Hammer VLSI Flow and Scaling out with Chiplets

Vikram Jain

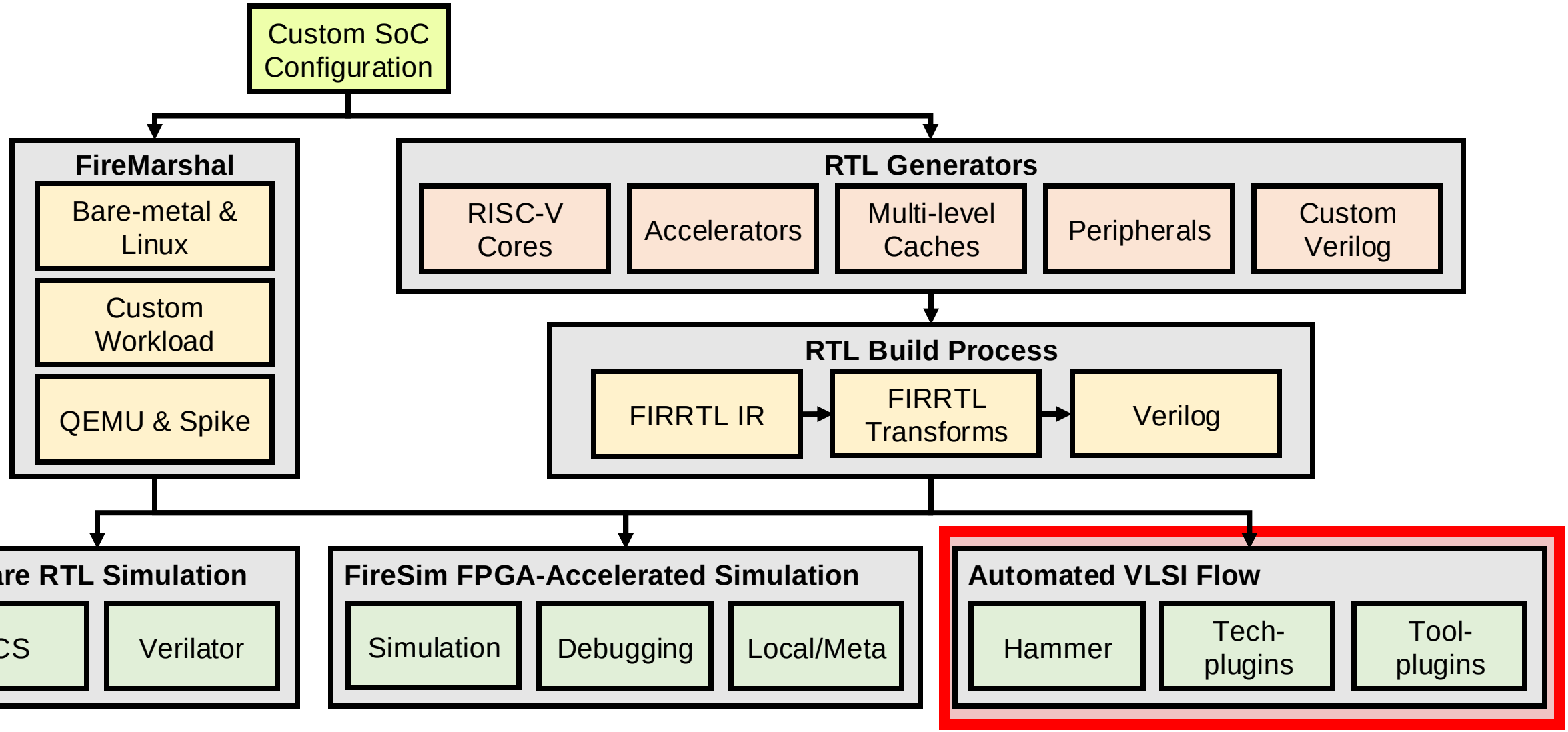
UC Berkeley

vikramj@berkeley.edu

Berkeley Architecture Research

CHIPYARD

Tutorial Roadmap



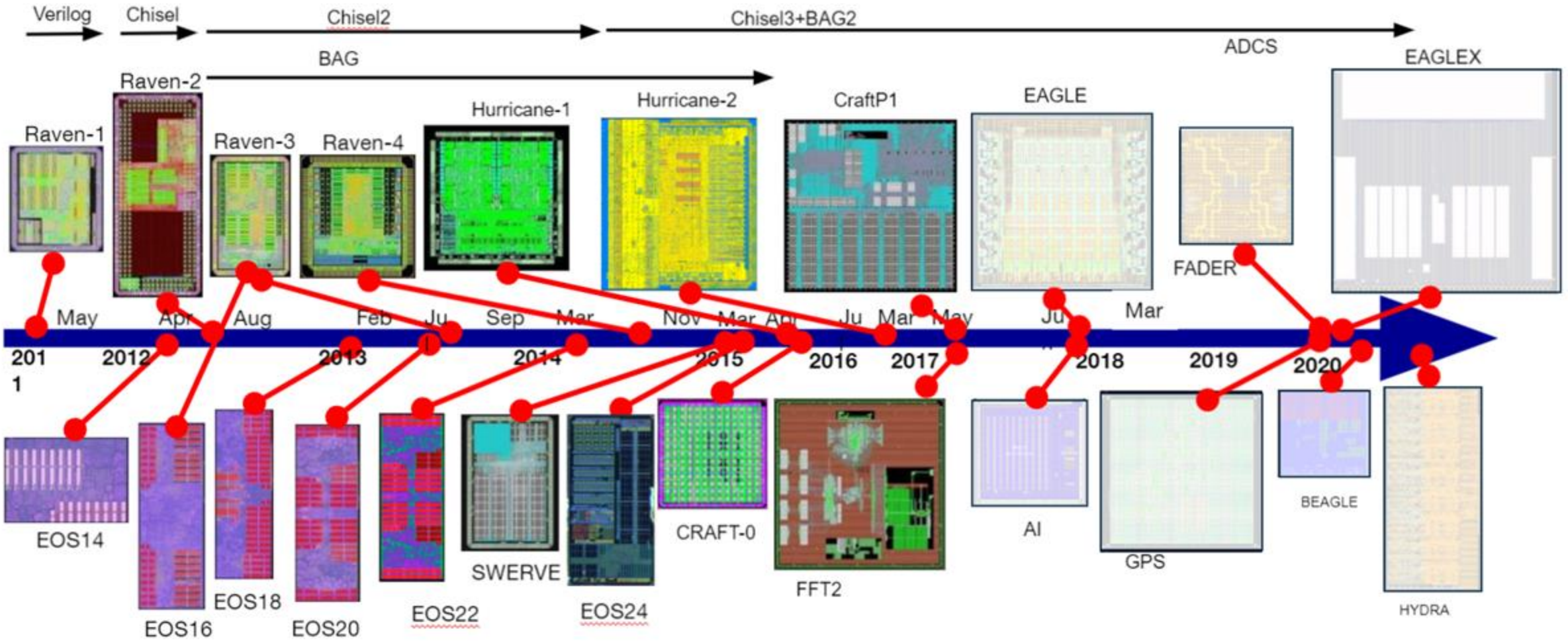
Agenda

- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

Agenda

- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

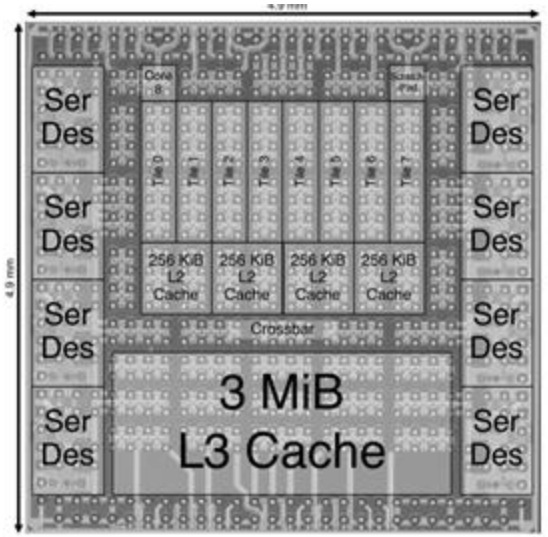
Hammer for Real Tapeouts



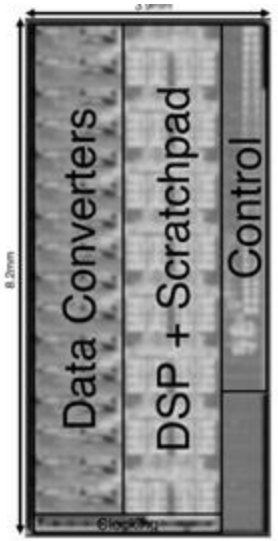
Raven, Hurricane: ST 28nm FDSOI, SWERVE: TSMC 28nm EOS: IBM 45nm SOI, CRAFT: 16nm TSMC,

Many Different Chips!

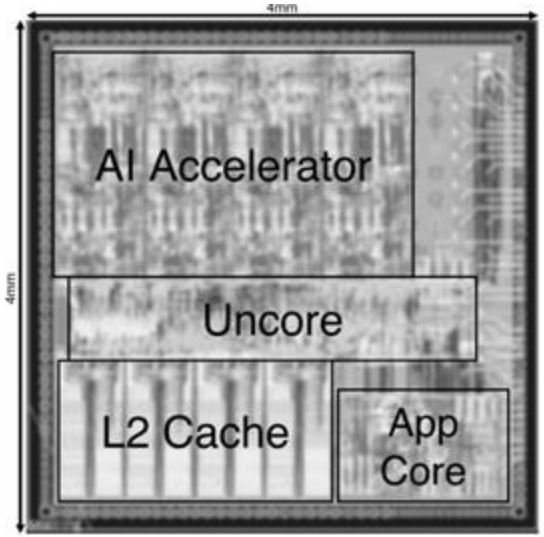
	Eagle [1]	HugeFlyingSoC	NavRx	WaterSerpent	MythicChip	OsciBear [2]	HDBinaryCore
Description	9-core RISC-V SoC	22-core RISC-V SoC	GPS receiver SoC	MU-MIMO baseband SoC	RISC-V SoC for ML	Bluetooth SoC	Hyperdim. computing proc.
Foundry Node	A 16nm	A 16nm	A 16nm	B 22nm	C 12nm	A 28nm, Sky130	A 28nm
Signoff Freq.	1.05 GHz	1.05 GHz	500 MHz	2 GHz	1.1 GHz	50 MHz	-
Hierarchy levels	3	3	1	3	2	1	1
Person-months	22	10	6	5	4	8, 1	8



Eagle [1]



WaterSerpent



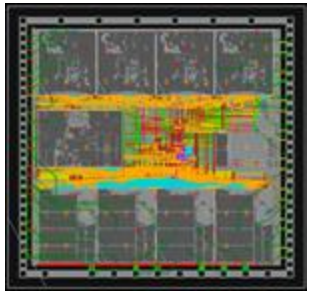
MythicChip

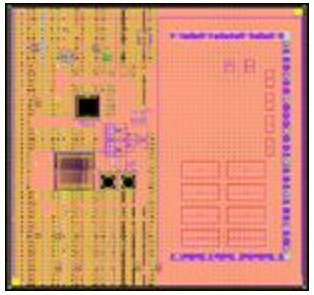
[1] C. Schmidt, et. al, *ISSCC 2021*
[2] D. Fritchman et. al, *IEEE SSCS Magazine, Spring 2022*

Hammer in Courses

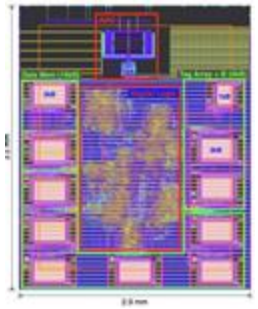
- Introduced in undergraduate digital circuits and systems labs:
  - http://github.com/EECS150 (ASAP7 and Sky130 plugins)
- Special topics ‘tapeout’ class
  - Spring 2024: 68 students with a mix of undergraduate and graduate students

2021 EE194/290C: OsciBear
TSMC 28nm

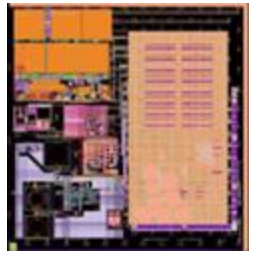




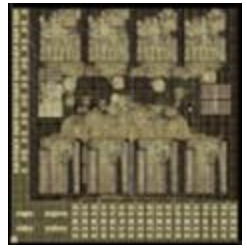
2022 EE194/290C: BearlyML (left) & SCuM-V (right)
Intel 16nm



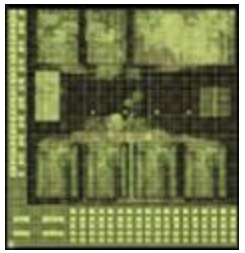
Sky130 MPW-2
Skywater 130nm



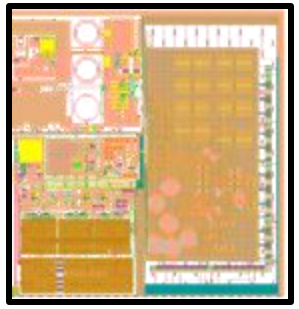
SCuM-V’23: 32b RISC-V core, BLE + 802.15.4, LDOs, references, radar



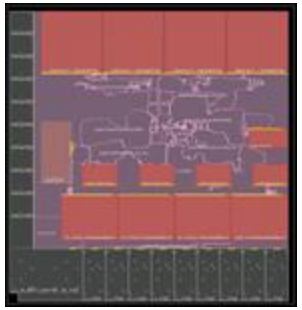
BearlyML’23: 4 RISC-V Rockets with custom sparse matrix acc, near-memory acc, NoC, L2$



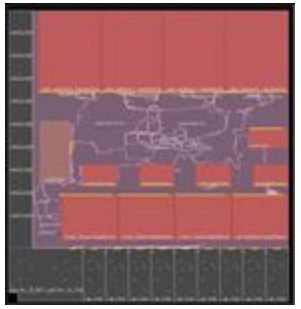
RoboChip’23: 2 RISC-V Rockets with Kalman, LQR acc, BooM + MTE, NoC, L2$



SCuM-V’24



BearlyML’24



DSPChip’24

Plugins Supported

Tech plugins	
Foundry	**Node**
A	16nm FinFET 28nm Planar
B	16nm FinFET 22nm FinFET
C	12nm FinFET 14nm FinFET
D	28nm SOI
Education	ASAP7 FreePDK45
Skywater	130nm

Tool plugins	
Action	**Tool**
Logic synthesis	Genus[C], Yosys, Vivado[X], DC[S]
Place and Route	Innovus[C], Vivado, OpenROAD, ICC[S]
DRC/LVS	Calibre[M], ICV[S], Magic/Netgen
Simulation	VCS[S], Xcelium[C]
Power, EM/IR	Joules[C], Voltus[C]
LEC	Conformal[C], Yosys

[C]Cadence [S]Synopsys [M]Siemens Mentor [X]Xilinx

Next generation technology nodes

Plugins to be Supported

Tech plugins	
Foundry	**Node**
A	16nm FinFET 28nm Planar
B	16nm FinFET 22nm FinFET 18A Gate-All-Around
C	12nm FinFET 14nm FinFET
D	28nm SOI
Education	ASAP7 FreePDK45
Skywater	130nm

Tool plugins	
Action	**Tool**
Logic synthesis	Genus[C], Yosys, Vivado[X], DC[S]
Place and Route	Innovus[C], Vivado, OpenROAD, ICC[S] Fusion Compiler[S]
DRC/LVS	Calibre[M], ICV[S], Magic/Netgen
Simulation	VCS[S], Xcelium[C]
Power, EM/IR	Joules[C], Voltus[C]
LEC	Conformal[C], Yosys

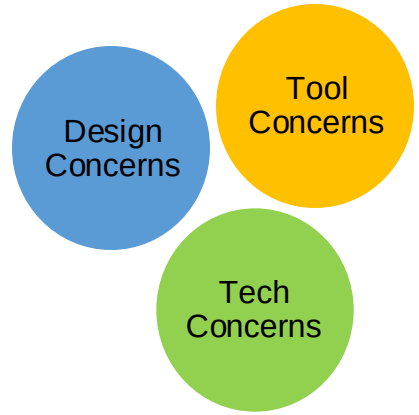
[C]Cadence [S]Synopsys [M]Siemens Mentor [X]Xilinx

Agenda

- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

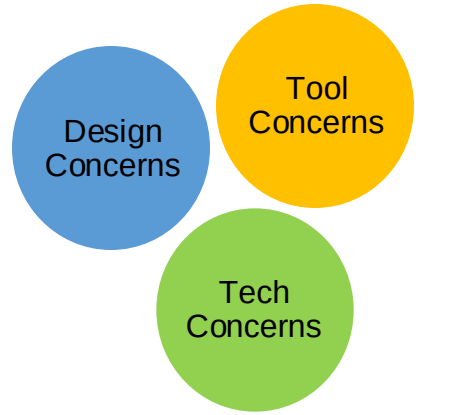
Hammer Design Principles

1. **Separation of Concerns**
   - Decouple design-, tool-, and tech-specific concerns



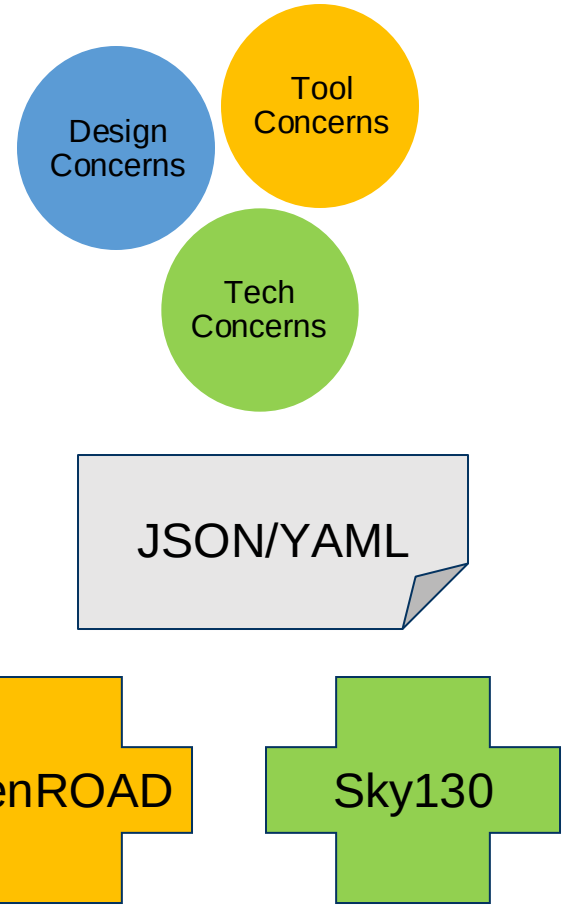
Hammer Design Principles

1. **Separation of Concerns**
   - Decouple design-, tool-, and tech-specific concerns
2. **Standardization**
   - Data interchange schema for constraints, options, files



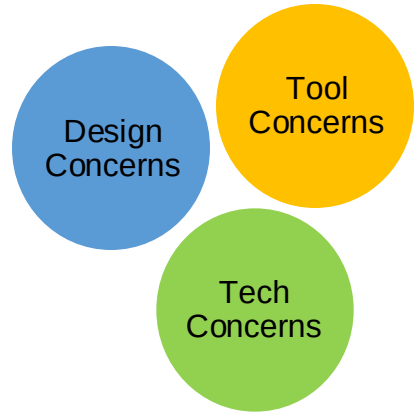
Hammer Design Principles

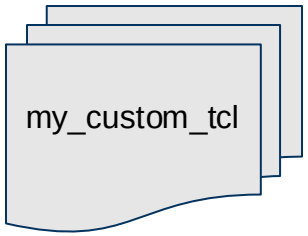
1. **Separation of Concerns**
   - Decouple design-, tool-, and tech-specific concerns
2. **Standardization**
   - Data interchange schema for constraints, options, files
3. **Modularity**
   - Interchangeable & shareable tool & tech plugins



Hammer Design Principles

1. **Separation of Concerns**
   - Decouple design-, tool-, and tech-specific concerns
2. **Standardization**
   - Data interchange schema for constraints, options, files
3. **Modularity**
   - Interchangeable & shareable tool & tech plugins
4. **Incremental Adoption**
   - Mix reusable & custom solutions

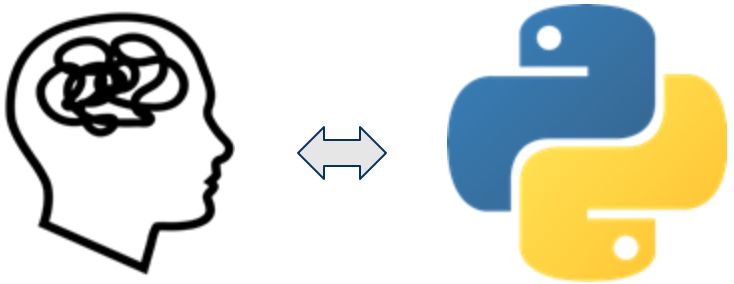




What is Hammer?

Hammer is:

… a Python framework for abstracting and building standardized flows

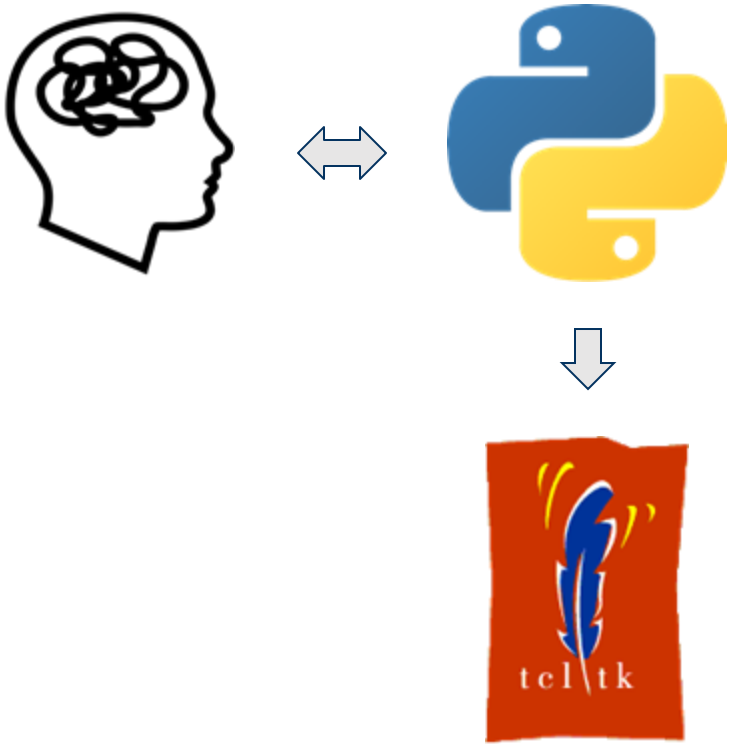


What is Hammer?

Hammer is:

… a Python framework for abstracting and building standardized flows

… not a typical CAD tool—it generates scripts and manages tool execution



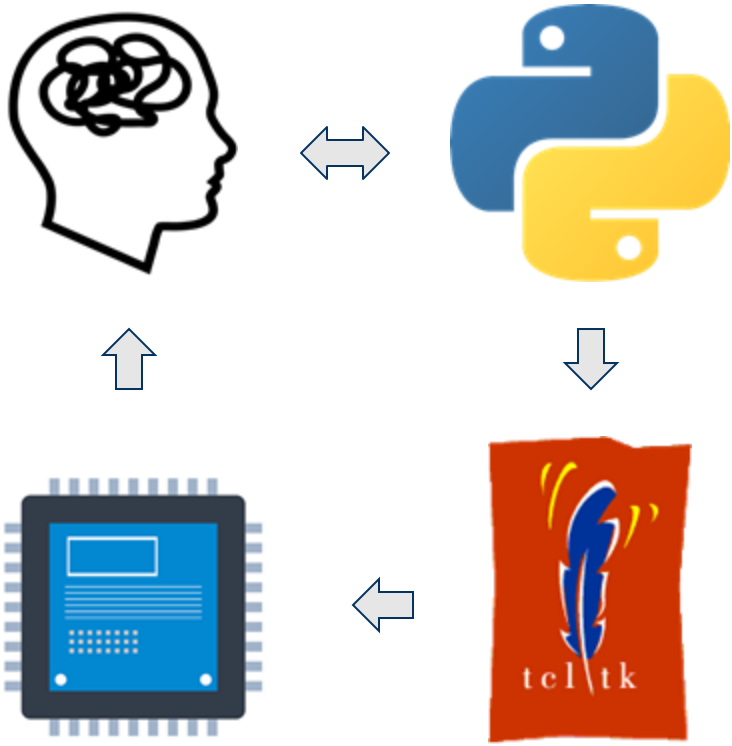
What is Hammer?

Hammer is:

… a Python framework for abstracting and building standardized flows

… not a typical CAD tool—it generates scripts and manages tool execution

… proven for architecture exploration, teaching, and research chips



What is Hammer?

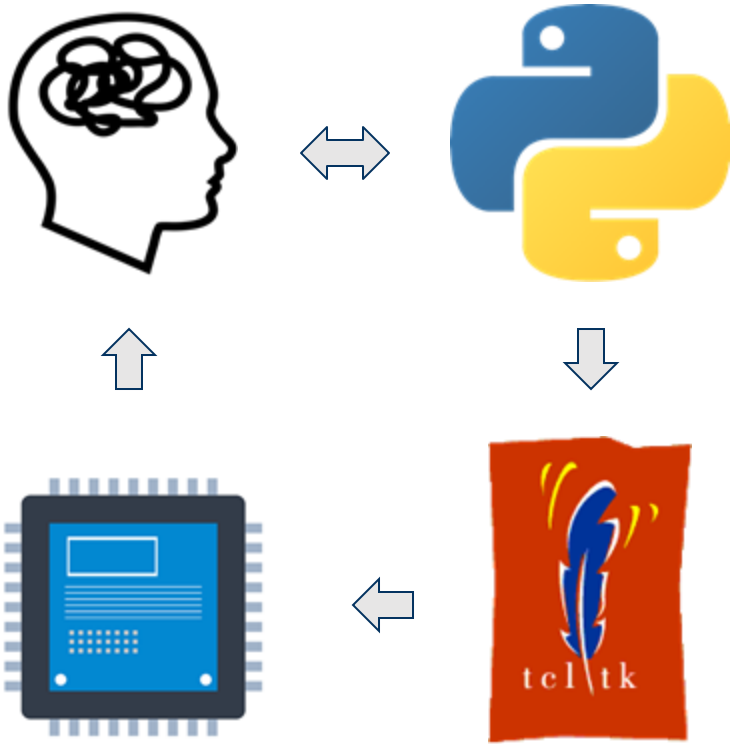
Hammer is:

… a Python framework for abstracting and building standardized flows

… not a typical CAD tool—it generates scripts and manages tool execution

… proven for architecture exploration, teaching, and research chips

… open-source!



What is Hammer?

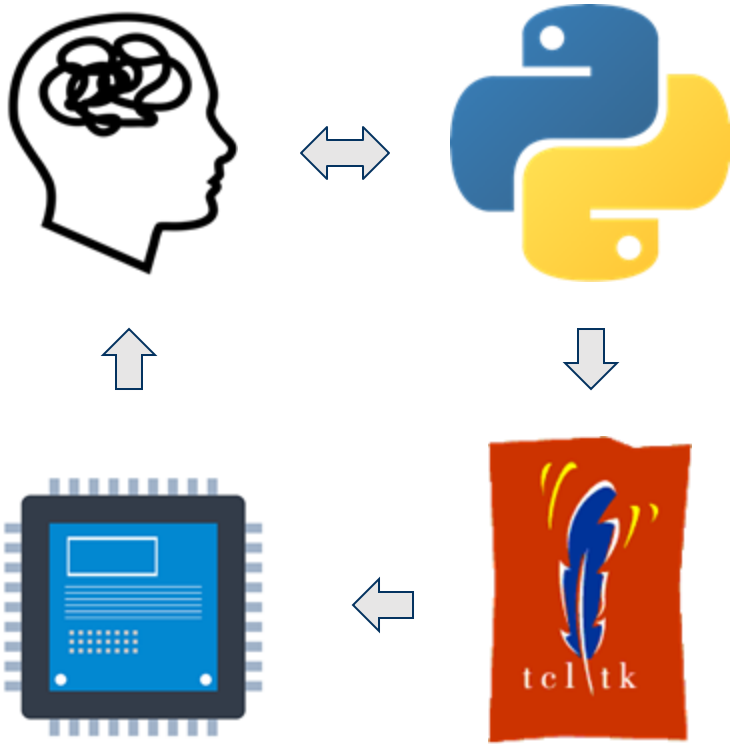
Hammer is:

… a Python framework for abstracting and building standardized flows

… not a typical CAD tool—it generates scripts and manages tool execution

… proven for architecture exploration, teaching, and research chips

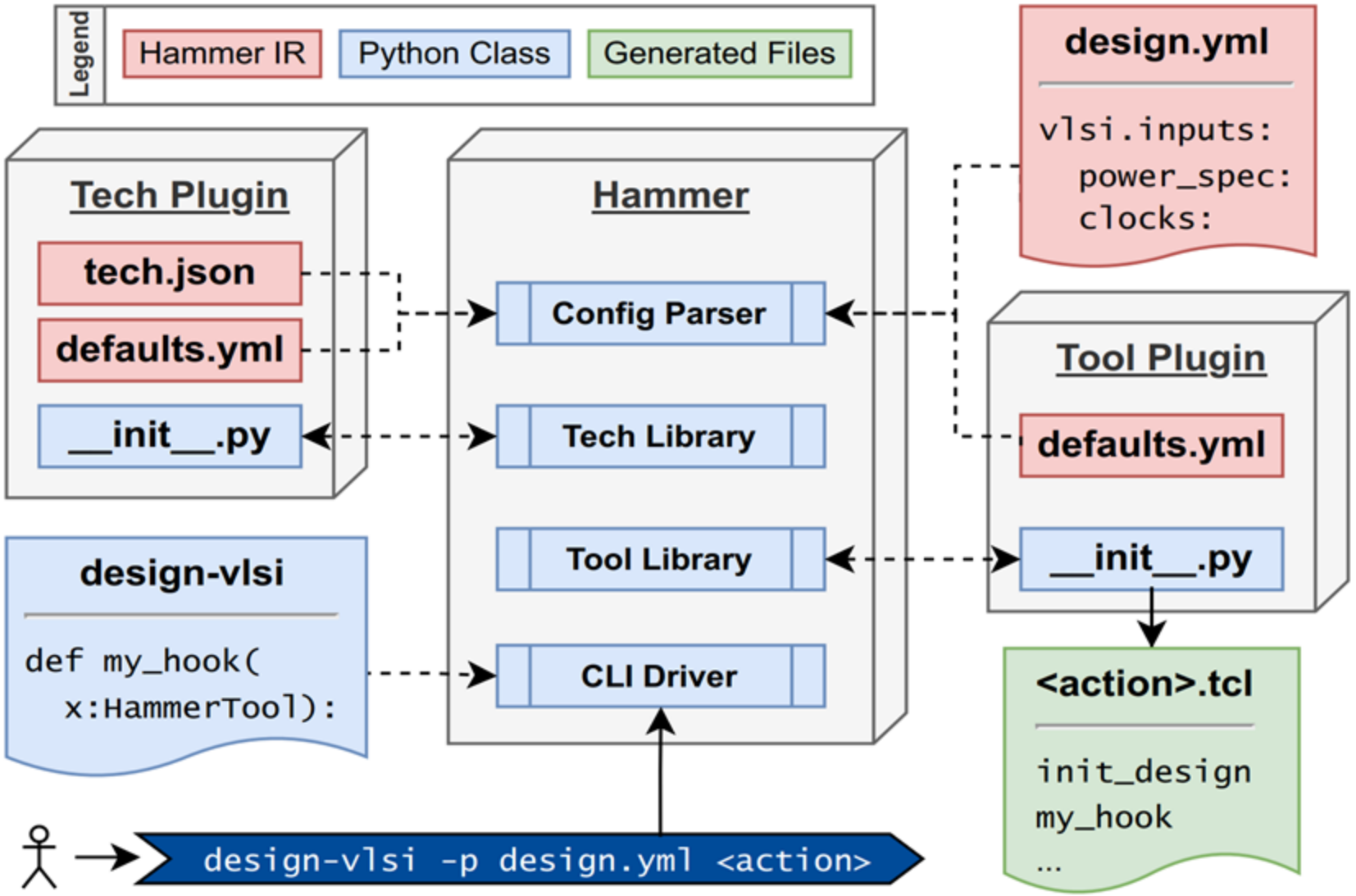
… open-source!



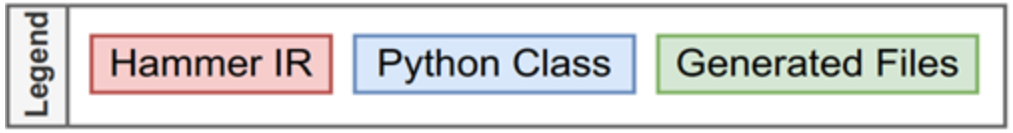
Project started in 2015, research chip use from 2016, class use from 2019, currently ~35k lines of code

Berkeley Architecture Research

Hammer Software Architecture



Hammer Intermediate Representation (IR)

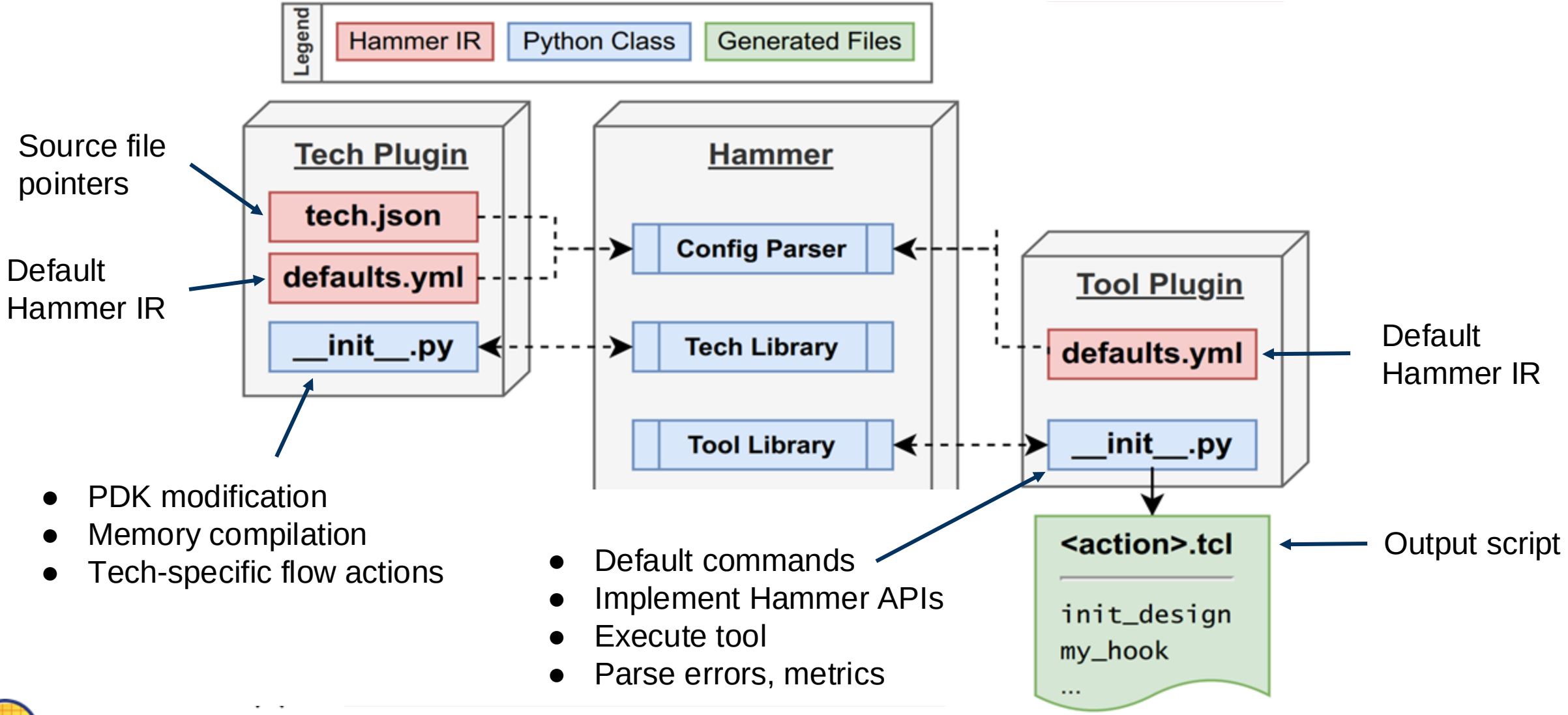


design.yml

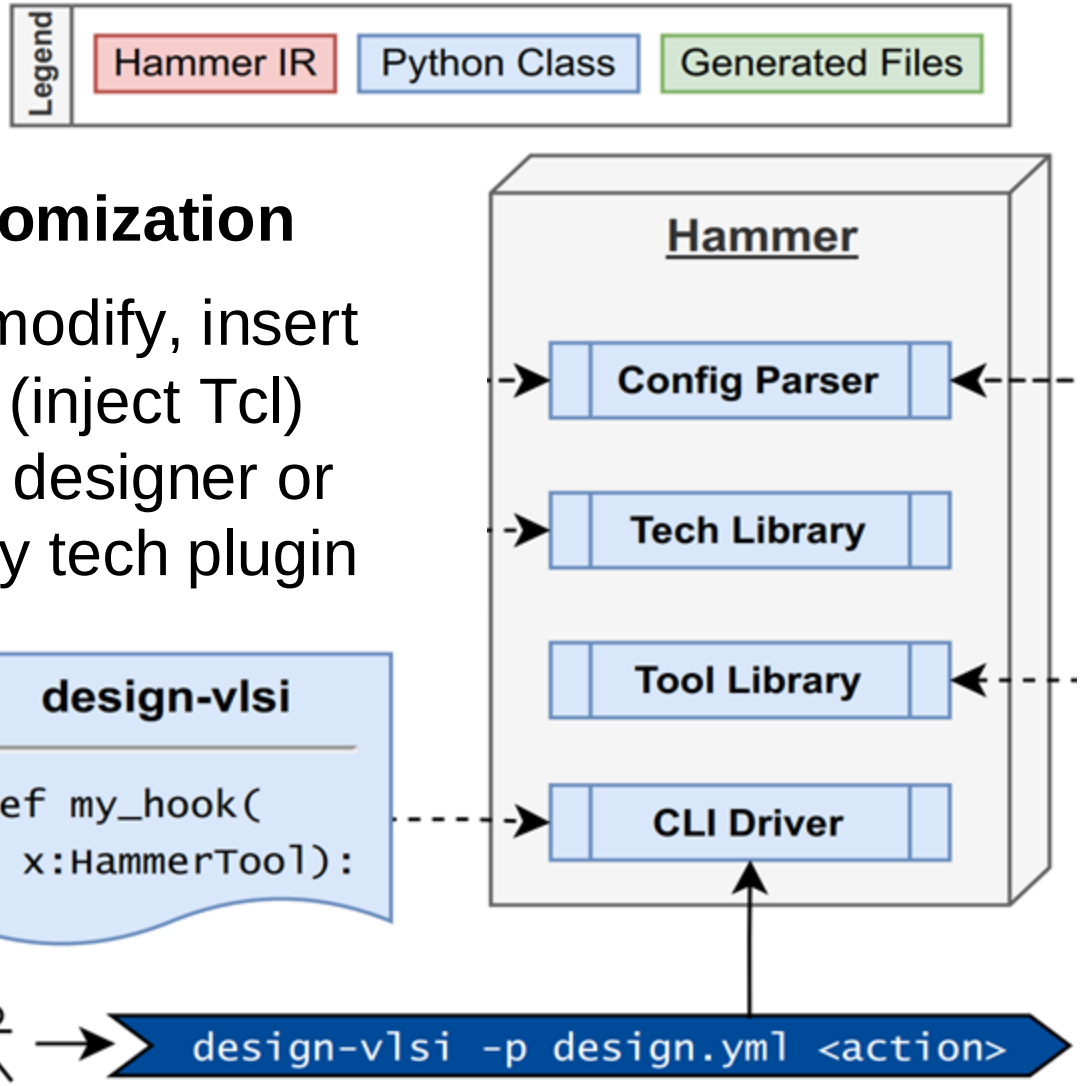
```
vlsi.inputs:
  power_spec:
  clocks:
```

- Standard data interchange format
  - Constraints, options, intermediate files, etc.
  - YAML for humans, JSON for programs (annotation format)
  - **De-embeds designer intent and expertise from Tcl scripts**
- IR Metaprogramming
  - Modify any IR key with traceable history, type- and validity-checking
  - **Mechanism for partitioning and customizing design intent**

Tool and Tech Plugins



Hooks and Drivers



Hooks = **customization**

- Replace, modify, insert flow steps (inject Tcl)
- Written by designer or supplied by tech plugin

Hammer Driver

- Parses all IR, hooks
- Auto-generates hierarchical flow graph as Makefile
- Easy-to-use CLI

Agenda

- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

Everyone can use Hammer

How: provide sensible defaults with methods to override

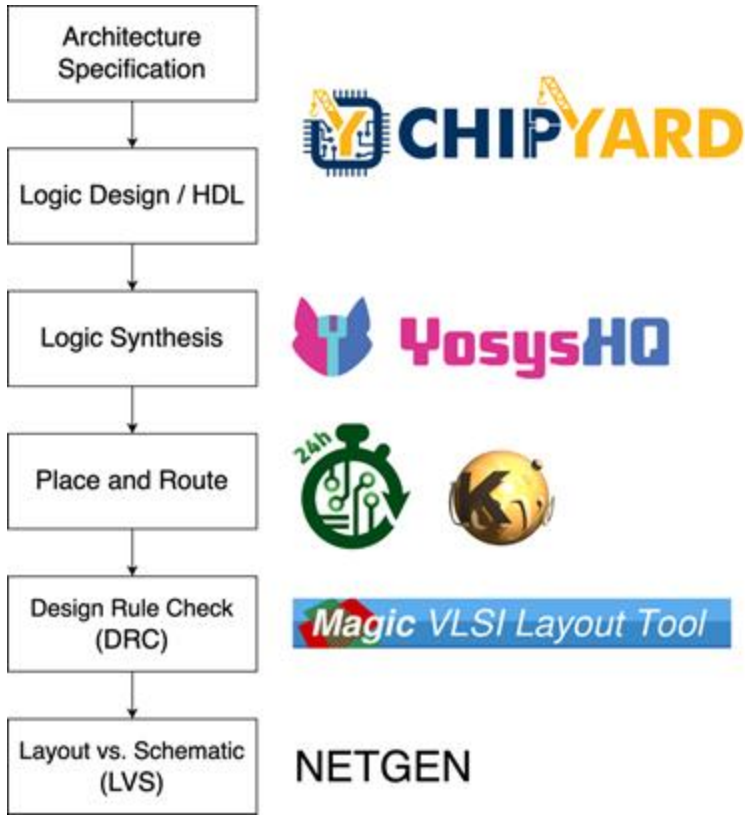
Sensible default	Override method
A default set of flow steps for every action (syn, par, etc.)	Hooks - inject your own steps anywhere
Auto-generated timing (SDC) & power (CPF) constraints	Use your own custom SDC and CPF files
Auto-generated power meshes from high-level parameters	Use foundry-provided or your own mesh generator
Auto-generated Makefile implementing flow graph	Running Hammer via command line, custom Makefiles

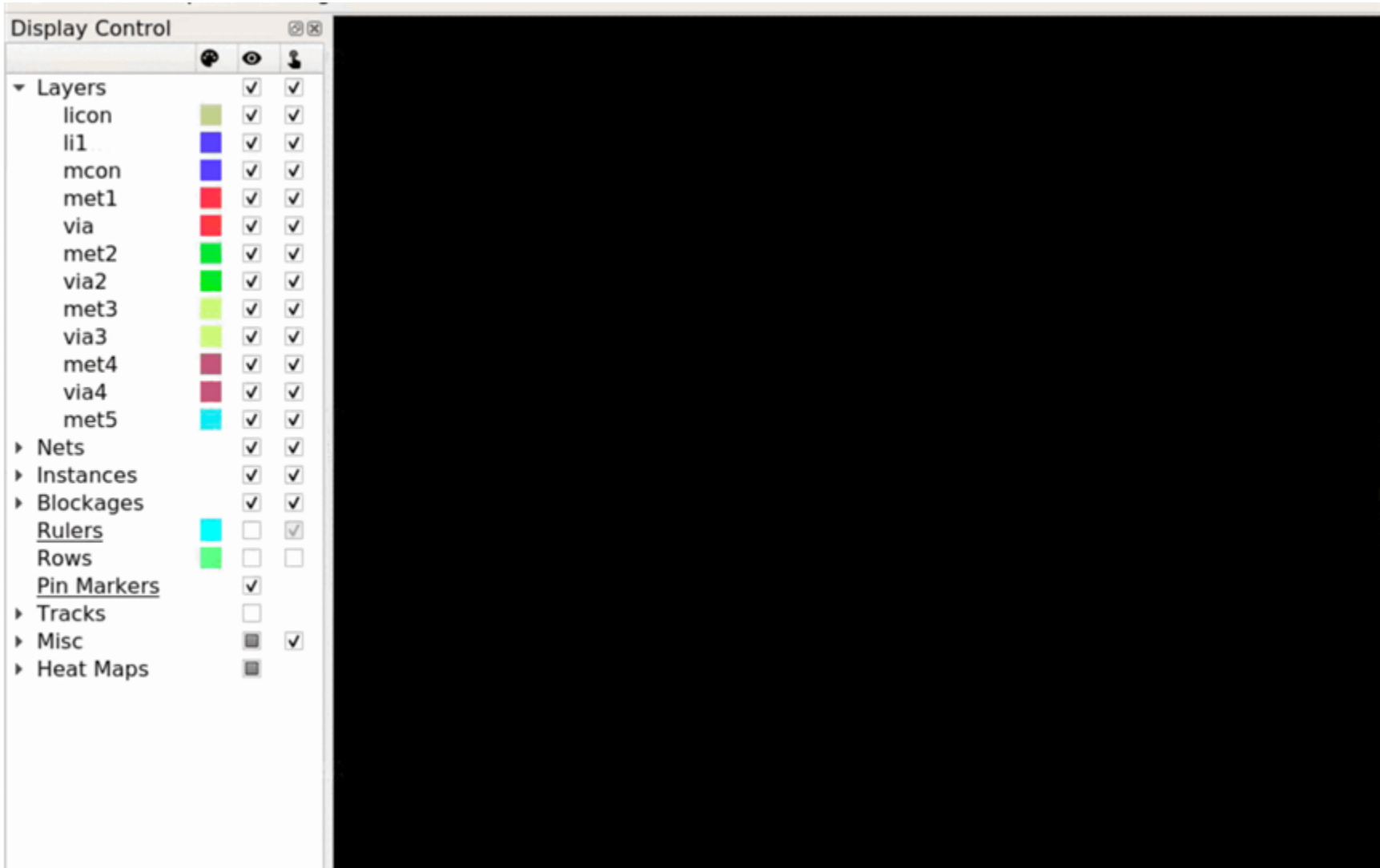
Result: gets you 80-90% of the way there out of the box

- Easily learn the VLSI flow, get early design feedback
- Chipyard examples with ASAP7, Sky130

Tutorial: TinyRocket RTL-to-GDS

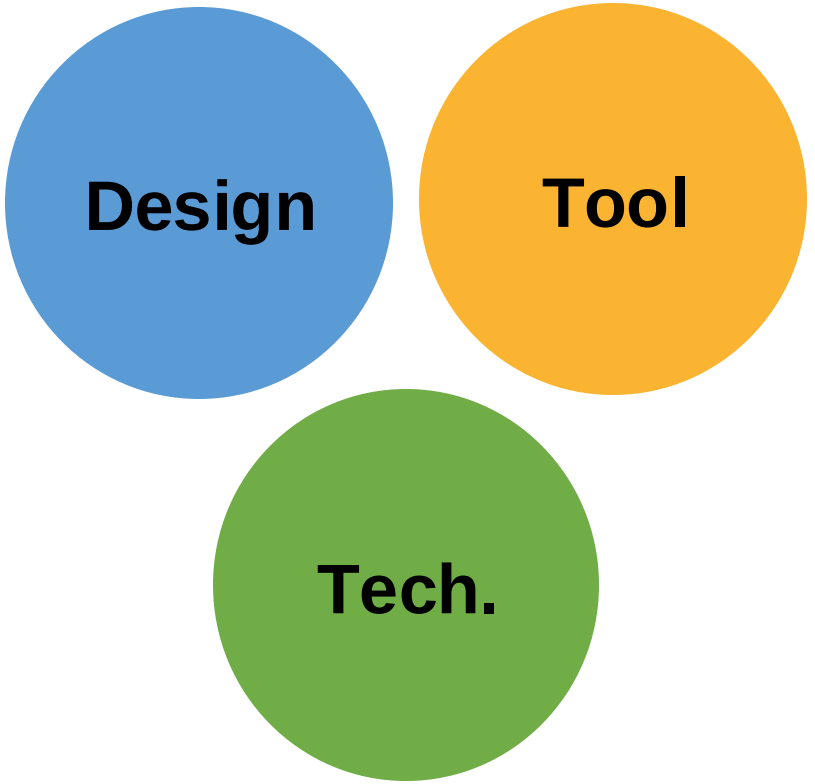
https://chipyard.readthedocs.io/en/latest/VLSI/Sky130-OpenROAD-Tutorial.html





Summary

- Physical design is hard—there are good reasons why most people try to avoid it.
  - Chips are growing in complexity
  - Un-natural evolution of the EDA/PDK stack
- Hammer helps separate design, tool, and technology concerns
  - Enables re-use
  - Enables advanced abstractions and generators
- Easy power and area evaluation
  - Using Hammer, open source PDK, commercial EDA



Learn More

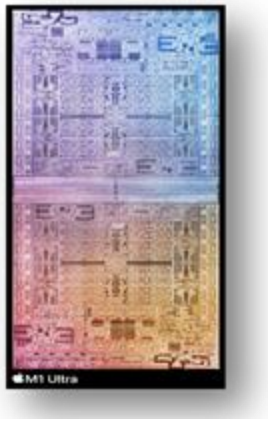
- Github: https://github.com/ucb-bar/hammer/
- Documentation: https://hammer-vlsi.readthedocs.io/
- Chipyard-specific documentation: https://chipyard.readthedocs.io/en/latest/VLSI/index.html
- Discussions/forum: https://github.com/ucb-bar/hammer/discussions
- Mentor plugin access request:
  - hammer-plugins-access@lists.berkeley.edu
- UCB Digital Design labs: https://github.com/EECS150/asic_labs_sp23
  - full lab releases coming soon!

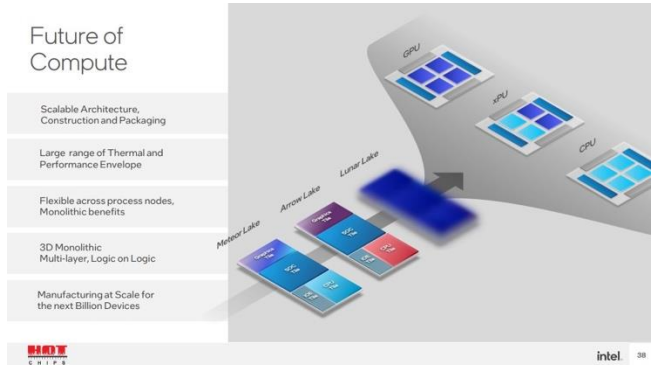
Agenda

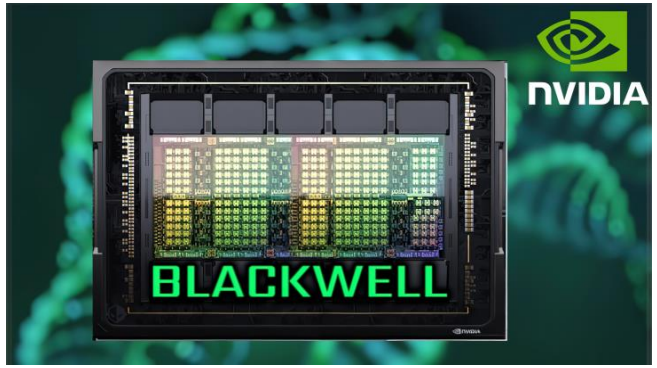
- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

Chiplets in academia

- **Industry** is embracing chiplets to **optimize cost** of high-volume performant products
  - **Secondary consideration** is to reduce the **NRE of domain-specific solutions**



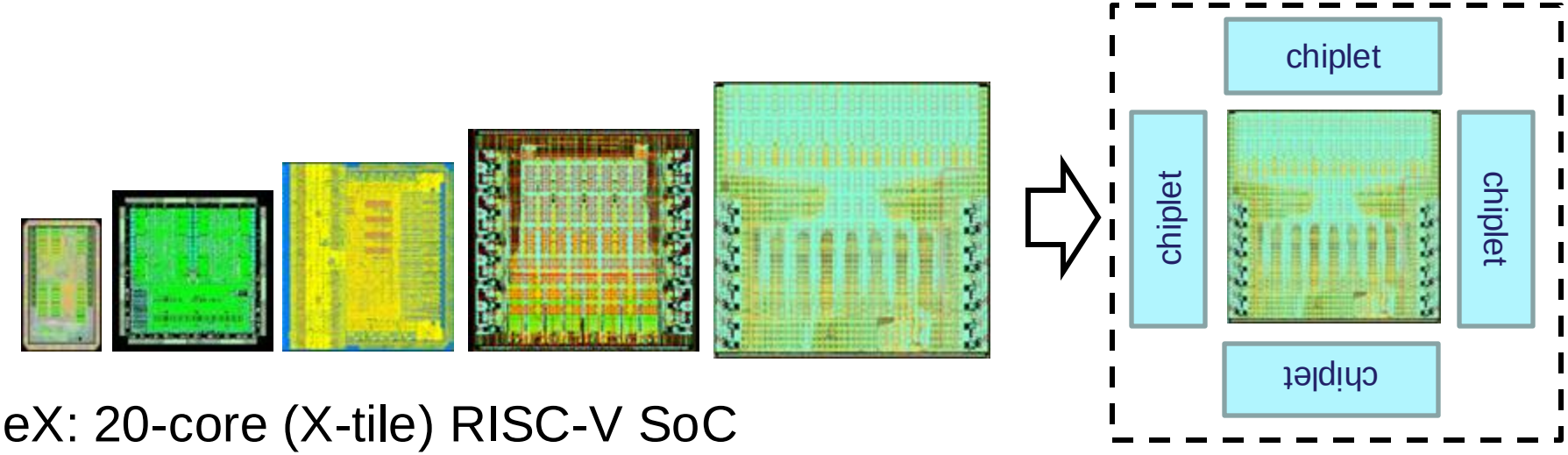




- **Academia Goal**: Enable development of complete functional and performant domain-specific systems by de-risking critical pieces (**focus on NRE**)
  - Chiplets can be reused, if based on standard interfaces
  - Chiplets enables bottom-up approach to scaling
  - Design cost is lower, but not negligible
  - We need to keep innovating, while demonstrating complete systems

Our motivation for chiplets

- We have designed increasingly more complex chips to test our design methodologies



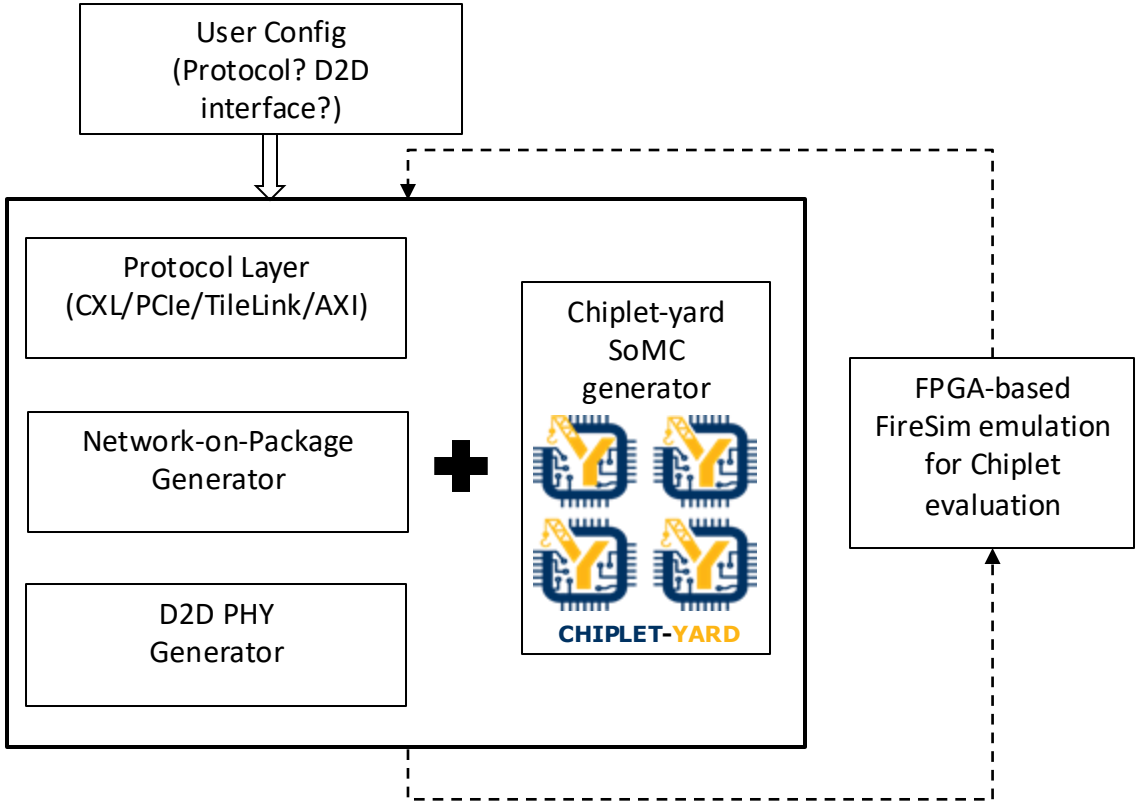
- EagleX: 20-core (X-tile) RISC-V SoC
  - 7.5mm x 7.5mm die in TSMC 16FFC
  - Boots Linux
  - Hard to justify repeatedly building chips like these in Academia
- Build a **base chiplet** that can be shared across multiple platforms
  - Innovation focused on partner chiplets
- Build many small chiplets to scale up and scale out

CHIPYARD
GEMMINI
HAMMER
FireSim

Framework for Chiplet Systems

- Need systematic framework for building, evaluating, and testing complete chiplet-based systems
- Need standard interfaces between chiplets for plug-and-play support with our and 3rd party IPs

- Tasks:
  - Chiplet-yard – extending Chipyard SoC generator for generating chiplets
  - Die-to-die standard interface generators
  - Networks-on-package generators
  - FireSim based FPGA emulation of multi-chiplet solution

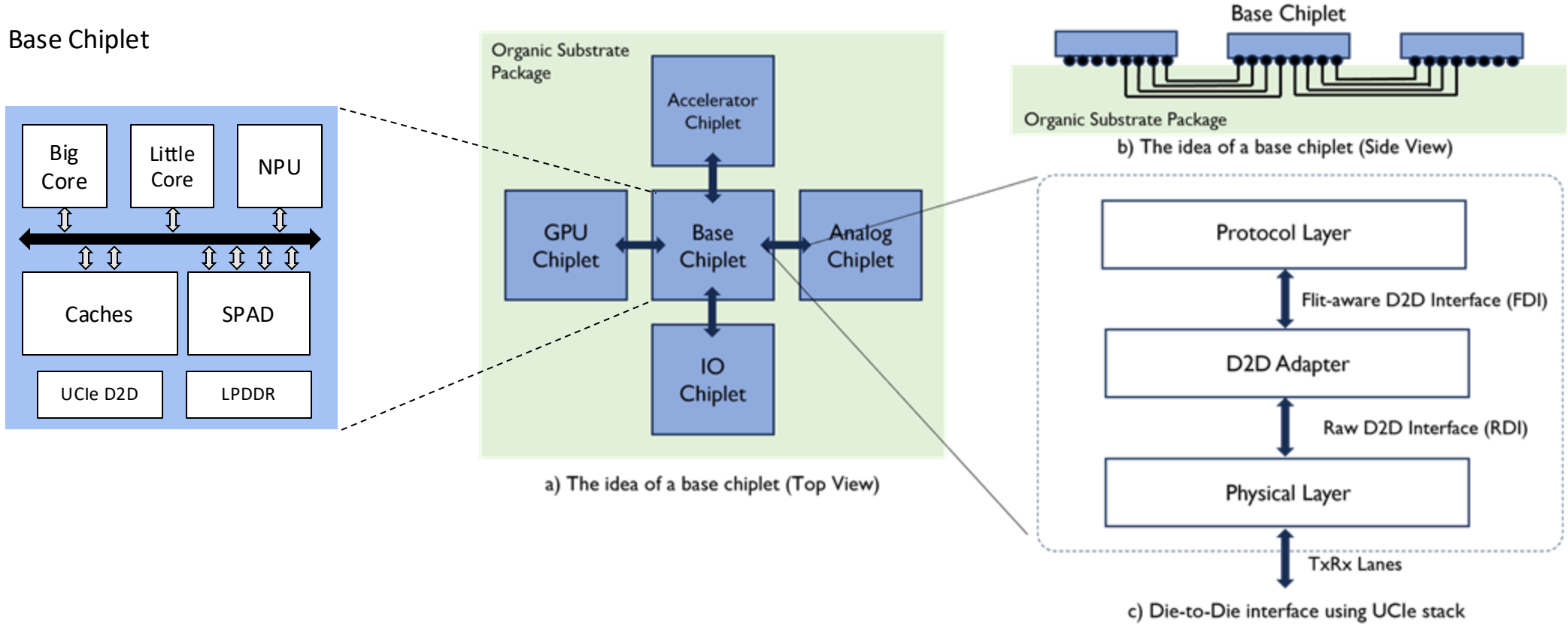


Agenda

- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

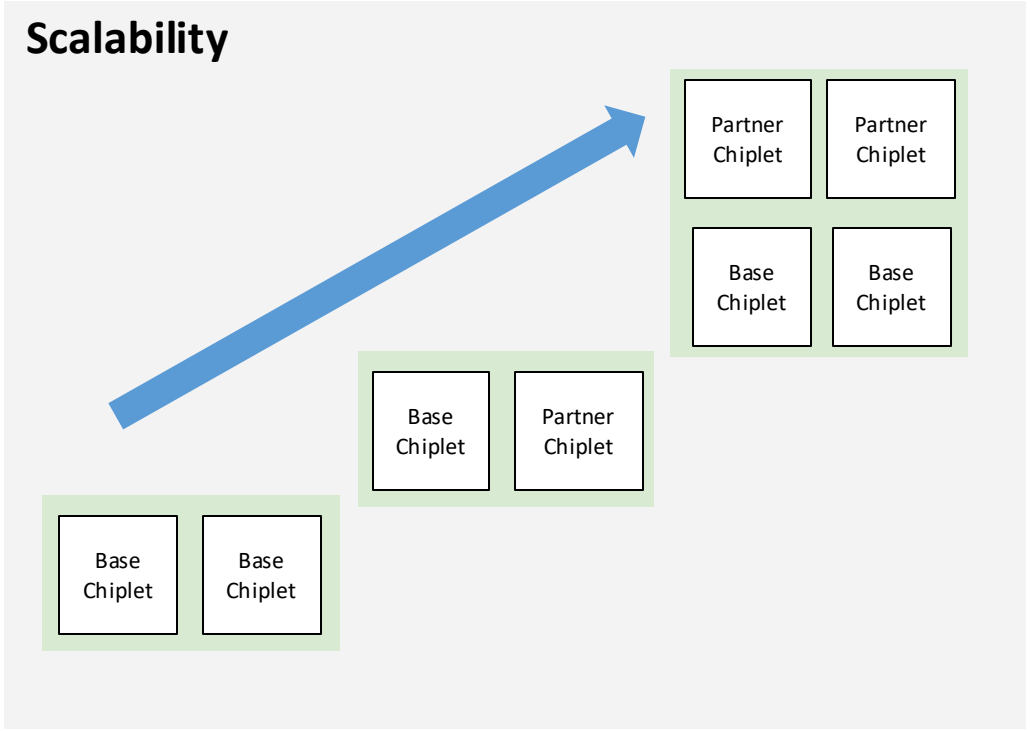
Objective 1: Reusable Base chiplet

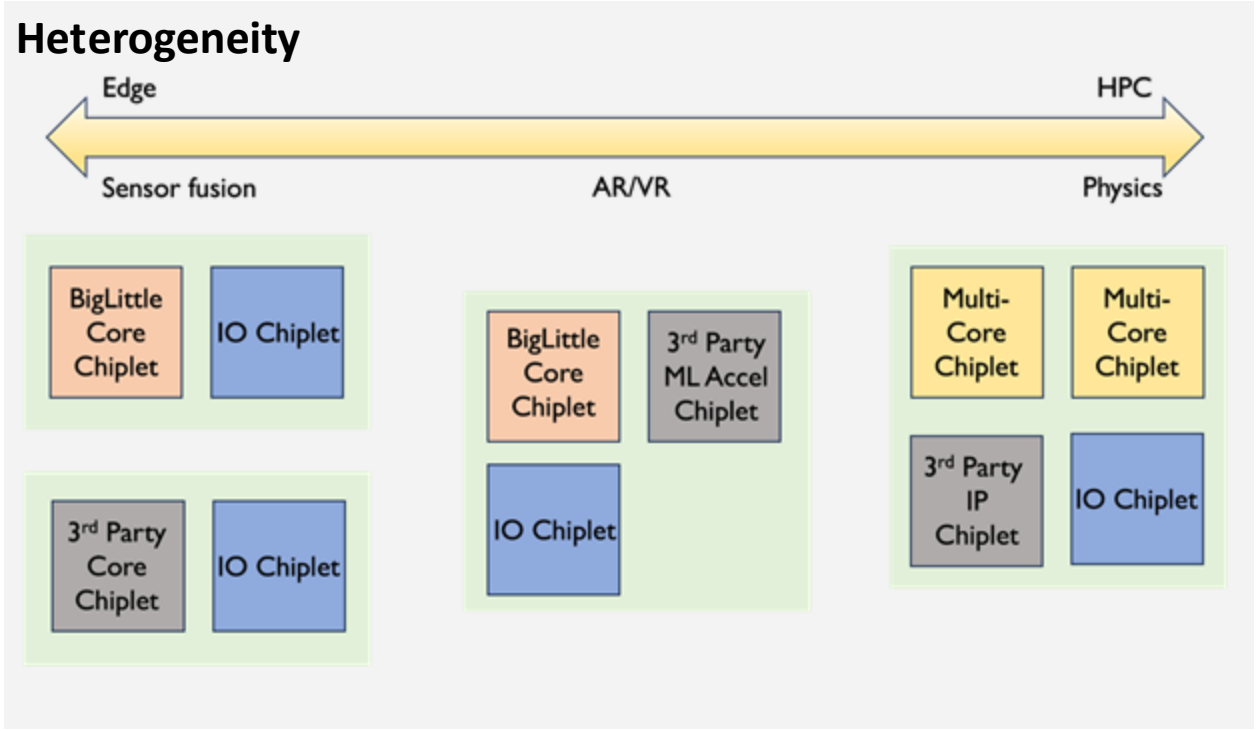
- Build a base chiplet which can be reused across multiple system configurations
- Base chiplet allows for reusability of IPs, with innovation focused on partner chiplets
- Base chiplet would consists of mix of CPUs, NPUs, Memory and D2D interfaces



Objective 2: Multi-chiplet configurations

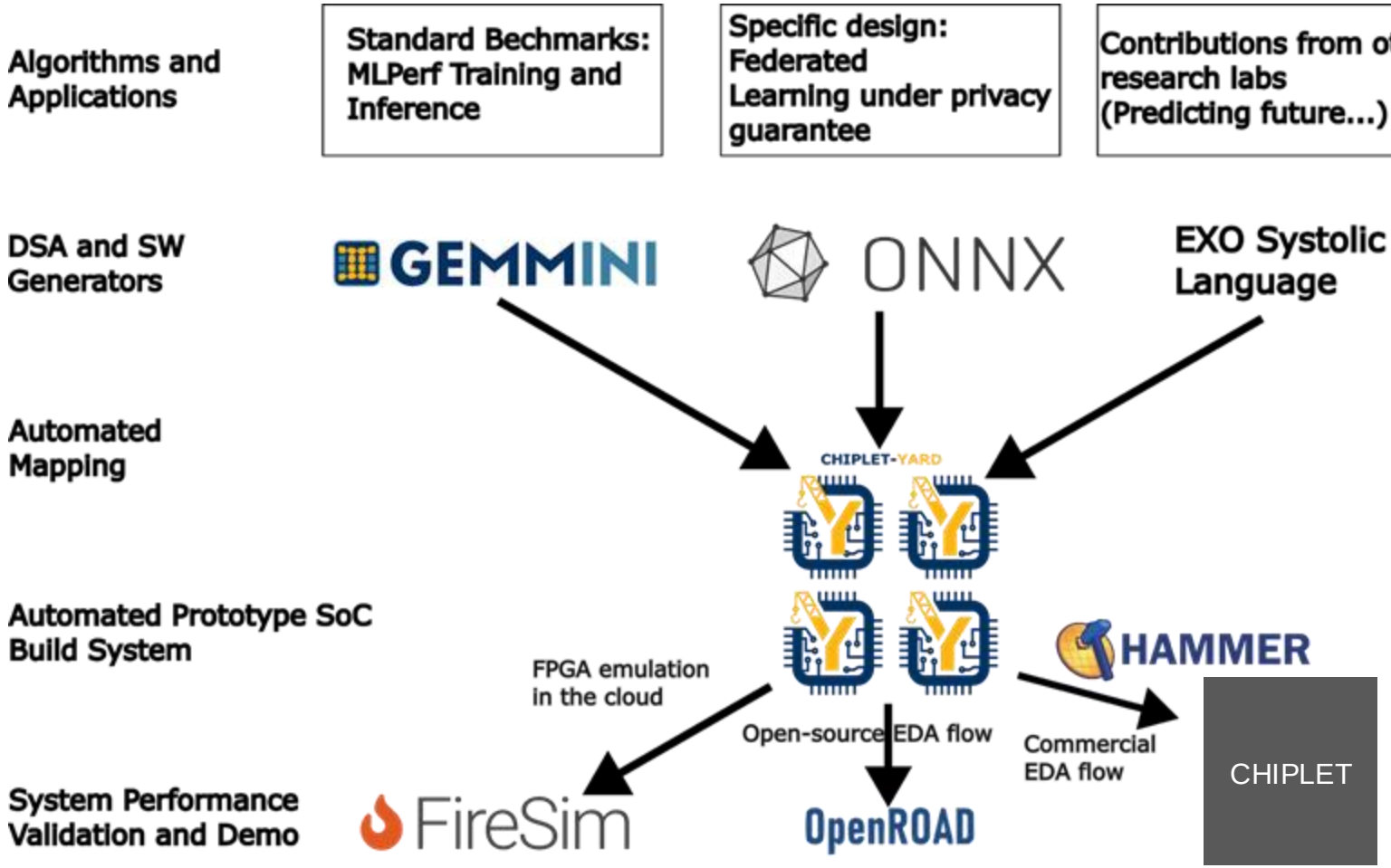
- Chiplet-yard will enable:
  - scaling of chiplets
  - broad spectrum of mix-and-match systems





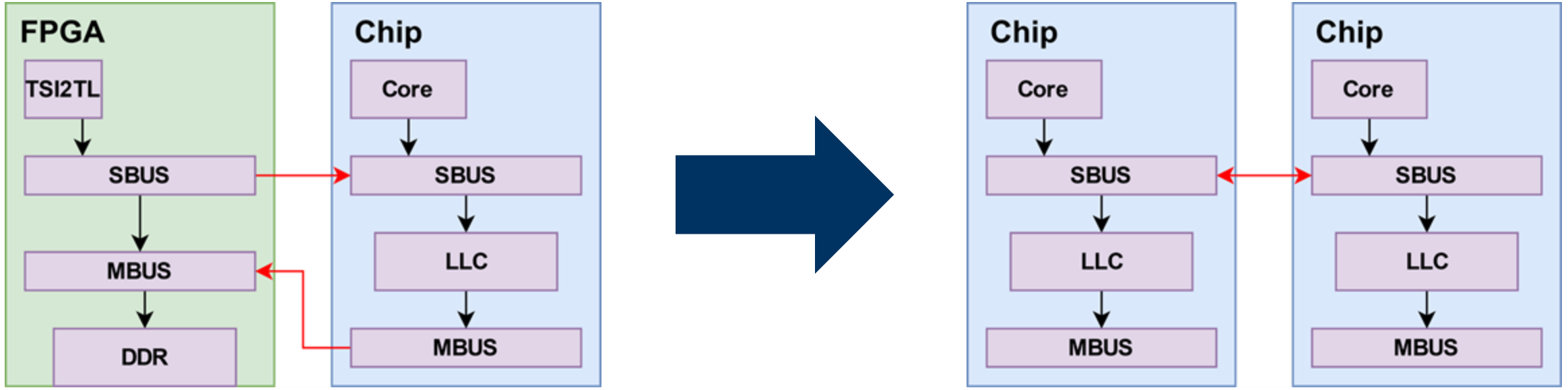
Objective 3: Full stack ecosystem

- Chiplet-yard will enable full stack ecosystem for rapid evaluation and SoP generation



Extending Chipyard for Chipletization

- Take advantage of existing IP
- Extend bringup infrastructure
  - Configurable bus connections off-chip
  - Make chip to FPGA APIs generic for multi-Chiptop



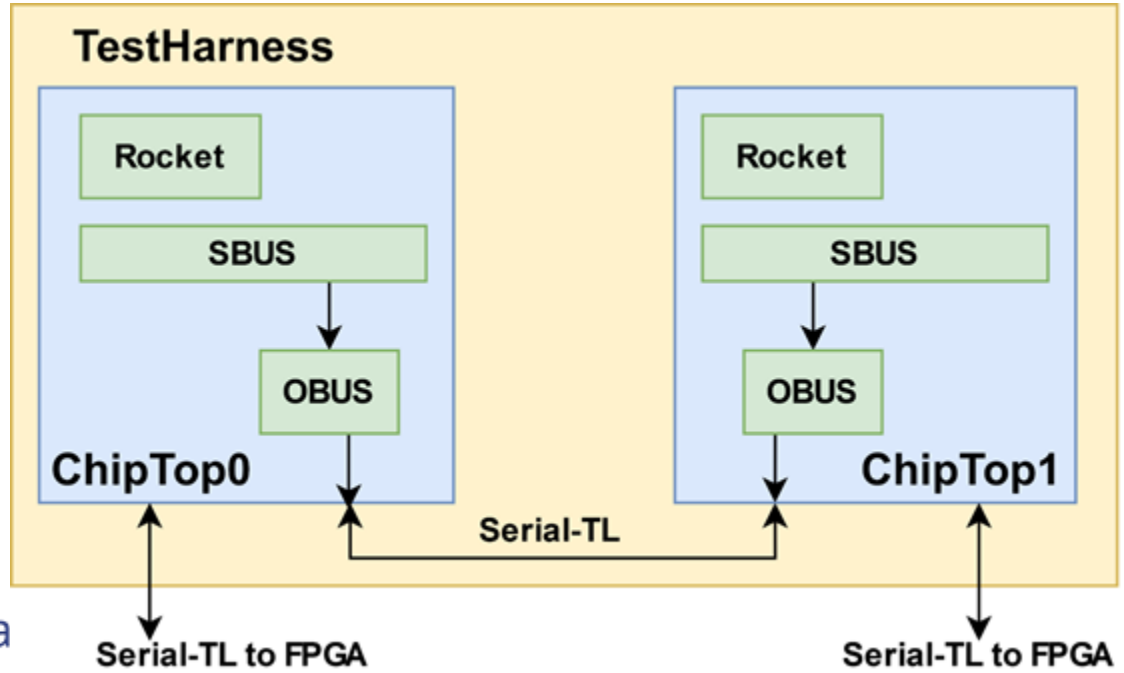
CHIPYARD
Berkeley Architecture Research

CHIPLET-YARD

Symmetric Config - Homogeneous chiplets

`$CYDIR/generators/chipyard/src/main/scala/config/ChipletConfigs.scala`

- 2 identical chips
  - 1 Rocket core
  - 2 Serial-TL Ports: chip<->ext mem, chip<->chip
  - Offchip bus (OBUS) off of system bus (SBUS)
- Each chip can access the memory of the other chip
- Manage 3 clock domains: chip-clock, outgoing-clock, incoming-clock
- Bring up multiple chips simultaneously



Summary

- We currently support multi-chip/chiplet configs and simulation in Chipyard
- Current configs are using non-coherent data exchange

Future works:

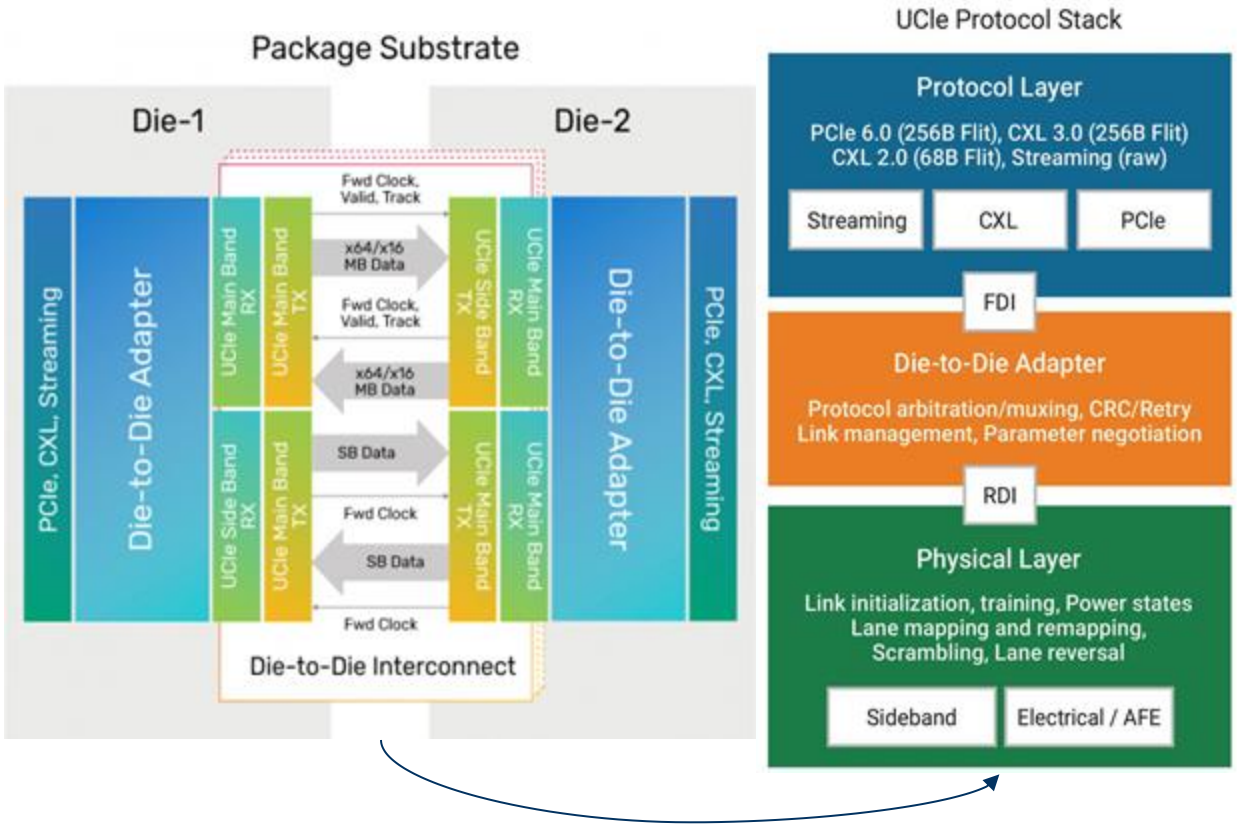
- Enable full cache coherency across chiplets
- Integrate UCIe for chip-to-chip communication
- Support chiplet simulation in FireSim
- Network-on-package generator

Agenda

- Hammer applications
- Overview of Hammer's abstractions
- Hammer community development
- Infrastructure for scale-out with chiplets
- Chiplet-yard for generating chiplets
- Die-to-die interface generators

UCIe Protocol Stack

- Layered protocol
  - Each layer performs distinct function
- Three layers
  - Protocol
  - Die-to-Die Adapter (Link Layer)
  - Physical Layer
- Layers are connected with standardized interfaces
  - Flit-aware D2D Interface (FDI)
  - Raw D2D Interface (RDI)



UCIe-lite

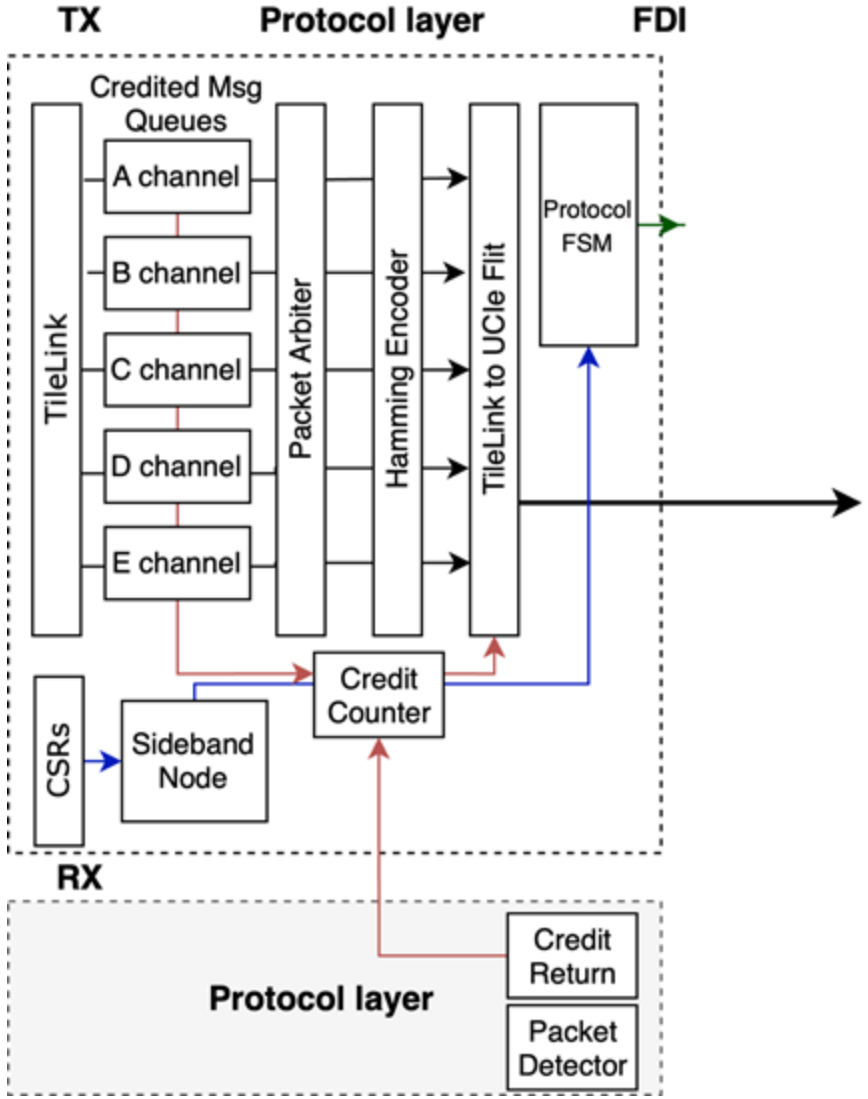
We are building generators for a reduced version of UCIe called UCI—lite

Incremental development: UCIe-lite will be extended to full UCIe in later versions

Protocol Layer	D2D Adapter	Physical Layer	Sideband
• **Streaming mode** • **Raw 64B flit mode** • **CRC added by protocol layer** • **Single protocol stack** • **Carries TileLink Uncached and Cached protocol** • No support for CXL/PCIe • No support for other flit modes	• **Stall req/ack** • **FDI/RDI State Machines** • **Link testing with parity** • **Retrain** • **LinkError** • **Dynamic clk gating** • **Power Management** • No ARB/MUX • No DLLP • No Flit cancel mechanism • No CRC and Retry	• **Link initialization/training** • **Byte to lane mapping for data transmission over Lanes** • **Transmitting and receiving sideband messages** • **Scrambling and training pattern generation** • **Free running clock mode only** • **Streaming mode only** • No PHY retrain • No lane reversal • No multi-module link	• **Full spec implementation** • **Reduced message set for streaming mode-raw only and for limited states of D2D/PHY** • **Three types of packets: Register access packets, Message without data, Message with data payload** • **Credit-based flow control** • **8ms timeout**

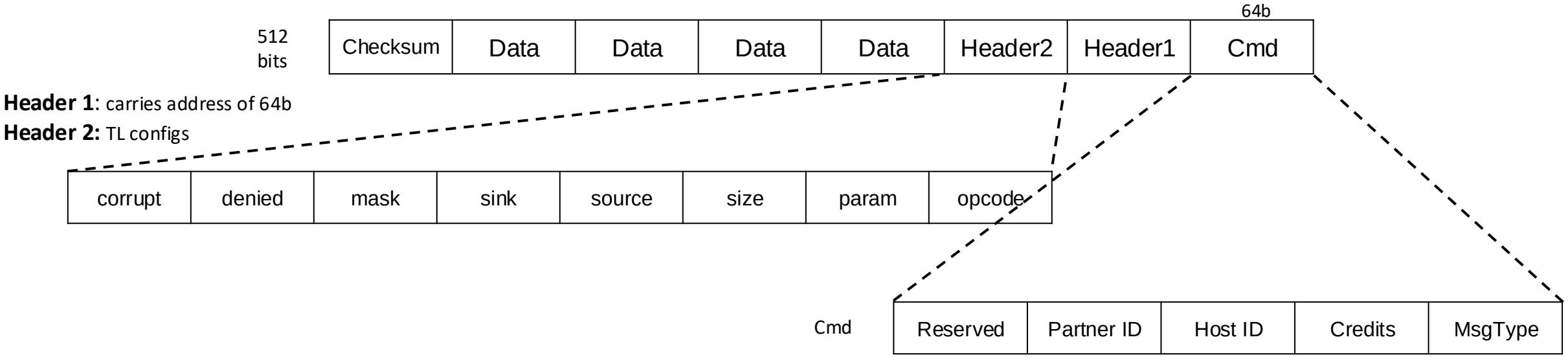
UCIe-Lite: Protocol Layer

- Protocol adapter connects to a bus/NoC in an SoC
- Sideband node uses memory-mapped registers for Sideband messaging
- We use TileLink as our frontend protocol, can be extended to any other bus protocols such as AXI, CHI, etc.
- Supports TL-Uncached and Cached packet translation
- Converts the TL messages to UCIe Raw Flit and sends over FDI interface to D2D adapter
- Adds checksum using hamming encoder/decoder
- Credit-based flow used to provide backpressure in multi-die systems



UCIe-Lite: Flit Format v1.0

- Initial version of UCIe Raw 64B flit format used in our implementation
- Reserved bits provided for future expansion



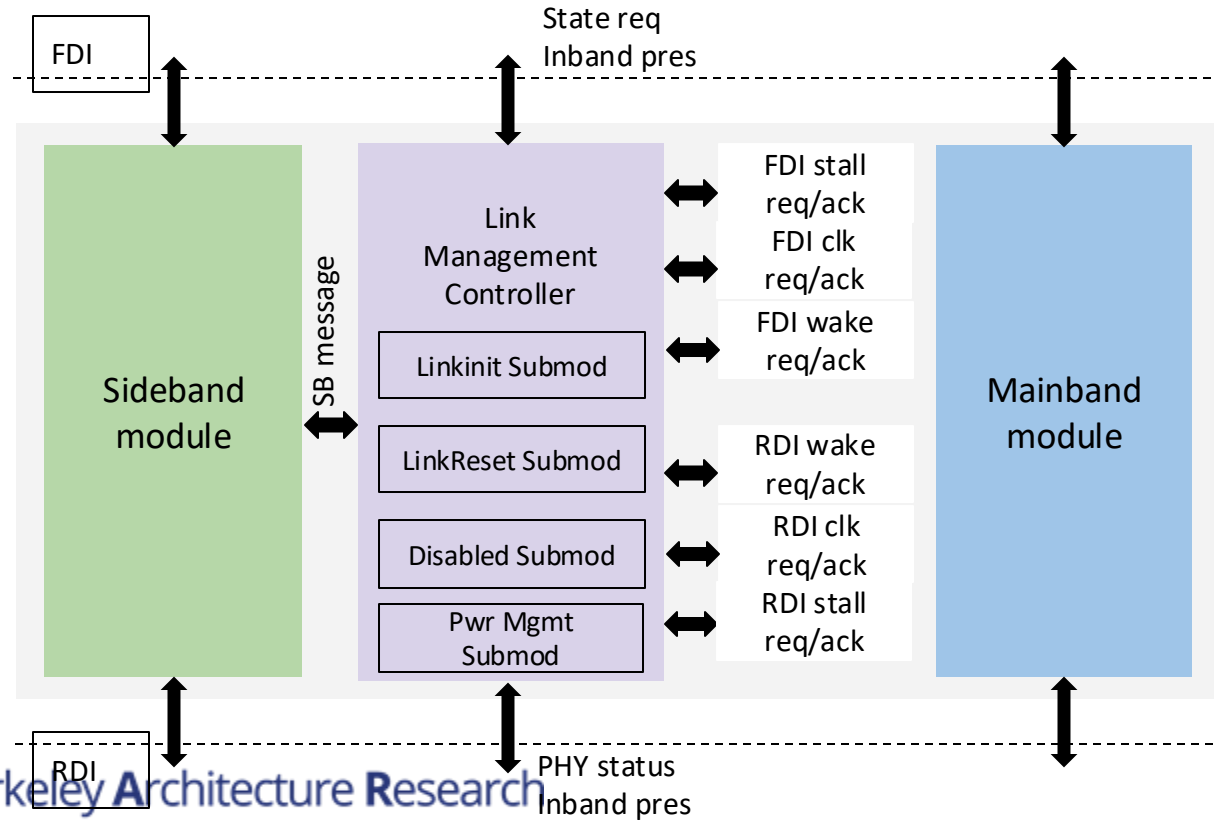
MsgType: 8 bits enum
Can be TL channel type, AXI, ReRoCC, Debug, reserved for future

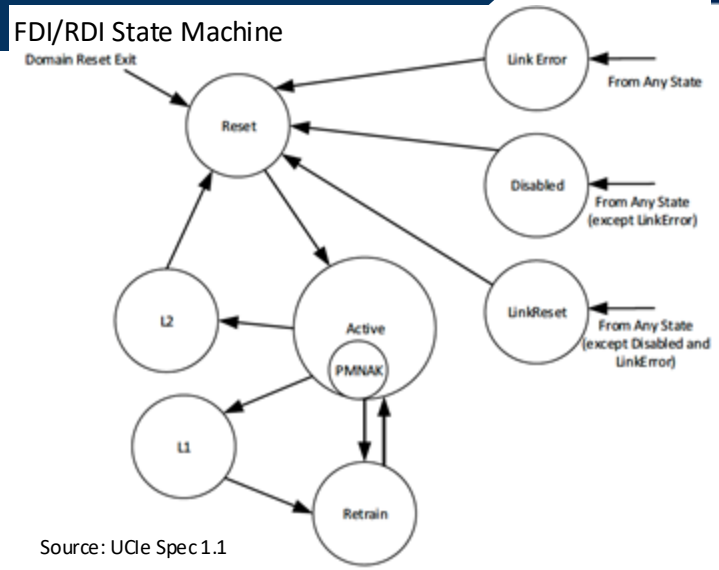
Credits: Carries the credit return for the different channels

Reserved in cmd can be used for QoS, discovery, etc.

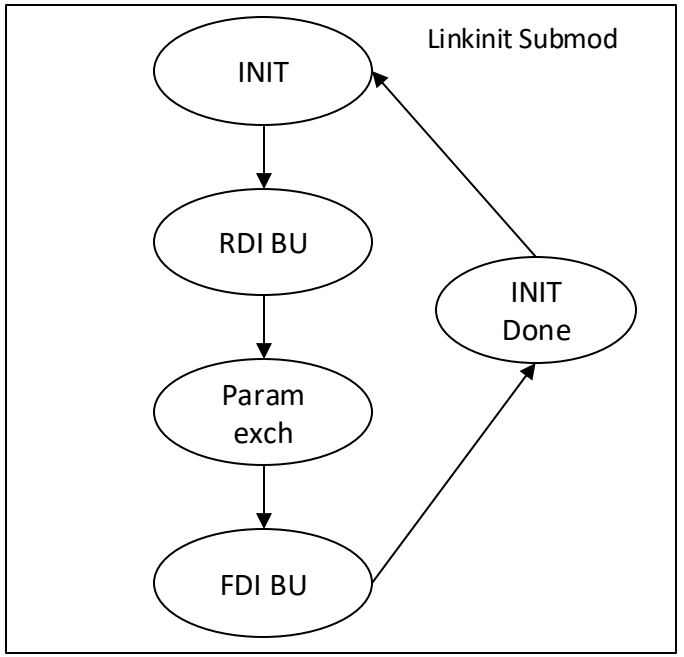
UCIe-Lite: D2D Adapter

- FDI/RDI state machines
- LinkInit module handles the Link Initialization steps of the UCIe specs
- Handles stall req/ack, clk req/ack and wake req/ack





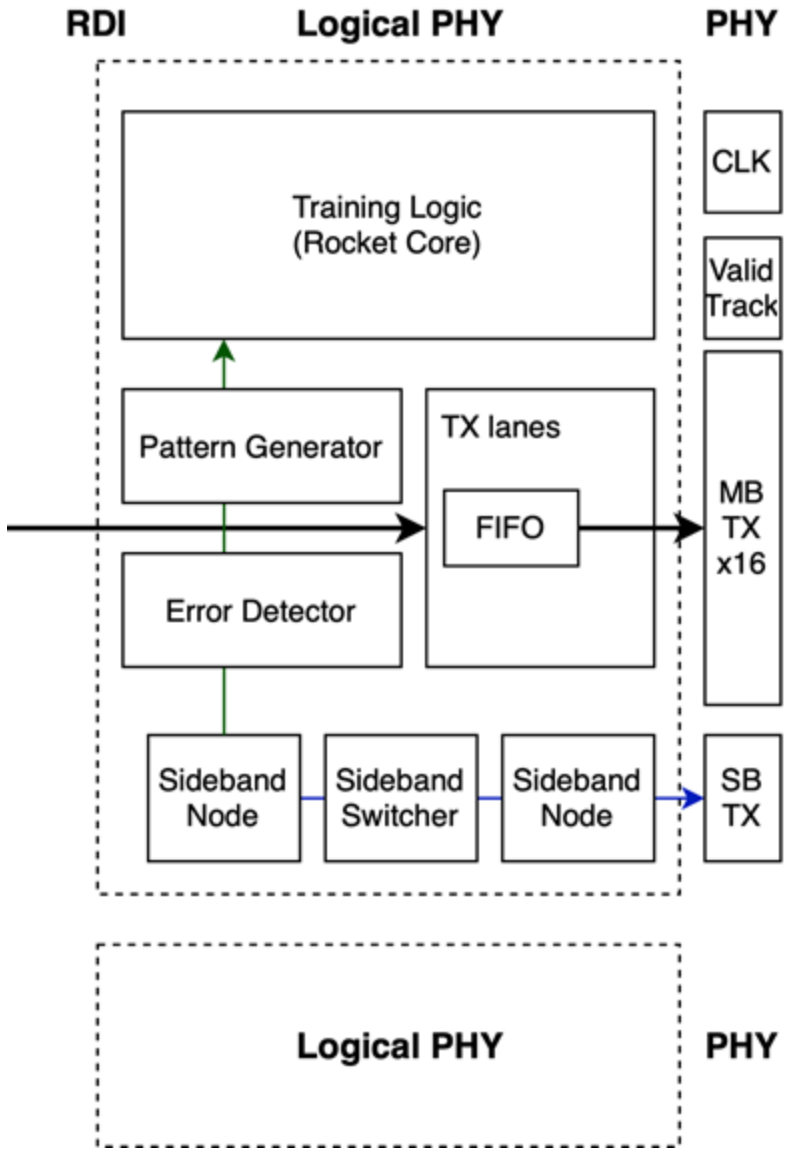
Source: UCIe Spec 1.1



Berkeley Architecture Research

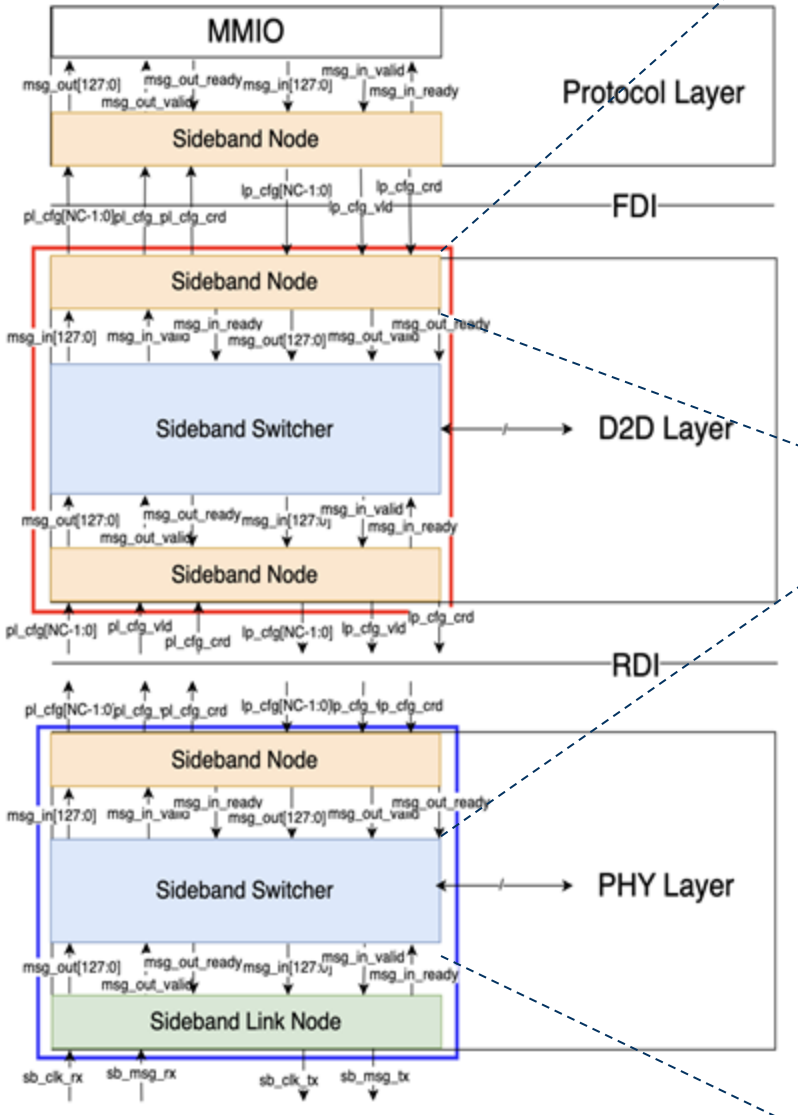
UCIe-Lite: Logical PHY

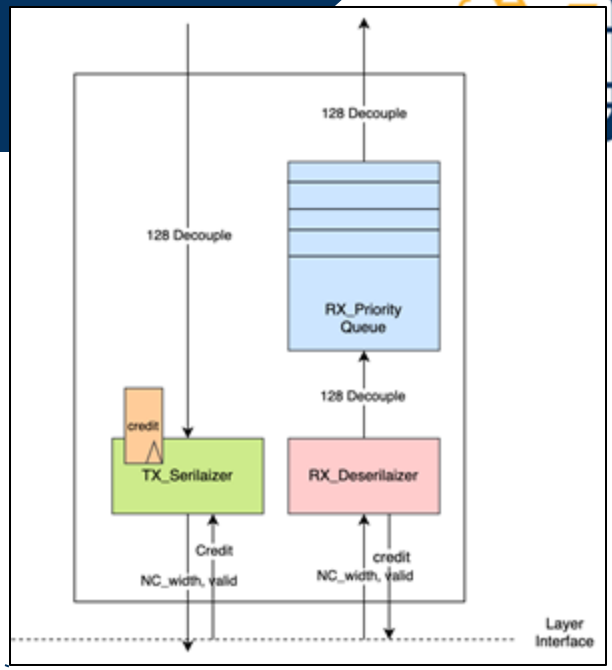
- Maps bytes to lane
- Link Training – we use a RISC-V core to trigger and handle training parameters
  - Link training parameters are stored as MMIO registers for the core to probe and update
- Link Initialization state machines for mainband and sideband
- Pattern generators for testing and scrambler functionality
- Error detector for linktraining

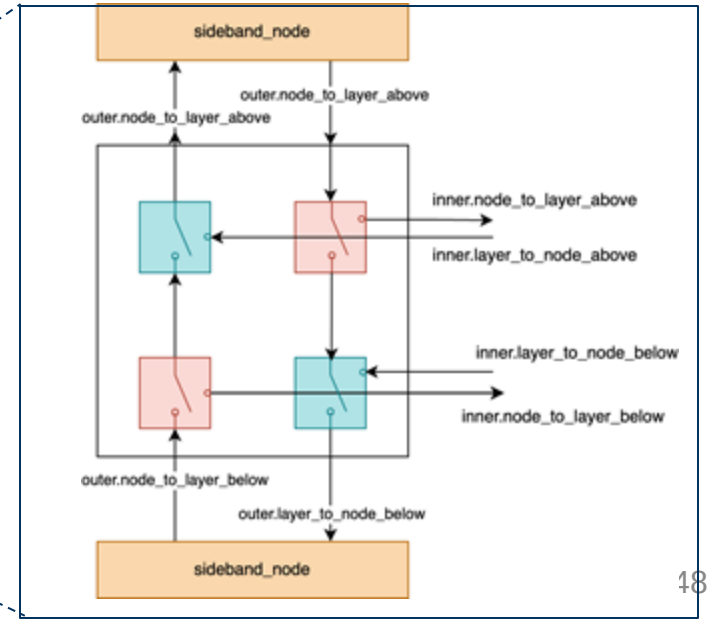


UCIe-Lite: Sideband

- Sideband module needs to be on every layer to send sideband messages
- Sideband node handles serdes and credit flow of SB packets
- Sideband switcher controls flow of messages to submodules or stack transfer

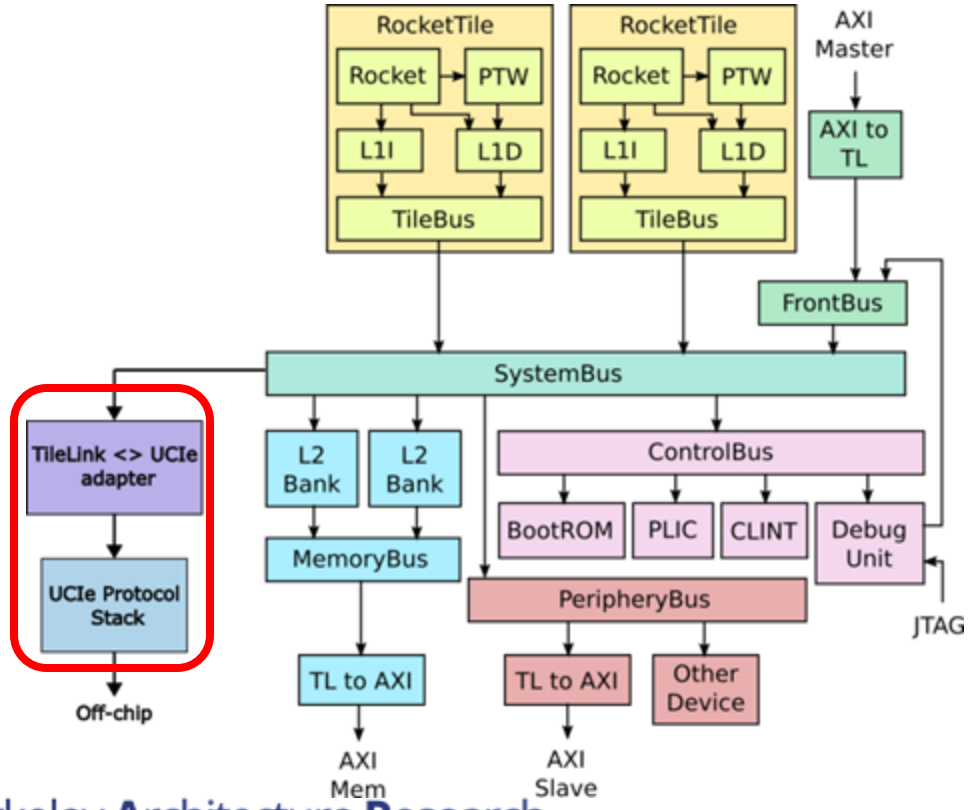


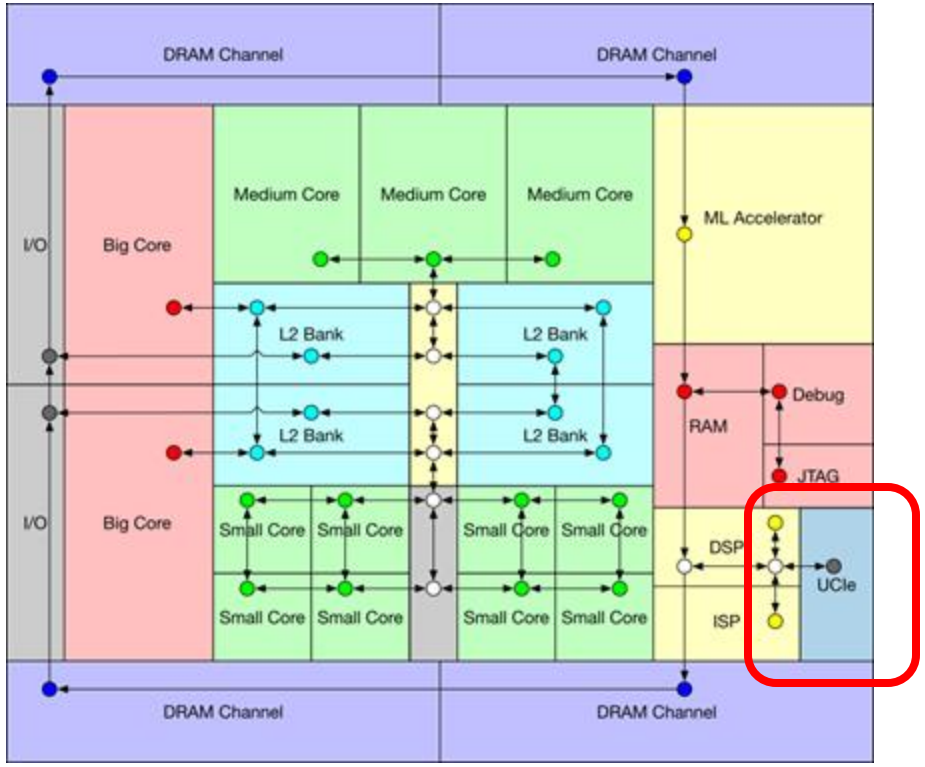




Integration into Chipyard

- UCIe stack can be attached to Sbus/Obus in Chipyard SoC
- It can also be instantiated as a Tile in Constellation NoC





Summary

- Die-to-Die interface are vital to enable performant and energy-efficient chiplet systems
- Open standards helps build an open ecosystem
- We want to provide an open-source generator for UCIe controller and PHY, to enable academia/industry to leverage this ecosystem
- In doing so, we want to showcase some interesting prototypes of heterogeneous chiplet systems

- UCIe-digital: https://github.com/ucb-ucie/uciedigital
- UCIe-analog: https://github.com/ucb-ucie/ucieanalog

UCIe-digital

UCIe-Analog