SoC Architecture and Components

Jerry Zhao

Use Cases

Custom SoC architecture
New blocks + reusing existing blocks

RTL Simulation running test binaries/micro-benchmarks

FPGA-accelerated simulation running full workloads

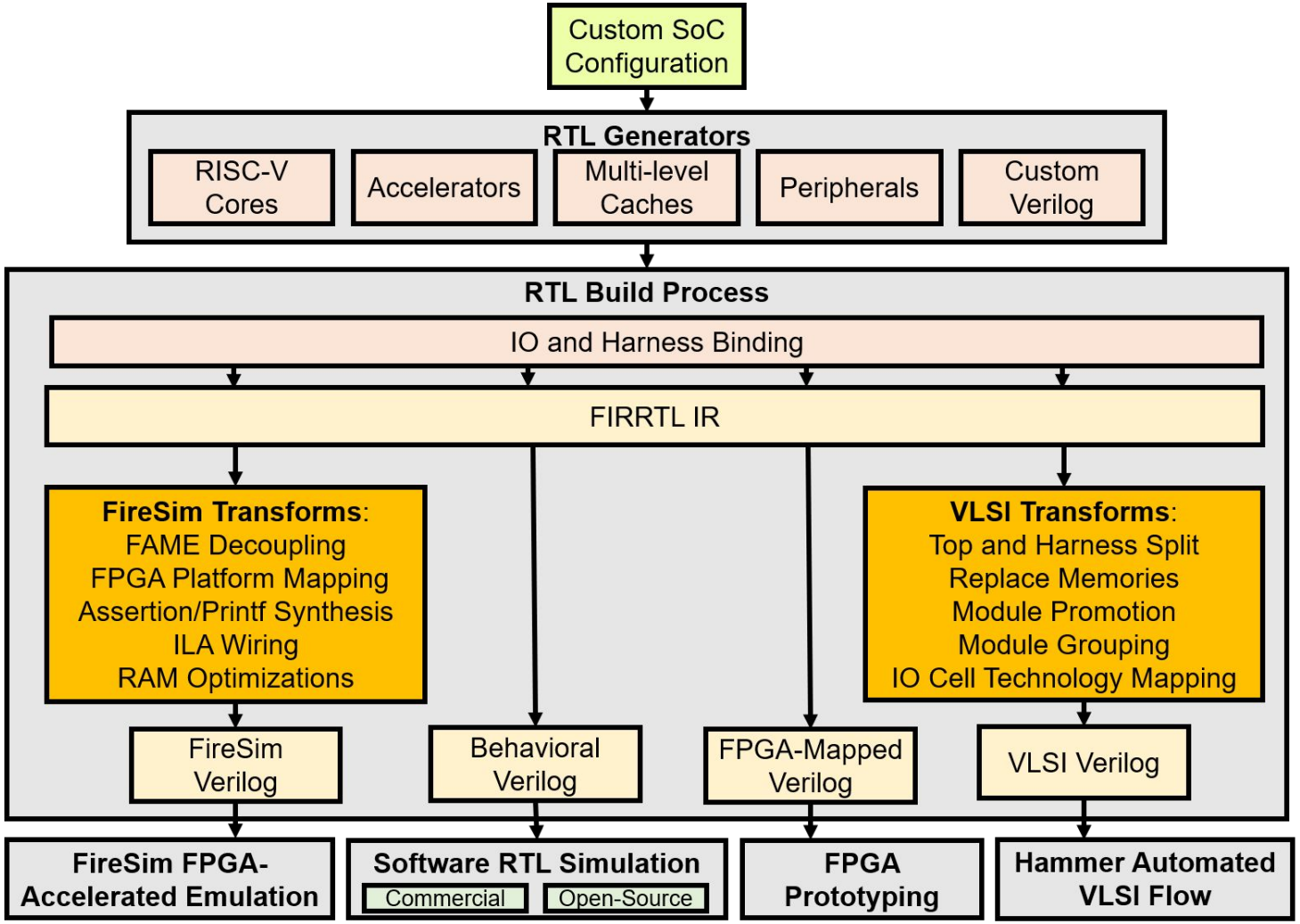
FPGA prototyping for fast cool demos

Tape-out the SoC to get actual silicon results

CHIPYARD Organization

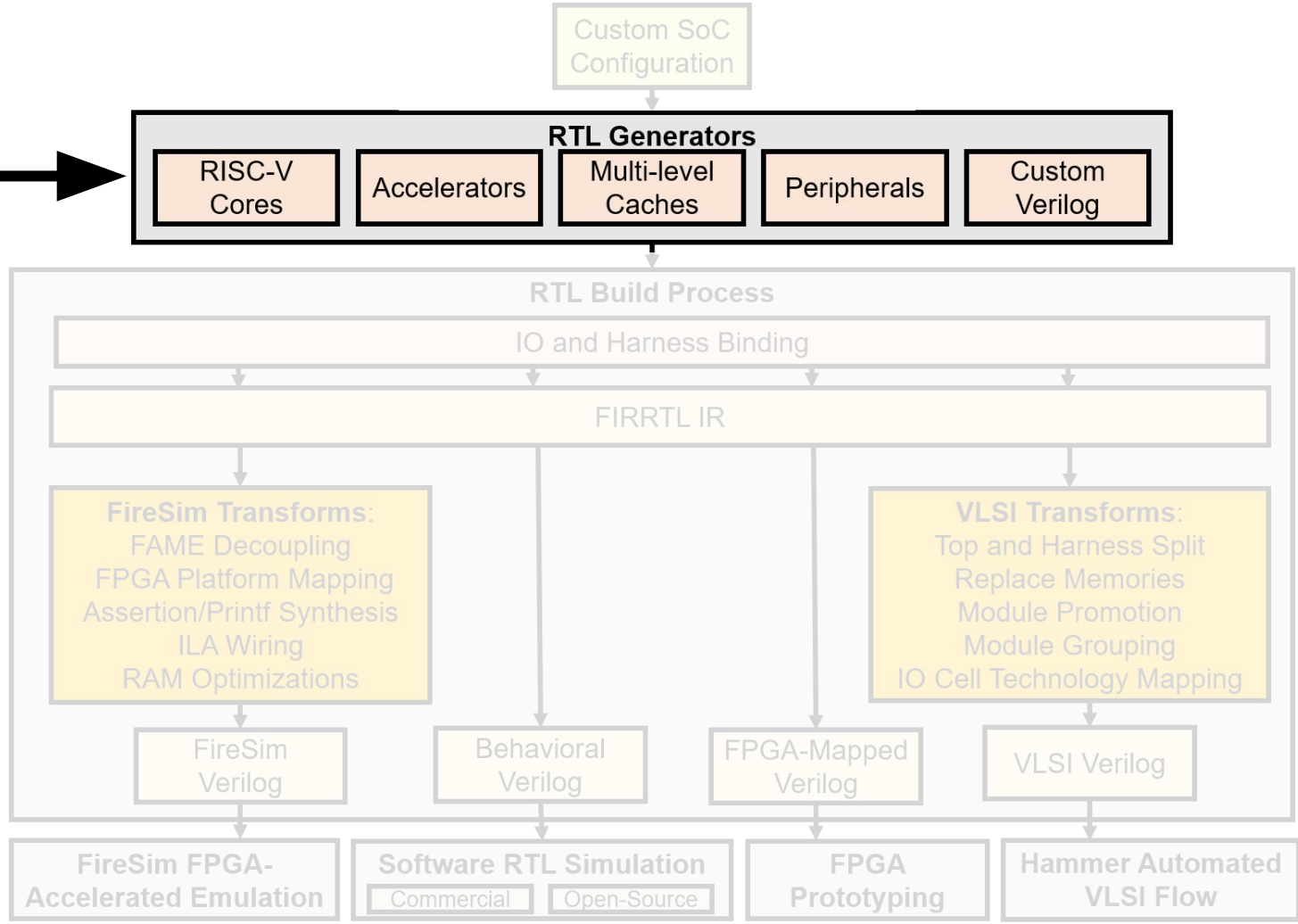
What is Chipyard?

- An organized **framework** for various SoC design tools
- A **curated IP library** of open-source RISC-V SoC components
- A **methodology** for agile SoC architecture design, exploration, and evaluation
- A **tapeout-ready chassis** for custom RISC-V SoCs

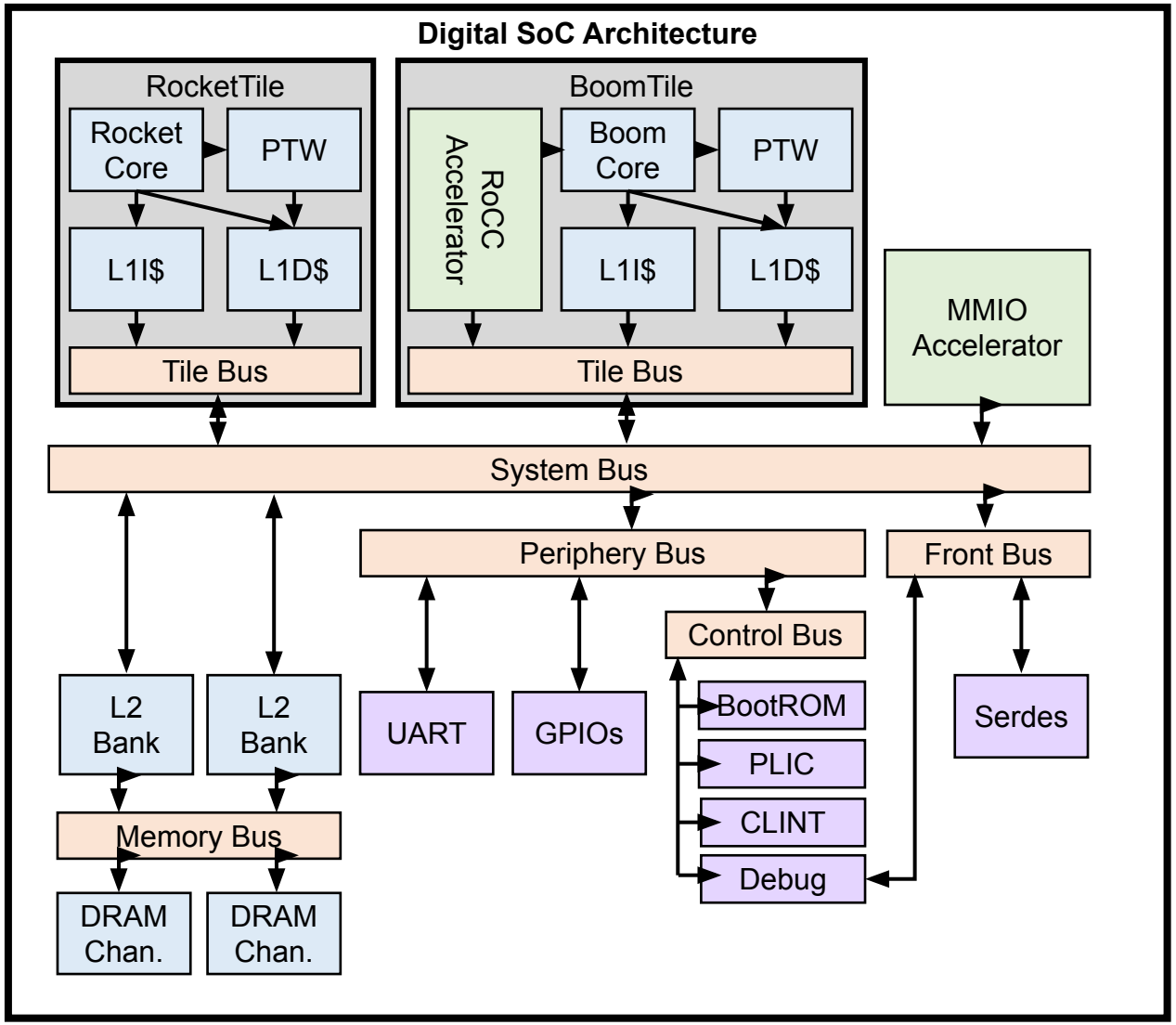


CHIPYARD Organization

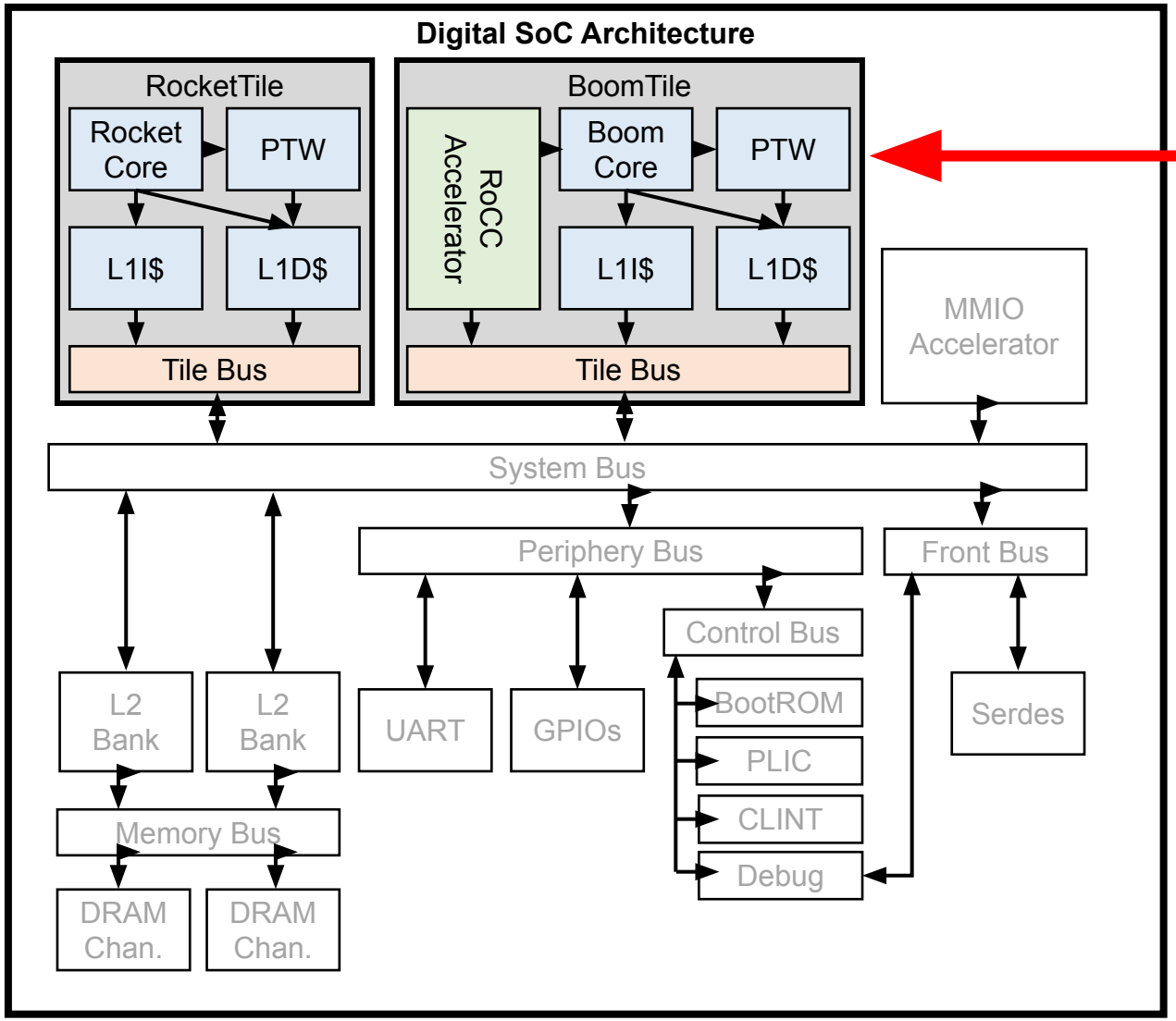
SoC architecture and generators



SoC Architecture



Tiles and Cores

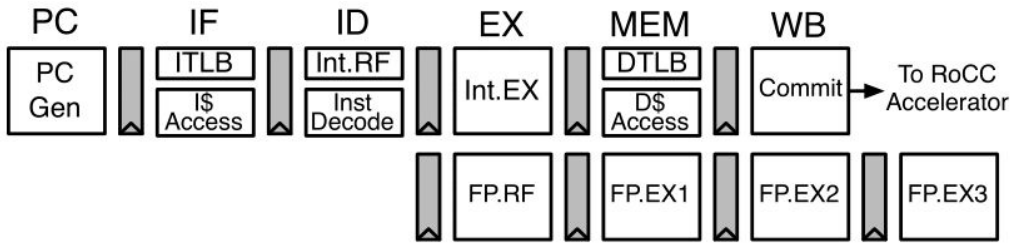


Tiles:

- Each Tile contains a RISC-V core and private caches
- Several varieties of Cores supported
- Interface supports integrating your own RISC-V core implementation


Chisel Cores

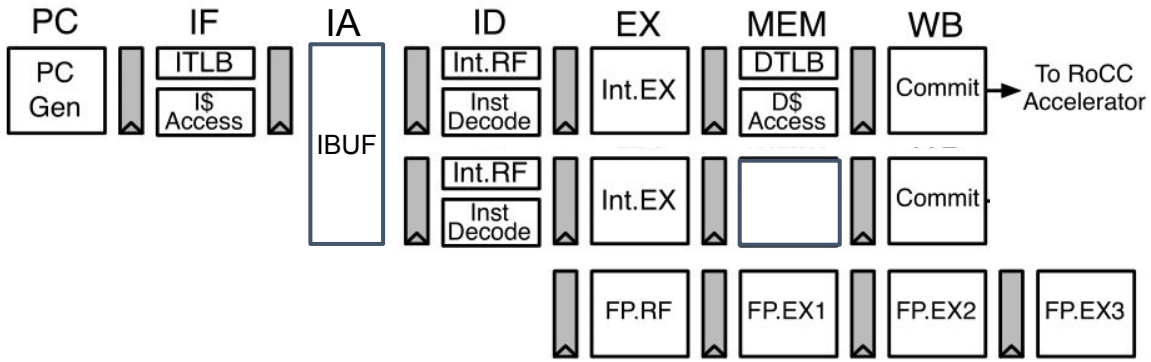
Chisel RISC-V Cores

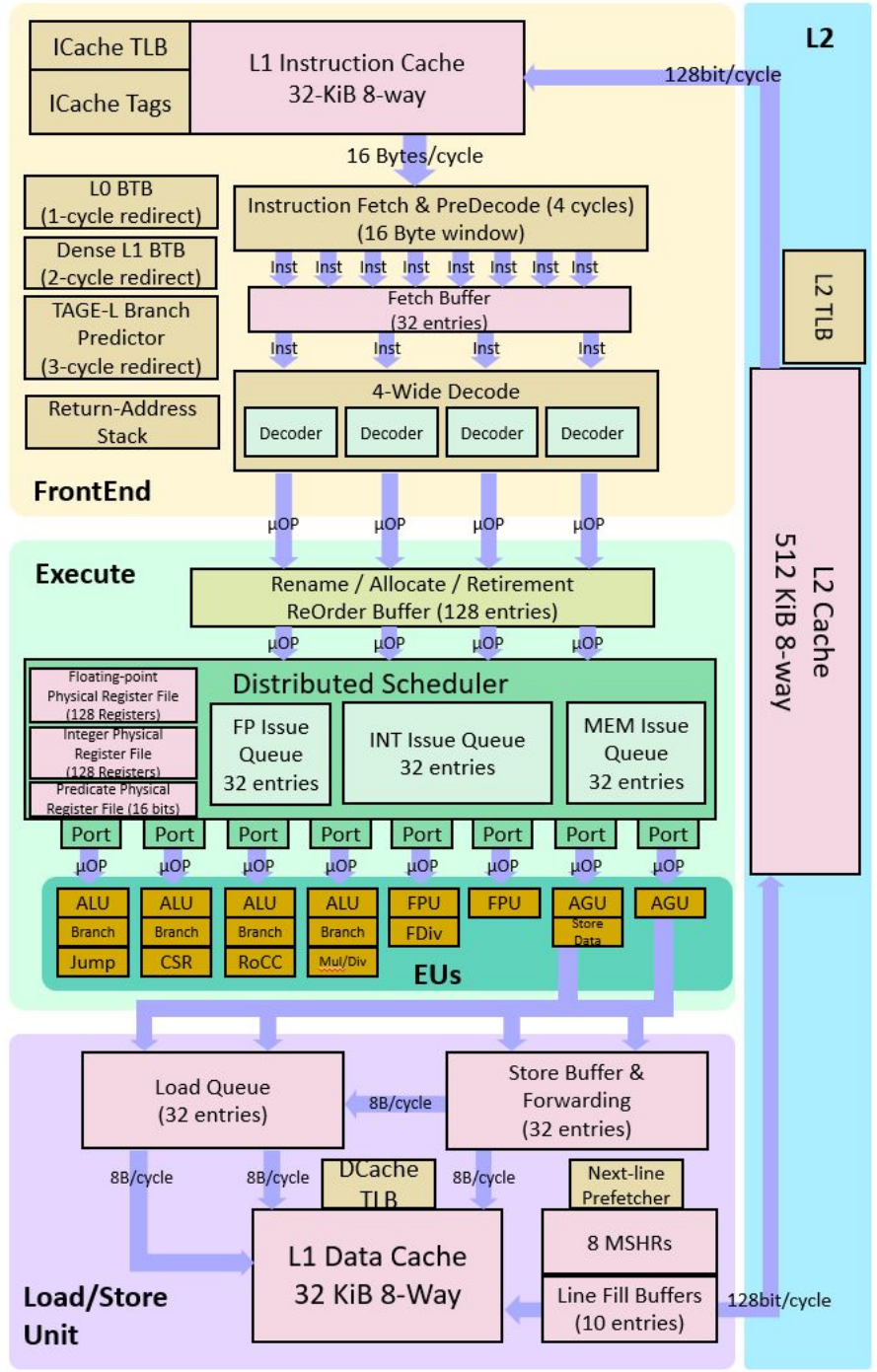


Rocket: 5-stage single-issue in-order

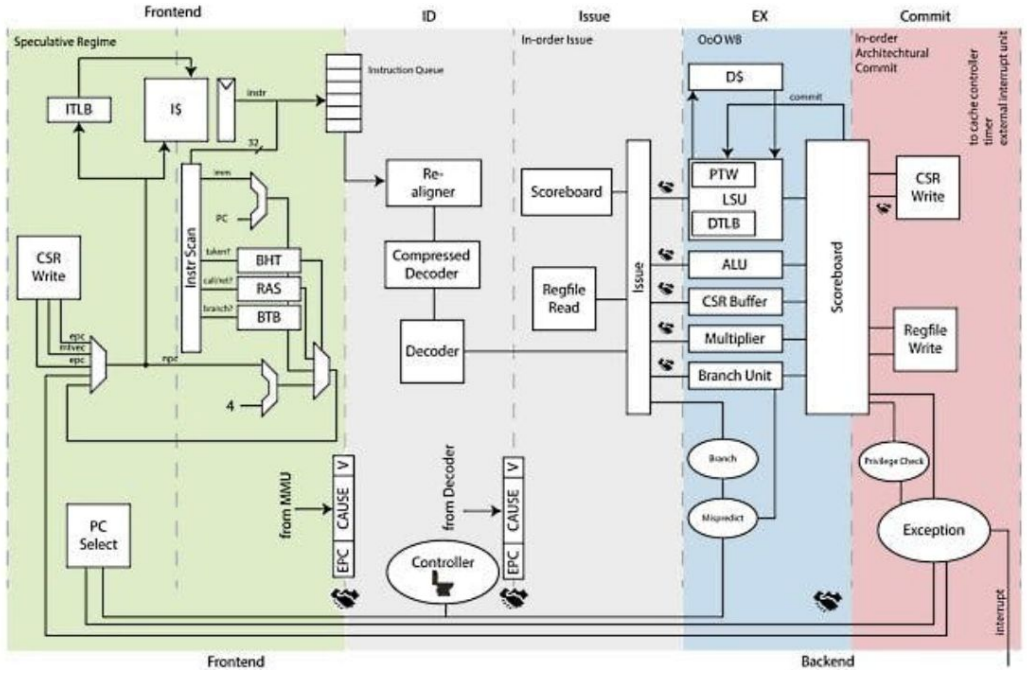
SonicBOOM: 12-stage superscalar out-of-order

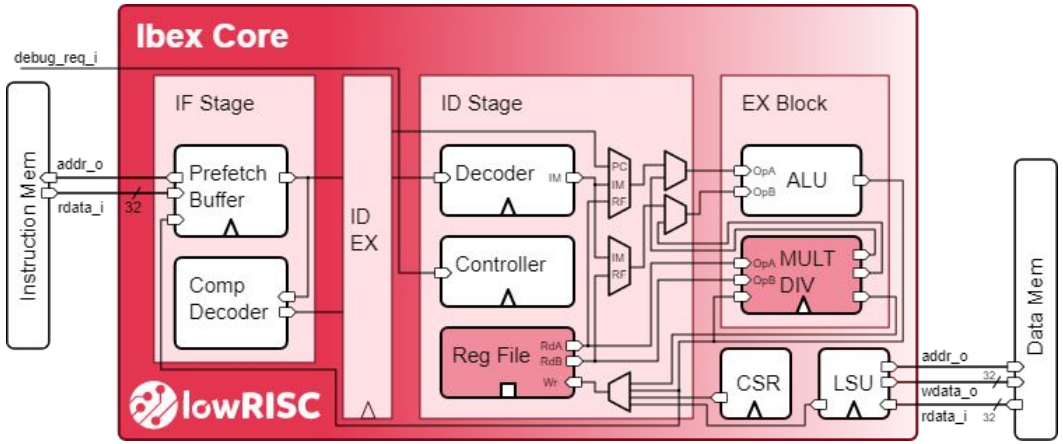
Shuttle: [NEW] 6-stage superscalar in-order





PULP Cores in CHIPYARD





CVA6 (Formerly Ariane):

- RV64IMAC 6-stage single-issue in-order core
- Open-source
- Implemented in SystemVerilog
- Developed at ETH Zurich as part of PULP,
- Now maintained by OpenHWGroup

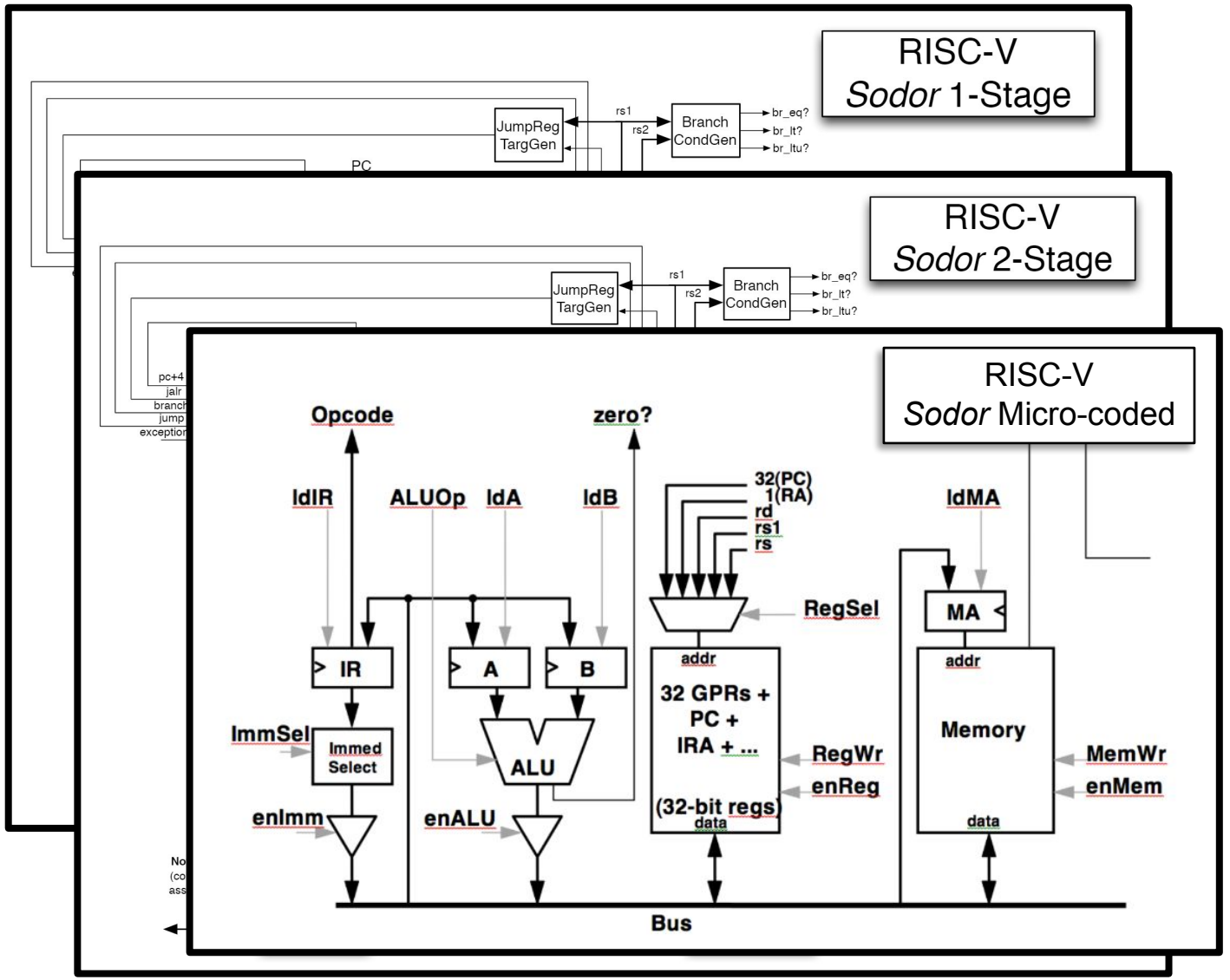
Ibex (Formerly Zero-RISCY):

- RV64IMC 2-stage single-issue in-order core
- Open-source
- Implemented in SystemVerilog
- Developed at ETH Zurich as part of PULP
- Now maintained by lowRISC

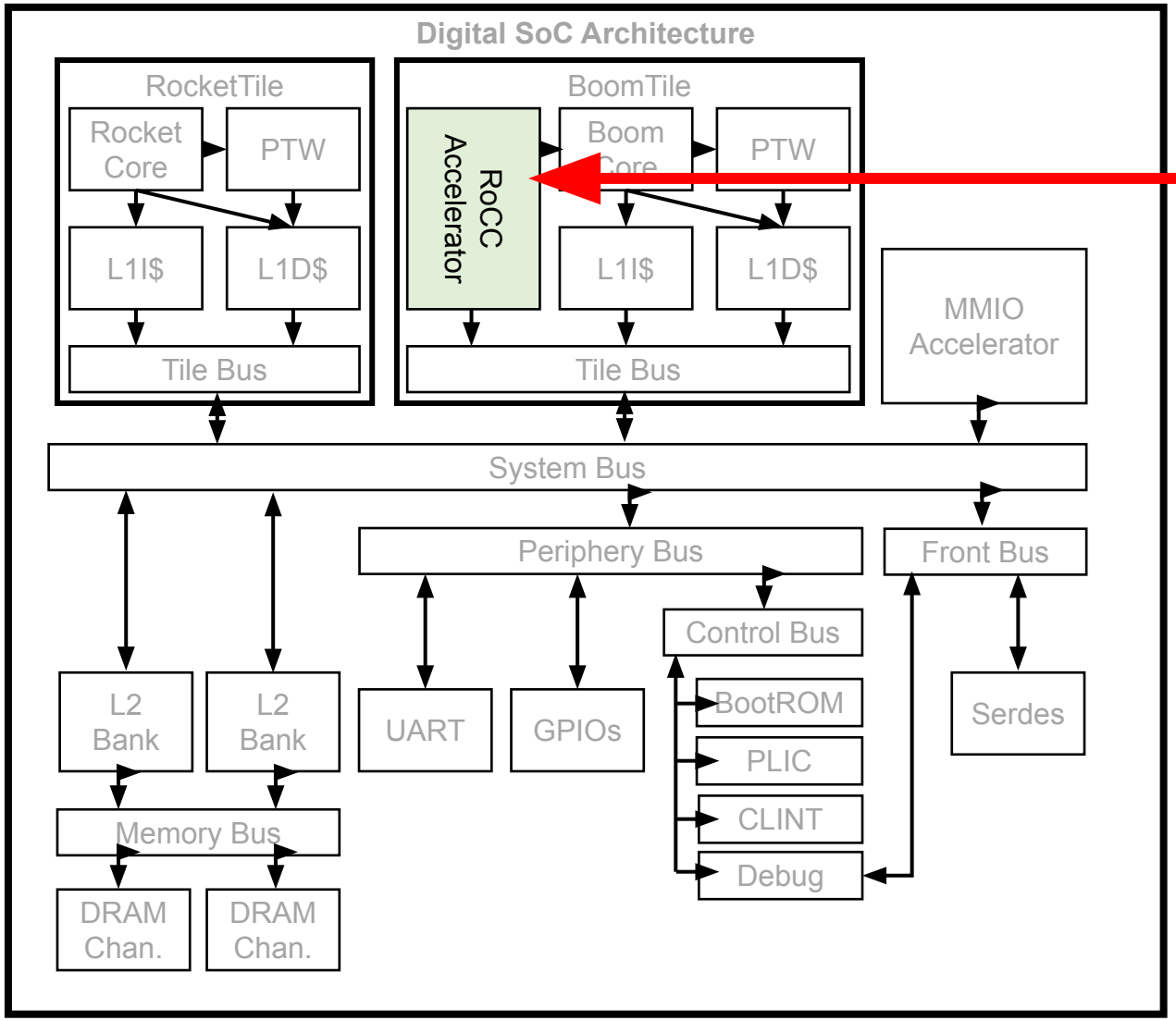
Sodor Education Cores

Sodor Core Collection

- Collection of RV32IM cores for teaching and education
- 1-stage, 2-stage, 3-stage, 5-stage implementations
- Micro-coded “bus-based” implementation
- Used in introductory computer architecture courses at Berkeley

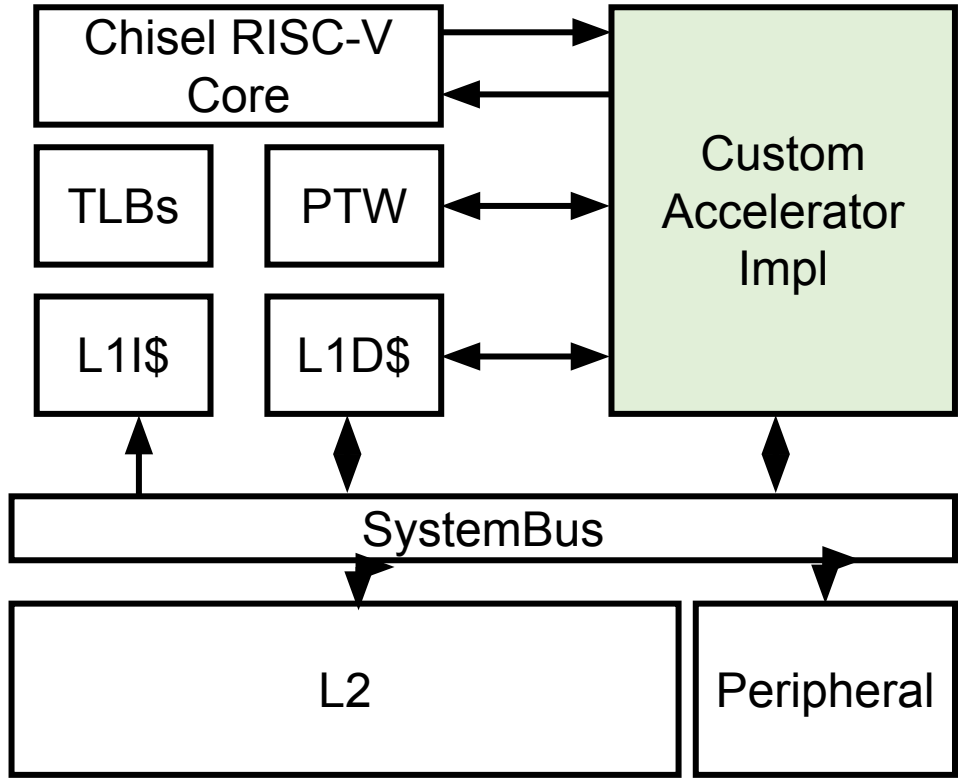


RoCC Accelerators

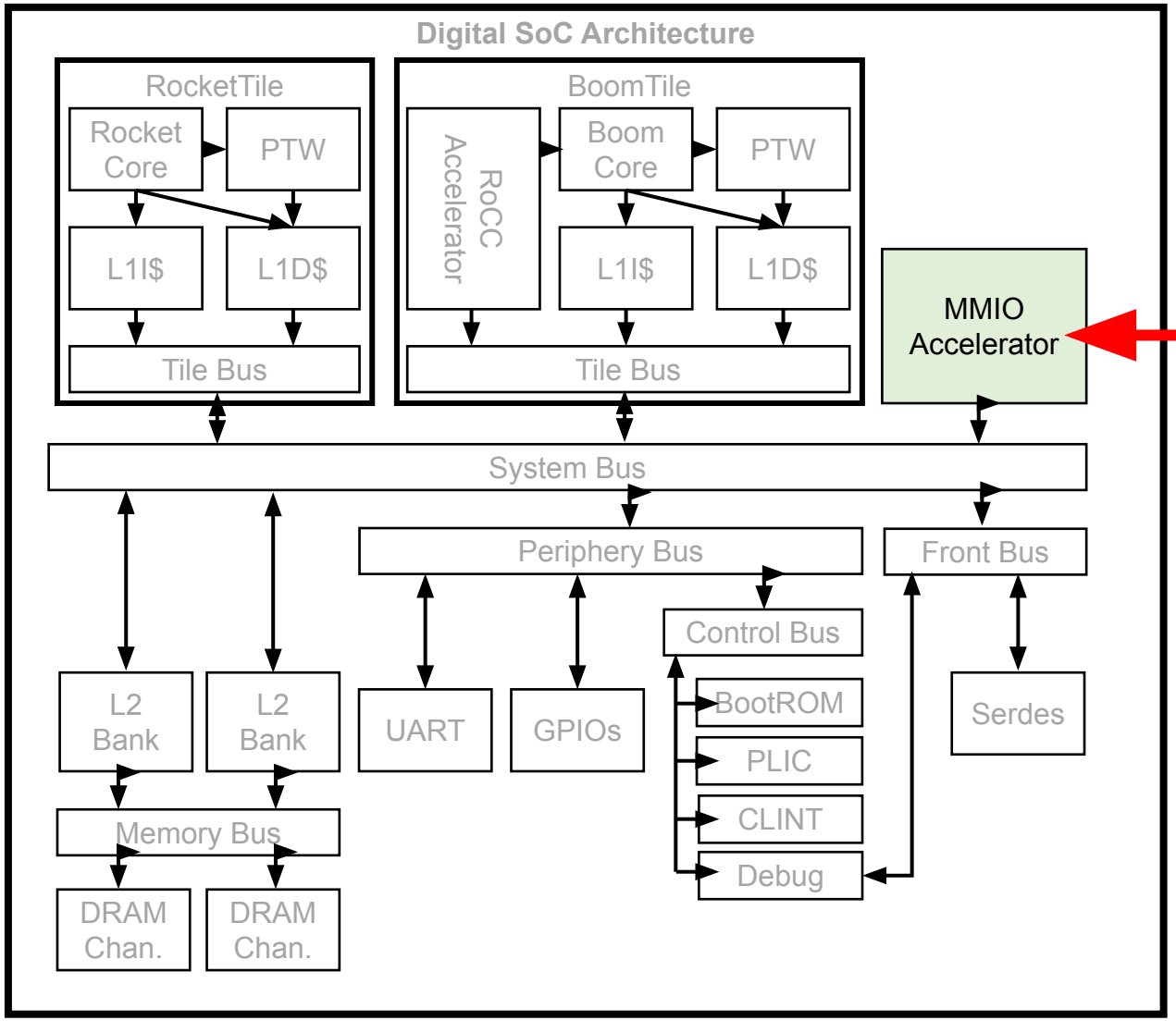


RoCC Accelerators:

- Tightly-coupled accelerator interface
- Attach custom accelerators to Rocket or BOOM cores
- Included example implementations



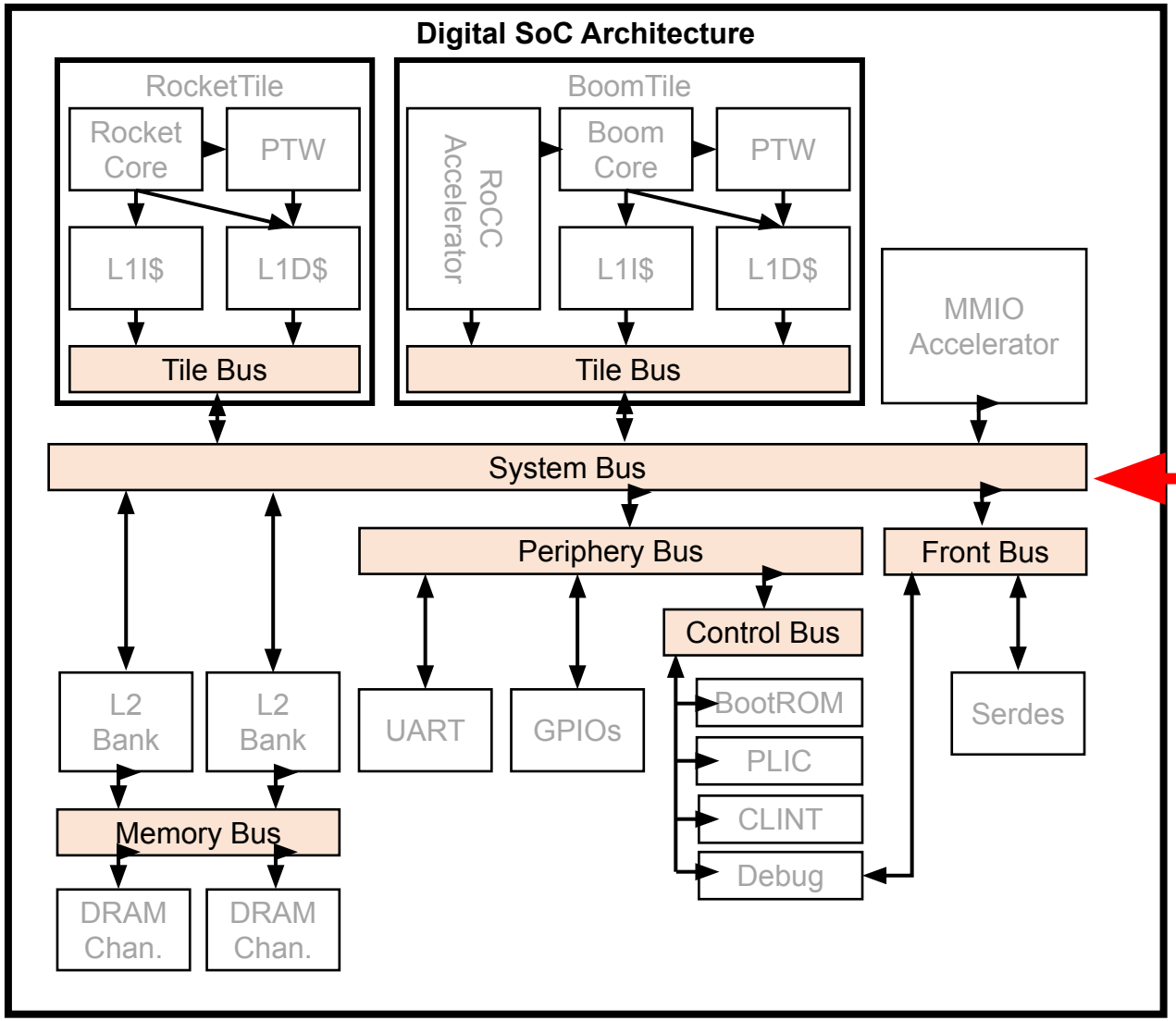
MMIO Accelerators



MMIO Accelerators:

- Controlled by MMIO-mapped registers
- Supports DMA to memory system
- Examples:
  - Nvidia NVDLA accelerator
  - FFT accelerator generator

Coherent Interconnect



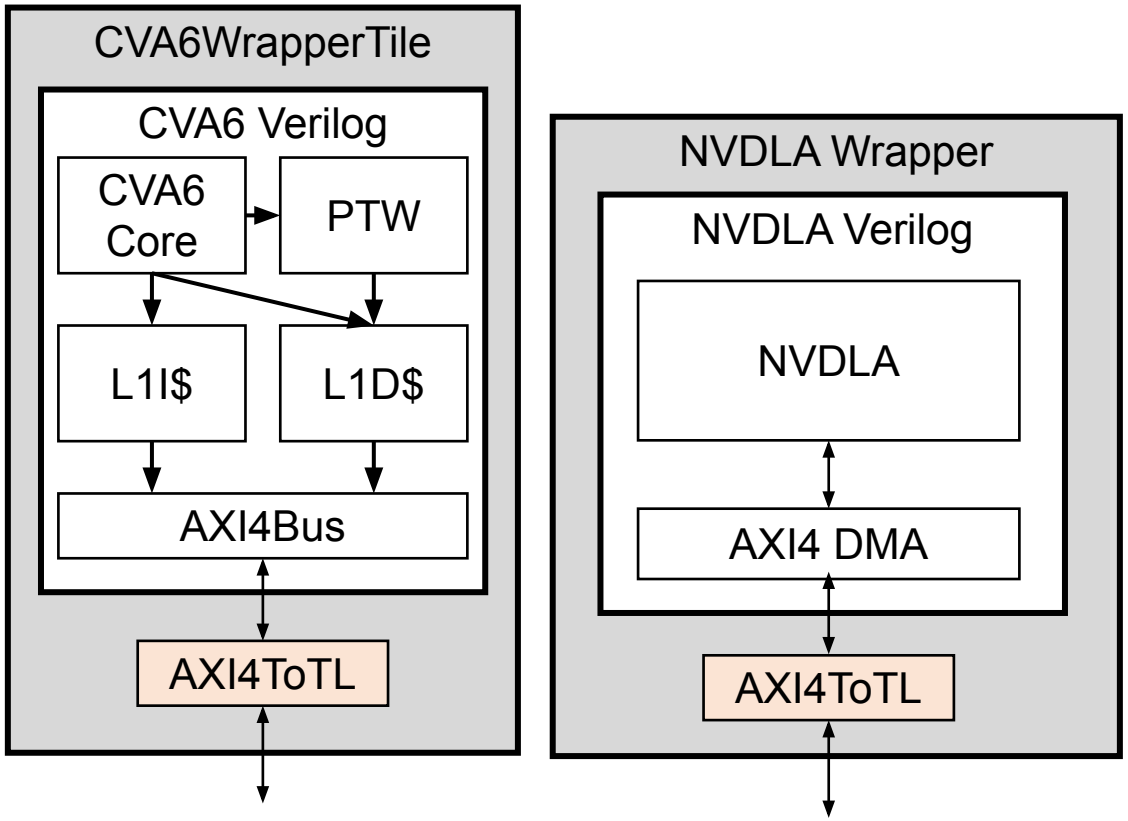
TileLink Standard:

- TileLink is open-source chip-scale interconnect standard
- Comparable to AXI/ACE
- Supports multi-core, accelerators, peripherals, DMA, etc

Interconnect IP:

- Library of TileLink RTL generators provided in RocketChip
- RTL generators for crossbar-based buses
- Width-adapters, clock-crossings, etc.
- Adapters to AXI4, APB
- **New:** Drop-in prefetchers

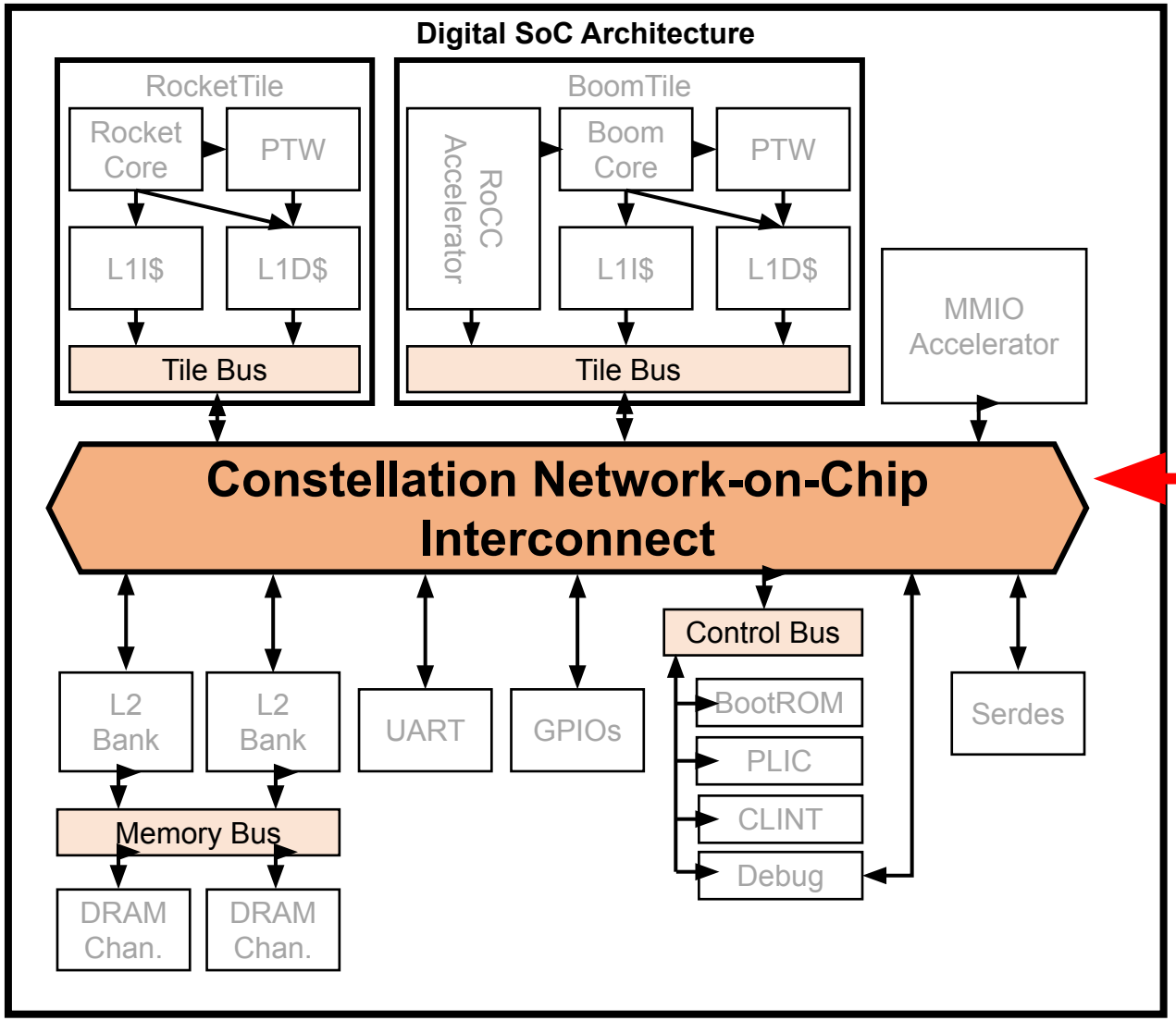
Protocol Shims



AMBA-to-TileLink shims enable easy integration with existing IP

- Works for cores/peripherals/accelerators
- Drop-in Verilog integration of CVA-6, NVDLA

NoC Interconnect



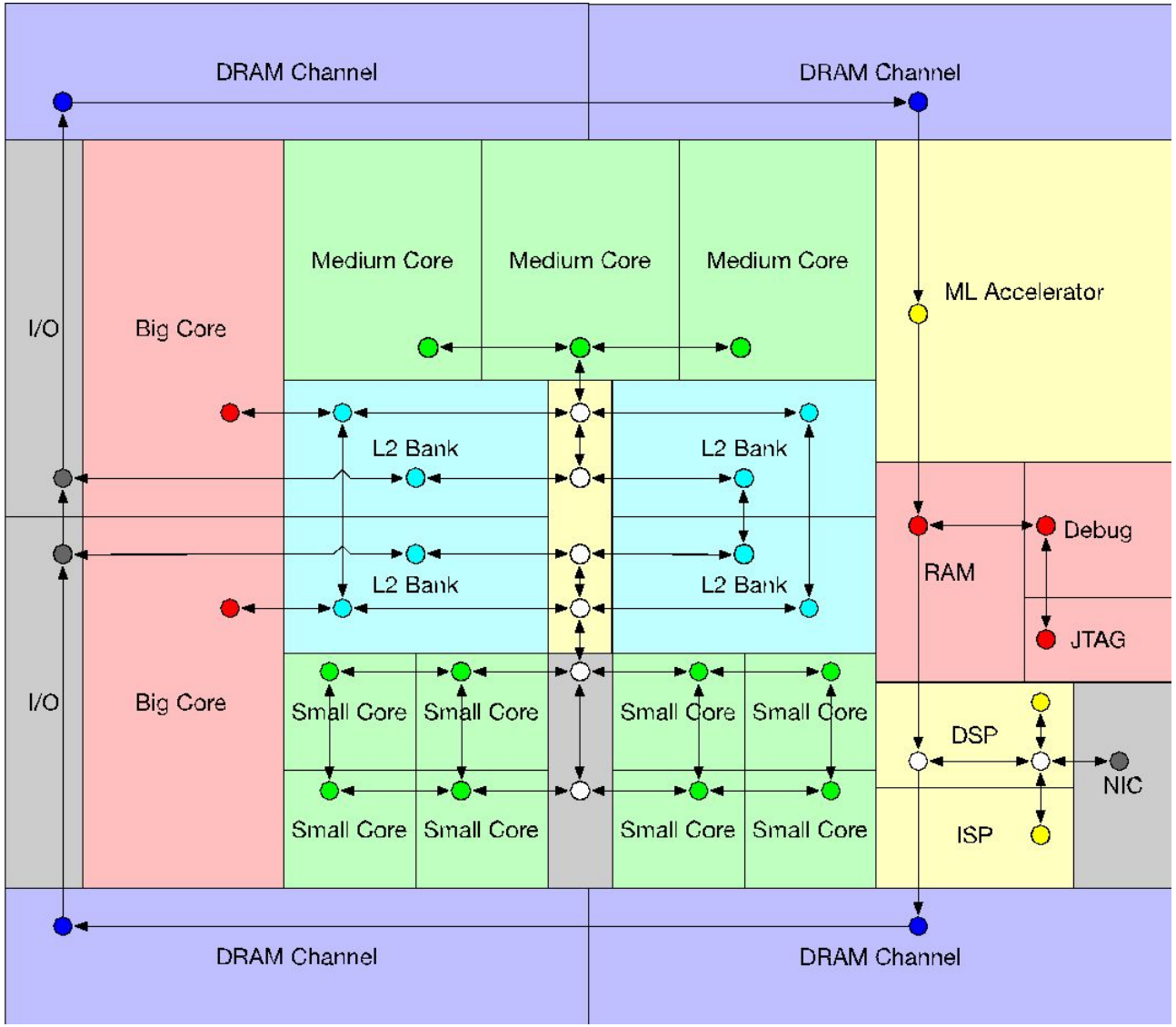
Constellation

- Drop-in replacement for TileLink crossbar buses

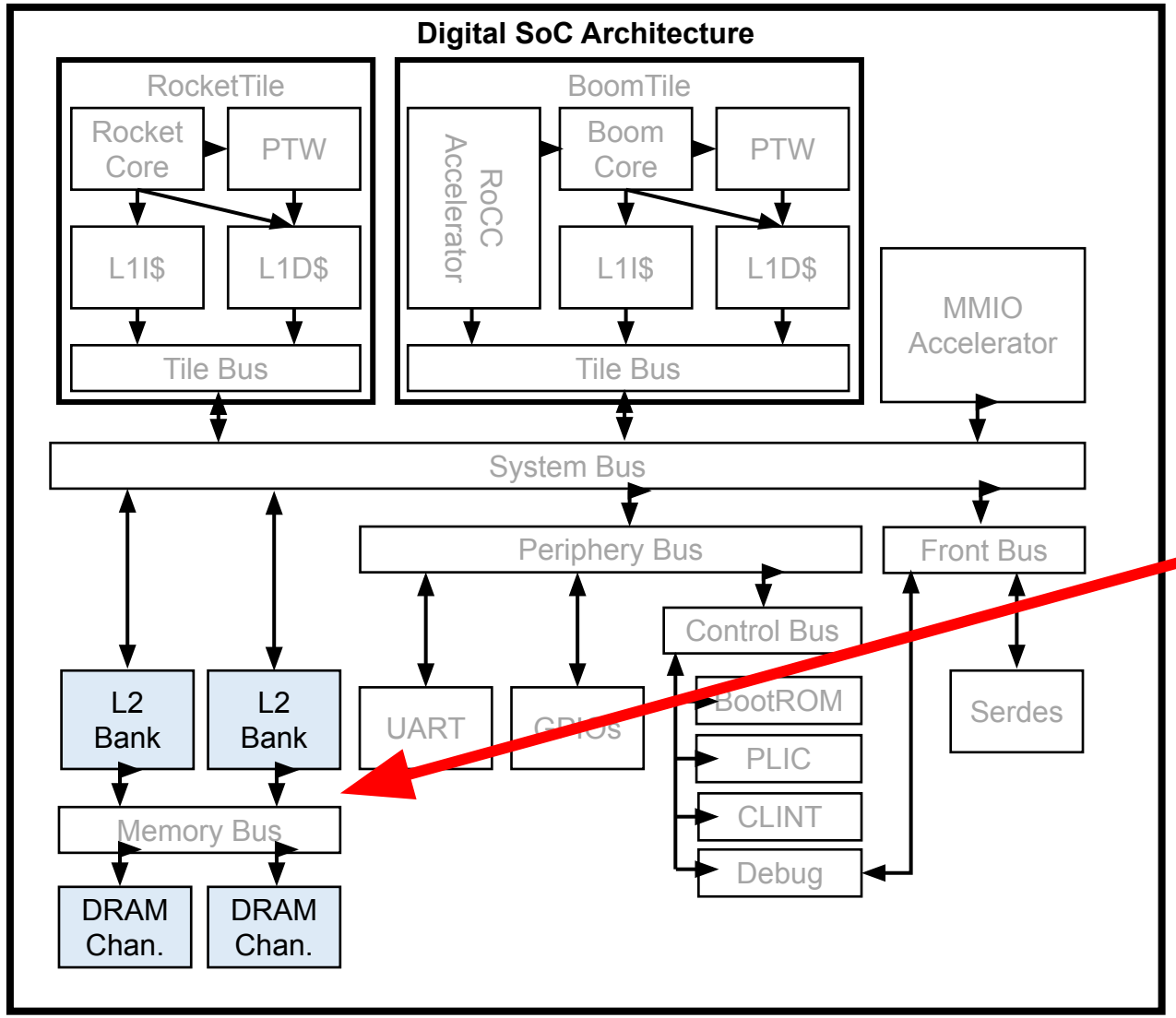
Constellation

A parameterized Chisel generator for SoC interconnects

- Protocol-independent transport layer
- Supports TileLink, AXI-4
- Highly parameterized
- Deadlock-freedom
- Virtual-channel wormhole-routing



L2/DRAM



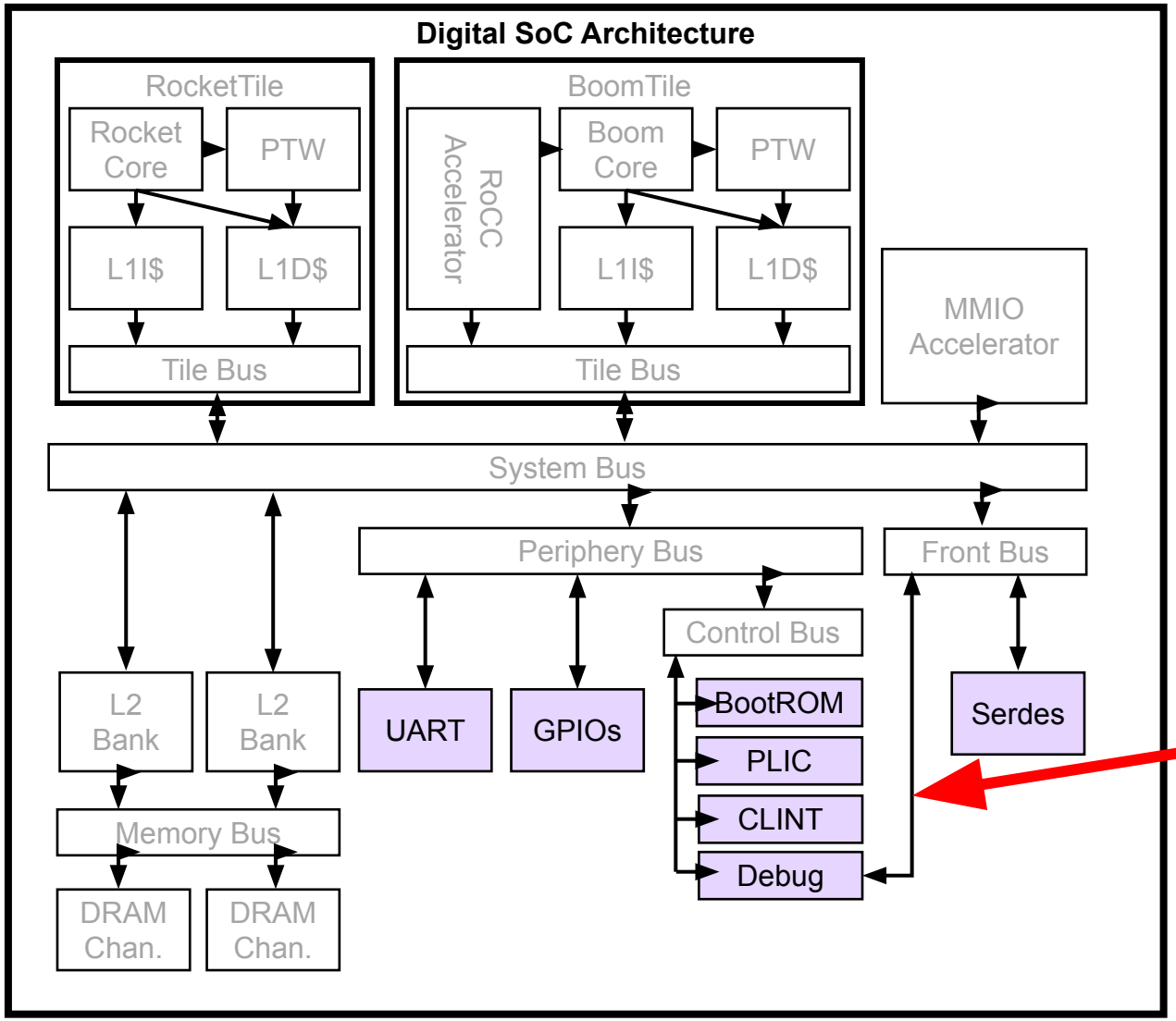
Shared memory:

- Open-source TileLink L2 developed by SiFive
  - Directory-based coherence with MOESI-like protocol
  - Configurable capacity/banking
- Support broadcast-based coherence in no-L2 systems
- Support incoherent memory systems

DRAM:

- AXI-4 DRAM interface to external memory controller
- Interfaces with DRAMSim

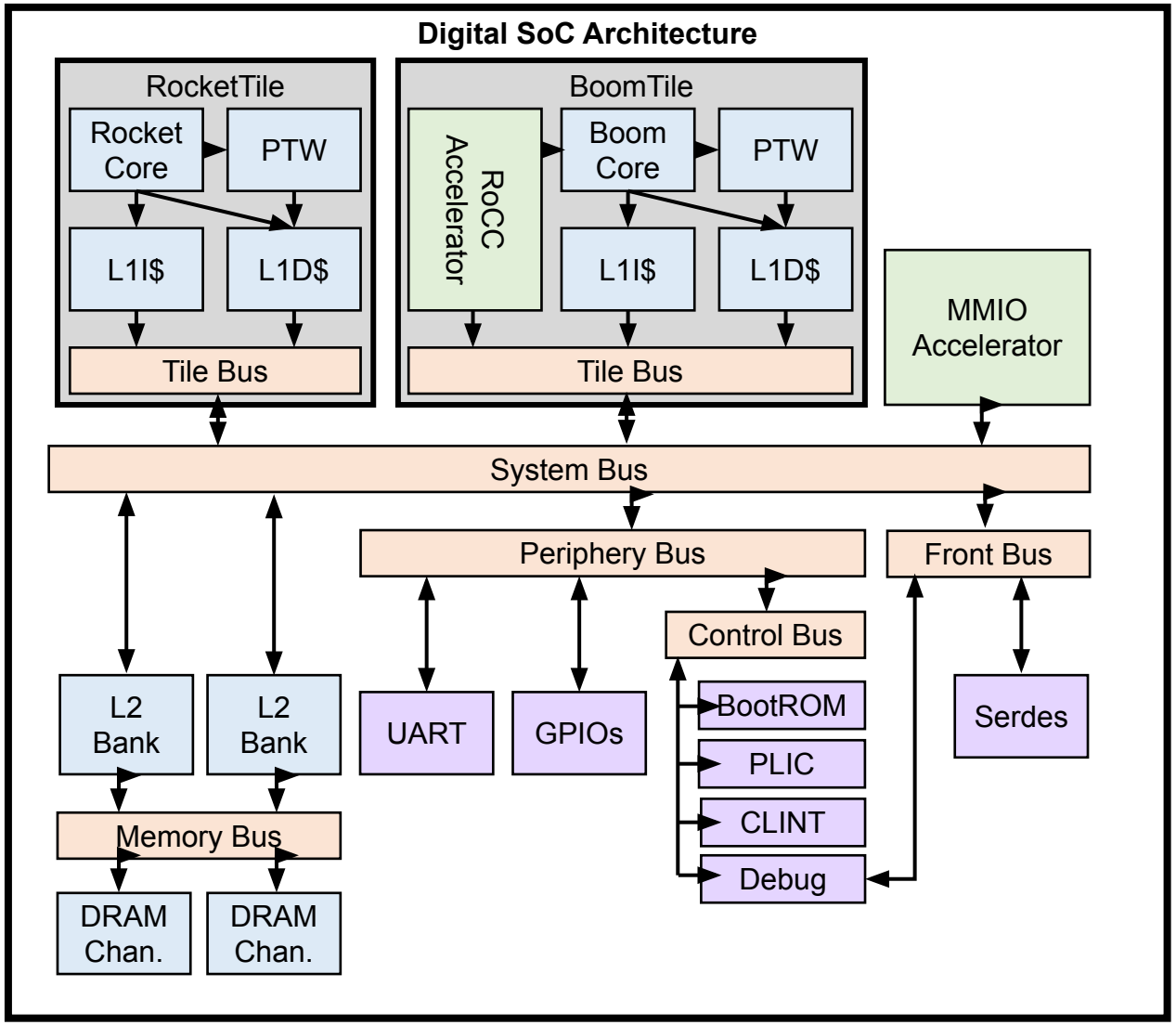
Peripherals and IO



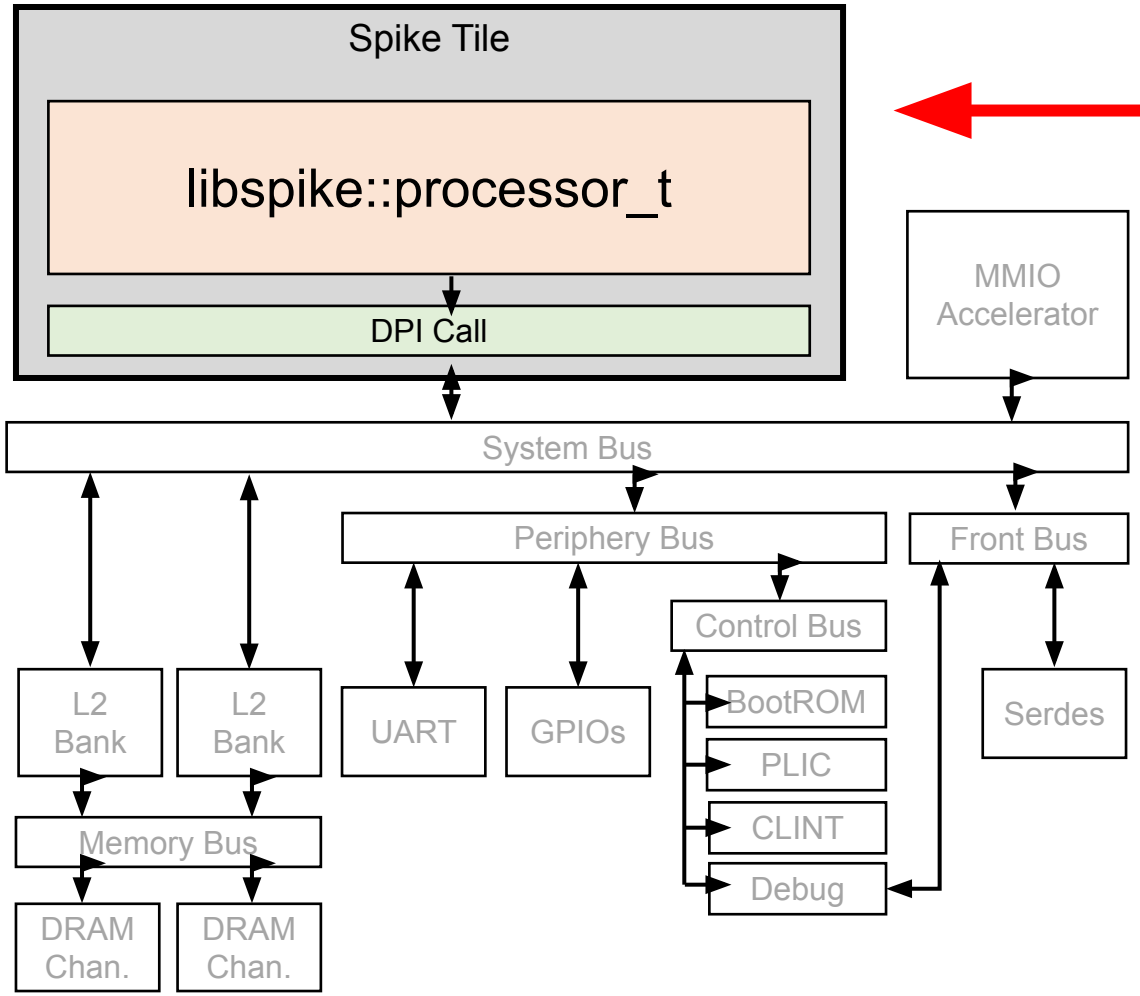
Peripherals and IO:

- Open-source RocketChip blocks
  - Interrupt controllers
  - JTAG, Debug module, BootROM
  - UART, GPIOs, SPI, I2C, PWM, etc.
- TestChipIP: useful IP for test chips
  - Clock-management devices
  - SerDes
  - Scratchpads

SoC Architecture



Spike Integration



Spike-as-a-Tile:

- DPI interface between RTL SoC simulation and SoC functional model
- Spike “virtual platform”
- Enables testing of complex device software in RTL simulation

Spike Integration

Spike: original fast C++ RISC-V functional model

Spike-as-a-Tile (Spike Virtual Platform):

- RTL-implemented memory system driven by Spike CPU model
- Useful for debugging/prototyping software for custom devices

Spike-Cosimulation:

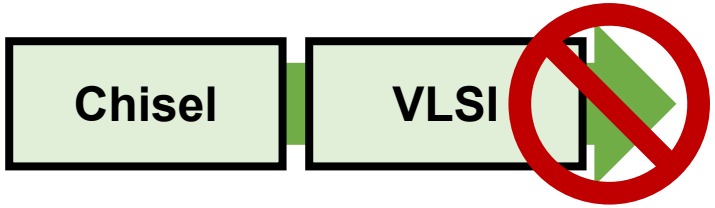
- Lock-step commit-trace checking of a RTL core against Spike reference model
- Useful for debugging cores

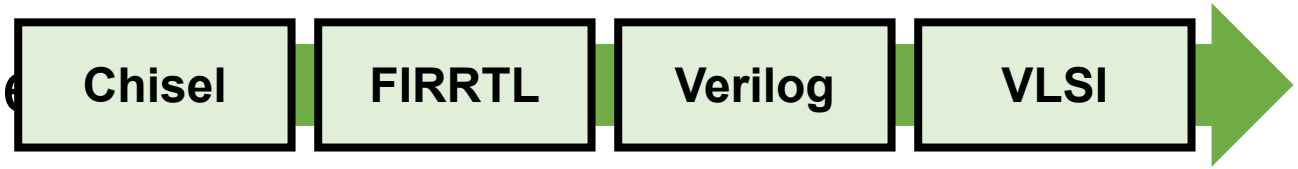
Spike-driven restartable architectural checkpointing:

- Generate a checkpoint in Spike, replay it in RTL simulation
- Useful for fast-forwarding to interesting regions of long programs

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

```
// Generalized FIR filter parameterized by coefficients
class FirFilter(bitWidth: Int, coeffs: Seq[Int]) extends M
  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(_ + _)
}
```

Flexible parameters:
- Enables development of highly flexible, parameterized HW generators

HDL, not HLS:
- Designers reason about wires, registers, gates, IO, etc.
- Familiar Wire, Reg, IO constructs makes Chisel beginner-friendly

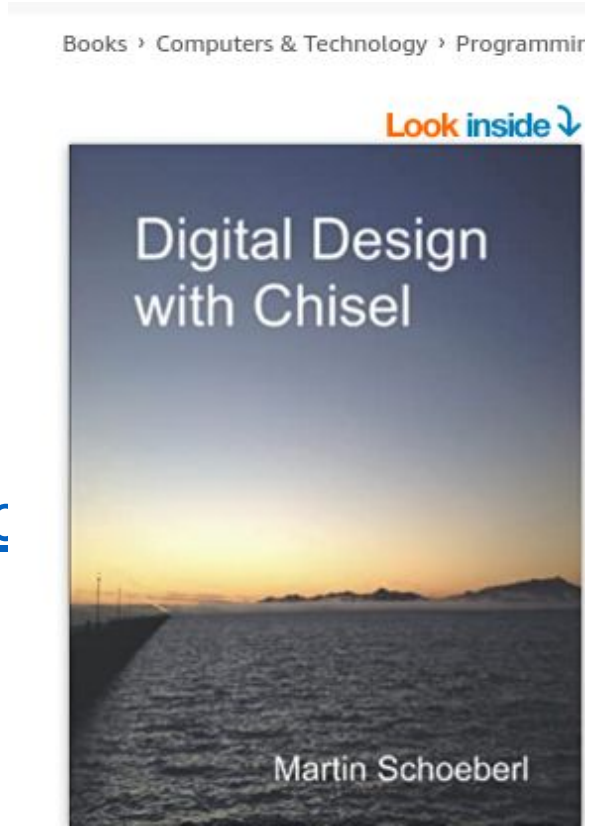
Designer-friendly features
- Powerful OOP/functional programming paradigms
- Strict type-checking encourages “correct-by-construction” design

Learning Chisel

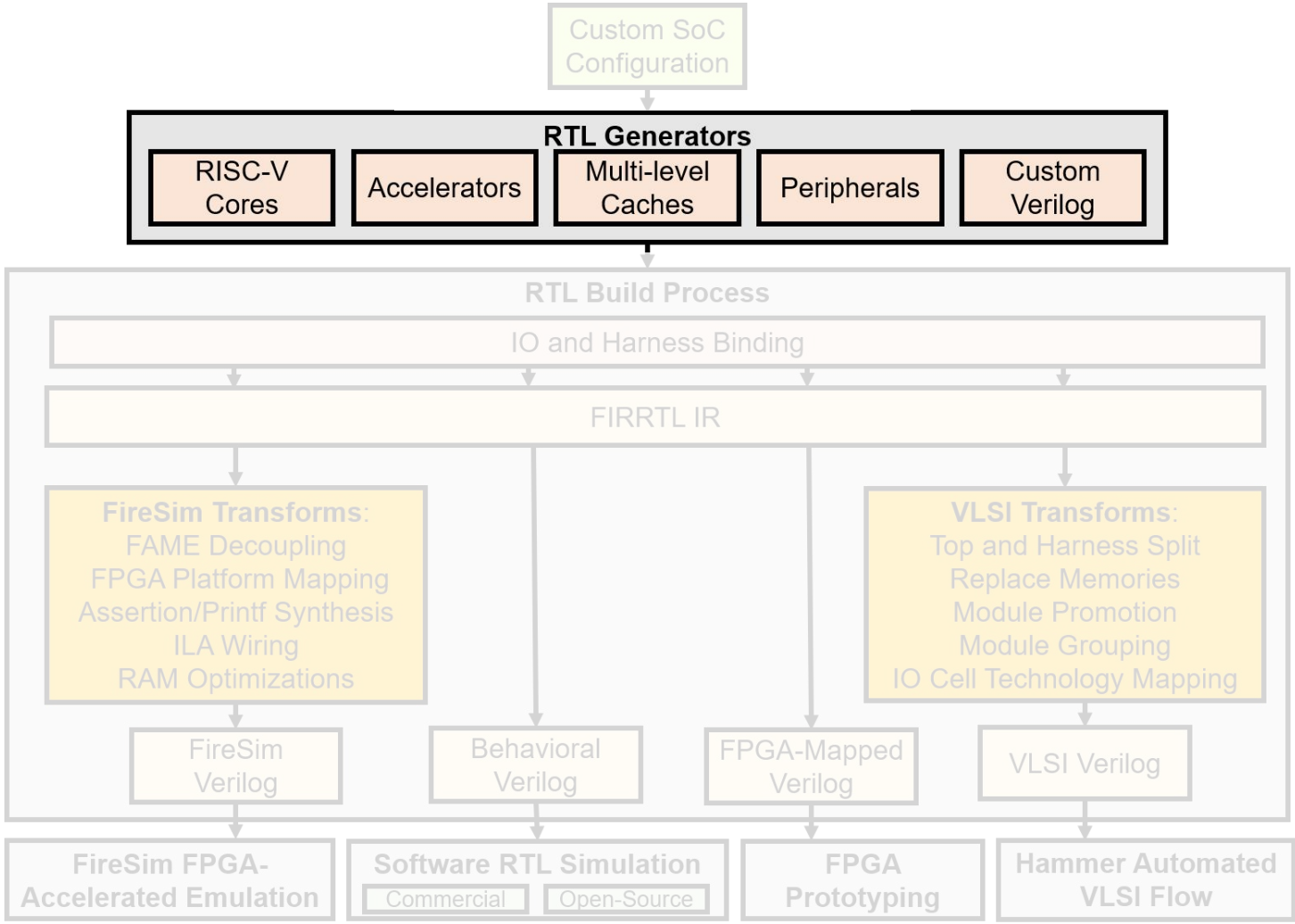
This tutorial does not expect any familiarity of Chisel

Many resources for learning Chisel

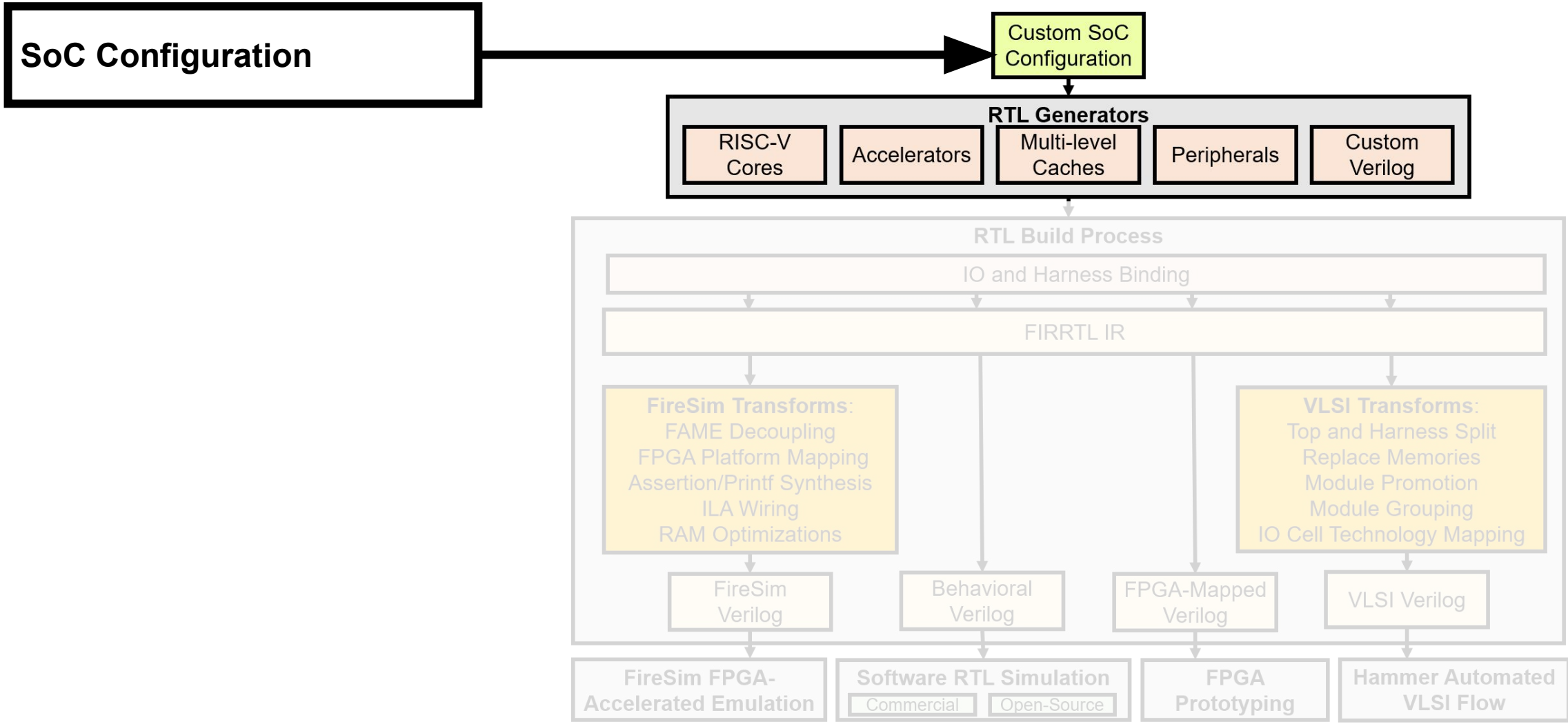
- Step-by-step tutorial: https://github.com/ucb-bar/chisel-tutorial
- Jupyter notebook: https://github.com/freechipsproject/chisel-bootcamp
- Chisel Textbook: https://www.imm.dtu.dk/~masca/chisel-book.html



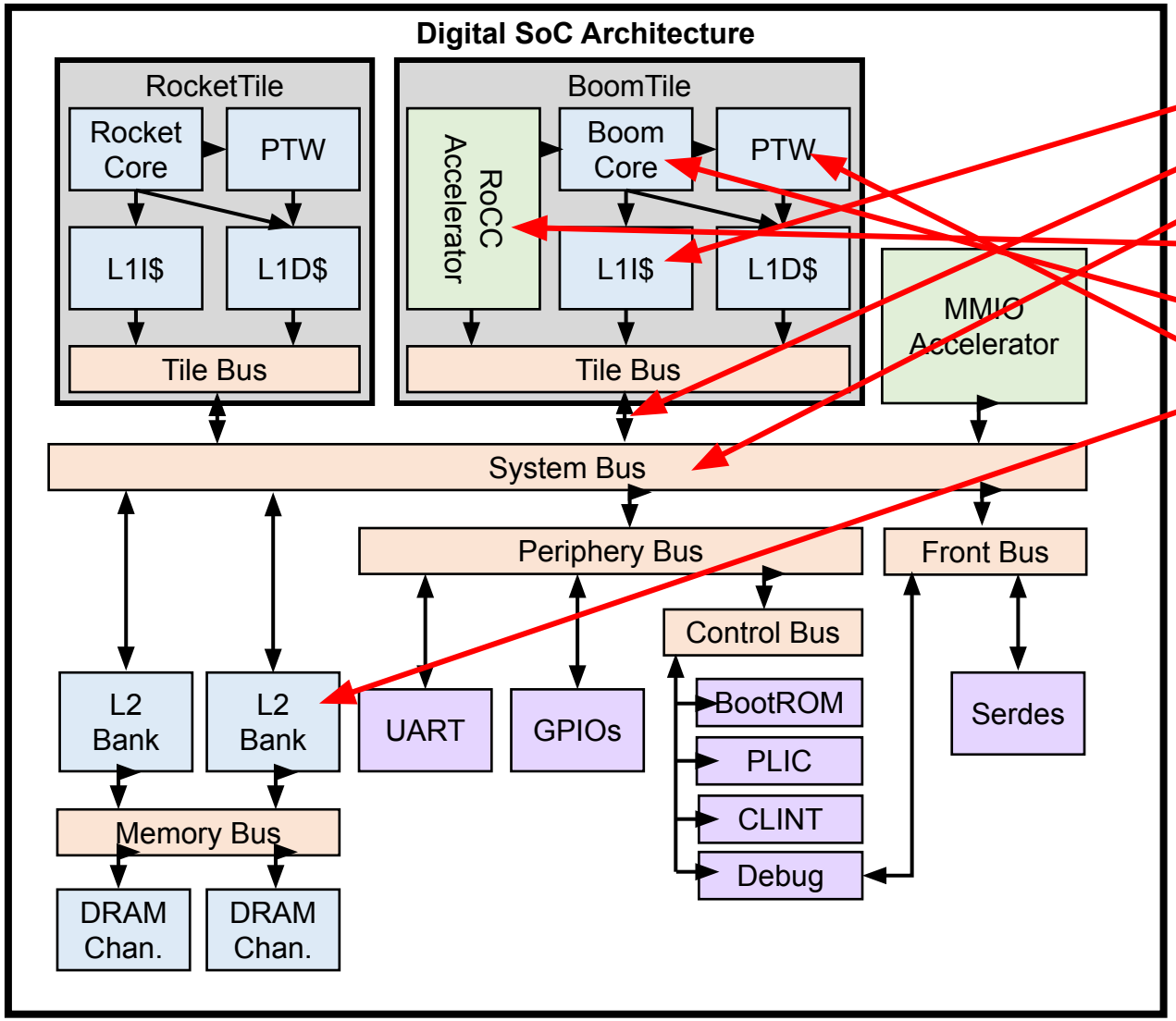

CHIPYARD Organization



CHIPYARD Organization

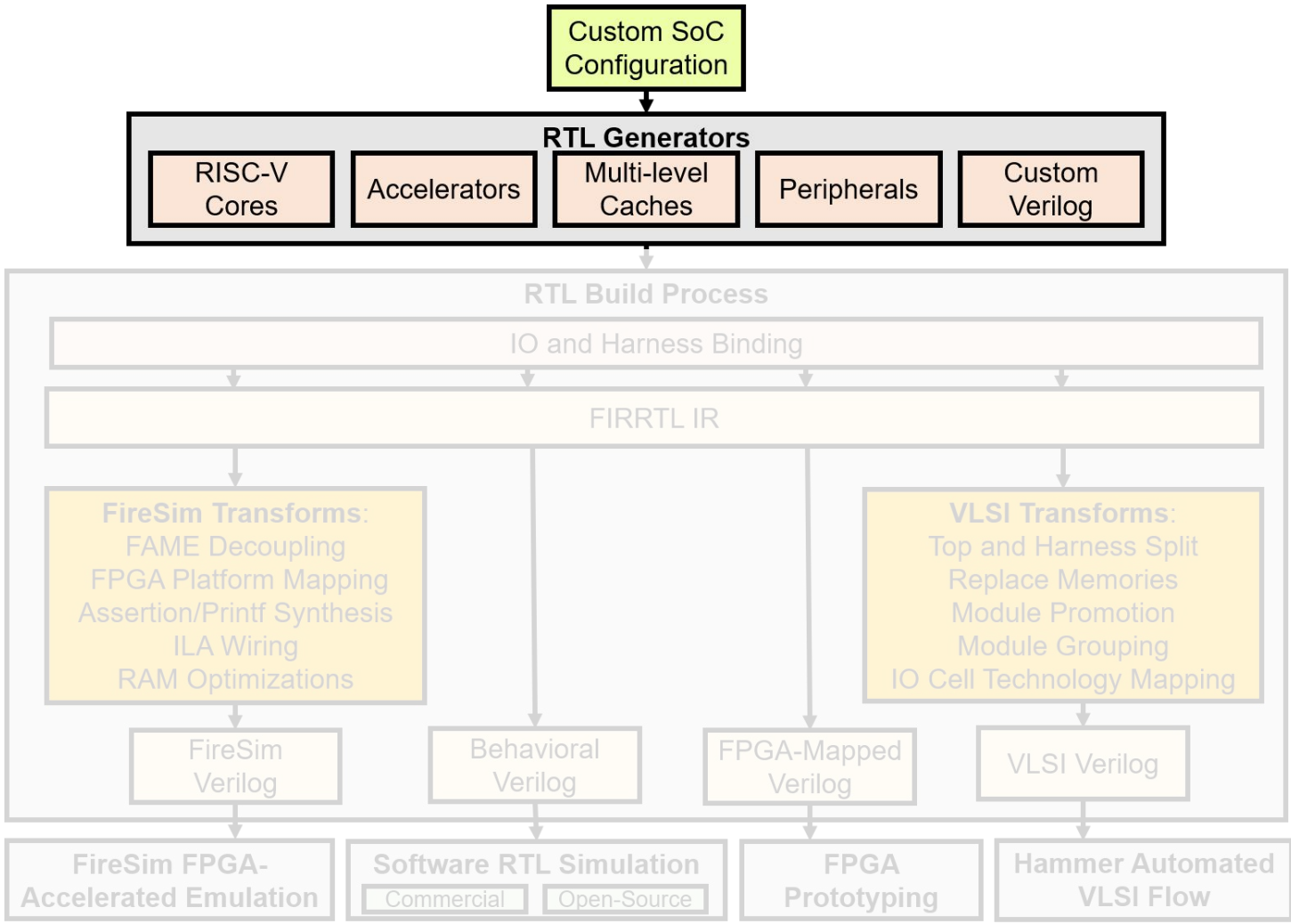


Composable Configurations




```
class CustomConfig extends Config(
  new WithL1CacheWays(4) ++
  new WithAsyncTiles ++
  new WithSystemBusWidth(128) +
  new WithFPGemmini ++
  new With3WideBooms ++
  new WithL2TLBs(512) ++
  new WithL2Sets(1024) ++

  new WithDefaultGemmini ++
  new WithNRocketCores(1) ++
  new WithNBoomCores(1) ++
  new WithBootROM ++
  new WithUART ++
  new WithJtagDTM ++
  new WithGPIOs ++
  new WithInclusiveCache(512) ++
)
```

CHIPYARD Organization



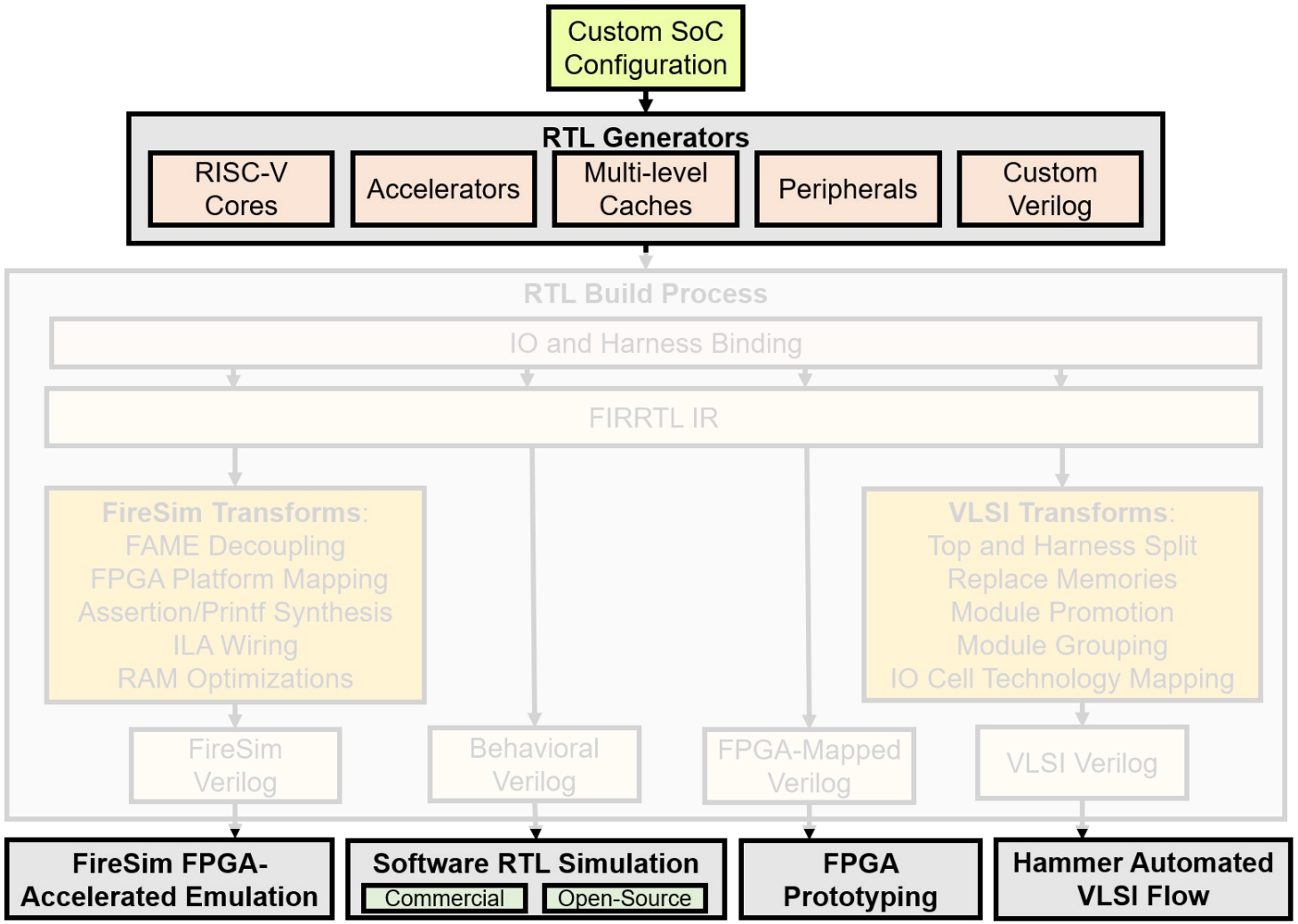
CHIPYARD Organization

SW RTL Simulation:
- RTL-level simulation with Verilator or VCS
- Hands-on tutorial next

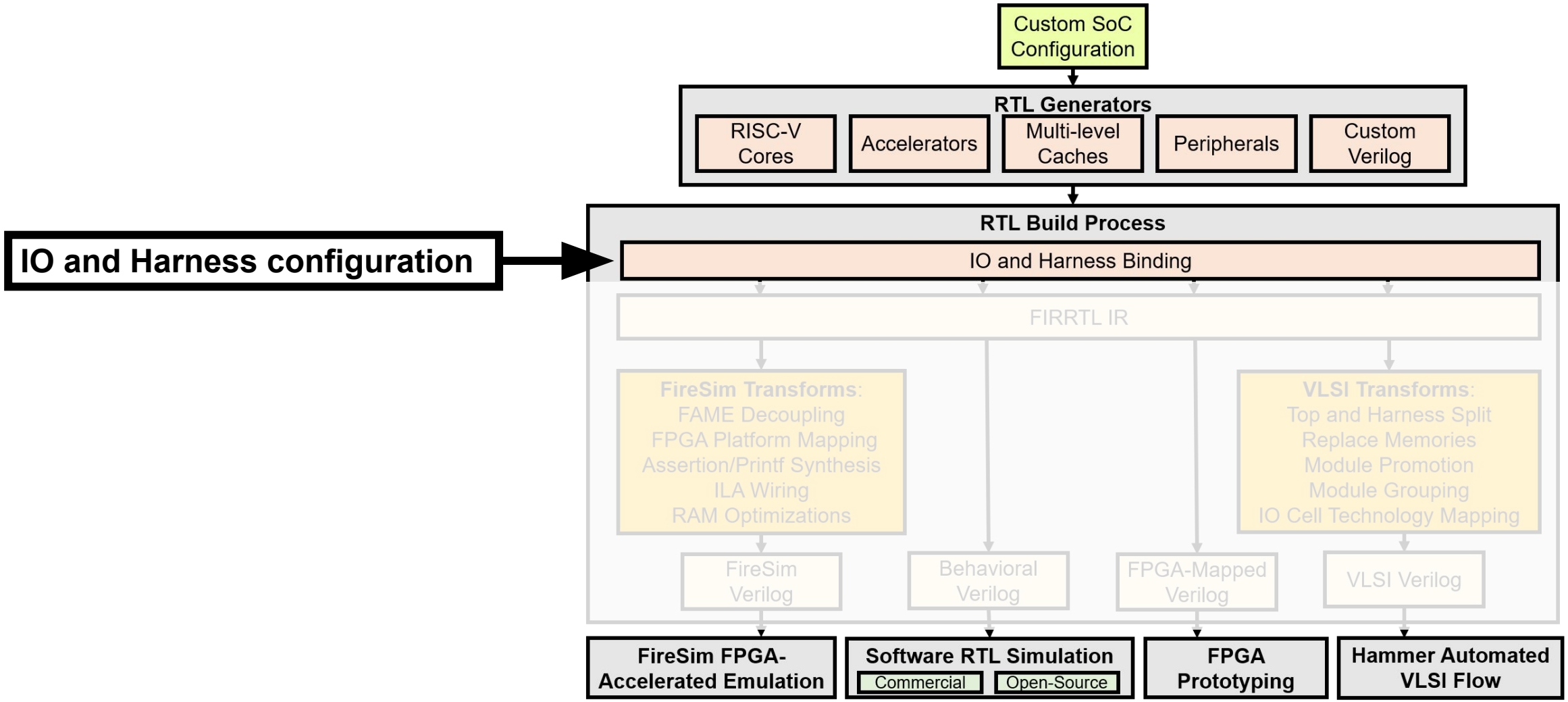
FPGA prototyping:
- Fast, non-deterministic prototypes
- Bringup platform for taped-out chips

Hammer VLSI flow:
- Tapeout a custom config in some process technology
- Overview of flow later

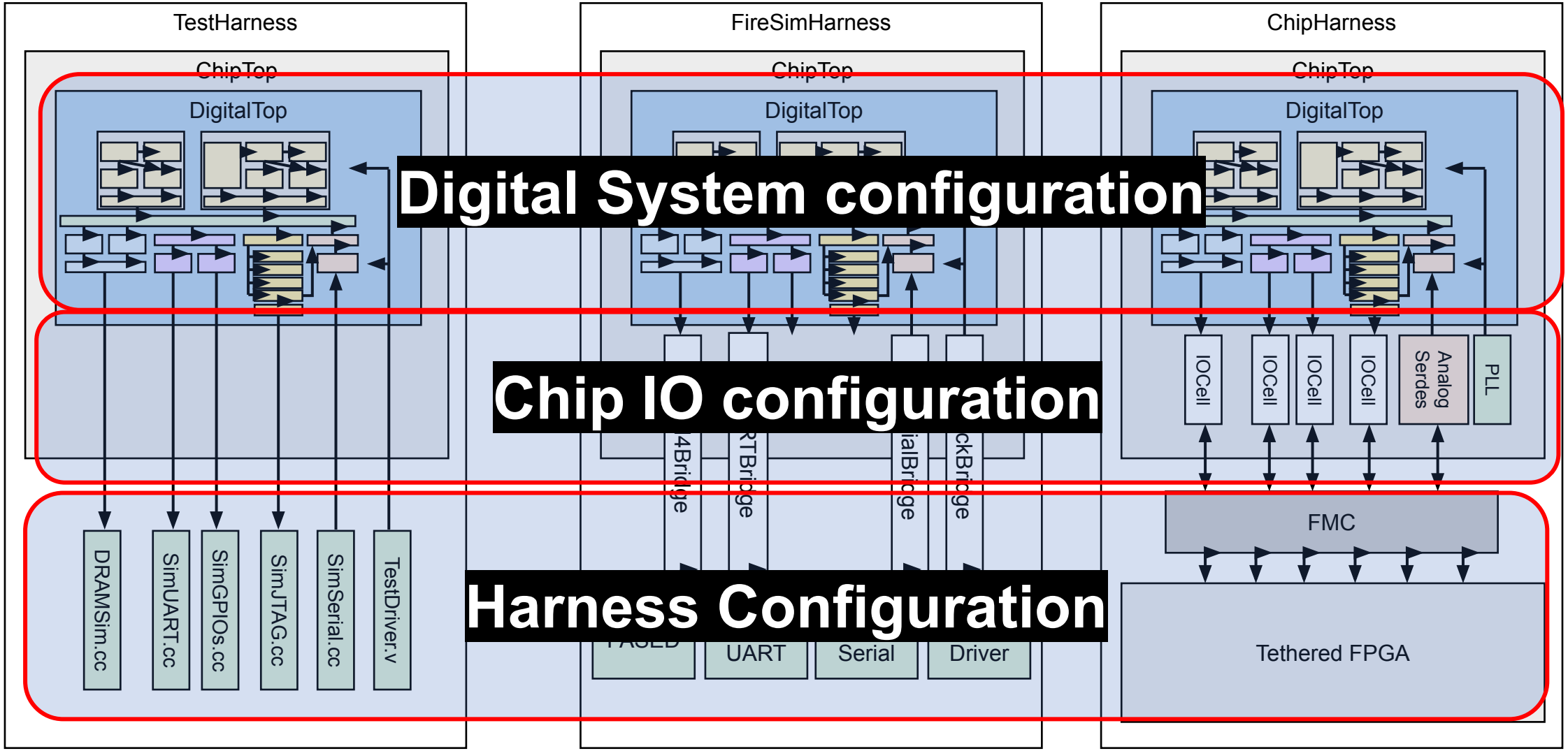
FireSim:
- Fast, accurate FPGA-accelerated simulations
- Hands-on tutorial later



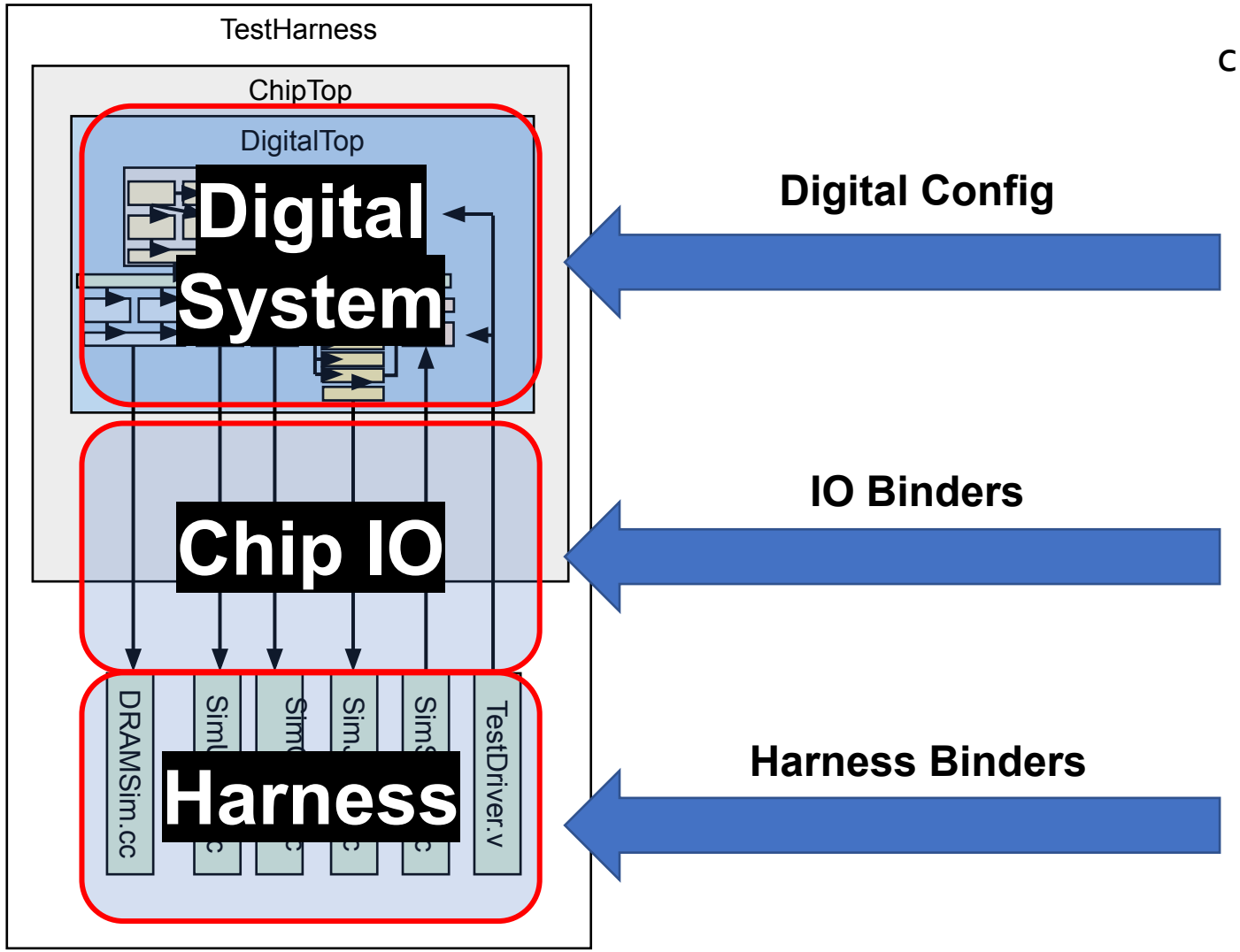

CHIPYARD Organization



Multipurpose

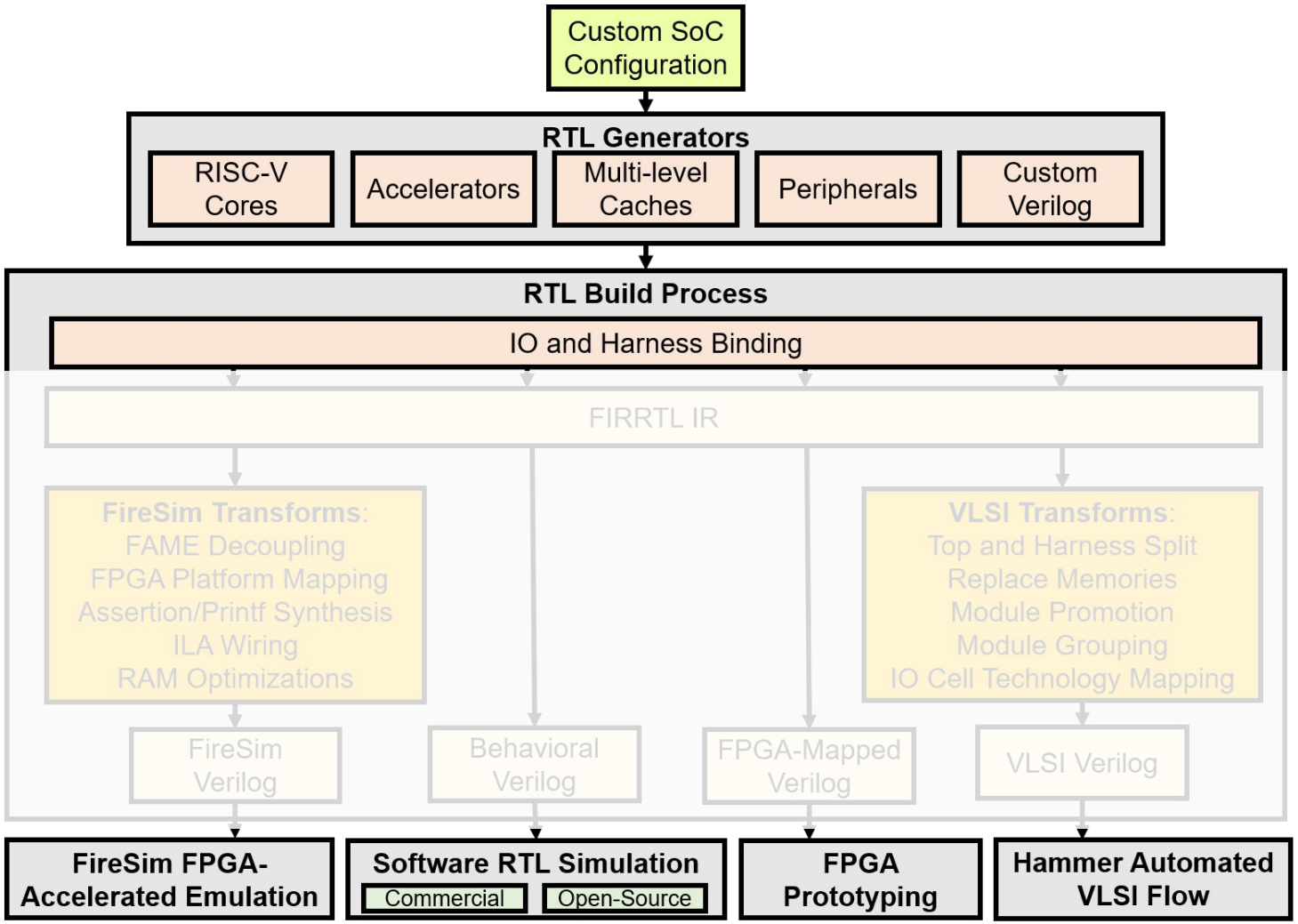


Configuring IO + Harness

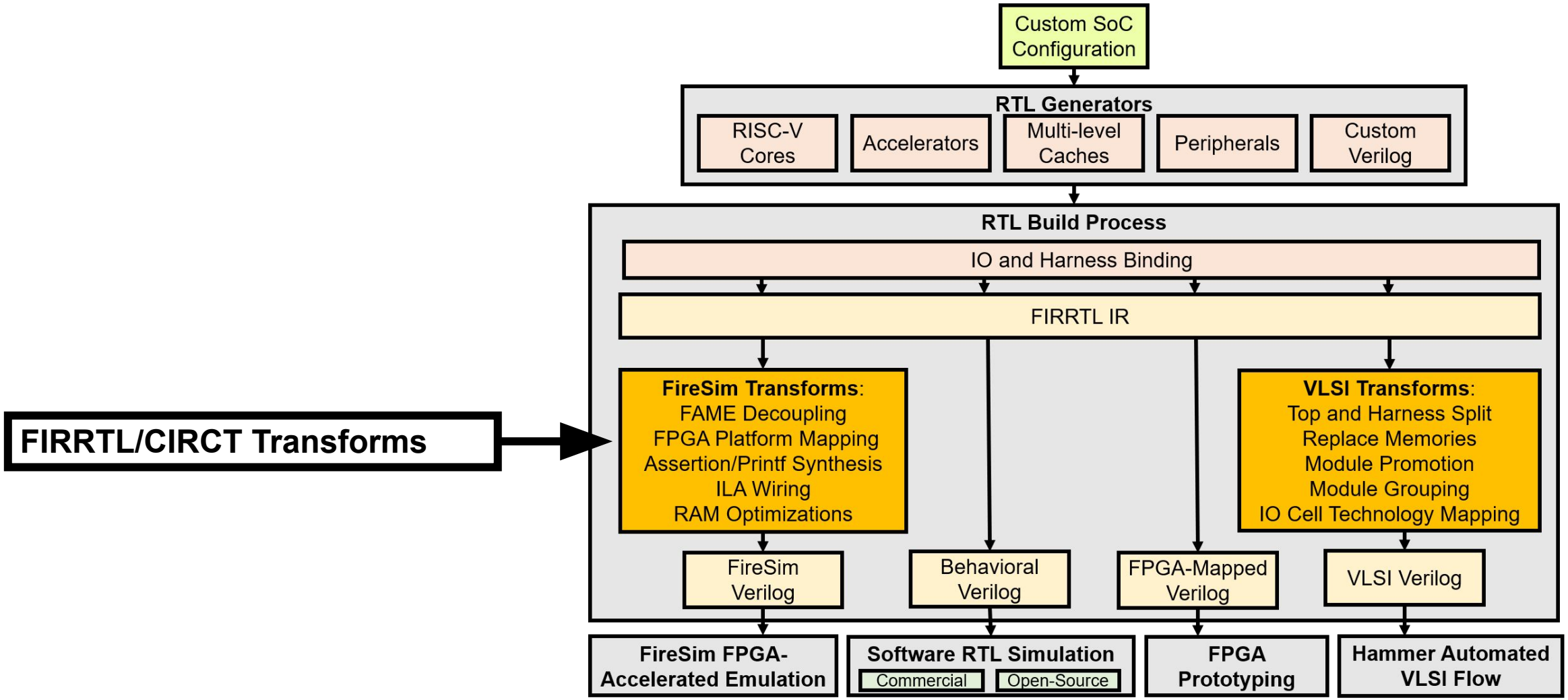



```
class CustomConfig extends Config(
  new WithDefaultGemmini ++
  new WithNRocketCores(1) ++
  new WithNBoomCores(1) ++
  new WithBootROM ++
  new WithUART ++
  new WithJtagDTM ++
  new WithGPIOs ++
  new WithInclusiveCache(512) ++

  new WithIOCellModels ++


  new WithDRAMSim ++
  new WithSimUART ++
  new WithSimJTAG ++
  new WithSimSerial

)
```


CHIPYARD Organization

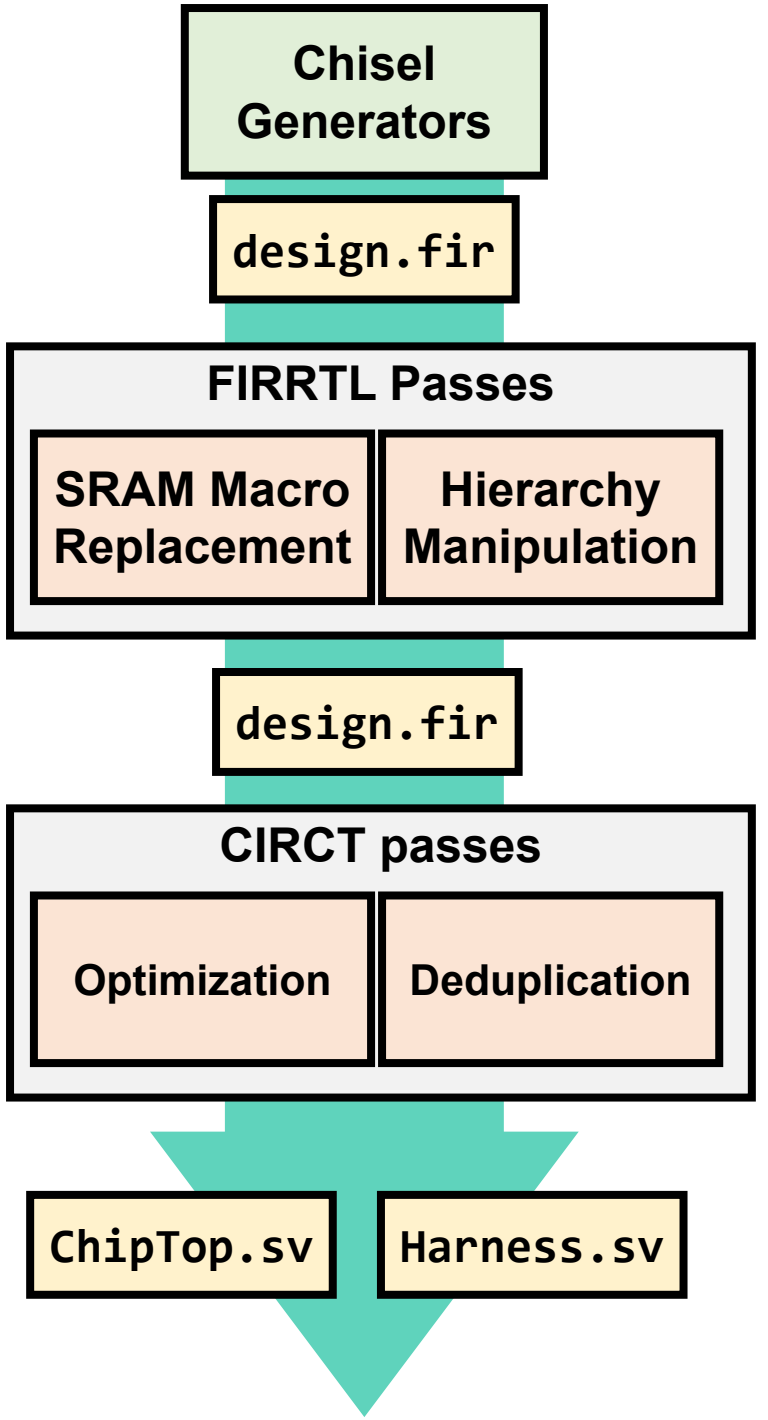


CHIPYARD Organization

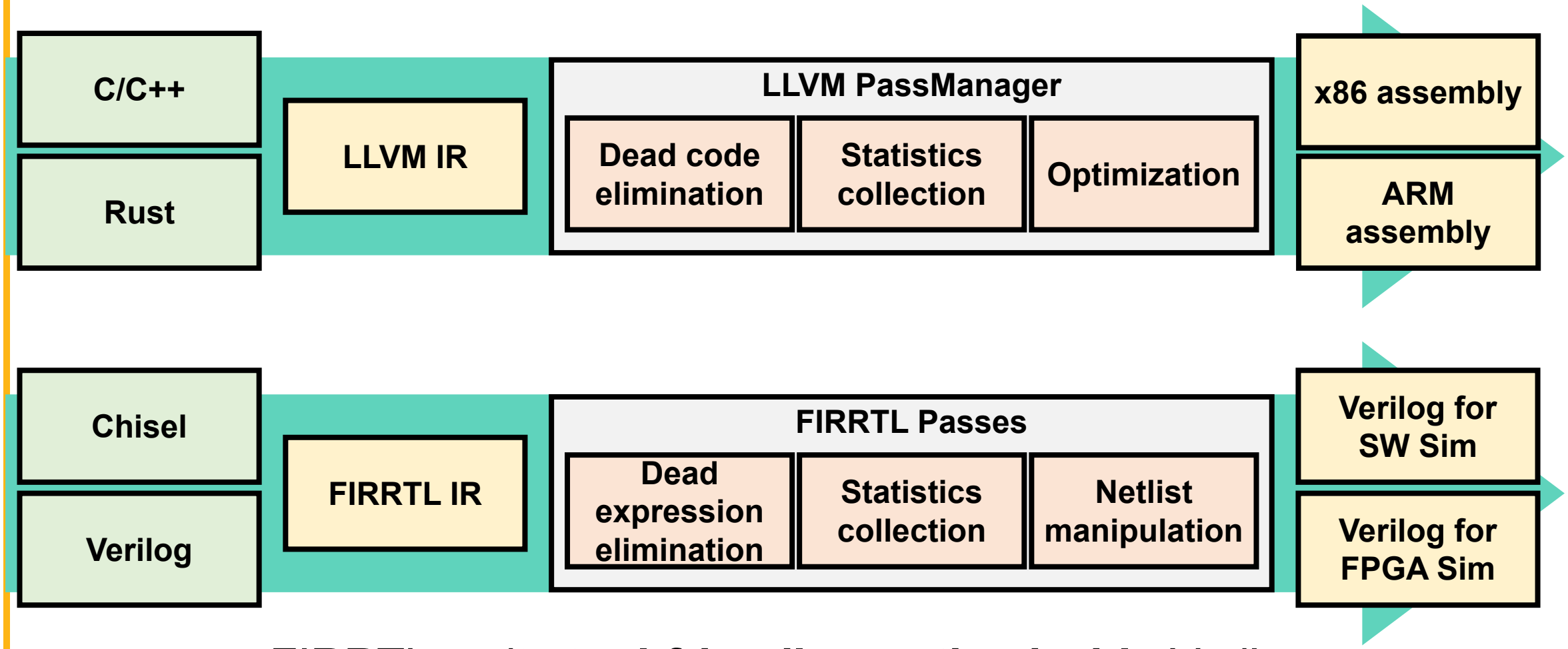



Elaboration Flow

- **Chisel** programs generate **FIRRTL** representations of hardware
- **FIRRTL passes** transform the target netlist
  - FireSim's AutoCounter/AutoILA/Printf use FIRRTL passes
  - VLSI flows can use FIRRTL passes to adjust the module hierarchy
- **NEW in Chipyard 1.9.1: CIRCT Backend**
  - CIRCT generates tool-friendly synthesizable Verilog from FIRRTL
  - Very fast/powerful FIRRTL compiler



FIRRTL



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

FIRRTL Passes

FireSim passes:

- Bridge target assertions/printfs to be visible on the host PC
- Provide FPGA utilization optimizations
- Implement debugging and analysis features (AutoILA, AutoCounter)

VLSI passes:

- Restructure module hierarchy
- Replace target memories with foundry SRAMs

FIRRTL provides an **abstraction layer** between **implementation** and **tooling**

CHIPYARD Organization

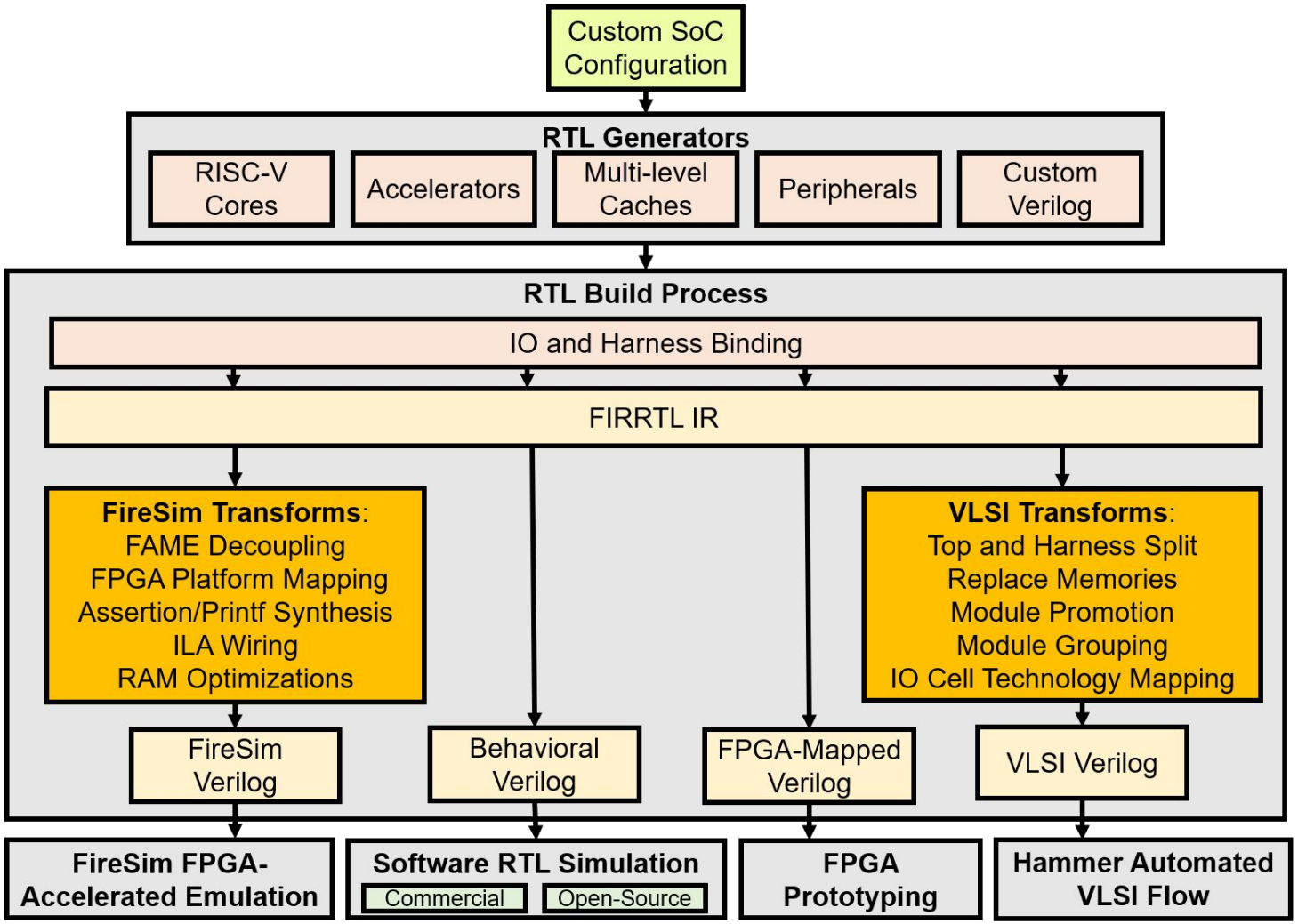
Configs: Describe parameterization of a multi-generator SoC

Generators: Flexible, reusable library of open-source Chisel generators (and Verilog too)

IOBinders/HarnessBinders: Enable configuring IO strategy and Harness features

FIRRTL/CIRCT Passes: Structured mechanism for supporting multiple flows

Target flows: Different use-cases for different types of users

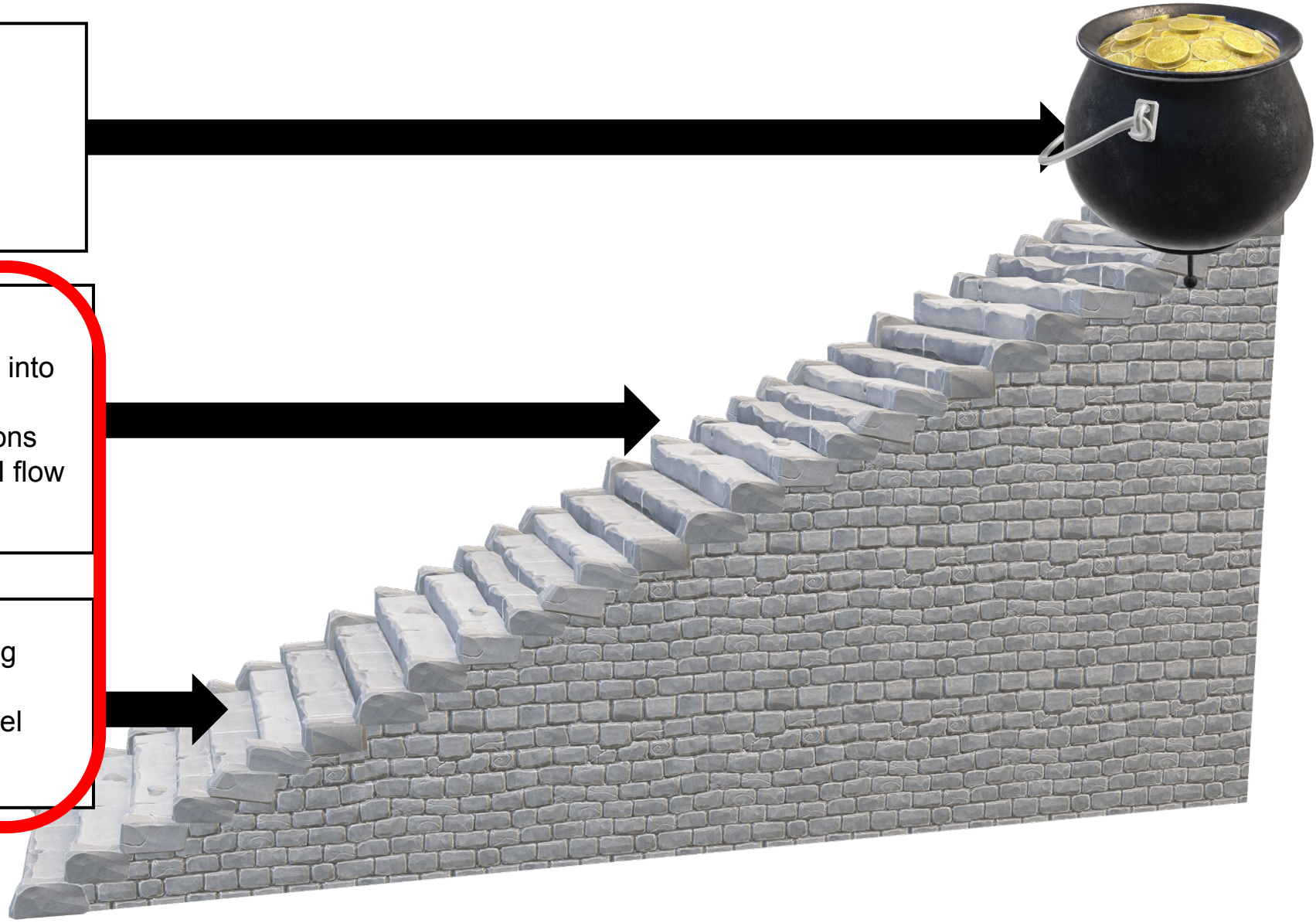


CHIPYARD Learning Curve

Advanced-level
- Configure custom IO/clocking setups
- Develop custom FireSim extensions
- Integrate and tape-out a complete SoC

Evaluation-level
- Integrate or develop custom hardware IP into Chipyard
- Run FireSim FPGA-accelerated simulations
- Push a design through the Hammer VLSI flow
- Build your own system

Exploratory-level
- Configure a custom SoC from pre-existing components
- Generate RTL, and simulate it in RTL level simulation
- Evaluate existing RISC-V designs



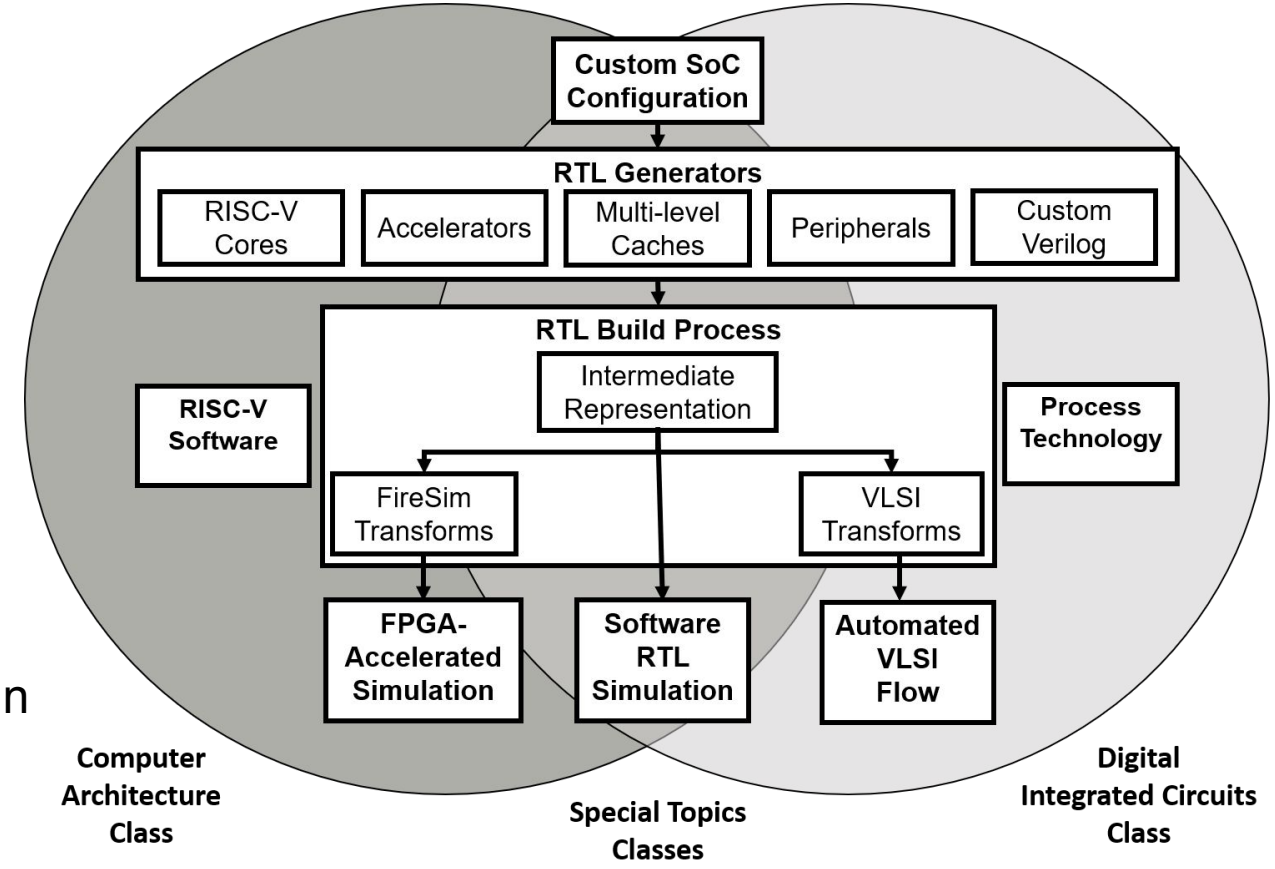
CHIPYARD For Education

Proven in many Berkeley Architecture courses

- Hardware for Machine Learning
- Undergraduate Computer Architecture
- Graduate Computer Architecture
- Advanced Digital ICs
- Tapeout HW design course

Advantages of common shared HW framework

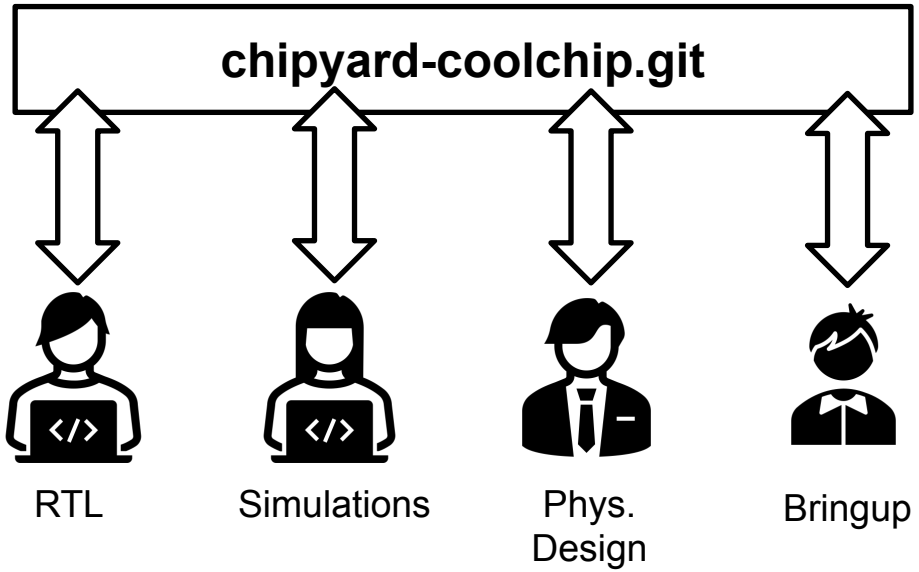
- Reduced ramp-up time for students
- Students learn framework once, reuse it in later courses
- Enables more advanced course projects (tapeout a chip in 1 semester)



CHIPYARD For Tapeouts

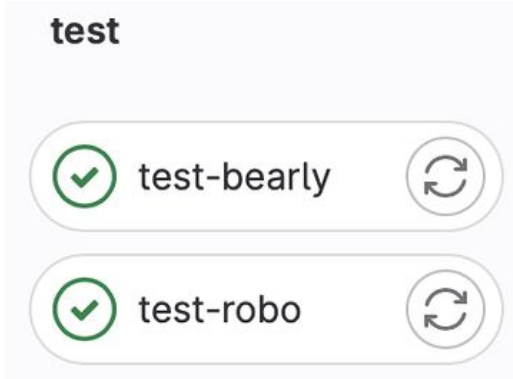
Standard Chipyard "Flow" For Tapeout

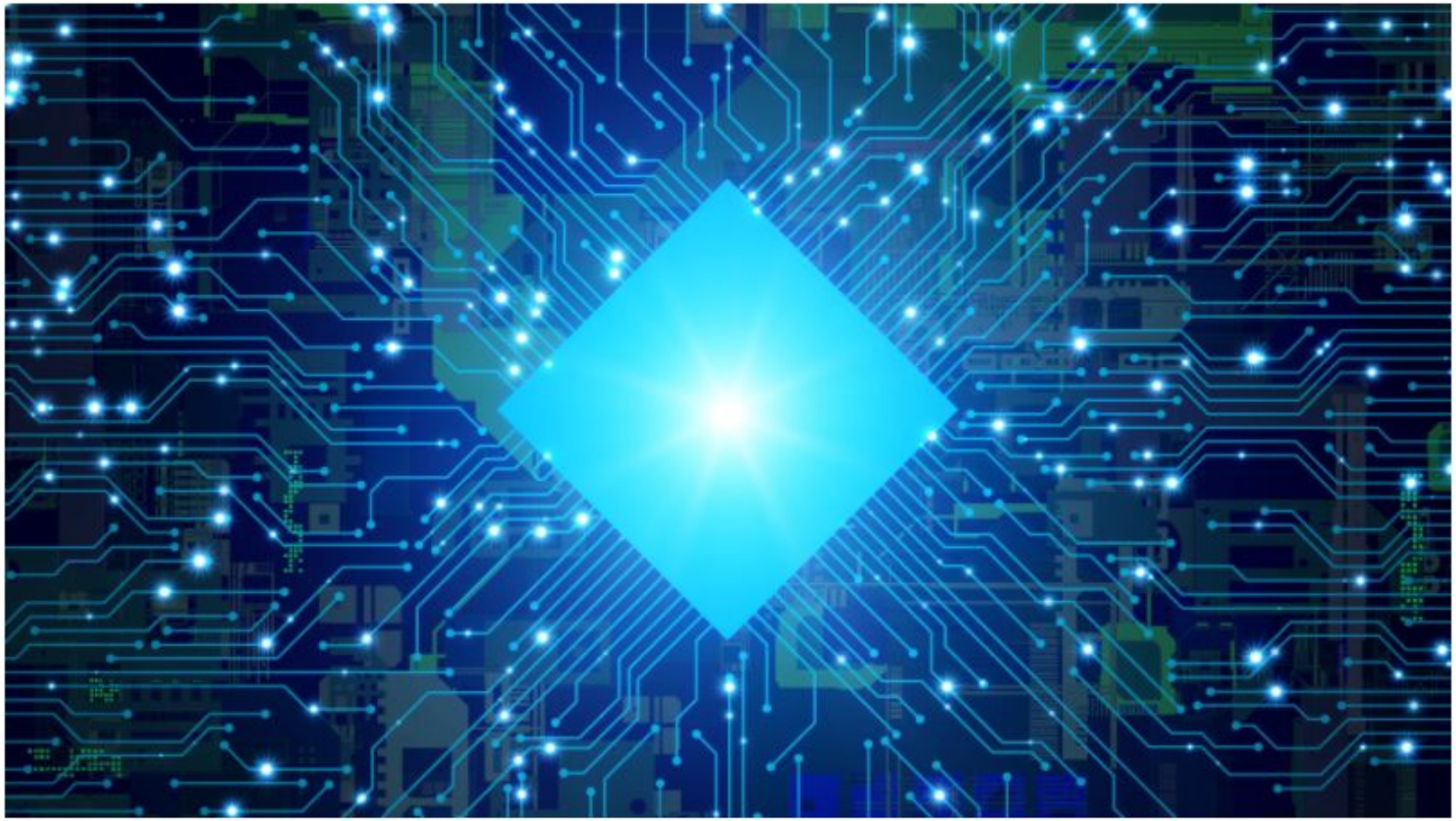
1. **RTL Development –** Develop new accelerators/devices and test rapidly
2. **FireSim –** Evaluate your design on real workloads
3. **Hammer -** Reusable/extensible VLSI flow
4. **Bringup –** generate FPGA bringup platforms using Chipyard



Chipyard is a single-source-of-truth for a chip

- Enables parallel workflows across different parts of the flow
- Reproducible environments simplify debugging
- Continuous integration for tapeouts





Berkeley Engineering students pull off novel chip design in a single semester. The class shows successful model for expanding entry into field of semiconductor design

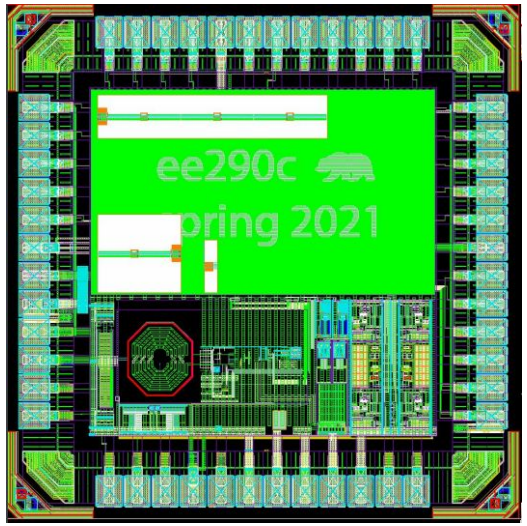
Berkeley engineering students pull off novel chip design in a single semester

Class shows successful model for expanding entry into field of semiconductor design

Taping-out Chips in Class

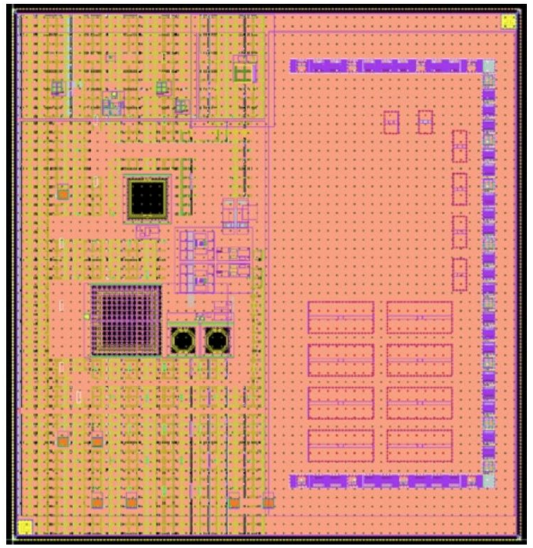
Spring 2021

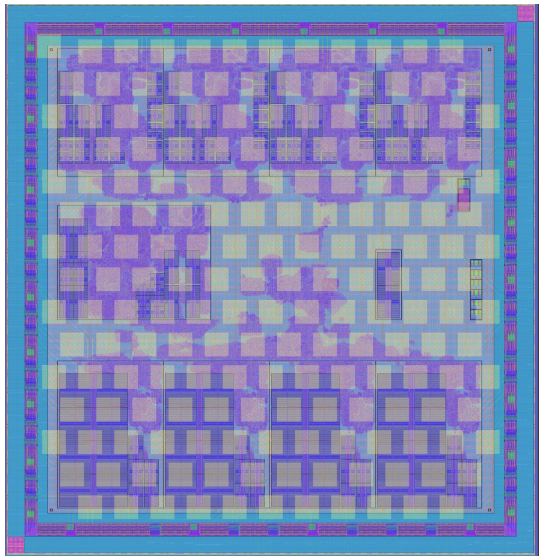
- **18** mostly masters/PhD
- **1 mm2** TSMC 28nm
- **16 weeks** from architecture to tapeout
- BLE mixed-signal IoT chip



Spring 2022

- **~40** mostly undergrads
- **2 x 4 mm^2** Intel 16nm
- **16 weeks** from architecture to tapeout
- BLE mixed-signal IoT chip
- Heterogeneous RISC-V SoC for sparse-ML and ISP





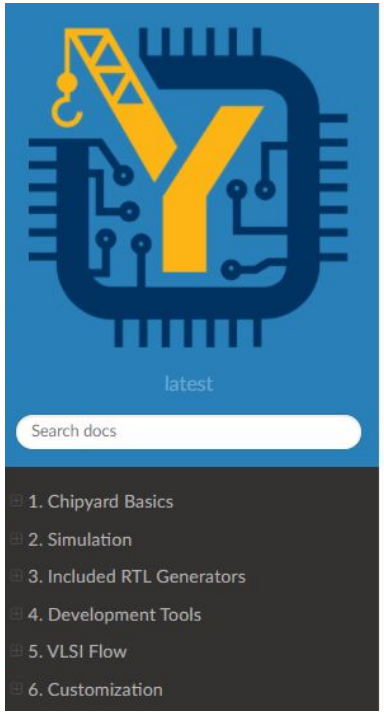

CHIPYARD For Research

- Add new accelerators for emerging applications
- Modify OS/driver/software
- Perform design-space exploration across many parameters
- Test and evaluate in RTL-simulation, FireSim
- Tapeout using HAMMER VLSI flow

CHIPYARD Community

Documentation:

- https://chipyard.readthedocs.io/en/dev/
- 133 pages
- Most of today's tutorial content is covered there

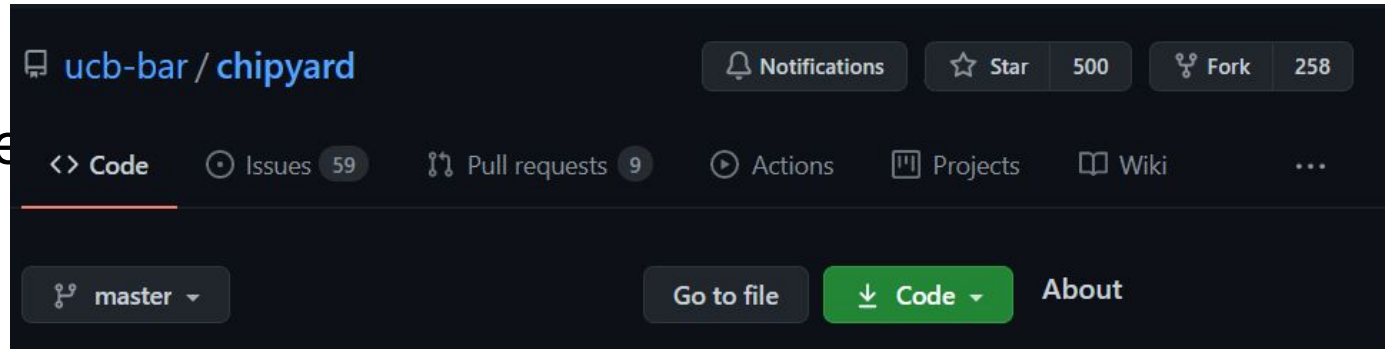


Mailing List:

- google.com/forum/#!forum/chipyard

Open-sourced:

- All code is hosted on GitHub
- Issues, feature-requests, PRs are welcomed



CHIPYARD

An open, extensible research and design platform for RISC-V SoCs

- Unified framework of parameterized generators
- One-stop-shop for RISC-V SoC design exploration
- Supports variety of flows for multiple use cases
- Open-sourced, community and research-friendly