NWB:N 2.0: An Ecosystem for Neurophysiology Data Standardization

Oliver Rübel

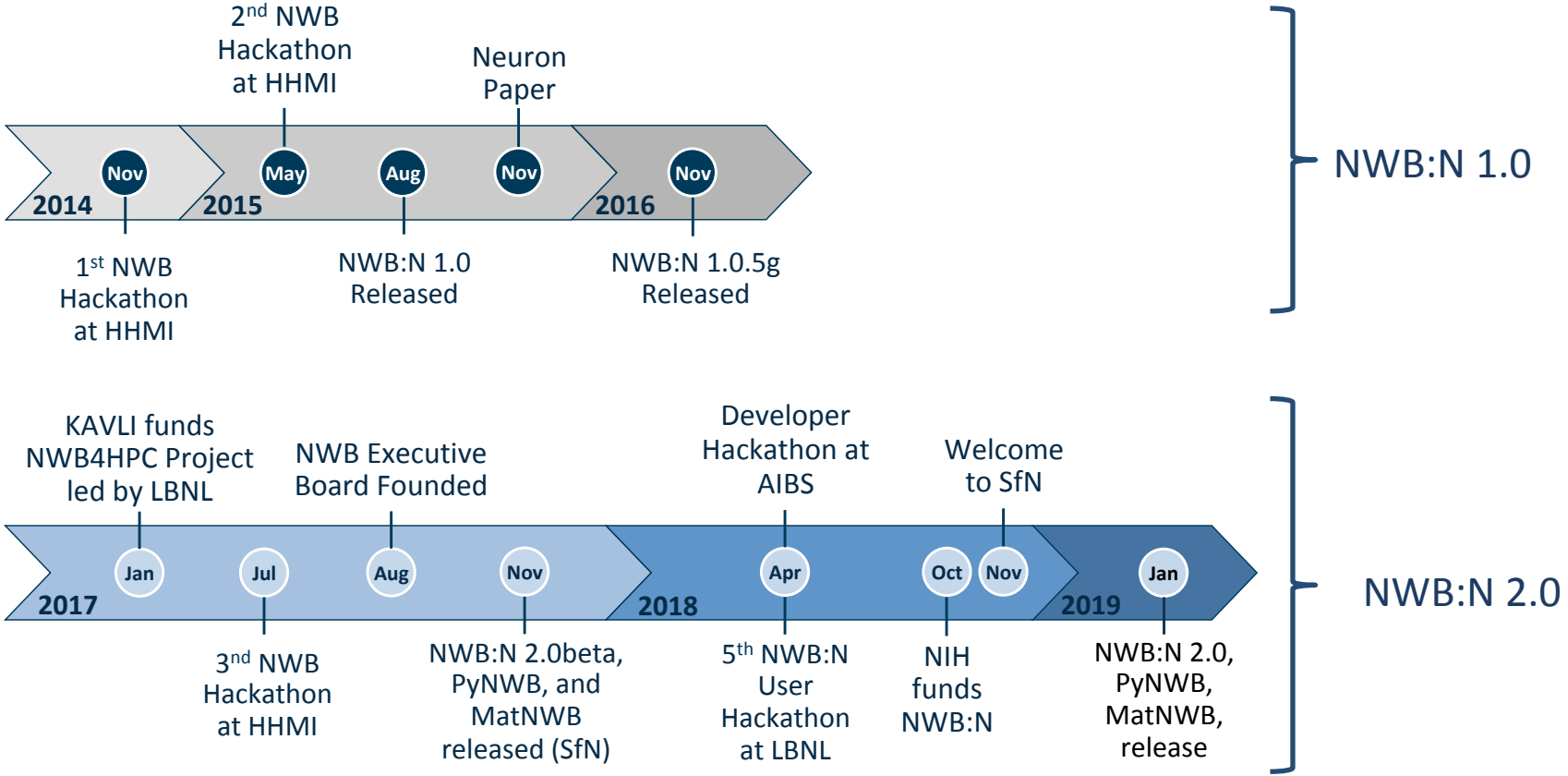
Computational Research Division, Lawrence Berkeley National Laboratory

Open Source Brain Workshop

Alghero, Sardinia

September 10, 2019

A brief history of NWB:N



Overview

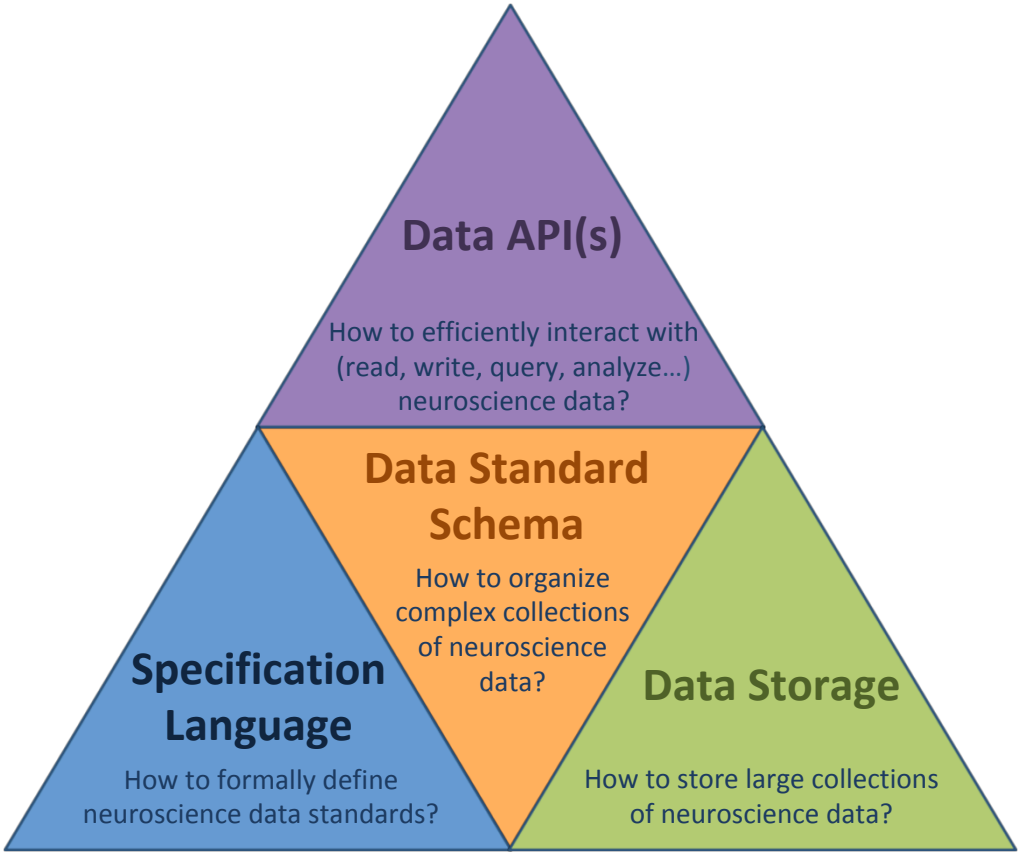
Motivation: Lack of standards for neurophysiology data and related metadata is the single greatest impediment to fully extracting return-on-investment from neurophysiology experiments, impeding interchange and reuse of data and reproduction of derived conclusions

NWB:N – An Ecosystem for Neuroscience Data Standardization

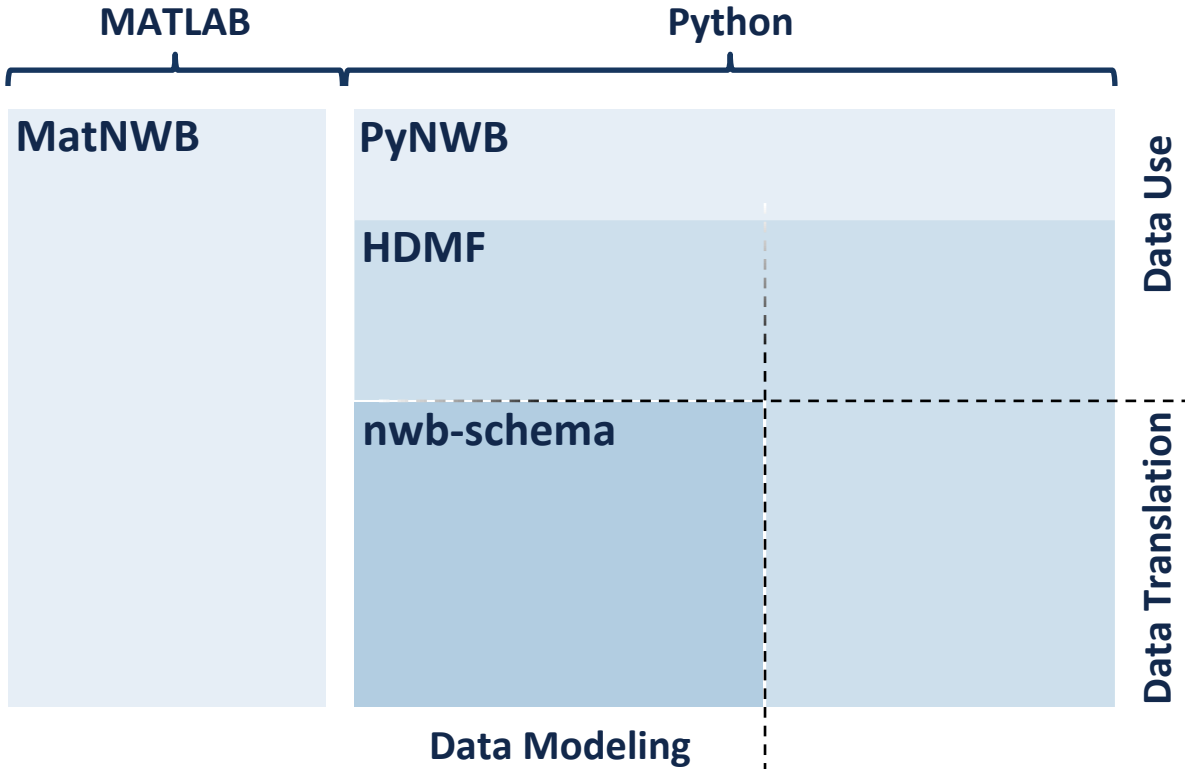
- The NWB:N data standard defines a unified data format for neurophysiology data, focused on the dynamics of groups of neurons measured under a large range of experimental conditions
- NWB:N is more than just a file format but it defines an ecosystem of tools, methods, and standards for storing, sharing,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

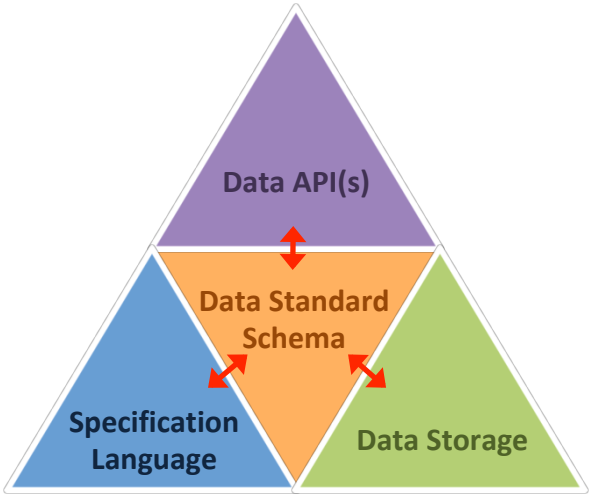
Goal: With NWB:N we aim to develop a next generation data format and software ecosystem that will enable standardization, sharing, and reuse of neurophysiology data and analyses, enhancing discovery and reproducibility

Main components of the NWB:N ecosystem

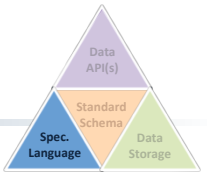


Main components of the NWB:N software stack





Specification language



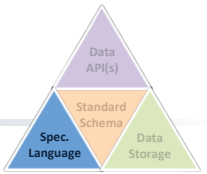
Goal: Enable the formal definition and programmatic interpretation of neuroscience data standards

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.) etc.
  - **Compound data types:** build complex data type, similar to structs
  - **Isodatetime:** ISO8061 datetime string, e.g. 2018-09-28T14:43:54.123+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

Defining reusable data types



`neurodata_type`: Defines reusable types that can be used elsewhere in other specifications

- Similar to a class in OOP
- All objects in a data standard must have either a unique name or `neurodata_type`

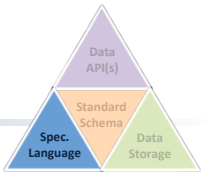
Enables reuse of types through inheritance and inclusion (i.e., composition) via the keys:

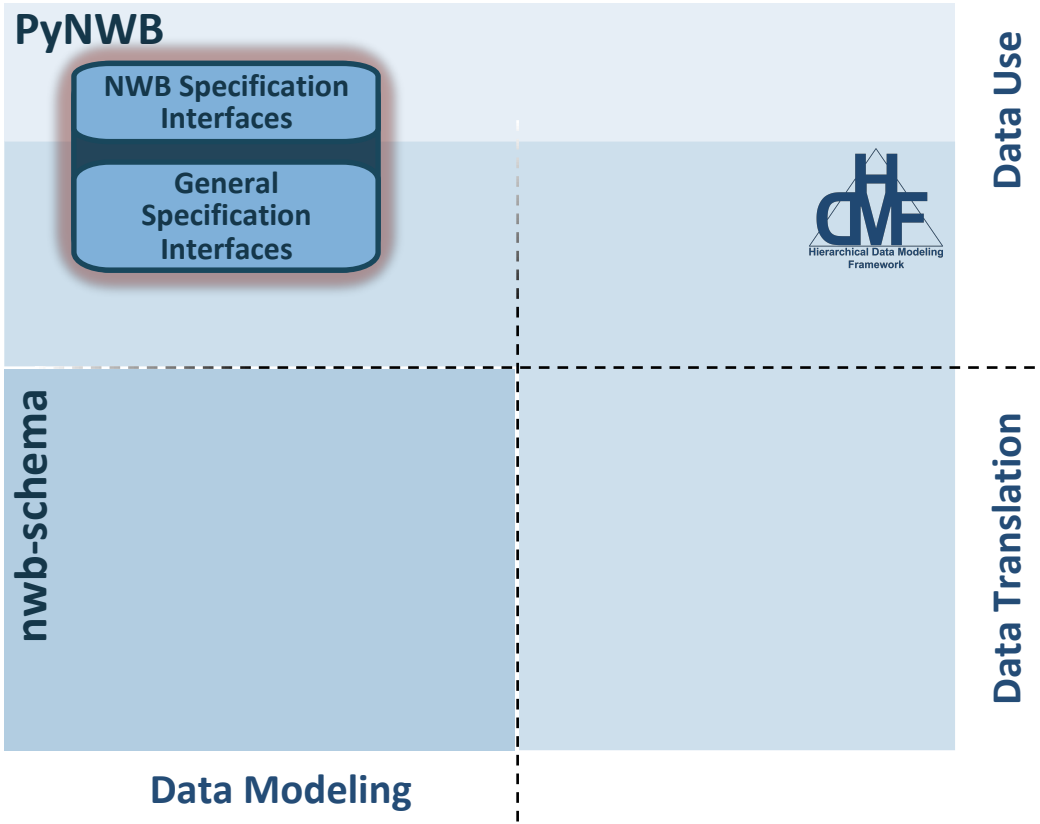
- **`neurodata_type_inc`**: include an existing type
- **`neurodata_type_def`**: defines a new type

The NWB:N schema is essentially a collection and organization of reusable `neurodata_types`:

- Extending NWB:N to support new data types typically amounts to defining new `neurodata_types` (building on existing core types as much as possible)

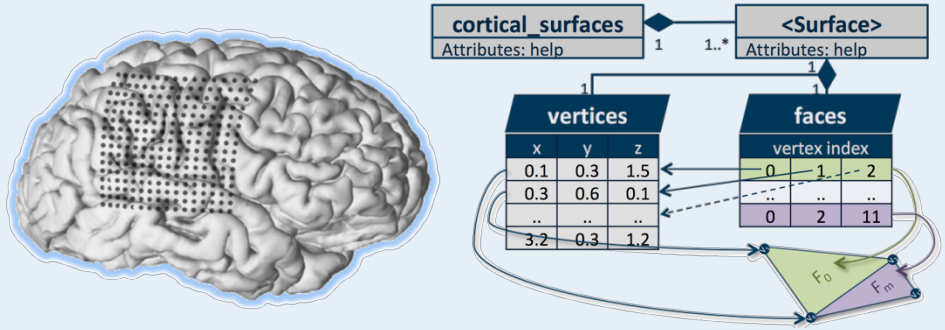
Specification language API





Example: Using the specification language to extend NWB:N

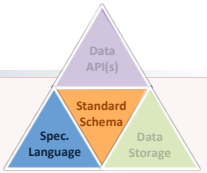
Creating the extension




```
from pynwb.spec import NWBDatasetSpec, NWBNamespaceBuilder,
                       NWBGroupSpec, NWBAttributeSpec
surface = NWBGroupSpec(neurodata_type_def='Surface',
            neurodata_type_inc='NWBDataInterface' ,
            quantity='+', doc='brain cortical surface')
surface.add_dataset(NWBDatasetSpec(
            doc='..' , shape=(None, 3),
            name='faces', dtype='uint' , dims=...)
surface.add_dataset(NWBDatasetSpec(
            doc='..', shape=(None, 3) ,
            name='vertices', dtype='float', dims=...)
surface.add_attribute(...)

ns_builder = NWBNamespaceBuilder(doc=..., name='ecog',
                  version='1.0', author='Ben Dichter', ...)
ns_builder.add_spec('ecog.extensions.yaml', surface)
ns_builder.export('ecog.namespace.yaml')
```

Using the extension



PyNWB: write

```
from pynwb import load_namespaces, get_class,
                  NWBHDF5IO, NWBFile ...
nwbfile = NWBFile(...)
load_namespaces('ecog.namespace.yaml')
Surface = get_class('Surface', 'ecog')
surf = Surface(faces=... , vertices=...,
               name='Surface_1'...)
nwbfile.add_acquisition(surf)
with NWBHDF5IO('surface_example.nwb', 'w') as io:
    io.write(nwbfile)
```

PyNWB: read

```
load_namespaces('ecog.namespace.yaml' )
io = NWBHDF5IO('surface_example.nwb', 'r')
nwbfile = io.read()
nwbfile.get_acquisition('Surface 1').vertices
```

MatNWB: write

```
generateExtension('ecog.namespace.yaml');
file = nwbfile(...)
surf = types.ecog.Surface(...
                  'faces', faces, 'vertices',
vertices);
file.acquisition.set('Surfaces_1', surf);
nwbExport(file, 'ecephys_tutorial.nwb')
```

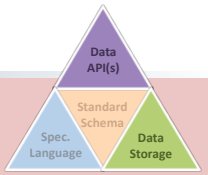
Example: Writing data using extensions

Creating the extension


```
from pynwb.spec import NWBDatasetSpec, NWBNamespaceBuilder,
                       NWBGroupSpec, NWBAttributeSpec
surface = NWBGroupSpec(neurodata_type_def='Surface',
            neurodata_type_inc='NWBDataInterface' ,
            quantity='+', doc='brain cortical surface')
surface.add_dataset(NWBDatasetSpec(
            doc='..' , shape=(None, 3),
            name='faces', dtype='uint' , dims=...)
surface.add_dataset(NWBDatasetSpec(
            doc='..', shape=(None, 3) ,
            name='vertices', dtype='float', dims=...)
surface.add_attribute(...)

ns_builder = NWBNamespaceBuilder(doc=..., name='ecog',
                  version='1.0', author='Ben Dichter', ...)
ns_builder.add_spec('ecog.extensions.yaml', surface)
ns_builder.export('ecog.namespace.yaml')
```

Using the extension



PyNWB: write

```
from pynwb import load_namespaces, get_class,
                  NWBHDF5IO, NWBFile ...
nwbfile = NWBFile(...)
load_namespaces('ecog.namespace.yaml')
Surface = get_class('Surface', 'ecog')
surf = Surface(faces=... , vertices=...,
               name='Surface_1'...)
nwbfile.add_acquisition(surf)
with NWBHDF5IO('surface_example.nwb', 'w') as io:
      io.write(nwbfile)
```

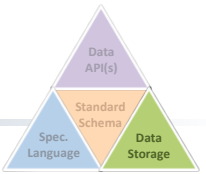
PyNWB: read

```
load_namespaces('ecog.namespace.yaml' )
io = NWBHDF5IO('surface_example.nwb', 'r')
nwbfile = io.read()
nwbfile.get_acquisition('Surface 1').vertices
```

MatNWB: write

```
generateExtension('ecog.namespace.yaml');
file = nwbfile(...)
surf = types.ecog.Surface(...
                  'faces', faces, 'vertices',
vertices);
file.acquisition.set('Surfaces_1', surf);
nwbExport(file, 'ecephys_tutorial.nwb')
```

Data storage



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

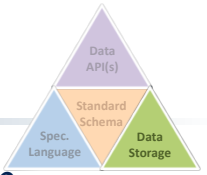
NWB:N uses HDF5 as its main file storage backend:

- 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

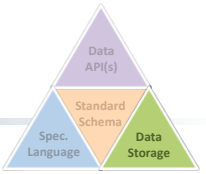
What is the right data storage backend?

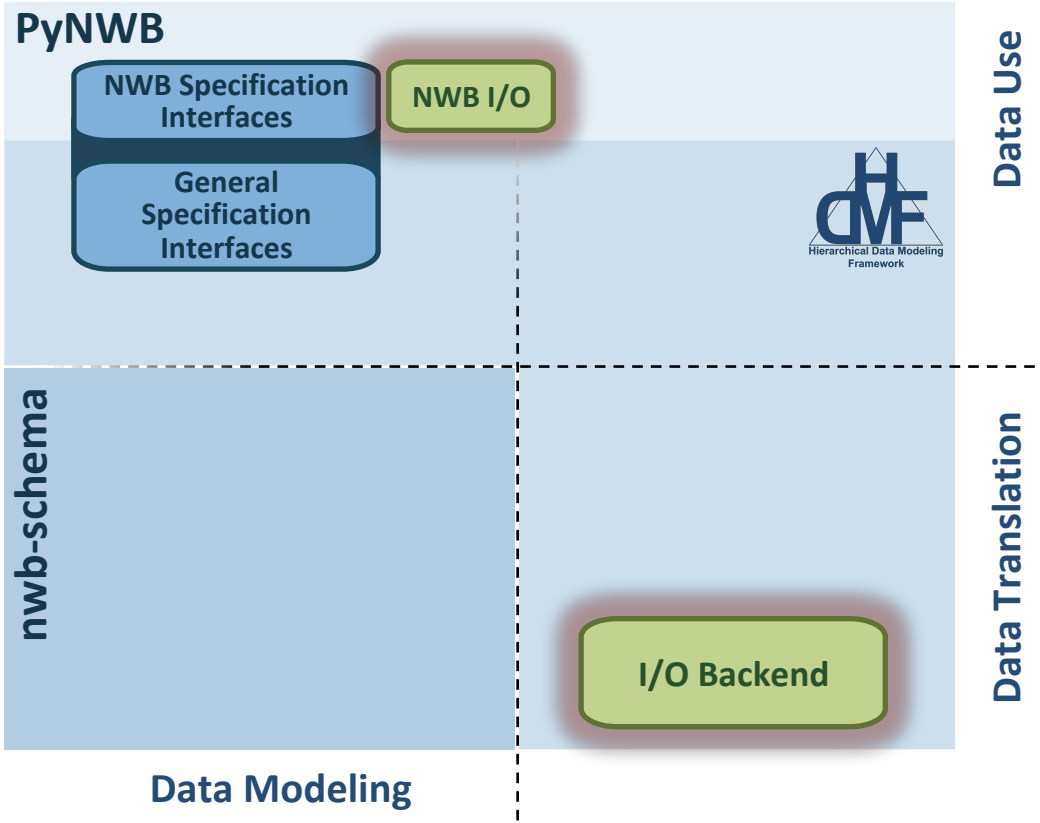


HDMF defines an abstract I/O layer, decoupling the I/O backend (e.g., HDF5) from the specification and in-memory data structures. This enables the design of alternate storage backends in the future, e.g., to:

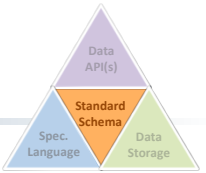
- Optimize storage for data management and analysis systems
- Support file-system-based storage (e.g., ZARR, ExDIR)
- Support integration of external standards (e.g., media formats)

Data storage API





NWB:N data standard schema

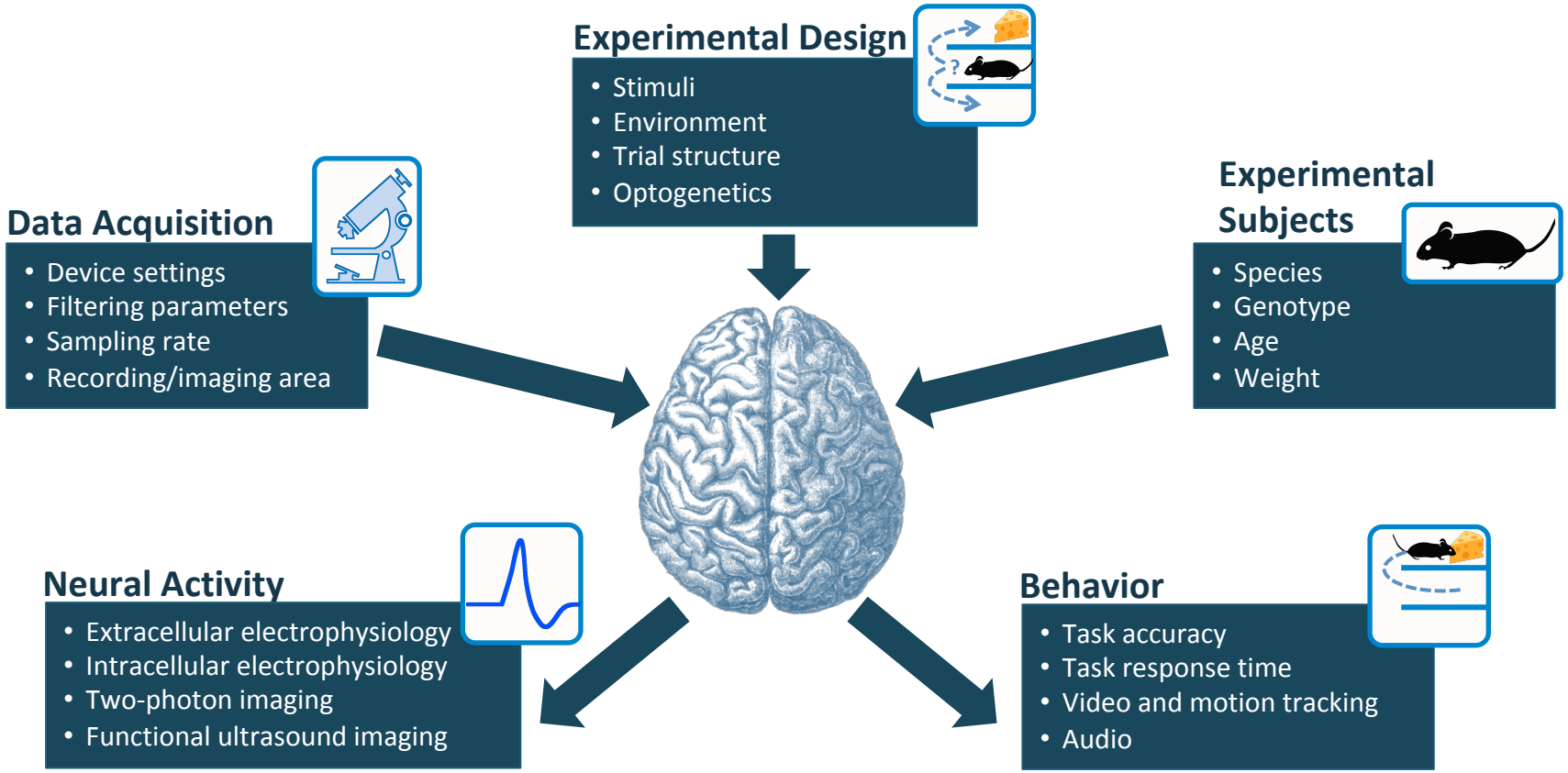


A format schema (a.k.a. specification) is a collection of YAML files that use the specification language to describe the organization of data in a data format via verifiable, computer and human readable documents

The **NWB:N data standard** defines:

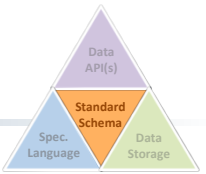
- **High-level data concepts:**
  - **TimeSeries** as base data type for all time series data. Timeseries are synchronized to a global clock with custom time stamps stored in dedicated sync group.
  - **NWBData / NWBContainer/ NWBDataInterace** as a common base for datasets, groups, and experimental data
  - **ProcessingModules** to logically group analyses
  - **DynamicTable** for storing complex data tables
- **A large collection of reusable `neurodata_types`** for storing a broad range of neurophysiology data types:
  - 20 different, specialized TimeSeries types for a broad range of use-cases, e.g., ElectricalSeries, OptogeneticSeries, PatchClampSeries etc
  - 22+ analysis containers, e.g., for BehavioralEvents, FeatureExtraction, Clustering …
- **Base hierarchy to logically group data:**
  - **/acquisition** for storage of data streams recorded from the system, e.g., recordings from electro- and optophysiology or behavioral tracking systems
  - **/processing** for storage of standardized processing modules, often as part of intermediate analyses required before scientific analysis, e.g., results from spike sorting, signal filtering, or image processing
  - **/intervals** for storage of experimental intervals, e.g., experimental epochs or trials
  - **/stimulus** for storage of stimulus data
  - **/general** for storage of experimental metadata, e.g., protocol, notes or device descriptions
  - **/analysis** for storage of lab-specific and custom scientific analysis data

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



Ragged arrays

High-level data structures (Part 1/2)

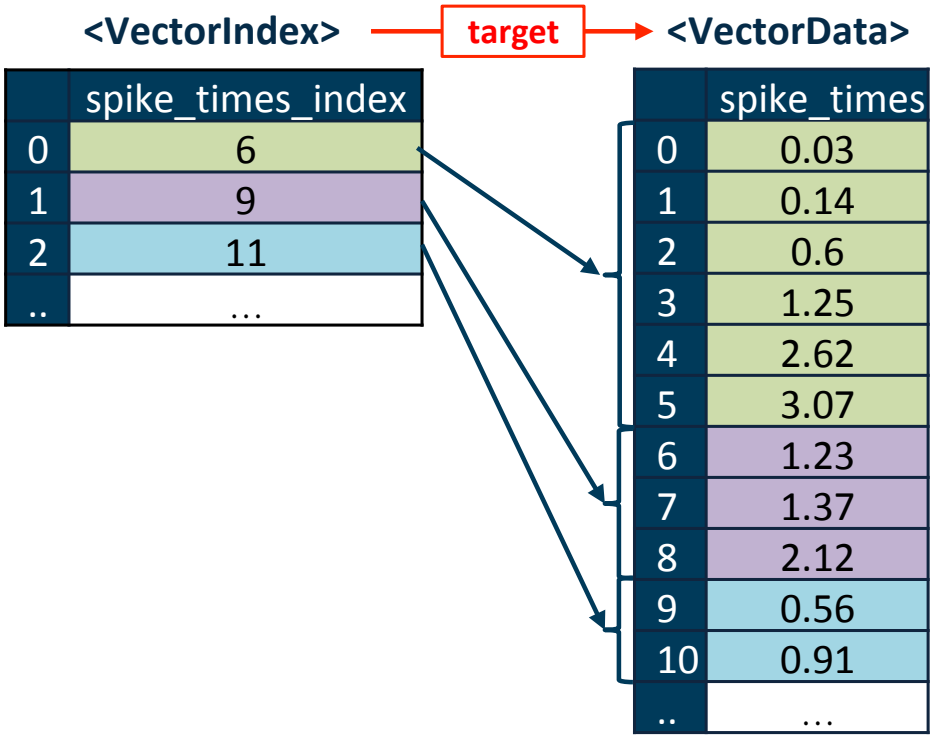


Ragged array (a.k.a. jagged array): An arrays where each element of the array is itself an array of variable length.

0	0.03	0.14	0.6	1.25	2.62	3.07
1	1.23	1.37	2.12			
2	0.56	0.91				
...	...	...	...	...		

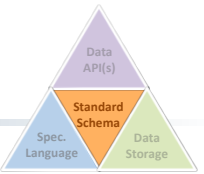
Example application: Feature extracting (e.g., detection of spikes or ROIs), where each feature is described by a variable-length vector, e.g., a pixel mask or spike times

Ragged arrays in NWB:N 2.0:



Tables

High-level data structures (Part 2/2)



Row-based compound table

electrodes (Dataset)						
	(id,	x,	y,	z, imp,	location, …)	
0	(0 ,	0.1,	0.3,	0.4,	4.5,	‘CA1’, …)
1	(1 ,	0.5,	0.4,	0.2,	10.4,	‘CA1’, …)
2	(2 ,	1.5,	2.6,	1.8,	1.1,	‘CA2’, …)
..	…					

Column-based `DynamicTable`

electrodes (Group)

	id	x	y	z	imp	location	…
0	0	0.1	0.3	0.4	4.5	CA1	
1	1	0.5	0.4	0.2	10.4	CA1	..
2	2	1.5	2.6	1.8	1.1	CA2	..
..	…	…	…	…	…	…	…

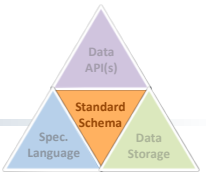
Hybrid table

electrodes (Group)

	id	(x,y,z)	imp	location	…
0	0	(0.1, 0.3, 0.4)	4.5	CA1	
1	1	(0.5, 0.4, 0.2)	10.4	CA1	..
2	2	(1.5, 2.6, 1.8)	1.1	CA2	..
..	…	…	…	…	…

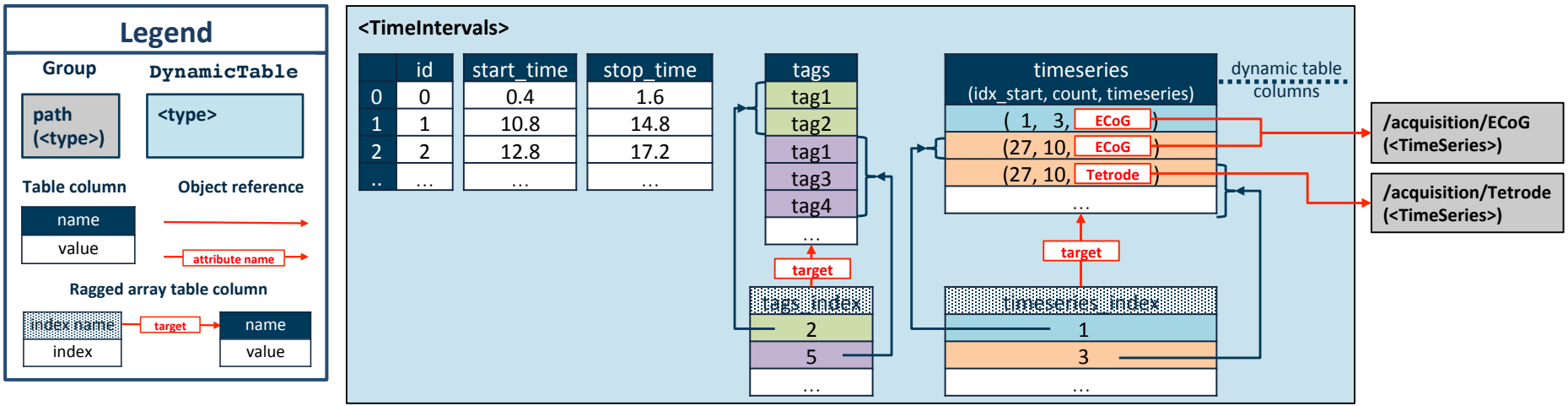
Managing time intervals

Example applications of tables/ragged arrays (Part 1/3)



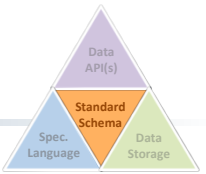
Time intervals: Ranges in time usually associated with particular events or features, e.g., spikes, experimental intervals, stimuli, behavior etc.

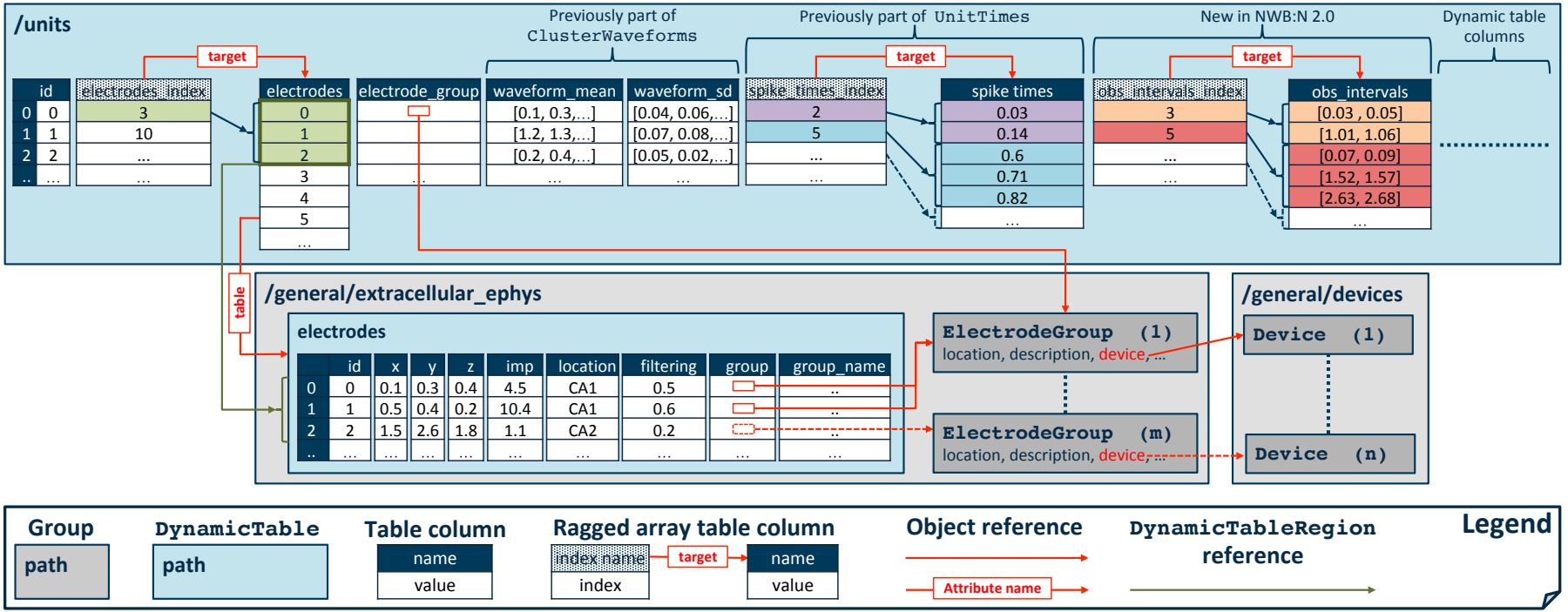
/intervals: NWB:N supports storage of an arbitrary collections of `TimeIntervals` tables, including the predefined tables for epochs, trials, and invalid times



Spiking units in electrophysiology

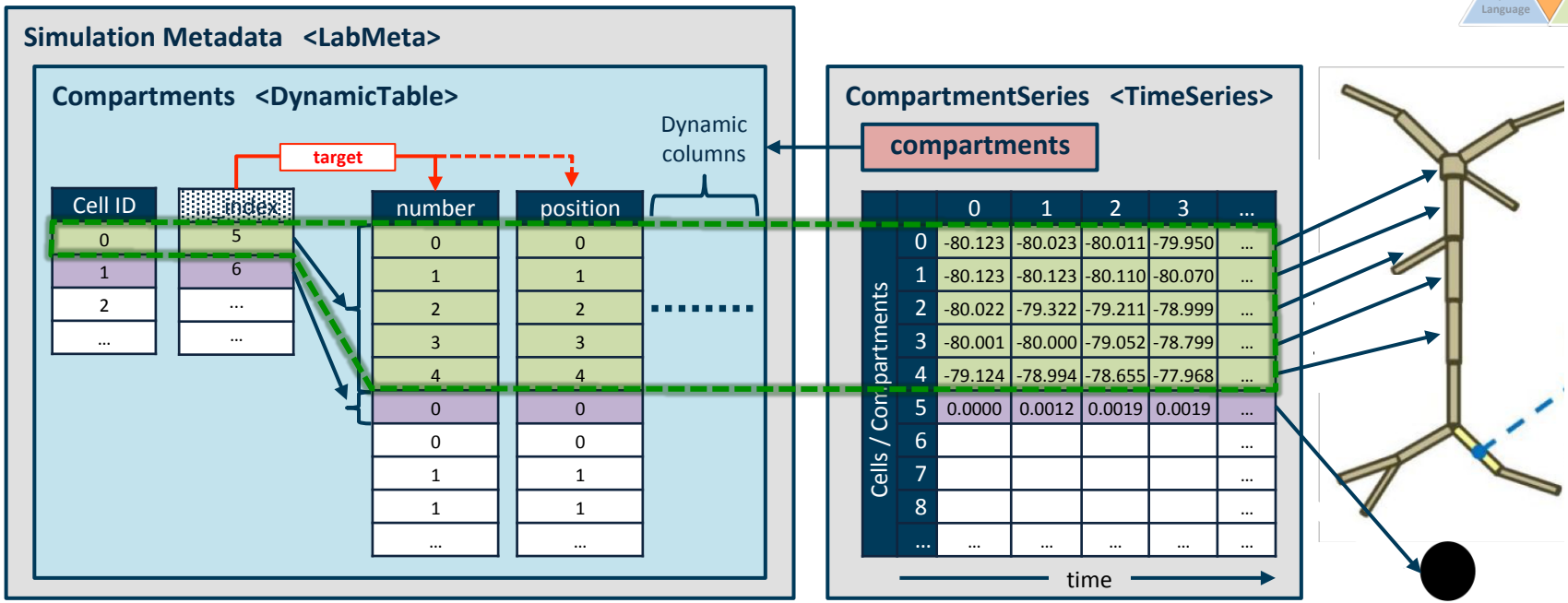
Example applications of tables/ragged arrays (Part 2/3)





Simulation output extension

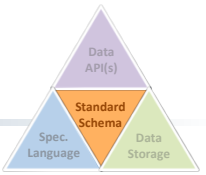
Example applications of tables/ragged arrays (Part 3/3)

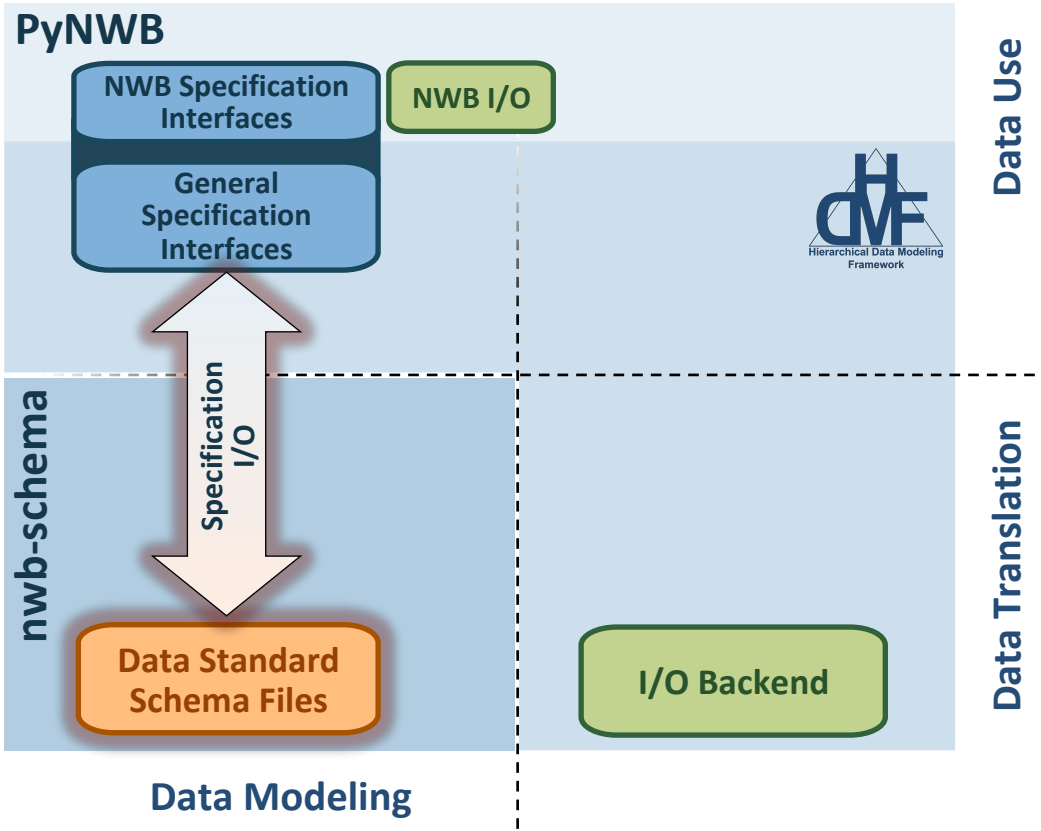


- Uses constant number of datasets/groups
- Scales to millions of neurons
- Facilitates efficient parallel read/write

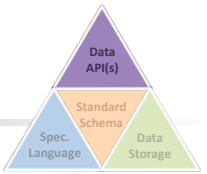
- Supports dynamic metadata
- Designed for NEURON data
- Compatible with SONATA

NWB:N data standard schema





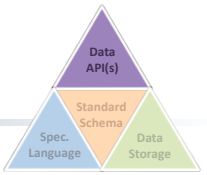
Python data API

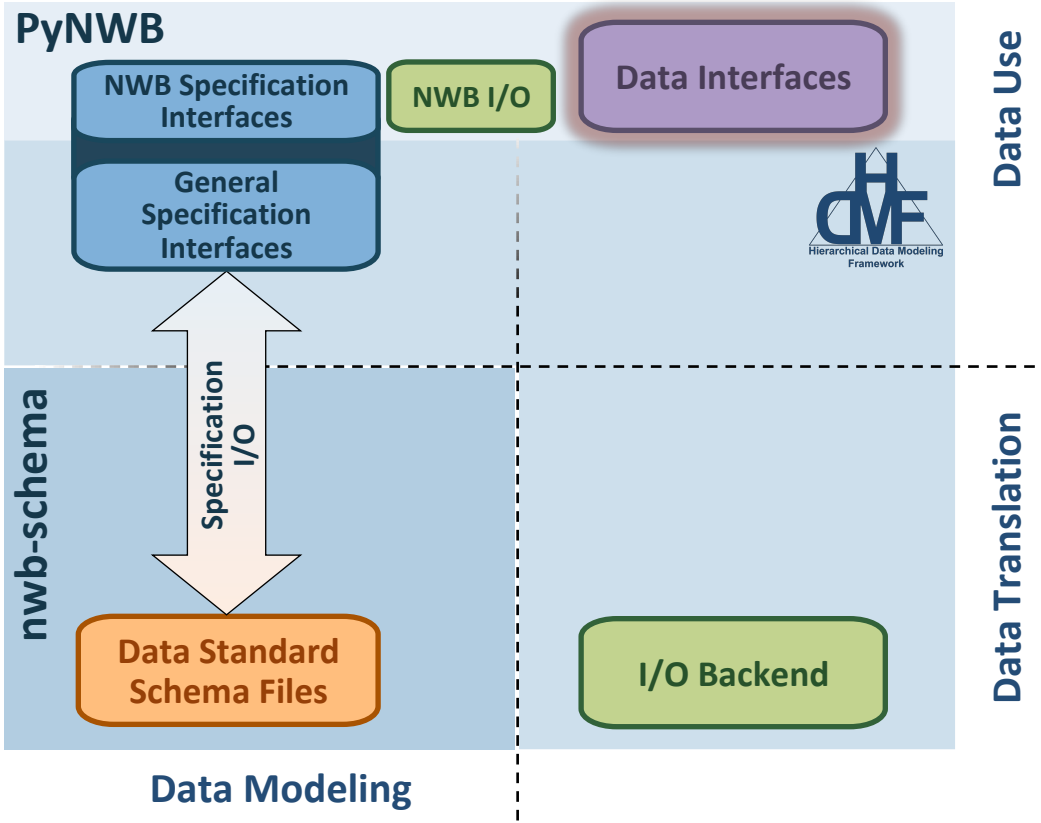


Container: Main in-memory data interface for applications:

- PyNWB has many of these -- one for each `neurodata_type` in the NWB:N standard schema
  - **Base classes and data structures:** pynwb.base,
  - **Main NWB:N file:** pynwb.file
  - **Electrophysiology:** pynwb.ecephys
  - **Optical Physiology:** pynwb.ophys
  - **Intracellular Ephys:** pynwb.icephys
  - **Opto-genetics:** pynwb.ogen
  - **Behavior:** pynwb.behavior
- For extensions PyNWB supports:
  - Automatic, dynamic generation of Container classes (i.e., no custom code is required to use extensions)
  - Creation of custom Container classes to represent new `neurodata_types`
- On read, data is loaded lazily

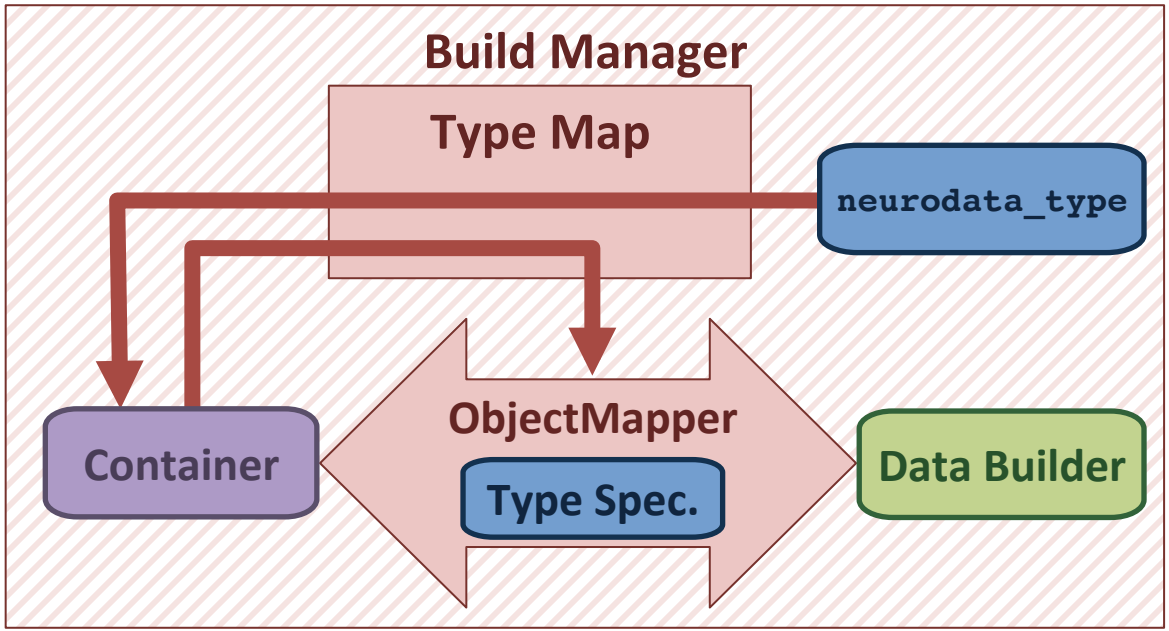
Python data API



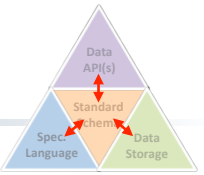


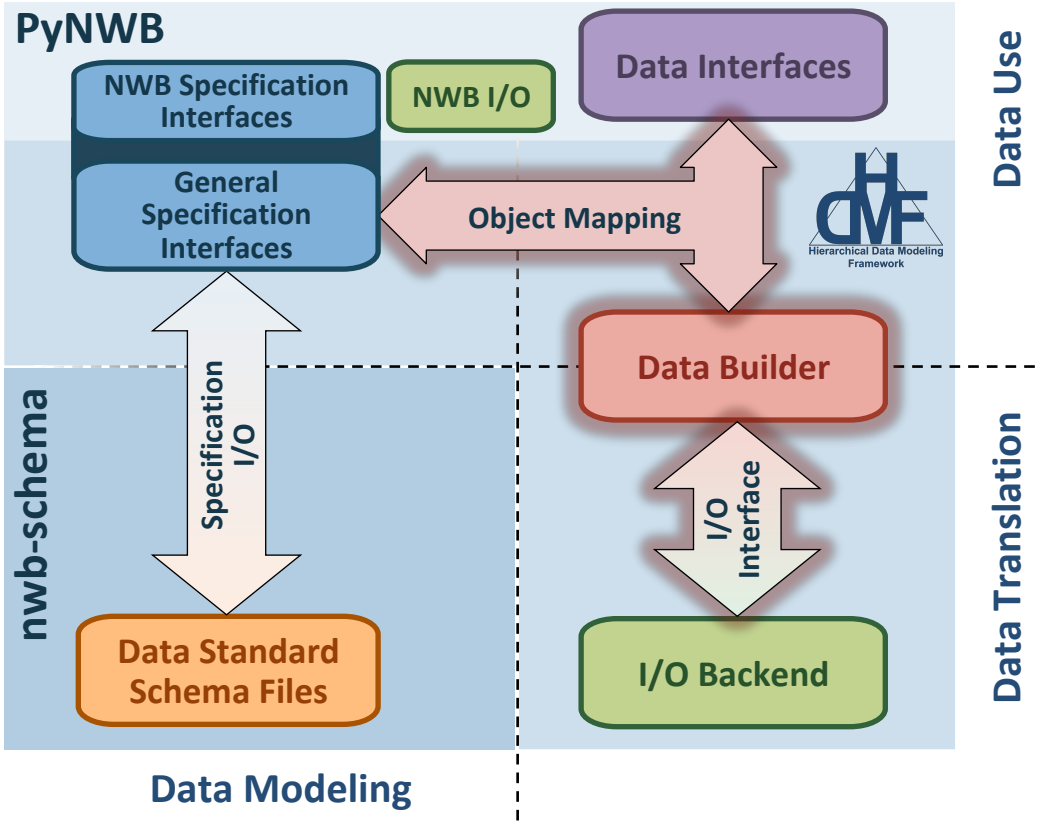
Python data mapping API

- **Builders:** Intermediary objects for I/O representing data primitives
  - GroupBuilder, DatasetBuilder, LinkBuilder, ReferenceBuilder, RegionBuilder
  - Backend readers and writers must return and accept these
- **ObjectMapper:** Map between Container attributes and specification components
- **TypeMap:** Map between:
  1. `neurodata_types`
  2. Container classes
  3. ObjectMapper classes
- **BuildManager:** Memoize Builders and Containers

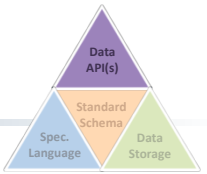


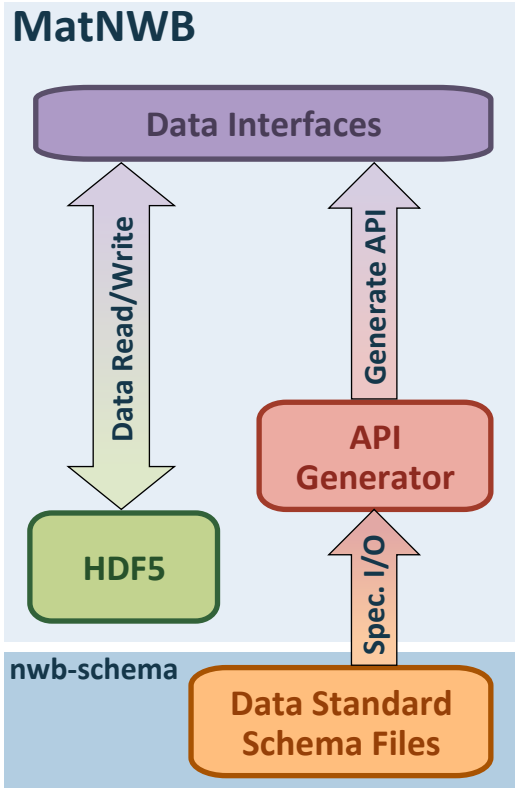
Python software stack





Matlab data API

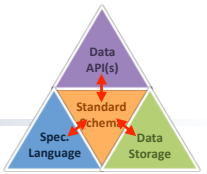


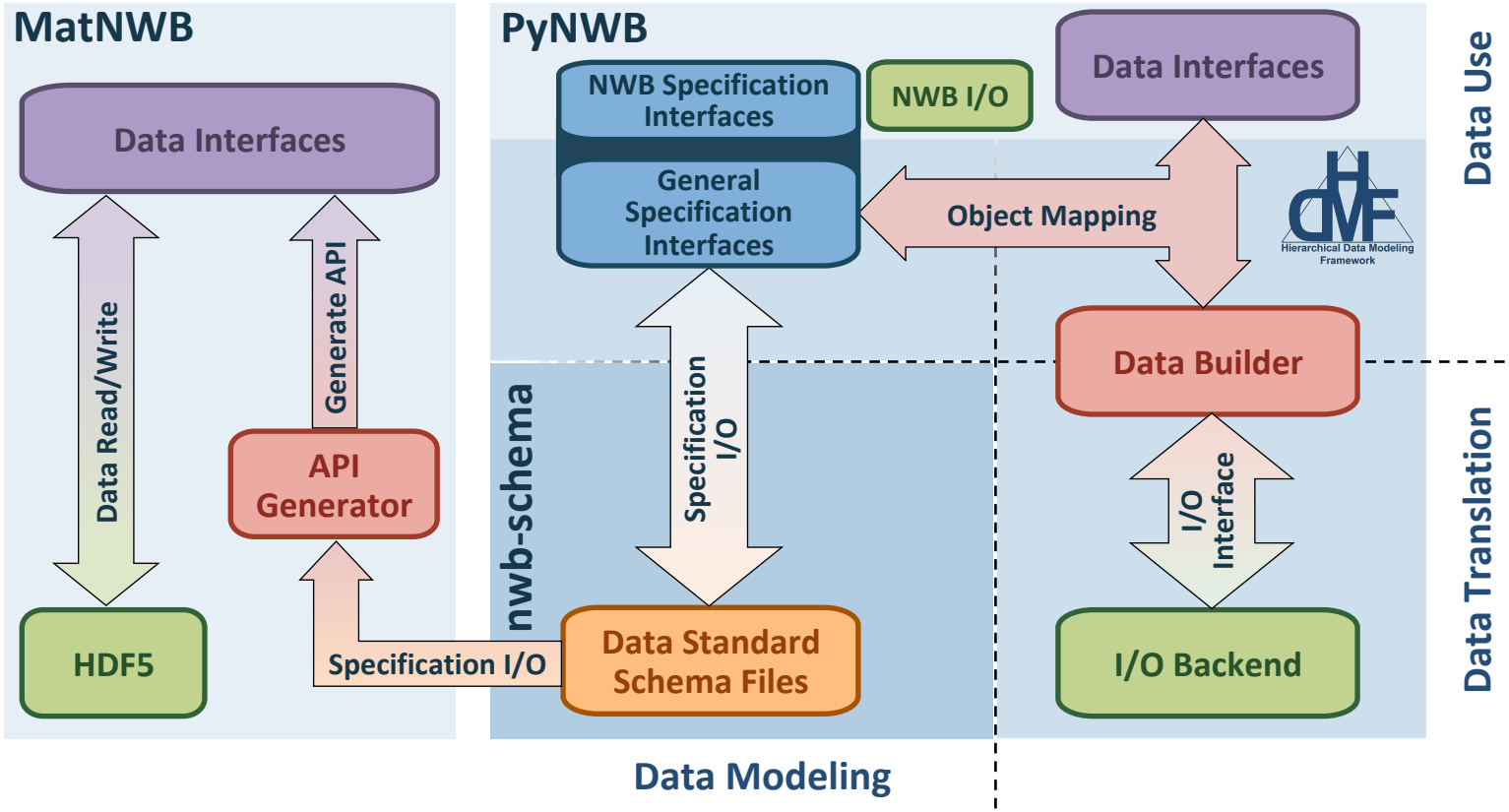


MatNWB generates from the NWB:N yaml standard schema files Matlab m-files defining classes that reflect the `neurodata_types` defined by the schema (i.e., similar to the Container classes in PyNWB)

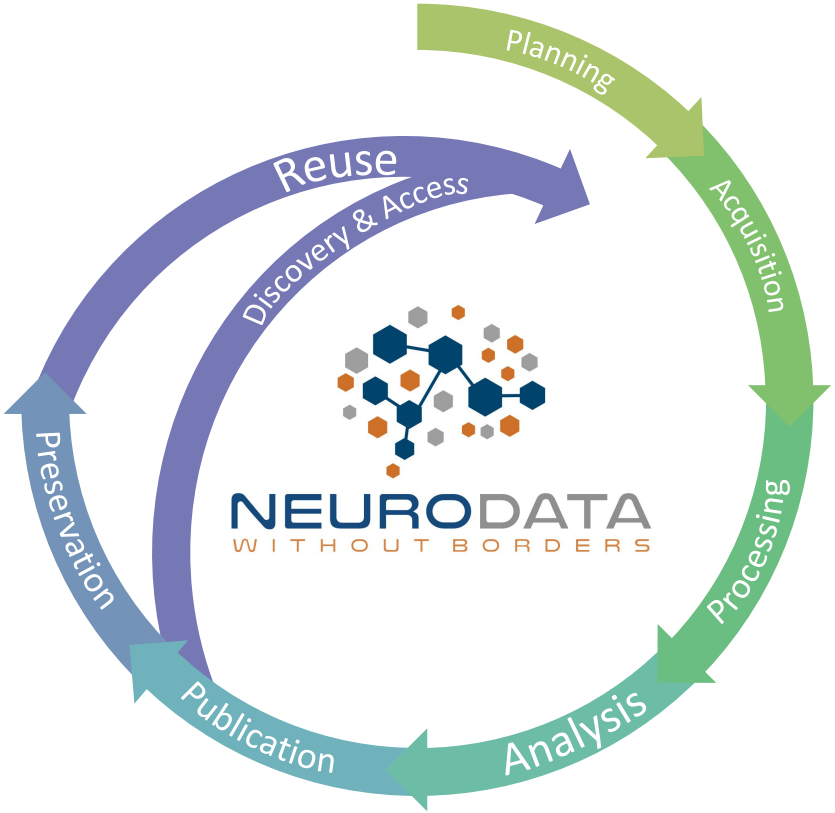
- MatNWB provides two main functions for this: `generateCore` and `generateExtension`
- Object attributes, relationships, and documentation are automatically generated to reflect the schema
- Supports construction, read and write of NWB:N files, including extensions

Advanced software architecture for data standardization





NWB:N technologies at the heart of the neurodata lifecycle

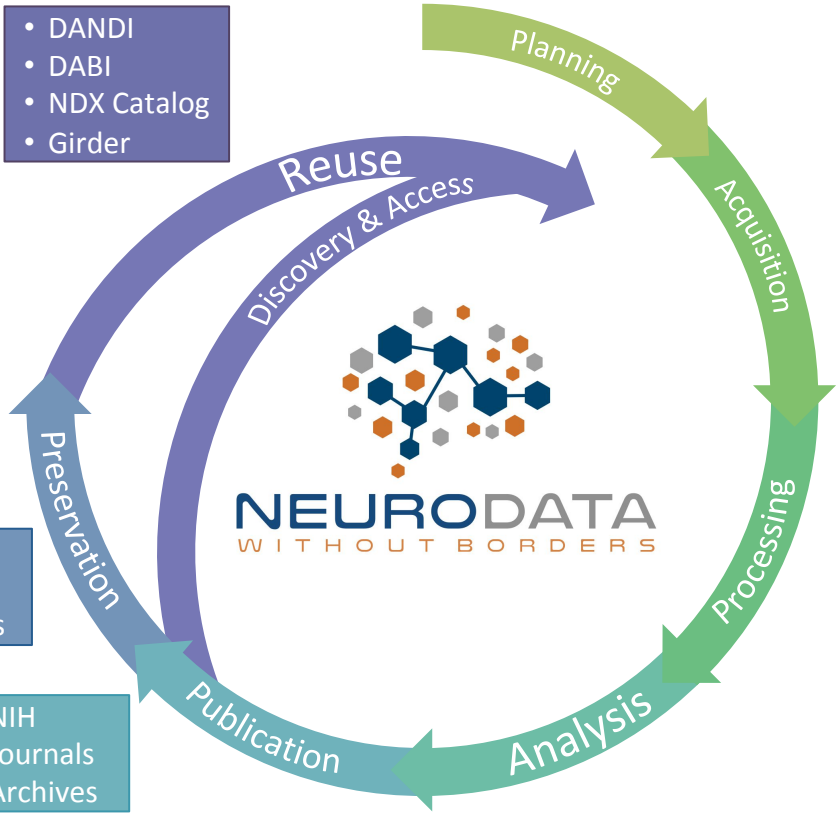


- Data standards are a critical conduit that facilitate:
  - Flow of data throughout the data lifecycle
  - Integration of data and software across phases of the data lifecycle
- NWB:N needs to support the needs of and integrate with technologies across the data lifecycle:
  - Work with (not compete with) existing and emerging data technologies
- NWB:N is a data standard for (not a standard of) neurophysiology experiments

Neurophysiology Tools

NWB:N technologies at the heart of the neurodata lifecycle and applications

- DANDI
- DABI
- NDX Catalog
- Girder



- HDF5
- Zarr
- Archives

- NIH
- Journals
- Archives

Electrophysiology	Optical Physiology
DataJoint, NWB Tutorials	
OpenEphys, Plexon, Neuralynx, Intan, TDT, SpikeGadgets, SpikeGLX	MiniScope ScanImage
PyNWB, MatNWB, HDMF, HDF5 tools/libs	
SpikeInterfaces MountainSort, KiloSort, npzsorting BrainStorm WaveClust, UltraMegaSort2000, KiloSort Kluster	NoRMCorre CNMF-E CELLMax EXTRACT
NWB-Explorer (OpenSourceBrain) NWB-JupyterWidgets	
BrainStorm, RAVE, Neo, SpikeWidgets, EcogVis, ephys-viz, …	CaImAn

Community engagement and outreach

Science Engagement

- **NWB User/Developer Days:**
  - 65+ scientists from 20 major institutions attended in 2018 at LBNL and AIBS
  - 43 users and developers from 29 major labs and research institutions in 2019 at HHMI
- The NIH proposal included 55 letters of support
- Many groups are already adopting NWB:N, e.g., FrankLab, ChangLab (UCSF), BouchardLab (LBNL), SvobodaLab (HHMI), MeisterLab (CalTech), Redwood (UCB), Allen Institute for Brain Science, among others.
- Broad outreach to tool builders:
- Broad outreach to foster publication of data:
- We have reached out to and are working with NIH BRAIN Initiative U19 projects
- KAVLI seed grants
- Simons data pilots

Industry Engagement

- Kitware (Visualization/CI)
- Vidrio (MatNWB)
- MathWorks (MATLAB)
- Vathes (DataJoint)

NEURODATA WITHOUT BORDERS

Governance

- NWB Executive Board
- NWB Technical Advisory Board
- KAVLI foundation
- NIH BRAIN Initiative

Community Resources

Open Source

GitHub
https://github.com/NeurodataWithoutBorders

Open Documentation

Read the Docs
https://neurodatawithoutborders.github.io

Developer/User Channels

slack
https://nwb-users.slack.com

Google Groups
https://groups.google.com/forum/#!forum/neurodatawithoutborders

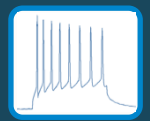
User Engagement

- User- and developer hackathons
- Outreach at neuroscience conferences (CoSyNe, SfN, BRAIN Initiative Investigators Meeting)
- Community discussion and surveys

Multi-disciplinary team science at work!

Applications


Extracellular electrophysiology


Intracellular electrophysiology


Optical physiology


Behavior

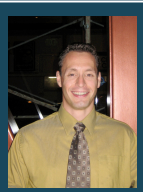

Simulations

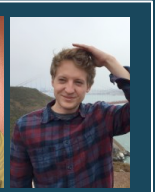
Project Team

LBNL


O. Rübel


A. J. Tritt


K. Bouchard


M. Dougherty

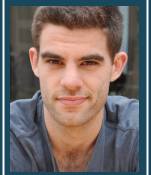

Ryan Ly

UCSF


E. Chang


L.M. Frank

Dichter LLC.


B. Dichter

Allen Institute for Brain Science


L. Ng


N. Graddis


N.Cain

byte physics

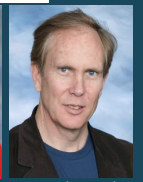

T. Braun

Broad collaborations

Kitware: Visualization and CI


M. Grauer


J.-C. Fillion-Robin


W. Schroeder


D. Ozturk

Vidrio/MathWorks: MatNWB


N. Clack


L. Niu

Vathes: Data Management


D. Yatsenko

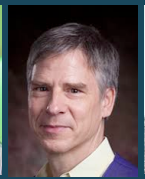

T. Nguyen

Applications and EB


K. Svoboda (HHMI)


F. Sommer (UCB)


M. Meister (Caltech)


C. Koch (AIBS)


K. Amunds (HBP)

Sponsors

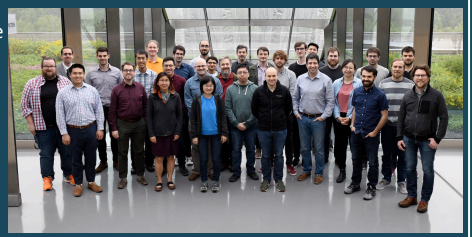
NIH The BRAIN Initiative®

THE KAVLI FOUNDATION

SIMONS FOUNDATION

Broader User and Developer Community

Apologies if your name/ team is missing! This slide only shows a very rough cut of some of the teams and people that work on developing NWB:N. Pictures and names of many, many important members of the NWB:N community are missing!



NEURODATA WITHOUT BORDERS

BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY

Online resources

Visit use online at nwb.org and neurodatawithoutborders.github.io

Contribute:

Create issues for bugs, feature request and pull requests on GitHub for schema, PyNWB, HDMF, and MatNWB. For details see: neurodatawithoutborders.github.io/contributing

Coming Soon:

nwb-extensions.github.io

Legal

- **Disclaimer:** This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

-