Hammer VLSI Flow

Jerry Zhao

UC Berkeley
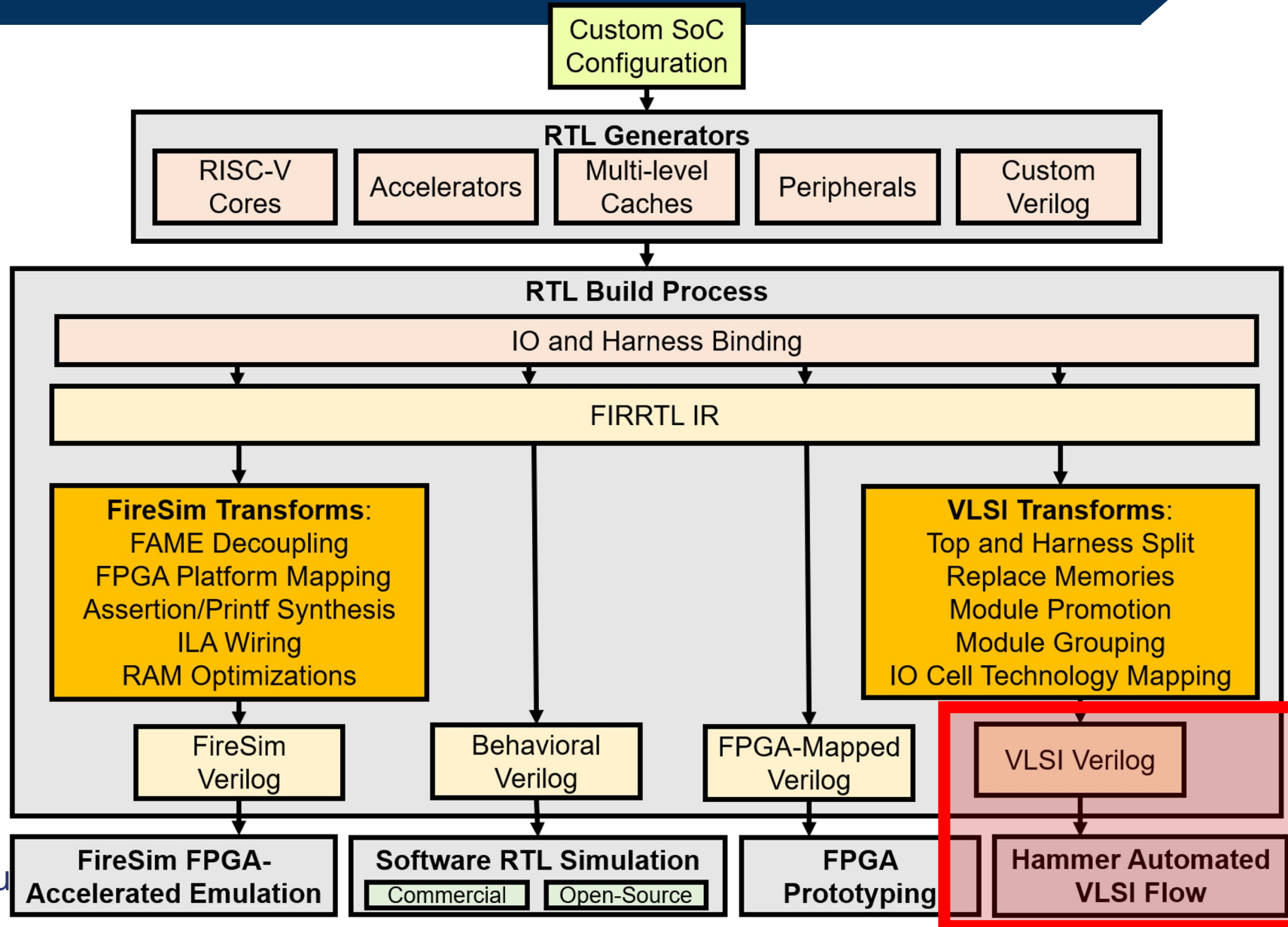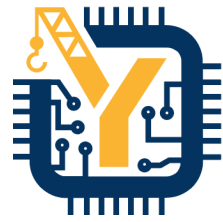
jerryz123@berkeley.edu
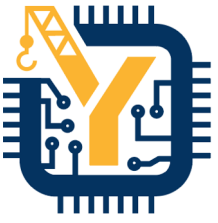
Berkeley Architecture Research

CHIPYARD

Tutorial Roadmap

Custom SoC Configuration

RTL Generators
- RISC-V Cores
- Accelerators
- Multi-level Caches
- Peripherals
- Custom Verilog

RTL Build Process

IO and Harness Binding

FIRRTL IR

FireSim Transforms:
FAME Decoupling
FPGA Platform Mapping
Assertion/Printf Synthesis
ILA Wiring
RAM Optimizations

VLSI Transforms:
Top and Harness Split
Replace Memories
Module Promotion
Module Grouping
IO Cell Technology Mapping

FireSim Verilog

Behavioral Verilog

FPGA-Mapped Verilog

VLSI Verilog

FireSim FPGA-Accelerated Emulation

Software RTL Simulation
Commercial   Open-Source

FPGA Prototyping

Hammer Automated VLSI Flow

Berkeley Architectu

Goals

- Hammer applications

- Fixing traditional VLSI flows: One size doesn't fit all

- Overview of Hammer's abstractions
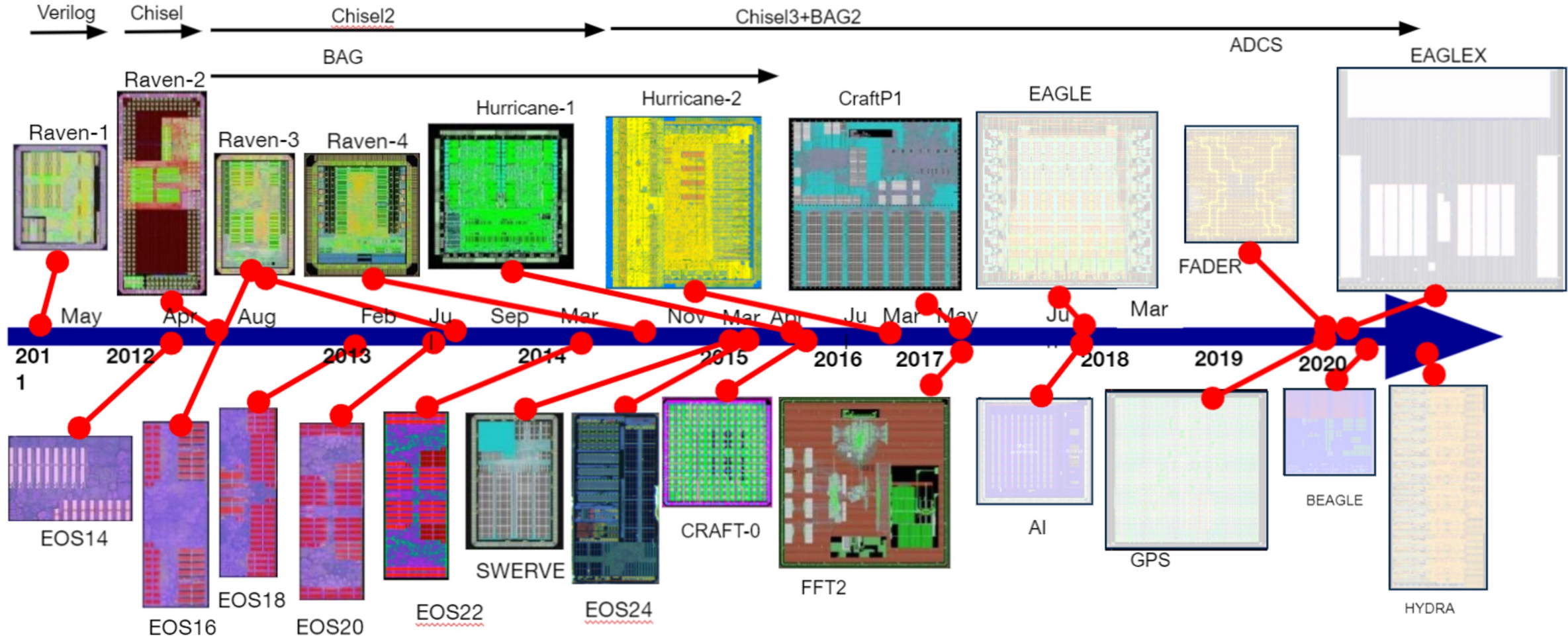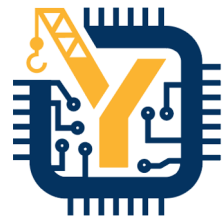
- Hammer open development and growth

Berkeley **A**rchitecture **R**esearch

Goals

- **Hammer applications**

- Fixing traditional VLSI flows: One size doesn't fit all

- Overview of Hammer's abstractions

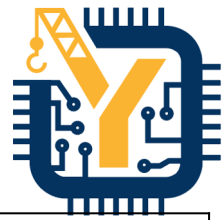- Hammer development model

Berkeley Architecture Research

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
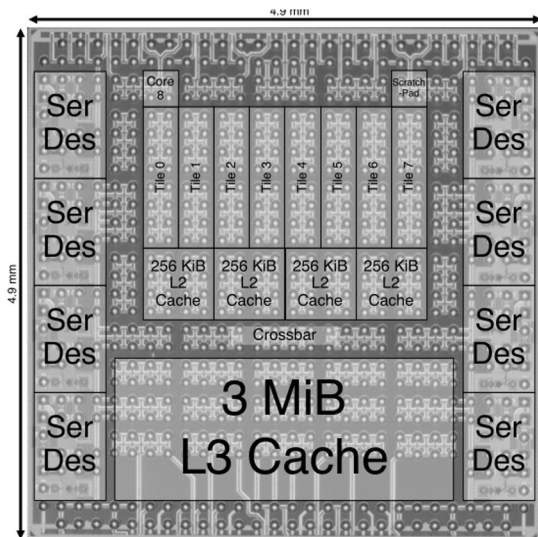
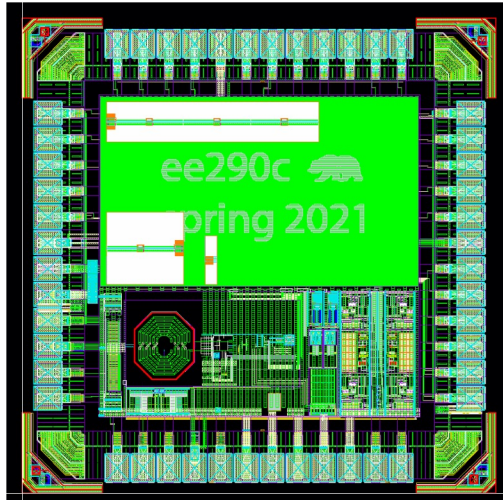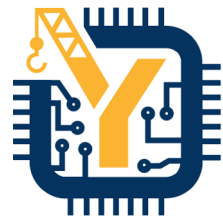
Eagle [1]
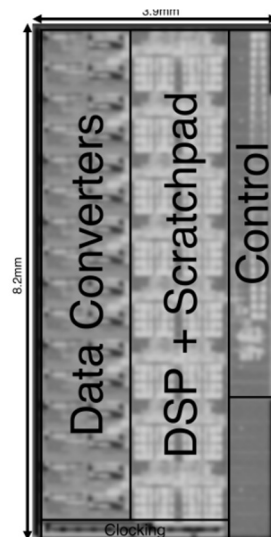
WaterSerpent
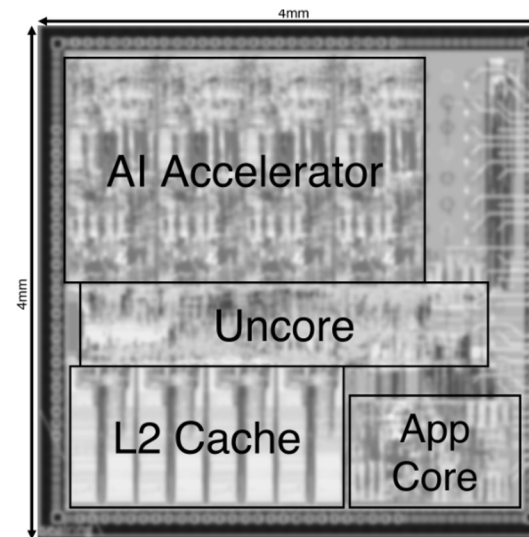
MythicChip

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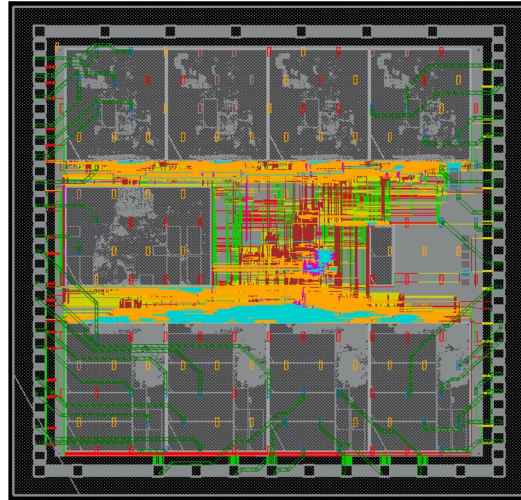[2] D. Fritchman et. al, *IEEE SSCS Magazine, Spring 2022*

Hammer in Courses

- Introduced in undergraduate digital circuits and systems labs:
  - http://github.com/EECS150 (ASAP7 and Sky130 plugins)
- Special topics 'tapeout' class
  - Spring 2022: 1 sophomore, 15 juniors, 19 seniors, 3 5th-yr MS, 2 MEng, 1 PhD

2021 EE194/290C: OsciBear
TSMC 28nm

2022 EE194/290C: BearlyML (left) & SCuM-V (right)
Intel 16nm
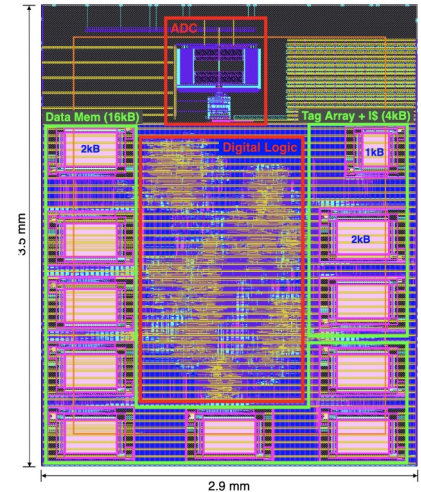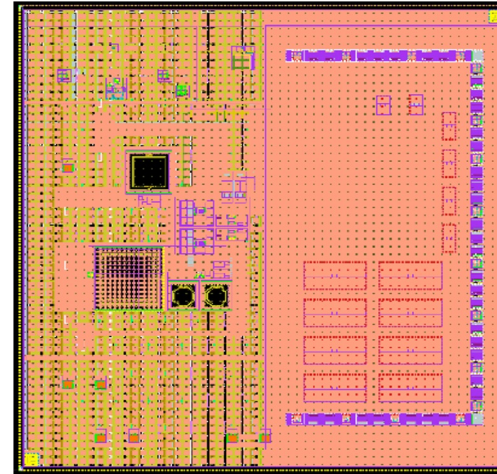
Sky130 MPW-2
Skywater 130nm

Berkeley Architecture Research

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

Berkeley **A**rchitecture **R**esearch

Increasing Flow Reusability

Legend: Design LoC, Tech plugin LoC, Hammer + Tool plugins LoC, % unique LoC

Y-axis: LoC (x1000, % unique)

X-axis: Chip (Time →)

Chips: Eagle, HugeFlyingSoC, NavRx, WaterSerpent, MythicChip, OsciBear (28), OsciBear (130), HDBinaryCore, BearlyML, SCuM-V

% unique LoC values: 24.7, 27.9, 21.3, 10, 7.9, 0.8, 0.7, 3.3, 2.5, 1.4

Berkeley Architecture Research
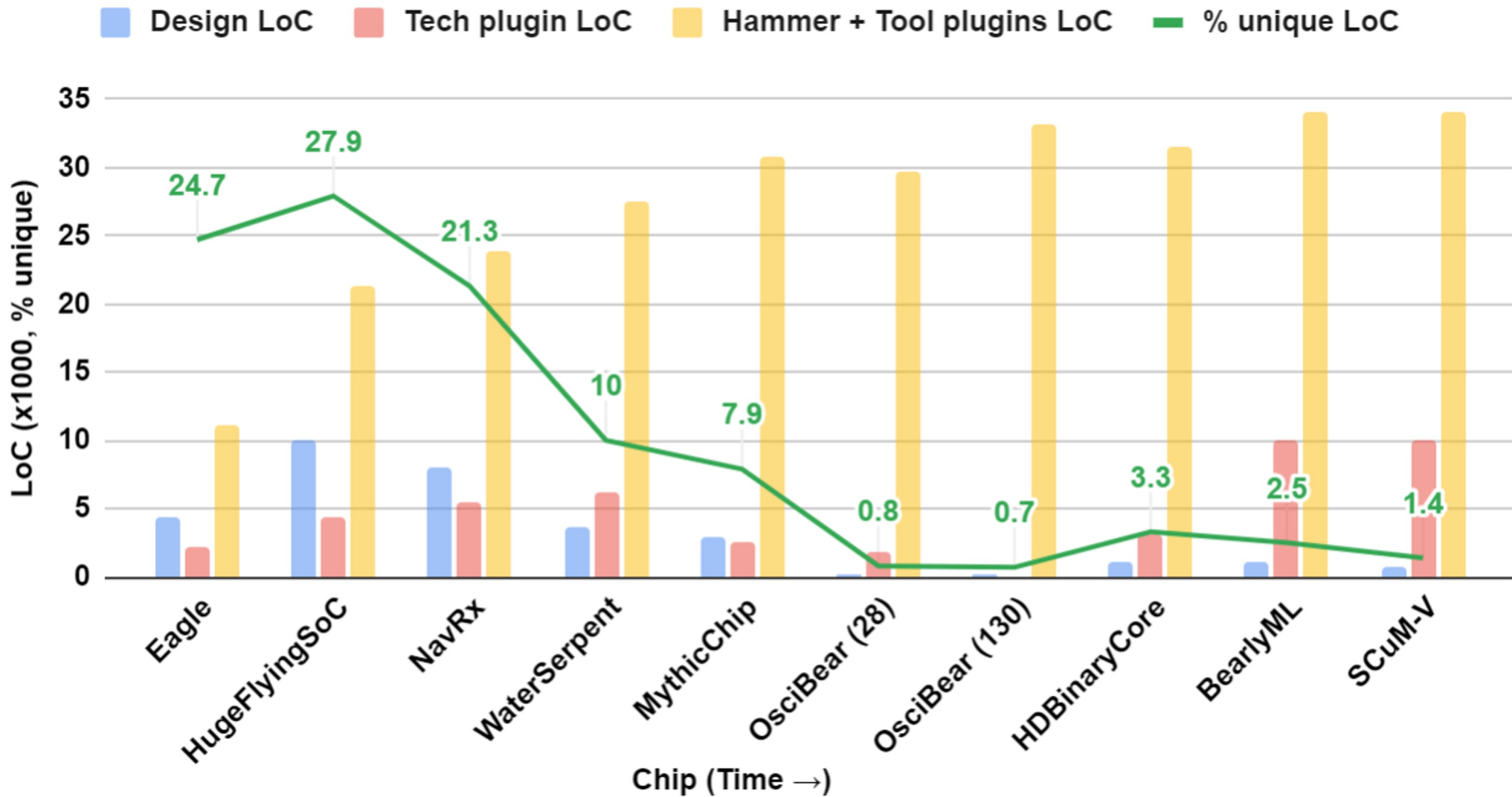
Tech Plugin Reusability
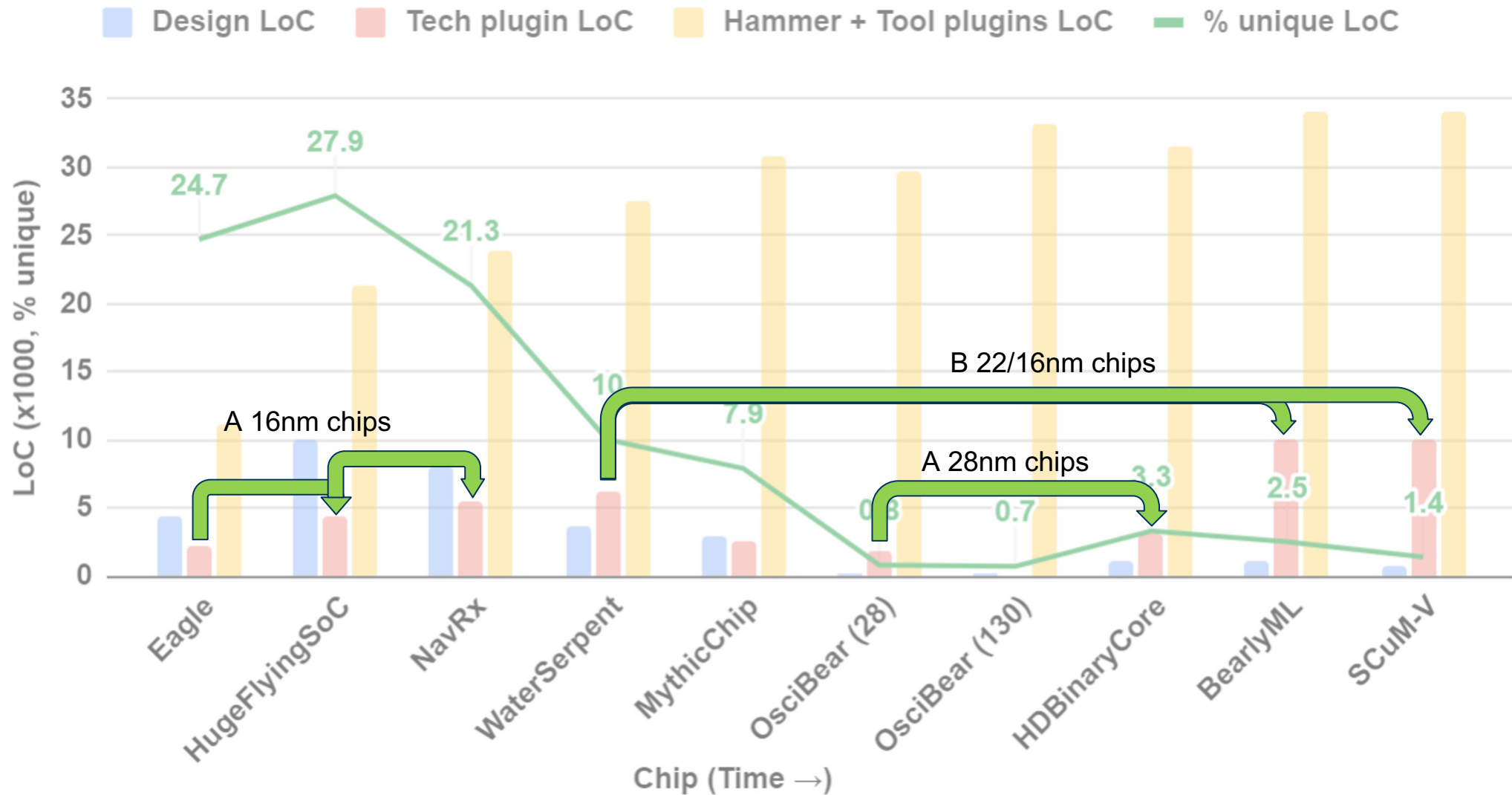
Legend: Design LoC · Tech plugin LoC · Hammer + Tool plugins LoC · % unique LoC

Y-axis: LoC (x1000, % unique)

X-axis: Chip (Time →) — Eagle, HugeFlyingSoC, NavRx, WaterSerpent, MythicChip, OsciBear (28), OsciBear (130), HDBinaryCore, BearlyML, SCuM-V

% unique LoC values: 24.7, 27.9, 21.3, 10, 7.9, 0.8, 0.7, 3.3, 2.5, 1.4
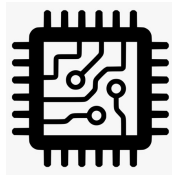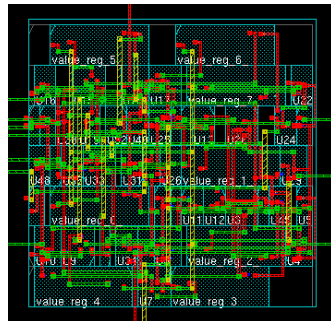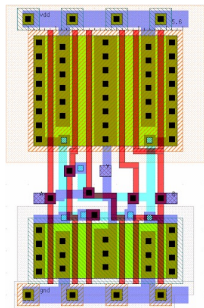
A 16nm chips
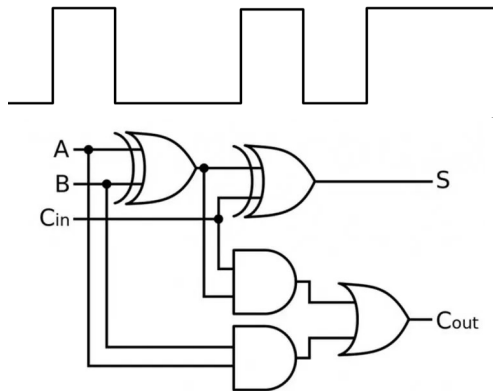B 22/16nm chips
A 28nm chips

Goals

- Hammer at 10,000 feet

- Fixing traditional VLSI flows: One size doesn't fit all

- Overview of Hammer's abstractions

- Hammer development model

Berkeley Architecture Research

ASIC Flow
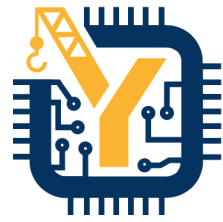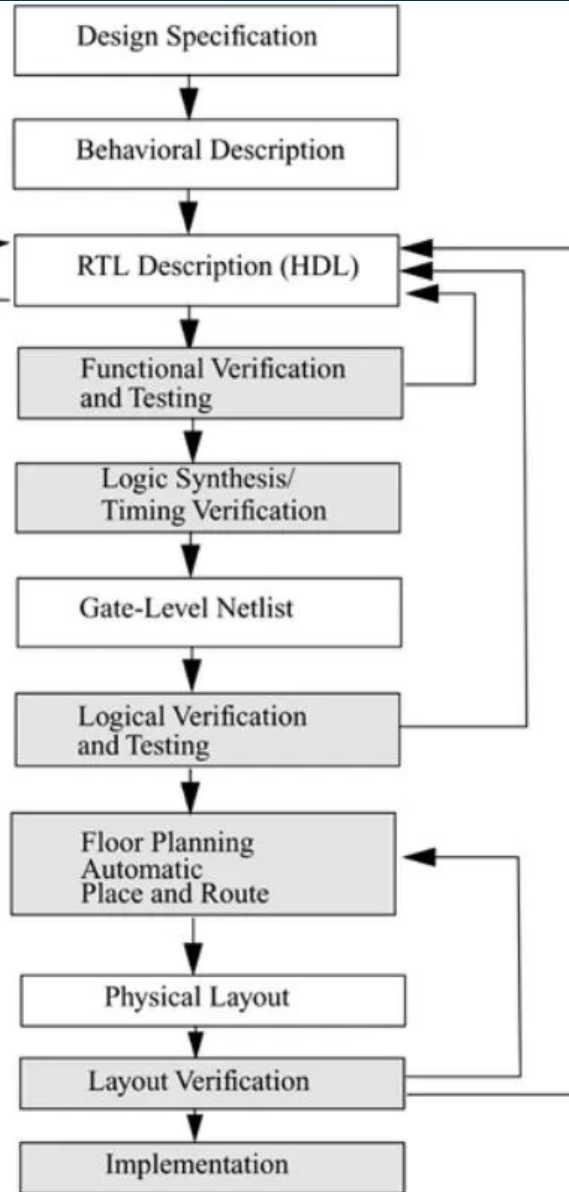
synthesis

place-and-route

DRC

LVS

14

Components of VLSI Flows

Design

- RTL
- IP cores
- Floorplan
- Constraints
- …

Berkeley Architecture Research

Components of VLSI Flows

Design

Tech

- RTL
- IP cores
- Floorplan
- Constraints
- …

- Intel 16
- TSMC N5
- SkyWater 130nm
- …

Berkeley Architecture Research

Components of VLSI Flows

Design

- RTL
- IP cores
- Floorplan
- Constraints
- …

Tech

- Intel 16
- TSMC N5
- SkyWater 130nm
- …

Tools

- Synopsys VCS
- Cadence Genus
- OpenROAD
- …
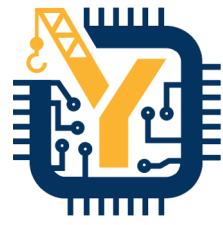
Berkeley **A**rchitecture **R**esearch

Why is Physical Design so Hard?

VLSI flows *are custom built for each project*

Berkeley **A**rchitecture **R**esearch

Why is Physical Design so Hard?

VLSI flows *are custom built for each project*

- Flows tailored to design/tech/tool combo
  - Different design? New constraints…
  - Switching tech? New rules, IP, SRAMs, …
  - Updated tool? Different commands…

Design Concerns

Tool Concerns

Technology Concerns

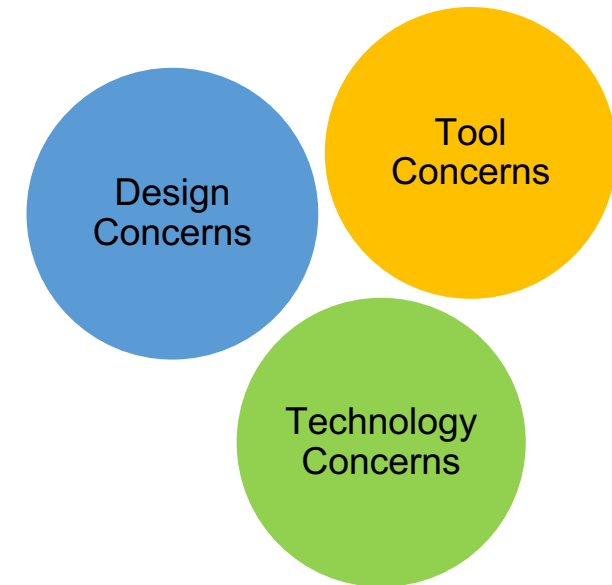**B**erkeley **A**rchitecture **R**esearch

Why is Physical Design so Hard?

VLSI flows *are custom built for each project*

- Flows tailored to design/tech/tool combo
  - Different design? New constraints…
  - Switching tech? New rules, IP, SRAMs, …
  - Updated tool? Different commands…
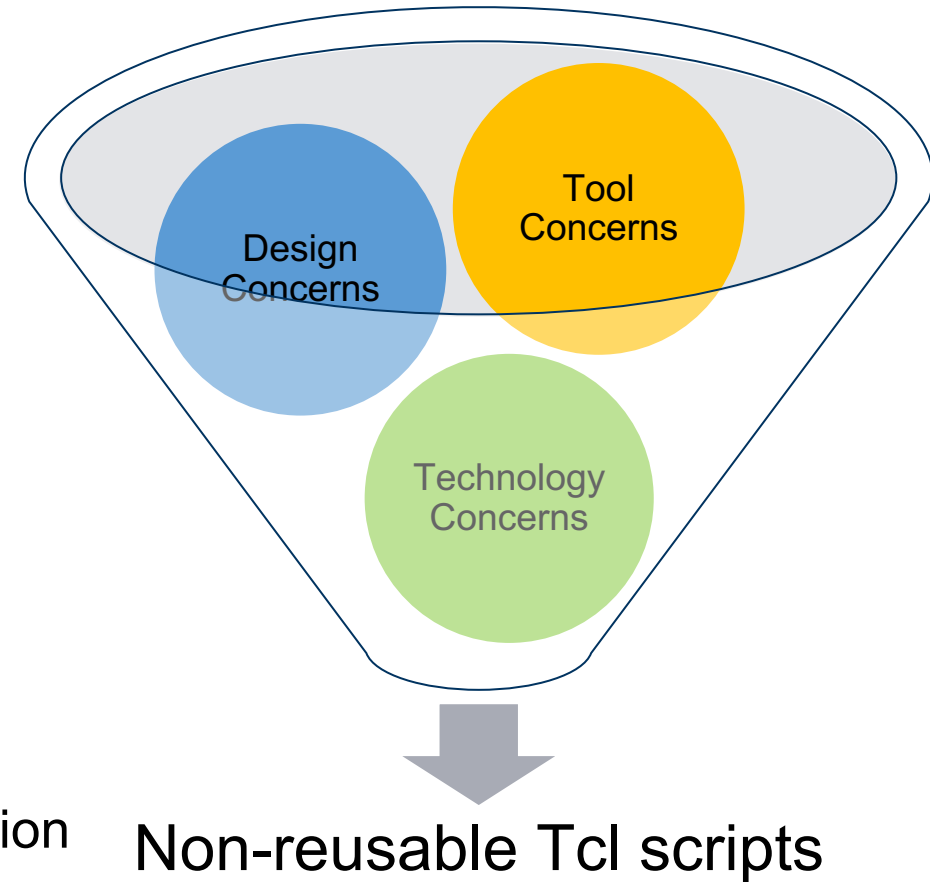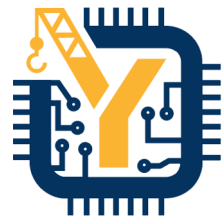- Design intent & expertise tied up in scripts!
  - Today, wide range of technology options

  and domain specialization demand architectural exploration

Non-reusable Tcl scripts

Example: Power Straps

- How to distribute power from package to transistors

From chip package

GND VDD GND VDD

M_{N+1}

GND

VDD

GND

M_N

To circuits
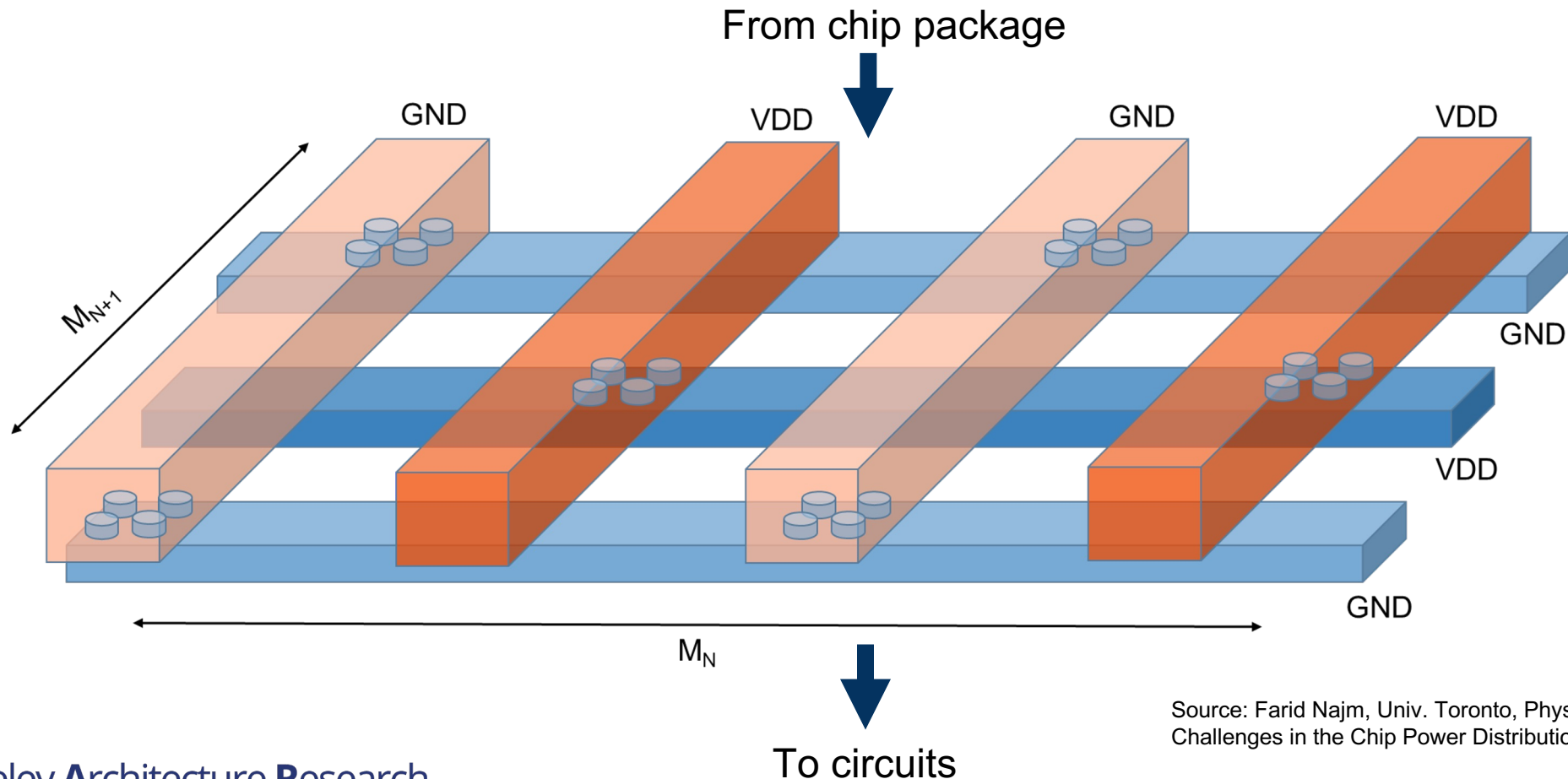
Source: Farid Najm, Univ. Toronto, Physical Design Challenges in the Chip Power Distribution Network

Example

- Hypothetical power strap creation command:


```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
  -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
  -area [get_bbox -of ModuleABC]  \
  -start [expr [lindex [lindex [get_bbox -of ModuleABC] 0] 0] + 1.234]

# Repeat for each layer!
```

Example

- Hypothetical power strap creation command:


```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
    -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
    -area [get_bbox -of ModuleABC]   \
    -start [expr [lindex [lindex [get_bbox -of ModuleABC] 0] 0] + 1.234]

# Repeat for each layer!
```
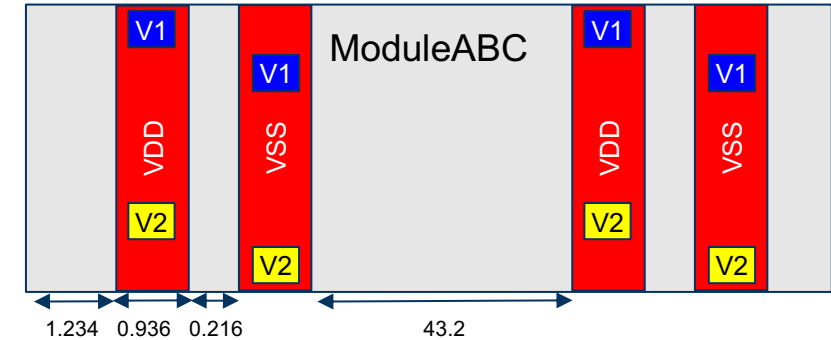
Tool-specific

The command + options

Example
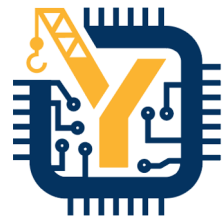
- Hypothetical power strap creation command:


```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
    -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
    -area [get_bbox -of ModuleABC]  \
    -start [expr [lindex [lindex [get_bbox -of ModuleABC] 0] 0] + 1.234]

# Repeat for each layer!
```

Tool-specific

The command + options

Tech-specific

DRC/ERC-compliant dimensions

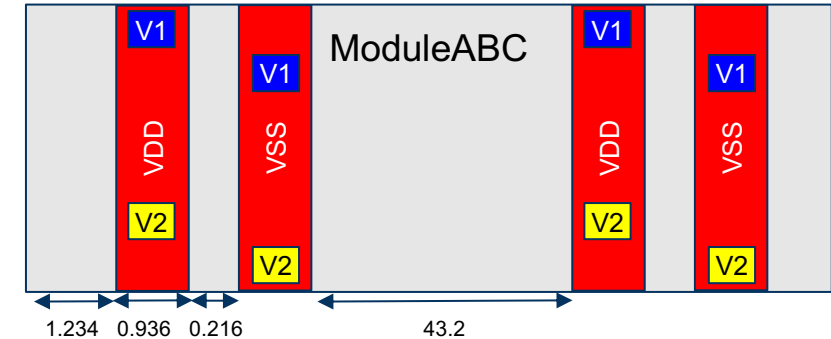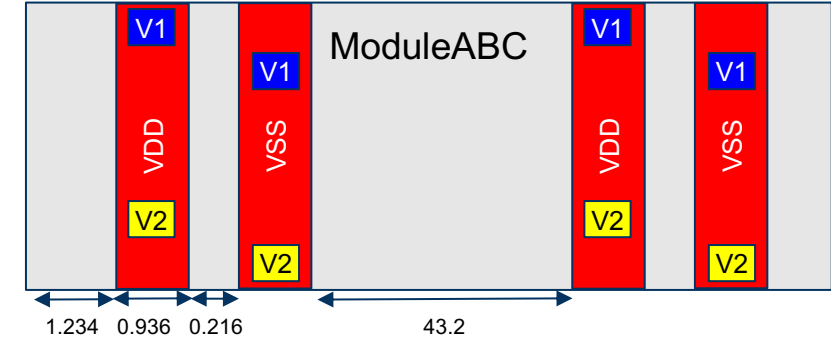**B**erkeley **A**rchitecture **R**esearch

Example

- Hypothetical power strap creation command:


```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
    -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
    -area [get_bbox -of ModuleABC]   \
    -start [expr [lindex [lindex [get_bbox -of ModuleABC] 0] 0] + 1.234]

# Repeat for each layer!
```
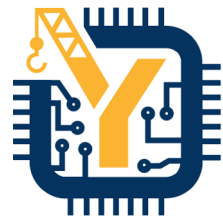
Tool-specific	Tech-specific	Design-specific
The command + options	DRC/ERC-compliant dimensions	Layers, power domains, floorplan, density

Berkeley Architecture Research

Goals

- Hammer applications

- Fixing traditional VLSI flows: One size doesn't fit all

- Overview of Hammer's abstractions

- Hammer example

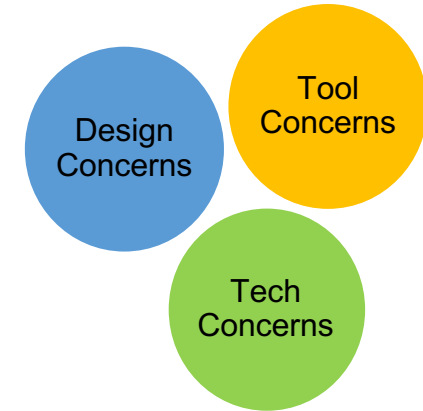Berkeley Architecture Research

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

Design Concerns

Tool Concerns

Tech Concerns

JSON/YAML

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

Design Concerns

Tool Concerns

Tech Concerns

JSON/YAML

OpenROAD

Sky130

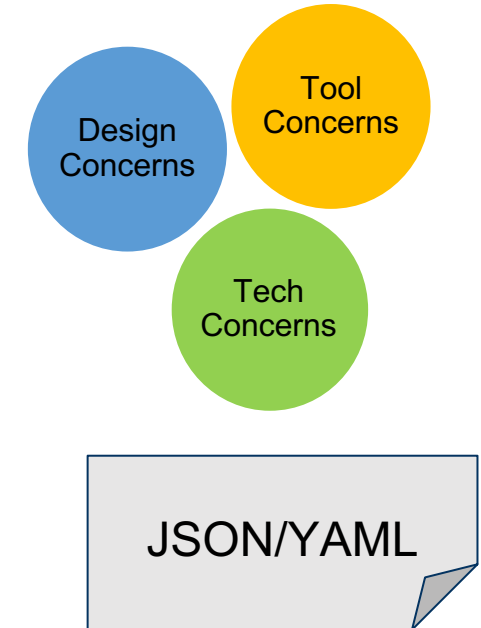Berkeley Architecture Research
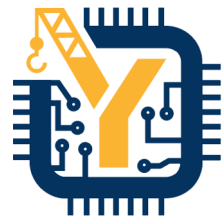
Hammer Design Principles
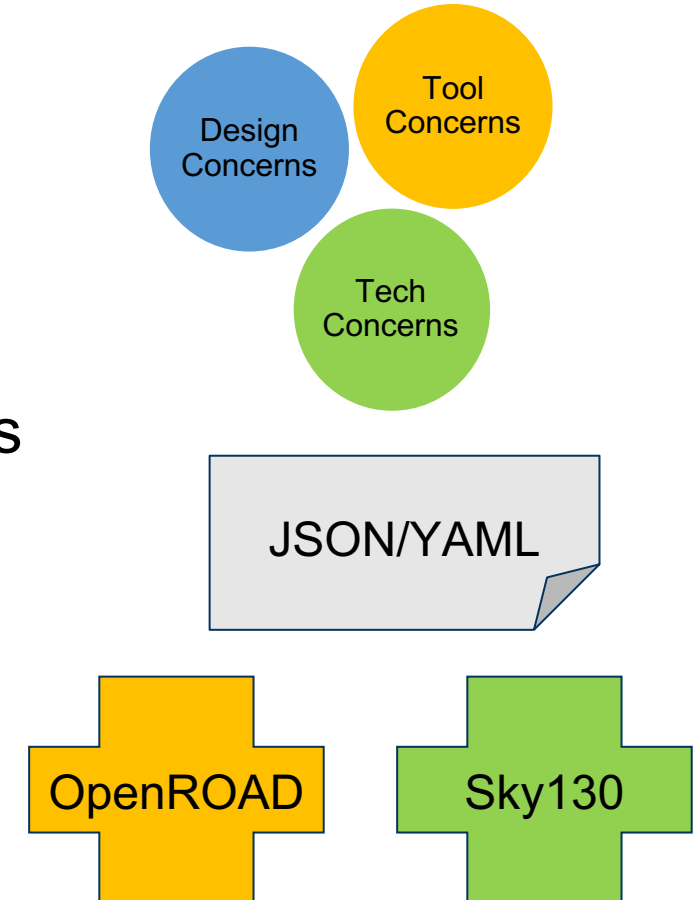
1. **Separation of Concerns**

   - 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
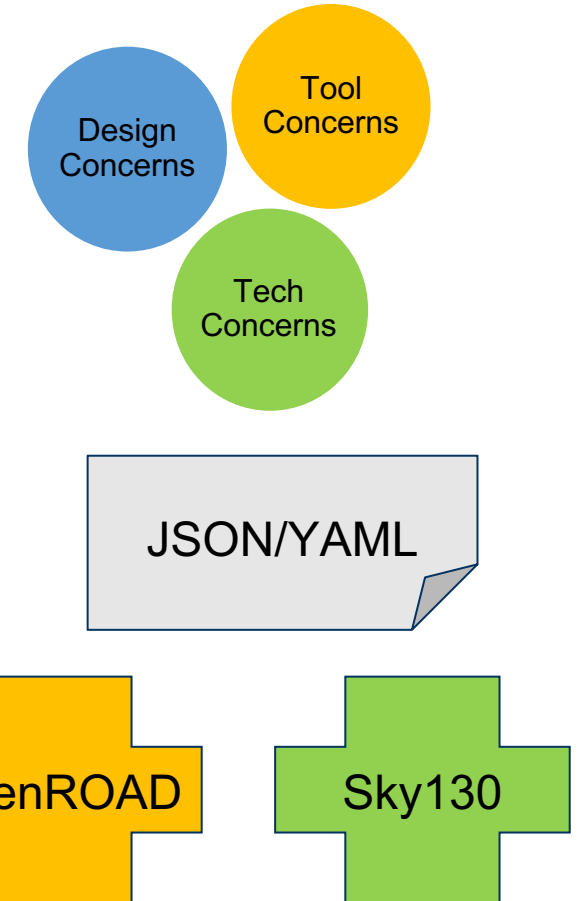
Design Concerns

Tool Concerns

Tech Concerns

JSON/YAML

OpenROAD

Sky130

my_custom_tcl

Berkeley **A**rchitecture **R**esearch
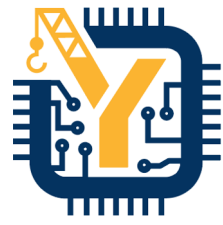
What is Hammer?

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

Berkeley Architecture Research

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!

Berkeley **A**rchitecture **R**esearch

What is Hammer?
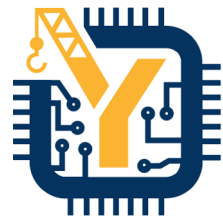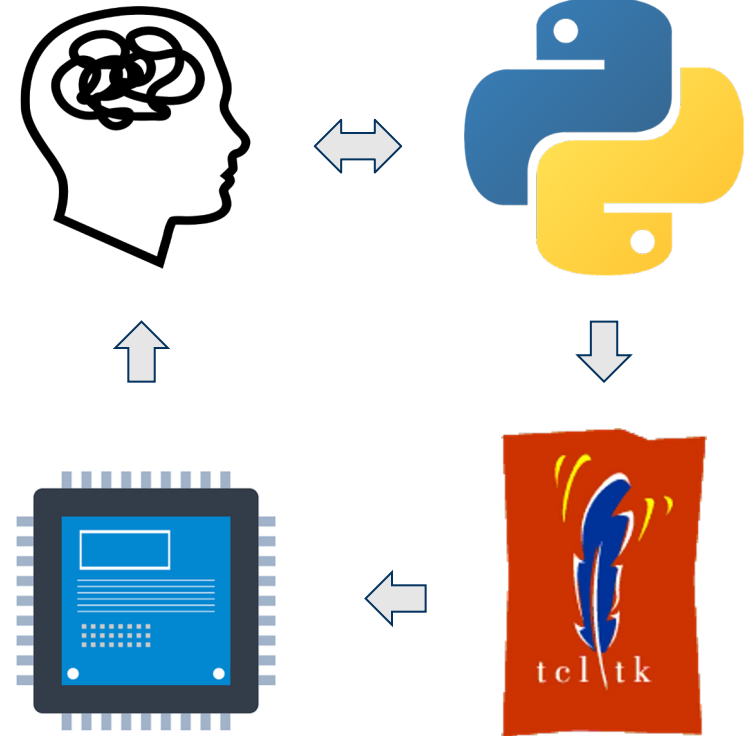
Hammer is:
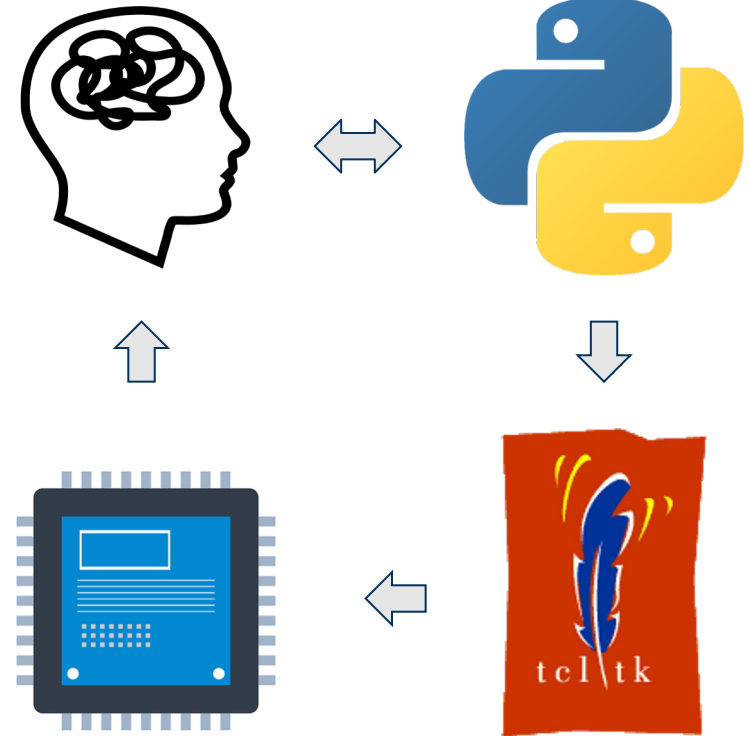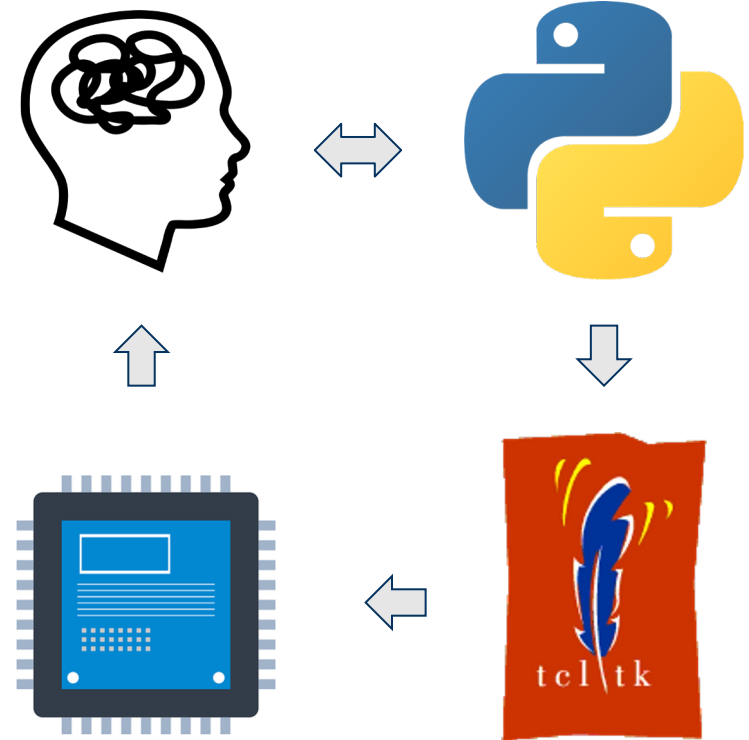
… a Python framework for abstracting and building standardized flows

… 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
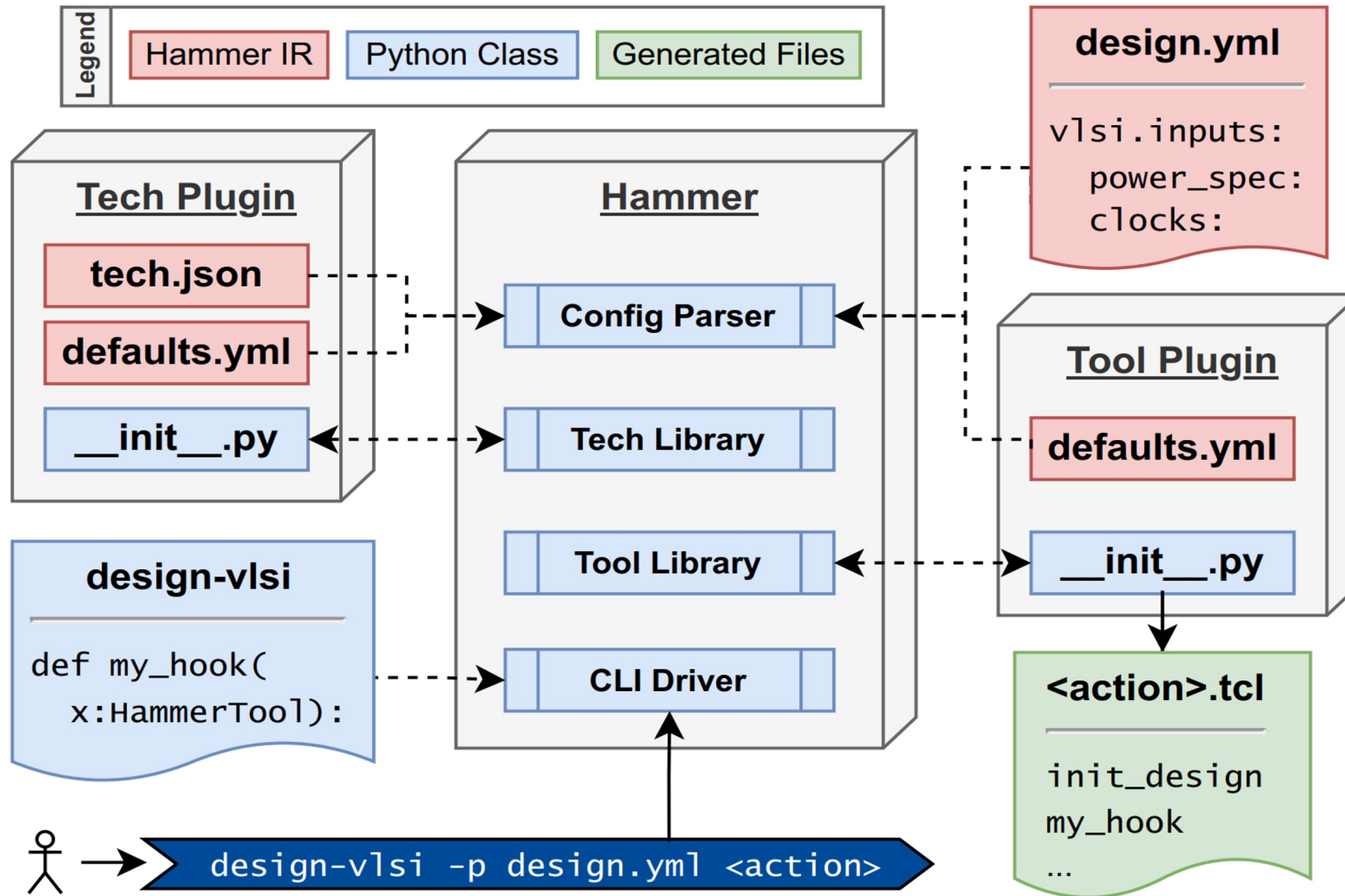
Hammer Intermediate Representation (IR)

Legend: Hammer IR | Python Class | Generated Files
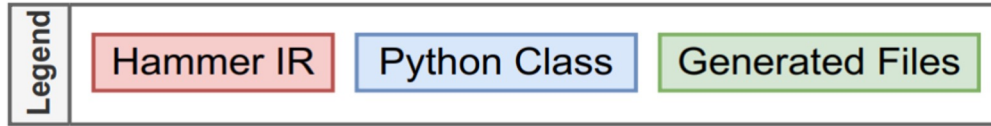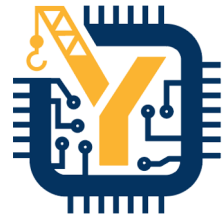
design.yml

```
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**

Berkeley **A**rchitecture **R**esearch

Tool and Tech Plugins

Legend: Hammer IR | Python Class | Generated Files

Tech Plugin
- tech.json
- defaults.yml
- __init__.py

Hammer
- Config Parser
- Tech Library
- Tool Library

Tool Plugin
- defaults.yml
- __init__.py

Source file pointers

Default Hammer IR

- PDK modification
- Memory compilation
- Tech-specific flow actions

Default Hammer IR

Output script

```
<action>.tcl

init_design
my_hook
…
```

- Default commands
- Implement Hammer APIs
- Execute tool
- Parse errors, metrics

Berkeley Architecture Research

Hooks and Drivers

Legend: Hammer IR | Python Class | Generated Files

Hooks = **customization**

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

design-vlsi
```
def my_hook(
    x:HammerTool):
```

Hammer
- Config Parser
- Tech Library
- Tool Library
- CLI Driver

```
design-vlsi -p design.yml <action>
```
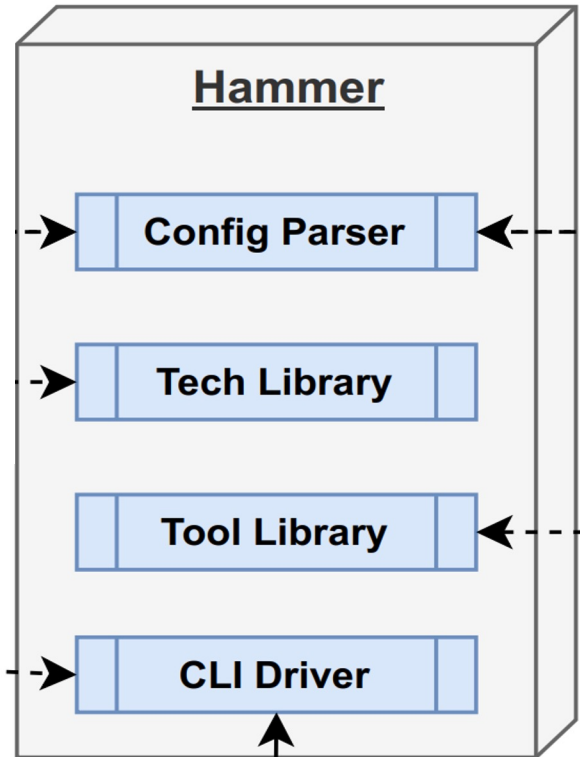
Hammer Driver

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

Berkeley Architecture Research

Hammer IR + Plugins
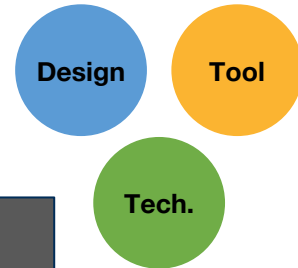
- Hammer **IR** (intermediate representation) codifies design information in JSON/YAML

- "Namespaces" = categories of attributes (e.g. `vlsi.core`)

- Metaprogramming

  - Modify attributes with additional Hammer IR snippets

  - Great for overriding tech- and tool-default settings

- Tool and technology plugins translate IR to Tcl scripts

  - Implement Hammer APIs

  - Include default settings, flow steps, and helper methods

  - Interchangeable = reusable!

Separated Concerns

Design   Tool

Tech.

```
# Specify clock signals
vlsi.inputs.clocks: [
  {name: "clock", period: "1ns", uncertainty: "0.1ns"}
]
# Generate Make include to aid in flow
vlsi.core.build_system: make
# Pin placement constraints
vlsi.inputs.pin_mode: generated
vlsi.inputs.pin.generate_mode: semi_auto
vlsi.inputs.pin.assignments: [
  {pins: "*", layers: ["M5", "M7"], side: "bottom"}
]
```

Berkeley **A**rchitecture **R**esearch

Goals

- Hammer at 10,000 feet

- Fixing traditional VLSI flows: One size doesn't fit all

- Overview of Hammer's abstractions

- **Hammer example**

Berkeley **A**rchitecture **R**esearch

Power Straps Example

- To specify power straps, need to know:
  - DRC rules
  - Target power dissipation
  - IR drop spec
  - Domain areas

Separated Concerns


```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
    -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
    -area [get_bbox -of ModuleABC]  \
    -start [expr [lindex [lindex [get_bbox -of ModuleABC] 0] 0] + 1.234]
```

- Hierarchical also adds physical constraints:
  - Tiled modules require pitch-matching
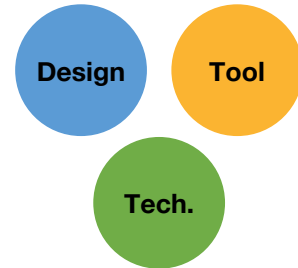 - Easy to make mistakes when reworking

Berkeley Architecture Research
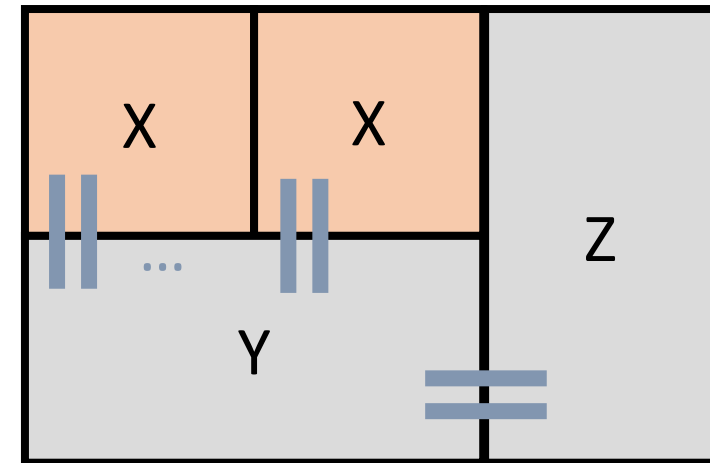
Power Straps Example

- **Don't make the designer do math!**
  - Codify design process in tech- and tool-agnostic code

- **Method:**
  - Determine valid pitches for hierarchical design
  - Automatically calculate offsets for hierarchical blocks
  - Generate layout-optimal, DRC clean straps
  - Specify intent at a higher-level than length units

- **Example: Using "By tracks" specification**
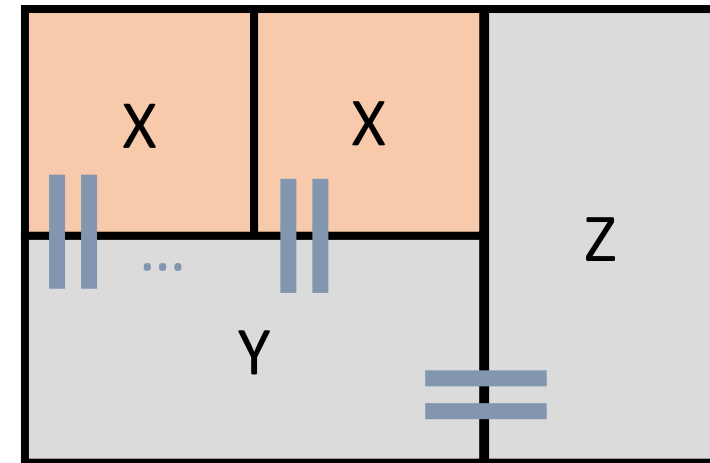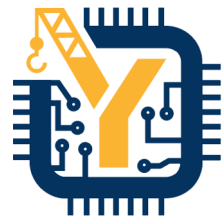
Separated Concerns

Design　Tool

Tech.

Goals

- Hammer at 10,000 feet

- Fixing traditional VLSI flows: One size doesn't fit all

- Overview of Hammer's abstractions

- Hammer community development

Berkeley **A**rchitecture **R**esearch

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
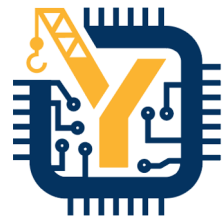- Chipyard examples with ASAP7, Sky130

# Everyone can contribute to Hammer

Modular design + override features = community contributions
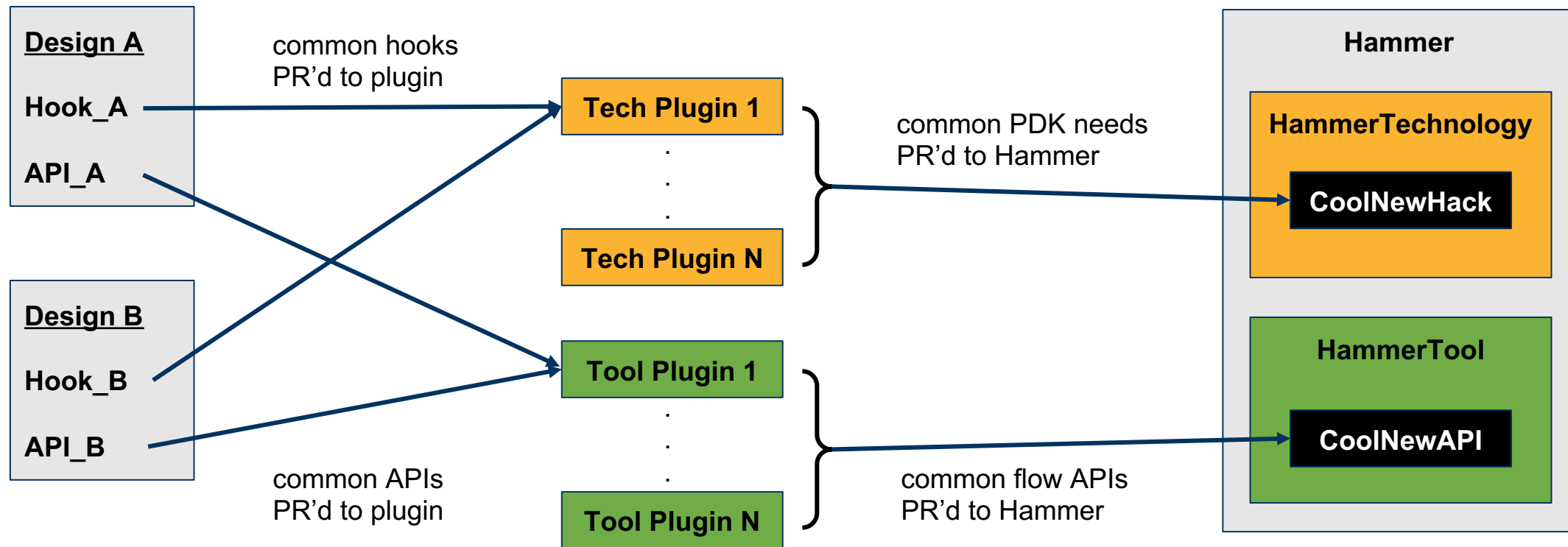
Summary

- 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

Design

Tool

Tech.

Berkeley Architecture Research
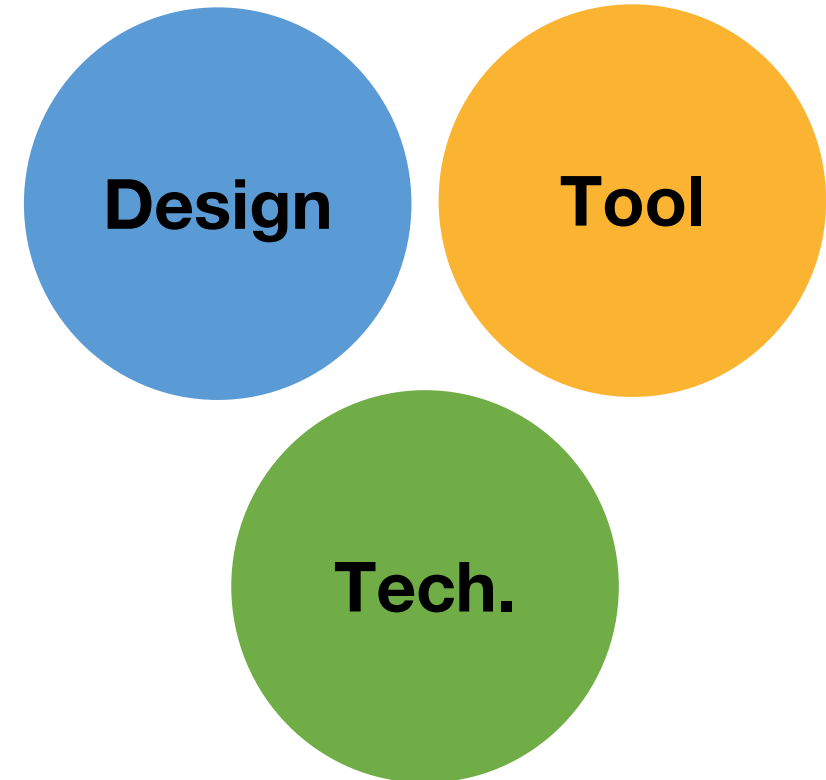
Future of Hammer

- More tool integration: metrics parsing, constraints feedback
- Aspect-Oriented Chisel Floorplanning
  - Higher-level abstractions
    - Hierarchical partitioning, power strap/pin alignment
    - BAG IP collateral generation/integration
  - Reconfiguring Chisel designs based on physical design feedback
    - e.g. change clock constraints based on timing metrics
- Abutment/partition-based hierarchical flow
- True multi-clock & power domain (for DVFS, etc.)
- Dev work is cyclical -> typically happens alongside tapeouts
  - Lots of ideas, help always wanted

Berkeley Architecture Research

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

- User mailing list: hammer-users@googlegroups.com

- Plugin access requests: hammer-plugins-access@lists.berkeley.edu
    - Cadence, Synopsys, and Mentor

- UCB Digital Design labs: https://github.com/EECS150/asic_labs_sp22
    - full lab releases coming soon!