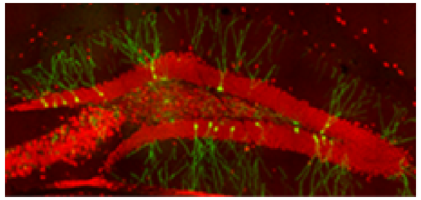
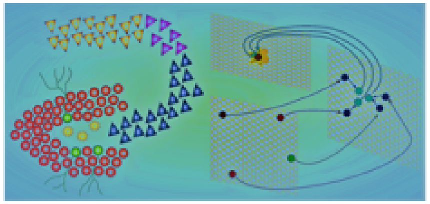
SAND2021-15045C

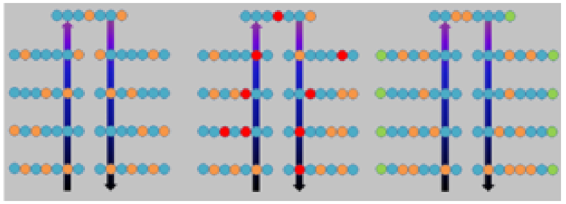
Sandia National Laboratories

Introduction to Fugu and N2A, Thoughts on Lava Integration

Neural Exploration & Research Lab





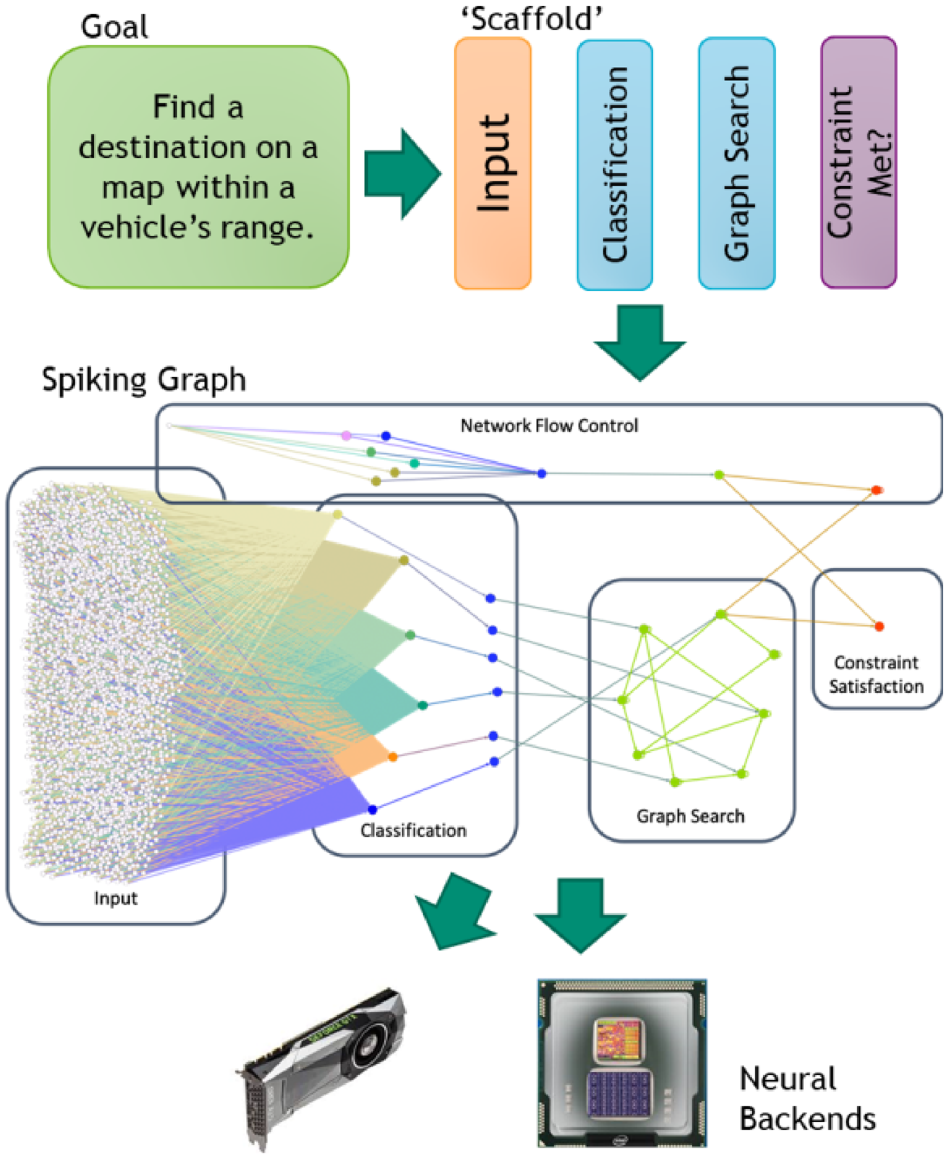


PRESENTED BY

Felix Wang, Fred Rothganger

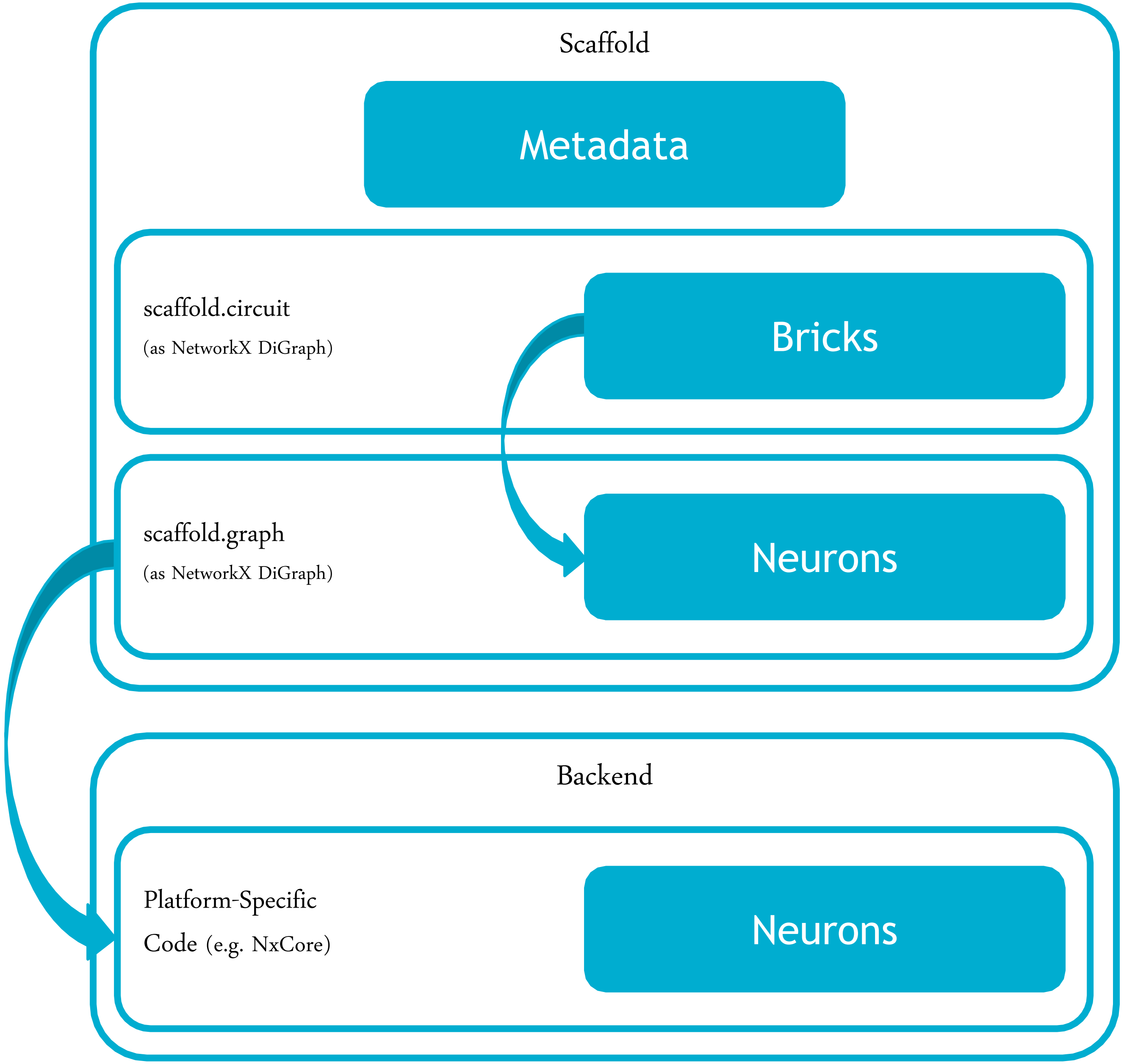
What is Fugu? And Why?

- **Neuromorphic Challenges**
  - Neuromorphic platforms remain a challenge to program
  - Lack of interoperability between research outputs
- **Technical Approach**
  - A unified, (mostly) hardware agnostic, framework to enable neuromorphic algorithm development
  - Focus on linking existing spiking neural networks software to develop application-level computation
  - Minimalist neural computation model (i.e. lowest common denominator) to enable portability to a wide range of hardware platforms
  - Standardized data formats for information transfer between computational kernels
  - Hardware-independent intermediate representations for ease of interoperability (and heterogeneous computing)
- **Results/Impact**
  - Support for multiple software backends completed; several hardware backends in development



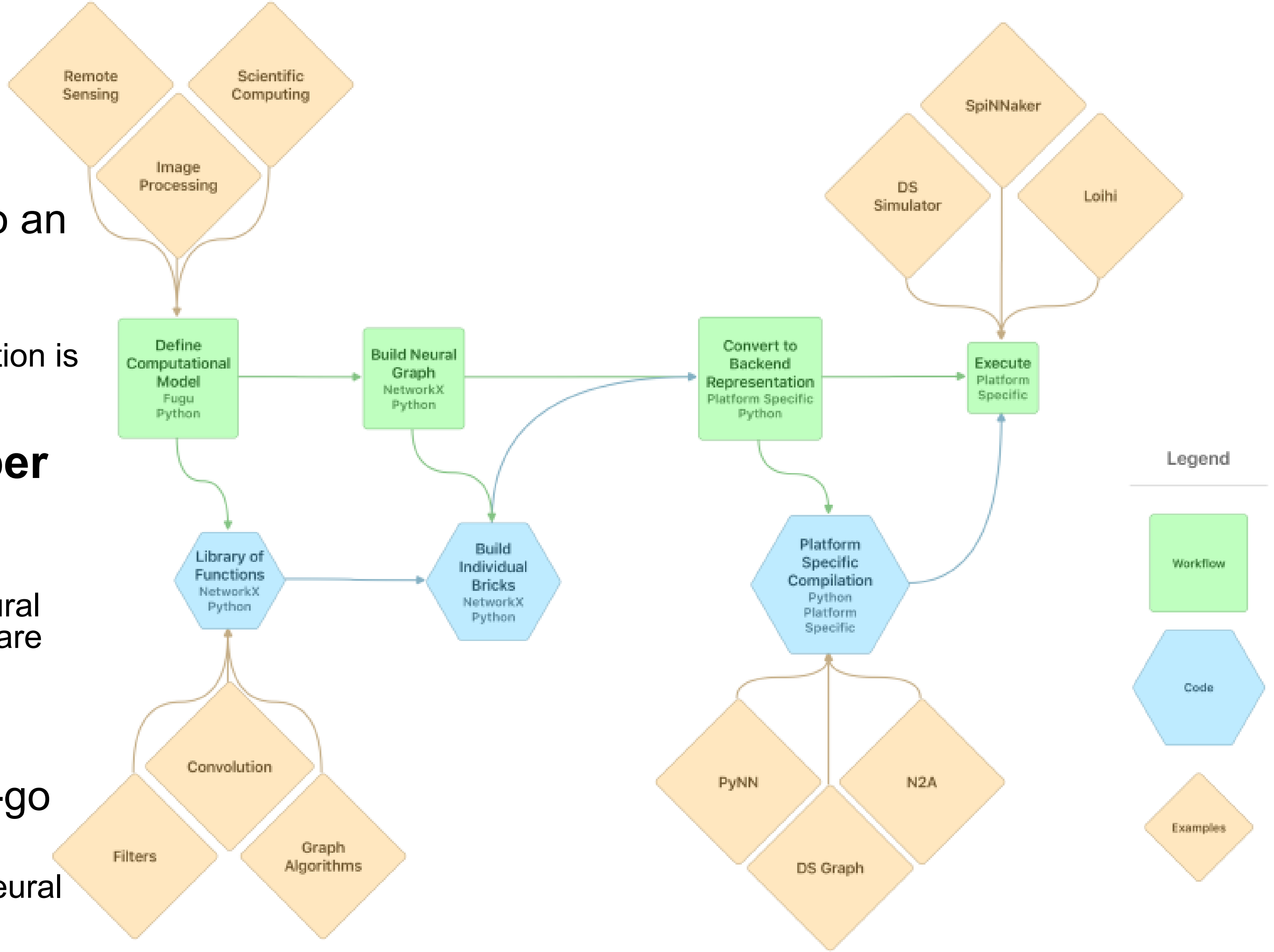
Building Algorithms in Fugu

- **Scaffolds and Bricks**
  - Bricks provide the framework for scaffolds to build the computational graph
    - Uses NetworkX directed graph (DiGraph)
  - Contains metadata
    - Synchronizing with other bricks
    - Data transfer/coding information
    - Neuron and synapse parameters
- **Neural Computation Model**
  - Simple leaky-integrate-and-fire (LIF) model
    - Membrane potential, decay constant, activation threshold, probability of spike
  - Point synapse model
    - Synaptic weight and delay
    - Instant decay (i.e. single-step integration)



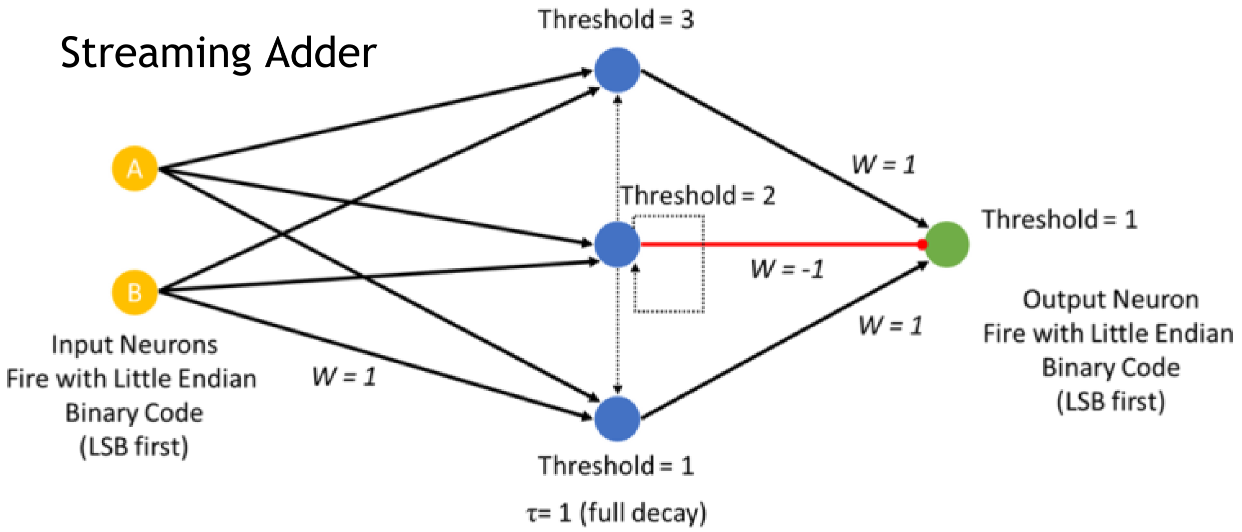
Different user-levels of Fugu

- **For the domain expert (end-user)**
  - Fugu is more about linking existing spiking neural network algorithms into an application
    - Operation is based on function, not structure
    - Underlying hardware and method of computation is as abstracted/hidden as possible
- **For the neural algorithms developer**
  - Fugu offers interoperability with other algorithms and new hardware
    - Standardized data formats and minimalist neural computation model simplifies extending software library
- **For the hardware designer**
  - Fugu helps provide (mostly) ready-to-go algorithms for benchmarking
    - Neural mini-apps implement representative neural computation workflows



Different user-levels of Fugu

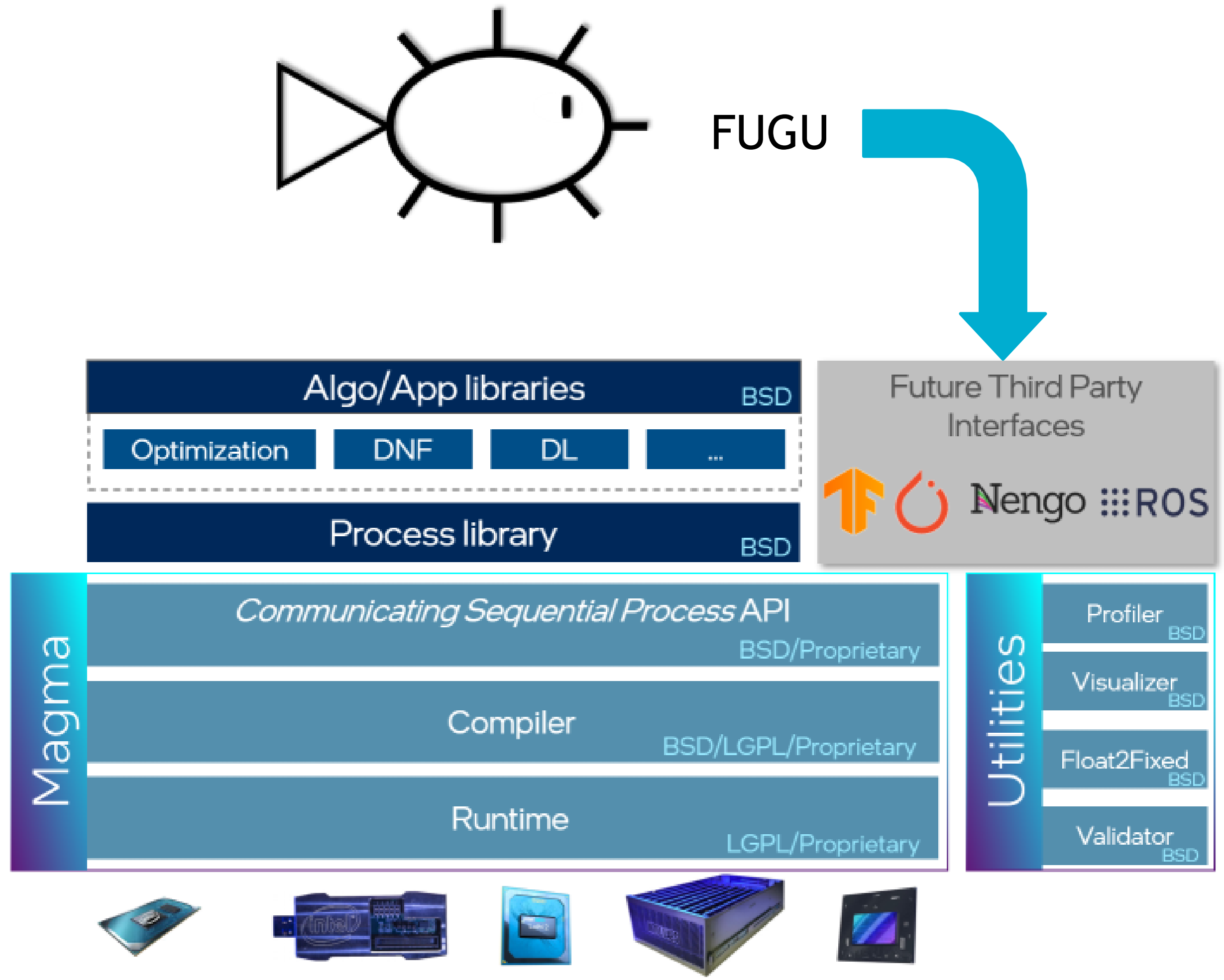
- **For the domain expert (end-user)**
  - Fugu is more about linking existing spiking neural network algorithms into an application
    - Operation is based on function, not structure
    - Underlying hardware and method of computation is as abstracted/hidden as possible
- **For the neural algorithms developer**
  - Fugu offers interoperability with other algorithms and new hardware
    - Standardized data formats and minimalist neural computation model simplifies extending software library
- **For the hardware designer**
  - Fugu helps provide (mostly) ready-to-go algorithms for benchmarking
    - Neural mini-apps implement representative neural computation workflows




```python
def build(self, graph, dimensionality, control_nodes, input_lists, input_codings):
    """
    Build streaming adder brick.

    Arguments:
        + graph - networkx graph to define connections of the computational graph
        + dimensionality - dictionary to define the shapes and parameters of the brick
        + control_nodes - dictionary of lists of auxillary networkx nodes.
           expected keys: 'complete' - A list of neurons that fire when the brick is done
        + input_lists - list of nodes that will contain input
        + input_coding - list of input coding formats

    Returns:
        + graph of a computational elements and connections
        + dictionary of output parameters (shape, coding, layers, depth, etc)
        + dictionary of control nodes ('complete')
        + list of output
        + list of coding formats of output
    """

    if len(input_codings) != 2:
        raise ValueError("adder takes in 2 input on size n")

    output_codings = [input_codings[0]]

    new_complete_node_name = self.name + '_complete'

    graph.add_node(new_complete_node_name,
                   index = -1,
                   threshold = 0.0,
                   decay =0.0,
                   p=1.0,
                   potential=0.0)

    graph.add_edge(control_nodes[0]['complete'], new_complete_node_name, weight=1.0, delay=2)

    complete_node = new_complete_node_name

    l = len(input_lists[0])

    #nodes
    graph.add_node(self.name + 'add', threshold=.9, decay=1.0, p=1.0, potential=0.0)
    graph.add_node(self.name + 'carry0', threshold=1.9, decay=1.0, p=1.0, potential=0.0)
    graph.add_node(self.name + 'carry1', threshold=2.9, decay=1.0, p=1.0, potential=0.0)
    graph.add_node(self.name + 'out', threshold=.9, decay=1.0, p=1.0, potential=0.0)
    #edges
    graph.add_edge(input_lists[0][0], self.name + 'add', weight=1.0, delay=1)
    graph.add_edge(input_lists[1][0], self.name + 'add', weight=1.0, delay=1)
    graph.add_edge(input_lists[0][0], self.name + 'carry0', weight=1.0, delay=1)
    graph.add_edge(input_lists[1][0], self.name + 'carry0', weight=1.0, delay=1)
    graph.add_edge(input_lists[0][0], self.name + 'carry1', weight=1.0, delay=1)
    graph.add_edge(input_lists[1][0], self.name + 'carry1', weight=1.0, delay=1)

    graph.add_edge(self.name + 'carry0', self.name + 'add', weight=1.0, delay=1)
    graph.add_edge(self.name + 'carry0', self.name + 'carry0', weight=1.0, delay=1)
    graph.add_edge(self.name + 'carry0', self.name + 'carry1', weight=1.0, delay=1)

    graph.add_edge(self.name + 'add', self.name + 'out', weight=1.0, delay=1)
    graph.add_edge(self.name + 'carry0', self.name + 'out', weight=-1.0, delay=1)
    graph.add_edge(self.name + 'carry1', self.name + 'out', weight=1.0, delay=1)

    self.is_built=True

    output_lists = [[self.name + 'out']]

    return (graph, self.dimensionality, [{'complete': complete_node}], output_lists, output_codings)
```

Thoughts on Fugu to Lava

- **The vision of Lava is similar,**
  - Neural computation is encapsulated into composable functions (e.g. Scaffolds/Bricks vs. Processes/SubProcesses)
    - This should make mapping Fugu to Lava relatively straightforward (e.g. wrapping Bricks as Lava Processes)
  - Support for multiple software/hardware platform implementations (e.g. Backend vs. ProcessModels)
- **But there are a few differences**
  - Fugu is more restrictive in its computational model (e.g. minimalist vs. maximalist)
    - Simpler but more specific algorithmic paradigm (e.g. use of precise spike timing, stochasticity)
  - Fugu code is more similar to laying out circuits in an electrical circuit simulator such as Xyce, than the asynchronous message-passing (CSP) in Lava

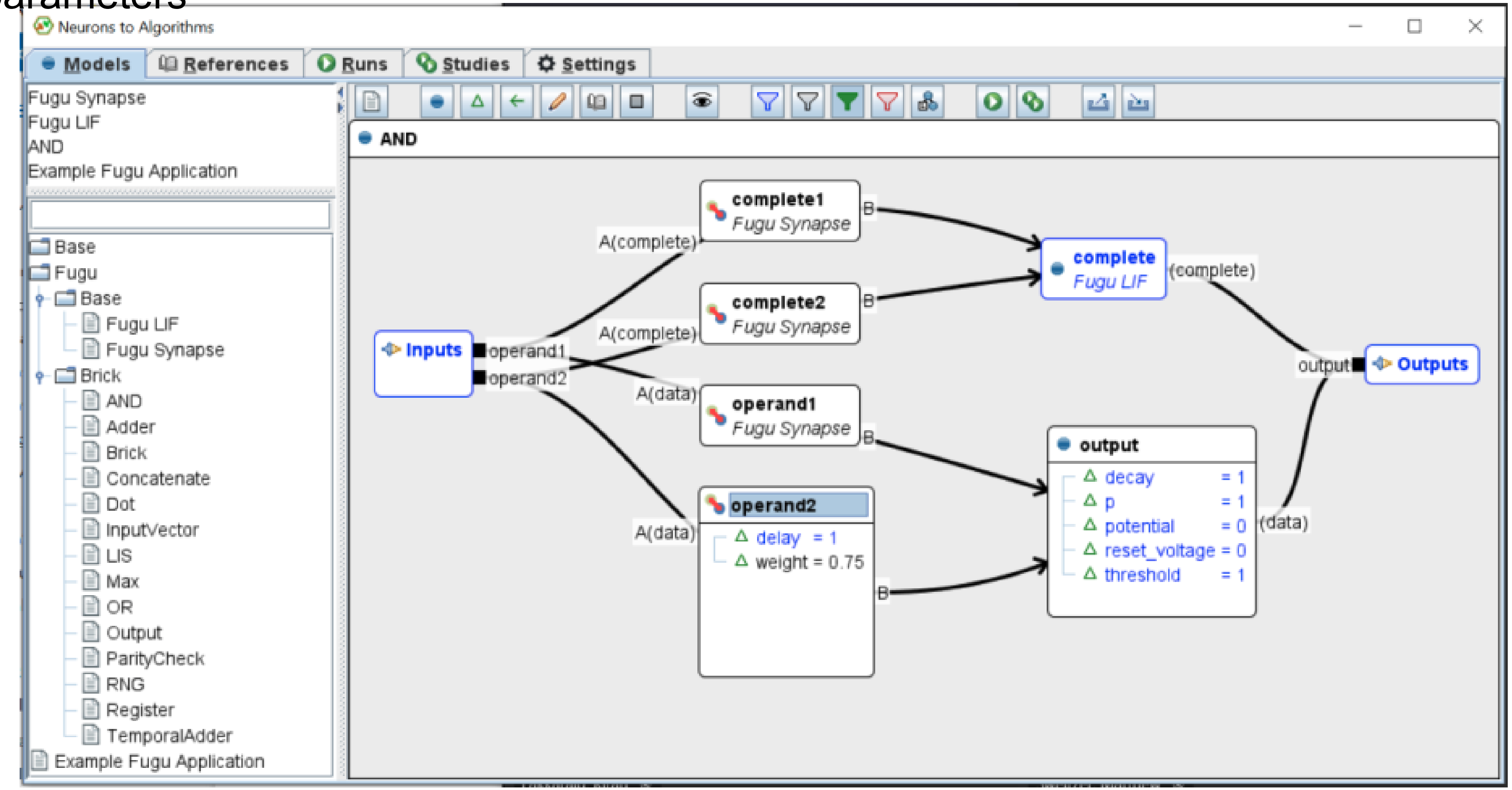


TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc. | PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc. | The "nine dots" ROS logo is a trademark of Open Robotics.

Lava software stack image taken from https://lava-nc.org/lava_architecture_overview.html

Fugu through N2A

- **N2A Brick Builder (live demo)**
  - N2A is a “language for modeling neural systems” (https://github.com/frothga/n2a)
  - Graphical tool for drag-and-drop to add parts, linking; property sheets to set parameters



N2A/Fugu example connecting LIF neurons and synapses to the inputs and outputs of a brick