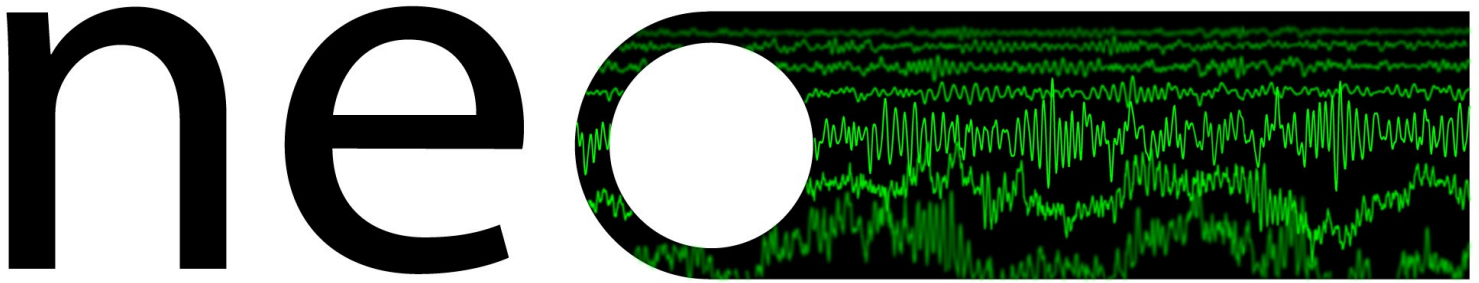
CodeJam #6 – Jülich January 2014



Samuel Garcia

neo : a 100% code jam project!

History :

- Discussions started in Freiburg 2009 (codejam #3).
- Neo 0.1 release some mouth. Coded almost alone.
- Neo 0.1 presented in Marseille 2010 (codejam #4) + discussion for neo 0.2
- Code sprint in Gif at Andrew's in 2011.
- Released on Feb 2012. neo 0.2
- Presentation Edinburgh 2012 (codemjam #5)
- Release neo 0.3 on June 2013
- Presentation Jülich 2014 (codemjam #6) + discussion for neo 0.4

Sorry for people that have seen this presentation several so much times.

What is neo ?

neo.core = a simple and intuitive set on objects for representing electrophysiological dataset in memory for python language.

neo.io = a common layer for reading/writing in the cacophony of file formats.

Goals ?

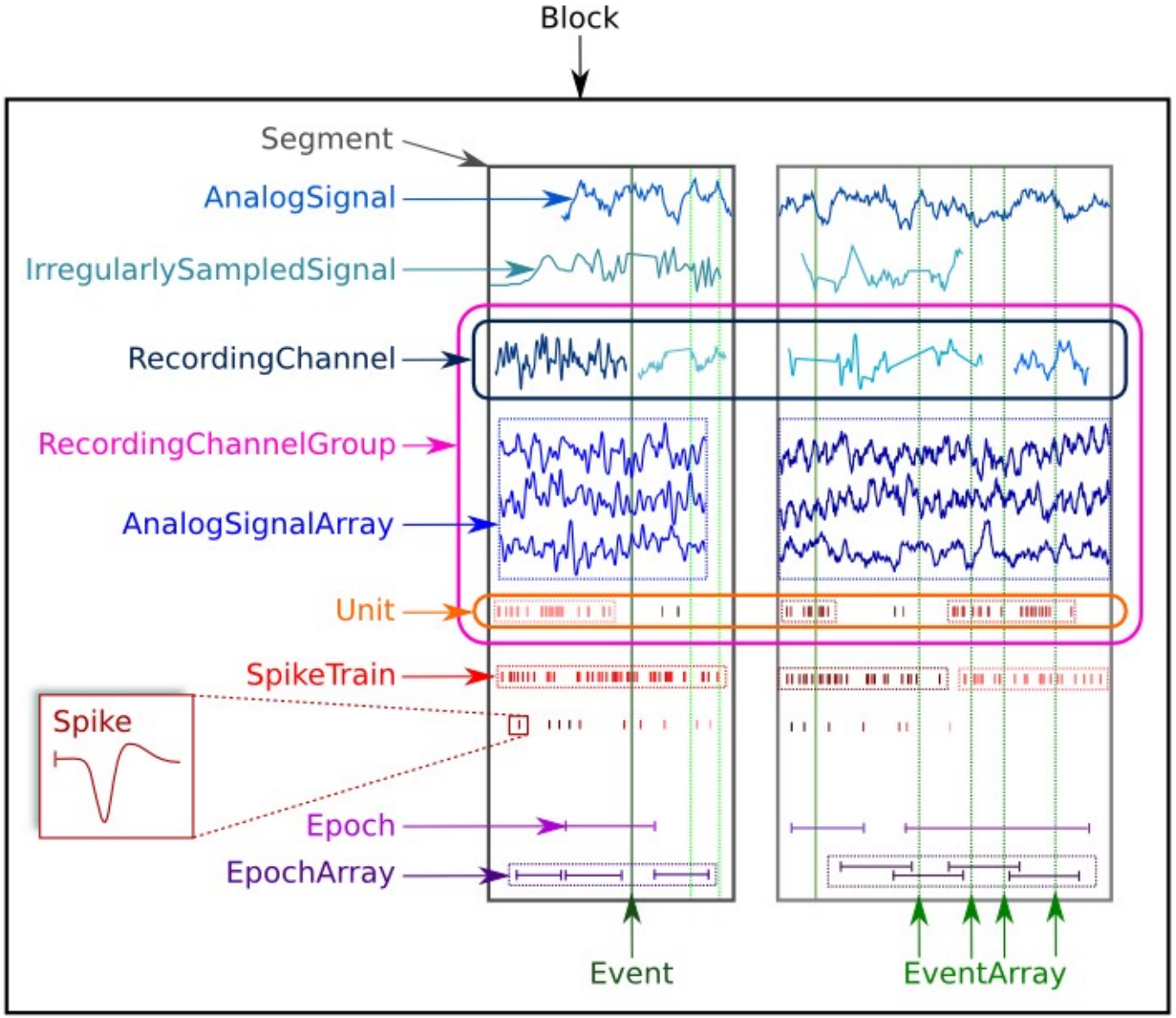
What are main interests :
• Interoperability between projects (g-node, pynn, OpenElectrophy, SpykeViewer, Mozaik)
• file reader/writter : A 5 min. installable, multiplatfrom, and easy to play.
• No plotting, no analyzing

Dependencies ?

Few = numpy and quantities
Optional for some IOs = pytables , scipy, ...

Equivalent project
• Neuroshare (ddl provide commercial)
• for neuro imaging: nibabel (python)
• Biosig C/C++

Class tour


Block
Segment
AnalogSignal
IrregularlySampledSignal
RecordingChannel
RecordingChannelGroup
AnalogSignalArray
Unit
SpikeTrain
Spike
Epoch
EpochArray
Event
EventArray

Class tour: Concept

3 types of objects:
• Data objects : AnalogSignal, SpikeTrain, EventArray, EpochArray
• Containers objects : Block, Segment
• Grouping objects : RecordingChannel, RecordingChannelGroup, Unit (ex Neuron)

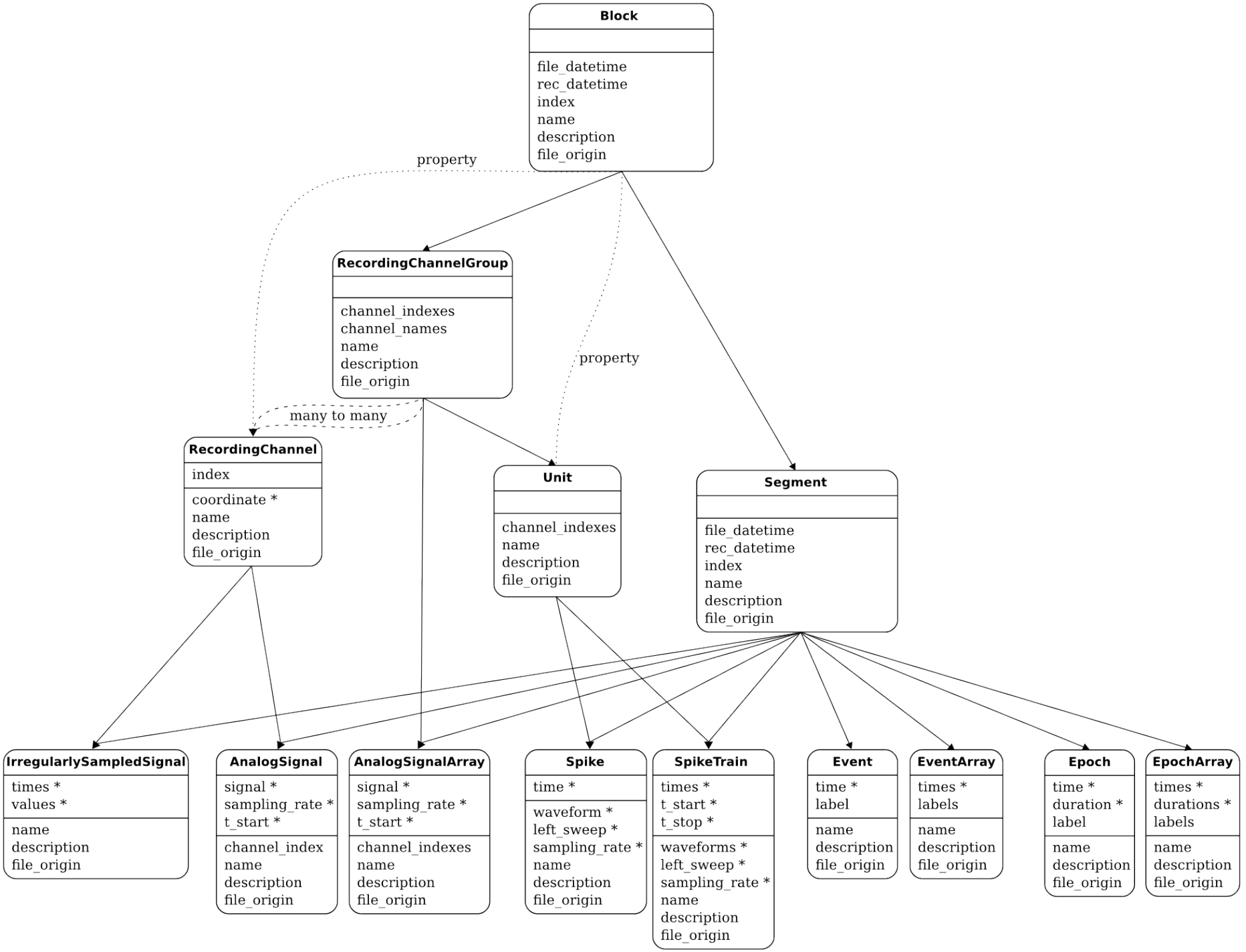
All object have 3 types of attributes:
• Required (AnalogSignal.sampling_rate, AnalogSignal.t_start, ...)
• Recommended (AnalogSignal.name, ...)
• Free in annotations dict:

```
>>> seg = Segment()
>>> seg.annotate(stimulus="step pulse", amplitude=10*nA)
>>> print(seg.annotations)
{'amplitude': array(10.0) * nA, 'stimulus': 'step pulse'}
```

SpikeTrain, AnalogSignal, and AnalogSIgnalArray inherits python-quantities:directly behave like np.array with units.

```
>>> import neo
>>> st = neo.SpikeTrain([3, 4, 5], units='sec', t_stop=10.0)
>>> print(st)
[ 3.  4.  5.] s
```

Class tour: schema



Class tour : definition

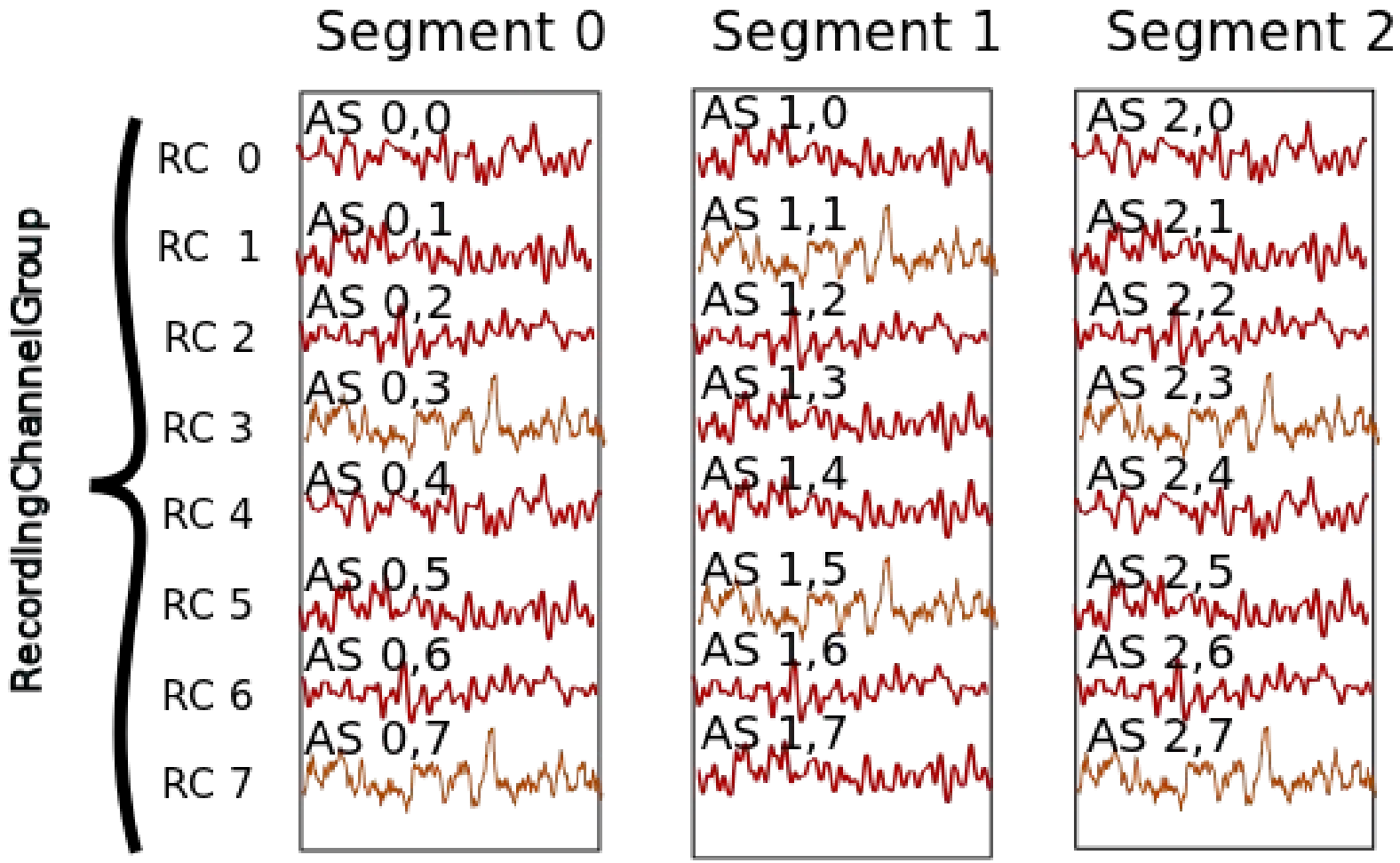
AnalogSignal: A regular sampling of a continuous, analog signal.
AnalogSignalArray: A regular sampling of a multichannel continuous analog signal. (2D NumPy array)
Spike: One action potential characterized by its time and waveform.
SpikeTrain: A set of action potentials (spikes) emitted by the same unit in a period of time (with optional waveforms).
Event and EventArray: A time point representng an event in the data, or an array of such time points.
Epoch and EpochArray: An interval of time representing a period of time in the data, or an array of such intervals.

Segment: A container for heterogeneous discrete or continous data sharing a common clock (time basis)
but not necessarily the same sampling rate, start time or end time. A Segment can be considered as equivalent
to a "trial", "episode", "run", "recording", etc., depending on the experimental context. May contain any of the data objects.
Block: The top-level container gathering all of the data, discrete and continuous, for a given recording session.
Contains Segment and RecordingChannelGroup objects.

RecordingChannelGroup: A group for associated RecordingChannel objects. This has several possible uses:
RecordingChannel objects of the same array.
Unit: A Unit gathers all the SpikeTrain objects within a common Block, possibly across several Segments,
that have been emitted by the same cell. A Unit is linked to RecordingChannelGroup objects from which it was detected.
This replaces the Neuron class in the previous version of Neo (v0.1).

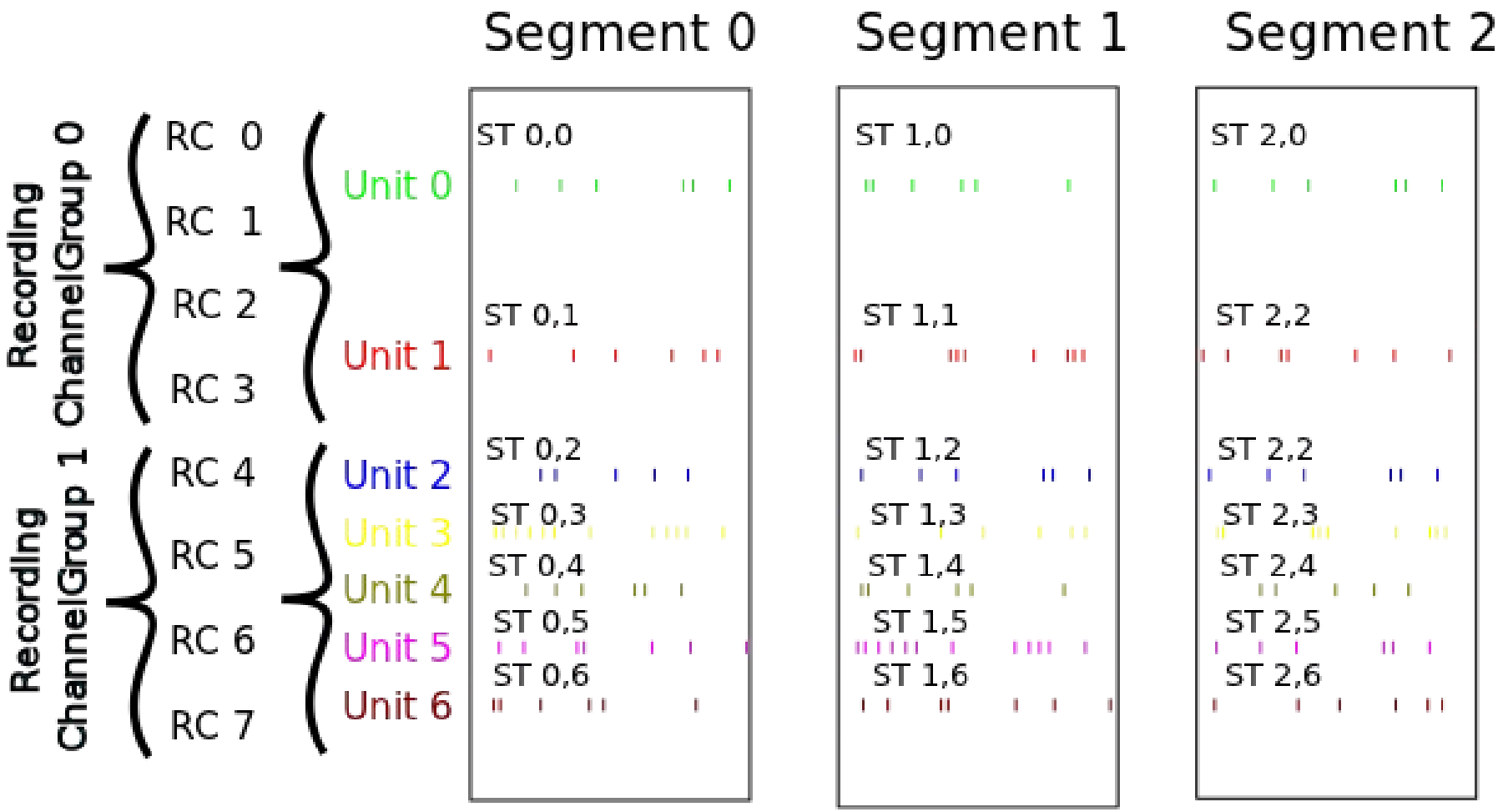
Class tour : Use case

RC = RecordingChannel
AS = AnalogSignal



Class tour : Use case

RC = RecordingChannel
ST = SpikeTrain



IO tour

First interest to have same classes :
Same API to read/write data files.

What is this API ?

- For each format you have an IO class
- The IO class can read or write one or several neo objects.

All formats are really different so we need a flexible API:

- Axon = Block+Segment+AnalogSignal+EventArray
- Plexon = Segment+SpikeTrain+AnalogSignal
- RAW = AnalogSignal

IO : tour

Module	Python 2	Python 3
AlphaOmegaIO	Yes	No
AsciiSignalIO	Yes	Yes
AsciiSpikeTrainIO	Yes	Yes
AxonIO	Yes	No
BlackrockIO	Yes	No
BrainwareDamIO	Yes	Yes
BrainwareF32IO	Yes	Yes
BrainwareSrcIO	Yes	Yes
ElanIO	Yes	No
HDF5IO	Yes	No
KlustakwikIO	Yes	No
MicromedIO	Yes	No
NeoMatlabIO	Yes	Yes
NeuroExplorerIO	Yes	No
NeuroscopeIO	Yes	Yes
PickleIO	Yes	Yes
PlexonIO	Yes	No
PyNNIO	Yes	Yes
RawBinarySignalIO	Yes	Yes
Spike2IO	Yes	Yes
TdtIO	Yes	No
WinEdrIO	Yes	Yes
WinWcpIO	Yes	Yes

IO tour : workflow

One class per format:

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

Different modes (file, dir, database, ...)

```
>>> from neo.io import MyFormatIO
>>> print MyFormatIO.mode
'file'
```

Examples

```
>>> reader = io.PlexonIO(filename='File_plexon_1.plx')
>>> reader = io.TdtIO(dirname='aep_05')
```

IO tour : workflow

Concept of readable/supported objects:

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

```
>>> MyFormatIO.readable_objects
[Segment]
```

Class offer reading method for readable objects

```
>>> seg = reader.read_segment()
>>> type(seg)
neo.core.Segment
```

All classes propose read() = read_all_block()

```
>>> bl = reader.read()
>>> print bl[0].segments[0]
neo.core.Segment
```

IO tour : workflow

Cascade option:

```
>>> seg = reader.read_segment( cascade=True)
>>> print(len(seg.analogsignals)) # this is N
>>> seg = reader.read_segment(cascade=False)
>>> print(len(seg.analogsignals)) # this is zero
```

Lazy option:

```
>>> seg = reader.read_segment(lazy=False)
>>> print(seg.analogsignals[0].shape) # this is N
>>> seg = reader.read_segment(lazy=True)
>>> print(seg.analogsignals[0].shape) # this is zero, the An
>>> print(seg.analogsignals[0].lazy_shape) # this is N
```

Deferred loading (load_lazy_object):

```
>>> lazy_sig = seg.analogsignals[0]  # Empty signal
>>> full_sig = reader.load_lazy_object(lazy_sig)
>>> print(lazy_sig.lazy_shape, full_sig.shape)  # Identical
>>> print(lazy_sig.segment)  # Has the link to the object "seg"
>>> print(full_sig.segment)  # Does not have the link: None
```

Projects using neo

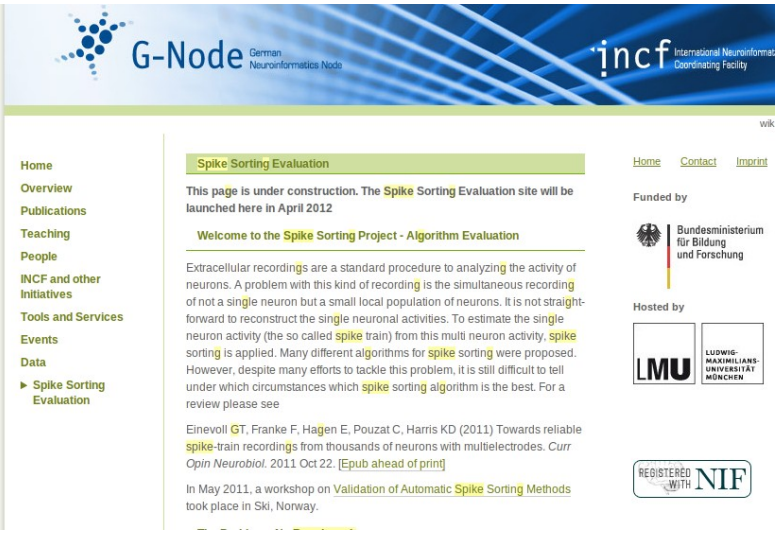
PyNN

SpykeViewer

Mozaik

OpenElectrophy

G-Node

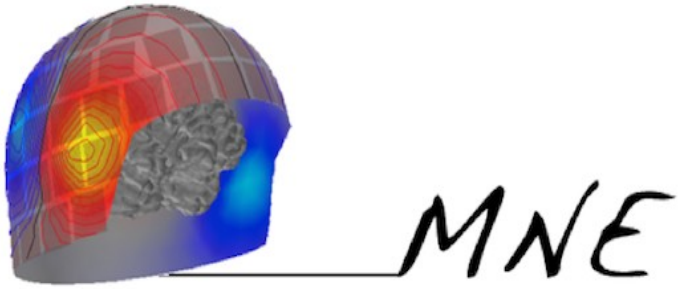


Helmholtz

DataJongleur

Projects that plan to use neo

Python MNE = EEG analysis and visualization



What is new 0.3

- Very similar to neo 0.2 (same model)
- New IO (ElphyIO, Brainware, Neuroscope, …)
- Performance improved.
- Small diff in neo.io:
  - read_all_blocks
  - Deferred loading with load_lazy_object.
- Code quality highly improved
  - Travis CI
  - Incredible clean up and test improvement thanks to heroic work of Todd Jennings.

Plans for neo 0.4

- Better model “auto description”
- Harmonize class (have or inherits quanitities)
- Converge Scalar vs Array object (SpikeTrain vs Spike)
- Change in relationship (RC, RCG and Unit directly to Block) ??
- Python 3.3 for all IO.
- Attribute as properties (test compliance)
- Add stream reading for IO that are able to (memmap or hdf5 or blaze)
- Why not NeoBlazeIO ? (Francesc Alted : are you volunteers ?)

In preparation project

- A visualization module for neo object in vispy ? Any volunteers for discussion ?

Pre Conclusion

Good point:

- Interoperability (5 projects)
- IO (23 class)
- API with relationship
- Domain specific (for name) but general design
- Force user to use quantities

Bad point:

- Model should simplified
- Not ready for “big data” (everything in memory, numpy)
- Force user to use quantities

Conclusion

- If your project generate data : write an IO for neo
- If your project manage signals and spikes in python : provide an interface to neo objects
- If your experimentalist colleague wants to read data set from commercial systems: neo.io

Post Conclusion

Like almost main of people here we send a submission for a paper in:
Frontier Python in Neuroscience II

Thank you to all editors!

If editor is in this room : why choosing a reviewer that code in Matlab ??? Is it for fun ?

Thanks to the the neo team

Samuel Garcia
Andrew Davison
Chris Rodgers
Pierre Yger
Luc Estabanez
Andrey Sobolev
Thierry Brizzi
Florent Jaillet
Philipp Rautenberg
Thomas Wachtler
Cyril Dejean
Robert Pröpper ← incredible effort for the project!
Domenico Guarino
Todd Jennings ← incredible effort for the project!