Hammer VLSI Flow

John Wright

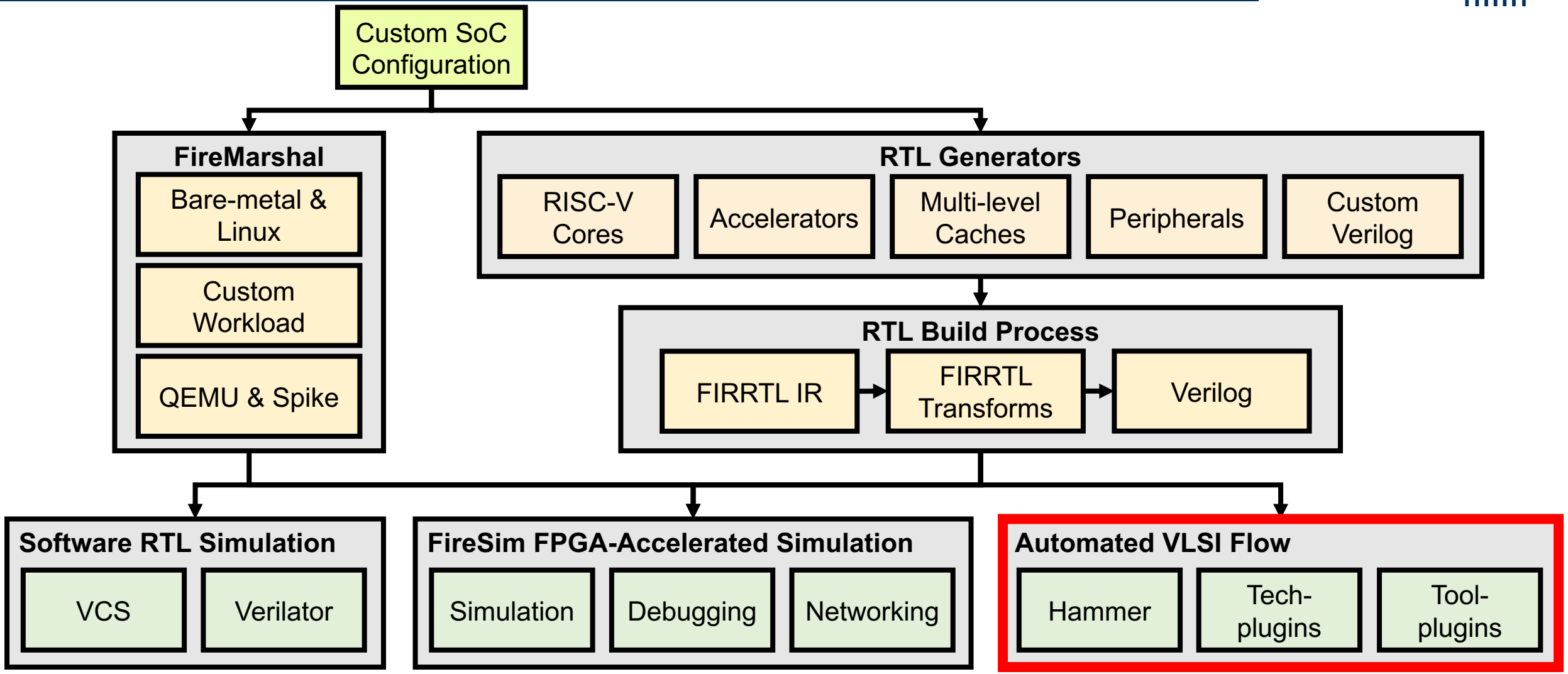
UC Berkeley

johnwright@berkeley.edu

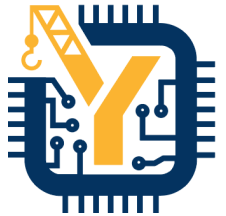
Berkeley Architecture Research

CHIPYARD

Tutorial Roadmap

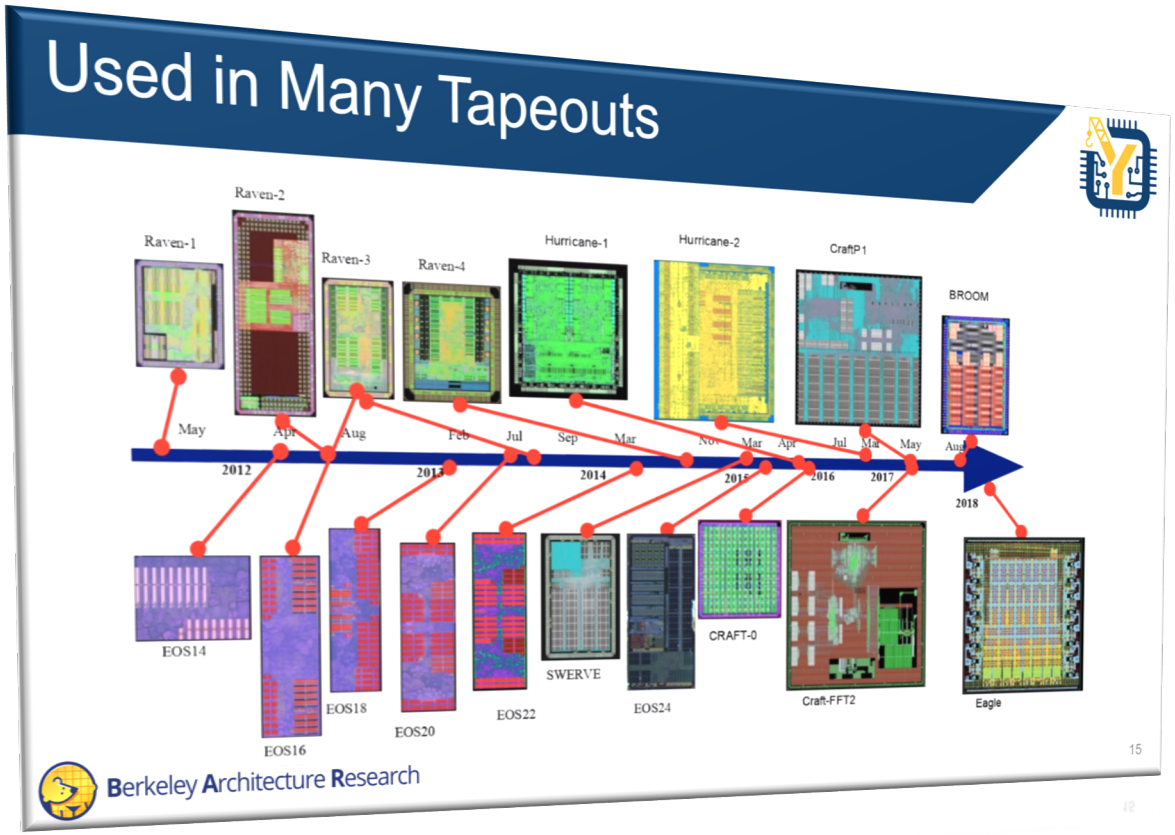


Agenda



- VLSI flow
- Challenges in VLSI flows
- Hammer physical design flow principles
- How to use Hammer
- Demo SHA3 accelerator physical design flow

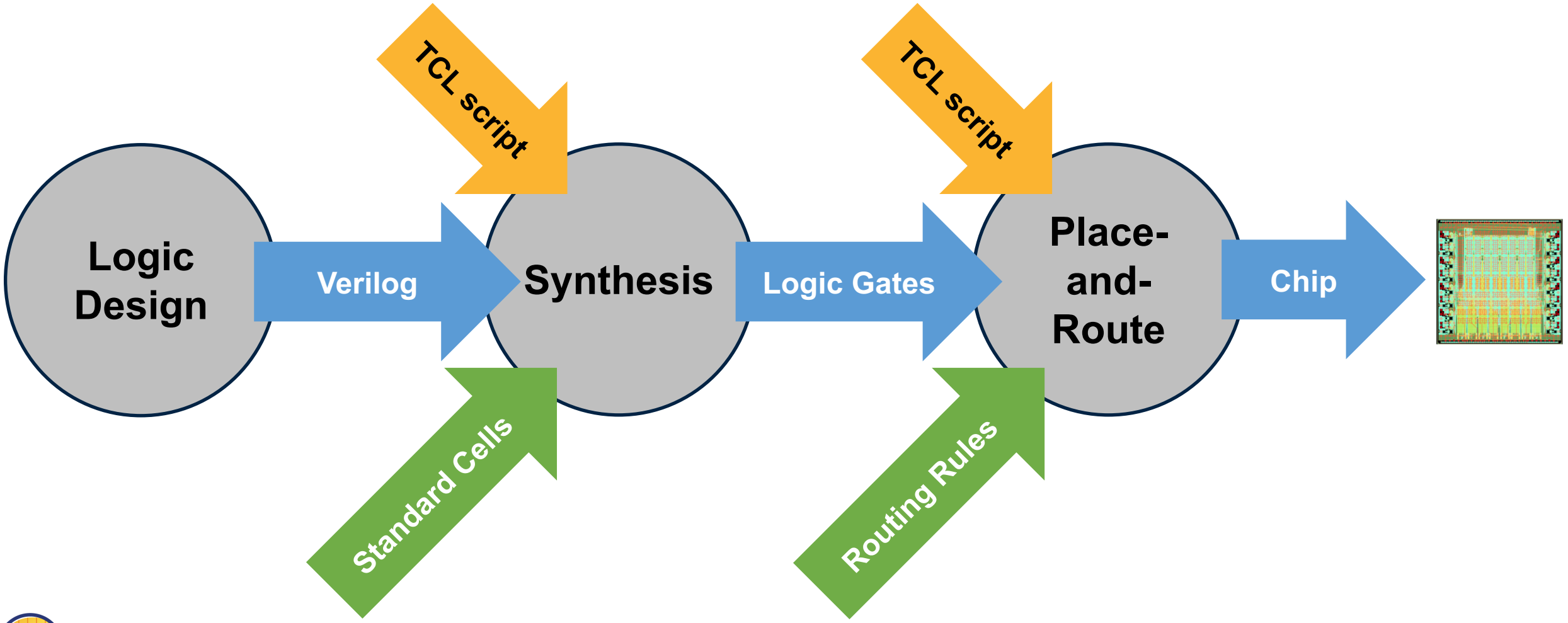
VLSI for Real Hardware



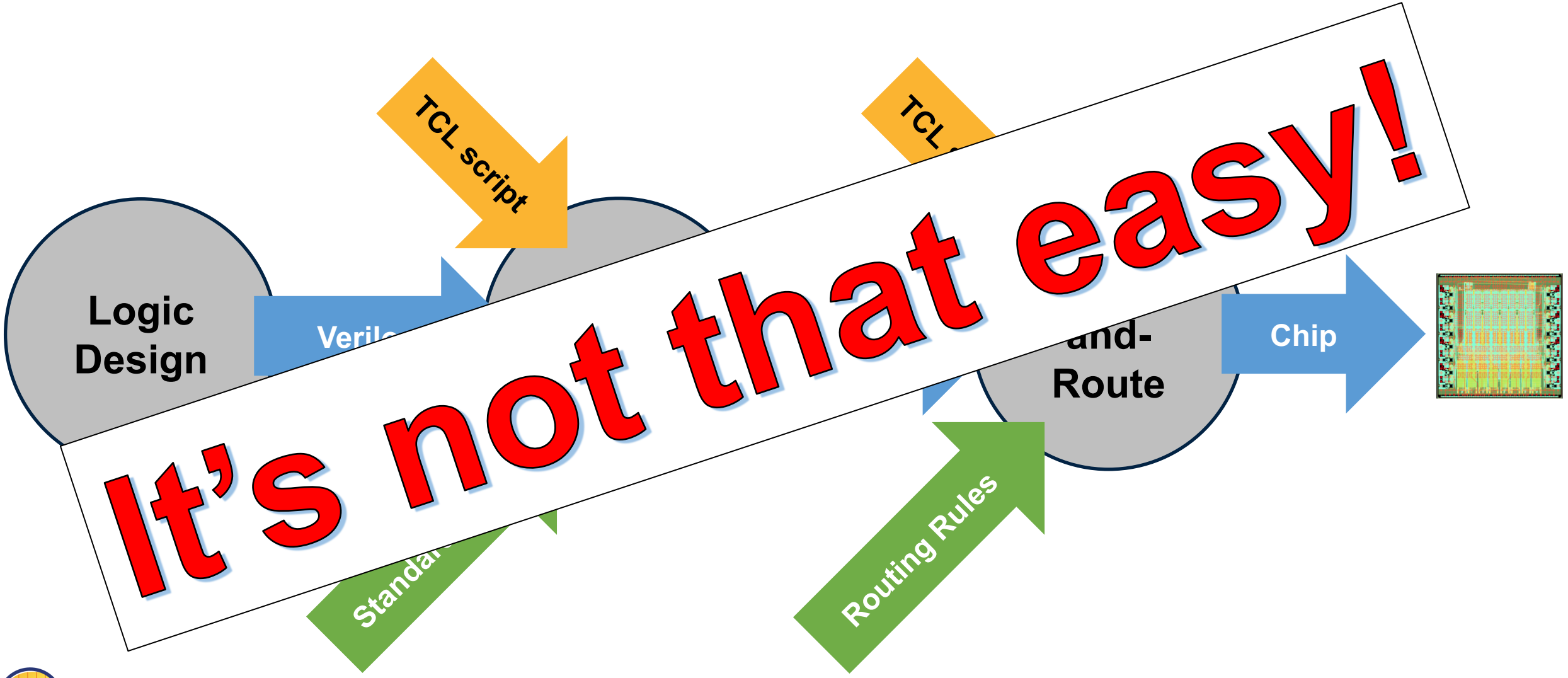
Remember this slide from the introduction?

Real hardware requires VLSI work!

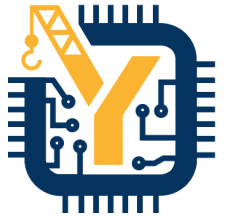
An “Advertised” VLSI Flow



An “Advertised” VLSI Flow

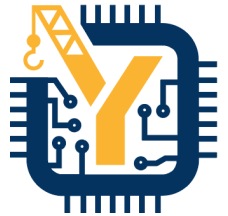


A Real VLSI Flow

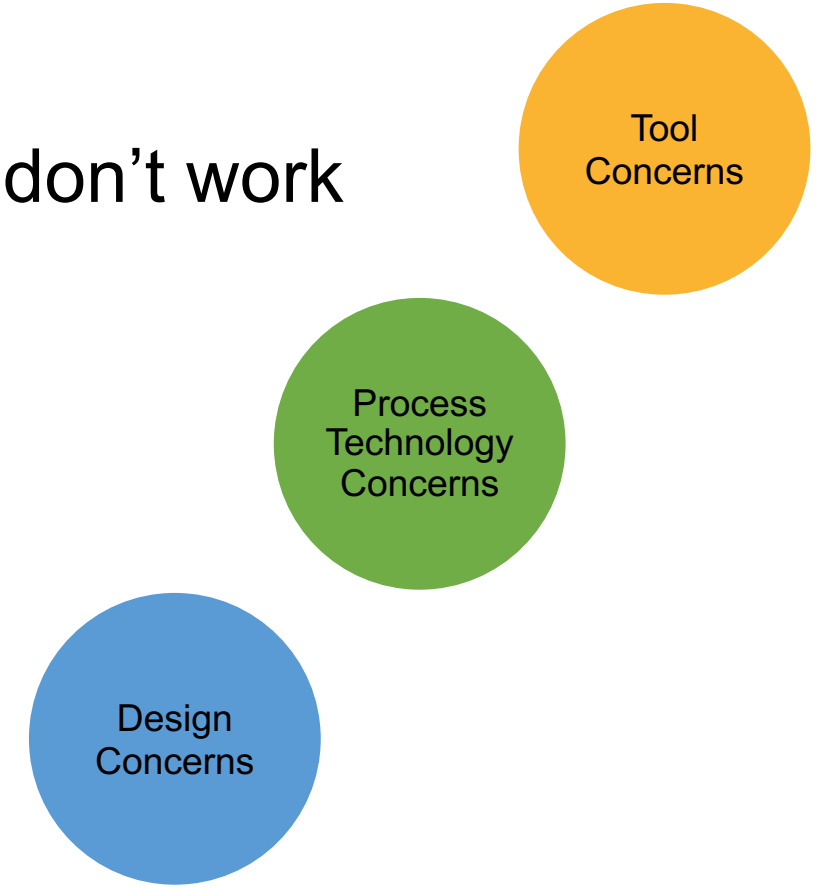


RTL is ready → Foundry delivers PDK tarball → Unzip PDK; Slowly discover there are missing CAD-tool-specific files → Send a few emails to the foundry → Download a new PDK → Try running synthesis → Find out you are using the wrong time units and standard cell library → Iterate on synthesis for a week → 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

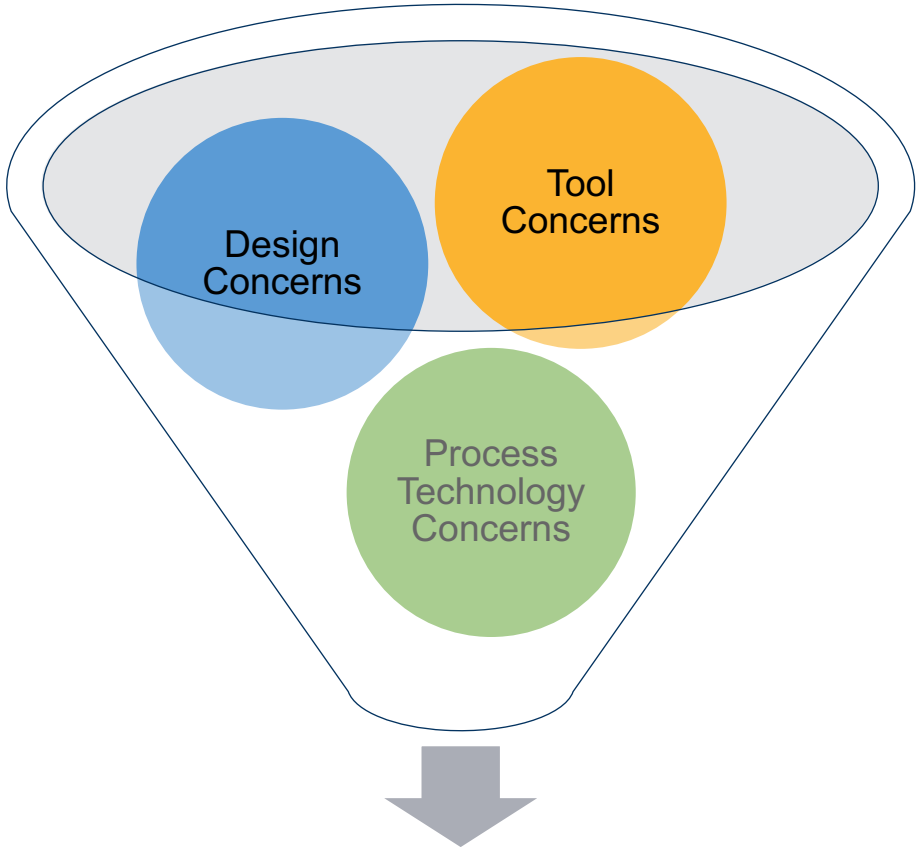


- The physical design (VLSI) flow must be rebuilt for each project
- Overhead compounded by
  - Changing CAD tool vendors
    - Commands change. Features work/ don't work
    - File formats / library locations
  - Using a new process technology
    - SRAMs (compiled/pre-generated?)
    - New DRC rules
  - Changing the design itself
    - Floorplanning / power / clock



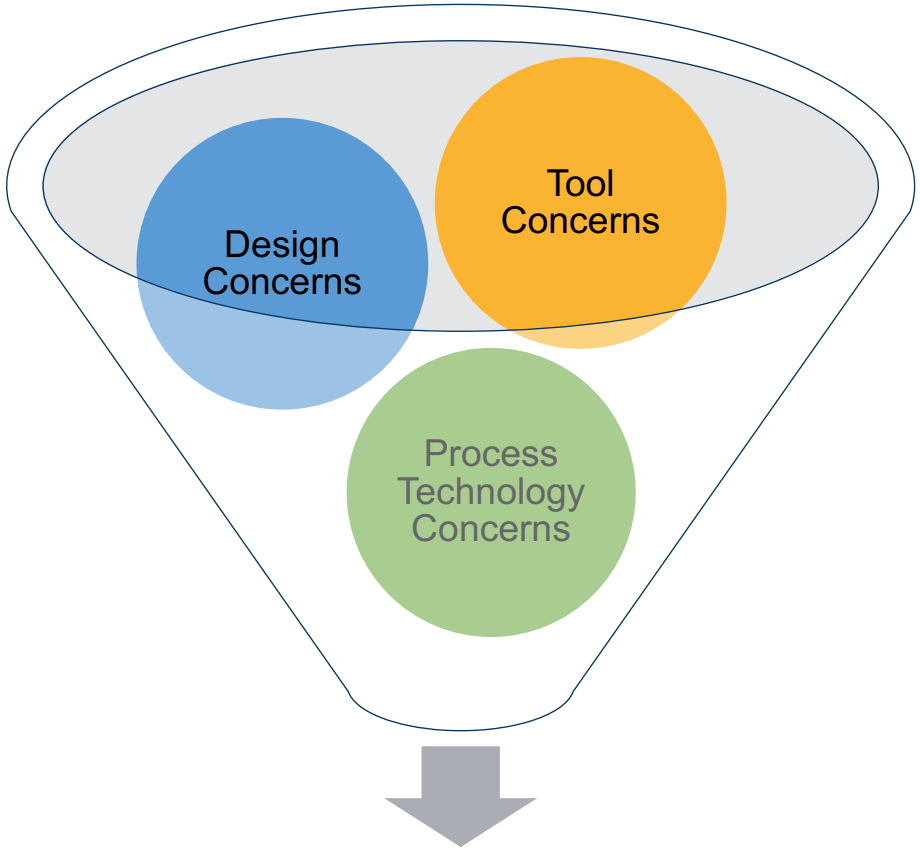
Why So Complicated?

- Designs are getting bigger and more complex
- EDA industry evolution
  - EDA tools evolve bottom-up through patches and acquisitions
  - No [real] common exchange formats or APIs
- All physical design concerns mixed together



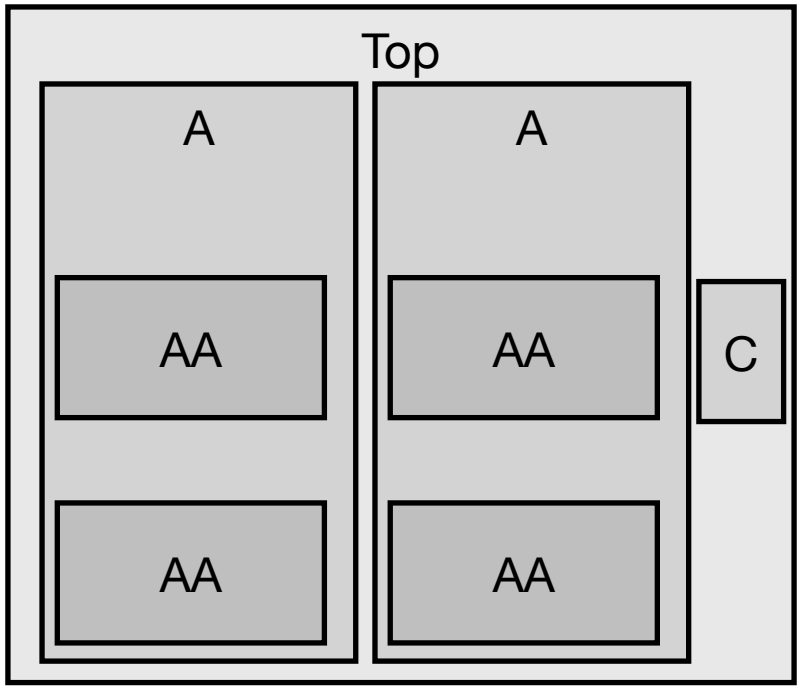
Why So Complicated?

- Designs are getting bigger and more complex
- EDA industry evolution
  - EDA tools evolve bottom-up through patches and acquisitions
  - No [real] common exchange formats or APIs
- All physical design concerns mixed together



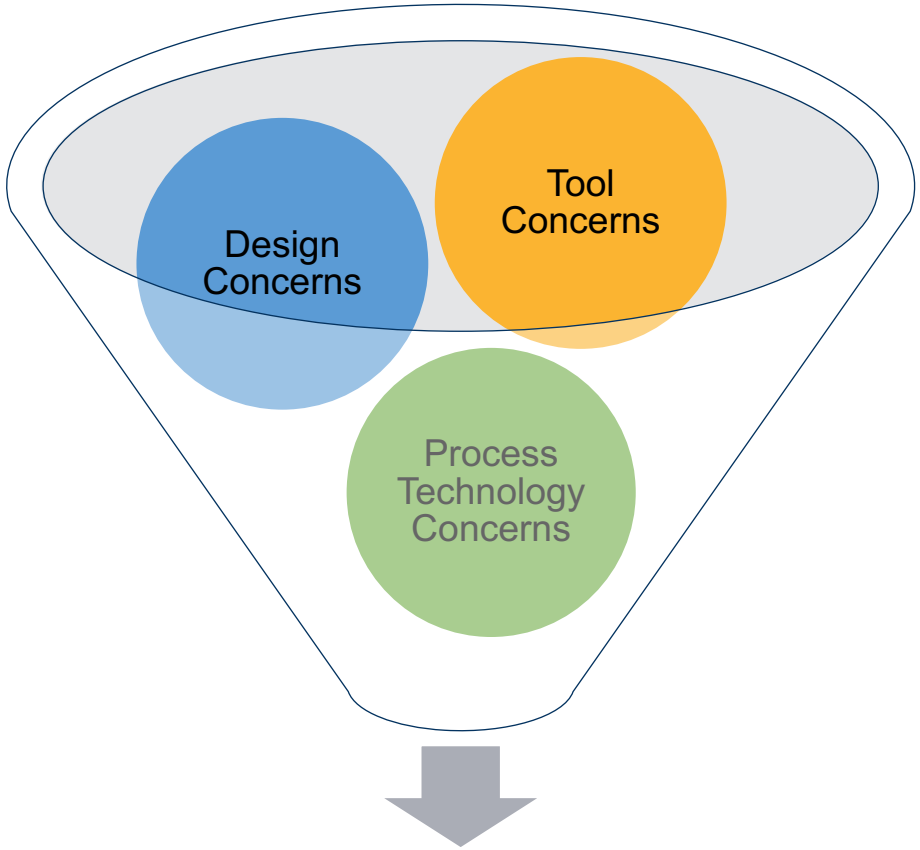
Hierarchical Design

- Why hierarchical physical design?
  - Modern chips are complex w/ prohibitively large place-and-route time
  - Divide-and-conquer alleviates this problem
- Problem: Hierarchical isn't "Free"
  - Floorplanning is complicated
    - Alignment of power straps, placement sites, and pins
  - Tools want physical and logical hierarchies to match
    - How to determine logical hierarchy?
  - Constraining timing on the I/O boundaries
- Hammer helps simplify hierarchical flows

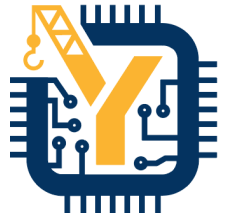


Why So Complicated?

- Designs are getting bigger and more complex
- EDA industry evolution
  - EDA tools evolve bottom-up through patches and acquisitions
  - No [real] common exchange formats or APIs
- All physical design concerns mixed together



Some TCL code…



- Consider a hypothetical power strap creation command:

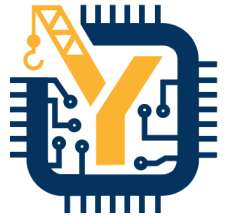
```
set some_proprietary_option M1
set some_other_proprietary_option M3
create_power_stripes -nets {VSS VDD} -layer M2 -direction vertical \
   -via_start M1 -via_stop M3 -group_pitch 43.200 -spacing 0.216 -width 0.936 \
   -area [get_bbox -of ModuleABC]   \
   -start [expr [lindex [lindex [get_bbox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

# Repeat for each layer!
```

- Writing a line of TCL to place 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 EULA

Hammer “Separation of Concerns”



- Solution: Add a layer of abstraction

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

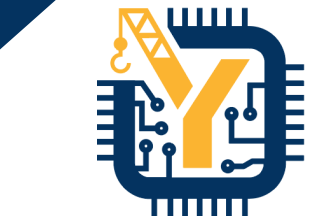
- Hammer Goal: specify all three separately
  - Allow reusability
  - Allow for multiple “small” experts instead of a single “super” expert
  - Build abstractions/APIs on top

Design:
- Floorplan
- Clocks
- Hierarchy

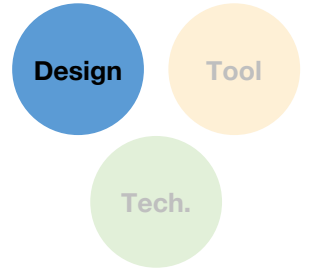
Tool:
- In/out files
- TCL code
- Tech. file formats

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

Design Concerns



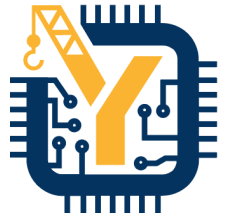
Separated Concerns



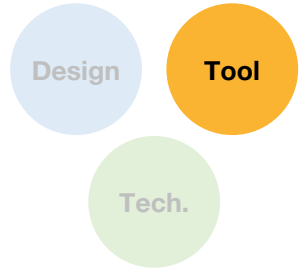
- Floorplan
  - Physical hierarchy
  - Placement constraints
  - Pin constraints
- Clock constraints: frequencies, pin delays, etc.
- Design modifications: retiming, scan insertion, etc.

- Solution: Store these in an Intermediate Representation (IR)
  - Emit from high-level input source
  - Consume by hammer to produce TCL commands and perform quality checks
  - **Need tool plugins to know what TCL to write!**

Tool Concerns

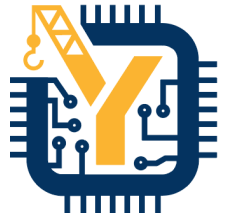


Separated Concerns

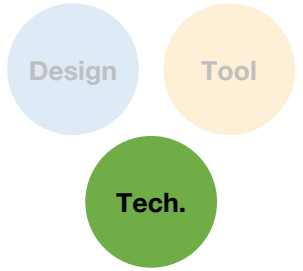


- Site-level problems
  - Installation path, license servers, tool versions
  - Allow these to be overridden by configuration files
- TCL command interface
  - Not standardized between vendors!
  - No commitment to preserve API across versions!
- Solution: Implement Python methods that emit TCL
  - Some implement standard Hammer “steps”
  - Some can be vendor-specific steps
- Solution: Codify in a “Tool Plugin”

Technology Concerns

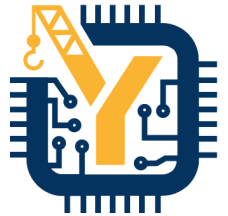


Separated Concerns

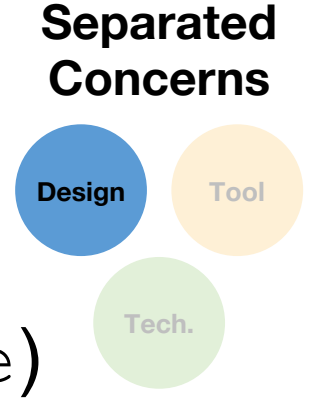


- PDK
  - Install directory
  - Technology files
  - Standard cells, SRAMs, other IP
  - Available PVT (Process/Voltage/Temperature) corners
- Technology-specific TCL commands/snippets
  - Include python methods that are included in the flow
- Solution: Codify in a “Technology Plugin”

Hammer IR

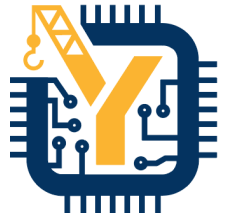


- Hammer IR codifies design information
- Also can override tech- and tool-specific settings
- Can be JSON or YAML (preferred)
- “Namespaces” separate categories of settings (e.g. `vlsi.core`)

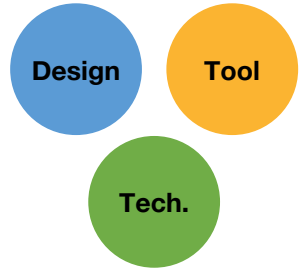



```
# Specify clock signals vlsi.inputs.clocks: [
  {name: "clock", period: "1ns", uncertainty: "0.1ns"}
]
# Generate Make include to aid in flow
vlsi.core.build_system: make
# Pin placement constraints
vlsi.inputs.pin_mode: generated
vlsi.inputs.pin.generate_mode: semi_auto
vlsi.inputs.pin.assignments: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

A Quick Example: Power Straps

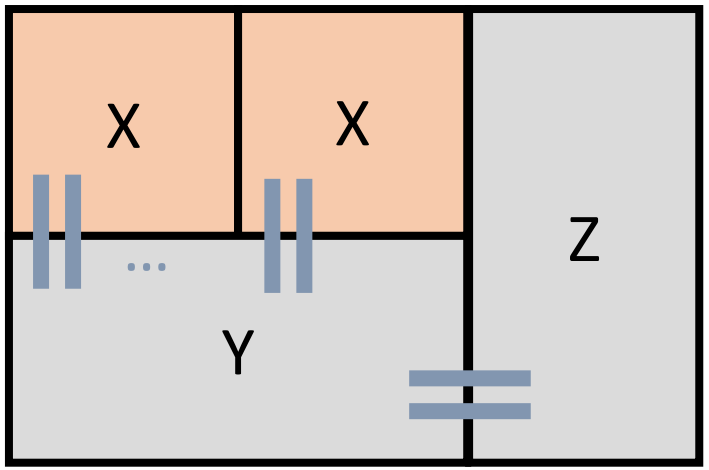


Separated Concerns

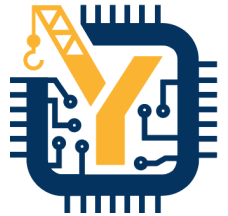


- To specify power straps, need to know:
  - DRC rules
  - Target power dissipation
  - IR drop spec
  - Domain areas

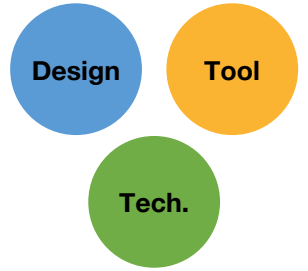
- Hierarchical also adds physical constraints:
  - Tiled modules require pitch-matching
  - Easy to make mistakes when reworking



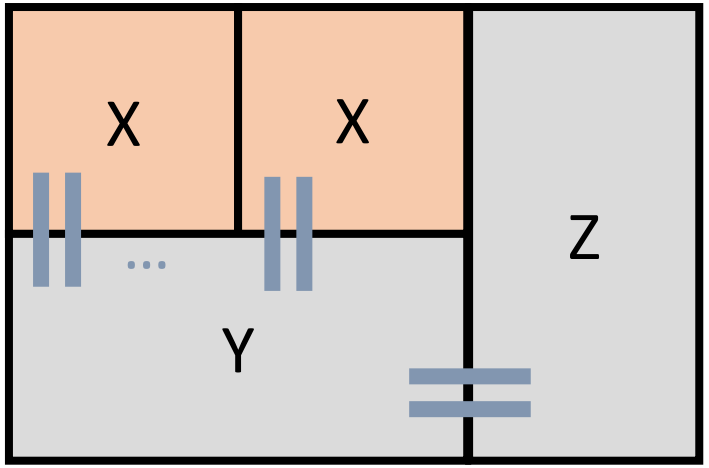
A Quick Example: Power Straps



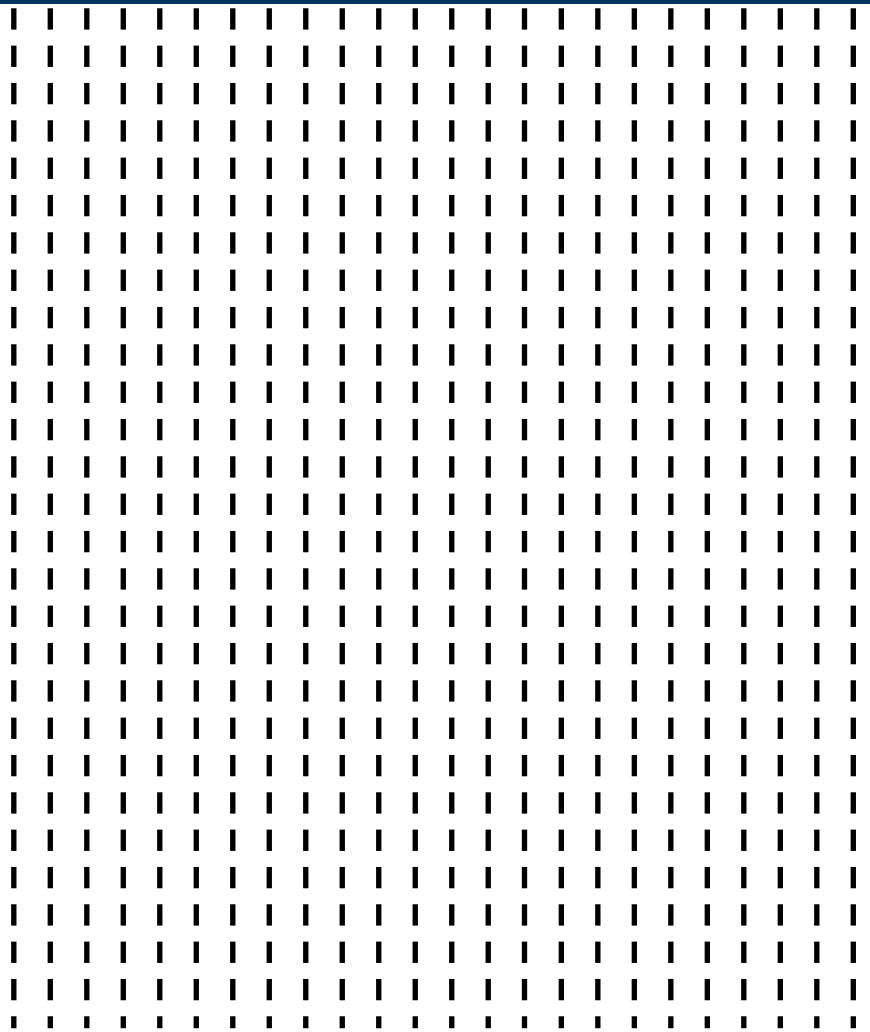
Separated Concerns

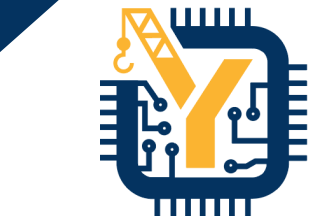


- Don’t make the designer do math
  - Codify design process in tech- and tool-agnostic code
- Method:
  - Determine valid pitches for hierarchical design
  - Automatically calculate offsets for hierarchical blocks
  - Generate layout-optimal, DRC clean straps
  - Specify intent at a higher-level than length units
- Example: Using ”By tracks” specification

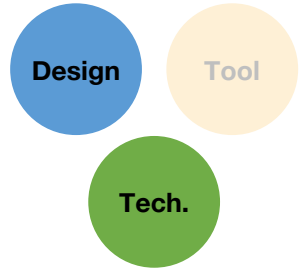


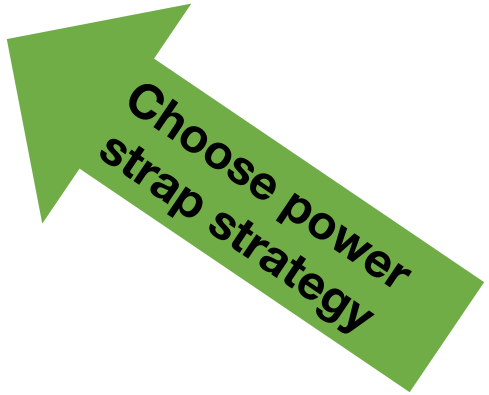
A Quick Example: Power Straps





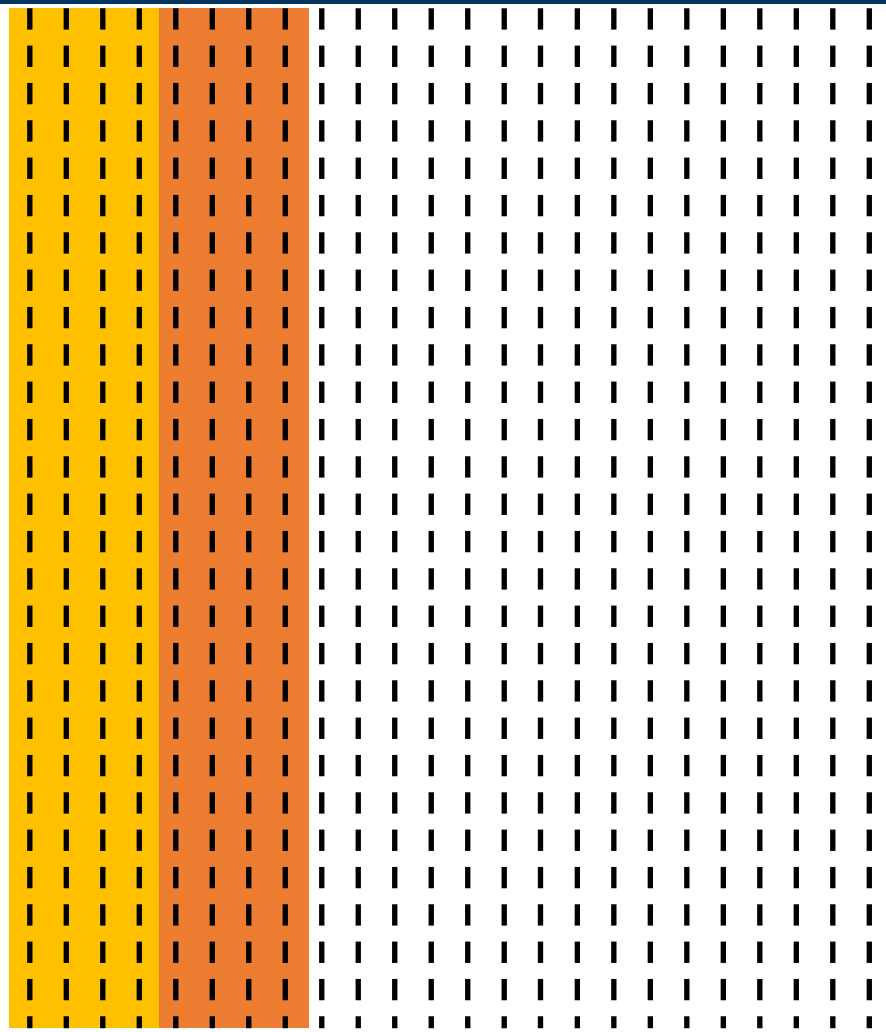
Separated Concerns






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

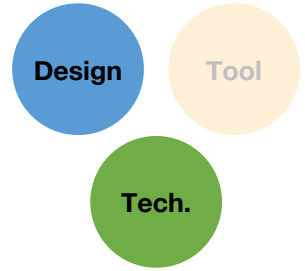
A Quick Example: Power Straps




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

Allocate tracks

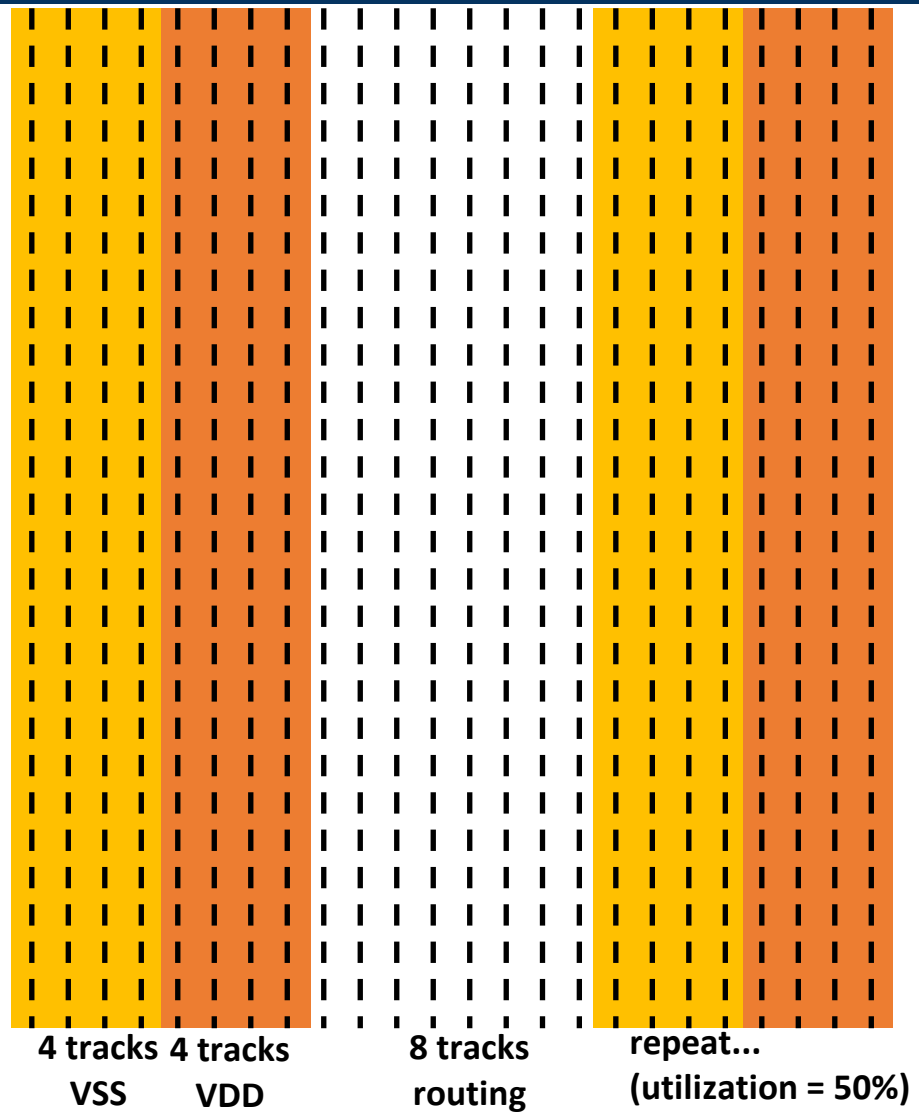
Separated Concerns




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

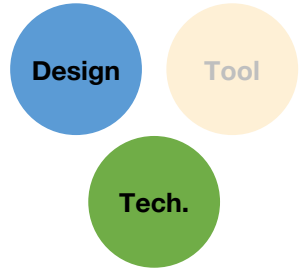
number of power domains = 2 (VDD, VSS)
tracks per group = 4 tracks x 2 domains = 8

A Quick Example: Power Straps




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

Separated Concerns

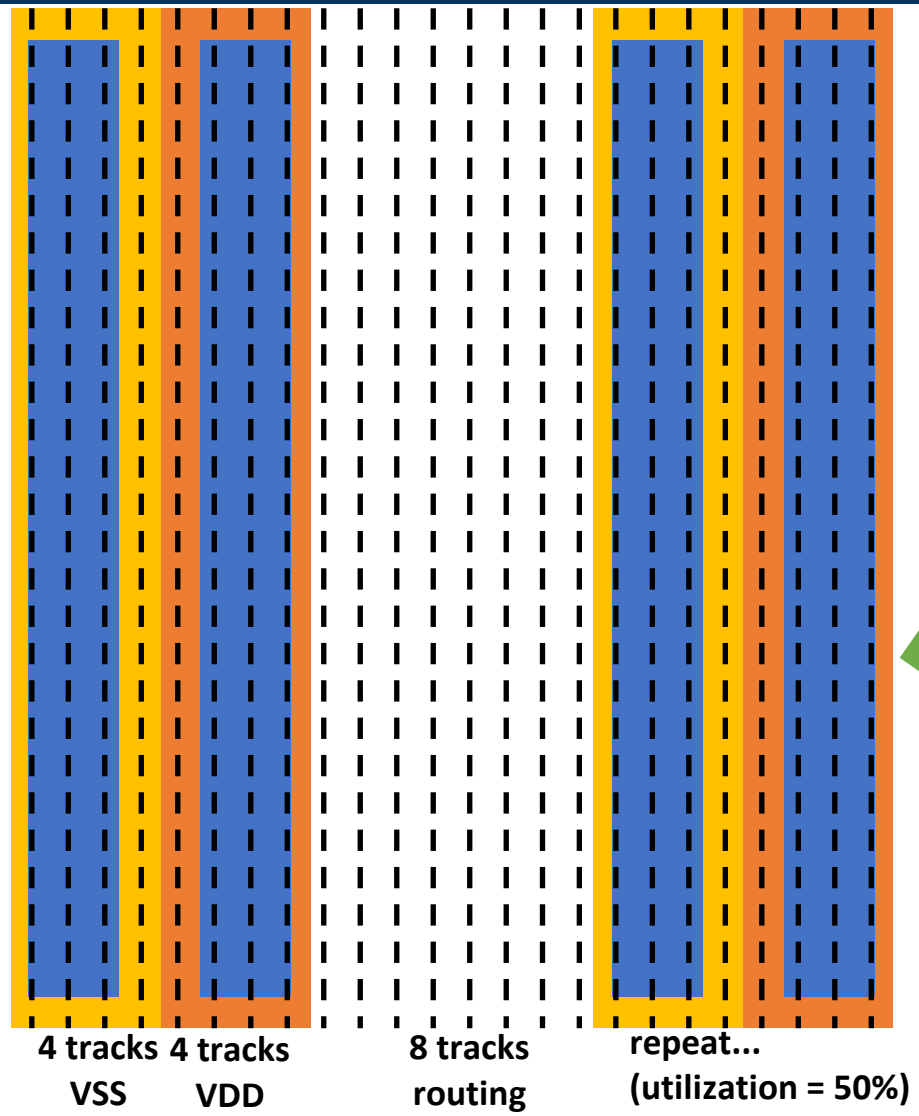


Determine pitch

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

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

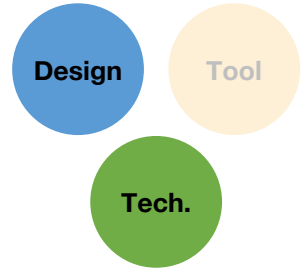
A Quick Example: Power Straps



Generate straps

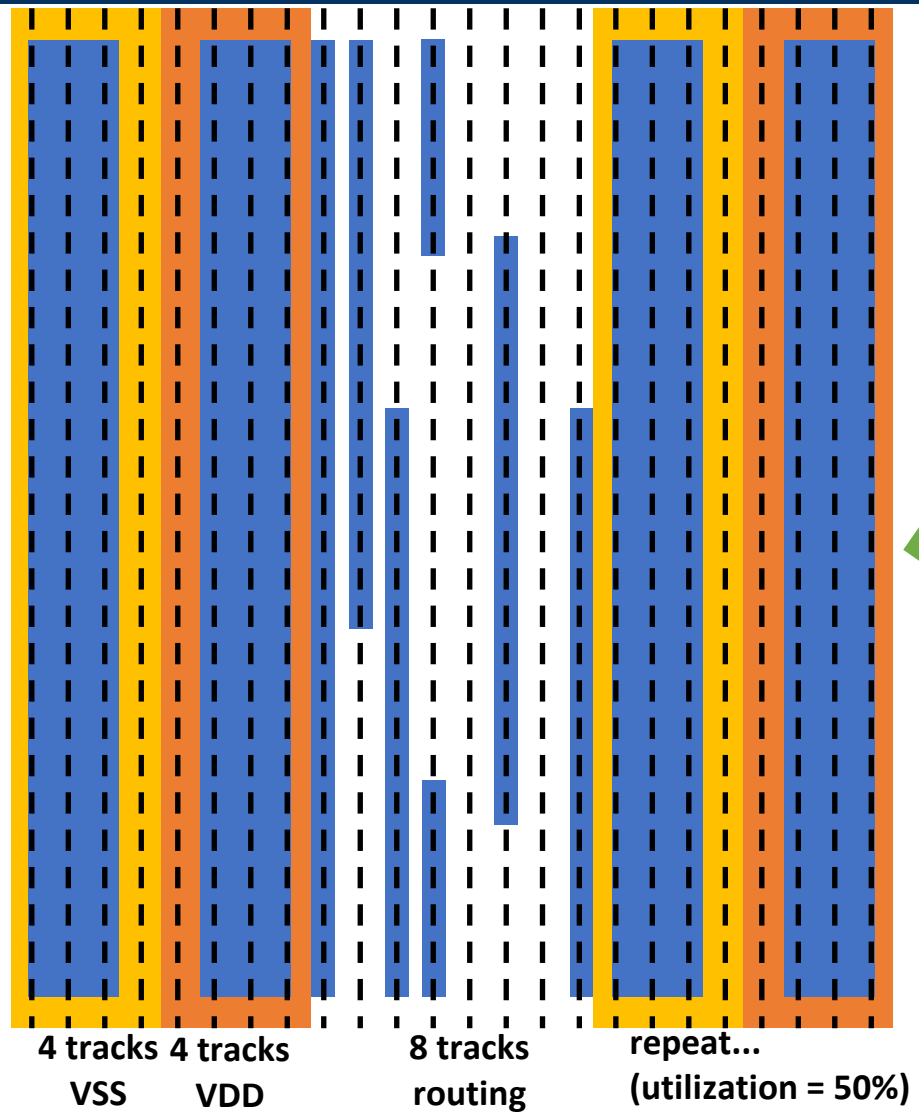
```
par.generate_power_straps_options:
    by_tracks:
        track_width: 4
        power_utilization: 0.5
        strap_layers:
            - M3
            - M4
            - M5
            - M6
            - M7
            - M8
            - M9
par.generate_power_straps_method: by_tracks
par.power_straps_mode: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

Separated Concerns



Berkeley Architecture Research

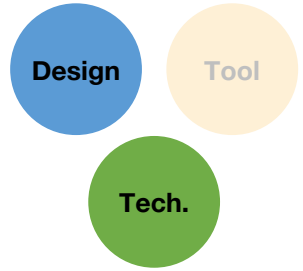
A Quick Example: Power Straps



Route design

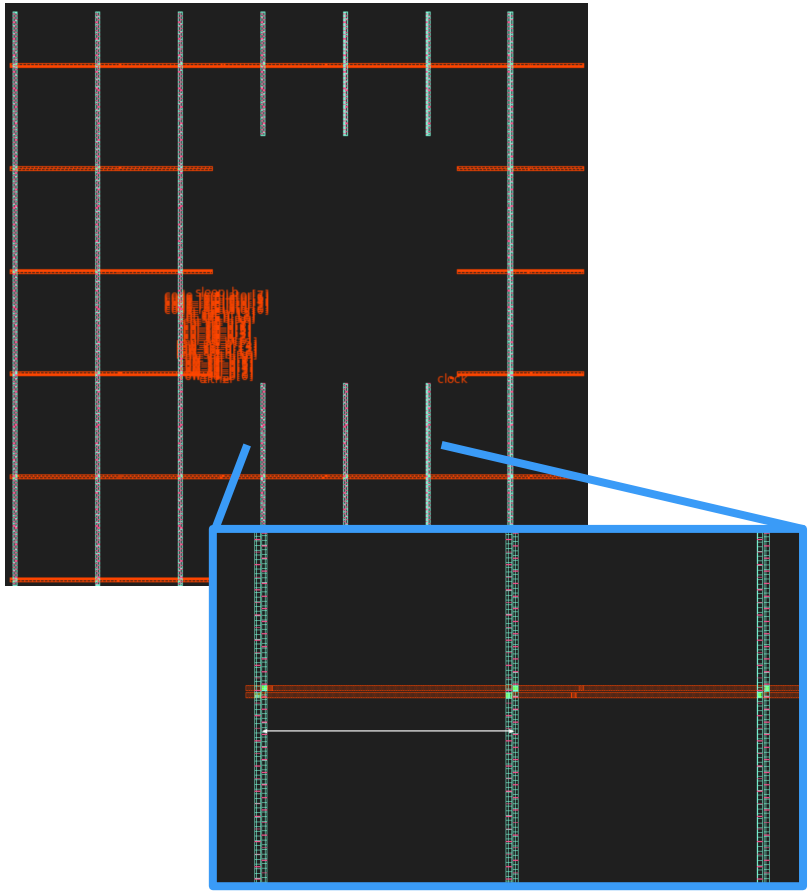
```
par.generate_power_straps_options:
    by_tracks:
        track_width: 4
        power_utilization: 0.5
        strap_layers:
            - M3
            - M4
            - M5
            - M6
            - M7
            - M8
            - M9
par.generate_power_straps_method: by_tracks
par.power_straps_mode: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

Separated Concerns

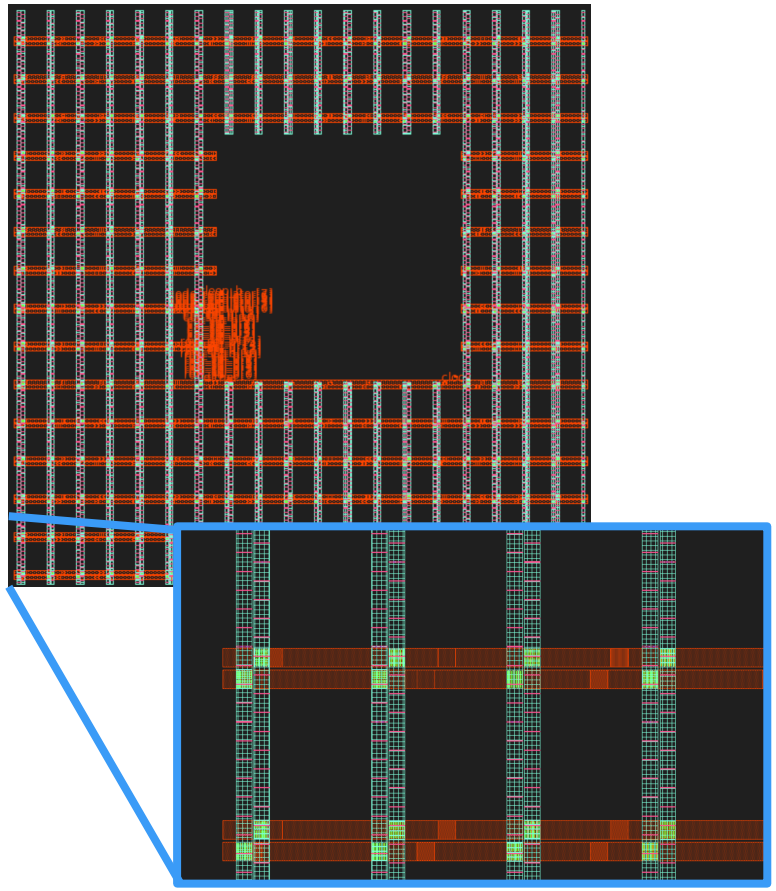


Berkeley Architecture Research

A Quick Example: Power Straps

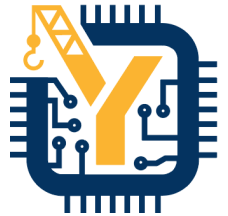


```
power_utilization: 0.05
track_width: 7
```



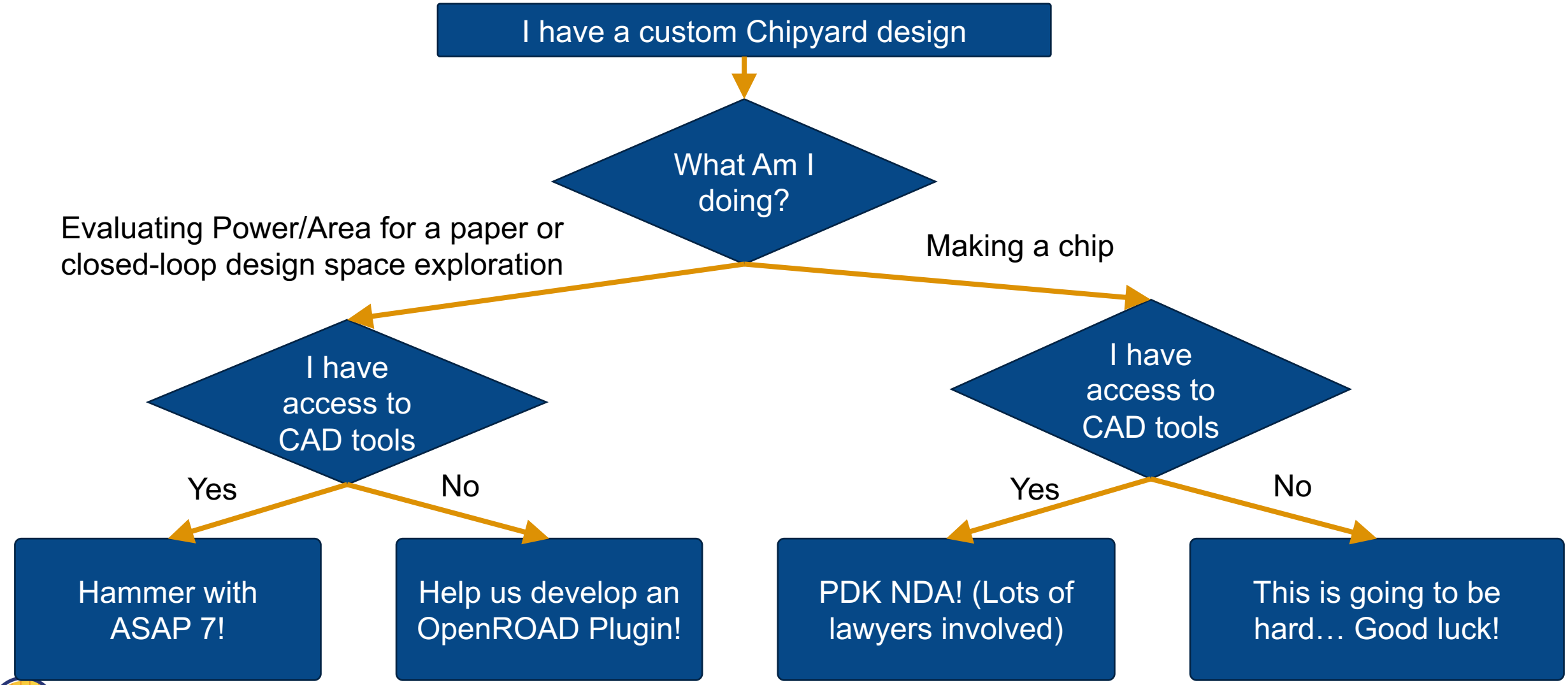
```
power_utilization: 0.25
track_width: 13
```

How to use Hammer

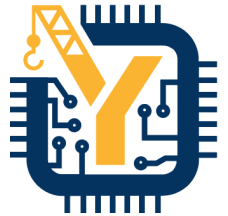


- Hammer can be found under `chipyard/vlsi/`
- Need to obtain tool and tech plug-ins separately dues to NDAs and EULAs
  - Welcome to the world of physical design…
- Priority use case is using proprietary CAD tools to build real chips
  - We are working on open-source alternatives to proprietary CAD tools
- ”Real” technologies need an NDA
  - Some ”Fake” technologies exist to allow example code sharing

Hammer User Decision Diagram



Hammer Demo with ASAP7

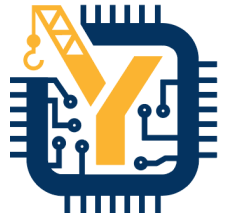


- This demo requires access to Cadence and Mentor CAD tool plugins. Due to licensing issues, access to these is controlled. Contact johnwright@berkeley.edu for more information.

- ASAP7 is a predictive-model PDK developed by Arizona State University. It is free to use for academic use, but requires payment for commercial use. More information is available here: http://asap.asu.edu/asap/

- This demo will walk you through using Hammer without running the CAD tools. Intermediate files will be provided for you to examine.

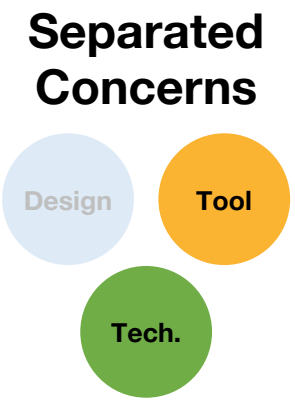
Reusability



We want to get the SHA3 accelerator through a simple physical design process by mixing and reusing plug-ins other people wrote:

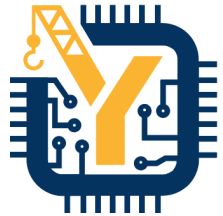
- Technology concerns
  - Example ASAP7 plugin: chipyard/vlsi/hammer/src/hammer-vlsi/technology/asap7
- Tools concerns (under EULA):
  - hammer-cadence-plugins: Genus (synthesis) & Innovus (P&R)
  - hammer-synopsys-plugins: VCS (simulation)
  - hammer-mentor-plugins: Calibre DRC & Calibre LVS

Hammer Plugins

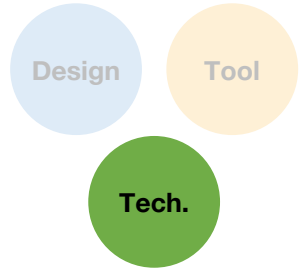


- Reminder: Two types of plugin: Tool and technology
  - <tool> is usually of the format <action>/<name>
  - e.g. par/innovus or syn/dc
- Tool plugins contain:
  - <tool>/defaults.yml – overridable default settings for the tool
  - <tool>/__init__.py – Reusable python methods that implement hammer APIs
- Technology plugins contain:
  - <name>.tech.json – pointers to relevant PDK files
  - defaults.yml – overridable default settings
  - <name>_hooks/<tool>/__init__.py – Reusable python methods

What are we re-using? Tech plugin



Separated Concerns



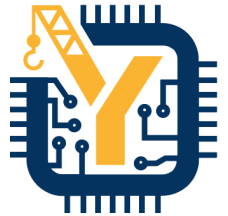

```
technology.core:
  # This key should exist in the stackups list in the tech json
  stackup: "asap7_3Ma_2Mb_2Mc_2Md"
  # This should specify the TOPMOST metal layer the standard
  #   cells use for power rails.
  # Note that this is not usually stackup specific; It is based
  #   on the std cell libraries 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"

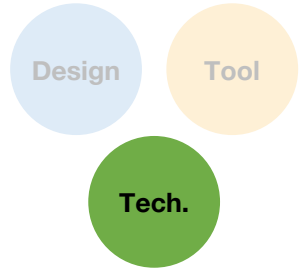
# Set the layer that blocks vias under bumps
vlsi.technology.bump_block_cut_layer: "V9"
```

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

How do I write new tech plugin?



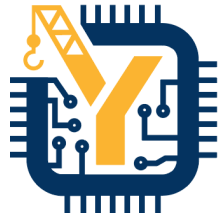
Separated Concerns



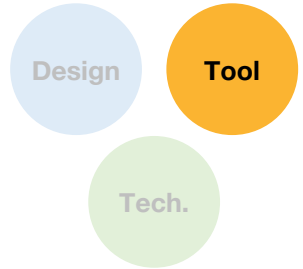
Turn unstructured information about the process technology into a structured representation:

Please refer to the Hammer docs at https://hammer-vlsi.readthedocs.io/en/latest/Technology/index.html

What are we re-using? Tool plugin



Separated Concerns



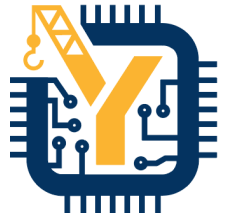

```
def init_environment(self) -> bool:
    self.create_enter_script()
    verbose_append = self.verbose_append

    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)))
    # ...
```

Ask us for access. Some commands are obfuscated so as not to violate EULAs.

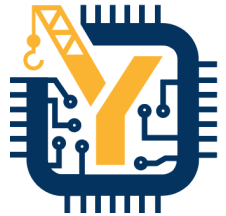
How do I write new CAD plugin?



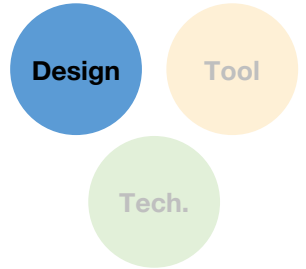
Implement Hammer IR APIs into the specific tool's commands through Reusable python methods

Please refer to the Hammer docs at https://hammer-vlsi.readthedocs.io/en/latest/CAD-Tools/Tool-Plugin-Setup.html

Design Concerns



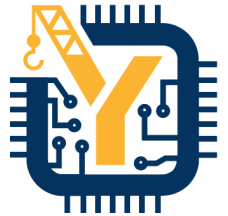
Separated Concerns



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
- Floorplanning
- Clock and power
- Hierarchy assembly
- Boilerplate: selecting the process technology and tools

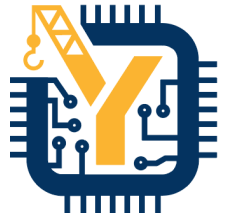
Example Project Structure



- chipyard/vlsi
  - example-vlsi – an extended hammer entry script with added steps (“hooks”)
  - example.yml – project-specific HammerIR
  - extra_libraries – a place for macro collateral (eg. .lib, .lef, .gds)
- For demo purposes, almost everything is in example.yml, but yml files can be separated for further organization
  - env.yml – workplace-specific build and license settings
- To run with the default steps:
  - hammer-vlsi -e env.yml -p example.yml syn
- To run with modified steps:
  - example-vlsi -e env.yml -p example.yml syn

} These will generate the relevant tcl scripts for the EDA tools

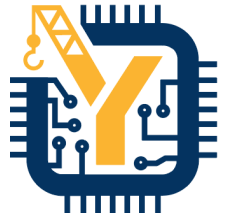
example.yml – Tech Plugin Choice



- Choose the ASAP7 tech plug-in

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

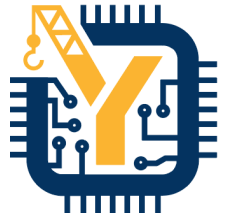
example.yml – Tool Plugin Choice



- From hammer-cadence-plugins:
  - Genus 18.13 for Synthesis
  - Innovus 18.1 for PnR
- For hammer-mentor-plugins:
  - Calibre for DRC and LVS
- Note: Verify the tools installation path
- Note: ASAP7 cannot use Innovus version >18.1 (ISRs also don't work)

```
# Tool options. Replace with your tool plugin of choice.
# Genus options
vlsi.core.synthesis_tool: "genus"
vlsi.core.synthesis_tool_path: ["hammer-cadence-
plugins/synthesis"]
vlsi.core.synthesis_tool_path_meta: "append"
synthesis.genus.version: "1813"
# Innovus options
vlsi.core.par_tool: "innovus"
vlsi.core.par_tool_path: ["hammer-cadence-plugins/par"]
vlsi.core.par_tool_path_meta: "append"
par.innovus.version: "181"
par.innovus.design_flow_effort: "standard"
par.inputs.gds_merge: true
# Calibre options
vlsi.core.drc_tool: "calibre"
vlsi.core.drc_tool_path: ["hammer-mentor-plugins/drc"]
vlsi.core.lvs_tool: "calibre"
vlsi.core.lvs_tool_path: ["hammer-mentor-plugins/lvs"]
```

example.yml – Power and Clocking



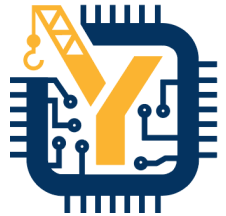
- Specify clock signal and constraints

```
# 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"
```

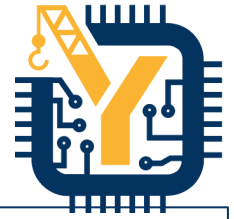
example.yml – Placement Constraints



- Placement constraints
- Top-level is Sha3AccelwBB
  - Highlighted in yellow
  - 300x300um with 1.08um margin on bottom (for DRC). highlighted blue
- Dummy hardmacro (“dco”) placed at (108, 108), no flipping
  - Highlighted in green

```
vlsi.inputs.placement_constraints:
  - path: "Sha3AccelwBB"
    type: toplevel
    x: 0
    y: 0
    width: 300
    height: 300
    margins:
      left: 0
      right: 0
      top: 0
      bottom: 0
  - path: "Sha3AccelwBB/dco"
    type: hardmacro
    x: 108
    y: 108
    width: 128
    height: 128
    orientation: r0
    top_layer: M9
  - path: "Sha3AccelwBB/place_obs_bottom"
    type: obstruction
    obs_types: ["place"]
    x: 0
    y: 0
    width: 300
    height: 1.08
```

example.yml – Analog/Hard IP

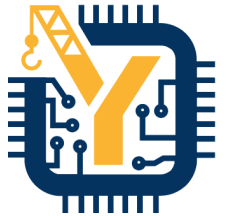


- Extra Libraries – Hard IP (analog blog, third party IP, etc.)
  - Specify the collateral files for each corner
- Specify “physical only” cells
  - Cells with no behavioral or other analysis details
- Include a “DCO” in the demo
  - Digitally-Controlled Oscillator

```
# Paths to extra libraries
vlsi.technology.extra_libraries_meta: ["append", "deepsubst"]
vlsi.technology.extra_libraries:
  - library:
      nldm liberty file_deepsubst_meta: "local"
      nldm liberty file:
"extra_libraries/example/ExampleDCO_PVT_0P63V_100C.lib"
      lef file_deepsubst_meta: "local"
      lef file: "extra_libraries/example/ExampleDCO.lef"
      gds file_deepsubst_meta: "local"
      gds file: "extra_libraries/example/ExampleDCO.gds"
      corner:
        nmos: "slow"
        pmos: "slow"
        temperature: "100 C"
      supplies:
        VDD: "0.63 V"
        GND: "0 V"

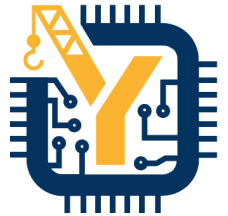
…
# Because the DCO is a dummy layout,
# we treat it as a physical-only cell
par.inputs.physical_only_cells_mode: append
par.inputs.physical_only_cells_list:
  - ExampleDCO
```

“Hooks”



- The “Magic TCL scripts” aren’t going away soon
  - A lot of expertise captured in these scripts
- Hammer is still under development
  - Least mature Chipyard project!
  - Need to earn user confidence
  - **However, used in many real tapeouts**
- Hooks enable insertion of custom python or TCL scripts within the Hammer-generated flow
  - ”Escape hatches”
  - Cleanly allows real-time development and workarounds
  - Allows prototyping of future APIs

“Hooks” - Example



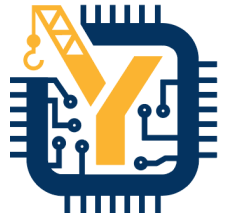
- Example of a technology-supplied hook
  - ASAP7 runs a Python script from Innovus to scale down post-P&R GDS
  - script_text is provided by the scale_gds_script method in ASAP7’s __init__.py
  - There is no equivalent Hammer API for such scaling
  - Inserted post write_design
  - Other examples: Custom fiducial placement, endcap cell placement

```
def scale_final_gds(x: hammer_vlsi.HammerTool) -> bool:
    x.append(```
      # Write script out to a temporary file and execute it
      set fp [open "{script_file}" "w"]
      puts -nonewline $fp "{script_text}"
      close $fp

python3 {script_file}
      ```.format(script_text=x.technology.scale_gds_script(x.output_gds_filename),
      script_file=os.path.join(x.run_dir, "gds_scale.py")))
    return True
```


```

example-vlsi



- Project-specific entry script extending hammer-vlsi
  - Inserts/modifies/removes existing Hammer steps with “hooks”
  - These are python methods that emit TCL

```
class ExampleDriver(CLIDriver):
    def get_extra_par_hooks(self) -> List[HammerToolHookAction]:
        extra_hooks = [

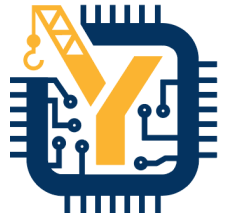
            # make_pre_insertion_hook will execute the custom hook before the specified step
            # hammer_vlsi.HammerTool.make_pre_insertion_hook("route_design", example_add_fillers),

            # 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"),

            # This is an example of a technology-supplied hook
            hammer_vlsi.HammerTool.make_post_insertion_hook("write_design", scale_final_gds)
        ]
        return extra_hooks
```

example.yml – Flow Make File



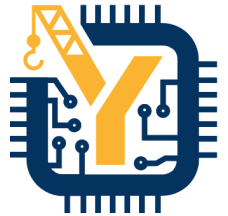
- Generate Makefiles with the relevant Hammer targets

```
# Generate Make include to aid in flow
vlsi.core.build_system: make
```

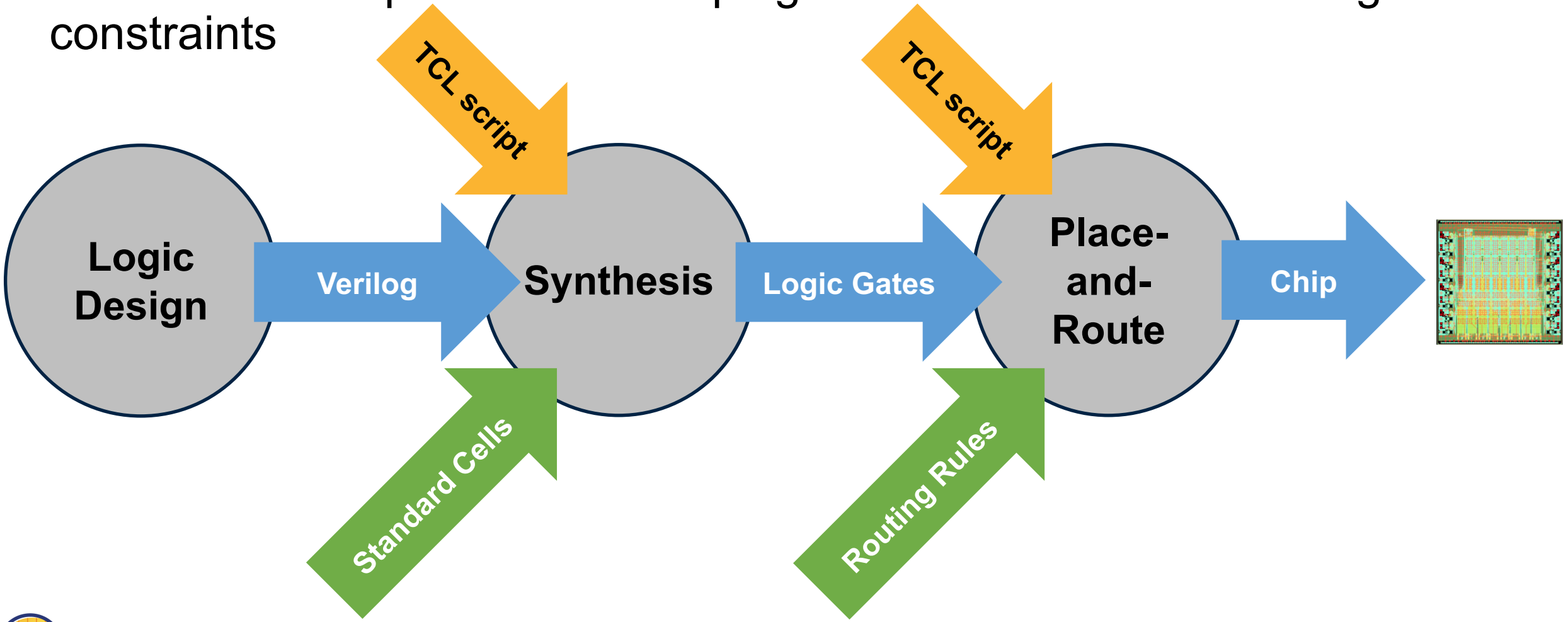
- If you have access to a compute infrastructure (e.g. LSF)
  - In your .yml, you can set the **vlsi.submit.submit_command** to **lsf**
  - This will allow the submission of heavy and/or paralle jobs to a compute cluster. For example **make –j drc lvs**

- https://hammer-vlsi.readthedocs.io/en/latest/Hammer-Use/Buildfile.html

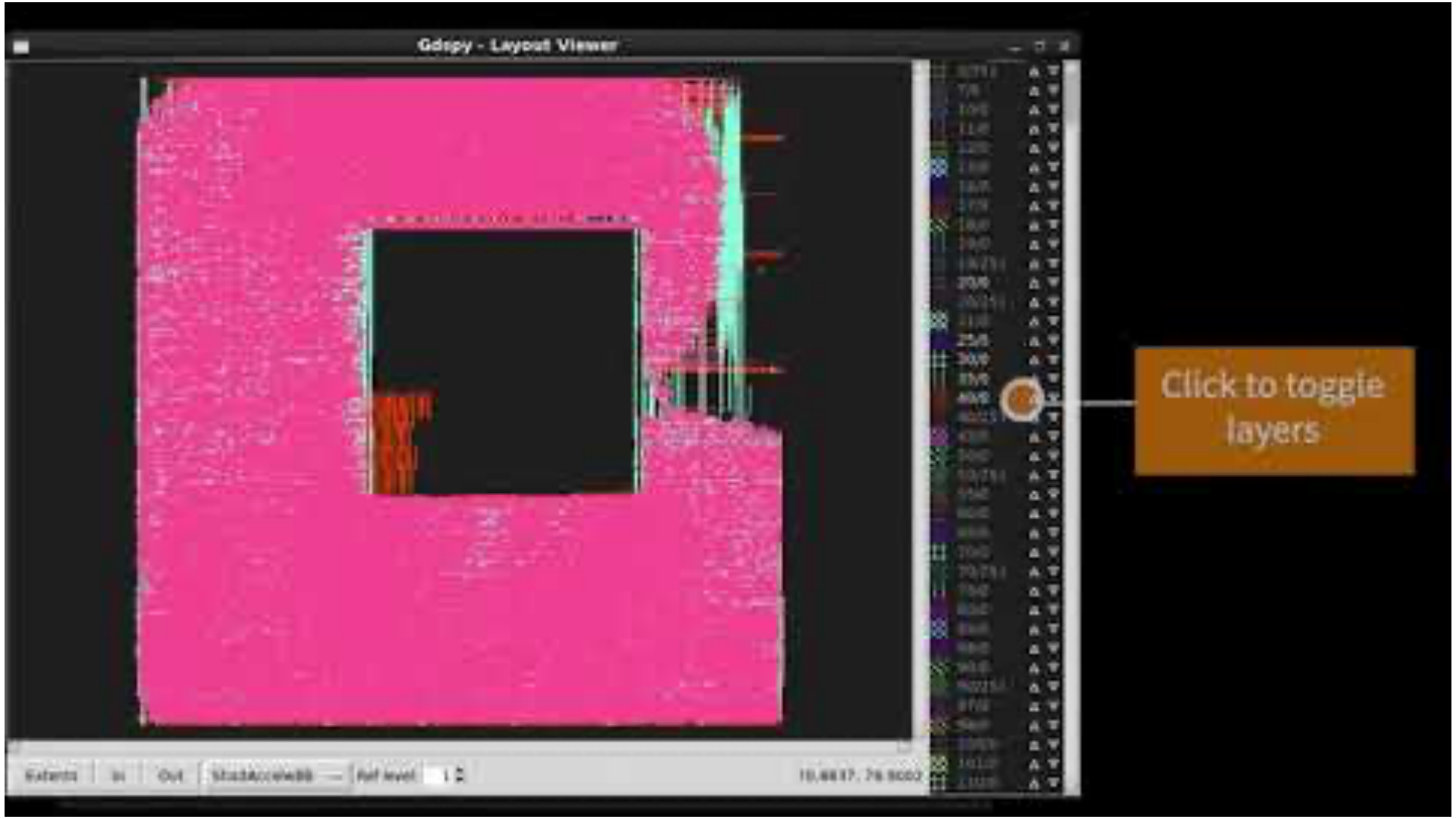
Getting closer to the advertised flow



- After we've setup the re-usable plug-ins and static custom design constraints

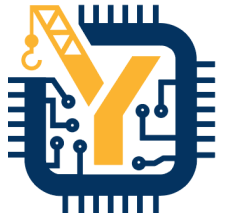


VLSI Flow Demo Video



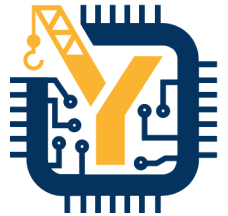
https://youtu.be/TiXeocDWdFA

Future of Hammer

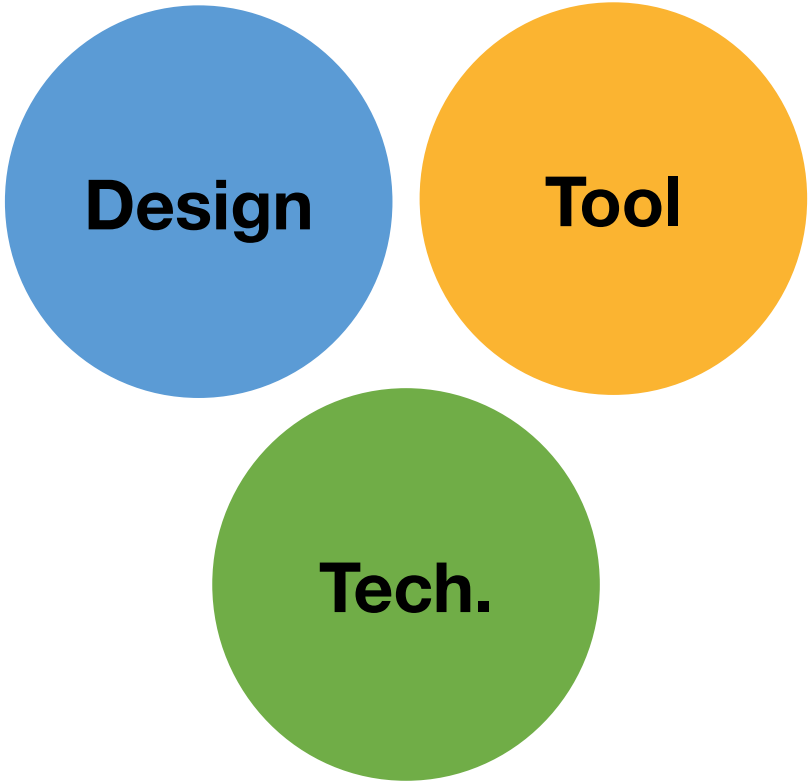


- HammerIR lends itself well to generators
  - Python floorplanning scripts to auto-calculate placement constraints
  - WIP: Scala API so that physical design data can be tied to Chisel generator
  - WIP: Composable floorplans using Aspect-Oriented Chisel
  - WIP: Check floorplans for hierarchical alignment (grids & pitches)
  - WIP: Generate clock constraints from FIRRTL
- Reconfiguring Chisel designs based on physical design feedback
- Additional signoff tools
  - IR drop, dynamic power analysis, LEC
- More physical design APIs
  - Pin placement, I/O timing budgeting

Summary



- Physical Design is hard—There are good reasons why most of the people in this room don't do 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
  - SHA3 demo



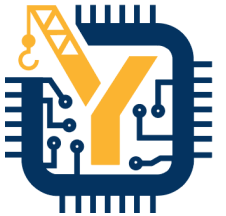
Acknowledgements & Questions

Thanks to the hammer development team

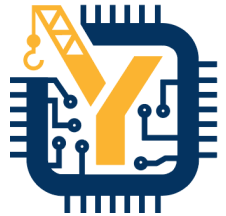
Edward Wang, Colin Schmidt, Harrison Liew, Daniel Grubb

Questions?

Backup Slides



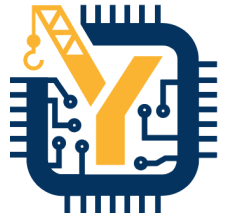
Running Synthesis



- Run **make syn** (Reminder: don't actually run this. This is a demo)
  - Inputs: RTL design (Verilog from FIRRTL)
  - Outputs: technology-mapped gate-level netlist

- Example results (for demo purposes) in ~/hammer-asap7-demo-master
  - Example Hammer build directory: `build-example/syn-rundir`
  - Example output gate-level netlist: `build-example/syn-rundir/Sha3AccelwBB.mapped.v`
  - Example output HammerIR for the next steps `build-example/syn-rundir/syn-output.json`

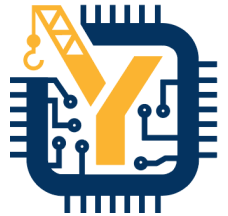
Running Place-and-Route



- Run **make par** (Reminder: don't actually run this. This is a demo)
  - Inputs: synthesized netlist
  - Outputs: routed gate-level netlist, GDSII (mask data)

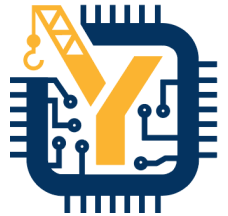
- Example results (for demo purposes) in ~/hammer-asap7-demo-master
  - Example Hammer build directory: `build-example/par-rundir`
  - Example output netlists (with and without power nets):
    `build-example/par-rundir/Sha3AccelwBB.lvs.v`
    `build-example/par-rundir/Sha3AccelwBB.sim.v`
  - Example output HammerIR for the next steps
    `build-example/par-rundir/par-output.json`

Viewing your “Chip”



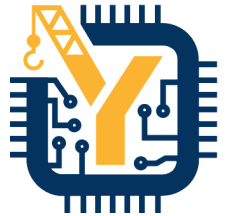
- Navigate to chipyard/vlsi/ and run:
  - ./view_gds.py ~/hammer-asap7-demo-master/build-example/par-rundir/Sha3AccelwBB.gds
- This will open up the GDS in a python-based layout viewer
  - Note: There are much, much better proprietary alternatives to this
- By default, this will only display metals 2 through 4 and their vias
  - Feel free to adjust in the view_gds.py script!
  - Note: more layers take more time to draw
  - Note: The standard cell layouts have been removed, so you’ll only see routing

Hammer Philosophy



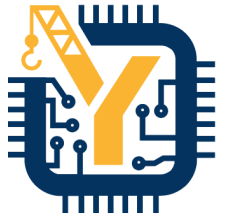
- Separate fundamental design decisions into reusable methods
- Always allow overrides
  - Designer can always write TCL to perform a task
- Use reasonable defaults
  - Often a baseline is good enough – it can be modified later
- Use agile software development
  - Prototype features in project repos
  - Upstream more generic versions of features to core hammer

Hammer Components



- A few submodules in chipyard/vlsi/
- Hammer
  - Contains the backend framework and features described in this tutorial
- Hammer Tool Plugins
  - Contains tool-specific implementations for the Hammer APIs
  - hammer-cadence-plugins: Genus (synthesis) & Innovus (P&R)
  - hammer-synopsys-plugins: VCS (simulation)
  - hammer-mentor-plugins: Calibre DRC & Calibre LVS
- Hammer Tech Plugins
  - Example ASAP7 plugin: chipyard/vlsi/hammer/src/hammer-vlsi/technology/asap7

The Make Infrastructure

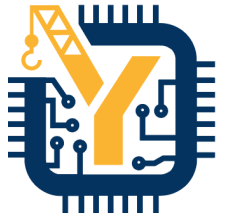


- Look at chipyard/vlsi/Makefile
  - We'll use **make** commands that wrap the longer Hammer commands

- Run **make buildfile** (Reminder: don't actually run this)
  - Note: This is run implicitly, but it's useful to do it explicitly
  - This will elaborate RTL if it has not been elaborated already
  - Based on the config, targets will be generated into build/hammer.d
  - There are steps prefixed with **redo-** that allow the user to bypass dependencies

- Navigate to ~/hammer-asap7-demo-master/build-example
  - Open hammer.d and look at the hammer driver commands it is running

Advanced Usage



- Finer control over steps to run
  - E.g. --from_step, --to_step, --only_step for fast iteration
  - Would like to have concept of database state to make this robust

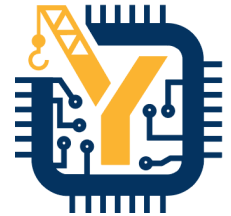
- Make-based build infrastructure to manage dependencies

Specify The Hierarchical Example

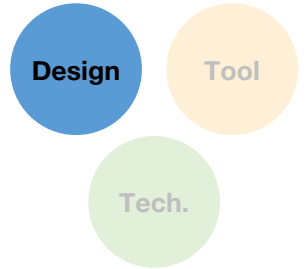

```
top_module: "Top"
```

```
manual_placement_constraints:
- Top
  - {type: toplevel, … }
```

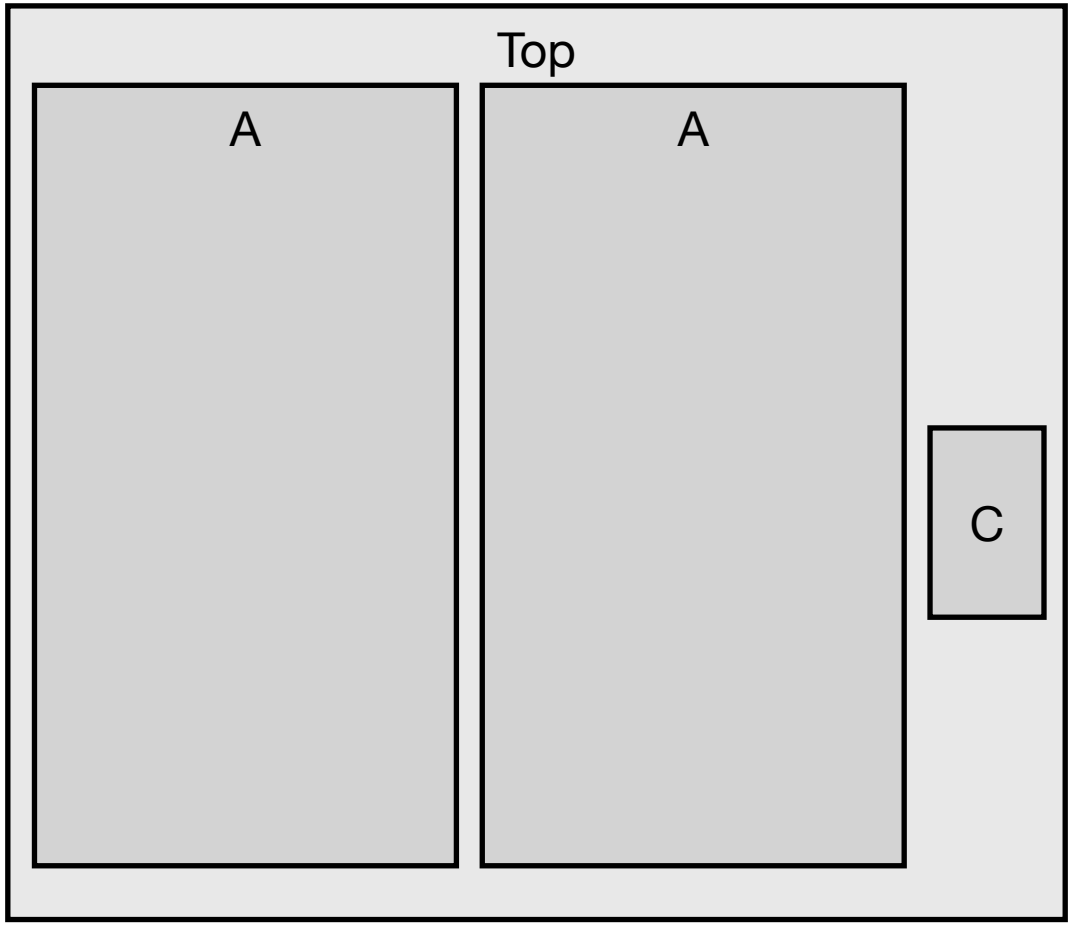
```
- vlsi.inputs.clocks: […]
```



Separated Concerns



Hierarchical Example




```
top_module: "Top"
manual_modules:
- Top
  - A
  - C


manual_placement_constraints:
- Top
  - {type: toplevel, … }
  - {type: hierarchical, master: A,
     path: Top/A_0, … }
- A
  - {type: toplevel, … }


constraints:
- Top
  - vlsi.inputs.clocks: […]
- A
  - vlsi.inputs.pin.assignments: […]
```

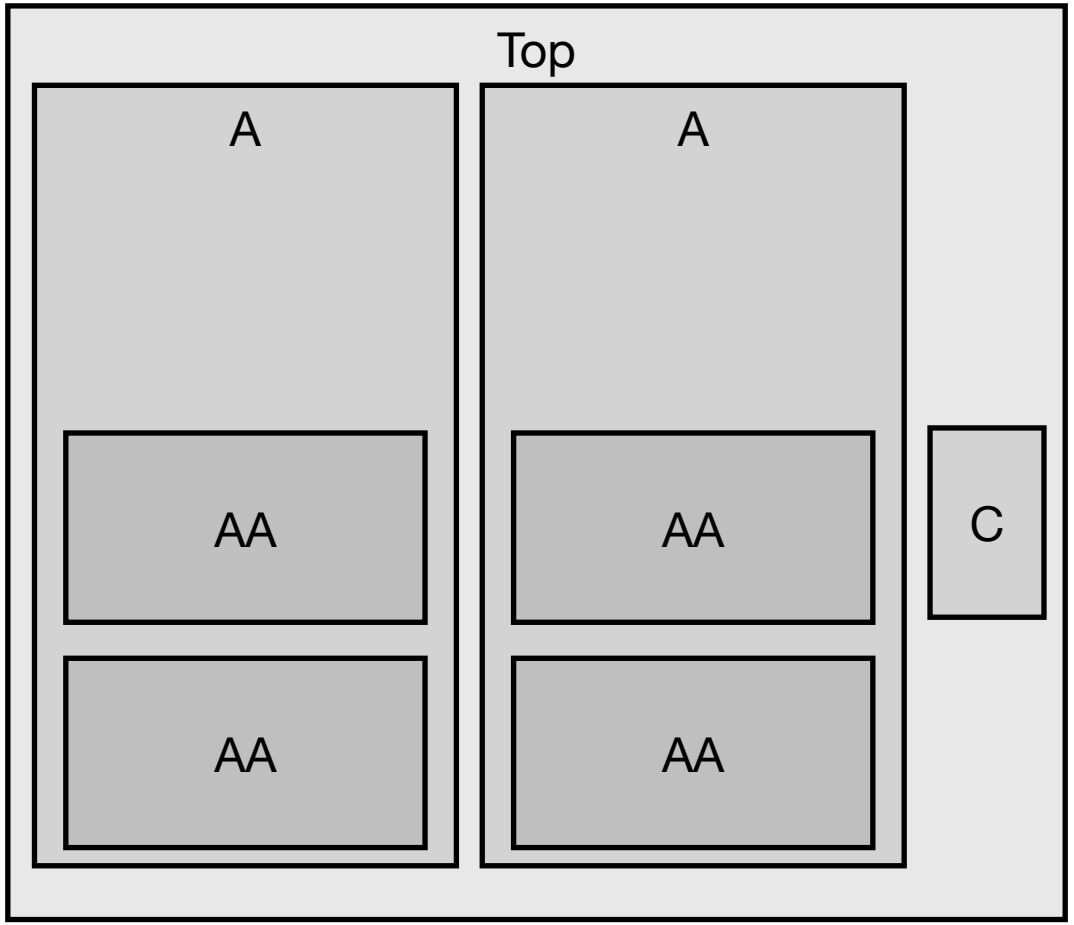
Separated Concerns

Design

Tool

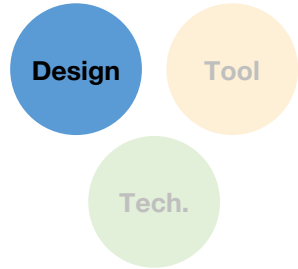
Tech.

Hierarchical Example

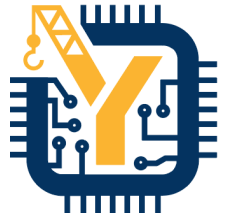



```
top_module: "Top"
manual_modules:
- Top
  - A
  - C
- A
  - AA
manual_placement_constraints:
- Top
  - {type: toplevel, … }
  - {type: hierarchical, master: A,
     path: Top/A_0, … }
- A
  - {type: toplevel, … }
  - {type: hierarchical, master: AA,
     path: A/AA_0, … }
constraints:
- Top
  - vlsi.inputs.clocks: […]
- A
  - vlsi.inputs.pin.assignments: […]
```

Separated Concerns



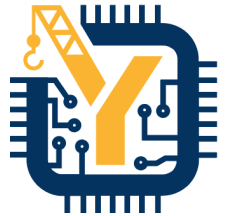
example.yml – Pin Placement



- SHA3 block integrated in a hierarchical design flow
- Need pins to connect to levels higher in the hierarchy
- semi_auto pin placement uses the CAD tool's default pin distribution
  - 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"}
]
```

example.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`

```
# 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
    track_start: 10
    power_utilization: 0.05
    power_utilization_M8: 1.0
    power_utilization_M9: 1.0
```