

Hammer VLSI Flow

Jerry Zhao

UC Berkeley

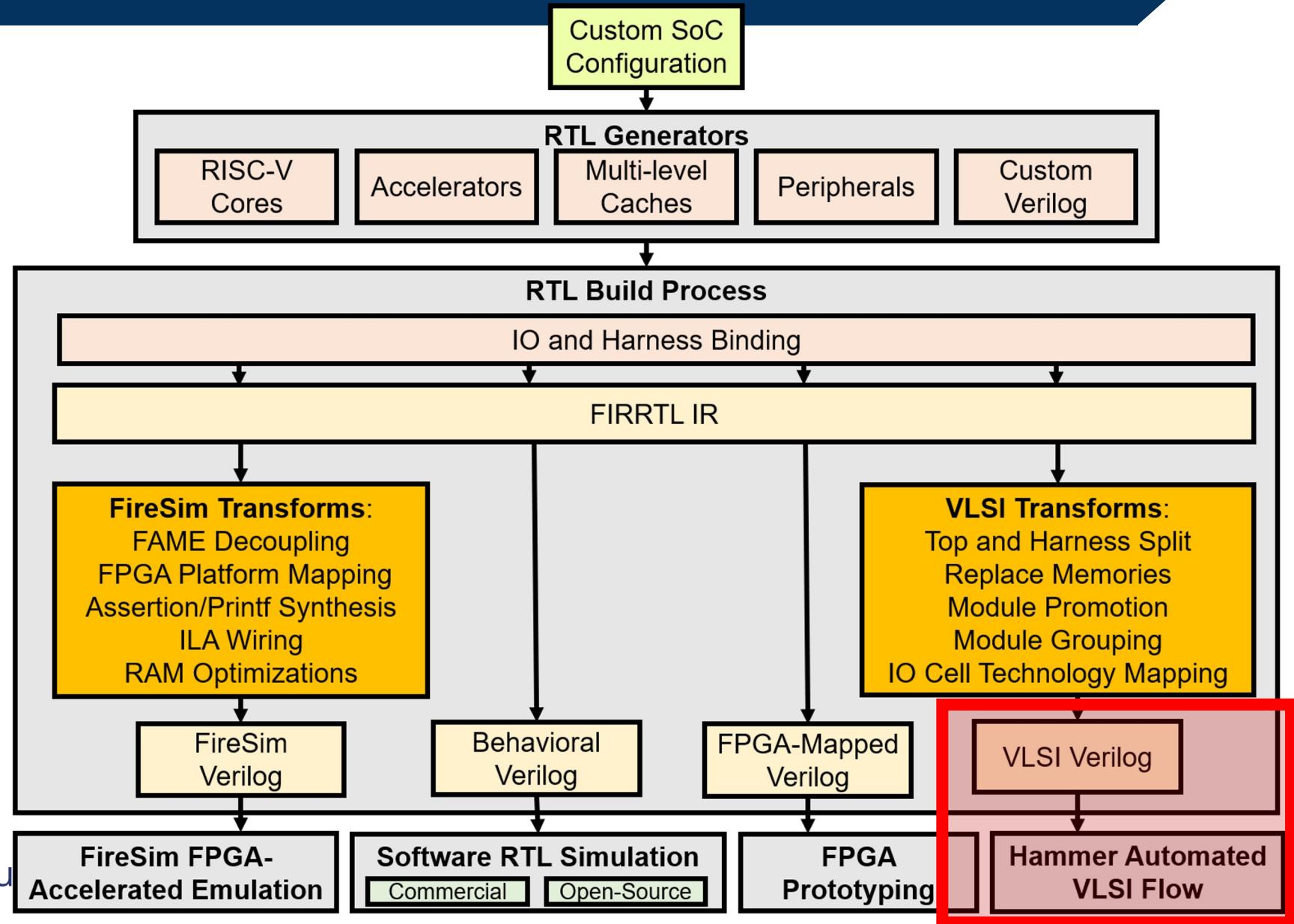
jerryz123@berkeley.edu



Berkeley
Architecture
Research

CHIPYARD

Tutorial Roadmap



Goals



- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Hammer Example
- Hammer community development



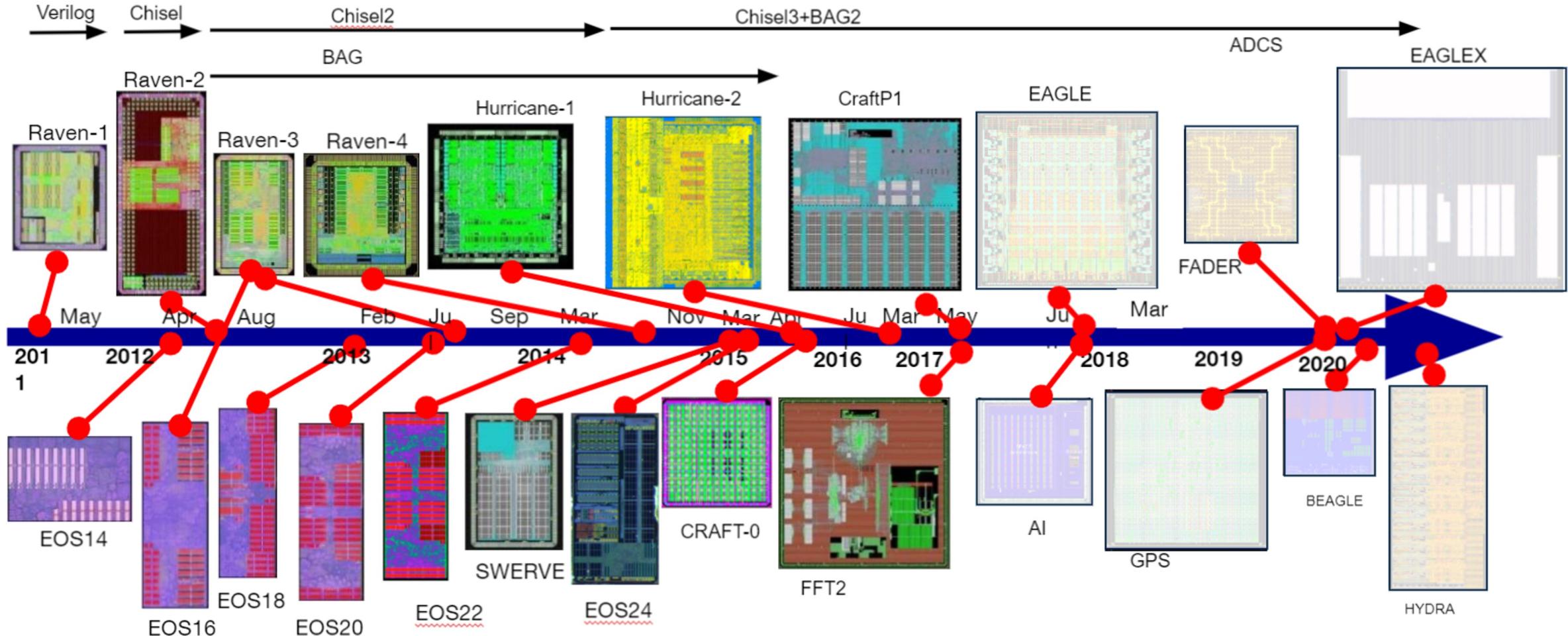
Goals



- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Hammer Example
- Hammer community development



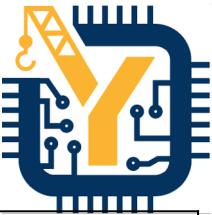
Hammer for Real Tapeouts



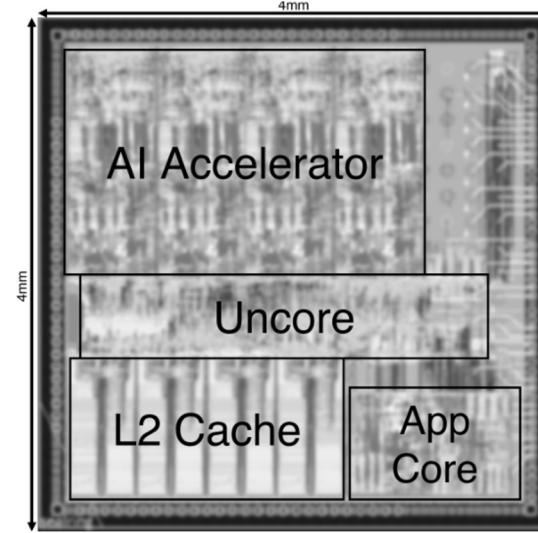
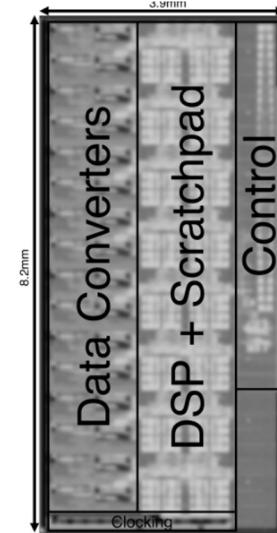
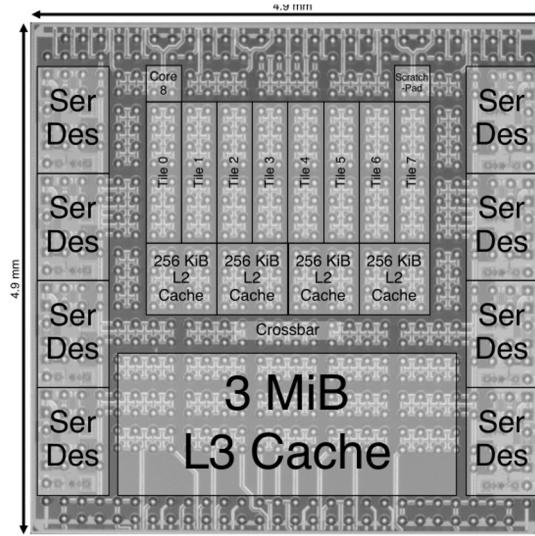
Raven, Hurricane: ST 28nm FDSOI, SWERVE: TSMC 28nm EOS: IBM 45nm SOI, CRAFT: 16nm TSMC,



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



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



Berkeley Architecture Research
Eagle [1]

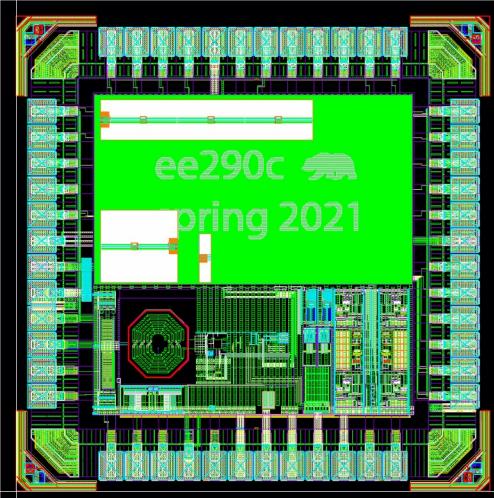
WaterSerpent

MythicChip

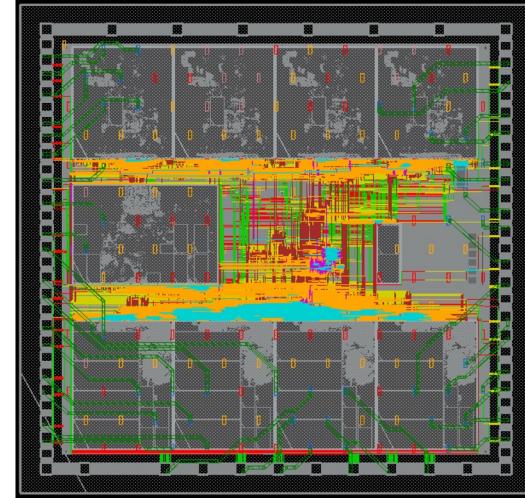
Hammer in Courses



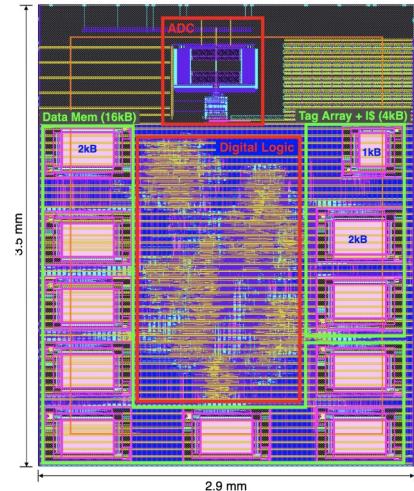
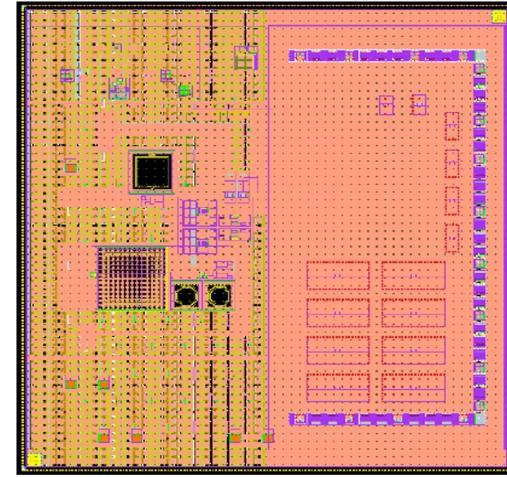
- Introduced in undergraduate digital circuits and systems labs:
 - <http://github.com/EECS150> (ASAP7 and Sky130 plugins)
- Special topics ‘tapeout’ class
 - Spring 2022: 1 sophomore, 15 juniors, 19 seniors, 3 5th-yr MS, 2 MEng, 1 PhD



2021 EE194/290C: OsciBear
TSMC 28nm



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



Sky130 MPW-2
Skywater 130nm



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

^CCadence

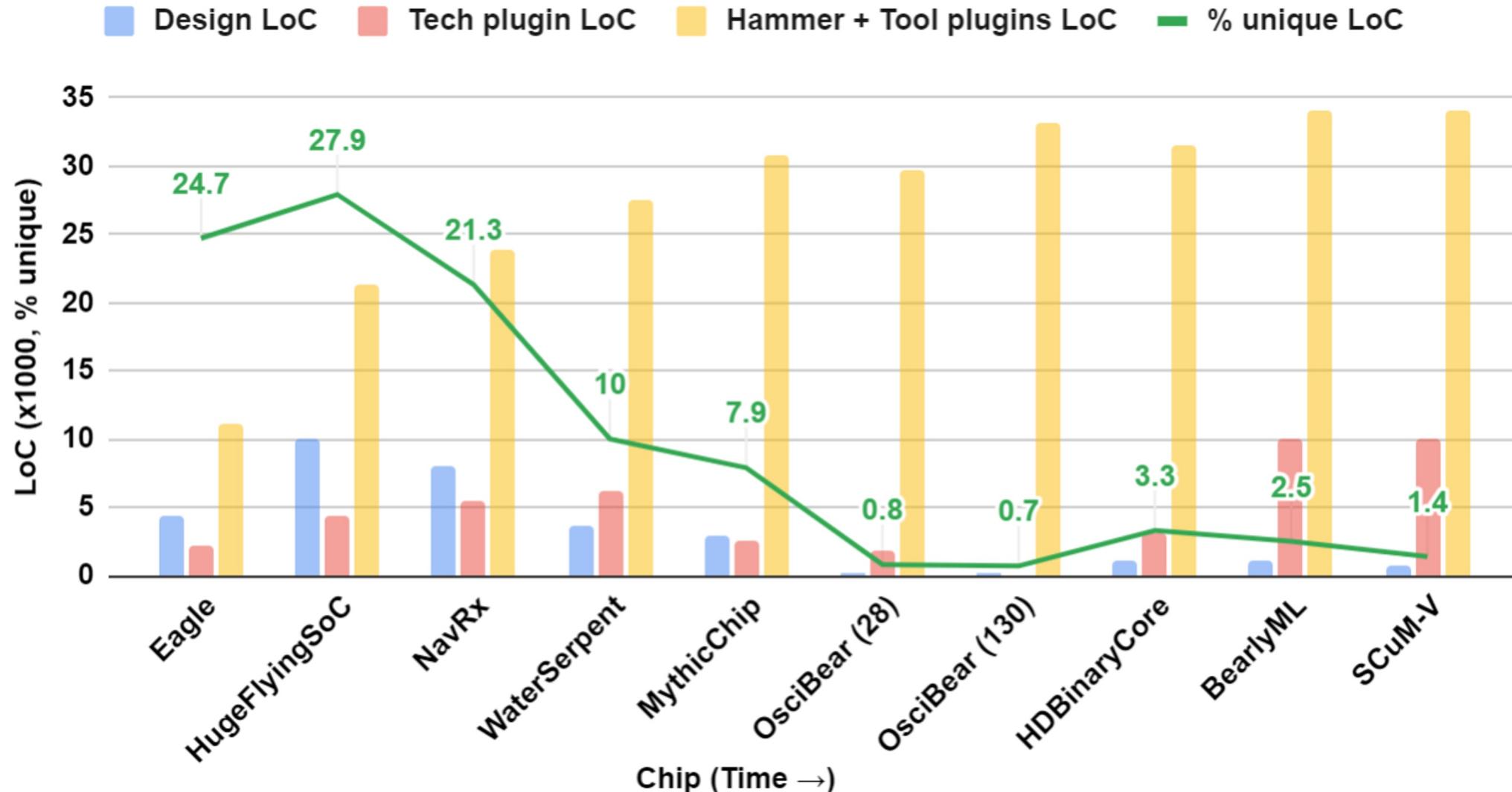
^SSynopsys

^MSiemens Mentor

^XXilinx

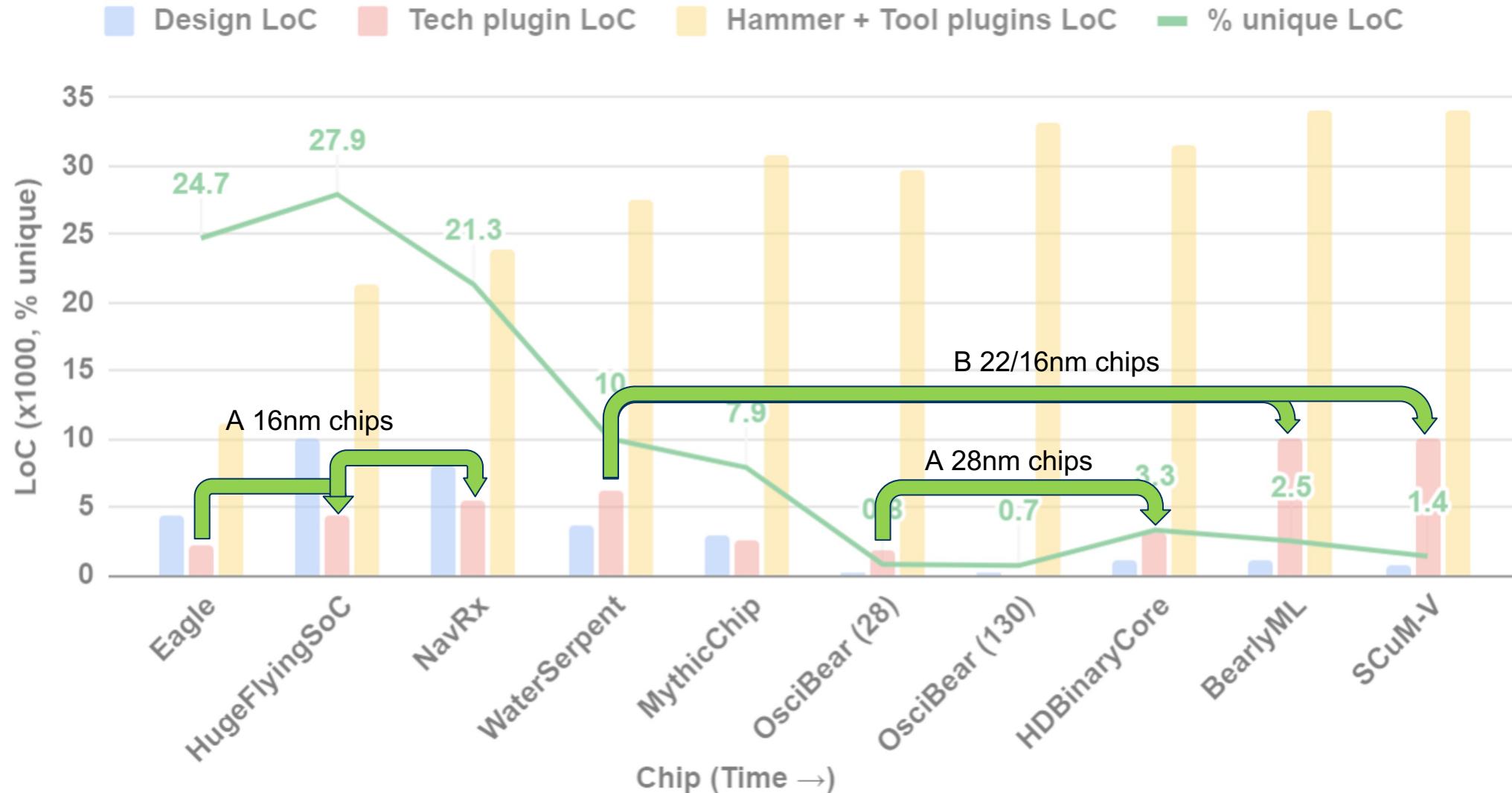


Increasing Flow Reusability





Tech Plugin Reusability



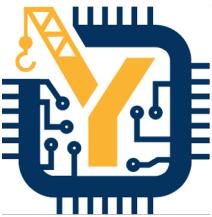
Goals



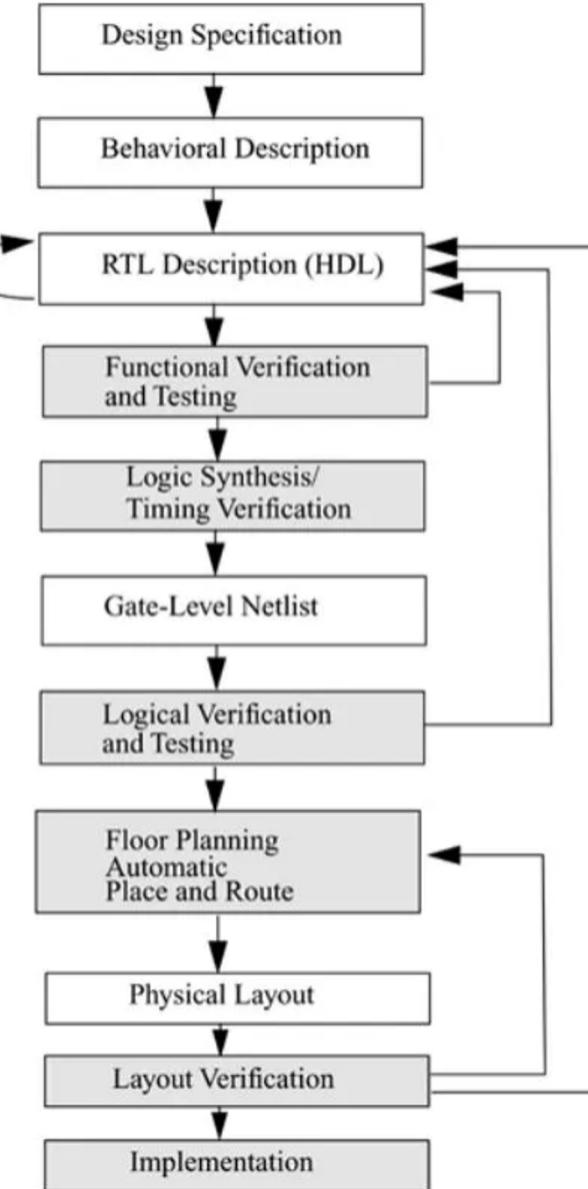
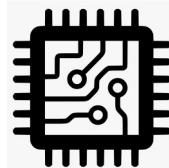
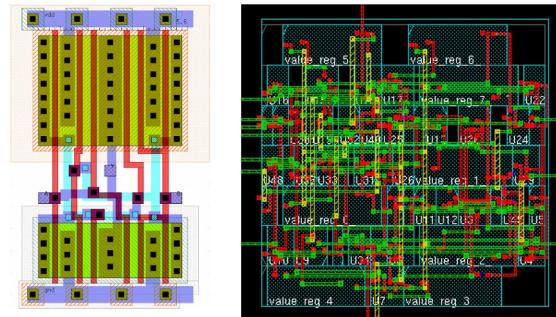
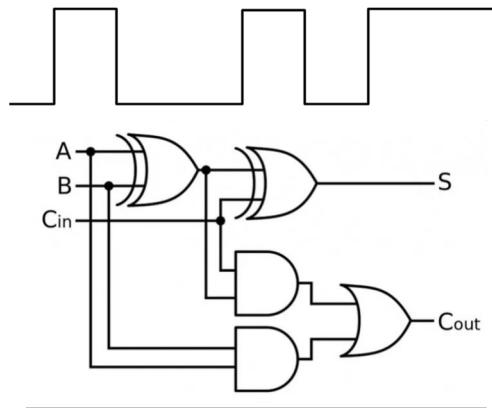
- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Hammer Example
- Hammer community development



ASIC Flow



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



synthesis

place-and-route

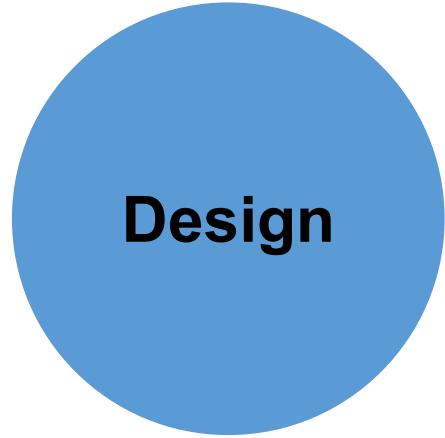
DRC

LVS



Berkeley Architecture Research

Components of VLSI Flows



- RTL
- IP cores
- Floorplan
- Constraints
- ...





Components of VLSI Flows

Design

Tech

- RTL
 - IP cores
 - Floorplan
 - Constraints
 - ...
- 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*

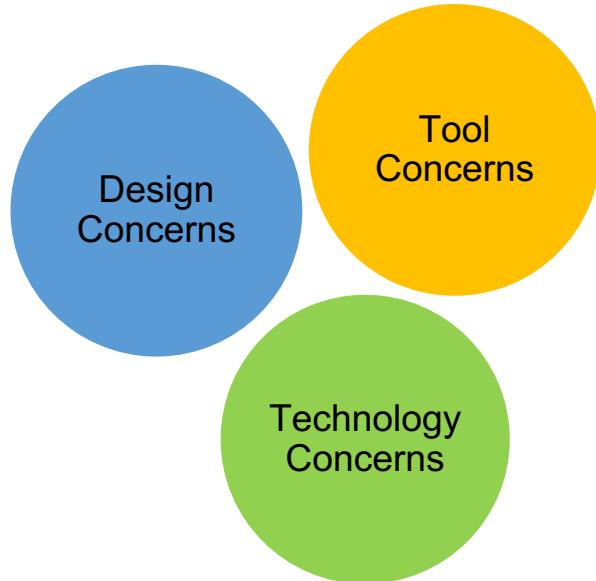




Why is Physical Design so Hard?

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...

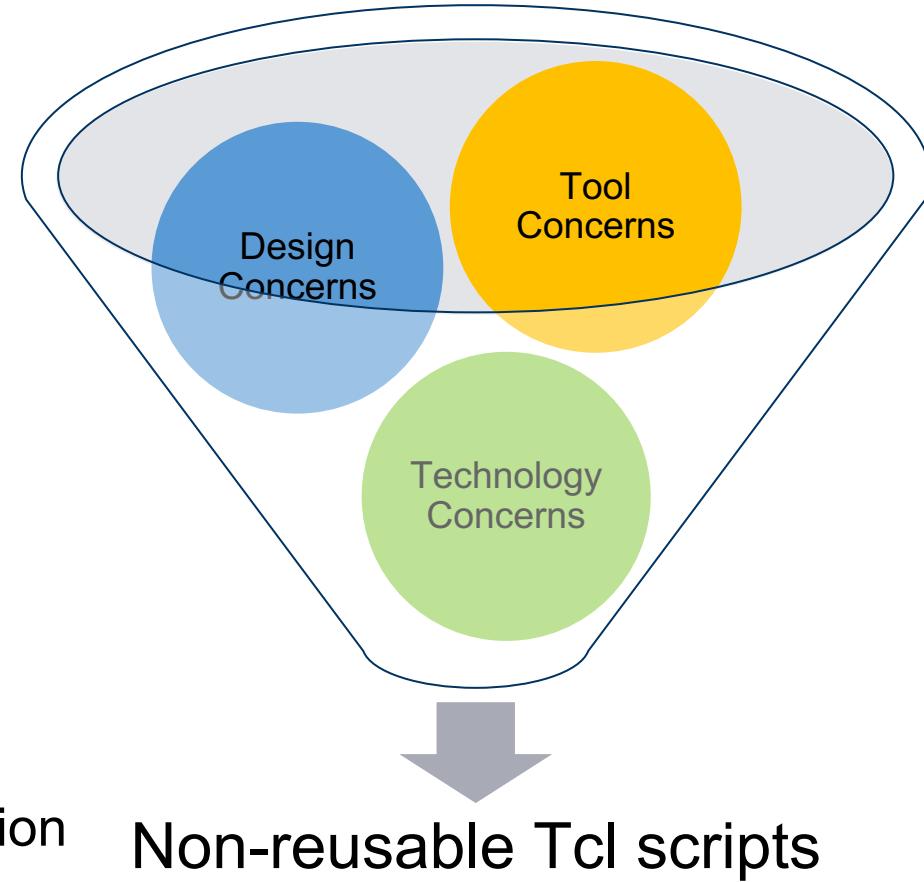




Why is Physical Design so Hard?

VLSI flows are *custom built* for each project

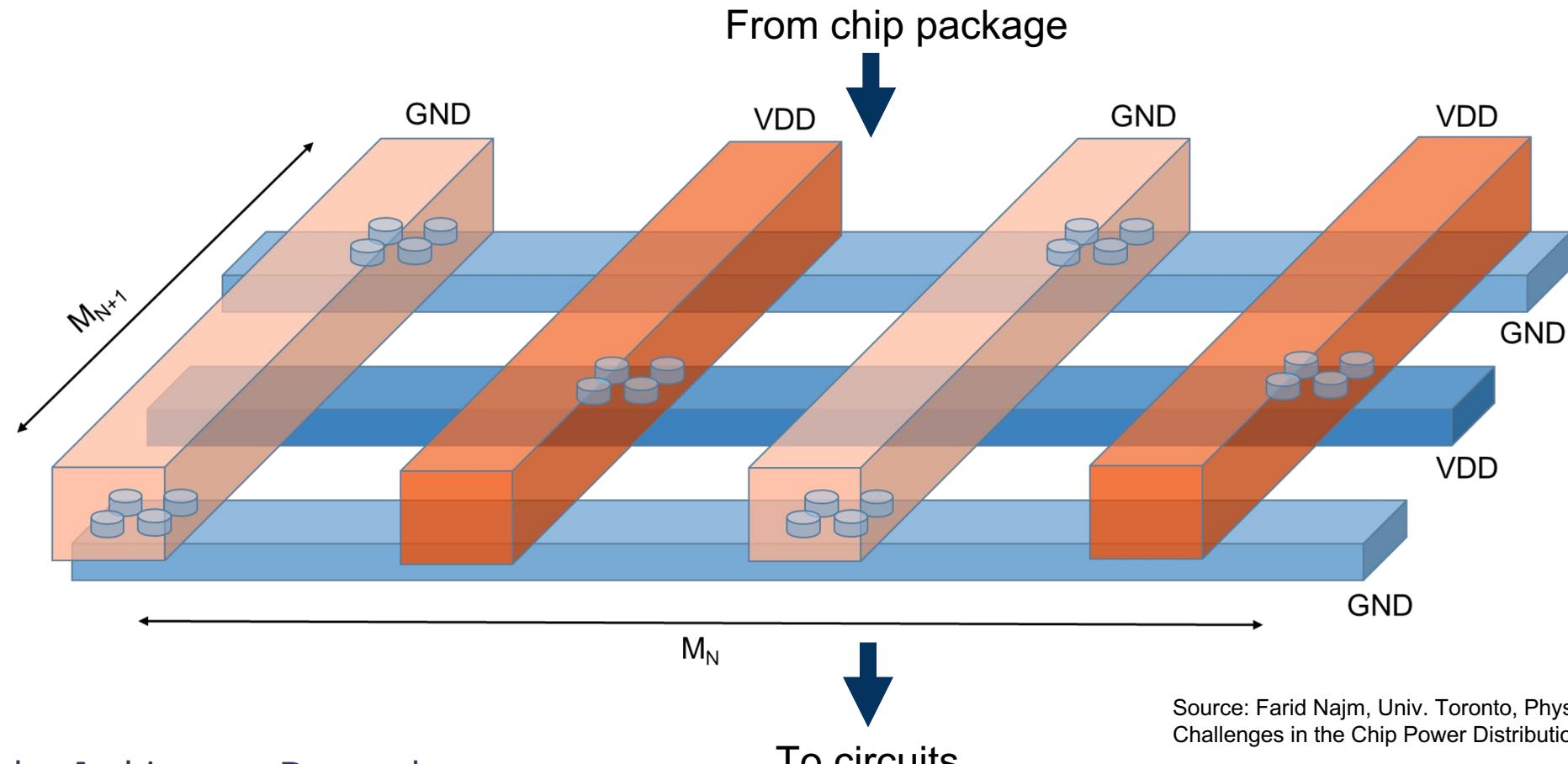
- Flows tailored to design/tech/tool combo
 - Different design? New constraints...
 - Switching tech? New rules, IP, SRAMs, ...
 - Updated tool? Different commands...
- Design intent & expertise tied up in scripts!
 - Today, wide range of technology options and domain specialization demand architectural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





Example: Power Straps

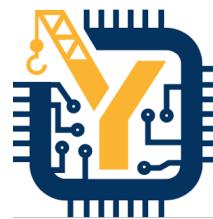
- How to distribute power from package to transistors



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



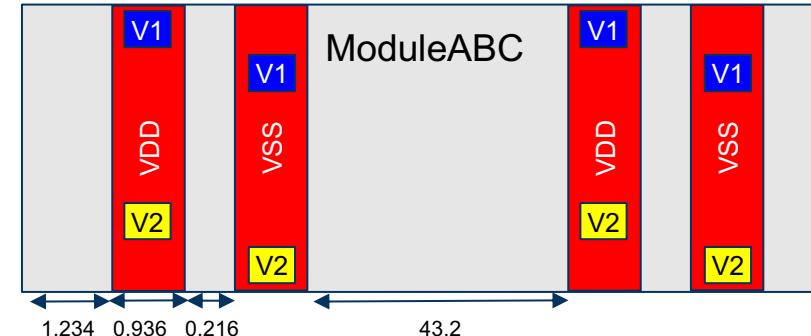
Example



- 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!
```



Example



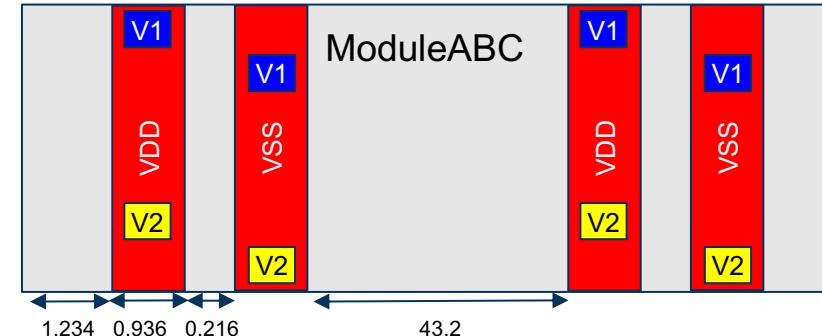
- Hypothetical power strap creation command:

```
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]
```

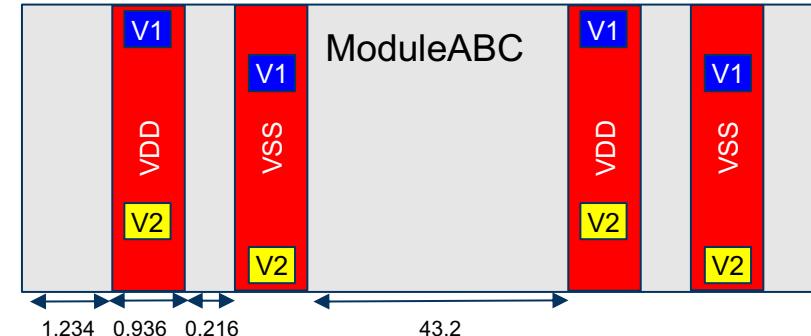
Repeat for each layer!

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

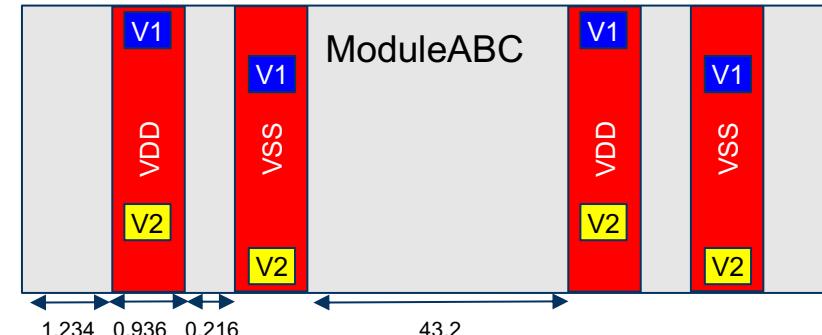
The command + options

Tech-specific

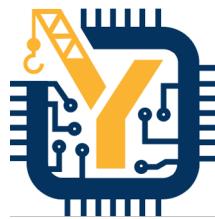
DRC/ERC-compliant dimensions

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
- Hammer Example
- Hammer community development

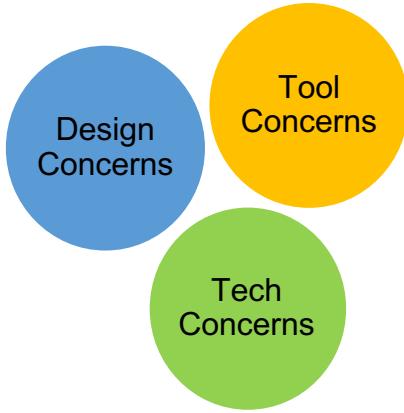


Hammer Design Principles



1. Separation of Concerns

- Decouple design-, tool-, and tech-specific concerns



Hammer Design Principles

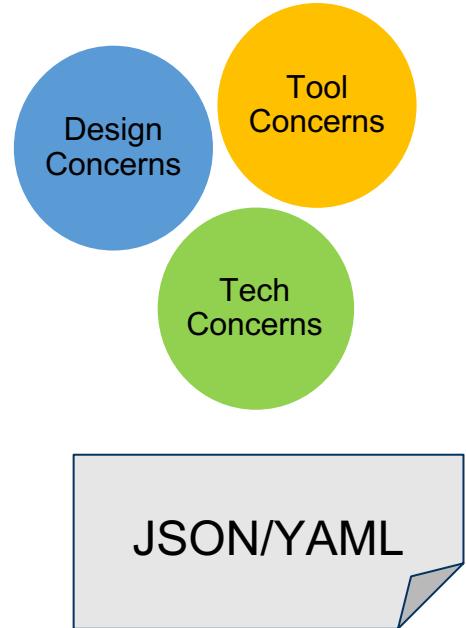


1. Separation of Concerns

- Decouple design-, tool-, and tech-specific concerns

2. Standardization

- Data interchange schema for constraints, options, files



Hammer Design Principles



1. Separation of Concerns

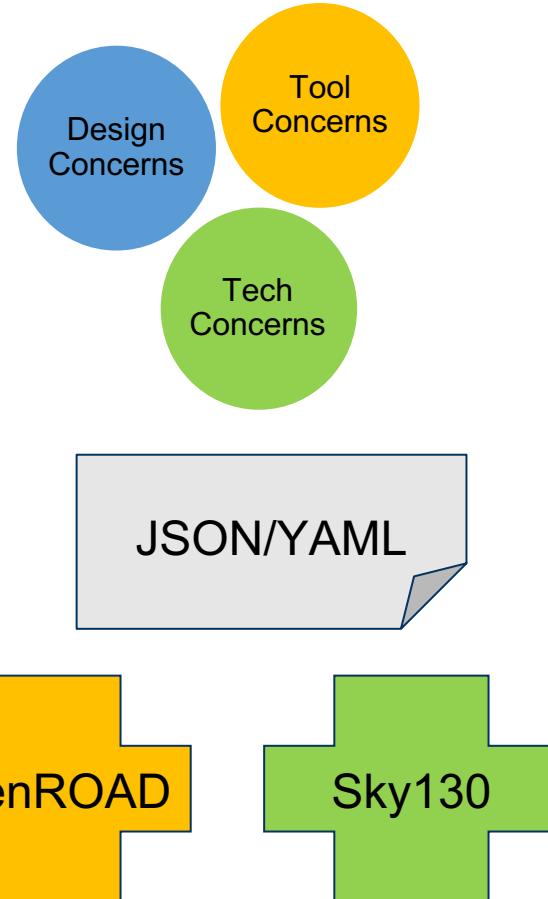
- Decouple design-, tool-, and tech-specific concerns

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

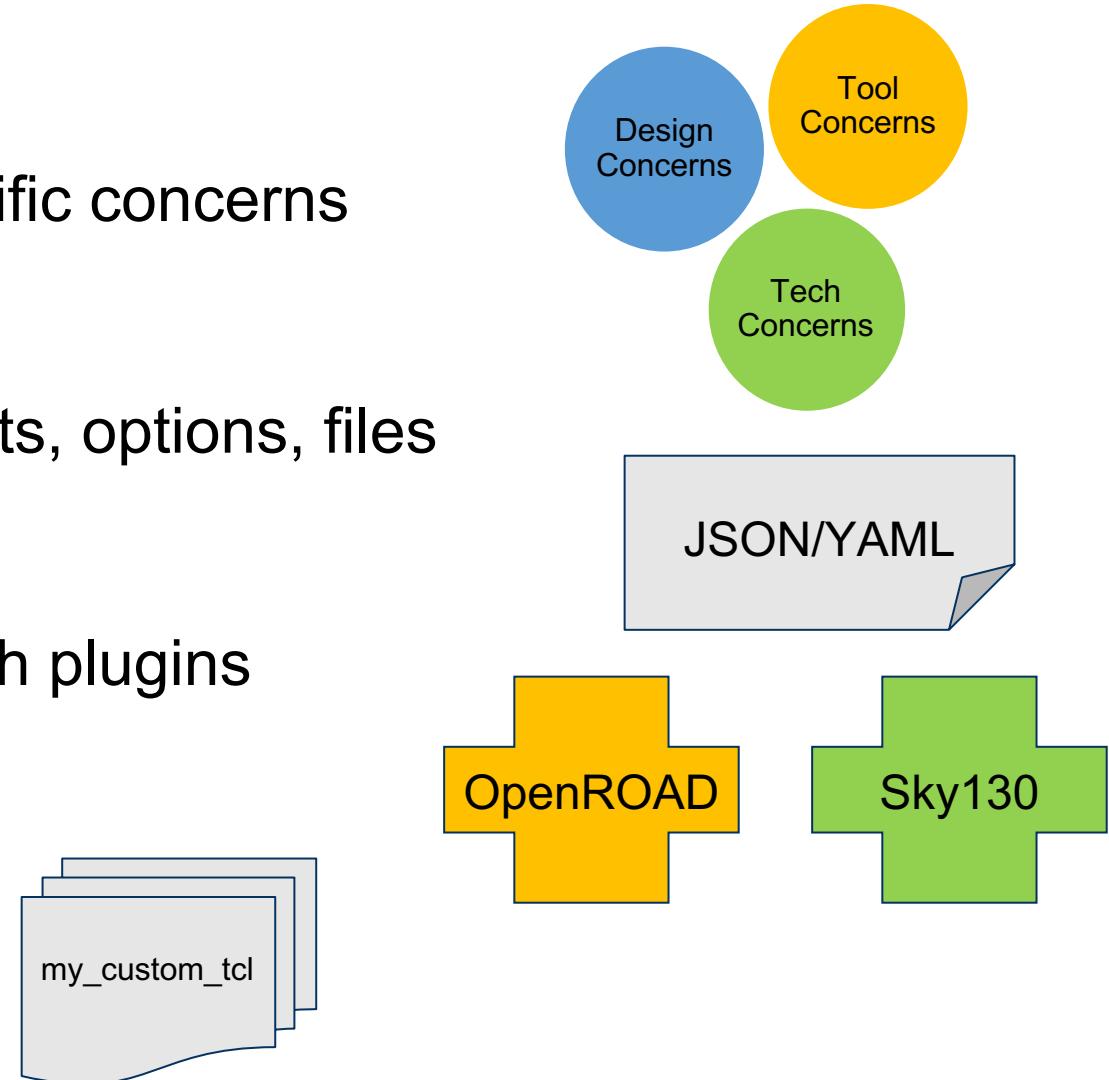
- Data interchange schema for constraints, options, files

3. Modularity

- Interchangeable & shareable tool & tech plugins

4. Incremental Adoption

- Mix reusable & custom solutions

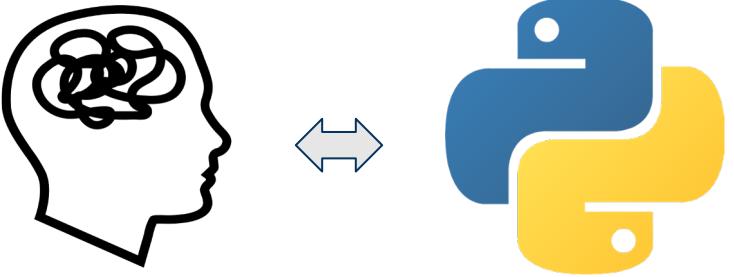




What is Hammer?

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?

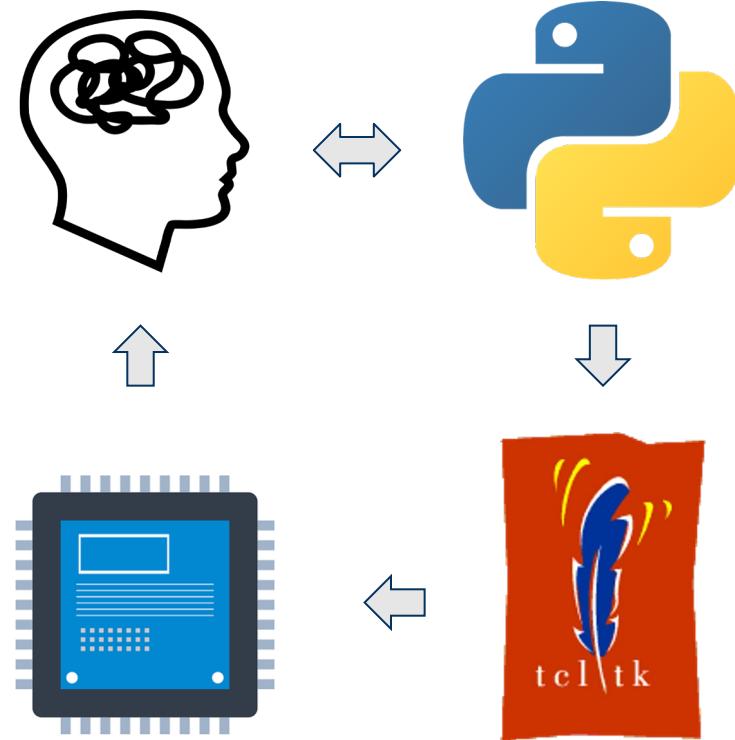


Hammer is:

... 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?



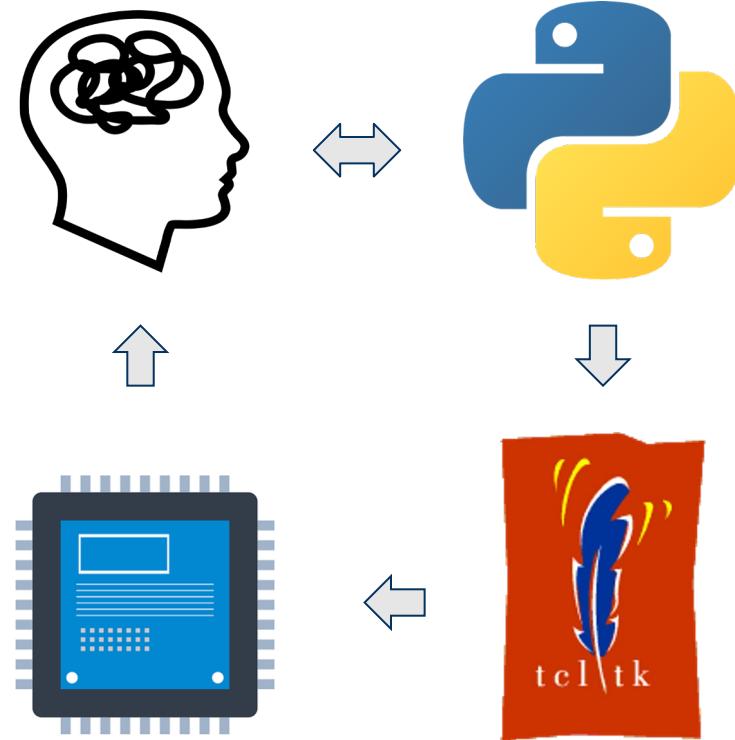
Hammer is:

... 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?

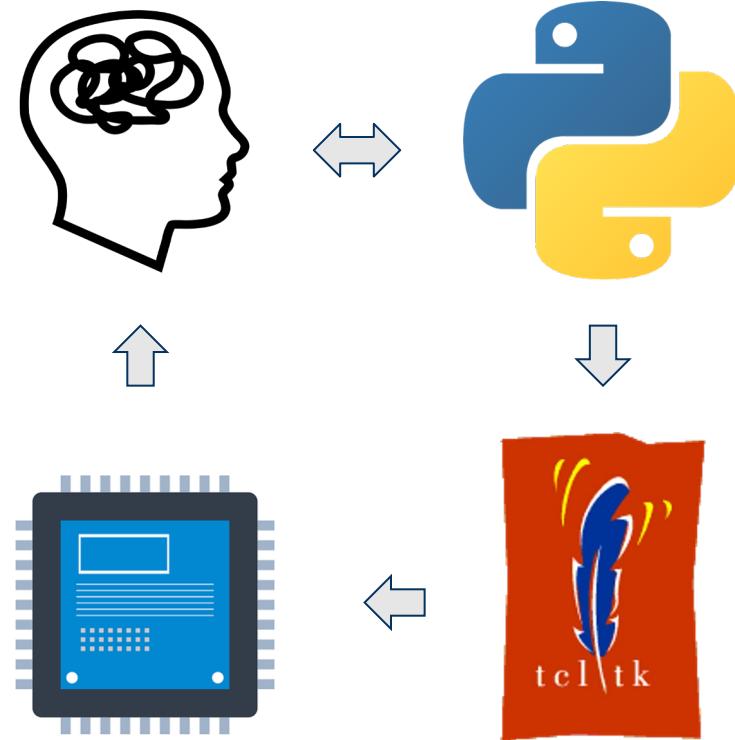
Hammer is:

... 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!

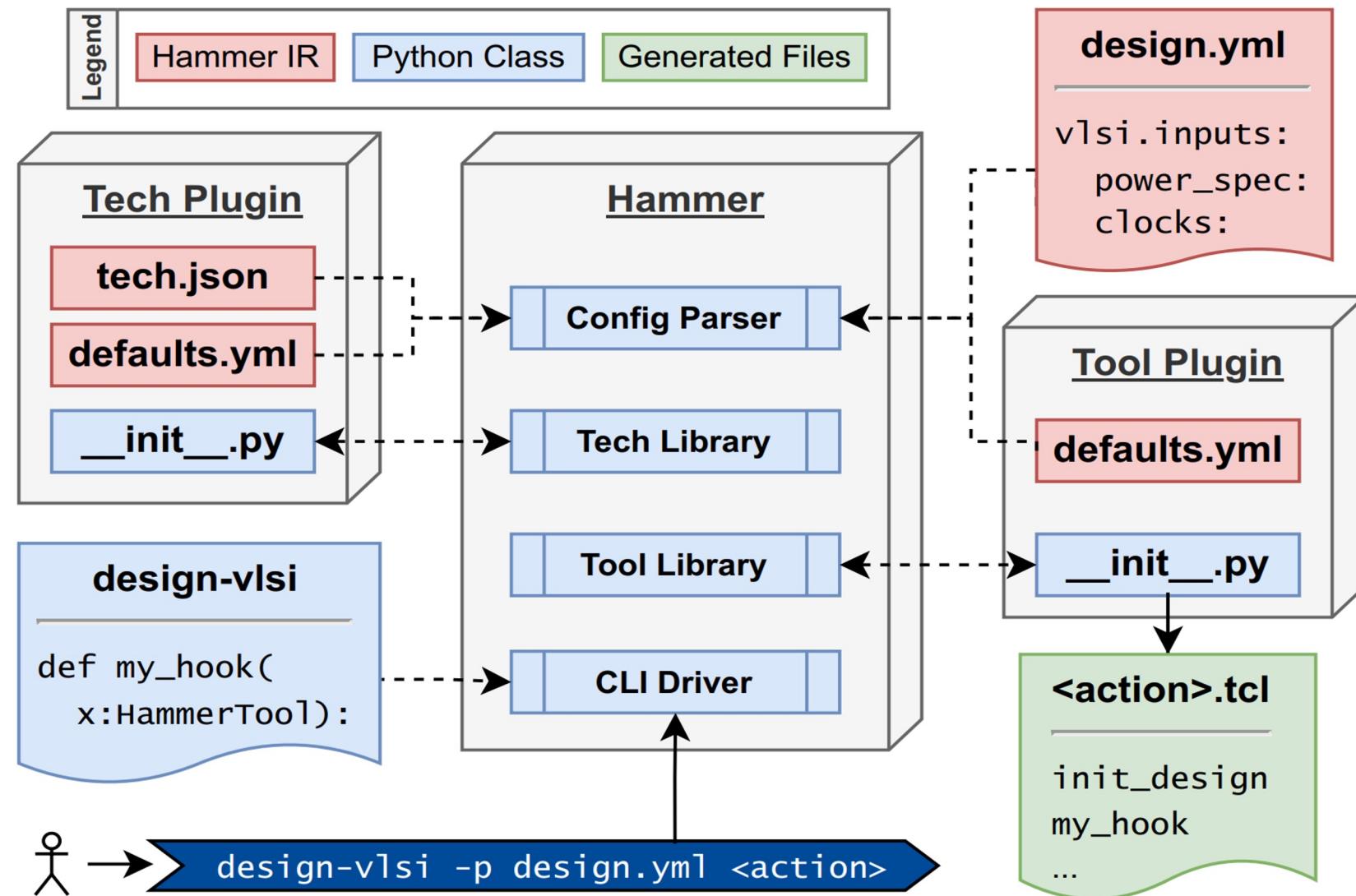


Project started in 2015, research chip use from 2016, class use from 2019, currently ~35k lines of code



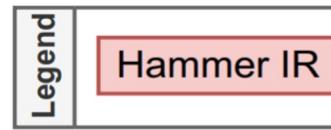


Hammer Software Architecture





Hammer Intermediate Representation (IR)



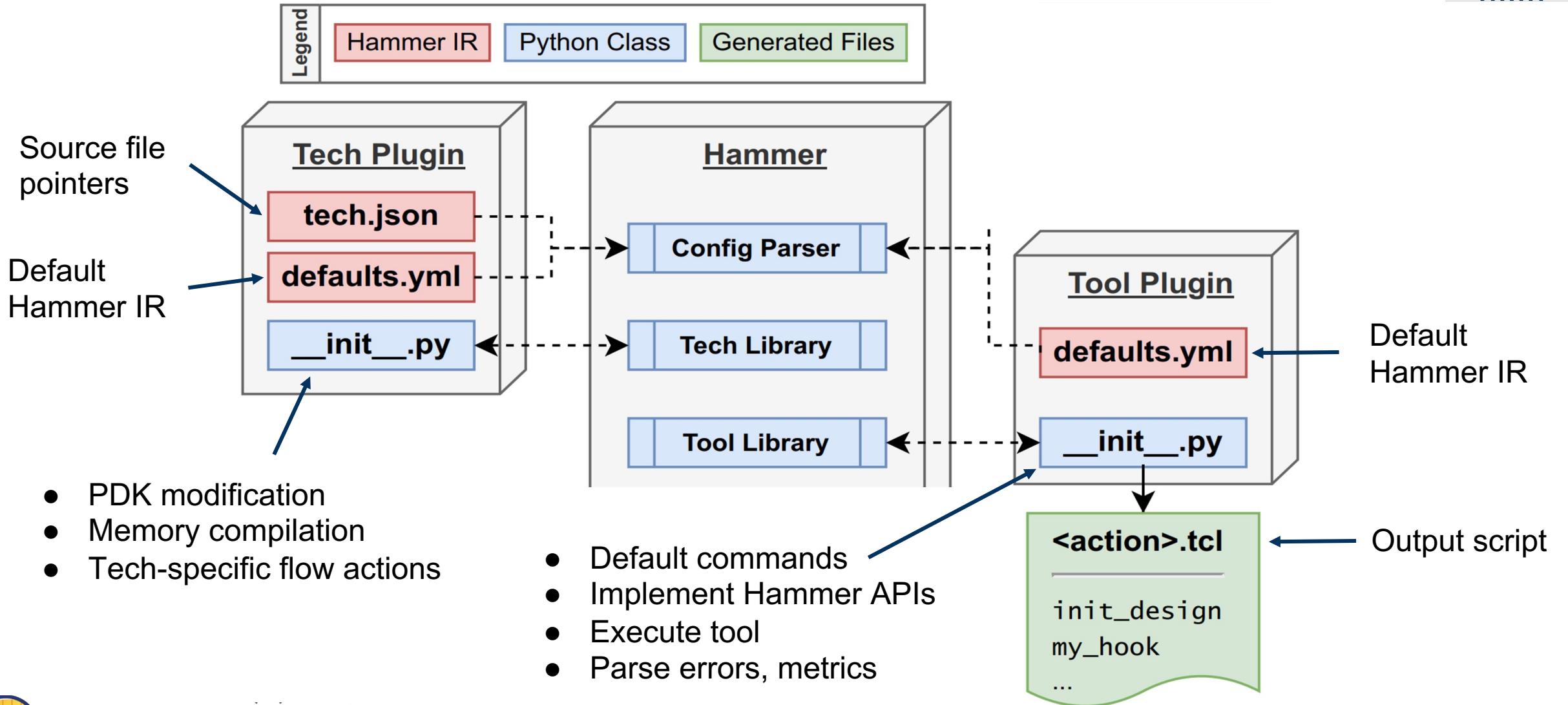
- Standard data interchange format
 - Constraints, options, intermediate files, etc.
 - YAML for humans, JSON for programs (annotation format)
 - **De-embeds designer intent and expertise from Tcl scripts**
- IR Metaprogramming
 - Modify any IR key with traceable history, type- and validity-checking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

design.yml

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



Tool and Tech Plugins



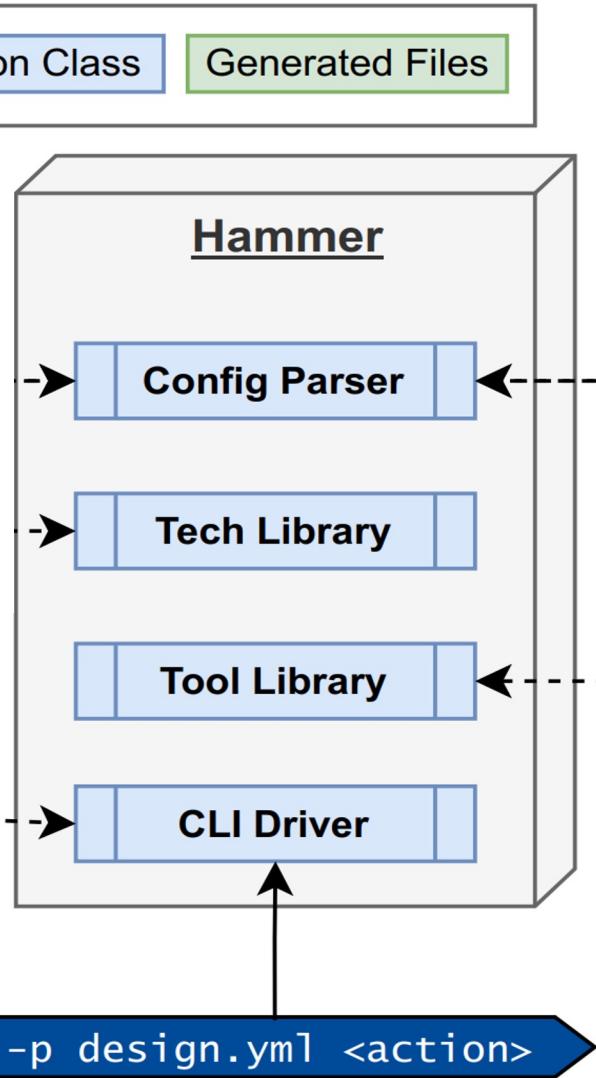
Hooks and Drivers



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 Driver

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



Goals



- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- **Hammer Example**
- Hammer community development





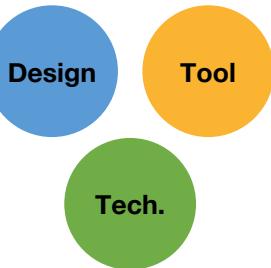
Power Straps Example

- To specify power straps, need to know:

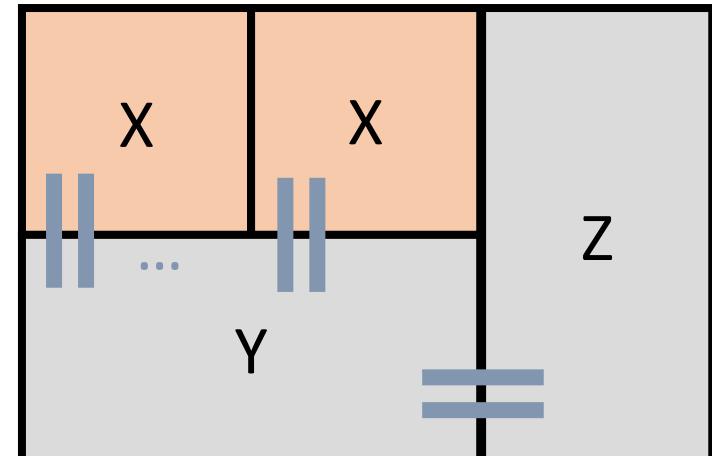
- DRC rules
- Target power dissipation
- IR drop spec
- Domain areas

```
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]
```

Separated Concerns



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

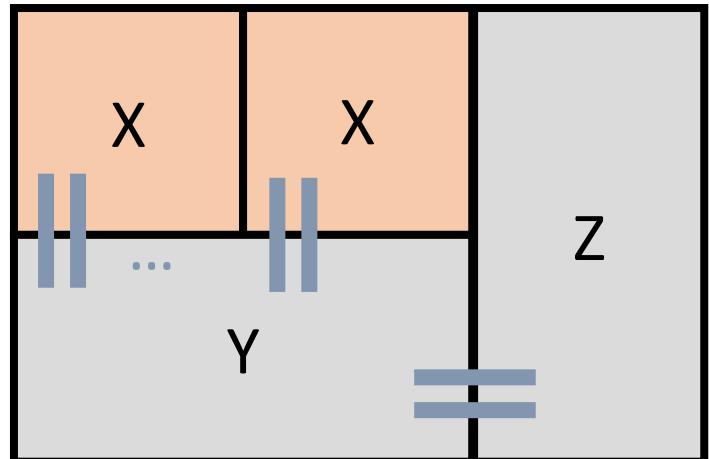
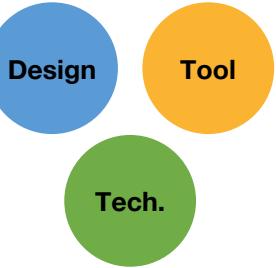




Power Straps Example

- 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

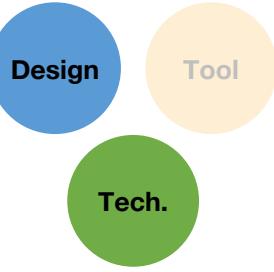
Separated
Concerns



Power Straps Example

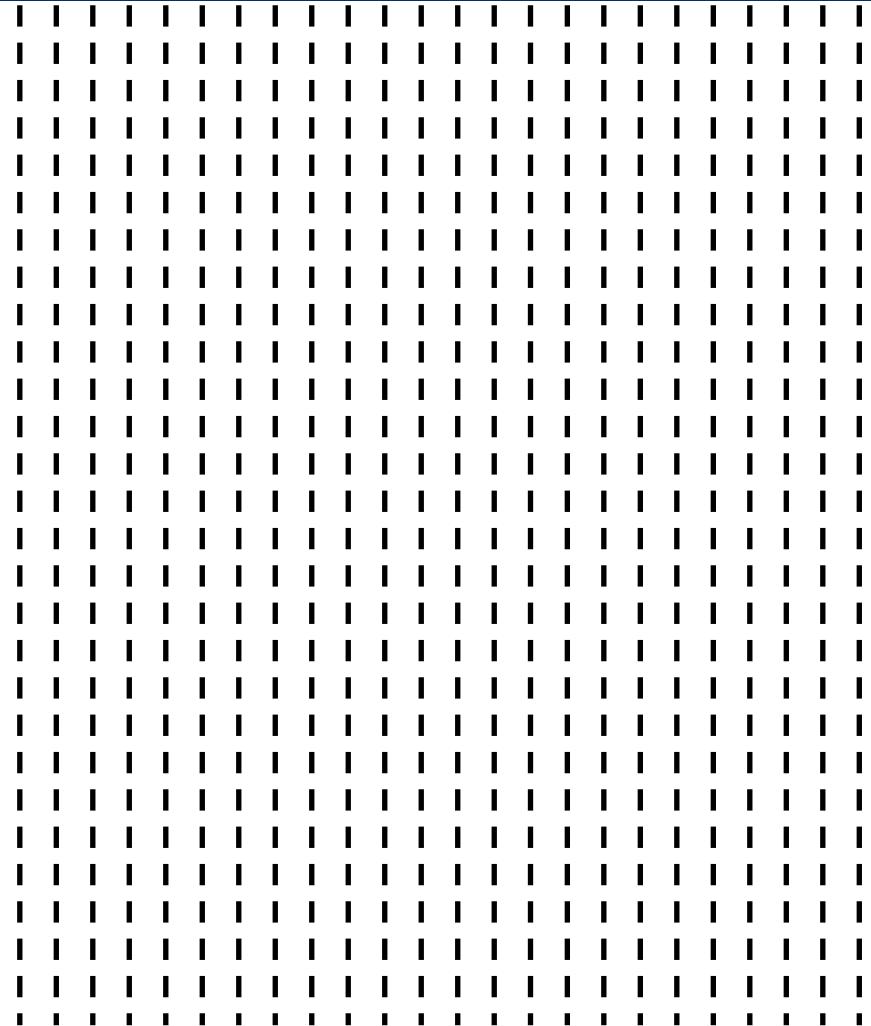


Separated
Concerns

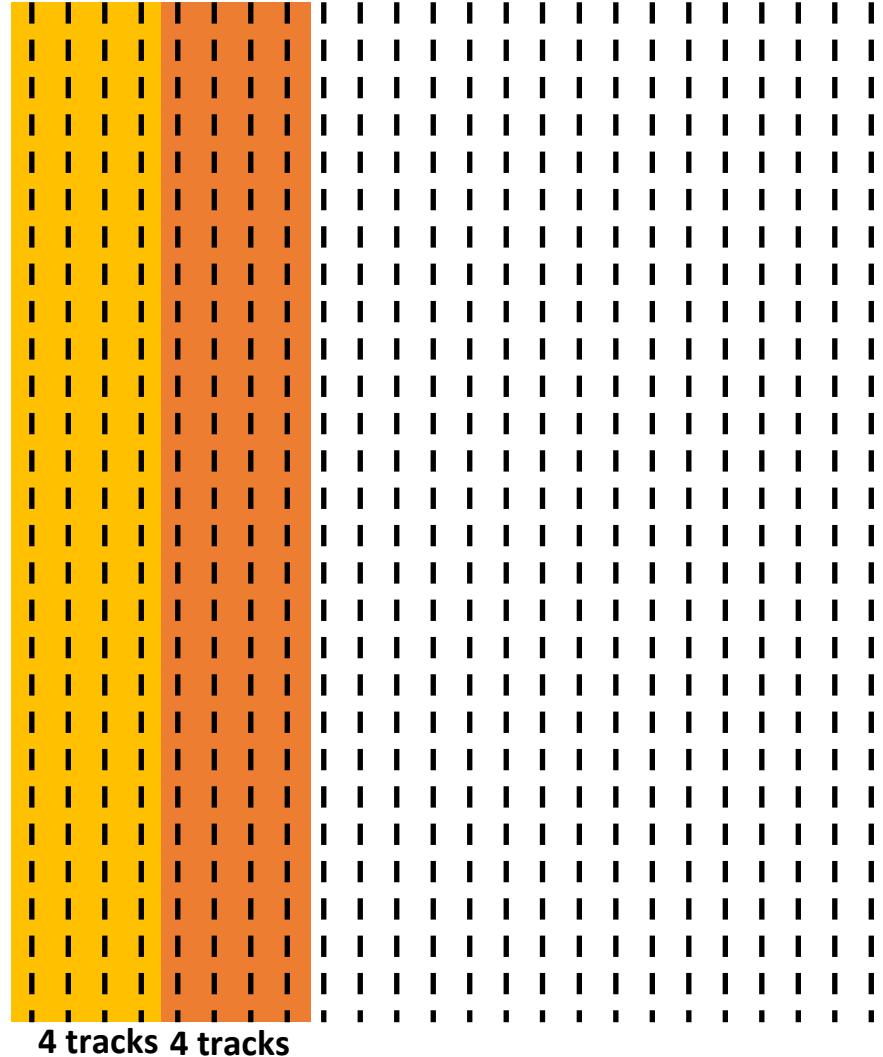


Choose power
strap strategy

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



Power Straps Example



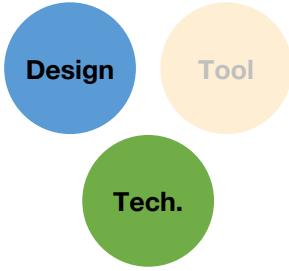
```
par.generate_power_straps_options:  
  by_tracks:  
    track_width: 4
```

Allocate tracks

```
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

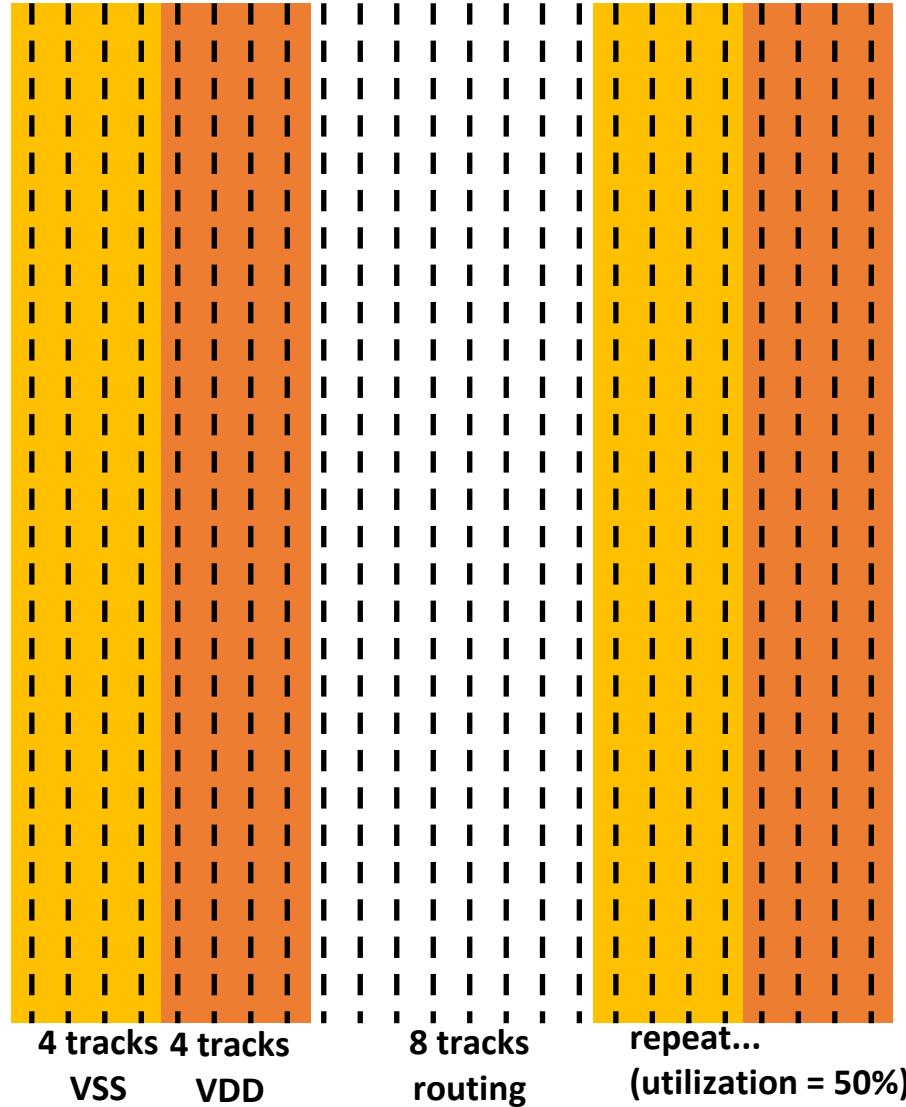
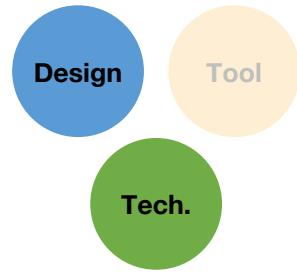
Separated
Concerns



Power Straps Example



Separated
Concerns



Determine pitch

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

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

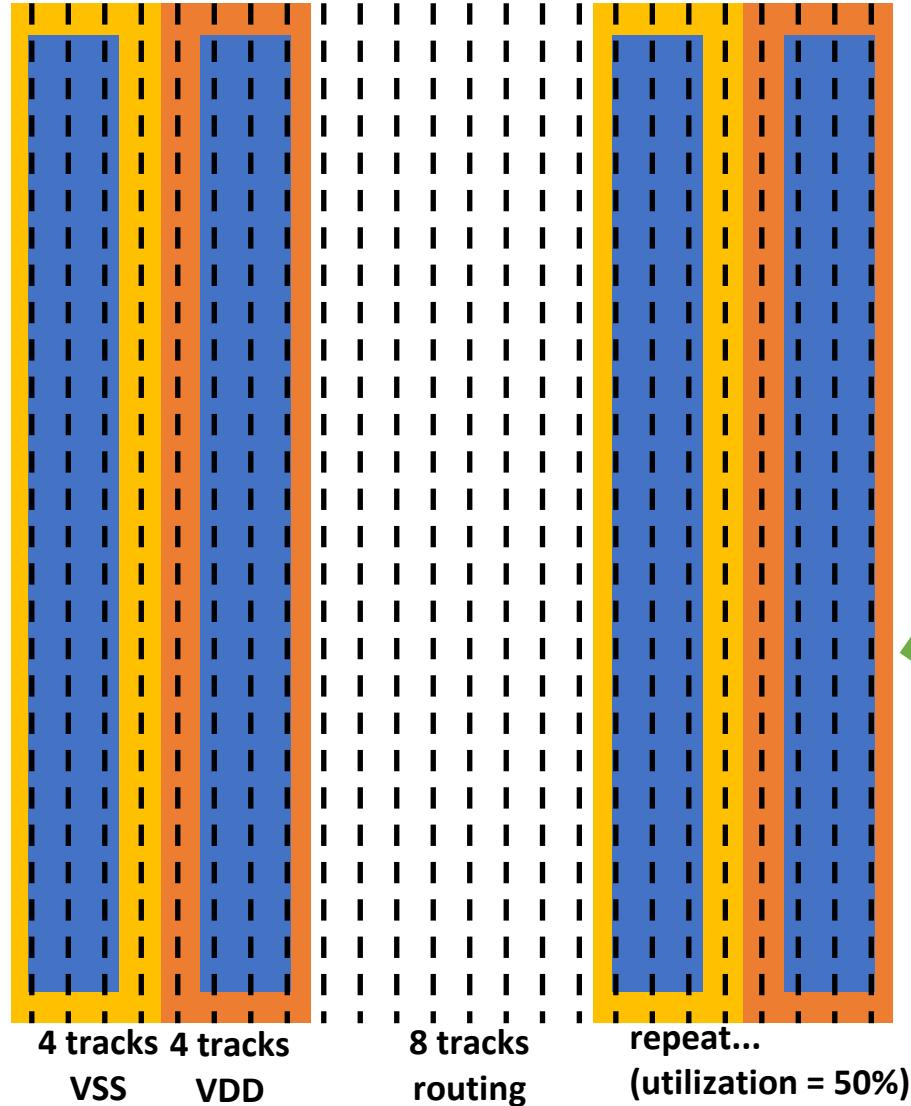
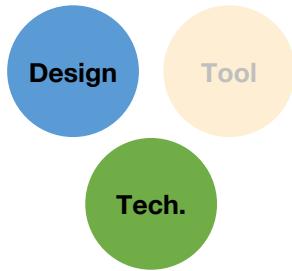
Group pitch = tracks per group / utilization
 $= 8 / 0.5 = 16$



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

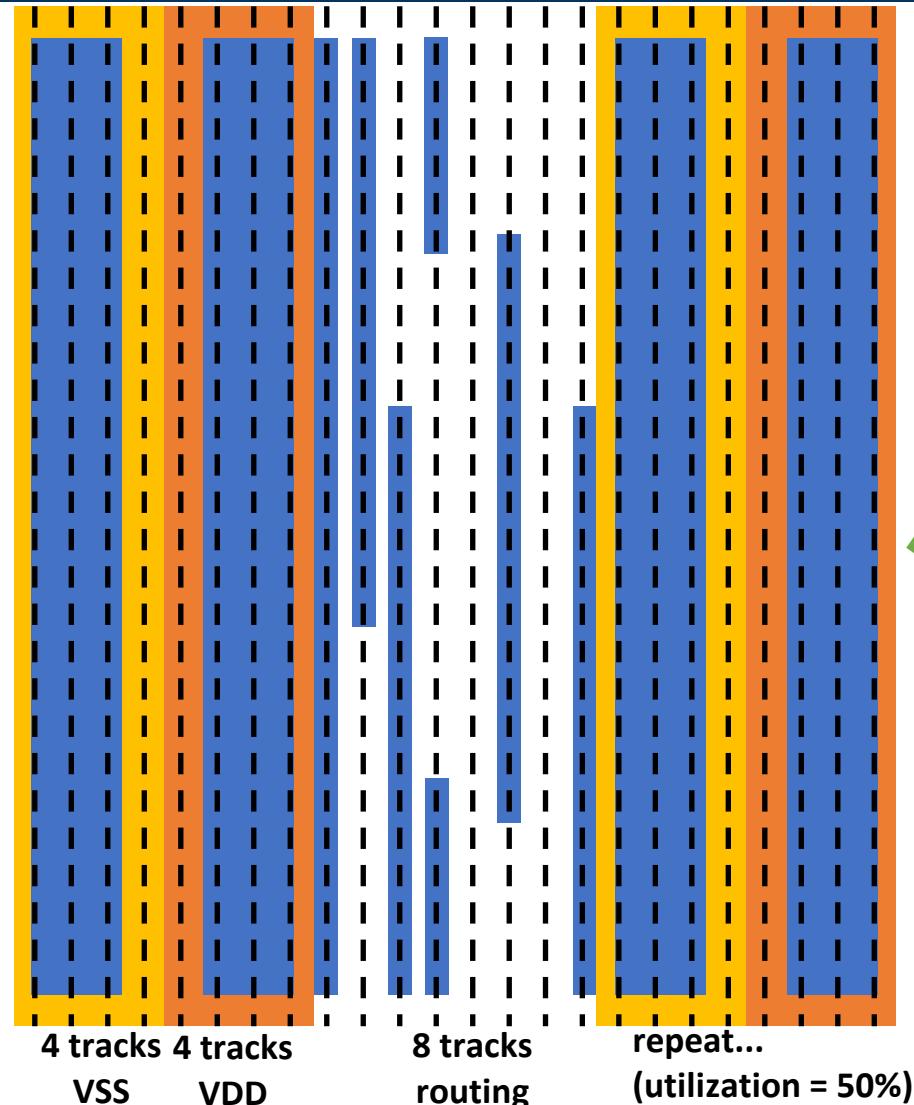
```
par.generate_power_straps_method: by_tracks
```

```
par.power_straps_mode: generate
```





Power Straps Example



par.generate_power_straps_options:
by_tracks:

track_width: 4

power_utilization: 0.5

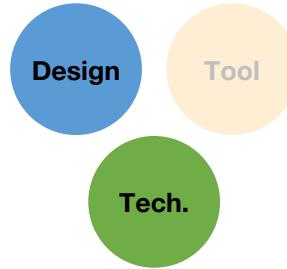
strap_layers:

- M3
- M4
- M5
- M6
- M7
- M8
- M9

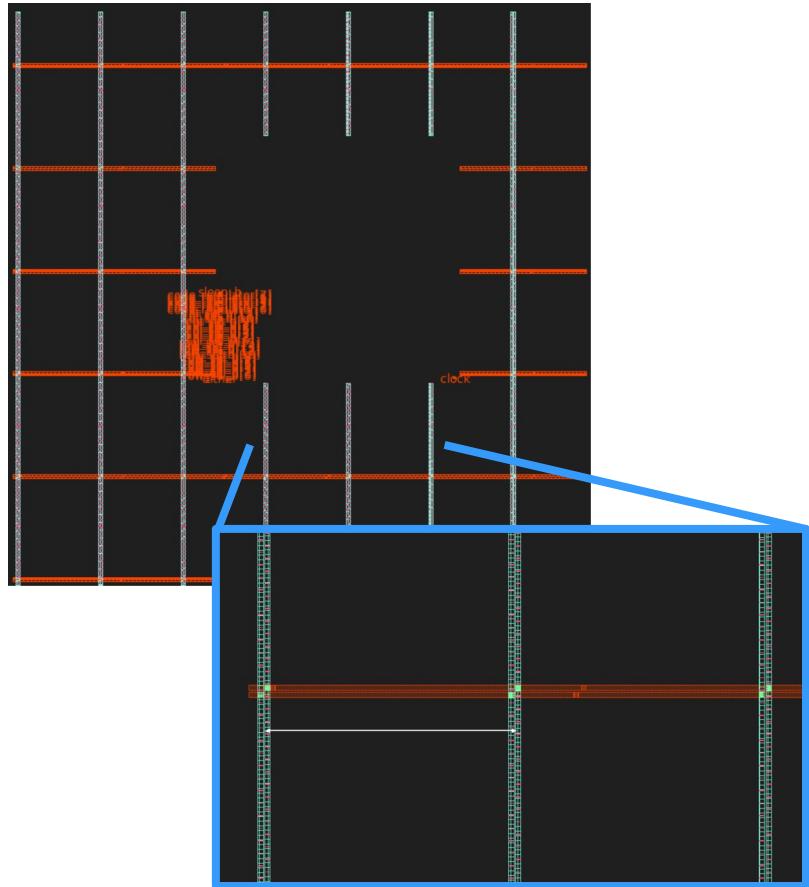
par.generate_power_straps_method: by_tracks

par.power_straps_mode: generate

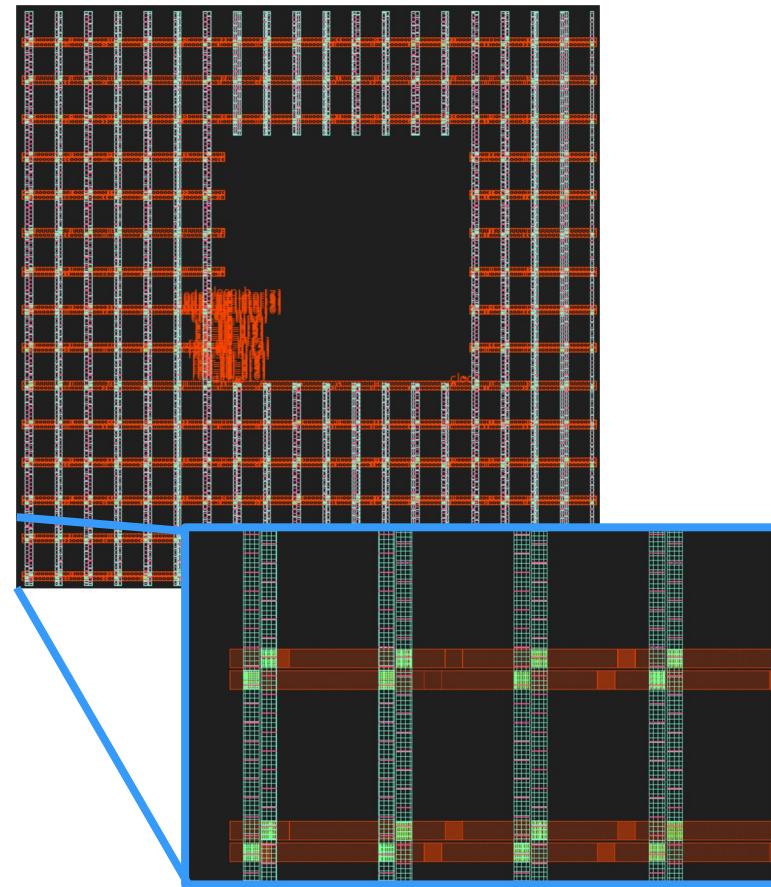
**Separated
Concerns**



Power Straps Example



power_utilization: 0.1



power_utilization: 0.3



Goals



- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Hammer Example
- Hammer community development





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

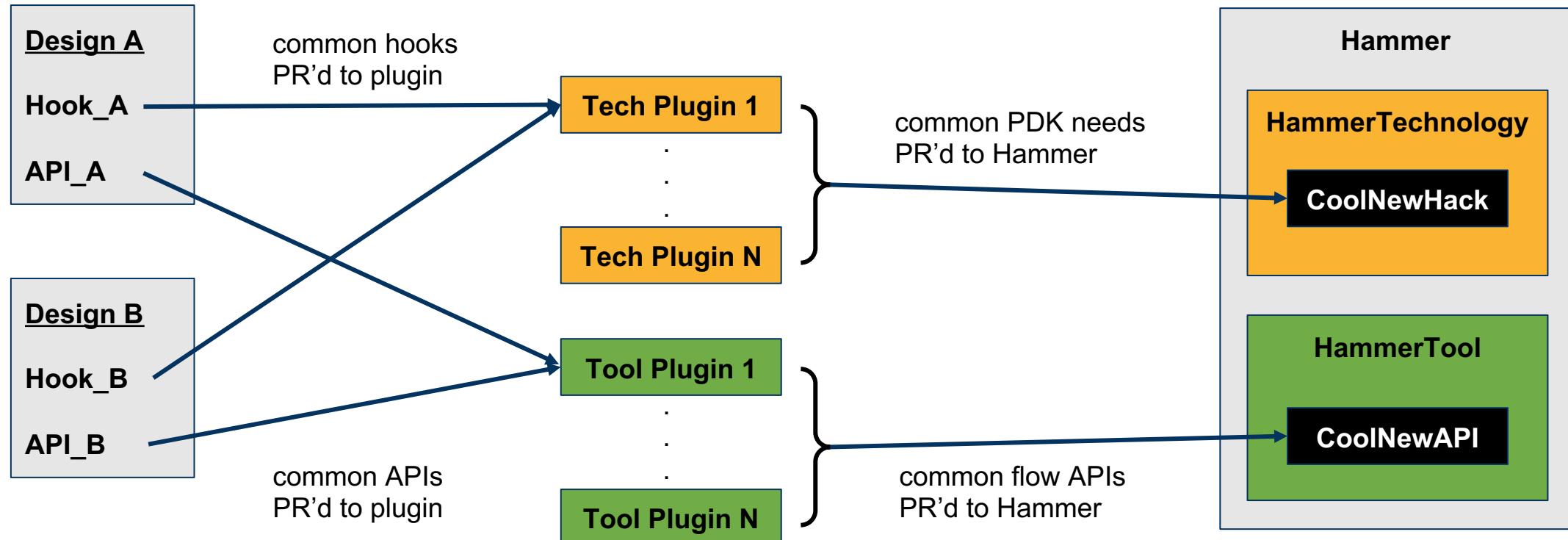
- Easily learn the VLSI flow, get early design feedback
- [Chipyard examples](#) with ASAP7, Sky130



Everyone can contribute to Hammer



Modular design + override features = community contributions





Recent Hammer Changes

- Published to PyPi
 - installed during chipyard ./build-setup.sh
 - pip install hammer-vlsi
- Cadence/Synopsys plugins merged into main Hammer

The screenshot shows the PyPi project page for 'hammer-vlsi 1.1.0'. The page has a blue header with navigation links for Help, Sponsors, Log in, and Register. The main title is 'hammer-vlsi 1.1.0' with a green 'Latest version' button. Below the title is a search bar and a pip installation link. A project description states: 'Hammer is a physical design framework that wraps around vendor specific technologies and tools to provide a single API to create ASICs.' On the left, there's a 'Navigation' sidebar with 'Project description' (selected), 'Release history', and 'Download files'. At the bottom, there are 'Project links' for 'Homepage' and 'Repository'. To the right, there's a large 'HAMMER' logo with a stylized hammer icon.

Search projects

Help Sponsors Log in Register

hammer-vlsi 1.1.0

✓ Latest version

Released: Mar 20, 2023

Hammer is a physical design framework that wraps around vendor specific technologies and tools to provide a single API to create ASICs.

Navigation

Project description

Release history

Download files

Project links

Homepage

Repository

hammer-pr-ci passing hammer-publish-ci passing

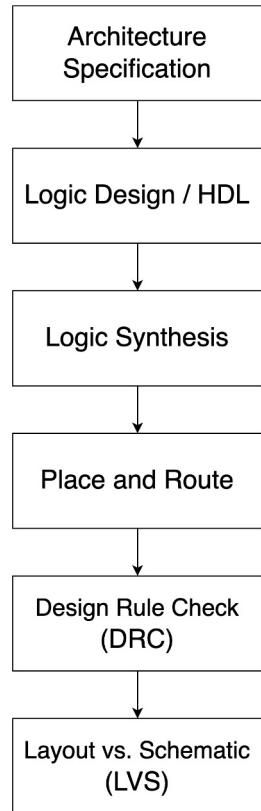
Highly Agile Masks Made Effortlessly from RTL (Hammer)



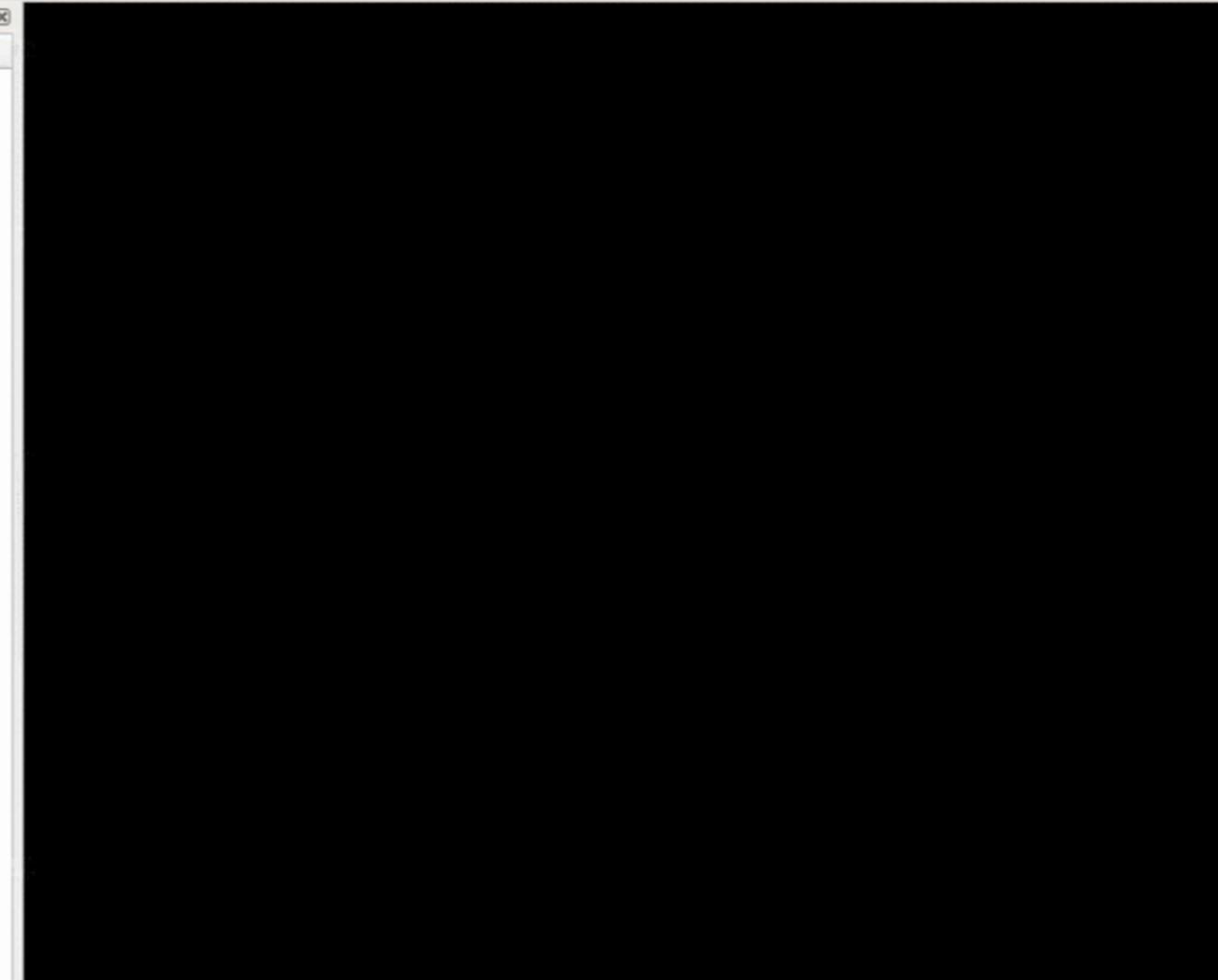


Tutorial: TinyRocket RTL-to-GDS

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



NETGEN

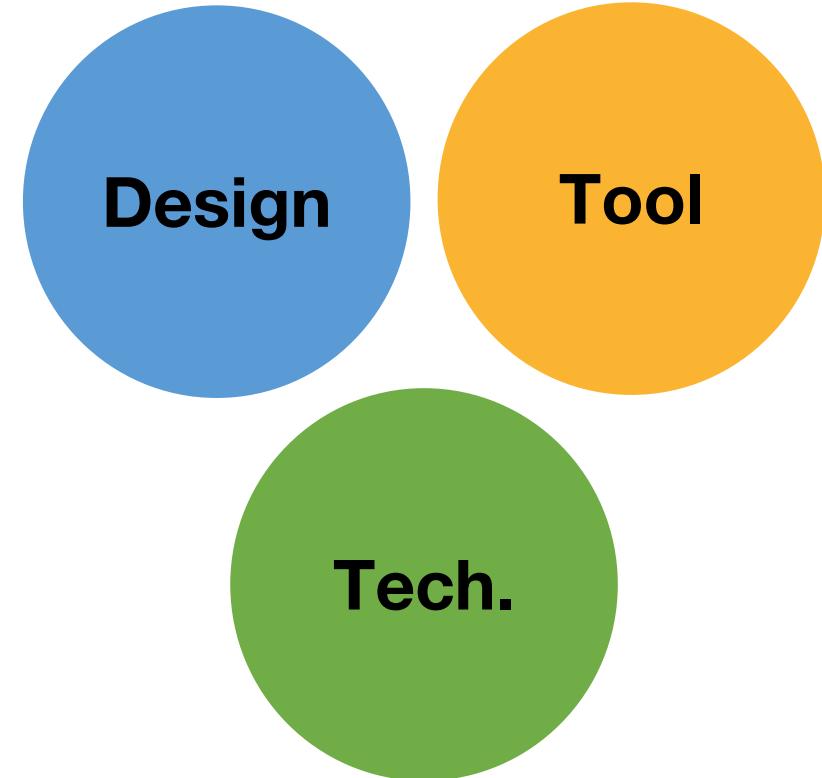


Berkeley Architecture Research



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





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



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!

