FireSim

Fast and Effortless FPGA-accelerated Hardware Simulation with On-Prem and Cloud Flexibility

https://fires.im

@firesimproject

Speaker: Abe Gonzalez

Berkeley Architecture Research

The architect/chip-developer’s design flow

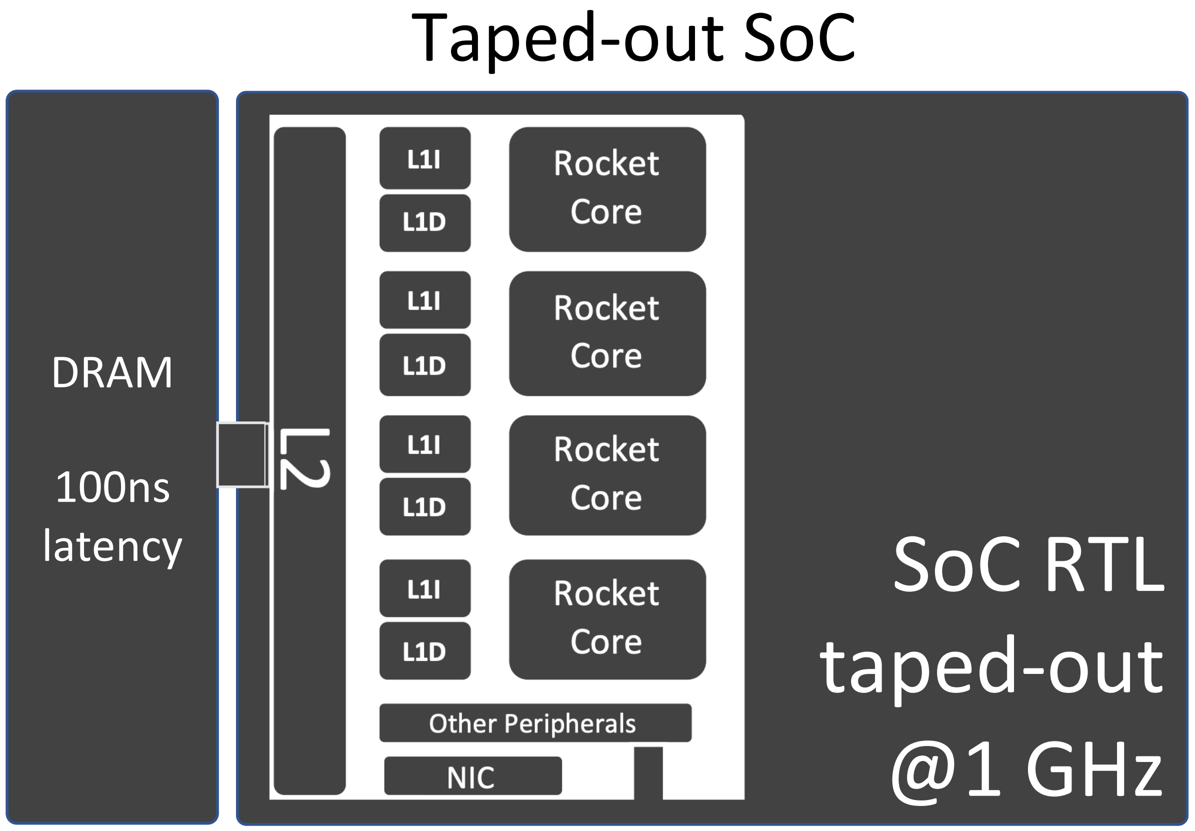
1. High-level Simulation
2. Write RTL + Software, plug into your favorite ecosystem (e.g. Chipyard)
3. Co-design in software RTL sim (e.g. Verilator, VCS, etc.)
   - Run microbenchmarks
4. Co-design in FPGA-accelerated simulation
   - Boot an OS and run the complete software stack, obtain realistic performance measurements
5. Tapeout → Chip
   - Boot OS and run applications, but no more opportunity for co-design

The architect/chip-developer's design flow

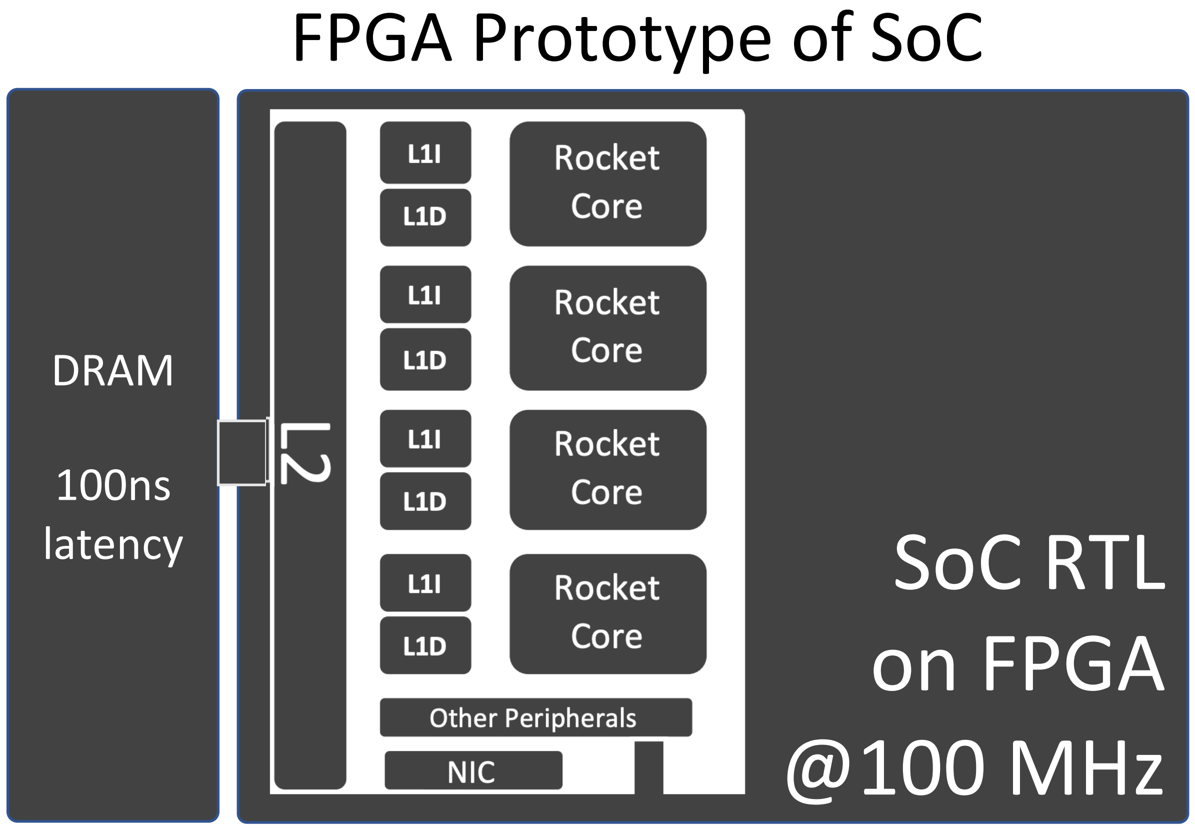
1. High-level Simulation
2. Write RTL + Software, plug into your favorite ecosystem (e.g. Chipyard)
3. Co-design in software RTL sim (e.g. Verilator, VCS, etc.)
   - Run microbenchmarks
4. **Co-design in FPGA-accelerated simulation**
   - **Boot an OS and run the complete software stack, obtain realistic performance measurements**
5. Tapeout → Chip
   - Boot OS and run applications, but no more opportunity for co-design

FireSim

What about FPGA prototyping?



SoC sees 100 cycle DRAM latency



SoC sees 10 cycle DRAM latency

Incorrect by a factor of 10!

Difficulties with FPGA Prototypes

In an FPGA prototype:

- Every FPGA clock executes one cycle of the simulated target
- Performance of FPGA-attached resources is exposed to the simulated world, e.g. DRAM, SD Card, UART, Ethernet, etc.

This leads to three problems:

1) Incorrect performance modeling: FPGA resources probably not an accurate representation of target system
   a) E.g., DRAM performance off by ***10x*** on previous slide
2) Simulations are non-deterministic
3) Different host FPGAs produce different simulation results

Want HW simulators that:

- Are as fast as silicon
- Are as detailed as silicon
- Have all the benefits of SW-based simulators
- Are low-cost

Our Thesis:

- FPGAs are the only viable basis technology

→ Build *FPGA-accelerated* simulators with SW-like flexibility using an *open-source* tool

How? Useful Trends Throughout the Stack

Open ISA

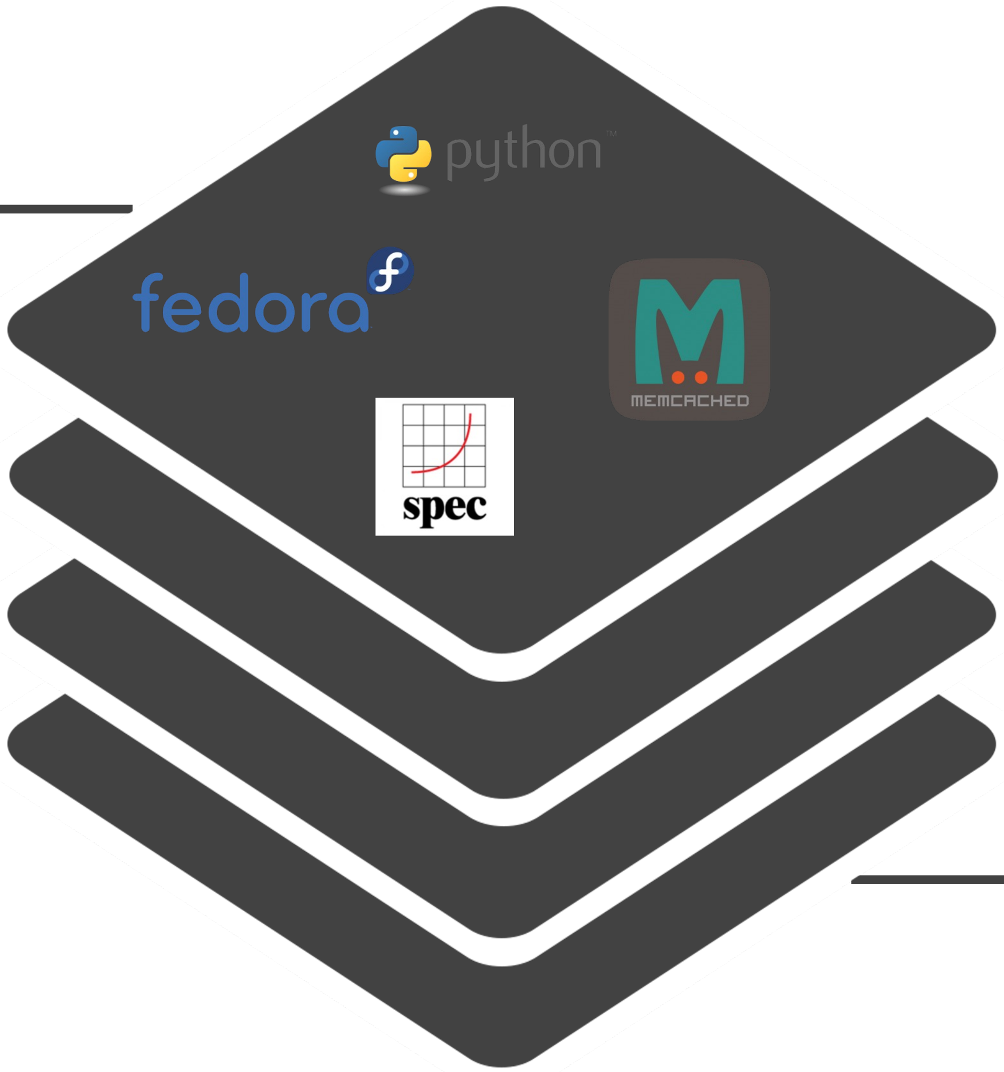
RISC-V

CHISEL

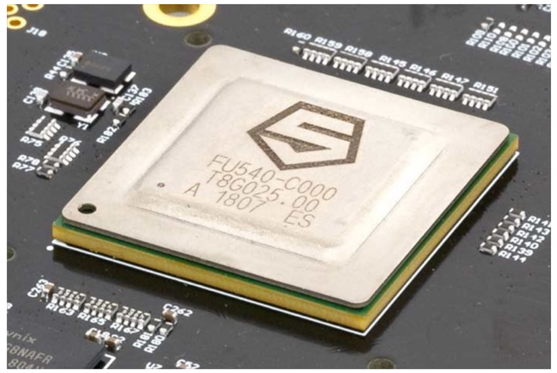
High-Productivity Hardware Design Language & IR

FIRRTL

Berkeley Architecture Research



Open, Silicon-Proven SoC Implementations



FPGAs in the Cloud

Amazon EC2 F1 Instances

Run Customizable FPGAs in the AWS Cloud

+ On Premise FPGAs

FireSim at 35,000 feet

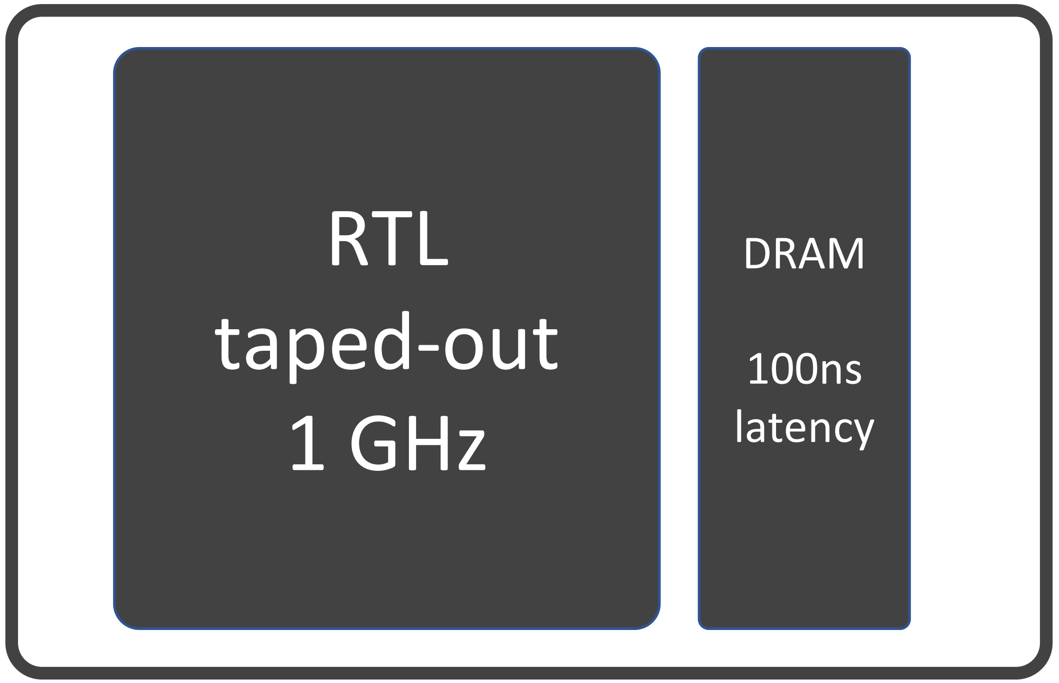
- Open-source, fast, automatic, deterministic FPGA-accelerated hardware simulation for pre-silicon verification and performance validation
- Ingests:
  - Your RTL design: FIRRTL (Chisel), blackbox Verilog
    - Or Chipyard-generated designs with Rocket Chip, BOOM, NVDLA, PicoRV32, and more
  - HW and/or SW IO models (e.g. UART, Ethernet, DRAM, etc.)
  - Workload descriptions
- Produces: Fast, cycle-exact simulation of your design + models around it
- Automatically deployed to on-prem or cloud FPGAs
  - E.g., Xilinx Alveo or AWS EC2 F1

Three Distinguishing Features of FireSim

1) Not FPGA prototypes, rather FPGA-accelerated simulators
   - Automatic transformation of RTL designs into FPGA-accelerated simulators
   - Enables new debugging, resource optimization, and profiling capabilities
2) Flexible scaling from on-prem to cloud FPGAs
   - Scale easily from one or more on-prem FPGAs to massively parallel simulations on elastic supply of cloud FPGAs
   - Standardized host platforms = easy to collaborate with other researchers and perform artifact evaluation
   - Heavy automation to hide FPGA complexity, regardless of on-prem or cloud platform
3) Open-source (https://fires.im)

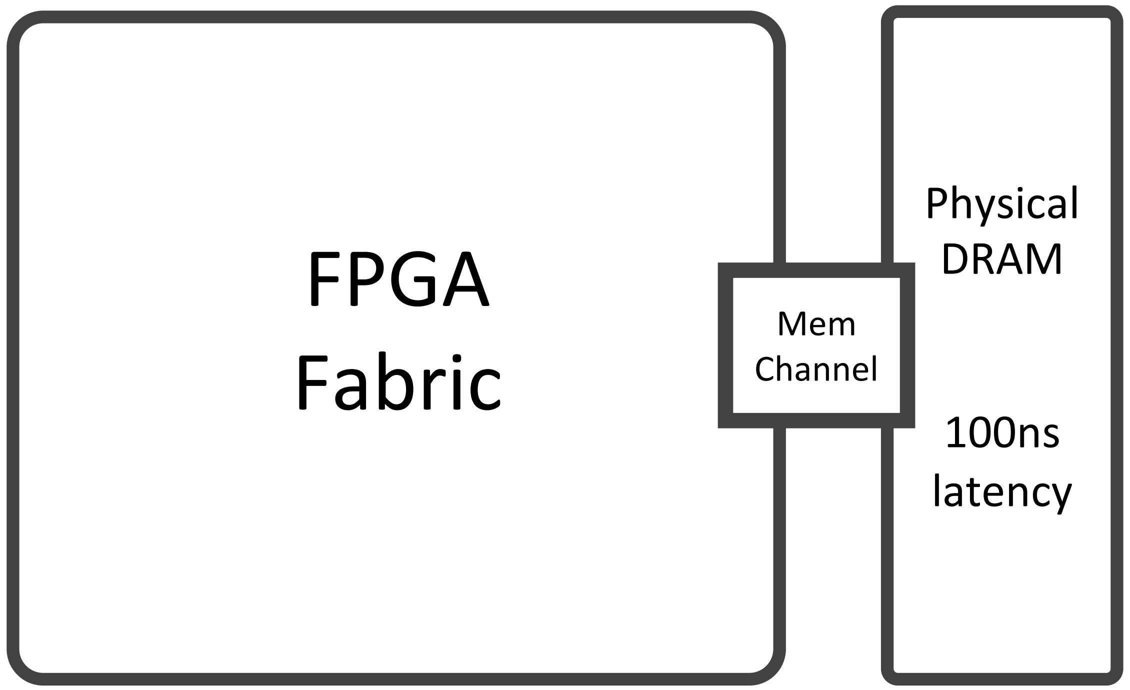
Separating Target and Host

Target: the machine under simulation



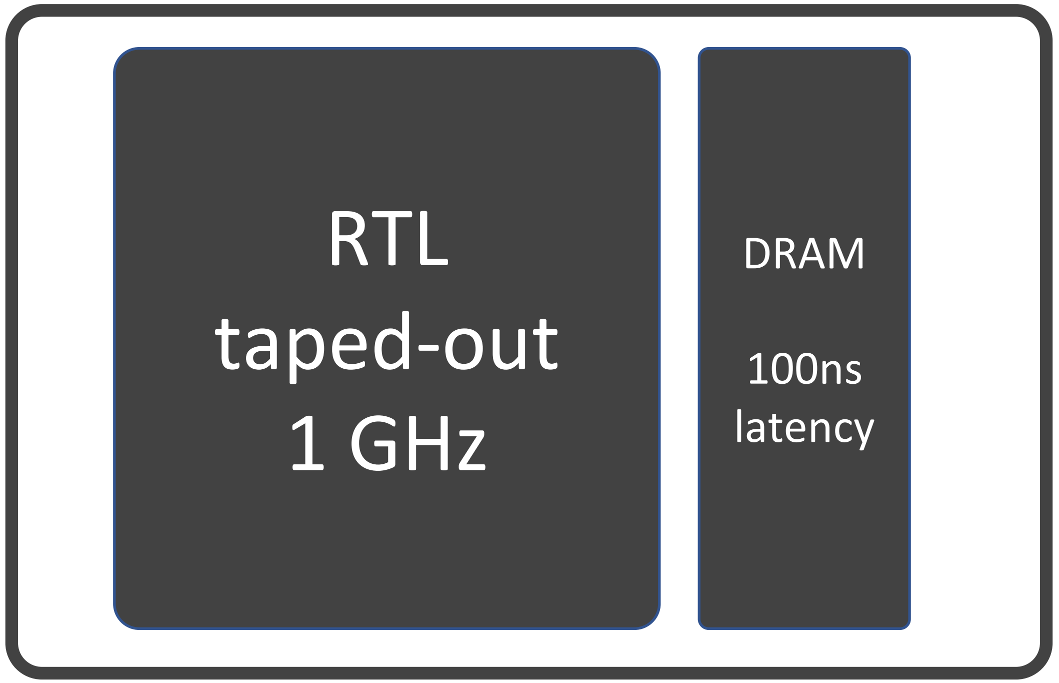
Closed simulation world.

Host: the machine executing (*hosting)* the simulation



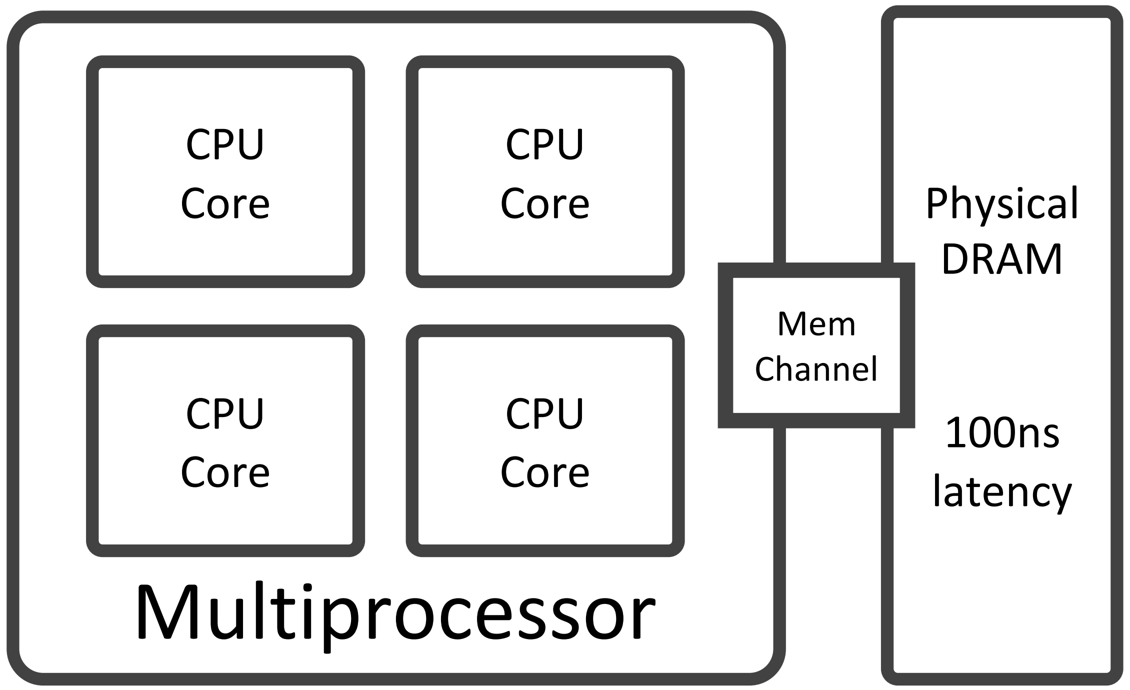
Separating Target and Host

Target: the machine under simulation

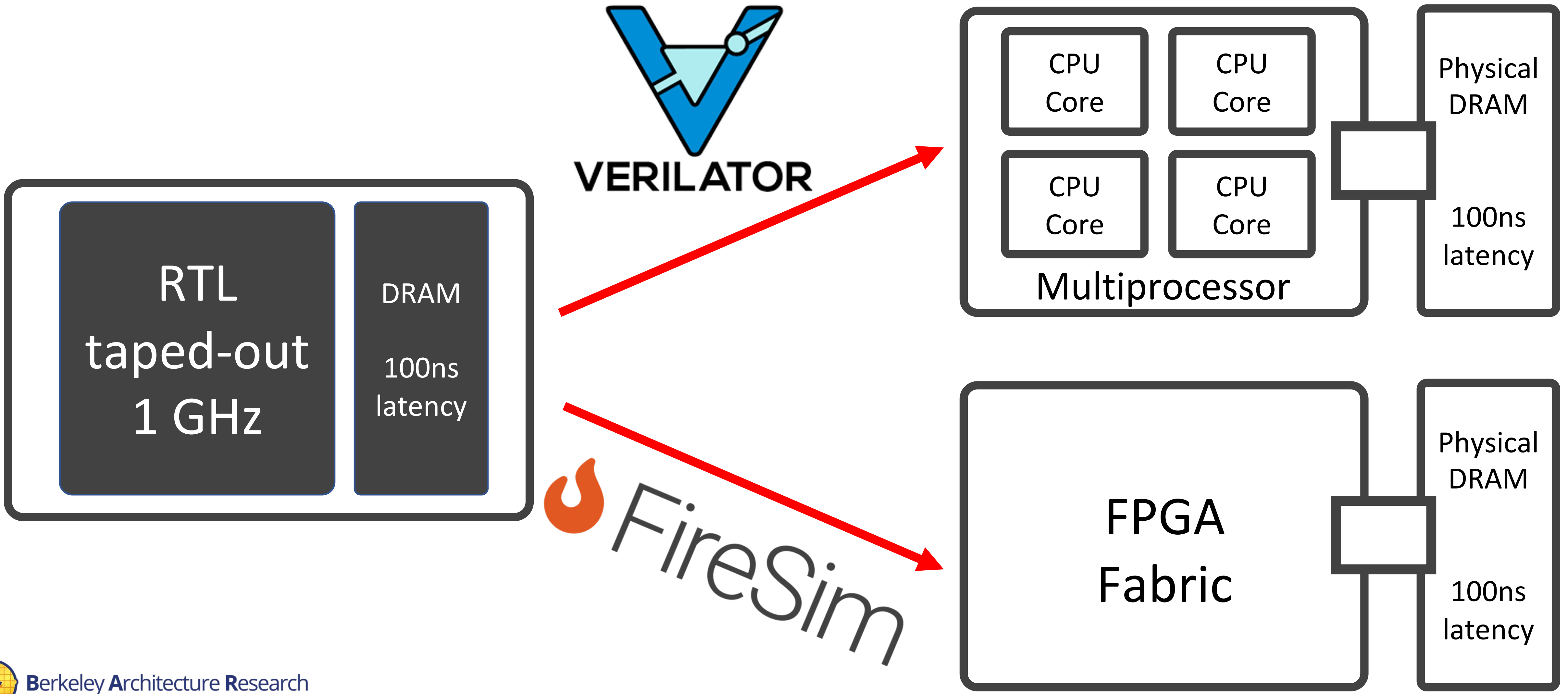


Closed simulation world.

Host: the machine executing (*hosting)* the simulation

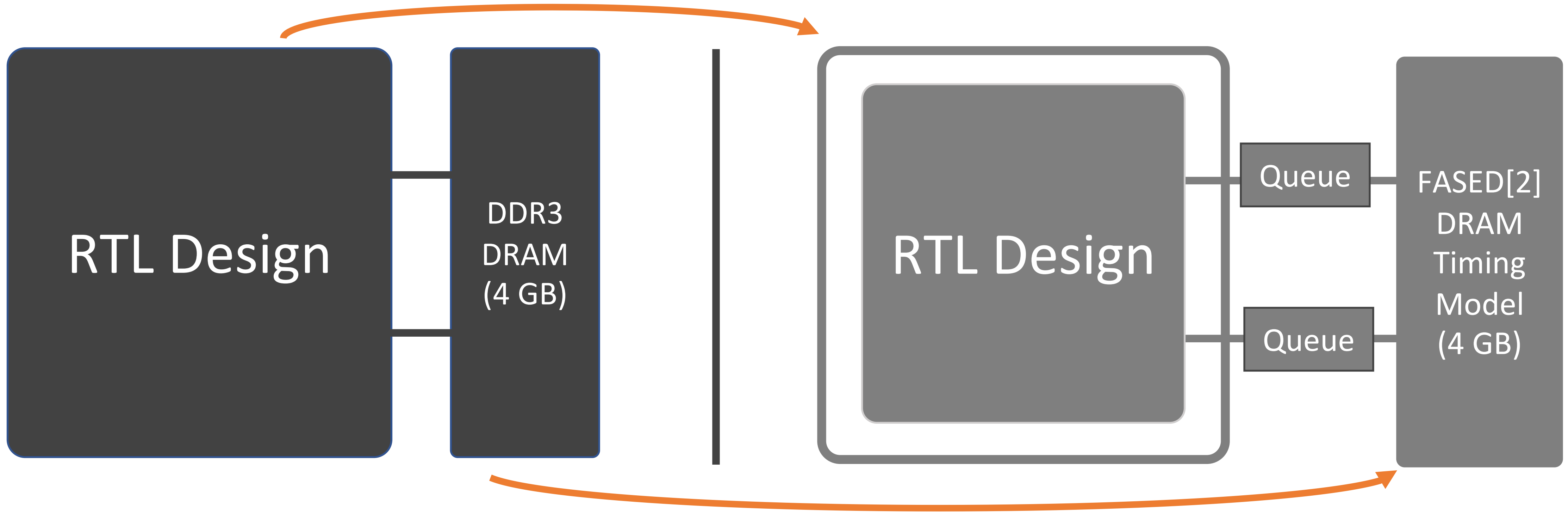


FireSim Generates FPGA-Hosted Simulators



Host Decoupling in FireSim: Transforming the Target

1) Convert RTL into a latency-insensitive [1] model using FIRRTL transform



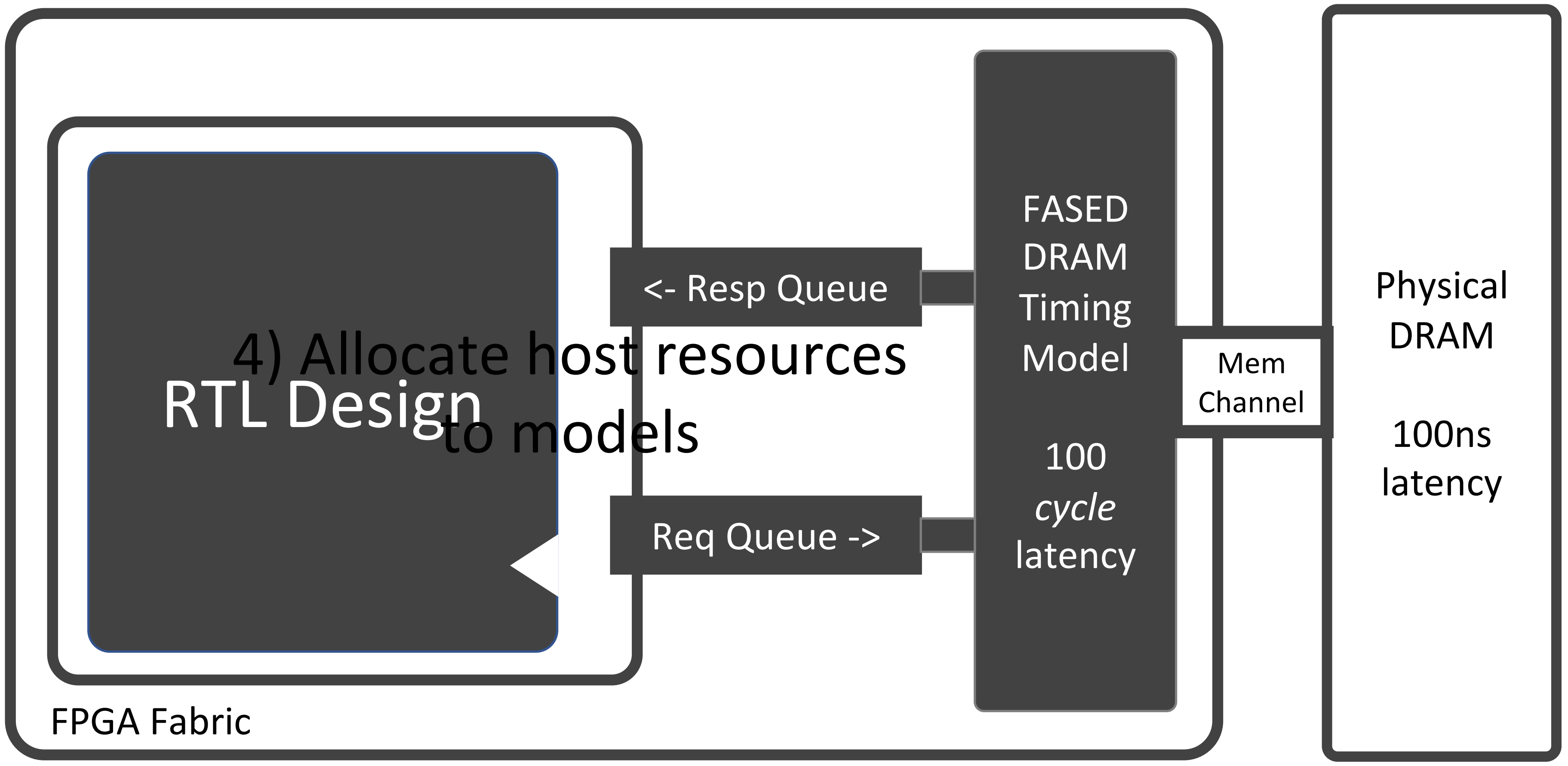
2) Generate FPGA-hosted model for DRAM [2] (think DRAMSim on an FPGA)

3) Generate queues (token channels) to connect the target models

[1] Theory of Latency Insensitive Design, Carloni et al, also see: RAMP

[2] FASED: FPGA-accelerated Simulation and Evaluation of DRAM, Biancolin et al

Host Decoupling in FireSim: Mapping to the FPGA



SoC sees realistic DRAM latency

Benefits of Host Decoupling on FPGAs

Simulations will now:

- Execute deterministically
- Produce identical results on different hosts (FPGAs & CPUs)

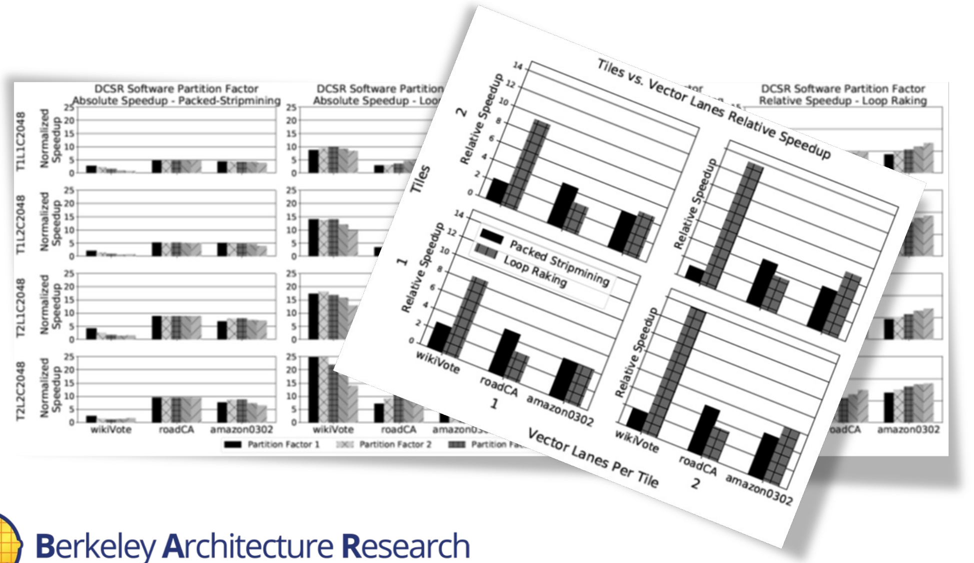
Decoupling enables support for:

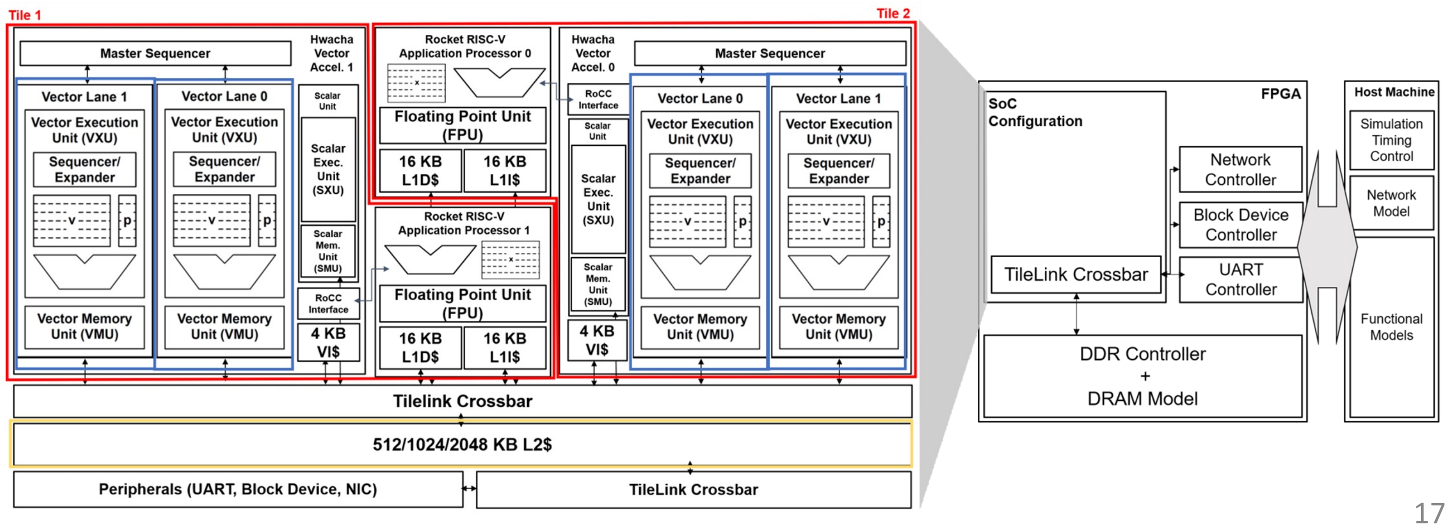
1. SW co-simulation (e.g. block device, network models)
2. Simulating large targets over distributed hosts (ISCA ‘18, Top Picks ‘18)
3. Non-invasive debugging and instrumentation (FPL ‘18, ASPLOS ’20, ASPLOS ‘23)
4. Multi-cycle resource optimizations (ICCAD ‘19)

What Can You Do With FireSim?

Example use cases: Evaluating SoC Designs

- "Classical" Performance Measurement
  - Run SPECint 2017 with full reference inputs on Rocket Chip in parallel on ~10 FPGAs within a day (e.g., in D. Biancolin, et. al., *FASED*, FPGA '19)
- Rapid Full-System Design Space Exploration
  - Can rapidly sweep parameter space of a design with FireSim automation
  - Data-parallel accelerators (Hwacha) and multi-core processors
  - Complex software stacks (Linux, OpenMP, GraphMat, Caffe)





Example use cases: Evaluating SoC Designs

- Security:
  - BOOM Spectre replication
    - A. Gonzalez, et. al., *Replicating and Mitigating Spectre Attacks on an Open Source RISC-V Microarchitecture,* CARRV ’19
  - Keystone Enclave performance evaluation
    - D. Lee, et. al., *Keystone*, EuroSys ’20
  - Mitigating cache attacks
    - G. Saileshwar, et. al., *MIRAGE: Mitigating Cache Attacks with a Randomized Fully-Associative Cache,* USENIX Security ’21
- Accelerator evaluation
  - Accelerator Orchestration:
    - Machine Learning (S. Kim, et. al., *AuRORA*, MICRO 2023)
  - Chisel-based accelerators:
    - Machine learning (H. Genc, et. al., *Gemmini*, DAC 2021)
    - Garbage collection (M. Maas, et. al., *A Hardware Accelerator for Tracing Garbage Collection,* ISCA ’18)
  - Integrating Verilog-based accelerators:
    - NVDLA (F. Farshchi, et. al. *Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim*. EMC2 ‘19)
    - HLS-based rapid prototyping (Q. Huang, et. al., *Centrifuge*, ICCAD ‘19)
  - Scale-out accelerators
    - nanoPU NIC-CPU co-design (S. Ibanez, et. al., *nanoPU*, OSDI ‘21)
    - Protobuf Accelerator (S. Karandikar, et. al., *A Hardware Accelerator for Protocol Buffers*, MICRO ’21. MICRO-54 Distinguished Artifact Winner.)
    - Compression Accelerators (S. Karandikar, et. al., *CDPU: Co-designing Compression and Decompression Processing Units for Hyperscale Systems*, ISCA ’23.)

Replicating Spectre-v1/2

- Use
  an
- Rep

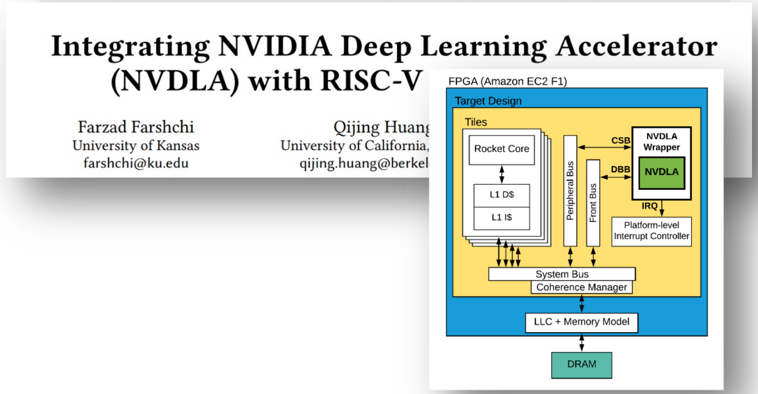
BOOM Hardware Security Research

```
Script started, file is uartlog
AFI PCI  Vendor ID: 0x1d0f, Device ID 0xf000
Using edma queue: /dev/edma0_queue_0
UART0 is here (stdin/stdout).
random min: 0x0, random max: 0xffffffffffffffff
Commencing simulation.

Print memory of _start
----------------------
Address    | 32b Data
----------------------
0x80000174: ef010113 --> DASM(ef010113)
0x80000178: 00113423 --> DASM(00113423)
0x8000017c: 00213823 --> DASM(00213823)
0x80000180: 00313c23 --> DASM(00313c23)
0x80000184: 02413023 --> DASM(02413023)
0x80000188: 02513423 --> DASM(02513423)
0x8000018c: 02613823 --> DASM(02613823)
0x80000190: 02713c23 --> DASM(02713c23)
0x80000194: 04813023 --> DASM(04813023)
0x80000198: 04913423 --> DASM(04913423)
0x8000019c: 04a13823 --> DASM(04a13823)
0x800001a0: 04b13c23 --> DASM(04b13c23)
0x800001a4: 06c13023 --> DASM(06c13023)
0x800001a8: 06d13423 --> DASM(06d13423)
0x800001ac: 06e13823 --> DASM(06e13823)
0x800001b0: 06f13c23 --> DASM(06f13c23)
0x800001b4: 09013023 --> DASM(09013023)
0x800001b8: 09113423 --> DASM(09113423)
0x800001bc: 09213823 --> DASM(09213823)
0x800001c0: 09313c23 --> DASM(09313c23)
0x800001c4: 0b413023 --> DASM(0b413023)
0x800001c8: 0b513423 --> DASM(0b513423)
0x800001cc: 0b613823 --> DASM(0b613823)
0x800001d0: 0b713c23 --> DASM(0b713c23)
0x800001d4: 0d813023 --> DASM(0d813023)
0x800001d8: 0d913423 --> DASM(0d913423)
0x800001dc: 0da13823 --> DASM(0da13823)
0x800001e0: 0db13c23 --> DASM(0db13c23)
0x800001e4: 0fc13023 --> DASM(0fc13023)
0x800001e8: 0fd13423 --> DASM(0fd13423)
0x800001ec: 0fe13823 --> DASM(0fe13823)
0x800001f0: 0ff13c23 --> DASM(0ff13c23)
0x800001f4: 34202573 --> DASM(34202573)
0x800001f8: 341025f3 --> DASM(341025f3)
0x800001fc: 00010613 --> DASM(00010613)
0x80000200: 238020ef --> DASM(238020ef)
0x80000204: 34151073 --> DASM(34151073)
0x80000208: 000022b7 --> DASM(000022b7)
0x8000020c: 8002829b --> DASM(8002829b)
 0  0 > centos@ip-192-168-2-15:~
```

```
FireSim Simulation Status @ 2019-01-11 19:26:39.178673
--------------------------------------------------------------------------------
This workload's output is located in:
/home/centos/firesim/deploy/results-workload/2019-01-11--19-26-24-adept-retreat-demo/
This run's log is located in:
/home/centos/firesim/deploy/logs/2019-01-11--19-26-24-runworkload-UG0ZM7OY9W83RENG.log
This status will update every 10s.
--------------------------------------------------------------------------------
Instances
--------------------------------------------------------------------------------
Instance IP:   192.168.2.15 | Terminated: False
--------------------------------------------------------------------------------
Simulated Switches
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Simulated Nodes/Jobs
--------------------------------------------------------------------------------
Instance IP:   192.168.2.15 | Job: adept-retreat-demo0 | Sim running: True
--------------------------------------------------------------------------------
Summary
--------------------------------------------------------------------------------
1/1 instances are still running.
1/1 simulations are still running.
--------------------------------------------------------------------------------
                                      2019-01-11 < 19:26  ip-192-168-2-9
```

17



Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V

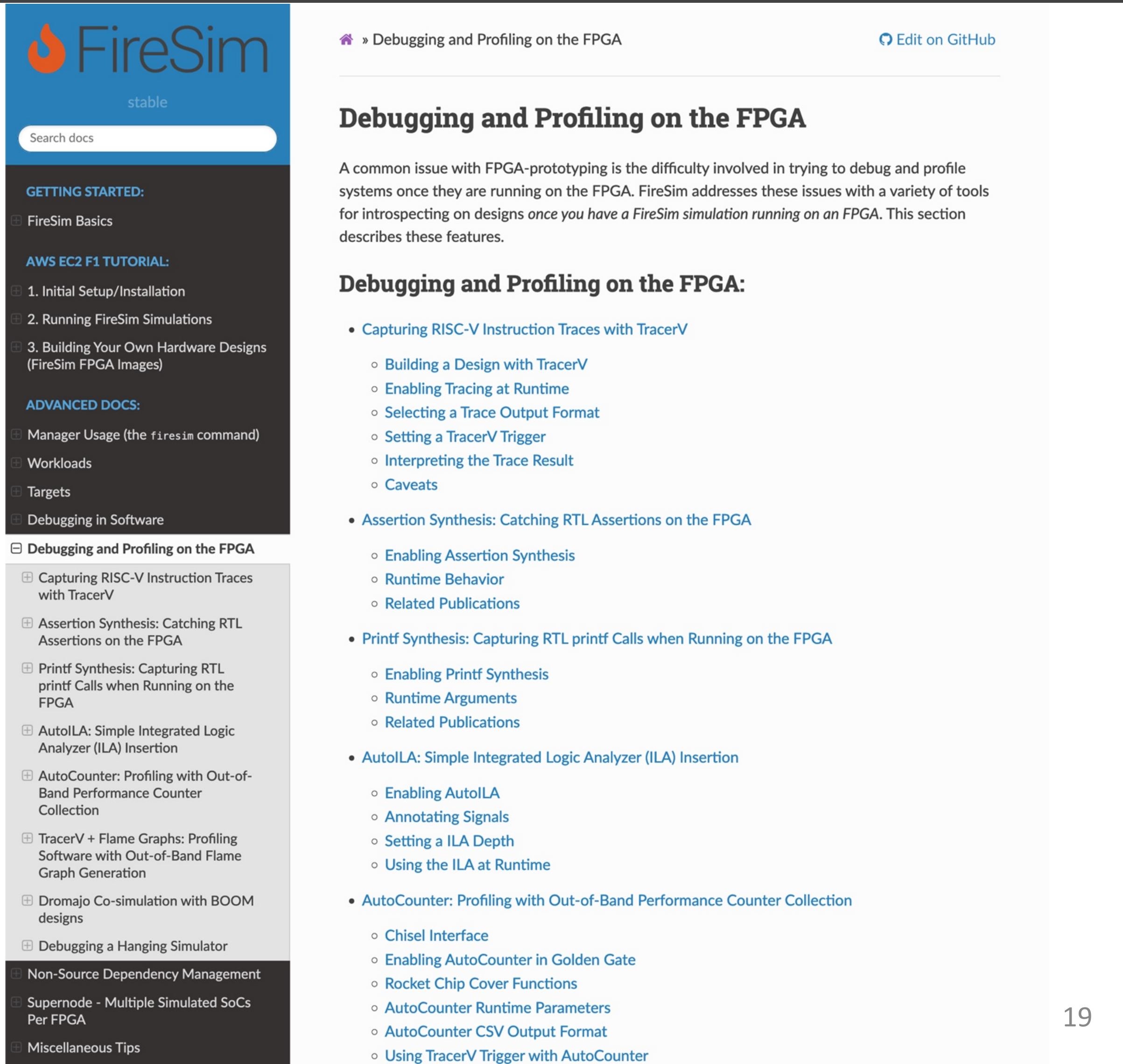
Farzad Farshchi
University of Kansas
farshchi@ku.edu

Qijing Huang
University of California,
qijing.huang@berkel

Berkeley Architecture Research

Example use cases: Debugging and Profiling SoC Designs

- Debugging a Chisel design at FPGA-speeds
  - Many FireSim debugging features: Assertion synthesis, printf synthesis, ILA insertion, etc.
  - e.g. FireSim Debugging Docs



FireSim
stable

Search docs

GETTING STARTED:

- FireSim Basics

AWS EC2 F1 TUTORIAL:

- 1. Initial Setup/Installation
- 2. Running FireSim Simulations
- 3. Building Your Own Hardware Designs (FireSim FPGA Images)

ADVANCED DOCS:

- Manager Usage (the `firesim` command)
- Workloads
- Targets
- Debugging in Software
- Debugging and Profiling on the FPGA
  - Capturing RISC-V Instruction Traces with TracerV
  - Assertion Synthesis: Catching RTL Assertions on the FPGA
  - Printf Synthesis: Capturing RTL printf Calls when Running on the FPGA
  - AutoILA: Simple Integrated Logic Analyzer (ILA) Insertion
  - AutoCounter: Profiling with Out-of-Band Performance Counter Collection
  - TracerV + Flame Graphs: Profiling Software with Out-of-Band Flame Graph Generation
  - Dromajo Co-simulation with BOOM designs
  - Debugging a Hanging Simulator
- Non-Source Dependency Management
- Supernode - Multiple Simulated SoCs Per FPGA
- Miscellaneous Tips

» Debugging and Profiling on the FPGA   Edit on GitHub

Debugging and Profiling on the FPGA

A common issue with FPGA-prototyping is the difficulty involved in trying to debug and profile systems once they are running on the FPGA. FireSim addresses these issues with a variety of tools for introspecting on designs *once you have a FireSim simulation running on an FPGA*. This section describes these features.

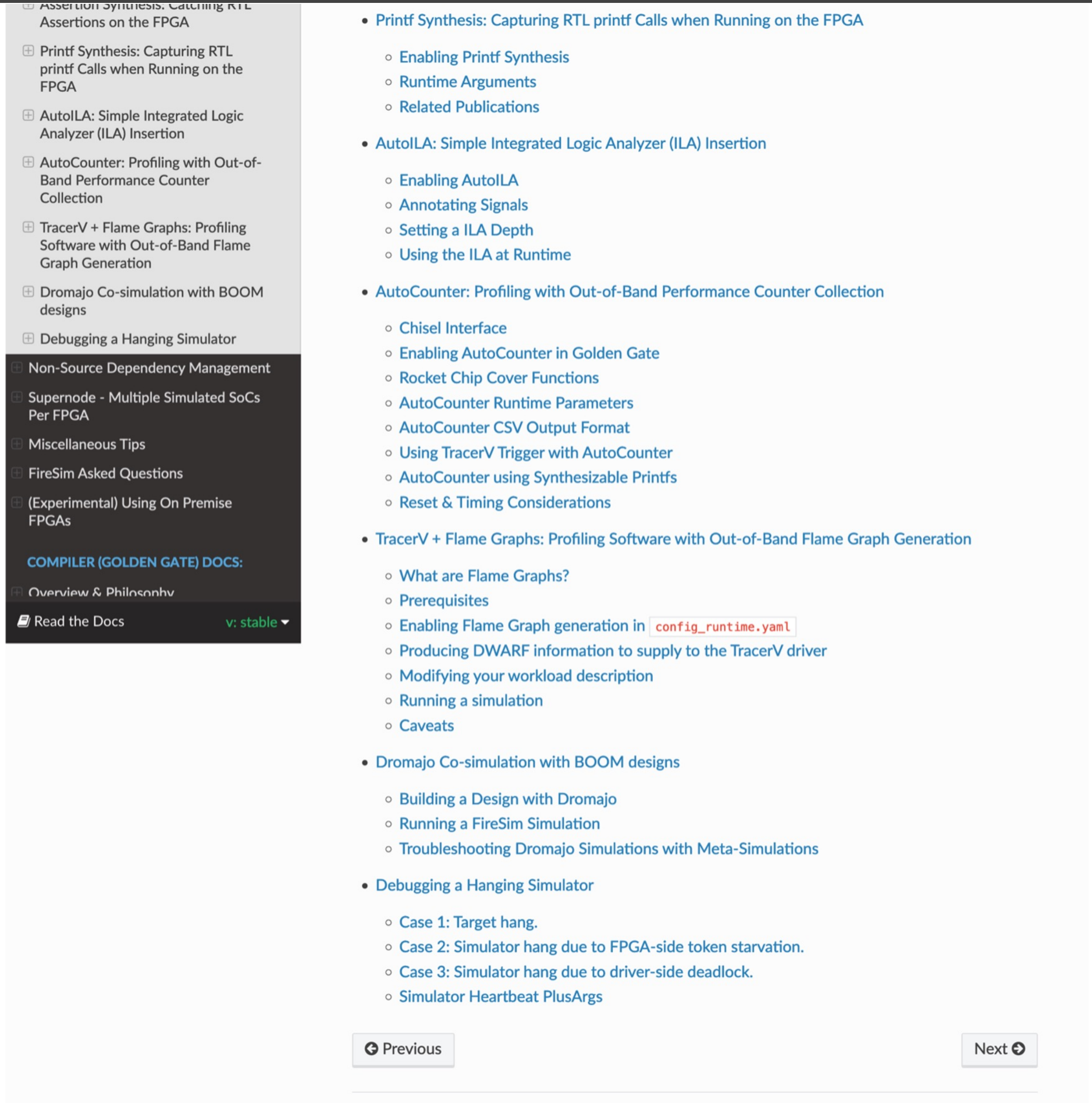
Debugging and Profiling on the FPGA:

- Capturing RISC-V Instruction Traces with TracerV
  - Building a Design with TracerV
  - Enabling Tracing at Runtime
  - Selecting a Trace Output Format
  - Setting a TracerV Trigger
  - Interpreting the Trace Result
  - Caveats
- Assertion Synthesis: Catching RTL Assertions on the FPGA
  - Enabling Assertion Synthesis
  - Runtime Behavior
  - Related Publications
- Printf Synthesis: Capturing RTL printf Calls when Running on the FPGA
  - Enabling Printf Synthesis
  - Runtime Arguments
  - Related Publications
- AutoILA: Simple Integrated Logic Analyzer (ILA) Insertion
  - Enabling AutoILA
  - Annotating Signals
  - Setting a ILA Depth
  - Using the ILA at Runtime
- AutoCounter: Profiling with Out-of-Band Performance Counter Collection
  - Chisel Interface
  - Enabling AutoCounter in Golden Gate
  - Rocket Chip Cover Functions
  - AutoCounter Runtime Parameters
  - AutoCounter CSV Output Format
  - Using TracerV Trigger with AutoCounter

Berkeley Architecture Research

Example use cases: Debugging and Profiling SoC Designs

- Debugging a Chisel design at FPGA-speeds
  - Many FireSim debugging features: Assertion synthesis, printf synthesis, ILA insertion, etc.
  - e.g. FireSim Debugging Docs



- Assertion Synthesis: Catching RTL Assertions on the FPGA
- Printf Synthesis: Capturing RTL printf Calls when Running on the FPGA
- AutoILA: Simple Integrated Logic Analyzer (ILA) Insertion
- AutoCounter: Profiling with Out-of-Band Performance Counter Collection
- TracerV + Flame Graphs: Profiling Software with Out-of-Band Flame Graph Generation
- Dromajo Co-simulation with BOOM designs
- Debugging a Hanging Simulator
- Non-Source Dependency Management
- Supernode - Multiple Simulated SoCs Per FPGA
- Miscellaneous Tips
- FireSim Asked Questions
- (Experimental) Using On Premise FPGAs

COMPILER (GOLDEN GATE) DOCS:

- Overview & Philosophy

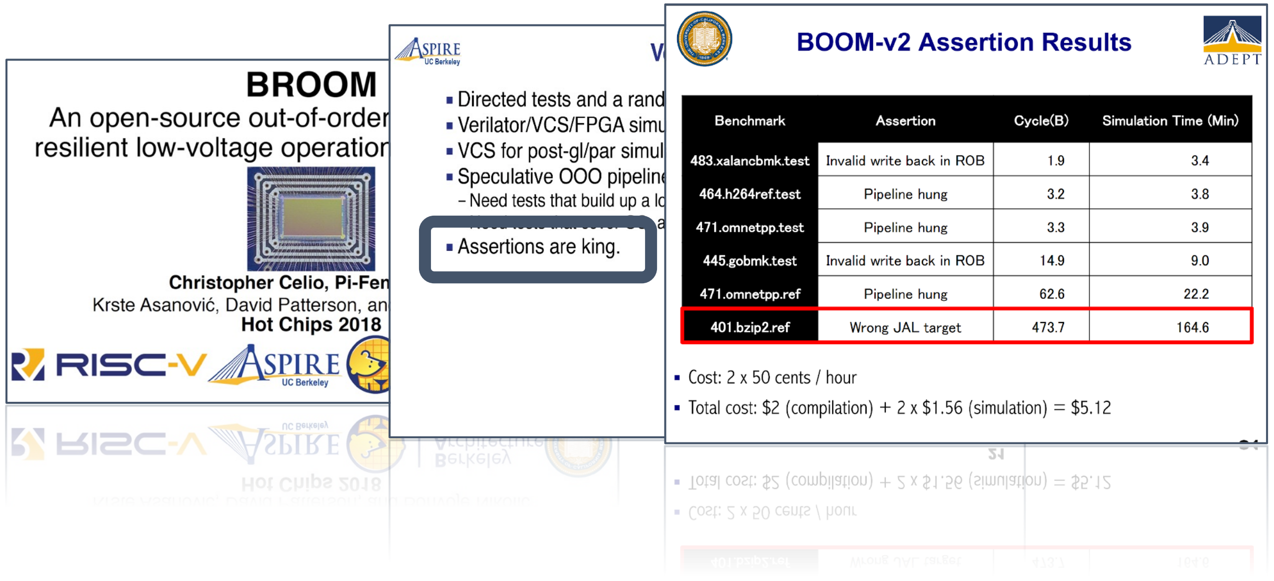
Read the Docs v: stable

- Printf Synthesis: Capturing RTL printf Calls when Running on the FPGA
  - Enabling Printf Synthesis
  - Runtime Arguments
  - Related Publications
- AutoILA: Simple Integrated Logic Analyzer (ILA) Insertion
  - Enabling AutoILA
  - Annotating Signals
  - Setting a ILA Depth
  - Using the ILA at Runtime
- AutoCounter: Profiling with Out-of-Band Performance Counter Collection
  - Chisel Interface
  - Enabling AutoCounter in Golden Gate
  - Rocket Chip Cover Functions
  - AutoCounter Runtime Parameters
  - AutoCounter CSV Output Format
  - Using TracerV Trigger with AutoCounter
  - AutoCounter using Synthesizable Printfs
  - Reset & Timing Considerations
- TracerV + Flame Graphs: Profiling Software with Out-of-Band Flame Graph Generation
  - What are Flame Graphs?
  - Prerequisites
  - Enabling Flame Graph generation in `config_runtime.yaml`
  - Producing DWARF information to supply to the TracerV driver
  - Modifying your workload description
  - Running a simulation
  - Caveats
- Dromajo Co-simulation with BOOM designs
  - Building a Design with Dromajo
  - Running a FireSim Simulation
  - Troubleshooting Dromajo Simulations with Meta-Simulations
- Debugging a Hanging Simulator
  - Case 1: Target hang.
  - Case 2: Simulator hang due to FPGA-side token starvation.
  - Case 3: Simulator hang due to driver-side deadlock.
  - Simulator Heartbeat PlusArgs

Berkeley Architecture Research

Example use cases: Debugging and Profiling SoC Designs

- Debugging a Chisel design at FPGA-speeds
  - Many FireSim debugging features: Assertion synthesis, printf synthesis, ILA insertion, Cosimulation, etc.
  - e.g. FireSim Debugging Docs
  - e.g. Fixing BOOM Bugs (D. Kim, et. al., *DESSERT*, FPL ’18)
- Profiling a custom RISC-V SoC at FPGA-speeds
  - e.g. HW/SW Co-design of a networked RISC-V system (S. Karandikar, et. al., *FirePerf*, ASPLOS 2020)



BROOM
An open-source out-of-order
resilient low-voltage operation

Christopher Celio, Pi-Fen
Krste Asanović, David Patterson, an
Hot Chips 2018

- Directed tests and a rand
- Verilator/VCS/FPGA simu
- VCS for post-gl/par simul
- Speculative OOO pipeline
  - Need tests that build up a lo
- Assertions are king.

BOOM-v2 Assertion Results

Benchmark	Assertion	Cycle(B)	Simulation Time (Min)
483.xalancbmk.test	Invalid write back in ROB	1.9	3.4
464.h264ref.test	Pipeline hung	3.2	3.8
471.omnetpp.test	Pipeline hung	3.3	3.9
445.gobmk.test	Invalid write back in ROB	14.9	9.0
471.omnetpp.ref	Pipeline hung	62.6	22.2
401.bzip2.ref	Wrong JAL target	473.7	164.6

- Cost: 2 x 50 cents / hour
- Total cost: $2 (compilation) + 2 x $1.56 (simulation) = $5.12

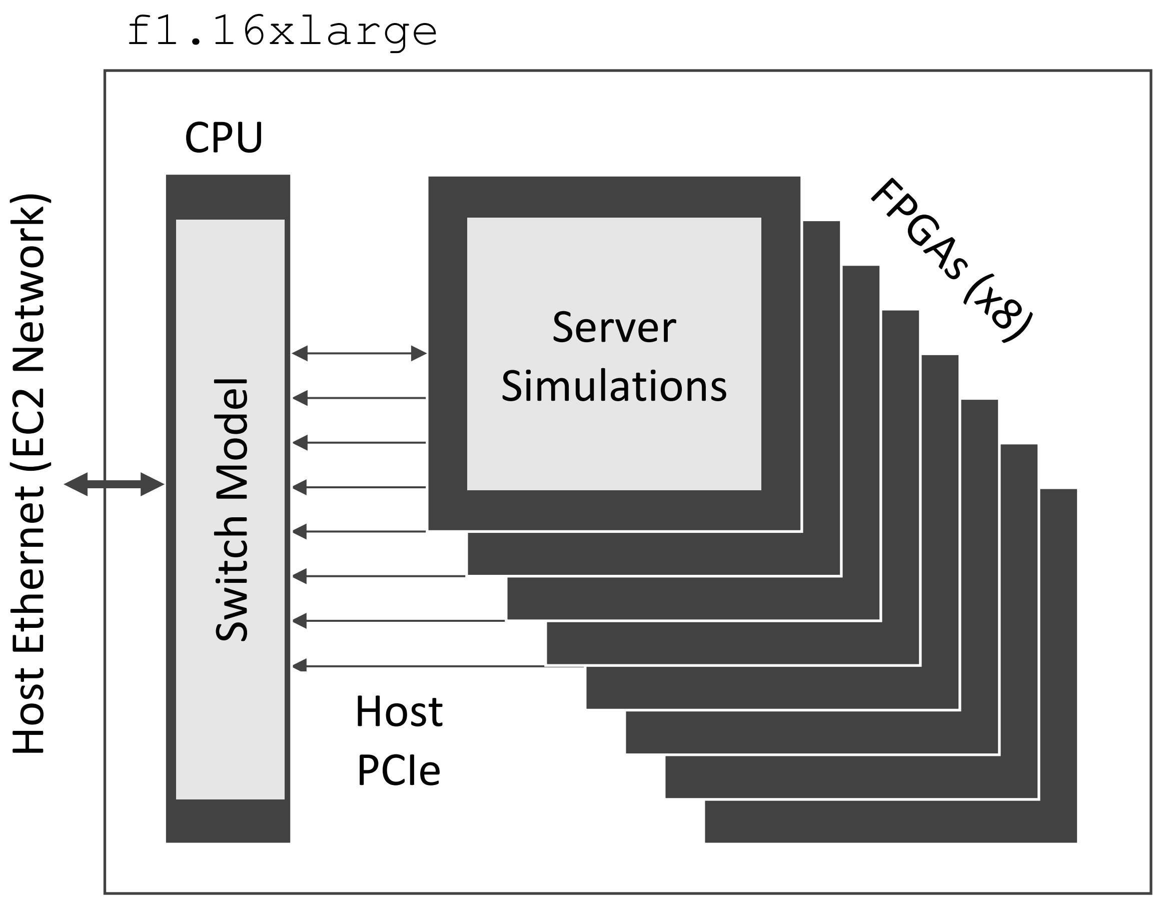
Berkeley Architecture Research

How-to-build a ***datacenter-scale*** FireSim simulation

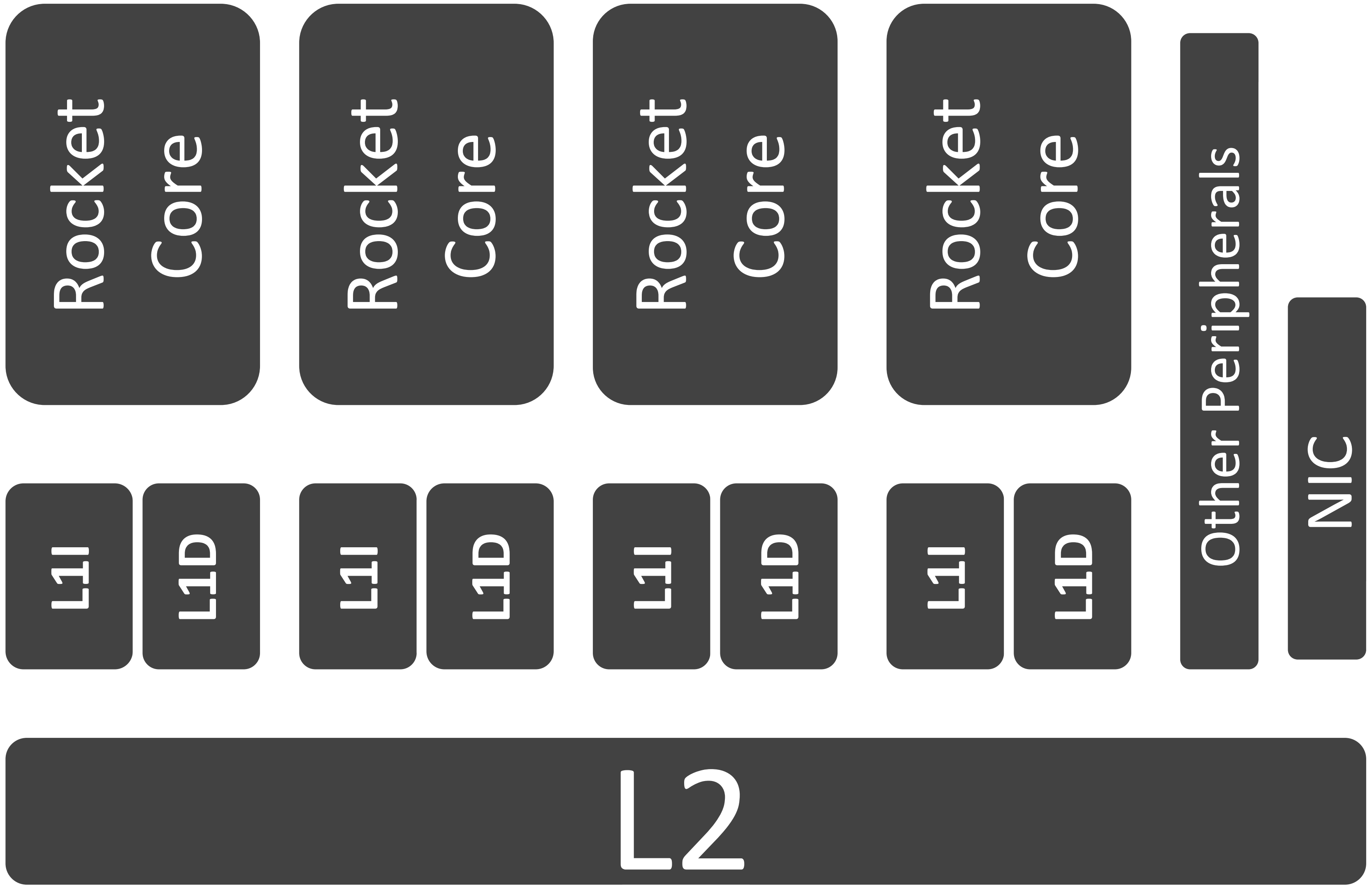
[1] S. Karandikar et. al., “FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud.” *ISCA 2018*
[2] S. Karandikar et. al., “FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud.” *IEEE Micro Top Picks 2018*

Mapping a datacenter simulation

- DC simulation requires:
  - Model hardware at scale, cycle-accurately
  - Run real software
- RTL and abstract SW model co-simulation
- Server Simulations
  - Good fit for the FPGA
  - We have tapeout-proven RTL: FAME-1 transform w/Golden-Gate
- Network simulation
  - Little parallelism in switch models (e.g. a thread per port)
  - Need to coordinate all the distributed server simulations
  - So use CPUs + host network



Step 1: Server SoC in RTL



Modeled System

- 4x RISC-V Rocket Cores @ 3.2 GHz

- 16K I/D L1$

- 256K Shared L2$

- 200 Gb/s Eth. NIC

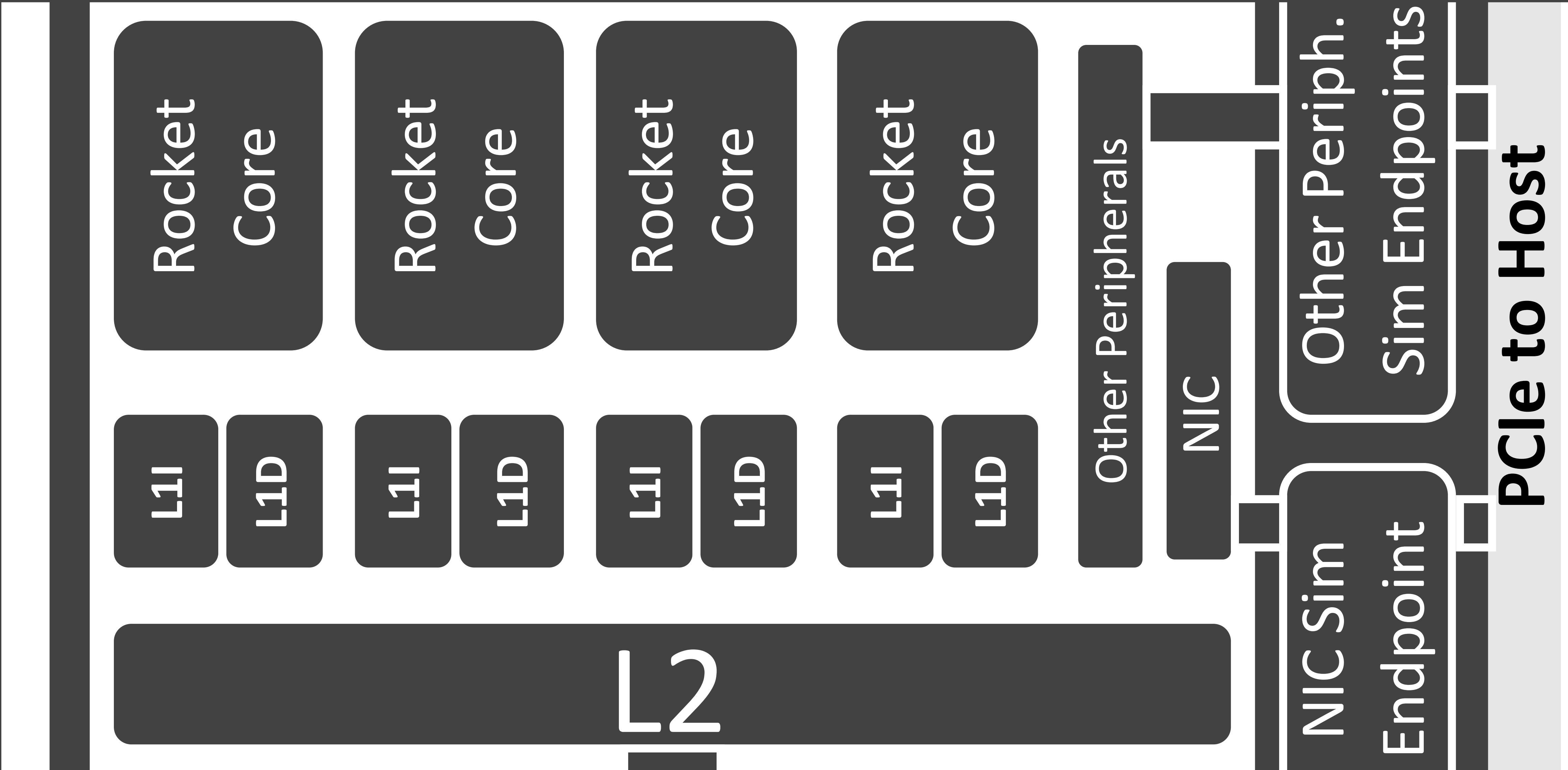
Resource Util.

- < ¼ of an FPGA

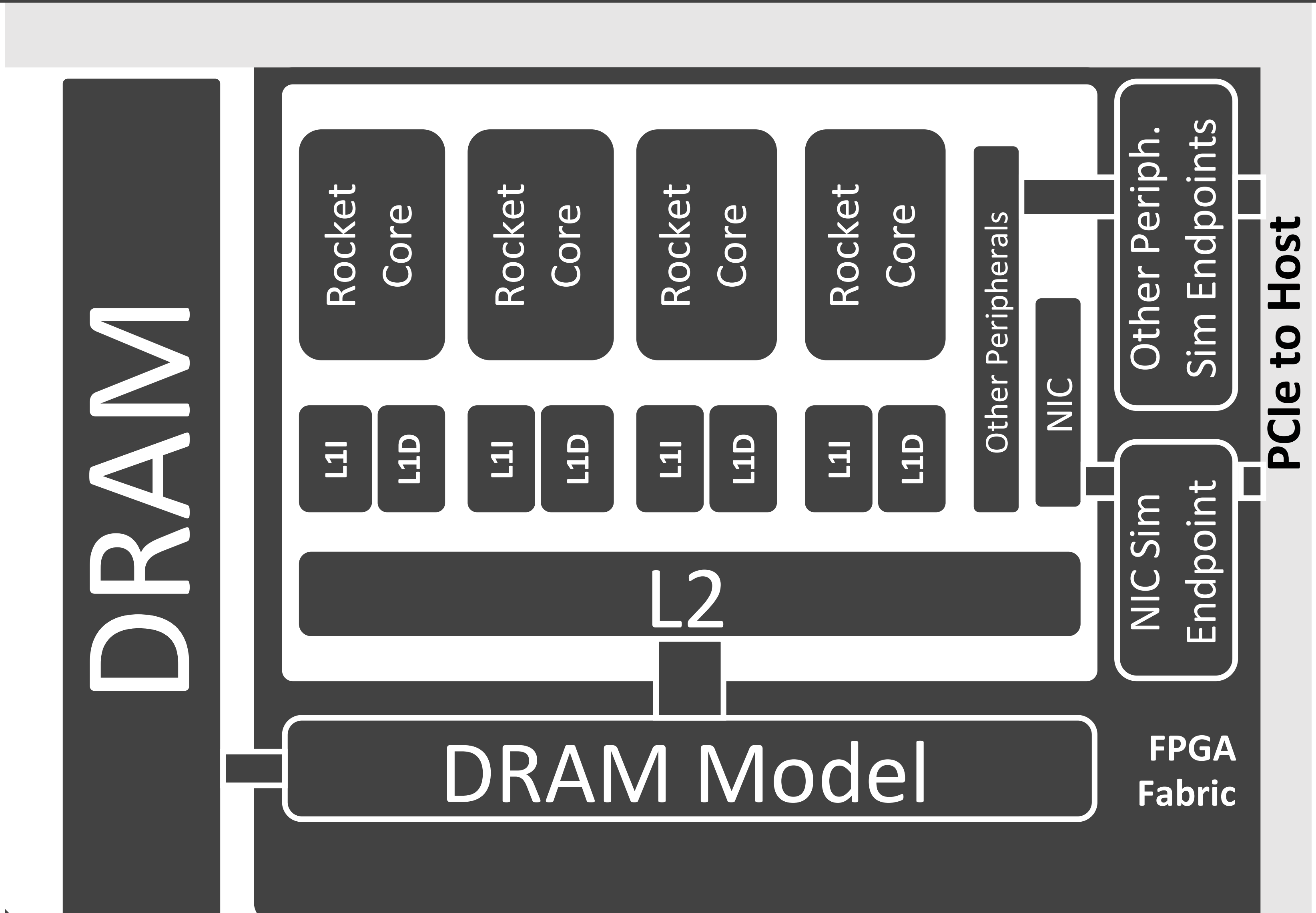
Sim Rate

- N/A

Step 1: Server SoC in RTL



Step 2: FPGA Simulation of one server blade



Modeled System

- 4x RISC-V Rocket Cores @ 3.2 GHz
- 16K I/D L1$
- 256K Shared L2$
- 200 Gb/s Eth. NIC
- 16 GB DDR3

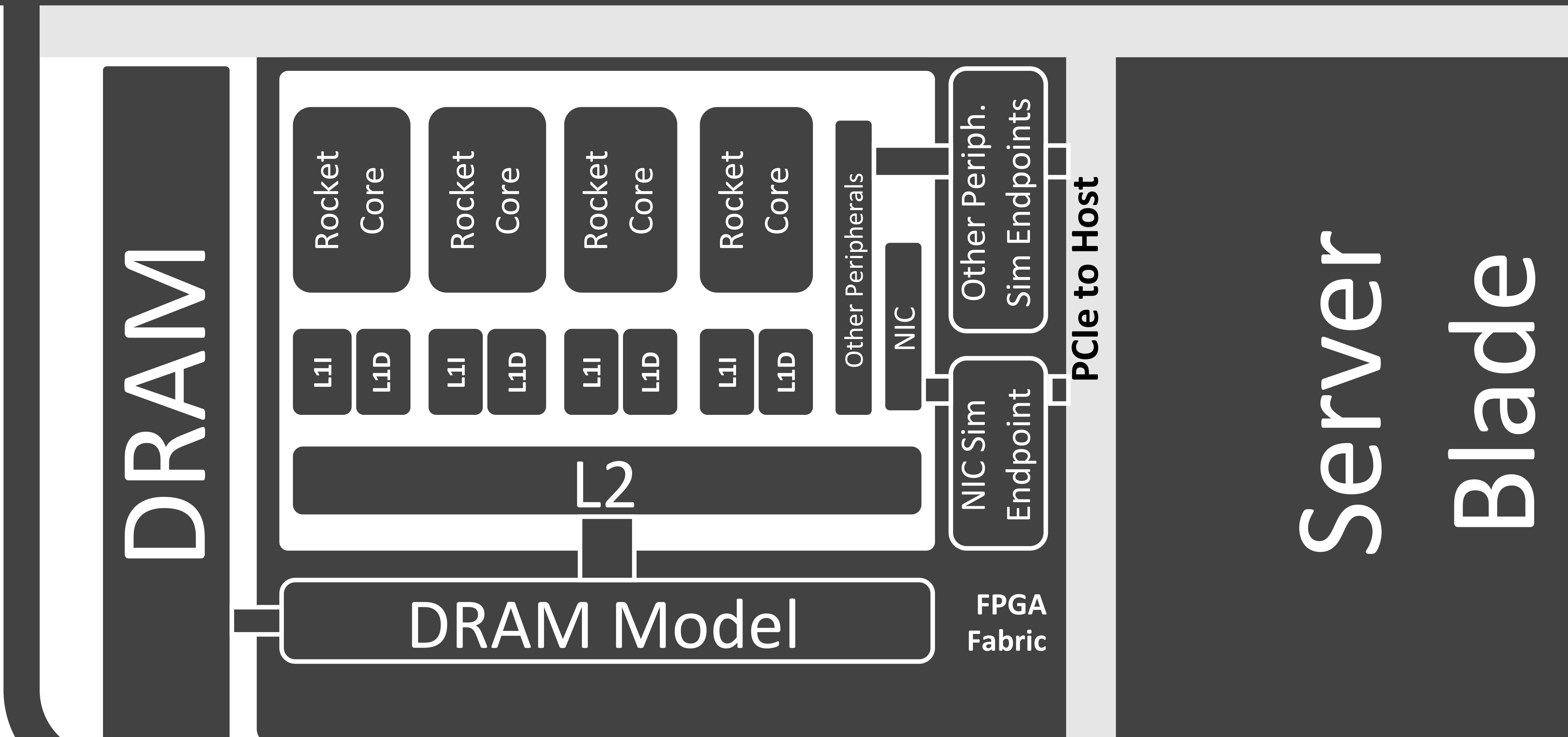
Resource Util.

- < ¼ of an FPGA
- ¼ Mem Chans

Sim Rate

- ~150 MHz
- ~40 MHz (netw)

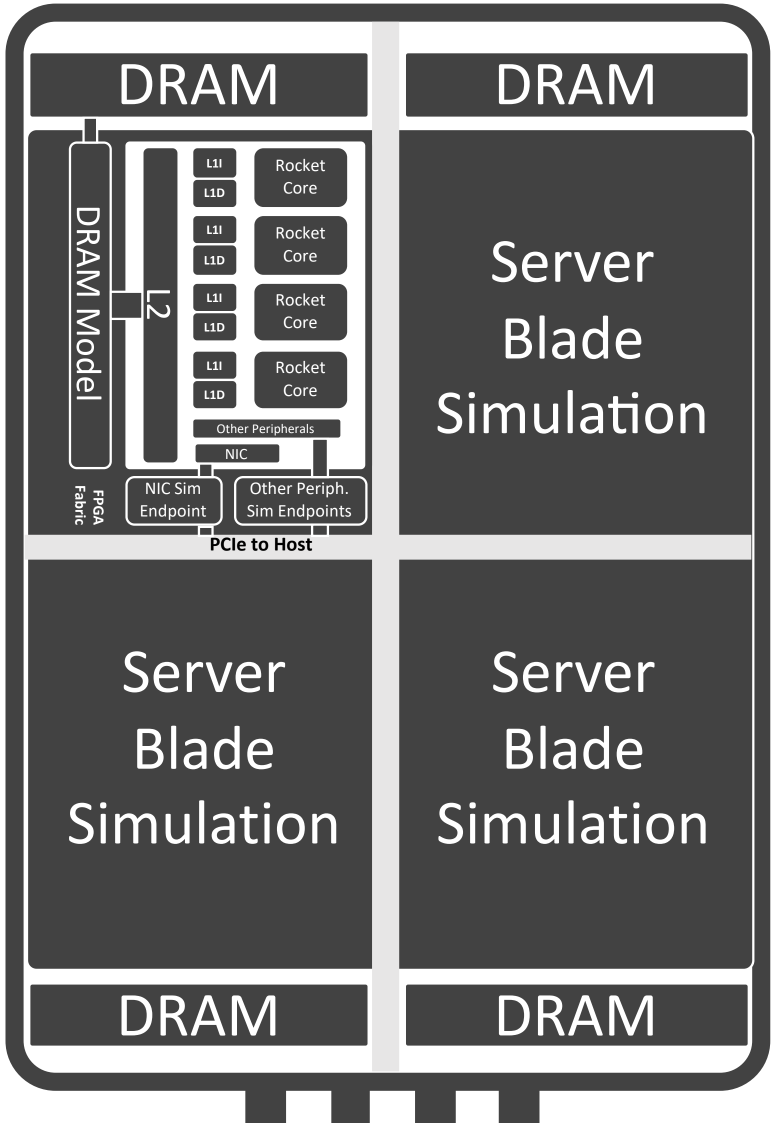
Step 2: FPGA Simulation of one server blade



Step 3: FPGA Simulation of 4 server blades

Cost:
$0.49 per hour (spot)

$1.65 per hour (on-demand)

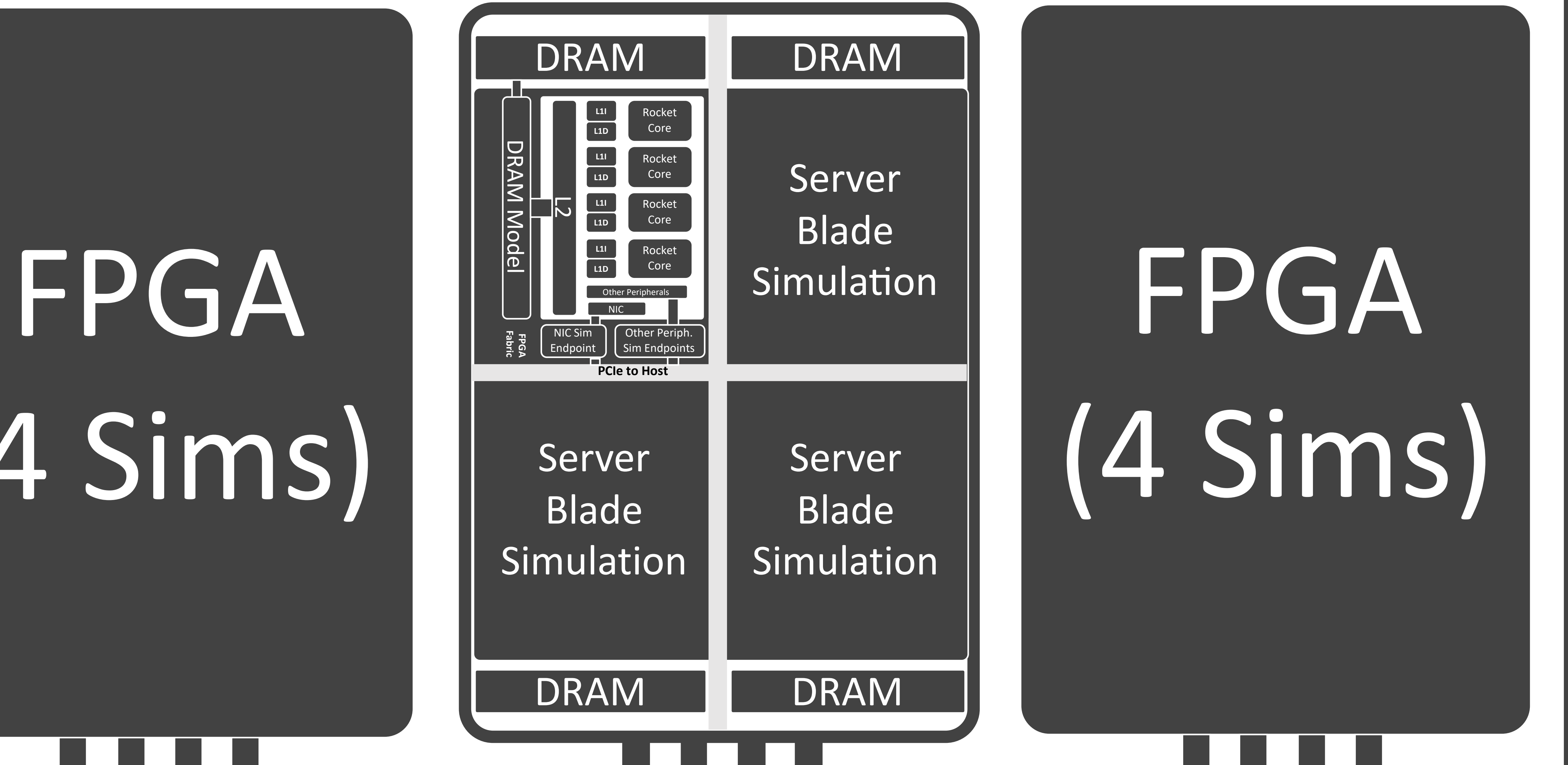


Modeled System
- 4 Server Blades
- 16 Cores
- 64 GB DDR3

Resource Util.
- < 1 FPGA
- 4/4 Mem Chans

Sim Rate
- ~14.3 MHz (netw)

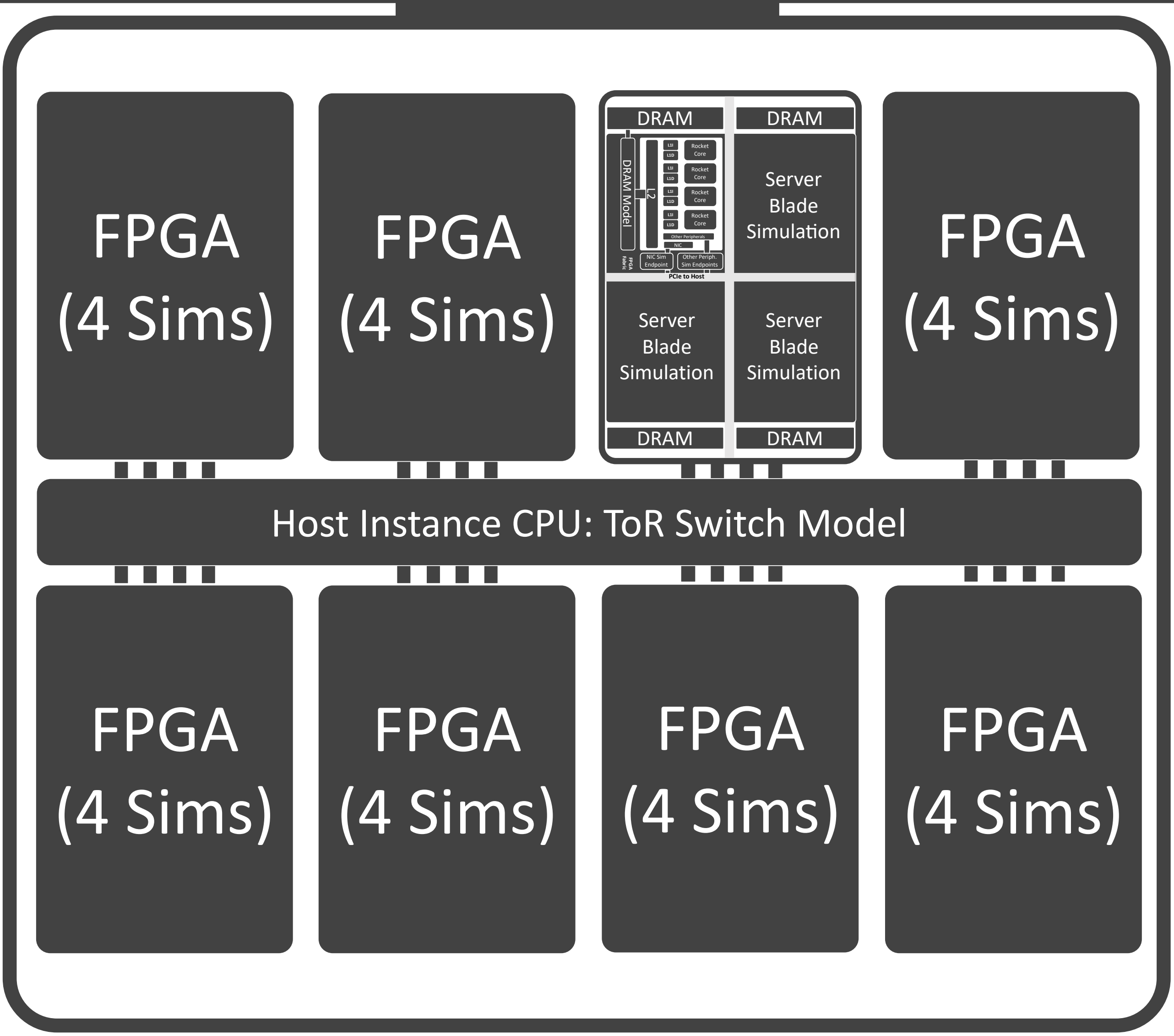
Step 3: FPGA Simulation of 4 server blades



Step 4: Simulating a 32 node rack

Cost:
$2.60 per hour (spot)

$13.20 per hour (on-demand)



Modeled System

- 32 Server Blades
- 128 Cores
- 512 GB DDR3
- 32 Port ToR Switch
- 200 Gb/s, 2us links

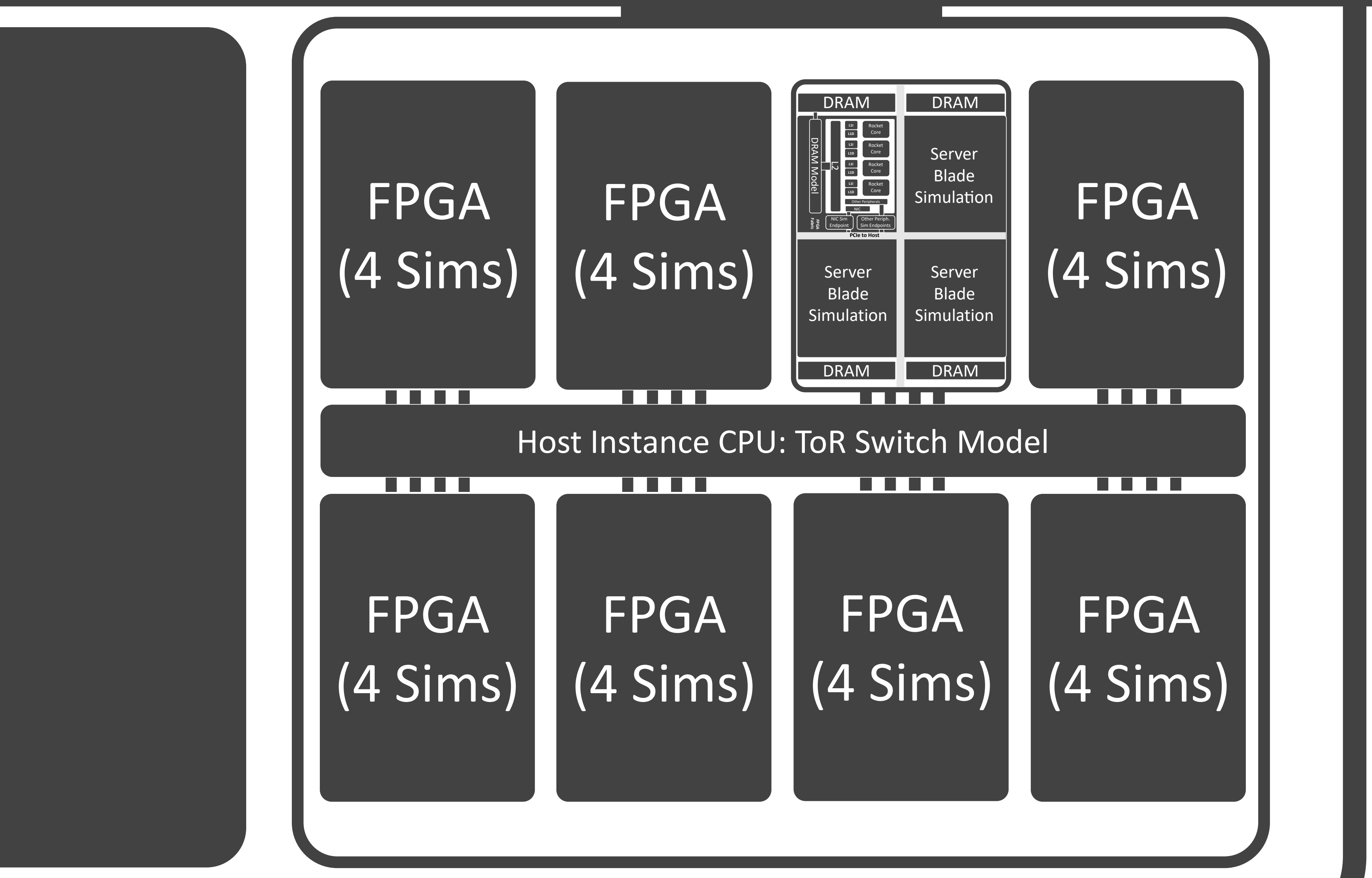
Resource Util.

- 8 FPGAs =
- 1x f1.16xlarge

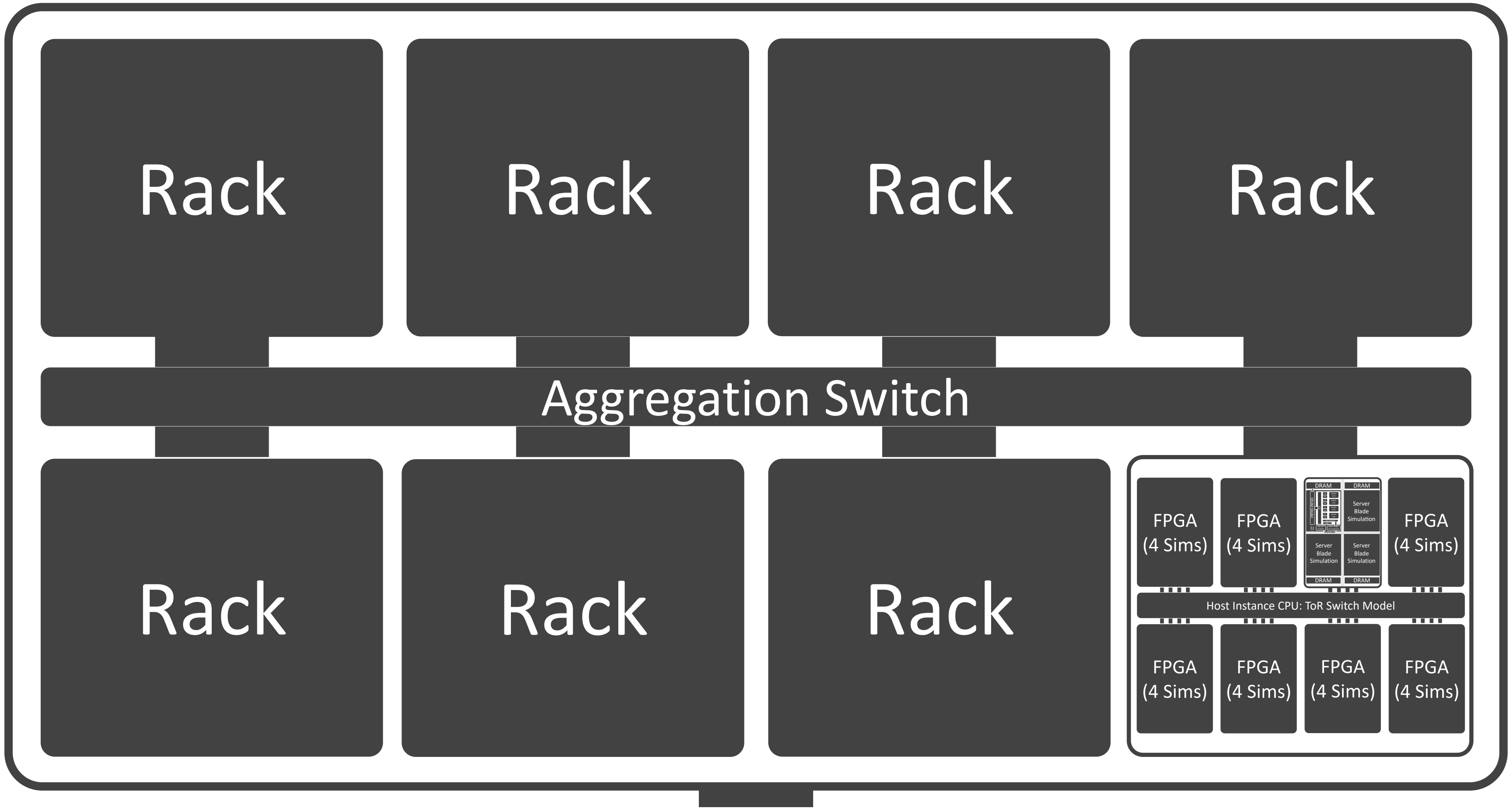
Sim Rate

- ~10.7 MHz (netw)

Step 4: Simulating a 32 node rack



Step 5: Simulating a 256 node "aggregation pod"



Modeled System

- 256 Server Blades
- 1024 Cores
- 4 TB DDR3
- 8 ToRs, 1 Aggr
- 200 Gb/s, 2us links

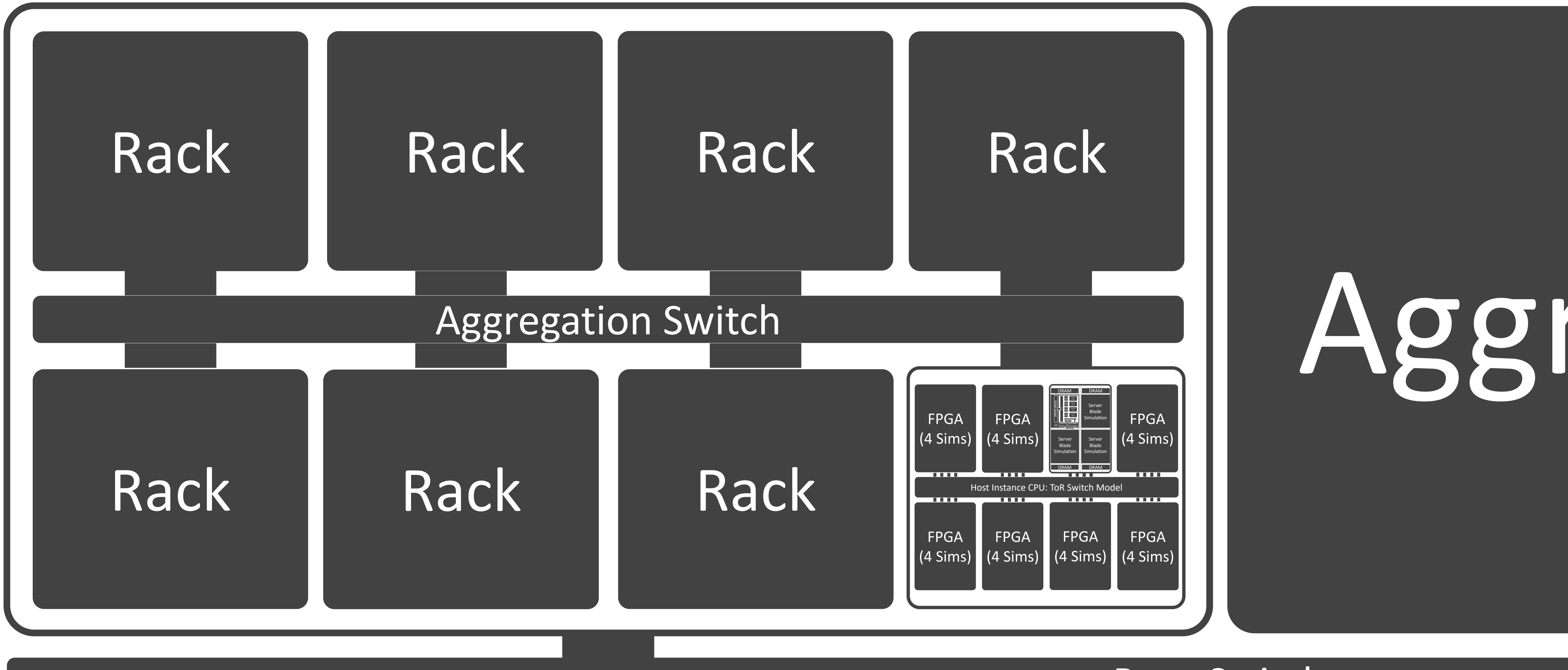
Resource Util.

- 64 FPGAs =
- 8x f1.16xlarge
- 1x m4.16xlarge

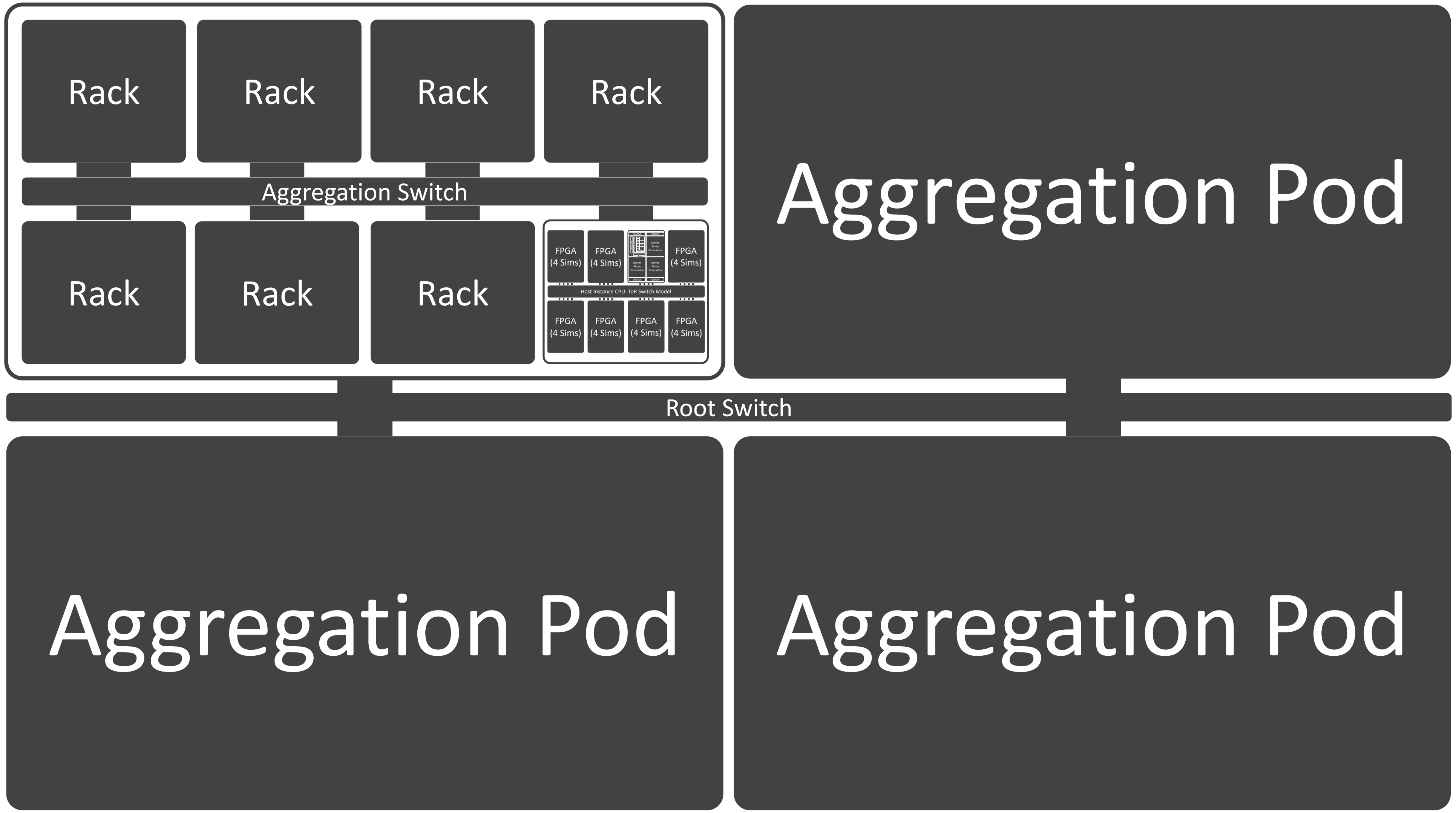
Sim Rate

- ~9 MHz (netw)

Step 5: Simulating a 256 node "aggregation pod"



Step 6: Simulating a 1024 node datacenter



Modeled System

- 1024 Servers
- 4096 Cores
- 16 TB DDR3
- 32 ToRs, 4 Aggr, 1 Root
- 200 Gb/s, 2us links

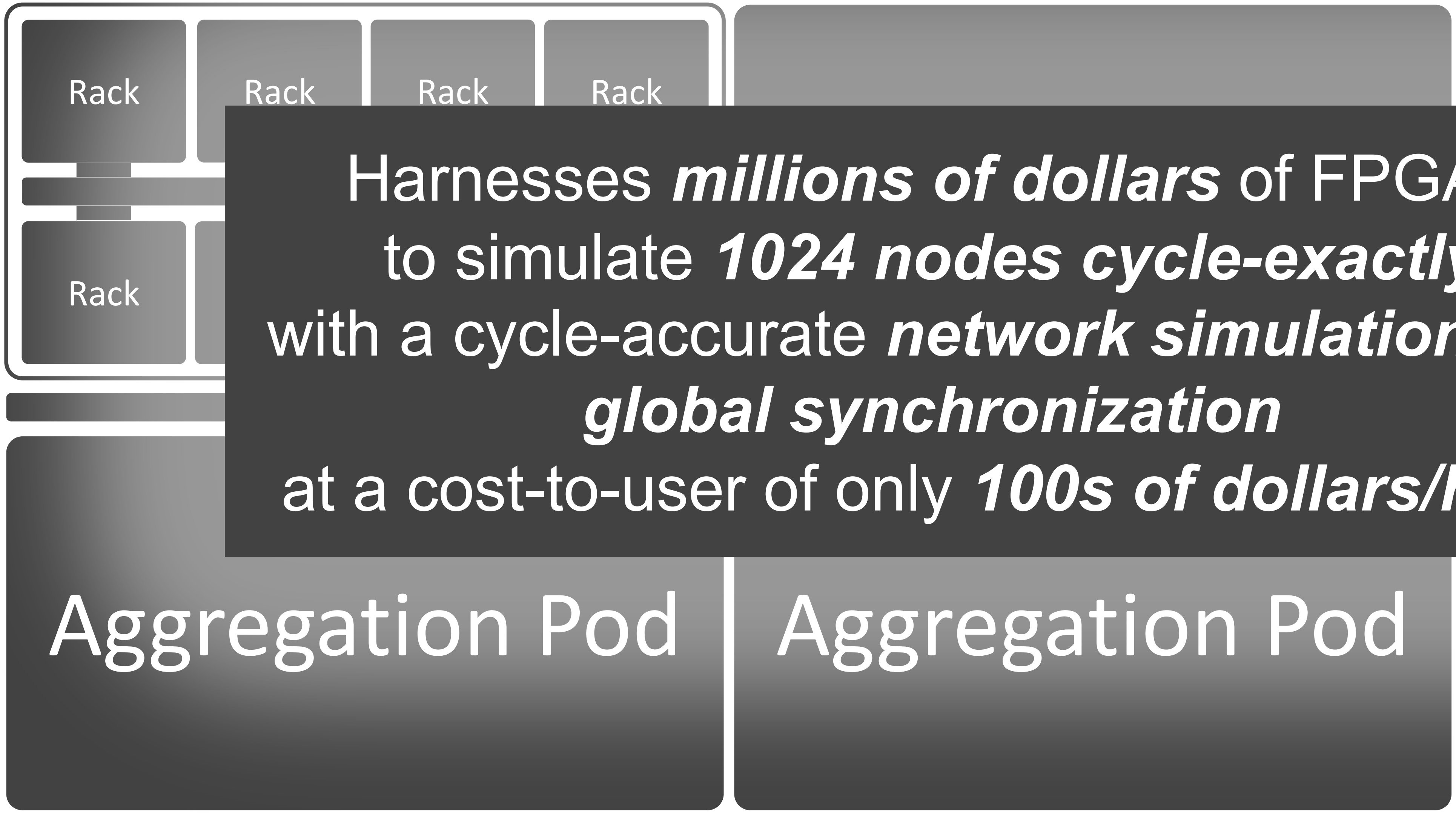
Resource Util.

- 256 FPGAs =
- 32x f1.16xlarge
- 5x m4.16xlarge

Sim Rate

- ~6.6 MHz (netw)

Step 6: Simulating a 1024 node datacenter



Harnesses ***millions of dollars*** of FPGAs
to simulate ***1024 nodes cycle-exactly***
with a cycle-accurate ***network simulation*** and
global synchronization
at a cost-to-user of only ***100s of dollars/hour***

Modeled System

- 1024 Servers
- Cores
- B DDR3
- oRs, 4 Aggr,
- Gb/s, 2us

rce Util.

FPGAs =

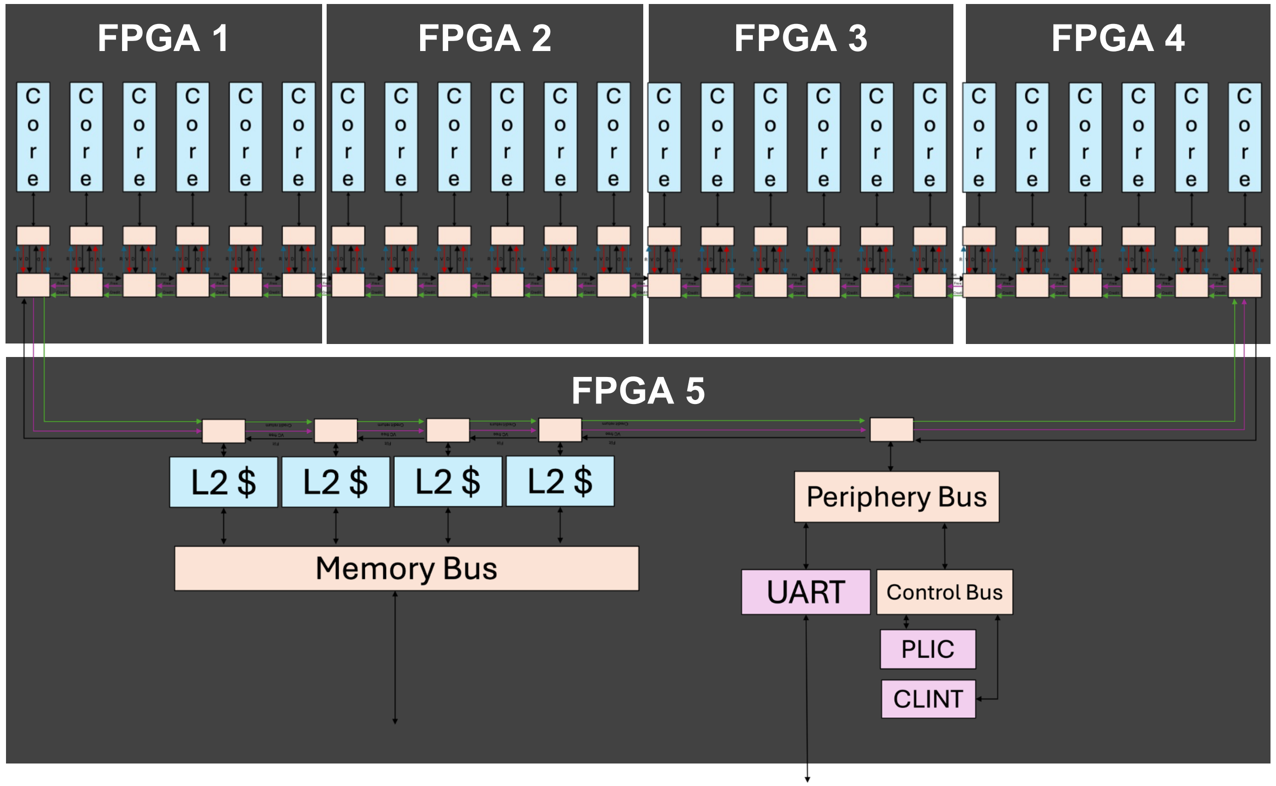
- 32x f1.16xlarge
- 5x m4.16xlarge

Sim Rate

- ~6.6 MHz (netw)

New Features!

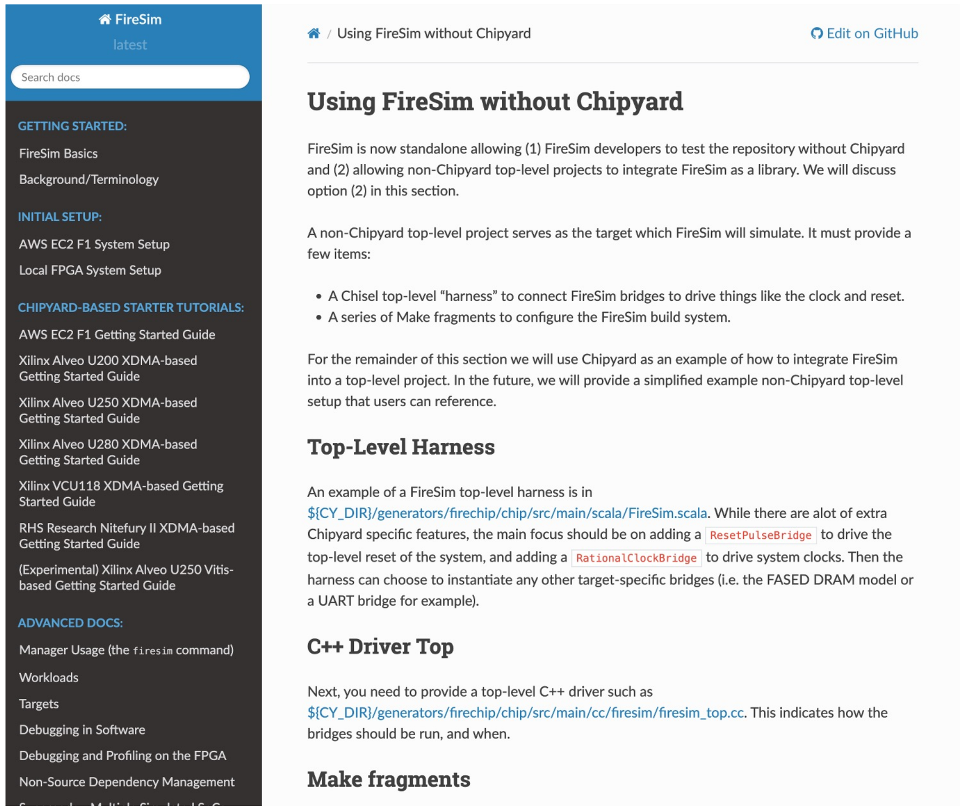
Supports larger designs with FireAxe



- FireAxe now upstreamed
  - (J. Whangbo, et. al., *FireAxe*, ISCA 2024)
  - Supports automatic partitioning of designs onto multiple FPGAs
  - Multiple partition modes and optimizations while being cycle-exact
- Example use cases
  - Partitioning extremely large BOOM out-of-order core with 2x FPGAs
  - Partitioning 24 BOOM Core SoC with 5x FPGAs

Simulating non-Chipyard-based SoCs

- What about your own non-Chipyard design?
  - Isolated testing of single RTL component
  - Unique SoC top-level specific to your needs
  - Other unique usages
- FireSim now supports this!
  - Use FireSim like Verilator/VCS
    - FireSim is now a library decoupled from top-level
  - Cleaner API for target-specific bridges + harnesses
  - Use modern Chisel (and/or older Chisel versions)
- New release coming soon!
  - New docs on library usage
  - Examples on non-SoC top-levels


FireSim
latest
Search docs
GETTING STARTED:
FireSim Basics
Background/Terminology
INITIAL SETUP:
AWS EC2 F1 System Setup
Local FPGA System Setup
CHIPYARD-BASED STARTER TUTORIALS:
AWS EC2 F1 Getting Started Guide
Xilinx Alveo U200 XDMA-based Getting Started Guide
Xilinx Alveo U250 XDMA-based Getting Started Guide
Xilinx Alveo U280 XDMA-based Getting Started Guide
Xilinx VCU118 XDMA-based Getting Started Guide
RHS Research Nitefury II XDMA-based Getting Started Guide
(Experimental) Xilinx Alveo U250 Vitis-based Getting Started Guide
ADVANCED DOCS:
Manager Usage (the firesim command)
Workloads
Targets
Debugging in Software
Debugging and Profiling on the FPGA
Non-Source Dependency Management
/ Using FireSim without Chipyard
Edit on GitHub

Using FireSim without Chipyard

FireSim is now standalone allowing (1) FireSim developers to test the repository without Chipyard and (2) allowing non-Chipyard top-level projects to integrate FireSim as a library. We will discuss option (2) in this section.

A non-Chipyard top-level project serves as the target which FireSim will simulate. It must provide a few items:

- A Chisel top-level "harness" to connect FireSim bridges to drive things like the clock and reset.
- A series of Make fragments to configure the FireSim build system.

For the remainder of this section we will use Chipyard as an example of how to integrate FireSim into a top-level project. In the future, we will provide a simplified example non-Chipyard top-level setup that users can reference.

Top-Level Harness

An example of a FireSim top-level harness is in ${CY_DIR}/generators/firechip/chip/src/main/scala/FireSim.scala. While there are alot of extra Chipyard specific features, the main focus should be on adding a `ResetPulseBridge` to drive the top-level reset of the system, and adding a `RationalClockBridge` to drive system clocks. Then the harness can choose to instantiate any other target-specific bridges (i.e. the FASED DRAM model or a UART bridge for example).

C++ Driver Top

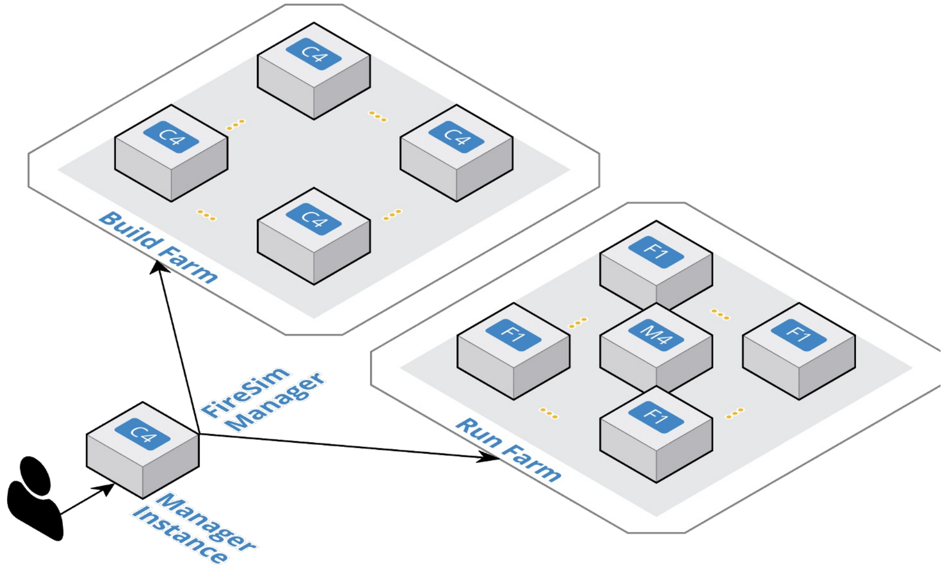
Next, you need to provide a top-level C++ driver such as ${CY_DIR}/generators/firechip/chip/src/main/cc/firesim/firesim_top.cc. This indicates how the bridges should be run, and when.

Make fragments

Conclusion

Productive Open-Source FPGA Simulation

- github.com/firesim/firesim, BSD Licensed
- An "easy" button for fast, FPGA-accelerated full-system simulation
  - Plug in your own RTL designs, your own HW/SW models
  - One-click: Parallel FPGA builds, Simulation run/result collection, building target software
  - Scales to a variety of use cases:
    - Networked (performance depends on scale)
    - Non-networked (150+ MHz), limited by your budget
- `firesim` command line program
  - Like `docker` or `vagrant`, but for FPGA sims
  - User doesn't need to care about distributed magic happening behind the scenes

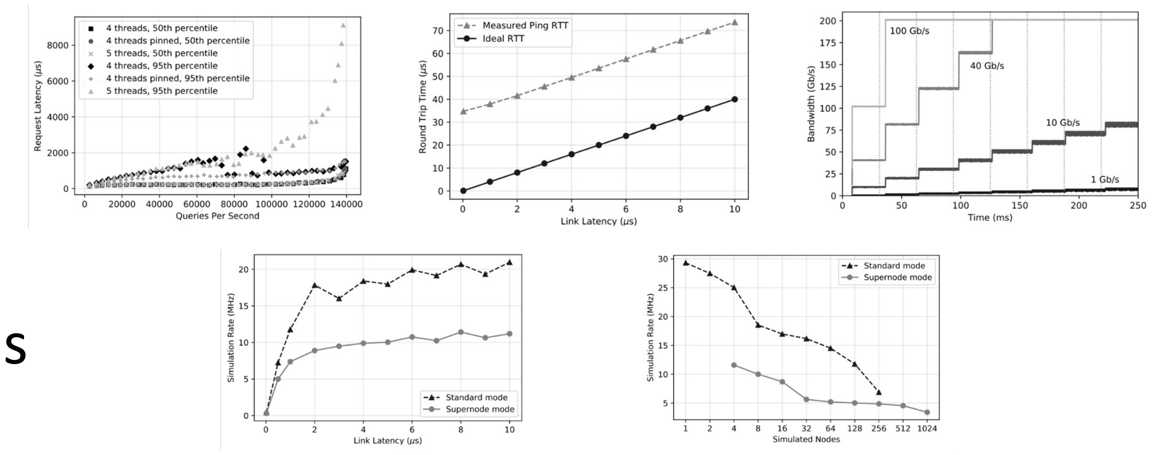


FireSim Developer Environment

Productive Open-Source FPGA Simulation

- Scripts can call `firesim` to fully automate distributed FPGA sim
  - **Reproducibility**: included scripts to reproduce ISCA 2018 results
  - e.g. scripts to automatically run SPECInt2017 with full **reference inputs** in ≈1 day
  - Many others included
  - Several user papers have gone through artifact evaluation using FireSim (*nanoPU*, *FirePerf*, *Protobuf accel.*, *MoCA*, *Simulator Independent Coverage*, etc.)
- 200+ pages of documentation: https://docs.fires.im
- AWS provides grants for researchers: https://aws.amazon.com/grants/
- Xilinx University Program provides FPGA donations for university researchers: https://www.xilinx.com/support/university.html

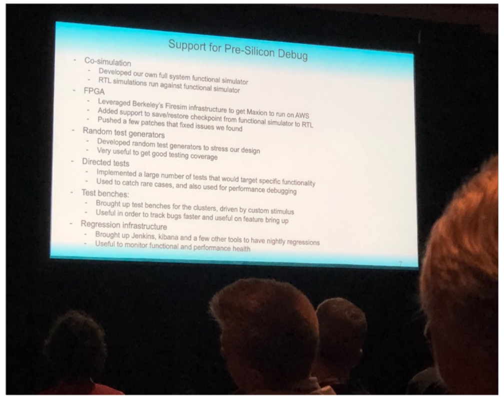
```
$ cd fsim/deploy/workloads
$ ./run-all.sh
```



Join the FireSim Community!: Open-source users and industrial users

- More than 200 mailing list members and 850 unique cloners per-week
- Projects with public FireSim support
  - Chipyard
  - Rocket Chip
  - Constellation NoC
  - BOOM
  - Hwacha Vector Accelerator
  - Keystone Secure Enclave
  - Gemmini + AuRORA
  - NVIDIA Deep Learning Accelerator (NVDLA)
    - NVIDIA Blog post: https://devblogs.nvidia.com/nvdla/
  - BOOM Spectre replication/mitigation
  - Protobuf/Compression Accelerator
  - Too many to list here!

- Companies publicly announced using FireSim
  - Esperanto Maxion ET
  - Intensivate IntenCore
  - SiFive validation paper @ VLSI'20
  - Galois and Lockheed Martin (DARPA SSITH/FETT)



Support for Pre-Silicon Debug

- Co-simulation
  - Developed our own full system functional simulator
  - RTL simulations run against functional simulator
- FPGA
  - Leveraged Berkeley's Firesim infrastructure to get Maxion to run on AWS
  - Added support to save/restore checkpoint from functional simulator to RTL
  - Pushed a few patches that fixed issues we found
- Random test generators
  - Developed random test generators to stress our design
  - Very useful to get good testing coverage
- Directed tests
  - Implemented a large number of tests that would target specific functionality
  - Used to catch rare cases, and also used for performance debugging
- Test benches:
  - Brought up test benches for the clusters, driven by custom stimulus
  - Useful in order to track bugs faster and useful on feature bring up
- Regression infrastructure
  - Brought up Jenkins, kibana and a few other tools to have nightly regressions
  - Useful to monitor functional and performance health

Esperanto announcement at RISC-V Summit 2018

FireSim in DARPA FETT

- DARPA SSITH: Building hardware defenses to address common software vulnerabilities
- DARPA FETT: How good are the defenses built in SSITH?
  - Multiple designs hosted for attack in FireSim [1]
- “Morpheus II: A RISC-V Security Extension for Protecting Vulnerable Software and Hardware”
  - Developed by UT Austin, U Mich., Agita Labs
  - Hosted on FireSim for FETT [2]
  - Over 500 attackers tried to break Morpheus II defenses, working for large bug bounties. None succeeded [3]

[1] K. Hopfer. Leveraging Amazon EC2 F1 Instances for Development and Red Teaming in DARPA’s First-Ever Bug Bounty Program. AWS APN Blog. May 2021.
[2] A. Harris, et. al., “Morpheus II: A RISC-V Security Extension for Protecting Vulnerable Software and Hardware”. In proceedings of the 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), December 2021.
[3] T. Austin., et. al., “Morpheus II: A RISC-V Security Extension for Protecting Vulnerable Software and Hardware”. In HotChips 33, August 2021.

Join the FireSim Community!: Academic Users and Awards

- **ISCA '18**: Maas et. al. HW-GC Accelerator (**Berkeley**)
- **MICRO '18**: Zhang et. al. "Composable Building Blocks to Open up Processor Design" (**MIT**)
- **RTAS '20**: Farshchi et. al. BRU (**Kansas**)
- **EuroSys '20**: Lee et. al. Keystone (**Berkeley**)
- **OSDI '21**: Ibanez et. al. nanoPU (**Stanford**)
- **USENIX Security '21**: Saileshwar et. al. MIRAGE (**Georgia Tech**)
- **CCS '21**: Ding et. al. "Hardware Support to Improve Fuzzing Performance and Precision" (**Georgia Tech**)
- **MICRO '21:** Karandikar et. al. "A Hardware Accelerator for Protocol Buffers" (**Berkeley/Google**)
- **MICRO '21**: Gottschall et. al. TIP (**NTNU**)
- **Over 20 *additional* user papers on the FireSim website:**
  - https://fires.im/publications/#userpapers

- Awards: FireSim ISCA '18 paper:
  - IEEE Micro Top Pick
  - CACM Research Highlights Nominee from ISCA '18
- Awards: FireSim users:
  - ISCA '18 Maas et. al.:
    - IEEE Micro Top Pick
  - MICRO '18 Zhang et. al.:
    - IEEE Micro Top Pick
  - MICRO '21 Gottschall et. al.:
    - MICRO-54 Best paper runner-up
  - MICRO '21 Karandikar et. al.:
    - MICRO-54 Distinguished Artifact winner
    - IEEE Micro Top Pick Honorable Mention
  - DAC '21 Genc et. al.:
    - DAC 2021 Best Paper winner
  - MICRO '23 Kim et. al.:
    - IEEE Micro Top Pick
  - ISCA '24 Whangbo et. al.:
    - Distinguished Artifact winner

Join the FireSim Community!: Academic Users and Awards

- **ISCA '18**: Maas et. al. HW-GC Accelerator (**Berkeley**)
- **MICRO '18**: Zhang et. al. "Composable Building Blocks to Open up Processor De
- **RTAS '20**: Fa
- **EuroSys '20**
- **OSDI '21**: Iban
- **USENIX Securi**
- **CCS '21**: Din Performance
- **MICRO '21:** Karandikar et. al. "A Hardware Accelerator for Protocol Buffers" (**Berkeley/Google**)
- **MICRO '21**: Gottschall et. al. TIP (**NTNU**)
- **Over 20 *additional* user papers on the FireSim website:**
  - https://fires.im/publications/#userpapers

FireSim has been used in published work from authors at over 20 academic and industrial institutions*

**actually used, not only cited*

- Awards: FireSim ISCA '18 paper:
  - IEEE Micro Top Pick
  - s Nominee from
  - l.:
    - nner-up
  - al.:
    - Artifact winner
    - orable Mention
  - DAC '21 Genc et. al.:
    - DAC 2021 Best Paper winner
  - MICRO '23 Kim et. al.:
    - IEEE Micro Top Pick
  - ISCA '24 Whangbo et. al.:
    - Distinguished Artifact winner

FireSim

Questions?

Berkeley Architecture Research

Learn More:

Web: https://fires.im

Docs: https://docs.fires.im

GitHub: https://github.com/firesim/firesim

Mailing List:
https://groups.google.com/forum/#!forum/firesim

@firesimproject

Email: abe.gonzalez@berkeley.edu

The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849, by DARPA, Award Number HR0011-12-2-0016, and by NSF CCRI ENS Chipyard Award #2016662. Research was also partially funded by SLICE/ADEPT Lab industrial sponsors and affiliates Amazon, Apple, Google, Intel, Qualcomm, and Western Digital, and RISE Lab sponsor Amazon Web Services. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.