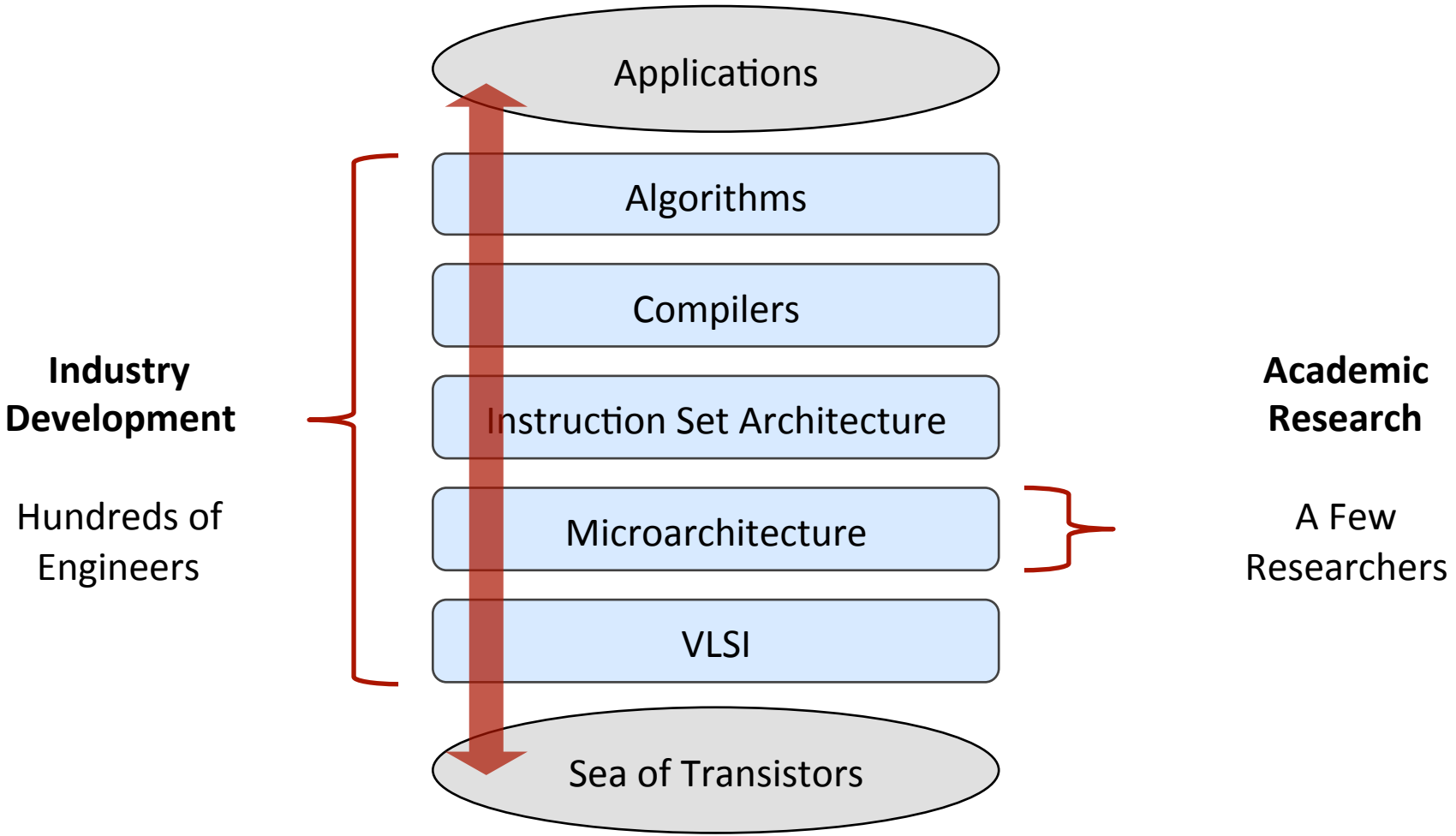
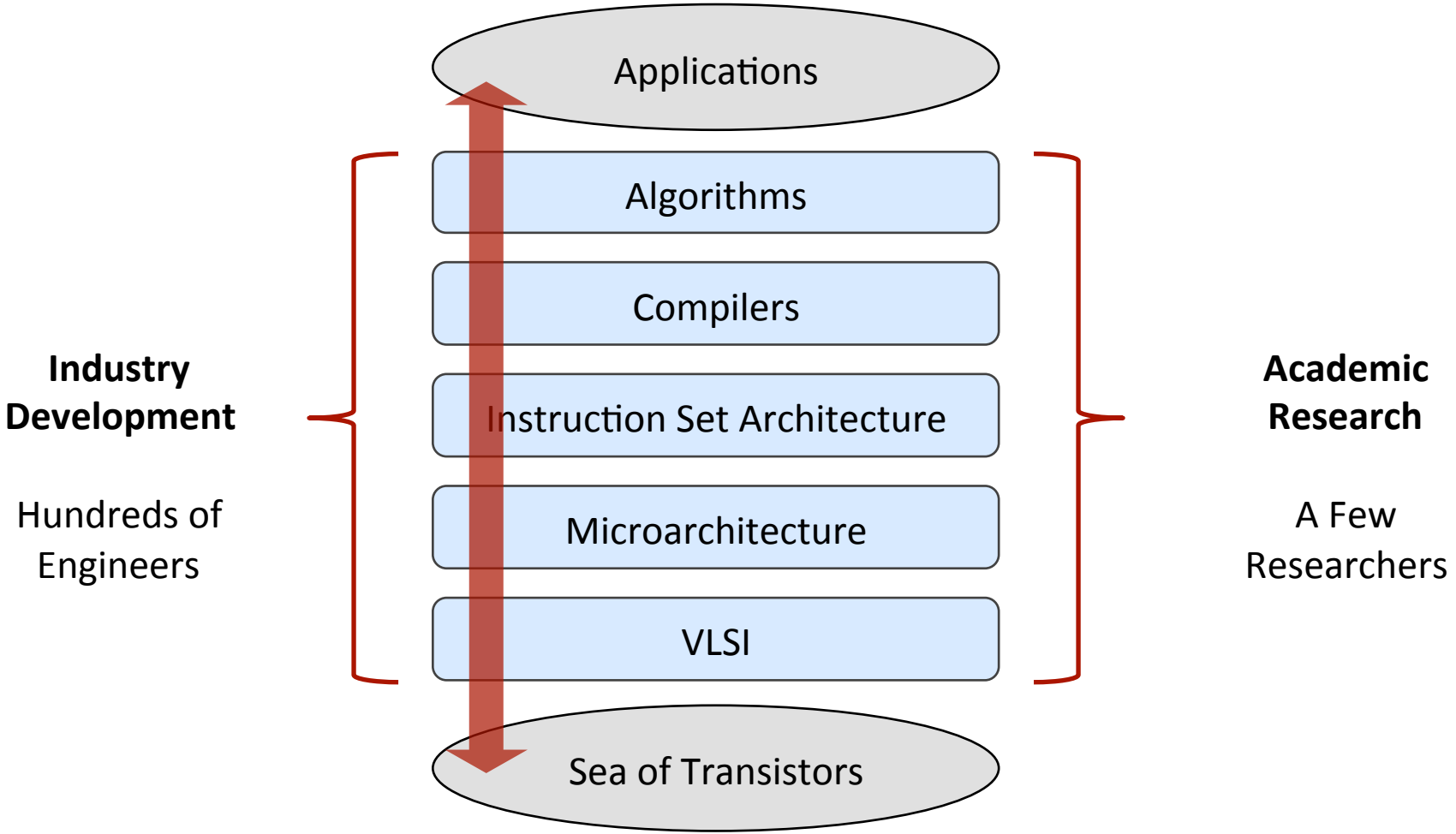
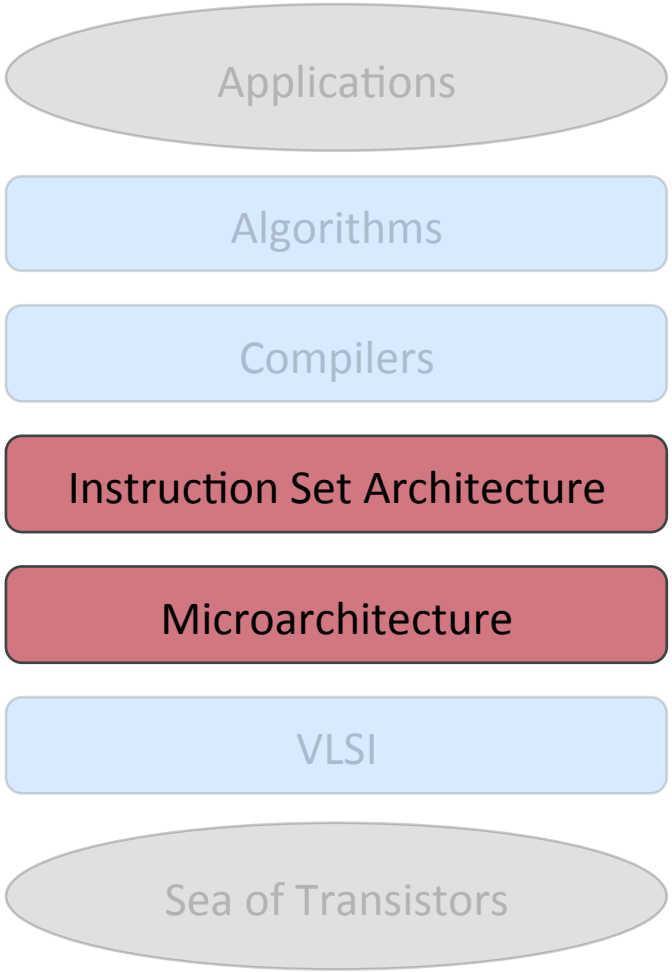
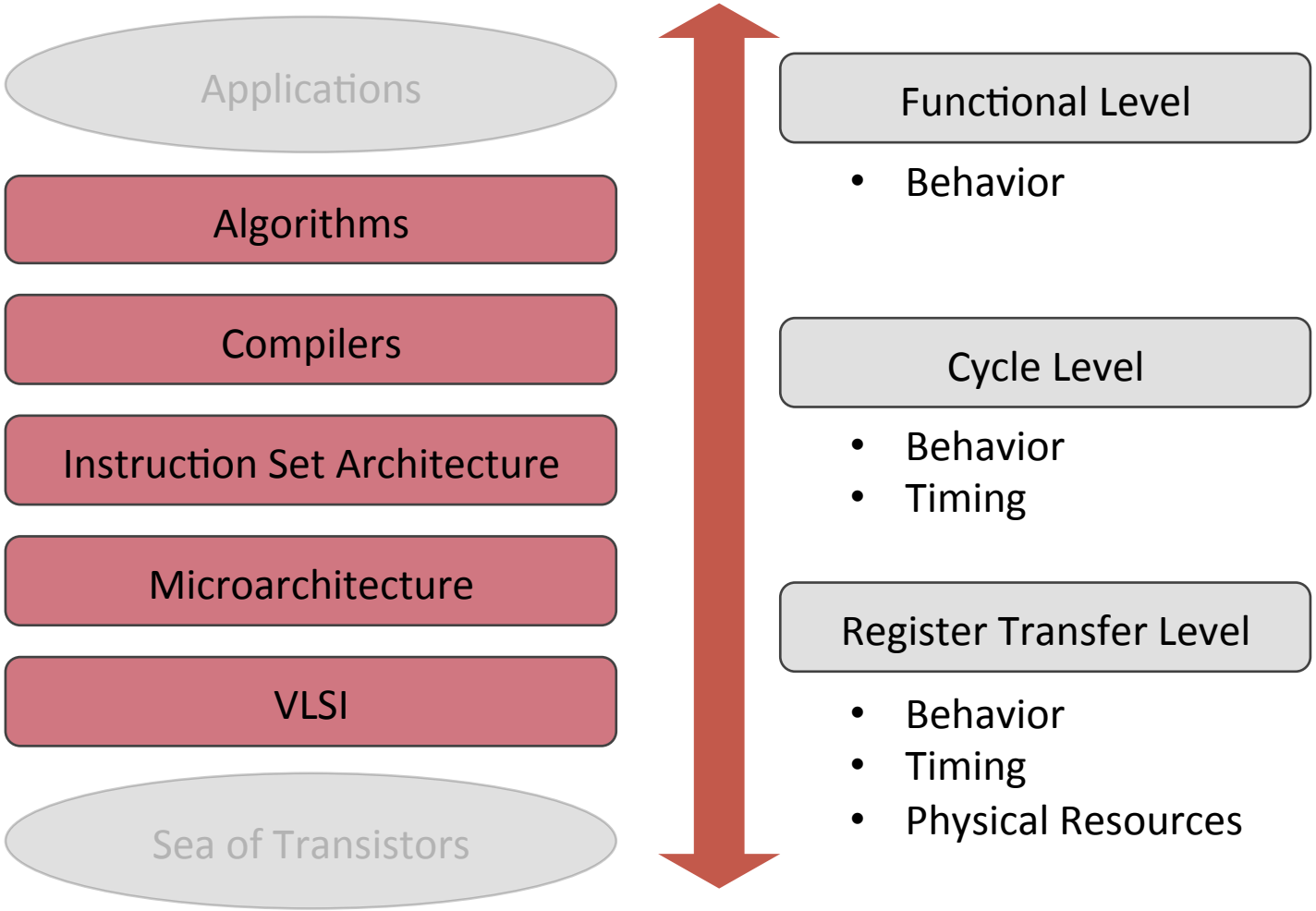

Applications
Algorithms
Compilers
Instruction Set Architecture
Microarchitecture
VLSI
Sea of Transistors
Industry Development
Hundreds of Engineers
Academic Research
A Few Researchers

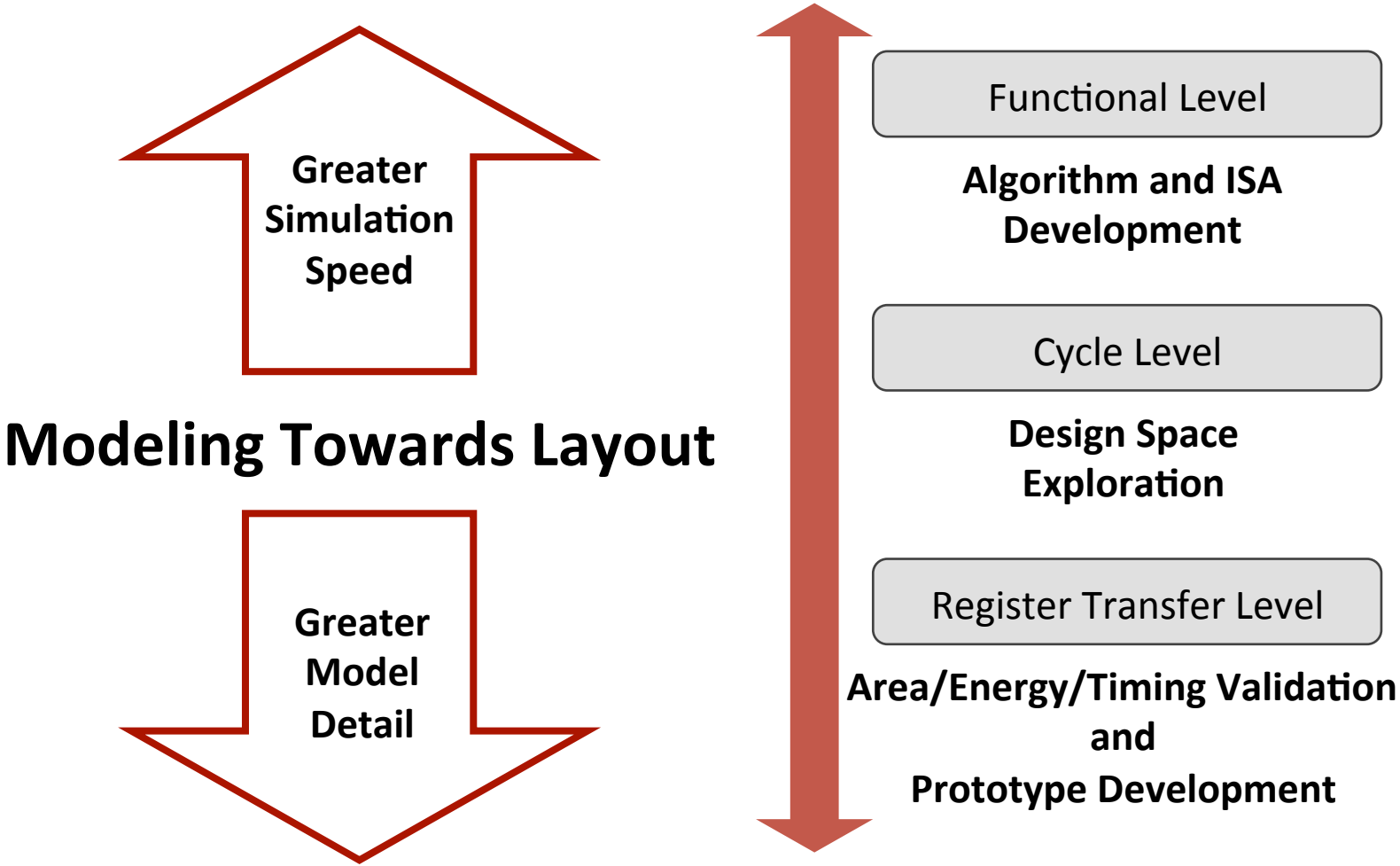

Applications
Algorithms
Compilers
Instruction Set Architecture
Microarchitecture
VLSI
Sea of Transistors
Industry Development
Hundreds of Engineers
Academic Research
A Few Researchers




**Cycle Level**

- Behavior
- Timing

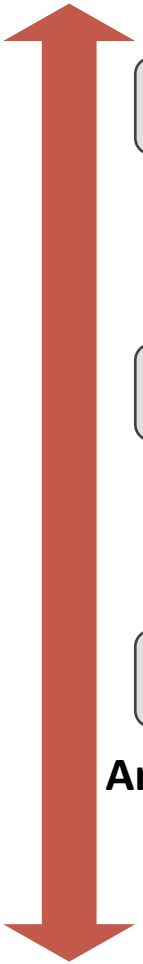

Applications
Algorithms
Compilers
Instruction Set Architecture
Microarchitecture
VLSI
Sea of Transistors
Functional Level
• Behavior
Cycle Level
• Behavior
• Timing
Register Transfer Level
• Behavior
• Timing
• Physical Resources


Greater Simulation Speed
Modeling Towards Layout
Greater Model Detail
Functional Level
Algorithm and ISA Development
Cycle Level
Design Space Exploration
Register Transfer Level
Area/Energy/Timing Validation and Prototype Development

MATLAB/Python Algorithm or
C++ Instruction Set Simulator

C++ Computer Architecture
Simulation Framework
(Object-Oriented)

Verilog or VHDL Design with
EDA Toolflow
(Concurrent-Structural)



Functional Level

**Algorithm and ISA Development**

Cycle Level

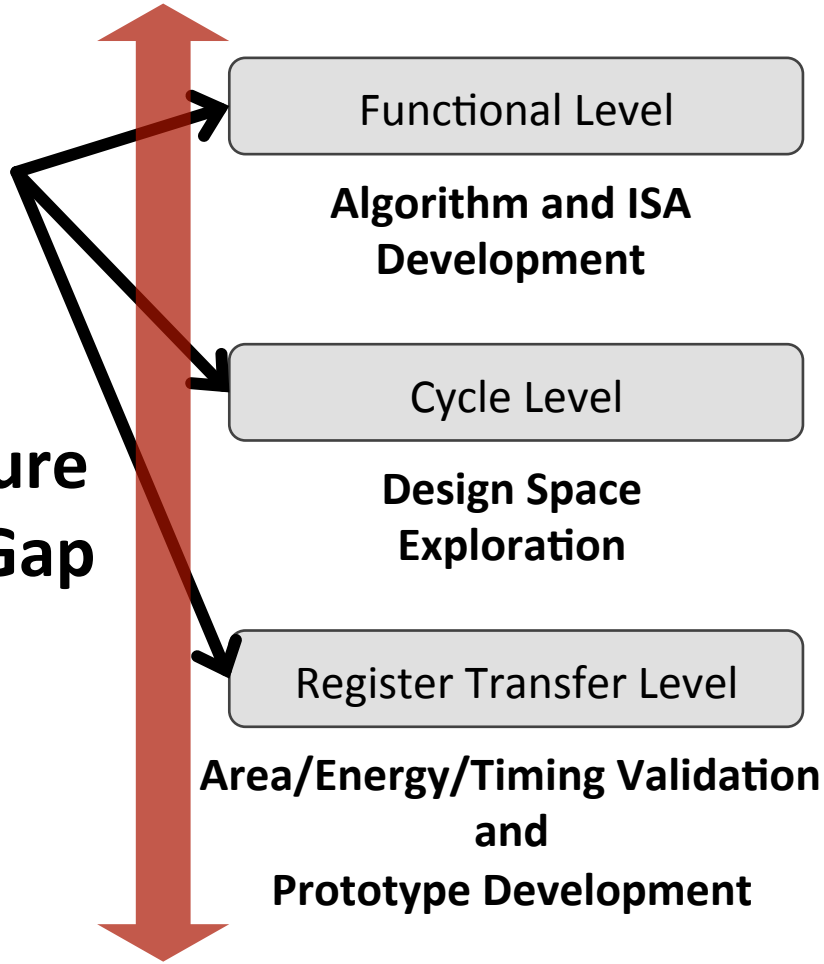
**Design Space Exploration**

Register Transfer Level

**Area/Energy/Timing Validation and Prototype Development**

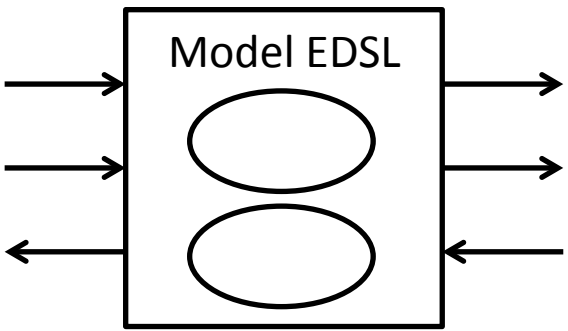
Different languages, patterns, and tools!

**The Computer Architecture Research Methodology Gap**

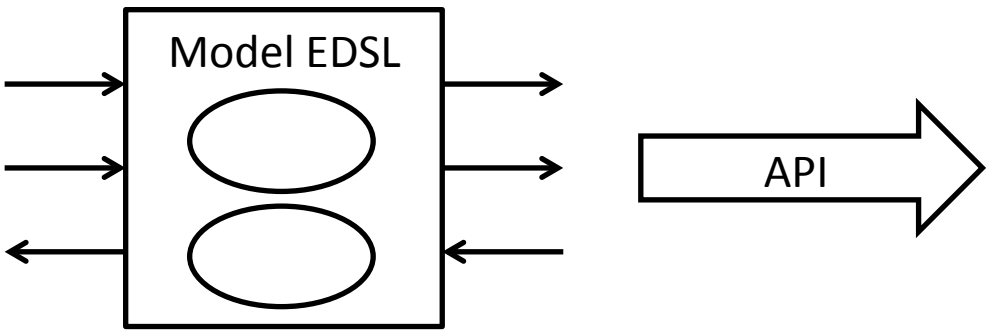


| | |
|---|---|
| • **Concurrent-Structural Modeling** (Liberty, Cascade, SystemC) | Consistent interfaces across abstractions |
| • **Unified Modeling Languages** (SystemC) | Unified design environment for FL, CL, RTL |
| • **Hardware Generation Languages** (Chisel, Genesis2, BlueSpec, MyHDL) | Productive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

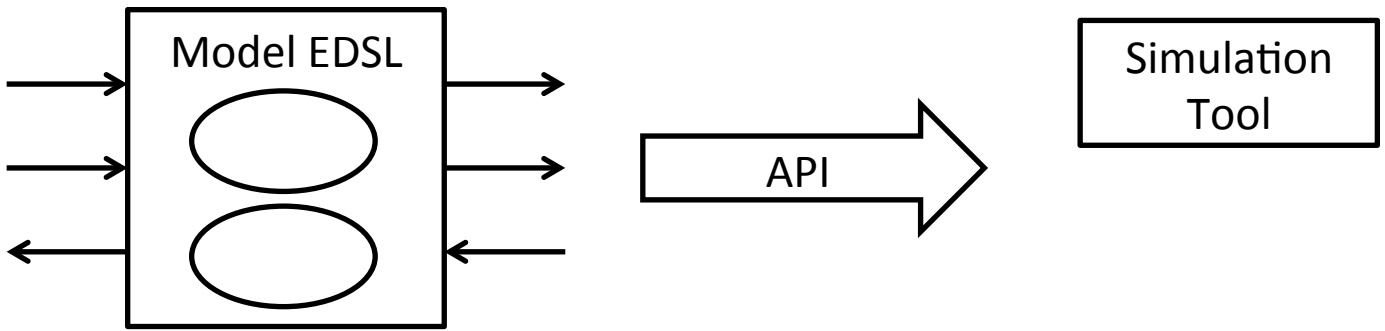
- A Python EDSL for concurrent-structural hardware modeling



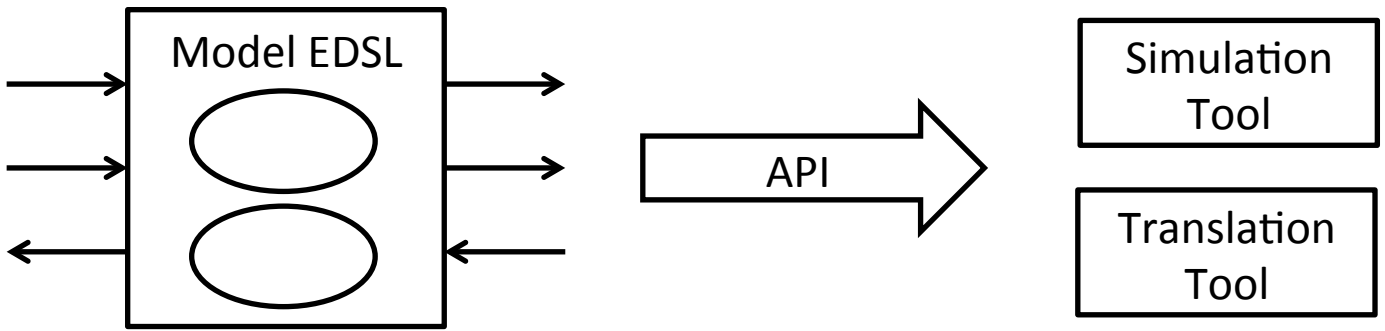
- A Python EDSL for concurrent-structural hardware modeling
- A Python API for analyzing models described in the PyMTL EDSL



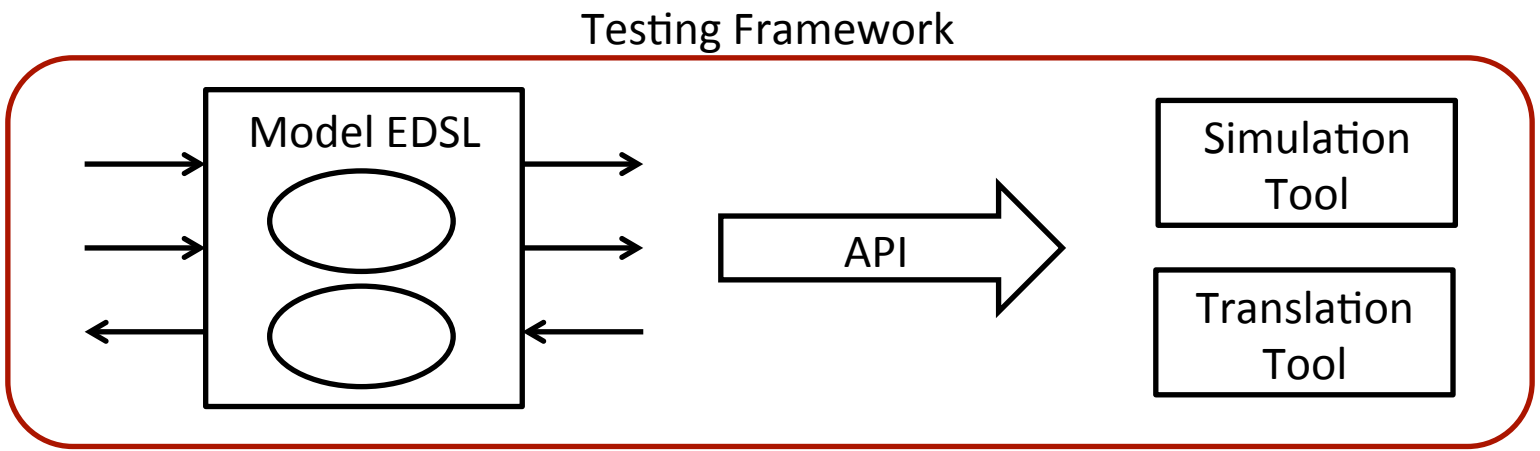
- A Python EDSL for concurrent-structural hardware modeling
- A Python API for analyzing models described in the PyMTL EDSL
- A Python tool for simulating PyMTL FL, CL, and RTL models



- A Python 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 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



# Model / Tool Split

**Specification**

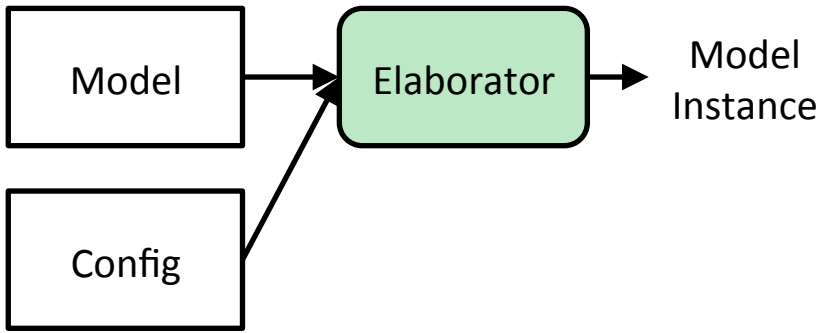
**Tools**

Output

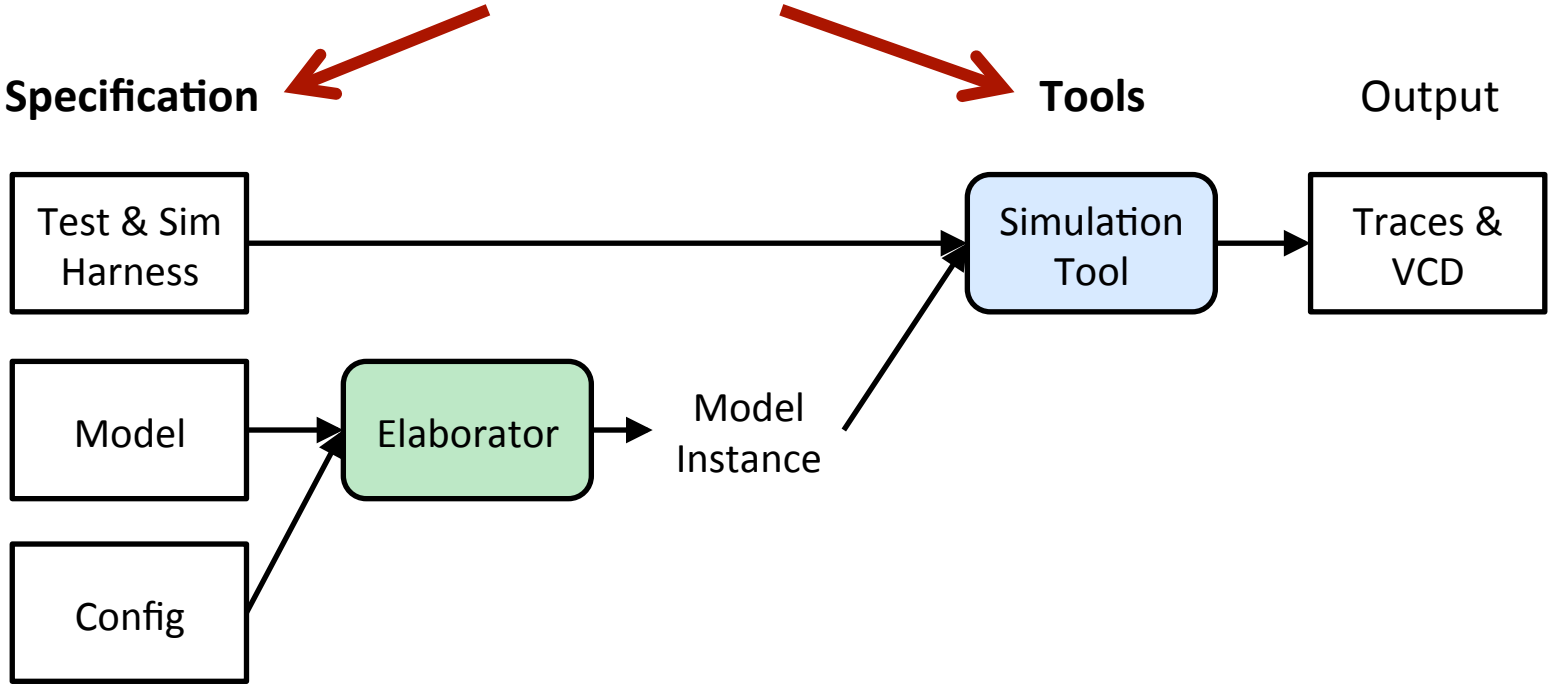
Model

# Model / Tool Split

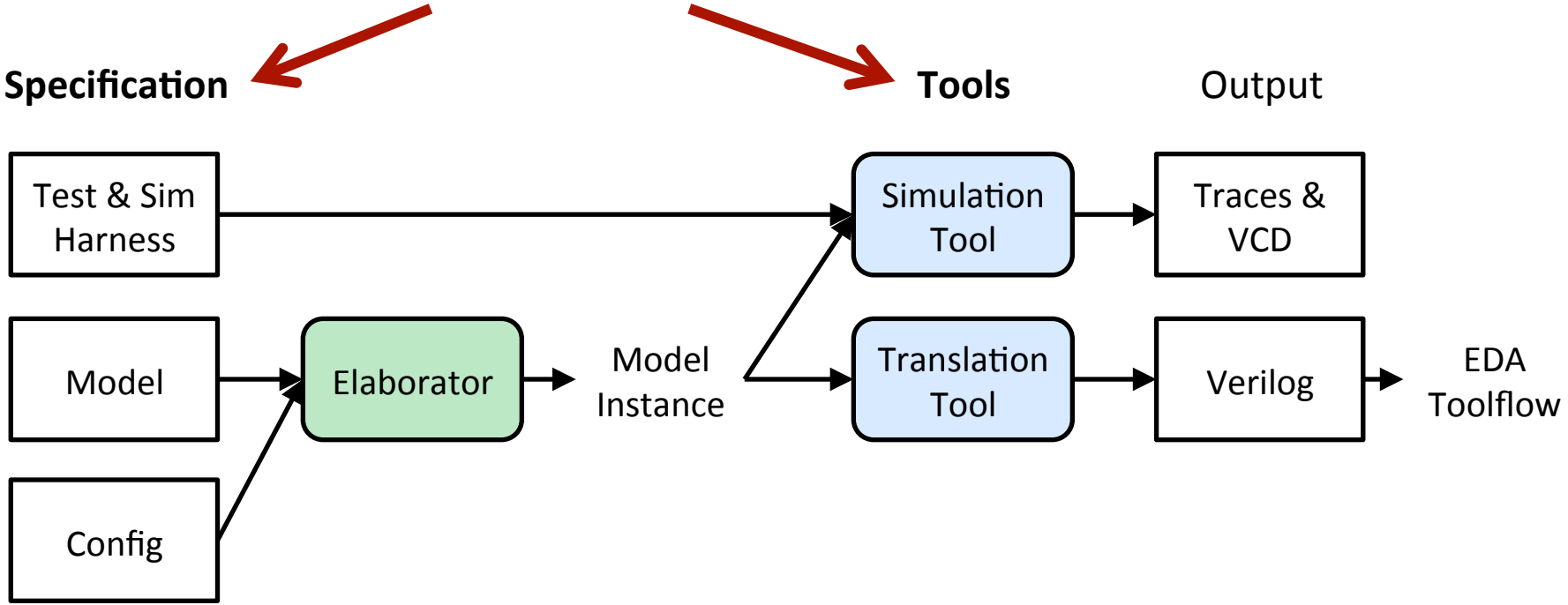
**Specification** **Tools** Output

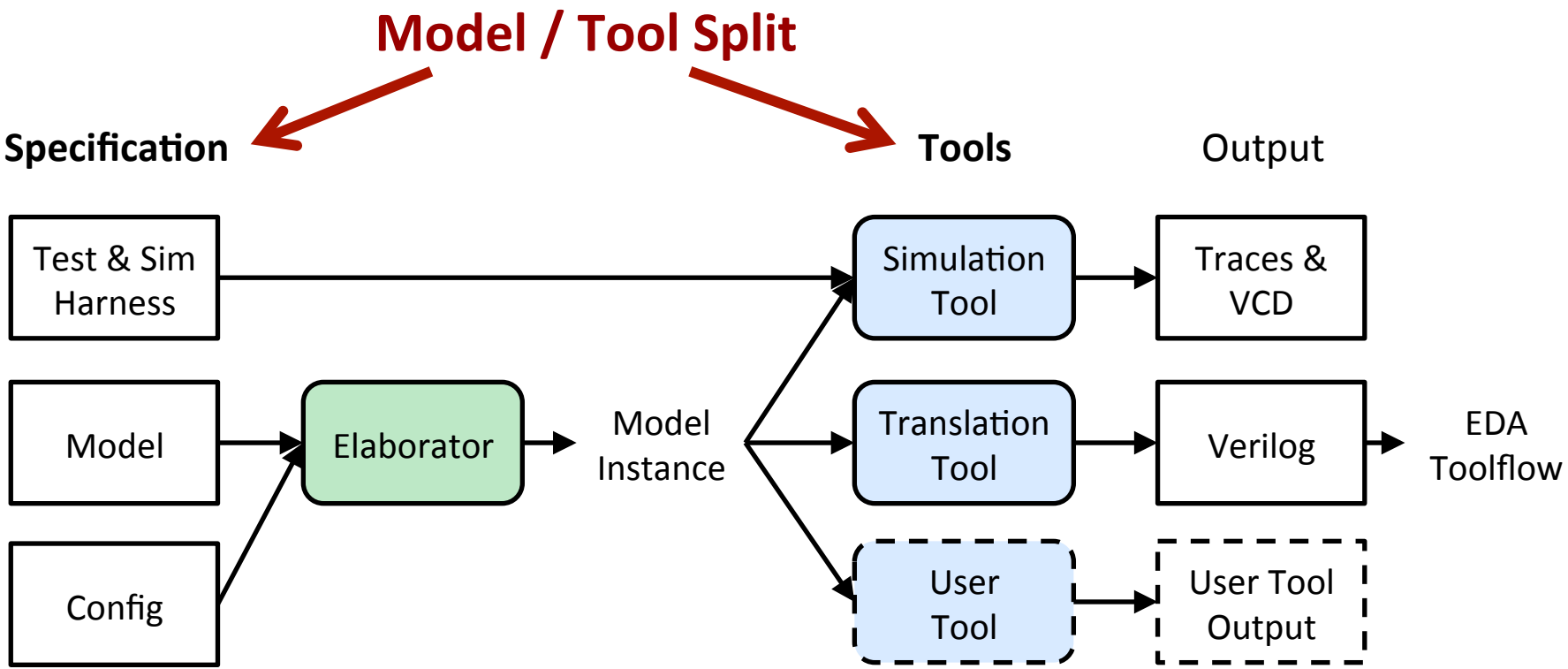


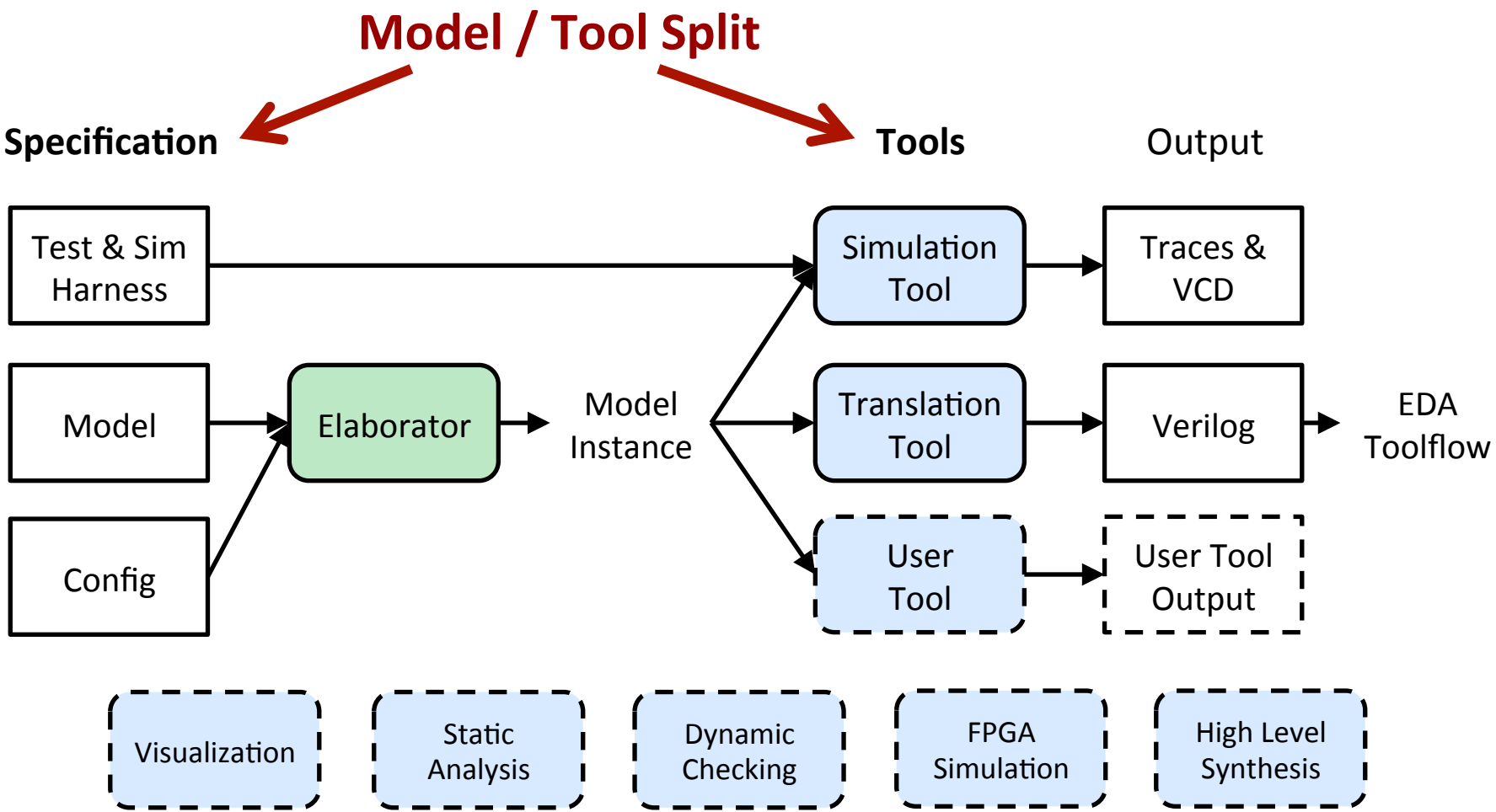
# Model / Tool Split



# Model / Tool Split




Model / Tool Split
Specification
Tools
Output
Test & Sim Harness
Model
Config
Elaborator
Model Instance
Simulation Tool
Translation Tool
User Tool
Traces & VCD
Verilog
User Tool Output
EDA Toolflow


Model / Tool Split
Specification
Tools
Output
Test & Sim Harness
Model
Config
Elaborator
Model Instance
Simulation Tool
Translation Tool
User Tool
Traces & VCD
Verilog
User Tool Output
EDA Toolflow
Visualization
Static Analysis
Dynamic Checking
FPGA Simulation
High Level Synthesis

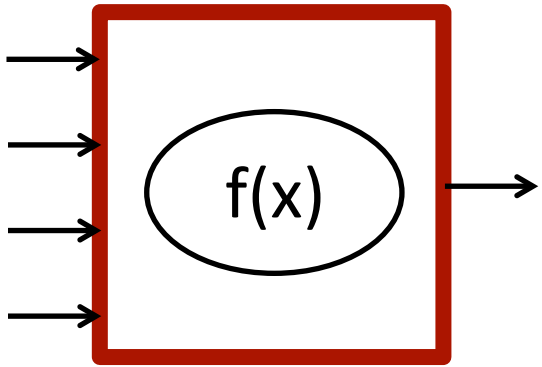
```
def max_unit( input_list ):
  return max( input_list )
```

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

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

class MaxUnitFL( Model ):
```

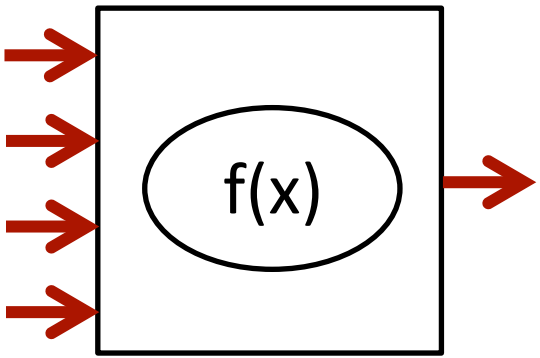
[ 3, 1, 2, 0 ] ---> f(x) ---> 3



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

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

```
class MaxUnitFL( Model ):
  def __init__( s, nbits, nports ):
    dtype  = Bits( nbits )
    s.in_  = InPort[nports]( dtype )
    s.out  = OutPort( dtype )
```

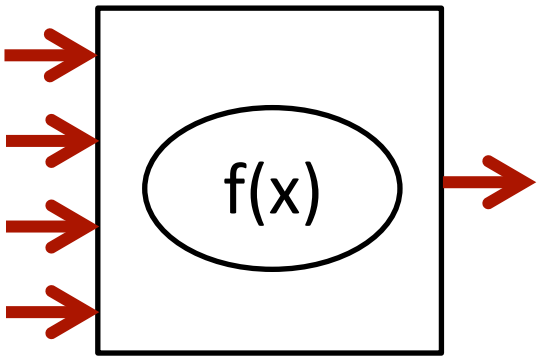


```
def max_unit( input_list ):
  return max( input_list )
```

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

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

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



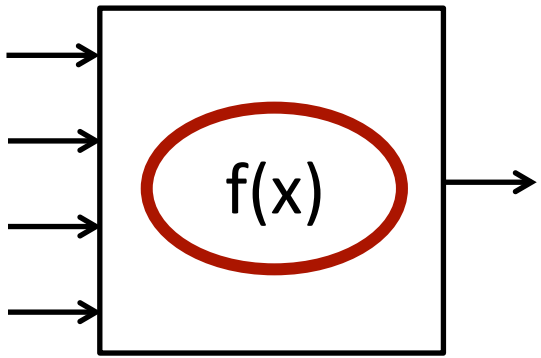
```
def max_unit( input_list ):
  return max( input_list )
```

[ 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():
```



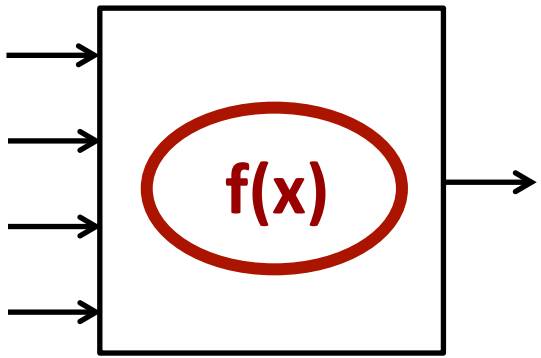
```
def max_unit( input_list ):
  return max( input_list )
```

[ 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_ )
```



```
def sorter_network( input_list ):
  return sorted( input_list )
```

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