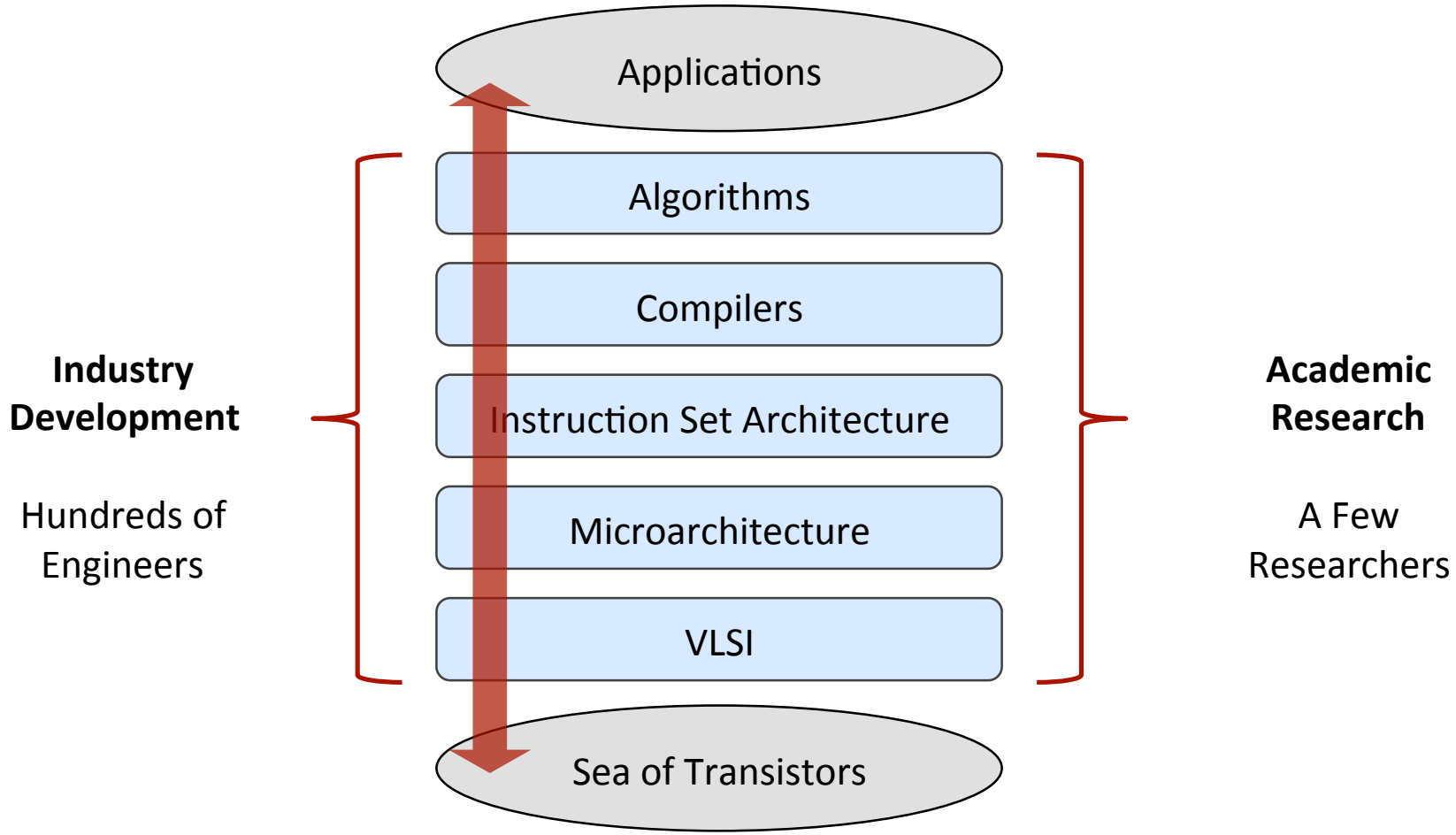
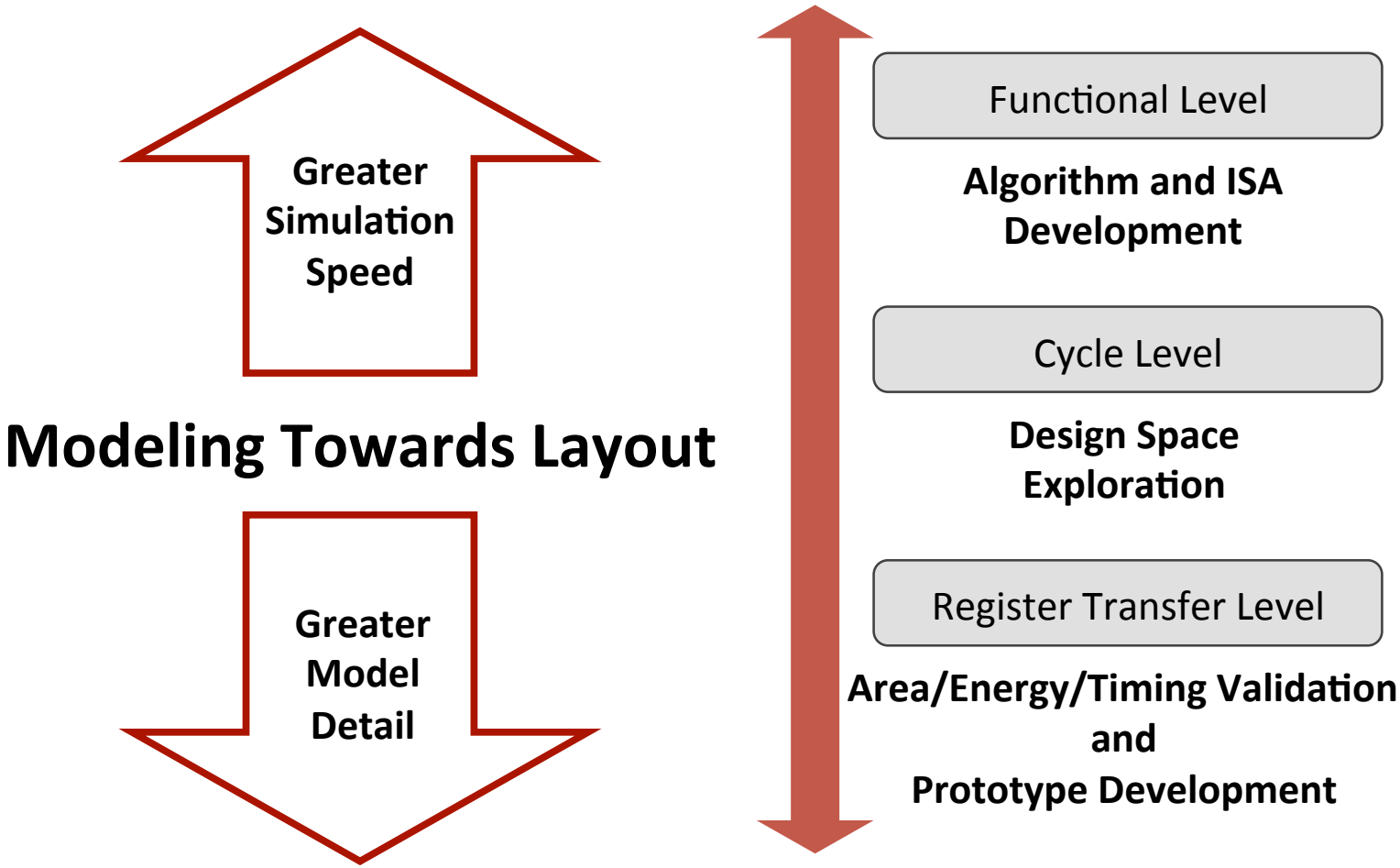
PyMTL/Pydgin Tutorial Schedule

8:30am – 8:50am	Virtual Machine Installation and Setup
8:50am – 9:00am	*Presentation:* PyMTL/Pydgin Tutorial Overview
9:00am – 9:10am	*Presentation:* Introduction to Pydgin
9:10am – 10:00am	*Hands-On:* Adding a GCD Instruction using Pydgin
10:00am – 10:10am	*Presentation:* Introduction to PyMTL
10:10am – 11:00am	*Hands-On:* PyMTL Basics with Max/RegIncr
11:00am – 11:30am	Coffee Break
11:30am – 11:40am	*Presentation:* Multi-Level Modeling with PyMTL
11:40am – 12:30pm	*Hands-On:* FL, CL, RTL Modeling of a GCD Unit

Computer Architecture Research Abstractions



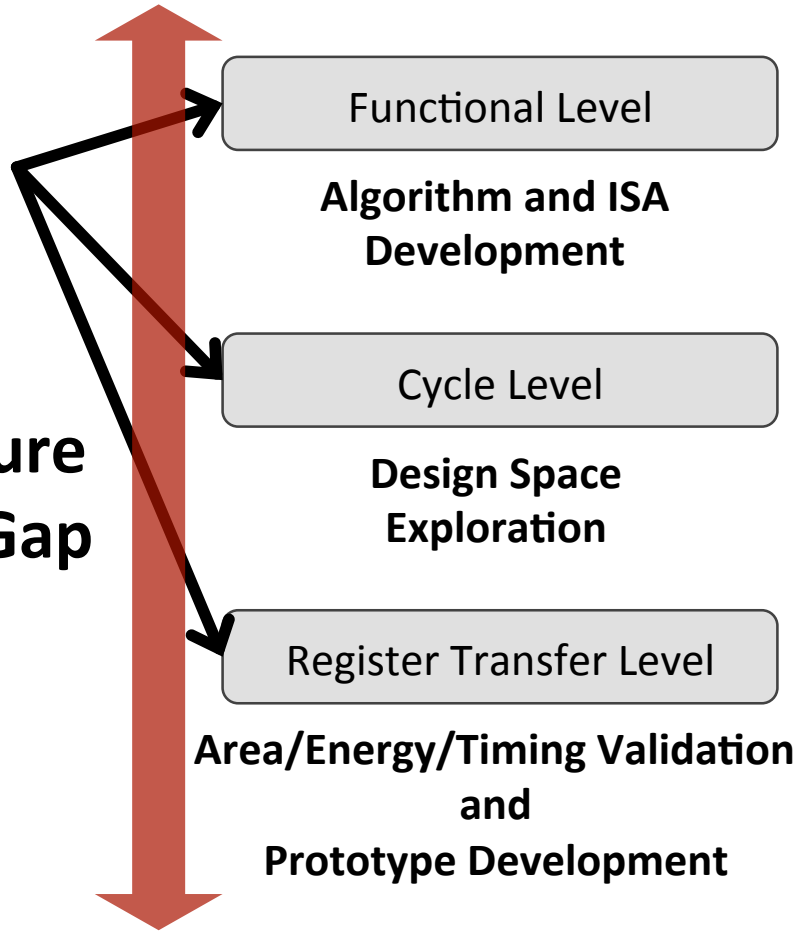
Computer Architecture Research Methodologies



Computer Architecture Research Toolflows

Different languages, patterns, and tools!

The Computer Architecture Research Methodology Gap

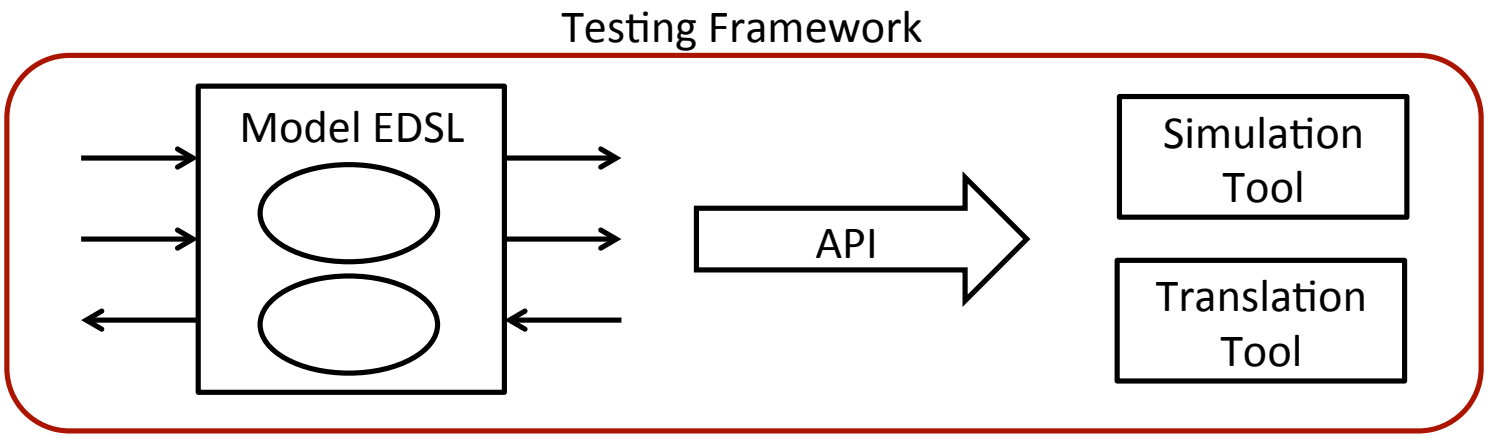


Great Ideas From Prior Frameworks

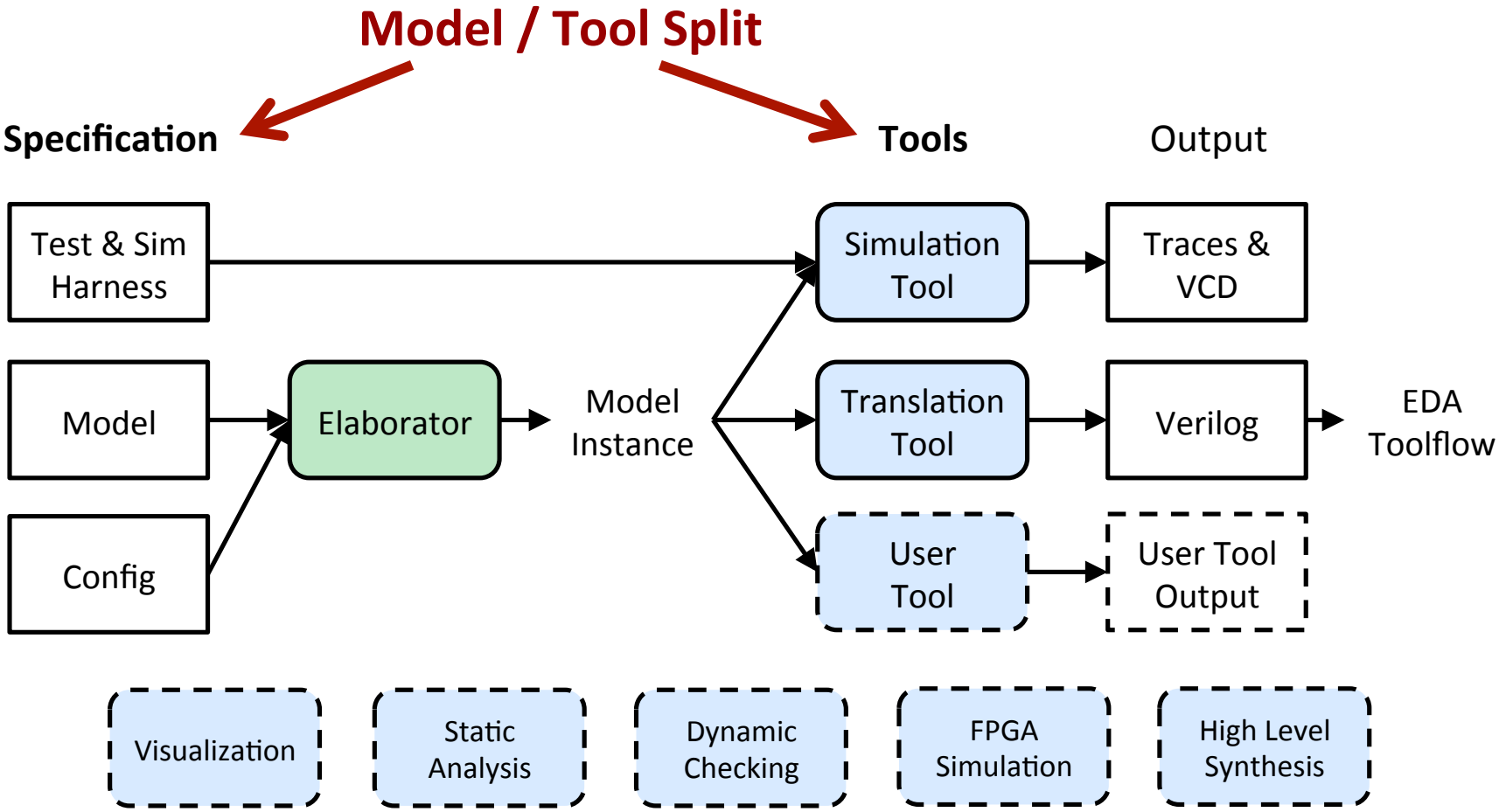
- **Concurrent-Structural Modeling** (Liberty, Cascade, SystemC) — Consistent interfaces across abstractions
- **Unified Modeling Languages** (SystemC) — Unified design environment for FL, CL, RTL
- **Hardware Generation Languages** (Chisel, Genesis2, BlueSpec, MyHDL) — Productive RTL design space exploration
- **HDL-Integrated Simulation Frameworks** (Cascade) — Productive RTL validation and cosimulation
- **Latency-Insensitive Interfaces** (Liberty, BlueSpec) — Component and test bench reuse

What is PyMTL?

- A Python EDSL for concurrent-structural hardware modeling
- A Python API for analyzing models described in the PyMTL EDSL
- A Python tool for simulating PyMTL FL, CL, and RTL models
- A Python tool for translating PyMTL RTL models into Verilog
- A Python testing framework for model validation



The PyMTL Framework



PyMTL 101: Traditional Model in Python

```
def max_unit( input_list ):
  return max( input_list )
```

[3, 1, 2, 0] ---> f(x) ---> 3

PyMTL 101: Model in PyMTL Embedded DSL

```
def max_unit( input_list ):
  return max( input_list )
```

[3, 1, 2, 0] ---> f(x) ---> 3

```
class MaxUnitFL( Model ):
  def __init__( s, nbits, nports ):

    s.in_  = InPort[nports]( nbits )
    s.out  = OutPort( nbits )


    @s.tick_fl
    def logic():
      s.out.next = max( s.in_ )
```