The MUSIC of the PyNNs

a friendlier interface for modelling with multiple simulators

Andrew Davison
UNIC, CNRS, Gif-sur-Yvette
BrainScaleS CodeJam #6
Jülich, 28th January 2014

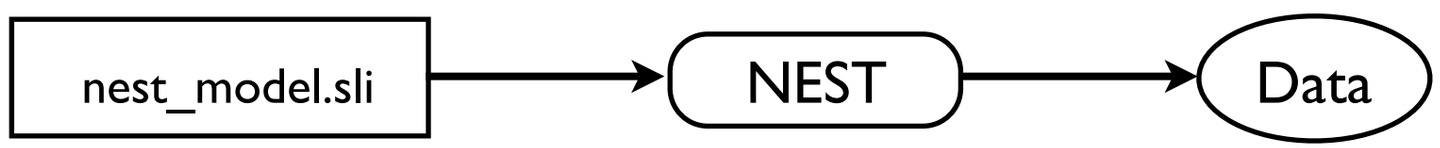
Verification, cross-checking and model sharing

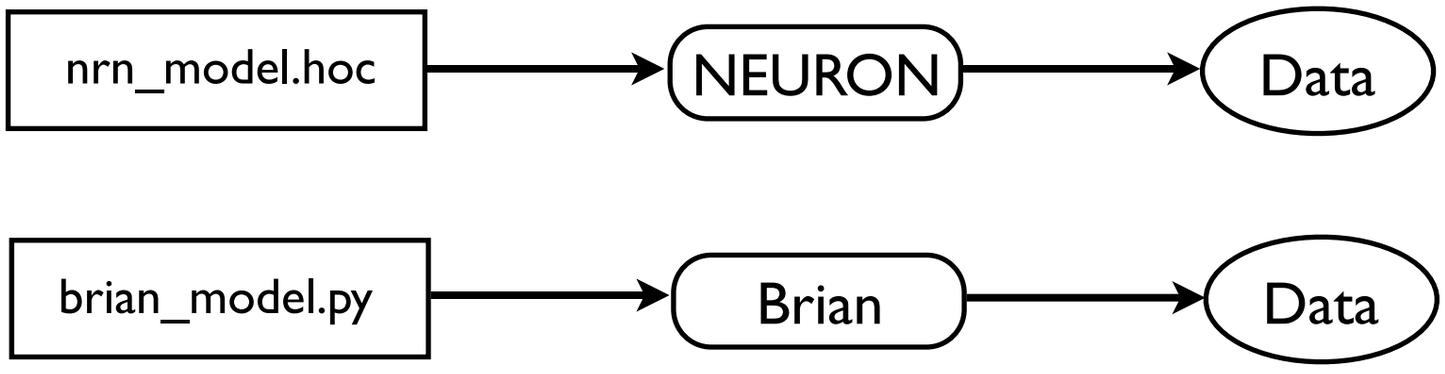
A single researcher or single lab cannot hope to model everything of interest.

Need to build on previous work: re-use and extend existing models.

But almost all models only run on a single simulator, and translation is challenging and time consuming.

Hence not reusable or testable.



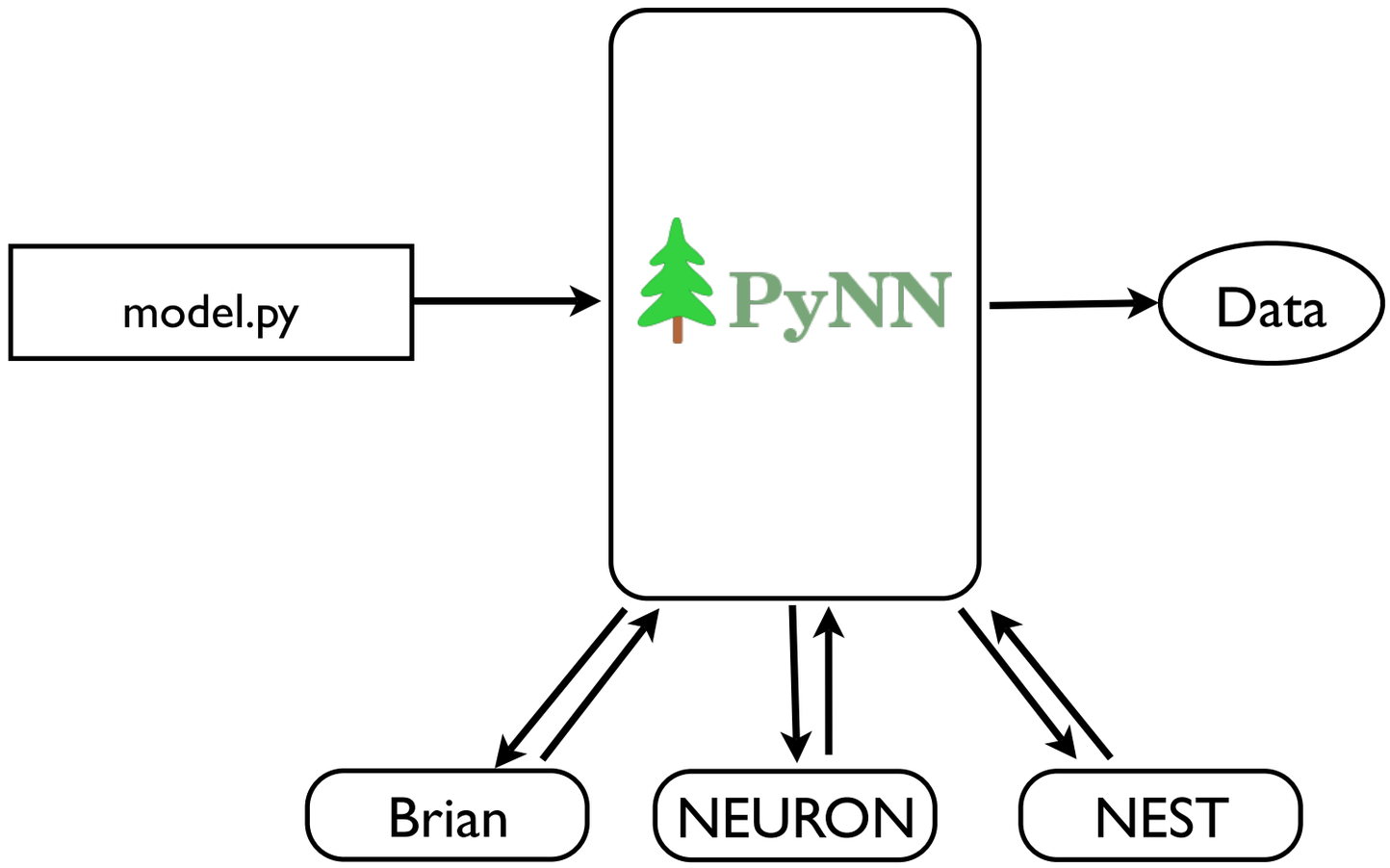


A common API for neuronal network modelling

"write the code for a simulation once, run it on any supported simulator without modification."

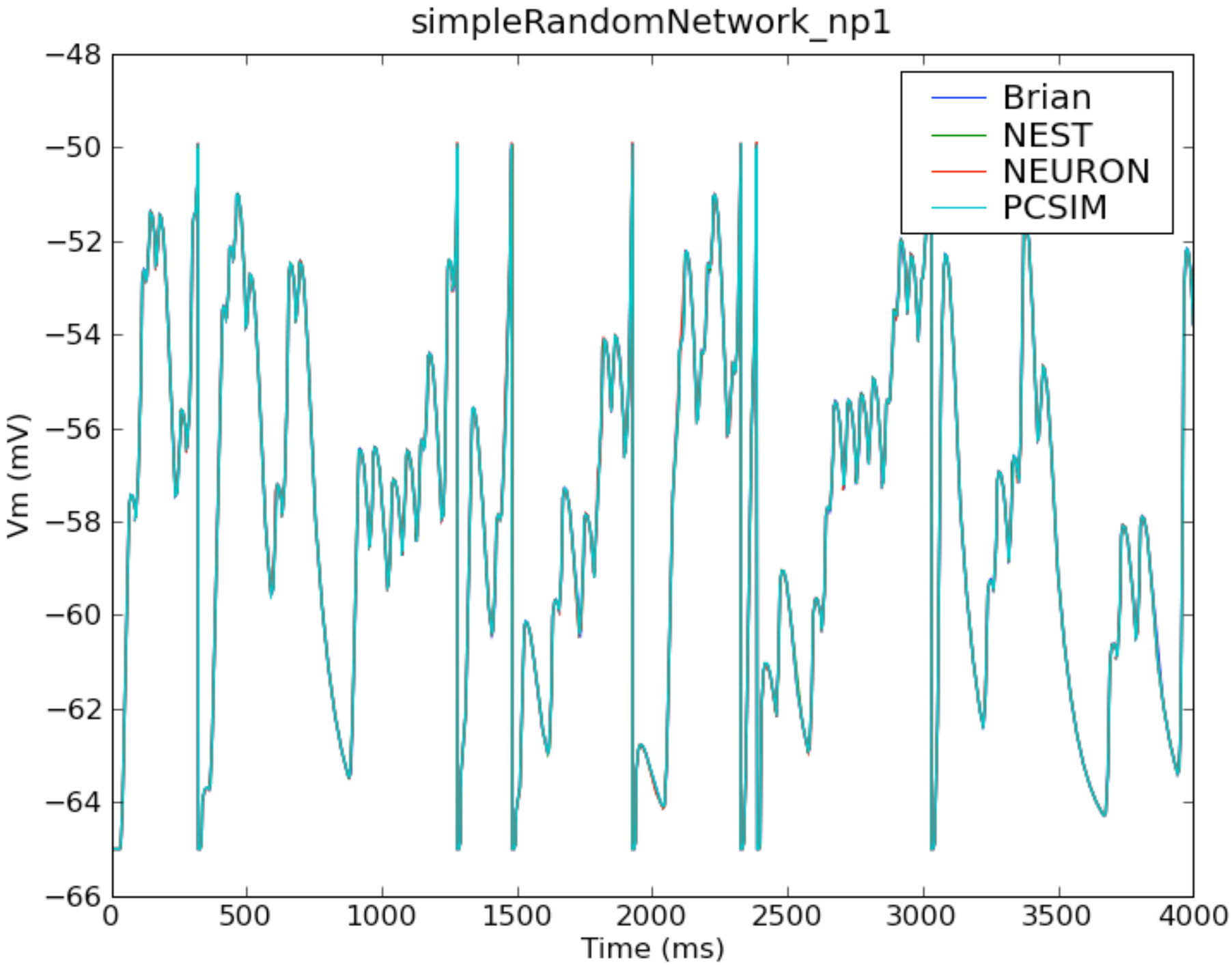
Goals:

- facilitate model sharing and reuse
- simplify validation of simulation results
- provide a common platform on which to build other tools (stimulation, analysis, visualization, GUIs)
- provide a more powerful API for neuronal network modelling (save scientist time)
- hide complexity of parallelization from user (increased computational efficiency without decreased scientist efficiency)



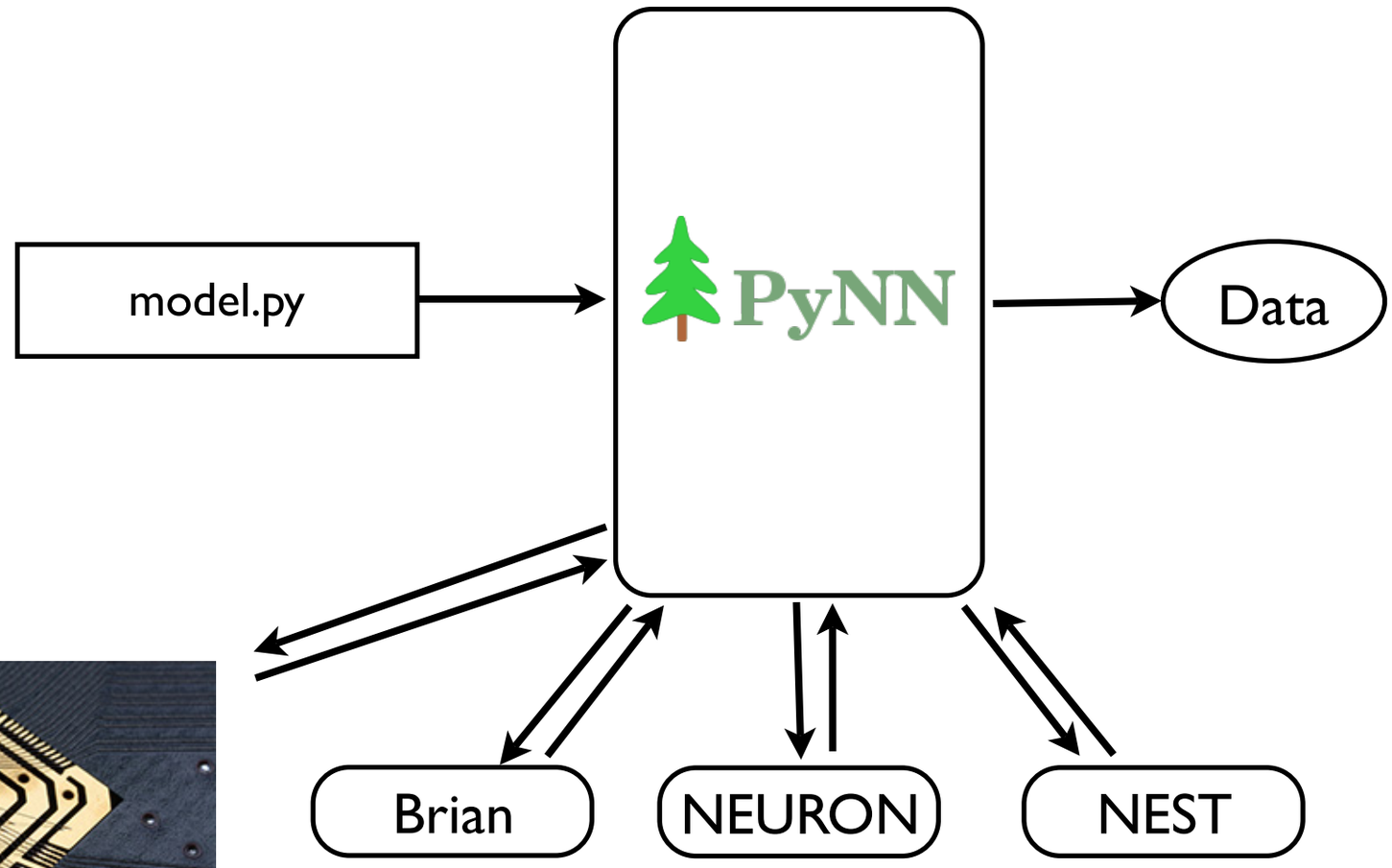
http://neuralensemble.org/PyNN/

A common API for neuronal network modelling



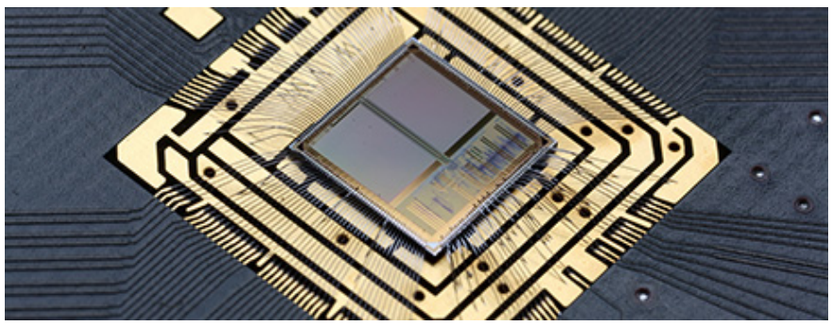
Neuromorphic computing

"write the code for a simulation once, run it on any supported simulator or hardware device without modification."



BrainScaleS

FACETS



neuromorphic hardware

http://neuralensemble.org/PyNN/

Installation

```
$ pip install PyNN   # stable (0.7.5)
```

0.8 beta - download from INCF Software Centre

Documentation

http://neuralensemble.org/PyNN/

Licence

CeCILL (GPL-equivalent)

Mailing list

https://groups.google.com/forum/#!forum/neuralensemble

```
sim.setup(timestep=0.1)
cell_parameters = {”tau_m”: 12.0, ”cm”: 0.8, ”v_thresh”: -50.0,
                   ”v_reset”: -65.0}
pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
                    label=”excitatory neurons”)
pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
                    label=”inhibitory neurons”)
all = pE + pI
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution(”normal”, (10.0, 2.0))
input.rset(”rate”, rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
all.inject(background)
weight_distr = random.RandomDistribution(”uniform”, (0.0, 0.1))
DDPC = sim.DistanceDependentProbabilityConnector
connector = DDPC(”exp(-d**2/400.0)”, weights=weight_distr,
                 delays=”0.5+0.01d”)
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
exc = sim.Projection(pE, all, connector, target=”excitatory”,
                     synapse_dynamics=plasticity)
inh = sim.Projection(pI, all, connector, target=”inhibitory”)
```

```
import pyNN.neuron as sim
sim.setup(timestep=0.1)
cell_parameters = {”tau_m”: 12.0, ”cm”: 0.8, ”v_thresh”: -50.0,
                   ”v_reset”: -65.0}
pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
                    label=”excitatory neurons”)
pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
                    label=”inhibitory neurons”)
all = pE + pI
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution(”normal”, (10.0, 2.0))
input.rset(”rate”, rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
all.inject(background)
weight_distr = random.RandomDistribution(”uniform”, (0.0, 0.1))
DDPC = sim.DistanceDependentProbabilityConnector
connector = DDPC(”exp(-d**2/400.0)”, weights=weight_distr,
                 delays=”0.5+0.01d”)
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
exc = sim.Projection(pE, all, connector, target=”excitatory”,
                     synapse_dynamics=plasticity)
inh = sim.Projection(pI, all, connector, target=”inhibitory”)
```

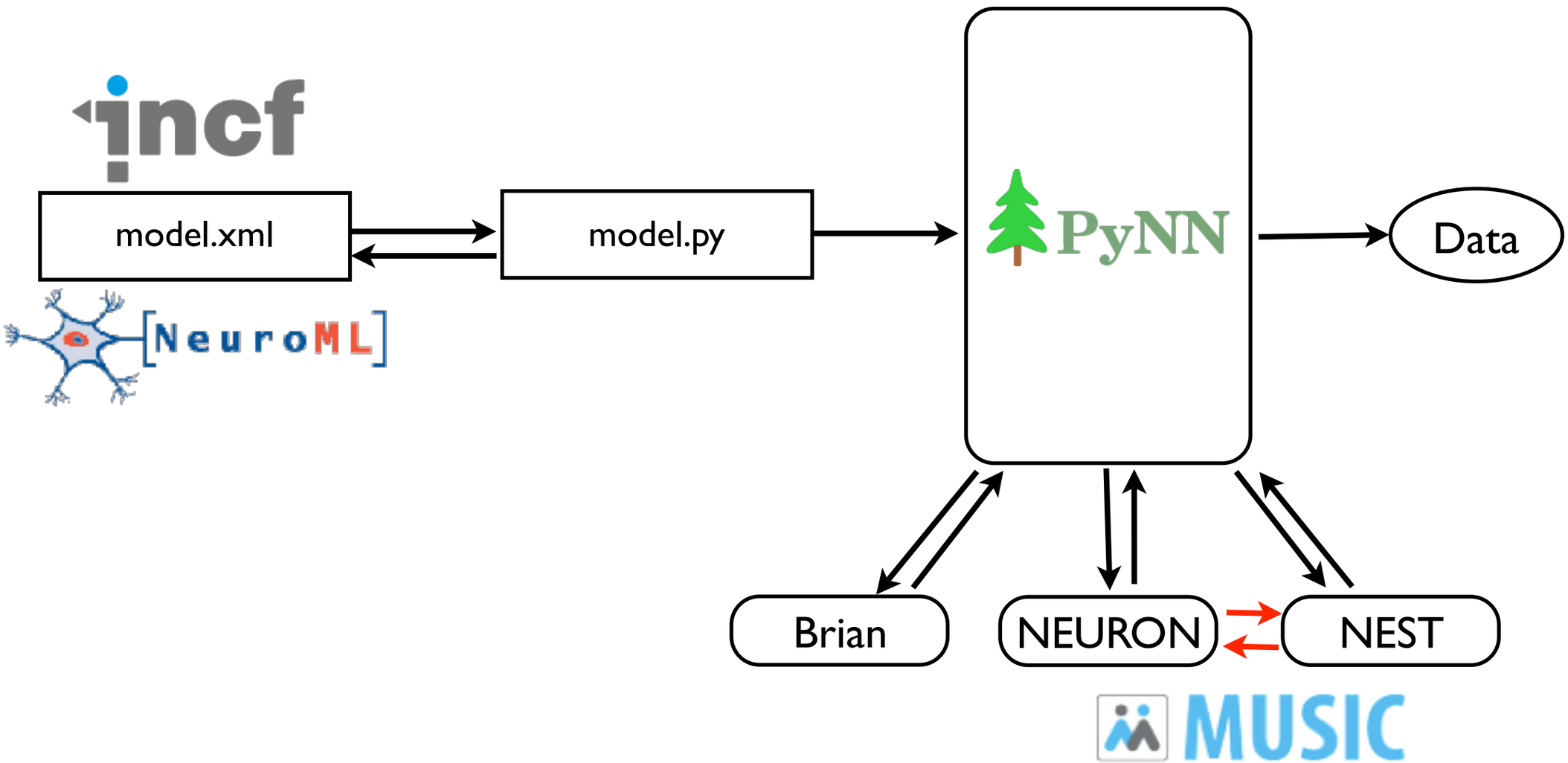
```
import pyNN.nest as sim
sim.setup(timestep=0.1)
cell_parameters = {”tau_m”: 12.0, ”cm”: 0.8, ”v_thresh”: -50.0,
                   ”v_reset”: -65.0}
pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
                    label=”excitatory neurons”)
pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
                    label=”inhibitory neurons”)
all = pE + pI
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution(”normal”, (10.0, 2.0))
input.rset(”rate”, rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
all.inject(background)
weight_distr = random.RandomDistribution(”uniform”, (0.0, 0.1))
DDPC = sim.DistanceDependentProbabilityConnector
connector = DDPC(”exp(-d**2/400.0)”, weights=weight_distr,
                 delays=”0.5+0.01d”)
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
exc = sim.Projection(pE, all, connector, target=”excitatory”,
                     synapse_dynamics=plasticity)
inh = sim.Projection(pI, all, connector, target=”inhibitory”)
```

```
import pyNN.brian as sim
sim.setup(timestep=0.1)
cell_parameters = {”tau_m”: 12.0, ”cm”: 0.8, ”v_thresh”: -50.0,
                   ”v_reset”: -65.0}
pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
                    label=”excitatory neurons”)
pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
                    label=”inhibitory neurons”)
all = pE + pI
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution(”normal”, (10.0, 2.0))
input.rset(”rate”, rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
all.inject(background)
weight_distr = random.RandomDistribution(”uniform”, (0.0, 0.1))
DDPC = sim.DistanceDependentProbabilityConnector
connector = DDPC(”exp(-d**2/400.0)”, weights=weight_distr,
                 delays=”0.5+0.01d”)
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
exc = sim.Projection(pE, all, connector, target=”excitatory”,
                     synapse_dynamics=plasticity)
inh = sim.Projection(pI, all, connector, target=”inhibitory”)
```

```
import pyHMF as sim
sim.setup(timestep=0.1)
cell_parameters = {”tau_m”: 12.0, ”cm”: 0.8, ”v_thresh”: -50.0,
                   ”v_reset”: -65.0}
pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
                    label=”excitatory neurons”)
pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
                    label=”inhibitory neurons”)
all = pE + pI
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution(”normal”, (10.0, 2.0))
input.rset(”rate”, rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
all.inject(background)
weight_distr = random.RandomDistribution(”uniform”, (0.0, 0.1))
DDPC = sim.DistanceDependentProbabilityConnector
connector = DDPC(”exp(-d**2/400.0)”, weights=weight_distr,
                 delays=”0.5+0.01d”)
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
exc = sim.Projection(pE, all, connector, target=”excitatory”,
                     synapse_dynamics=plasticity)
inh = sim.Projection(pI, all, connector, target=”inhibitory”)
```

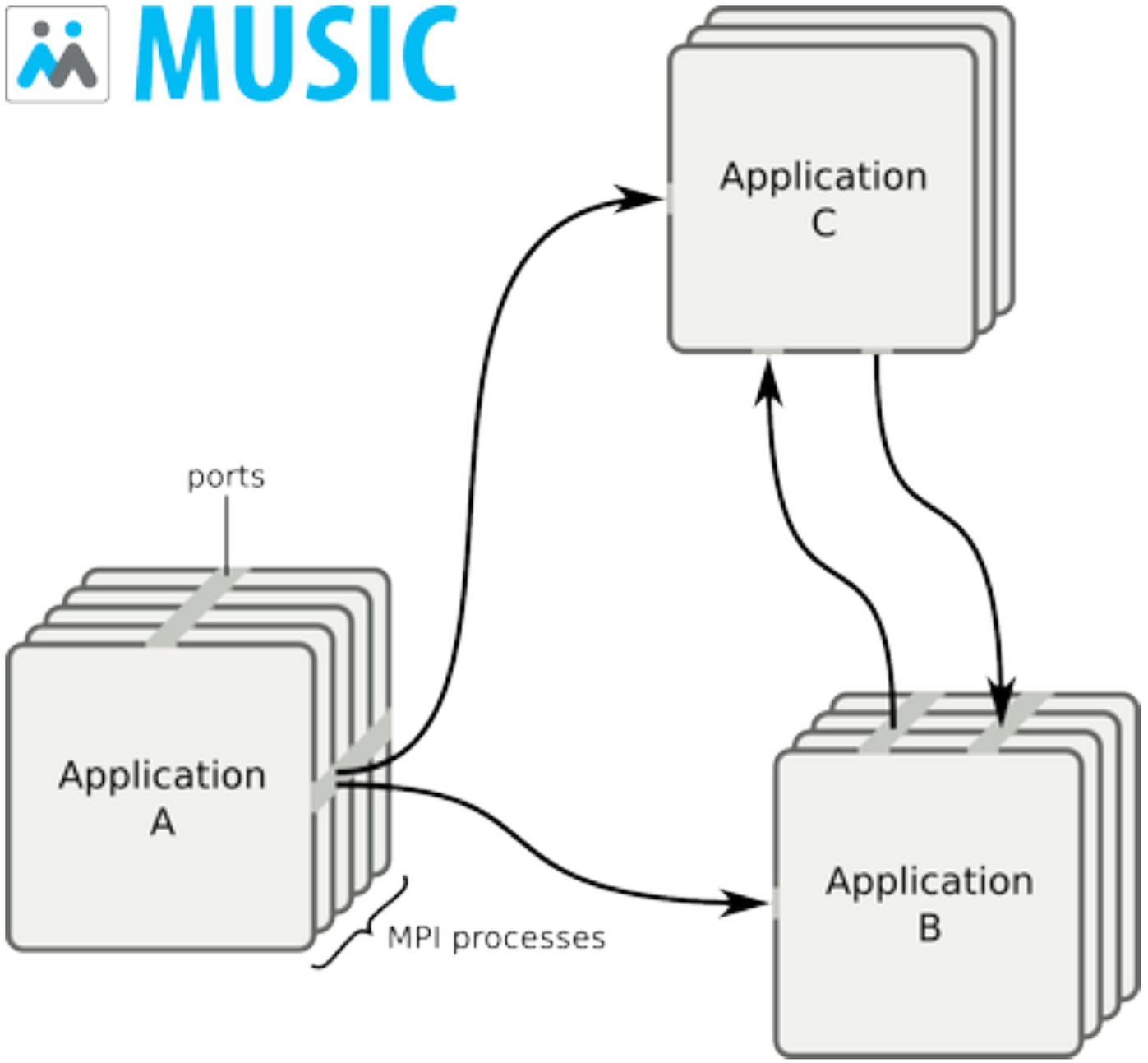
```
import pyNN.spiNNaker as sim
sim.setup(timestep=0.1)
cell_parameters = {”tau_m”: 12.0, ”cm”: 0.8, ”v_thresh”: -50.0,
                   ”v_reset”: -65.0}
pE = sim.Population((100,100), sim.IF_cond_exp, cell_parameters,
                    label=”excitatory neurons”)
pI = sim.Population((50,50), sim.IF_cond_exp, cell_parameters,
                    label=”inhibitory neurons”)
all = pE + pI
input = sim.Population(100, sim.SpikeSourcePoisson)
rate_distr = random.RandomDistribution(”normal”, (10.0, 2.0))
input.rset(”rate”, rate_distr)
background = sim.NoisyCurrentSource(mean=0.1, stdev=0.01)
all.inject(background)
weight_distr = random.RandomDistribution(”uniform”, (0.0, 0.1))
DDPC = sim.DistanceDependentProbabilityConnector
connector = DDPC(”exp(-d**2/400.0)”, weights=weight_distr,
                 delays=”0.5+0.01d”)
TMM = sim.TsodyksMarkramMechanism
depressing = sim.DynamicSynapse(fast=TMM(U=0.5,tau_rec=800.0))
exc = sim.Projection(pE, all, connector, target=”excitatory”,
                     synapse_dynamics=plasticity)
inh = sim.Projection(pI, all, connector, target=”inhibitory”)
```

A common API for neuronal network modelling



Multi-simulations

MUSIC




```
$ mpirun -np 128 music conf.music
```

conf.music

```
stoptime=1.0
[A]
  binary=my_lgn_model
  np=32
  out -> B.in
  out -> C.in
[B]
  binary=nest
  args=...
  np=32
  to_L5_pyramidal -> C.in
  from_L5_pyramidal <- C.out
[C]
  binary=nrniv
  args=my_simulation.hoc
  np=64
```

MUSIC: multi-simulation coordinator

- An API allowing large scale neuron simulators which use MPI internally to exchange data during runtime
- An implementation of a C++ library providing this API

The purpose of MUSIC

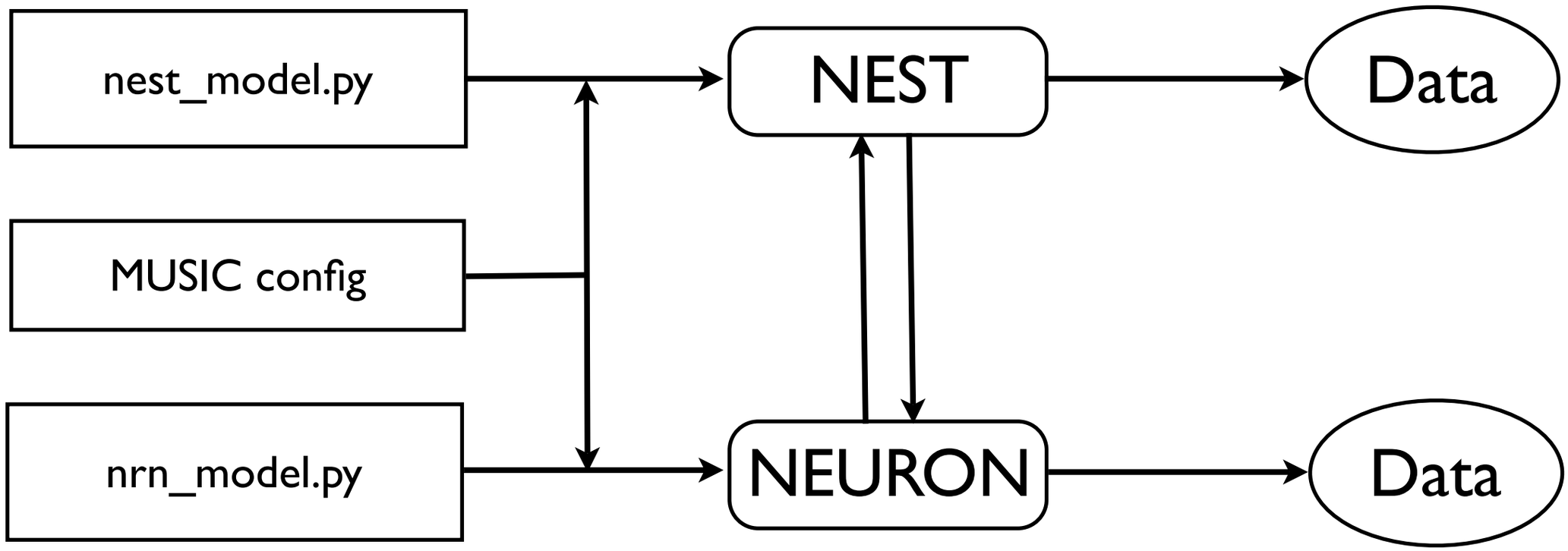
Allow multiple applications to run together and communicate within the parallel computer

- On-line pre- or post-processing of huge amounts of data for a parallel simulator within the cluster
- Connect models developed for different parallel simulators
- Run multiple instances of the same simulator on different parameter sets in one parallel job
- Promote re-usability through modularity

Supporting simulators

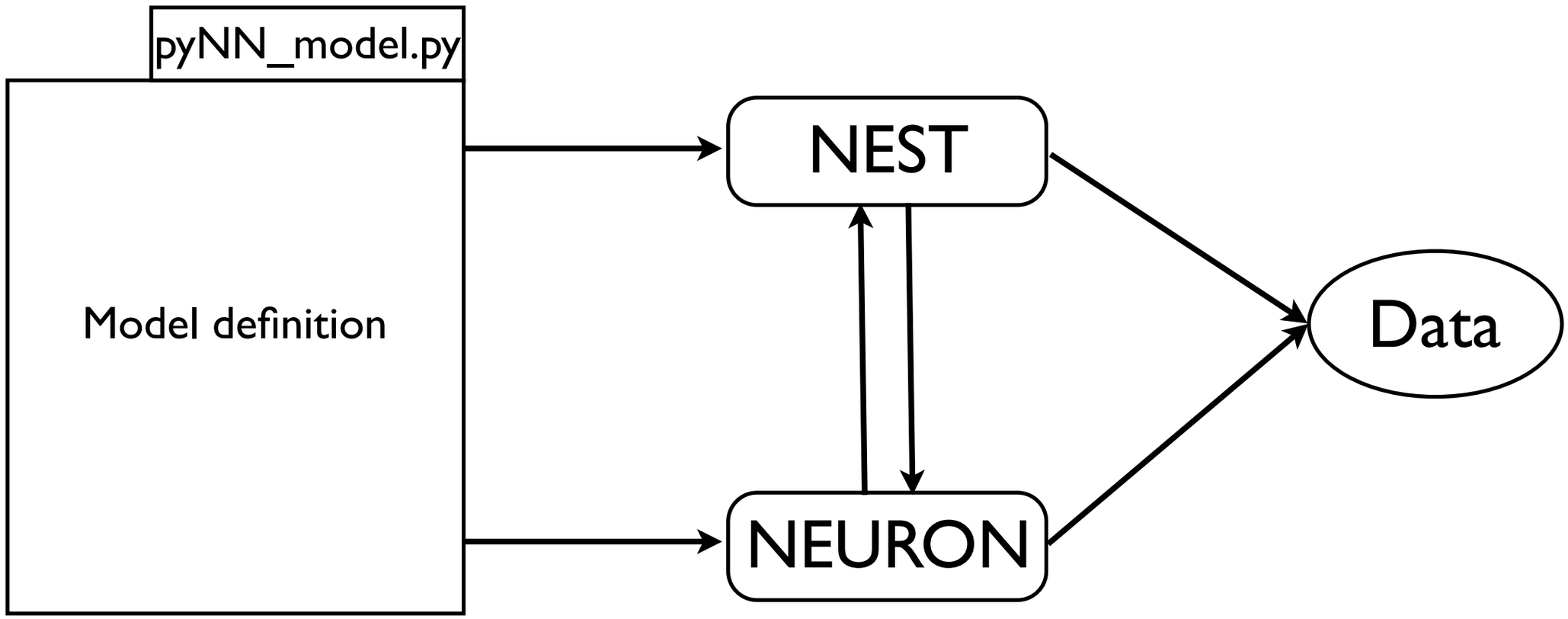
- NEST — Moritz Helias and Jochen Eppler
- MOOSE — Niraj Dudani and Johannes Hjorth
- NEURON — Michael Hines

Multi-simulations




```
$ mpirun music conf.music
```

Multi-simulations in PyNN




```
$ mpirun -np 31 python pyNN_model.py
```

Mikael Djurfeldt, Eilif Muller, Jochen Eppler

Multi-simulations in PyNN

```
from pyNN import music

vizapp = music.Config("vizapp", 1, "/path/to/vizapp", "args")
sim1, sim2, viz = music.setup(music.Config("neuron", 10),
                              music.Config("nest", 20),
                              vizapp)

sim1.setup(timestep=0.025)
sim2.setup(timestep=0.1)
pE = sim1.Population(10000, sim.IF_cond_exp(), label="excitatory")
pI = sim2.Population(2500, sim.IF_cond_exp(), label="inhibitory")
```

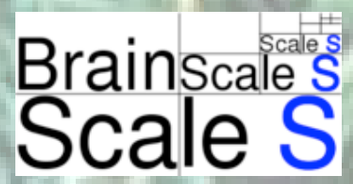
```
def connector(sim):
    DDPC = getattr(sim, "DistanceDependentProbabilityConnector")
    return DDPC("exp(-d**2/400.0)", weights=0.05, delays="0.5+0.01d")
e2e = sim1.Projection(pE, pE, connector(sim1),
                      receptor_type="excitatory")
e2i = music.Projection(pE, pI, connector(sim2),
                       receptor_type="excitatory")
i2i = sim2.Projection(pI, pI, connector(sim2),
                      receptor_type="inhibitory")

output = music.Port(pE, "spikes", viz, "pE_spikes_viz")

music.run(1000.0)
```

http://neuralensemble.org/PyNN

EC (FACETS, BrainScaleS)
CNRS



Pierre Yger
Daniel Brüderle
Jens Kremkow
Mike Hull
Mikael Djurfeldt
Subhasis Ray
Jan Antolik
Joël Chavas

Eilif Muller
Jochen Eppler
Dejan Pecevski
Michael Schmuker
Bernhard Kaplan
Yury Zaytsev
Tom Close

@apdavison
http://andrewdavison.info

PyNN

Hear The Music Of The Pines

Hear the music of the pines-
Murmuring through the climbing vines,
Sighing through the tree tops high,
Floating upward to the sky,
Then descending where I lie-
Hear the music of the pines!

What sweet thoughts the music brings,
What new gladness from it springs-
As reclining, in a dream,
Watch I, listless, a sunbeam
Dancing on the silvery stream-
What sweet thoughts the music brings!

Hear the music of the pines!
How it 'round my fancy twines-
While fragrances of flowers fill
All the pulses of my will
As I, lingering, linger still-
Hear the music of the pines!

by Timothy Thomas Fortune (1856-1928).