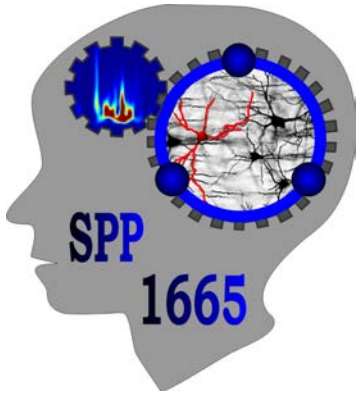


JÜLICH
FORSCHUNGSZENTRUM

UNIC

G-Node
www.g-node.org

BrainScaleS

Human Brain Project

Helmholtz Portfolio Theme
SMHB
Supercomputing and Modeling for the Human Brain

RWTH AACHEN UNIVERSITY

Neuroinformatics infrastructures for the analysis of ephys data

Institute of Neuroscience and Medicine (INM-6)
Jülich Research Center, Germany

SPP1665 Data Analysis Course

Nov 24, 2014 Michael Denker

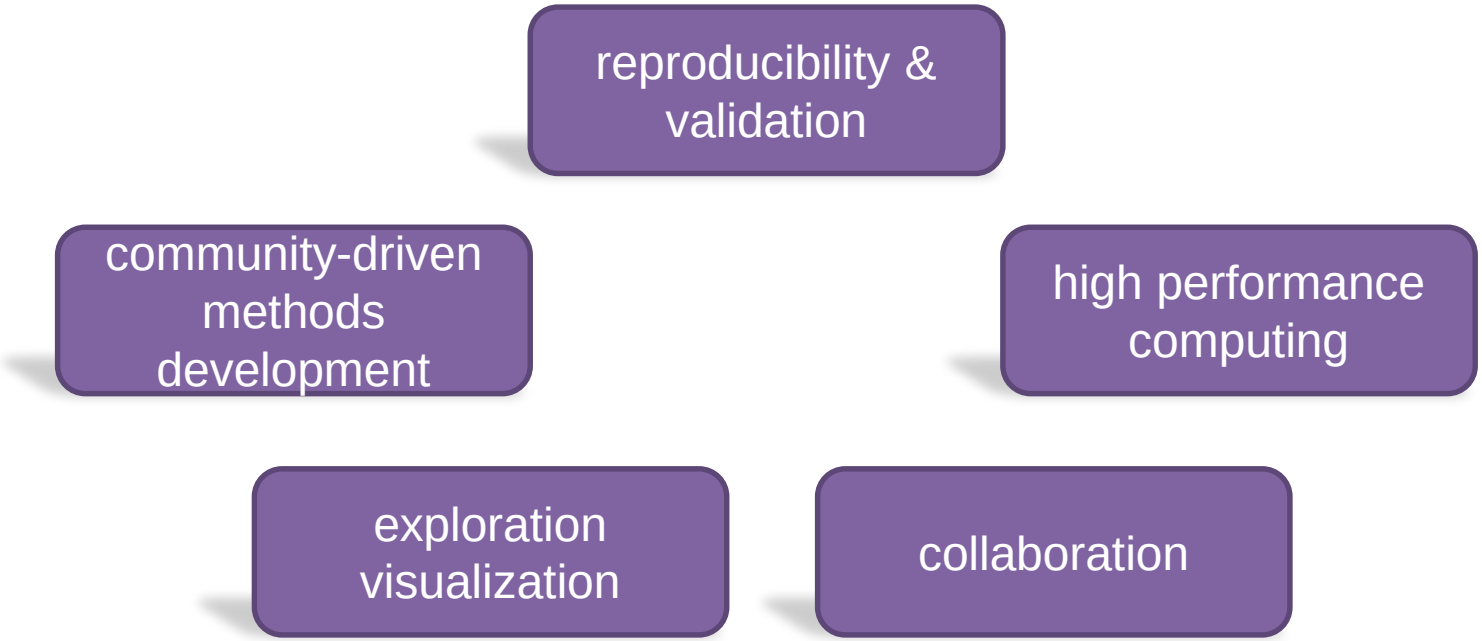
JÜLICH
FORSCHUNGSZENTRUM

Using Neo to work with e-phys data

Nov 24, 2014 Michael Denker

Improving workflows for electrophysiology

Goal: Modular ***software habitat and related infrastructures*** to support ***integrated data analysis workflows***

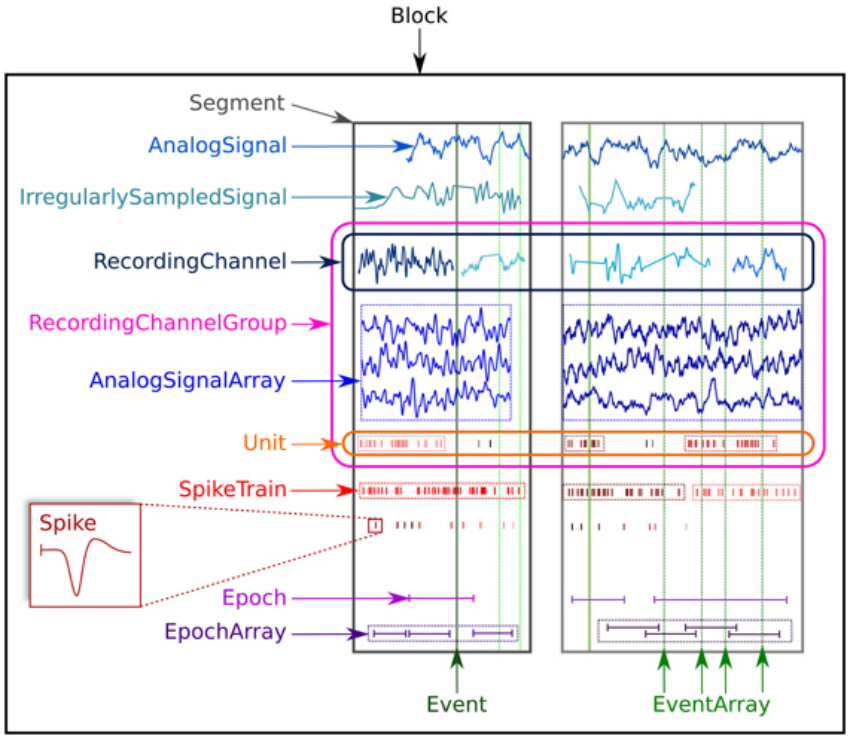


SMHB WP4 Task 4.5: Software tool for correlation analyses of multi-channel time series at multiple scales (Grün)

HBP WP 5.3: Tools for the analysis of functional data (Grün, Denker, Davison)

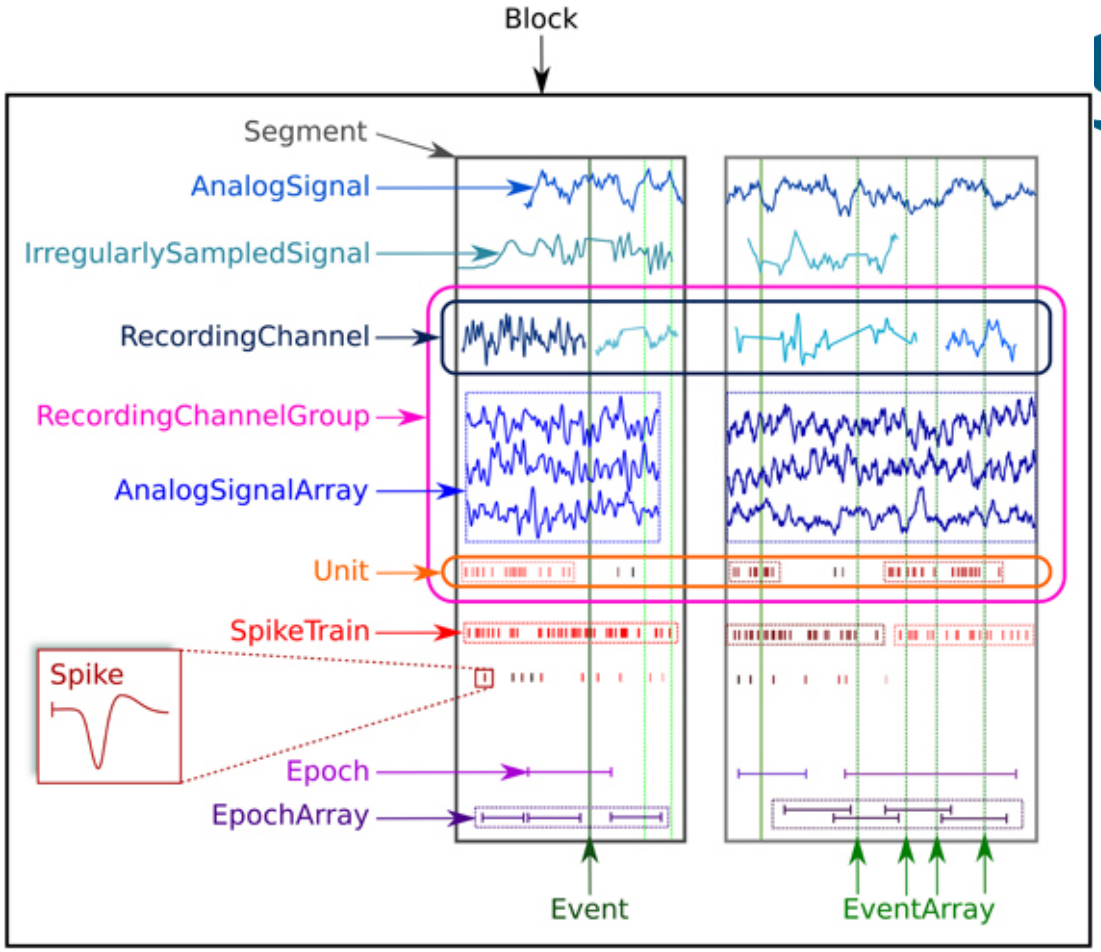
Neo: data representations for electrophysiology

Source: wikipedia



Garcia et al. (2014) Front Neuroinform.

- *Neo*: data representations to support ***common interface*** to consumers of electrophysiological data
- Data representations in Python for the types of data commonly found in an experiment freed from specific semantics (ex: *RecordingChannelGroup*)

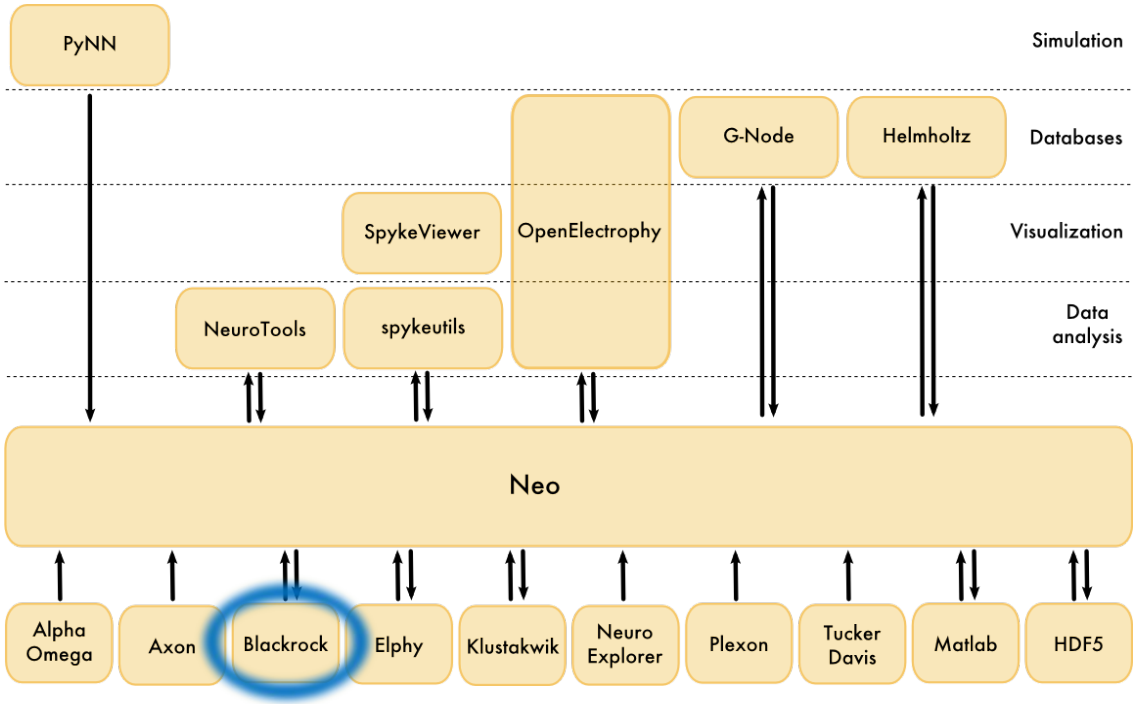


- Relationships between data items: i.e., Block → Segment → AnalogSignal
- ***Hierarchical*** bidirectional relationships (parent ↔ child)
- Can contain references to data objects across container hierarchy
- Metadata as annotations for every Neo core object
- Designed from perspective of ***data providers*** and ***data consumers***

Goals of Neo

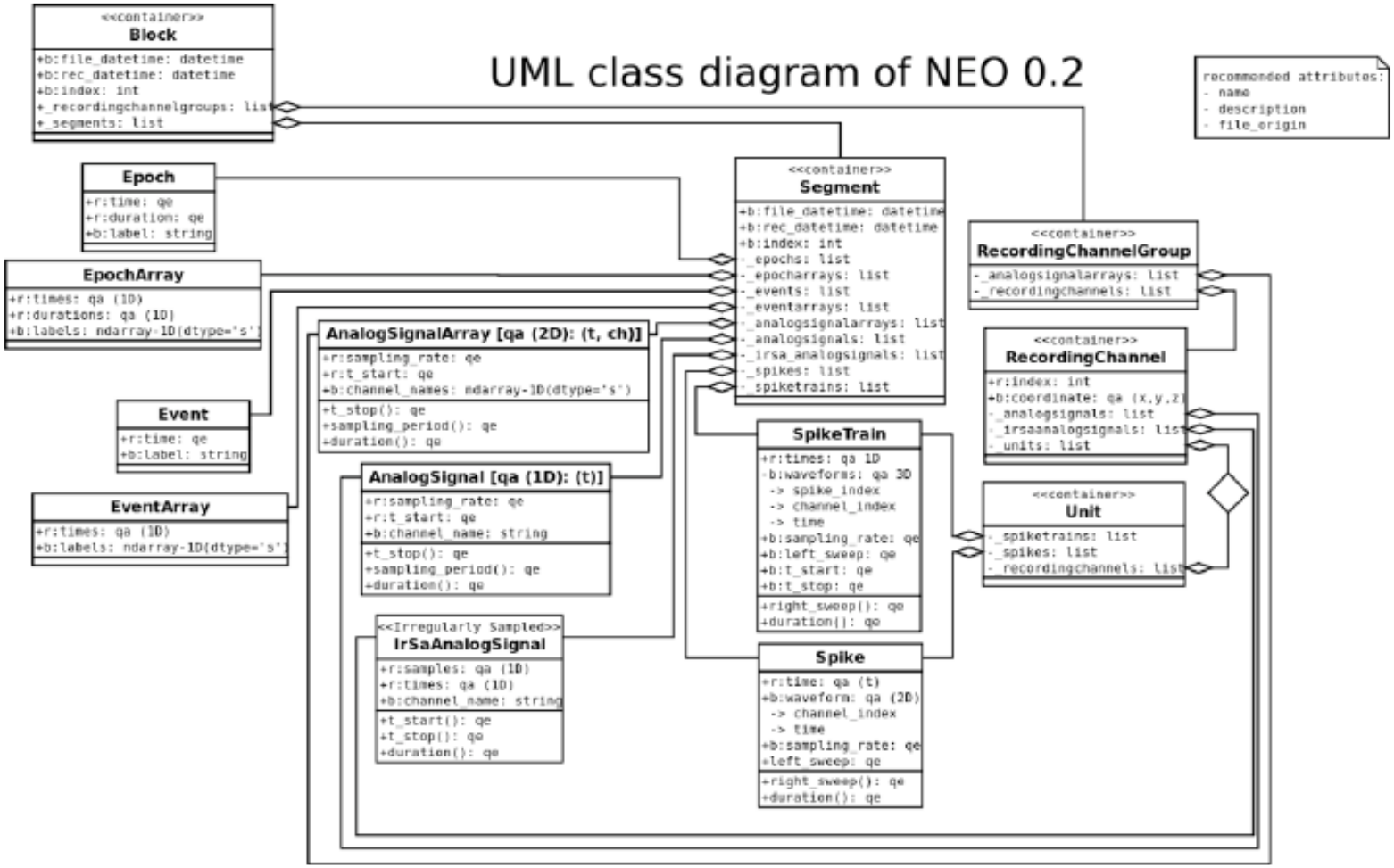
- provide ***a common set of base classes for neurophysiology data, to improve interoperability*** between Python tools for data analysis/visualization/storage/simulation (OpenElectrophy, NeuroTools, elephant, G-Node, Helmholtz, DABBSIP, PyNN...)
- provide tools ***to facilitate access*** to these base classes from a programmer's perspective (e.g., filtering the object tree, time slicing objects,…)
- prioritize ***simplicity*** (e.g. no data analysis methods included), ***performance*** (based on numpy) and ***correctness*** (explicit units/dimensionality checking)
- be able to read from/write to ***many data formats***

Connecting software resources via Neo



- *Neo*: evolving community standard for common to support the interoperability of software tools
- Corner stone: tested and efficient ***file I/O modules***
- Current INM-6 activity: development of a flexible file I/O for one (soon several) of the most desired file formats

Neo Class diagram



First example usage: Working with Neo objects

```
>>> from neo import AnalogSignal, Segment
>>> import numpy
>>> from quantities import uV, Hz

>>> data = numpy.random.uniform(size=100)*uV
>>> signal = AnalogSignal(data, sampling_rate=420*Hz)
>>> isinstance(signal, numpy.ndarray)
True

# Time keeping for analog signals:
>>> signal.t_start
array(0.0) * s
>>> signal.t_stop
array(0.1) * s
>>> signal[20:80].t_stop
array(0.08) * s

# Adding annotations:
>>> signal.annotate(cell_id="20100405a")
```

First example usage: Working with the object

```
# Filtering
>>> signal2=AnalogSignal(data+1*uV, sampling_rate=420*Hz)
>>> signal2.annotate(cell_id="20100405b")
>>> seg=Segment()
>>> seg.analogsignals.append(signal)
>>> seg.analogsignals.append(signal2)

>>> seg.filter(cell_id="20100405a")
[AnalogSignal in 1.0 uV with 100 float64 values
annotations: {'cell_id': '20100405a'}
channel index: None
sampling rate: 420.0 Hz
time: 0.0 s to 0.238095238095 s]

# Working with the object:
>>> print
    numpy.mean(seg.filter(cell_id="20100405a")[0].magnitude)
0.545829319931
```

JÜLICH
FORSCHUNGSZENTRUM

odML: practical experience with metadata management

Nov 24, 2014 Michael Denker

Complex meta data in a behavioral experiment
What is meta data?

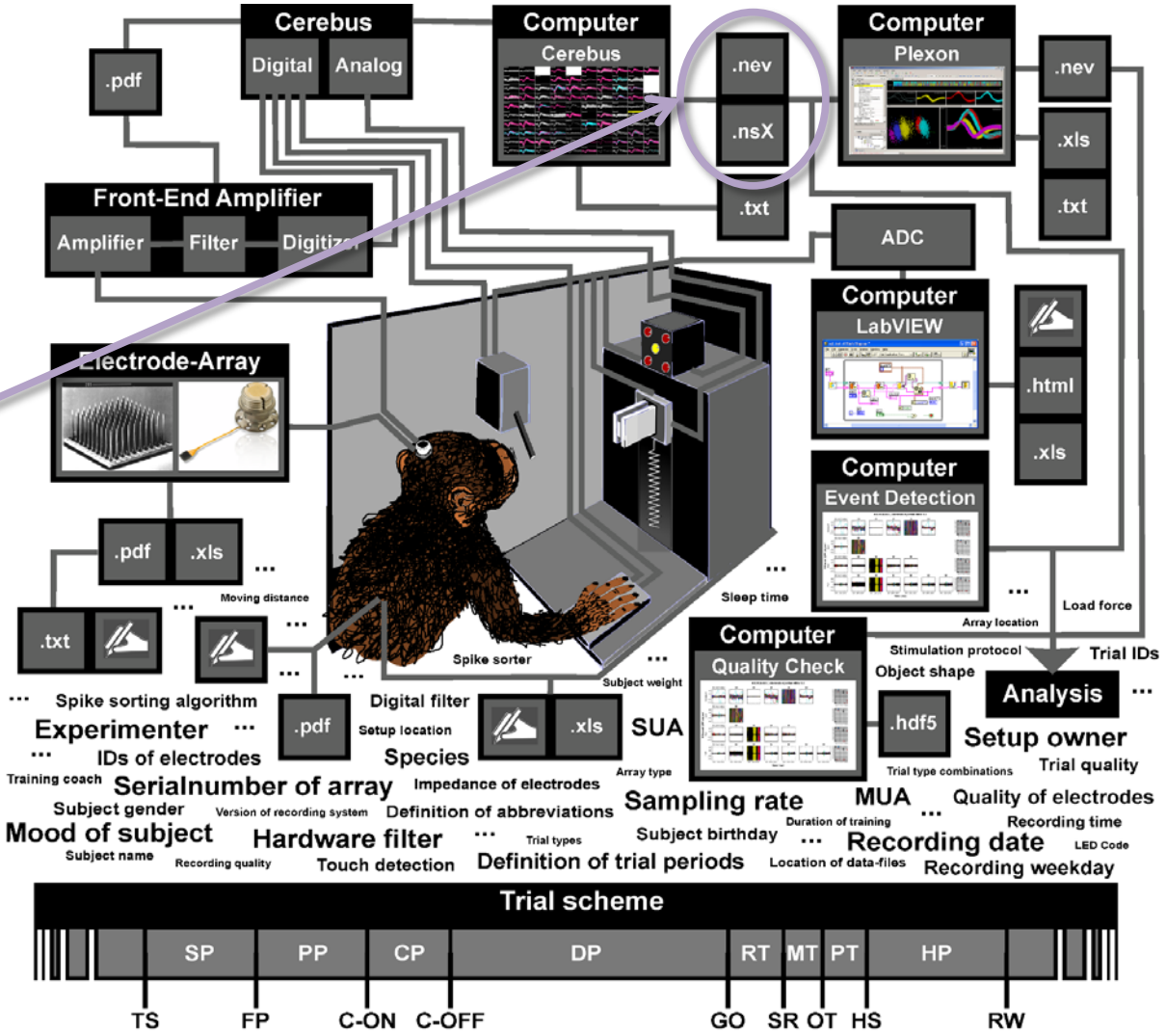
Reach to grasp exp.:

- 120 trials / subsession
- ~ 5 subsessions / day
- ~ 70 days / monkey
- 3 monkeys

Neural data in:

.2 files

.2 formats



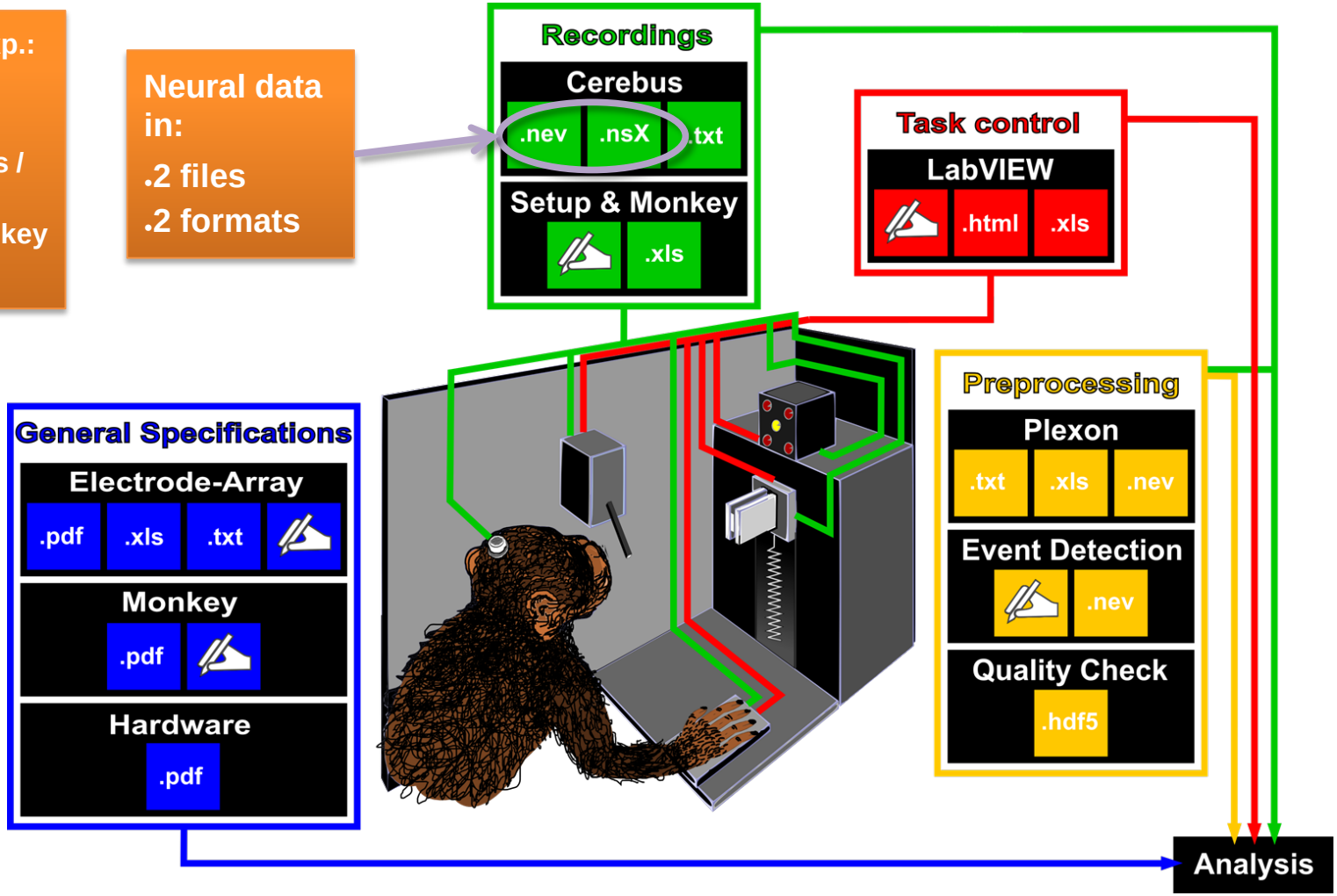
Zehl, Denker, Stoewer, Jaillet, Brochier, Riehle, Wachtler, Grün (in prep.)

Complex meta data in a behavioral experiment
Ordering metadata

JÜLICH FORSCHUNGSZENTRUM

Reach to grasp exp.:
- 120 trials / subsession
- ~ 5 subsessions / day
- ~ 70 days / monkey
- 3 monkeys

Neural data in:
.2 files
.2 formats



Zehl, Denker, Stoewer, Jaillet, Brochier, Riehle, Wachtler, Grün (in prep.)

odML: meta data for electrophysiological experiments

JÜLICH FORSCHUNGSZENTRUM

odML open metadata Markup Language

Example: Stimulus
Stimulus duration = 500.0 ms ± 0.0 ms

odML value types:

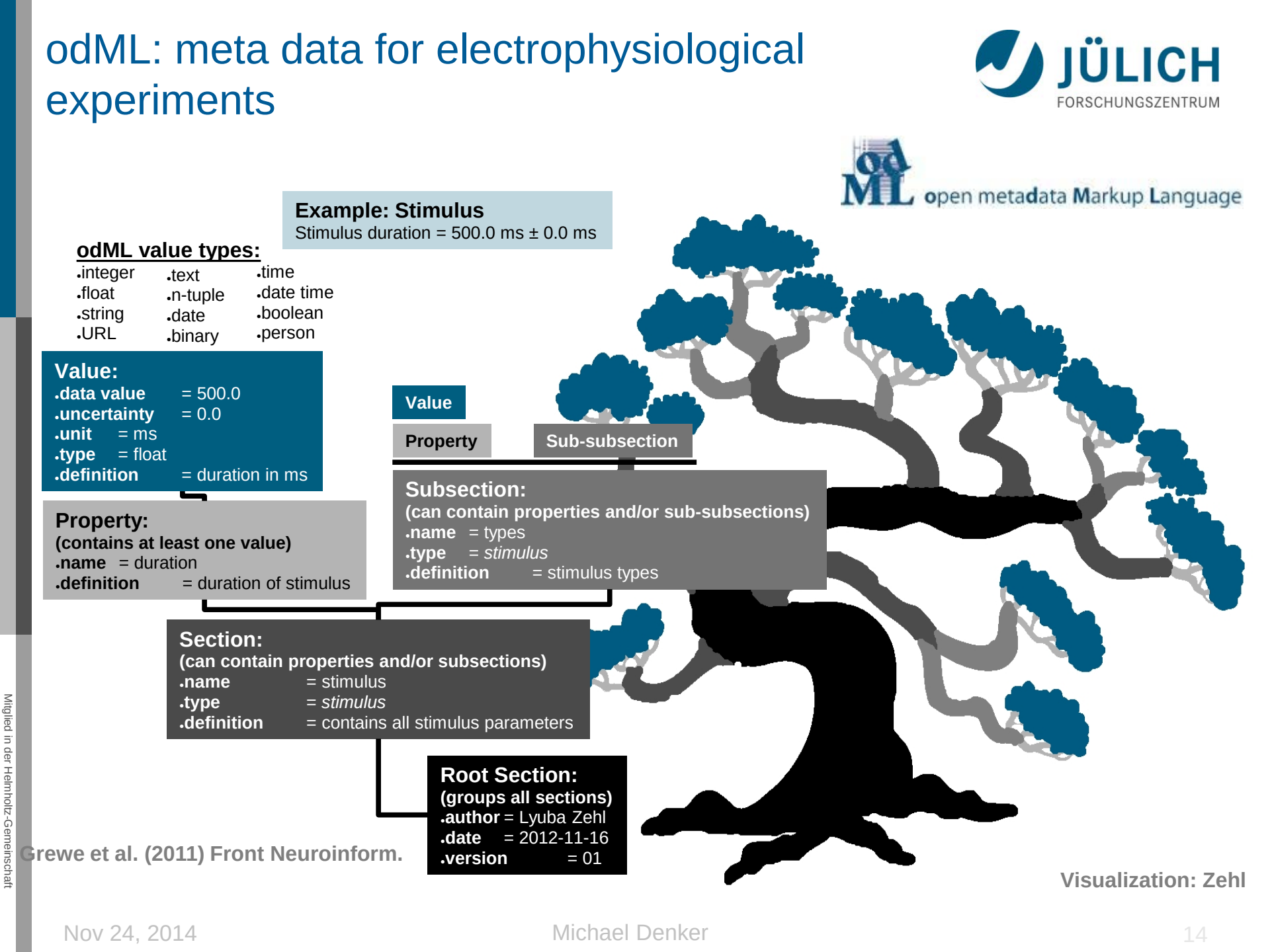
.integer	.text	.time
.float	.n-tuple	.date time
.string	.date	.boolean
.URL	.binary	.person

Value:
.data value = 500.0
.uncertainty = 0.0
.unit = ms
.type = float
.definition = duration in ms

Property:
(contains at least one value)
.name = duration
.definition = duration of stimulus

Section:
(can contain properties and/or subsections)
.name = stimulus
.type = *stimulus*
.definition = contains all stimulus parameters

Root Section:
(groups all sections)
.author = Lyuba Zehl
.date = 2012-11-16
.version = 01



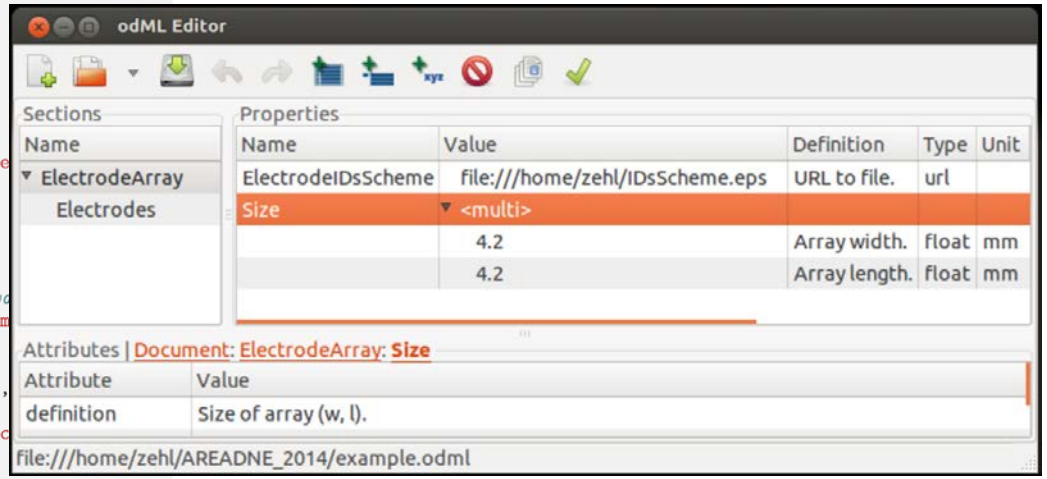
Grewe et al. (2011) Front Neuroinform.

Visualization: Zehl

odML: compiled via scripts and the editor

Code 1: How to generate an odML file in Python

```python
import odml
import datetime

# Generate an odml-document
odml_document = odml.Document(author = "Lyuba Zehl",
                              date = datetime.date(2014, 6,

# Generate an odml-section
odml_section = odml.Section(name = "ElectrodeArray",
                            definition = "Information on ele

# Generate an odml-subsection
odml_subsection = odml.Section(name = "Electrodes",
                               definition = "Information on

# Generate odml-properties with their corresponding odml-va
odml_value_a = odml.Value(data = "file:///home/zehl/IDsSchem
                          dtype = odml.DType.url,
                          definition = "URL to file.")
odml_property_a = odml.Property(name = "ElectrodeIDsScheme",
                                value = odml_value_a,
                                definition = "Scheme of elec

odml_value_b1 = odml.Value(data = 4.2,
                           dtype = odml.DType.float,
                           unit = "mm",
                           definition = "Array width.")
odml_value_b2 = odml.Value(data = 4.2,
                           dtype = odml.DType.float,
                           unit = "mm",
                           definition = "Array length.")
odml_property_b = odml.Property(name = "Size",
                                value = [odml_value_b1, odml_value_b2],
                                definition = "Size of array (w, l).")

# Append all odml-objects to odml-tree
odml_section.append(odml_subsection)
odml_section.append(odml_property_a)
odml_section.append(odml_property_b)
odml_document.append(odml_section)

# Save the odml-document with all its content
write_to = "/home/zehl/AREADNE_2014/example.odml"
odml.tools.xmlparser.XMLWriter(odml_document).write_file(write_to)
```



Improve python libraries for generation and access of meta data information

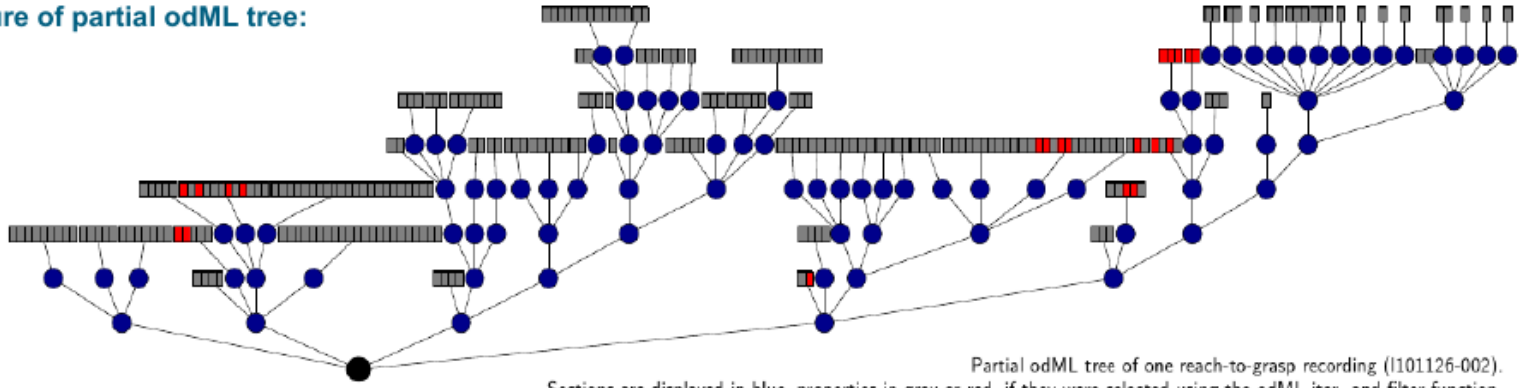
Tutorial on building successful meta data collections using odML

Accessible templates for common odML sections (e.g., common hardware) to be provided

Visualization: Zehl

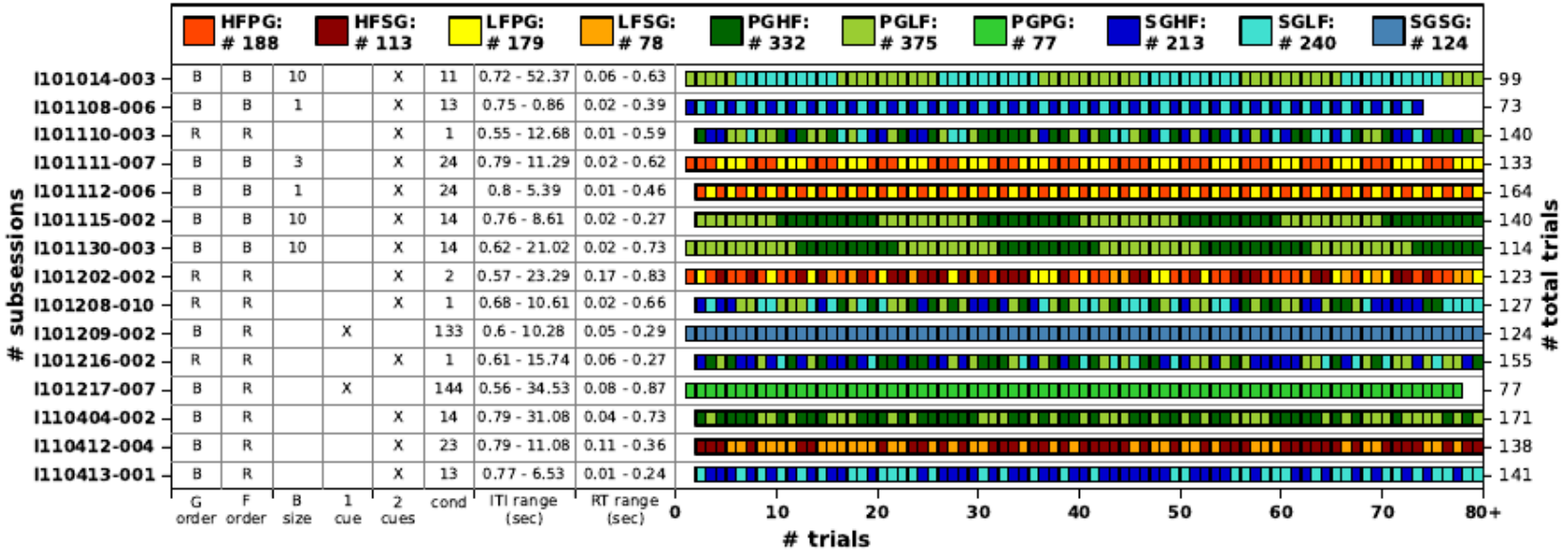
Why meta data becomes important

Structure of partial odML tree:



Partial odML tree of one reach-to-grasp recording (l101126-002).
Sections are displayed in blue, properties in grey or red, if they were selected using the odML iter- and filter-function.

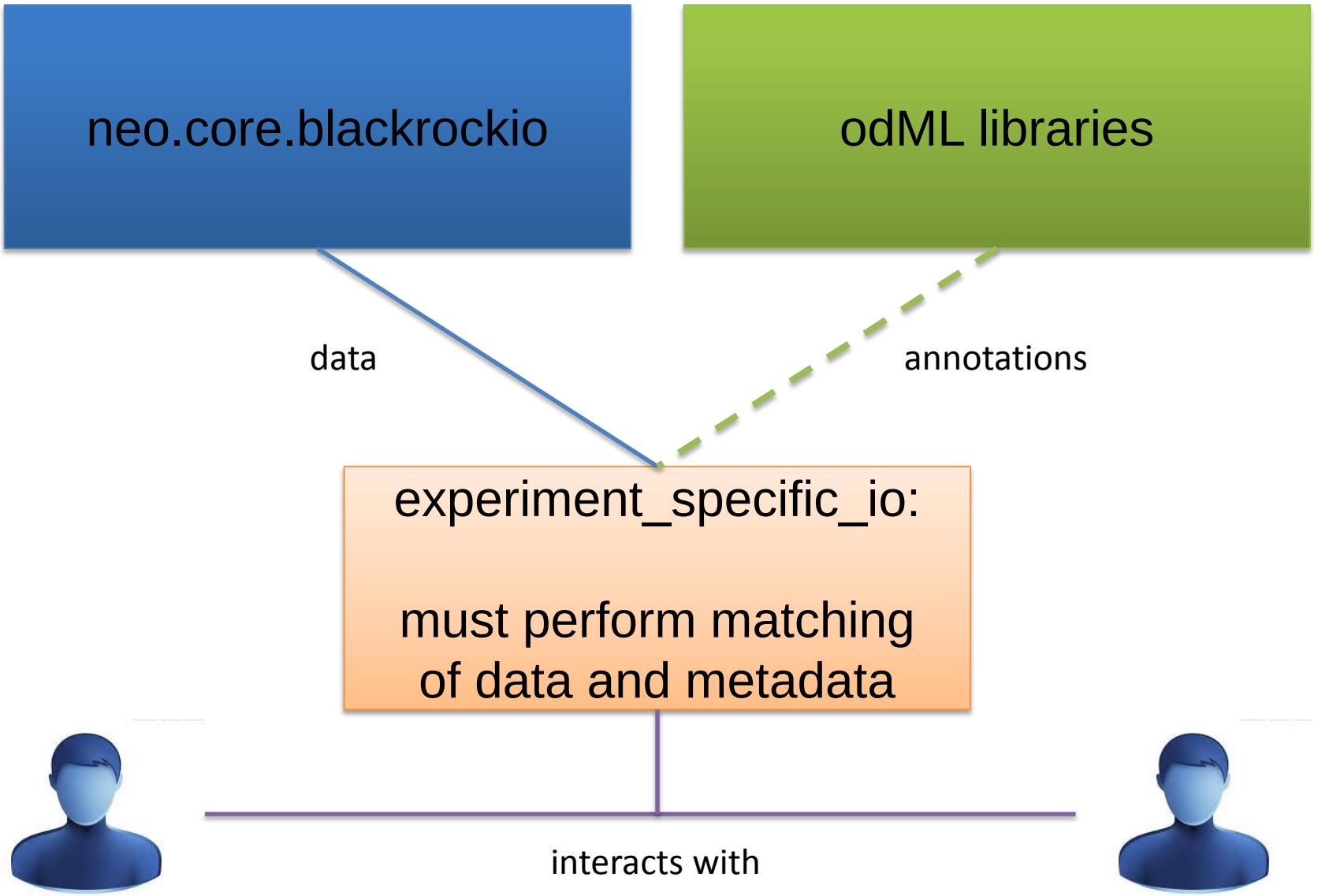
monkey: Lilou || # sessions: 15 || # subsessions: 15 || # standard: 15 || # trials: 1919 || # trials/subsession: 127



	G order	F order	B size	1 cue	2 cues	cond	ITI range (sec)	RT range (sec)	# total trials
l101014-003	B	B	10		X	11	0.72 - 52.37	0.06 - 0.63	99
l101108-006	B	B	1		X	13	0.75 - 0.86	0.02 - 0.39	73
l101110-003	R	R			X	1	0.55 - 12.68	0.01 - 0.59	140
l101111-007	B	B	3		X	24	0.79 - 11.29	0.02 - 0.62	133
l101112-006	B	B	1		X	24	0.8 - 5.39	0.01 - 0.46	164
l101115-002	B	B	10		X	14	0.76 - 8.61	0.02 - 0.27	140
l101130-003	B	B	10		X	14	0.62 - 21.02	0.02 - 0.73	114
l101202-002	R	R			X	2	0.57 - 23.29	0.17 - 0.83	123
l101208-010	R	R			X	1	0.68 - 10.61	0.02 - 0.66	127
l101209-002	B	R		X		133	0.6 - 10.28	0.05 - 0.29	124
l101216-002	R	R			X	1	0.61 - 15.74	0.06 - 0.27	155
l101217-007	B	R		X		144	0.56 - 34.53	0.08 - 0.87	77
l110404-002	B	R			X	14	0.79 - 31.08	0.04 - 0.73	171
l110412-004	B	R			X	23	0.79 - 11.08	0.11 - 0.36	138
l110413-001	B	R			X	13	0.77 - 6.53	0.01 - 0.24	141

0 10 20 30 40 50 60 70 80+

Challenge:
Combining data and metadata



Example usage: Loading

To load a file format which is implemented in a MyFormatIO class

```
>>> from neo.io import MyFormatIO
>>> file = MyFormatIO("myfile.dat")
```

To know what types of objects are supported by this io interface:

```
>>> file.supported_objects
[Segment , AnalogSignal , SpikeTrain, Event, Spike ]
```

Not all supported objects can be read directly. For instance, many formats supports AnalogSignal but you can't access them directly: you must read a Segment and access your AnalogSignal like that:

```
>>> seg = file.read_segment()
>>> seg.get_analogsignals()
```

To have the list of directly readeable objects :

```
>>> file.readable_objects
[Segment]
```

To read the entire file :

```
>>> result = file.read()
>>> type(result)
neo.core.Segment
```


Source: A. Davison

Example: BlackrockIO header info

```
class BlackrockIO(BaseIO):
    is_readable = True
    is_writable = False
    supported_objects = [neo.Block, neo.Segment,
        neo.AnalogSignal, neo.SpikeTrain, neo.EventArray,
        neo.RecordingChannelGroup, neo.RecordingChannel]
    readable_objects = [neo.Block]
    writeable_objects = []
    has_header = False
    is_streameable = False
    read_params = {}
    write_params = {}
    name = 'Blackrock'
    description = 'This IO reads .nev/.nsX file of the
        Blackrock (Cerebus) recordings system.'
    extensions = ['ns' + str(_) for _ in range(1, 7)]
    extensions.append('nev')
    mode = 'file'
```

Example: BlackrockIO readers

```
def read_segment(self, n_start=None, n_stop=None,
                 load_waveforms = False, nsx_num = None,
                 lazy=False, cascade=True):

        """Reads one Segment.

        The Segment will contain one AnalogSignal for each
        channel and will go from n_start to n_stop (in
        samples).

        Arguments:
            n_start : time in samples that the Segment begins
            n_stop : time in samples that the Segment ends

        Python indexing is used, so n_stop is not inclusive.

        Returns a Segment object containing the data.
        """
```

Future efforts

Driving forces:

- INCF Electrophysiology Data Sharing Task Force
- Neurodata without Borders (NWB) initiative
- Klustakwik File Format
- Allen Brain Institute Format
- ….

Goal: Automatic combination of data and metadata

→ Standard file format for ephys data combining

- Raw data
- Processed data
- Metadata
- Relationships between objects
- Provenance trace for objects

Possible framework currently discussed: NIX (HDF5 based)
+ Modular Semantic Structure

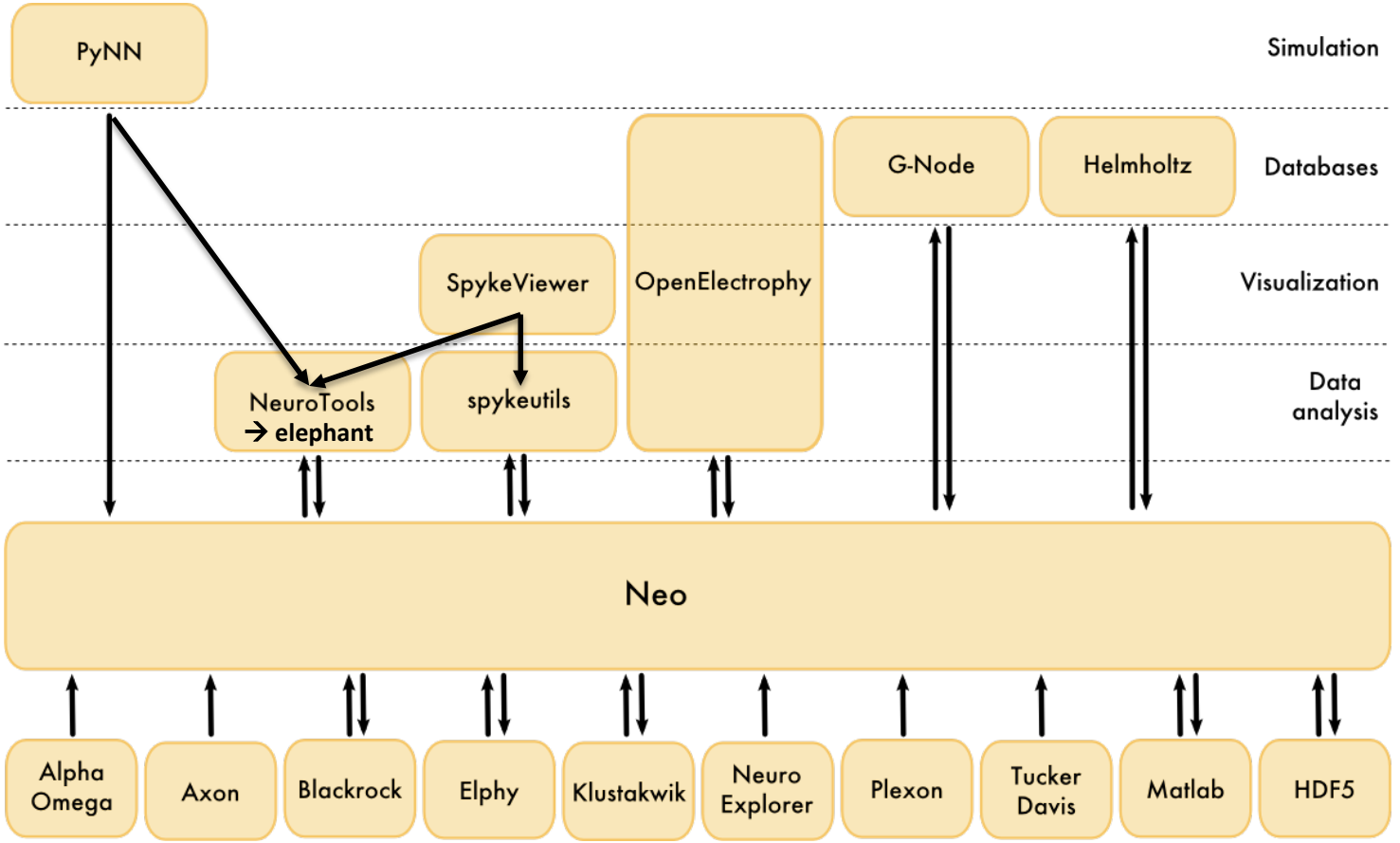
JÜLICH
FORSCHUNGSZENTRUM

High-level tools for data analysis in Python and beyond

Nov 24, 2014

Michael Denker

Connecting software resources via Neo



Selected Python based Analysis frameworks

NeuroTools
http://neuralensemble.org/trac/NeuroTools
- One of the first attempts: "*try: reduce(duplication,community)*"
- Developed in the Facets/BrainScales EU initiatives
- Relatively open management; collection of various tools
- Introduces Spike and LFP class concepts

nitime (nipy)
http://http://nipy.org/nitime/
- Time-frequency tools (originally for neuroimaging)

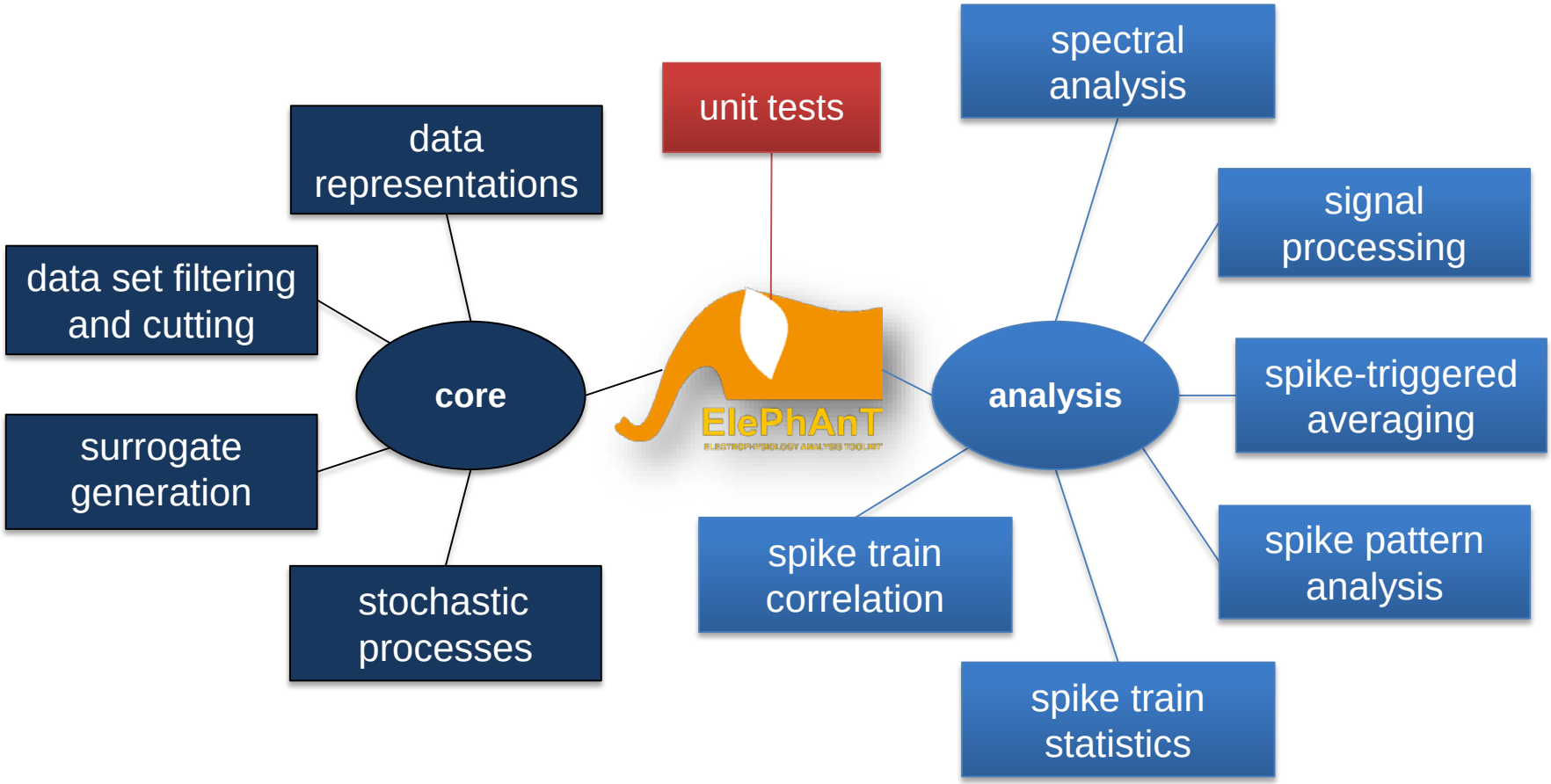
MNE
https://github.com/mne-tools/mne-python
- Magnetoencephalography (MEG) and Electroencephalography (EEG) in Python
- Neo interface work in progress

openElectrophy
http://neuralensemble.org/trac/OpenElectrophy
- GUI-based analysis framework with focus on analysis steps that require interactive visualization
- Further develops Spike and LFP class concepts as Neo prototypes during initial analysis

spykeutils / spykeviewer
http://spykeutils.readthedocs.org/en/0.4.1/
- Integrated Neo viewer and analysis tool (viewer) and underlying tools
- Sophisticated plugin architecture – allows to easily integrate your own analysis or other toolboxes

elephant
https://github.com/NeuralEnsemble/elephant
- Very young initiative emerging in part from *Neurotools* to provide a base analysis layer for common spike train and LFP analysis methods built on Neo
- Combines expertise and efforts from multiple developers of other tools
- Designed to be pluggable into higher levels, e.g., spykeviewer (GUI software), Unified Portal (online portals), NEST/PYNN (simulation)

elephant: Electrophysiology Analysis Toolkit

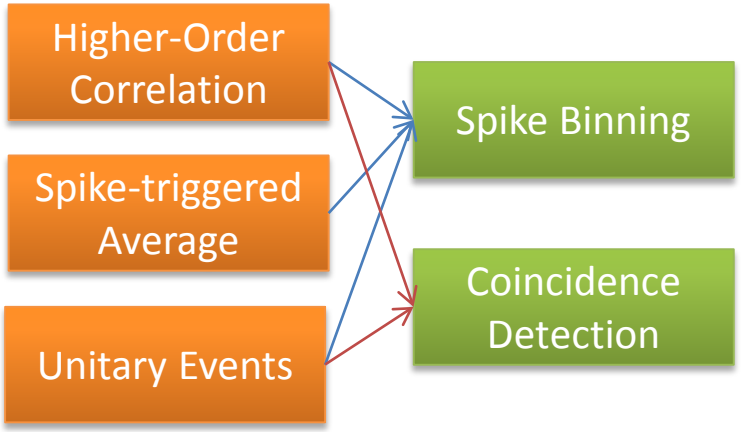


- Analyze electrophysiological multi-scale data obtained from experiments and simulations
- Current status: github repository in progress; connect with existing efforts (e.g., *OpenElectrophy*, *NeuroTools*, *spykeutils*)

Contacts: **Denker, Yegenoglu, Davison, Pröpper, Garcia et al**

Challenges and Design Goals

- Rich repertoire of analysis functions built on the neo framework
- Allow comparison of different analyses methods by a highly modular design
- Flexible data management to integrate into a number of different workflow use cases



INCF Workshop “New perspectives on Workflows and Data Management for the Analysis of Electrophysiological Data

JÜLICH FORSCHUNGSZENTRUM

Documentation and provenance tracking solutions to support

- better, reproducible analysis processes
- the ability cope with the required levels of flexibility and data size

Presented technologies:

- **Provenance W3C standard:** enable publication and exchange of provenance information represented using widely available file formats [3]
- **Sumatra:** command-line tool to enable automatic capture and convenient browsing of provenance data from simulation and analysis programs [4]
- **LabLog:** tool provides a project-centric environment to capture data and metadata [5]
- **Research Objects:** framework and standards structure and exchange analysis results for improved reproducibility [6]
- **ISA metadata tracking tools:** facilitate the management of complex experimental data and analysis by providing rich descriptions of experiment and analysis [7]

http://incf.org/activities/workshops/scientific-workshops

INCF Workshop “New perspectives on Workflows and Data Management for the Analysis of Electrophysiological Data

Data structures and software libraries that

- enable interfacing of data from various sources
- integrate methods for data manipulation and analysis

Presented technologies:

- **odML (Open metaData Markup Language):** XML-based management and storage of metadata and ontology information [12]
- **Neo:** implements a generic data model and file I/Os for electrophysiological data [13]
- **Spyke Viewer:** Neo-based GUI application for data visualization and analysis [14]
- **CRCNS.org:** Online platform / marketplace for data and analysis sharing [15]
- **SEEK4Science:** Web platform for sharing heterogeneous data sets, models and research results [16]

http://incf.org/activities/workshops/scientific-workshops

INCF Workshop “New perspectives on Workflows and Data Management for the Analysis of Electrophysiological Data

JÜLICH FORSCHUNGSZENTRUM

Workflow management systems that allow

- the automated and flexible processing of complex analysis workflows
- easy access to parallelization technologies of high-performance computing (HPC)

Presented technologies:

- **CARMEN:** virtual cloud-based laboratory for electrophysiological data that combines data and analysis sharing with online data manipulation [8]
- **nipype:** non-graphical workflow pipeline solution in Python developed in the context of imaging software [9]
- **Taverna:** fully featured graphical, extensible workflow system with connectors for popular HPC interfaces and online workflow monitoring [10]
- **Human Brain Project Platform Solution:** all-round solution to workflow management using integrated data hosting, provenance tracking, and access to HPC [11]

http://incf.org/activities/workshops/scientific-workshops

INCF Workshop “New perspectives on Workflows and Data Management for the Analysis of Electrophysiological Data

Tools for supercomputing and parallelization that

- provide scientists with easy access to parallel computing [17]
- integrate smoothly into the analysis workflow

Presented technologies:

- **PyCOMPS:** system to allow enable transparent, hardware agnostic trivial parallelization of analysis code
- **UNICORE:** general purpose grid management software that provides users with an intuitive GUI and command line client, application integration mechanisms, and basic workflow management facilities [18]

http://incf.org/activities/workshops/scientific-workshops

Towards large integrative loops in Neuroscience

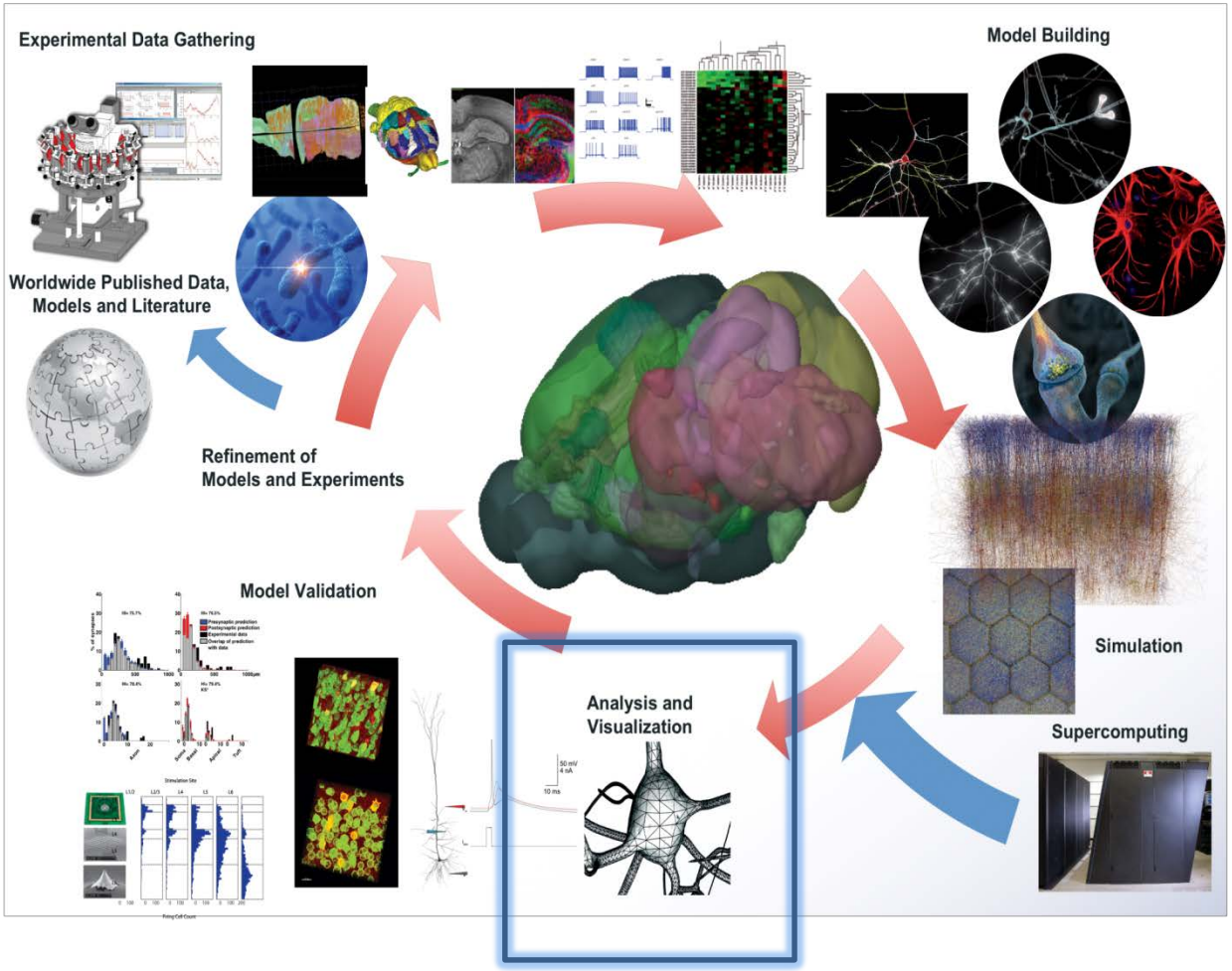


Image: HBP