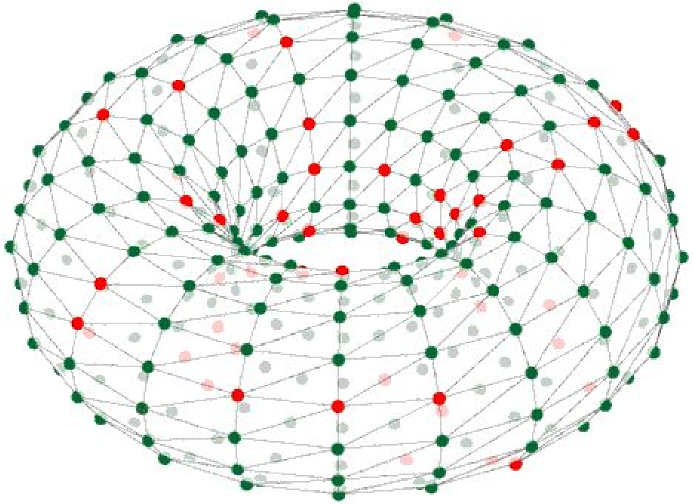
Running PyNN Simulations on SpiNNaker

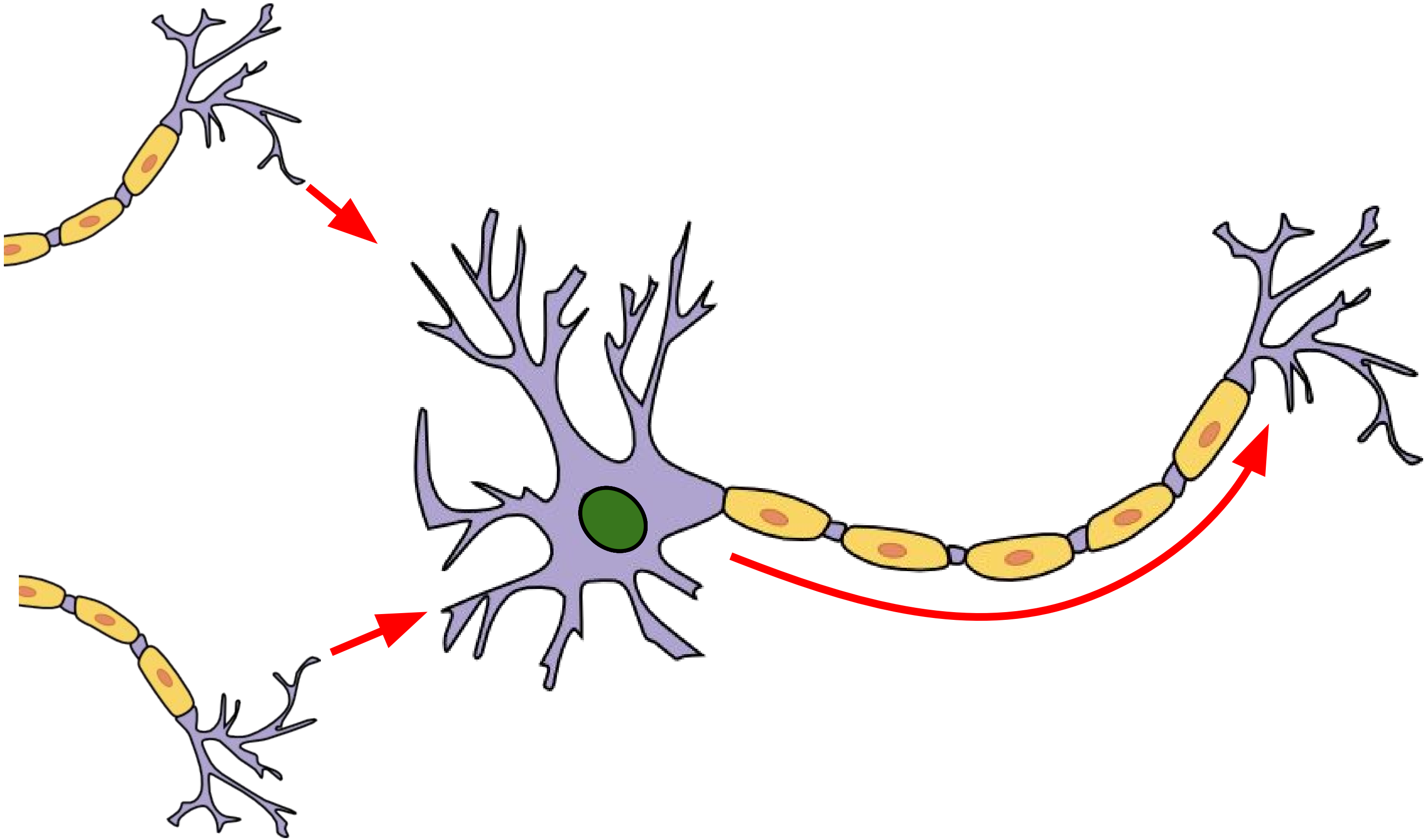


Alan Stokes

Young Scientist HBP

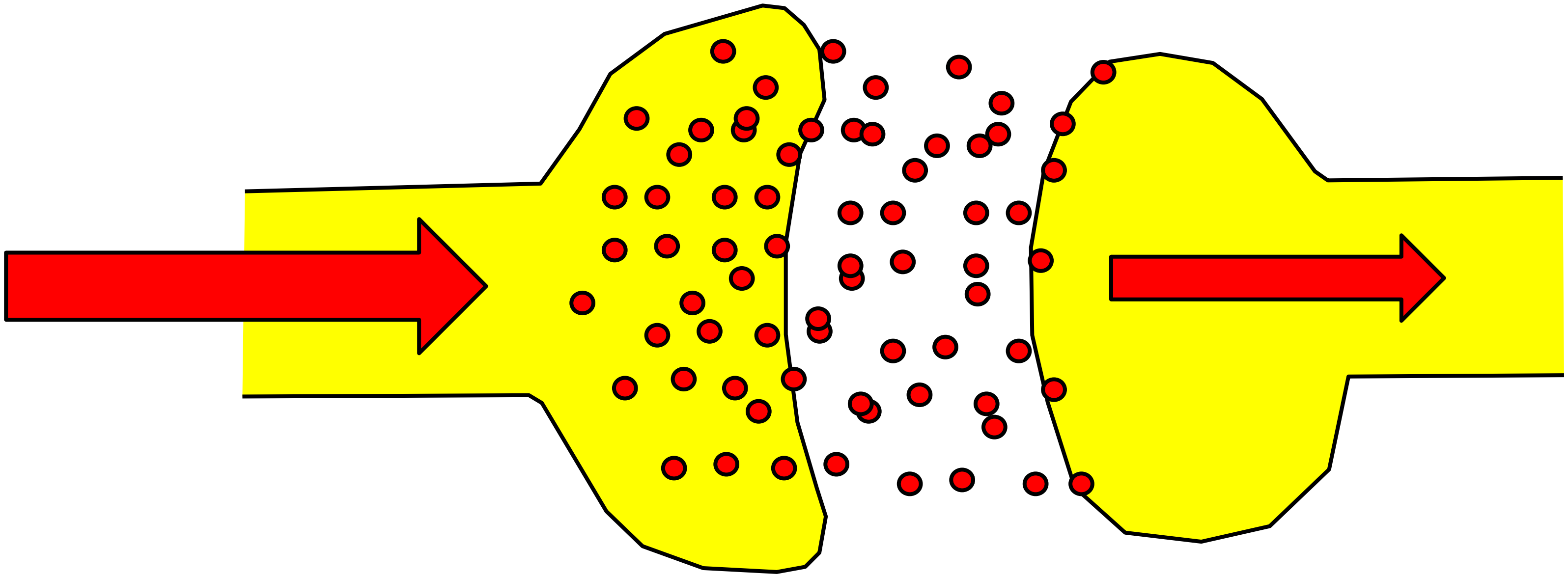
Human Brain Project

Spiking Neural Networks



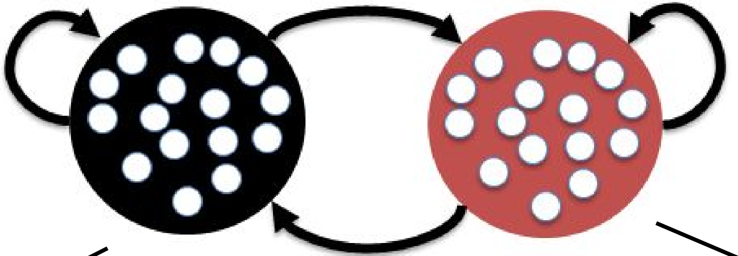
SpiNNaker

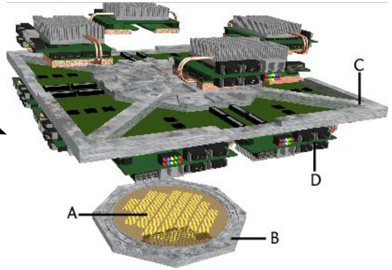
Spiking Neural Networks



What is PyNN?

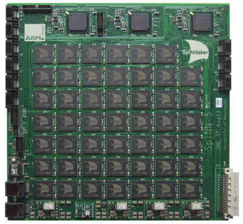
Human Brain Project





nest::
initiative

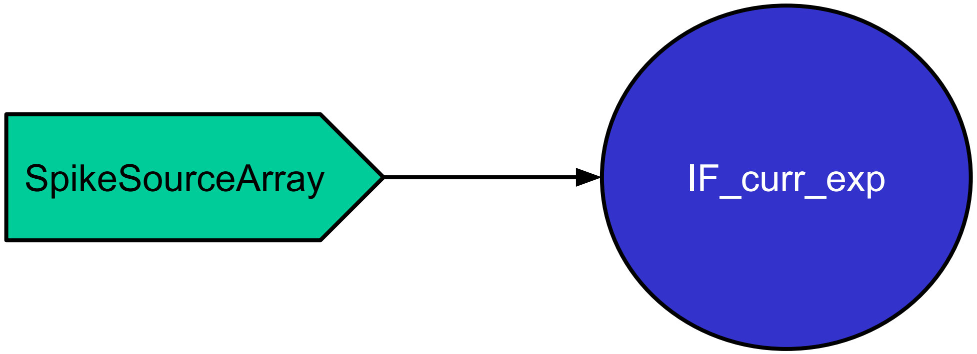
BRIAN



NEURON
for computer simulations of neurons and neural networks

SpiNNaker

A Simple PyNN Network



A Simple PyNN Network

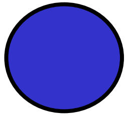
```
import pyNN.spiNNaker as p
```

A Simple PyNN Network

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
```

A Simple PyNN Network

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label=“pop_1”)
```

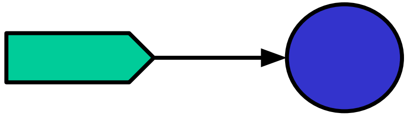


A Simple PyNN Network

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                     {'spike_times': [0]}, label="input")
```

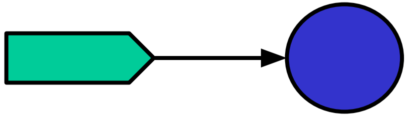
A Simple PyNN Network

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                     {'spike_times': [0]}, label="input")
input_proj = p.Projection(input, pop_1, p.OneToOneConnector(
    weights=5.0, delays=1))
```



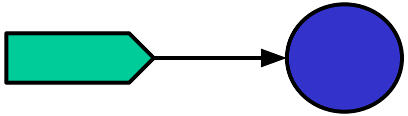
A Simple PyNN Network

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                     {'spike_times': [0]}, label="input")
input_proj = p.Projection(input, pop_1, p.OneToOneConnector(
    weights=5.0, delays=1))
pop_1.record()
pop_1.record_v()
```



A Simple PyNN Network

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                     {'spike_times': [0]}, label="input")
input_proj = p.Projection(input, pop_1, p.OneToOneConnector(
    weights=5.0, delays=1))
pop_1.record()
pop_1.record_v()
p.run(10)
```



Edit ~/.spynnaker.cfg

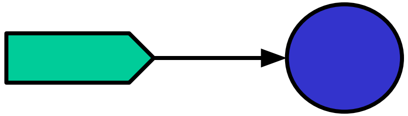
```
[Machine]
#-------
# Information about the target SpiNNaker board or machine:
# machineName:     The name or IP address of the target board
# version:         Version of the Spinnaker Hardware Board (1-5)
# machineTimeStep: Internal time step in simulations in usecs.
# timeScaleFactor: Change this to slow down the simulation time
#                      relative to real time.
#-------
machineName     = None
version         = None
#machineTimeStep = 1000
#timeScaleFactor = 1
```

Edit ~/.spynnaker.cfg

```
[Machine]
#-------
# Information about the target SpiNNaker board or machine:
# machineName:     The name or IP address of the target board
# version:         Version of the Spinnaker Hardware Board (1-5)
# machineTimeStep: Internal time step in simulations in usecs.
# timeScaleFactor: Change this to slow down the simulation time
#                      relative to real time.
#-------
machineName     = 192.168.240.253
version         = 3
#machineTimeStep = 1000
#timeScaleFactor = 1
```

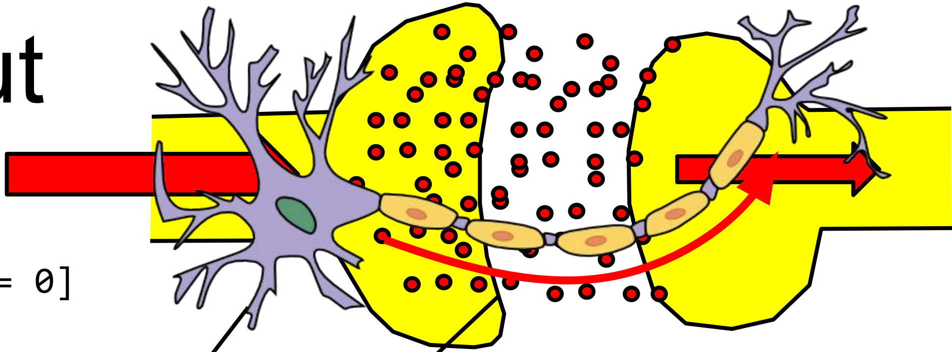
A Simple PyNN Network

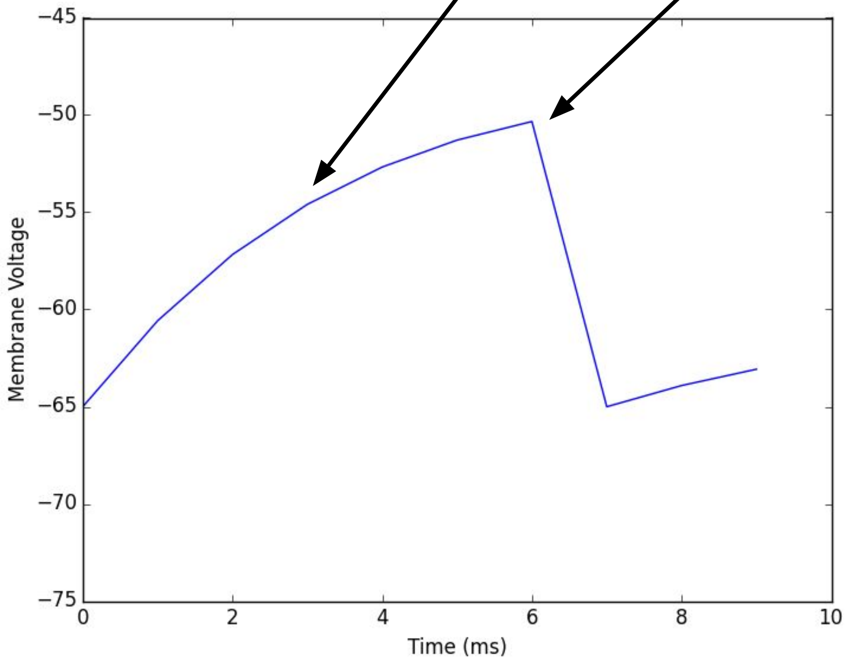
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label=“pop_1”)
input = p.Population(1, p.SpikeSourceArray,
                     {‘spike_times’: [0]}, label=“input”)
input_proj = p.Projection(input, pop_1, p.OneToOneConnector(
    weights=5.0, delays=1))
pop_1.record()
pop_1.record_v()
p.run(10)
spikes = pop_1.getSpikes()
v = pop_1.get_v()
```



Plotting Output

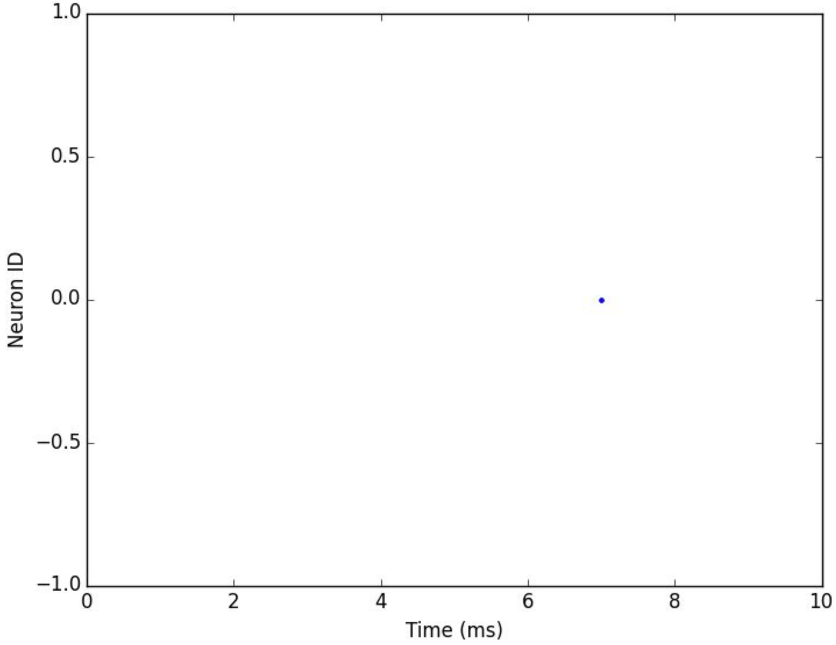
```
import pylab
time = [i[1] for i in v if i[0] == 0]
membrane_voltage = [i[2] for i in v if i[0] == 0]
pylab.plot(time, membrane_voltage)
pylab.xlabel("Time (ms)")
pylab.ylabel("Membrane Voltage")
pylab.axis([0, 10, -75, -45])
pylab.show()
```





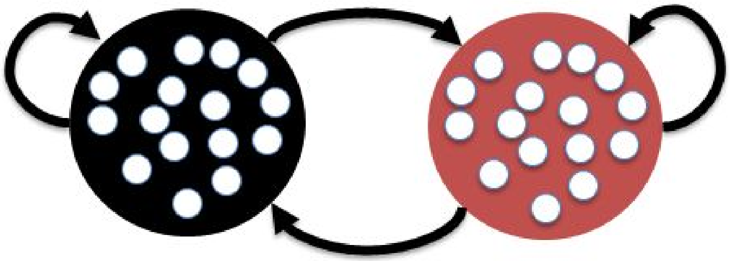
Plotting Output

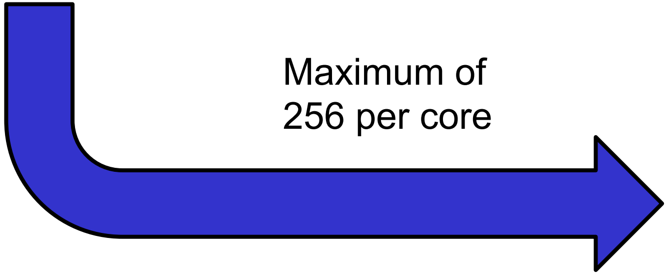
```
import pylab
spike_time = [i[1] for i in spikes]
spike_id = [i[0] for i in spikes]
pylab.plot(spike_time, spike_id, ".")
pylab.xlabel("Time (ms)")
pylab.ylabel("Neuron ID")
pylab.axis([0, 10, -1, 1])
pylab.show()
```

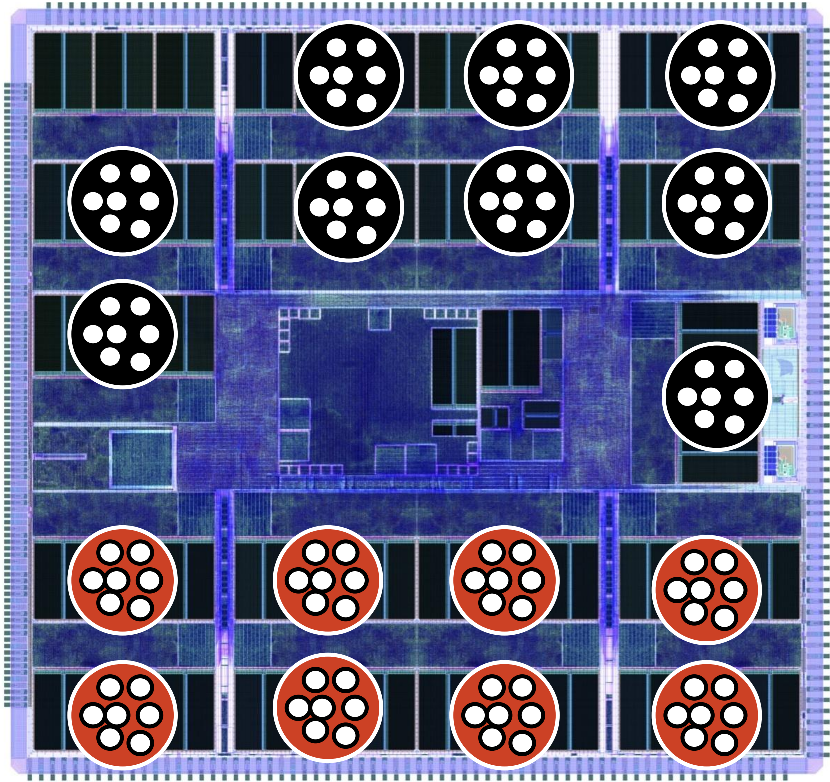


Limitations of PyNN on SpiNNaker: Neurons Per Core

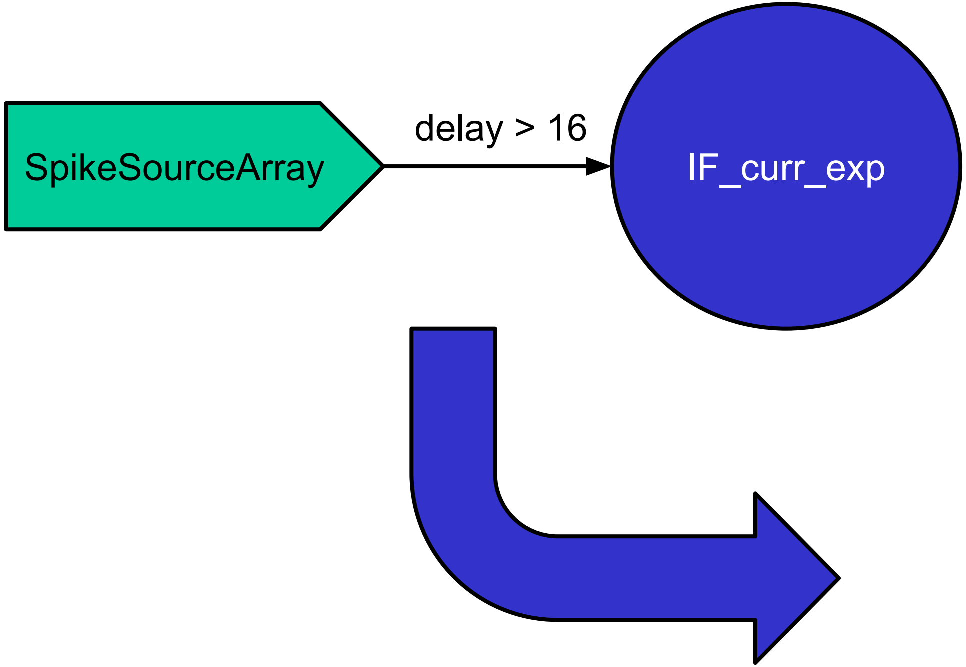
Human Brain Project

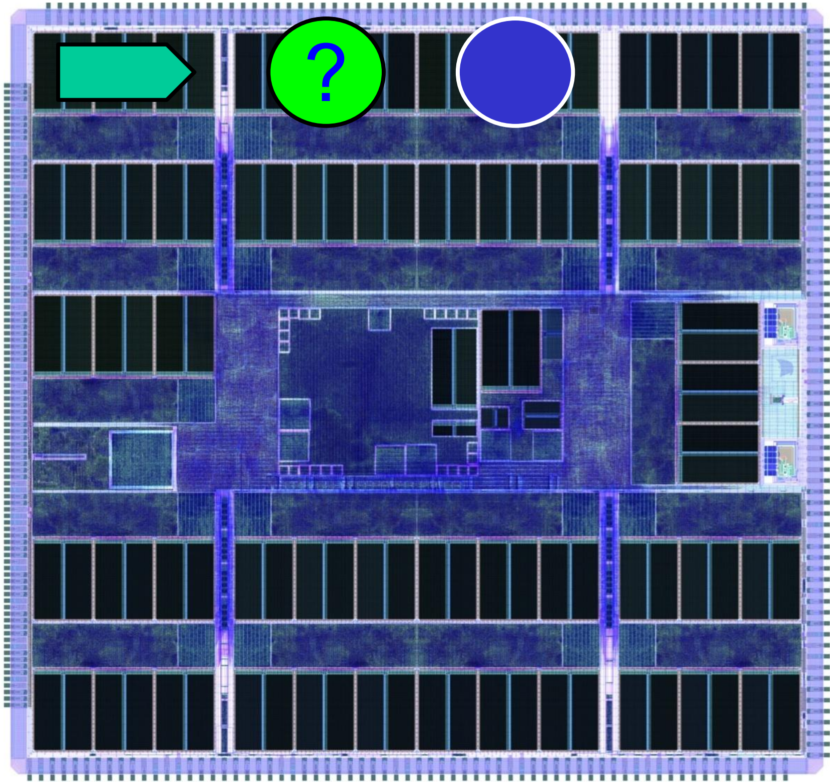






Limitations of PyNN on SpiNNaker:
Number of cores available

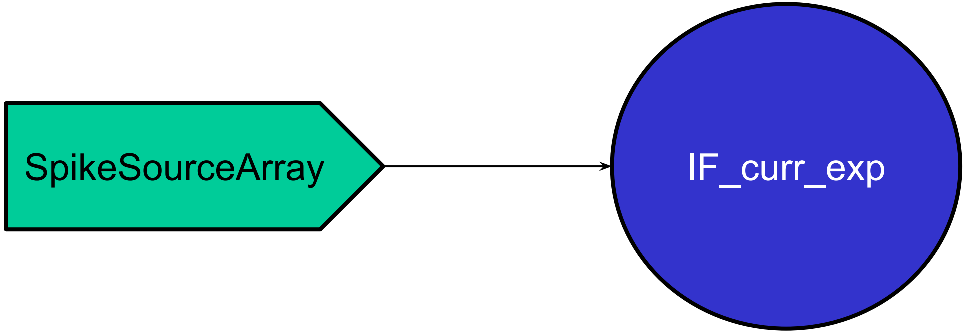




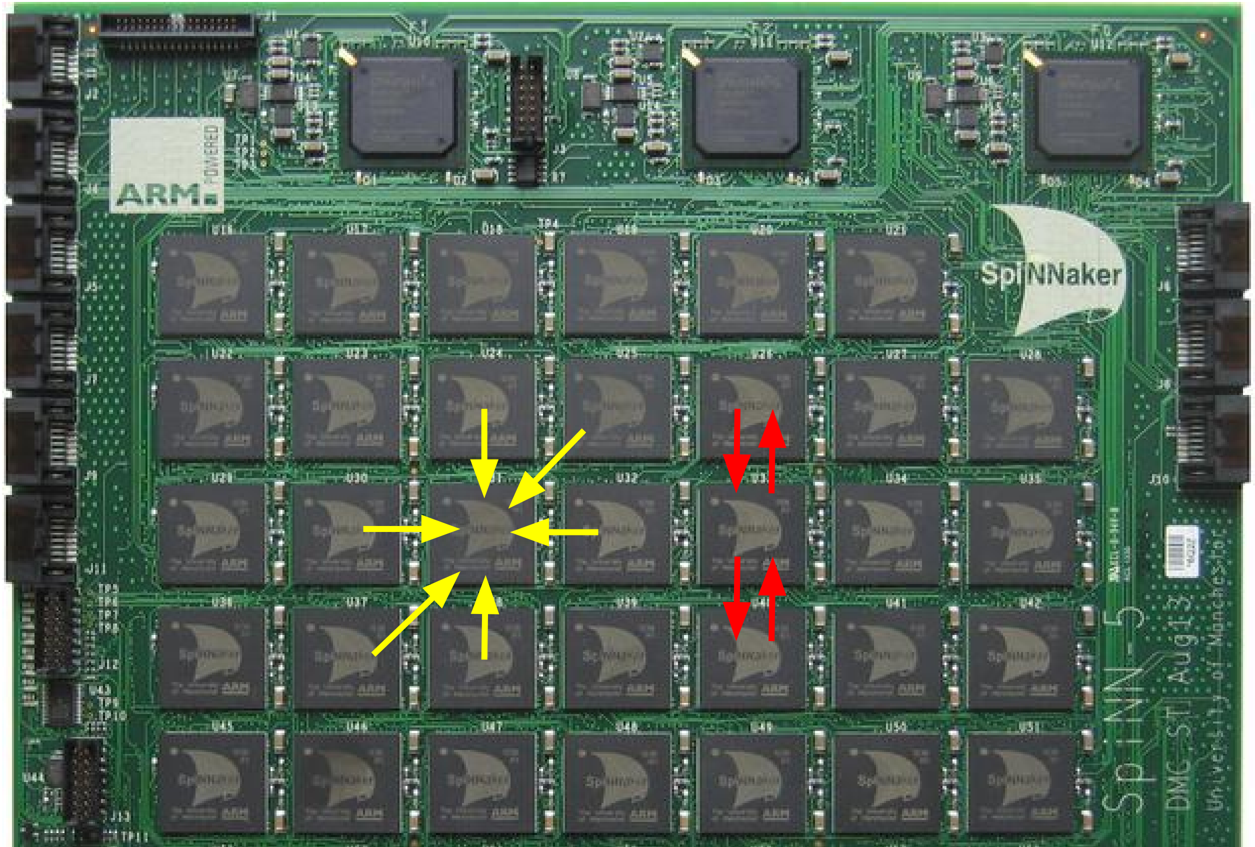
Limitations of PyNN on SpiNNaker:
Limited memory for recording/ playback

1. A 4 chip SpiNNaker board contains 480 Meg of storage.
2. Let's assume the SpikeSourceArray requires 120 Meg for the spike data.
3. Let's assume that the If_curr_exp contains 12288 atoms and therefore is split between 3 chips with 48 cores.
4. Let's assume the the If_curr_exp requires 2 Meg for static data.
5. Let's assume your recording Spikes, gsyn, voltage from the If_curr_exp and running for 1 hour.
6. Each core of the If_curr_exp needs 17 bytes to store gsyn, spikes, and voltage per millisecond.
7. This means each core can run for 6 minutes before it runs out of memory.

We have functionality to fix this!

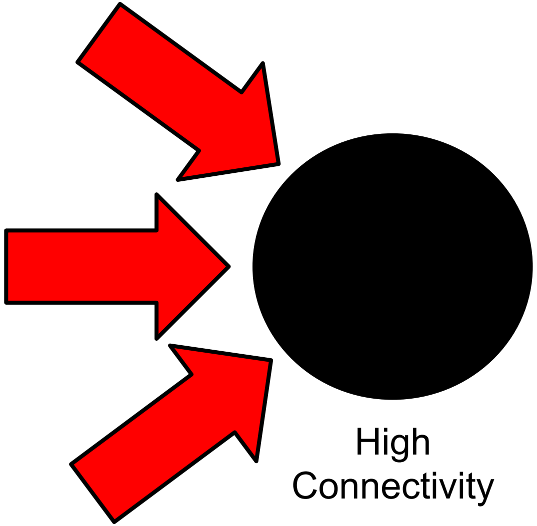


Limitations of PyNN on SpiNNaker:
Dropped Packets (Missing Spikes)

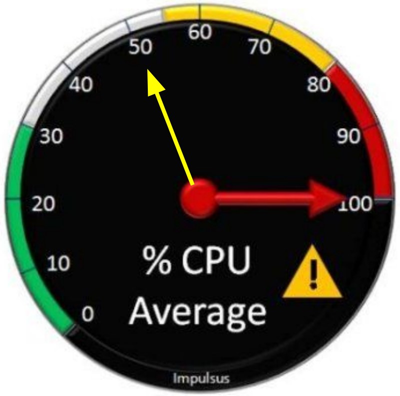


SpiNNaker-Specific PyNN

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p.set_number_of_neurons_per_core(p.IF_curr_exp, 100)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
```



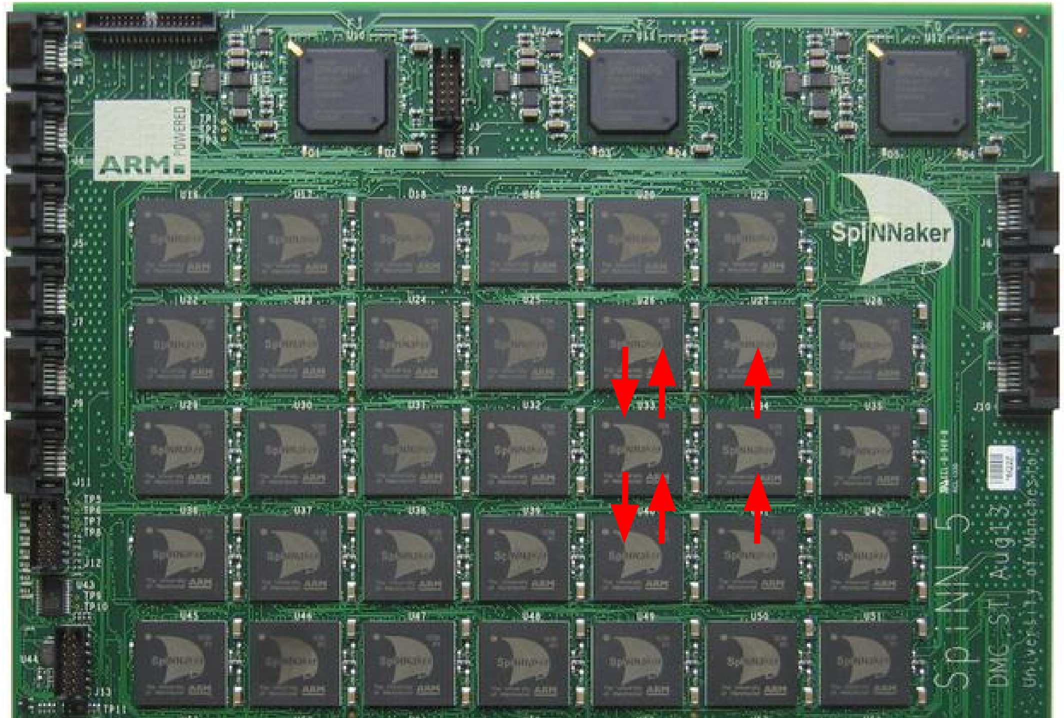
High Connectivity



Plasticity

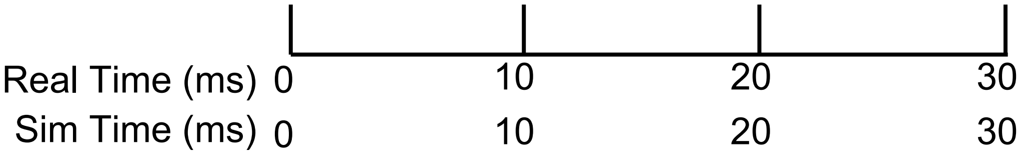
SpiNNaker-Specific PyNN

```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
pop_1 = p.Population(1, p.IF_curr_exp, {}, label=“pop_1”)
pop_1.add_placement_constraint(x=1, y=1)
```



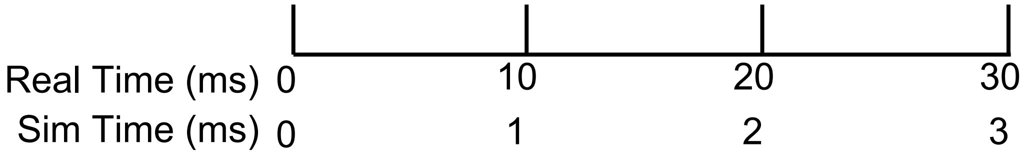
Configuration with spynnaker.cfg

```
[Machine]
machineName       = None
version           = None
timeScaleFactor   = 1
```

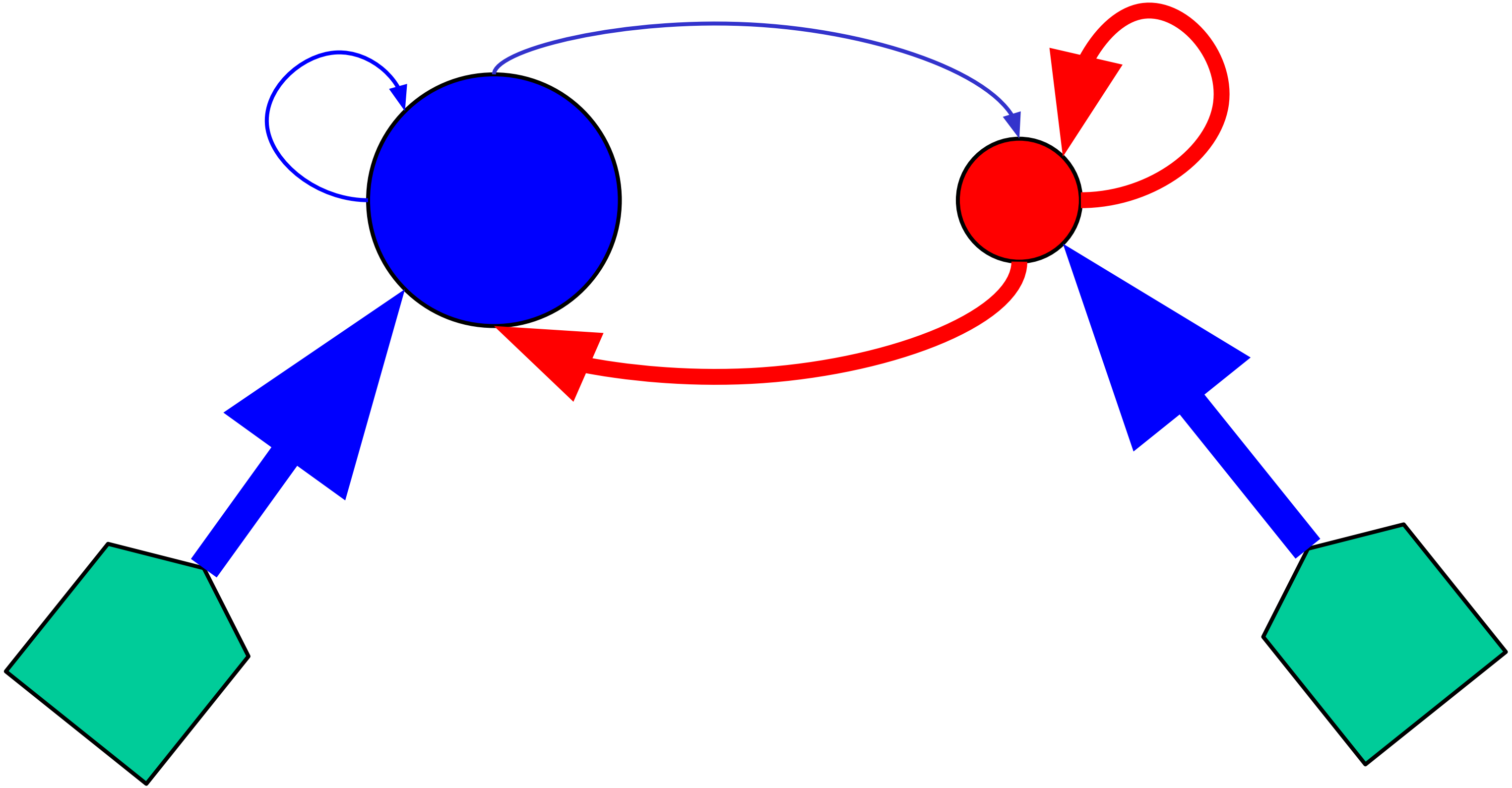


Configuration with spynnaker.cfg

```
[Machine]
machineName       = None
version           = None
timeScaleFactor   = 10
```



Balanced Random Network



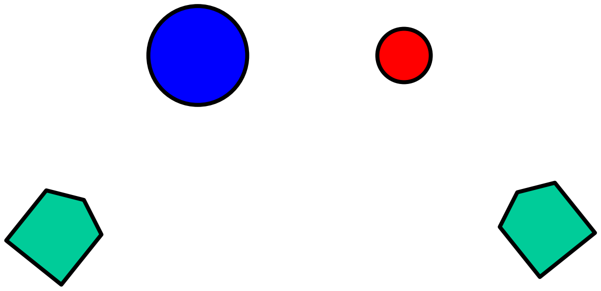
Balanced Random Network

```
import pyNN.spiNNaker as p
import pylab
from pyNN.random import RandomDistribution

p.setup(timestep=0.1)
n_neurons = 1000
n_exc = int(round(n_neurons * 0.8))
n_inh = int(round(n_neurons * 0.2))
```

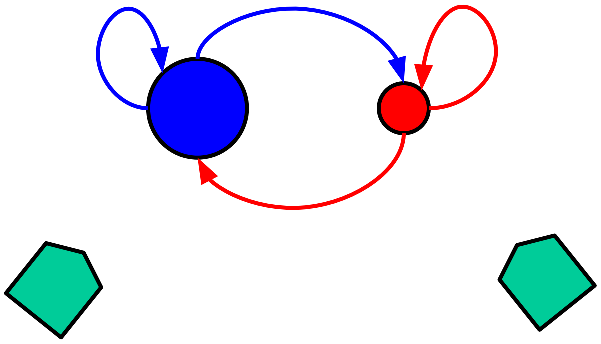
Balanced Random Network

```
pop_exc = p.Population(n_exc, p.IF_curr_exp, {},
                       label="Excitatory")
pop_inh = p.Population(n_inh, p.IF_curr_exp, {},
                       label="Inhibitory")
stim_exc = p.Population(n_exc, p.SpikeSourcePoisson,
                        {"rate": 10.0}, label="Stim_Exc")
stim_inh = p.Population(n_inh, p.SpikeSourcePoisson,
                        {"rate": 10.0}, label="Stim_Inh")
```



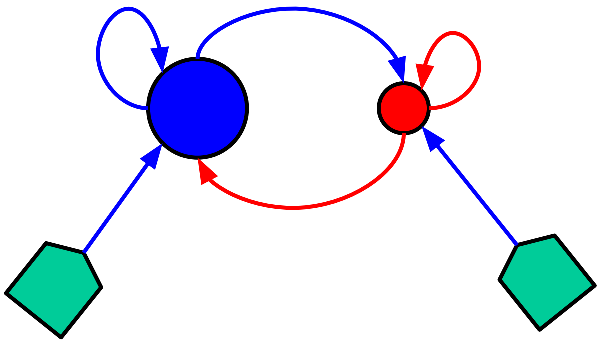
Balanced Random Network

```
conn_exc = p.FixedProbabilityConnector(0.1, weights=0.2,
                                       delays=2.0)
conn_inh = p.FixedProbabilityConnector(0.1, weights=-1.0,
                                       delays=2.0)
p.Projection(pop_exc, pop_exc, conn_exc, target="excitatory")
p.Projection(pop_exc, pop_inh, conn_exc, target="excitatory")
p.Projection(pop_inh, pop_inh, conn_inh, target="inhibitory")
p.Projection(pop_inh, pop_exc, conn_exc, target="inhibitory")
```



Balanced Random Network

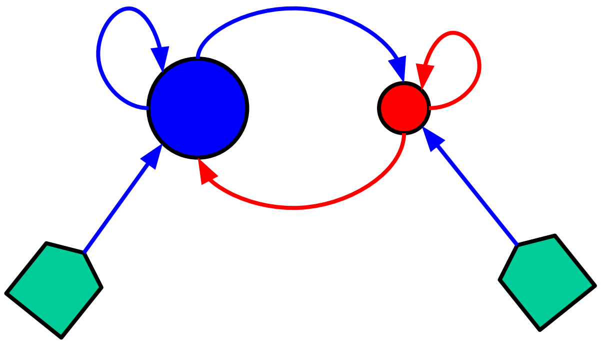
```
delays_stim = RandomDistribution("uniform", [1.0, 1.6])
conn_stim = p.OneToOneConnector(weights=2.0,
                                delays=delays_stim)
p.Projection(stim_exc, pop_exc, conn_stim, target="excitatory")
p.Projection(stim_inh, pop_inh, conn_stim, target="excitatory")
```



Balanced Random Network

```
pop_exc.initialize("v", RandomDistribution("uniform",
                                            [-65.0, -55.0]))
pop_inh.initialize("v", RandomDistribution("uniform",
                                            [-65.0, -55.0]))

pop_exc.record()
p.run(1000)
```



Balanced Random Network

```
spikes = pop_exc.getSpikes()
pylab.plot([i[1] for i in spikes], [i[0] for i in spikes], ".")
pylab.xlabel("Time (ms)")
pylab.ylabel("Neuron ID")
pylab.axis([0, 1000, -1, n_exc + 1])
pylab.show()
```