Chipyard + FireSim

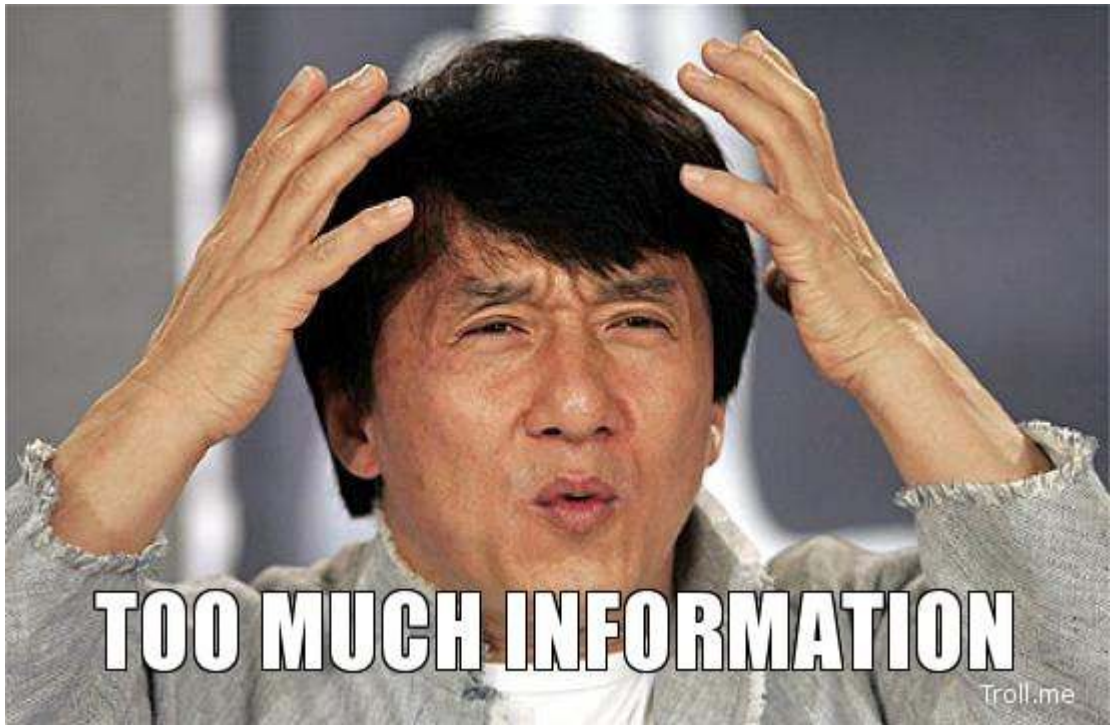
Generating and Evaluating Complex RISC-V SoCs

Abraham Gonzalez
EE290-2 Hardware for Machine Learning
Spring 2021
UC Berkeley

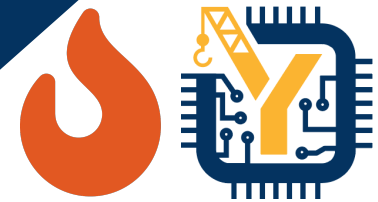
Berkeley Architecture Research

This Session

- Chipyard Introduction
- FireSim Introduction
- AWS Account Setup
- FireSim Initial Setup

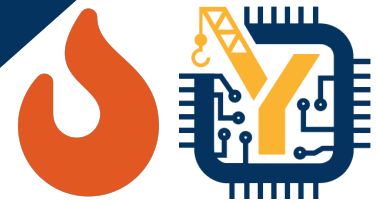


A **Golden Age** in Computer Architecture



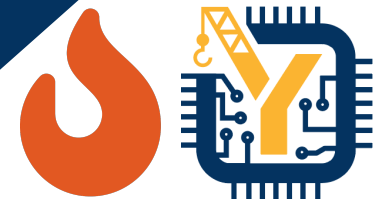
- Traditional scaling is over
- Custom microarchitectures, HW/SW codesign enable performance gains through specialization
- Golden age for **open-source** across the entire stack
  - Open-source hardware
  - Open ISAs
  - Open-source operating systems, compilers, applications
- Let's build systems from open-source components, use custom IP for specialization

What are the Challenges?



- There's a lot of open IP … how do I use any of it?
  - What do I do with a dump of open-source Verilog?
  - How to build a system out of mismatched components?
  - IP is only as good as the infrastructure around it
- How to obtain performance measurements quickly for my custom HW/SW system?
  - SW architectural simulators are slow and inaccurate
  - Hardware emulators are pricey, hard to use
  - Can't wait until tape-out to get design feedback
- How do I perform agile tape-outs with small teams?
  - Tape-out with 10s of engineers/graduate students

Two hard questions, answered!



- Where do I get a collection of well-tested hardware IP + tooling + complex software stacks that run on it?

- How do I quickly obtain performance measurements for a novel HW/SW system?

What can I do with these tools?

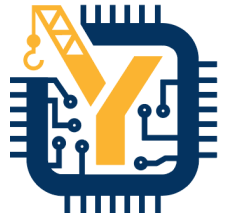
Evaluate Functionality, Performance, Power, Area, Frequency *for real HW/SW systems*, quickly and easily, with small teams of engineers

Chipyard Basics

Berkeley
Architecture
Research

CHIPYARD

Berkeley Has Generated a Lot of Stuff!



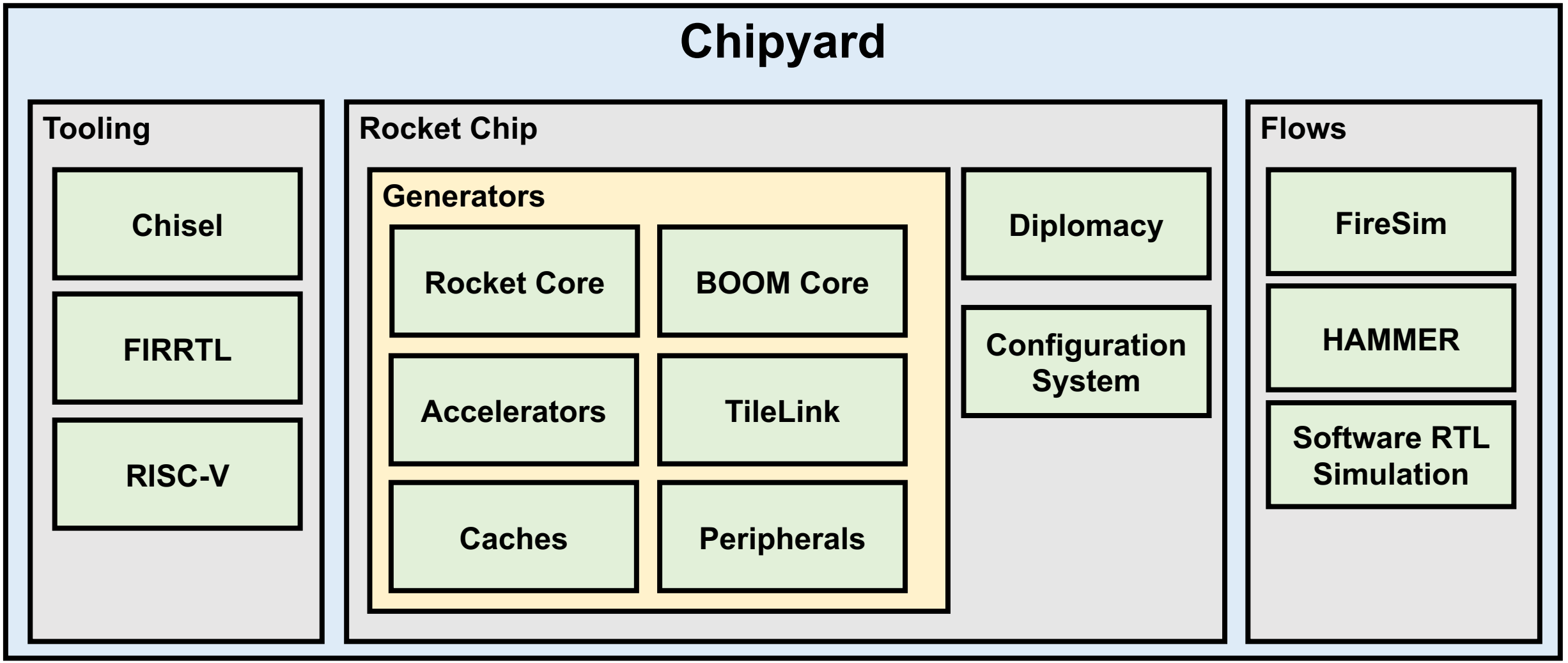
Berkeley Architecture Research has developed and open-sourced:

- Chisel
- BOOM Core
- Diplomacy
- FireSim
- FIRRTL
- Rocket Core
- TileLink
- Configuration System
- RISC-V
- Accelerators
- Caches
- Peripherals
- HAMMER

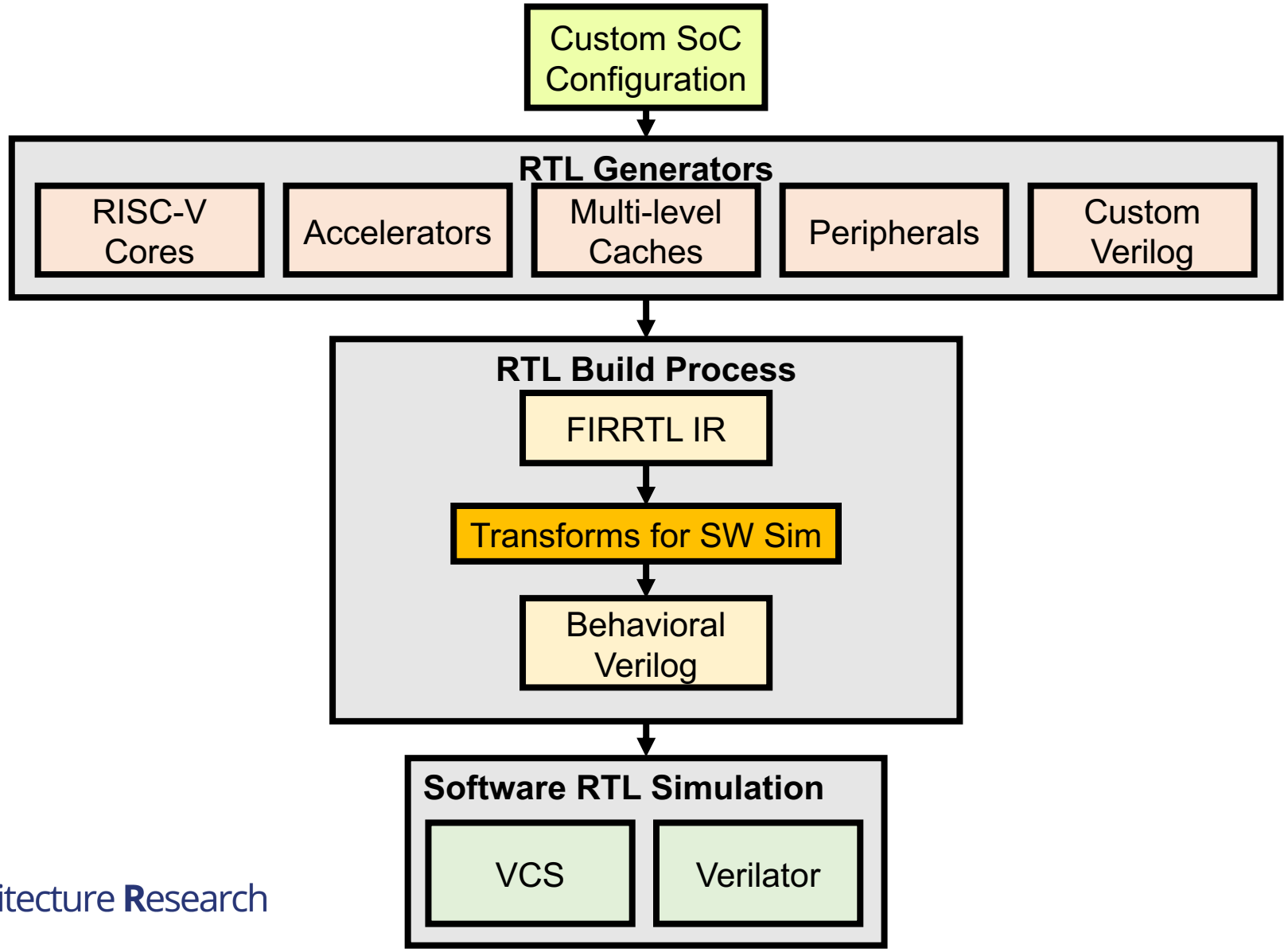
But...

How do I put this all together to do something useful

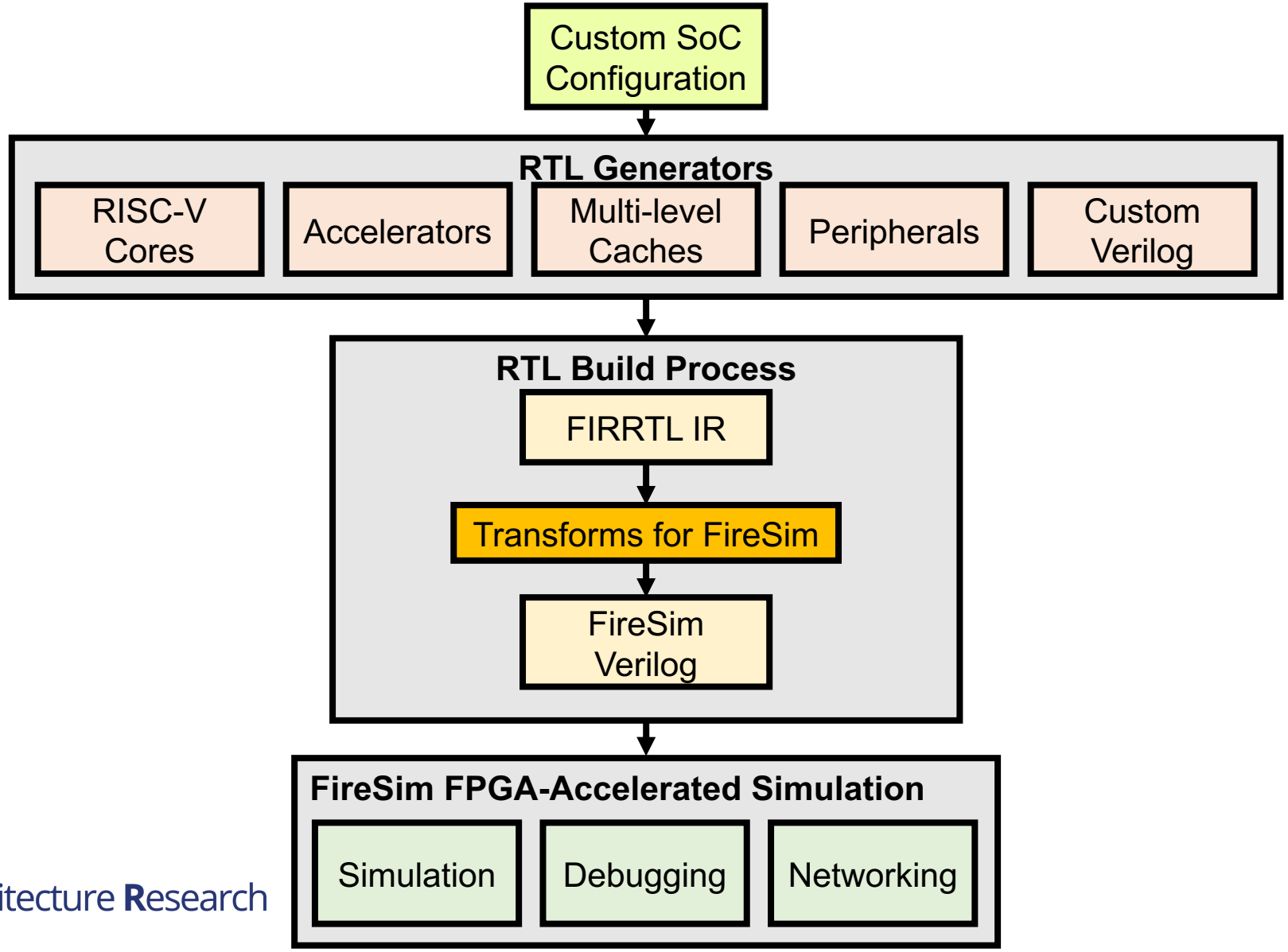
Chipyard


Chipyard
Tooling
Chisel
FIRRTL
RISC-V
Rocket Chip
Generators
Rocket Core
BOOM Core
Accelerators
TileLink
Caches
Peripherals
Diplomacy
Configuration System
Flows
FireSim
HAMMER
Software RTL Simulation

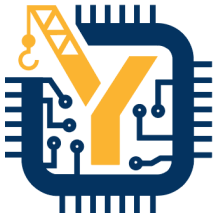
Chipyard SW RTL Simulation



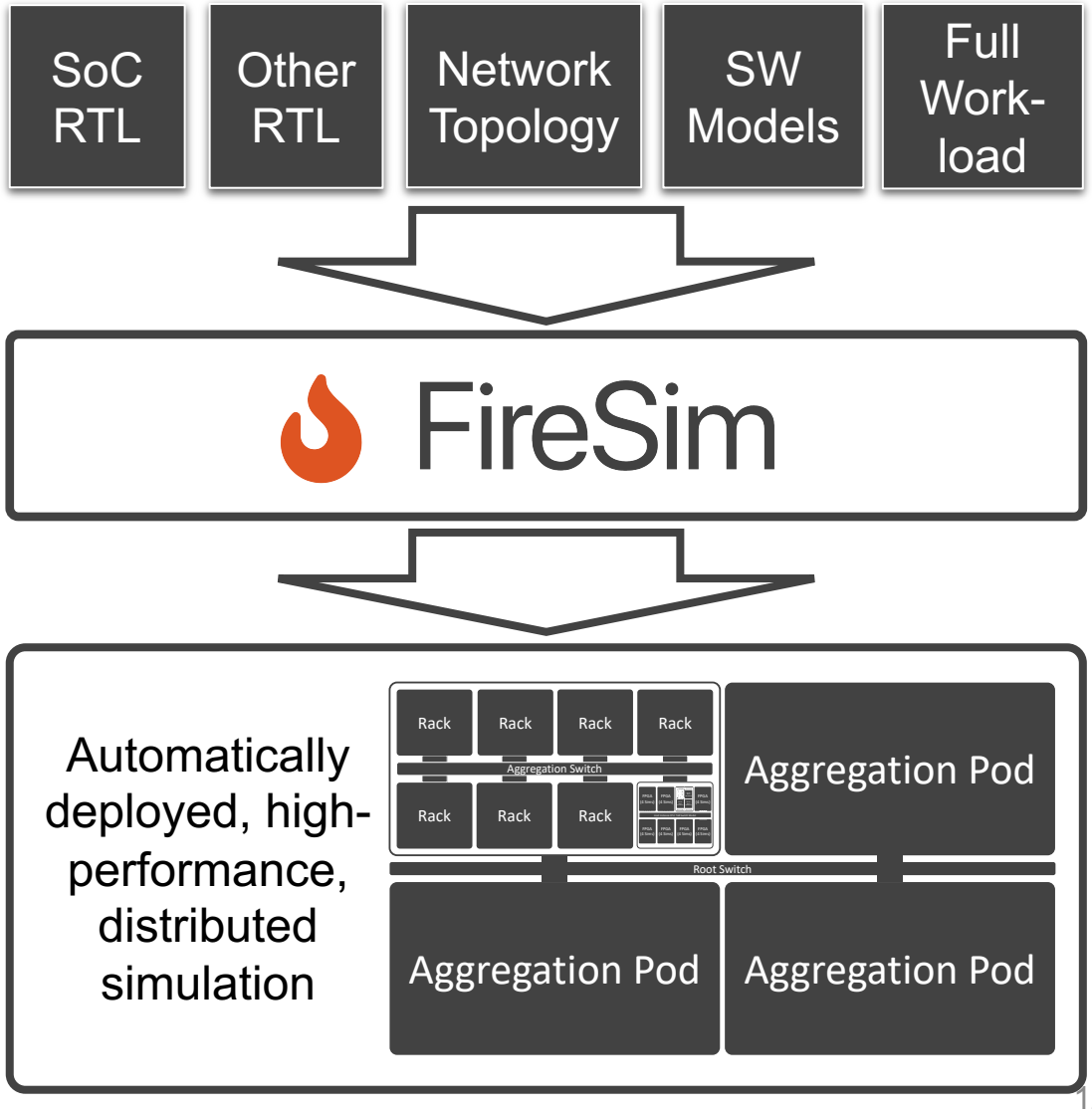
Chipyard targeting FireSim



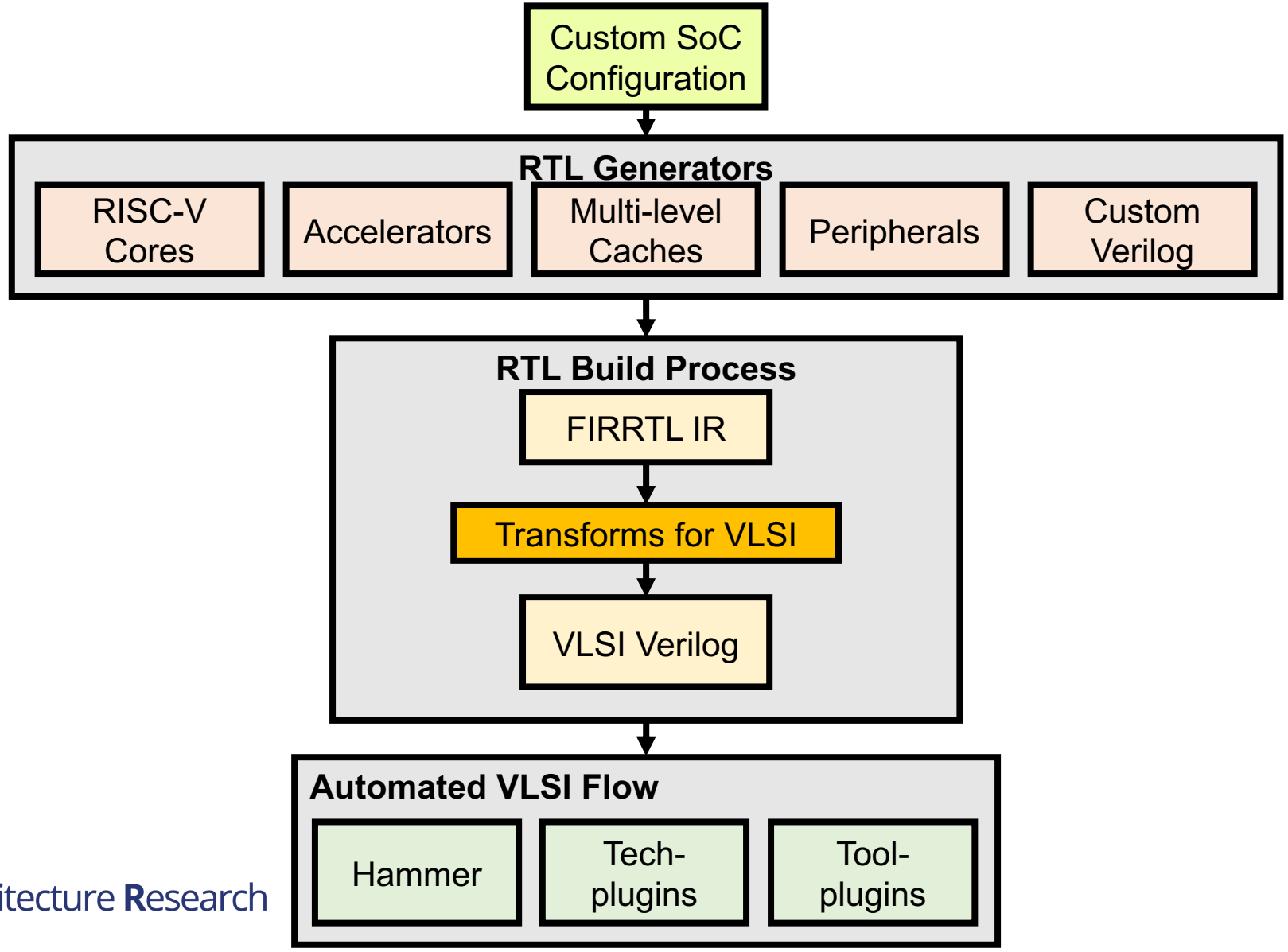
FireSim



- FPGA-accelerated simulation on the Amazon EC2 public cloud
- Originally developed for cycle-exact architectural exploration of data-center clusters
- Not FPGA prototypes, rather FPGA-accelerated simulators
- Cloud FPGAs
  - Inexpensive, elastic supply of large FPGAs
  - Easy to collaborate with other researchers

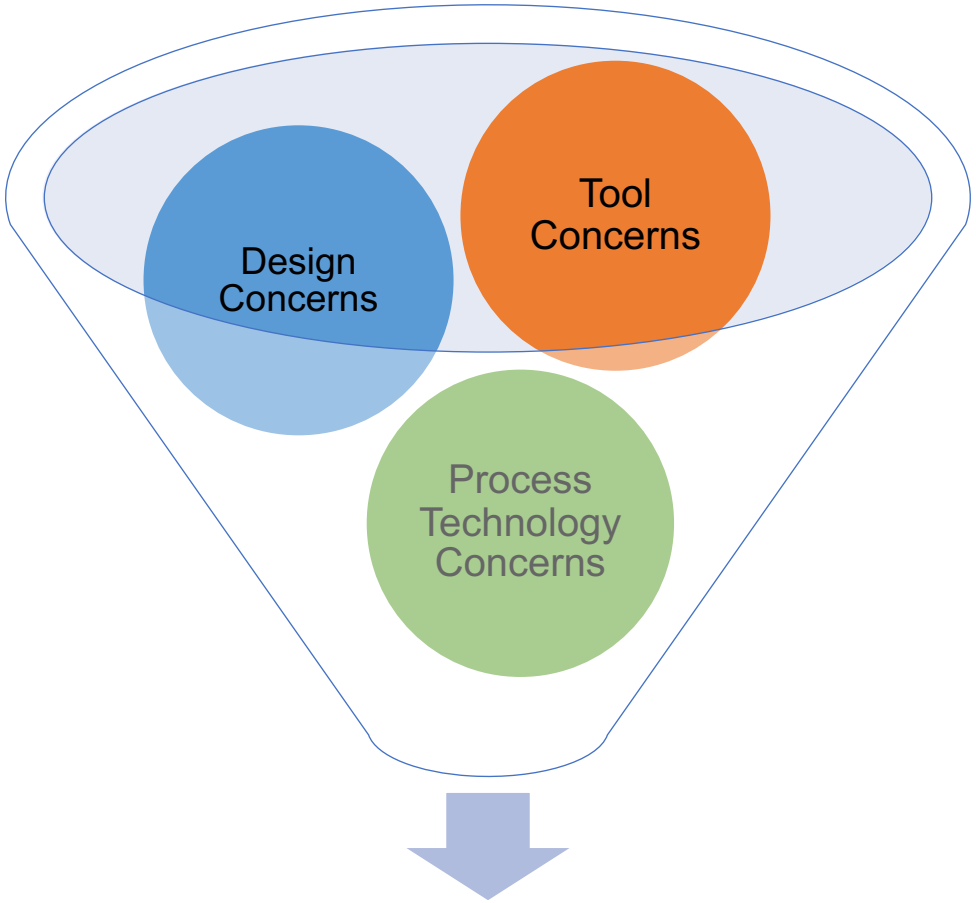


Chipyard VLSI Flow

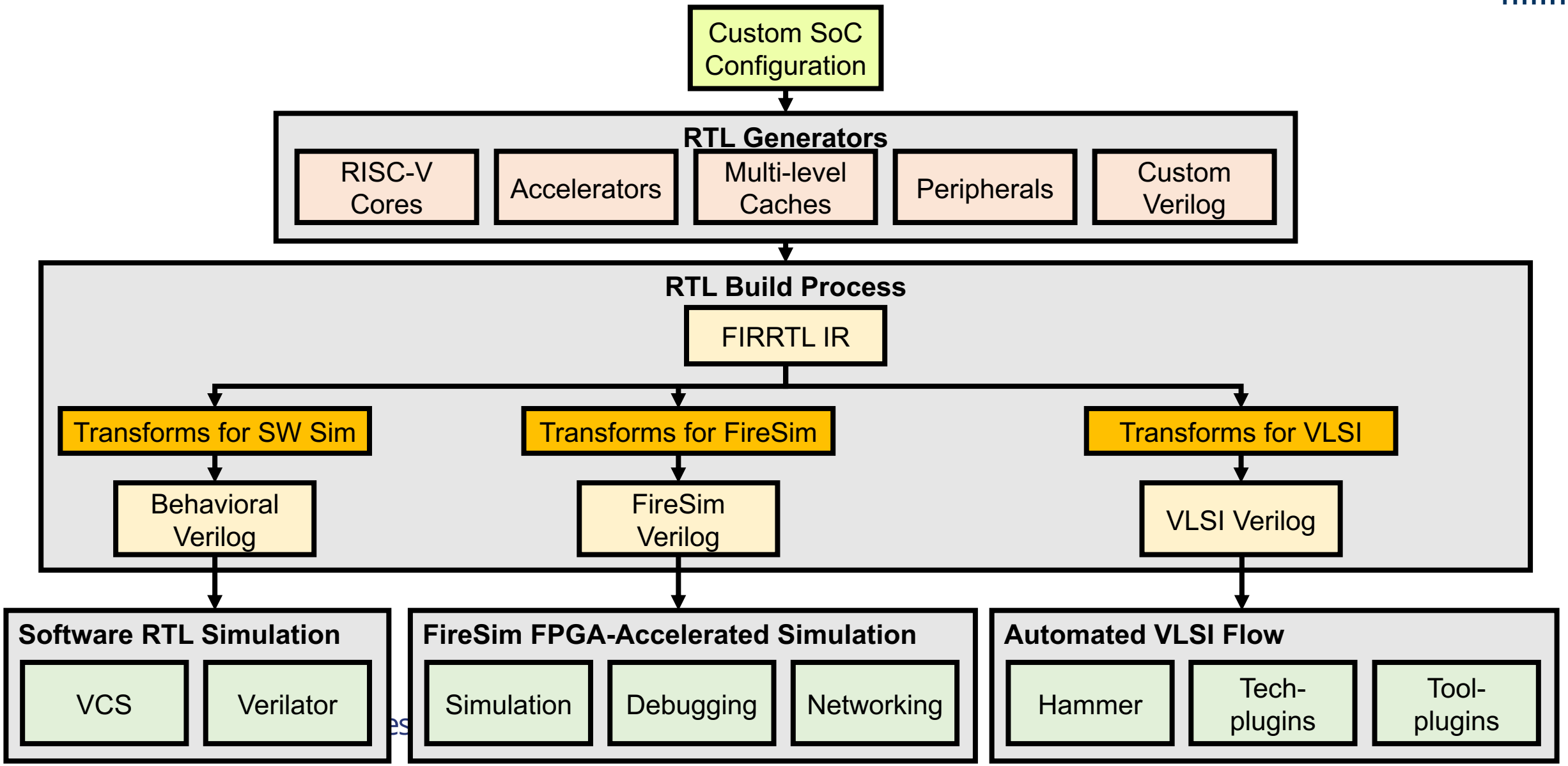


Hammer

- Modular VLSI flow
  - Allow reusability
  - Allow for multiple “small” experts instead of a single “super” expert
  - Build abstractions/APIs on top
  - Improve portability
  - Improve hierarchical partitioning
- Three categories of flow input
  - Design-specific
  - Tool/Vendor-specific
  - Technology-specific



Chipyard Unified Flows

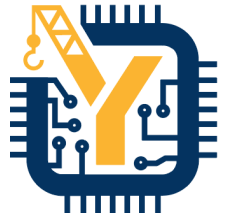


Chipyard Tooling

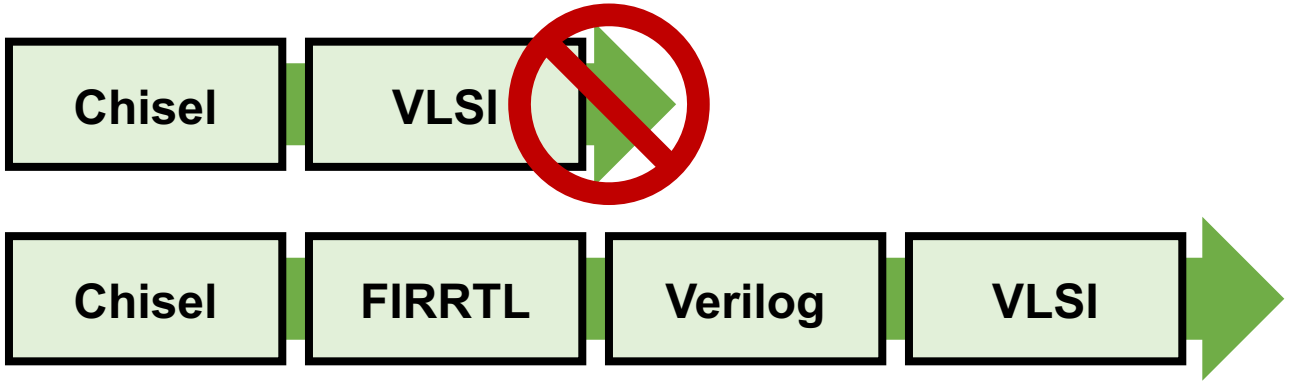
Berkeley
Architecture
Research

CHIPYARD

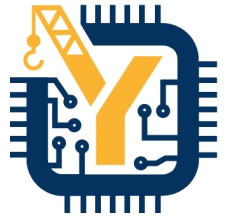
Chisel



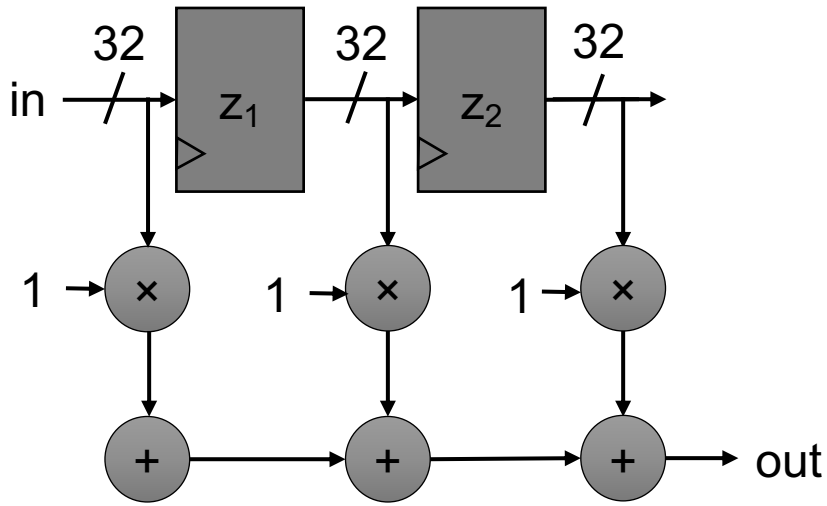
- Chisel – Hardware Construction Language built on Scala
- What Chisel **IS NOT:**
  - **NOT** Scala-to-gates
  - **NOT** HLS
  - **NOT** tool-oriented language
- What Chisel **IS:**
  - Productive language for **generating** hardware
  - Leverage **OOP/Functional programming** paradigms
  - Enables design of **parameterized generators**
  - **Designer-friendly**: low barrier-to-entry, high reward
  - **Backwards-compatible:** integrates with Verilog black-boxes



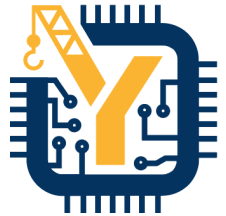
Chisel Example



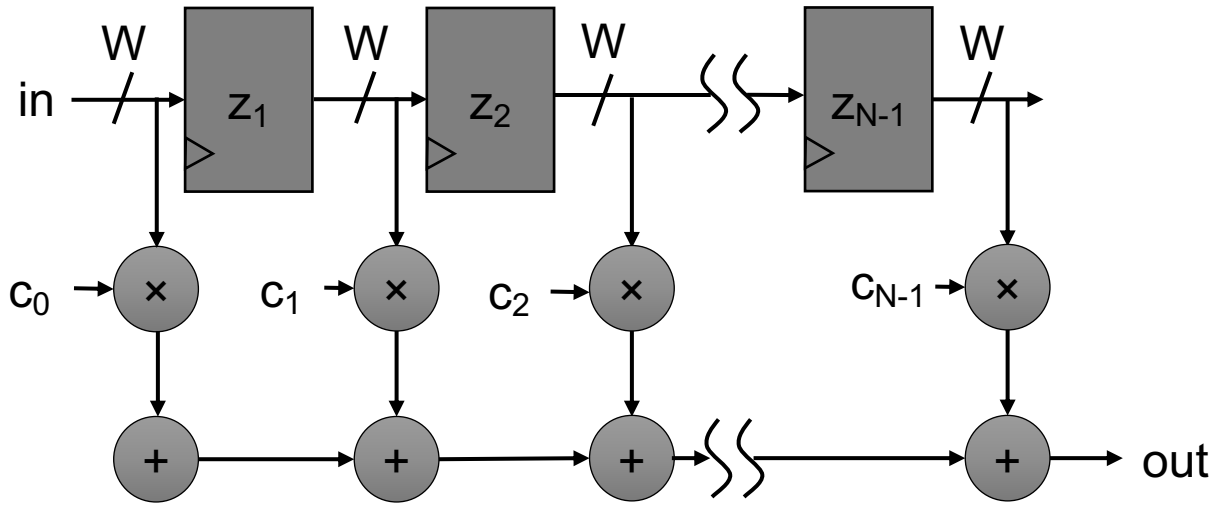
```
// 3-point moving average implemented in the
style of a FIR filter
class MovingAverage3 extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(32.W))
    val out = Output(UInt(32.W))
  })
  val z1 = RegNext(io.in)
  val z2 = RegNext(z1)

  io.out := io.in + z1 + z2
}
```



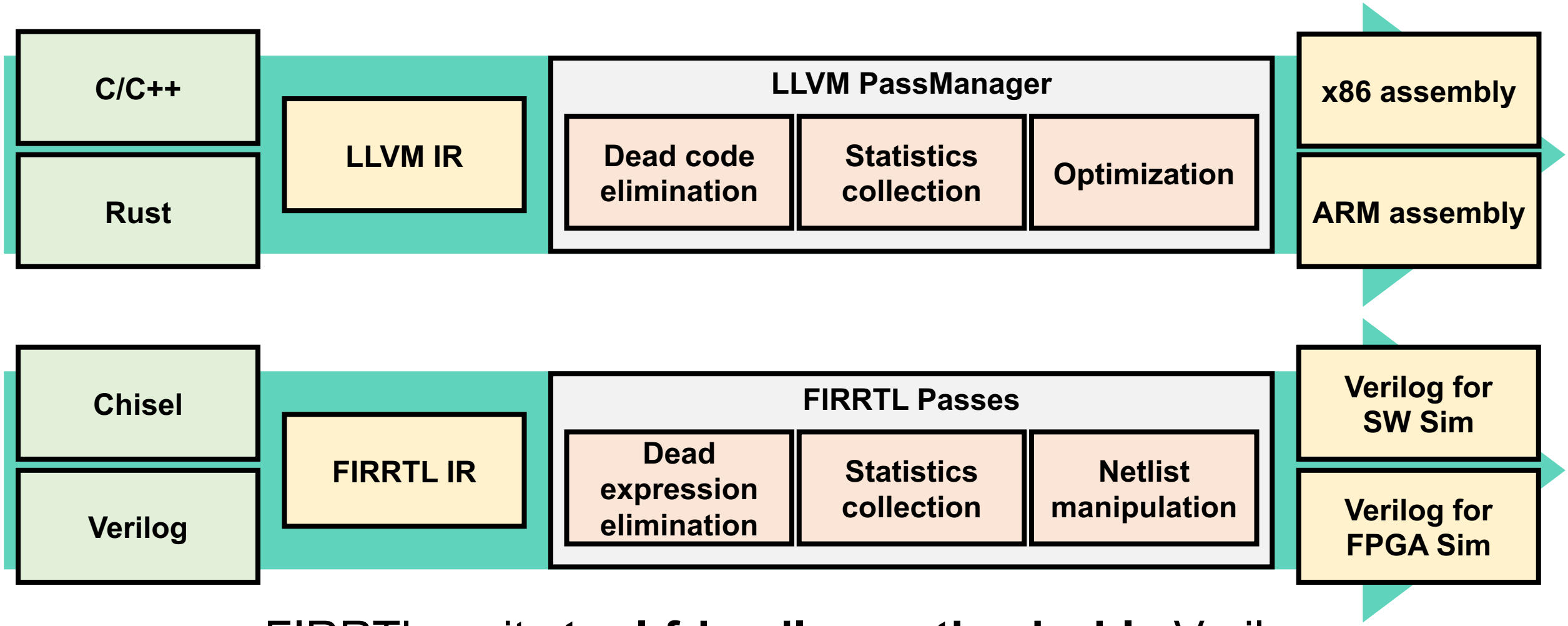
Chisel Example



```
// Generalized FIR filter parameterized by coefficients
class FirFilter(bitWidth: Int, coeffs: Seq[Int]) extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(bitWidth.W))
    val out = Output(UInt(bitWidth.W))
  })
  val zs = Wire(Vec(coeffs.length, UInt(bitWidth.W)))
  zs(0) := io.in
  for (i <- 1 until coeffs.length) {
    zs(i) := RegNext(zs(i-1))
  }
  val products = zs zip coeffs map {
    case (z, c) => z * c.U
  }
  io.out := products.reduce(_ + _)
}
```



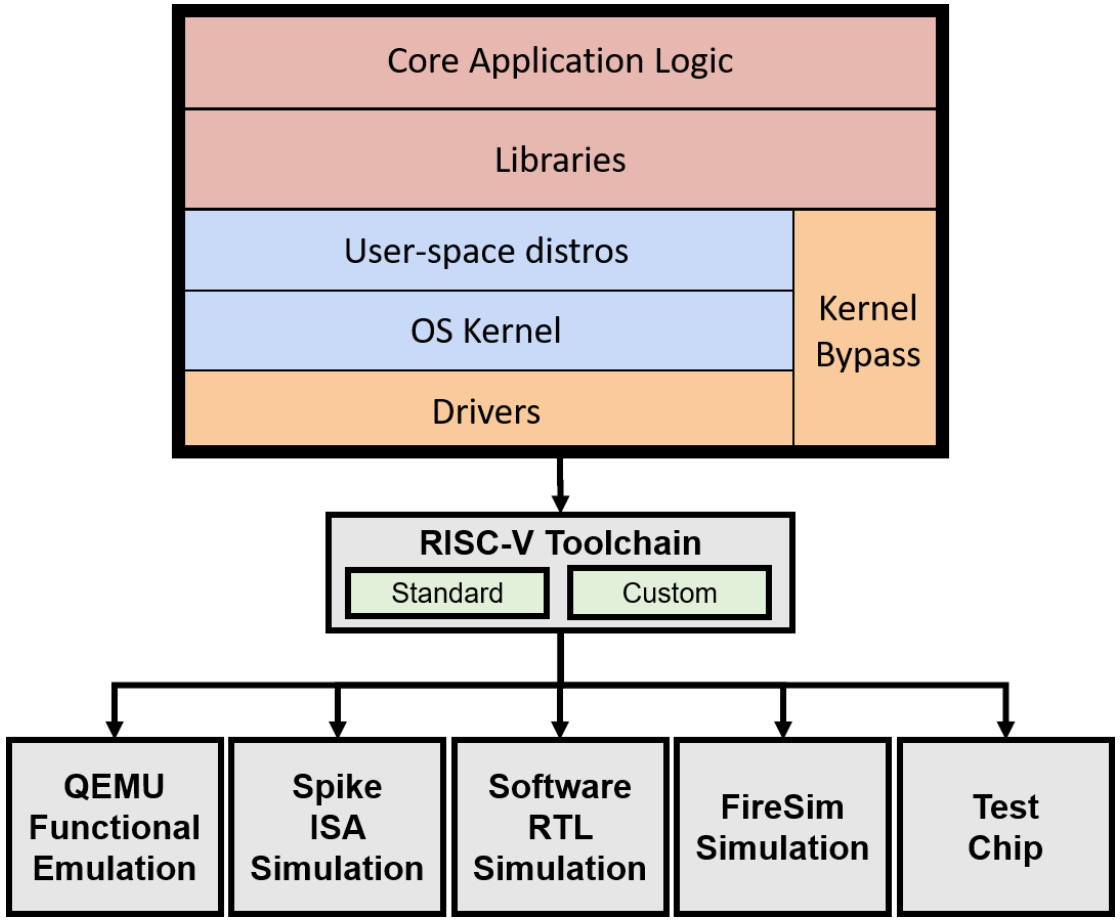
FIRRTL – LLVM for Hardware



FIRRTL emits **tool-friendly, synthesizable** Verilog

Software

- Compatible standard RISC-V Tools versions
- ESP-Tools as a non-standard equivalent SW tools package with custom accelerator extensions (Hwacha, Gemmini)
- Improved BareMetal testing flow
  - Use libgloss and newlib instead of in-house syscalls
- FireMarshal workload management

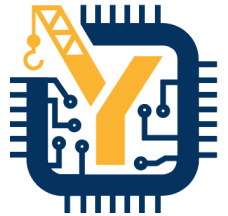


Rocket Chip Generators

Berkeley
Architecture
Research

CHIPYARD

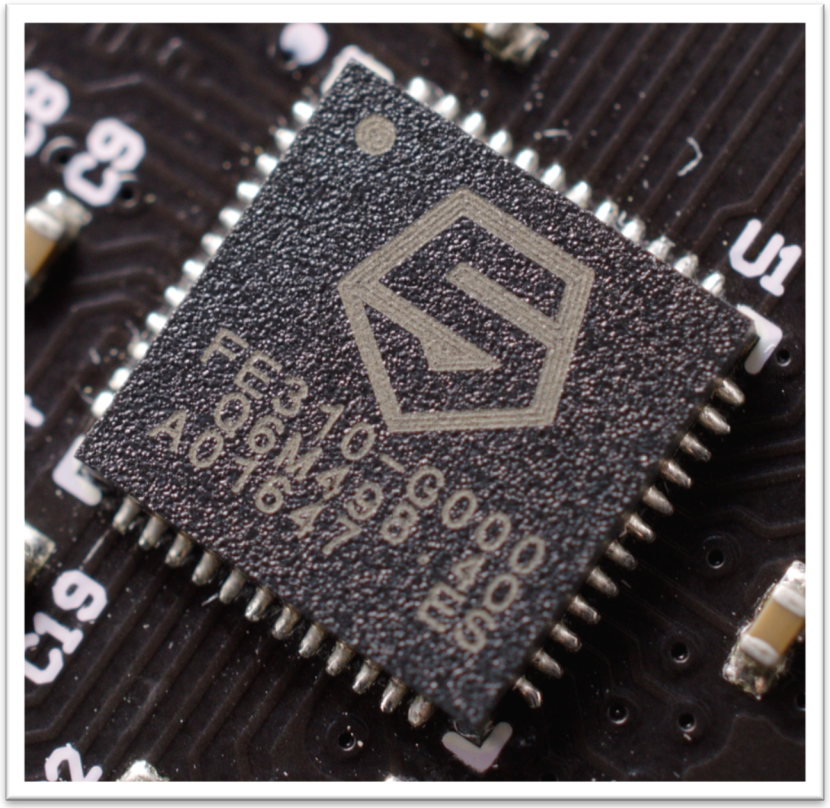
What is Rocket Chip?



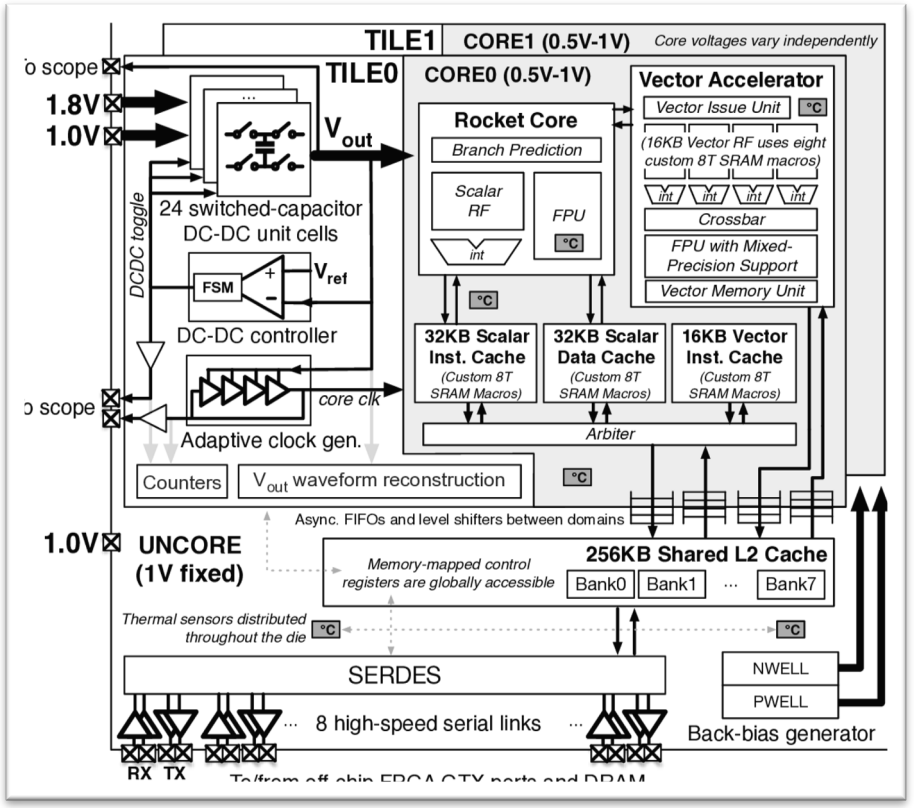
- A highly parameterizable and modular SoC generator
  - Replace default Rocket core w/ your own core
  - Add your own coprocessor
  - Add your own SoC IP to uncore
- A library of reusable SoC components
  - Memory protocol converters
  - Arbiters and Crossbar generators
  - Clock-crossings and asynchronous queues
- The largest open-source Chisel codebase
- Developed at UC Berkeley, now maintained by many
  - SiFive, CHIPS Alliance, UC Berkeley

Generating Varied SoCs

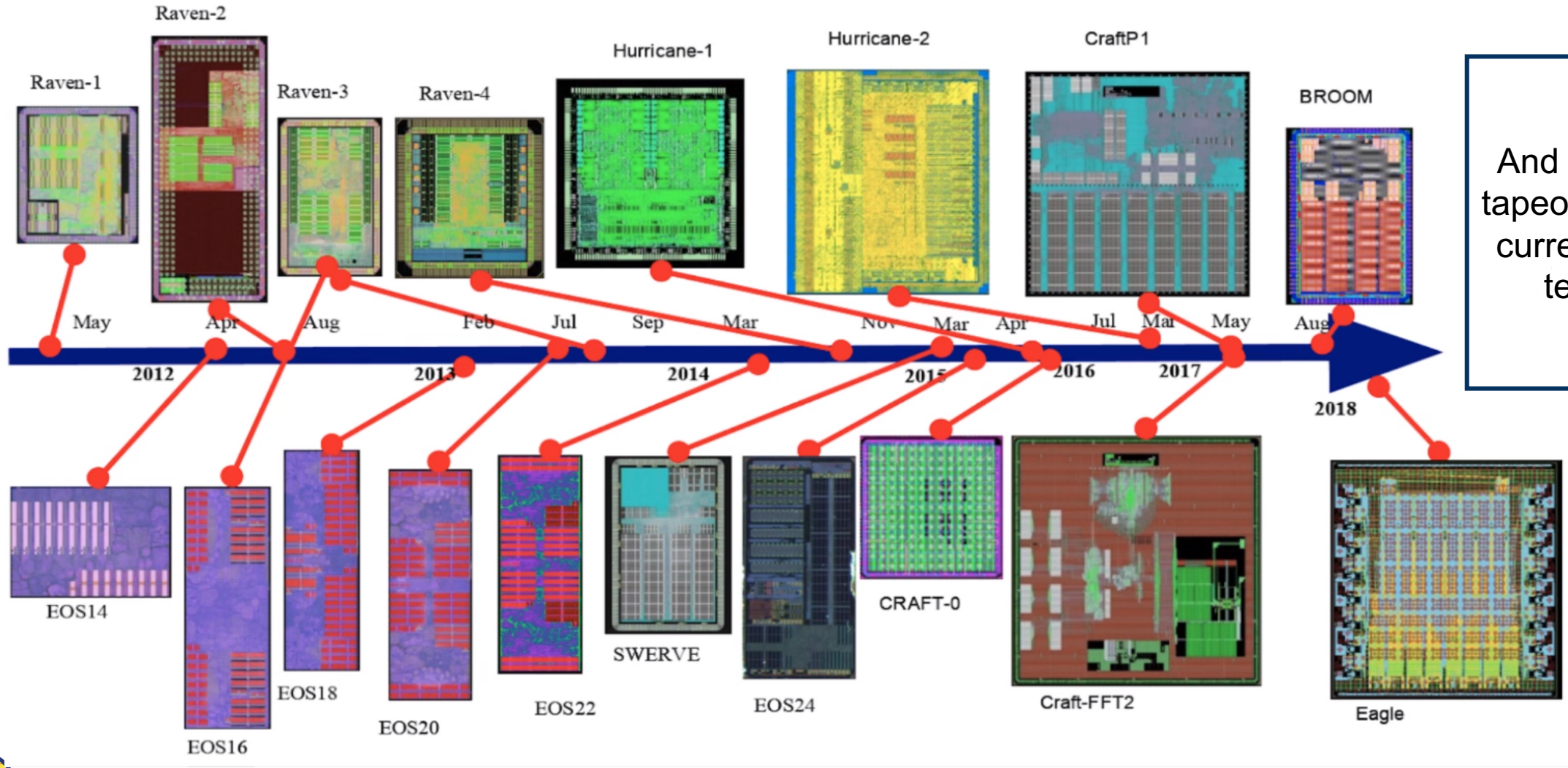
In industry: **SiFive Freedom E310**



In academia: **UCB Hurricane-1**

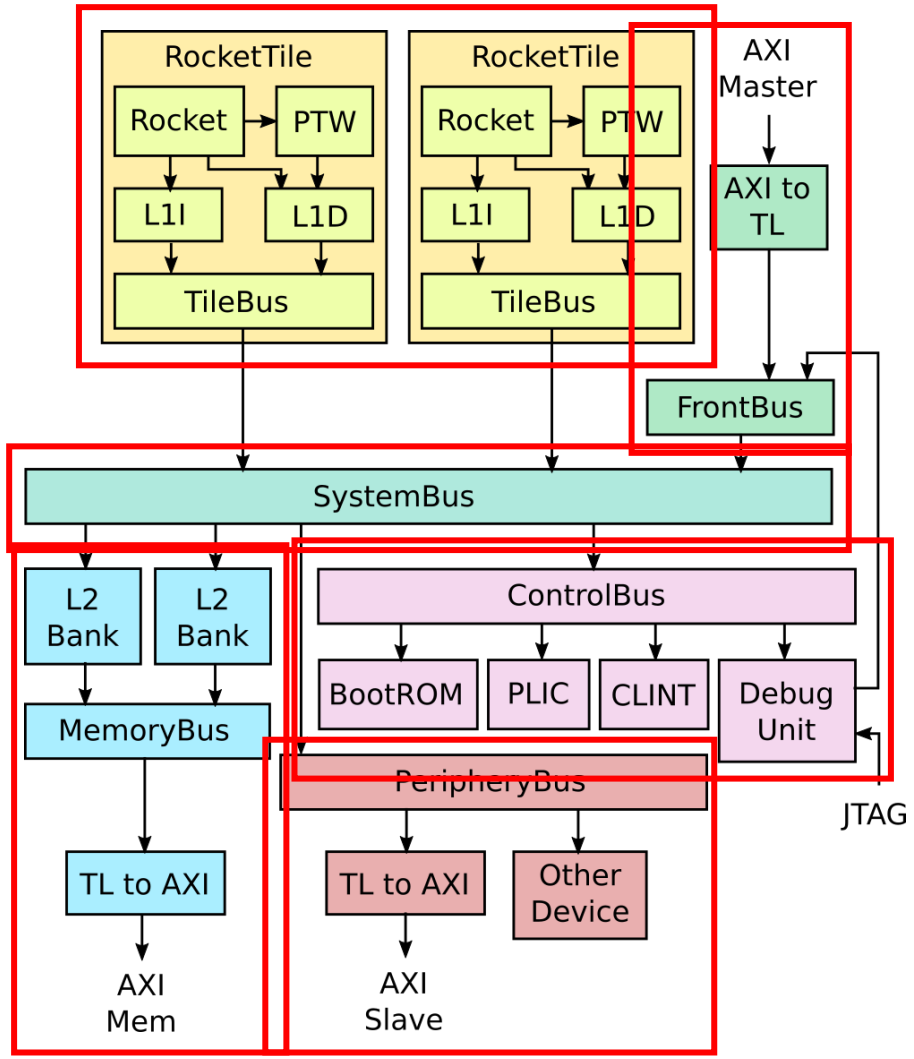


Used in Many Tapeouts



And many new tapeouts that are currently under testing…

Structure of a Rocket Chip SoC



Tiles: unit of replication for a core

- CPU
- L1 Caches
- Page-table walker

L2 banks:

- Receive memory requests

FrontBus:

- Connects to DMA devices

ControlBus:

- Connects to core-complex devices

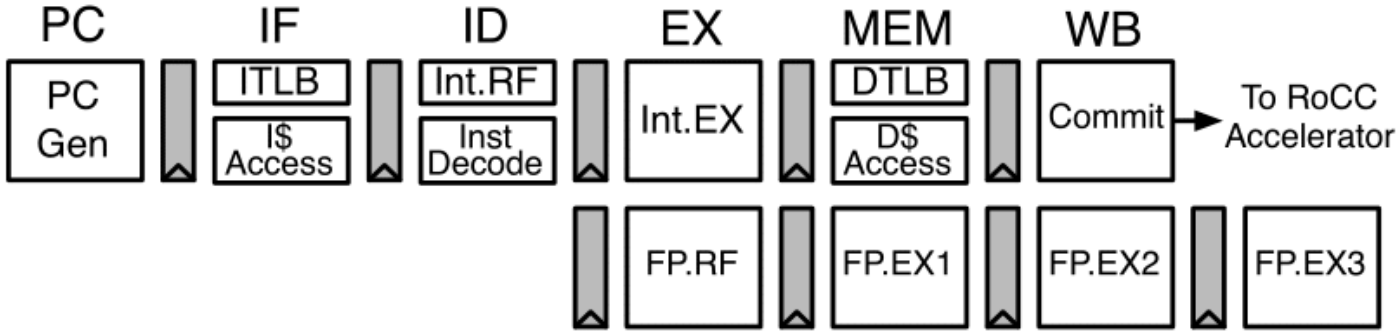
PeripheryBus:

- Connects to other devices

SystemBus:

- Ties everything together

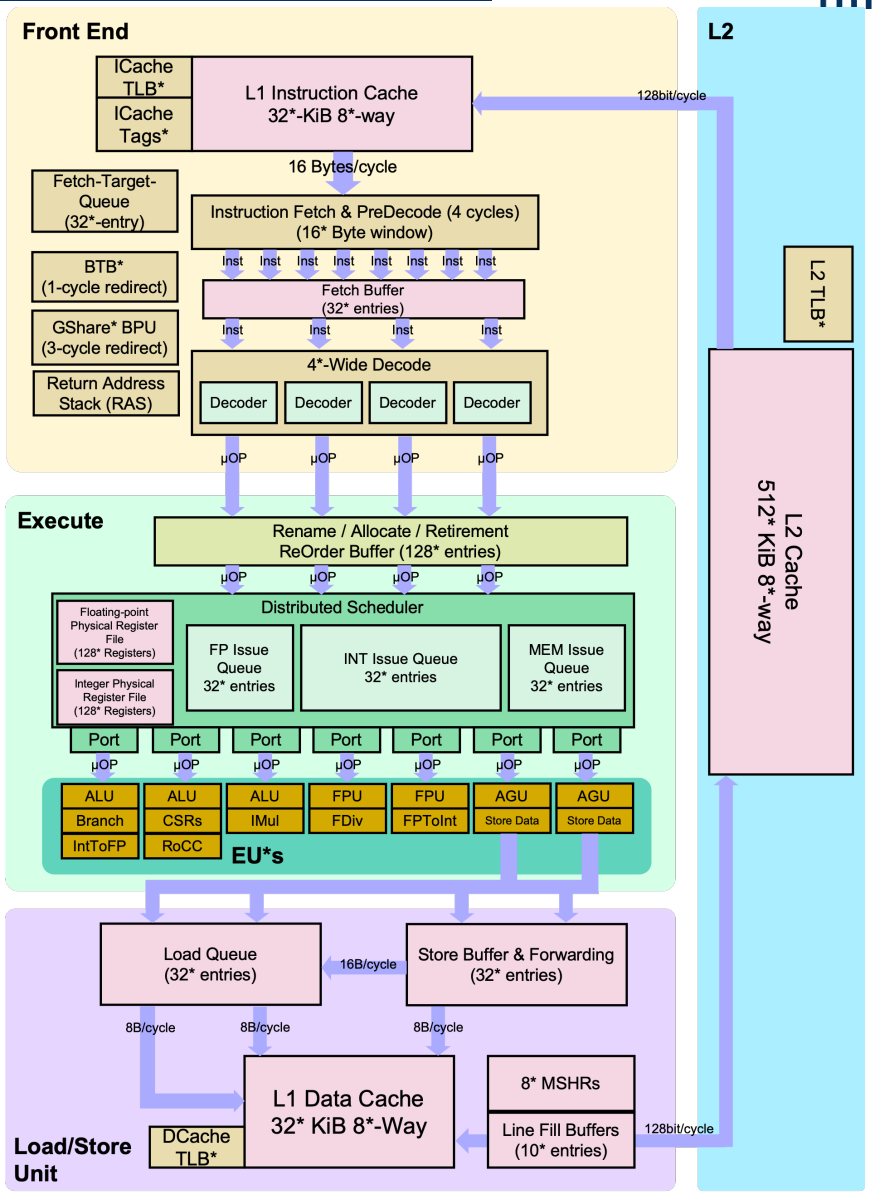
The Rocket In-Order Core



- First open-source RISC-V CPU
- In-order, single-issue RV64GC core
  - Floating-point via Berkeley hardfloat library
  - RISC-V Compressed
  - Physical Memory Protection (PMP) standard
  - Supervisor ISA and Virtual Memory
- Boots Linux
- Supports Rocket Chip Coprocessor (RoCC) interface
- L1 I$ and D$
  - Caches can be configured as scratchpads

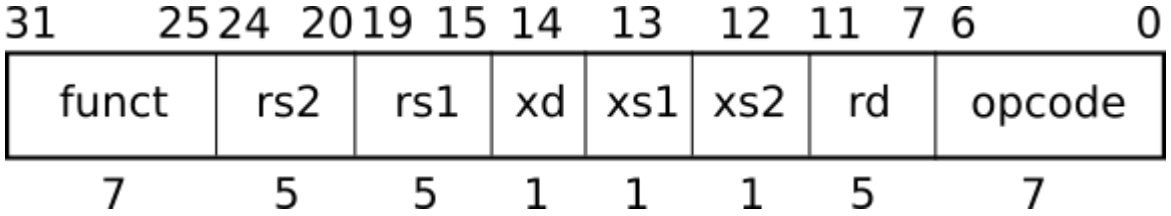
BOOM: The Berkeley Out-of-Order Machine

- Superscalar RISC-V OoO core
- Fully integrated in Rocket Chip ecosystem
- Open-source and described in Chisel
- Parameterizable generator
- Taped-out (BROOM at HC18, BEAGLE ’21)
- Full RV64GC ISA support
  - FP, RVC, Atomics, PMPs, VM, Breakpoints, RoCC
  - Runs real OS’s, software
- Drop-in replacement for Rocket

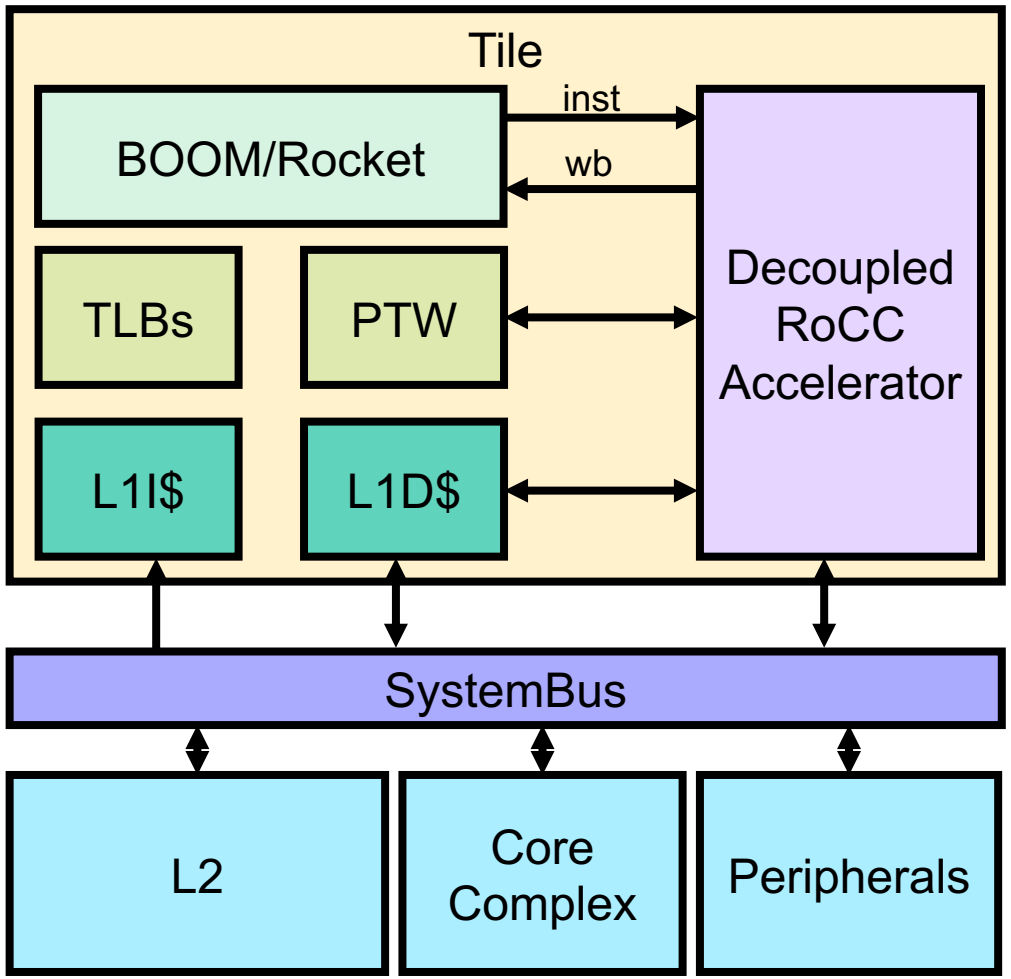


RoCC Accelerators

- **RoCC:** Rocket Chip Coprocessor
- Execute custom RISC-V instructions for a custom extension

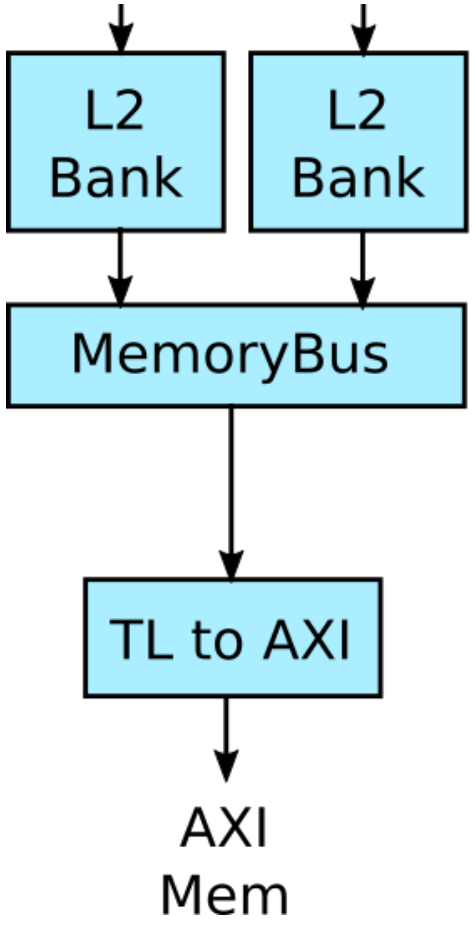


- Examples of RoCC accelerators
  - Vector accelerators
  - Memcpy accelerator
  - Machine-learning accelerators
  - Java GC accelerator

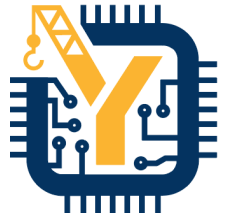


L2 Cache and Memory System

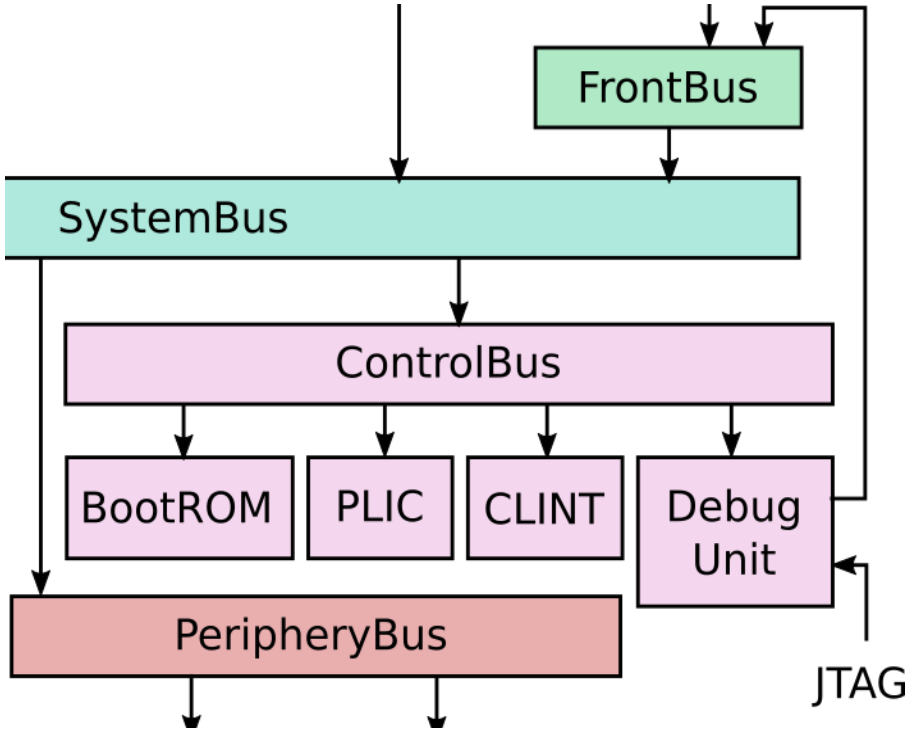
- Multi-bank shared L2
  - SiFive's open-source IP
  - Fully coherent
  - Configurable size, associativity
  - Supports atomics, prefetch hints
- Non-caching L2 Broadcast Hub
  - Coherence w/o caching
  - Bufferless design
- Multi-channel memory system
  - Conversion to AXI4 for compatible DRAM controllers



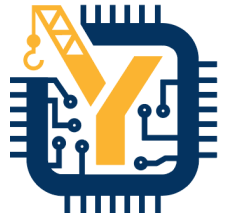
Core Complex Devices



- BootROM
  - First-stage bootloader
  - DeviceTree
- PLIC
- CLINT
  - Software interrupts
  - Timer interrupts
- Debug Unit
  - DMI
  - JTAG

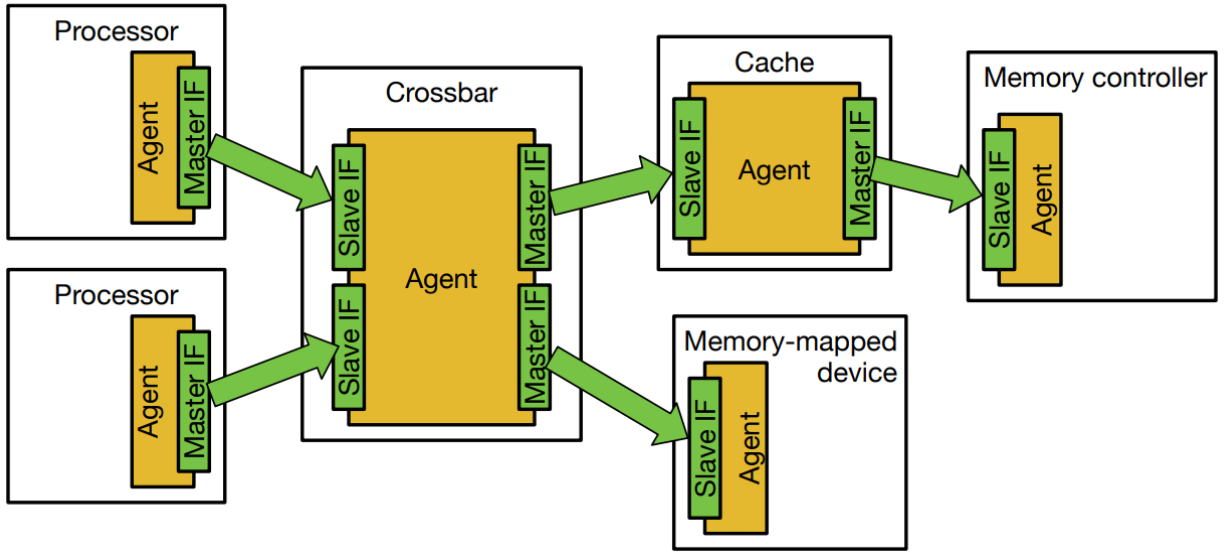


Other Chipyard Blocks



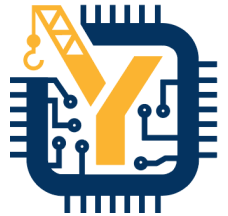
- **Hardfloat:** Parameterized Chisel generators for hardware floating-point units
- **IceNet:** Custom NIC for FireSim simulations
- **SiFive-Blocks:** Open-sourced Chisel peripherals
  - GPIO, SPI, UART, etc.
- **TestchipIP:** Berkeley utilities for chip testing/bringup
  - Tethered serial interface
  - Simulated block device
- **Hwacha:** Decoupled vector-fetch RoCC accelerator
- **Gemmini:** Systolic-array matrix multiplication RoCC accelerator
- **SHA3:** Educational SHA3 RoCC accelerator

TileLink Interconnect



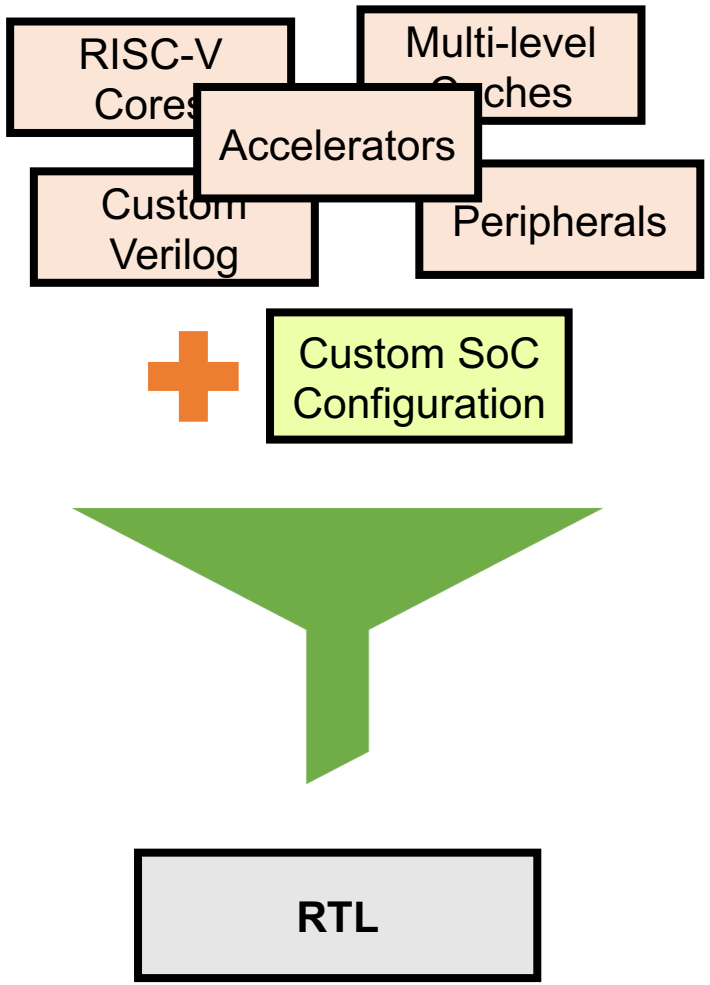
- Free and open chip-scale interconnect standard
- Supports multiprocessors, coprocessors, accelerators, DMA, peripherals, etc.
- Provides a physically addressed, shared-memory system
- Supports cache-coherent shared memory, MOESI-equivalent protocol
- Verifiable deadlock freedom for conforming SoCs

TileLink Interconnect



- Three different protocol levels with increasing complexity
  - TL-UL (Uncached Lightweight)
  - TL-UH (Uncached Heavyweight)
  - TL-C (Cached)
- Rocket Chip provides library of reusable TileLink widgets
  - Conversion to/from AXI4, AHB, APB
  - Conversion among TL-UL, TL-UH, TL-C
  - Crossbar generator
  - Width / logical size converters
  - TLMonitor conformance checker

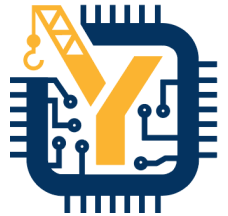
Integration



Challenges of configuring generators

- How do we allow parameterization of interdependent generators?
  - The memory system is a function of the number of cores, cache size, DRAM size, MMIO addresses, etc.
  - **Solution: Diplomatic parameter negotiation**
- How do we flexibly and reusably describe a system's parameterization?
  - Enable rapid design-space exploration
  - **Solution: Context-dependent parameterization (aka Rocketchip Configs)**

Diplomacy



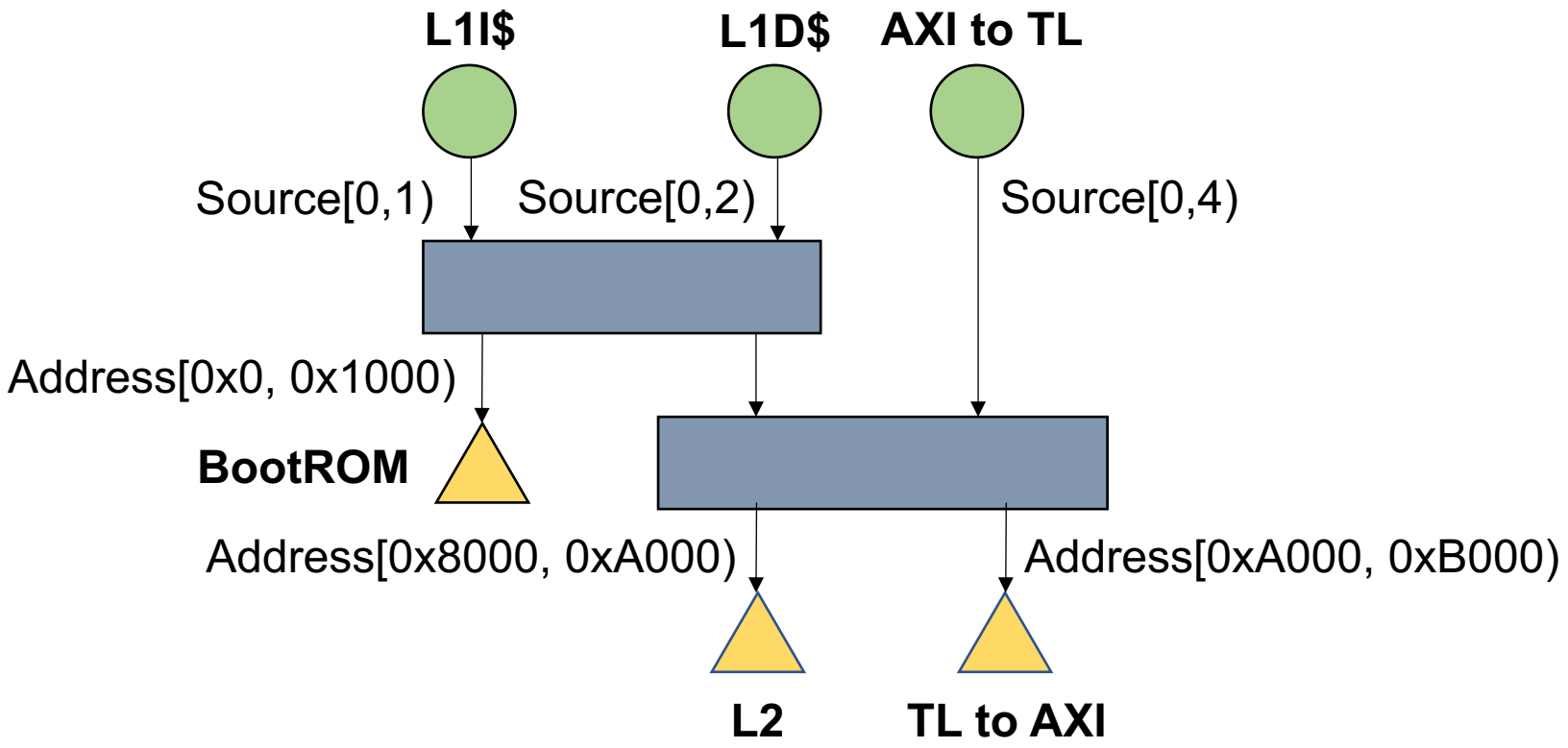
Problem: Interconnects are difficult to parameterize correctly

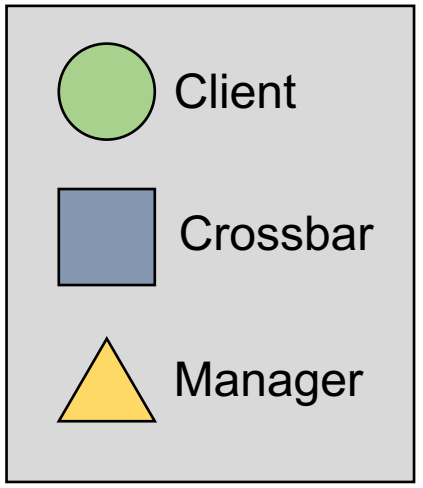
- Complex interconnect graph with many nodes
- Nodes are independently parameterized (but are interdependent)

Diplomacy: Framework for negotiating parameters between Chisel generators

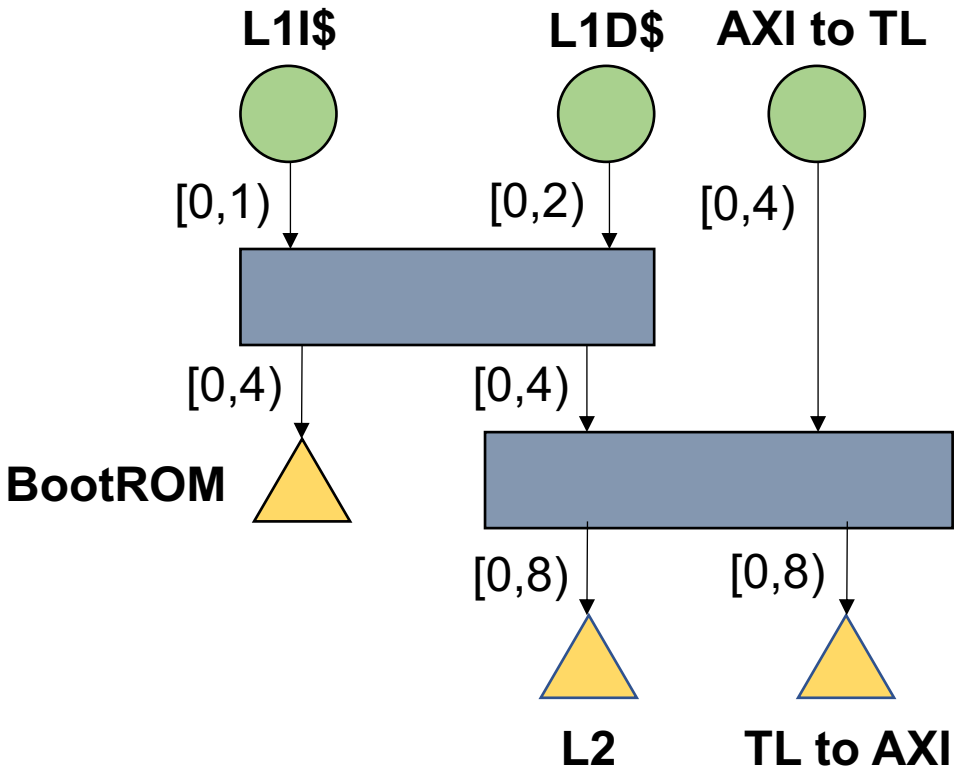
- Graphical abstraction of interconnectivity
- Diplomatic lazy modules follow two-phase elaboration
  - **Phase one**: nodes exchange configuration information with each other and decide final parameters
  - **Phase two**: Chisel RTL elaborates using calculated parameters
- Used extensively by RocketChip TileLink generators

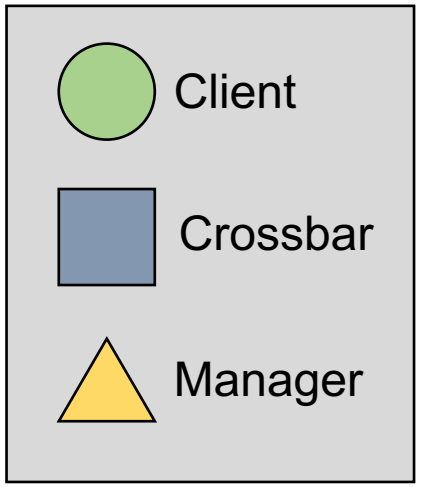
Diplomacy Example



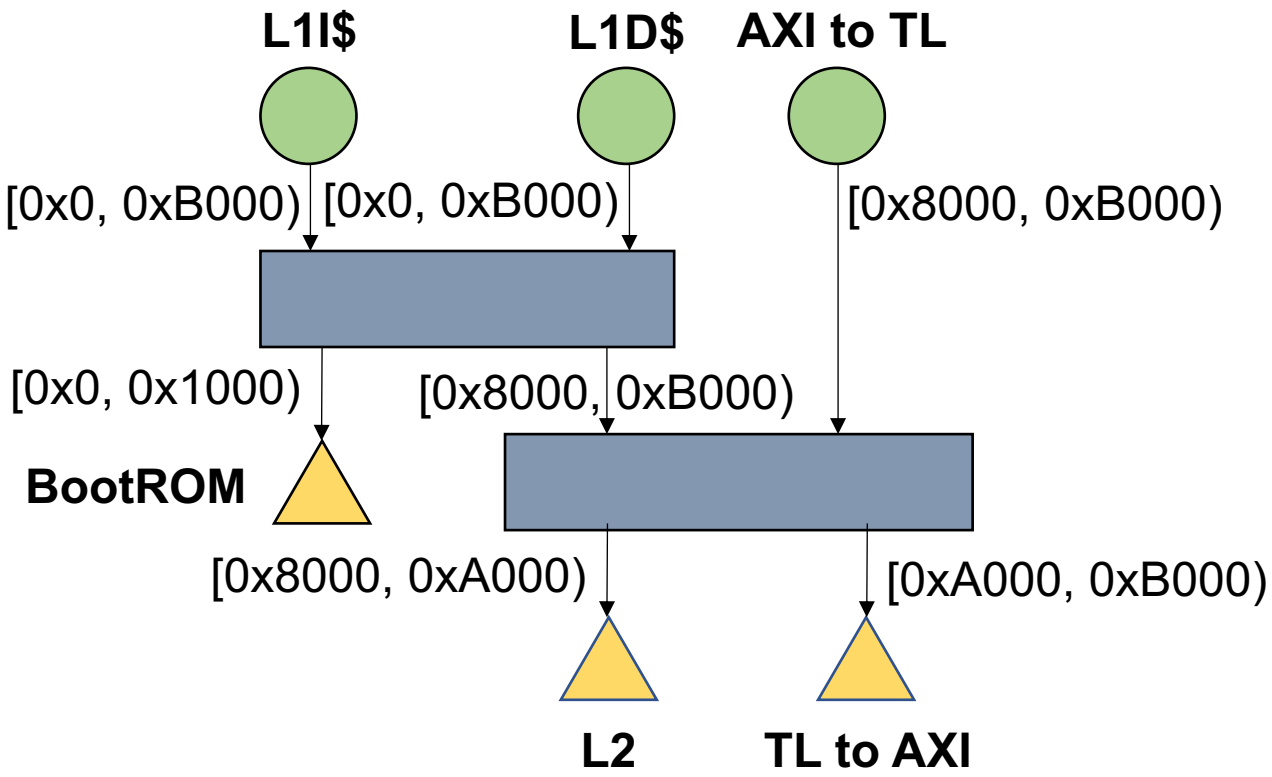


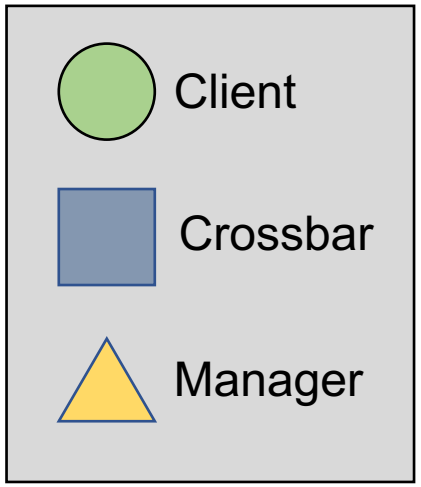
Diplomacy Example



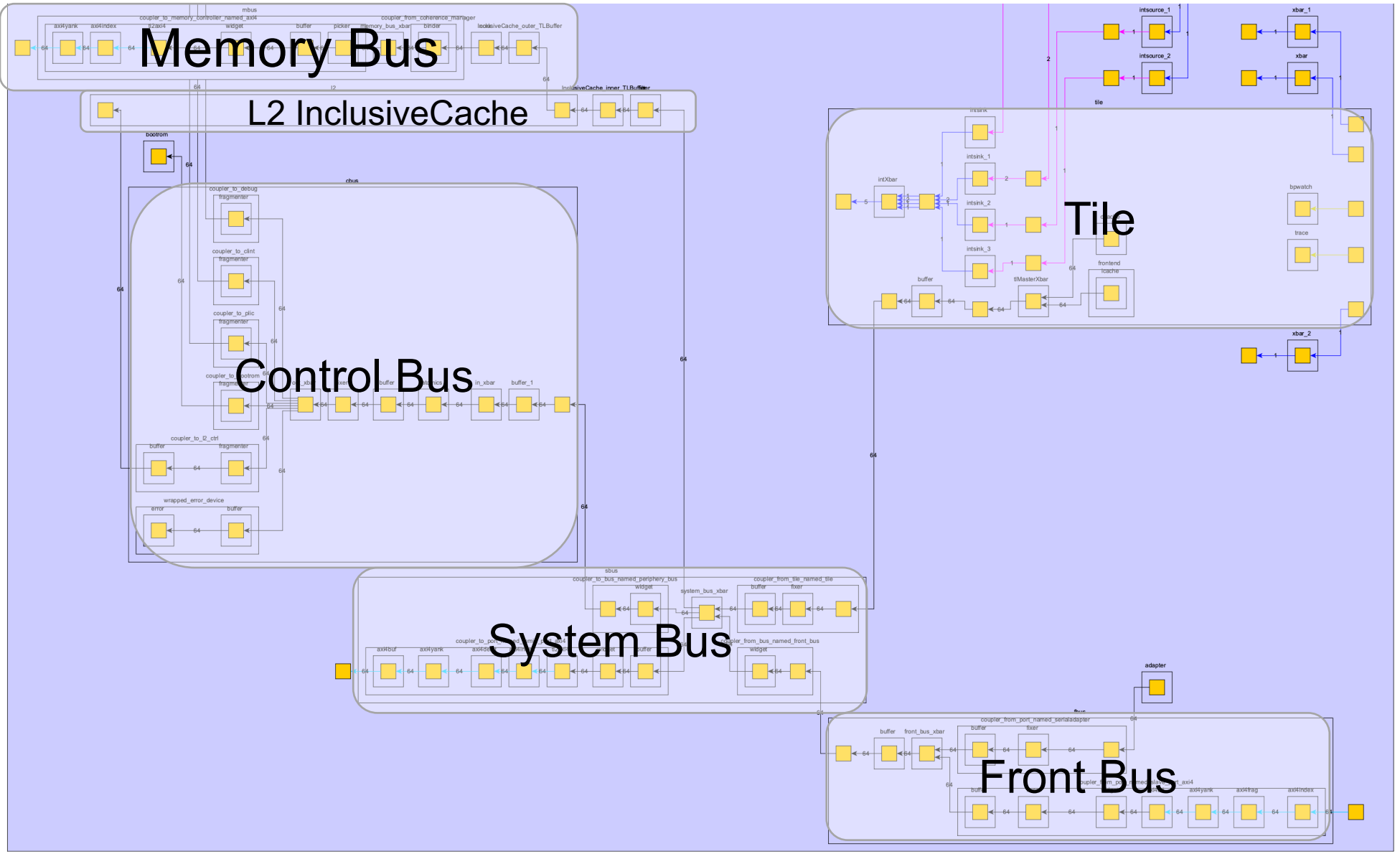


Diplomacy Example

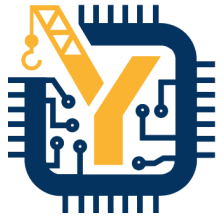




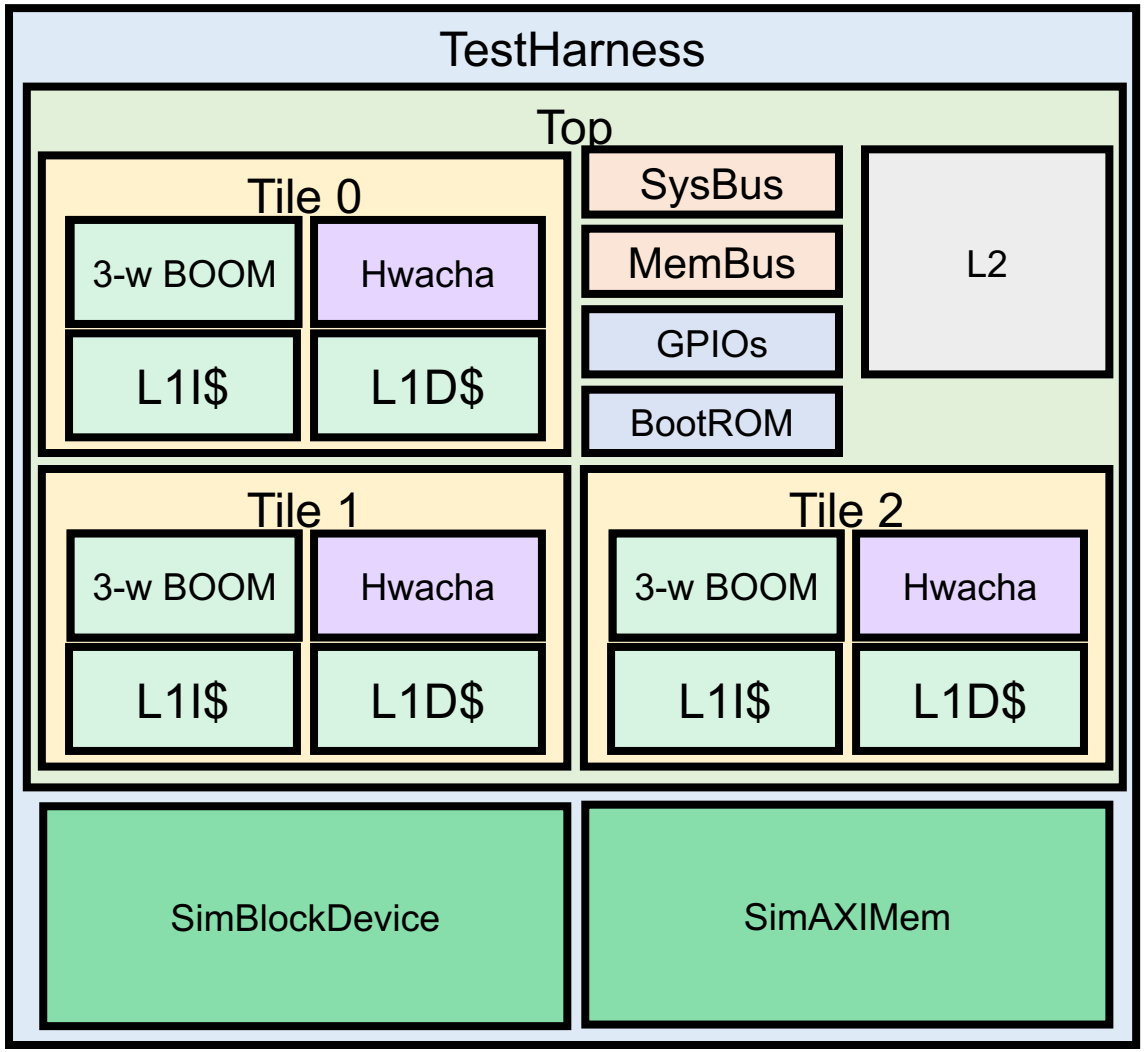
Diplomacy-generated Graph



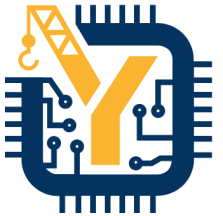
Rocket Chip Configuration



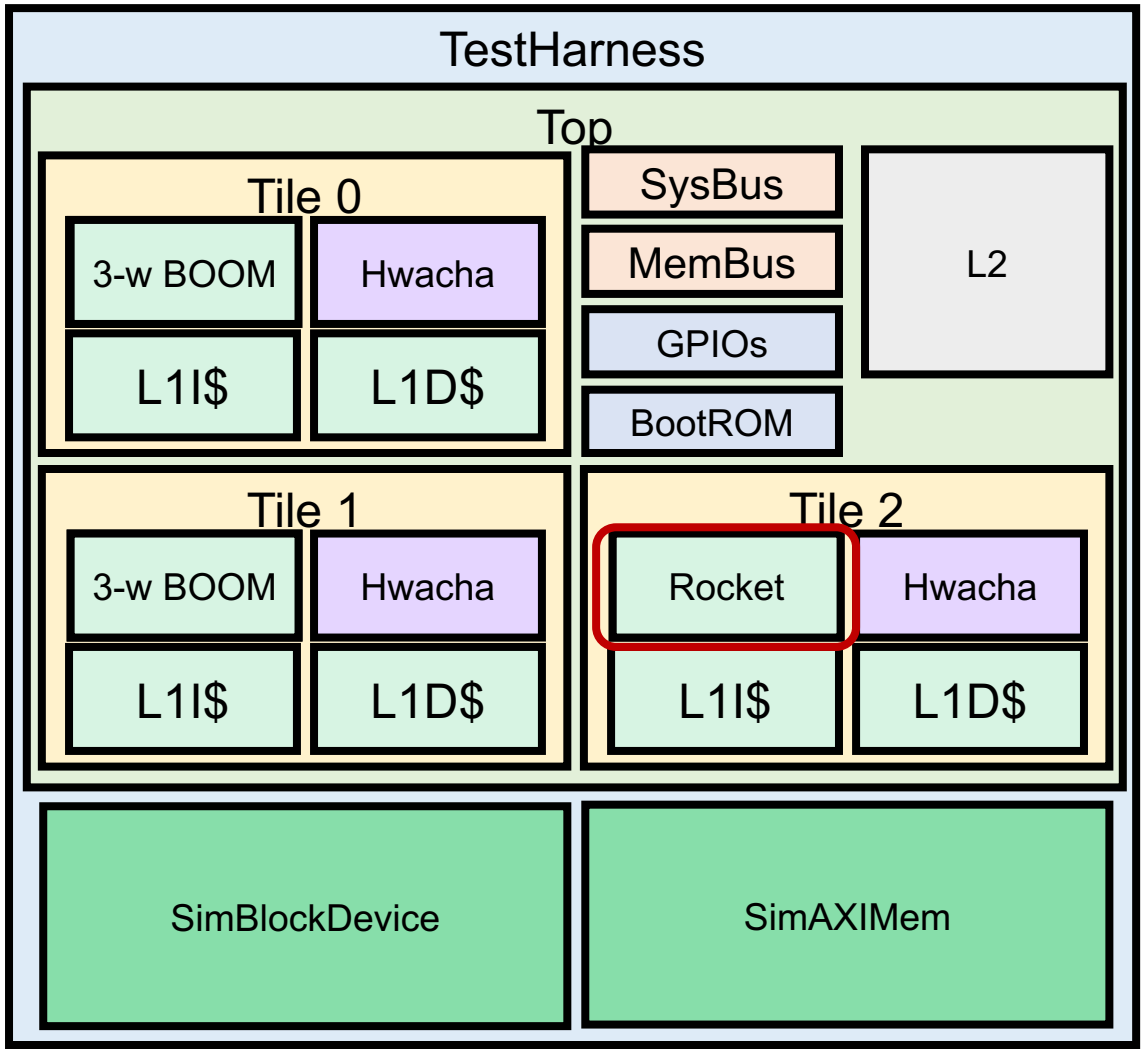
```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new hwacha.DefaultHwachaConfig            ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(3)         ++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```



Rocket Chip Configuration

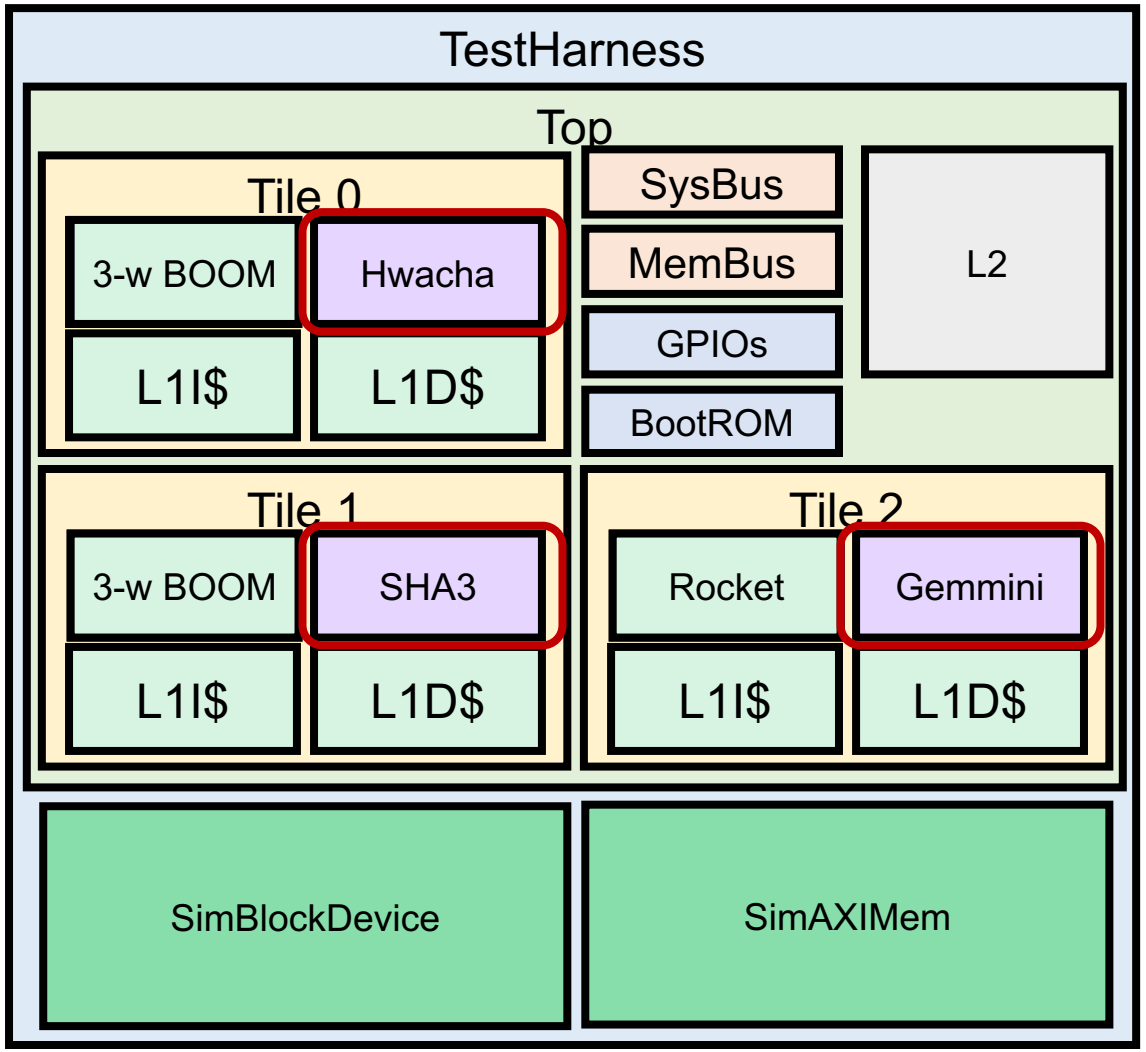


```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new hwacha.DefaultHwachaConfig            ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(2)         ++
  new rocketchip.subsystem.WithNBigCores(1)++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```



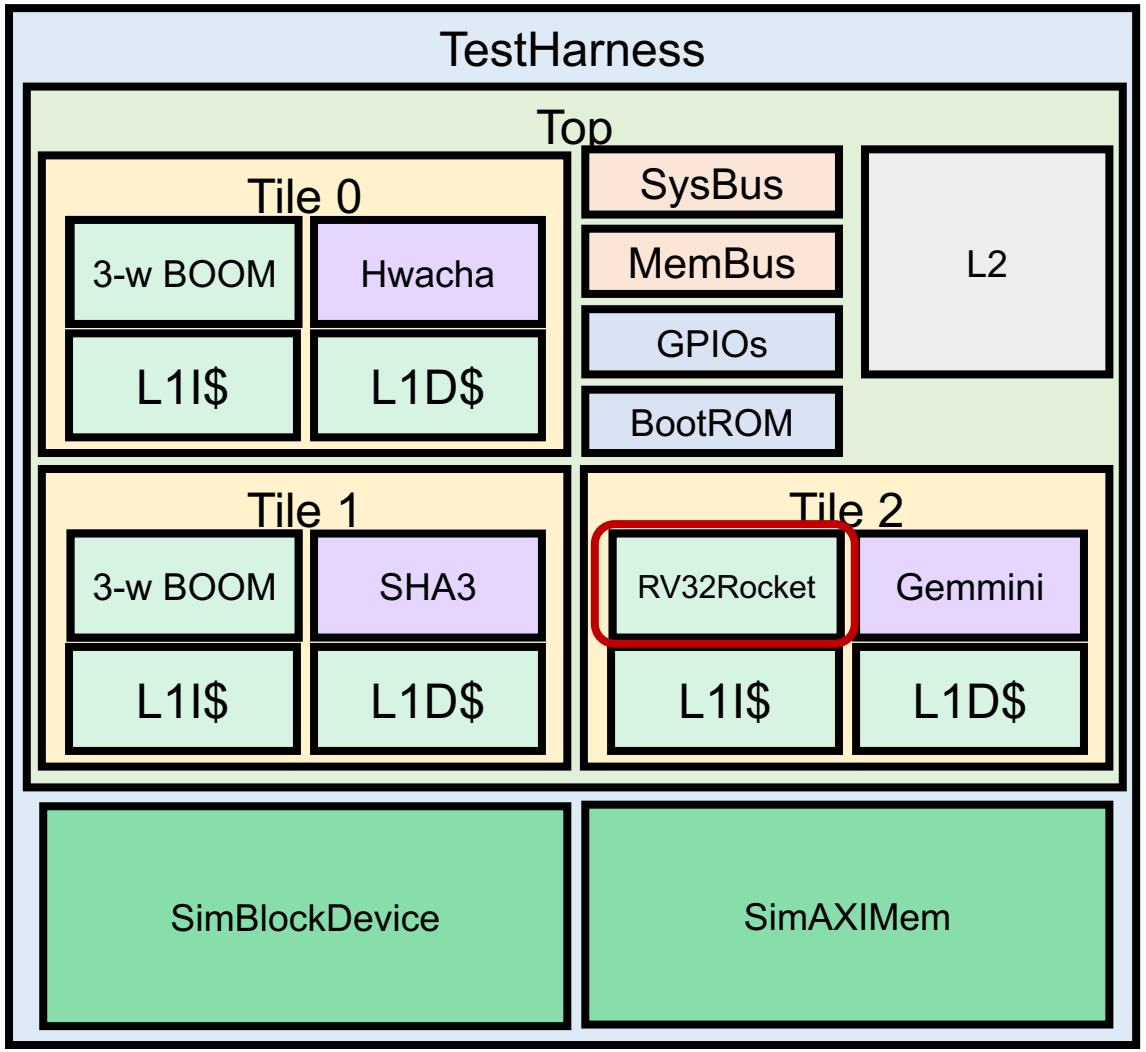
Rocket Chip Configuration

```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new WithMultiRoCCGemmini(2)               ++
  new WithMultiRoCCSha3(1)                  ++
  new WithMultiRoCCHwacha(0)                ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(2)         ++
  new rocketchip.subsystem.WithNBigCores(1)++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```



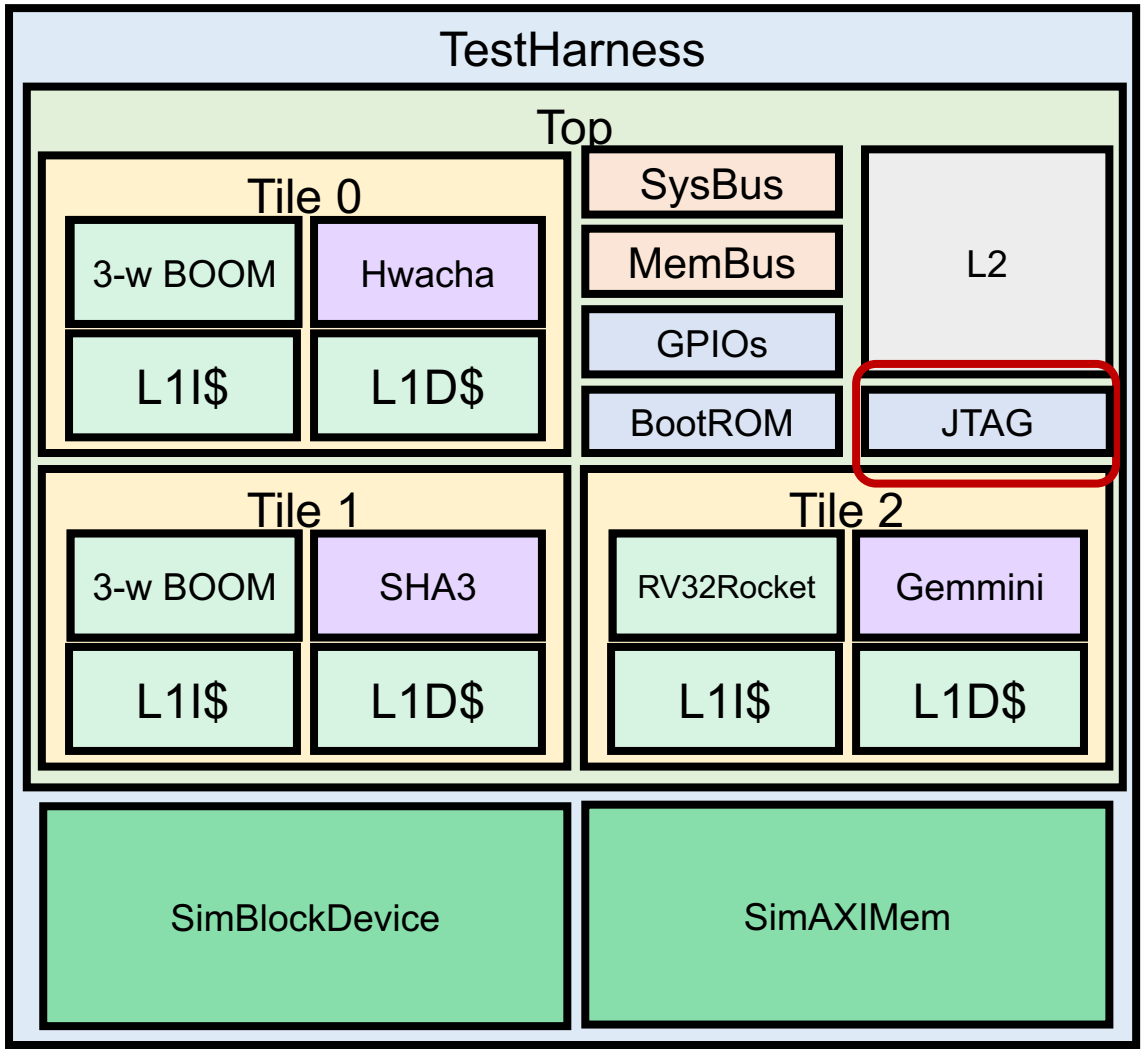
Rocket Chip Configuration

```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new WithMultiRoCCGemmini(2)               ++
  new WithMultiRoCCSha3(1)                  ++
  new WithMultiRoCCHwacha(0)                ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(2)         ++
  new rocketchip.subsystem.WithRV32         ++
  new rocketchip.subsystem.WithNBigCores(1)++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```

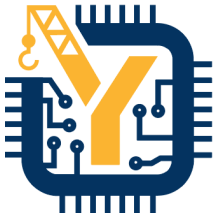


Rocket Chip Configuration

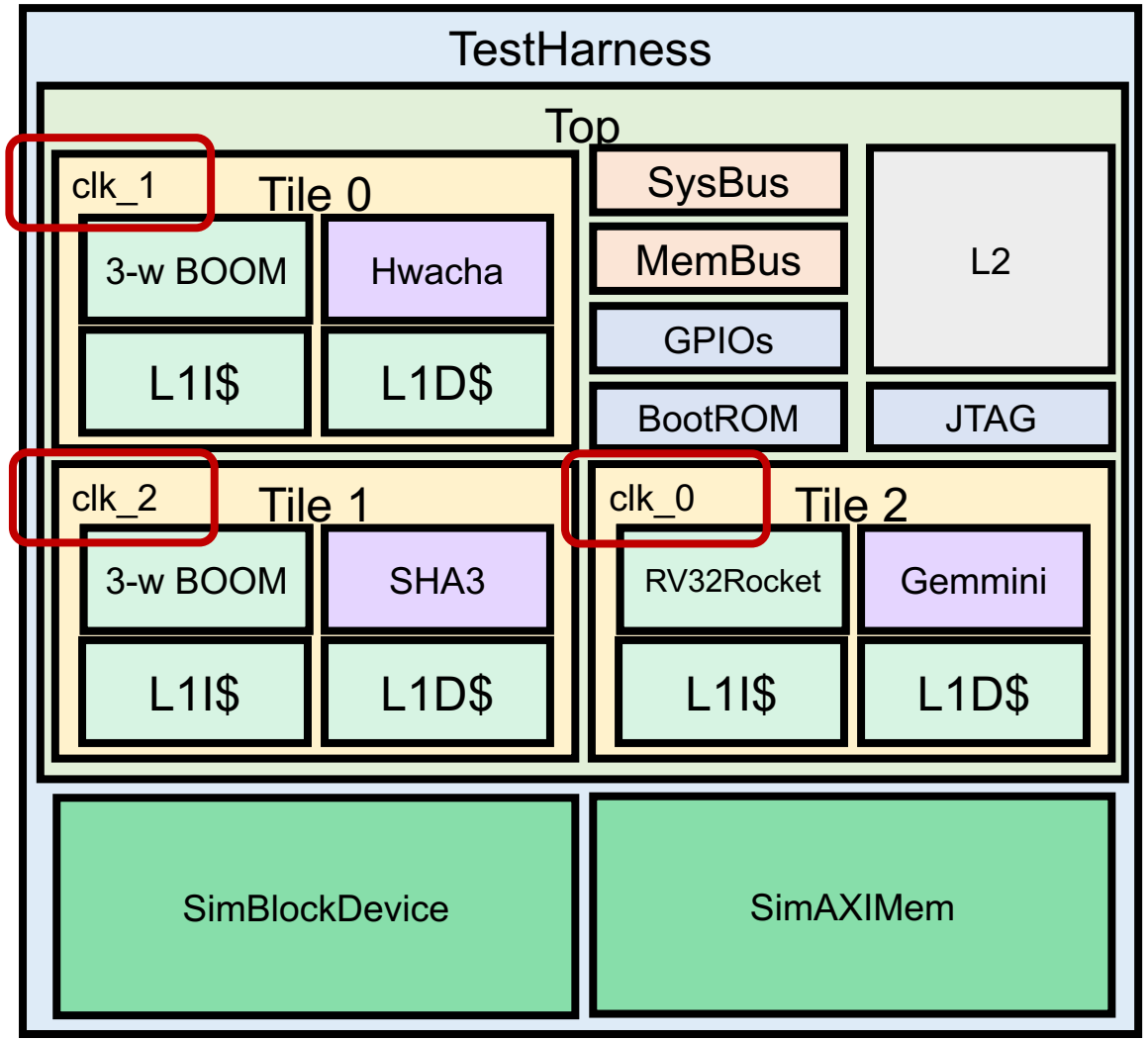
```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithJtagDTM                           ++
  new WithBootROM                           ++
  new WithMultiRoCCGemmini(2)               ++
  new WithMultiRoCCSha3(1)                  ++
  new WithMultiRoCCHwacha(0)                ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(2)         ++
  new rocketchip.subsystem.WithRV32         ++
  new rocketchip.subsystem.WithNBigCores(1)++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```



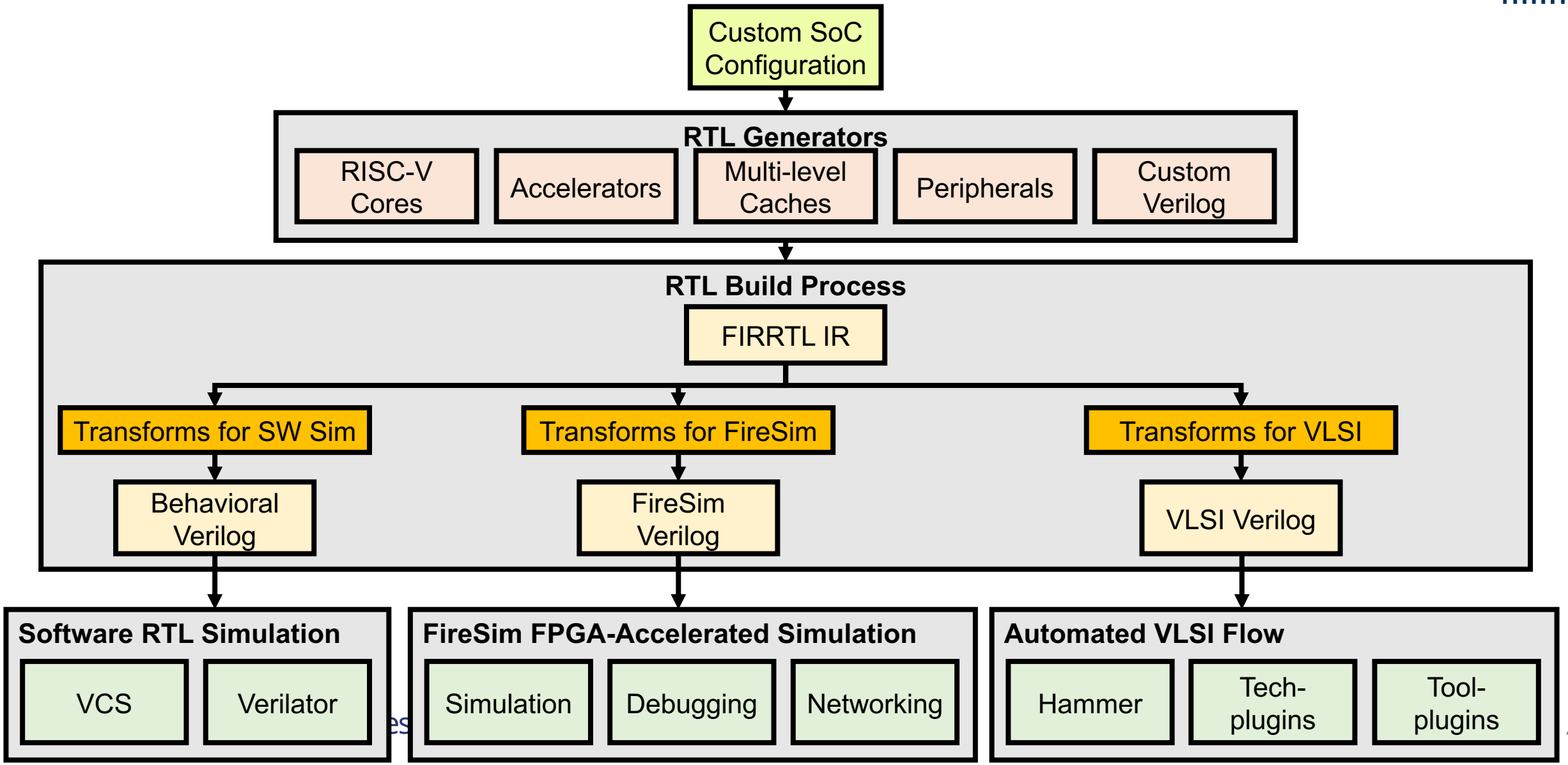
Rocket Chip Configuration



```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithJtagDTM                           ++
  new WithBootROM                           ++
  new WithRationalBoomTiles                 ++
  new WithRationalRocketTiles               ++
  new WithMultiRoCCGemmini(2)               ++
  new WithMultiRoCCSha3(1)                  ++
  new WithMultiRoCCHwacha(0)                ++
  new WithInclusiveCache(capacityKB=1024)   ++
  new boom.common.WithLargeBooms            ++
  new boom.system.WithNBoomCores(2)         ++
  new rocketchip.subsystem.WithRV32         ++
  new rocketchip.subsystem.WithNBigCores(1)++
  new WithNormalBoomRocketTop               ++
  new rocketchip.system.BaseConfig)
```



Chipyard Unified Flows



FireSim

Scalable FPGA-Accelerated Cycle-Accurate Hardware Simulation in the Cloud

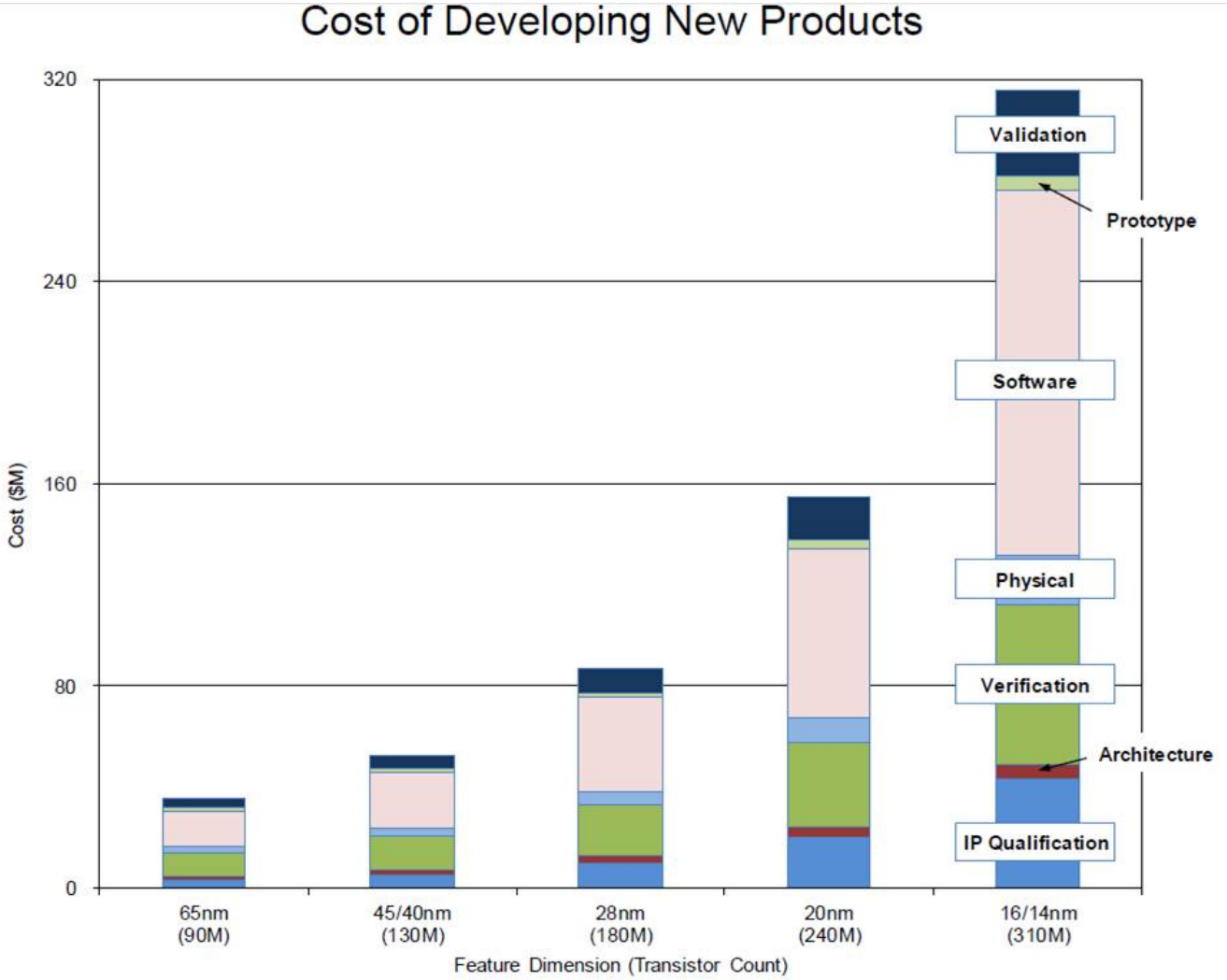
https://fires.im

@firesimproject

Sagar Karandikar, David Biancolin, Howard Mao, Alon Amid, Nathan Pemberton, Albert Magyar, Albert Ou, Randy Katz, Borivoje Nikolić, Jonathan Bachrach, Krste Asanović

Berkeley Architecture Research

The Non Recurring Engineering (NRE) Cost Barrier



Source: IBS

- NRE is a huge barrier to building chips
- Many sources; death by a thousand cuts

→ Requires a large joint effort

<u>FireSim's focus:</u>

Enable better *full-system* simulation for pre-silicon

- Verification
- Validation
- Software development

Want:

- As fast as silicon
- As detailed as silicon
- All the benefits of SW-based simulator
- Low cost

Reality: can only pick ~2.5

Our Thesis:

- FPGAs are the only viable basis technology

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

Useful Trends Throughout the Stack

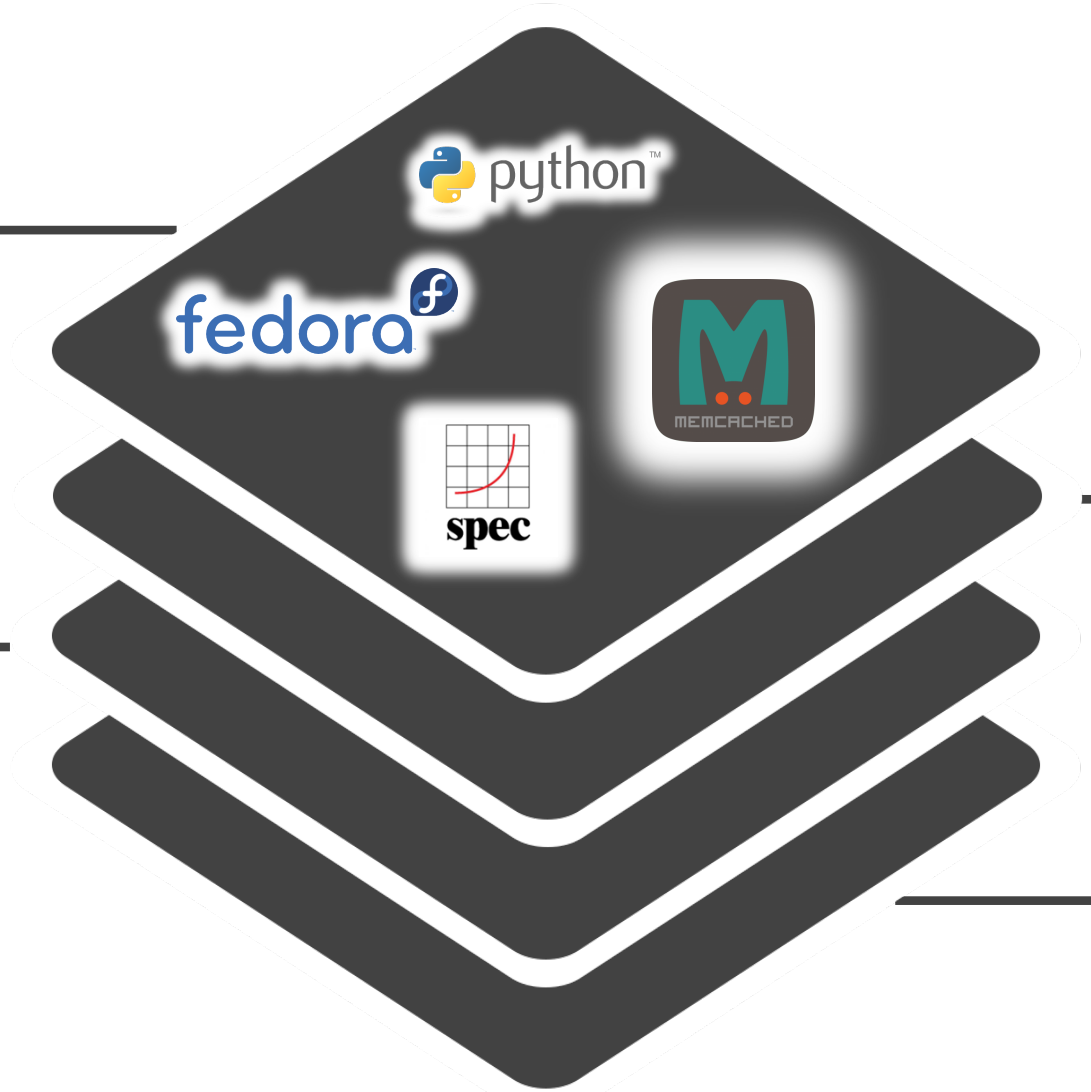
Open ISA

RISC-V

CHISEL

High-Productivity Hardware Design Language & IR

FIRRTL



Open, Silicon-Proven SoC Implementations

FPGAs in the Cloud

Amazon EC2 F1 Instances

Run Customizable FPGAs in the AWS Cloud

FireSim at 35,000 feet

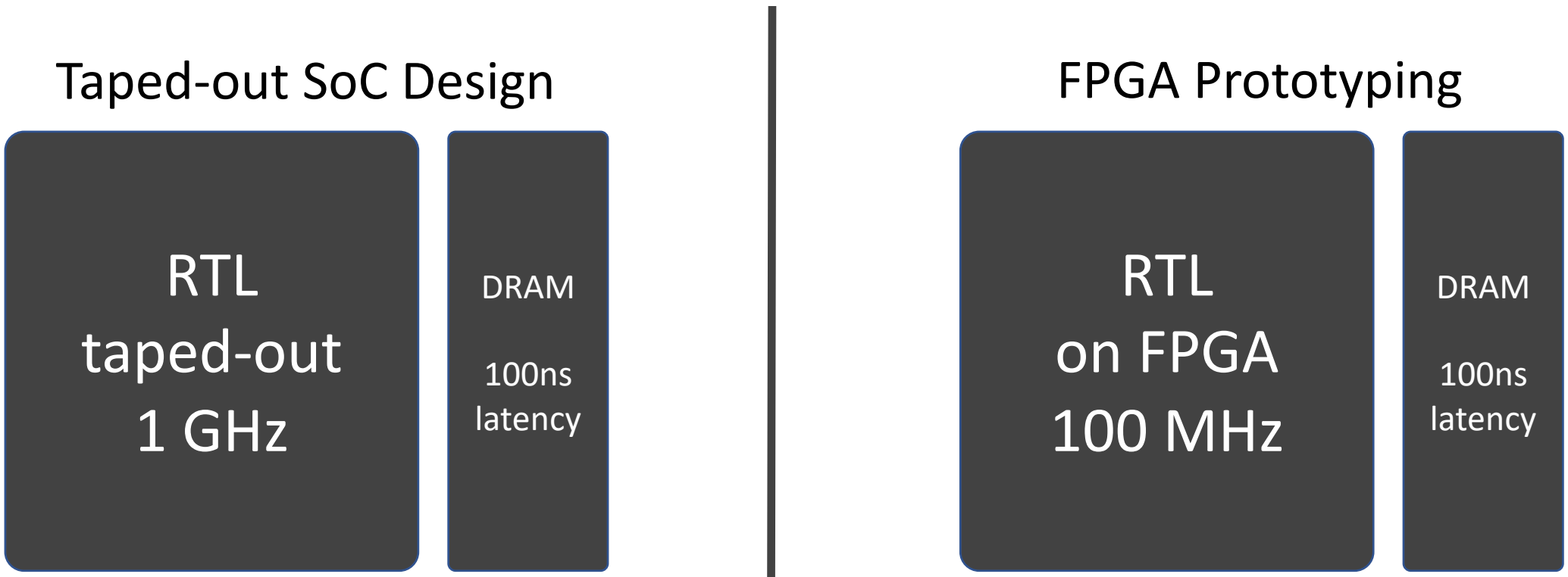
- Open-source, fast, automatic, deterministic FPGA-accelerated hardware simulation for pre-silicon verification and performance validation
- Ingests:
  - Your RTL design (FIRRTL, either via Chisel or Verilog via Yosys*)
  - HW and/or SW IO models (e.g. UART, Ethernet, DRAM, etc.)
  - Workload descriptions
- Produces:
  - Fast, cycle-exact simulation of your design + models around it
  - Automatically deployed to cloud FPGAs (AWS EC2 F1)

[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**

Three Distinguishing Features of FireSim

1) Not FPGA prototypes, rather FPGA-accelerated simulators
   - Akin to commercial FPGA emulation platforms
2) Uses cloud FPGAs
   - Inexpensive, elastic supply of large FPGAs
   - Easy to collaborate with other researchers
3) Open-source

Why is FPGA Prototyping Insufficient?



The Difficulty with FPGA Prototypes

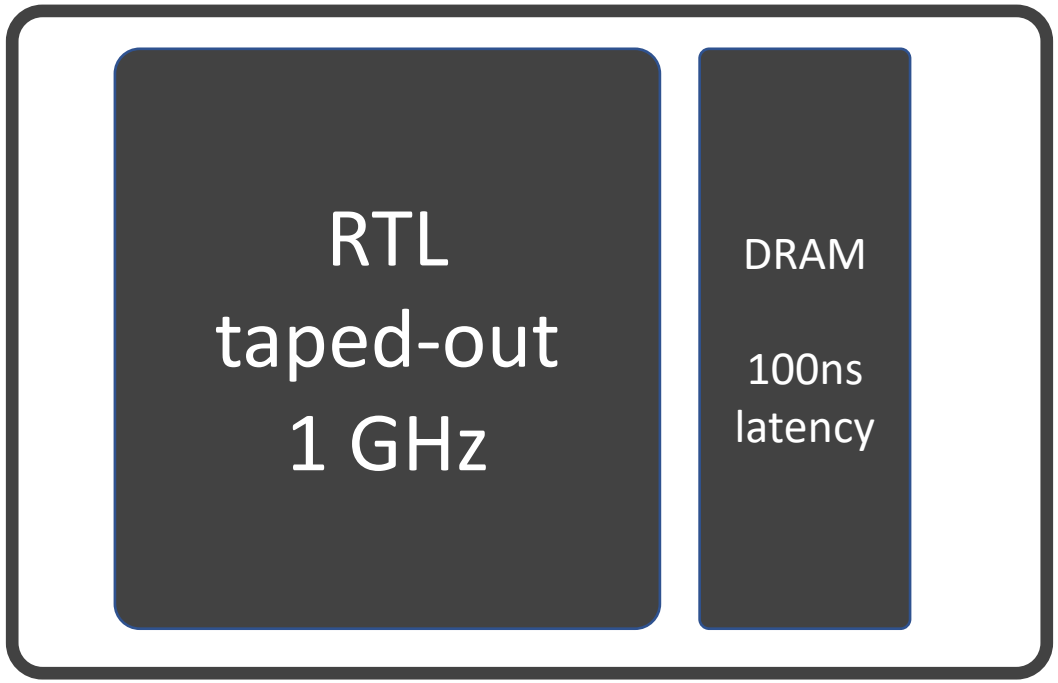
- Every FPGA clock executes one cycle of the simulated machine
- Exposes latencies of FPGA resources to the simulated world.

Three implications:

1) FPGA resources may not be an accurate model (ex. previous slide)
2) Simulations are non-deterministic
3) Different host FPGAs produce different simulation results

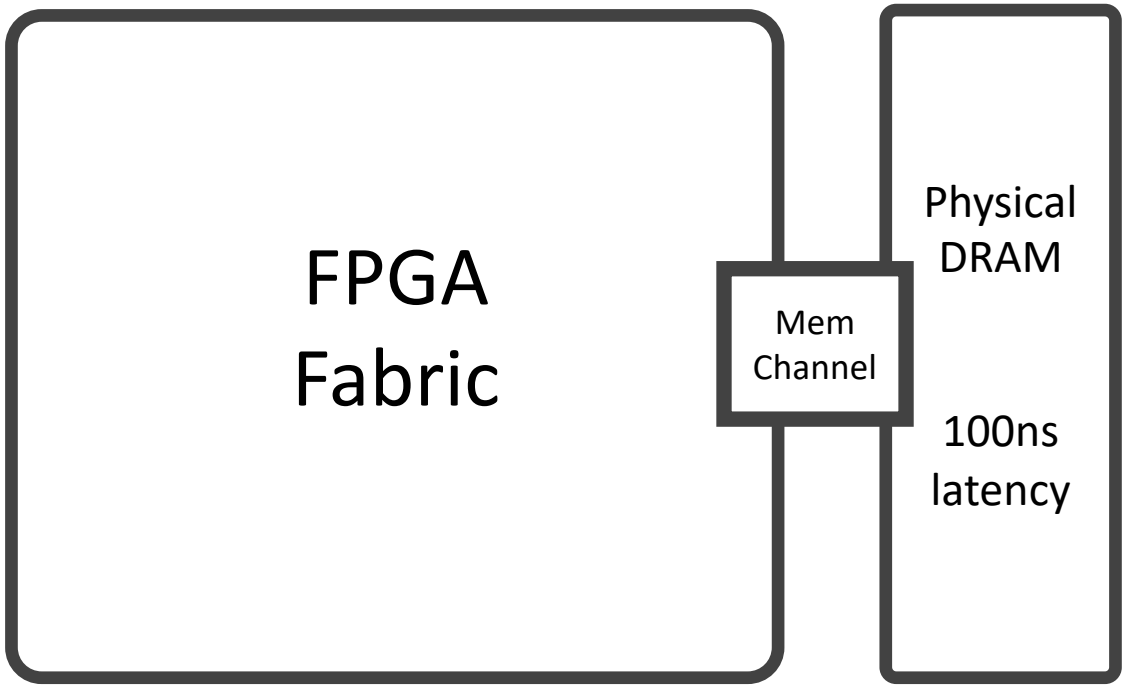
Separating Target and Host

Target: the machine under simulation



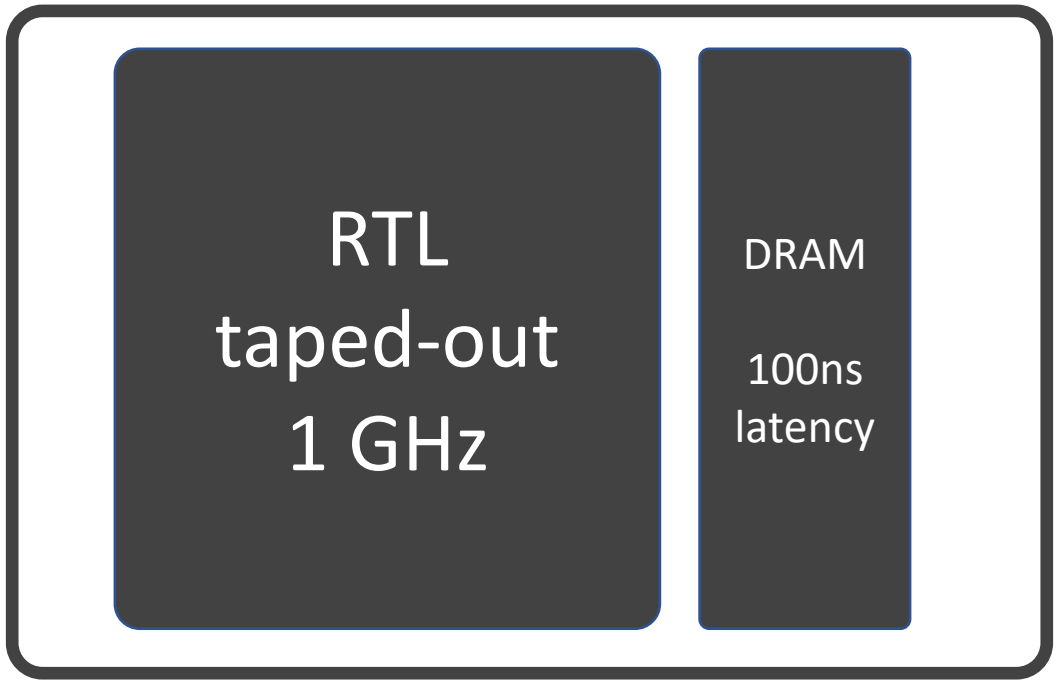
Closed simulation world.

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



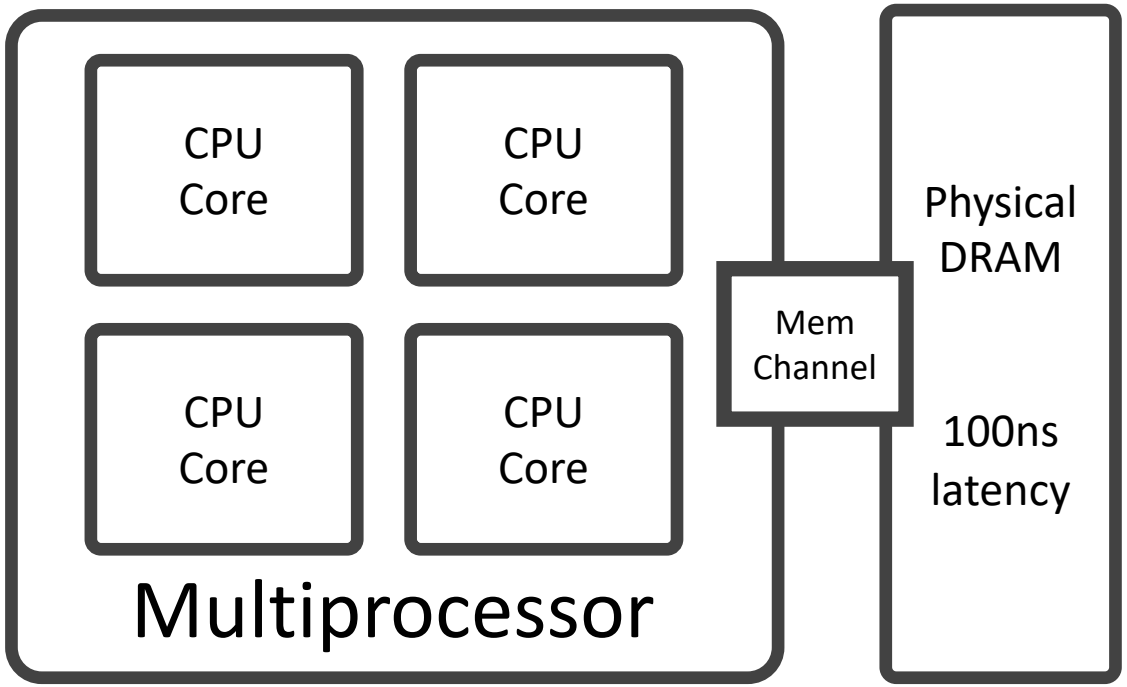
Separating Target and Host

Target: the machine under simulation

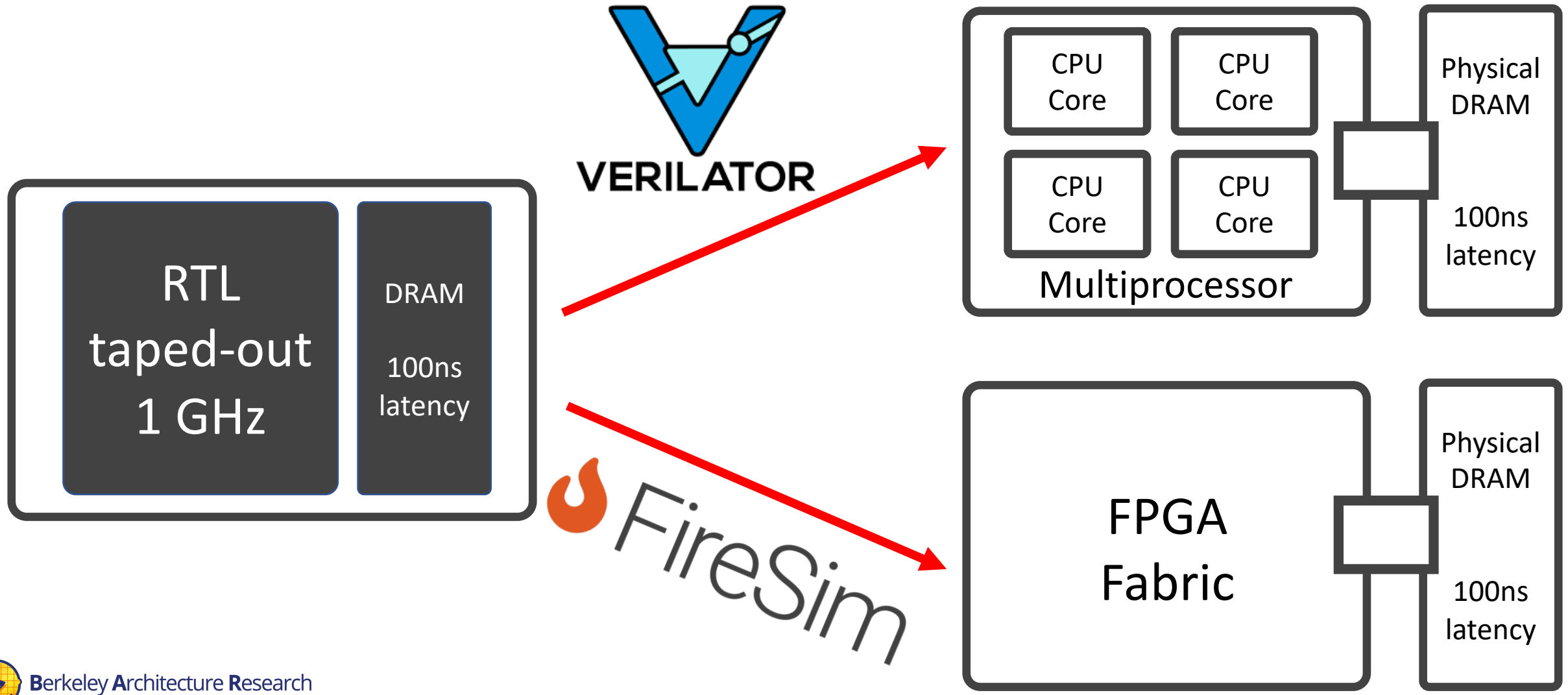


Closed simulation world.

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

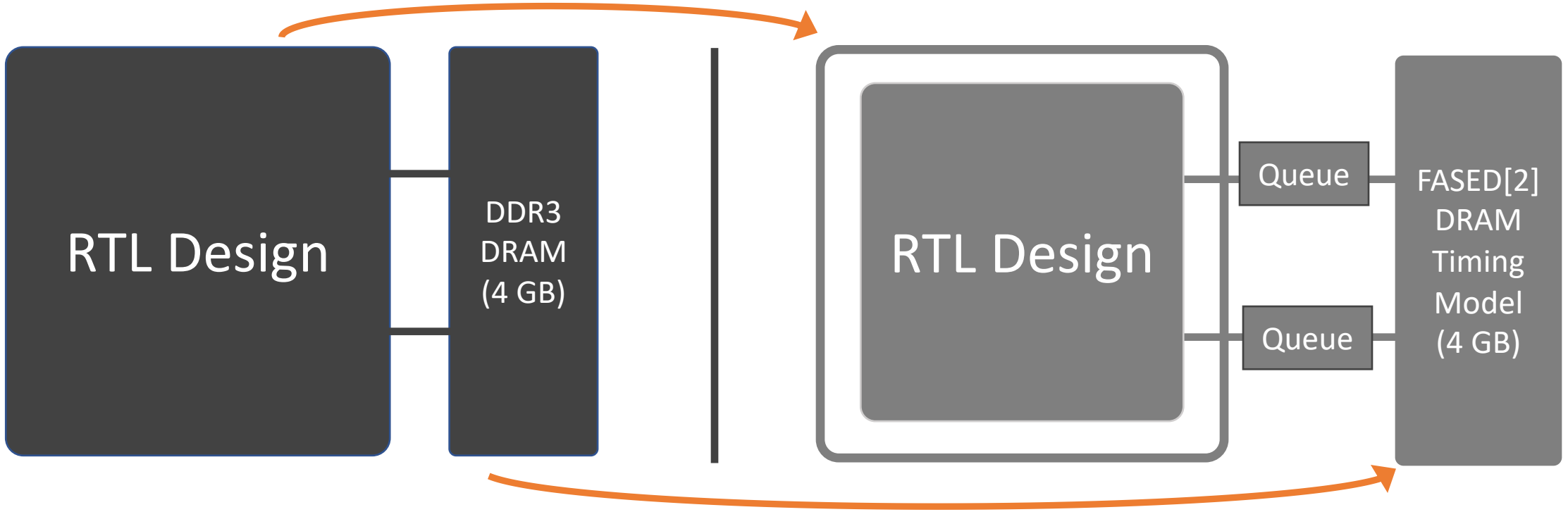


FireSim Generates FPGA-Hosted Simulators



Host Decoupling in FireSim: Transforming the Target

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



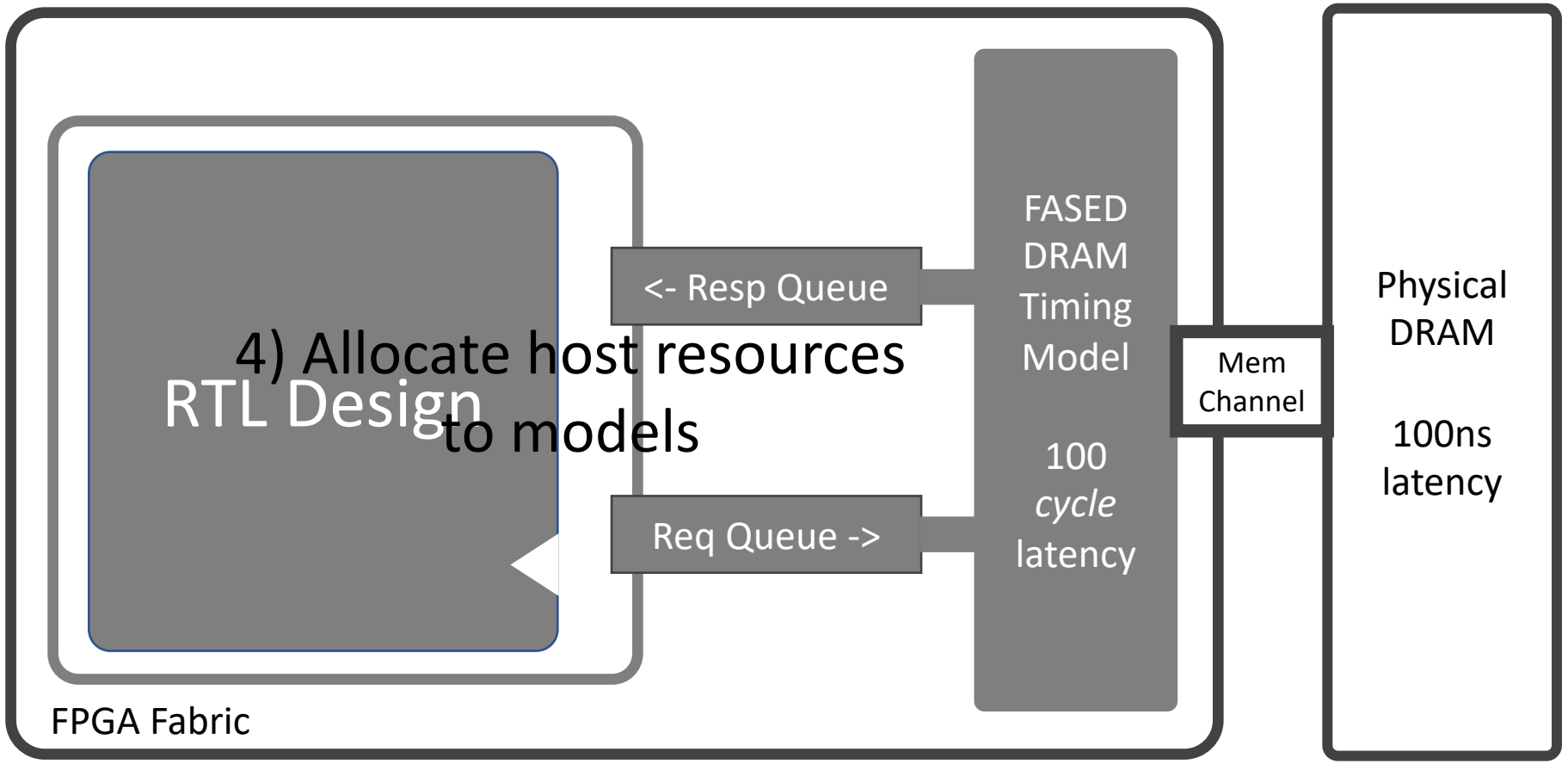
2) Generate FPGA-hosted model for DRAM[2] (think DRAMSim on an FPGA)

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:

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

This enables support for:

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

The Two Key Pieces of FireSim

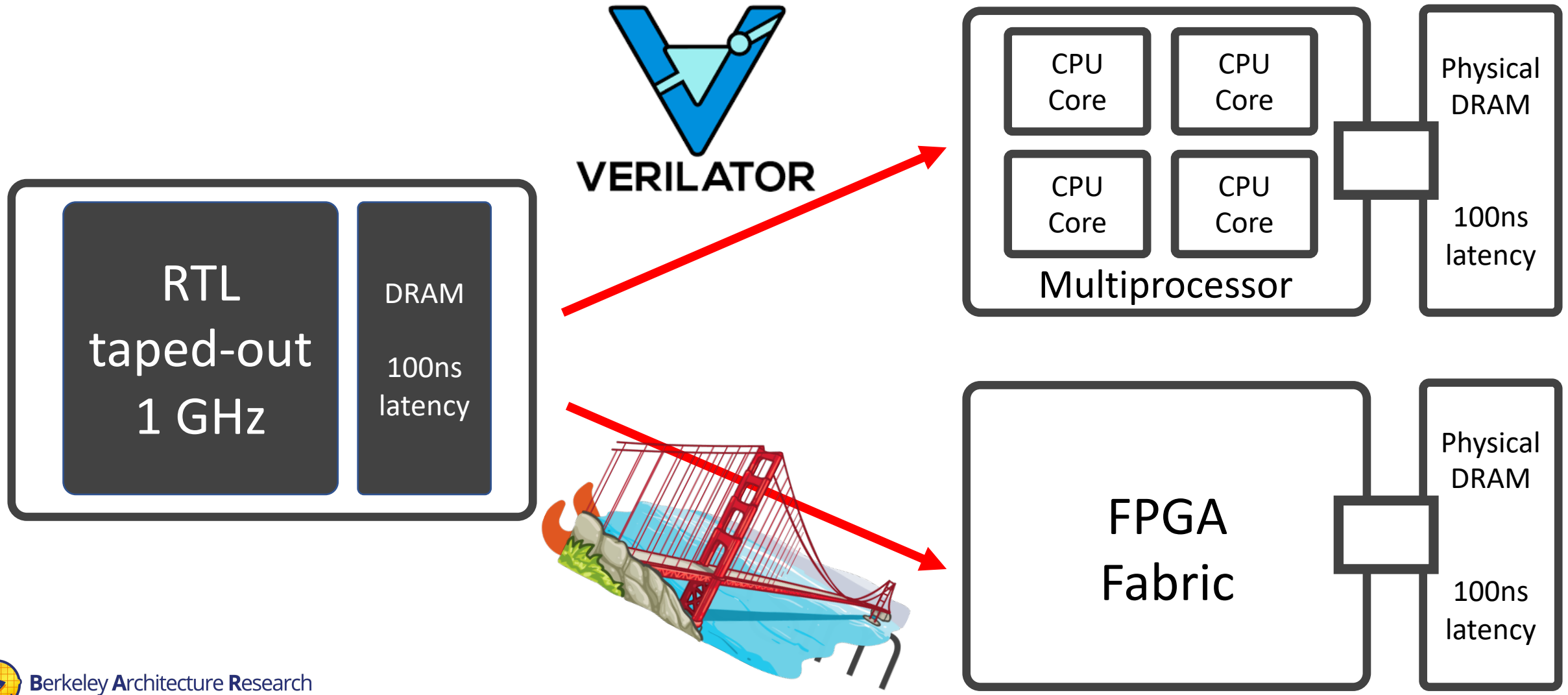
1) The Compiler → Golden Gate

- Consumes RTL (FIRRTL) and generates emulator sources

2) The Manager → `firesim`

- Puts everything together, distributes out to the cloud:
  - simulator builds (GG compilation, Bitstream generation)
  - simulations (workloads, FPGA images, SW drivers, etc.)

Golden Gate: The Analogy From Before



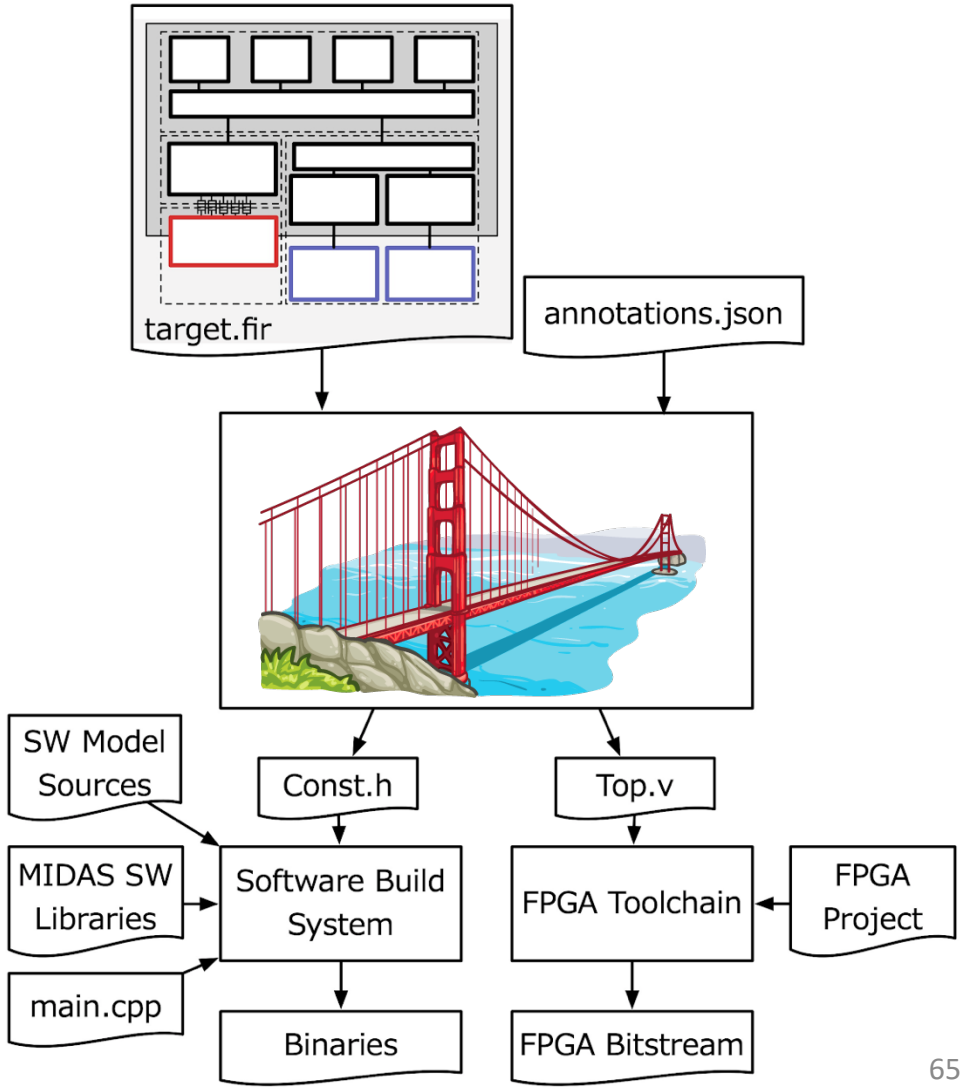
Golden Gate Compiler

Inputs:

- FIRRTL & annos from a Chipyard generator
- Compiler configuration

→ Produces sources for a simulator that are:

- deterministic
- support co-simulation of software models
- *area-optimized to fit more on the FPGA*

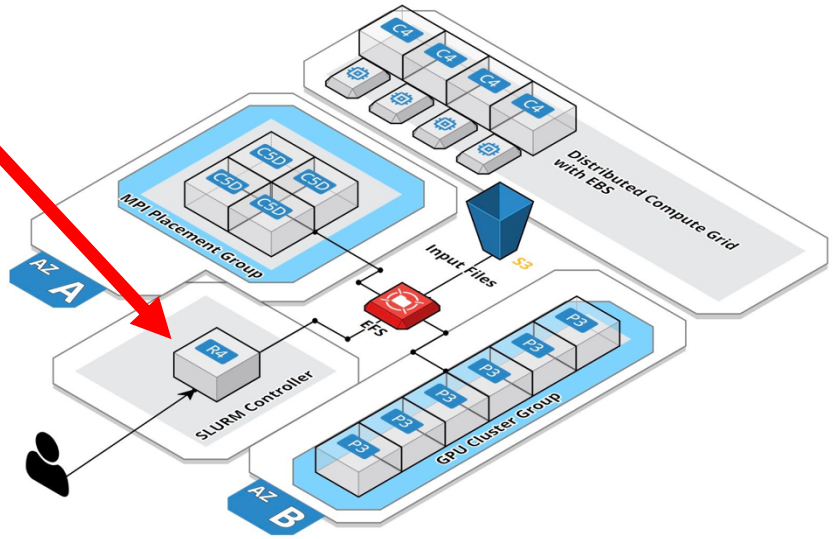


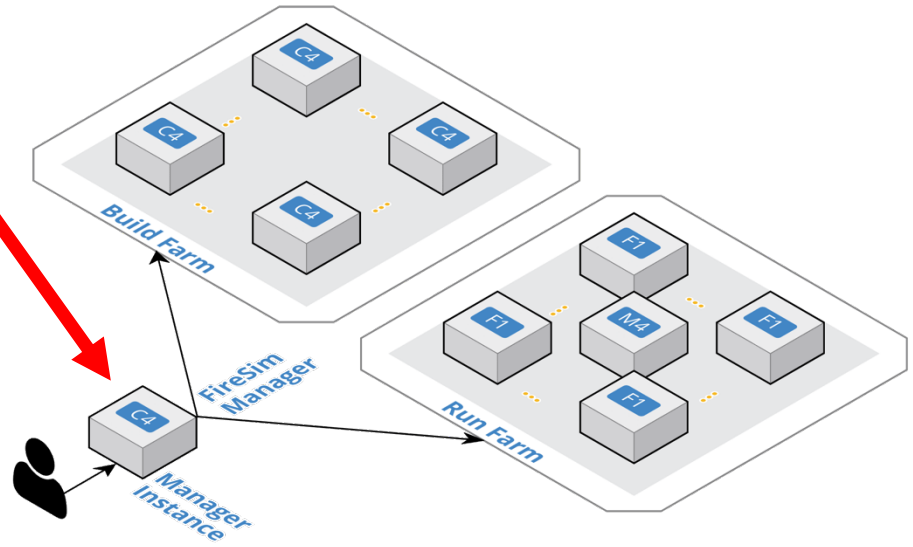
The Manager: A Second Analogy

slurm
workload manager

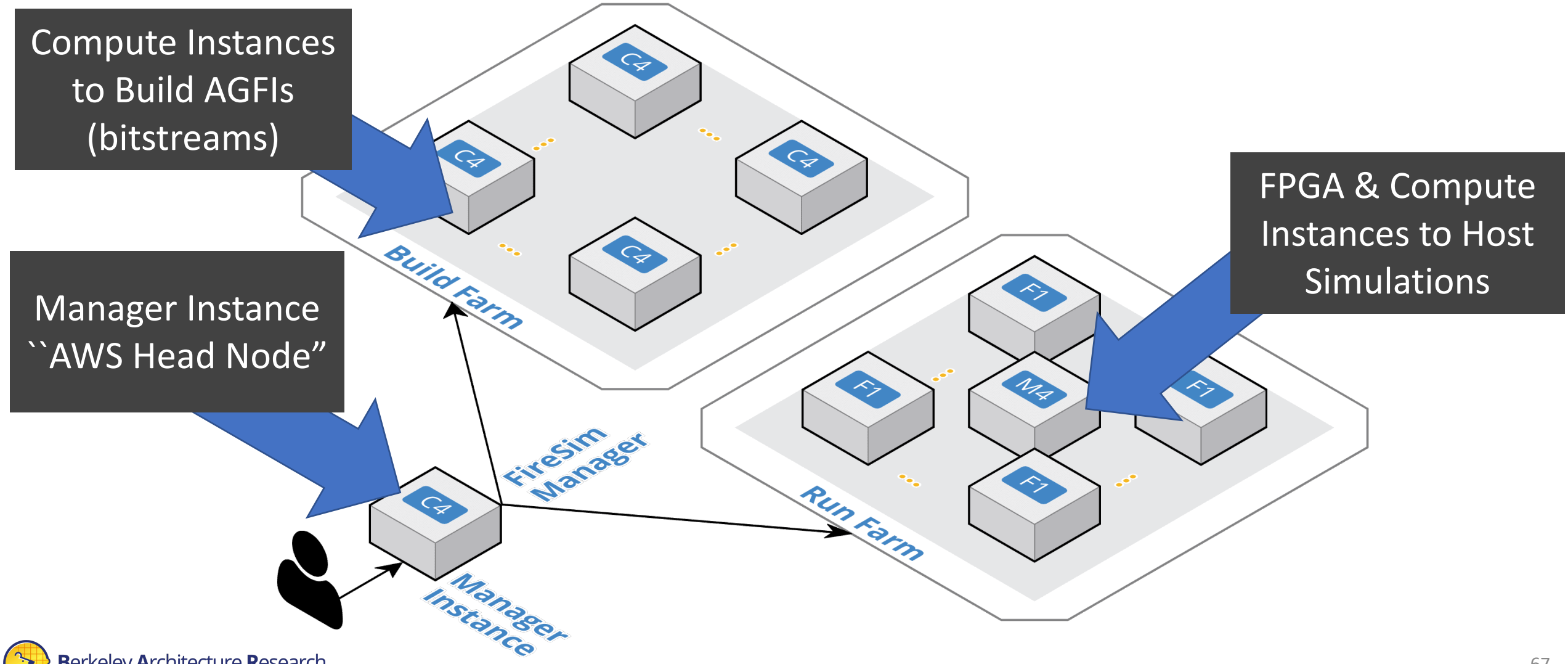
FireSim

Scale-out/Distribute Simulations

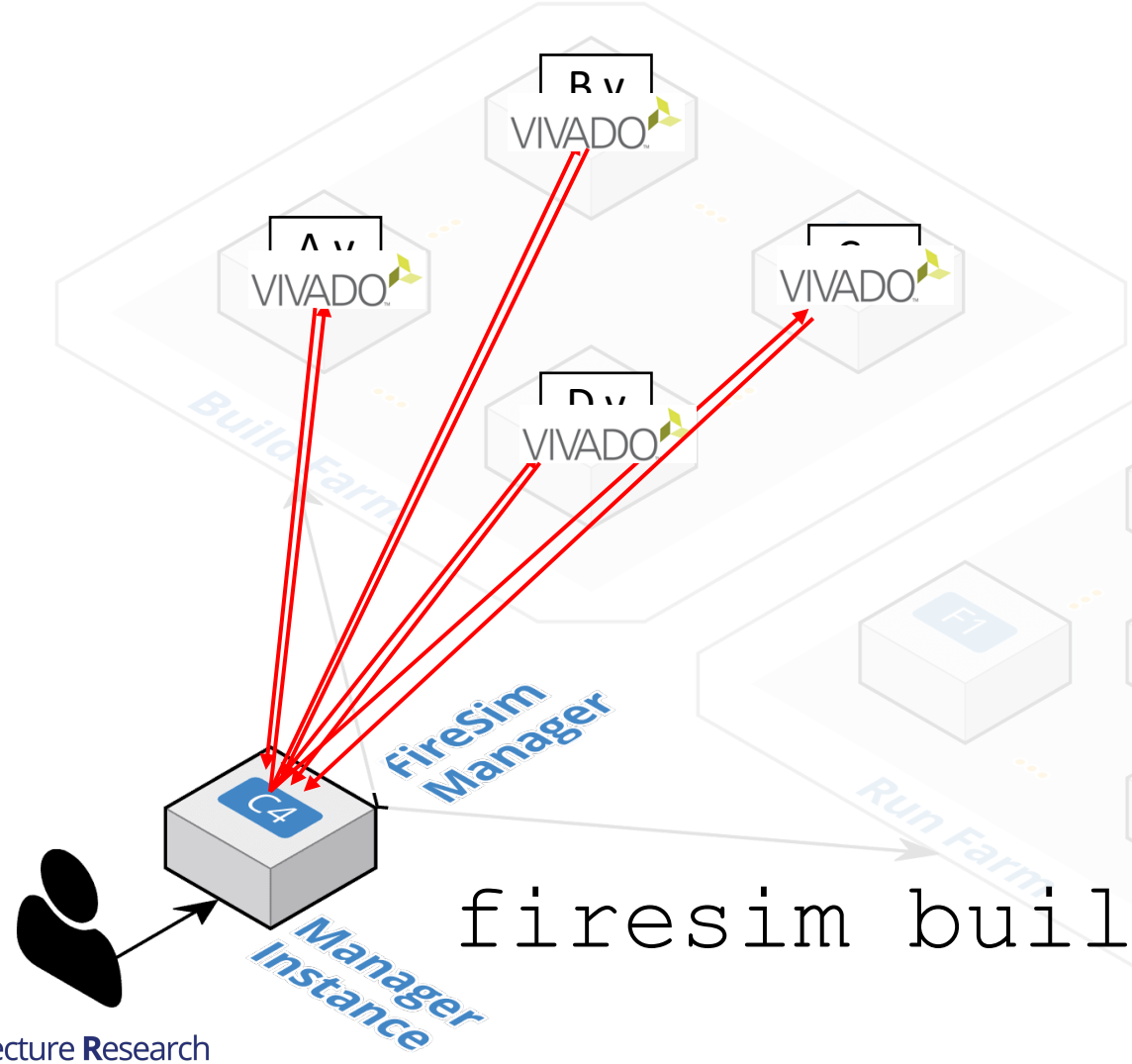




How Does FireSim use EC2?



Automated Simulator Builds

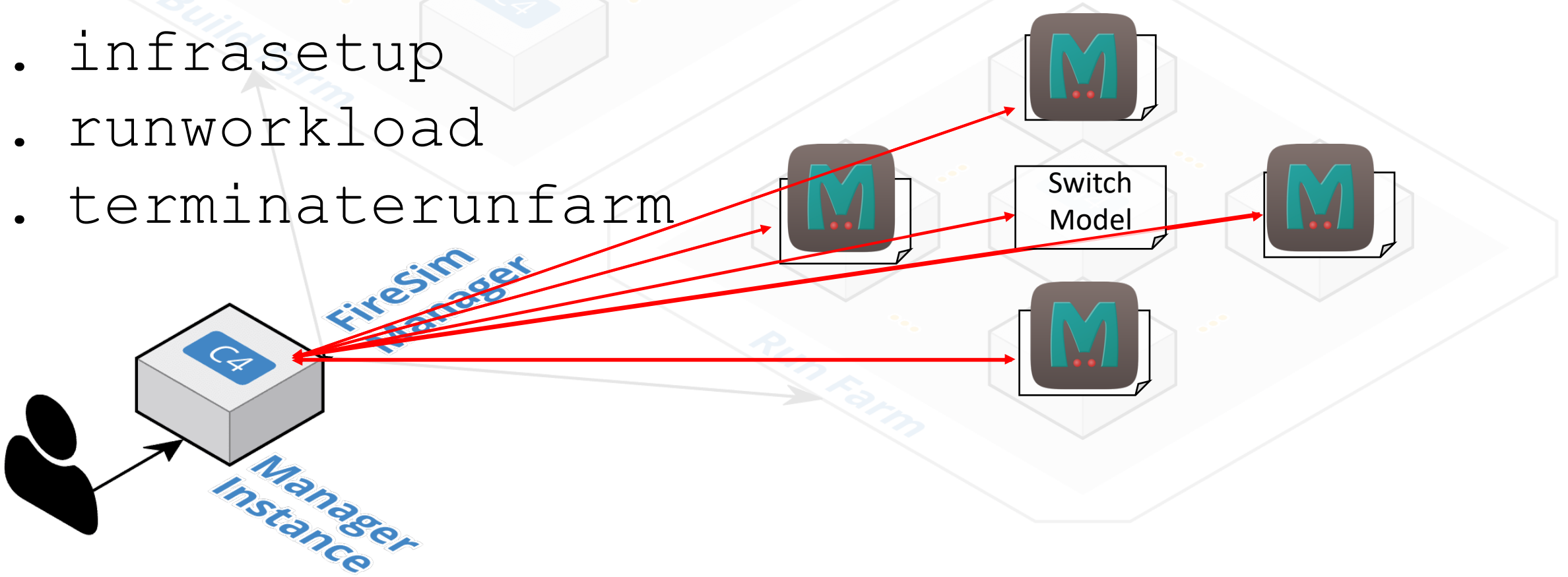


1. Spin up instances
2. Deliver sources
3. Run Vivado
4. Copies back results
5. Teardown instances

```
firesim buildafi
```

Automated Simulation Management

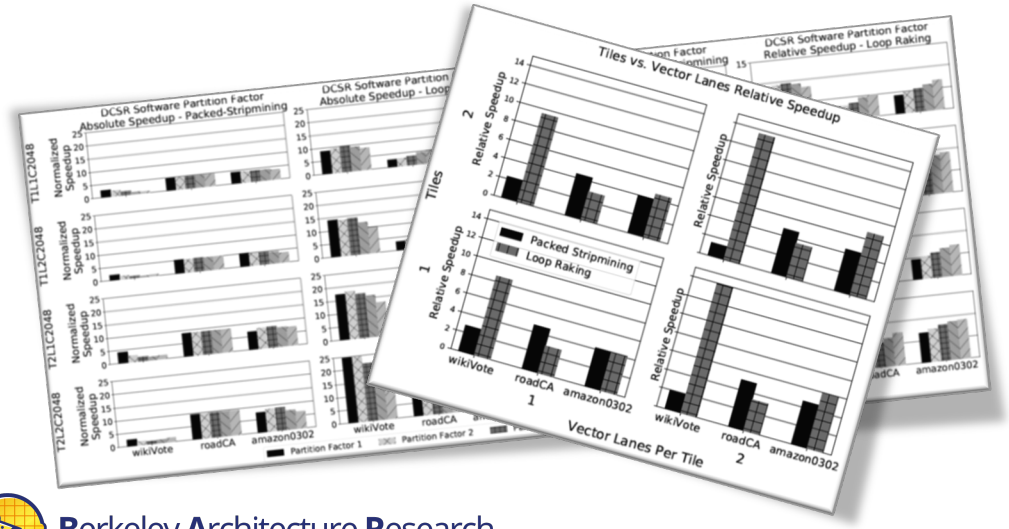
1. `launchrunfarm`
2. `infrasetup`
3. `runworkload`
4. `terminaterunfarm`

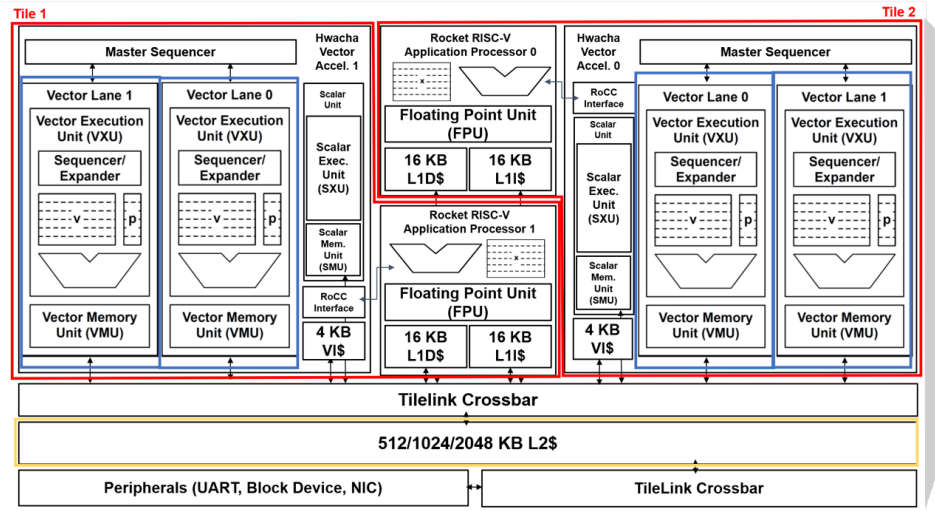


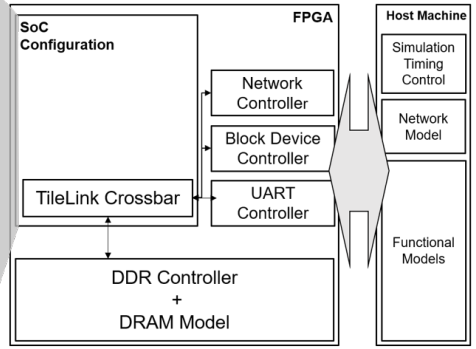
FireSim Features and Example use-cases

Evaluating SoC Designs

- Performance
  - SPECInt2017 with reference inputs on Rocket Chip within a day
- Full-System Design Space Exploration
  - Data-parallel accelerators (Hwacha) and multi-core processors
  - Complex software stacks (Linux, OpenMP, GraphMat, Caffe)

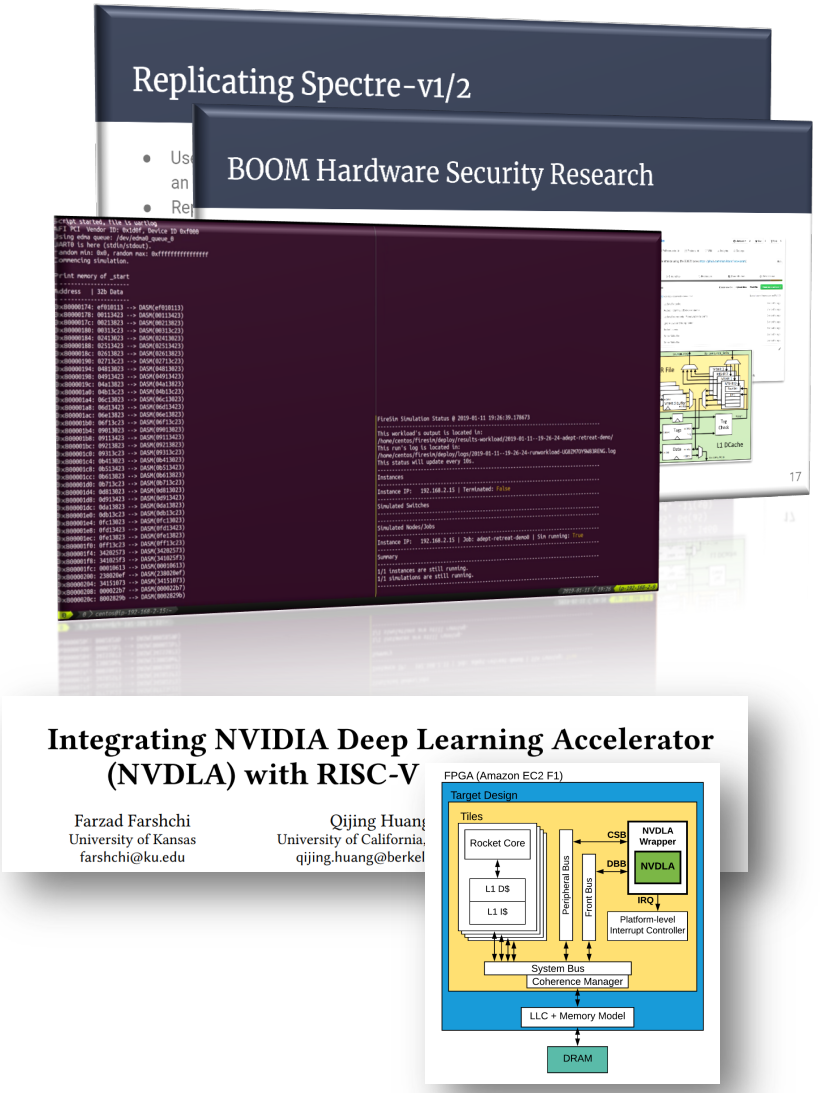






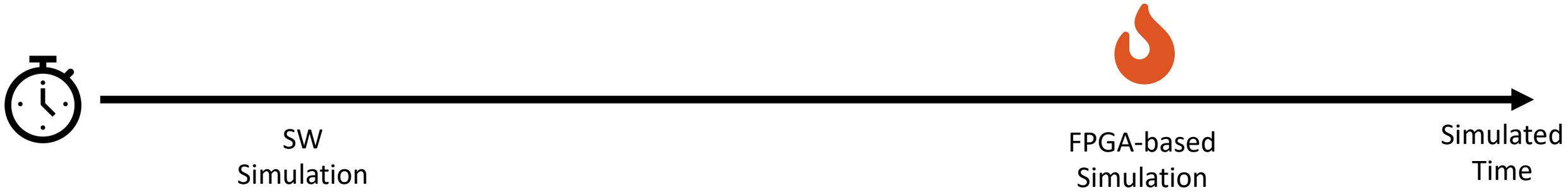
Evaluating SoC Designs

- Security
  - Replicate and identify uarch-level attacks, using pre-silicon RTL
  - BOOM Spectre replication
- Accelerator prototyping (integrated with Rocket Chip)
  - Chisel-based ML accelerators (Gemmini)
  - Open-source accelerator evaluation (NVDLA)
  - HLS-based rapid prototyping (Centrifuge)



Debugging and Profiling SoC Designs

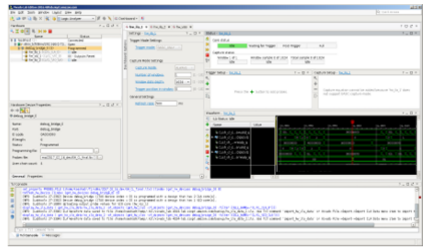
- High simulation speed in FPGA-based simulation enables advanced debugging and profiling tools.
- Reach “deep” in simulation time, and obtain large levels of coverage and data



Debugging and Profiling SoC Designs

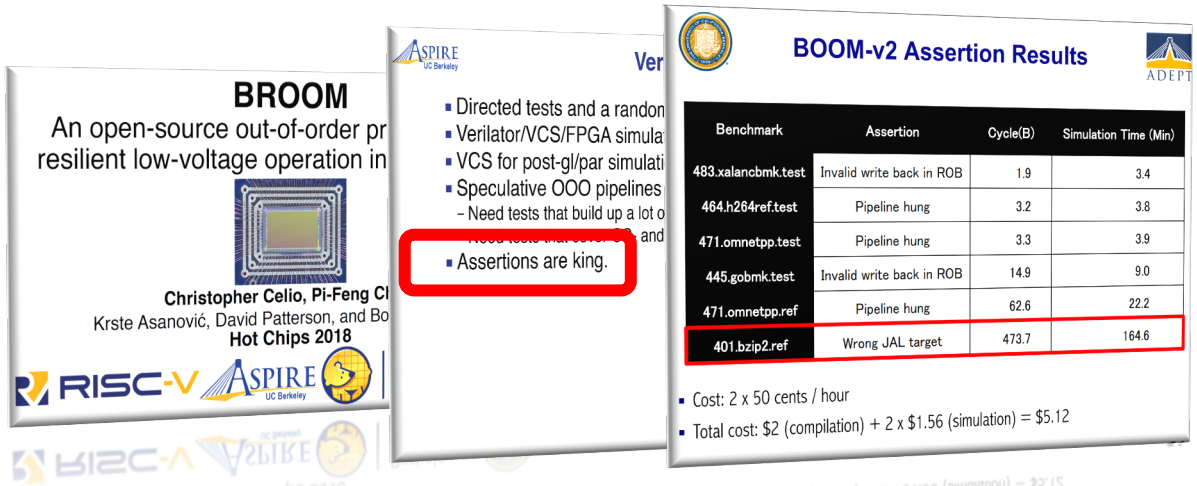
- AutoILA: Easy-to-use Integrated Logic Analyzer (ILA) support
  - User annotates interesting signals in the target design Chisel
  - Rest of the wiring is automatic
  - Use standard Vivado tools to control/collect data from ILA
- Example AutoILA use cases:
  - Identify billion-cycle RVC instruction fetch bugs and load bugs in BOOM during Linux boot
  - Identifying AMO bug in the Hwacha accelerator deep into simulation (after Linux boot)

```
import midas.targetutils.FpgaDebug

class SomeModuleIO(implicit p: Parameters) extends SomeIO()(p){
   val out1 = Output(Bool())
   val in1 = Input(Bool())
   FpgaDebug(out1, in1)
}
```

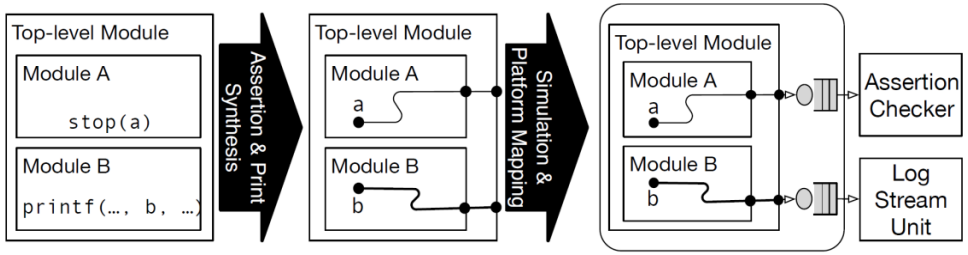


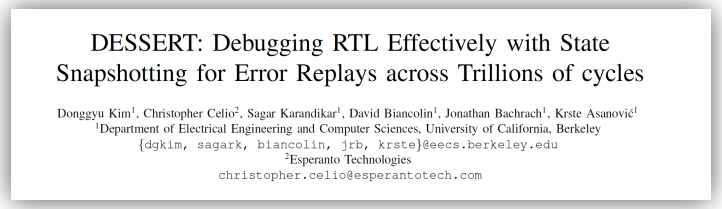
Debugging and Profiling SoC Designs

- Assertion Synthesis, Printf Synthesis
  - Common software debugging primitives
  - Automatic integration with simulation host
  - Assertions helped Identify BOOM bugs trillions of cycles into execution



From: BROOM: An open-source Out-of-Order processor with resilient low-voltage operation in 28nm CMOS, Christopher Celio, Pi-Feng Chiu, Krste Asanovic, David Patterson and Borivoje Nikolic. HotChip 30, 2018





DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of cycles

Donggyu Kim[1], Christopher Celio[2], Sagar Karandikar[1], David Biancolin[1], Jonathan Bachrach[1], Krste Asanović[1]
[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{dgkim, sagark, biancolin, jrb, krste}@eecs.berkeley.edu
[2]Esperanto Technologies
christopher.celio@esperantotech.com

From: Donggyu Kim, Christopher Celio, Sagar Karandikar, David Biancolin, Jonathan Bachrach, and Krste Asanović, "DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of cycles", FPL 2018

BOOM Example

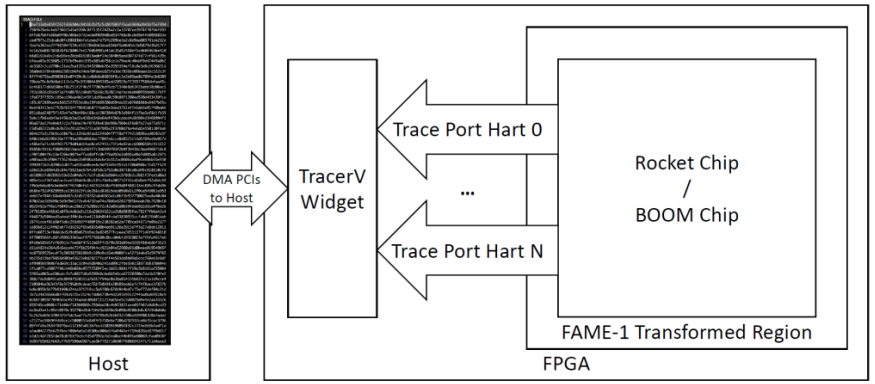
- How it looks in the UART output (while Linux is booting):

```
[    0.008000] VFS: Mounted root (ext2 filesystem) on device 253:0.
[    0.008000] devtmpfs: mounted
[    0.008000] Freeing unused kernel memory: 148K
[    0.008000] This architecture does not have kernel memory protection.
mount: mounting sysfs on /sys failed: No such device
Starting syslogd: OK
Starting klogd: OK
Starting mdev...
mdev: /sys/dev: No such file or directory
[id: 1840, module: Rob, path: FireBoom.boom_tile_1.core.rob]
Assertion failed: [rob] writeback (0) occurred to an invalid ROB entry.
    at rob.scala:504 assert (!(io.wb_resps(i).valid && MatchBank(GetBankIdx(rob_idx)) &&
 at cycle: 1112250469

*** FAILED *** (code = 1841) after 1112250485 cycles
time elapsed: 307.8 s, simulation speed = 3.61 MHz
FPGA-Cycles-to-Model-Cycles Ratio (FMR): 2.77
Beats available: 2165
Runs 1112250485 cycles
[FAIL] FireBoom Test
SEED: 1569631756
 at cycle 4294967295
```

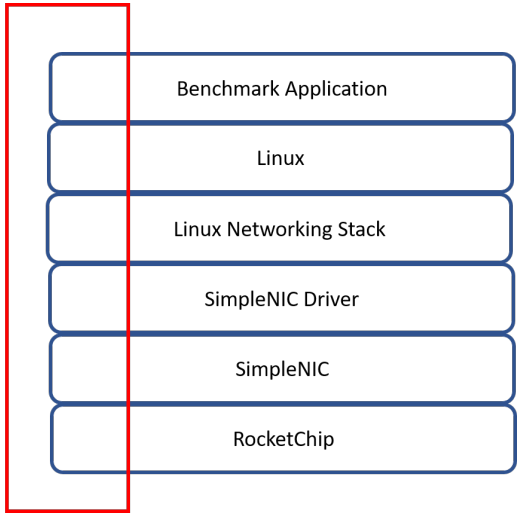
It would take ~62 hours to hit this assertion is SW RTL simulation (at 5 KHz sim rate), vs. just a few minutes in FireSim

Debugging and Profiling SoC Designs

- TracerV: Out-of-band collection of instruction traces from RISC-V systems
  - Profiling does not perturb execution
- Useful for kernel and hypervisor level cycle-sensitive profiling. Examples Use Cases:
  - Co-Optimization of NIC, Network Driver, Linux
  - Keystone Secure Enclave Project
  - High-performance hardware-specific code



Why Is This Slow ?

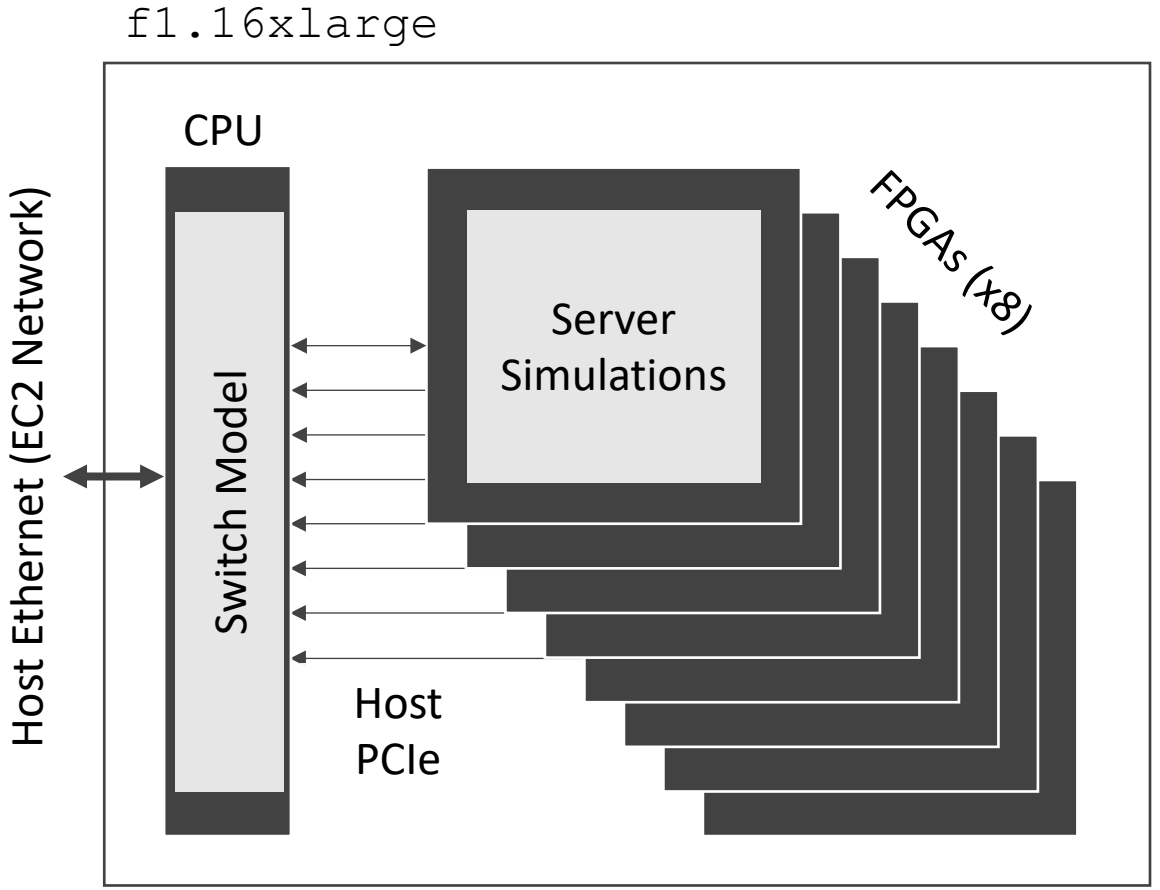


More about this in ASPLOS 2020:

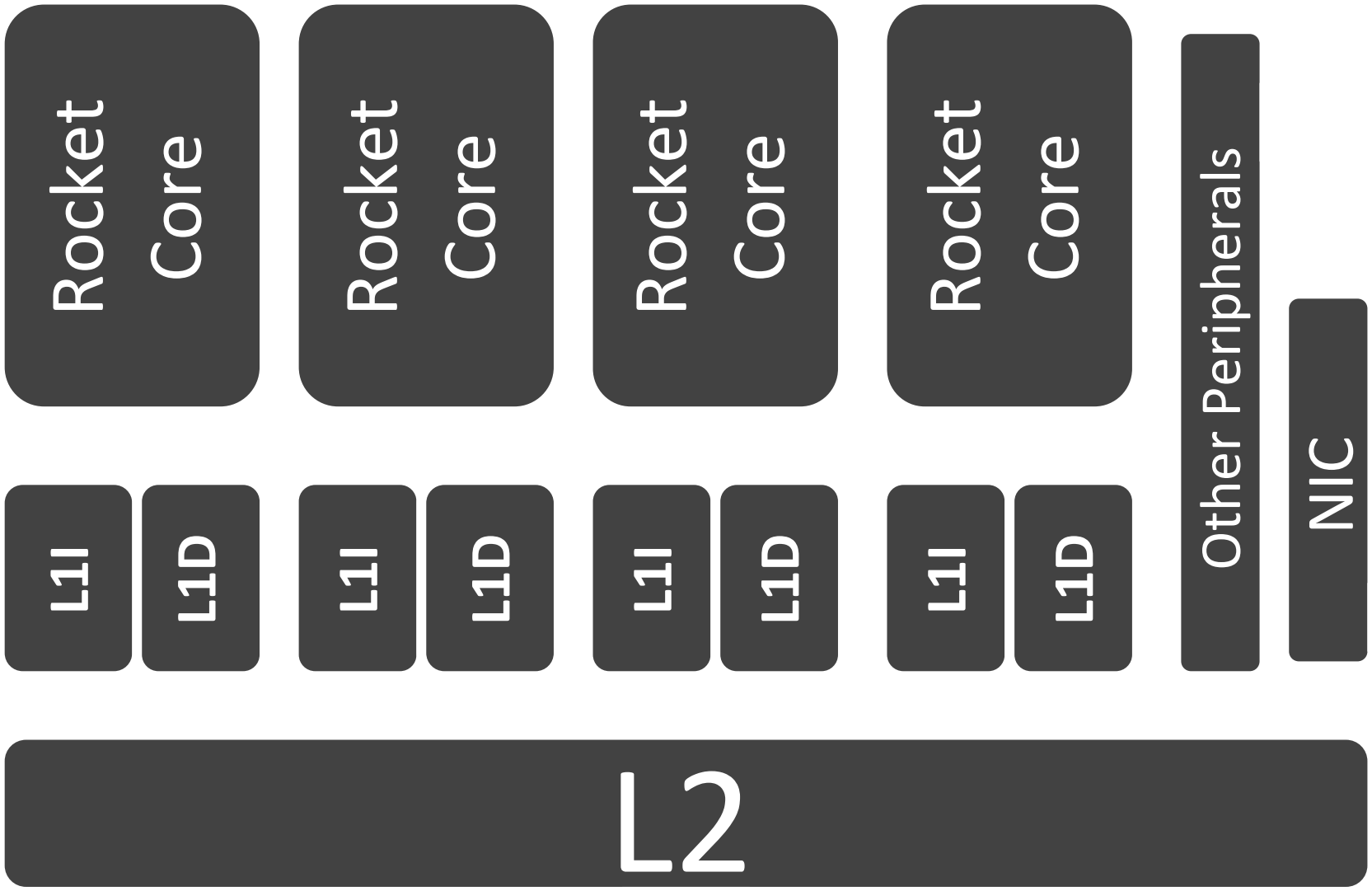
S. Karandikar, et al. "**FirePerf: FPGA-Accelerated Full-System Hardware/Software Performance Profiling and Co-Design**". *To appear in ASPLOS 2020*. https://sagark.org/assets/pubs/fireperf-asplos2020.pdf

Scaling-Out SoC Designs

- Scale-out simulation
  - 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

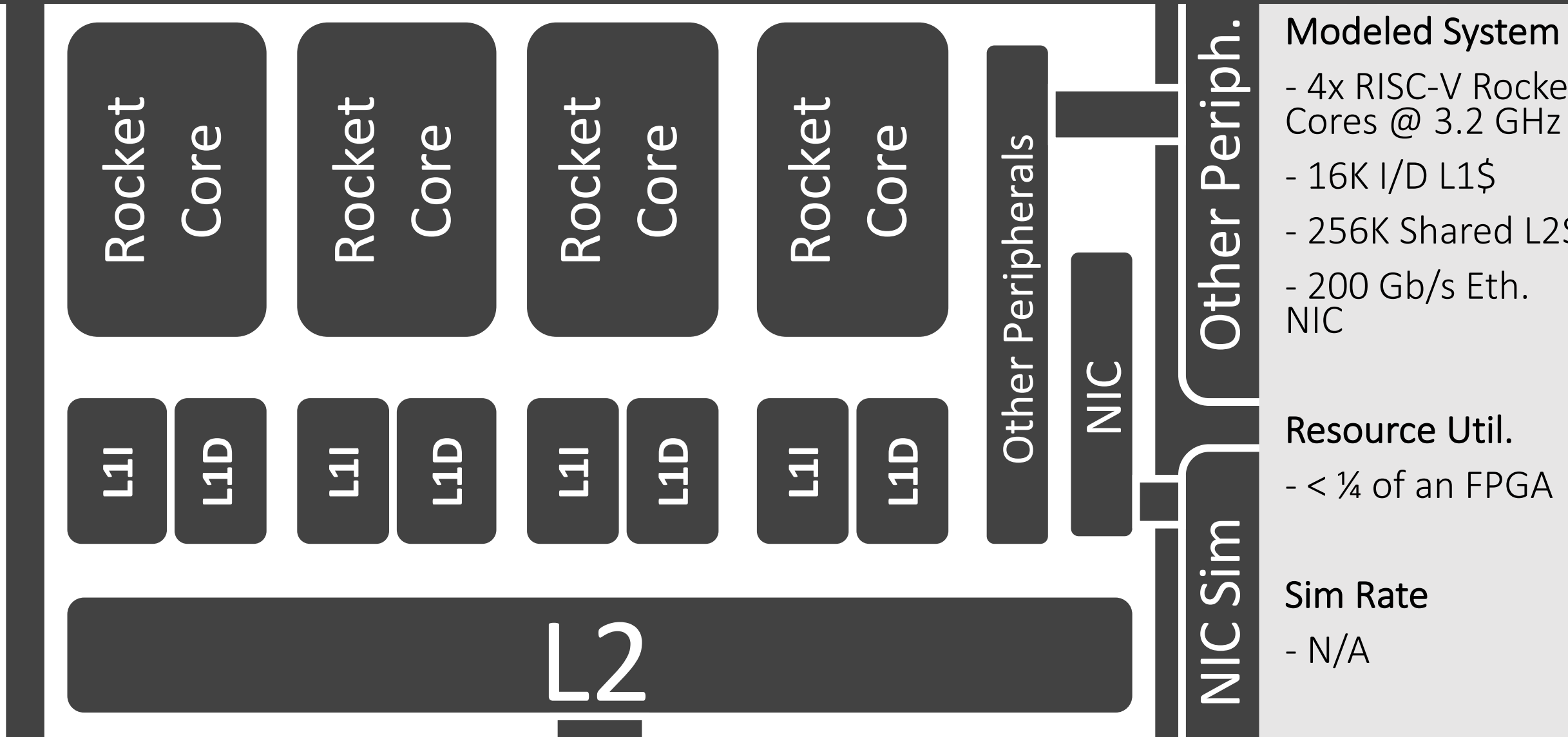
Resource Util.

- < ¼ of an FPGA

Sim Rate

- N/A

Step 1: Server SoC in RTL



Modeled System

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

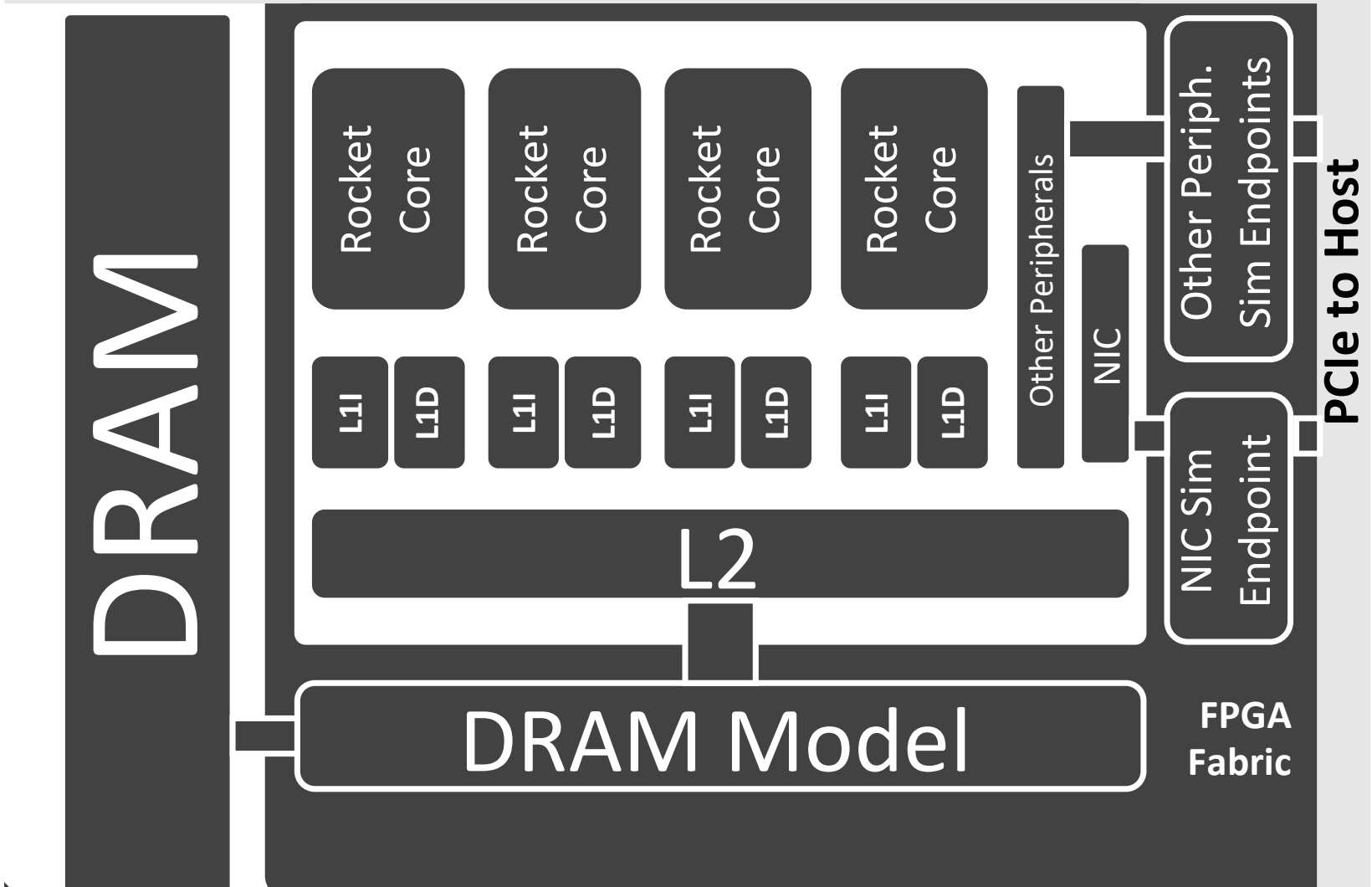
Resource Util.

- < ¼ of an FPGA

Sim Rate

- N/A

Step 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

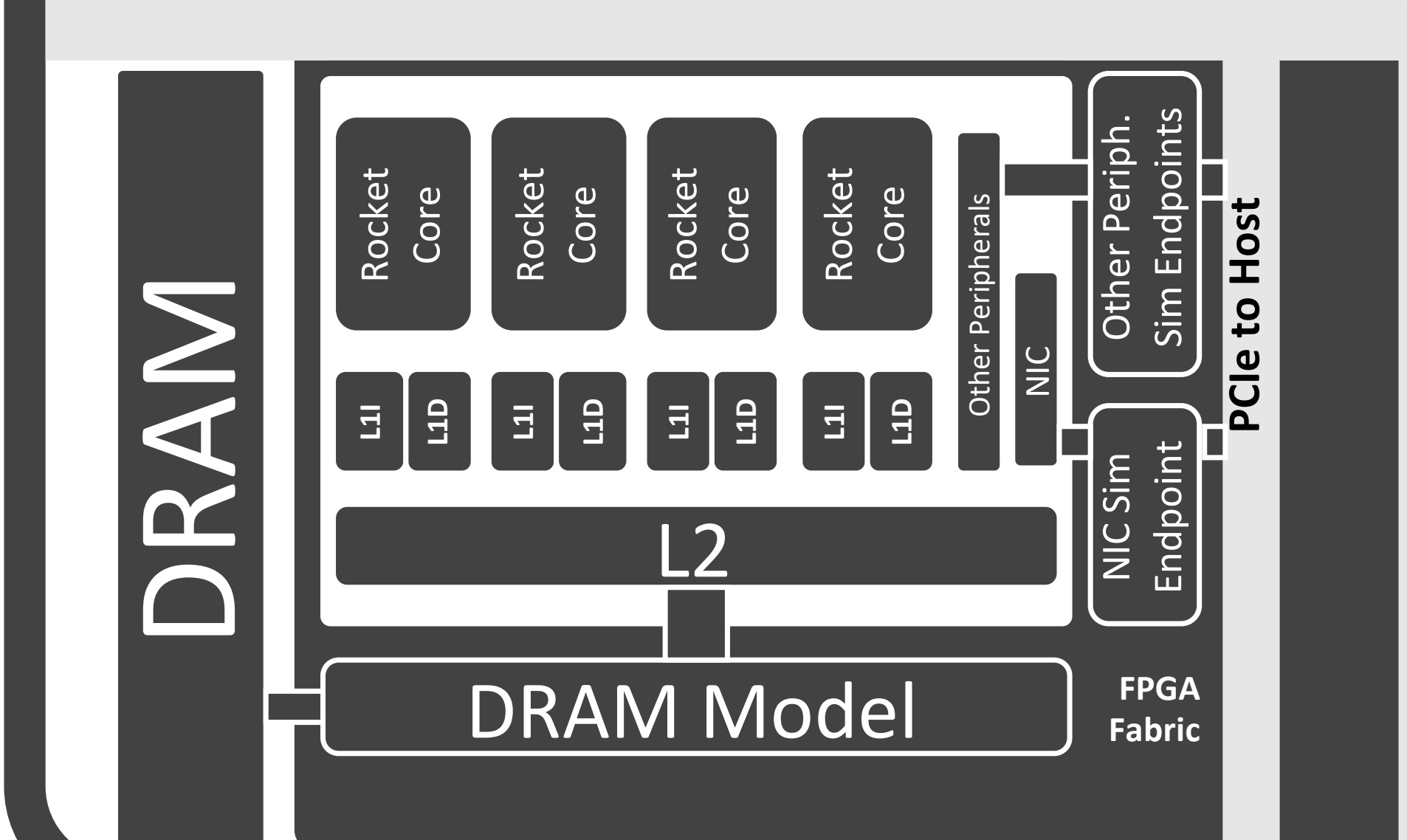
Resource Util.

- < ¼ of an FPGA
- ¼ Mem Chans

Sim Rate

- ~150 MHz
- ~40 MHz (netw)

Step 2: FPGA Simulation of one server blade



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

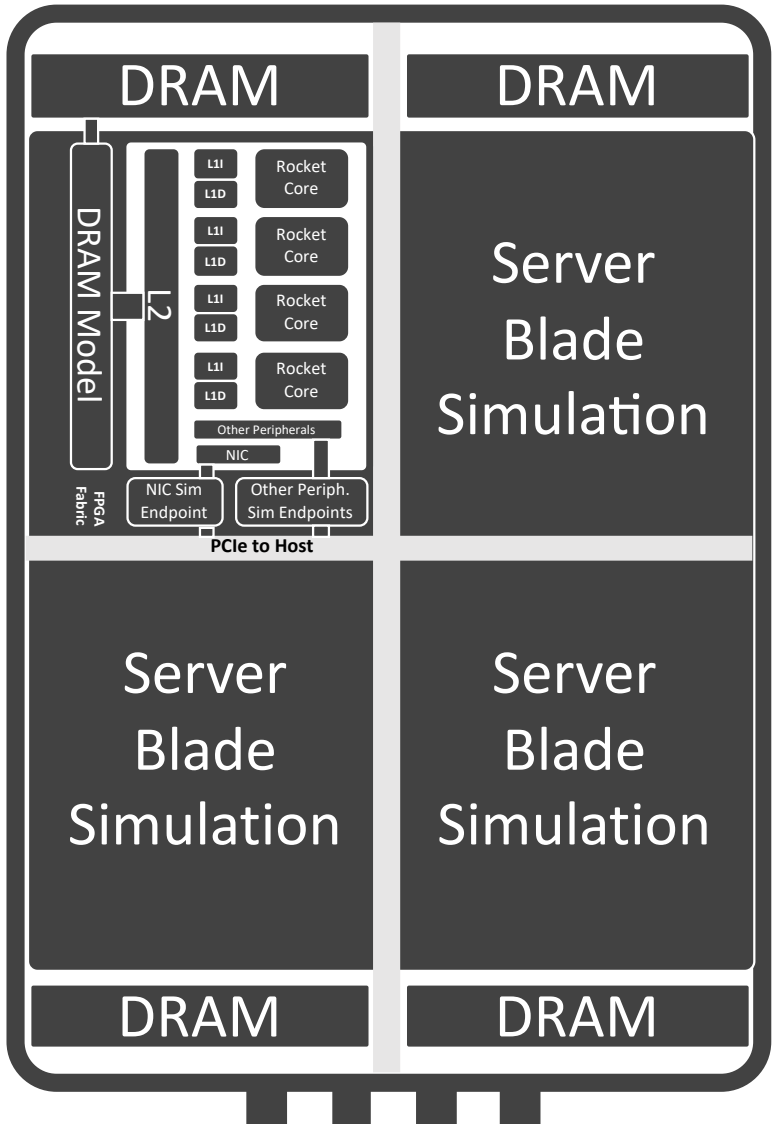
Resource Util.

- < ¼ of an FPGA
- ¼ Mem Chans

Sim Rate

- ~150 MHz
- ~40 MHz (netw)

Step 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.

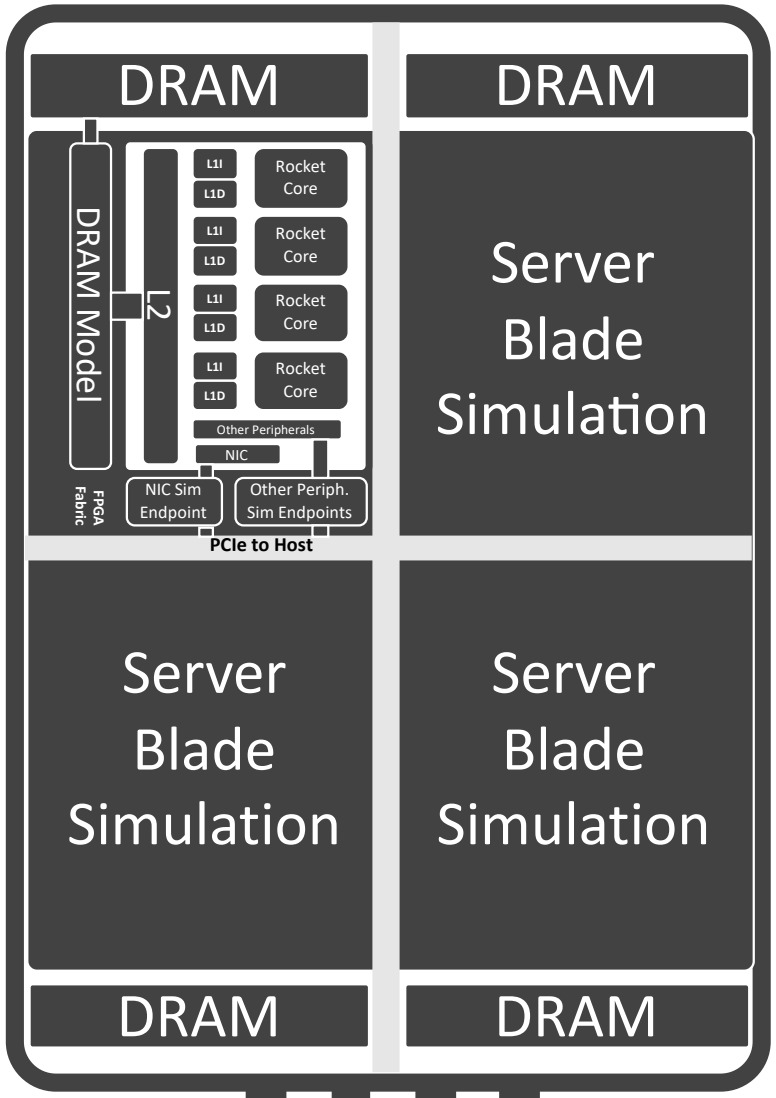
- < 1 FPGA
- 4/4 Mem Chans

Sim Rate

- ~14.3 MHz (netw)

Step 3: FPGA Simulation of 4 server blades

FPGA
4 Sims)



FPGA
(4 Sim

Modeled System

- 4 Server Blades
- 16 Cores
- 64 GB DDR3

Resource Util.

- < 1 FPGA
- 4/4 Mem Chans

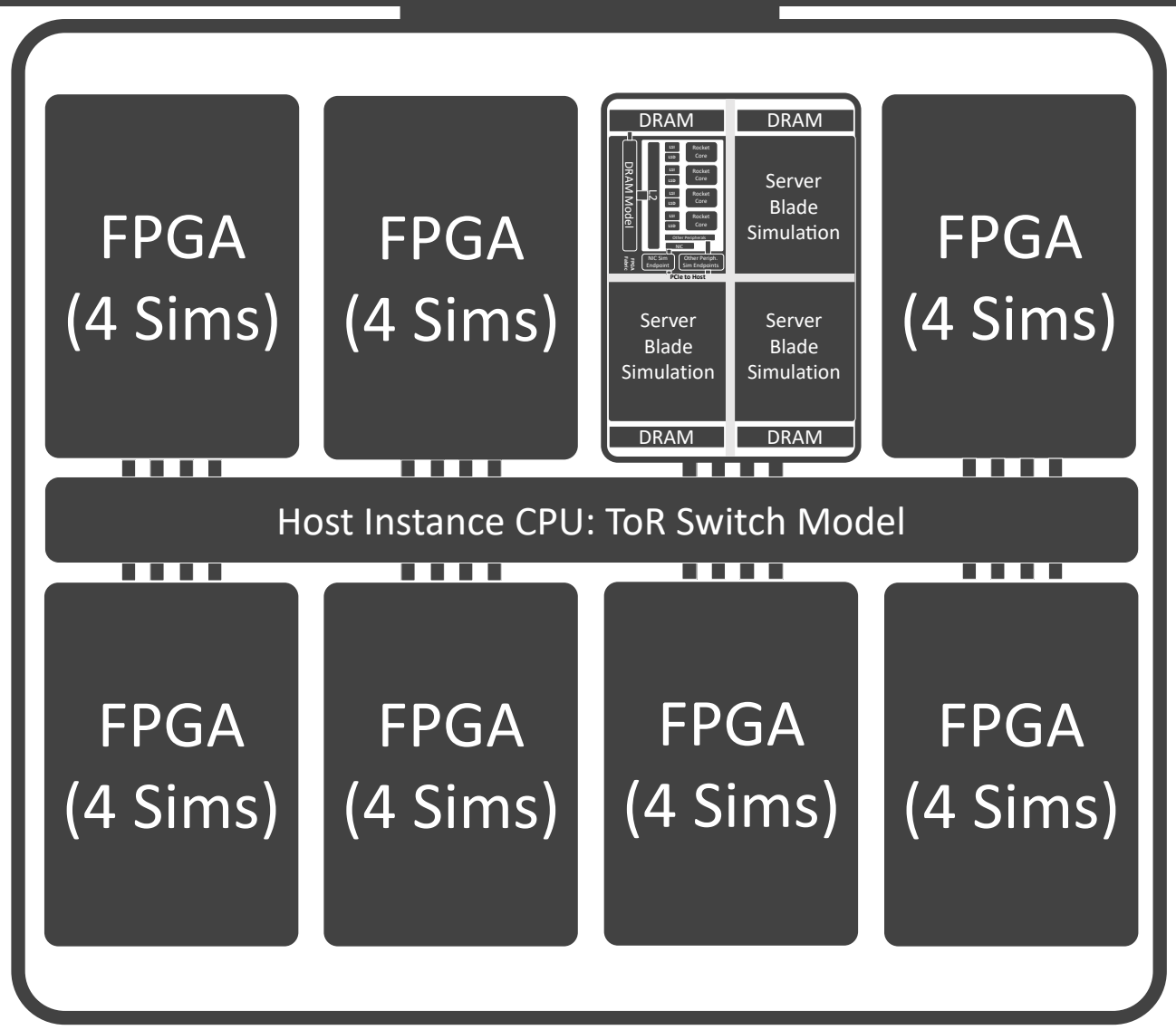
Sim Rate

- ~14.3 MHz (netw)

Step 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

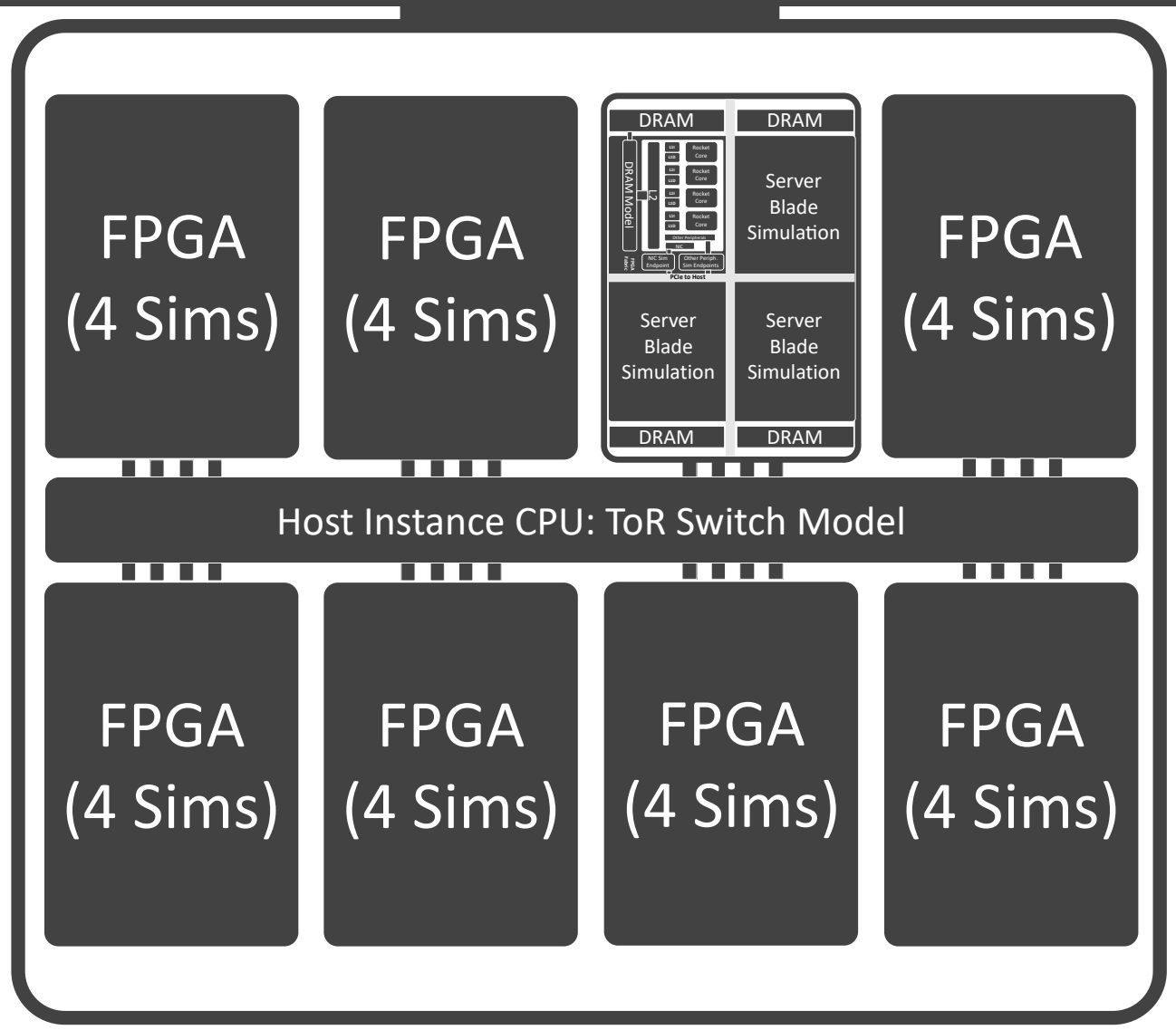
Sim Rate

- ~10.7 MHz (netw)

Step 4: Simulating a 32 node rack

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

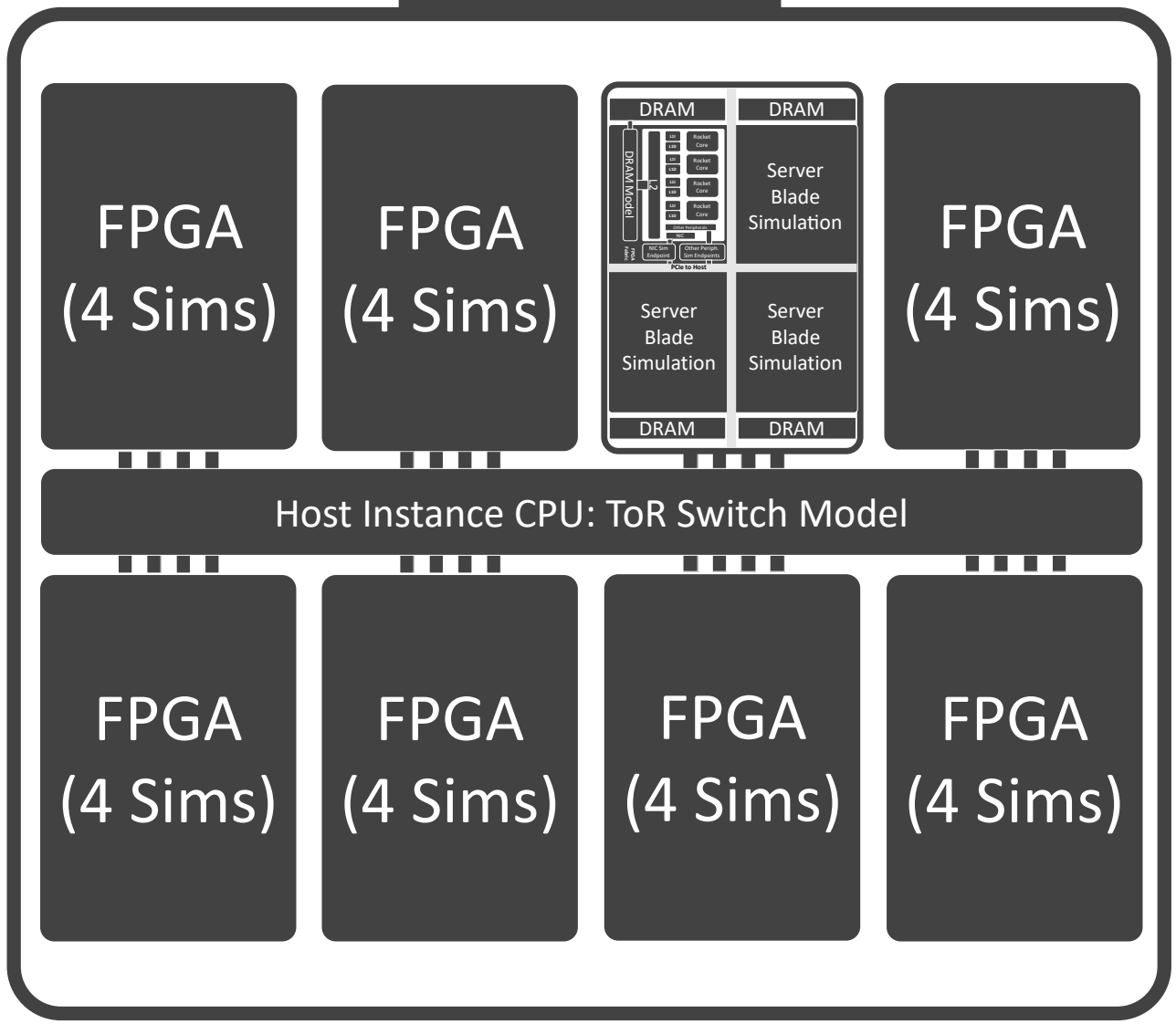
Resource Util.

- 8 FPGAs =
- 1x f1.16xlarge

Sim Rate

- ~10.7 MHz (netw)

Step 4: Simulating a 32 node rack



Modeled System

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

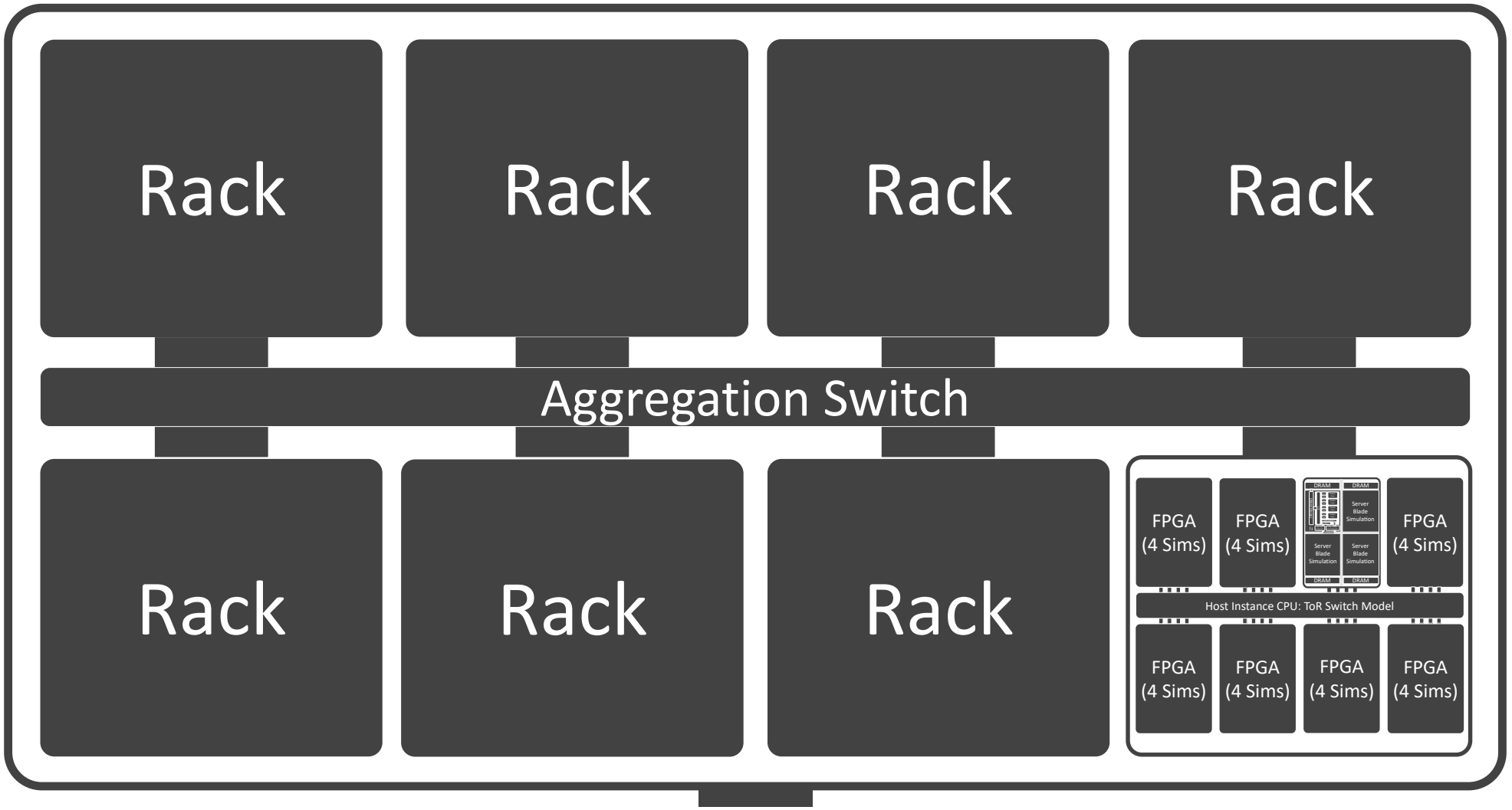
Resource Util.

- 8 FPGAs =
- 1x f1.16xlarge

Sim Rate

- ~10.7 MHz (netw)

Step 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

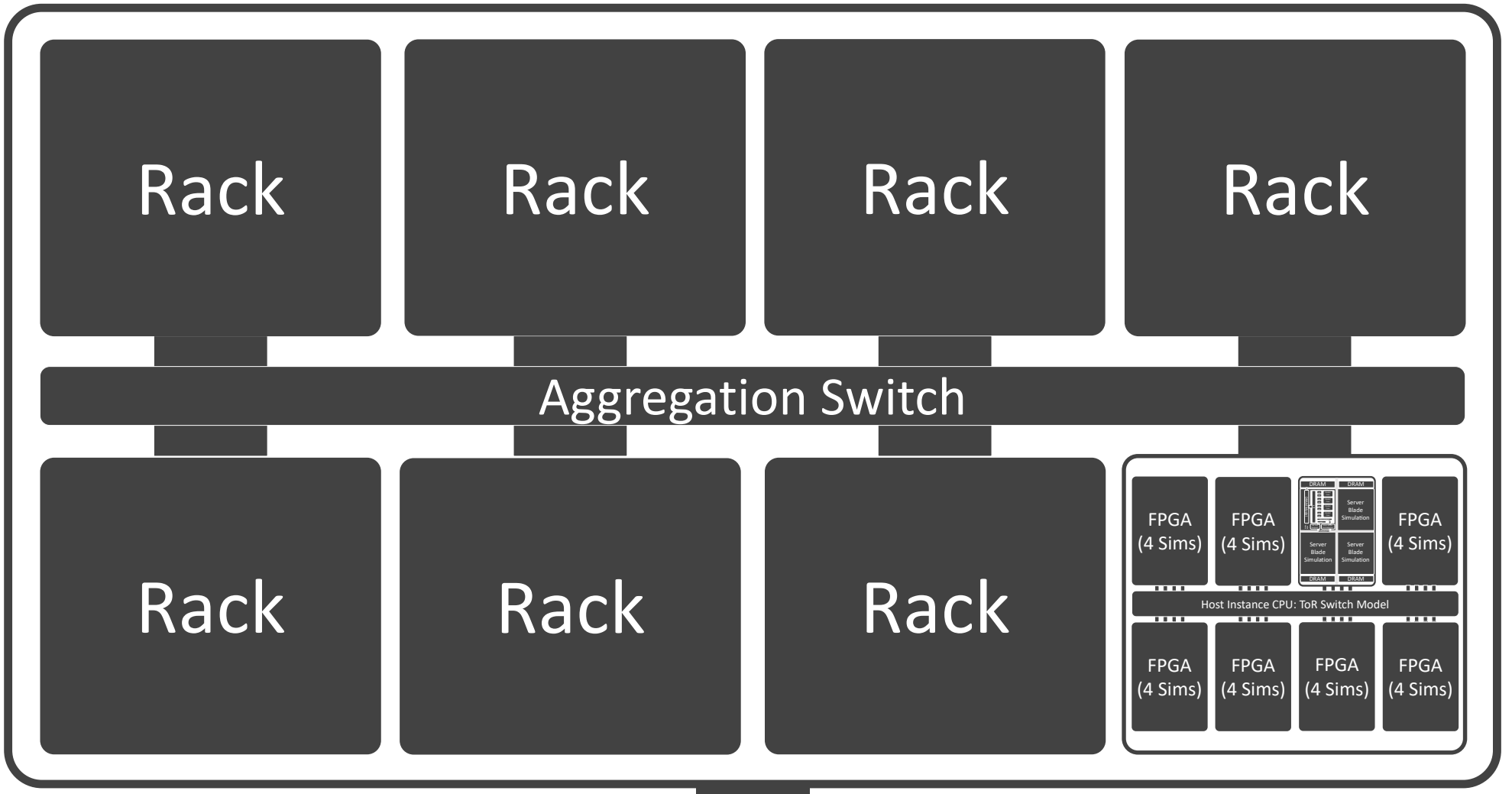
Resource Util.

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

Sim Rate

- ~9 MHz (netw)

Step 5: Simulating a 256 node “aggregation pod”



Modeled System

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

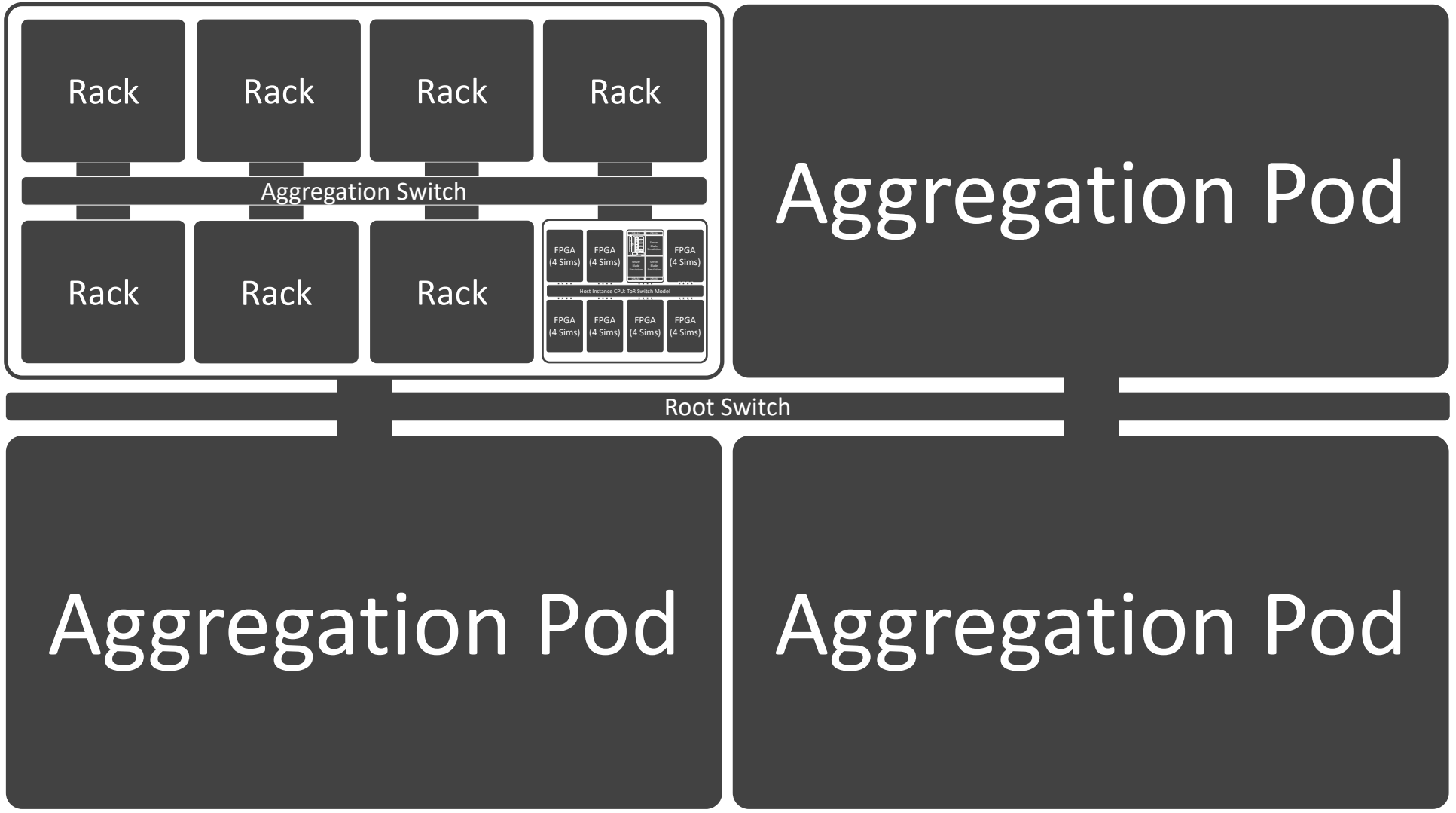
Resource Util.

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

Sim Rate

- ~9 MHz (netw)

Step 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

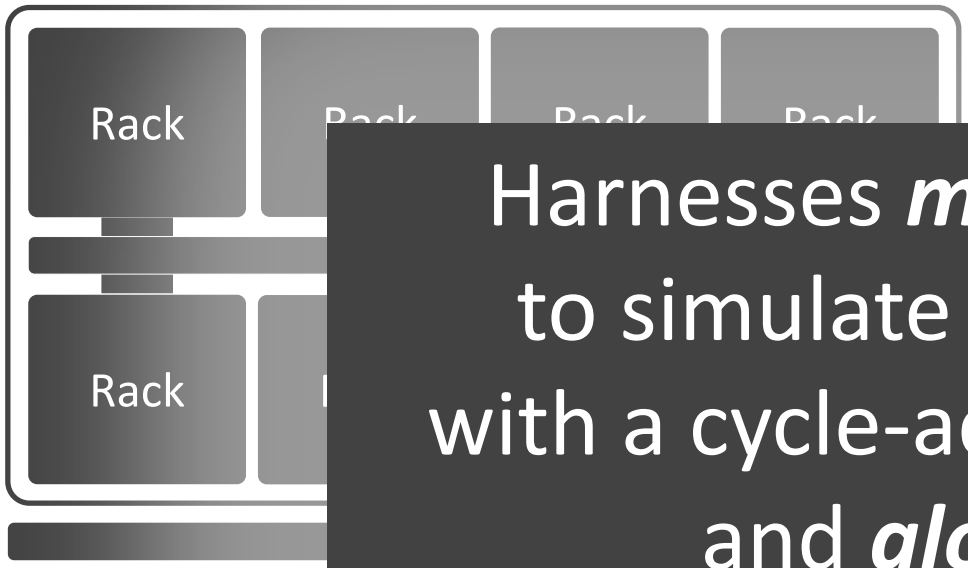
Resource Util.

- 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***

Aggregation Pod

Aggregation Pod

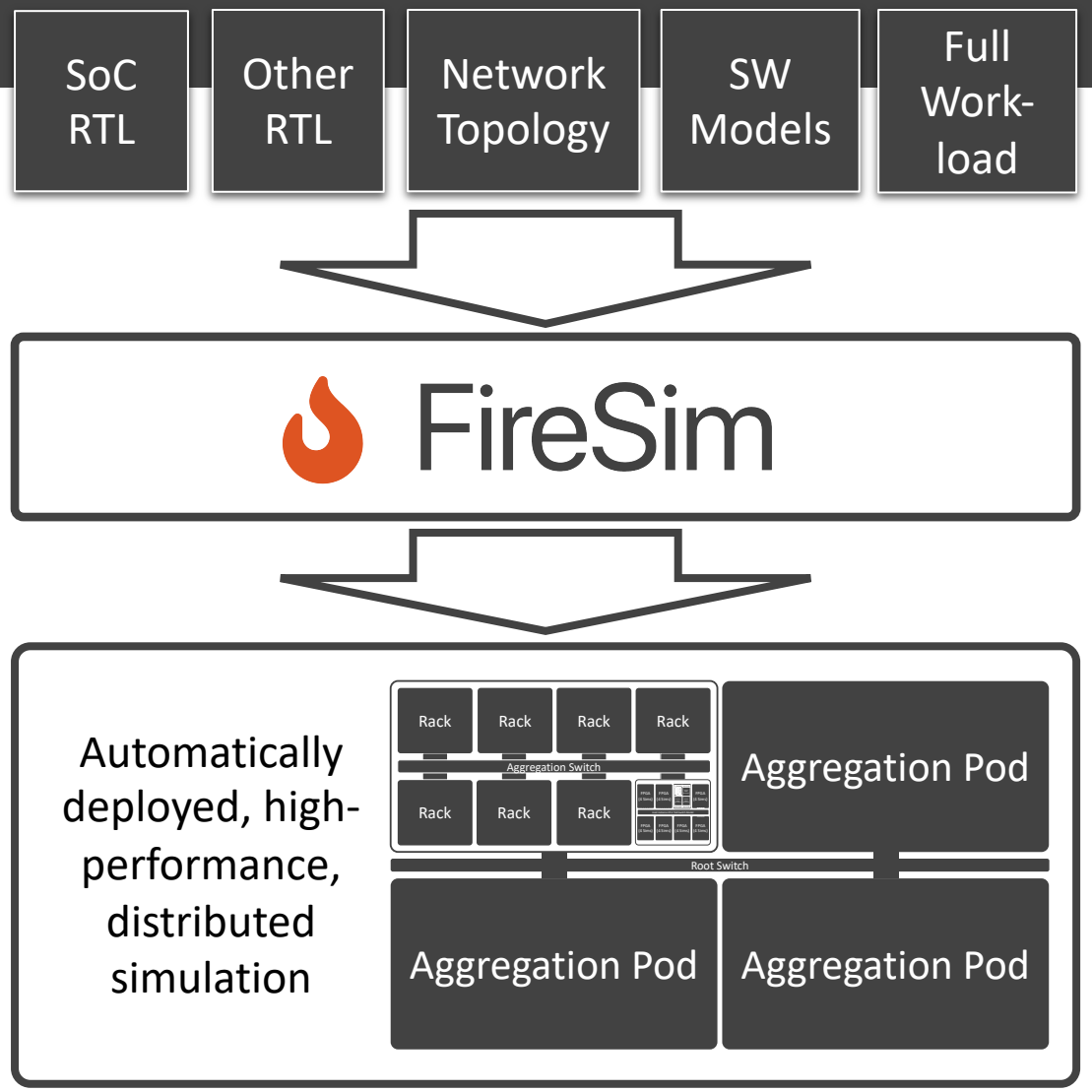
Modeled System
- 1024 Servers
- 6 Cores
- TB DDR3
- oRs, 4 Aggr, 1
- Gb/s, 2us

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

Sim Rate
- ~6.6 MHz (netw)

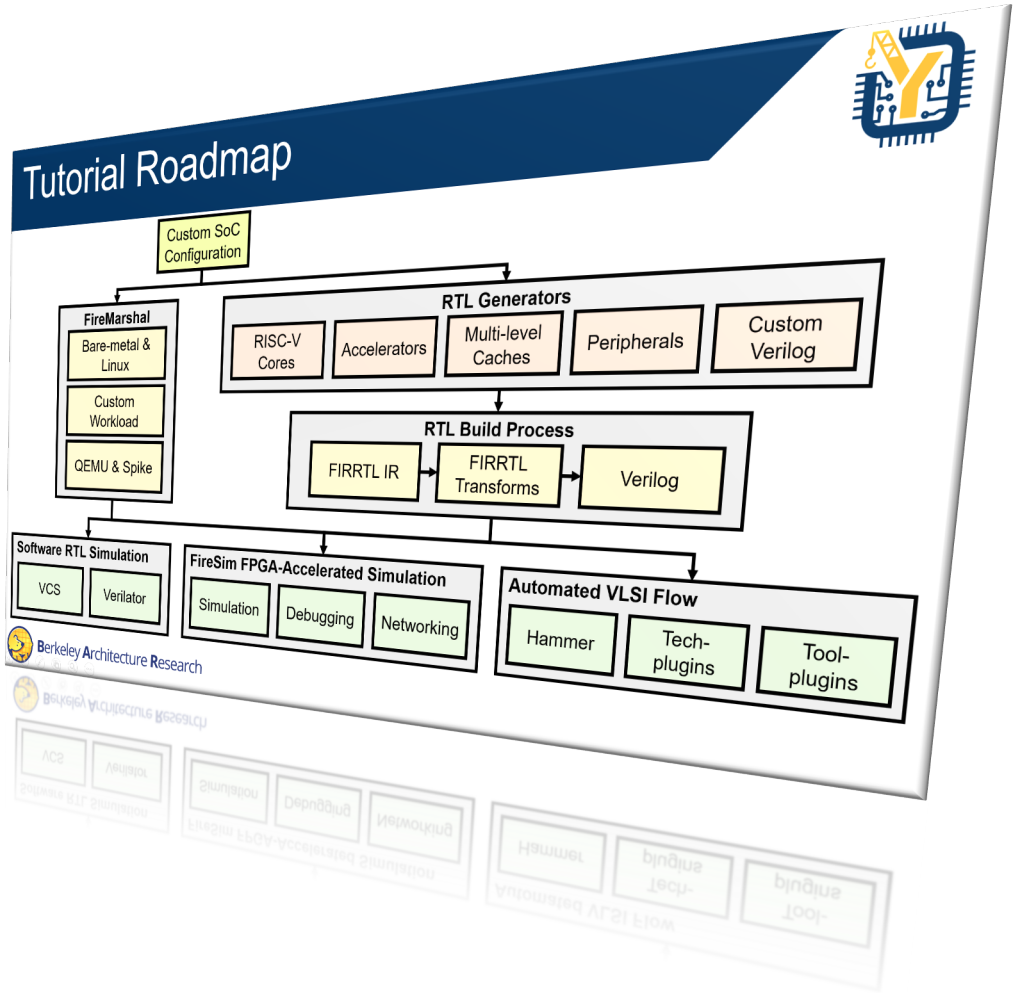
FireSim Recap

- We can prototype scalable-systems built on **arbitrary RTL** at **unprecedented scale**
  - + Mix software models when desired
- Simulation is **automatically built and deployed**
- Automatically **deploy real workloads** and collect results
- **Open-source**, runs on Amazon EC2 F1, **no capex**
- **Evaluate, Debug, Scale-out**

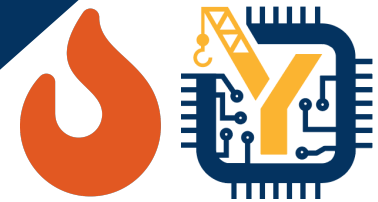


Recap

- End-to-End Evaluation
- Chipyard
  - Multi-flow infrastructure
  - Composing SoC using generators
  - Adding and simulating custom accelerators
  - Hammer VLSI flow
- FireSim
  - Full-system FPGA-accelerated simulation
  - Compiler and manager components
  - Debugging and instrumentation
  - Network simulation



Learn More



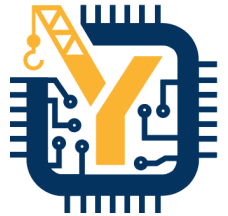
- Chipyard
  - GitHub: https://github.com/ucb-bar/chipyard/
  - Docs: https://chipyard.readthedocs.io/en/latest/index.html
  - Mailing List: https://groups.google.com/forum/#!forum/chipyard

- FireSim
  - Website: https://fires.im/
  - GitHub: https://github.com/firesim/firesim/
  - Docs: https://docs.fires.im/en/latest/
  - Mailing List: https://groups.google.com/forum/#!forum/firesim

Amazon Web Services (AWS)

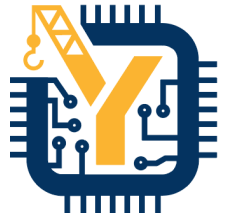
Berkeley
Architecture
Research

AWS and FireSim



- AWS benefits for FPGA-accelerated simulation
  - Pay for usage - No capital investment for large expensive FPGA
  - Elasticity - Enable to run many parallel simulations
  - Collaborative infrastructure – ya'll are going to use an FPGA image/bitstream that the course staff prepared – sharing is relatively easy
- In a research setting, users may have a large amount of AWS credits. In a class setting, we need to be more resource-conscious
- Initial FireSim AWS infrastructure setup
  - Takes a while, but it's a one-time cost
  - Setup procedure designed to minimize potential for mistakes
  - We'll now go through it step-by-step

Lab 3 Instructions – AWS Setup



Set-up an AWS account, based on the instructions in the matching page on the FireSim docs

1 Introduction

This lab will provide you with hands-on experience on the implications of mapping large matrix multiplication operations onto 2D systolic array accelerators, and give you experience using the Amazon Web Services (AWS) EC2 public cloud for FPGA-accelerated simulation.

As most neural network models do not fit in on-chip memory, loop blocking/tiling is an important tool in writing a performant neural network implementation. The additional degree of freedom afforded by the scratchpad in ML accelerators requires further planning of data re-use within the scratchpad. Furthermore, the size of the compute array adds an additional constraint on the tiling hierarchy.

There is a wide body of literature on loop blocking and scheduling for standard scalar processors. However, this body of literature is less extensive with regards to on-chip accelerators with dedicated memories that may be connected to the memory hierarchy in various forms.

Specifically, the DMA of the Gemmini accelerator is currently connected to the shared L2 cache of the scalar host processor. As such, it may see caching affects from the tiling scheme, as well as other parts of the program.

In this lab you will continue using the Chipyard and Gemmini platforms from the previous lab to further improve the performance of DNN execution using ML accelerators through software optimization.

1.1 FireSim and Amazon Web Services (AWS)

In order to use FireSim on AWS (you will find additional information about FireSim in the next section), you will need to open an AWS account. **AWS is a paid service, so be careful about your usage (more details to follow).** Do not wait with this process until the last minute, because it might have around a 1-day latency due to humans-in-the-loop. The lecture on Wednesday, February 26th, will provide a step-by-step tutorial describing the procedures in this section. We recommend following the instructions in this page[1] of the FireSim documentation.

you will receve an email from us with a $200 AWS credit promo code. This amount should be more than sufficient for the tasks in this assignment, assuming you manage you resources and track your AWS expenses. In addition, you can recieve another promo code of $100 by signing up as a student in AWS Educate[3] with your `berkeley.edu` email address. **Do not** sign up for the AWS Educate Starter Kit, because you will not have access to the FPGA-based F1 instances which are required for this lab. You will need to redeem your AWS promo code credits by following the instructions[4].

Once you have opened an AWS account an applied you promo code for credits, follow the initial FireSim AWS infrastructure setup instructions in this page[5] of the FireSim documentation. **Do not** continue to setting up your manager instance (we will provide you with a prepared manager AMI to shorten some of the build preparation process).

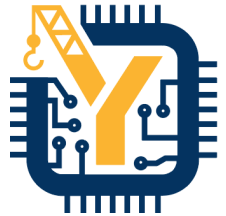
2 Background

2.1 Loop Tiling and Scheduling

We will be working on tiling matrix multiplication, since Gemmini accelerates matrix multiplication operations, and we saw in the previous lab that convolutions can be lowered to matrix multiplication operations. As a reminder, a standard nested-loop matrix multiplication operation looks as follows:

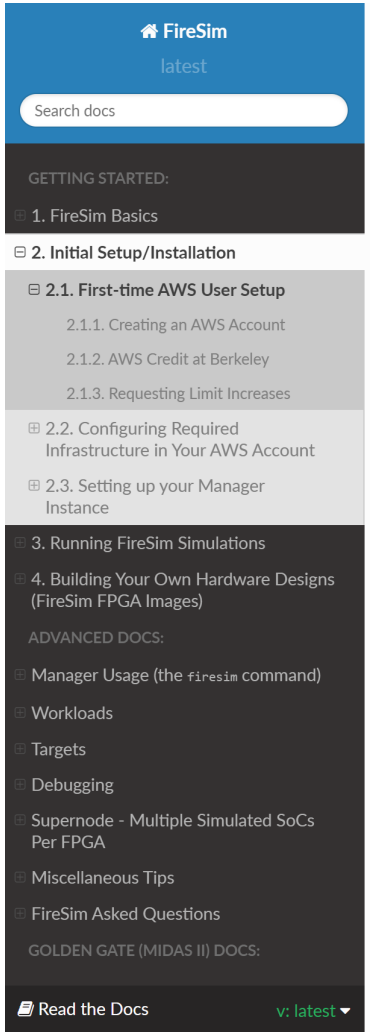
[1] https://docs.fires.im/en/latest/Initial-Setup/First-time-AWS-User-Setup.html
[2] https://forms.gle/YLKirgLGpdzpupk59
[3] https://aws.amazon.com/education/awseducate/
[4] https://aws.amazon.com/awscredits/
[5] https://docs.fires.im/en/latest/Initial-Setup/Configuring-Required-Infrastructure-in-Your-AWS-Account.html

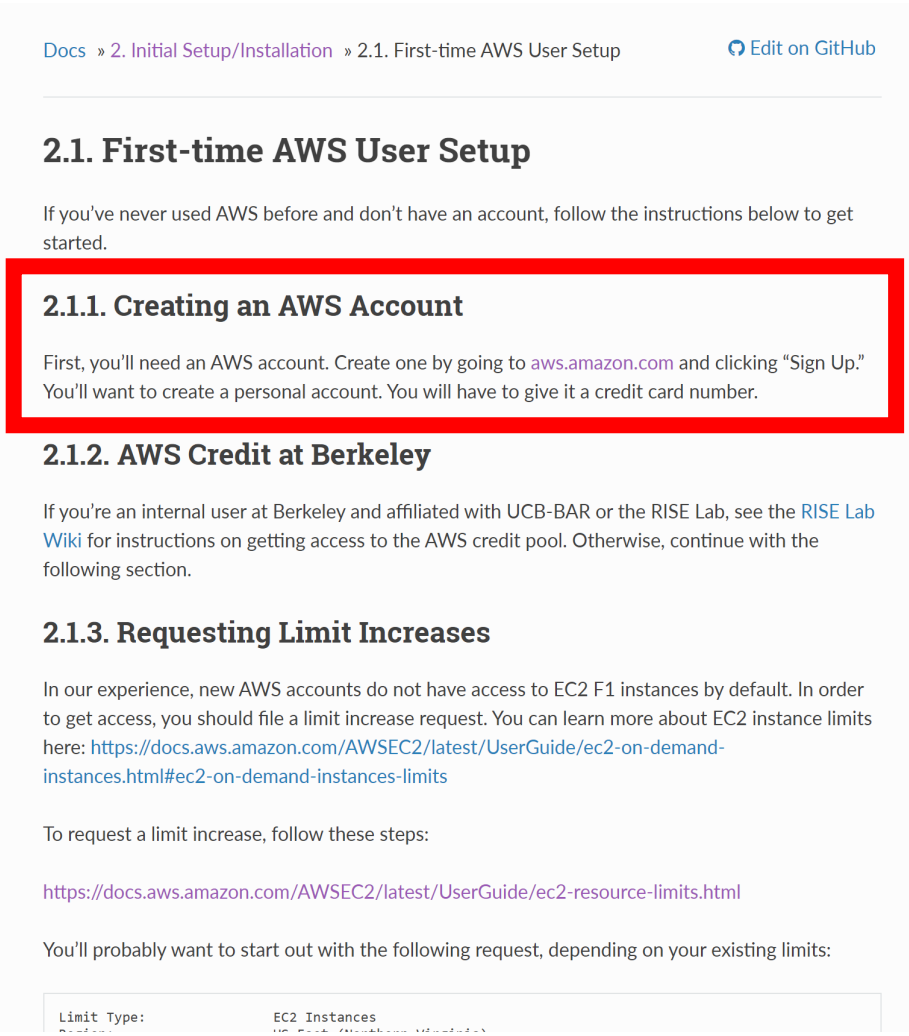
FireSim Docs – AWS User Setup



https://docs.fires.im/

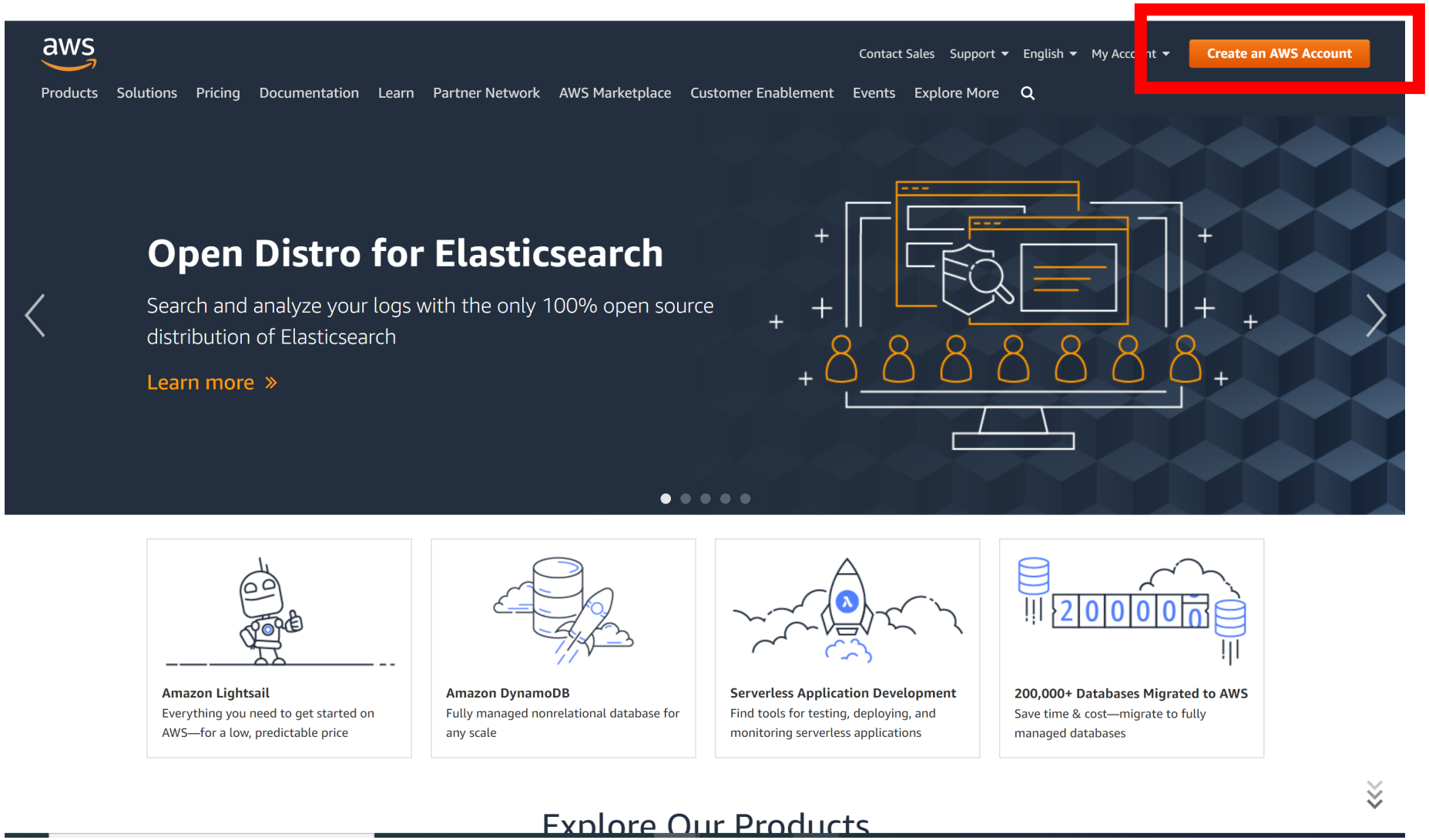
First-time AWS User Setup



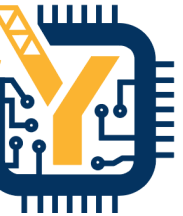


Opening an AWS Account

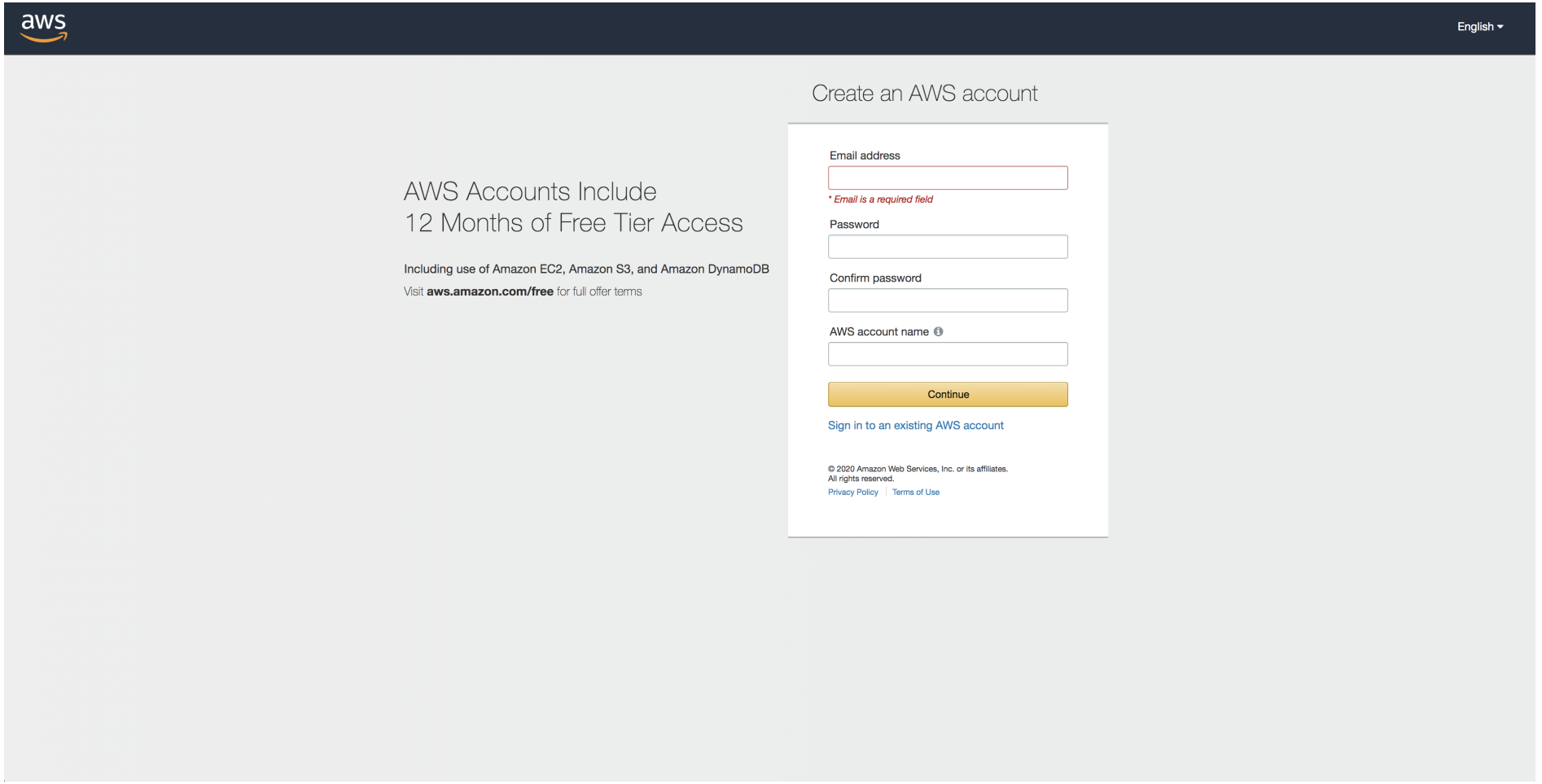
Create an AWS Account



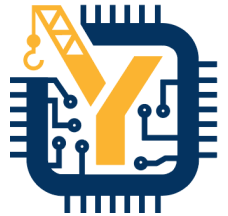
Opening an AWS Account



Sign up details

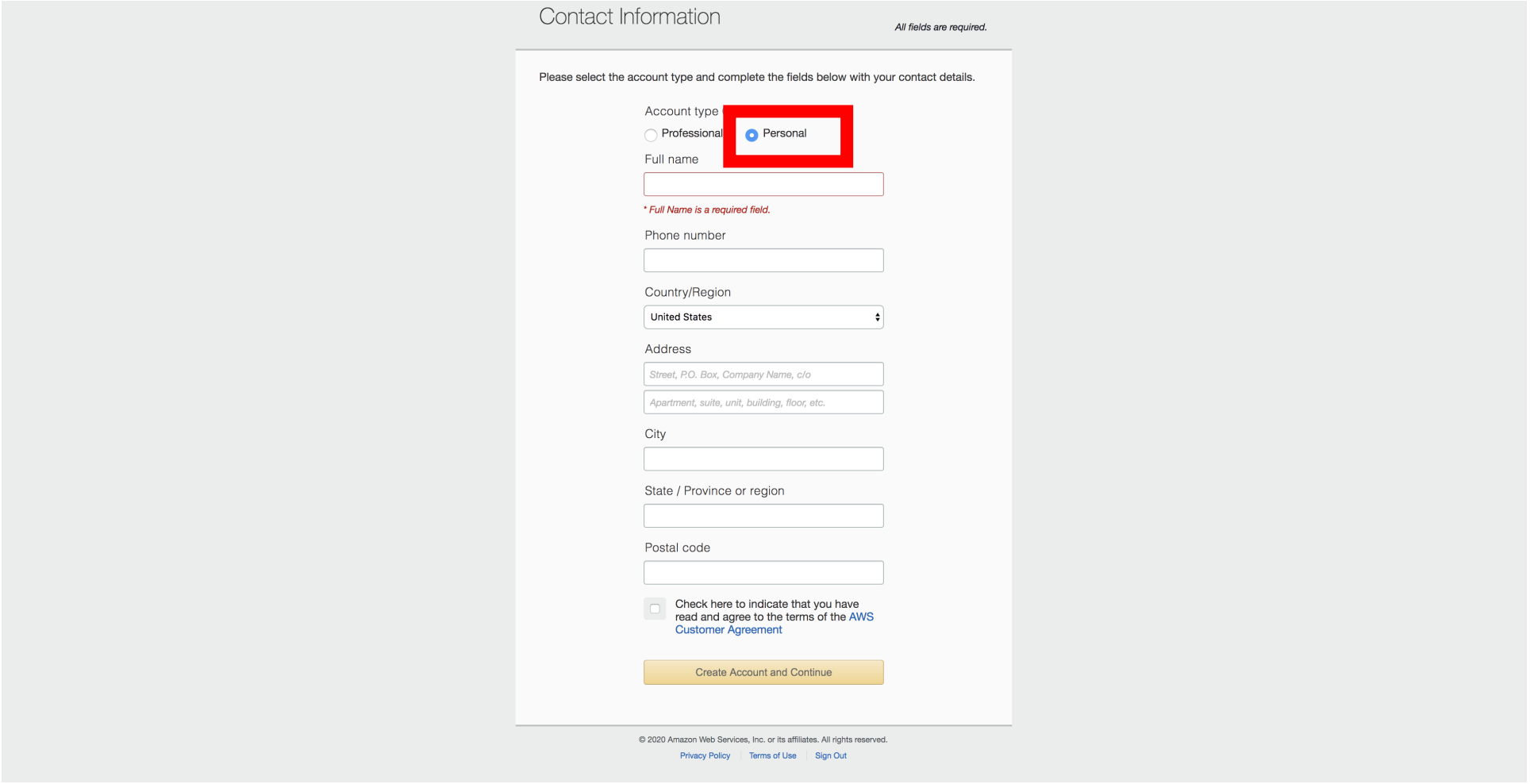


Opening an AWS Account



Additional sign up details

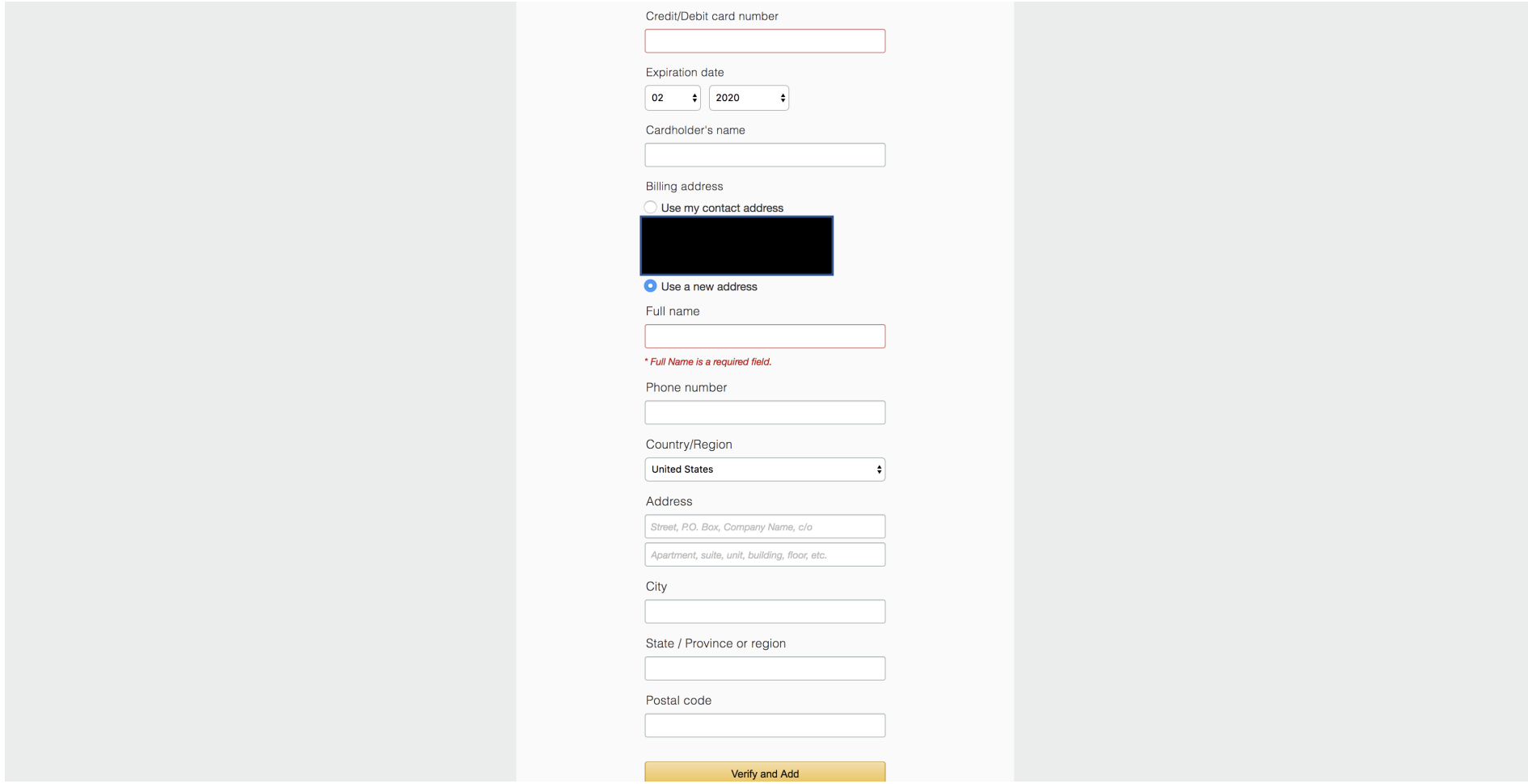
Make sure to pick a **Personal** account



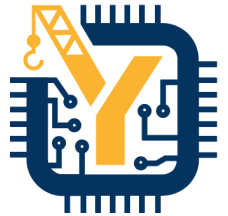
Opening an AWS Account

Payment details:

If you plan on using AWS in the future for other projects (personal or otherwise), you should probably sign up with your credit card. Otherwise….

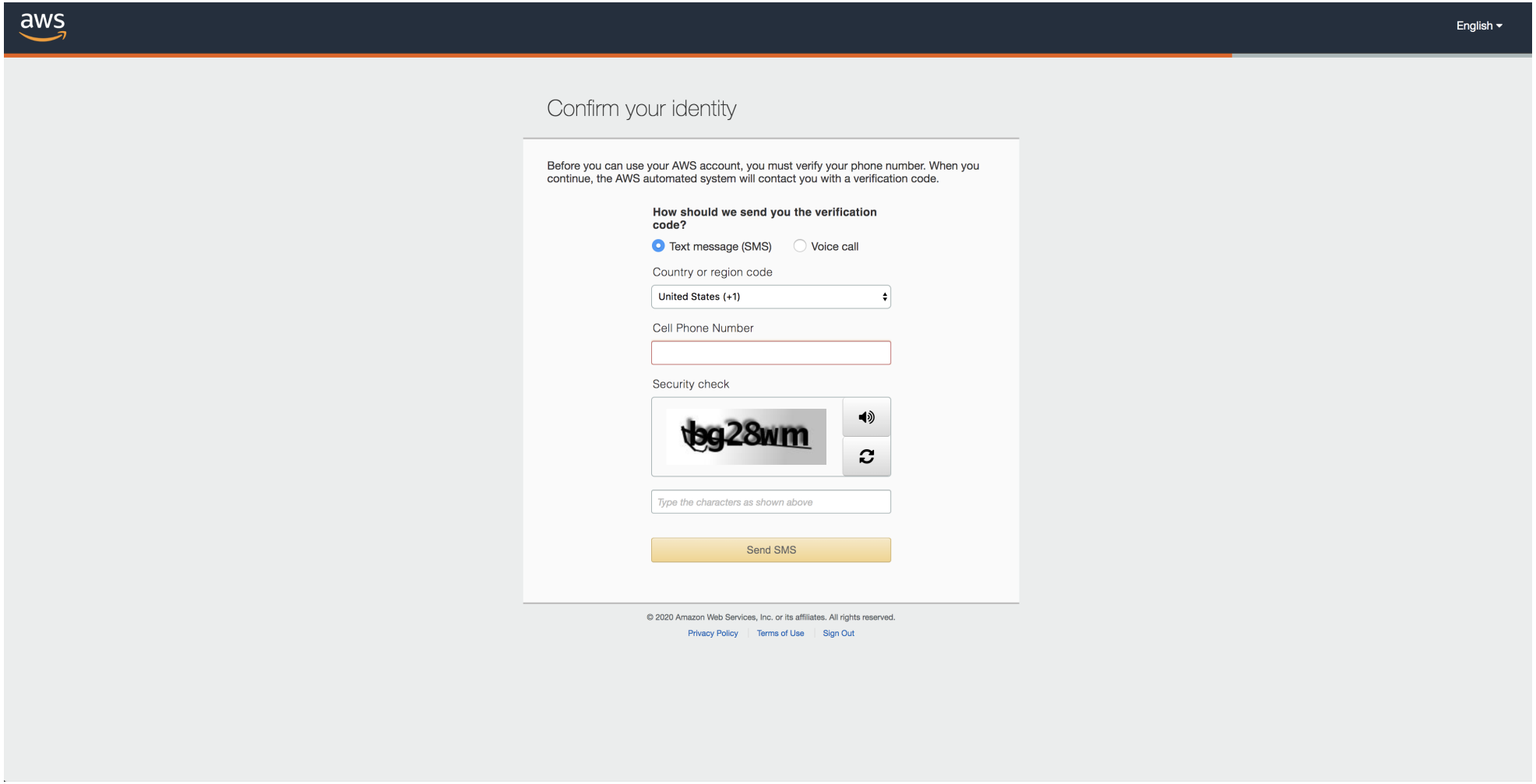


Opening an AWS Account



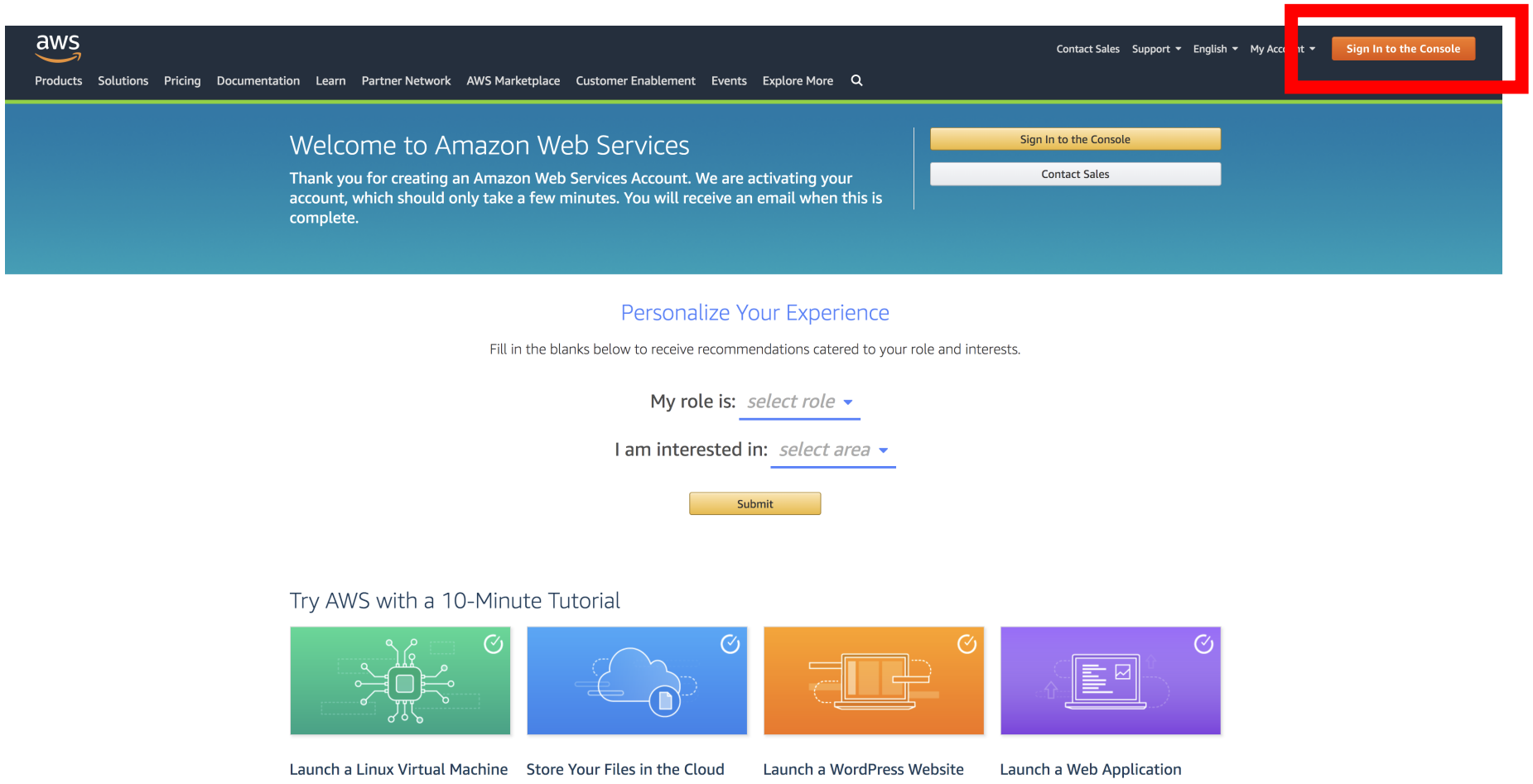
Standard confirmation stuff…

And that should be it for signing up!



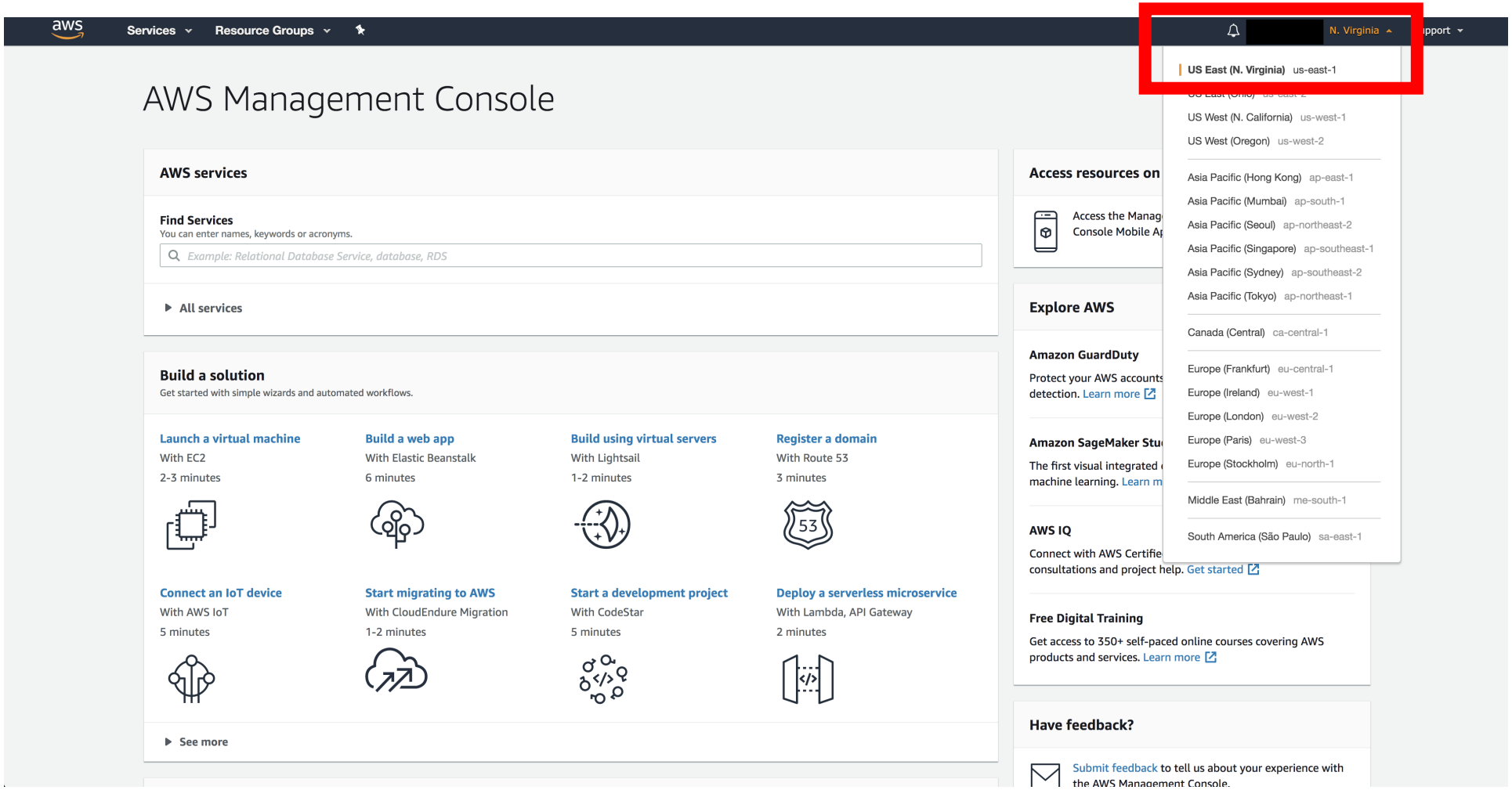
Opening an AWS Account

Log in to your AWS console

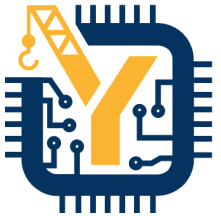


Opening an AWS Account

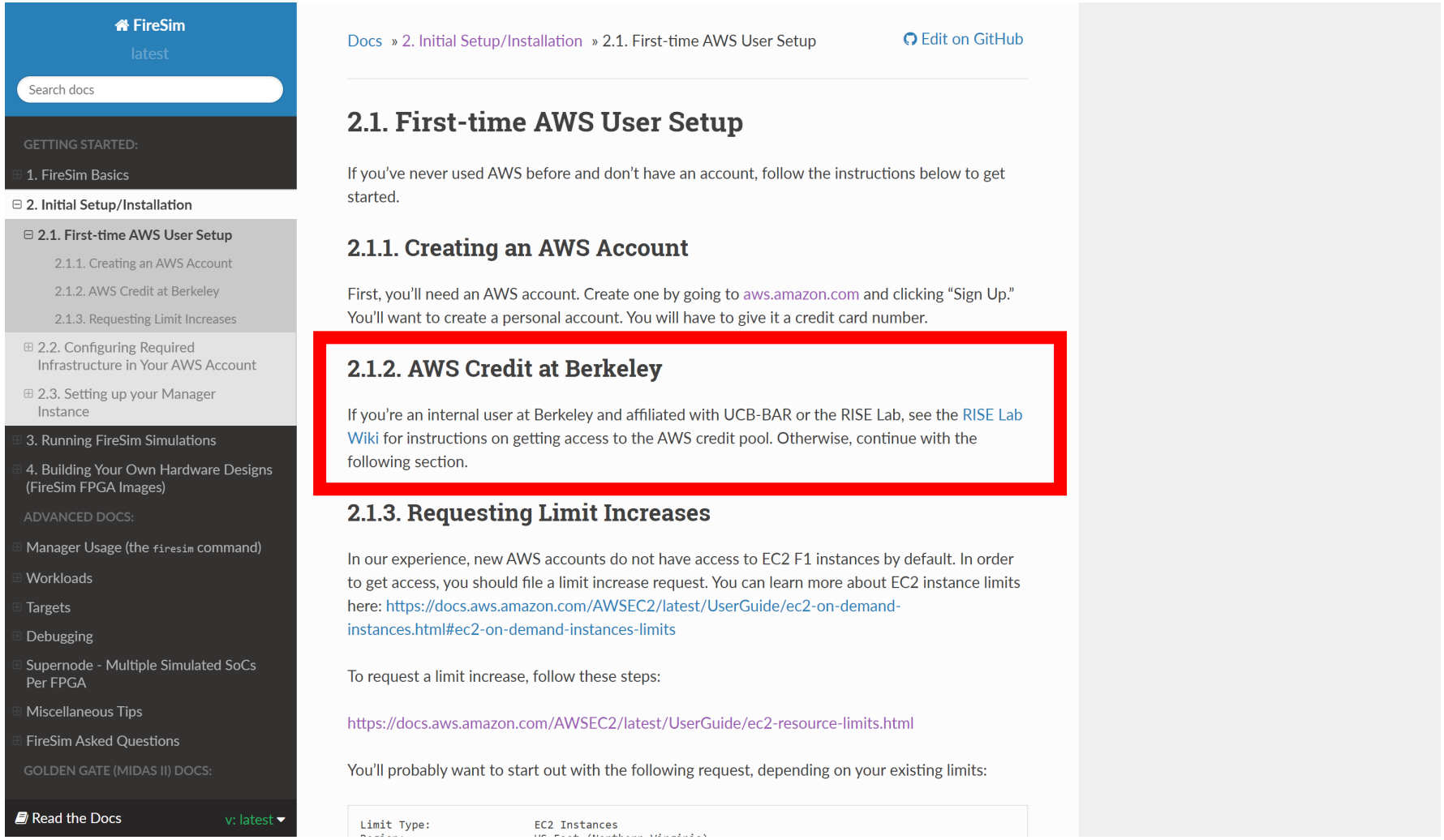
Make sure your region
Is set to US East (North Virginia), a.k.a us-east-1



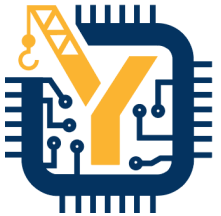
FireSim Docs – AWS User Setup



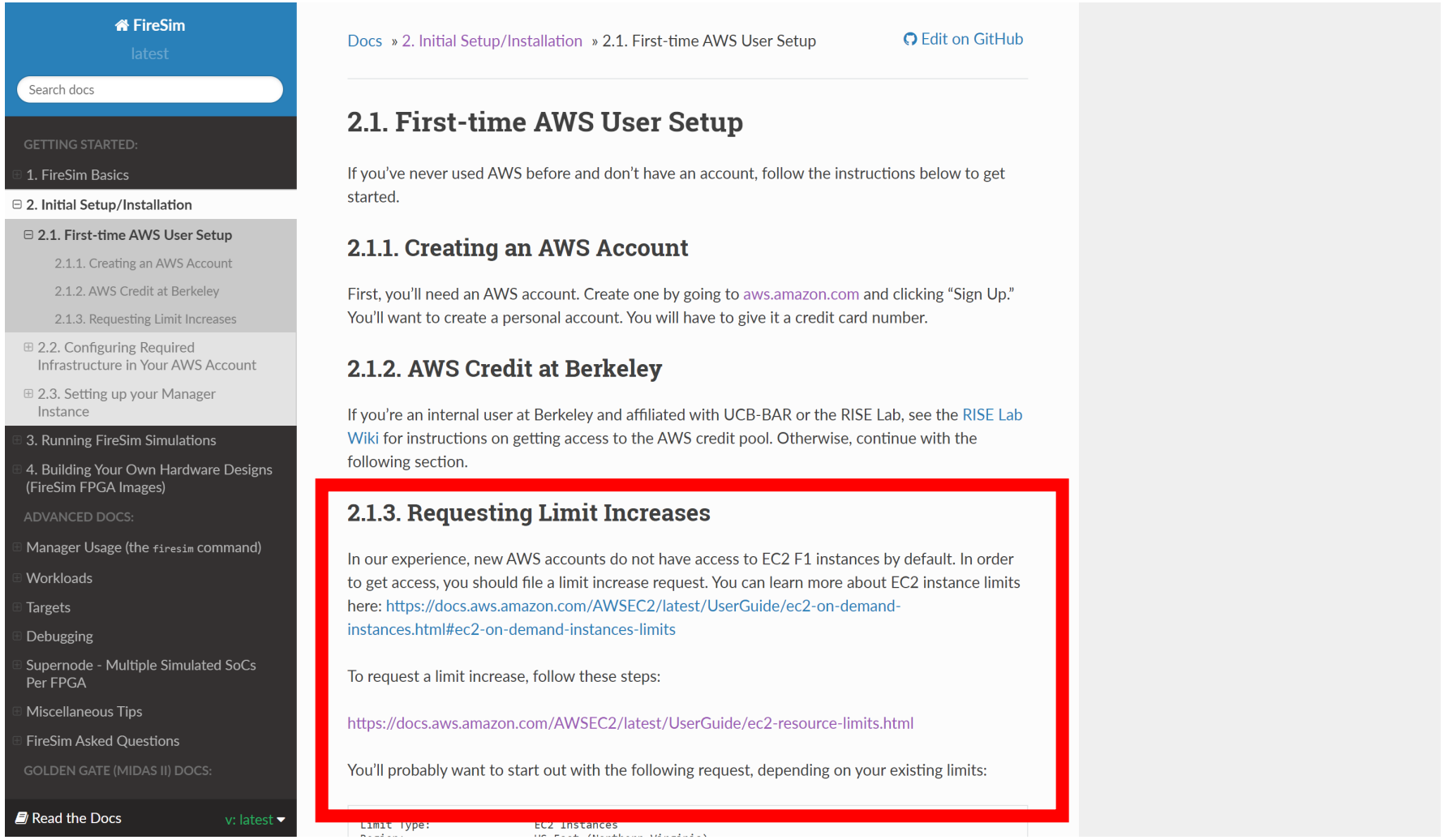
If you are affiliated with a lab that has access to centralized AWS credits, we recommend checking with your lab about associating your account with the lab's central AWS payment pool



FireSim Docs – AWS User Setup



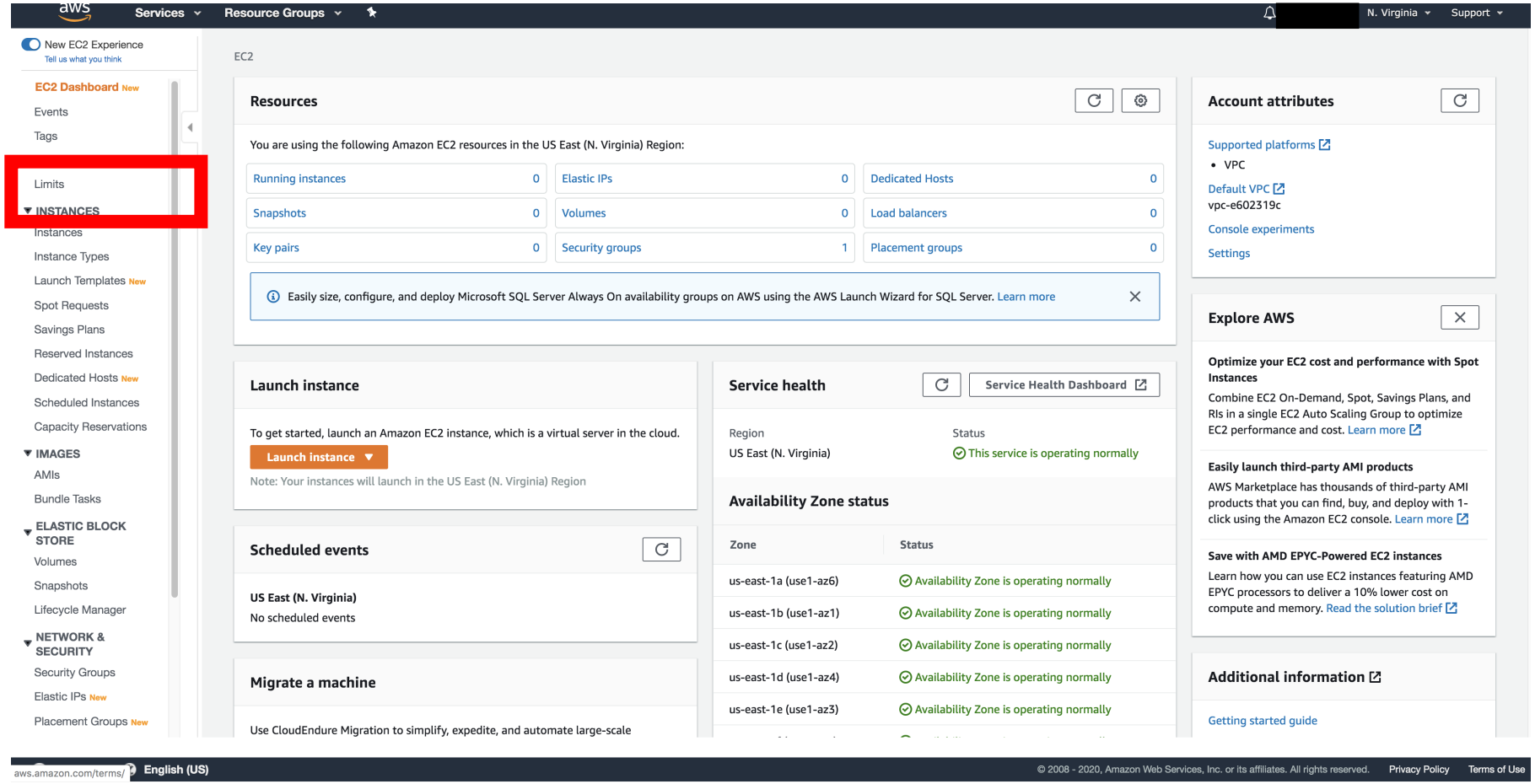
We will be using AWS F1 FPGA instances, which are not a “common” type of instance. Therefore, we need to request a “limit increase” for this type of instance. This process has a human-in-the-loop, so it might take a few hours up to a couple of days



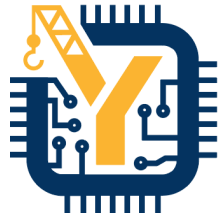
AWS F1 Instance Limit Increase

Access you EC2 dashboard at https://console.aws.amazon.com/ec2

Select “Limits” from the menu on the left

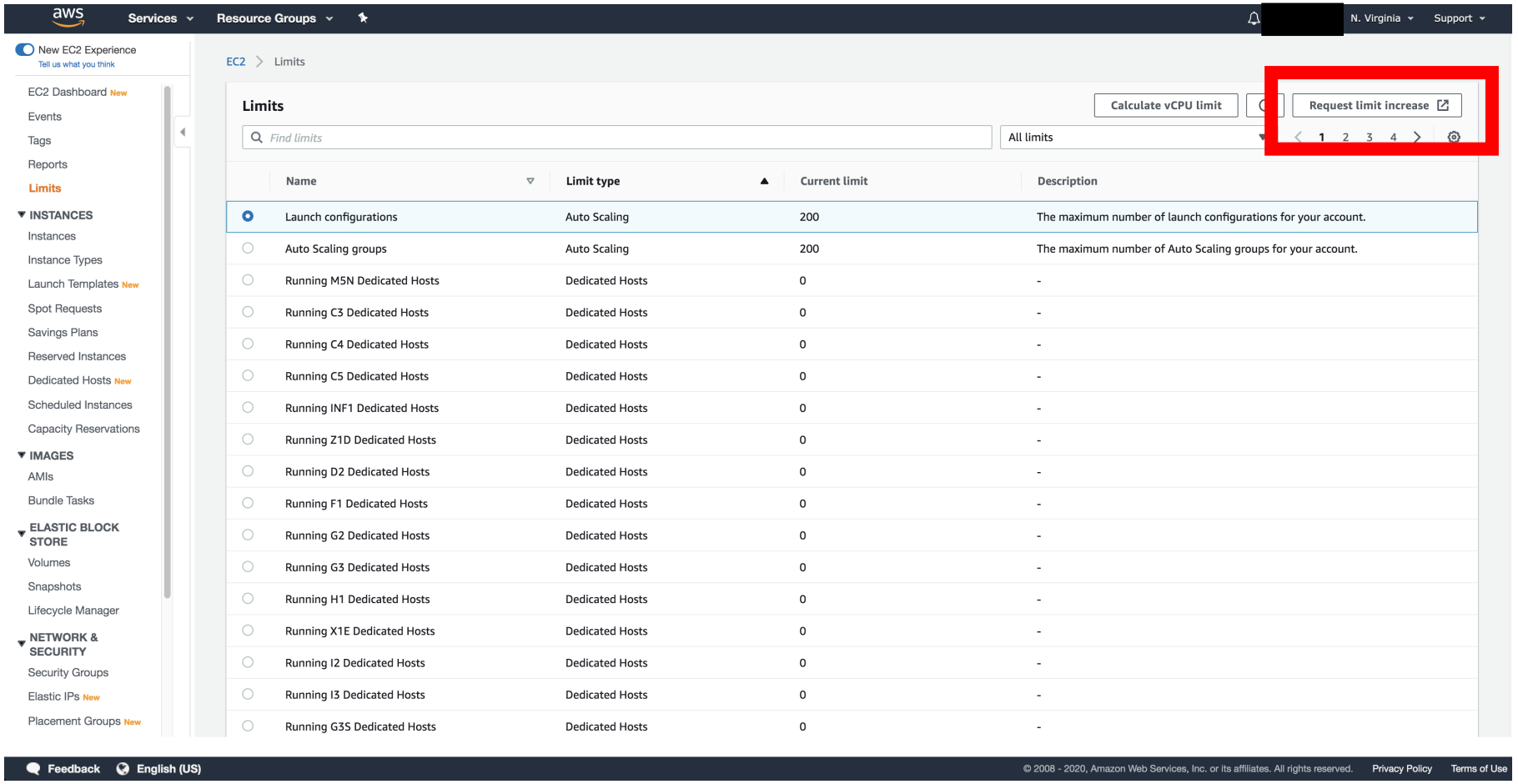


AWS F1 Instance Limit Increase



Select “Request limit increase” on the right side of the window

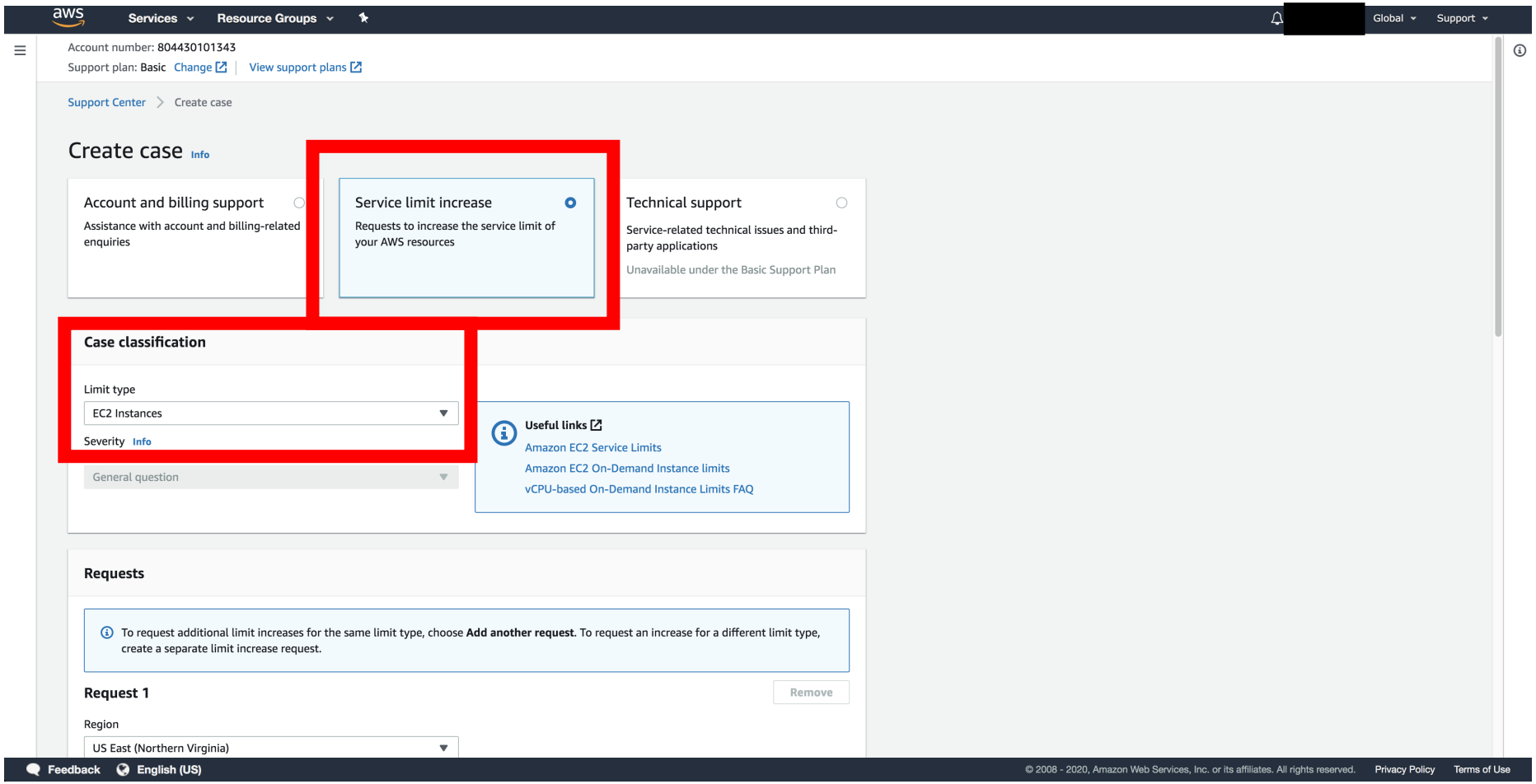
This will open a new support case with AWS



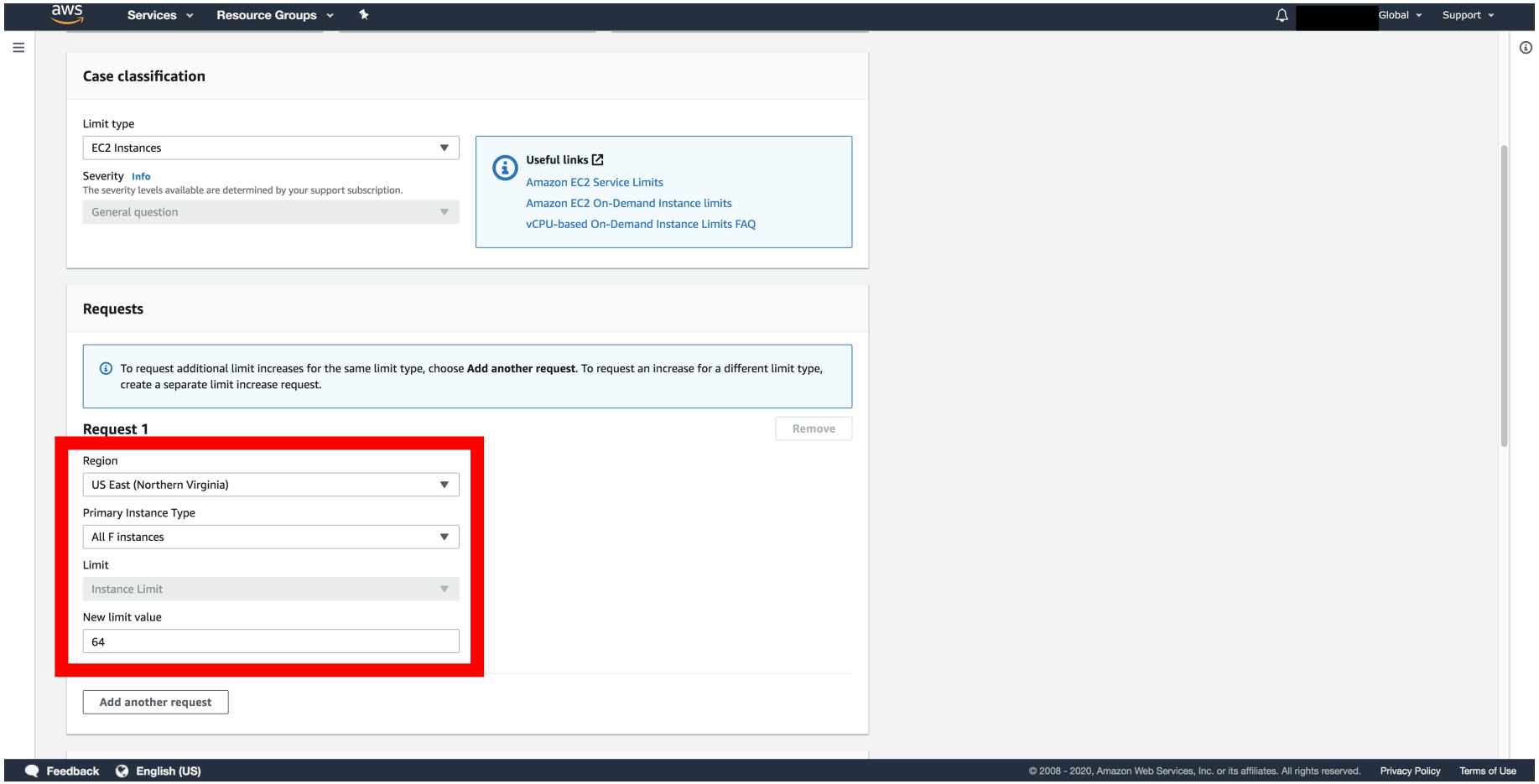
AWS F1 Instance Limit Increase

Select the Service limit increase case type.

For Case classification, select “EC2 Instances”



AWS F1 Instance Limit Increase



Select the following options:

Region: **US East (Northern Virginia)**

Primary Instance Type: **All F Instances**

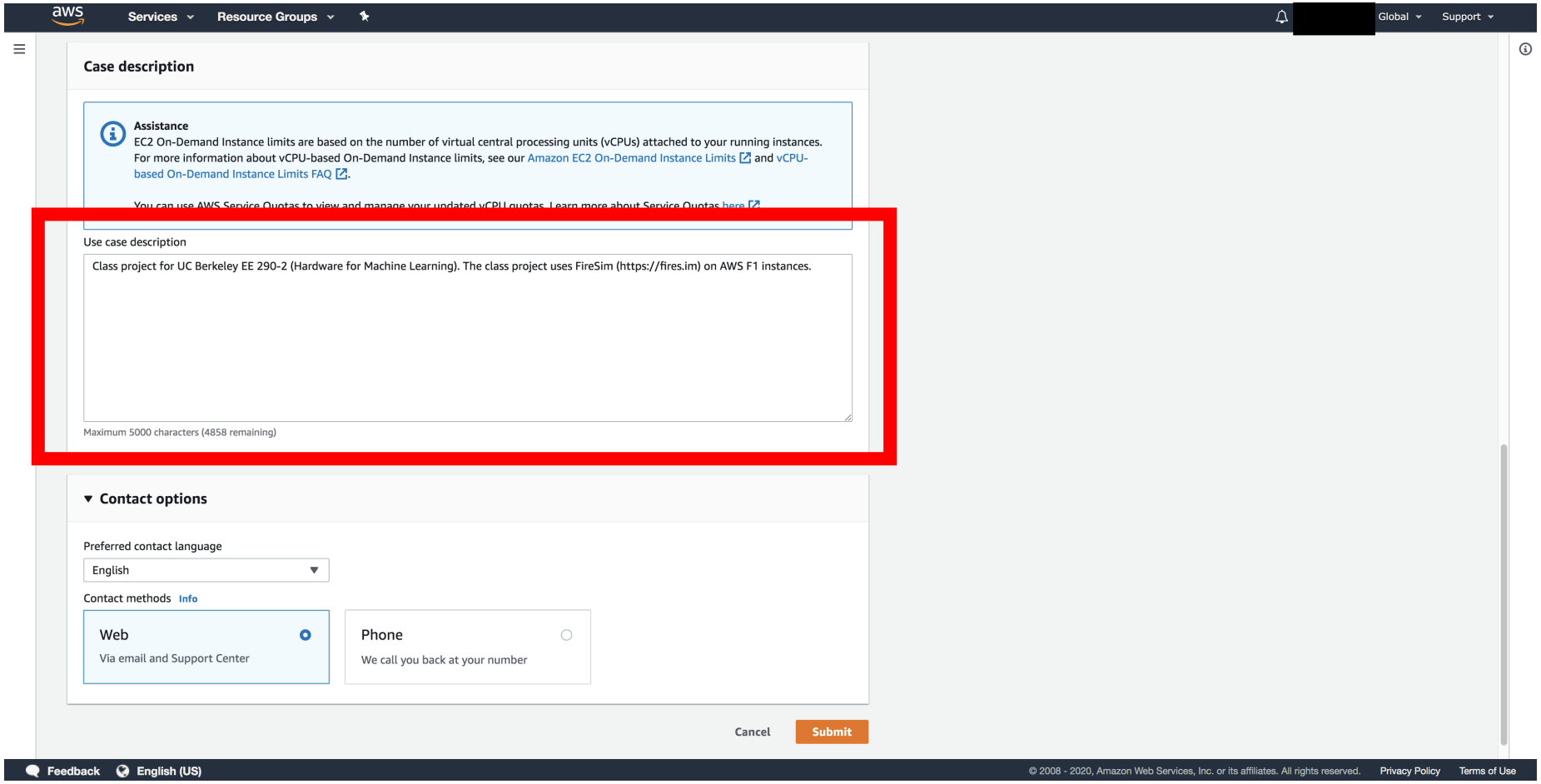
Limit: **Instance Limit**

New limit value: **64**

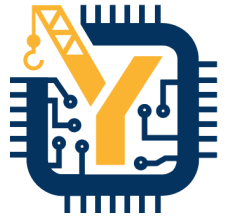
AWS F1 Instance Limit Increase

Use case description:

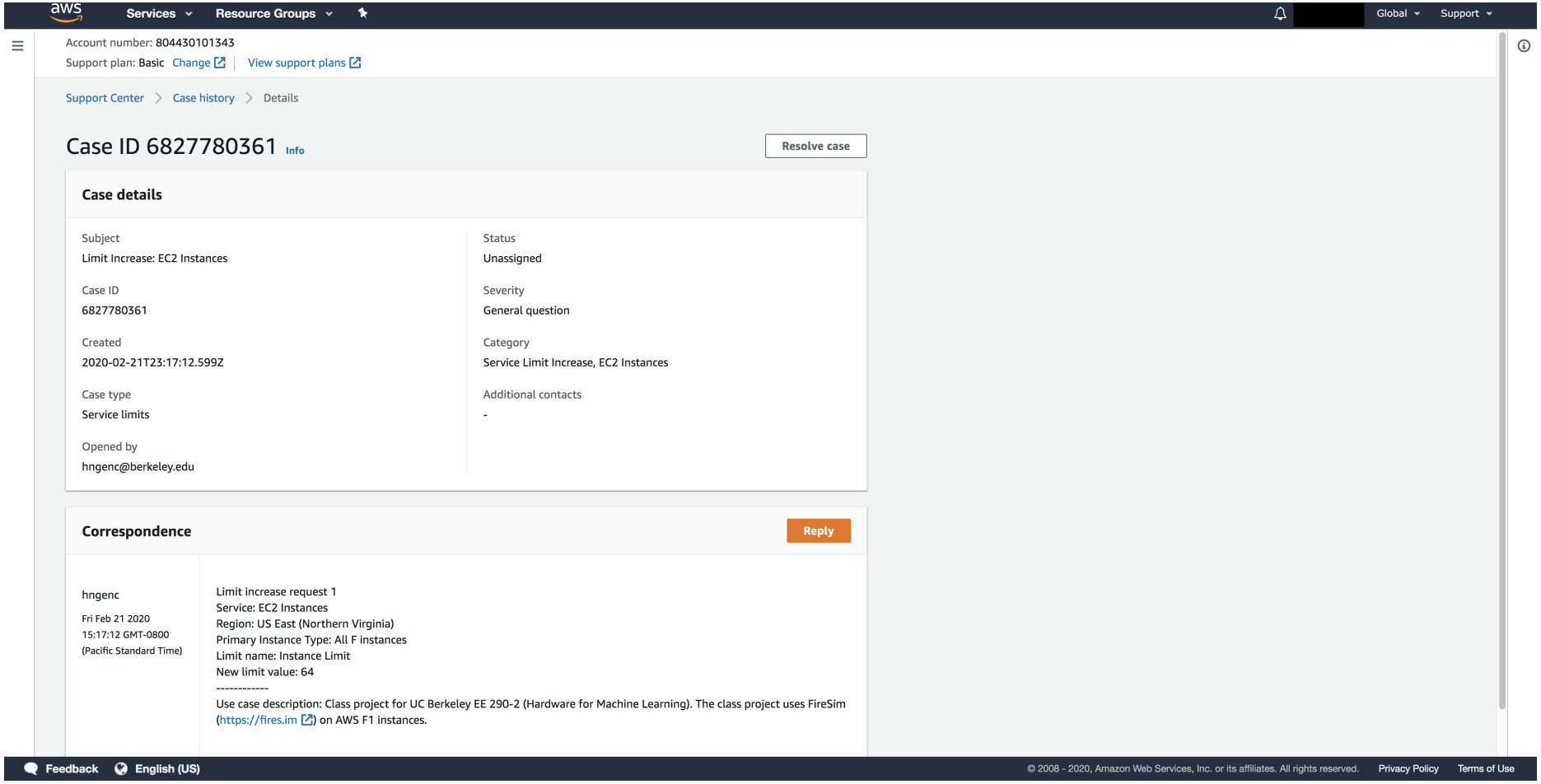
Class project for UC Berkeley EE 290-2 (Hardware for Machine Learning). The class project uses FireSim (https://fires.im) on AWS F1 instances.



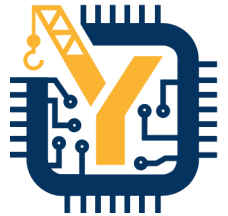
AWS F1 Instance Limit Increase



This will open an AWS support case. You should receive a few confirmation emails.



AWS F1 Instance Limit Increase



One of those emails will tell you the request requires further internal review. This should hopefully not take more than a day.

We are asking up-to 64 F1 instances but if you get less (~16) that should be fine

RE: [CASE 6827780361] Limit Increase: EC2 Instances Inbox ×

no-reply-aws@amazon.com <no-reply-aws@amazon.com> 3:21 PM (4 minutes ago)
to me

Hello!

I am following up to notify you that we've received your limit increase request.

I see that you have requested the following:
[US_EAST_1]: EC2 Instances / Instance Limit (All F instances), New Limit = 64

This specific limit increase request requires further internal review before approval and I have initiated that review.

I understand this increase is important to you and I will do my best to get you a prompt answer.

I will notify you as soon as I have an update. Thank you for your patience while I work with our Service Team.

To contact us again about this case, please return to the AWS Support Center using the following URL:

https://console.aws.amazon.com/support/home#/case/?displayId=6827780361&language=en

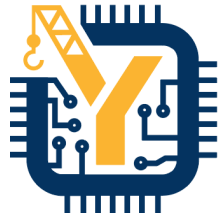
(If you are connecting by federation, log in before following the link.)

*Please note: this e-mail was sent from an address that cannot accept incoming e-mail. Please use the link above if you need to contact us again about this same issue.

==========
Learn to work with the AWS Cloud. Get started with free online videos and self-paced labs at
http://aws.amazon.com/training/
==========

Amazon Web Services, Inc. is an affiliate of Amazon.com, Inc. Amazon.com is a registered trademark of Amazon.com, Inc. or its affiliates.

Lab 3 Instructions – Class AWS Form



Tell us about your AWS account through a Google Form

1 Introduction

This lab will provide you with hands-on experience on the implications of mapping large matrix multiplication operations onto 2D systolic array accelerators, and give you experience using the Amazon Web Services (AWS) EC2 public cloud for FPGA-accelerated simulation.

As most neural network models do not fit in on-chip memory, loop blocking/tiling is an important tool in writing a performant neural network implementation. The additional degree of freedom afforded by the scratchpad in ML accelerators requires further planning of data re-use within the scratchpad. Furthermore, the size of the compute array adds an additional constraint on the tiling hierarchy.

There is a wide body of literature on loop blocking and scheduling for standard scalar processors. However, this body of literature is less extensive with regards to on-chip accelerators with dedicated memories that may be connected to the memory hierarchy in various forms.

Specifically, the DMA of the Gemmini accelerator is currently connected to the shared L2 cache of the scalar host processor. As such, it may see caching affects from the tiling scheme, as well as other parts of the program.

In this lab you will continue using the Chipyard and Gemmini platforms from the previous lab to further improve the performance of DNN execution using ML accelerators through software optimization. You will also be using the FireSim platform (within Chipyard) on the AWS EC2 public cloud.

1.1 FireSim and Amazon Web Services (AWS)

In order to use FireSim on AWS (you will find additional information about FireSim in the next section), you will need to open an AWS account. **AWS is a paid service, so be careful about your usage (more details to follow).** Do not wait with this process until the last minute, because it might have around a 1-day latency due to humans-in-the-loop. The lecture on Wednesday, February 26th, will provide a step-by-step tutorial describing the procedures in this section. We recommend following the

When you are done opening your AWS account, please fill out this form[2]. After you fill out the form, you will receve an email from us with a $200 AWS credit promo code. This amount should be more than sufficient for the tasks in this assignment, assuming you manage you resources and track your AWS expenses. In addition, you can recieve another promo code of $100 by signing up as a student in AWS Educate[3] with your `berkeley.edu` email address. **Do not** sign up for the AWS Educate Starter Kit, because you will not have access to the FPGA-based F1 instances which are required for this lab. You will need to redeem your AWS promo code credits by following the instructions[4].

FireSim AWS infrastructure setup instructions in this page[5] of the FireSim documentation. **Do not** continue to setting up your manager instance (we will provide you with a prepared manager AMI to shorten some of the build preparation process).

2 Background

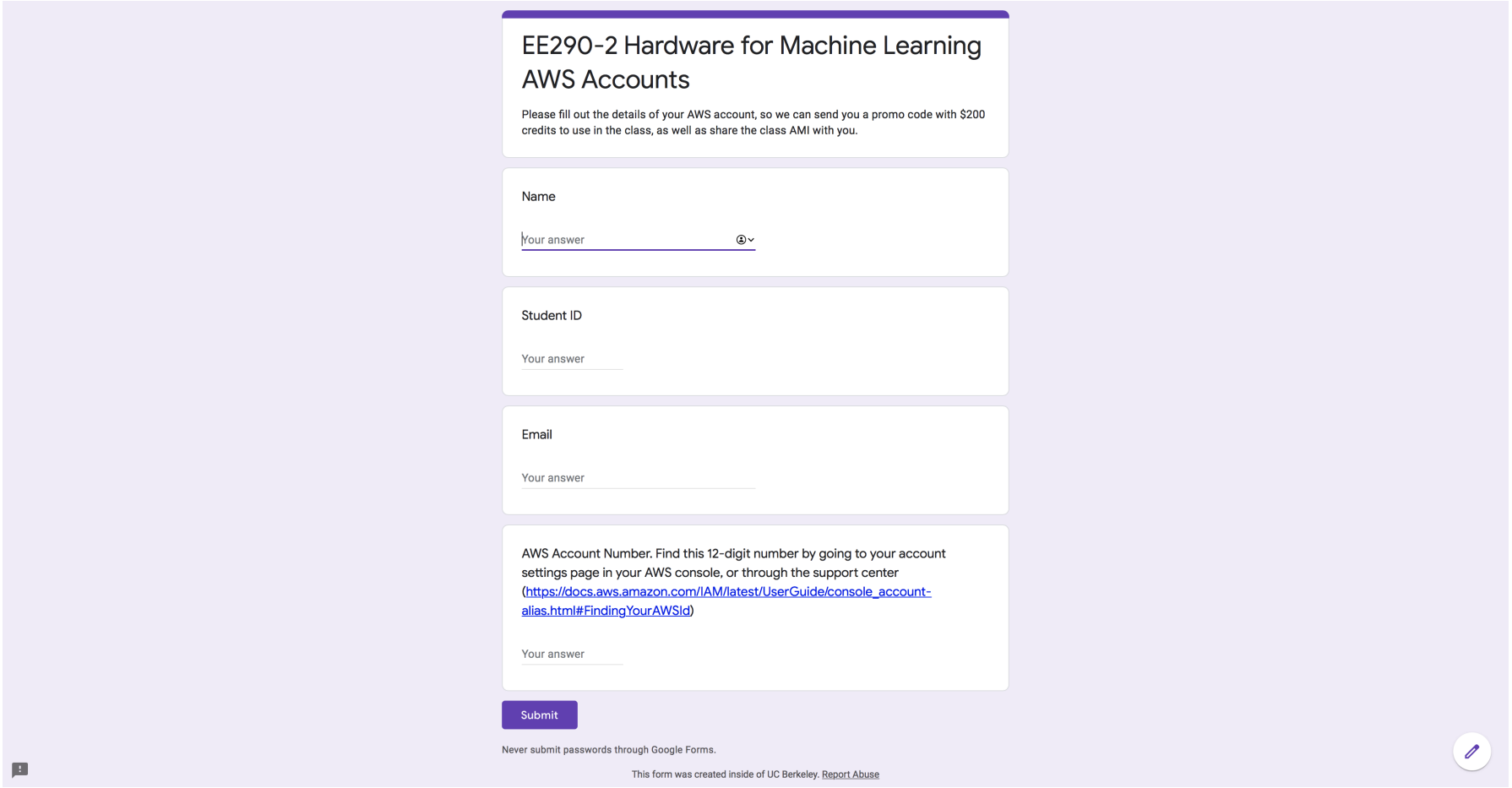
2.1 Loop Tiling and Scheduling

We will be working on tiling matrix multiplication, since Gemmini accelerates matrix multiplication operations, and we saw in the previous lab that convolutions can be lowered to matrix multiplication operations. As a reminder, a standard nested-loop matrix multiplication operation looks as follows:

[1] https://docs.fires.im/en/latest/Initial-Setup/First-time-AWS-User-Setup.html
[2] https://forms.gle/YLKirgLGpdzpupk59
[3] https://aws.amazon.com/education/awseducate/
[4] https://aws.amazon.com/awscredits/
[5] https://docs.fires.im/en/latest/Initial-Setup/Configuring-Required-Infrastructure-in-Your-AWS-Account.html

Berkeley Architecture Research

Tell Us About Your AWS Account

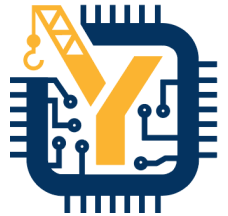


Fill the form at https://forms.gle/ALvSrq9NK32xmWXy9

We need you to tell us about you AWS account for two reasons:

1. So we can send you a promo code for $200 credits
2. So we can share with you the prebuild AMI (image)

Tell Us About Your AWS Account



Within a day (hopefully less), you will get an email from us with an AWS promo code

EE290 AWS Promo Code Inbox ×

Alon Amid
to me

Hello Hasan

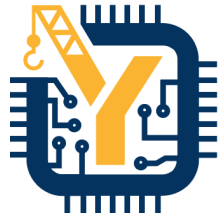
This is your AWS Promo code:

PC10UOS07VQN4O2

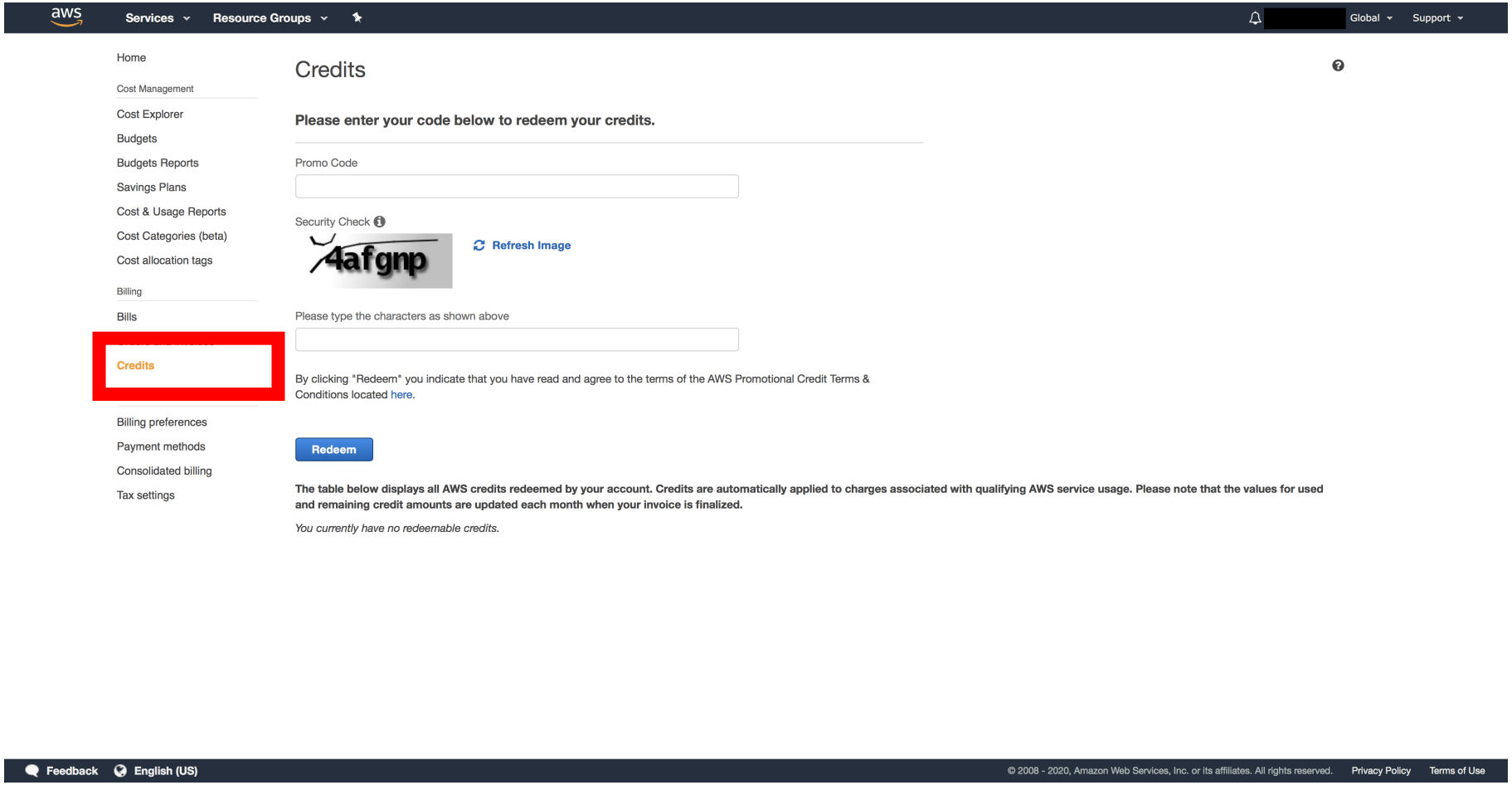
Best,
EE290 Staff

Reply Forward

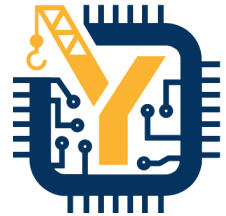
Redeem AWS Credits



In your AWS billing console/dashboard, select Credits, fill the promo code, and Redeem



Lab 3 Instructions –FireSim Infra. Setup



Once you have redeemed your credits, continue setting up the FireSim infrastructure

1 Introduction

This lab will provide you with hands-on experience on the implications of mapping large matrix multiplication operations onto 2D systolic array accelerators, and give you experience using the Amazon Web Services (AWS) EC2 public cloud for FPGA-accelerated simulation.

As most neural network models do not fit in on-chip memory, loop blocking/tiling is an important tool in writing a performant neural network implementation. The additional degree of freedom afforded by the scratchpad in ML accelerators requires further planning of data re-use within the scratchpad. Furthermore, the size of the compute array adds an additional constraint on the tiling hierarchy.

There is a wide body of literature on loop blocking and scheduling for standard scalar processors. However, this body of literature is less extensive with regards to on-chip accelerators with dedicated memories that may be connected to the memory hierarchy in various forms.

Specifically, the DMA of the Gemmini accelerator is currently connected to the shared L2 cache of the scalar host processor. As such, it may see caching affects from the tiling scheme, as well as other parts of the program.

In this lab you will continue using the Chipyard and Gemmini platforms from the previous lab to further improve the performance of DNN execution using ML accelerators through software optimization. You will also be using the FireSim platform (within Chipyard) on the AWS EC2 public cloud.

1.1 FireSim and Amazon Web Services (AWS)

In order to use FireSim on AWS (you will find additional information about FireSim in the next section), you will need to open an AWS account. **AWS is a paid service, so be careful about your usage (more details to follow).** Do not wait with this process until the last minute, because it might have around a 1-day latency due to humans-in-the-loop. The lecture on Wednesday, February 26th, will provide a step-by-step tutorial describing the procedures in this section. We recommend following the instructions in this page[1] of the FireSim documentation.

When you are done opening your AWS account, please fill out this form[2]. After you fill out the form, you will receve an email from us with a $200 AWS credit promo code. This amount should be more than sufficient for the tasks in this assignment, assuming you manage you resources and track your AWS expenses. In addition, you can recieve another promo code of $100 by signing up as a student in AWS Educate[3] with your `berkeley.edu` email address. **Do not** sign up for the AWS Educate Starter Kit, because you will not have access to the FPGA-based F1 instances which are required for this lab. You

Once you have opened an AWS account an applied you promo code for credits, follow the initial FireSim AWS infrastructure setup instructions in this page[5] of the FireSim documentation. **Do not** continue to setting up your manager instance (we will provide you with a prepared manager AMI to shorten some of the build preparation process).

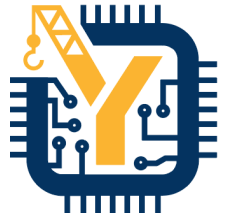
2 Background

2.1 Loop Tiling and Scheduling

We will be working on tiling matrix multiplication, since Gemmini accelerates matrix multiplication operations, and we saw in the previous lab that convolutions can be lowered to matrix multiplication operations. As a reminder, a standard nested-loop matrix multiplication operation looks as follows:

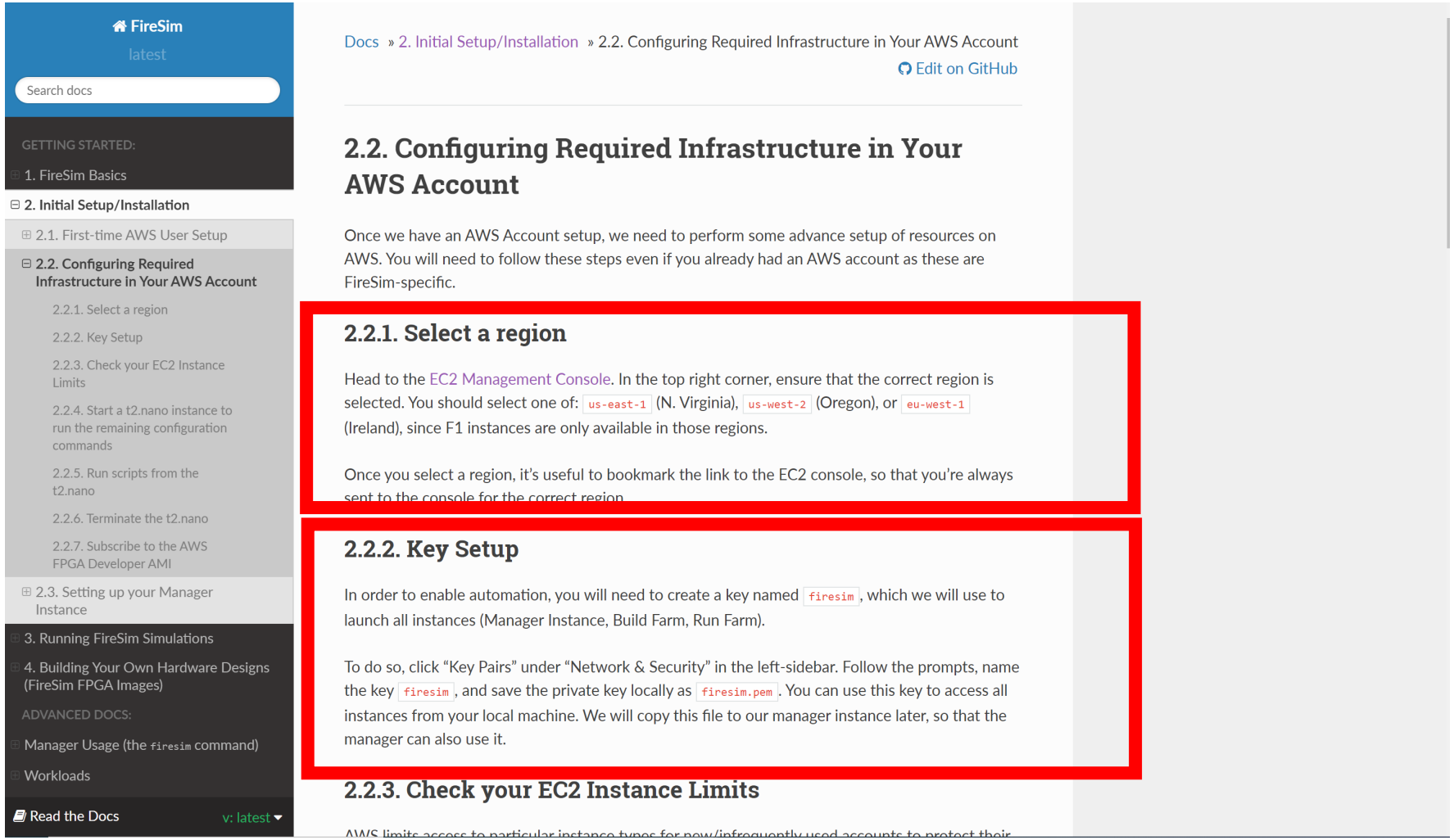
[1] https://docs.fires.im/en/latest/Initial-Setup/First-time-AWS-User-Setup.html
[2] https://forms.gle/YLKirgLGpdzpupk59
[3] https://aws.amazon.com/education/awseducate/
[4] https://aws.amazon.com/awscredits/
[5] https://docs.fires.im/en/latest/Initial-Setup/Configuring-Required-Infrastructure-in-Your-AWS-Account.html

FireSim Docs – Configuring Infrastructure



"Select a region": we already did this (us-east-1, Northern Virgina), remember?

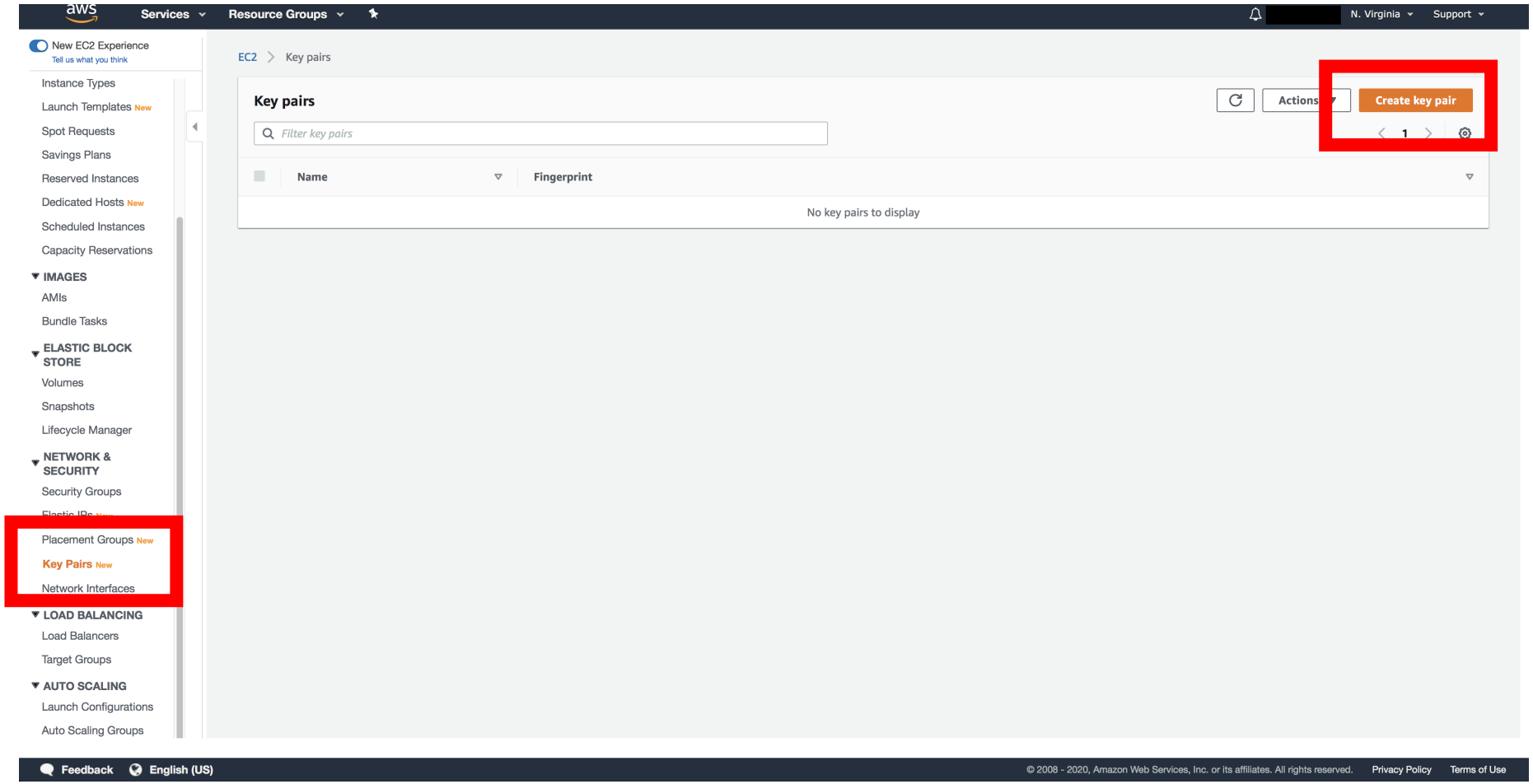
"Key setup": go back to your EC2 console



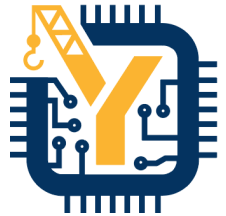
Key Setup

Select “Key Pairs” on the left menu.

Then select “Create key pair” on the top right

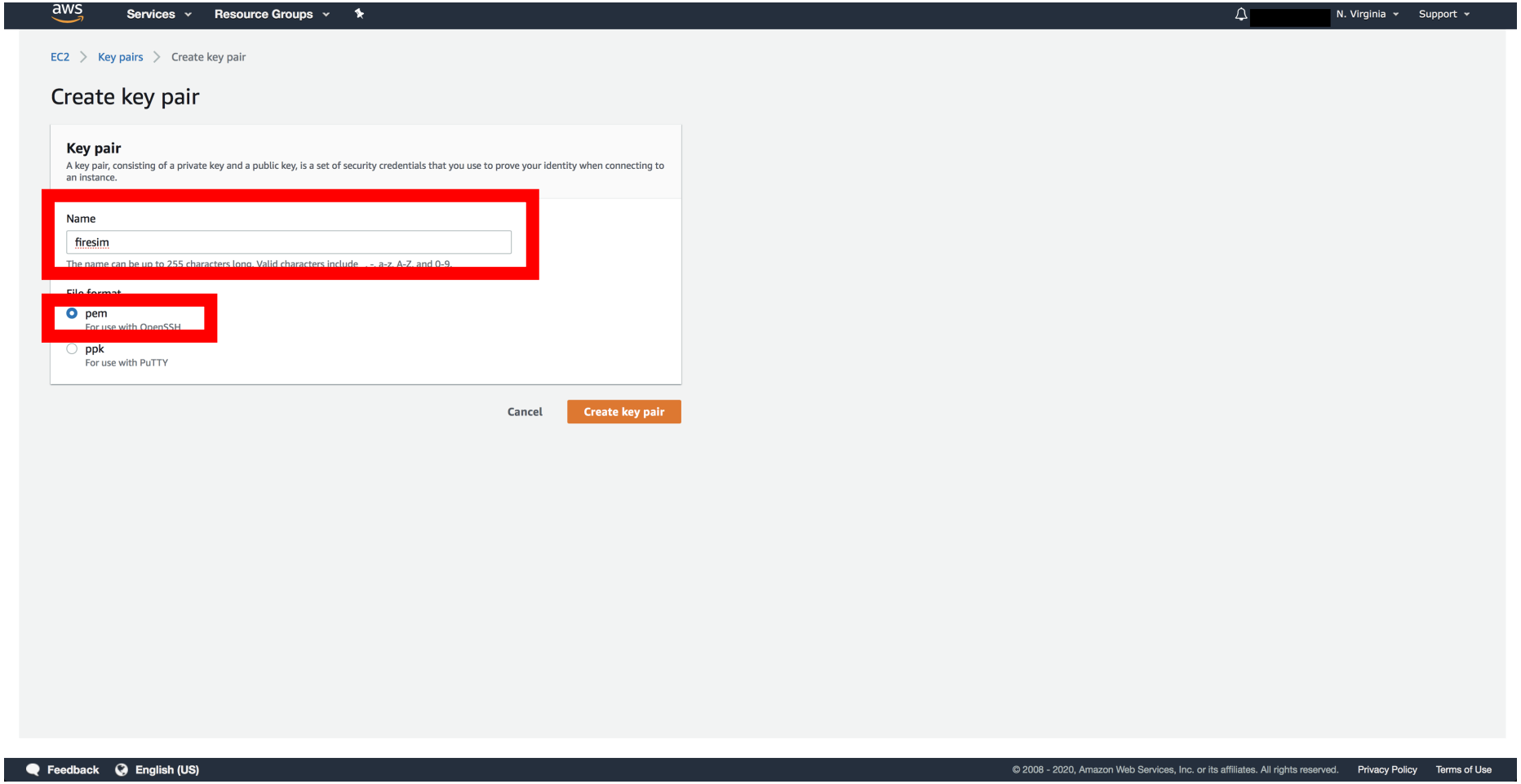


Key Setup

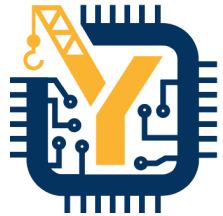


Name the key “firesim”, and make sure it’s in “pem” format.

Download the key, and **make sure you keep it in a safe place, and back it up.**



FireSim Docs – Configuring Infrastructure



"Check your EC2 Instance Limits": We've already requested a limit increase. Hopefully, it was approved by now.

Start a t2.nano instance

2.2.3. Check your EC2 Instance Limits

AWS limits access to particular instance types for new/infrequently used accounts to protect their infrastructure. You should make sure that your account has access to `f1.2xlarge`, `f1.4xlarge`, `f1.16xlarge`, `m4.16xlarge`, and `c5.4xlarge` instances by looking at the "Limits" page in the EC2 panel, which you can access here. The values listed on this page represent the maximum number of any of these instances that you can run at once, which will limit the size of simulations (# of nodes) that you can run. If you need to increase your limits, follow the instructions on the Requesting Limit Increases page. To follow this guide, you need to be able to run one `f1.2xlarge` instance and two `c5.4xlarge` instances.

2.2.4. Start a t2.nano instance to run the remaining configuration commands

To avoid having to deal with the messy process of installing packages on your local machine, we will spin up a very cheap `t2.nano` instance to run a series of one-time aws configuration commands to setup our AWS account for FireSim. At the end of these instructions, we'll terminate the `t2.nano` instance. If you happen to already have `boto3` and the AWS CLI installed on your local machine, you can do this locally.

Launch a `t2.nano` by following these instructions:

1. Go to the EC2 Management Console and click "Launch Instance"
2. On the AMI selection page, select "Amazon Linux AMI...", which should be the top option.
3. On the Choose an Instance Type page, select `t2.nano`.
4. Click "Review and Launch" (we don't need to change any other settings)
5. On the review page, click "Launch"
6. Select the `firesim` key pair we created previously, then click Launch Instances.
7. Click on the instance name and note its public IP address.

2.2.5. Run scripts from the t2.nano

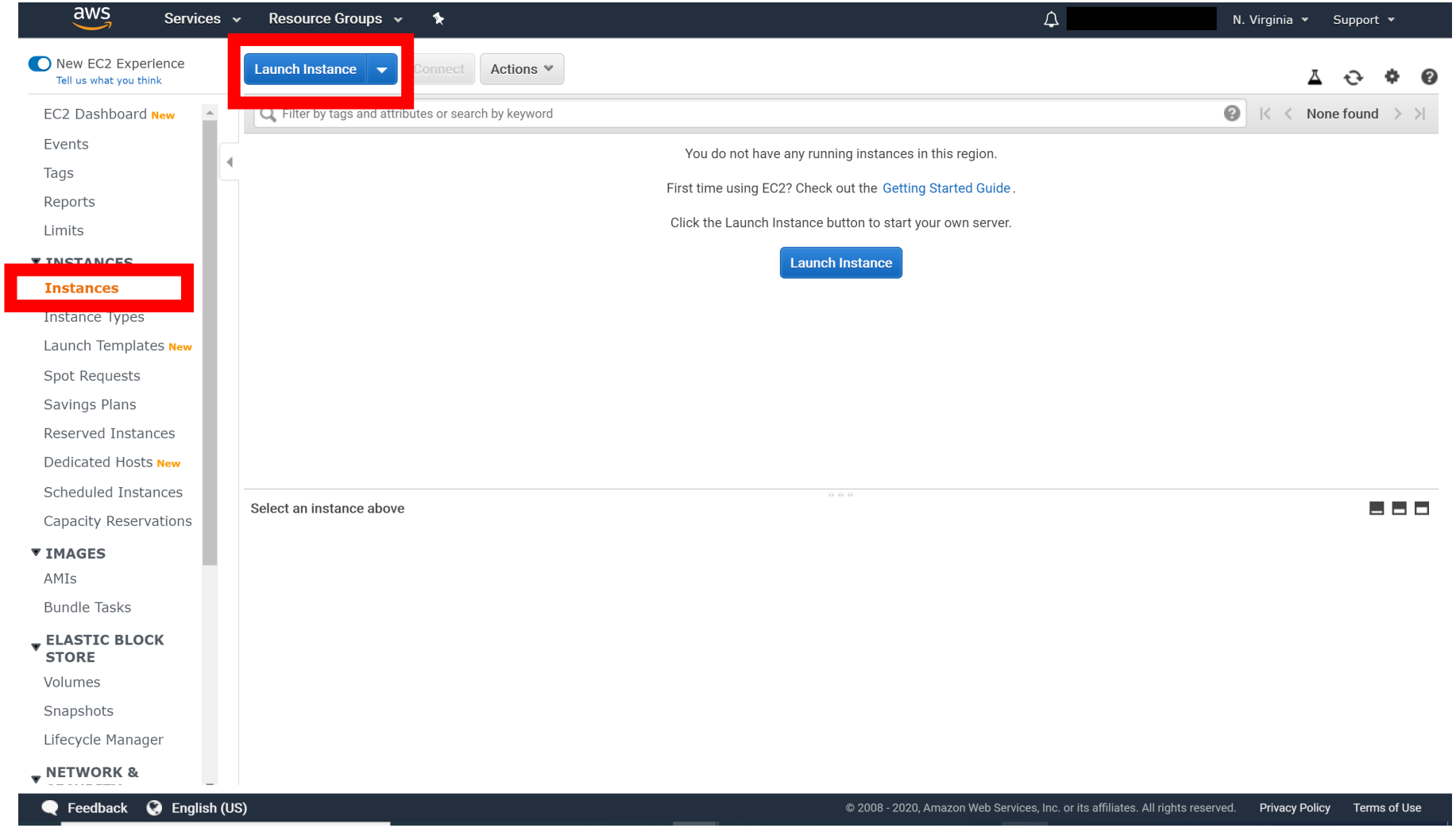
SSH into the `t2.nano` like so:

Berkeley Architecture Research

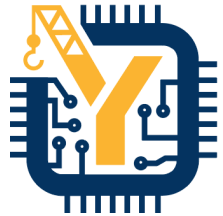
Configuring FireSim AWS Infrastructure

Select “Instances” on the left menu.

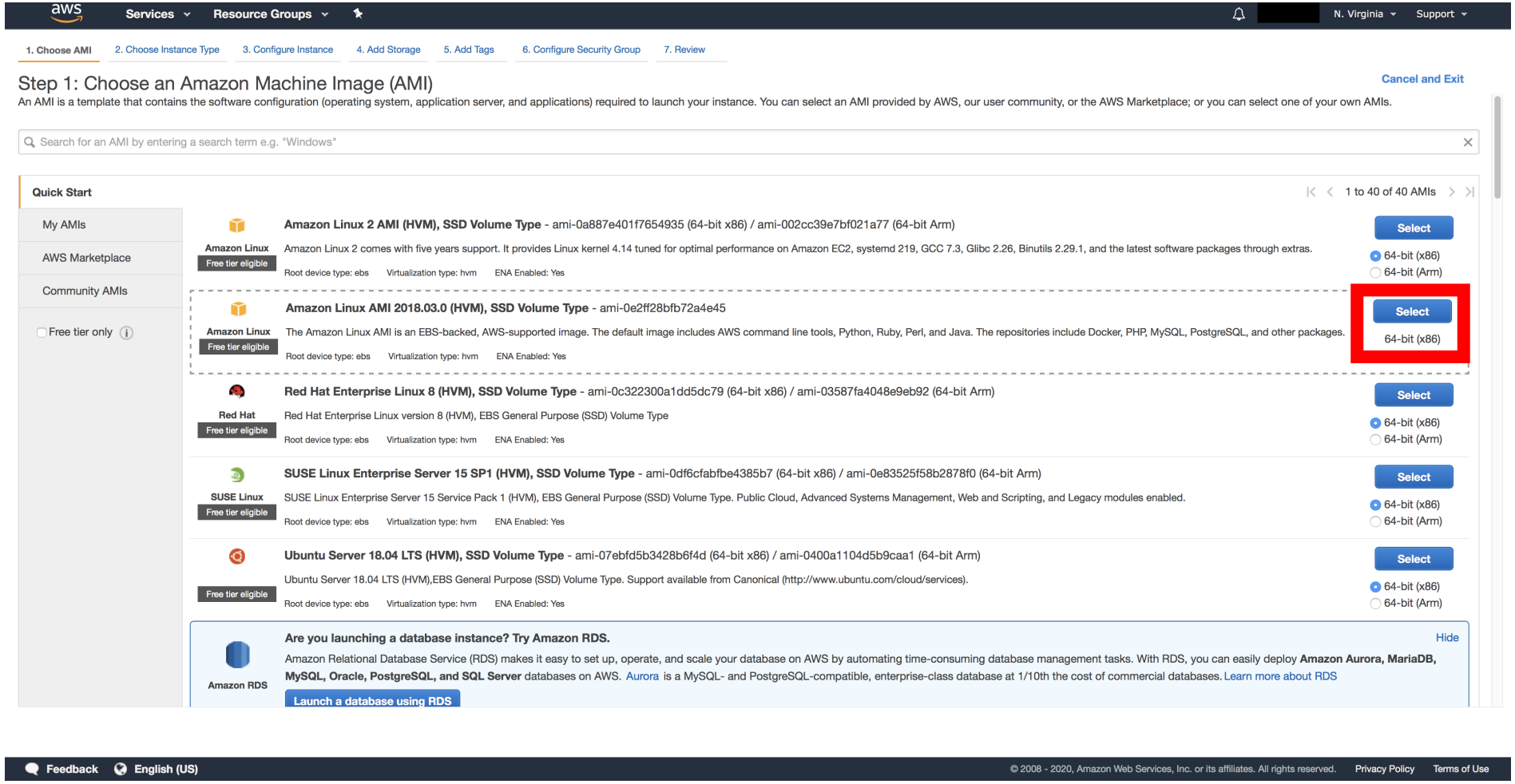
Then select “Launch Instance”



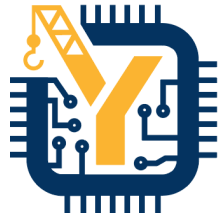
Configuring FireSim AWS Infrastructure



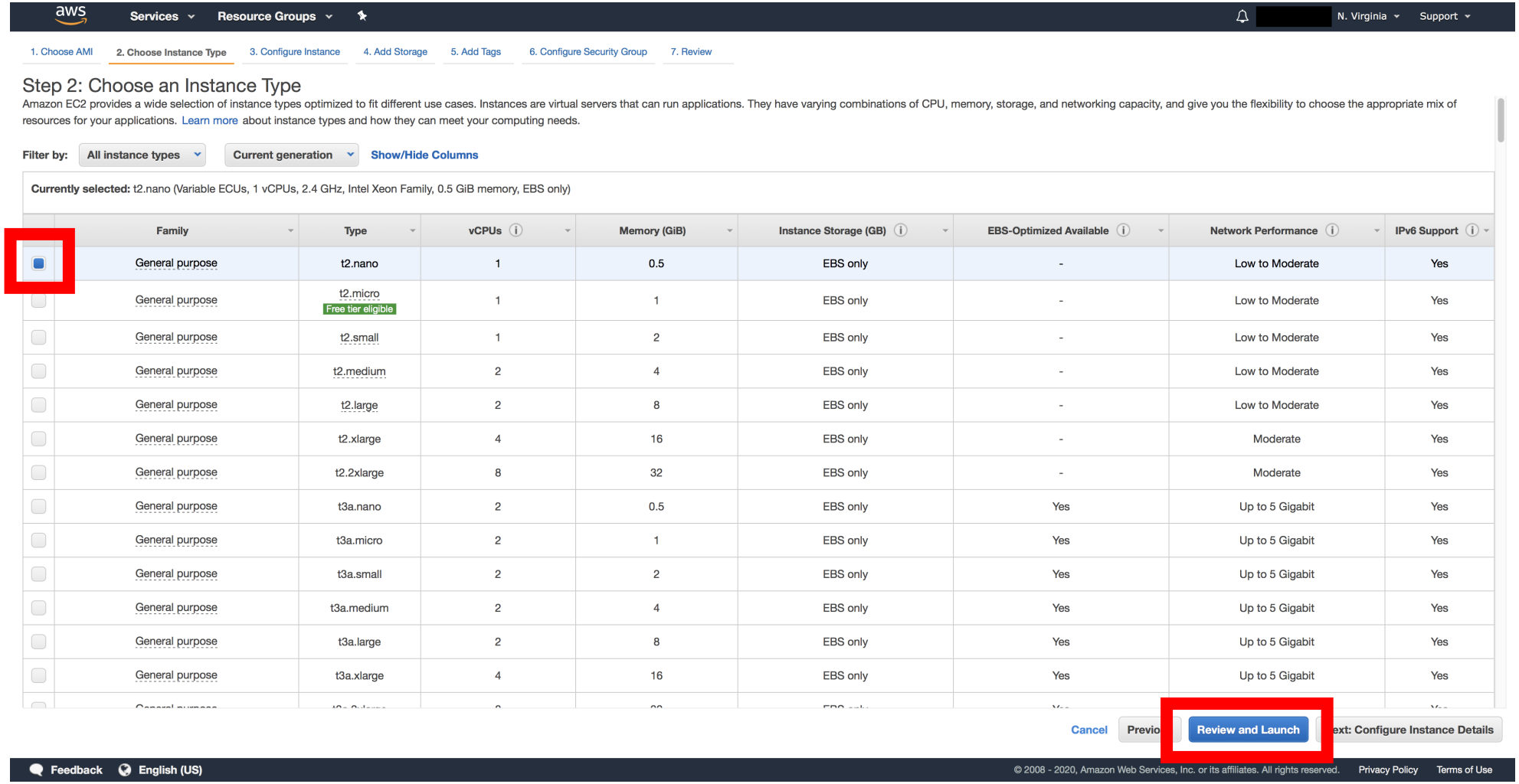
Select the “Amazon Linux 2 AMI”



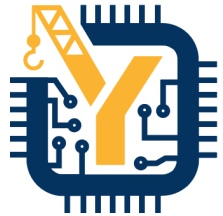
Configuring FireSim AWS Infrastructure



Select a t2.nano instance, and then click “Review and Launch”

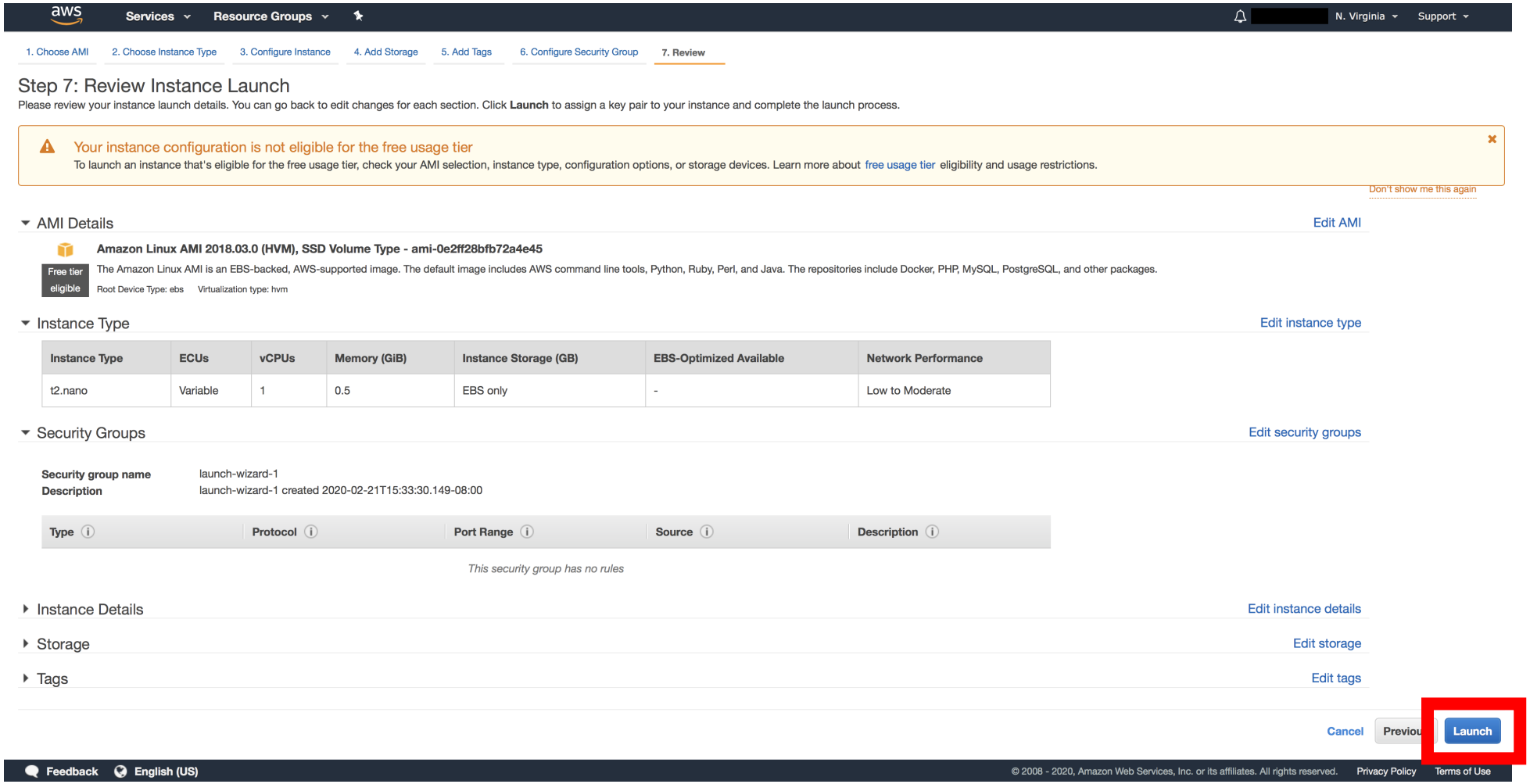


Configuring FireSim AWS Infrastructure



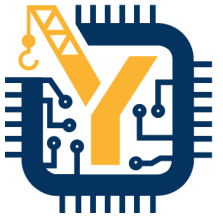
Review the details, and Launch.

It should then prompt you to select a key pair

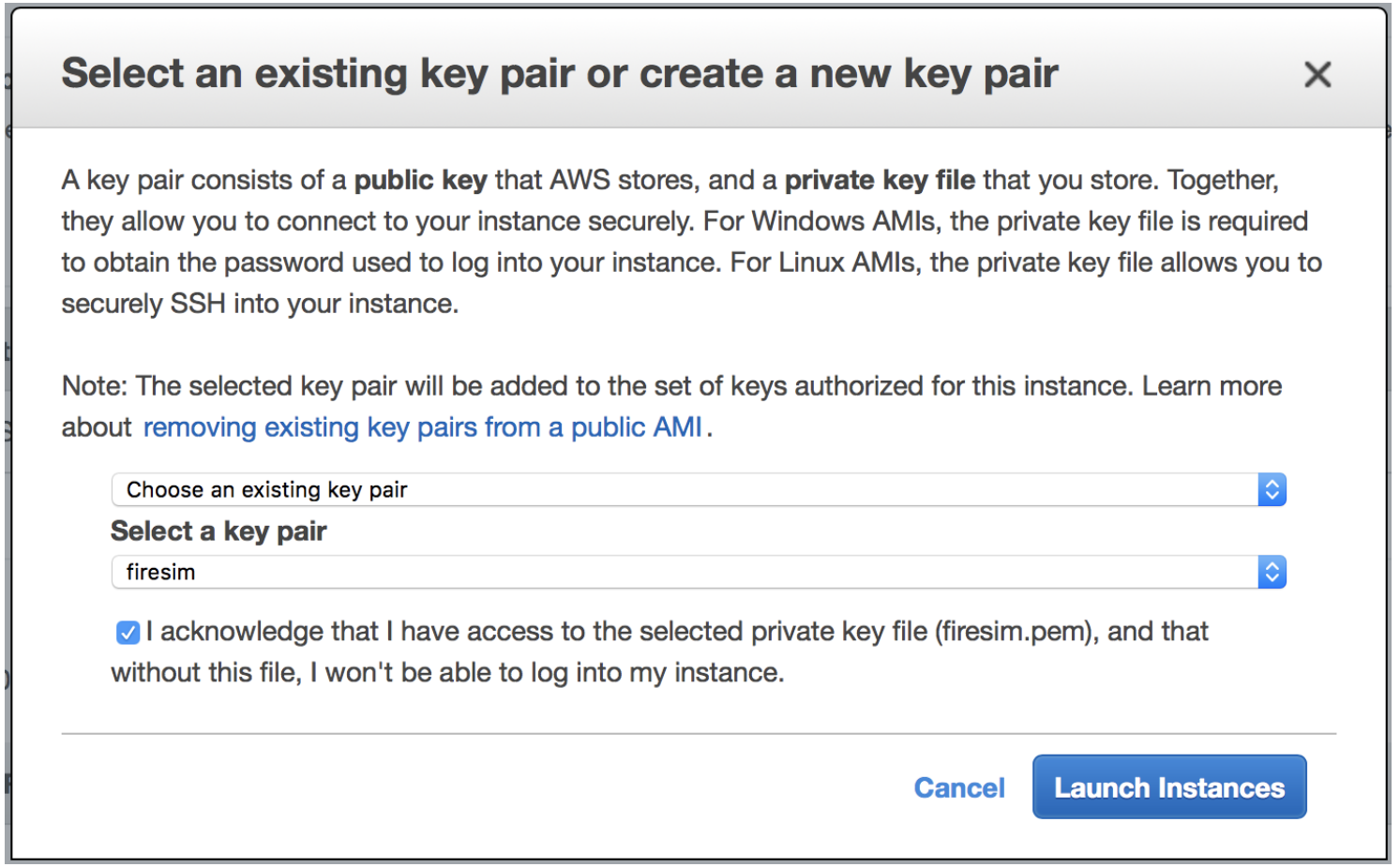


Berkeley Architecture Research

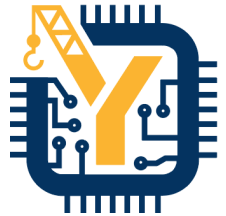
Configuring FireSim AWS Infrastructure



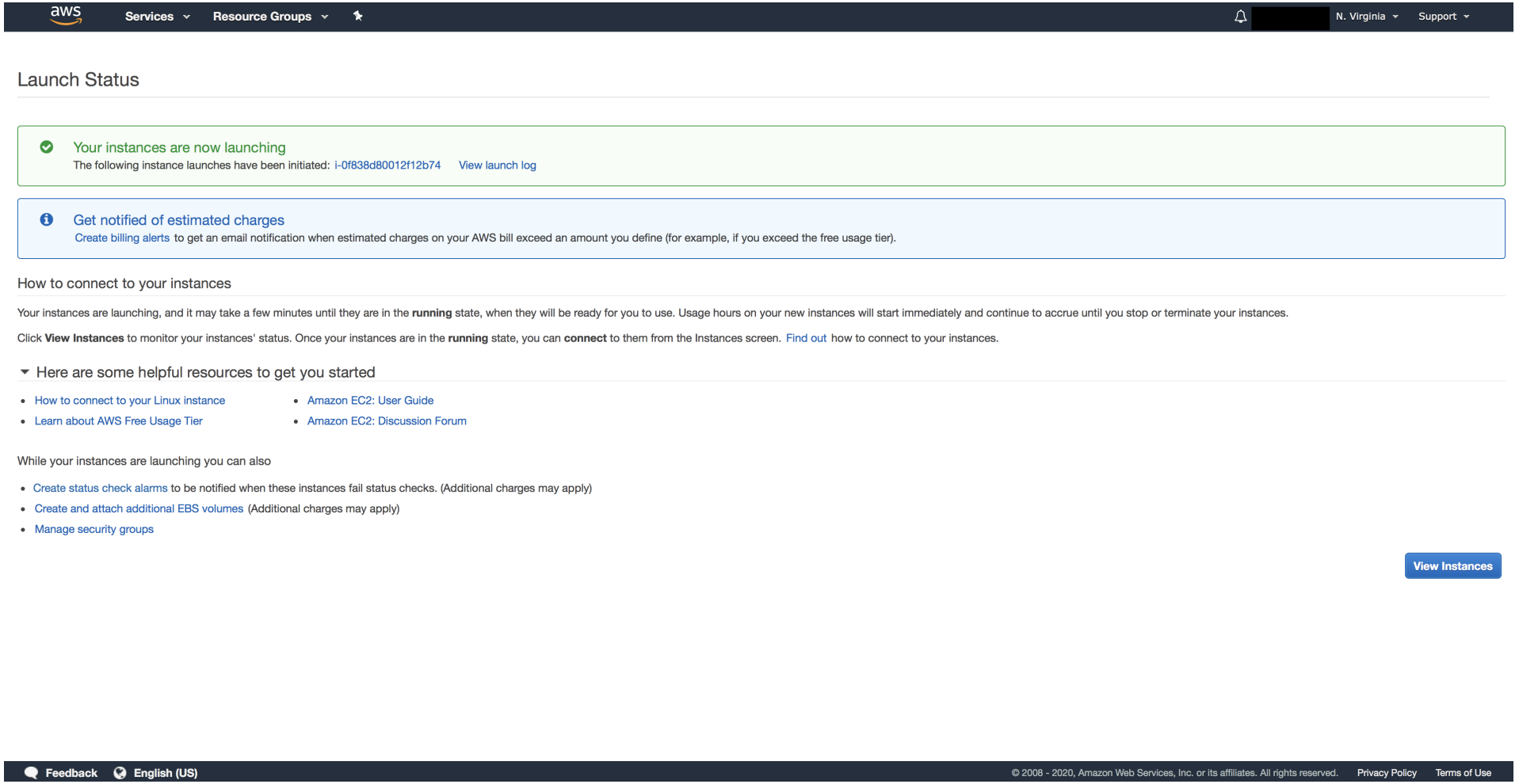
Choose “Choose and existing key pair”,
and then select the “firesim” key pair that you created previously



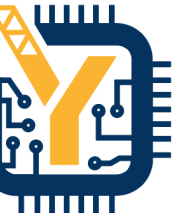
Configuring FireSim AWS Infrastructure



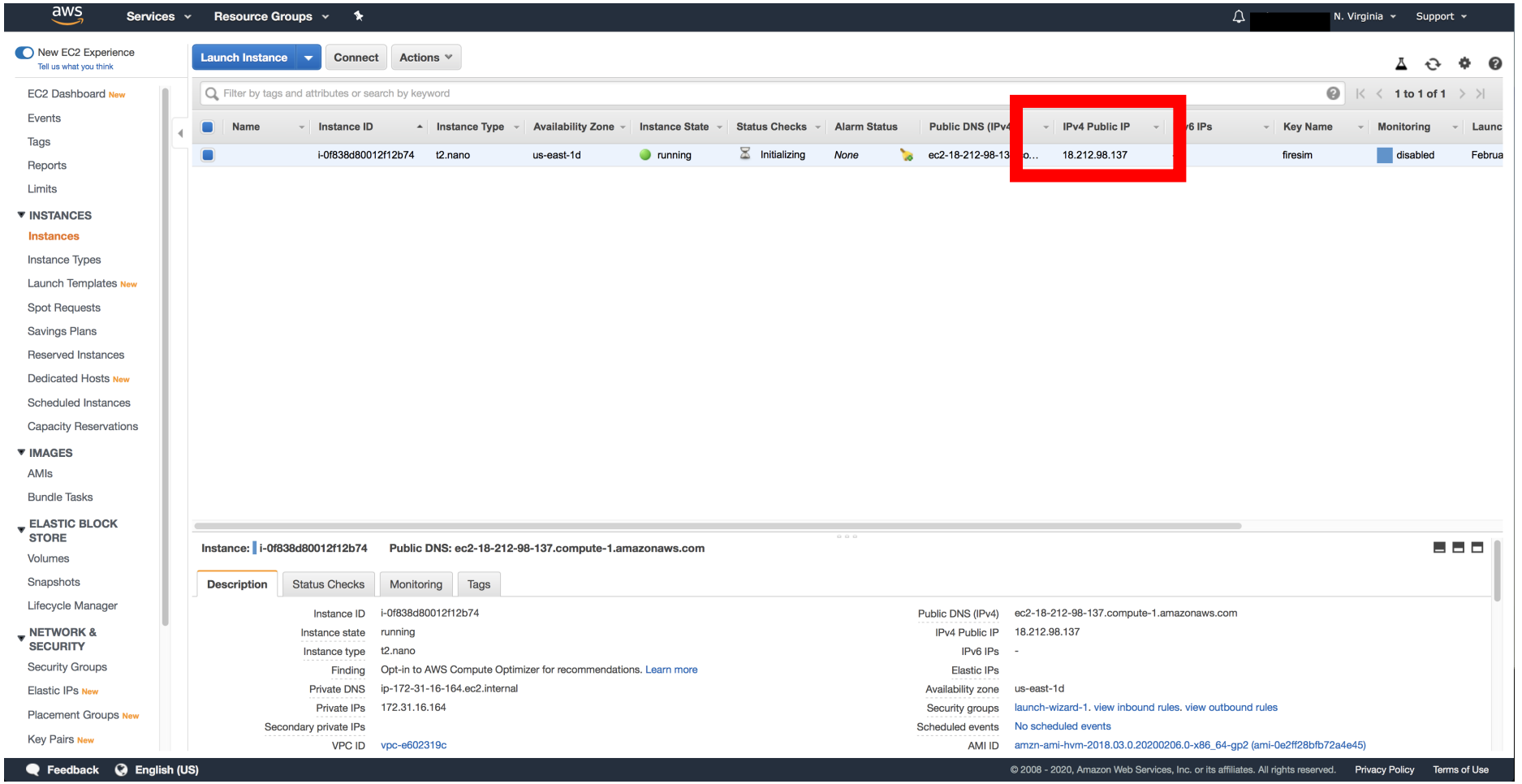
You will see a confirmation screen that your instance has launched



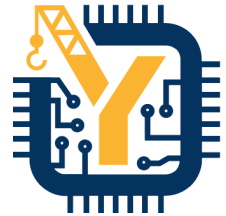
Configuring FireSim AWS Infrastructure



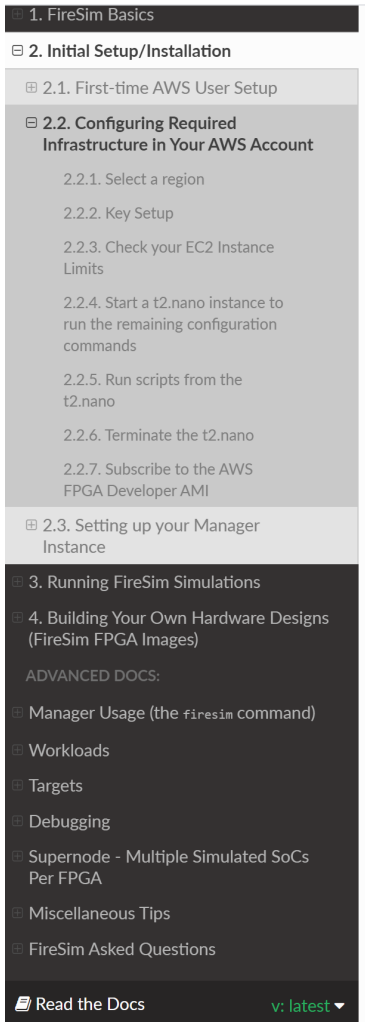
You will also now see the detail of your instance in the instances list, including the public IP address of your instance



FireSim Docs – Configuring Infrastructure



We now need to run the setup script on the t2.nano



2.2.5. Run scripts from the t2.nano

SSH into the `t2.nano` like so:

```
ssh -i firesim.pem ec2-user@INSTANCE_PUBLIC_IP
```

Which should present you with something like:

```
Last login: Mon Feb 12 21:11:27 2018 from 136.152.143.34

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
4 package(s) needed for security, out of 5 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-30-2-66 ~]$
```

On this machine, run the following:

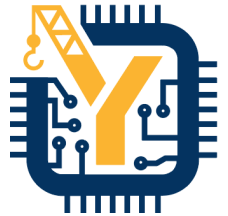
```
aws configure
[follow prompts]
```

See https://docs.aws.amazon.com/cli/latest/userguide/tutorial-ec2-ubuntu.html#configure-cli-launch-ec2 for more about aws configure. Within the prompt, you should specify the same region that you chose above (one of `us-east-1`, `us-west-2`, `eu-west-1`) and set the default output format to `json`. You will need to generate an AWS access key in the "Security Credentials" menu of your AWS settings (as instructed in https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys).

Again on the `t2.nano` instance, do the following:

```
sudo yum -y install python-pip
sudo pip install boto3
wget https://raw.githubusercontent.com/firesim/firesim/master/scripts/aws-setup.py
python aws-setup.py
```

Configuring FireSim AWS Infrastructure



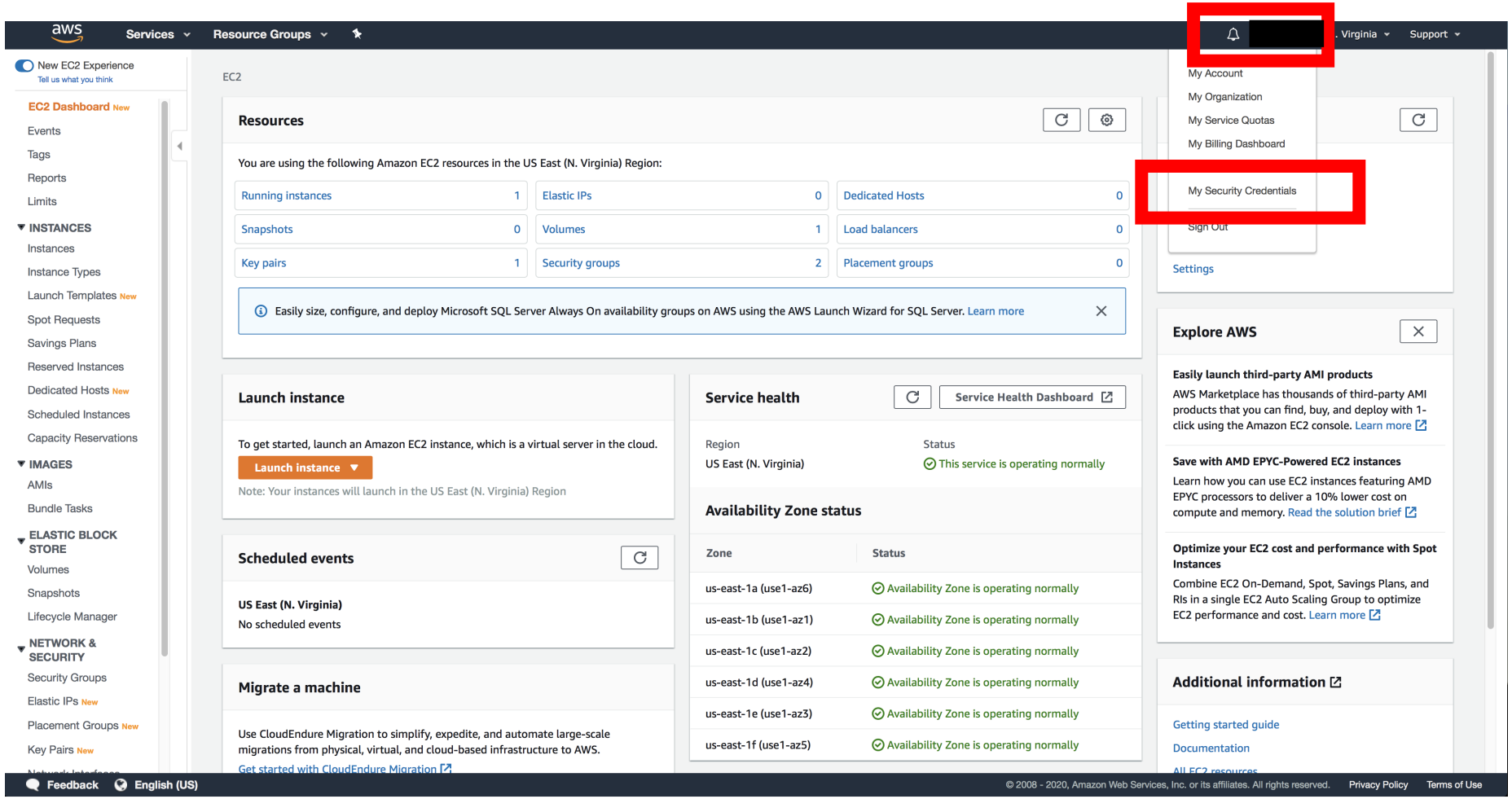
```
$ chmod 400 firesim.pem
$ ssh -i firesim.pem ec2-user@18.212.98.137

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|
https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
4 package(s) needed for security, out of 10 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-16-164 ~]$ aws configure
```

Before running this command, we need to setup AWS Access Keys

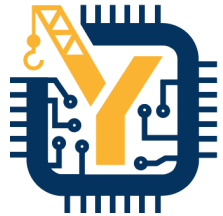
Berkeley Architecture Research

Configuring FireSim AWS Infrastructure

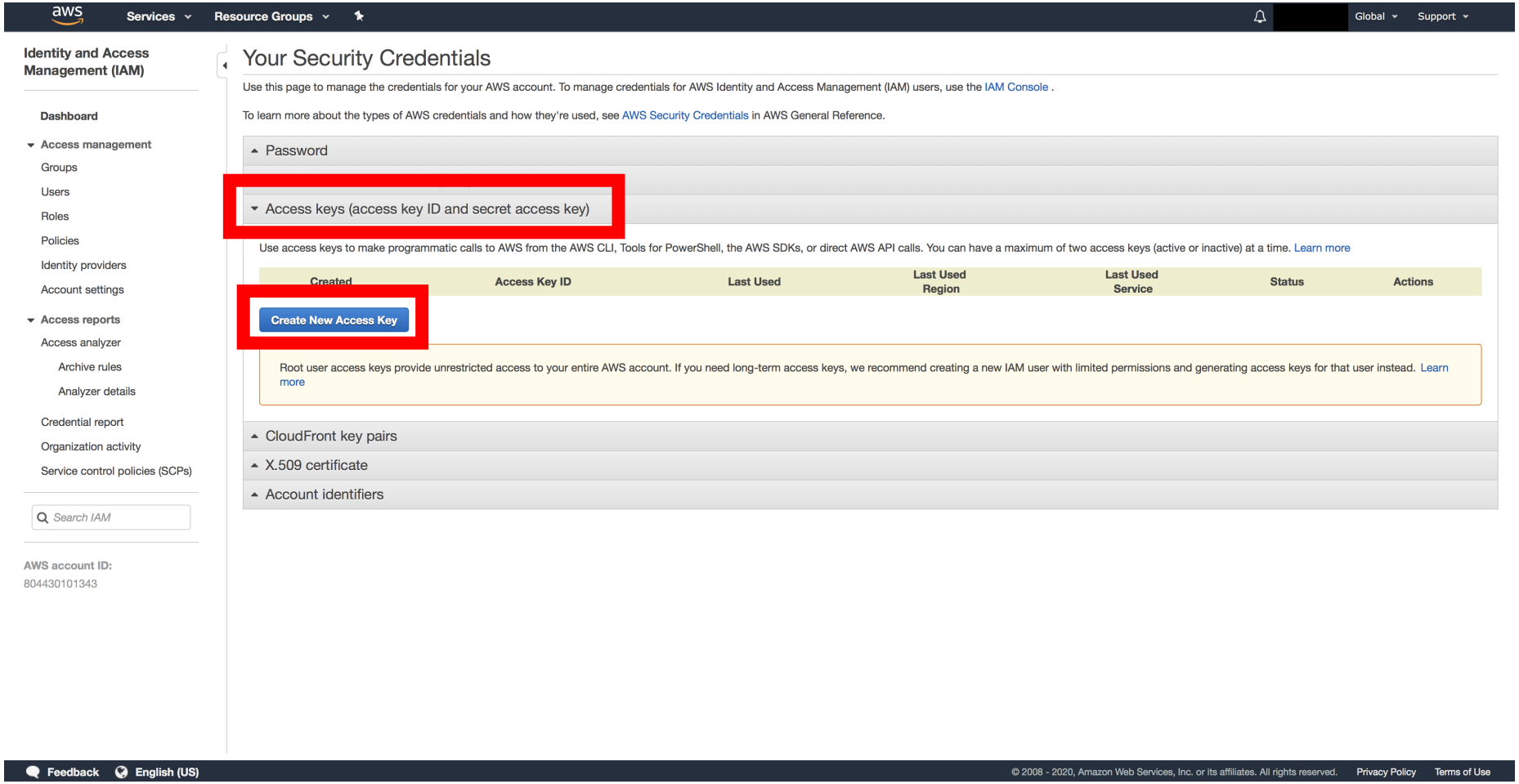
In you AWS EC2 console, select “My Security Credentials” from your user menu on the top right corner



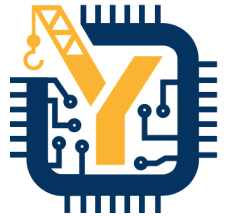
Configuring FireSim AWS Infrastructure



In the “Access keys” section, create a new access key

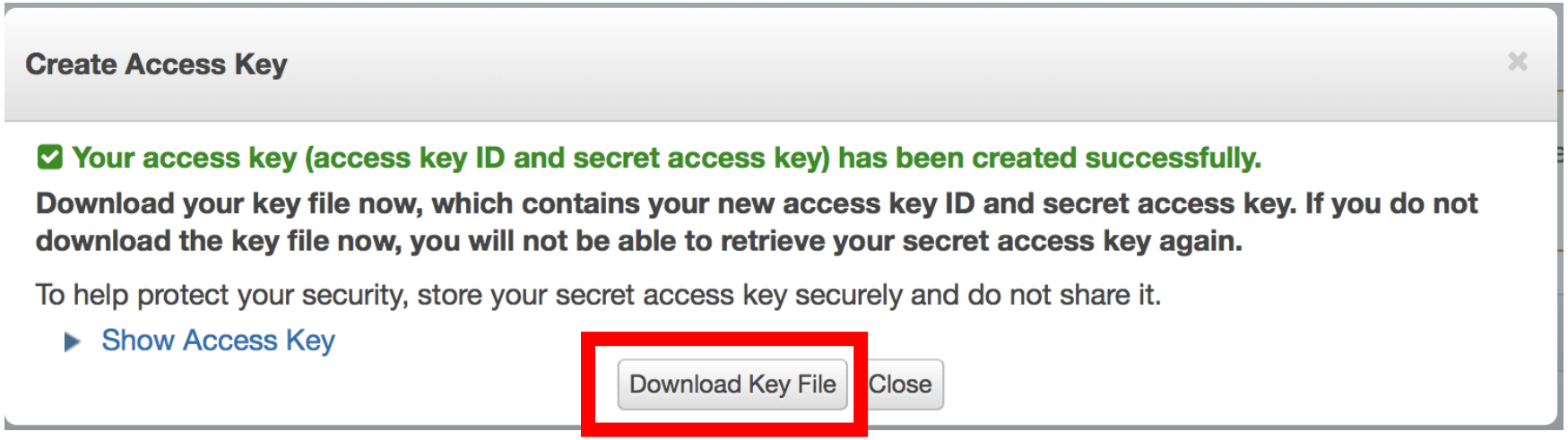


Configuring FireSim AWS Infrastructure

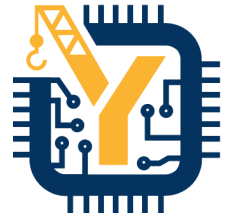


Download your root key file. This is a csv file that includes the Access Key ID, and the Secret Access Key (you will need both of them for the `aws configure` command)

SAVE THIS IN A SPECIAL LOCATION



Configuring FireSim AWS Infrastructure



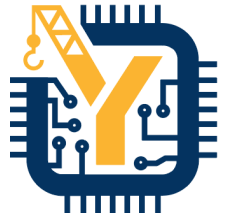
Continue with the `aws configure` command, using your AWS Access Key.

For default region name, enter: **us-east-1**

For default output format, enter: **json**

```
[ec2-user@ip-172-31-16-164 ~]$ aws configure
AWS Access Key ID [None]: XXXXXXXX
AWS Secret Access Key [None]: XXXXXXXXXX
Default region name [None]: us-east-1
Default output format [None]: json
```

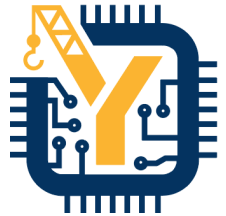
Configuring FireSim AWS Infrastructure



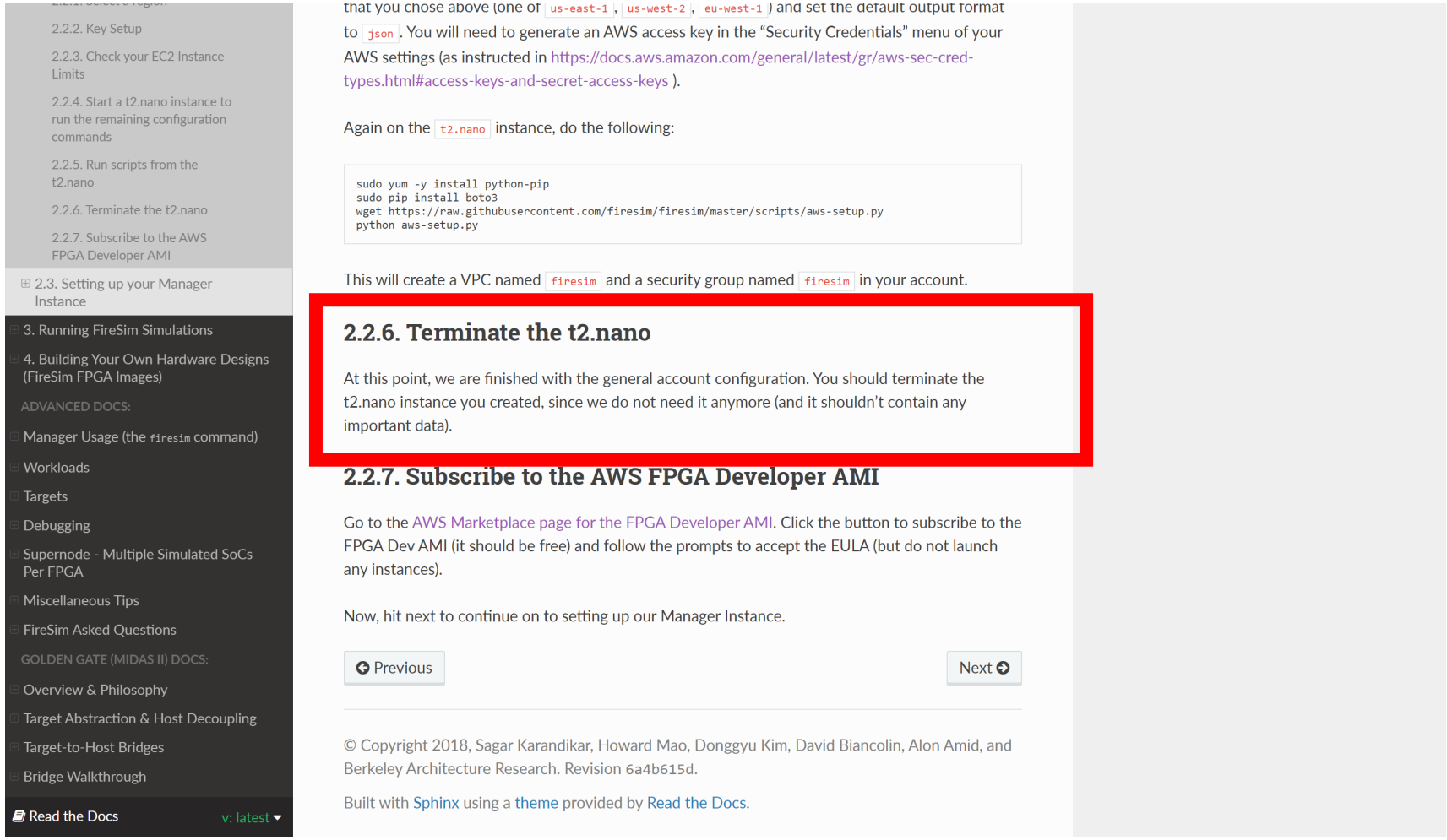
Run the setup scripts as instructed in the FireSim docs setup instructions:

```
sudo yum -y install python-pip
sudo pip install boto3
wget https://raw.githubusercontent.com/firesim/firesim/master/scripts/aws-setup.py
python aws-setup.py
…

Creating VPC for FireSim...
Success!
Creating a subnet in the VPC for each availability zone...
Success!
Creating a security group for FireSim...
Success!
```

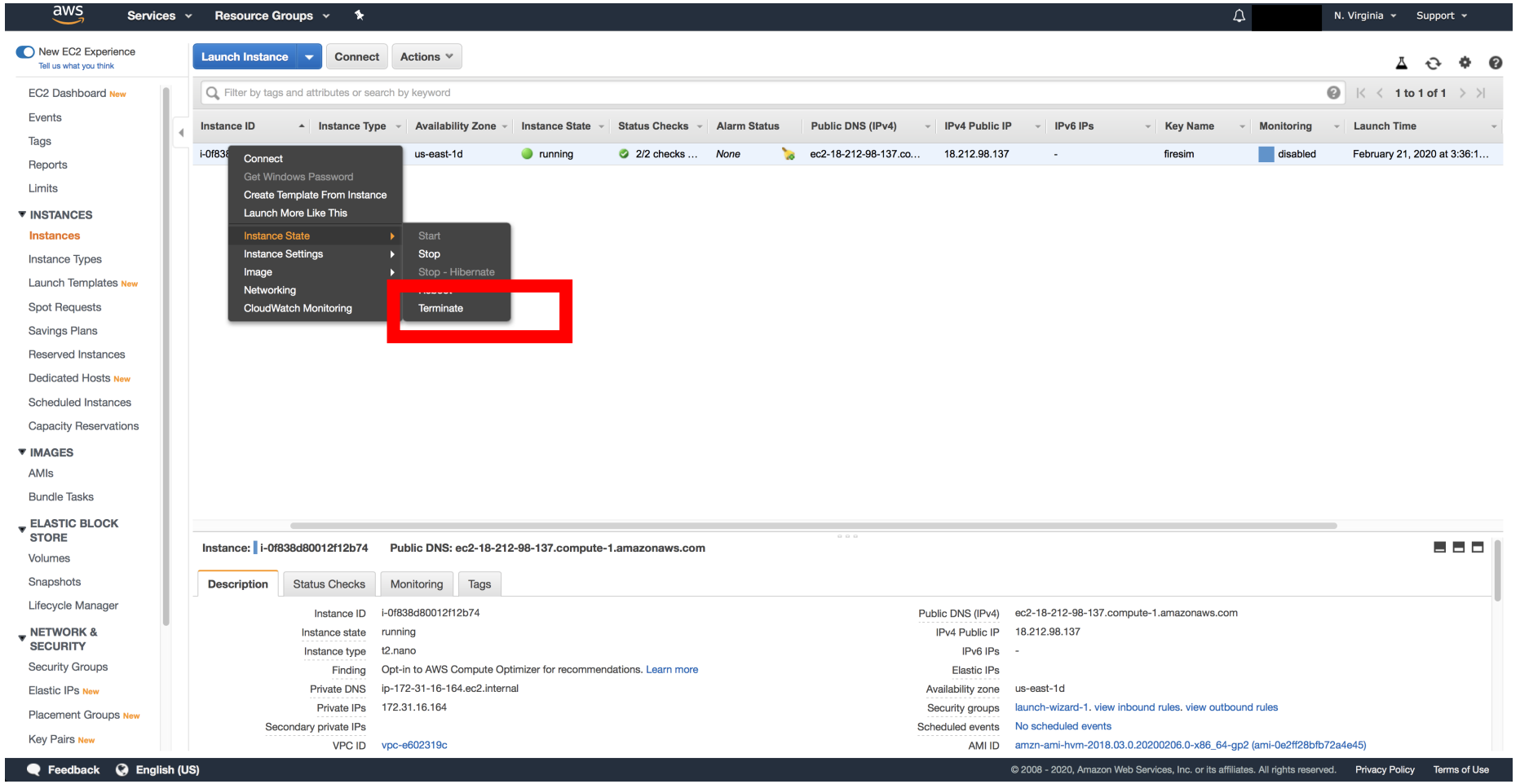
FireSim Docs – Configuring Infrastructure



To terminate the t2.nano instance, go back to your EC2 console, to the instances list

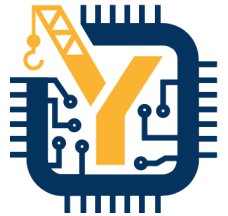


Configuring FireSim AWS Infrastructure

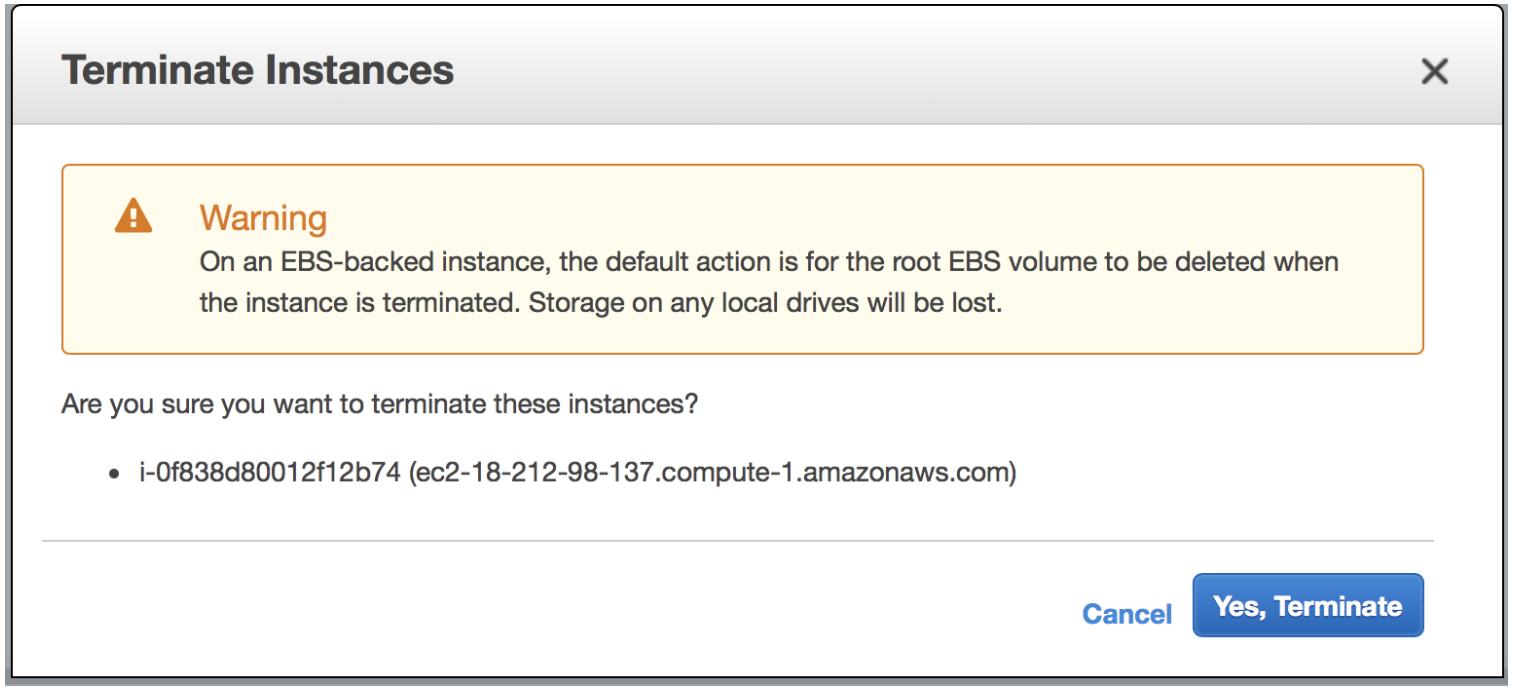


Right click on the instance entry in the table, and within the “Instance State” menu, select “Terminate”.

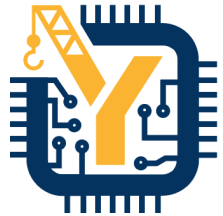
Configuring FireSim AWS Infrastructure



Confirm the instance termination



FireSim Docs – Configuring Infrastructure



Subscribe to the AWS FPGA Developer AMI (click on the link)

that you chose above (one of `us-east-1`, `us-west-2`, `eu-west-1`) and set the default output format to `json`. You will need to generate an AWS access key in the "Security Credentials" menu of your AWS settings (as instructed in https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys).

Again on the `t2.nano` instance, do the following:

```
sudo yum -y install python-pip
sudo pip install boto3
wget https://raw.githubusercontent.com/firesim/firesim/master/scripts/aws-setup.py
python aws-setup.py
```

This will create a VPC named `firesim` and a security group named `firesim` in your account.

2.2.6. Terminate the t2.nano

At this point, we are finished with the general account configuration. You should terminate the t2.nano instance you created, since we do not need it anymore (and it shouldn't contain any important data).

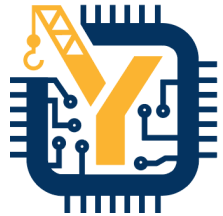
2.2.7. Subscribe to the AWS FPGA Developer AMI

Go to the AWS Marketplace page for the FPGA Developer AMI. Click the button to subscribe to the FPGA Developer AMI (it should be free) and follow the prompts to accept the EULA (but do not launch any instances).

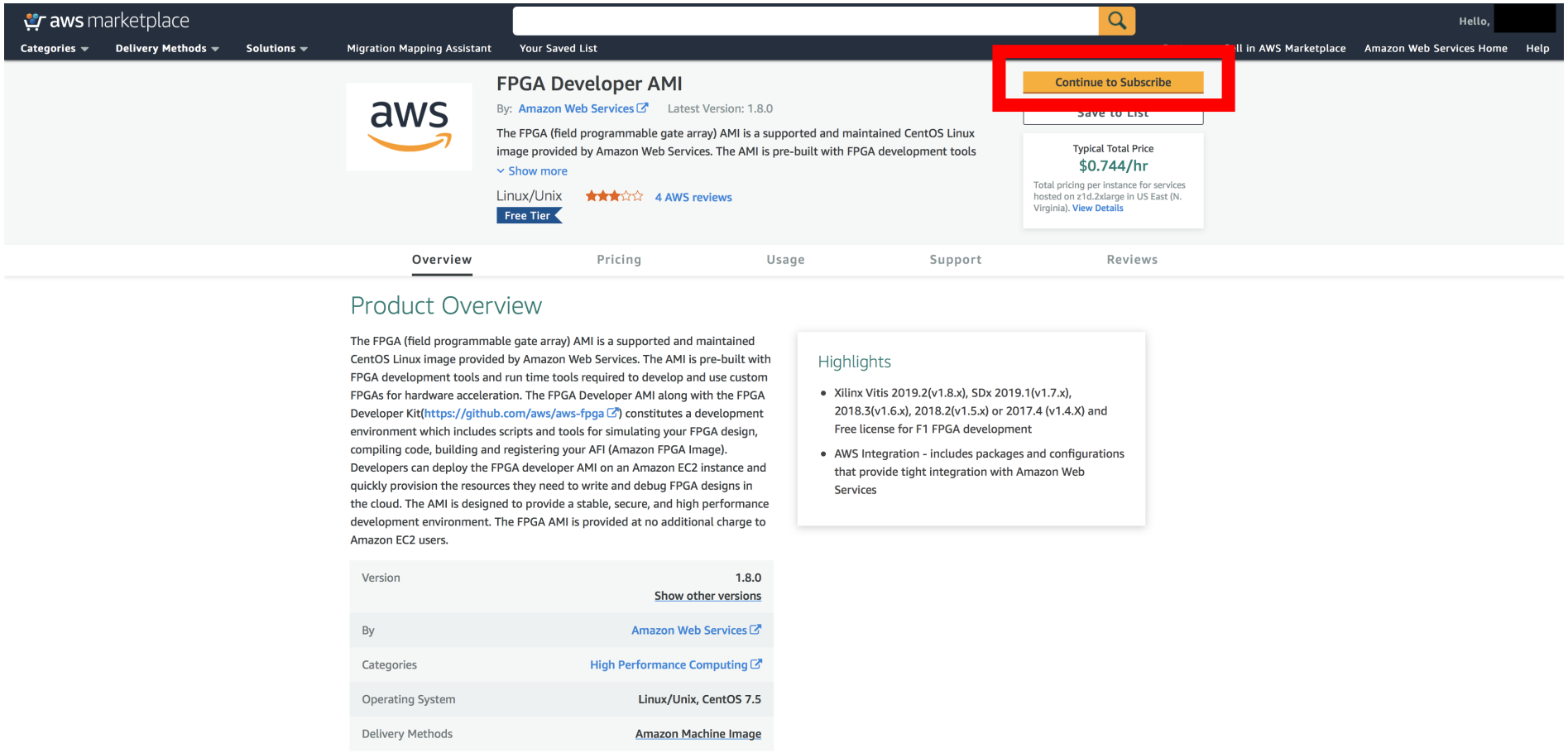
Now, hit next to continue on to setting up our Manager Instance.

Previous Next

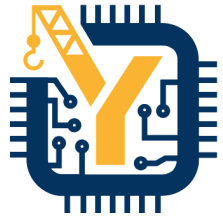
Configuring FireSim AWS Infrastructure



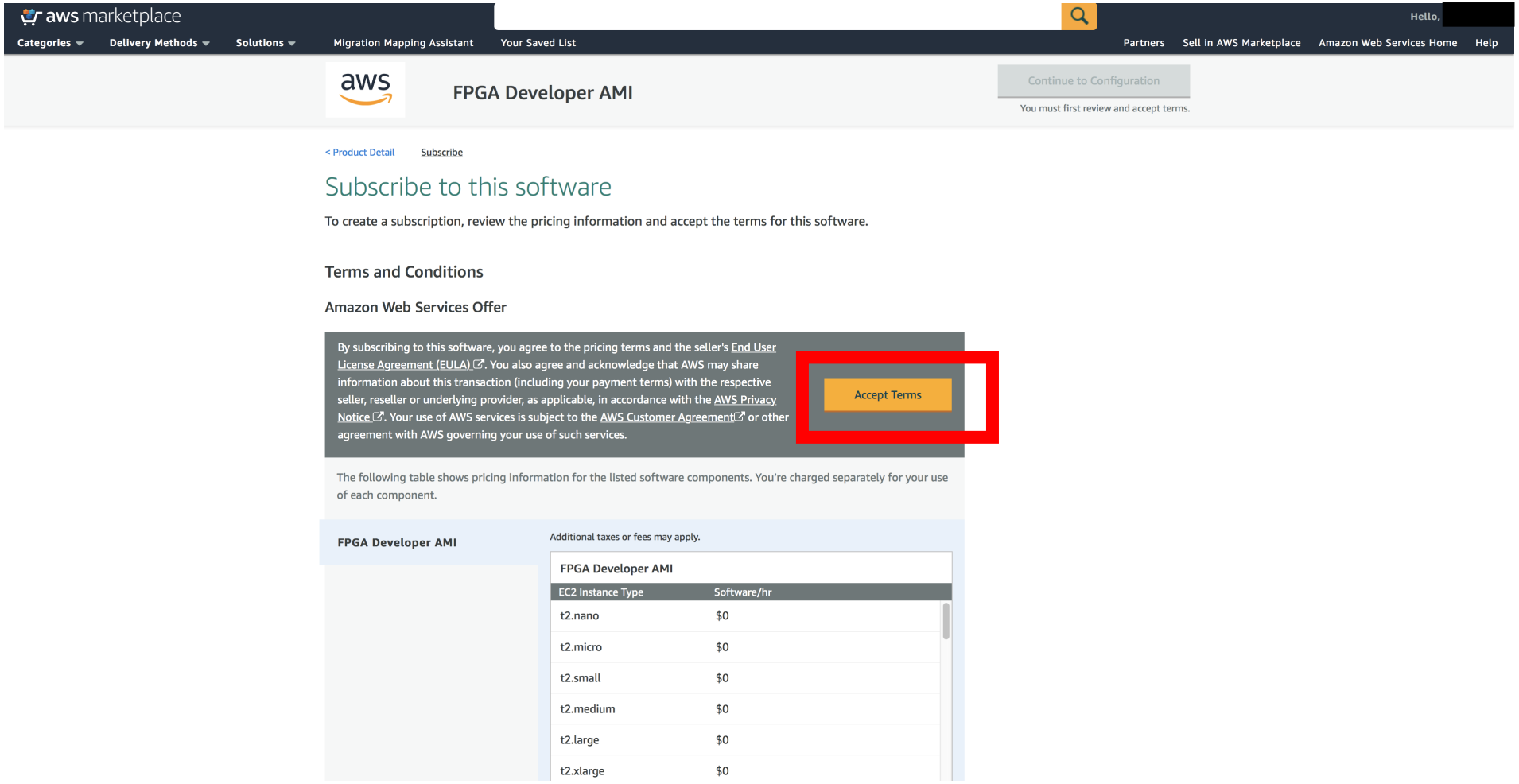
Subscribe to the FPGA Developer AMI



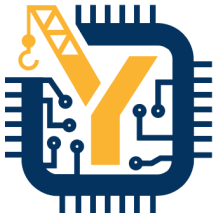
Configuring FireSim AWS Infrastructure



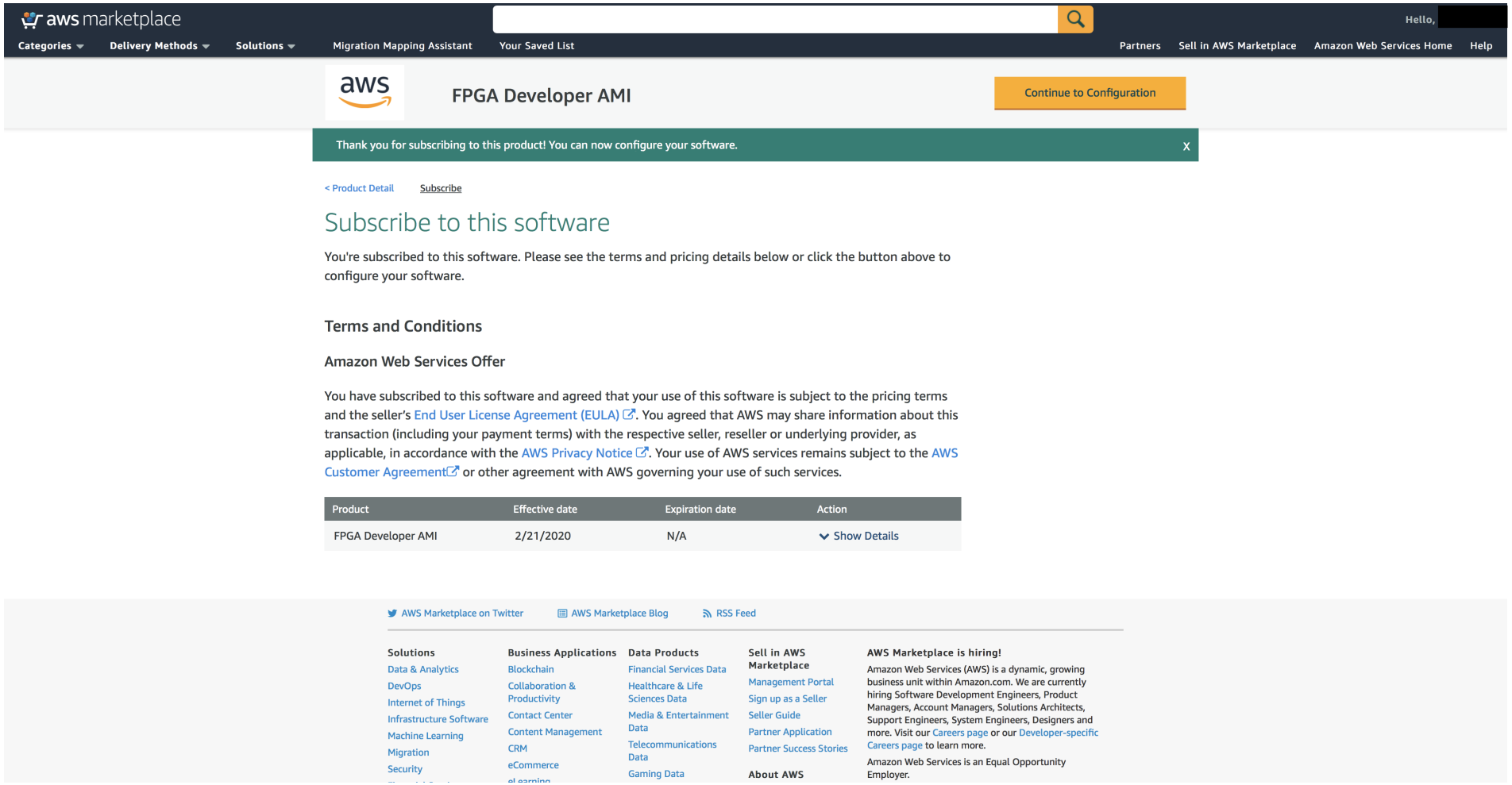
Accept the Terms



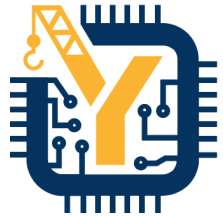
Configuring FireSim AWS Infrastructure



Receive confirmation for subscribing to the FPGA Developer AMI



Lab 3 Instructions - Manager Instance



- Do this when you are starting the second part of the lab
- Note that these instructions diverge from the main FireSim documentation, since we have prepared an image for you with pre-built tools
  - This is in order to save time. This setup process is long enough :)

model implemented in the non-standard version of the Spike ISA simluator. The Spike ISA simulator was originally used as a "golden model" for the RISC-V ISA. As a functional simulator, in Spike every instruction takes only 1 cycle to execute (no matter how complicated the instruction is, or whether it should have memory latency).

A binary of the Spike simulator is located in the software development tools that you get when you source `/home/ff/ee290-2/chipyard-env.sh` in Chipyard. Spike should be on your path after you source this file, so in order to run a software binary in Spike, you should just need to run the following command:

```
spike --extension=gemmini <path/to/software/binary>
```

Spike has various visibility features that may help you while debugging your software implementations. You can read about most of these features in https://github.com/ucb-bar/esp-isa-sim/#interactive-debug-mode

You are also provided with a special version of Spike, which includes additional details about the functional simulation of the Gemmini functional model. In order to enable this version of spike, you will need to source the `/home/ff/ee290-2/chipyard-debug-env.sh` file instead of `/home/ff/ee290-2/chipyard-env`. This version generates many print statements, so we recommend using it only when neccessary. This version is identical to the regular Spike version (command line options, flags, etc.), with the addition of these print statements.

```
source /home/ff/ee290-2/chipyard-debug-env.sh
spike --extension=gemmini <path/to/software/binary>
```

3.2 FireSim FPGA-Accelerated Simulation

FireSim is an FPGA-accelerated cycle-exact simulation platform which uses FPGAs on the AWS EC2 public cloud. In contrast to software RTL simulation, FPGA-accelerated simulation enables us to run long workloads (billions-trillions of cycles) within reasonable wall-clock time. Running these workloads in software RTL simulation would take many hours/days/weeks. In contrast to standard FPGA prototyping, FireSim's simulation maintains cycle accurate timing behavior of the entire system (including memory and peripherals). For example, the simulations you are going to run in this lab are going to use a timing-accurate DDR3 memory model. Additional information about FireSim can be found on the FireSim website (https://fires.im/) and in the FireSim documentation at https://docs.fires.im/en/latest/.

FireSim is included as part of the Chipyard framework, which we used in Lab 2. FireSim is located in the `sims/firesim` directory of Chipyard. You will use FireSim in this lab in order to evaluate the performance of real DNN models on the Gemmini accelerator RTL using simulations which take billions of cycles to run.

The remainder of this section will guide you through setting up a FireSim manager instance for this assignment, which you will use for the second half of the assignment. We recommend going through this part of the setup of the manager instance only after you completed the first part of the assignment (on the local eda machines), since this will help conserve AWS resources while your instance is not in active use.

FireSim uses a central *manager* instance in order to manage its operations on AWS. In order to set up your FireSim manager instance, head to the EC2 Management Console. In the top right corner, ensure that the correct region is selected.

To launch a manager instance, follow these steps:

1. From the main page of the EC2 Management Console, click `Launch Instance`. We use an on-demand instance here, so that your data is preserved when you stop/start the instance, and your data is not lost when pricing spikes on the spot market.

2. When prompted to select an AMI, search for the following AMI name: `Berkeley EE290-2 FireSim Manager AMI - Spring 2020`. If you have completed the form in the the introduction section, and recieved an AWS promocode from us, it should appear in the "My AMIs" section (we are sharing this AMI manually with you, since there are limitations on public AMIs in AWS. If more than a day has passed since you completed the form in the introduction section and the AMI doesn't appear in your , please email us). ****DO NOT USE ANY OTHER VERSION.****[7].
3. When prompted to choose an instance type, select the instance type of your choosing. A good choice is a `r5.large`.
4. On the "Configure Instance Details" page, first make sure that the `firesim` VPC is selected in the drop-down box next to "Network". Any subnet within the `firesim` VPC is fine. Additionally, check the box for "Protect against accidental termination." This adds a layer of protection to prevent your manager instance from being terminated by accident. You will need to disable this setting before being able to terminate the instance using usual methods.
5. You can skip the "Add Tags" page.
6. On the "Configure Security Group" page, select the "firesim" security group that was automatically created for you earlier.
7. On the review page, click the button to launch your instance. Make sure you select the `firesim` key pair that we setup earlier.

Note: once the instances was launched, your AWS account is charged for its use. You should "stop" your manager instance when you are not using it for more than a few hours—especially at night. This will make sure your allocated AWS credits will suffice for this lab.

FireSim recommends using mosh instead of `ssh`, or using `ssh` with a screen/tmux session running on your manager instance to ensure that long-running jobs are not killed by a bad network connection to your manager instance. In either case, `ssh` (or `mosh`) into your instance (e.g. `ssh -i firesim.pem centos@YOUR_INSTANCE_IP`). Now that the manager instance is started, copy the private key that you downloaded from AWS earlier when you set up the intrastructure (`firesim.pem`) to `~/firesim.pem` on your manager instance. This step is required to give the manager access to the instances it launches for you.

Go into the firesim directory in `chipyard/sims/firesim` on the manager instance, and source the `sourceme-f1-manager.sh` file.

```
$ source sourceme-f1-manager.sh
```

Finally, run the `firesim managerinit` command.

```
$ firesim managerinit
```

This will first prompt you to setup AWS credentials on the instance, which allows the manager to automatically manage build/simulation nodes (these are the same credentials you entered in the infrastructure setup section with the `aws configure` command (choose the `us-east-1` region, and `json` default output format).

Next, it will create initial configuration files we will use. Finally, it will prompt you for an email address, which is used to send email notifications upon FPGA build completion and optionally for workload completion. You can leave this blank if you do not wish to receive any notifications.

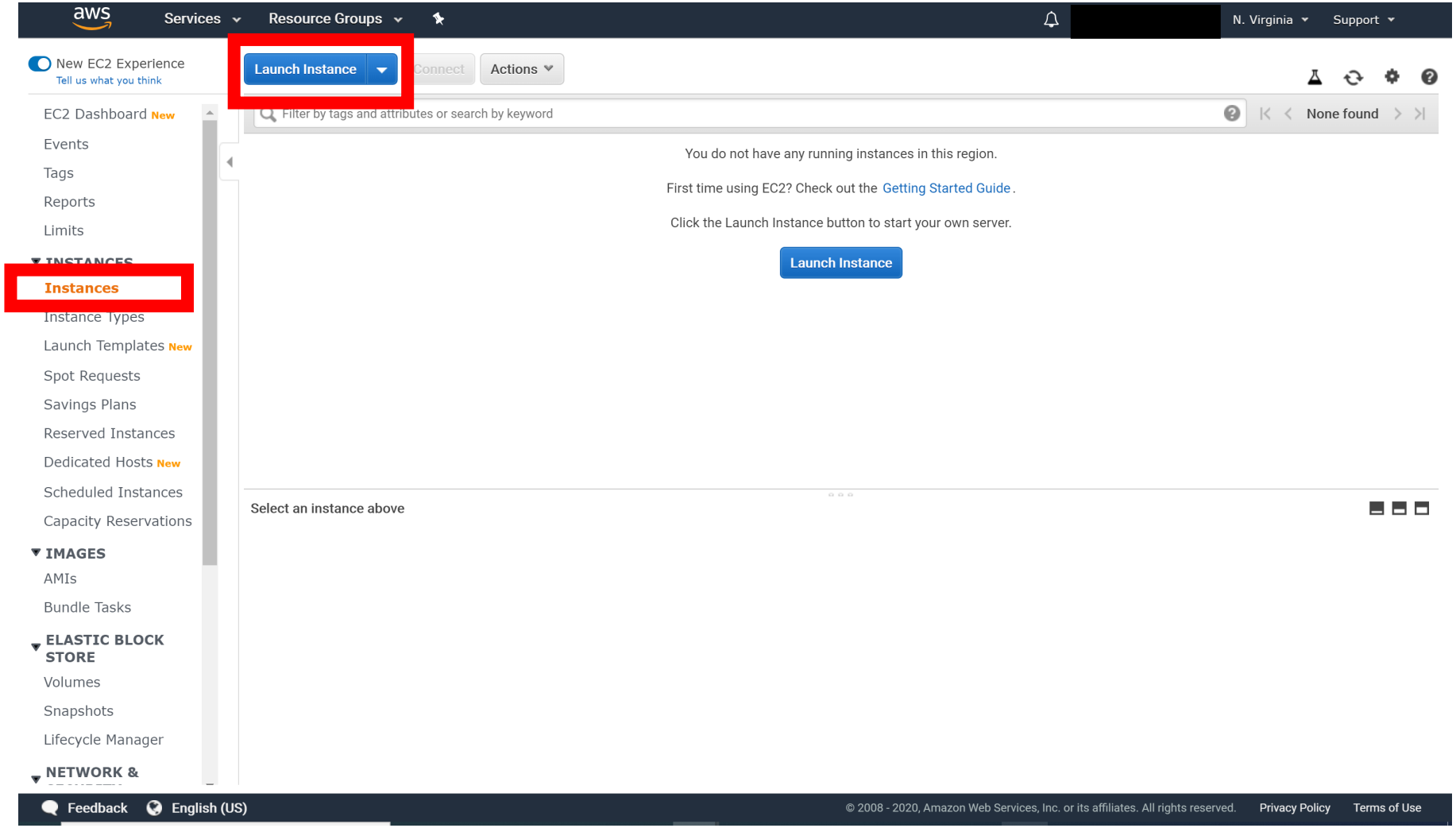
[7] This is a version of the Amazon FPGA Developers AMI version 1.6 in which we have pre-installed Chipyard, FireSim, the software toolchain, and other dependancies and configurations found on the `ee290` branch of Chipyard, in order to save you time

Berkeley Architecture Research

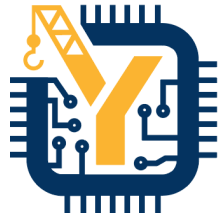
Setting Up Your Manager Instance

Select “Instances” on the left menu of the EC2 console.

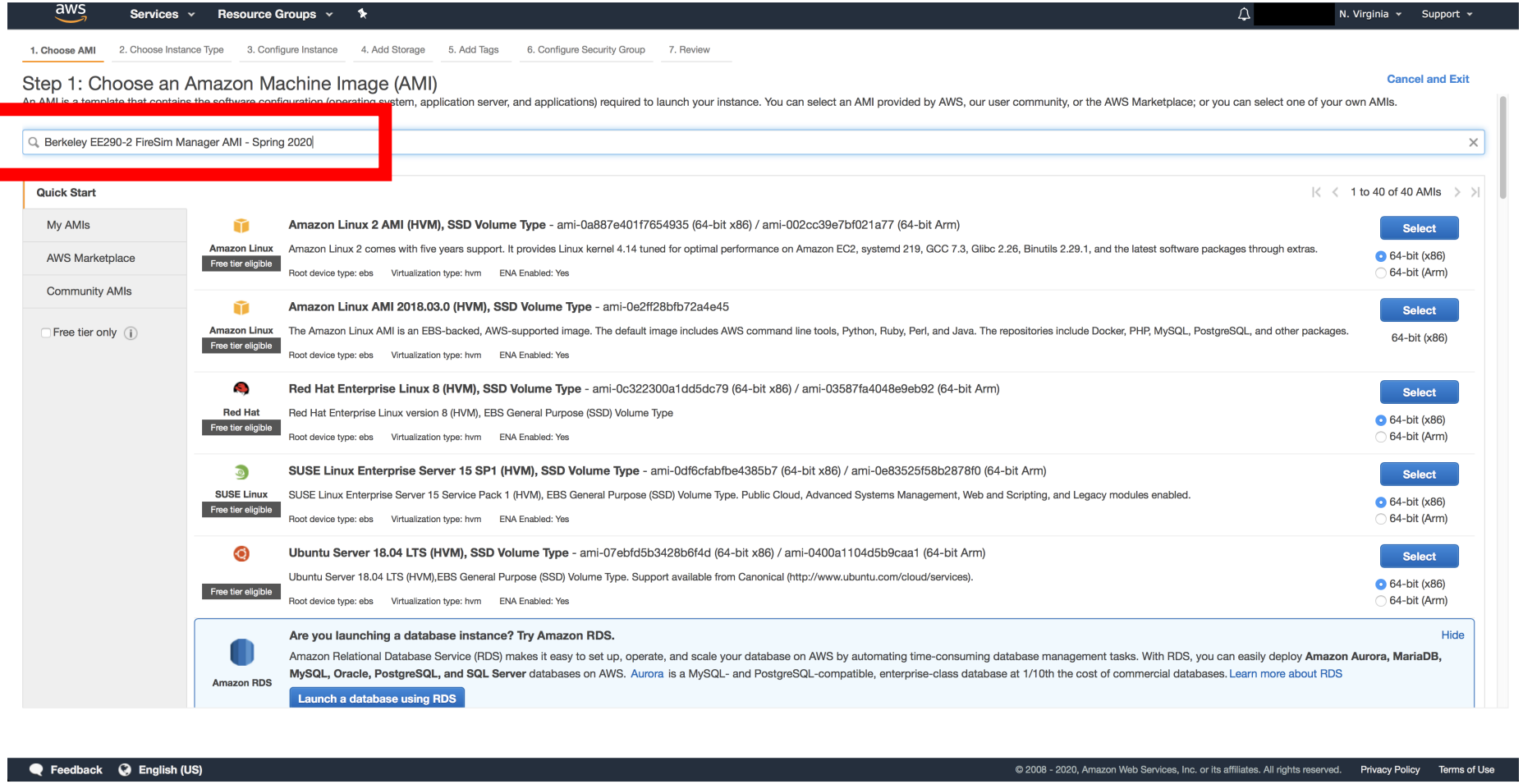
Then select “Launch Instance” on the top left blue button



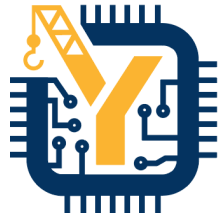
Setting Up Your Manager Instance



Search for “Berkeley EE290-2 FireSim Manager AMI – Spring 2021”



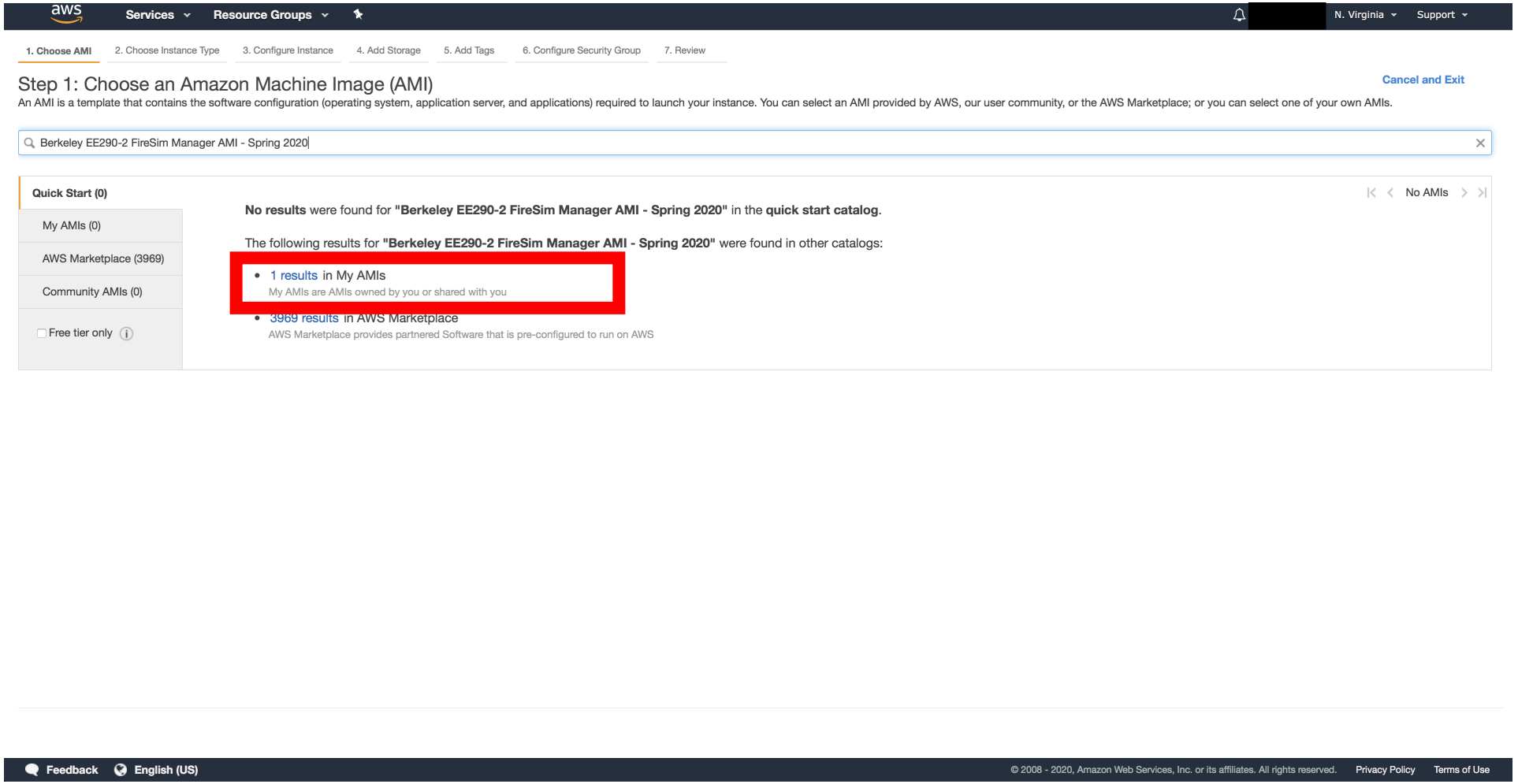
Setting Up Your Manager Instance



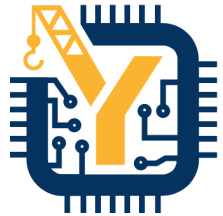
You should see 1 result in “My AMIs”.

If you don’t see a result, please make sure you have filled out the form which allocate AWS credit for you. Otherwise, please email us.

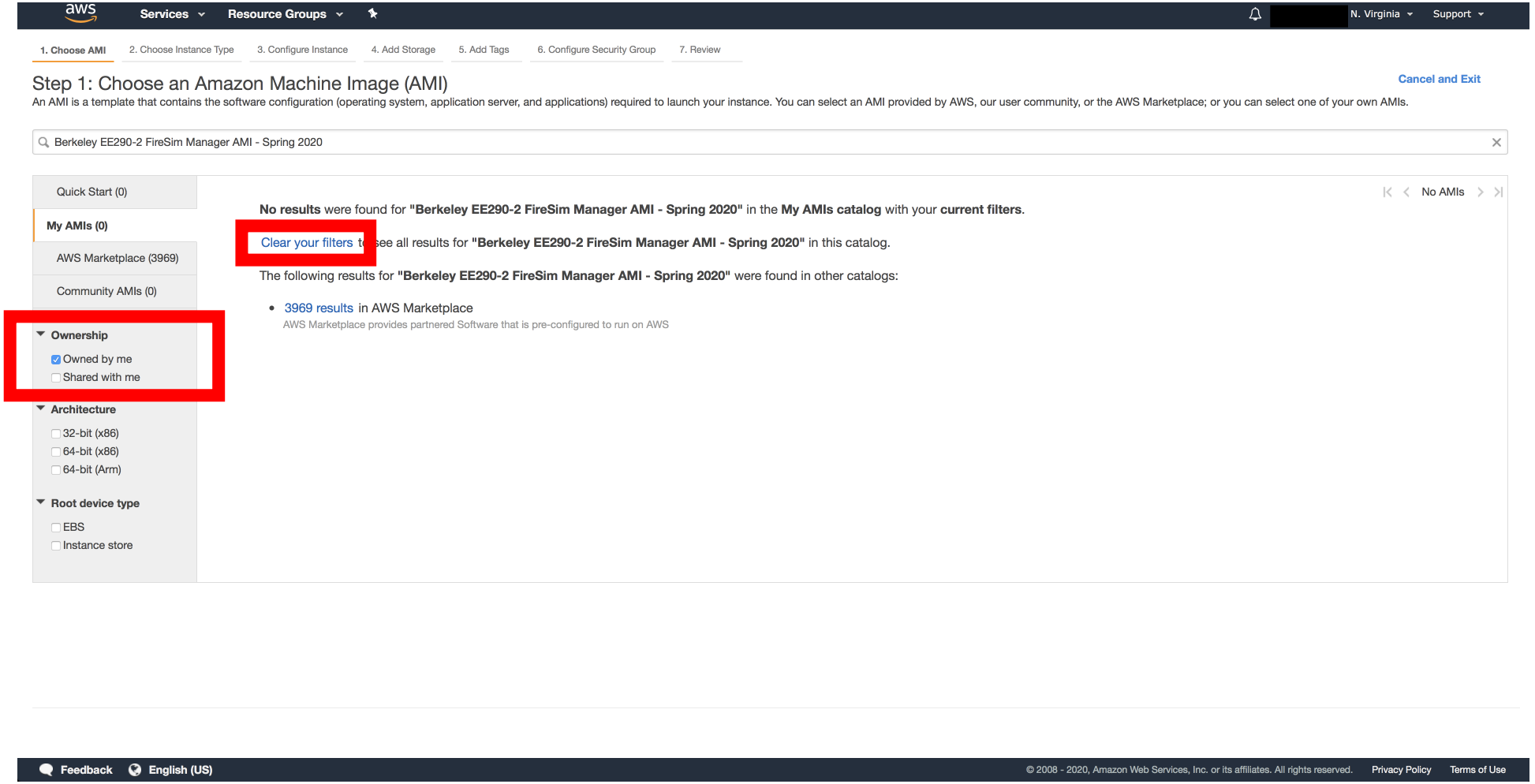
Choose this result.



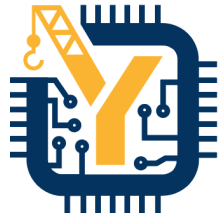
Setting Up Your Manager Instance



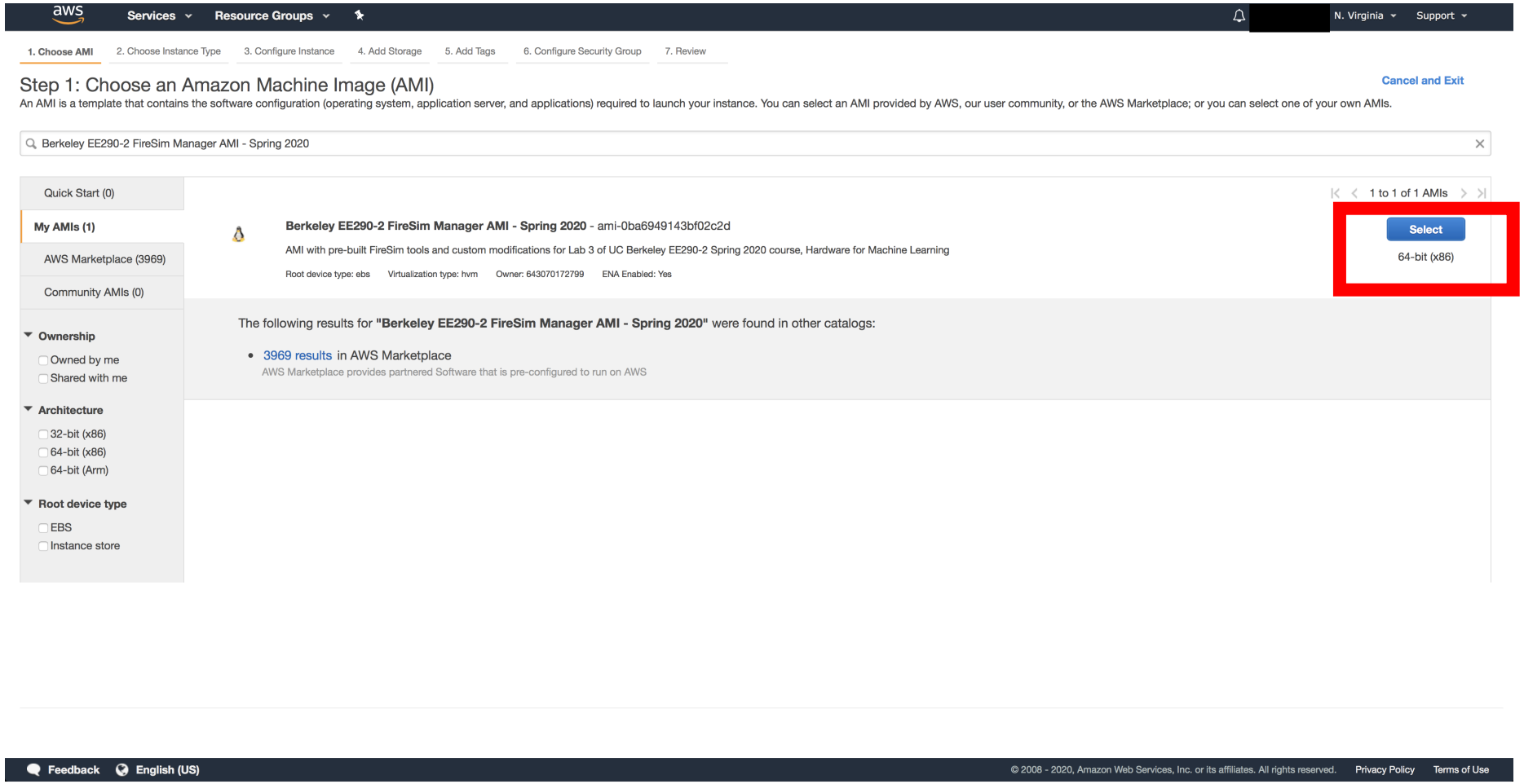
You will likely need to clear you filters, since the filters by default filter only AMIs that are owned by you (and this one is an AMI that we shared with you)



Setting Up Your Manager Instance



Select “Berkeley EE290-2 FireSim Manager AMI – Spring 2021”

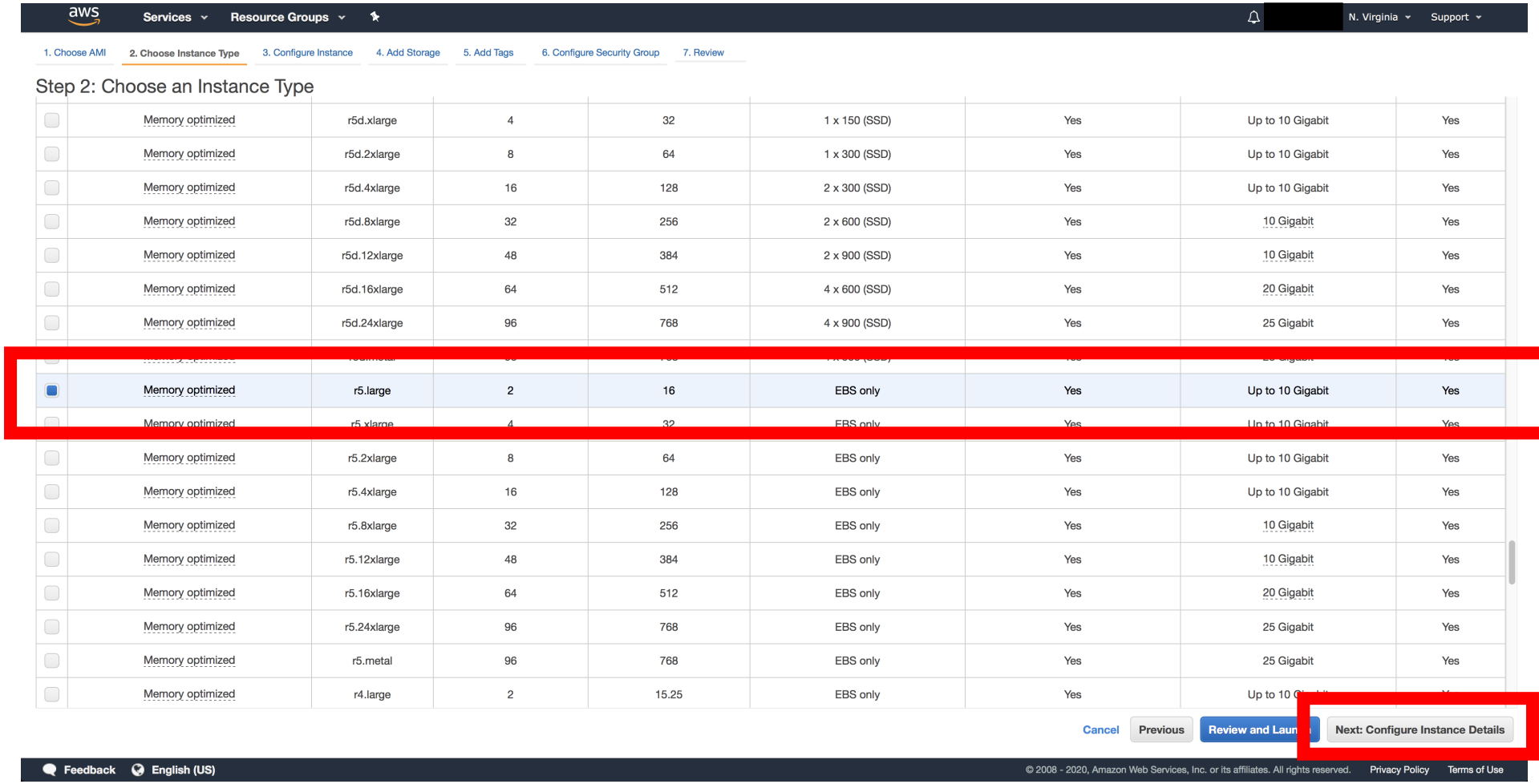


Setting Up Your Manager Instance

Select an r5.large instance type.

We have chosen this instance type as a balance between cost and performance. If you prefer a faster instance that will cost more, you may choose c5.4xlarge (but keep track of your billing!)

Proceed with “Next: Configure Instance Details” (**Not “Review and Launch”**)

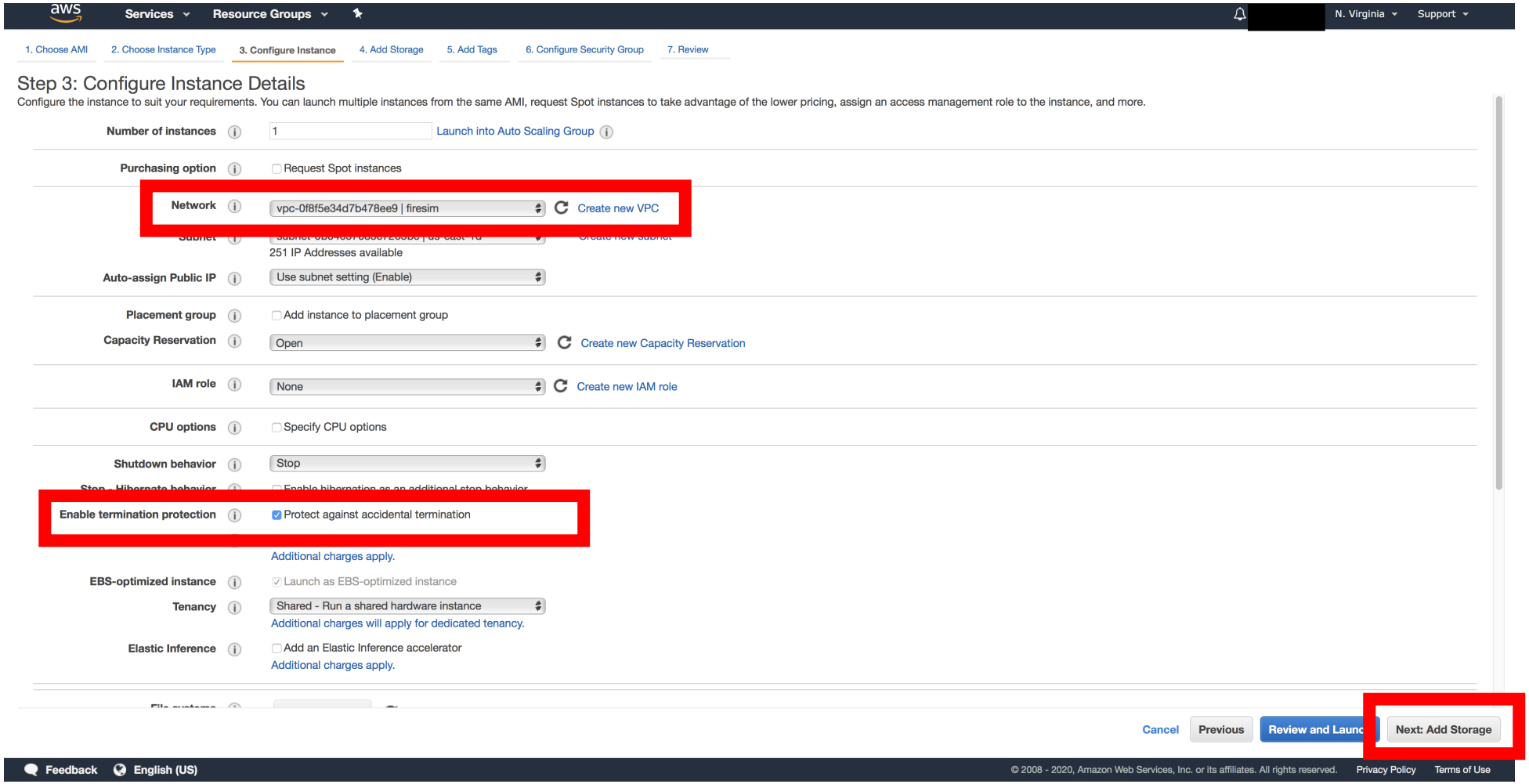


Setting Up Your Manager Instance

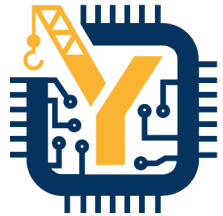
Select the “firesim” VPC under “Network”

Protect the instance against accidental termination

Proceed with “Next: Add Storage” (**Not “Review and Launch”**)

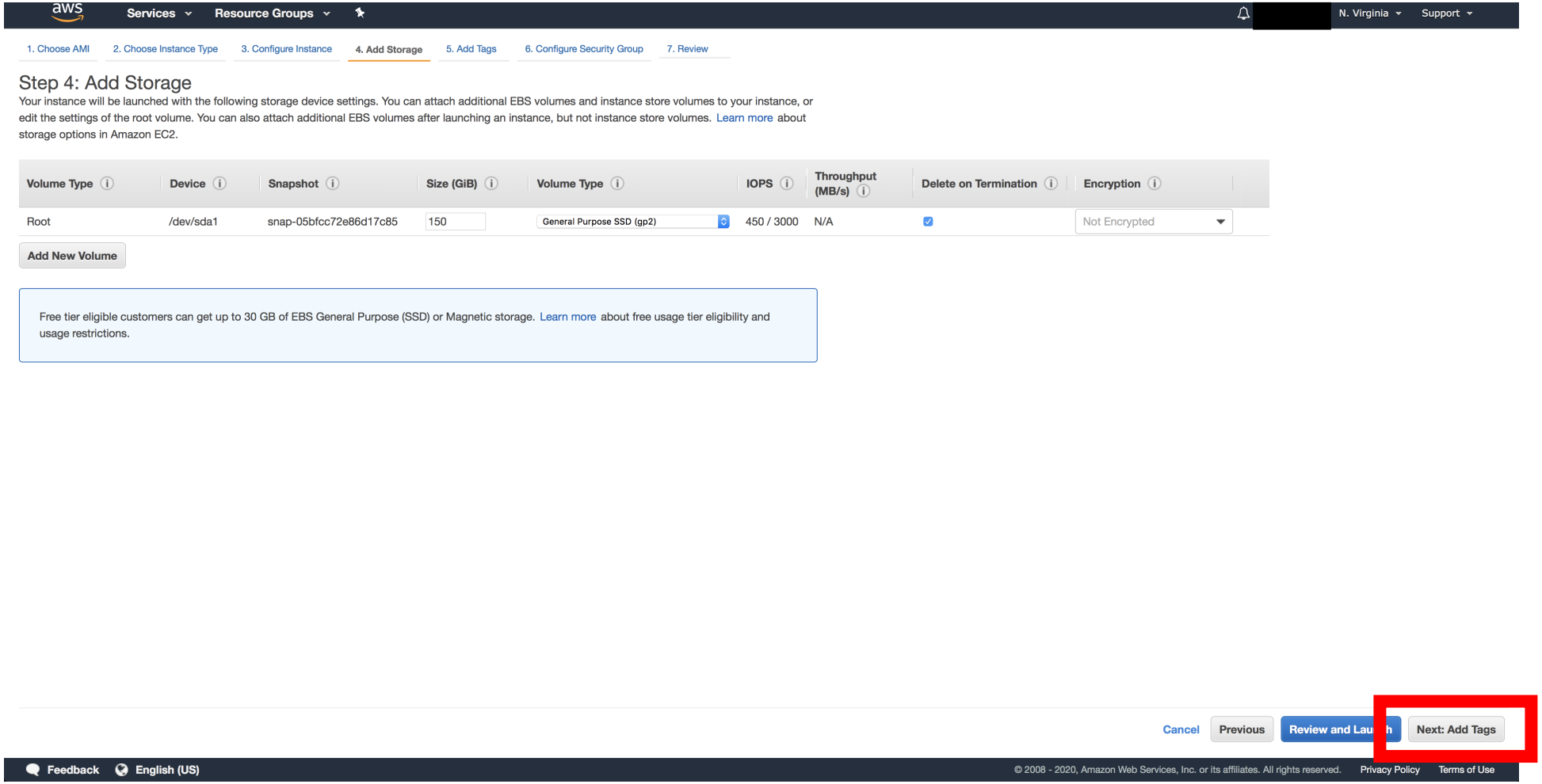


Setting Up Your Manager Instance

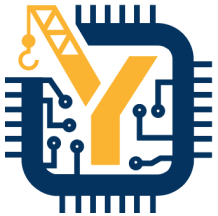


150 GiB of storage should be enough

Proceed with “Next: Add Tags” (**Not “Review and Launch”**)

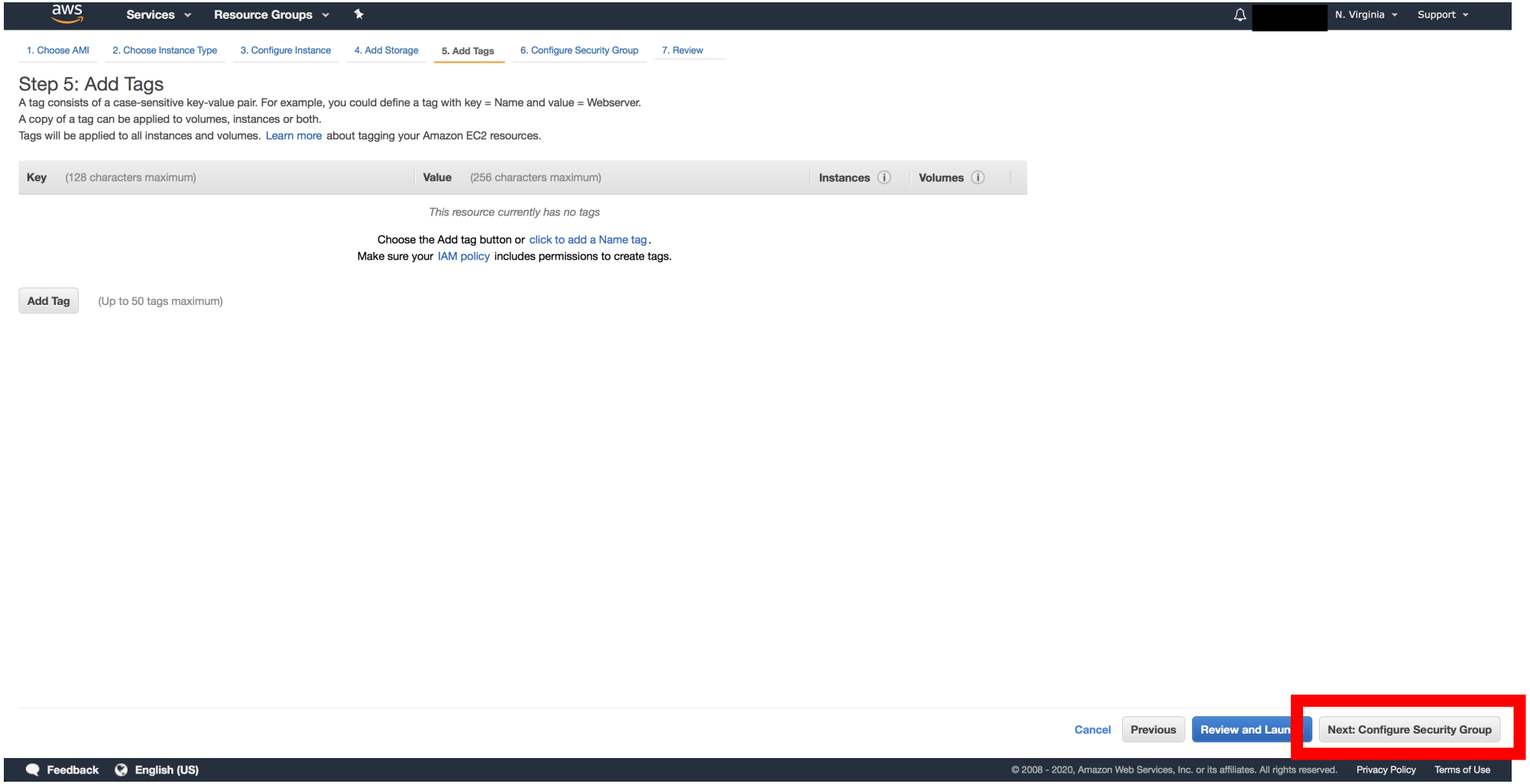


Setting Up Your Manager Instance

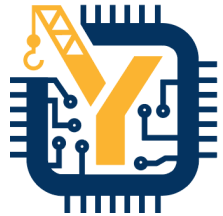


No need to do anything with tags

Proceed with "Next: Configure Security Group" (**Not "Review and Launch"**)

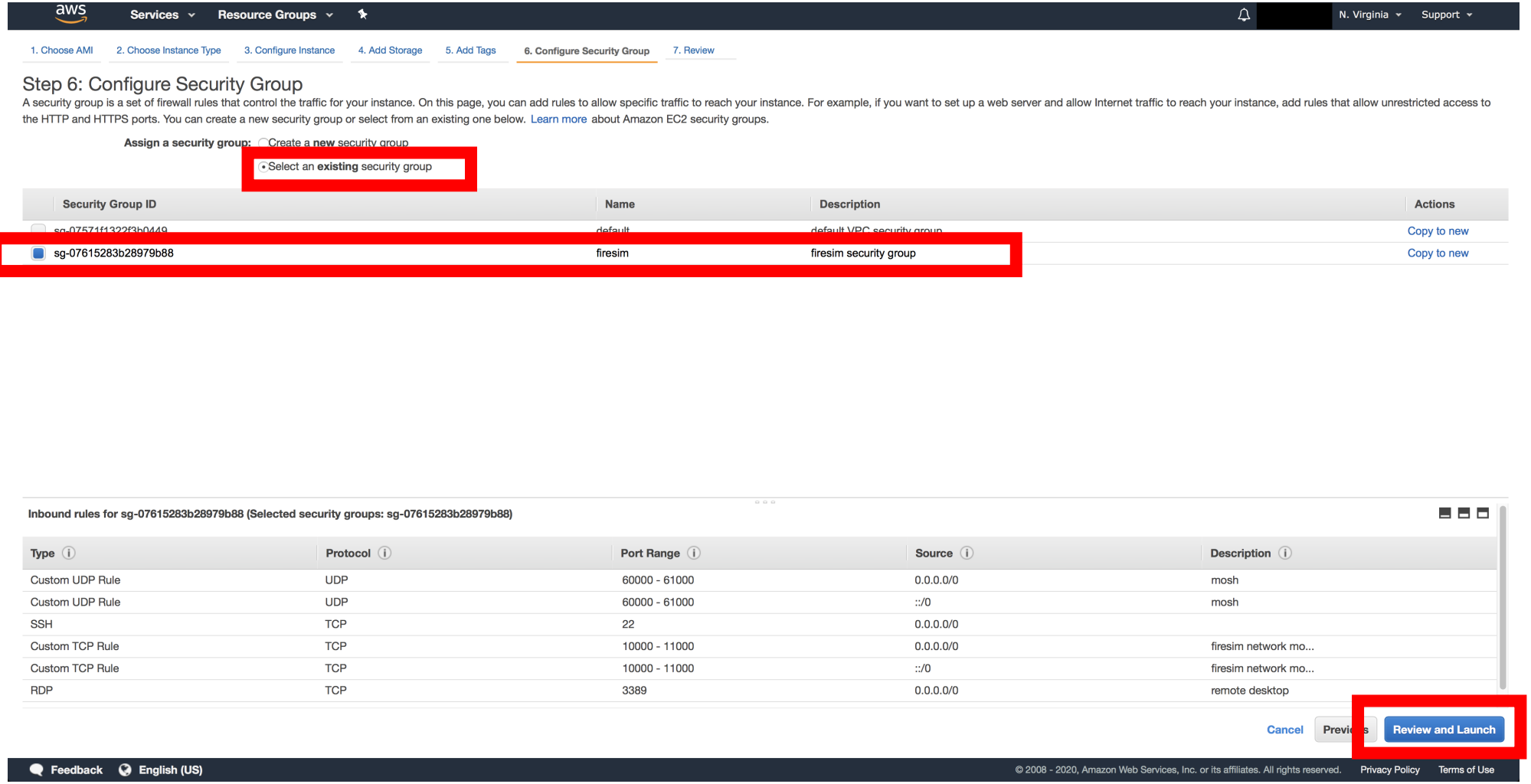


Setting Up Your Manager Instance



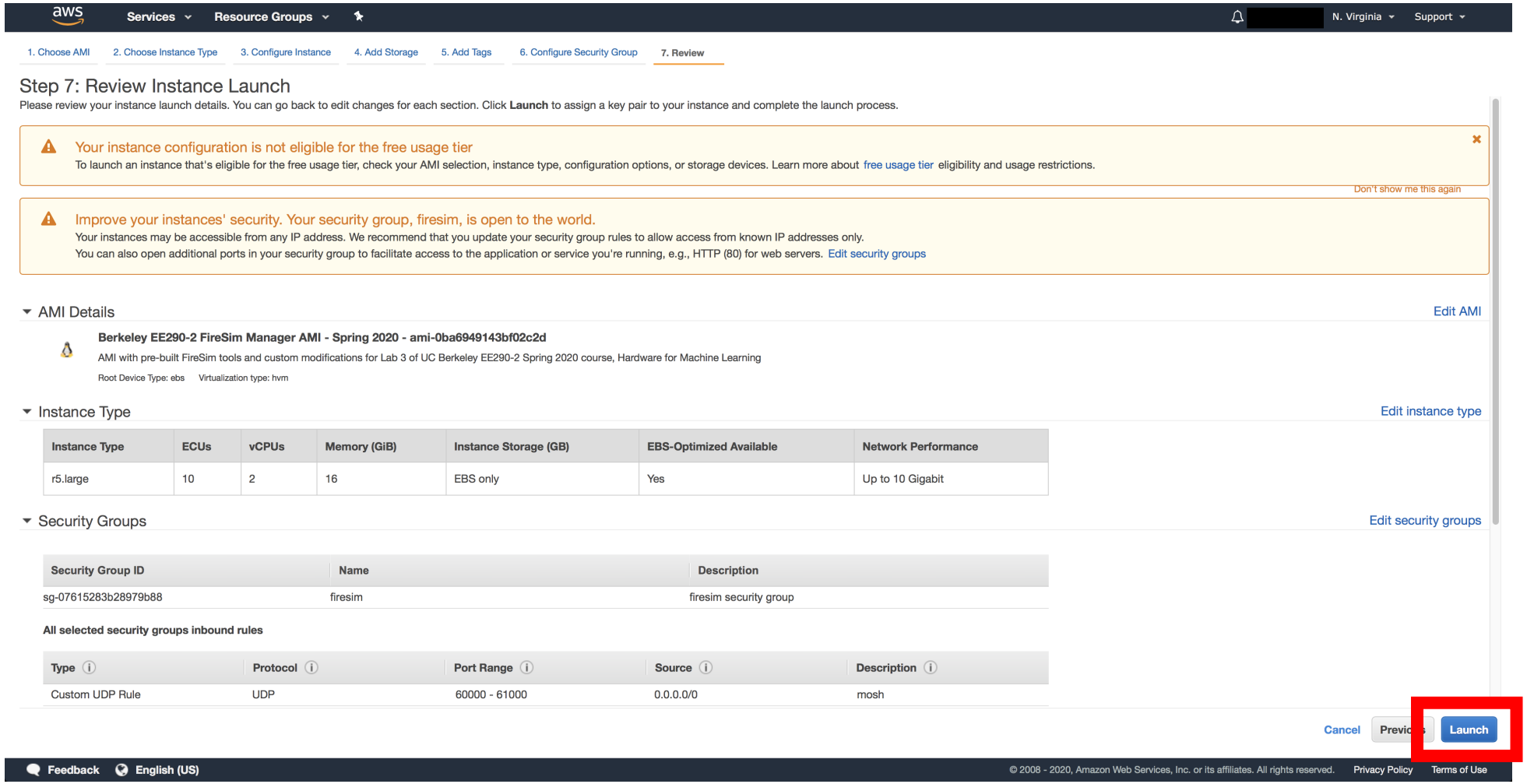
Select an existing security group, and select the firesim security group

Proceed with “Review and Launch”



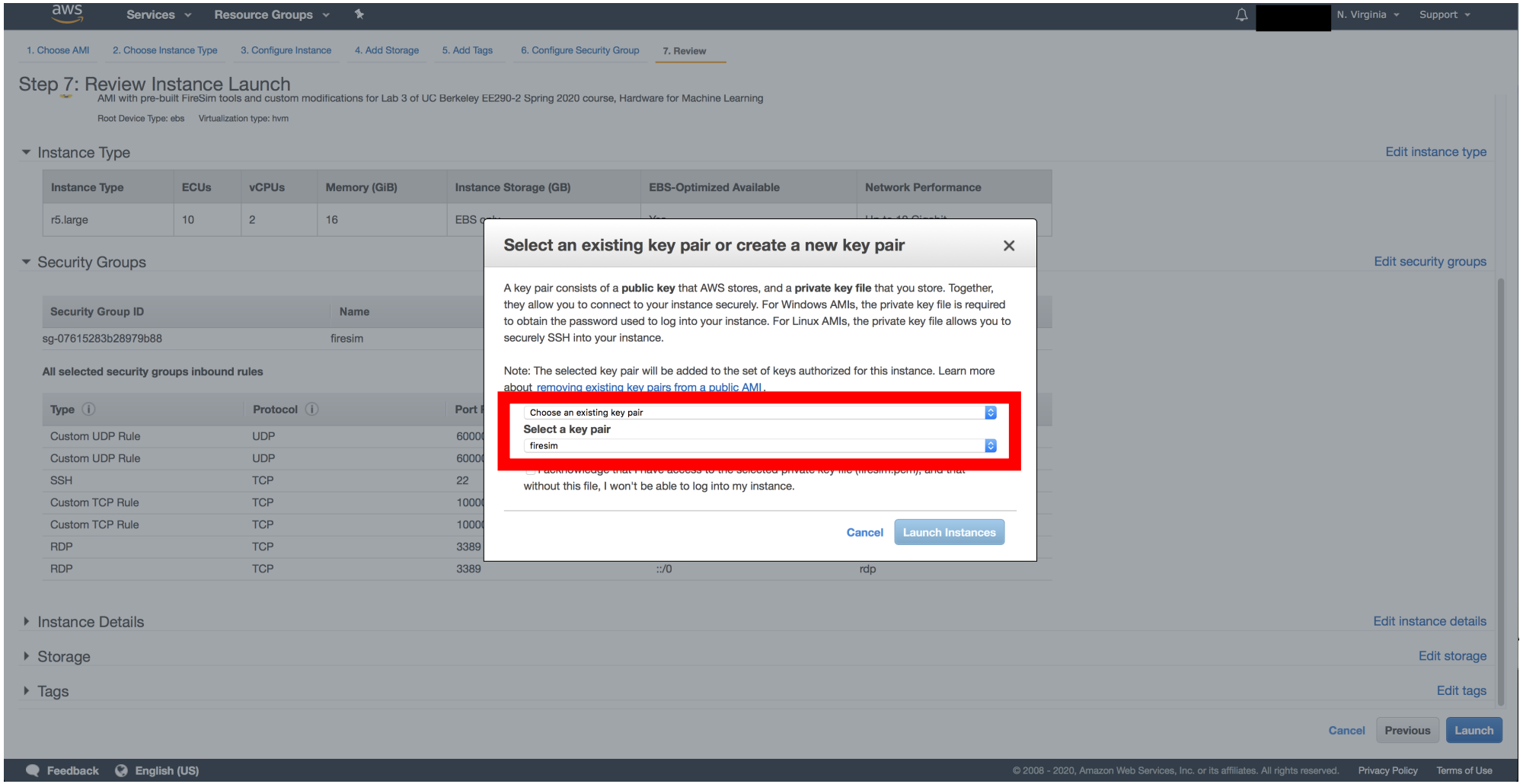
Setting Up Your Manager Instance

Review the details of the instance, and click “Launch”

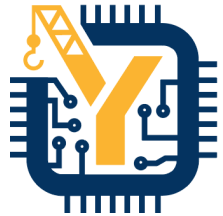


Setting Up Your Manager Instance

Choose the “firesim” key pair, which we created in the earlier setup stages.

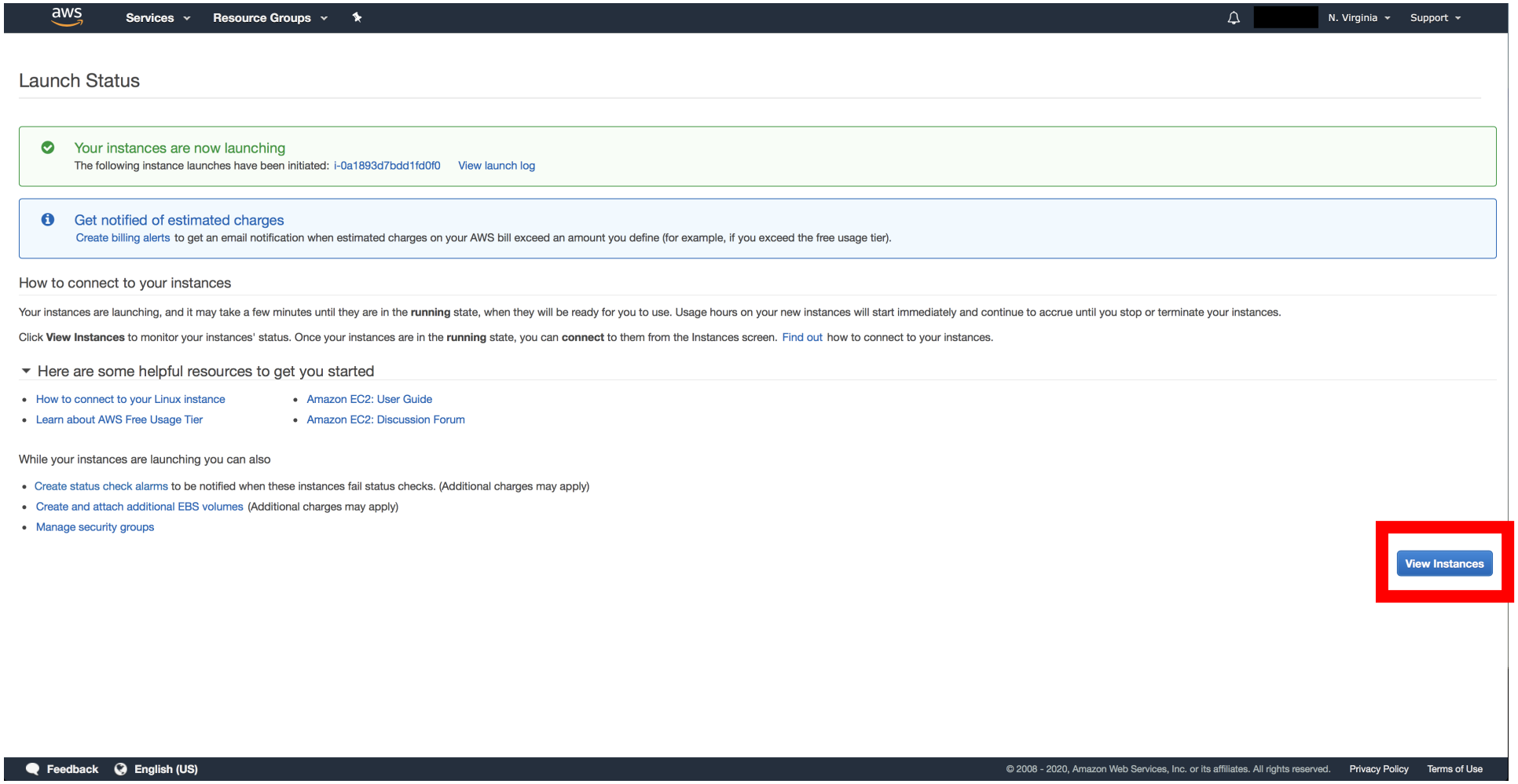


Setting Up Your Manager Instance

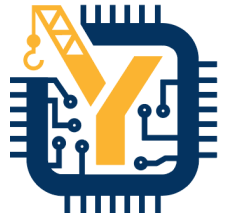


Your instance is now launching.

View it in the instances list



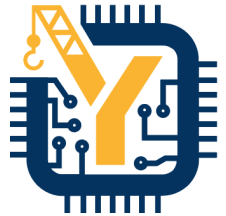
Setting Up Your Manager Instance



ssh/mosh into your manager instance: `ssh -i firesim.pem centos@<INSTANCE_IP>`

```
The authenticity of host '54.226.188.196 (54.226.188.196)' can't be established.
ECDSA key fingerprint is SHA256:MEBfVUYG7SYhdWjNReFav8SfqxvKD7tx2TALPbFrpCk.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.226.188.196' (ECDSA) to the list of known hosts.
Last login: Fri Feb 21 18:47:37 2020 from a6.millennium.berkeley.edu

 ___ ___ ___   _     ___ _____  __    _   __  __ ___
| __| _ \/ __| /_\   |   \| __\ \ / /  /_\ |  \/  |_ _|
| _||  _/ (_ |/ _ \  | |) | _| \ V /  / _ \| |\/| || |
|_| |_|  \___/_/ \_\ |___/|___| \_/  /_/ \_\_|  |_|___|
AMI Version:         1.6.0
Readme:              /home/centos/src/README.md
GUI Setup Steps:     /home/centos/src/GUI_README.md
GUI Setup script:    /home/centos/src/scripts/setup_gui.sh
AMI Release Notes:   /home/centos/src/RELEASE_NOTES.md
Xilinx Tools:        /opt/Xilinx/
Developer Support:   https://github.com/aws/aws-fpga/blob/master/README.md#developer-support
Centos Common code: /srv/git/centos-git-common
centos@ip-192-168-3-54.ec2.internal:~$
```

Setting Up Your Manager Instance



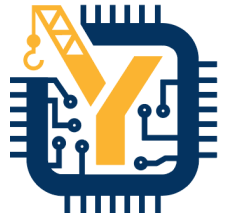
Copy the `firesim.pem` file from your local machine into the home directory of your manager instance

```
centos@ip-192-168-3-54.ec2.internal:~$ scp <LOCAL_MACHINE>/firesim.pem ~/
```

Source the `env.sh` and `sourceme-f1-manager.sh` files in `chipyard` and `chipyard/sims/firesim`. You will need to do this every time you open a terminal session in your manager instance.

```
centos@ip-192-168-3-54.ec2.internal:~$ cd chipyard/sims/firesim
centos@ip-192-168-3-54.ec2.internal:~/chipyard/sims/firesim$ source sourceme-f1-manager.sh
Identity added: /home/centos/firesim.pem (/home/centos/firesim.pem)
success: firesim.pem added to ssh-agent
```

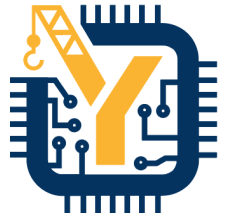
Setting Up Your Manager Instance



Run `firesim managerinit` to initialize your manager. This will require similar details to the `aws configure` command from earlier

```
centos@ip-192-168-3-54.ec2.internal:~/chipyard/sims/firesim$ firesim managerinit
FireSim Manager. Docs: http://docs.fires.im
Running: managerinit
Running aws configure. You must specify your AWS account info here to use the FireSim Manager.
[localhost] local: aws configure
AWS Access Key ID [None]: XXXXXXXXXXXX
AWS Secret Access Key [None]: XXXXXXXXXXXX
Default region name [None]: us-east-1
Default output format [None]: json
Backing up initial config files, if they exist.
Creating initial config files from examples.
If you are a new user, supply your email address [abc@xyz.abc] for email notifications (leave blank if you do
not want email notifications): john.doe@berkeley.edu
You should receive a message at
john.doe@berkeley.edu
asking to confirm your subscription to FireSim SNS Notifications. You will not
receive any notifications until you click the confirmation link.
FireSim Manager setup completed.
```

Setting Up Your Manager Instance

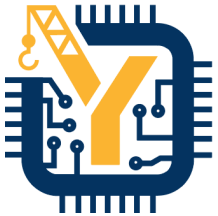


We're done with setting up the FireSim manager instance!

You can now either:

- Proceed to execute the second part of the lab assignment (evaluation of complete DNNs on Gemmini in FirSim)
- Stop you manager instance, until you are ready for the second part of the assignment

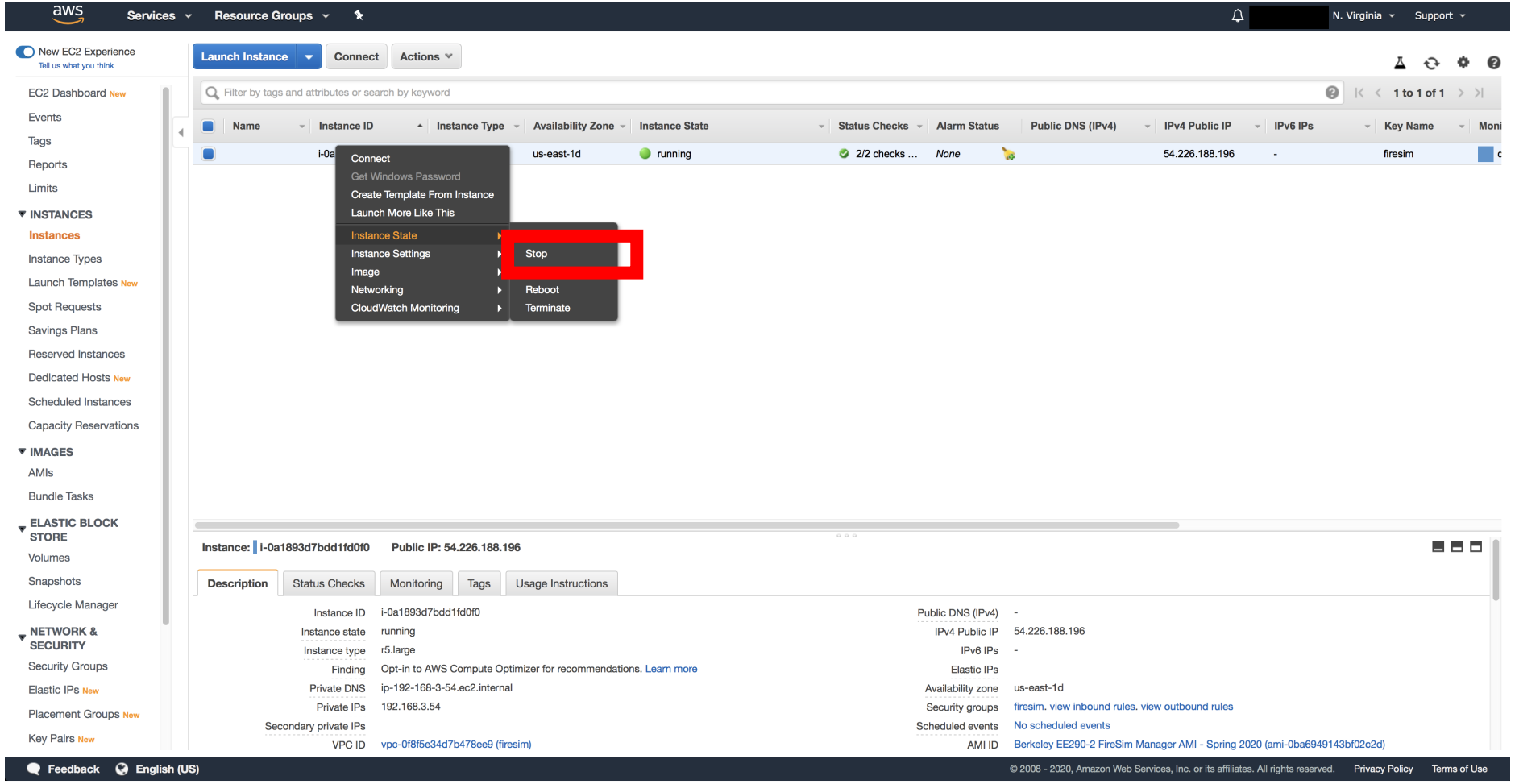
Stopping Your Manager Instance



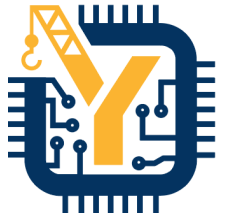
Stopping you manager instance:

Right click on the instance entry in the table, and within the “Instance State” menu, select “Stop”.

A stopped instance is like “sleep” on your laptop. While the instance is stopped (as opposed to terminated), you are still charged a certain amount of money, but significantly less.



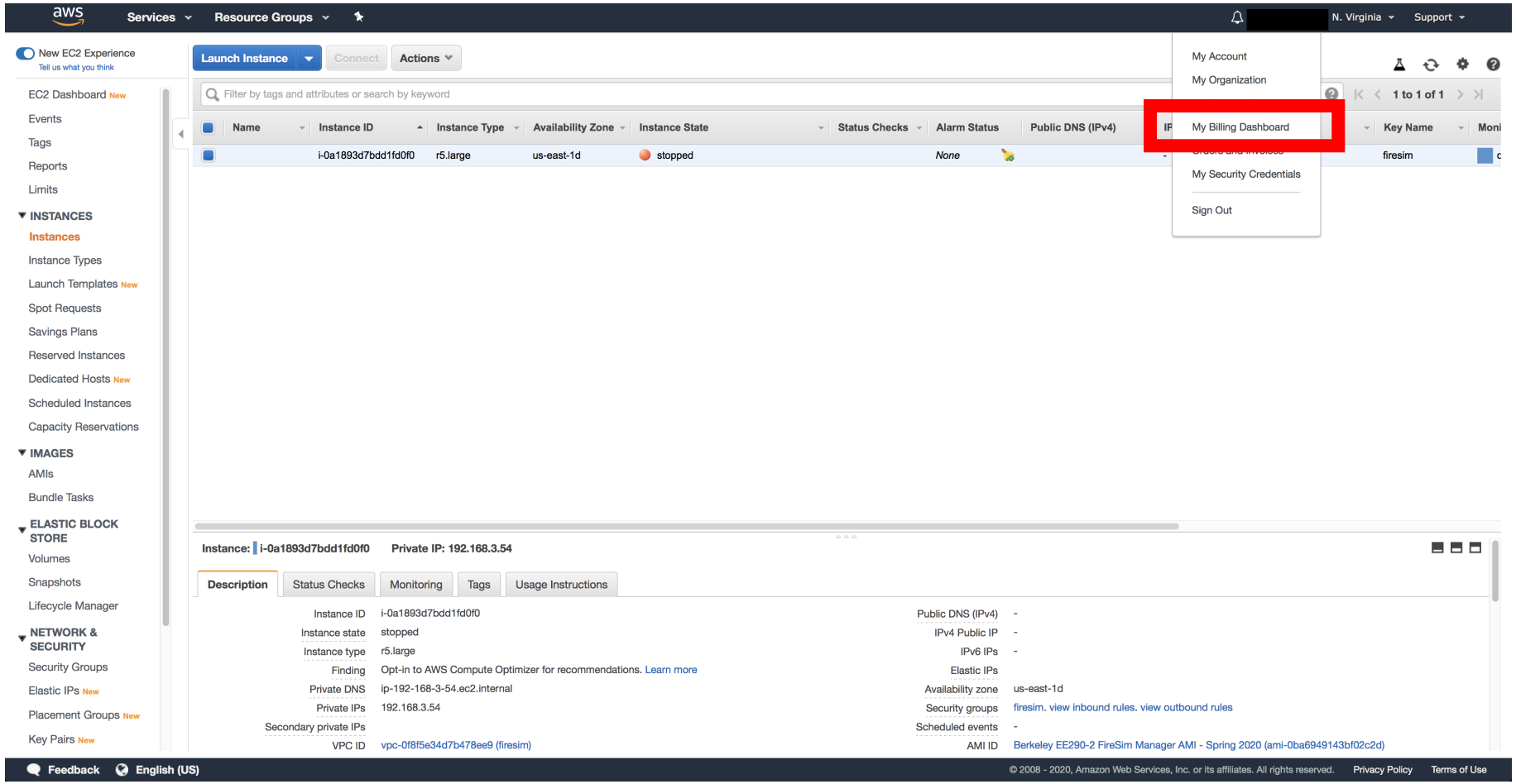
Billing



- We really don’t want you to get charged any (real) money
- Check your billing!
- Billing in AWS is not immediate. It is delayed by several hours/days after your actual usage
- If you’re running out of credits, please contact the course staff

Billing

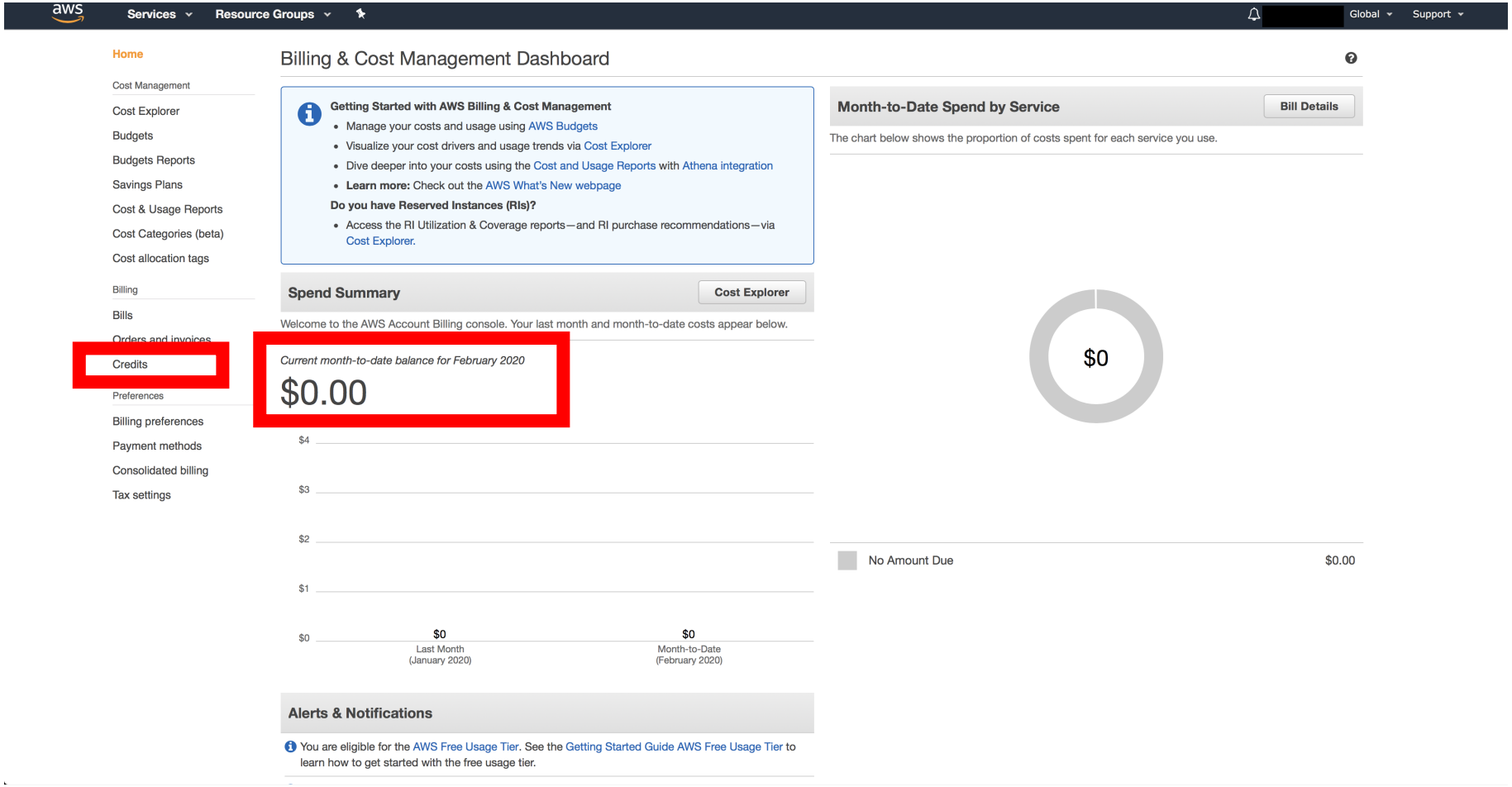
In the user menu on the top right corner of your EC2 console, select “My Billing Dashboard”



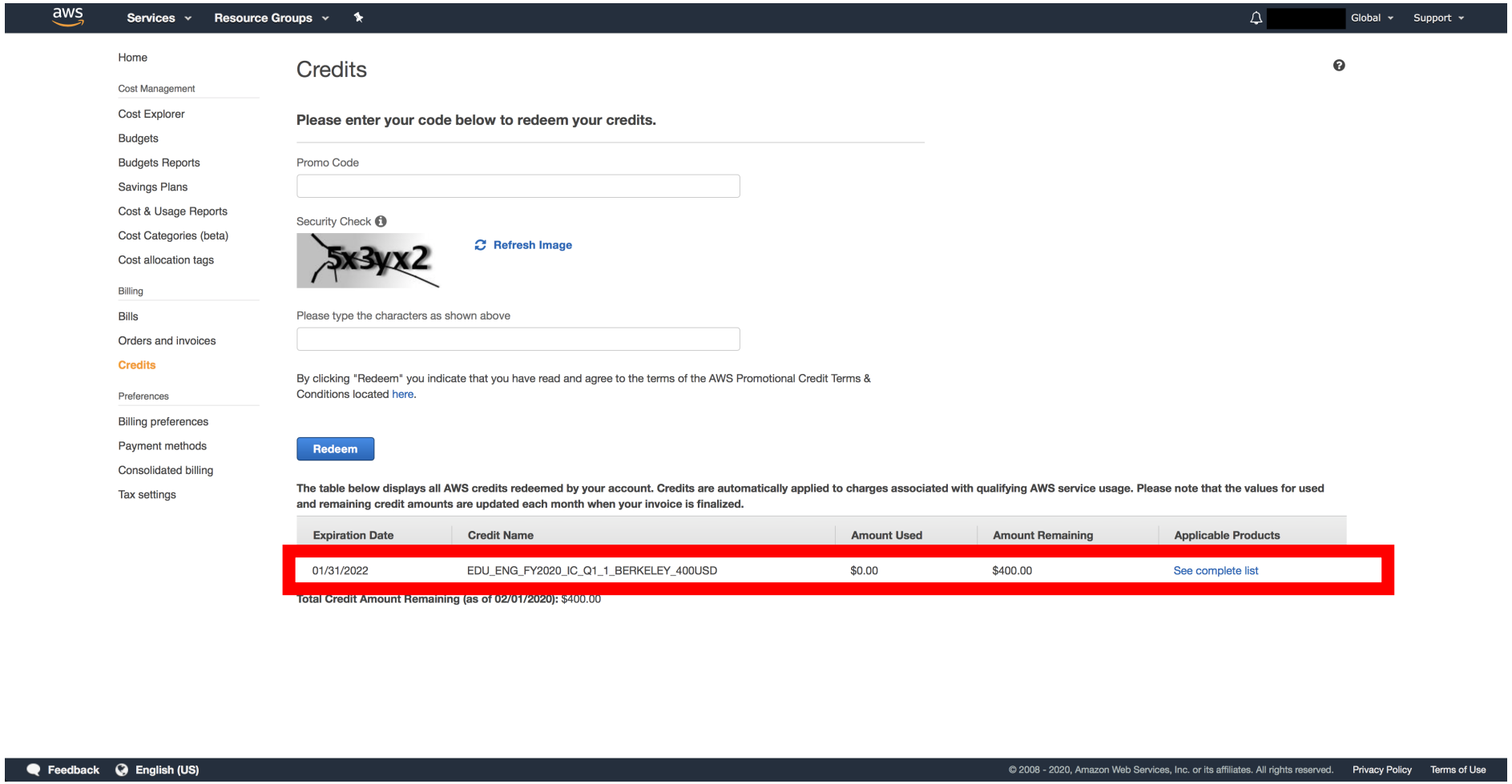
Billing

Spending summary: make sure this doesn't exceed $150 (because this will mean you're getting very close to your $200 credit limit)

You may also view your promo code credit usage in the "Credit" section of the menu of the left



Billing

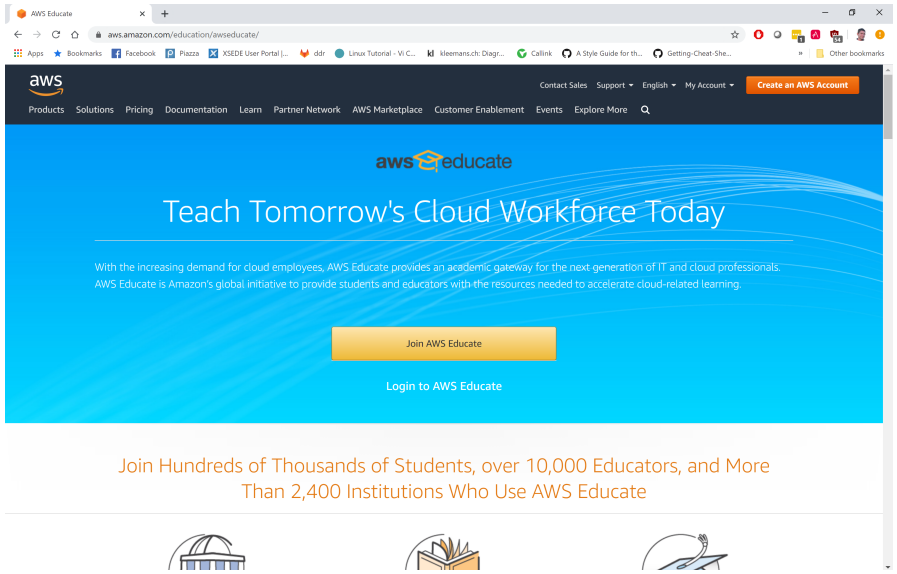


You will see you class credit entry in the table, and the amount used out of it.

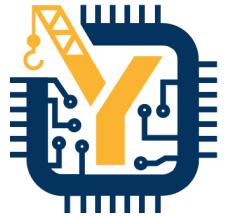
If you are running out of credits, please contact the course staff. We have some extra promo codes for special cases.

AWS Educate

- Another method to get more credits, is through AWS Educate
- Can get you an extra $100 credits
- https://aws.amazon.com/education/awseducate/
- Make sure NOT to choose the Starter Kit (otherwise, you won't have access F1 instances or the promo codes)



AWS Recap



- Opening an AWS account
- EE290-2 AWS details form
- FireSim infrastructure setup
- FireSim manager instance
- Billing tracking