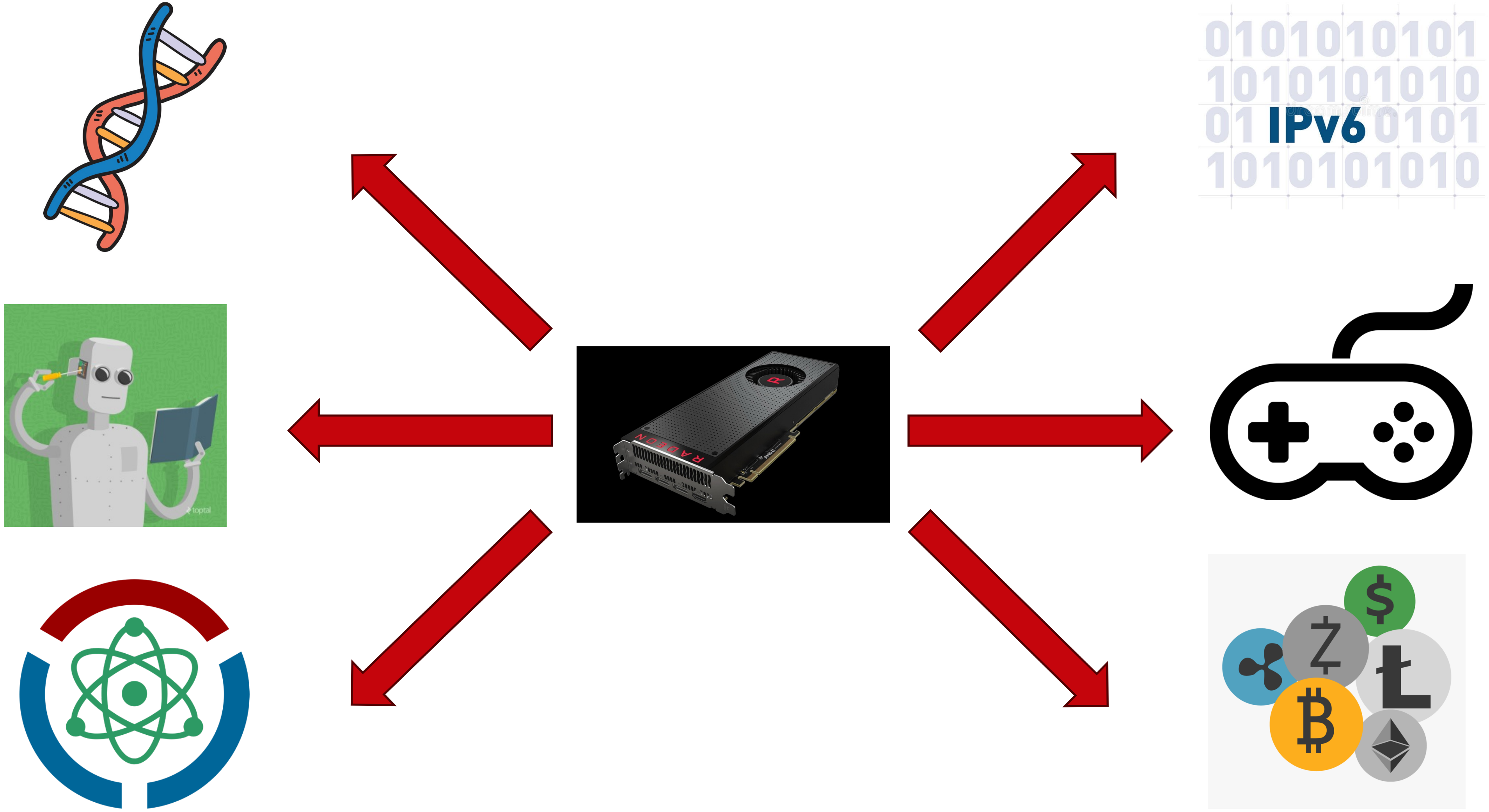
Simulation Support for Fast and Accurate Large-Scale GPGPU & Accelerator Workloads

Vishnu Ramadas*, Matthew Poremba^, Bradford M. Beckmann^, and Matthew D. Sinclair*^

*University of Wisconsin-Madison, ^AMD Research & Advanced Development

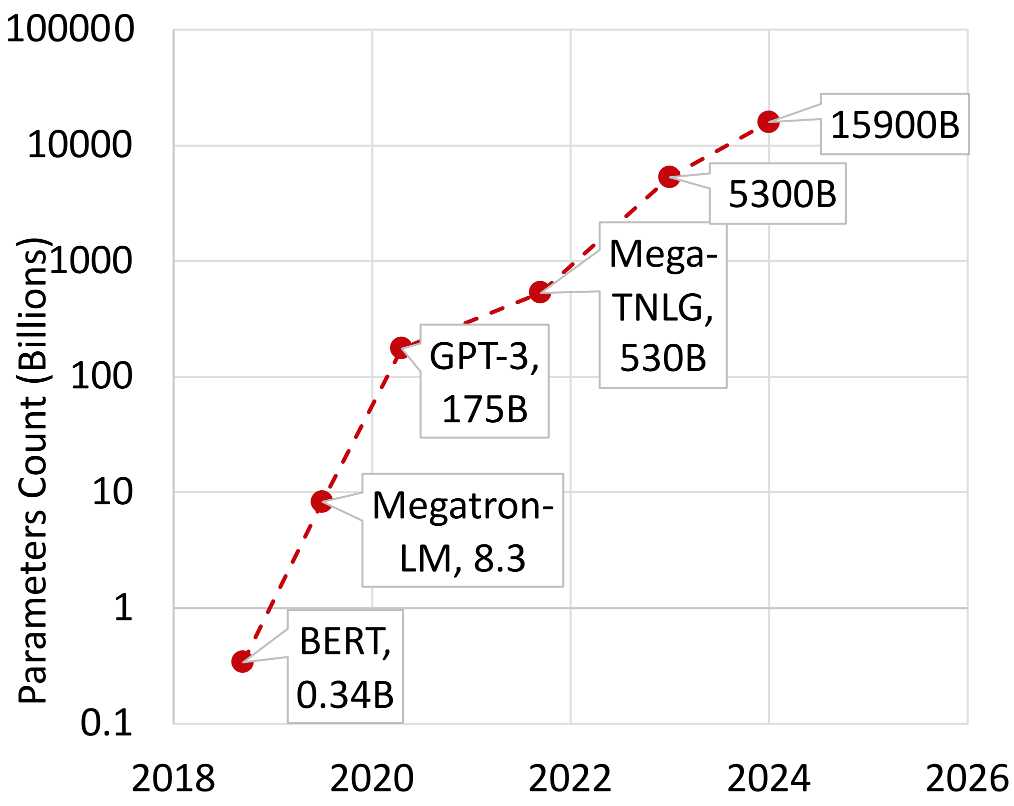
vramadas@wisc.edu

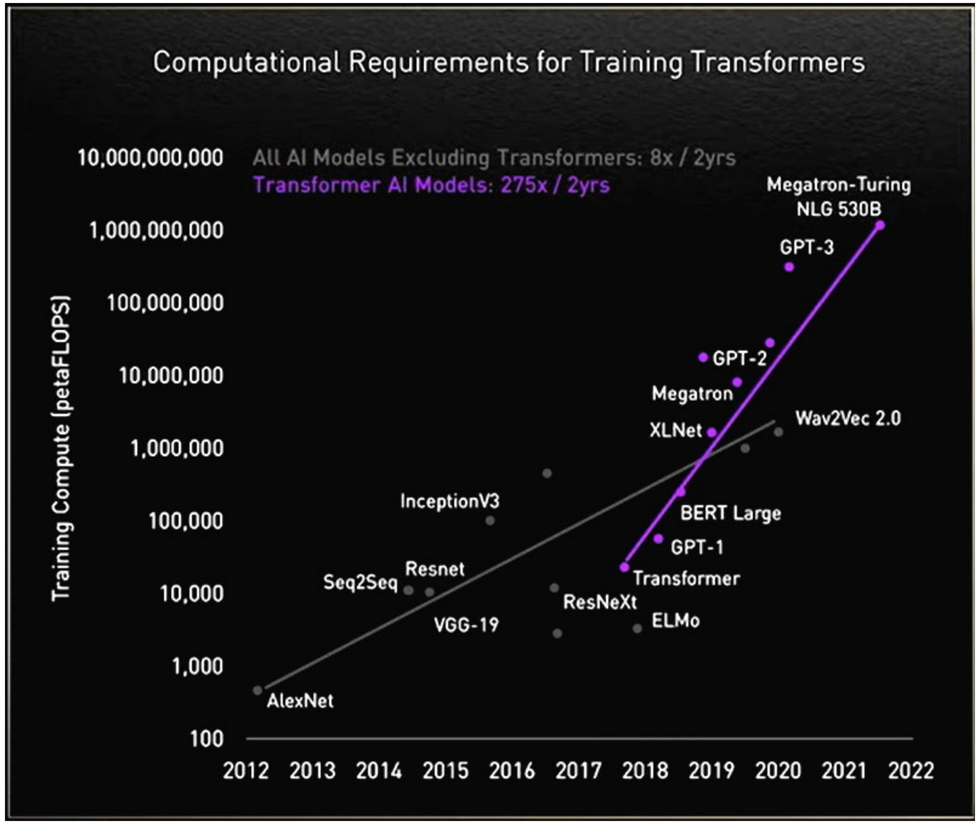
Applications are Increasingly Diverse



High fidelity tools crucial for early-stage design exploration, optimizations for these workloads

Key Challenge 1: Application Scaling





Simulating entire workloads would take months (or years) in modern cycle level simulators

Need to do better!

Source:
1. https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/
2. https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/

Key Challenge 2: Simulator Capabilities

	Fast	Full System	High Fidel.	Scalable	No RTL
gem5	✘	✓	✓	✘	✓
SST	✓	✘	✘	✓	✓
FireSim	✓	✓	✓	✘	✘
Our Goal	✓	✓	✓	✓	✓

Can we enhance simulators to overcome their downsides?

Why gem5?

- gem5 uniquely suited to model systems with CPUs, GPUs, and accelerators
  - Can model both homogeneous and heterogeneous systems
  - Widely used in academia, industry, and national labs (6000+ citations)
  - Can research µarch, caches, main memory, I/Cs, interfaces, coherence, consistency, …
  - Models entire system, including OS and runtime – not reliant on external tools, traces
  - Full system effects likely increasingly important as application diversity and co-design increase

However, must efficiently support modern accelerator workloads

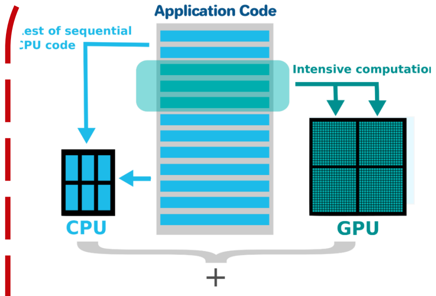
How Can We Scalably Run Large-Scale Workloads in gem5?

- Holistic solution that scalably runs modern frameworks in gem5:
  - Key components:

PyTorch

TensorFlow

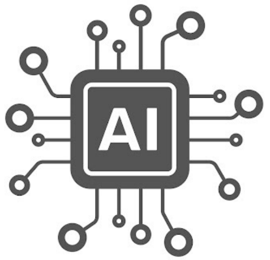
Large-Scale App Support



Mixed-Fidelity Simulations

Application Checkpointing

Scalable Simulations



Custom Accelerators

Universality

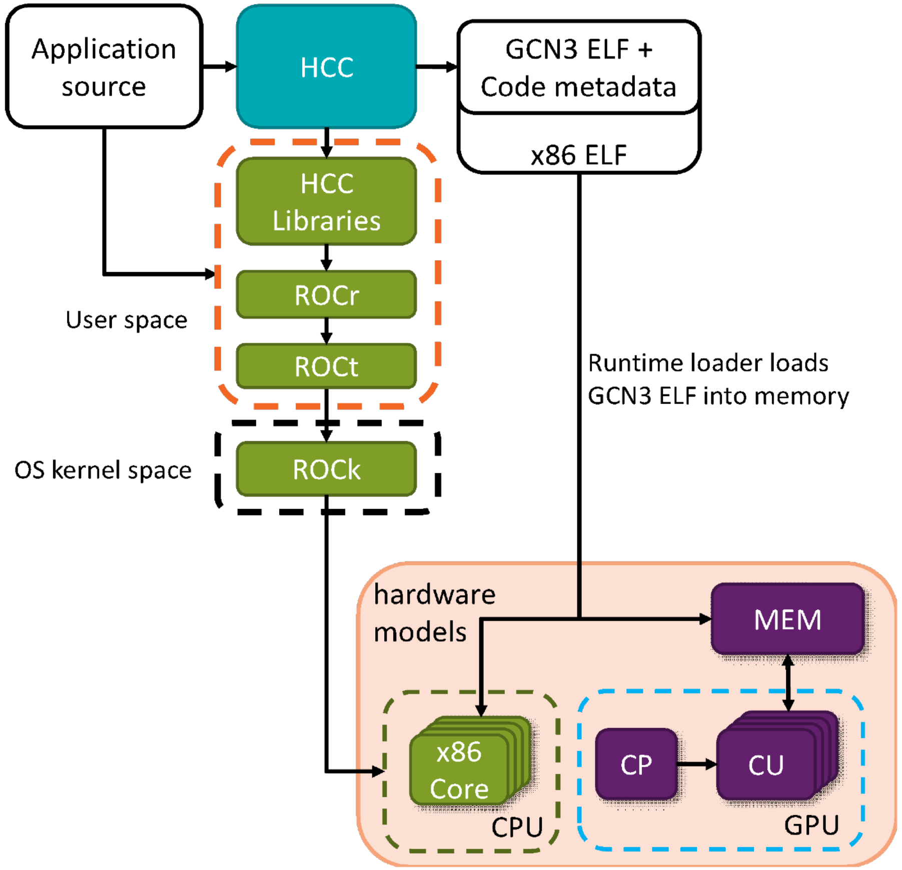
Preliminary Results: Only 1.58x - 3x slower than bare metal

Today's Focus: GPGPUs

Outline

- Motivation
- **Background**
- Design
- Conclusion & Future Work

CPU-GPU Support in gem5

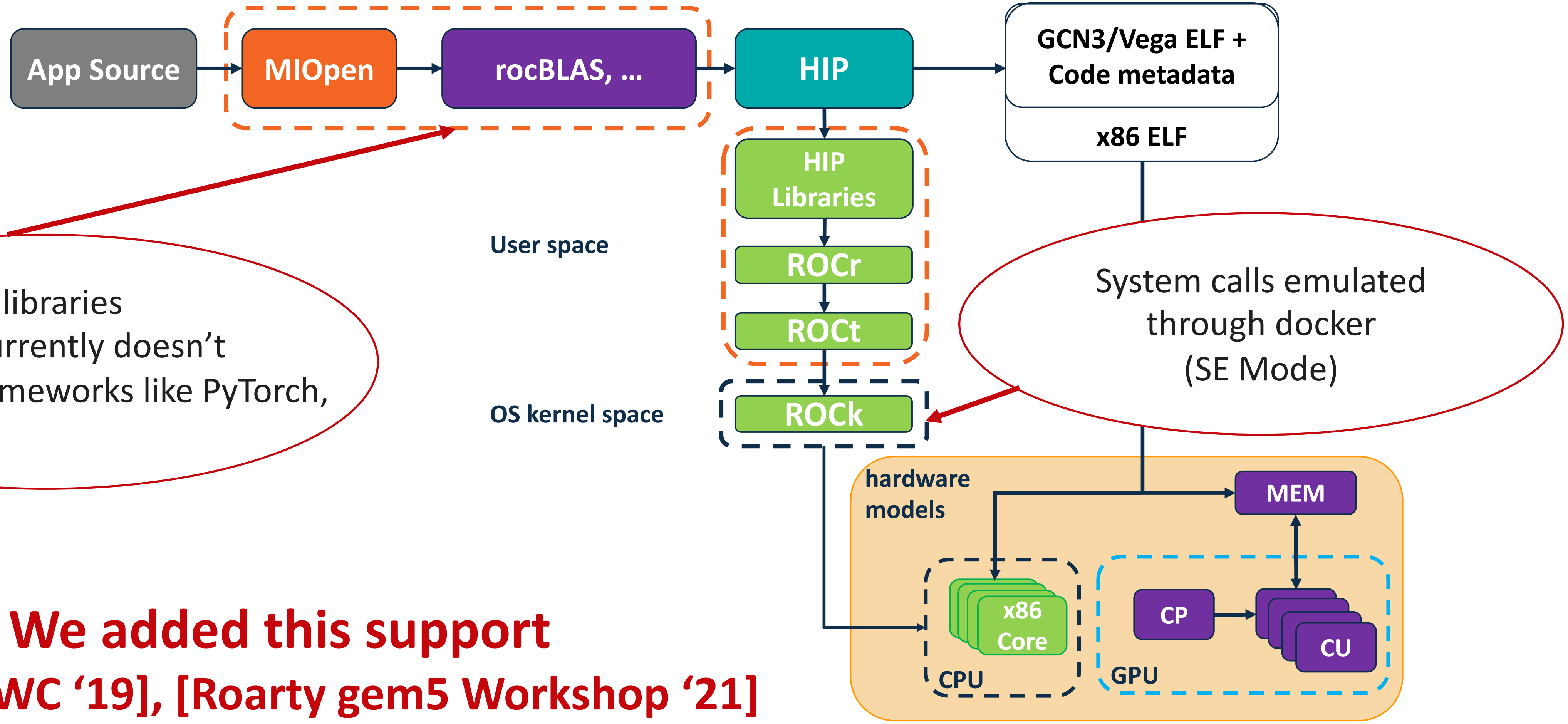


- Supports complex systems with CPUs, GPUs, interconnects, memory, etc.
  - Execution-driven, cycle-level
  - ISA: Alpha, ARM, MIPS, PowerPC, RISC-V, SPARC, x86
  - CPUs: AtomicSimple, TimingSimple, KVM, Minor, O3
- Current GPU models [Gutierrez, et al. HPCA '18]
  - Simulates HIP applications (AMD's GPGPU language)
  - Recently added support for MI200/MI300 GPUs

gem5's GPU Simulation modes

- AMD GPUs: ROCm (Radeon Open Compute) stack to interface with CPU(s)
- ROCm stack:
  - Runtime layer – ROCr
  - Thunk (user-space driver) ROCt
  - Kernel fusion driver (KFD) – ROCk (in Linux)
  - MIOpen – machine intelligence (MI/ML) library
  - rocBLAS – BLAS (e.g., GEMMs) library
  - HIP (roughly: LLVM backend, clang front-end)
- Syscall emulation (SE) mode: simulate all except ROCk, which gem5 emulates via docker
- Full system (FS) mode: simulated disk image containing the entire ROCm stack

gem5 CPU-GPU SE Mode Modern Workload Support



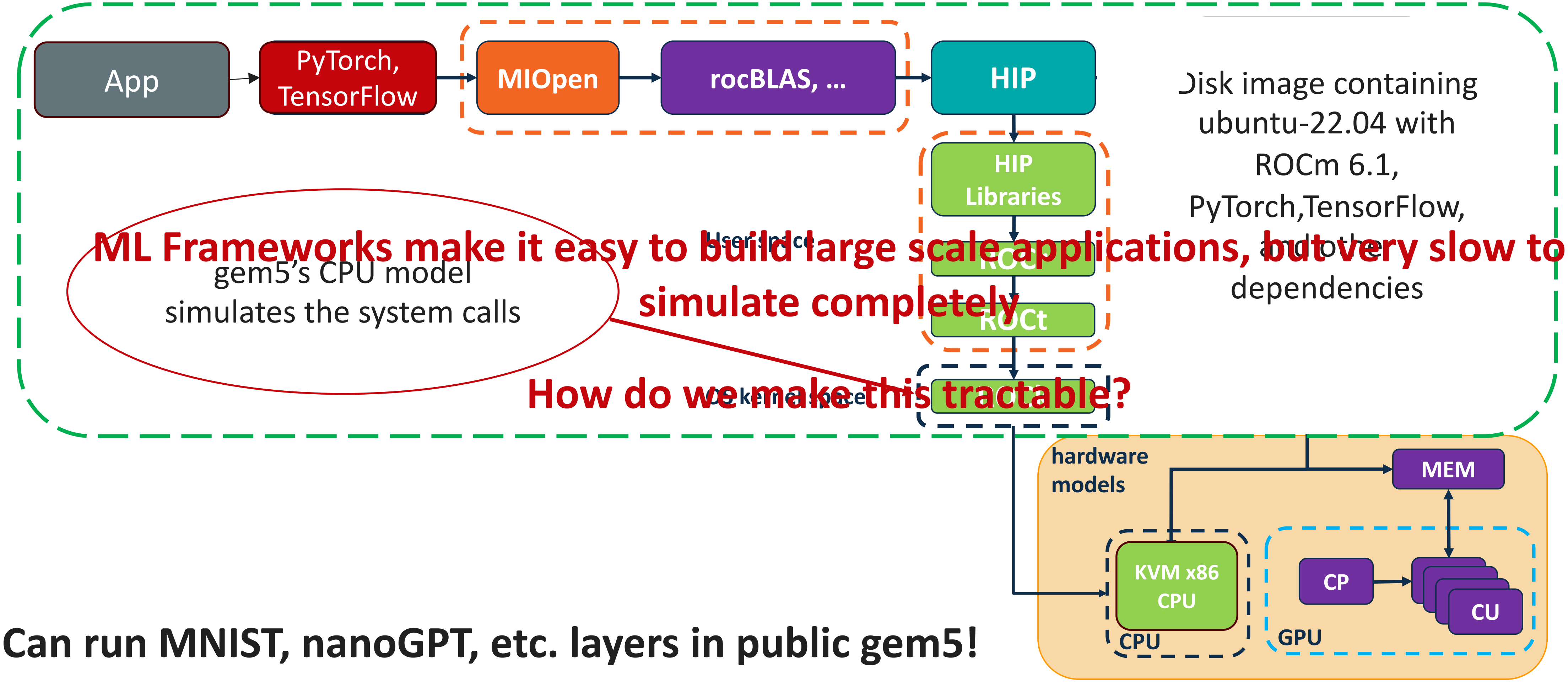
APIs to HIP libraries
SE mode currently doesn’t support frameworks like PyTorch, TensorFlow

We added this support
[Alsop IISWC ‘19], [Roarty gem5 Workshop ‘21]

Outline

- Motivation
- Background
- **Design**
- Conclusion & Future Work

Large-Scale Workloads: Enabling PyTorch/TensorFlow (gem5 GPUFS Mode Support)



ML Frameworks make it easy to build large scale applications, but very slow to simulate completely

How do we make this tractable?

Can run MNIST, nanoGPT, etc. layers in public gem5!

Our Vision to Run Large-Scale Workloads

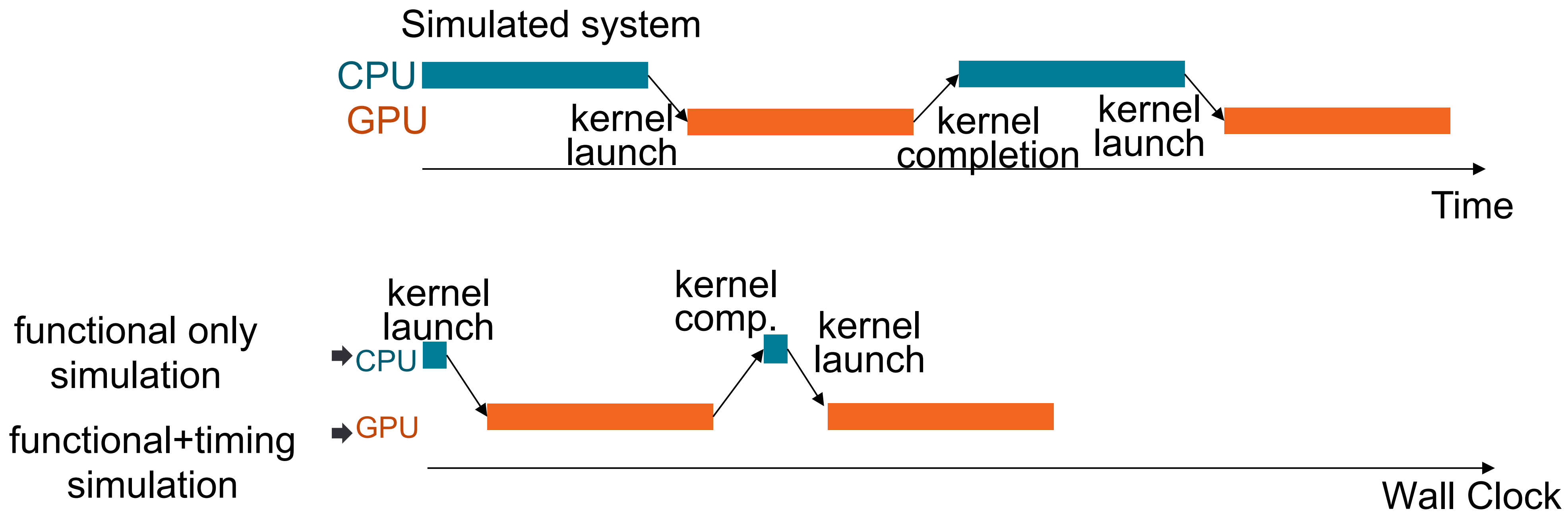
- Not all application phases require high fidelity
  - Some functions/code blocks are “more important” to its behavior

- **Key Insight 1:** Some application regions can be run on lower fidelity models
  - Can leverage KVM CPU support to fast-forward through these regions
  - **Mixed Fidelity Simulation**: only simulate regions of interest with high fidelity models

- **Key Insight 2:** Some application phases simulate same data/code many times
  - Can create checkpoint after less important phases (e.g., file reading)
    - All subsequent simulations restore checkpoint, avoiding repeated simulations
  - Or: simulate more important, but repeated phase once then utilize checkpoint

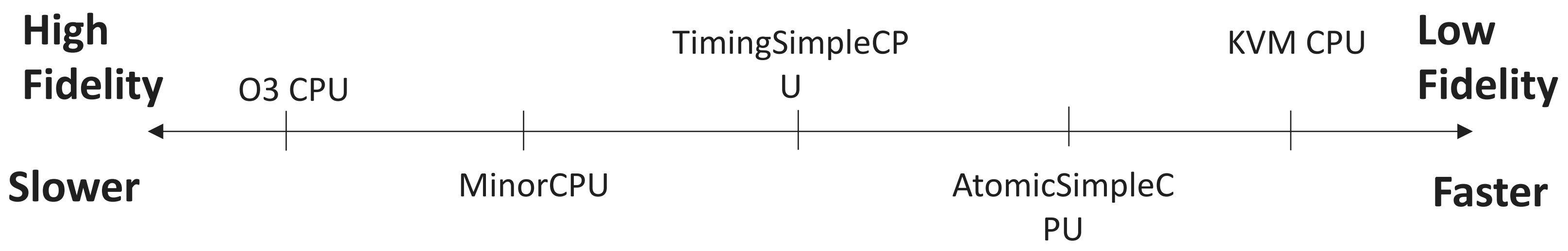
Mixed Fidelity for Less Important Application Phases

- May not want to fully simulate certain phases of applications
- Solution: Utilize host CPU to fast forward through CPU code



Mixed Fidelity for Less Important Application Phases (Cont.)

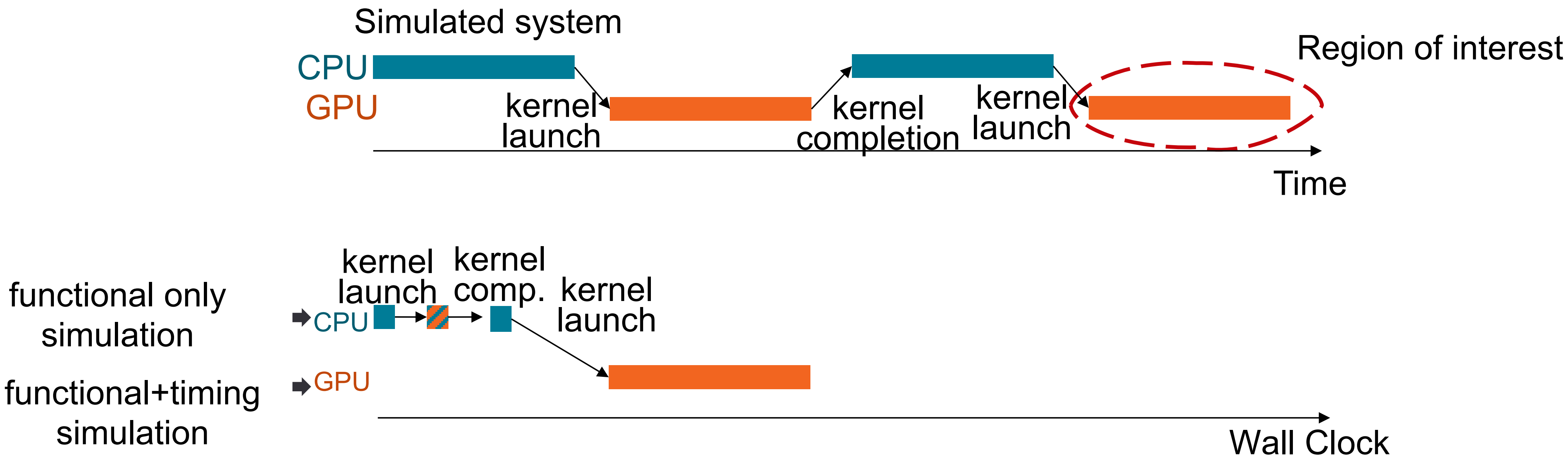
- Observation: GPU model more simulation time intensive than CPU models
- Idea: Leverage PyTorch's/TensorFlow's CPU offloading feature
  - Offload phases (GPU kernels) requiring less fidelity to faster/lower fidelity CPU models



- Reduces simulation runtime without significantly compromising fidelity

Mixed Fidelity for Less Important Application Phases (Cont.)

- Offload GPU Kernels onto CPU and run them at low fidelity



Preliminary Results: Only 1.58x - 3x slower than bare metal

Mixed Fidelity Simulations: How Much Does This Help?

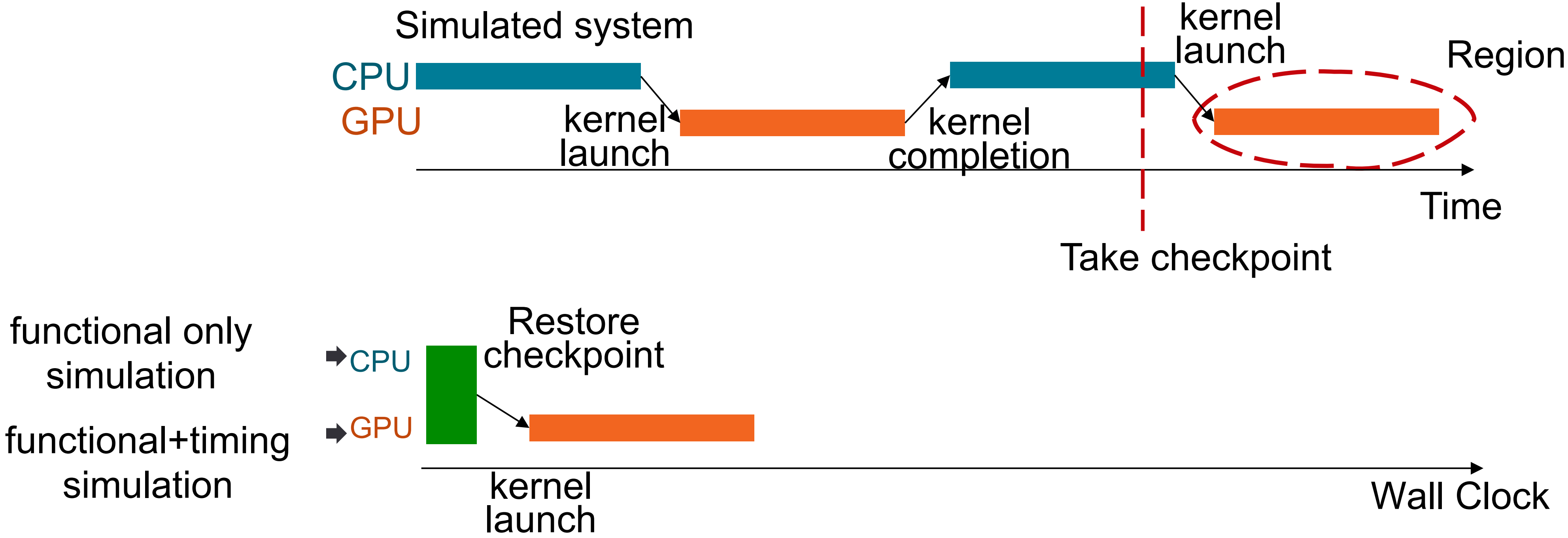
- Cycle Level GPU Simulation : 10-50 KIPS
- Functional KVM Simulation : 100s MIPS
  - KVM CPU emulating GPU : 10s MIPS
- Conservative speedup for a kernel containing 2B SIMD instructions:
  - 11 hours of cycle-level GPU simulation
  - 3 minutes to execute on KVM CPU – single threaded

Mixed Fidelity makes gem5 speeds <u>much</u> closer to real HW

On-going Work: full set of results for GPU workloads

Application Checkpointing: Further Speeding up Simulations

- Need not re-run parts of application when simulating multiple times
- Solution: Checkpoint before region of interest during first run and restore later



Can combine with fast-forwarding to further speed up

We added this support and released it publicly

Application Checkpointing: How Much Does This Help?

- Ran 100 kernel iterations of square (matrix-vector addition GPU program):
  - End-to-end Application runtime in gem5: 1076.33 sec
- Created a checkpoint after 95 kernel iterations:
  - Restoration runtime: 122.94 sec (89 % faster)
- Fidelity Comparison:
  - Compare last five kernels of original application with the five kernels after restoration

Metric	Original Application	Checkpoint Restoration	Difference
Number of GPU Clock Cycles	126336037500	126405051500 Cycles	0.05%
# ALU Instruction	284208	283800	0.14%
# Memory Instructions	39136	39080	0.14%

On-going Work: full set of results for GPU workloads

Outline

- Motivation
- Background
- Design
- **Conclusion & Future Work**

Conclusion

- Simulation tools must evolve to scalably model modern workloads
- gem5 Vision: Swiss army knife that efficiently supports modern frameworks
  - Run CPUs, GPUs, and accelerators; enables cross-layer, early-stage exploration
  - Frameworks for Large-Scale Workload Simulation
  - Mixed Fidelity Simulation
  - Application Checkpointing
- Our work enables previously not possible research
- Next Steps:
  - Integrate accels. into mainline gem5 (e.g., gem5-SALAM [Rogers et al., MICRO '20])
  - Profile ML workloads to annotate regions for reduced fidelity & checkpointing

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON