The PyNN module for the FACETS waferscale neuromorphic hardware system

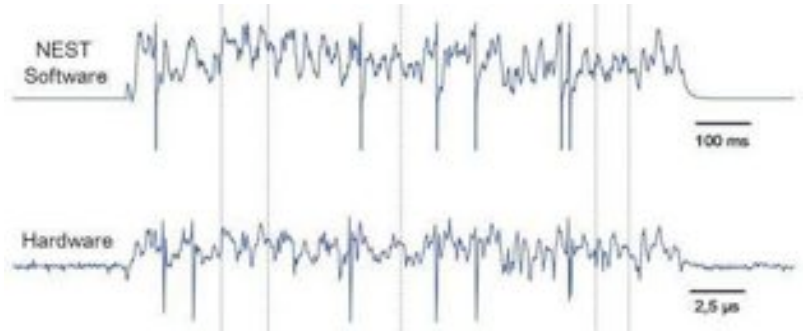
`pyNN.hardware.stage2` (& `stage1`)

Eric Müller

`http://www.kip.uni-heidelberg.de/vision/`
Kirchhoff Institute for Physics,
University of Heidelberg

2009-10-09

FACETS Hardware Stage 1



- 384 gLIF neurons, $\sim 98\,\mathrm{k}$ synapses
- highly accelerated operation (speed-up $10^4 - 10^5$)
- long- and short-term plasticity
- mapping biological $\Leftrightarrow$ hardware value domain
- remote access, configurable neuron and synapse parameters
- measurable: spike output, membrane potential, weights

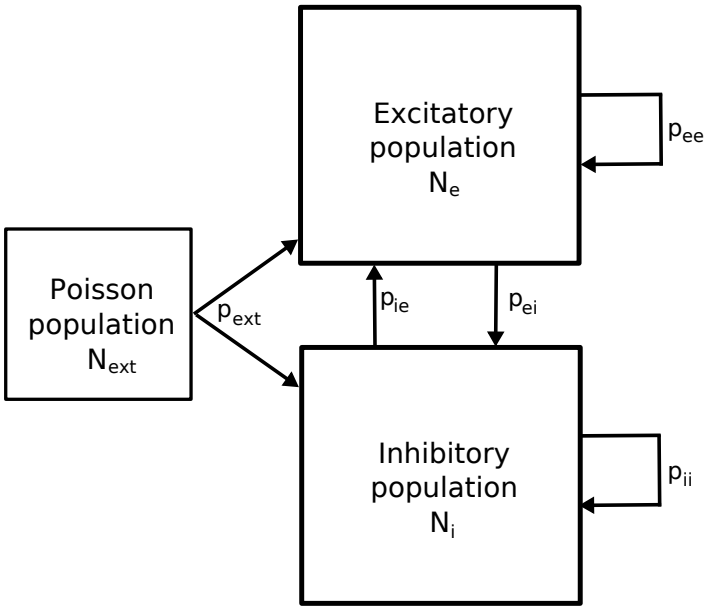
Schemmel et al.: Modeling Synaptic Plasticity within Networks of Highly Accelerated I&F Neurons, ISCAS 2007

Software status

- `pyNN.hardware.stage1`:
  - procedural PyNN interface
  - object-oriented PyNN interface
  - simple check of hardware constraints
- up & running
- experiments ...

Recurrent network dynamics

- Daniel Brüderle[a] & Jens Kremkow[b]
- compare the Stage 1 hardware system to a simulator
- systematical parameter sweep to generate different states of network activity
- e.g. this recurrent network architecture

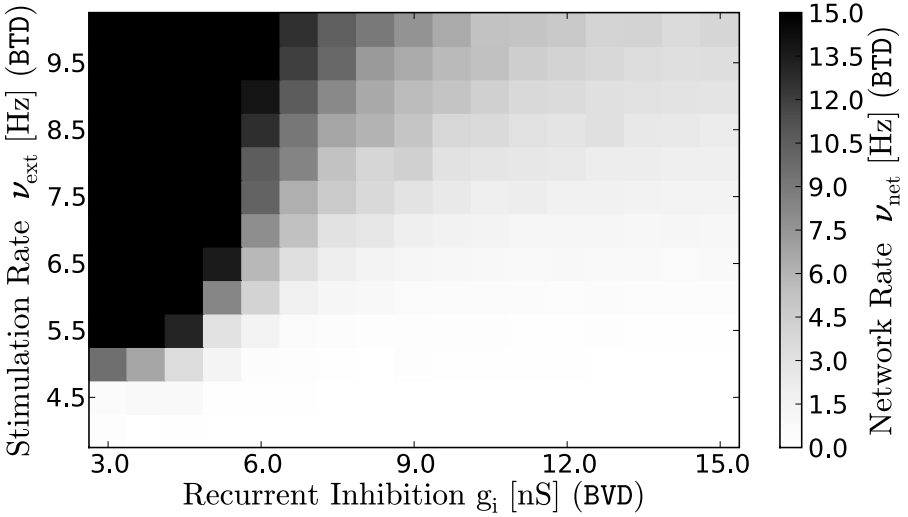


[a] University of Heidelberg
[b] CNRS, Marseille, France

Recurrent network dynamics

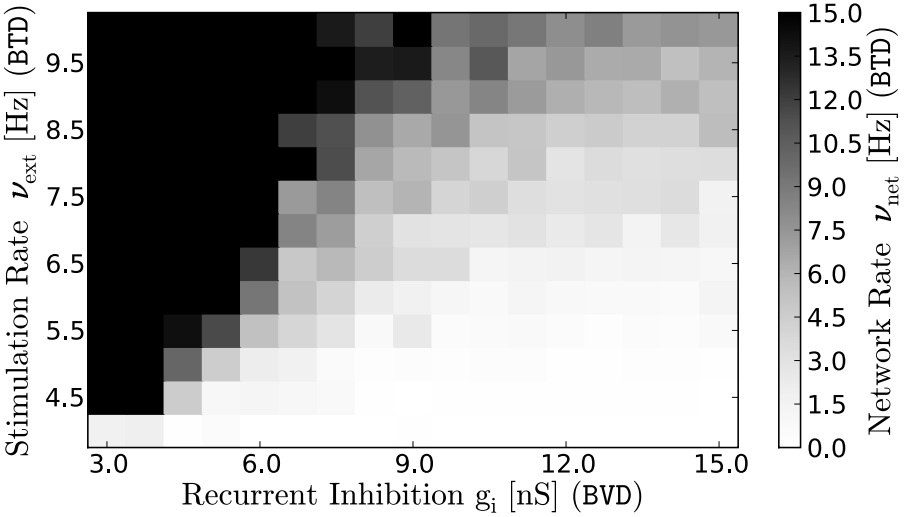
Firing rate

Stage 1



(a) Hardware: Firing Rate ν_{net}

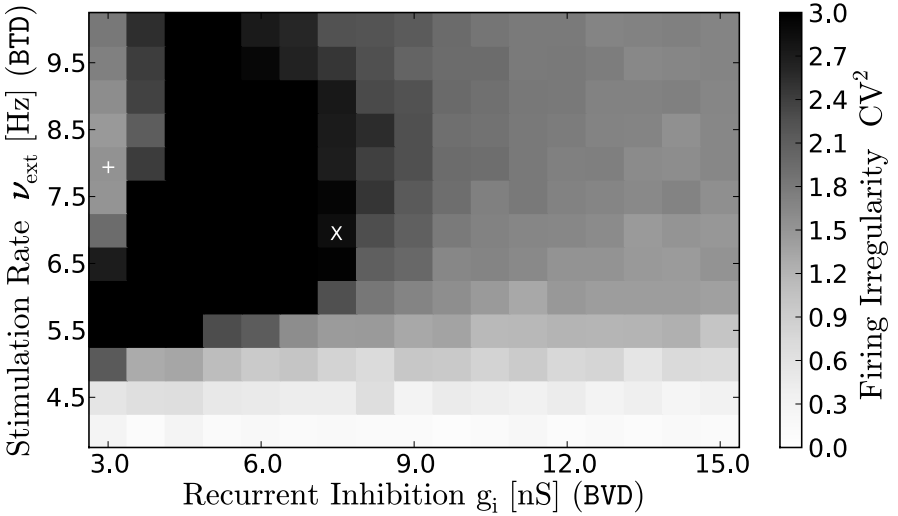
Nest



(b) NEST: Firing Rate ν_{net}

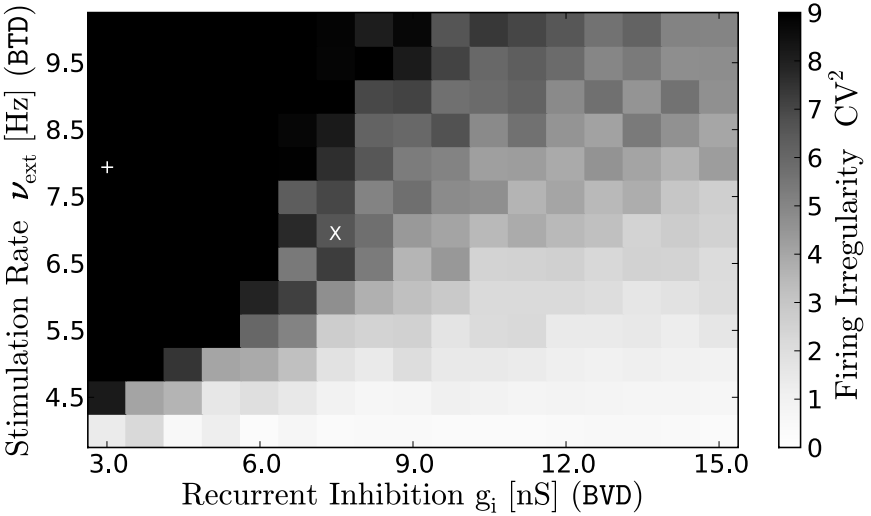
Irregularity

Stage 1



(c) Hardware: Irregularity CV^2

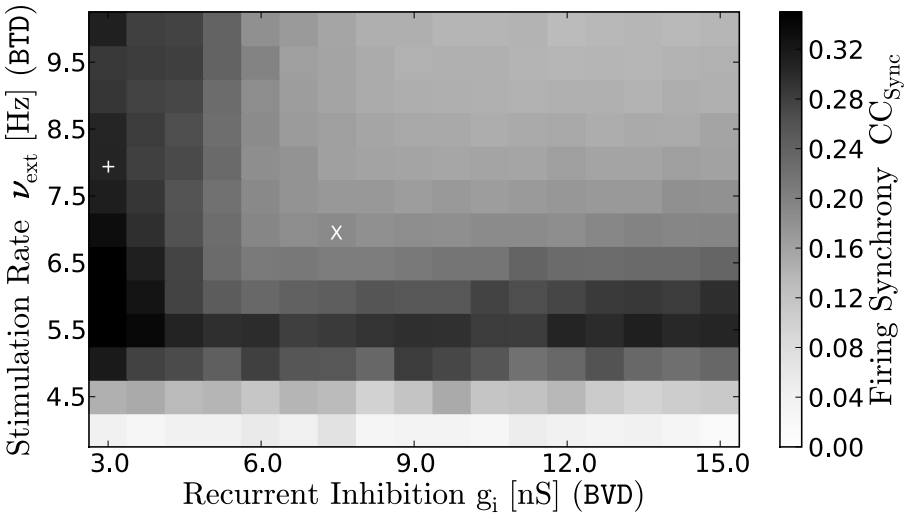
Nest



(d) NEST: Irregularity CV^2

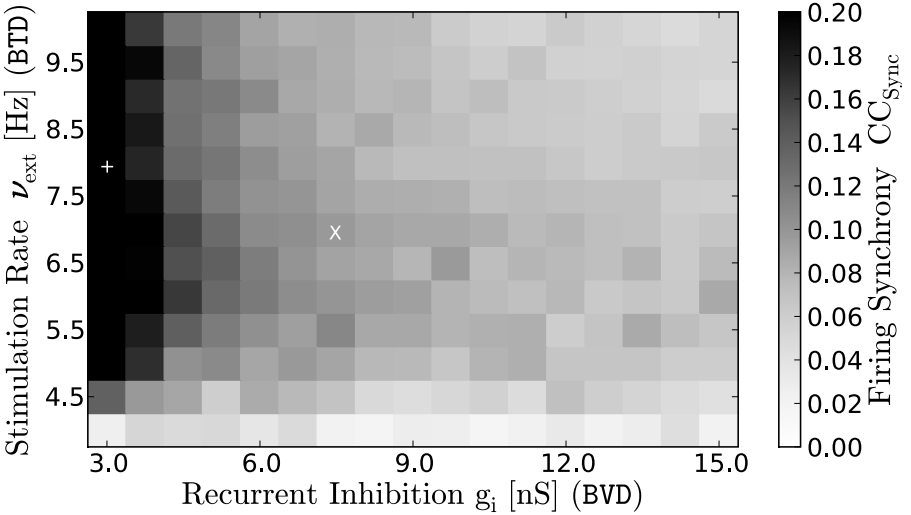
Synchrony

Stage 1



(e) Hardware: Synchrony CC_{Sync}

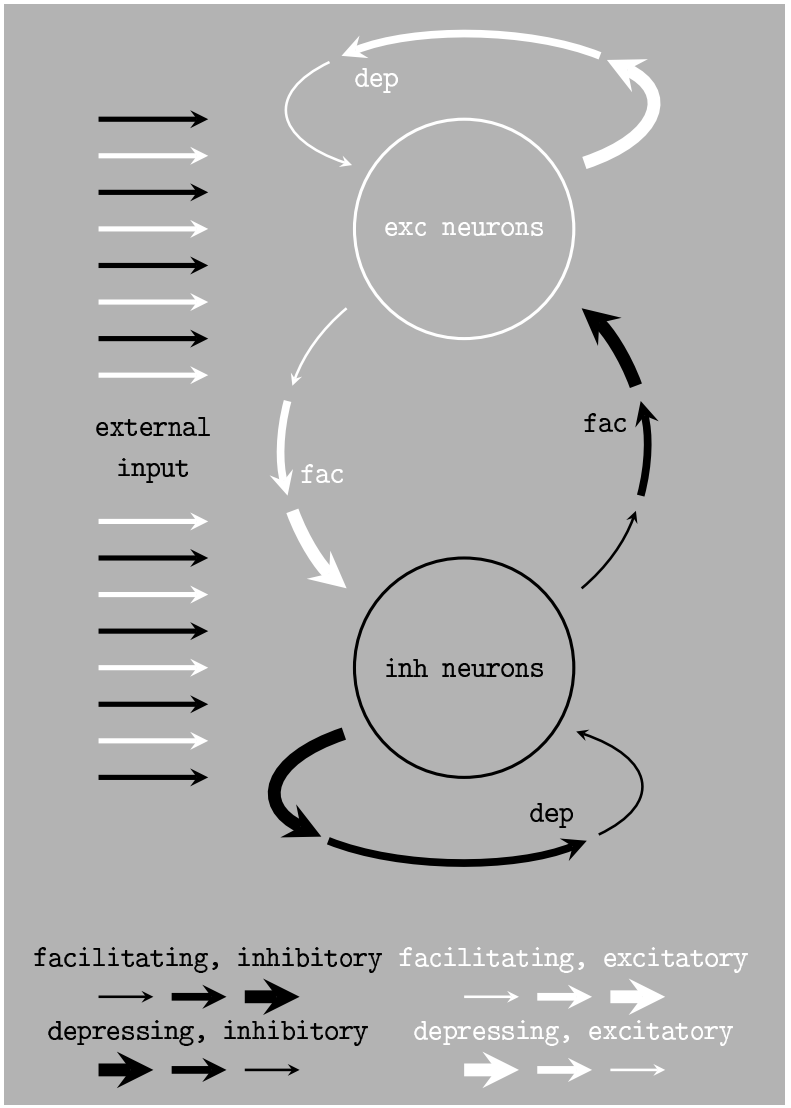
Nest



(f) NEST: Synchrony CC_{Sync}

Self-Stabilizing Network Architectures

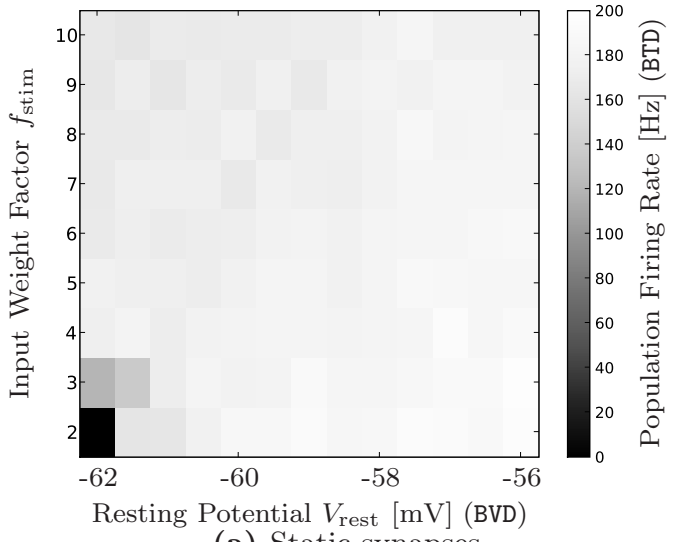
- Johannes Bill[a] & Klaus Schuch[b]
- inspired by Sussillo et al., 2007
- excitatory population P_e comprising $N_e = 144\,\text{neurons}$
- inhibitory population P_i consisting of $N_i = 48\,\text{neurons}$
- interconnected via depressing and facilitating synapses – see schematic



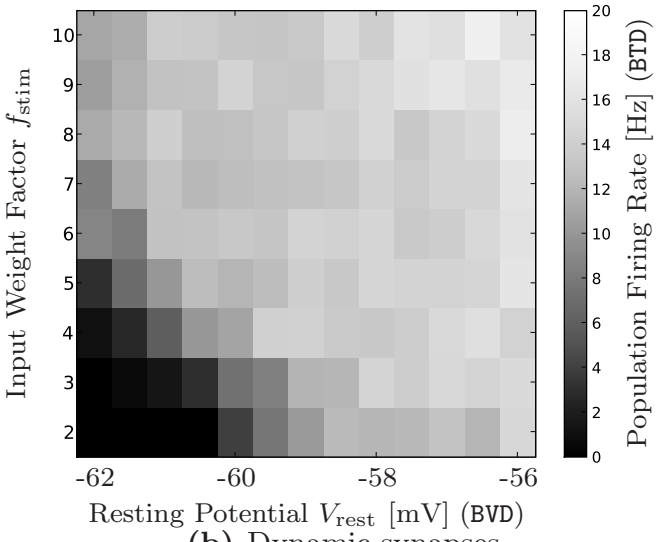
[a]University of Heidelberg/TU Graz
[b]TU Graz

Self-Stabilizing Network Architectures

HW

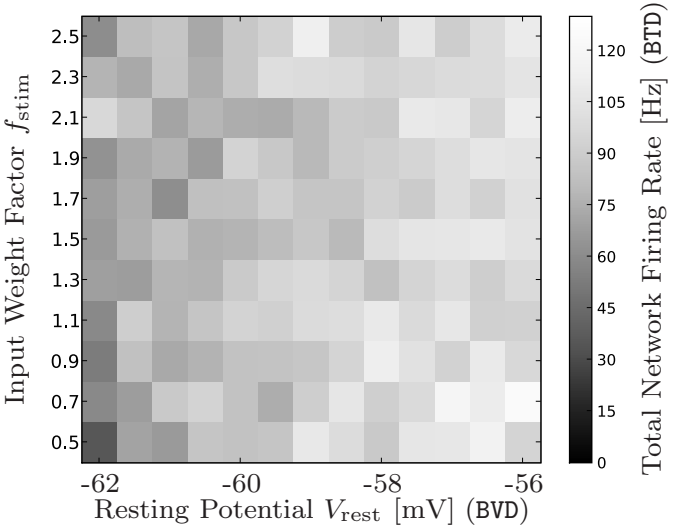


(a) Static synapses.

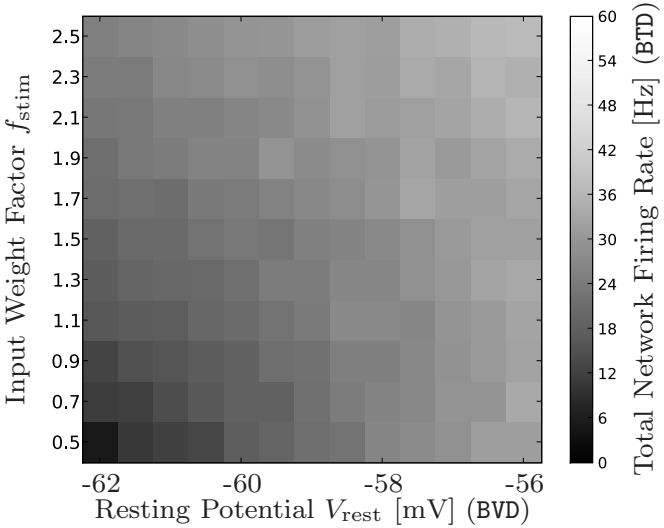


(b) Dynamic synapses.

PC-SIM



(c) Case C: Internal synapses static and strong.

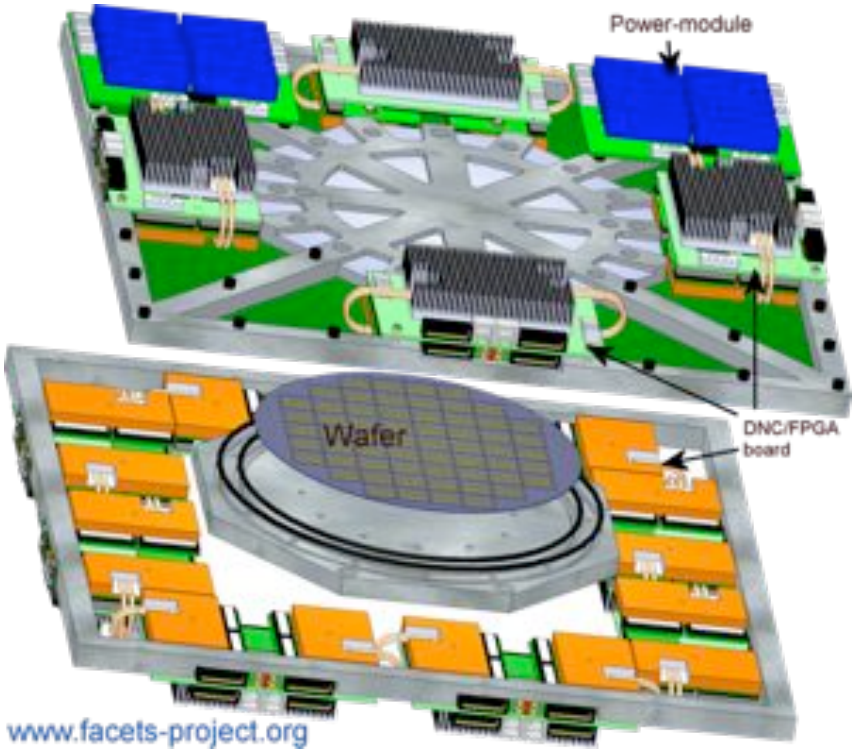


(d) Case D: Internal synapses dynamic and strong.

Goals

- comparing simulators and real hardware implementation
  - chip specification, debugging
- cooperation with external users
  - non-expert usability $\Rightarrow$ PyNN

FACETS Hardware Stage 2



- first HICANN[a] chip prototype produced
  - $\sim 110\,\mathrm{k}$ hw-synapses,
    ≤ 512 AdEx neurons
    $\#_{\mathrm{inputs}} \times \#_{\mathrm{neurons}} = \#_{\mathrm{hwsynapses}}$
  - long- and short-term plasticity
  - accelerated operation (speed-up 10^4)
- wafer system (2010?): ~ 350 HICANNs
- basic functionality test in preparation
- development of low level control software

[a] High Input Count Analog Neural Network

Status – pyNN.hardware.stage2

- first use case: FACETS Demonstrator
  - an emulation of the real hardware system
  - low-level software connecting the hardware system and the existing simulation software stack is still missing
- procedural PyNN-API supported, except for
  - `record_gsyn`, `*_gsyn` – hardware constraint, g_{syn} not observable
  - `record_v` – hardware solution unknown (ADCs?) so far, recording via oscilloscope possible
- basic translation from object-oriented API to low-level API working

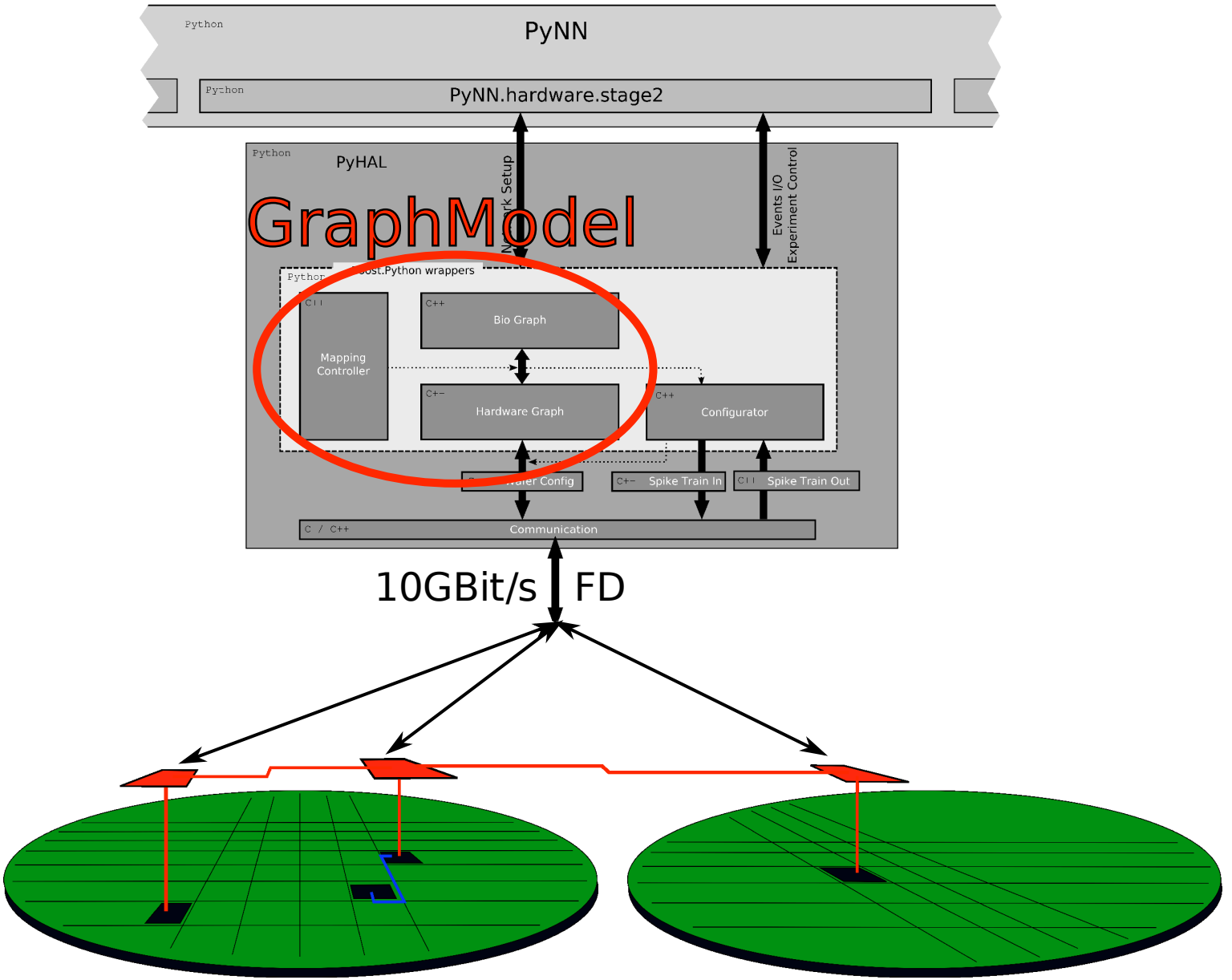
Setup – Graph Model

Operating Interface

Mapping Software

Digital (Layer2)

Analog (Layer1)



Bio model

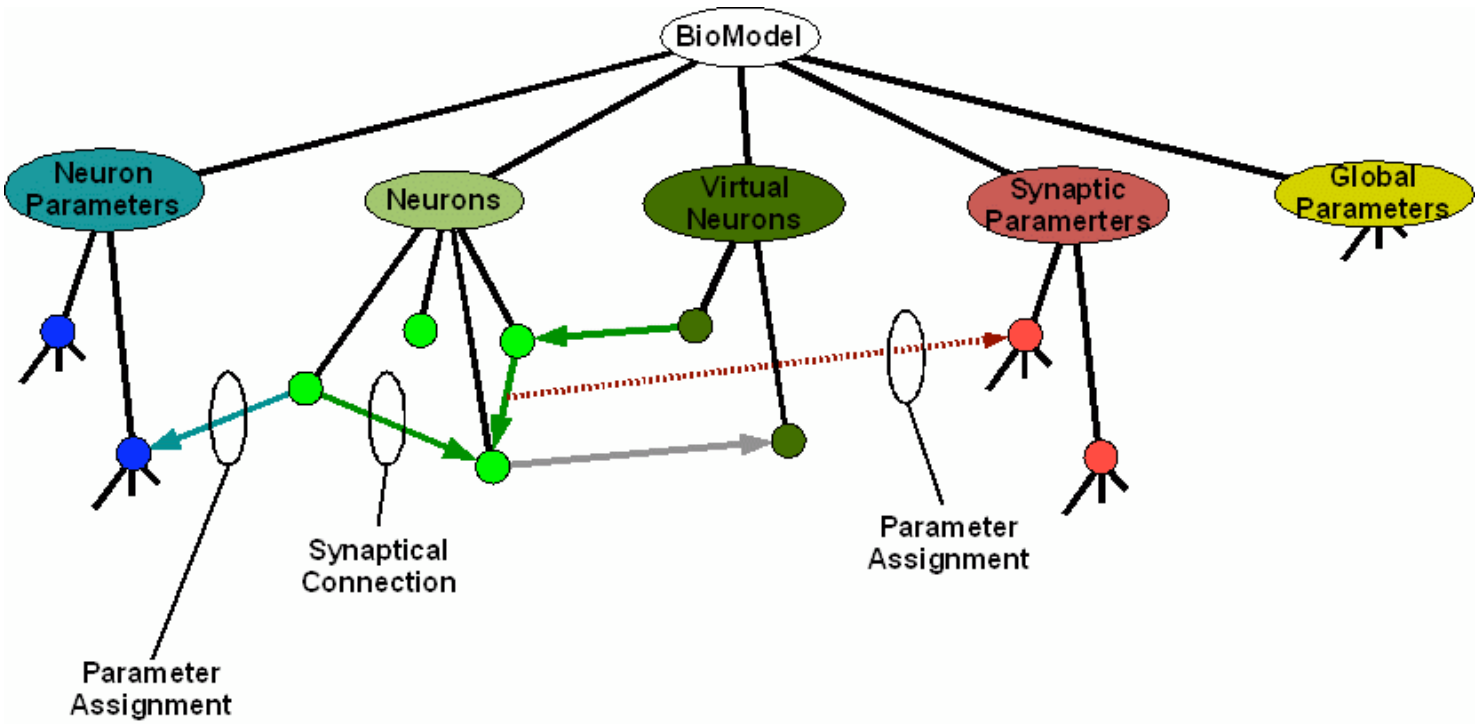


Figure by TUD

Hardware model

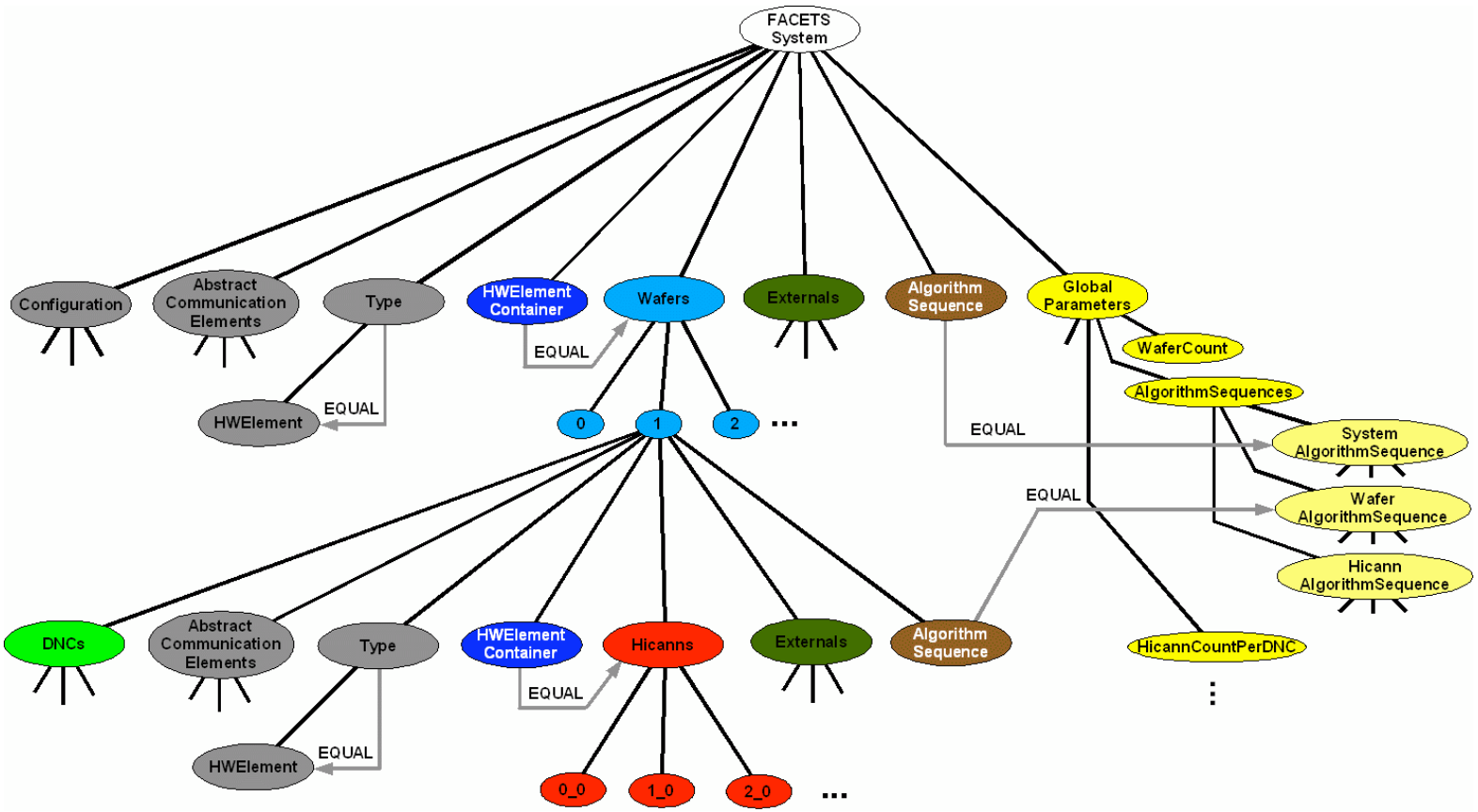
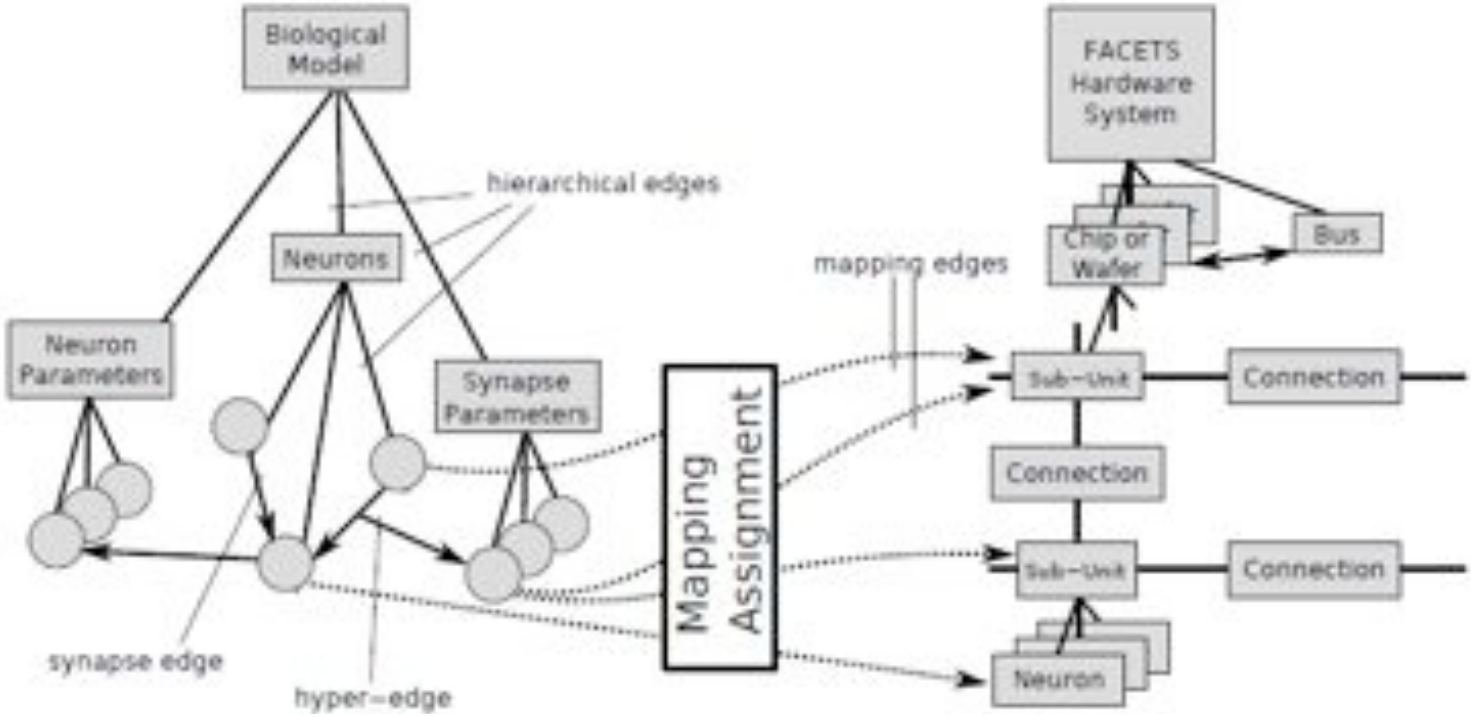


Figure by TUD

Mapping between models



Routing

- 64 horizontal busses per HICANN row $\Rightarrow$ approx. $1\,\mathrm{k}$ per wafer
- 256 vertical busses per HICANN column $\Rightarrow$ approx. $9\,\mathrm{k}$ per wafer
- sparse switch matrices connect vertical, horizontal busses & HICANNs
- complex routing

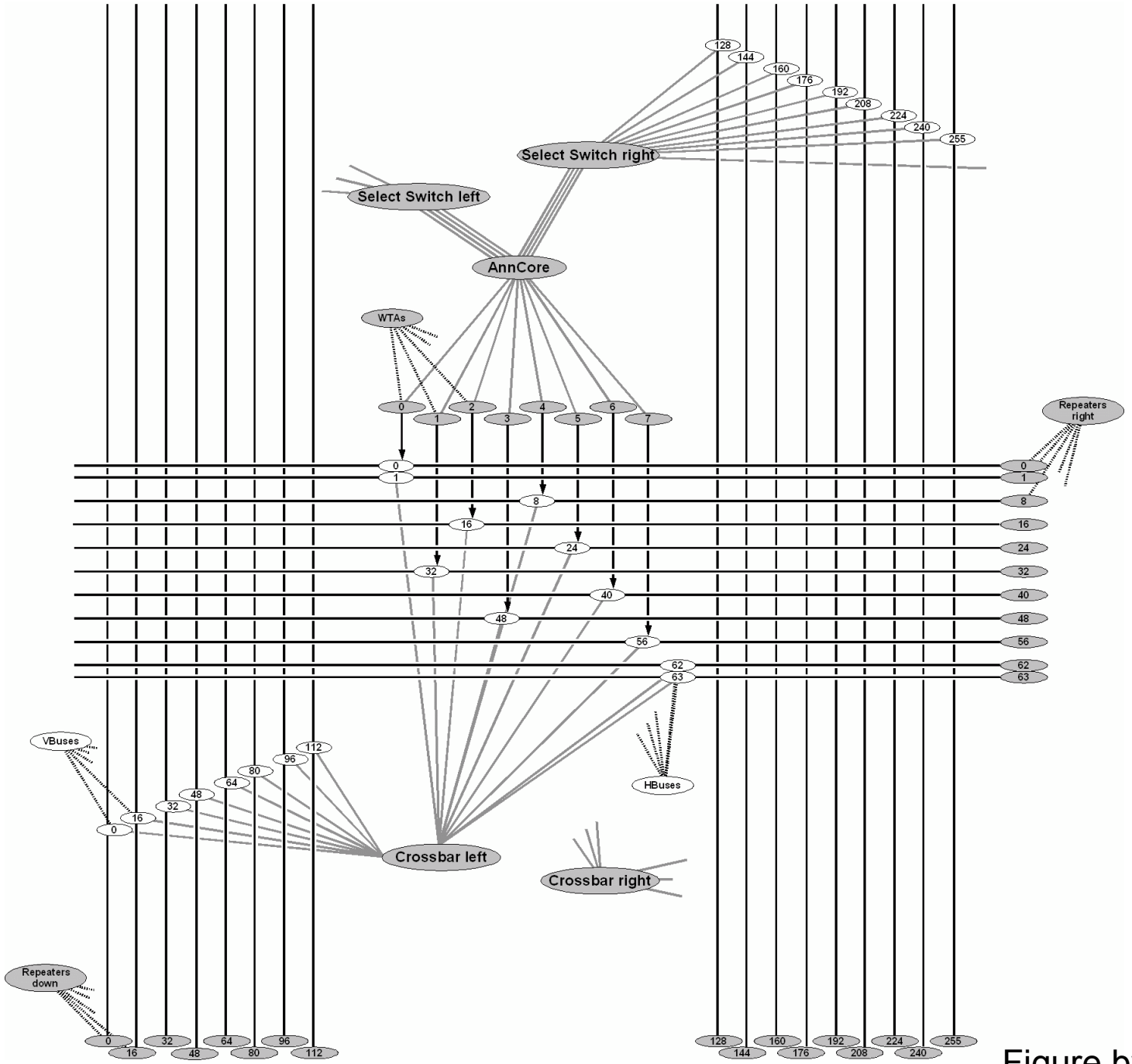
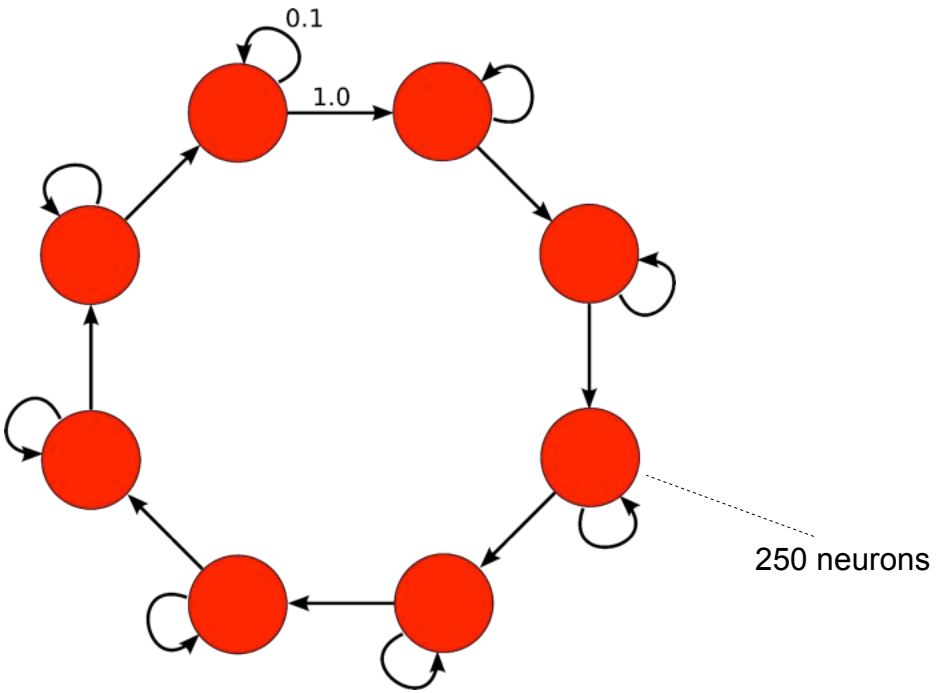


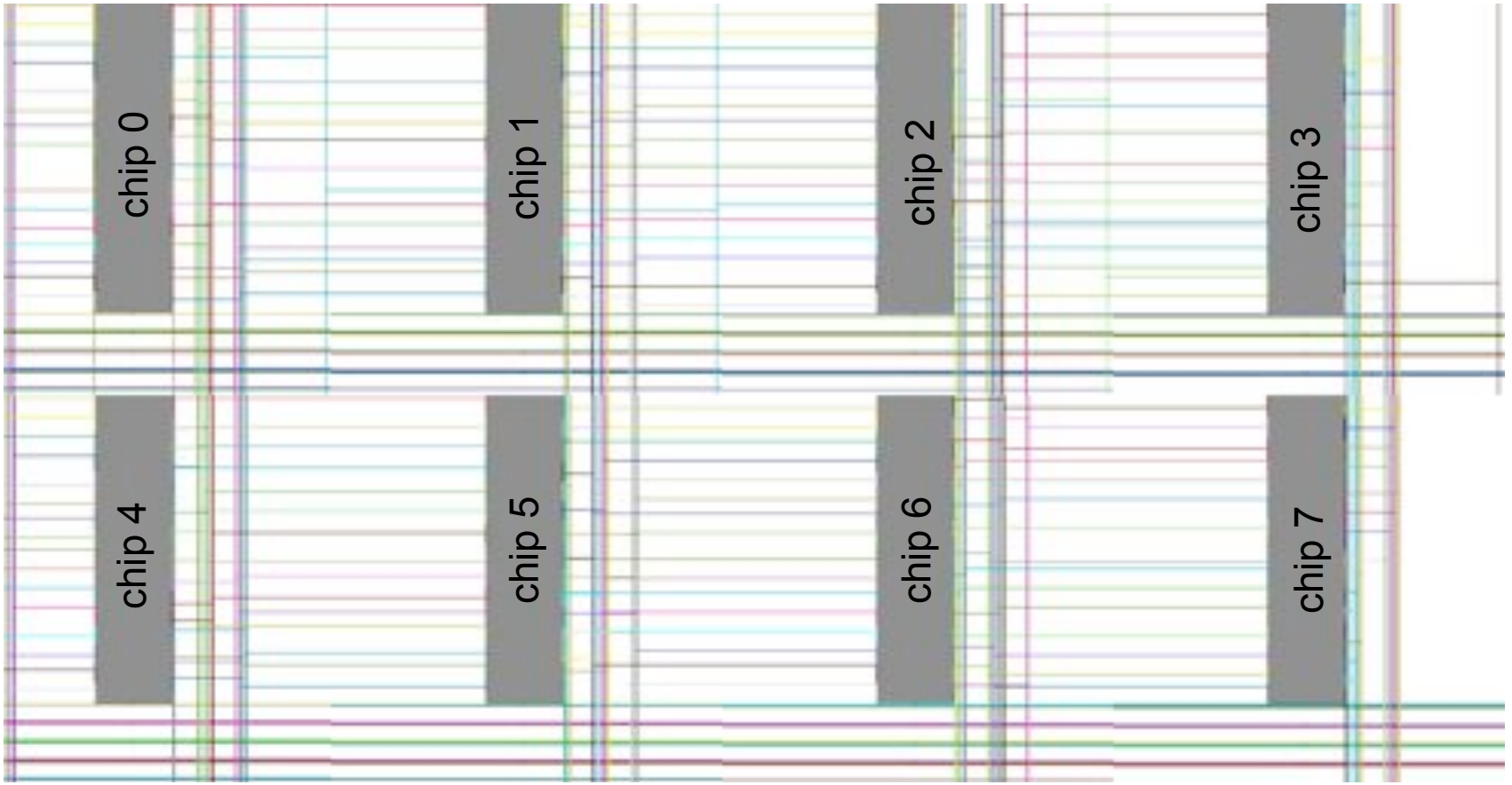
Figure by TUD

A test of the placing and mapping steps



- dummy network example
- populations connected in a ring structure (e.g. synfire chain)

Routing after random placing



intelligent mapping before routing

NForceCluster Algorithm

- clusters neurons according to similarity criterions
- e.g. identical input sources

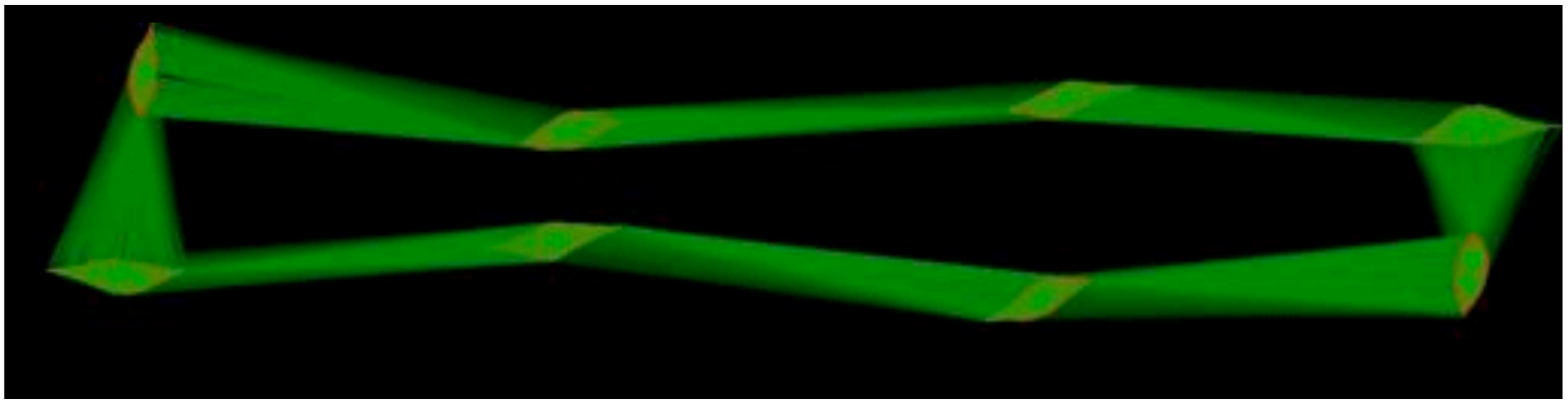
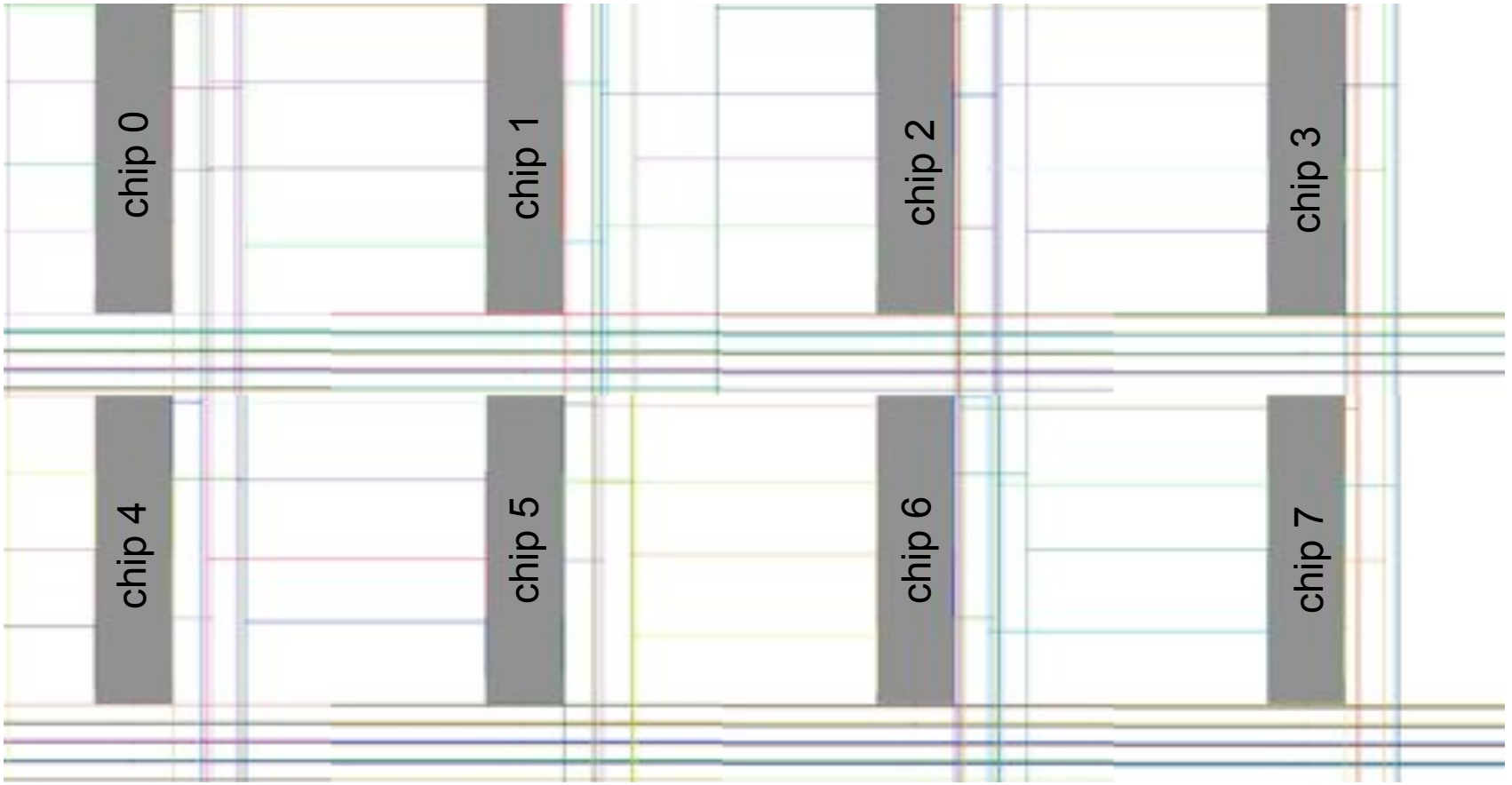


Figure by TUD

routing after *intelligent* placing – NForceCluster Algorithm



Pre- & postmap analysis

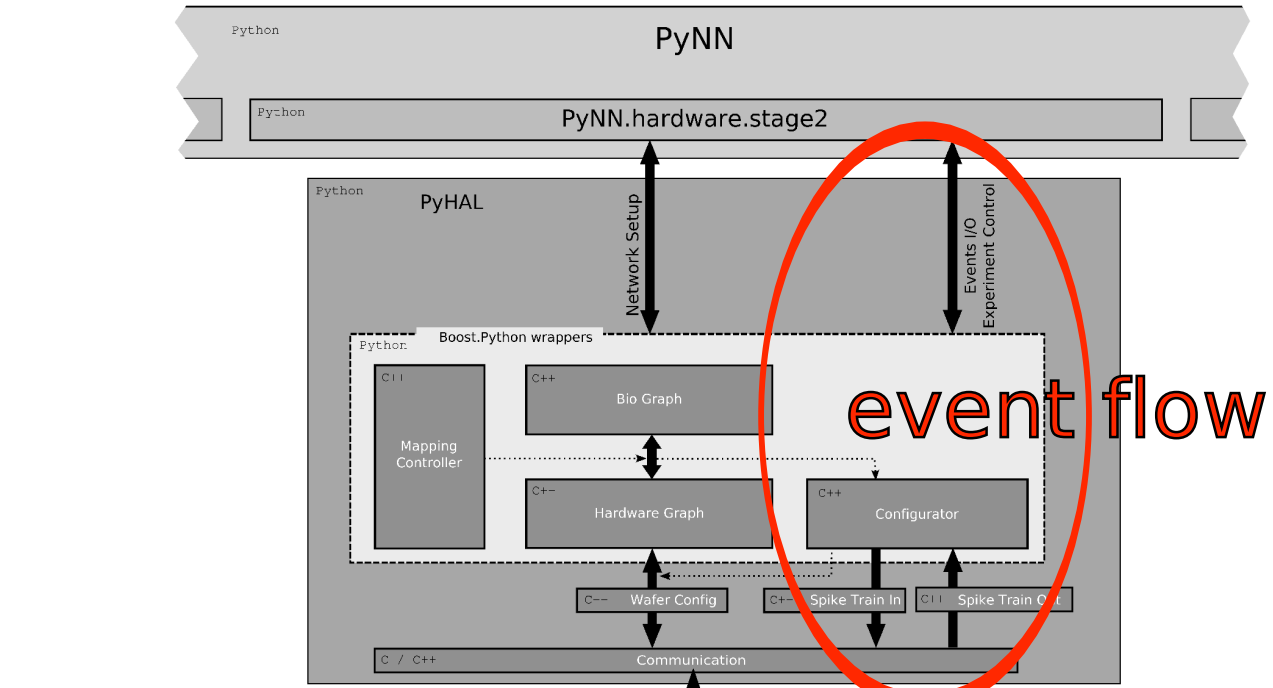
- debugging mapping and routing
- measure for mapping quality

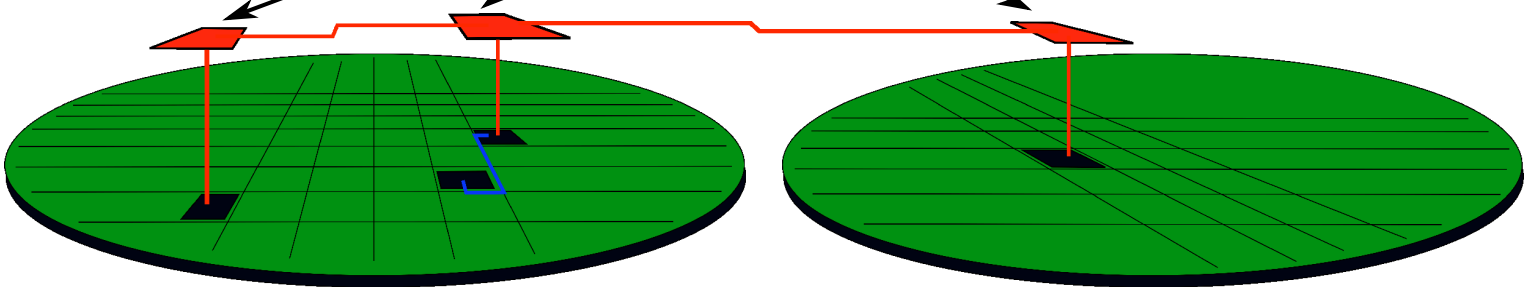
1. create representation of bio model in a flat format (PyNN script, procedural API, every connection, every parameter in a single row)
2. run mapping and routing steps (synaptic loss due to connectivity constrains)
3. create representation of the mapped hardware model
4. differences between scripts due to
   - neuron loss, synaptic loss (limited bandwidth and routing resources)
   - parameter distortion (limited parameter range, flexibility, delays)

Runtime control – event flow

Operating Interface

Mapping Software





Digital (Layer2)

Analog (Layer1)

- after setup, the *Configurator* controls event flow
  - input- and output spikes
  - runtime chip control (recording, read-out of parameters, setting parameters)

Incremental configuration in PyNN

- main problem: clustering of parameter space takes a lot of time
- placing and routing steps are slow ($1\,\mathrm{min}$ – $1\,\mathrm{h}$, depends on model)
- but parameter changes can be typically done at zero cost $\Rightarrow$ parameter sweeps still possible
- some topological changes do not require rerunning the mapping steps
- we will need support for *incremental* configuration: in our interface: `setup()`, `connect()`, `run()`, `reconnect()`, `run()`, ...
- PyNN support for `disconnect()` , setting and resetting parameters of high-level objects?

Backend-specific pre-processing of data

- multi-wafer systems will generate too many spikes to analyze (or store) in real-time
- even a single wafer can't be fully recorded
  - on-wafer bandwidth is in the range of several TB/s
  - off-wafer (FPGA) bandwidth: $\sim 50\,\mathrm{GB/s}$
  - host-FPGA bandwidth: $2\,\mathrm{GB/s}$
- reduce data rates – FPGA can do a lot of preprocessing
- statistical and averaged measures evaluated online
  - avg. firing rates, isi, measures for synchrony, irreg., etc.
  - low-pass filtering of analog (e.g. sub-threshhold membrane trace) data, avg. weight over time

Conclusion: Open issues

- hardware specific unit tests
- fragile make flow needs to be replaced by a better build system
  - we have to link to the GraphModel:
    C++ and dependencies on other libraries
- create dummy PyNN module for virtual hardware (using the FACETS Demonstrator simulation code)
- lots of control software missing
- waiting for the real wafer system