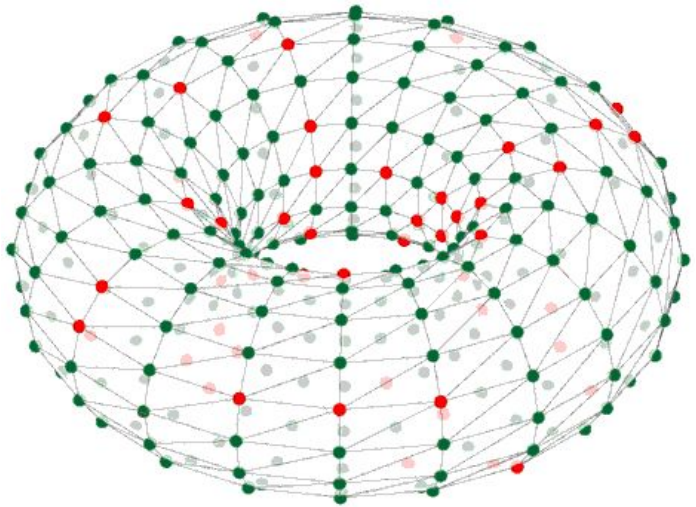
Simple Data I/O and visualisation



Alan Stokes, Andrew Rowley

SpiNNaker Workshop
October 2017

European Research Council
Established by the European Commission
Human Brain Project
EPSRC

MANCHESTER 1824

Contents

Summaries

- Standard PyNN support summary.

External Device module

- 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.

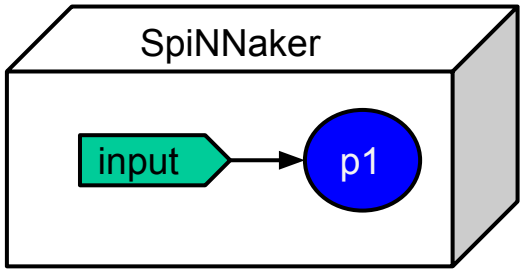
Visualisation

- Live visualisation.

Standard PyNN support (Summary)

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

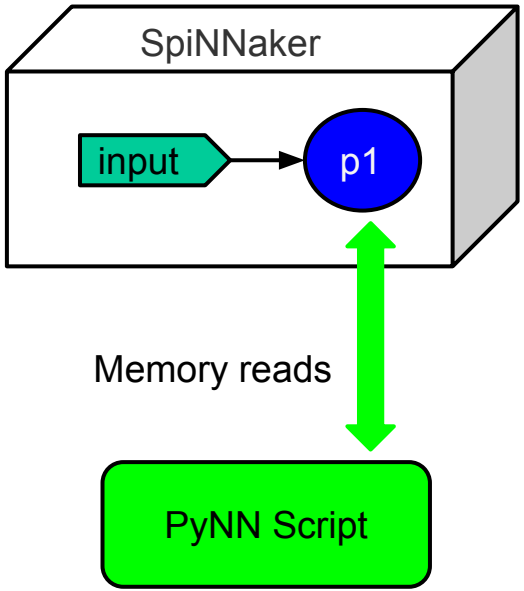
```
import spynnaker8 as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp(), label="pop_1")
input = p.Population(1, p.SpikeSourceArray(spike_times=[0]),
                         label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(),
                                 p.StaticSynapse(weight=0.1, delay=3))
p1.record(["spikes", "v"])
```



Standard PyNN support (Summary)

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

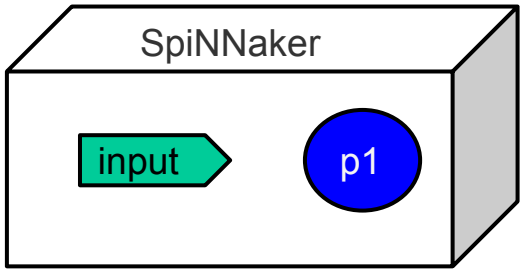
```
import spynnaker8 as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp(), label="pop_1")
input = p.Population(1, p.SpikeSourceArray(spike_times=[0]),
                         label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(),
                                 p.StaticSynapse(weight=0.1, delay=3))
p1.record(["spikes", "v"])
p.run(5000)
spikes = p1.get_data("spikes")
v = p1.get_data("v")
```



Standard PyNN support (Summary)

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

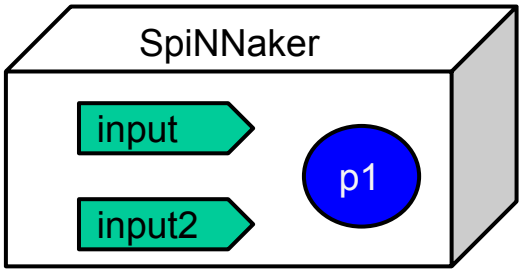
```
import spynnaker8 as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp(), label="pop_1")
input = p.Population(1, p.SpikeSourceArray(spike_times=[0])
                     , label="input")
```



Standard PyNN support (Summary)

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

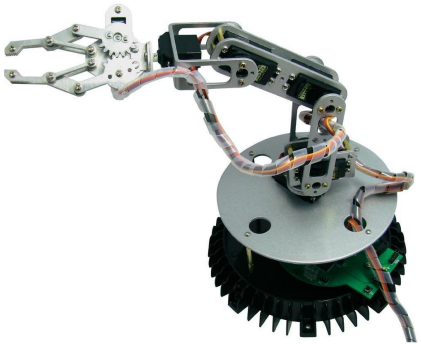
```
import spynnaker8 as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp(), label="pop_1")
input = p.Population(1, p.SpikeSourceArray(spike_times=[0]),
                         label="input")
input2 = p.Population(1, p.SpikeSourcePoisson(rate=100, duration=50),
                          label='input2')
```



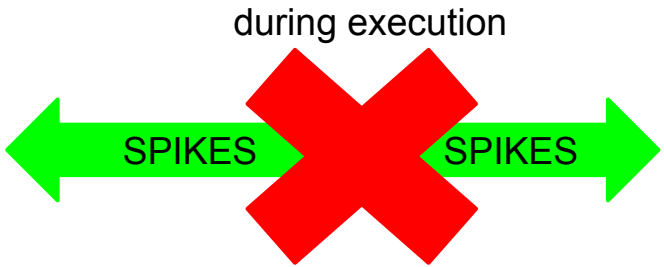
Standard PyNN support (Summary)

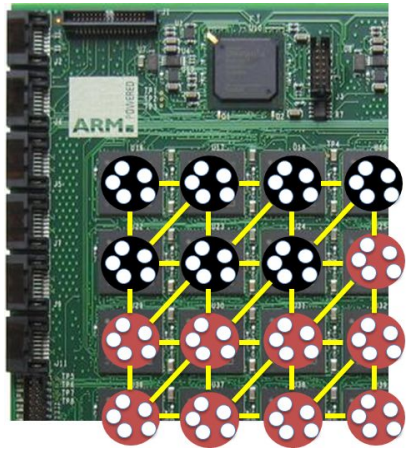
Restrictions

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.

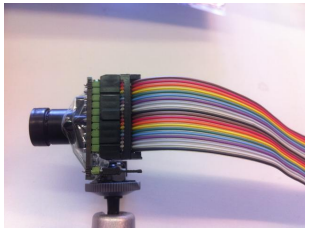


http://www.conrad.com/ce/en/product/191516/Arexx-RA1-PRO-Metallic-Robot-Arm





External Device Module

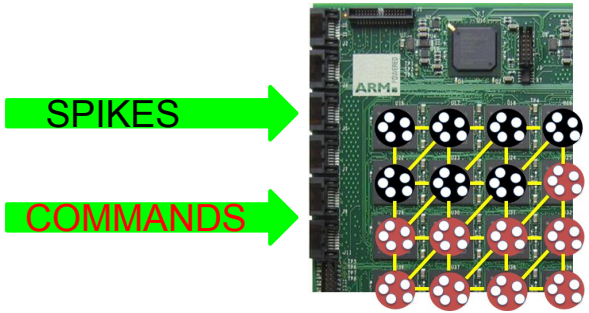


SPIKES via
spinnLink interface

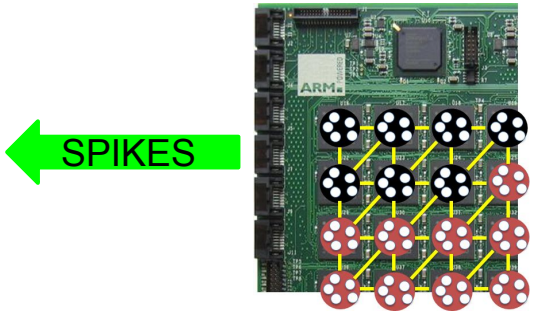
Why? what?

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

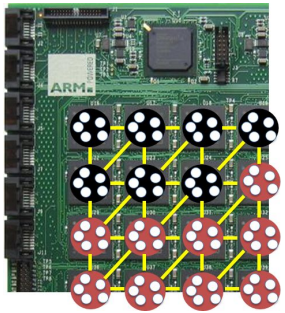
What does it Includes?



1. Live injection of events and commands into SpiNNaker



2. Live streaming of events from populations.



3. External devices support: Covered Later

External Device Plugin

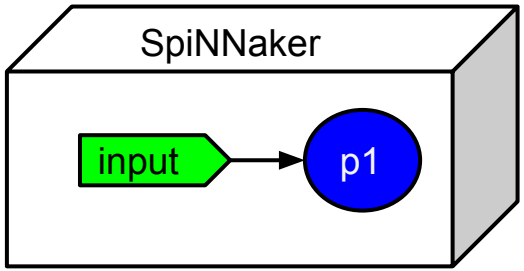
Caveats:

- 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

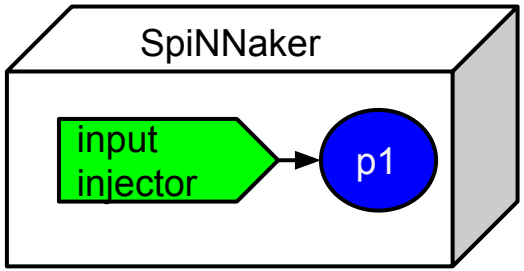
```
import spynnaker8 as p


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(),
                                 p.StaticSynapse(weight=0.1, delay=3))
# 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))
p.Projection(pop_forward, pop_forward, Frontend.FromListConnector(loop_forward))
```



Injecting spikes into PyNN scripts

PyNN script changes: Declaring an injector population

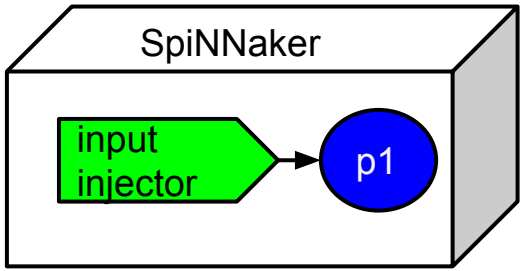
```
import spynnaker8 as p


p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp, {}, label="pop_1")
input_injector = p.Population(
    1, p.external_devices.SpikeInjector(), label="injector)
input_proj = p.Projection(input_injector, p1, p.OneToOneConnector(),
                          p.StaticSynapse(weight=0.1, delay=3))
# 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))
p.Projection(pop_forward, pop_forward, Frontend.FromListConnector(loop_forward))
```



Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

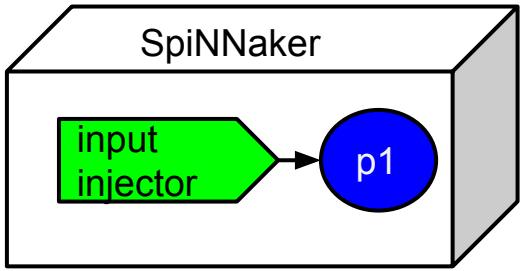
```
.....
# create python injector
run_condition = Condition()
running = True

def send_spike(label, sender):
        running = True
        run_condition.acquire()
                if running:
                        run_condition.release()
                        sender.send_spike(label, 0, send_full_keys=True)
                else:
                        run_condition.release()
```

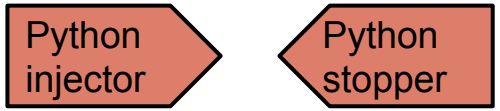


Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

```
.............
# create python stopper
def stop_flow(label, sender):
        run_condition.acquire()
        running = False
        run_condition.release()
```

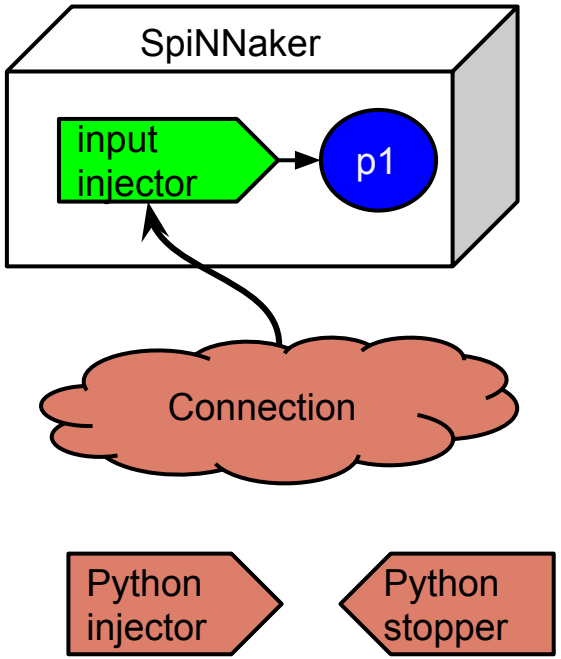




Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

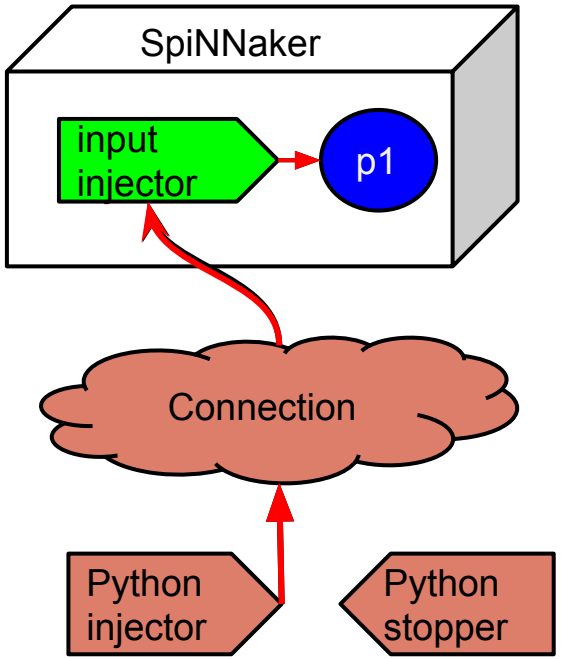
```
.............
# set up python injector connection
live_spikes_connection =
    p.external_devices.SpynnakerLiveSpikesConnection(
        send_labels=["spike_sender"])
```



Injecting spikes into PyNN scripts

PyNN script changes: Setting up python injector

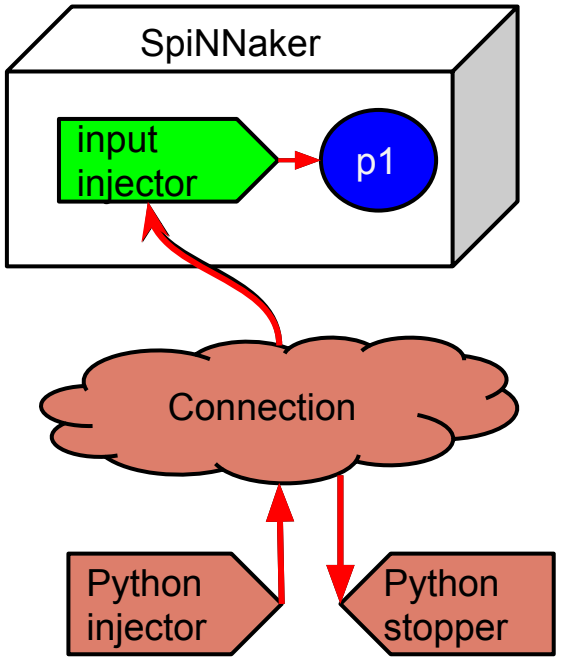
```
.............
# set up python injector connection
live_spikes_connection =
    p.external_devices.SpynnakerLiveSpikesConnection(
        receive_labels=None, local_port=19996,
        send_labels=["spike_sender"])
# register python injector with injector connection
live_spikes_connection.add_start_resume_callback(
        "spike_sender",  send_spike)
```



Injecting spikes into PyNN scripts

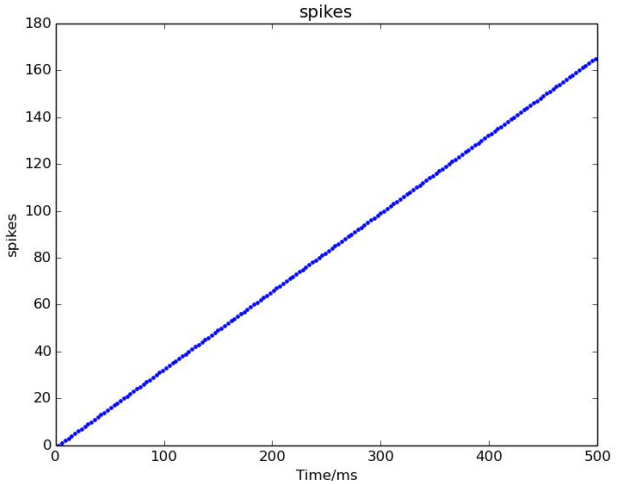
PyNN script changes: Setting up python injector

```
.....
# set up python injector connection
live_spikes_connection =
    p.external_devices.SpynnakerLiveSpikesConnection(
        receive_labels=None, local_port=19996,
        send_labels=["spike_sender"])
# register python injector with injector connection
live_spikes_connection.add_start_resume_callback(
        "spike_sender",  send_spike)
# register python stopper with connection
live_spikes_connection.add_pause_stop_callback(
        "spike_sender", stop_flow)
p.run(500)
```



Injecting spikes into PyNN scripts

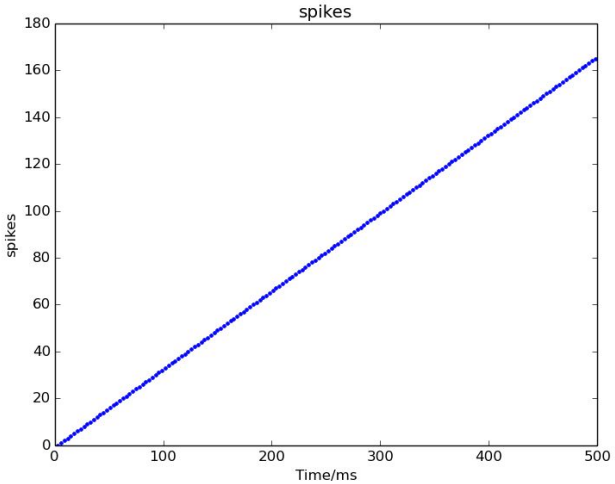
Behaviour with (SpikeSourceArray)



SAME!!!!!

BUT BORING!!!!

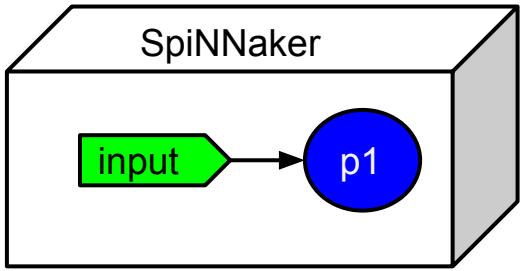
Behaviour with Live injection!



Live output from PyNN scripts

PyNN script changes: declaring live output population

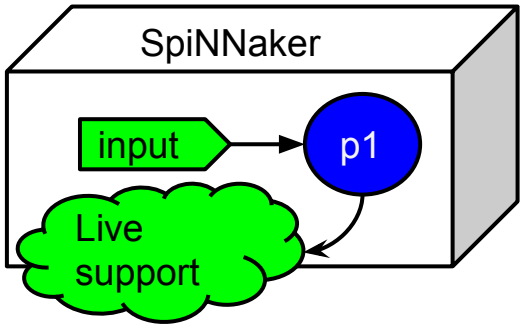
```
import spynnaker8 as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp(), label="pop_1")
input = p.Population(1, p.SpikeSourceArray(spike_times=[0]),
                         label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(),
                                 p.StaticSynapse(weight=0.1, delay=3))
```



Live output from PyNN scripts

PyNN script changes: declaring live output population

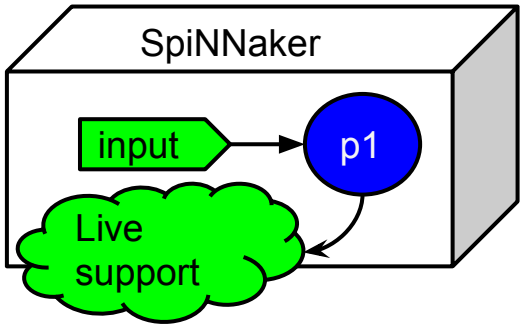
```
import import spynnaker8 as p
p.setup(timestep=1.0)
p1 = p.Population(1, p.IF_curr_exp(), label="pop_1")
input = p.Population(1, p.SpikeSourceArray(spike_times=[0]),
                         label="input")
input_proj = p.Projection(input, p1, p.OneToOneConnector(),
                                 p.StaticSynapse(weight=0.1, delay=3))
# declare a live output for a given population.
p.external_devices.activate_live_output_for(p1)
```



Live output from PyNN scripts

PyNN script changes: python receiver

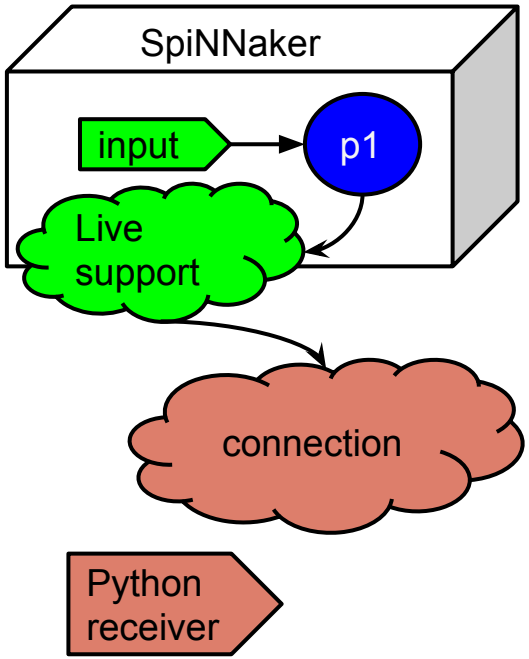
```
.....
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
    for neuron_id in neuron_ids:
        print "Received spike at time {} from {}-{}"
            .format(time, label, neuron_id)
```



Live output from PyNN scripts

PyNN script changes: python receiver

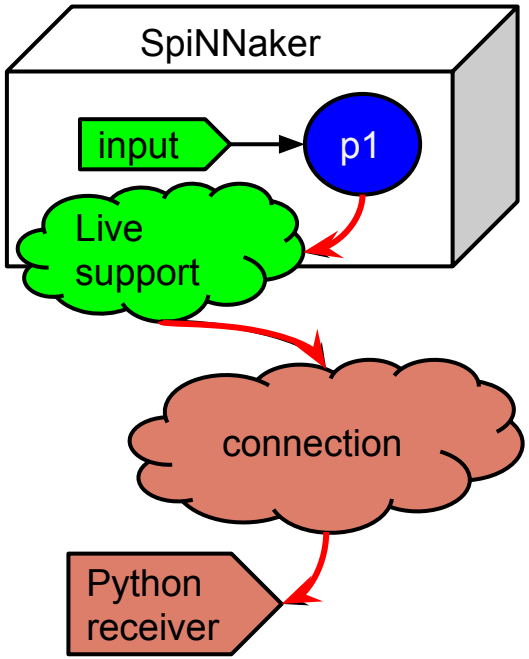
```
.....
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
        for neuron_id in neuron_ids:
                print “Received spike at time {} from {}-{}”
                        .format(time, label, neuron_id)
# set up python live spike connection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



Live output from PyNN scripts

PyNN script changes: python receiver

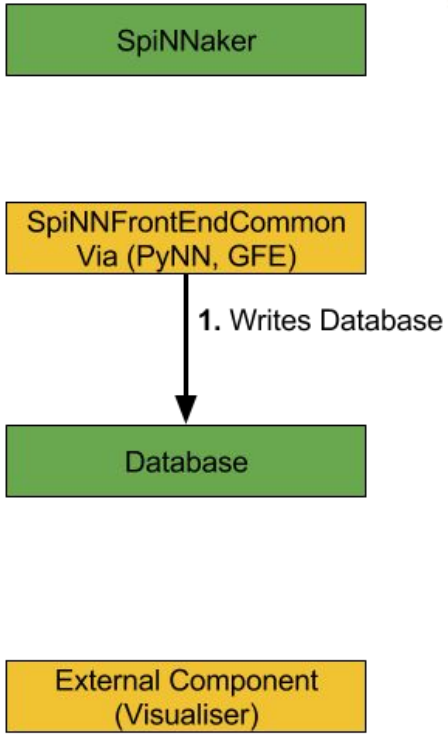
```
.....
# declare python code when received spikes for a timer tick
def receive_spikes(label, time, neuron_ids):
        for neuron_id in neuron_ids:
                print “Received spike at time {} from {}-{}”
                        .format(time, label, neuron_id)
# set up python live spike connection
live_spikes_connection =
    p.external_devices.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)
```



Technical Detail!!!

Notification protocol under the hood!

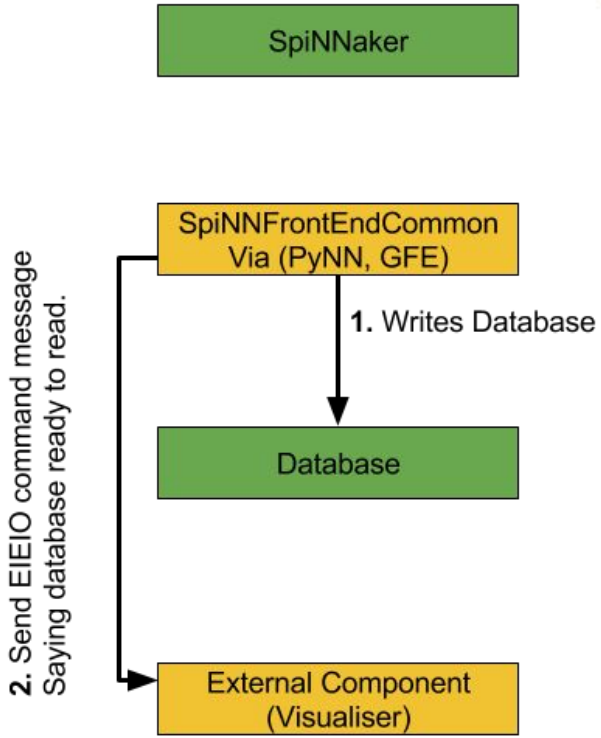
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.



Technical Detail!!!

Notification protocol under the hood!

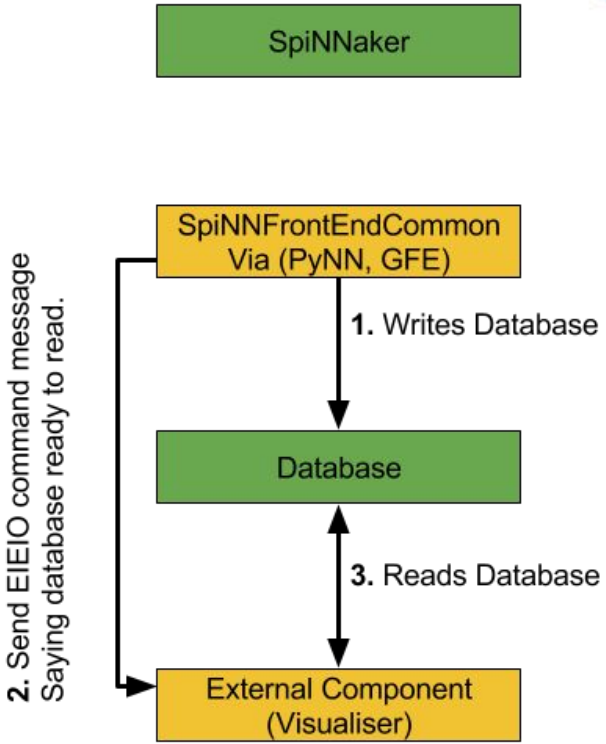
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.



Technical Detail!!!

Notification protocol under the hood!

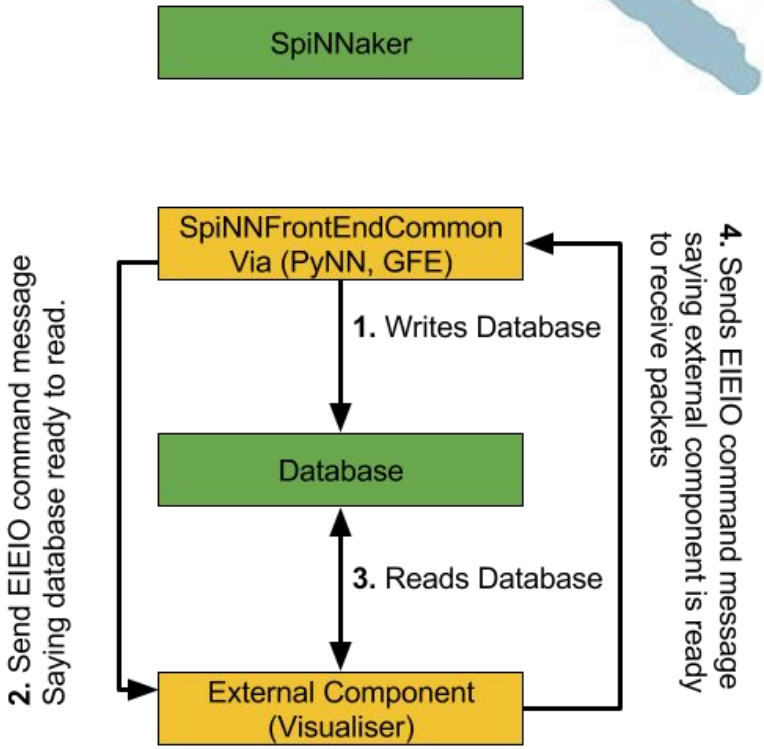
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.



Technical Detail!!!

Notification protocol under the hood!

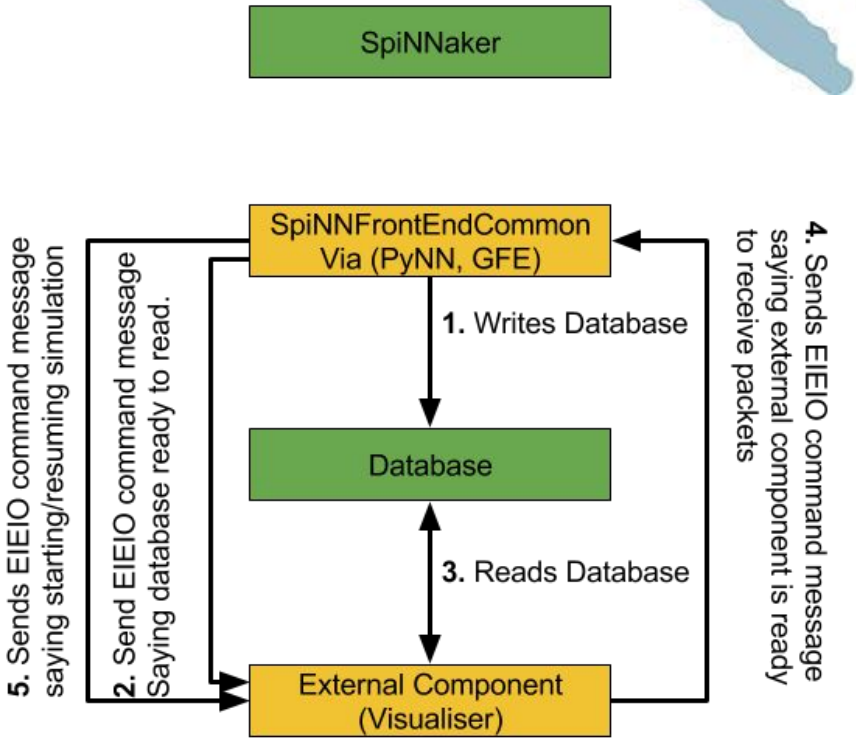
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.



Technical Detail!!!

Notification protocol under the hood!

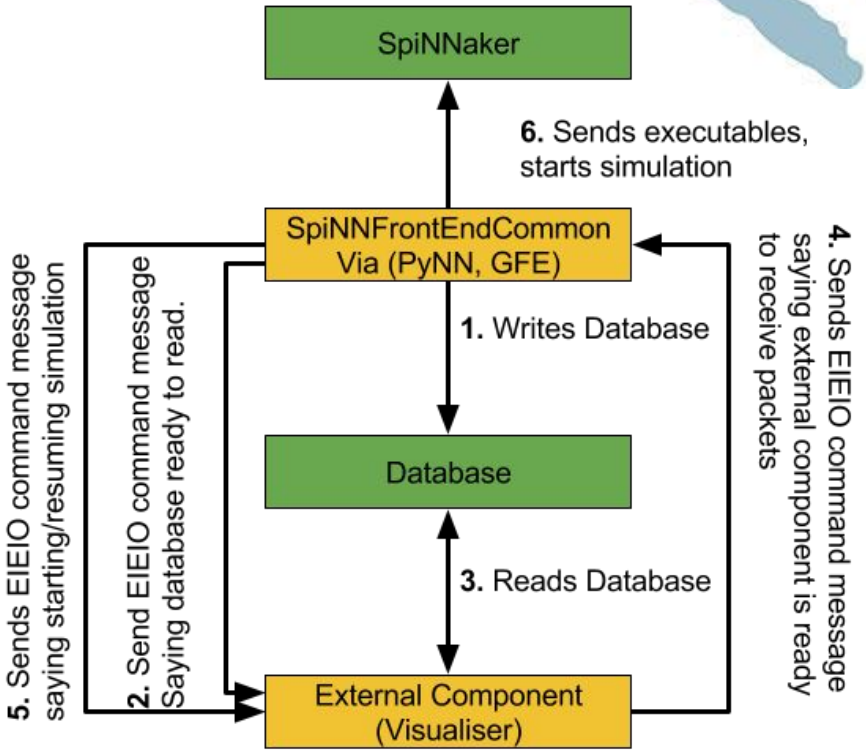
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.



Technical Detail!!!

Notification protocol under the hood!

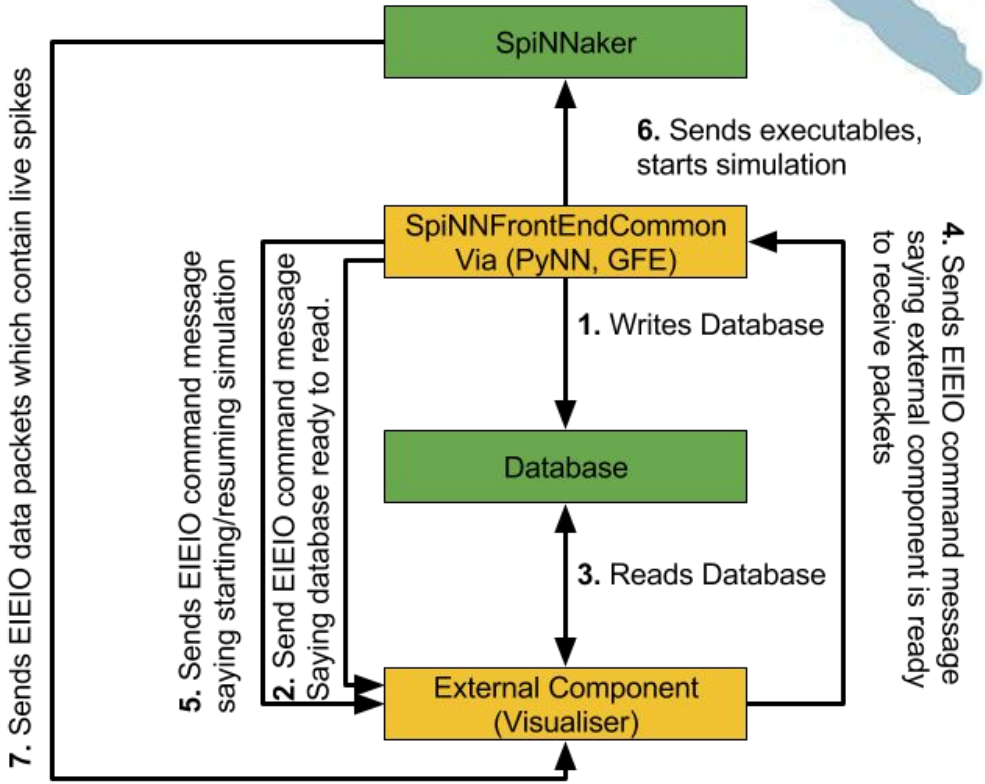
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.
- Auto-pause-and-resume functionality will result
  In steps 4-8 being repeated.



Technical Detail!!!

Notification protocol under the hood!

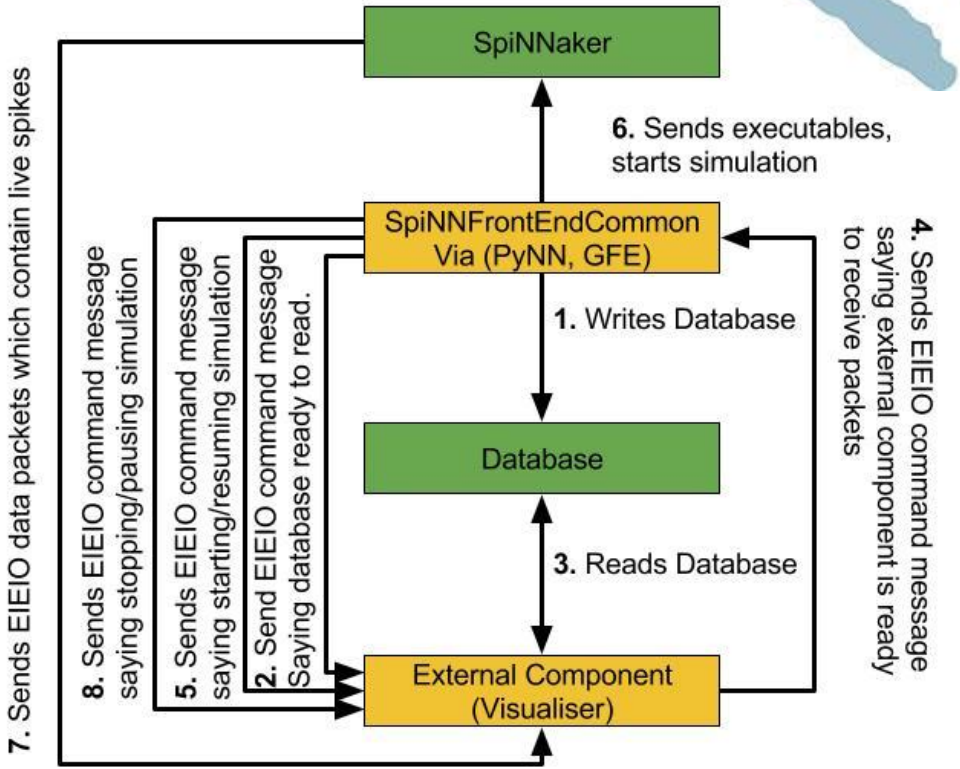
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.
- Auto-pause-and-resume functionality will result
  In steps 4-8 being repeated.



Technical Detail!!!

Notification protocol under the hood!

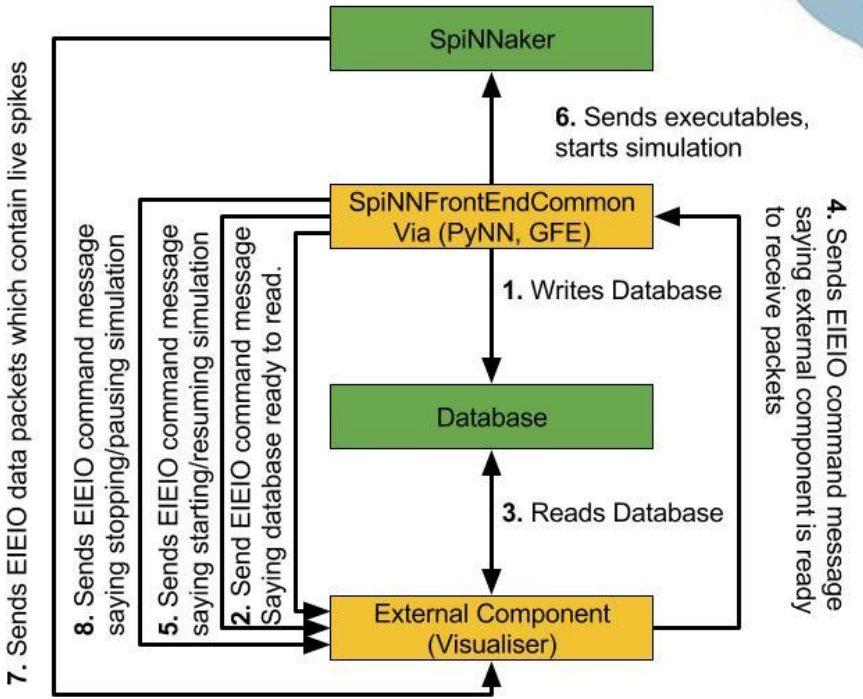
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.
- Auto-pause-and-resume functionality will result
  In steps 4-8 being repeated.



Technical Detail!!!

Notification protocol under the hood!

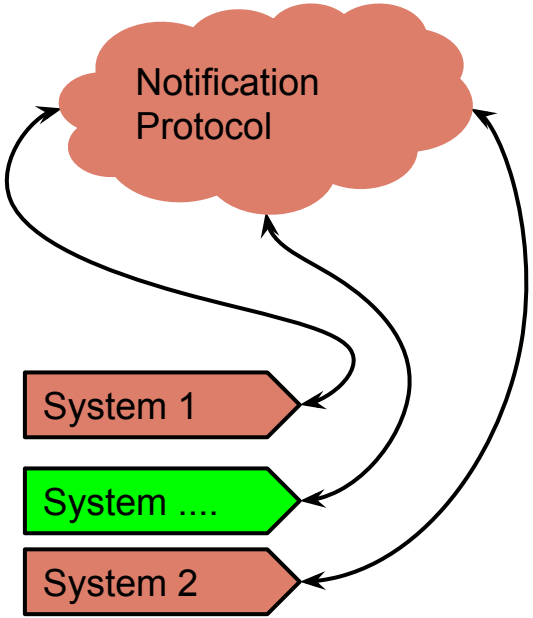
- Everything so far uses the notification protocol.
- It supplies data to translate spikes into population ids.
- Auto-pause-and-resume functionality will result
  In steps 4-8 being repeated.
- If you have more than 1 system running to inject and/or receive, then you need to register this with the notification protocol.



Injecting spikes into PyNN scripts

PyNN script changes: registering a system to the notification protocol

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



Thanks for listening

Any questions?!