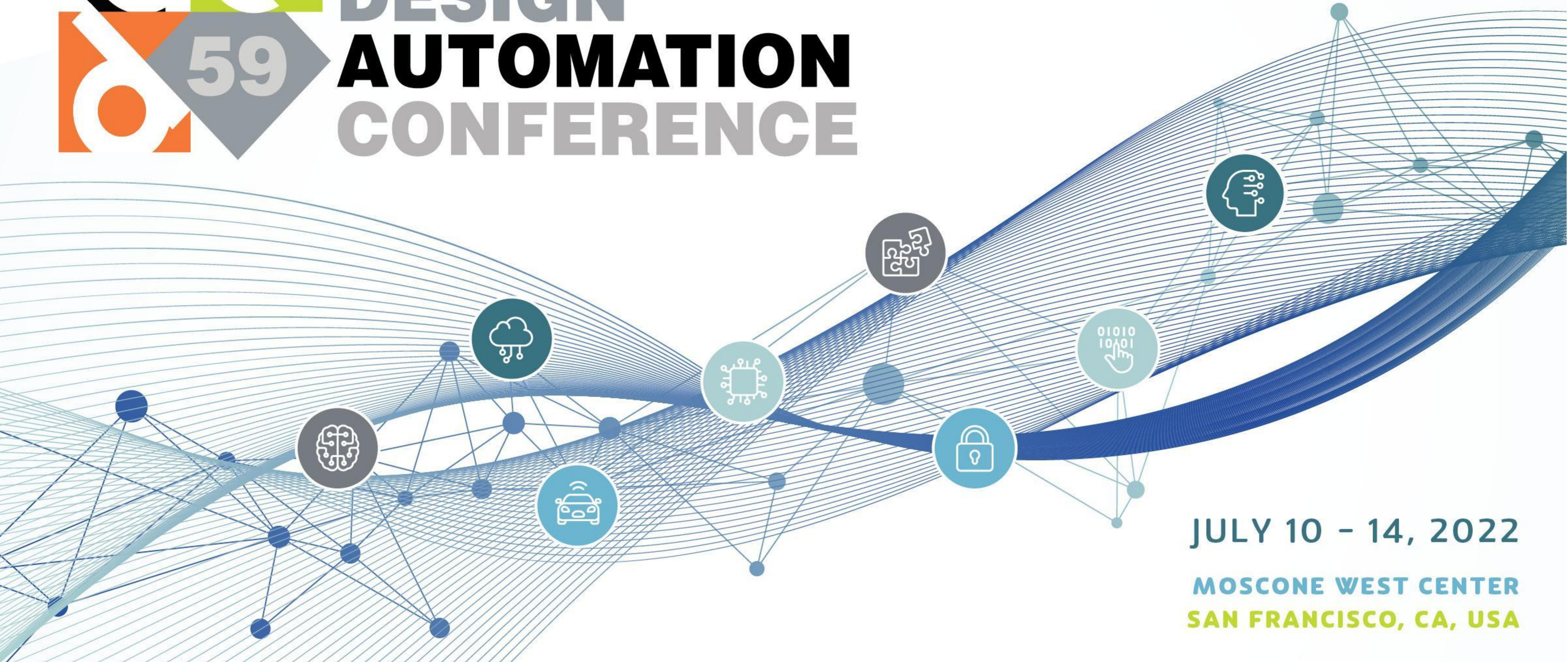
DESIGN AUTOMATION CONFERENCE

59



JULY 10 - 14, 2022

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

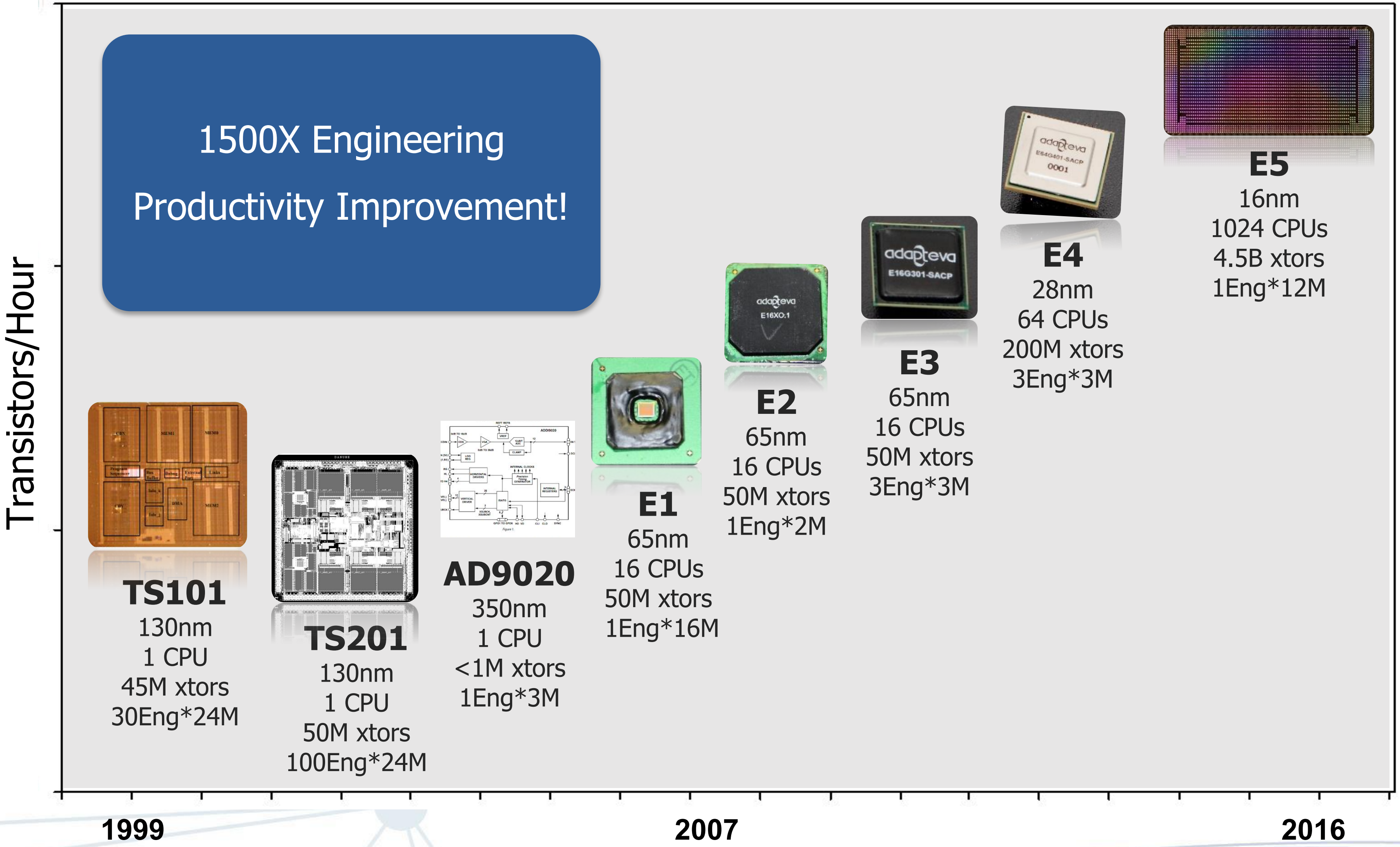
DESIGN AUTOMATION CONFERENCE

A Distributed Approach to Silicon Compilation

Andreas Olofsson, William Ransohoff, Noah Moroze
Zero ASIC Corporation
Cambridge, MA
{andreas,will,noah}@zeroasic.com

SiliconCompiler

My Background: *25 Years of Pushing Polygons*



1500X Engineering Productivity Improvement!

TS101
130nm
1 CPU
45M xtors
30Eng*24M

TS201
130nm
1 CPU
50M xtors
100Eng*24M

AD9020
350nm
1 CPU
<1M xtors
1Eng*3M

E1
65nm
16 CPUs
50M xtors
1Eng*16M

E2
65nm
16 CPUs
50M xtors
1Eng*2M

E3
65nm
16 CPUs
50M xtors
3Eng*3M

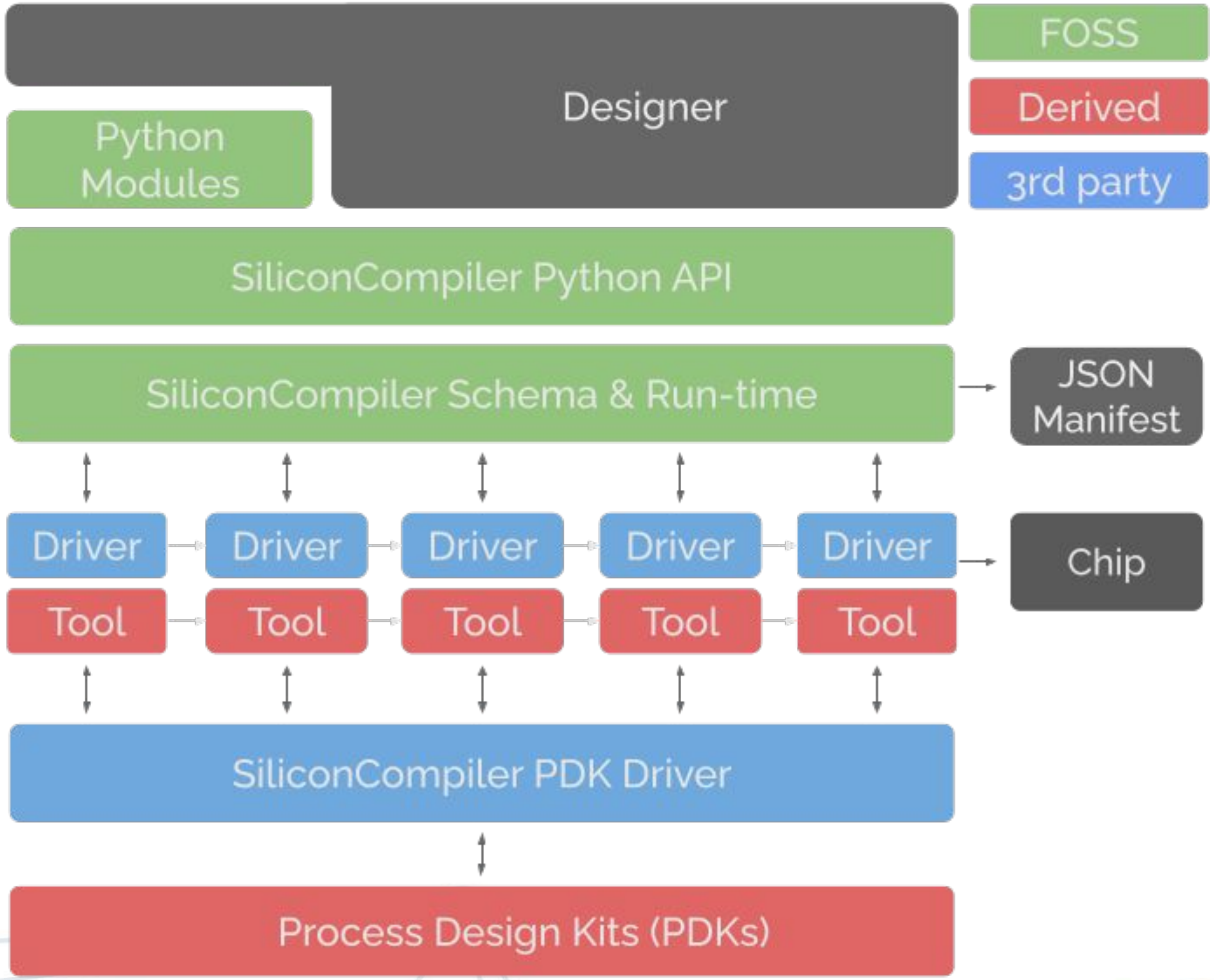
E4
28nm
64 CPUs
200M xtors
3Eng*3M

E5
16nm
1024 CPUs
4.5B xtors
1Eng*12M

LESSONS LEARNED:

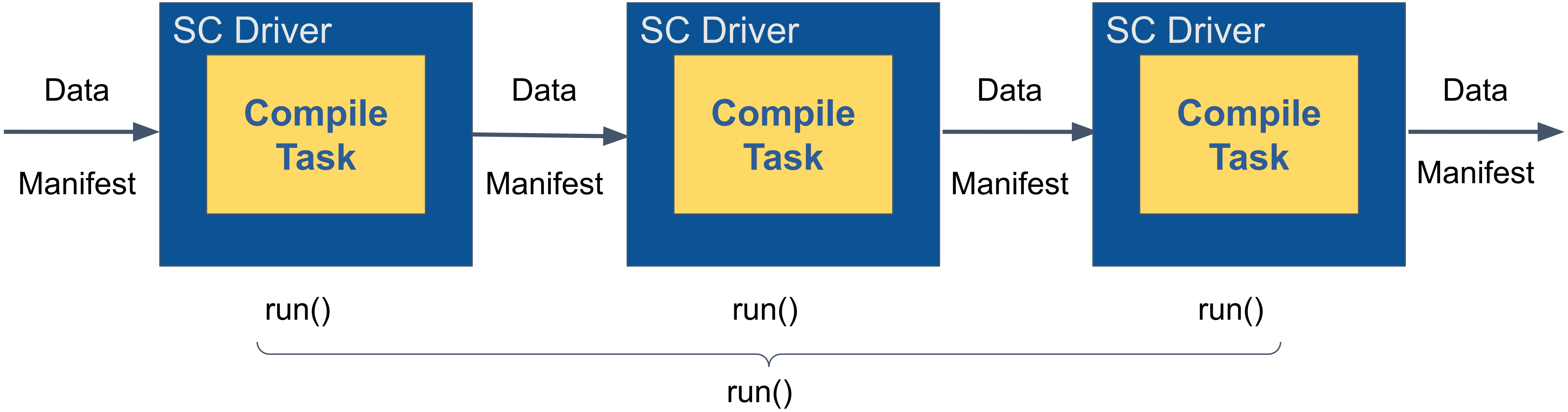
- Make PNR part of the arch loop
- Design for Layout (DFL)
- Design for EDA (DFE)
- Fight the FUD (not that hard!)
- Automate everything

SiliconCompiler: *A modular build system for hardware*


Designer
FOSS
Derived
3rd party
Python Modules
SiliconCompiler Python API
SiliconCompiler Schema & Run-time
JSON Manifest
Driver
Driver
Driver
Driver
Driver
Tool
Tool
Tool
Tool
Tool
Chip
SiliconCompiler PDK Driver
Process Design Kits (PDKs)

- "Make for hardware"
- Standardized build schema (json)
- Python OO API
- Flowgraph based execution
- Developed with cloud first approach
- Automated actions/metrics tracking
- Built for commercial AND open source ASIC/FPGA tools.
- **SRC**: https://github.com/siliconcompiler
- **DOCS: https://docs.siliconcompiler.com**

Basic Operation: *Configure, Run, Observe*

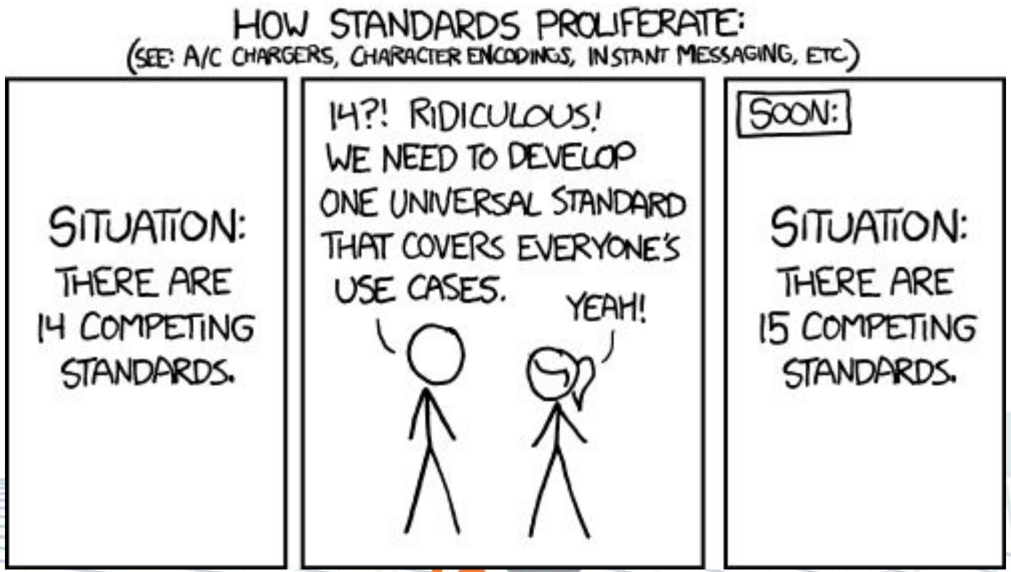


1. Compilation tasks (ie EDA tools) wrapped with unified driver interfaces
2. Runtime manifest defines the “what, how, when, who, why” of compilation.
3. Sequence of tasks run based on static flowgraph defined in manifest.
4. Every configuration and tracked runtime metric reflected in the manifest.

The Manifest: *An Open "CAD Standard"*

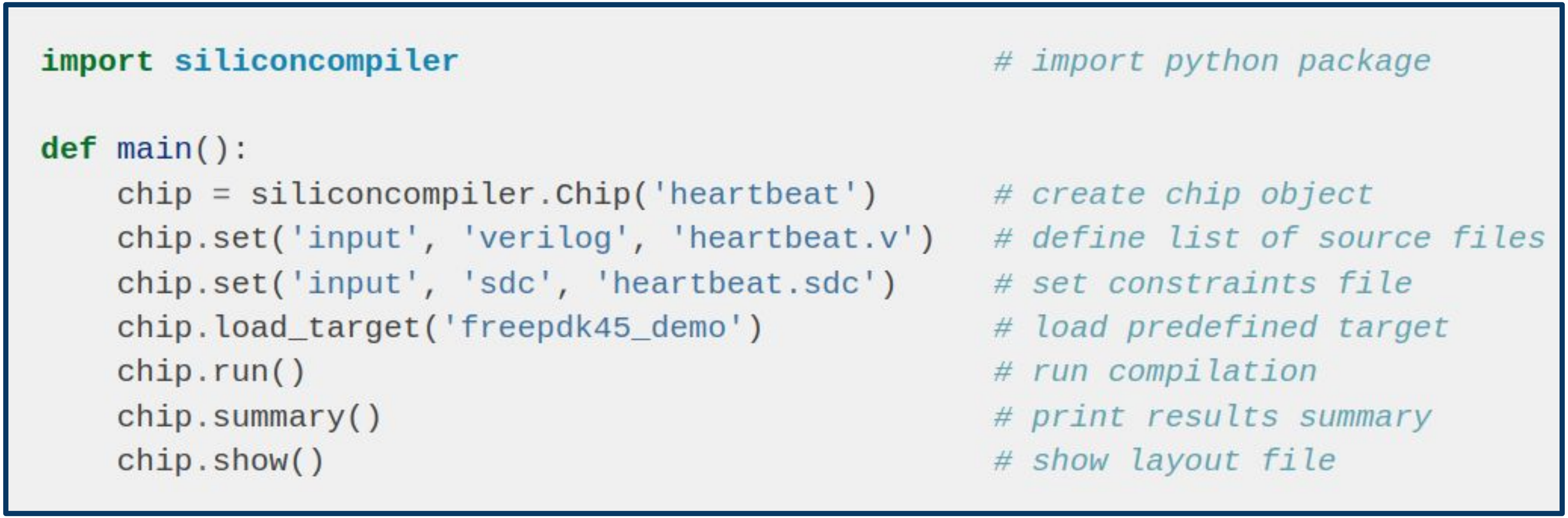
Group	Parameters	Examples
asic	46	diearea , maxfanout,
intput/output	2	sdc, rtl, def, …
constraint	8	PVT, SDC, checks, …
options	50	loglevel , skip, optmode, path, …
unit	10	Time, voltage, current, ...
pdk	50	Runset, stackup, process,...
tool	29	Options, exename, license,...
flowgraph	9	Inputs, weights, goals
checklist	9	Rationale, criteria,
metric	45	Setupwns, errors, warnings, …
datasheet	39	Abs voltage, setup, hold
package	32	Dependency, author, …

- A unified compilation manifest
- Standardize all common settings
- Bypass parameters for "one-offs"
- Validated with 5 PDKs, 35+ EDA tools, ASIC/FPGA/HLS flows


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Source: https://xkcd.com/

Designer View: *Configure, Run, Observe*



```python
import siliconcompiler                            # import python package

def main():
    chip = siliconcompiler.Chip('heartbeat')      # create chip object
    chip.set('input', 'verilog', 'heartbeat.v')   # define list of source files
    chip.set('input', 'sdc', 'heartbeat.sdc')     # set constraints file
    chip.load_target('freepdk45_demo')            # load predefined target
    chip.run()                                    # run compilation
    chip.summary()                                # print results summary
    chip.show()                                   # show layout file
```

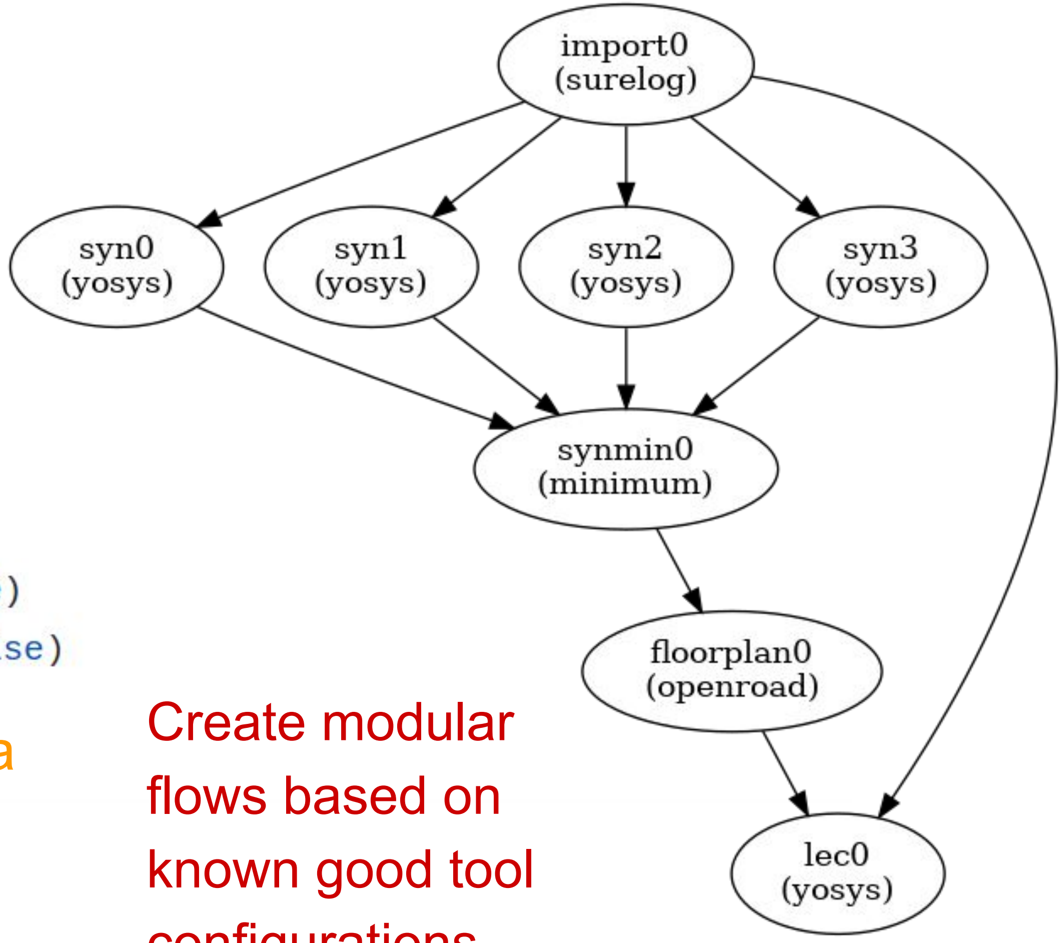
EDA View: *Tools and Flows*

setup(): Capture all TCL, cmdline, ENV settings

```
# Standard Setup
chip.set('tool', tool, 'exe', 'yosys')
chip.set('tool', tool, 'vswitch', '--version')
chip.set('tool', tool, 'version', '>=0.13', clobber=False)
chip.set('tool', tool, 'format', 'tcl', clobber=False)
chip.set('tool', tool, 'option', step, index, '-c', clobber=False)
chip.set('tool', tool, 'refdir', step, index, refdir, clobber=False)
```

post_process(): Convert per tool free form text into data

```
chip.set('metric', step, index, 'errors', 0, clobber=True)
with open(step + ".log") as f:
    for line in f:
        area = re.search(r'Chip area for module.*\:\s+(.*)', line)
        cells = re.search(r'Number of cells\:\s+(.*)', line)
        if area:
            chip.set('metric', step, index, 'cellarea', round(float(area.group(1)),2), clobber=True)
        elif cells:
            chip.set('metric', step, index, 'cells', int(cells.group(1)), clobber=True)
```

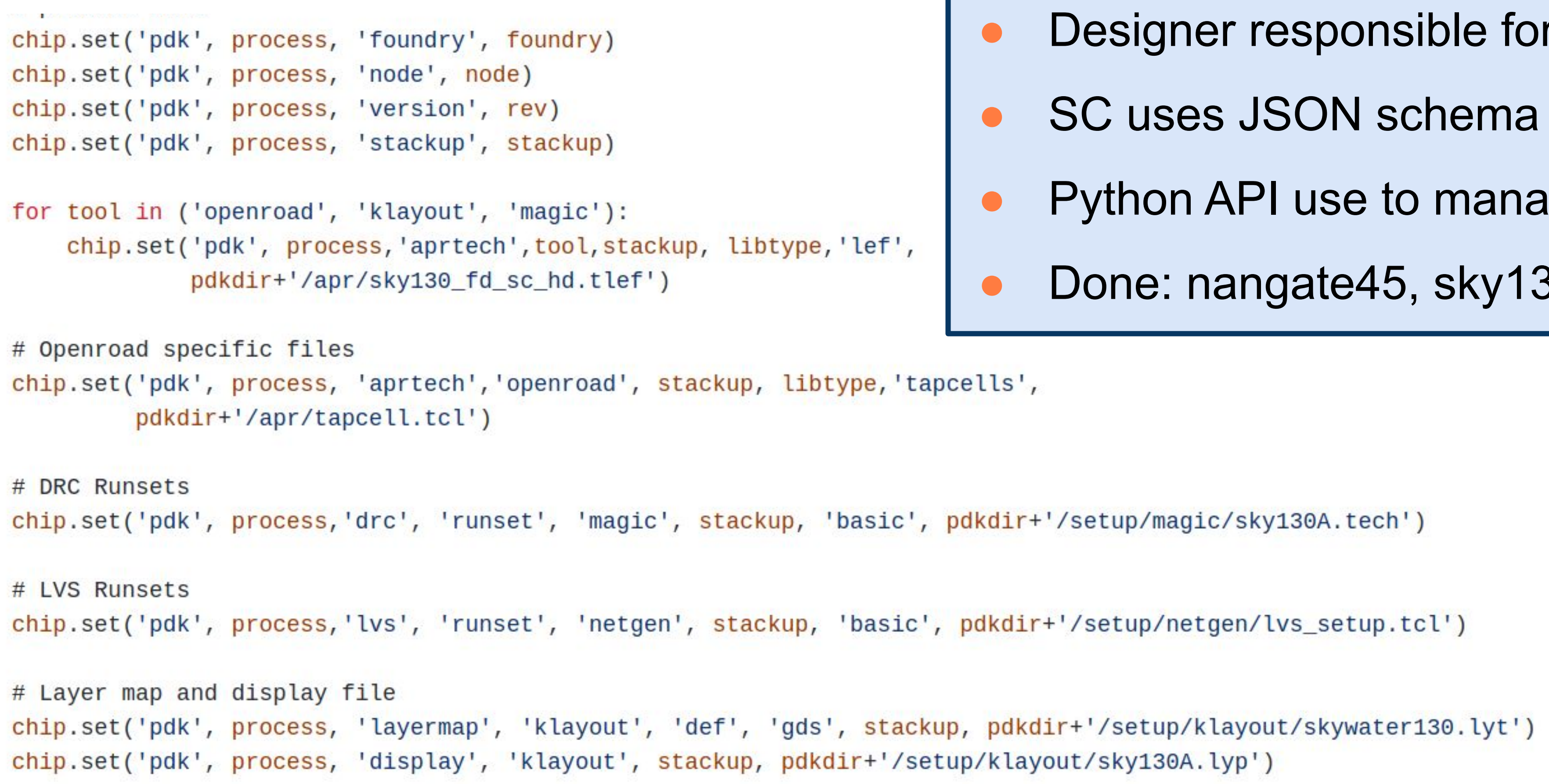


Create modular flows based on known good tool configurations. “mix and match”)

Foundry View: *We can do better than “tarballs”!*

- Modern PDKs shipped as tar balls with 1000’s of files.
- Designer responsible for learning structure!
- SC uses JSON schema as golden PDK file manifest
- Python API use to manage setup complexity
- Done: nangate45, sky130, asap7, gf12lp, intel16

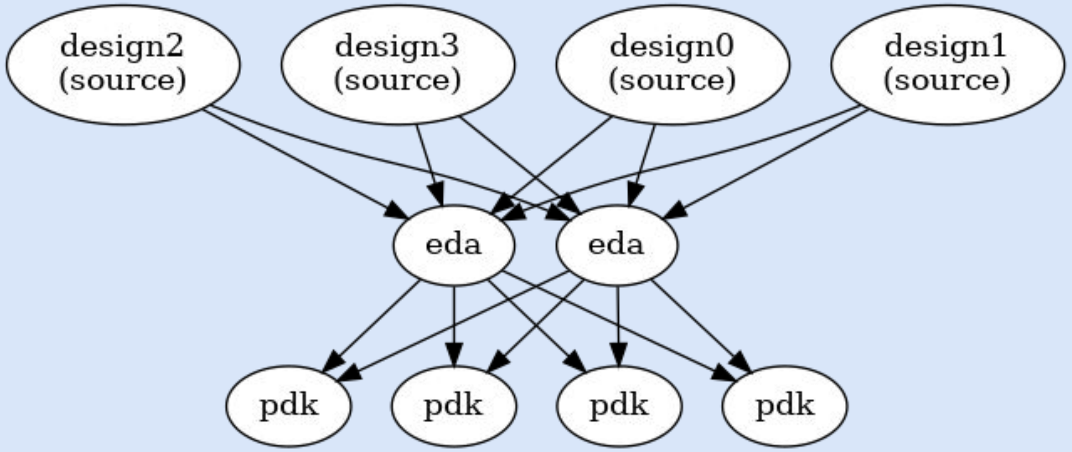
setup(): # skywater130.py



```
chip.set('pdk', process, 'foundry', foundry)
chip.set('pdk', process, 'node', node)
chip.set('pdk', process, 'version', rev)
chip.set('pdk', process, 'stackup', stackup)

for tool in ('openroad', 'klayout', 'magic'):
    chip.set('pdk', process,'aprtech',tool,stackup, libtype,'lef',
             pdkdir+'/apr/sky130_fd_sc_hd.tlef')

# Openroad specific files
chip.set('pdk', process, 'aprtech','openroad', stackup, libtype,'tapcells',
         pdkdir+'/apr/tapcell.tcl')

# DRC Runsets
chip.set('pdk', process,'drc', 'runset', 'magic', stackup, 'basic', pdkdir+'/setup/magic/sky130A.tech')

# LVS Runsets
chip.set('pdk', process,'lvs', 'runset', 'netgen', stackup, 'basic', pdkdir+'/setup/netgen/lvs_setup.tcl')

# Layer map and display file
chip.set('pdk', process, 'layermap', 'klayout', 'def', 'gds', stackup, pdkdir+'/setup/klayout/skywater130.lyt')
chip.set('pdk', process, 'display', 'klayout', stackup, pdkdir+'/setup/klayout/sky130A.lyp')
```

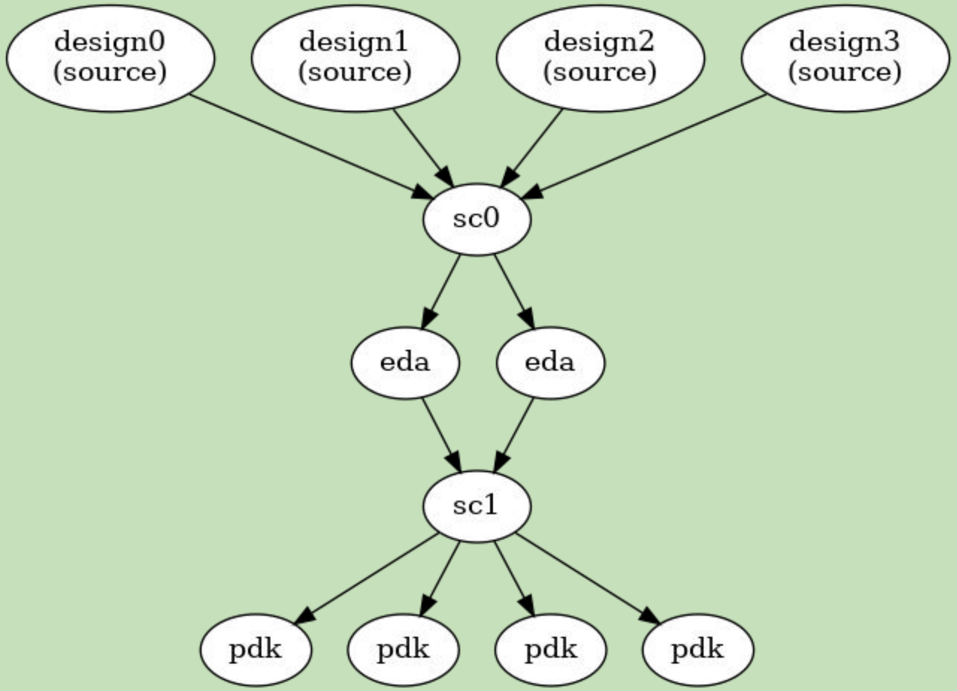
IDEA #1: *We need “LLVM like IR for CAD”*

Status Quo



Total Flows = D x E x P
D = Designs, E = EDA vendors, P = PDKs
Total Flows (D=100,E = 4,P = 10) = **4,000**

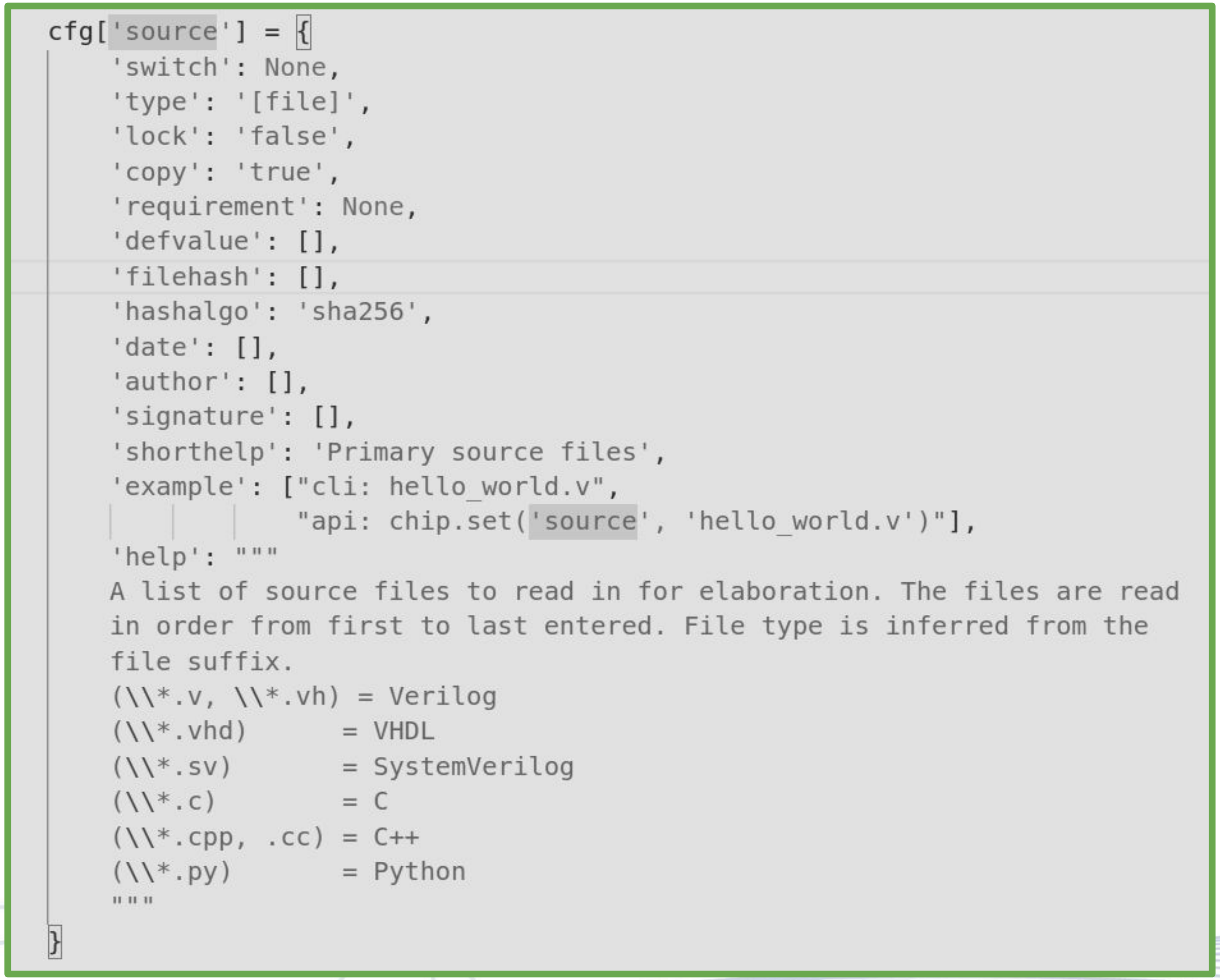
Our Approach



Total Flows (D=100,E = 4,P = 10) = **114**

Lynn Conway, Reminiscences of the VLSI Revolution: How a Series of Failures Triggered a Paradigm Shift in Digital Design

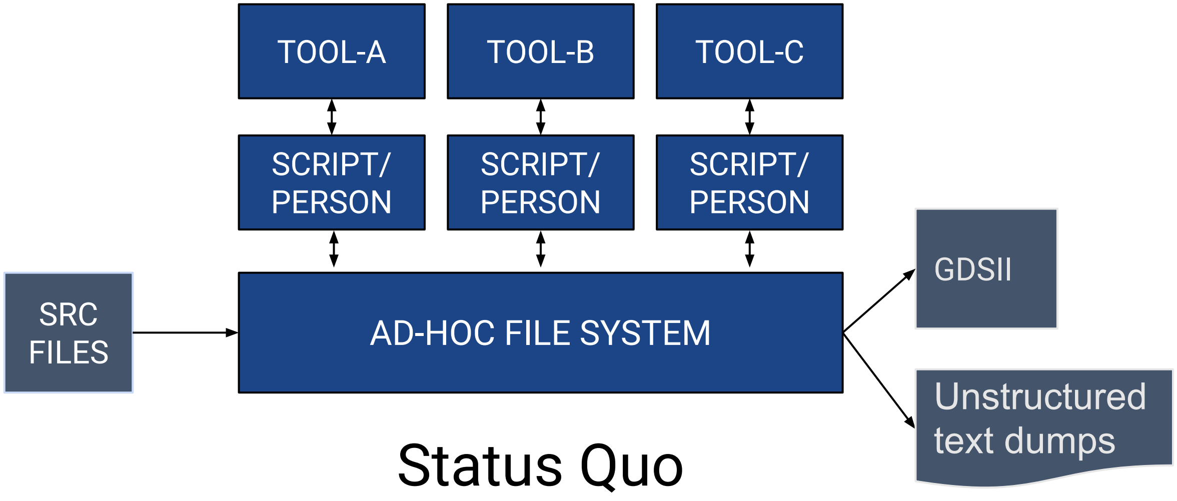
IDEA #2: *Unify config and artifacts in one manifest*



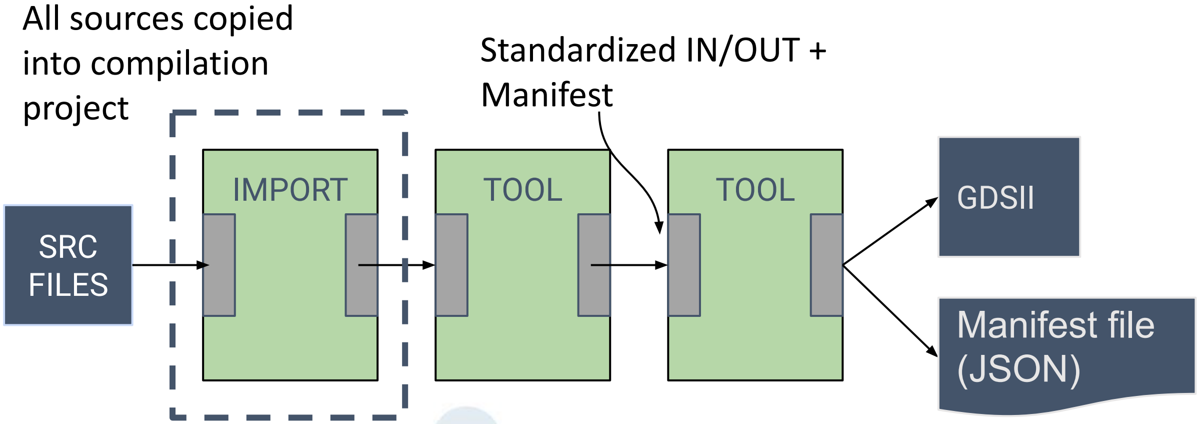
```
cfg['source'] = {
    'switch': None,
    'type': '[file]',
    'lock': 'false',
    'copy': 'true',
    'requirement': None,
    'defvalue': [],
    'filehash': [],
    'hashalgo': 'sha256',
    'date': [],
    'author': [],
    'signature': [],
    'shorthelp': 'Primary source files',
    'example': ["cli: hello_world.v",
                "api: chip.set('source', 'hello_world.v')"],
    'help': """
    A list of source files to read in for elaboration. The files are read
    in order from first to last entered. File type is inferred from the
    file suffix.
    (\\*.v, \\*.vh) = Verilog
    (\\*.vhd)       = VHDL
    (\\*.sv)        = SystemVerilog
    (\\*.c)         = C
    (\\*.cpp, .cc) = C++
    (\\*.py)        = Python
    """
}
```

- Unfied JSON “Object of Truth”
- Language agnostic
- Tracks setup and metrics
- > 200 dynamic parameters
- ASCII readable
- Signature tracking
- Embedded execution flow
- Job tracking/hashing
- Policy control
- Job history recording
- Package management

IDEA #3: *Cradle to Grave Compilation Manifests*



Status Quo

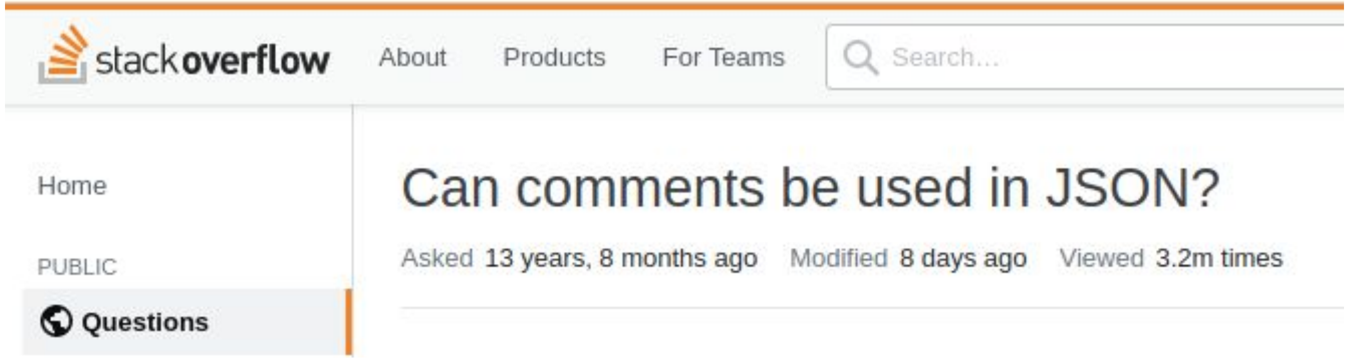

All sources copied into compilation project
Standardized IN/OUT + Manifest
SRC FILES
IMPORT
TOOL
TOOL
GDSII
Manifest file (JSON)

Silicon Compiler

- “A checksum for hardware”
- SC wraps around each tool to capture inputs and outputs.
- Each task produces a golden manifest along with task outputs.
- A manifest includes all configuration settings, actions, metrics up to that time.
- Supports automated packaging, audits, archiving, replay actions, checks
- Typical compressed JSON manifest is <1MB.

IDEA #4: *Stop abusing YAML/JSON for setup!*

- *"JSON is an open standard file format and data interchange format"* [wiki]
- Not a programming language
- Doesn't scale with complexity!
- Instead, use set/get pattern to drive json values through Python.



Can comments be used in JSON?

Asked 13 years, 8 months ago Modified 8 days ago Viewed 3.2m times

Create API to leverage the awesome power of Python!

Function	Description
set/add()	Set, add manifest value
get()	Get manifest value
getkeys()	Return list of manifest keys

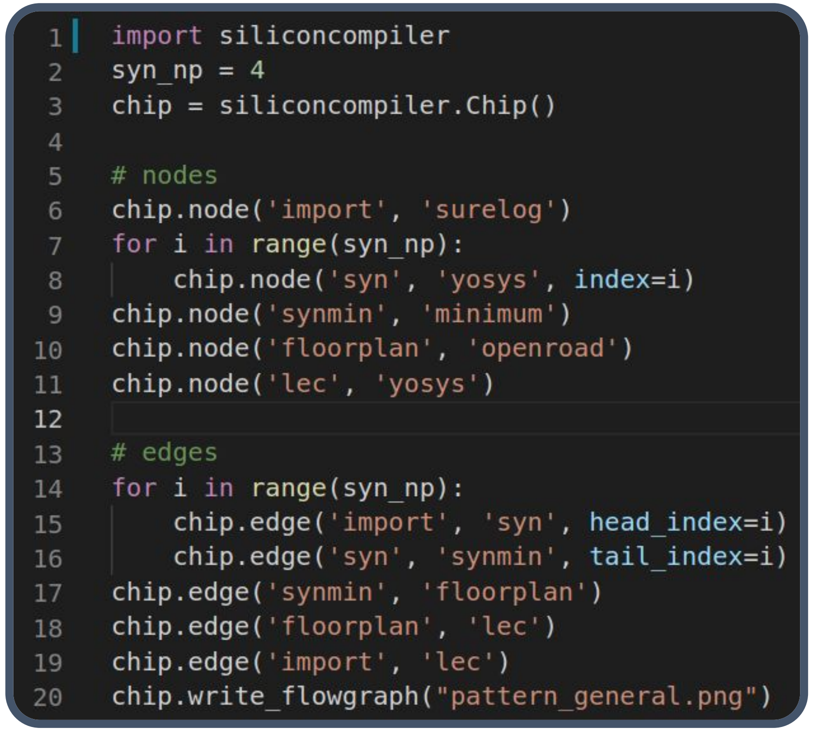
IDEA #5: *Scalable execution model*

"Serial Legacy Flow"

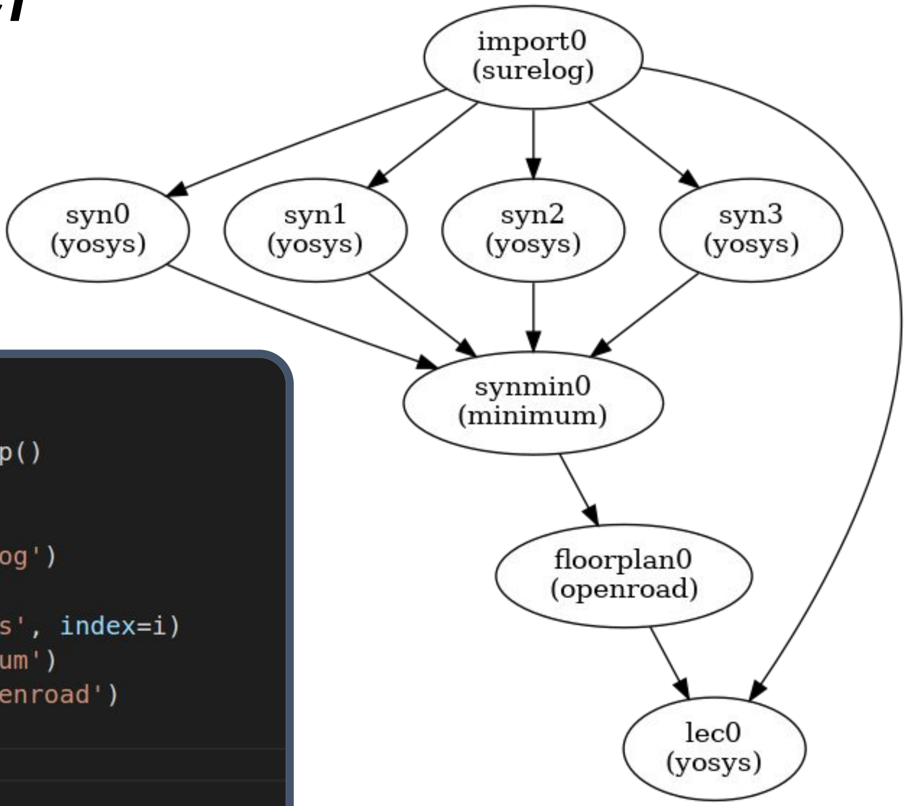
- Make files, BASH, Python, PERL,...
- SMP centric (not cloud centrics
- Fork/join parallelism with LSF/GRD
- LSF/Grid
- Low productivity and agility
- Not elastic
- No support for client/server architeture

"Cloud First Flow"

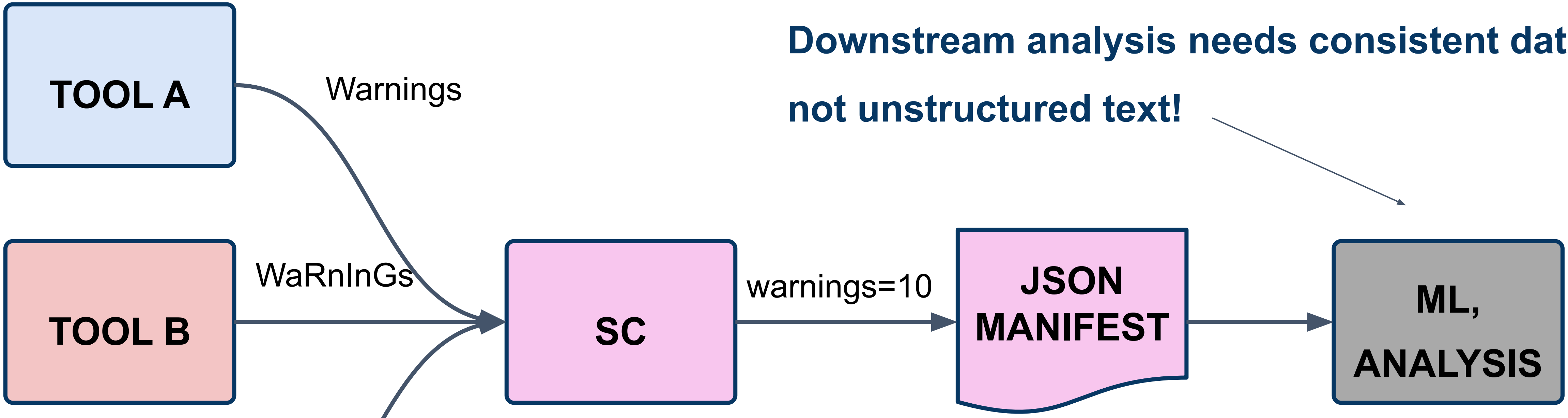
- Static flowgraph execution model
- Fow defined in schema manifest
- Python wrapper functions for ease of use
- Single call run() task for by user
- Runtime schedules on local machine or remote
- Built in SLURM job management



```python
import siliconcompiler
syn_np = 4
chip = siliconcompiler.Chip()

# nodes
chip.node('import', 'surelog')
for i in range(syn_np):
    chip.node('syn', 'yosys', index=i)
chip.node('synmin', 'minimum')
chip.node('floorplan', 'openroad')
chip.node('lec', 'yosys')

# edges
for i in range(syn_np):
    chip.edge('import', 'syn', head_index=i)
    chip.edge('syn', 'synmin', tail_index=i)
chip.edge('synmin', 'floorplan')
chip.edge('floorplan', 'lec')
chip.edge('import', 'lec')
chip.write_flowgraph("pattern_general.png")
```



IDEA #6: *Standardize Metrics*

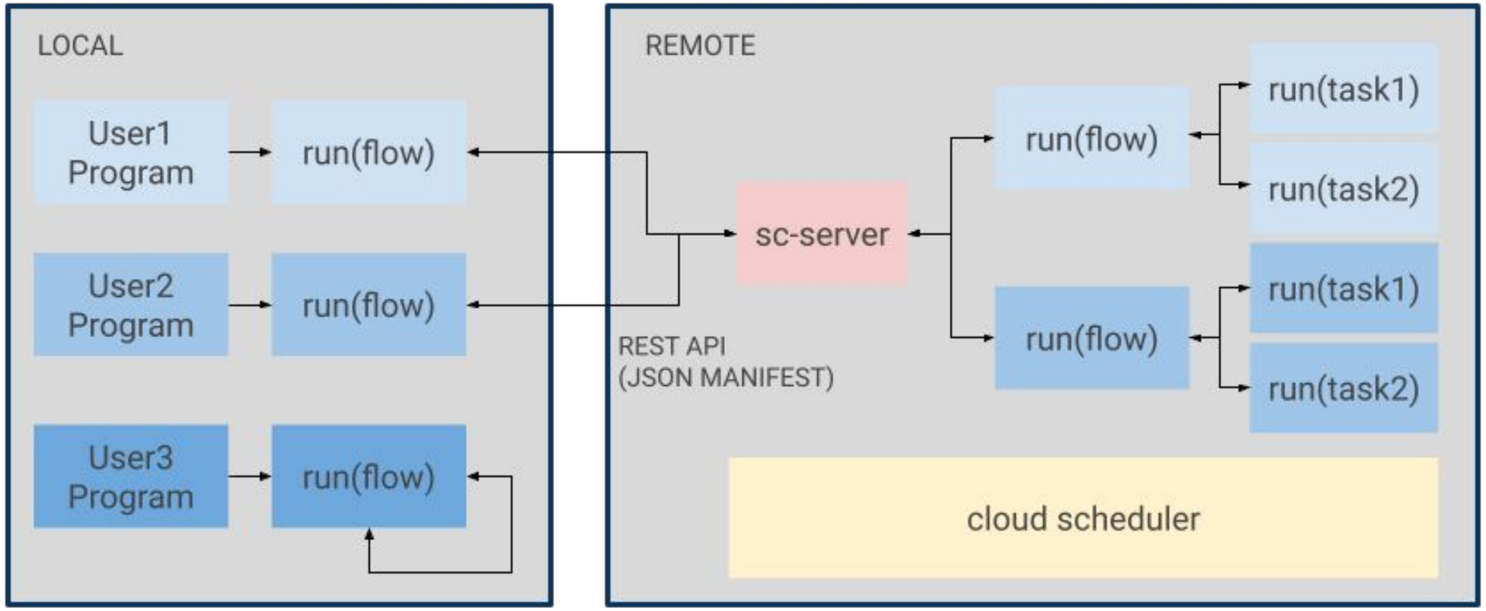


Downstream analysis needs consistent data, not unstructured text!

- 40+ standardized terms
- Error/warnings/drvs
- PPA
- Runtime resources
- Design data (bufs, regs,...)

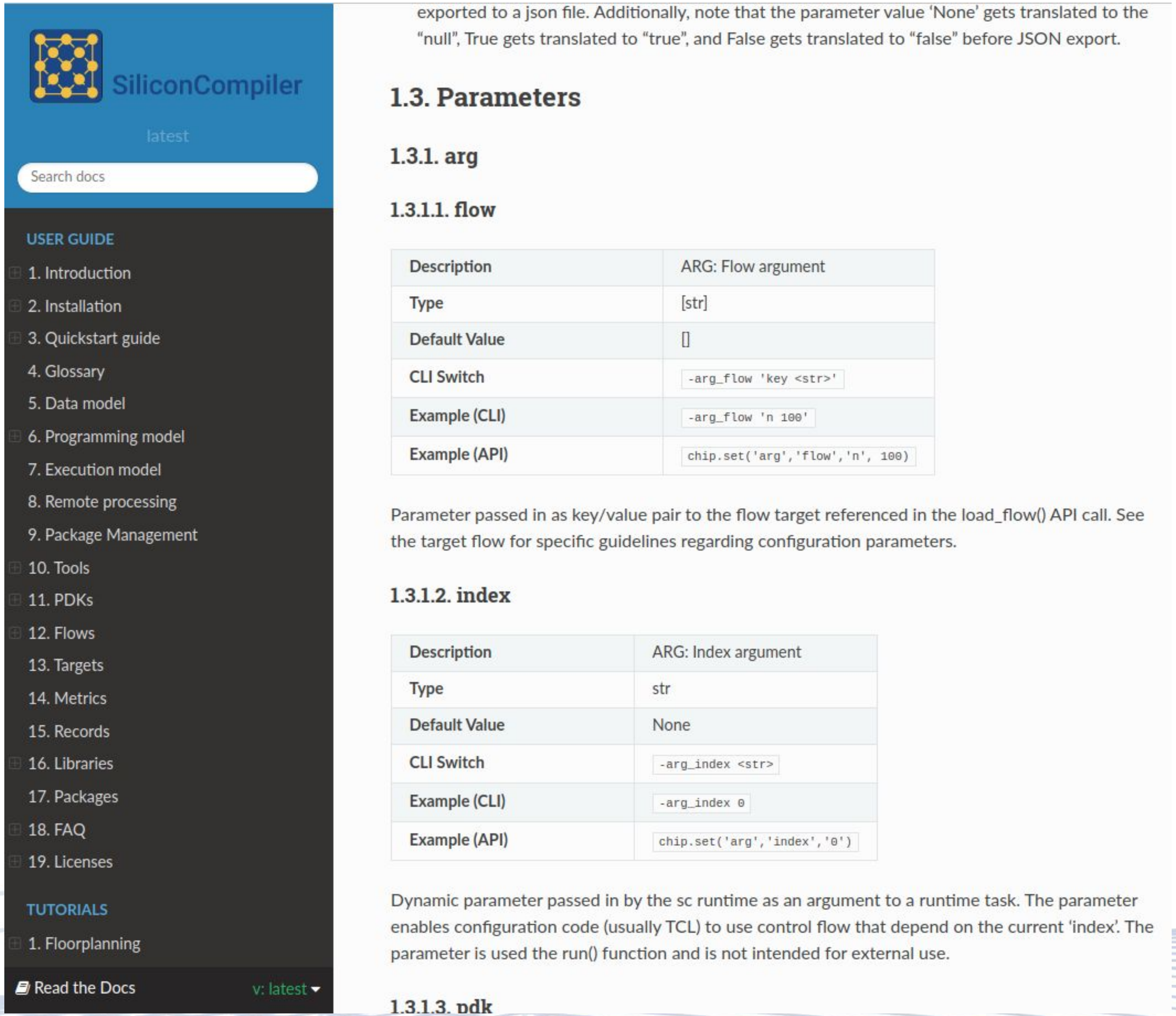
Prof. Kahng, Jinwook Jung (IBM Research), Seungwon Kim and Ravi Varadarajan gave Tutorial #1, "IEEE CEDA DATC RDF and METRICS2.1: Toward a Standard Platform for ML-Enabled EDA and IC Design"

IDEA #7: *Client/Server model to remove barriers*



Observation	**Architecture Decision**
Linux has 1% market share	Support Windows, MacOS, Linux natively
Installation/EDA/PDK pain for EDA is real	Zero install client/server remote REST API
Modern chip design needs massive compute	Platform agnostic **scalable** execution model
Cloud compute can be expensive	Support local execution

IDEA #8: *Auto-generated documentation*



exported to a json file. Additionally, note that the parameter value 'None' gets translated to the "null", True gets translated to "true", and False gets translated to "false" before JSON export.

1.3. Parameters

1.3.1. arg

1.3.1.1. flow

Description	ARG: Flow argument
Type	[str]
Default Value	[]
CLI Switch	`-arg_flow 'key <str>'`
Example (CLI)	`-arg_flow 'n 100'`
Example (API)	`chip.set('arg','flow','n', 100)`

Parameter passed in as key/value pair to the flow target referenced in the load_flow() API call. See the target flow for specific guidelines regarding configuration parameters.

1.3.1.2. index

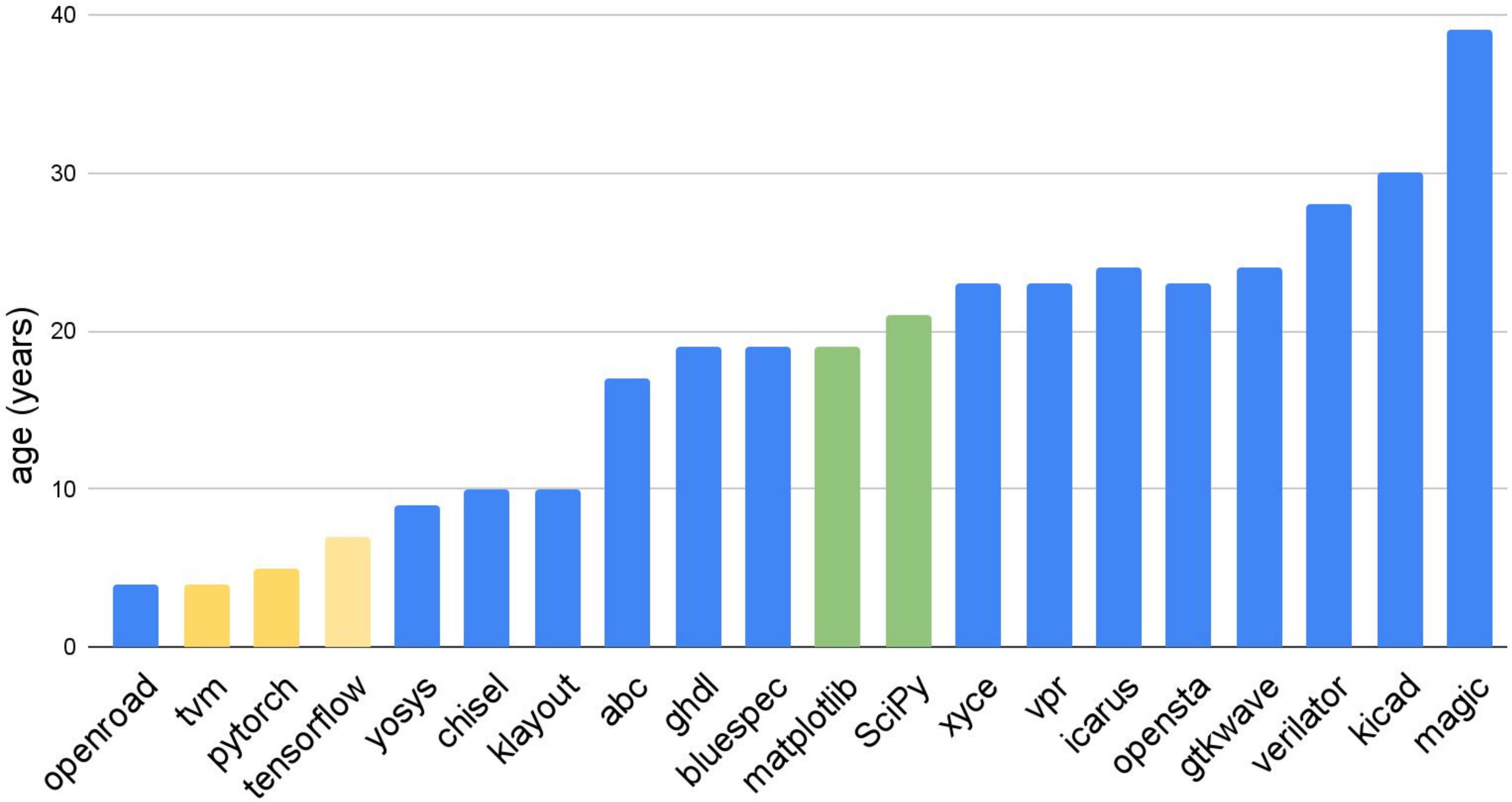
Description	ARG: Index argument
Type	str
Default Value	None
CLI Switch	`-arg_index <str>`
Example (CLI)	`-arg_index 0`
Example (API)	`chip.set('arg','index','0')`

Dynamic parameter passed in by the sc runtime as an argument to a runtime task. The parameter enables configuration code (usually TCL) to use control flow that depend on the current 'index'. The parameter is used the run() function and is not intended for external use.

1.3.1.3. pdk

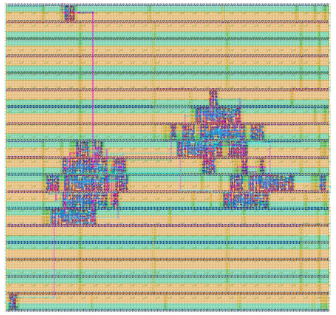
- Most OSS projects don't document well
- Address problem through automation
- Everything starts with the JSON schema
- Inspired by Sphinx, doxygen
- Make documentation a design principle
- Based on Sphinx auto-doc system
- Custom Python code for all scalable docs generators (schema, flows, tools, pdks, libraries, target).
- 338 pages of docs and counting!
- https://docs.siliconcompiler.com

IDEA #9: *Built-to-last*

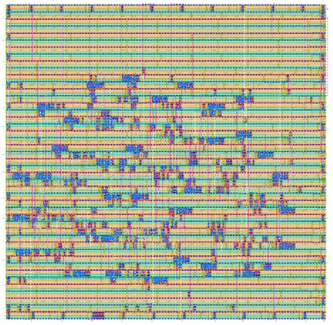


- Build to scale (LLVM inspired)
- Plan for 10 years
- Developer redundancy
- Ensure long term funding
- Make wise dependencies choices!
- Leverage others (surelog,OR,...)

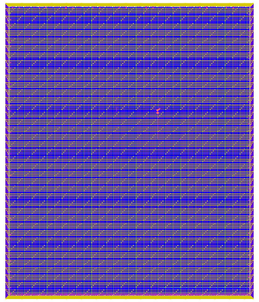
Demonstrator #1*:* Gallery of Python generated layouts



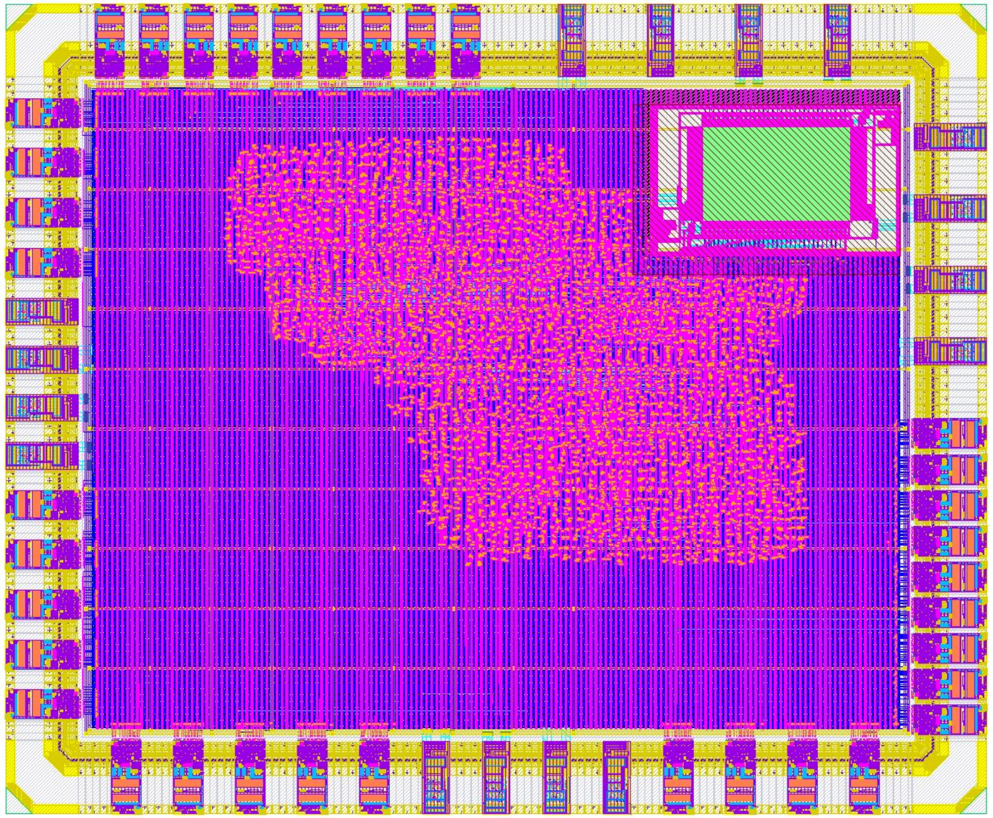
Heartbeat *(FreePDK45)*



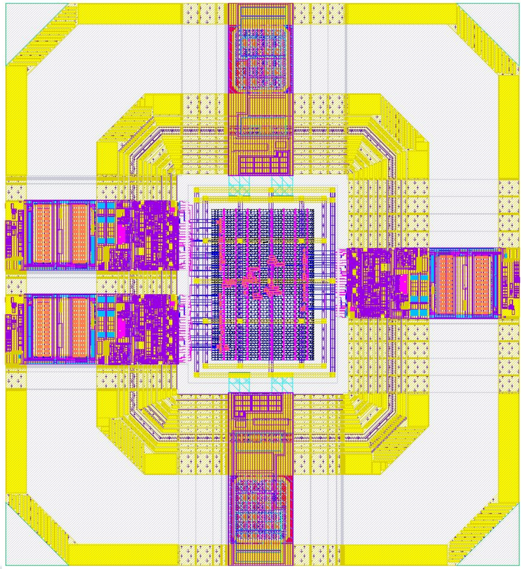
GCD *(FreePDK45)*



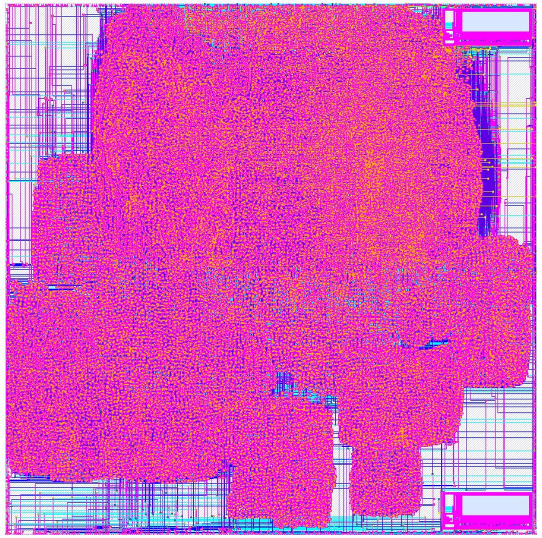
Caravel *(Sky130)*



ZeroSoC *(Sky130)*

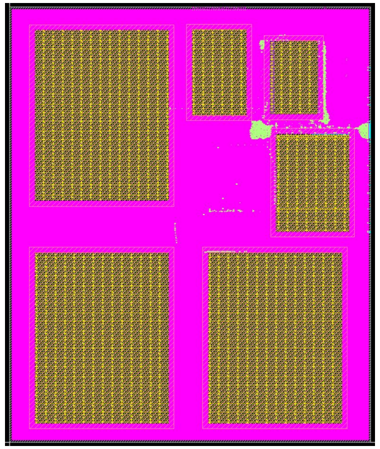


Heartbeat w/ padring
(Sky130)

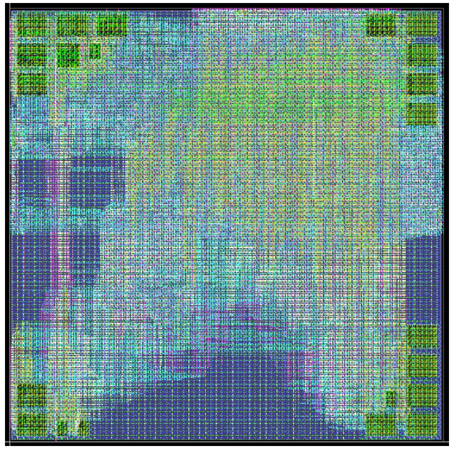


CVA6/Ariane
(Sky130)

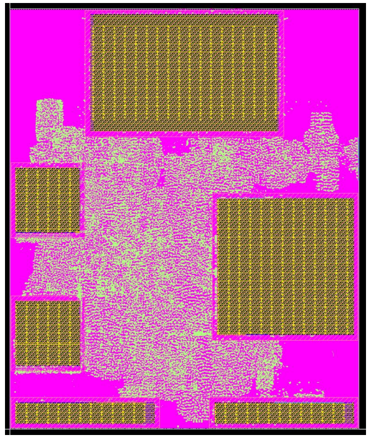
Demonstrator #2: OpenROAD flow integration



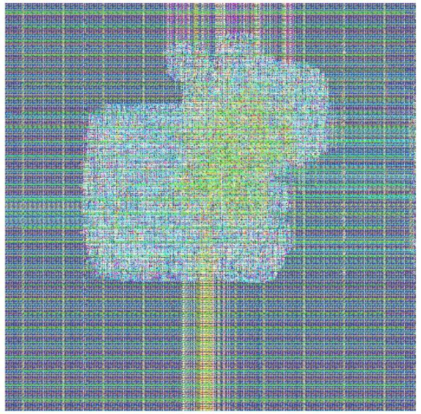
Chameleon
(sky130)



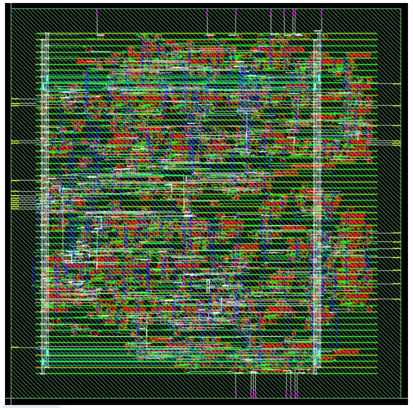
BlackParrot
(freepdk45)



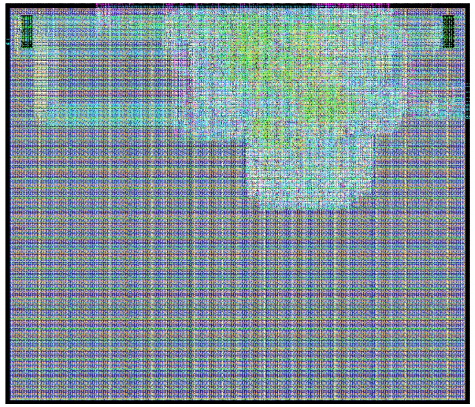
Microwatt
(sky130)



Ibex
(freepdk45)



uart
(asap7)



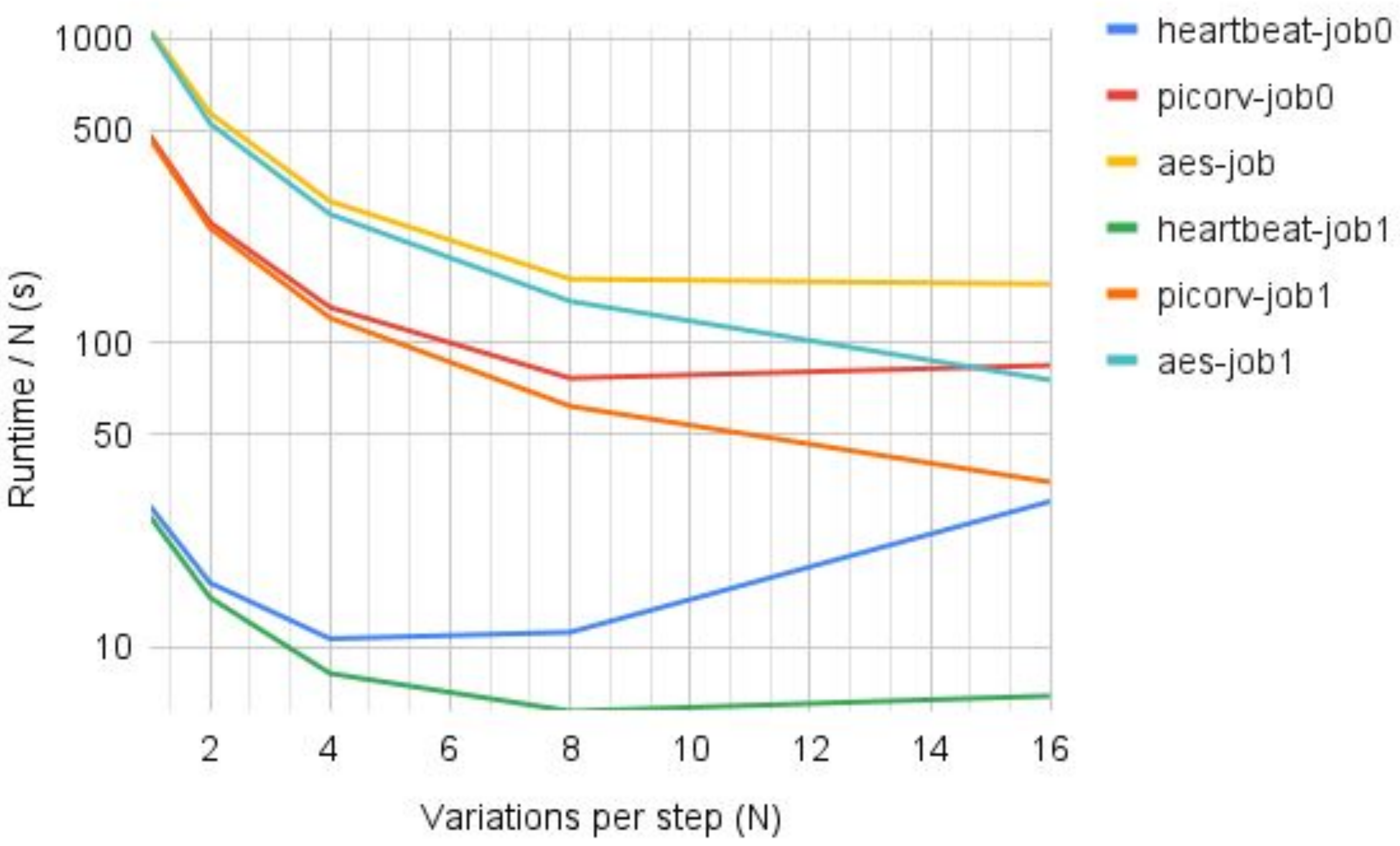
tinyrocket
(freepdk45)

- https://github.com/The-OpenROAD-Project/OpenROAD-flow-script
- Created SC “flow/Makefile.py” to replicate “flow/Makefile” (130 Python LOC)
- See “flow/scripts/sc” for rest of code

Try it!

```
$ python3 Makefile.py -DESIGN_CONFIG=<config.mk>
```

Demonstrator #3: *Design of Experiments*



Task:

- Select the settings that gives the best PPA for your design.

Experiment:

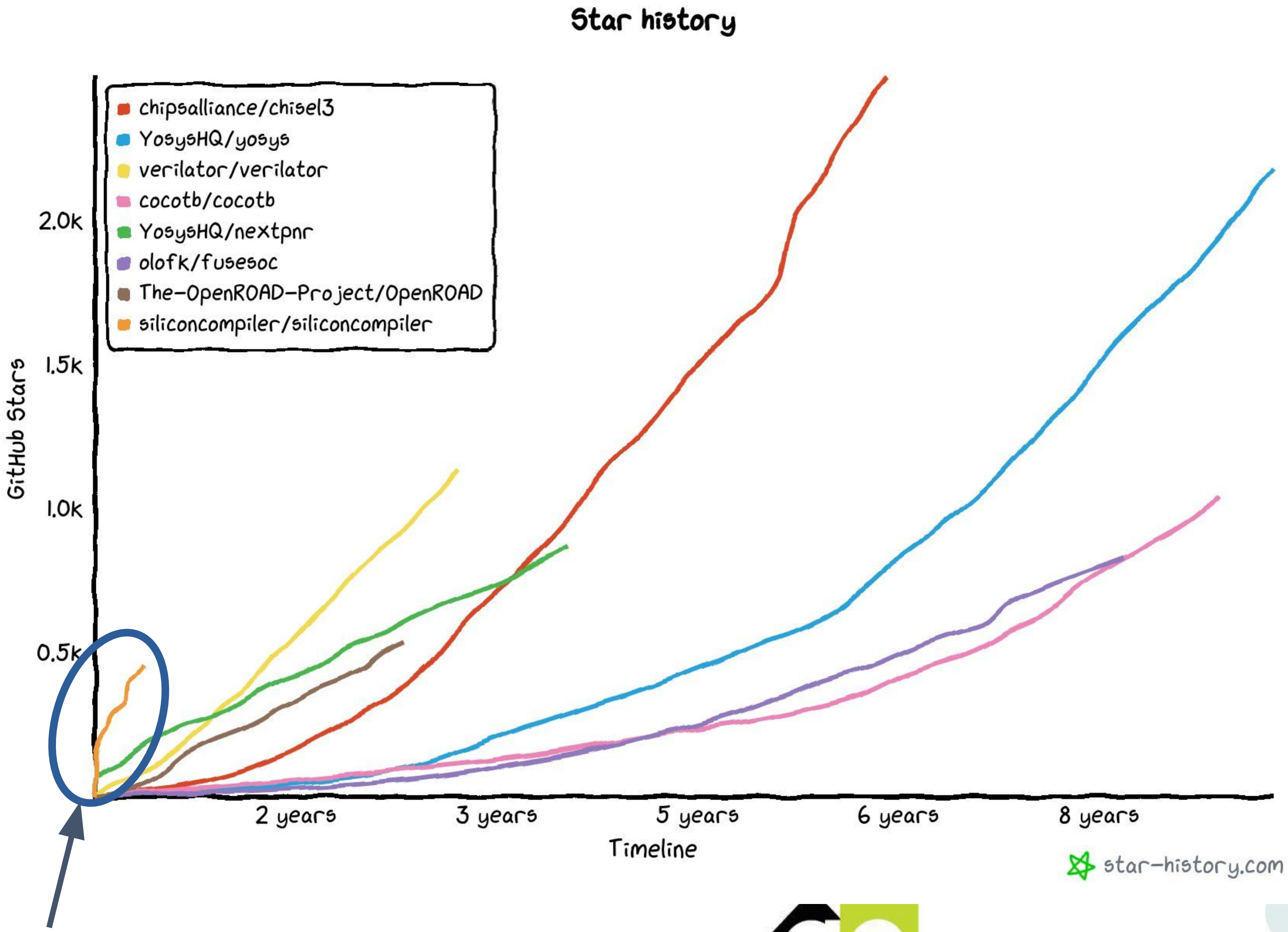
- RTL2GDS flow using Yosys/Openroad
- Run N independent jobs for syn, place, cts, route.
- Built-in min step selects best one
- See examples/benchmark/benchmark.py

Conclusions:

- 10X speedup demonstrated
- Static flowgraphs a good fit for CAD!
- Python multiprocessing “good enough”
- Manifest read/write overhead minimal
- Runs on SMP machines and in cloud

Summary: *This is just the beginning…*

PDKs	intel16, gf12lp, sky130, asap7, freepdk45
Languages	C (Bambu), SV, VHDL, Chisel, Migen/Amaranth, Bluespec
Simulation	Verilator, Icarus, GHDL
Synthesis	Yosys, Vivado, Synopsys, Cadence
ASIC APR	OpenRoad, Synopsys, Cadence
FPGA APR	VPR, nextpnr, Vivado
Viewer	OR Klayout, Cadence, Synopsys
DRC/LVS	Magic, Mentor, Synopsys



SiliconCompiler released Dec 2021