# EE241B : Advanced Digital Circuits

## Lecture 3 – Building SoCs

Borivoje Nikolić

CHIPYARD

https://github.com/ucb-bar/chipyard

# Announcements

- Assignment 1 posted this week
- No class on February 16 (ISSCC)

# Projects

- Done in pairs or alone
- Due dates:
  - Abstract: February 19
    - Title, a paragraph and 5 references
  - Midterm report: March 19, before Spring break
    - 4 pages, paper study
  - Final report: May 1
    - 6 pages
    - Design
  - Final exam is on April 29 (last class)

# Assigned Reading

On an SoC generator

- A. Amid, et al, “Chipyard: Integrated design, simulation, and implementation framework for custom SoCs,” IEEE Micro, 2020.

On transistor models:

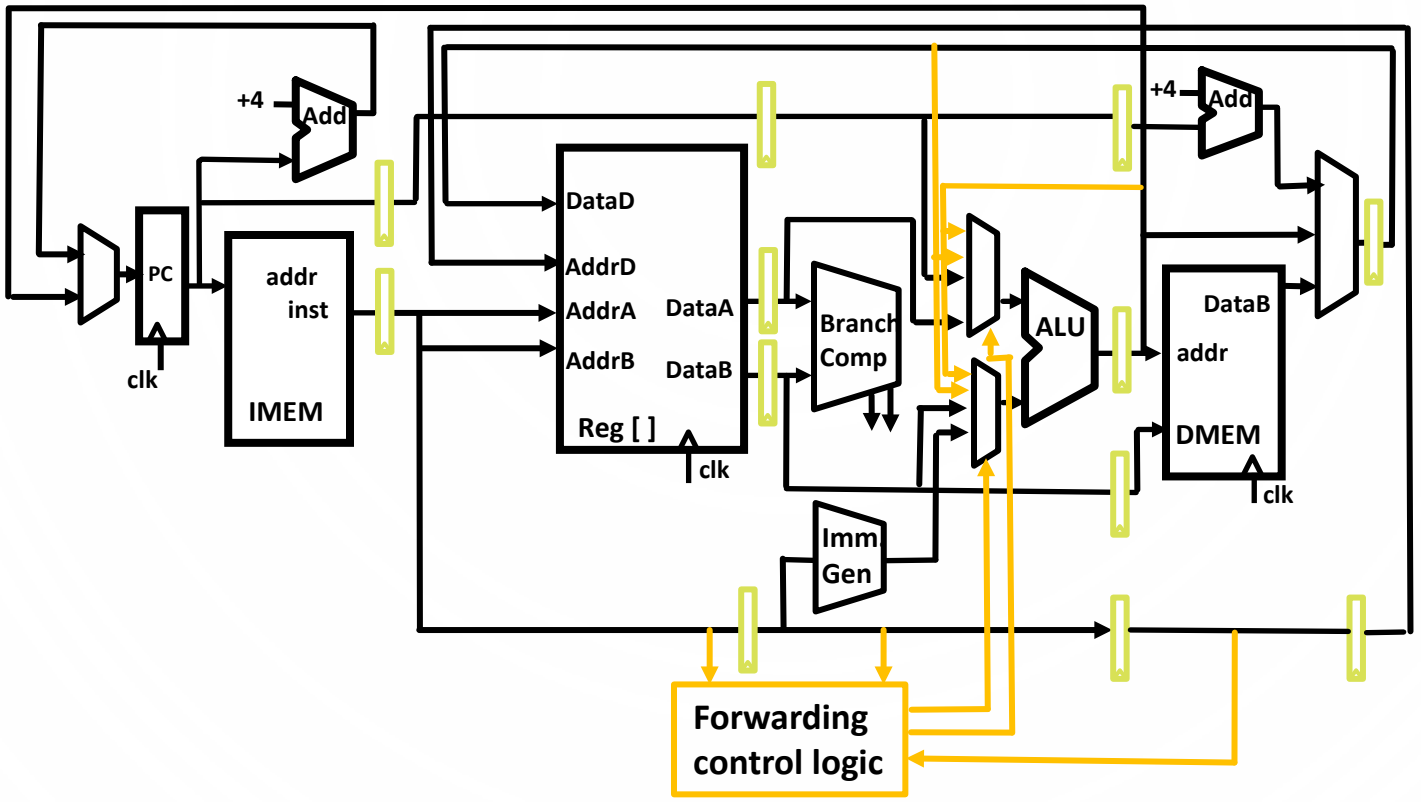
- R.H. Dennard et al, “Design of ion-implanted MOSFET's with very small physical dimensions” IEEE Journal of Solid-State Circuits, April 1974.
  - Just the scaling principles
- C.G. Sodini, P.-K. Ko, J.L. Moll, "The effect of high fields on MOS device and circuit performance," IEEE Trans. on Electron Devices, vol. 31, no. 10, pp. 1386 - 1393, Oct. 1984.
- K.-Y. Toh, P.-K. Ko, R.G. Meyer, "An engineering model for short-channel MOS devices" IEEE Journal of Solid-State Circuits, vol. 23, no. 4, pp. 950-958, Aug. 1988.
- T. Sakurai, A.R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," IEEE Journal of Solid-State Circuits, vol. 25, no. 2, pp. 584 - 594, April 1990.

# Outline

- SoC generator: Chipyard
  - Great for class and other projects

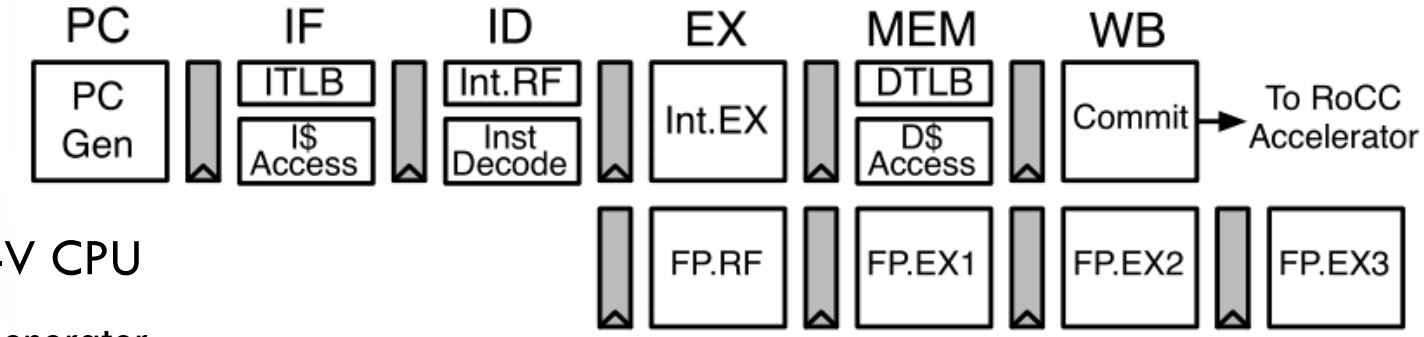
# RISC-V Processor Core

# RISC-V Core




- Everyone has seen (or possibly designed) a 5-stage RISC-V (RV32I) core
- Needs memory (cache/scratchpad), co-processors, peripherals

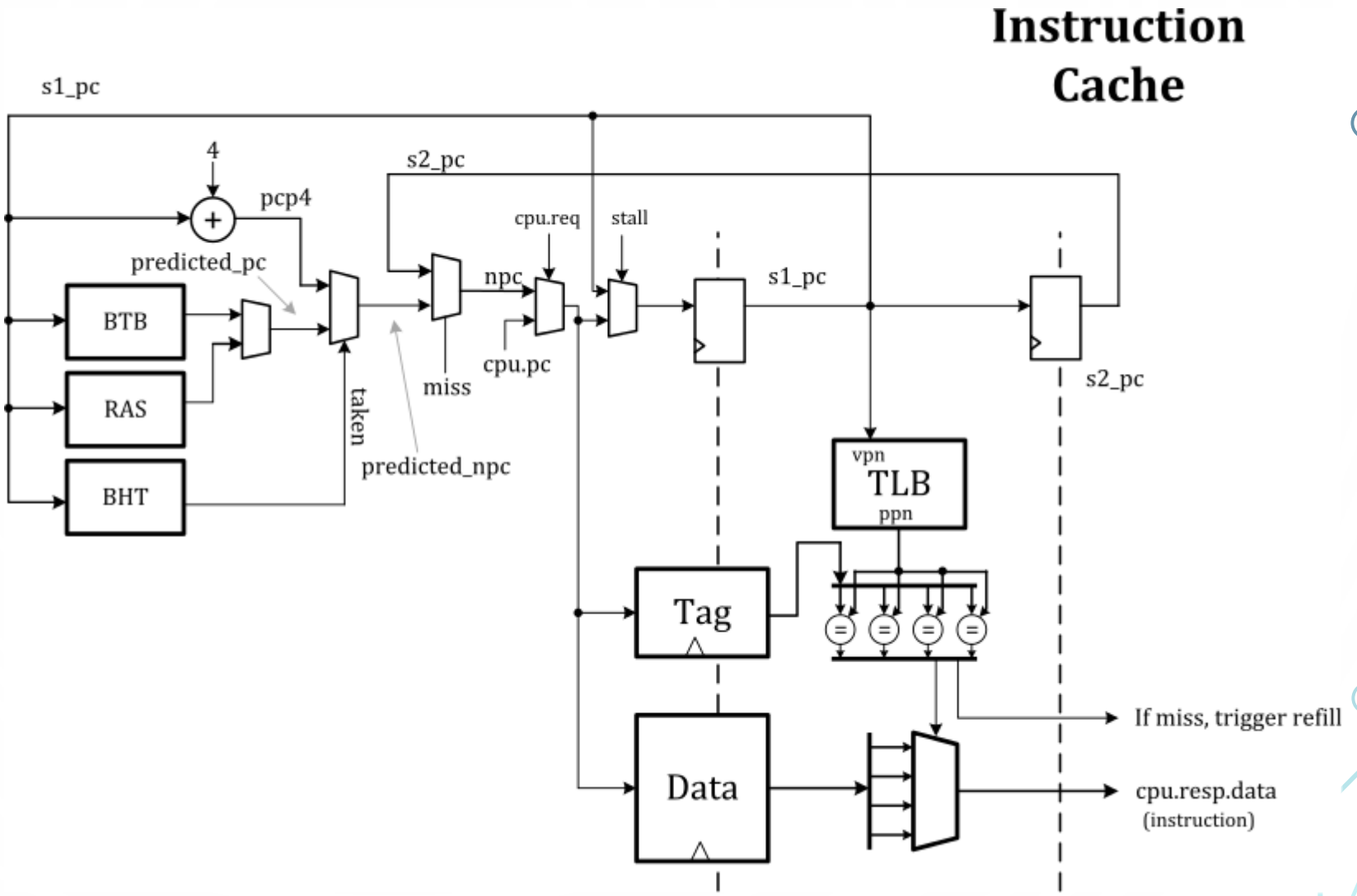
# Rocket In-Order Core




- First open-source RISC-V CPU
  - Designed as a Chisel generator
- 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

# Inside Rocket

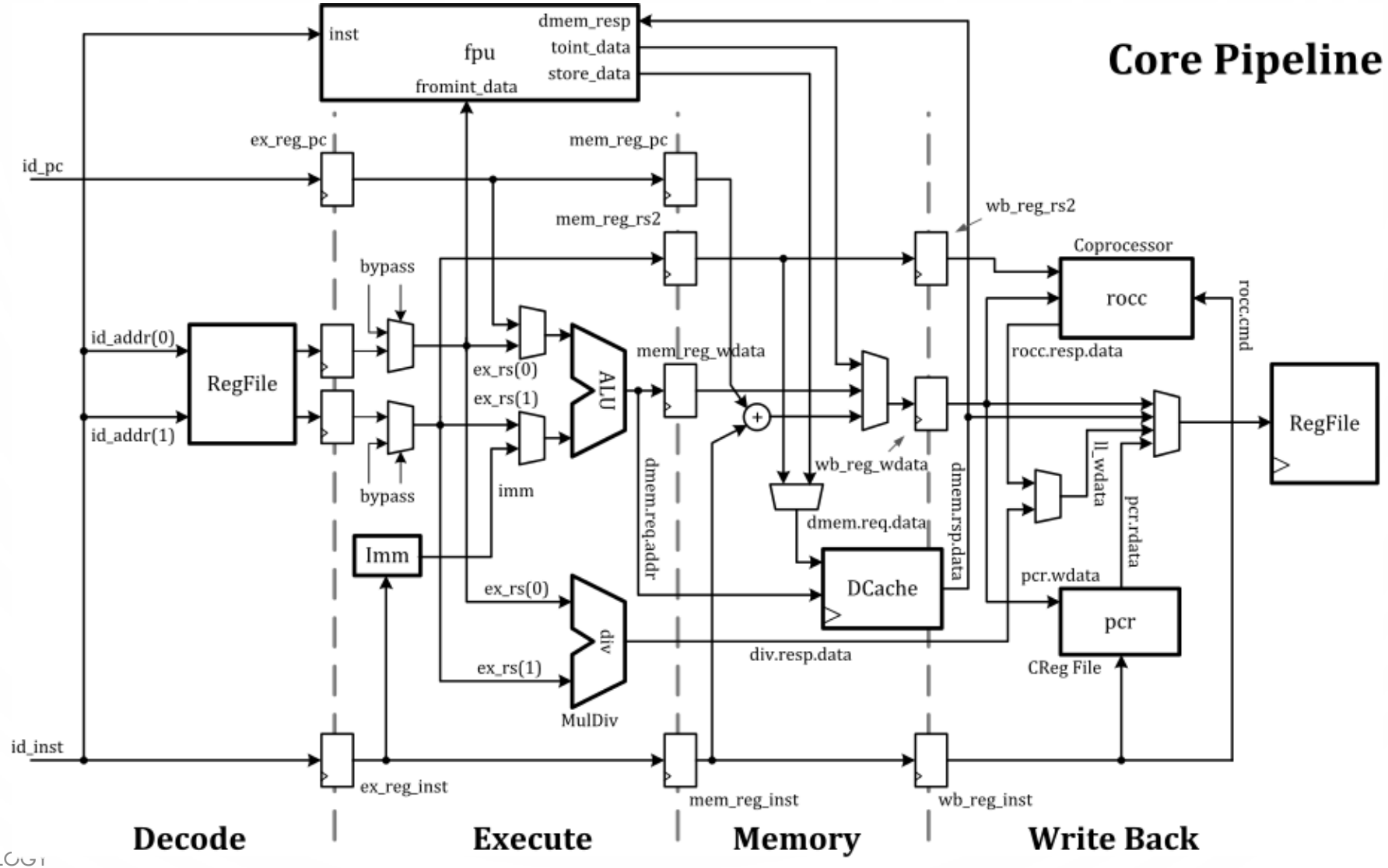
- Front-end
  - Fetches instructions from I$




https://www.cl.cam.ac.uk/~jrrk2/docs/tagged-memory-v0.1/rocket-core/

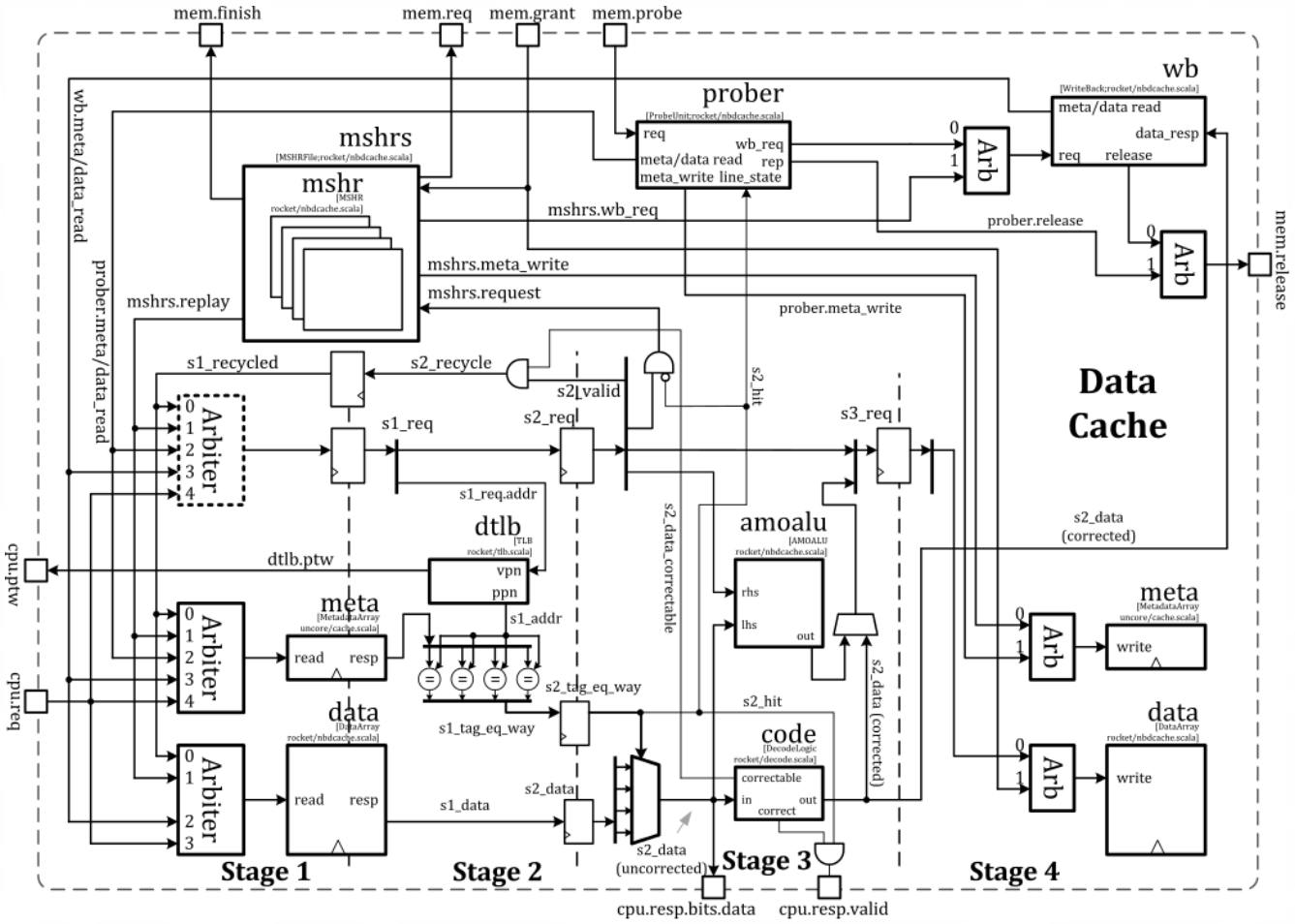
# Rocket Pipeline

- 1 integer ALU/IMUL/IDIV + optional FPU



# Data Cache

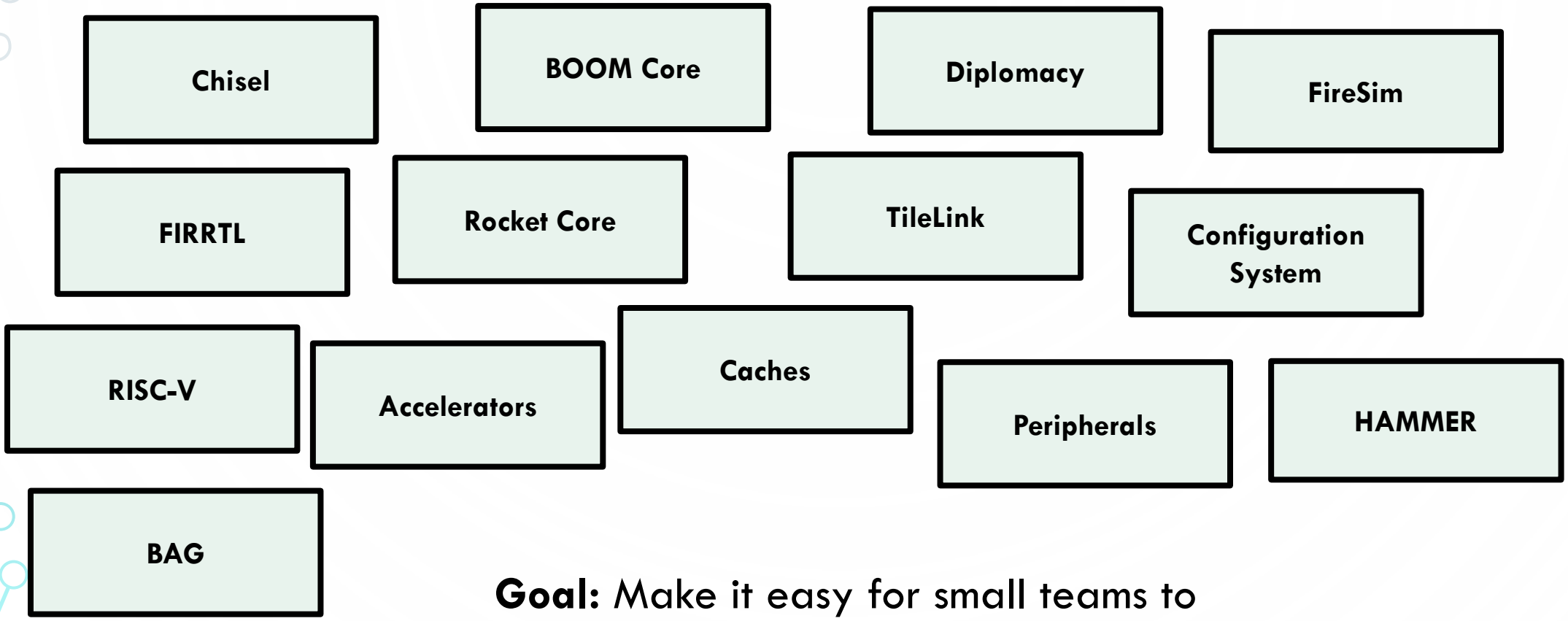
- L1 cache



# Chipyard: SoC Generator

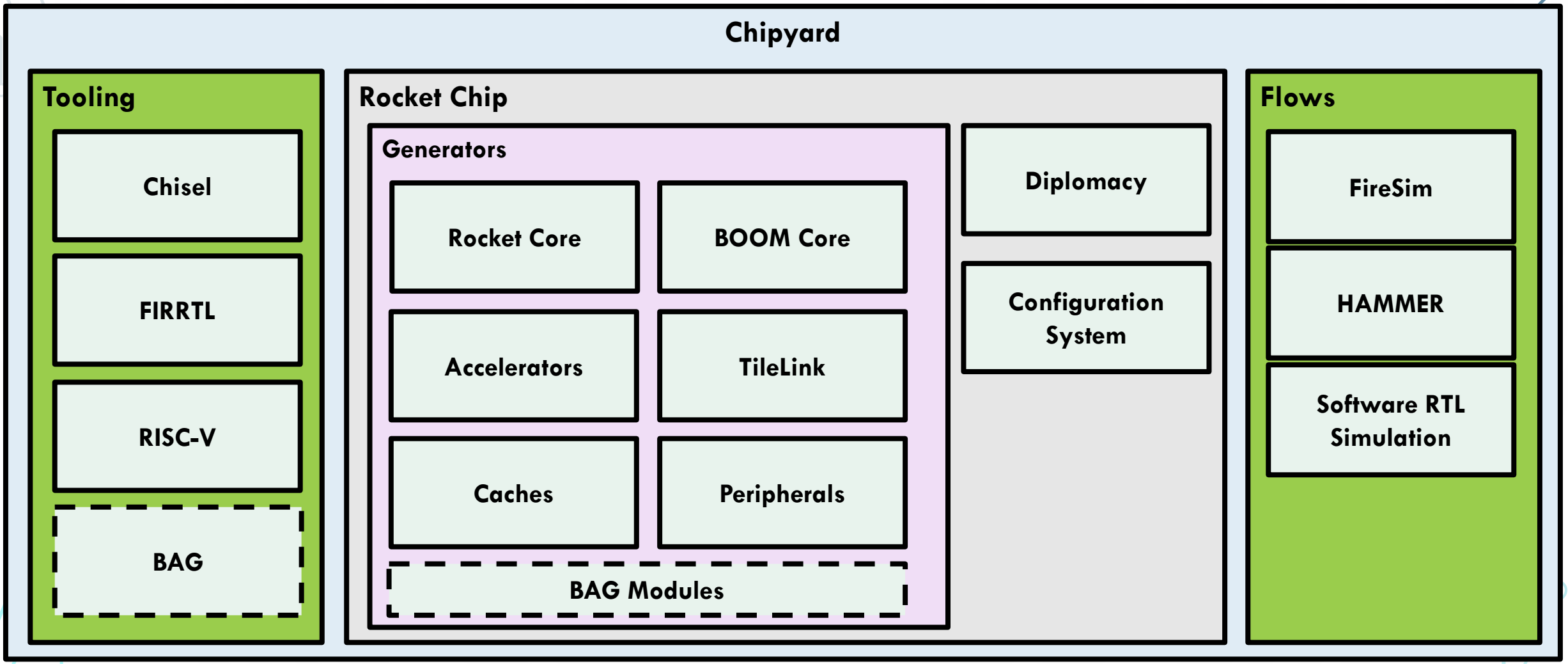
# There is a Lot of Open-Source Stuff!

- Many open source components:




**Goal:** Make it easy for small teams to
**design, integrate, simulate,**
**validate workload performance** and **tape-out** a custom SoC

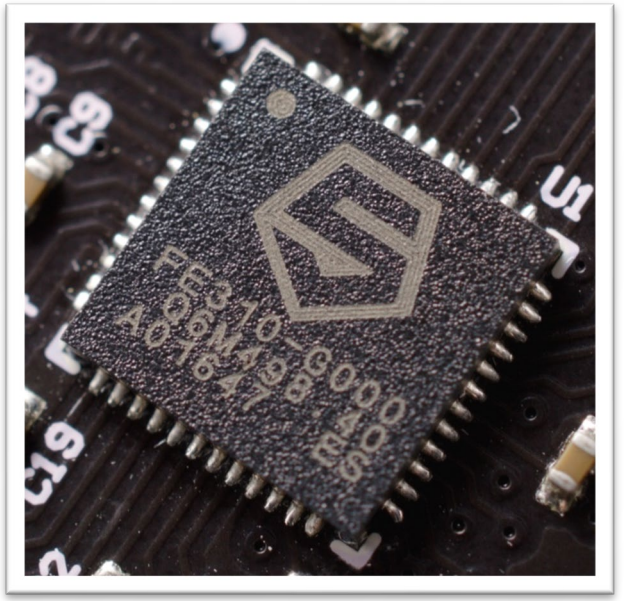
# Chipyard


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

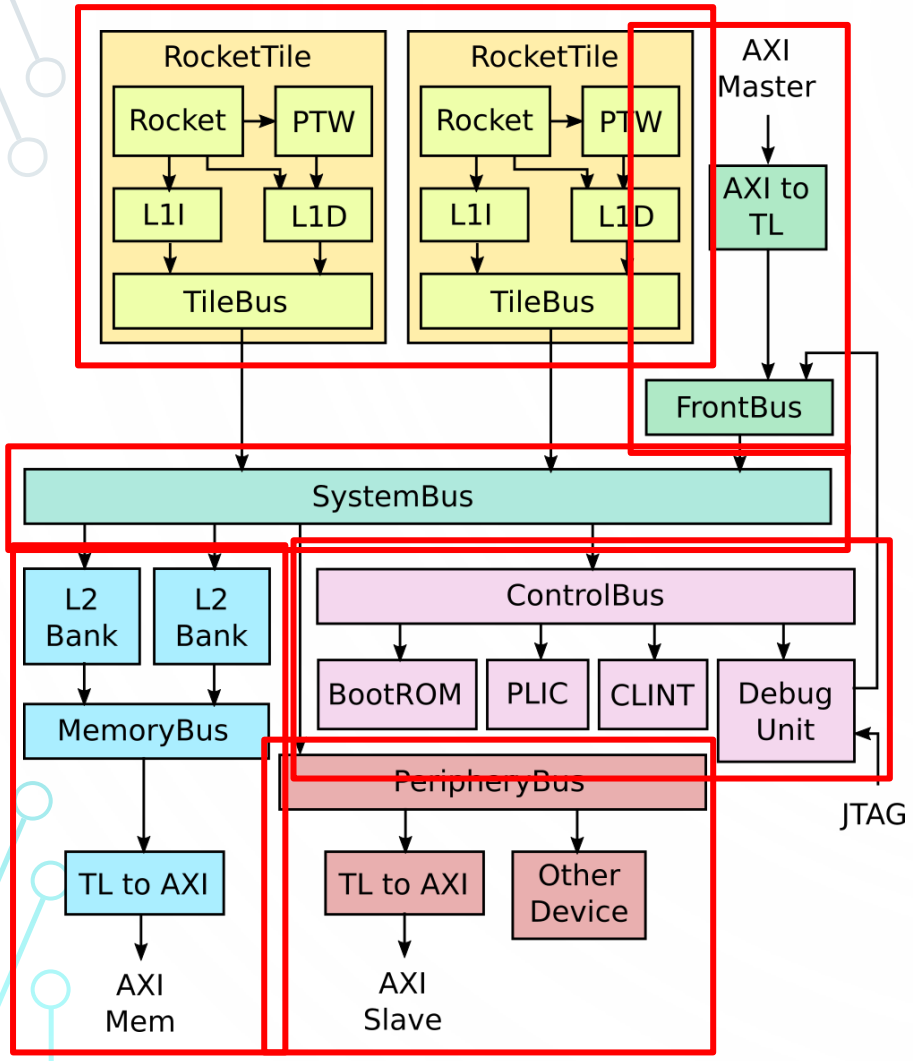
# What is Rocket Chip?

- A highly parameterizable 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
  - Scala allows advanced generator features
- Developed at Berkeley, now maintained by many
  - SiFive, CHIPS Alliance, UC Berkeley

In industry: **SiFive Freedom E310**



# Structure of a Rocket Chip SoC




**Tiles:** unit of replication for a core

- CPU (Rocket, BOOM, Ariane)
- L1 Caches
- Page-table walker

**L2 banks:**

- Receive memory requests

**FrontBus:**

- Connects to DMA devices

**ControlBus:**

- Connects to core-complex devices

**PeripheryBus:**

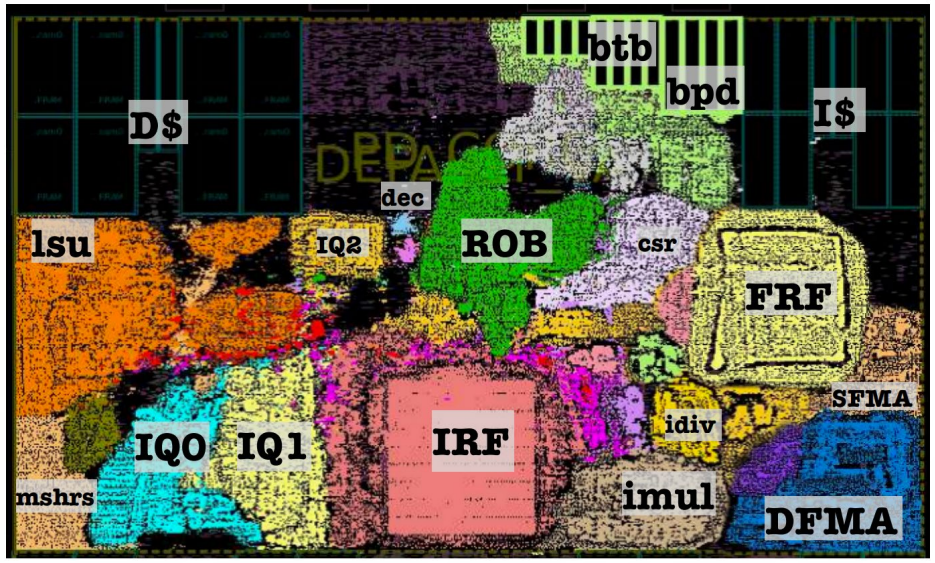
- Connects to other devices

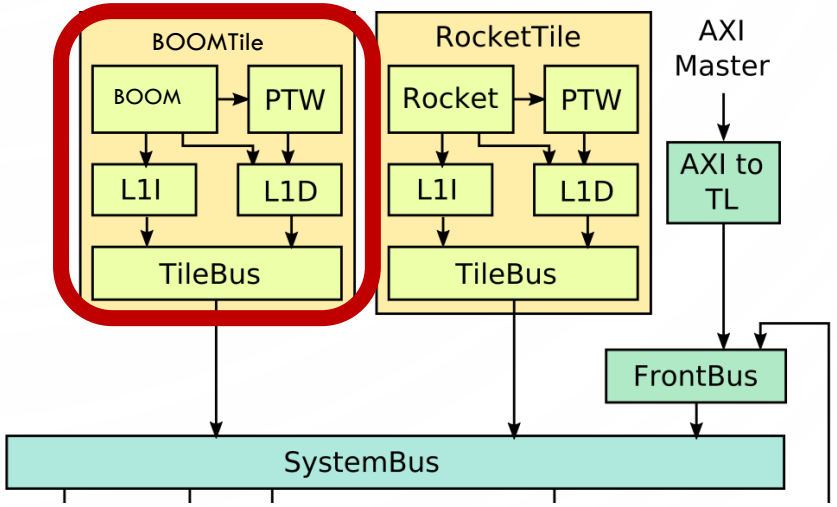
**SystemBus:**

- Ties everything together

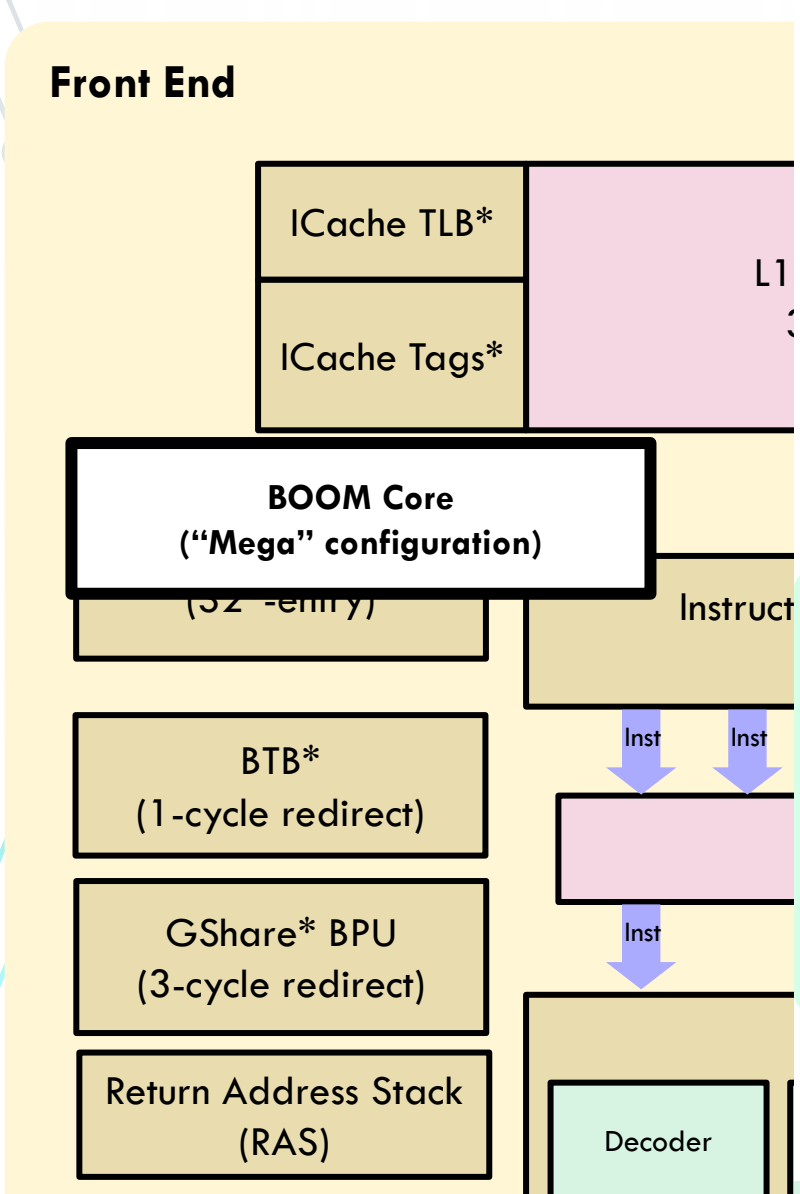
# BOOM: The Berkeley Out-of-Order Machine

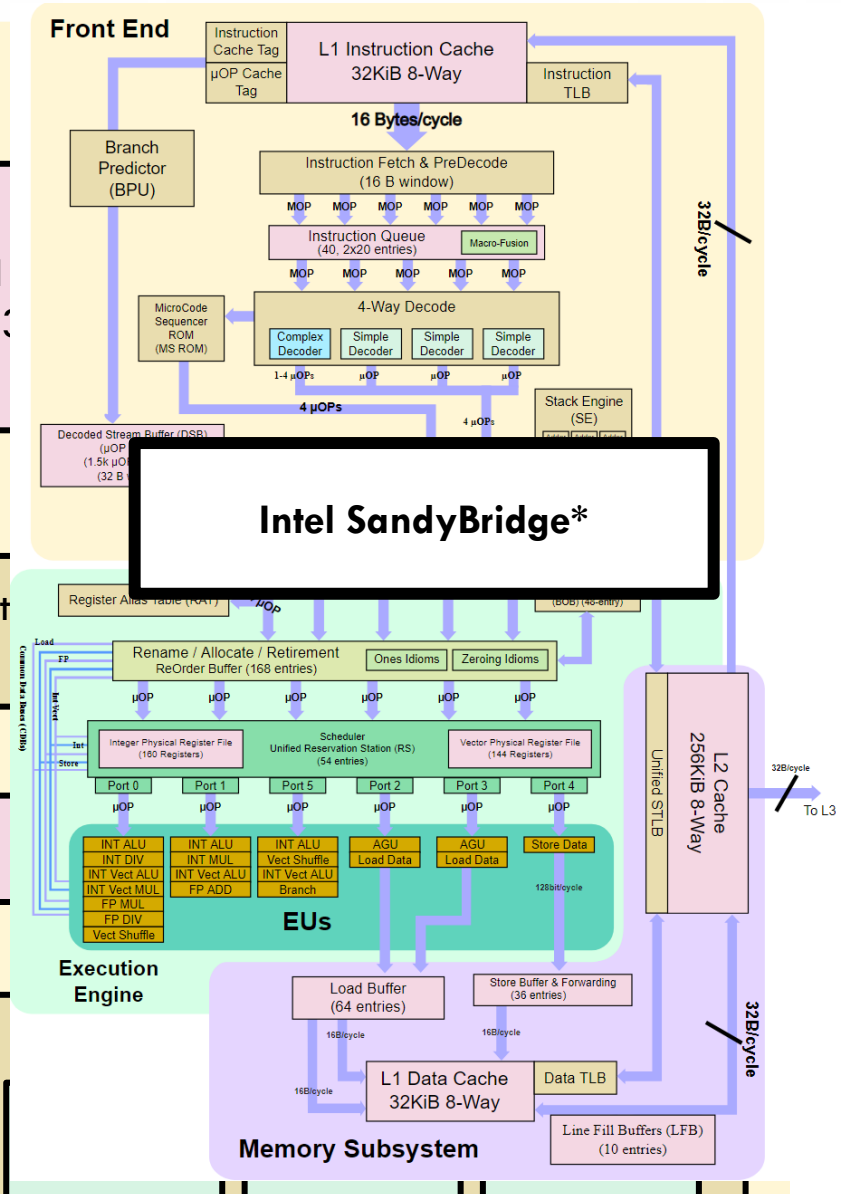
- Superscalar RISC-V OoO core
- Fully integrated in Rocket Chip ecosystem
- Open-source
- Described in Chisel
- Parameterizable generator
- Taped-out (BROOM; VLSI'18)
  - Updated: https://boom-core.org/
- Full RV64GC ISA support
  - FP, RVC, Atomics, PMPs, VM, Breakpoints, RoCC
  - Runs real OS's, software
- Drop-in replacement for Rocket

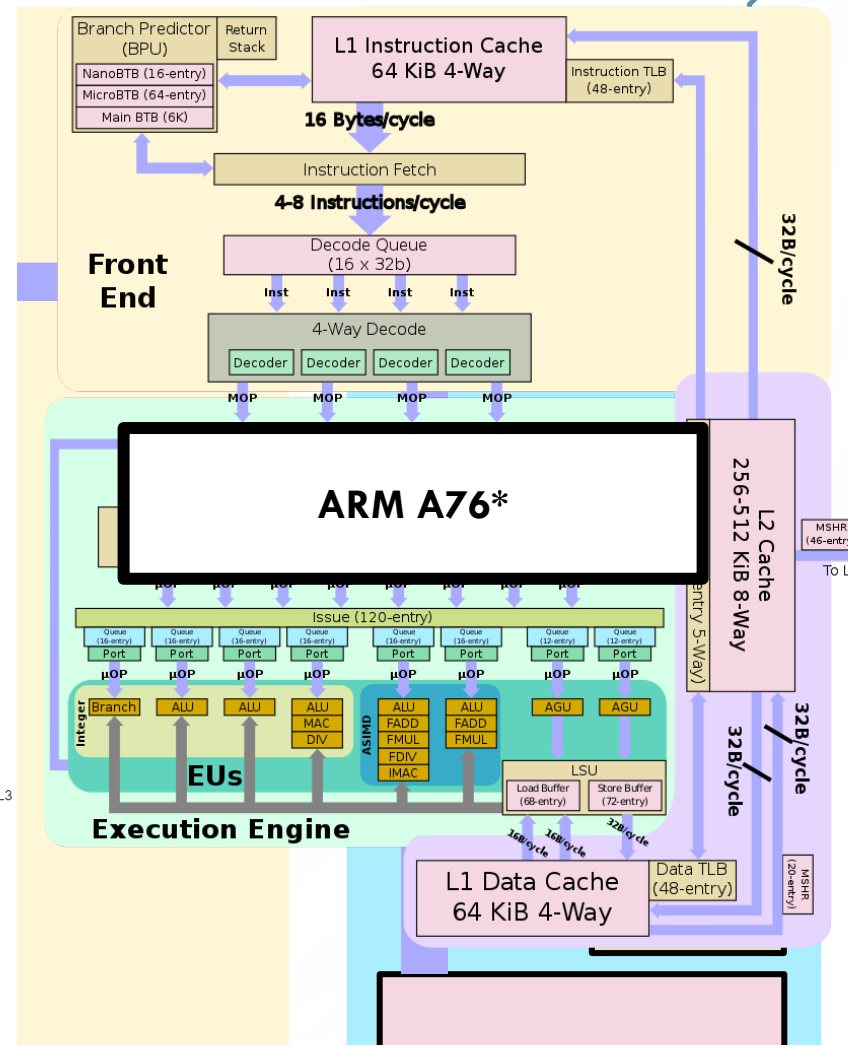


# BOOM Microarchitecture



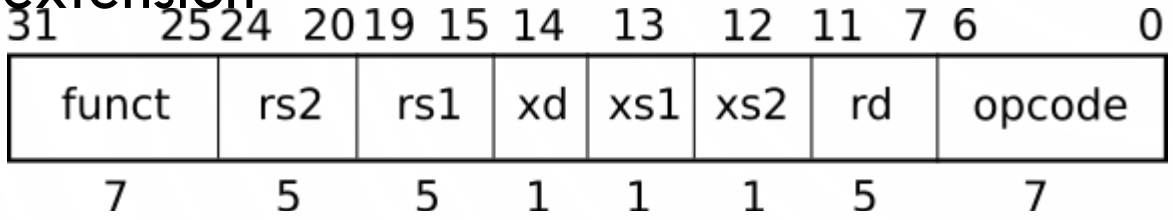



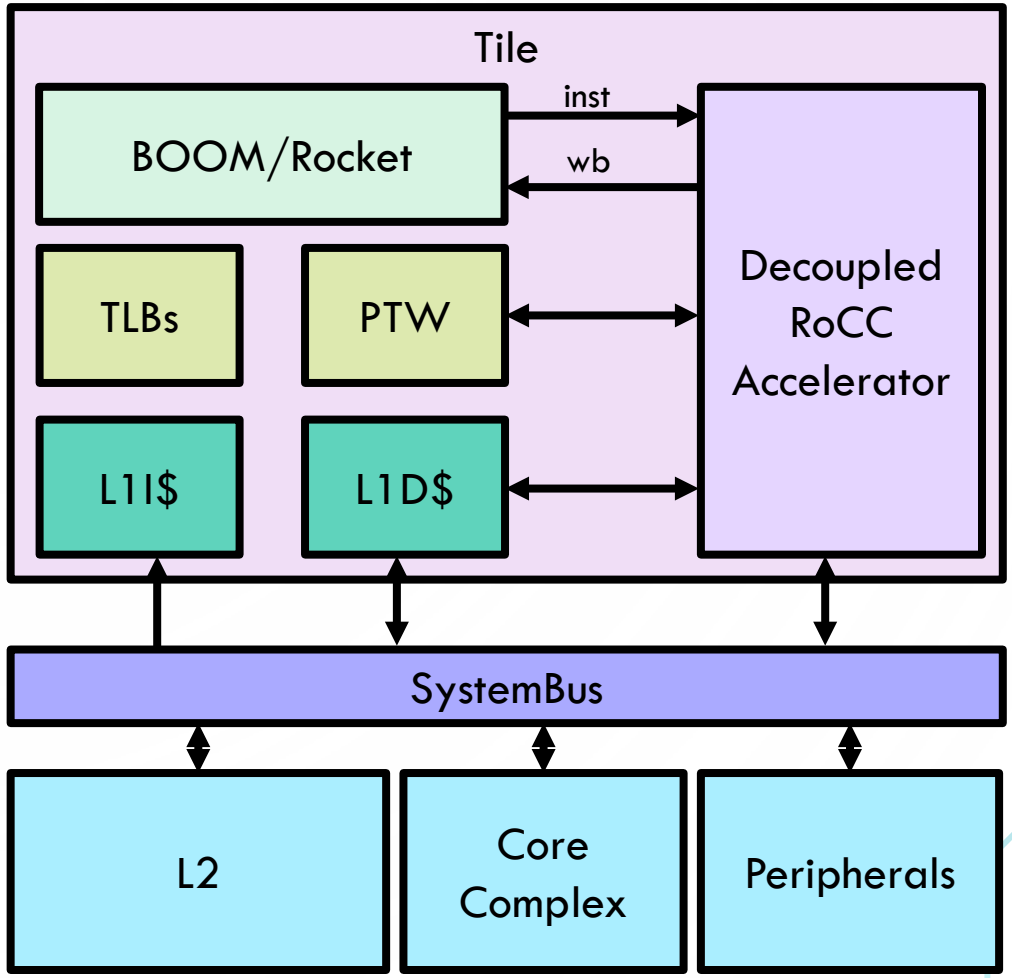

*Block diagram from WikiChip

# RoCC Accelerators

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

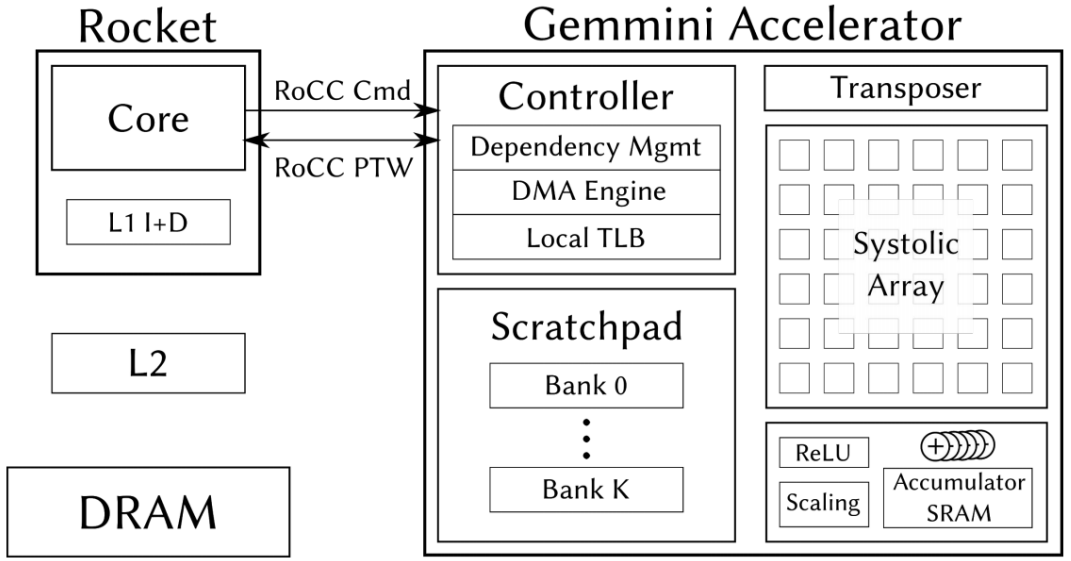



- Examples of RoCC accelerators
  - Vector accelerators
  - Memcpy accelerator
  - Machine-learning accelerators (Gemmini, NVDLA)
    - See the second part of the tutorial!
  - Java GC accelerator



# Gemmini: A Systolic(ish) Generator

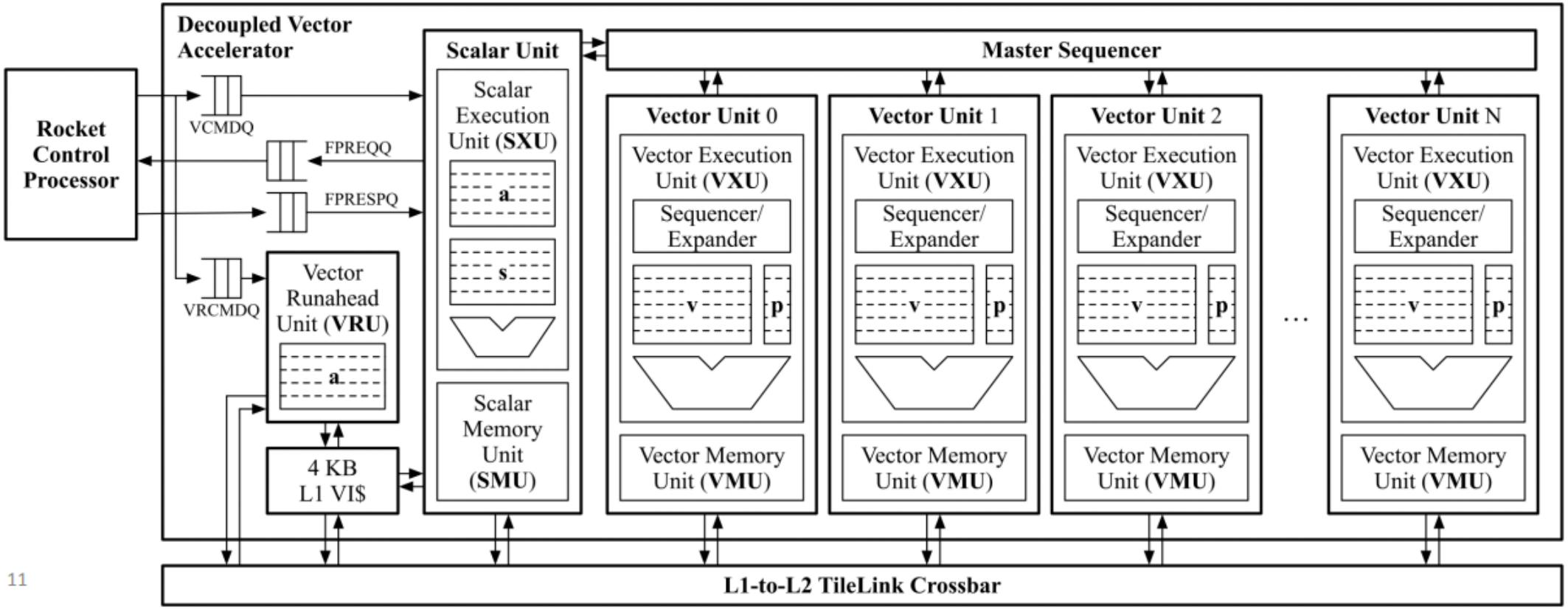
- Systolic Array Accelerator
- Fully configurable
  - Dataflow – Output/Weight Stationary
  - Dimensions
  - Bitwidths/Datatypes
  - Pipeline Depth
  - Memory capacity
  - Memory banking
  - Memory Bus Width




https://www.github.com/ucb-bar/gemmini

# Hwacha: A Vector Accelerator

- Configurable access/execute decoupled vector architecture*

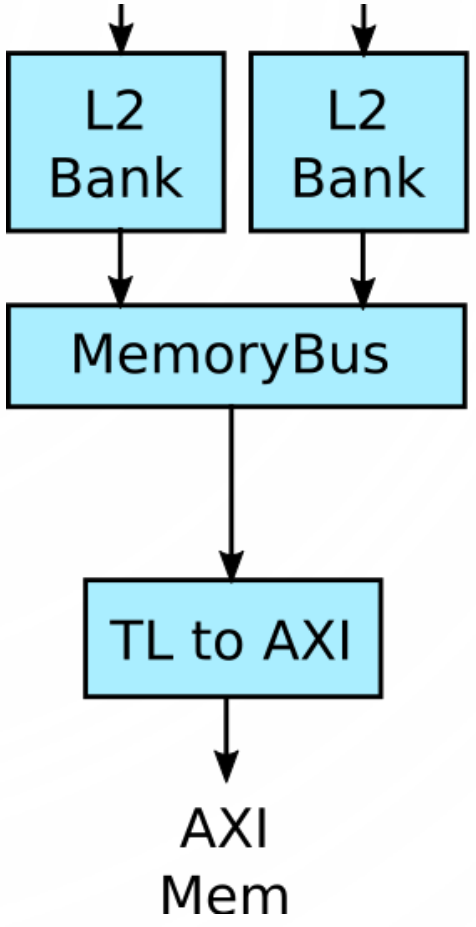





* Not based on RV-V

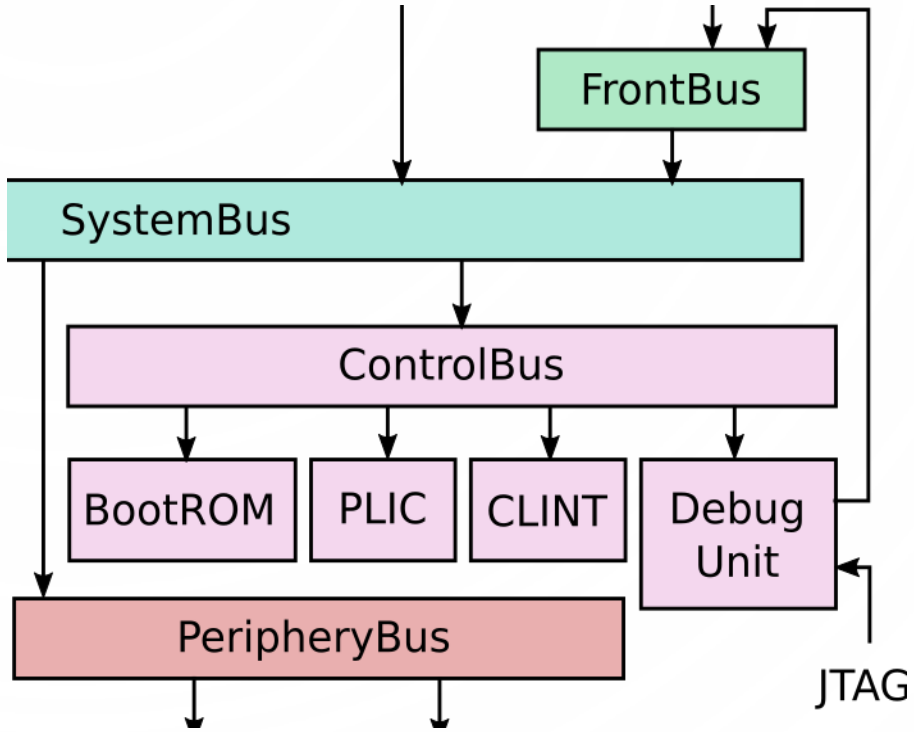
# L2 Cache and Memory System

- 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

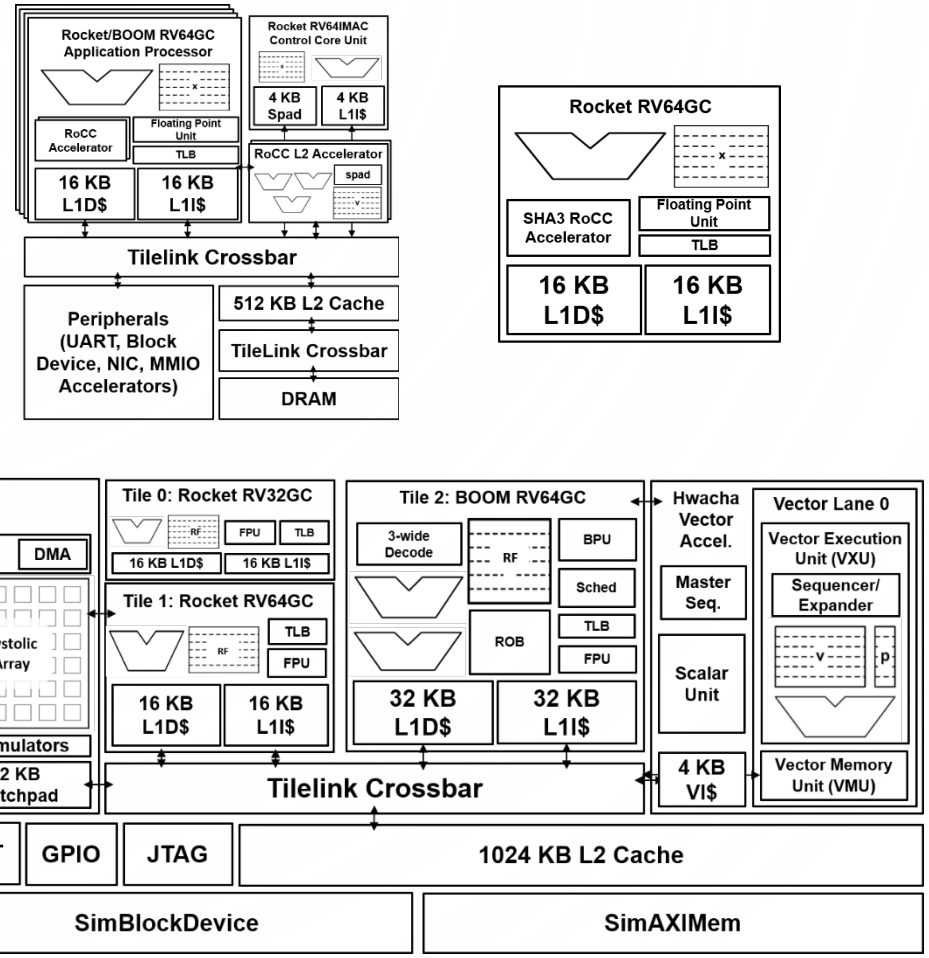


# Other Chipyard Blocks

- **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
- **SHA3:** Educational SHA3 RoCC accelerator

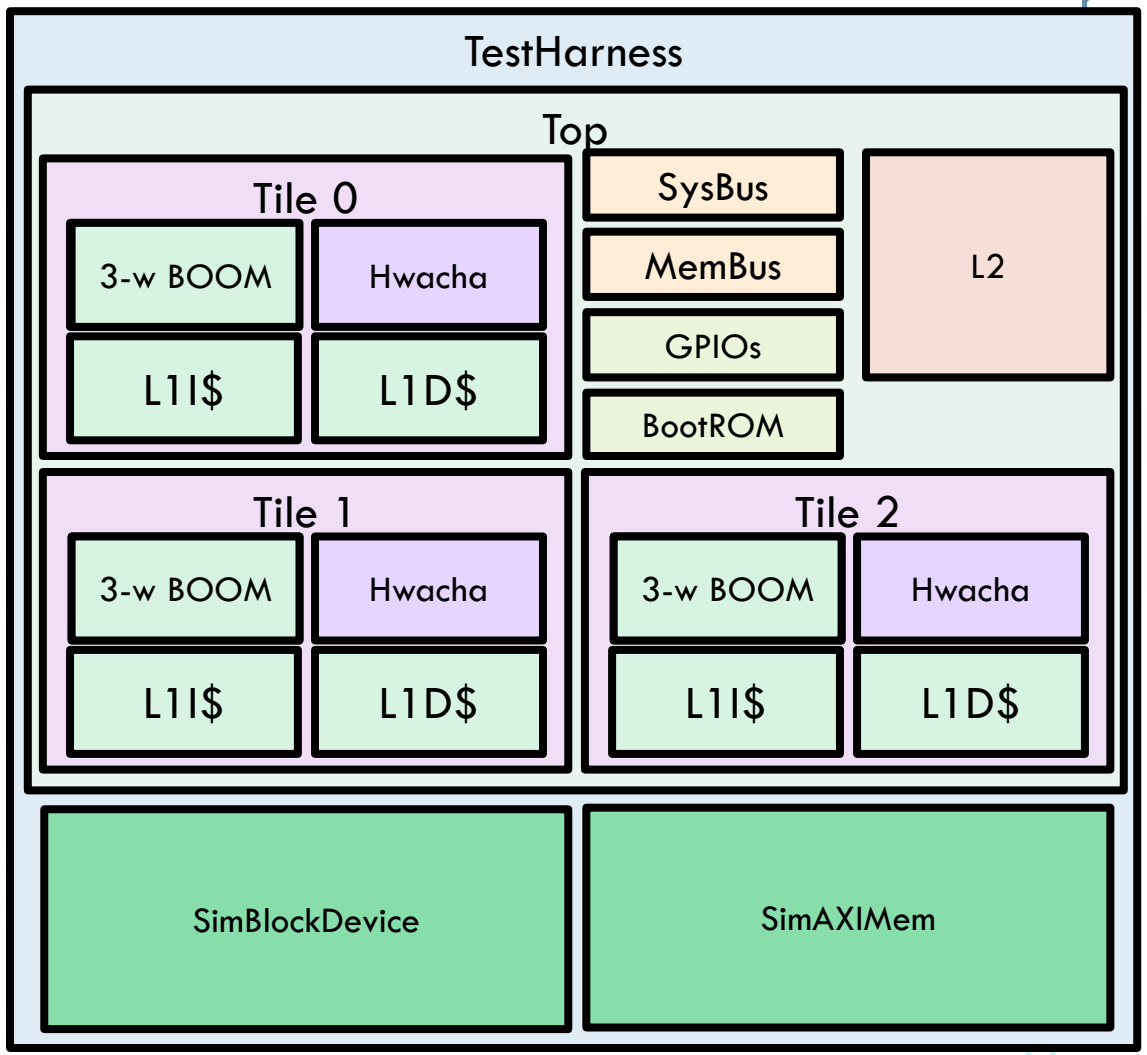
# Customization

- Cores and controllers: Intra-core Rocket/BOOM configurations
  - Control core / PMU as an example
- Simple RoCC accelerators
  - SHA3 as an 'instructional' demo
- Complex RoCC accelerators
  - Hwacha and Gemmini as examples
- MMIO Tilelink accelerators
- Peripherals



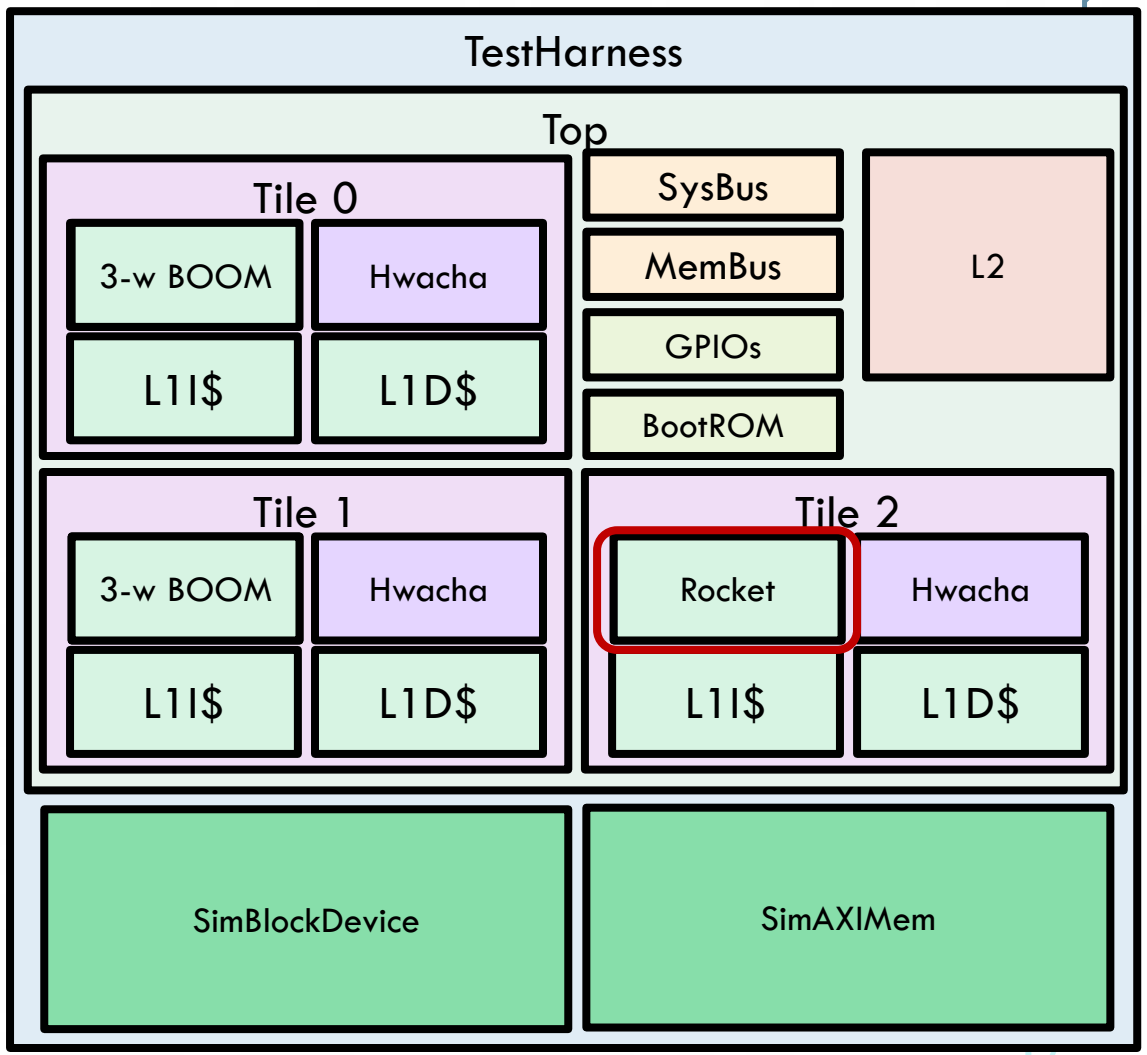
# Rocket Chip Configuration

```
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)
```



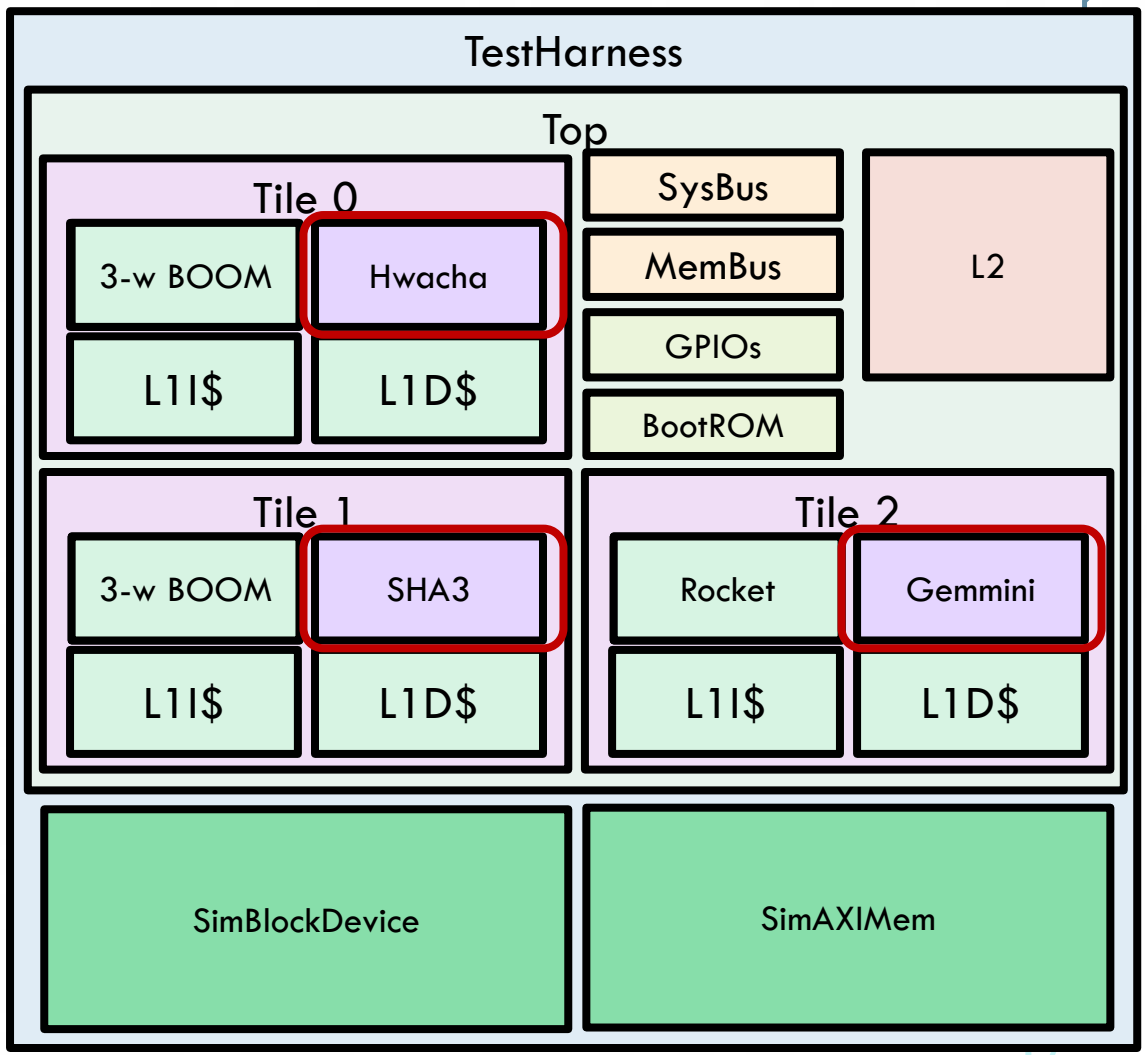
# Rocket Chip Configuration

```
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)
```



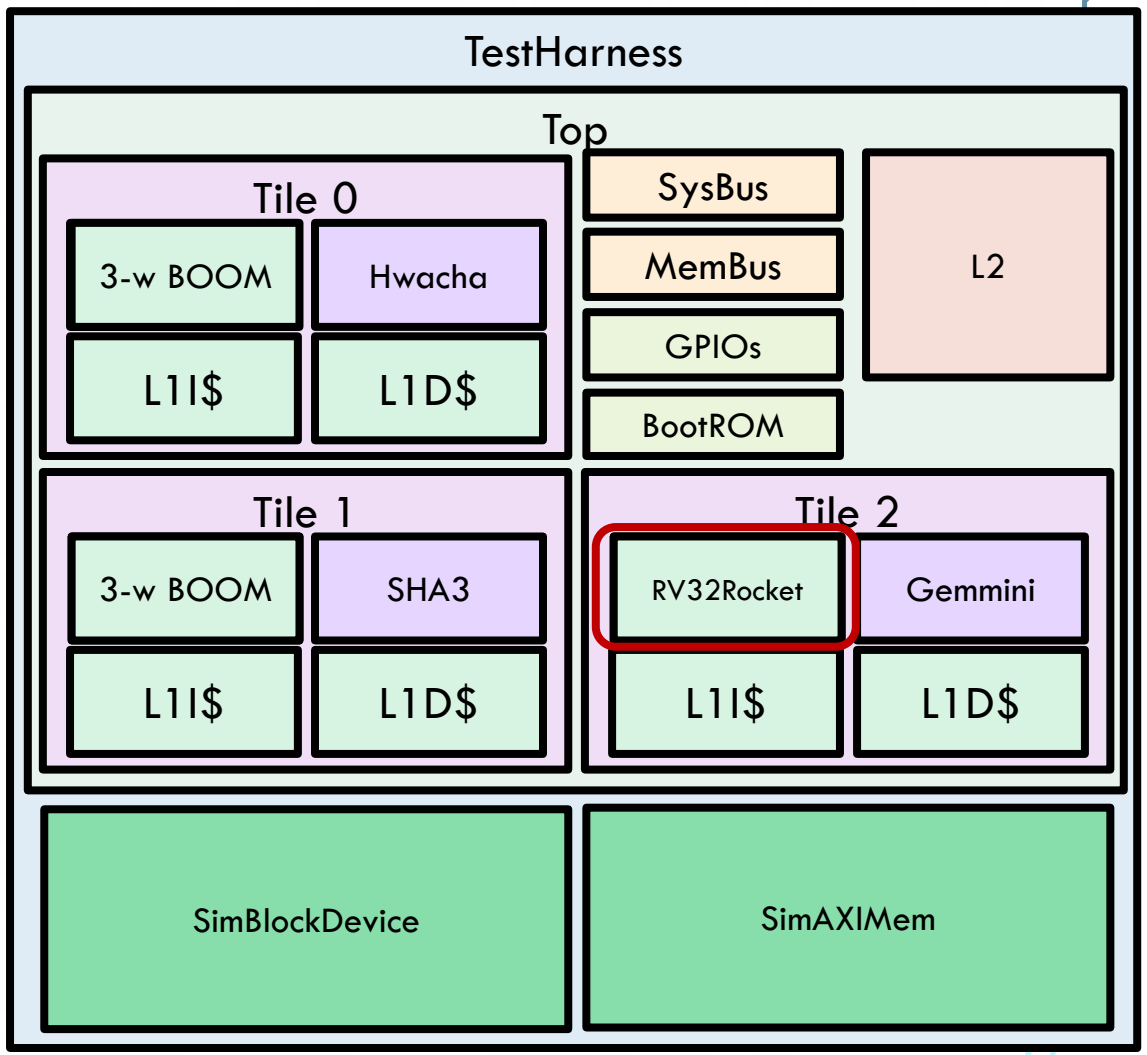
# Rocket Chip Configuration

```
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)
```



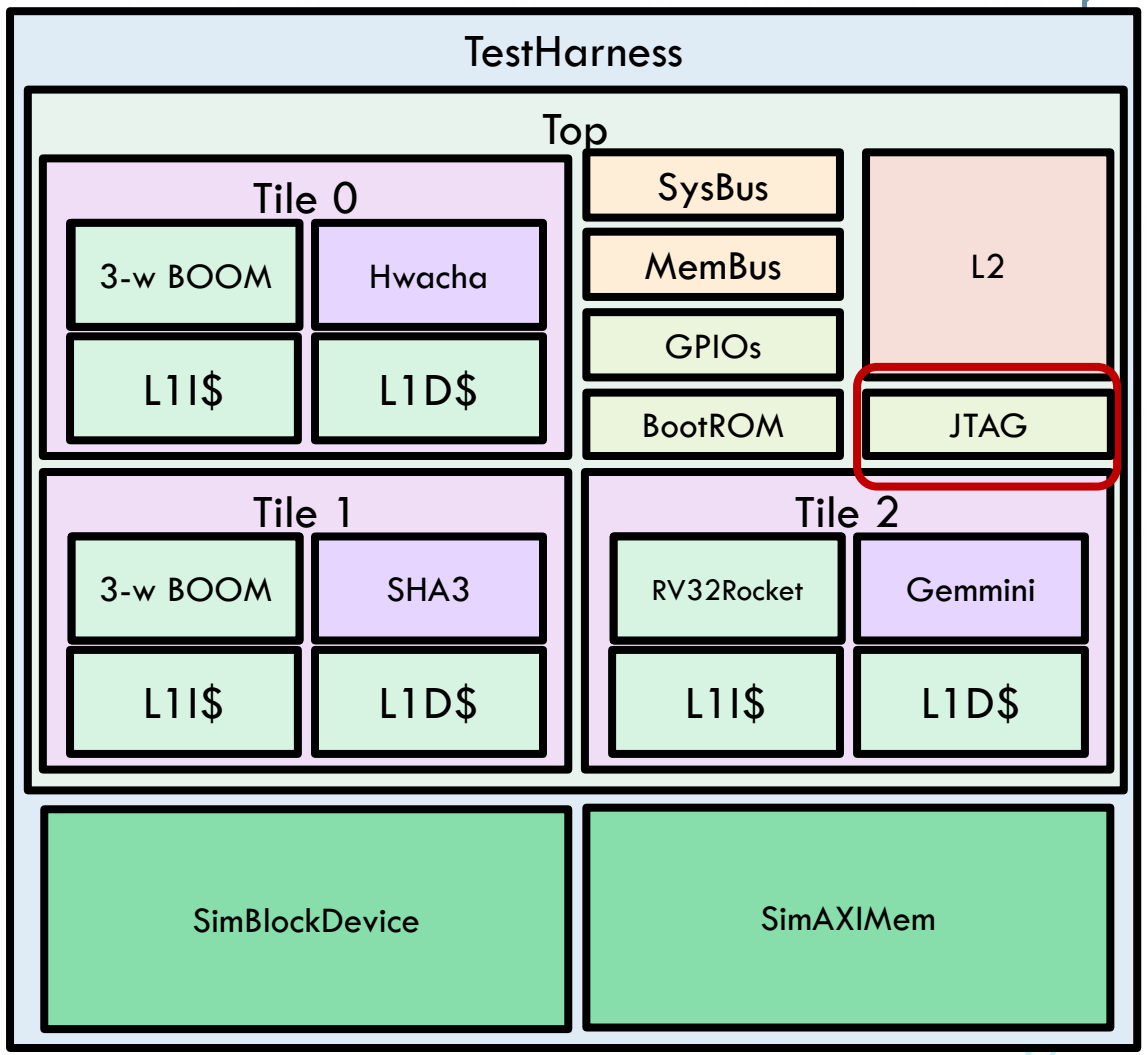
# Rocket Chip Configuration

```
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)
```



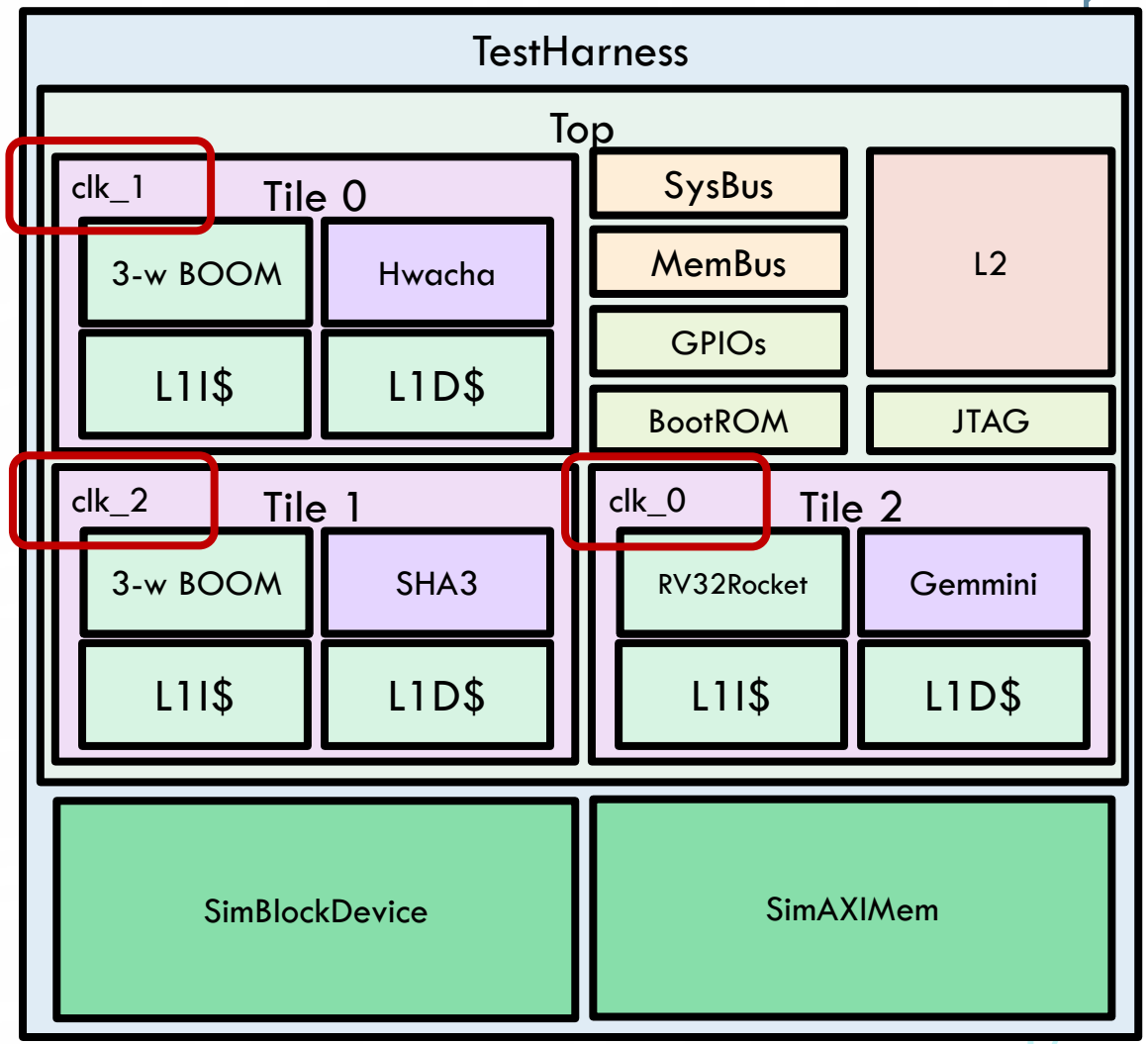
# Rocket Chip Configuration

```
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)
```



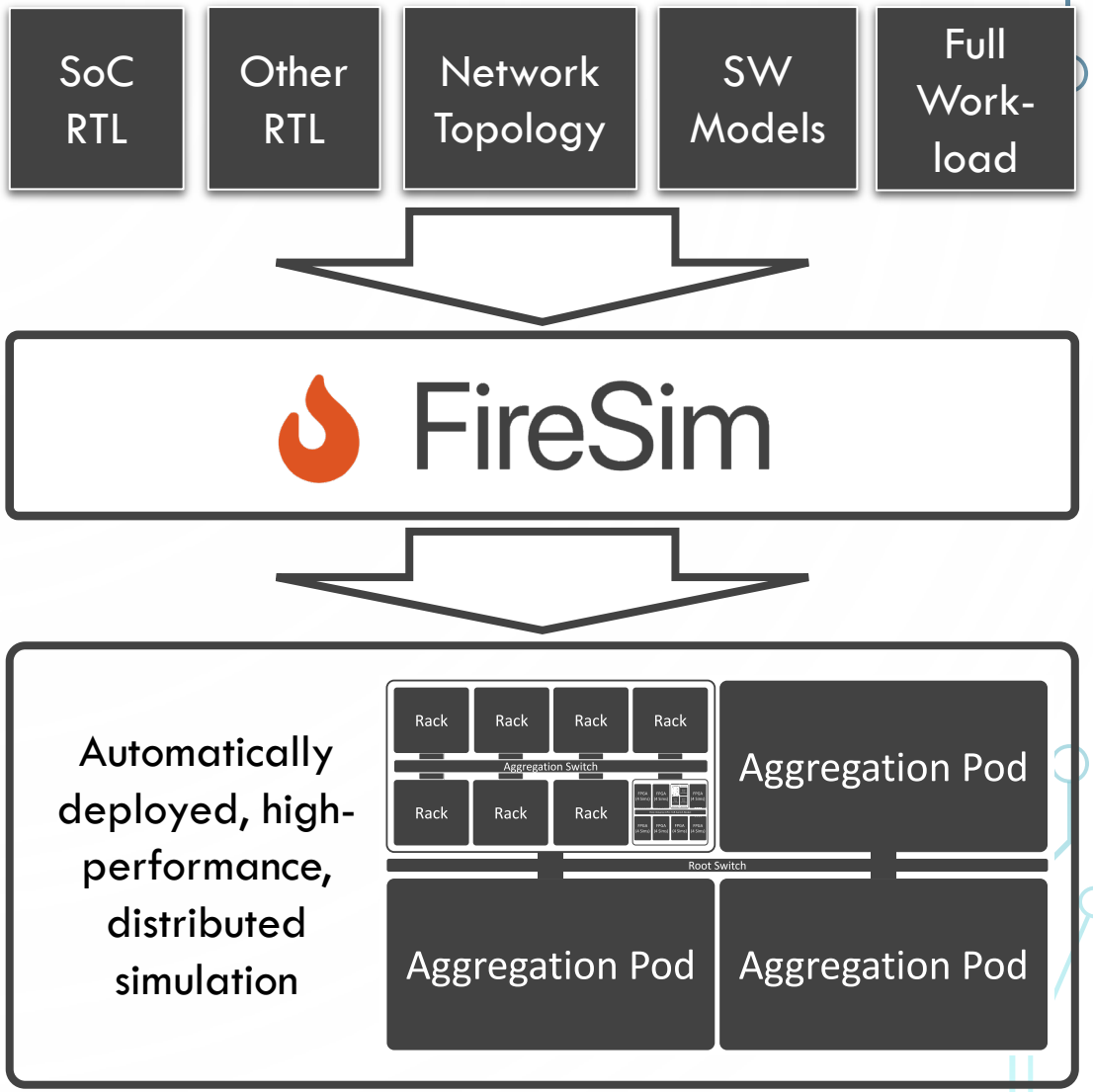
# Rocket Chip Configuration

```
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)
```



# 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

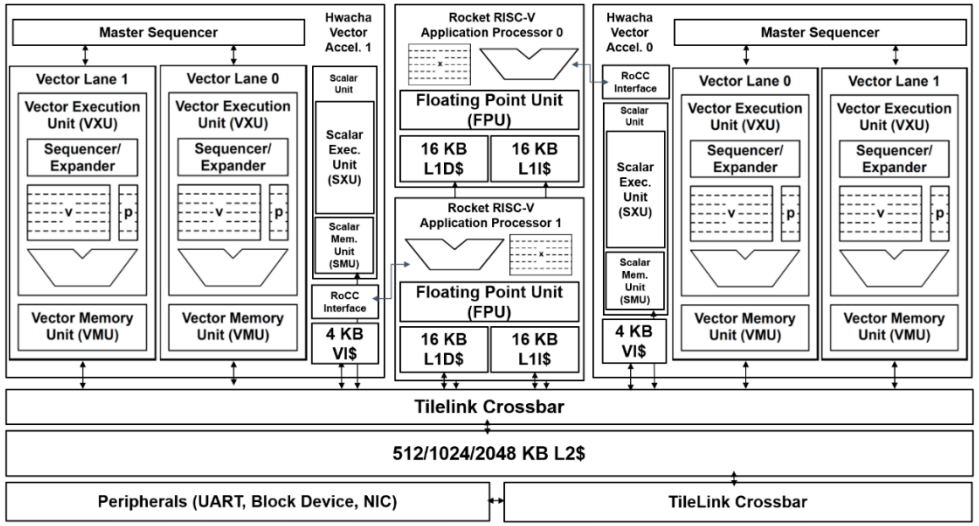


# FireSim

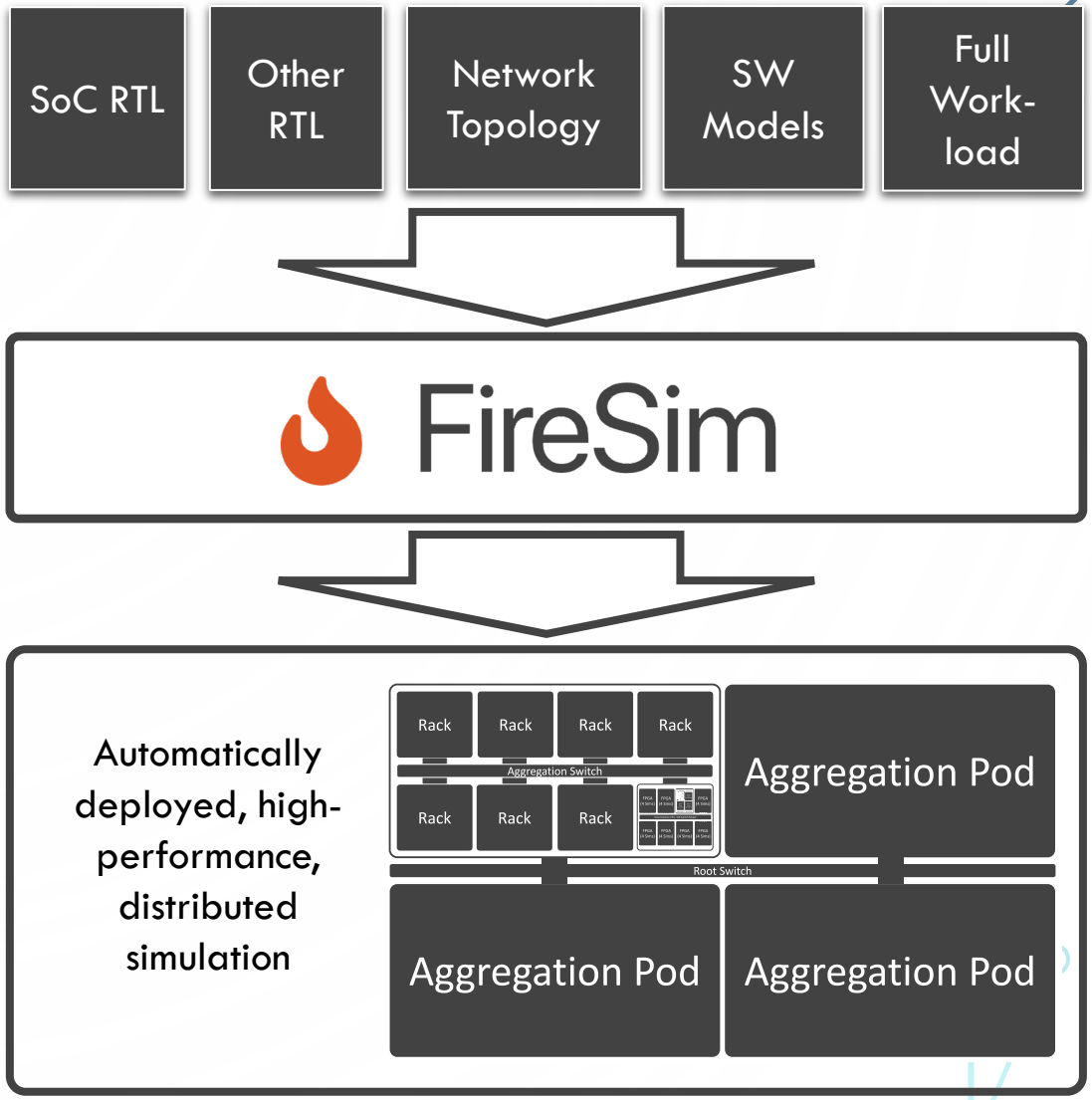
- Cycle-exactly simulating large SoCs on cloud FPGAs @10s-100s of MHz
- Open-source: https://fires.im
- Targets:
  (1) Architecture evaluation
  (2) Validate application on a pre-Si SoC

S. Karandikar, ISCA '18, IEEE Micro TopPicks '18, CARRV '19

Example of (2): PageRank on Rocket+Hwacha

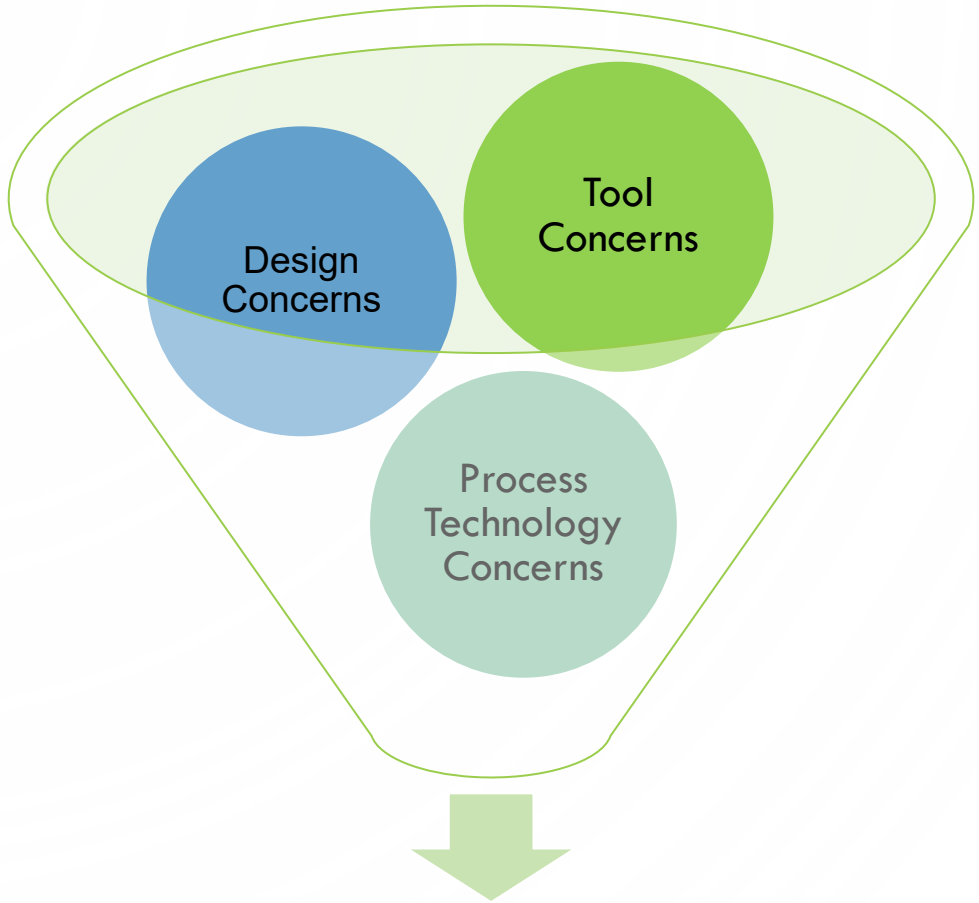



A. Amid, CARRV'19



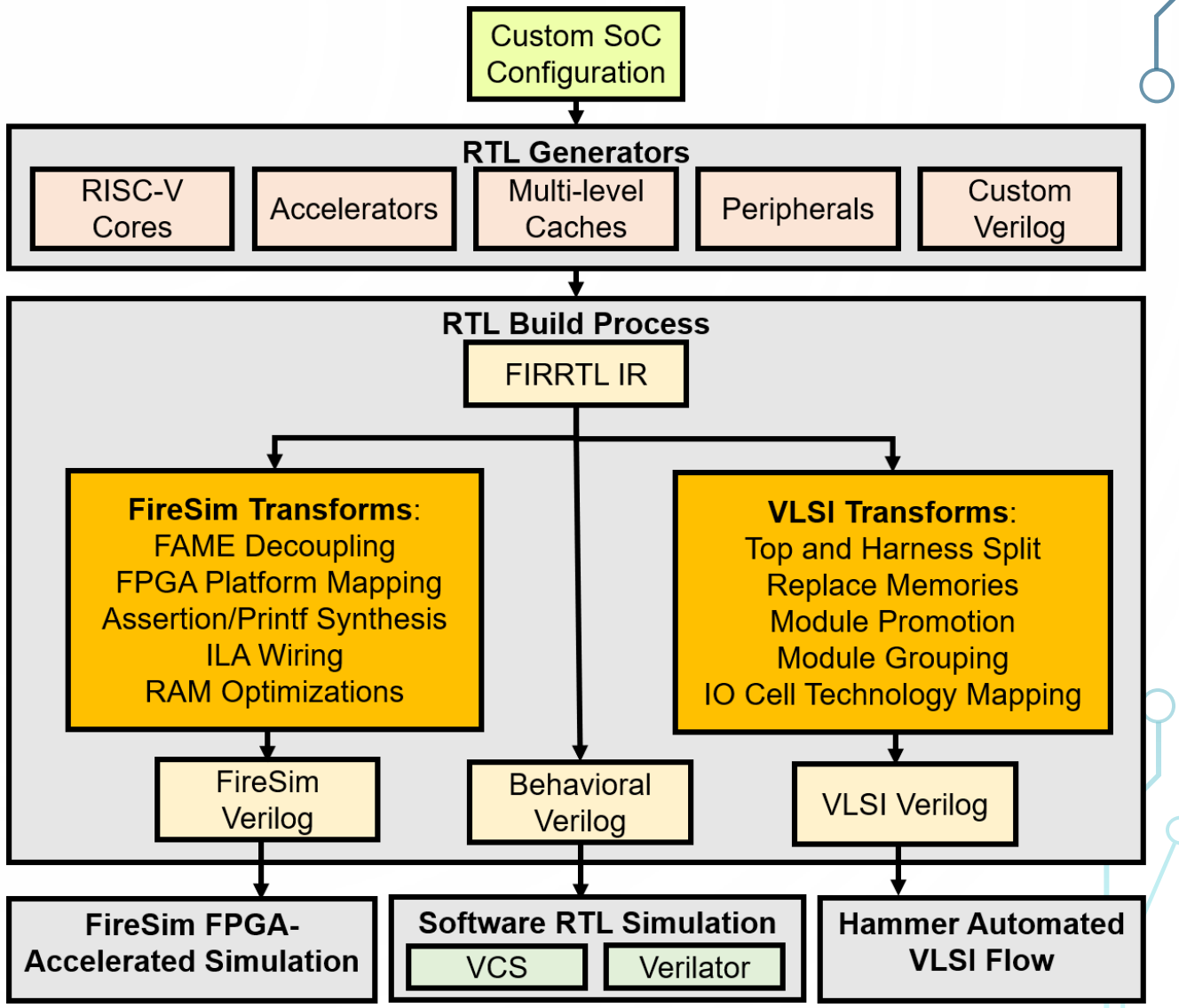
# Hammer

- 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



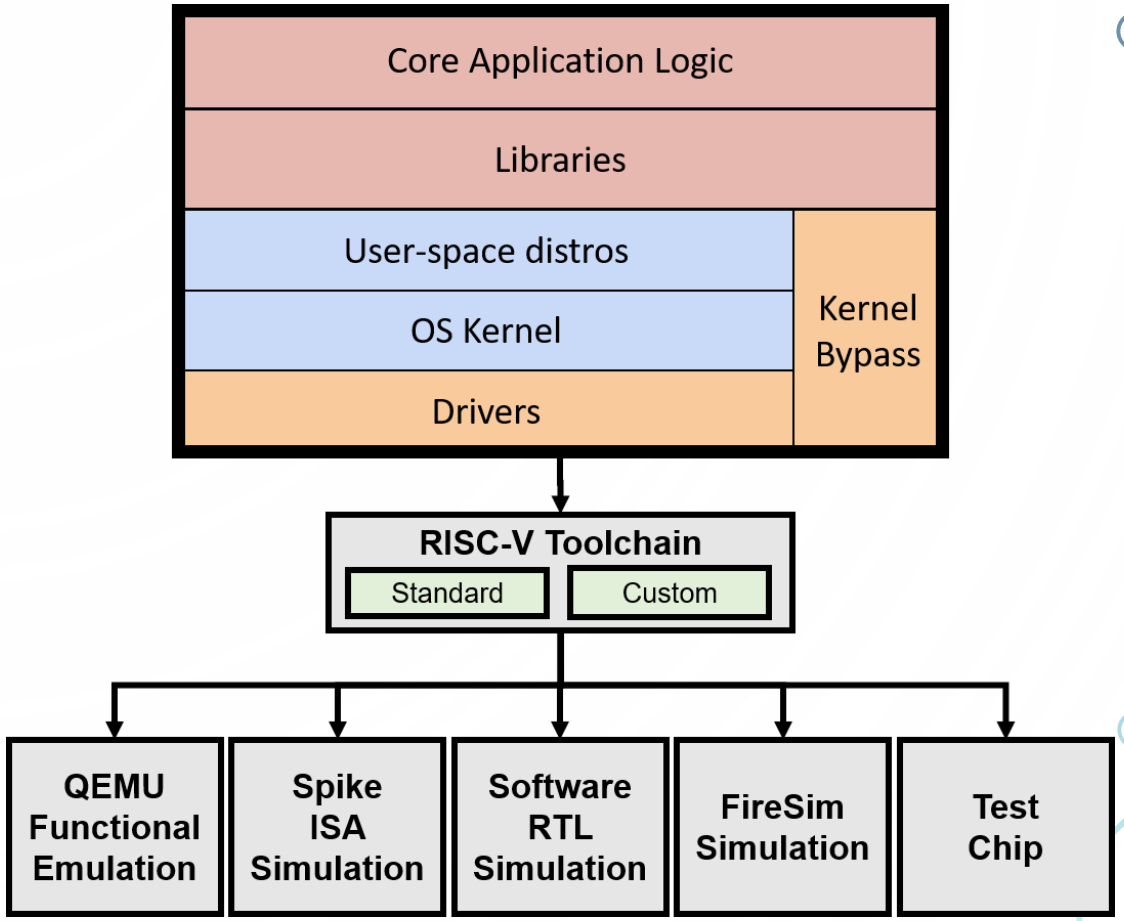
# Simulation/Implementation Targets

- Custom hardware design is not just about generated IP blocks!
- Different collaterals for different simulation or implementation targets
  - Design cycle RTL simulation
  - Verification / validation
  - VLSI flow



# 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



# Summary

- We will use Chipyard to generate a minimalist RISC-V SoC for logic design and circuits experiments
- Labs will exercise the design flow
- Think of labs that can:
  - Test a circuit idea in a larger system
  - Design a block to improve SoC
    - Co-processor/Accelerator
    - Peripheral device

# Next Lecture

- Features of modern technologies