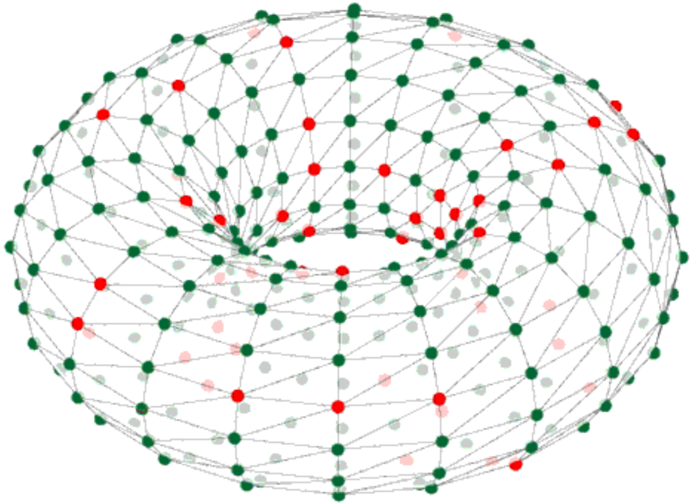
Simple Data I/O and visualisation



Alan B Stokes

SpiNNaker Workshop
September 2015

European Research Council
Established by the European Commission

Human Brain Project

MANCHESTER 1824

Contents

Summaries

- Standard PyNN support summary.

External Device Plugin

- What is it, why we need it?
- Usage caveats.

Input

- Injecting spikes into a executing PyNN script.

Output

- Live streaming of spikes from a PyNN script.

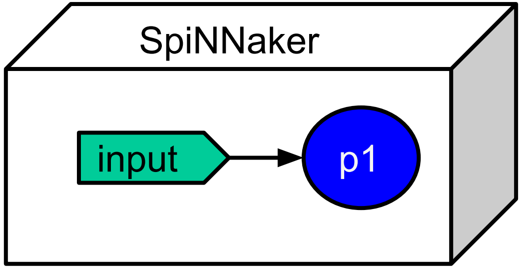
Visualisation

- Live visualisation.

Standard PyNN support (Summary)

- Supports post execution gathering of certain attributes:
  - aka transmitted spikes, voltages etc.

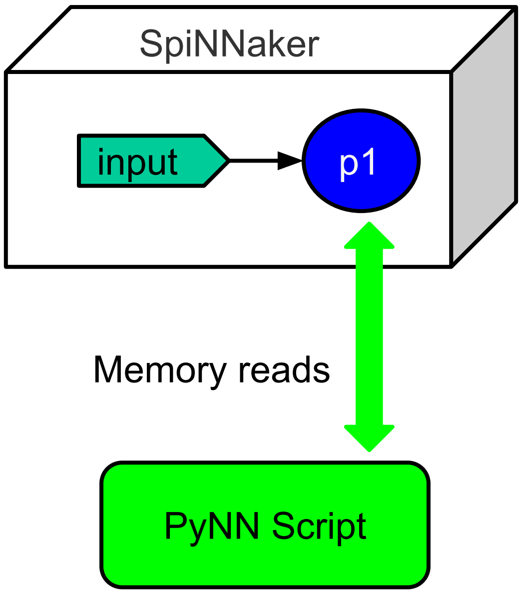
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                {'spike_times': [0]}, label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(
   weights=5.0, delays=1))
p1.record()
p1.record_v()
```



Standard PyNN support (Summary)

- Supports post execution gathering of certain attributes:
  - aka transmitted spikes, voltages etc.

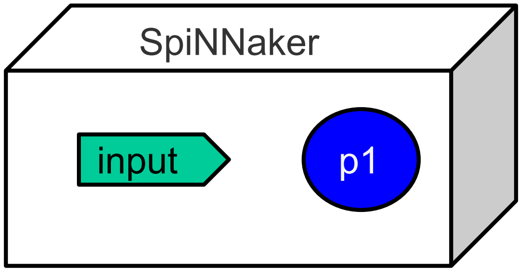
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label=“pop_1”)
input = p.Population(1, p.SpikeSourceArray,
                {‘spike_times’: [0]}, label=“input”)
input_proj = p.Projection(input, p1, p.OneToOneConnector(
  weights=5.0, delays=1))
p1.record()
p1.record_v()
p.run(5000)
spikes = p1.getSpikes()
v = p1.get_v()
```



Standard PyNN support (Summary)

- Supports spike sources of:
  - Spike Source Array, Spike source poisson.

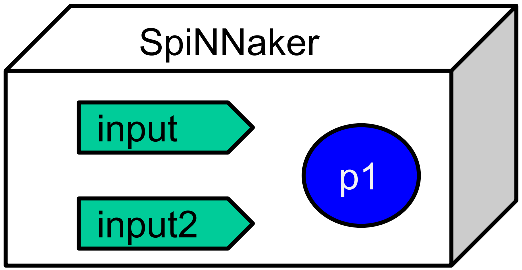
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                {'spike_times': [0]}, label="input")
```



Standard PyNN support (Summary)

- Supports spike sources of:
  - Spike Source Array, Spike source poisson.

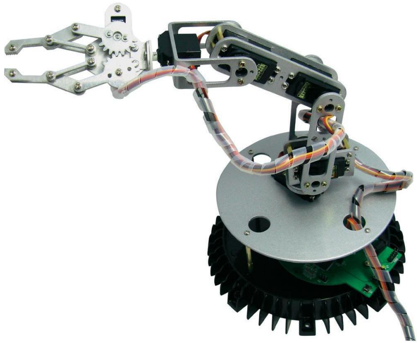
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
               {'spike_times': [0]}, label="input")
input2 = p.Population(1, p.SpikeSourcePoisson,
               {'rate':100, 'duration':50}, label='input2')
```



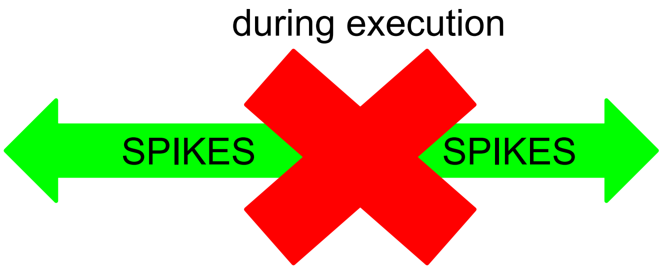
Standard PyNN support (Summary)

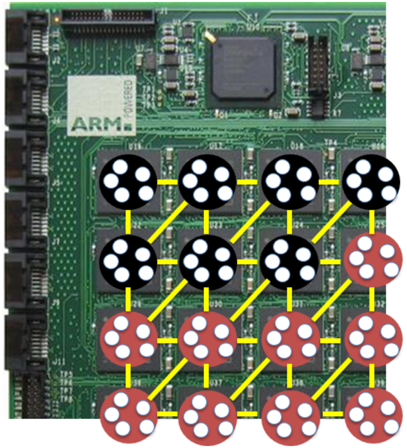
Restrictions

1. Recorded data is stored on SDRAM on each chip.
2. Data to be injected has to be known up-front, or rate based.
3. No support for closed loop execution with external devices.



http://www.conrad.com/ce/en/product/191516/Arexx-RA1-PRO-Metallic-Robot-Arm



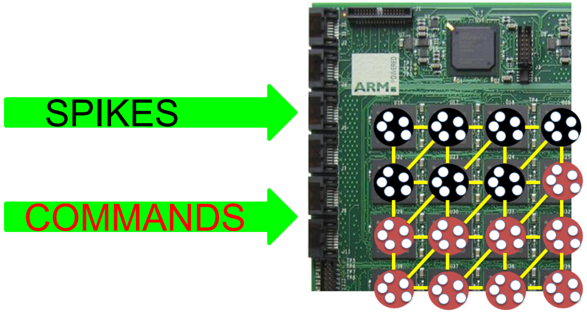


External Device Plugin

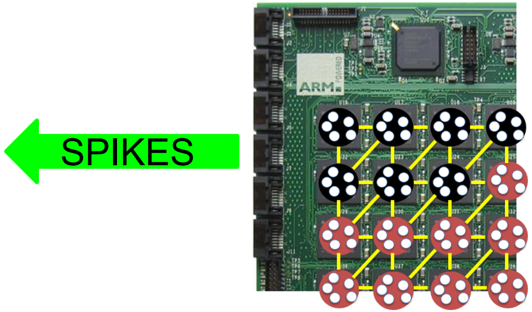
Why? what?

1. Contains functionality for PyNN scripts.
2. Not official PyNN!!!

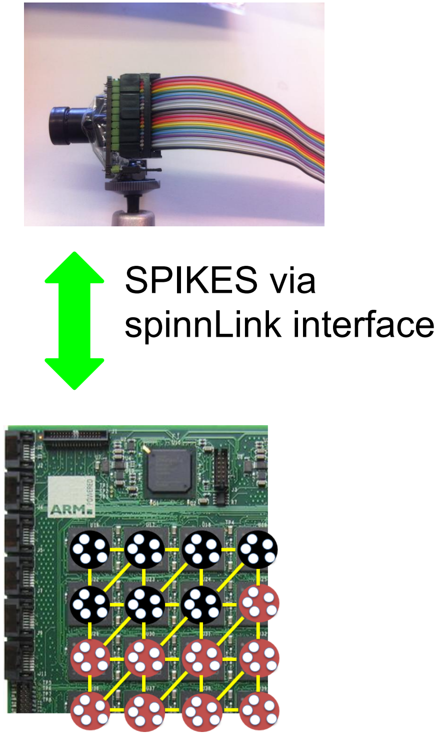
What does it Includes?



1. Live injection of events and commands into SpiNNaker



2. Live streaming of events from populations.



3. External devices support: Covered on Thursday at 13:00

External Device Plugin

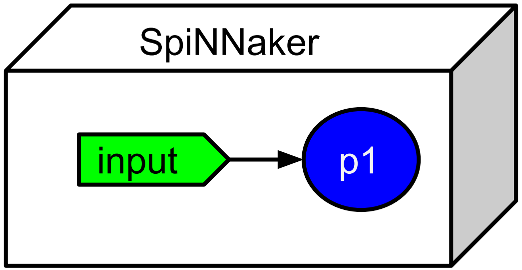
Caveats:

- Injection and live output currently only usable only with the ethernet connection,
- Limited bandwidth of:
    - A small number of spikes per millisecond time step, per ethernet,
    - Shared with both injection and live output,
- Best effort communication,
- Has a built in latency,
- Spinnaker commands not supported by other simulators,
- Loss of cores for injection and live output support,
- You can only feed a live population to one place.

Injecting spikes into PyNN scripts

PyNN script changes

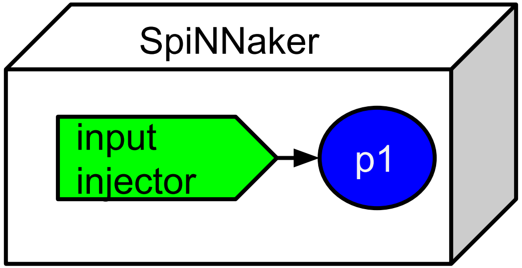
```
import pyNN.spiNNaker as p

p.setup(timestep=1.0)
p1 = 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, p1, p.OneToOneConnector(
    weights=5.0, delays=1))
# loop(synfire connection)
loop_forward = list()
for i in range(0, n_neurons - 1):
        loop_forward.append((i, (i + 1) % n_neurons, weight_to_spike, 3))
Frontend.Projection(pop_forward, pop_forward, Frontend.FromListConnector(loop_forward))
```



Injecting spikes into PyNN scripts

PyNN script changes: Declaring an injector population

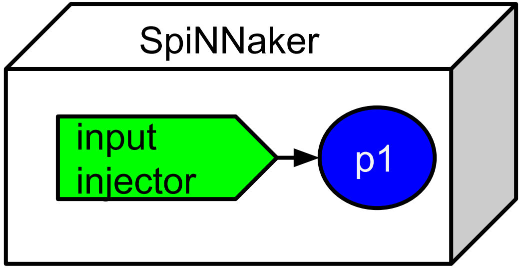
```
import pyNN.spiNNaker as p
import spynnaker_external_devices_plugin.pyNN as ExternalDevices
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input_injector = p.Population(1, ExternalDevices.SpikeInjector,
                              {'port':95768}, label="injector)
input_proj = p.Projection(input_injector, p1, p.OneToOneConnector(
   weights=5.0, delays=1))
# loop(synfire connection)
loop_forward = list()
for i in range(0, n_neurons - 1):
        loop_forward.append((i, (i + 1) % n_neurons, weight_to_spike, 3))
Frontend.Projection(pop_forward, pop_forward, Frontend.FromListConnector(loop_forward))
```

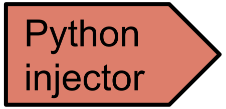


Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

```
.............
# create python injector
def send_spike(label, sender):
        sender.send_spike(label, 0, send_full_keys=True)
```

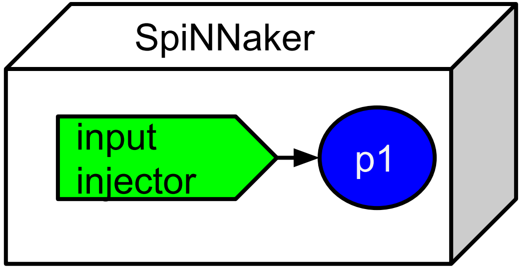


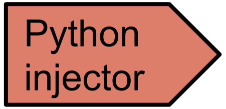


Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

```
............
# create python injector
def send_spike(label, sender):
        sender.send_spike(label, 0, send_full_keys=True)
# import python injector connection
from spynnaker_external_devices_plugin.pyNN.connections.\
spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection
```

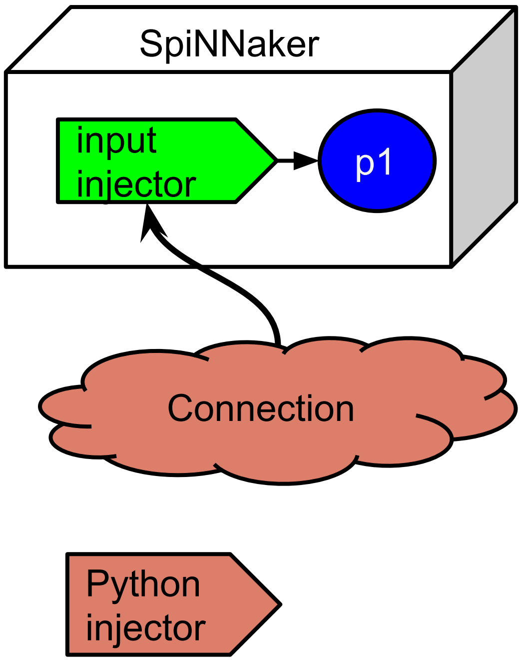




Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

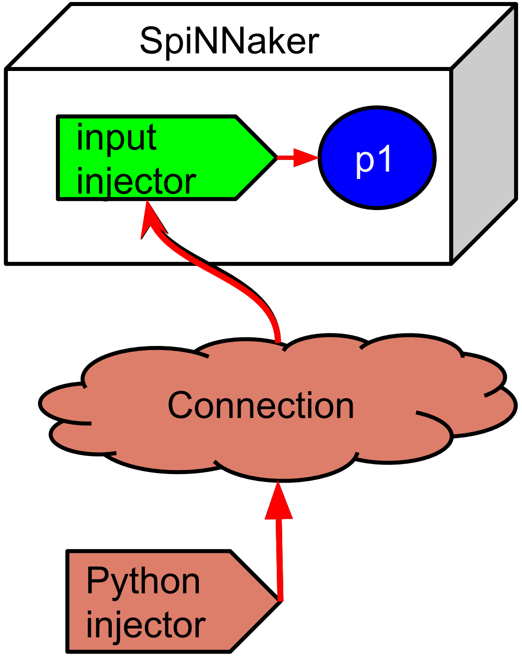
```
............
# create python injector
def send_spike(label, sender):
        sender.send_spike(label, 0, send_full_keys=True)
# import python injector connection
from spynnaker_external_devices_plugin.pyNN.connections.\
spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection
# set up python injector connection
live_spikes_connection = SpynnakerLiveSpikesConnection(
    receive_labels=None, local_port=19996, send_labels=[“spike_sender”])
```



Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

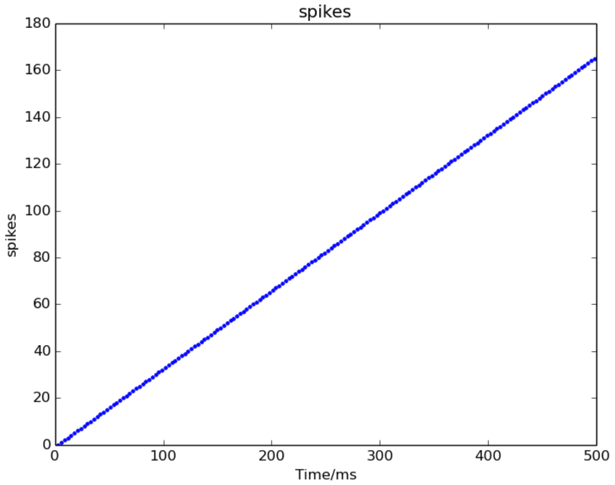
```
............
# create python injector
def send_spike(label, sender):
        sender.send_spike(label, 0, send_full_keys=True)
# import python injector connection
from spynnaker_external_devices_plugin.pyNN.connections.\
spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection
# set up python injector connection
live_spikes_connection = SpynnakerLiveSpikesConnection(
        receive_labels=None, local_port=19996, send_labels=[“spike_sender”])
# register python injector with injector connection
live_spikes_connection.add_start_callback(“spike_sender”,  send_spike)
p.run(500)
```



Injecting spikes into PyNN scripts

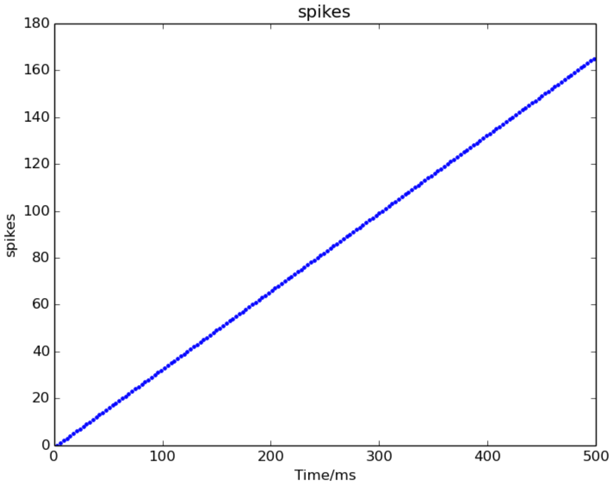
Behaviour with (SpikeSourceArray)

Behaviour with Live injection!



SAME!!!!!

BUT BORING!!!!



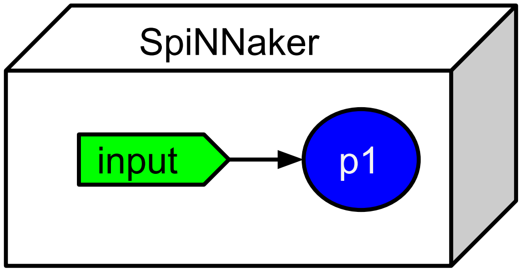
DEMO TIME!!! Injection

DEMO!!!!

Live output from PyNN scripts

PyNN script changes: declaring live output population

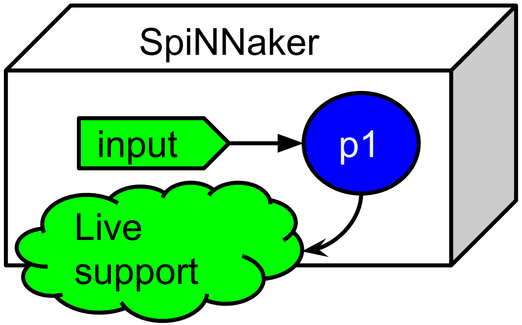
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                {'spike_times': [0]}, label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(
  weights=5.0, delays=1))
```



Live output from PyNN scripts

PyNN script changes: declaring live output population

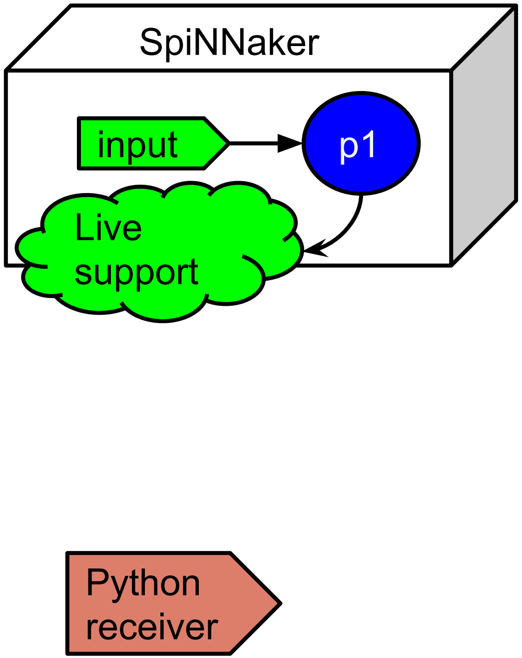
```
import pyNN.spiNNaker as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input = p.Population(1, p.SpikeSourceArray,
                {'spike_times': [0]}, label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(
  weights=5.0, delays=1))
# declare a live output for a given population.
import spynnaker_external_devices_plugin.pyNN as ExternalDevices
ExternalDevices.activate_live_output_for(p1)
```



Live output from PyNN scripts

PyNN script changes: python receiver

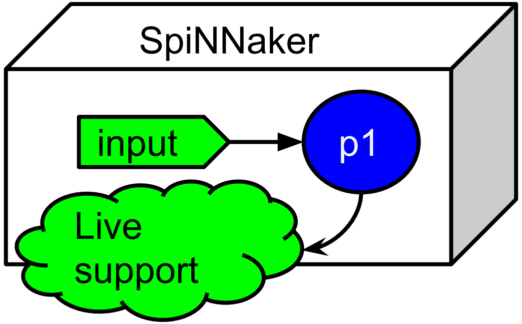
```
…………….
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
        for neuron_id in neuron_ids:
                print “Received spike at time {} from {}-{}”
                        .format(time, label, neuron_id)
```

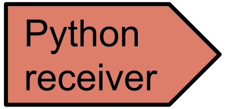


Live output from PyNN scripts

PyNN script changes: python receiver

```
…………….
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
        for neuron_id in neuron_ids:
                print "Received spike at time {} from {}-{}"
                        .format(time, label, neuron_id)
# import python live spike connection
from spynnaker_external_devices_plugin.pyNN.connections.\
spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection
```

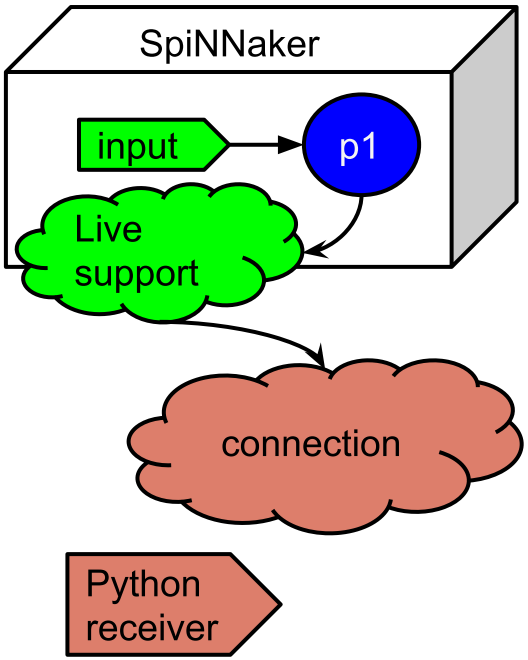




Live output from PyNN scripts

PyNN script changes: python receiver

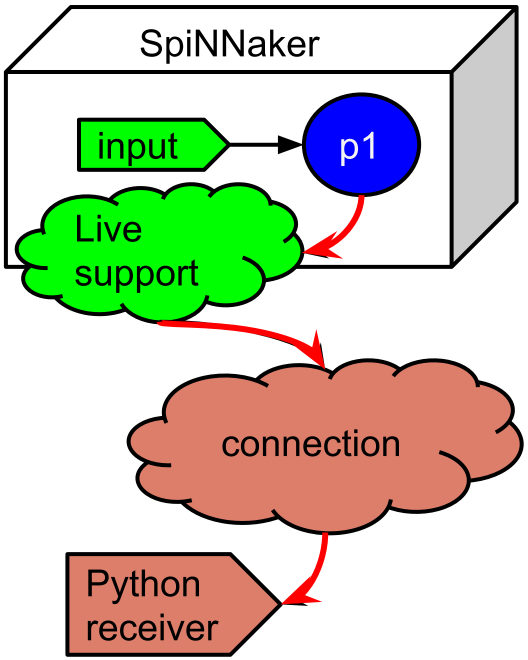
```
……………
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
        for neuron_id in neuron_ids:
                print "Received spike at time {} from {}-{}"
                        .format(time, label, neuron_id)
# import python live spike connection
from spynnaker_external_devices_plugin.pyNN.connections.\
spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection
# set up python live spike connection
live_spikes_connection = SpynnakerLiveSpikesConnection(
    receive_labels=["receiver"], local_port=19995, send_labels=None)
```



Live output from PyNN scripts

PyNN script changes: python receiver

```
...............
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
          for neuron_id in neuron_ids:
                    print “Received spike at time {} from {}-{}”
                             .format(time, label, neuron_id)
# import python live spike connection
from spynnaker_external_devices_plugin.pyNN.connections.\
spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection
# set up python live spike connection
live_spikes_connection = SpynnakerLiveSpikesConnection(
          receive_labels=[“receiver”], local_port=19995, send_labels=None)
# register python receiver with live spike connection
live_spikes_connection.add_receive_callback(“receiver”, receive_spikes)
p.run(500)
```



MANCHESTER 1824

DEMO TIME!!! receive live spikes

DEMO!!!!

Visualisation

How current supported visualisations work:

1. Uses the live output functionality as discussed previously.

2. Uses the c based receiver and is planned to be open source for users to augment with their own special visuals.

3. Currently contains raster plot support.

Visualisation

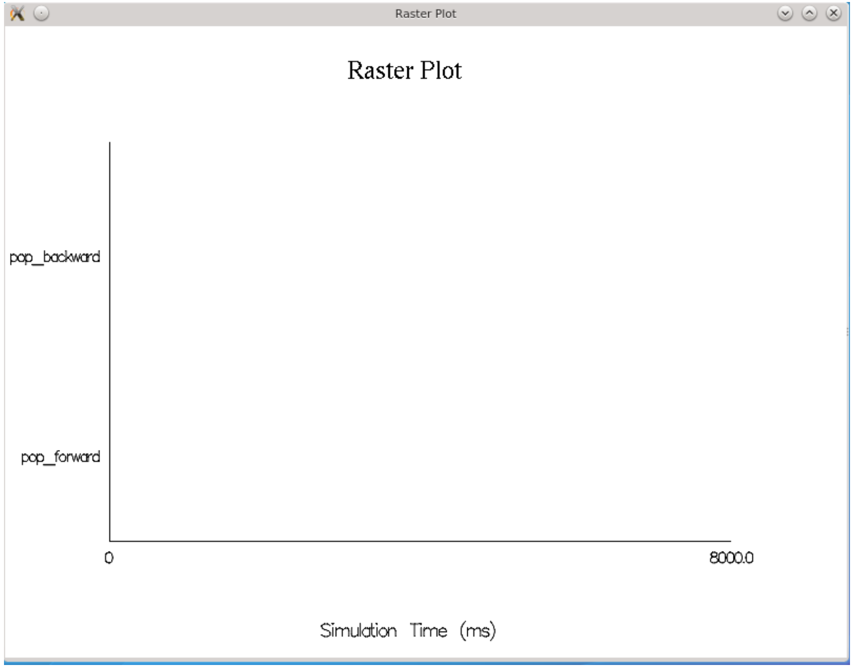
```
cspc277-visualiser-) make -f Makefile.linux
cspc277-visualiser-) .............
cspc277-visualiser-) ./vis -colour_map test_data/spikeio_colours
cspc277-visualiser-)
awaiting tool chain hand shake to say database is ready
```

Input parameters:

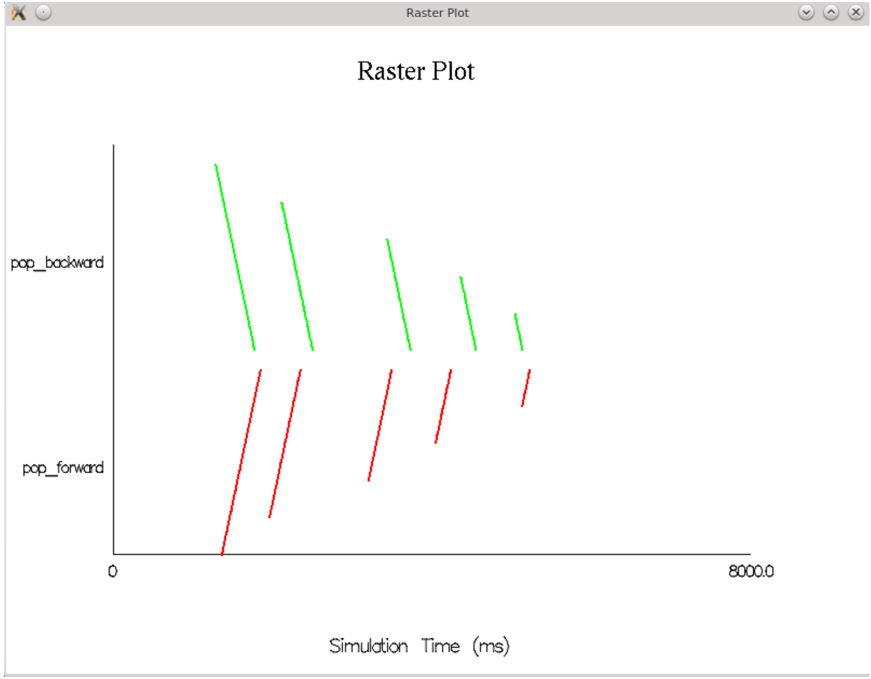
- -colour_map
  - Path to a file containing the population labels to receive, and their associated colours
- -hand_shake_port
  - optional port which the visualiser will listen to for database hand shaking
- -database
  - optional file path to where the database is located, if needed for manual configuration
- -remote_host
  - optional remote host, which will allow port triggering

Visualisation

Before run



After run



DEMO TIME!!! visualiser and injection of spikes

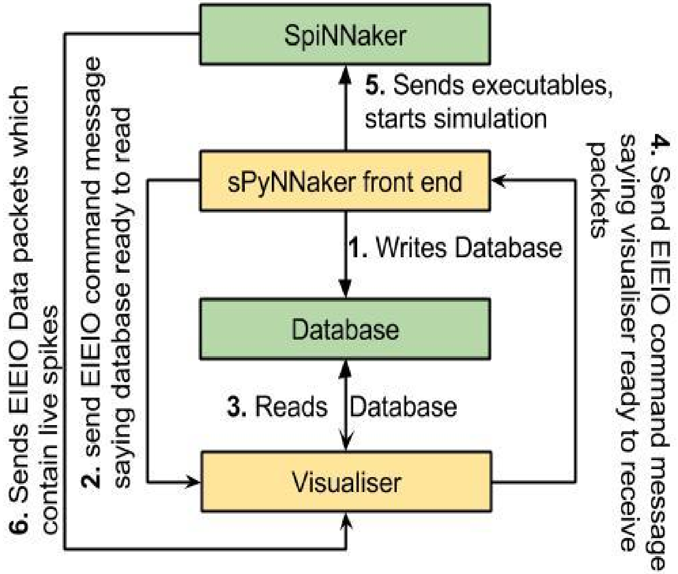
DEMO!!!!

MANCHESTER 1824

Technical Detail!!!

Notification protocol under the hood!

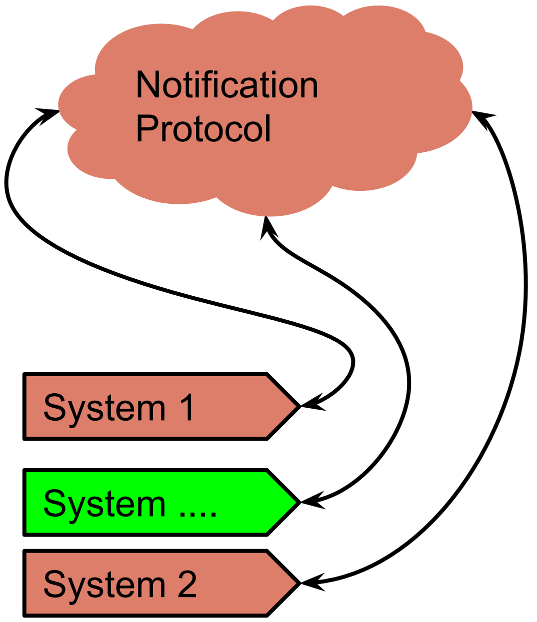
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.
- If you have more than 1 system running to inject and/or receive, then you need to register this with the notification protocol.



Injecting spikes into PyNN scripts

PyNN script changes:
registering a system to the notification protocol

```
…………
# register socket addresses for each system
p.register_database_notification_request(
    hostname=”local_host”
    notify_port=19990,
    ack_port=19992)
p.register_database_notification_request(
    hostname=”local_host”
    notify_port=19993,
    ack_port=19987)
p.register_database_notification_request(
    hostname=”local_host”
    notify_port=19760,
    ack_port=19232)
```



Thanks for listening

Any questions?!