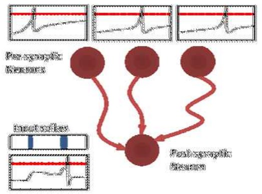
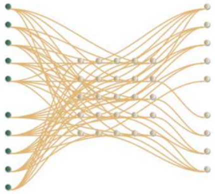
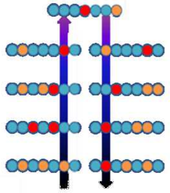
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

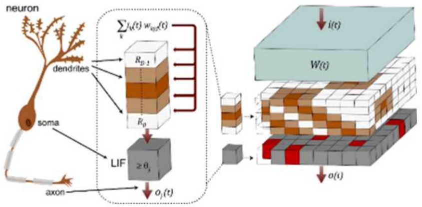
SAND2019-4662C

Mosaics





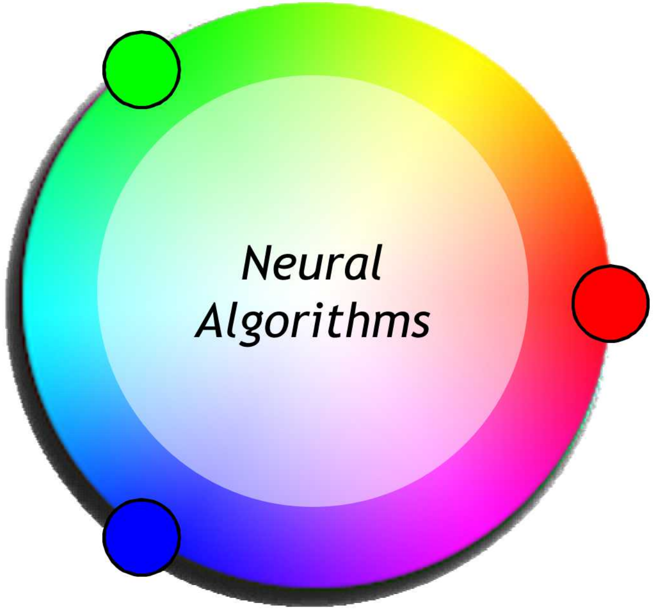


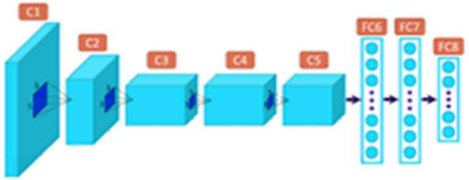


PRESENTED BY

Brad Aimone

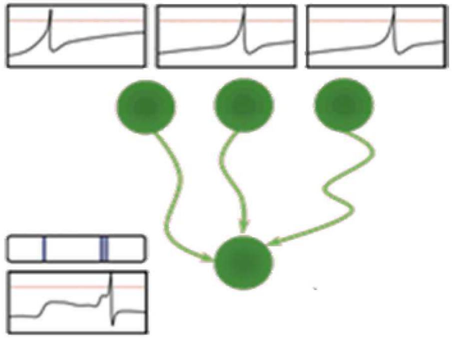
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

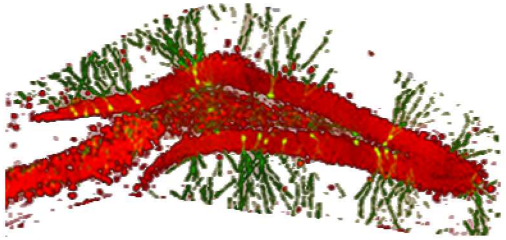




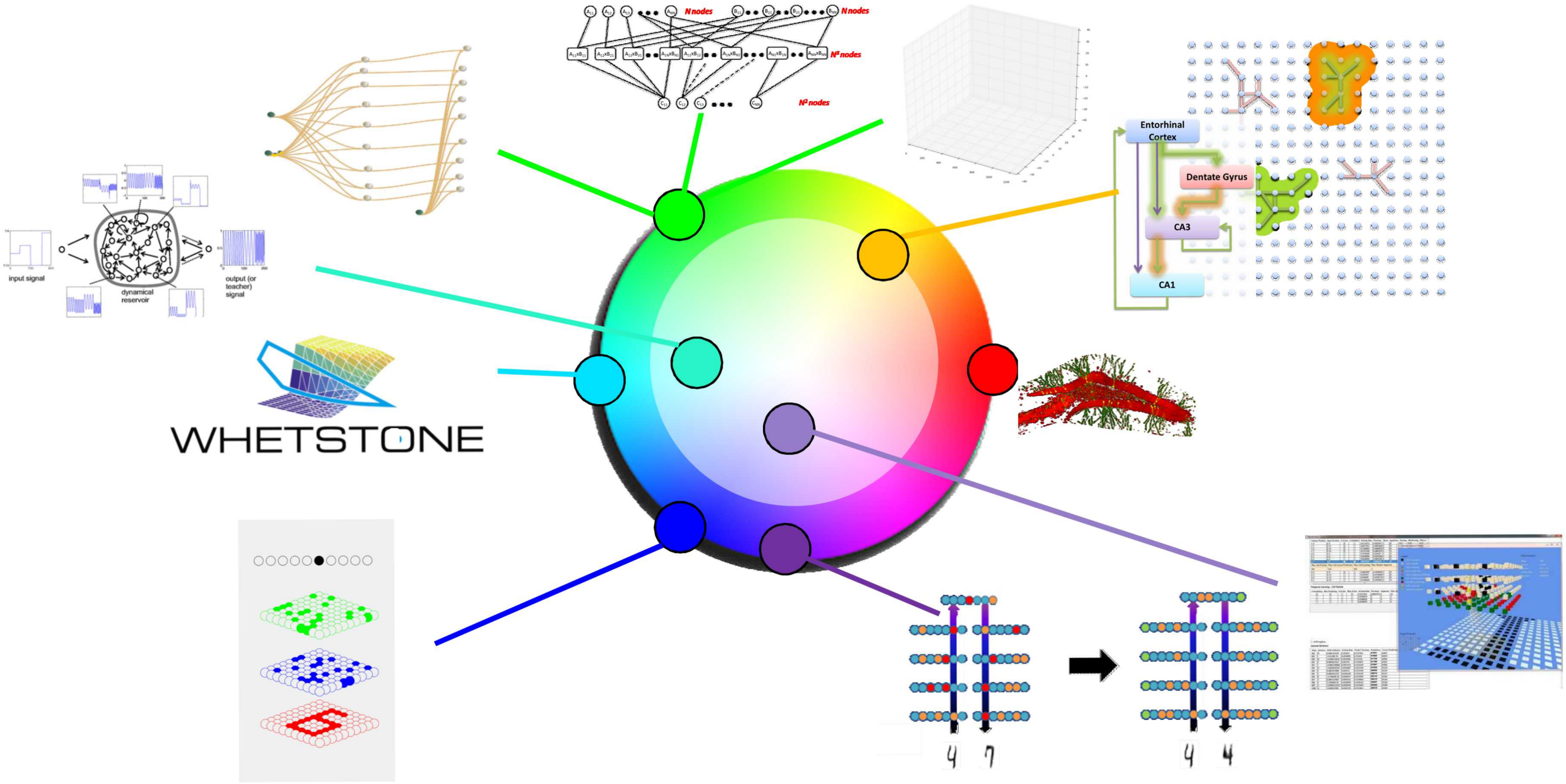
Artificial neural networks
- Generic layers of non-linear nodes
- SGD optimization of weights
- Powerful machine learning capabilities through learning sequential non-linear mappings and function approximation

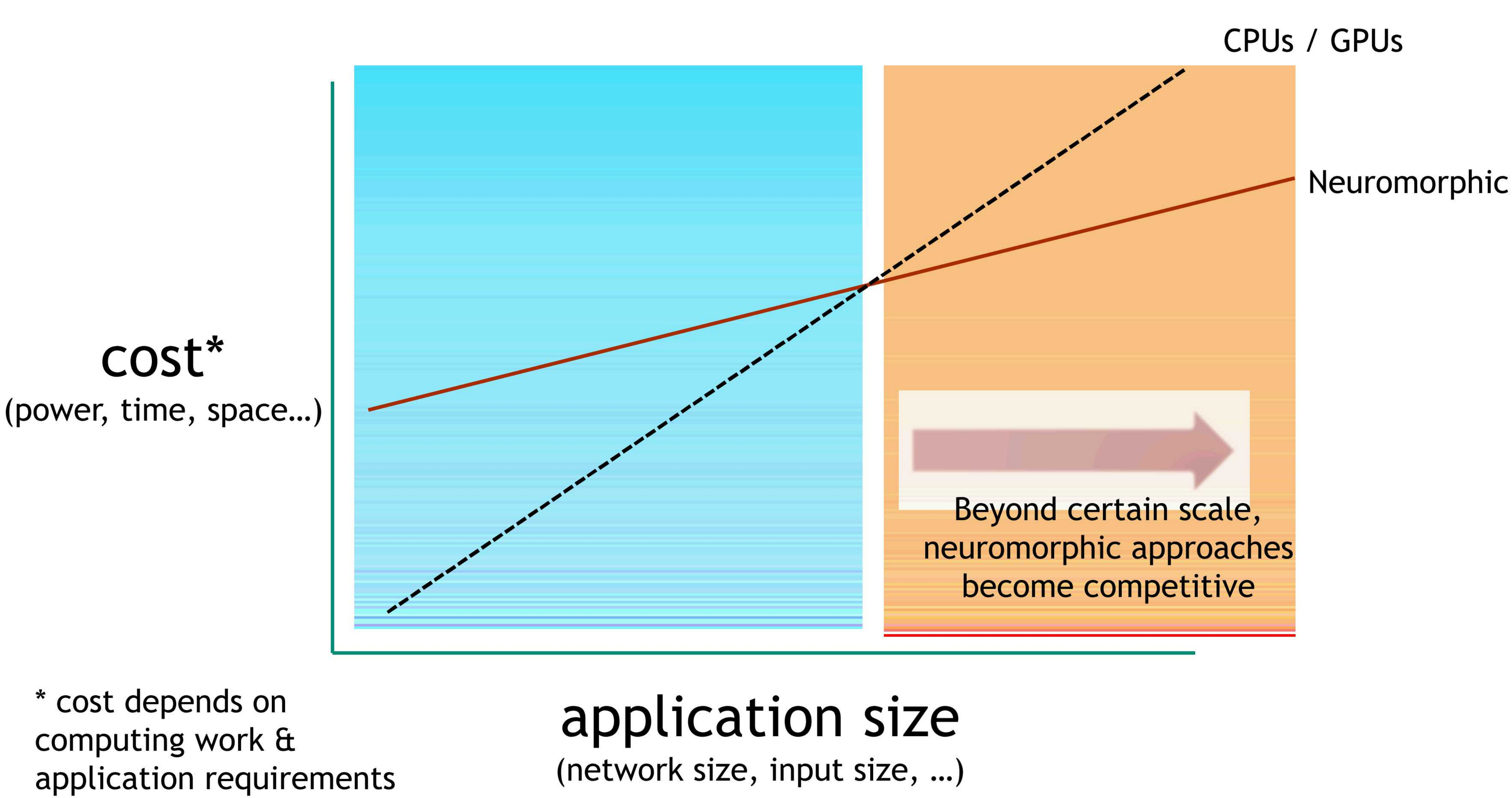
Spiking neural algorithms
- Hand-crafted circuits of spiking neurons
- Model of parallel computation
- Energy efficiency through event-driven communication and high fan-in logic



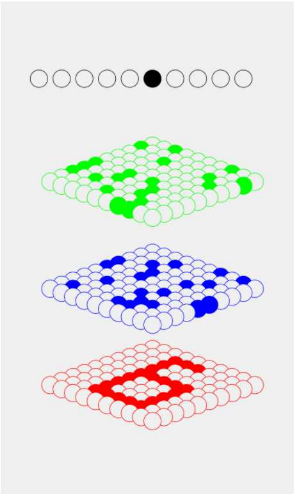


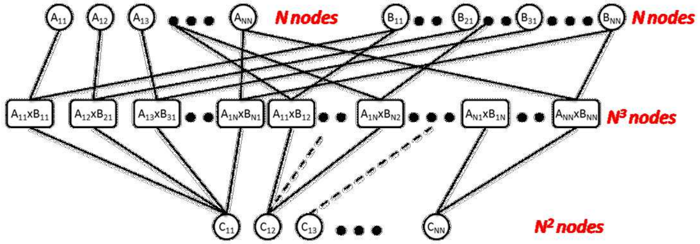
Neuroscience-constrained algorithms
- Circuit architecture based on local and regional neural connectivity
- Computation incorporates broad range of neural plasticity and dynamics
- *Generally still unexplored from algorithms perspective*

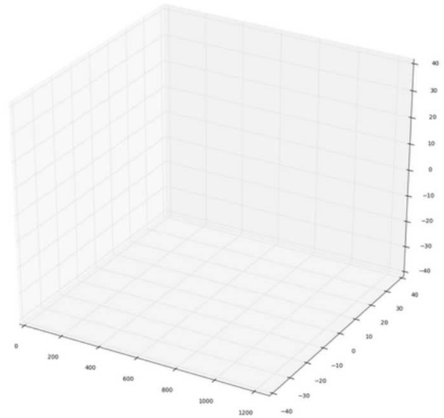

N nodes
N nodes
N² nodes
N² nodes
Entorhinal Cortex
Dentate Gyrus
CA3
CA1
input signal
dynamical reservoir
output (or teacher) signal
WHETSTONE
4 7
4 4



* cost depends on computing work & application requirements







Neural Networks

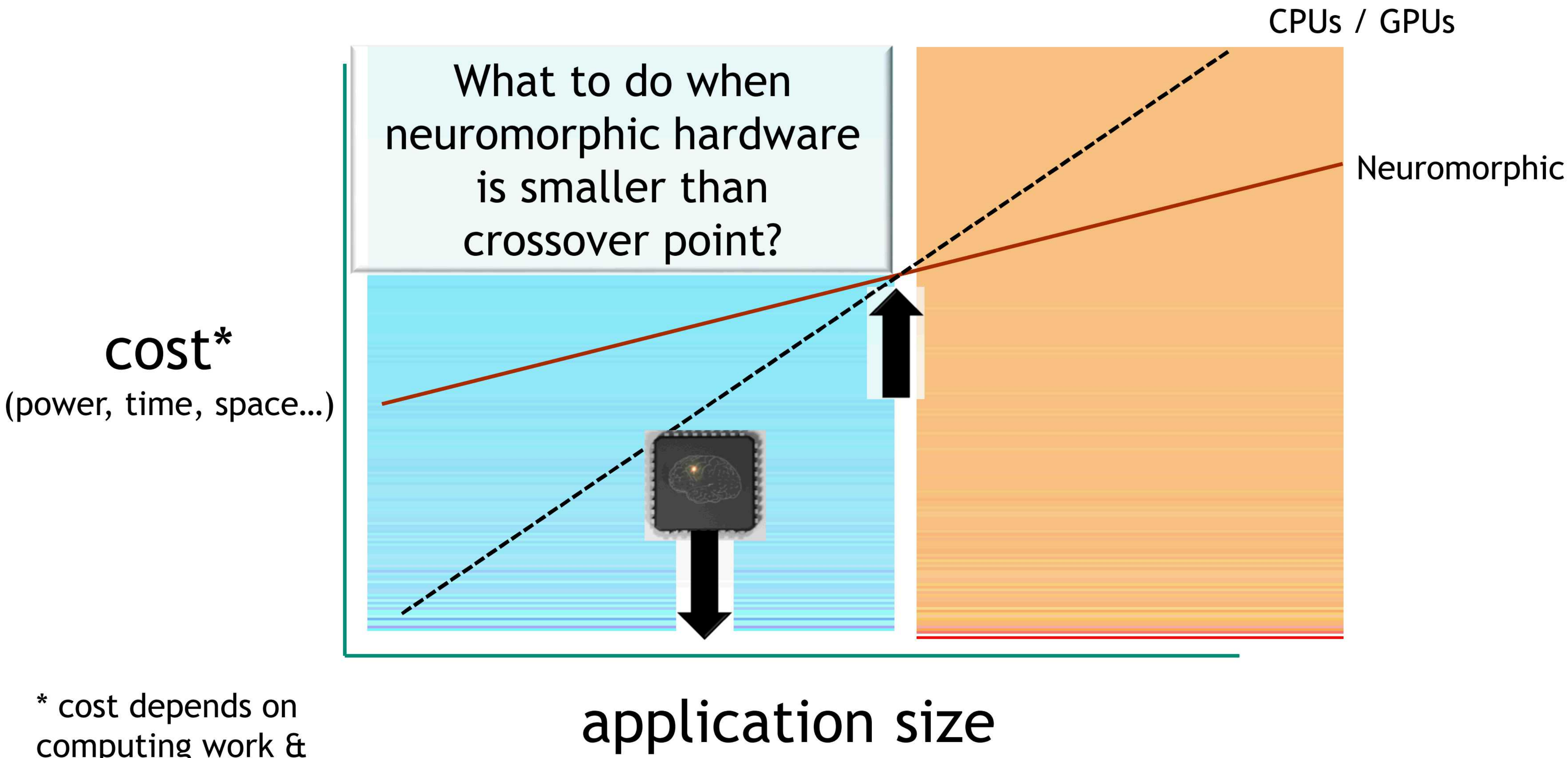
- Better-suited for neuromorphic hardware than many other machine learning techniques
- ANNs only became broadly world class when they reached substantially large sizes

Matrix Multiplication

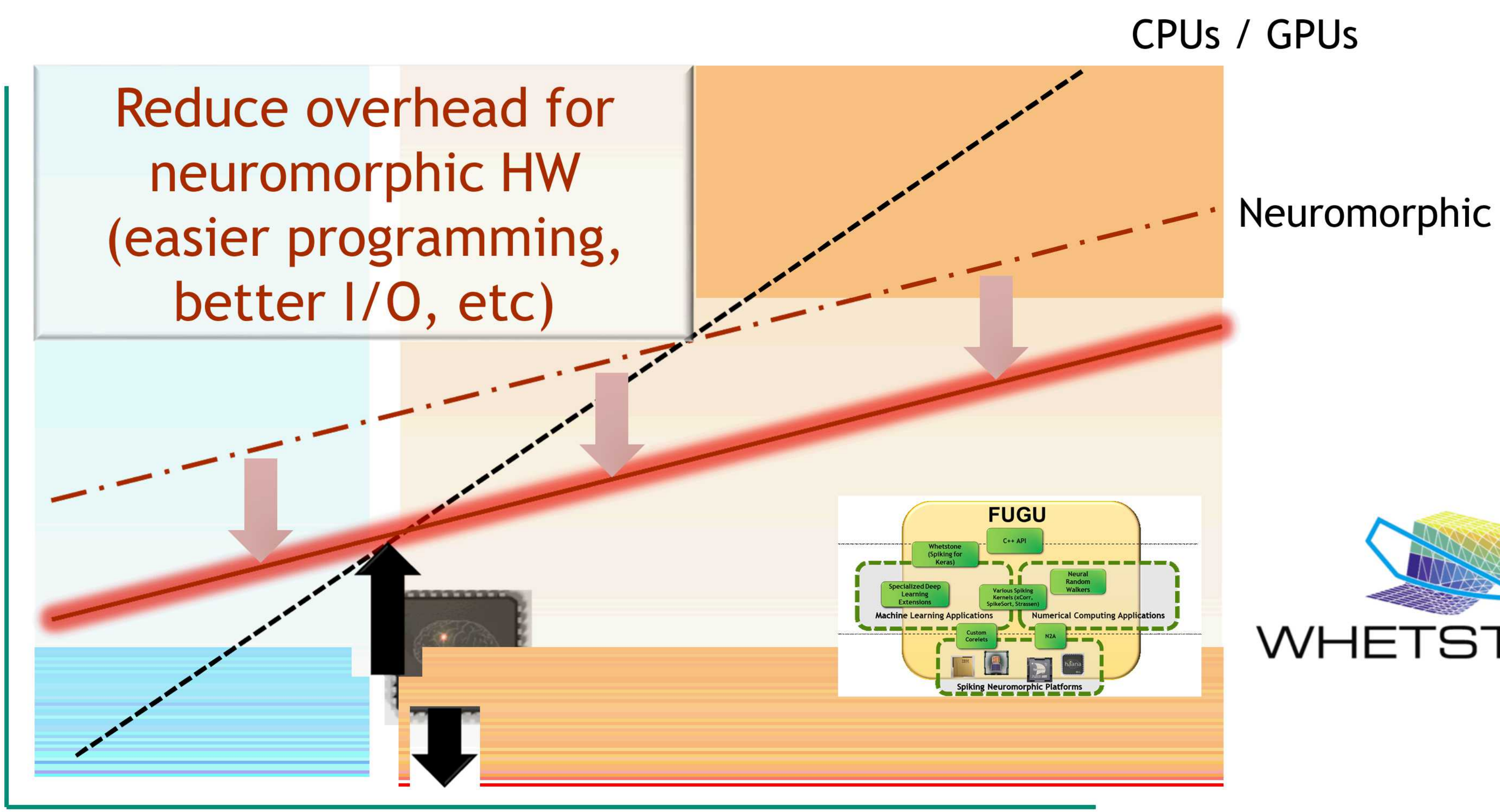
- Neural algorithms can improve implementation of Strassen-techniques
- Strassen techniques only make sense for large matrix multiplications

Partial Differential Equations

- Neural algorithms can efficiently implement Monte Carlo solutions for solving diffusion-based PDEs
- Monte Carlo methods make most sense for high-dimensional PDEs



* cost depends on computing work & application requirements

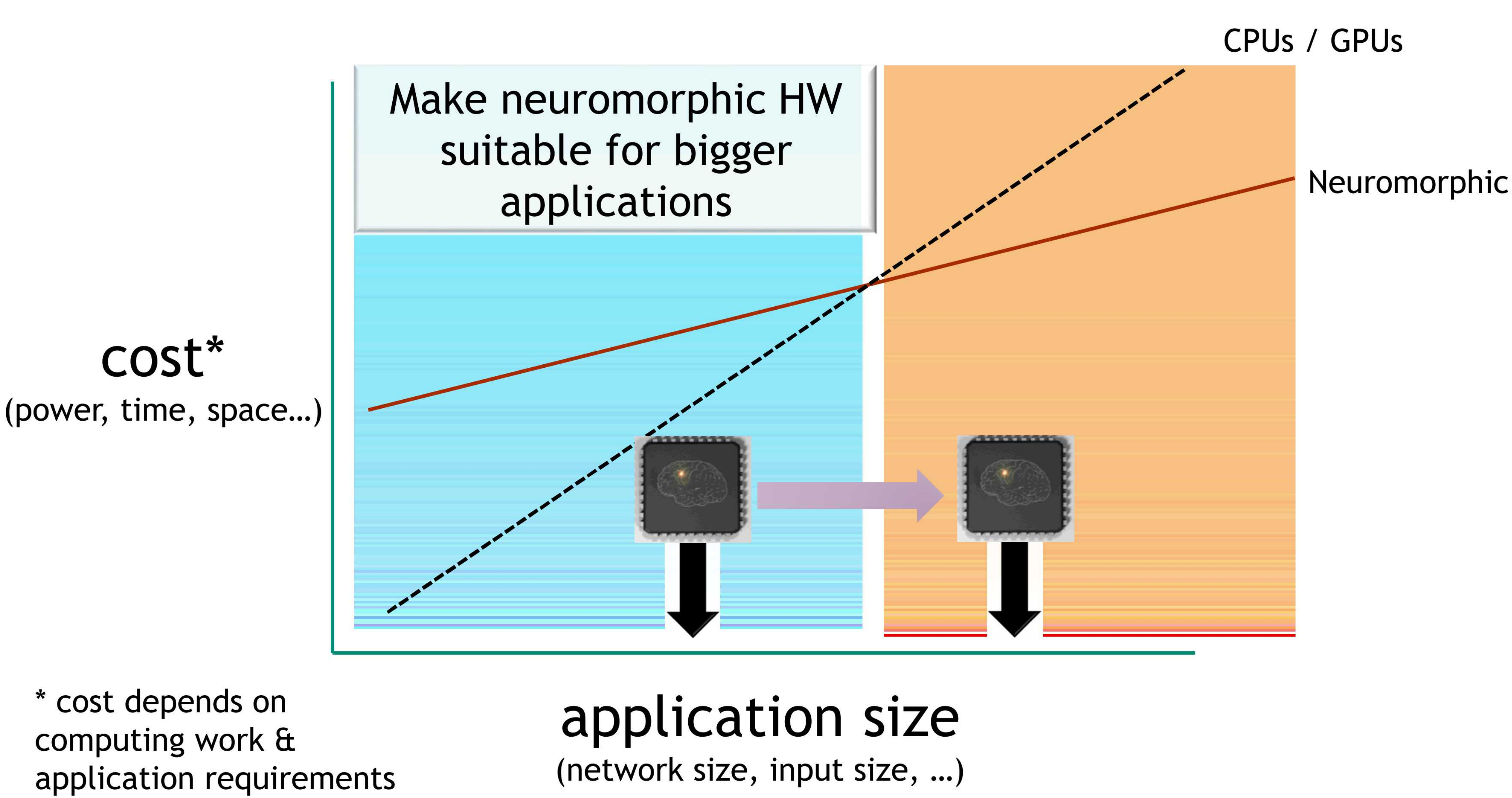


cost*
(power, time, space…)

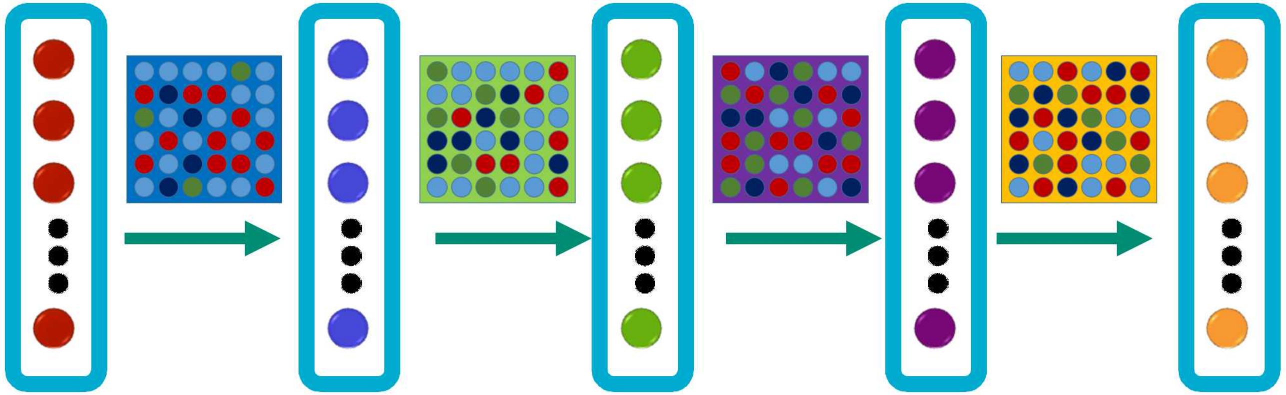
application size
(network size, input size, …)

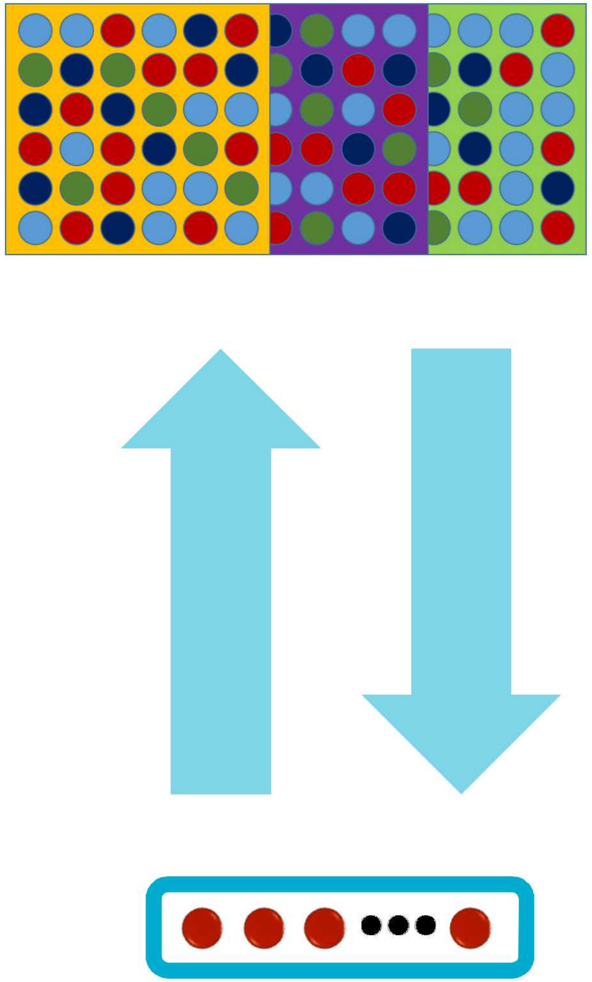
* cost depends on computing work & application requirements

WHETSTONE



* cost depends on computing work & application requirements





Memory is cheap. We should take advantage of that.

More synapses (whether as arrays or tables or whatever) is not really the problem

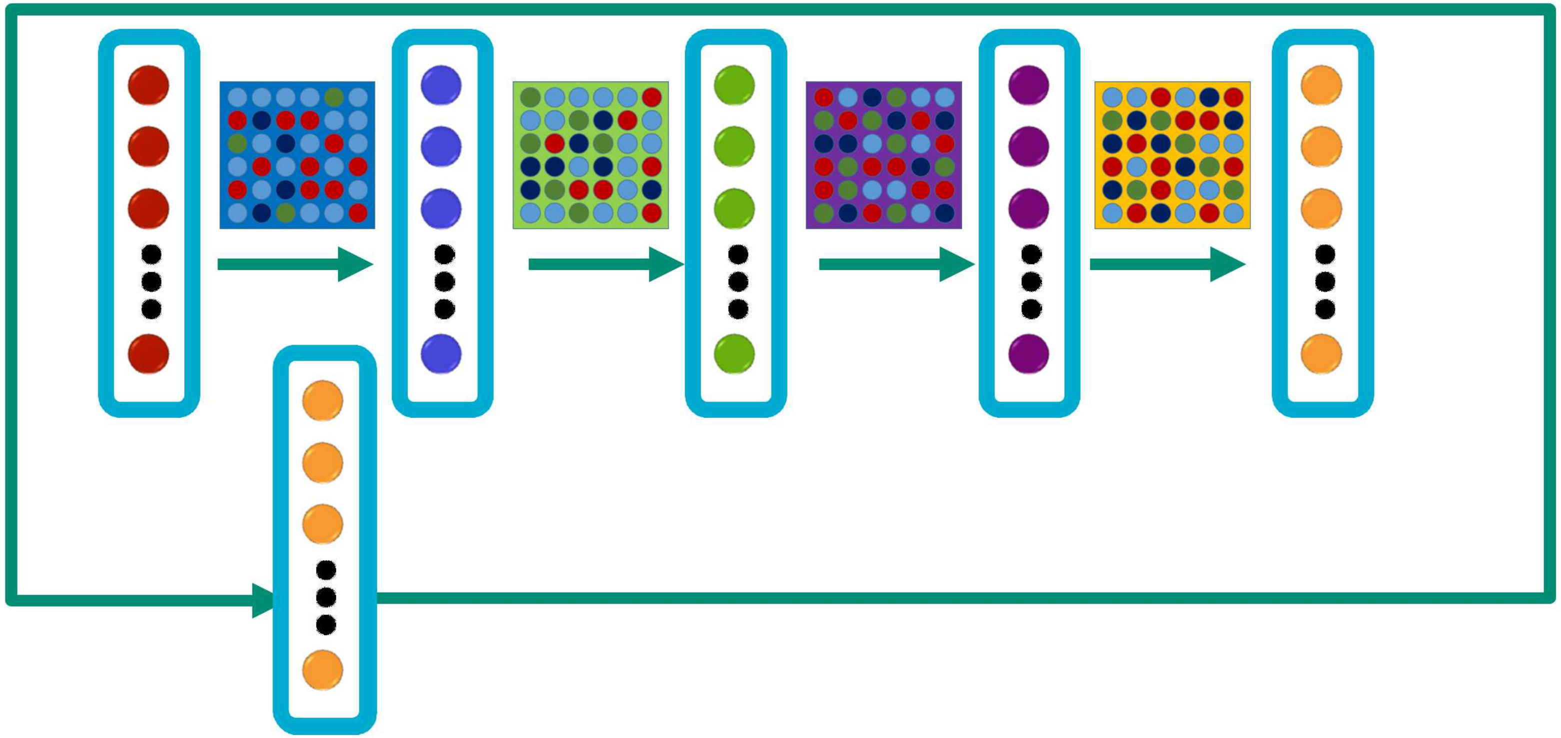
Communication is expensive. And it scales poorly with neurons and synapses…

Can we formulate an approach to mitigate this?

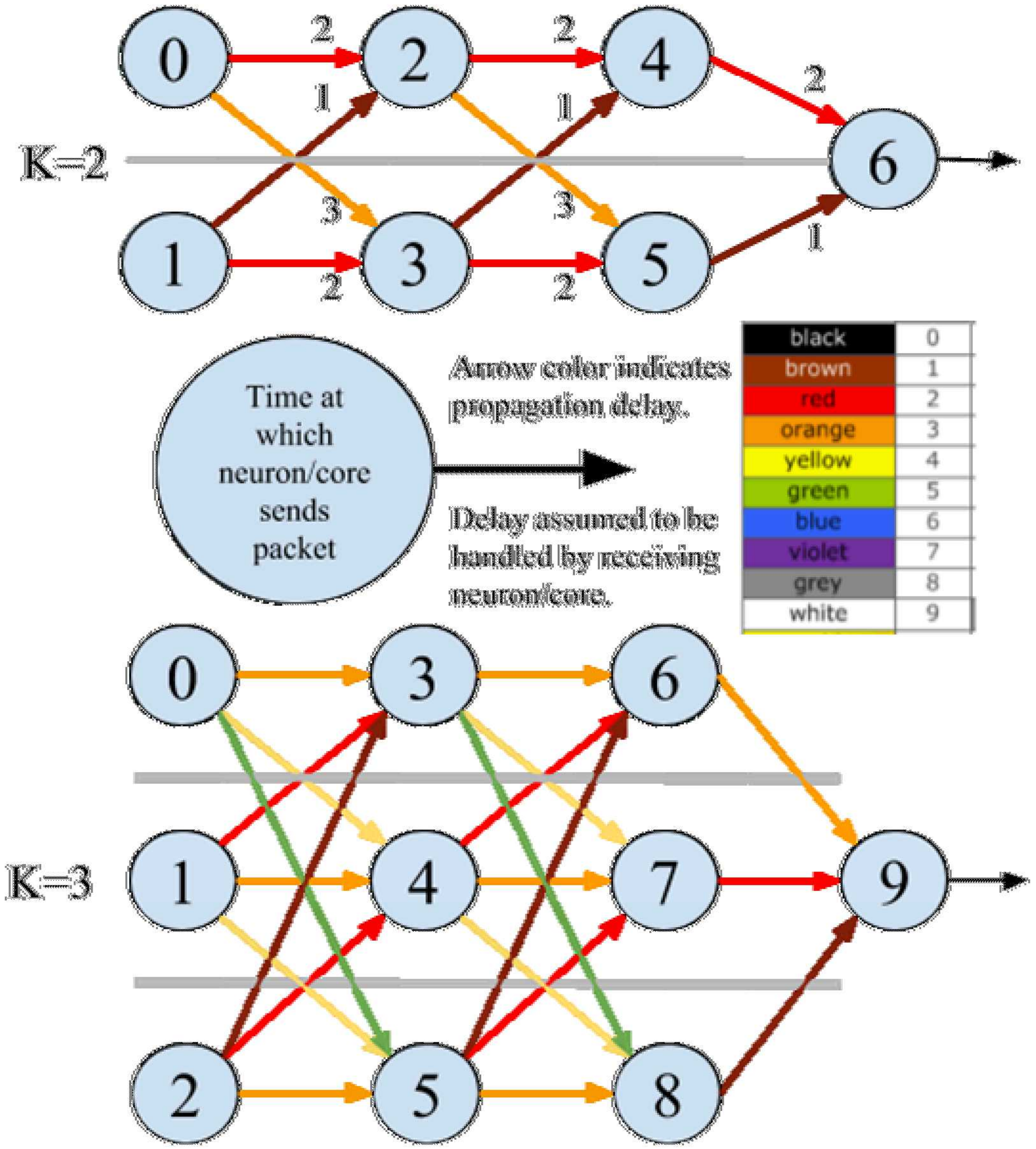
Neurons are not as cheap. And we need a lot of them.

But depending on the algorithm, they are often unused for large periods of time.

Can we better take advantage of what we have?



Temporal Groups



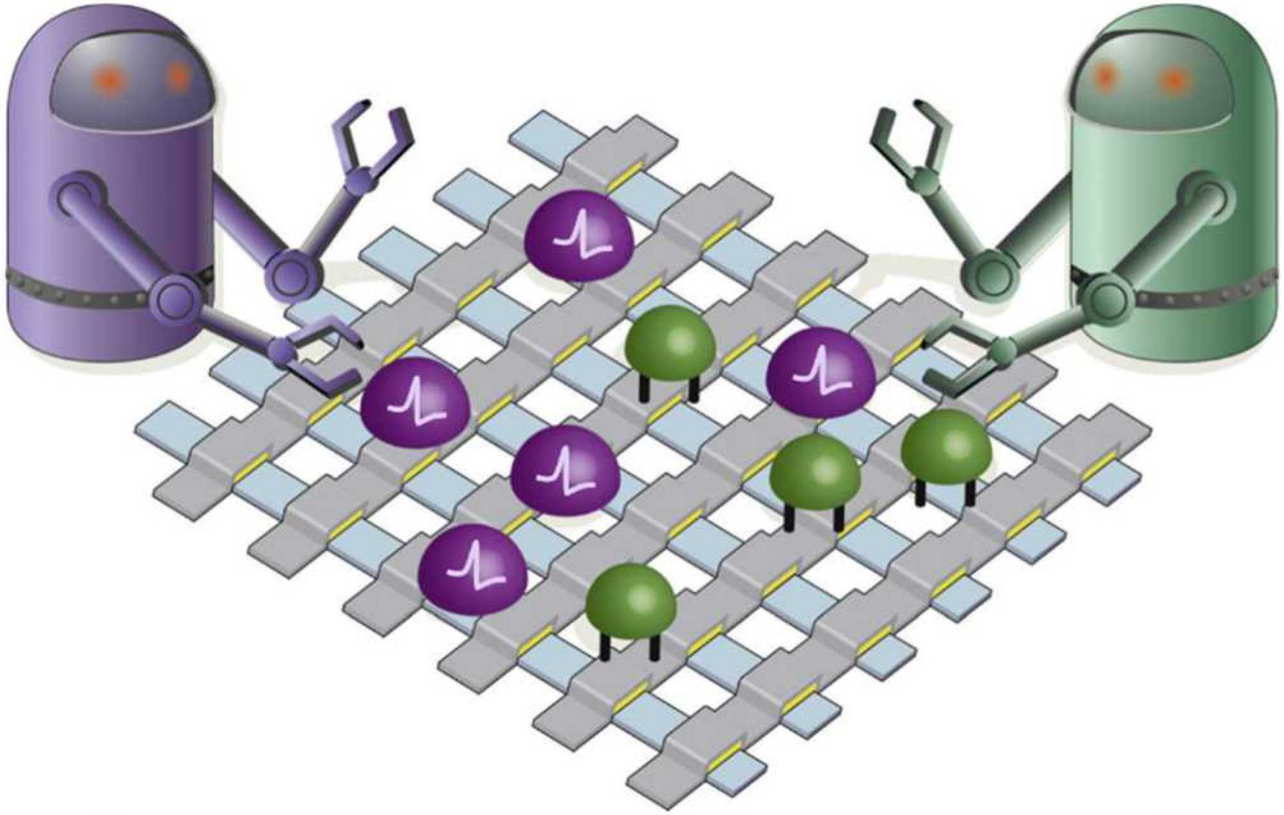
Divide each layer into K temporal-groups. Assign each group a relative-firing-time (accounting for the global minimum delay). Create a data structure that allows looking up the relative-firing-time based on the neuron-id.

When making a connection from one neuron to another, set the delay to be the relative-firing-time of the postsynaptic-neuron minus the relative-firing-time of the presynaptic-neuron.

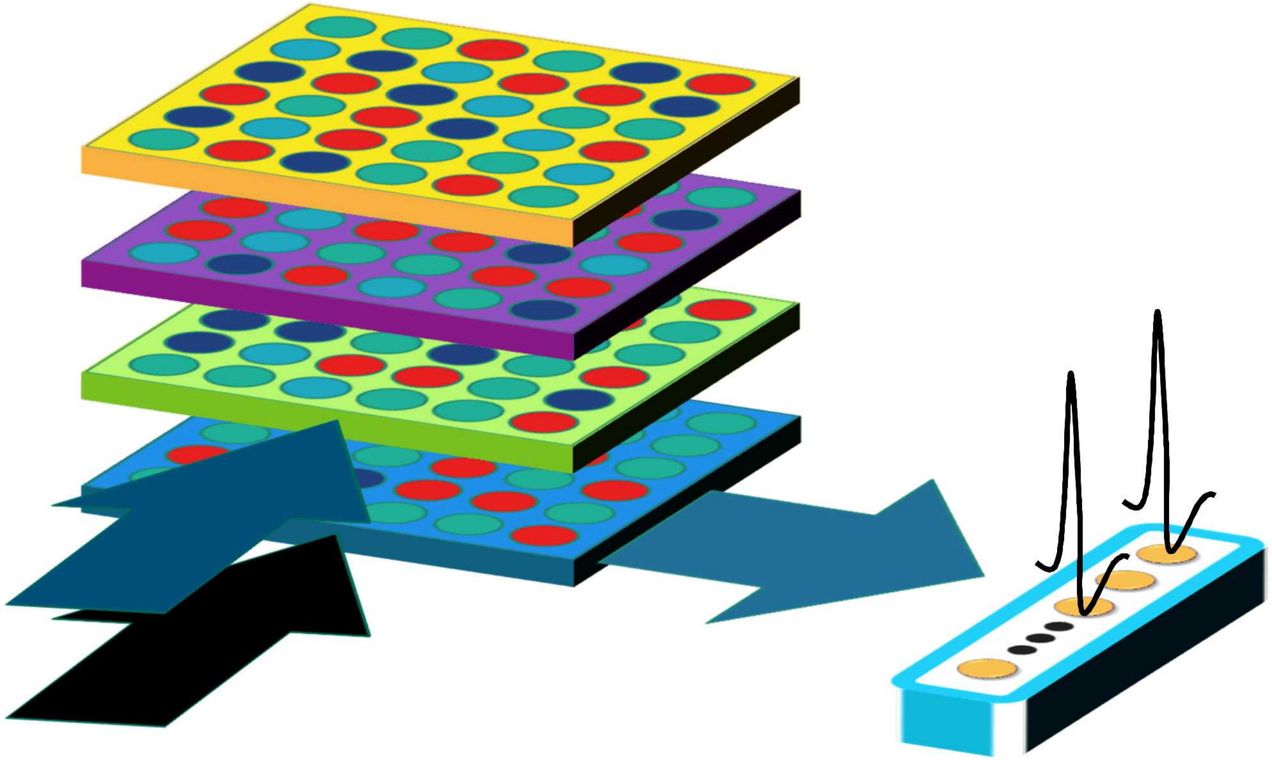
When scheduling the spikes of a spike-source array, schedule them at times offset from the sample presentation time based on the relative-firing-time of the source-neuron.

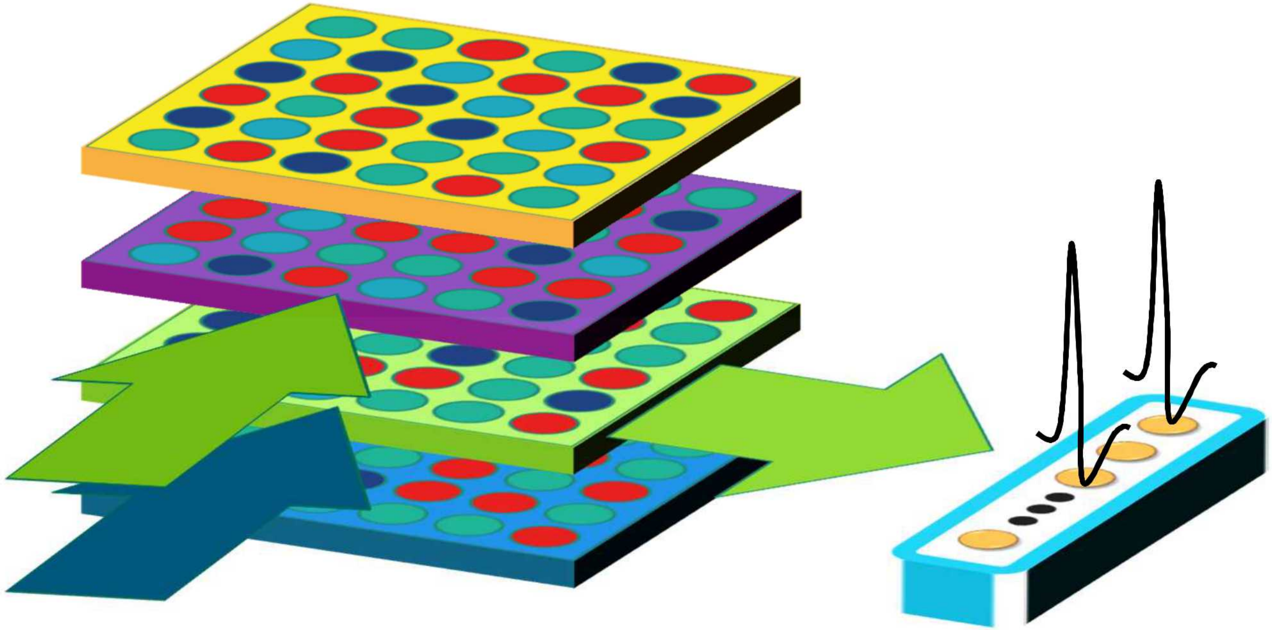
All the neurons for the output layer of a classifier should be placed in the same temporal-group, which simplifies reading output.

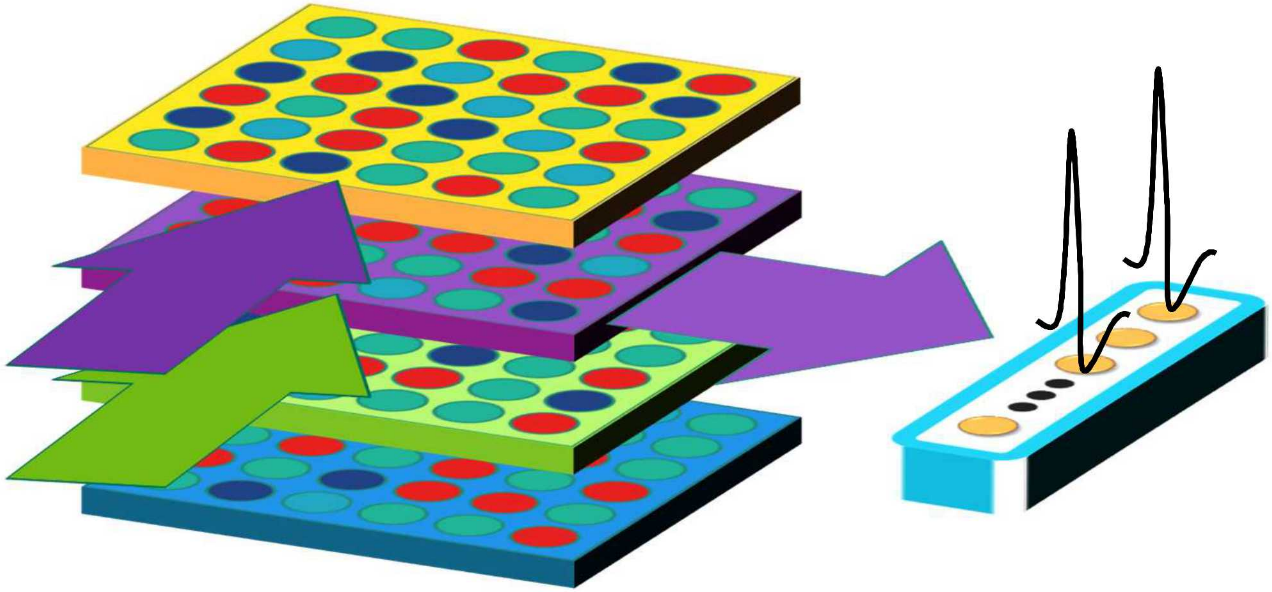
In the case of a semantic segmenter (for example), the output would also have to be temporally-staggered, and so some decoding would be necessary.



Green & Aimone, Nature Electronics 2019







Obviously this won’t work for everything

But it will work for a lot of potential applications

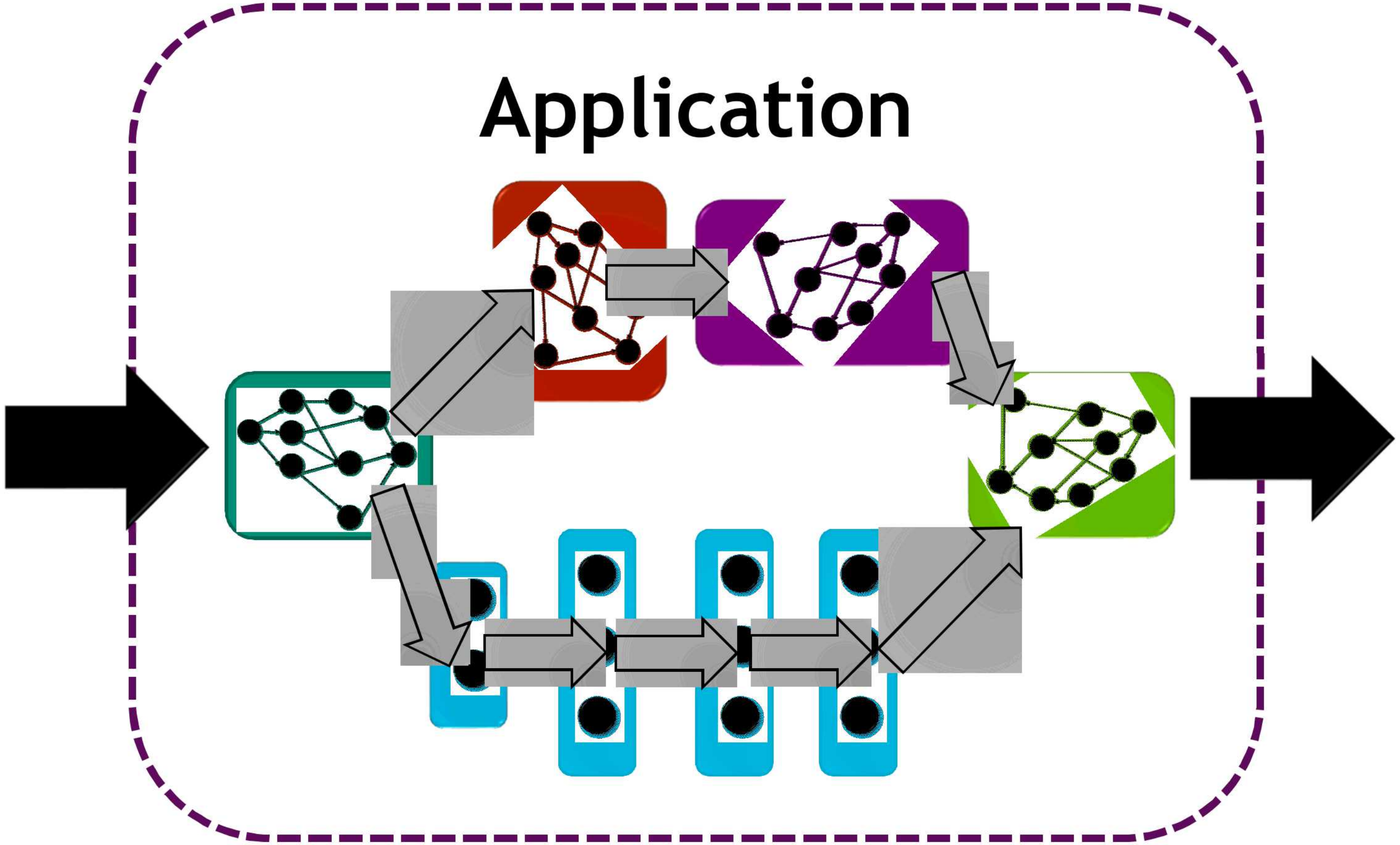
Machine Learning

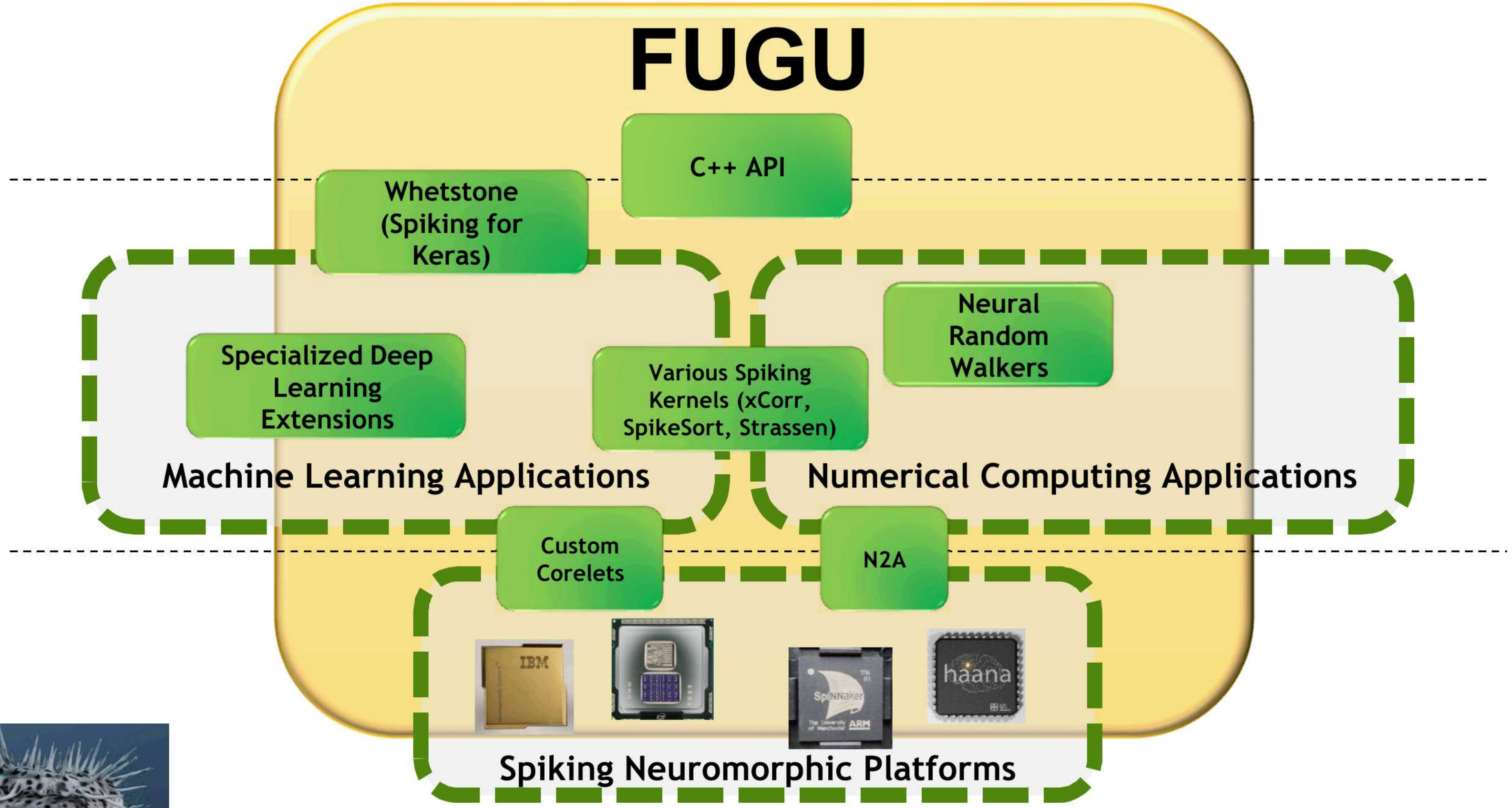
Whetstone

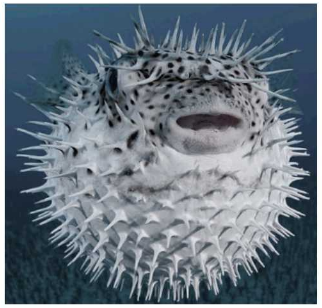
Convolutions

k-Nearest Neighbor

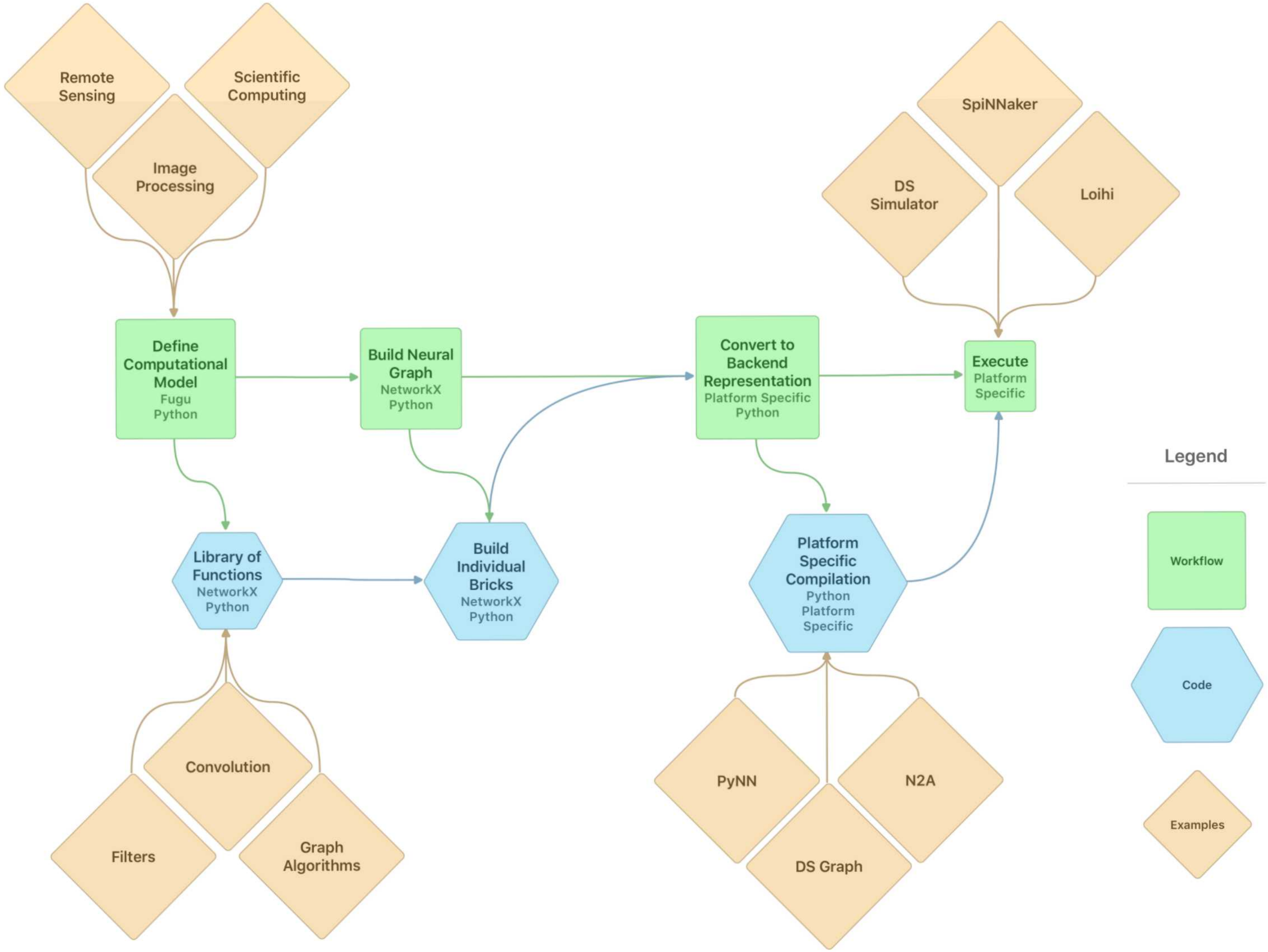
Support Vector Machines







Fugu = pufferfish (why? Pufferfish have spikes…)


Remote Sensing
Scientific Computing
Image Processing
Define Computational Model
Fugu
Python
Build Neural Graph
NetworkX
Python
Convert to Backend Representation
Platform Specific
Python
Execute
Platform Specific
SpiNNaker
DS Simulator
Loihi
Library of Functions
NetworkX
Python
Build Individual Bricks
NetworkX
Python
Platform Specific Compilation
Python
Platform Specific
Convolution
Filters
Graph Algorithms
PyNN
N2A
DS Graph
Legend
Workflow
Code
Examples

Thanks everyone for coming to NICE 2019!