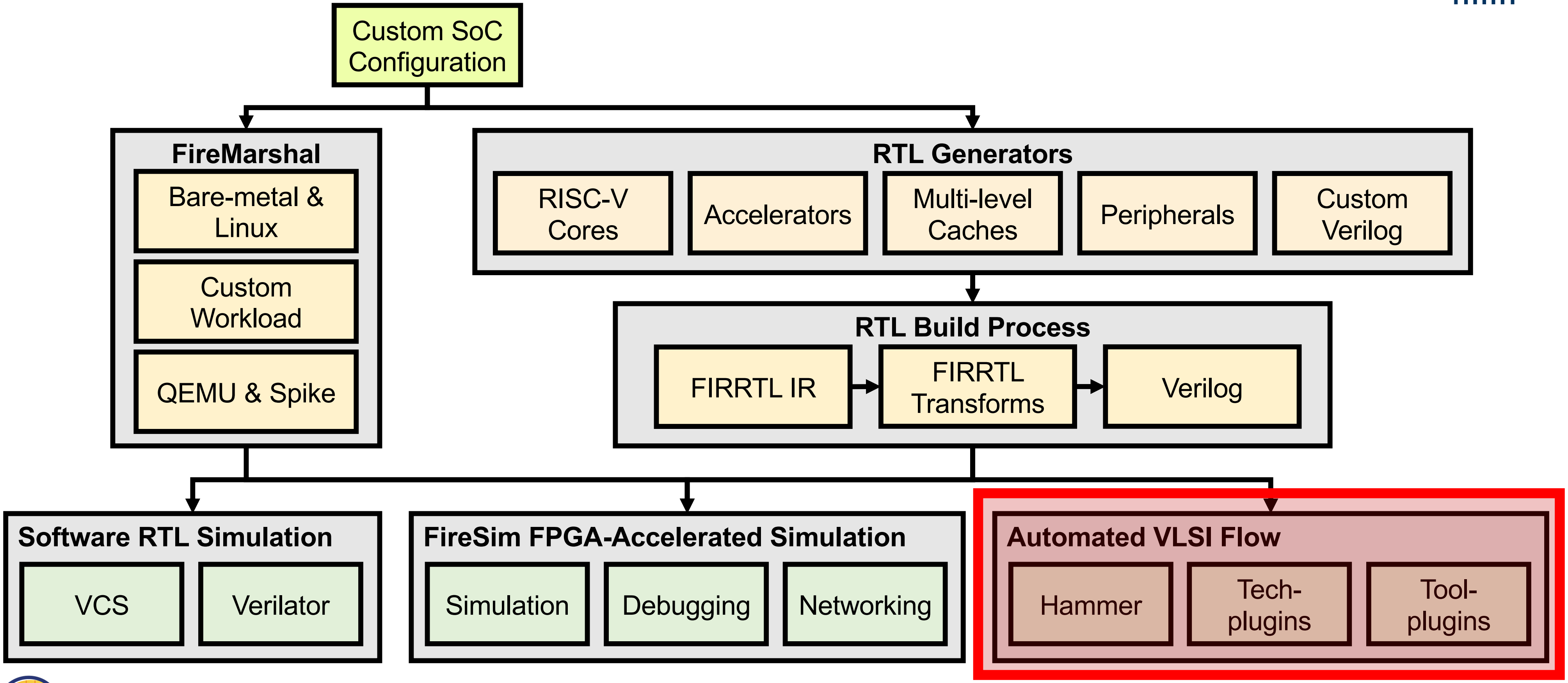
Hammer VLSI Flow

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

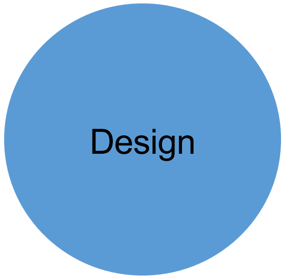
Tool-plugins

Goals

- Demystify the physical design (VLSI) flow
- Overview of Hammer’s abstractions
- Get you started with a TinyRocketConfig in ASAP7
- Under the hood: plugins, hooks, etc.

ASIC Design Environment

ASIC Design Environment


Design

- RTL
  - ARM core, TPU, ...

ASIC Design Environment

Design

- RTL
  - ARM core, TPU, ...

Technology Process

- Transistor process
  - “PDK” or Process Design Kit
  - Intel 7nm, TSMC 5nm, ...

ASIC Design Environment

Design

- RTL
  - ARM core, TPU, ...

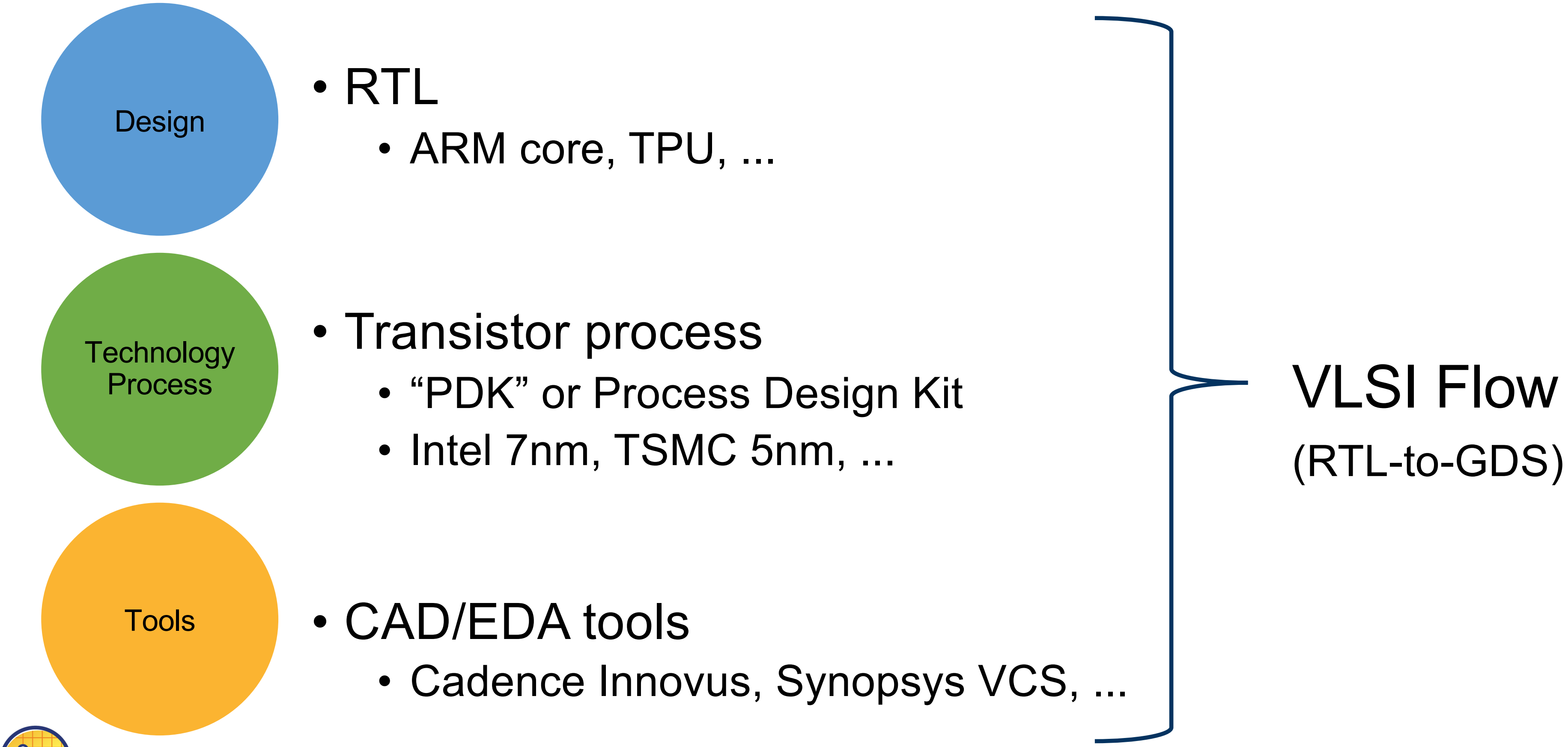
Technology Process

- Transistor process
  - “PDK” or Process Design Kit
  - Intel 7nm, TSMC 5nm, ...

Tools

- CAD/EDA tools
  - Cadence Innovus, Synopsys VCS, ...

ASIC Design Environment



- RTL
  - ARM core, TPU, ...

- Transistor process
  - “PDK” or Process Design Kit
  - Intel 7nm, TSMC 5nm, ...

- CAD/EDA tools
  - Cadence Innovus, Synopsys VCS, ...

VLSI Flow

(RTL-to-GDS)

ASIC Design Environment

Design

- RTL
  - ARM core, TPU, ...

Technology Process

- Transistor process
  - “PDK” or Process Design Kit
  - Intel 7nm, TSMC 5nm, ...

Tools

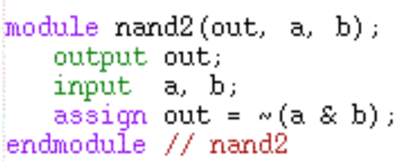
- CAD/EDA tools
  - Cadence Innovus, Synopsys VCS, ...

VLSI Flow

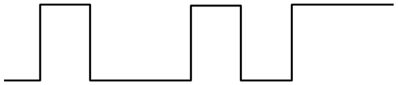
(RTL-to-GDS)

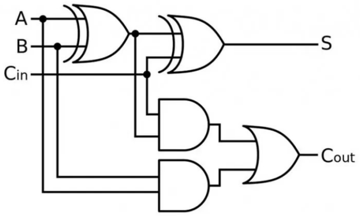
Hammer!

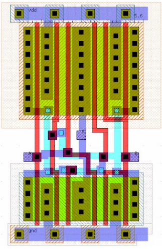
ASIC Flow

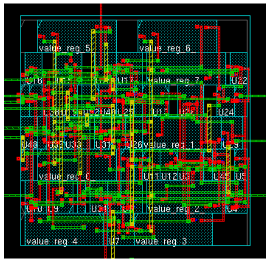


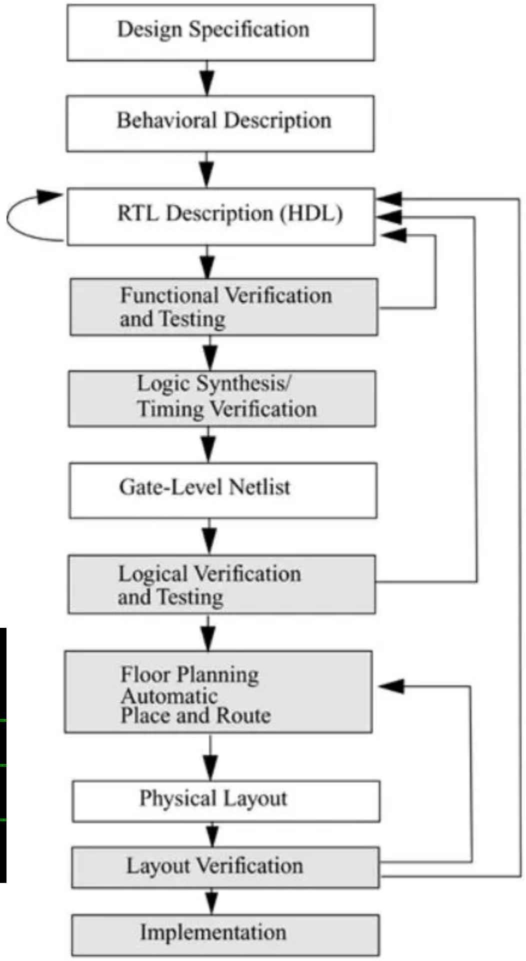
```
module nand2(out, a, b);
   output out;
   input  a, b;
   assign out = ~(a & b);
endmodule // nand2
```











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

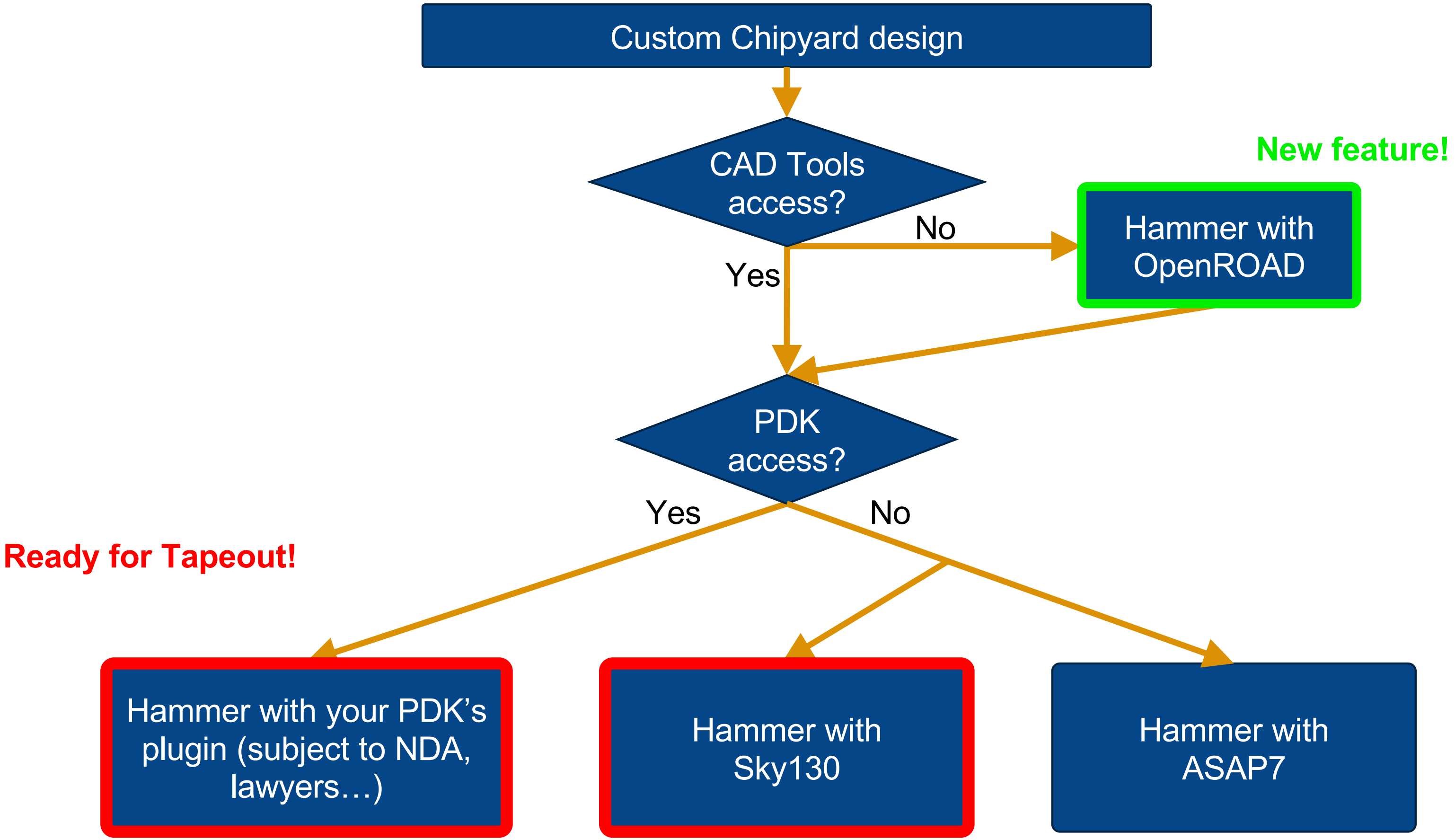
synthesis

place-and-route

DRC

LVS

Motivation: Hammer Decision Tree

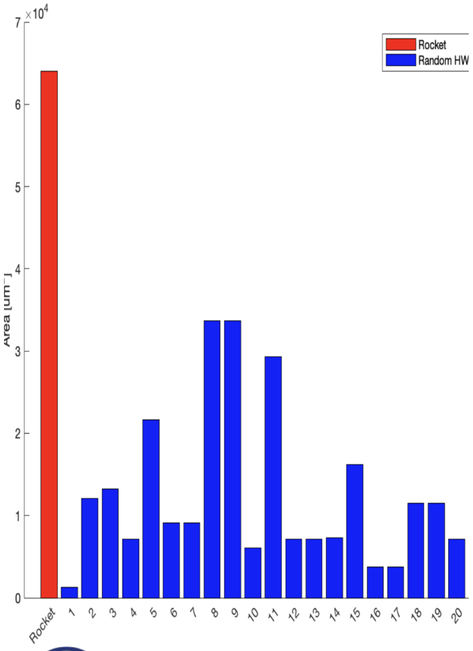


Berkeley Architecture Research

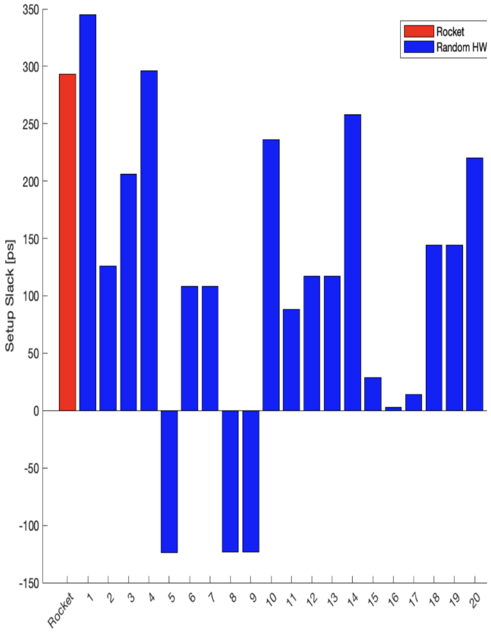
Motivation: Design Space Exploration

post-synthesis results using Hammer

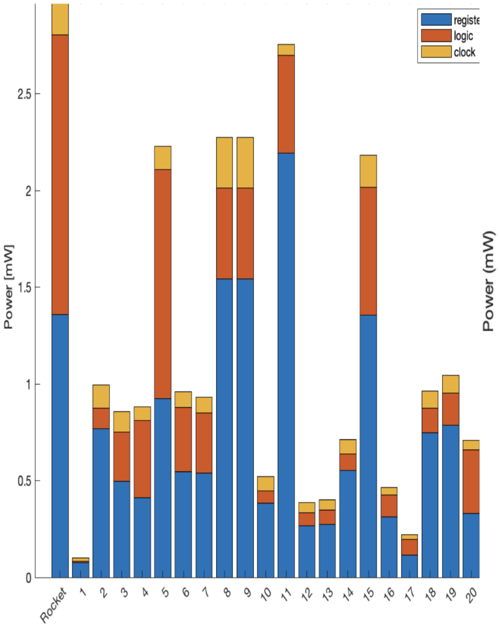
Area



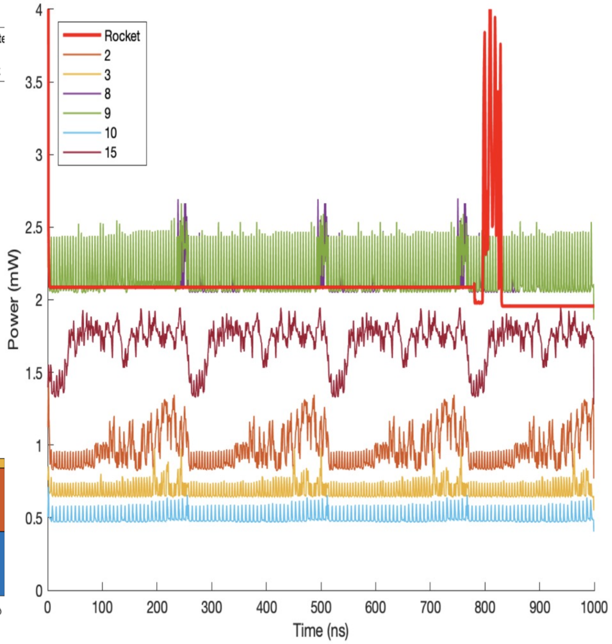
Setup Slack



Avg Power

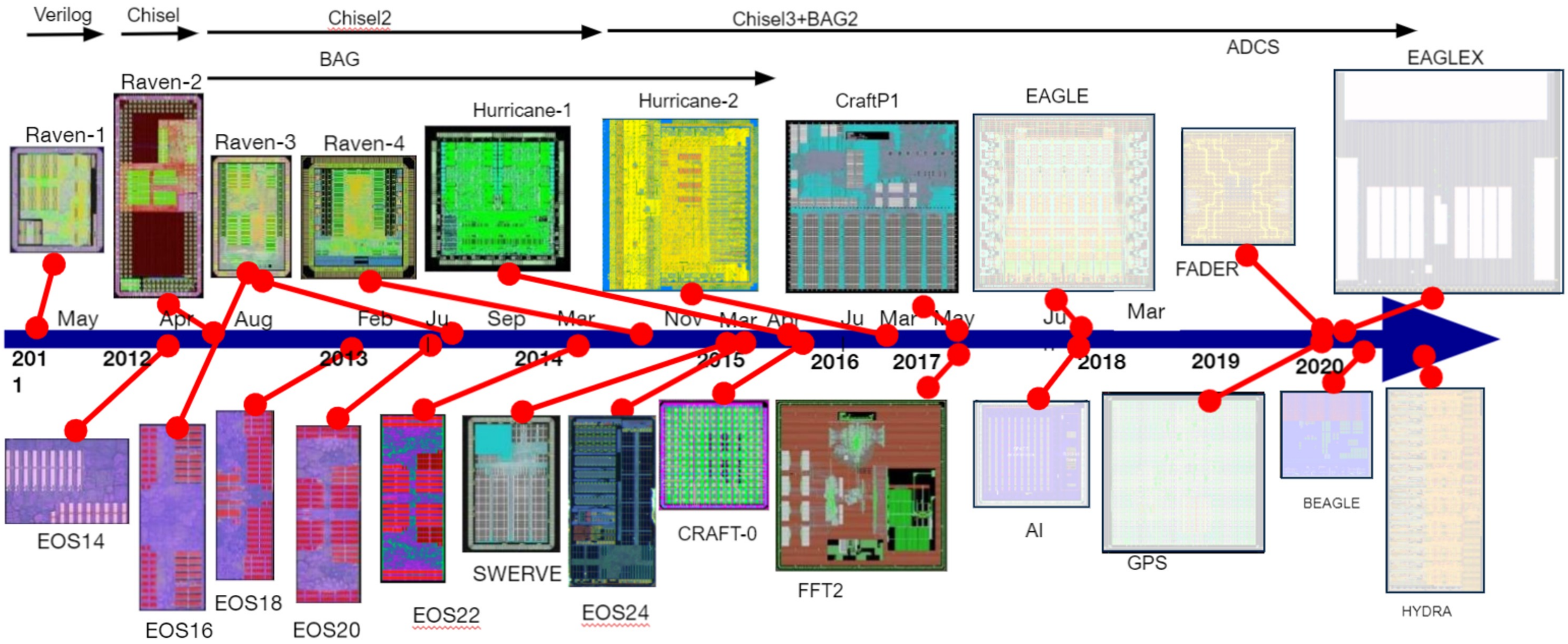


Power Profiles



Berkeley Architecture Research

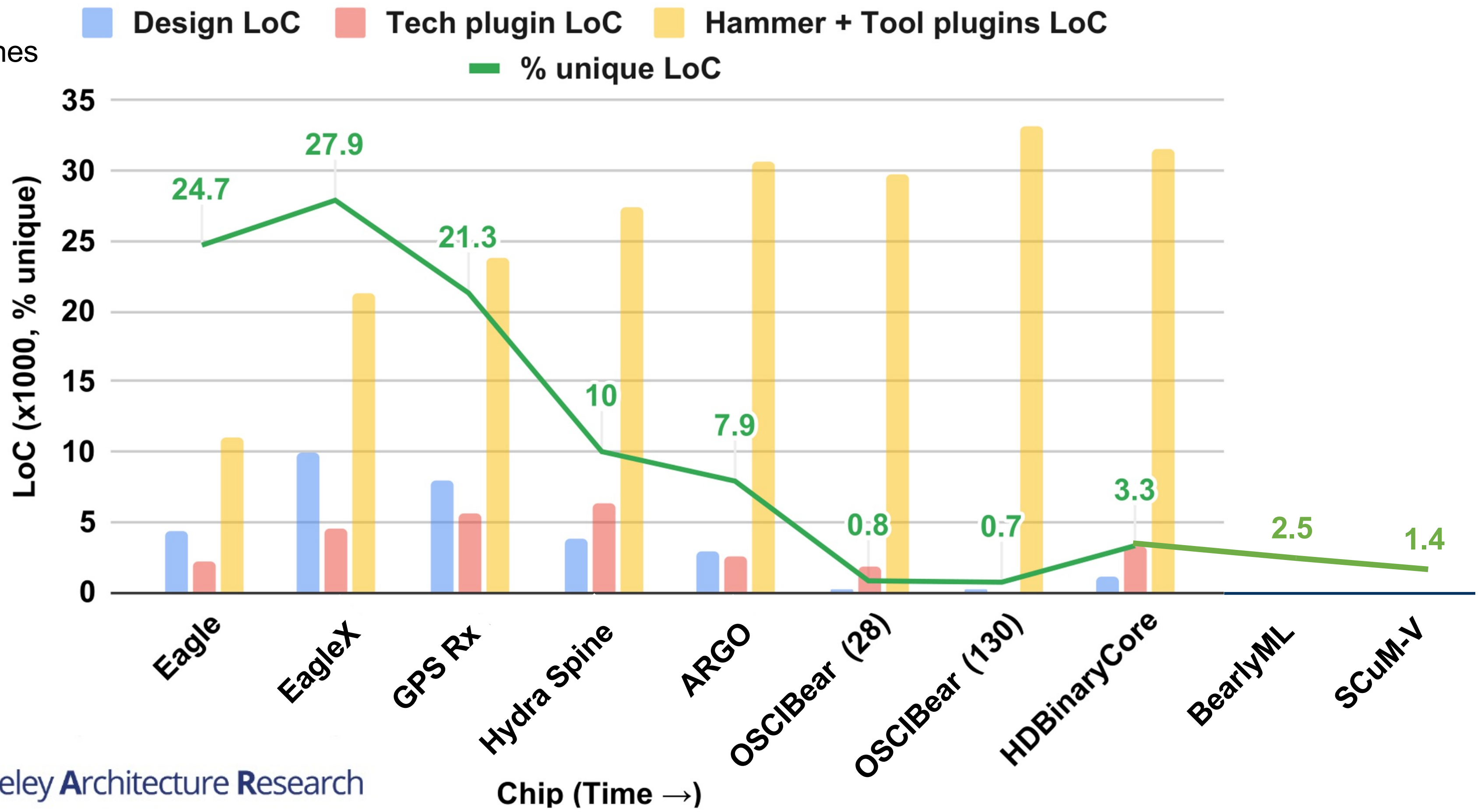
Motivation: Real Tapeouts



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

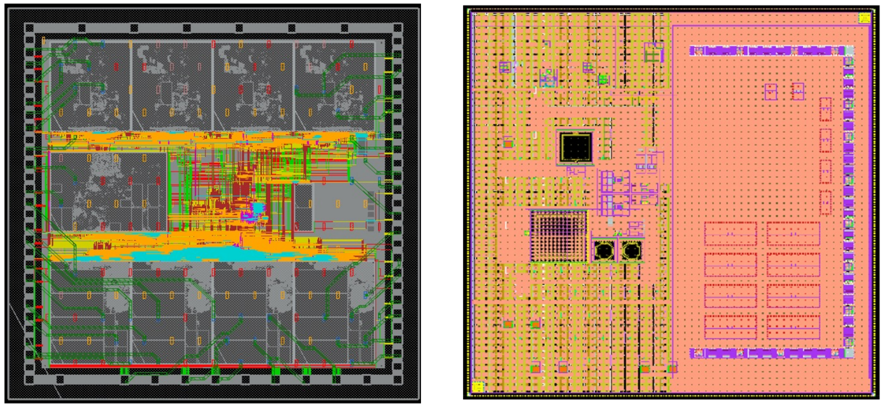
Motivation: Increasing Reusability

LoC = lines of code

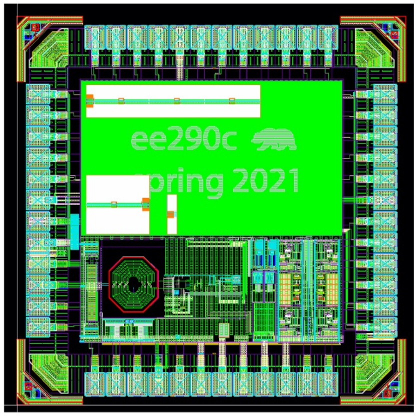


Motivation: Teaching

- ASAP7: EECS151 & EE251B labs/projects
  - 32-bit RISC-V core
  - Digital PLLs, LDOs, PUFs, RNGs
  - https://github.com/EECS150/asic_labs_sp22
  - working on releasing these completely with ASAP7/Sky130 + Cadence tools!
- Intel16: EE194/290C Tapeout (Sp ‘22)
  - Bluetooth SoC
  - Multi-core ML SoC
- TSMC 28HPC: EE194/290C (Sp ‘21)
  - Bluetooth SoC



2022 EE194/290C chips: BearlyML (left) & SCuM-V (right)



2021 EE194/290C chip: Bluetooth, AES, Rocket

How things will work

```
# command 1
> echo “Chipyard Rules!”

# command 2
> do_this arg1 arg2
```

Terminal Section

```
# Technology Setup
# Technology used is ASAP7
vlsi.core.technology: asap7
# Specify dir with ASAP7 tarball
technology.asap7.tarball_dir: ""
```

Inside-a-File Section

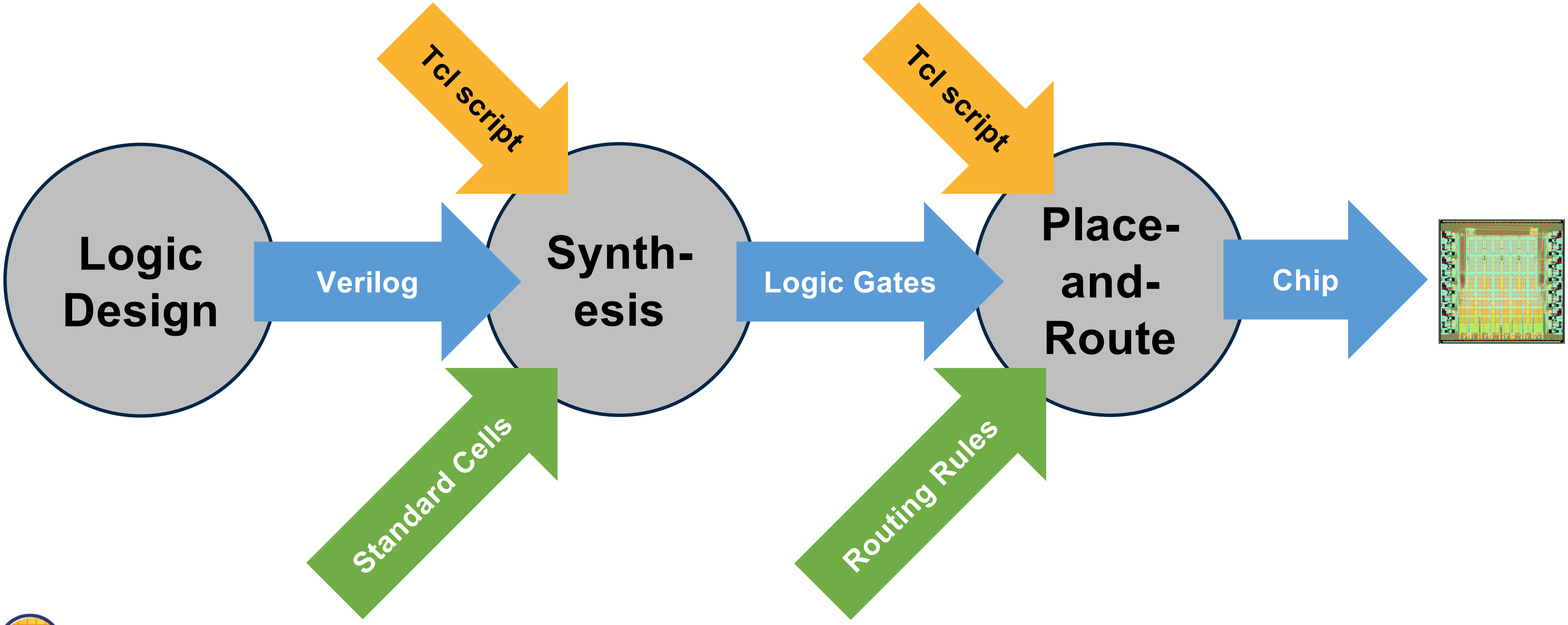
Hammer Features

- PDKs: ASAP7, Skywater 130nm
- Commercial Tools: Cadence [Genus, Innovus, Joules, Voltus], Siemens Calibre, Synopsys [DC, VCS]
- Open-source Tools: Yosys, OpenROAD, Magic, Netgen
- Flow customization with hooks
- SRAM selection
- Floorplanning API
- Power strap generation
- RTL-to-GDS tutorials with Chipyard in ASAP7 and Sky130
- Used in undergrad Digital Design at Berkeley
- Verified with successful tapeouts in TSMC 28, Intel 16, GF 12

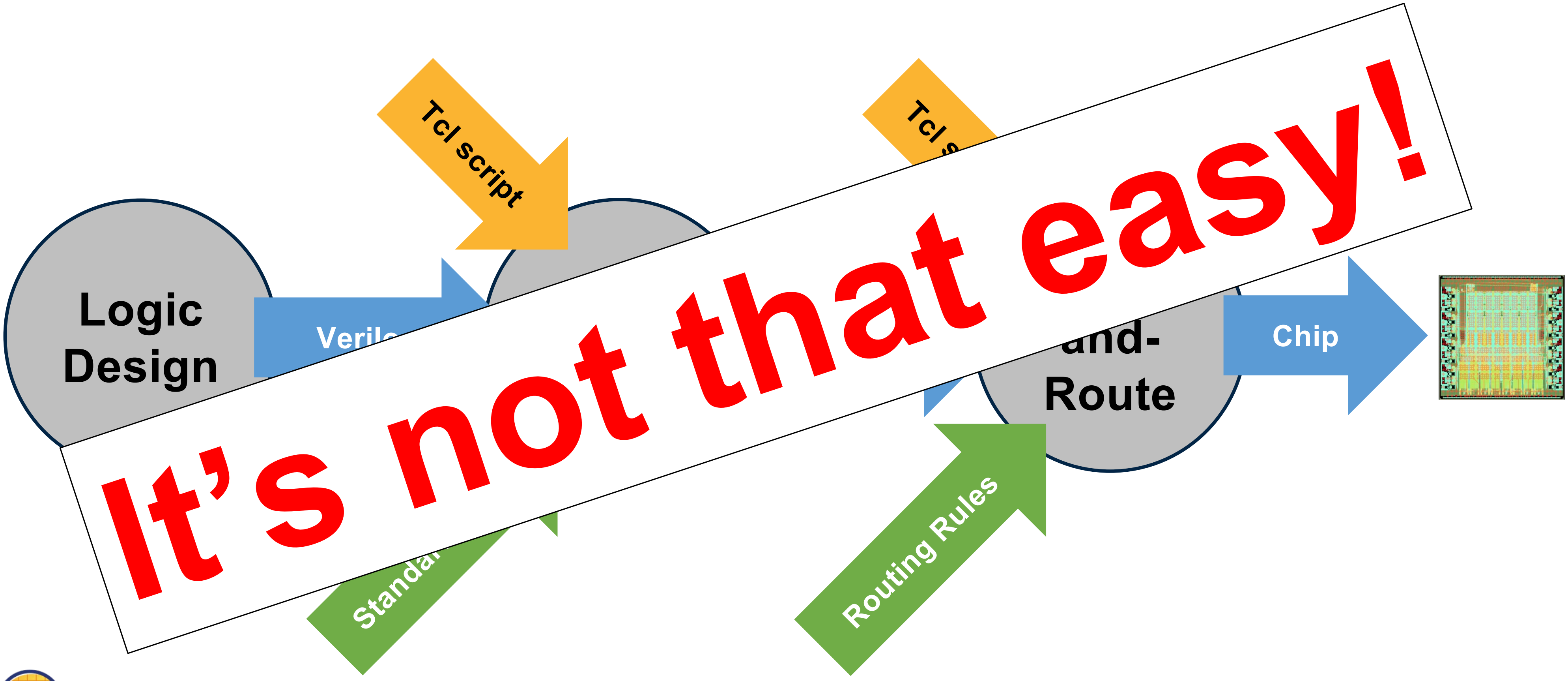
What is Hammer, and why?

Berkeley Architecture Research

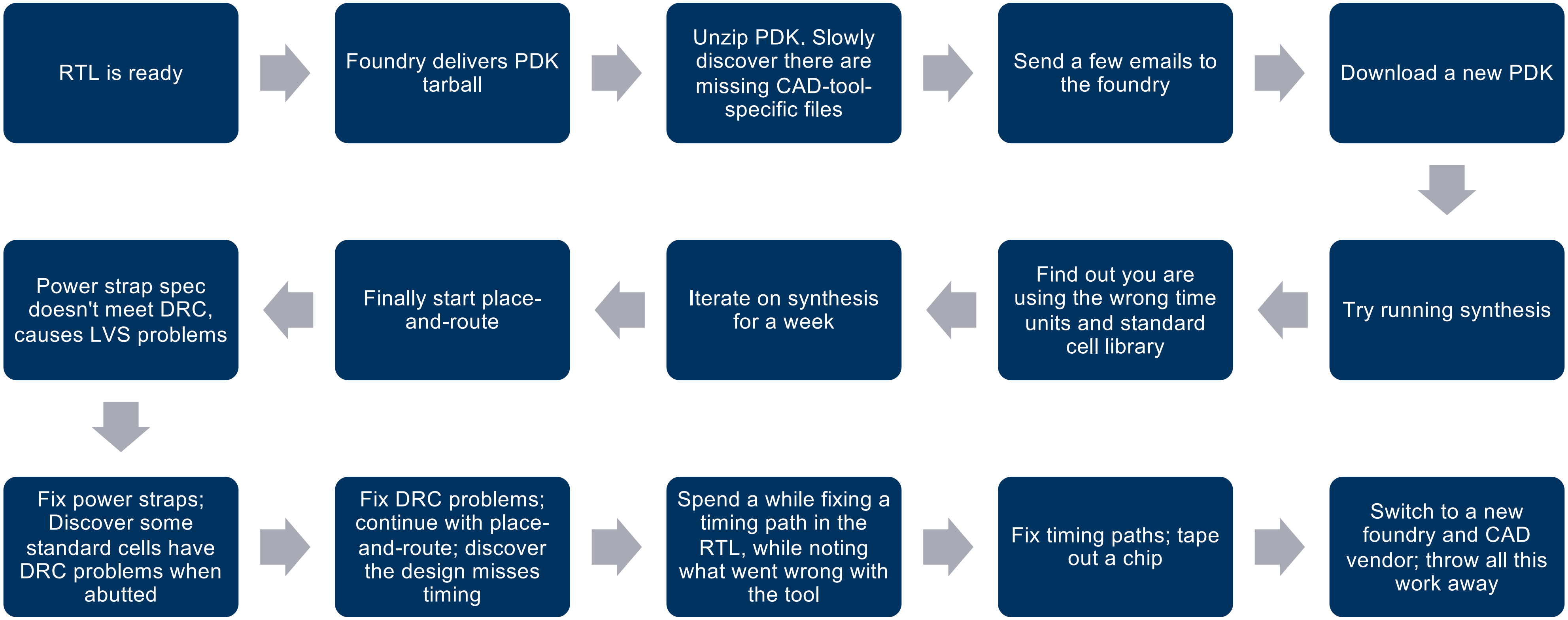
An “Advertised” VLSI Flow



An “Advertised” VLSI Flow



A Real VLSI Flow



RTL is ready

Foundry delivers PDK tarball

Unzip PDK. Slowly discover there are missing CAD-tool-specific files

Send a few emails to the foundry

Download a new PDK

Try running synthesis

Find out you are using the wrong time units and standard cell library

Iterate on synthesis for a week

Finally start place-and-route

Power strap spec doesn't meet DRC, causes LVS problems

Fix power straps; Discover some standard cells have DRC problems when abutted

Fix DRC problems; continue with place-and-route; discover the design misses timing

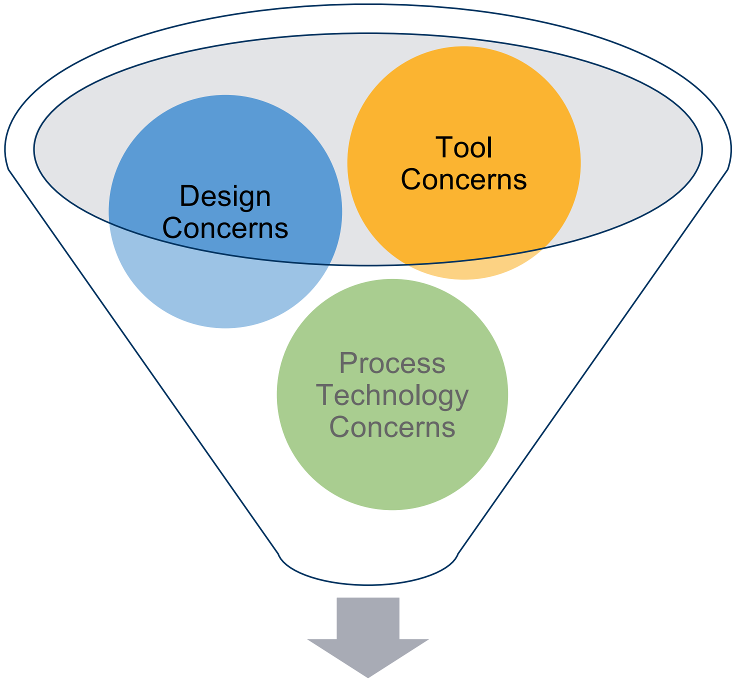
Spend a while fixing a timing path in the RTL, while noting what went wrong with the tool

Fix timing paths; tape out a chip

Switch to a new foundry and CAD vendor; throw all this work away

A Real VLSI Flow

- **Problem**: VLSI flows must be **rebuilt** for each project
- Overhead compounded by
  - Changing CAD tools
    - Commands / features change
    - File formats / library locations
  - New process technology
    - SRAMs (compiled/pre-generated?)
    - DRC rules
  - Different design
    - Floorplanning / power / clock

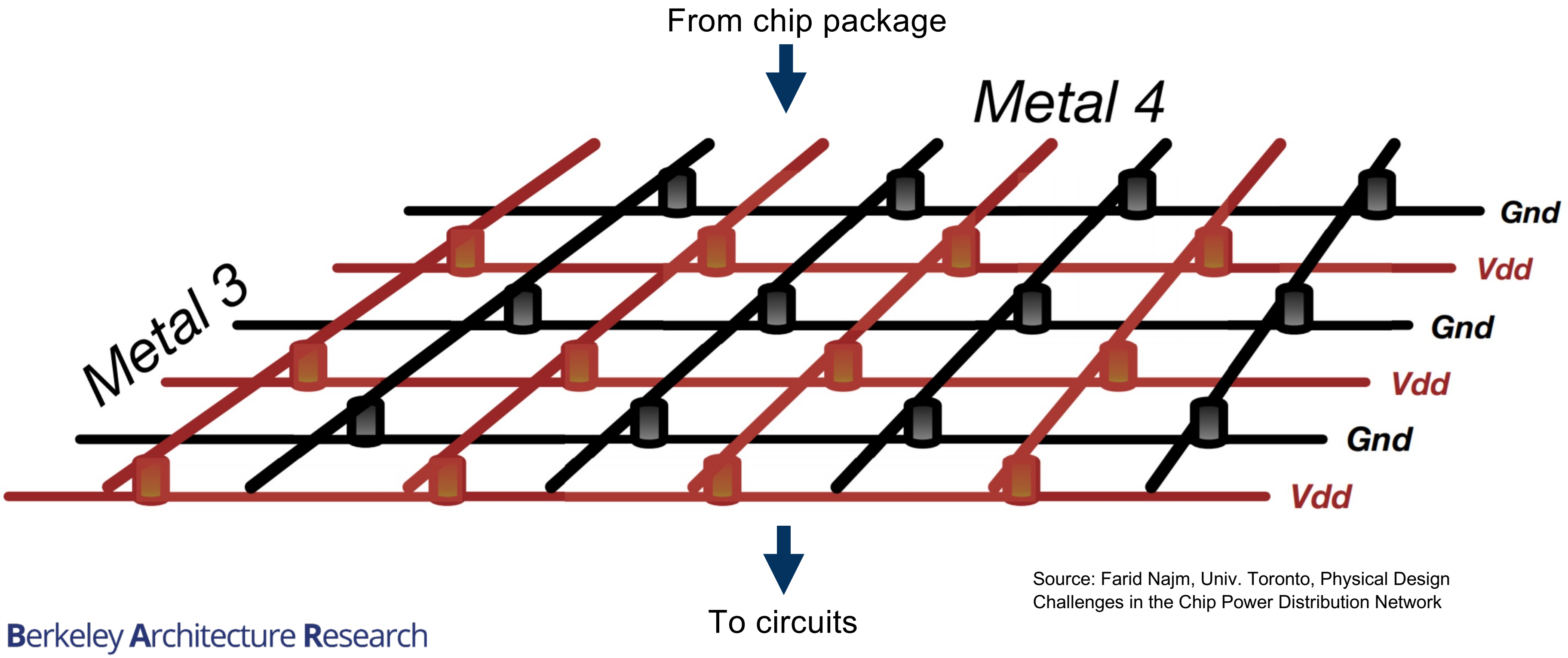


Non-reusable Tcl Script

Berkeley Architecture Research

Example: Power Straps

- All real circuits consume power! How to distribute it?



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

Example

- Consider a 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!
```

- These lines of Tcl for a set of power straps contains:
  - The command itself and its options (tool-specific)
  - DRC-clean spacing, width, and direction information (technology-specific)
  - Group pitch, domain, floorplan information (design-specific)

*Fake commands inspired by real commands due to ULA

Hammer “Separation of Concerns”

- **Solution**: Add a layer of **abstraction**

- Three categories of flow input
  - Design-specific
  - Tool/Vendor-specific
  - Technology-specific

- Hammer principle: specify all three separately
  - Allow reusability
  - Allow for multiple “small” experts instead of a single “super” expert
  - Build abstractions/APIs on top

Design:

- Floorplan
- Clocks
- Hierarchy

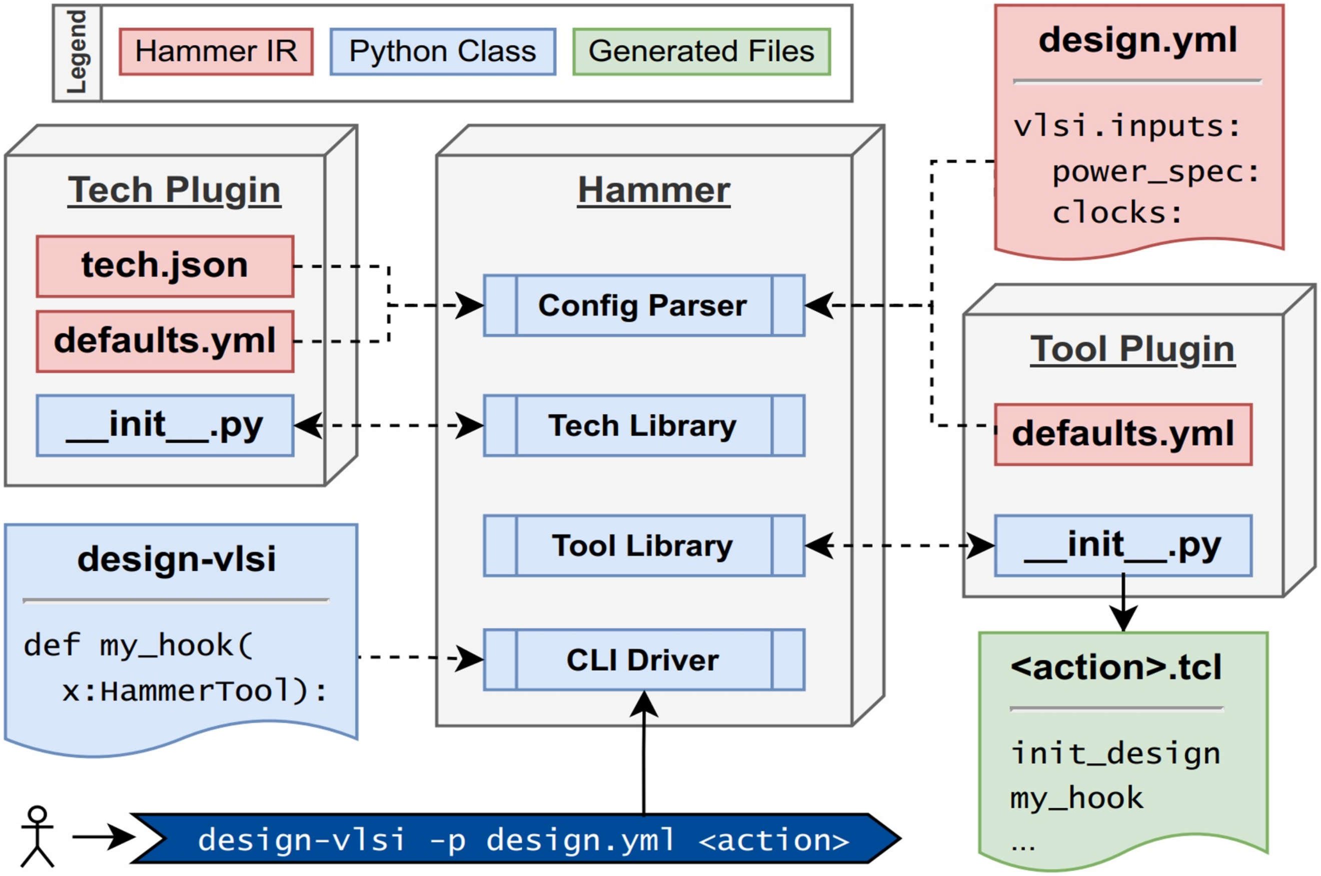
Tool:

- In/out files
- Tcl code
- Tech. file formats

Tech.:

- SRAMs
- Std. cells
- Stack-up
- Power straps

Hammer Software Architecture



Legend: Hammer IR | Python Class | Generated Files

design.yml

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

Tech Plugin

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

Hammer

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

Tool Plugin

- defaults.yml
- __init__.py

design-vlsi

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

<action>.tcl

```
init_design
my_hook
…
```

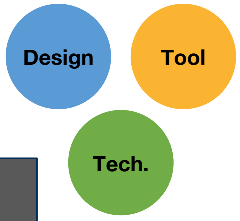
```
design-vlsi -p design.yml <action>
```

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!

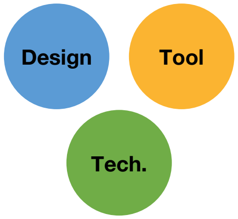
```
# Specify clock signals
vlsi.inputs.clocks: [
  {name: "clock", period: "1ns", uncertainty: "0.1ns"}
]
# Generate Make include to aid in flow
vlsi.core.build_system: make
# Pin placement constraints
vlsi.inputs.pin_mode: generated
vlsi.inputs.pin.generate_mode: semi_auto
vlsi.inputs.pin.assignments: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

Separated Concerns



Power Straps Example

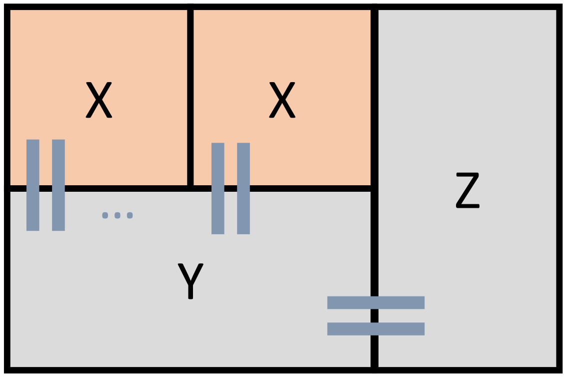
Separated Concerns



- 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



Berkeley Architecture Research

Power Straps Example

Separated Concerns

Design

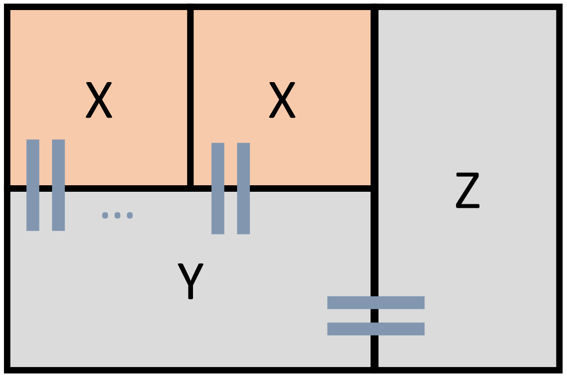
Tool

Tech.

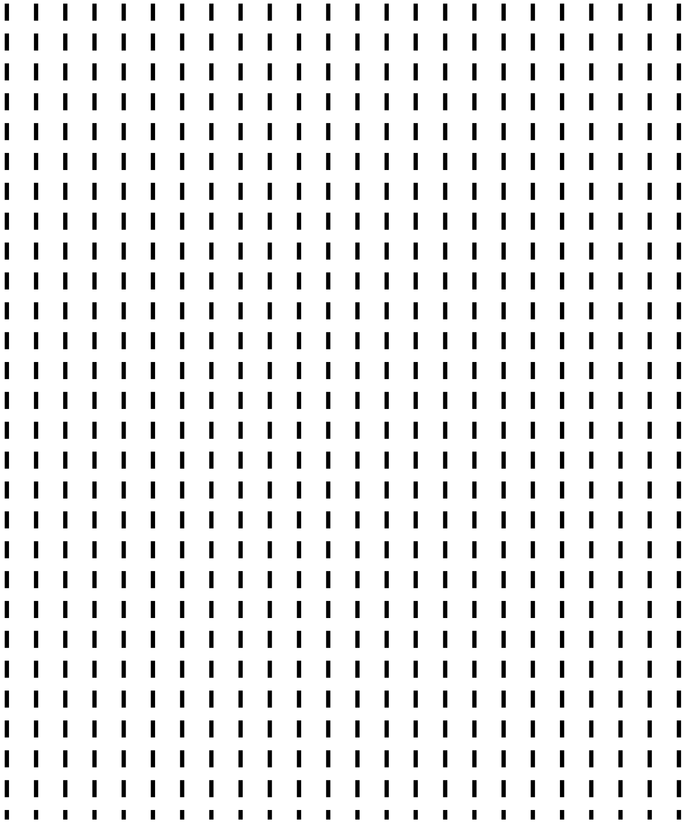
- Don’t make the designer do math!
  - Codify design process in tech- and tool-agnostic code

- 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



Choose power strap strategy

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

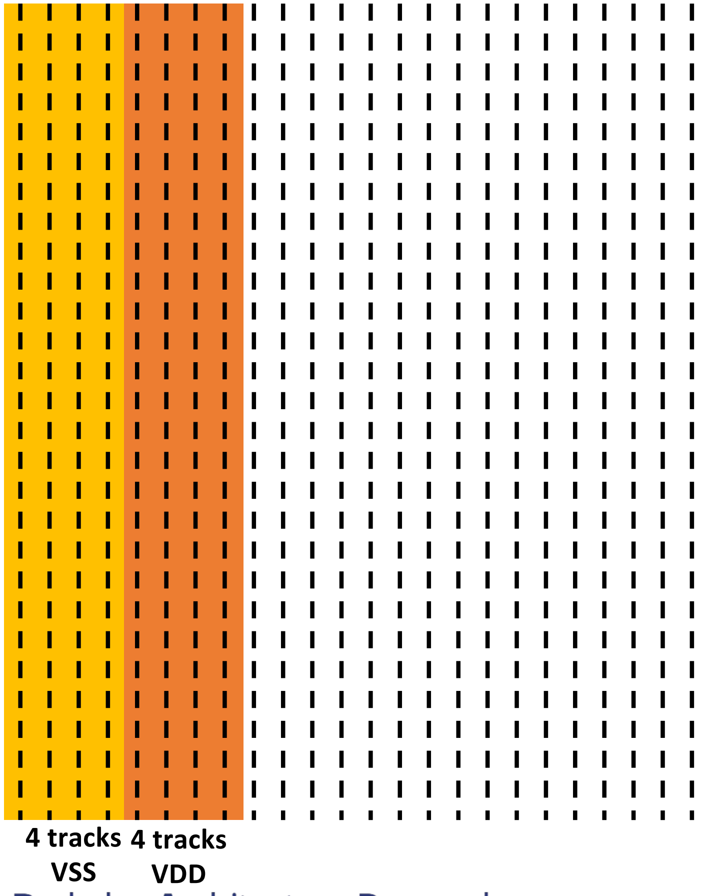
Separated Concerns

Design

Tool

Tech.

Power Straps Example



Allocate tracks

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

Separated Concerns

Design

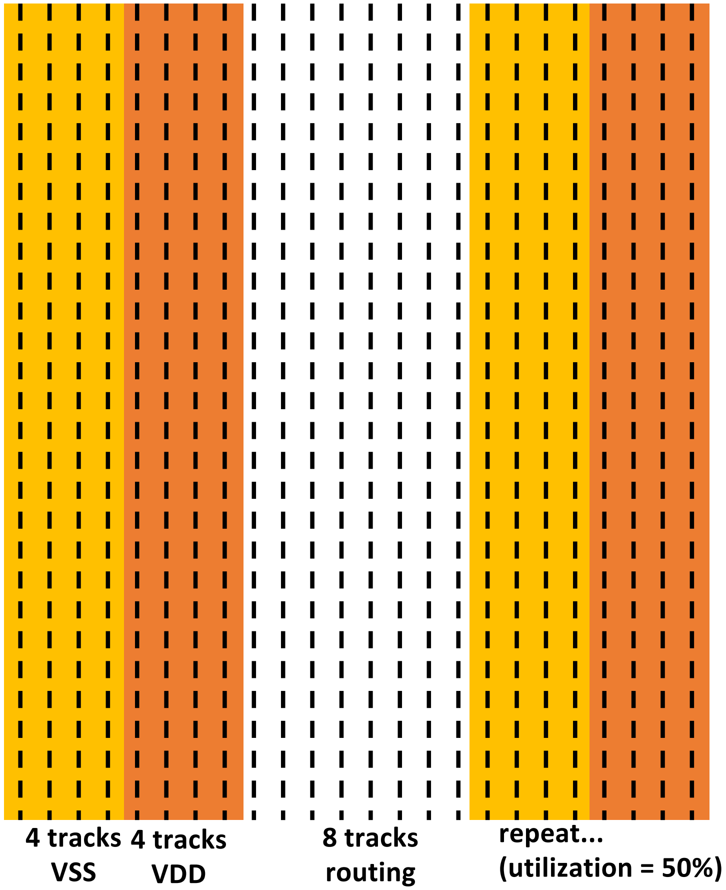
Tool

Tech.

```
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
```

Separated Concerns

Design

Tool

Tech.

Determine pitch

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

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

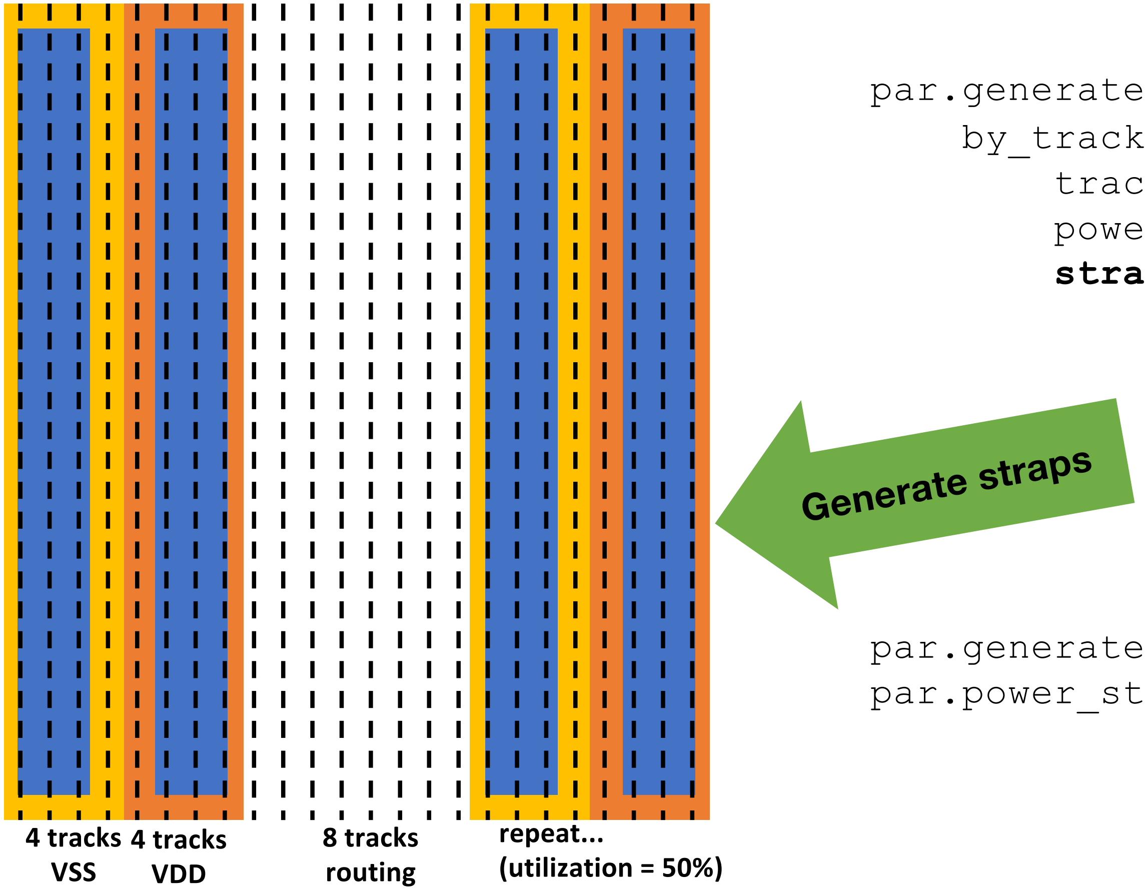
Power Straps Example

Separated Concerns

Design

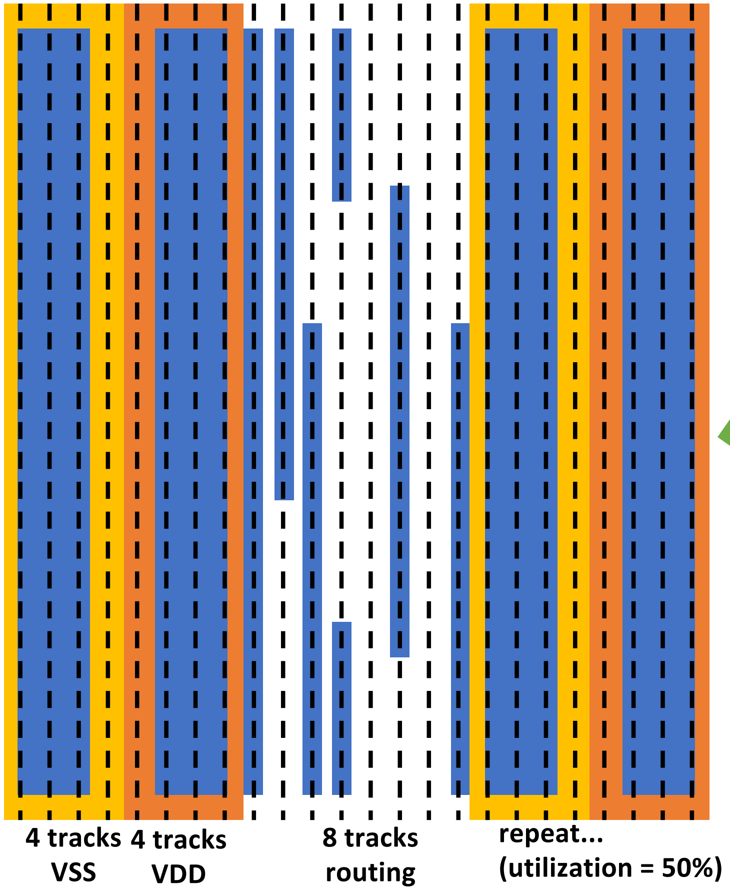
Tool

Tech.



```
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



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
```

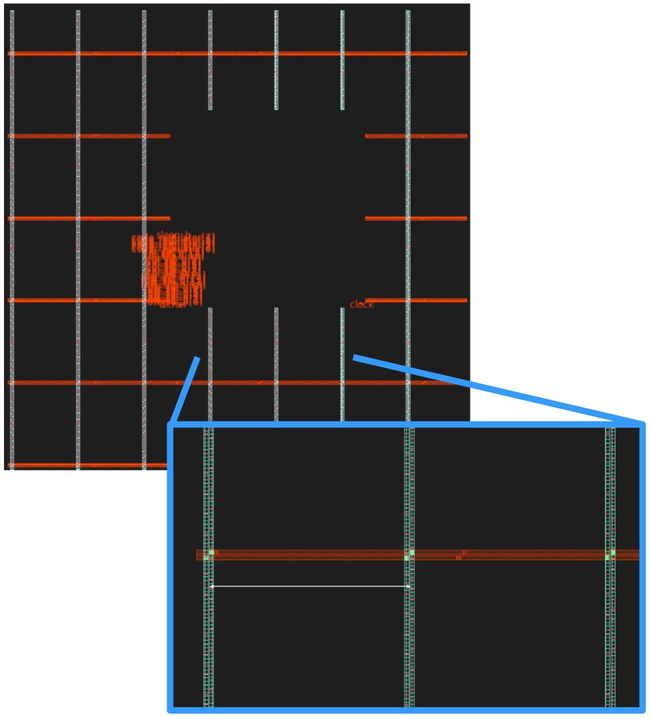
Separated Concerns

Design

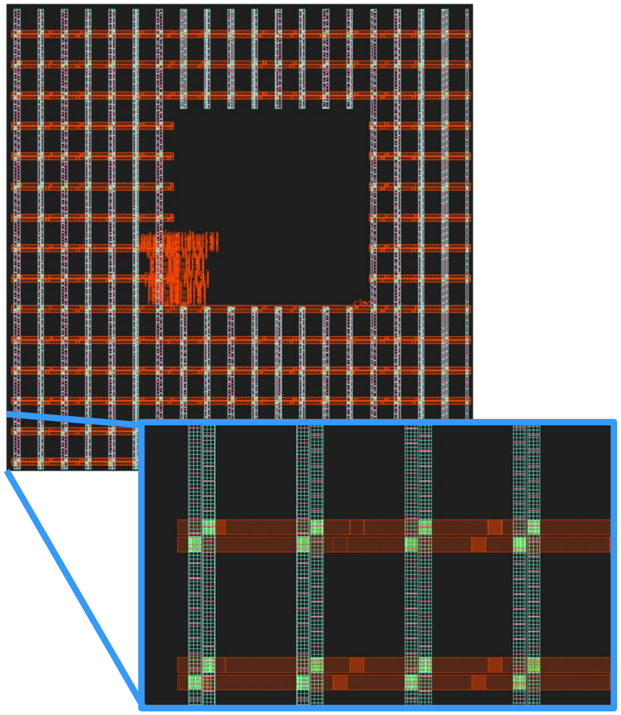
Tool

Tech.

Power Straps Example



`power_utilization: 0.1`



`power_utilization: 0.3`

ASAP7 Example

How to use Hammer in Chipyard

- Hammer integrated under `chipyard/vlsi/`
- Priority: building real chips with proprietary CAD tools
  - We are **currently** working on open-source CAD (see OpenROAD, etc.)
- Many prerequisites
  - Welcome to the world of physical design…
- “Real” technologies need an NDA
  - Some “fake” technologies exist to allow example code sharing

```
chipyard/
  vlsi/
    Makefile
    env.yml
    example-asap7.yml
    example-tools.yml
    example-vlsi
    hammer/
    hammer-cadence-plugins/
    hammer-mentor-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
build.sbt
```

Hammer + ASAP7 Example

- Prerequisites in tutorial README
  - E.g. access to CAD tools, PDK download, etc.
- ASAP7 PDK
  - Predictive 7nm FinFET process developed by Arizona State University + ARM
  - Intended for rapid design space exploration, NOT manufacturable
  - Includes standard cell library, transistor models, extraction/signoff decks
  - Does not include SRAMs, IO cells, etc.
  - Download the PDK and Calibre decks separately
- This demo is NOT interactive (you need CAD tool access!). Intermediate files may be provided for you to examine upon request.

Getting Started: Hammer Setup

- Initialize Hammer plugins

```
> cd chipyard
> ./scripts/init-vlsi.sh asap7
```

Wrapper around `git submodule init` to clone the plugins repositories (do this once)

- Define the Hammer environment into the shell

```
> cd vlsi
> export HAMMER_HOME=$PWD/hammer
> Source $HAMMER_HOME/sourceme.sh
```

Do this each time you use Hammer in a new terminal

Getting Started: Environment

- Fill in your license servers/files and tool environment variables

```
# Base path to where Mentor tools are installed
mentor.mentor_home: ""
# Mentor license server/file
mentor.MGLS_LICENSE_FILE: ""
# Base path to where Cadence tools are installed
cadence.cadence_home: ""
# Cadence license server/file
cadence.CDS_LIC_FILE: ""
# Base path to where Synopsys tools are installed
synopsys.synopsys_home: ""
# Synopsys license server/files
synopsys.SNPSLMD_LICENSE_FILE: ""
synopsys.MGLS_LICENSE_FILE: ""
```

```
chipyard/
  vlsi/
    Makefile
    env.yml
    example-asap7.yml
    example-tools.yml
    example-vlsi
    hammer/
    hammer-cadence-plugins/
    hammer-mentor-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
build.sbt
```

Getting Started: Tools, PDK

- Fill in the path to your PDK + Calibre deck tarball

```
# Technology Setup
# Technology used is ASAP7
vlsi.core.technology: asap7
# Specify dir with ASAP7 Calibre deck tarball
technology.asap7.tarball_dir: "/path/to/asap7"
# Specify PDK and std cell install directories
#technology.asap7.pdk_install_dir: "/path/to/asap7/asap7PDK_r1p7"
#technology.asap7.stdcell_install_dir: "/path/to/asap7/asap7sc7p5t_27"
```

```
chipyard/
  vlsi/
    Makefile
    env.yml
    example-asap7.yml
    example-tools.yml
    example-vlsi
    hammer/
    hammer-cadence-plugins/
    hammer-mentor-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
build.sbt
```

Building the Design

Let’s elaborate a TinyRocketConfig:

```
> make buildfile CONFIG=TinyRocketConfig
```

This does the following:

- Runs the generator for Top and Harness
  - For the Top, MacroCompiler maps memories to ASAP7’s dummy SRAMs

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
```

<long-name> should be chipyard.TestHarness.TinyRocketConfig

This will take a while!

Building the Design

Let’s elaborate a TinyRocketConfig:

```
> make buildfile CONFIG=TinyRocketConfig
```

This does the following:

- Runs the generator for Top and Harness
  - For the Top, MacroCompiler maps memories to ASAP7’s dummy SRAMs
- Hammer generates a set of ExtraLibrarys for each selected SRAM

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
```

The first time Hammer is run with ASAP7, the PDK is hacked into build/<long-name>-ChipTop/tech-asap7-cache

Building the Design

Let’s elaborate a TinyRocketConfig:

```
> make buildfile CONFIG=TinyRocketConfig
```

This does the following:

- Runs the generator for Top and Harness
  - For the Top, MacroCompiler maps memories to ASAP7’s dummy SRAMs
- Hammer generates a set of ExtraLibrarys for each selected SRAM
- Hammer generates a set of Make targets implementing the VLSI flow graph and the input Hammer IR to the flow

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
```

You should reference these targets when/if you want to manually run flow steps

Running the VLSI Flow

Run a simple RTL-level functional simulation:

```
> make sim-rtl CONFIG=TinyRocketConfig \
BINARY=$RISCV/riscv64-unknown-
elf/share/riscv-tests/isa/rv64ui-p-simple
# Do this if the terminal hangs
> reset
```

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      sim-<rtl/par>-rundir/
        simv
      syn-rundir/
      par-rundir/
      power-par-rundir/
```

Berkeley Architecture Research

Running the VLSI Flow

Run a simple RTL-level functional simulation:

```
> make sim-rtl CONFIG=TinyRocketConfig \
BINARY=$RISCV/riscv64-unknown-elf/share/riscv-tests/isa/rv64ui-p-simple
```

Run Genus synthesis and Innovus P&R:

```
> make syn CONFIG=TinyRocketConfig
> make par CONFIG=TinyRocketConfig
```

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      sim-<rtl/par>-rundir/
        simv
      syn-rundir/
      par-rundir/
      power-par-rundir/
```

Result netlists, GDS, etc. appear in here

This will take an hour or so!
Lots of tool log output too…

Running the VLSI Flow

Run a simple RTL-level functional simulation:

```
> make sim-rtl CONFIG=TinyRocketConfig \
BINARY=$RISCV/riscv64-unknown-
elf/share/riscv-tests/isa/rv64ui-p-simple
```

Run Genus synthesis and Innovus P&R:

```
> make syn CONFIG=TinyRocketConfig
> make par CONFIG=TinyRocketConfig
```

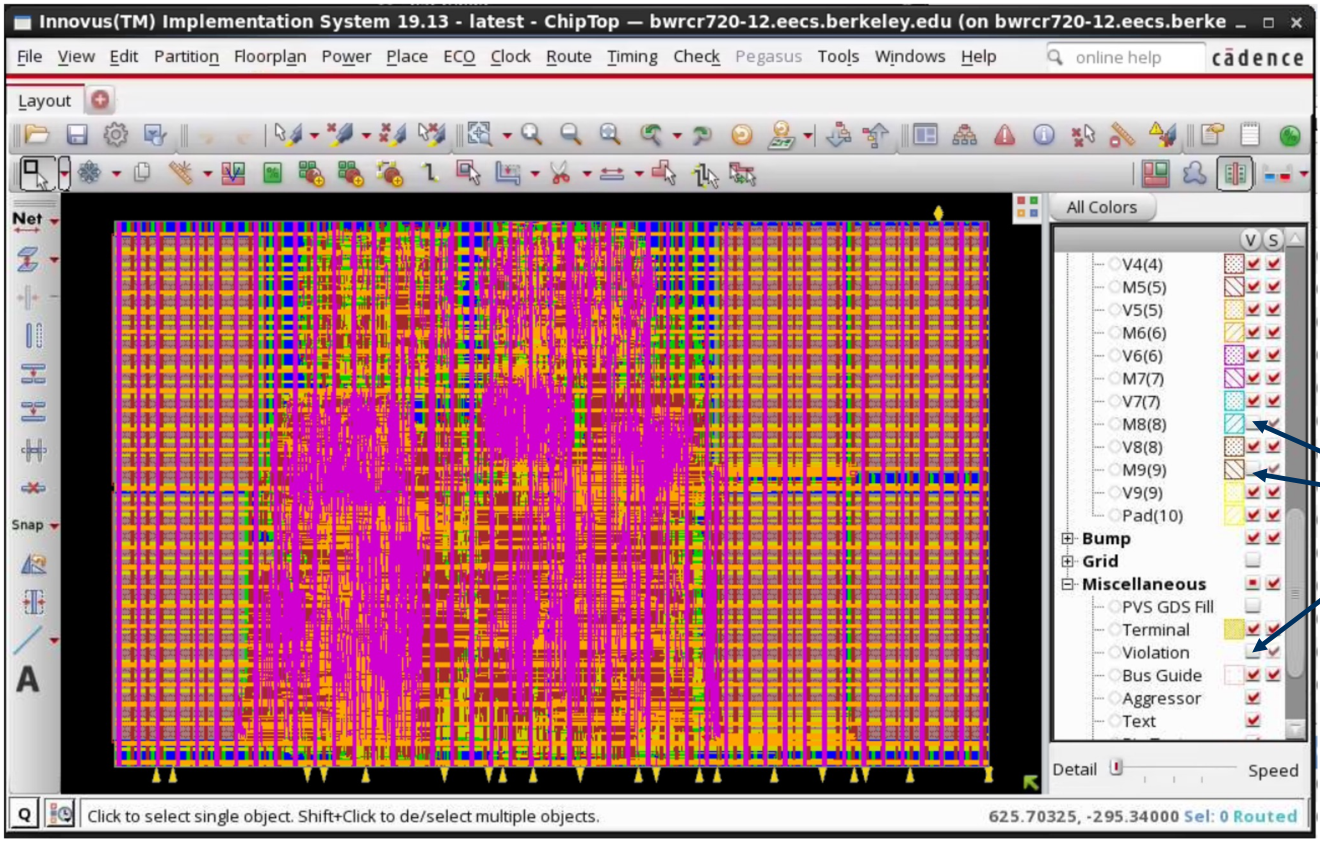
(Optional) View the final Innovus database:

```
> cd build/<long-name>-ChipTop/par-rundir
> ./generated-scripts/open_chip
```

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      sim-<rtl/par>-rundir/
        simv
      syn-rundir/
      par-rundir/
      power-par-rundir/
```

Innovus experience recommended!

Running the VLSI Flow



Innovus(TM) Implementation System 19.13 - latest - ChipTop — bwrcr720-12.eecs.berkeley.edu (on bwrcr720-12.eecs.berke

Click to select single object. Shift+Click to de/select multiple objects.

625.70325, -295.34000 Sel: 0 Routed

You can get this picture by deselecting M8, M9, and violations

Post-P&R Analysis

Do a simple post-P&R sim + power/rail analysis:

```
# ASAP7 gate-level sim fails, force timeout
> make sim-par CONFIG=TinyRocketConfig
BINARY=$RISCV/riscv64-unknown-
elf/share/riscv-tests/isa/rv64ui-p-simple
timeout_cycles=1000
# Do this if terminal hangs
> reset
# Run Voltus power/rail analysis
> make power-par CONFIG=TinyRocketConfig
```

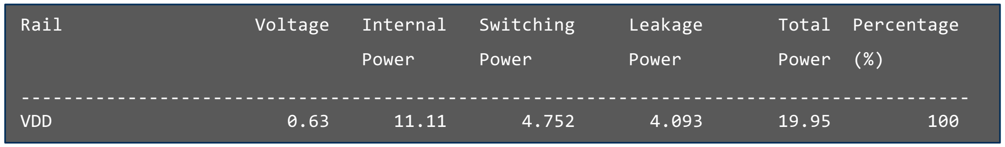
```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      sim-<rtl/par>-rundir/
        simv
      syn-rundir/
      par-rundir/
      power-par-rundir/
```

With a real PDK, you should be able to successfully run gate-level sim and get true waveform-based power numbers

Berkeley Architecture Research

Post-P&R Analysis

```
chipyard/
  vlsi/
    build/<long-name>-ChipTop/
      power-par-rundir/
        staticPowerReports.<corner>/
          power.rpt
        activePowerReports.<corner>/
          power.rpt
        staticRailReports/
        activeRailReports/
```

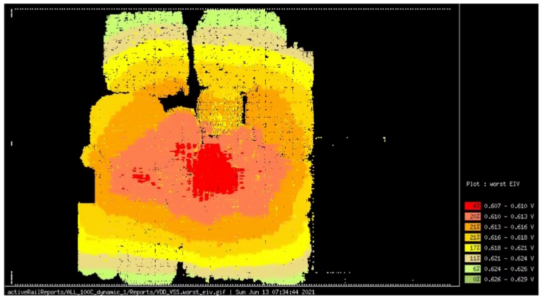


```
Rail                     Voltage   Internal   Switching     Leakage      Total   Percentage
                                   Power      Power         Power        Power   (%)
-----------------------------------------------------------------------------------------
VDD                         0.63      11.11       4.752       4.093      19.95          100
```

Static power analysis results, per corner

Vectorless dynamic power analysis results, per corner

Corresponding rail analysis results, sub-directories created per corner + run



Signoff

Run design rule checking (DRC) and layout-versus-schematic (LVS):

```
> make drc CONFIG=TinyRocketConfig
> make lvs CONFIG=TinyRocketConfig
```

(Optional) View DRC/LVS results in Calibre:

```
> cd build/<long-name>.ChipTop/drc-rundir
> ./generated_scripts/view_drc
> cd build/<long-name>.ChipTop/lvs-rundir
> ./generated_scripts/view_lvs
```

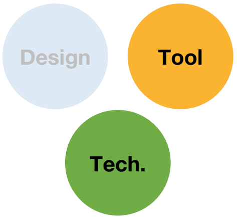
```
chipyard/
  vlsi/
    build/<long-name>.ChipTop/
      drc-rundir/
        drc_results.rpt
      lvs-rundir/
        lvs_results.rpt
```

ASAP7 does not pass DRC and often fails LVS checking. Calibre experience recommended.

Under the Hood

Under the Hood: Plugins

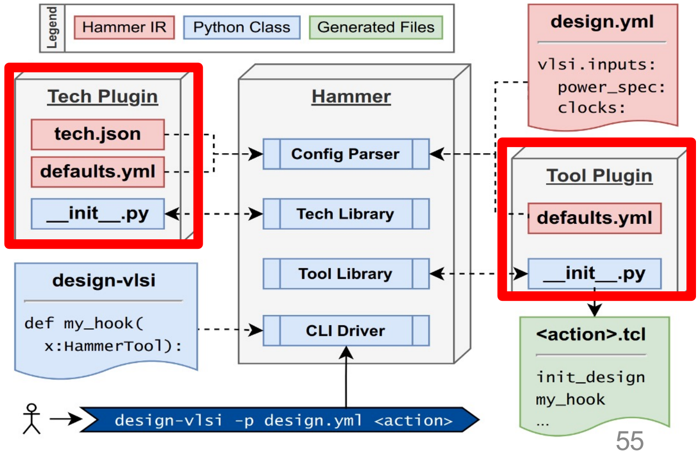
Separated Concerns



- Two types of plugin: Tool and Technology
  - `<tool>` is usually of the format `<action>/<name>`
  - e.g. `par/innovus` or `syn/dc`

- Tool plugins contain:
  - `<tool>/defaults.yml` – overridable default settings for the tool
  - `<tool>/__init__.py` – Reusable python methods, implement Hammer APIs

- Technology plugins contain:
  - `<name>.tech.json` – pointers to relevant PDK files
  - `defaults.yml` – overridable default settings
  - `<name>/<tool>/__init__.py` – Reusable python methods



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

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

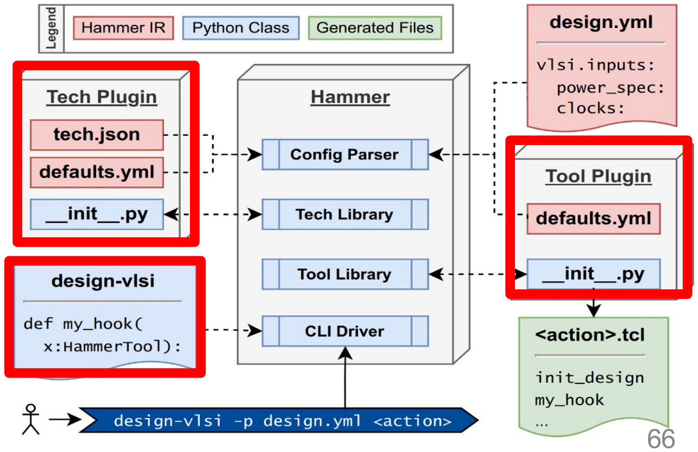
```
init_design
my_hook
...
```

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

Berkeley Architecture Research

Under the Hood: “Hooks”

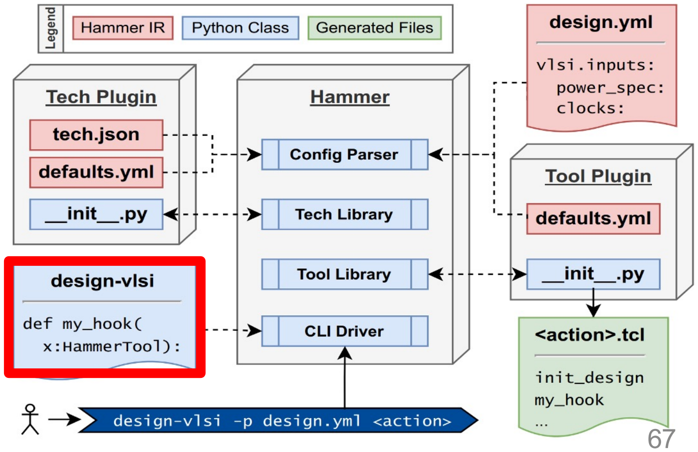
- The “Magic Tcl scripts” aren’t going away soon
  - Foundry reference flow, previous tapeouts
  - A lot of expertise captured in these scripts
- Hooks enable insertion of custom Python and Tcl scripts within the Hammer-generated flow
  - Quick-and-dirty
  - Cleanly allows “hacks” and workarounds
  - Allows prototyping of future APIs
  - Upstreamed hooks available for future re-use



“Hooks” – example-vlsi

- Example placeholders for hooks
  - For: setting non-default tool settings, custom fiducial/endcap placement, etc.
  - Can also be defined by technology plugins. ASAP7 example:
    - `asap7_scale_gds_script` method in ASAP7’s `__init__.py`
    - Inserted as a hook after `write_design`. Runs a Python script (`scale_gds.py`)

```
def example_tool_settings(x: hammer_vlsi.HammerTool) -> bool:
    if x.get_setting("vlsi.core.technology") == "asap7":
        x.append('''
# Place custom TCL here
set_db route_design_bottom_routing_layer 2
set_db route_design_top_routing_layer 7
''')
    return True
```



Berkeley Architecture Research

Everyone can use Hammer

How: provide sensible defaults with methods to override

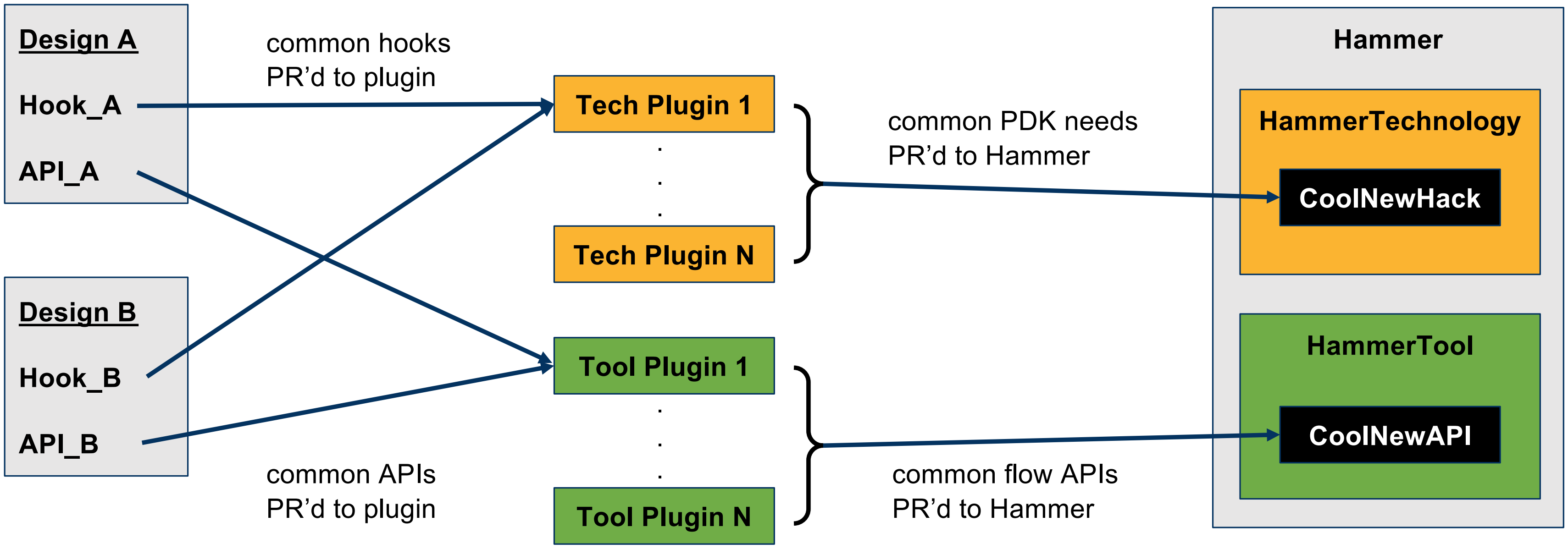
Sensible default	Override method
A default set of flow steps for every action (syn, par, etc.)	Hooks - inject your own steps anywhere
Auto-generated timing (SDC) & power (CPF) constraints	Use your own custom SDC and CPF files
Auto-generated power meshes from high-level parameters	Use foundry-provided 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

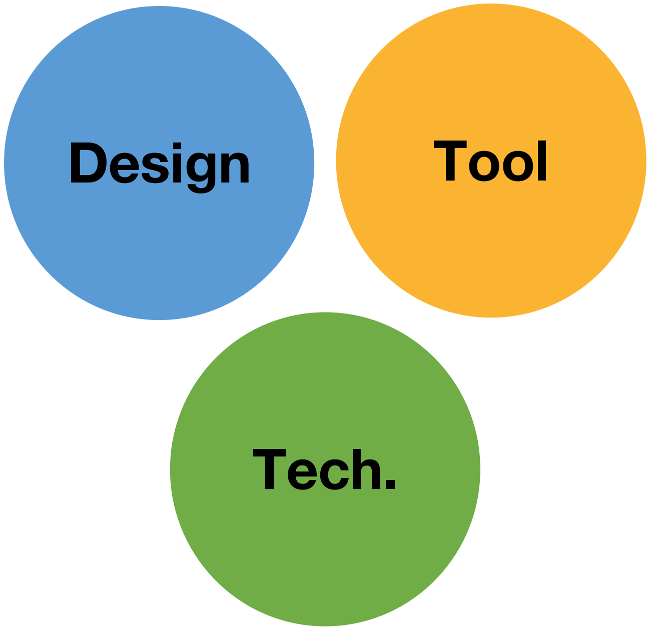
Modular design + override features = community contributions



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



Design

Tool

Tech.

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
- 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!

Coming up…

FPGA Prototyping