SpiNNaker2 Tutorial: Beyond Neural Simulation

NICE 2025, 28/03/2025, Heidelberg

Bernhard Vogginger, Florian Feiler, Wadjih Bencheikh,
Gabriel Béna, Mahmoud Akl

TECHNISCHE UNIVERSITÄT DRESDEN

SpiNNcloud

Sponsored by

Gefördert durch:
Bundesministerium für Wirtschaft und Klimaschutz
aufgrund eines Beschlusses des Deutschen Bundestages

ESCADE

PRIMI
Performance in Robots Interaction via Mental Imagery

Funded by the European Union

ScaDS.AI DRESDEN LEIPZIG

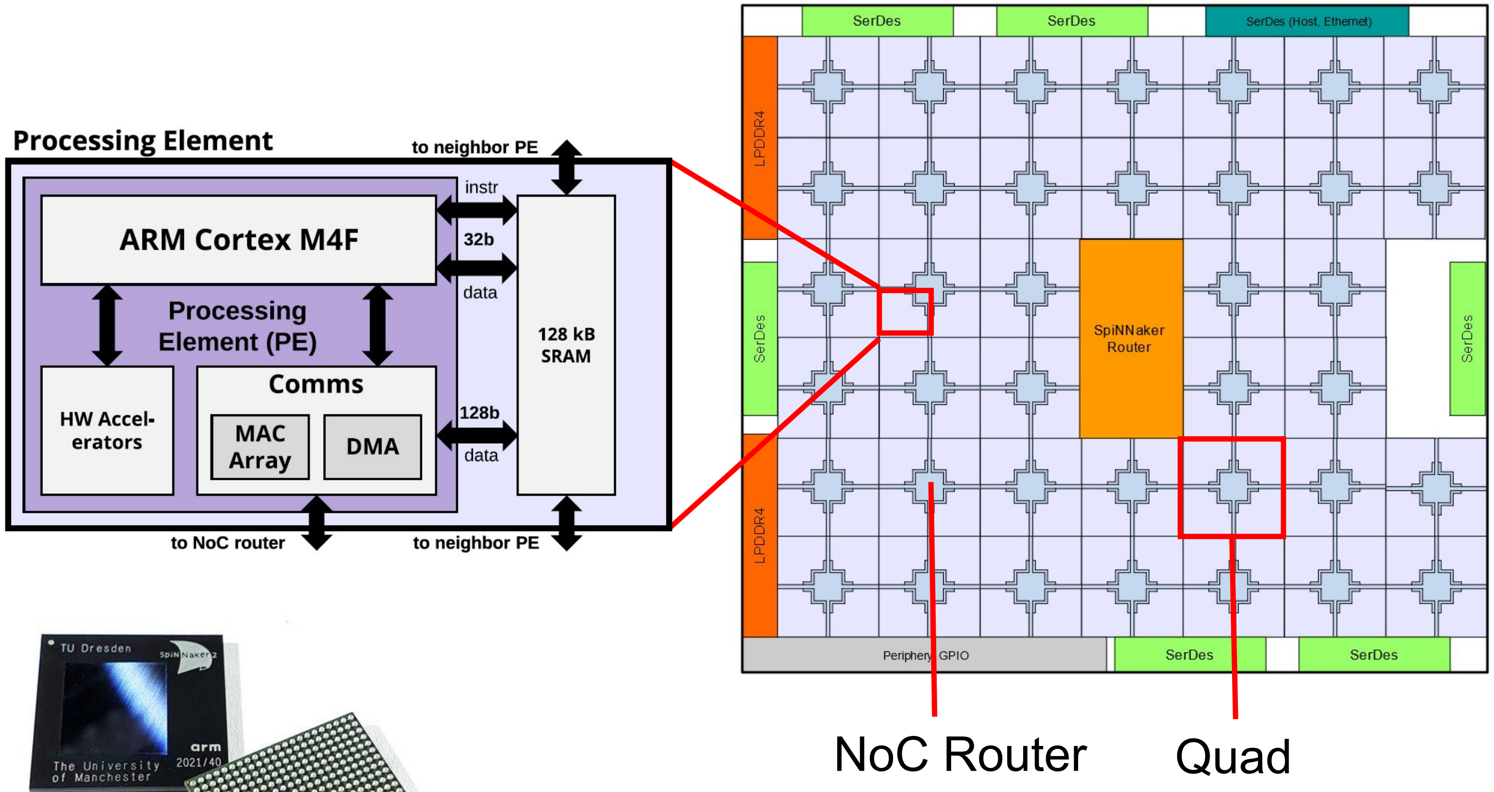
GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

SACHSEN
Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des vom Sächsischen Landtag beschlossenen Haushaltes.

Overview

- Introduction to SpiNNaker2 HW & SW
- Instruction on Access to SpiNNaker2 JupyterHub
- Demos:
  1. Py-spinnaker2 intro
  2. Brunel E/I network
  3. QUBO (Optimization)
  4. NIR for Inference of Spiking CNN for N-MNIST
  5. Event-Prop (Live demo)
- Discussion & Outlook

SpiNNaker2 Chip



102 mm², GF 22FDX technology

SpiNNaker2 Chip

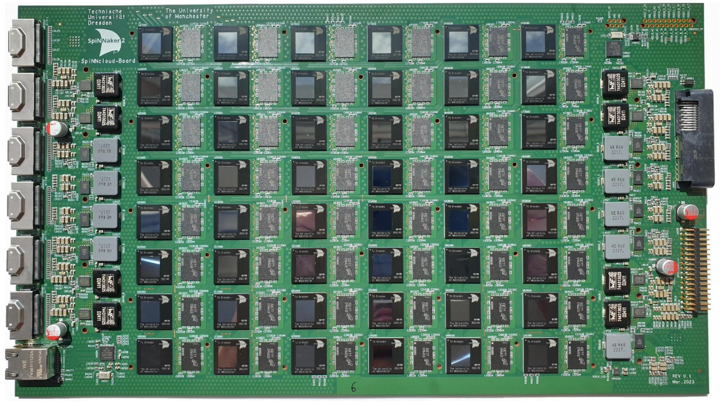
- 152 processing elements
  - Arm Cortex-M4F processor
  - 128 KB SRAM
  - DNN accelerator (MAC array)
  - exp, log, RNG accelerator
- SpiNNaker router
- 6 chip-to-chip links
- LPDDR4 interface

Key Features

- scalable event-based communication
- brain-inspired “neuromorphic” computing using ARM & accelerators
- Efficient DNN processing using MAC array

The „SpiNNcloud“ at TU Dresden

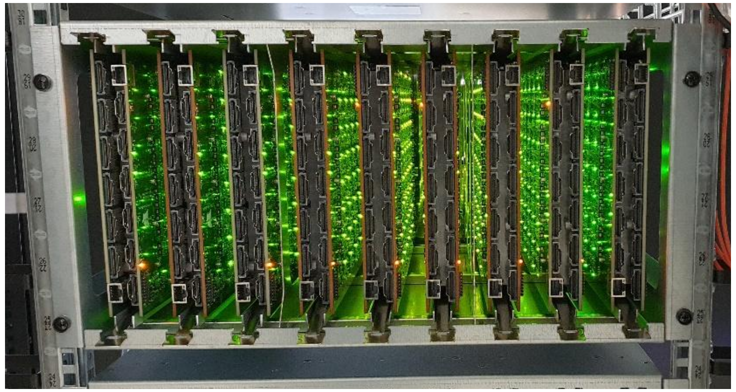
SpiNNcloud Board



48 SpiNNaker2 chips with 2 GB DRAM each

Card frame

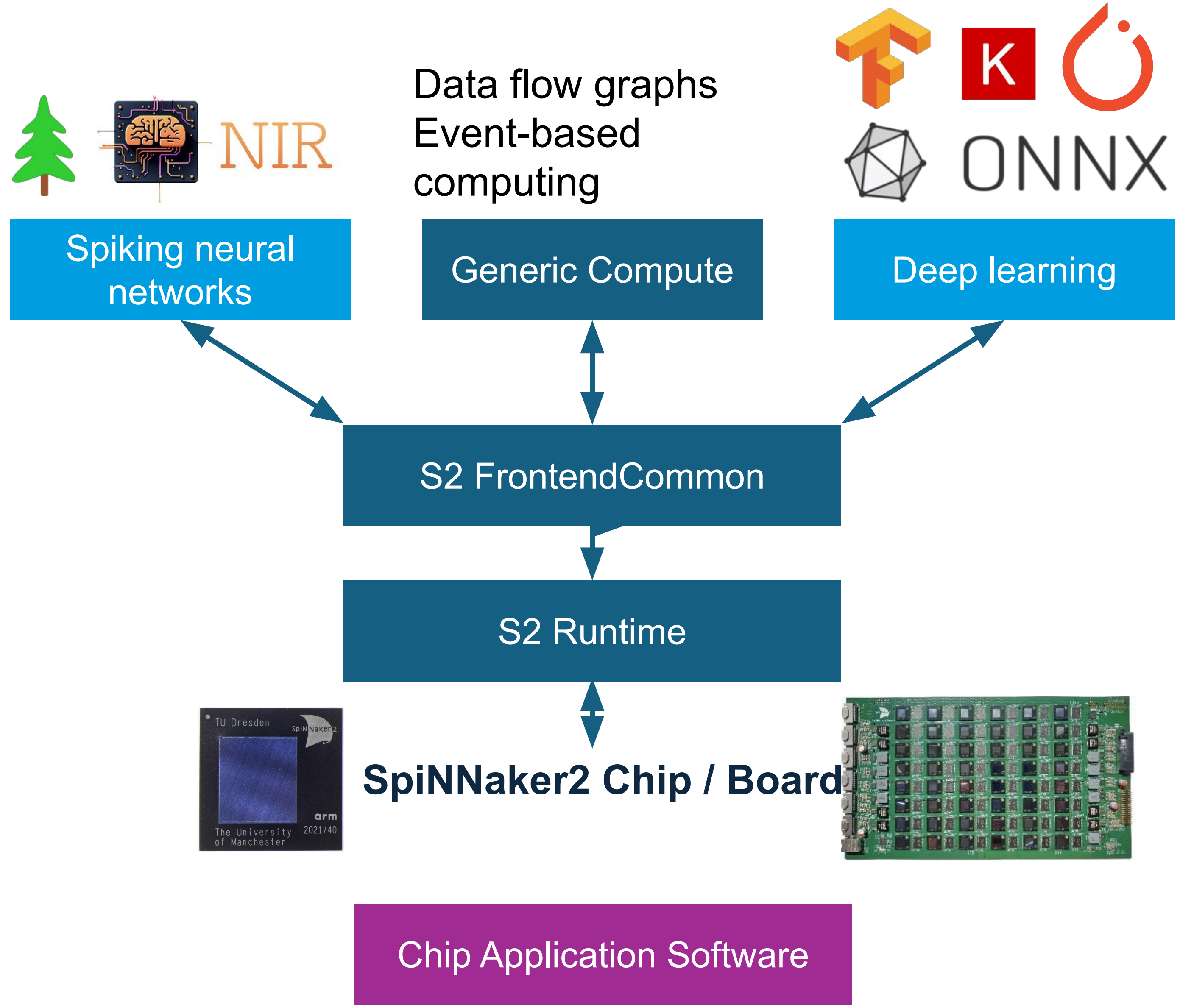
With 18 boards and water cooling



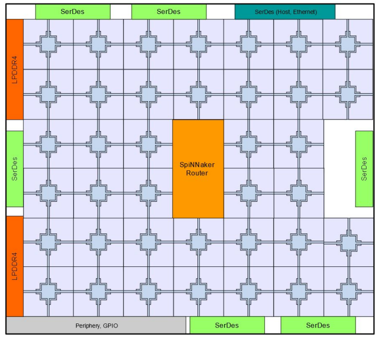
5 Million Core Machine

8 racks with 5 card frames

SpiNNaker2 Software Stack

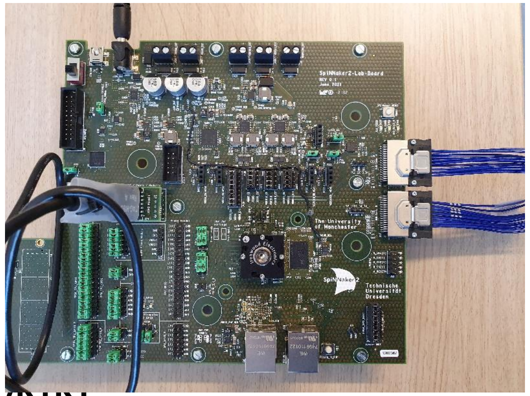


py-spinnaker2: SNN & hybrid SNN/DNN



Light-weight Python interface for SNNs or hybrid networks

- Code: https://gitlab.com/spinnaker2/py-spinnaker2
- Docs: https://spinnaker2.gitlab.io/py-spinnaker2/



User interface:

- Define SNN in terms of Populations and Projections with API similar to PyNN
- Various neuron models and spike sources
- Recording of spikes and voltages
- If no chip available, use brian2 backend

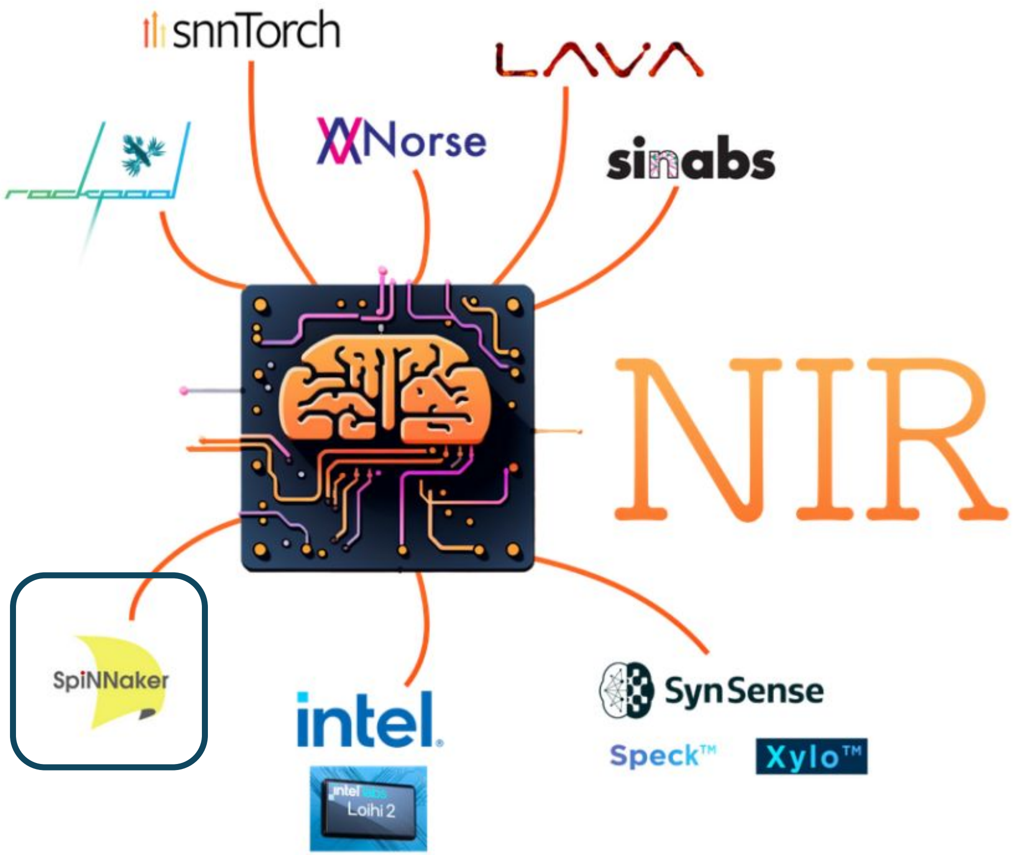
BRIAN

```
from spinnaker2 import snn, hardware

neuron_params = {
        "threshold":1.,
        "alpha_decay":0.9,
        }

stim = snn.Population(
        size=10,
        neuron_model="spike_list",
        params={0:[1,2,3], 5:[20,30]},
        name="stim")

pop1 = snn.Population(
        size=20,
        neuron_model="lif",
        params=neuron_params,
        name="pop1")
```

Used for all hands-on examples

Neuromorphic intermediate representation (NIR)

- Support to run deep SNN on SpiNNaker2:
  Train in any SNN training framework and convert via NIR to py-spinnaker2
- NIR was developed together with the neuromorphic community (started in Telluride 2023)
  Serves as an intermediate representation format similar to ONNX for DNN
  Goal: greater interoperability
  Supports 7 software frameworks (e.g. snnTorch, lava, Norse, NengoDL) and 4 hardware systems
  https://neuroir.org/



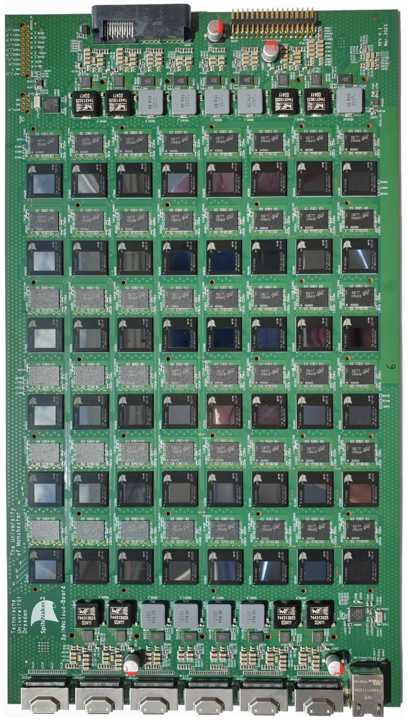
See demo “04-nir”

PyNN: Migration of sPyNNaker

Large-scale SNN simulation using PyNN

- Will support large set of PyNN neuron models and also pasticity mechanisms
- Will re-use large parts from SpiNNaker1 stack (pyNN.sPyNNaker)
- Current work:
  - Adaption of low-level software: scamp2, sark2, SpiNNman2
- Next steps of the migration:
  - Adapt PACMAN for configuration of new SpiNNaker2 router
  - Port first neuron models with static synapses
- Availability: H2/2025 for 48-node boards

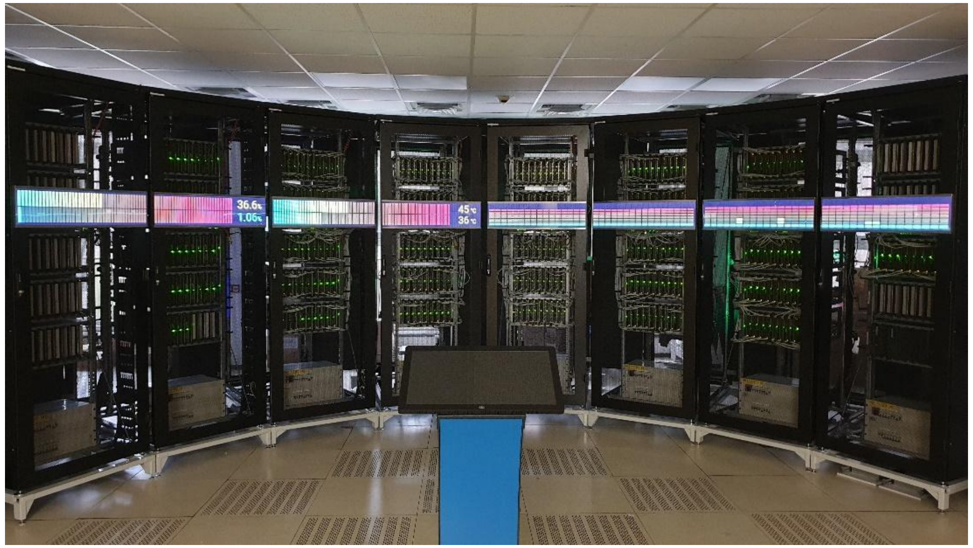
SpiNNcloud Board



48 SpiNNaker2 chips
with 2 GB DRAM each

Hands-on Tutorial

- Jupyterhub (Accessible via TUD-VPN): http://jupyter.spinncloud.et.tu-dresden.de/
- Accounts are temporary and just for this workshop
- Access to one 48-node board per user
- Make sure to set the IP address of your board in the code



SpiNNaker2 Documentation & Resources

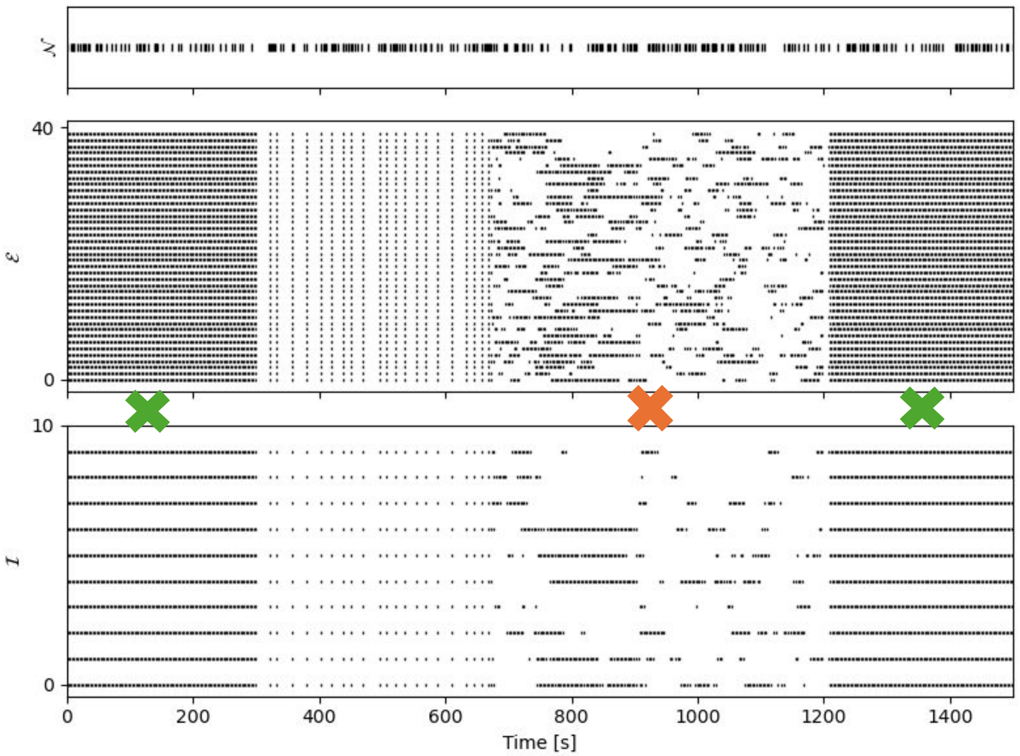
- General SpiNNaker2 Documentation: https://spinnaker2.gitlab.io/
- SNN API: https://spinnaker2.gitlab.io/py-spinnaker2/snn-api/
- Neuron Models: https://spinnaker2.gitlab.io/py-spinnaker2/neuron-models/
- Git repository with Jupyter Notebooks: https://gitlab.com/spinnaker2/tutorial-nice-2025/

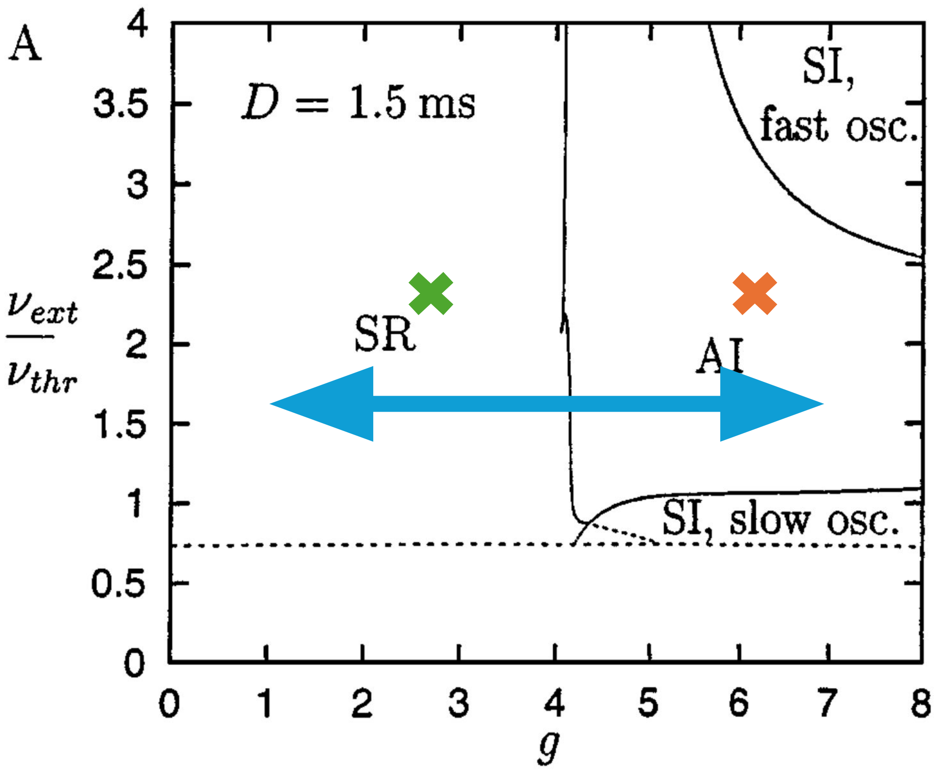
SpiNNcloud

[NEST Source]

Brunel Network

- EI network, all-to-all sparse connectivity incl. autapses, Poisson input to all neurons (1:N)
- N = NE + NI + 1 = scale * (4 + 1) + 1
- Aim: Balanced network of LIF neurons w/ alpha synapses
  - Stable regimes of different behaviour, synchronous vs. asynchronous, regular vs. irregular,…





SpiNNcloud

Quadratic Unconstrained Binary Optimization (QUBO)

- Numerous combinatorial optimization problems, such as Graph/Number Partitioning, Maxcut, SAT, Graph Coloring, and TSP, can be efficiently represented using **QUBO**'s streamlined mathematical expressions

- QUBO objective function:

Type of problem: depending on the type of the problem, the objective can be to minimize or maximize this function

Q-matrix: usually symmetric or upper-triangular. Coefficients and shape depend on the optimization problem

$$\text{minimize/maximize } y = x^t Q x$$

Vector of binary decision variables: Represents a candidate solution

- QUBO problems are solved using **quantum annealing**, **simulated annealing**, and **genetic algorithms**

QUBO and SNNs

- Natural correspondence between **binary decision variables** and many **neuromorphic primitives** such as **binary neurons**, **spiking neurons**, etc.

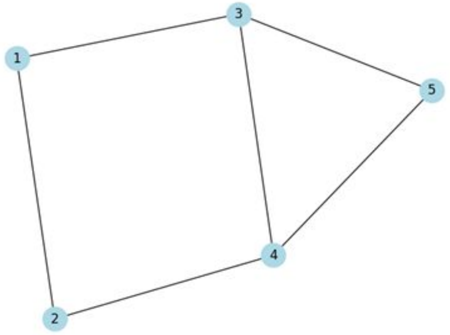
- Linear and quadratic weights correspond to biases and synapses

- The Q-matrix describes the construction of a recurrent network of spiking neurons. The resulting **network dynamics** will attempt to **converge** to a solution

- Random excitation of neurons injects energy into the system. Possible sources of randomness on SpiNNaker: **spike trains**, **input weights**, **synaptic delays**, **noise on membrane potentials**

QUBO: Spinnaker2 pipeline

1. Mapping Problems to QUBO



$$Q = \begin{bmatrix} -2 & 1 & 1 & 0 & 0 \\ 1 & -2 & 0 & 1 & 0 \\ 1 & 0 & -3 & 1 & 1 \\ 0 & 1 & 1 & -3 & 1 \\ 0 & 0 & 1 & 1 & -2 \end{bmatrix}$$

Glover, Fred, Gary Kochenberger, and Yu Du. "Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models." 4or 17 (2019): 335-371.

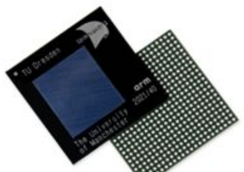
2. Build SNN based on Q-matrix

Graph nodes → neurons
Graph edges → synaptic weights

Specialized **neurons** with similar **dynamics** as simulated annealing, which **evolve towards energy minimum** for optimization

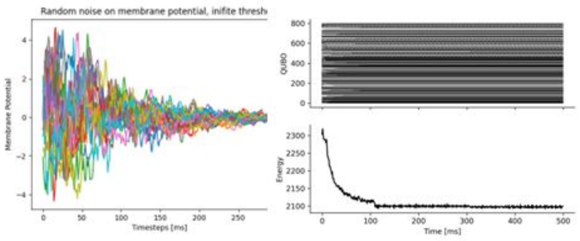
Spikes or membrane potentials can be used to encode the optimal QUBO solution

3. Run on SpiNNaker & evaluate



At each time step we get a new **solution candidate** based on the **output spiking** activity.

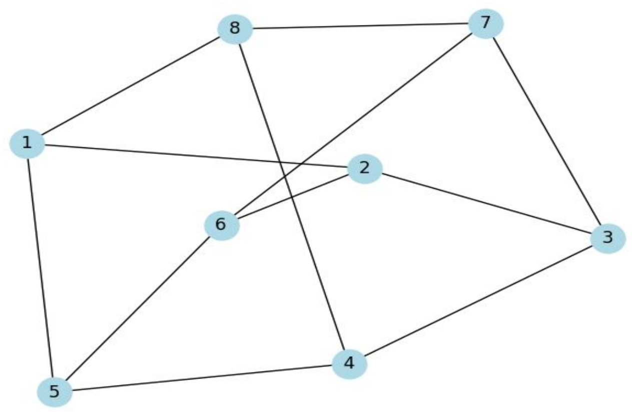
Example of **decaying noise** injected into the system and **convergence** to **solution**



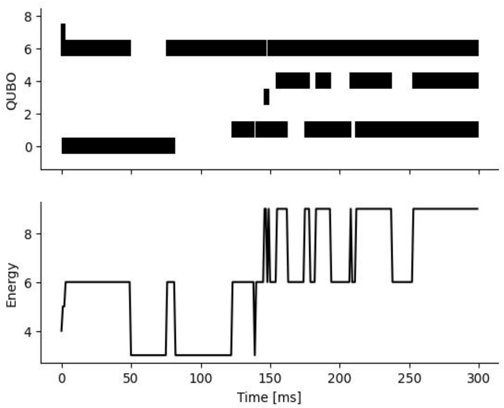
SpiNNcloud

QUBO: Maxcut example

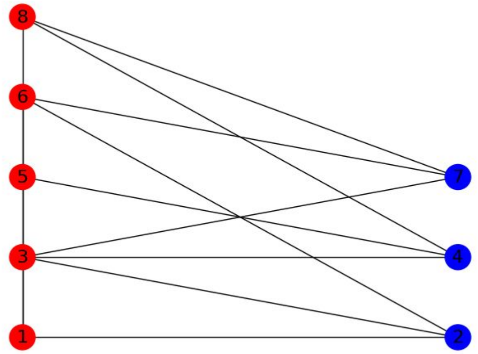
Graph (8 nodes)



SpiNNaker SNN simulation



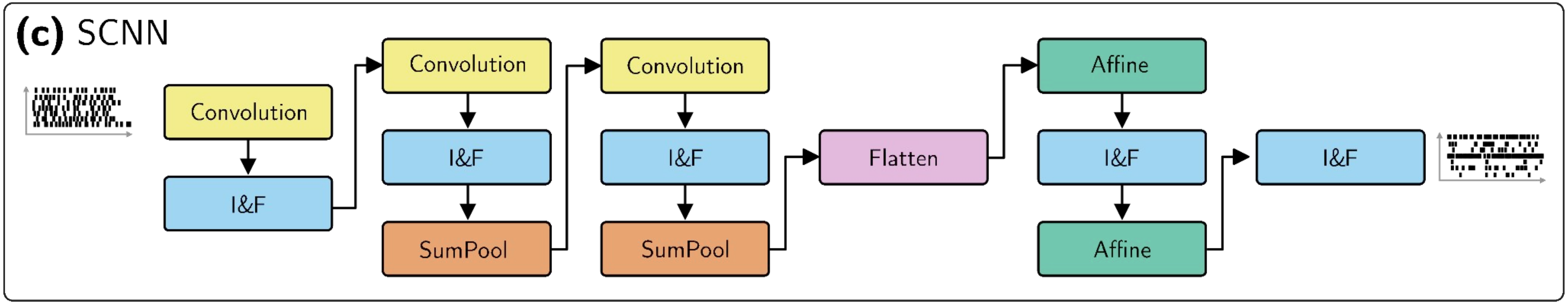
Partitioned Graph (maximum energy)



SpiNNcloud

NIR → SpiNNaker2

Demo: Spiking CNN for N-MNIST



Results from paper:

	Simulator						Both	Hardware		
	Nengo	**Norse**	**Rockpool**	**Sinabs**	**snnTorch**	**Spyx**	**Lava**†	**Speck**	**SpiNNaker2**	**Xylo**
SCNN	98.1%	98.1%	N/A	<u>98.5%</u>	97.9%	97.1%	98.2%	95.4%	98.2%	N/A

SCNN in sinabs and py-spinnaker2

```python
model = nn.Sequential(
    nn.Conv2d(2, 20, 5, 1, bias=False),
    backend.IAFSqueeze(shape=[batch_size, 20, 30, 30]),
    nn.AvgPool2d(2, 2),
    nn.Conv2d(20, 32, 5, 1, bias=False),
    backend.IAFSqueeze(shape=[batch_size, 32, 11, 11]),
    nn.AvgPool2d(2, 2),
    nn.Conv2d(32, 128, 3, 1, bias=False),
    backend.IAFSqueeze(shape=[batch_size, 128, 3, 3]),
    nn.AvgPool2d(2, 2),
    nn.Flatten(),
    nn.Linear(128, 500, bias=False),
    backend.IAFSqueeze(shape=[batch_size, 500]),
    nn.Linear(500, 10, bias=False),
)
```

```python
from spinnaker2 import snn

# Populations
pop_in = snn.Population(2312, neuron_model="spike_list", params=input_spikes)
pop_1 = snn.Population(4096, neuron_model="lif_conv2d", params=params_p1)
pop_2 = snn.Population(4096, neuron_model="lif_conv2d", params=params_p2)
pop_3 = snn.Population(512, neuron_model="lif_conv2d", params=params_p3)
pop_4 = snn.Population(256, neuron_model="lif_no_delay", params=params_p4)
pop_out = snn.Population(10, neuron_model="lif_no_delay", params=params_out)

# Projections
proj_1 = snn.Conv2dProjection(pop_in, pop_1, weight_1, params_proj_1)
proj_2 = snn.Conv2dProjection(pop_1, pop_2, weight_2, params_proj_2)
proj_3 = snn.Conv2dProjection(pop_2, pop_3, weight_3, params_proj_3)
proj_4 = snn.Projection(pop_3, pop_4, conn_list_proj_4)
proj_5 = snn.Projection(pop_4, pop_out, conn_list_proj_5)

# Network
net = snn.Network("SCNN")
net.add(pop_in, pop_1, pop_2, pop_3, pop_4, pop_out,
        proj_1, proj_2, proj_3, proj_4, proj_5)
```

Outlook & Discussion

- 5M core machine in Dresden – next 6 months:
  - Applications on single 48-chip boards
  - Set up SLURM
  - Set up remote access for academia