Neurodata Without Borders – An Ecosystem for Standardizing Diverse Neuorophysiology Data

Andrew Tritt

Computational Research Division

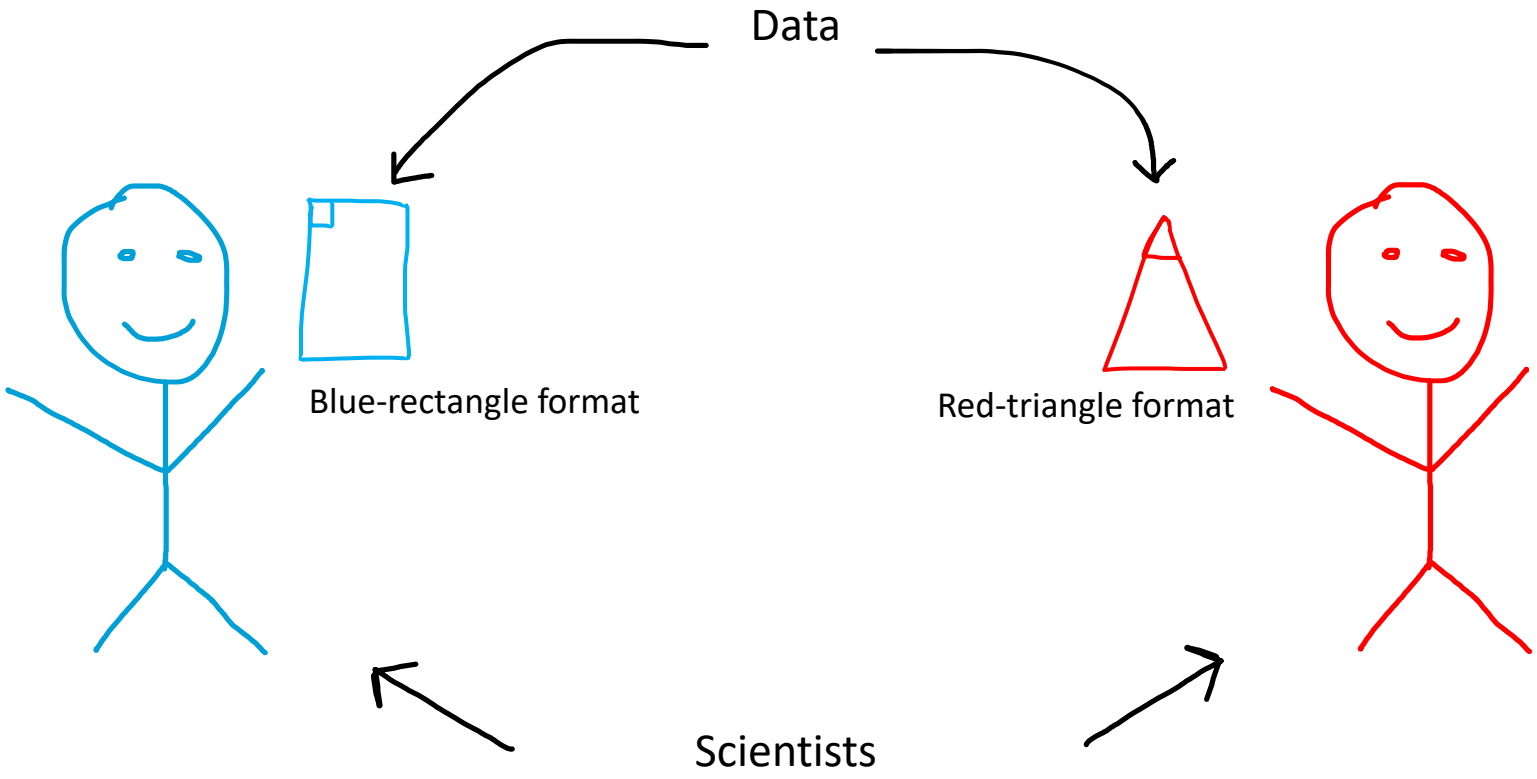
Lawrence Berkeley National Laboratory

HDF5 Users Group 2020

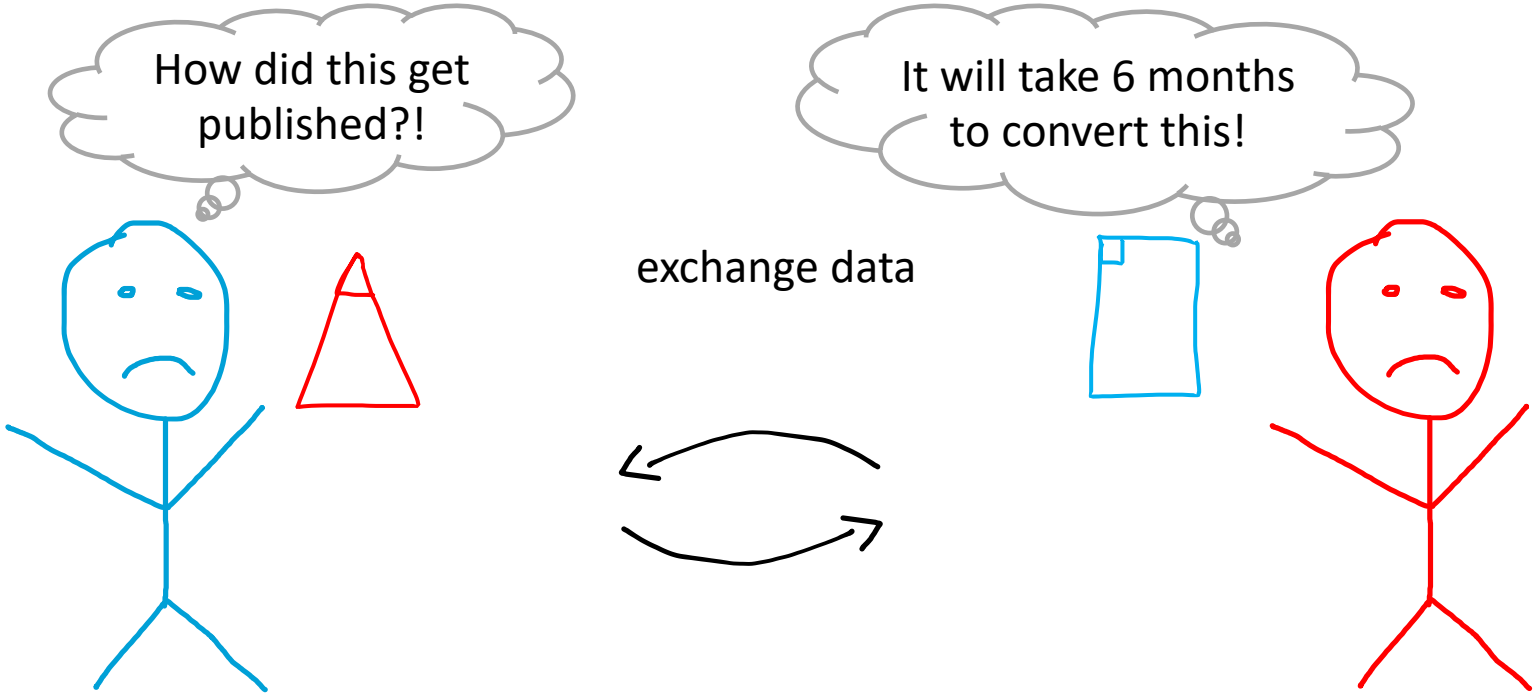
October 16, 2020

nwb.org

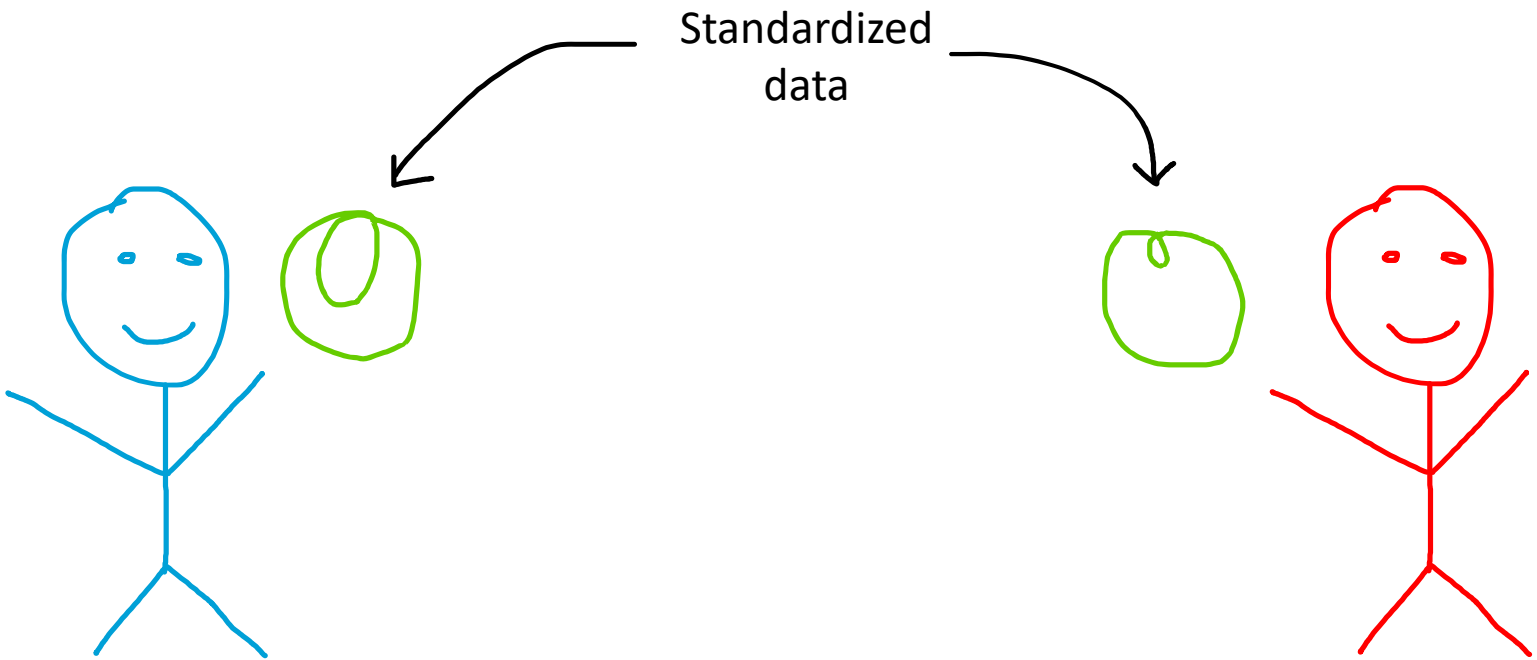
Community science…



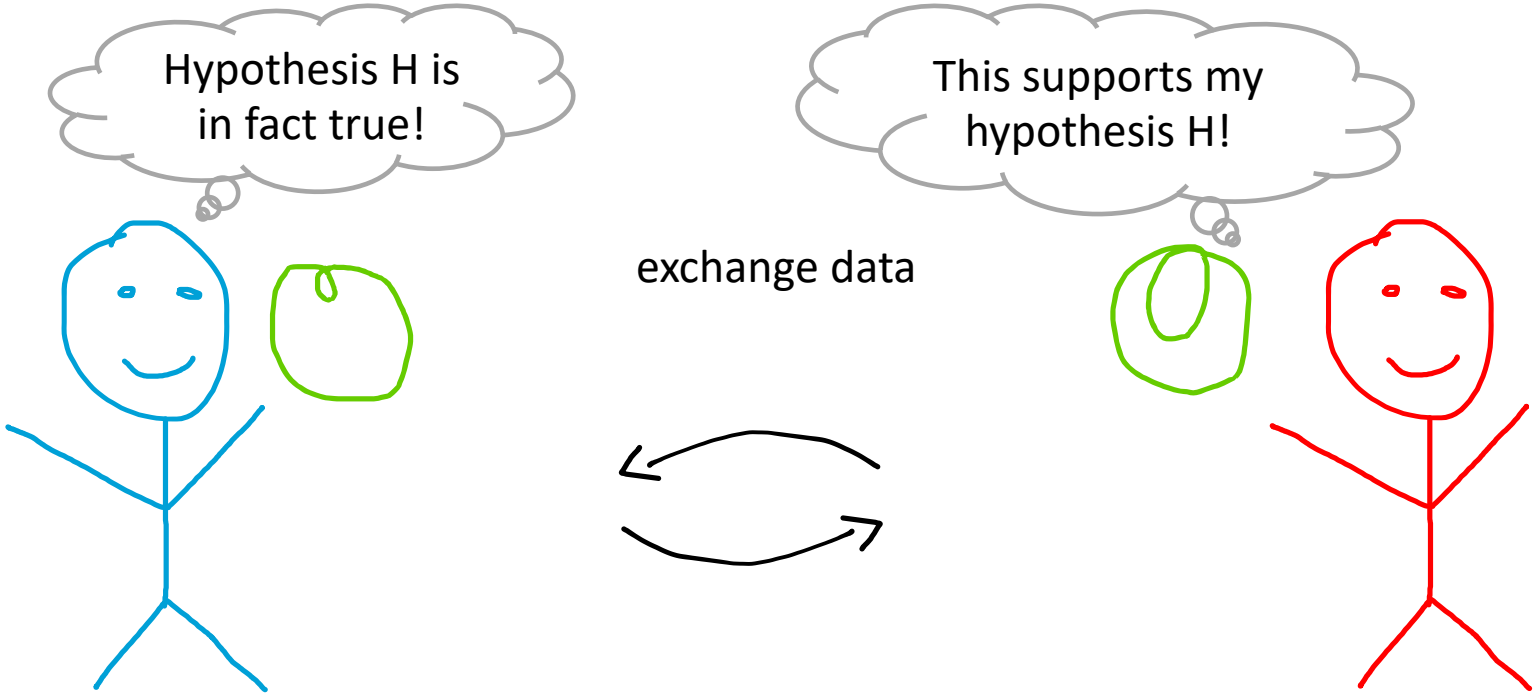
Community science…



Community science…



Community science…



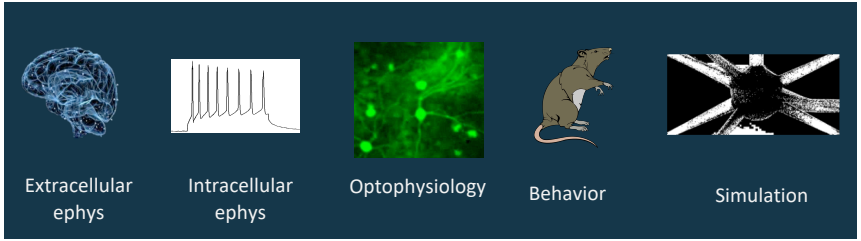
Neurodata Without Borders

Motivation: community science needs a stable target for data

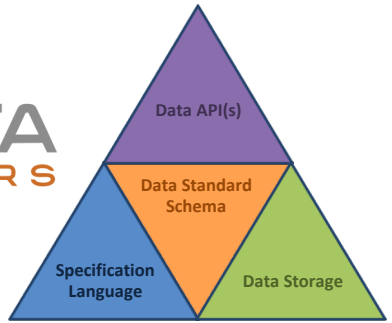
NWB – An Ecosystem for Neurophysiology Data Standardization

- Defines a unified format for neurophysiology data
  → focus on dynamics of groups of neurons

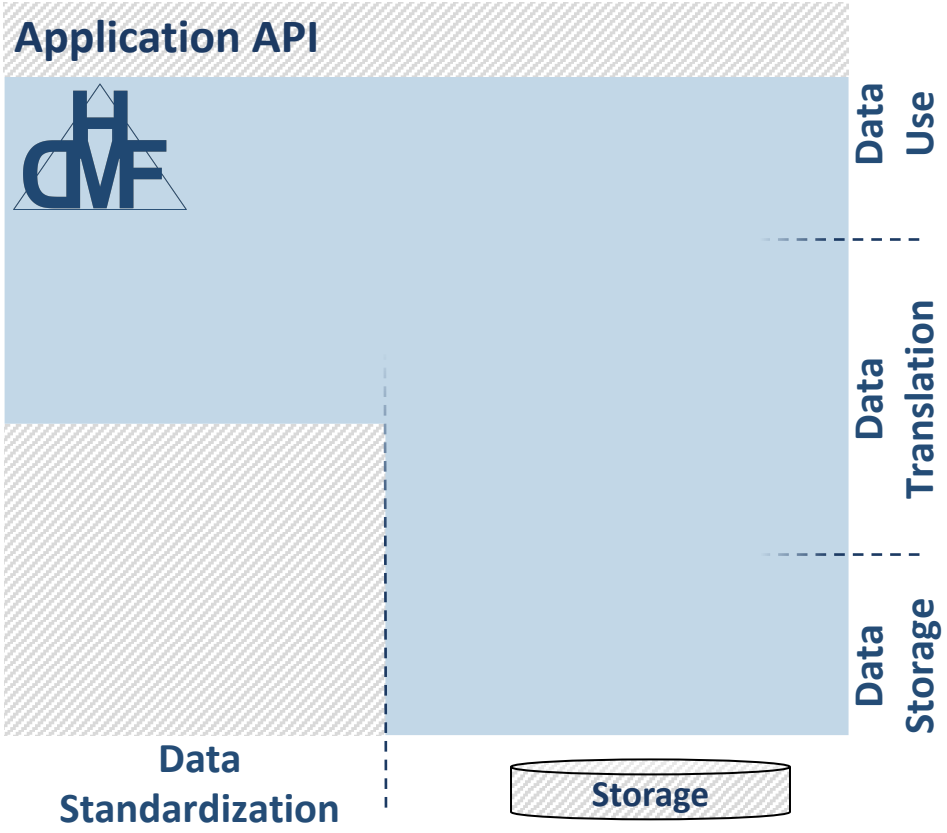
- More than just a file format
  - tools, methods, and standards for storing, sharing, and analyzing complex neurophysiology data

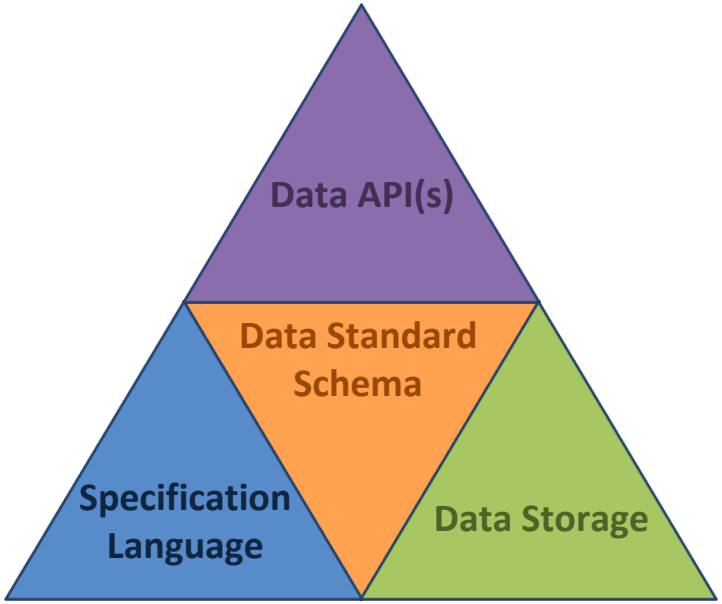


NEURODATA WITHOUT BORDERS



HDMF Software Stack





Specification language

Goal:

- Enable the formal definition of data standards that is *machine and human* readable

Format specification language

- Schema for defining hierarchical data schemas

Main primitives of the specification language:

- **Object primitives**
- **Data type specifications**
- **Namespace specification**

Specification language

Object primitives:

- **Group:** collection of objects i.e. subgroups, datasets, links
- **Dataset:** n-dimensional array with associated data type, dimensions, etc.
- **Attribute:** metadata attached to a Group or Dataset
- **Link:** a soft link to given target data_type

Specification language

Data type specifications

- **Basic data types:** strings and numeric types
- **Compound data types:** complex data types
- **isodatetime:** ISO8061 date-time string
  - e.g. 2018-09-28T14:43:54.123+02:00
- **Object reference:** a link that can be as a value
- **Region reference:** a link to a dataset that also stores slice

Defining reusable data types

`data_type`: Defines reusable type

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

Enables reuse of types through inheritance and:

- **`data_type_inc`**: include an existing type
- **`data_type_def`**: defines a new type

A schema is a collection of reusable `data_types`

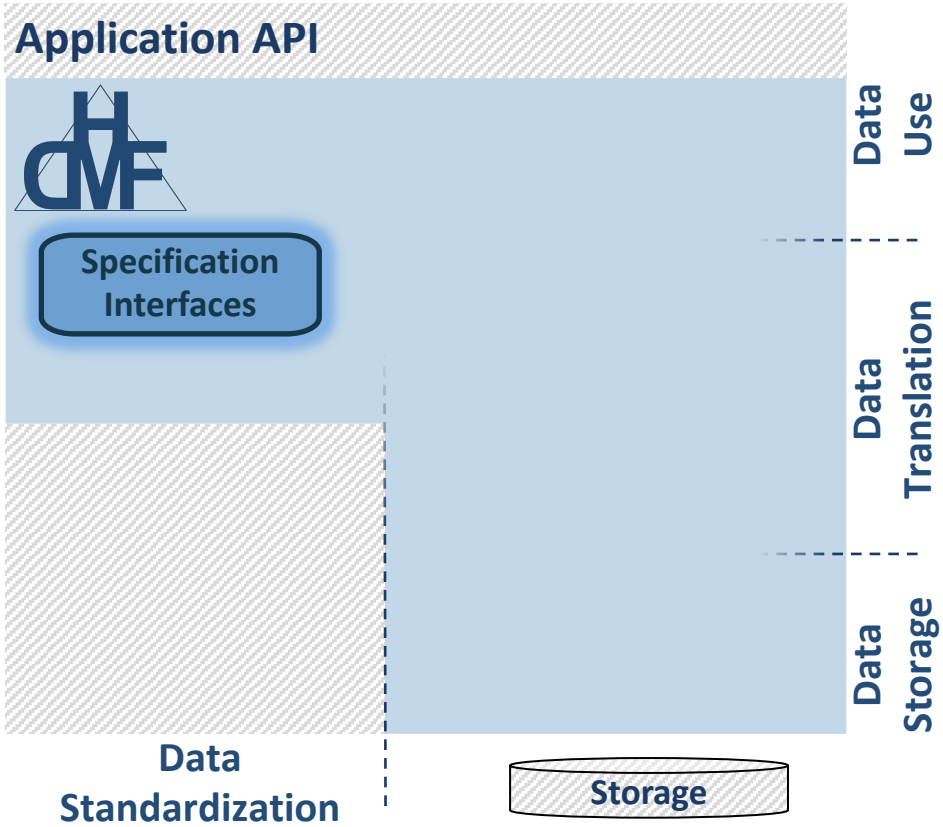
- Extend by:
  - defining new `data_types`
  - extending existing `data_types`

Specification language

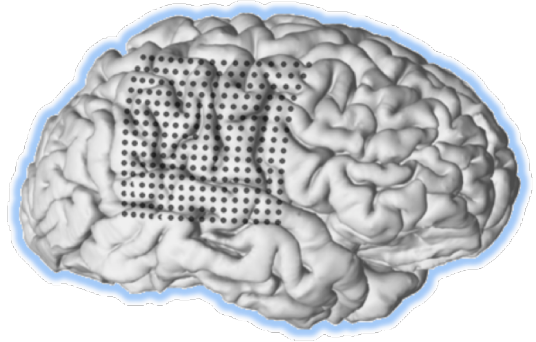
Namespace specification:

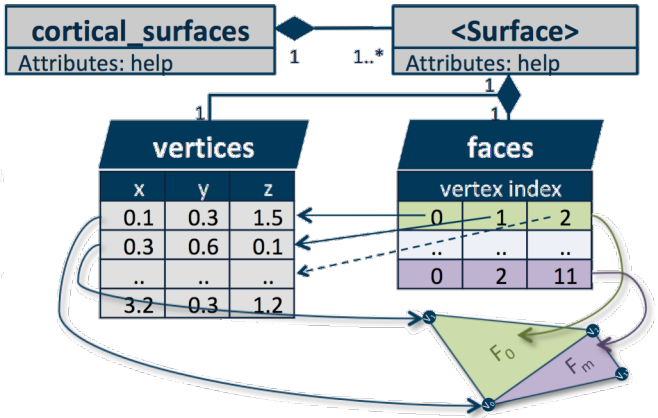
- define a standard
- insulate extensions from each other
- enable the creation of new data standards
- extensions shared in the form of a namespace

Specification Language API



Specification



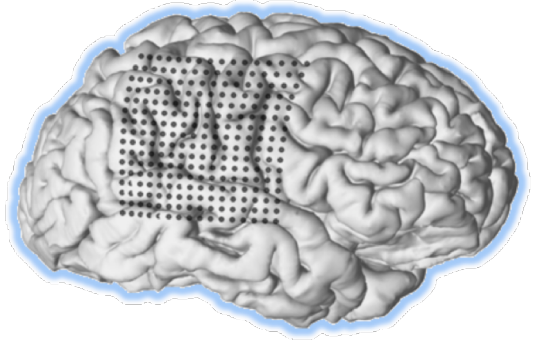


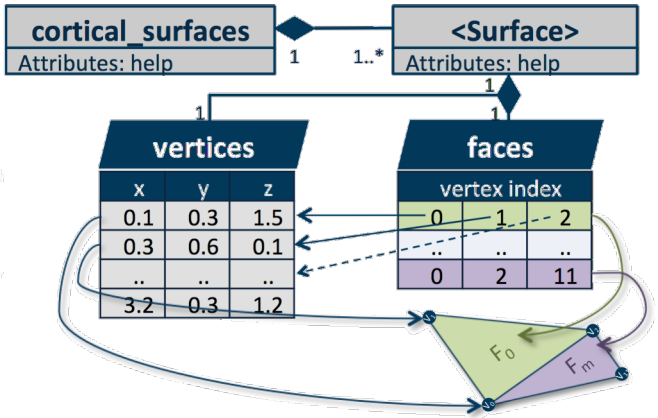
Creating the extension

```
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')
```

Specification





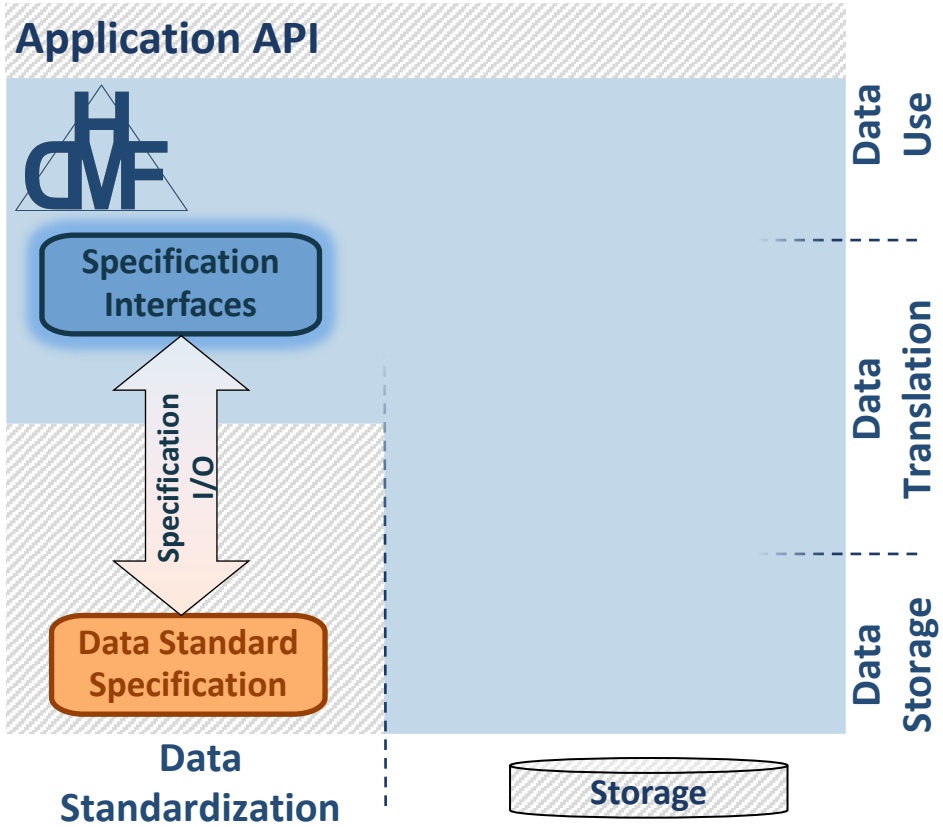
Write data

```
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)
```

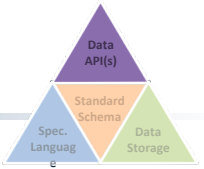
Read data

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

Specification Language API



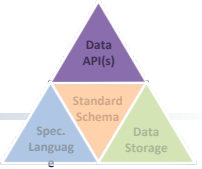
Data API

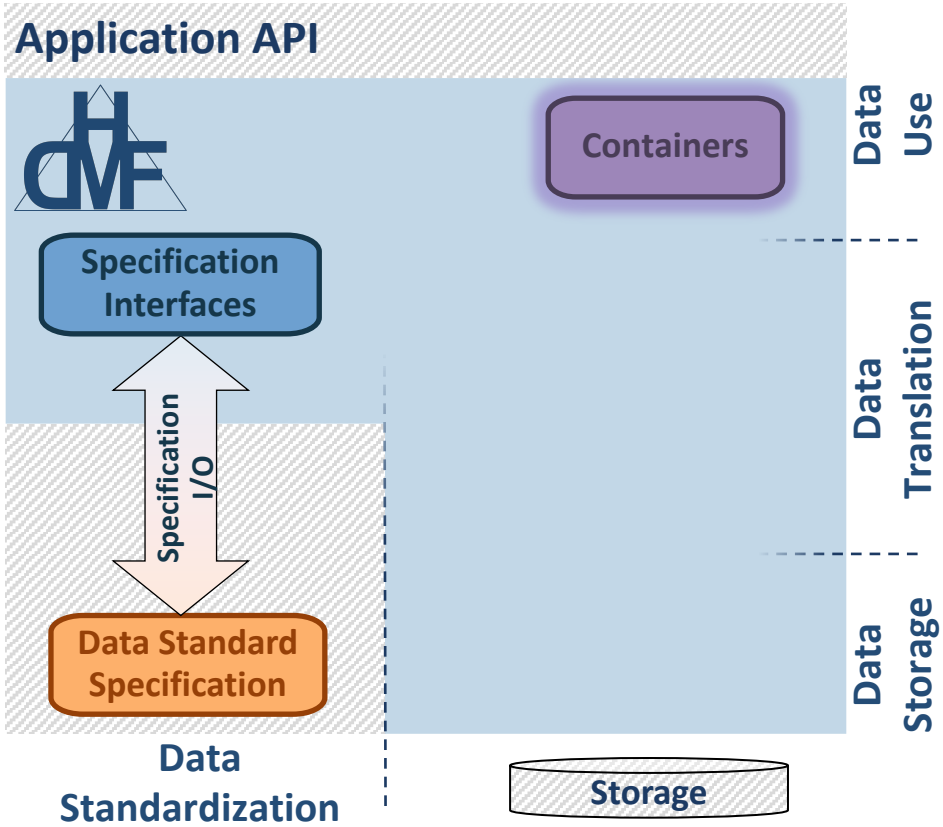


Container: Main in-memory data interface for applications

- One for each `data_type` in the standard schema
  - Class dynamically generated if one does not exist
- HDMF provides base classes and decorators to facilitate the implementation of new Container classes
- Containers hold handles to on-disk data → data lazily loaded
  - backend dependent

Data API


Data API(s)
Standard Schema
Spec. Languag e
Data Storage


Application API
Containers
Specification Interfaces
Specification I/O
Data Standard Specification
Data Standardization
Storage
Data Use
Data Translation
Data Storage

Data storage

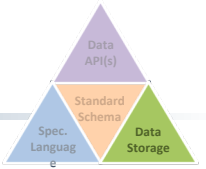
Primary function:

- Map primitives to storage
- Decouple backend from specification and in-memory data structures

- **Builders:** Intermediary objects for I/O representing data primitives
  - GroupBuilder, DatasetBuilder, LinkBuilder, ReferenceBuilder, RegionBuilder

- **HDMFIO:** I/O Interface for reading/writing builders
  - Need to define: read_builder, write_builder, open, close, and __init__ functions
  - HDF5IO and ZarrIO

Data storage

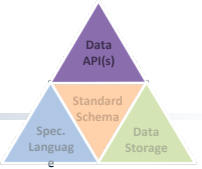


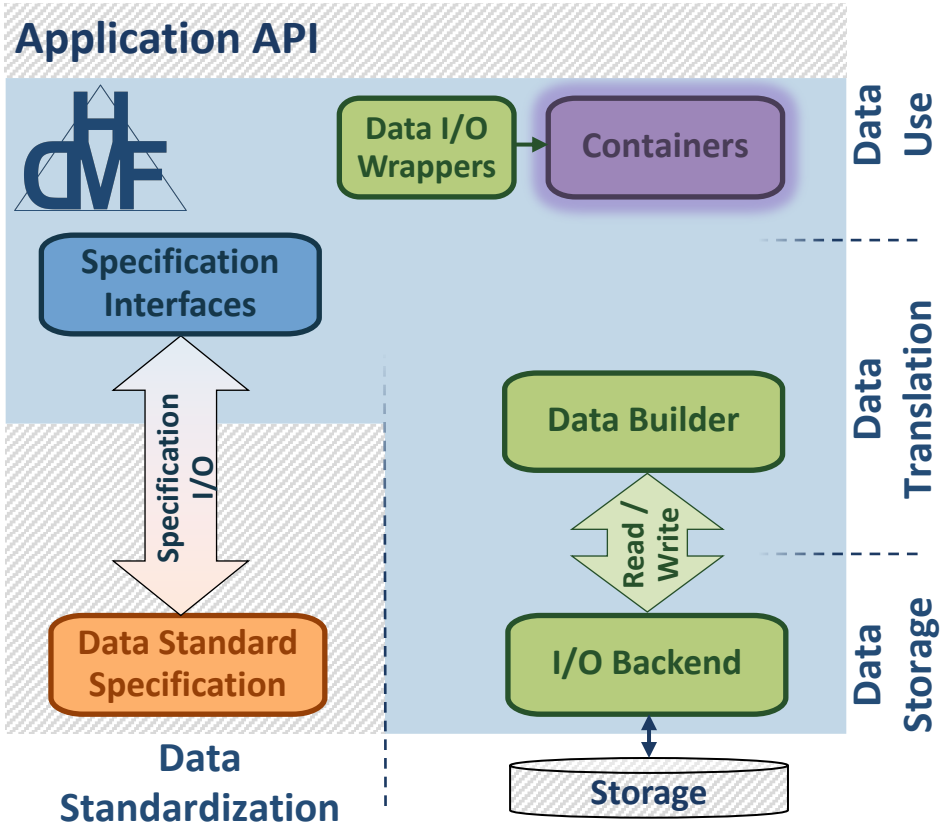
Primary function:

- Map primitives to storage
- Decouple backend from specification and in-memory data structures

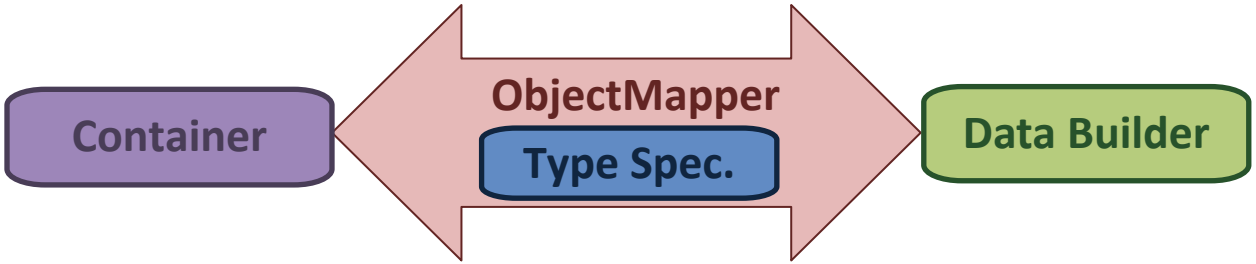
- **Data I/O wrappers:**
  - **DataIO:** Define chunking, compression, linking, modular storage etc. to optimize storage and I/O
  - **DataChunkIterator:** Iterator to produce data chunks for iterative data write (e.g., for data streaming or large data import)

Data API


Data API(s)
Standard Schema
Spec. Languag e
Data Storage

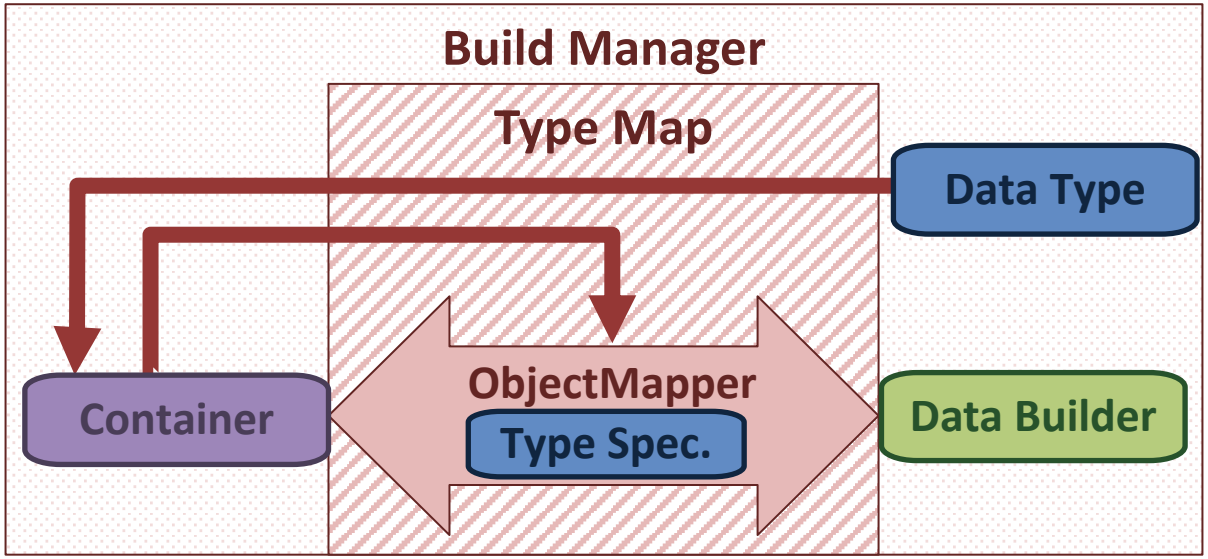

Application API
Data I/O Wrappers
Containers
Specification Interfaces
Specification I/O
Data Standard Specification
Data Builder
Read / Write
I/O Backend
Storage
Data Use
Data Translation
Data Storage
Data Standardization

ObjectMapper



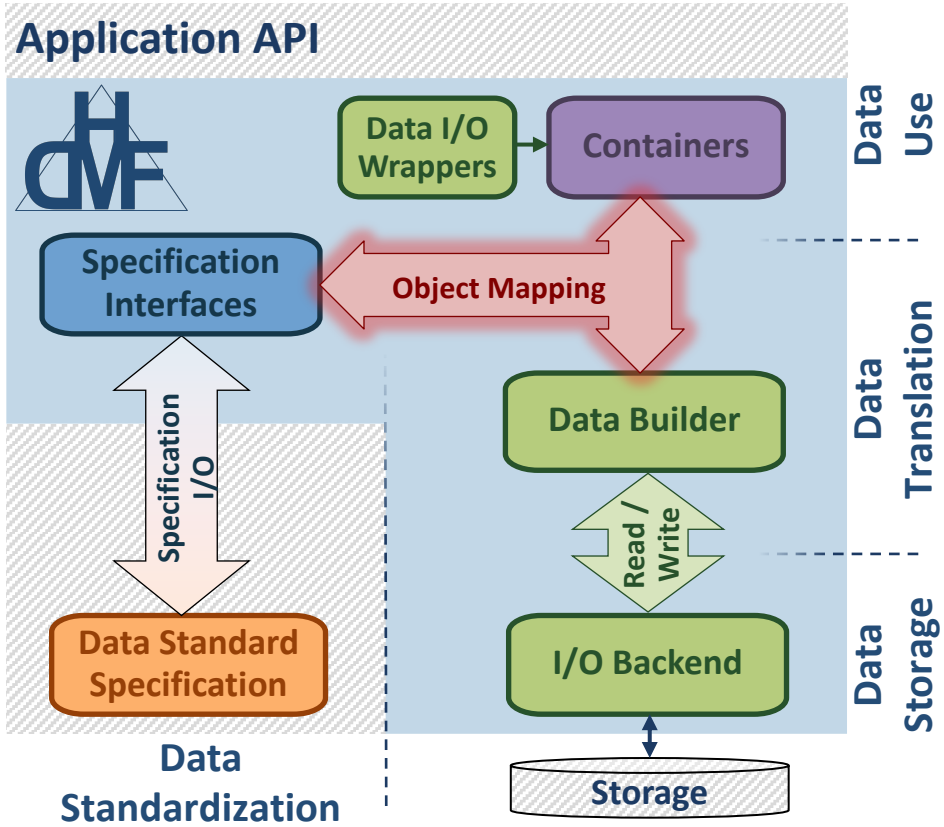
- Maintains mapping between *Container attributes* and *specification components*
- Constructed from a `data_type` specification
- One for each `data_type`
  - base functionality or provided by user

Data mapping API



- **TypeMap:** Map between: `data_types`, Container classes and ObjectMapper classes
  - Given `data_type`, return Container class
  - Given Container class, return ObjectMapper
- **BuildManager:** Responsible for memoizing Builders and Containers
  - Given a Builder, return a Container and vice versa

HDMF Software Stack



Neurodata Without Borders

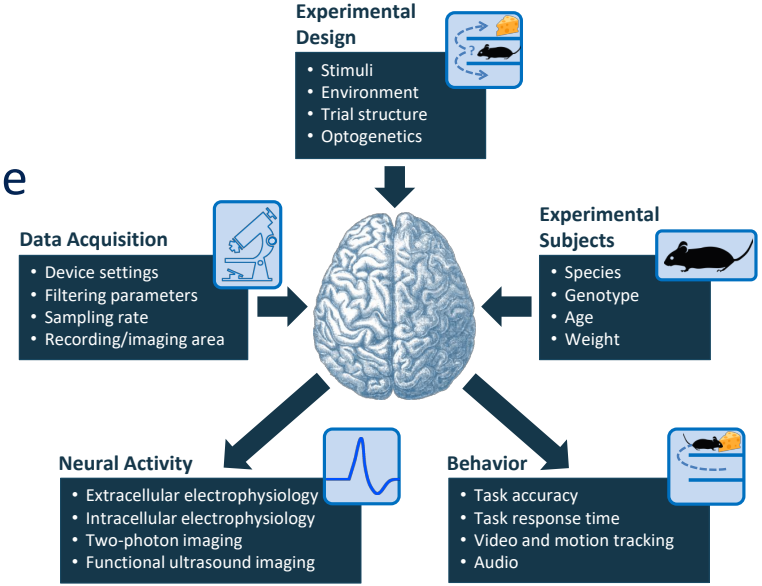
A format schema (a.k.a. specification)

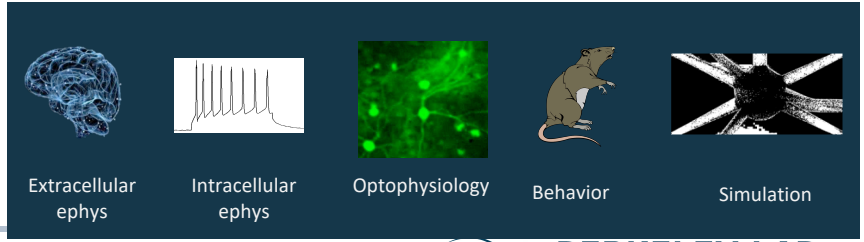
- a collection of YAML files that formally describes the organization of data
- covers data types throughout data life cycle

The **NWB data standard** defines:

- **A catalog of `neurodata_types`**
  - 20 different TimeSeries data types
  - 22+ analysis data types

- **Base hierarchy to logically group data**
  - acquisition, data processing, experimental metadata





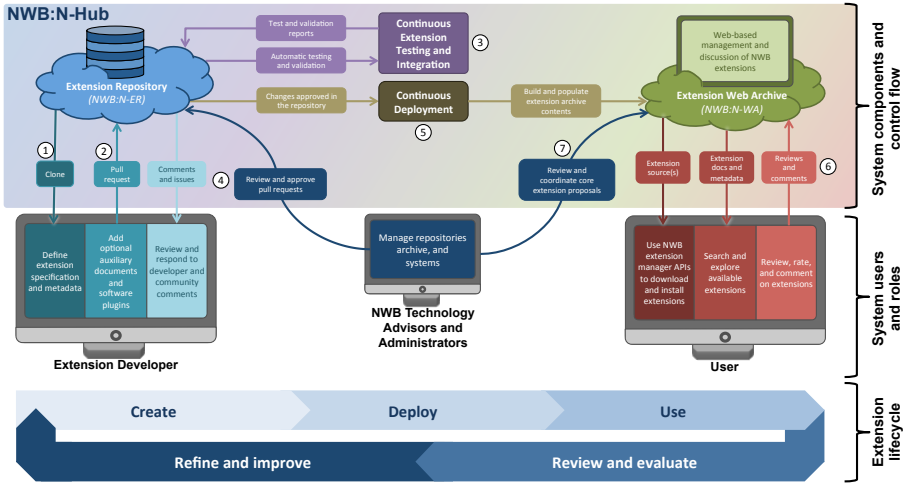
NDX Catalog

Goal: Enable community adoption, extension, and curation of NWB and integration of **new use cases**

Neurodata Extensions Catalog

Approach:

- Tools for creating extensions (PyNWB, nwb-docutils, git template)
- Recipe for deploying extension
- Online catalog for sharing
- Guidelines for versioning, sharing, and review of extension



Resources: https://nwb-extensions.github.io/

Online resources

Visit us online at

neurodatawithoutborders.github.io

and

nwb.org

Contribute

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