An advanced data software architecture for Neurodata Without Borders (NWB) to enable efficient management, use and sharing of neurophysiology data

O. Rübel[1*], A. Tritt[1*], D. Camp[1], E.F. Chang[3], D. Donofrio[1], L.M. Frank[4], F.T. Sommer[5], K. Bouchard[2]

[1] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley CA
[2] Biological Systems and Engineering Division, Lawrence Berkeley National Laboratory, Berkeley CA
[3] Department of Neurological Surgery, UC San Francisco, San Francisco, CA
[4] Department of Physiology, UC San Francisco, San Francisco, CA
[5] Redwood Center for Theoretical Neuroscience, UC Berkeley, Berkeley, CA
[*] These authors have contributed equally to this work

Overview

To maximize the return-on-investment into creation of neuroscience data sets and enhance reproducibility, it is critical to share data through standardized and extensible data model and management solutions. In addition to standardizing data and metadata, support for fast data read/write and high-performance, parallel data analysis are critical to enable labs to keep up with ever growing data volumes. The Neurodata Without Borders: Neurophysiology (NWB:N) effort was an important step towards generating a unified data format for cellular-based neurophysiology data for a multitude of use cases. To enable broad adoption of NWB:N, easily accessible tools and an advanced software strategy aimed at facilitating the use, extension, integration, and maintenance of NWB:N are critically needed. The KAVLI-sponsored NWB-4-HPC project aims to ensure that the software instantiation of NWB:N adheres to these principles and enables efficient management and processing of large-scale neuroscience data sets. Here, we apply software engineering principles to create an advanced software architecture and define abstractions to enable separation of the NWB specification language, format specification, data storage, and data API(s).

Specification Language:
How to formally define neuroscience data standards?

To support the formal and verifiable specification of neurodata file formats, NWB:N defines and uses the NWB:N specification language. To organize complex data, the specification language uses easy-to-use primitives, e.g.: Groups, Datasets, Attributes, and Links. For NWB:N 2.0 we have simplified and extended the specification language to ease readability, interpretability and expressiveness. Specifically, we:

- Simplified the reuse of *neurodata_types* via inheritance and inclusion of types
- Added support for object- and region reference data types and improved specification of links, to ease explicit modeling of references between data
- Added support for compound data types, enabling the specification of tables and complex data types
- Added support for default names and values
- Replaced values encoded in keys with explicit key/value pairs to avoid collision of keys and enhance explicit human interpretation
- Enable storage and sharing of specifications via YAML (in addition to JSON)
- Developed tools to automatically generate human-readable documents from specs
- **Documentation:** http://schema-language.readthedocs.io
- **Release Notes:** http://schema-language.readthedocs.io/en/latest/specification_language_release_notes.html

Format Specification:
How to organize complex collections of neuroscience data?

The NWB:N schema is specified via the specification language and formally defines the organization of neuroscience data via the NWB:N data format. For NWB:N 2.0 we:

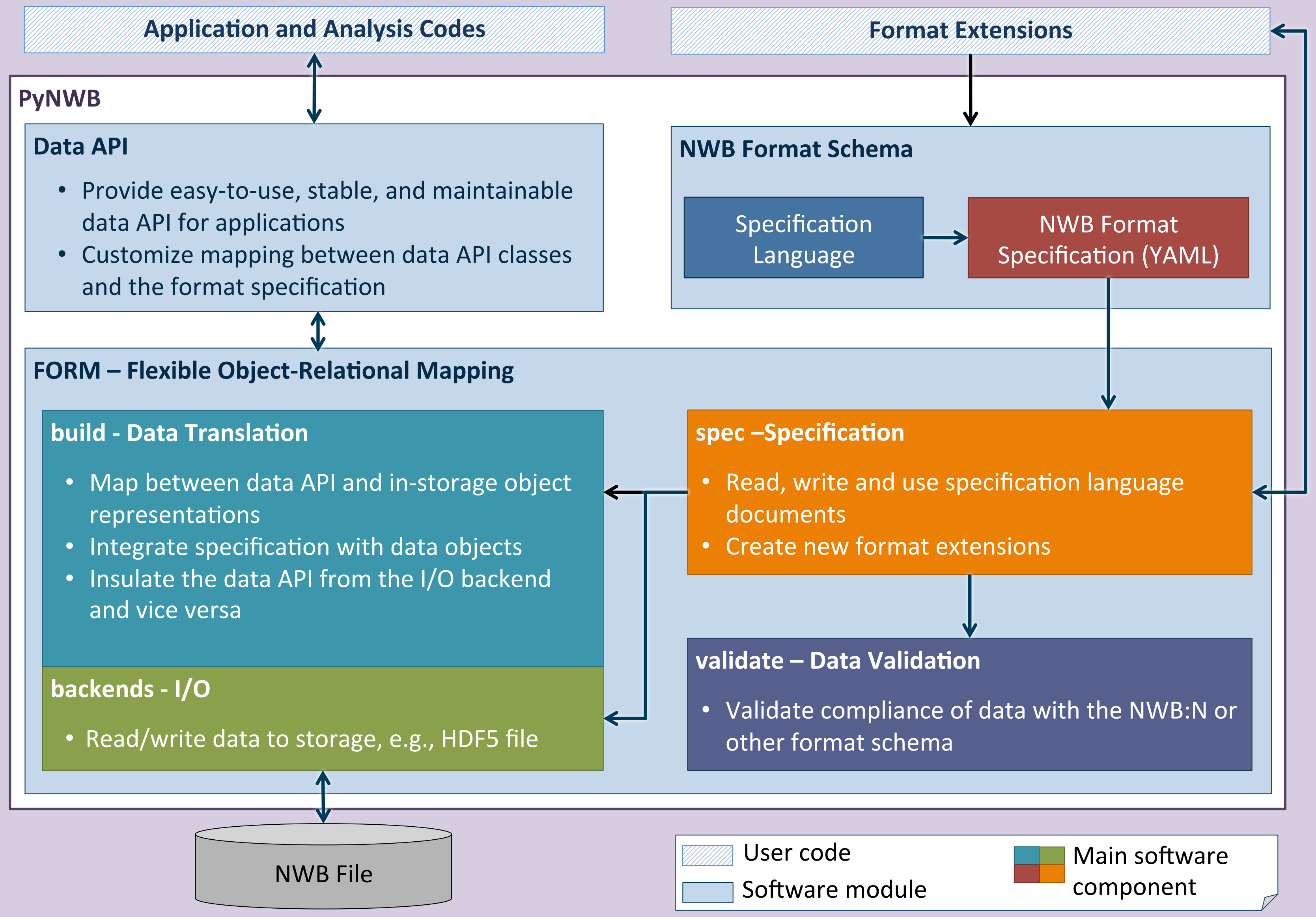
- Introduced the concept of *NWBContainer* and *NWBData* as common base neurodata types for improved data modeling and to improve and clarify the organization of processed data in NWB:N
- Reorganized electrode metadata to facilitate access, search, organization, and interpretation via a combination of:
  - A central *ElectrodeTable* for per-electrode metadata and
  - Accompanying *ElectrodeGroup* containers for collective electrode metadata
- Replaced implicit links and data-structures with explicit links and relationship models to facilitate unique direct human and programmatic data interpretation
- Assigned unique *neurodata_type* / *name* to all objects for improved identifiability
- Improved consistency and completeness of metadata and object names
- Improved governance of the schema via new formal release and documentation mechanisms using YAML and the dedicated nwb-schema GitHub repository
- **Documentation:** http://nwb-schema.readthedocs.io
- **Release Notes:** http://nwb-schema.readthedocs.io/en/latest/format.html#release-notes-nwb-format

Data Storage:
How to store large collections of neuroscience data?

A primary function of the data storage is to map NWB:N primitives (Groups, Datasets, Attributes, Links etc.) to storage. Currently NWB:N uses HDF5 as main storage format.

- By decoupling the various aspects of NWB:N, PyNWB enables the design of new storage backends for NWB:N, e.g., using file-system semantics or databases.
- We have created documentation describing the mapping of the NWB:N specification to HDF5: http://nwb-storage.readthedocs.io

PyNWB: Enabling users to efficiently interact with NWB:N data and format specifications



Application and Analysis Codes

Format Extensions

PyNWB

Data API
- Provide easy-to-use, stable, and maintainable data API for applications
- Customize mapping between data API classes and the format specification

NWB Format Schema

Specification Language → NWB Format Specification (YAML)

FORM – Flexible Object-Relational Mapping

build - Data Translation
- Map between data API and in-storage object representations
- Integrate specification with data objects
- Insulate the data API from the I/O backend and vice versa

backends - I/O
- Read/write data to storage, e.g., HDF5 file

spec –Specification
- Read, write and use specification language documents
- Create new format extensions

validate – Data Validation
- Validate compliance of data with the NWB:N or other format schema

NWB File

▨ User code
Software module
Main software component

Overview

With PyNWB we have developed a new modular software architecture and API to enable users/developers to efficiently interact with the NWB data format, format files, and specifications. This novel software architecture lays the foundation for the design of advanced APIs for data management, query and discovery, and integration of NWB:N with state-of-the-art data analytics codes optimized for high-performance computing systems. Importantly this architecture decouples the various aspects of NWB:N:

1) the specification language,
2) format specification,
3) storage backend, and
4) data API

This allows each to be used and maintained independently.

`pynwb:` An easy-to-use data API for NWB

PyNWB provides a stable, intuitive, object-oriented application programming interface (API) for NWB:N for development of user applications and analysis codes. The user API is independent of the storage backend and provides easy mechanisms for reading and writing of NWB:N data, including advanced I/O features, e.g., iterative streaming data write and data compression.

`pynwb.form:` Flexible object-relational mapping

The `form` module defines a general library for creating scientific data formats and builds the core infrastructure for PyNWB . Through its flexible object-relational mapping functionality, `form` allows us to decouple the data API, format specification, and I/O backends from each other enabling the flexible design of: 1) advanced user APIs, 2) new I/O backends, and 3) data formats and extensions.

`pynwb.spec:` Creating new extension for NWB:N

PyNWB provides convenient data structures for creating, reading, and writing format extensions using the NWB:N specification language. To avoid collisions between extension and facilitate versioning and documentation, extensions are organized via namespaces.

```
from pynwb import NWBNamespaceBuilder, NWBGroupSpec, NWBAttributeSpec

ns_builder = NWBNamespaceBuilder('Extensions for my Lab', "mylab")
ext = NWBGroupSpec('A custom ElectricalSeries for my lab',
                   attributes=[NWBAttributeSpec('trode_id', 'int','tetrode id')],
                   neurodata_type_inc='ElectricalSeries',
                   neurodata_type_def='TetrodeSeries')

ns_builder.add_spec("mylab.extensions.yaml", ext)
ns_builder.export("mylab.namespace.yaml")
```

`pynwb:` Reading/writing data using extensions

PyNWB allows users to create their own interface classes for extensions, but it also supports the automatic, dynamic creation of *NWBContainer* classes to allow users to directly use their extensions to read/write data without having to write additional code for PyNWB:

```
from pynwb import get_class, load_namespaces
# Load the namespace
ns_path = load_namespaces("mylab.namespace.yaml")
# Get the class for our extension
TetrodeSeries = get_class('TetrodeSeries', 'mylab')
```

`pynwb.validate:` Validating files

We can easily validate data files using the NWB:N core specification via:

```
python -m pynwb.validate myfile.nwb
```

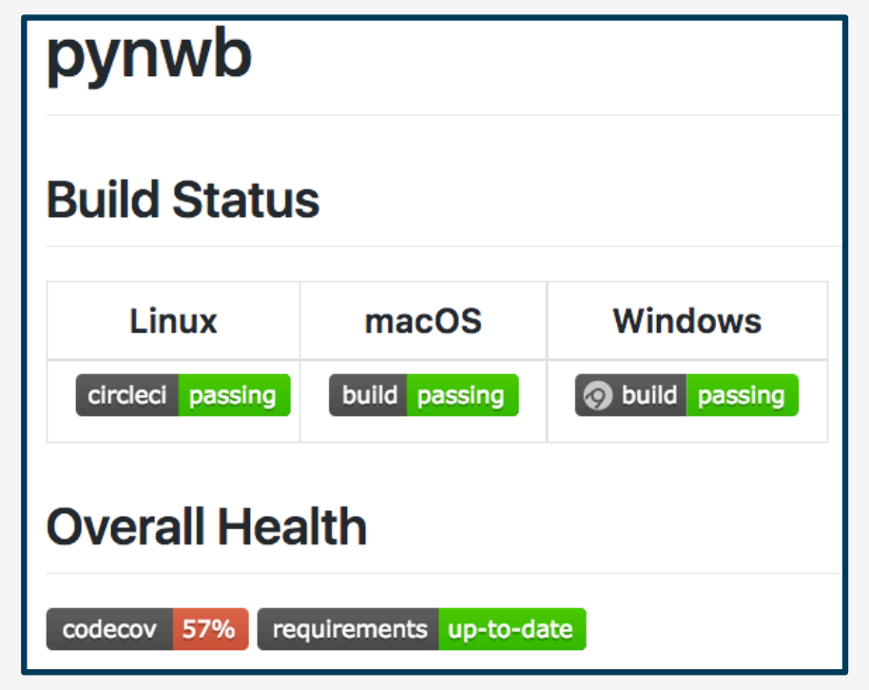
as well as using user-defined format extensions via:

```
python -m pynwb.validate -p mylab.namespace.yaml myfile.nwb
```

Working with the neuroscience community towards better science solutions

We continue to see a vibrant community forming around NWB:N. Below we highlight several collaborative efforts with and contributions by the NWB:N community related to the development of NWB 2.0 beta.

Using Modern Software Processes for PyNWB



pynwb

Build Status

Linux	macOS	Windows
circleci passing	build passing	build passing

Overall Health

codecov 57% requirements up-to-date

To enable a collaborative development process we have adopted modern open-source development and project management processes using GitHub. To ensure software quality and accessibility we have developed: 1) an advanced unit and integration test suite, 2) continuous integration to ensure regular testing on all major systems, 3) code standards and health checks, and 4) software deployment paths (e.g., via PIP and CONDA)

Main Contributors: J.-C. Fillion-Robin, D. Ozturk, C. Kotfila, M. Grauer, W. Schroeder (Kitware); A. Tritt (LBNL)

MatNWB: Making NWB:N 2.0 Accessible via Matlab

MatNWB is a Matlab API for NWB:N 2.0. MatNWB supports generation of Matlab classes for representing NWB:N *neurodata_types* directly from the NWB:N format specification in YAML. Using this approach, MatNWB supports convenient read and write of NWB:N HDF5 files, including data defined via custom format extensions.

Main Contributors: K. Svoboda (Janelia Farms HHMI); N. Clack and L. Niu (Vidriotech); A. Tritt and O. Rübel (LBNL)

Community Review and Contribution

Sponsored by the Kavli Foundation and HHMI Janelia a NWB Hackathon was held on 07/31 – 08/1/2017 at HHMI Janelia Farms for community discussion, review, and development of NWB:N 2.0 and related activities. Continued engagement has also been fostered via the NWB:N Slack channel, GitHub Issues, NWB:N Google group, and countless emails and telecons.

Attendees and contributors: C. Martin, M. Chun, S. Albin (Kavli), K. Bouchard, O. Rübel, A. Tritt, M. Dougherty (LBNL), L. Ng, N. Cain, J. Kiggins (AIBS), T. Davidson (UCSF), M. Grauer, W. Schroeder (Kitware), J. Teeters, S. Mackesey, P. Ježek, C. Holdgraf (UCB), K. Svoboda, D. Gennady, U. Lowell (HHMI Janelia), S. Mckenzie, D. Tingley (NYU) among many others

NWB:N Governance

To guide the next phase of growth, and to coordinate the various efforts, NWB: Neurophysiology has created a governance structure. This will allow NWB:N to grow in a bottom-up, but coordinated manner. Participation in the project is open to any interested neuroscientist, with overall planning coordinated by an Executive Board (EB) with seven members, each appointed for a term of 2-3 years.

Current Members of the Executive Board: K. Bouchard (LBNL), M. Chun (KAVLI), L. Frank (UCSF), C. Koch (AIBS), Markus Meister (Caltech), F. Sommer (UCB), K. Svoboda(HHMI Janelia)

Legacy Data Read

Data in the Allen Cell Types Database (http://celltypes.brain-map.org/) is organized in NWB 1.x format. Using the PyNWB legacy data read module enables users to read and interact with this data using the PyNWB API.

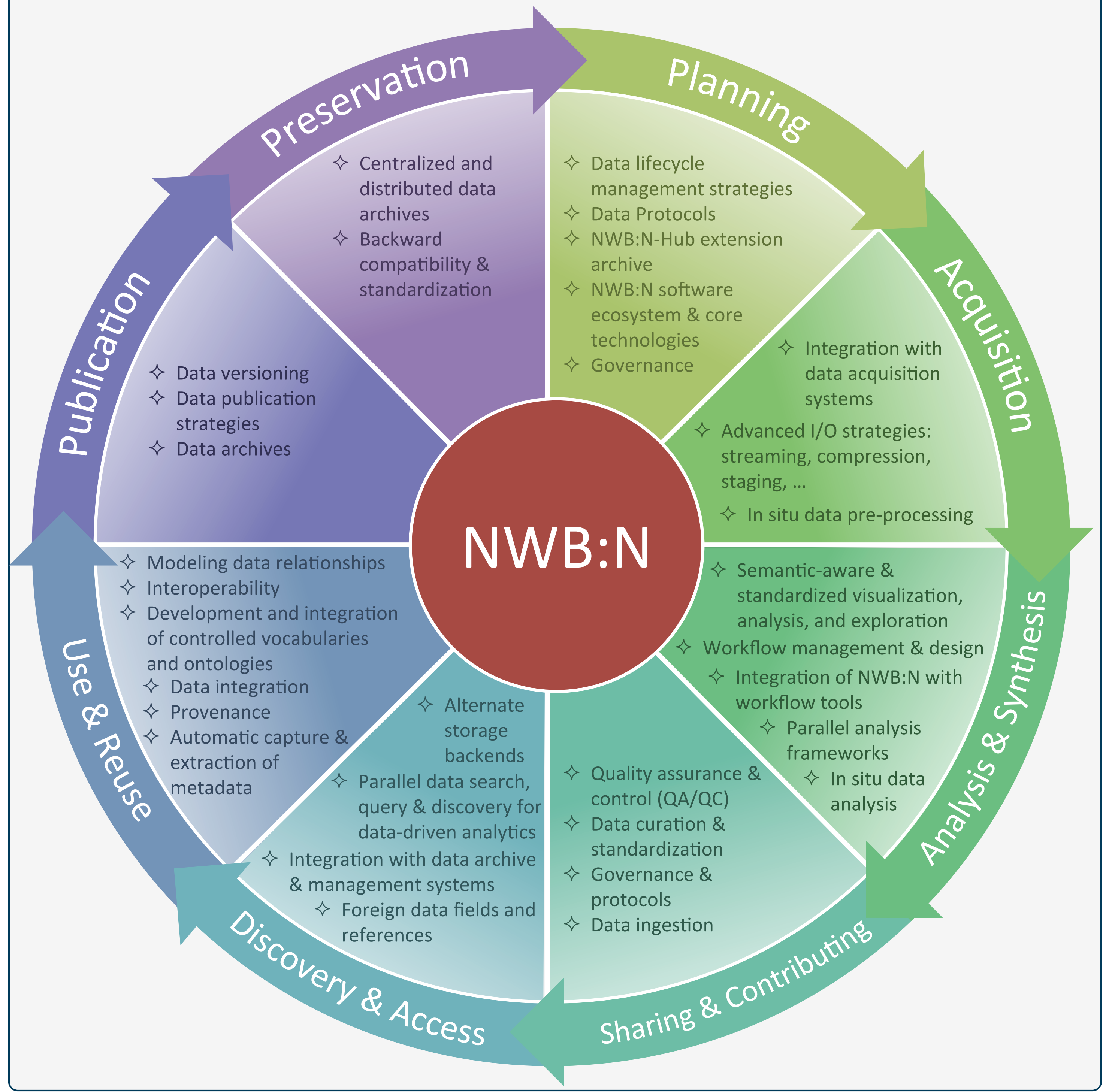
Main Contributors: N. Cain, L. Ng (Allen Institute for Brain Science) and A. Tritt (LBNL)

NWB 2.0 Beta Release

A first public beta release of PyNWB (for Python 2.7.x and >3.5) and NWB 2.0 is available for SfN. The intent of this beta release is to enable early adopters to start exploring the new format and software. While development on NWB 2.0 has been progressing rapidly, further changes to the APIs as well as the format are still planed between this beta and the first full release of NWB 2.0. A full release of NWB 2.0 is planned for Spring/Summer 2018. Detailed documentation of the various aspects of the NWB:N project are available here:

- **Neurodata Without Borders main site:** http://www.nwb.org/
- **General overview of NWB:N** : http://nwb-overview.readthedocs.io
- **NWB:N specification language:** http://schema-language.readthedocs.io
- **NWB:N format specification:** http://nwb-schema.readthedocs.io
- **NWB:N data storage:** http://nwb-storage.readthedocs.io
- **PyNWB:** http://pynwb.readthedocs.io
- **Sources:** All sources are available on GitHub: https://github.com/NeurodataWithoutBorders

Vision and future directions
NWB:N technologies at the heart of the neurodata lifecycle


Preservation
Planning
Acquisition
Analysis & Synthesis
Sharing & Contributing
Discovery & Access
Use & Reuse
Publication
NWB:N
✧ Centralized and distributed data archives
✧ Backward compatibility & standardization
✧ Data lifecycle management strategies
✧ Data Protocols
✧ NWB:N-Hub extension archive
✧ NWB:N software ecosystem & core technologies
✧ Governance
✧ Integration with data acquisition systems
✧ Advanced I/O strategies: streaming, compression, staging, …
✧ In situ data pre-processing
✧ Semantic-aware & standardized visualization, analysis, and exploration
✧ Workflow management & design
✧ Integration of NWB:N with workflow tools
✧ Parallel analysis frameworks
✧ In situ data analysis
✧ Quality assurance & control (QA/QC)
✧ Data curation & standardization
✧ Governance & protocols
✧ Data ingestion
✧ Alternate storage backends
✧ Parallel data search, query & discovery for data-driven analytics
✧ Integration with data archive & management systems
✧ Foreign data fields and references
✧ Modeling data relationships
✧ Interoperability
✧ Development and integration of controlled vocabularies and ontologies
✧ Data integration
✧ Provenance
✧ Automatic capture & extraction of metadata
✧ Data versioning
✧ Data publication strategies
✧ Data archives