<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30am –</td>
<td>Virtual Machine Installation and Setup</td>
</tr>
<tr>
<td>8:50am –</td>
<td>Presentation: PyMTL/Pydgin Tutorial Overview</td>
</tr>
<tr>
<td>9:00am –</td>
<td>Presentation: Introduction to Pydgin</td>
</tr>
<tr>
<td>9:10am –</td>
<td>Hands-On: Adding a GCD Instruction using Pydgin</td>
</tr>
<tr>
<td>10:00am –</td>
<td>Presentation: Introduction to PyMTL</td>
</tr>
<tr>
<td>10:10am –</td>
<td>Hands-On: PyMTL Basics with Max/RegIncr</td>
</tr>
<tr>
<td>11:00am –</td>
<td>Coffee Break</td>
</tr>
<tr>
<td>11:30am –</td>
<td>Presentation: Multi-Level Modeling with PyMTL</td>
</tr>
<tr>
<td>11:40am –</td>
<td>Hands-On: FL, CL, RTL Modeling of a GCD Unit</td>
</tr>
</tbody>
</table>
Multi-Level Modeling

Modeling Towards Layout

- Greater Simulation Speed
- Greater Model Detail

Functional Level
- Algorithm and ISA Development
- Cycle Level
- Design Space Exploration
- Register Transfer Level
- Area/Energy/Timing Validation and Prototype Development
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 be used to verify each model!
- Requires good design, the use of latency-insensitive interfaces helps considerably.
def sorter_network( input_list ):
    return sorted( input_list )

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
def sorter_network( input_list ):
    return sorted( input_list )

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