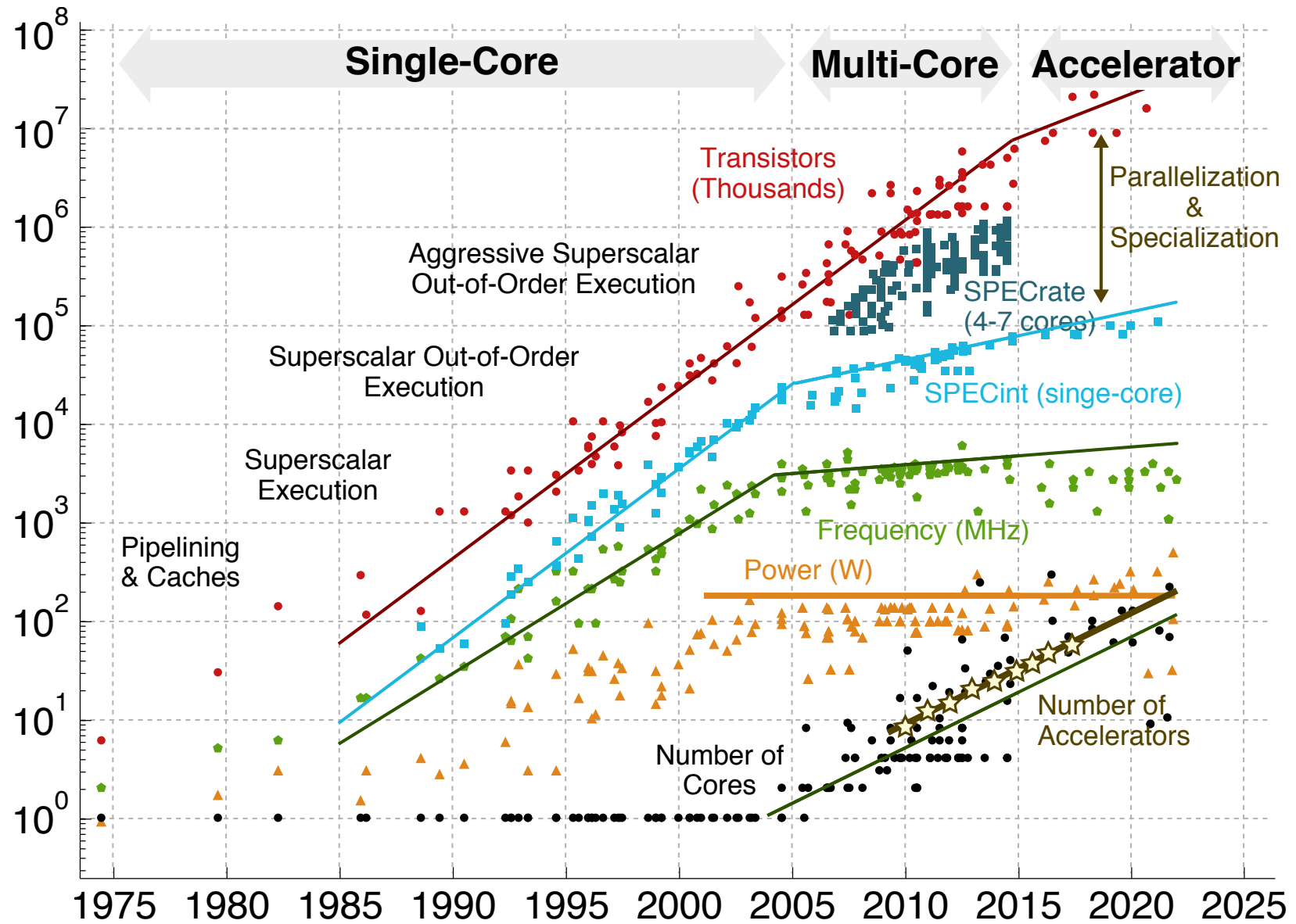




A New Era of Open-Source Hardware

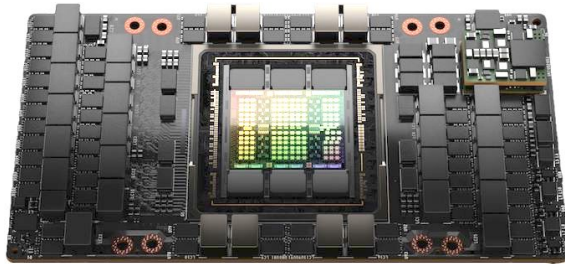
Christopher Batten

Computer Systems Laboratory
Electrical and Computer Engineering
Cornell University



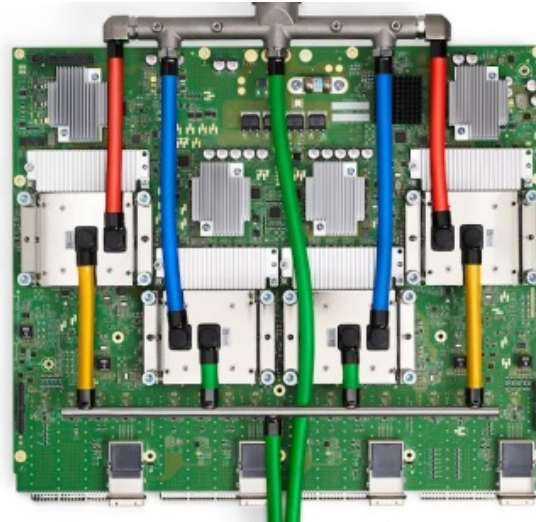
C. Batten, M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, K. Rupp & [Y. Shao, IEEE Micro'15] & [C. Leiserson, Science'20]

Accelerators for Machine Learning in the Cloud



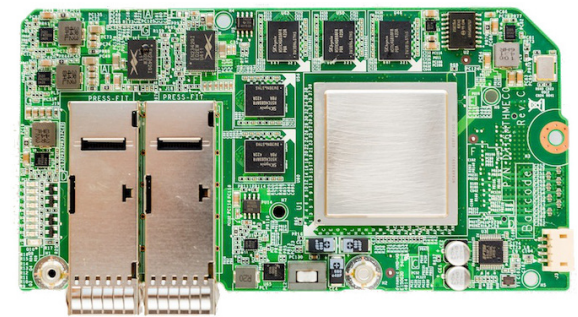
NVIDIA DGX Hopper

- ▶ Graphics processor specialized just for accelerating machine learning
- ▶ Available as part of a complete system with both the software and hardware designed by NVIDIA



Google TPU v4

- ▶ Custom chip specifically designed to accelerate Google's TensorFlow C++ library
- ▶ Tightly integrated into Google's data centers



Microsoft Catapult

- ▶ Custom FPGA board for accelerating Bing search and machine learning
- ▶ Accelerators developed with/by app developers
- ▶ Tightly integrated into Microsoft data center's and cloud computing platforms

Accelerators for Machine Learning at the Edge



Amazon Echo

- ▶ Developing AI chips so Echo line can do more on-board processing
- ▶ Reduces need for round-trip to cloud
- ▶ Co-design the algorithms and the underlying hardware

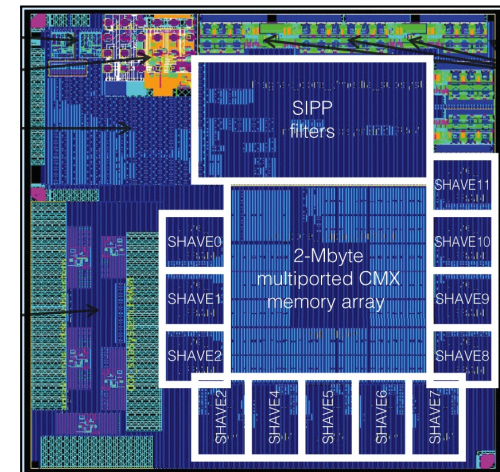


Facebook Oculus

- ▶ Starting to design custom chips for Oculus VR headsets
- ▶ Significant performance demands under strict power requirements



Movidius Myriad 2



Top-five software companies are all building custom accelerators

- ▶ **Facebook:** w/ Intel, in-house AI chips
- ▶ **Amazon:** Echo, Oculus, networking chips
- ▶ **Microsoft:** Hiring for AI chips
- ▶ **Google:** TPU, Pixel, convergence
- ▶ **Apple:** SoCs for phones and laptops

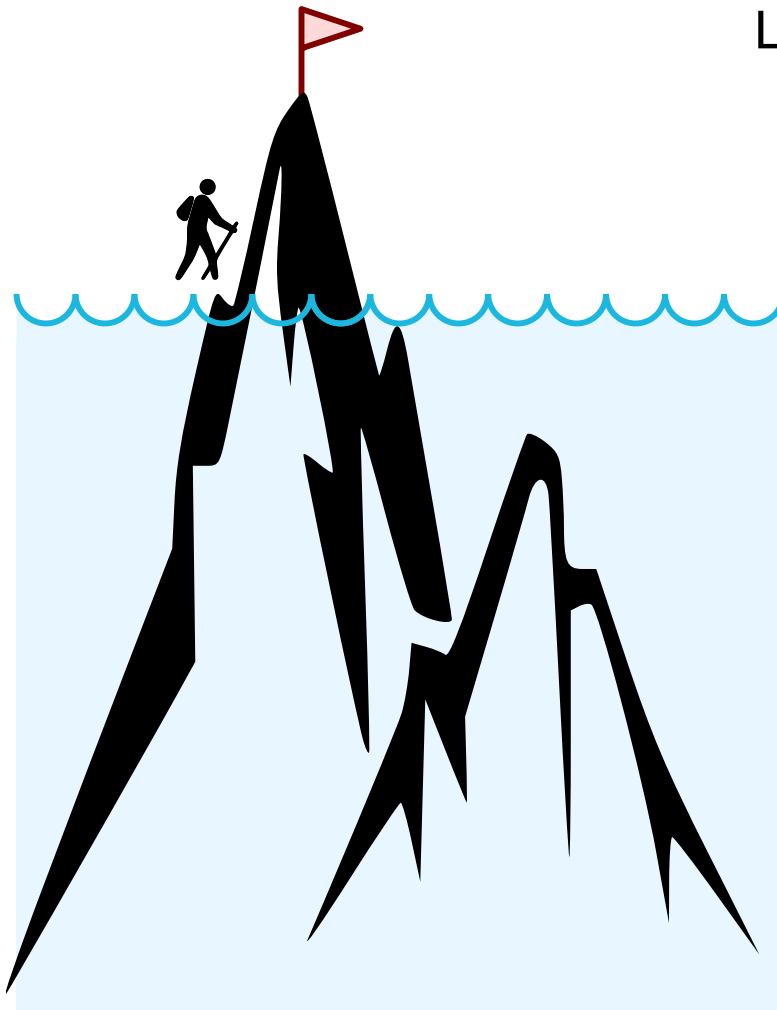
Chip startup ecosystem for machine learning accelerators is thriving!

How can we **accelerate innovation** in **accelerator-centric** hardware design?

- ▶ **Graphcore**
- ▶ **Nervana**
- ▶ **Cerebras**
- ▶ **Wave Computing**
- ▶ **Horizon Robotics**
- ▶ **Cambricon**
- ▶ **DeePhi**
- ▶ **Esperanto**
- ▶ **SambaNova**
- ▶ **Eyeriss**
- ▶ **Tenstorrent**
- ▶ **Mythic**
- ▶ **ThinkForce**
- ▶ **Groq**
- ▶ **Lightmatter**

Software Innovation Today

Like climbing an iceberg – much is hidden!



Your proprietary code

- Instagram
- \$500K seed with 13 people → \$1B

Open-source software

- Python
- Django
- Memcached
- Postgres/SQL
- Redis
- nginx
- Apache, Gnuicorn
- Linux
- GCC

"What Powers Instagram:
Hundreds of Instances,
Dozens of Technologies"
<https://goo.gl/76fWrM>

Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16

Hardware Innovation Today



Like climbing a mountain – nothing is hidden!

What you have to build

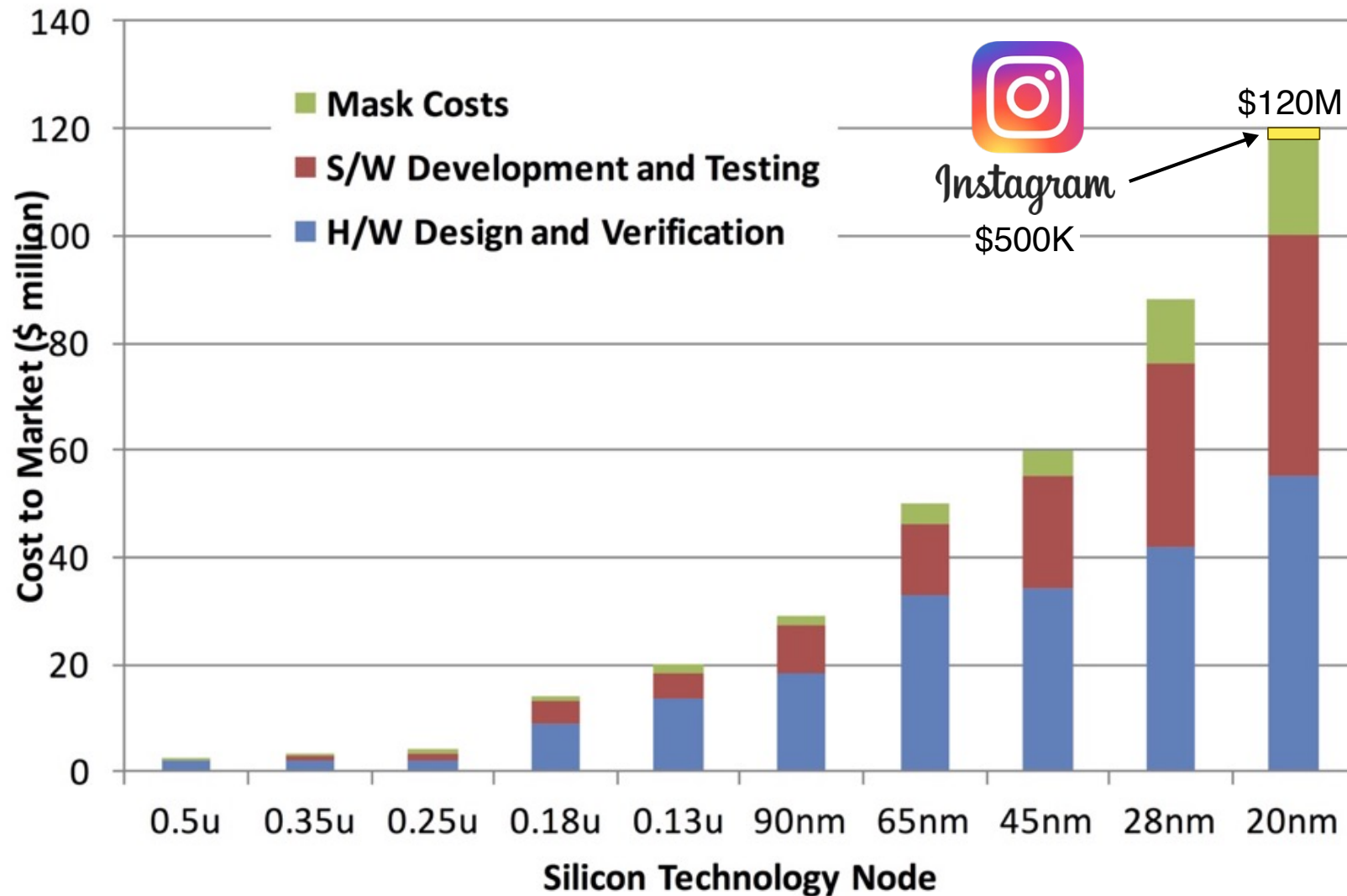
- New machine learning accelerator
- Other unrelated components, anything you cannot afford to buy or for which COTS IP does not do

Closed source

- ARM A57, A7, M4, M0
- ARM on-chip interconnect
- Standard cells, I/O pads, DDR Phy
- SRAM memory compilers
- VCS, Modelsim
- DC, ICC, Formality, Primetime
- Stratus, Innovus, Voltus
- Calibre DRC/RCX/LVS, SPICE

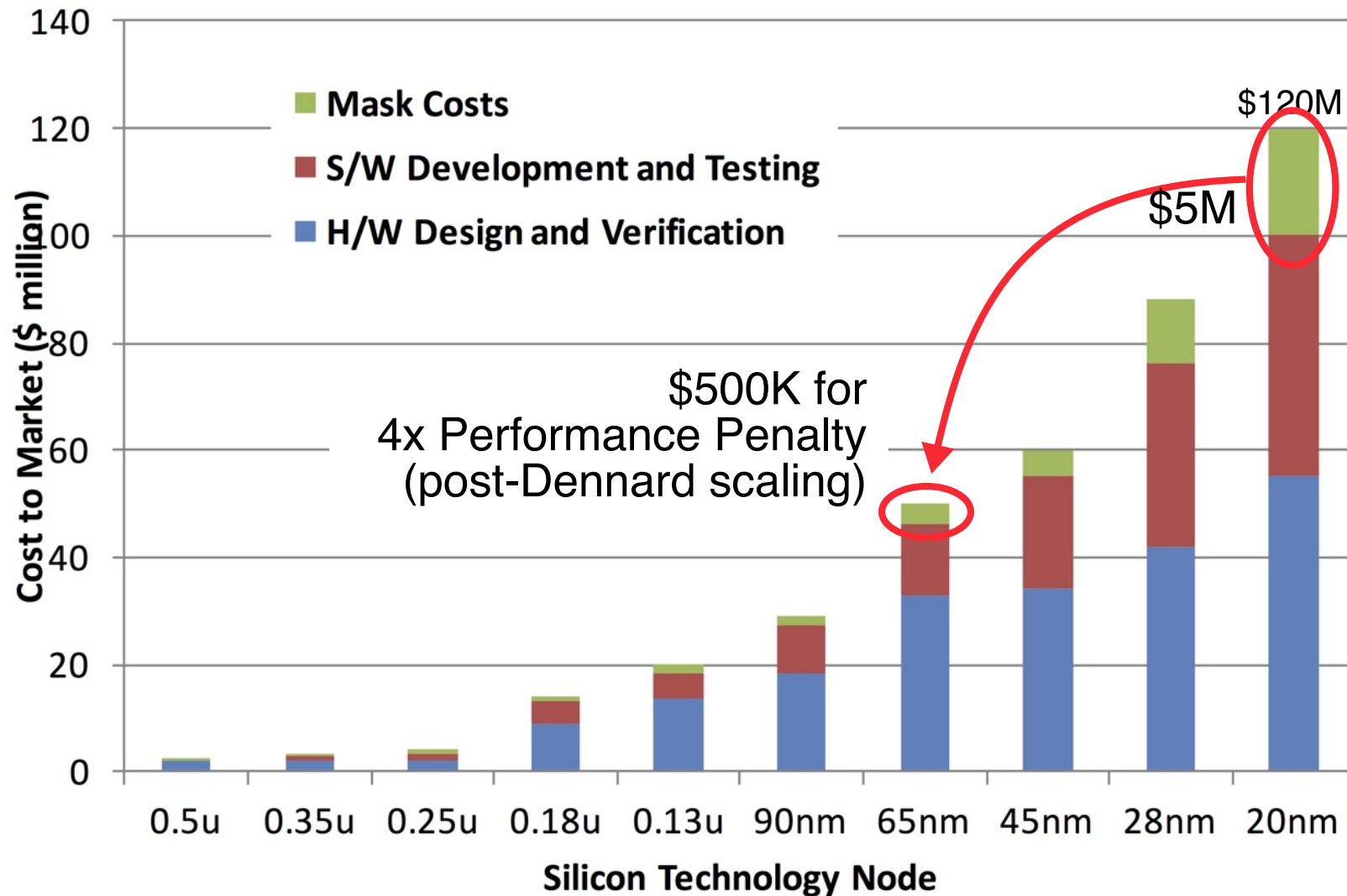
Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16

Chip Costs Are Skyrocketing



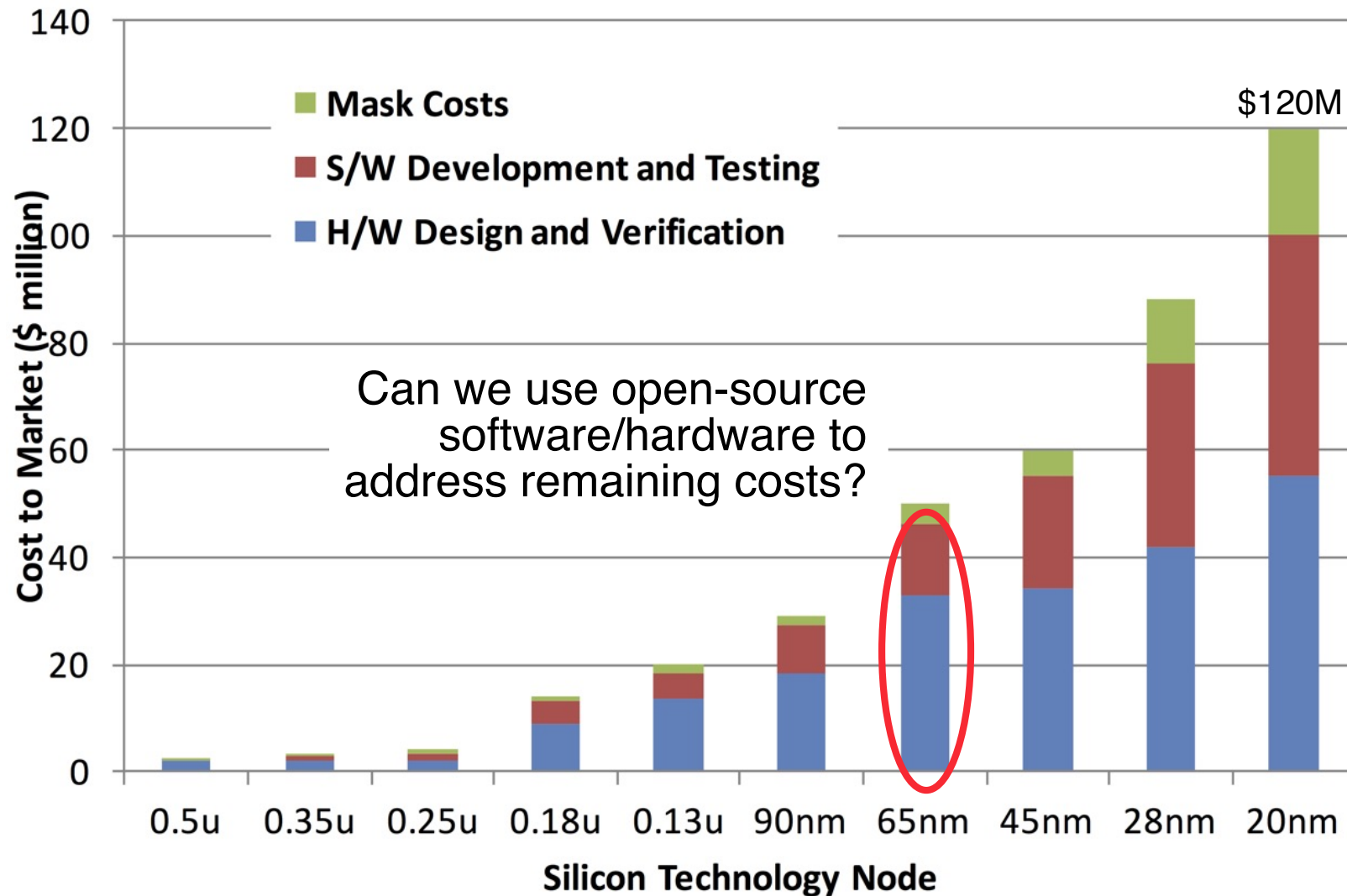
Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16; original: International Business Strategies & T. Austin.

Minimum Viable Product/Prototype



Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16; original: International Business Strategies & T. Austin.

Minimum Viable Product/Prototype



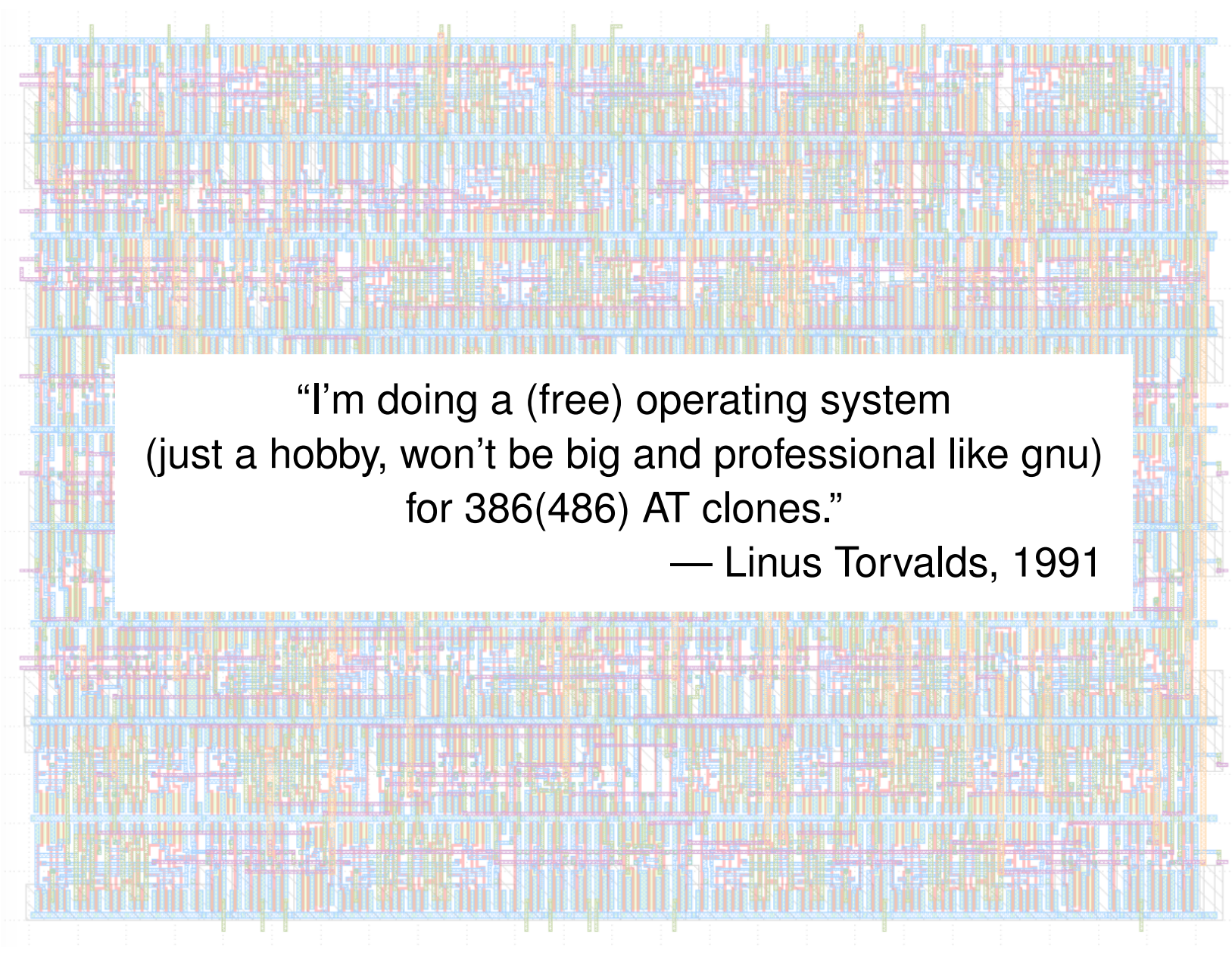
Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16; original: International Business Strategies & T. Austin.

How can HW design be more like SW design?

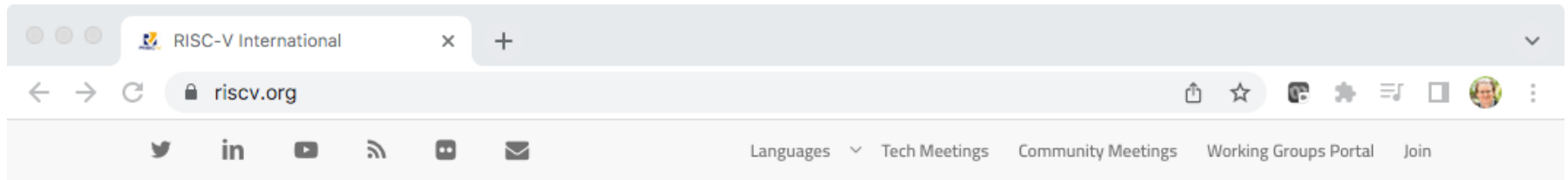
Open-Source	Software	Hardware
high-level languages	Python, Ruby, R, Javascript, Julia	Chisel, PyMTL, PyRTL, MyHDL, JHDL, Clash, Calyx
libraries	C++ STL, Python std libs	BaseJump
systems	Linux, Apache, MySQL, memcached	Rocket, Pulp/Ariane, OpenPiton, Boom, FabScalar, MIAOW, Nyuzi
standards	POSIX	RISC-V ISA, RoCC, TileLink
tools	GCC, LLVM, CPython, MRI, PyPy, V8	Icarus Verilog, Verilator, qflow, Yosys, TimberWolf, qrouter, magic, klayout, ngspice
methodologies	agile software design	agile hardware design
cloud	IaaS, elastic computing	IaaS, elastic CAD

c. 2018

```
# Ubuntu Server 16.04 LTS (ami-43a15f3e)
% sudo apt-get update
% sudo apt-get -y install build-essential qflow
% mkdir qflow && cd qflow
% wget http://opencircuitdesign.com/qflow/example/map9v3.v
% qflow synthesize place route map9v3 # yosys, graywolf, grouter
% wget http://opencircuitdesign.com/qflow/example/osu035_stdcells.gds2
% magic # design def/lef -> magic format
>>> lef read /usr/share/qflow/tech/osu035/osu035_stdcells.lef
>>> def read map9v3.def
>>> writeall force map9v3
% magic # stdcell gds -> magic format
>>> gds read osu035_stdcells.gds2
>>> writeall force
% magic map9v3
>>> gds write map9v3 # design + stdcells magic format -> gds
% sudo apt-get -y install libqt4-dev-bin libqt4-dev libz-dev
% wget http://www.klayout.org/downloads/source/klayout-0.24.9.tar.gz
% tar -xzvf klayout-0.24.9.tar.gz && cd klayout-0.24.9
% ./build.sh -noruby -nopython
% wget http://www.csl.cornell.edu/~cbatten/scmos.lyp
% ./bin.linux-64-gcc-release/klayout -l scmos.lyp ../map9v3.gds
```



“I’m doing a (free) operating system
(just a hobby, won’t be big and professional like gnu)
for 386(486) AT clones.”
— Linus Torvalds, 1991



About RISC-V

Membership

RISC-V Exchange

Technical

News & Events

Community



What's New!



RISC-V Announces First New Specifications of 2022, Adding to 16 Ratified in 2021 | RISC-V International

RISC-V Community News | Announcements, What's New

Efficient Trace, Supervisor Binary Interface, Unified Extensible Firmware Interface, and Zmmul Multiply-Only Extension Accelerate Embedded- and Large-System Design. Six Additional Specifications Already In the Pipeline As Development Extends Into Vertical...

RISC-V: The Open era of computing



Check out our local language pages!

日本語のページをご覧ください!

访问我们的中文页面!

Get in touch!

Press: press@riscv.org

Analysts: analysts@riscv.org

General: info@riscv.org

Industry Interest in RISC-V is Growing



RISC-V Hardware *and* Software Ecosystem

Software

Open-source software:

Gcc, binutils, glibc, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS, ...

Commercial software:

Lauterbach, Segger, IAR, Micrium, ExpressLogic, Ashling, AntMicro, Imperas, UltraSoC ...



ISA specification

Golden Model

Compliance

Hardware

Open-source cores:

Rocket, BOOM, RI5CY, Ariane, PicoRV32, Piccolo, SCR1, Shakti, Swerv, Hummingbird, ...

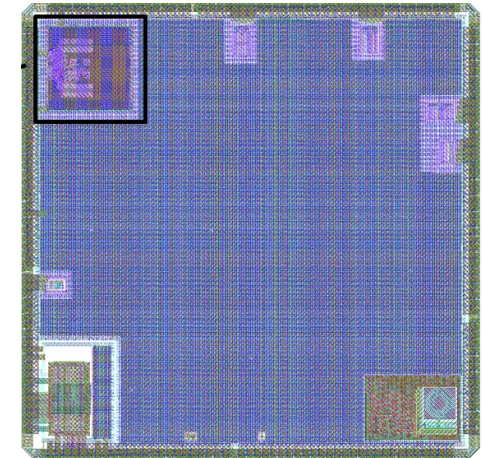
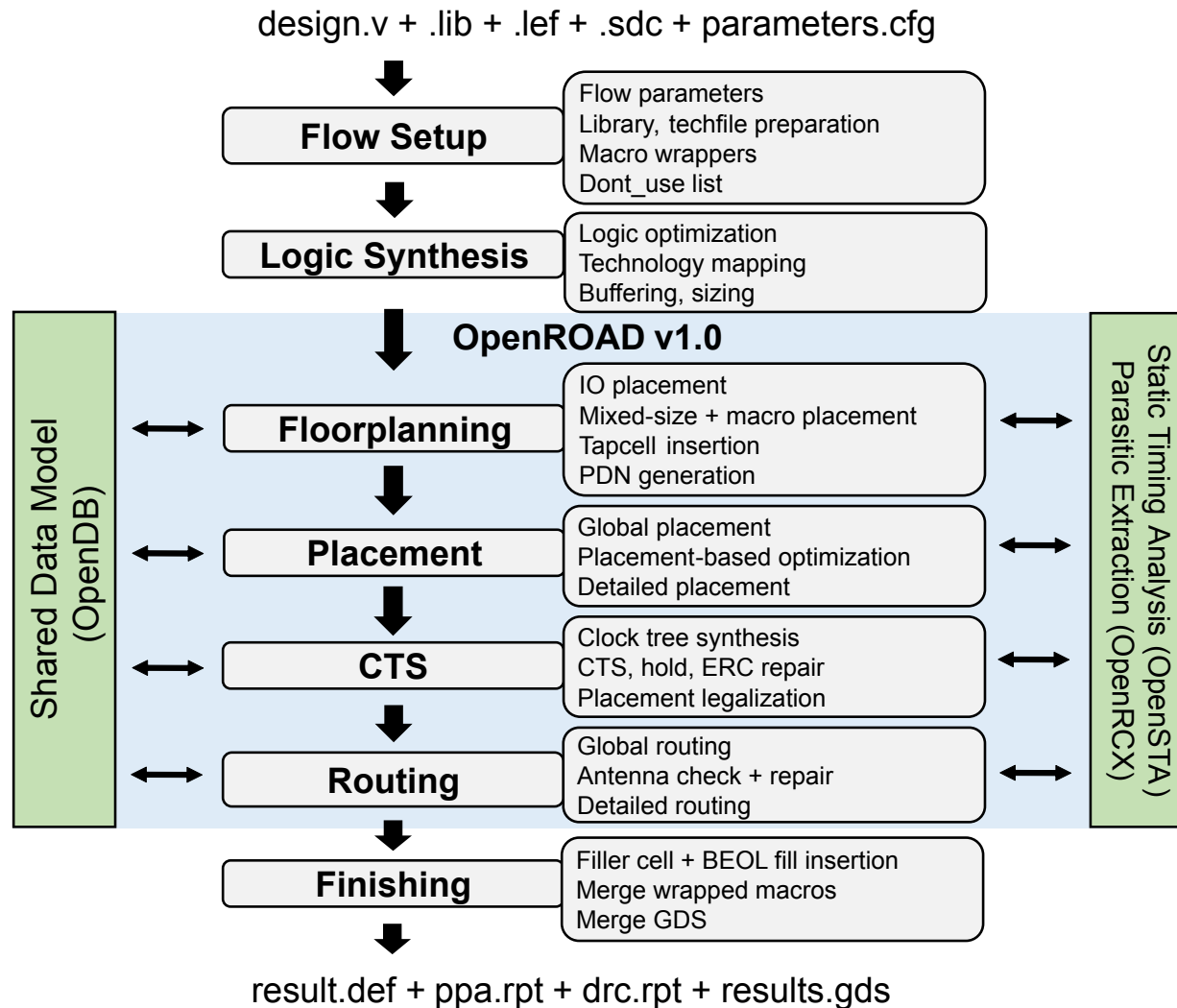
Commercial core providers:

Andes, Bluespec, Cloudbear, Codaip, Cortus, C-Sky, InCore, Nuclei, SiFive, Syntacore, ...

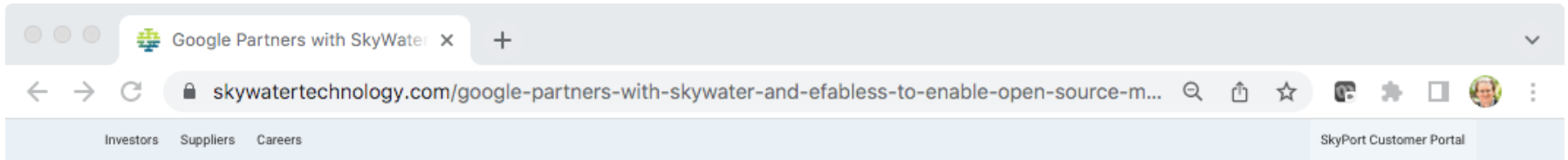
Inhouse cores:

Nvidia, +others

OpenROAD: The Future of Open-Source EDA



OpenTitan SoC
 GF12LP



Services ↓ Technologies ↓ Markets ↓ Resources ↓ About ↓

Get Started

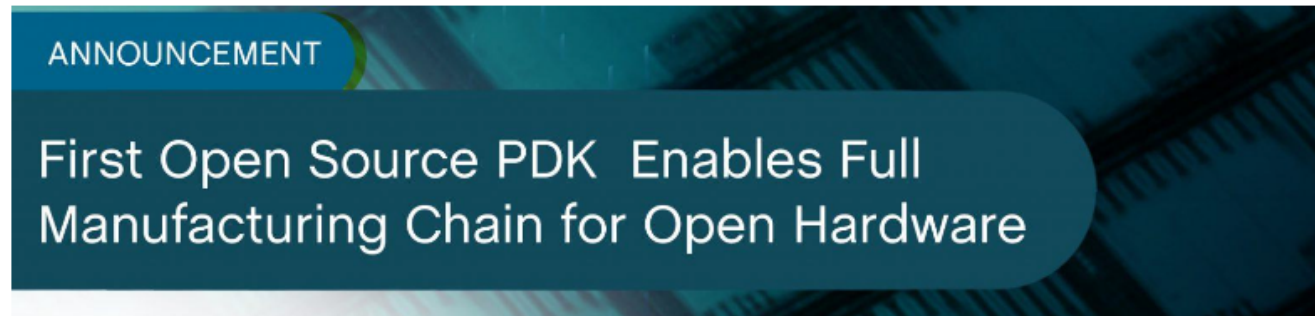


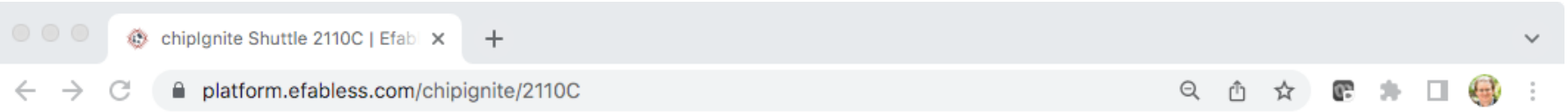
Google Partners with SkyWater and Efabless to Enable Open Source Manufacturing of Custom ASICs

SkyWater 130nm
 SkyWater 90nm
 GF 180nm

11/12/2020 | Press Releases

Share: [f](#) [t](#) [in](#)





efabless.com

Projects ▾

Tools ▾

Marketplace ▾

Community ▾

Company ▾

Login

Sign Up

chipignite

2110C

chipignite

Rapid IC Creation

Shuttle 2110C



13 of 40 project slots reserved

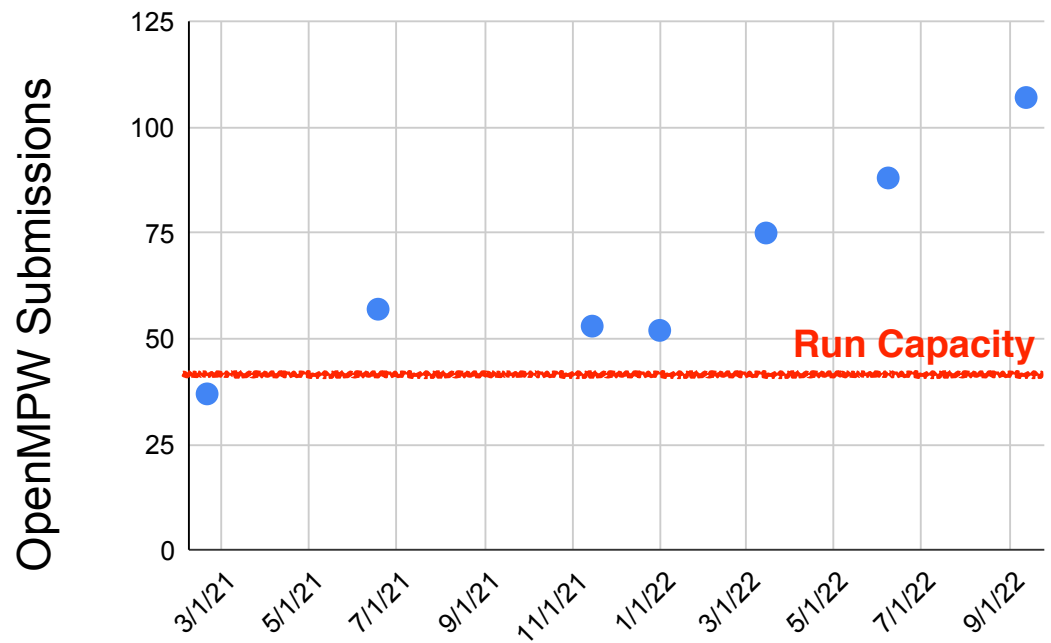
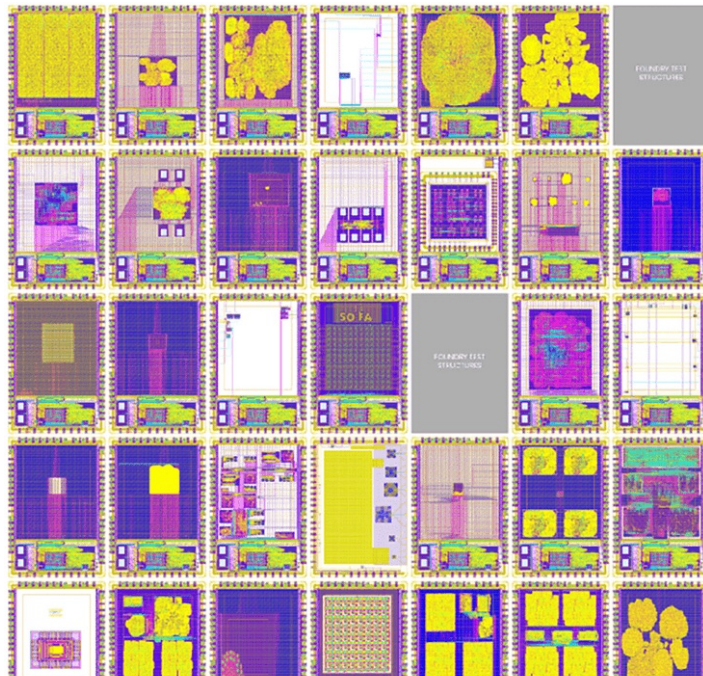
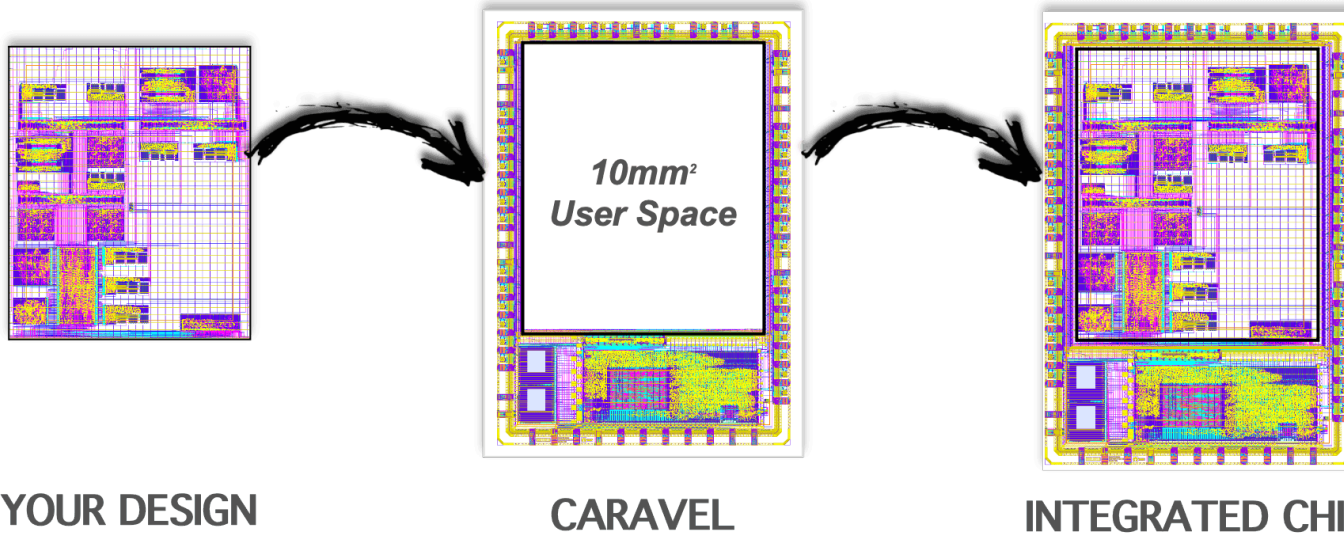
Tapeout:

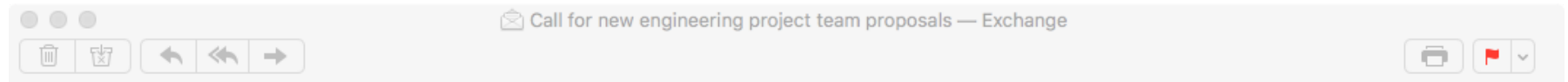
Deliver November 26, 2021 23:59 PT


y: April 01, 2022

A composite image featuring a circular inset of a chip layout. A white box highlights a "10 mm² User Design Area". A callout box points to a "WCSP Package" area. To the right, a white box displays "\$9750 per project" in red. Below the chip layout is a photograph of a green printed circuit board (PCB) populated with various electronic components.

Help





Associate Dean for Undergraduate Programs 

Archive - Exchange February 22, 2022 at 12:08 PM

[Details](#)



Call for new engineering project team proposals

To: ENGRFACULTY-L, ENGRFACULTYINCSANDBEE-L, ENGRACADEMICS-L, Cc: Lauren Stulgis,

Reply-To: Associate Dean for Undergraduate Programs

Colleagues,

Through a generous donor gift creating the **Shen Fund for Social Impact** we have the opportunity to fund multiple new engineering project teams. This program is designed to bring together new student teams under a faculty member's mentorship to address significant social challenges through novel and/or advanced engineering solutions. Falling under the Project Team Umbrella, the program will fund up to three new teams per year, with each supported for a three-year period at \$30K/yr. The teams will also be provided space and support to design and implement these projects.

Proposals may be submitted by either faculty looking to guide a group of students, or by students who will engage with a faculty member to form the teams.

Attached to this e-mail are three documents:

- **Shen Fund FAQ Sp22.pdf**: More fully describes the nature of the projects and the goals of the program (also copied to the e-mail below).
- **Shen Fund Proposal Template Sp22.docx**: Short project proposal form.
- **Shen Funded Projects Summary_Sp22.pdf**: A summary document of a currently funded teams.

The ideal project will likely develop through discussions with Lauren Stulgis (as director of the project teams) and me. Feel free to reach out to us with rough ideas and concepts and we can help to try to develop a viable proposal.

Proposals will be considered as they arrive, with discussions to strengthen each within the program constraints. The initial application is a simple document identifying the primary goals, technical challenges and plans, timeline and budget, and currently engaged personnel.

Proposals must be uploaded directly to Box by email to: Proposa.zeuyhp9wqg5p8teo@u.box.com. The first round of decisions will be made based on submissions received by **11:59pm on Sunday, March 13, 2022**.

Again, please feel free to contact me or Lauren Stulgis with any questions or to discuss potential projects.

Prof. Alan Zehnder
Associate Dean for Undergraduate Programs
177 Rhodes Hall
Phone: (607) 255-9181
email: eng_ugdean@cornell.edu

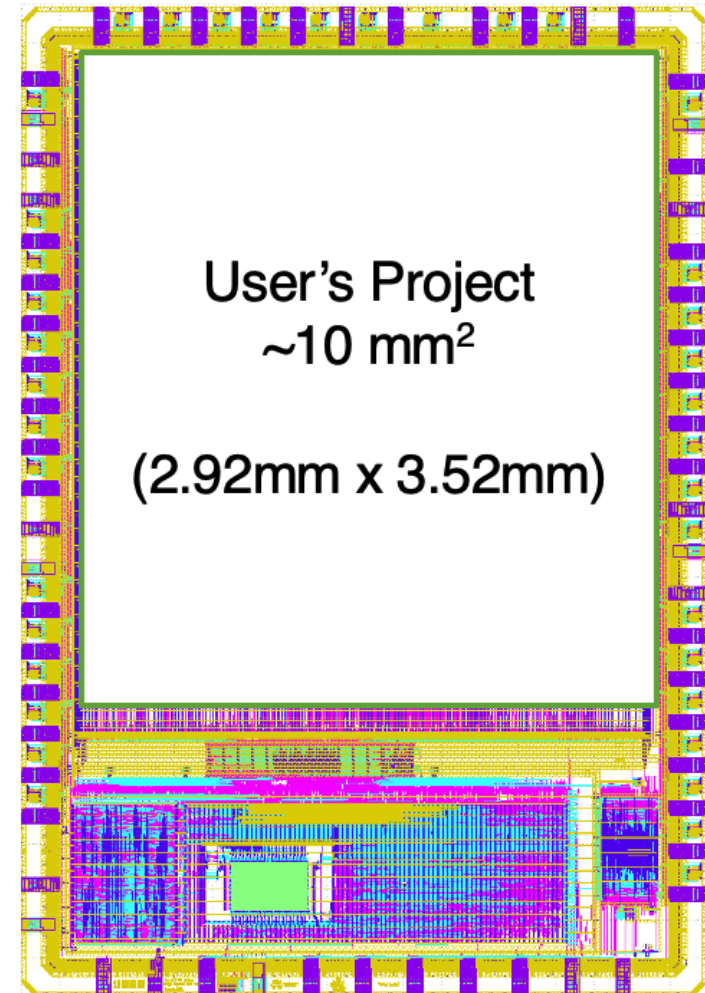
C2S2: Cornell Custom Silicon Systems Project Team

Three-year student-led project team to tapeout a custom chip in SkyWater 130nm to implement a proof-of-concept system for a campus partner

- ▶ Open RISC-V ISA
- ▶ Open-Source VexRISCV microcontroller
- ▶ Open-Source OpenROAD chip flow
- ▶ Open PDK for SkyWater 130nm
- ▶ OpenMPW + Chiplgnite w/ efabless

100+ applications → 25 team members

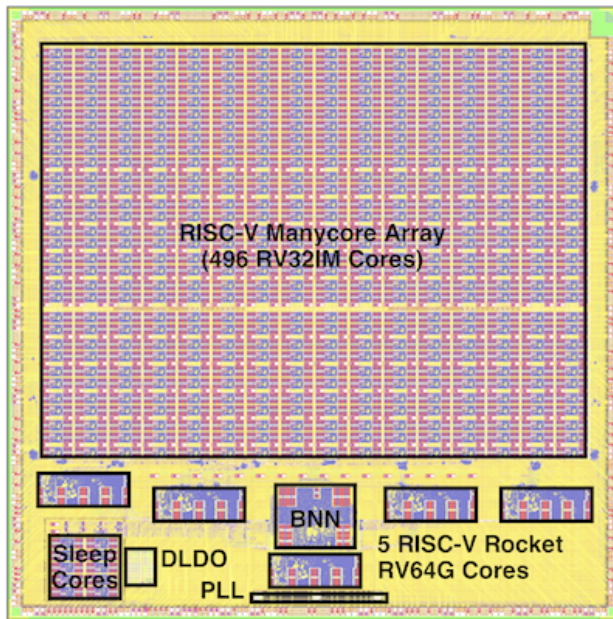
- ▶ Digital & Verification Subteam
- ▶ Analog Subteam
- ▶ Software Subteam
- ▶ System Architecture Subteam




```

1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire  ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14         @update
15         def comb_logic():
16             s.out @= s.tmp + 1

```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation

Gradually-Typed HDLs

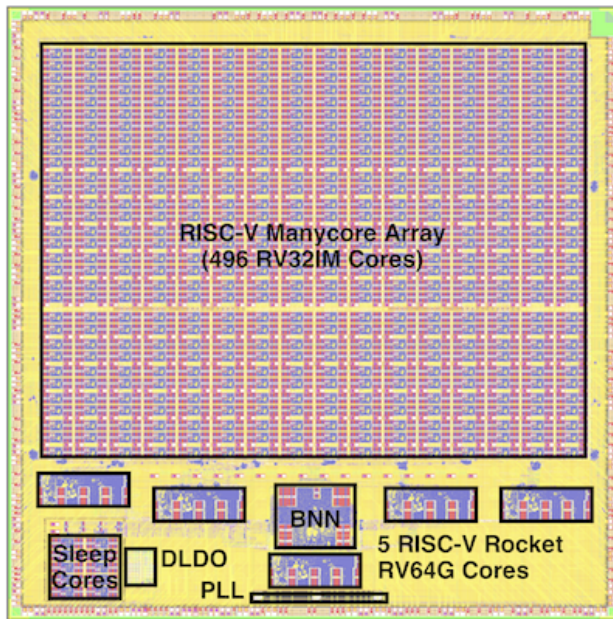
Property-Based Random Testing

A Call to Action

```

1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire   ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14         @update
15         def comb_logic():
16             s.out @= s.tmp + 1

```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation

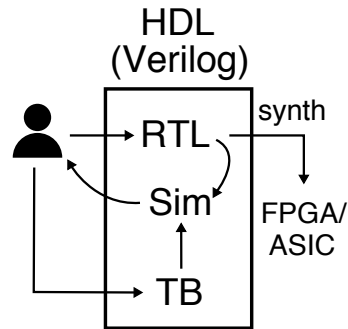
Gradually-Typed HDLs

Property-Based Random Testing

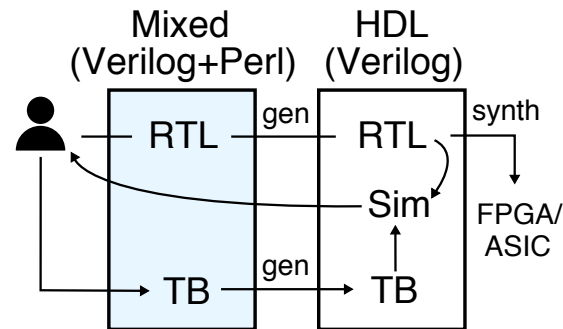
A Call to Action

Traditional Hardware Design Methodologies

HDL Hardware Description Language

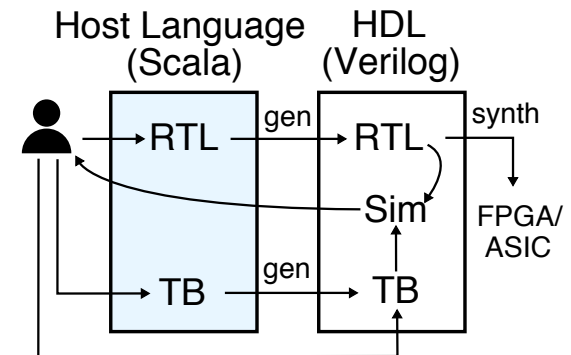


HPF Hardware Preprocessing Framework



Example: Genesis2

HGF Hardware Generation Framework



Example: Chisel

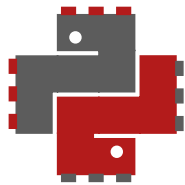
- ✓ Fast edit-sim-debug loop
- ✓ Single language for structural, behavioral, + TB
- ✗ Difficult to create highly parameterized generators

- ✗ Slower edit-sim-debug loop
- ✗ Multiple languages create "semantic gap"
- ✓ Easier to create highly parameterized generators

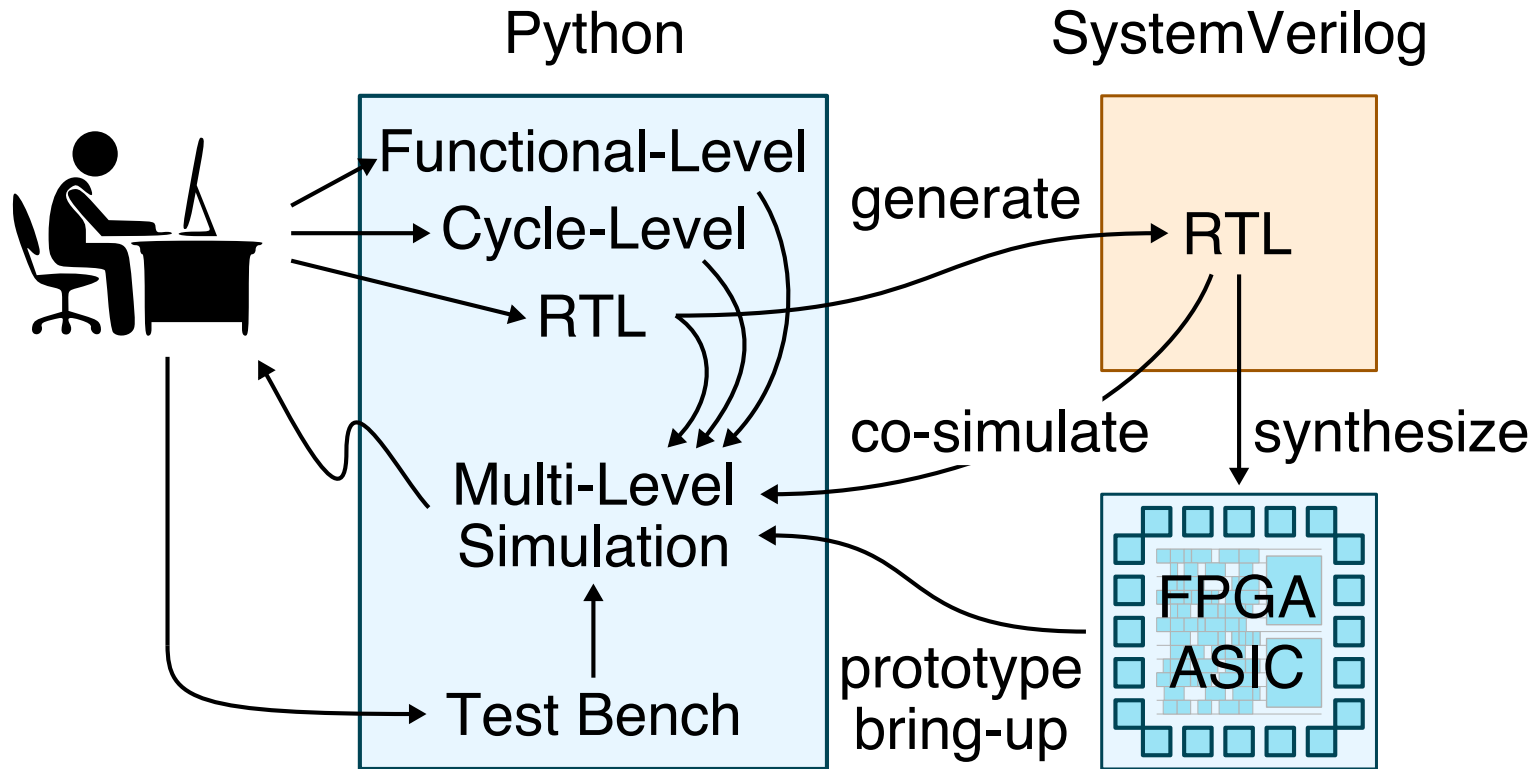
- ✗ Slower edit-sim-debug loop
- ✓ Single language for structural + behavioral
- ✓ Easier to create highly parameterized generators
- ✗ Cannot use power of host language for verification

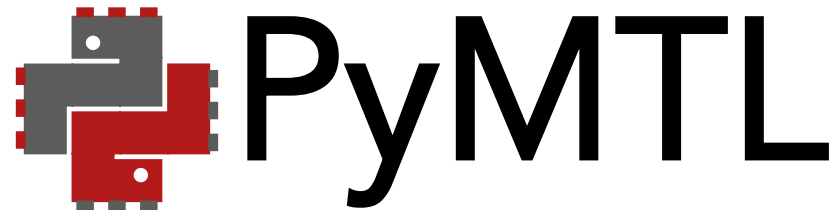
Are HGFs the best we can do in terms of a **productive** hardware design methodology?

PyMTL



Python-based hardware generation, simulation, and verification framework which enables productive RTL design and multi-level modeling

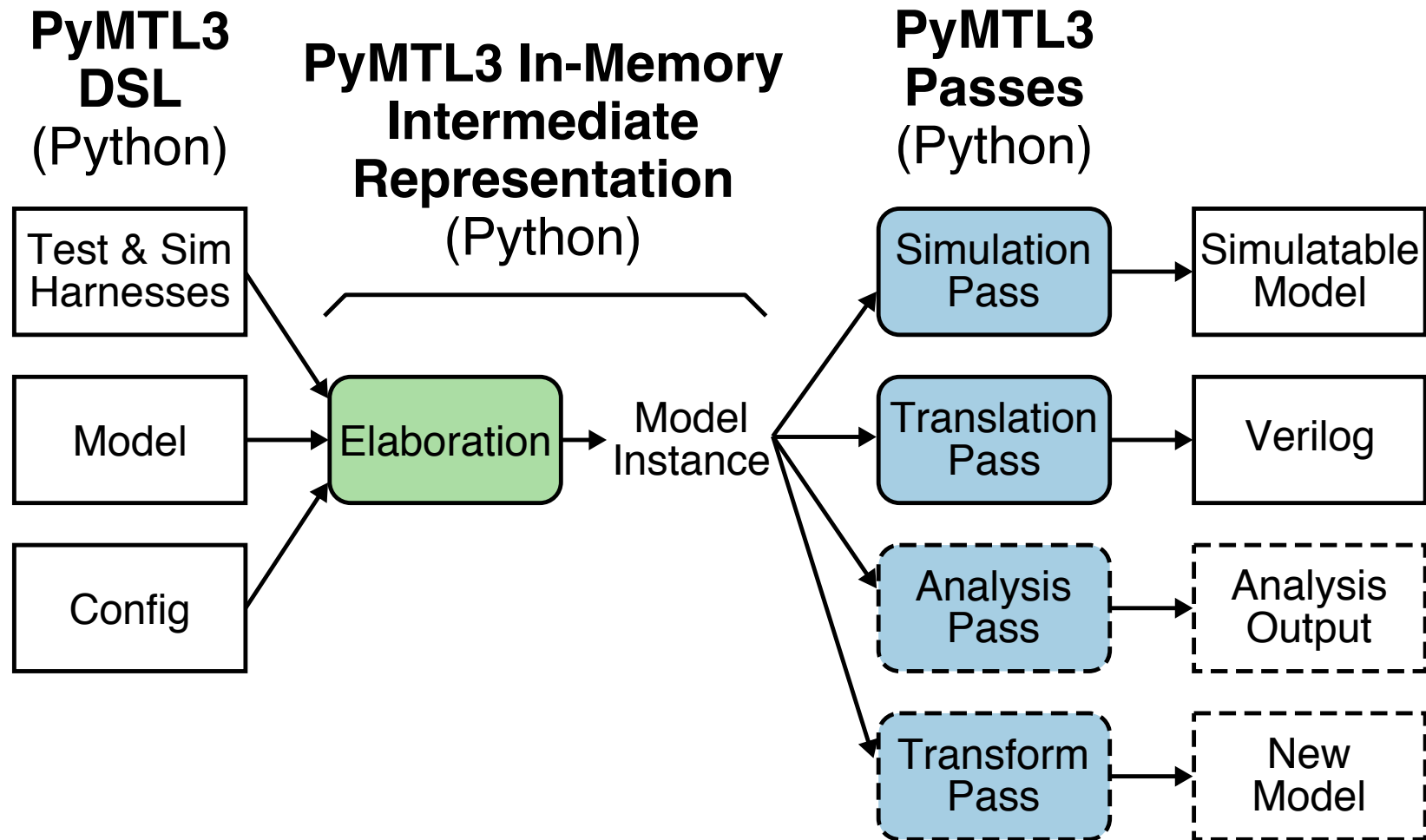




- ▶ **PyMTL2**: <https://github.com/cornell-brg/pymtl>
 - ▷ released in 2014
 - ▷ extensive experience using framework in research & teaching

- ▶ **PyMTL3**: <https://github.com/pymtl/pymtl3>
 - ▷ official release in May 2020
 - ▷ adoption of new Python3 features
 - ▷ significant rewrite to improve productivity & performance
 - ▷ cleaner syntax for FL, CL, and RTL modeling
 - ▷ completely new Verilog translation support
 - ▷ first-class support for method-based interfaces

The PyMTL3 Framework



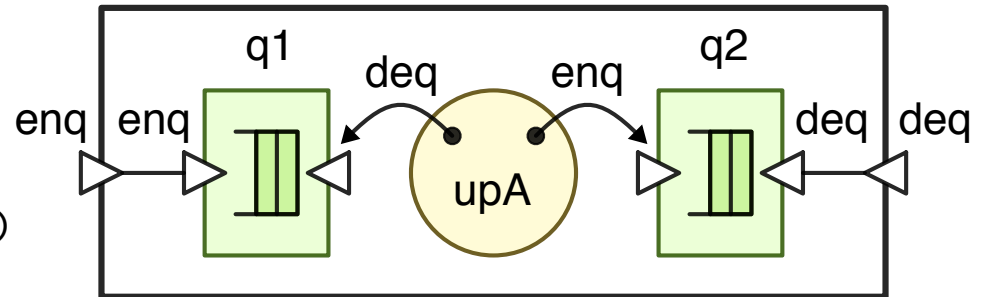
PyMTL3 High-Level Modeling

```

1 class QueueFL( Component ):
2     def construct( s, maxsize ):
3         s.q = deque( maxlen=maxsize )
4
5     @non_blocking(
6         lambda s: len(s.q) < s.q.maxlen )
7     def enq( s, value ):
8         s.q.appendleft( value )
9
10    @non_blocking(
11        lambda s: len(s.q) > 0 )
12    def deq( s ):
13        return s.q.pop()

```

- ▶ FL/CL components can use method-based interfaces
- ▶ Structural composition via connecting methods



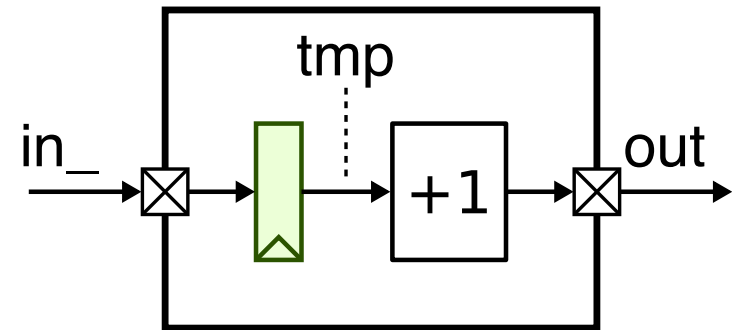
```

14 class DoubleQueueFL( Component ):
15     def construct( s ):
16         s.enq = CalleeIfcCL()
17         s.deq = CalleeIfcCL()
18
19         s.q1 = QueueFL(2)
20         s.q2 = QueueFL(2)
21
22         connect( s.enq,      s.q1.enq )
23         connect( s.q2.deq,  s.deq )
24
25     @update
26     def upA():
27         if s.q1.deq.rdy() and s.q2.enq.rdy():
28             s.q2.enq( s.q1.deq() )

```

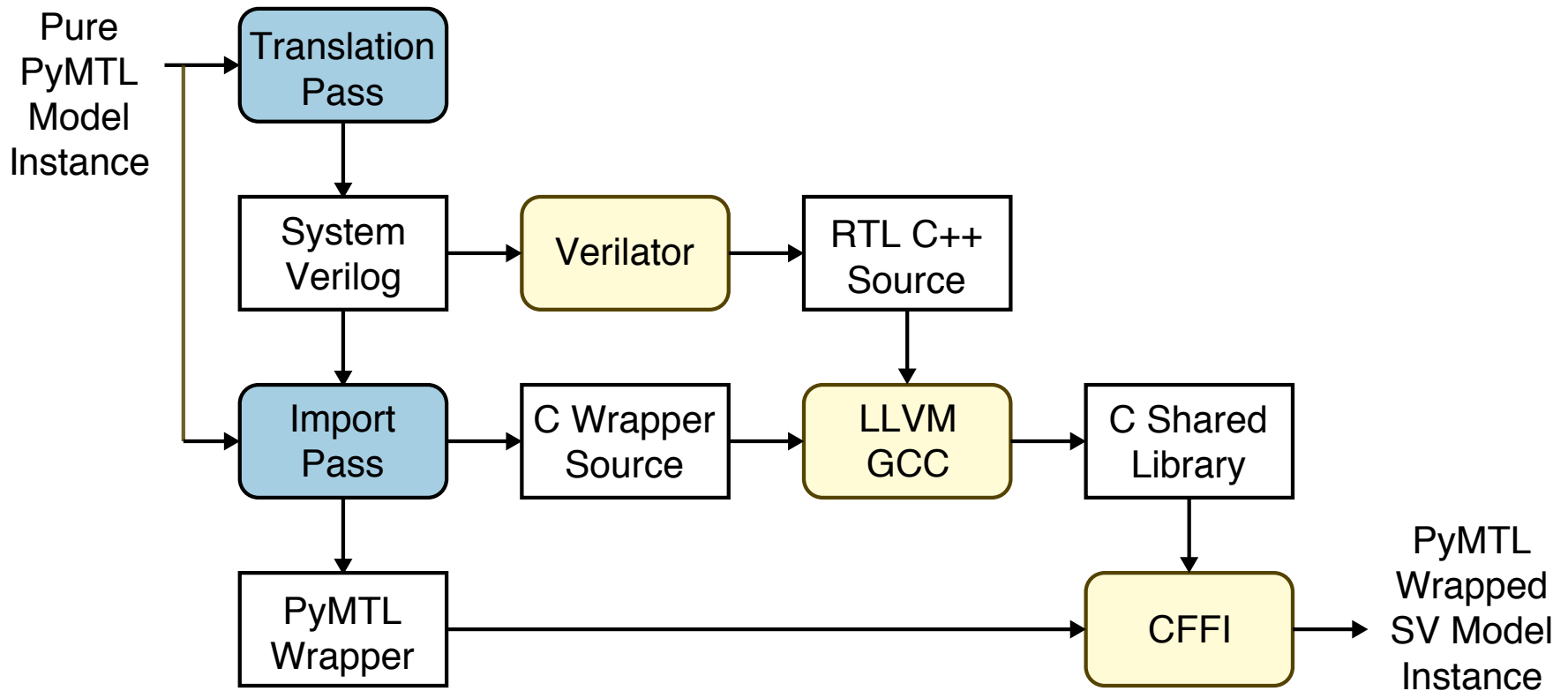
PyMTL3 Low-Level Modeling

```
1 from pymtl3 import *
2
3 class RegIncrRTL( Component ):
4
5     def construct( s, nbits ):
6         s.in_ = InPort ( nbits )
7         s.out = OutPort( nbits )
8         s.tmp = Wire   ( nbits )
9
10        @update_ff
11        def seq_logic():
12            s.tmp <<= s.in_
13
14        @update
15        def comb_logic():
16            s.out @= s.tmp + 1
```



- ▶ Hardware modules are Python classes derived from Component
- ▶ construct method for constructing (elaborating) hardware
- ▶ ports and wires for signals
- ▶ update blocks for modeling combinational and sequential logic

SystemVerilog Translation and Import



- ▶ Translation+import enables easily testing translated SystemVerilog
- ▶ Also acts like a JIT compiler for improved RTL simulation speed
- ▶ Can also import external SystemVerilog IP for co-simulation

What is PyMTL3 for and not (currently) for?

▶ **PyMTL3 is for ...**

- ▷ Taking an accelerator design from concept to implementation
- ▷ Construction of highly-parameterizable CL models
- ▷ Construction of highly-parameterizable RTL design generators
- ▷ Rapid design, testing, and exploration of hardware mechanisms
- ▷ Interfacing models with other C++ or Verilog frameworks

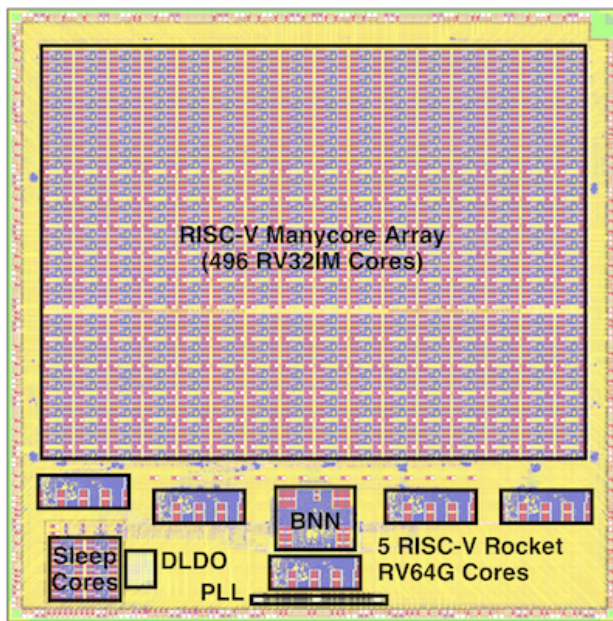
▶ **PyMTL3 is not (currently) for ...**

- ▷ Python high-level synthesis
- ▷ Many-core simulations with hundreds of cores
- ▷ Full-system simulation with real OS support
- ▷ Users needing a complex OOO processor model “out of the box”

```

1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire  ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14         @update
15         def comb_logic():
16             s.out @= s.tmp + 1

```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation

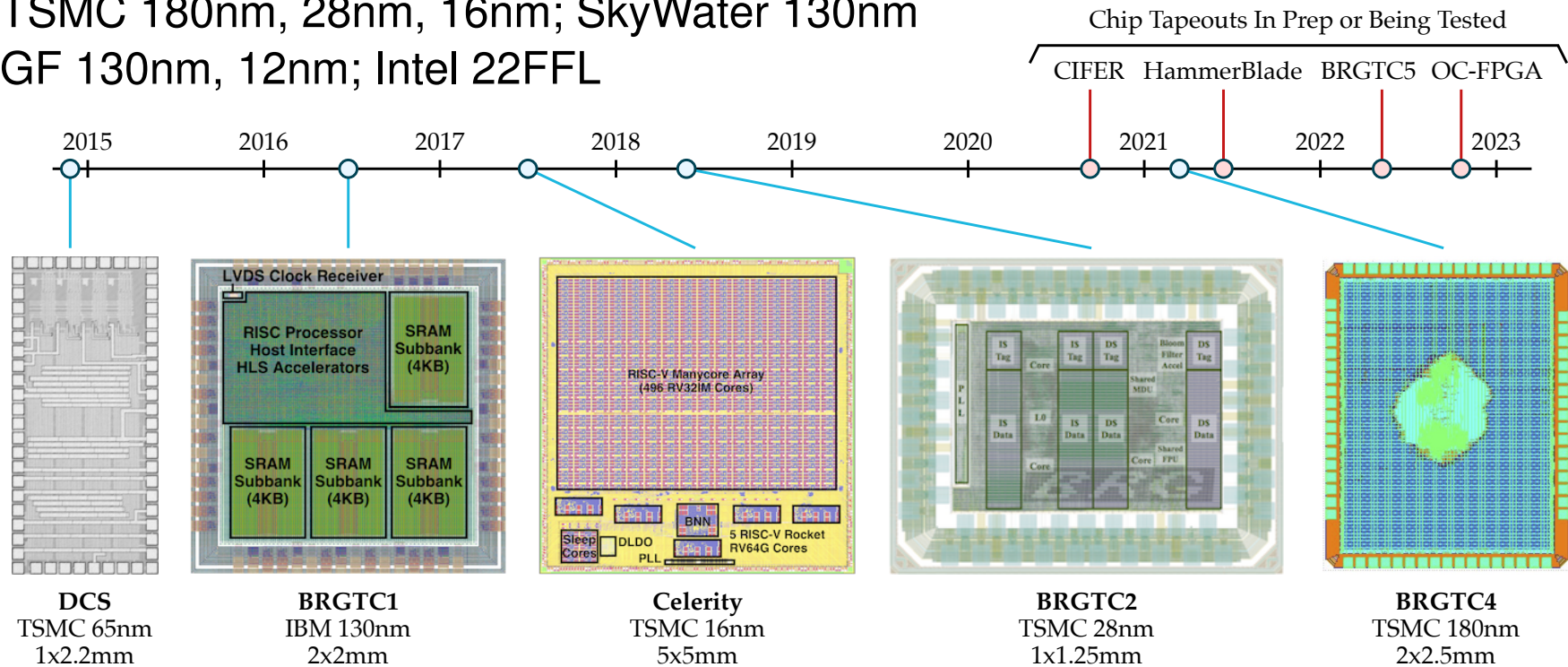
Gradually-Typed HDLs

Property-Based Random Testing

A Call to Action

PyMTL has been used in many chip tapeouts

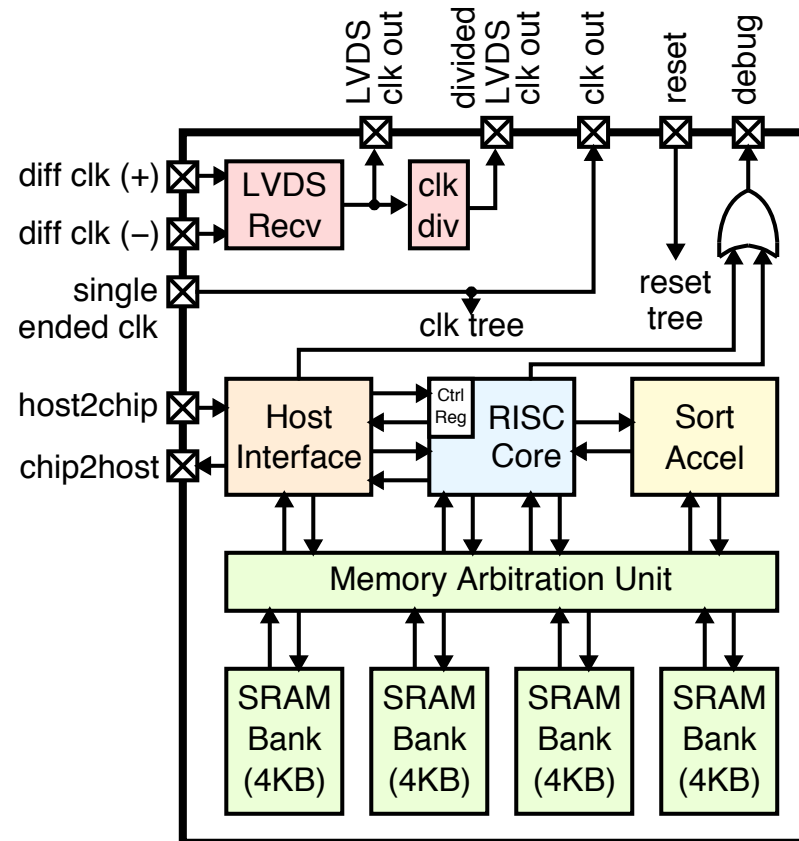
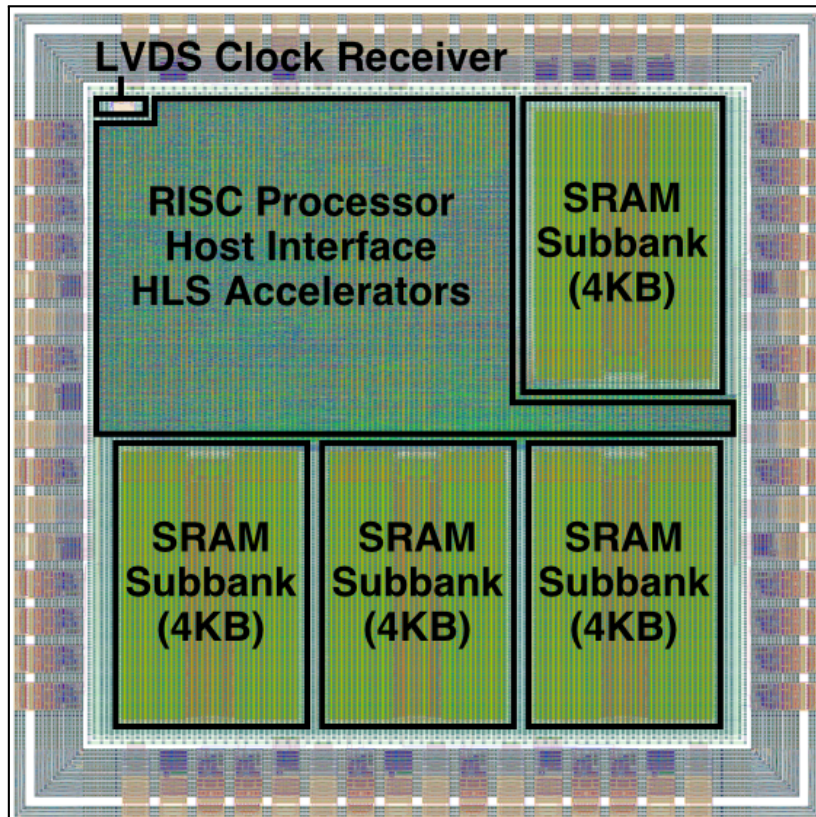
TSMC 180nm, 28nm, 16nm; SkyWater 130nm
 GF 130nm, 12nm; Intel 22FFL



- ▶ Simple RISC-V cores
- ▶ Coarse-grain reconfigurable arrays
- ▶ Clustered manycore architectures

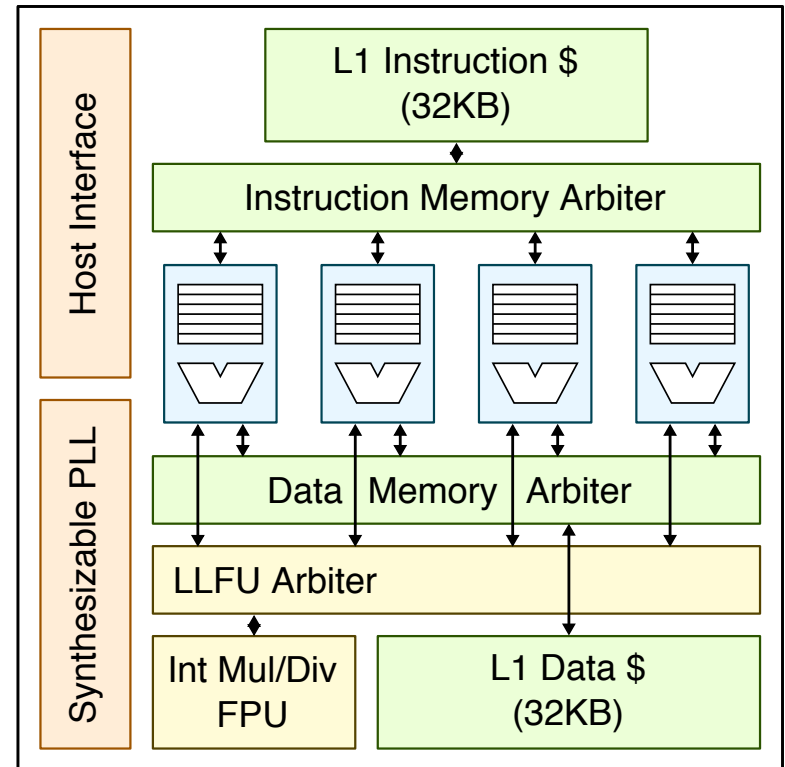
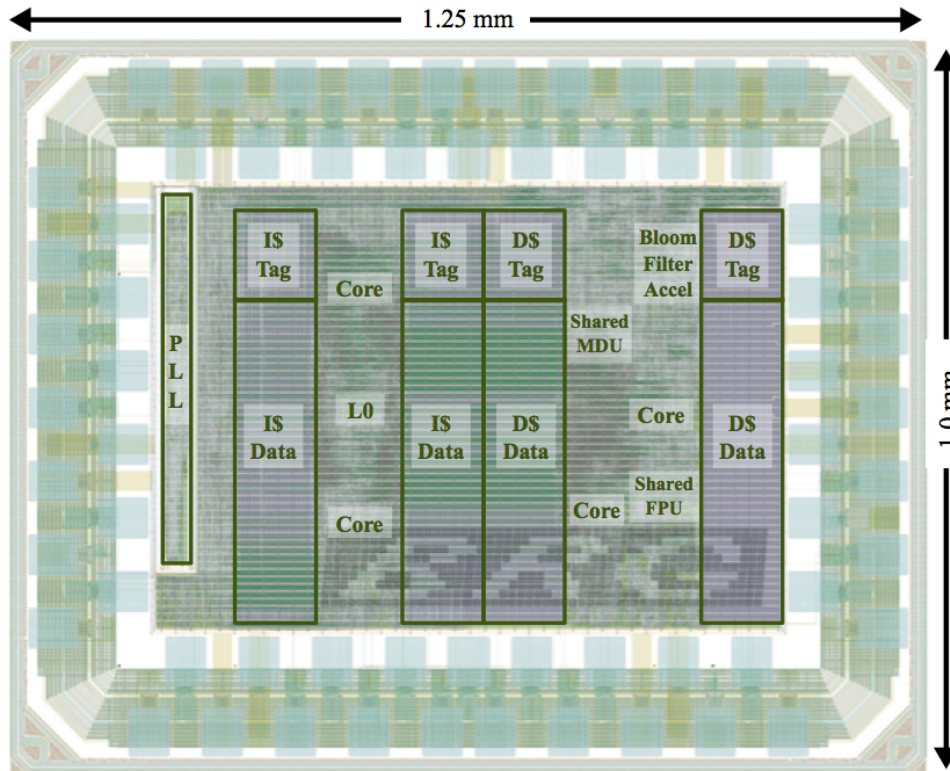
- ▶ Mesh on-chip networks
- ▶ Crossbar interconnects

BRG Test Chip #1 (2016)



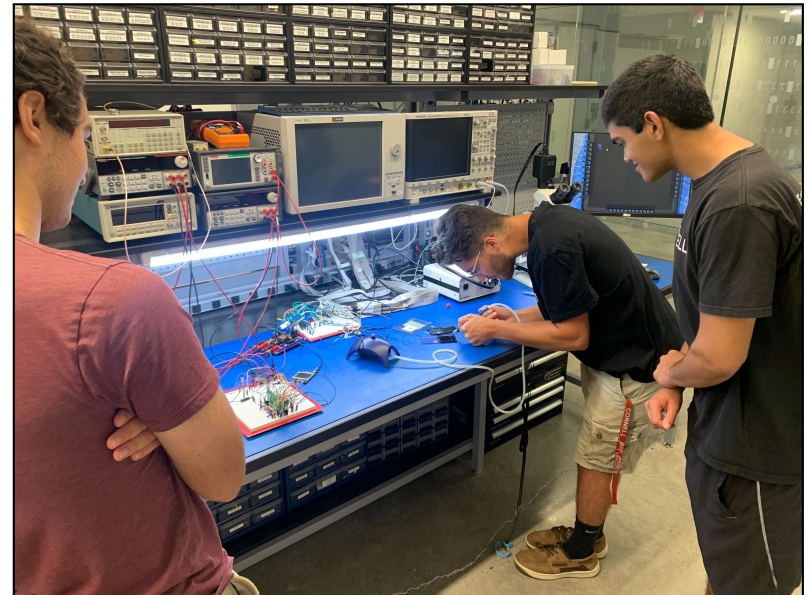
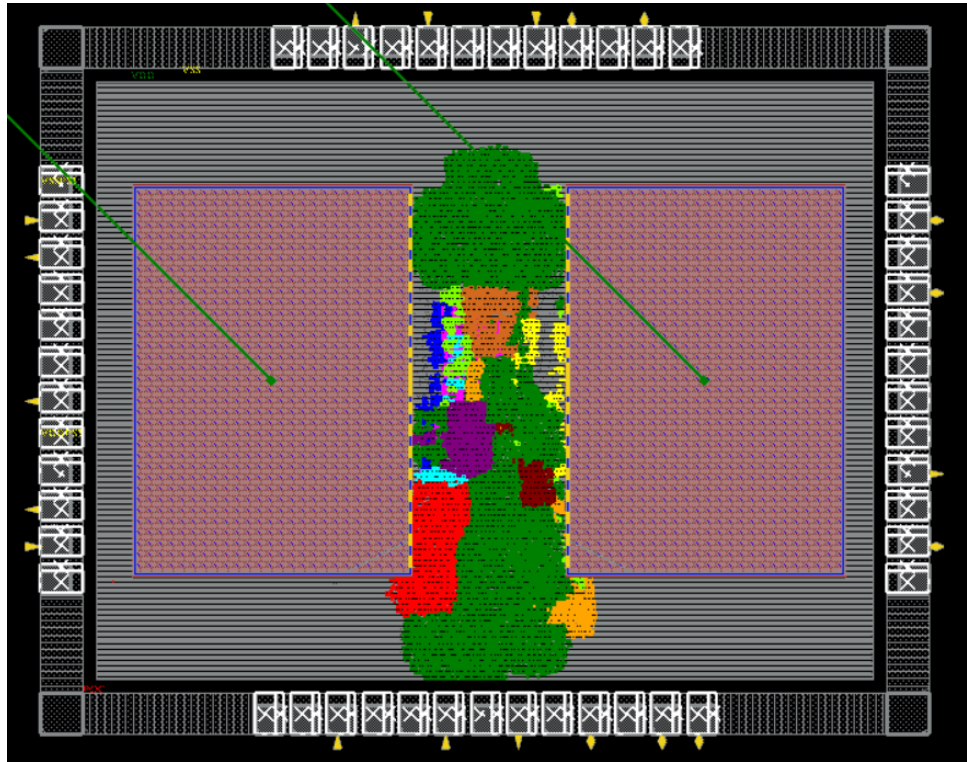
RISC processor, 16KB SRAM, HLS-generated accelerator
 2x2mm, 1.2M-trans, IBM 130nm
 95% done using PyMTL2

BRG Test Chip #2 (2018)



Four RISC-V RV32IMAF cores with “smart” sharing of L1\$/LLFU
 1x1.2mm, 6.7M-trans, TSMC 28nm
 95% done using PyMTL2

BRG Test Chip #5 (2022)

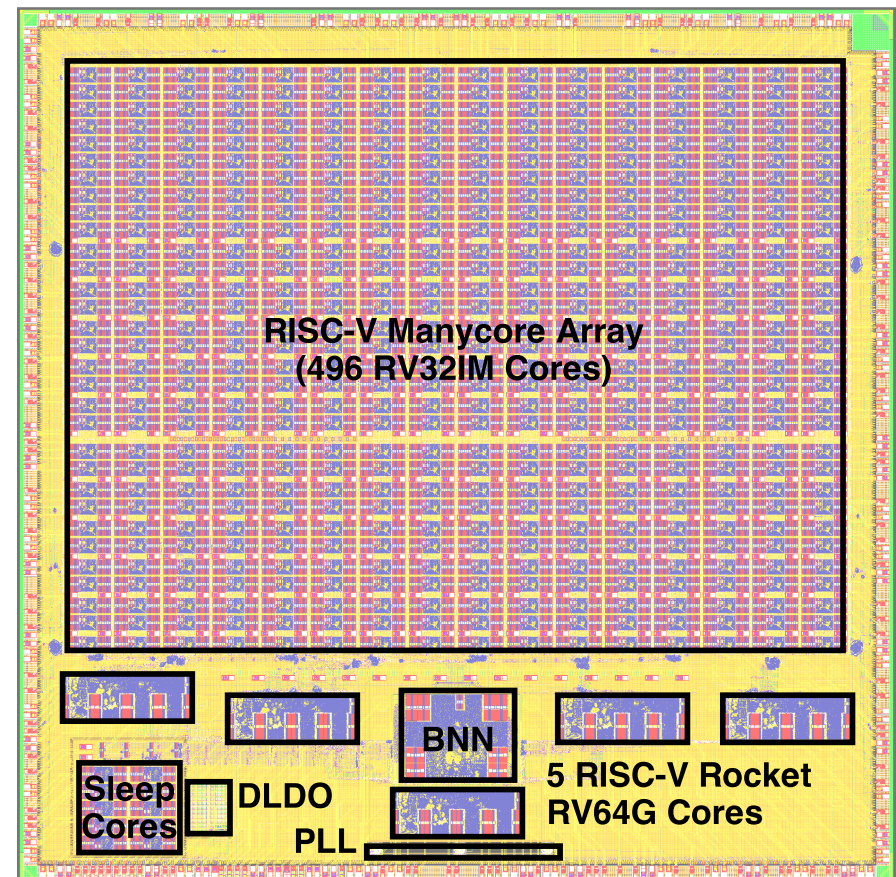


RISC-V RV32IM core with 32-KB of SRAM
SPI minion for config; SPI master and GP I/O for peripherals
2x2.5mm, TSMC 180nm
100% done using PyMTL3

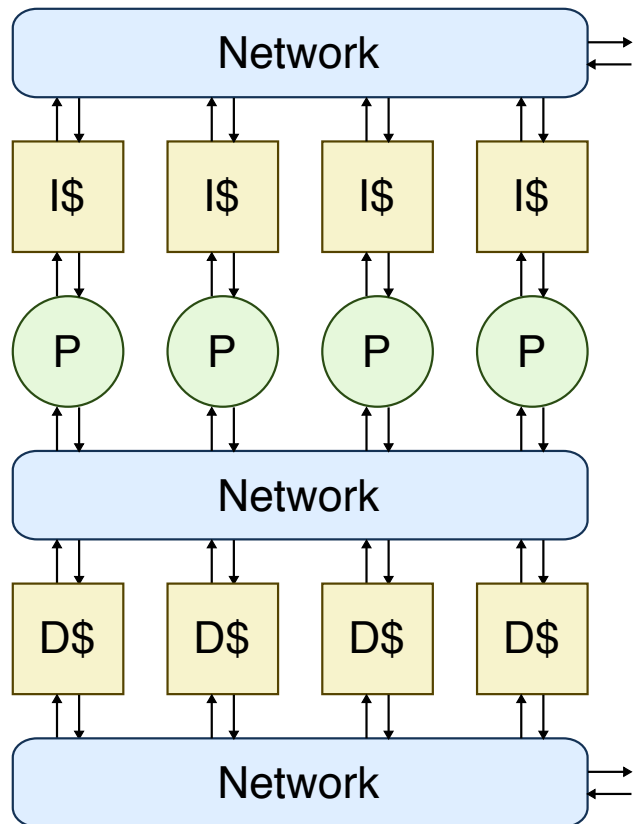
Celerity System-on-Chip

Target Workload: High-Performance Embedded Computing

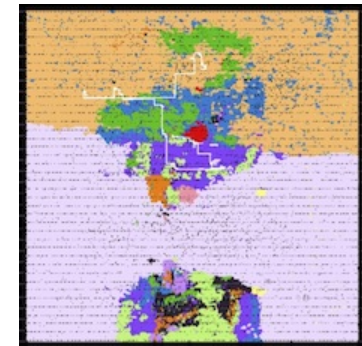
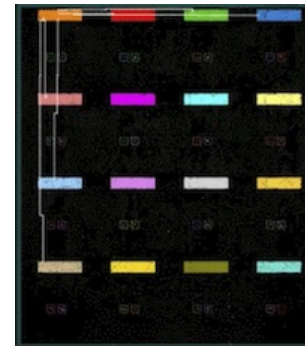
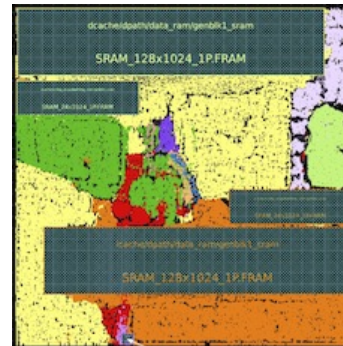
- ▶ 5×5 mm in TSMC 16 nm FFC
- ▶ 385 million transistors
- ▶ 511 RISC-V cores
 - ▷ 5 Linux-capable Rocket cores
 - ▷ 496-core tiled manycore
 - ▷ 10-core low-voltage array
- ▶ 1 BNN accelerator
- ▶ 1 synthesizable PLL
- ▶ 1 synthesizable LDO Vreg
- ▶ 3 clock domains
- ▶ 672-pin flip chip BGA package
- ▶ 9-months from PDK access to tape-out



PyMTL3 for Undergraduate and Graduate Courses

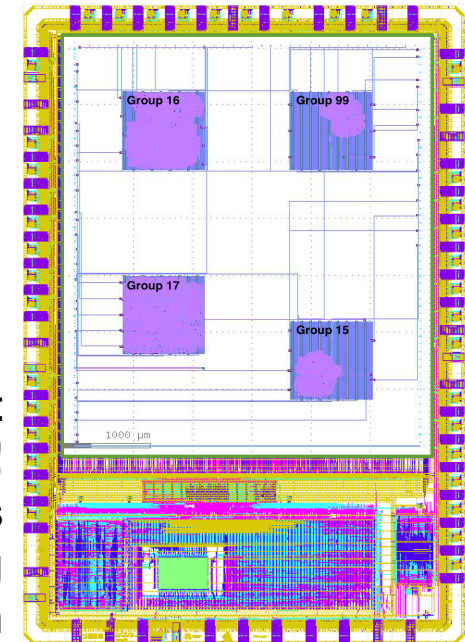


Computer Arch Course
Labs use PyMTL for verification,
PyMTL or Verilog for RTL design



Chip Design Course
Labs use PyMTL for
verification, PyMTL or
Verilog for RTL design,
standard ASIC flow

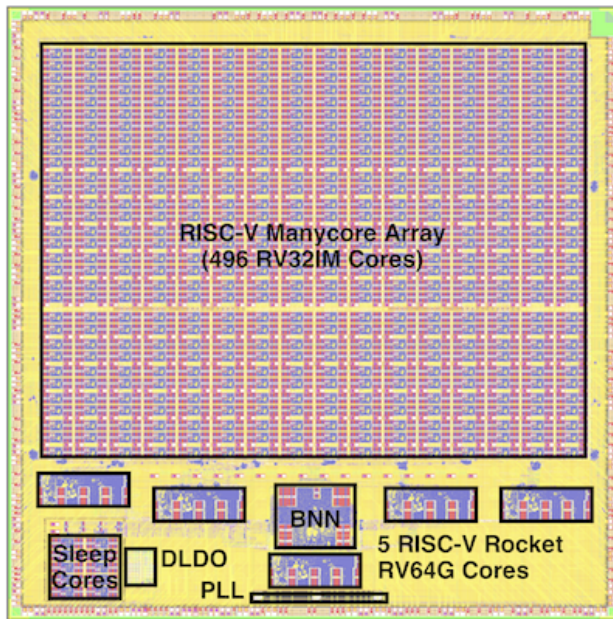
**First Teaching Tapeout
in 10+ years!**
Four student projects
All use PyMTL for testing
Two use PyMTL for design



```

1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire  ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14         @update
15         def comb_logic():
16             s.out @= s.tmp + 1

```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation

Gradually-Typed HDLs

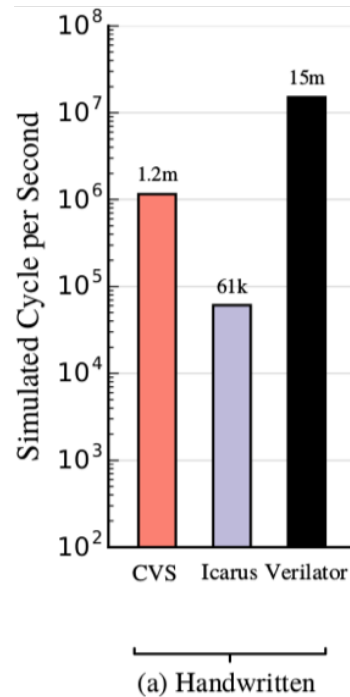
Property-Based Random Testing

A Call to Action

Evaluating HDLs, HGFs, and HGSFs

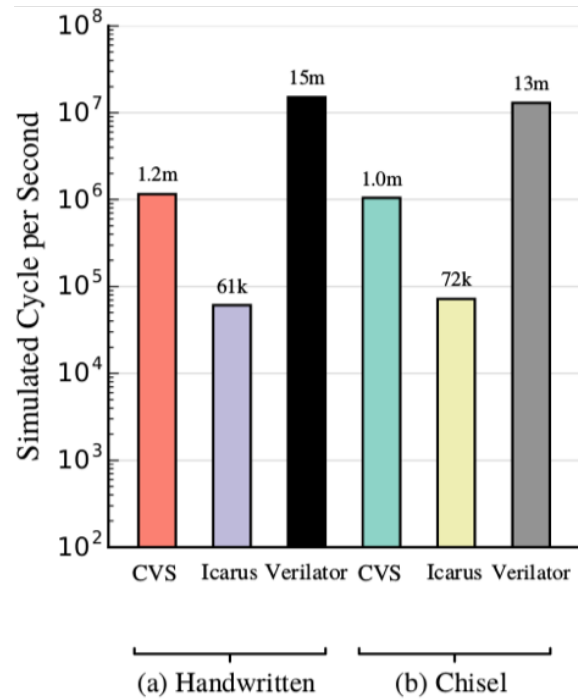
- ▶ Apple-to-apple comparison of simulator performance
- ▶ 64-bit radix-four integer iterative divider
- ▶ All implementations use same control/datapath split with the same level of detail
- ▶ Modeling and simulation frameworks:
 - ▷ Verilog: Commercial verilog simulator, Icarus, Verilator
 - ▷ HGF: Chisel
 - ▷ HGSFs: PyMTL, MyHDL, PyRTL, Migen

Productivity/Performance Gap



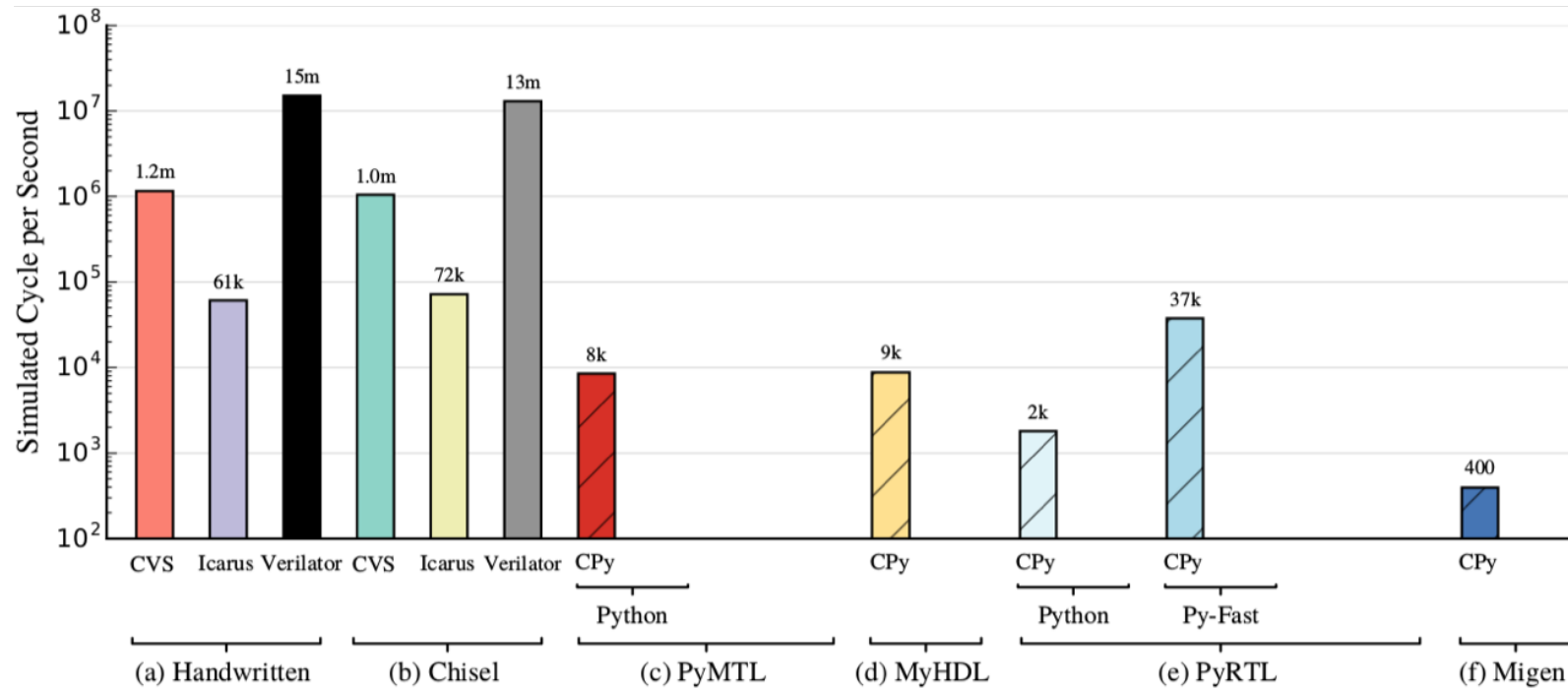
- ▶ Higher is better
- ▶ Log scale (gap is larger than it seems)
- ▶ Commercial Verilog simulator is $20\times$ faster than Icarus
- ▶ Verilator requires C++ testbench, only works with synthesizable code, takes significant time to compile, but is $200\times$ faster than Icarus

Productivity/Performance Gap



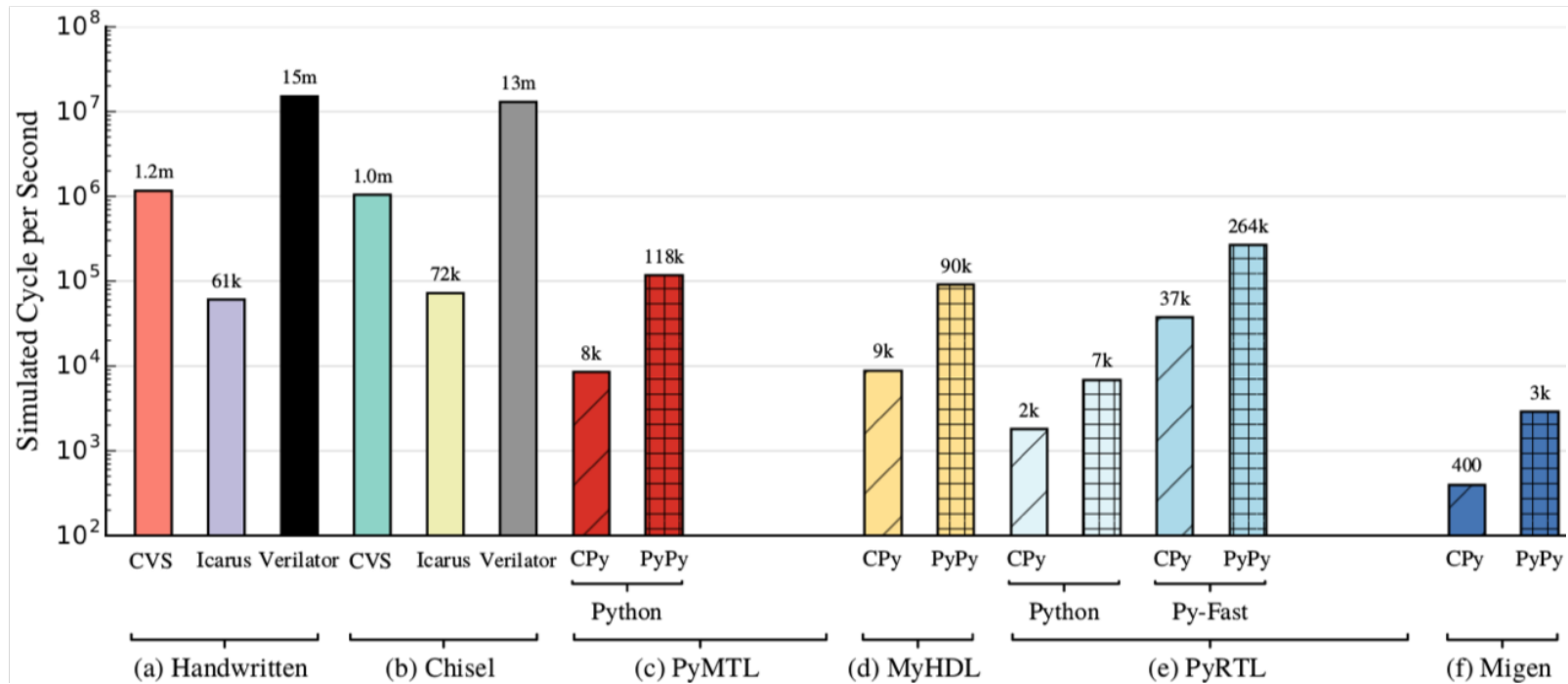
- ▶ Chisel (HGF) generates Verilog and uses Verilog simulator

Productivity/Performance Gap



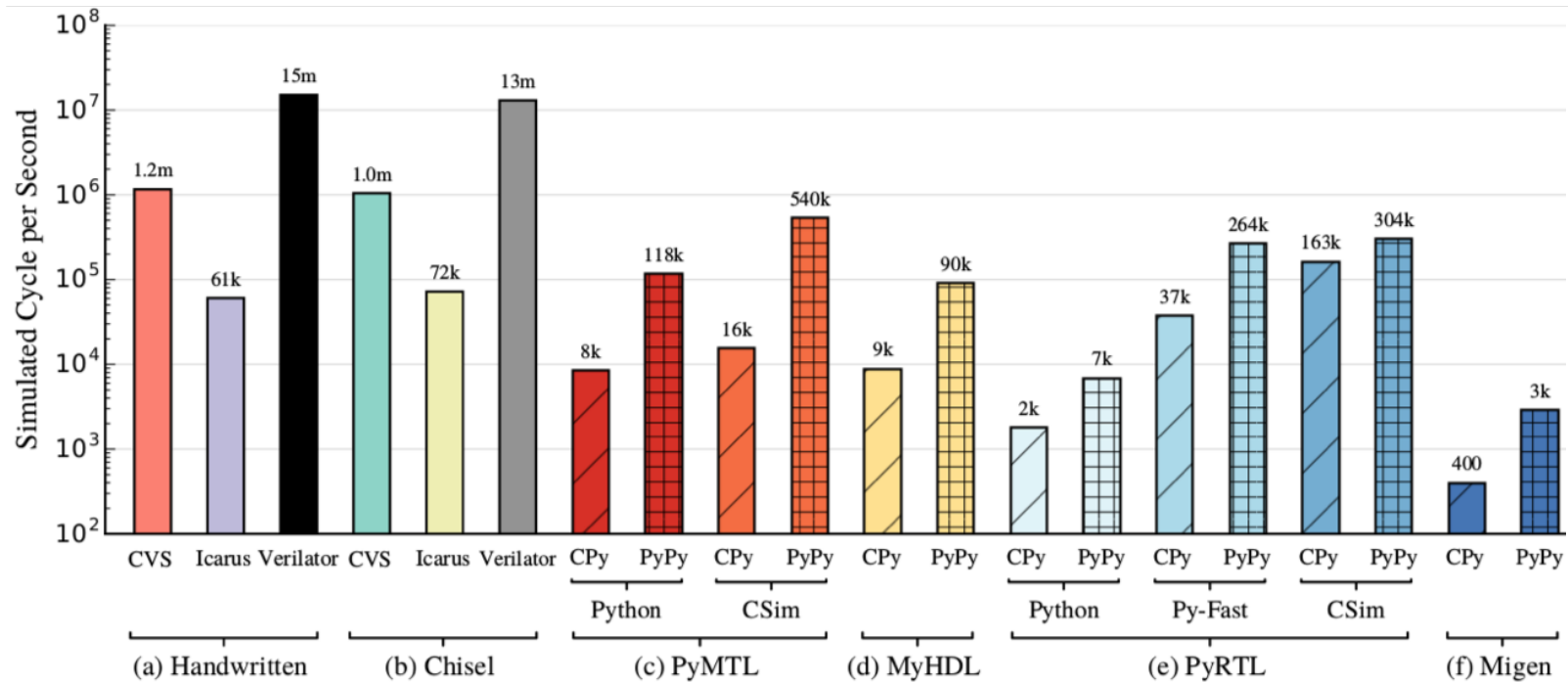
- ▶ Using CPython interpreter, Python-based HGSFs are much slower than commercial Verilog simulators; even slower than Icarus!

Productivity/Performance Gap



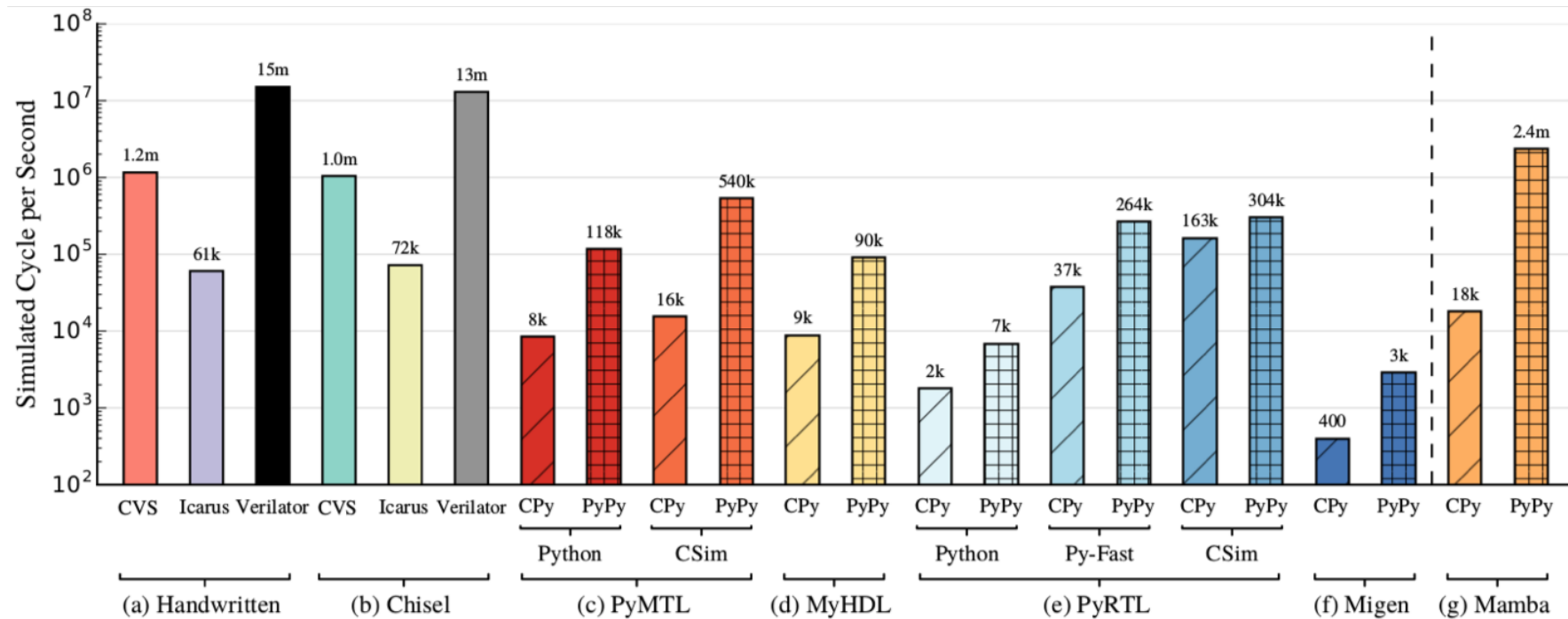
- ▶ Using PyPy JIT compiler, Python-based HGSFs achieve $\approx 10\times$ speedup, but still significantly slower than commercial Verilog simulator

Productivity/Performance Gap



- ▶ Hybrid C/C++ co-simulation improves performance but:
 - ▷ only works for a synthesizable subset
 - ▷ may require designer to simultaneously work with C/C++ and Python

Productivity/Performance Gap



- ▶ PyMTL3 achieves impressive simulation performance by co-optimizing the framework and JIT

PyMTL3 Performance

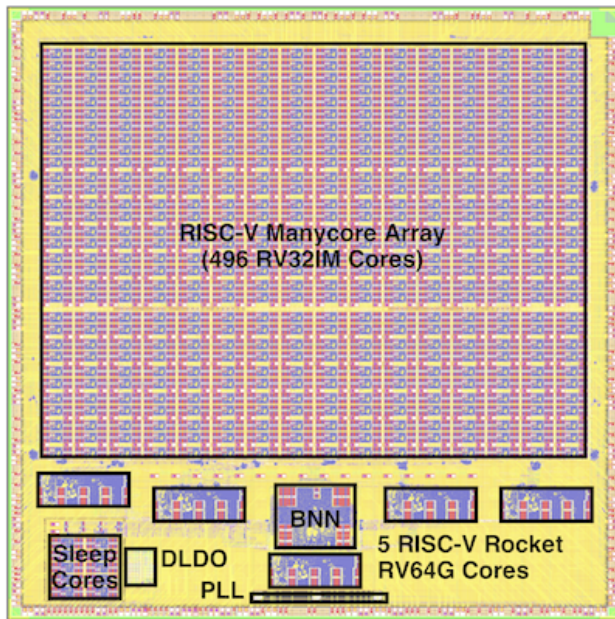
Technique	Divider	1-Core	16-core	32-core
Event-Driven	24K CPS	6.6K CPS	155 CPS	66 CPS
JIT-Aware HGSF				
+ Static Scheduling	13×	2.6×	1×	1.1×
+ Schedule Unrolling	16×	24×	0.4×	0.2×
+ Heuristic Toposort	18×	26×	0.5×	0.3×
+ Trace Breaking	19×	34×	2×	1.5×
+ Consolidation	27×	34×	47×	42×
HGSF-Aware JIT				
+ RPython Constructs	96×	48×	62×	61×
+ Huge Loop Support	96×	49×	65×	67×

- ▶ RISC-V RV32IM five-stage pipelined cores
- ▶ Only models cores, no interconnect nor caches

```

1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire  ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14         @update
15         def comb_logic():
16             s.out @= s.tmp + 1

```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research





















JIT-Compiled Simulation

Gradually-Typed HDLs

Property-Based Random Testing

A Call to Action

Dynamically vs. Statically Typed HDLs

	Design Productivity	Testing Productivity	Simulation Performance	Static Correctness Guarantees
Verilog/SystemVerilog	Low 	Low 	High 	Low 
Bluespec	Medium 	Low 	Medium 	High 
Clash/Chisel/SpinalHDL	Medium 	Low 	Medium 	Medium 
PyRTL/MyHDL/Migen/ PyMTL	High 	High 	Low 	None 
PyMTL3	High 	High 	Medium 	Low 

Can we achieve the best of both dynamically and statically typed HDLs in a single unified framework?

Gradually Typed HDLs

```

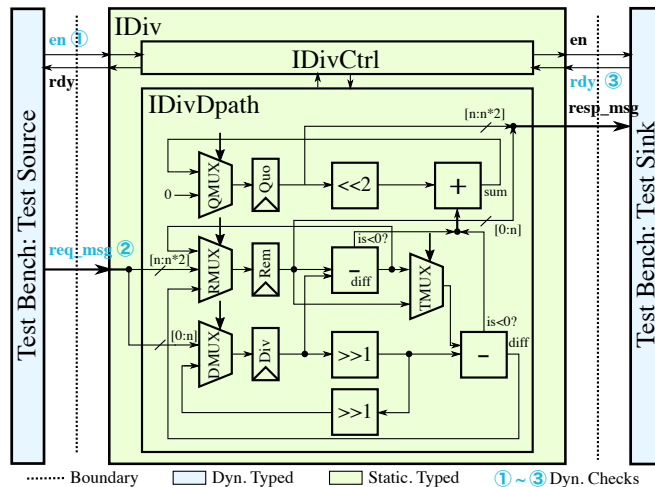
1 class Foo:
2     bar = 42
3     def g(x):
4         x.bar = 'hello world'
5     def f(x:Object({bar:Int})) -> Int:
6         g(x)
7         return x.bar
8 f(Foo())

```

Dynamic

Static

Code in Reticulated Python,
a Gradually Typed Dialect of Python



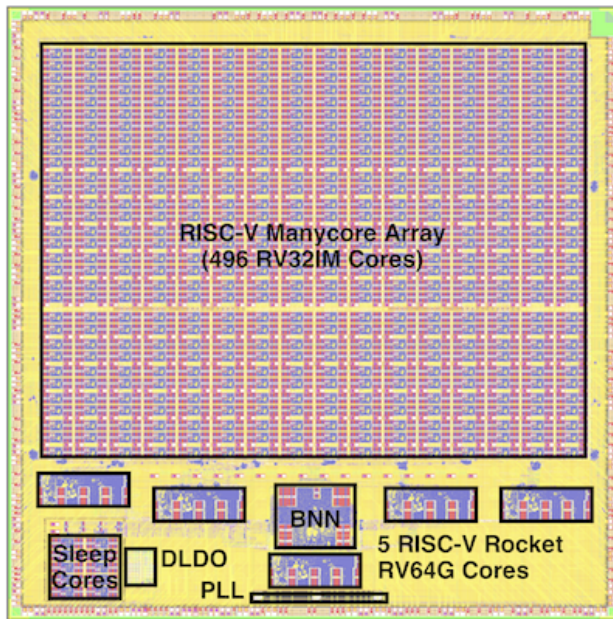
Component Hierarchy in GT-HDL

```

1 from pymtl3 import *
2
3 T_W = \
4     TypeVar( "T_W", bound=Bits )
5
6 class RegIncrRTL( Component ):
7
8     def construct( s, W: Type[T_W] ):
9         s.in_ = InPort ( W )
10        s.out = OutPort( W )
11        s.tmp = Wire   ( W )
12
13    @update_ff
14    def seq_logic():
15        s.tmp <<= s.in_
16
17    @update
18    def comb_logic():
19        s.out @= s.tmp + 1

```

```
1 from pymtl3 import *
2
3 class RegIncrRTL( Component ):
4
5     def construct( s, nbits ):
6         s.in_ = InPort( nbits )
7         s.out = OutPort( nbits )
8         s.tmp = Wire ( nbits )
9
10        @update_ff
11        def seq_logic():
12            s.tmp <<= s.in_
13
14        @update
15        def comb_logic():
16            s.out @= s.tmp + 1
```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation

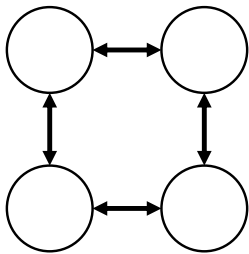
Gradually-Typed HDLs

Property-Based Random Testing

A Call to Action

Testing HW Design Generators is Challenging

Testing a specific ring network instance requires a number of different test cases

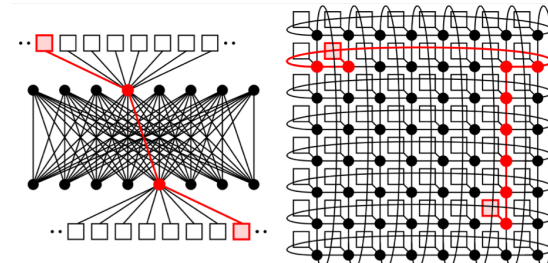


```
test_ring_1pkt_2x2_0_chnl
test_ring_2pkt_2x2_0_chnl
test_ring_2pkt_2x2_0_chnl
test_ring_self_2x2_0_chnl
test_ring_clockwise_2x2_0_chnl
test_ring_aclockwise_2x2_0_chnl
test_ring_neighbor_2x2_0_chnl
test_ring_tornado_2x2_0_chnl
test_ring_backpressure_2x2_0_chnl
...
```

```
pkt( src=0, dst=1, payload=0xdeadbeef )
pkt( src=0, dst=3, payload=0x00000003 )
pkt( src=1, dst=0, payload=0x00010000 )
pkt( src=1, dst=2, payload=0x00010002 )
pkt( src=2, dst=1, payload=0x00020001 )
pkt( src=2, dst=3, payload=0x00020003 )
pkt( src=3, dst=2, payload=0x00030002 )
pkt( src=3, dst=0, payload=0x00030000 )
pkt( src=0, dst=1, payload=0x00001000 )
pkt( src=1, dst=2, payload=0x10002000 )
pkt( src=2, dst=3, payload=0x20003000 )
pkt( src=3, dst=0, payload=0x30000000 )
pkt( src=0, dst=3, payload=0x00003000 )
pkt( src=1, dst=0, payload=0x10000000 )
pkt( src=2, dst=1, payload=0x20001000 )
pkt( src=3, dst=2, payload=0x30002000 )
...
```

Ideal testing technique:

1. Detect error quickly with **small number of test cases**
2. The failing test case has **minimal number of transactions**
3. The bug trace has **simplest transactions**
4. The failing test case has the **simplest design**



A design generator can have many parameters: topology, routing, flow control, channel latency

Software Testing Techniques

▶ Complete Random Testing (CRT)

- ▷ Randomly generate input data
- ▷ Detects error quickly
- ▷ Debug complicated test case

▶ Iterative Deepened Testing (IDT)

- ▷ Gradually increase input complexity
- ▷ Finds bug with simple input
- ▷ Takes many test cases to find bug

▶ Property-Based Testing (PBT)

- ▷ Search strategies, auto shrinking
- ▷ Detects error quickly
- ▷ Produces minimal failing test case
- ▷ Increasingly state-of-the-art in software testing

```
def gcd( a, b ):
    while b > 0:
        a, b = b, a % b
    return a

def test_crt():
    for _ in range( 100 ):
        a = random.randint( 1, 128 )
        b = random.randint( 1, 128 )
        assert gcd( a, b ) == math.gcd( a, b )

def test_idt():
    for a_max in range( 1, 128 ):
        for b_max in range( 1, 128 ):
            assert gcd( a, b ) == math.gcd( a, b )

@hypothesis.given(
    a = hypothesis.strategies.integers( 1, 128 ),
    b = hypothesis.strategies.integers( 1, 128 ),
)
def test_pbt( a, b ):
    assert gcd( a, b ) == math.gcd( a, b )
```


PyH2 Creatively Adopts PBT for SW to Test HW

- ▶ PyH2 combines **PyMTL3**, a unified hardware modeling framework, with **Hypothesis**, a PBT framework for Python software and creates a property-based testing framework for hardware
- ▶ PyH2 leverages PBT to explore not just the input values for an HW design but to also **explore the parameter values** used to configure an HW design generator

	Complete Random Testing	Iterative Deepened Testing	PyH2
Small number of test cases to find bug	✓	X	✓
Small number transactions in bug trace	X	✓	✓
Simple transactions in bug trace	X	✓	✓
Simple design instance for bug trace	X	✓	✓

PyMTL3 Publications

- ▶ Shunning Jiang, et al., “Mamba: Closing the Performance Gap in Productive Hardware Development Frameworks.” *55th ACM/IEEE Design Automation Conf. (DAC)*, June 2018.
- ▶ Shunning Jiang, Peitian Pan, Yanghui Ou, et al., “PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification.” *IEEE Micro*, 40(4):58–66, Jul/Aug. 2020.
- ▶ Shunning Jiang*, Yanghui Ou*, Peitian Pan, et al., “PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies.” *IEEE Design & Test*, 38(2):53–61, Apr. 2021.
- ▶ Shunning Jiang, Yanghui Ou, Peitian Pan, et al., “UMOC: Unified Modular Ordering Constraints to Unify Cycle- and Register-Transfer-Level Modeling.” *58th ACM/IEEE Design Automation Conf. (DAC)*, Dec. 2021.

Theme Article: Agile and Open-Source Hardware

PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

Shunning Jiang, Peitian Pan, Yanghui Ou,
and Christopher Batten
Cornell University

Abstract—In this article, we present PyMTL3, a Python framework for open-source hardware modeling, generation, simulation, and verification. In addition to compelling benefits from using the Python language, PyMTL3 is designed to provide flexible, modular, and extensible workflows for both hardware designers and computer architects. PyMTL3 supports a seamless multilevel modeling environment and carefully designed modular software architecture using a sophisticated in-memory intermediate representation and a collection of passes that analyze, instrument, and transform PyMTL3 hardware models. We believe PyMTL3 can play an important role in jump-starting the open-source hardware ecosystem.

■ **Due to the** breakdown of transistor scaling and the slowdown of Moore’s law, there has been an increasing trend toward energy-efficient

system-on-chip (SoC) design using heterogeneous architectures with a mix of general-purpose and specialized computing engines. Heterogeneous SoCs emphasize both flexible parameterization of a single design block and versatile composition of numerous different design blocks, which have imposed significant challenges to state-of-the-art hardware modeling and

Digital Object Identifier 10.1109/MM.2020.2997638

Date of publication 25 May 2020; date of current version 30 June 2020.

58

0272-1732 © 2020 IEEE

Published by the IEEE Computer Society

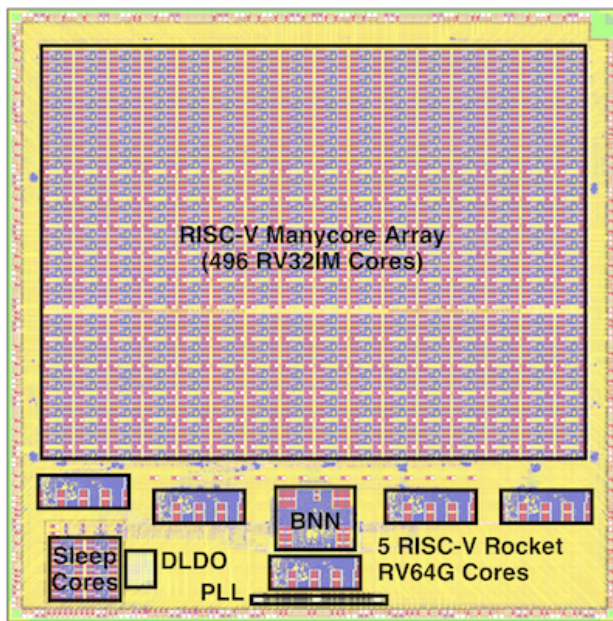
IEEE Micro

Authorized licensed use limited to: Cornell University Library. Downloaded on July 03, 2020 at 00:53:45 UTC from IEEE Xplore. Restrictions apply.

```

1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire  ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14         @update
15         def comb_logic():
16             s.out @= s.tmp + 1

```



A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

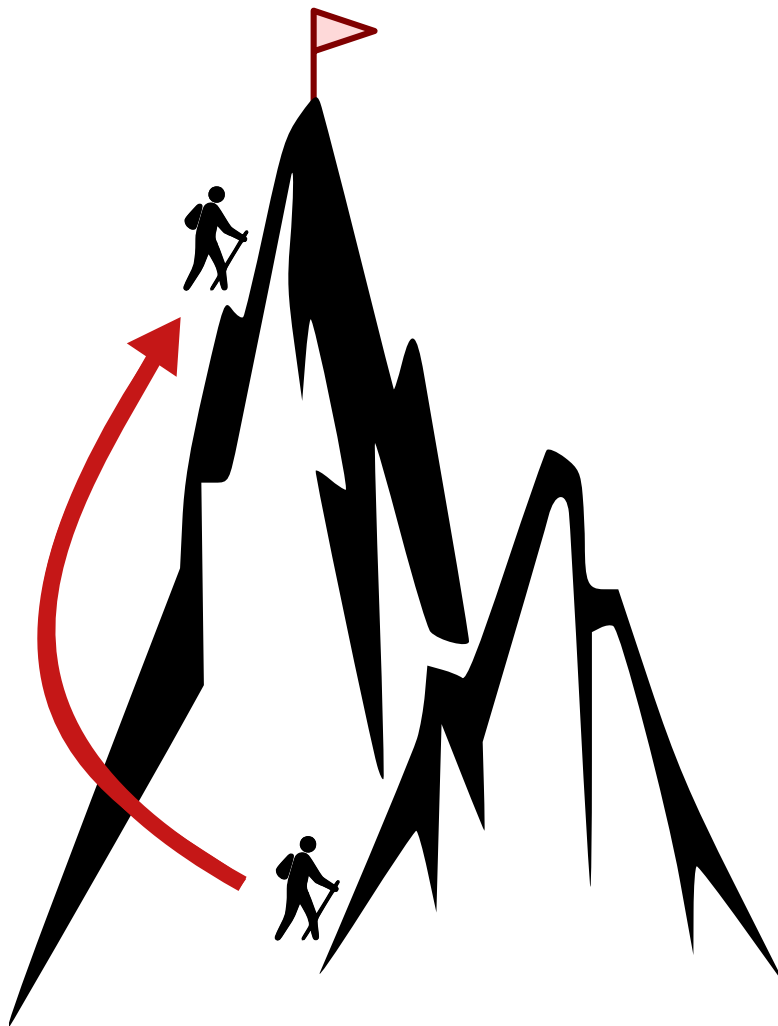
JIT-Compiled Simulation

Gradually-Typed HDLs

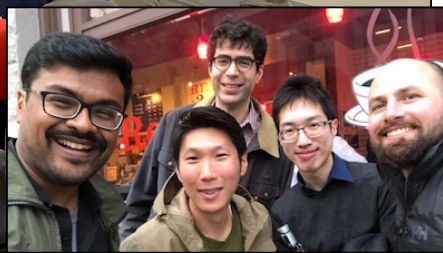
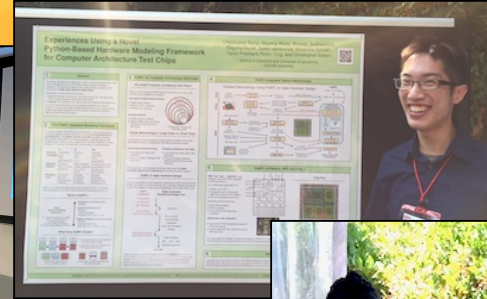
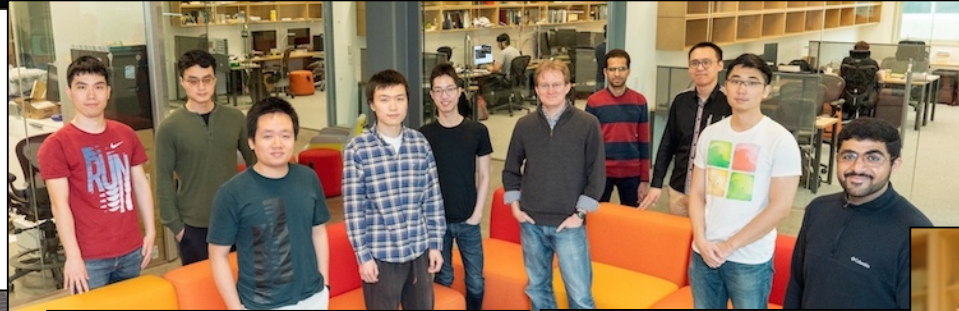
Property-Based Random Testing

A Call to Action

A Call to Action



- ▶ Open-source hardware needs developers who
 - ▷ ... are idealistic
 - ▷ ... have lots of free time
 - ▷ ... will work for free
- ▶ Who might that be?
Students!
- ▶ Academics have a practical and ethical motivation for using, developing, and promoting open-source electronic design automation tools and open-source hardware designs



This work was supported in part by NSF XPS Award #1337240, NSF CRI Award #1512937, NSF SHF Award #1527065, AFOSR YIP Award #FA9550-15-1-0194, DARPA Young Faculty Award #N66001-12-1-4239, DARPA POSH Award #FA8650-18-2-7852, DARPA SDH Award #FA8650-18-2-7863, a Xinux University Program industry gift, and the the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA, and equipment, tool, and/or physical IP donations from Intel, NVIDIA, Synopsys, and ARM.

Thanks to the core PyMTL developers, **Derek Lockhart**, **Shunning Jiang**, **Peitian Pan**, **Yanghui Ou**, along with Khalid Al-Hawaj, Moyang Wang, Tuan Ta, Ji Kim, Shreesha Srinath, Berkin Ilbeyi, Dilan Lakhani, Jack Brzozowski, Kyle Infantino, Yixiao Zhang, Jacob Glueck, Aaron Wisner, Gary Zibrat, Christopher Torng, Cheng Tan, Raymond Yang, Kaishuo Cheng, Carl Friedrich Bolz, David MacIver, and Zac Hatfield-Dodds for their help designing, developing, testing, and using PyMTL

The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of any funding agency.