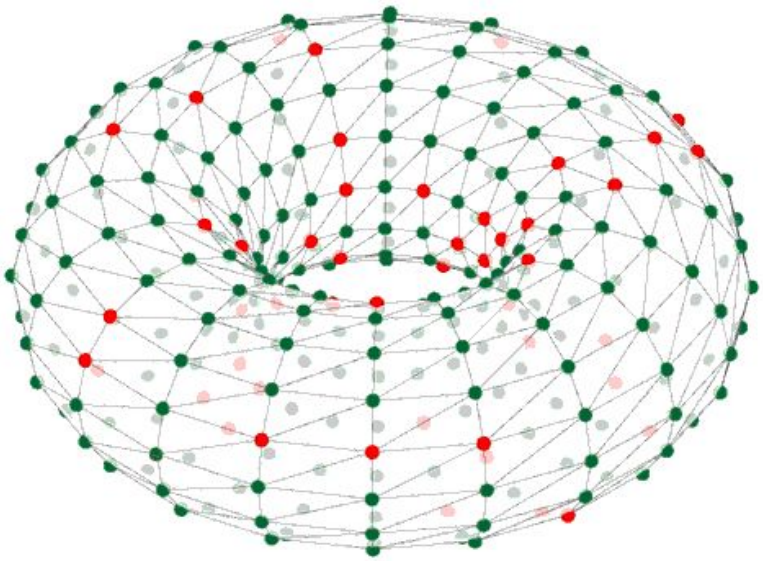
MANCHESTER 1824

Adding New Neuron Models



Andrew Gait, Andrew Rowley, Michael Hopkins

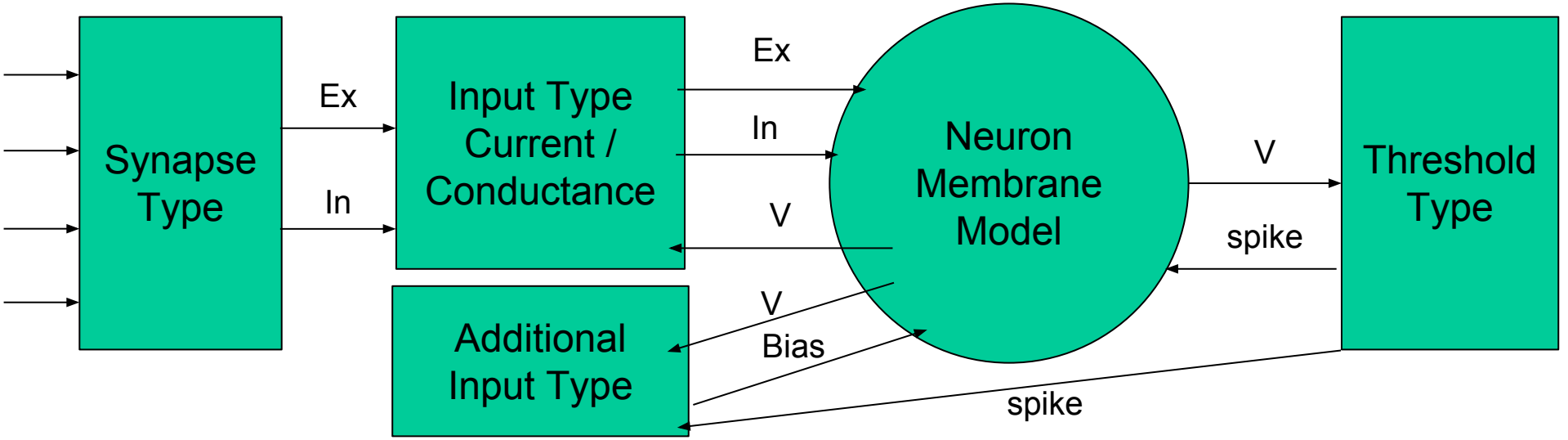
erc
European Research Council
Established by the European Commission

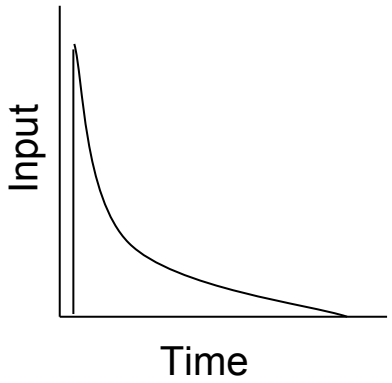
Human Brain Project

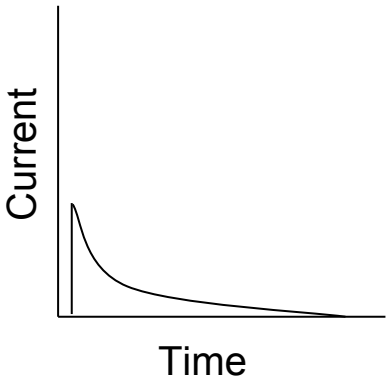
EPSRC

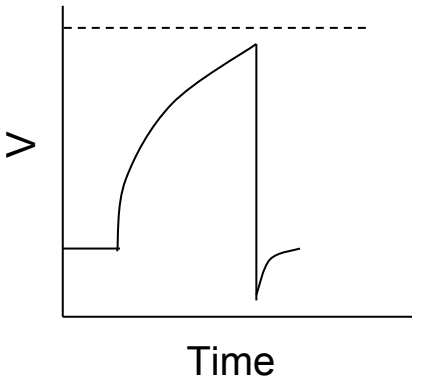
SpiNNaker

sPyNNaker Neuron Models





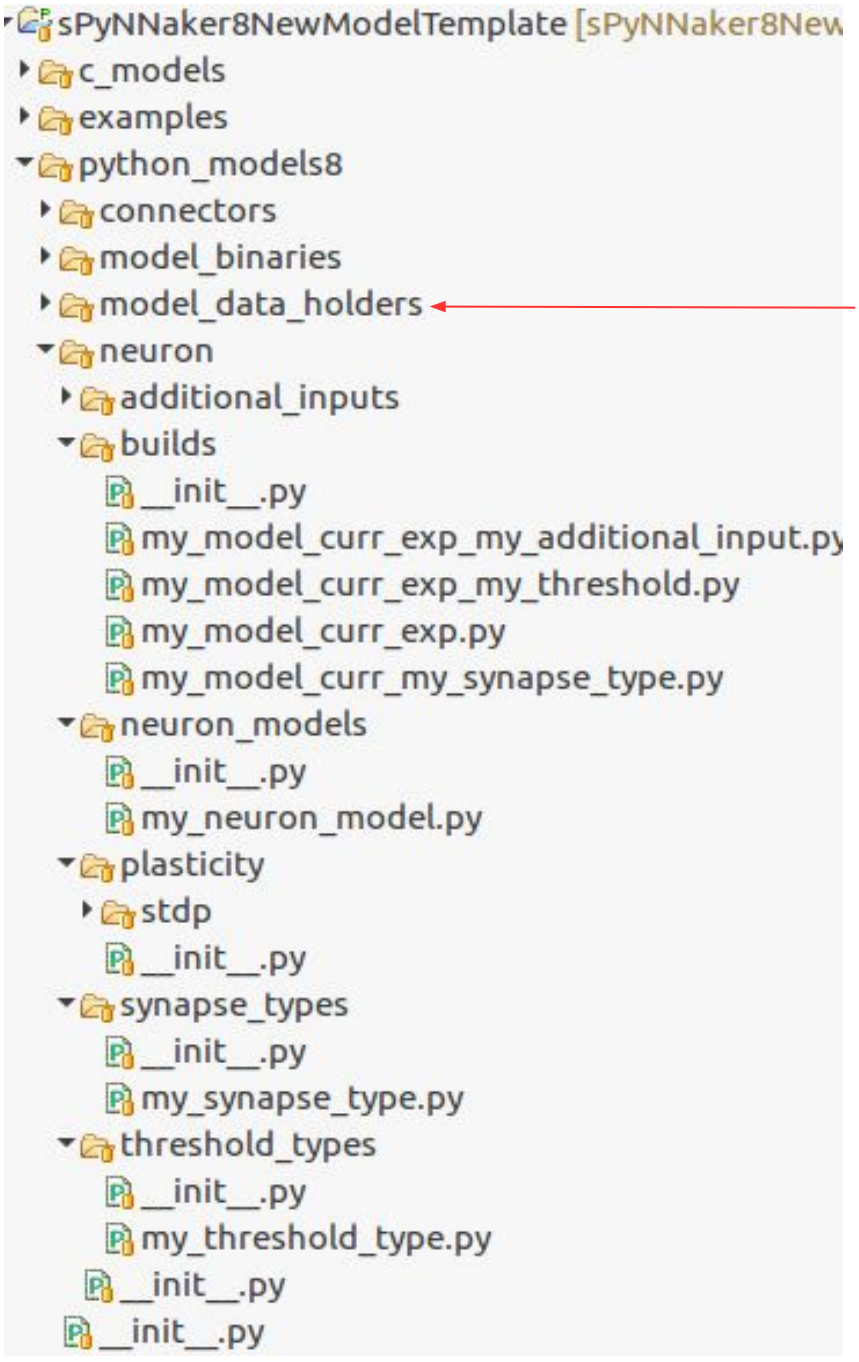




New Model Template

C code template

Python code template

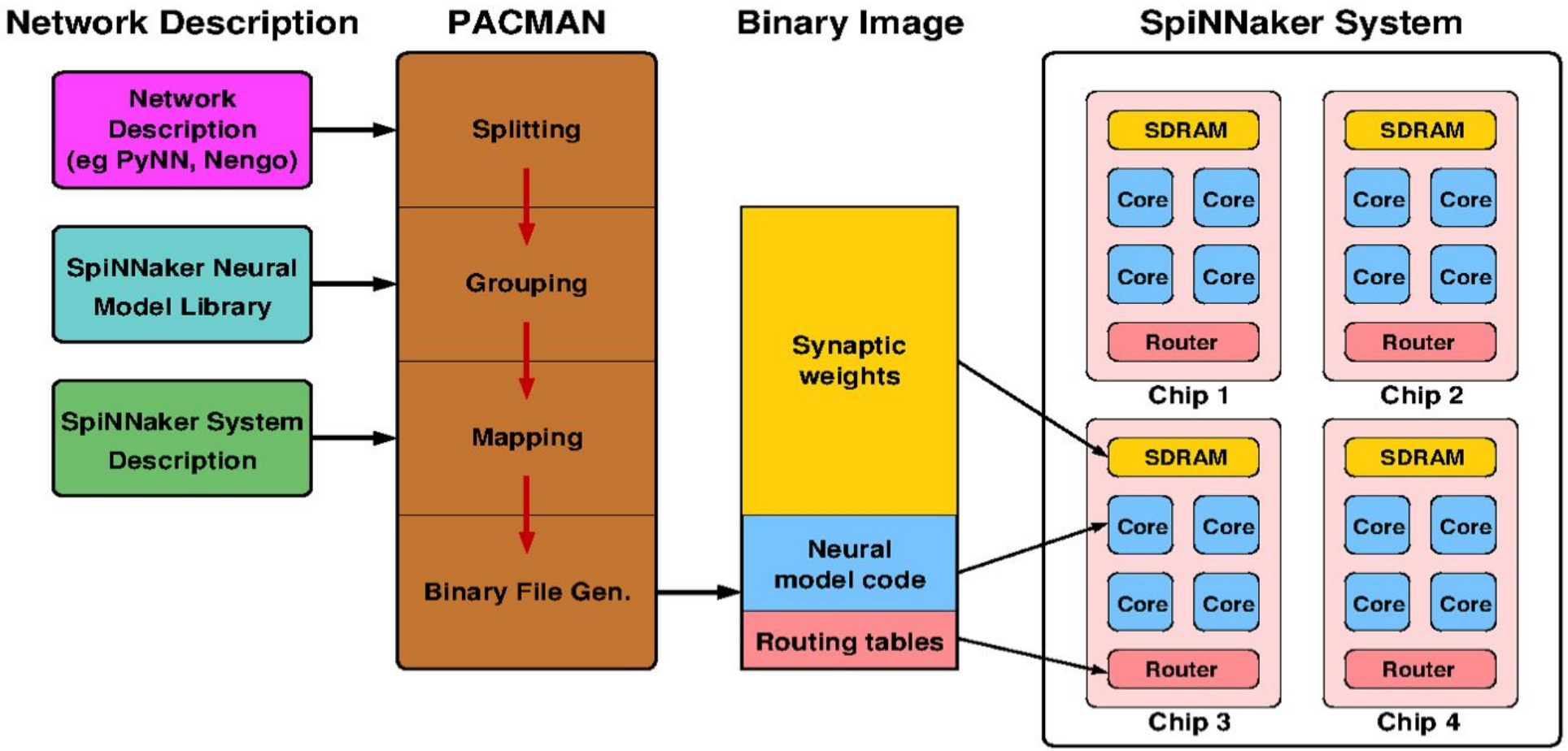


PyNN 0.8

Required code separation

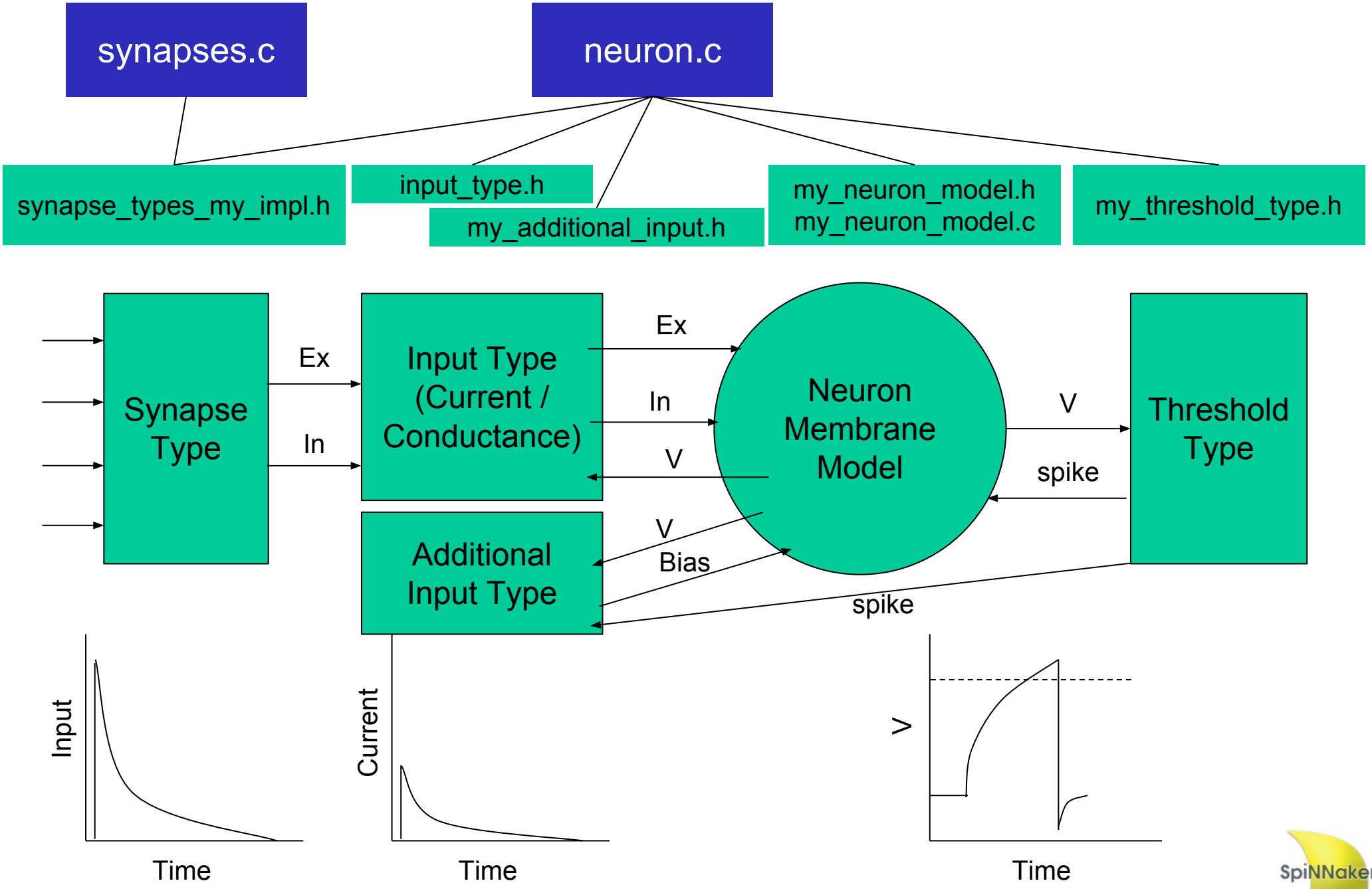
- Any new neuron model requires both C and Python code
- C code makes the actual executable (on SpiNNaker), Python code configures the setup and load phases (on the host)
- These are separate but must be perfectly coordinated
- In almost all cases, the C code will be solving an ODE which describes how the neuron state evolves over time and in response to input (as described in previous talk ***Maths & fixed-point libraries***)

Required code separation



- We will first describe the C requirements

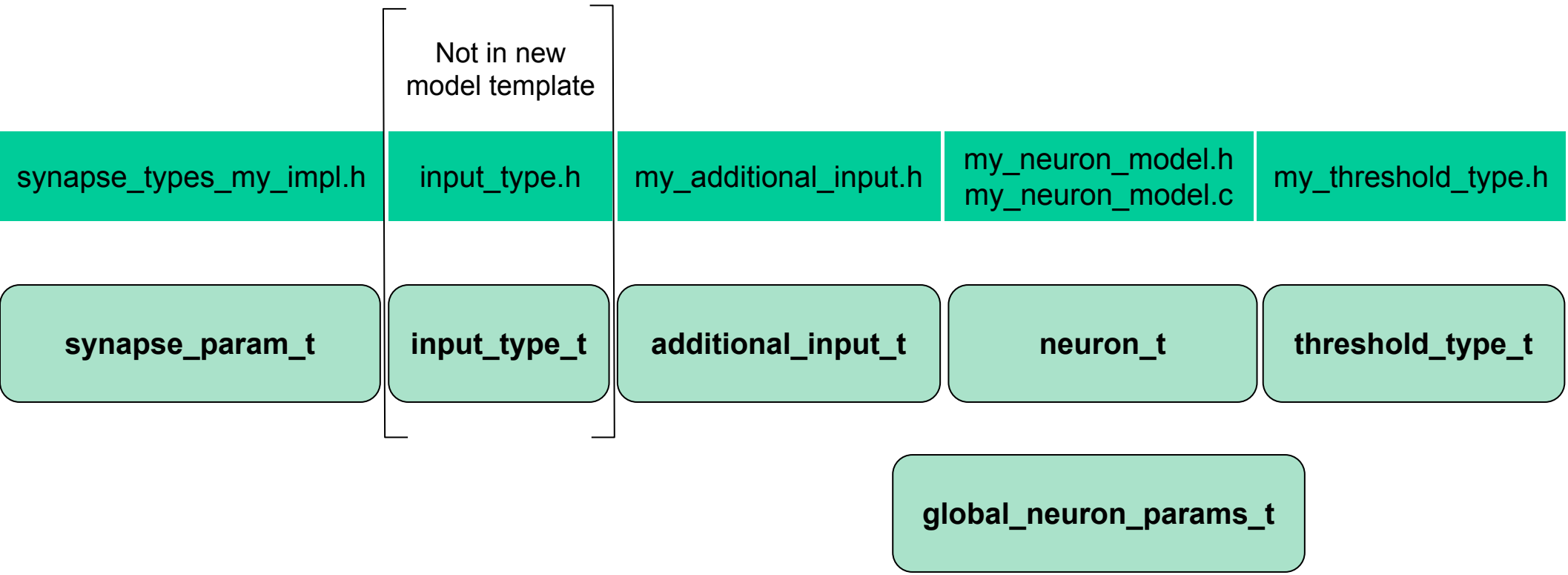
sPyNNaker Neuron Models - C



C Data Structures and Parameters

- The parameters and state of a neuron at any point in time need to be stored in memory
- For each neuron, the C header defines the ordering and size of each stored value
- The C types can be standard integer and floating-point, or ISO draft standard fixed-point, as required (see earlier talk ***Maths & fixed-point libraries***)
- There is also one global data structure which services all neurons on a core

sPyNNaker Neuron Models - C Structures



Specific neuron model - data structure

- An example of the C data structure using the Izhikevich neuron

```
#include "neuron-model.h"

//     Izhikevich neuron data struct defined in neuron_model_izh_impl.h

typedef struct neuron_t {

// nominally 'fixed' parameters - abstract units
   REAL   A;
   REAL   B;
   REAL   C;
   REAL   D;

// Variable-state parameters
   REAL   V;           // nominally in [mV]
   REAL   U;

// offset current [nA]
   REAL   I_offset;

// current timestep - simple correction for threshold
   REAL   this_h;

} neuron_t;
...
```

Neuron model: global data structure

```
...

/*
   Global data structure defined in neuron_model_izh_impl.h
*/

typedef struct global_neuron_params_t {

// Machine time step in milliseconds
   REAL   machine_timestep_ms;

} global_neuron_params_t;
```

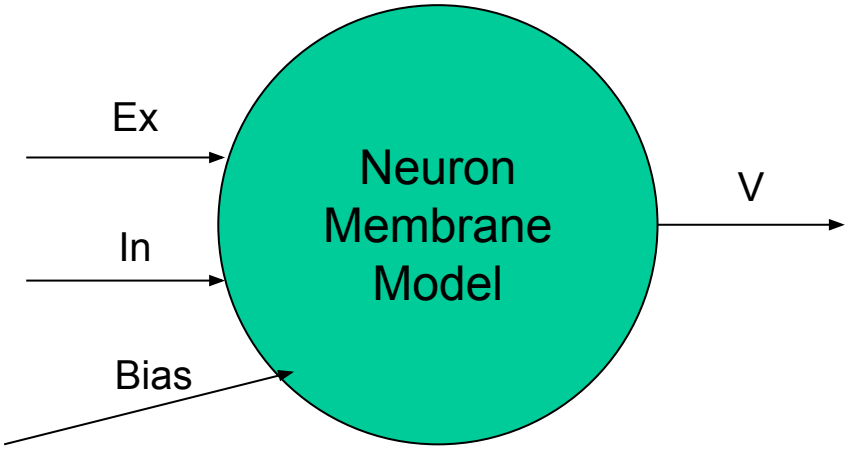
Implementing the state update (1)

- Neuron models are typically described as systems of initial value ODEs
- At each time step, the internal state of each neuron needs to be updated in response to inherent dynamics and synaptic input
- There are many ways to achieve this; there will usually be a 'best approach' (in terms of balance between accuracy & efficiency) for each neuron model

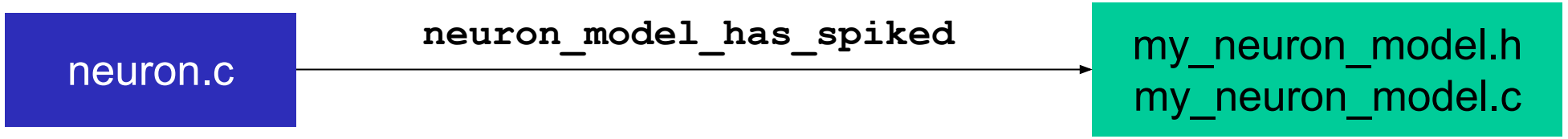
Implementing the state update (2)

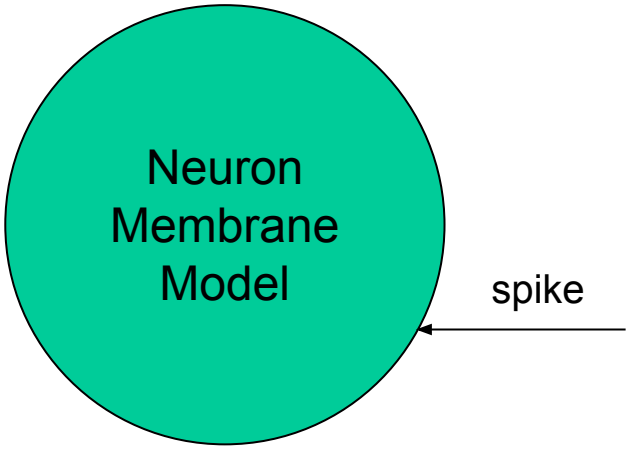
- A published paper gives a lot more detail: Hopkins & Furber (2015), “Accuracy and Efficiency in Fixed-Point Neural ODE Solvers” , *Neural Computation* **27**, 1–35
- The key function will always be *neuron_model_state_update*(); the other functions are mainly used to support this and allow debugging etc.
- Continuing the Izhikevich example by describing the key interfaces

sPyNNaker Neuron Models - C Interfaces

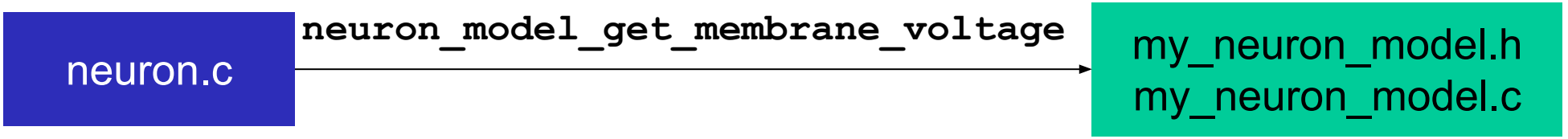


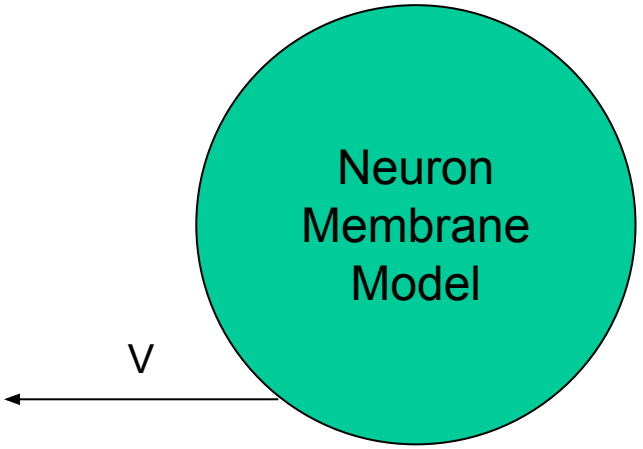
sPyNNaker Neuron Models - C Interfaces





sPyNNaker Neuron Models - C Interfaces





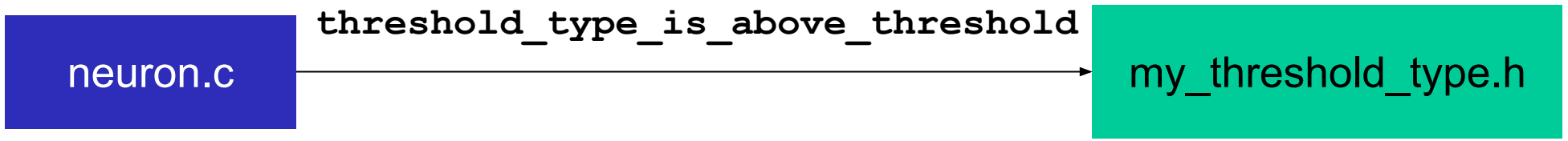
Neuron model API in neuron_model.h

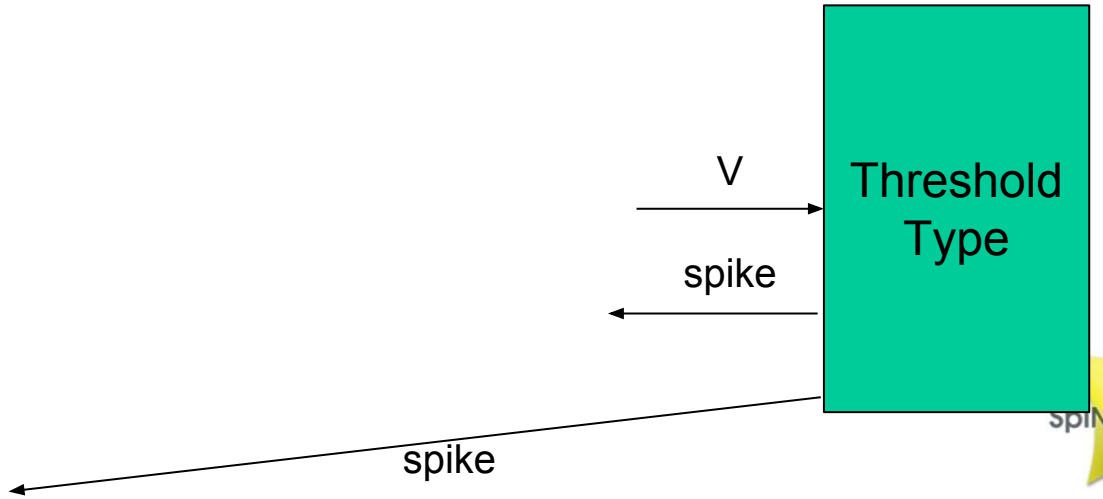
```
•
•// Forward declaration of neuron type (creates a definition for a pointer to a
•//  neuron parameter struct
•typedef struct neuron_t* neuron_pointer_t;
•
•// Forward declaration of global neuron parameters
•typedef struct global_neuron_params_t* global_neuron_params_pointer_t;
•
•// set the global neuron parameters
•void neuron_model_set_global_neuron_params(
•    global_neuron_params_pointer_t params);
•
•// primary function called in timer loop after synaptic update
•state_t neuron_model_state_update(
•                input_t exc_input, input_t inh_input, input_t external_bias,
•                neuron_pointer_t neuron );
•
•// Indicates that the neuron has spiked
•void neuron_model_has_spiked( neuron_pointer_t neuron );
•
•// Get the neuron membrane voltage for a given neuron parameter set
•state_t neuron_model_get_membrane_voltage(  restrict neuron_pointer_t neuron );
•
•// print out neuron parameters (i.e. values which don't change, for debug)
•void neuron_model_print_parameters(  restrict neuron_pointer_t neuron );
•
•// print out state variables (i.e. values that might change, for debug)
•void neuron_model_print_state_variables(  restrict neuron_pointer_t neuron );
•
```

Specific neuron model - key functions

```
// simplified version of Izhikevich neuron code defined in neuron_model_izh_impl.c

static const REAL SIMPLE_TQ_OFFSET = REAL_CONST(1.85);

// key function in timer loop that updates neuron state and returns membrane voltage
state_t neuron_model_state_update(
                input_t exc_input, input_t inh_input, input_t external_bias,
                neuron_pointer_t neuron) {

// collect inputs
    input_t input_this_timestep = exc_input - inh_input
                                    + external_bias + neuron->I_offset;

// most balanced ESR update found so far
    _rk2_kernel_midpoint(neuron->this_h, neuron, input_this_timestep);
    neuron->this_h = global_params->machine_timestep_ms;

// return the value of the membrane voltage
    return neuron->V;
}

// make the discrete changes to state after a spike has occurred
void neuron_model_has_spiked( neuron_pointer_t neuron ) {
    // reset membrane voltage
    neuron->V  = neuron->C;
    // offset 2nd state variable
    neuron->U += neuron->D;
    // simple threshold correction - next timestep (only) gets a bump
        neuron->this_h = global_params->machine_timestep_ms * SIMPLE_TQ_OFFSET;
}
```

sPyNNaker Neuron Models - C Interfaces





Threshold Models: Interface and Implementation

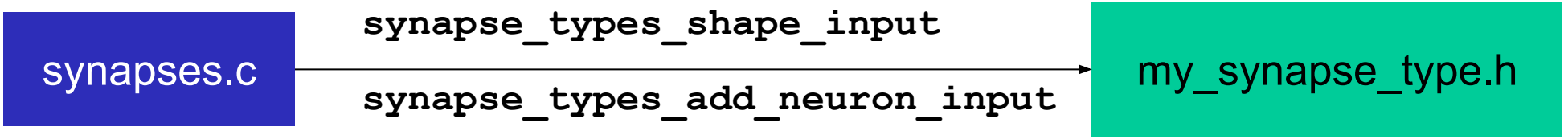
- Interface

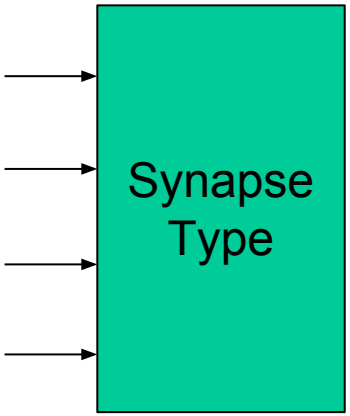
```
// Forward declaration of the threshold pointer type (used to access all operations)
typedef struct threshold_type_t* threshold_type_pointer_t;


// Determines if the value given is above the threshold value (main function)
static inline bool threshold_type_is_above_threshold(
        state_t value, threshold_type_pointer_t threshold_type );
```

- Static Threshold Implementation

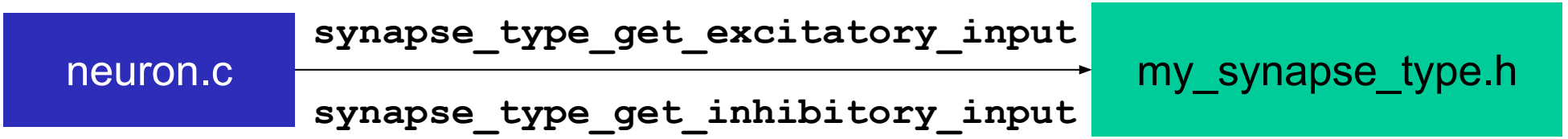
```
typedef struct threshold_type_t {

    // The value of the static threshold
    REAL threshold_value;

} threshold_type_t;


static inline bool threshold_type_is_above_threshold(state_t value,
                        threshold_type_pointer_t threshold_type) {

    return REAL_COMPARE(value, >=, threshold_type→threshold_value);
}
```

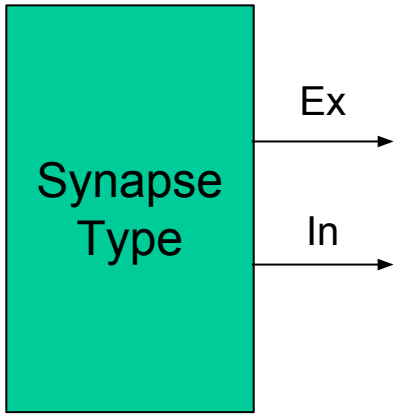
sPyNNaker Neuron Models - C Interfaces



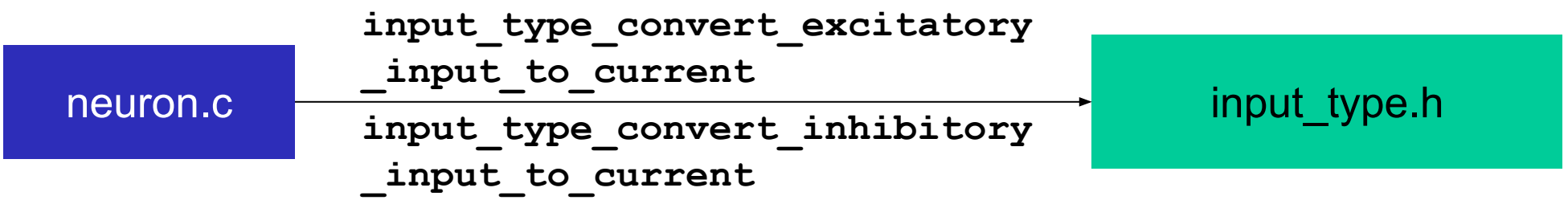


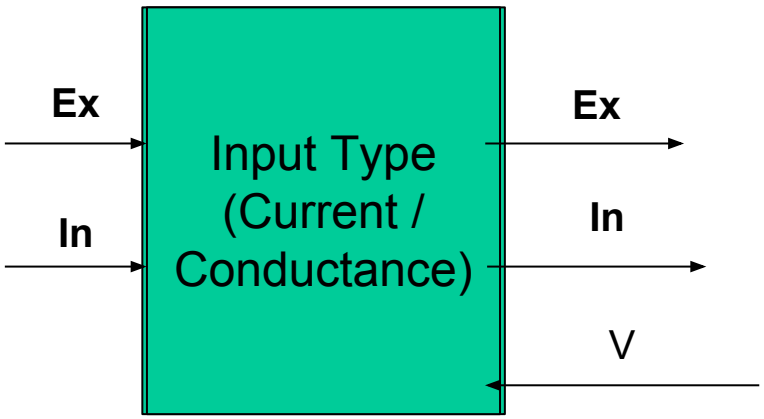
sPyNNaker Neuron Models - C Interfaces



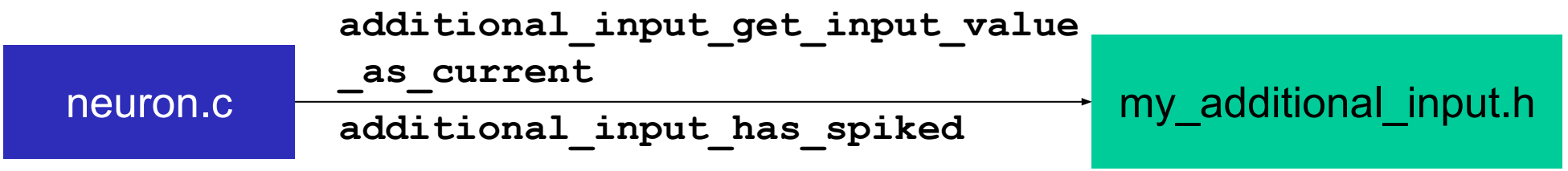


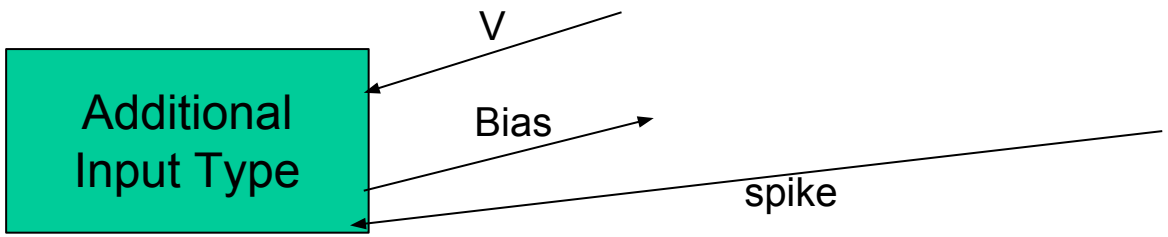
sPyNNaker Neuron Models - C Interfaces





sPyNNaker Neuron Models - C Interfaces





sPyNNaker Neuron Models - C Models

synapse_types_exponential.h

synapse_types_dual_excitatory_exponential.h

synapse_types_delta_impl.h

input_type_current.h

input_type_conductance.h

additional_input_none_impl.h

additional_input_ca2_adaptive_impl.h

neuron_model_izh_impl.c / .h

neuron_model_lif_impl.c / .h

threshold_type_static.h

threshold_type_external_device_control.h

threshold_type_maass_stochastic.h

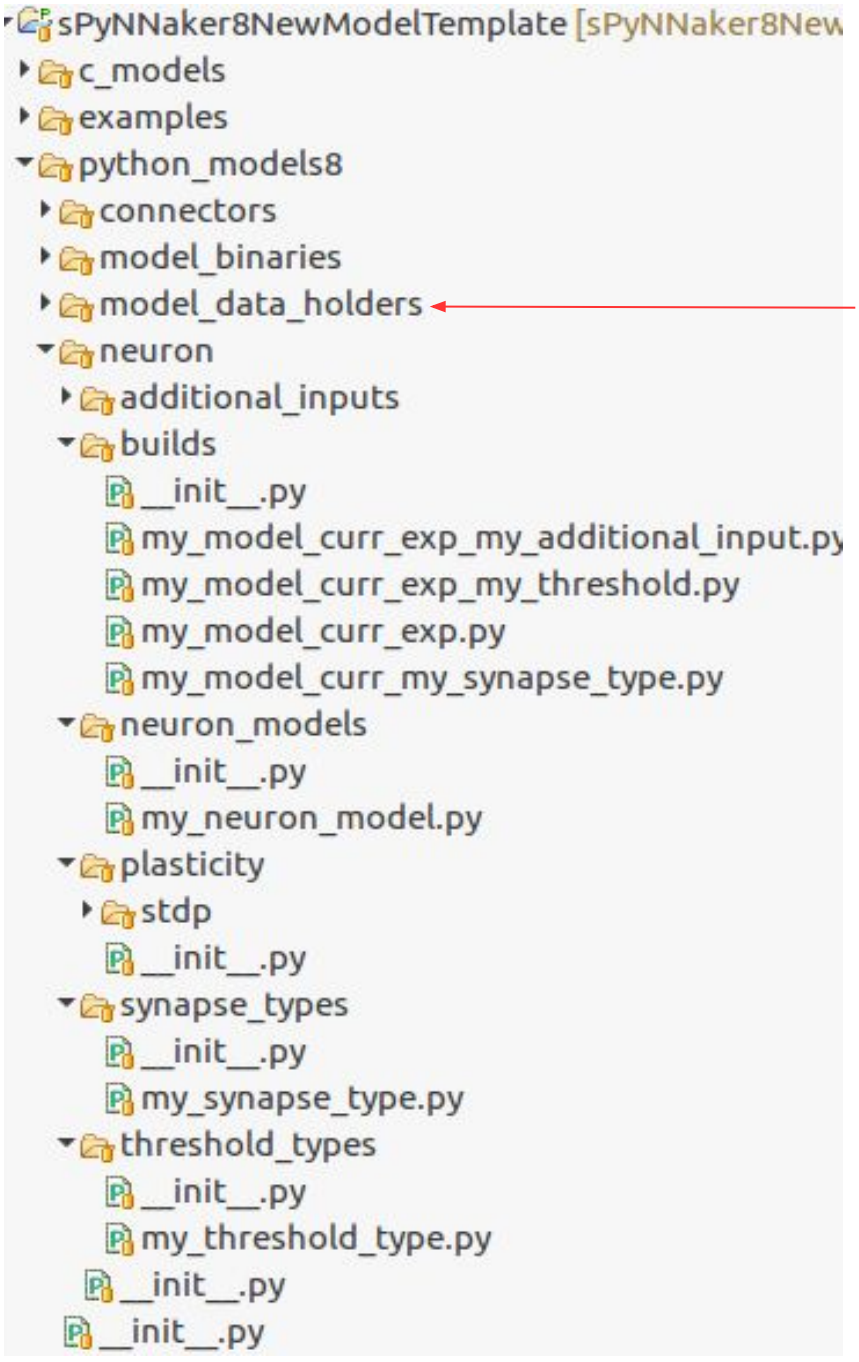
sPyNNaker Neuron Models - Makefile

```
APP = my_model_curr_exp

# This is the folder where things will be built (this will be created)
BUILD_DIR = build/

# This is the neuron model implementation
NEURON_MODEL = $(EXTRA_SRC_DIR)/neuron/models/my_neuron_model.c

# This is the header of the neuron model, containing the definition of neuron_t
NEURON_MODEL_H = $(EXTRA_SRC_DIR)/neuron/models/my_neuron_model.h

# This is the (optional) header containing the additional input type
ADDITIONAL_INPUT_H = $(EXTRA_SRC_DIR)/neuron/additional_inputs/my_additional_input.h

# This is the header containing the input type (current in this case)
INPUT_TYPE_H = $(SOURCE_DIR)/neuron/input_types/input_type_current.h

# This is the header containing the threshold type (static in this case)
THRESHOLD_TYPE_TYPE_H = $(SOURCE_DIR)/neuron/threshold_types/threshold_type_static.h

# This is the header containing the synapse shaping type (exponential in this case)
SYNAPSE_TYPE_H = $(SOURCE_DIR)/neuron/synapse_types/synapse_types_exponential_impl.h

# This is the synapse dynamics type (in this case static i.e. no synapse dynamics)
SYNAPSE_DYNAMICS = $(SOURCE_DIR)/neuron/plasticity/synapse_dynamics_static_impl.c

# This includes the common Makefile that hides away the details of the build
include ../Makefile.common
```

New Model Template

C code template

Python code template



PyNN 0.8

Python interface - Why?

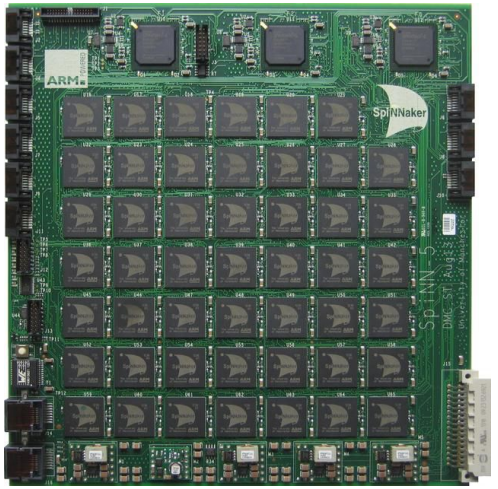
PyNN Script

sPyNNaker

PACMAN

DataSpecification

SpiNNMan



Python Interface - Parameters

- Parameters can be:
  - Individual values
  - Array of values (one per neuron)
  - RandomDistribution

- Normalise Parameters (in the __init__ function)
  - utility_calls.convert_param_to_numpy(
    param, n_neurons)

sPyNNaker Neuron Models - Python

AbstractPopulationVertex → MyNeuronModel

```
get_n_cpu_cycles_per_neuron
get_n_neural_parameters
get_n_global_parameters
get_neural_parameters
get_global_parameters
get_neural_parameter_types
```

C

```
struct my_input_type {
  accum  param_1;
  uint32 param_2;
  accum  state_var;
}
```

3

Python

```
[
 NeuronParameter(self._param_1, DataType.S1615)
 NeuronParameter(self._param_2, DataType.UINT32)
 NeuronParameter(self._var_init, DataType.S1615)
]
```

Specific neuron model - data structure

- An example of the C data structure using the Izhikevich neuron

```
#include "neuron-model.h"

//     Izhikevich neuron data struct defined in neuron_model_izh_impl.h

typedef struct neuron_t {

// nominally 'fixed' parameters - abstract units
   REAL   A;
   REAL   B;
   REAL   C;
   REAL   D;

// Variable-state parameters
   REAL   V;           // nominally in [mV]
   REAL   U;

// offset current [nA]
   REAL   I_offset;

// current timestep - simple correction for threshold
   REAL   this_h;

} neuron_t;
...
```

Python Interface: DataType

```
from spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model \
    import AbstractNeuronModel
from spynnaker.pyNN.models.abstract_models  import AbstractContainsUnits

from data_specification.enums  import DataType

from enum import Enum


class _IZH_TYPES(Enum)
    A = (1, DataType.S1615)
    B = (2, DataType.S1615)
    C = (3, DataType.S1615)
    D = (4, DataType.S1615)
    V_INIT = (5, DataType.S1615)
    U_INIT = (6, DataType.S1615)
    I_OFFSET = (7, DataType.S1615)
    THIS_H = (8, DataType.S1615)

    def __new__(cls, value, data_type):
        obj = object.__new__(cls)
        obj._value_ = value
        obj._data_type = data_type
        return obj

    @property
    def data_type(self):
        return self._data_type
```

Python Interface: DataType

```
from spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model \
    import AbstractNeuronModel
from spynnaker.pyNN.models.abstract_models  import AbstractContainsUnits

from data_specification.enums  import DataType

from enum import Enum
    ...


class _IZH_GLOBAL_TYPES(Enum):
    TIMESTEP = (1, DataType.S1615)

    def __new__(cls, value, data_type):
        obj = object.__new__(cls)
        obj._value_ = value
        obj._data_type = data_type
        return obj

    @property
    def data_type(self):
        return self._data_type

class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):

    def __init__(self, n_neurons, a, b, c, d, v_init, u_init, i_offset):
        AbstractNeuronModel.__init__( self)
        AbstractContainsUnits.__init__( self)
        self._n_neurons = n_neurons
```

Python Interface - initializer

```
from spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model \
    import AbstractNeuronModel
from spynnaker.pyNN.models.abstract_models  import AbstractContainsUnits
    ...


class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):
    def __init__(self, n_neurons, a, b, c, d, v_init, u_init, i_offset):
        AbstractNeuronModel.__init__( self)
        AbstractContainsUnits.__init__( self)

        self._units = {
            'a' : "ms",
            'b' : "ms",
            'c' : "mV",
            'd' : "mV/ms",
            'v_init' : "mV",
            'u_init' : "mV/ms",
            'i_offset' : "nA"}

        self._n_neurons = n_neurons
        self._a = utility_calls.convert_param_to_numpy(a, n_neurons)
        self._b = utility_calls.convert_param_to_numpy(b, n_neurons)
        self._c = utility_calls.convert_param_to_numpy(c, n_neurons)
        self._d = utility_calls.convert_param_to_numpy(d, n_neurons)
        self._v_init = utility_calls.convert_param_to_numpy(v_init, n_neurons)
        self._u_init = utility_calls.convert_param_to_numpy(u_init, n_neurons)
        self._i_offset = utility_calls.convert_param_to_numpy(
            i_offset, n_neurons)
```

Python Interface - properties

```
class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):
    ...


    @property
    def a(self):
        return self._a

    @a.setter
    def a(self, a):
        self._a = utility_calls.convert_param_to_numpy(a,  self._n_neurons)


    @property
    def b(self):
        return self._b

    @b.setter
    def b(self, b):
        self._b = utility_calls.convert_param_to_numpy(b,  self._n_neurons)


    ...
```

Python Interface - state initializers

```
class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):
    ...

    def initialize_v(self, v_init):
        self._v_init = utility_calls.convert_param_to_numpy(v_init,  self._n_neurons)

    def initialize_u(self, u_init):
        self._u_init = utility_calls.convert_param_to_numpy(u_init,  self._n_neurons)
```

Python Interface - parameters

```
class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):
    ...
    @overrides(AbstractNeuronModel.get_n_neural_paramters)
    def get_n_neural_parameters(self):
        return 8

    @inject_items({"machine_time_step": "MachineTimeStep"})
    @overrides(AbstractNeuronModel.get_neural_paramters,
               additional_arguments={ 'machine_time_step' })
    def get_neural_parameters(self, machine_time_step):
        return [
            # REAL A
            NeuronParameter( self._a, _IZH_TYPES.A.data_type),
            # REAL B
            NeuronParameter( self._b, _IZH_TYPES.B.data_type),
            # REAL C
            NeuronParameter( self._c, _IZH_TYPES.C.data_type),
            # REAL D
            NeuronParameter( self._d, _IZH_TYPES.D.data_type),
            # REAL V
            NeuronParameter( self._v_init, _IZH_TYPES.V_INIT.data_type),
            # REAL U
            NeuronParameter( self._u_init, _IZH_TYPES.U_INIT.data_type),
            # REAL I_offset
            NeuronParameter( self._i_offset, _IZH_TYPES.I_OFFSET.data_type),
            # REAL this_h
            NeuronParameter(
               machine_time_step / 1000.0, _IZH_TYPES.THIS_H.data_type)
        ]
```

Python Interface - global parameters

```
class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):
    ...


    def get_n_global_parameters(self):
        return 1


    @inject_items({"machine_time_step": "MachineTimeStep"})
    @overrides(AbstractNeuronModel.get_neural_paramters,
               additional_arguments={ 'machine_time_step' })
    def get_global_parameters(self, machine_time_step):
        return [
            NeuronParameter(
                machine_time_step /  1000.0,
                _IZH_GLOBAL_TYPES.TIMESTEP.data_type)
        ]

    @overrides(AbstractNeuronModel.get_global_parameter_types)
    def get_global_parameter_types(self):
        return [item.data_type for item in _IZH_GLOBAL_TYPES]
```

Python Interface - injection

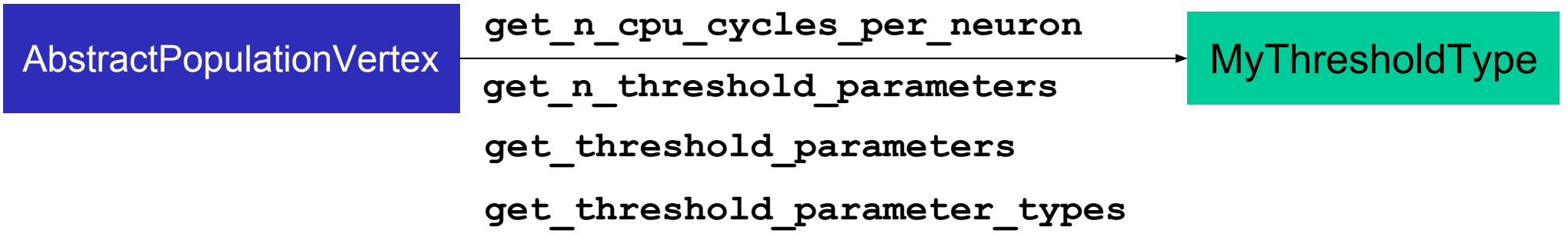
```
@inject_items({"machine_time_step": "MachineTimeStep"})
def get_global_parameters(self, machine_time_step):
```

- Some items can be “injected” from the interface
  - Specify a dictionary of parameter name to “type” to inject
  - Parameter is in addition to the interface
- Common types include:
  - MachineTimeStep
  - TimeScaleFactor
  - TotalRunTime

Python Interface - CPU usage

```
class NeuronModelIzh(AbstractNeuronModel, AbstractContainsUnits):
    ...

    def get_n_cpu_cycles_per_neuron(self):

        # A bit of a guess
        return 150
```

sPyNNaker Neuron Models - Python



Threshold Models: Interface and Implementation

- Interface

```
// Forward declaration of the threshold pointer type (used to access all operations)
typedef struct threshold_type_t* threshold_type_pointer_t;


// Determines if the value given is above the threshold value (main function)
static inline bool threshold_type_is_above_threshold(
        state_t value, threshold_type_pointer_t threshold_type );
```

- Static Threshold Implementation

```
typedef struct threshold_type_t {

    // The value of the static threshold
    REAL threshold_value;

} threshold_type_t;


static inline bool threshold_type_is_above_threshold(state_t value,
                        threshold_type_pointer_t threshold_type) {

    return REAL_COMPARE(value, >=, threshold_type→threshold_value);
}
```

Python Interface - Threshold type

```
class ThresholdTypeStatic(AbstractThresholdType, AbstractContainsUnits):
    """ A threshold that is a static value
    """


    def __init__(self, n_neurons, v_thresh):
        AbstractThresholdType.__init__( self)
        AbstractContainsUnits.__init__( self)

        self._units = {'v_thresh': "mV"}

        self._n_neurons = n_neurons
        self._v_thresh = utility_calls.convert_param_to_numpy(
            v_thresh, n_neurons)
```

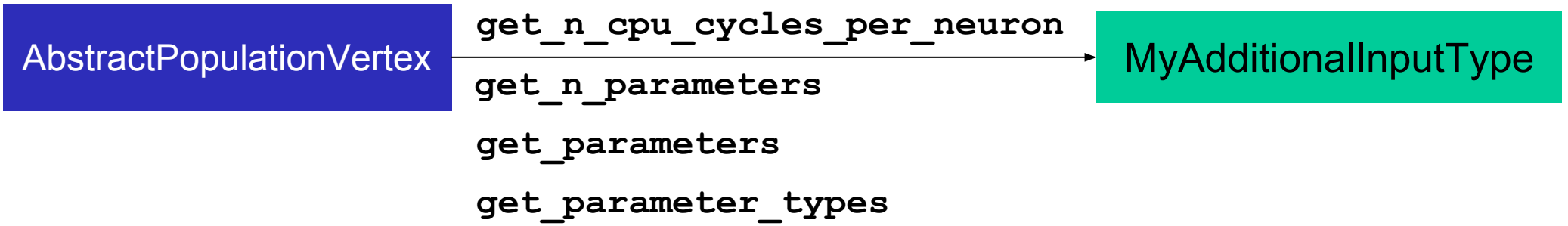
Python Interface - Threshold type

```
class ThresholdTypeStatic(AbstractThresholdType, AbstractContainsUnits):
    """ A threshold that is a static value
    """
    ...


    @property
    def v_thresh(self):
        return self._v_thresh


    @v_thresh.setter
    def v_thresh(self, v_thresh):
        self._v_thresh = utility_calls.convert_param_to_numpy(
            v_thresh, self._n_neurons)
```

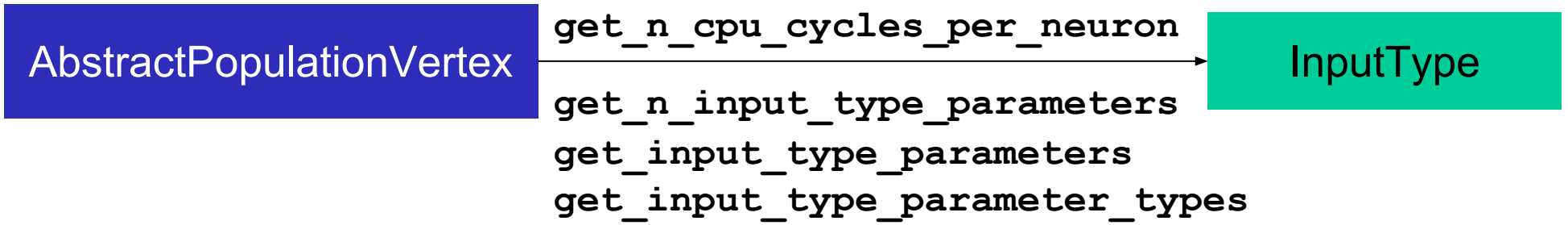
Python Interface - Threshold type

```
class ThresholdTypeStatic(AbstractThresholdType, AbstractContainsUnits):
    """ A threshold that is a static value
    """
    ...


    def get_n_threshold_parameters(self):
        return 1


    def get_threshold_parameters(self):
        return [
            NeuronParameter( self._v_thresh, _STATIC_TYPES.V_THRESH.data_type)
        ]


    def get_n_cpu_cycles_per_neuron(self):


        # Just a comparison, but 2 just in case!
        return 2
```

sPyNNaker Neuron Models - Python



sPyNNaker Neuron Models - Python

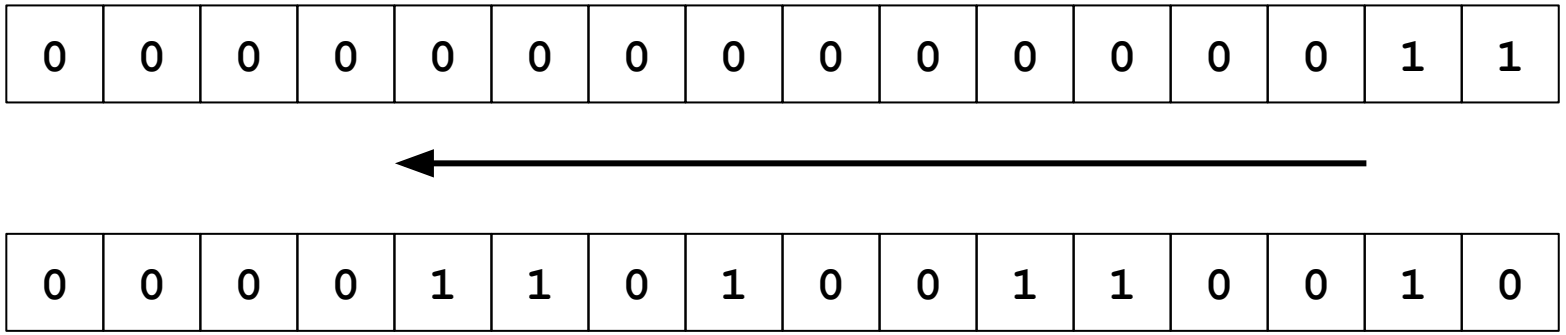


sPyNNaker Neuron Models - Python

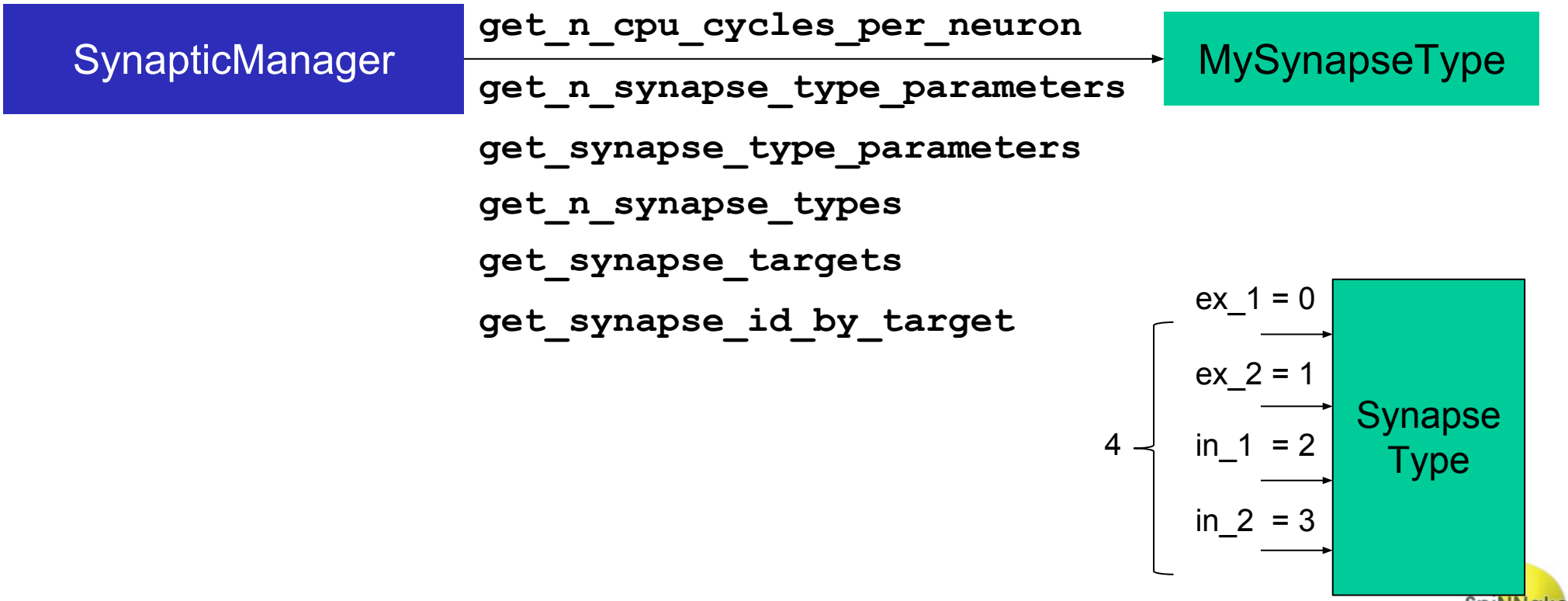
AbstractPopulationVertex

```
get_n_cpu_cycles_per_neuron
get_n_input_type_parameters
get_input_type_parameters
get_input_type_parameter_types
get_global_weight_scale
```

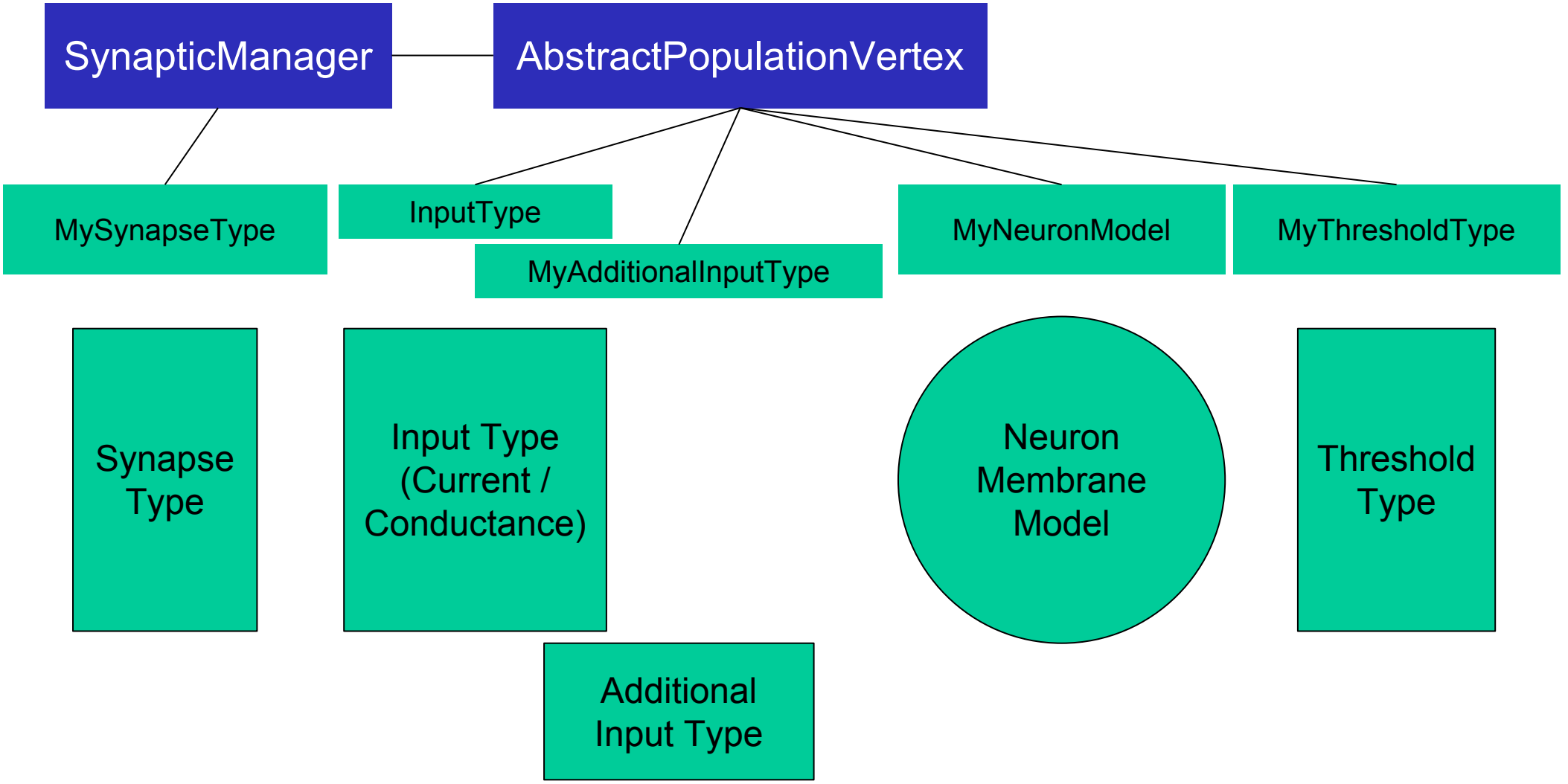
InputType



sPyNNaker Neuron Models - Python



sPyNNaker Neuron Models - Python build



Python Build - Class Definition

```
from spynnaker.pyNN.models.neuron.neuron_models  import NeuronModelIzh
from spynnaker.pyNN.models.neuron.synapse_types  import SynapseTypeExponential
from spynnaker.pyNN.models.neuron.input_types  import InputTypeCurrent
from spynnaker.pyNN.models.neuron.threshold_types  import ThresholdTypeStatic
from spynnaker.pyNN.models.neuron  import AbstractPopulationVertex


# global objects
DEFAULT_MAX_ATOMS_PER_CORE =  255
_IZK_THRESHOLD =  30.0

class IzkCurrExpBase(AbstractPopulationVertex):
```

Python Build

```
class IzkCurrExpBase(AbstractPopulationVertex):

    _model_based_max_atoms_per_core = DEFAULT_MAX_ATOMS_PER_CORE

    default_parameters = {
        'a': 0.02, 'c': -65.0, 'b': 0.2, 'd': 2.0, 'i_offset': 0,
        'u_init': -14.0, 'v_init': -70.0, 'tau_syn_E': 5.0, 'tau_syn_I': 5.0,
        'isyn_exc': 0, 'isyn_inh': 0}
```

Python Build - initializer (1)

```
class IzkCurrExpBase(AbstractPopulationVertex):

...
    def __init__(
            self, n_neurons,
            spikes_per_second=AbstractPopulationVertex.
            none_pynn_default_parameters[ 'spikes_per_second' ],
            ring_buffer_sigma=AbstractPopulationVertex.
            none_pynn_default_parameters[ 'ring_buffer_sigma' ],
            incoming_spike_buffer_size=AbstractPopulationVertex.
            none_pynn_default_parameters[ 'incoming_spike_buffer_size' ],
            constraints=AbstractPopulationVertex.none_pynn_default_parameters[
                'constraints' ],
            label=AbstractPopulationVertex.none_pynn_default_parameters[
                'label' ],
            a=default_parameters[ 'a'], b=default_parameters[ 'b'],
            c=default_parameters[ 'c'], d=default_parameters[ 'd'],
            i_offset=default_parameters[ 'i_offset'],
            u_init=default_parameters[ 'u_init'],
            v_init=default_parameters[ 'v_init'],
            tau_syn_E=default_parameters[ 'tau_syn_E'],
            tau_syn_I=default_parameters[ 'tau_syn_I'],
            isyn_exc=default_parameters[ 'isyn_exc'],
            isyn_inh=default_parameters[ 'isyn_inh']):


        neuron_model = NeuronModelIzh(
            n_neurons, a, b, c, d, v_init, u_init, i_offset)
        ...
```

Python Build - initializer (2)

```
class IzkCurrExpBase(AbstractPopulationVertex):

...
    def __init__(
        ...
            isyn_exc=default_parameters[ 'isyn_exc'],
            isyn_inh=default_parameters[ 'isyn_inh']):

        neuron_model = NeuronModelIzh(
            n_neurons, a, b, c, d, v_init, u_init, i_offset)
        synapse_type = SynapseTypeExponential(
            n_neurons, tau_syn_E, tau_syn_I, isyn_exc, isyn_inh)
        input_type = InputTypeCurrent()
        threshold_type = ThresholdTypeStatic(n_neurons, _IZK_THRESHOLD)

        AbstractPopulationVertex.__init__(
            self, n_neurons=n_neurons, binary= "IZK_curr_exp.aplx", label=label,
            max_atoms_per_core=IzkCurrExpBase._model_based_max_atoms_per_core,
            spikes_per_second=spikes_per_second,
            ring_buffer_sigma=ring_buffer_sigma,
            incoming_spike_buffer_size=incoming_spike_buffer_size,
            model_name= "IZK_curr_exp", neuron_model=neuron_model,
            input_type=input_type, synapse_type=synapse_type,
            threshold_type=threshold_type, constraints=constraints)
```

Python Build - max atoms

```
class IzkCurrExpBase(AbstractPopulationVertex):
    ...

    @staticmethod
    def set_model_max_atoms_per_core(new_value):
        IzkCurrExpBase._model_based_max_atoms_per_core = new_value


    @staticmethod
    def get_max_atoms_per_core():
        return IzkCurrExpBase._model_based_max_atoms_per_core
```

sPyNNaker Neuron Models - Python Build

```
class MyModelCurrExpBase(AbstractPopulationVertex):

    default_parameters = {
        'my_param': 1.0,
        'tau_syn_E': 5.0, 'tau_syn_I': 5.0, 'v_thresh': -50.0
        'isyn_exc': 0.0, 'isyn_inh': 0.0
    }

    def __init__(
            self, n_neurons, my_param=default_parameters['my_param'],
            tau_syn_E=default_parameters['tau_syn_E'],
            Tau_syn_i=default_parameters['tau_syn_I'],
            v_thresh=default_parameters['v_thresh'],
            isyn_exc=default_parameters['isyn_exc'],
            isyn_inh=default_parameters['isyn_inh']):

        AbstractPopulationVertex.__init__(
            self, n_neurons=n_neurons,
            ...
            neuron_model=MyNeuronModel(n_neurons, my_param),
            input_type=InputTypeCurrent(),
            synapse_type=SynapseTypeExponential(
                n_neurons, tau_syn_I, tau_syn_E, isyn_exc, isyn_inh),
            threshold_type=ThresholdTypeStatic(n_neurons, v_thresh),
            additional_input=None
            model_name="MyModelCurrExpBase",
            binary="my_model_curr_exp.aplx",
        )
```

Python Build - DataHolder (1)

- This is only required since version 0.8 of PyNN

```
from spynnaker.pyNN.models.neuron  import AbstractPopulationVertex
from spynnaker8.utilities  import DataHolder
from spynnaker.pyNN.models.neuron.builds  import IzkCurrExpBase


class IzkCurrExpDataHolder(DataHolder):
    def __init__(
        self,

        spikes_per_second=AbstractPopulationVertex.
        none_pynn_default_parameters['spikes_per_second'],

        ring_buffer_sigma=AbstractPopulationVertex.
        none_pynn_default_parameters['ring_buffer_sigma'],

        incoming_spike_buffer_size=AbstractPopulationVertex.
        none_pynn_default_parameters['incoming_spike_buffer_size'],

        constraints=AbstractPopulationVertex.none_pynn_default_parameters[
            'constraints'],

        label=AbstractPopulationVertex.none_pynn_default_parameters[
            'label'],
```

Python Build - DataHolder (2)

- This is only required since version 0.8 of PyNN

```
from spynnaker.pyNN.models.neuron  import AbstractPopulationVertex
from spynnaker8.utilities  import DataHolder
from spynnaker.pyNN.models.neuron.builds  import IzkCurrExpBase


class IzkCurrExpDataHolder(DataHolder):
    def __init__(
...
        a=IzkCurrExpBase.default_parameters[ 'a'],
        b=IzkCurrExpBase.default_parameters[ 'b'],
        c=IzkCurrExpBase.default_parameters[ 'c'],
        d=IzkCurrExpBase.default_parameters[ 'd'],
        i_offset=IzkCurrExpBase.default_parameters[ 'i_offset'],
        u_init=IzkCurrExpBase.default_parameters[ 'u_init'],
        v_init=IzkCurrExpBase.default_parameters[ 'v_init'],
        tau_syn_E=IzkCurrExpBase.default_parameters[ 'tau_syn_E'],
        tau_syn_I=IzkCurrExpBase.default_parameters[ 'tau_syn_I'],
        isyn_exc=IzkCurrExpBase.default_parameters[ 'isyn_exc'],
        isyn_inh=IzkCurrExpBase.default_parameters[ 'isyn_inh']):
    DataHolder.__init__(
        self,
        {'spikes_per_second': spikes_per_second,
          'ring_buffer_sigma': ring_buffer_sigma,
          'incoming_spike_buffer_size': incoming_spike_buffer_size,
          'constraints': constraints, 'label': label, 'a': a, 'b': b,
          'c': c, 'd': d, 'i_offset': i_offset, 'u_init': u_init,
          'v_init': v_init, 'tau_syn_E': tau_syn_E, 'tau_syn_I': tau_syn_I,
          'isyn_exc': isyn_exc, 'isyn_inh': isyn_inh})
```

Python Build - DataHolder (3)

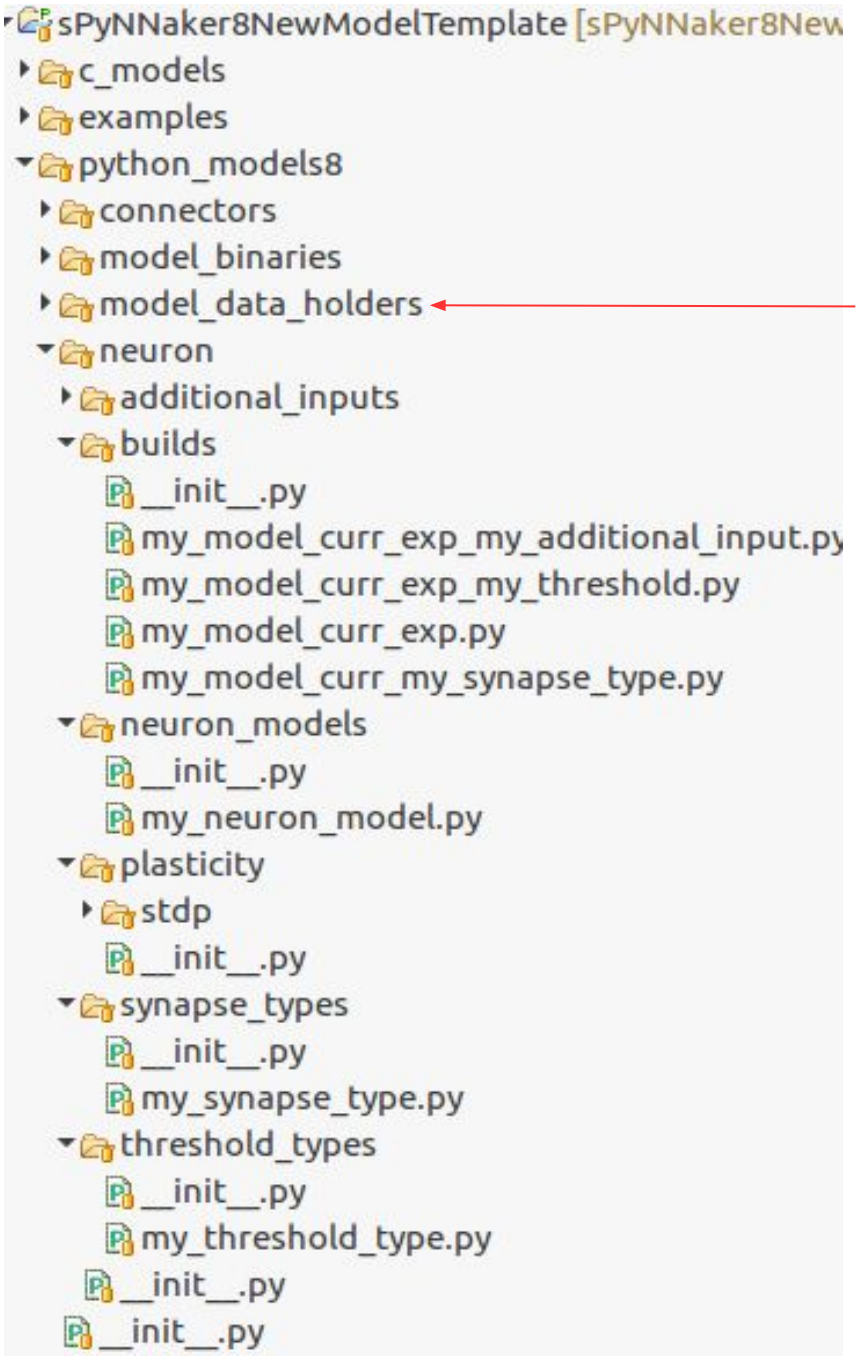
- This is only required since version 0.8 of PyNN

```
from spynnaker.pyNN.models.neuron  import AbstractPopulationVertex
from spynnaker8.utilities  import DataHolder
from spynnaker.pyNN.models.neuron.builds  import IzkCurrExpBase


class IzkCurrExpDataHolder(DataHolder):

...

    @staticmethod
    def build_model():
        return IzkCurrExpBase
```

New Model Template

C code template

Python code template

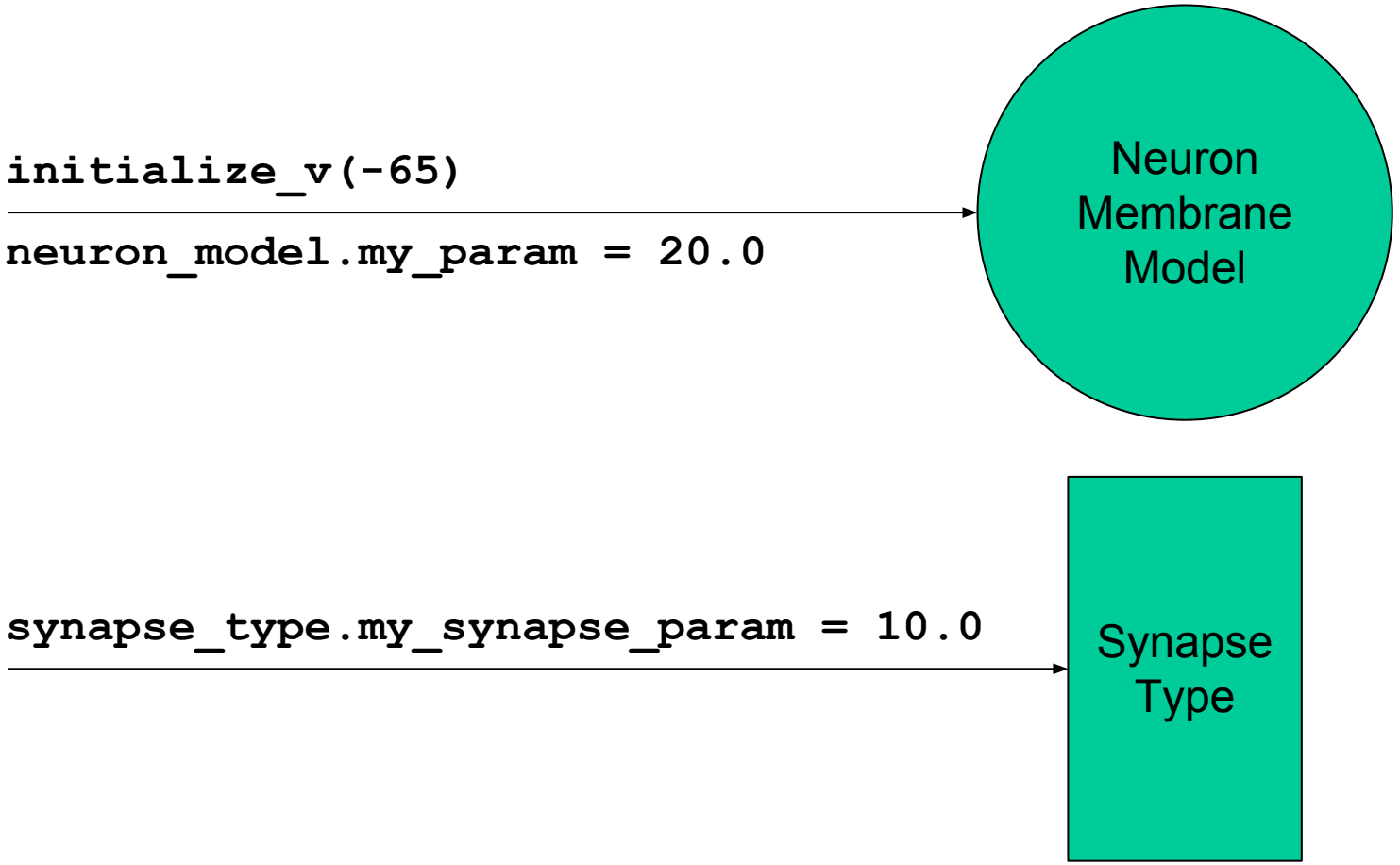


PyNN 0.8

sPyNNaker Neuron Models - PyNN

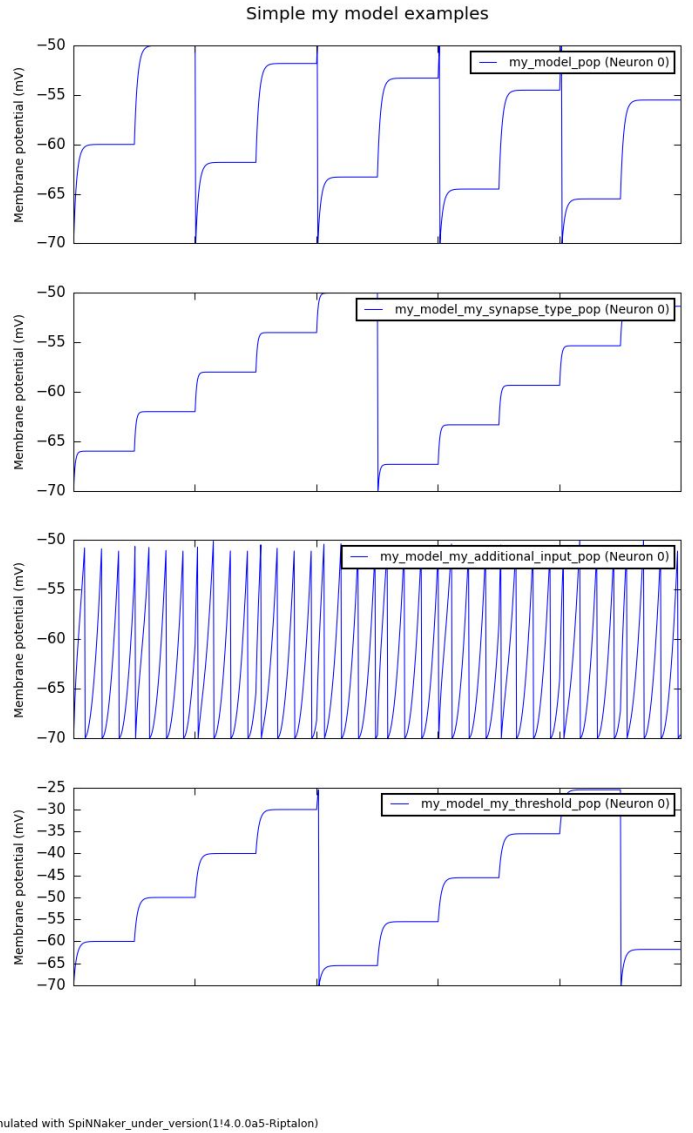
PyNN Script

```
pop.initialize(v=-65.0)   # initialize("v", -65.0)

pop.set(my_param=20.0)   # set("my_param", 20.0)

pop.set(my_synapse_param=10.0)
```



Using your model with PyNN

```
import pyNN.spiNNaker as p  (Check this...!)
from python_models8.model_data_holders.my_model_curr_exp_data_holder \
    import MyModelCurrExpDataHolder as My_Model_Curr_Exp

# set parameters
time_step = 1.0
weight = 2.0
spike_times = range( 0, run_time, 100)

p.setup(time_step)

spikeArray = {"spike_times": spike_times}
Input_pop = p.Population(
    1, p.SpikeSourceArray(**spikeArray), label= "input")

myModelCurrExpParams = {
    "my_parameter": -70.0,
    "i_offset": i_offset
}
my_model_pop = p.Population(
    1, My_Model_Curr_Exp(**myModelCurrExpParams),
    label="my_model_pop")
p.Projection(
    Input_pop, my_model_pop,
    p.OneToOneConnector(), receptor_type='excitatory',
    synapse_type=p.StaticSynapse(weight=weight))

# etc. etc.
```



Summary

- Define a new neuron model in C (solving the ODE), make sure that the parameters are correctly passed to it from Python / PyNN
- Define the different components of the model in a similar manner
- If a component function you require is not available in the NewModelTemplate, our advice is to copy the code in sPyNNaker and sPyNNaker8 and edit it

- Coming soon: a rewrite of NewModelTemplate to allow the user to define their neuron model in a more independent manner from its components

sPyNNaker Neuron Models - Lab

http://spinnakermanchester.github.io/spynnaker/4.0.0/PyNNOnSpiNNakerExtensions.html

or

http://spinnakermanchester.github.io

Running PyNN Scripts …. (1st link)

Extending sPyNNaker with New Neuron … (6th link)

or

See your paper version