Chipyard Basics

Howie Mao, Jerry Zhao

UC Berkeley

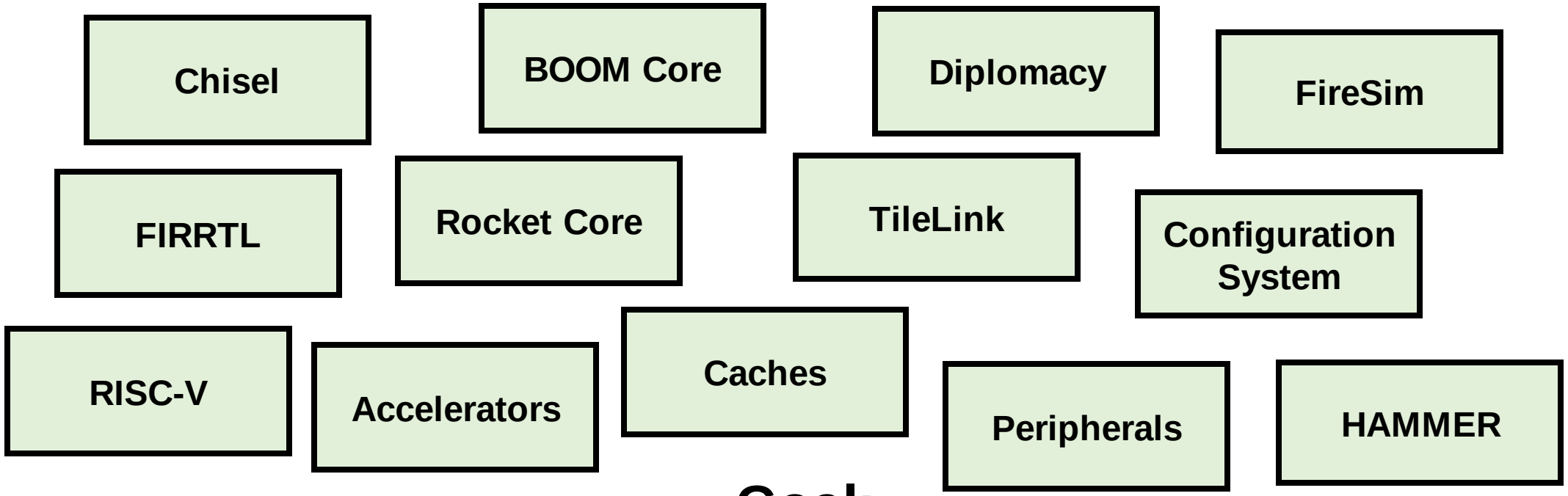
{zhemao,jzh}@berkeley.edu

Berkeley Architecture Research

CHIPYARD

Motivation

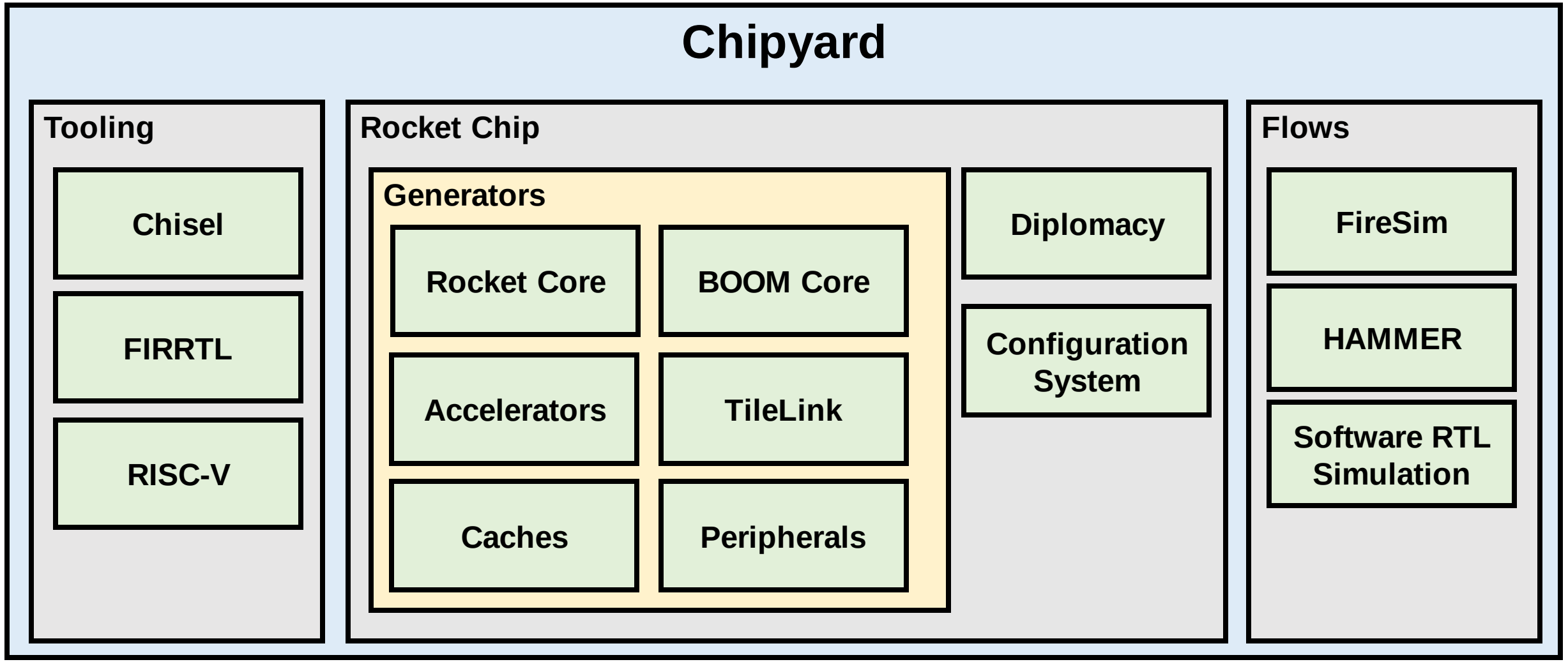
Berkeley Architecture Research has developed and open-sourced:



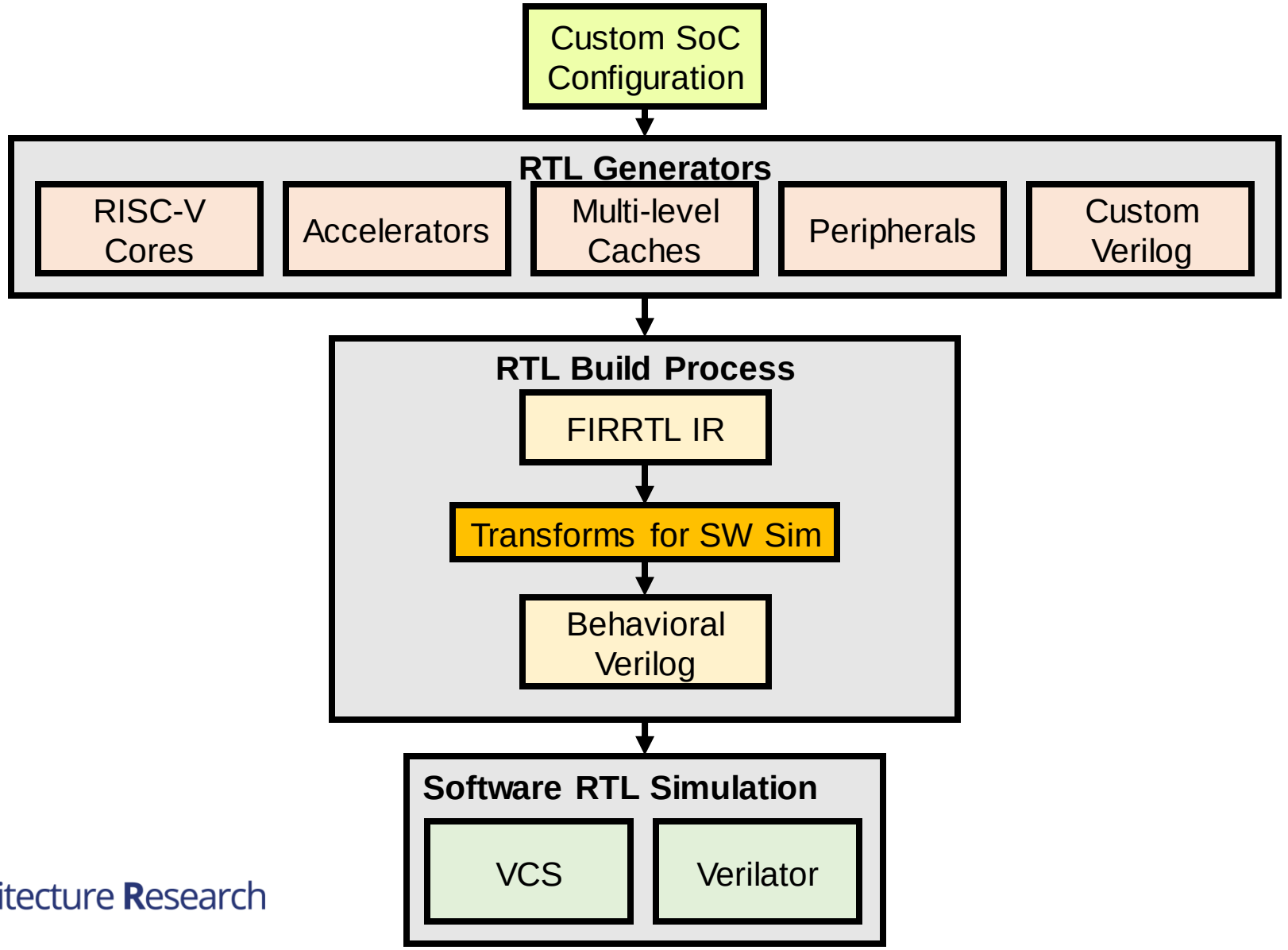
Goal:

Make it easy for small teams to
design, integrate, simulate, and **tape-out** a custom SoC

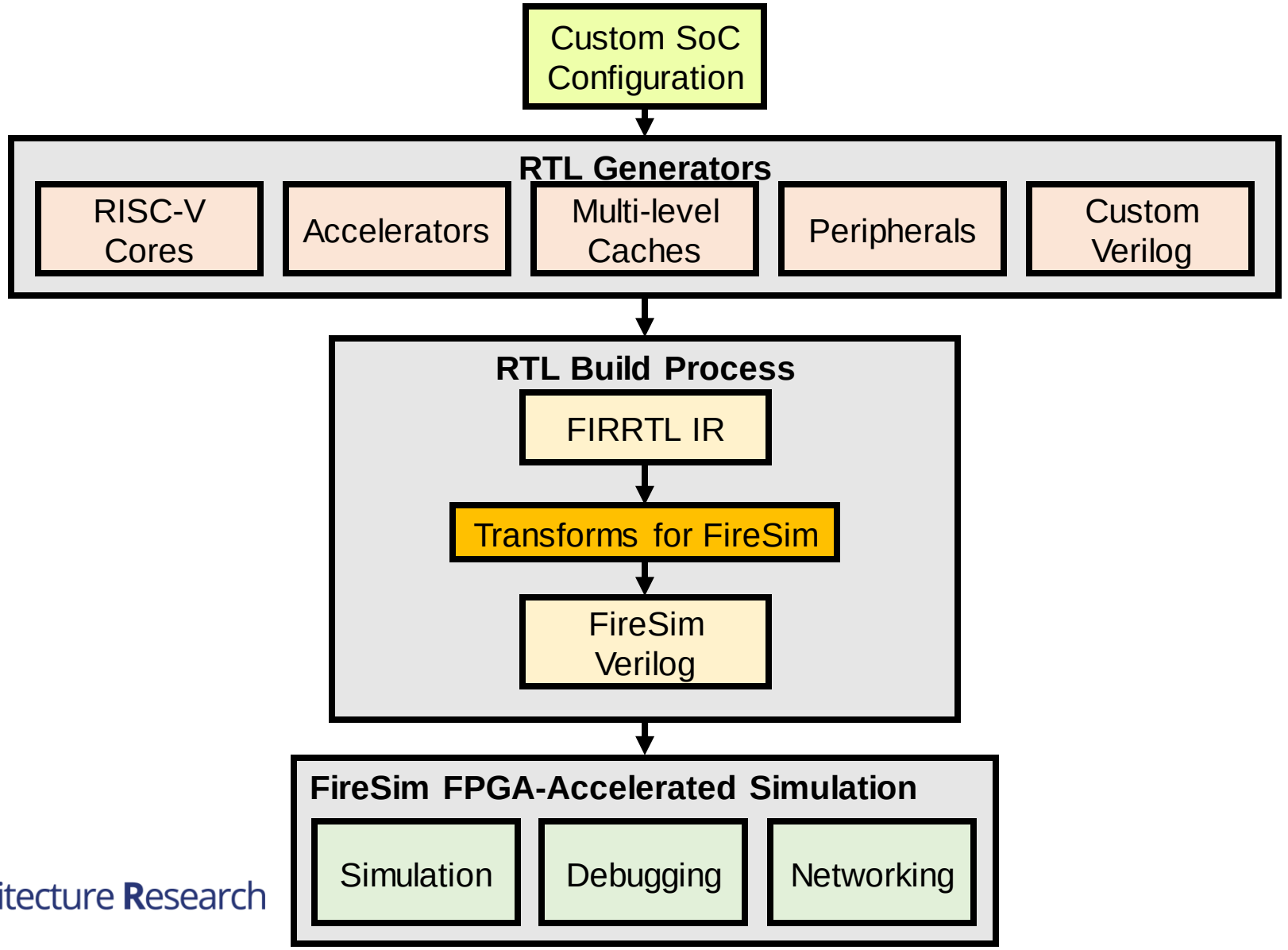
Chipyard


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

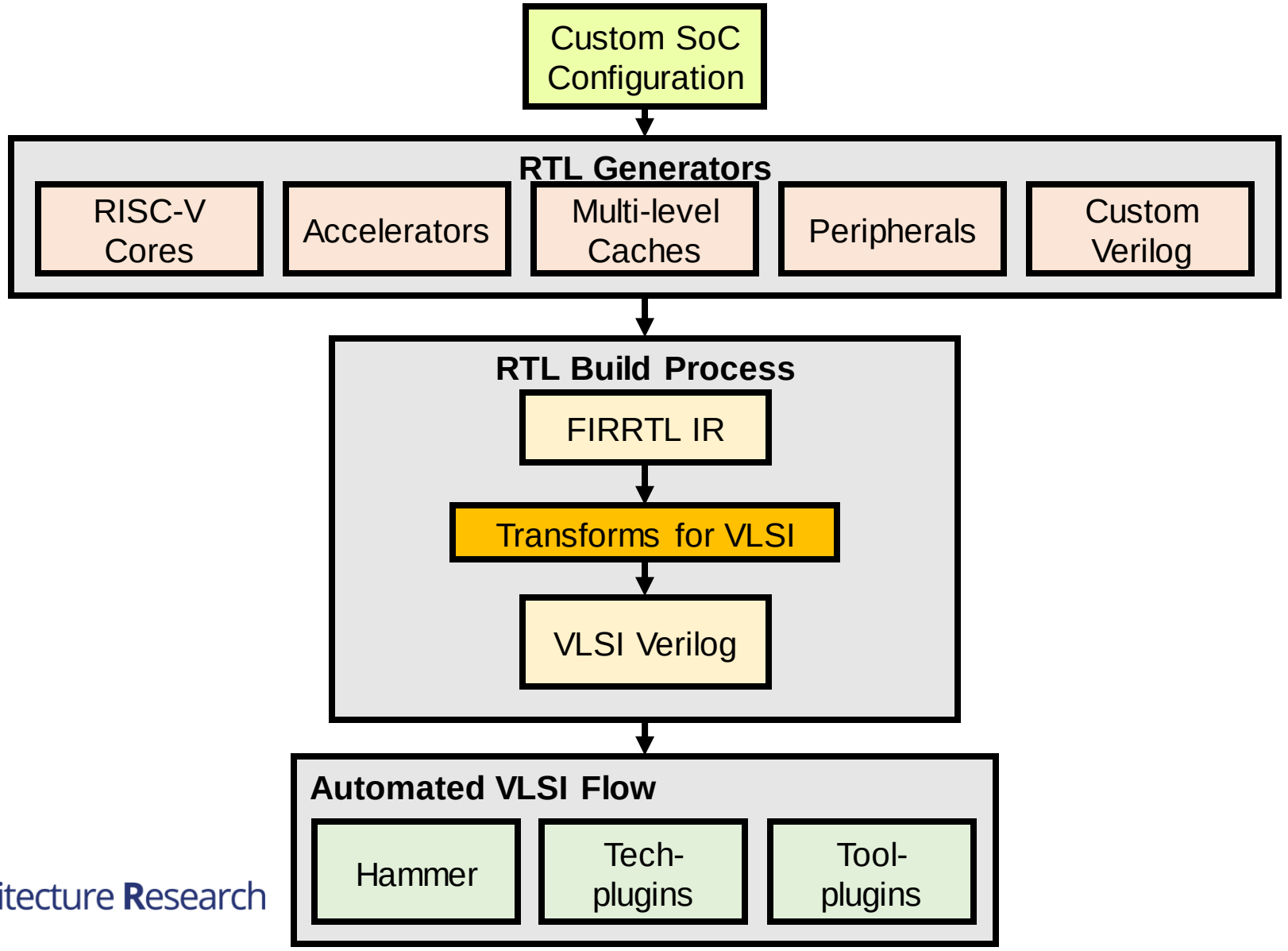
Chipyard SW RTL Simulation



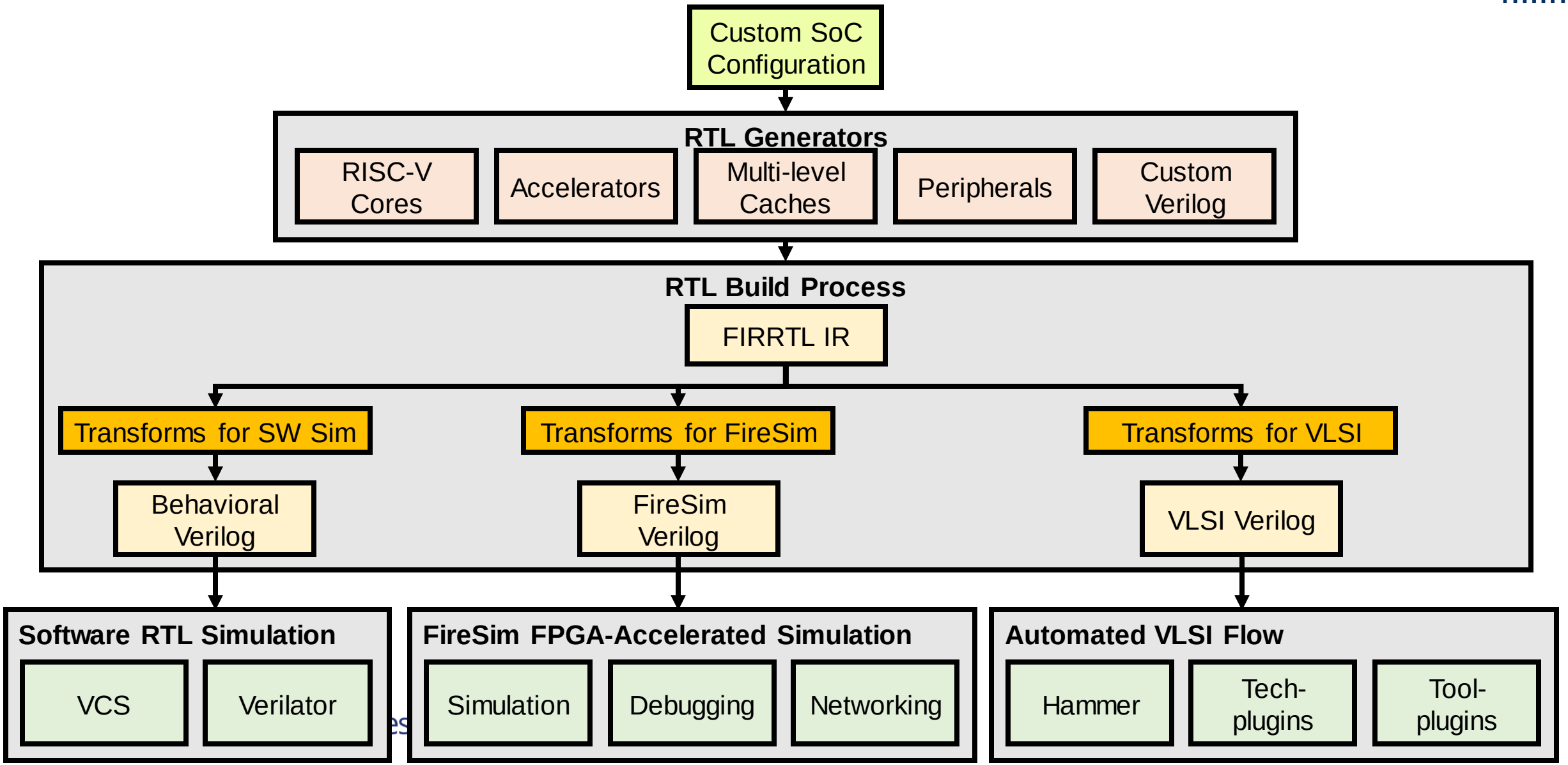
Chipyard targeting FireSim



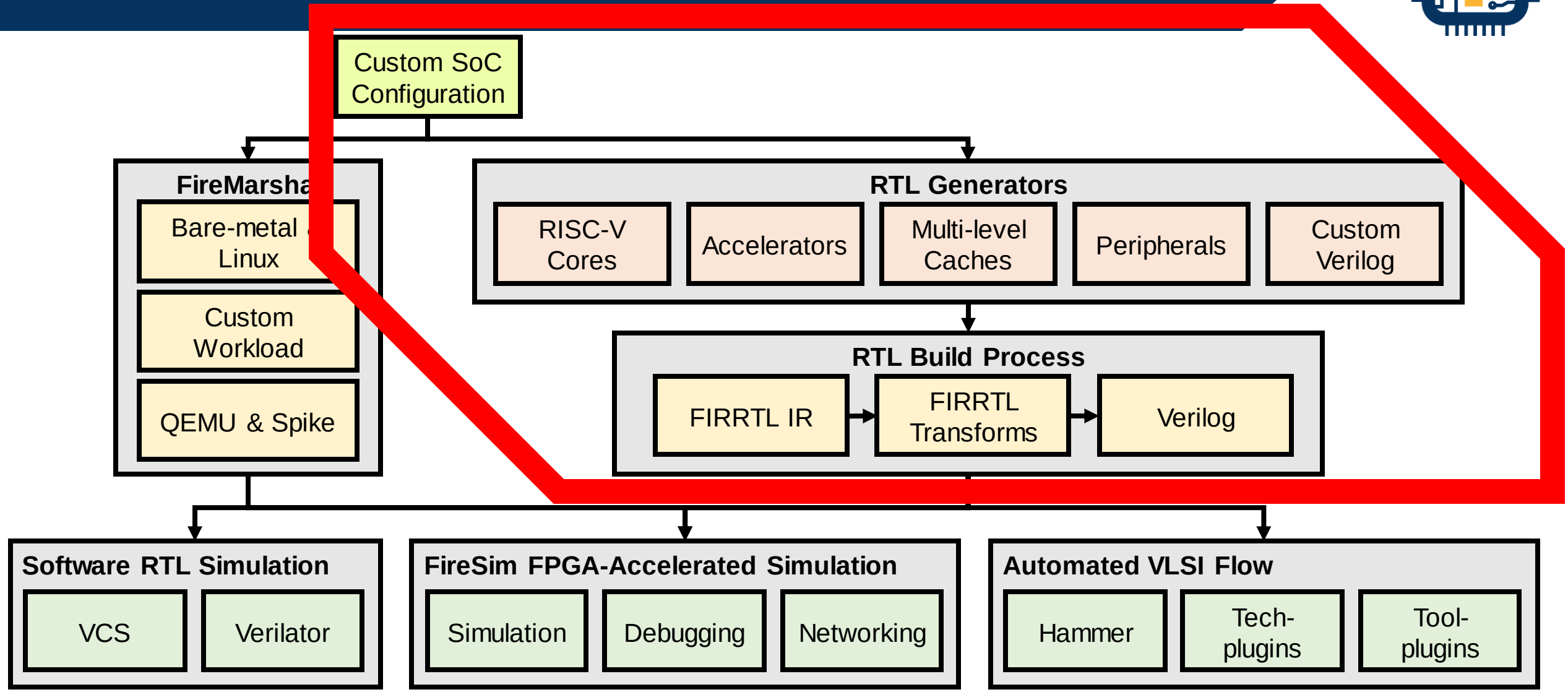
Chipyard VLSI Flow



Chipyard Unified Flows



Tutorial Roadmap



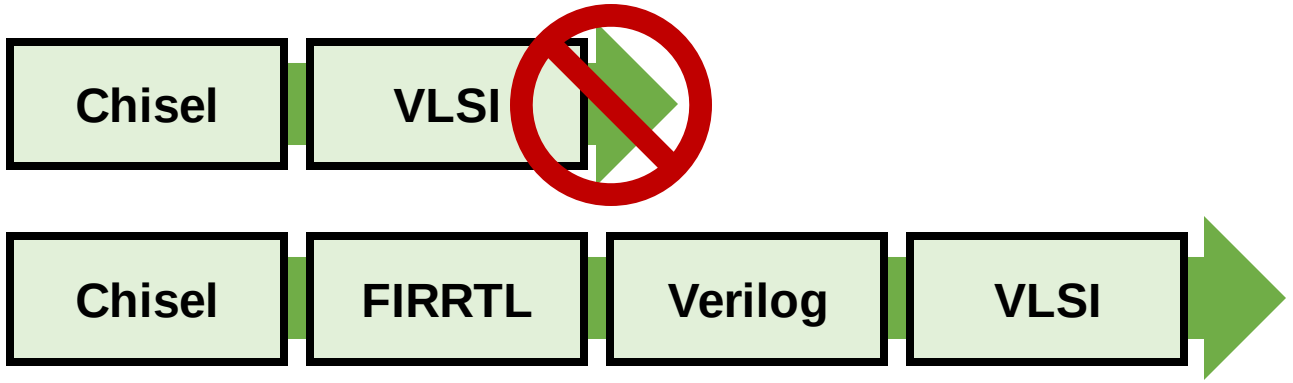
Chipyard Tooling

Berkeley
Architecture
Research

CHIPYARD

Chisel

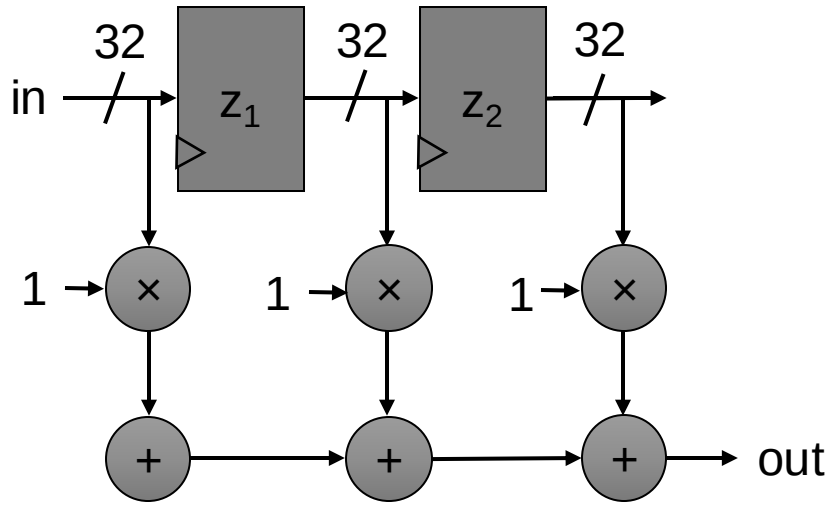
- Chisel – Hardware Construction Language built on Scala
- What Chisel **IS NOT:**
  - **NOT** Scala-to-gates
  - **NOT** HLS
  - **NOT** tool-oriented language
- What Chisel **IS:**
  - Productive language for **generating** hardware
  - Leverage **OOP/Functional programming** paradigms
  - Enables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



Chisel Example

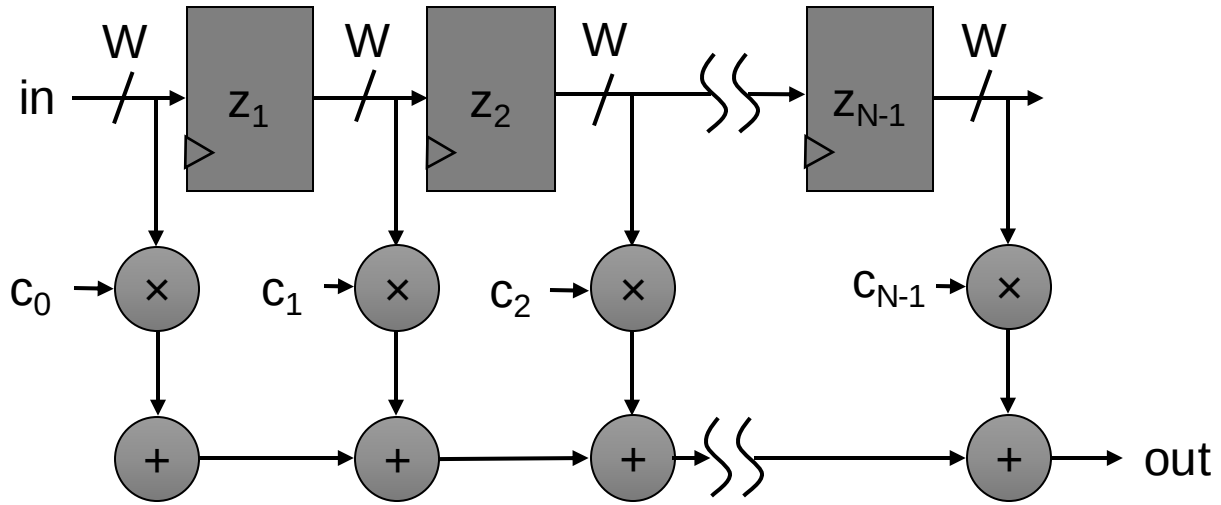
```
// 3-point moving average implemented in the
style of a FIR filter
class MovingAverage3 extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(32.W))
    val out = Output(UInt(32.W))
  })
  val z1 = RegNext(io.in)
  val z2 = RegNext(z1)

  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(_ + _)
}
```



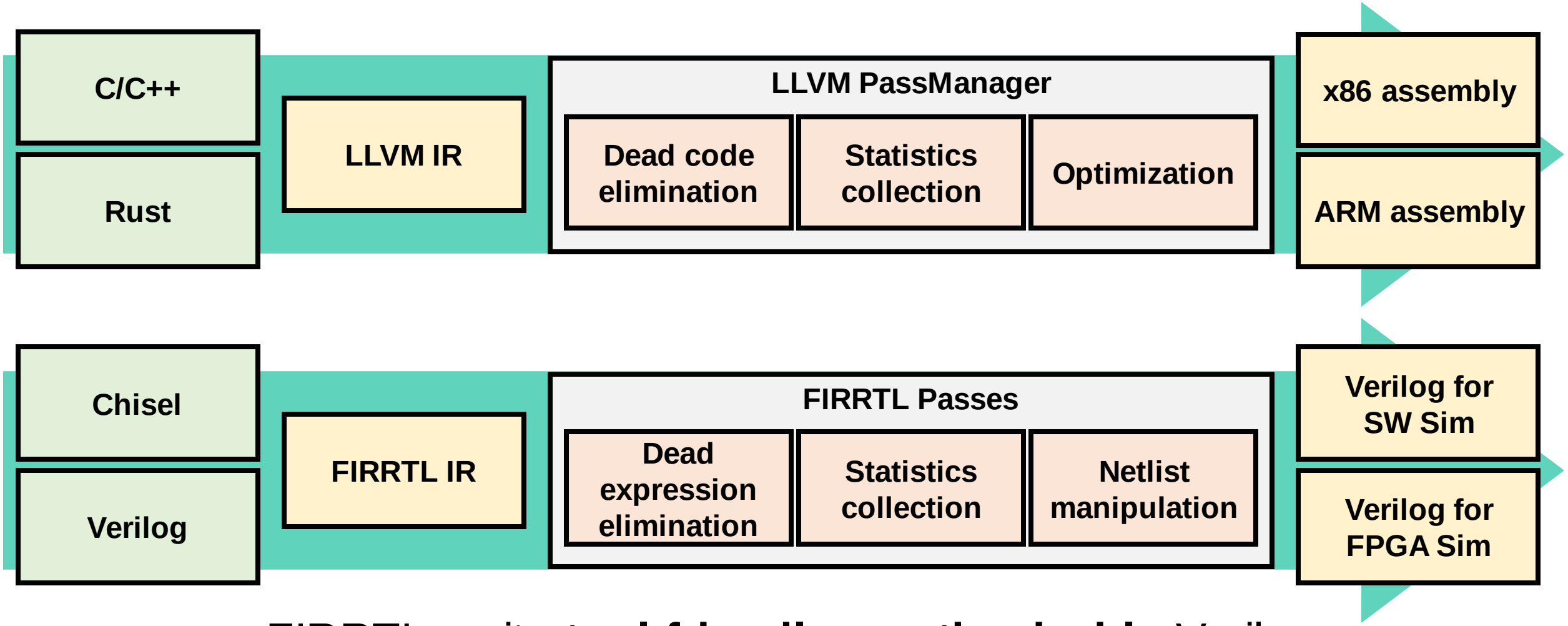
Chisel Example

```
// Basic implementation
val basic3Filter = Module(new MovingAverage3)

// Parameterized implementation
val better3Filter = Module(new FirFilter(32, Seq(1, 1, 1)))

// Generator is reusable
val delayFilter = Module(new FirFilter(8, Seq(0, 1)))
val triangleFilter = Module(new FirFilter(8, Seq(1, 2, 3, 2, 1)))
```

FIRRTL – LLVM for Hardware



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

Rocket Chip Generators

Berkeley
Architecture
Research

CHIPYARD

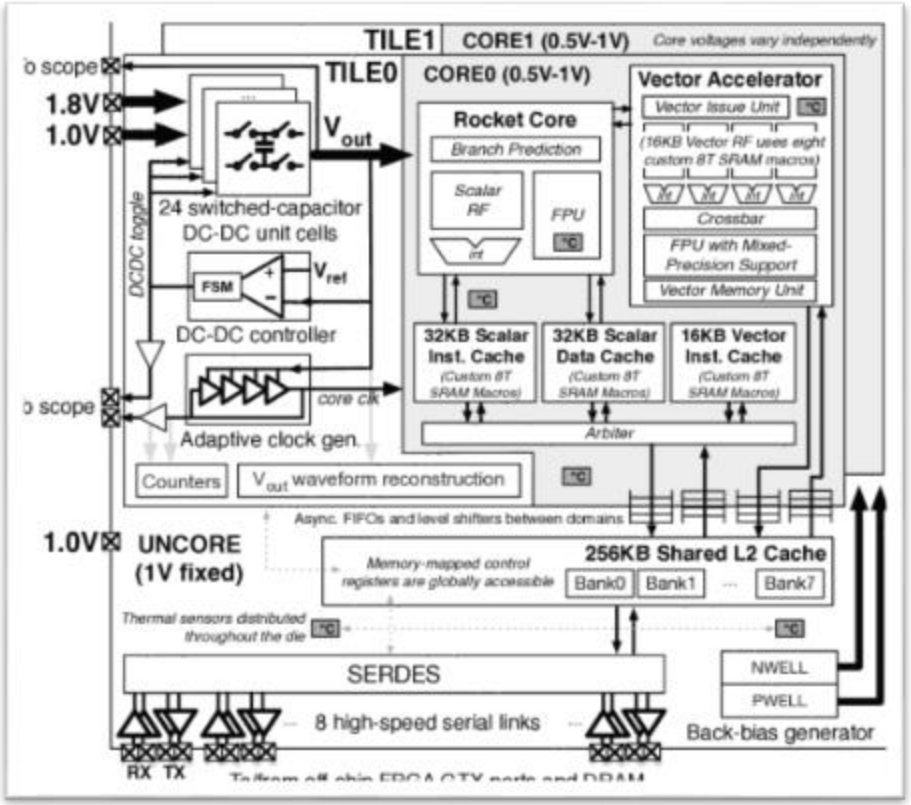
What is Rocket Chip?

- 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 Berkeley, now maintained by many
  - SiFive, ChipsAlliance, Berkeley

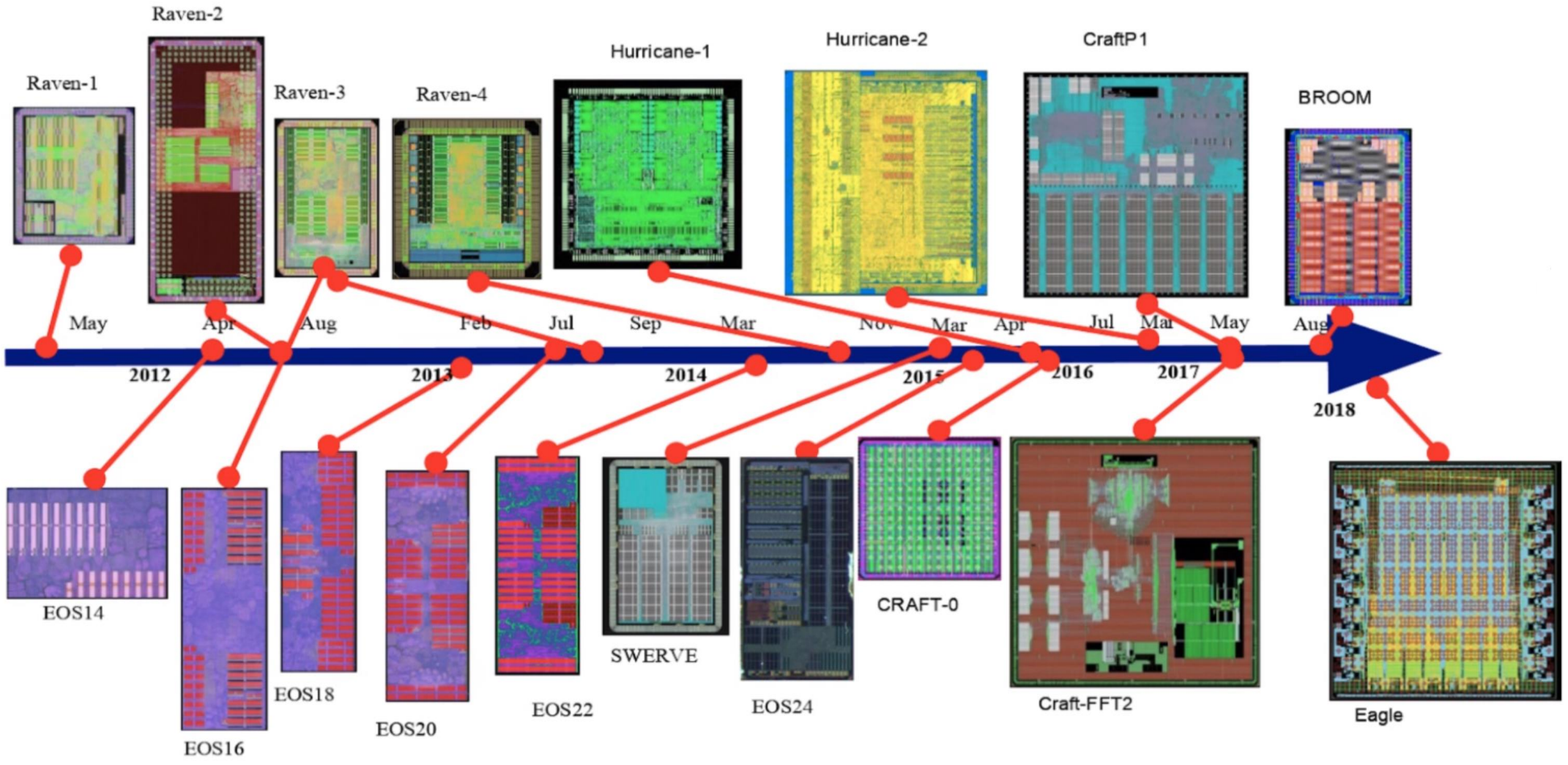
Generating Varied SoCs

In industry: **SiFive Freedom E310**

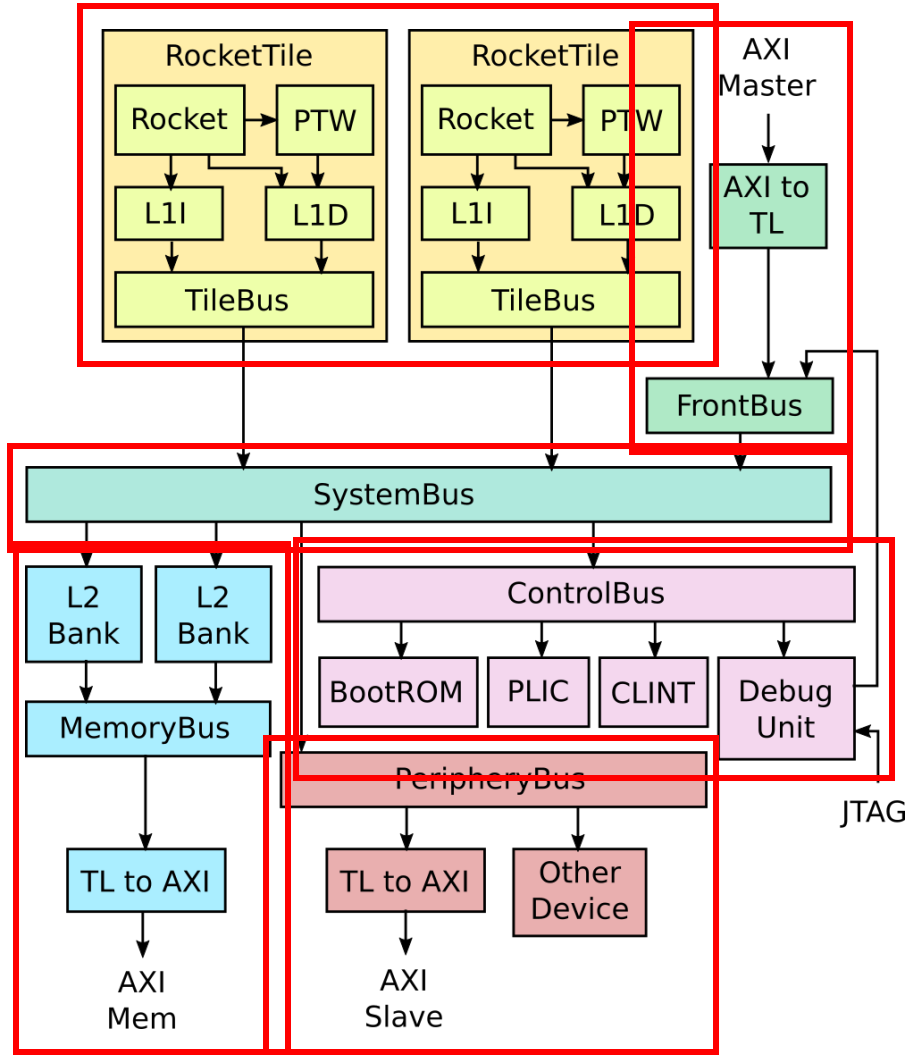
In academia: **UCB Hurricane-1**



Used in Many Tapeouts



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

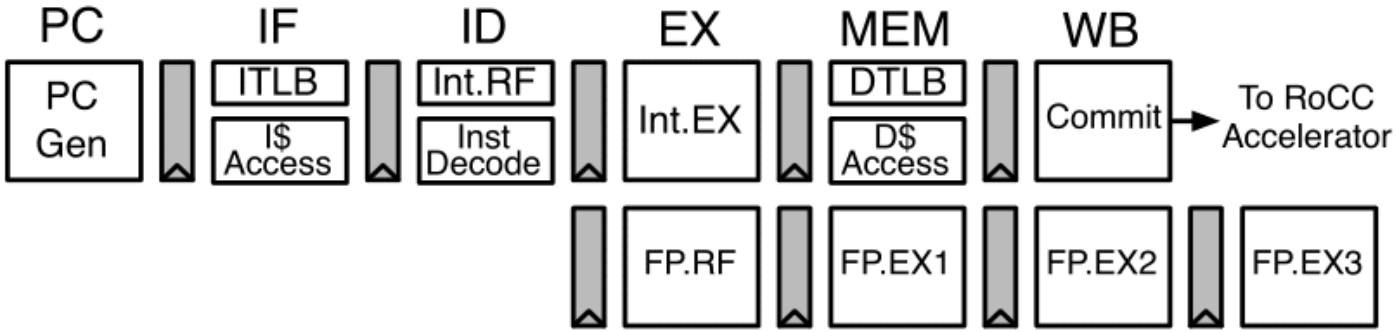
PeripheryBus:

- Connects to other devices

SystemBus:

- Ties everything together

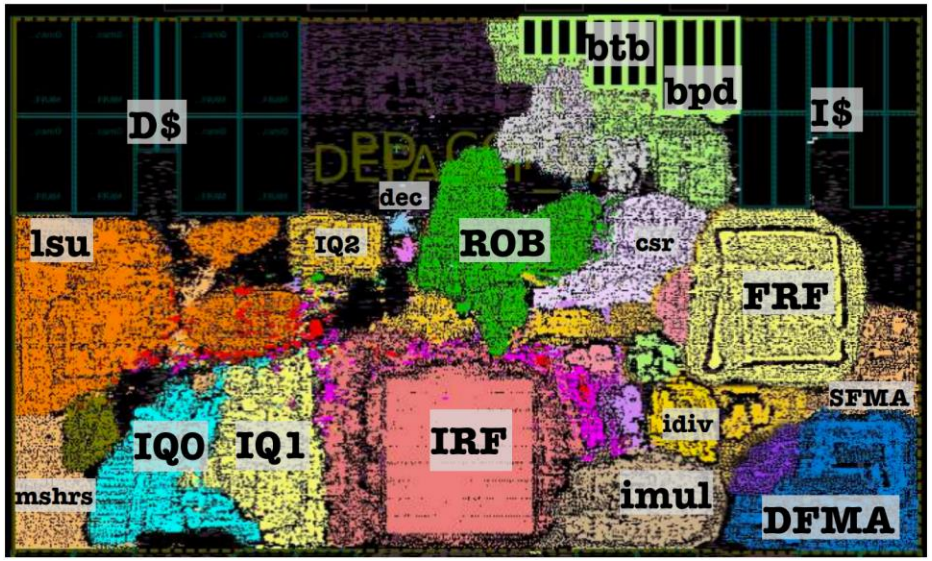
The Rocket In-Order Core

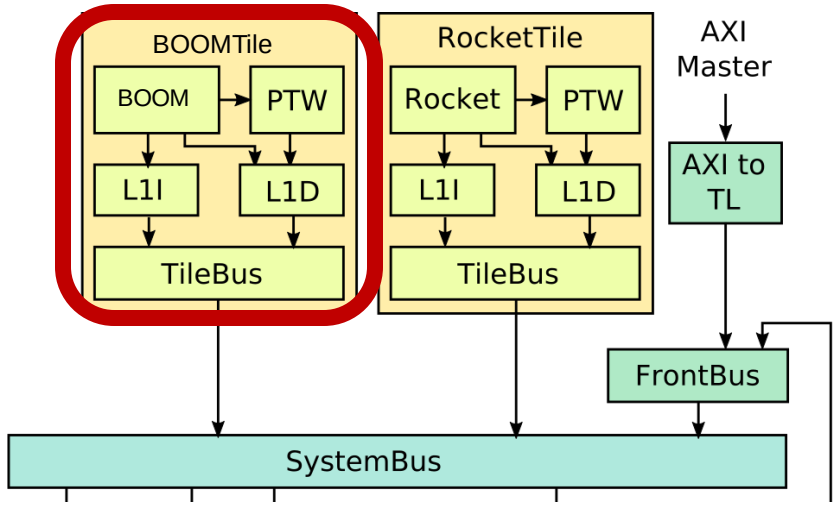


- First open-source RISC-V CPU
- In-order, single-issue RV64GC core
  - Floating-point via Berkeley hardfloat library
  - RISC-V Compressed
  - Physical Memory Protection (PMP) standard
  - Supervisor ISA and Virtual Memory
- Boots Linux
- Supports Rocket Chip Coprocessor (RoCC) interface
- L1 I$ and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

BOOM: The Berkeley Out-of-Order Machine

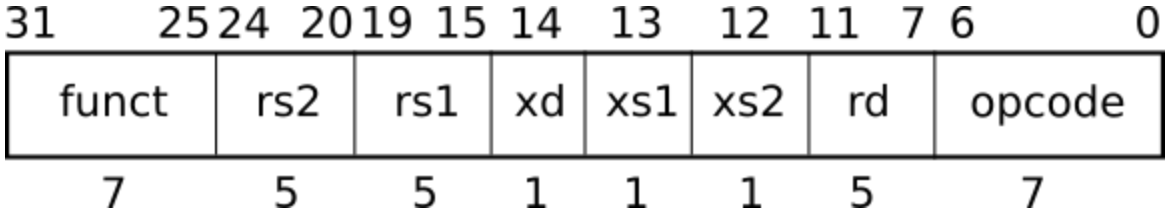
- Superscalar RISC-V OoO core
- Fully integrated in Rocket Chip ecosystem
- Open-source
- Described in Chisel
- Parameterizable generator
- Taped-out (BROOM at HC18)
- Full RV64GC ISA support
  - FP, RVC, Atomics, PMPs, VM, Breakpoints, RoCC
  - Runs real OS's, software
- Drop-in replacement for Rocket



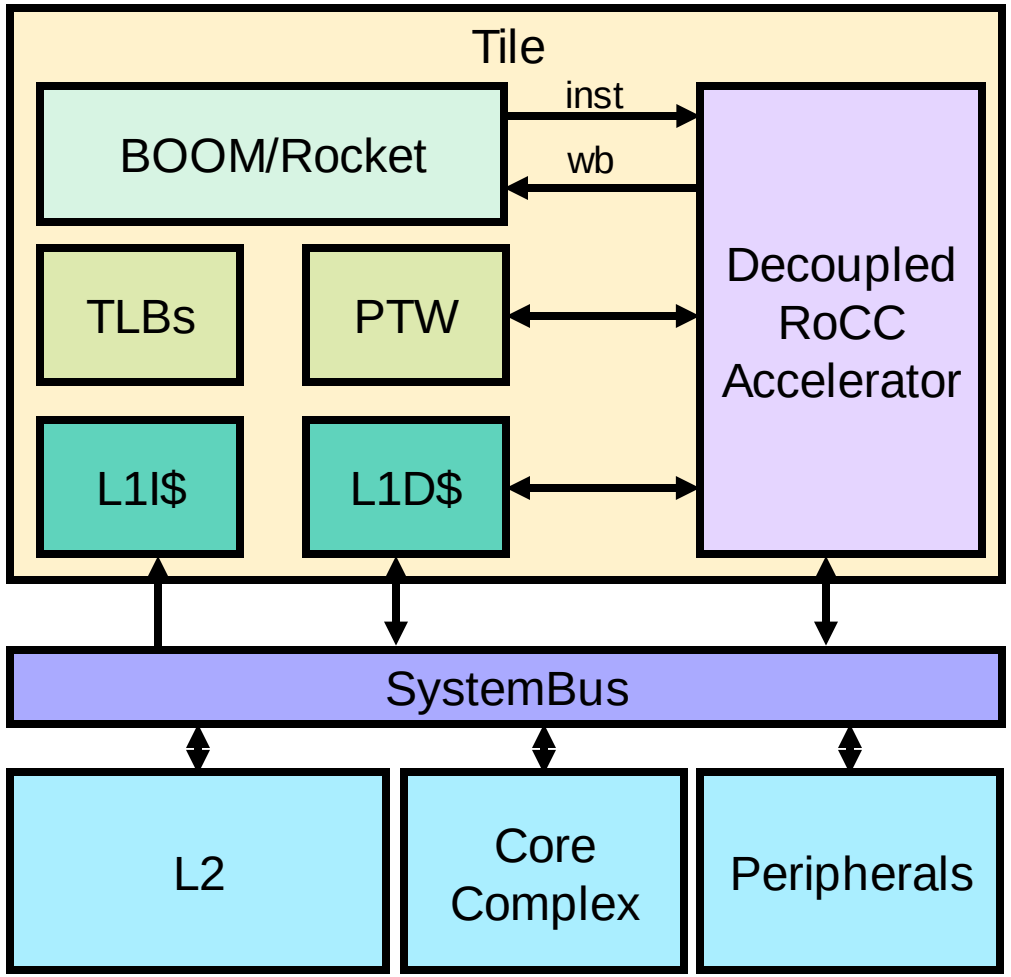


RoCC Accelerators

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

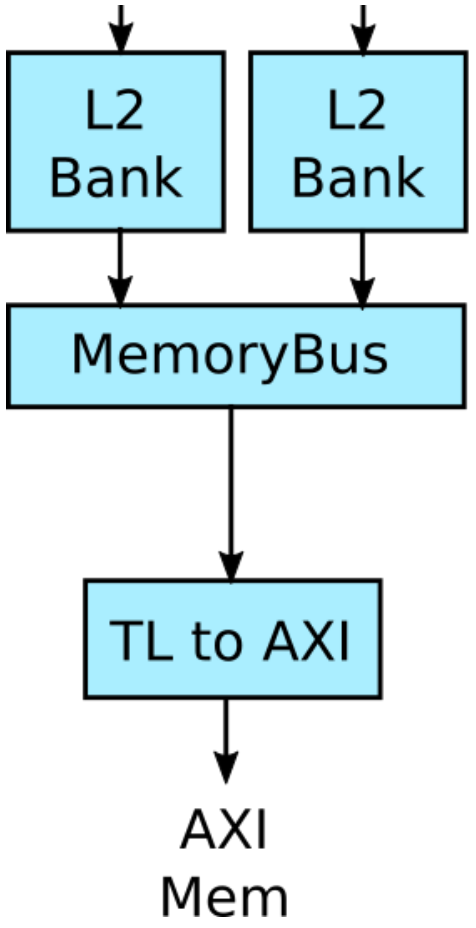


- Examples of RoCC accelerators
  - Vector accelerators
  - Memcpy accelerator
  - Machine-learning accelerators
  - Java GC accelerator



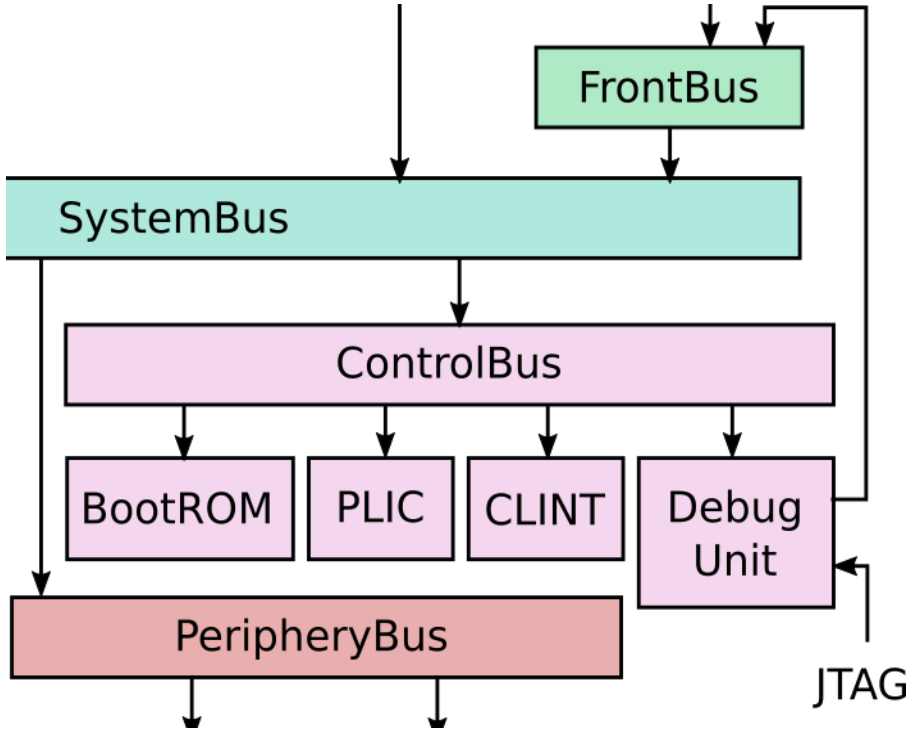
L2 Cache and Memory System

- 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

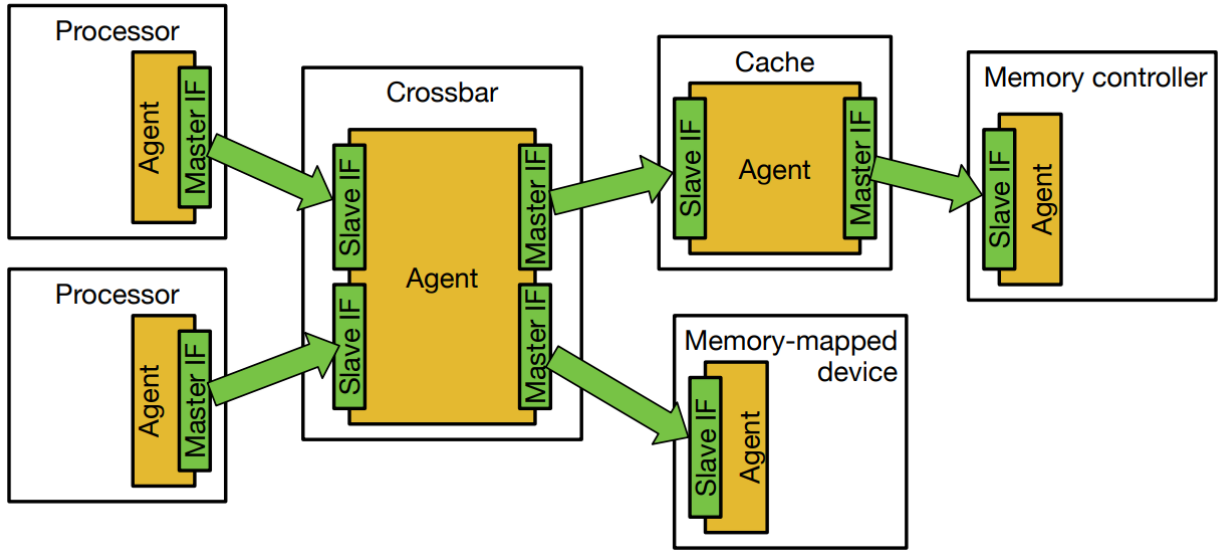
- BootROM
  - First-stage bootloader
  - DeviceTree
- PLIC
- CLINT
  - Software interrupts
  - Timer interrupts
- Debug Unit
  - DMI
  - JTAG



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
- **SHA3:** Educational SHA3 RoCC accelerator

TileLink Interconnect

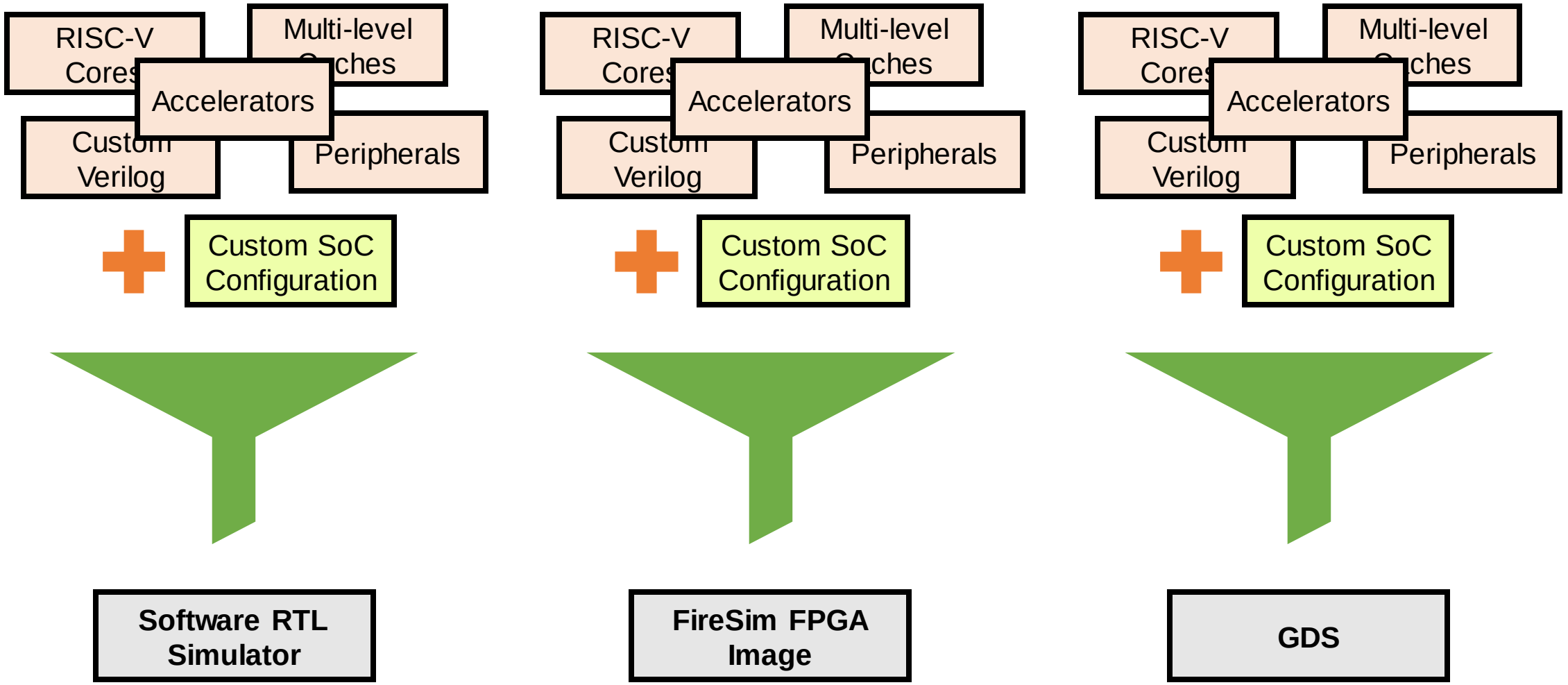


- Free and open chip-scale interconnect standard
- Supports multiprocessors, coprocessors, accelerators, DMA, peripherals, etc.
- Provides a physically addressed, shared-memory system
- Supports cache-coherent shared memory, MOESI-equivalent protocol
- Verifiable deadlock freedom for conforming SoCs

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



Diplomacy

Problem: Interconnects are difficult to parameterize correctly

- Complex interconnect graph with many nodes
- Nodes are independently parameterized

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 Examples

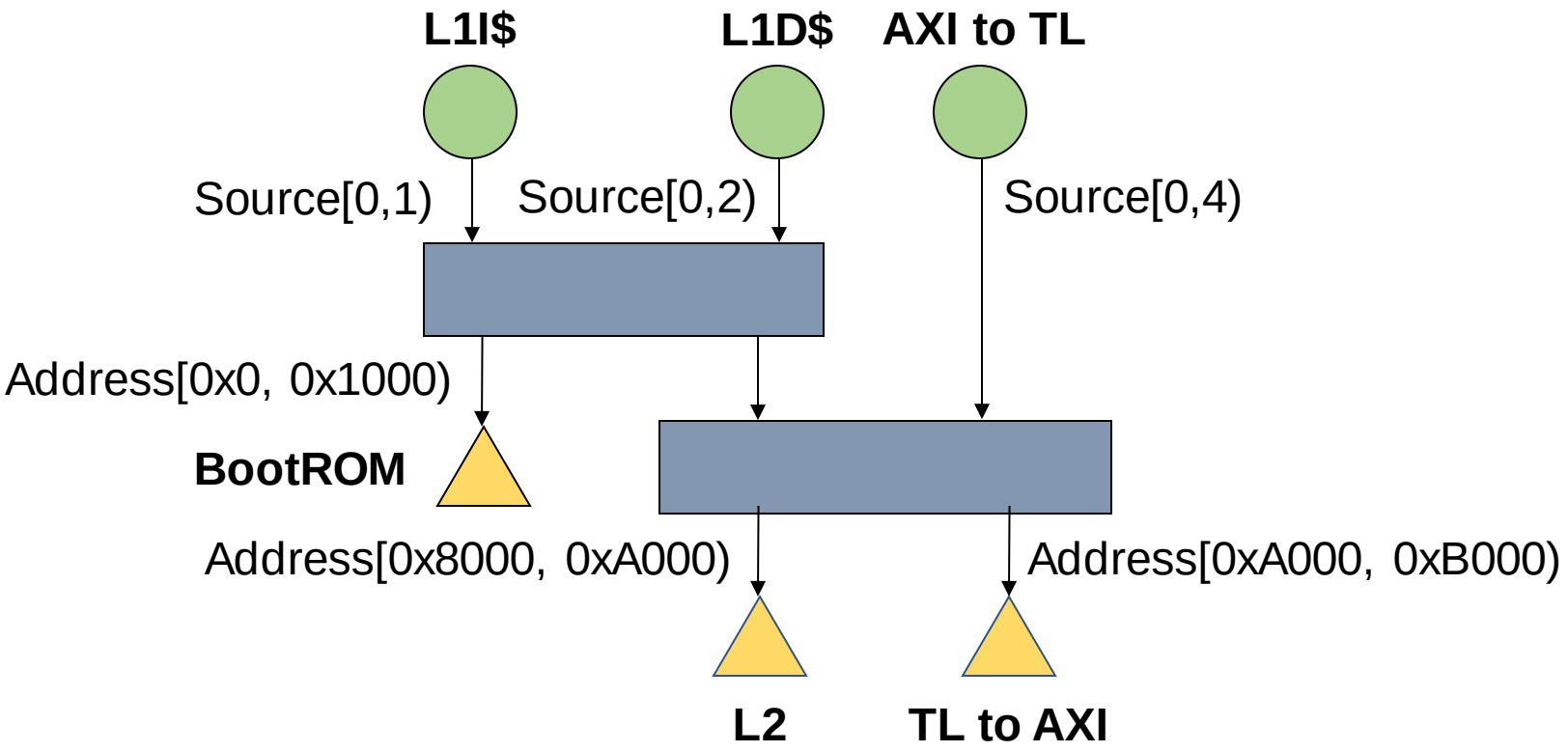
Diplomatic parameters

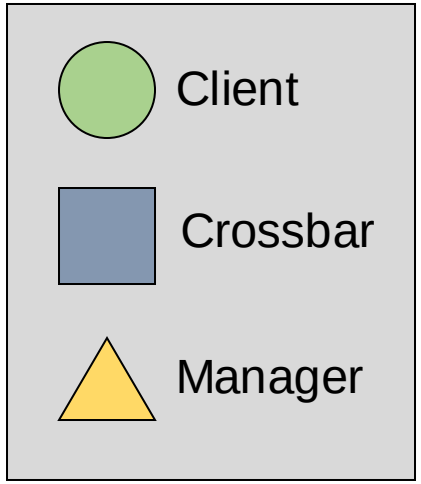
- Type and size of supported operations
- Physical memory attributes – modifiability, executability, cacheability
- Ordering requirements on operations (ex: FIFO)
- Presence and widths of fields in wire bundles (ex: source ID bits)

Useful applications:

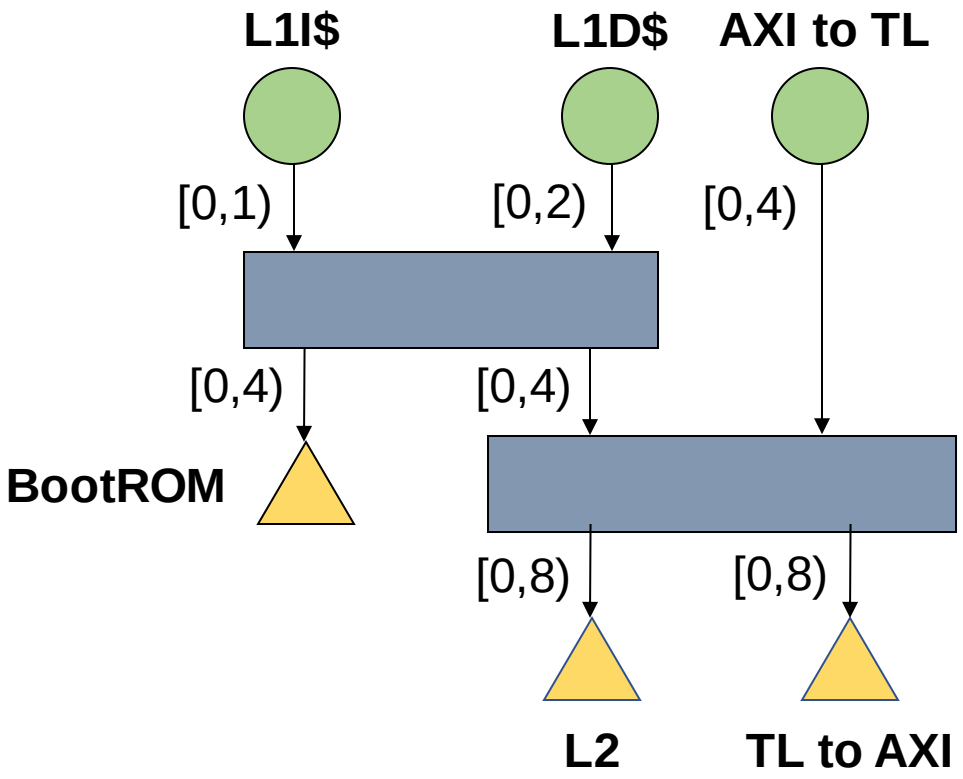
- Automatically insert TLMonitor protocol correctness checkers
- Discover AtomicAutomata topology violations

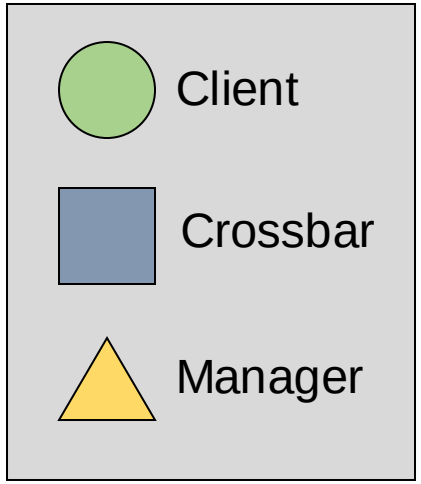
Diplomacy Example



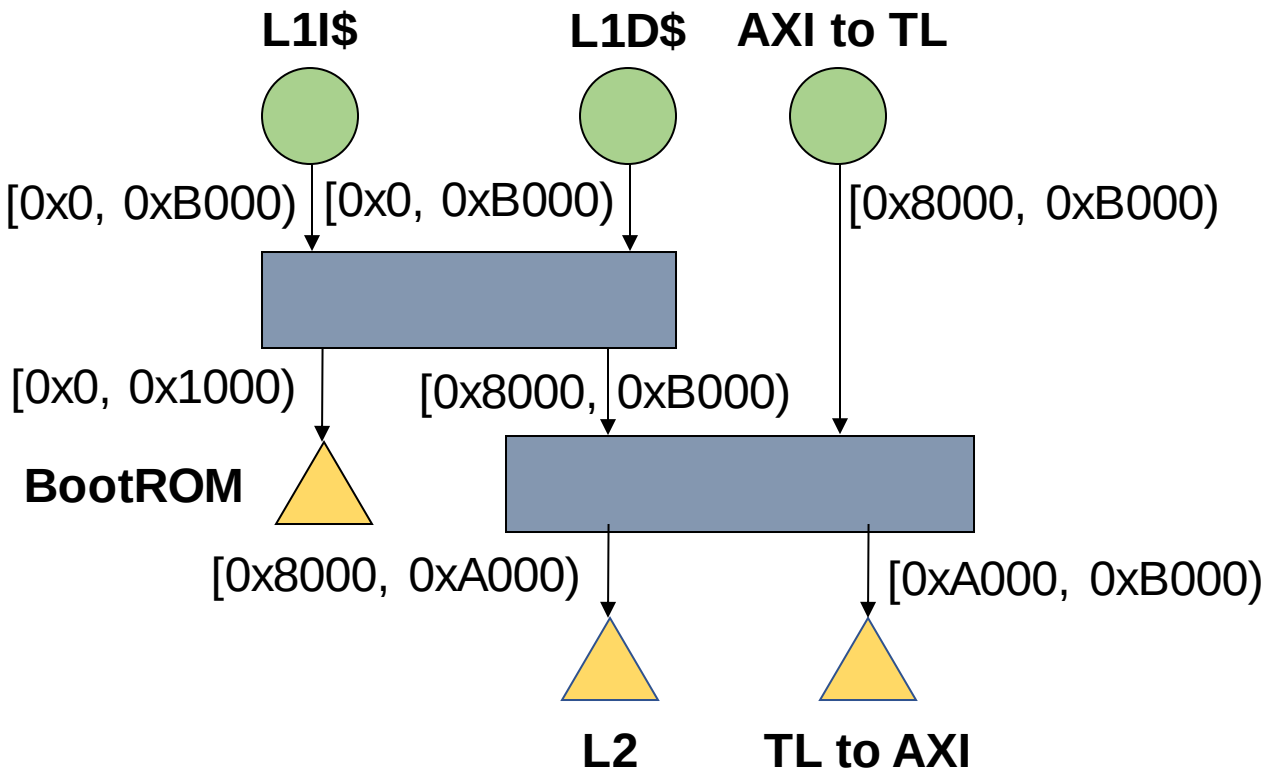


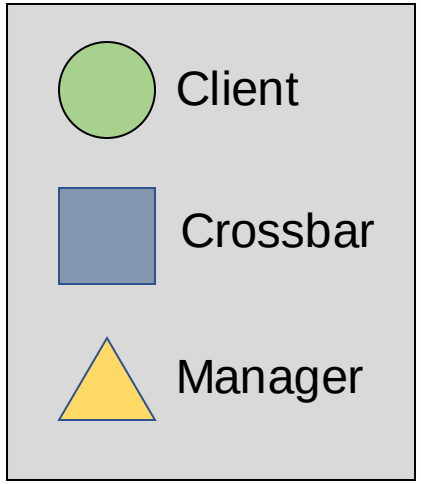
Diplomacy Example



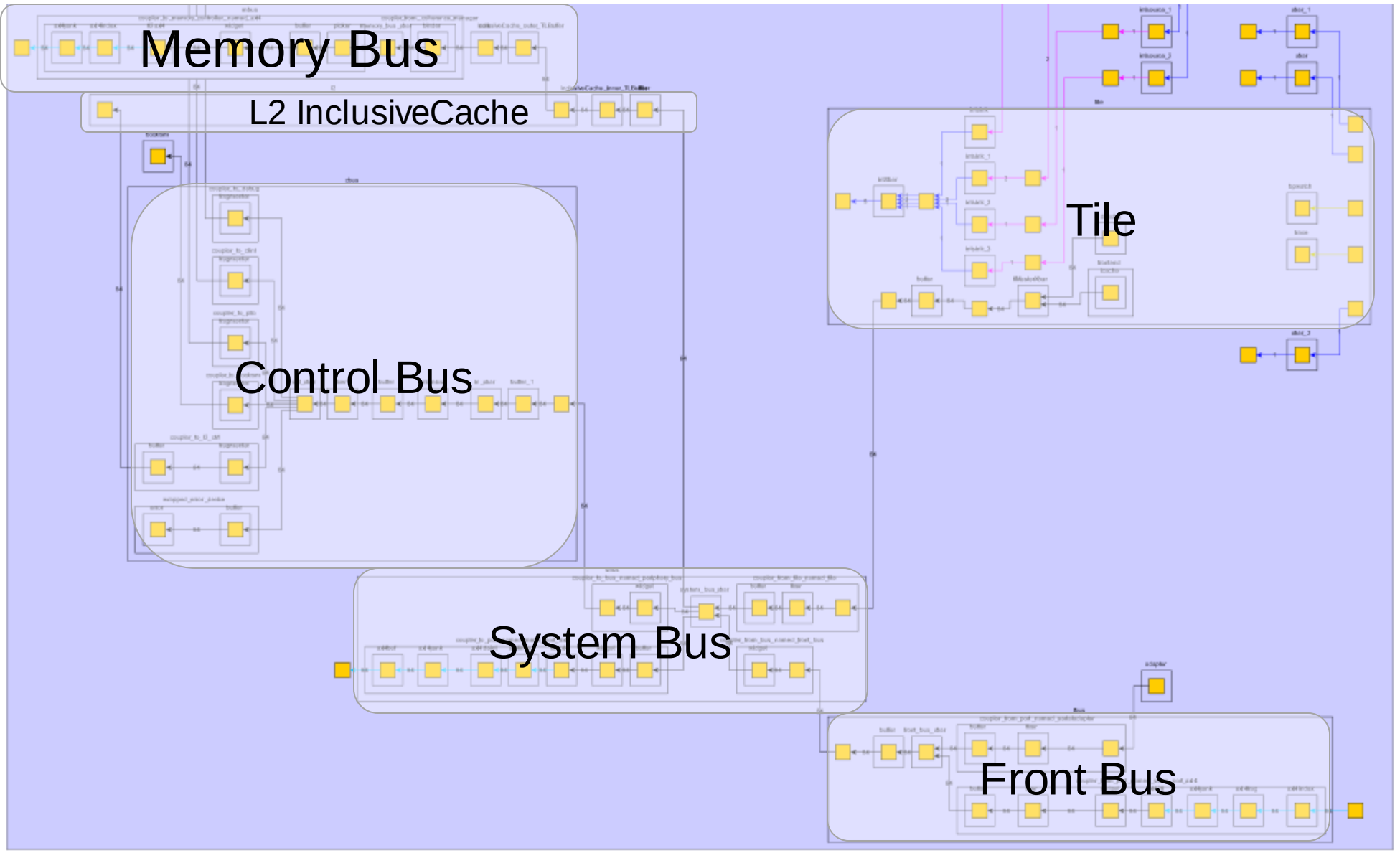


Diplomacy Example



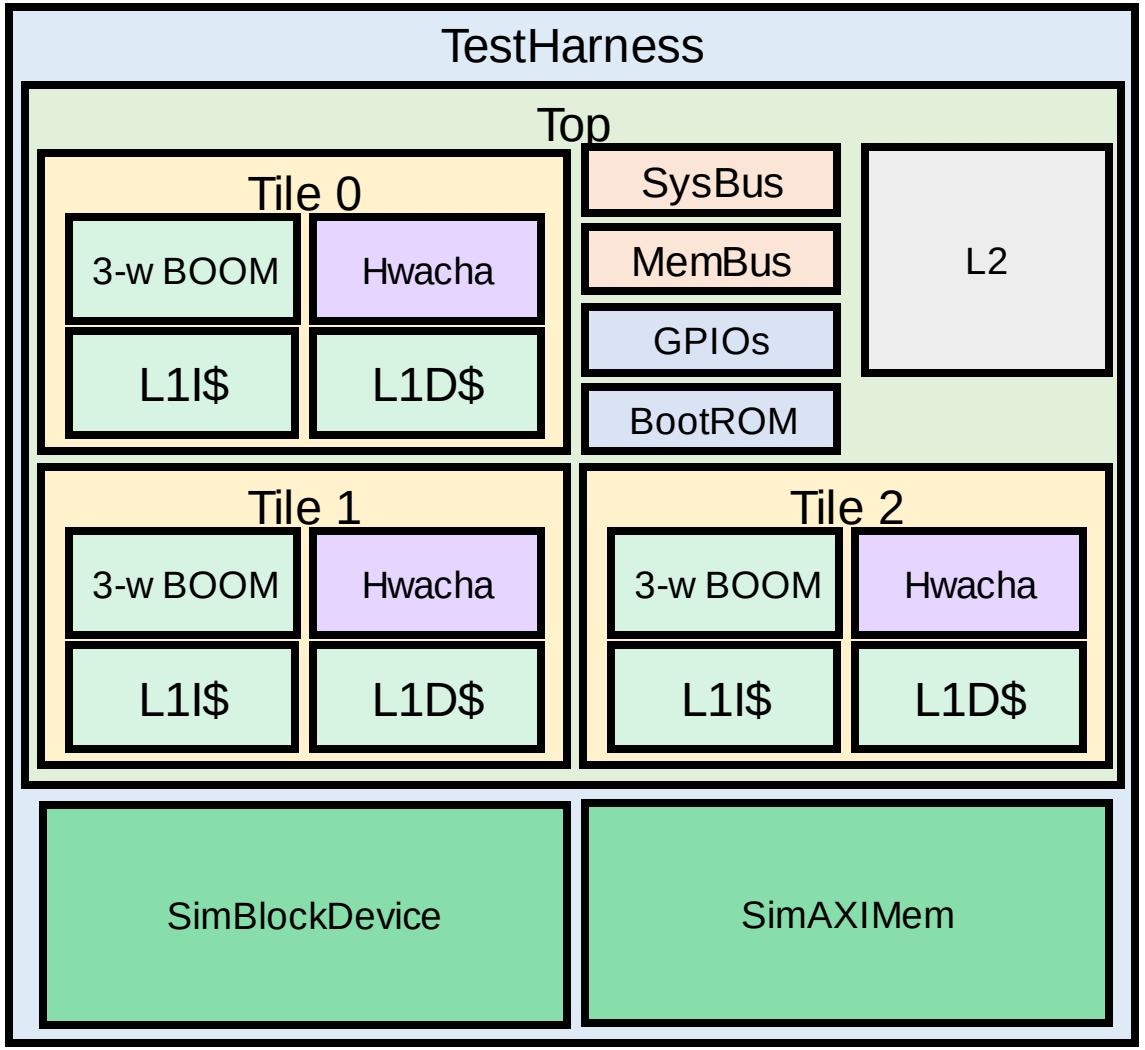


Diplomacy-generated Graph



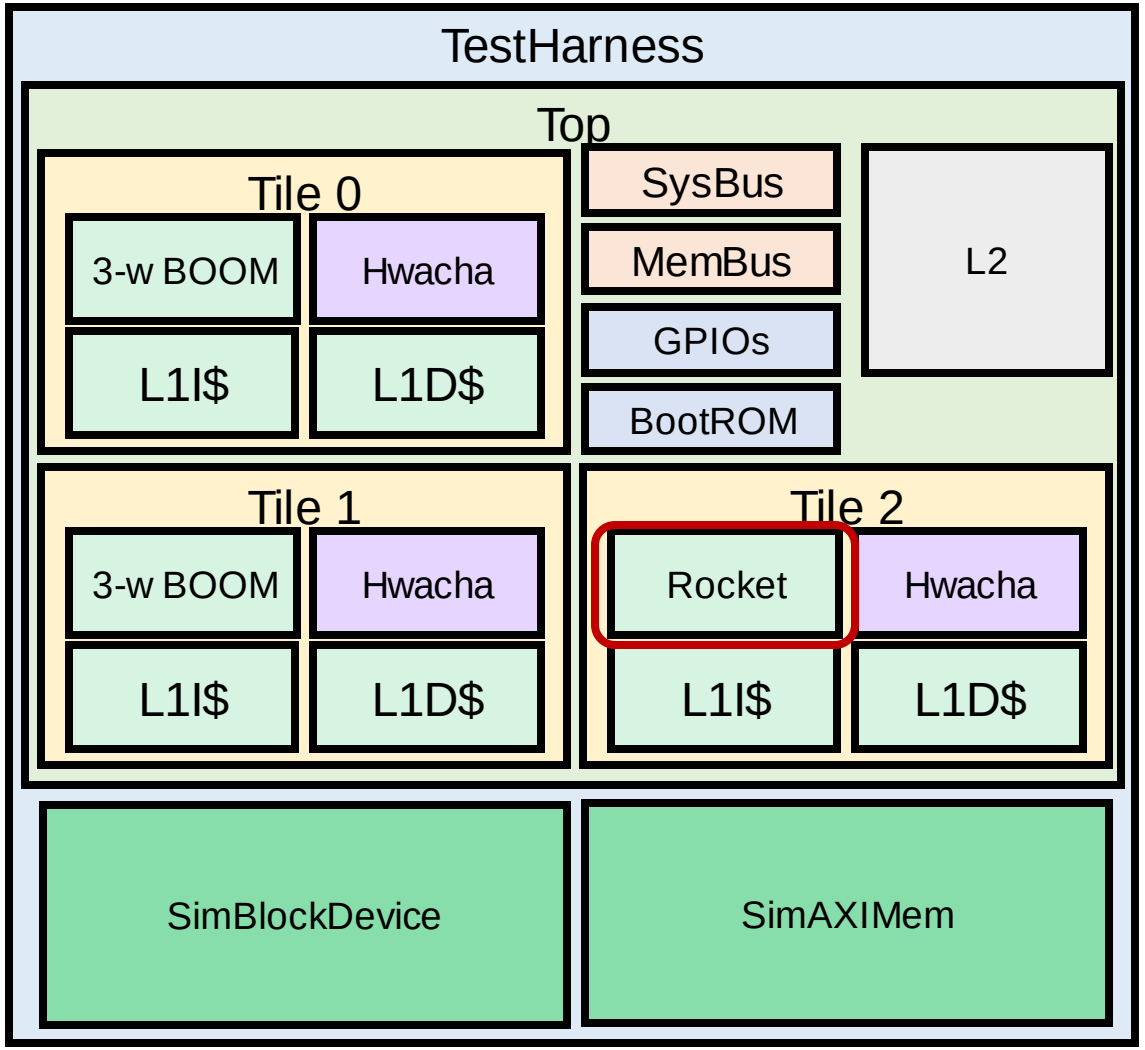
Rocket Chip Configuration

```
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)
```



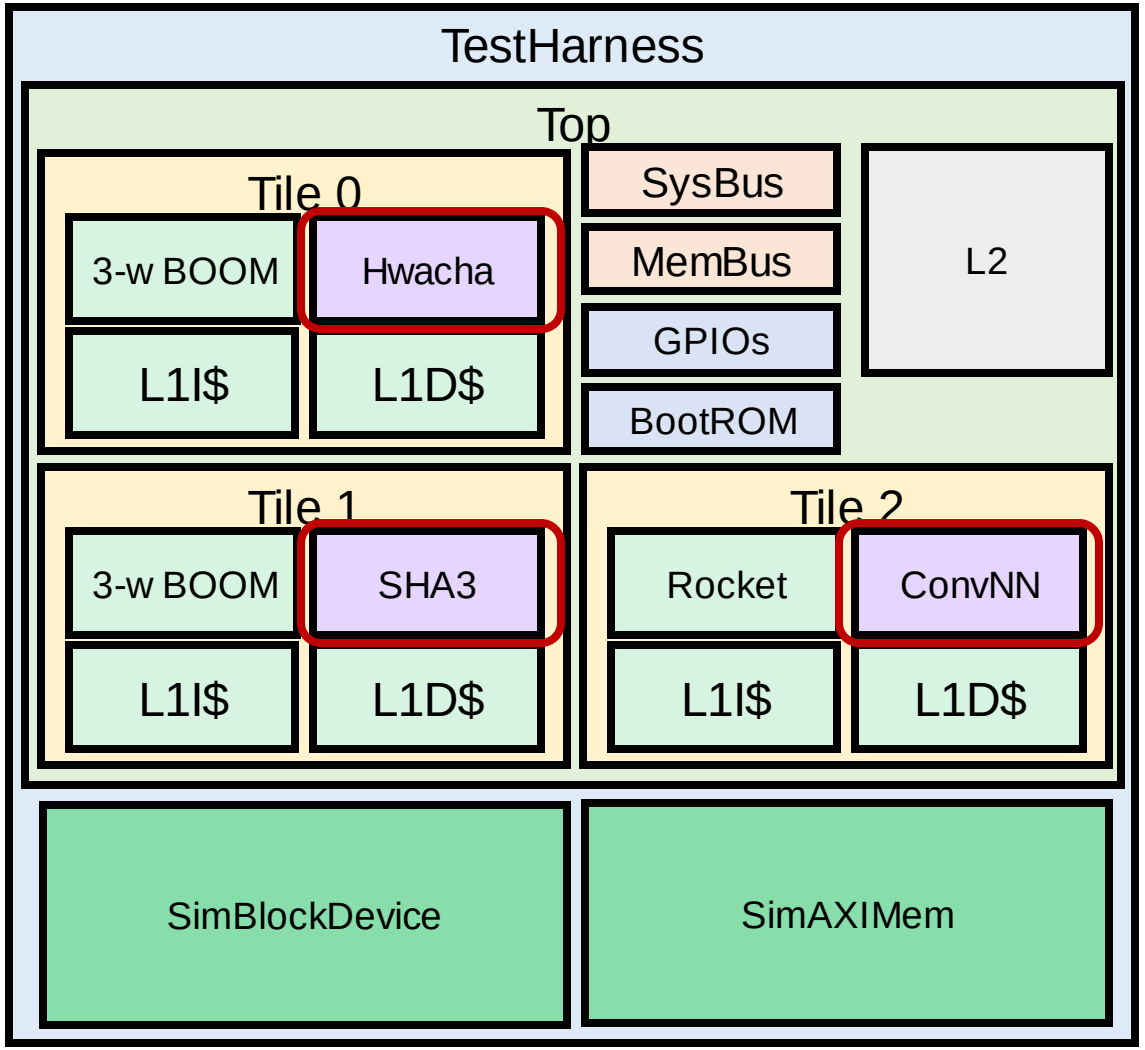
Rocket Chip Configuration

```
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)
```



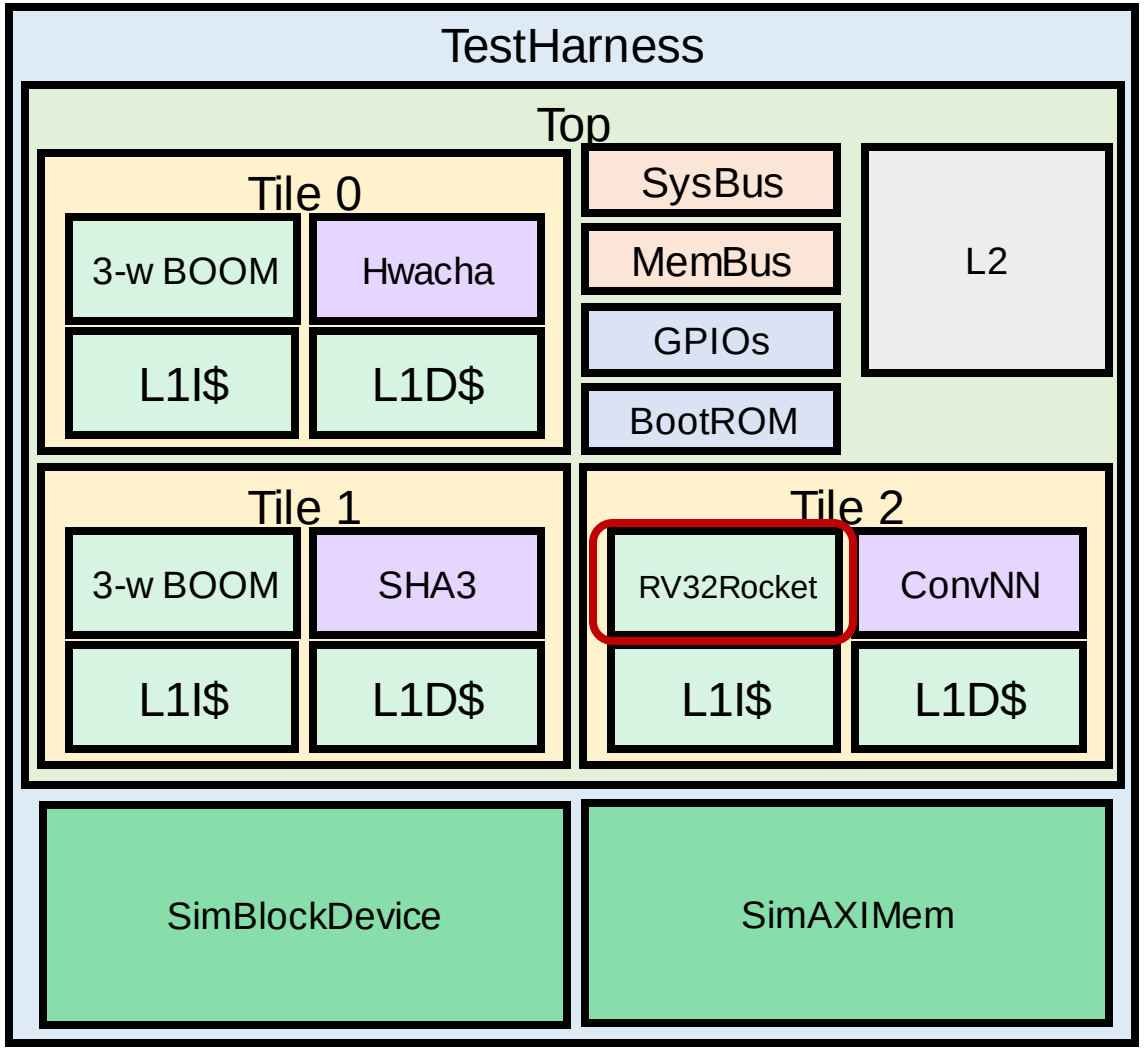
Rocket Chip Configuration

```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new WithMultiRoCCConvAccel(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)
```



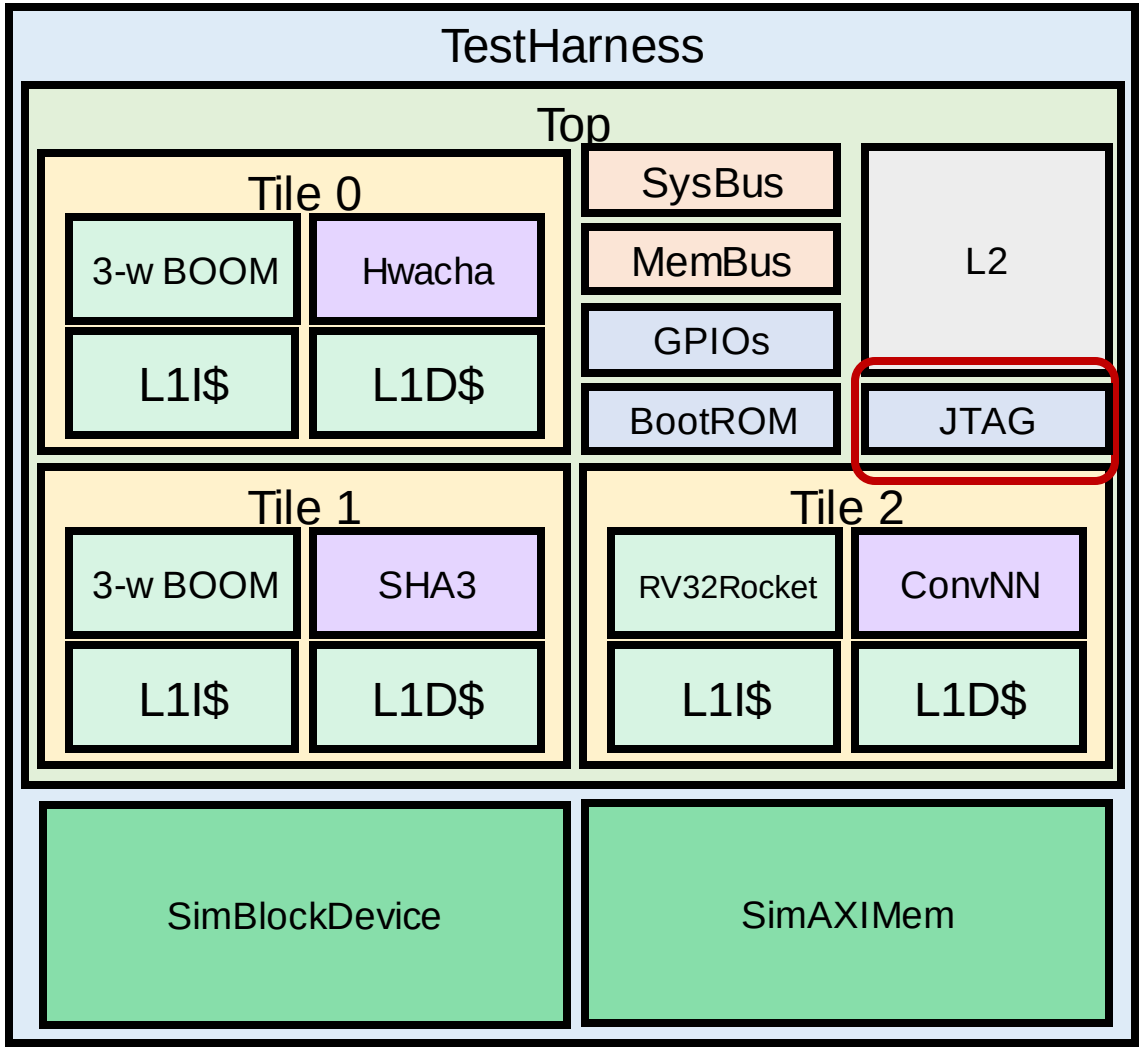
Rocket Chip Configuration

```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithBootROM                           ++
  new WithMultiRoCCConvAccel(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

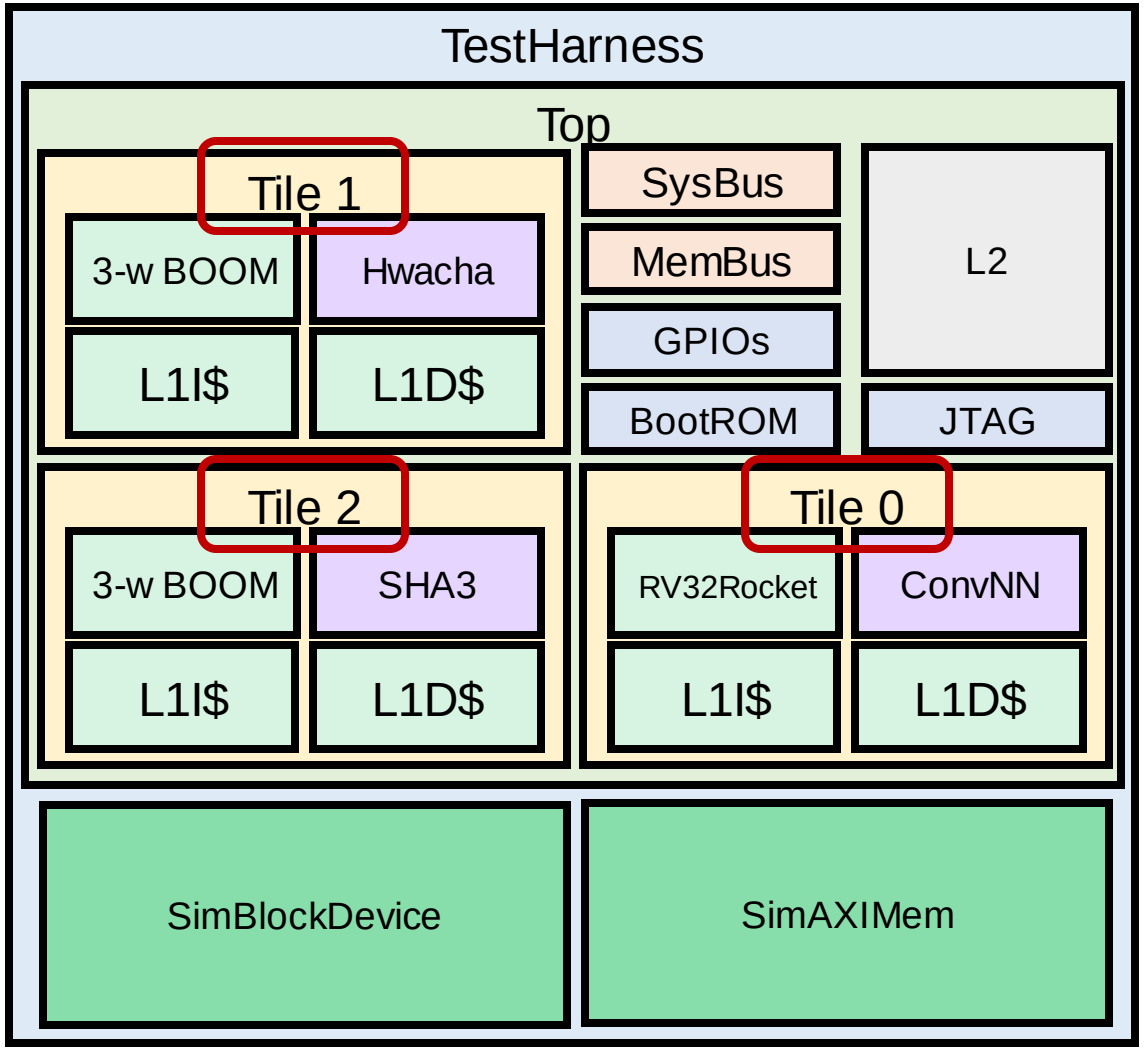
```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithJtagDTM                           ++
  new WithBootROM                           ++
  new WithMultiRoCCConvAccel(2)             ++
  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)
```



Berkeley Architecture Research

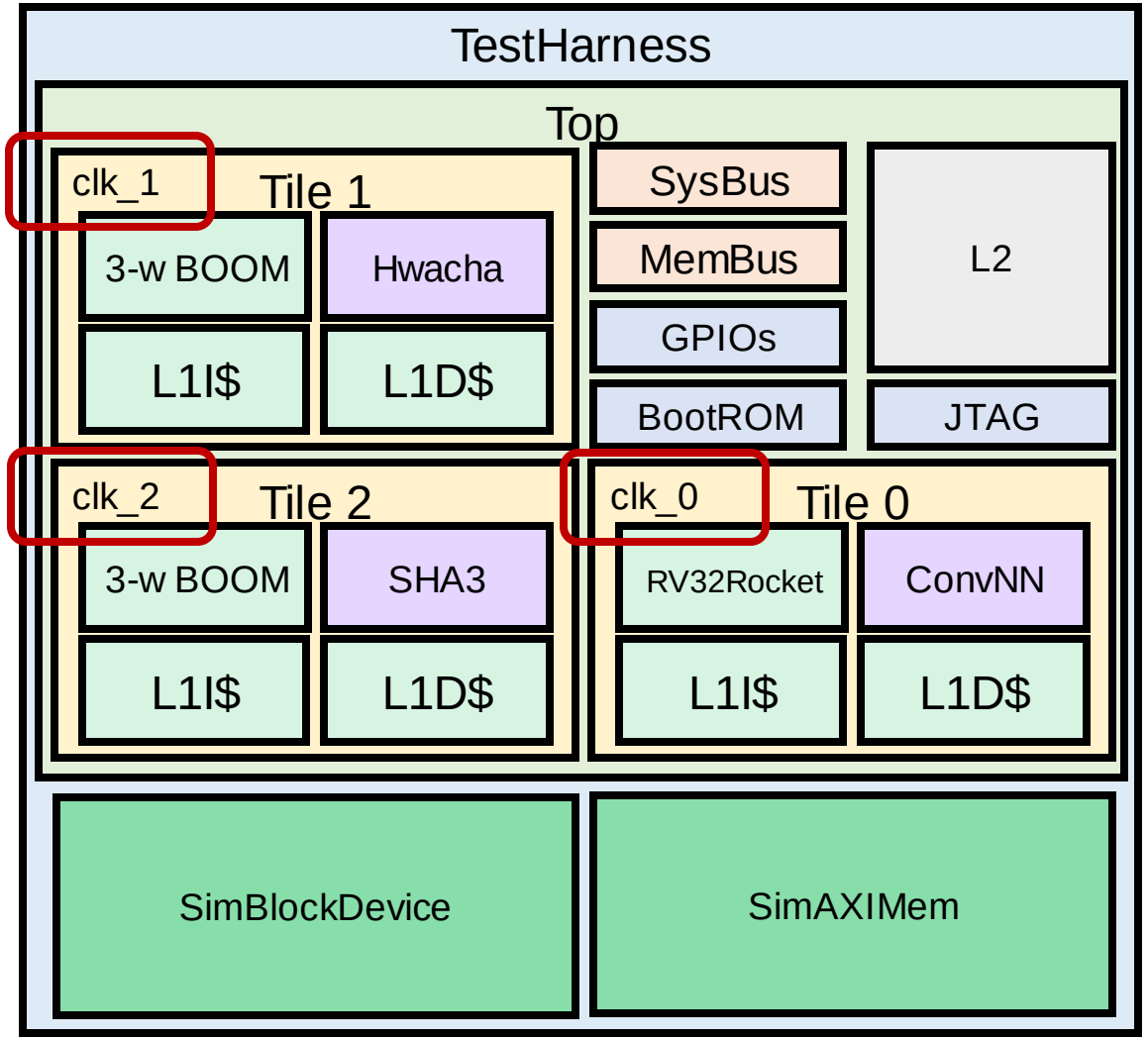
Rocket Chip Configuration

```
class MyCustomConfig extends Config(
  new WithExtMemSize((1<<30) * 2L)          ++
  new WithBlockDevice                       ++
  new WithGPIO                              ++
  new WithJtagDTM                           ++
  new WithBootROM                           ++
  new WithRenumberHarts(rocketFirst=true)   ++
  new WithMultiRoCCConvAccel(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 WithRenumberHarts(rocketFirst=true)   ++
  new WithRationalBoomTiles                 ++
  new WithRationalRocketTiles               ++
  new WithMultiRoCCConvAccel(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 is Community-Friendly

Documentation:

- https://chipyard.readthedocs.io/en/dev/
- 85 pages
- Documents components, flows
- Links to sub-project documentation
- Most of today's tutorial content is covered there

Continuous Integration:

- Cloud-hosted
- https://circleci.com/gh/ucb-bar/chipyard/tree/master

Chipyard is Research-Friendly

- Add new accelerators/custom instructions
- Modify OS/driver/software
- Perform design-space exploration across many parameters
- Test in software and FPGA-sim before tape-out

Stay-tuned for Chipyard-based research from Berkeley

- New chips
- New accelerators

High-level questions?

- Next is a hands-on tutorial led by Abe Gonzalez