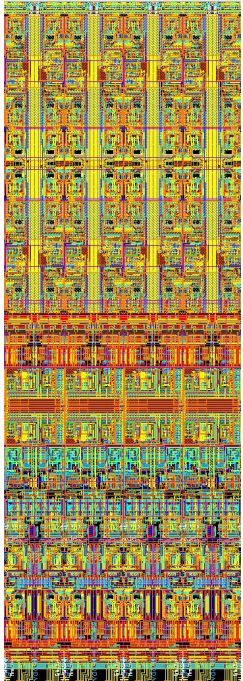
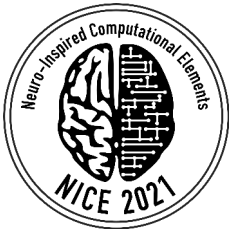


BrainScaleS

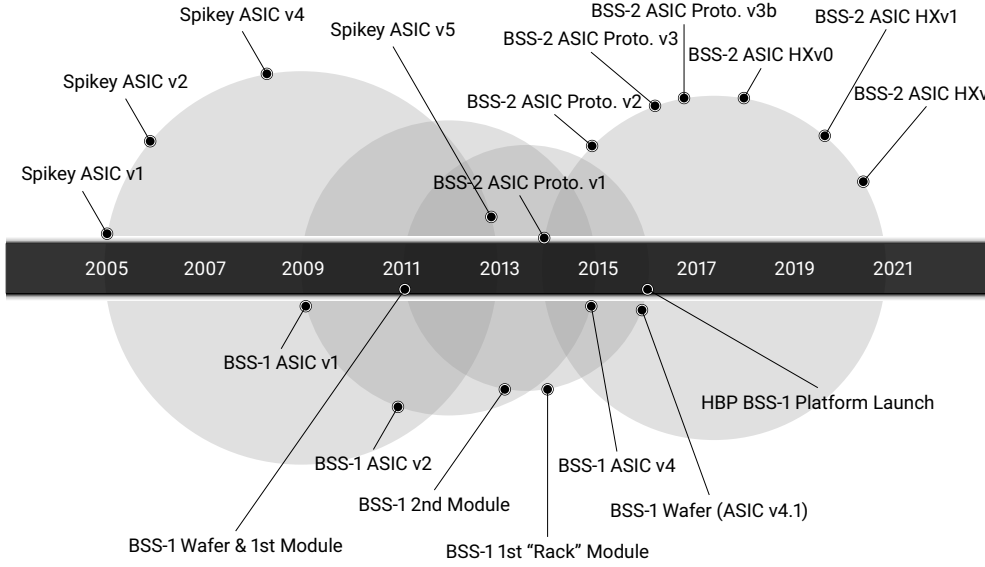
Development Methodologies and Operating System

Eric Müller

on behalf of Electronic Vision(s)

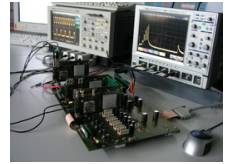


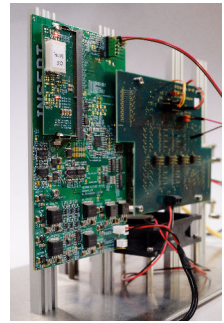
BrainScaleS — Hardware Timeline

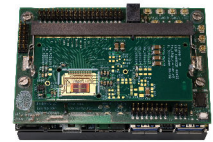


BrainScaleS Architectures & Systems — Software Timeline

- Spikey
  - targets mobile single-chip systems, some multi-chip capabilities
  - Active 05/2006 – 10/2017
- BrainScaleS-1
  - targets wafer-scale systems
  - Active since 09/2009
- BrainScaleS-2
  - targets mobile & wafer-scale systems
  - Active since 03/2015

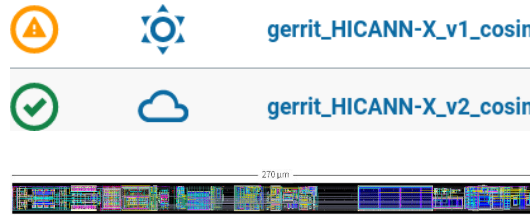






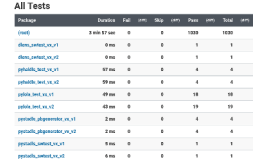
Overview: Software Tasks[0]

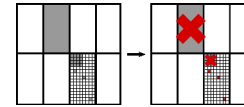
- Hardware/software co-development *(not covered)*



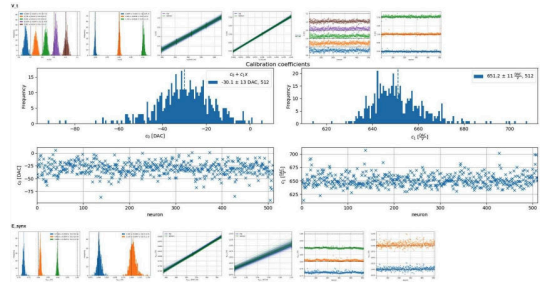
[0] work by O. J. Breitwieser, S. Schmitt, C. Mauch, P. Spilger, Y. Stradmann, H. Schmidt, J. Montes, A. Emmel, M. Czierlinski, J. Kaiser, S. Billaudelle, F. Ebert, M. Güttler, J. Ilmberger, A. Leibfried, J. Weis and many more.

Overview: Software Tasks[0]





- Hardware/software co-development
- Hardware commissioning, testing, verification and characterization *(not covered)*



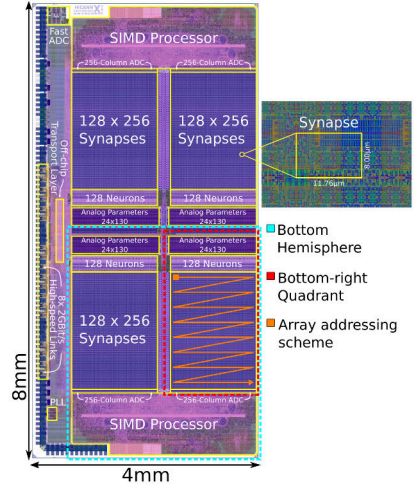
[0] work by O. J. Breitwieser, S. Schmitt, C. Mauch, P. Spilger, Y. Stradmann, H. Schmidt, J. Montes, A. Emmel, M. Czierlinski, J. Kaiser, S. Billaudelle, F. Ebert, M. Güttler, J. Ilmberger, A. Leibfried, J. Weis and many more.

Overview: Software Tasks[0]

- Hardware/software co-development
- Hardware commissioning, testing, verification and characterization
- Hardware usage (conducting experiments)
  - **Low abstraction level**

[0] work by O. J. Breitwieser, S. Schmitt, C. Mauch, P. Spilger, Y. Stradmann, H. Schmidt, J. Montes, A. Emmel, M. Czierlinski, J. Kaiser, S. Billaudelle, F. Ebert, M. Güttler, J. Ilmberger, A. Leibfried, J. Weis and many more.

Task: Low abstraction level



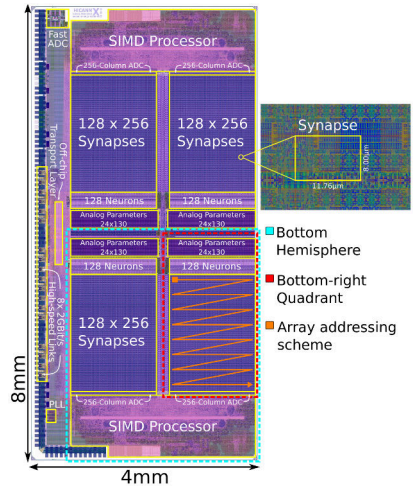
Task: Low abstraction level[1]

```python
def configure_synapses(*args):
    """
    Configure routing crossbar, PADI bus, synapse drivers,
    and parts of the synapse array.
    """
    fisch_builder = fisch.PlaybackProgramBuilder()
    fisch_builder.write(anncore_center_ba, fisch.Omnibus(0xffff))
    config_builder.merge_back(fisch_builder)

    # synapse array
    correlation_switch_quad = haldls.ColumnCorrelationQuad()
    switch = correlation_switch_quad.ColumnCorrelationSwitch()
    switch.enable_internal_causal = True
    switch.enable_internal_acausal = True
    for s in range(4):
        correlation_switch_quad.set_switch(s, switch)

    for sq in iter_all(halco.ColumnCorrelationQuadOnDLS):
        config_builder.write(sq, correlation_switch_quad,
                             haldls.Backend.Omnibus)

    current_switch_quad = haldls.ColumnCurrentQuad()
    switch = current_switch_quad.ColumnCurrentSwitch()
    switch.enable_synaptic_current_excitatory = True
    switch.enable_synaptic_current_inhibitory = True
    for s in range(4):
        current_switch_quad.set_switch(s, switch)
```



[1] E. Müller, S. Schmitt, C. Mauch, et al. “The Operating System of the Neuromorphic BrainScaleS-1 System”. In: arXiv preprint (Mar. 2020). arXiv: 2003.13749 [cs.NE], E. Müller, C. Mauch, P. Spilger, O. J. Breitwieser, et al. “Extending BrainScaleS OS for BrainScaleS-2”. In: arXiv preprint (Mar. 2020). arXiv: 2003.13750 [cs.NE].

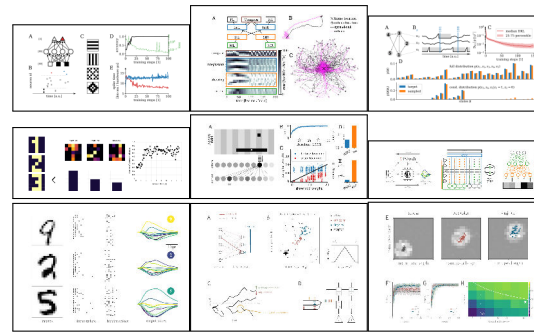
Task: Low abstraction level

```
def configure_synapses(*args):
    """
    Configure routing crossbar, PADI bus, synapse drivers,
    and parts of the synapse array.
    """
    fisch_builder = fisch.PlaybackProgramBuilder()
    fisch_builder.write(anncore_center_ba, fisch.Omnibus(0xffff))
    config_builder.merge_back(fisch_builder)

    # synapse array
    correlation_switch_quad = haldls.ColumnCorrelationQuad()
    switch = correlation_switch_quad.ColumnCorrelationSwitch()
    switch.enable_internal_causal = True
    switch.enable_internal_acausal = True
    for s in range(4):
        correlation_switch_quad.set_switch(s, switch)

    for sq in iter_all(halco.ColumnCorrelationQuadOnDLS):
        config_builder.write(sq, correlation_switch_quad,
                             haldls.Backend.Omnibus)

    current_switch_quad = haldls.ColumnCurrentQuad()
    switch = current_switch_quad.ColumnCurrentSwitch()
    switch.enable_synaptic_current_excitatory = True
    switch.enable_synaptic_current_inhibitory = True
    for s in range(4):
        current_switch_quad.set_switch(s, switch)
```



. . . fits experts!

Overview: Software Tasks[0]

- Hardware/software co-development
- Hardware commissioning, testing, verification and characterization
- Hardware usage (conducting experiments)
  - Low abstraction level
  - **High(er)-level usage**

[0] work by O. J. Breitwieser, S. Schmitt, C. Mauch, P. Spilger, Y. Stradmann, H. Schmidt, J. Montes, A. Emmel, M. Czierlinski, J. Kaiser, S. Billaudelle, F. Ebert, M. Güttler, J. Ilmberger, A. Leibfried, J. Weis and many more.

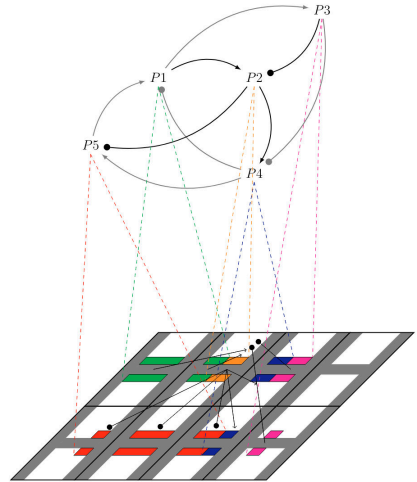
Task: High(er)-level usage

- **User-defined (initial) experiment configuration**[1]
  - Neural network topology
  - Plasticity rules (synapse-dynamical, structural, homeostatic)
    - Code generation for the embedded processors *(not covered)*
    - Structured host-PPU communication *(not covered)*
  - Static and dynamic I/O (closed-loop setups, interaction w/ environment, agents)
  - Parameterization (incl. parameter domain translation)
  - $\rightarrow$ Place & route task
- Hardware resource access
- Experiment run control

[1] E.g. initial `PyNN.brainscales2` implementation by M. Czierlinski, P. Spilger.

Task: High(er)-level usage — Place and Route[2]

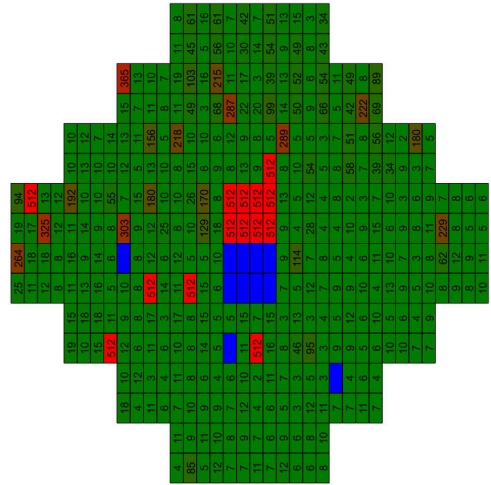
- Map neural network graphs to hardware



[2] F. C. Passenberg. “Improving the BrainScaleS-1 place and route software towards real-world waferscale experiments”. Master thesis. Ruprecht-Karls-Universität Heidelberg, 2019.

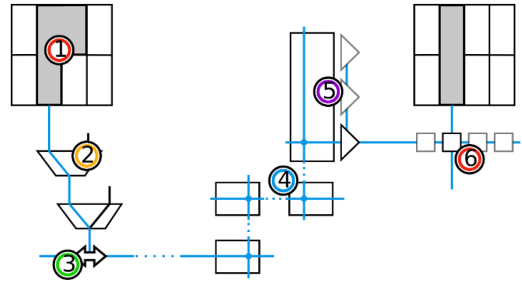
Task: High(er)-level usage — Place and Route

- Map neural network graphs to hardware
- Inhomogeneous substrate



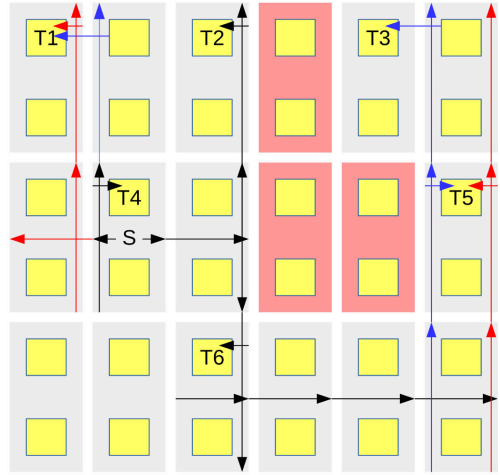
Task: High(er)-level usage — Place and Route

- Map neural network graphs to hardware
- Inhomogeneous substrate
- On-wafer communication on BSS-1 is circuit-switched



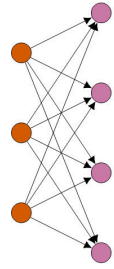
Task: High(er)-level usage — Place and Route

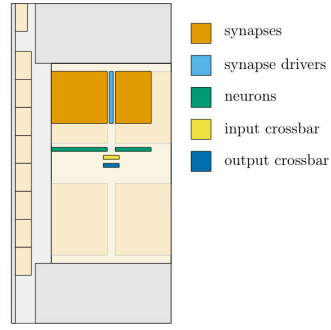
- Map neural network graphs to hardware
- Inhomogeneous substrate
- On-wafer communication on BSS-1 is circuit-switched
- Routing algorithms



Task: High(er)-level usage — Place and Route for ANNs[2]

- BSS-2 supports non-spiking operation (analog inference engine)

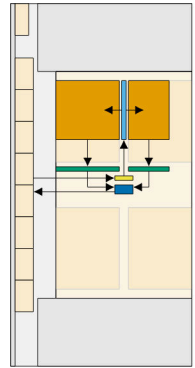


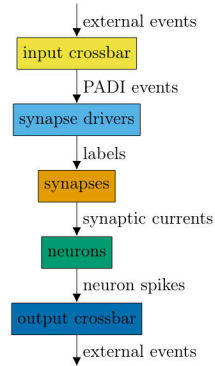


[2] P. Spilger. "From Neural Network Descriptions to Neuromorphic Hardware — A Signal-Flow Graph Compiler Approach". Master thesis. Ruprecht-Karls-Universität Heidelberg, 2021, P. Spilger, E. Müller, et al. "hxtorch: PyTorch for BrainScaleS-2 — Perceptrons on Analog Neuromorphic Hardware". In: Proceedings of the Workshop on IoT, Edge, and Mobile for Embedded Machine Learning (ITEM)/ECML-PKDD 2020. June 2020. arXiv: 2006.13138 [cs.NE].

Task: High(er)-level usage — Place and Route for ANNs

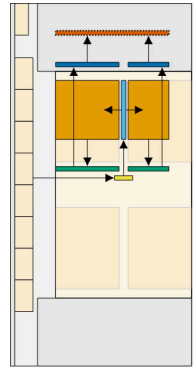
- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware?

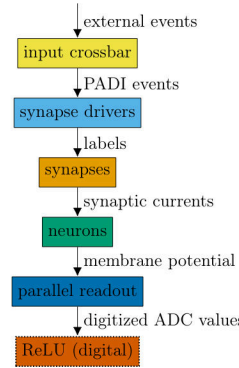




Task: High(er)-level usage — Place and Route for ANNs

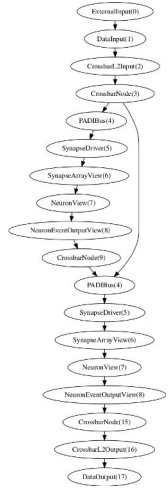
- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware





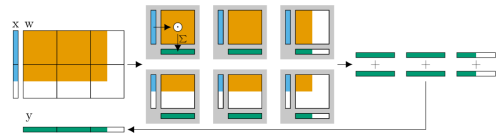
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”



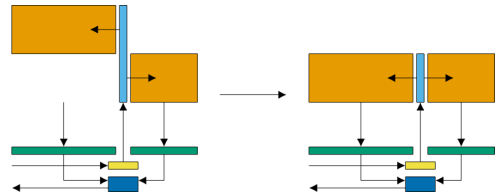
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- partitioning of large problems



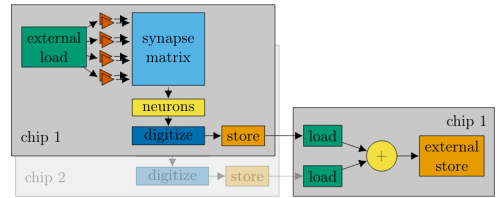
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- Usage optimization & partitioning of large problems



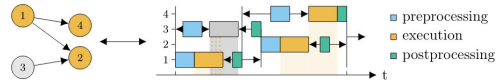
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- Usage optimization & partitioning of large problems
- Execution dependency graph



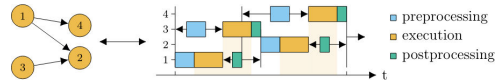
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- Usage optimization & partitioning of large problems
- Execution dependency graph
- Scheduling of dependency graph to hardware resources



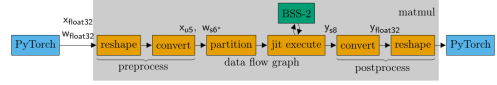
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- Usage optimization & partitioning of large problems
- Execution dependency graph
- Scheduling of dependency graph to hardware resources



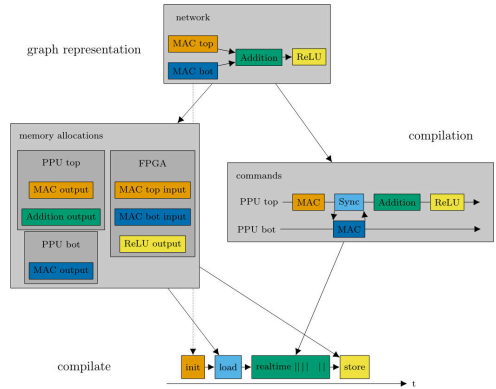
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- Usage optimization & partitioning of large problems
- Execution dependency graph
- Scheduling of dependency graph to hardware resources
- Integration in PyTorch



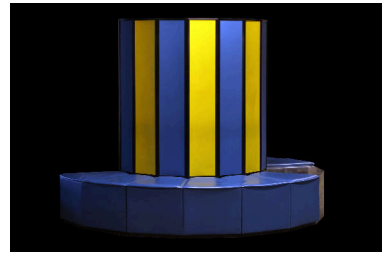
Task: High(er)-level usage — Place and Route for ANNs

- BSS-2 supports non-spiking operation (analog inference engine)
- Data flow graph for BSS hardware
- The BSS-2 “IR”
- Usage optimization & partitioning of large problems
- Execution dependency graph
- Scheduling of dependency graph to hardware resources
- Integration in PyTorch
- Support for standalone inference



Overview: Software Tasks[0]

- Hardware/software co-development
- Hardware commissioning, testing, verification and characterization
- Hardware usage (conducting experiments)
  - Low abstraction level
  - High(er)-level usage
  - **Platform operation**



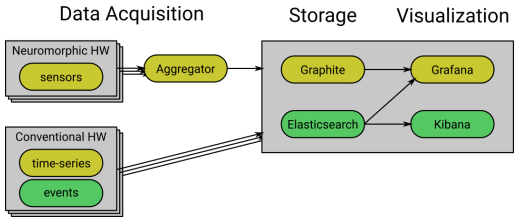
[0] work by O. J. Breitwieser, S. Schmitt, C. Mauch, P. Spilger, Y. Stradmann, H. Schmidt, J. Montes, A. Emmel, M. Czierlinski, J. Kaiser, S. Billaudelle, F. Ebert, M. Güttler, J. Ilmberger, A. Leibfried, J. Weis and many more.

Tasks: Platform Operation

- **Resource monitoring & alerting**

Tasks: Platform Operation — Resource Monitoring & Alerting[2]

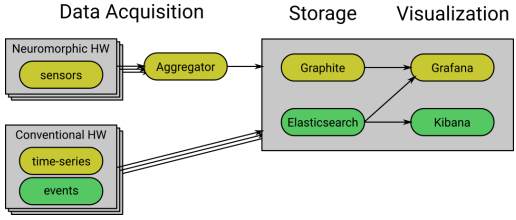
- Sensors (e.g. temp, currents, voltages, . . .)
- Events (e.g. power or experiment state changes, . . .)
- Alerts based on events or violations to operating range

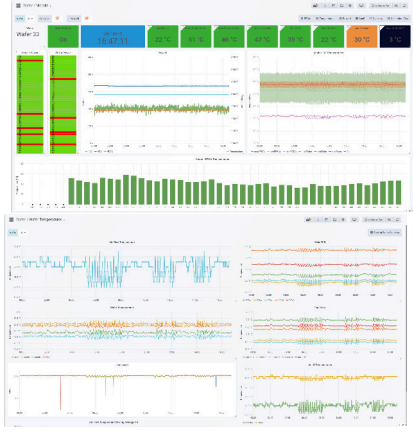


[2] work by M. Güttler, C. Mauch.

Tasks: Platform Operation — Resource Monitoring & Alerting[2]

- Sensors (e.g. temp, currents, voltages, . . .)
- Events (e.g. power or experiment state changes, . . .)
- Alerts based on events or violations to operating range



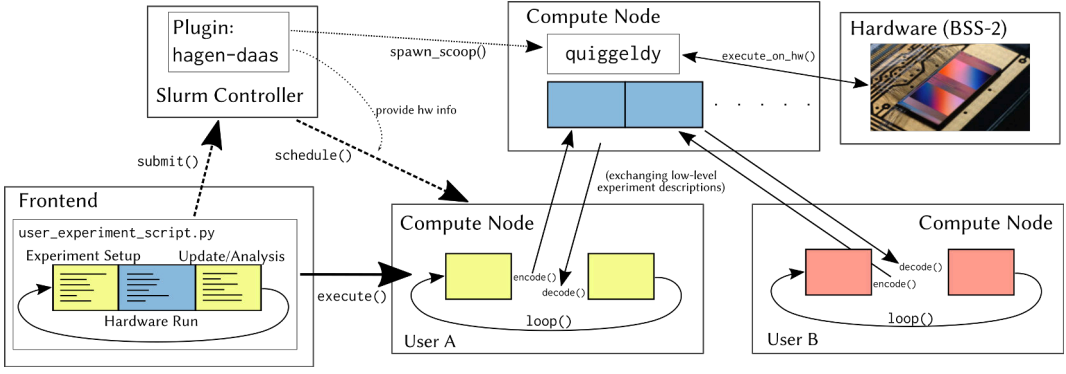


[2] work by M. Güttler, C. Mauch.

Tasks: Platform Operation

- Resource monitoring & alerting
- **Resource management**

Tasks: Platform Operation — Resource Management[3]

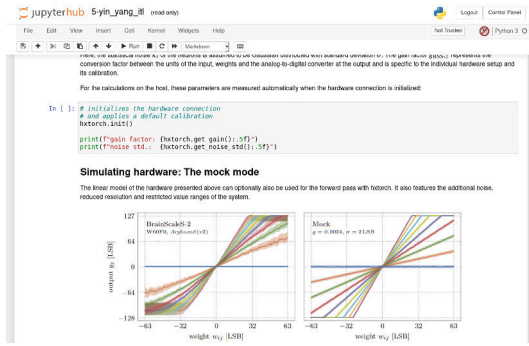


- SLURM + plugin for neuromorphic resource selection
- Custom experiment scheduler
- Used in our hands-on demos: 27k (32 active users) + 41k (42 active users) experiments scheduled to 8 single-chip setups

[3] work by O. J. Breitwieser, C. Mauch.

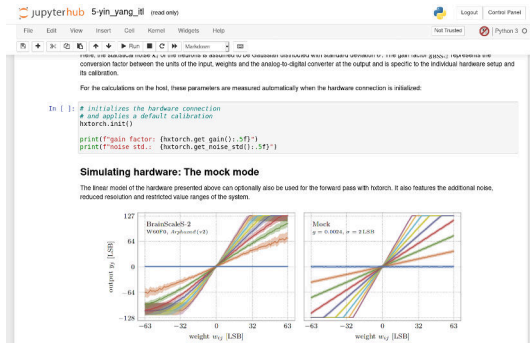
Tasks: Platform Operation

- Resource monitoring & alerting
- Resource management
- Neuromorphic hardware as a service

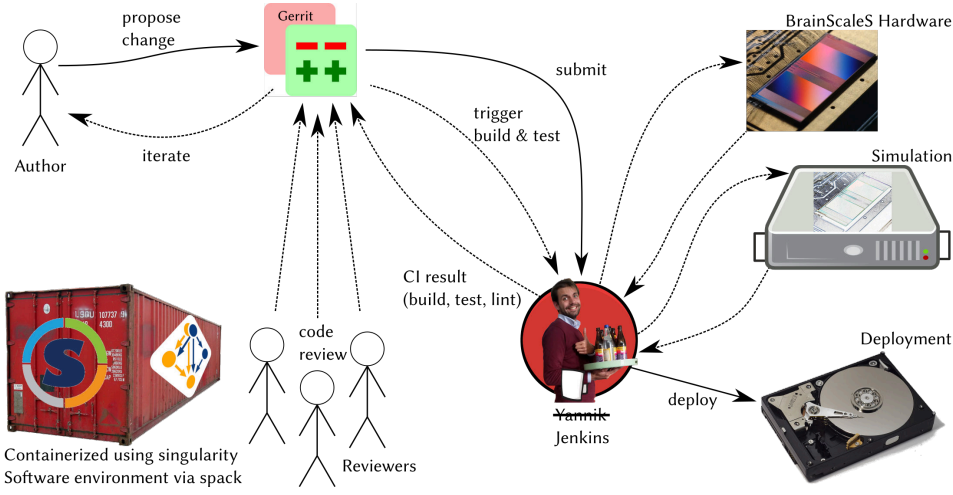


Tasks: Platform Operation

- Resource monitoring & alerting
- Resource management
- Neuromorphic hardware as a service
  - Reproducibility
  - Sustainability



Development Methodologies: CR, CI, CD, Co-Dev. & Containers[4]



[4] work by Y. Stradmann, O. J. Breitwieser, A. Baumbach and others.

Development Methodologies: CR, CI, CD, Co-Dev. & Containers

Covered by this workflow:

- Software development for BrainScaleS
  - RTL (FPGA and ASIC): digital testing against simulation
- Modifications to the containerized environment
  - incl. external software dependencies
- Infrastructure (neuromorphic resources)
- Models, experiments and publications
- Executable documentation (cf. BrainScaleS hands-on at NICE)
- (Not yet integrated: full-software-stack-full-chip simulation)

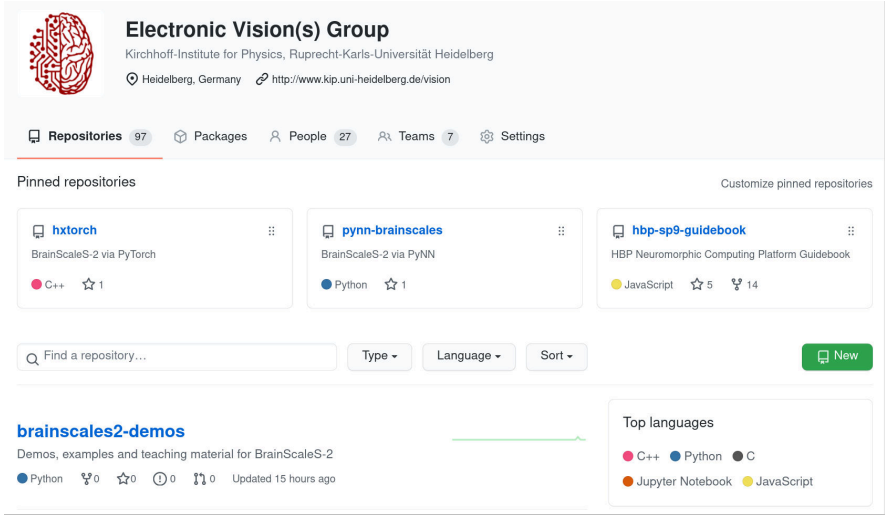
Summary

- Long-lived neuromorphic hardware architectures
  $\rightarrow$ technical debt
- Different aspects of hardware usage and “top-level” use cases
  $\rightarrow$ users have different views and requirements
- BrainScaleS-2 introduces more operation modes (standalone, ANNs)
  $\rightarrow$ minimize development overhead
- During hardware commissioning resources are scarce and large-scale installations are stationary
  $\rightarrow$ time sharing, resource management and monitoring
- Large(r)-scale usage (many users and/or platform operation)
  $\rightarrow$benefits from efforts towards reproducibility and sustainability

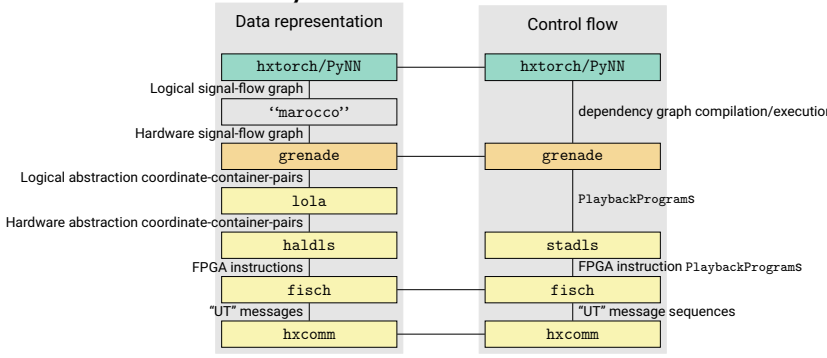
Electronic Vision(s)

Backup

Open Source — https://github.com/electronicvisions



BrainScaleS-2 Software Layers



- Layers encapsulate aspects of the system: e.g. communication, FPGA “programs”, configurable units, aggregate chip view, signal graphs $\rightarrow$ one change in hardware should only affect one corresponding layer.
- “Modern” C++, running on x86_64 and ARM64, some parts build on BSS-2 embedded processors (PPC)
- Possibility for serialization provided at each layer transition
- Most layers provide complete Python wrapping