# Hammer VLSI Flow

Nayiri Krzysztofowicz

UC Berkeley

nayiri@berkeley.edu
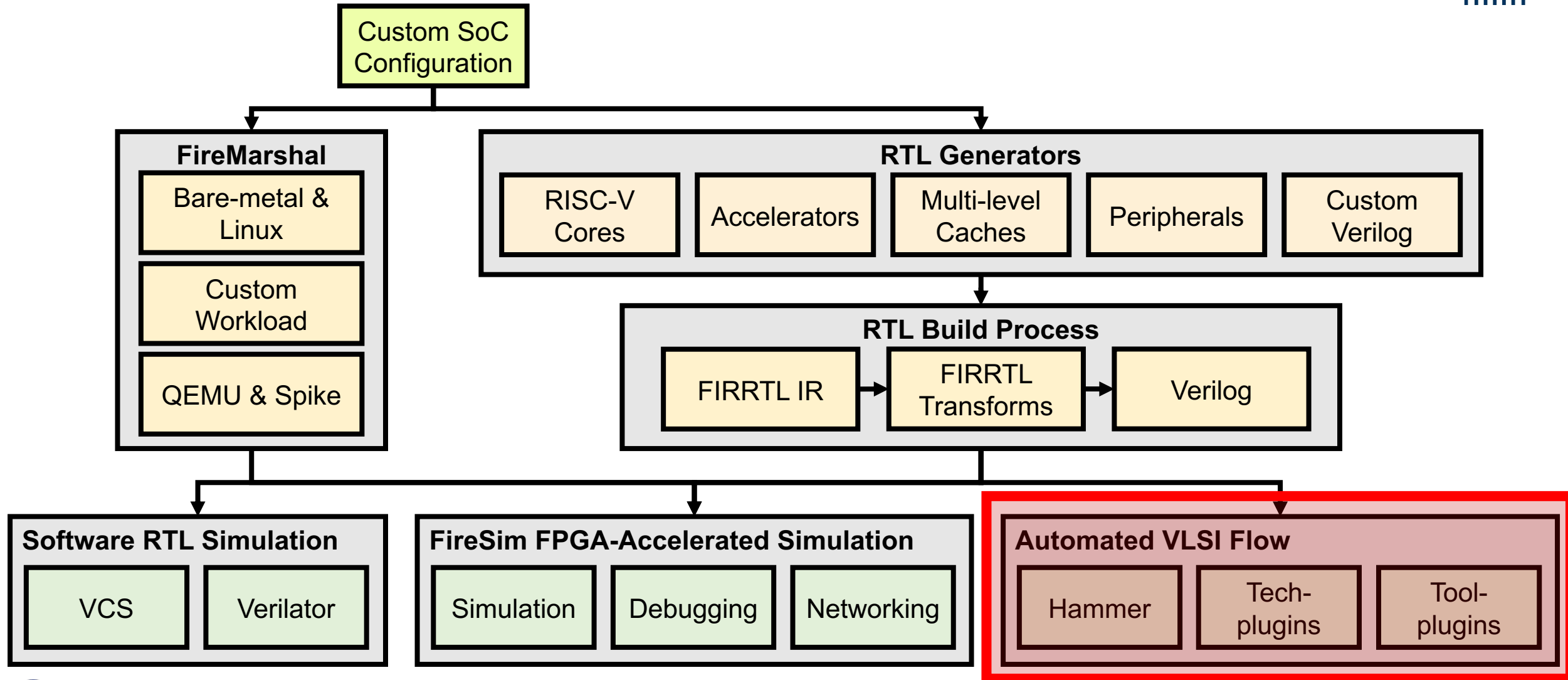
Slides adapted from Harrison Liew
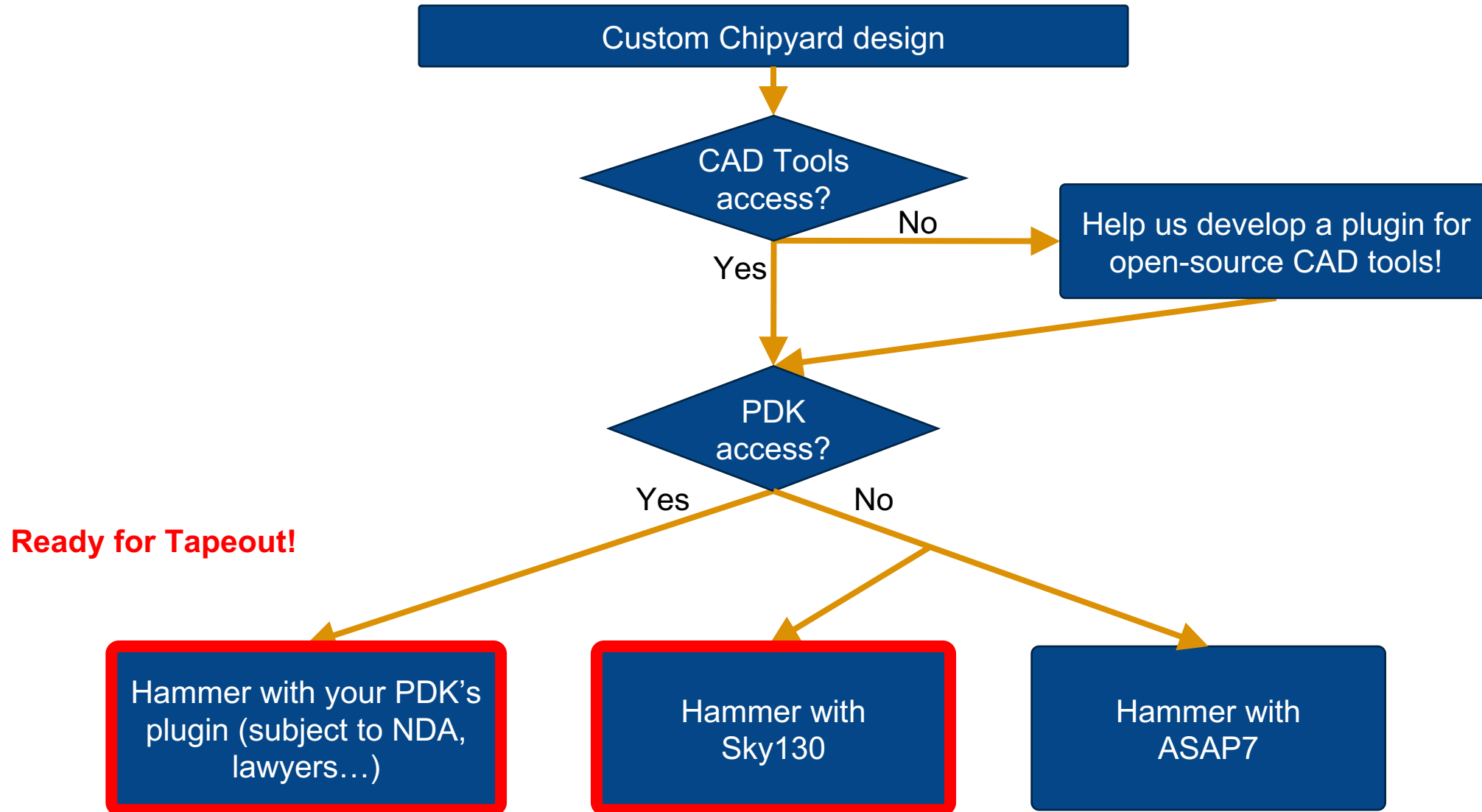
Berkeley Architecture Research

CHIPYARD

# Tutorial Roadmap

# Motivation: Hammer Decision Tree



**Custom Chipyard design**

**CAD Tools access?**
- No → **Help us develop a plugin for open-source CAD tools!**
- Yes →

**PDK access?**
- Yes → **Ready for Tapeout!** → **Hammer with your PDK's plugin (subject to NDA, lawyers…)**
- No → **Hammer with Sky130** / **Hammer with ASAP7**

Berkeley Architecture Research
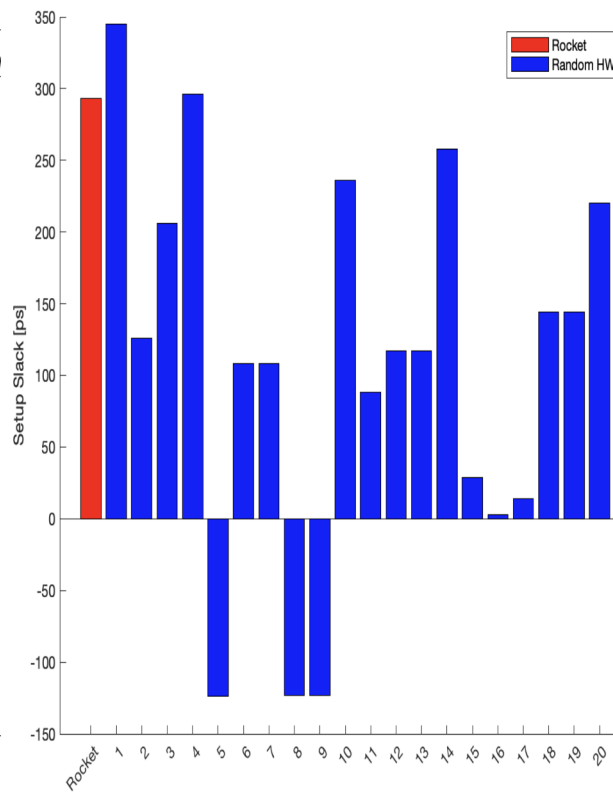
# Motivation: Design Space Exploration
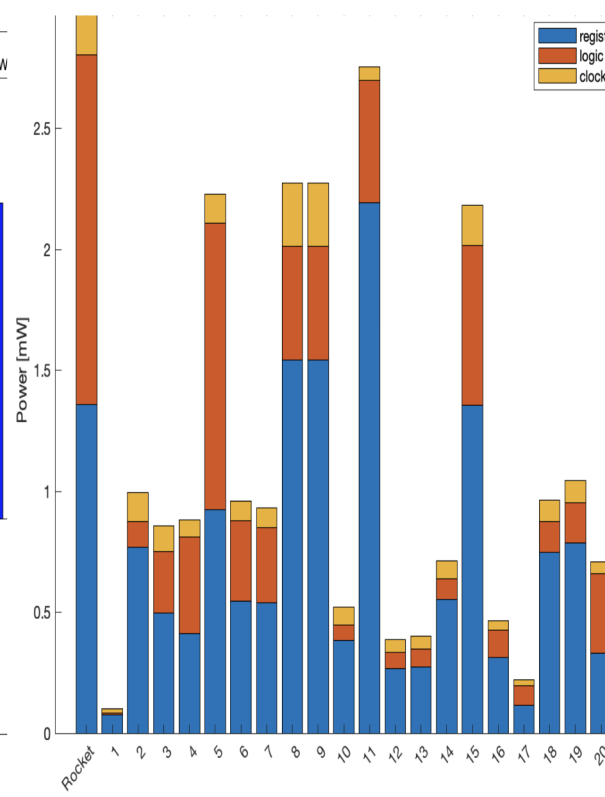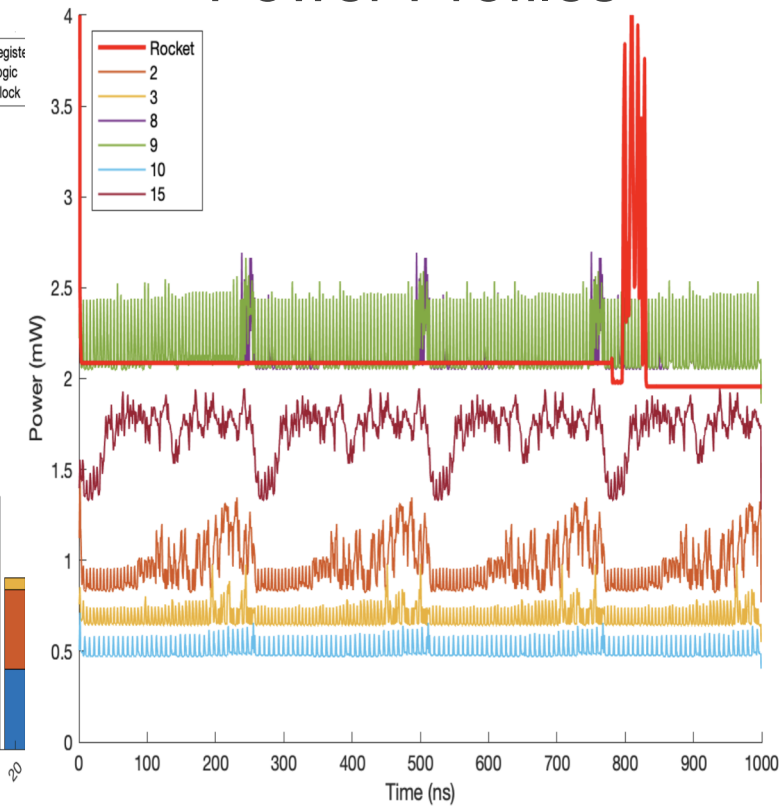
## post-synthesis results using Hammer
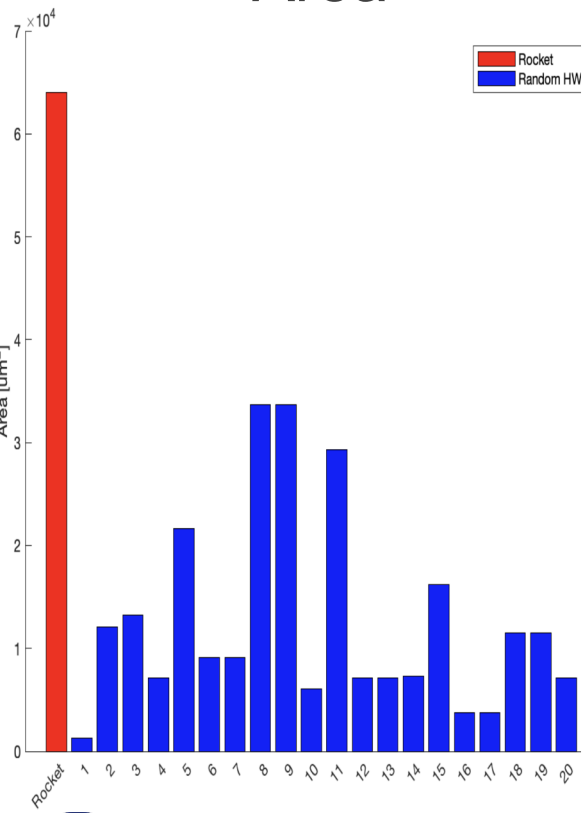


Area | Setup Slack | Avg Power | Power Profiles
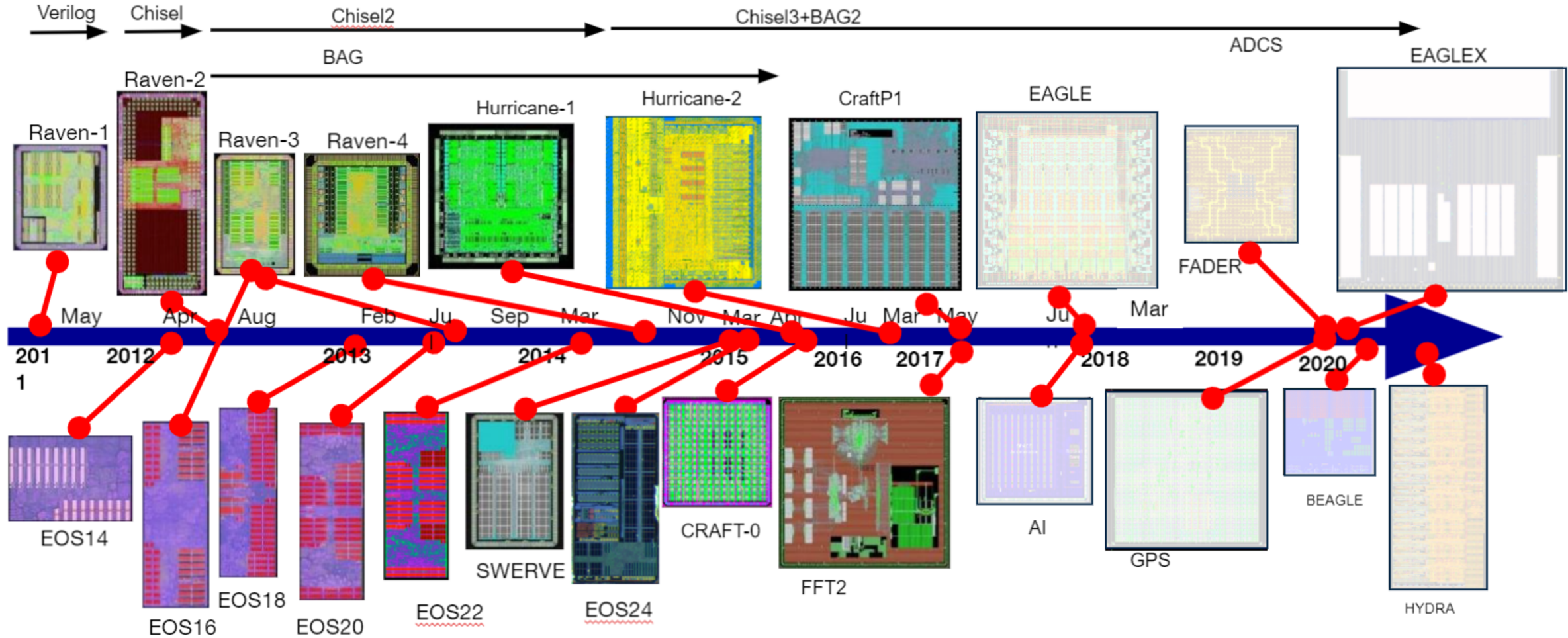
# Motivation: Real Tapeouts



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

Berkeley Architecture Research

# 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.

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

# 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

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

# Hammer Features

- PDKs: ASAP7, Skywater 130nm
- Commercial Tools: Cadence [Genus, Innovus, Joules, Voltus], Siemens Calibre, Synopsys [DC, VCS]
- Open-source Tools: Yosys, OpenROAD
- 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
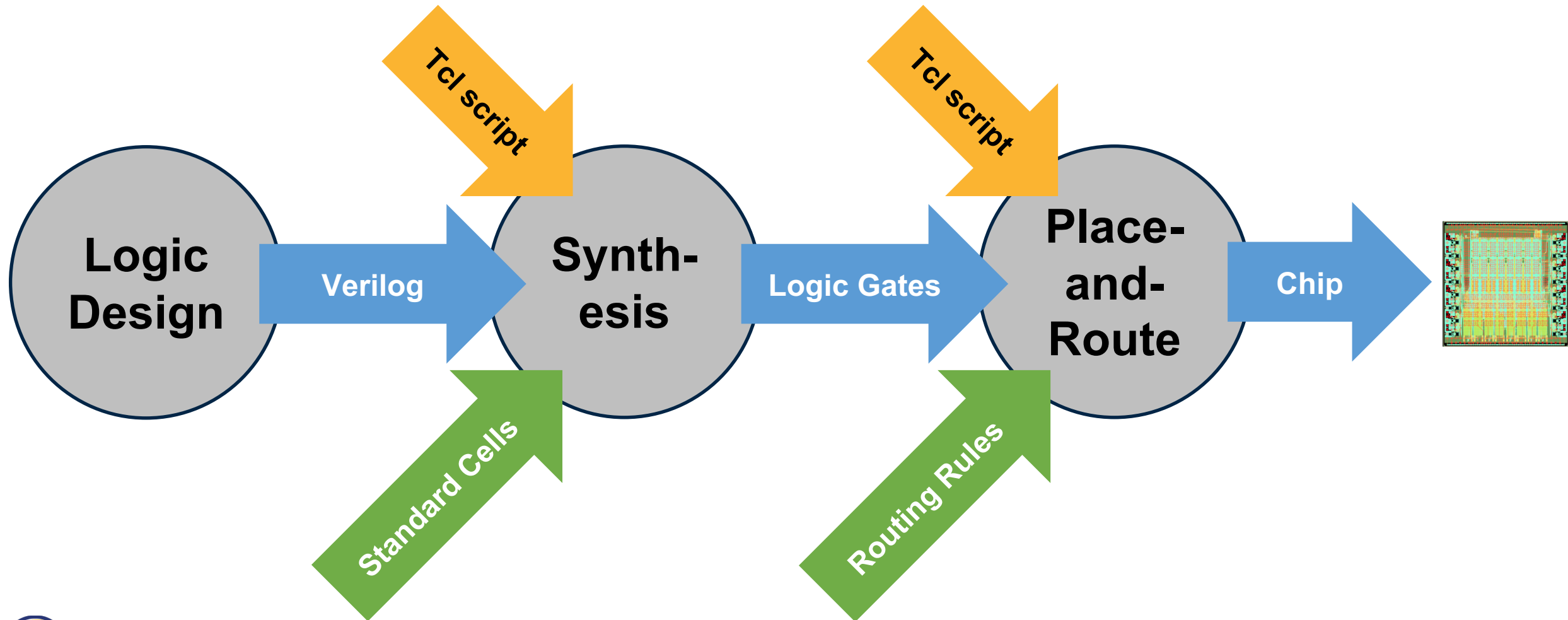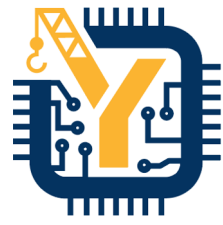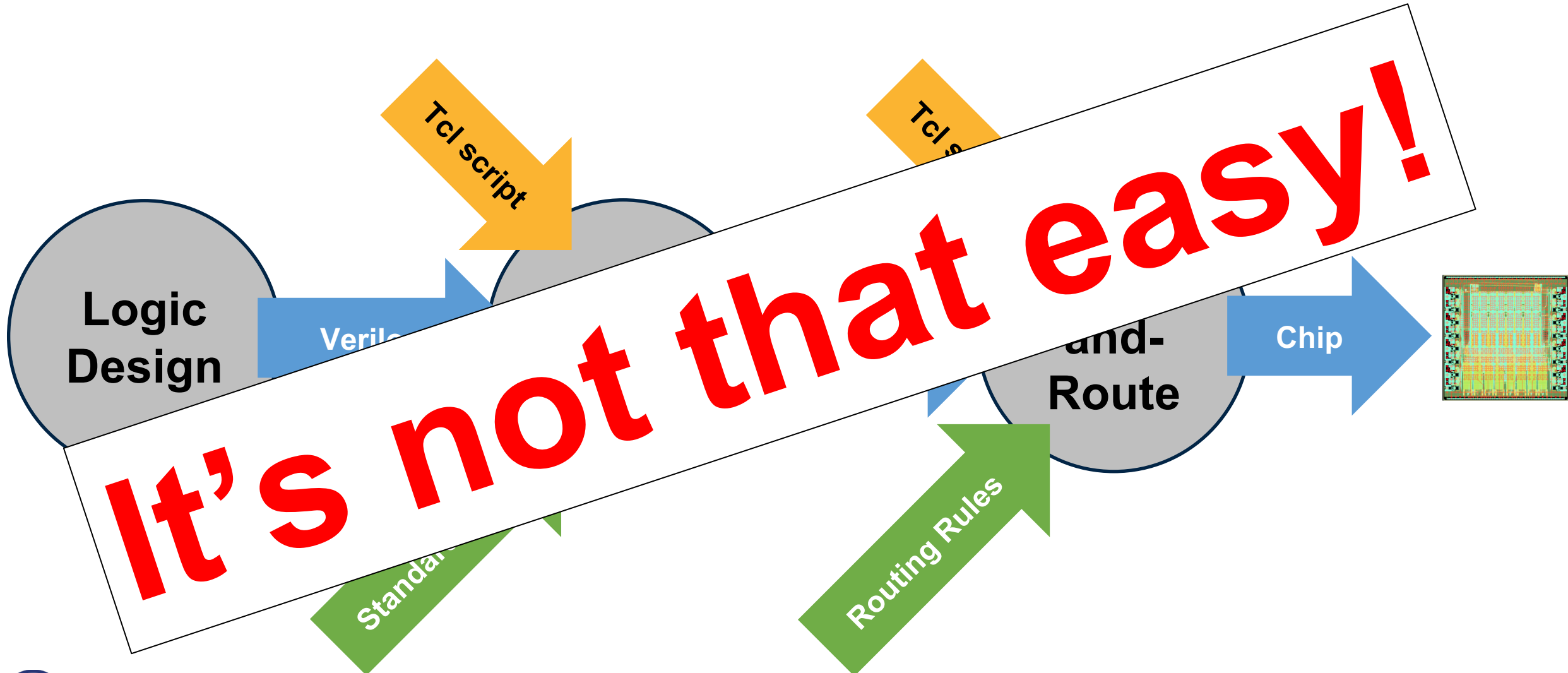- Verified with successful tapeouts in TSMC 28, Intel 16, GF 12

**Berkeley Architecture Research**

# What is Hammer, and why?

# An "Advertised" VLSI Flow

It's not that easy!

Logic Design → Verilog → Tcl script → Standard Cells → Routing Rules → Place-and-Route → Chip

# 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

**B**erkeley **A**rchitecture **R**esearch
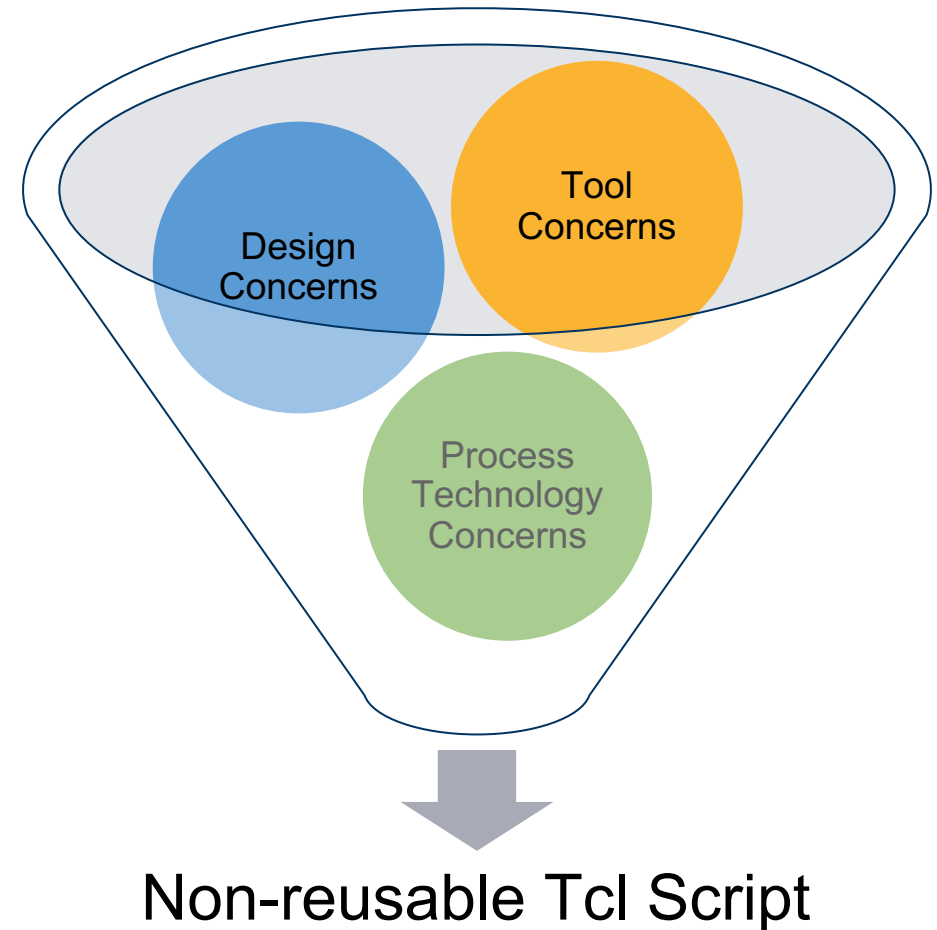
# A Real VLSI Flow

- **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

# Example: Power Straps

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



From chip package

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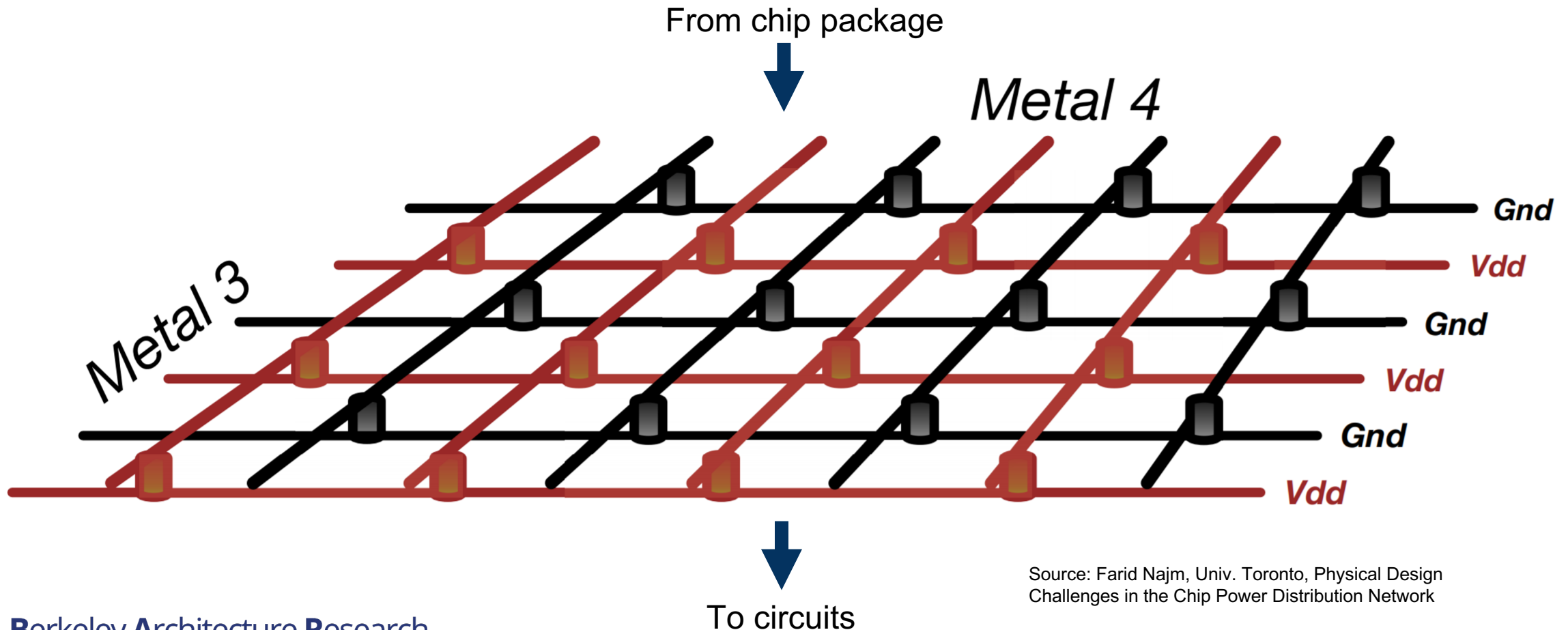
Source: Farid Najm, Univ. Toronto, Physical Design Challenges in the Chip Power Distribution Network

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

14

# 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

Berkeley Architecture Research

# 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
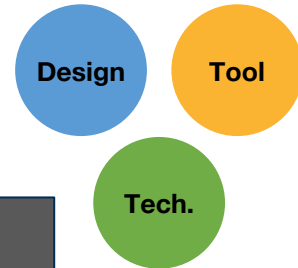- Tcl code
- Tech. file formats

**Tech.:**
- SRAMs
- Std. cells
- Stack-up
- Power straps

**Berkeley Architecture Research**

# Hammer IR + Plugins

- Hammer IR 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"}
]
```
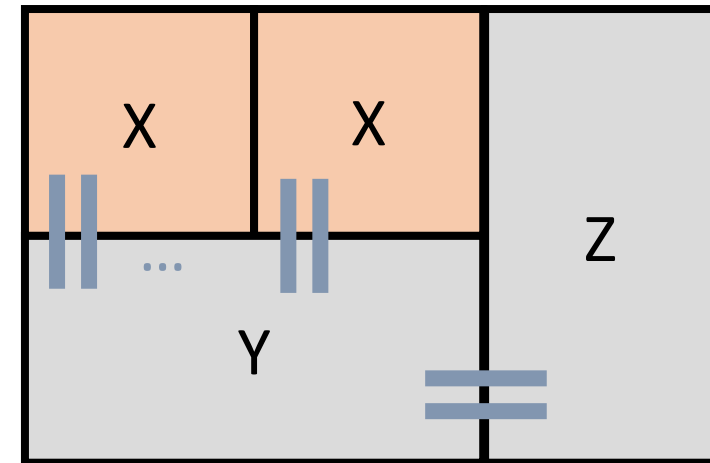
# Power Straps Example

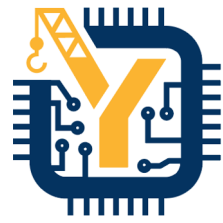**Separated Concerns**
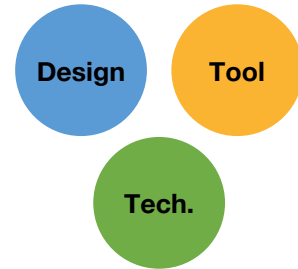
- To specify power straps, need to know:
  - DRC rules
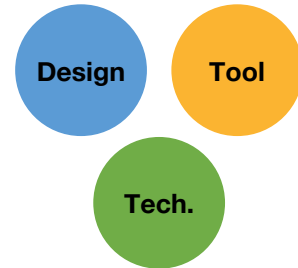  - Target power dissipation
  - IR drop spec
  - Domain areas

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



**Berkeley Architecture Research**

# Power Straps Example
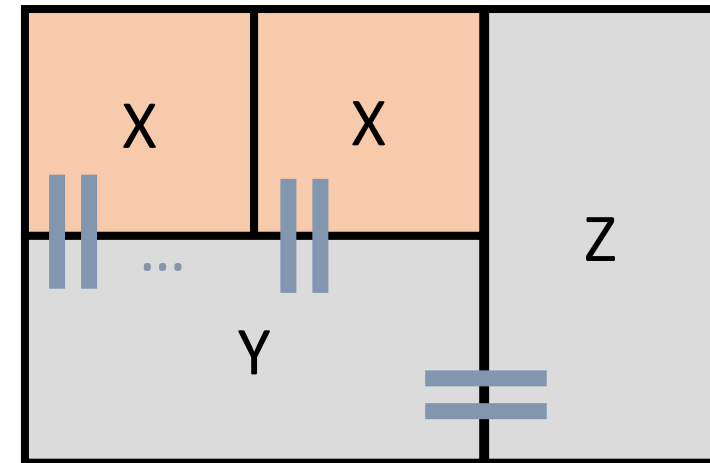
- Don't make the designer do math!
  - Codify design process in tech- and tool-agnostic code

- 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**

# Power Straps Example

**Separated Concerns**

Choose power strap strategy
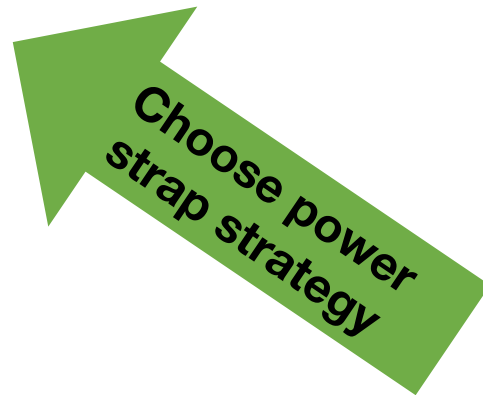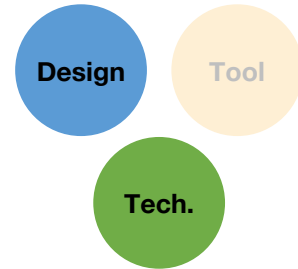
```
par.generate_power_straps_method: by_tracks
par.power_straps_mode: generate
```

# Power Straps Example



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

**Allocate tracks**

**Separated Concerns**

Design    Tool

Tech.

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

**4 tracks VSS   4 tracks VDD**

**number of power domains = 2 (VDD, VSS)
tracks per group = 4 tracks x 2 domains = 8**
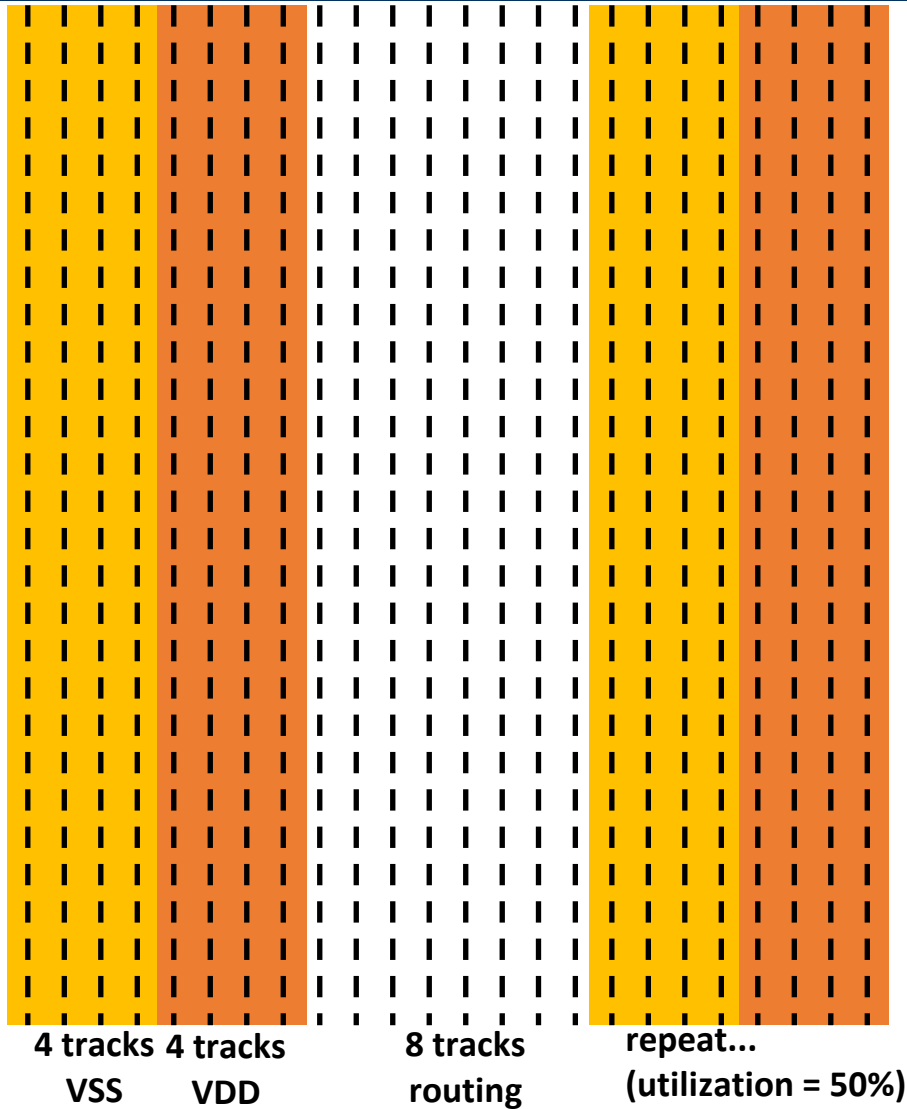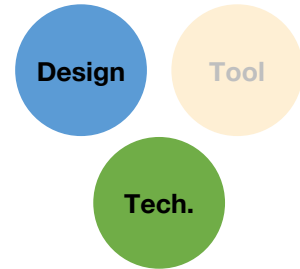
Berkeley **A**rchitecture **R**esearch

# Power Straps Example



**Separated Concerns**

Design   Tool

Tech.

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

Determine pitch

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

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

4 tracks
VSS

4 tracks
VDD

8 tracks
routing

repeat...
(utilization = 50%)

Berkeley Architecture Research

# 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
```
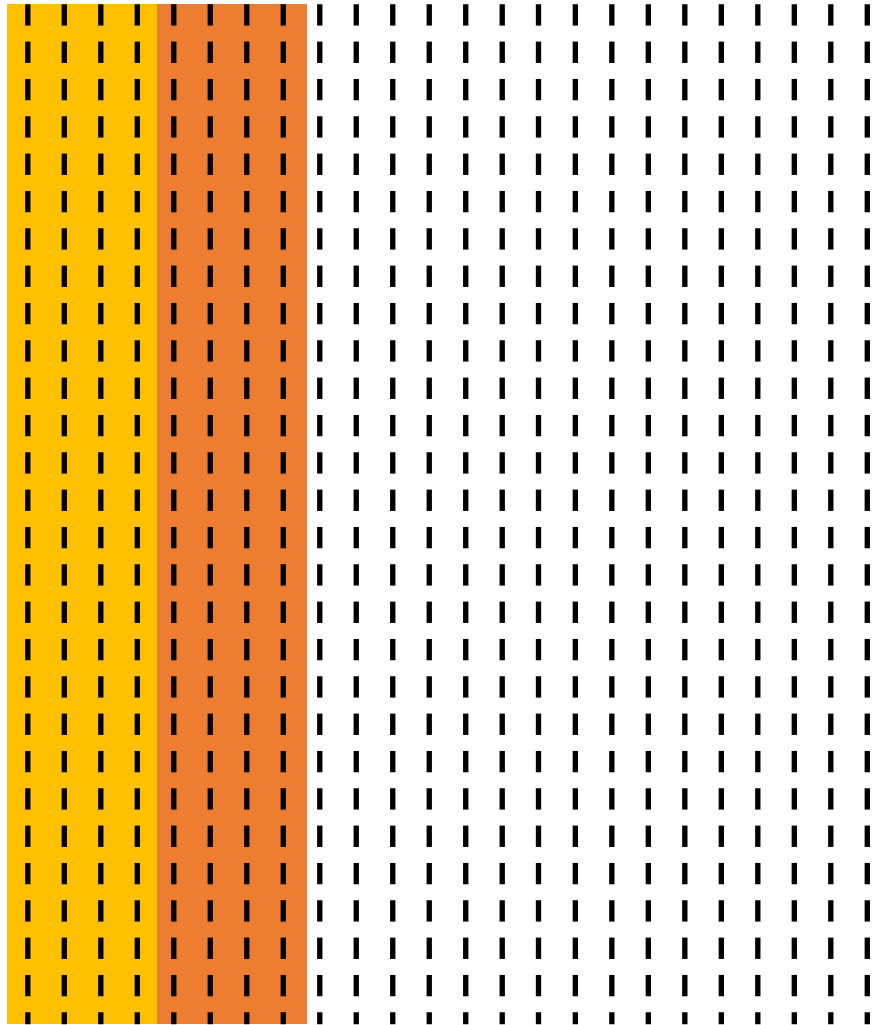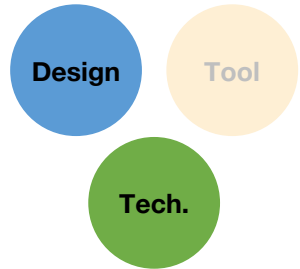
Generate straps

4 tracks VSS   4 tracks VDD   8 tracks routing   repeat... (utilization = 50%)

# 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
```

**Route design**

4 tracks VSS    4 tracks VDD    8 tracks routing    repeat... (utilization = 50%)

Berkeley Architecture Research

# Power Straps Example



power_utilization: 0.1
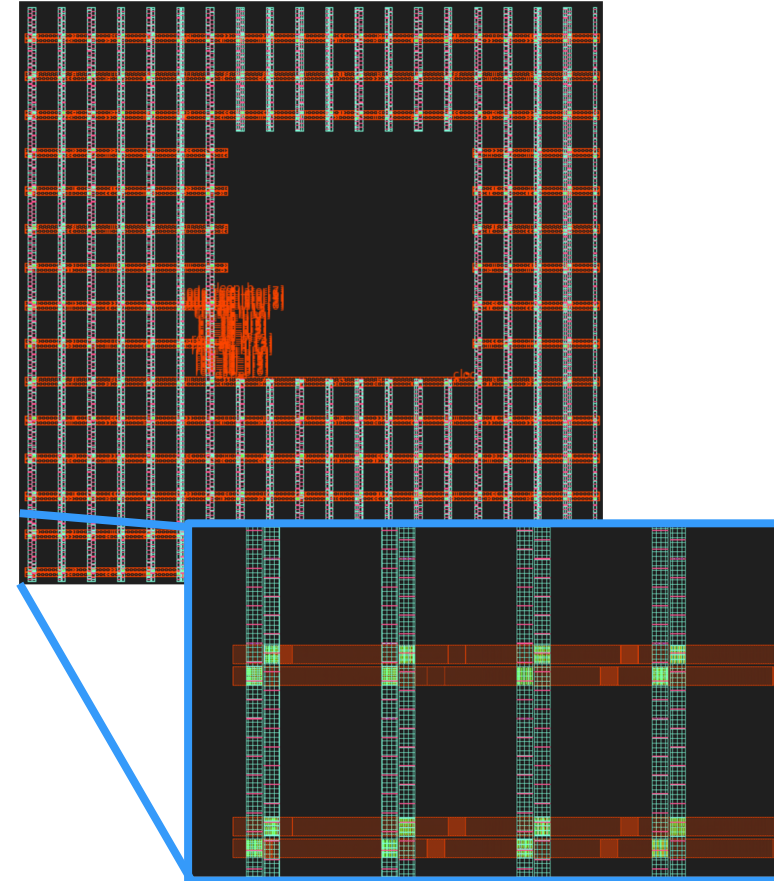
power_utilization: 0.3

# 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
```

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

# 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.

Berkeley Architecture Research

# Getting Started: Hammer Setup
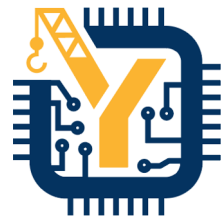
- Initialize Hammer plugins

```
> 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

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

# 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
```

Berkeley Architecture Research

# 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
```

Berkeley Architecture Research

# Getting Started: Tools, PDK

- Enter the version strings for your installed tools

```
# Genus version
synthesis.genus.version: "1813"
# Innovus version
par.innovus.version: "191_ISR3"
# Calibre DRC version
drc.calibre.version: "2017.3_38.30"
# etc...
```

```
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!

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

# 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

Berkeley Architecture Research

# Building the Design

Let's elaborate a TinyRocketConfig:

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

This does the following:

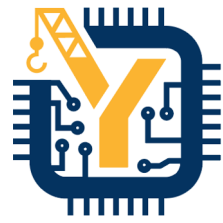- Runs the generator for Top and Harness
  - For the Top, [MacroCompiler](MacroCompiler) maps memories to ASAP7's dummy SRAMs

- 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/
```

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

# 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
```

Result netlists, GDS, etc. appear in here

```
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/
```

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

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

# 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!

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

# Running the VLSI Flow
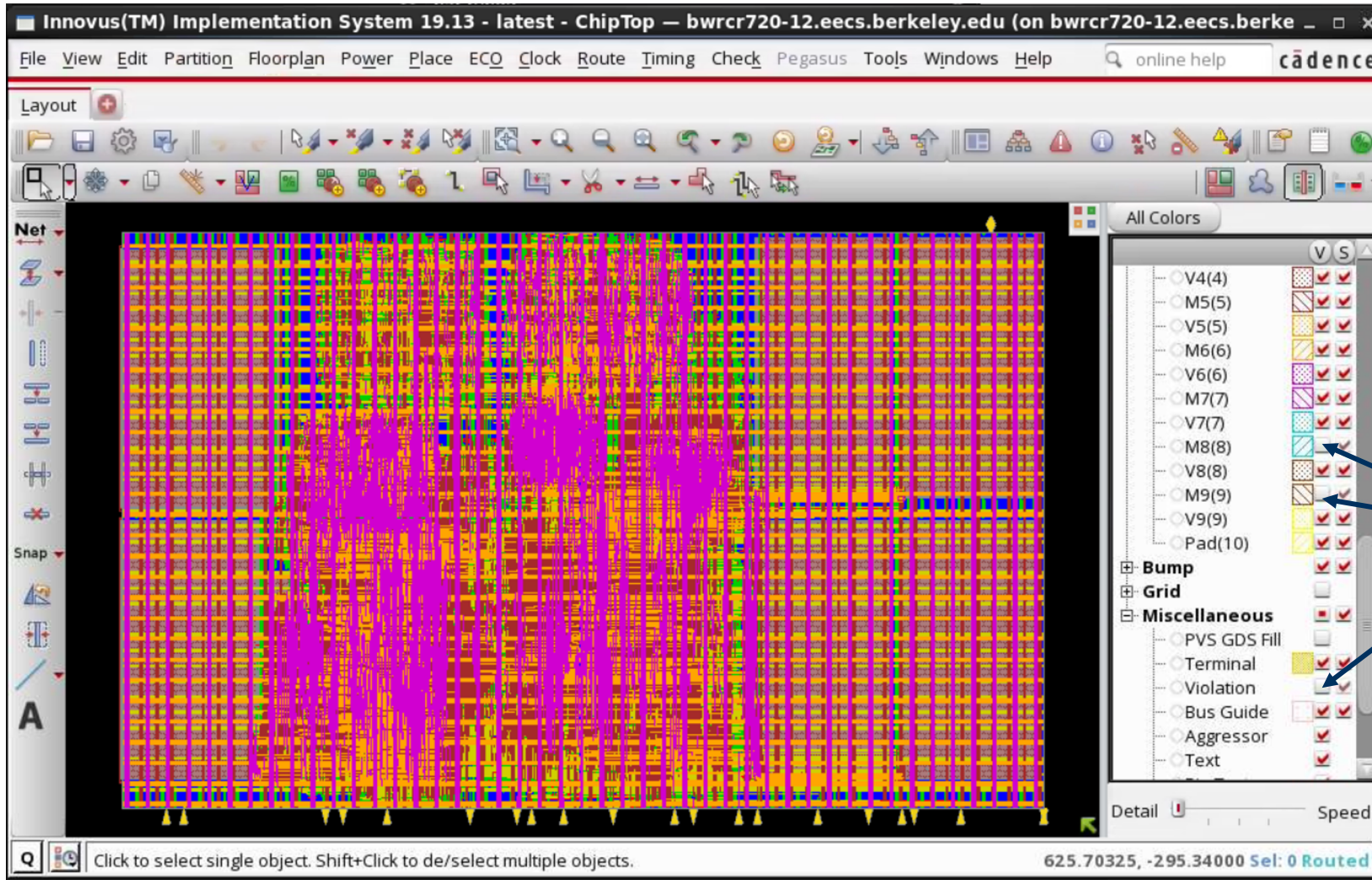


You can get this picture by deselecting M8, M9, and violations

# 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

# 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 Power | Switching Power | Leakage Power | Total Power | Percentage (%) |
|------|---------|----------------|-----------------|---------------|-------------|----------------|
| 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



**B**erkeley **A**rchitecture **R**esearch
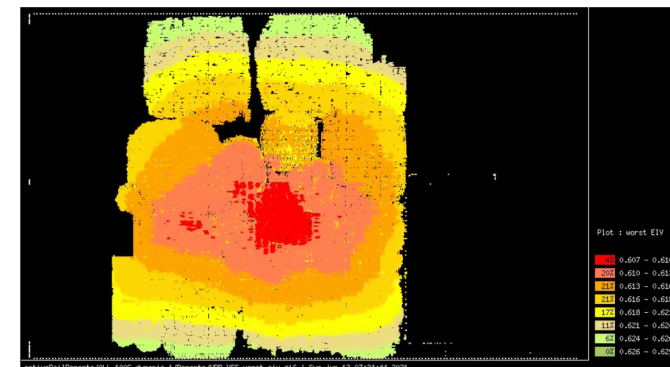
# Signoff

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

```
> make drc CONFIG=TinyRocketConfig
> make lvs CONFIG=TinyRocketConfig
```
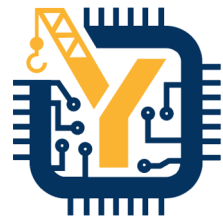
(Optional) View DRC/LVS results in Calibre:

```
> 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
```

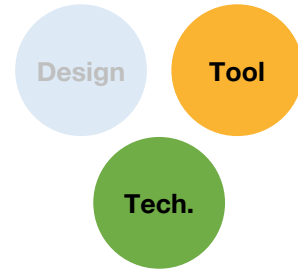ASAP7 does not pass DRC and often fails LVS checking. Calibre experience recommended.

Berkeley Architecture Research

# Under the Hood

# Under the Hood: Plugins

- 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

**Separated Concerns**

Design

Tool

Tech.

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

# What are we re-using? Tool plugin

**Separated Concerns**

Design    **Tool**

Tech.
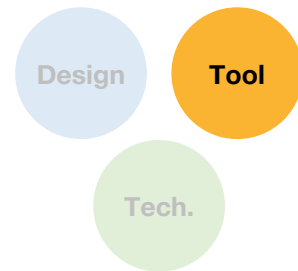
```python
def init_environment(self) -> bool:
    self.create_enter_script()
    verbose_append = self.verbose_append

    verbose_append("set some_cad_variable 123")
    verbose_append("read_corner_files {}".format(mmmc_path))
    if self.hierarchical_mode.is_nonleaf_hierarchical():
        for module in self.get_input_modules():
            verbose_append("read_hier_module –name {}".format(module))

    lef_files = self.technology.read_libs([
        hammer_tech.filters.lef_filter
    ], hammer_tech.HammerTechnologyUtils.to_plain_item)
    verbose_append("read_lef {{ {files} }}".format(
        files=" ".join(lef_files)))
    # ...
```

Remember to request tool plugin access by emailing hammer-plugins-access@lists.berkeley.edu. Some commands are obfuscated so as not to violate EULAs.

Berkeley Architecture Research

# How do I write new tool plugin?

**Separated Concerns**

Design    **Tool**

Tech.

Implement Hammer IR APIs into the specific tool's commands through reusable Python methods
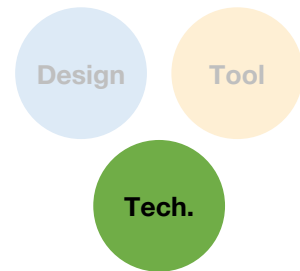
Please refer to the Hammer docs at
https://hammer-vlsi.readthedocs.io

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

# What are we re-using? Tech plugin

```
technology.core
  # This key should exist in the stackups list in the tech json
  stackup: "asap7_3Ma_2Mb_2Mc_2Md"
  # This should specify the TOPMOST metal layer the standard
  #   cells use for power rails.
  # Note that this is not usually stackup specific; It is based
  #   on the std cell libraries themselves
  std_cell_rail_layer: "M1"
  # This is used to provide a reference master for generating power rails
  tap_cell_rail_reference: "{TAPCELL*}"

# Set standard cell LEF placement site
vlsi.technology.placement_site: "coreSite"

# Set the layer that blocks vias under bumps
Vlsi.technology.bump_block_cut_layer: "V9"
```

You can view these at: chipyard/vlsi/hammer/src/hammer-vlsi/technology/asap7/defaults.yml

# How do I write new tech plugin?

**Separated Concerns**

Design    Tool

Tech.

Turn unstructured information about the process technology into a structured representation:

Please refer to the Hammer docs at
https://hammer-vlsi.readthedocs.io/

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

# Under the Hood: Design Concerns
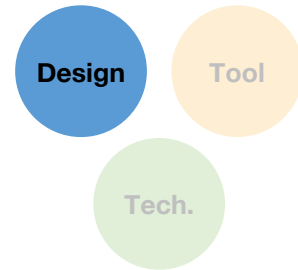
These are the meat of the physical design process in Hammer, and specified in the main project directory
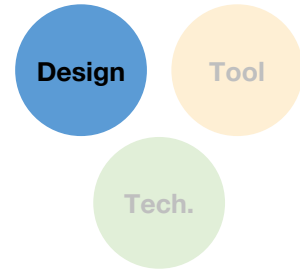
- Integrating analog IP or other hard IP
  - Examples: foundry SRAMs, PLLs, ADCs, etc.

- Floorplanning

- Clock and power

- Hierarchy assembly

- Boilerplate: selecting the process technology and tools

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

# example-asap7.yml – Power and Clocking

**Separated Concerns**

- Specify clock signal and constraints

```
# Specify clock signals
vlsi.inputs.clocks: [
    {name: "clock", period: "1ns", uncertainty: "0.1ns"}
]
```
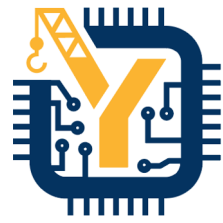
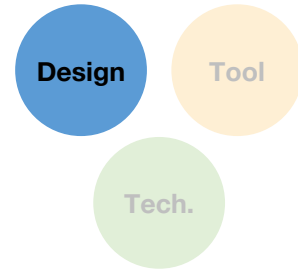- Specify automatic generation of a simple power specification

```
# Hammer will auto-generate a CPF for simple power designs;
see hammer/src/hammer-vlsi/defaults.yml for more info
vlsi.inputs.power_spec_mode: "auto"
vlsi.inputs.power_spec_type: "cpf"
```

**Berkeley Architecture Research**

# example-asap7.yml – Placement Constraints

**Separated Concerns**

Design

Tool

Tech.

```yaml
vlsi.inputs.placement_constraints:
  - path: "ChipTop"
    type: toplevel
    x: 0
    y: 0
    width: 800
    height: 500
    margins:
      left: 0
      right: 0
      top: 0
      bottom: 0
  - path:
"ChipTop/system/tile_prci_domain/tile_reset_domain/til
e/dcache/data/data_arrays_0/data_arrays_0_ext/mem_0_0"
    type: hardmacro
    x: 550
    y: 25
    orientation: "r0"
    top_layer: "M4"
    master: "SRAM1RW4096x8"
```

Top-level is ChipTop, 800um x 500um

DCache SRAM hardmacro placed at (550, 25), no flipping, is instance of SRAM1RW4096x8

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

51

# example-asap7.yml – Pin Placement

- Need pins to connect to levels higher in the hierarchy
- `semi_auto` pin placement uses the CAD tool's default pin distribution methods
  - Pin Placement on metal layers M5, M7

```
# 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"}
]
```

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

# example-asap7.yml – Power Straps

- Generate power straps using the previously mentioned custom Hammer API ("`by_tracks`")

- Auto-generate straps with 2um space to blockages

- Maximizes width within track_width (7) tracks while satisfying DRC

- Override defaults per layer by appending _<layer>
  - Example: `power_utilization_M9`

**Separated Concerns**

Design

Tool

Tech.

```
# Power Straps
par.power_straps_mode: generate
par.generate_power_straps_method: by_tracks
par.blockage_spacing: 2.0
par.generate_power_straps_options:
  by_tracks:
    strap_layers:
      - M3
      - M4
      - M5
      - M6
      - M7
      - M8
      - M9
    pin_layers:
      - M9
    track_width: 7 # minimum allowed for M2 & M3
    track_spacing: 0
    track_spacing_M3: 1 # to avoid M2 shorts at
higher density
    track_start: 10
    power_utilization: 0.2
    power_utilization_M8: 1.0
    power_utilization_M9: 1.0
```

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

# sram_generator-output.json – Analog/Hard IP

- Extra Libraries – Hard IP (analog, third party IP, etc.)
  - Specify the collateral files for each corner
- If needed: specify "physical only" cells
  - Cells with no behavioral or other analysis details

**Separated Concerns**

Design   Tool

Tech.

```
"vlsi.technology.extra_libraries": [
    {
        "library": {
            "name": "SRAM1RW4096x8",
            "corner": {
                "nmos": "slow",
                "pmos": "slow",
                "temperature": "100.0 C"
            },
            "lef file": "/path/to/SRAM1RW4096x8_x4.lef",
            "gds file": "/path/to/SRAM1RW4096x8_x4.gds",
            "nldm liberty file":
"/path/to/SRAM1RW4096x8_PVT_0P63V_100C.lib",
            "provides": [
                {
                    "lib_type": "sram",
                    "vt": "SRAM"
                }
            ],
            "supplies": {
                "GND": "0 V",
                "VDD": "0.63 V"
            },
            "verilog sim": "hammer/src/hammer-
vlsi/technology/asap7/sram_compiler/memories/behavioral/
sram_behav_models.v"
        }
    },
...
"vlsi.technology.extra_libraries_meta": ["append"]
```

Berkeley Architecture Research

54

# 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

**Berkeley Architecture Research**

# "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
```

# Driver – example-vlsi

- Project-specific entry script extending hammer-vlsi
    - Location for specifying hooks needed for each tool, hierarchical module, and custom flow actions

```python
class ExampleDriver(CLIDriver):
    def get_extra_par_hooks(self) -> List[HammerToolHookAction]:
        extra_hooks = [
            # Default set of steps can be found in the CAD tool plugin's __init__.py

            # make_pre_insertion_hook will execute the custom hook before the specified step
            # SYNTAX: make_pre_insertion_hook("EXISTING_STEP", INSERTED_HOOK)
            # hammer_vlsi.HammerTool.make_pre_insertion_hook("route_design", example_add_fillers),

            # make_post_insertion_hook will execute the custom hook after the specified step
            # hammer_vlsi.HammerTool.make_post_insertion_hook("init_design", example_tool_settings),

            # make_replacement_hook will replace the specified step with a custom hook
            # hammer_vlsi.HammerTool.make_replacement_hook("place_tap_cells", example_place_tap_cells),

            # make_removal_hook will remove the specified step from the flow
            hammer_vlsi.HammerTool.make_removal_hook("place_bumps"),

            # The target step in any of the above calls may be a default step or another one of your custom hooks
        ]
        return extra_hooks
```

Berkeley Arch

# 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.**
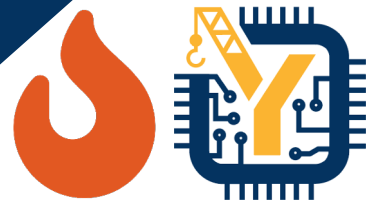
**Berkeley Architecture Research**
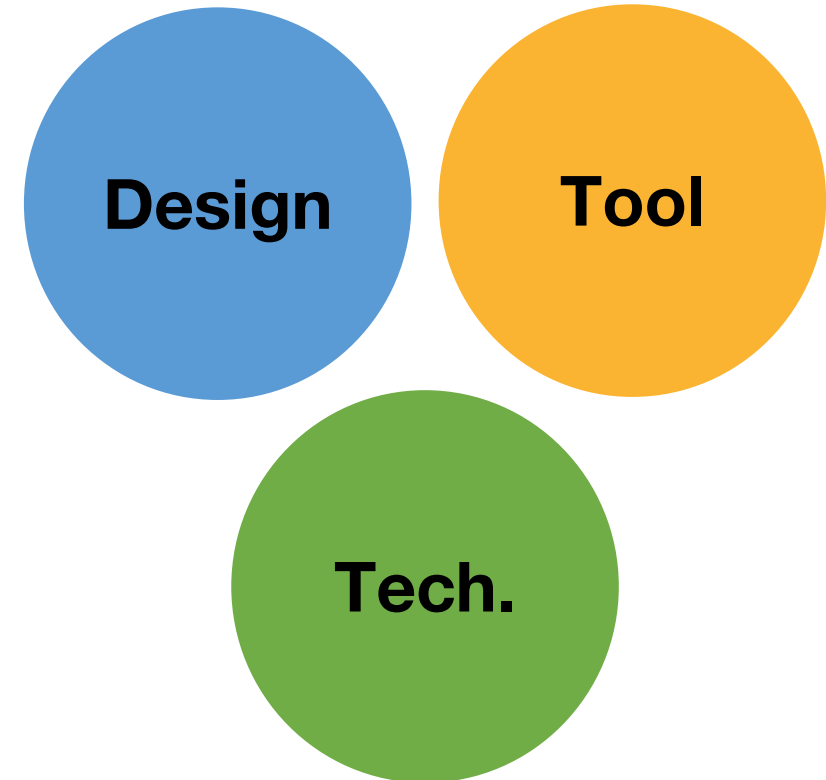
# Future of Hammer
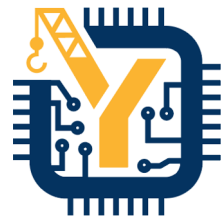
- More tool integration: metrics parsing, constraints feedback
- Open-source tools + PDKs: OpenROAD
- Aspect-Oriented Chisel Floorplanning
  - Higher-level abstractions
    - Hierarchical partitioning, power strap/pin alignment
    - BAG IP collateral generation/integration
  - Reconfiguring Chisel designs 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

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

# Coming up…
# FPGA Prototyping