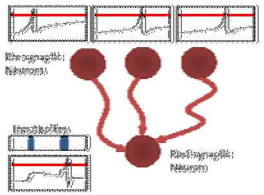
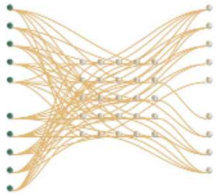
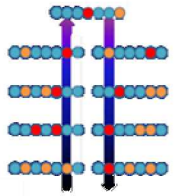
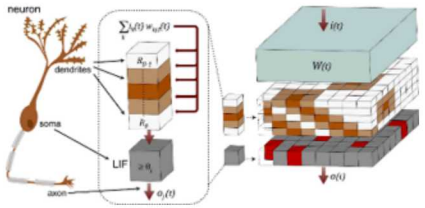
SAND2018-9248C

Neuromorphic Hardware Implementation of Spiking Algorithms for Markov Random Walks









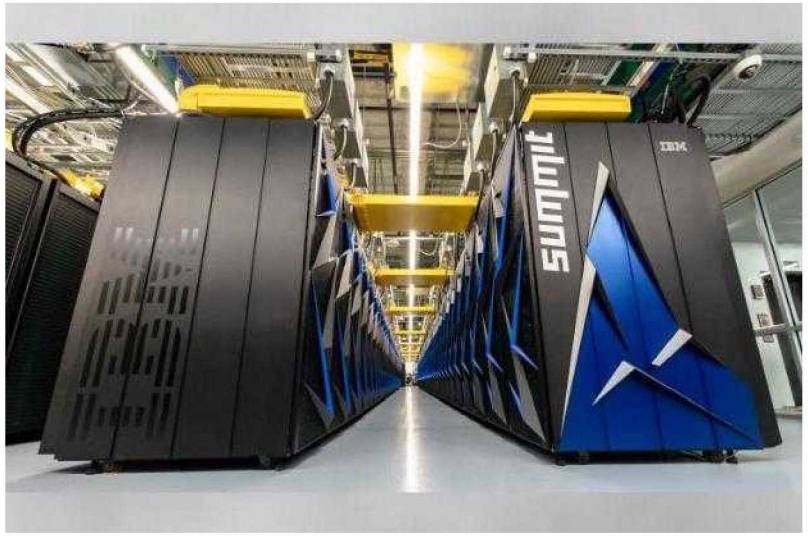
PRESENTED BY

Brad Aimone

Aaron Hill, Rich Lehoucq, Ojas Parekh, and William Severa

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.

If I had a supercomputer of neuromorphic chips, what could I do?

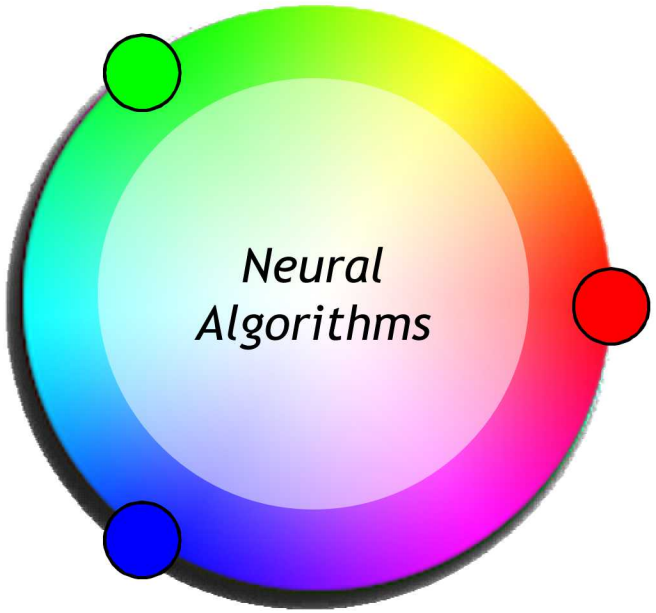


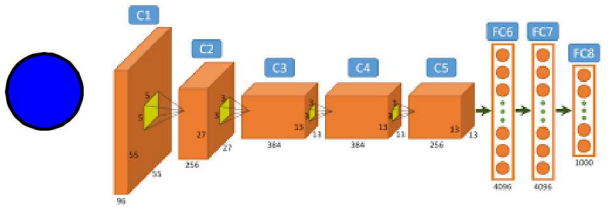
Oak Ridge Summit (June 2018)
202752 CPU Cores, 27648 GPU Cores
~100 PFLOPS
15MW Power

SpiNNaker 106 Machine (TBD)
1036800 ARM Cores
~1B Neurons
100 kW Power

Can we simulate any applications that require systems like Summit on low-power systems like SpiNNaker?

The broad palette of neural computing



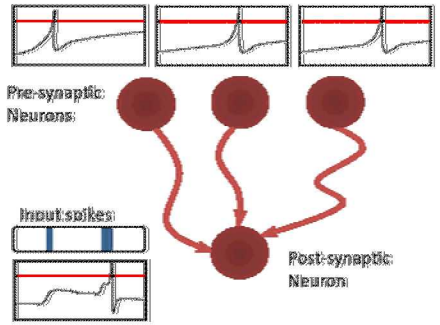


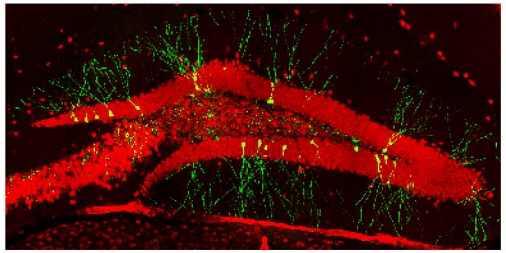
Artificial neural networks

- Generic layers of non-linear nodes
- SGD optimization of weights
- Powerful machine learning capabilities through learning sequential non-linear mappings and function approximation

Spiking neural algorithms

- Hand-crafted circuits of spiking neurons
- Model of parallel computation
- Energy efficiency through event-driven communication and high fan-in logic

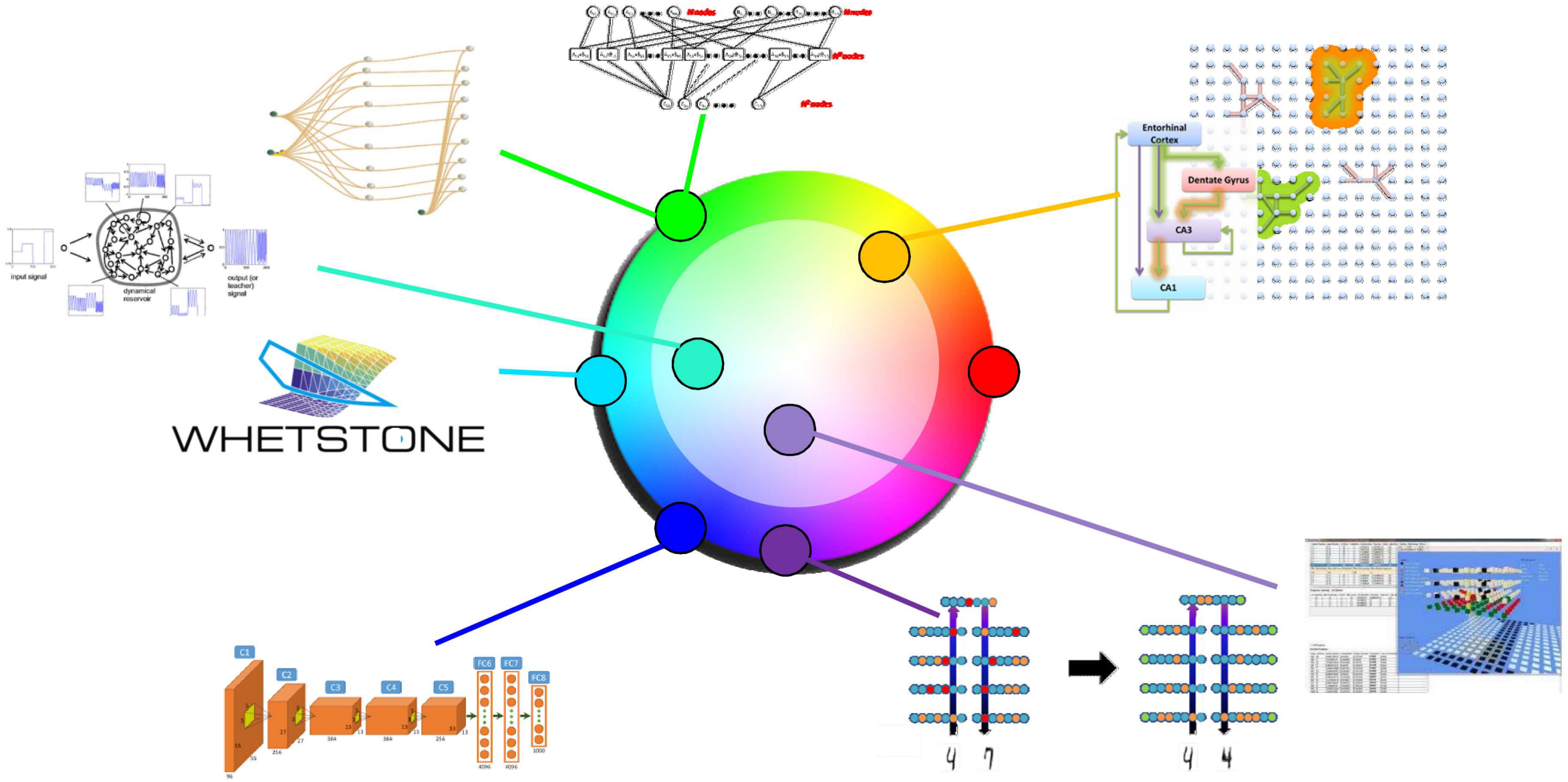




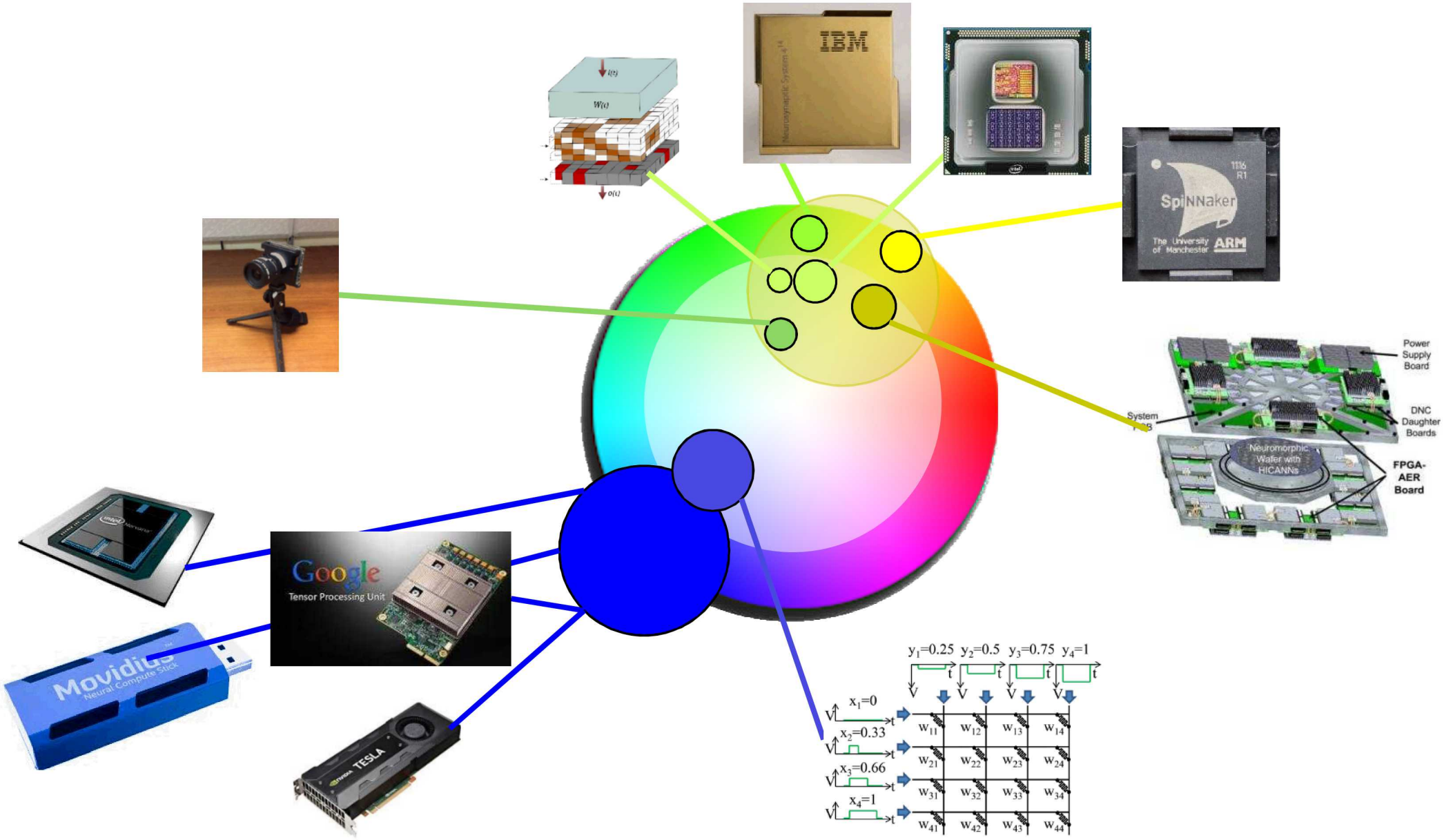
Neuroscience-constrained algorithms

- Circuit architecture based on local and regional neural connectivity
- Computation incorporates broad range of neural plasticity and dynamics
- *Generally still unexplored from algorithms perspective*

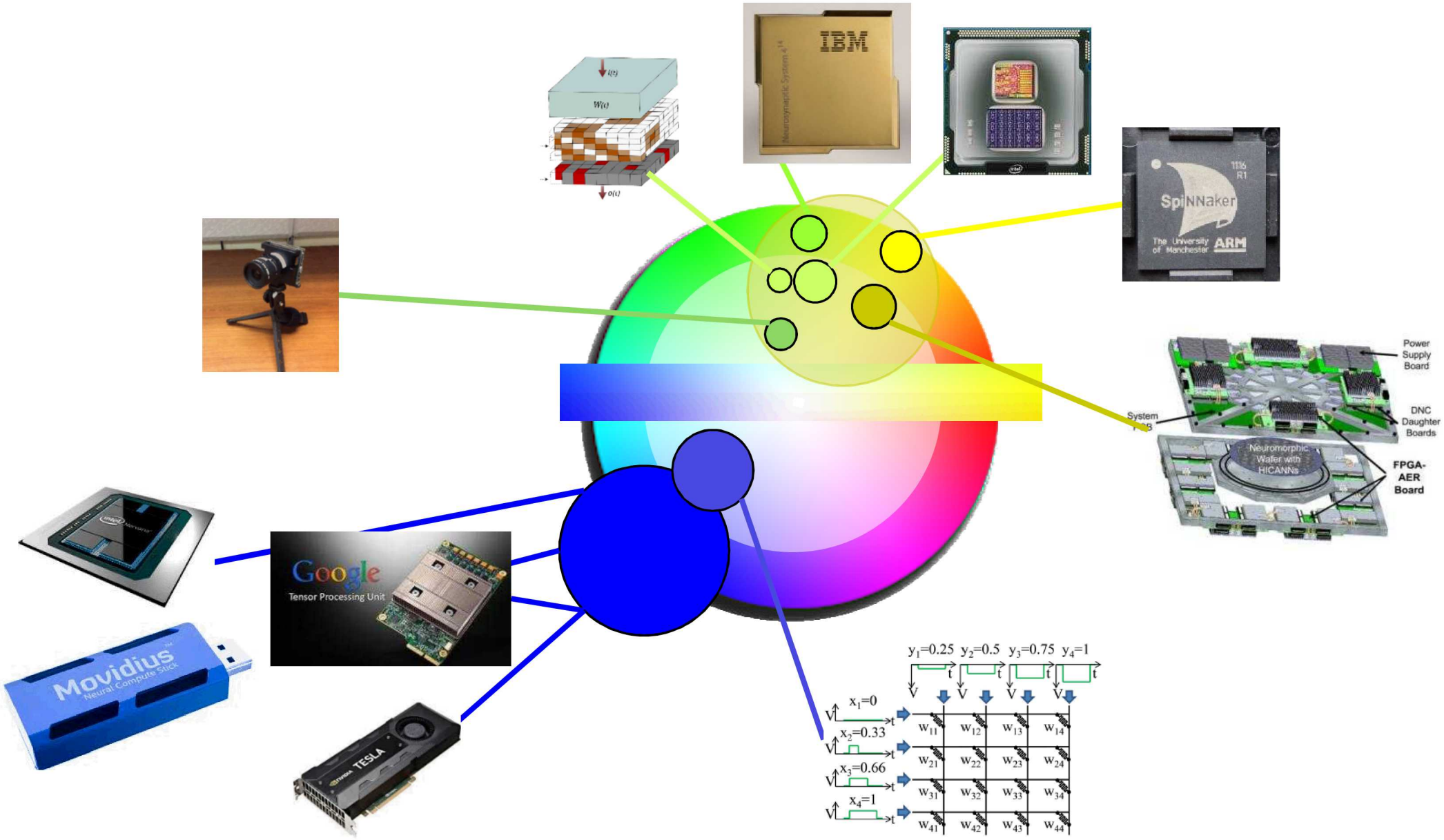
Most neural algorithms efforts have focused on either spiking or ANNs



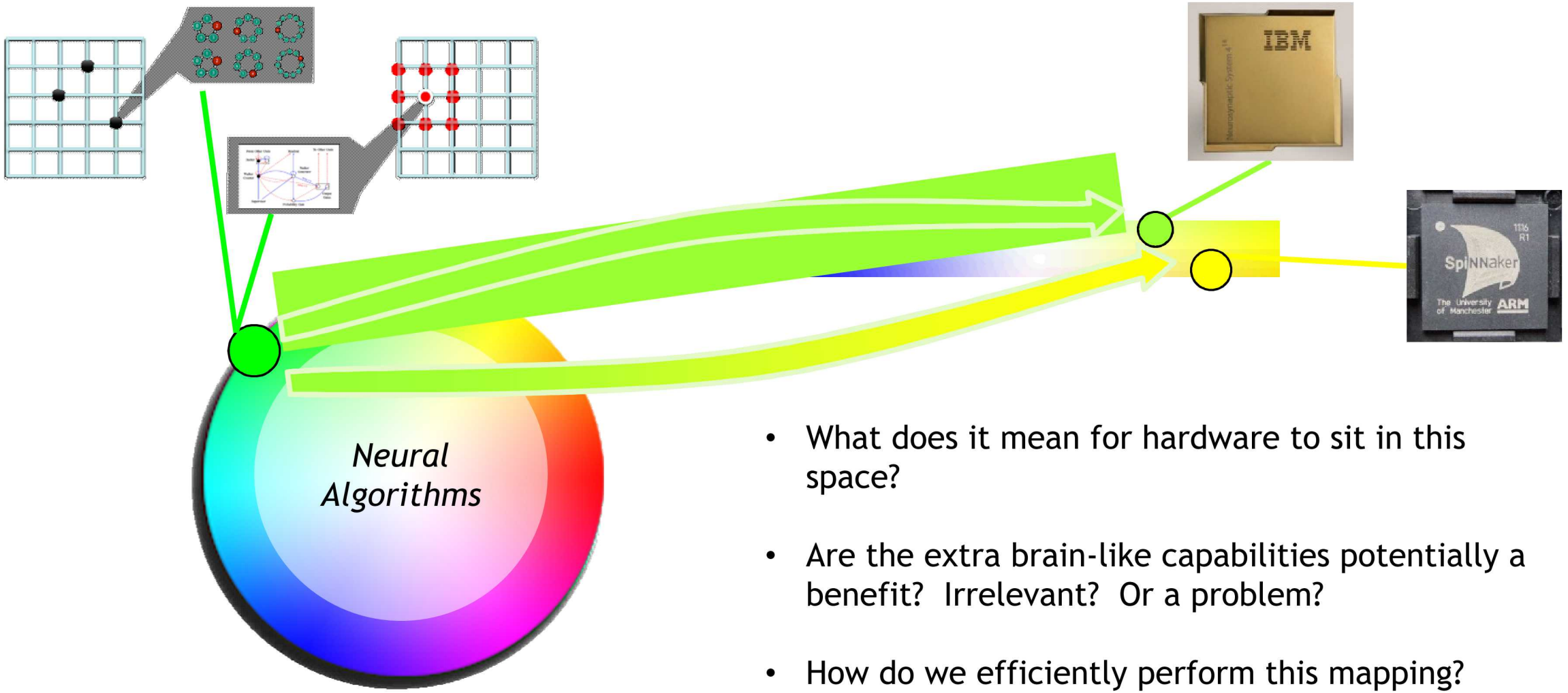
Hardware efforts have fallen along an axis between loosely biological spiking architectures and neural network accelerators



Hardware efforts have fallen along an axis between loosely biological spiking architectures and neural network accelerators



How efficiently will two spiking neural algorithms for simulating random walks map to spiking neural hardware?



- What does it mean for hardware to sit in this space?
- Are the extra brain-like capabilities potentially a benefit? Irrelevant? Or a problem?
- How do we efficiently perform this mapping?

Spiking random walk algorithms – diffusion is a pure scientific computing task for spiking algorithms

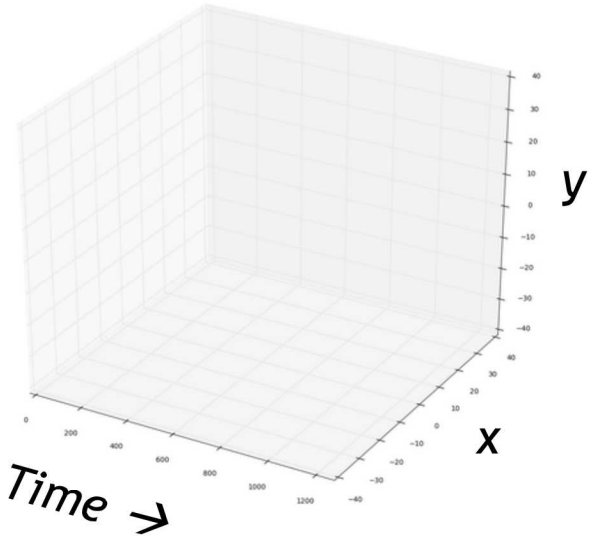
Diffusion can be modeled either as a deterministic PDE or a stochastic process

- For an initial distribution of particles, P_0, what is distribution of particles at time *t*?
- Diffusion can be modeled as the PDE

$$\frac{\partial C(x,t)}{\partial t} = D\frac{\partial^2 C(x,t)}{\partial x^2}$$

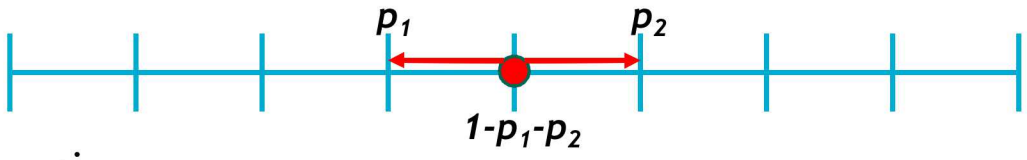
with B.C. + I.C.

- Stochastic process implements many random walkers to statistically approximate a solution
  - Mean position of *N* walkers approaches expected mean of deterministic solution at rate of 1/sqrt(N)



One dimensional random walk case

Model:



- Stochastic Brownian motion
- Particle can either move or not (1-D case: probability p_1 right or p_2 left, with $1-p_1-p_2$ for no move)
- Approximating PDE solutions requires sampling over MANY particles

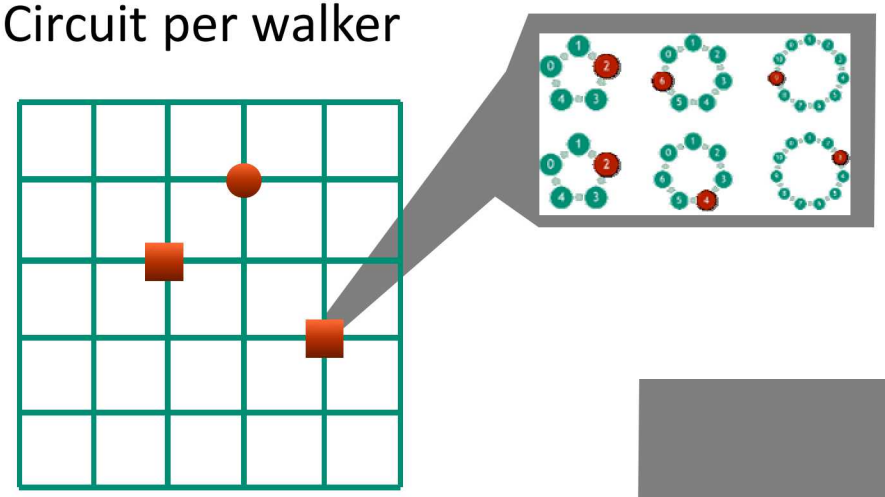
Goal: Ensemble of neurons that represent stochastic particles, such that

- Efficient to update (randomly add / subtract value)
- Has sparse representation
- Requires few neurons
- Scalable across multiple dimensions / multiple particles

Two spiking algorithms for random walk with different costs and benefits

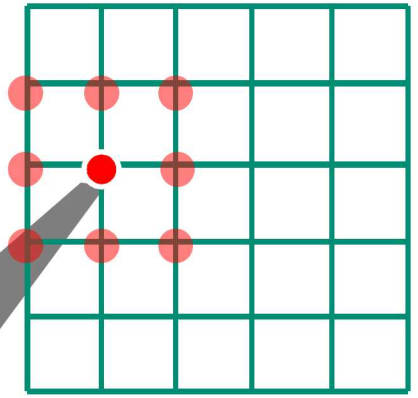
Particle Method

Circuit per walker



Density Method

Circuit per position



Walkers can be represented by sets of neuron rings with modular code

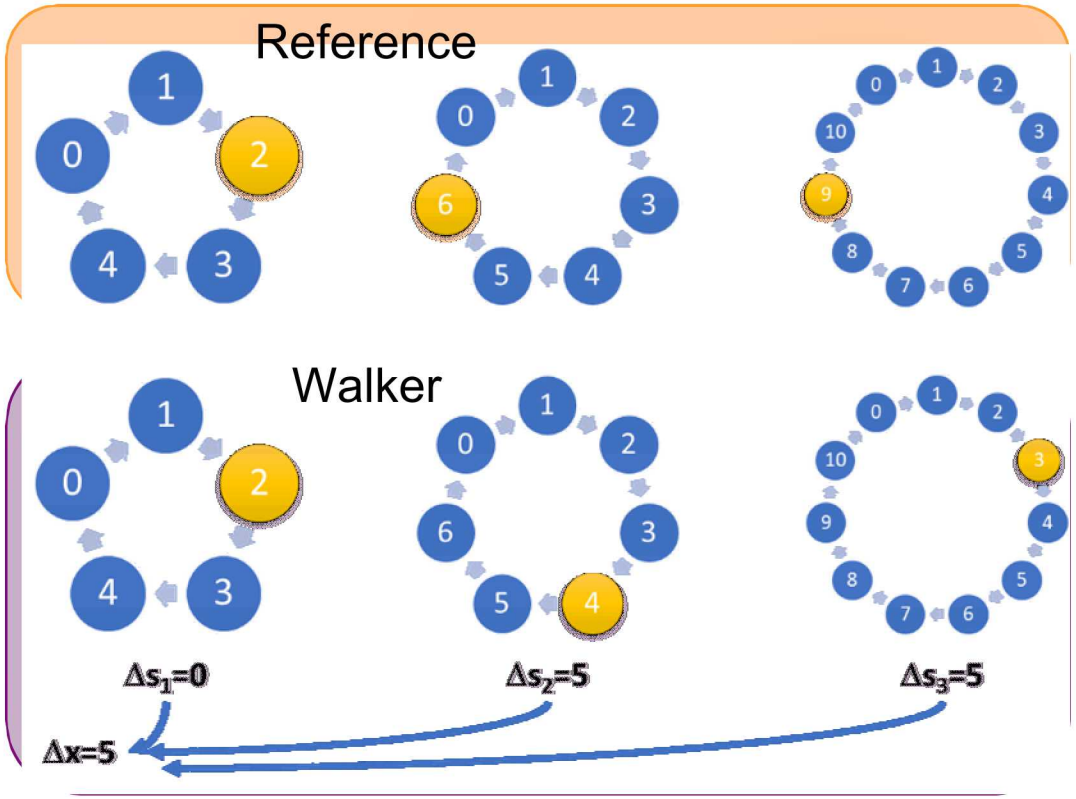
Cycles evolve continuously over time with spiking dynamics

Modular code

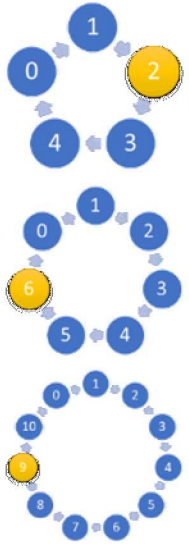
- Capacity = product of ring sizes

Distance from origin or boundary is difference between walker and reference

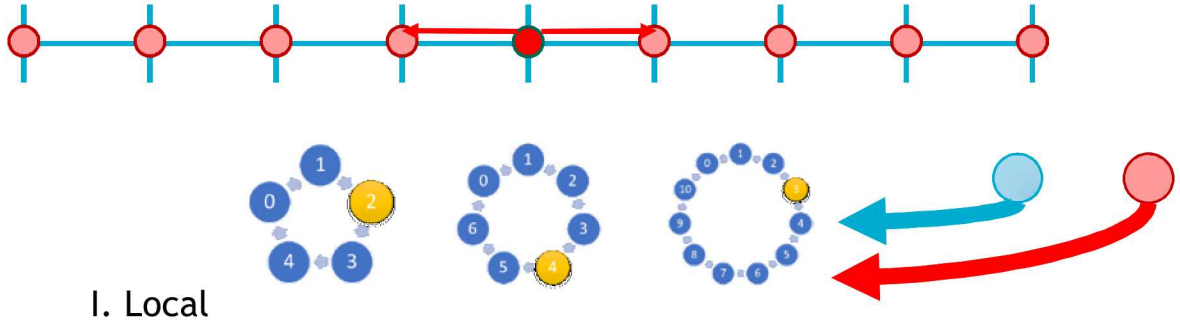
Updates can be performed by pushing walker rings forward or back by 1 while keeping reference constant



Modular model is highly compact and inexpensive to update



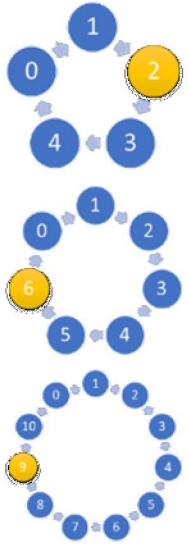
II. Global Reference



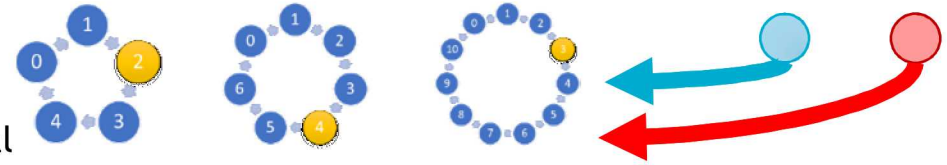
I. Local Reference

Measure	Cost (for k locations; 1-D case)
Position memory per walker	O(k^(1/N)), where N is # rings of length s_n, s.t. $\Pi_N(s_n)>k$
Connection memory per walker	Σ_N (9*s_n+2*(3+s_n mod 3))
Total neurons per walker	2+Σ_N (s_n+2*(3+s_n mod 3))
Time per physical timestep	2
Position energy per timestep	O(N)
Update energy per timestep	O(N)

Modular model is highly compact and inexpensive to update



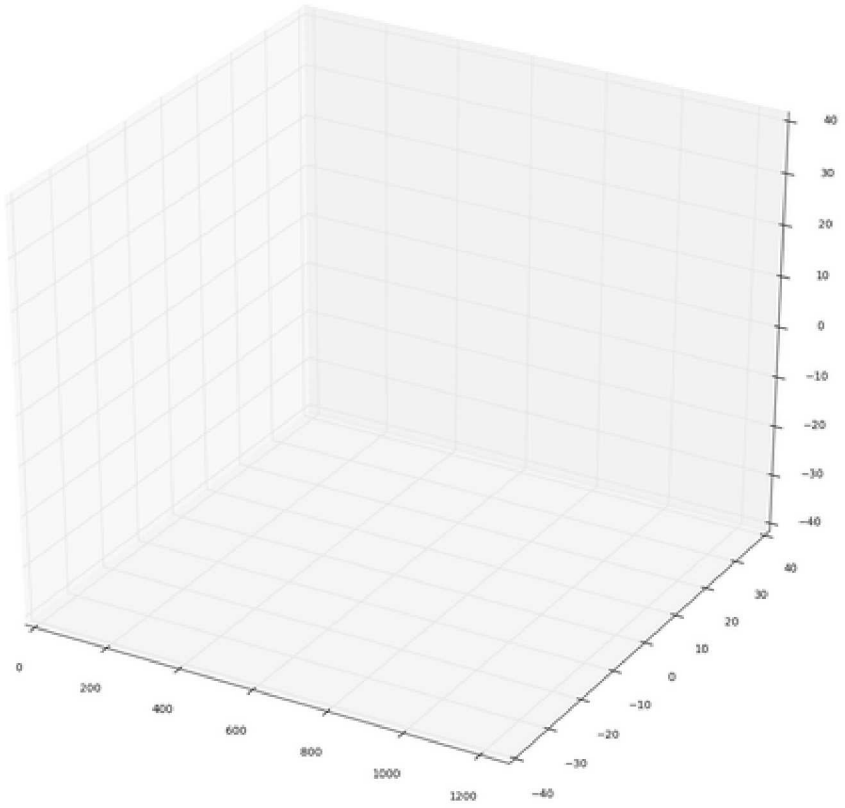
II. Global Reference

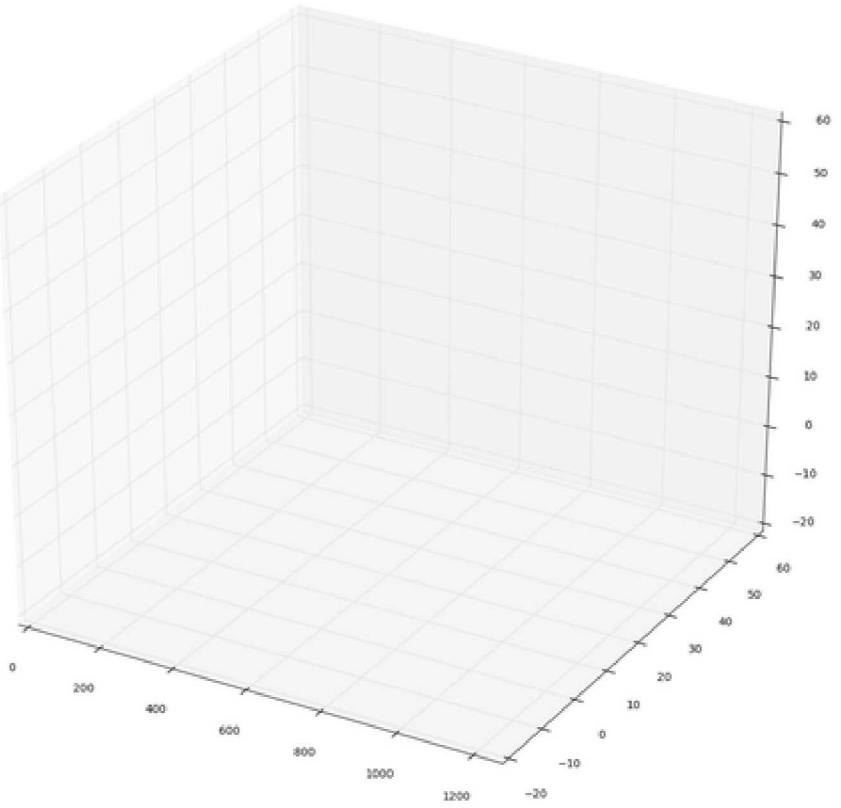


I. Local Reference

Measure	k = 1000, 1D	k = 1000, 2D	k=10000, 1D
Modular set	{7, 11, 13}	{7, 11, 13} x2	{17, 23, 29}
Position memory per walker	1001	1002001	11339
Connection memory per walker	293	583	639
Total neurons per walker	59	118	101
Time per physical timestep	2	2	2
Position energy per timestep	3	6	3
Update energy per timestep	3	3	3

Neural random walkers evolve as expected

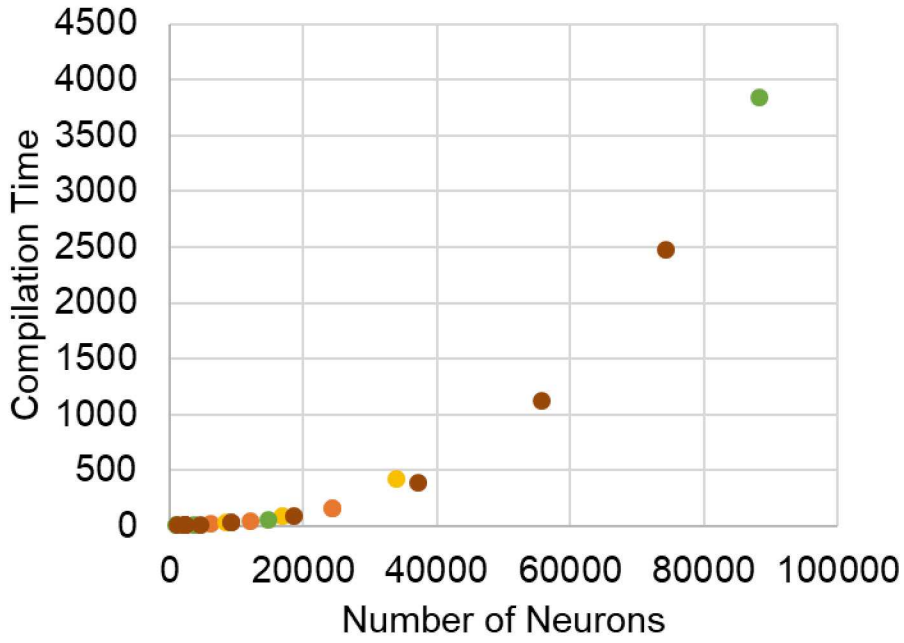




SpiNNaker 48 chip test board
250 walkers, 1250 time steps
with equal probability of moving (left) or strong bias (right)

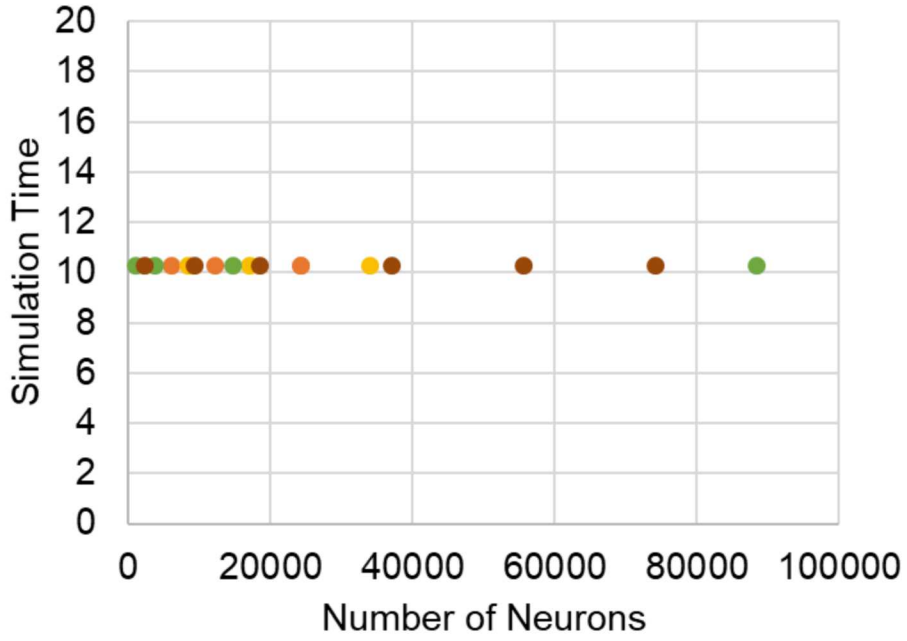
Results of mapping particle-based RW to SpiNNaker

- Graph partitioning and compilation is currently prohibitive
  - Even though more walkers is simple repetition of circuit, software stack cannot readily account for it
  - In practice, mapping likely only needs to be performed once, and it is independent of simulation time (not shown)



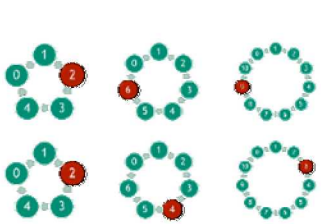
Results of mapping particle-based RW to SpiNNaker

- Graph partitioning and compilation is currently prohibitive
  - Even though more walkers is simple repetition of circuit, software stack cannot readily account for it
  - In practice, mapping likely only needs to be performed once, and it is independent of simulation time (not shown)
- Simulation on SpiNNaker is truly constant time – more walkers requires more neurons and more cores, but is embarrassingly parallel
  - Simulation IS throttled back to mitigate implications of communication bottlenecks and is running quite slow



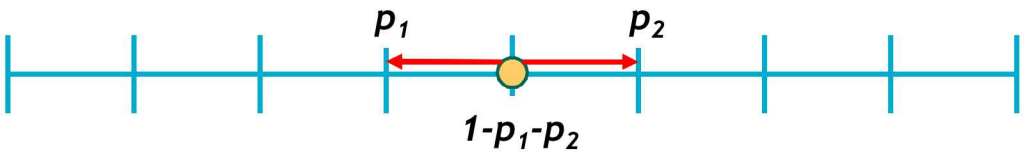
Challenges in putting *particle random walk* algorithm on SpiNNaker

- At hardware level, SpiNNaker is a parallel Von Neumann RISC architecture, with memory and communication fabric heavily skewed towards event-driven neural applications
  - **Generic connectivity, event-driven communication, bias towards simple computation**
  - **Fabric is well-suited for spiking neural algorithms**
- Most straightforward approach to use SpiNNaker is through PyNN (specifically sPyNNaker backend)
  - **PyNN is developed for generic computational neuroscience models**
  - **Use case envisions large ensembles of a small number of populations with statistical connectivity**
  - **sPyNNaker library implements a subset of PyNN tools**

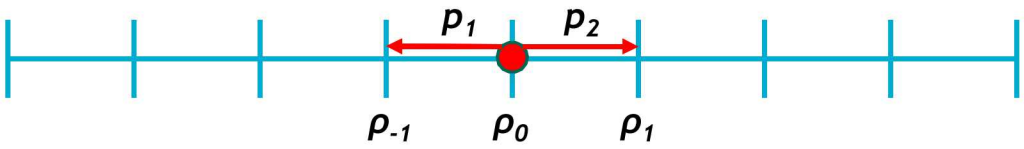


Mismatch of programming and compilation models with generic spiking algorithms likely results in a 50% efficiency penalty on mapping random walks to SpiNNaker!

Neural approach enables us to model either *walkers* or *mesh*



Instead of each particle owning a population of neurons representing the particle's location…

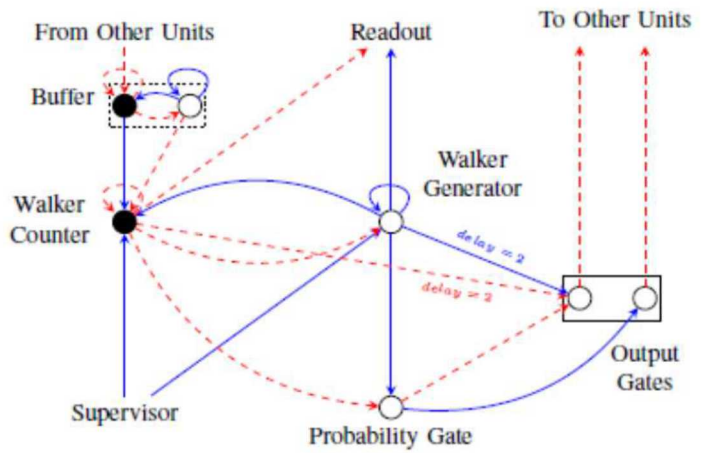


…have each location own a population of neurons that represent the probability density of particles

Density model repeats basic circuit at every vertex of mesh

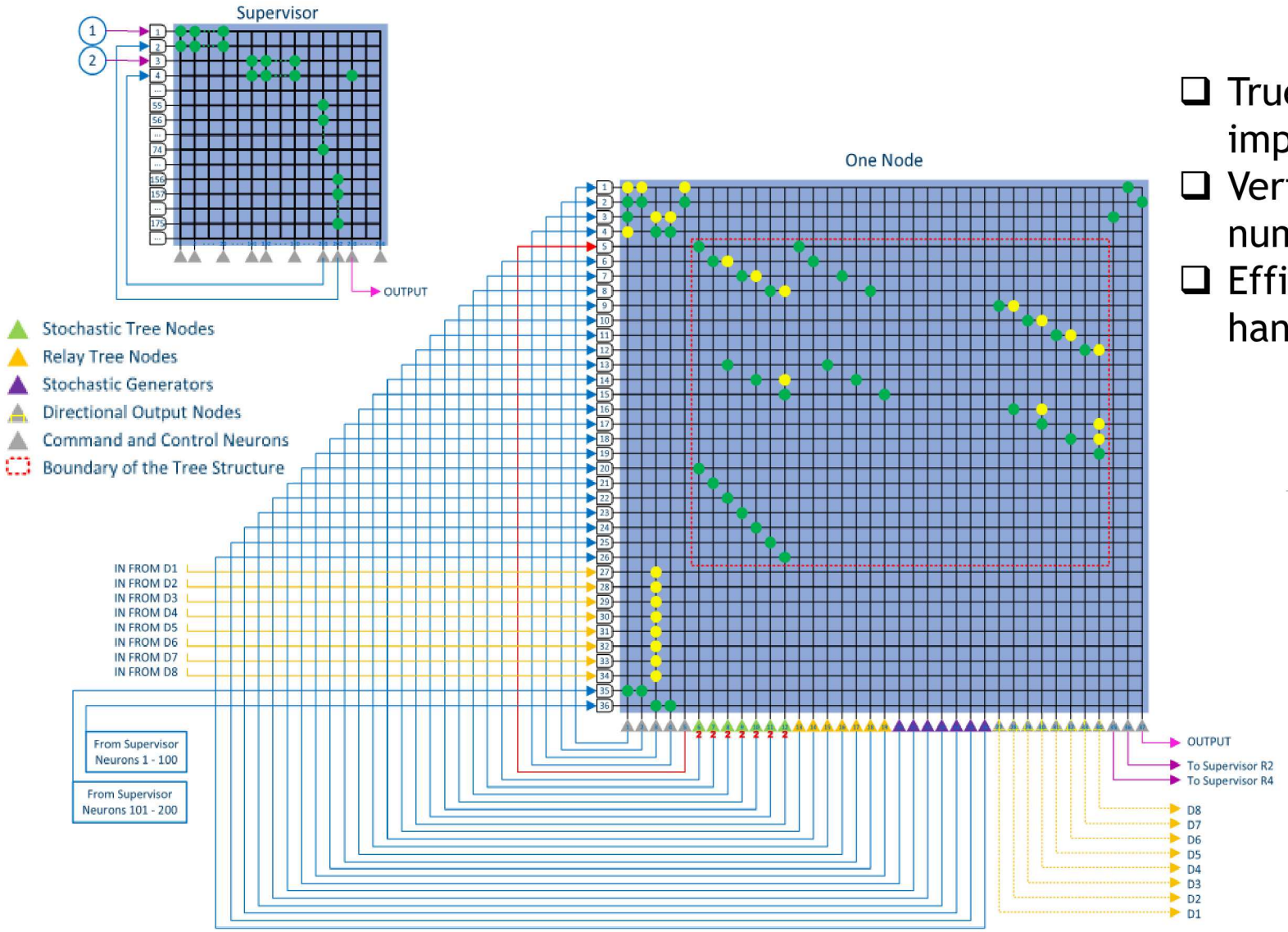
Each vertex encodes density of particles in the internal potential of certain nodes

Each time step "hands off" particles to connected vertices according to probabilistic maps

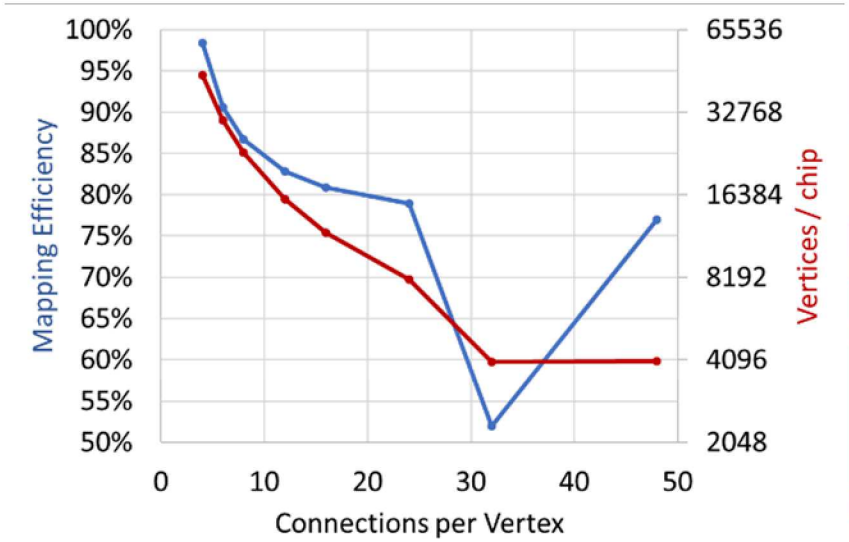


Measure	Cost (for k locations, simulating N walkers; 1-D case)
Walker memory	O(1)
Connection memory	O(k)
Total neurons	O(k)
Time per physical timestep	O(max(ρ_i)), where ρ_i is the density of walkers at each location
Position energy per timestep	O(N)
Update energy per timestep	O(N)

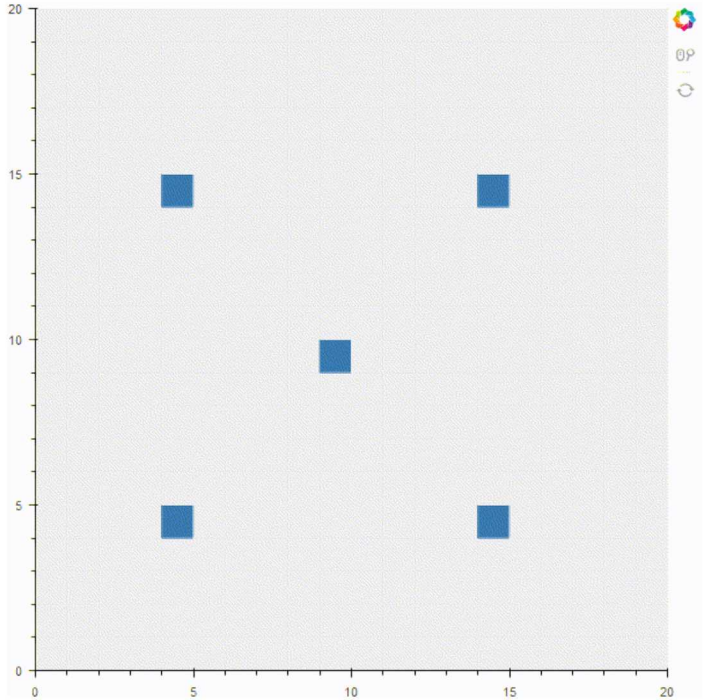
TrueNorth Corelet for ND-Random Walk Vertex



- TrueNorth is quite efficient at implementing density RW algorithm
- Vertex size scales linearly with number of outputs
- Efficiency can increase with by-hand mapping

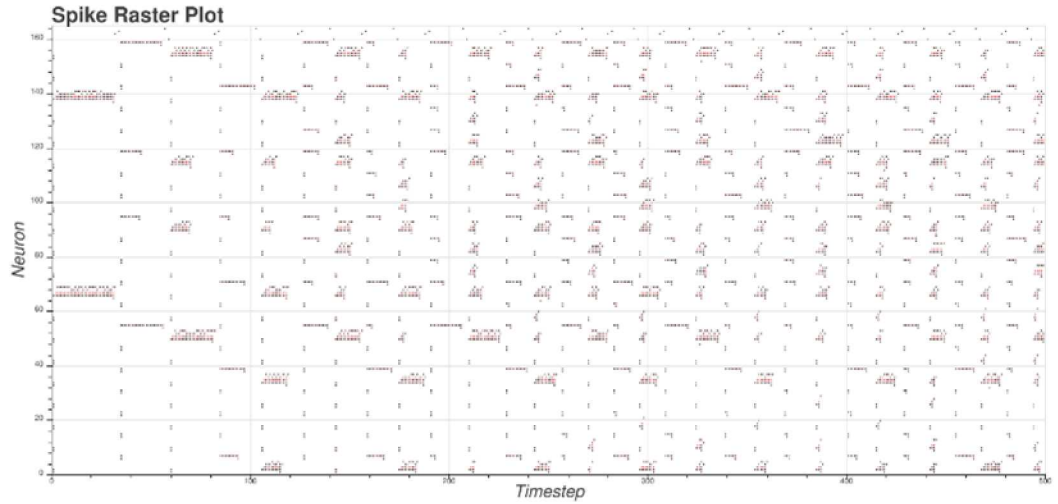


Speed of simulation depends on density distribution of walkers



Simulation on IBM TrueNorth hardware

- 20x20 grid
- 30 walkers initialized at 5 points
- 2000 timepoints

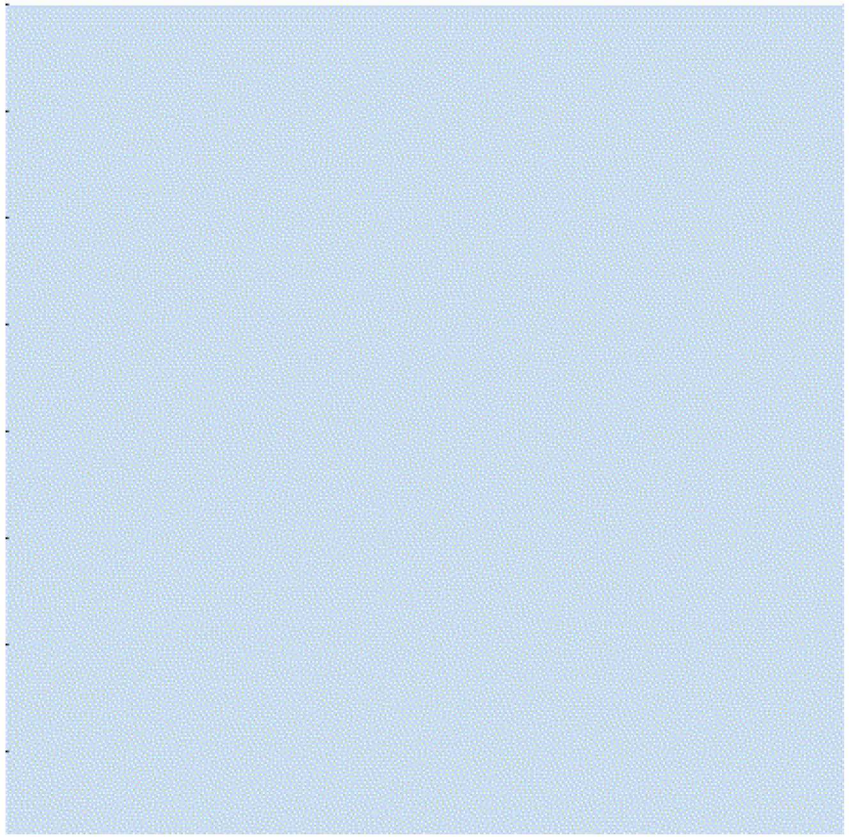


Density model can model arbitrary graphs, enabling complex behaviors

- This example uses a mesh with one-way edges that permit walkers to move into colored parts of the image, but not out

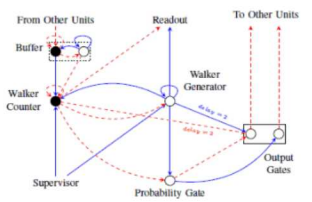
- Scale of simulation
  - 1600 vertices
  - 8000 walkers
  - 1000 timesteps
  - 19205 neurons
  - 60802 synapses

- We expect this could run in ~1 second on TrueNorth



Challenges in putting *density random walk* algorithm on TrueNorth

- At hardware level, TrueNorth has a set of conventional crossbar memory cores, with communication fabric heavily skewed towards event-driven neural applications
  - **Limited to spiking communication, connection graph biased heavily towards local circuits**
  - **Fabric is well-suited for spiking neural algorithms**
- Most straightforward approach to program TrueNorth through custom Corelets
  - **EEDN tool is not appropriate for custom algorithms**
  - **Compilation and optimization is non-trivial**

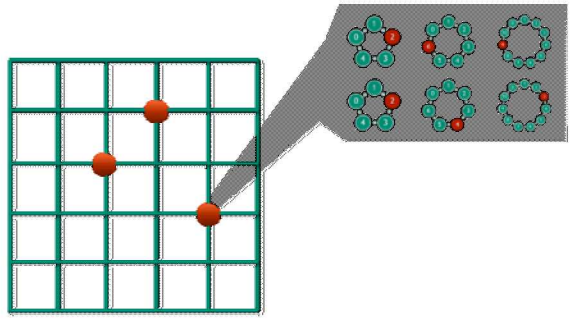


Communication restrictions introduced by cross-bar architectures are ***NOT*** a significant impediment to local communication in density random walk algorithm!

Each method offers unique advantages

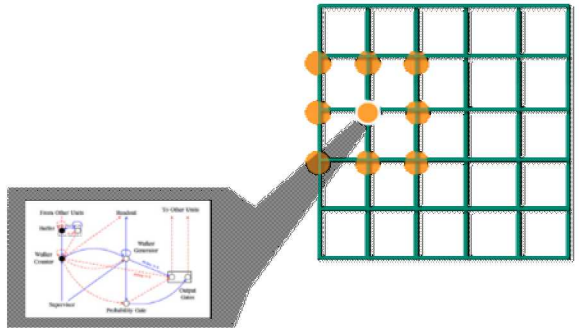
Particle method

- Path dependent behavior is readily available
- Communication is entirely local within particles (embarrassingly parallel)
- With unlimited neurons, can run in constant time
- **Ideal for sparse particles in large spaces**

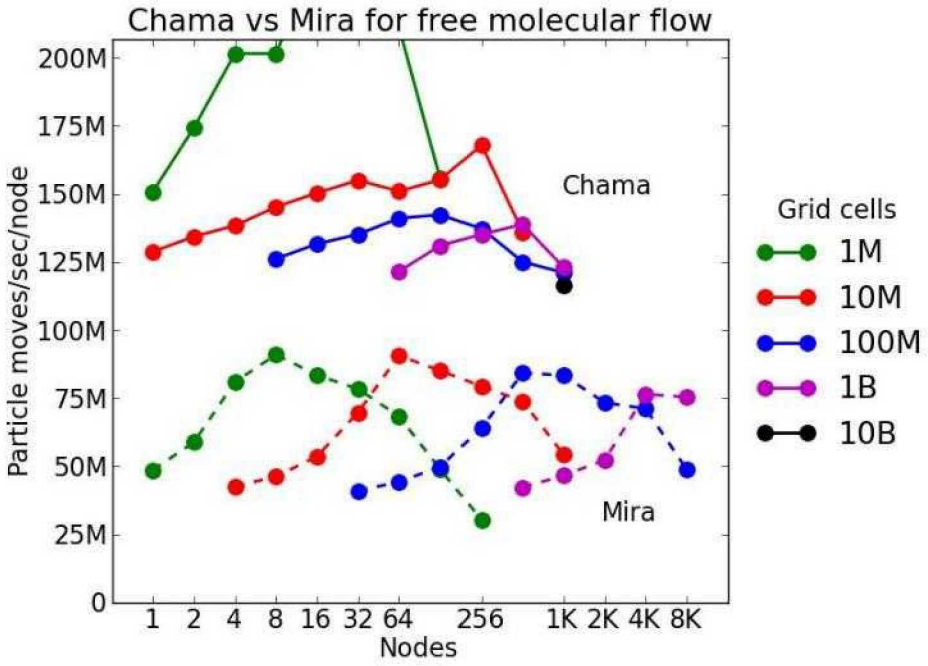


Density method

- Densities are readily available at all times
- Non-local or other complex graphs can easily be implemented
- With limited neurons, can tradeoff statistical approximation (i.e., number of walkers) with longer or shorter simulations
- **Ideal for dense particles in small spaces**



So can we say anything about mapping codes to a supercomputer?

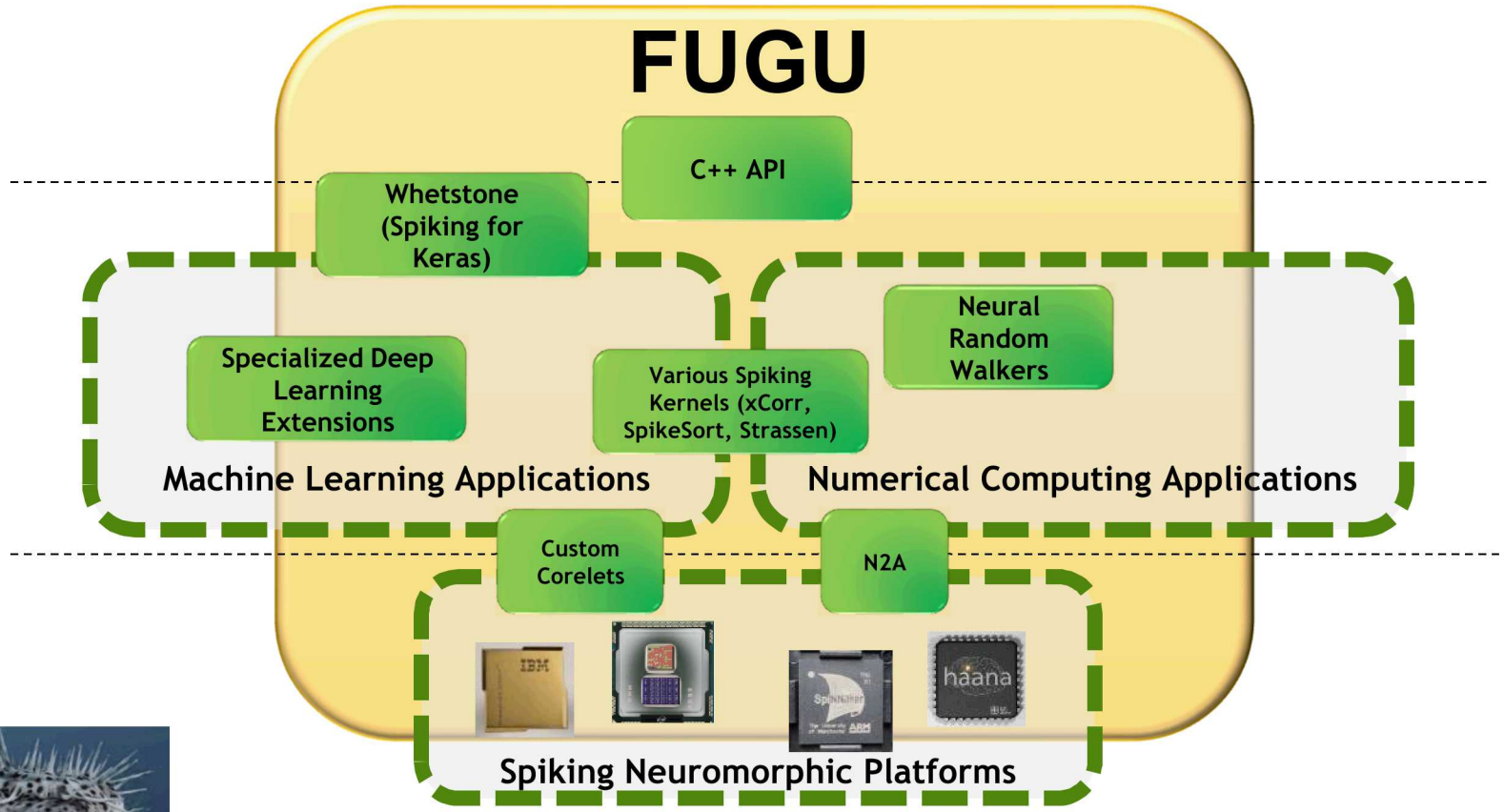


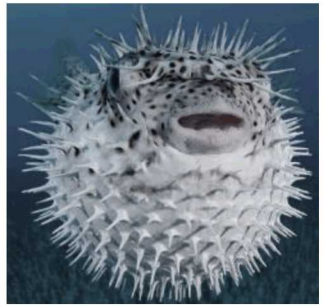
http://sparta.sandia.gov/bench.html

- Free molecular flow is a benchmark problem for HPC systems (see SPARTA Direct Simulation Monte Carlo benchmark ←)
  - Each Mira node ~80W
- Where would neural hardware end up?
  - *Most neural hardware is throttled back in speed (kHz vs GHz) to save power*
  - *Most HPC systems are tuned for speed, not power*
- 1M grid cells in 3D (6 possible movements) would require ~34 TrueNorth chips.
  - Ignoring chip-chip delays and neuron overhead, **could run up to 1,000M moves per second at ~4W**

- 10M walkers on SpiNNaker would require ~250,000 chips without custom neuron types

Neural random walk algorithms fit into emerging ASC-framework *Fugu* for neural codes





Fugu = pufferfish (why? Pufferfish have spikes…)

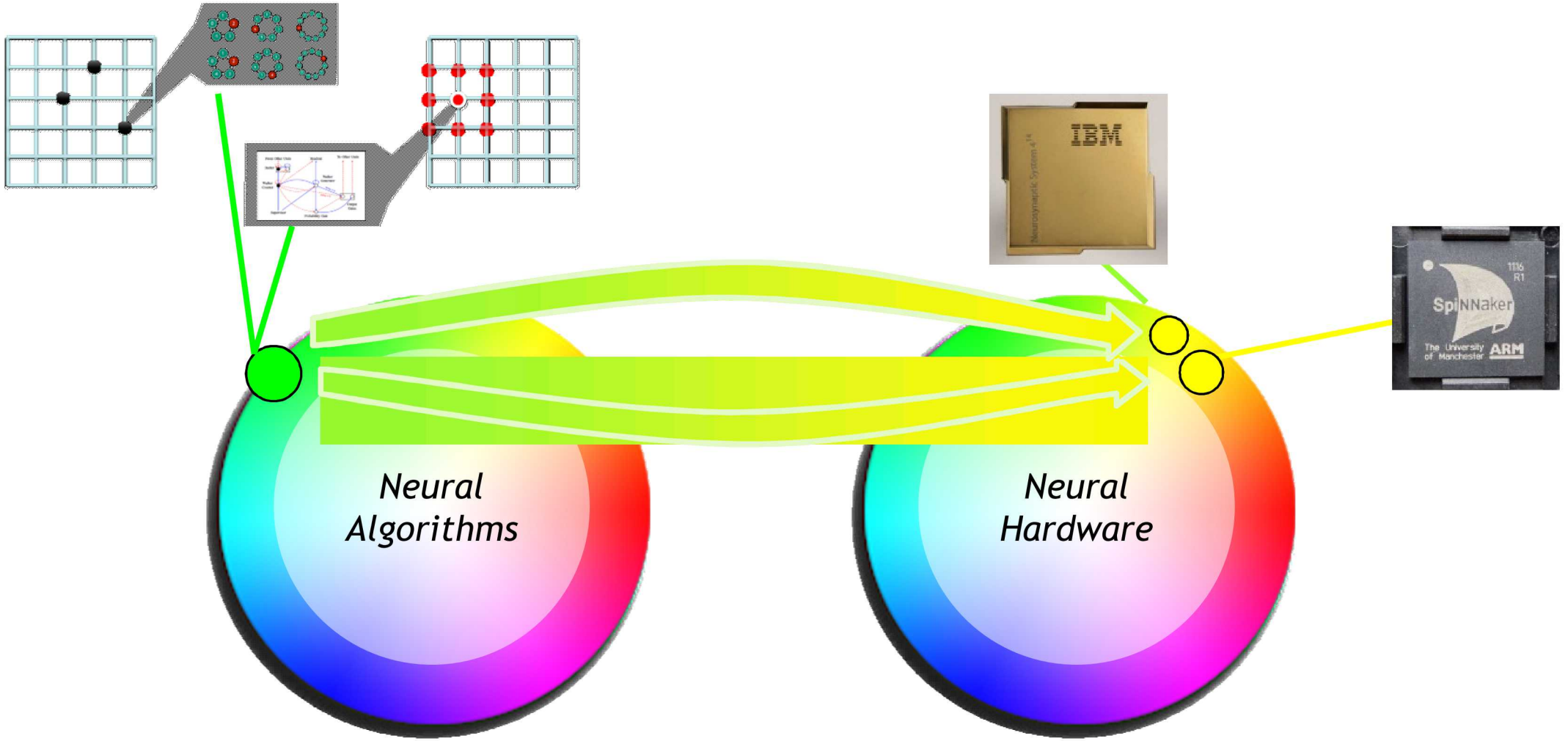
Conclusions and Future Questions…

- **Today's neuromorphic hardware is already suitable for large scale scientific computing applications**
  - Potential to offer real power savings
  - TrueNorth-like architectures are particularly well-suited for diffusion-based applications

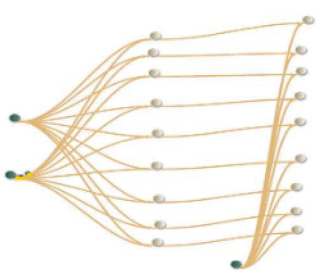
- **Programming environment remains a challenge**
  - Software stack designed for neurobiology-inspired algorithms is not well suited for efficient scientific codes
  - Compilation efficiency and time is a near-term impediment

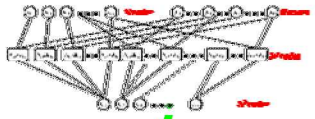
- **Still many open questions and opportunities for spiking neural algorithms for scientific computing**
  - Boundary conditions, interactions, non-local movement, high dimensions, etc…
  - What are the tradeoffs between different neural information representations?
  - Can we leverage learning?

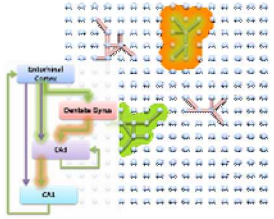
Thank You!

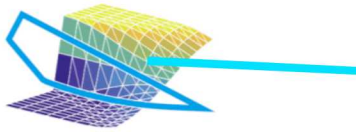

IBM
SpiNNaker
The University of Manchester
ARM
Neural Algorithms
Neural Hardware

Quick Ad: We are actively looking for postdocs and staff to help bring the right neural computing solutions to real world applications!









WHETSTONE