

## Standards and Tools in Neuroscience

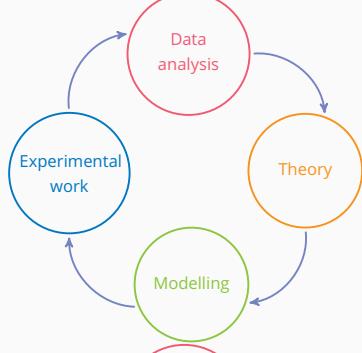
A summary of the Open Source Brain workshop,  
September 2019

Ankur Sinha  
Ph.D. candidate: UH Biocomputation Group, UK,  
Volunteer: Fedora Project.

1/8

## The problem statement

### Neuroscience is complex, and massive



2/8

### Free/Open Neuroscience

Free/Open science:  
Scientific material should be easily, openly [accessible to all](#).

3/8

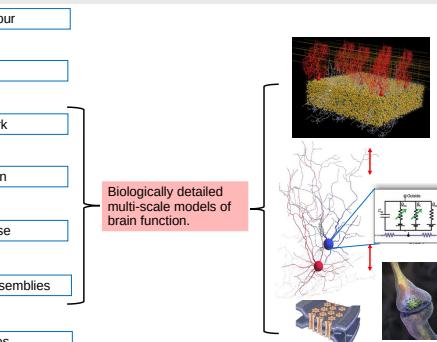
### Experimental neuroscience data is heterogeneous, multiscale and analysis is complex

Anatomy	Electrophysiology	Functional imaging	Behaviour
Receptor Immunohistochemistry	Single/ensemble channel recordings	Synaptic imaging	Restricted task
Neuronal morphologies	Whole cell patch-clamp recordings	Single cell imaging	Freely moving
Brain mapping & Connectomes	Multielectrode array	Population imaging	Natural environment

How can we structure neuroscience data to facilitate reuse?

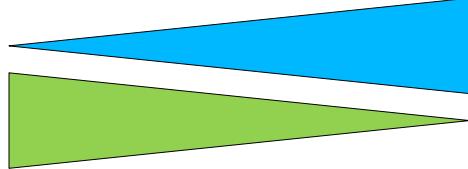
Increasing spatial scale and complexity

### Models of brain function span multiple spatial scales



### A scaling problem

Model complexity →



← Transparency, accessibility, reproducibility, reuse.....and utility as a scientific tool

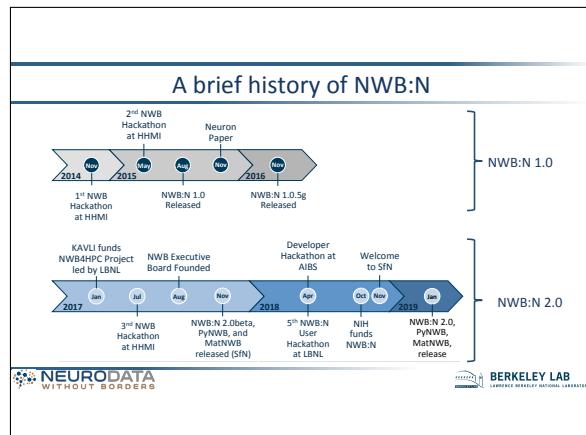
### Standards: the common tongue

## NWB:N 2.0: An Ecosystem for Neurophysiology Data Standardization

**Oliver Rübel**  
Computational Research Division, Lawrence Berkeley National Laboratory

Open Source Brain Workshop  
Alghero, Sardinia  
September 10, 2019



### Overview

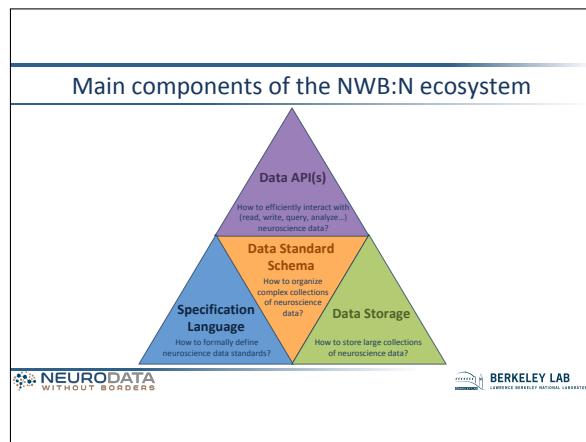
**Motivation:** Lack of standards for neurophysiology data and related metadata is the single greatest impediment to fully extracting return-on-investment from neurophysiology experiments, impeding interchange and reuse of data and reproduction of derived conclusions

**NWB:N – An Ecosystem for Neuroscience Data Standardization**

- The NWB:N data standard defines a unified data format for neurophysiology data, focused on the dynamics of groups of neurons measured under a large range of experimental conditions
- NWB:N is more than just a file format but it defines an ecosystem of tools, methods, and standards for storing, sharing, and analyzing complex neurophysiology data

**Goal:** With NWB:N we aim to develop a next generation data format and software ecosystem that will enable standardization, sharing, and reuse of neurophysiology data and analyses, enhancing discovery and reproducibility



### Specification language

**Goal:** Enable the formal definition and programmatic interpretation of neuroscience data standards

**Format specification language:** Schema for defining hierarchical data schemas

**Main primitives of the specification language:**

- Object primitives:**
  - Group: A group is a collection of objects i.e. subgroups, datasets, links
  - Dataset: n-dimensional array with associated data type, dimensions, etc.
  - Attribute: small metadata dataset defined on a Group or Dataset
  - Link: Like a POSIX soft link to given target neurodata\_type (specifying a Group or Dataset)
- Data type specifications:**
  - Basic data types: strings (ascii, utf8), numeric types (float, int, uint, bool etc.)
  - Compound data types: build complex data type, similar to structs
  - Isodatetime: ISO8601 datetime string, e.g. 2018-09-20T14:34:12Z-02:00
  - Object reference: Like a link to given neurodata\_type but stored as values of a dataset (or attribute)
  - Region reference: Links to regions (i.e., select subsets) of datasets stored as as values of a dataset (or attribute)
- Namespace specification:**
  - Used to define a namespace for format specifications
  - Needed to define and avoid collisions between extensions and to enable the creation of new data standards

### Data storage

**Primary function:** Map NWB: primitives (Groups, Datasets, Attributes etc.) to storage

**NWB:N uses HDF5 as its main file storage backend:**

- Supports large-scale storage of complex data collections in a single file
- Optimized for performance (parallel I/O, advanced I/O filters etc.)
- Supported across platforms and programming languages (Matlab, Python, C/C++, Fortran, R...)
- Targets long-term support

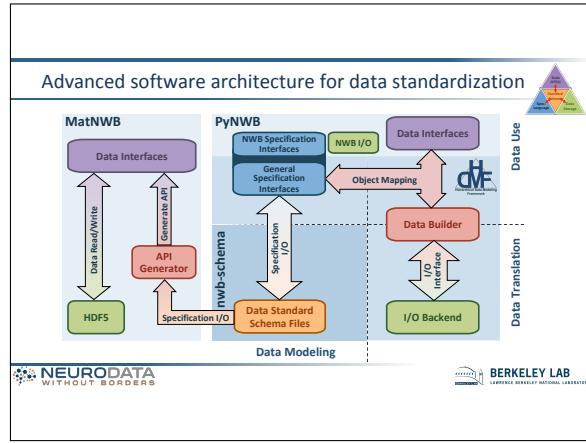
**NWB:N supports advanced I/O features:**

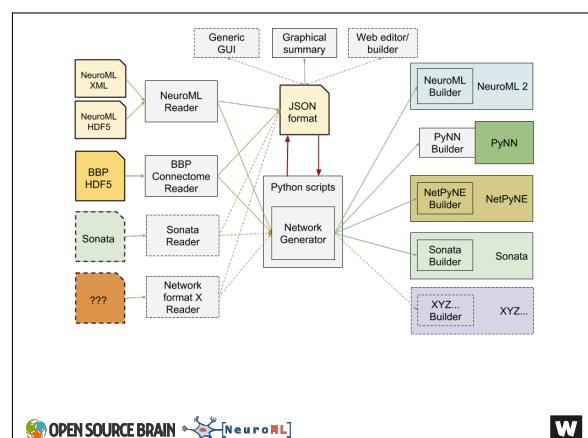
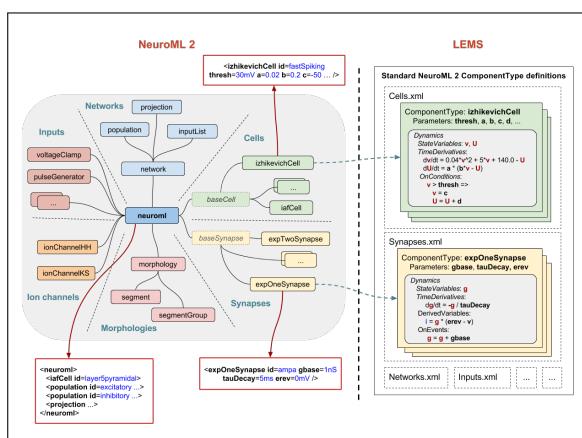
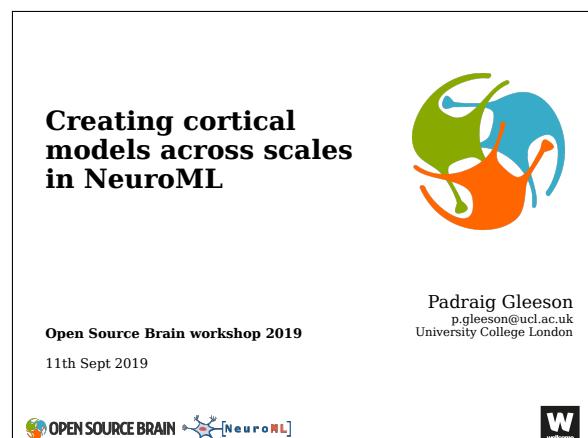
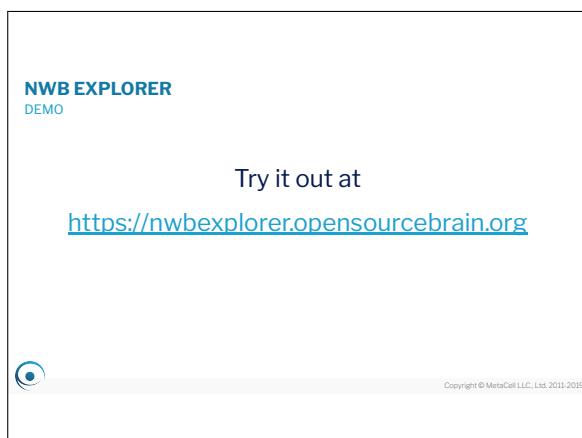
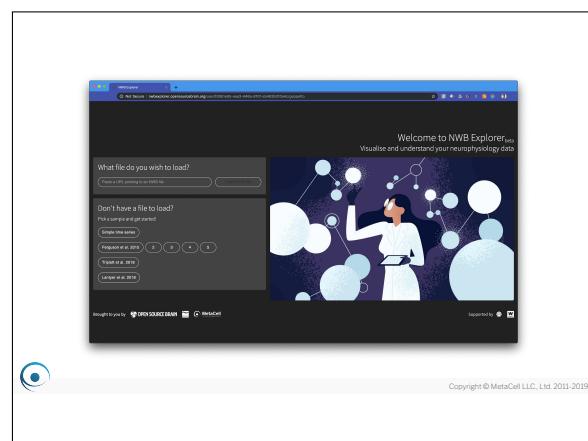
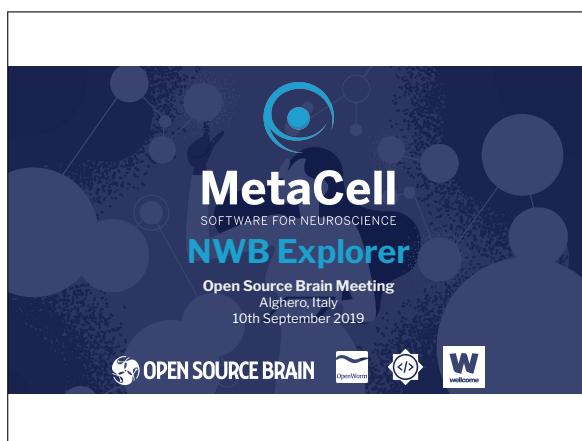
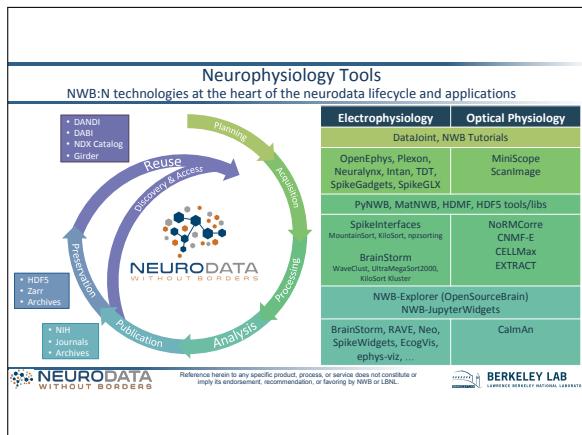
- Lazy data load → Fast file open and efficient memory usage even for very large files
- Chunking → Optimize data layout, storage, and I/O
- I/O filter → E.g. use compression to reduce storage cost
- Self-contained data storage → Store all data in a single file (e.g. for sharing)
- Modular data storage → Store data across multiple files and integrated via external links
- Iterative data write → Support data streaming, reduce memory usage, and optimize I/O

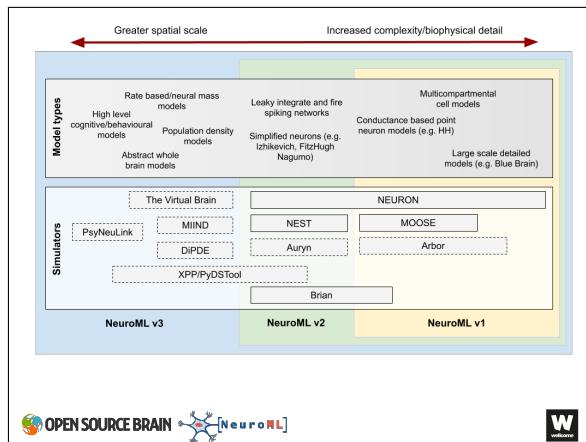
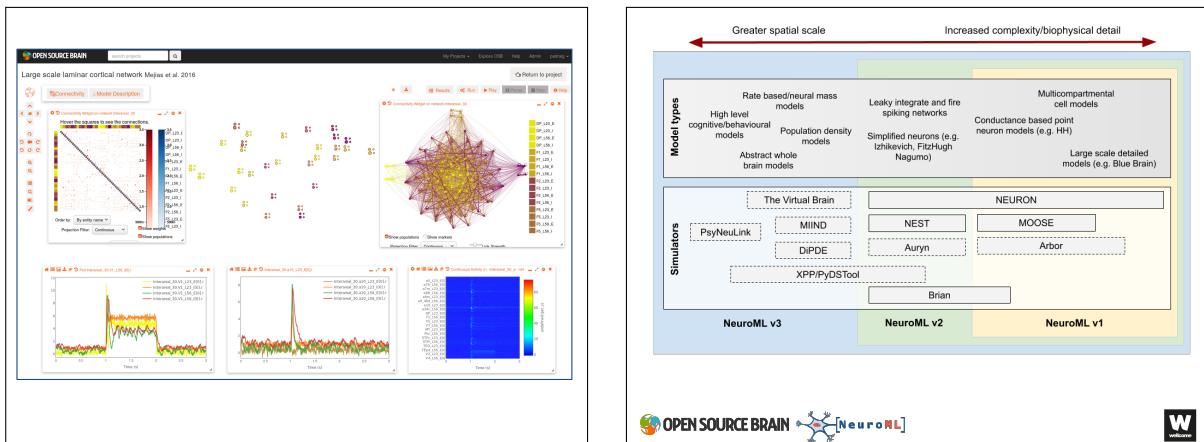
 

### NWB:N supports complex collections of data required for understanding the brain







This figure shows a screenshot of the INCF SIG on Standardised Representations of Network Structures page. The header includes the INCF logo and navigation links for 'About', 'Network', 'Activities', 'Resources', and 'Search'. The main content area has a title 'INCF SIG on Standardised Representations of Network Structures' and a brief description of the SIG's purpose. It also lists 'Contact info' (p.gleeson@ucl.ac.uk) and 'Members' from various institutions.

## Converting simulator specific formats to NeuroML2

Open Source Brain Meeting 2019

Boris Marin  
boris.marin@ufabc.edu.br

Universidade Federal do ABC

### Converting NMODL to NeuroML

The Simple™, OSB sponsored way of converting models to NeuroML2

mailto:p.gleeson@ucl.ac.uk

### The NEURON simulator

Defining models in *NEURON*

<https://www.neuron.yale.edu/neuron/>

- Cells, Networks: *hoc* language (accessible from Python)
  - morphologies
  - synaptic connections
  - *.hoc* files
- Ion Channels (membrane mechanisms): *NM*
  - *.mod* files

### What is NeuroML, and why should I care?

Why can OSB process any NeuroML2 file?

- NML is *structured* (not unlike a *Type System*)

### It is all about **Structure**

Structure in NeuroML / NMODL

- A *Type System* (composability rules) is what grants NML its superpowers
- nmodl is also powerful, but can be used as a general purpose language
  - VERBATIM blocks
  - many different ways of achieving same goal
  - prone to *unstructuredness*
- OSB could in theory treat nmodl the same way as NML...
  - .... if only people stuck to "good practices"!

Levels of Abstraction	
<h2>NeuroML2</h2> <pre>&lt;ionChannelHH id="kChan" conductance="10pS" species="k"&gt;     &lt;gateHrates id="n" instances="4"&gt;         &lt;forwardRate type="HHExplLinearRate" rate="0.1per_ms" midpoint="-55mV" scale="10mV"/&gt;         &lt;reverseRate type="HHExplRate" rate="0.125per_ms" midpoint="-65mV" scale="-80mV"/&gt;     &lt;/gateHrates&gt; &lt;/ionChannelHH&gt;</pre>	
<h2>NMODL</h2> <pre>BREAKPOINT {     SOLVE states METHOD cnexp     gk = gkbar * n ^ 4     ik = gk * (v-ek) } INITIAL{     n = alpha(v) / (alpha(v) + beta(v)) } DERIVATIVE states{     n' = (1 - n) * alpha(v) - n * beta(v) }</pre>	<pre>FUNCTION alpha(Vm(mV))/(ms){     LOCAL x     UNITSOFF     x = (Vm + 55) / 10     if(fabs(x) &gt; 1e-6) {         alpha = 0.1*x/(1-exp(-x))     }else{         alpha = 0.1/(1-0.5*x)     }     UNITON }</pre>

## Levels of Abstraction

### Declarative vs Imperative

- NeuroML2 operates (at least syntactically) closer to the level of abstraction employed by electrophysiologists
- The gory details exist, but elsewhere: *LEMS*
  - i.e. what to do with  $\alpha$ ,  $\beta$ ; the definition of an *ExpRate*; how all of that is converted to conductances/currents...
- But we seldom need (want!) to interact with that level (look under the hood)

```
<Network ...>
    <Cell ...>
        <Channel ...>
            <Gate ...>
                <Rate ...>
```

```
SOLVE{...} METHOD euler
...
DERIVATIVE{...}
...
FUNCTION trap(v){...}
```

Levels of Abstraction

## NetPyNE: structured network specification

i) `popParam['EBC_L2'] = {  
 'celltypes': 'PYR',  
 'TPM': 1,  
 'prob': 0.95,  
 'xrange': [100, 400],  
 'yrange': [50],  
 'zrange': [50]}`

ii) `popParam['EBC_L5'] = {  
 'celltypes': 'PYR',  
 'TPM': 1,  
 'prob': 0.95,  
 'xrange': [100, 400],  
 'yrange': [50],  
 'zrange': [50]}`

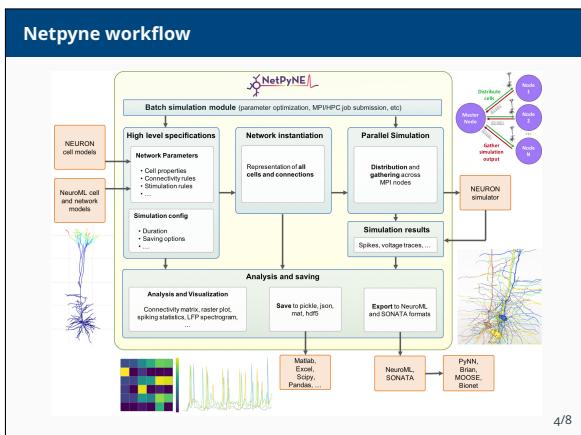
iii) `cellParams['PYR_simple'] = {  
 'conds': { 'cellType': 'PYR'},  
 'params': { 'vinit': '-65',  
 'vrest': '-65',  
 'vthresh': '(-65)',  
 'vreset': '(-65)',  
 'gbar': '0.12',  
 'gbase': '0.036',  
 'gmax': '0.036',  
 'gmin': '-70}}}`

iv) `importCellParam (  
 label = 'PYR_simple',  
 model = 'NetPyNE',  
 type = 'cell',  
 cellClass = 'PYR_simple',  
 cellName = 'PYR_simple',  
 cellType = 'cell')`

v) `synapseParam['MEG'] = {  
 'pre': 'MEG',  
 'post': 'PYR',  
 'weight': 0.9,  
 'delay': 0}`

vi) `connection['EBC_B2'] = {  
 'pre': 'EBC_L2',  
 'post': 'EBC_L5',  
 'probability': 1.0, 'dist': '0.1/200',  
 'weight': 0.001, 'delay': 0, 'synapses': 1}`

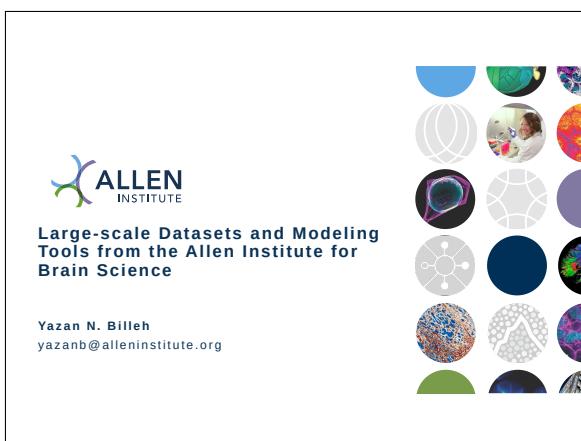
vii) `connection['EBC_B2'] = {  
 'pre': 'EBC_L5',  
 'post': 'EBC_L2',  
 'probability': 1.0, 'dist': '0.1/200',  
 'weight': 0.001, 'delay': 0, 'synapses': 1}`



The figure shows a screenshot of the Netpyne graphical user interface (GUI). The main window is titled "Netpyne GUI". It features several panels for managing a neural network:

- DEFINING YOUR NETWORK**: This panel contains three sub-panels:
  - Cell Types**: Displays a list of cell types with their corresponding morphology and parameters.
  - Spikes**: Shows raster plots and PSTHs for a cell labeled "Cell 0, Prg E2".
  - Connections**: A 3D visualization of connections between neurons, with a color scale from 0.00000 to 0.00005.
- CREATE NETWORK**: This panel contains two sub-panels:
  - Cell Library**: A list of available cell types: "celltype 1", "celltype 2", "celltype 3", "celltype 4", "celltype 5", "celltype 6", "celltype 7", "celltype 8", "celltype 9", "celltype 10", "celltype 11", "celltype 12", "celltype 13", "celltype 14", "celltype 15", "celltype 16", "celltype 17", "celltype 18", "celltype 19", "celltype 20", "celltype 21", "celltype 22", "celltype 23", "celltype 24", "celltype 25", and "celltype 26".
  - Network Diagram**: A 3D visualization of the network structure with a color scale from 0.00000 to 0.00005.

At the bottom left, there are "Create" and "Update" buttons. On the right side, there is a "NETPYNE" watermark.



The diagram illustrates the Allen Institute's research focus and its location in Seattle. At the top left is the **ALLEN INSTITUTE** logo. To its right are logos for **ALLEN INSTITUTE FOR BRAIN SCIENCE**, **ALLEN INSTITUTE FOR CELL SCIENCE**, and **ALLEN INSTITUTE FOR IMMUNOLOGY**. Further right is the **PAUL G. ALLEN FRONTIERS GROUP** logo. Below these is a photograph of the Seattle skyline at dusk, featuring the Space Needle and various buildings. To the left of the photo is the text: **hard problems complexity foundational biology**. An arrow points from this text to the word **big science** below it. Another arrow points from **big science** to the word **open science** below it. To the right of the photo is the text: **data knowledge tools**. Below the main text area is the **ALLEN INSTITUTE** logo again, followed by the URL [braincellatlas.org](http://braincellatlas.org) and the number **[22]**.

- Established 2003 by Paul G. Allen
- South Lake Union, Seattle, WA
- 500 employees++

# CORE PRINCIPLES

## Team Science

Interdisciplinary teams working towards common goal



## Big Science

Large-scale projects with robust, massive data



## Open Science

All resources available online at [brain-map.org](http://brain-map.org) or [allencell.org](http://allencell.org)

---

 ALLEN INSTITUTE

[alleninstitute.org](http://alleninstitute.org) | 3

The figure is a horizontal timeline showing the development of various Allen Institute public resources from 2003 to 2014. Each resource is represented by a circular icon with its name and year at the top, and a brief description below.

- 2003:** Allen Institute for Brain Science Launched
- 2004:** Allen Mouse Brain Atlas
- 2005:** Allen Developing Mouse Brain Atlas
- 2006:** NIH Blueprint Non-Human Primate Atlas
- 2007:** Allen Human Brain Atlas
- 2008:** BrainSpan Atlas of the Developing Human Brain
- 2009:** Allen Mouse Brain Connectivity Atlas
- 2010:** Allen Cell Type Database
- 2011:** Allen Brain Observatory
- 2012:** Allen Dementia and TB Database
- 2013:** Allen GAP Database

**Our Models and Modeling Software Are Freely Available to the Community**

Brain Modeling ToolKit (BMTK): <https://alleninstitute.github.io/bmtk/>

```
graph TD; BMTK[BMTK] --> ModelBuilder[Model Builder]; BMTK --> NEURON[NEURON<br/>for empirically-based simulations]; BMTK --> nest[nest<br/>DIPDE]; BMTK --> FilterNet[FilterNet<br/>...]
```

The diagram illustrates the integration of the Brain Modeling ToolKit (BMTK) with other modeling tools. At the top, a blue box labeled "BMTK" has arrows pointing down to four separate components: "Model Builder", "NEURON for empirically-based simulations", "nest DIPDE", and "FilterNet". The "Model Builder" component is associated with a brain neuron icon. The "nest DIPDE" component is associated with a brain cell and a small plot icon. The "FilterNet" component is associated with a plot icon. An ellipsis "..." is positioned above the "FilterNet" component.

## Our Models and Modeling Software Are Freely Available to the Community



# Human Brain Project

## How model standardization enables new tools and applications in neuroscientific research

### Insights from the HBP

**Motivation**

```

graph TD
    CI([Conceptual Idea]) --> MB(Model Building)
    A([Algorithm]) --> MB
    MB --> EC(Experimental characterization)
    MB --> LED(Large Experimental Databases)
    MB --> UP(user PC)
    MB --> MS(Model Sharing)
    EC --> LED
    LED <--> MS
    MS --> RM(Released Model)
    MS --> MD(Model Database)
    MD --> APP(Applications)
    APP --> MB
    APP --> BIA(biologically-inspired AI)
    Applications: medicine, robotics, brain diseases, ...
    
```

Model production pipeline within the infrastructure and research goals of the Human Brain Project

The diagram illustrates the PyNN architecture, which provides a unified interface for neuronal network simulators. It is organized into several layers:

- Simulator-independent environments for developing neuroscience models:**
  - keep the advantages of having multiple simulators or hardware devices
  - but remove the translation barrier.
- Three (complementary) approaches:**
  - GUI (e.g. neuroConstruct)
  - XML-based language (e.g. NeuroML, NineML)
  - interpreted language (e.g. Python)
- PyNN API:** The central interface layer, shown as a yellow box containing:
  - PyNN seed
  - PyNN brain
  - PyNN neuron
  - PyNN recorder
  - PyNN hardware interface
  - PyNN multithread
- Simulator-specific PyNN module:** A layer above the API, containing:
  - PyNEST
  - PyBrain
  - PyMAGE
  - PyNeuralML
  - PyNestML
  - PyNeuroML
  - PyPACMAN
  - PyNNhardware
  - PyNNmultithread
- Python interpreter:** A layer below the specific PyNN modules.
- Native interpreter:** A layer below the Python interpreter.
- Simulator kernel:** The bottom layer, containing NEST, Brian, NEURON, and NEURON++. Arrows indicate communication paths between these components and the PyNN layers.

Legend at the bottom:

- Green double-headed arrow: Direct communication
- Blue double-headed arrow: Code generation

```
Sonata (pyNN support)
```

Large-scale simulation of biophysically-detailed neuronal circuits  
→ sets specific constraints

the SONATA Data Format emerges as the standard optimized for performance for simulation, analysis and visualization of large-scale circuits  
(joint initiative of Blue Brain Project and the Allen Institute for Brain Science)

**Import from Sonata format**

```
from pyNN.serialization import import_from_sonata, load_sonata_simulation_plan
import pyNN.neuron as sim

simulation_plan = load_sonata_simulation_plan("simulation_config.json")
sim.setup(**simulation_plan)
net = import_from_sonata("circuit_config.json", sim)
simulation_plan.execute(net)
export_to_sonata(net, "sonata_output_dir")
```

**Export to Sonata format**

```
from pyNN.netwerk import Network
from pyNN.serialization import export_to_sonata

sim.setup()
...
# create populations, projections, etc.
...
# add populations and projections to a Network
net = Network()
pop1, ..., popN, proj1, ..., projN, ... = ...
export_to_sonata(net, "sonata_output_dir")
```

## SciUnit

<https://github.com/scidash/sciunit>

---

**Include a validation framework in model development**

- ✓ **What is SciUnit?**
  - A **Test-driven** framework for formally validating scientific models against data.
  - It employs the concept of **Capabilities**.
  
- ✓ **What are Tests?**
  - A procedure intended to establish the quality, performance, or reliability of a model
  
- ✓ **What are Capabilities?**
  - interfaces through which the model and the validation framework communicate
  - implemented as methods (functions) within the model

```

graph LR
    Model[Model] -->|implements| Capability[Capability]
    Capability -->|simulated observations| Score[Score]
    Score -->|experimental observations| Test[Test]
    Test -->|requires| Model
  
```

The diagram illustrates the SciUnit workflow. It starts with a **Model** box, which has an arrow labeled **implements** pointing to a **Capability** box. From the **Capability** box, an arrow labeled **simulated observations** points to a **Score** box. From the **Score** box, an arrow labeled **experimental observations** points to a **Test** box. Finally, an arrow labeled **requires** points from the **Test** box back to the **Model** box.

**Score**      **See Part II**

Requires the participation of modellers:

- ✓ support to wrap your models for SciUnit
- ✓ add/request new tests to the library
- ✓ critique existing tests
- ✓ suggest new features

## Test Packages

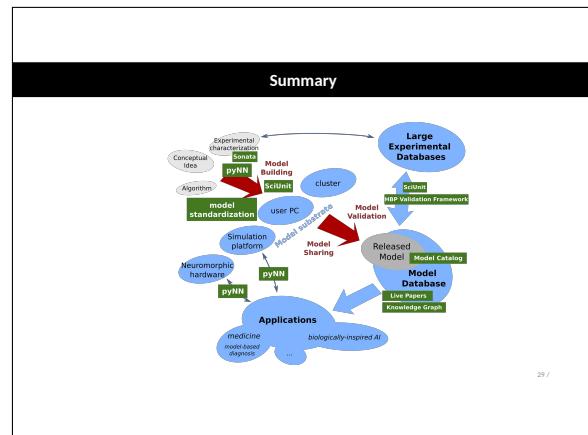
The overall test suite has been divided into a number of components, some containing validation tests specific to particular brain regions, others more generic. All validation tests are written in Python, using the SciUnit framework. Some of these are listed below:

**Test suites for specific brain regions**

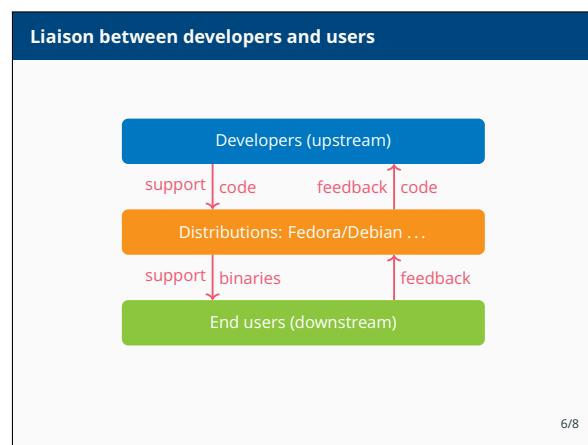
- ❑ **HippoUnit:** <https://github.com/Kalilab/hippounit>
- ❑ **HippoNetworkUnit:** <https://github.com/pedroernesto/HippoNetworkUnit>
- ❑ **CerebUnit:** <https://github.com/lungs/cerebellum-unit>
- ❑ **BasalUnit:** <https://github.com/appukuttan-shailesh/basalunit>

**Test suites for model features, independent of cell type or brain region**

- ❑ **MorphoUnit:** <https://github.com/appukuttan-shailesh/morphounit>
- ❑ **NetworkUnit:** [https://github.com/mvonpapen/simrest\\_validation](https://github.com/mvonpapen/simrest_validation)
- ❑ **eFELUnit:** <https://github.com/appukuttan-shailesh/eFELunit>



## NeuroFedora: marketing pitch



## Search: "NeuroFedora"

"Live" ISO now ready to download (demo)  
 Mailing list: [neuro-sig@lists.fedoraproject.org](mailto:neuro-sig@lists.fedoraproject.org)  
 IRC: #fedora-neuro on Freenode  
 Telegram: [t.me/NeuroFedora](https://t.me/NeuroFedora)  
 Documentation [neuro.fedoraproject.org](http://neuro.fedoraproject.org)  
 Blog: [neuroblog.fedoraproject.org](http://neuroblog.fedoraproject.org)  
 Pagure.io (FOSS Git forge): [neuro-sig/NeuroFedora](https://pagure.io/neuro-sig/NeuroFedora)

## License

This presentation is made available under a Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) license.