SAND2024-05223C

Simulating Advanced Architectures for Fast Exploration

James A. Boyle[1,2], Mark Plagge[2], Suma George Cardwell[2], Frances S. Chance[2], Andreas Gerstlauer[1]

[1]System-Level Architecture and Modeling (SLAM) Lab
Department of Electrical and Computer Engineering
The University of Texas at Austin

[2] Sandia National Laboratories, Albuquerque, NM

The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

SLAM Lab
System-Level Architecture and Modeling Group

Sandia National Laboratories

SAND 2024-xxxxxx

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Tutorial Setup

- **Interactive tutorial with hands-on demo**
  - Live walk-through & exercises
  - Linux & command-line based
- **Linux Docker image provided**
  - SANA-FE image: **`jamesaboyle/sana-fe`**
  - Install from source possible but not recommended for this tutorial
- **Docker Desktop available at: docker.com/products/docker-desktop/**

Outline

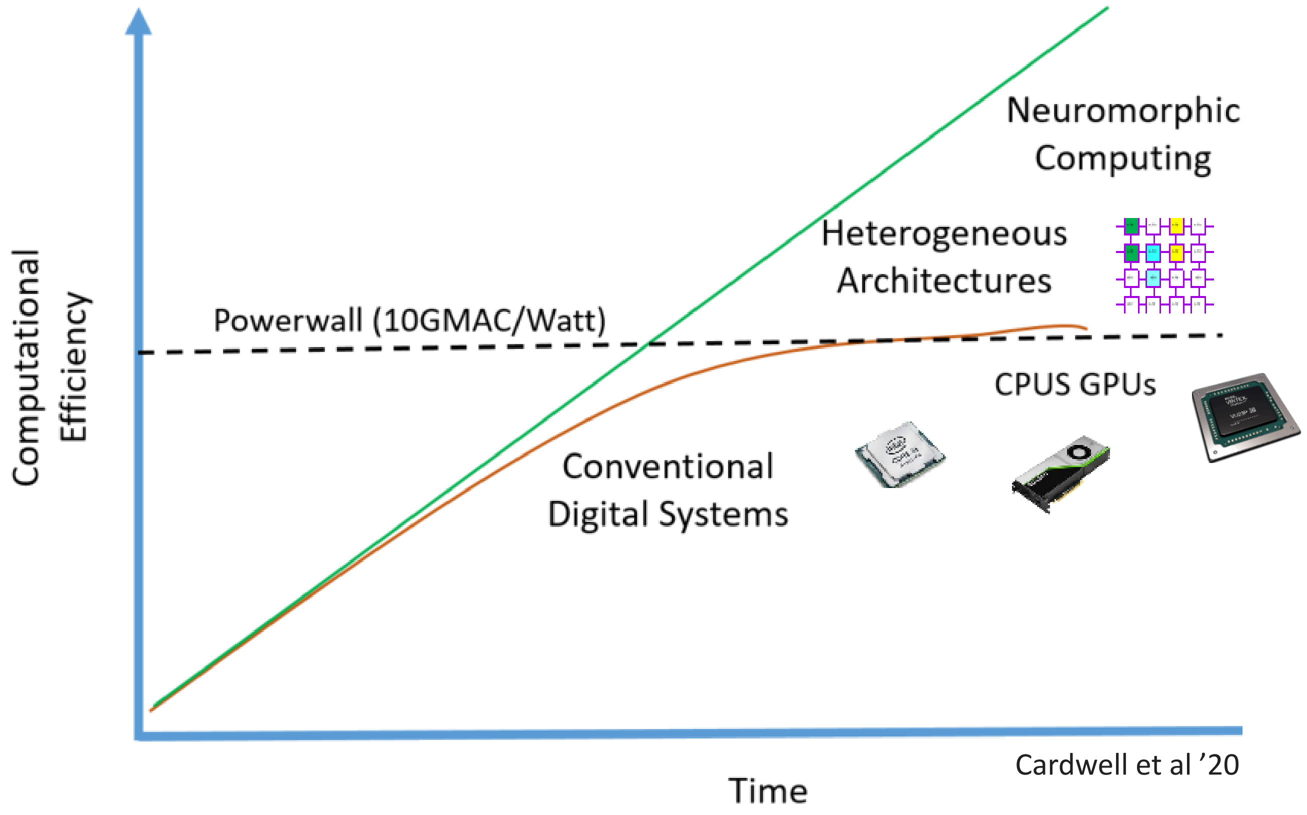
✓ Tutorial Setup

- **Background**
- **Hands-on Demo**
- **Mapping Challenge**

Background

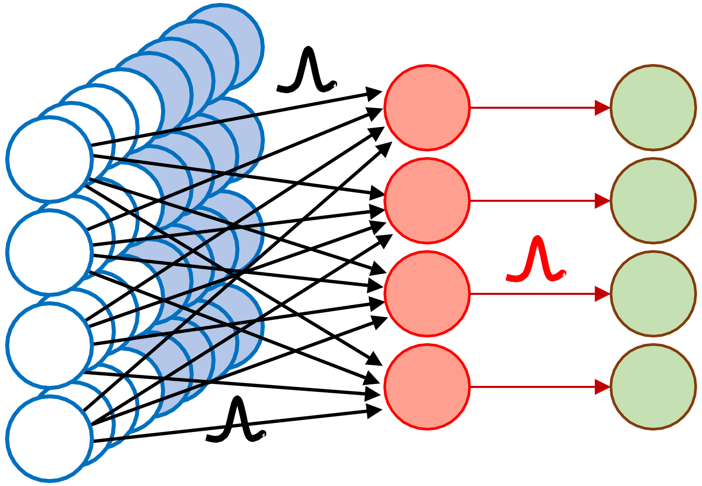
- **Power efficiency is critical**
  - Limits of scaling
  - Increased computing demands

- **Neuromorphic H/W**
  - Neural-inspired
  - Different architectures proposed
  - Novel design elements

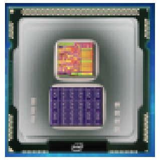


Spiking Hardware Platforms

- **Various chips proposed & deployed**
  - Execute spiking neural networks (SNN)
  - Achieve higher efficiency than conventional H/W
- **Different design approaches**
  - Digital designs
  - Analog & mixed-signal designs
  - Neural models, fully-custom, wafer-scale

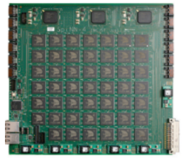


Digital Platforms



Intel Loihi 1&2

Davies 2018



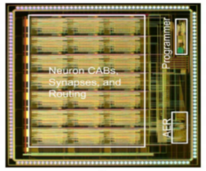
SpiNNaker 1&2

Furber 2016

IBM TrueNorth

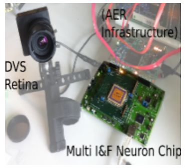
Akopyan 2016

Analog/Mixed-signal Platforms

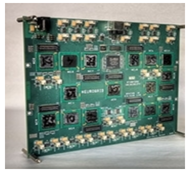


GT Neuron

Brink 2013

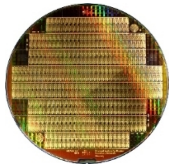


DYNAPSEL



NeuroGrid

Benjamin 2014



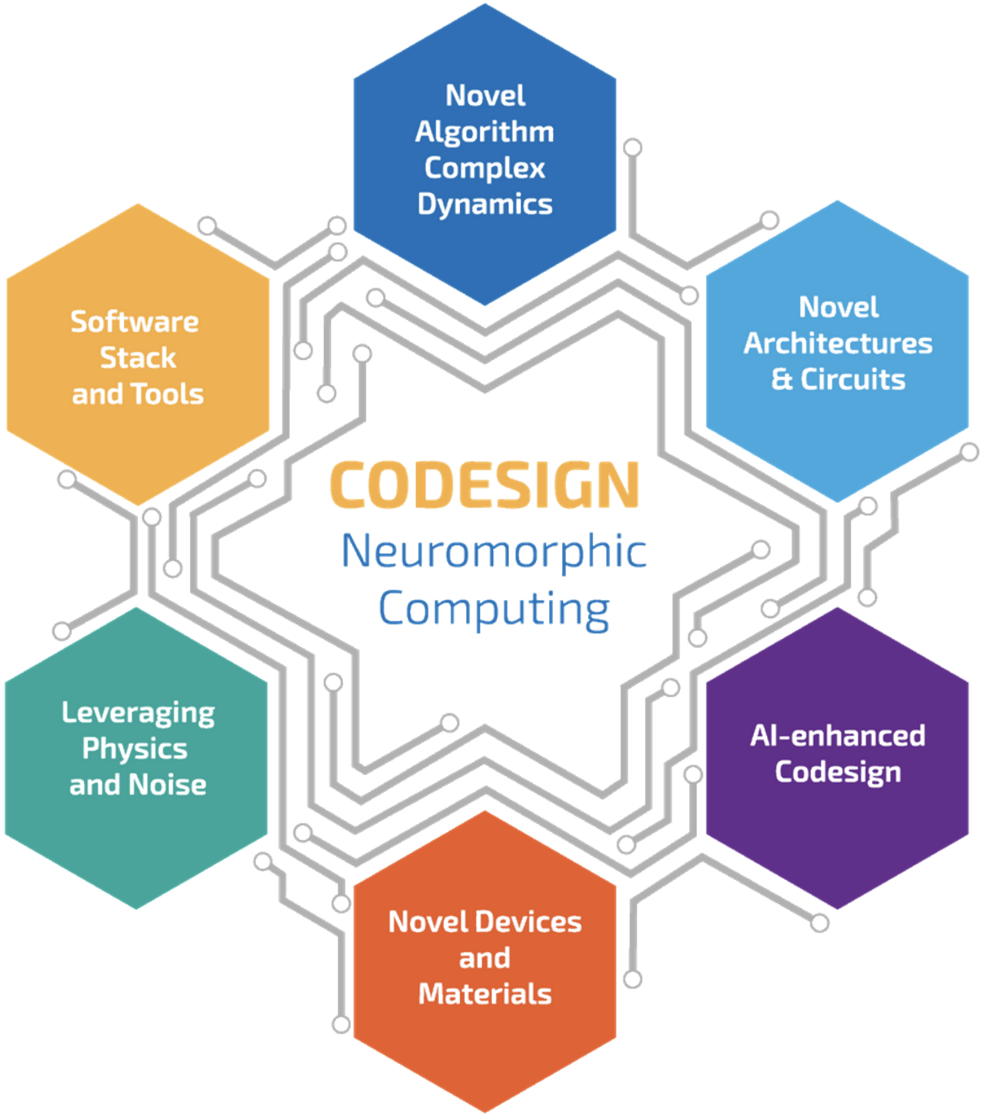
BrainScaleS-2

Pehle 2022

Neuromorphic Codesign

- **Application & architecture codesign**
  - Architecture design-space exploration
  - Algorithm development
  - Optimize for power efficiency

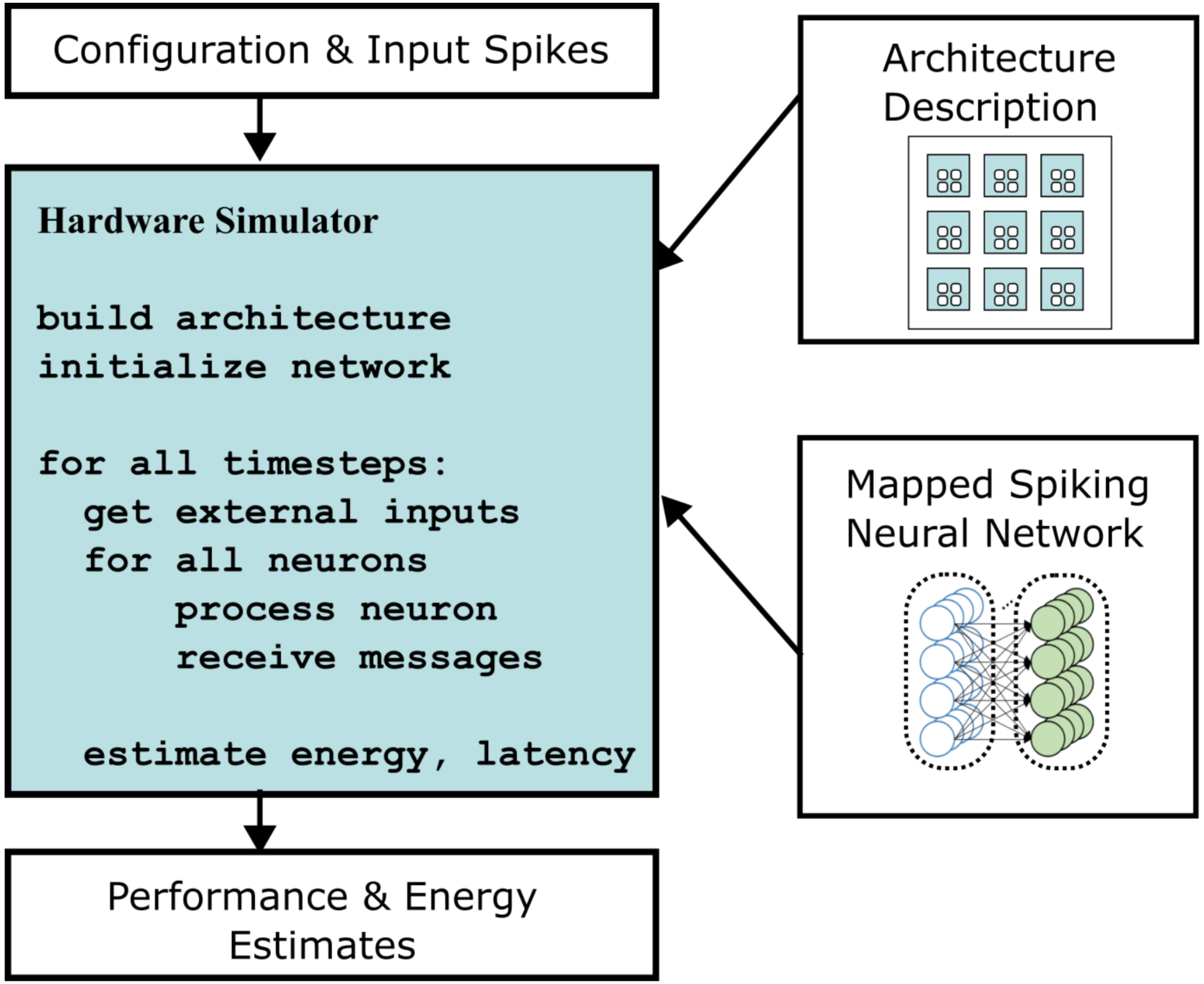
- **Need for architecture level tools**
  - Model new architectures
  - Rapid performance & energy estimates
  - Generic & extensible



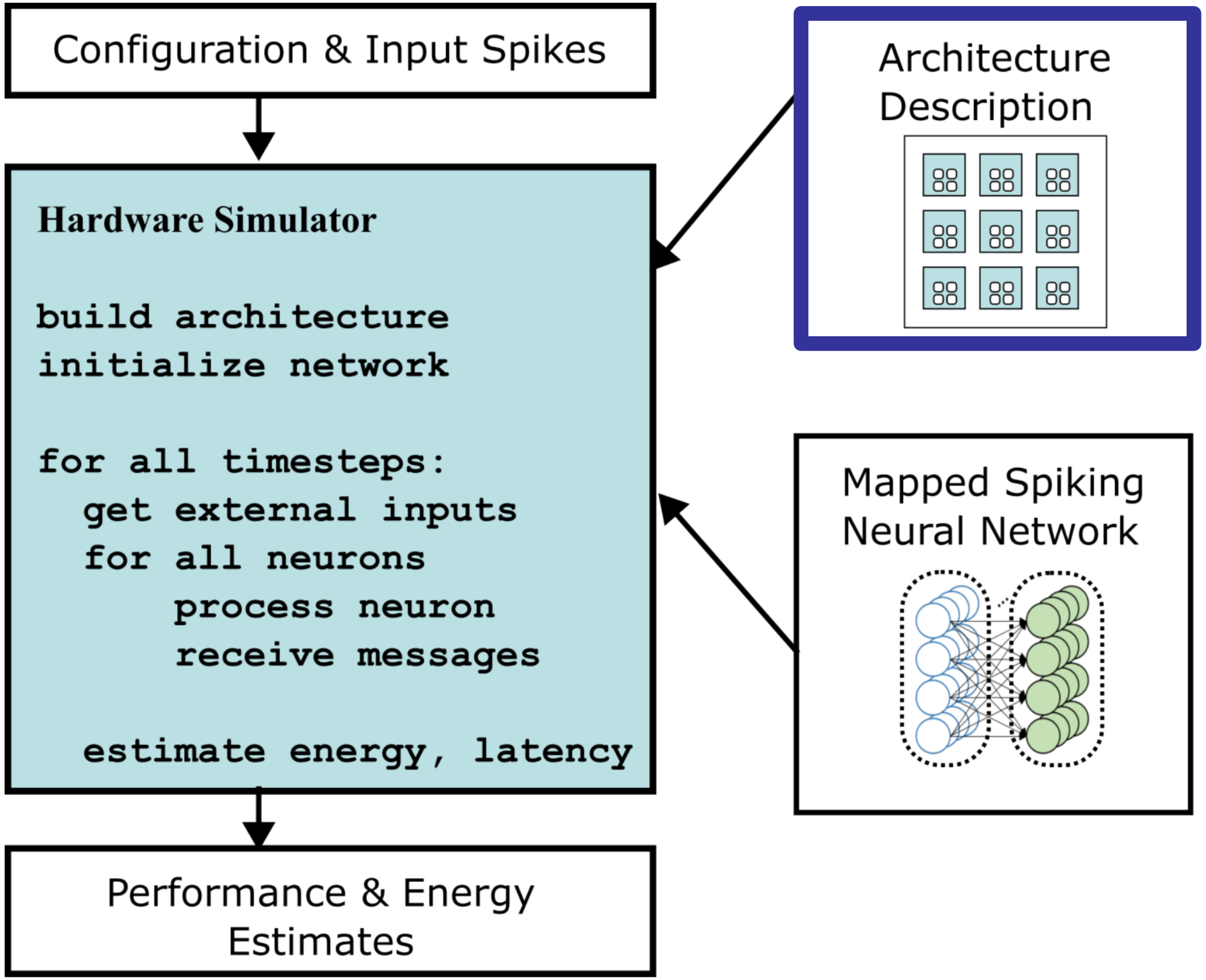
SANA-FE

Simulating Advanced Neuromorphic Architectures for Fast Exploration

SANA-FE Overview

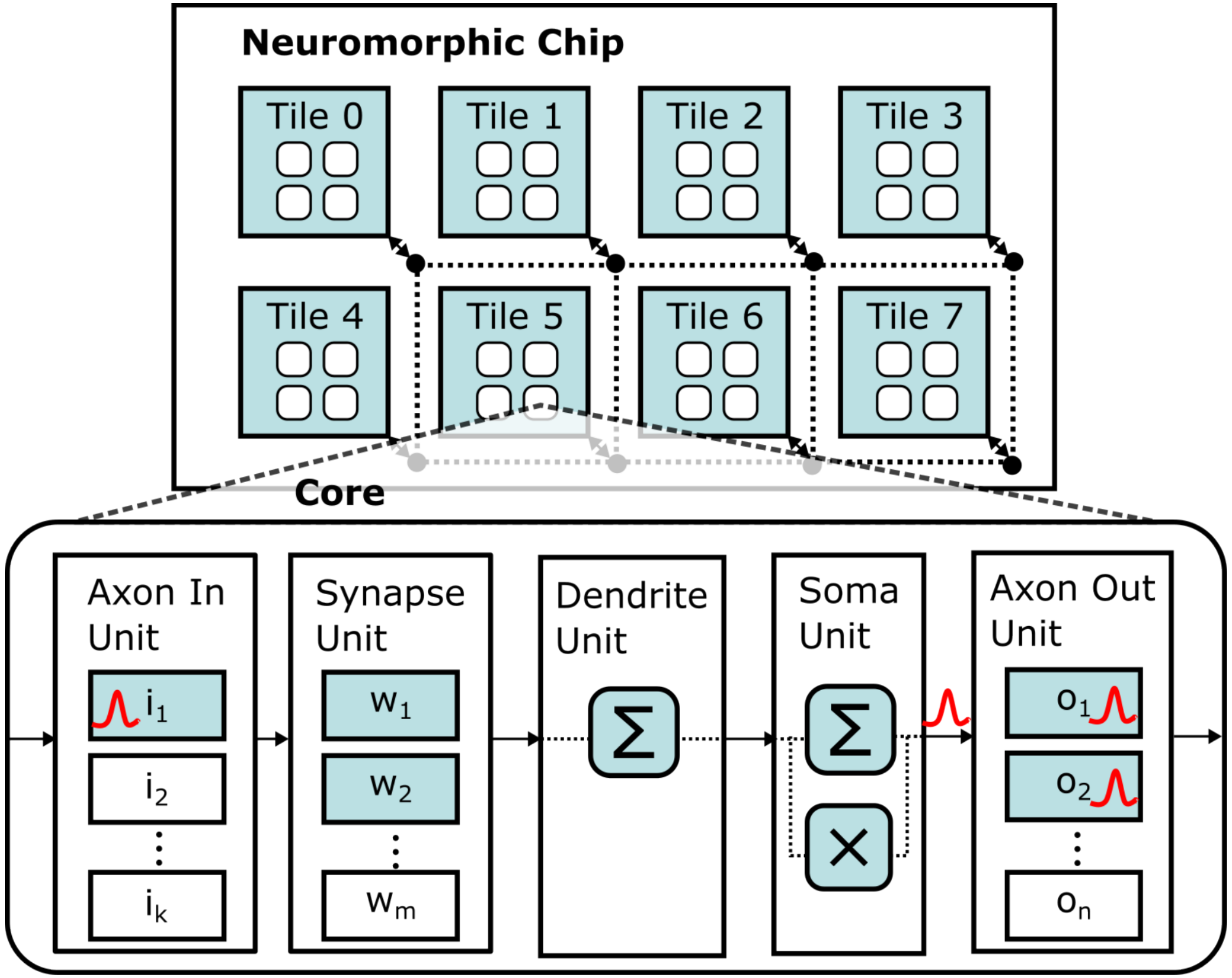


SANA-FE Overview



Spiking Architecture Template

- **Tile-based architecture**
  - Network-on-chip connecting neural cores
- **Many cores per tile**
  - Cores simulate group of mapped neurons
  - Local shared memory
- **Core pipeline**
  - Axon stage
  - Synapse stage
  - Dendrite stage
  - Soma stage

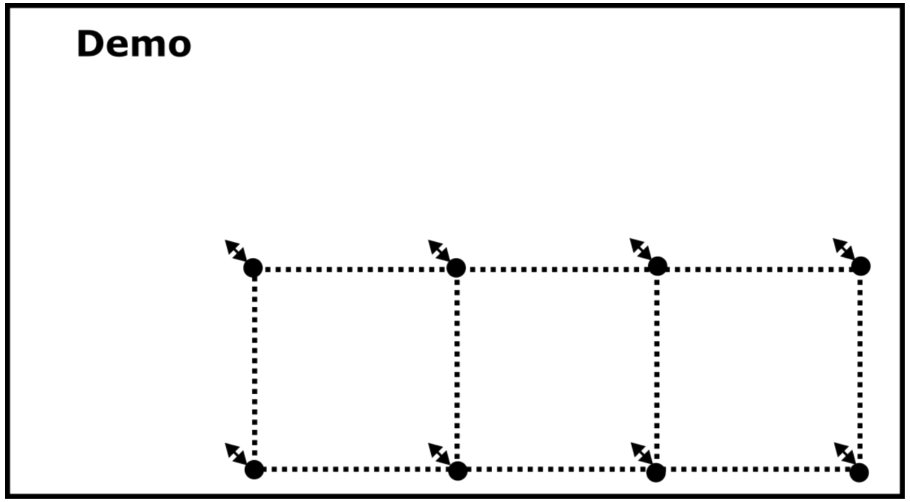


Architecture Description

- **Describes different H/W architectures**
  - Represents different existing & future spiking designs based on common features
  - Defines compute elements of chip
  - YAML-based, flexible & extensible

Architecture Description

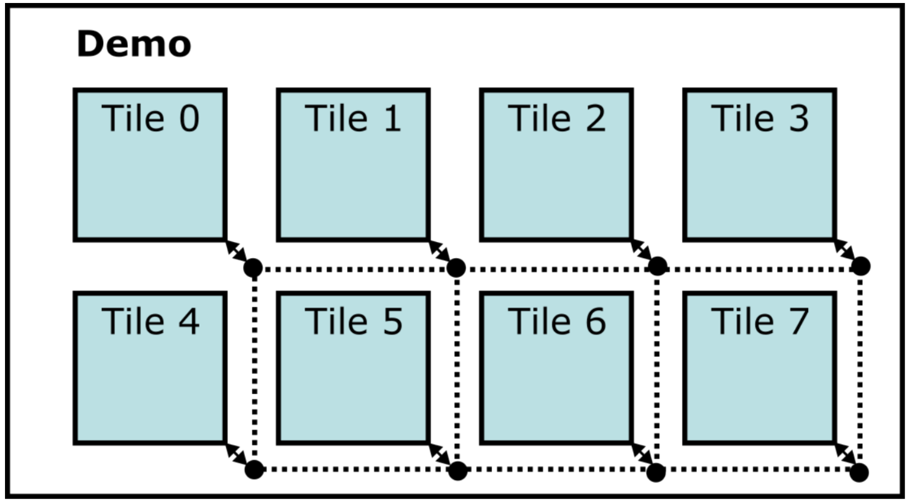
- **Describes different H/W architectures**
  - Represents different existing & future spiking designs based on common features
  - Defines compute elements of chip
  - YAML-based, flexible & extensible




```yaml
architecture:
  name: demo
```

Architecture Description

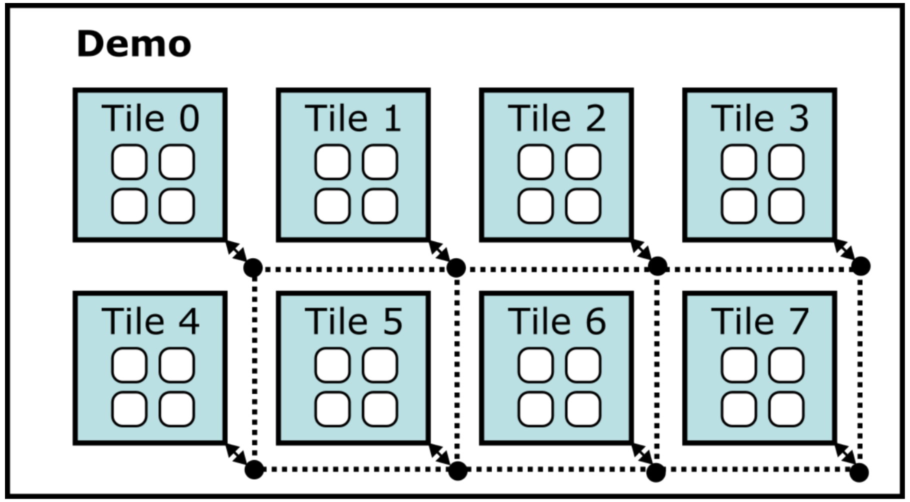
- **Describes different H/W architectures**
  - Represents different existing & future spiking designs based on common features
  - Defines compute elements of chip
  - YAML-based, flexible & extensible




```
architecture:
 name: demo
 tile:
   - name: demo_tile[0..7]
     attributes:
      energy_east_west: 1e-12
      latency_east_west: 2e-9
      ...
```

Architecture Description

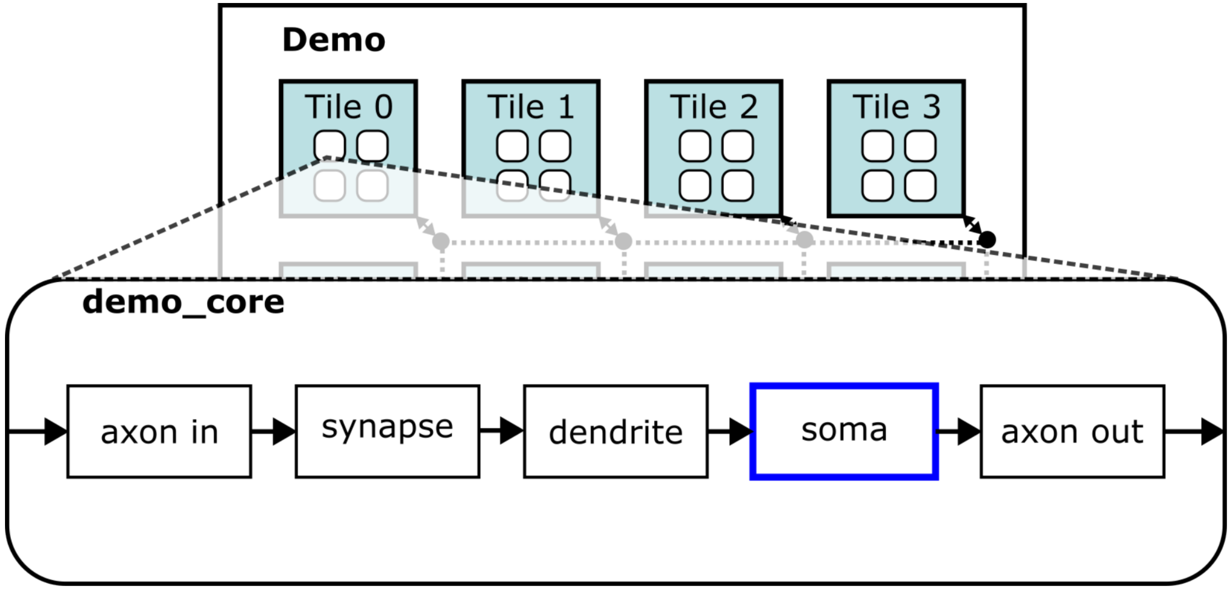
- **Describes different H/W architectures**
  - Represents different existing & future spiking designs based on common features
  - Defines compute elements of chip
  - YAML-based, flexible & extensible




```yaml
architecture:
 name: demo
 tile:
  - name: demo_tile[0..7]
    attributes:
     energy_east_west: 1e-12
     latency_east_west: 2e-9
     ...
    core:
     - name: demo_core[0..3]
```

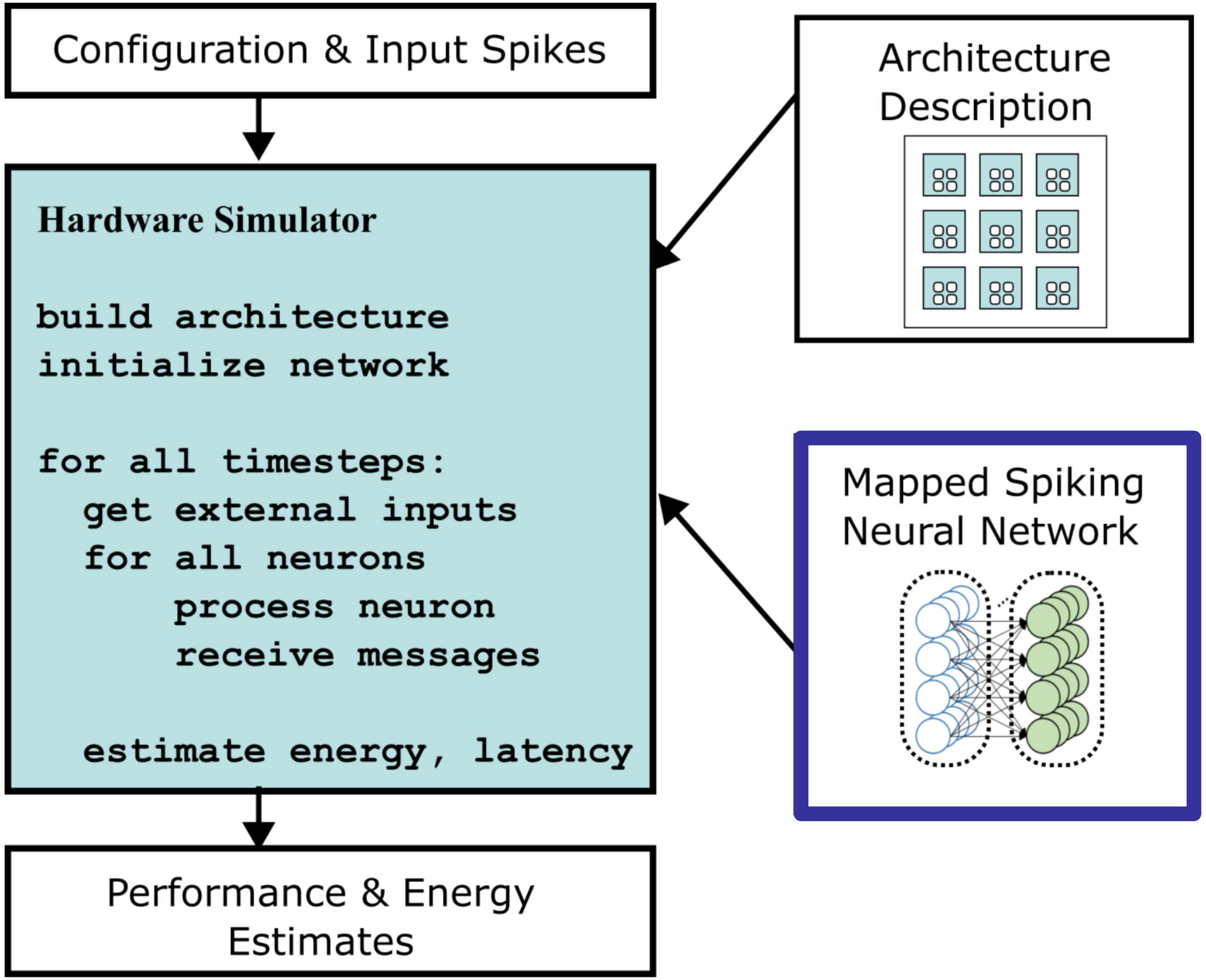
Architecture Description

- **Describes different H/W architectures**
  - Represents different existing & future spiking designs based on common features
  - Defines compute elements of chip
  - YAML-based, flexible & extensible




```yaml
architecture:
 name: demo
 tile:
   - name: demo_tile[0..7]
     attributes:
      energy_east_west: 1e-12
      latency_east_west: 2e-9
      ...
     core:
      - name: demo_core[0..3]
        soma:
          - name: core_lif
            attributes:
             energy_spiking: 68e-12
             latency_spiking: 30e-9
...
```

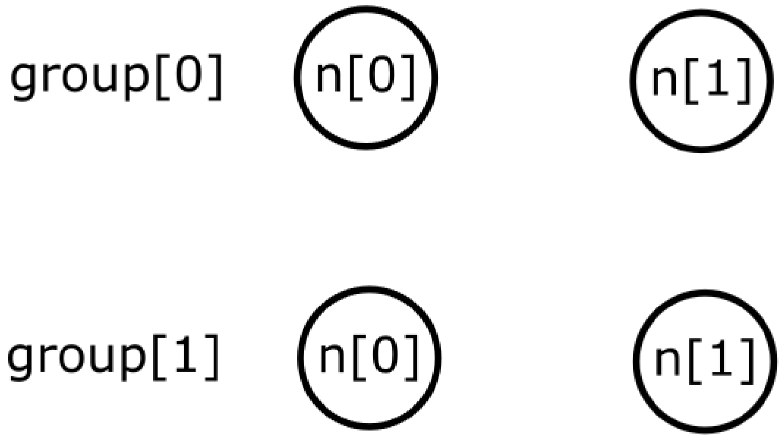
SANA-FE



Configuration & Input Spikes
Hardware Simulator
build architecture
initialize network
for all timesteps:
get external inputs
for all neurons
process neuron
receive messages
estimate energy, latency
Performance & Energy Estimates
Architecture Description
Mapped Spiking Neural Network

Mapped Spiking Neural Network

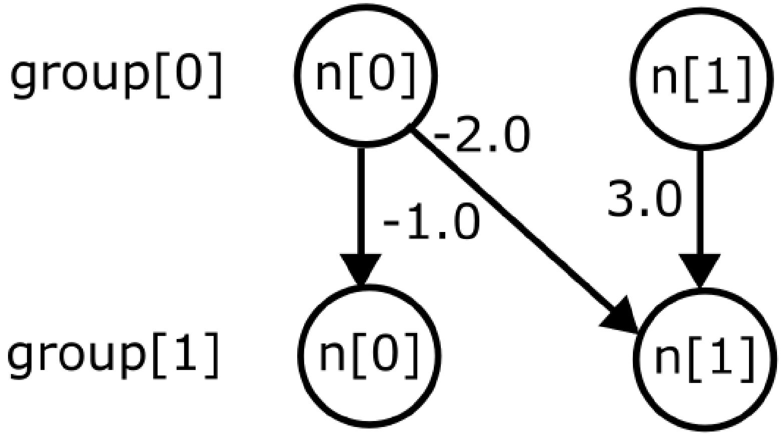
- **Describes SNN application**
  - One entry per line
  - Groups (g), neurons (n), edges (e) and H/W mappings to cores (&)
  - Optional list of named attributes




```
## Groups and neurons
g 2 threshold=1.0 reset=0.0
g 2 threshold=2.0 reset=0.0
n 0.0 bias=1.0 connections_out=1
n 0.1 bias=1.0 connections_out=1
n 1.0 bias=0.0 connections_out=1
n 1.1 bias=0.0
```

Mapped Spiking Neural Network

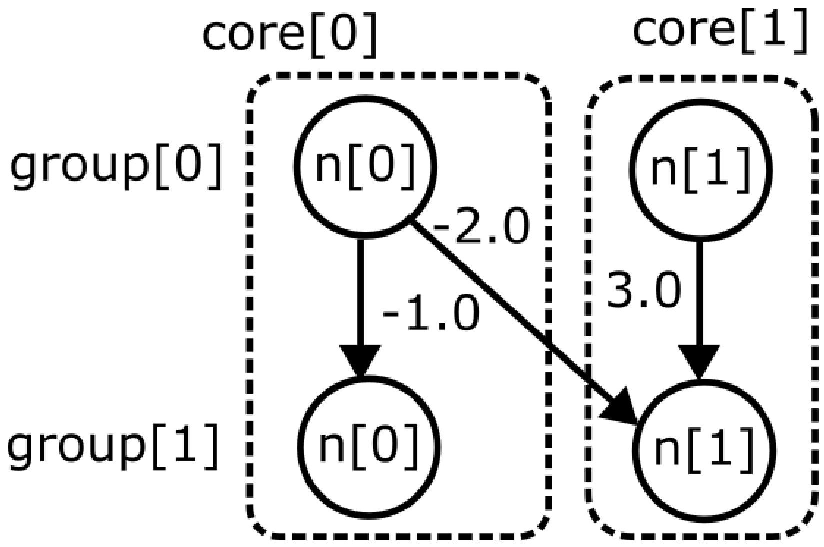
- **Describes SNN application**
  - One entry per line
  - Groups (g), neurons (n), edges (e) and H/W mappings to cores (&)
  - Optional list of named attributes




```
## Groups and neurons
g 2 threshold=1.0 reset=0.0
g 2 threshold=2.0 reset=0.0
n 0.0 bias=1.0 connections_out=2
n 0.1 bias=1.0 connections_out=1
n 1.0 bias=0.0
n 1.1 bias=0.0
## Edges
e 0.0->1.0 weight=-1.0
e 0.1->1.1 weight=-2.0
e 1.0->1.1 weight=3.0
```

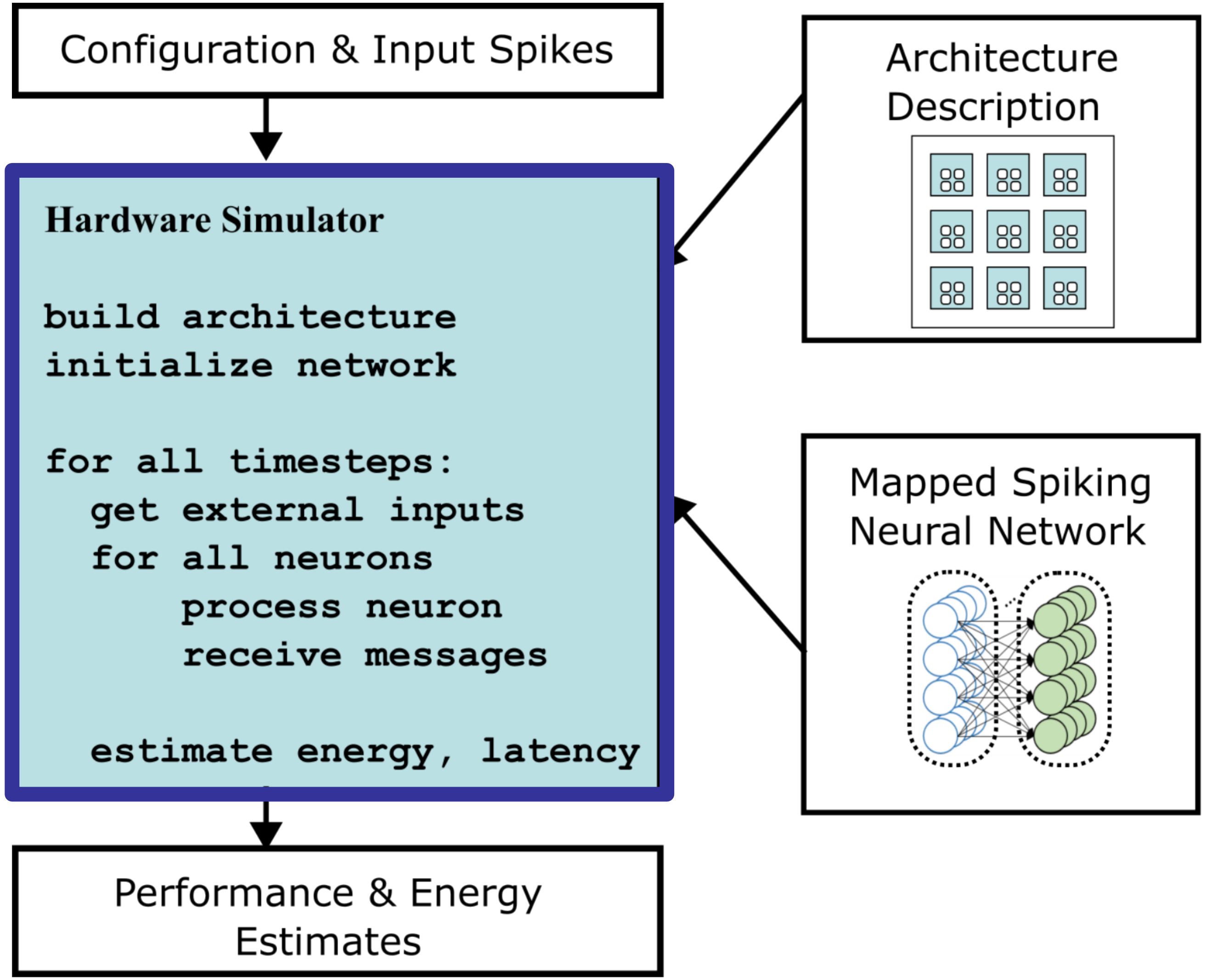
Mapped Spiking Neural Network

- **Describes SNN application**
  - One entry per line
  - Groups (g), neurons (n), edges (e) and H/W mappings to cores (&)
  - Optional list of named attributes




```
## Groups and neurons
g 2 threshold=1.0 reset=0.0
g 2 threshold=2.0 reset=0.0
n 0.0 bias=1.0 connections_out=2
n 0.1 bias=1.0 connections_out=1
n 1.0 bias=0.0
n 1.1 bias=0.0
## Edges
e 0.0->1.0 weight=-1.0
e 0.0->1.1 weight=-2.0
e 0.1->1.1 weight=3.0
## Mappings
& 0.0@0.0
& 0.1@0.0
& 1.0@0.1
& 1.1@0.1
```

SANA-FE


Configuration & Input Spikes
Hardware Simulator
build architecture
initialize network
for all timesteps:
get external inputs
for all neurons
process neuron
receive messages
estimate energy, latency
Performance & Energy Estimates
Architecture Description
Mapped Spiking Neural Network

Simulator Kernel

- **Executes application on a given architecture**
  - Loads architecture and SNN from file
  - Simulates on-chip activity in loop
- **Detailed performance output**
  - Estimate energy & latency every time-step
  - Spike traces & H/W insight
- **Abstract coarse-grained**
  - Fast time-step based simulation
  - Compared to event-driven

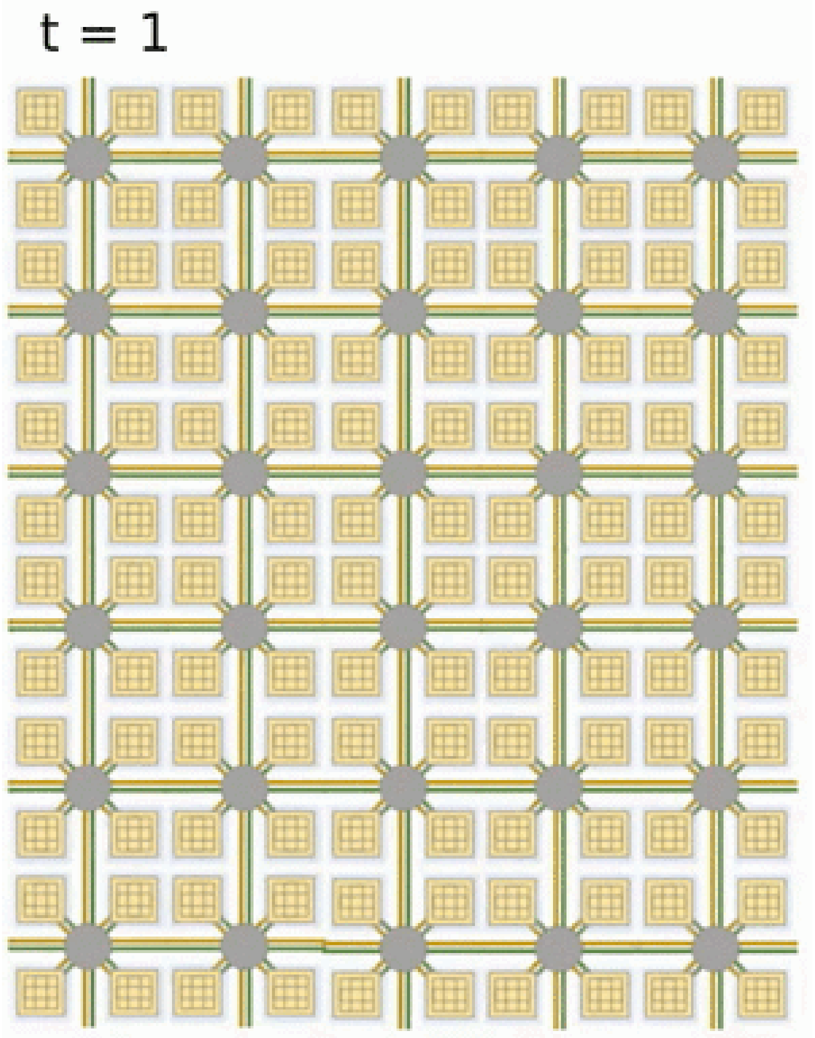
Hardware Simulator

```
build architecture
initialize network

for all timesteps:
   get external inputs
   for all neurons
         process neuron
         receive messages

   estimate energy, latency
```

Time-step Based Execution

- **Digital chips execute in logical time**
  - Core iterates over mapped neurons
  - Neurons share core H/W resources
  - Improved scaling

- **Time-step based approach**

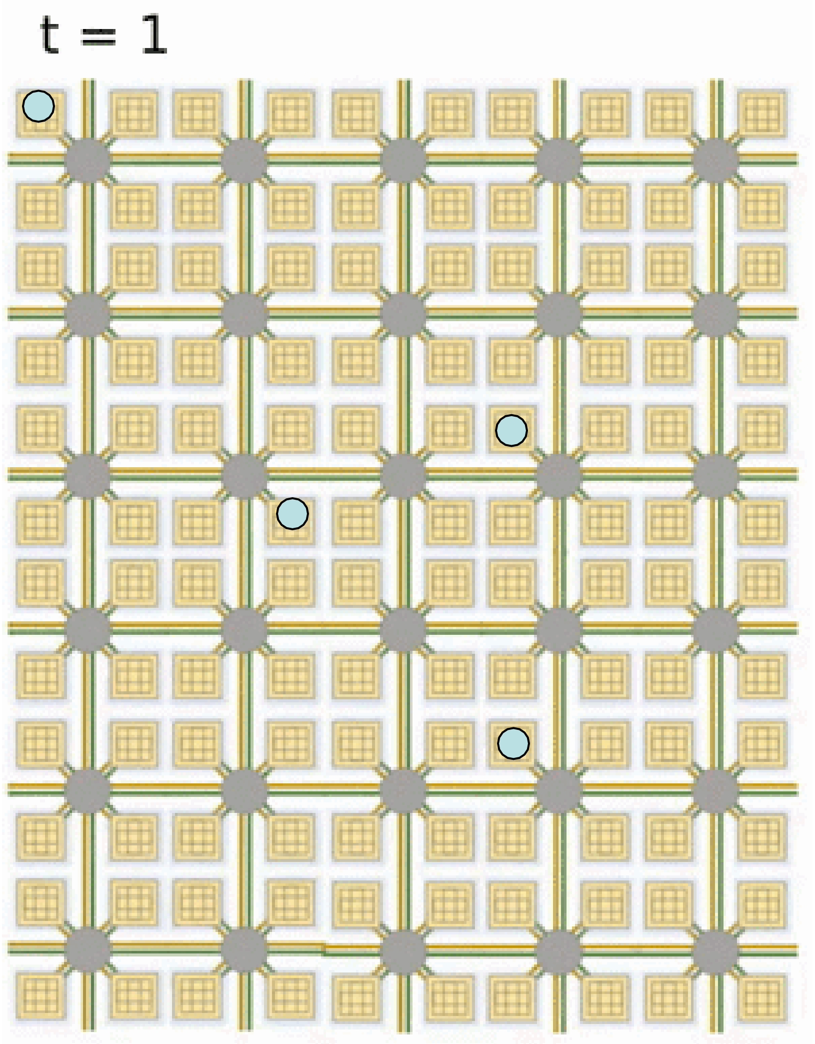


Wikichip [accessed 2023]

Time-step Based Execution

- **Digital chips execute in logical time**
  - Core iterates over mapped neurons
  - Neurons share core H/W resources
  - Improved scaling

- **Time-step based approach**
  - Update neuron dynamics for small time increment

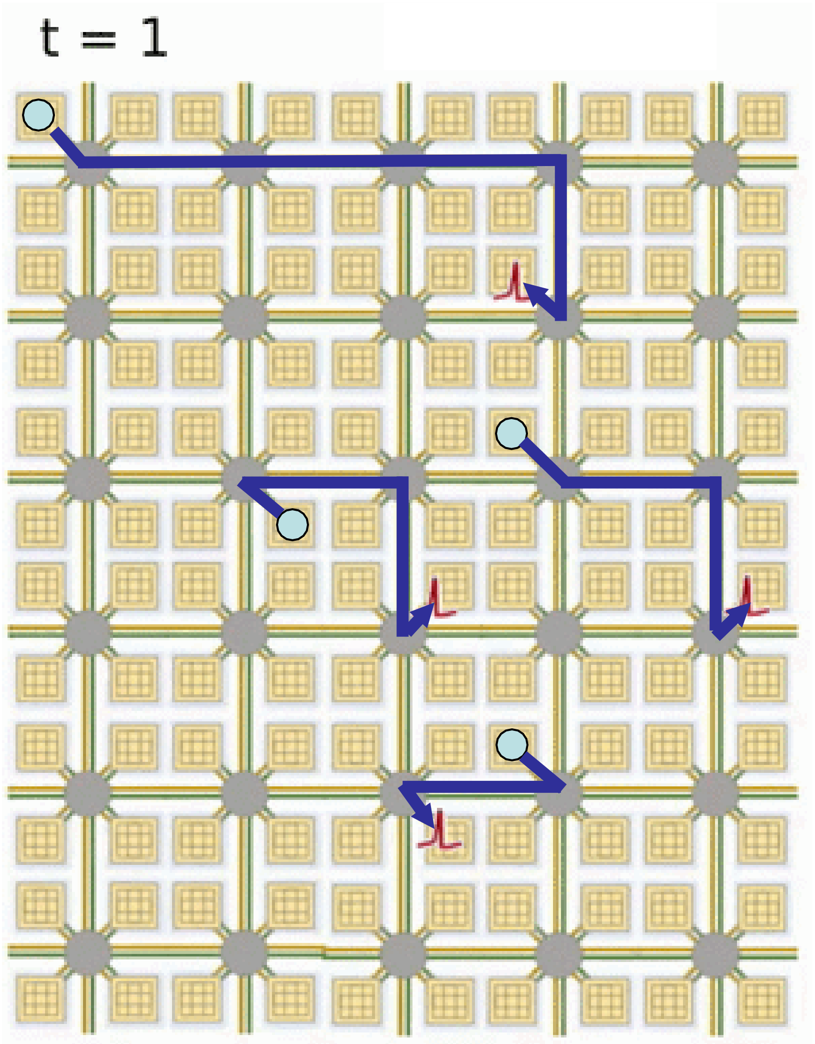


Wikichip [accessed 2023]

Time-step Based Execution

- **Digital chips execute in logical time**
  - Core iterates over mapped neurons
  - Neurons share core H/W resources
  - Improved scaling

- **Time-step based approach**
  - Update neuron dynamics for small time increment
  - Cores exchange spike messages

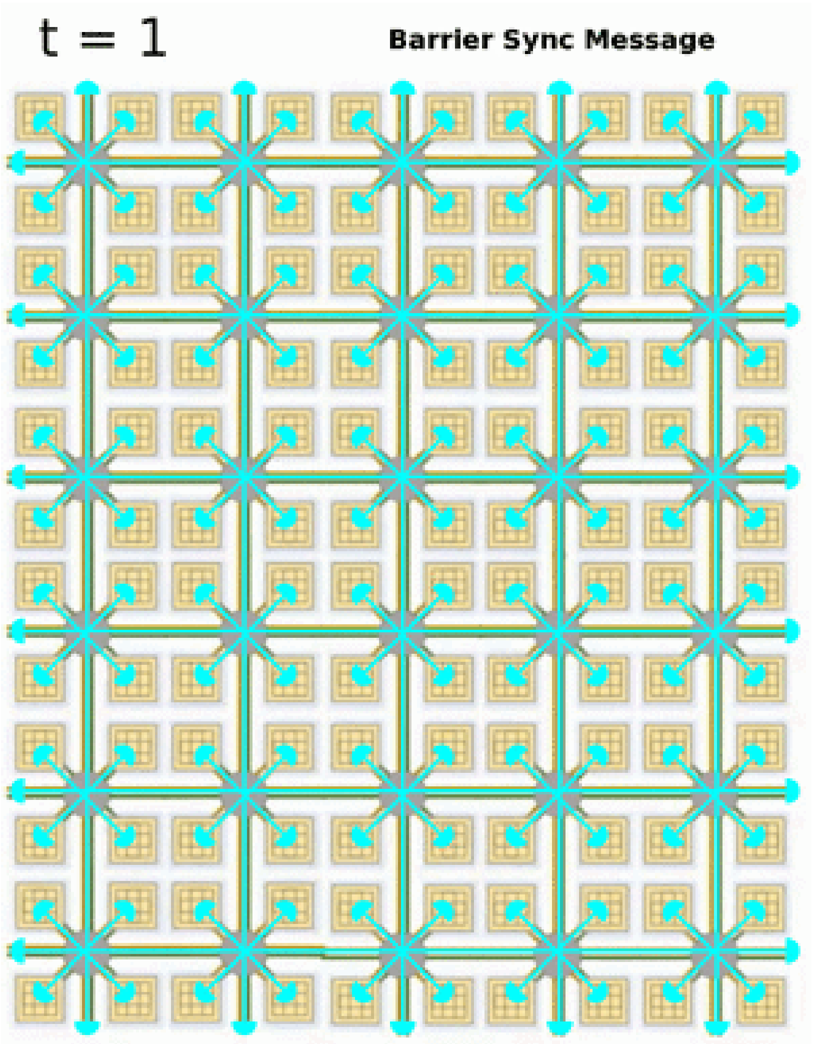


Wikichip [accessed 2023]

Time-step Based Execution

- **Digital chips execute in logical time**
  - Core iterates over mapped neurons
  - Neurons share core H/W resources
  - Improved scaling

- **Time-step based approach**
  - Update neuron dynamics for small time increment
  - Cores exchange spike messages
  - Barrier to synchronize all cores

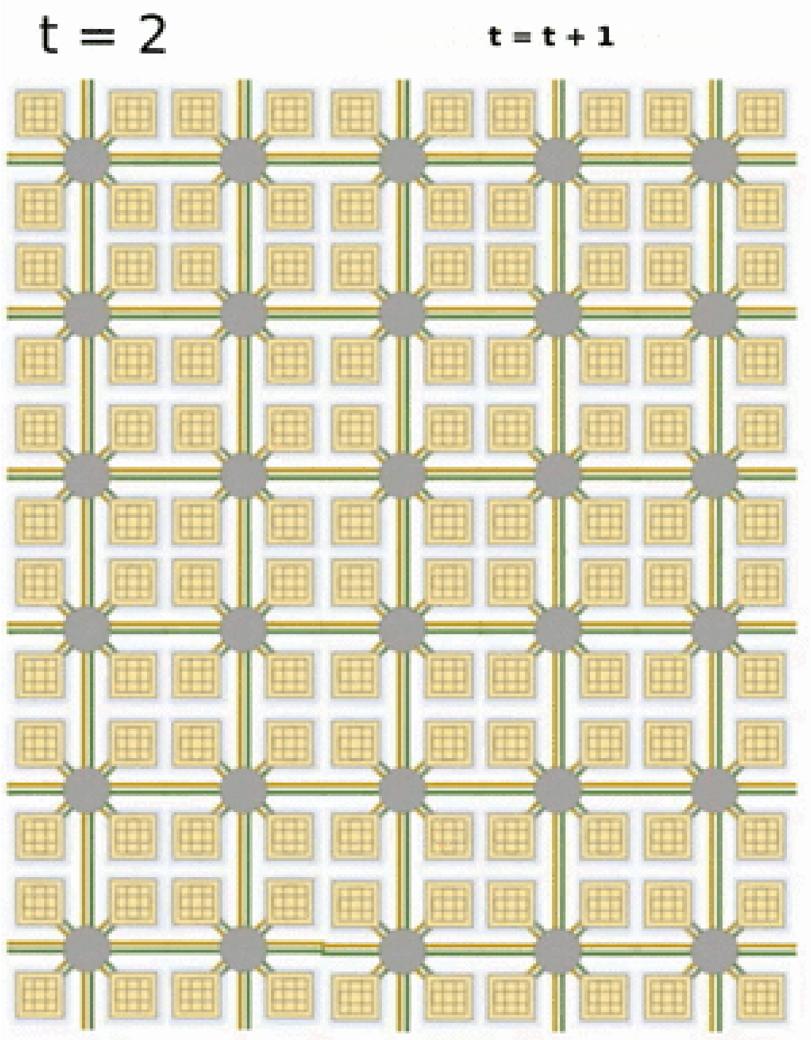


Wikichip [accessed 2023]

Time-step Based Execution

- **Digital chips execute in logical time**
  - Core iterates over mapped neurons
  - Neurons share core H/W resources
  - Improved scaling

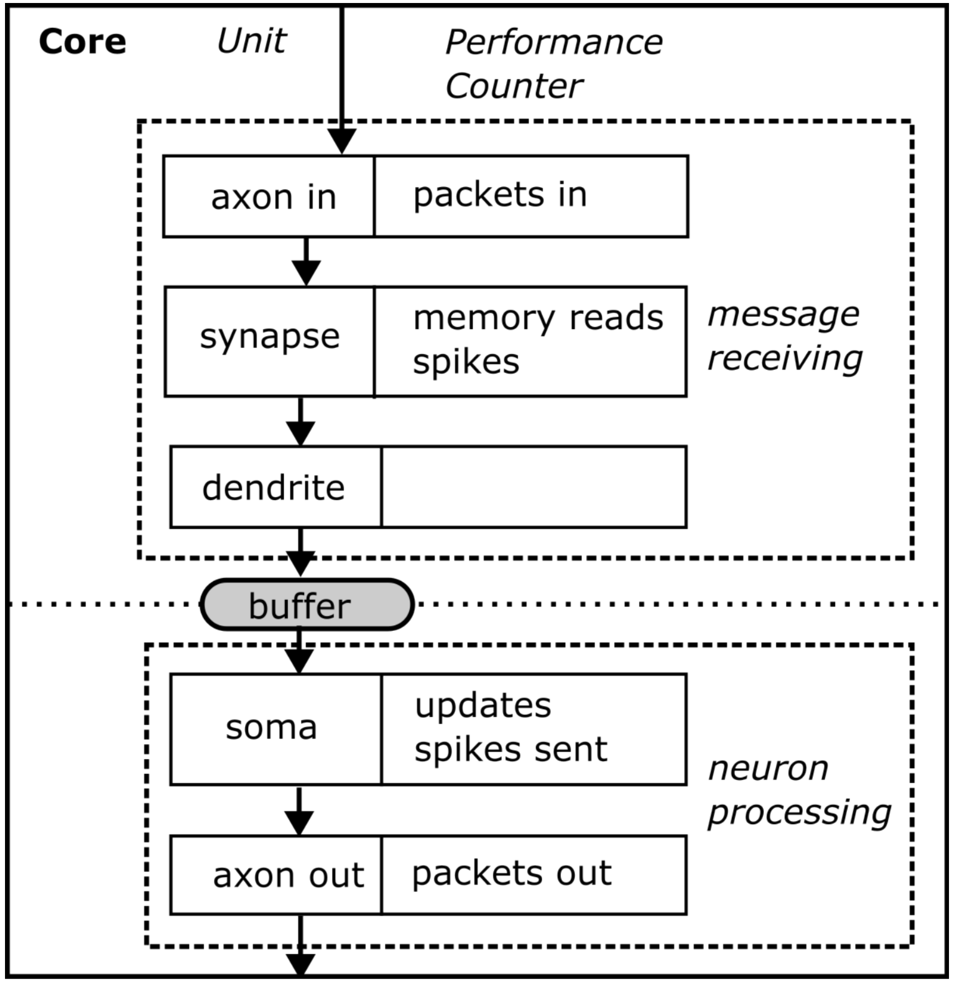
- **Time-step based approach**
  - Update neuron dynamics for small time increment
  - Cores exchange spike messages
  - Barrier to synchronize all cores
  - Increment time-step count



Wikichip [accessed 2023]

Simulator Design

- **Simulate two-stage time-step**
  - Calculate neuron dynamics according to soma model & updates neurons firing
    - *Neuron processing* stage
  - Process received spikes
    - *Message receiving* stage
- **Track & calculate total activity**
  - For power estimates every time-step
  - Sum total energy over all cores
  - Calculate latency as maximum of all stages in all cores



Outline

- ✓ Tutorial Setup
- ✓ Background
- Hands-on Demo
- Mapping Challenge

SANA-FE Tutorial

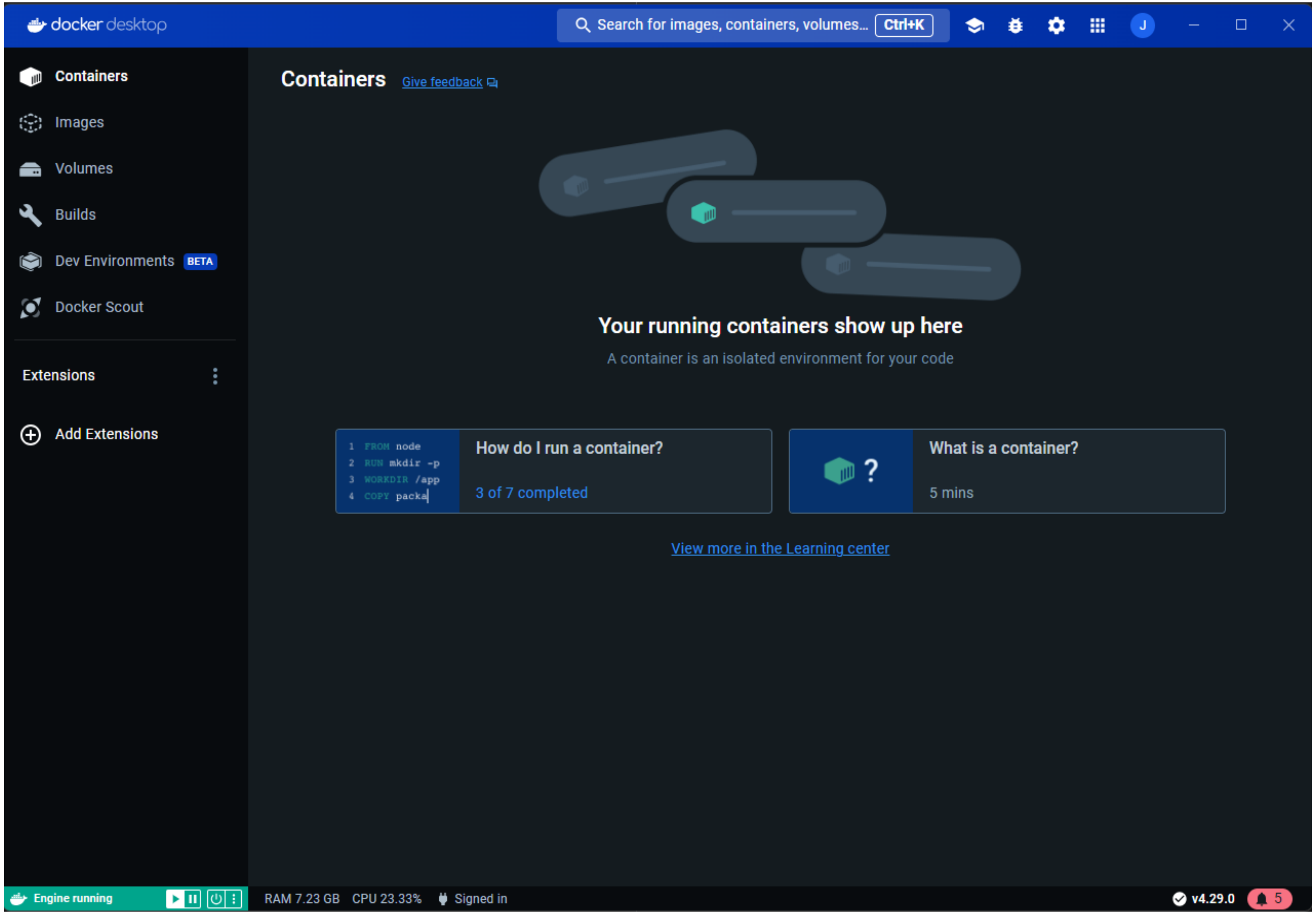
- **Interactive & hands-on demo**
  - Docker environment: `jamesaboyle/sana-fe`
  - Demonstrates SANA-FE on real-world example
  - Exercises and open-ended challenge

- **Online tutorial instructions**

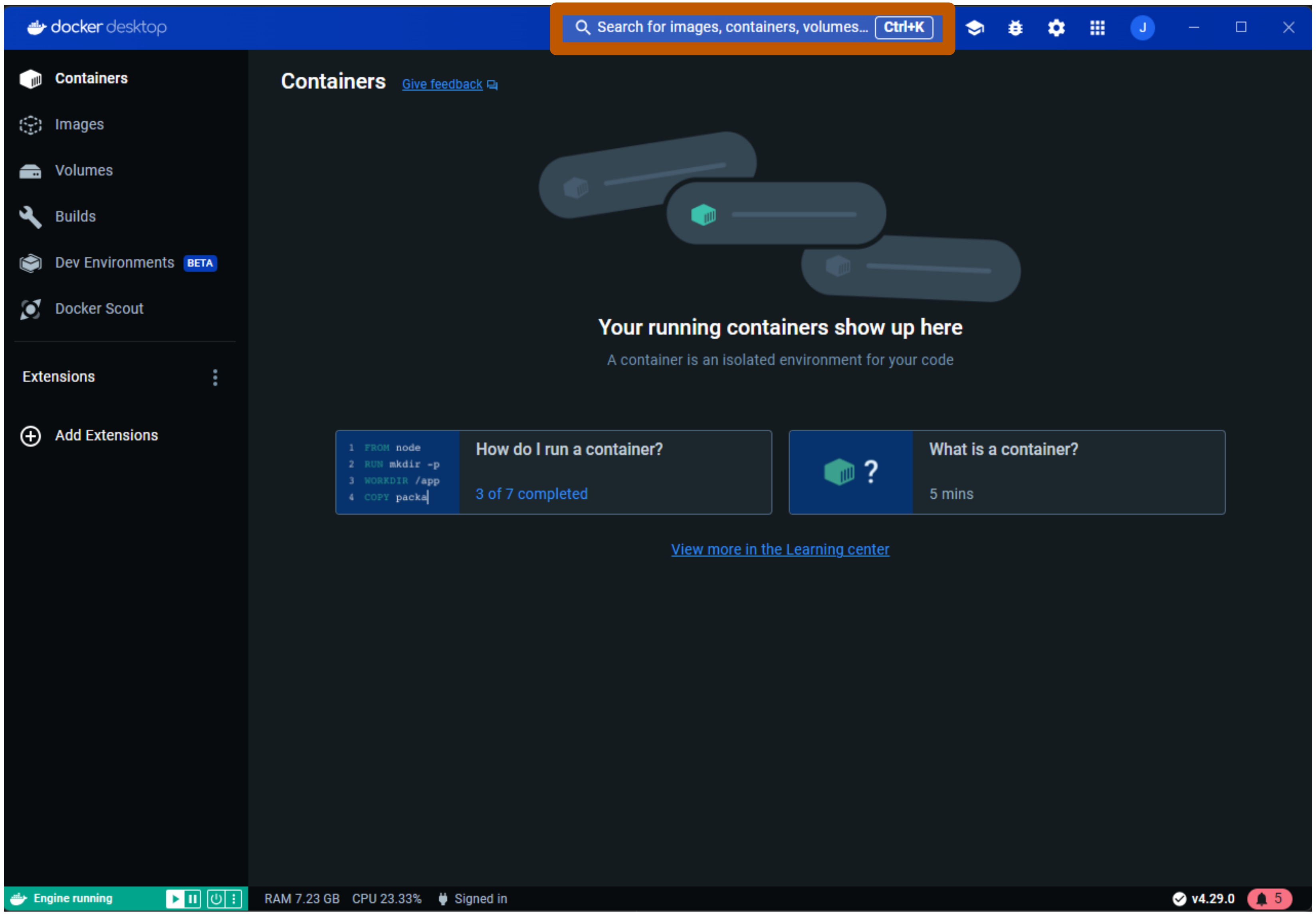
  github.com/SLAM-Lab/SANA-FE/
  - In “tutorial” folder
  - View “TUTORIAL.md”
  - Or use QR code (shown right)

github.com/SLAM-Lab/SANA-FE/blob/main/tutorial/TUTORIAL.md

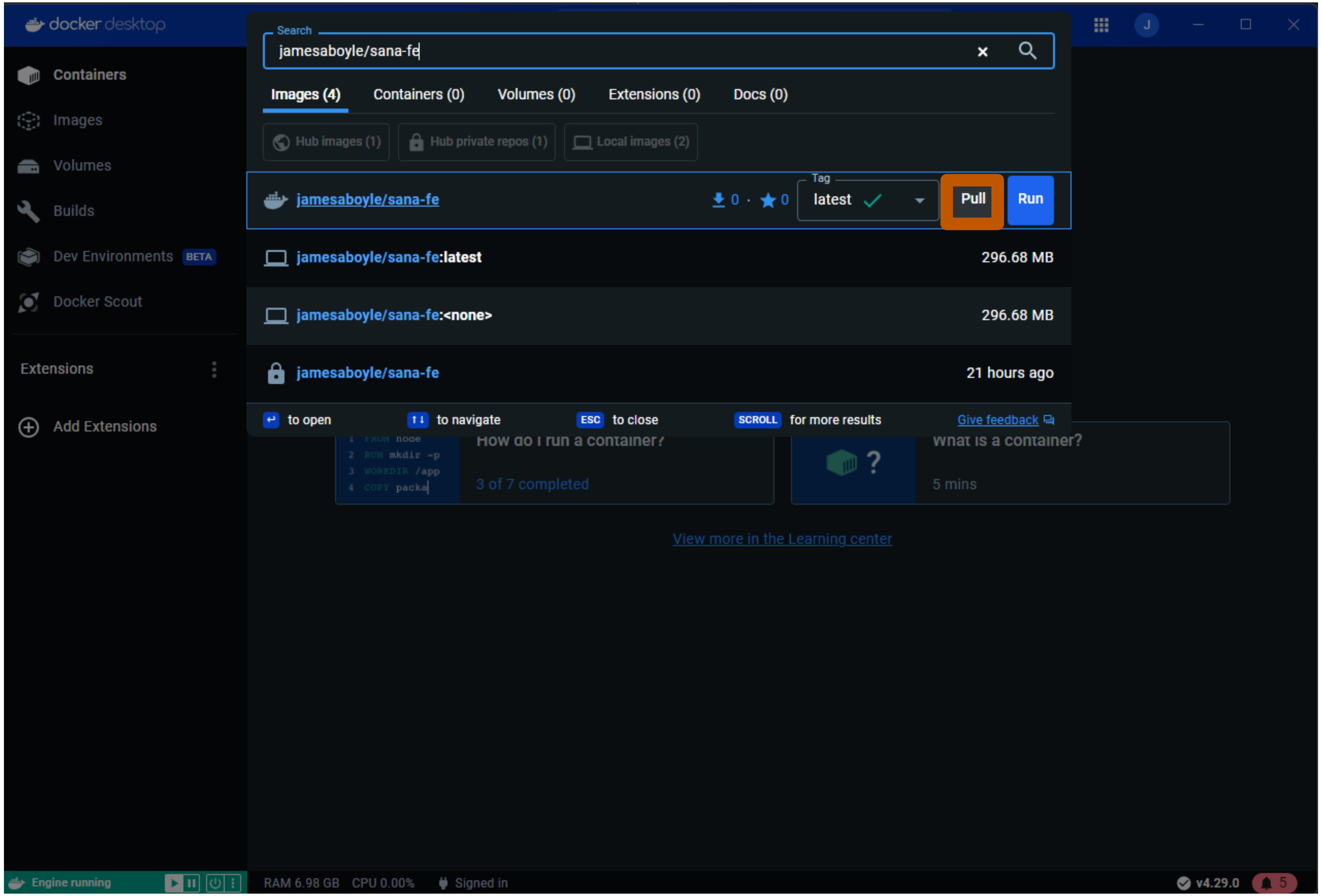
Docker Setup



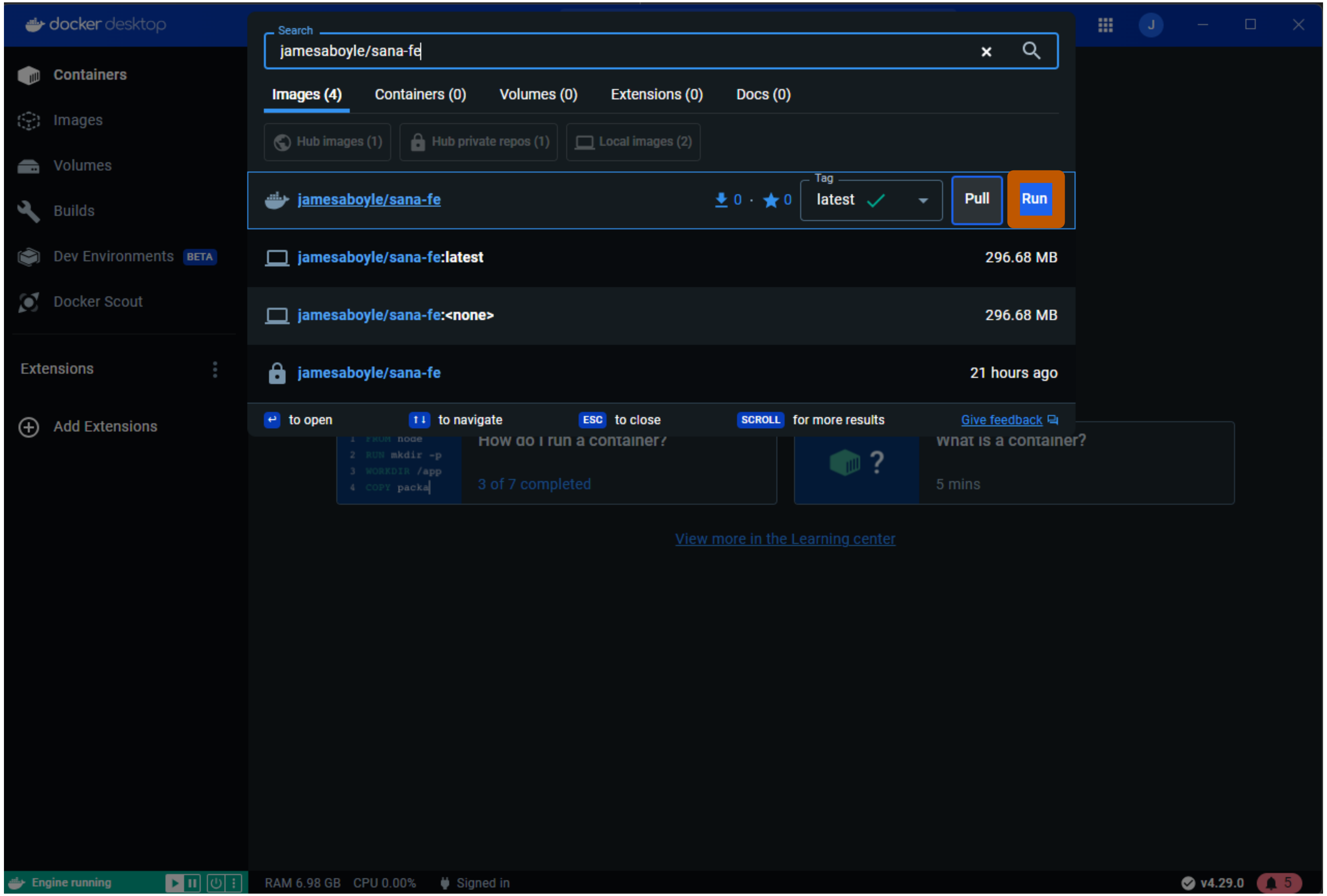
Docker Setup



Docker Setup



Docker Setup



[Optional] Installing SANA-FE without Docker

This is only needed if developing SANA-FE

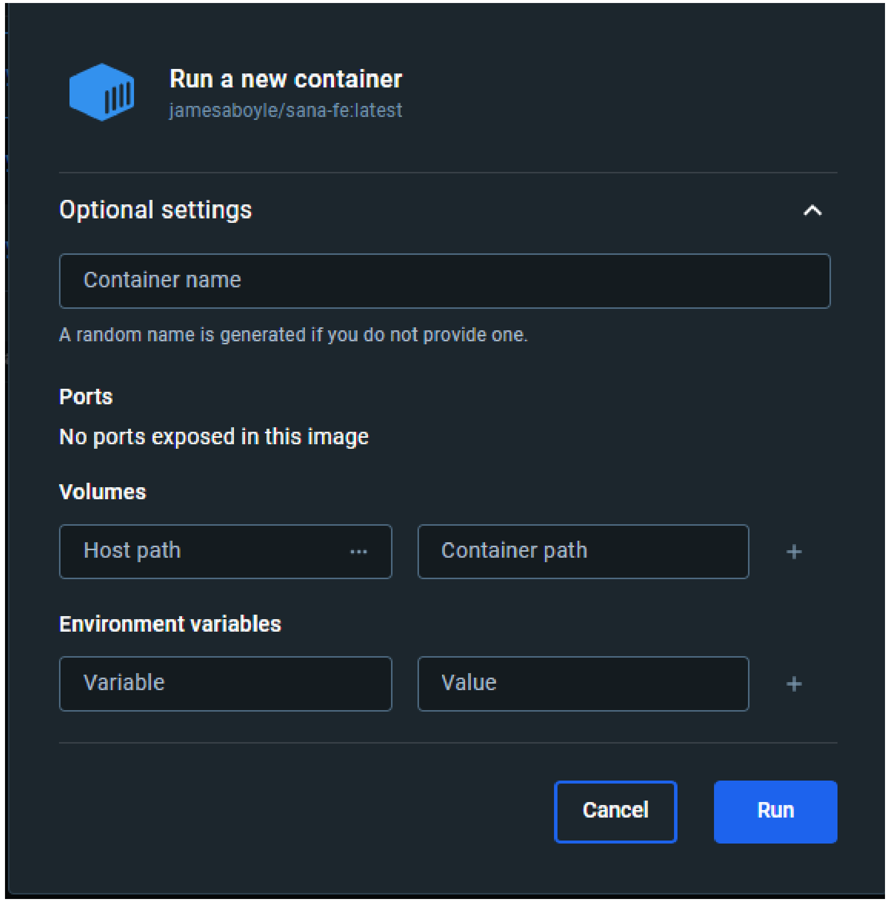
- **Build & run dependencies**
  - Build requires C compiler (C99 or later) and **make**
  - Run scripts require Python ≥ 3.6 and **pyyaml**

- **Make-based build**
  - Linux recommended
  - Code in top-level directory in `*.c` and `*.h` files

```
# git clone https://github.com/SLAM-Lab/SANA-FE sana-fe
# cd sana-fe
# make
# python -m venv ./venv && source ./venv/bin/activate
# pip install --upgrade pip && pip install pyyaml numpy
```

Running SANA-FE

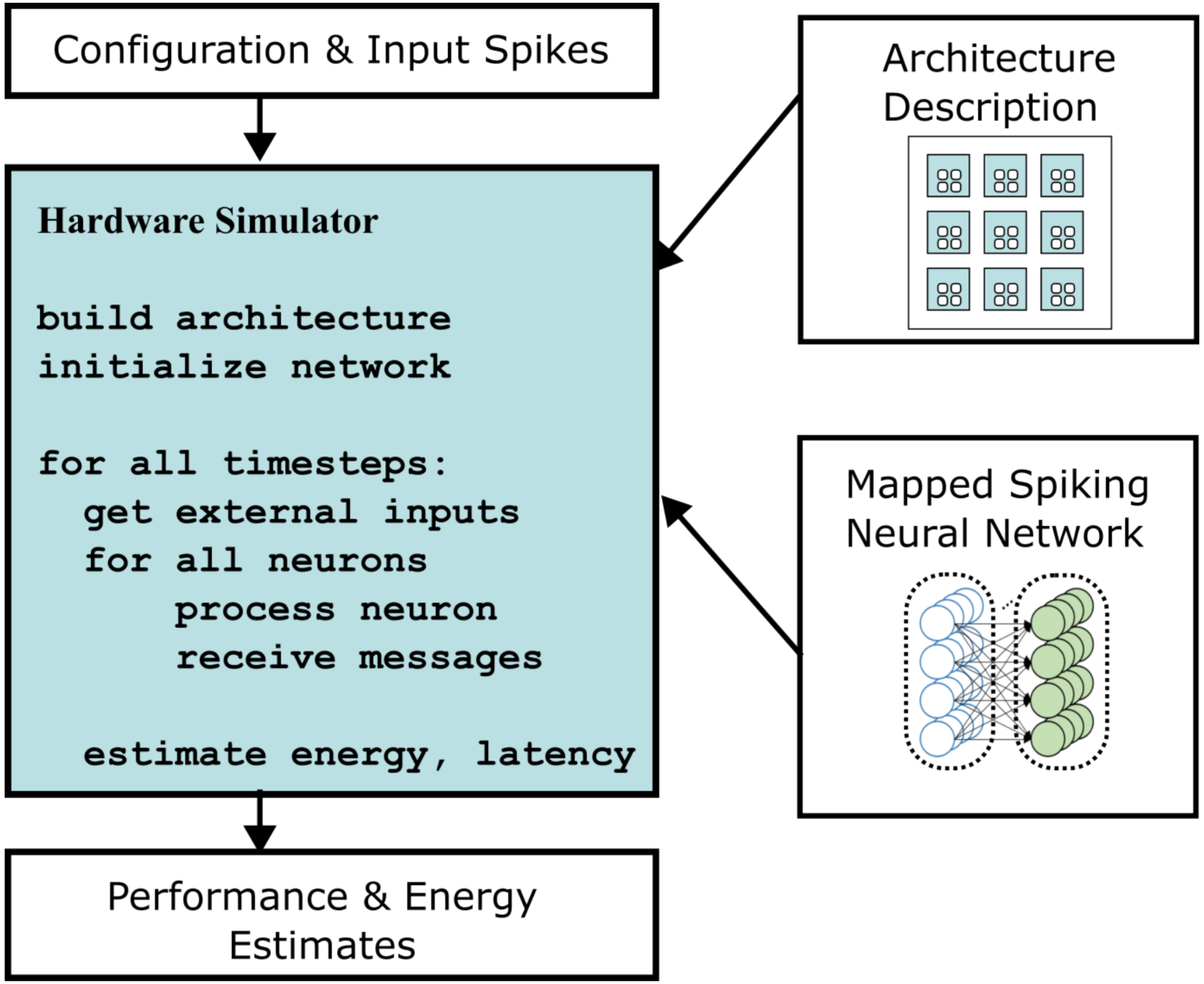
- **Start SANA-FE Docker image**
  - “Run” →“Optional Settings”→“Volumes”
    - “Host path”: Folder in host environment
    - “Container path”: **`/tutorial`**
  - “Container”→“Exec” tab starts Linux shell



- **Run small SANA-FE simulation**
  - In “Exec” shell run command below:
  - Parses demo inputs & executes simulator kernel for 1000 time-steps

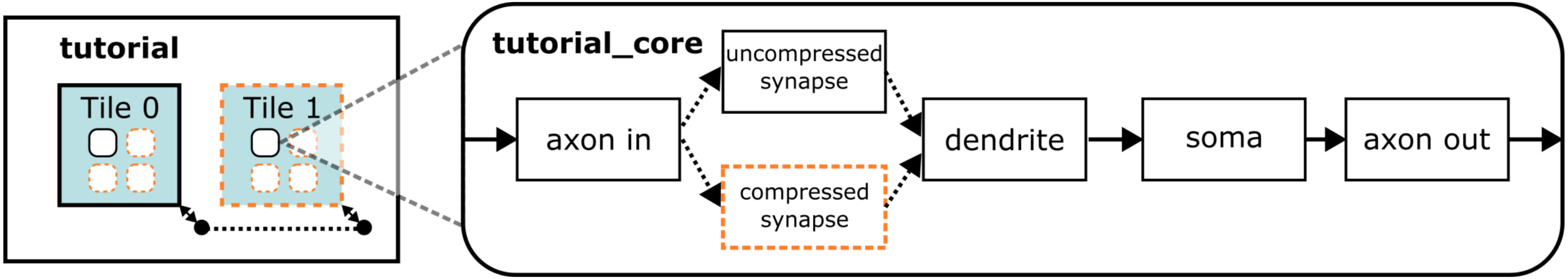
```
# python3 sim.py tutorial/arch.yaml tutorial/snn.net 1000
```

SANA-FE Overview



Architecture Description Example

```
# cat /tutorial/arch.yaml
# diff -I wall run_summary.yaml arch_results
```



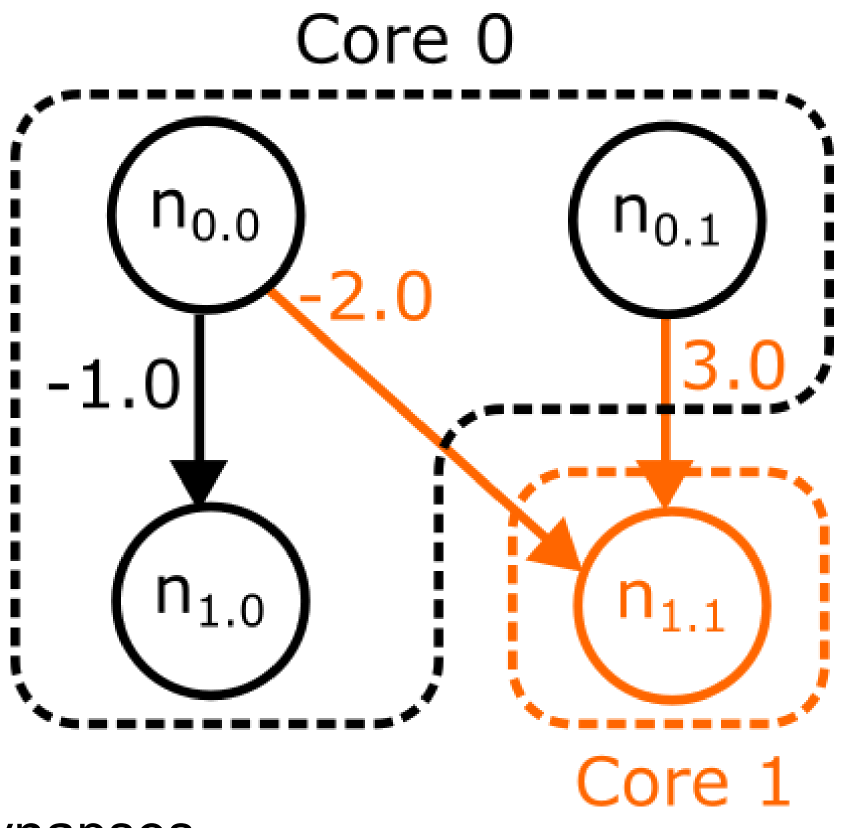
Exercises:

1. Change the cost of updating neurons from 0 ns & 0 pJ to 2 ns & 2 pJ
2. Duplicate tiles twice and cores four times per tile (8 cores total)
3. Add a new synapse unit for compressed synapses. Energy & latency costs of reading compressed synapses are 0.5 pJ and 2 ns respectively

Mapped Spiking Neural Network Example

```
# cat /tutorial/snn.net
# diff -I wall run_summary.yaml snn_results
```

Neuron	Group	Bias	Synapse Type
0.0	0	0.2	-
0.1	0	**0.5**	-
1.0	1	0	**Compressed**
1.1	1	**0**	**Compressed**



Exercises:

1. Define neuron $n_{1.1}$
2. Add edges from neurons $n_{0.0}$ & $n_{0.1}$ to neuron $n_{1.1}$
3. Set the bias of neuron $n_{0.1}$ to 0.5
4. Configure neurons in group 1 to use compressed synapses

Simulator Outputs

```
# python3 sim.py -o tutorial tutorial/arch.yaml tutorial/snn.net 10
# cat tutorial/run_summary.yaml
```

- **SANA-FE run-time summary**
  - Numbers of cores, axons, etc.
  - Total latency, energy & power
  - Results saved to YAML file

- **Optionally enabled traces**
  - Spikes (-s)
  - Neuron potentials (-v)
  - Performance statistics (-p)
  - Spike message packets (-m)

```
/ # cat tutorial/run_summary.yaml
git_version:
energy: 1.160000e-10
sim_time: 9.000000e-08
total_spikes: 5
total_packets: 5
total_neurons_fired: 6
wall_time: 0.000732
timesteps: 10
/ #
```

Neuron Traces

```
# python3 sim.py -s -v -o tutorial tutorial/arch.yaml tutorial/snn.net 10
# cat tutorial/spikes.csv
# cat tutorial/potential.csv
```

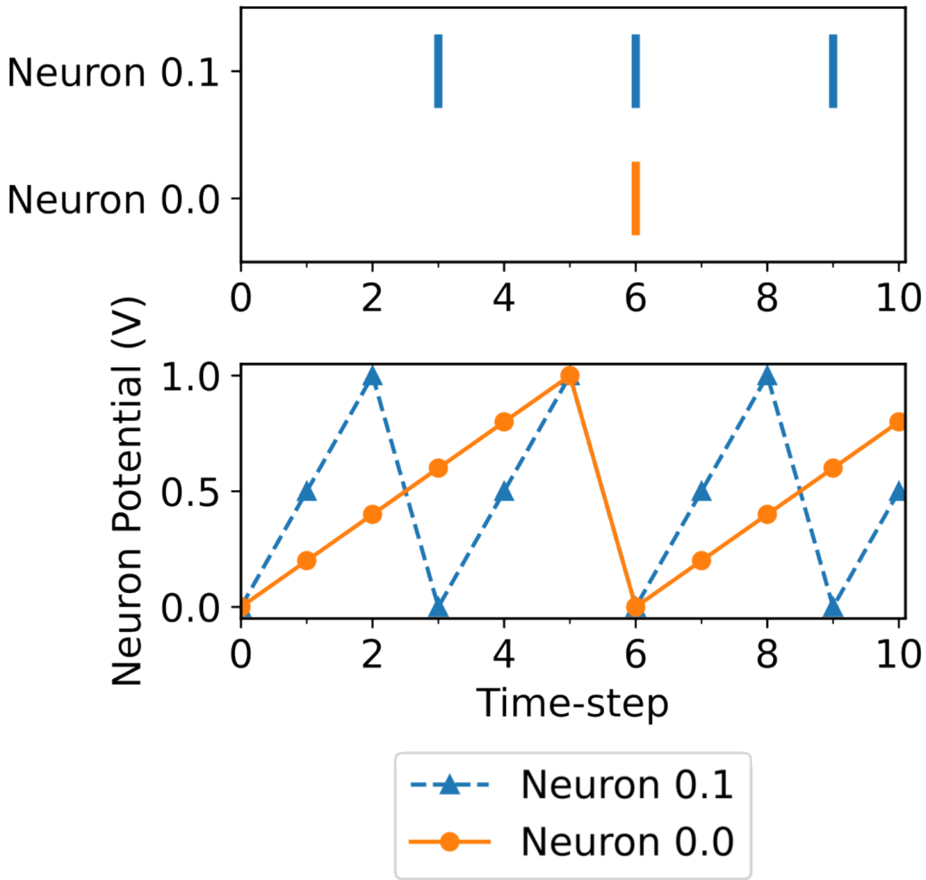
- **Probes select observed neurons**

  `log_spikes` → `spikes.csv`

  `log_potential` → `potential.csv`

- **Spike and voltage traces**
  - Spikes: line per spike event
  - Membrane potentials: line per time-step & column per probe

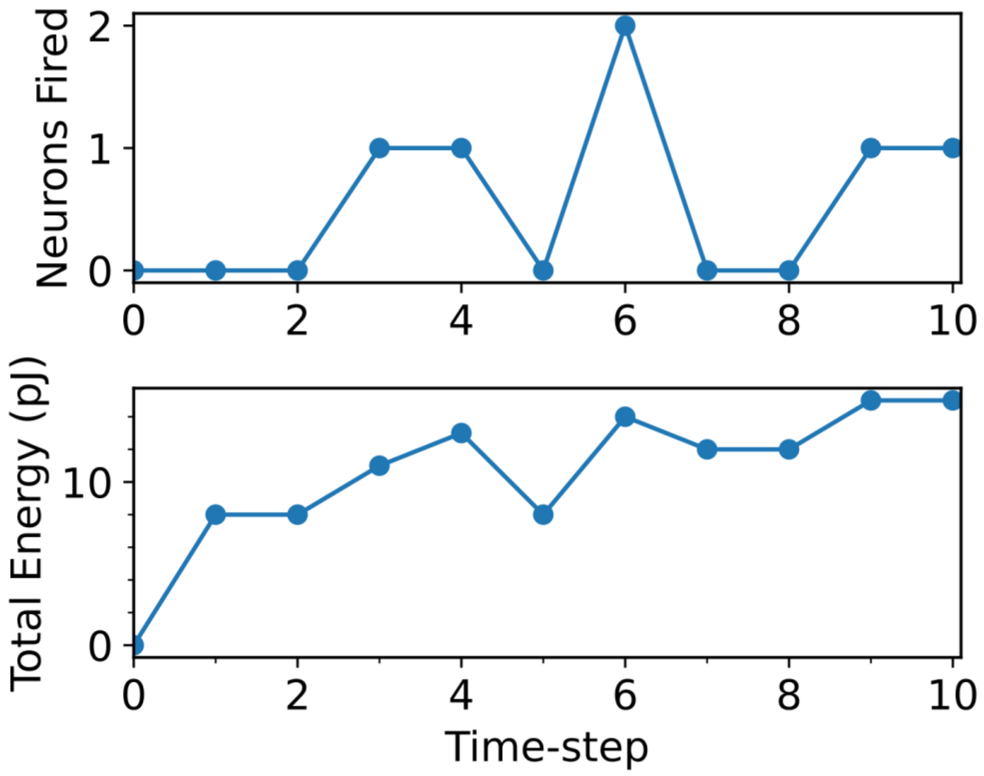
- **Exercise:**
  1. Visualize the neuron membrane potentials



Hardware Traces

```
# python3 sim.py -m -p -o tutorial tutorial/arch.yaml tutorial/snn.net 10
# cat tutorial/perf.csv
# cat tutorial/messages.csv
```

- **Detailed statistics per time-step**
  - H/W performance across entire chip
  - Messages sent over network
- **Performance and message traces**
  - Performance trace: line per time-step
  - Messages: line per spike message



Real-world Application

- **Gesture categorization**
  - Event data from neuromorphic sensor (IBM)
  - Classify hand gestures from 11 gesture types

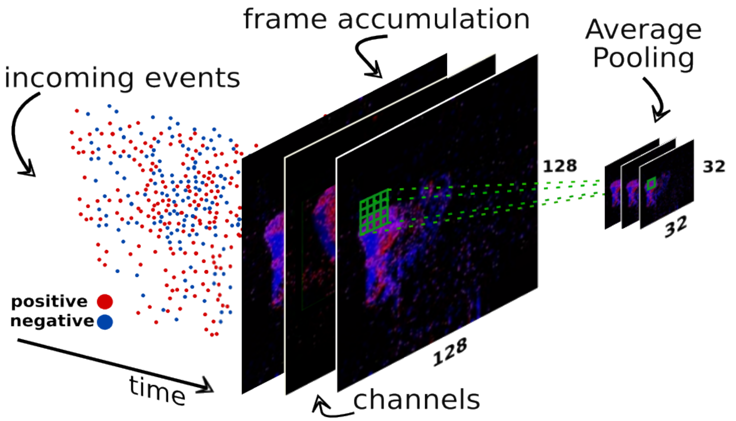
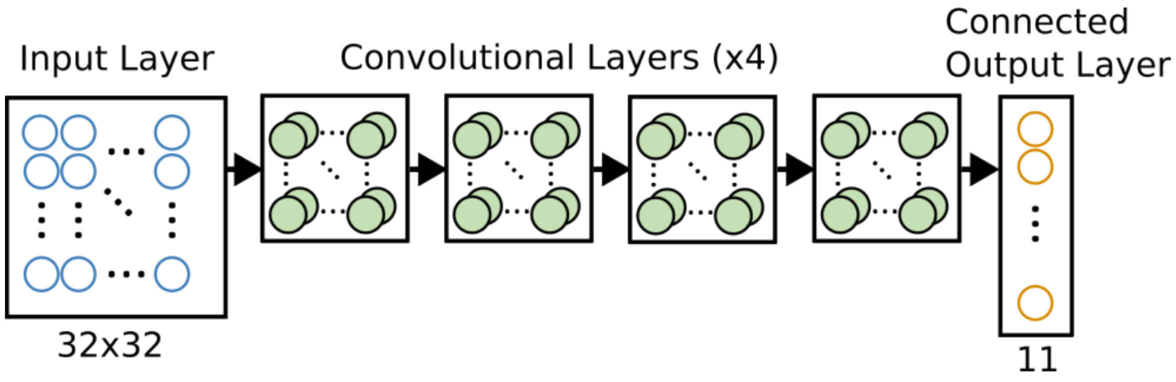


Image reproduced from Massa et al., 2020

- **SNN for gesture categorization**
  - Trained using Keras & SNN Toolbox [Massa '20]
  - SNN has 4 convolutional layers & 1 fully connected layer

- **Categorization on Intel's Loihi**
  - SNN compiled using NxTF
  - Frames presented for 128 time-steps



Automating SANA-FE

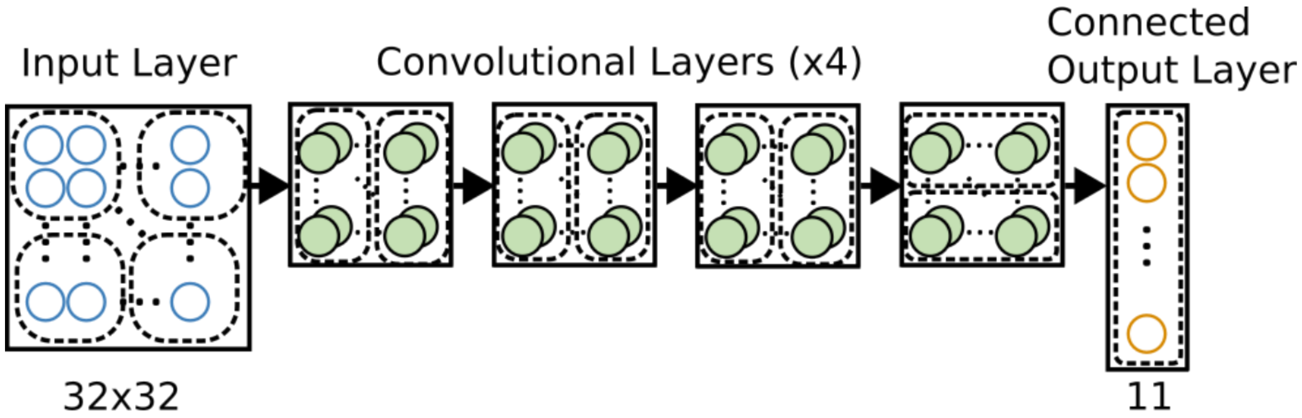
- **SANA-FE scripting capabilities**
  - Automates architecture parsing, SNN generation & runs
  - Library for defining neurons, groups & SNN layers
  - Enables design-space exploration
- **Script to run gesture application**
  - Generates SNN from kernel weights
  - Maps SNN to H/W cores
  - Runs simulation & parses results

```
[main.c:228:main()] Running simulation.
[main.c:235:main()] *** Time-step 100 ***
[main.c:235:main()] *** Time-step 200 ***
[main.c:235:main()] *** Time-step 300 ***
[main.c:235:main()] *** Time-step 400 ***
[main.c:235:main()] *** Time-step 500 ***
[main.c:235:main()] *** Time-step 600 ***
[main.c:235:main()] *** Time-step 700 ***
[main.c:235:main()] *** Time-step 800 ***
[main.c:235:main()] *** Time-step 900 ***
[main.c:235:main()] *** Time-step 1000 ***
[main.c:240:main()] ***** Run Summary *****
git_version:
energy: 3.451703e-03
sim_time: 2.659117e-02
total_spikes: 51830490
total_packets: 2495985
total_neurons_fired: 367770
wall_time: 22.517683
timesteps: 1000
[main.c:250:main()] Average power consumption: 0.129806 W.
[main.c:259:main()] Run finished.
Energy-Delay product: 9.178482126251e-05
/ #
```

```
# python3 tutorial/dvs_challenge.py
```

Gesture Mapping Challenge

- **Optimize SNN H/W mapping**
  - Using DVS gesture application
  - Same SNN can be mapped to different H/W cores
  - Update H/W mapping in **`dvs_challenge.py`**



- **Best mapping wins**
  - Optimize for smallest energy-delay product (Total Energy × Total Run-time)
  - Valid mappings only
    - Simulation must run & post-run checks pass
    - Maximum 1024 neurons per core

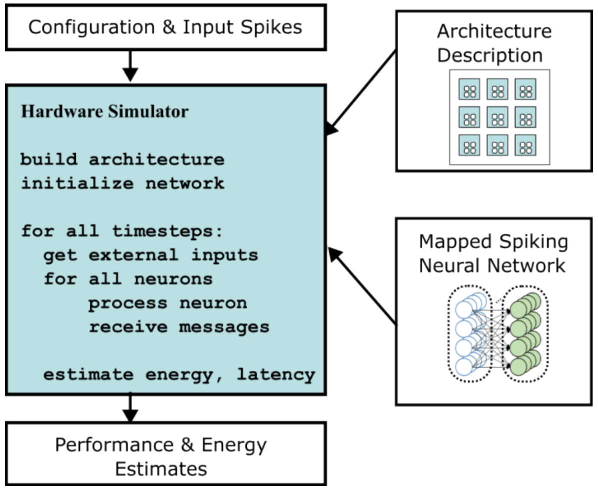
- **Submit results of best run to: james.boyle@utexas.edu**

Upcoming Features

- **Move from C to C++**
  - Base hardware classes provided
  - PyBind11 interface with Python
- **Support for neuromorphic ecosystem**
  - Fugu & Lava integration
  - User plug-ins & custom neurons models
- **Support new components**
  - Mixed-signal architectures & novel devices
  - Dendritic computing

SANA-FE

- **Generic & extensible**
  - User-defined architecture & SNN
  - Supports range of spiking architectures
- **Fast & accurate**
  - Time-step based approach
  - Detailed hardware activity for each time-step
  - Accurately estimates performance & energy
- **Future work**
  - Support other existing architectures & scale to larger designs
  - Adapt other neuromorphic benchmark applications
  - Model analog architectures & novel devices
  - Integrate with other frameworks e.g., SST, Fugu & La va



Access at: https://github.com/SLAM-Lab/SANA-FE