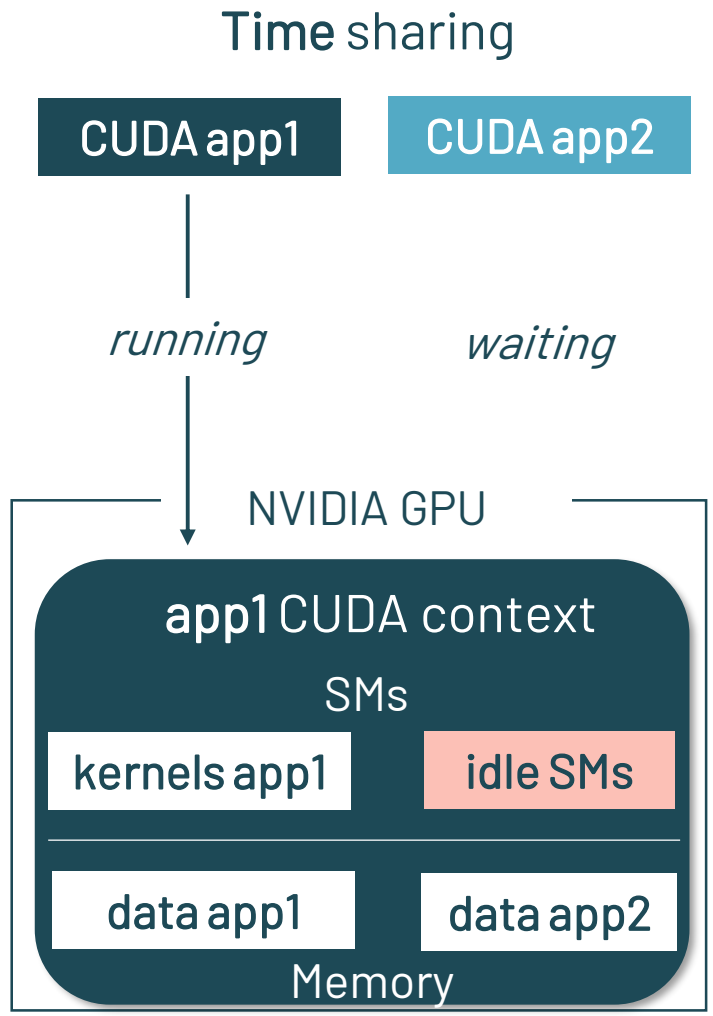
Lack of accelerator sharing

1. **NVIDIA GPUs** support by default **time-sharing**
   - Only **one app** uses the GPU at **any given time**



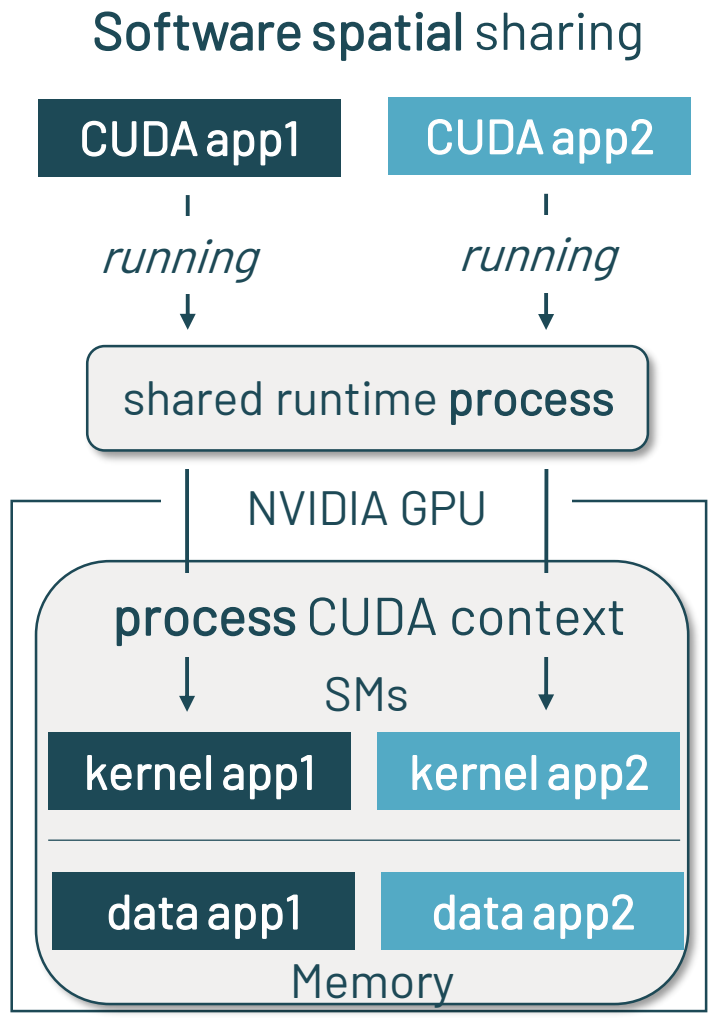
Lack of accelerator sharing

1. **NVIDIA GPUs** support by default **time-sharing**
   - Only **one app** uses the GPU at **any given time → idleness**



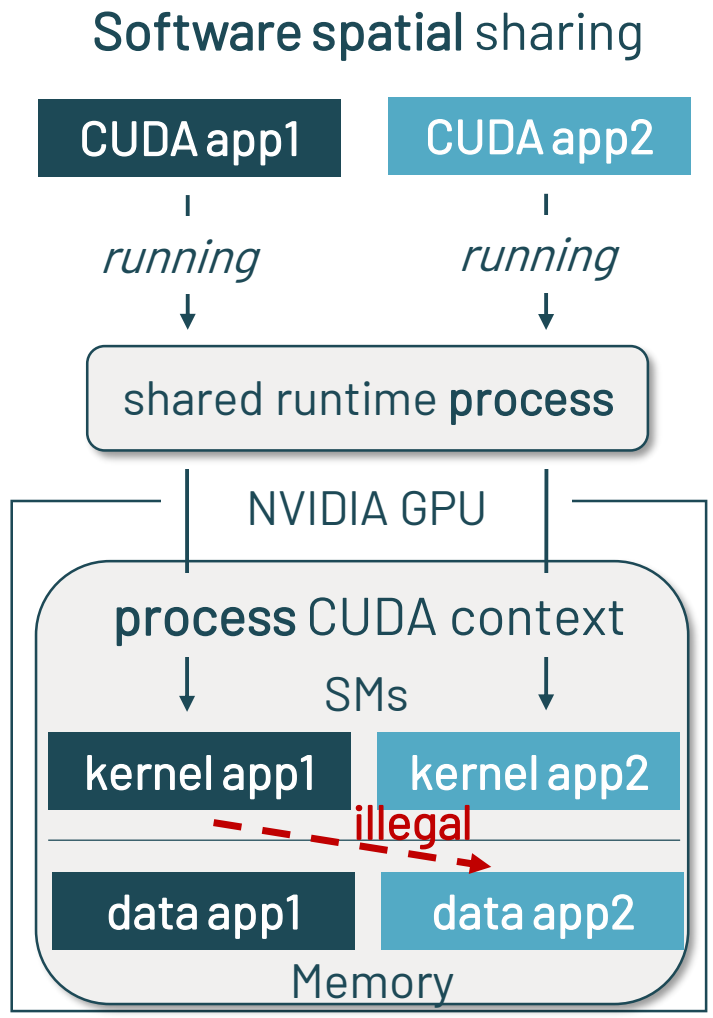
Lack of accelerator sharing

1. **NVIDIA GPUs** support by default **time-sharing**
   - Only **one app** uses the GPU at **any given time → idleness**
2. **Software spatial** sharing such as NVIDIA **MPS**
   - Applications run **concurrently** in a GPU
   - Requires a single GPU context



Lack of accelerator sharing

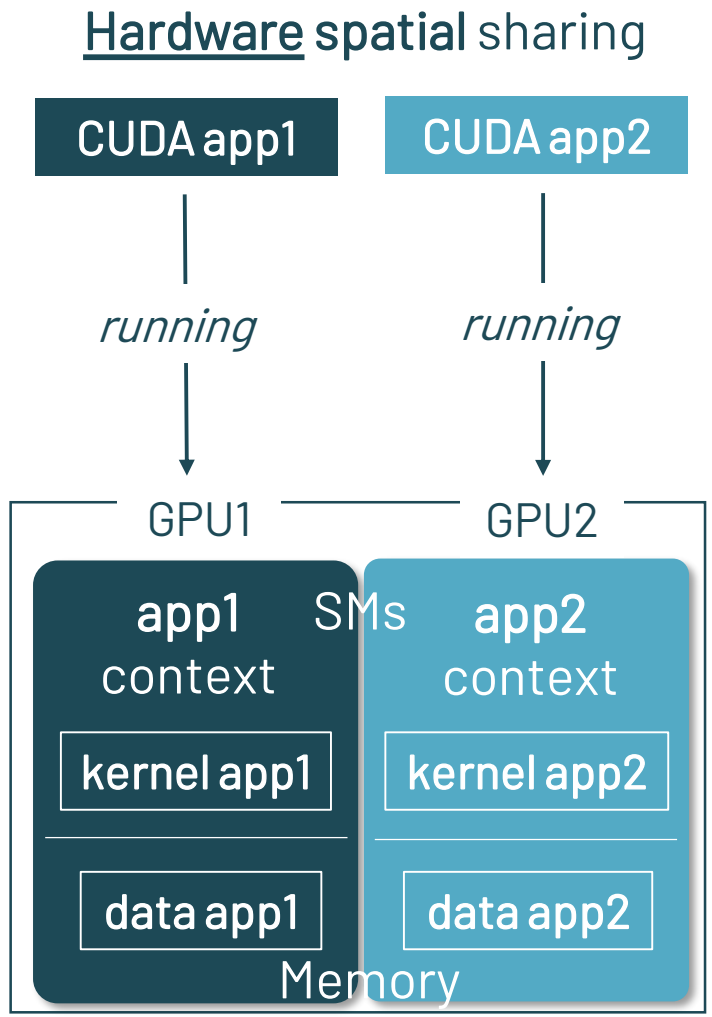
1. **NVIDIA GPUs** support by default **time-sharing**
   - Only **one app** uses the GPU at **any given time → idleness**
2. **Software spatial** sharing such as NVIDIA **MPS**
   - Applications run **concurrently** in a GPU
   - Requires a single GPU context → **No protection**



Lack of accelerator sharing

1. **NVIDIA GPUs** support by default **time-sharing**
   - Only **one app** uses the GPU at **any given time → idleness**
2. **Software spatial** sharing such as NVIDIA **MPS**
   - Applications run **concurrently** in a GPU
   - Requires a single GPU context → No protection
3. **Hardware** spatial sharing such as NVIDIA **MIG**
   - **Partitions** the GPU **statically** to independent partitions (GPUs)
   - **Changing** the partition **scheme** requires GPU **reset**

✓ **Safe software spatial sharing** using kernel binary **code instrumentation**



Thesis statement

Provide transparent and efficient sharing of heterogeneous accelerators for real-world applications in a server

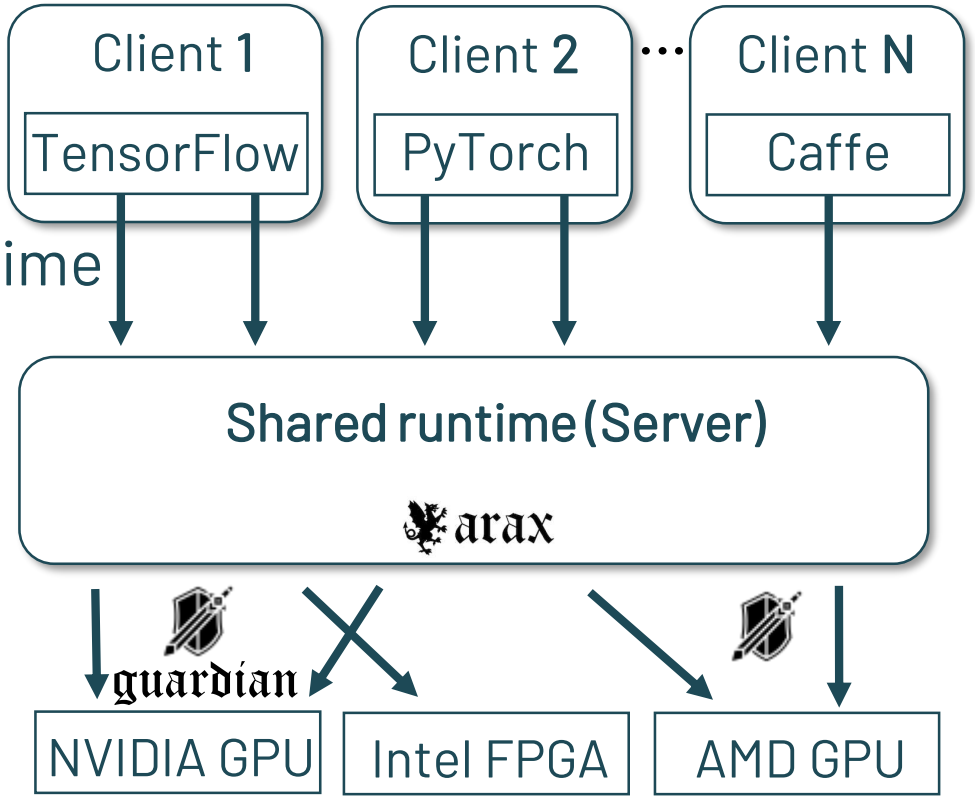
Multiple accelerators → Elastic sharing

Single accelerator → Spatial sharing

Thesis contributions

A runtime for **transparent**, **elastic,** and **spatial** sharing of **multiple accelerators**

- Specific contributions
  - ✓ Per **task dynamic** accelerator **assignment** at runtime
    - → elastic and spatial sharing
      - A shared process managing apps and accelerators
  - ✓ **Protect memory** and **control flow** instructions
    - → protected spatial sharing
      - Code instrumentation at the GPU kernel binary code



Outline

- Introduction
- Thesis statement and contributions
- ➢ **Elastic application to accelerator assignment (Arax)**
- Protected accelerator spatial sharing (Guardian)
- Conclusions

Arax: A Runtime Framework for Decoupling Applications from Heterogeneous Accelerators, SoCC'22
Guardian: Data Isolation for Multi-Tenant GPU Sharing, Under submission

Abstract accelerator(s)

✓ Goal: **Abstracting** accelerator **type**, **number, and set** from apps

Arax Application

- Arax uses three main primitives

Abstract accelerator(s)

✓ Goal: **Abstracting** accelerator **type**, **number, and set** from apps

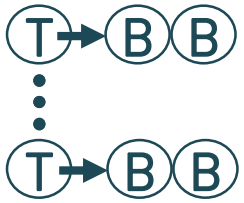
- Arax uses three main primitives

1. **Tasks** (Ⓣ): hide **accelerator-specific** information
   - Represent individual kernels and data transfers

Abstract accelerator(s)

✓ Goal: **Abstracting** accelerator **type**, **number, and set** from apps

Arax Application

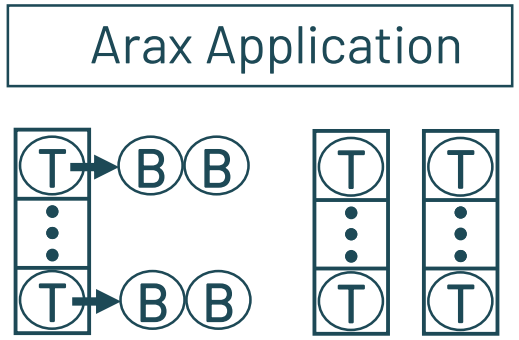


- Arax uses three main primitives

1. Tasks (Ⓣ): hide accelerator-specific information
   - Represent individual kernels and data transfers
2. **Buffers** (Ⓑ): hide **accelerator memory**
   - Opaque identifiers that represent task input/output data
   - Used to keep track of data dependencies in Arax

Abstract accelerator(s)

✓ Goal: **Abstracting** accelerator **type**, **number, and set** from apps



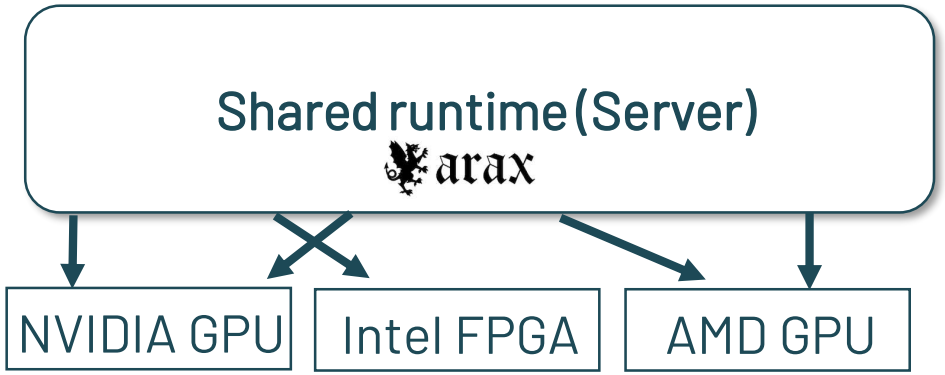
- Arax uses three main primitives

1. Tasks (Ⓣ): hide accelerator-specific information
   - Represent individual kernels and data transfers
2. Buffers (Ⓑ): hide accelerator memory
   - Opaque identifiers that represent task input/output data
   - Used to keep track of data dependencies in Arax
3. **Task Queues** (▭): express **task order**
   - Arax ensures in-order execution in each queue
   - Applications can allocate several queues for concurrency

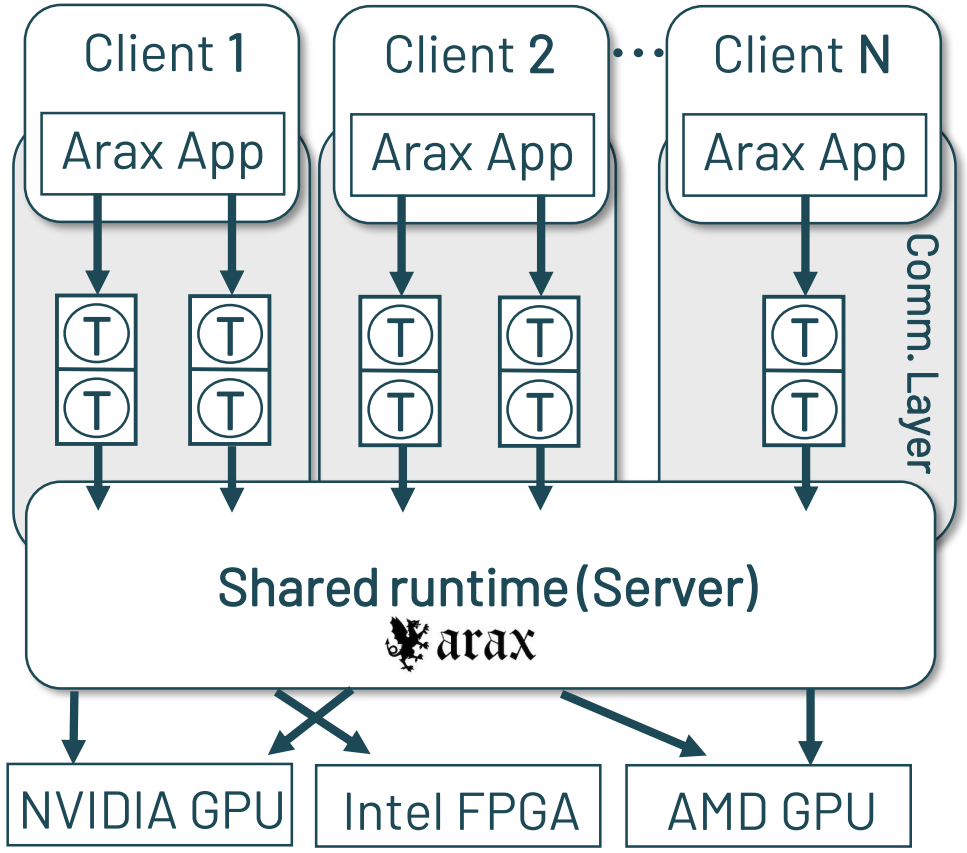
Global resource management across applications

✓ Goal: **Optimize** accelerator **use** across applications

- Arax uses a **shared runtime** process for **all apps**
  - Each **application** runs in a **separate address space**
  - The **runtime** (server) has a **global view** of apps & accelerators



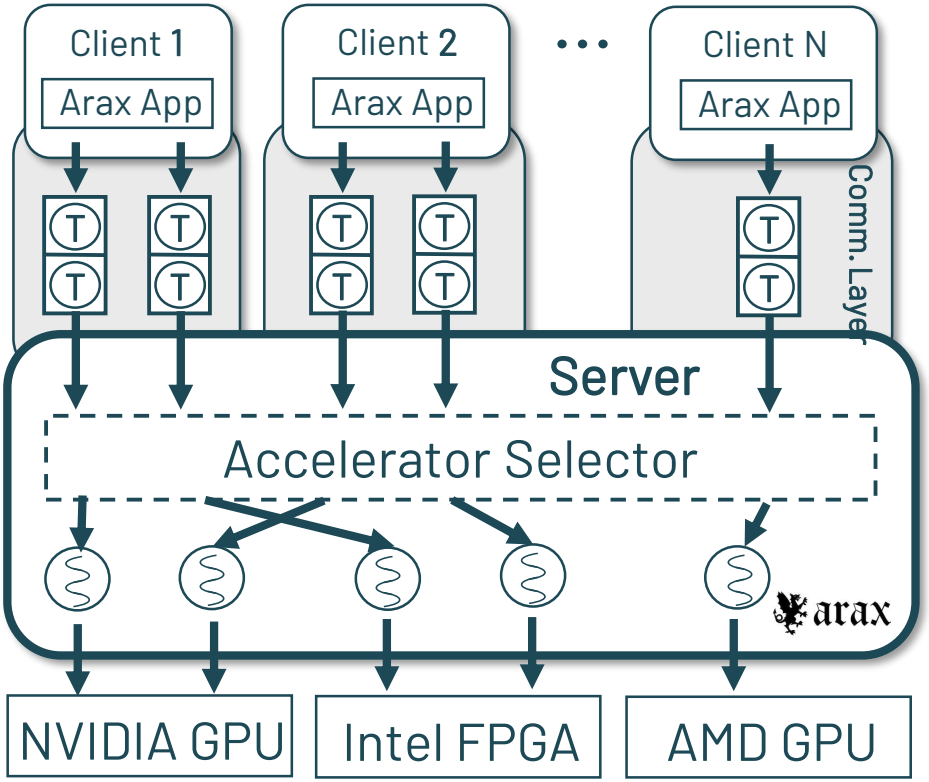
Global resource management across applications



✓ Goal: **Optimize** accelerator **use** across applications

- Arax uses a **shared runtime** process for **all apps**
  - Each **application** runs in a **separate address space**
  - The **runtime** (server) has a **global view** of apps & accelerators
- Arax uses **shared memory** for **communication**
  - Task and buffer synchronization
  - Allocation of in-transit buffers
  - Tracking of data location

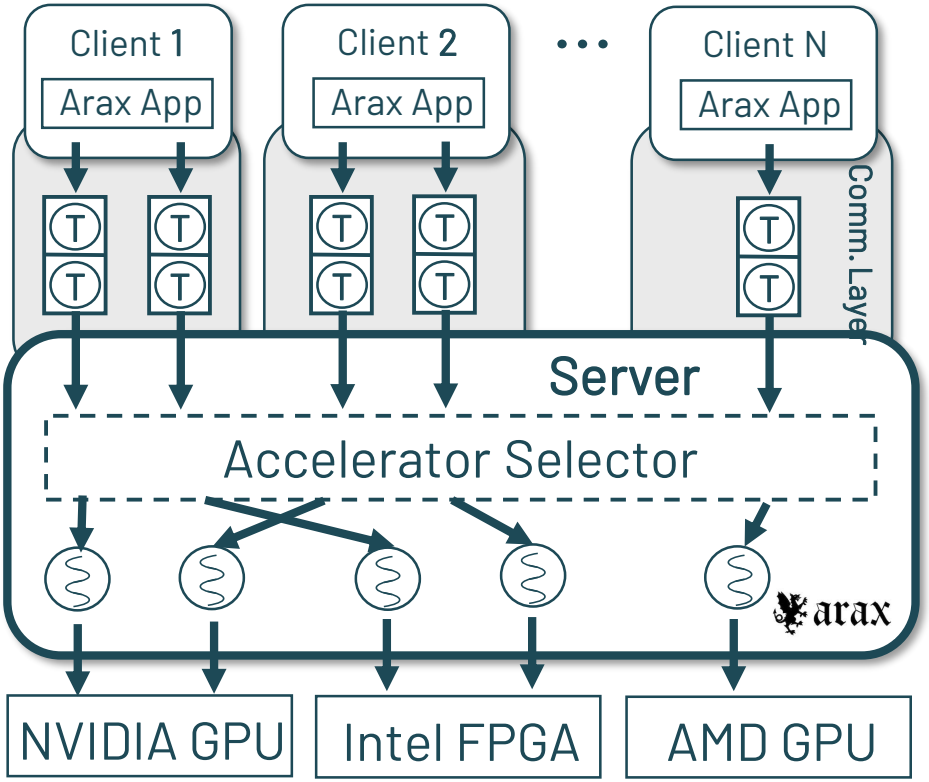
Dynamic task assignment at runtime



✓ Goal: **Adaptation** to application **load change**

- Arax has a **completely different** execution model
  - Existing runtimes: **Assignment** → Issue
  - Arax runtime: Issue → **Assignment**

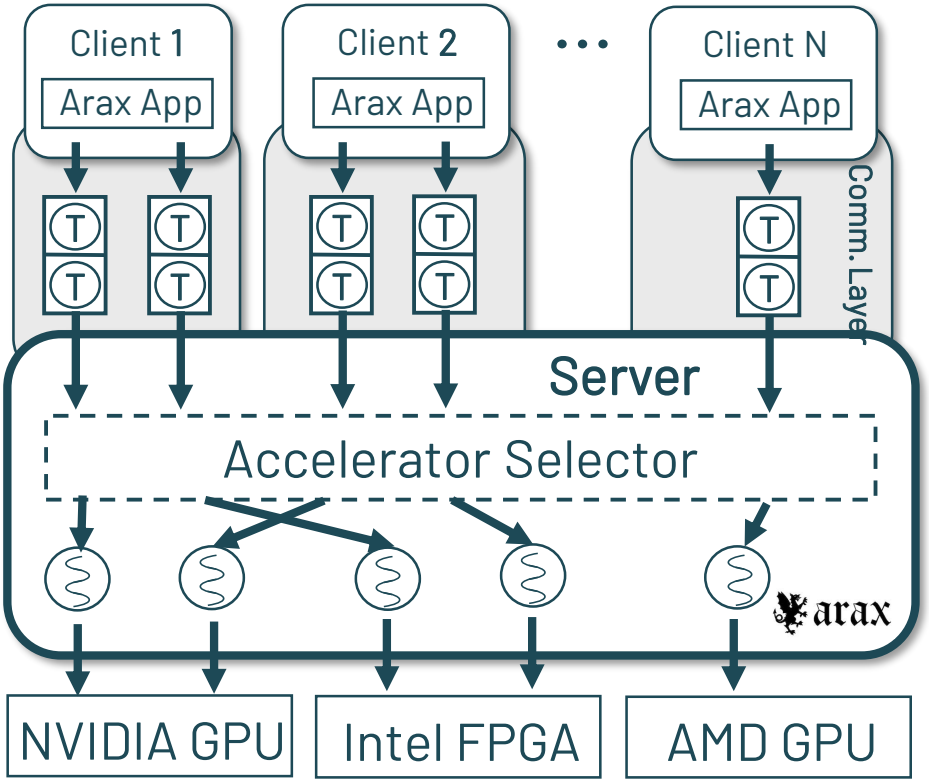
Dynamic task assignment at runtime



✓ Goal: **Adaptation** to application **load change**

- Arax has a **completely different** execution model
  - Existing runtimes: **Assignment** → Issue
  - Arax runtime: Issue → **Assignment**
- Arax moves **all** task management to the **server**
  - Select accelerator, transfer data, issue kernel, manage memory
  - Applications only issue tasks

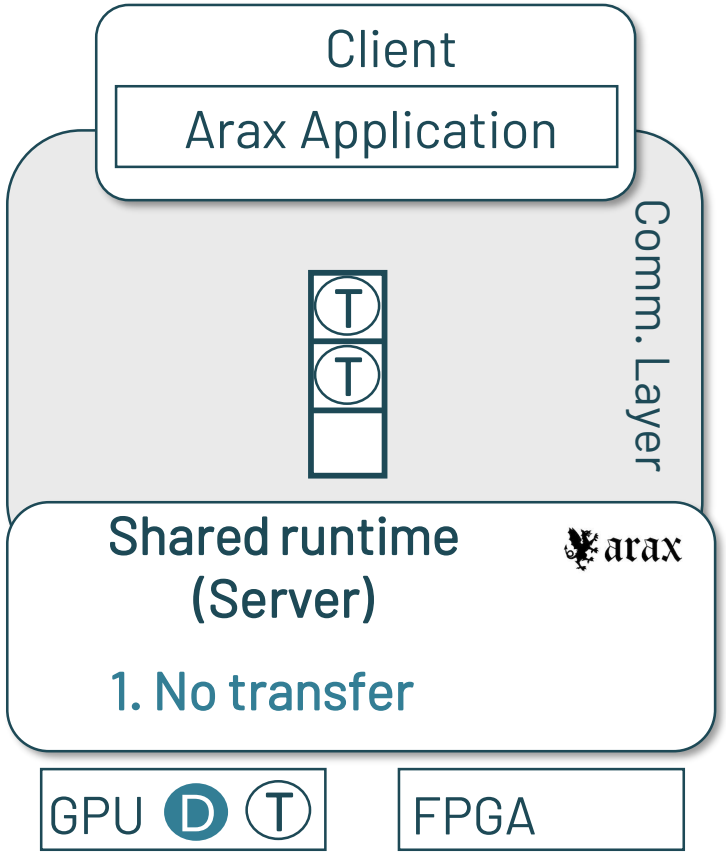
Dynamic task assignment at runtime



✓ Goal: **Adaptation** to application **load change**

- Arax has a **completely different** execution model
  - Existing runtimes: **Assignment** → Issue
  - Arax runtime: Issue → **Assignment**
- Arax moves **all** task management to the **server**
  - Select accelerator, transfer data, issue kernel, manage memory
  - Applications only issue tasks
- To perform the task management, the **Arax server:**
  - Holds all the kernels supported per accelerator → registry
  - Identifies the appropriate accelerator → accelerator selector
  - Handles thousands of tasks and queues → multi-threaded
  - Maintains task order → accelerator streams

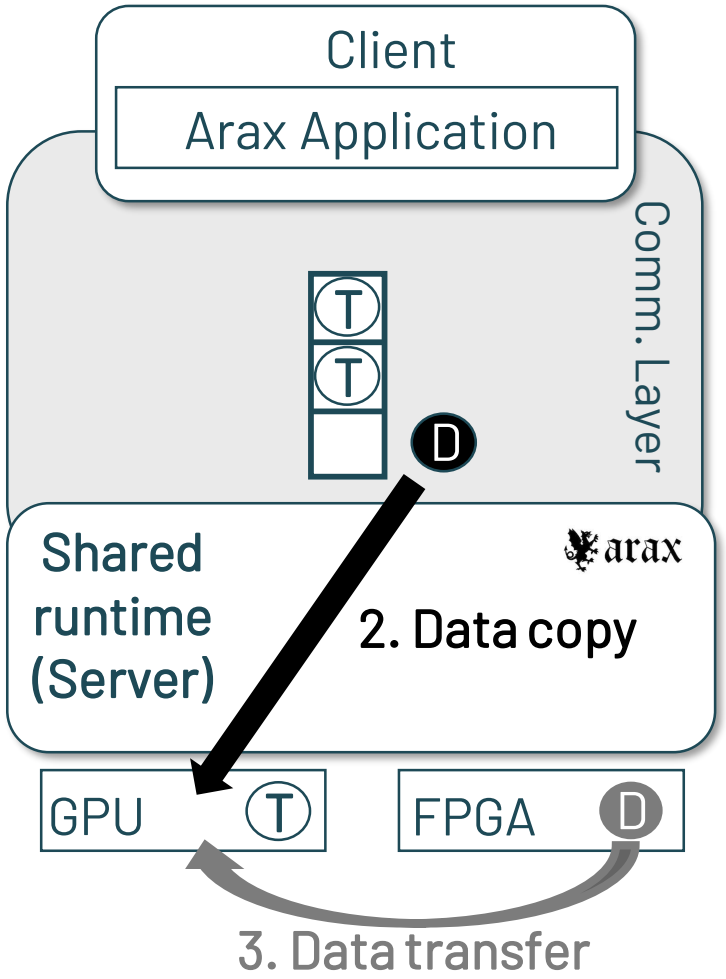
Keep track of task data



✓ Goal: **Flexibility** in task placement

- Keep **track** of task data
- Prepare data for task execution **lazily**
  1. Same accelerator → No transfer

Keep metadata per task



✓ Goal: **Flexibility** in task placement

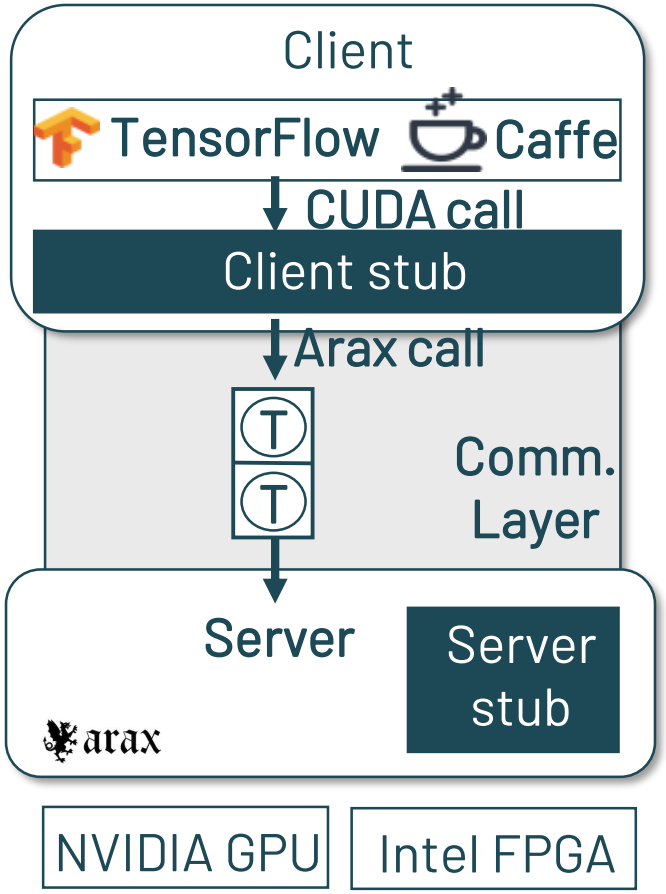
- Keep **track** of task data
- Prepare data for task execution **lazily**
  1. Same accelerator → No transfer
  2. Staging area → Data copy (HostToDevice)
  3. Other accelerator → Data transfer (DeviceToDevice)

Transparent spatial sharing

✓ Goal: **Collocate** tasks from **different apps** on the same accelerator

- Each **accelerator** has a **mechanism** for **spatial sharing**
  - GPUs → streams
  - FPGAs → multi-kernel bitstreams and command queues
- Arax **unifies** and **hides** these mechanisms
  - Reconfigures FPGAs depending on concurrently executing kernels
  - Uses a single CUDA context for all streams in each NVIDIA GPU
- Arax apps share **transparently different accelerator** types
  - Without knowing that they share an accelerator with other apps

Support real-world applications



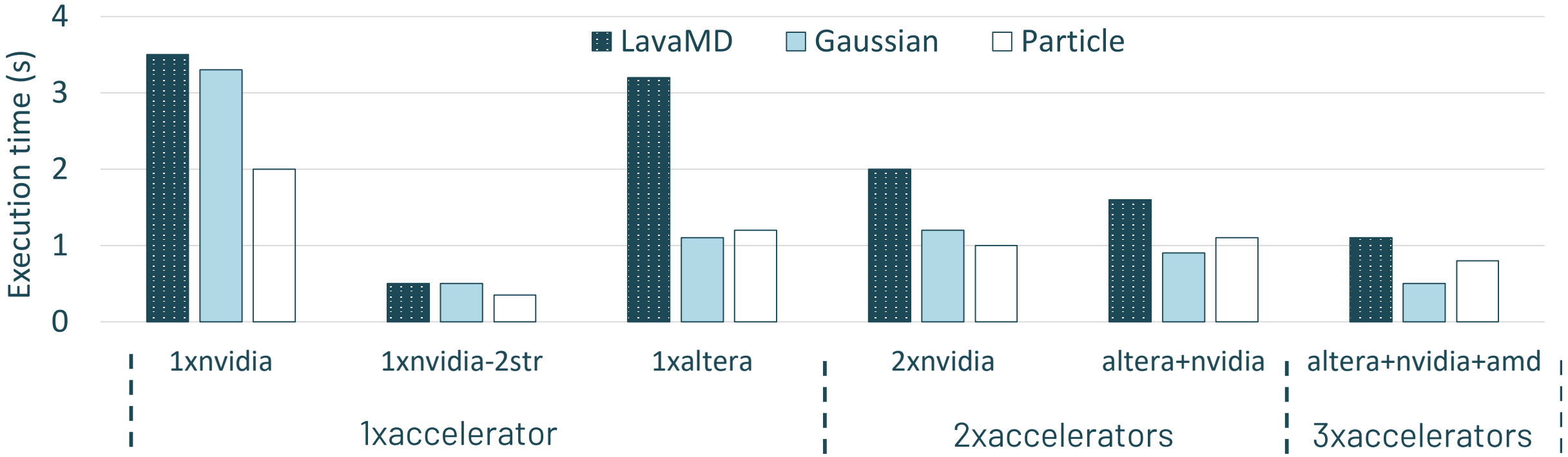
- ✓ Goal: **Minimize** the **porting effort**
- Arax provides tools to
  - Adjust **CUDA apps to Arax API → client stub**
  - Add a **new accelerator** and its kernels under **Arax → server stub**
- For the **client** stub
  - We **intercept** more than **2000 distinct CUDA calls**
  - 183 runtime + 249 driver + 1600 high-level lib calls (e.g., cuBLAS, cuDNN)
- For the **server** stub
  - We **extract** and load app **kernels** to the Arax server
- We perform this process **once** for CUDA 10.2
  - We can run Rodinia, Caffe, and TensorFlow with **zero effort**

Testbed

- Two server configurations with different accelerator types
  1. NVIDIA GPU, AMD GPU, and Intel FPGA
  2. Two RTX 2080 NVIDIA GPUs
- Microbenchmarks and real-world applications
  - Rodinia heterogenous benchmarks suite
  - Caffe deep learning framework
  - TensorFlow+Keras machine learning framework
- **We port applications to Arax once**
  - **Arax transparently manages accelerators in each configuration**
  - **Applications execute unmodified with different resources**

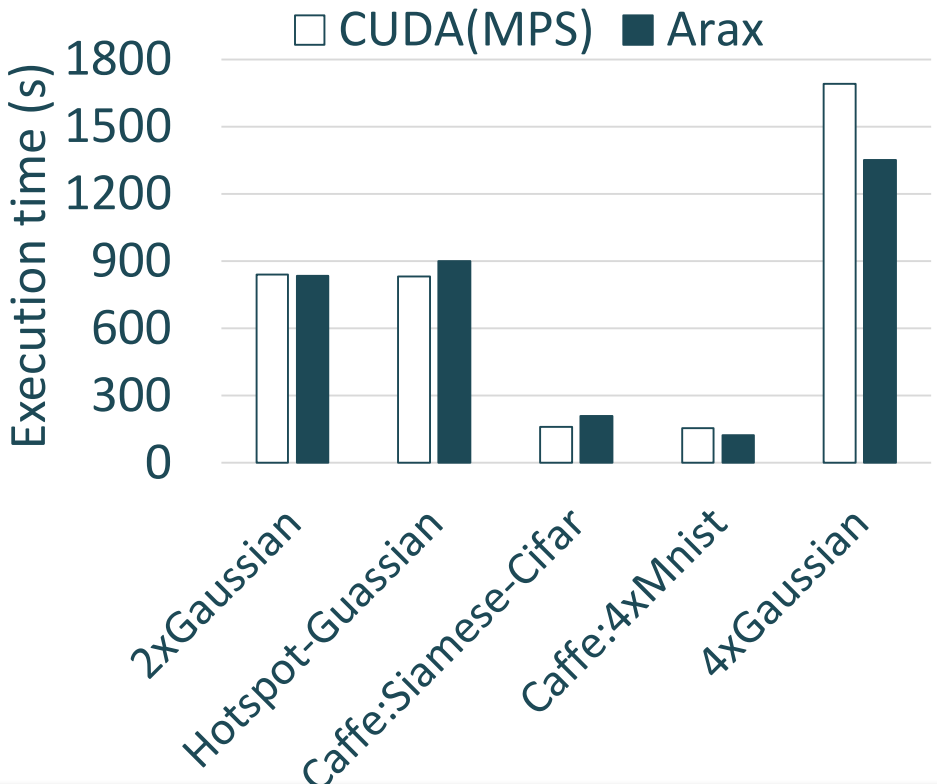
Transparent use of multiple and heterogeneous accelerators

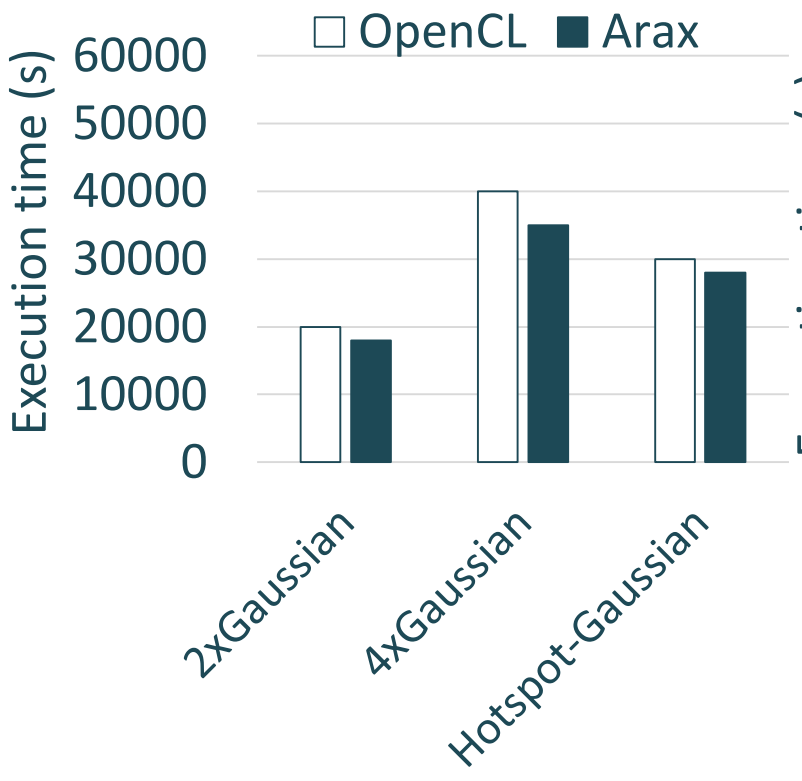
- **One app** uses **multiple** accelerators of the **same** and **different** types → Elastic sharing
- We port **CUDA** Rodinia to Arax API **once**!
  - Then they run transparently to multiple and heterogeneous accelerators

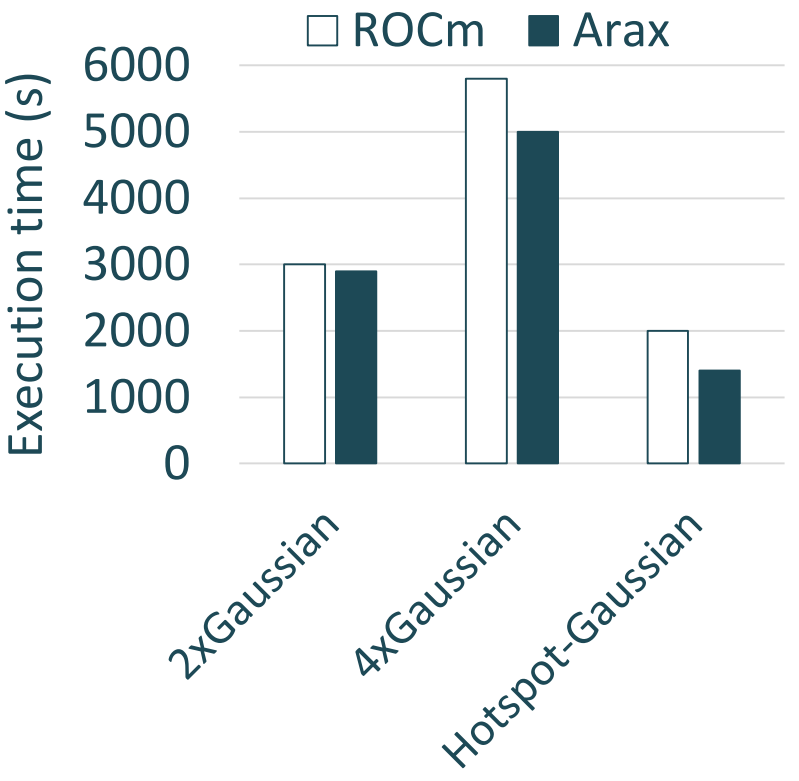


Spatial sharing for heterogenous accelerators

- **Collocate** multiple **apps** to the **same accelerator regardless** of their **type**
  - Several mixes of Rodinia and Caffe that share a single accelerator (NVIDIA-AMD GPU, Intel FPGA)
  - **Comparable performance to native spatial sharing mechanisms**

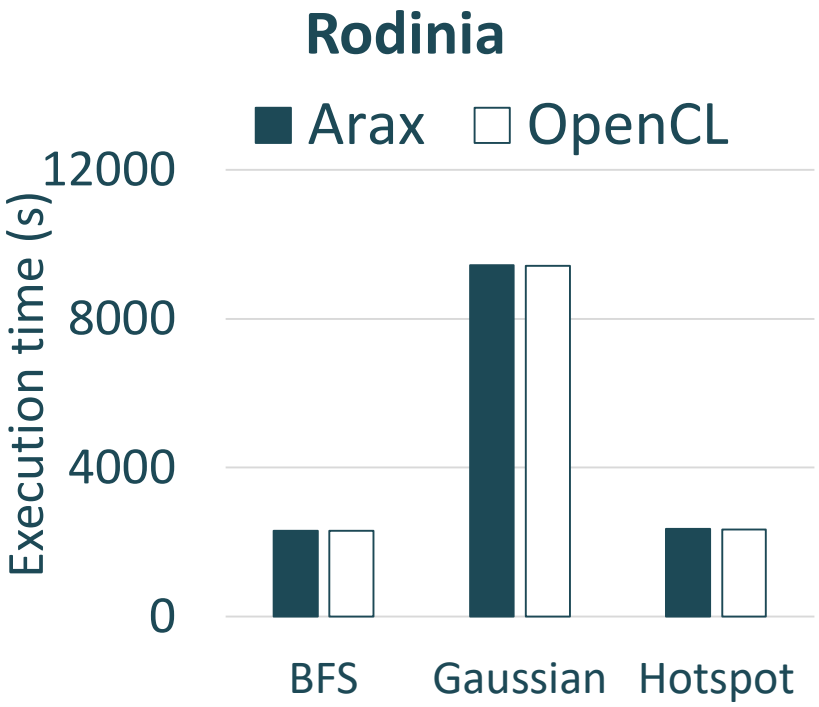






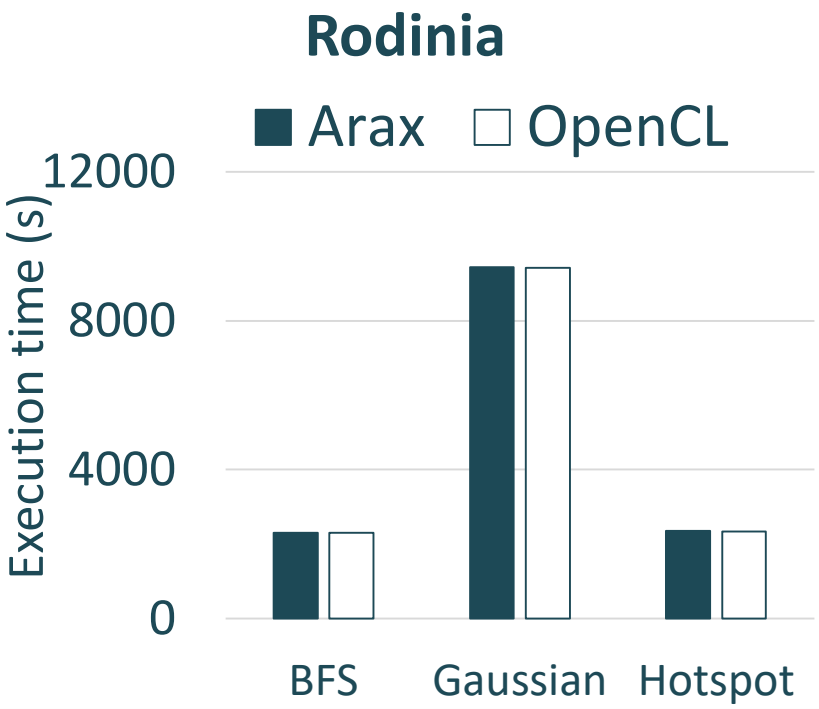
Overhead of Arax compared to native execution

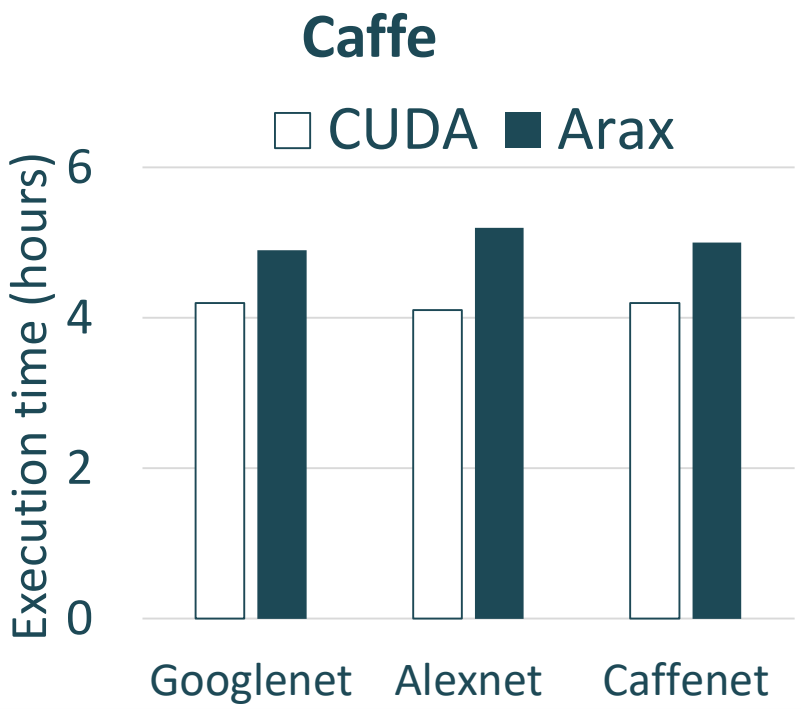
- Arax overhead is mainly due to kernel computation-to-communication (c2c) ratio
  - **High c2c**: up to 5% (BFS, Gaussian, Hotspot, LavaMD, etc.) → common case
  - Low c2c: up to 70% (NW, pathfinder) → rare case

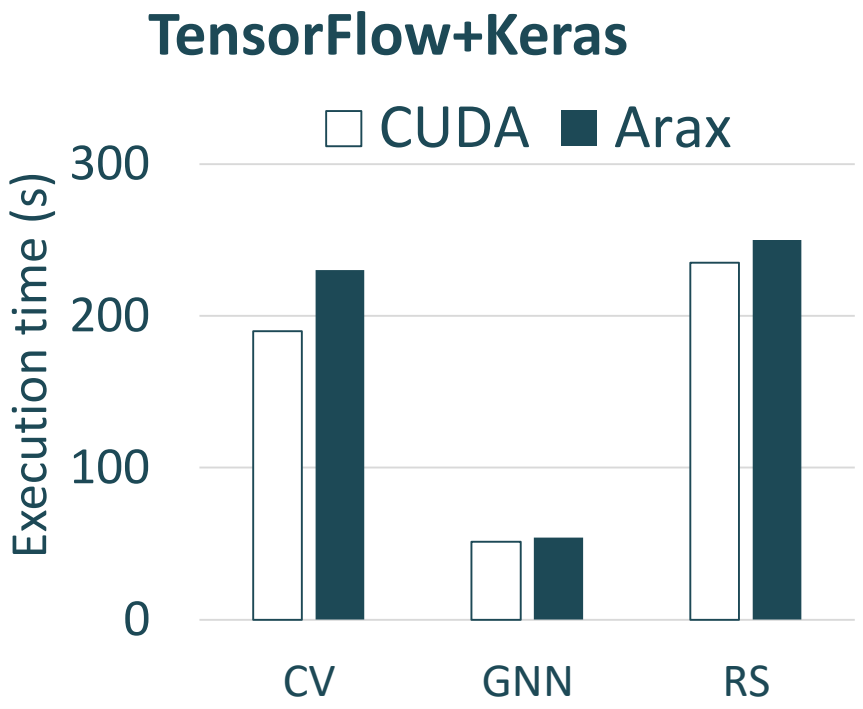


Overhead of Arax compared to native execution

- Arax overhead is mainly due to kernel computation-to-communication (c2c) ratio
  - **High c2c**: up to 5% (BFS, Gaussian, Hotspot, LavaMD, etc.) → common case
  - Low c2c: up to 70% (NW, pathfinder) → rare case
- For **real-world** apps (Caffe, TensorFlow) the overhead is 5-28%







Summary

- Existing approaches assign **statically apps** to **accelerators → under-utilization**

✓ **Arax** is a runtime that enables **elastic** and **spatial sharing** of accelerators by

- **Hiding** the accelerator **type**, **number**, and **set** from applications
- **Assigning tasks** to accelerators **dynamically** at runtime
- **Transferring data just before** task execution
- Offering **transparent heterogeneous** accelerator **spatial sharing**
- **Supporting real-world applications** using an auto-porting tool

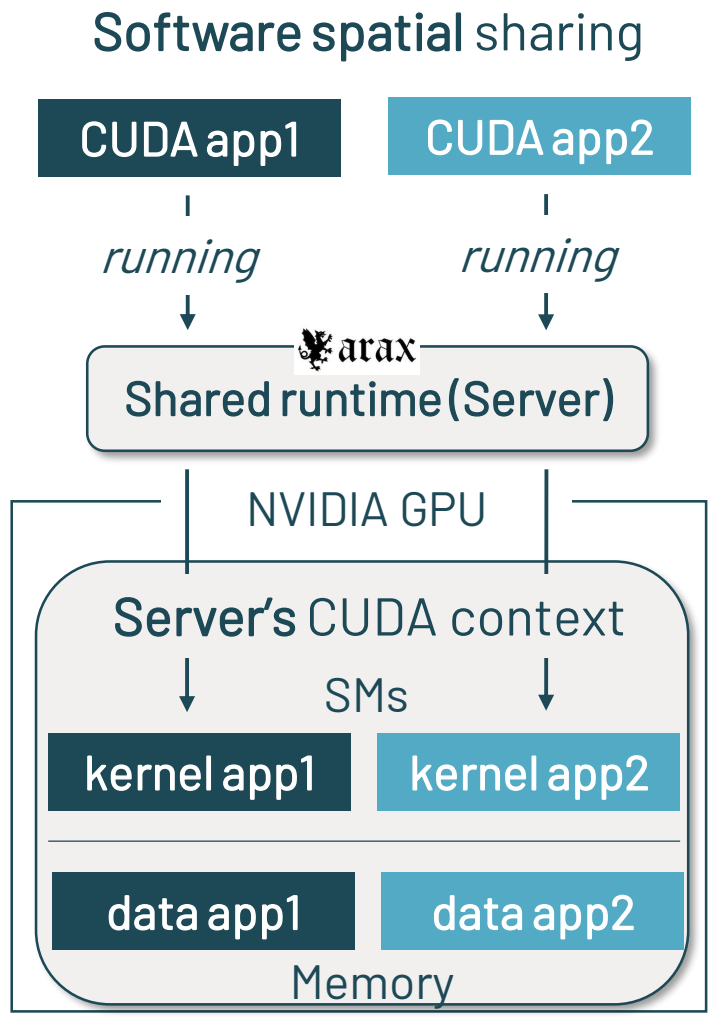
Outline

- Introduction
- Thesis statement and contributions
- Elastic application to accelerator assignment (Arax)
- ➢ **Protected accelerator spatial sharing (Guardian)**
- Conclusions

Arax: A Runtime Framework for Decoupling Applications from Heterogeneous Accelerators, SoCC'22
Guardian: Data Isolation for Multi-Tenant GPU Sharing, Under submission

Software spatial sharing has memory safety issues

- GPU spatial sharing requires a **single GPU context**
  - Common GPU address space

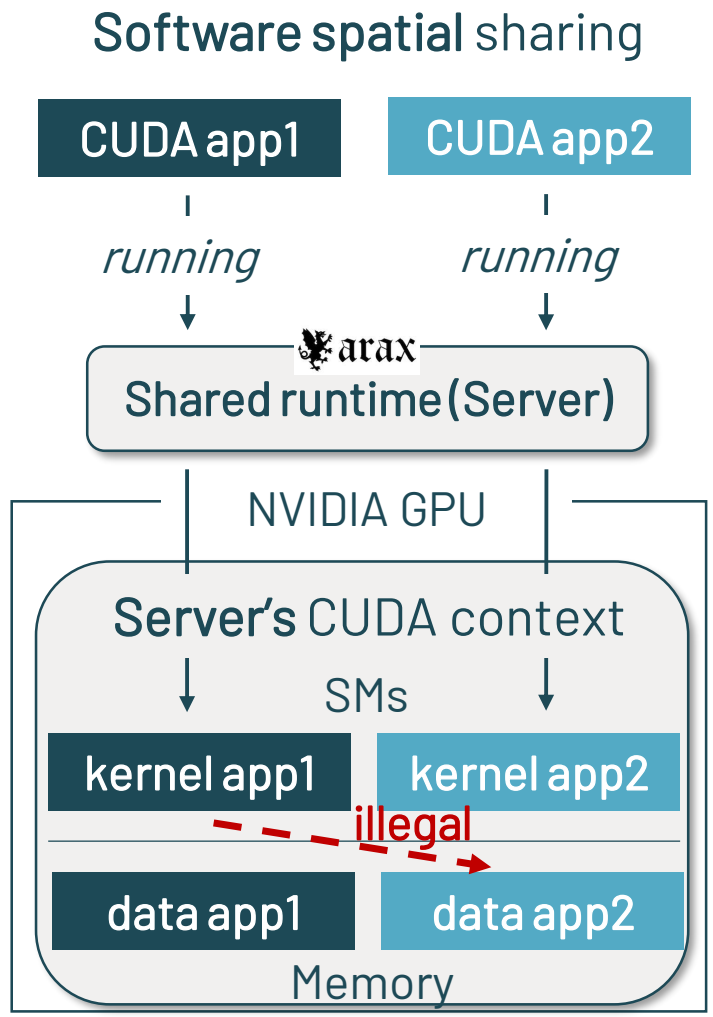


[6] NSDI '18, G-Net: Effective GPU Sharing in NFV Systems
[7] NVIDIA Multi-Process Service (MPS)

Software spatial sharing has memory safety issues

- GPU spatial sharing requires a **single GPU context**
  - Common GPU address space
- An **application** can **read or modify** the data of **another** app
- **Arax** and other approaches [6,7] **do not provide protection**

✓ **Protect memory** and **control flow instructions of kernels** that share a GPU spatially



[6] NSDI '18, G-Net: Effective GPU Sharing in NFV Systems
[7] NVIDIA Multi-Process Service (MPS)

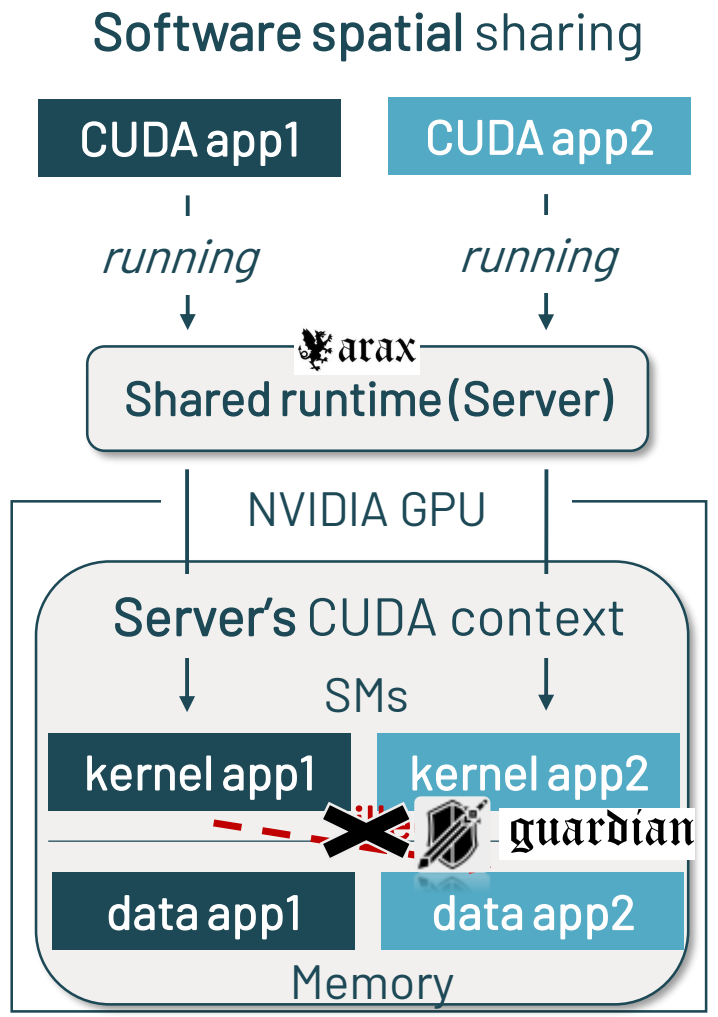
Guardian

- A **GPU protection** approach
  - Makes **Arax sharing safe** and **deployable**

✓ **Divides** the **GPU memory into partitions**
  - **Each partition** is assigned to **only one** application

✓ **Protects memory and control flow instruction of kernels**
  - Add **bound checking** instructions before **loads-stores** and **branches**

✓ **Prohibits** apps from **directly accessing** the **GPU**
  - **Using** the Arax **client-server model**
  - Making **Arax server** a trusted process with **exclusive GPU access**



GPU memory partitioning

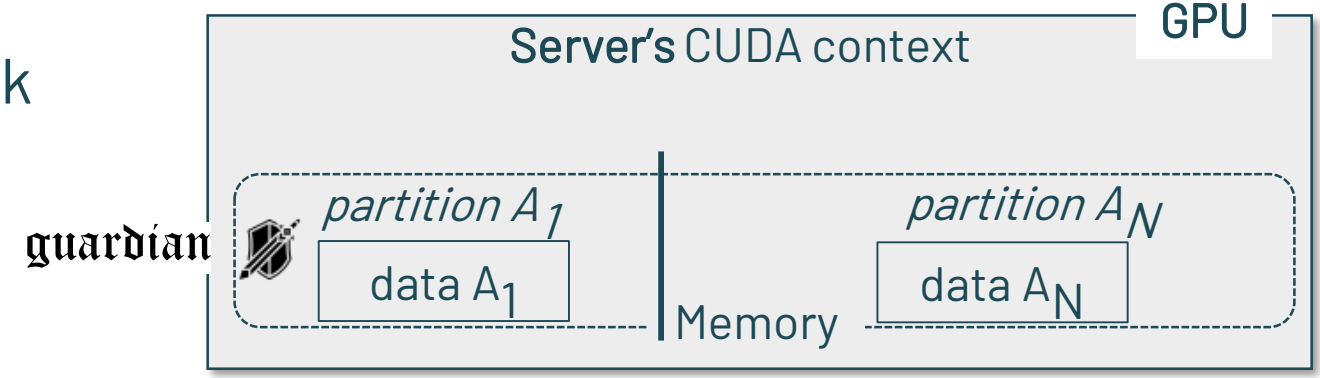
✓ Goal: **Isolate** the **common** GPU **address space**

Guardian uses a **custom allocator**

- Implemented in the Arax server

- The Guardian **allocator**
  - Reserves **all** the GPU memory
  - Splits memory into **partitions**
  - Assigns a **partition** exclusively to an **application**

- A **partition** is a **contiguous memory** block
  - To reduce the overhead + metadata

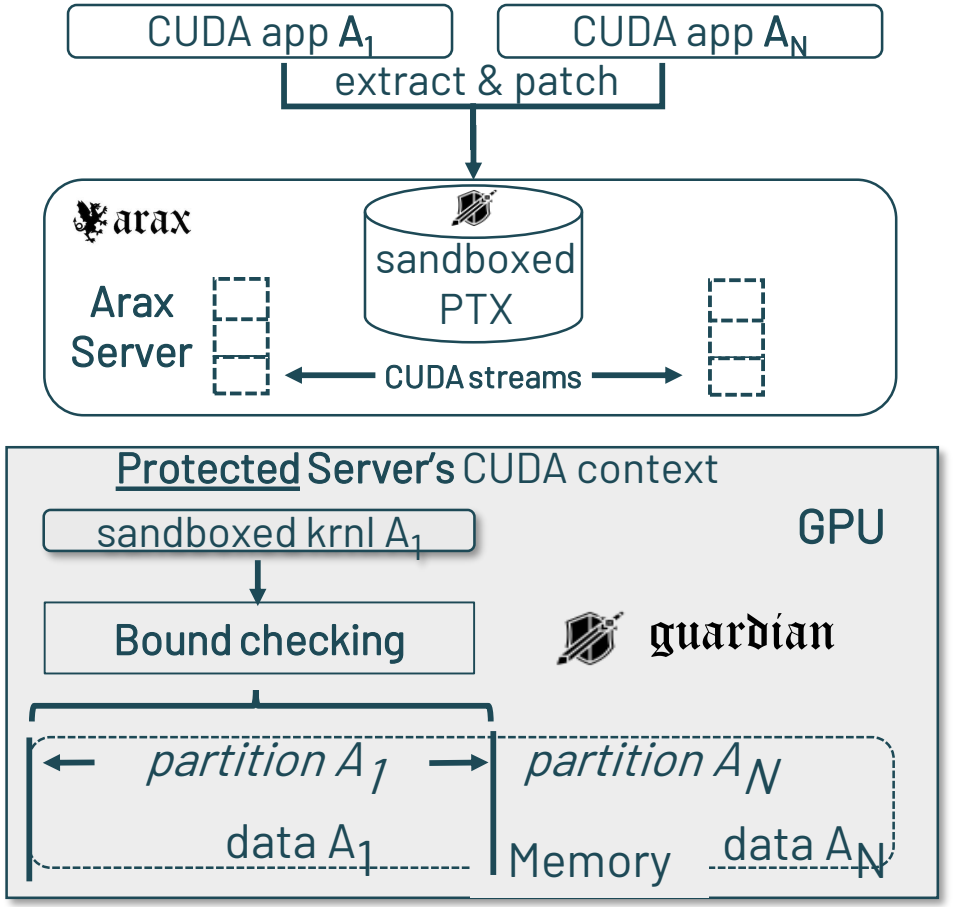


Protect GPU kernels

✓ Goal: **Lightweight** checks for **memory access**

- Guardian **once** during an **offline** phase
  1. **Extracts** kernel **PTX** available even in closed-source libs
  2. **Adds bounds checking** instructions before *ld* and *st*
  3. **Compiles** the sandboxed **PTX**

- We examine three bound checking approaches
  - Address checking (If-checks)
  - Address fencing bitwise AND-OR
  - Address fencing modulo

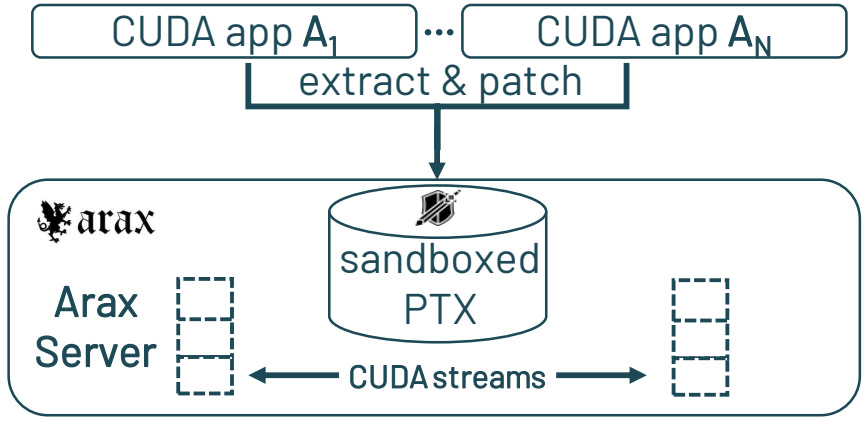


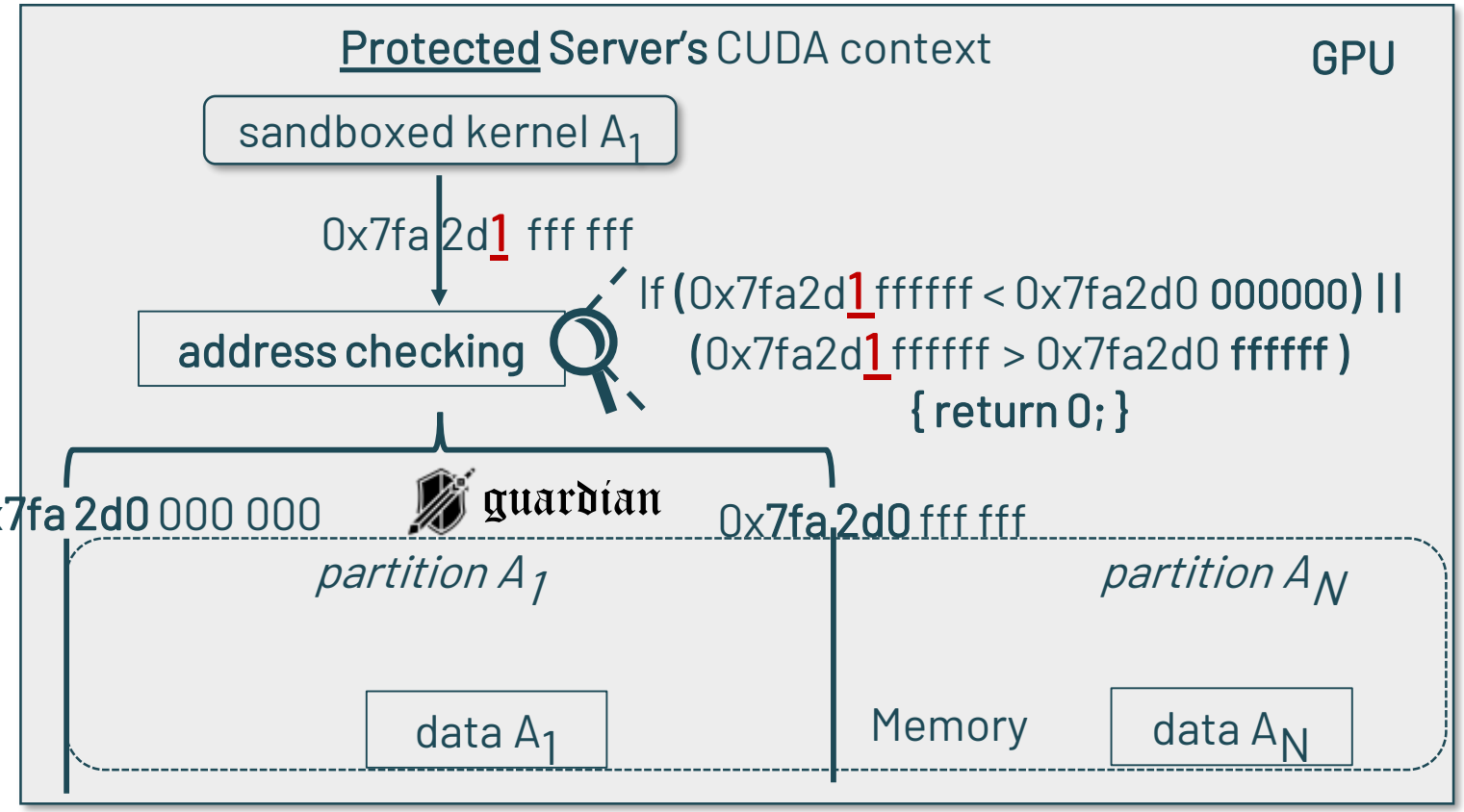
Address checking (if checks)

- Check each address against partition bounds

+ Offers illegal access **detection**

- **High overhead** → 80 cycles





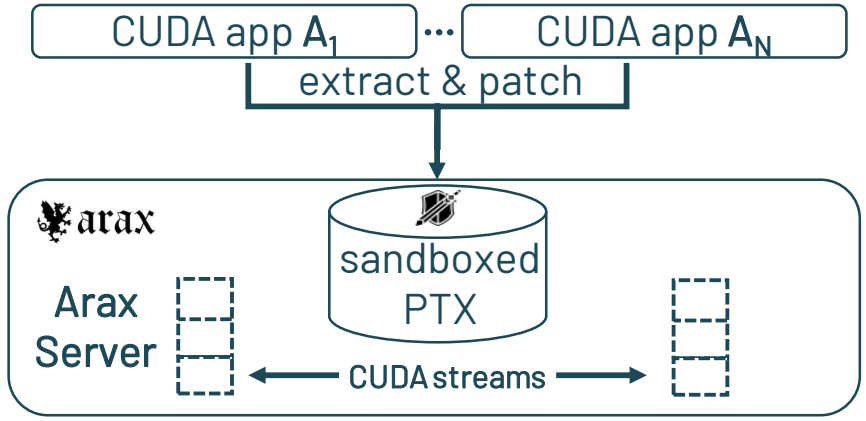
Address fencing with bitwise AND-OR

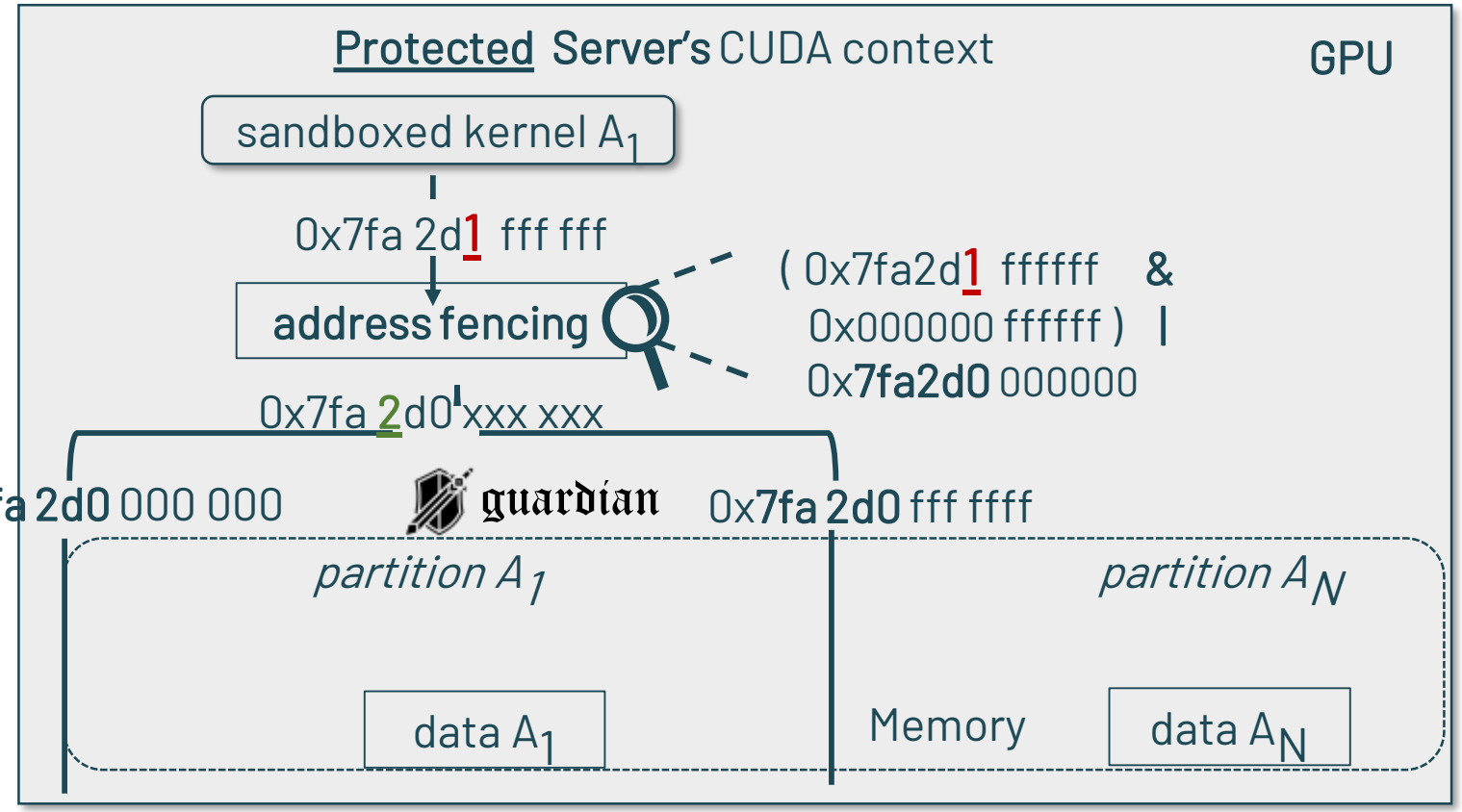
- An illegal address will wrap around

+ **Low overhead** → 8 cycles

- **No** illegal address **detection**

- **Power-of-two partition size**





Address fencing with modulo

- An illegal address will wrap around
  - fenced_addr = part_base + ((illegal_addr - part_base) % part_size)

\+ **No power-of-two partition size**

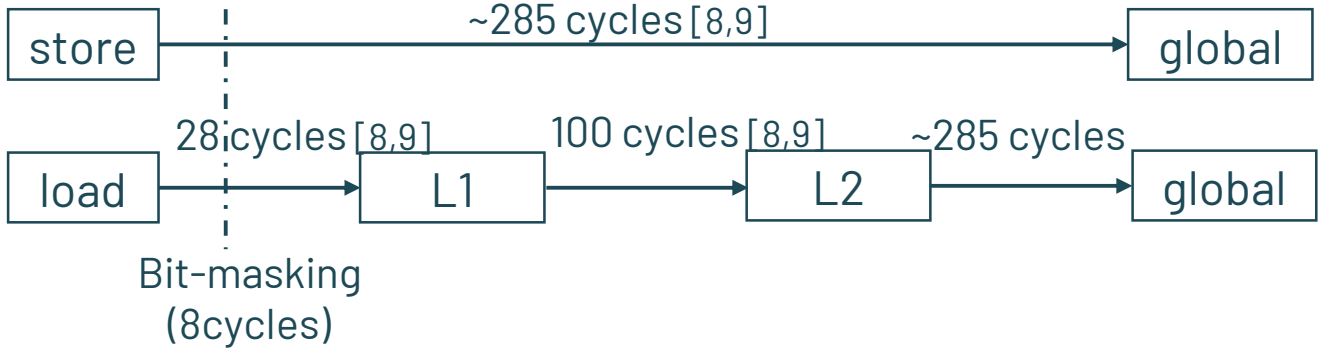
\- No illegal address detection

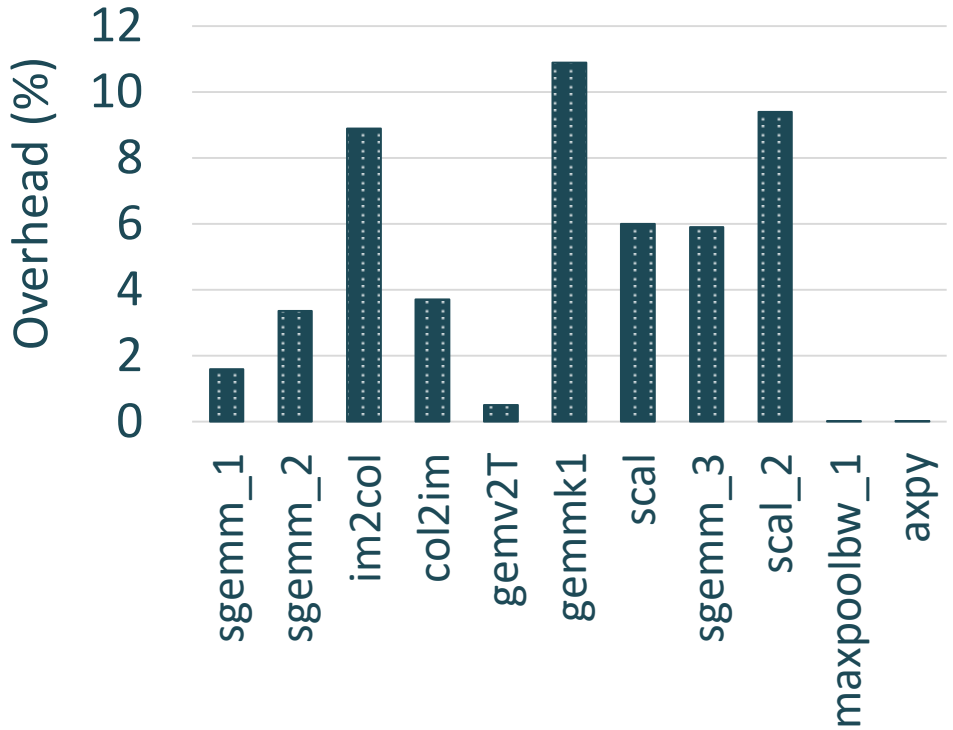
- **Medium overhead** → 28 cycles
  - Using our inline modulo implementation (3x instructions)
  - No modulo 64bit in CUDA ISA

Approach	Overhead	Illegal access detection	Power-of-two partition size
If-checks	High	Yes	No
Bitwise	Low	No	Yes
Modulo	Medium	No	Yes

Overhead of bit-masking (AND-OR) per kernel

- Lenet **sandboxed kernels** overhead is on **average 3.2%** compared to native
- **Bit-masking** overhead depends on
  - The **latency** of **loads** and **stores**





- **Cache hit ratio**
  - Caffe and PyTorch: L1 hit ratio → 37% and L2 → 72%

[8] Is Data Placement Optimization Still Relevant On Newer GPUs?, OSTI'18
[9] Dissecting the NVIDIA volta GPU architecture via microbenchmarking, Arxiv

Protect control flow instructions

✓ Goal: **Prevent jumping over** bound checks

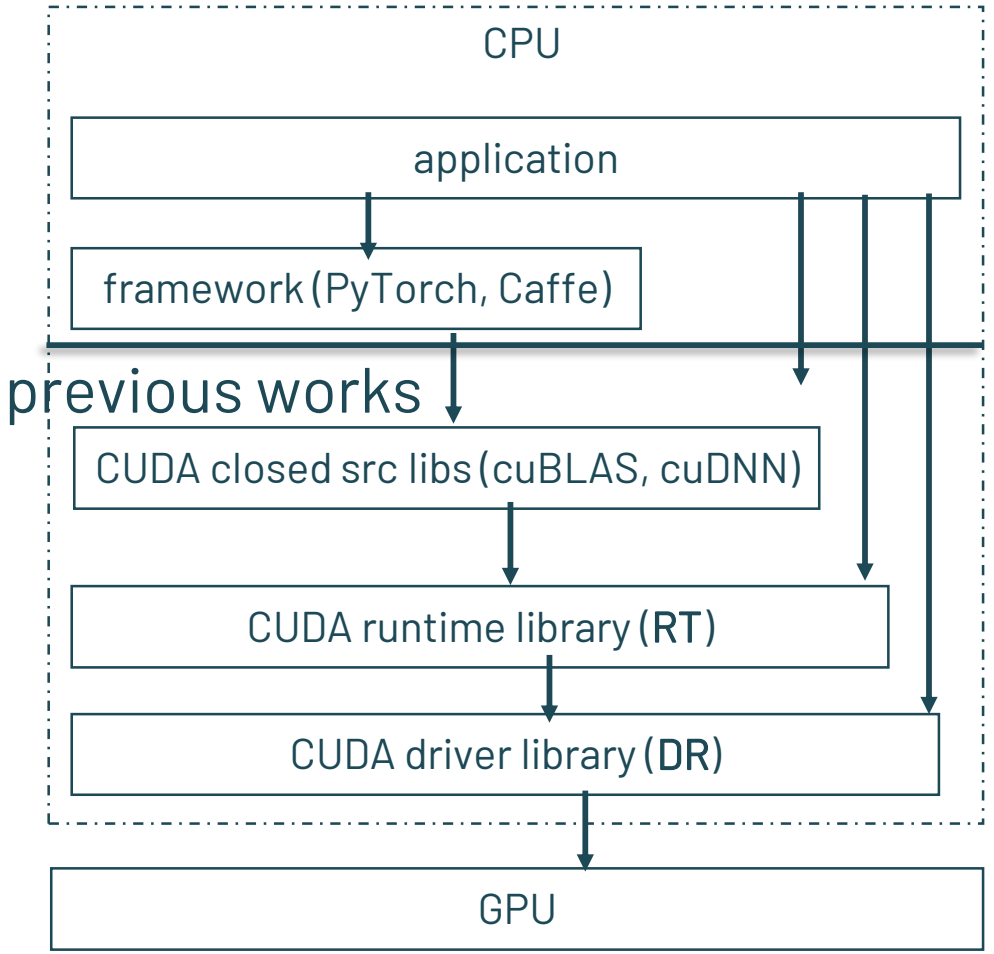
- **Direct branches** are **safe**
  - Jump to **labels defined** in a PTX
  - **Wrong labels** lead to **compilation errors**
- **Indirect branches** are **unsafe**
  - Use a **register** to **index** an array of labels
  - This **register** can **not** be validated at **compile** time
- Guardian applies a **mask** to the **index** relative to the array size

High-level libraries perform "implicit" CUDA RT/DR calls

✓ Goal: **Protect** high-level **calls** of **closed-source accelerated** libs (e.g., cuBLAS, cuDNN, cuFFT)

- Real-world apps use **heavily** accelerated libraries
- **Accelerated libs** contain **high-level** function **calls** that perform **implicit CUDA RT** and **DR** calls
  - cublasIsamax: cudaMalloc, cudaMemcpy, cudaLaunch
- **Previous works** treated these calls as **black-box**

➢ **No** protection



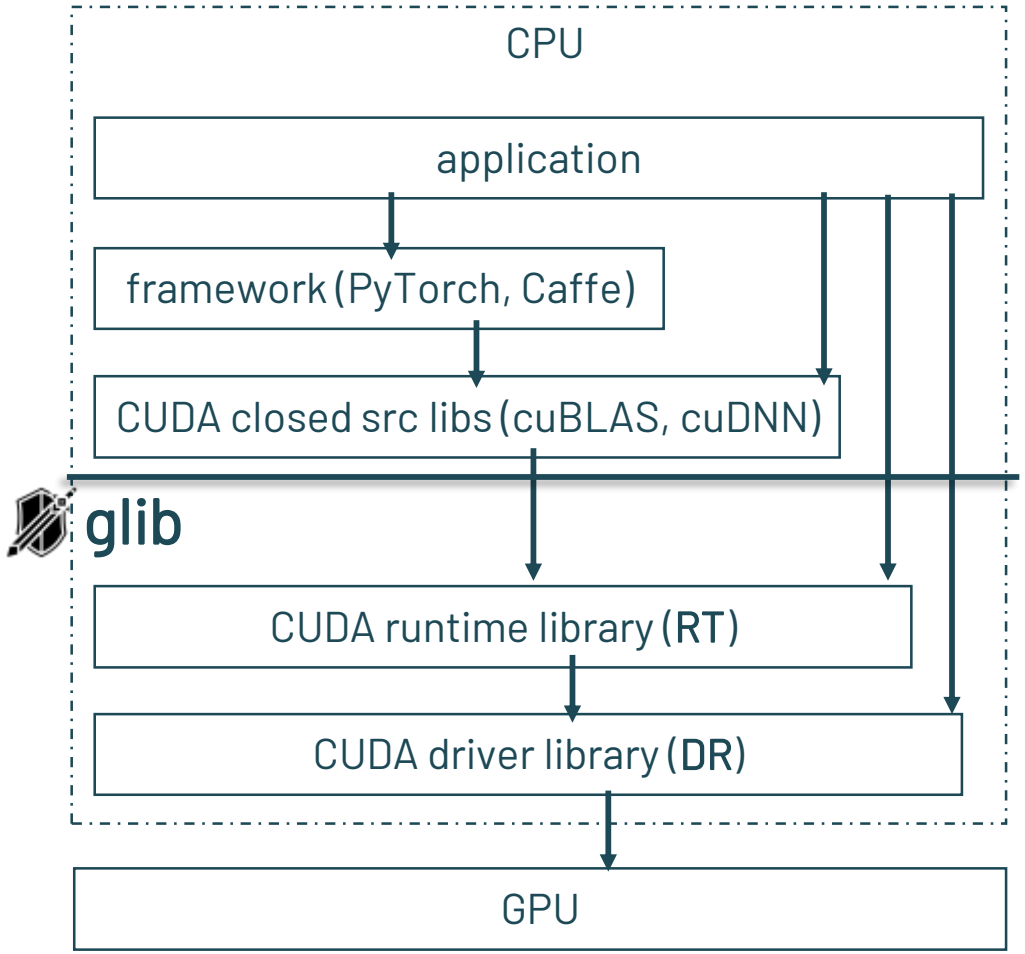
High-level libraries perform "implicit" CUDA RT/DR calls

✓ Goal: **Protect** high-level **calls** of **closed-source accelerated** libs (e.g., cuBLAS, cuDNN, cuFFT)

- Real-world apps use **heavily** accelerated libraries
- **Accelerated libs** contain **high-level** function **calls** that perform **implicit CUDA RT** and **DR** calls
  - cublasIsamax: cudaMalloc, cudaMemcpy, cudaLaunch
- **Previous works** treated these calls as **black-box**

➢ **No** protection

- Guardian intercepts low-level **RT** and **DR** libs
  - Using glib: a dynamically loaded library
- Guardian apps need to link with the static version of CUDA accelerated libs



Prevent bypassing Guardian checks

✓ Goal: **Disallow direct GPU access** from applications

- **Guardian** inherits Arax's **client-server architecture**
  - **Applications** or clients run in a **different address space** than the server
- **CUDA calls** are **intercepted** at the **client** side and **forwarded** to the server
- The **server** is the **only entity** with **access** to **GPUs**
  - **Receives, checks,** and **executes all GPU calls** on behalf of applications

Prevent bypassing Guardian checks

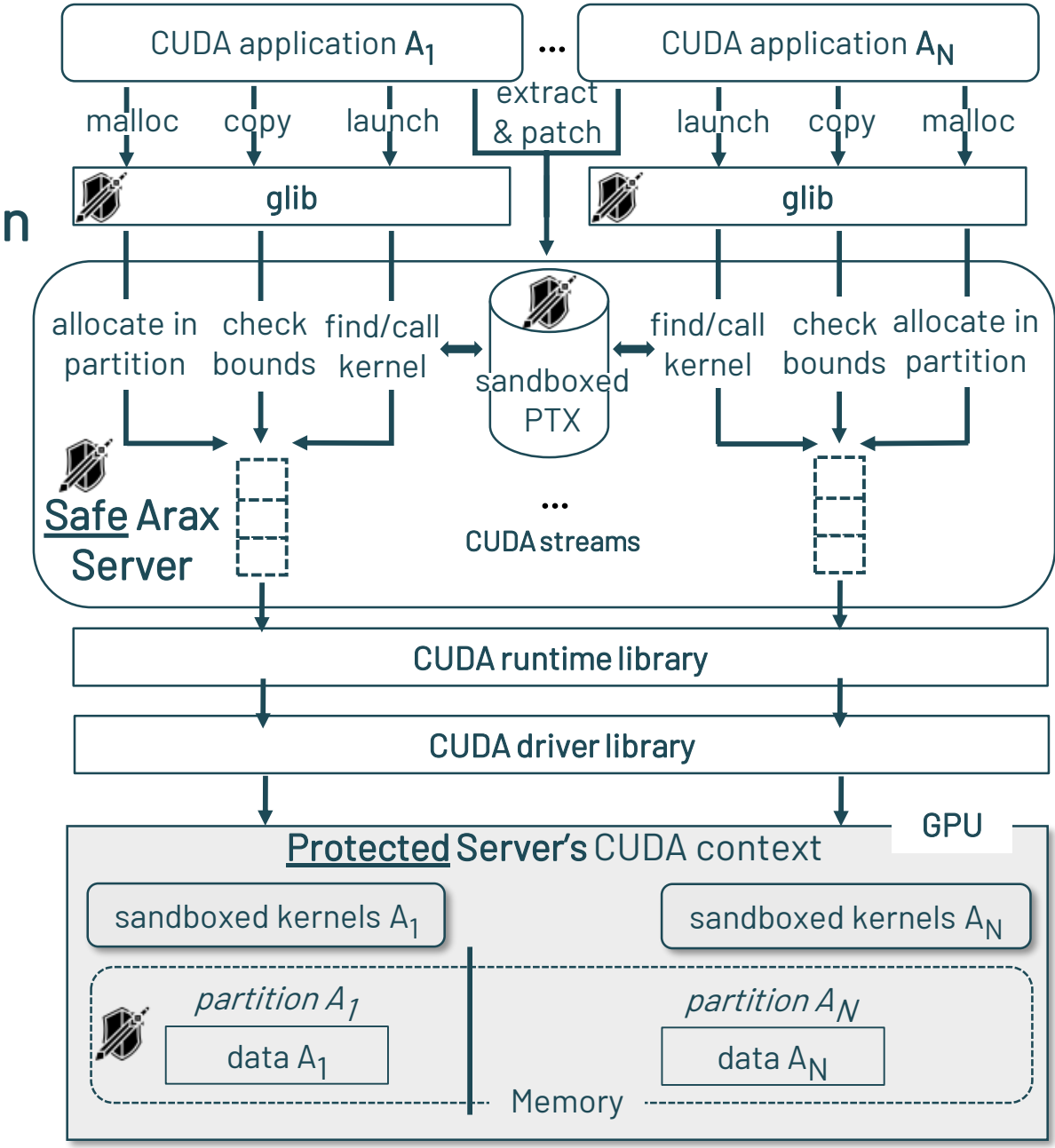
✓ Goal: **Disallow direct GPU access** from applications

- **Guardian** inherits Arax's **client-server architecture**
  - **Applications** or clients run in a **different address space** than the server
- **CUDA calls** are **intercepted** at the **client** side and **forwarded** to the server
- The **server** is the **only entity** with **access** to **GPUs**
  - **Receives, checks,** and **executes all GPU calls** on behalf of applications
- Guardian's interception approach is **more robust** than previous works [10,11, 12]
  - Guardian intercepts **only** CUDA **runtime** and **driver** library: ~**430** CUDA calls
  - **Previous works** intercept **and** high-level calls of CUDA accelerated libs **> 1600** calls
  - **Previous works** maintain **more calls and high-level** calls are **complex** and **change** rapid

[10] Europar'20, Cricket: A virtualization layer for distributed execution of CUDA applications with checkpoint/restart support
[11] IPDPS'22, DSGF: Disaggregated GPUs for Serverless functions
[12] SoCC'22, Arax: A Runtime Framework for Decoupling Applications from Heterogeneous Accelerators

Guardian CUDA call invocation

- **Malloc** returns ptrs **inside** each app's **partition**
- A **copy succeeds** if:
  - **Src** and **dst** pointers are **inside** the **partition**
- For each kernel **launch**
  - Call the **sandboxed** version of the **kernel**
  - Pass **extra** parameters: **mask** + **base** address



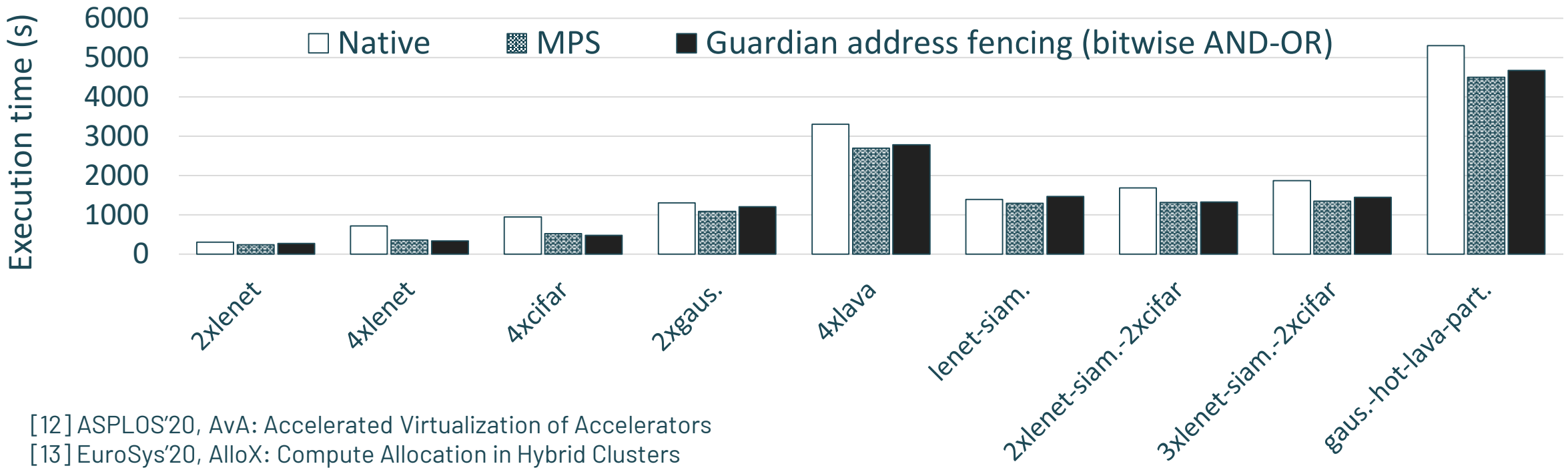
Testbed

- Two server configurations with different GPUs
  - NVIDIA RTX A4000 and NVIDIA GeForce RTX 3080 Ti
- Microbenchmarks and real-world applications
  - Rodinia benchmarks suite (issuing *hundreds* of kernels)
  - Caffe deep learning framework (issuing *billions* of kernels)
  - PyTorch machine learning framework (issuing *billions* of kernels)
- We evaluate Guardian using two deployments
  - Spatial sharing
  - Standalone applications

Lib/Framework	#kernels	#loads	#stores
cuBLAS	4115	341249	106399
cuFFT	5173	175256	371932
cuRAND	204	4949	3610
Rodinia	23	544	285
Caffe	1294	87267	32946
PyTorch	27987	2083978	857987

GPU sharing

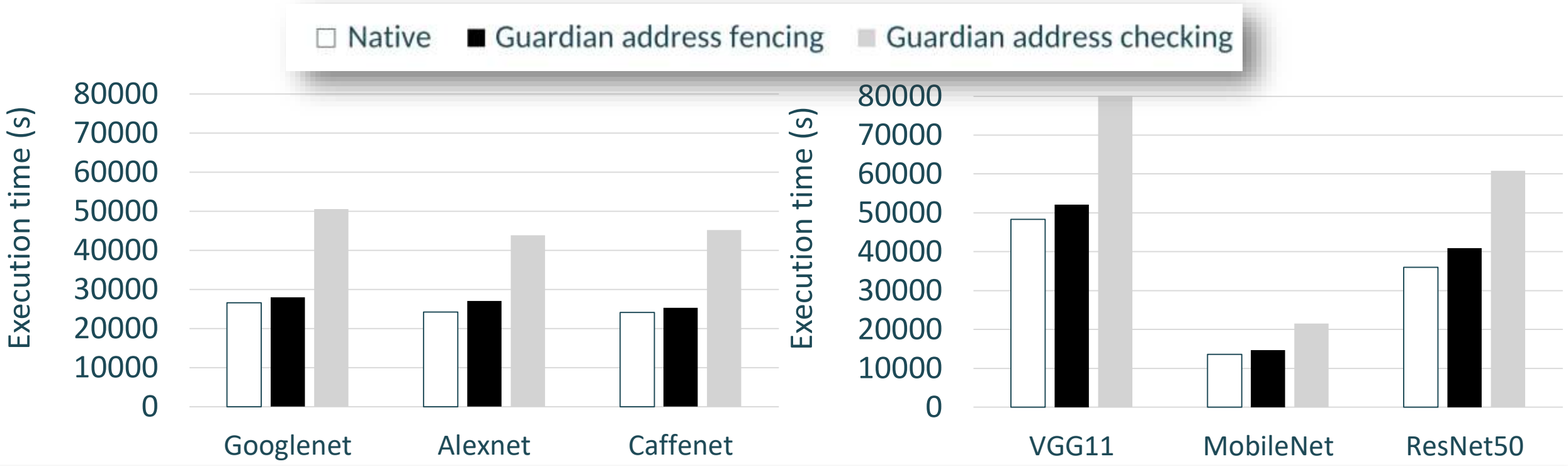
- Compare Guardian with
  - MPS: No protection **nor multi-tenancy (**only applications **from the same user co-execute)**
  - Native CUDA runtime: **Time-sharing** used from **previous works** [12,13]
- **Comparable** performance to **MPS** and up to **2x better** to **Native** CUDA runtime



[12] ASPLOS'20, AvA: Accelerated Virtualization of Accelerators
[13] EuroSys'20, AlloX: Compute Allocation in Hybrid Clusters

Overhead of Guardian without sharing

- Includes call interception and checking overheads
- Address **fencing** overhead is from **4.5% - 12%** compared to **native CUDA**
- Address **checking** has **1.7x worst** execution time compared to **native CUDA**



Summary

- Guardian is a **GPU protection** approach that enables **safe spatial sharing**
  - It is **easily deployable:** No extra HW or application/kernel source code
  - It **supports closed-source** libs: RT/DR interception + PTX instrumentation
  - It incurs **low overhead:** Address-fencing (bitwise AND-OR)

Outline

- Introduction
- Thesis statement and contributions
- Elastic application to accelerator assignment (Arax)
- Protected accelerator spatial sharing (Guardian)

➢ **Conclusions**

Arax: A Runtime Framework for Decoupling Applications from Heterogeneous Accelerators, SoCC'22
Guardian: Data Isolation for Multi-Tenant GPU Sharing, Under submission

Conclusions

We design and implement a runtime that enables elastic and spatial accelerator sharing for real-world applications

- Our approach has the following features
  - ✓ **Enables per-task dynamic** accelerator **assignment** → elastic sharing
  - ✓ **Protects memory** and **control flow** instructions → spatial sharing

Future work

- Use zero-copy techniques to minimize the extra copy overhead
- Compile PTX to LLVM-IR to support Intel and AMD GPUs
  - To run complex frameworks to heterogeneous accelerators
- Integrate Arax to a resource manager to support distributed environments
- Implement a more efficient GPU allocator to reduce wasted space

Acknowledgements

- FORTH-ICS Graduate Scholarships → September 2017 – now
  - Vineyard (GA 687628)
  - EVOLVE (GA 825061)
  - EUPILOT (GA 101034126)
  - DEEP-SEA (GA 955606)
  - HiPEAC (GA 871174)

Publications

1. Stelios Mavridis, **Manos Pavlidakis**, Ioannis Stamoulias, Christos Kozanitis, Nikos Chrysos, Christoforos Kachris, Dimitrios Soudris, and Angelos Bilas. 2017. VineTalk: Simplifying software access and sharing of FPGAs in datacenters. In Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL '17).
2. **Manos Pavlidakis**, SteliosMavridis, Nikos Chrysos, and Angelos Bilas. 2020. TReM: A Task Revocation Mechanism for GPUs. In Proceedings of the 22th IEEE International Conference on High Performance Computing and Communications (HPCC '20).
3. **Manos Pavlidakis**, Stelios Mavridis, Antony Chazapis, Giorgos Vasiliadis, and Angelos Bilas. Arax: A Runtime Framework for Decoupling Applications from Heterogeneous Accelerators. In Proceedings of the 13th ACM Symposium on Cloud Computing (SoCC '22).
4. **Manos Pavlidakis**, Giorgos Vasiliadis, SteliosMavridis, Anargiros Argiros, Antony Chazapis, and Angelos Bilas. Guardian: Data Isolation for Multi-Tenant GPU Sharing. (Under submission).

Transparent spatial sharing of multiple and heterogeneous accelerators

Manos Pavlidakis
manospavl@ics.forth.gr

Questions?

PhD defense