A Software Stack for Neuromorphic Computing

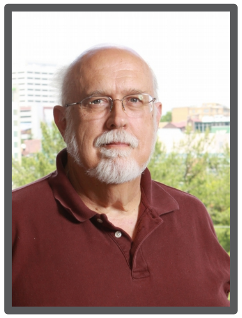
THE UNIVERSITY OF TENNESSEE KNOXVILLE

James S. Plank
Mark E. Dean
Garrett S. Rose
Catherine D. Schuman

OAK RIDGE National Laboratory

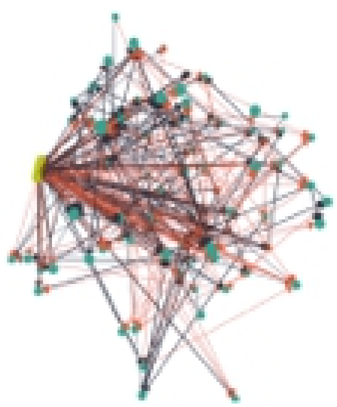
July 19, 2017
Neuromorphic Computing Symposium
Knoxville, Tennessee

What our group looked like in 5/2015


Katie and Doug developed NIDA:

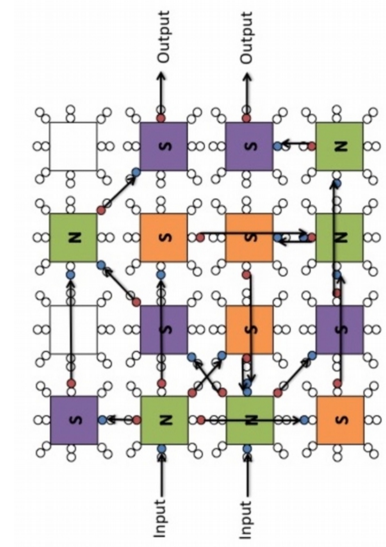
- Simulator
- Custom applications
- Custom EO
- Custom visualization (Meg)



Mark developed DANNA:

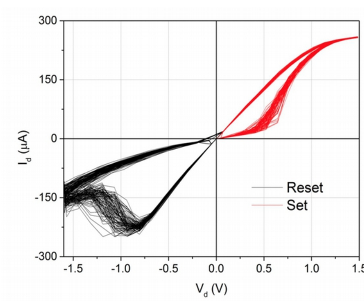
- FPGA Implementation (Chris)
- Hand-tooled networks
- Communications board (Jason)



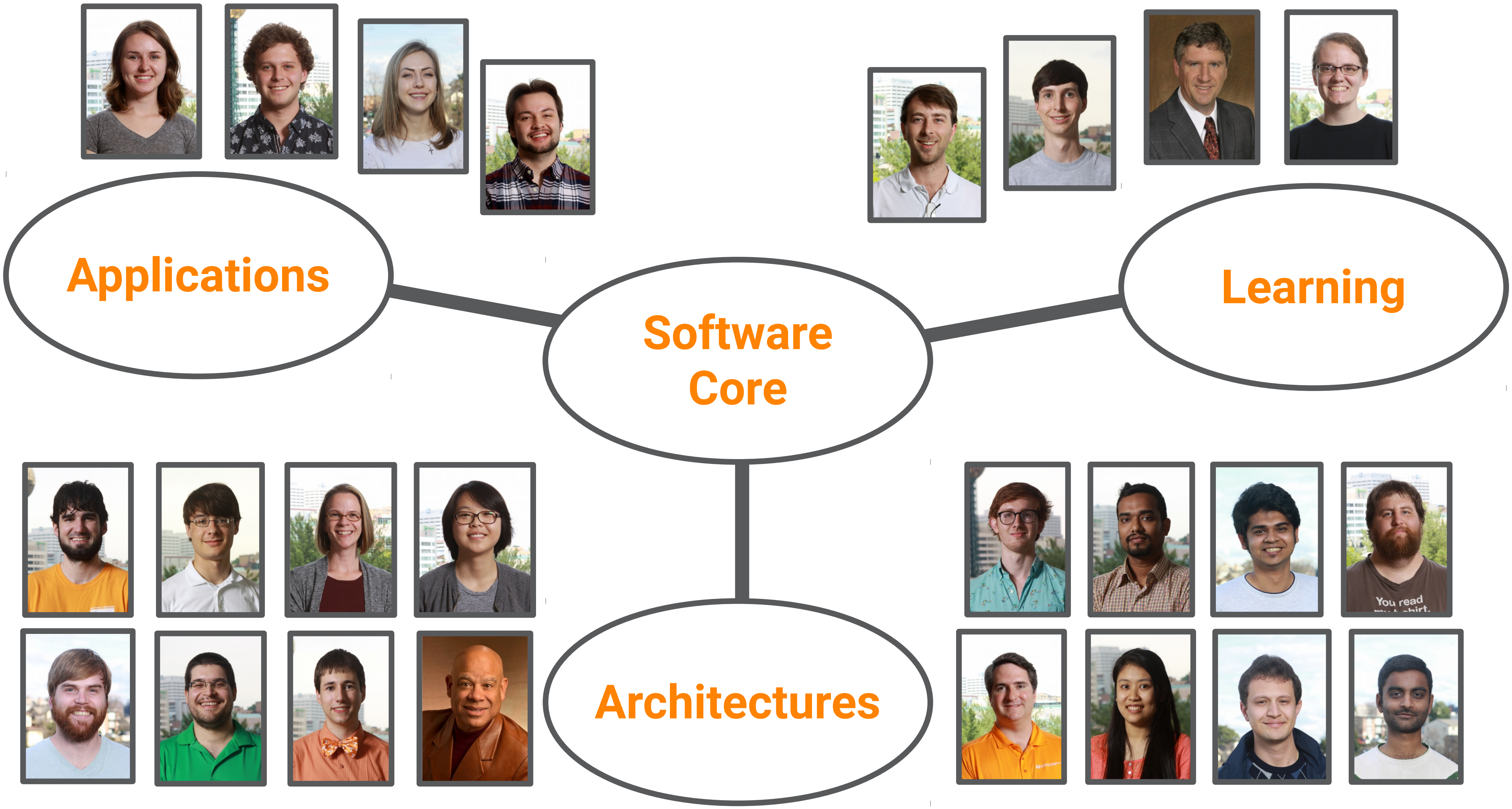


Garrett developed mrDANNA:

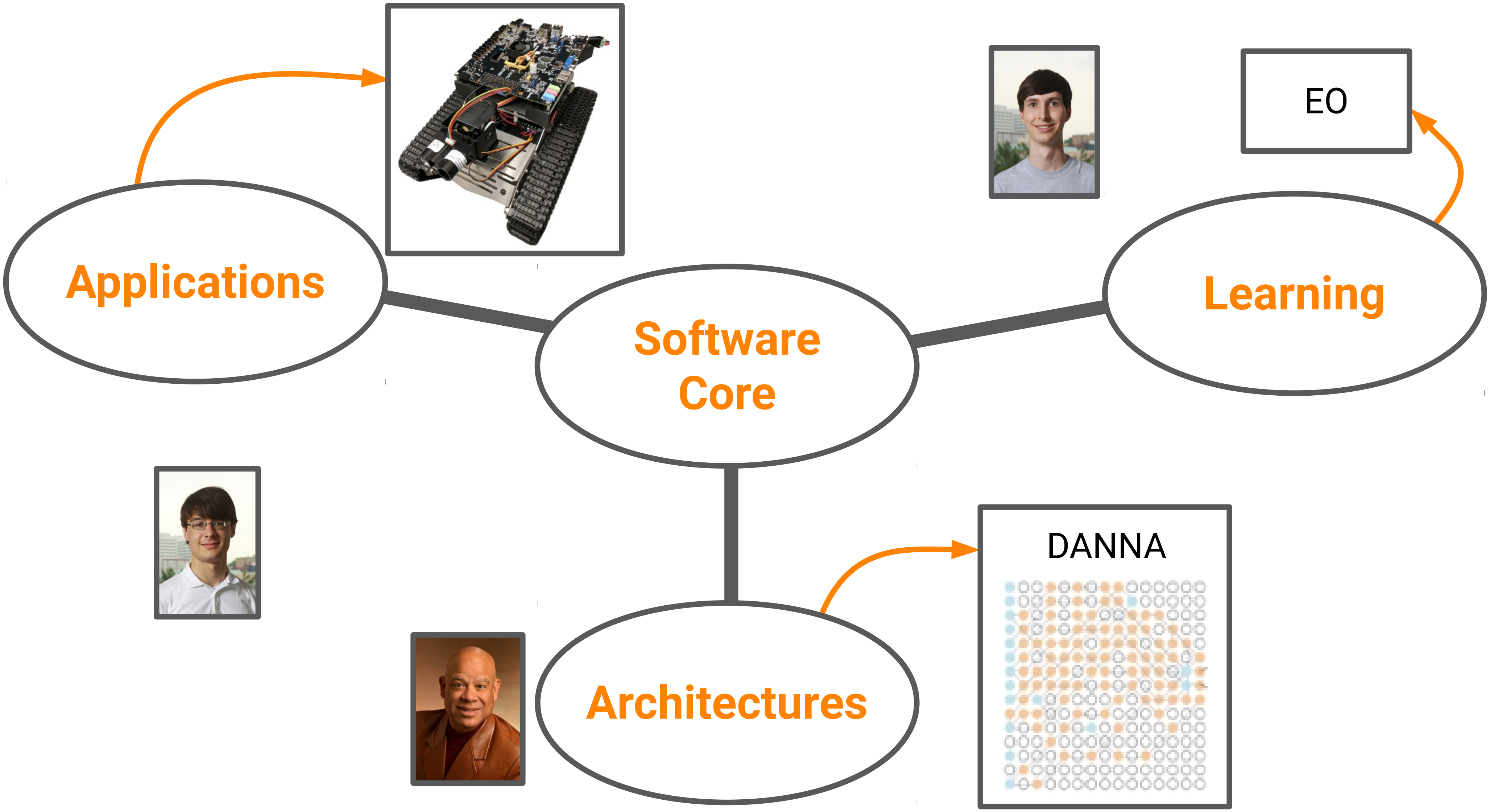
- Memristor modeling
- Simulator in SPICE (Gangotree)
- Hand-tooled networks

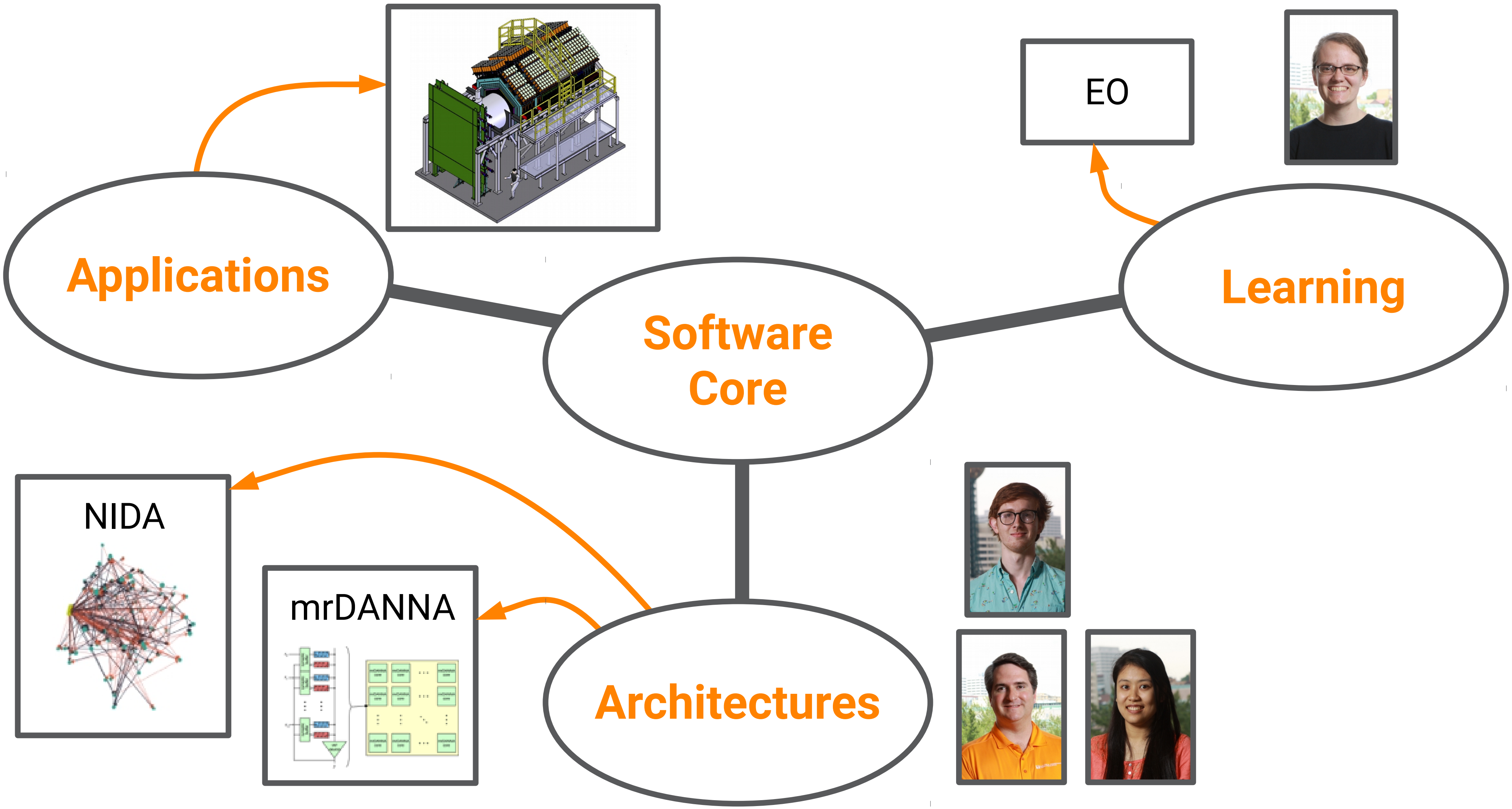
What it looks like now



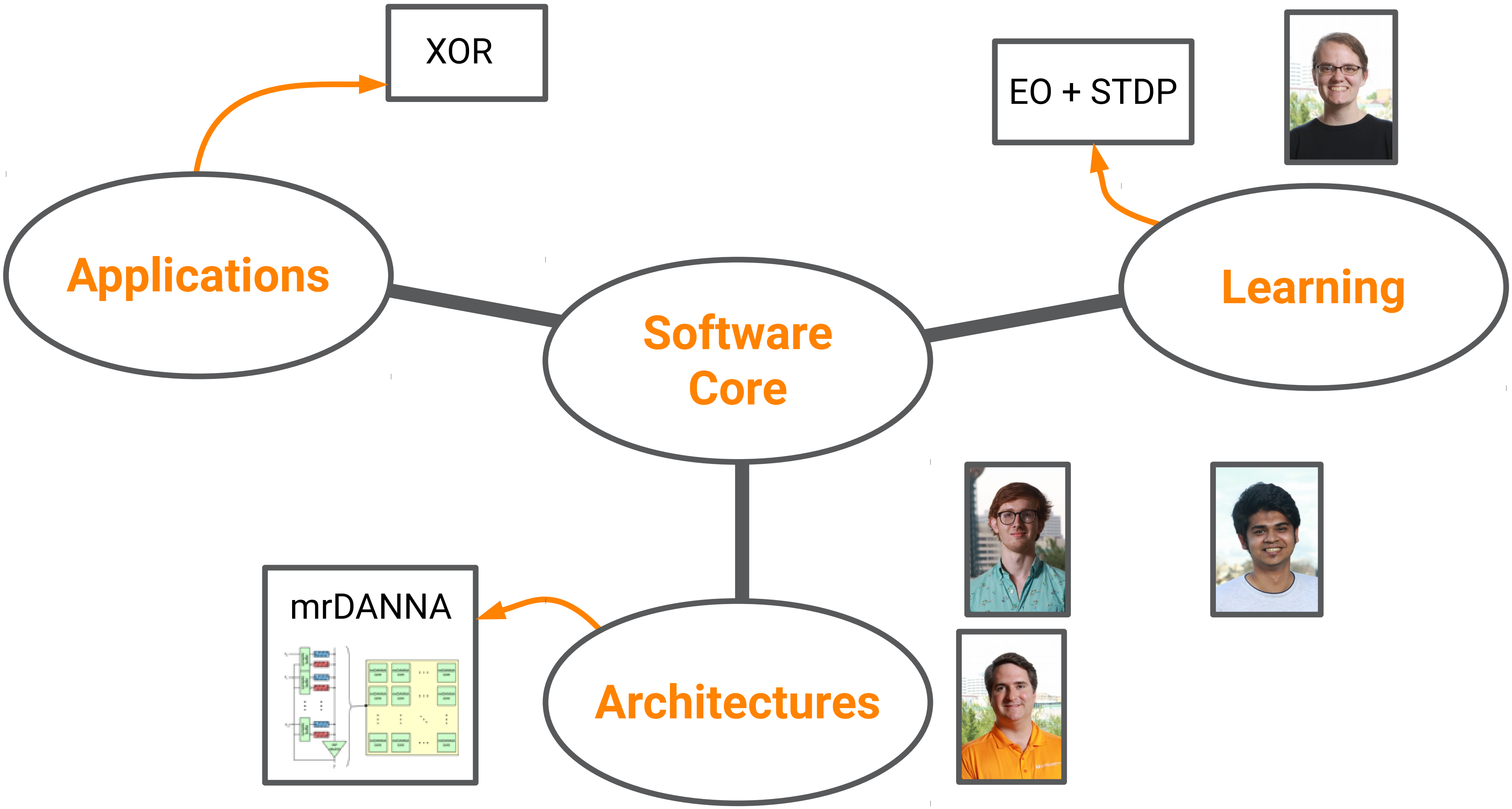
You've seen it in three of our talks here



You've seen it in three of our talks here



You've seen it in three of our talks here



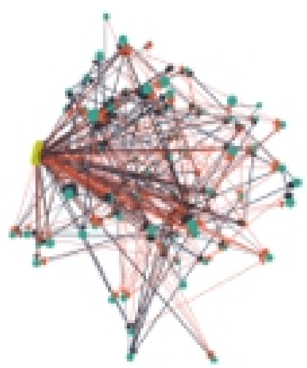
What is in this talk

- What an architecture means in our software stack.
- The structure of an application in this stack.
- How to put “learning” into its appropriate place.
- Some lessons learned with respect to software and a project of the scope of this one.

What an architecture means

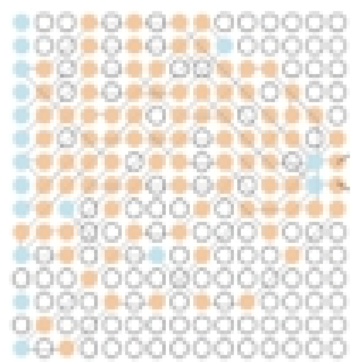
The architecture encompasses the computing model, constraints and connectivity.

NIDA



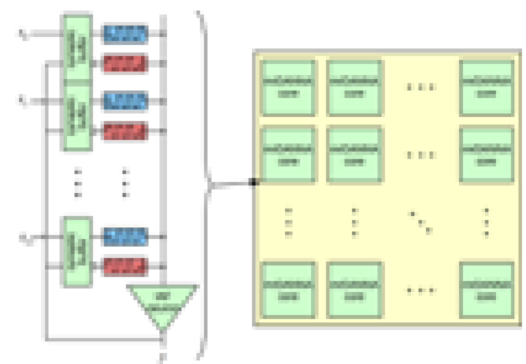
- 3D
- Analog
- Synapses defined by Euclidean distance.

DANNA



- 2D
- Digital
- Synapses programmable but constrained.

mrDANNA



- 2D
- Mixed Analog/Digital
- Synapses programmable but constrained.

What an architecture means

Within the software stack, the architecture must define a network and a device.

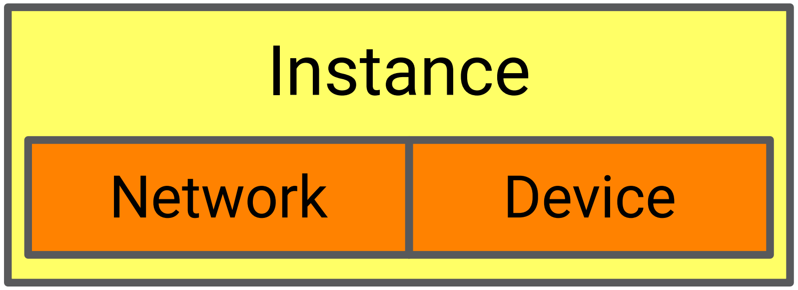
Network ≈ "Program"

- Serialize / Deserialize
- Define inputs & outputs
- Primitives for learning (more on this later)

Device ≈ "Processor"

- Load / Pull Network
- Apply input charge events
- Read output charge events
- Run
- Capture State

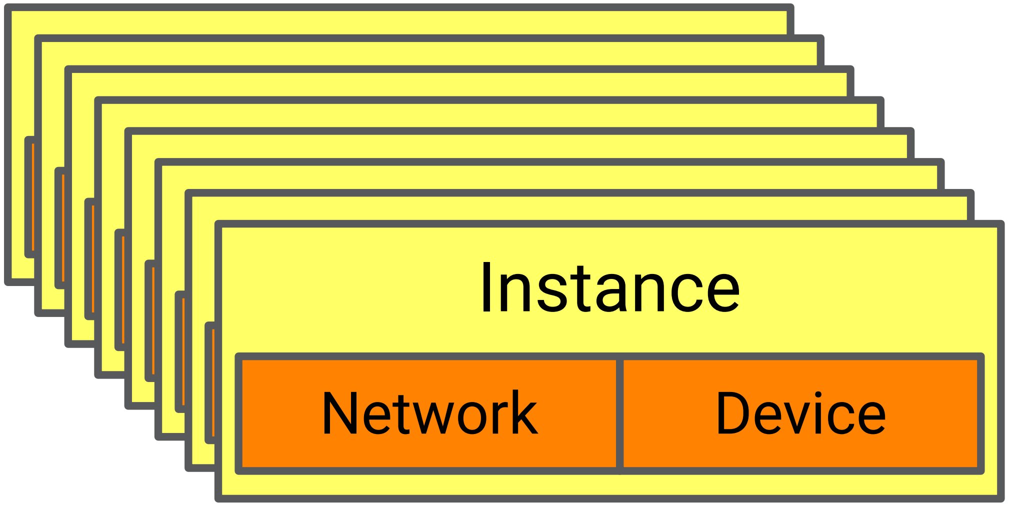
Within the core, an instance drives execution.



- Start job
- Execute
- Stop job

- Why do we need this?

Within the core, an instance drives execution.

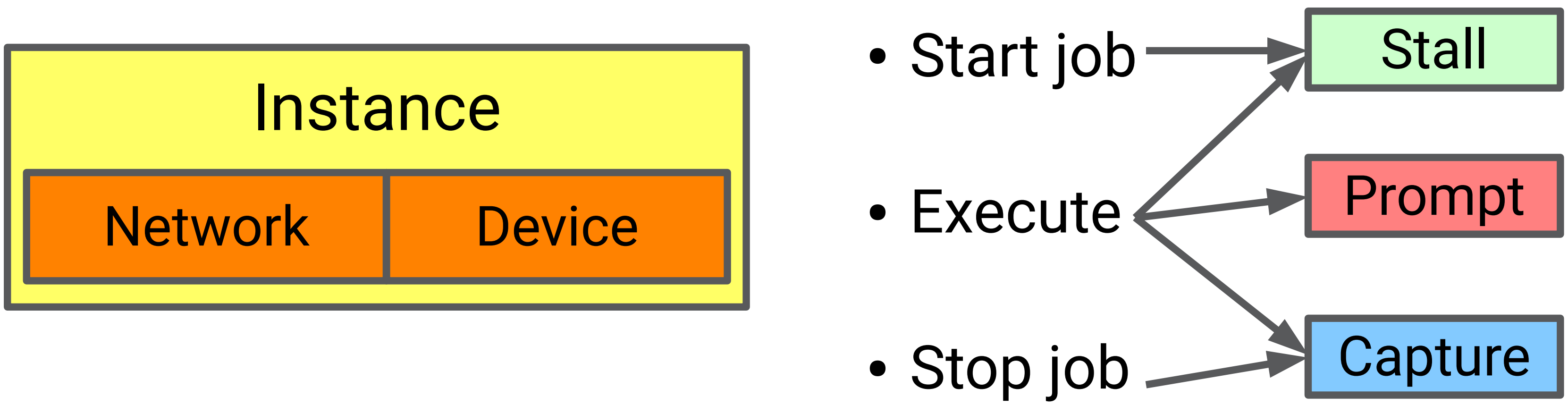


- Start job
- Execute
- Stop job

Gives you a handle on an execution

- EO / GPU's / Advanced applications

Within the core, an instance drives execution.

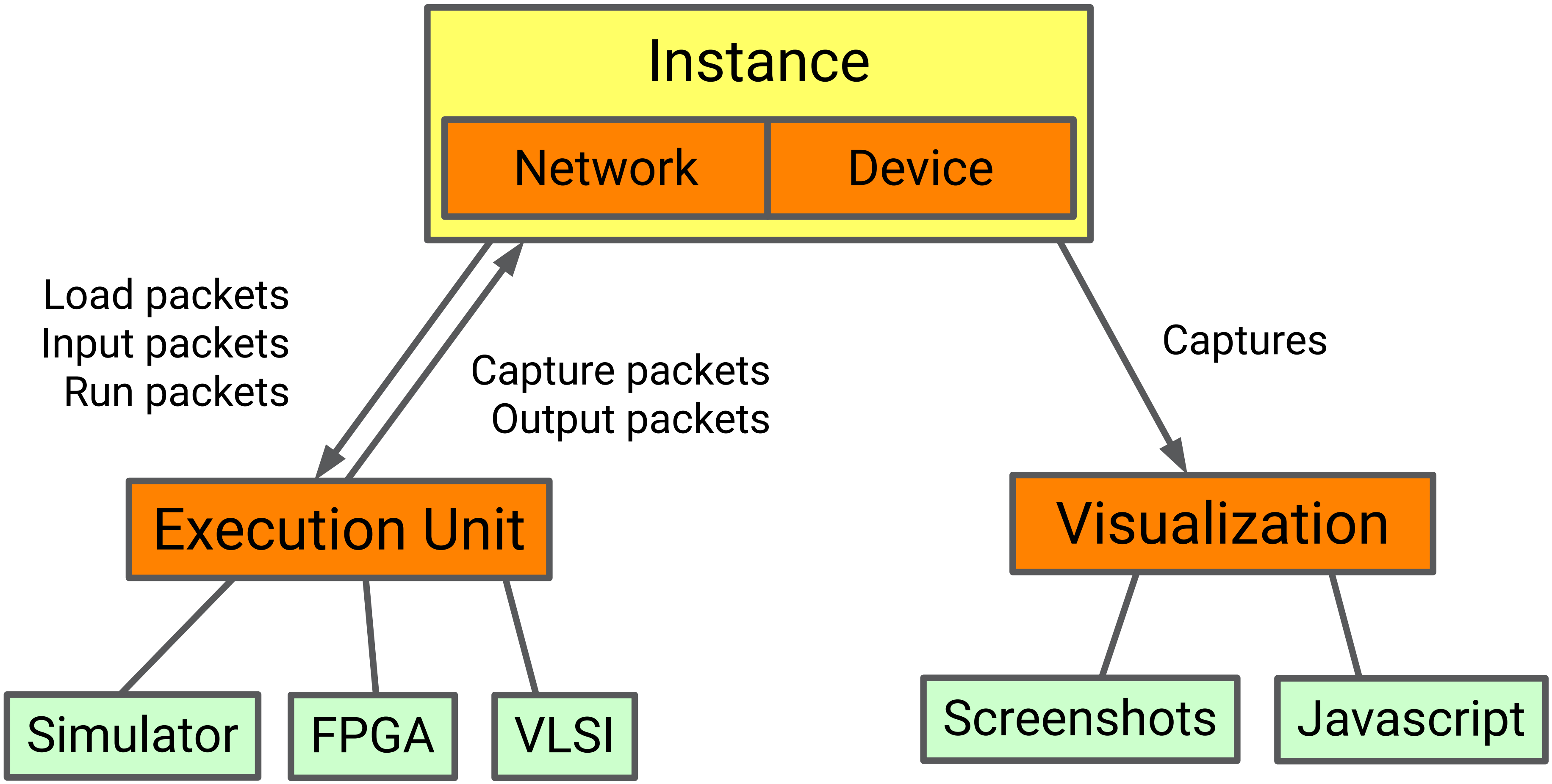


Allows the core to implement architecture-independent functionality.

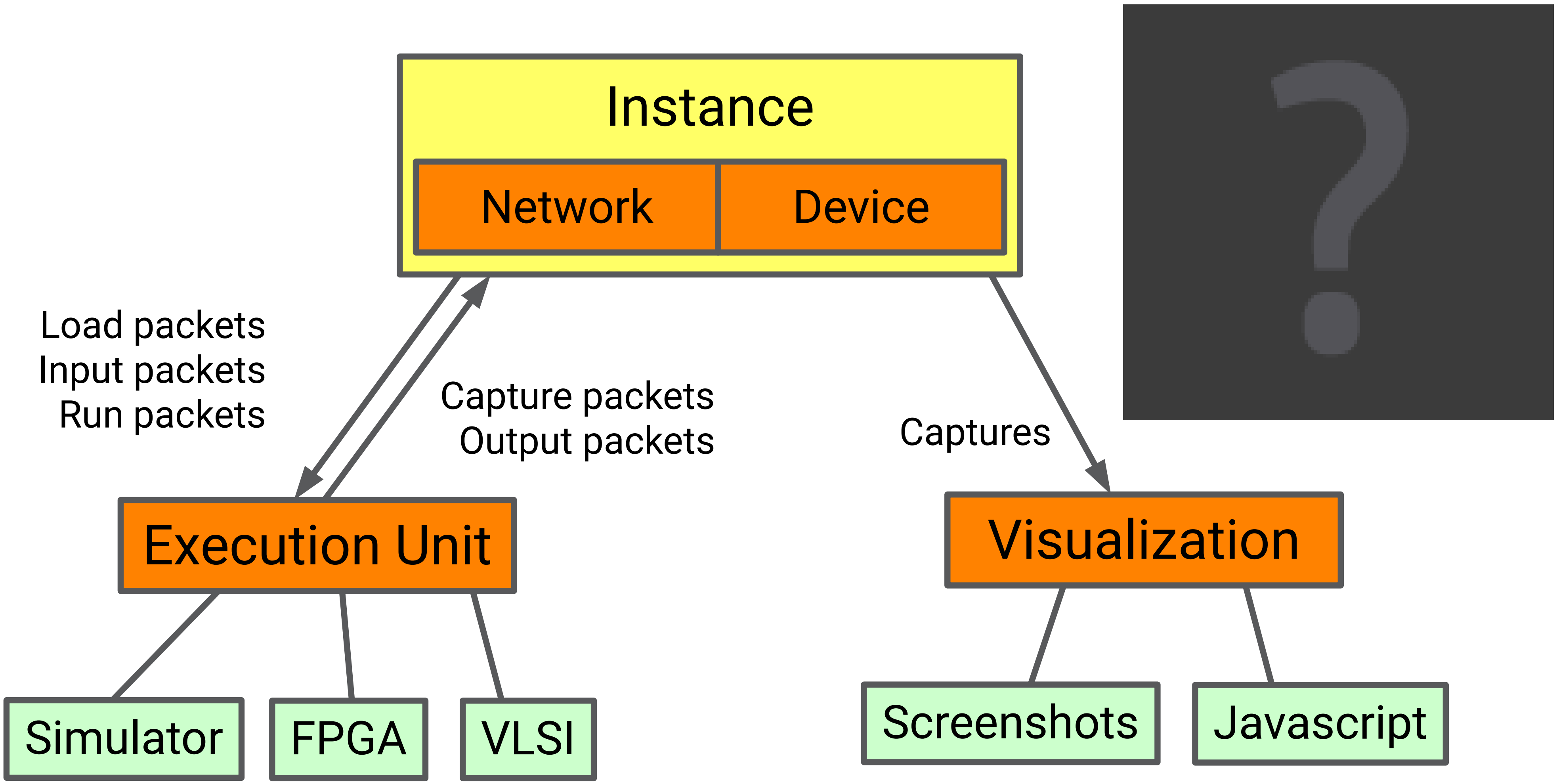
Architectures end up with four components

Network Program	**Device** Processor
Execution Unit Simulation, Hardware	**Visualization** Processes events & captures Static (screenshots) Live

For example, with DANNA



For example, with DANNA



The structure of an application.

Our canonical application structure has 5 components:

Libraries:

Per Run
Application Library
Read-Only

Per Run
Neuromorphic Library
Read-Only

Per Run
Learning Library
Read-Only

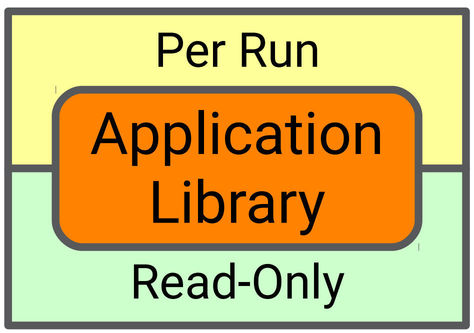
Programs:

Application Driver

Neuromorphic Driver

The structure of an application.

The application library implements the guts of the application.

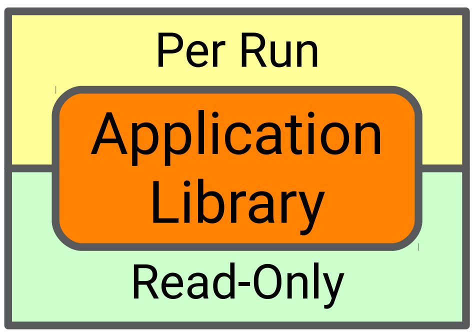


GetApplicationState()

UpdateApplicationState()

The structure of an application.

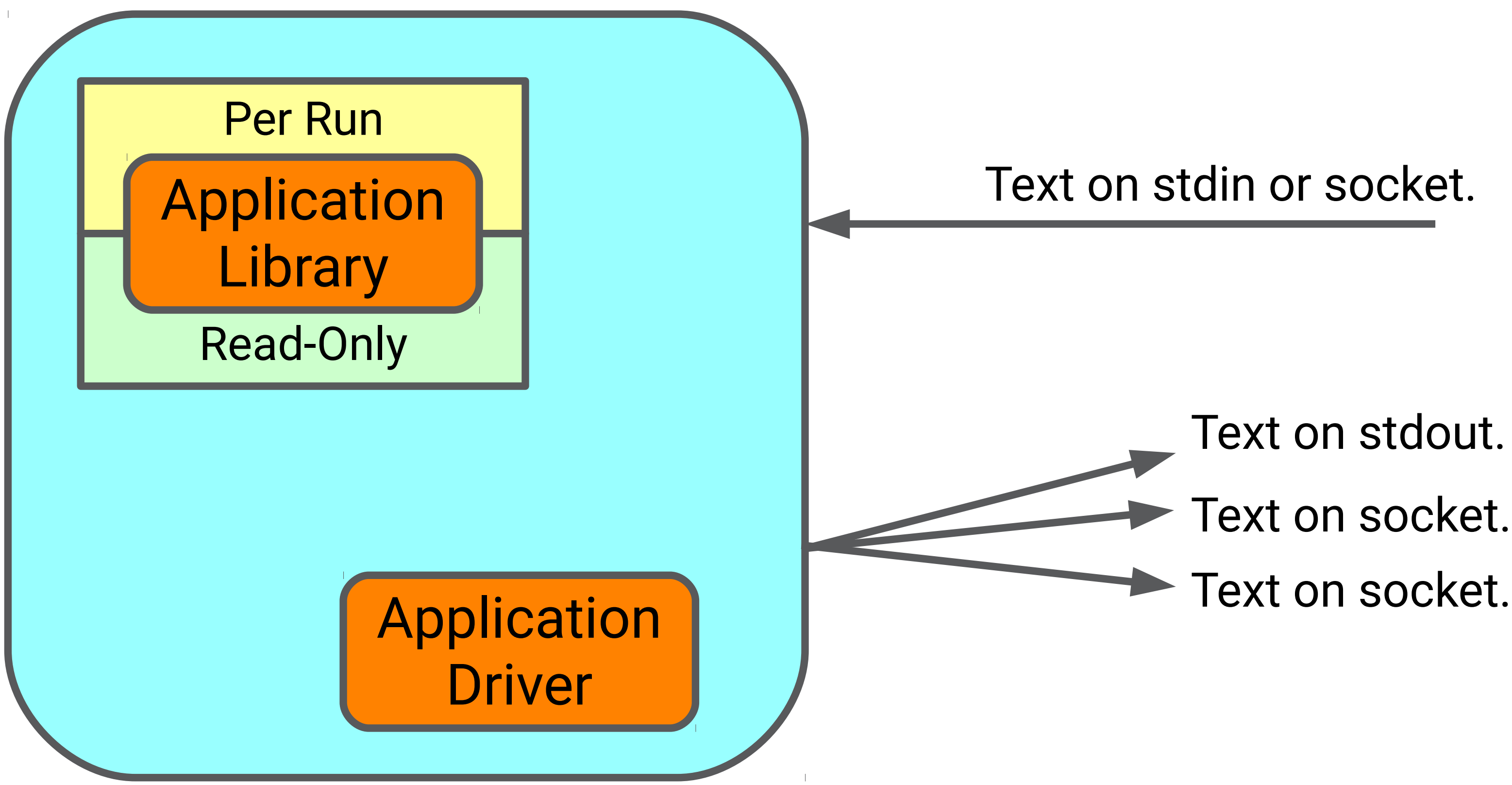
The application library implements the guts of the application.



GetApplicationState()
UpdateApplicationState()

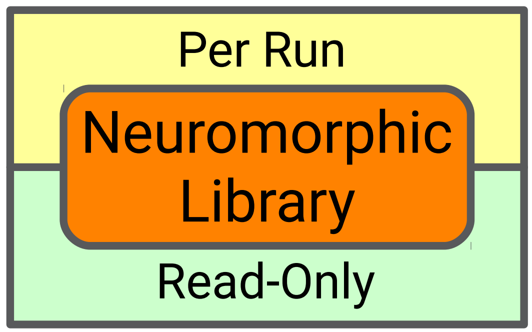
The structure of an application.

Application Program exists without anything neuromorphic.



The structure of an application.

The neuromorphic library implements
instance → application and back

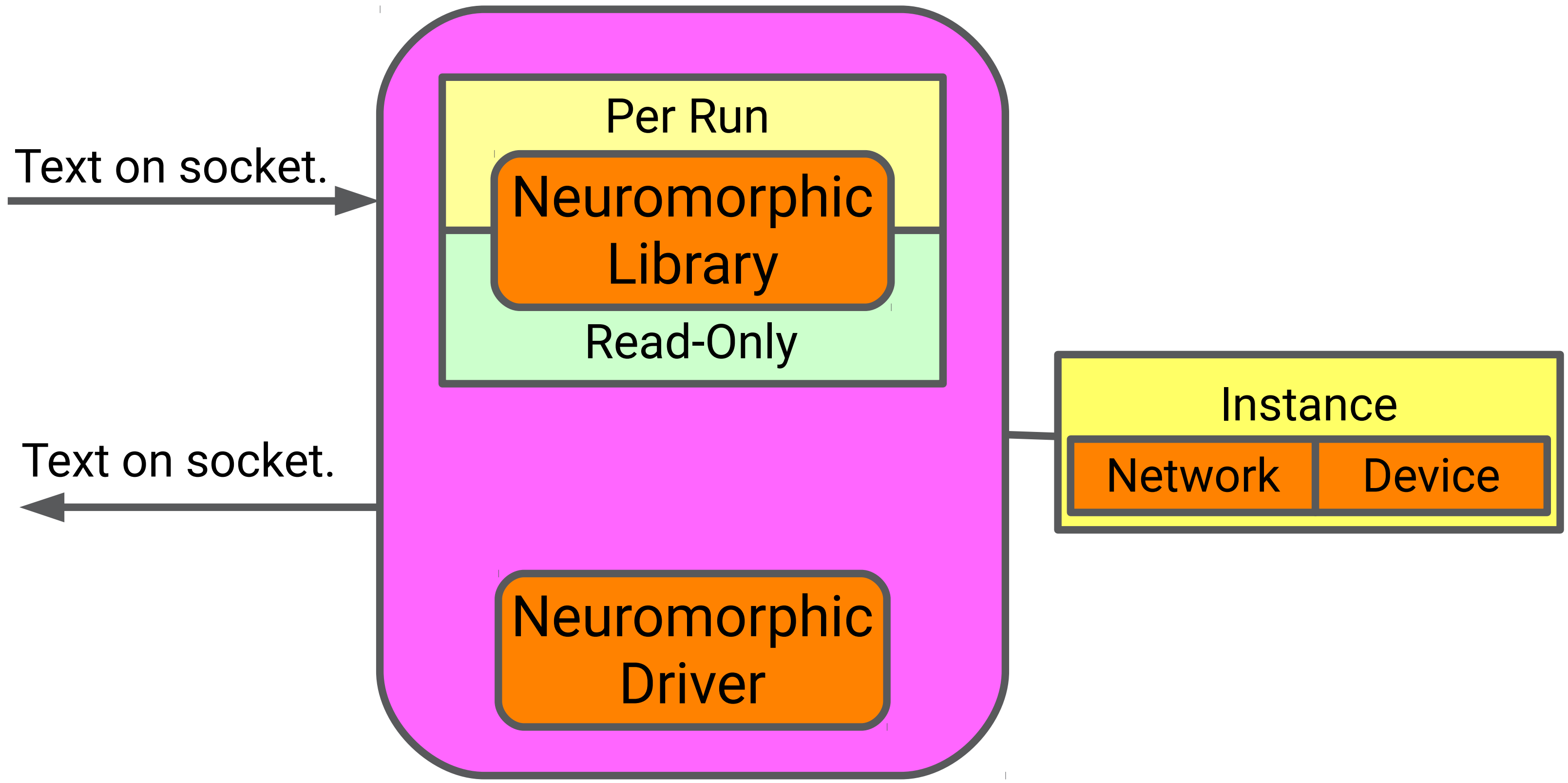


AppState_To_Inputs()

Outputs_To_AppInput()

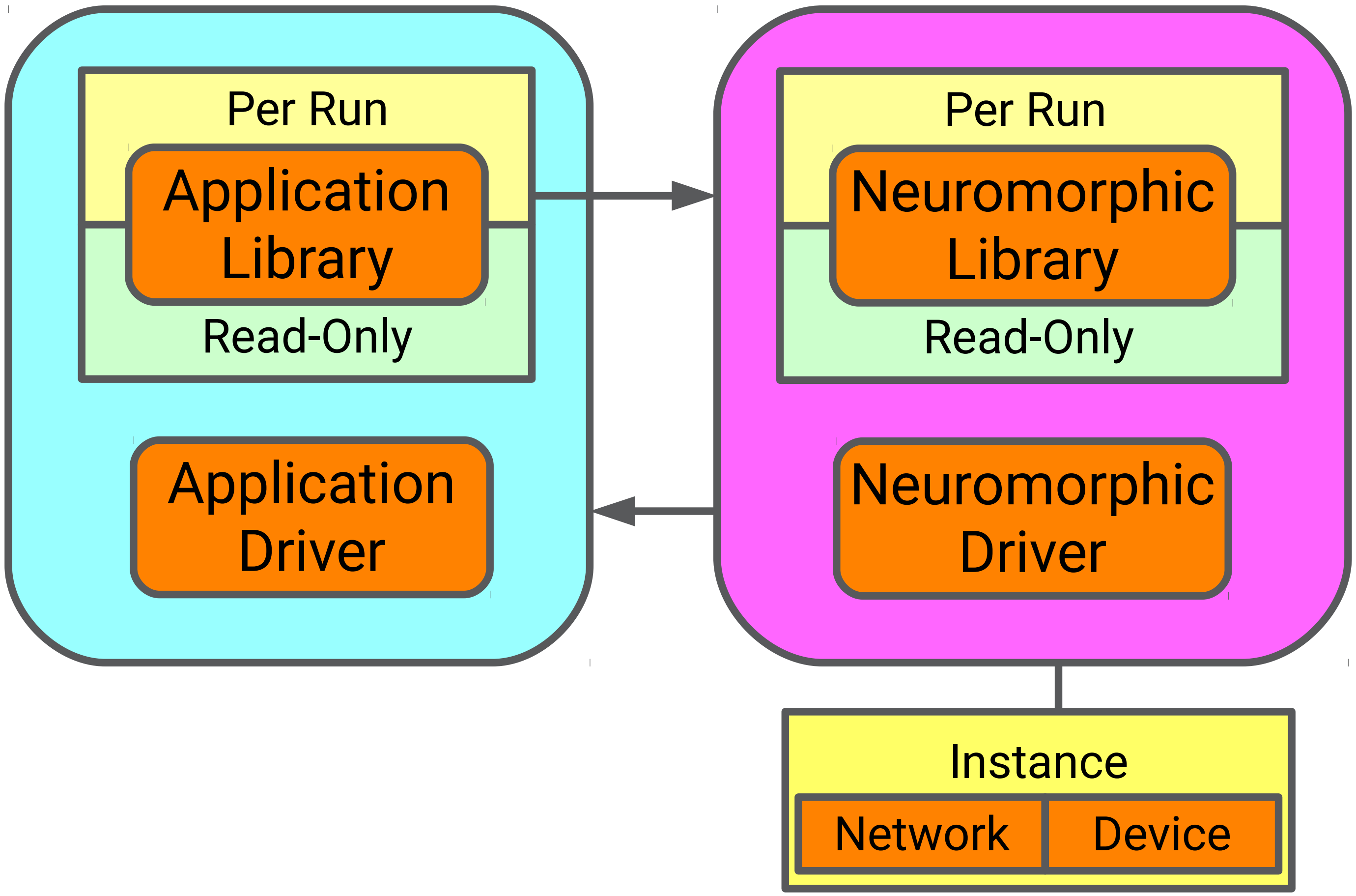
The structure of an application.

Neuromorphic Program interacts with application over sockets, and “runs” an instance.



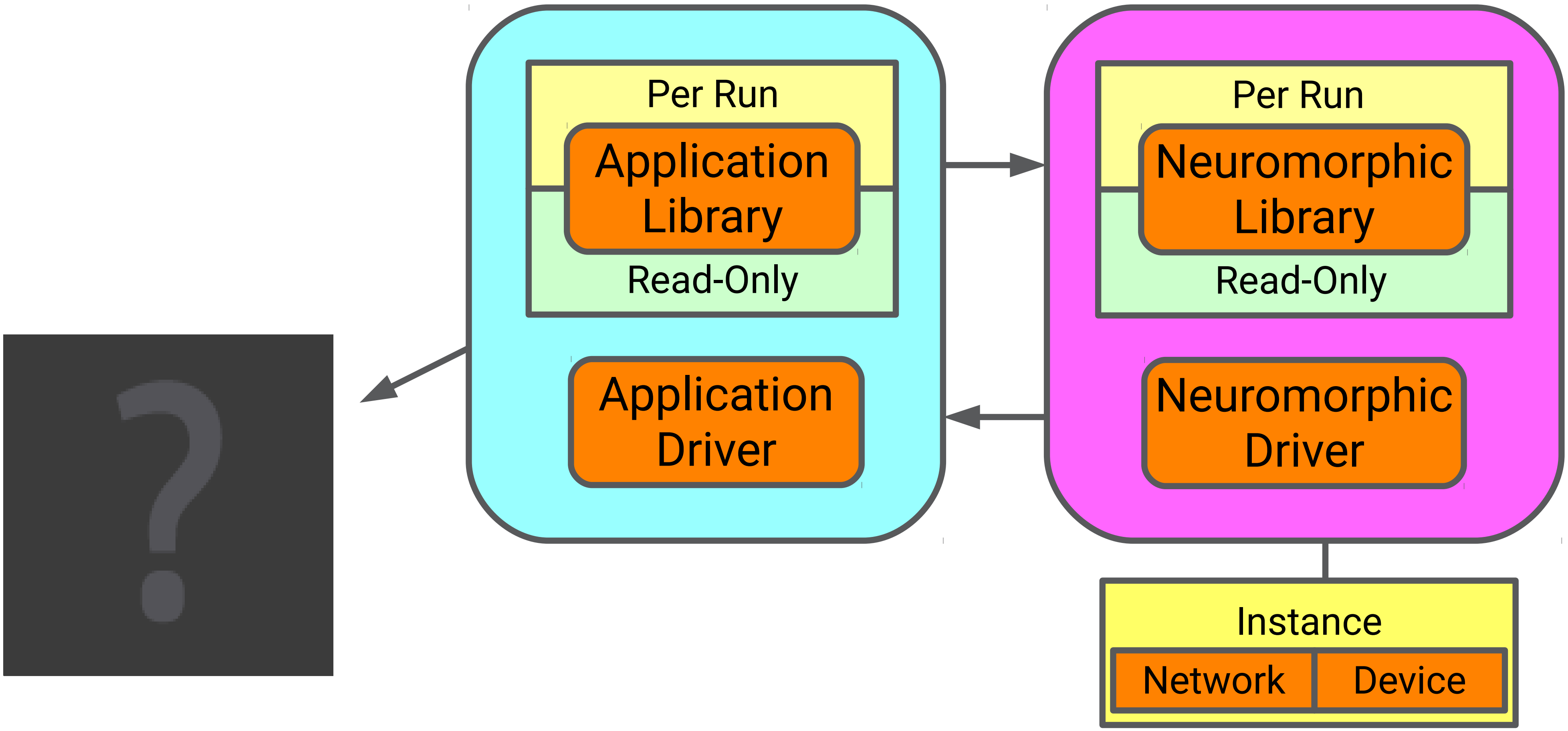
The structure of an application.

Application program and neuromorphic program compose very nicely for testing and demonstration.



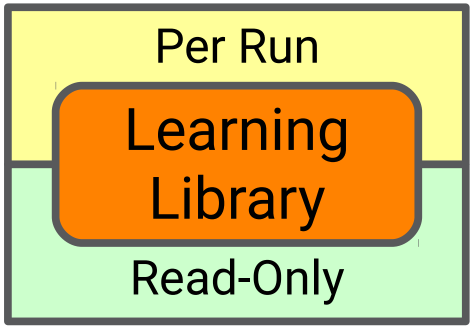
The structure of an application.

Application program and neuromorphic program compose very nicely for testing and demonstration.



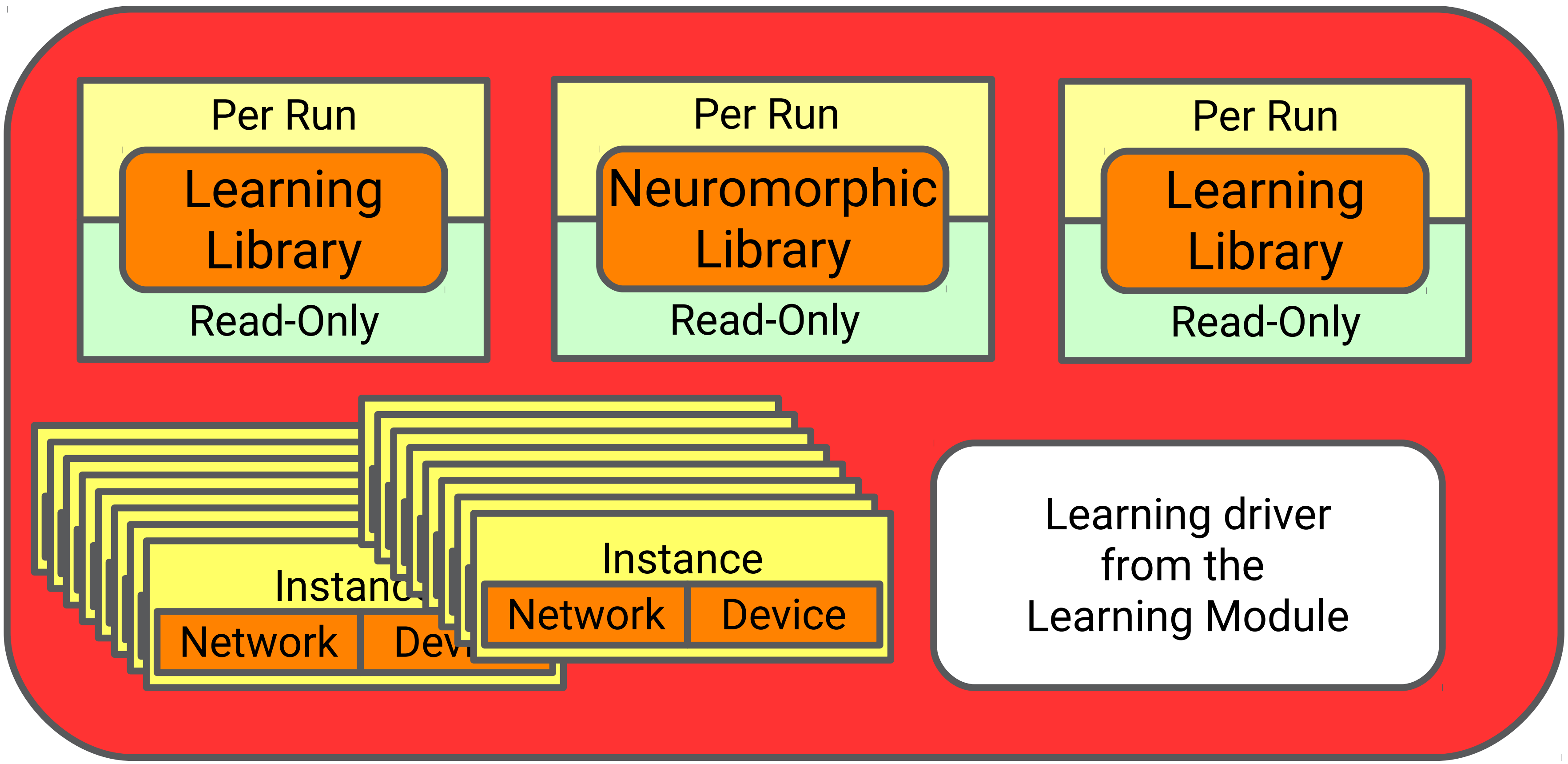
The structure of an application.

Learning library expresses application needs to the learning layer, and defines fitness: instance → value.



The structure of an application.

All of the libraries are compiled with a driver from the Learning Module to develop networks.



Current Applications

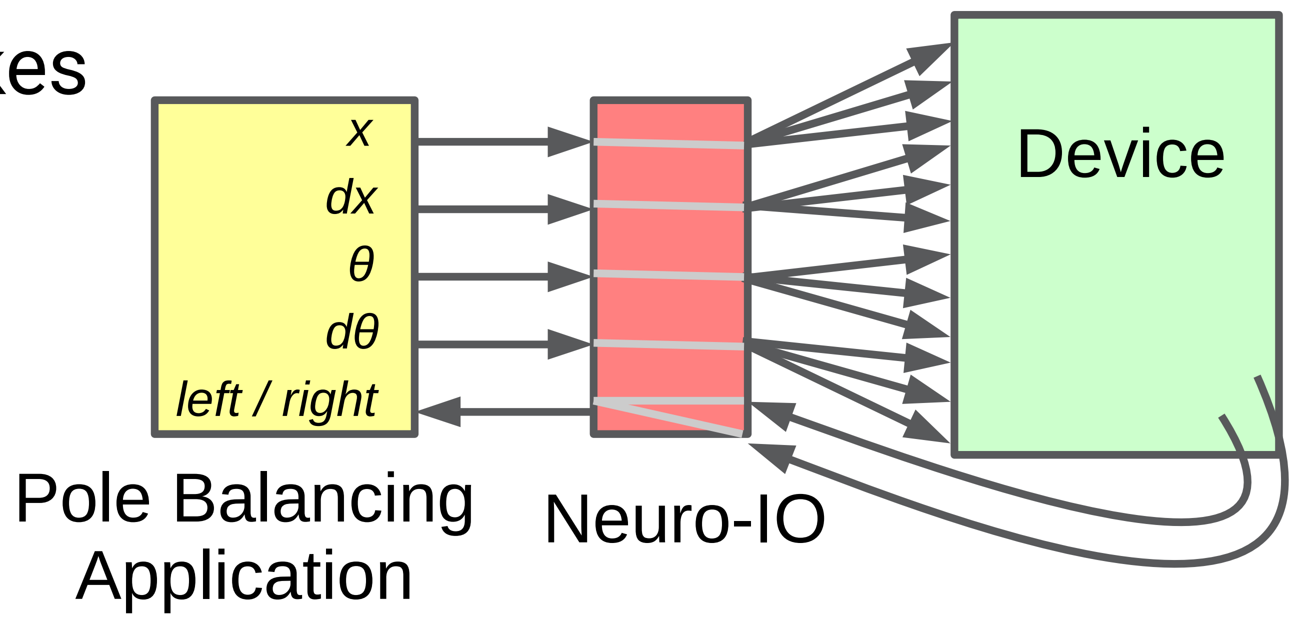
- Control
  - Pole, Flappy, RoboNAV, Helicopter, FF-SA
- Classification
  - UCI Database (Iris, Cancer, etc.), Audio
- Security
  - Anomaly Detection (e.g. Numenta)
- Microapplications: Benchmarking & Composition
  - Binary Ops, Pulse Comparison

RoboNAV on DANNA

Pole Balancer on NIDA

Application Support: Neuro-IO

- Map application state values to neuromorphic input spikes:
  - Rate-Coding, Binning, Charge Values
  - And their combination.
- Ditto output spikes
  - Counting
  - Voting
  - Binning



Learning – Where does it go?

- Current learning techniques:
  - EO: Evolutionary Optimization
  - Unsupervised Learning (STDP)
  - Supervised Learning (Ditto)

Still in "research" mode

Mature enough to be a module

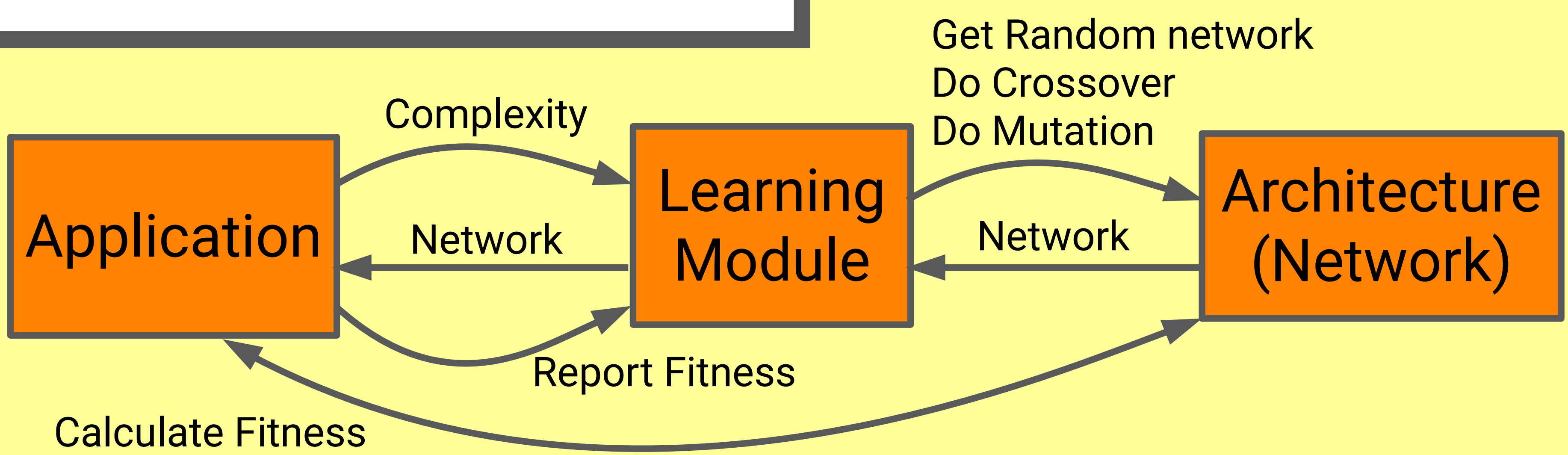
TENN LAB
NEUROMORPHIC
ARCHITECTURES. LEARNING. APPLICATIONS.

Learning – Where does it go?

- The Current Learning Module
  - Manages epochs & populations
  - Directs crossover & mutations, *but doesn't do them.*
  - Manages parallelism, both within a machine and within a cluster (or Titan).

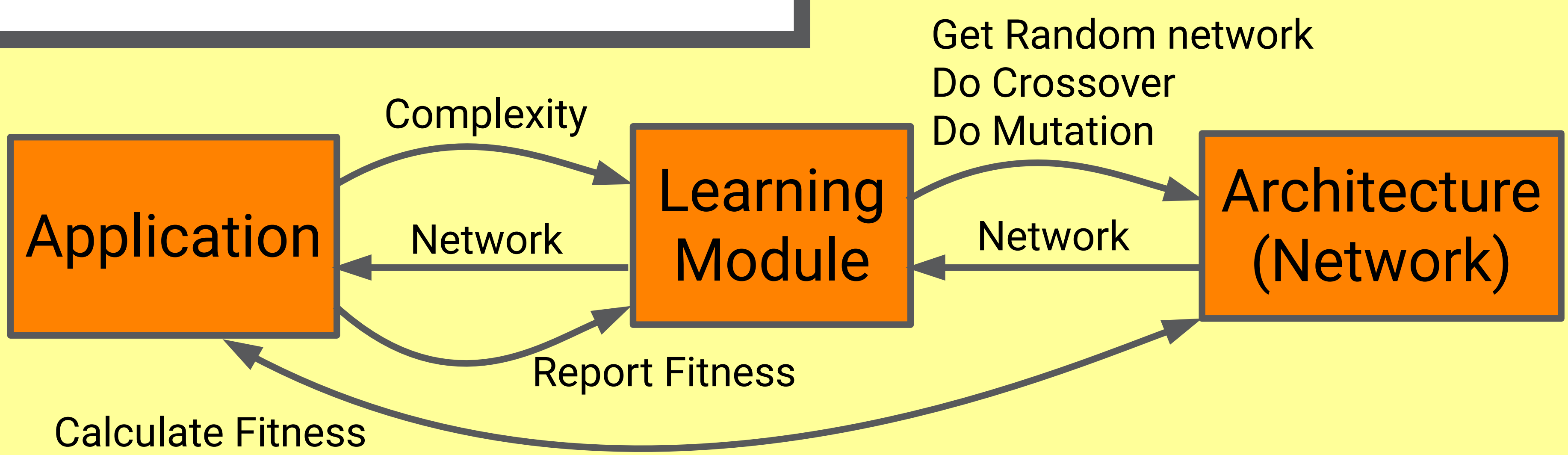
Learning – Where does it go?

- The Current Learning Module
  - Manages epochs & populations
  - Directs crossover & mutations, *but doesn't do them.*
  - Manages parallelism, both within a machine and within a cluster (or Titan).



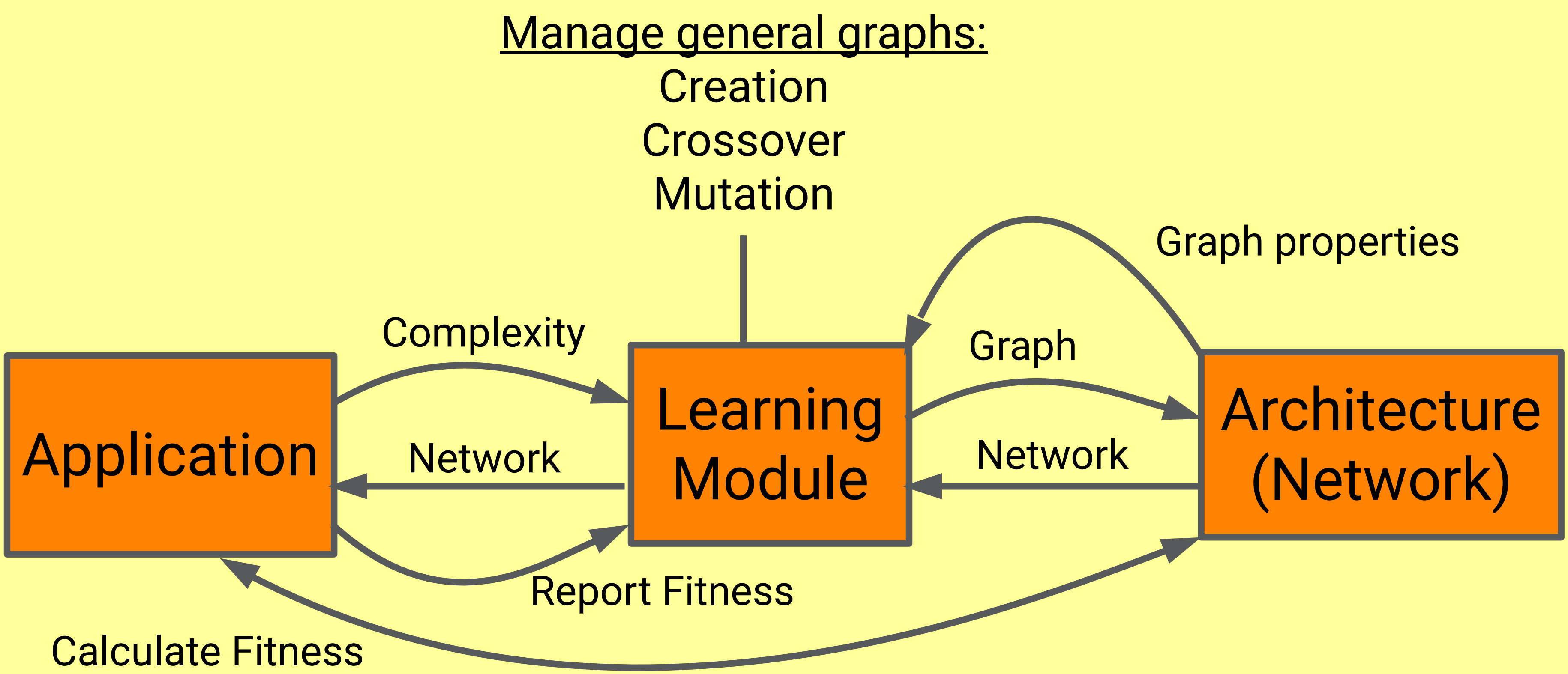
Learning – Where does it go?

- The problem with this approach
  - Large burden on the architecture developer.
  - Does not give the learning module the ability to do anything fancy.



Learning – Where does it go?

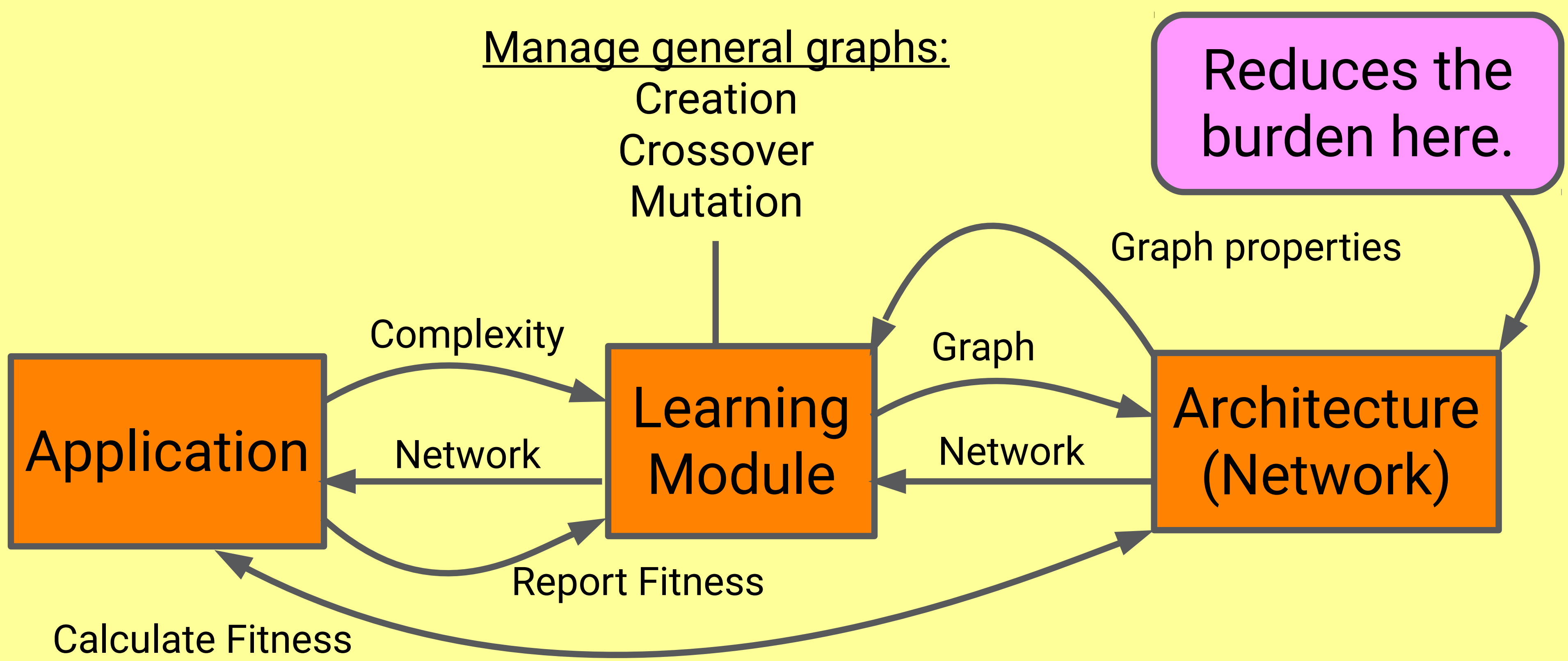
- Instead – put a parameterized graph engine into the learning module.



TENN LAB
NEUROMORPHIC
ARCHITECTURES. LEARNING. APPLICATIONS.

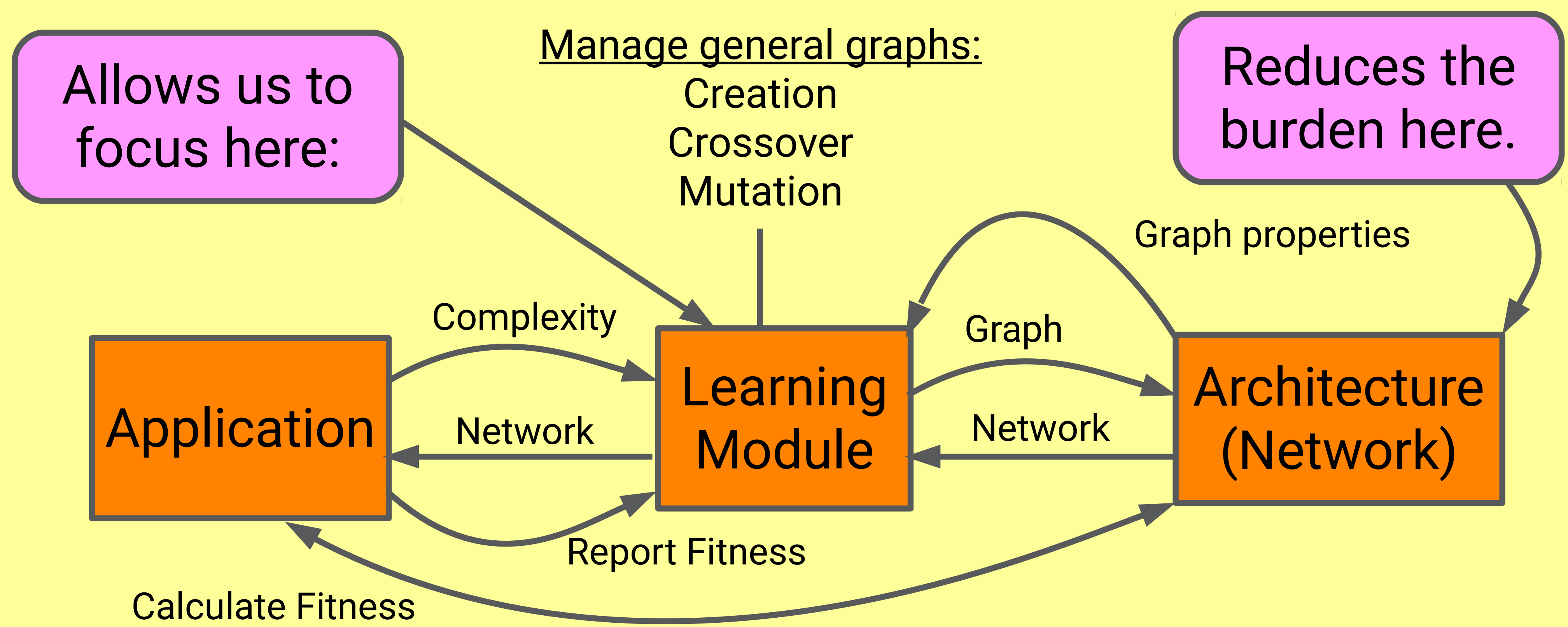
Learning – Where does it go?

- Instead – put a parameterized graph engine into the learning module.



Learning – Where does it go?

- Instead – put a parameterized graph engine into the learning module.



Learning – Status

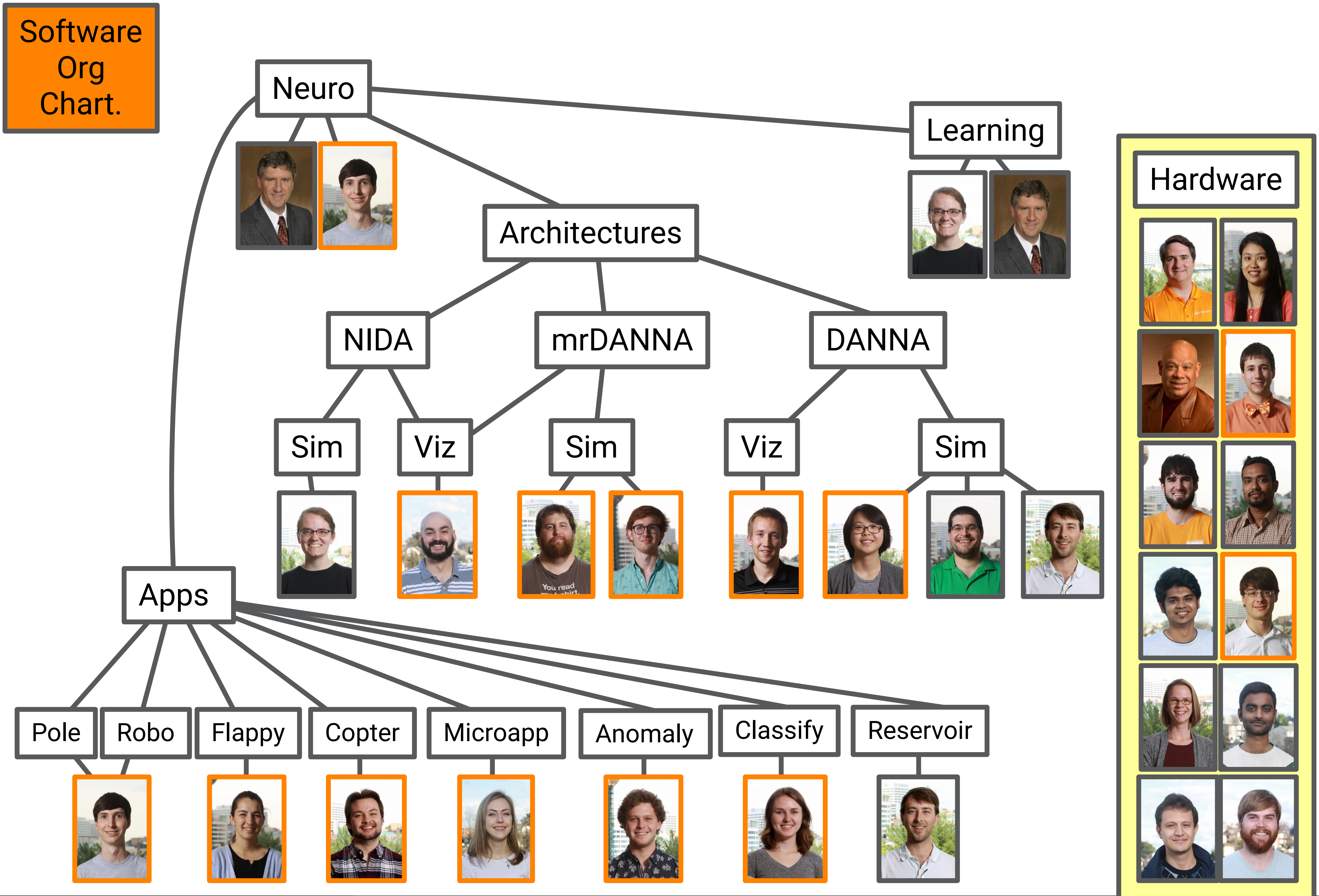
- Still in a feature branch – waiting on mrDANNA.
- Much easier to explore architectural features.
- Poised to exploit speciation / minimal augmenting topologies (NEAT & beyond).
- Still need to explore a more structured approach to STDP.

Some lessons learned (high level)

- Performing simultaneous research on a variety of areas, and getting them to impact each other takes a careful eye on software design.
- There are a lot of un-sexy things that go into a successful hardware/software research project.
- Figuring out how to program applications on neuromorphic computing devices is a **larger challenge** than developing the devices.

Some lessons learned (low level)

- Managing a software team in academia takes an iron fist and a thick skin.
- Program like it's 1998...
- One key to success is decomposing your research space into units that fit your workforce.



TENN LAB
NEUROMORPHIC
ARCHITECTURES. LEARNING. APPLICATIONS.

A Software Stack for Neuromorphic Computing

THE UNIVERSITY OF TENNESSEE KNOXVILLE

OAK RIDGE National Laboratory

James S. Plank
Mark E. Dean
Garrett S. Rose
Catherine D. Schuman

Thank you
AFRL
DOE
NSF

July 19, 2017
Neuromorphic Computing Symposium
Knoxville, Tennessee