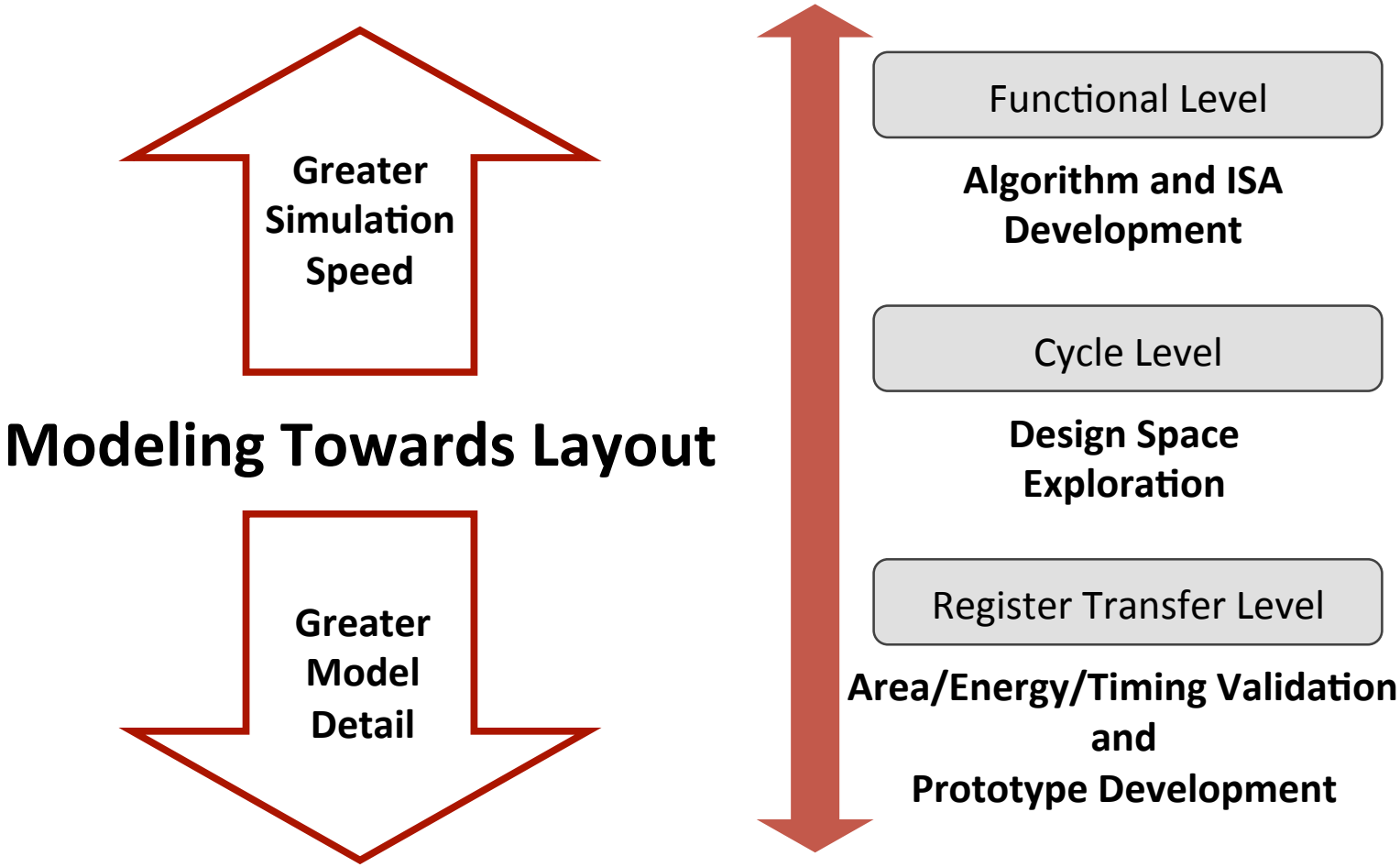
PyMTL/Pydgin Tutorial Schedule

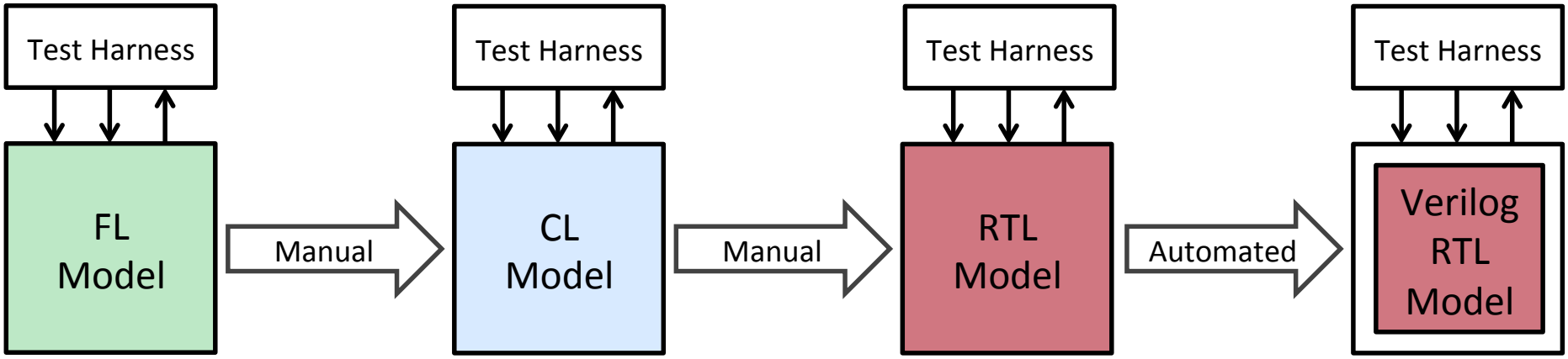
8:30am – 8:50am	Virtual Machine Installation and Setup
8:50am – 9:00am	*Presentation:* PyMTL/Pydgin Tutorial Overview
9:00am – 9:10am	*Presentation:* Introduction to Pydgin
9:10am – 10:00am	*Hands-On:* Adding a GCD Instruction using Pydgin
10:00am – 10:10am	*Presentation:* Introduction to PyMTL
10:10am – 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

Multi-Level Modeling



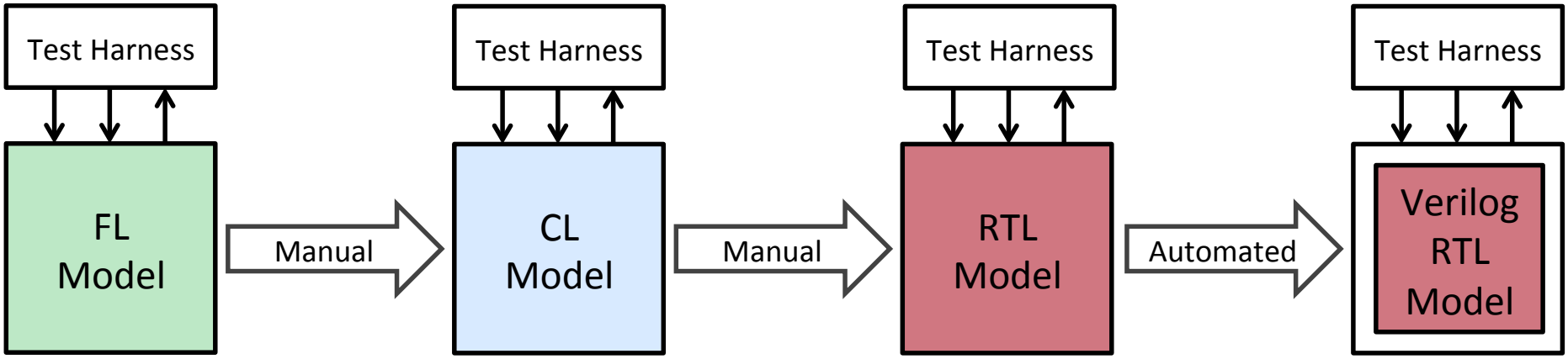
Multi-Level Modeling in PyMTL

- FL modeling allows for the rapid creation of a working model. Designers can quickly experiment with interfaces and protocols.
- This design is *manually refined* into a PyMTL CL model that includes timing, which is useful for rapid design space exploration.
- Promising architectures can again be *manually refined* into a PyMTL RTL implementation to accurately model resources.



Multi-Level Modeling in PyMTL

- Verilog generated from PyMTL RTL can be passed to an EDA toolflow for accurate area, energy, and timing estimates.
- Throughout this process, the same PyMTL test harnesses can used to verify each model!
- Requires good design, the use of latency-insensitive interfaces helps considerably.



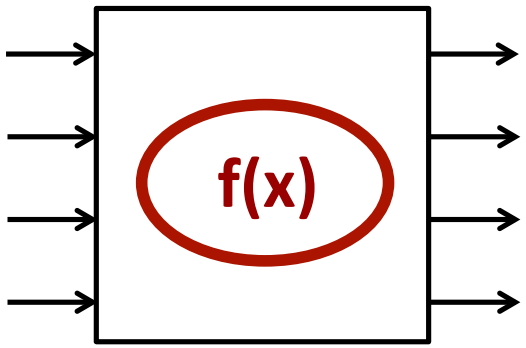
FL Model in PyMTL

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

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

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

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




```
    @s.tick_fl
    def logic():
      for i, v in enumerate( sorted( s.in_ ) ):
        s.out[i].next = v
```

CL Model in PyMTL

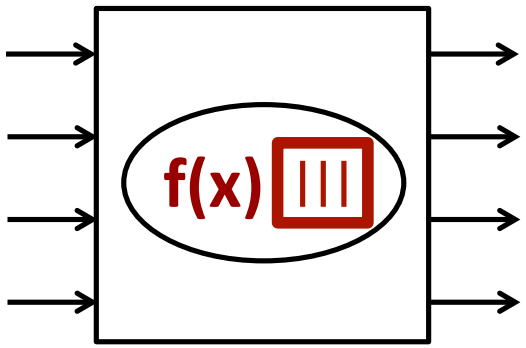
```
def sorter_network( input_list ):
  return sorted( input_list )
```

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

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

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


    @s.tick_cl
    def logic():
      # behavioral logic + timing delays
```



RTL Model in PyMTL

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

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

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

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


    @s.tick_rtl
    def seq_logic():
      # sequential logic

    @s.combinational
    def comb_logic():
      # combinational logic
```