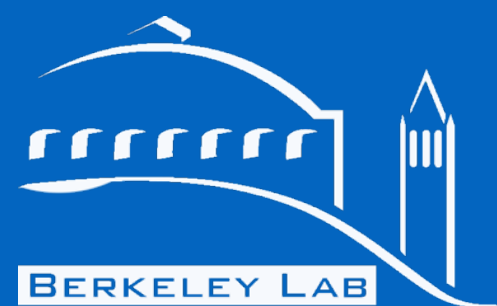


Kris Bouchard, Paolo Calafiura, David Clark, David
Donofrio, Maurice Garcia-Sciveres, Jesse Livezey

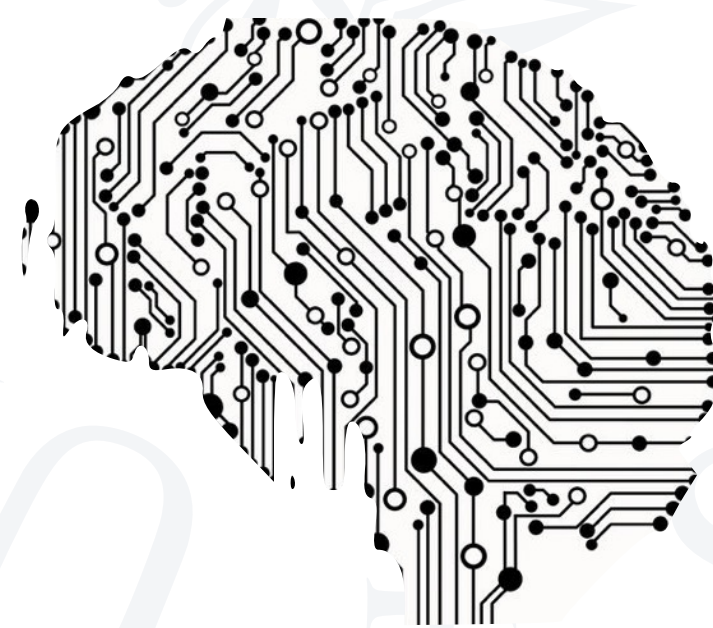


IBM's TrueNorth

Porting a common tracking algorithm to neuromorphic architectures

Rebecca Carney

rcarney@lbl.gov

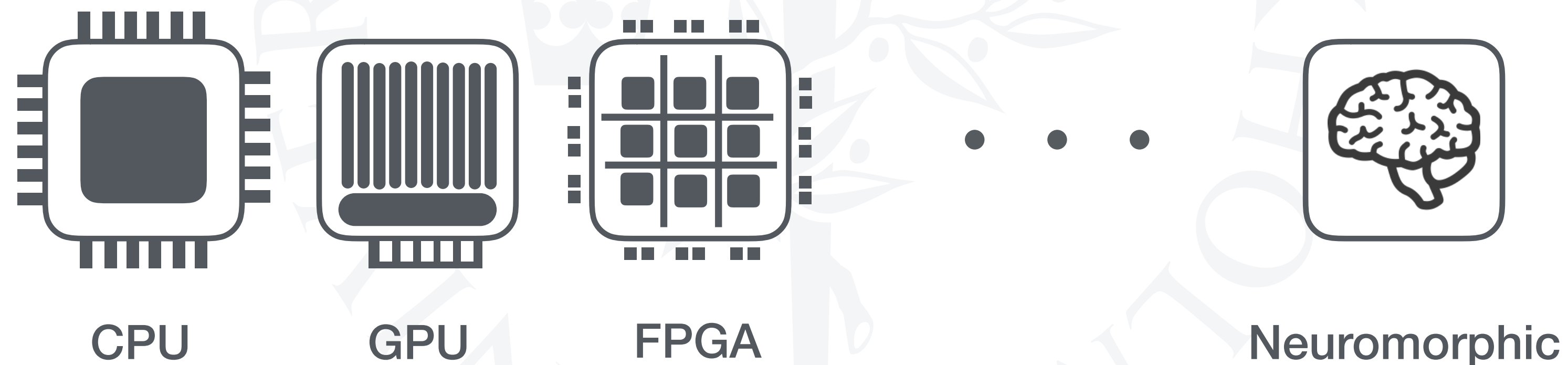


Introduction

With future colliders attaining never-before-reached luminosities and data rates, new methods of computation and new architectures on which to perform those computations are being explored.

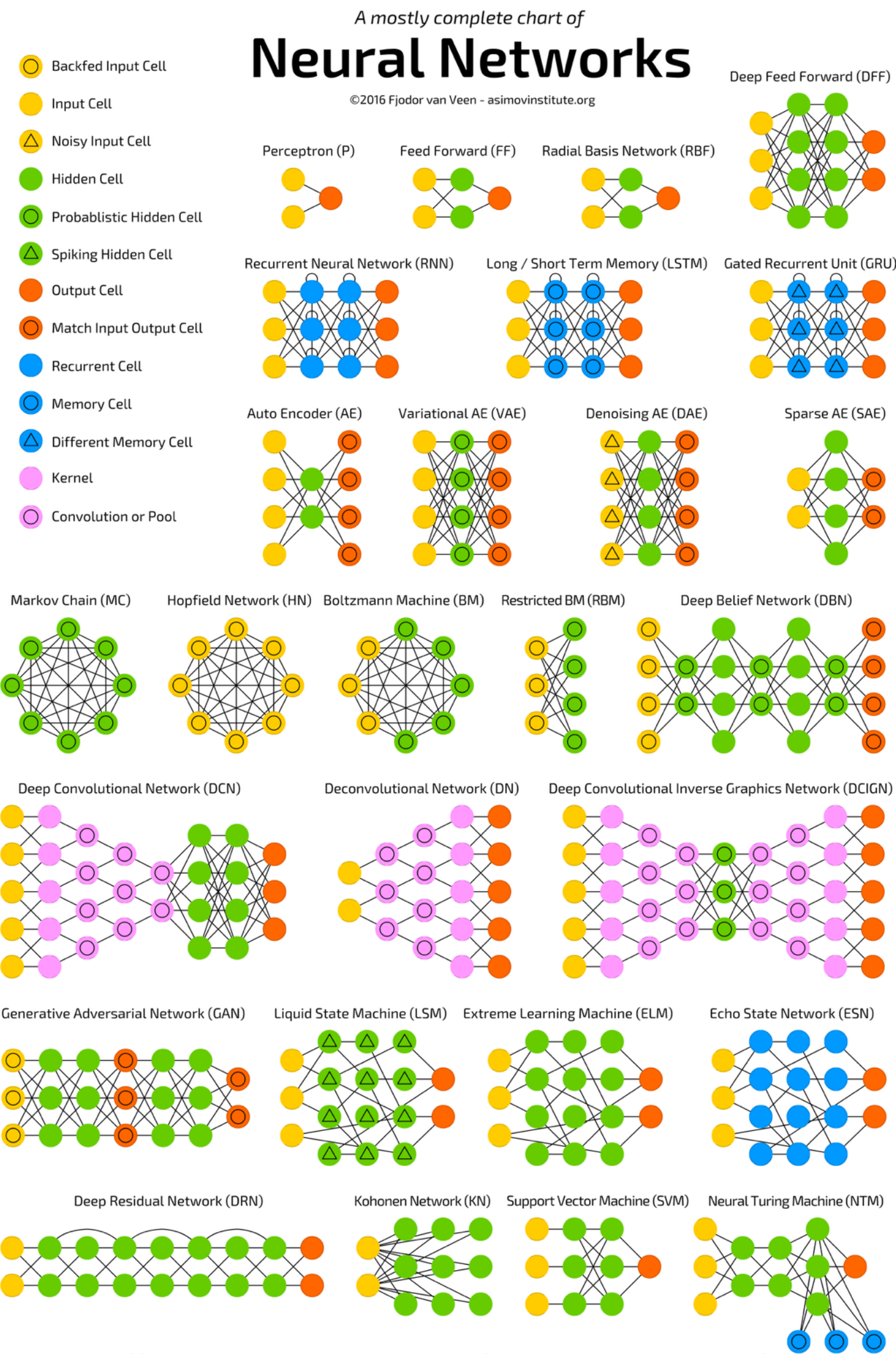
We've seen popular tracking algorithms like Kalman filters ported to GPU's and FPGA's. But what about neuromorphic computing?

IBM Research gave us access to their first neuromorphic chip but how easy is it to program? What are its limitations? And is it something we could feasibly add to our toolkit in the future?

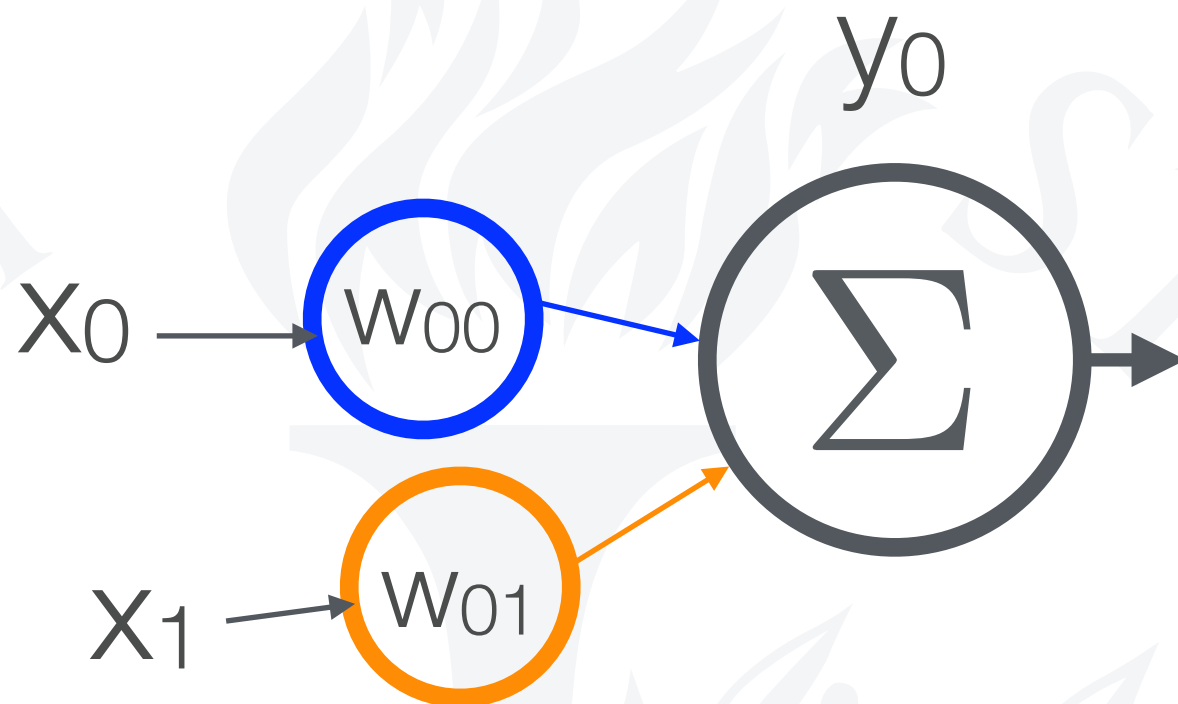


In this talk I will explore these questions through an implementation of a Kalman filter in TrueNorth.

Topology



Sum of weighted inputs



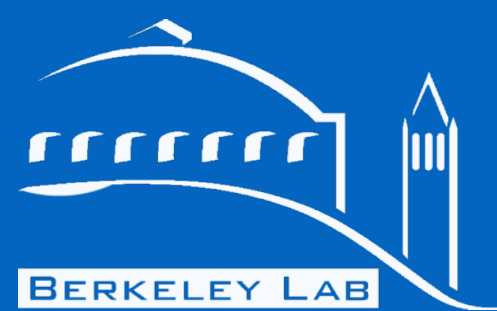
$$y_j = \sum_i w_{ij} x_i$$

- ANN's and neuromorphic networks have much in common:
 - A variety of topologies
 - Computational nodes called neurons that operate on the weighted sum of their inputs.
 - Apply some function to that sum to produce an output.

Activation (transfer) function

Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1 + x }$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU)[11]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Randomized leaky rectified linear unit (RReLU)[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [1]
Exponential linear unit (ELU)[13]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
S-shaped rectified linear activation unit (SReLU)[14]		$f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters.
Adaptive piecewise linear (APL) [15]		$f(x) = \max(0, x) + \sum_{s=1}^S a_s^* \max(0, -x + b_s^*)$
SoftPlus[16]		$f(x) = \ln(1 + e^x)$
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$

Arguably weight training is also a key characteristic of ANN's but is closely tied to topology and activation function.

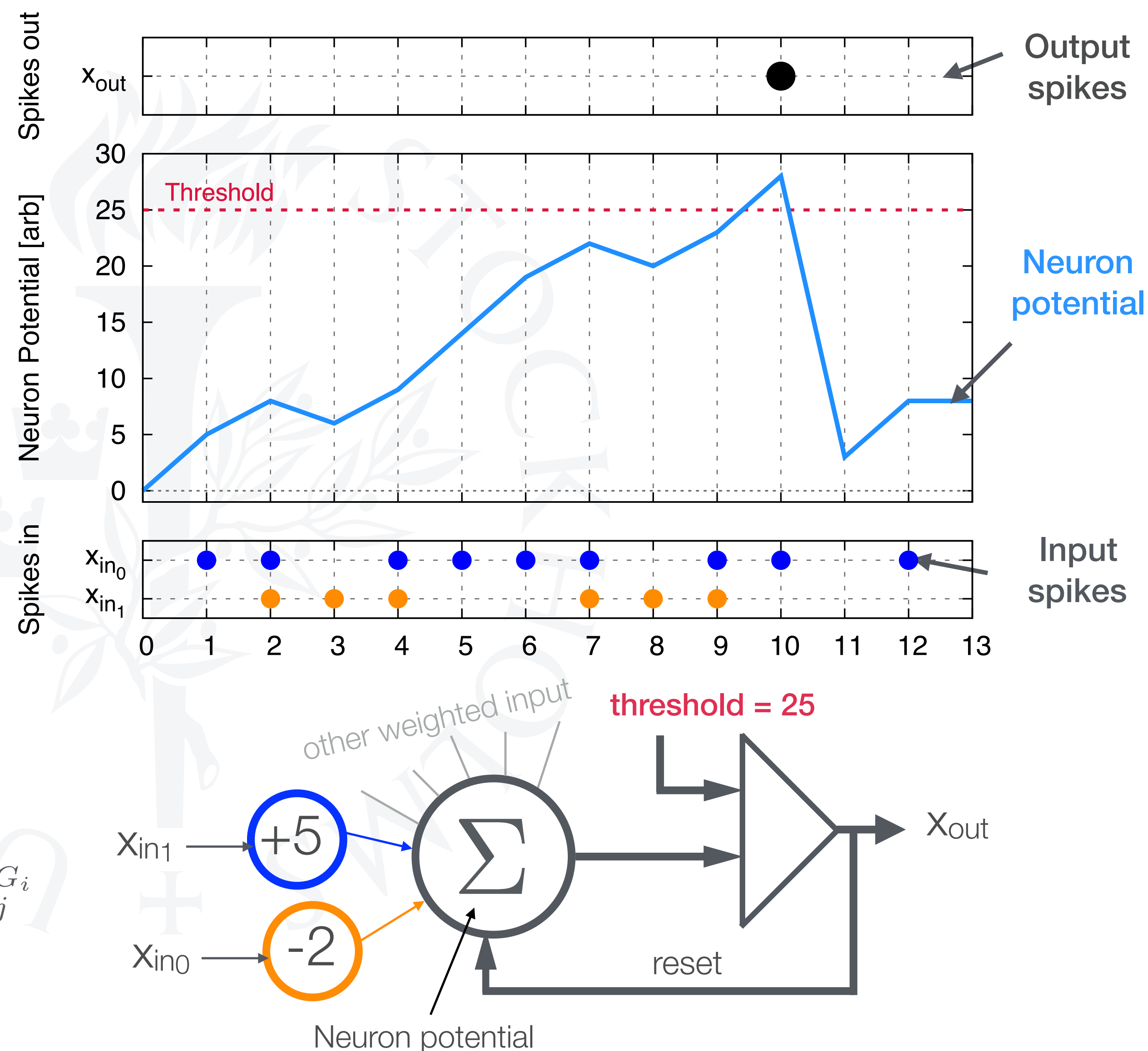


The key difference

Inherent time dependence between iterations of the network: the potential is stored until it goes over threshold and then a reset rule is applied.

- The key differences are:
 - Neuromorphic neurons do not have activation functions.
 - The weighted sum is thresholded¹.
 - Data is communicated by spikes.
 - Some neuromorphic neurons attempt to mimic biological neurons closely and apply shaping to their spikes or use analogue electronics to model different stages of neuron communication.

$$V_j(t) = V_j(t - 1) + \sum_{i=0}^{256} n_i(t) \times w_{i,j} \times s_j^{G_i}$$



¹ In the simplest models

- The dendrite is connected to the soma which is connected to the axon which is terminated in synapses.
- Synaptic transmission changes properties in the soma membrane potential.
- Integration of multiple inputs.
- At a threshold voltage ion channel will be forced open and an avalanche of charge carriers. This will cause a current pulse or spike.

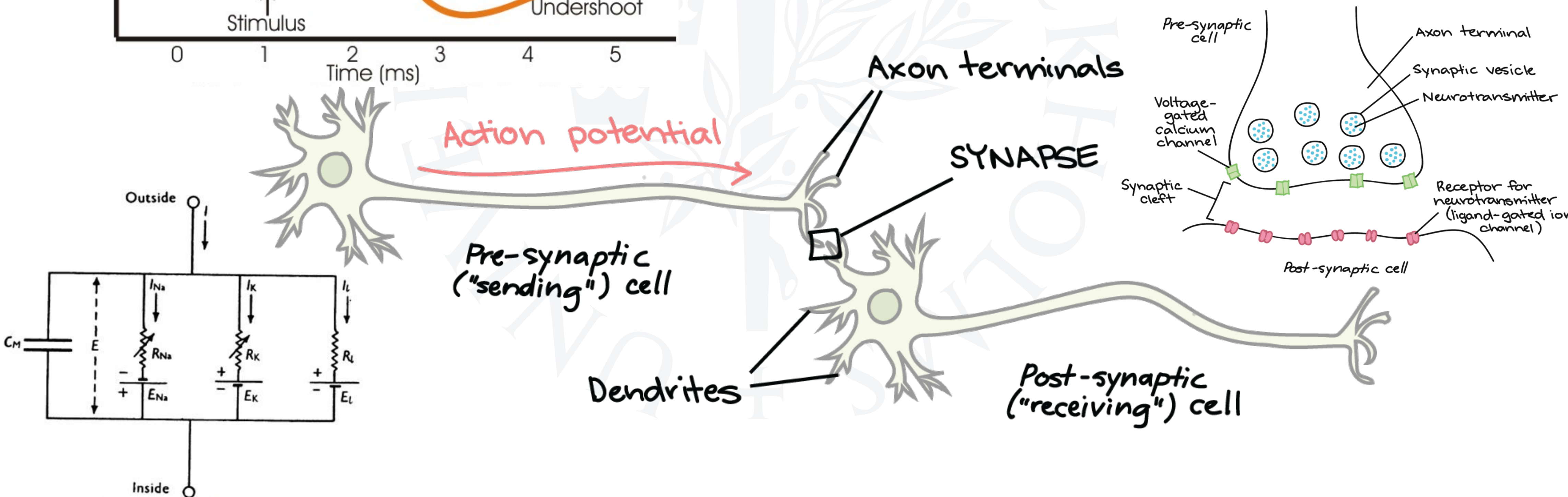
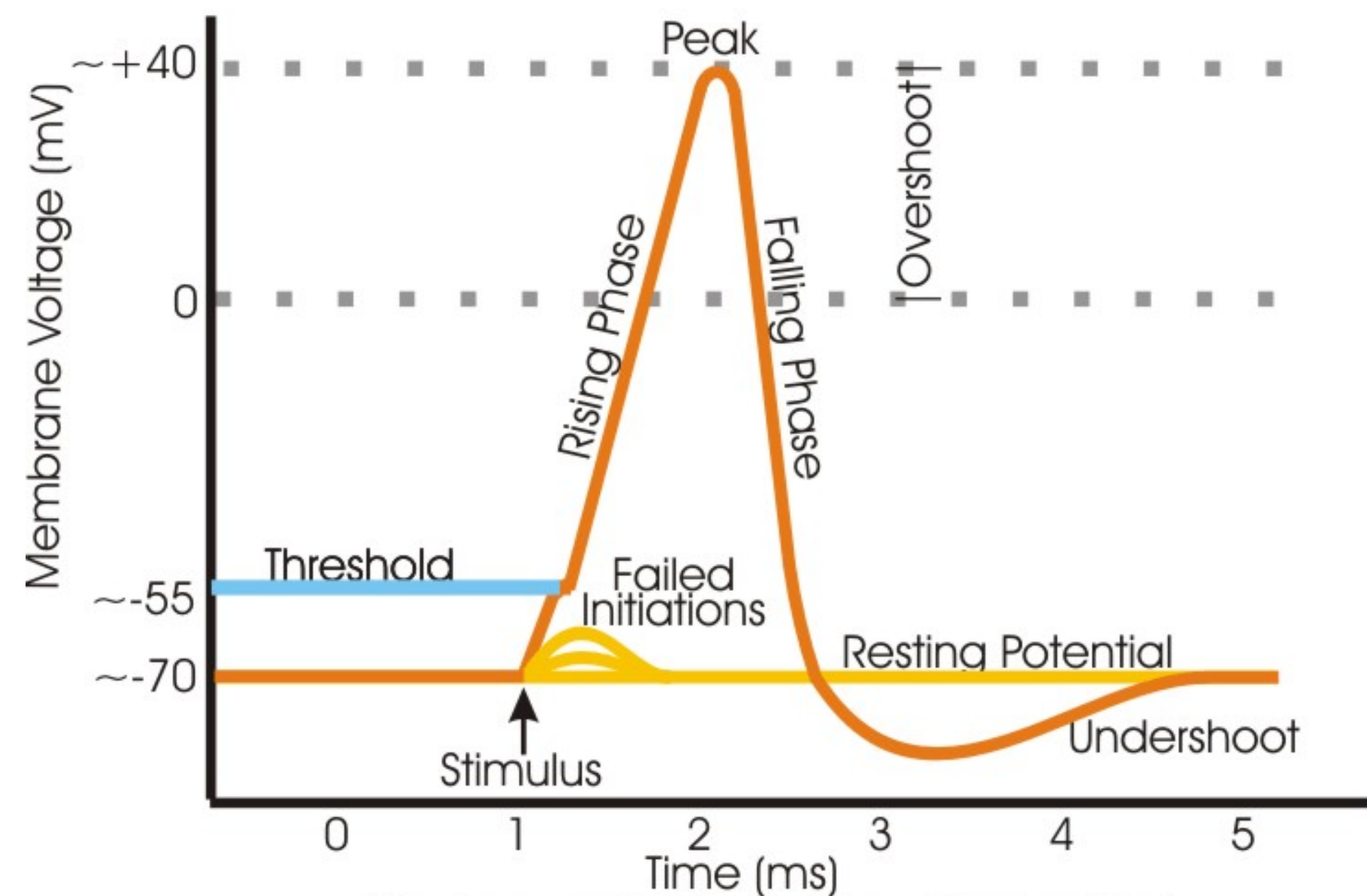
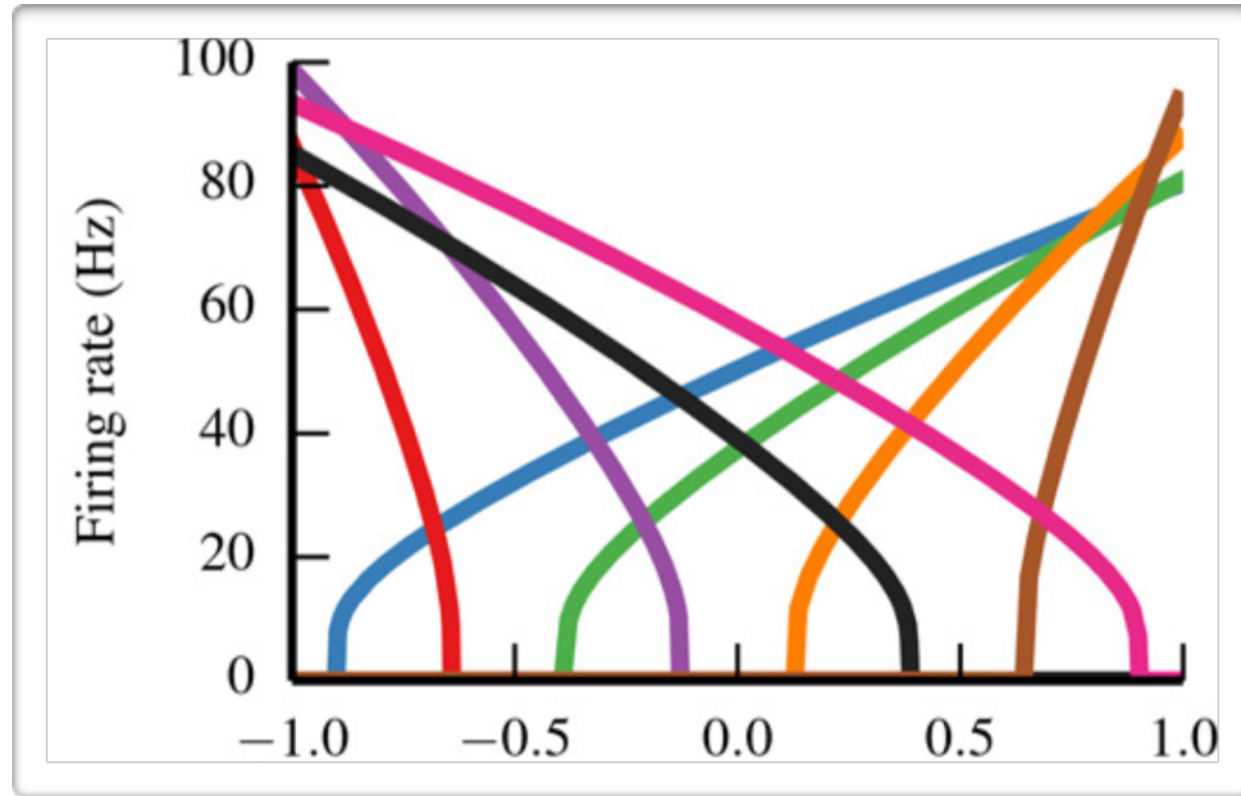
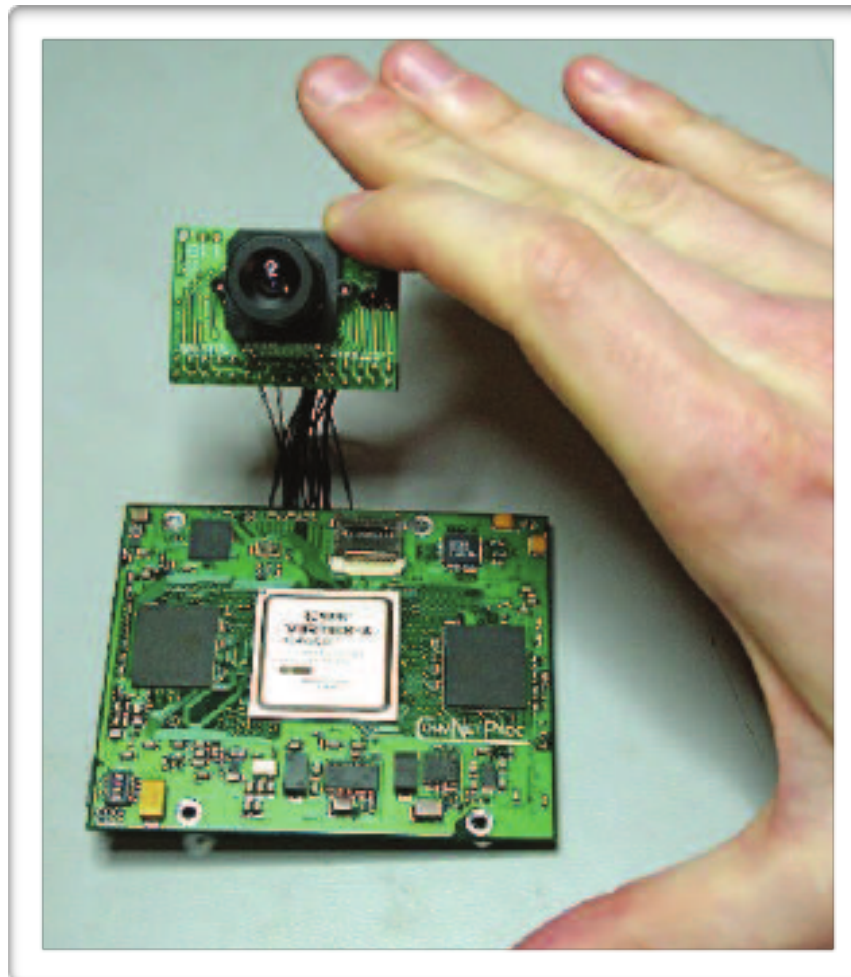


Figure 1: Circuit Model of an Axon [1]

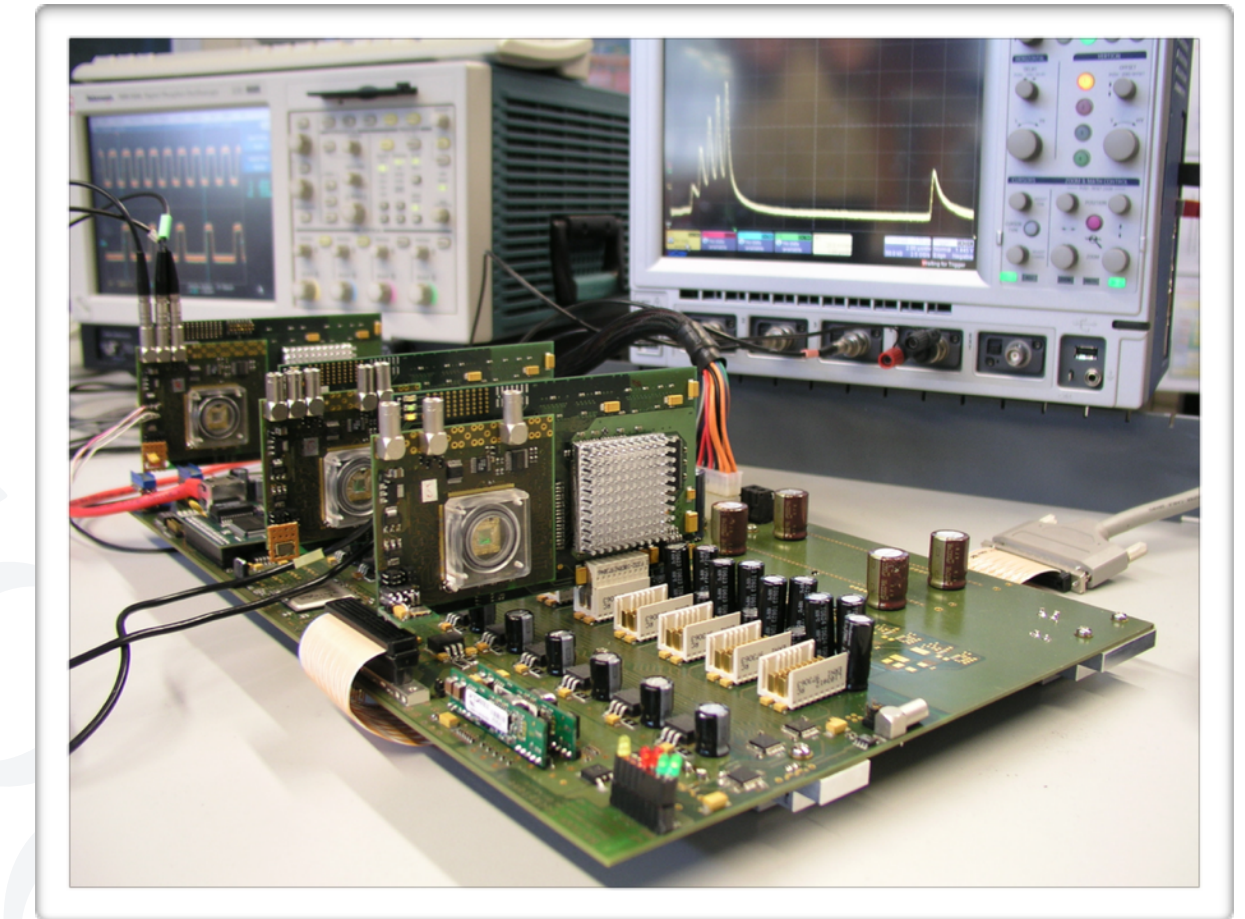


2013-18: [Neuromorph and Brainstorm](#) @ Stanford



2008: [NeuFlow](#)
@ NYU (J. LeCunn)

2016: [Spinnaker 0.5M core machine](#)
@ HBP/Manchester

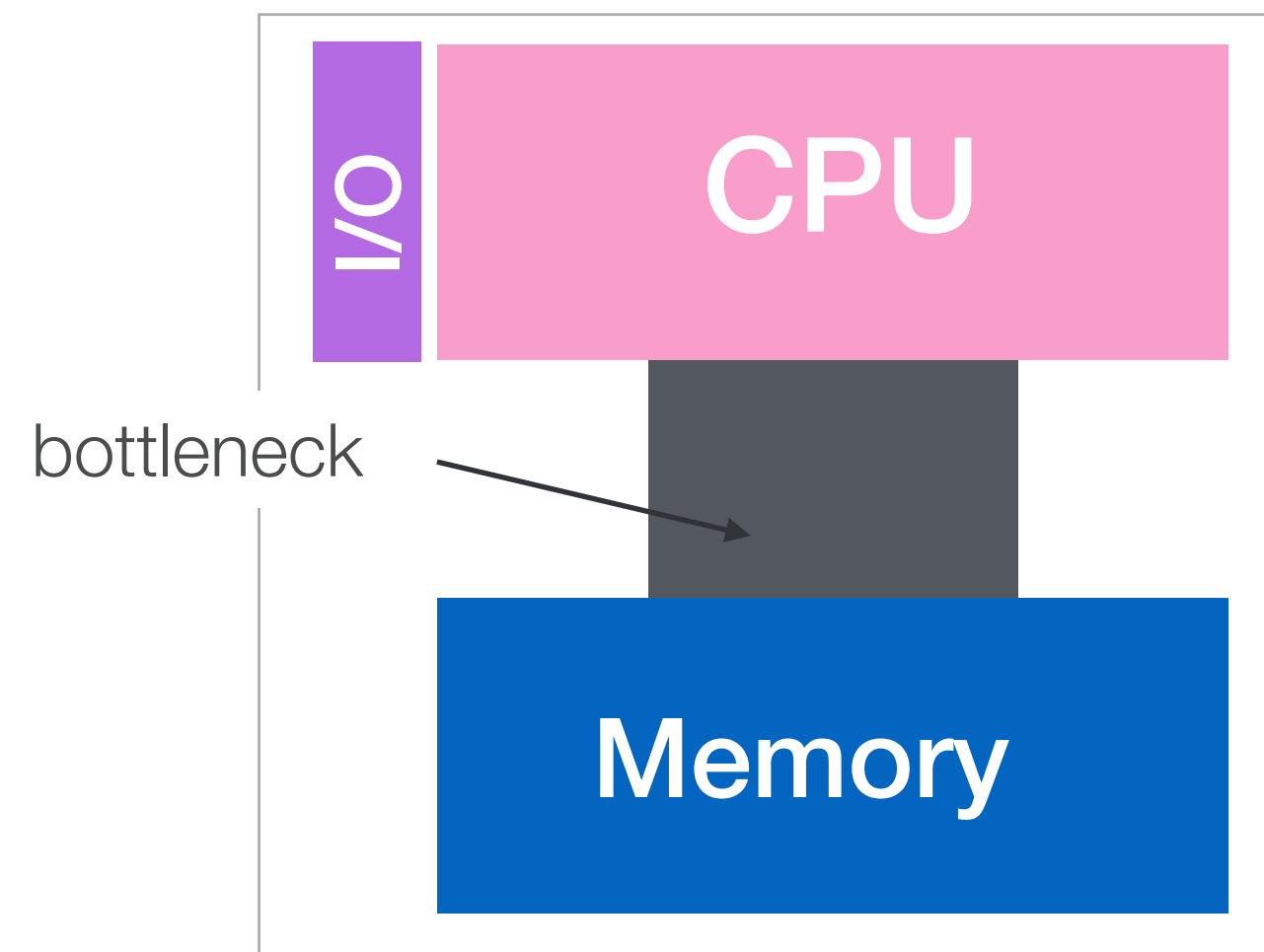


2011: [Spikey](#) @ Heidelberg

1. **Spinnaker**: scalable, low-power units
 2. **Brainstorm**: NEF and populations of neurons
 3. **Spikey/BrainScaleS**: analogue spikes, neuroplasticity, 10k speedup.
- NeuFlow**: Conv. neural network on-chip. Not really neuromorphic.. but an example of NN hardware!

[More neuron-inspired computing at NICE 2016 \[link\]](#)

[NICE 2017 was in early March '17 @ IBM Almaden, slides will appear in a few weeks](#)

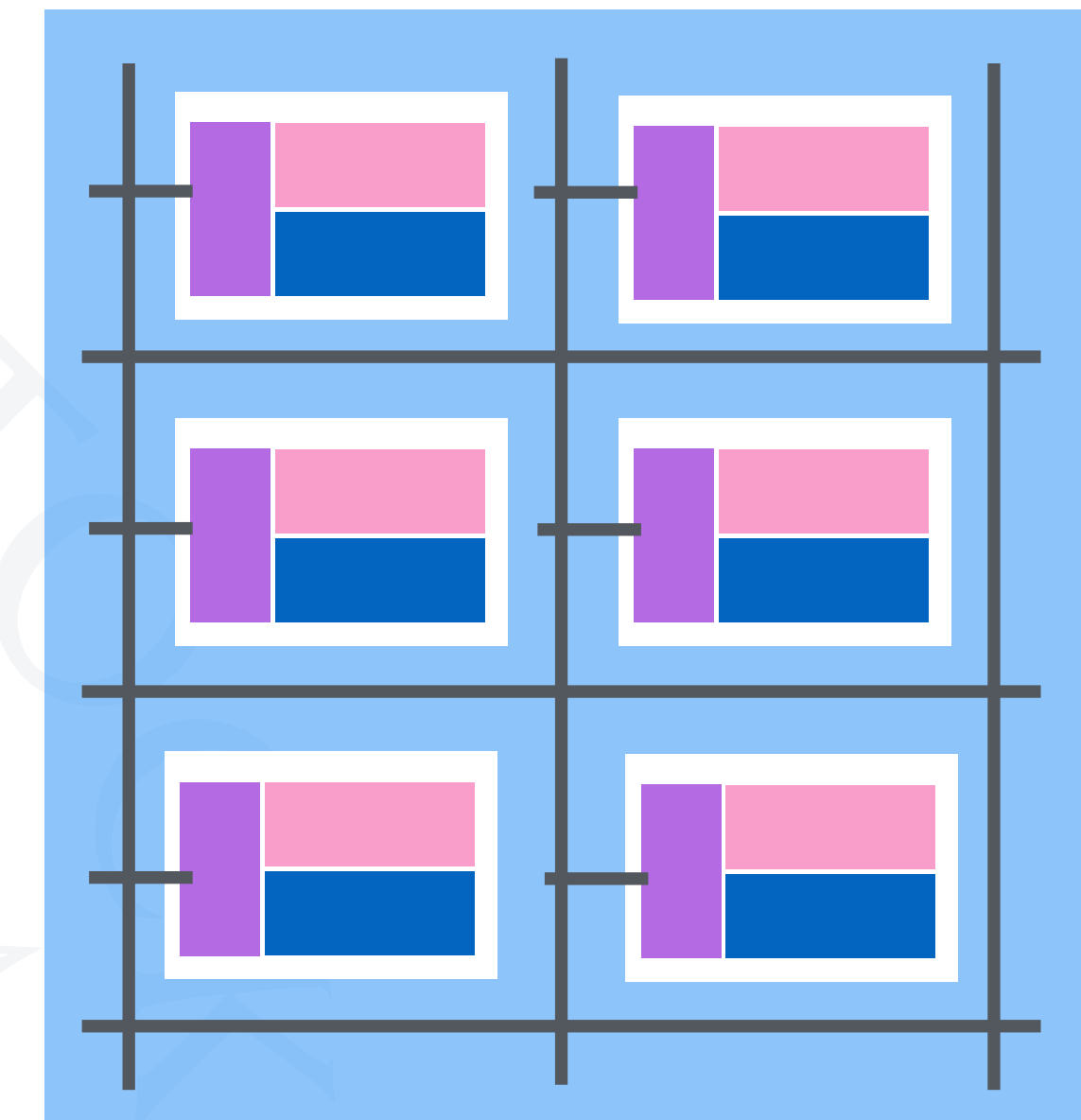


Traditional von Neumann architecture

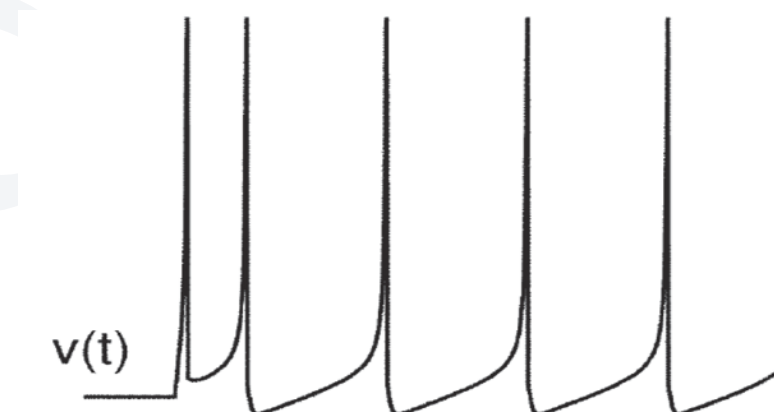
01000011

Binary n-bit words

Instruction set



Distributed, parallel architecture

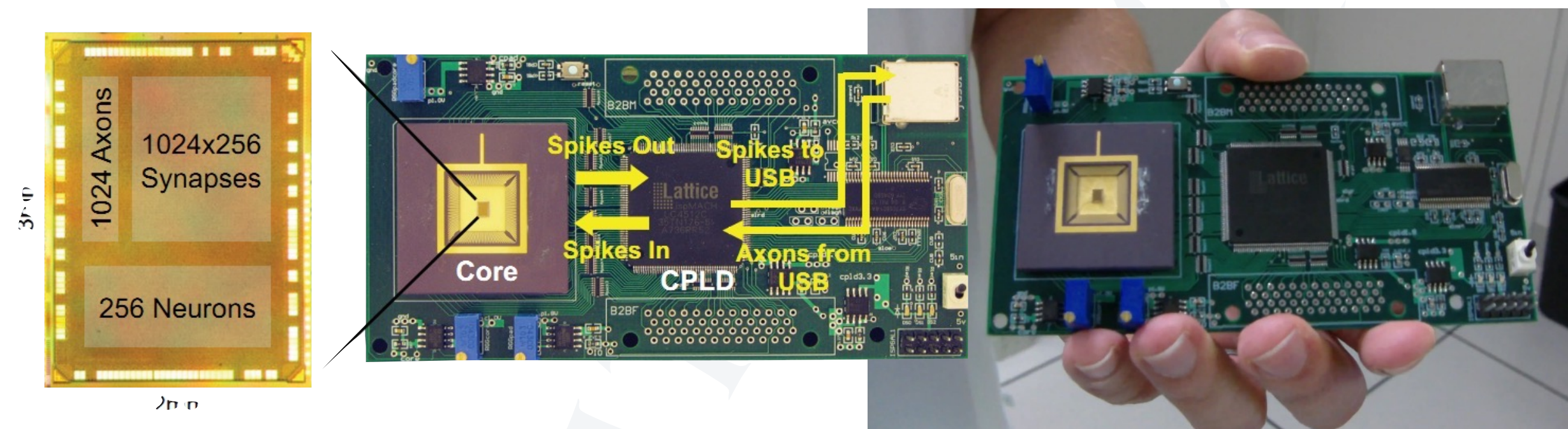


Temporally separated spikes

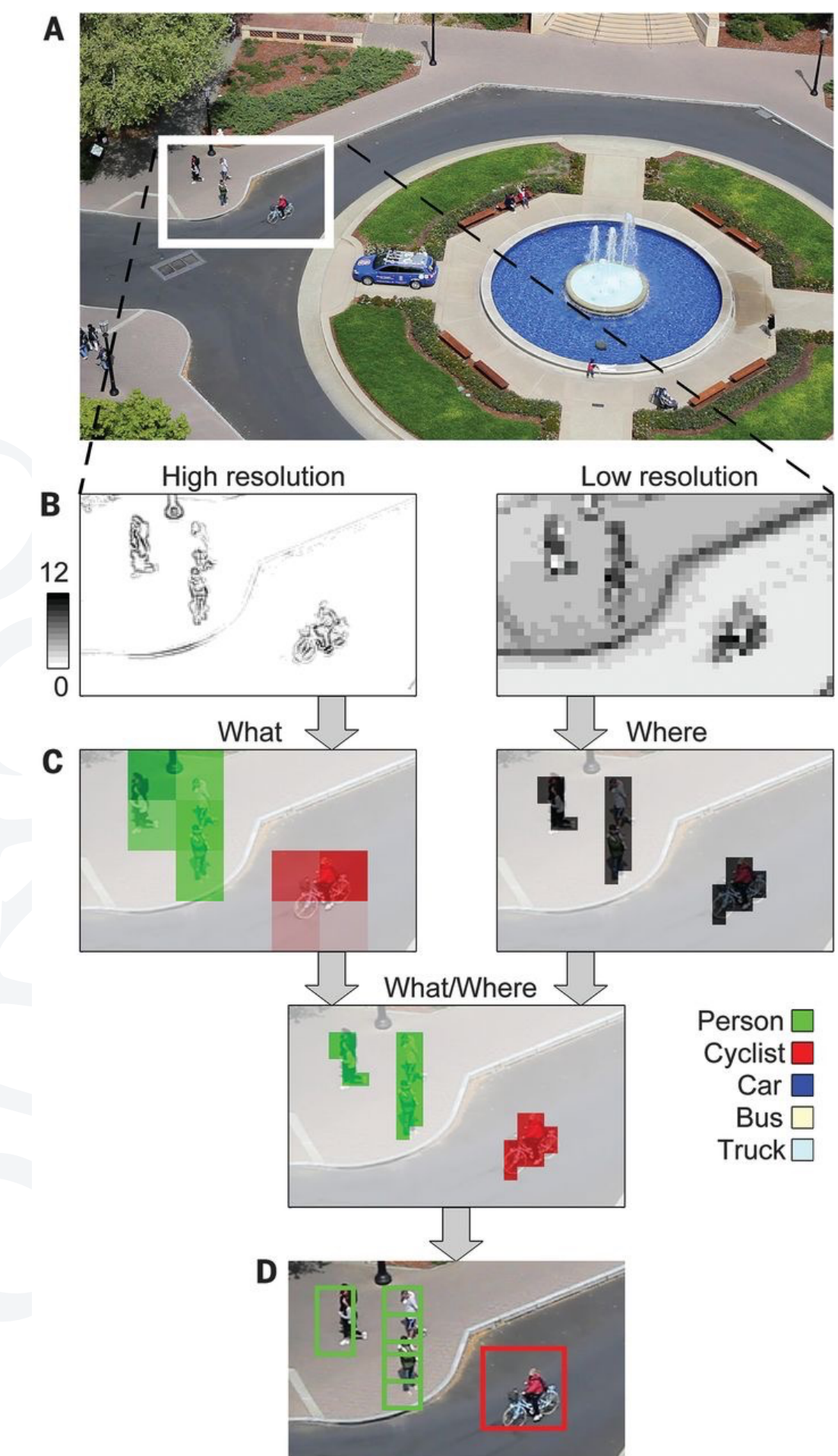
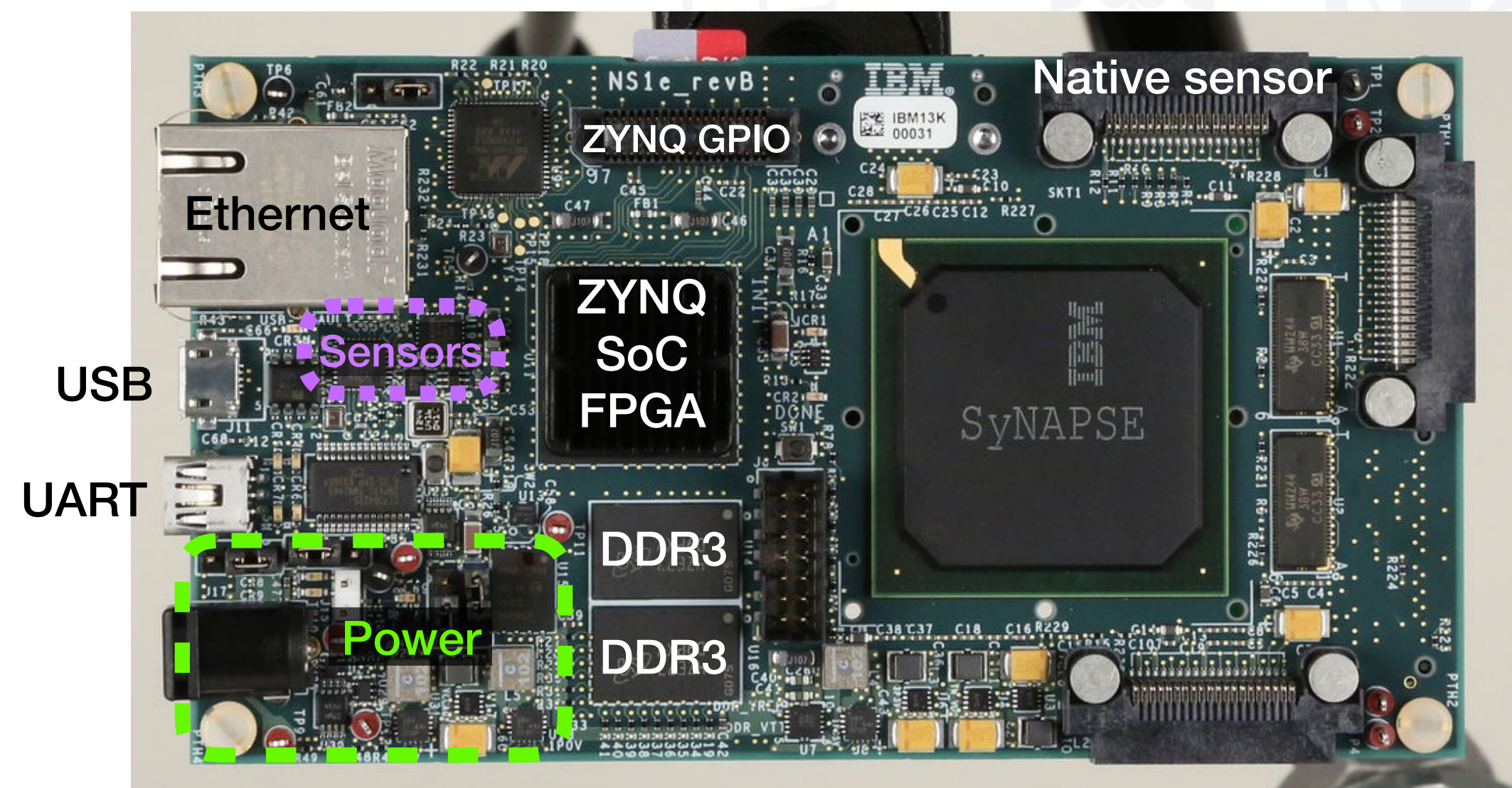
Connectivity and weight model file



2011 - a single core demonstrator



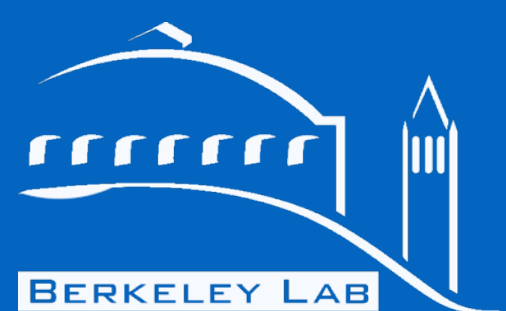
2014/15 - TrueNorth chip opened up to select institutes

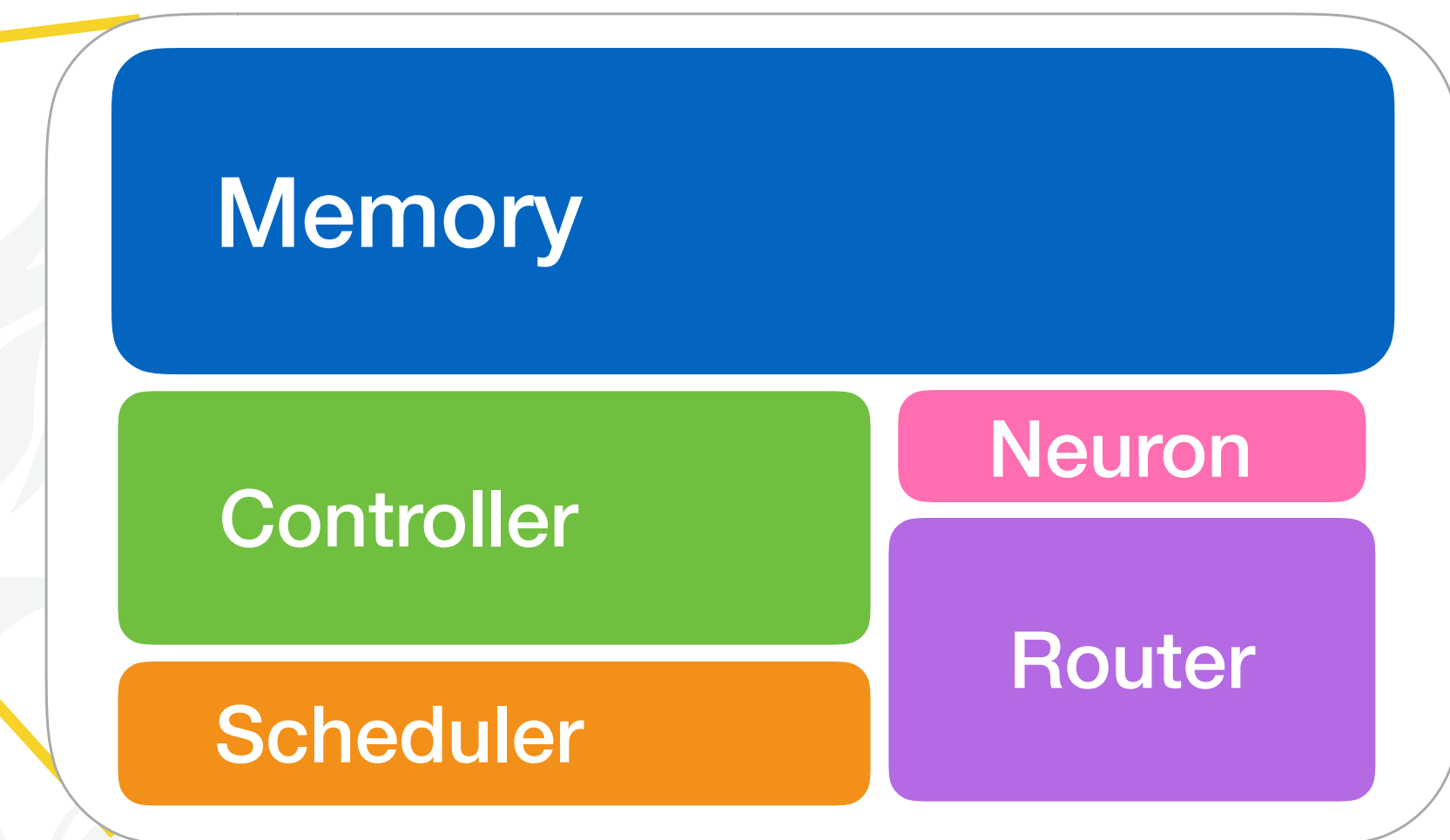
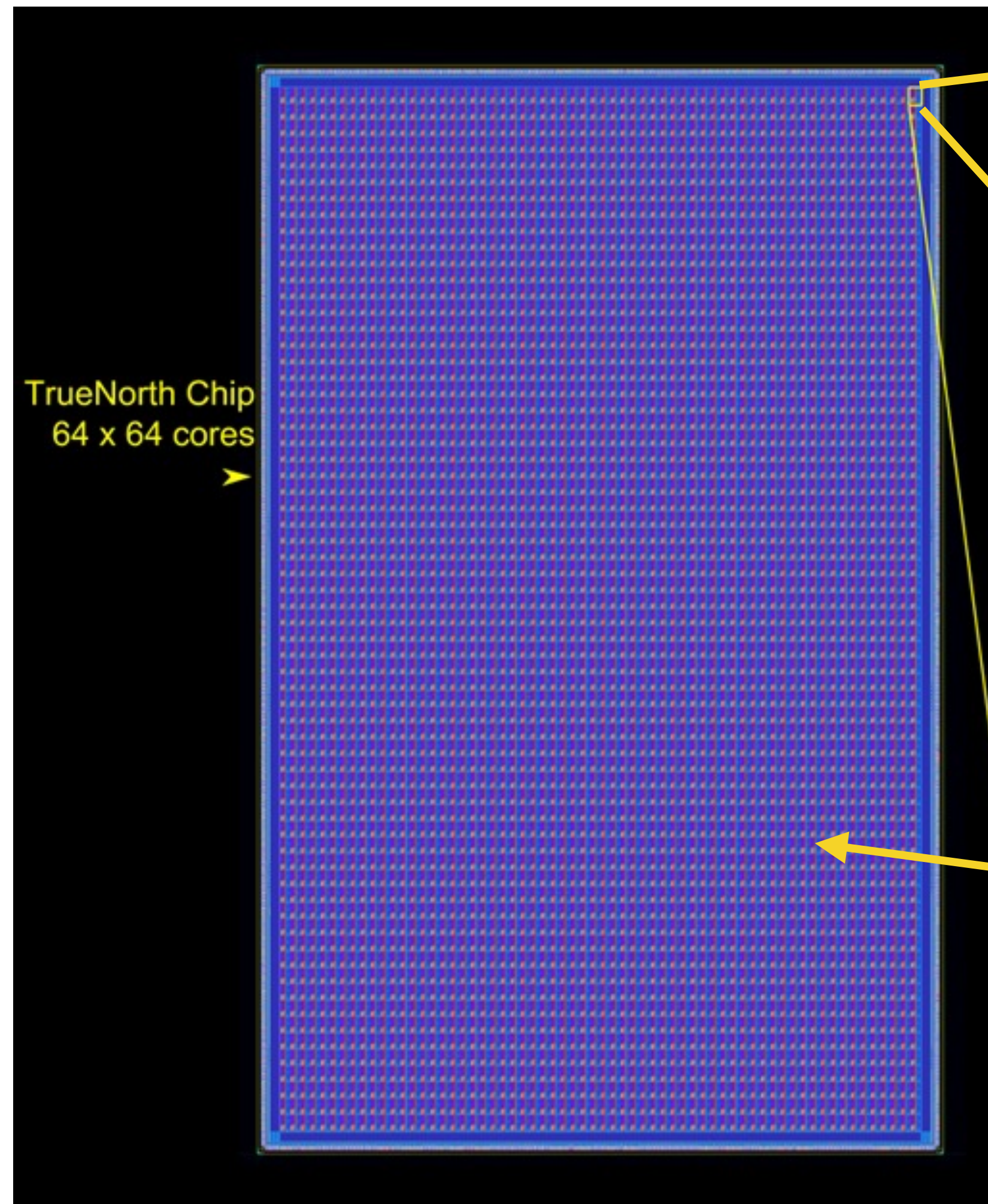
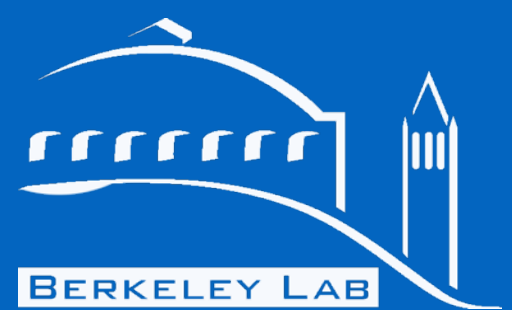


- Project is from DARPA SyNAPSE: metric of a one million neuron brain-inspired processor.
- Not biologically motivated! Low power, scalability, and connectivity were the motivating factors.
- (Unofficial) summary of project [here](#).

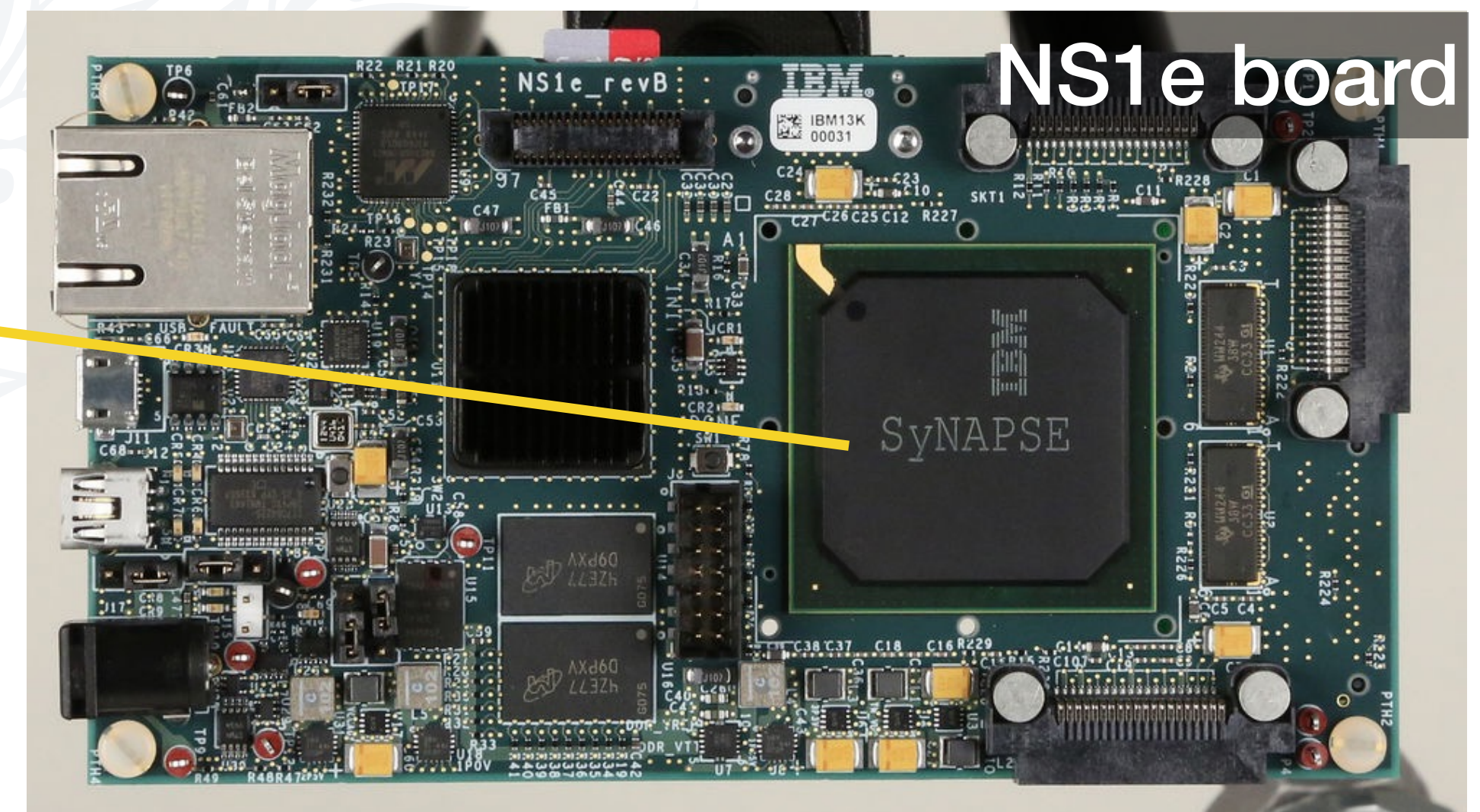


Neovision tower challenge

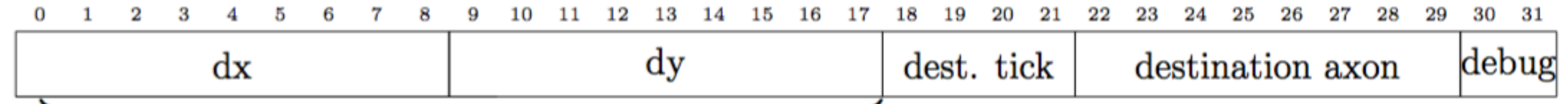
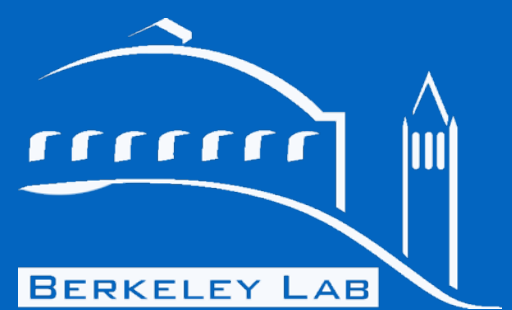




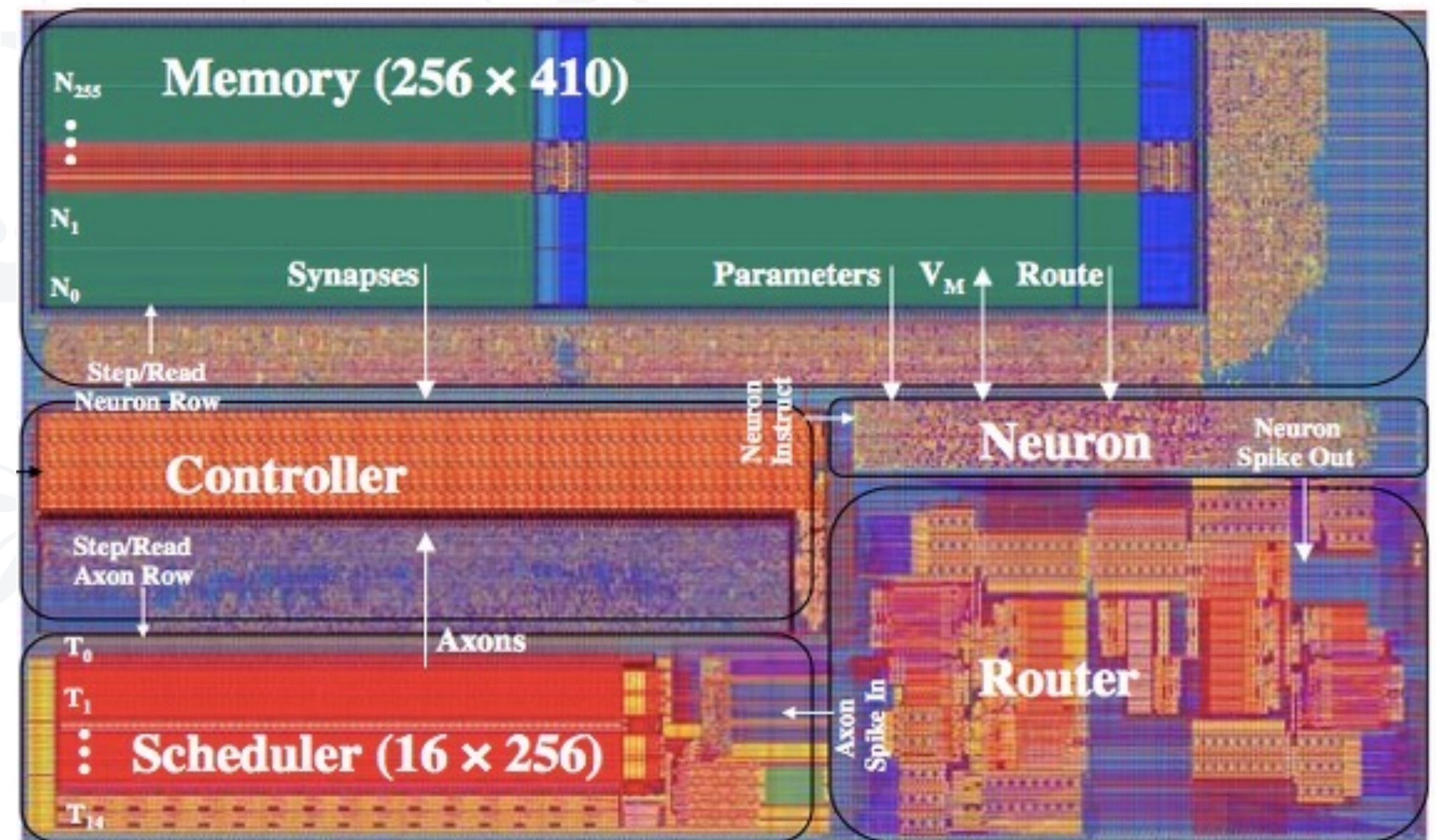
256 neurons
and axons
per core

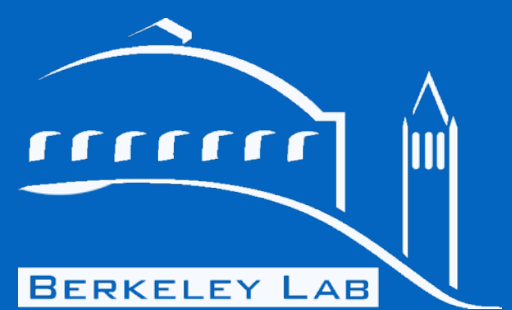


- Low power regime, therefore: 10's-100's of mW
- 1kHz synchronization point (read: clock)
- Fully digital, simulator has 1-to-1 correspondence with chip



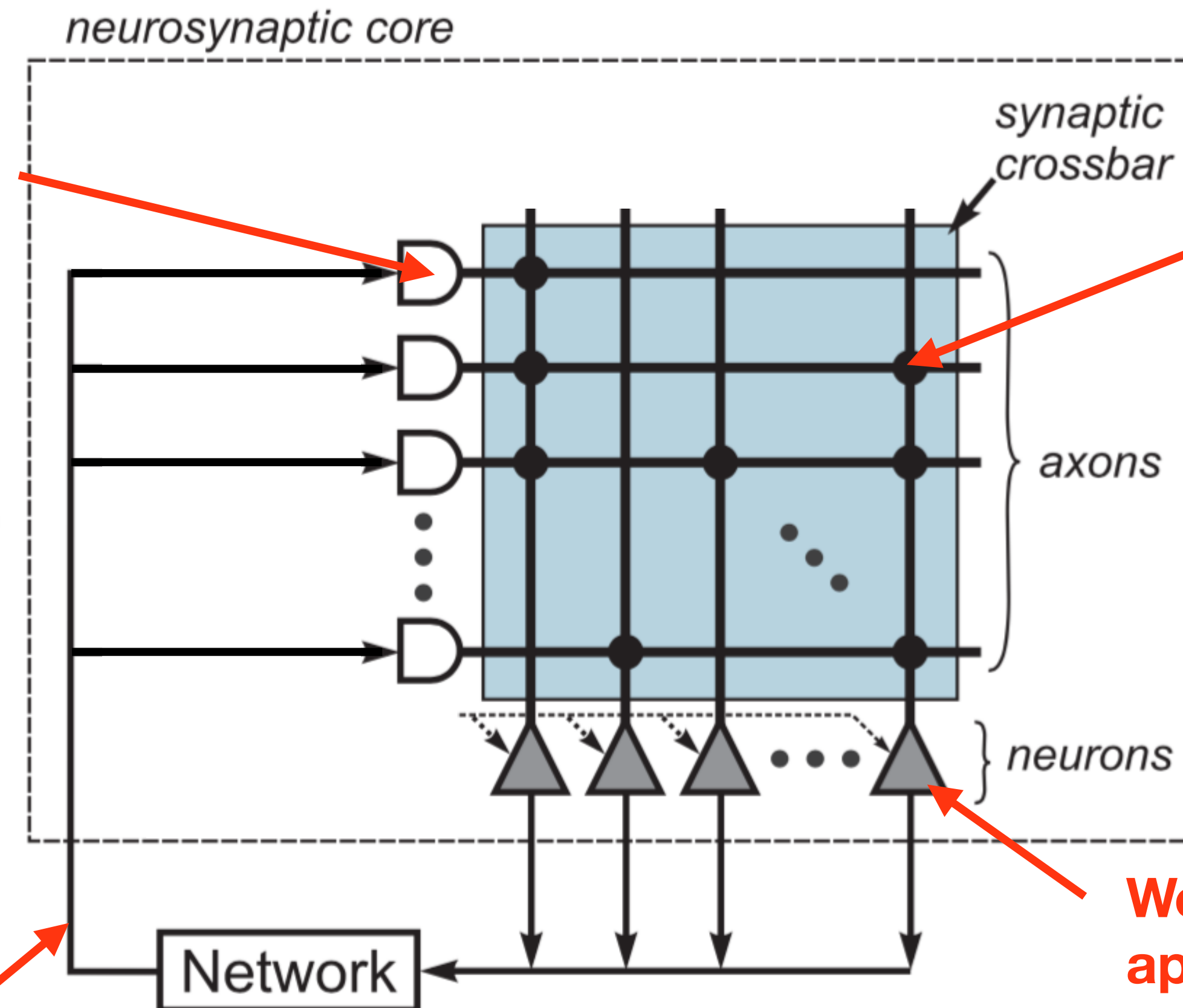
- TrueNorth has a [mix of synchronous and asynchronous \(event driven\) logic](#).
- To enforce global order an externally generated 'tick' is used. This is not a traditional clock but its edge is used to [trigger](#) the controller to begin processing spikes in the core.
- Communication between cores happens asynchronously by a 4-phase handshake.
 - Each spike carries a 4-bit 'destination tick' (i.e. you can add a delay to the timing, assuming the spike packet arrives when it is supposed to.)
 - Scheduler then stores the spikes in the correct part of the memory before their allotted tick.





Axons are the xbar input. Also provide a label $\in \{1,2,3,4\}$

Synapses $\in \{0,1\}$



Weights and threshold applied in neuron. A signed 8-bit weight can be saved per axon *label*.

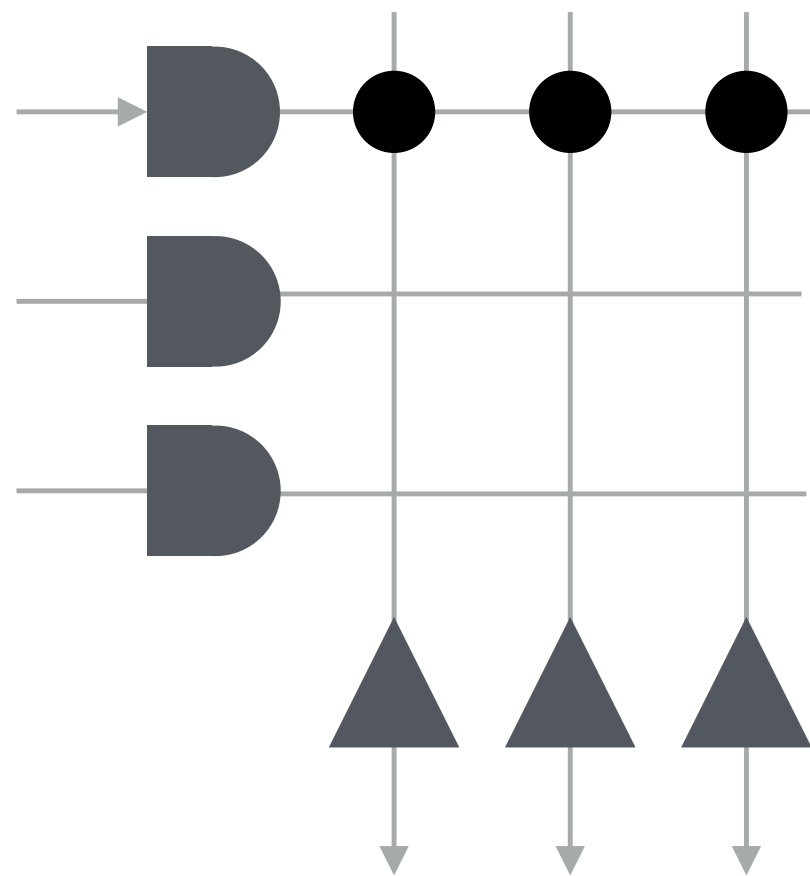
Connections are between 1 neuron and 1 axon. Fan out/in requires multiple neurons/axons.

Axons = input + label

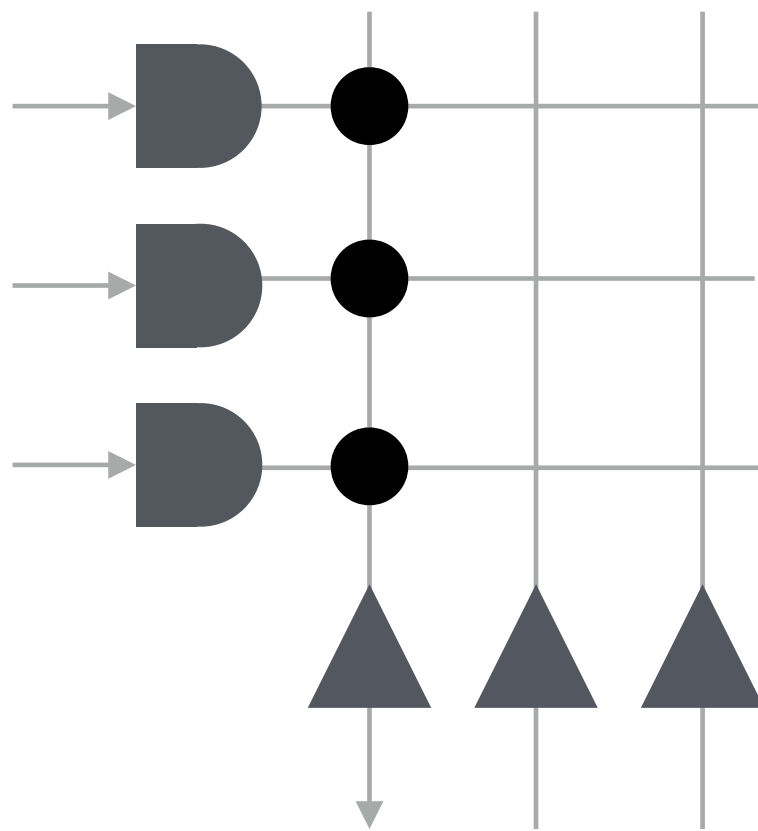
Neuron = computation + output

Synapses = connecting axons and neurons

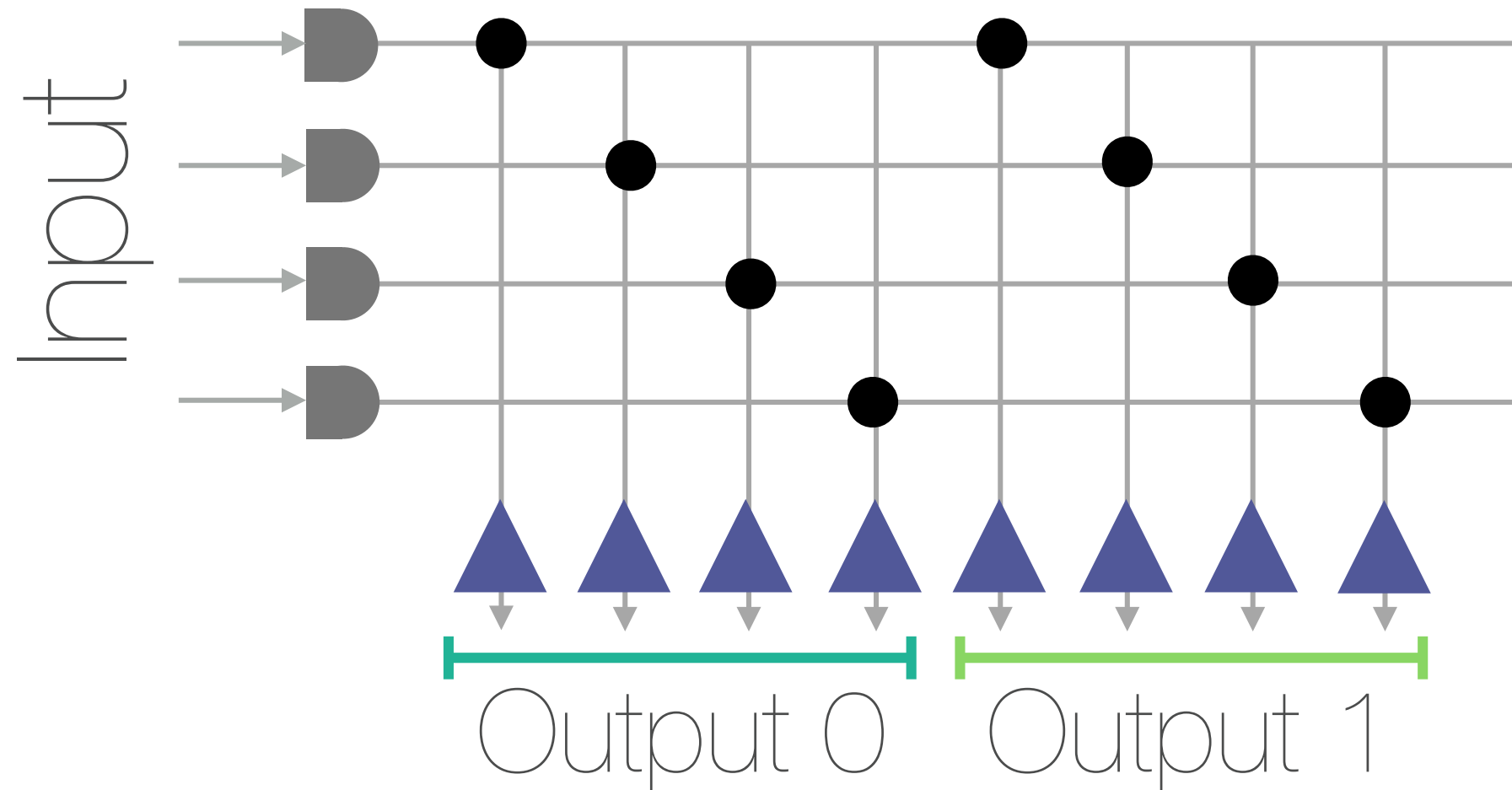
Crossbar = synapse frame



Many neurons connected to one axon

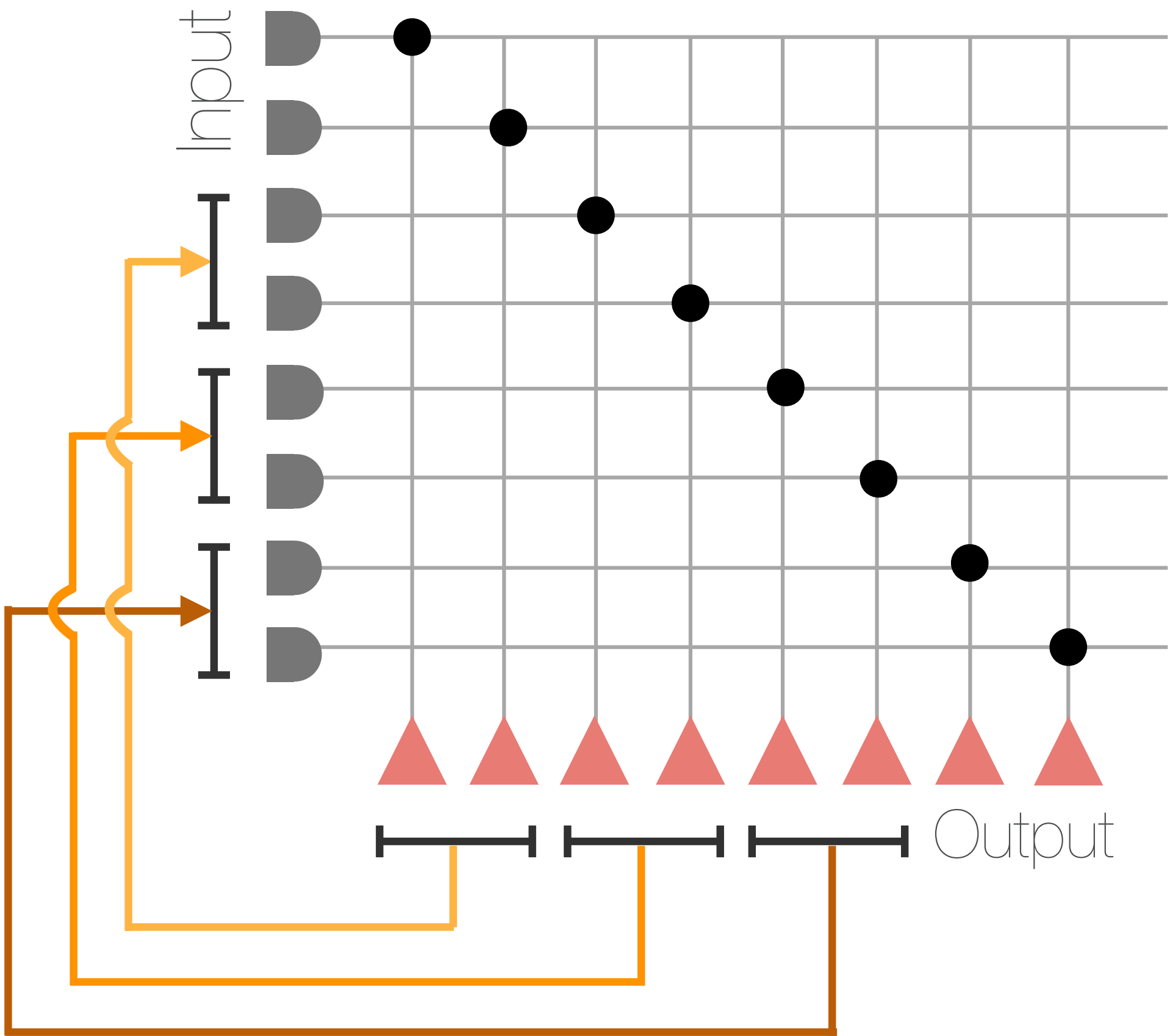


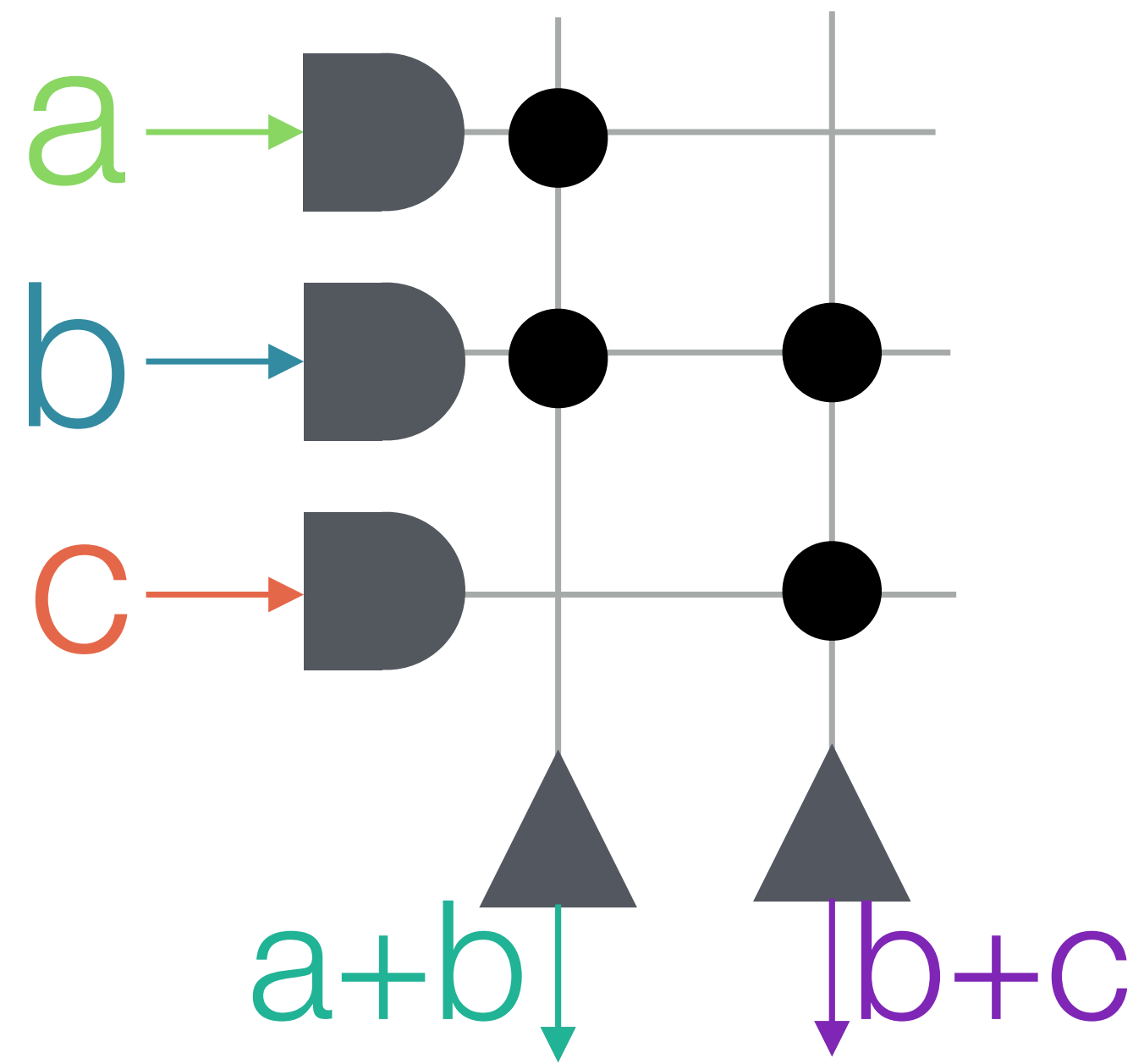
Many axons connected to one neuron



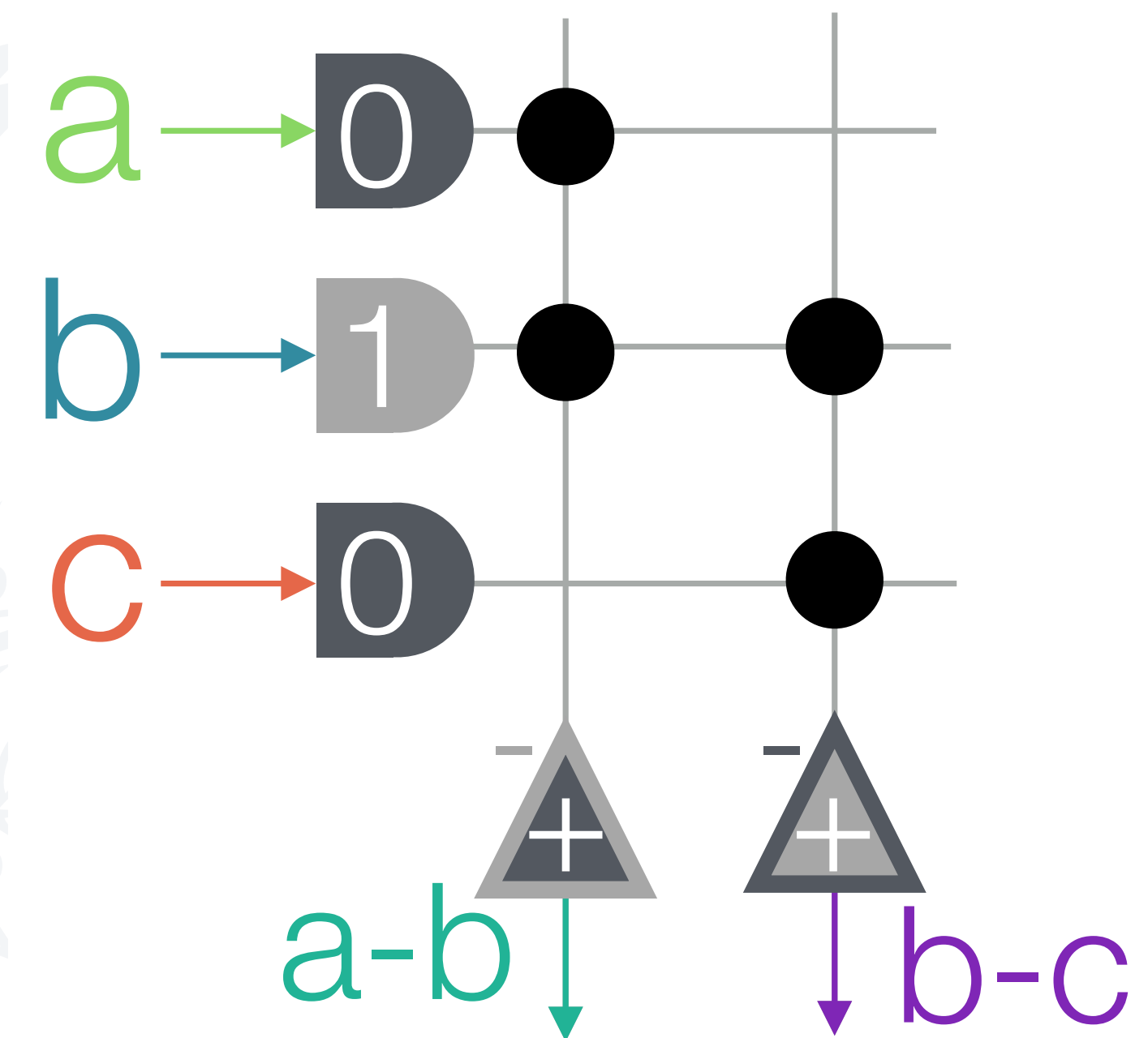
(1:2) splitter for 4-pin connector

Programmable delay unit for 2-pin connector.
Min delay = 4 ticks, max delay = 46 ticks



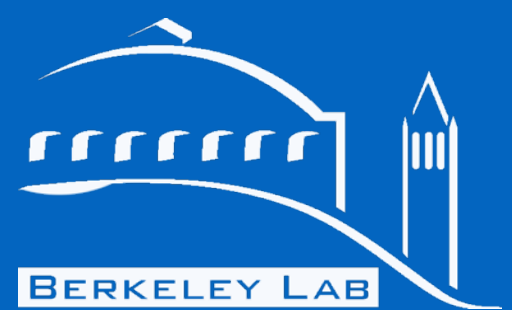


Simple addition

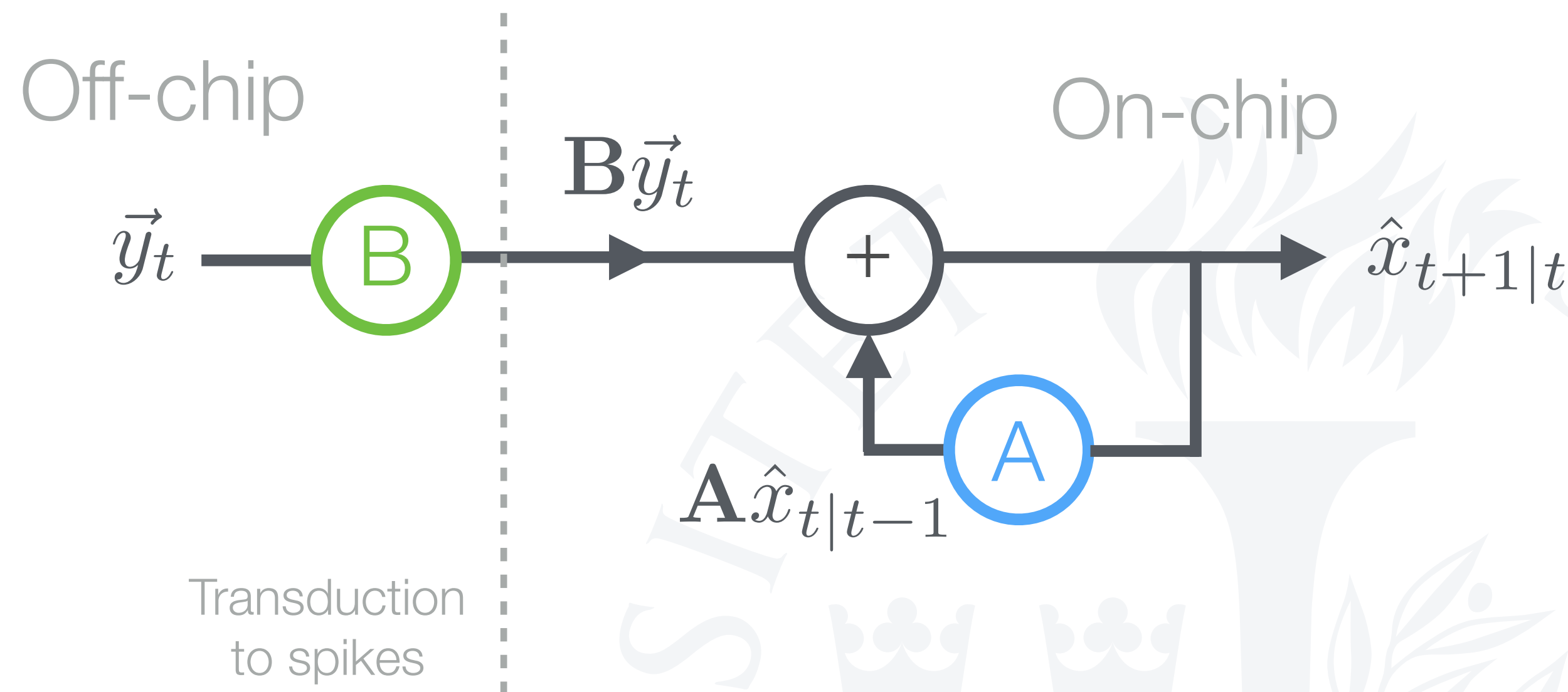


Simple subtraction
(with axon labels)

Intercore communication:
packets and ticks



Go to Desktop...



- Kalman filters have been implemented on ANNs before:
 - Could try to mimic a pre-existing ANN impl. (see TN's efforts with Caffe/Tea).
 - Investigated 'Neural engineering' approach - relies on non-digital neurons.
 - Settled on a simple update step with multiplication and addition. Constrained to steady-state KF through lack of updatable weight registers and back-prop. constraints.

y = state measurement
 x = state prediction
 F = transition matrix
 L = Kalman gain
 w = process noise
 v = measurement noise

A state estimate and a noisy measurement

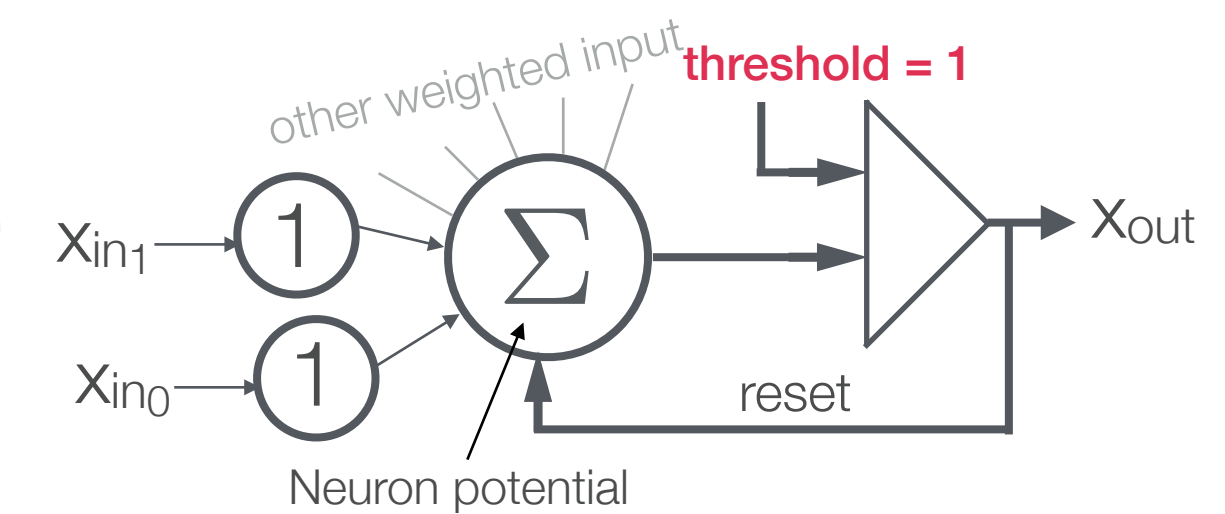
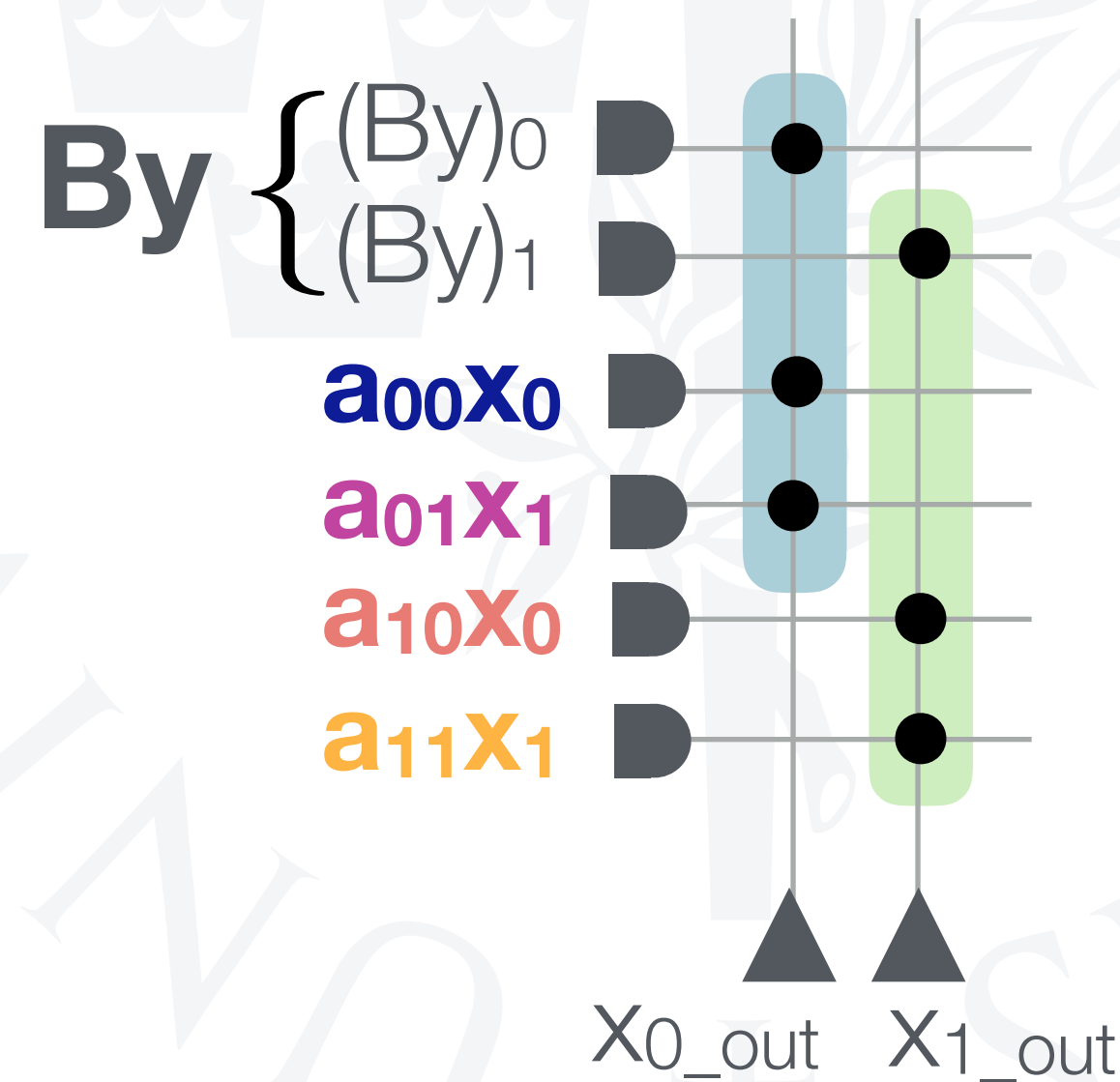
$$\vec{x}_{t+1} = \mathbf{F}\vec{x}_t + \vec{w}_t \quad , \quad \vec{y}_t = \vec{x}_t + \vec{v}_t$$

Steady state Kalman filter update

$$\begin{aligned}
 \hat{x}_{t+1|t} &= (\mathbf{F} - \mathbf{L})\hat{x}_{t|t-1} + \mathbf{L}\vec{y}_t \\
 &= \mathbf{A}\hat{x}_{t|t-1} + \mathbf{B}\vec{y}_t
 \end{aligned}$$

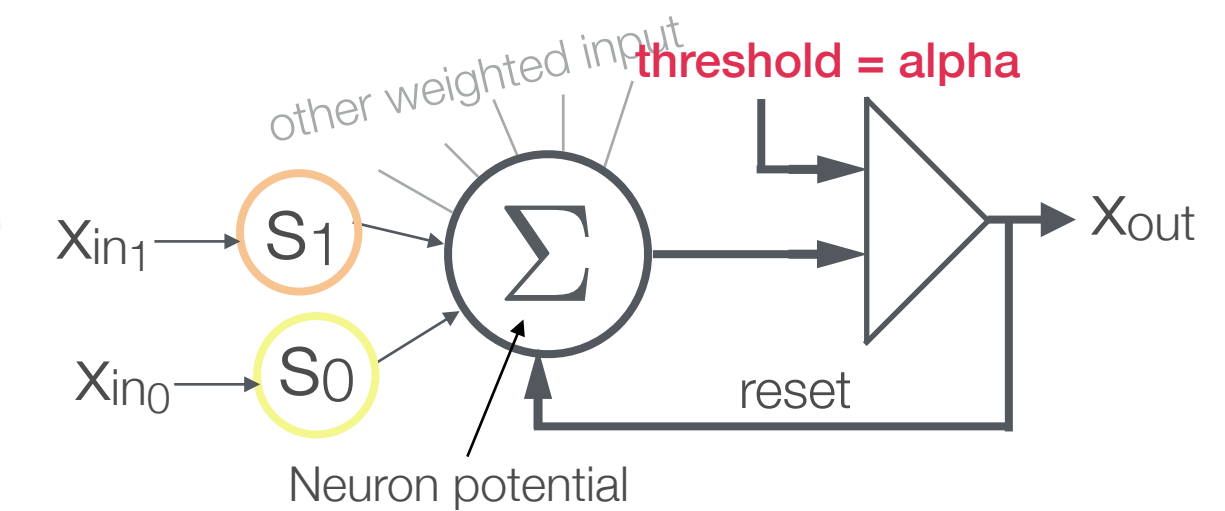
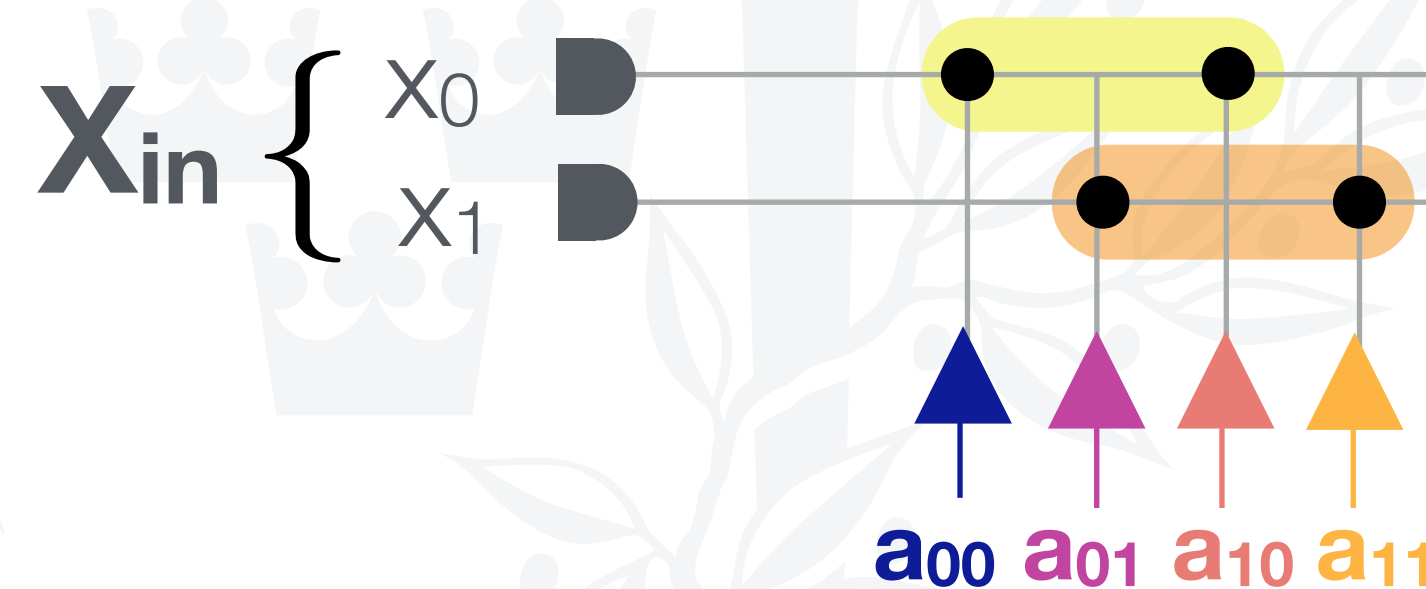
Addition on TrueNorth

$$Ax + By = \begin{pmatrix} a_{00}x_0 + a_{01}x_1 \\ a_{10}x_0 + a_{11}x_1 \end{pmatrix} + \begin{pmatrix} (By)_0 \\ (By)_1 \end{pmatrix}$$



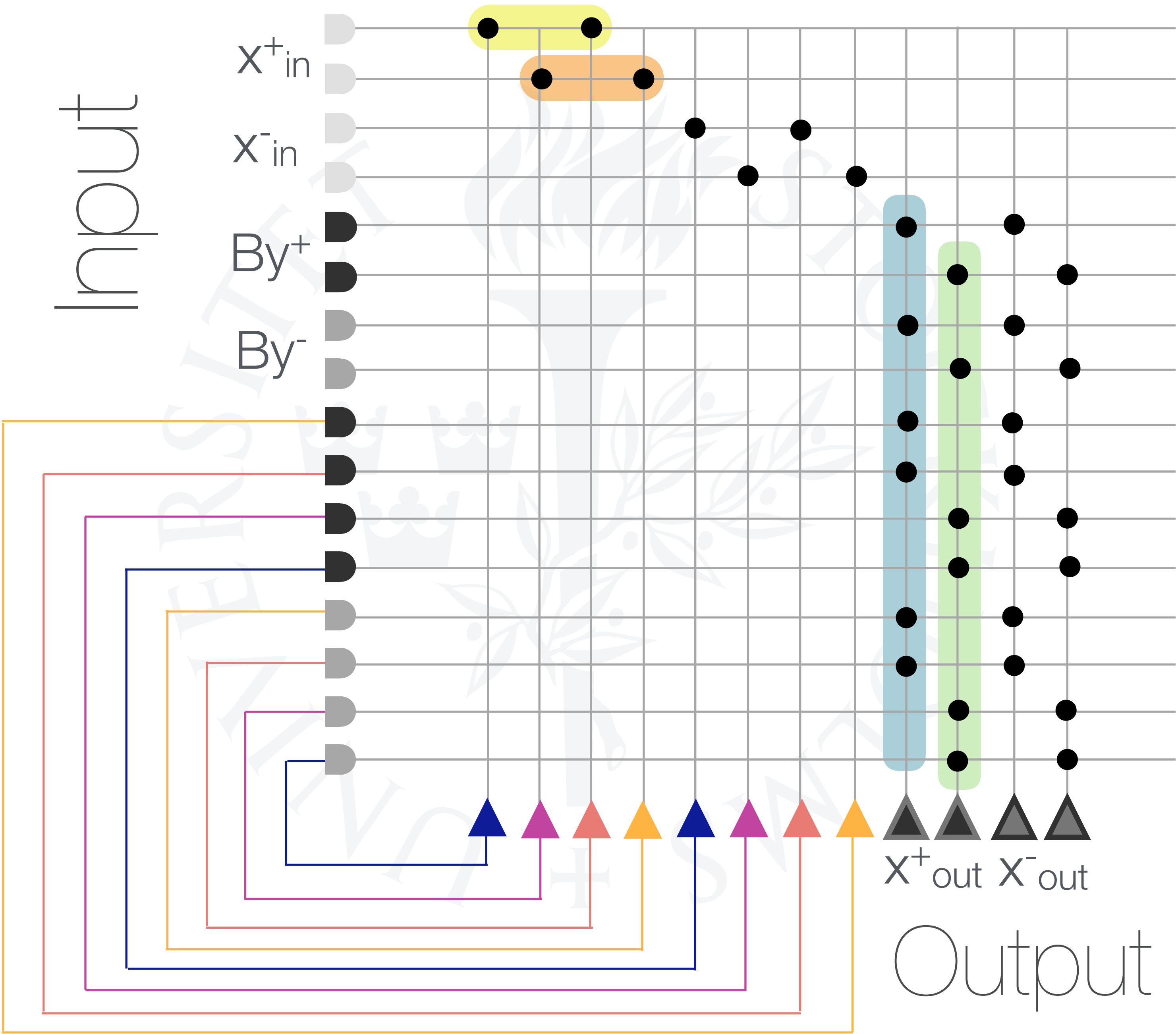
Matrix multiplication

$$Ax = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} a_{00}x_0 + a_{01}x_1 \\ a_{10}x_0 + a_{11}x_1 \end{pmatrix}$$



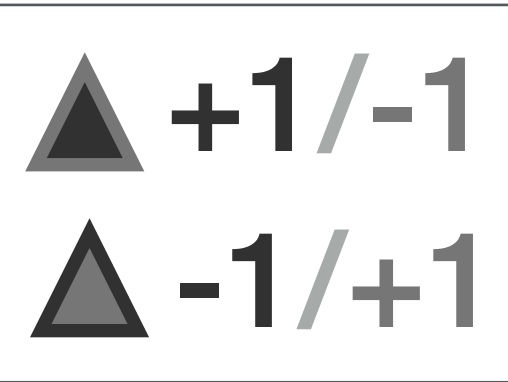
Matrix element weights encoded as $\frac{\text{neuron weight}}{\text{threshold}} = \frac{s}{\alpha}$

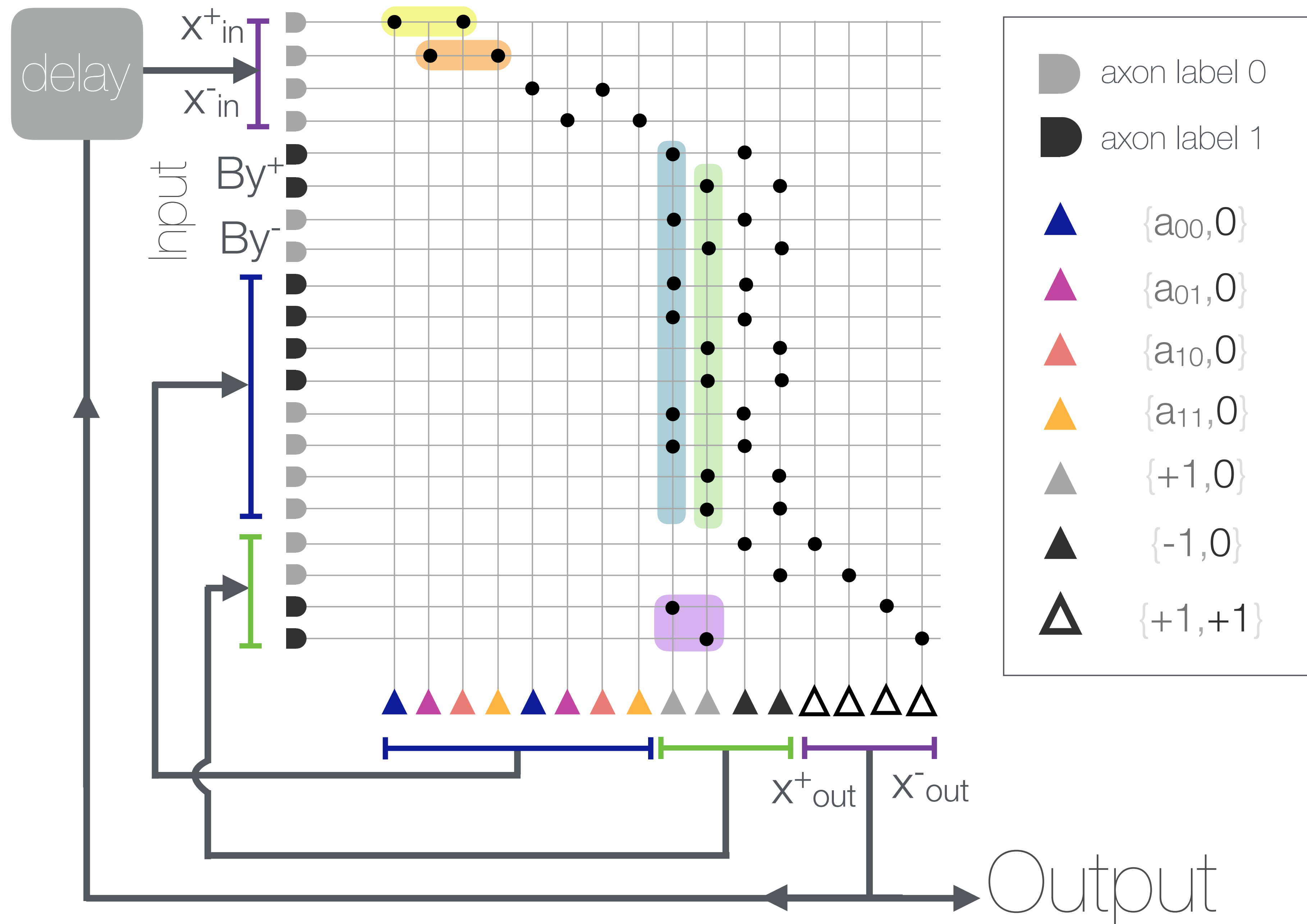
For example to represent a trained weight of 0.512: $\frac{s = 64}{\alpha = 125}$



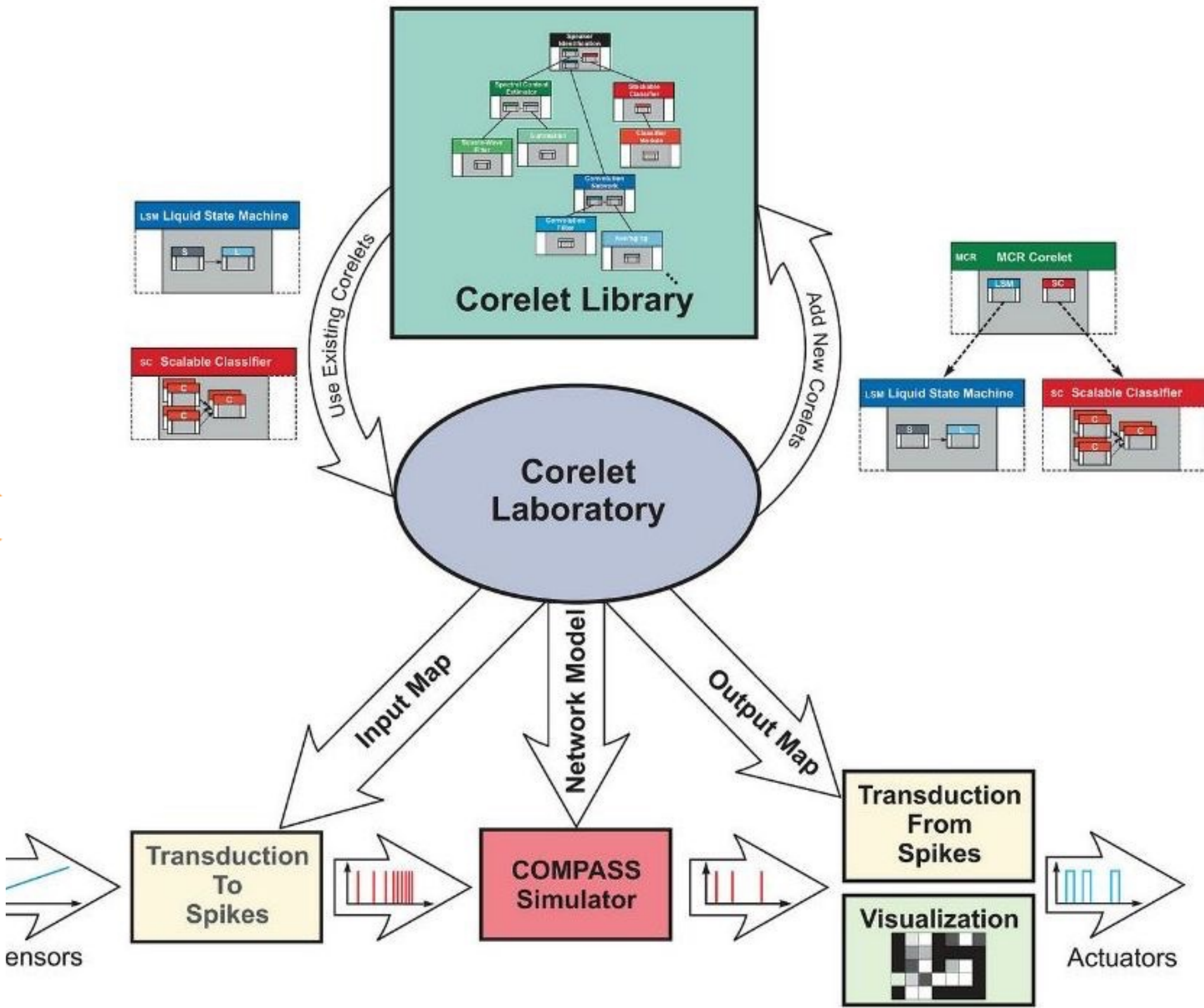
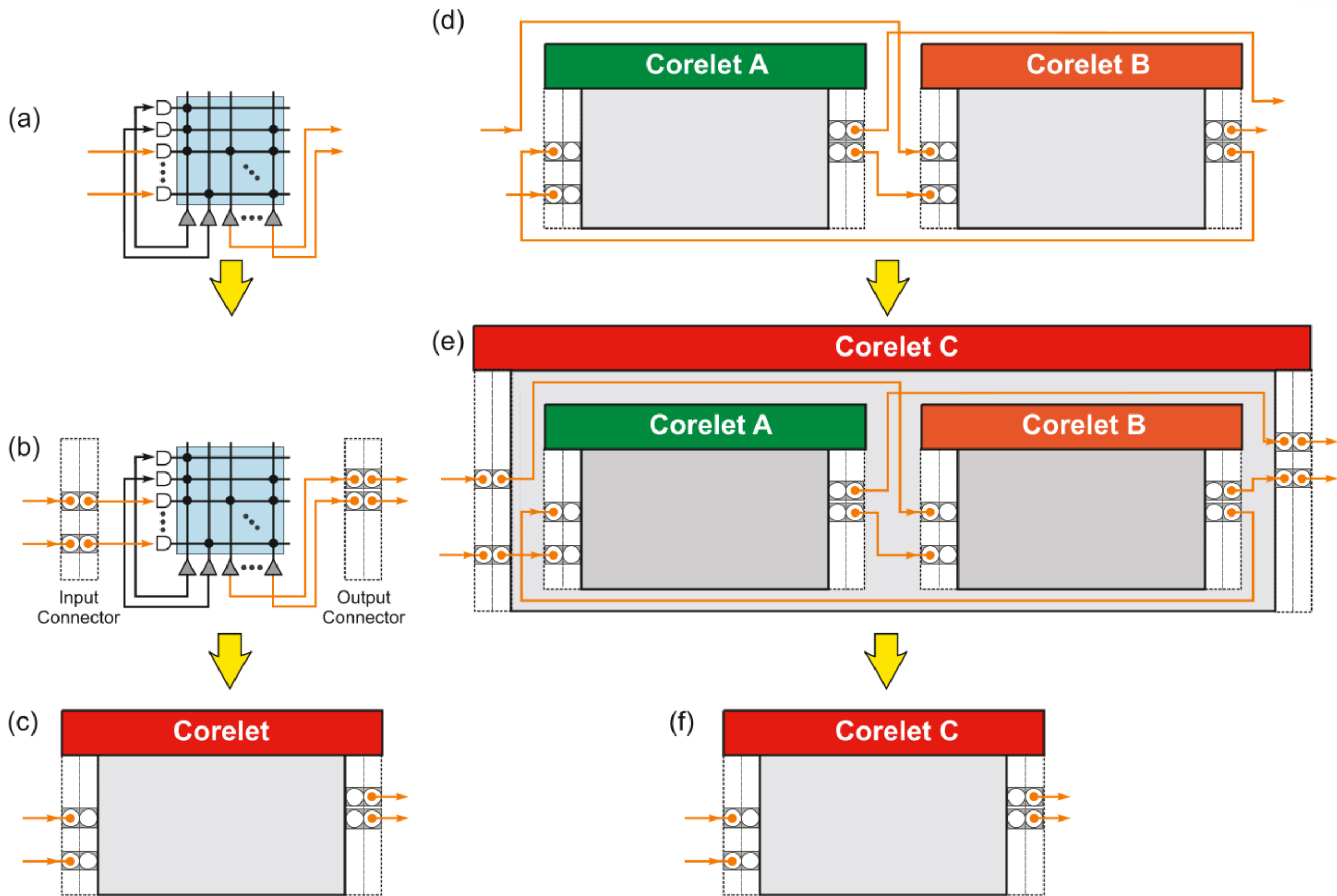
Two channels, to deal with positive and negative values as no logic to deal with signed numbers in spikes.

Not shown in this cartoon: logic to combine positive and negative channels after firing.





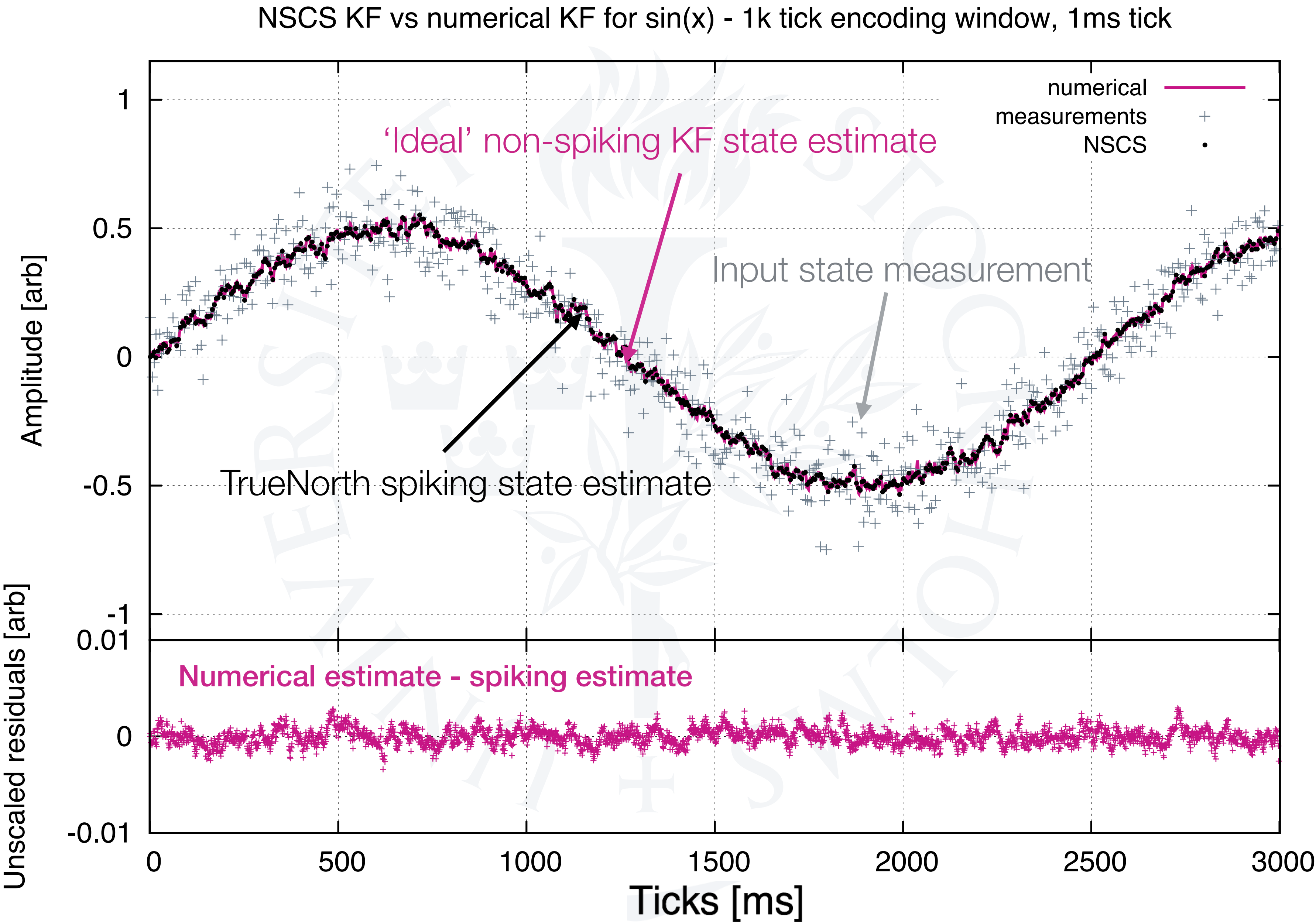
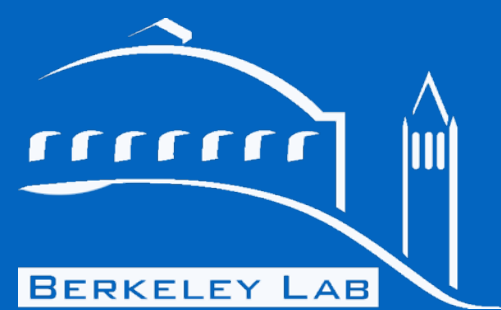
From crossbar and neuron pars
to a model file

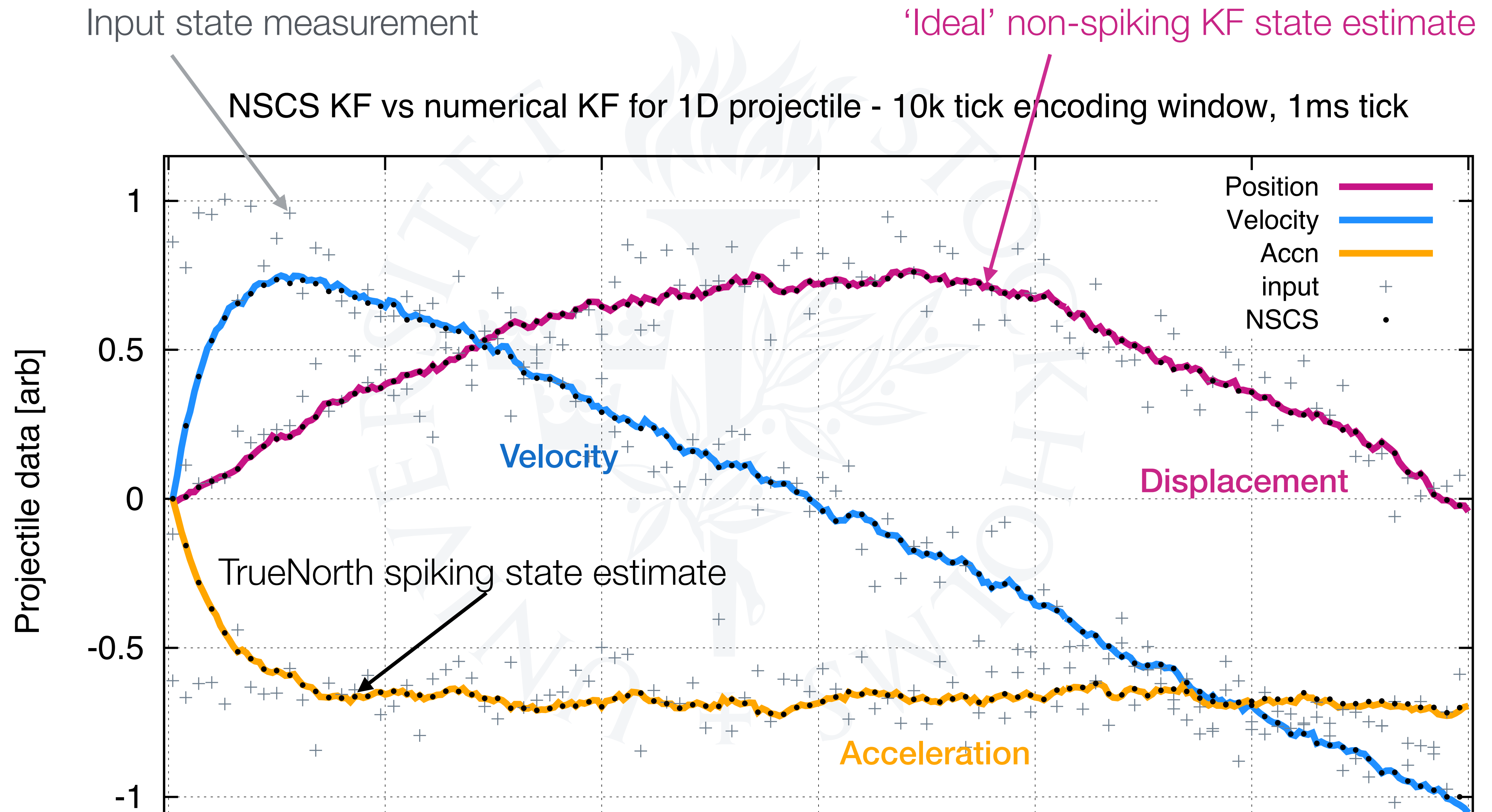


From crossbar... to corelet...

... to model file

This is all very well but does it actually work?





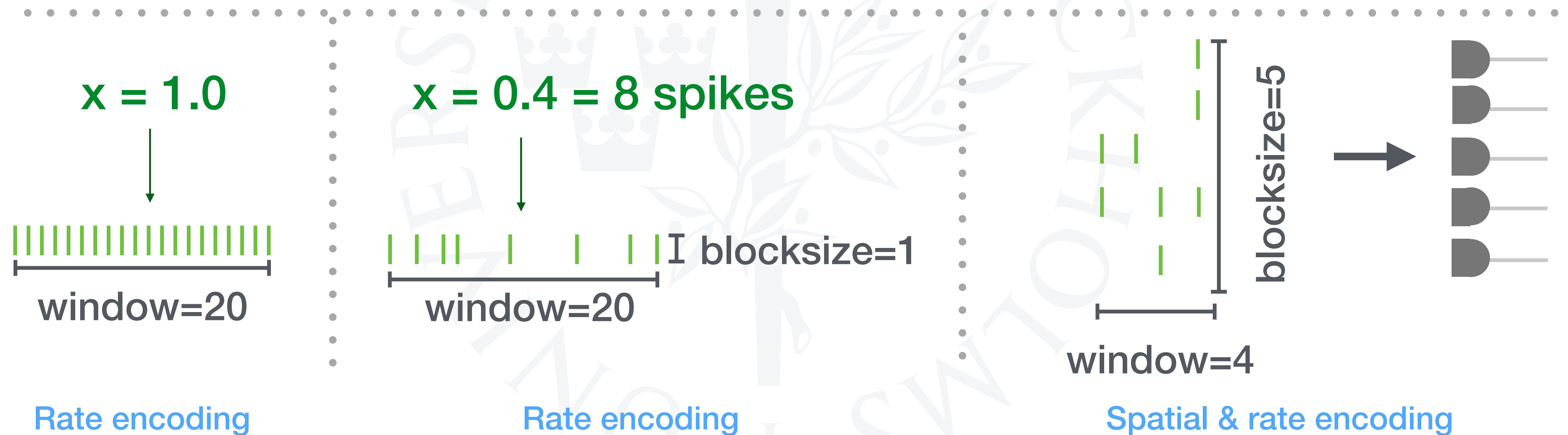
In binary:

$100 \neq 010 \neq 001$

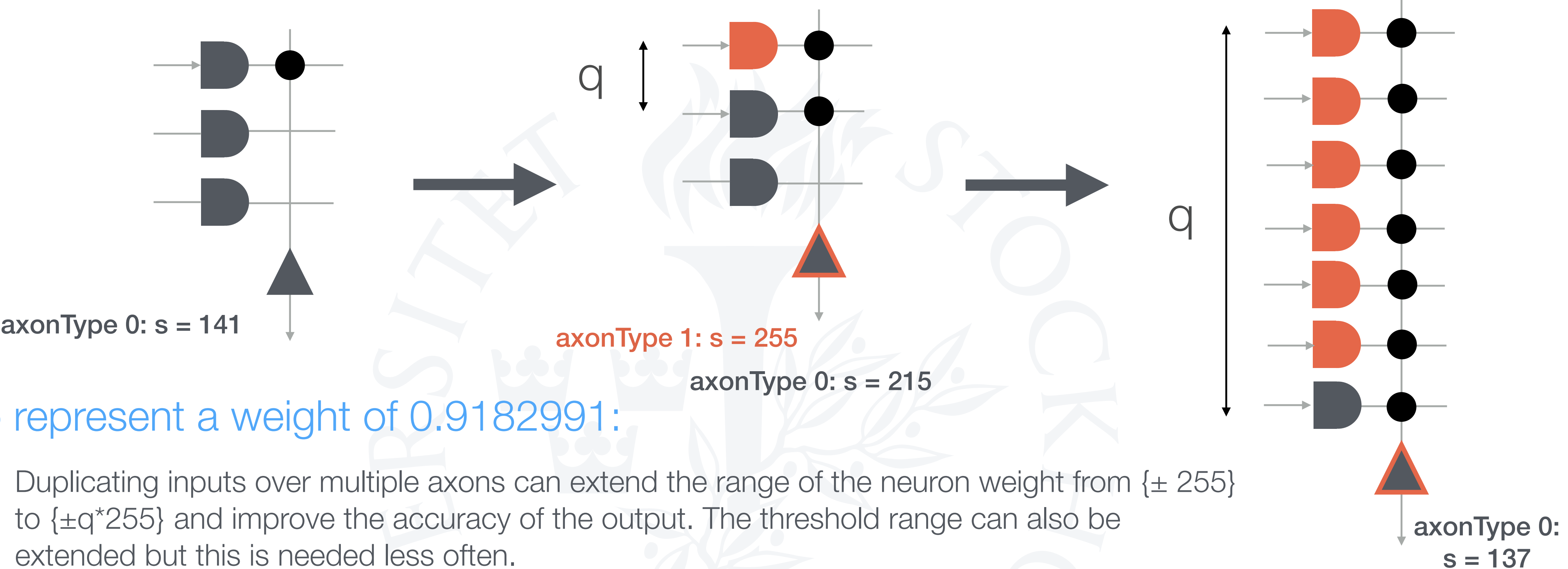
In unary:

$100 == 010 == 001$

- TrueNorth communicates data through spikes.
 - We have chosen to encode spikes in unary as it lends more flexibility to crossbar formulation.
 - Encoding of values can be done in space (across axons) as well as in time (across multiple ticks).



Although in all of these cases there are 20 spikes (or lack of spikes) per word, there is a clear tradeoff between latency and axons used.

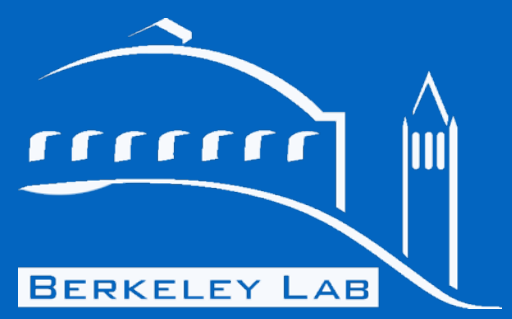


To represent a weight of 0.9182991:

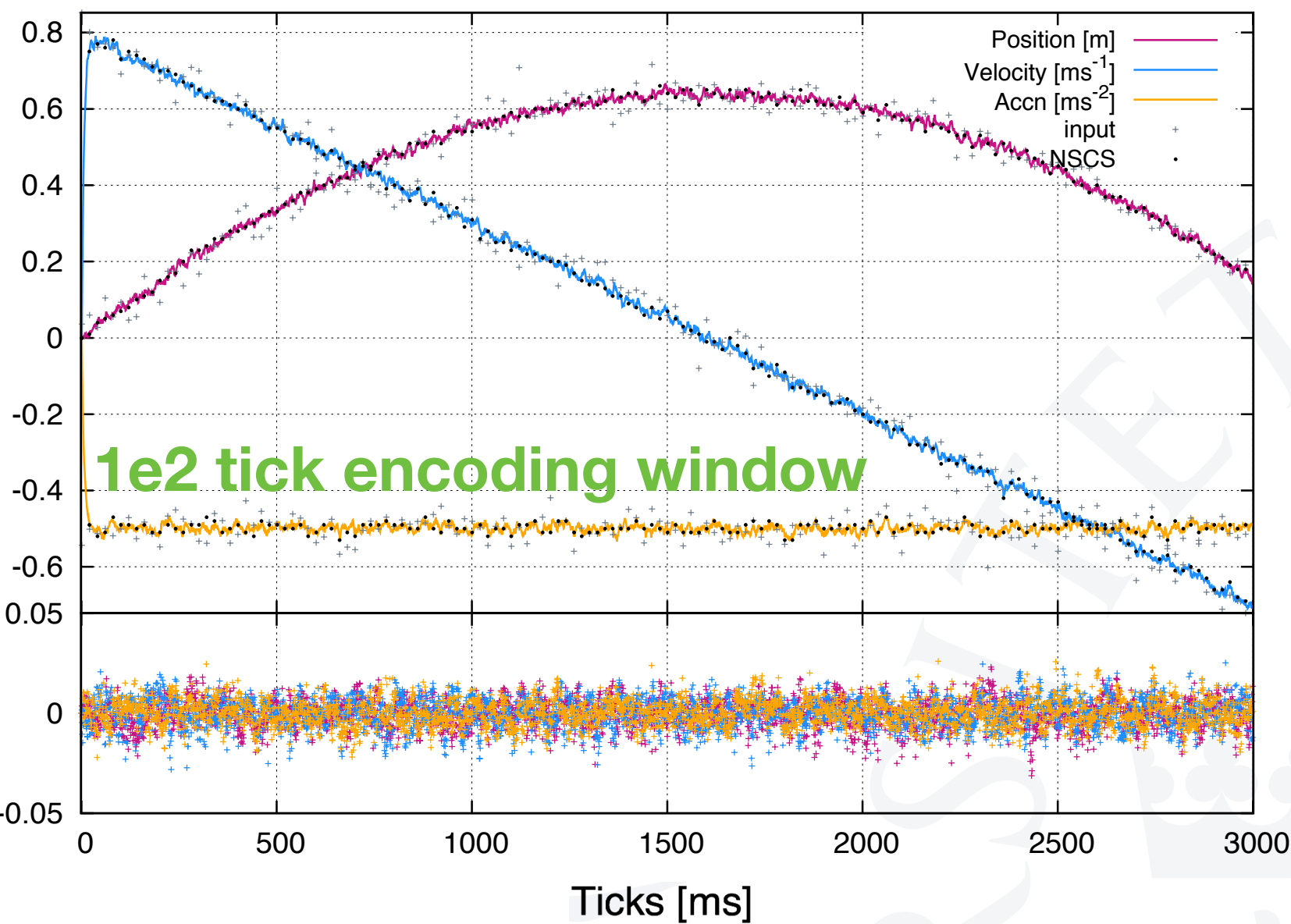
Duplicating inputs over multiple axons can extend the range of the neuron weight from $\{\pm 255\}$ to $\{\pm q \cdot 255\}$ and improve the accuracy of the output. The threshold range can also be extended but this is needed less often.

8-bit neuron weight:	$s = 45$	2*(8)-bit neuron weight:	$s = 251$	7*(8)-bit neuron weight:	$s = 1922$
18-bit neuron threshold:	$\alpha = 49$	18-bit neuron threshold:	$\alpha = 306$	18-bit neuron threshold:	$\alpha = 2093$

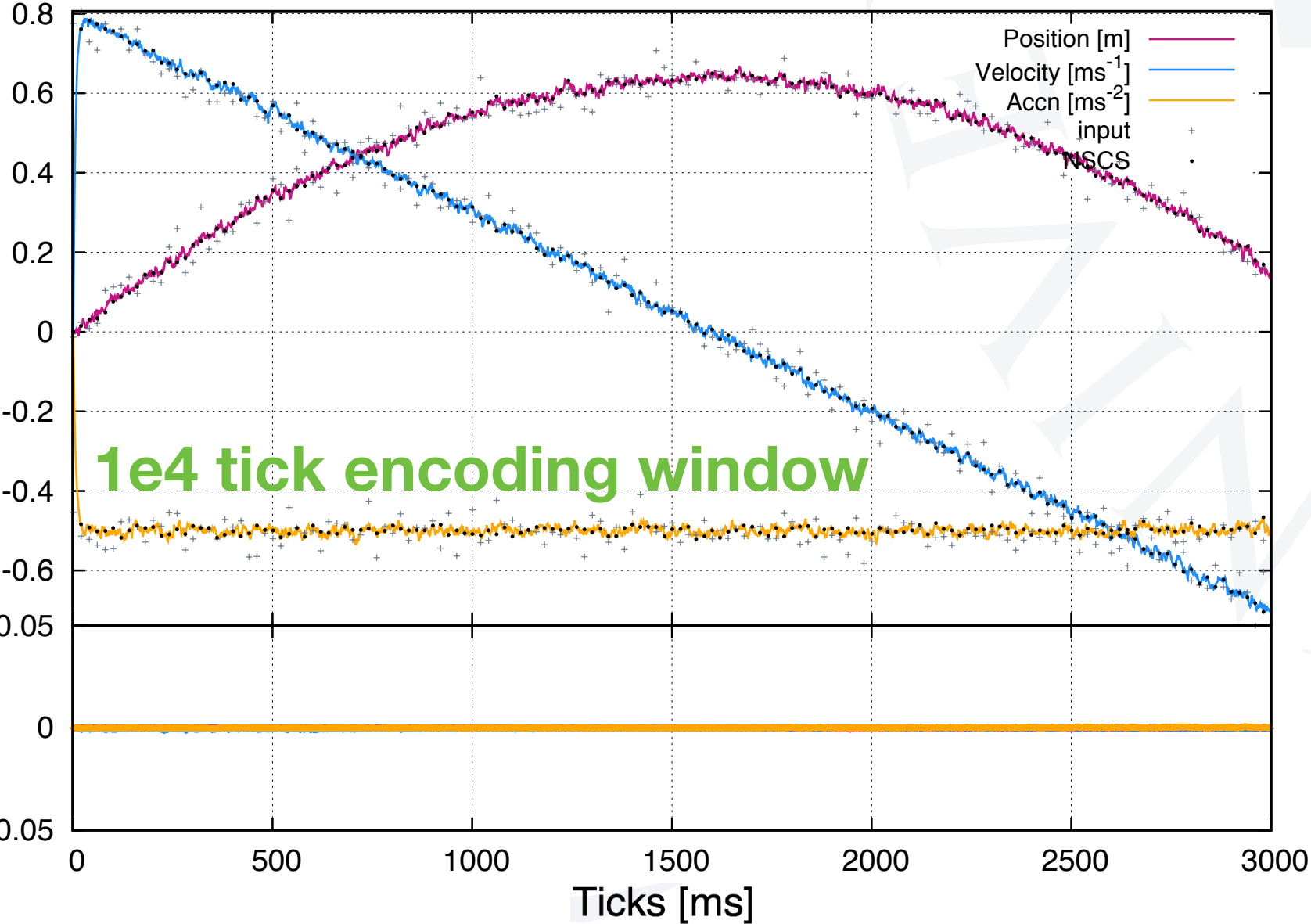
Using 1 axon, trained weight to 3 s.f.	$\frac{s}{\alpha} = 0.91837$	Using 2 axons, trained weight to 4 s.f.	$\frac{s}{\alpha} = 0.91830$	Using 7 axons, trained weight to 7 s.f.	$\frac{s}{\alpha} = 0.91829909$
---	------------------------------	--	------------------------------	--	---------------------------------



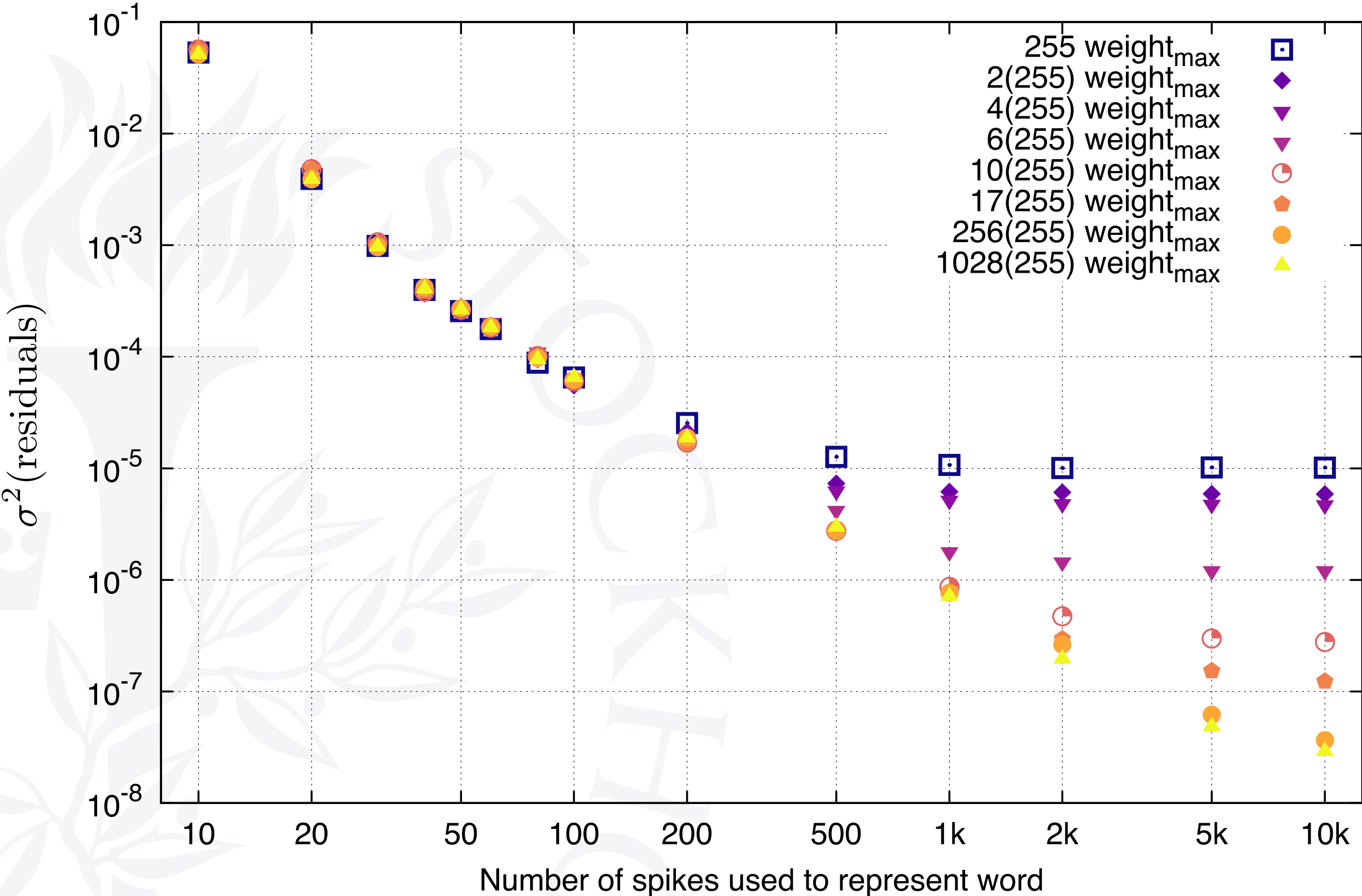
1D Projectile tracking: 100 tick encoding, 1e-3 process noise, 1e-3 meas. noise, 1e-3 sampling rate



1D Projectile tracking: 10k tick encoding, 1e-3 process noise, 1e-3 meas. noise, 1e-3 sampling rate



Effect of word size on MSE between TN KF and ideal KF for 1D projectile



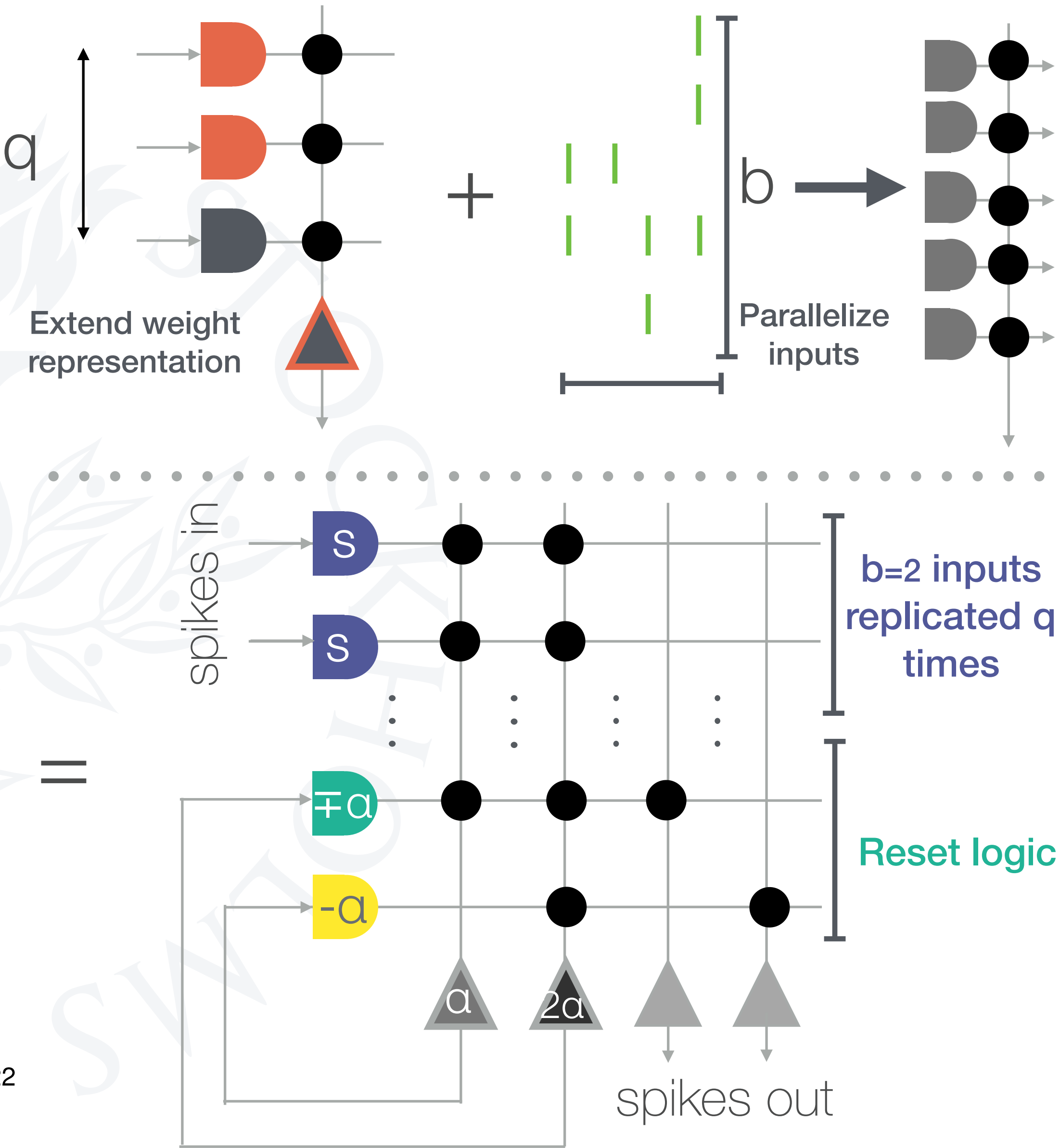
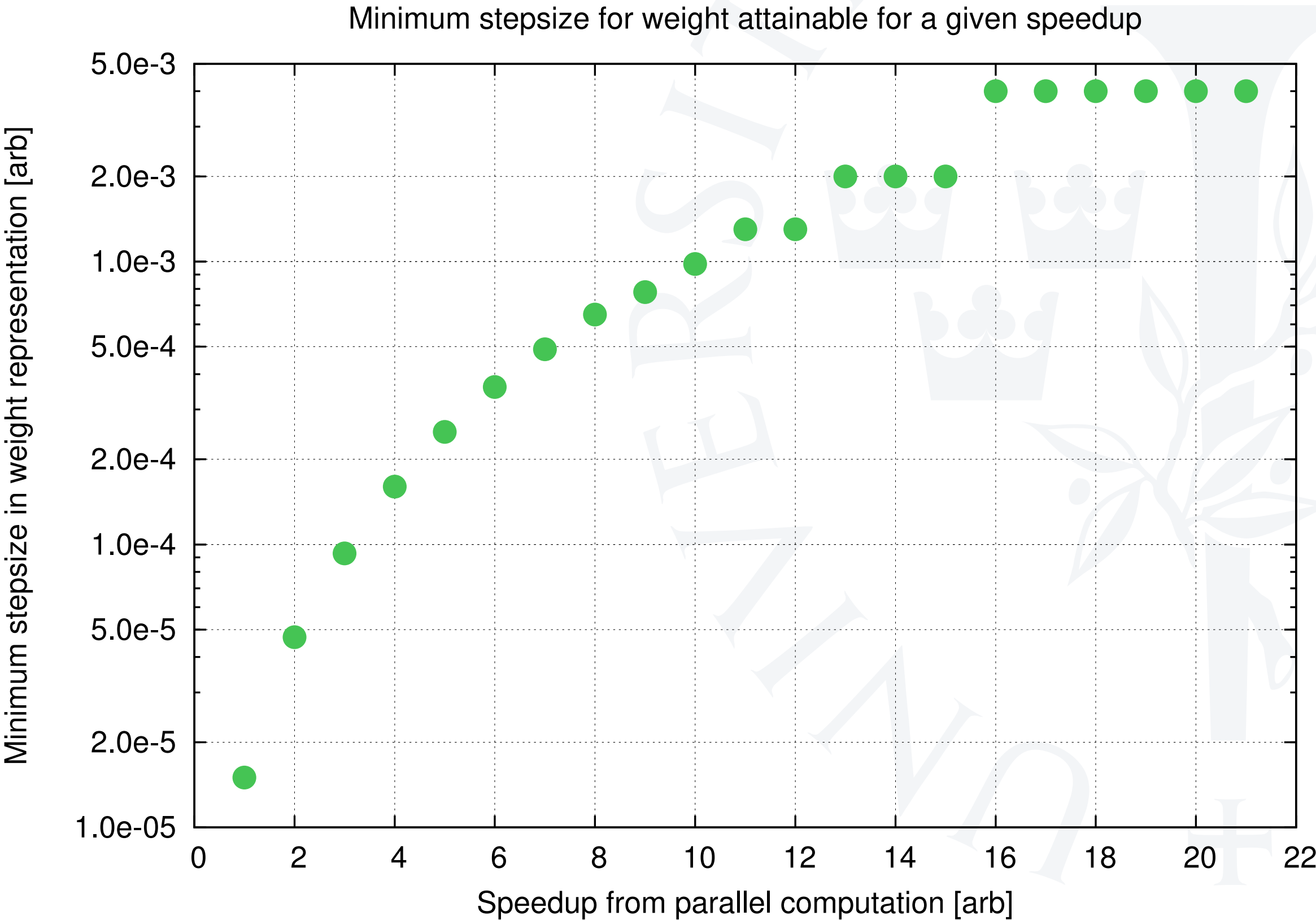
Data representation in spikes

Data representation does have an effect on accuracy:

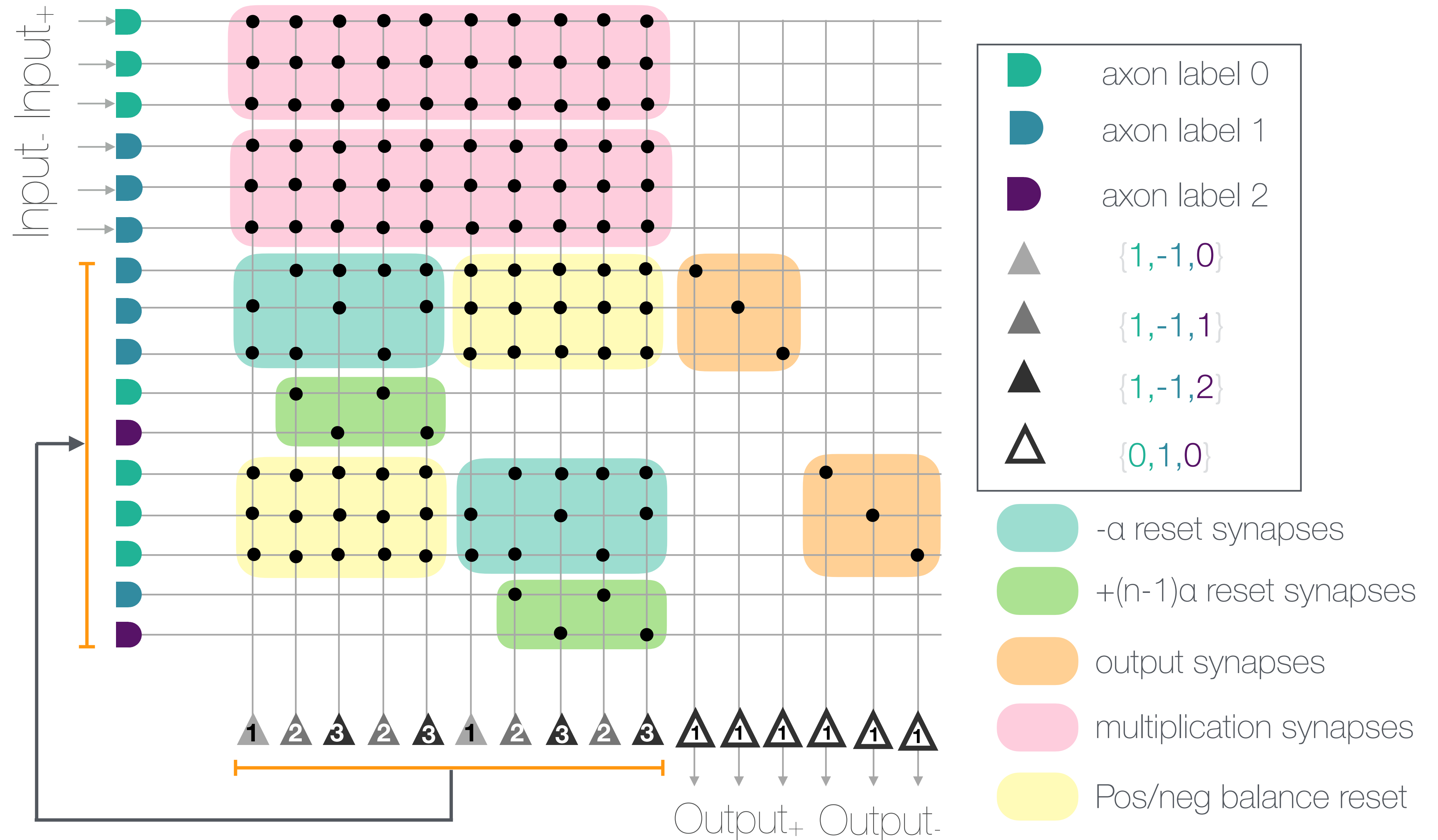
- There is a limit to the accuracy that can be achieved using more spikes per word for a given neuron weight register size.
- Using more spikes per data word will increase the amount of real estate consumed by the model or the latency of each estimate depending on which encoding scheme is used.

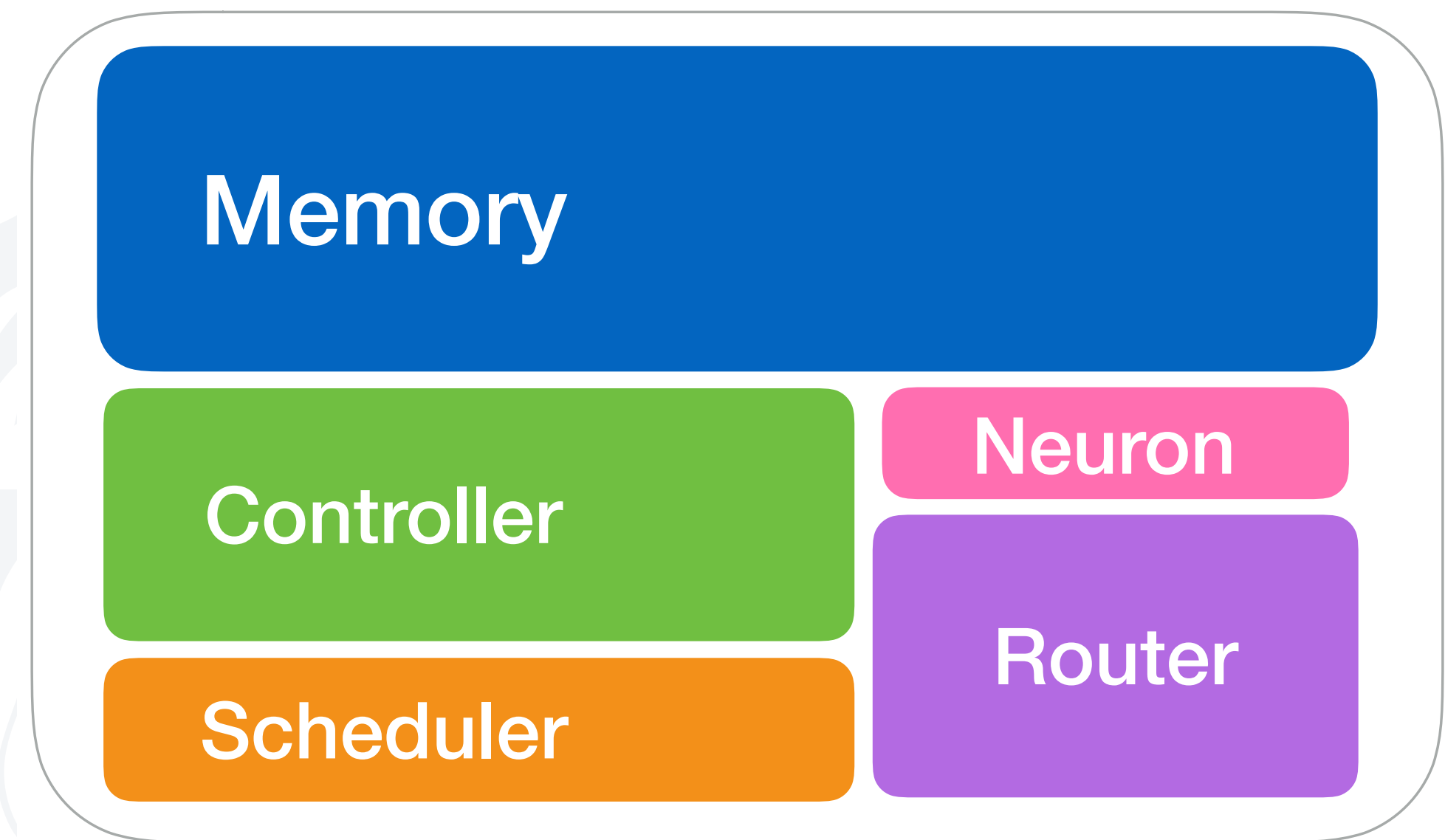
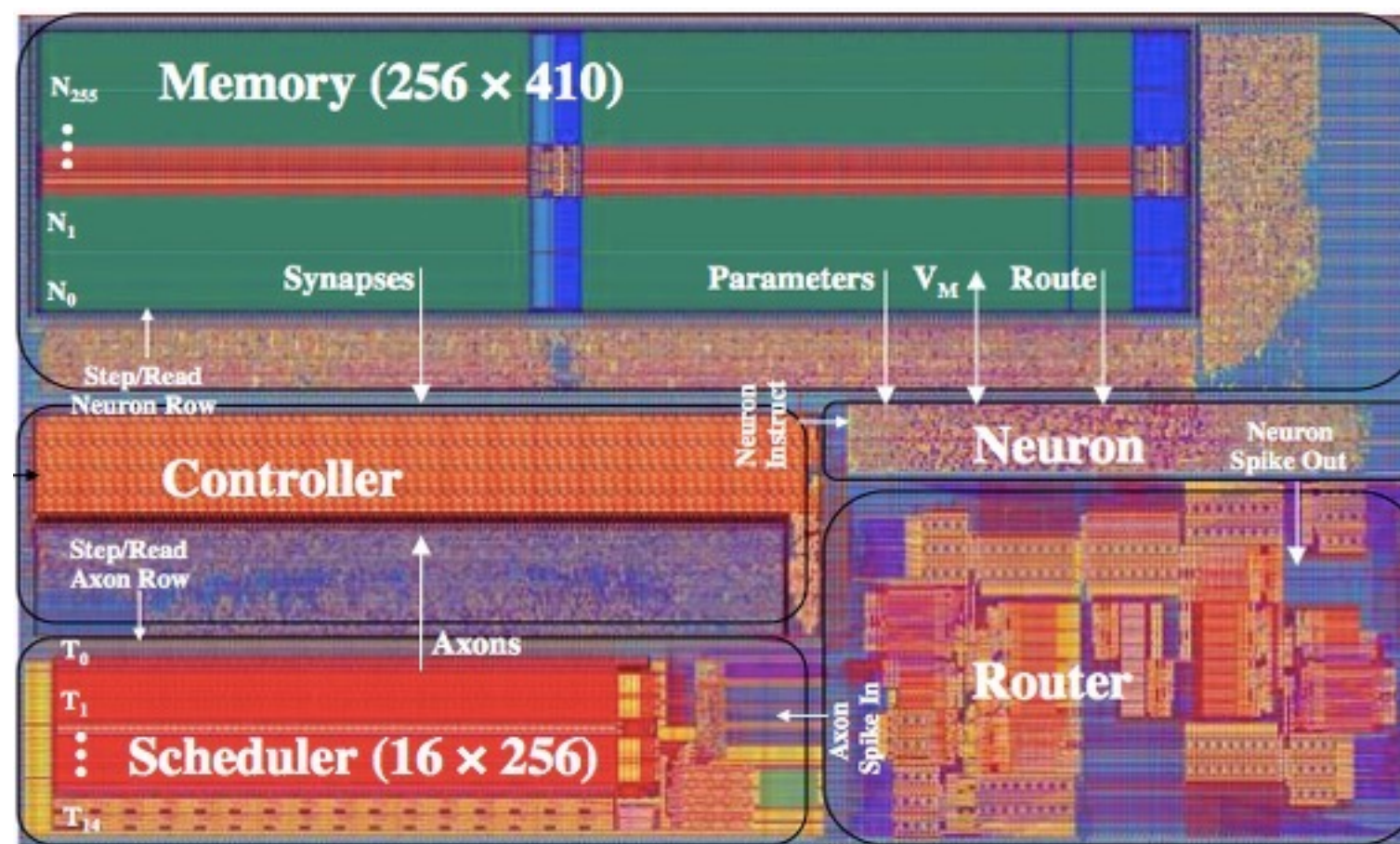
Parallelism reduces latency at the expense of weight representation

Because of the finite size of a core if more neurons are used for parallel computation, less are available to extend the range of the weight and threshold registers.



But what about computation time? Can we increase the rate of the global 'tick'?





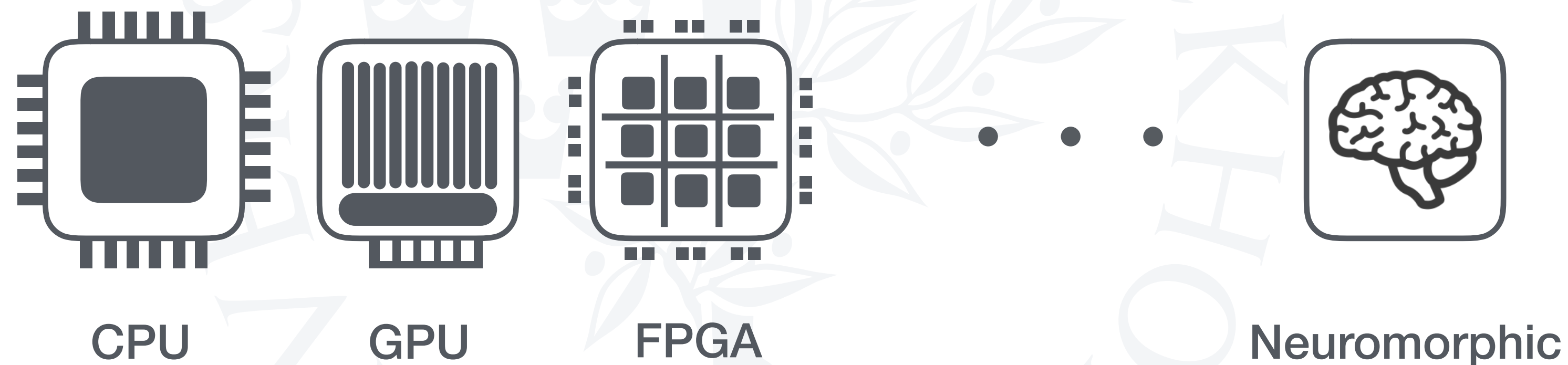
Remainders in the neuron potential

- **Scheduler error**: spike delivery is ensured to be longer than max latency between source and destination. So if chip has a large model implemented and tick is sped up - spike generator may refuse to create file.
- **Token ctrl error**: If the spikes can't be processed in time, because of the volume of spikes in a single core.
- A TrueNorth chip can only take so many spikes per port, and then there is a firmware and software pipeline that must match this **throughput**. We see on the order of **15 million spikes** per second total throughput, which is **480 Mbps** data rate between FPGA and TN.

We never managed to cause either of the first 2 failure modes with our simple serial kaman filter. However we did cause the last error to be thrown when we tried to increase the tick rate from 1kHz to 10kHz. A 2kHz tick was possible.

Throttling arose when using a serial spike train and $< 0.1\%$ of the chip area. Unlikely more complicated models could be sped up on this version of TrueNorth.

- To achieve the lower power constraints of the DARPA project the TrueNorth team has had to compromise with programming flexibility and neuron complexity in this chip.
- Not only is it not fast enough for our computing needs it cannot be sped up.
- Although parallelism can be used to reduce latency there is an upper limit of 21x.
- The chip communicates with spikes: which must be encoded. The chip cannot be operated without an FPGA or equivalent to encode/decode and transmit spikes. An external 'tick' must also be supplied.
- This is not an ANN but a neuromorphic neural network, and only an approximate one and because of that cannot implement either real-time weight updates or back-prop.



- We have adapted CPU's, GPU's and FPGA's for our needs. Could neuromorphic chips feature in our toolkit in 10 years time? Perhaps but it seems more likely we would use an ASIC ANN.

Questions?

McCulloch-Pitts neuron

The McCulloch-Pitts neuron is a simplified version of the perceptron, without a bias and with no Hebbian learning rules. It has a binary output based on a thresholded step function.

$$y = \text{step} \left(\sum_{i=1}^N n_i s_i \right)$$

TrueNorth neuron

II. NEURON SPECIFICATION

Our neuron model is based on the leaky integrate-and-fire neural model with a constant leak, which we augmented in several ways. We begin by briefly reviewing the classic leaky integrate-and-fire neural model, followed by an in-depth description of our neuron model.

A. The Leaky Integrate and Fire (LIF) Neuron

The operation of the leaky integrate-and-fire (LIF) neuron model with a constant leak is described by five basic opera-

$$y = \text{step} \left(V_j(t-1) + \sum_{i=1}^{N=256} n_i(t) s_j^{G_i} w_{i,j} + \lambda_j \right)$$

Hodgkins-Huxley neuron

Hodgkins-Huxley model is a biologically plausible neuron model that accounts for leakage across the membrane from different ions and different rates of ions. The I_i term is a parameterized model of the conductance across the membrane, with at least 20 parameters.

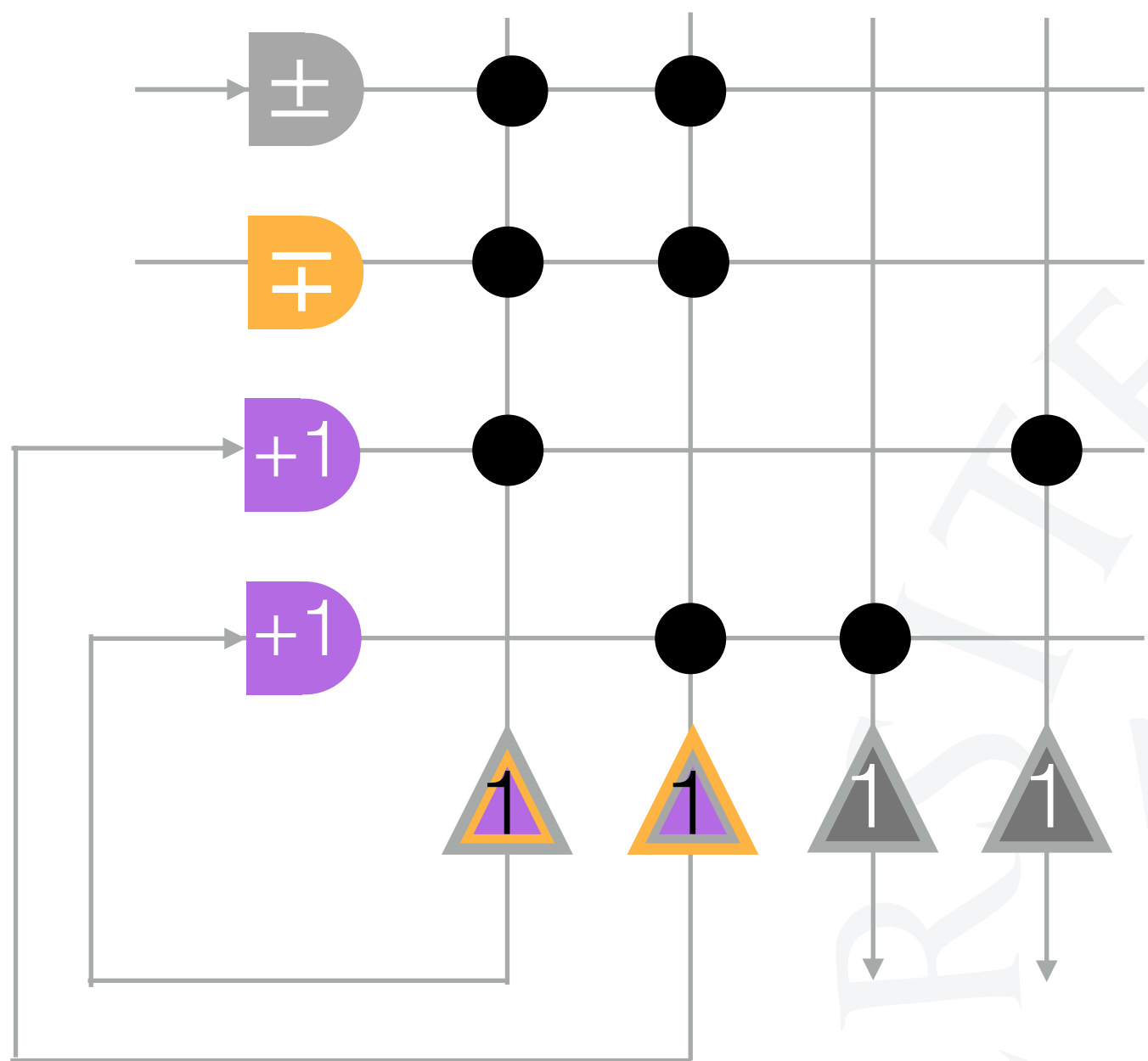
$$C_m \frac{dV(t)}{dt} = - \sum_i I_i(t, V)$$

Leaky Integrate-and-fire neuron

One of the simplifications to this model to make it tractable is the LIF neuron. Leaky integrate-and-fire is a reduced complexity subset of HH. Grouping any current independent of input as a constant leak.

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t)}{R_m}$$

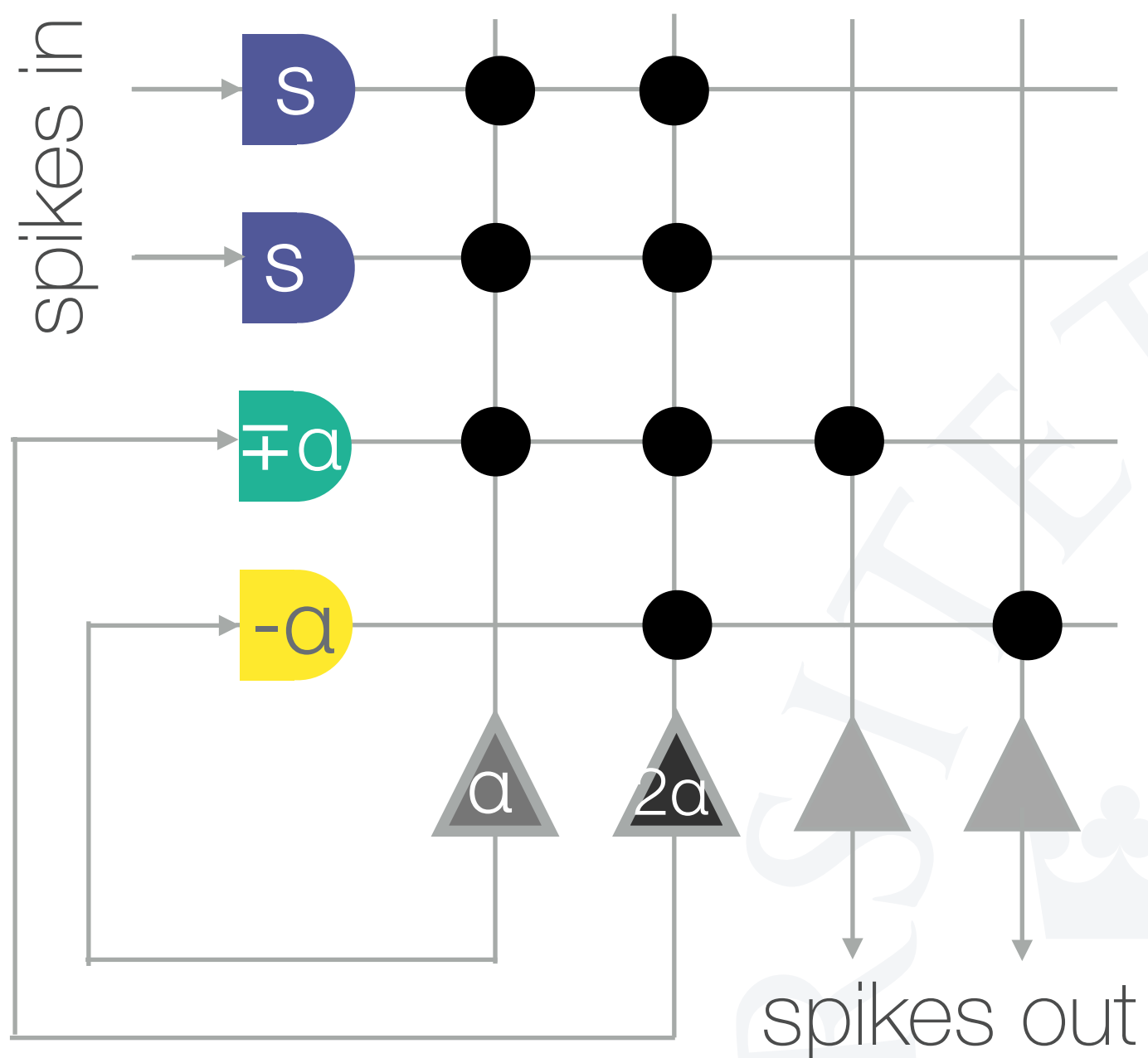
Calling TN neurons LIF seems to be a misnomer. They are far closer McCulloch-Pitts neurons and have none of the time-dependent capabilities of LIF neurons (aside from the persistent neuron potential).



$V_{\text{Neuron \#0}}$	$V_{\text{Neuron \#1}}$	Spikes in	Spikes out
$0 + 1 - 1 = 0$	$0 - 1 + 1 = 0$	1,1	0,0
$0 - 1 = -1$	$0 + 1 - 1 = 0$	0,1	0,0
$-1 + 1 + 1 - 1 = 0$	$0 - 1 = -1$	1,0	0,1
$0 + 1 - 1 = 0$	$-1 + 1 - 1 = -1$	1,0	1,0
0	$-1 + 1 = 0$	0,0	1,0
Total spikes		+3, -2	$2 - 1 = 1$

Preserving the remainder affects performance

- Spike trains cannot encode negative values. There is no easy way of dealing with signed logic that could be implemented in neuromorphic neurons.
- Instead, positive and negative values are dealt with in parallel trains.
- These are combined in addition corelets, like the bar shown above for a serial addition of positive and negative numbers.
- The recurrent network above correctly changes the neuron potential in the opposite signed neuron if the positive or negative one were to spike.



Remainders in the neuron potential

- In the serial case the reset scheme for multiplication is that when a neuron fires its potential is reduced by α . In the parallel implementation all neurons in the block are reset by $(\alpha \times \# \text{ neurons that spiked})$. The logic to implement this is shown in the xbar to the left.
- Other parallel implementations either reset all neurons to zero after a tick, or reset all neurons to zero after a data word: thus throwing away neuron remainders.
 - Previous TN groups have shown this to work for classification nets - does it work for us?

$V_{\text{Neuron \#0}}$	$V_{\text{Neuron \#1}}$	Spikes in	Spikes out
$0+12-\textcolor{red}{7}=5$	$0+12 = 12$	1,1	0,0
$5+12-\textcolor{red}{7}=10$	$12+12-\textcolor{blue}{7}-\textcolor{red}{14}=3$	1,1	0,1
$10+12-\textcolor{red}{7}-\textcolor{blue}{7}=8$	$3+12-\textcolor{red}{14}=1$	1,1	1,1
$8+12-\textcolor{red}{7}-\textcolor{blue}{7}=6$	$1+12 = 13$	1,1	1,1
$6+12-\textcolor{red}{7}=11$	$13+12-\textcolor{blue}{7}-\textcolor{red}{14}=4$	1,1	1,0
$11+12-\textcolor{red}{7}-\textcolor{blue}{7}=9$	$4+12-\textcolor{red}{14}=2$	1,1	1,1
$9+12-\textcolor{red}{7}-\textcolor{blue}{7}=7$	$2+12-\textcolor{red}{14}=0$	1,1	1,1
$7-\textcolor{blue}{7} = 0$	$7-\textcolor{blue}{7} = 0$	0,0	1,1
Total spikes		14	12

As an example let: $s = 6$, $\alpha = 7$ to represent the trained weight of $6/7$.

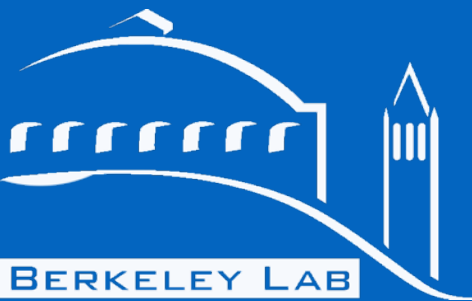
Then, for blocksize = 2 (as shown in the cartoon), if we send in 14 spikes we expect:

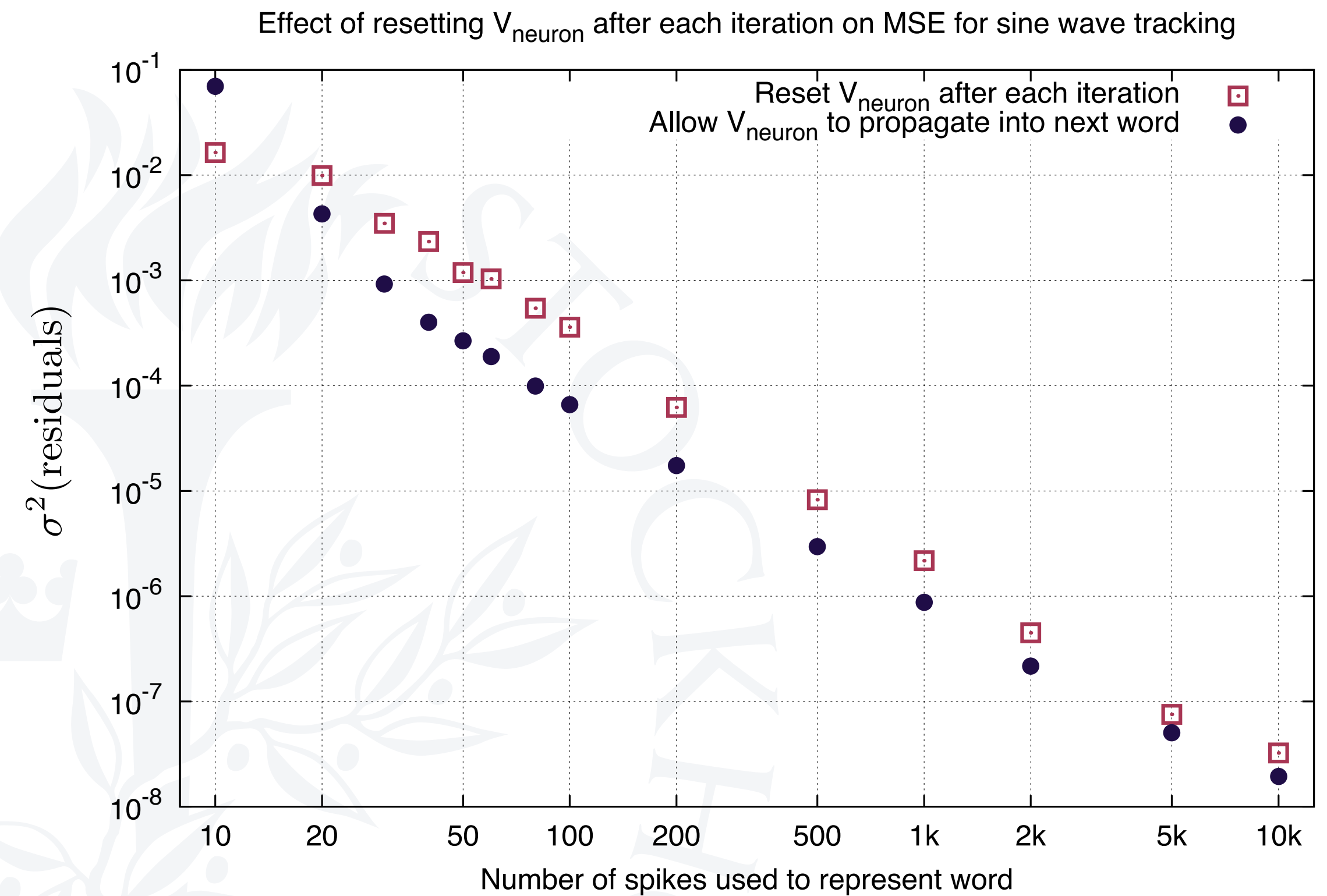
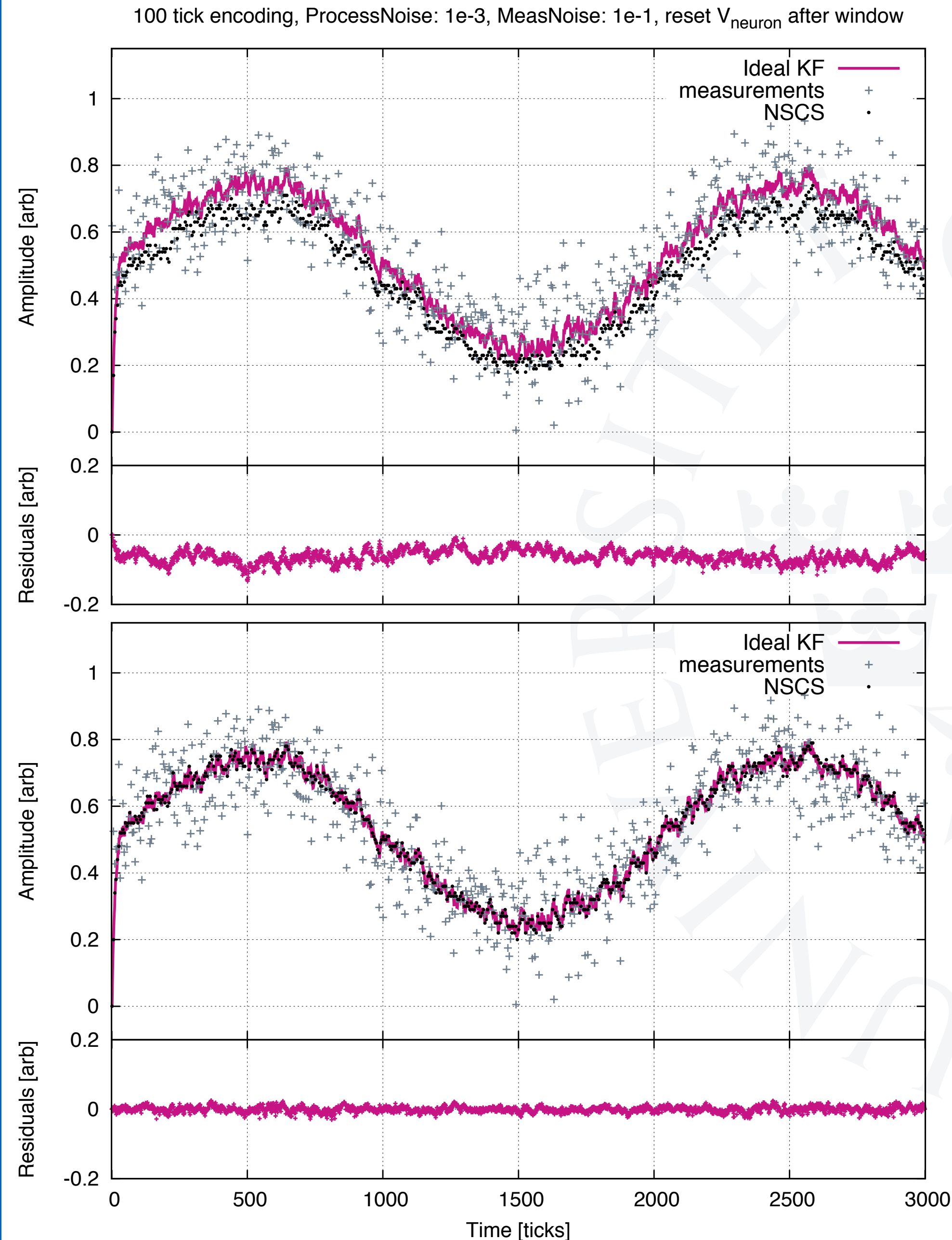
$$14 \times (6/7) = 12$$

spikes out.

Note that because of the reset line the output takes an additional tick to be output.

Red shows self-reset, **blue** shows reset due to recurrent connection from other neurons in block firing.





Preserving the remainder affects performance

Other TrueNorth projects do not preserve a remainder, including their RNN projects. Our reset crossbar, whilst adding complexity, improves the estimate.

- Again, there is a dependency on data representation in number of spikes.
- Resetting the potential after each word is not the right solution for this use case of the chip.

NSCS chip simulation exactly
matches NS1e output

