Hammer VLSI Flow

Nayiri Krzysztofowicz

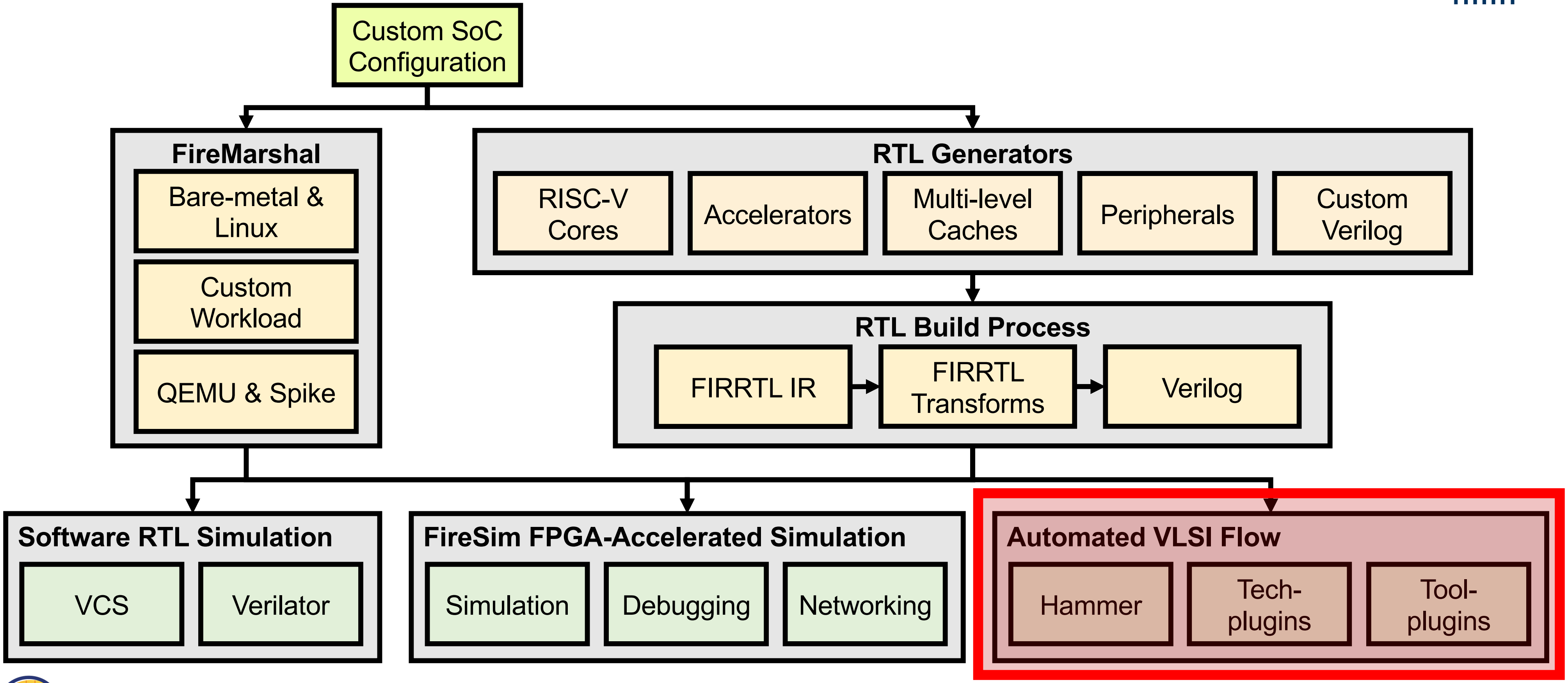
UC Berkeley

nayiri@berkeley.edu

Berkeley
Architecture
Research

CHIPYARD

Tutorial Roadmap



Custom SoC Configuration

FireMarshal
- Bare-metal & Linux
- Custom Workload
- QEMU & Spike

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

RTL Build Process
- FIRRTL IR
- FIRRTL Transforms
- Verilog

Software RTL Simulation
- VCS
- Verilator

FireSim FPGA-Accelerated Simulation
- Simulation
- Debugging
- Networking

Automated VLSI Flow
- Hammer
- Tech-plugins
- Tool-plugins

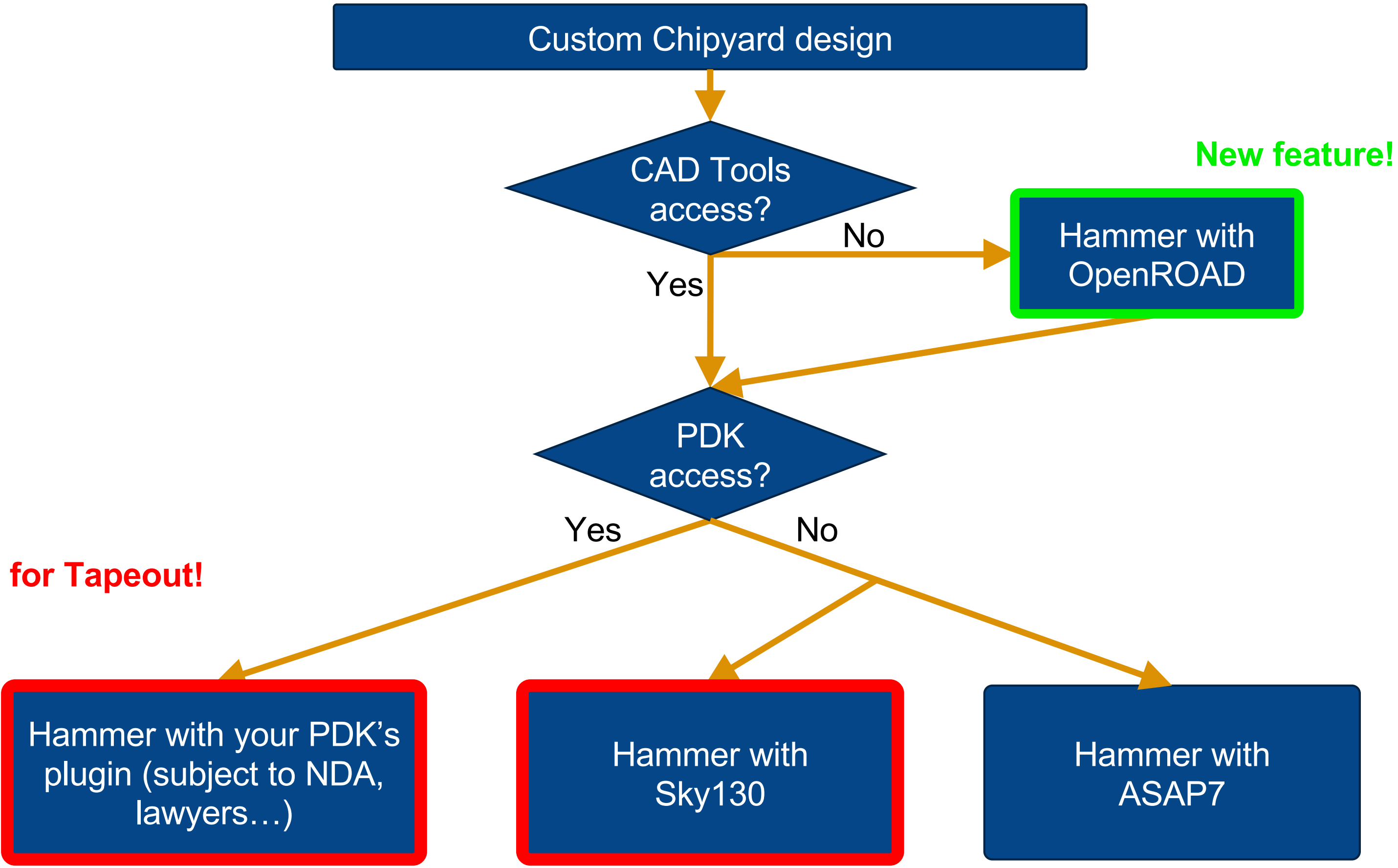
Goals

- Hammer applications
- Fixing traditional VLSI flows: One size doesn’t fit all
- Overview of Hammer’s abstractions
- Get you started with a TinyRocketConfig in Sky130 + OpenROAD
- Under the hood: plugins, hooks, etc.

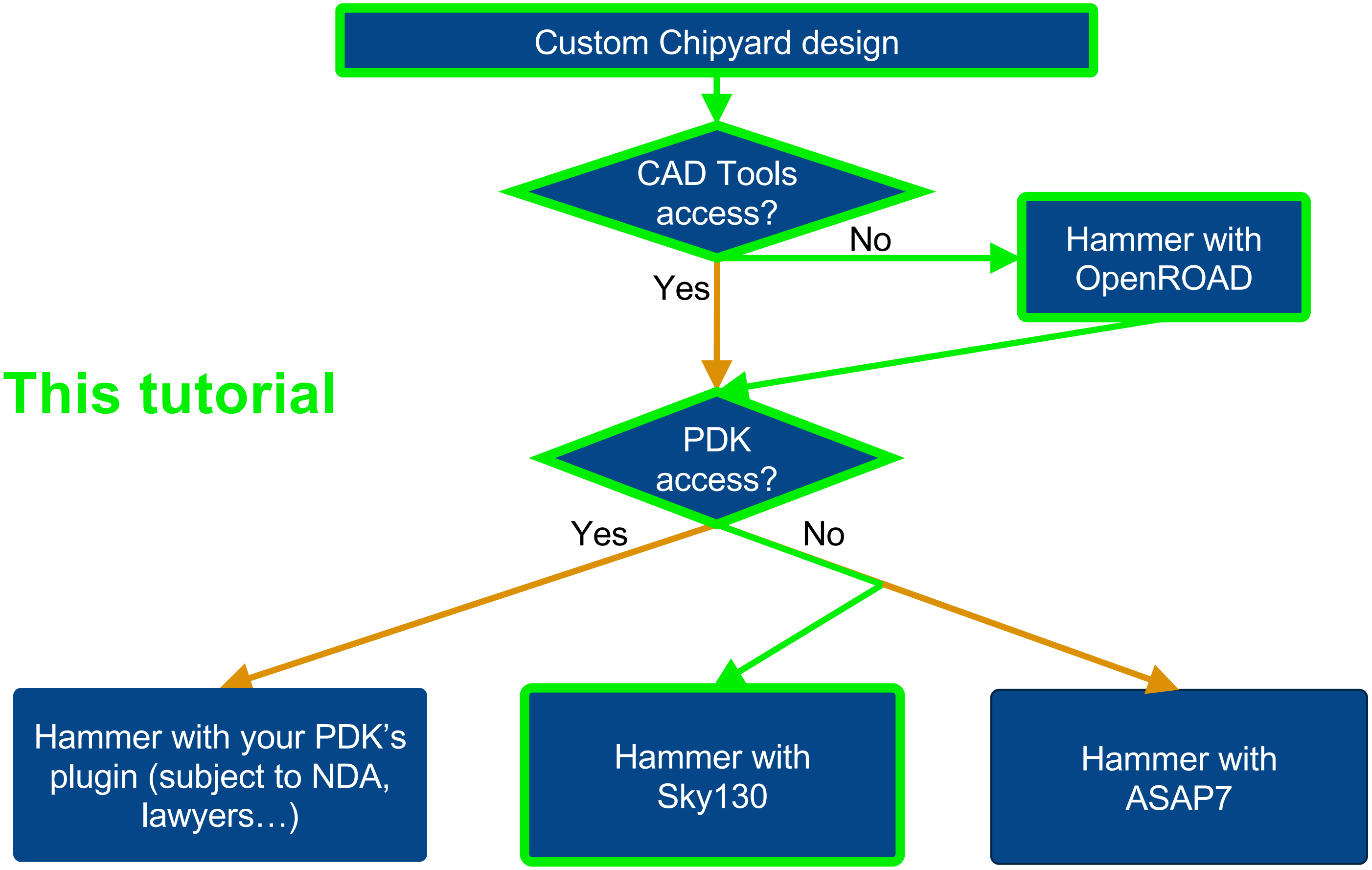
Goals

- Hammer applications
- Fixing traditional VLSI flows: One size doesn’t fit all
- Overview of Hammer’s abstractions
- Get you started with a TinyRocketConfig in Sky130 + OpenROAD
- Under the hood: plugins, hooks, etc.

Hammer Decision Tree

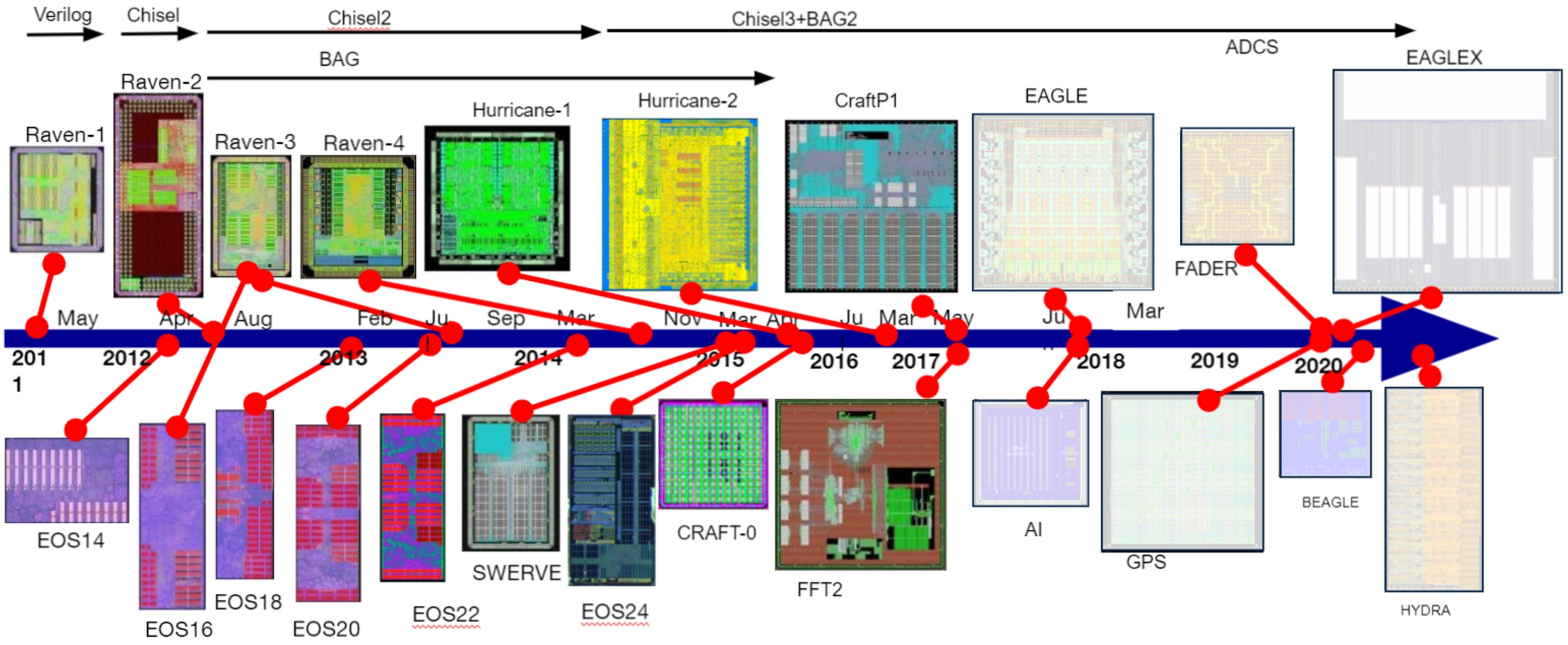


Hammer Decision Tree



This tutorial

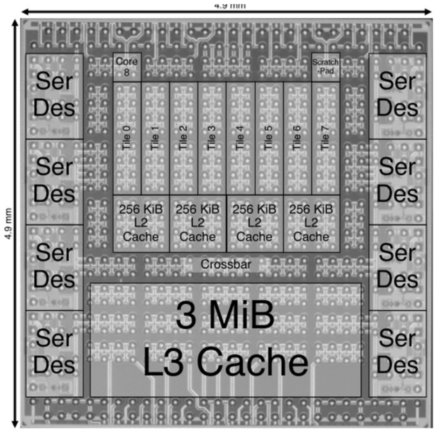
Hammer for Real Tapeouts



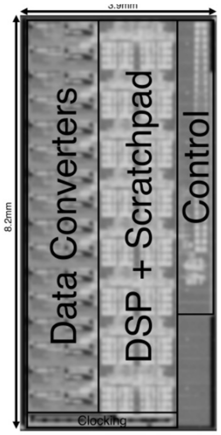
Raven, Hurricane: ST 28nm FDSOI, SWERVE: TSMC 28nm EOS: IBM 45nm SOI, CRAFT: 16nm TSMC,

Many Different Chips!

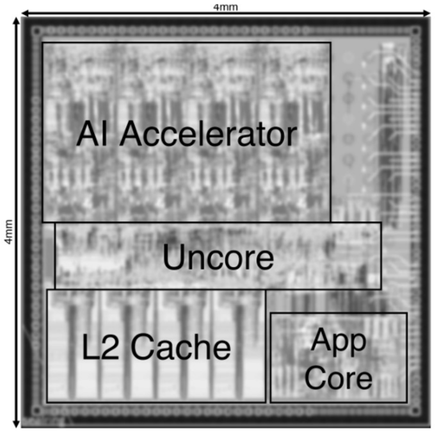
	Eagle [1]	HugeFlyingSoC	NavRx	WaterSerpent	MythicChip	OsciBear [2]	HDBinaryCore
Description	9-core RISC-V SoC	22-core RISC-V SoC	GPS receiver SoC	MU-MIMO baseband SoC	RISC-V SoC for ML	Bluetooth SoC	Hyperdim. computing proc.
Foundry Node	A 16nm	A 16nm	A 16nm	B 22nm	C 12nm	A 28nm, Sky130	A 28nm
Signoff 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

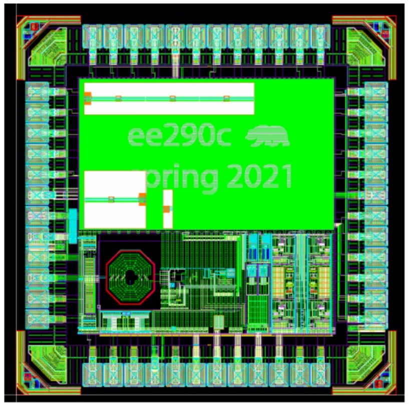


MythicChip

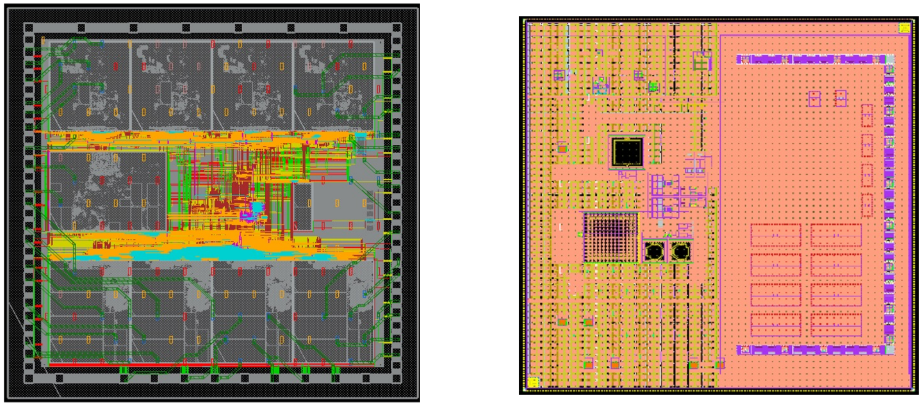
[1] C. Schmidt, et. al, *ISSCC 2021*
[2] D. Fritchman et. al, *IEEE SSCS Magazine, Spring 2022*

Hammer in Courses

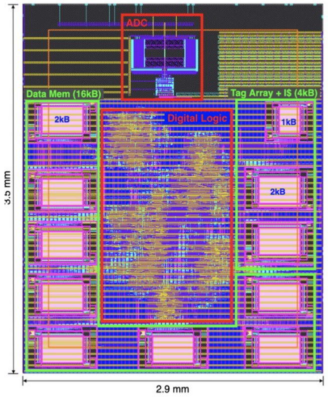
- Introduced in undergraduate digital circuits and systems labs:
  - http://github.com/EECS150 (ASAP7 and Sky130 plugins)
- Special topics ‘tapeout’ class
  - Spring 2022: 1 sophomore, 15 juniors, 19 seniors, 3 5th-yr MS, 2 MEng, 1 PhD



2021 EE194/290C: OsciBear
TSMC 28nm



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



Sky130 MPW-2
Skywater 130nm

Berkeley Architecture Research

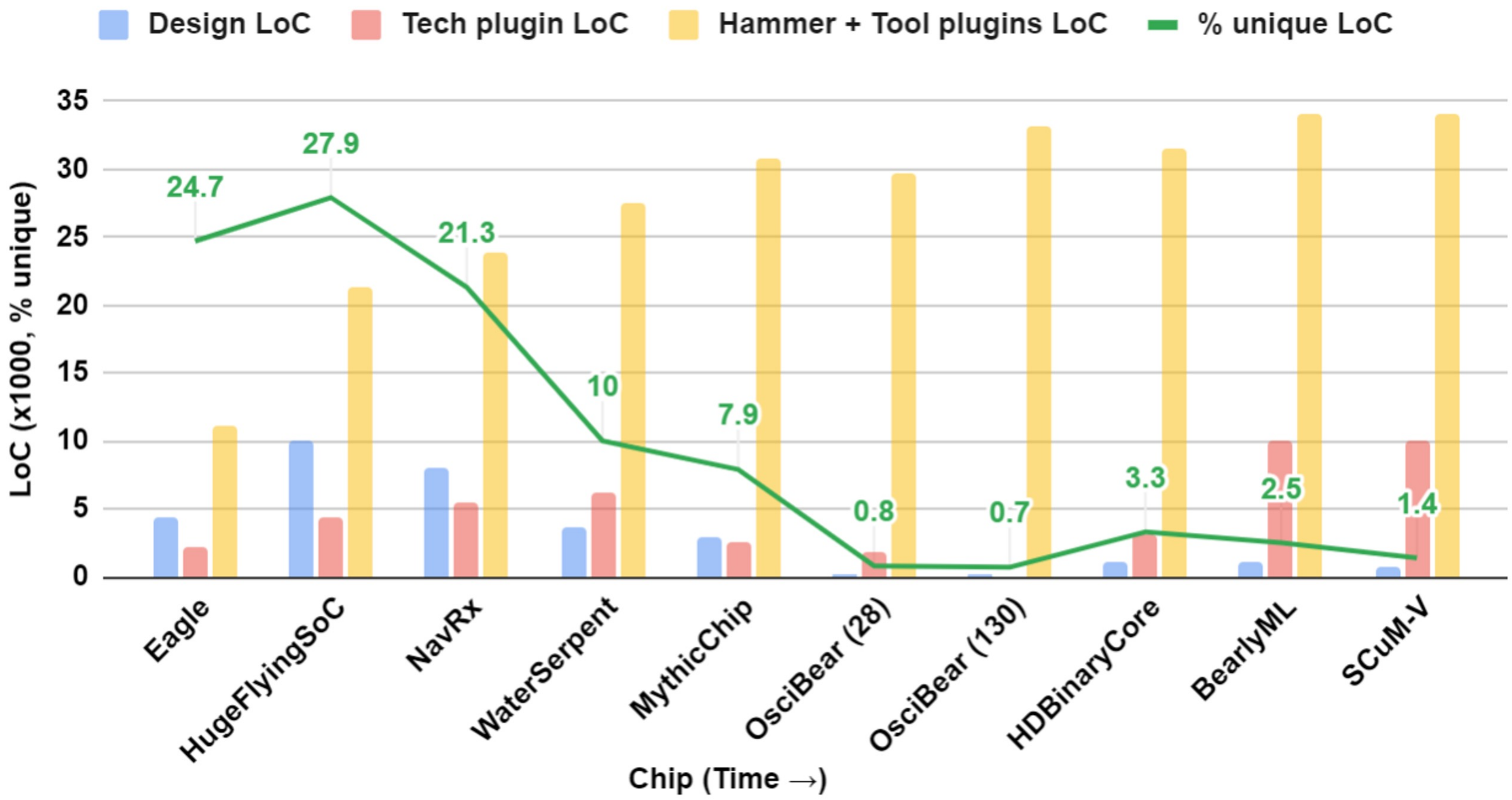
Plugins Supported

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

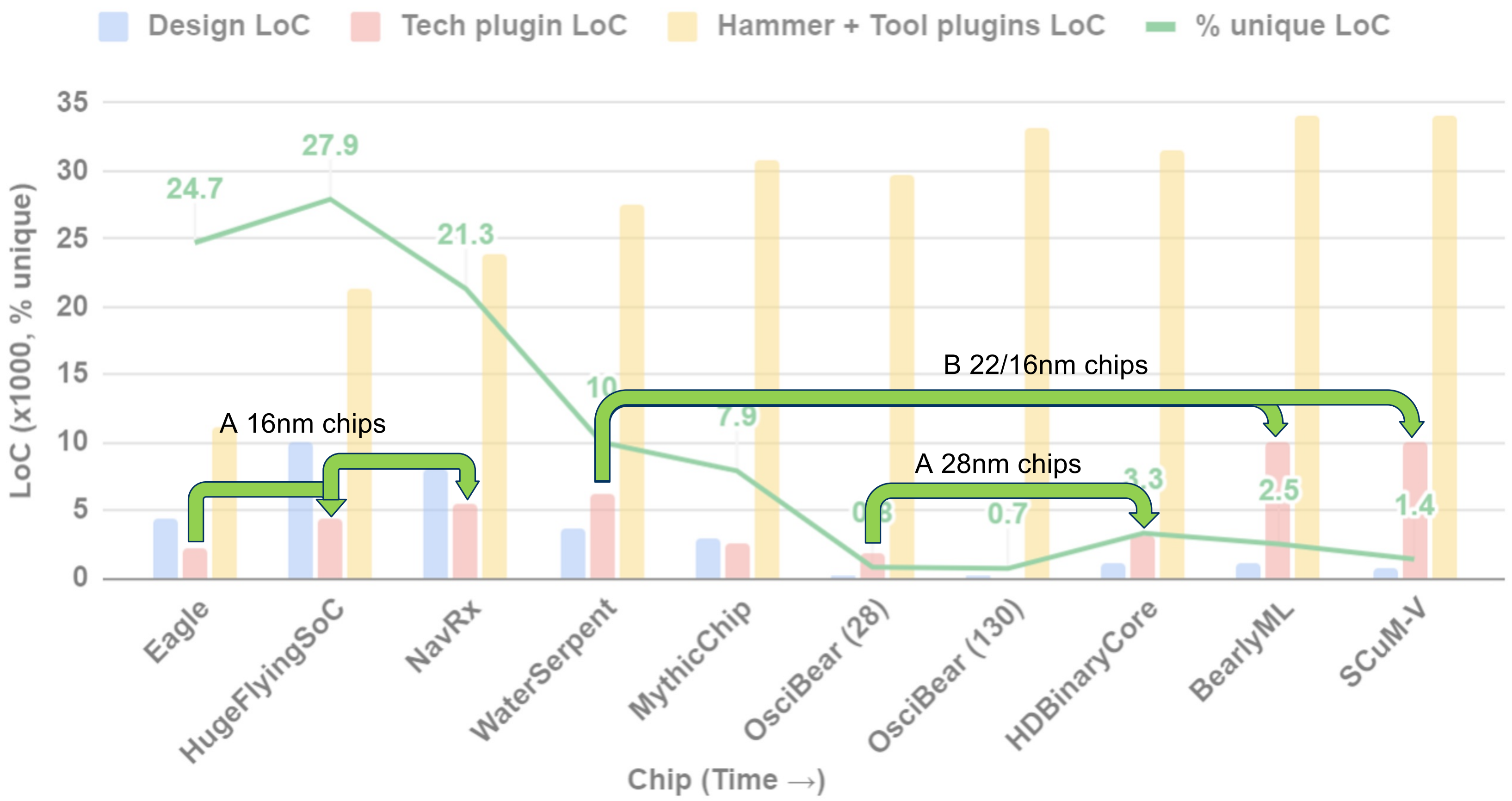
^{C}Cadence ^{S}Synopsys ^{M}Siemens Mentor ^{X}Xilinx

Increasing Flow Reusability



Berkeley Architecture Research

Tech Plugin Reusability



A 16nm chips

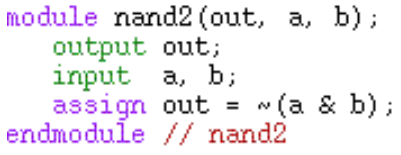
B 22/16nm chips

A 28nm chips

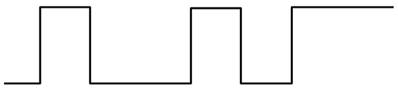
Goals

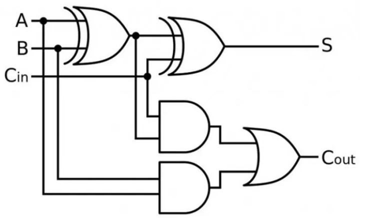
- Hammer at 10,000 feet
- Fixing traditional VLSI flows: One size doesn’t fit all
- Overview of Hammer’s abstractions
- Get you started with a TinyRocketConfig in Sky130 + OpenROAD
- Under the hood: plugins, hooks, etc.

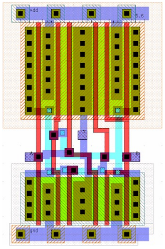
ASIC Flow

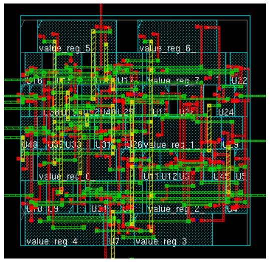


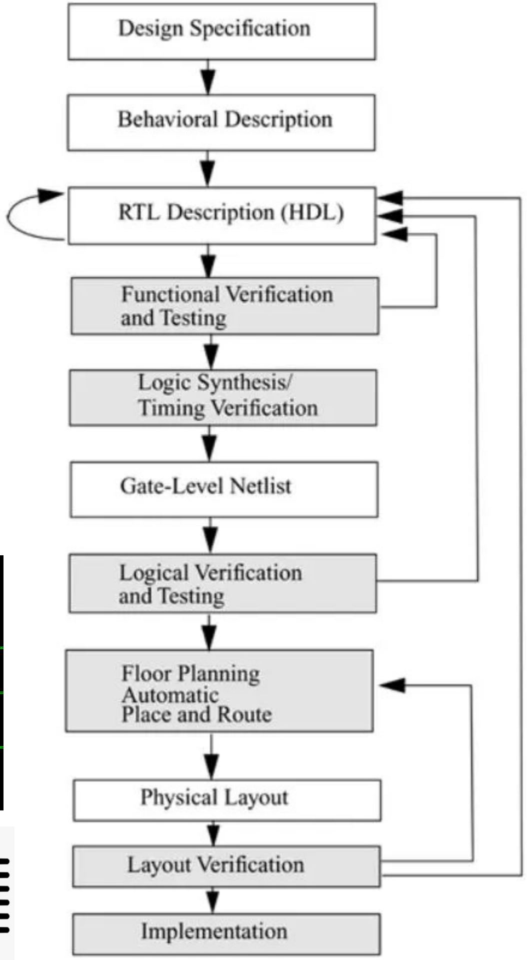
```
module nand2(out, a, b);
   output out;
   input  a, b;
   assign out = ~(a & b);
endmodule // nand2
```










Design Specification
Behavioral Description
RTL Description (HDL)
Functional Verification and Testing
Logic Synthesis/ Timing Verification
Gate-Level Netlist
Logical Verification and Testing
Floor Planning Automatic Place and Route
Physical Layout
Layout Verification
Implementation

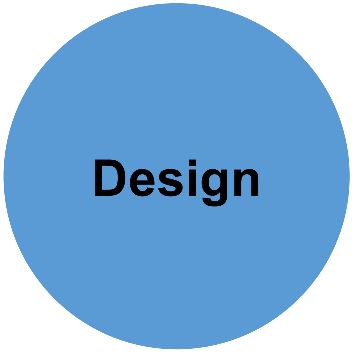
synthesis

place-and-route

DRC

LVS

Components of VLSI Flows


Design

- RTL
- IP cores
- Floorplan
- Constraints
- …

Components of VLSI Flows

Design

- RTL
- IP cores
- Floorplan
- Constraints
- …

Tech

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

Components of VLSI Flows

Design

- RTL
- IP cores
- Floorplan
- Constraints
- …

Tech

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

Tools

- Synopsys VCS
- Cadence Genus
- OpenROAD
- …

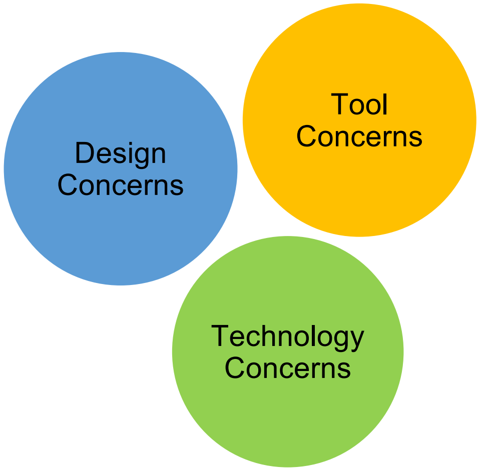
Why is Physical Design so Hard?

VLSI flows *are custom built for each project*

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…



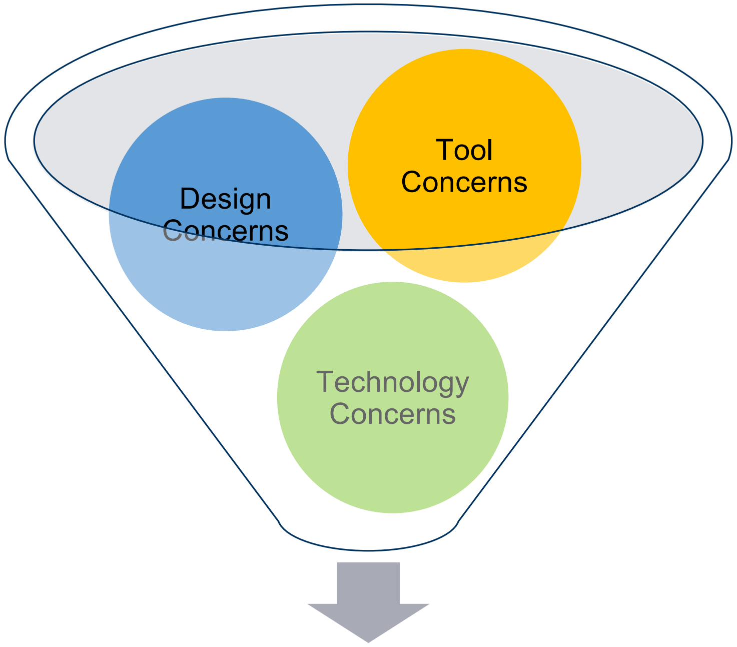
Berkeley Architecture Research

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

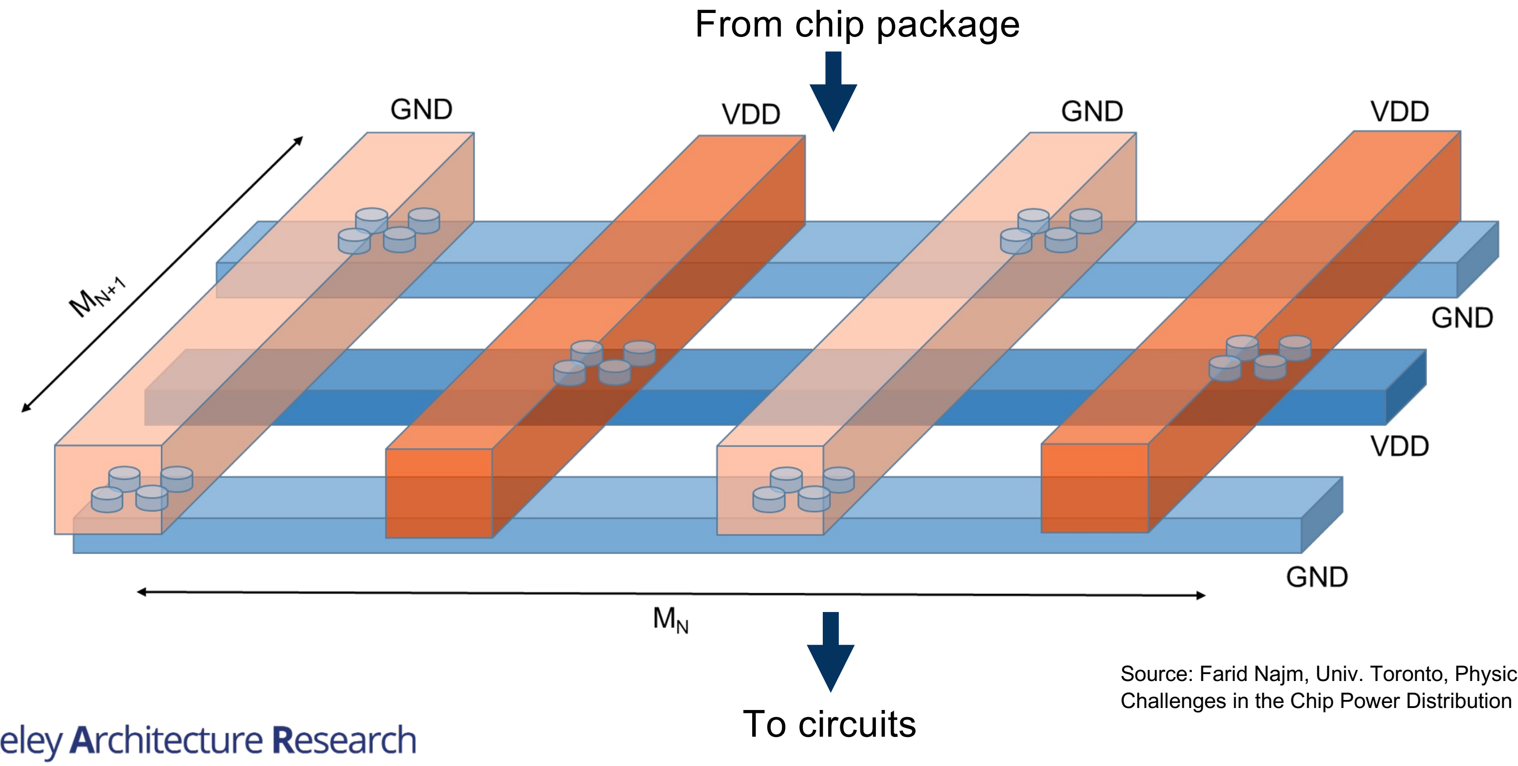
    and domain specialization demand architectural exploration



Non-reusable Tcl scripts

Example: Power Straps

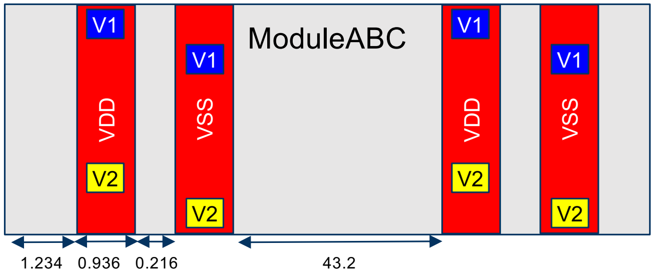
- How to distribute power from package to transistors



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

Example

- Hypothetical power strap creation command:

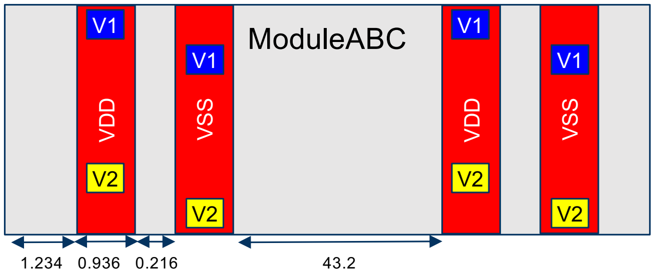


```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
  -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
  -area [get_bbox -of ModuleABC]  \
  -start [expr [lindex [lindex [get_bbox -of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

# Repeat for each layer!
```

Berkeley Architecture Research

Example

- Hypothetical power strap creation command:



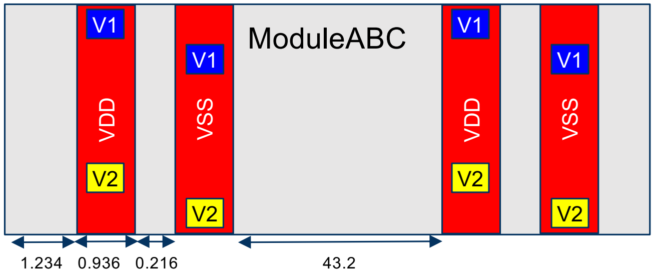
```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
  -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
  -area [get_bbox -of ModuleABC]  \
  -start [expr [lindex [lindex [get_bbox -of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

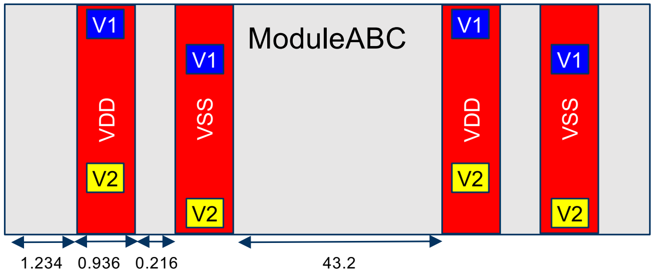
🟧 The command + options

Tech-specific

🟩 DRC/ERC-compliant dimensions

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

Design-specific

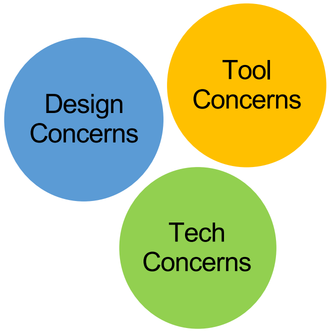
Layers, power domains, floorplan, density

Goals

- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Get you started with a TinyRocketConfig in Sky130 + OpenROAD
- Under the hood: plugins, hooks, etc.

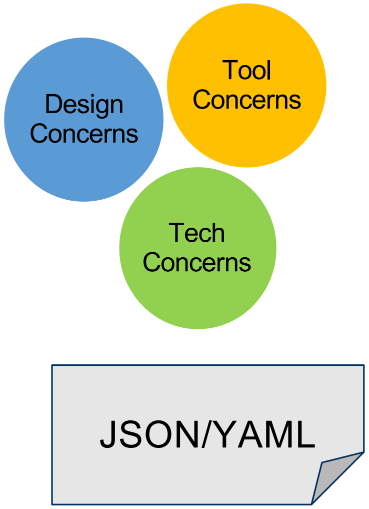
Hammer Design Principles

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



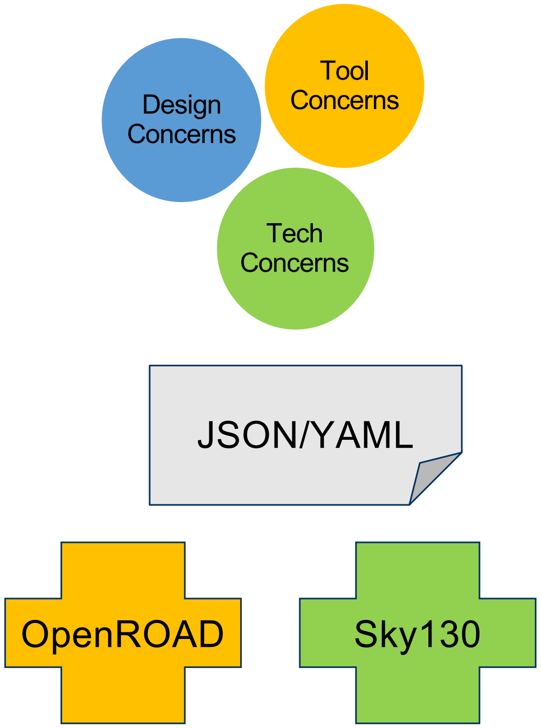
Hammer Design Principles

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



Design Concerns

Tool Concerns

Tech Concerns

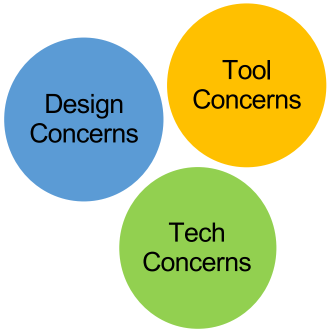
JSON/YAML

OpenROAD

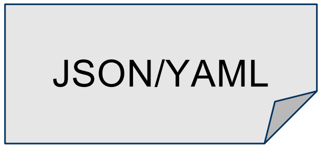
Sky130

Hammer Design Principles

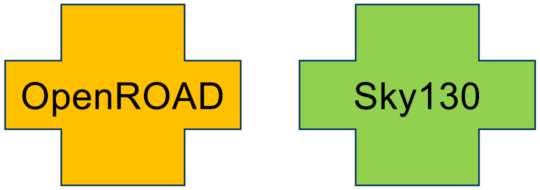
1. **Separation of Concerns**
   - Decouple design-, tool-, and tech-specific concerns



2. **Standardization**
   - Data interchange schema for constraints, options, files



3. **Modularity**
   - Interchangeable & shareable tool & tech plugins

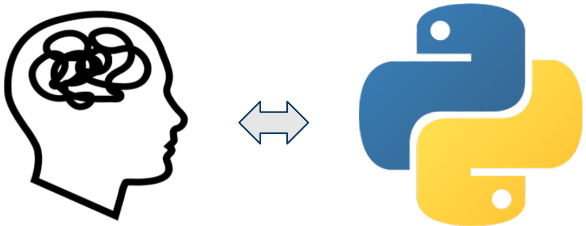


4. **Incremental Adoption**
   - Mix reusable & custom solutions

What is Hammer?

Hammer is:

… a Python framework for abstracting and building standardized flows

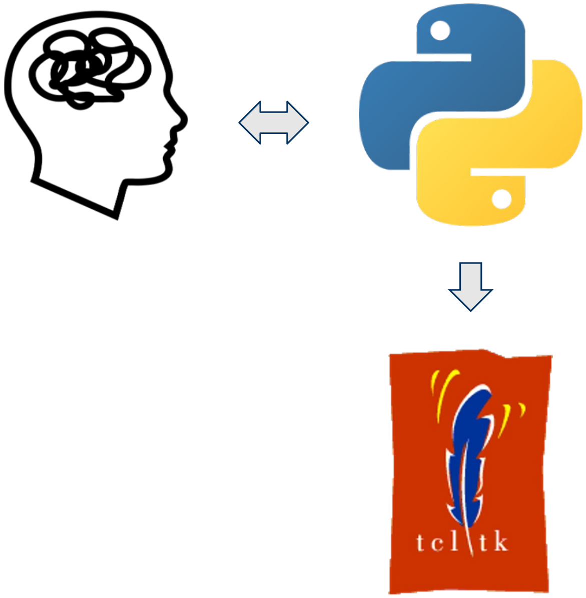


What is Hammer?

Hammer is:

… a Python framework for abstracting and building standardized flows

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



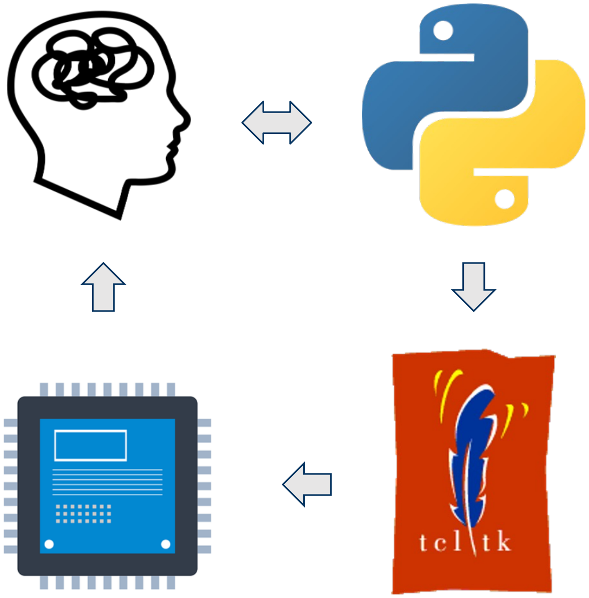
What is Hammer?

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?

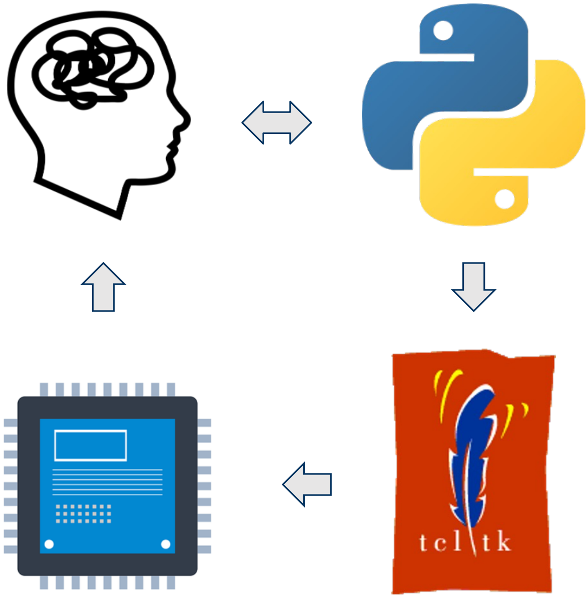
Hammer is:

… 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!



Berkeley Architecture Research

What is Hammer?

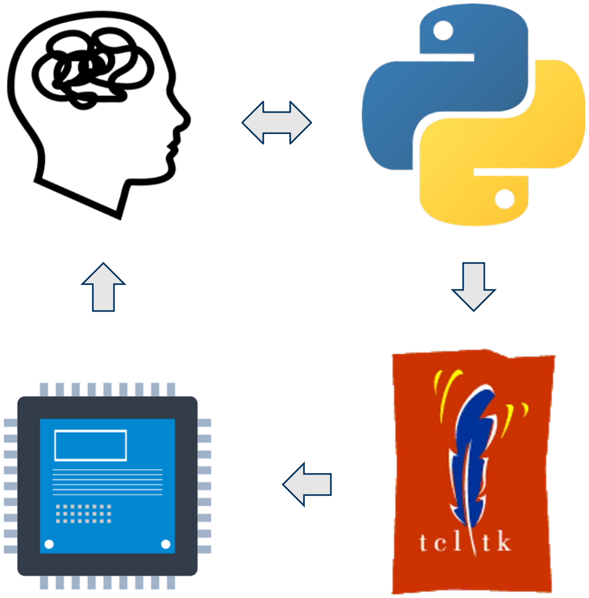
Hammer is:

… a Python framework for abstracting and building standardized flows

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

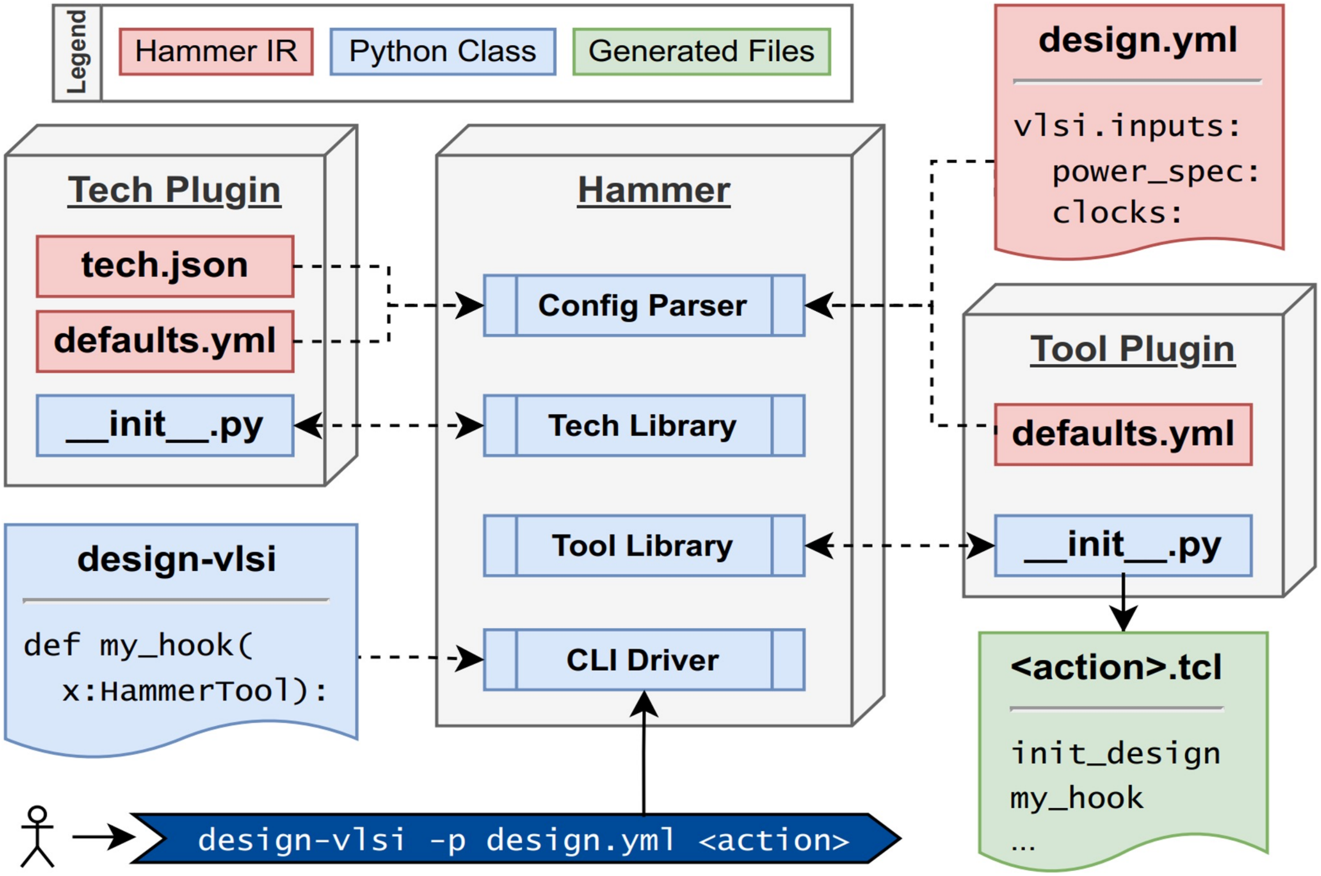
… proven for architecture exploration, teaching, and research chips

… open-source!



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

Hammer Software Architecture



Berkeley Architecture Research

Hammer Intermediate Representation (IR)

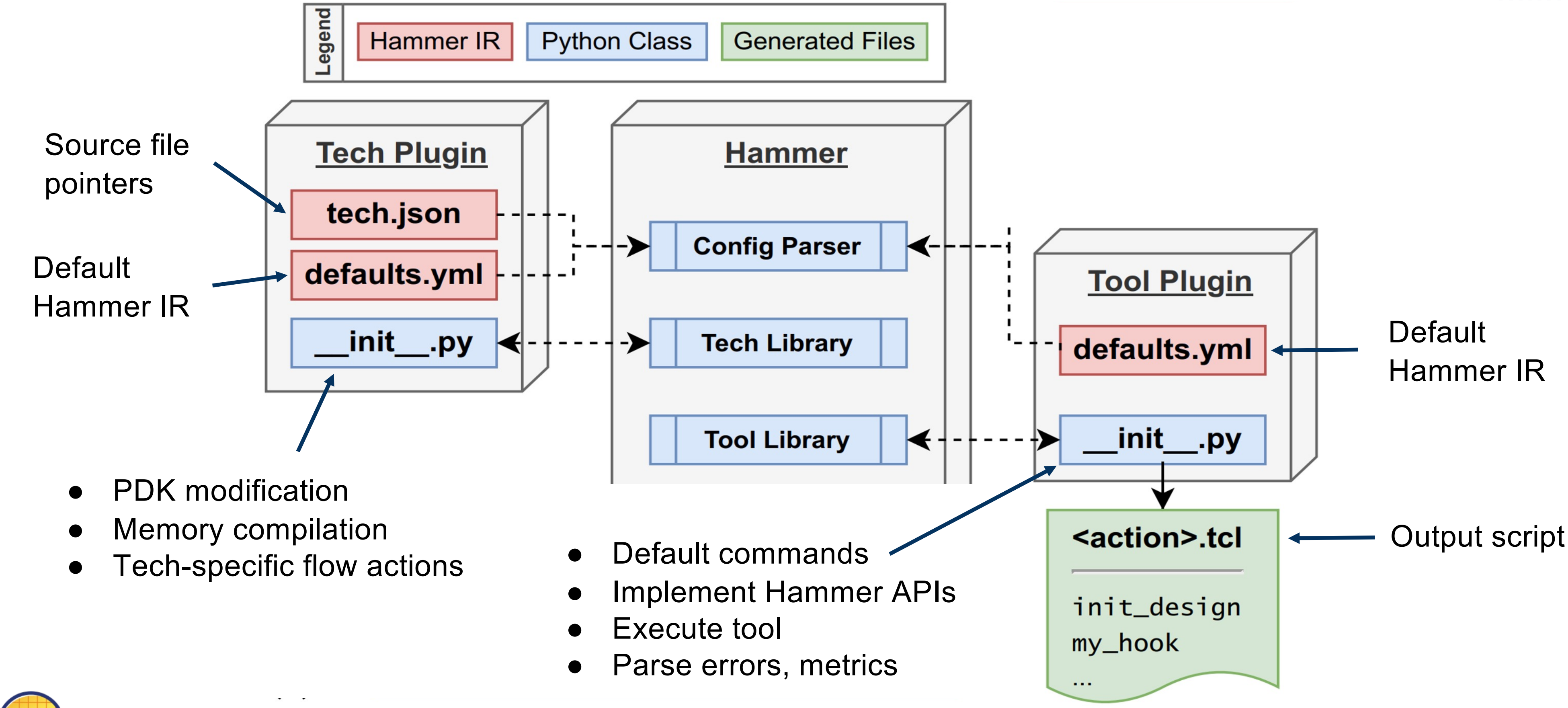
Legend: 🟥 Hammer IR | 🟦 Python Class | 🟩 Generated Files

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**

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

<action>.tcl

```
init_design
my_hook
...
```

Source file pointers

Default Hammer IR

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

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

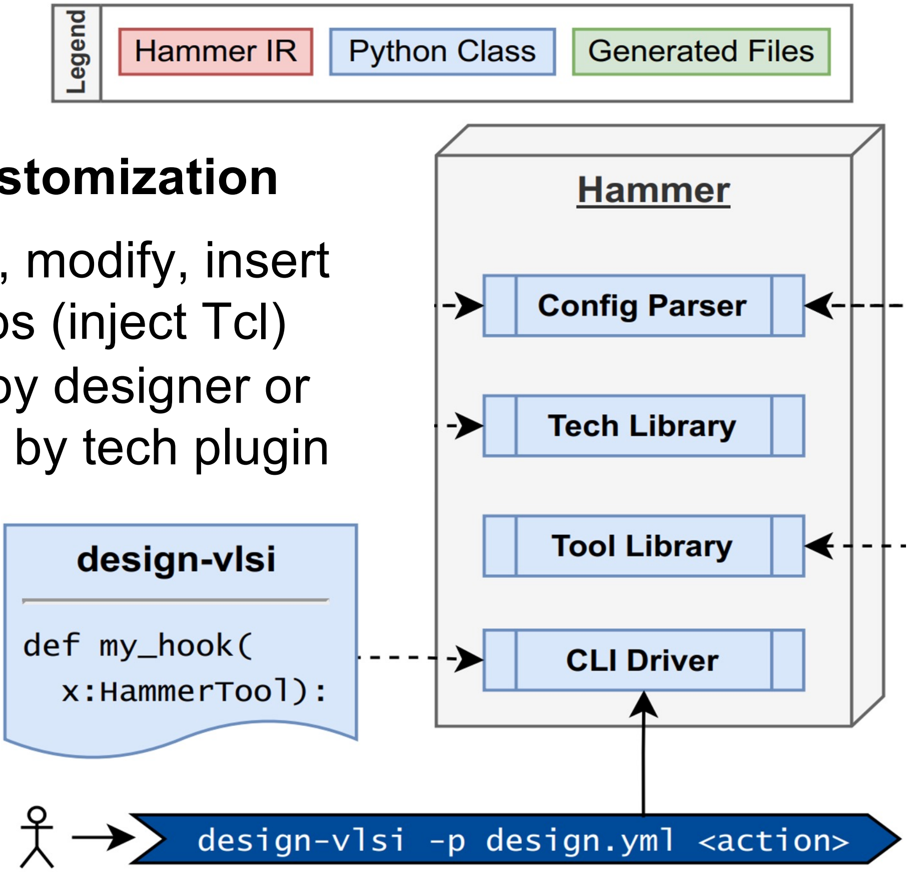
Default Hammer IR

Output script

Hooks and Drivers

Hooks = **customization**

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



```
def my_hook(
  x:HammerTool):
```

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

Hammer Driver

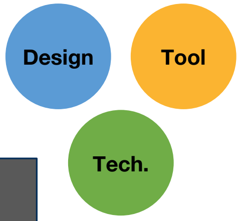
- Parses all IR, hooks
- Auto-generates hierarchical flow graph as Makefile
- Easy-to-use CLI

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!

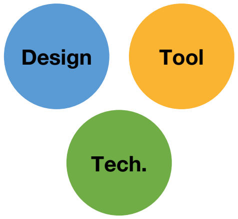
```
# Specify clock signals
vlsi.inputs.clocks: [
  {name: "clock", period: "1ns", uncertainty: "0.1ns"}
]
# Generate Make include to aid in flow
vlsi.core.build_system: make
# Pin placement constraints
vlsi.inputs.pin_mode: generated
vlsi.inputs.pin.generate_mode: semi_auto
vlsi.inputs.pin.assignments: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

Separated Concerns



Power Straps Example

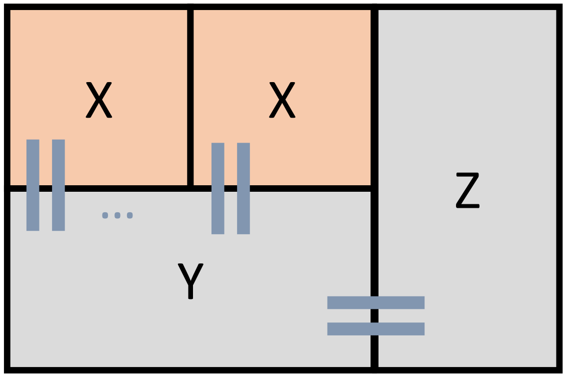
Separated Concerns



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

```
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

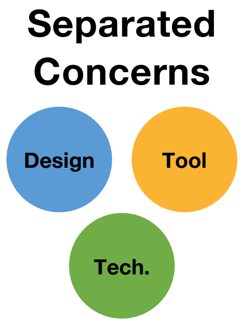


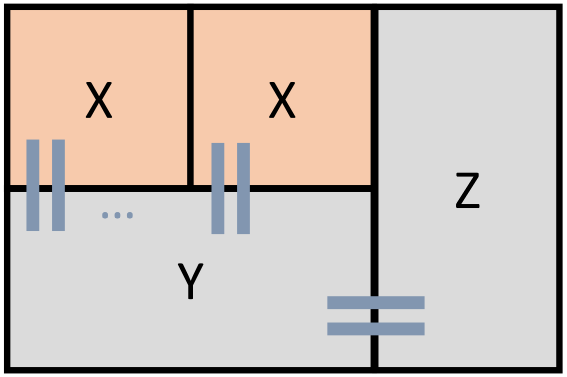
Power Straps Example

- 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





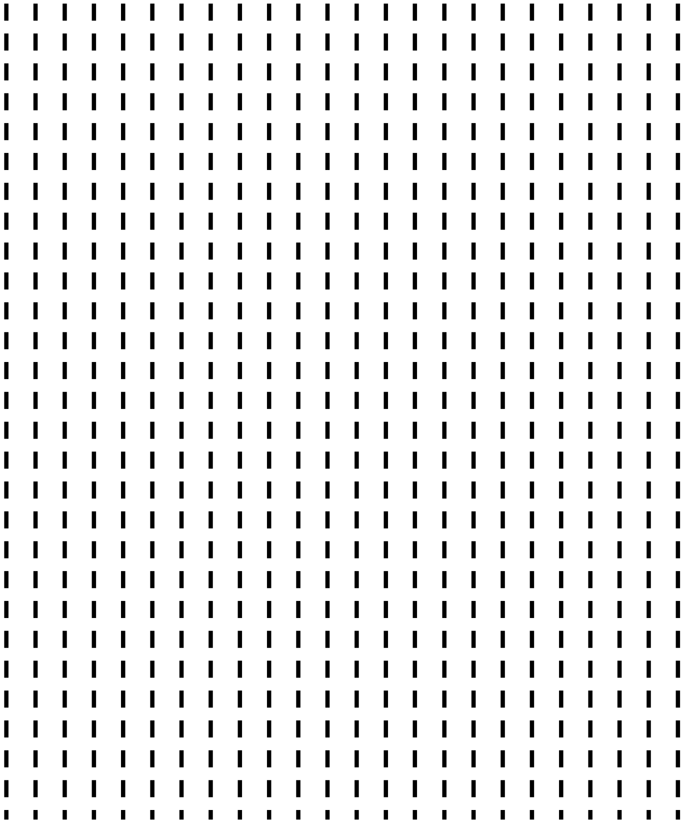
Power Straps Example

Separated Concerns

Design

Tool

Tech.



Choose power strap strategy

```
par.generate_power_straps_method: by_tracks
par.power_straps_mode: generate
```

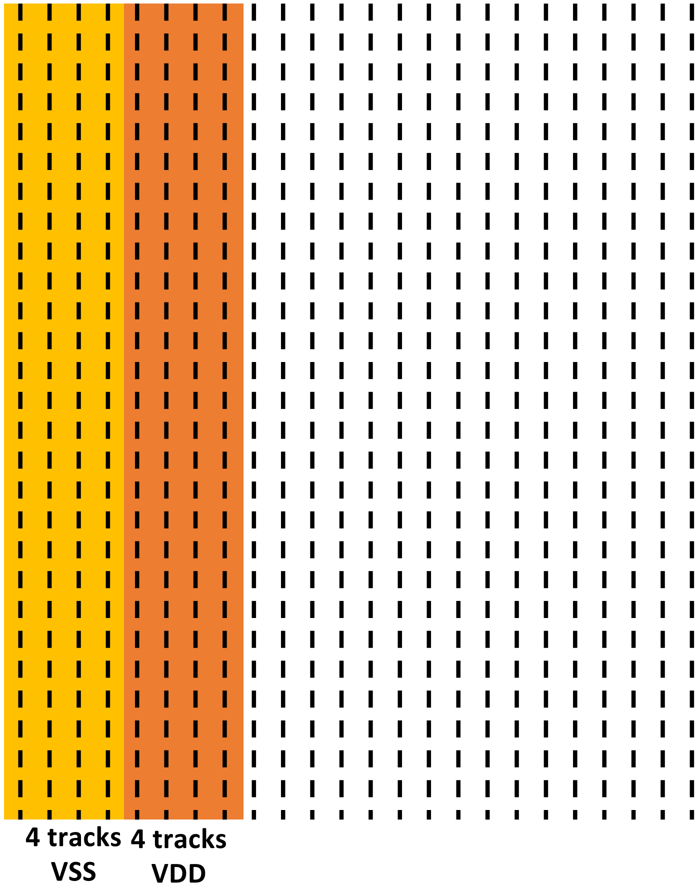
Power Straps Example

Separated Concerns

Design

Tool

Tech.



Allocate tracks

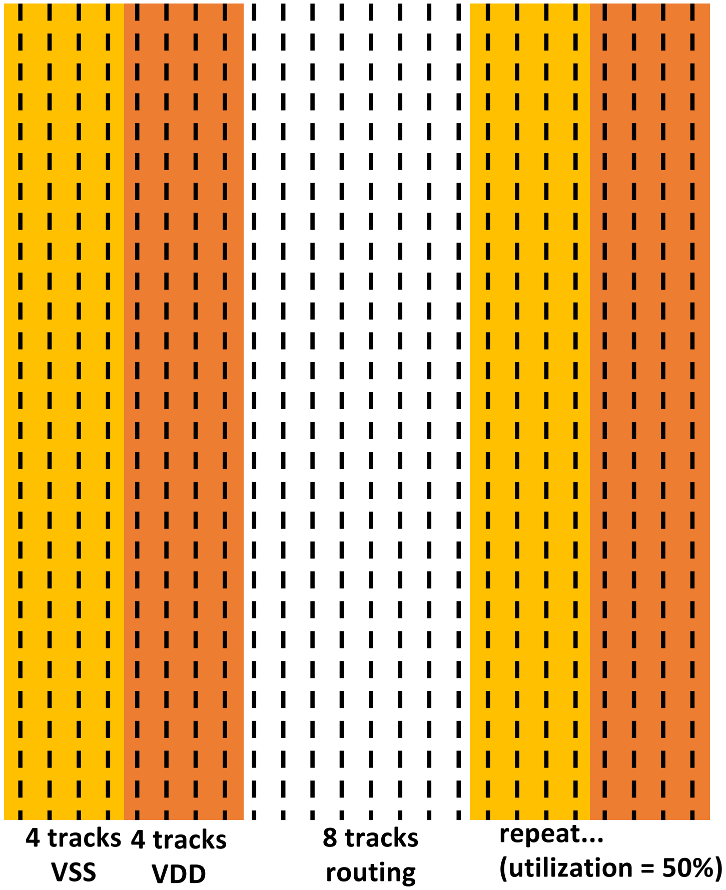
```
par.generate_power_straps_options:
    by_tracks:
        track_width: 4
```

```
par.generate_power_straps_method: by_tracks
par.power_straps_mode: generate
```

number of power domains = 2 (VDD, VSS)

tracks per group = 4 tracks x 2 domains = 8

Power Straps Example



```
par.generate_power_straps_options:
    by_tracks:
        track_width: 4
        power_utilization: 0.5
```

Determine pitch

Separated Concerns

Design

Tool

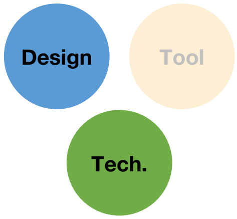
Tech.

```
par.generate_power_straps_method: by_tracks
par.power_straps_mode: generate
```

Group pitch = tracks per group / utilization
= 8 / 0.5 = 16

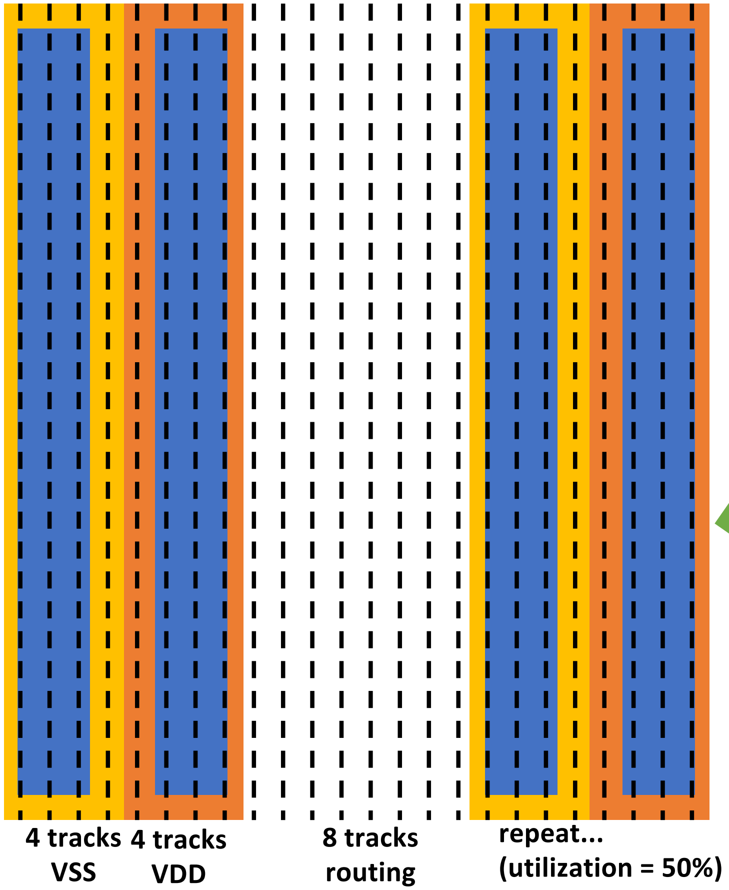
Power Straps Example

Separated Concerns

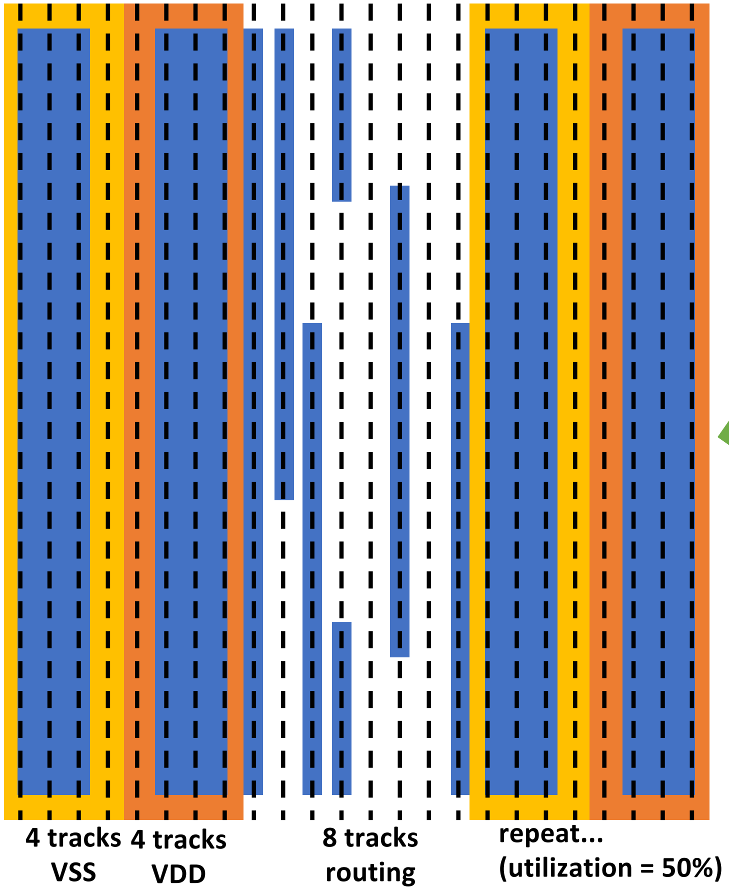


```
par.generate_power_straps_options:
    by_tracks:
        track_width: 4
        power_utilization: 0.5
        strap_layers:
            - M3
            - M4
            - M5
            - M6
            - M7
            - M8
            - M9
par.generate_power_straps_method: by_tracks
par.power_straps_mode: generate
```

Generate straps



Power Straps Example



Route design

```
par.generate_power_straps_options:
    by_tracks:
        track_width: 4
        power_utilization: 0.5
        strap_layers:
            - M3
            - M4
            - M5
            - M6
            - M7
            - M8
            - M9
par.generate_power_straps_method: by_tracks
par.power_straps_mode: generate
```

Separated Concerns

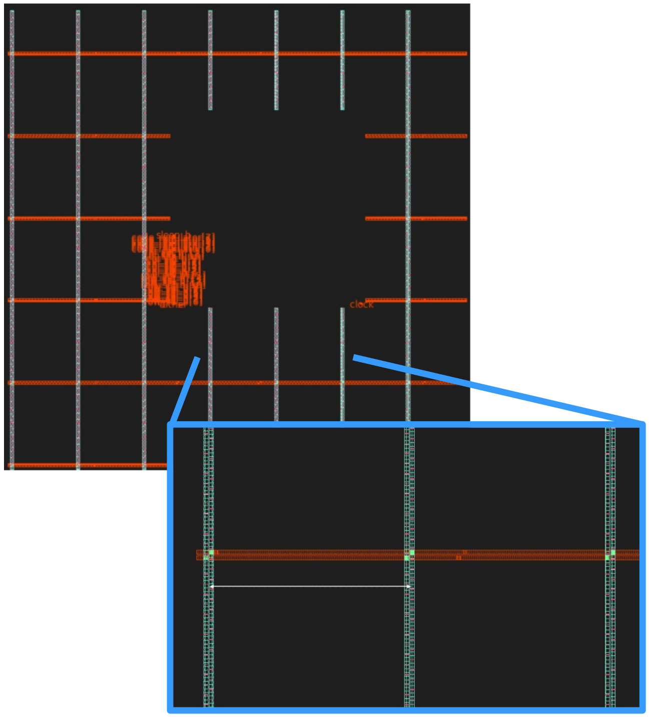
Design

Tool

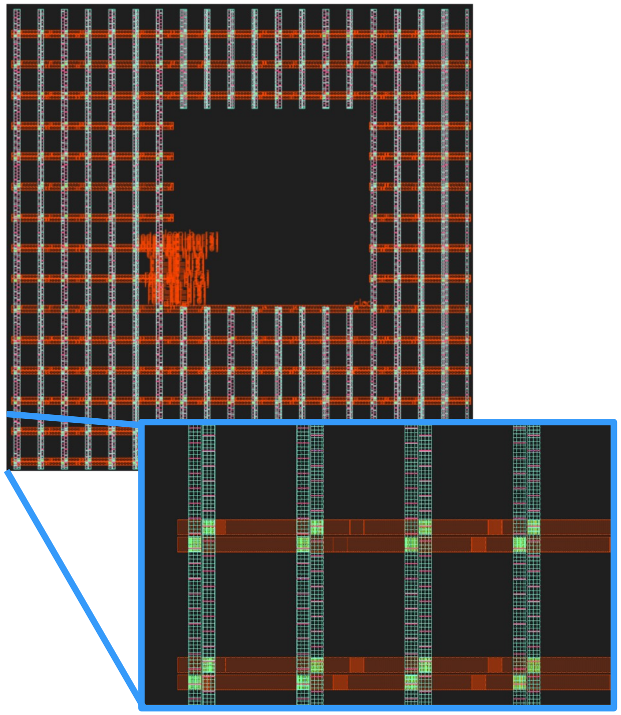
Tech.

Berkeley Architecture Research

Power Straps Example



`power_utilization: 0.1`



`power_utilization: 0.3`

Coming up…

Hammer Tutorial

Berkeley Architecture Research