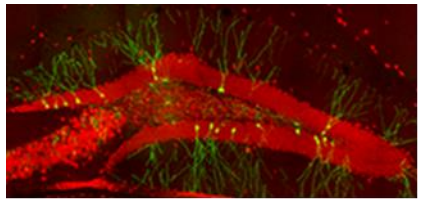
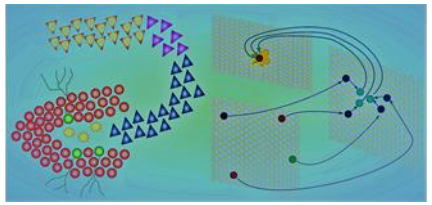
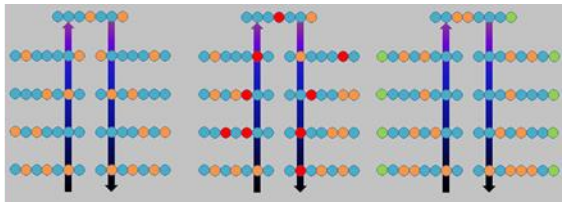
Neural Mini-Apps as a Tool for Neuromorphic Computing Insight

PRESENTED BY

Craig M. Vineyard, PhD (cmviney@sandia.gov)

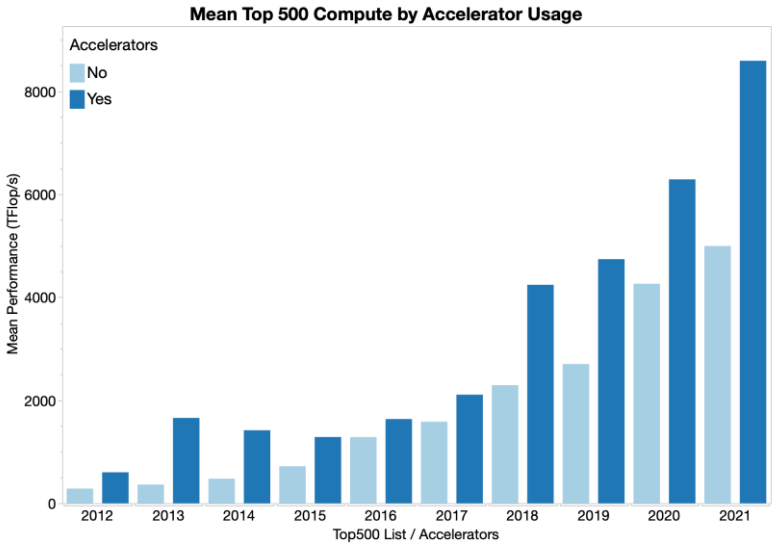
Team: Brad Aimone, Suma Cardwell, Frances Chance, Srideep Musuvathy, Fred Rothganger, William Severa, Darby Smith, Corinne Teeter, Craig Vineyard, Felix Wang

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

All Benchmarks Are Wrong – Some Are Useful



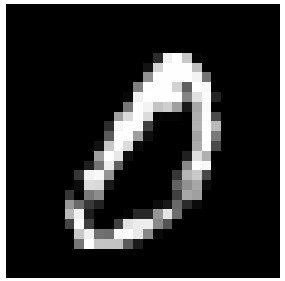
Conventional benchmarks

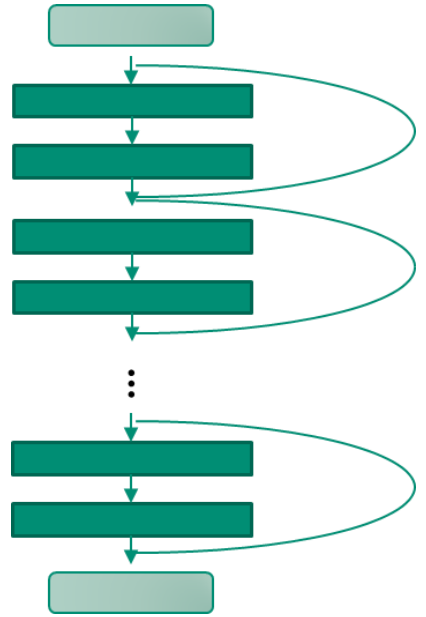
- Defined by the processing to be executed

AI/ML benchmarks

- Sometimes defined by a particular model, other times a task or dataset
- Offers flexibility & mirrors ML
- Makes direct comparisons difficult

And yet that's roughly what benchmarks are for – articulating information about a computational approach & enabling comparisons

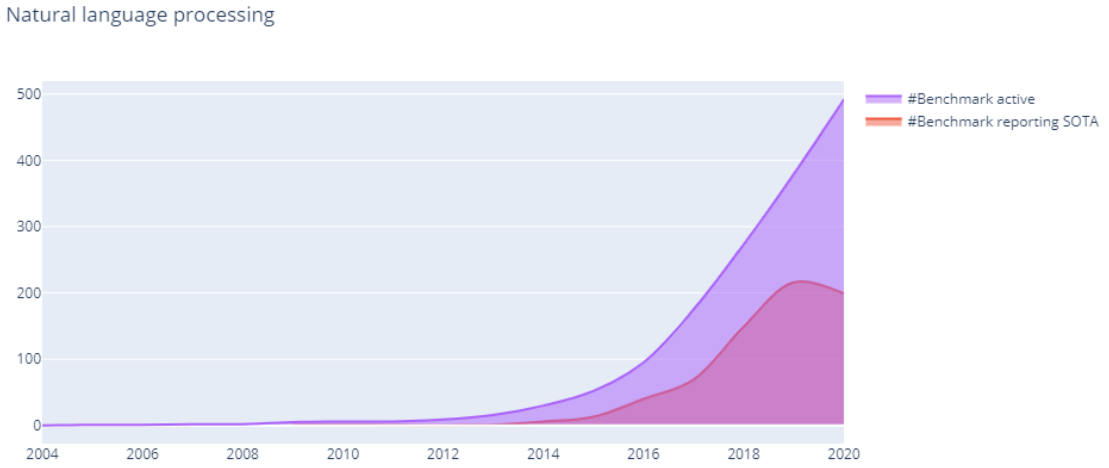


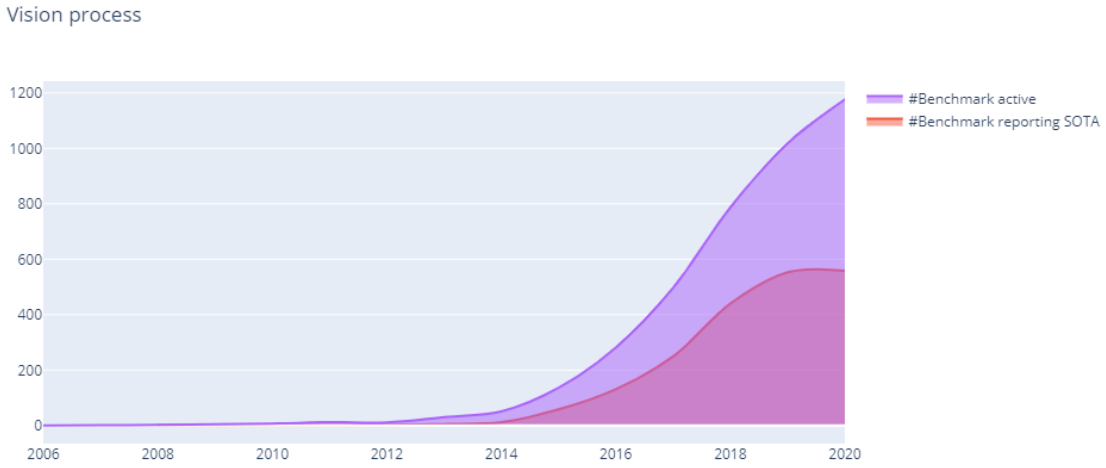


All Benchmarks Are Wrong – Some Are Useful

However, many of these efforts fall flat

- Barbosa-Silva et al. surveyed 1688 benchmarks across AI tasks
- Of these only 1111 are benchmarks with results at three or more times





Barbosa-Silva, Adriano, et al. "Mapping global dynamics of benchmark creation and saturation in artificial intelligence." *arXiv preprint arXiv:2203.04592* (2022).

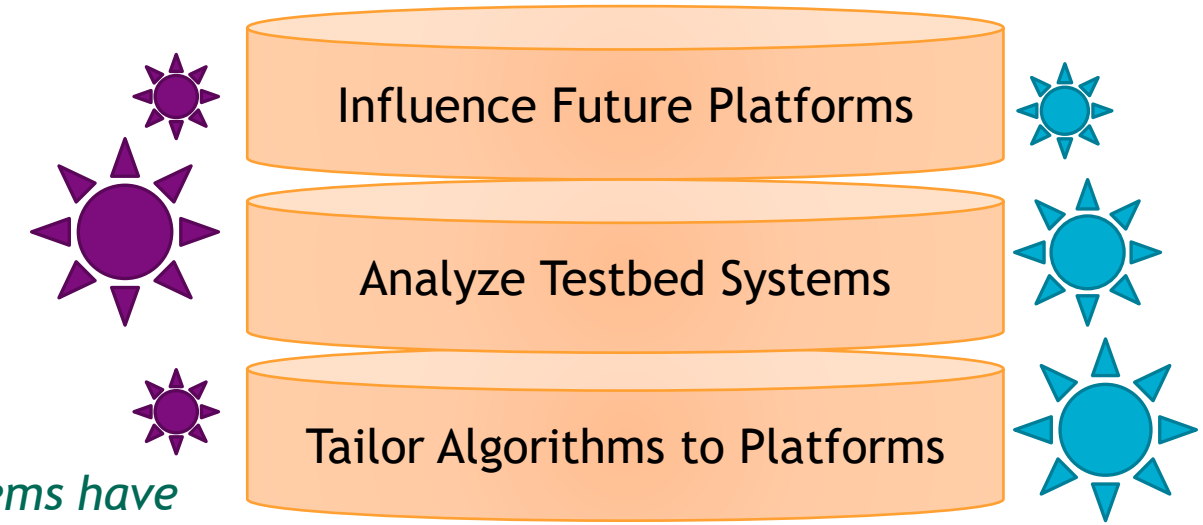
➢ How can we meaningfully bring insight & understanding to neuromorphic computing???

Why Mini-Apps for Neuromorphic?

From Heroux et al., 2009: "there is a middle ground for small, self-contained programs that, like benchmarks, contain the performance-intensive computations of a large-scale application, but are large enough to also contain the context of those computations."

These Mini-Apps would enable:

- Interaction with external research communities
- Simulators
- Early node architecture studies
- Network scaling studies
- New language and programming models
- Compiler tuning

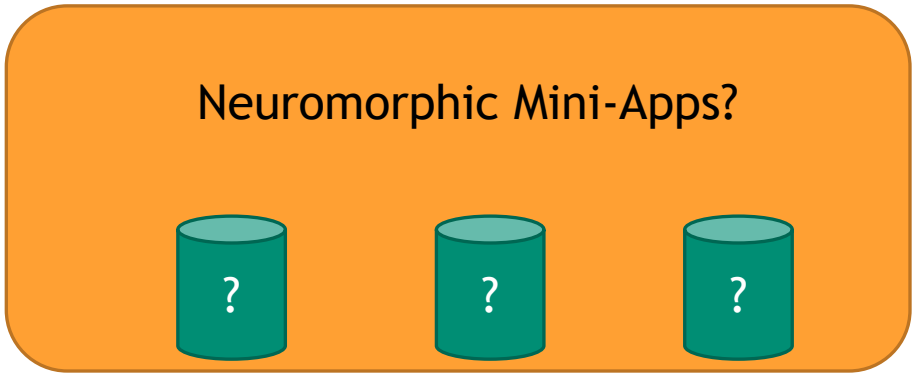


Conventional systems have less uncertainty at algorithm and programming level

NMC systems have considerable uncertainty at algorithm level

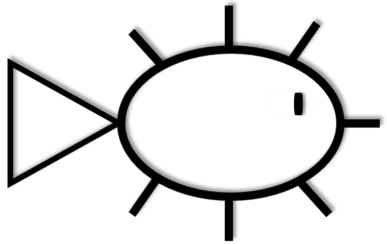
Conventional and Neuromorphic Mini-Apps

Mini-Apps

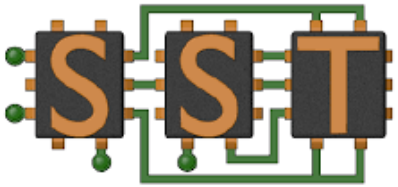


kokkos RAJA

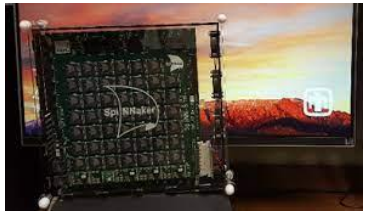
Platform-Agnostic Programming Layer



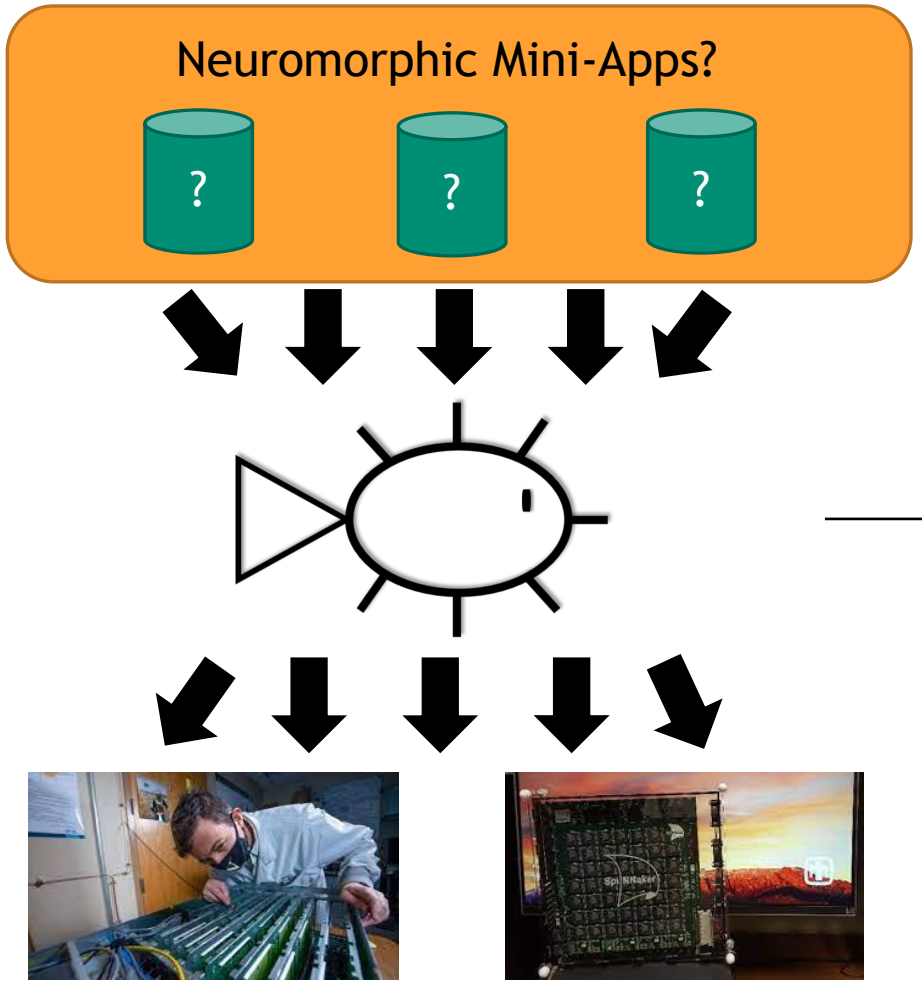
FUGU



Back-Ends



Fugu addresses two key challenges of neuromorphic programming



Composability

Deploying applications on neuromorphic hardware requires implementing algorithms within neural circuits

- Need to be able to build applications from well designed kernels
- Need to take advantage of features offered by spiking neuron model

Portability

Programming neuromorphic platforms requires a *graph* of neurons (nodes) and synapses (edges)

- Need to represent neural algorithms in common graph format
- Need ability to translate graph into backend specific constraints

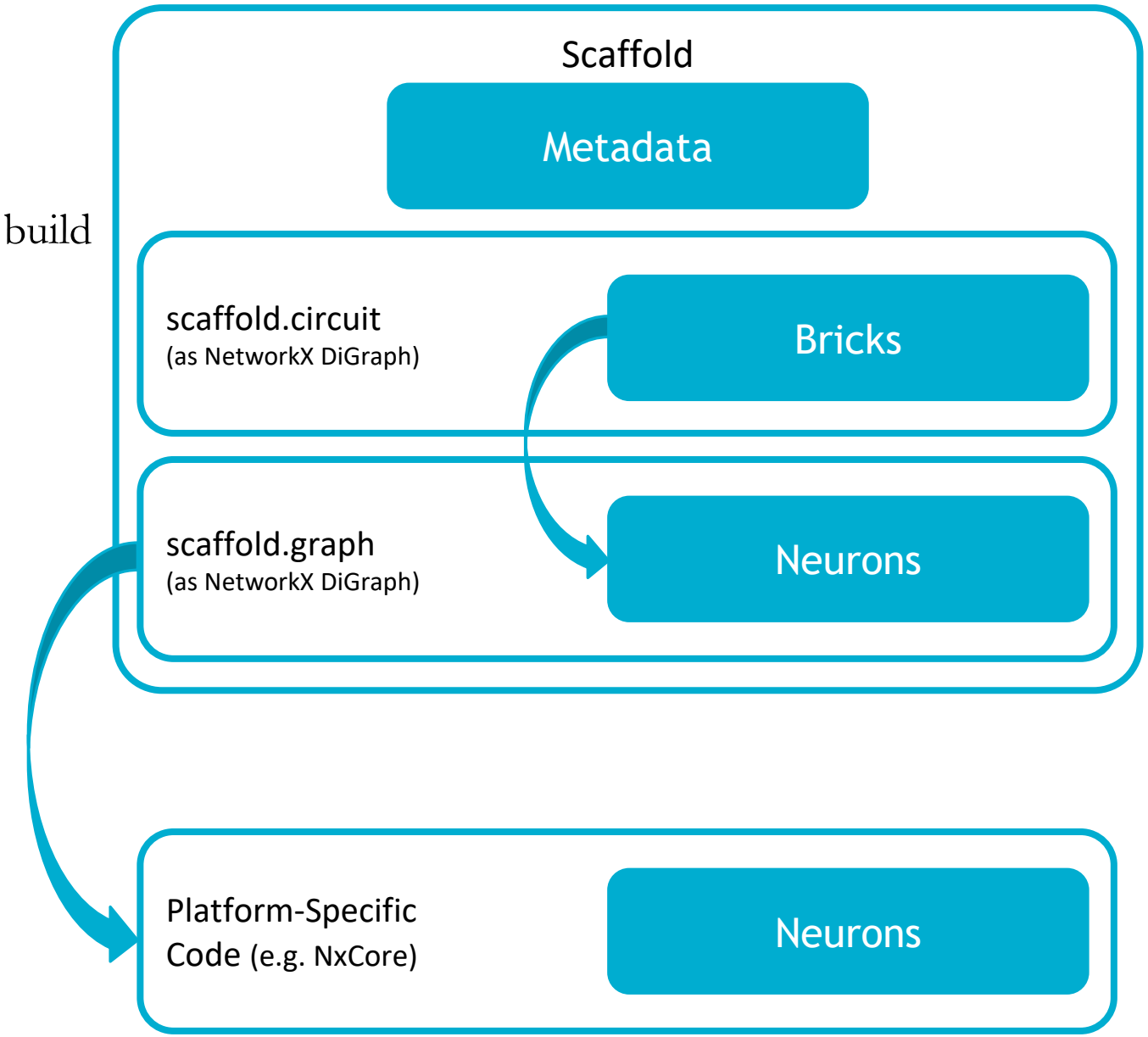
Building Algorithms in Fugu

Scaffolds and Bricks

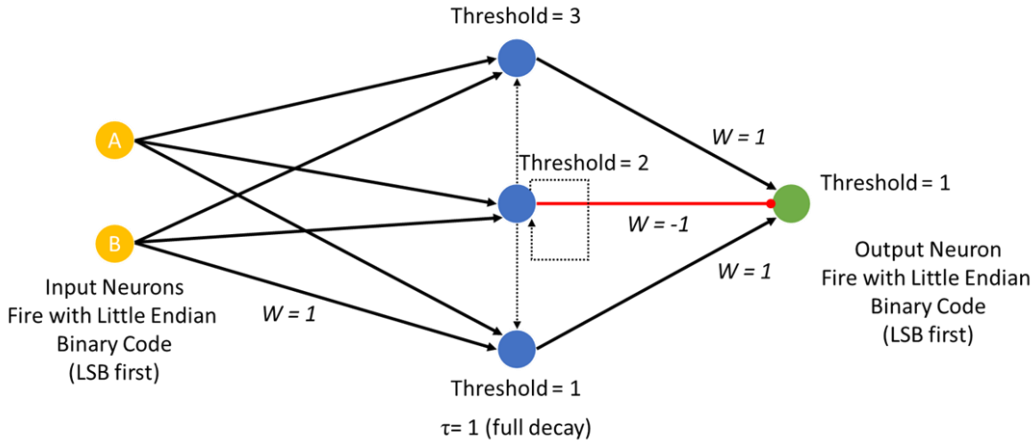
- Bricks provide the framework for scaffolds to build the computational graph
  - Uses NetworkX directed graph (DiGraph)
- Contains metadata
  - Synchronizing with other bricks
  - Data transfer/coding information
  - Neuron and synapse parameters

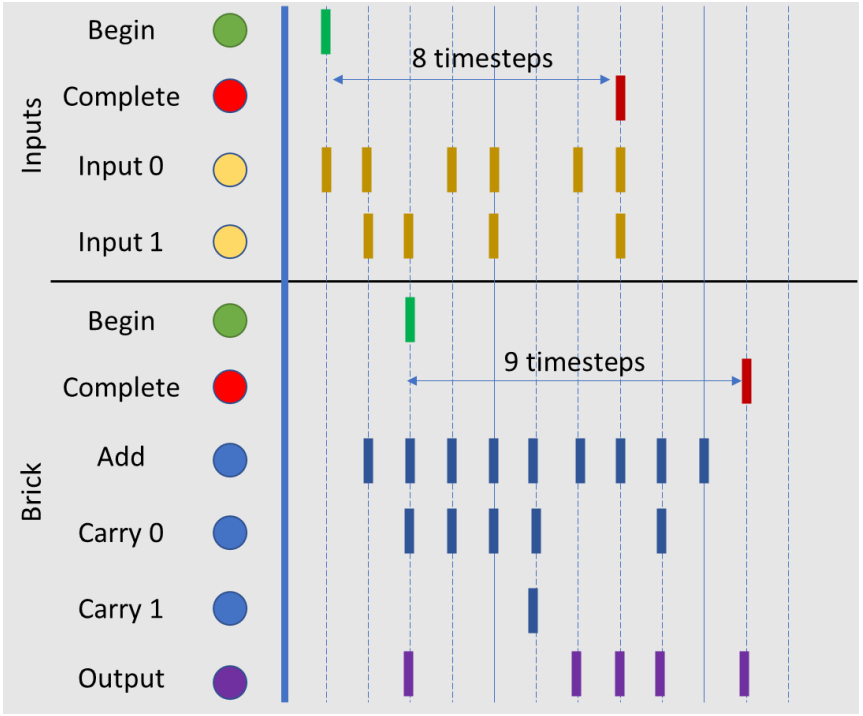
Neural Computation Model

- 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)



Fugu – Binary streaming adder



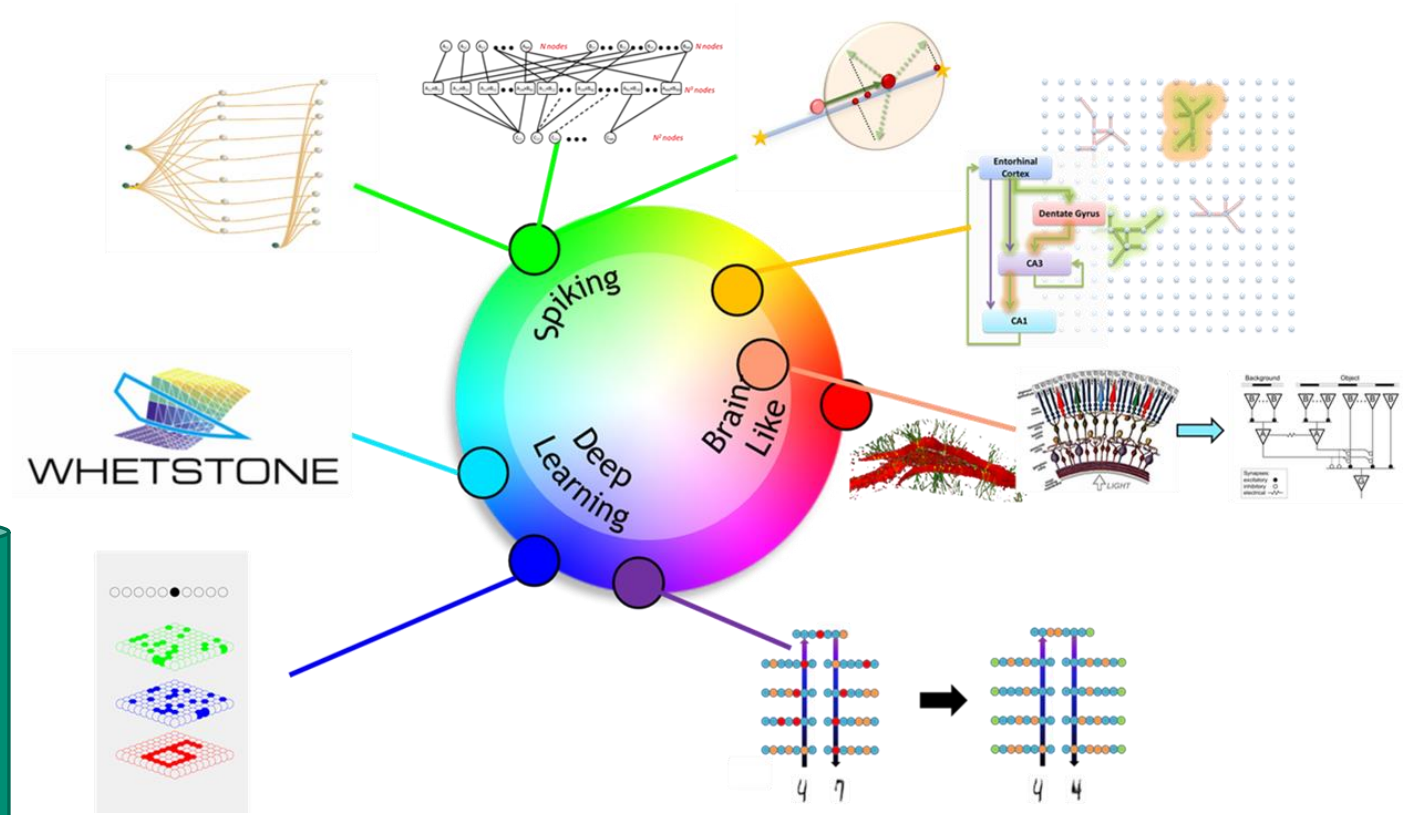



```
# Simple addition example

scaffold = Scaffold()

# For addition, what we want to do is create a Scaffold taking two inputs and the adder brick

scaffold.add_brick(Vector_Input(np.array([[1, 1, 0, 1, 1, 0, 1, 1]]), coding='binary-L', name='Input0', time_dimens
scaffold.add_brick(Vector_Input(np.array([[0, 1, 1, 0, 1, 0, 0, 1]]), coding='binary-L', name='Input1', time_dimens
scaffold.add_brick(streaming_adder(name='adder1_'), [(0,0), (1, 0)], output=True)

scaffold.lay_bricks()
scaffold.summary(verbose=1)

backend = snn_Backend()
backend_args = {}
backend_args['record'] = 'all'
backend.compile(scaffold, backend_args)
result = backend.run(30)
print(result)
```

Initial Neural Mini-Apps

Neuromorphic is *potentially* good for many different classes of computing applications

Scientific Computing
- *Similar to existing Mini-Apps*
- *Well-defined tasks*



Today's Machine Learning
- *Emphasis of much neuromorphic research*
- *High bar with GPUs*

Tomorrow's Artificial Intelligence (computing's future)
- *Area of biggest potential impact*
- *Poorly-defined algorithms*

Different Mini-Apps leverage different features of spiking neurons

Mini-App / Kernel	Unbounded Fan-In (Fan-in > 2)	Threshold (e.g., λ>1)	Synapse Delays	Decay (e.g., τ>0)	Random Noise	Online Learning
Streaming Arithmetic	★	★	★			
MA#1 Random Walk	★	★	★	★	★	
MA#2 Sparse Coding	★	★		★		
MA#3 Shortest Path	★		★	★		

Neural Mini-App Structure

Single-line Python interface

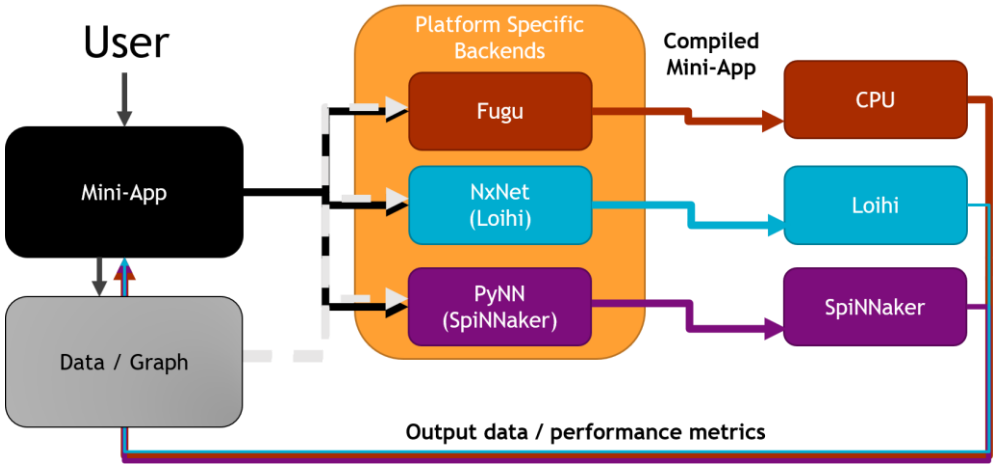
- python fluence_mini_app.py --run_mode loihi --neural_timesteps 10000 -v 100 -dt .02 -ss .05 -da .2 -M 200

Can run multiple backends from same function

- Currently have worked with Fugu, Loihi, SpiNNaker

Flags to set Mini-App specific parameters

- Scaling parameters (e.g., # neuromorphic timesteps, # of walkers)
- Implementation parameters (e.g., angle precision, time precision)
- Physics parameters (e.g., particle velocity, scattering probabilities)



Neuromorphic Random Walk

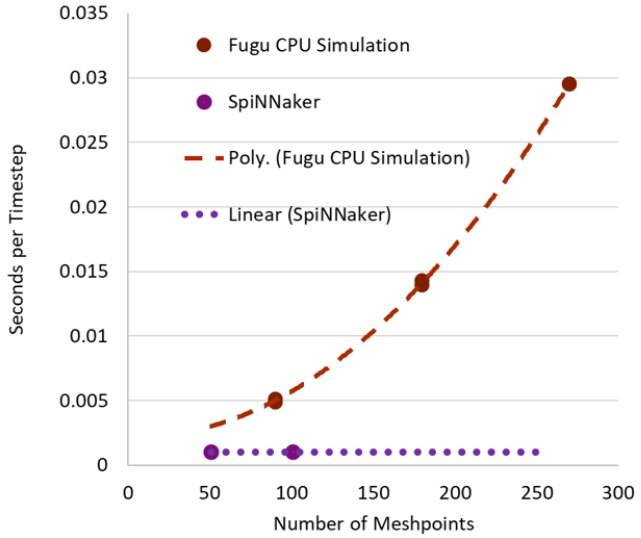
Discrete time Markov Chain (DTMC)

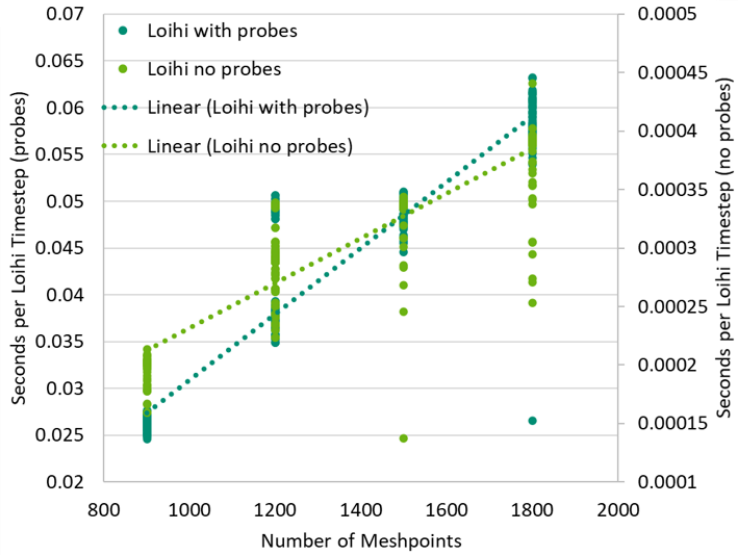
- Particle Angular Fluence: the time-integrated flux of particles traveling through media given as a function of position and velocity
- Particles travel at a constant speed and experience relative velocity scattering over a small region of space
- Conventional approach models walkers & tracks states – neuromorphic models state & tracks walkers

Smith, J. Darby, et al. "Neuromorphic scaling advantages for energy-efficient random walk computations." Nature Electronics (2022): 1-11.

Parameterization	Number of total walkers, Size of direction/relative velocity/angular discretization, Time step size of simulation, Size of the state space, Size of positional discretization
Scaling	Walkers, Mesh size
Metrics	Energy cost of walkers, Time to run, Space to run

Example Results -





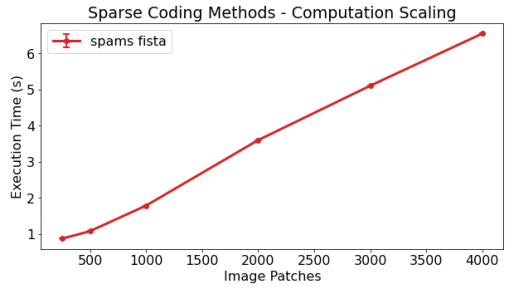
Neural Sparse Coding

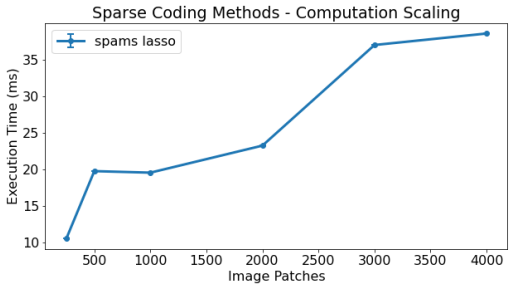
Sparse Coding or Sparse Dictionary Learning

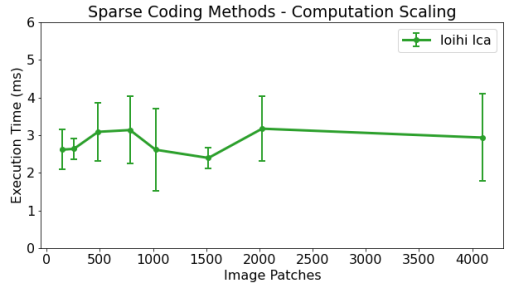
- Method of modeling data by decomposing it into sparse linear combinations of elements of a given overcomplete basis set
- On neuromorphic, the LASSO (least absolute shrinkage and selection operator) computation for sparse coding can be approximated with the spike-based algorithm LCA (locally competitive algorithm)
  - Implemented as rate-coded neurons with inhibitory connections between competing dictionary elements

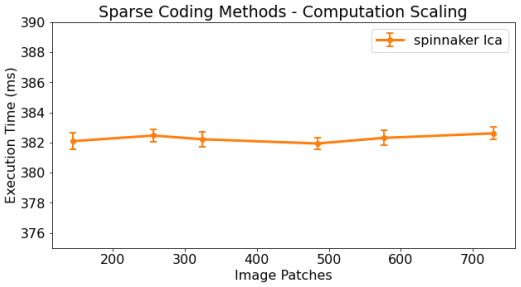
Parameterization	Size of image, Size of image patch, Size of the dictionary, Stride of image patch, Desired sparsity
Scaling	Problem size via # of image patches, Parameters
Metrics	Time for setup, Time for reconstruction, Reconstruction performance, Reconstruction sparsity, Compute resource usage, Energy resource usage

Example Results -









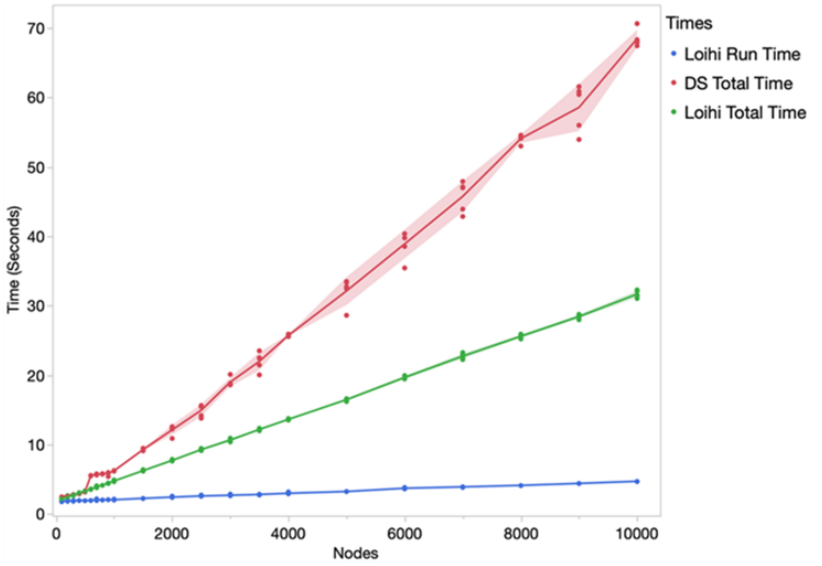
Neural Graph Analysis

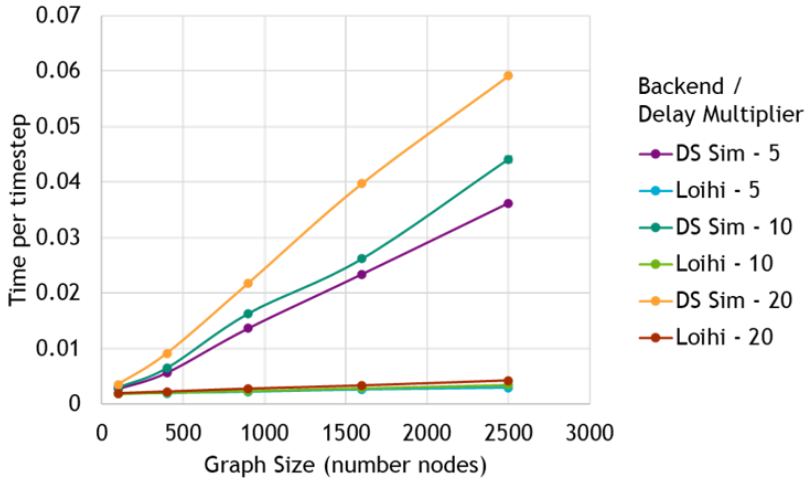
Single Source Shortest Path (SSSP)

- Between a source and target node, what is the shortest path (and path length) that connects the two
- SNN is straightforward – each vertex in the source graph is a neuron, each edge is a synapse between neurons, & graph weights equate to delays
  - The source neuron receives input driving it to spike send ensuing spikes through the SNN
  - Shortest path length is determined when the target spikes & monitoring edges can yield the path

Parameterization	Graph generation (uniformly random tree, small world), Nodes, Weight range, Max runtime, Source, Target
Scaling	Graph scale, Weight/delay range
Metrics	Total time, Time for setup

Example Results -



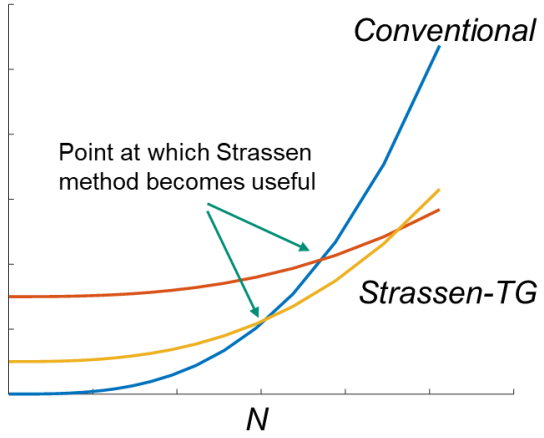


Why Mini-Apps for Neuromorphic?

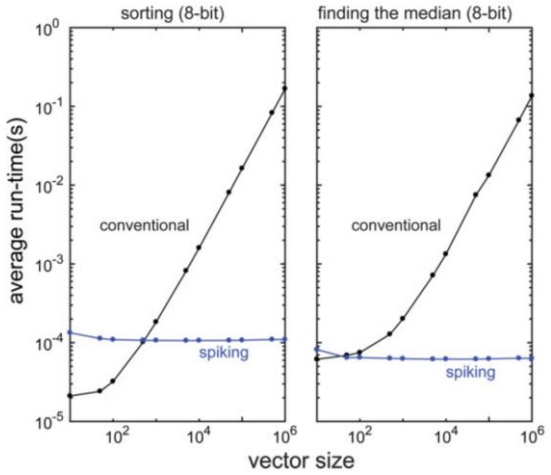
From Heroux et al., 2009: "there is a middle ground for small, self-contained programs that, like benchmarks, contain the performance-intensive computations of a large-scale application, but are large enough to also contain the context of those computations."

Fugu design principles such as compositionality allow us to explore not only core computations, but also application context

- E.g. Random Walk mini-app illustrates this as rigorous physics simulation details are included
- Prior work has shown a neuromorphic advantage may require considering problem setup and scaling as well as the computation



Parekh, Ojas, et al. "Constant-depth and subcubic-size threshold circuits for matrix multiplication." *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*. 2018.



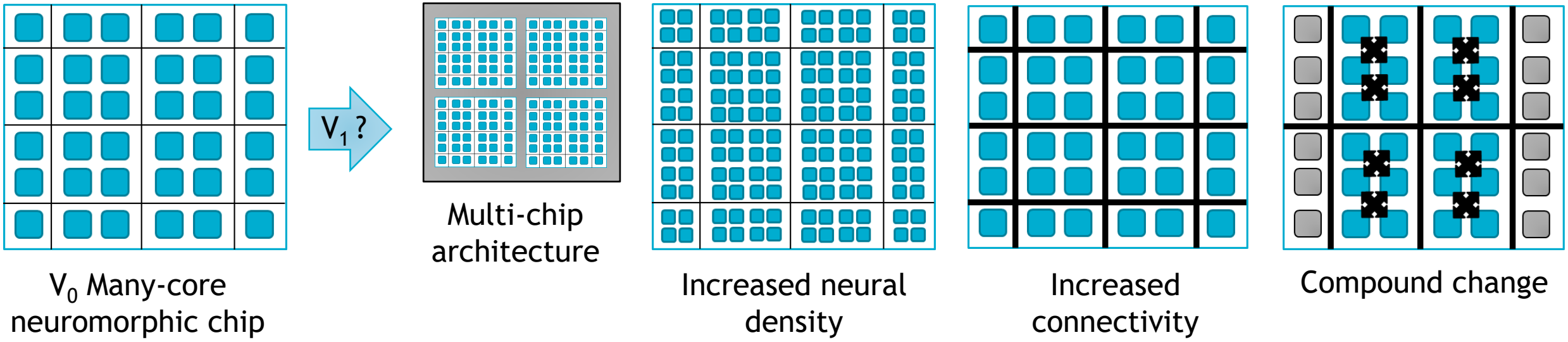
Verzi, Stephen J., et al. "Computing with spikes: The advantage of fine-grained timing." *Neural computation* 30.10 (2018): 2660-2690.

Why Mini-Apps for Neuromorphic?

Benchmarking is data from an architecture – Mini-apps yield data about an architecture

- While the former is a subset of latter, they offer different insight
- Importantly, mini-apps enable means of understanding novel architectures like neuromorphic

For example – next generation architectures may progress in several ways:



Neuromorphic Mini-Apps offer a means of quantitatively investigating implications of emerging architectures

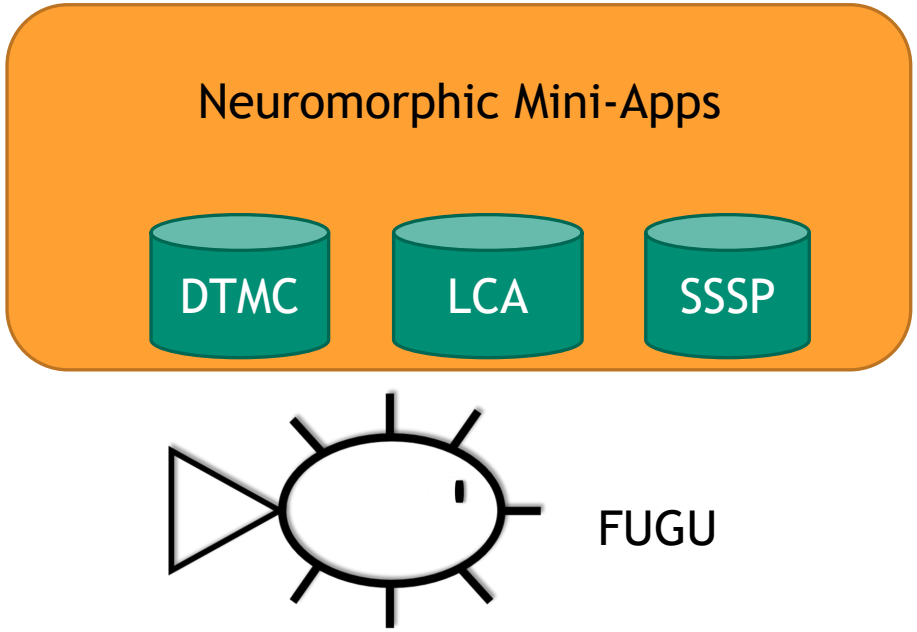
Conclusions

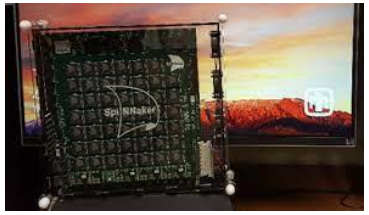
Three Mini-Apps introduced here

- Diversity of applications
  - Scientific computing
  - Machine learning
  - Data / graph analytics
- Stress architectures in different manners
  - Sparsity & density of activations and connectivity
  - Deterministic vs non-deterministic
  - Different workflows
- Pursue NMC advantage beyond acceleration of minimal computational kernels
- Positions us to understand and influence emerging architectures

Fugu as an intermediate representation

- Facilitates programming and compositionality
- Provides stable framework to optimize algorithms despite evolving hardware





Looking forward – continue to expand and integrate!

Neuromorphic Mini-Apps

DTMC | LCA | SSSP | ?? | ??

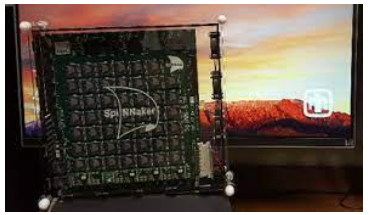
MANTEVO

Conventional Mini-Apps

FUGU

kokkos

RAJA

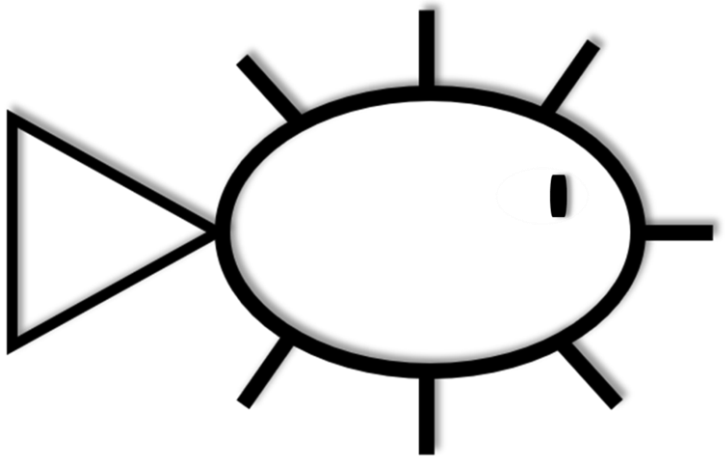


??

??

SST

Thank you!



https://github.com/SNL-NERL/Fugu