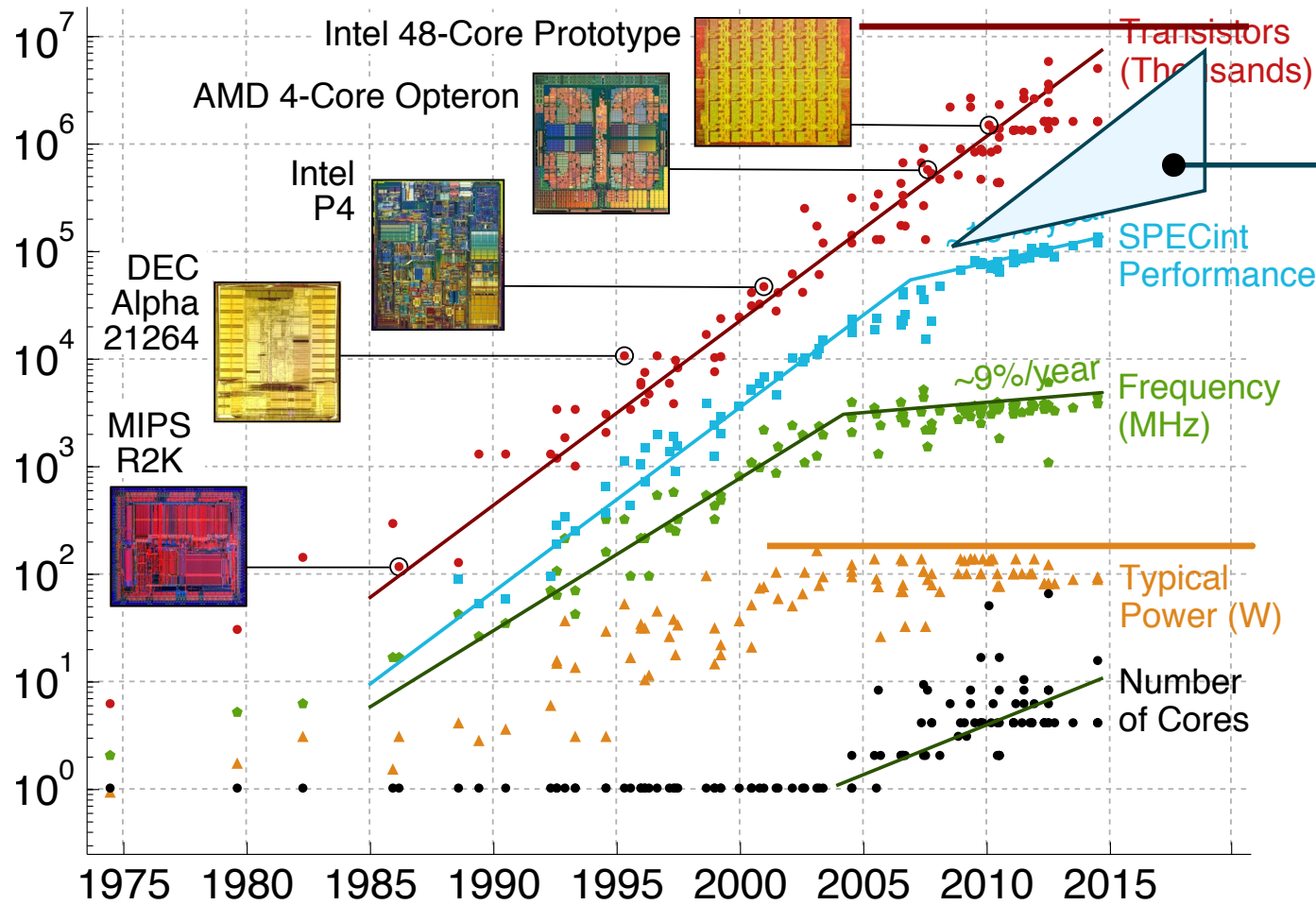


# **A New Era of Open-Source System-on-Chip Design**

Christopher Batten

Computer Systems Laboratory  
Electrical and Computer Engineering  
Cornell University

# Motivating Trends in Computer Architecture



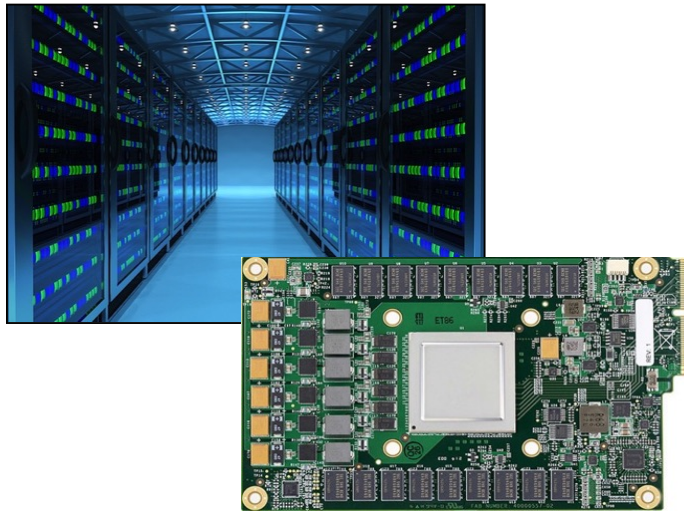
Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten

## Hardware Specialization

- Data-Parallelism via GPGPUs & Vector
- Fine-Grain Task-Level Parallelism
- Instruction Set Specialization
- Subgraph Specialization
- Application-Specific Accelerators
- Domain-Specific Accelerators
- Coarse-Grain Reconfig Arrays
- Field-Programmable Gate Arrays

# Hardware Specialization from Cloud to IoT

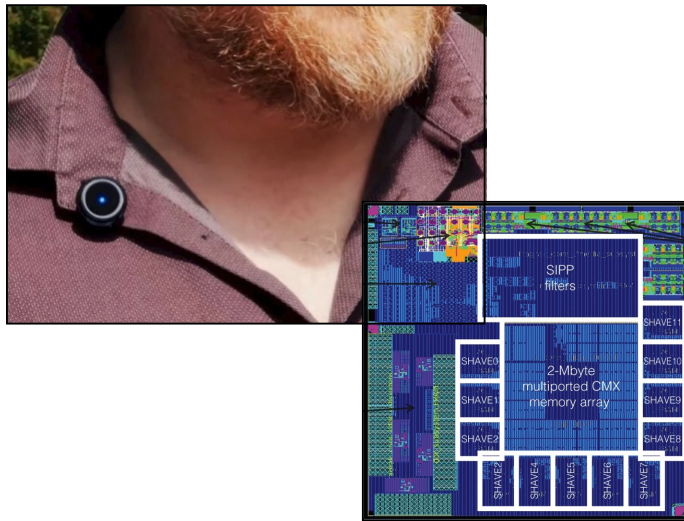
Cloud Computing



## Google TPU

- ▶ Training is done using the TensorFlow C++ framework
- ▶ Training can take weeks
- ▶ Google TPU is custom chip to accelerate training and inference

Internet of Things



## Movidius Myriad 2

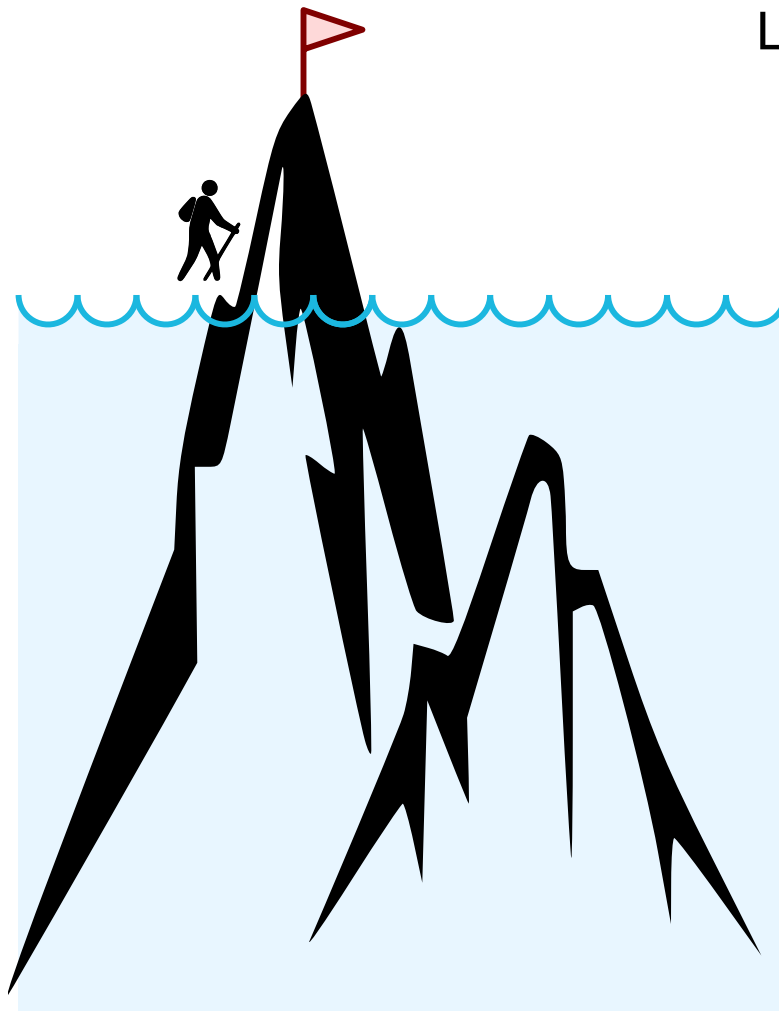
- ▶ Custom chip for ML on embedded IoT devices
- ▶ Specifically focused on vision processing
- ▶ 12 specialized vector VLIW processors

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

How can we **accelerate**  
innovation in  
**accelerator-centric**  
system-on-chip design?

# 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

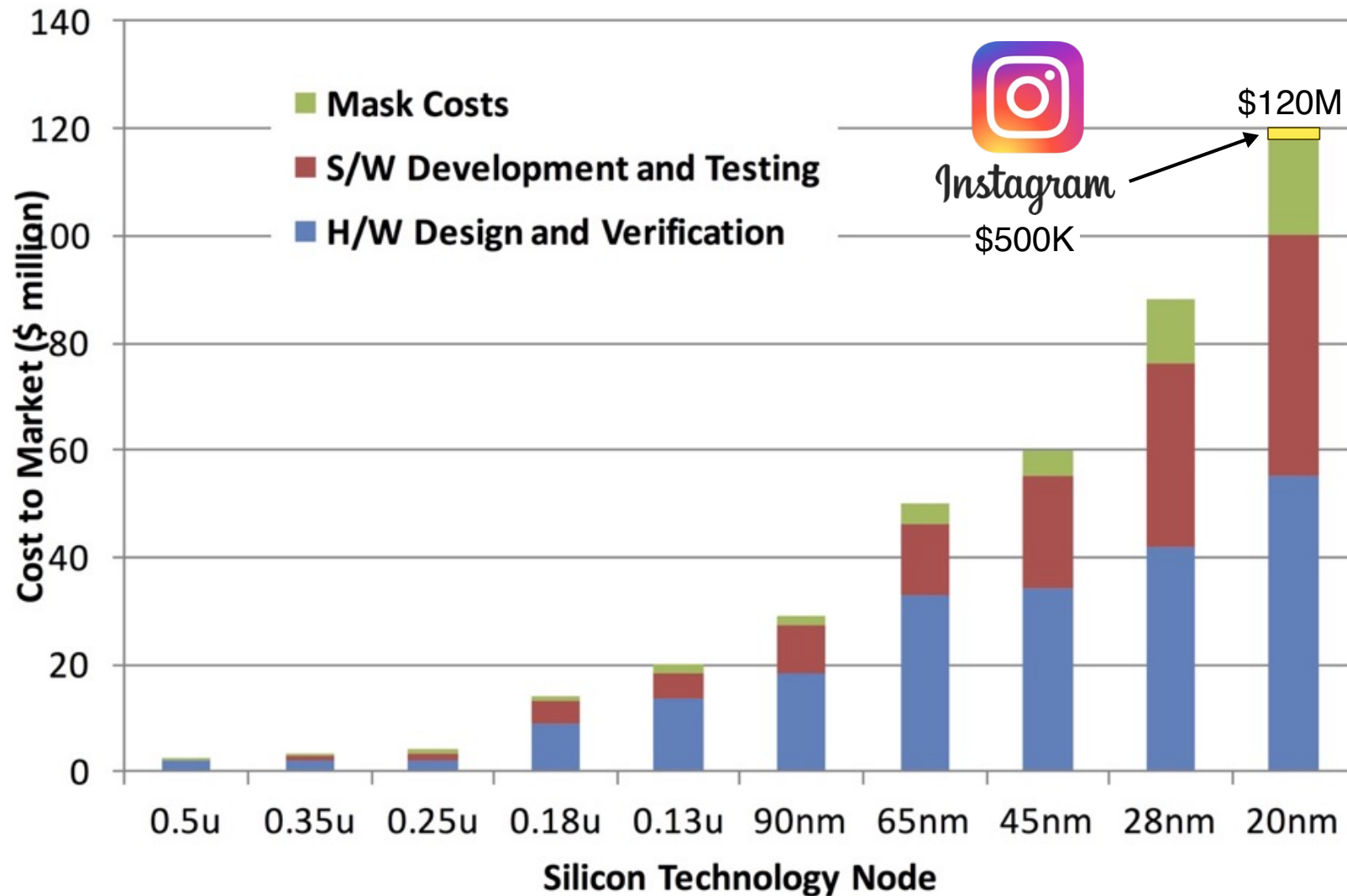
- New machine learning accelerator
- Other unrelated components, anything you cannot afford to buy or for which COTS IP does not do

## 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

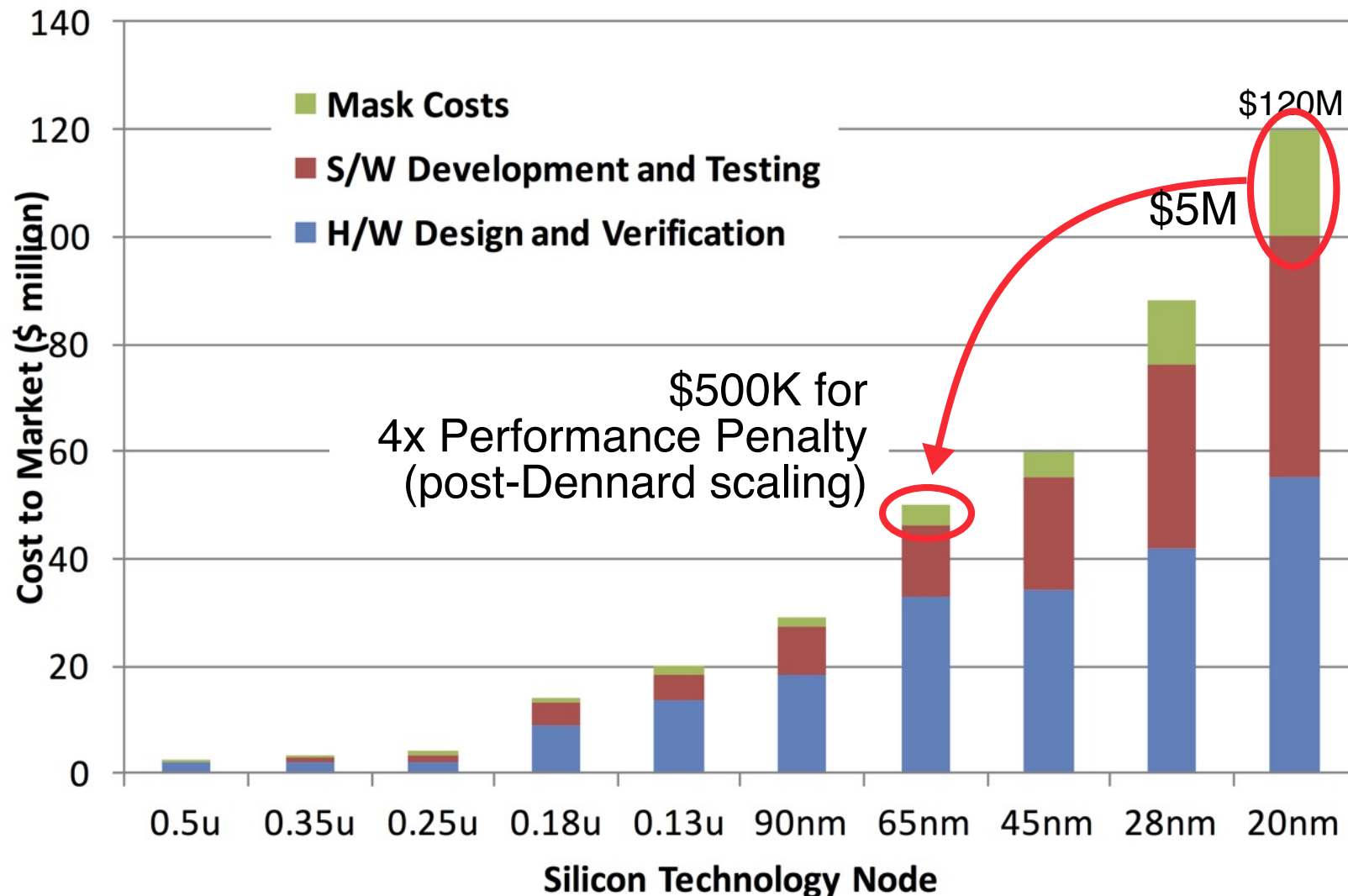
Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16

# Chip Costs Are Skyrocketing



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

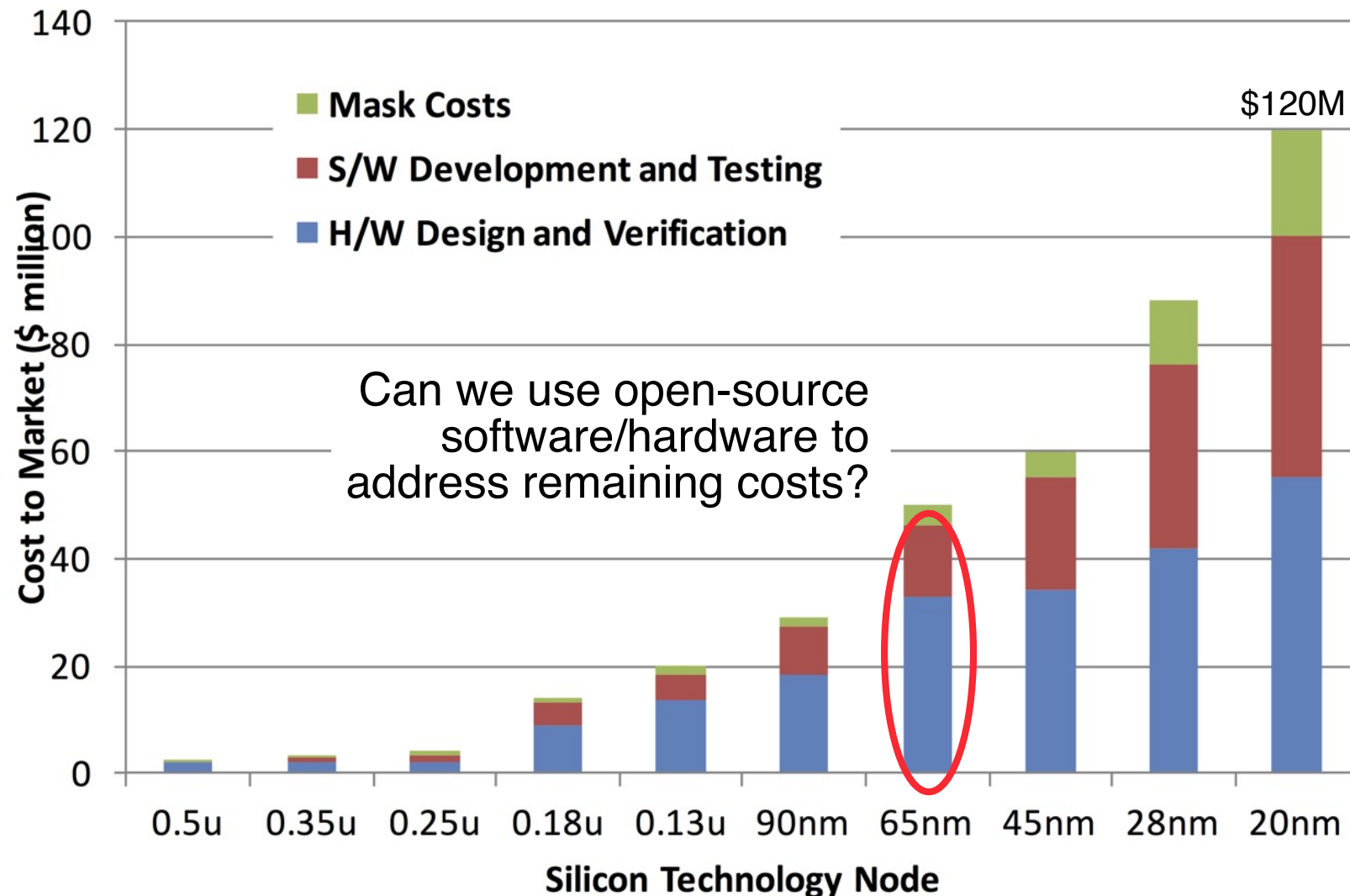
# Minimum Viable Product/Prototype



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



# Minimum Viable Product/Prototype

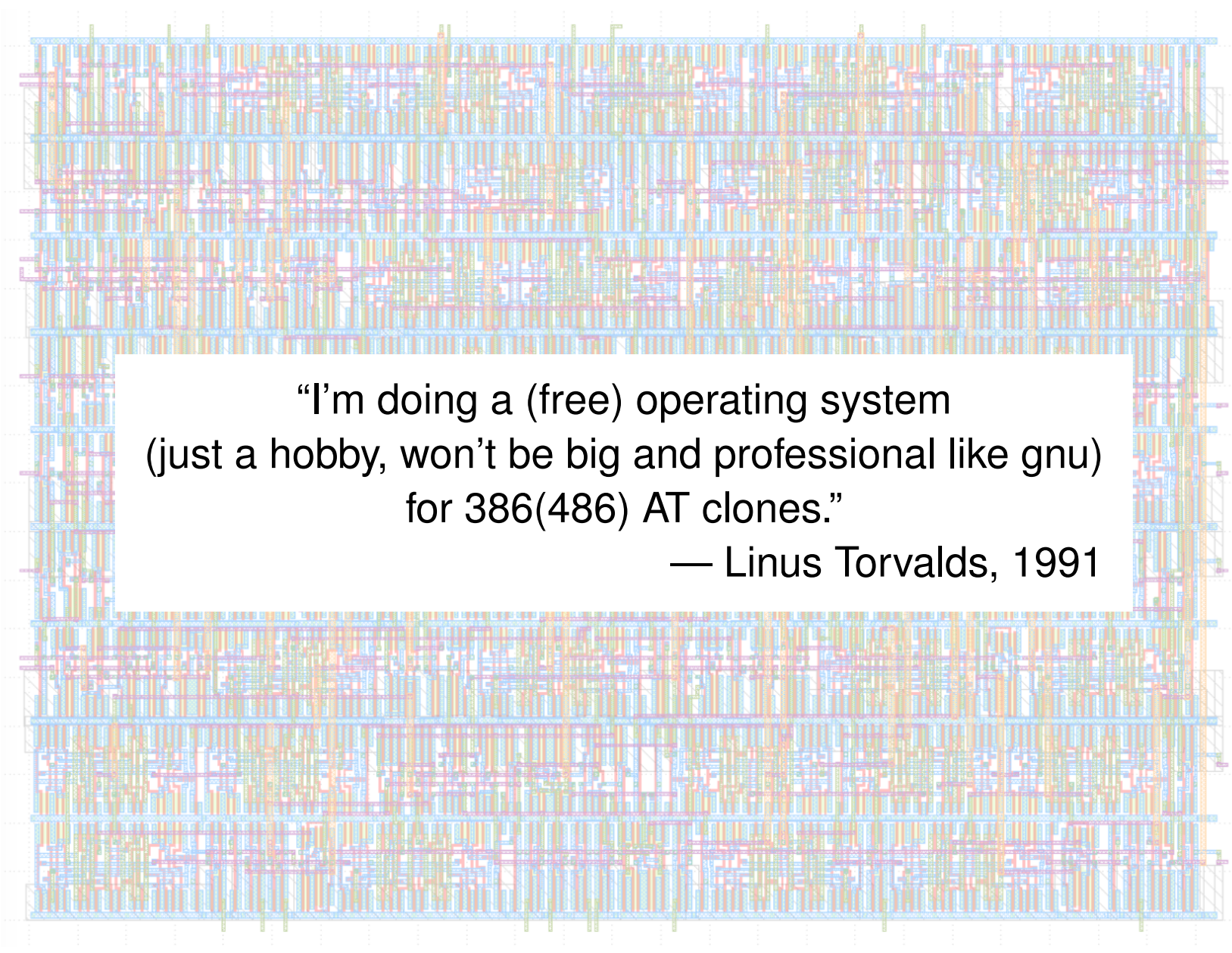


Adapted from M. Taylor, "Open Source HW in 2030," Arch 2030 Workshop @ ISCA'16; original: International Business Strategies & T. Austin.

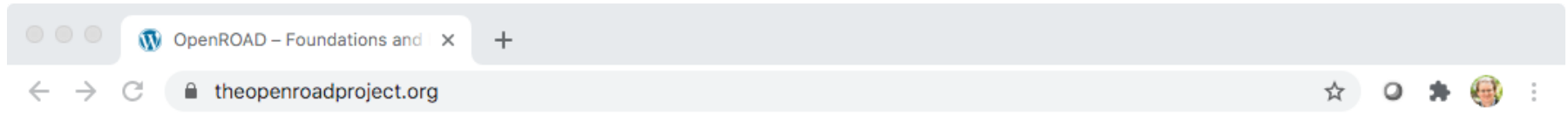
# How can HW design be more like SW design?

Open-Source	Software	Hardware
high-level languages	Python, Ruby, R, Javascript, Julia	Chisel, PyMTL, PyRTL, MyHDL, JHDL, Clash
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

```
# 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



# OpenROAD

Subscribe

Home People News and Events Publications Outreach

## DEMOCRATIZING HARDWARE DESIGN

The OpenROAD project attacks the barriers of Cost, Expertise and Uncertainty (i.e., Risk) that block the feasibility of hardware design in advanced technologies.

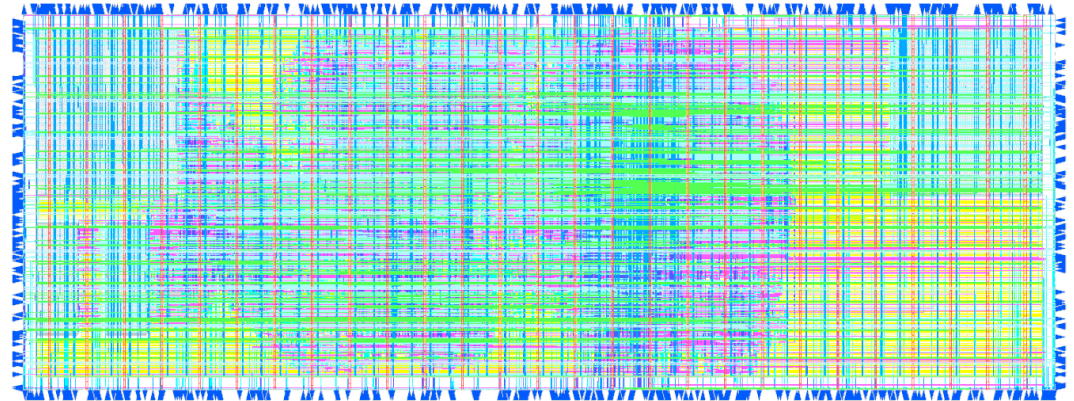
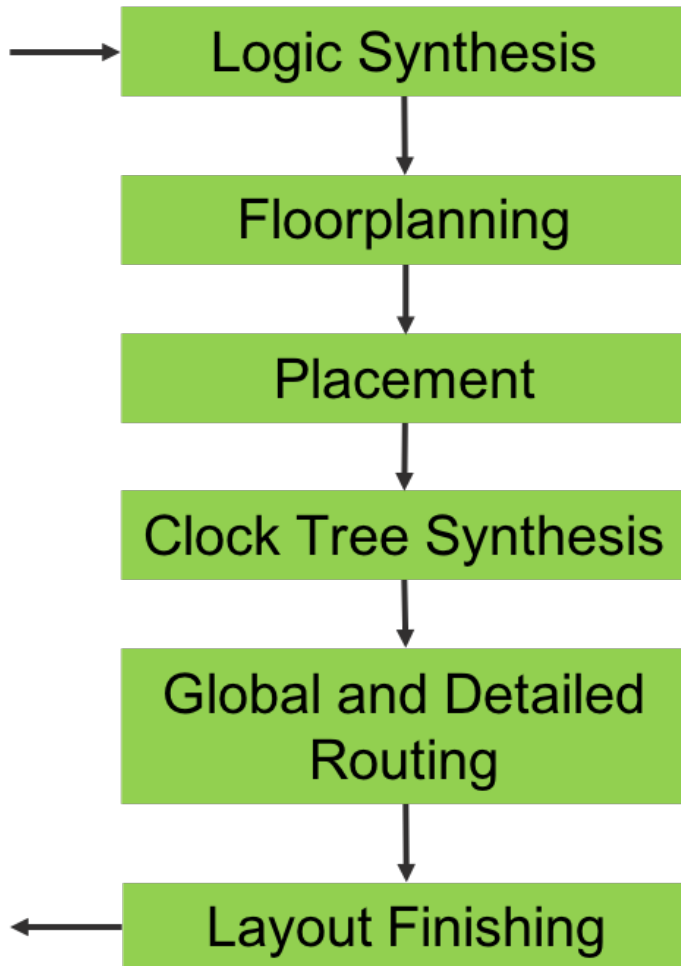
[READ MORE](#)



OpenROAD Retweeted

**Matt Guthaus** 22 Jul  
@efabless OpenLane went publi today. @OpenROAD\_EDA + @Gc @SkyWaterFoundry 130nm #OpenPDK. Soon, @UCSC\_Open too.

# OpenROAD: The Future of Open-Source EDA?

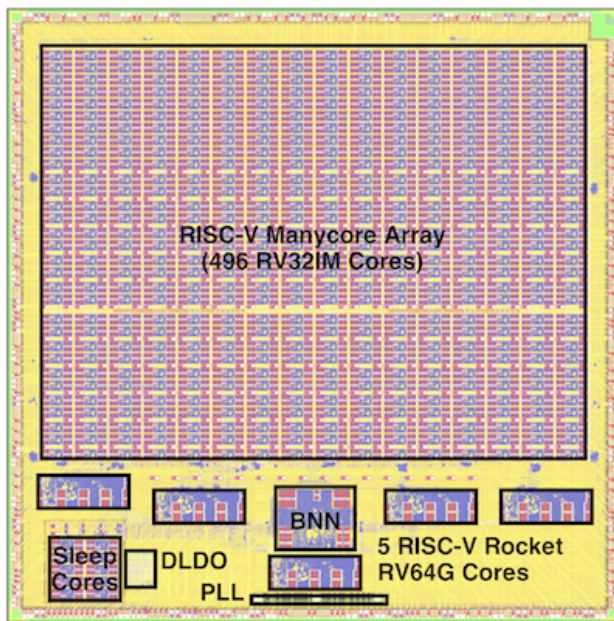


- ▶ Open-source RV32IM core from UW
- ▶ 17K standard-cell instances
- ▶ 66K bits of SRAM
- ▶ TSMC 65LP technology
- ▶ DRC-clean RTL-to-GDS

```

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 SoC Design

## ▶ The PyMTL3 Framework

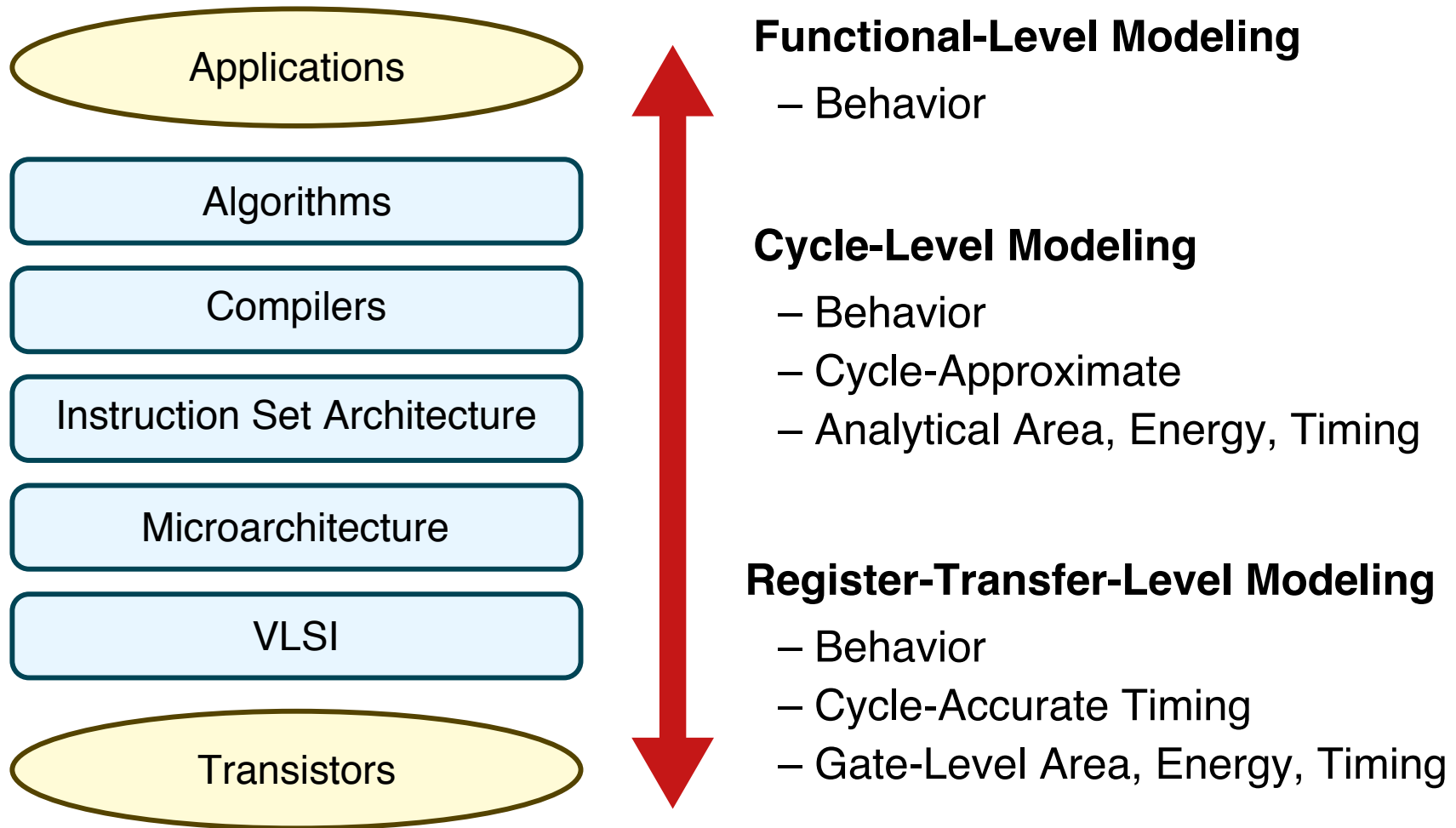
- ▶ PyMTL3 Motivation
- ▶ PyMTL3 Overview
- ▶ PyMTL3 Demo
- ▶ PyMTL3 & Open-Source Hardware

## ▶ The Celerity SoC

- ▶ Celerity Architecture
- ▶ Celerity Case Study
- ▶ Celerity & Open-Source Hardware

## ▶ A Call to Action

# Multi-Level Modeling Methodologies





# Multi-Level Modeling Methodologies

## Multi-Level Modeling Challenge

FL, CL, RTL modeling use very different languages, patterns, tools, and methodologies

**SystemC** is a good example of a unified multi-level modeling framework

Is SystemC the best we can do in terms of **productive** multi-level modeling?



## Functional-Level Modeling

- Algorithm/ISA Development
- MATLAB/Python, C++ ISA Sim

## Cycle-Level Modeling

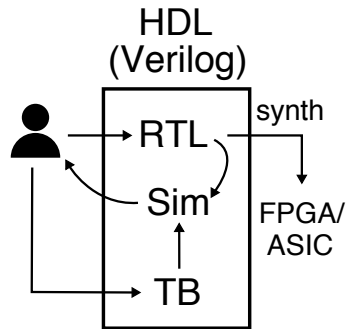
- Design-Space Exploration
- C++ Simulation Framework
- SW-Focused Object-Oriented
- gem5, SESC, McPAT

## Register-Transfer-Level Modeling

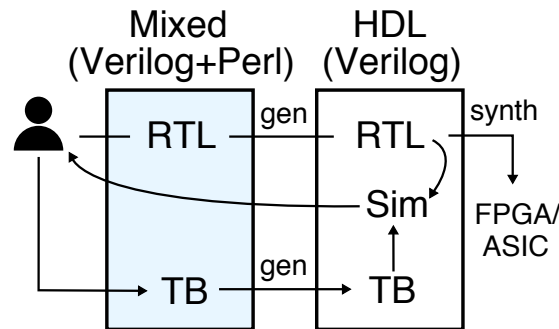
- Prototyping & AET Validation
- Verilog, VHDL Languages
- HW-Focused Concurrent Structural
- EDA Toolflow

# Traditional RTL Design Methodologies

## HDL Hardware Description Language

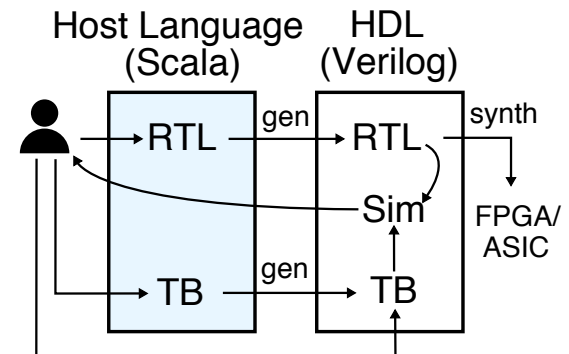


## HPF Hardware Preprocessing Framework



Example: Genesis2

## HGF Hardware Generation Framework



Example: Chisel

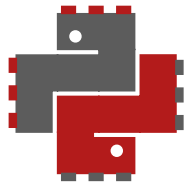
- ✓ Fast edit-sim-debug loop
- ✓ Single language for structural, behavioral, + TB
- ✗ Difficult to create highly parameterized generators

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

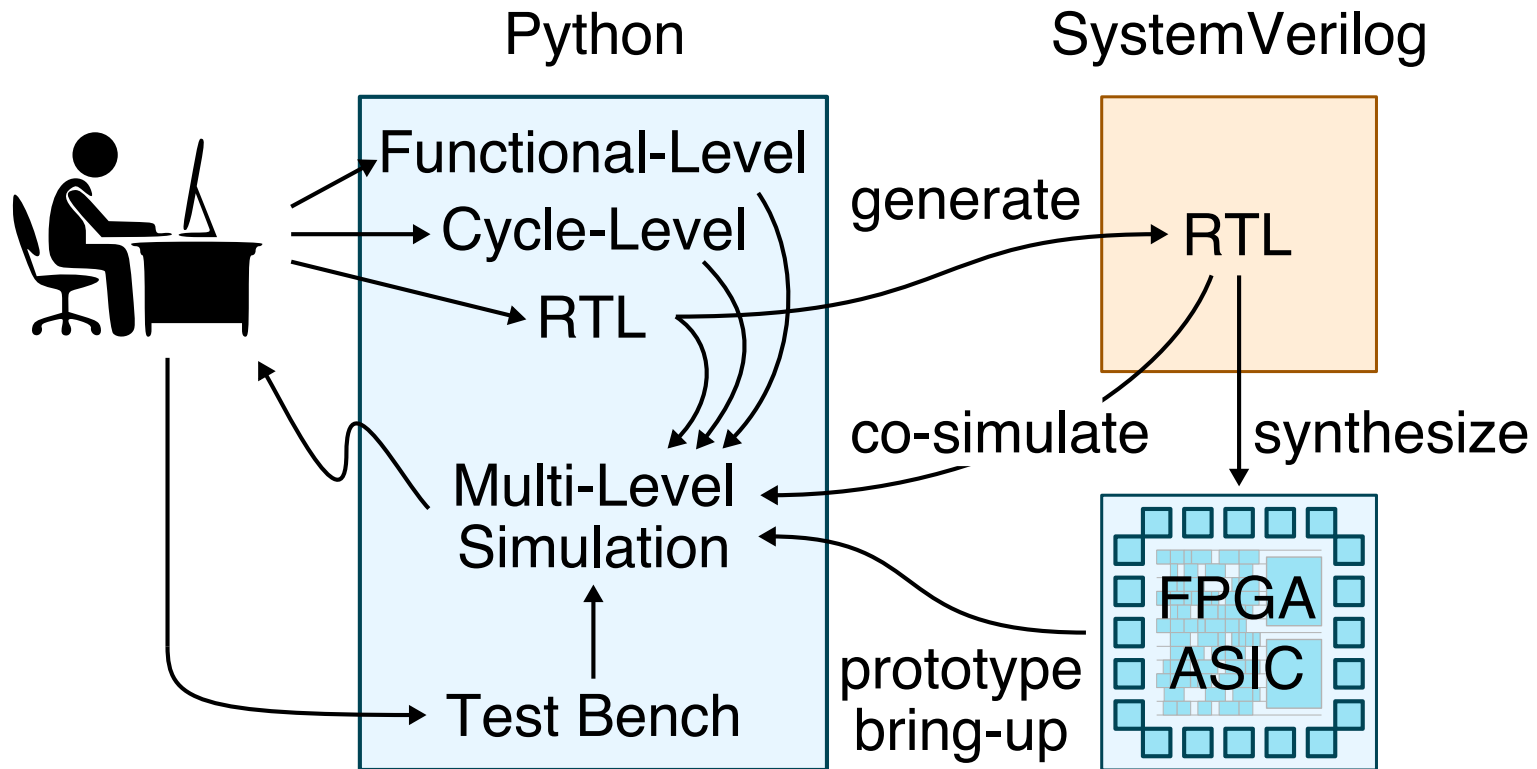
- ✗ Slower edit-sim-debug loop
- ✓ Single language for structural + behavioral
- ✓ Easier to create highly parameterized generators
- ✗ Cannot use power of host language for verification

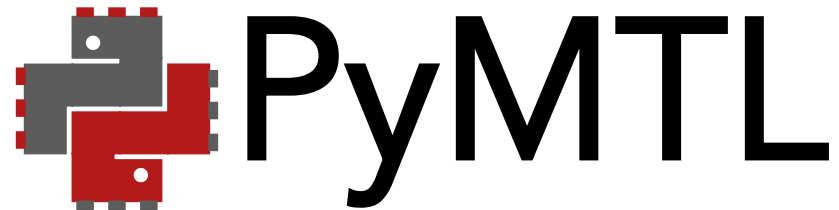
Is Chisel the best we can do in terms of a **productive** RTL design methodology?

# PyMTL



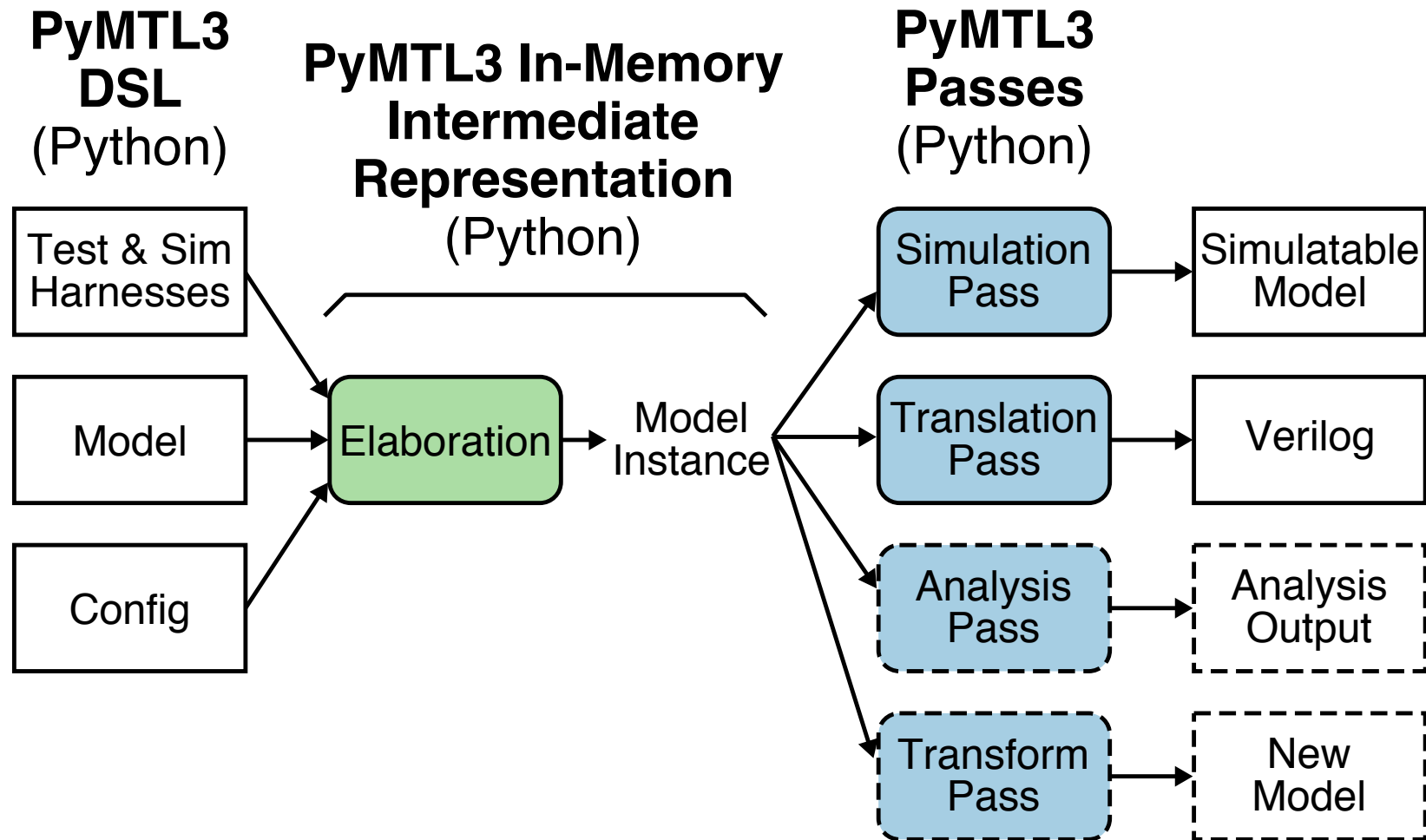
Python-based hardware generation, simulation, and verification framework which enables productive multi-level modeling and RTL design





- ▶ **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



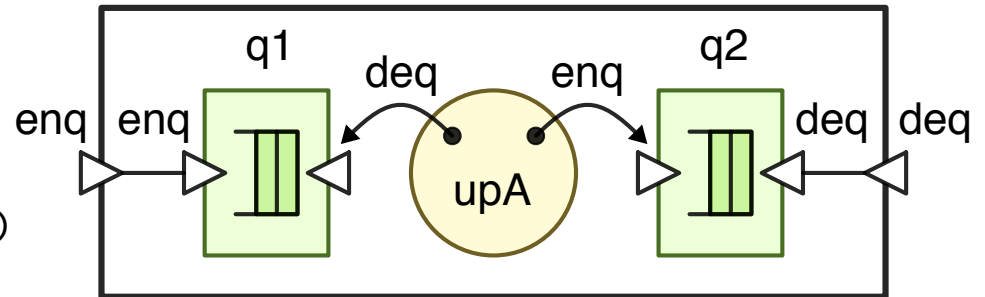
# PyMTL3 High-Level Modeling

```

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() )

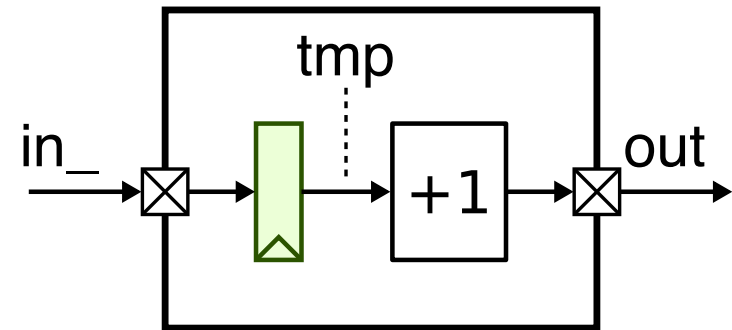
```

# PyMTL3 Low-Level Modeling

```

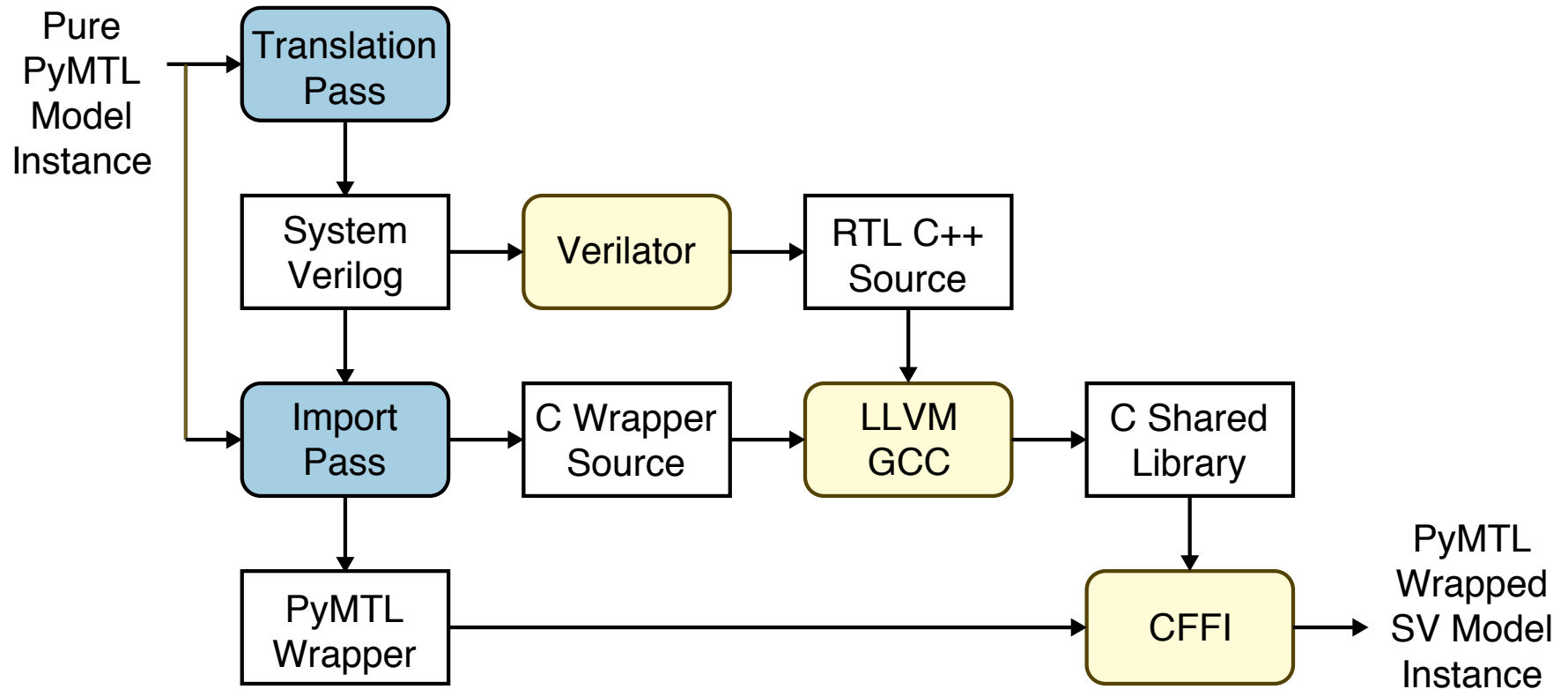
1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5      def construct( s, nbits ):
6          s.in_ = InPort ( nbits )
7          s.out = OutPort( nbits )
8          s.tmp = Wire   ( nbits )
9
10         @update_ff
11         def seq_logic():
12             s.tmp <<= s.in_
13
14        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

```
% python3 -m venv pymtl3
% source pymtl3/bin/activate
% pip install pymtl3
% python

>>> from pymtl3 import *

>>> a = Bits8(6)
>>> a
>>> b = Bits8(3)
>>> b
>>> a | b
>>> a << 4

>>> c = (a << 4) | b
>>> c
>>> c[4:8]

>>> from pymtl3.examples.ex00_quickstart \
        import FullAdder
>>> import inspect
>>> print(inspect.getsource(FullAdder))

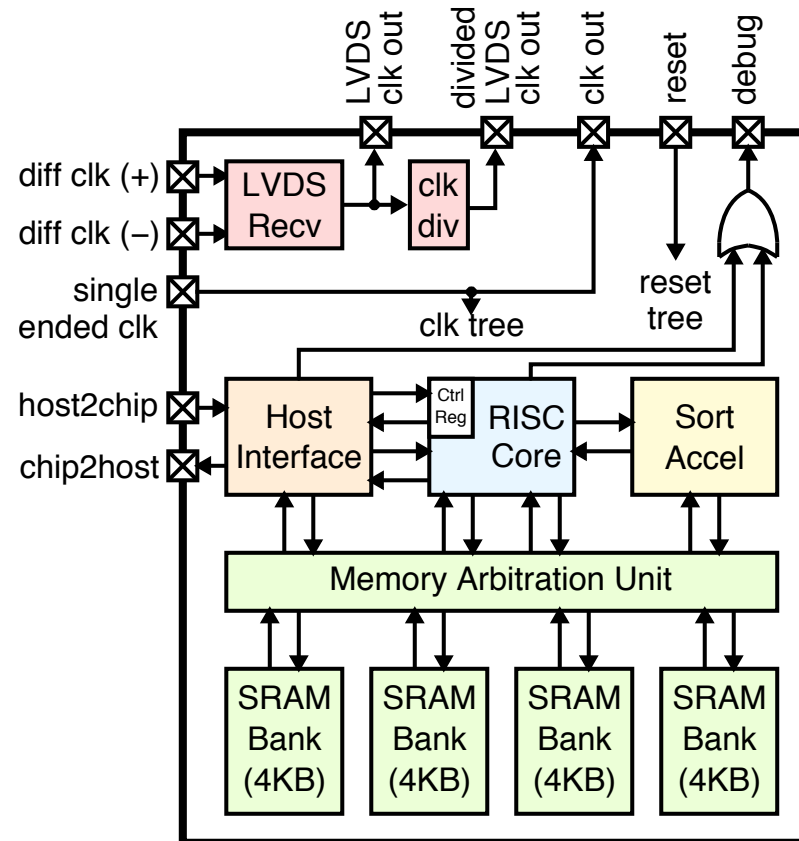
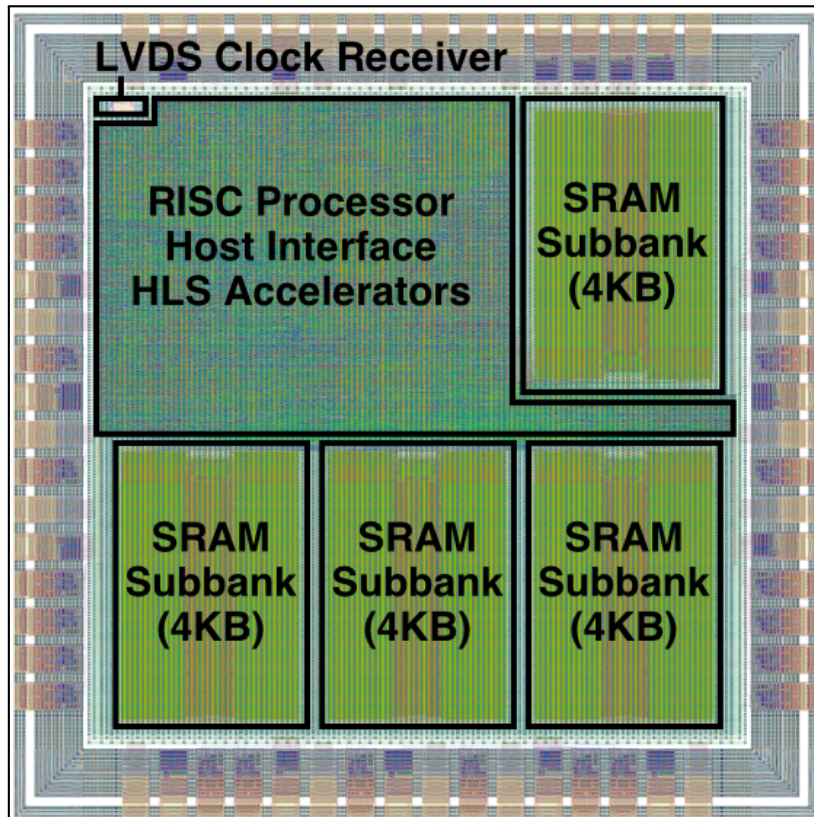
>>> fa = FullAdder()
>>> fa.apply(
        DefaultPassGroup(textwave=True) )
>>> fa.sim_reset()

>>> fa.a    @= 0
>>> fa.b    @= 1
>>> fa.cin  @= 0
>>> fa.sim_tick()

>>> fa.a    @= 1
>>> fa.b    @= 0
>>> fa.cin  @= 1
>>> fa.sim_tick()

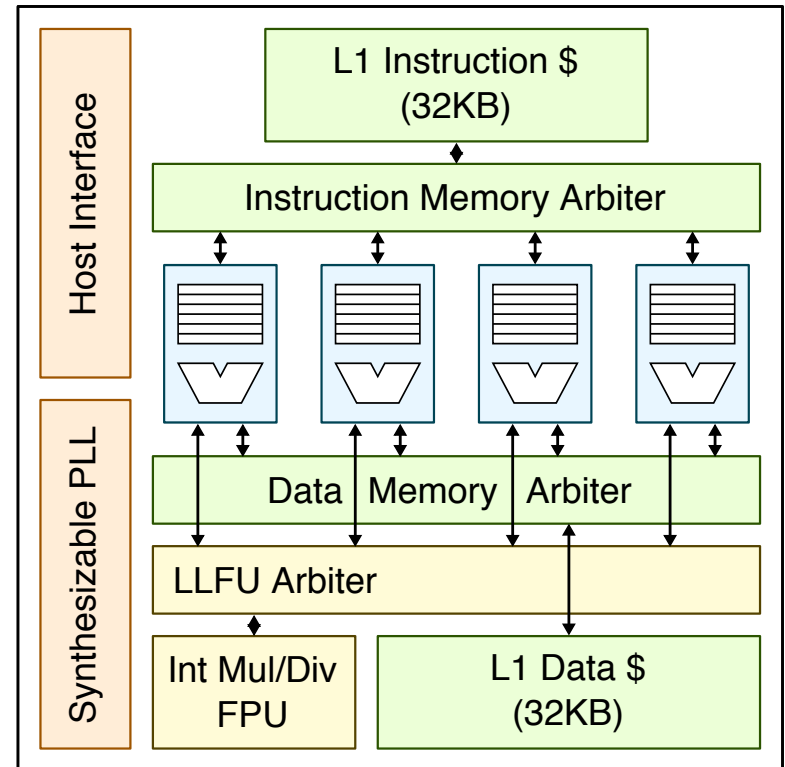
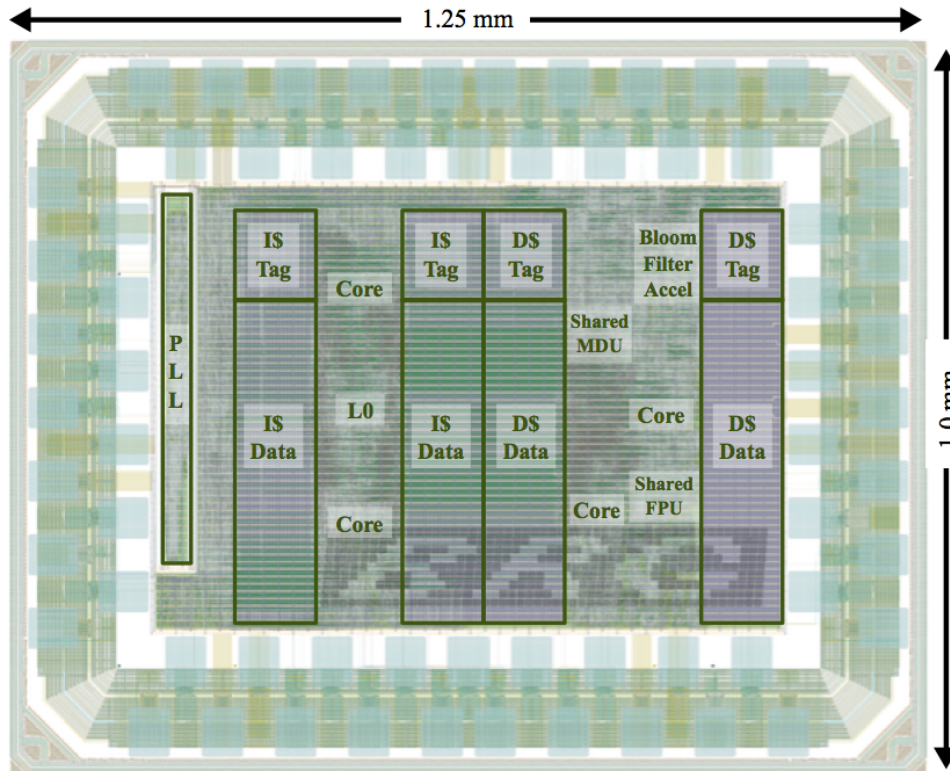
>>> fa.print_textwave()
```

# PyMTL2 ASIC Tapeout #1 (2016)



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

# PyMTL2 ASIC Tapeout #2 (2018)



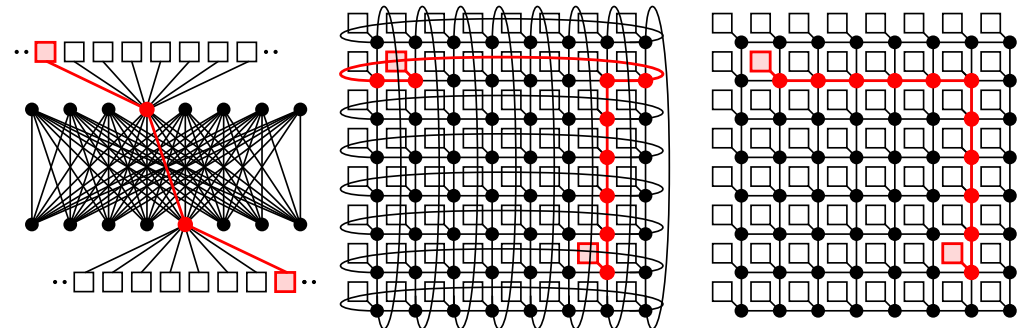
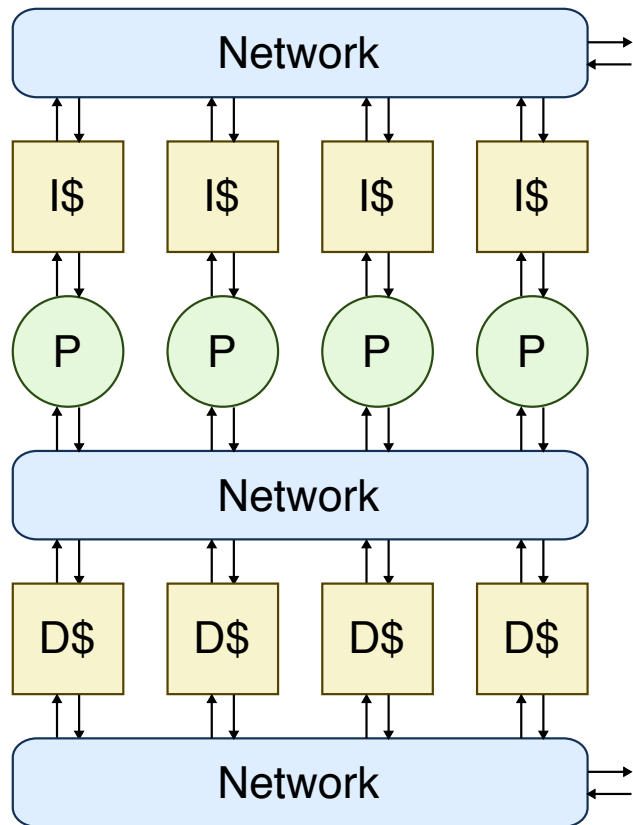
Four RISC-V RV32IMAF cores with “smart” sharing of L1\$/LLFU  
 1x1.2mm, 6.7M-trans, TSMC 28nm  
 95% done using PyMTL2

# PyMTL and Open-Source Hardware

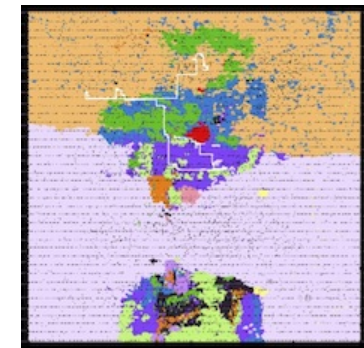
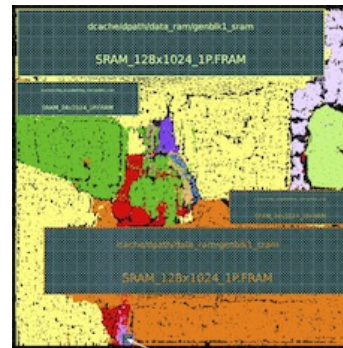
---

- ▶ State-of-the-art in open-source HDL simulators
  - ▷ *Icarus Verilog*: Verilog interpreter-based simulator
  - ▷ *Verilator*: Verilog AOT-compiled simulator
  - ▷ *GHDL*: VHDL AOT-compiled simulator
  - ▷ No open-source simulator supports modern verification environments
  
- ▶ PyMTL as an open-source design, simulation, verification environment
  - ▷ Open-source hardware developers can use Verilog RTL for design and Python, a well-known general-purpose language, for verification
  - ▷ PyMTL for FL design enables creating high-level golden reference models
  - ▷ PyMTL for RTL design enables creating highly parameterized hardware components which is critical for encouraging reuse in an open-source ecosystem

# PyMTL and Open-Source Hardware



**DARPA POSH Open-Source Hardware Program**  
 PyMTL used as a powerful open-source generator  
 for both design and verification



**Undergraduate Comp Arch Course**  
 Labs use PyMTL for verification,  
 PyMTL or Verilog for RTL design

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

# PyMTL Publications

- ▶ Derek Lockhart, et al., “PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research.” *47th ACM/IEEE Int’l Symp. on Microarchitecture (MICRO)*, Dec. 2014.
- ▶ 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, et al., “PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification.” *IEEE Micro*, 40(4):5866, Jul/Aug. 2020.
- ▶ Shunning Jiang\*, Yanghui Ou\*, et al., “PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies.” *IEEE Design & Test*, to appear.

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

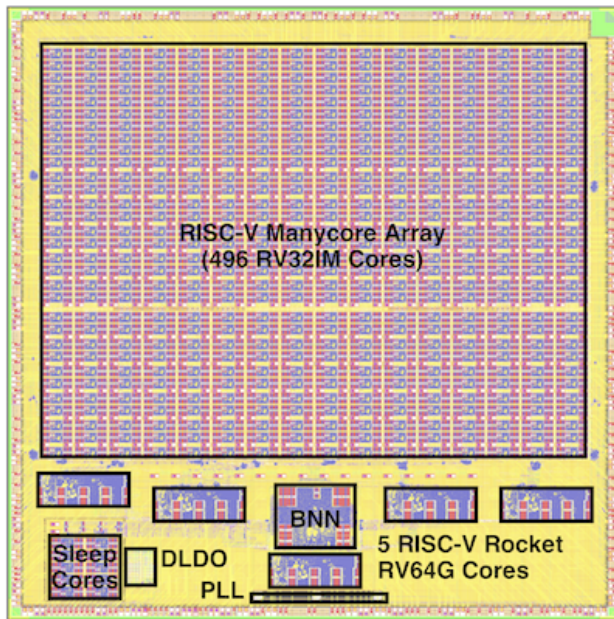
system-on-chip (SoC) design using heterogeneous architectures with a mix of general-purpose and specialized computing engines. Heterogeneous SoCs emphasize both flexible parameterization of a single design block and versatile composition of numerous different design blocks, which have imposed significant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

```

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 SoC Design

## ► The PyMTL3 Framework

- ▷ PyMTL3 Motivation
- ▷ PyMTL3 Overview
- ▷ PyMTL3 Demo
- ▷ PyMTL3 & Open-Source Hardware

## ► The Celerity SoC

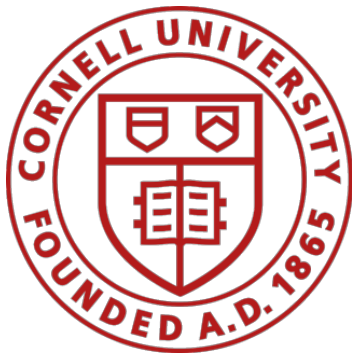
- ▷ Celerity Architecture
- ▷ Celerity Case Study
- ▷ Celerity & Open-Source Hardware

## ► A Call to Action



# The Celerity System-on-Chip Team

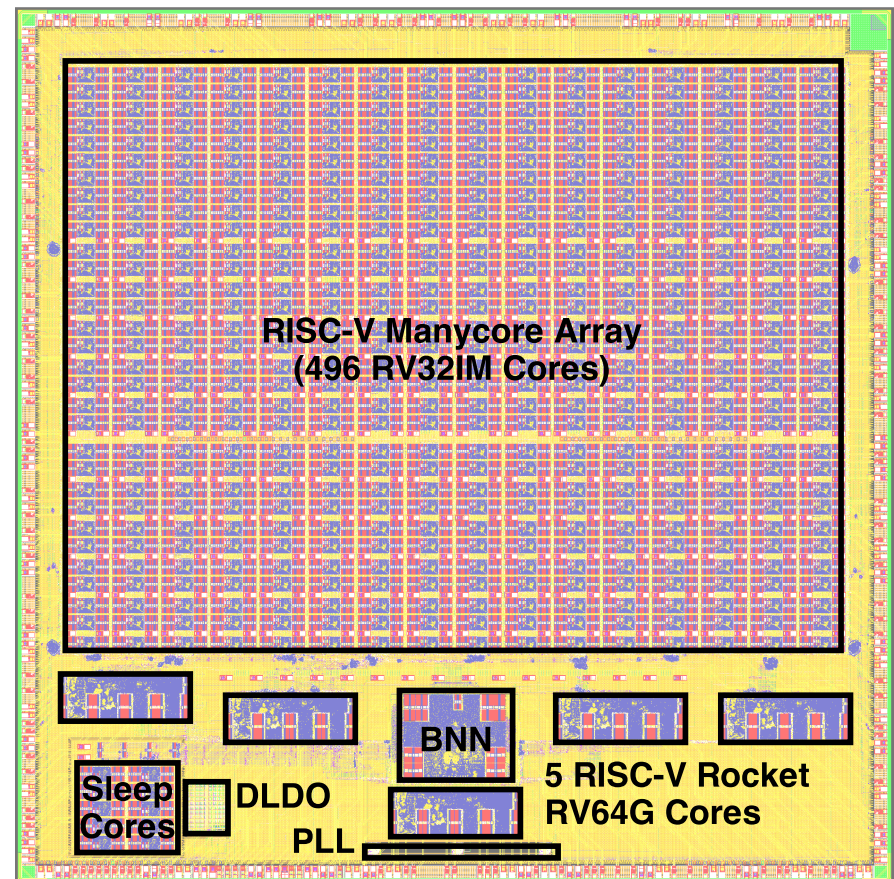
Tutu Ajayi, Khalid Al-Hawaj, Aporva Amarnath, Steve Dai, Scott Davidson, Paul Gao, Gai Liu, Atieh Lotfi, Julian Puscar, Anuj Rao, Austin Rovinski, Loai Salem, Ningxiao Sun, Christopher Torng, Luis Vega, Bandhav Veluri, Xiaoyang Wang, Shaolin Xie, Chun Zhao, Ritchie Zhao, Christopher Batten, Ronald G. Dreslinski, Ian Galton, Rajesh K. Gupta, Patrick P. Mercier, Mani Srivastava, Michael B. Taylor, and Zhiru Zhang



# Celerity System-on-Chip Overview

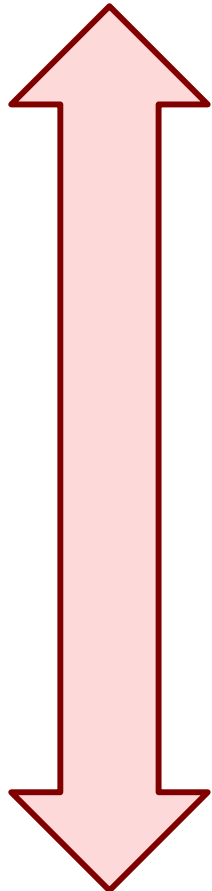
Target Workload: High-Performance Embedded Computing

- ▶ 5 × 5mm 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



# Tiered Accelerator Fabrics

**Flexibility**

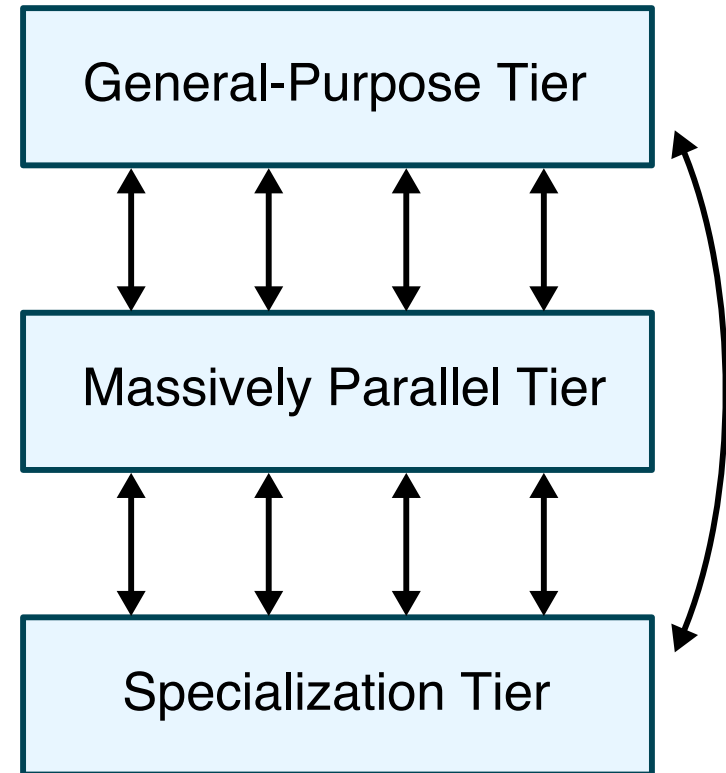


- General-purpose computation
- Operating systems, I/O

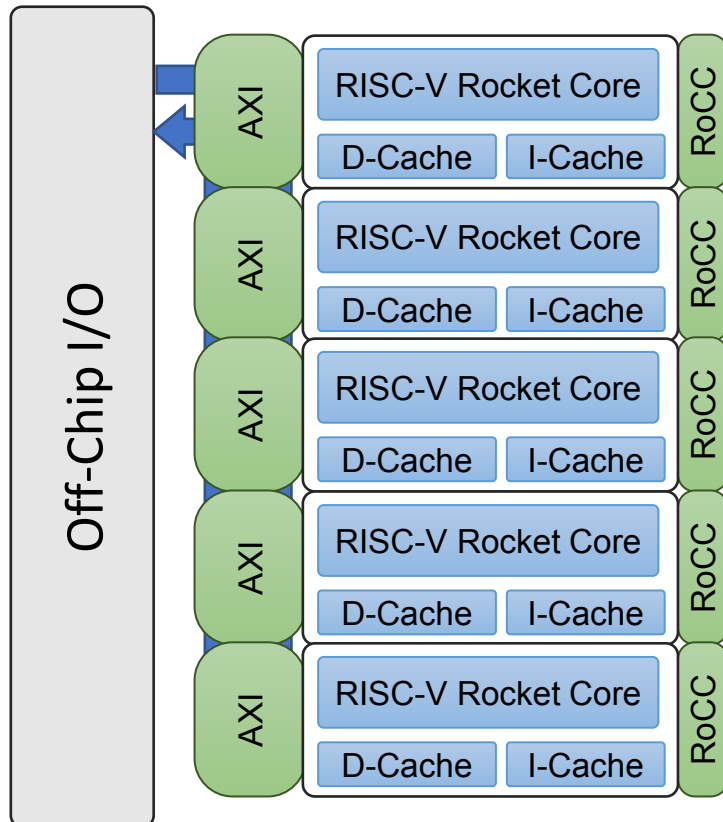
- Flexible and energy-efficient
- Exploits coarse- and fine-grain parallelism

- Fixed-function
- Extremely energy efficient

**Efficiency**

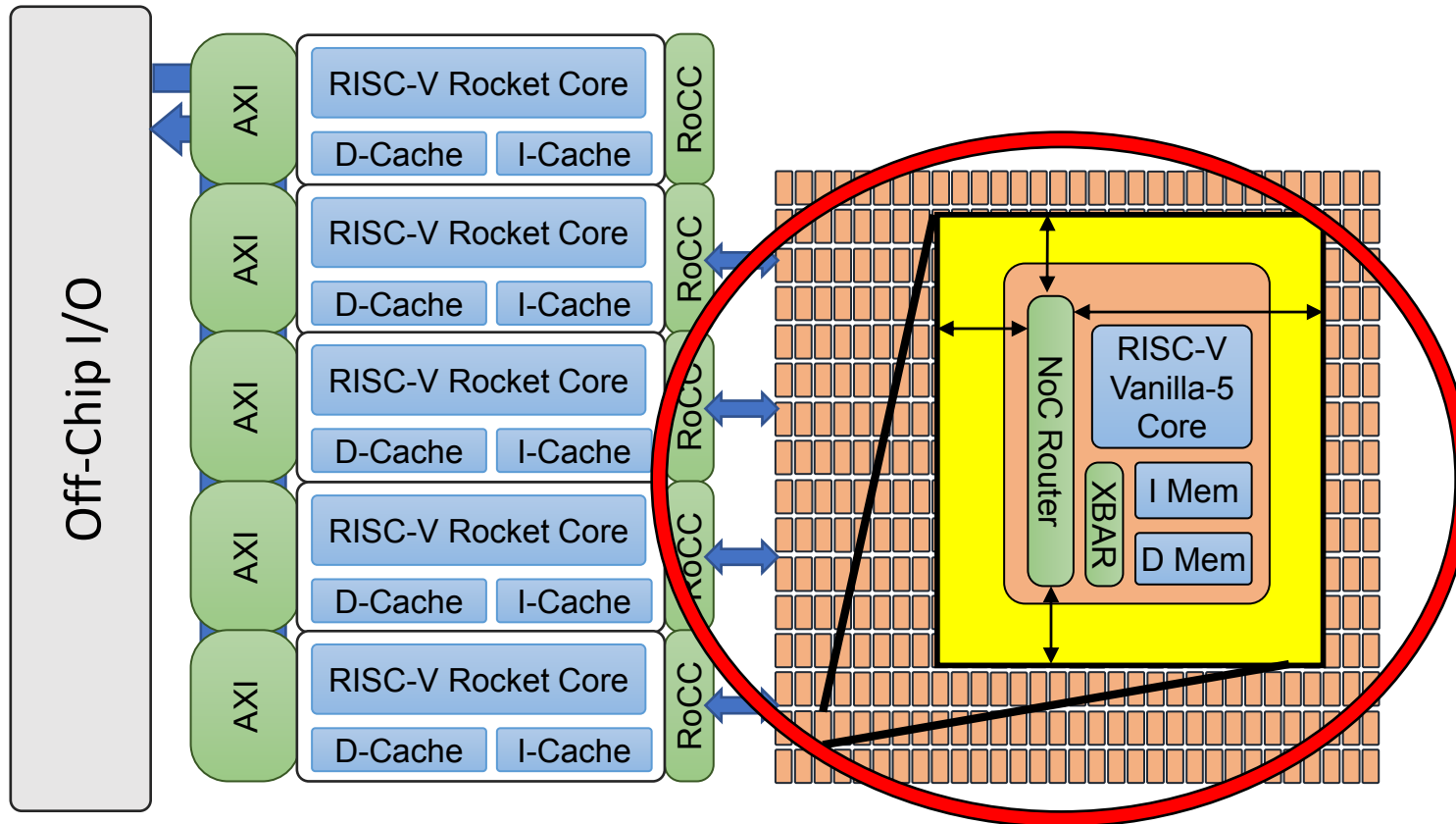


# Celerity: General-Purpose Tier



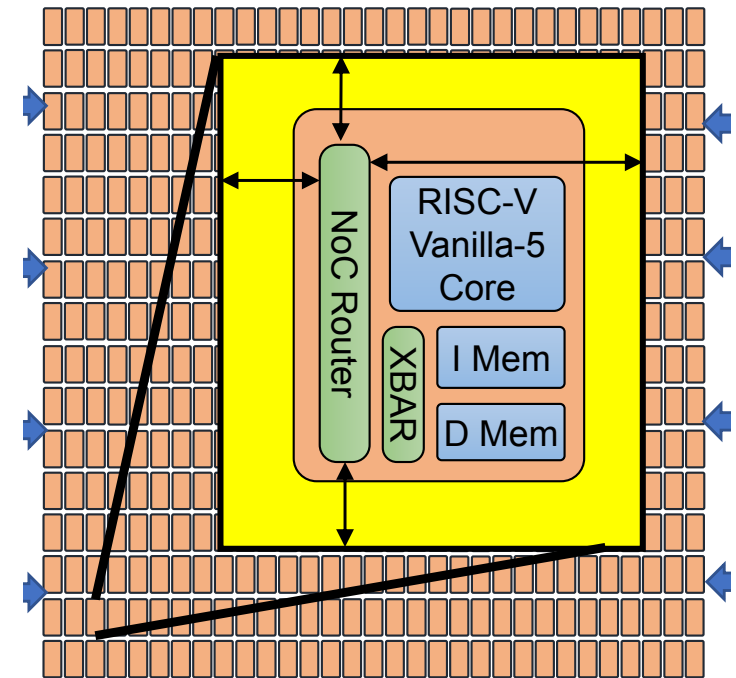
- ▶ Role of the General-Purpose Tier
  - ▷ General-purpose computation
  - ▷ Exception handling
  - ▷ Operating system (e.g., TCP/IP)
  - ▷ Cache memory hierarchy for all tiers
- ▶ In Celerity
  - ▷ 5 Rocket cores from UC Berkeley
  - ▷ Generated from Chisel
  - ▷ RV64G ISA
  - ▷ 5-stage, in-order, scalar processor
  - ▷ Double-precision floating point
  - ▷ I-Cache: 16KB 4-way assoc.
  - ▷ D-Cache: 16KB 4-way assoc.
  - ▷ 0.97 mm<sup>2</sup> per core @ 625 MHz

# Celerity: Massively Parallel Tier

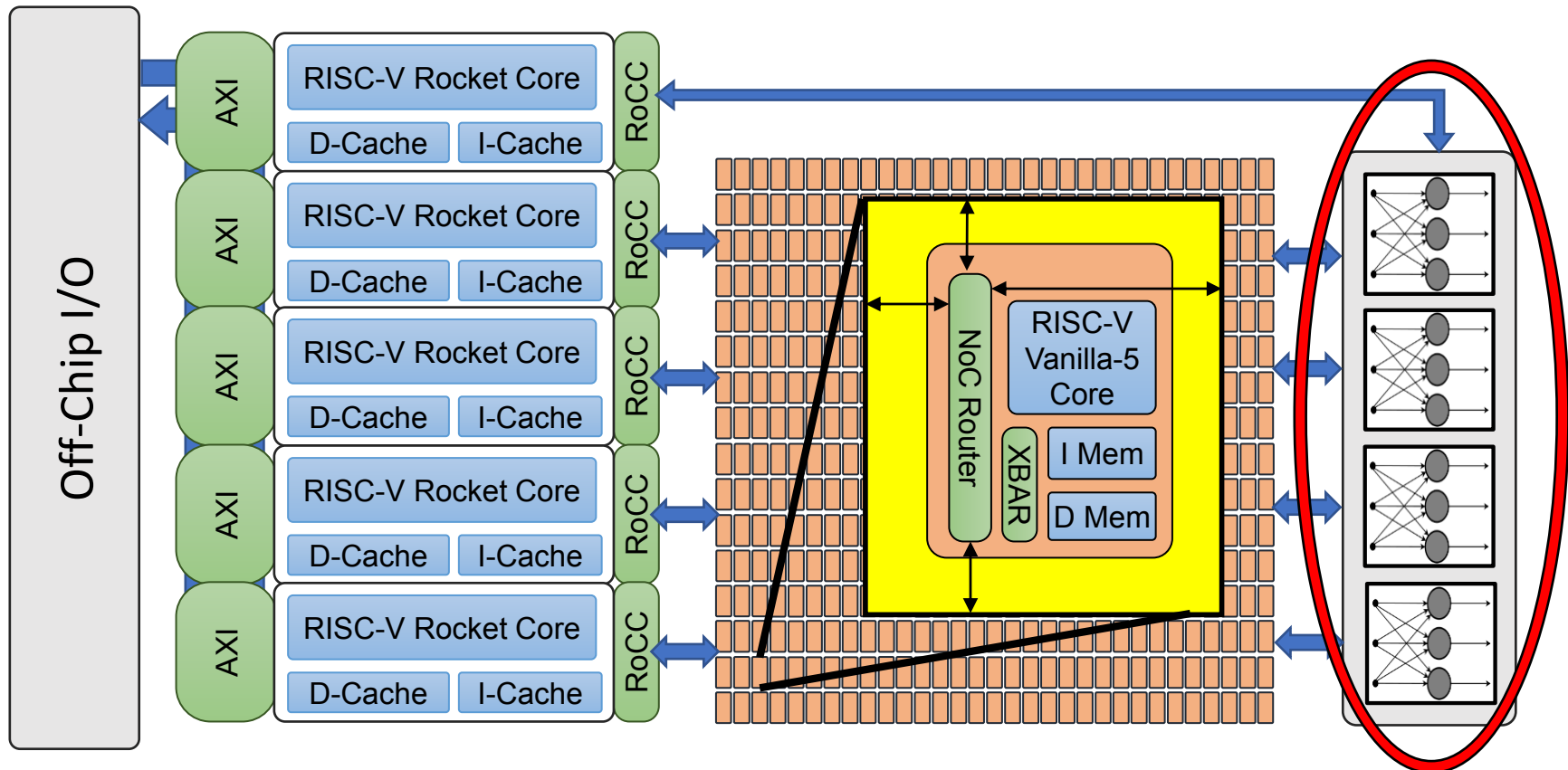


# Celerity: Massively Parallel Tier

- ▶ Role of the Massively Parallel Tier
  - ▷ Improve energy efficiency over general-purpose tier by exploiting massive parallelism
- ▶ 496 low-power RISC-V Vanilla-5 cores
  - ▷ RV32IM ISA
  - ▷ 5-stage, in-order, scalar cores
  - ▷ 4KB instruction memory per tile
  - ▷ 4KB data memory per tile
  - ▷ 0.024 mm<sup>2</sup> per tile @ 1.05 GHz
- ▶ 16 × 31 tiled mesh array
  - ▷ MIMD programming model
  - ▷ XY-dimension network-on-chip (NoC)
  - ▷ 32 b/cycle channels
  - ▷ Manycore I/O uses same network

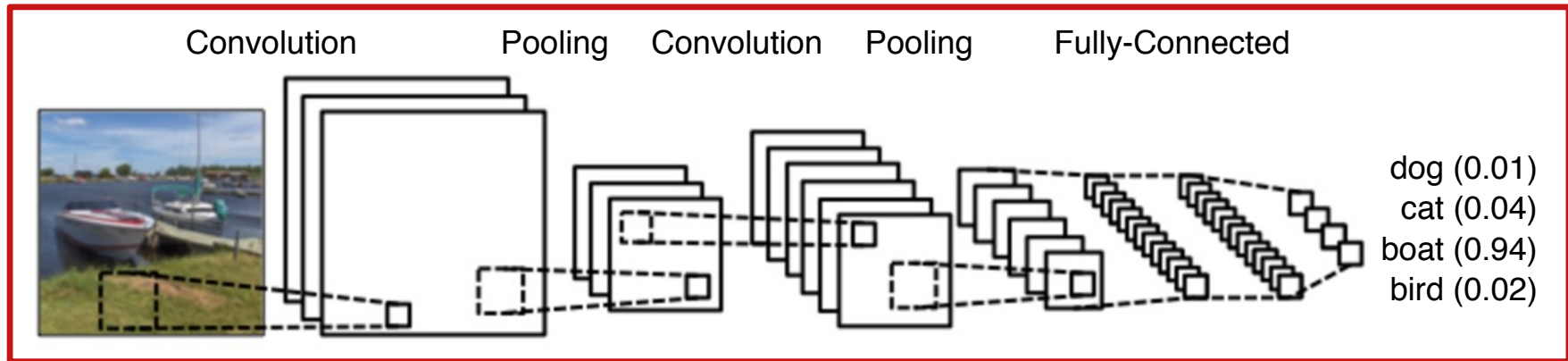


# Celerity: Specialization Tier

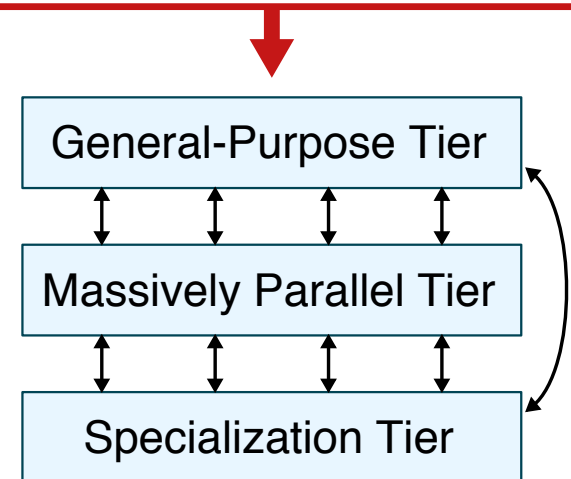


Role of Specialization Tier  
 Ultra-high-energy efficiency for critical applications

# Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

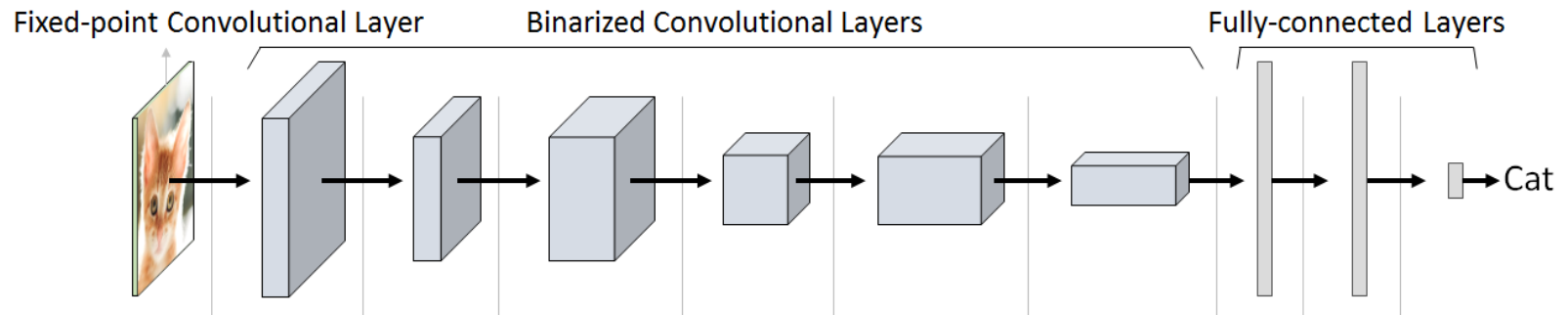


- ▶ **Step 1:** Implement the algorithm using the general-purpose tier
- ▶ **Step 2:** Accelerate algo using either massively parallel tier **OR** specialization tier
- ▶ **Step 3:** Improve performance by cooperatively using both the specialization **AND** the massively parallel tier





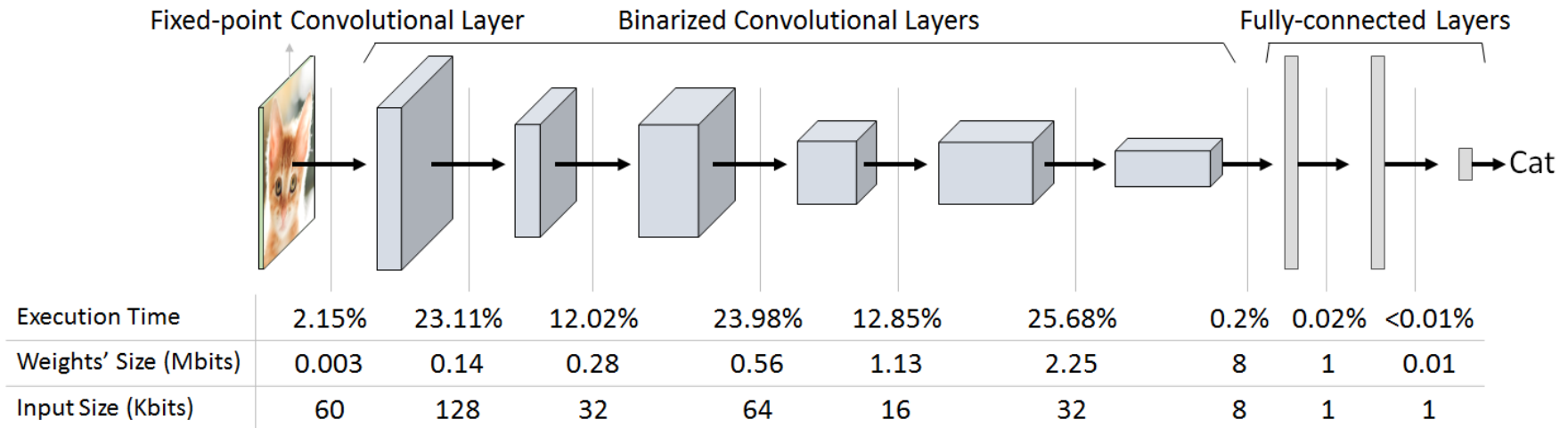
# Step 1: Algo to App – Binarized Neural Networks



- ▶ Training usually uses floating point, while inference usually uses lower precision weights and activations (often 8-bit or lower) to reduce implementation complexity
- ▶ Recent work has shown single-bit precision weights and activations can achieve an accuracy of 89.8% on CIFAR-10
- ▶ Performance target requires ultra-low latency (batch size of one) and high throughput (60 classifications/second)

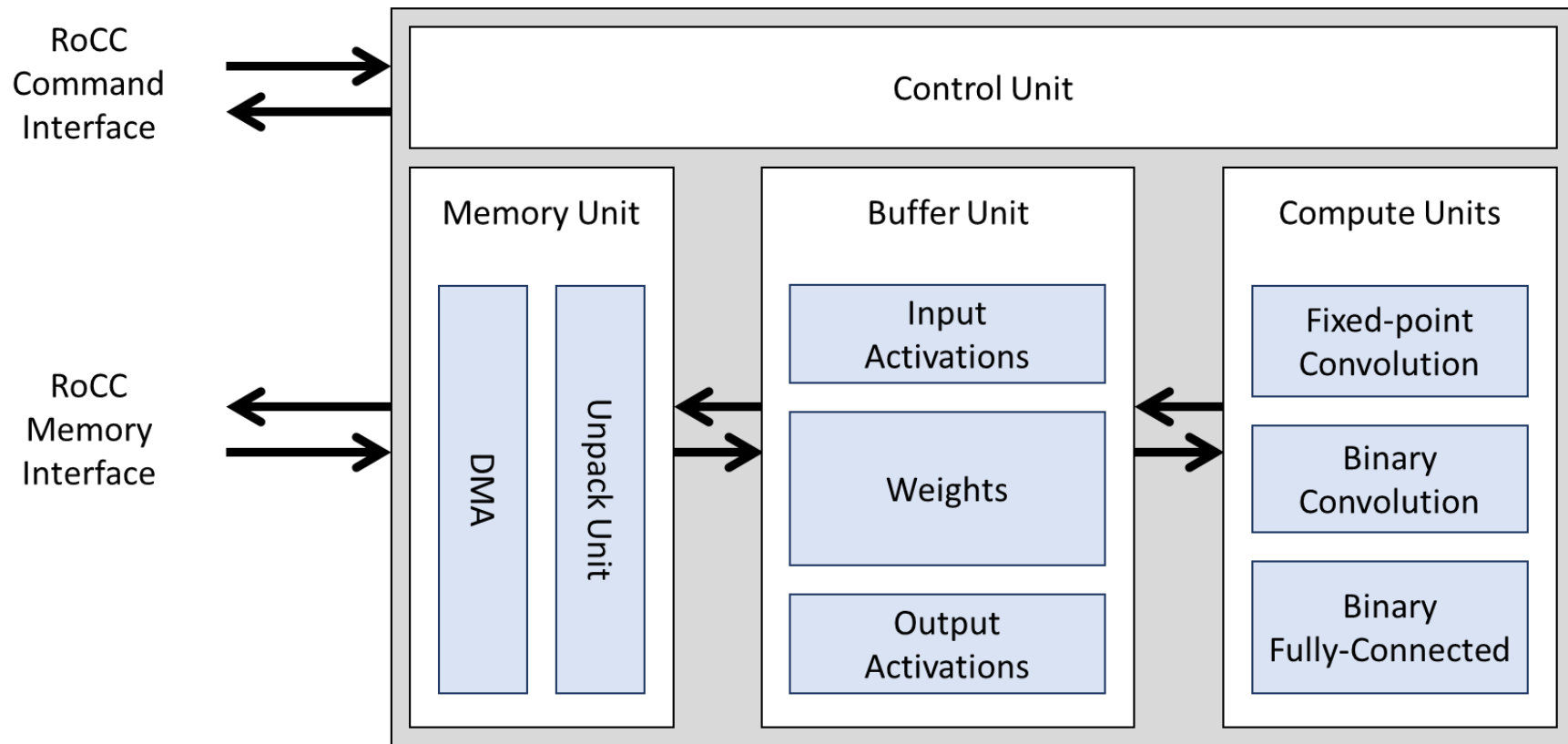
M. Rastergari, et al. "Xnor-net ..." ICCV, 2016; M. Courbariaux, et al. "Binarized neural networks ..." arXiv:1602.02830, 2016.

# Step 1: Algo to App – Characterizing BNN Execution

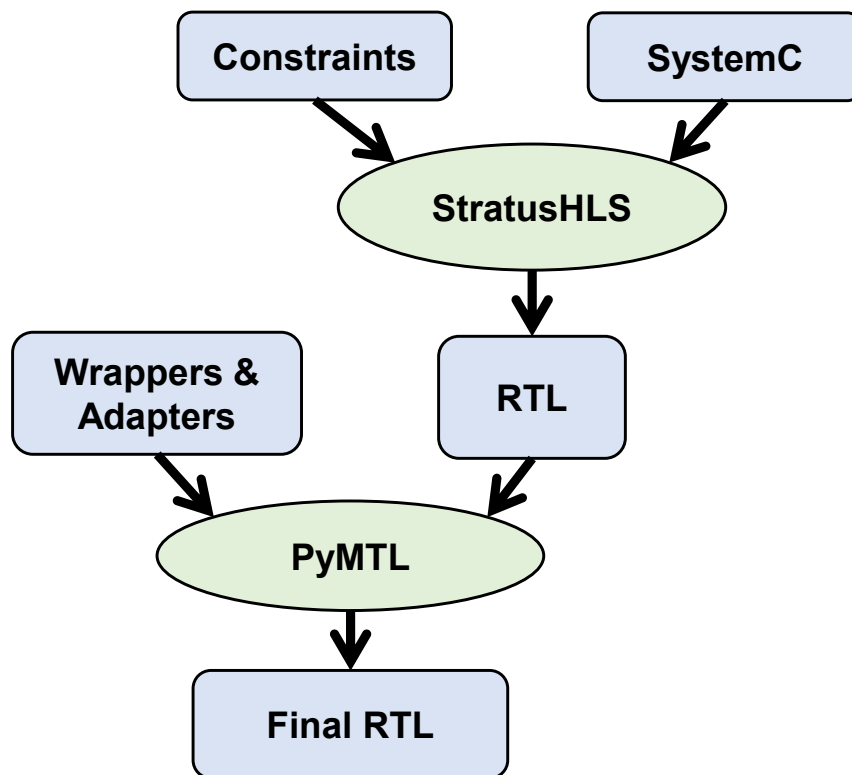


- ▶ Using just the general-purpose tier is  $200\times$  slower than performance target of 60 classifications/second
- ▶ Binarized convolutional layers consume over 97% of the dynamic instruction count
- ▶ Perfect acceleration of just the binarized convolutional layers is still  $5\times$  slower than performance target

## Step 2: App to Accel – BNN Specialized Accelerator



## Step 2: App to Accel – Design Methodology



- ▶ Enabled quick implementation of an accelerator for an emerging algorithm
  - ▷ Algo to initial accelerator in weeks
  - ▷ Rapid design-space exploration
- ▶ Greatly simplified timing closure
  - ▷ Improved clock frequency by 43% in few days
  - ▷ Easily mitigated long paths at interfaces with latency insensitive design
- ▶ Tools are still evolving
  - ▷ Six weeks to debug tool bug with data- dependent access to multi-dimensional arrays

# Performance Benefits of Cooperatively Using the Specialization and Massively Parallel Tiers

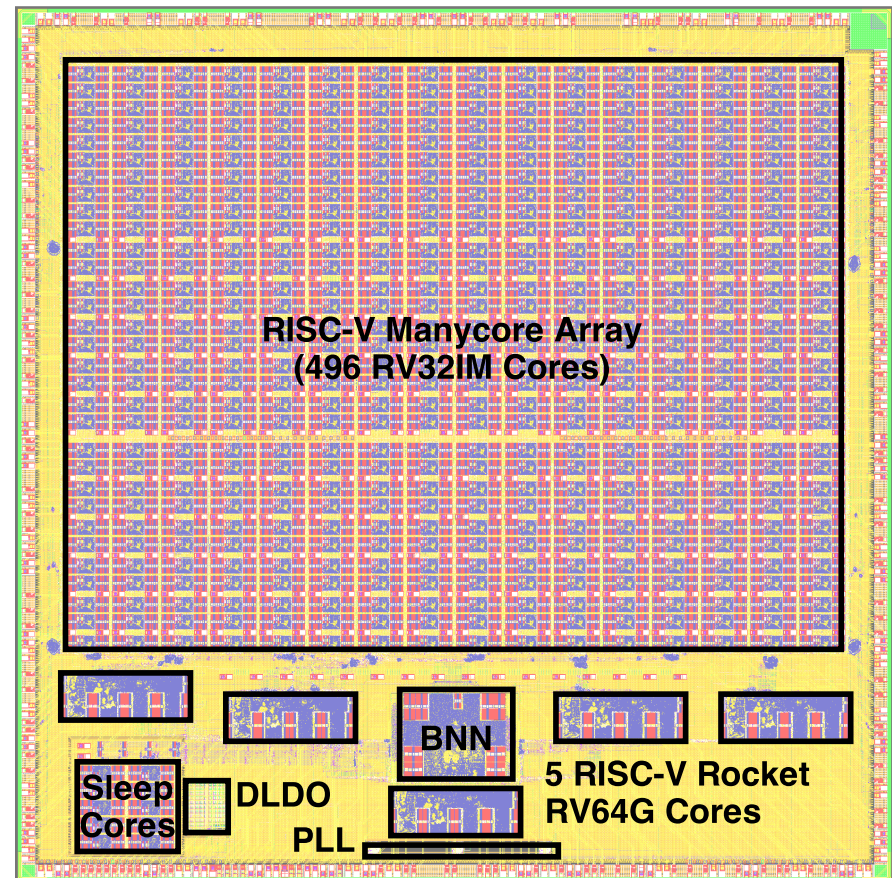
	Step 1 GP Tier	Step 2 Spec Tier	Step 3 Spec+MP Tiers
<b>Runtime (ms)</b>	4,024.0	20.0	3.2
<b>Performance (images/sec)</b>	0.3	50.0	312.5
<b>Power (Watts)</b>	0.1	0.2	0.4
<b>Efficiency (images/J)</b>	2.5	250.0	625.0
<b>Relative Efficiency</b>	1×	100×	250×

- ▶ *GP Tier*: Software implementation assuming ideal performance estimated with an optimistic one instruction per cycle
- ▶ *Spec/MP Tier*: Full-system post-place-and-route gate-level simulation of the spec/MP tiers running with a frequency of 625 MHz

# How were we able to build such a complex SoC?

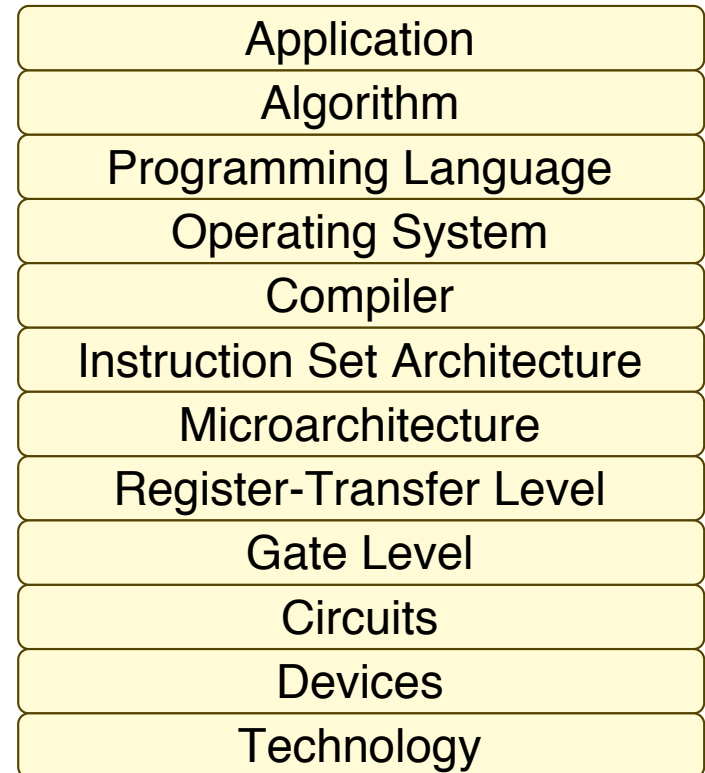
- ▶ 5 × 5mm in TSMC 16 nm FFC
- ▶ 385 million transistors
- ▶ mixed-signal design
- ▶ front- *and* back-end design
  
- ▶ in nine months
- ▶ with 10 core graduate students
- ▶ with little tapeout experience
- ▶ across four locations

**Open-source software and hardware was critical to the success of the project!**



# Leveraging the Open-Source RISC-V Ecosystem

- ▶ **RISC-V Software Toolchain**
  - ▷ Complete, off-the-shelf software stack for both GP and manycore
- ▶ **RISC-V Instruction Set Architecture**
  - ▷ Designed to be modular and extensible
  - ▷ Easy to connect to RoCC interface
  - ▷ Standard instruction verification suites
- ▶ **RISC-V Microarchitecture**
  - ▷ Rocket: high-performance RV64G core
  - ▷ Vanilla-5: high-efficiency RV32IM core
  - ▷ Standard on-chip network specs
- ▶ **RISC-V VLSI and System Design**
  - ▷ Previous spins of chips for reference
  - ▷ Turn-key FPGA gateware



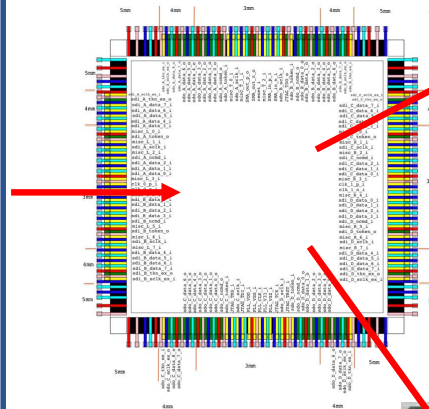
Developed at UC Berkeley  
<http://riscv.org>

# Leveraging the Open-Source BaseJump Ecosystem

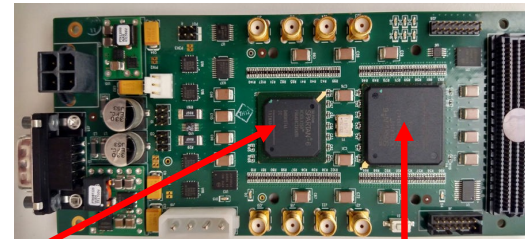
**BaseJump STL:**  
Standard library of  
hardware components

HDL Design

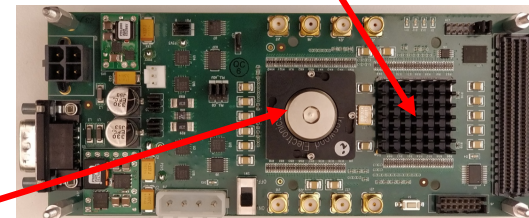
**BaseJump Socket:**  
IO Pading



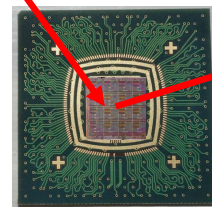
**BaseJump DoubleTrouble:**  
HW Emulation Motherboard



**BaseJump:**  
Open FPGA  
Firmware



**BaseJump RealTrouble:**  
Bring-up Motherboard



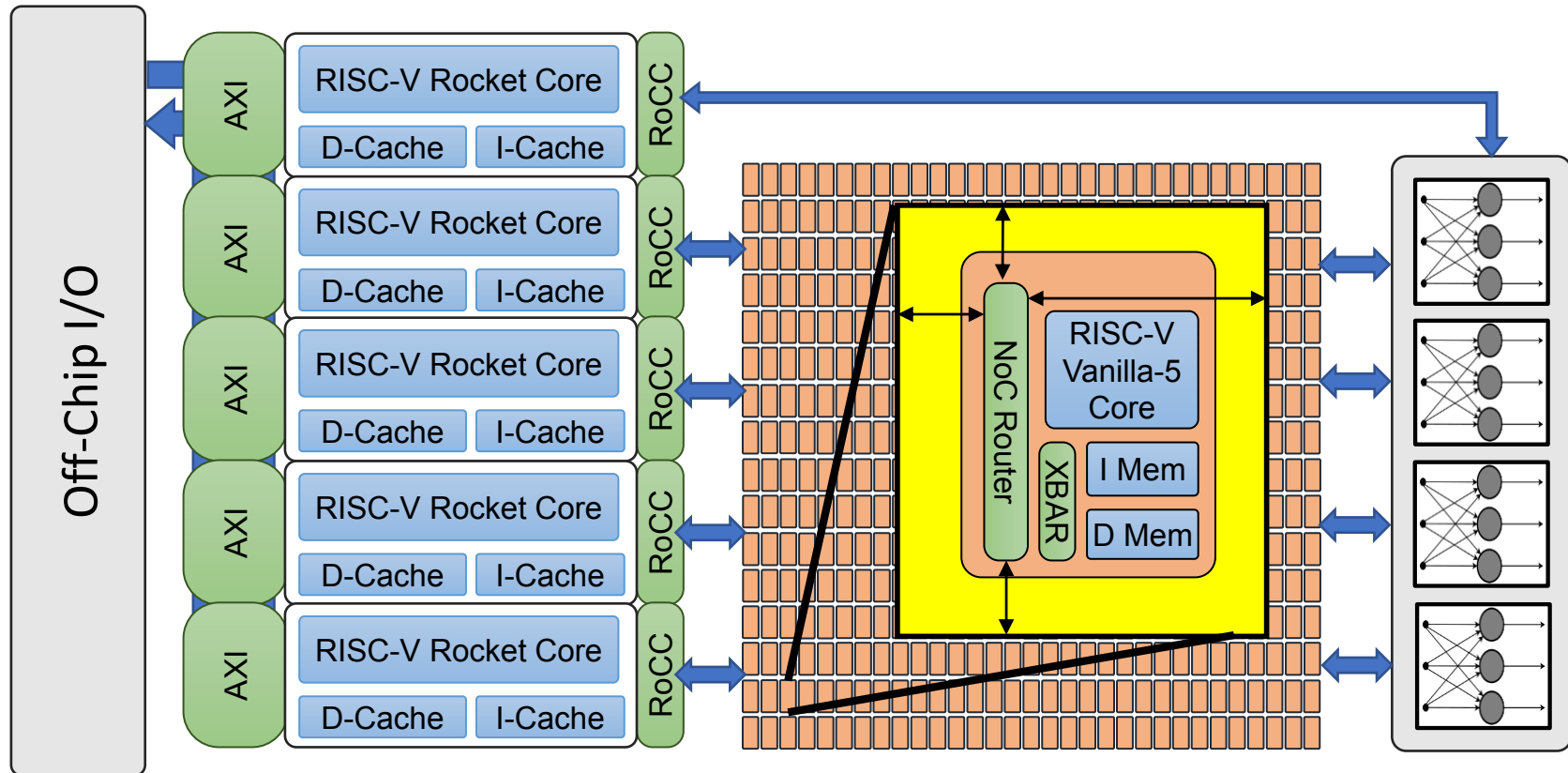
**BaseJump Socket:**  
BGA Package

Developed at UCSD and  
University of Washington

<http://bjump.org>



# Contributing Back to the Open-Source Ecosystem



<http://opencelerity.org>

Upstream patches to gem5 for multi-core RISC-V simulation

# Celerity Publications

- ▶ Tutu Ajayi et al., “Celerity: An Open-Source RISC-V Tiered Accelerator Fabric.” *29th ACM/IEEE Symp. on High-Performance Chips (HOTCHIPS)*, Aug. 2017.
- ▶ Scott Davidson et al., “The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips.” *IEEE Micro*, 38(2):3041, Mar/Apr. 2018.
- ▶ Austin Rovinski et al., “A 1.4 GHz 695 Giga RISC-V Inst/s 496-core Manycore Processor with Mesh On-Chip Network and an All-Digital Synthesized PLL in 16nm CMOS.” *IEEE Symp. on VLSI Circuits (VLSI)*, June 2019.
- ▶ Austin Rovinski et al., “Evaluating Celerity: A 16nm 695 Giga-RISC-V Instructions/s Manycore Processor with Synthesizable PLL.” *IEEE Solid-State Circuits Letters*, 2(12):289292, Dec. 2019.

THEME ARTICLE: Hot Chips

## The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric:

Fast Architectures and Design Methodologies for Fast Chips

Rapidly emerging workloads require rapidly developed chips. The Celerity 16-nm open-source SoC was implemented in nine months using an architectural trifecta to minimize development time: a general-purpose tier comprised of open-source Linux-capable RISC-V cores, a massively parallel tier comprised of a RISC-V tiled manycore array that can be scaled to arbitrary sizes, and a specialization tier that uses high-level synthesis (HLS) to create an algorithmic neural-network accelerator. These tiers are tied together with an efficient heterogeneous remote store programming model on top of a flexible partial global address space memory system.

Emerging workloads have extremely strict energy-efficiency and performance requirements that are difficult to attain. Increasingly, we see that specialized hardware accelerators are necessary to attain these requirements. But accelerator development is time-intensive, and accelerator behavior cannot be easily modified to adapt to changing workload properties. These factors motivate new architec-

IEEE Micro  
March/April 2018

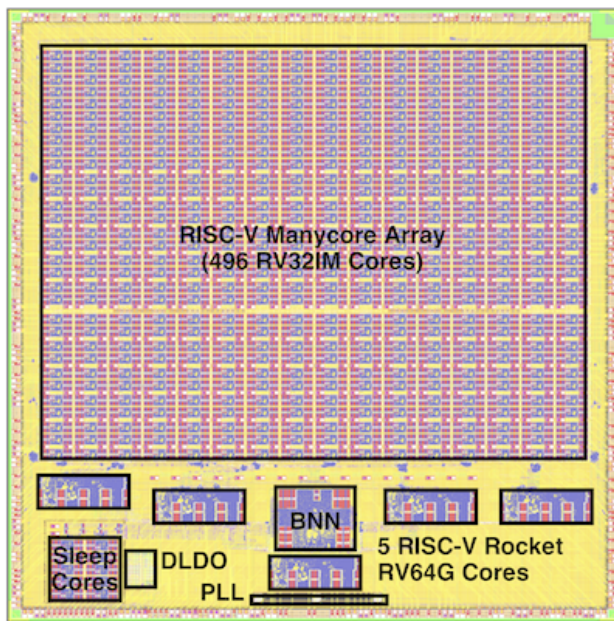
30

Published by the IEEE Computer Society  
0272-1732/18/\$33.00 ©2018 IEEE

```

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 SoC Design

## ► The PyMTL3 Framework

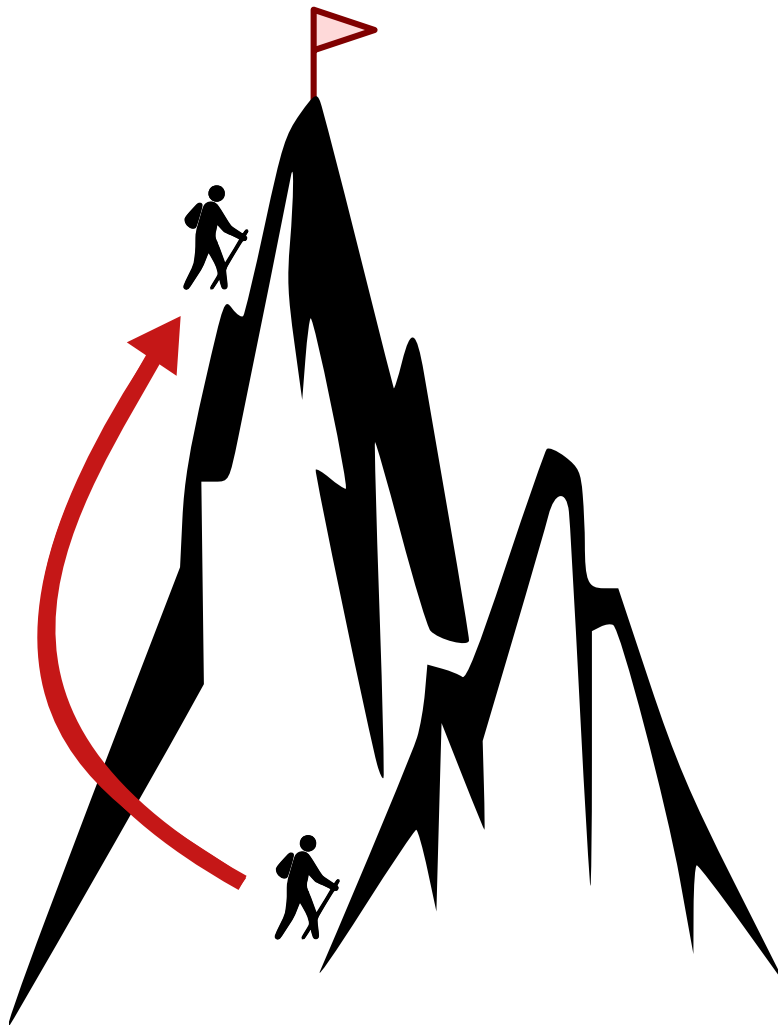
- ▷ PyMTL3 Motivation
- ▷ PyMTL3 Overview
- ▷ PyMTL3 Demo
- ▷ PyMTL3 & Open-Source Hardware

## ► The Celerity SoC

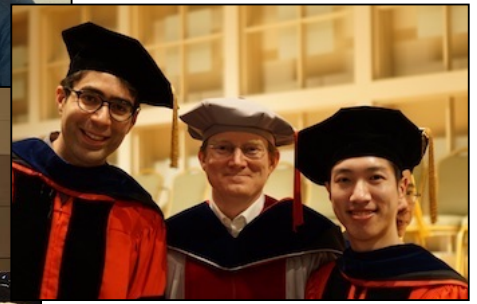
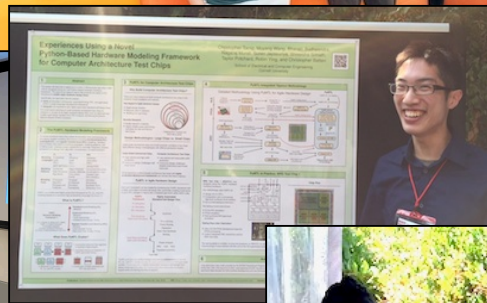
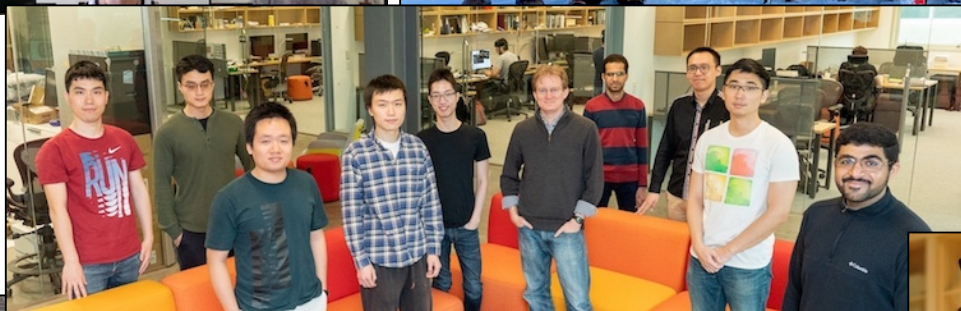
- ▷ Celerity Architecture
- ▷ Celerity Case Study
- ▷ Celerity & Open-Source Hardware

## ► 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 **Derek Lockhart** (original PyMTL2 developer), Ji Kim, Shreesha Srinath, Berkin Ilbeyi, Yixiao Zhang, Jacob Glueck, Aaron Wisner, Gary Zibrat, Christopher Torng, Cheng Tan, Raymond Yang, Kaishuo Cheng, Jack Weber, Carl Friedrich Bolz, David Maclver, and Zac Hatfield-Dodds for their help designing, developing, testing, and using PyMTL2 and PyMTL3

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.