Introduction to NIX and NIX-integration into RELACS

Jan Grewe

Neuroethology
Institute for Neurobiology
Eberhard Karls Universität, Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

A brief history of non-reproducible science

Lack of information as a problem

Nature Reviews Genetics | AOP, published online 27 December 2012; doi:10.1038/nrg3394

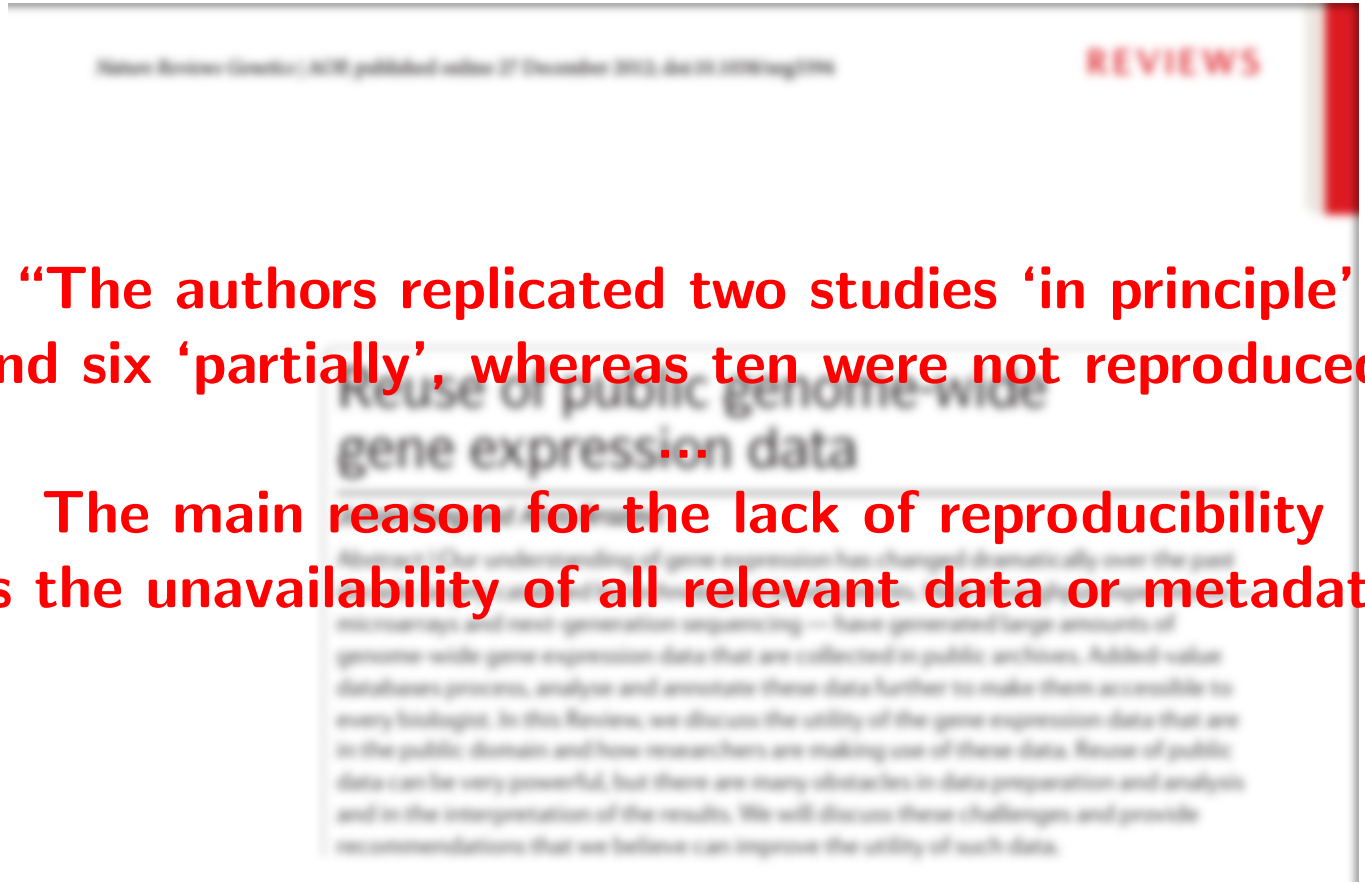
Reuse of public genome-wide gene expression data

Johan Rung and Alvis Brazma

Abstract | Our understanding of gene expression has changed dramatically over the past decade, largely catalysed by technological developments. High-throughput experiments — microarrays and next-generation sequencing — have generated large amounts of genome-wide gene expression data that are collected in public archives. Added-value databases process, analyse and annotate these data further to make them accessible to every biologist. In this Review, we discuss the utility of the gene expression data that are in the public domain and how researchers are making use of these data. Reuse of public data can be very powerful, but there are many obstacles in data preparation and analysis and in the interpretation of the results. We will discuss these challenges and provide recommendations that we believe can improve the utility of such data.

A brief history of non-reproducible science

Lack of information as a problem



“The authors replicated two studies ‘in principle’ and six ‘partially’, whereas ten were not reproduced, ...

The main reason for the lack of reproducibility was the unavailability of all relevant data or metadata:”

A brief history of non-reproducible science

Lack of information as a problem

Cell Reports
Commentary

Cell PRESS
Open ACCESS

Sorting Out the FACS: A Devil in the Details

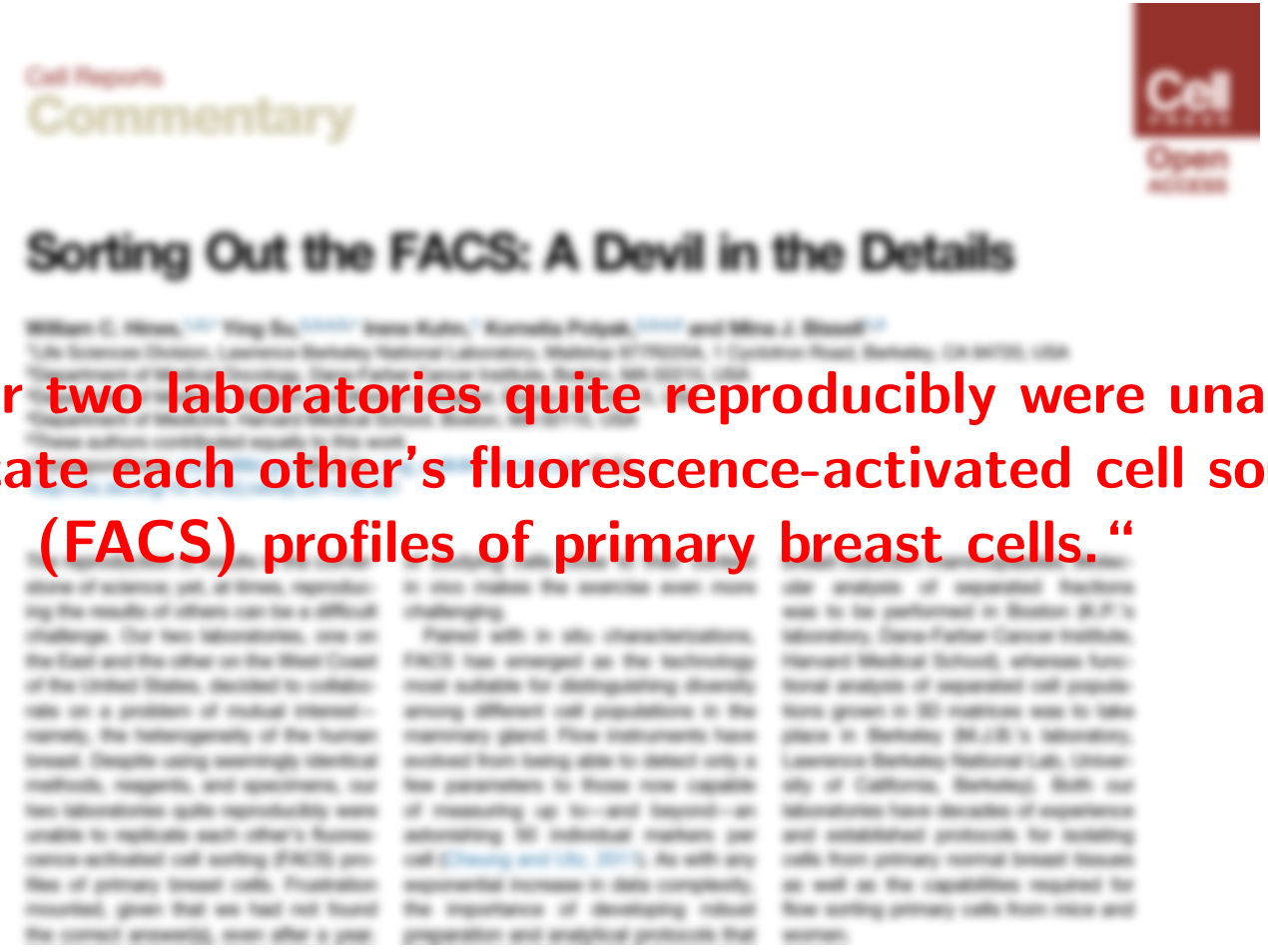
William C. Hines,[1,5,*] Ying Su,[2,3,4,5,*] Irene Kuhn,[1] Kornelia Polyak,[2,3,4,5] and Mina J. Bissell[1,5]
[1]Life Sciences Division, Lawrence Berkeley National Laboratory, Mailstop 977R225A, 1 Cyclotron Road, Berkeley, CA 94720, USA
[2]Department of Medical Oncology, Dana-Farber Cancer Institute, Boston, MA 02215, USA
[3]Department of Medicine, Brigham and Women's Hospital, Boston, MA 02115, USA
[4]Department of Medicine, Harvard Medical School, Boston, MA 02115, USA
[5]These authors contributed equally to this work
*Correspondence: chines@lbl.gov (W.C.H.), ying_su@dfci.harvard.edu (Y.S.)
http://dx.doi.org/10.1016/j.celrep.2014.02.021

The reproduction of results is the cornerstone of science; yet, at times, reproducing the results of others can be a difficult challenge. Our two laboratories, one on the East and the other on the West Coast of the United States, decided to collaborate on a problem of mutual interest—namely, the heterogeneity of the human breast. Despite using seemingly identical methods, reagents, and specimens, our two laboratories quite reproducibly were unable to replicate each other's fluorescence-activated cell sorting (FACS) profiles of primary breast cells. Frustration mounted, given that we had not found the correct answer(s), even after a year. of studying cells close to their context in vivo makes the exercise even more challenging.

Paired with in situ characterizations, FACS has emerged as the technology most suitable for distinguishing diversity among different cell populations in the mammary gland. Flow instruments have evolved from being able to detect only a few parameters to those now capable of measuring up to—and beyond—an astonishing 50 individual markers per cell (Cheung and Utz, 2011). As with any exponential increase in data complexity, the importance of developing robust preparation and analytical protocols that breast reduction mammoplasties. Molecular analysis of separated fractions was to be performed in Boston (K.P.'s laboratory, Dana-Farber Cancer Institute, Harvard Medical School), whereas functional analysis of separated cell populations grown in 3D matrices was to take place in Berkeley (M.J.B.'s laboratory, Lawrence Berkeley National Lab, University of California, Berkeley). Both our laboratories have decades of experience and established protocols for isolating cells from primary normal breast tissues as well as the capabilities required for flow sorting primary cells from mice and women.

A brief history of non-reproducible science

Lack of information as a problem



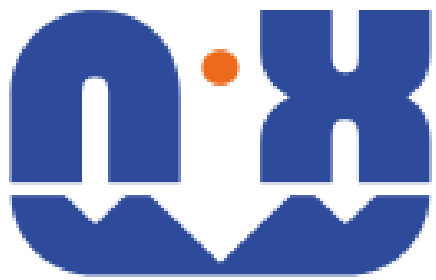
"... our two laboratories quite reproducibly were unable to replicate each other's fluorescence-activated cell sorting (FACS) profiles of primary breast cells."

Towards standardization in the Neurosciences

Reproducing results is hindered by the lack of data

- Improving the situation in the neurosciences was the reason to establish the INCF and the national nodes.
- The German Neuroinformatics Node (G-Node, at the Bernstein Center in Munich, established 2008) is one of the founding members.
- We (J.B. and J.G) have been involved right from the start.
- Were members of the INCF "Standards for electrophysiology" Task-force.
- Discussions therein were the starting point of the *NIX* development back in 2012.

Introduction to the *NIX* Project



What is *NIX*?

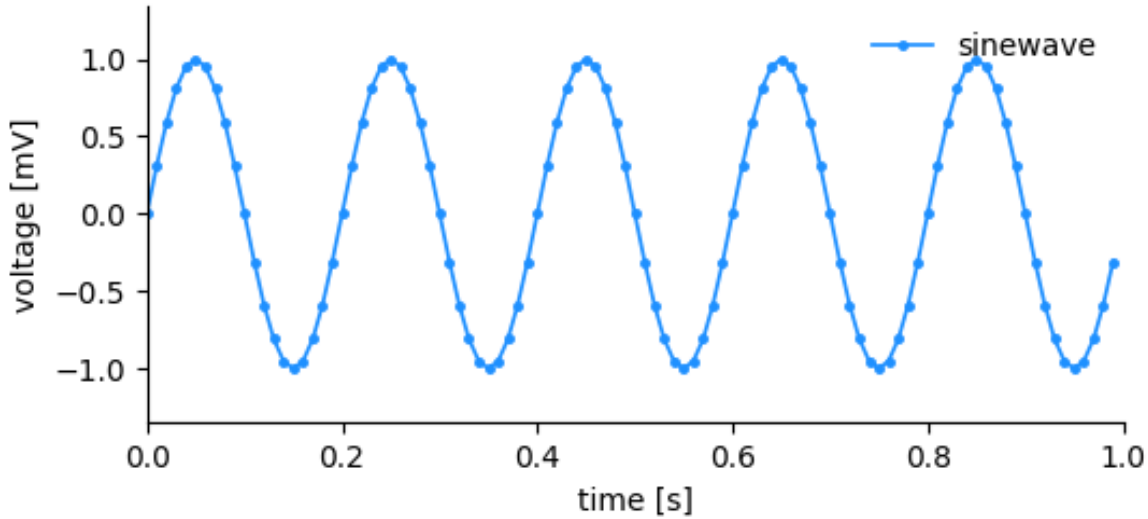
Neuroscience Information exchange format

Design goals:

- Create a **generic** data model that can store a multitude of scientific data structures.
- Self-contained, i.e. all required information is stored in one container.
- We always want to be able to create a basic, yet completely labeled, plot of the data without asking someone else or addressing a different resource.
- Use as little entities as possible.
- Support embedded data annotations.
- Support standardization.
- Free, open-source, ...

Introduction to *NIX*

Which information needs to be stored?



The plot shows voltage measurement taken at given times. We need to store:

1. the y-values (voltage measurements).
2. the x-values (times at which the data was sampled).
3. the label of the y-axis (voltage) and the unit (mV).
4. the label of the x-axis (time) and the unit (s).
5. the name of the trace for the legend.

Introduction to *NIX*

Which information needs to be stored?

Some of the aforementioned information is ...

- directly related to the data.
- related to the dimensions of the dataset.

Introduction to *NIX*

Which information needs to be stored?

Some of the aforementioned information is ...

- directly related to the data.
- related to the dimensions of the dataset.

Accordingly, ...

- data-related information goes into the **DataArray** entity.
- dimension-related info goes into “Dimension descriptors”.

Introduction to *NIX*

The **DataArray**

- The **DataArray** is the core entity of the *NIX* data model.
- It stores the actual data.
- Plus a few more pieces of information.

DataArray

Field	Type
id	string
name	string
type	string
definition	string
label	string
unit	string
data	n-d data
dtype	double, int, ...
dimensions	Dimension[]
sources[]	Source []
metadata	Section
polynom_coefficients	double []
expansion_origin	double
createdAt	datetime
updatedAt	datetime

Introduction to *NIX*

Dimension descriptors

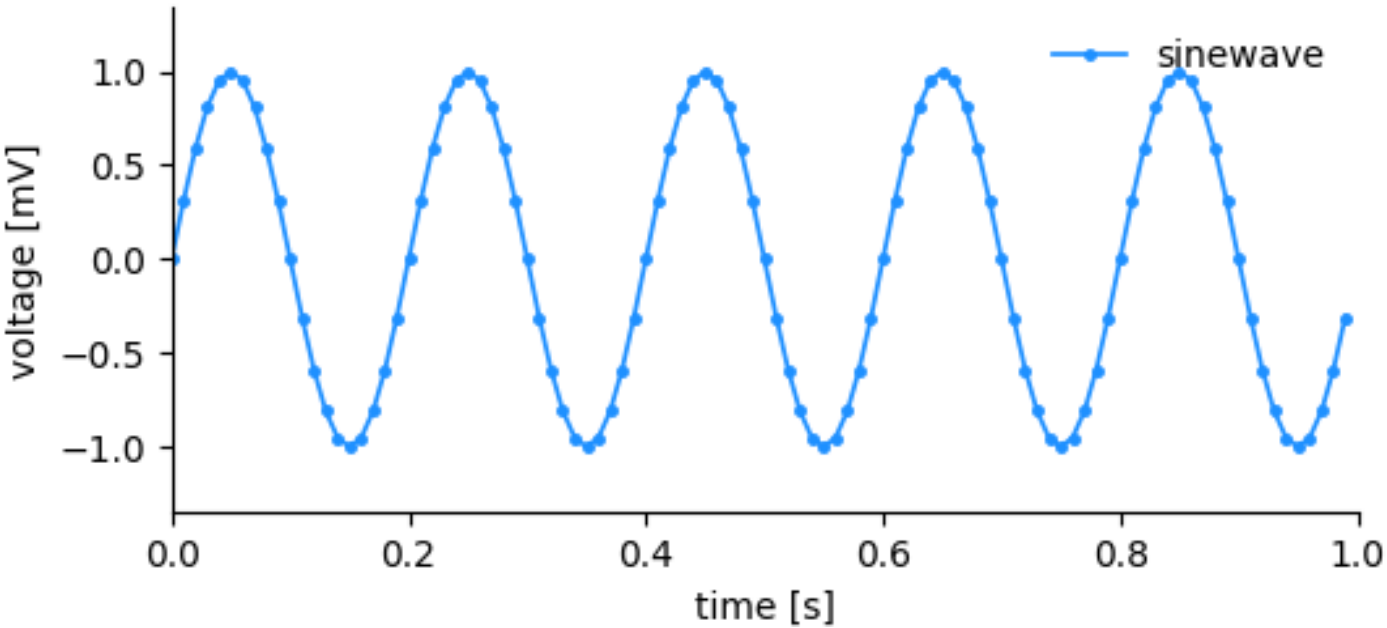
Three (and a half) types of dimensions:

1. SampledDimension
2. RangeDimension
3. SetDimension

Introduction to *NIX*

Dimension Descriptors

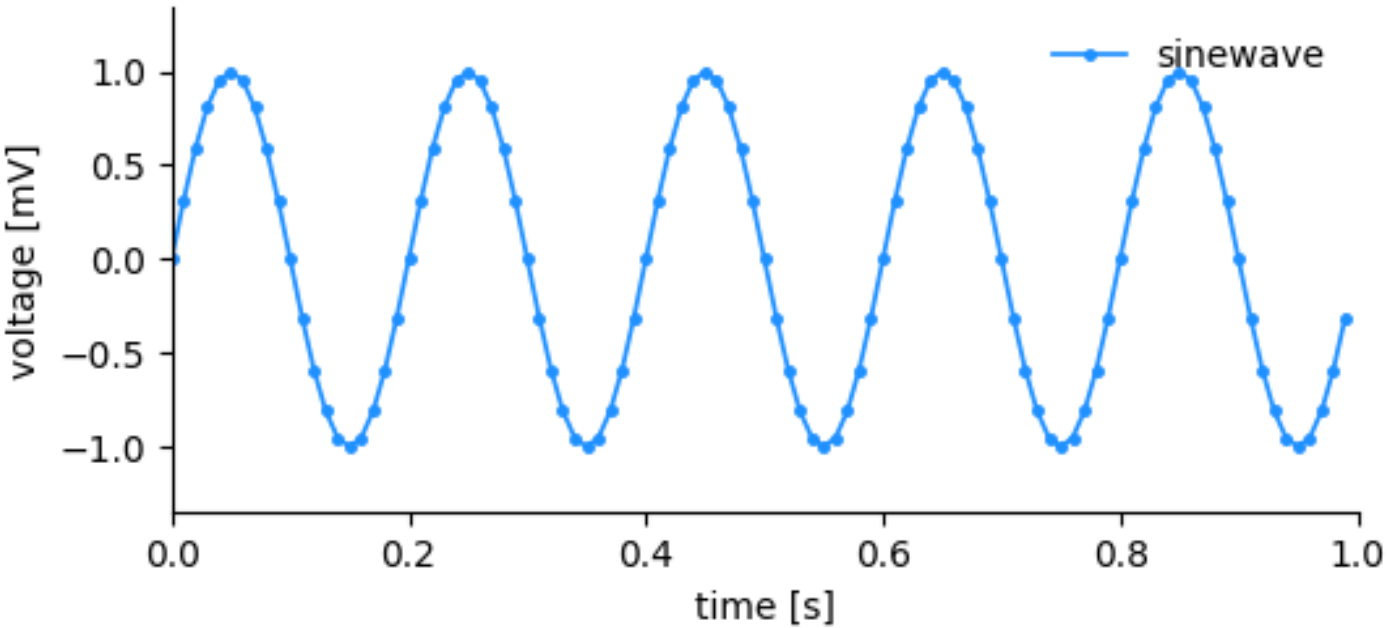
1: SampledDimension



Introduction to *NIX*

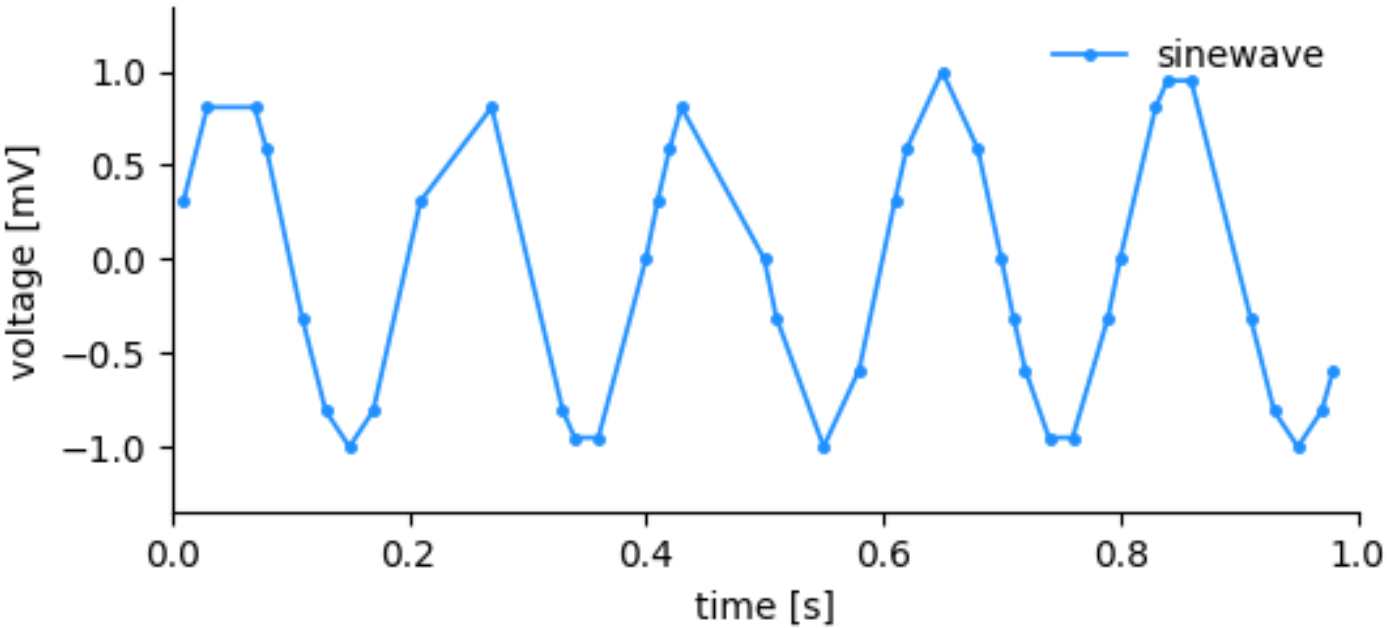
Dimension Descriptors

1: SampledDimension

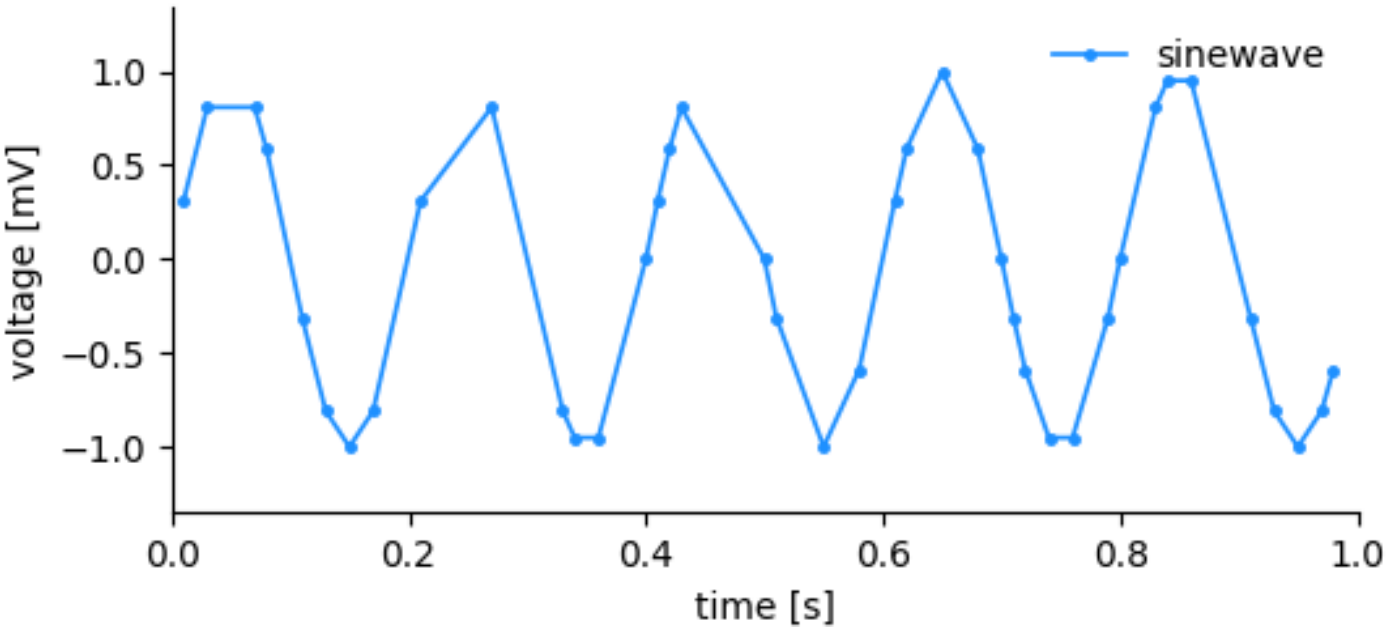


Used if data has been sampled in regular intervals. The descriptor stores the **label**, the **unit**, the **sampling interval** and the **offset** of the dimension (axis).

2: RangeDimension



2: RangeDimension

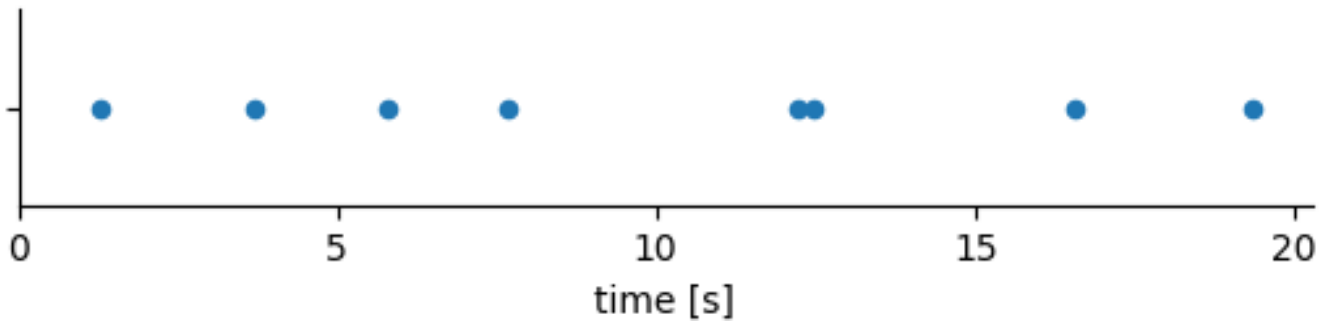


Used if the data has been sampled in irregular intervals. The descriptor stores the **label**, the **unit**, and the **ticks** of the dimension (axis).

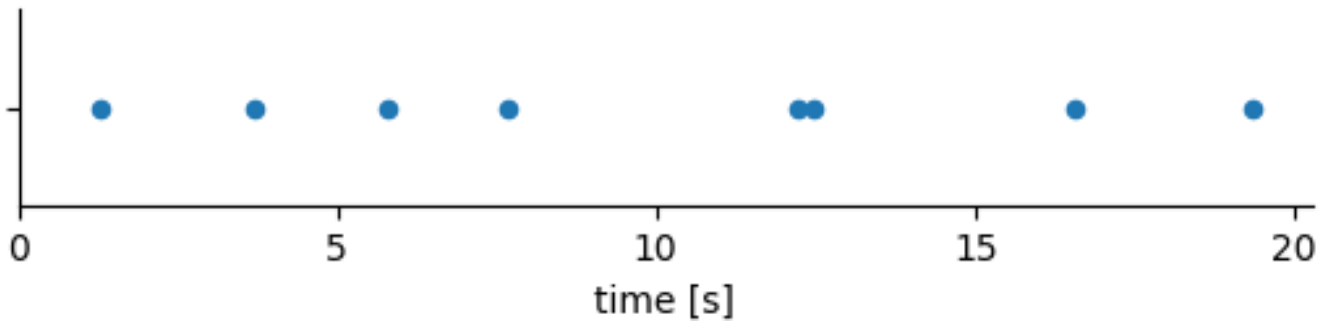
Introduction to *NIX*

Dimension Descriptors

2b: AliasRangeDimension



2b: AliasRangeDimension

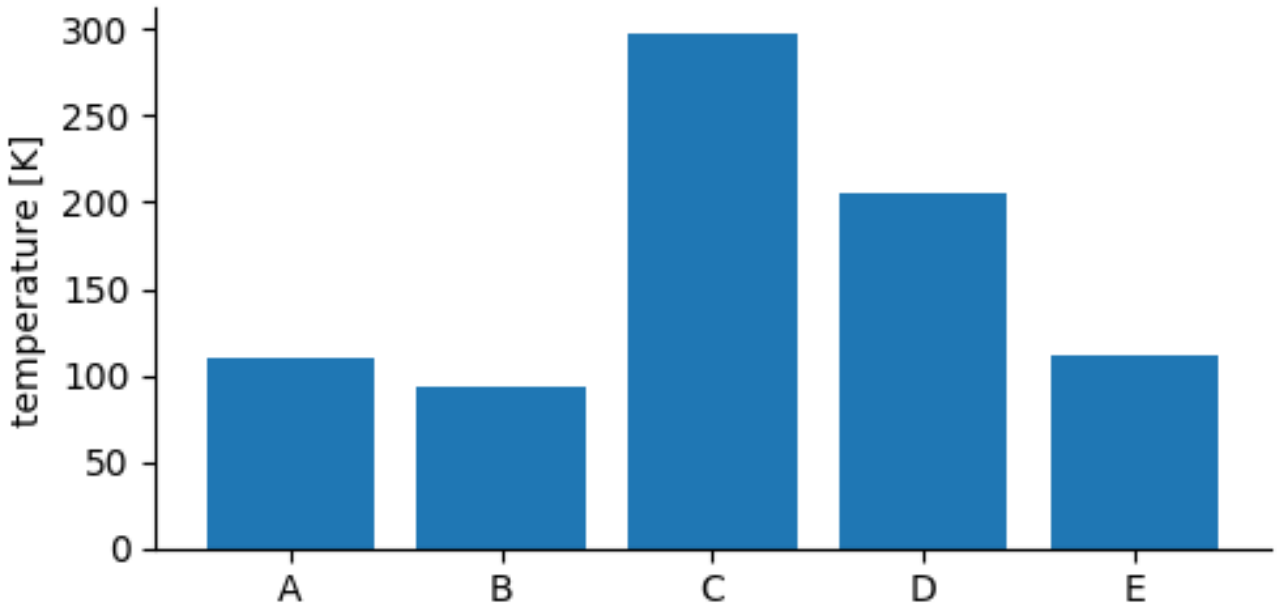


Used if the stored data are e.g the time-points of certain events. The descriptor is basically a link to the stored data itself and can only be applied when the data is 1D.

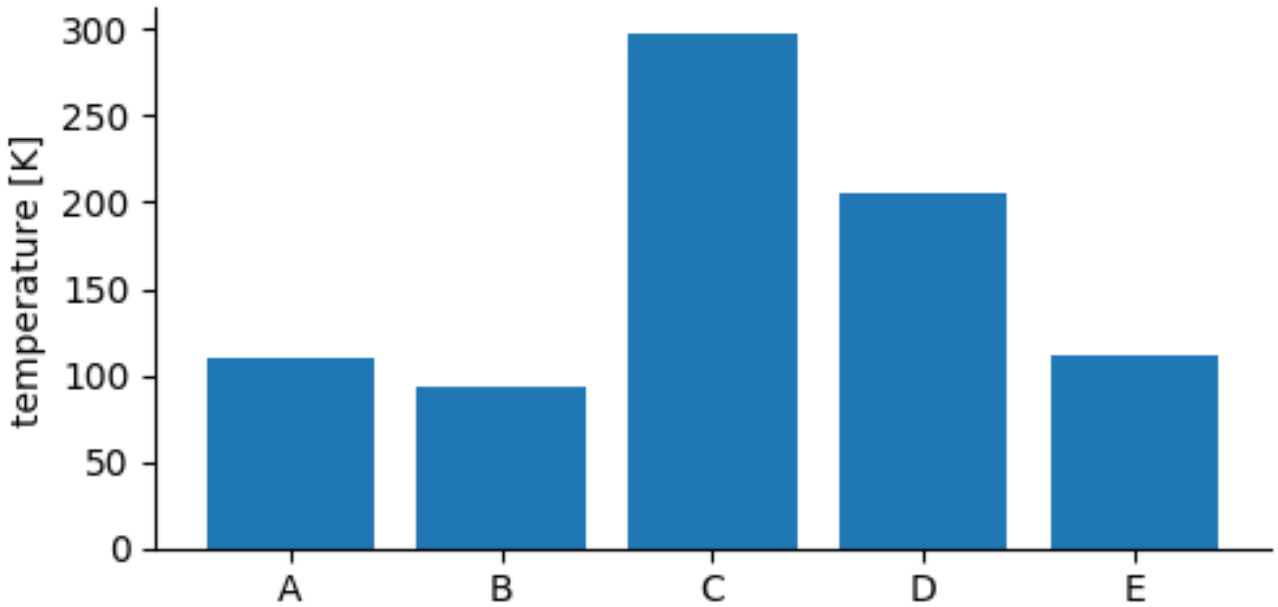
Introduction to *NIX*

Dimension Descriptors

3: SetDimension



3: SetDimension



Used if the axis represents categories that may not have a natural order. The descriptor stores (optional) **label**s for each entry along the respective dimension.

Introduction to *NIX*

So far, we can ...

1. store n-dimensional data.
2. provide information about type of values and the unit.
3. describe the dimensions of the data.

Introduction to *NIX*

So far, we can ...

1. store n-dimensional data.
2. provide information about type of values and the unit.
3. describe the dimensions of the data.

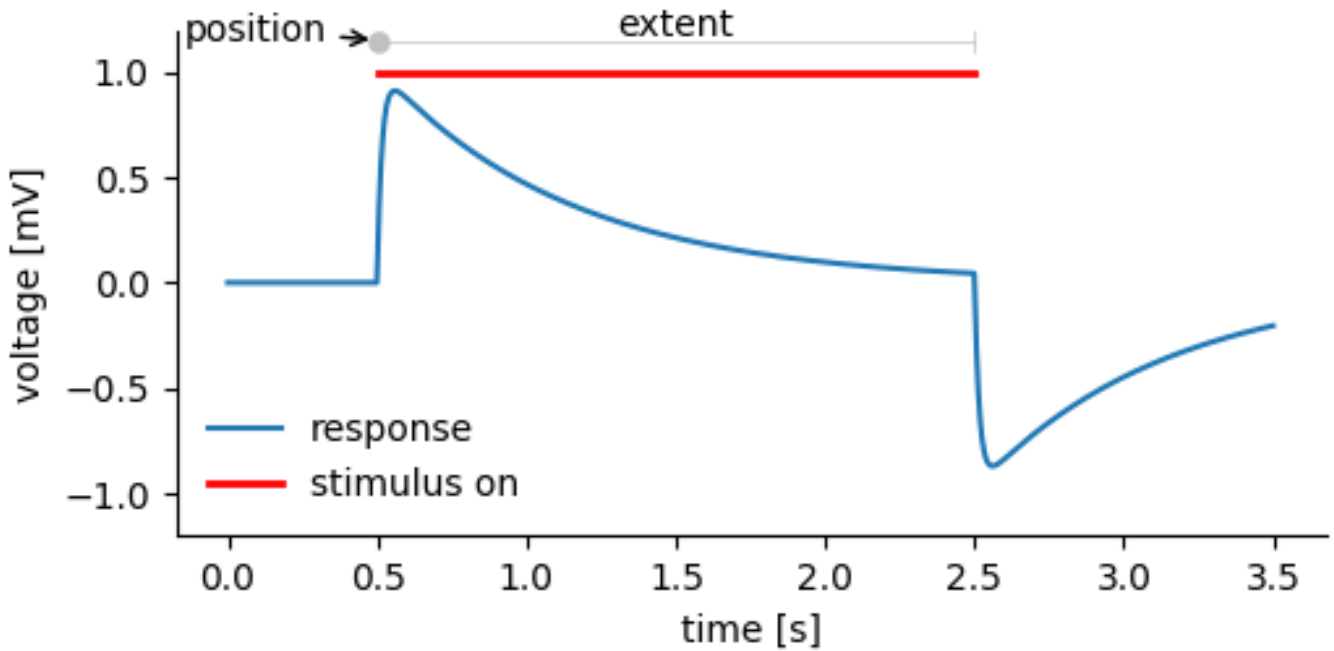
But we want more:

1. Annotating/tagging data (regions or points).
2. Add arbitrary metadata.

Introduction to *NIX*

Tagging things

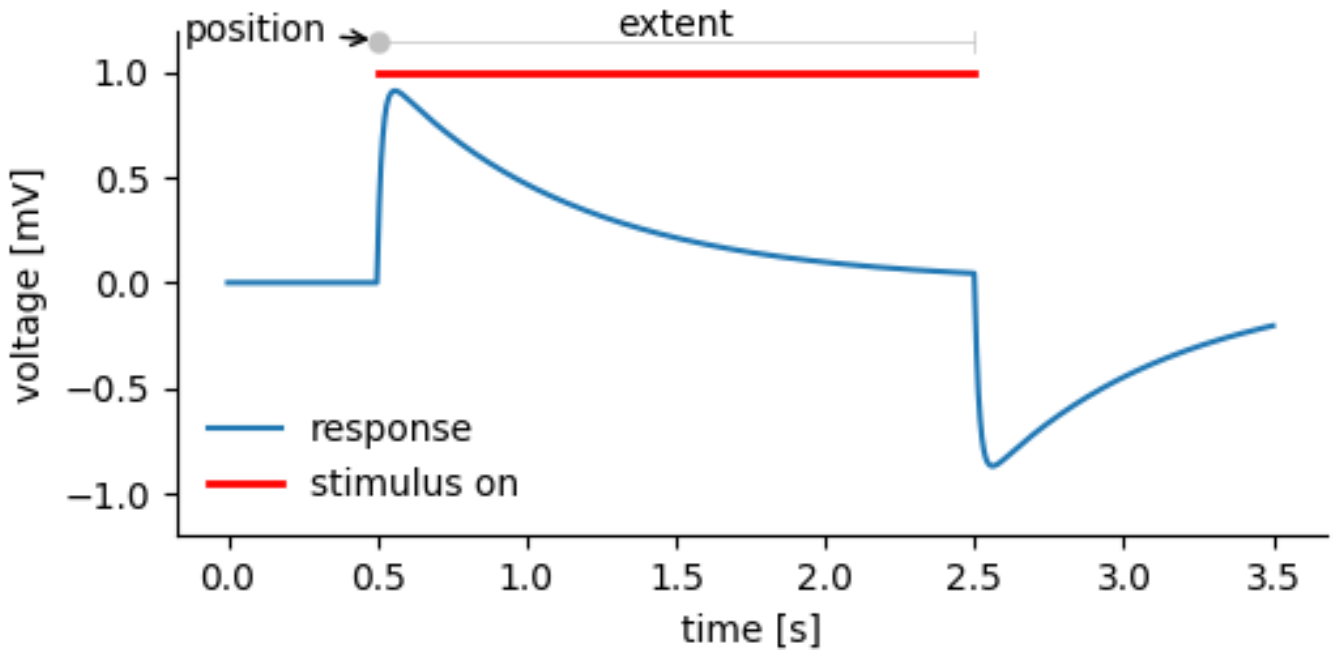
We may record a system's response over time and want to note (tag) the time a stimulus was on.



Introduction to *NIX*

Tagging things

We may record a system's response over time and want to note (tag) the time a stimulus was on.

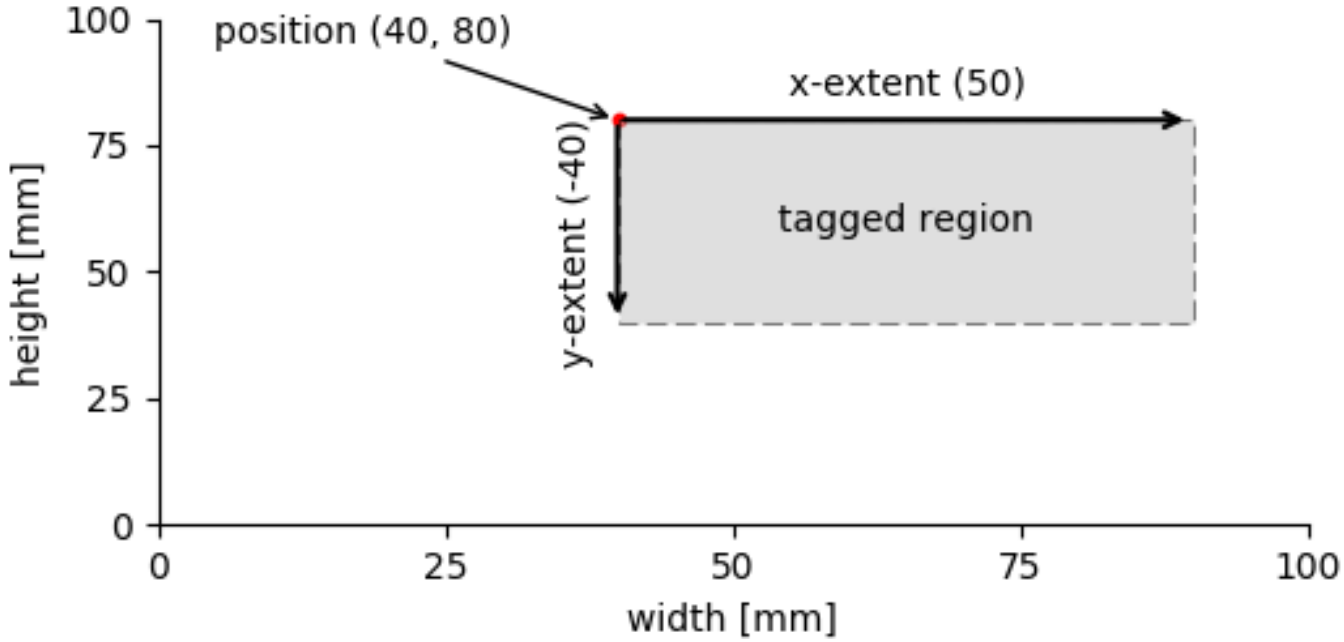


Tags are used to annotate points or regions in the data stored in a **DataArray**.

Introduction to *NIX*

Tagging things in 2D

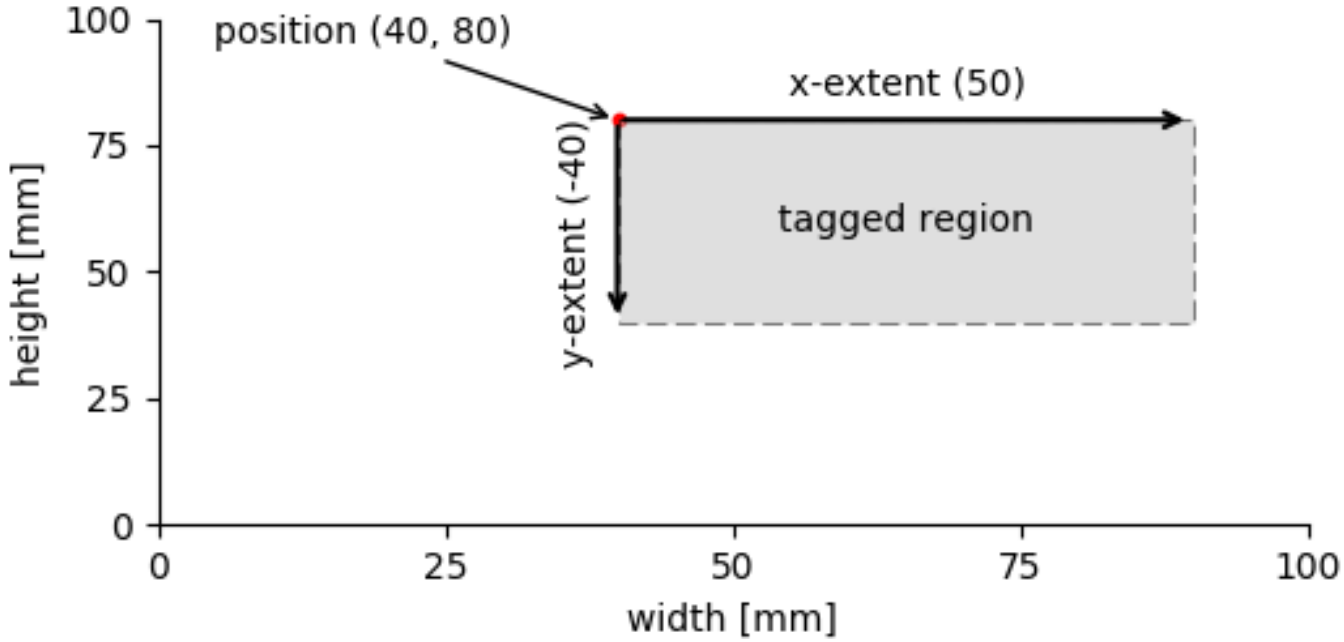
The approach can be extended into n-D.



Introduction to *NIX*

Tagging things in 2D

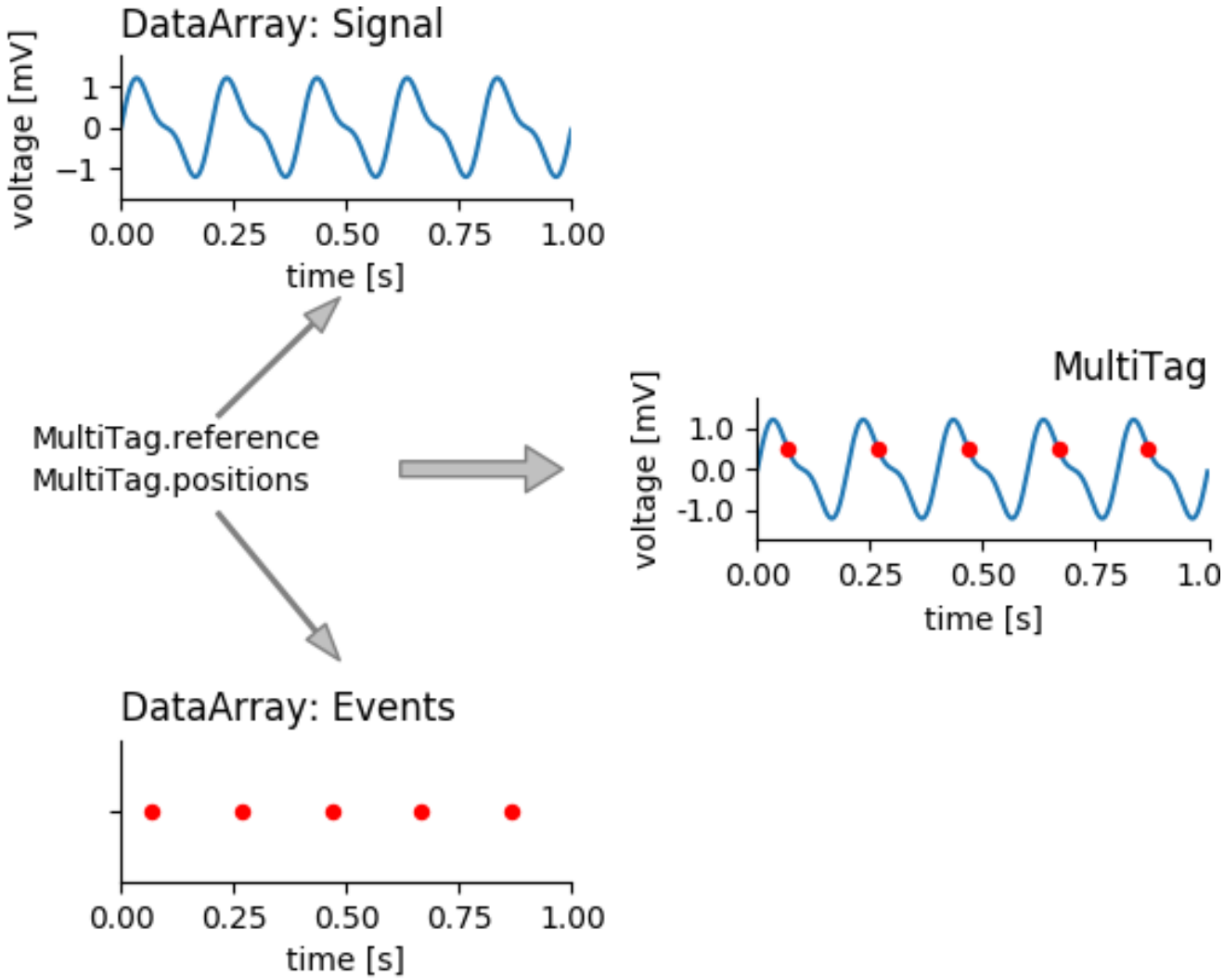
The approach can be extended into n-D.



position and **extent** are now vectors of length n.

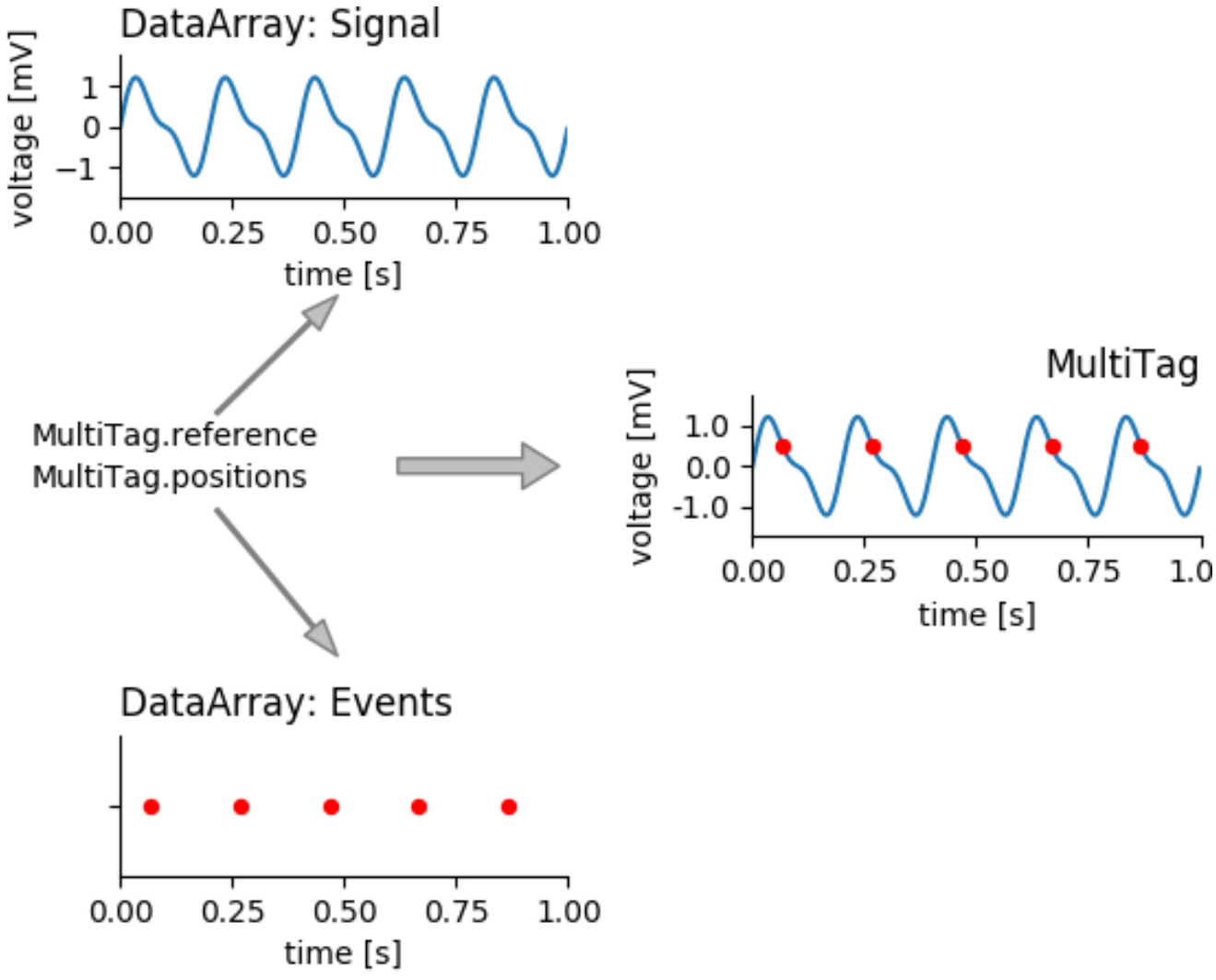
Introduction to *NIX*

Tagging many things at once



Introduction to *NIX*

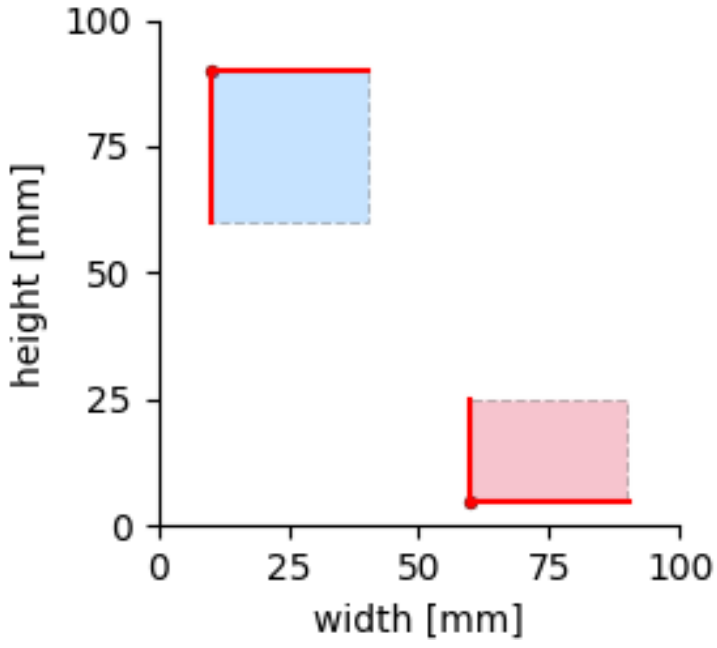
Tagging many things at once



A **MultiTag** entity is used to bind two (or more) **DataArrays**.

Introduction to *NIX*

Tagging multiple regions in 2D



Positions DataArray

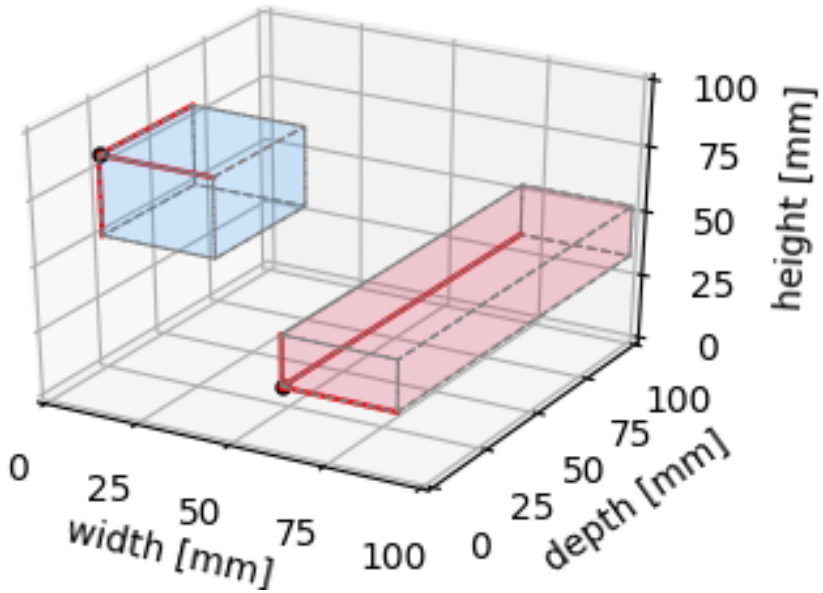
	Pos 1	Pos 2
width	10	60
height	90	5

Extents DataArray

	Ext 1	Ext 2
width	30	30
height	-30	20

Introduction to *NIX*

Tagging multiple hyperslaps



Positions DataArray

	Pos 1	Pos 2
width	10	60
height	90	20
depth	10	5

Extents DataArray

	Ext 1	Ext 2
width	30	30
height	-30	20
depth	40	60

Introduction to *NIX*

Tagging

Tags act as a hub that ...

- links **DataArrays**.
- tags points or regions in n-D.

But we need more:

- A way to give some more meaning to the tagged regions.
- Add metadata.

Tag

Field	Type
id	string
name	string
type	string
definition	string
position	double []
extent	double []
units	string []
references	DataArray []
features	Feature []
sources	Source []
metadata	Section
createdAt	datetime
updatedAt	datetime

Introduction to *NIX*

Features

Feature entities are used to attach some more information to tagged regions.

For example:

- Each tagged region has characteristics that need to be stored (e.g. the $\frac{\Delta f}{f}$ ratio in a region of interest).
- In each tagged regions a stimulus of a certain intensity was used.
- The power spectrum of the neuronal response in the tagged regions have been estimated.
- ...

Introduction to *NIX*

Features

Feature entities are used to attach some more information to tagged regions.

For example:

- Each tagged region has characteristics that need to be stored (e.g. the $\frac{\Delta f}{f}$ ratio in a region of interest).
- In each tagged regions a stimulus of a certain intensity was used.
- The power spectrum of the neuronal response in the tagged regions have been estimated.
- ...

Feature

Field	Type
id	string
link_type	LinkType
data	DataArray

Depending on the type of feature there are three ways how the features of a tag should be interpreted (by the library or the user).

Introduction to *NIX*

Features

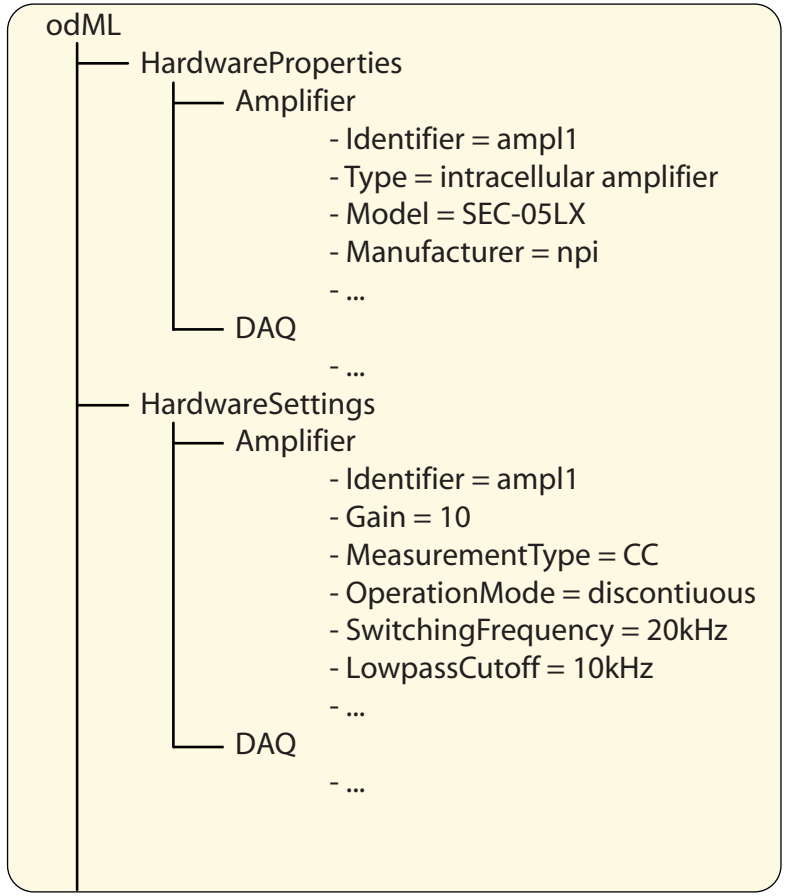
Depending on the type of feature there are three ways how the features of a tag should be interpreted (by the library or the user).

1. **Indexed**: For each position of the Tag, there is an entry along the first dimension of the data which contains the feature.
2. **Tagged**: positions and extents of the Tag also apply to the feature.
3. **Untagged**: all the data stored in the feature relates to the Tag irrespective of positions and extents.

Introduction to *NIX*

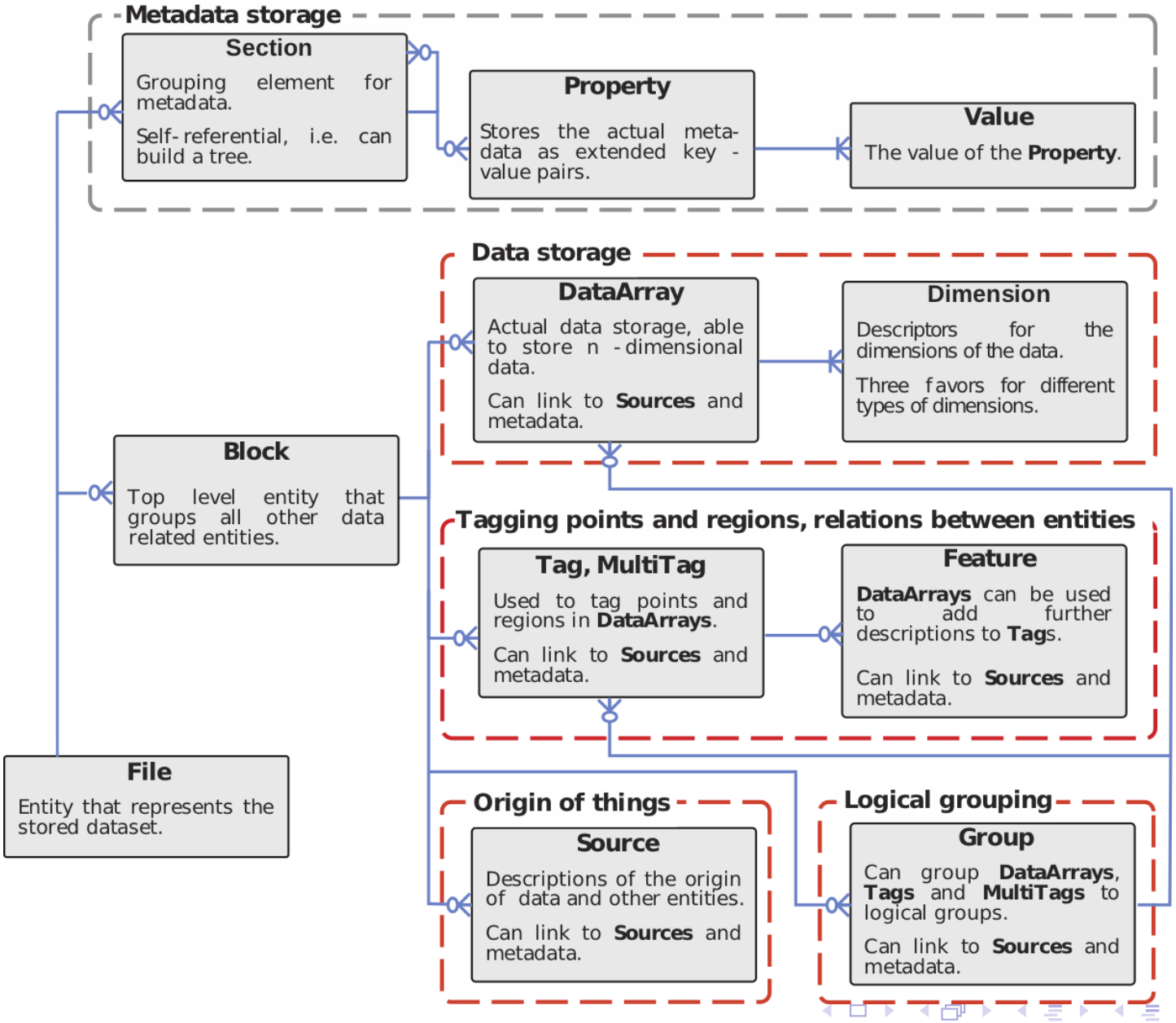
Adding arbitrary metadata

1. Many entities of the *NIX* data model allow adding more metadata.
2. We use a simple tree-like structure of **Sections**, **Properties** and **Values** for storing these.



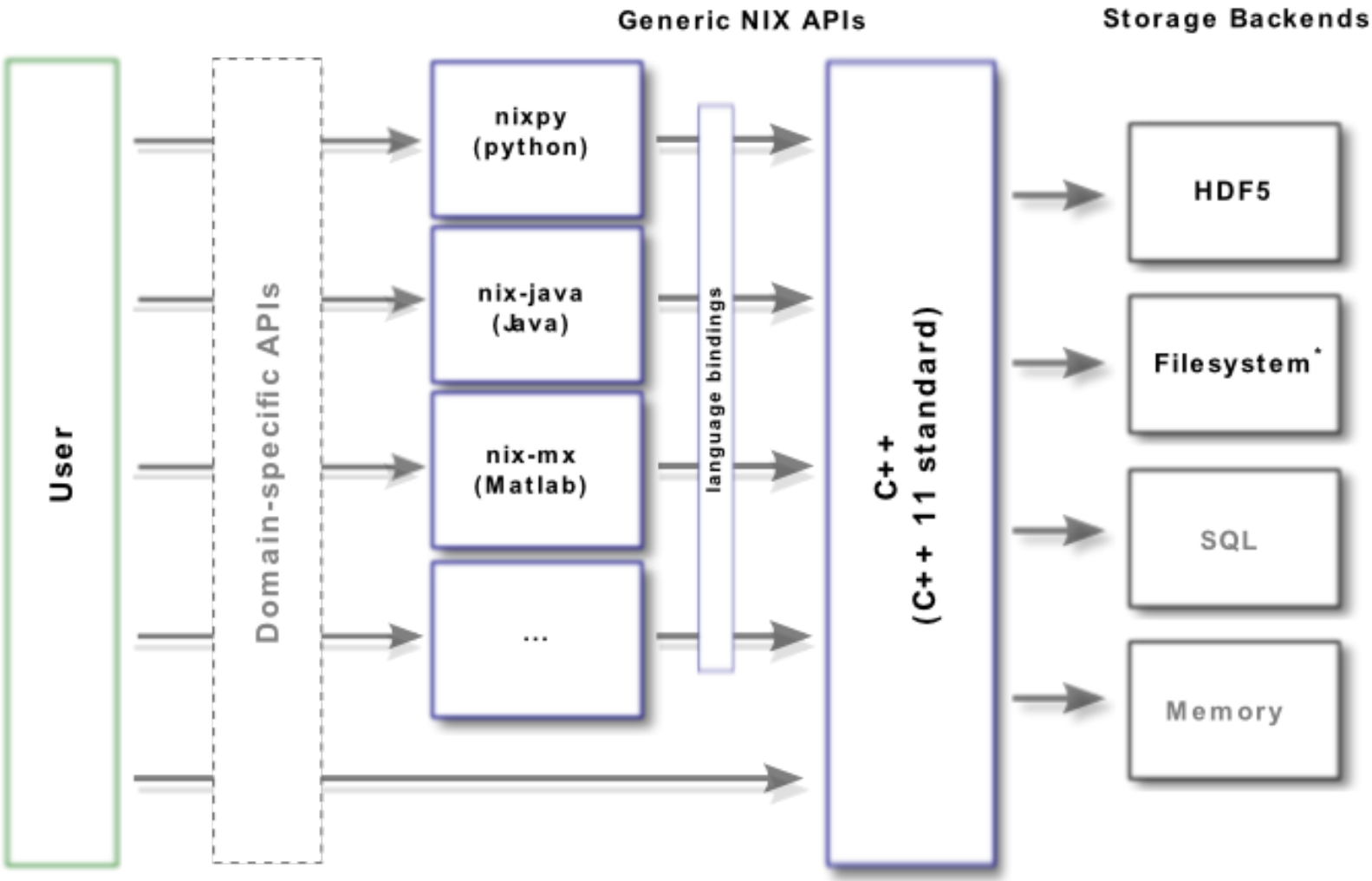
Introduction to *NIX*

NIX Data model



Introduction to *NIX*

API design



* experimental state

Introduction to *NIX*

NIX ecosystem

1. NIX C++ library:
   `https://github.com/g-node/nix`
2. nixpy python library:
   `https://github.com/g-node/nixpy`
3. nix-java java library:
   `https://github.com/g-node/nix-java`
4. nix-mx matlab library:
   `https://github.com/g-node/nix-mx`

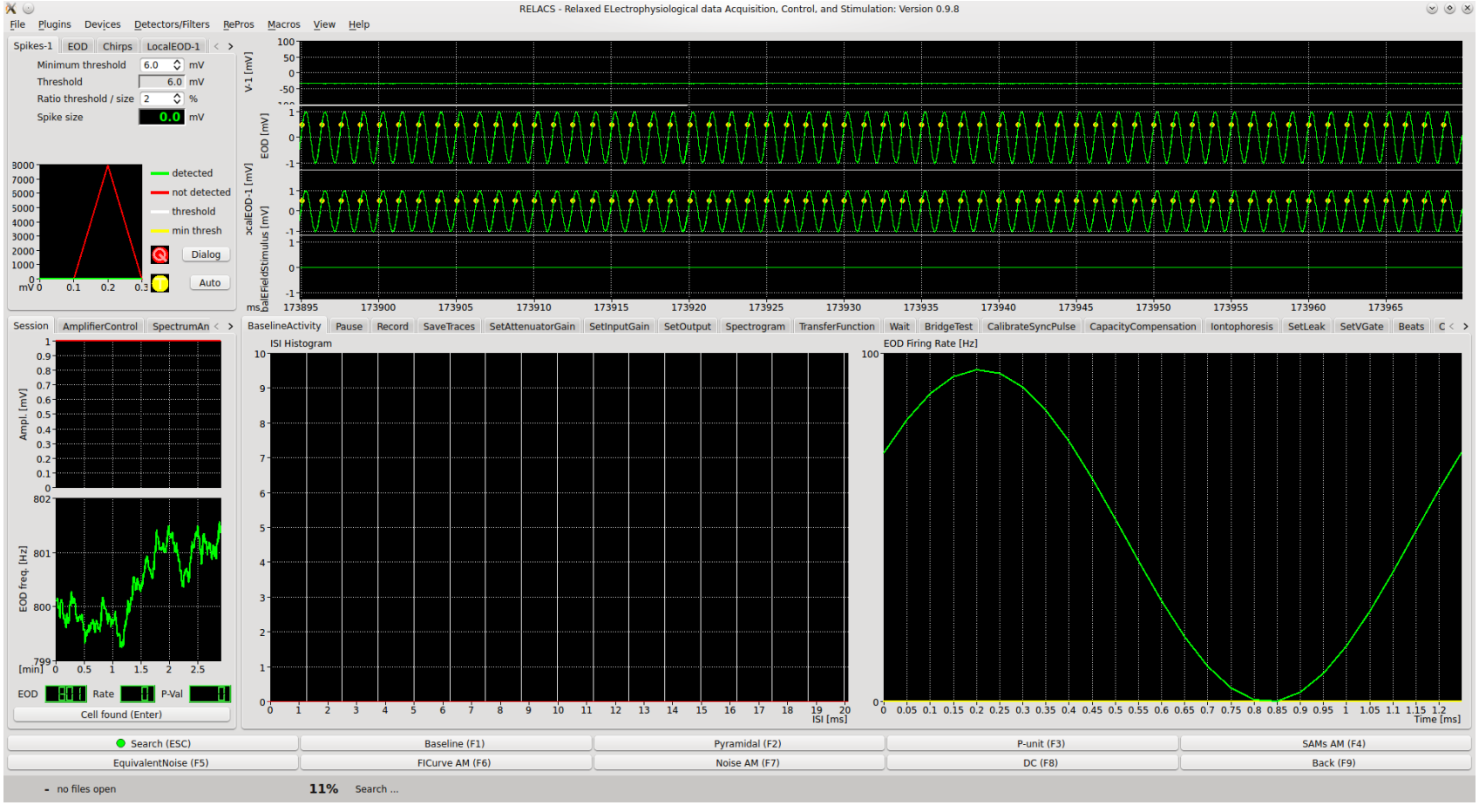
5. storage backend for the NEO object model
   `neuralensemble.org`
6. nixview: Viewer for nix files
   `https://github.com/bendalab/nixview`

phew ...

Integration of *NIX* in RELACS

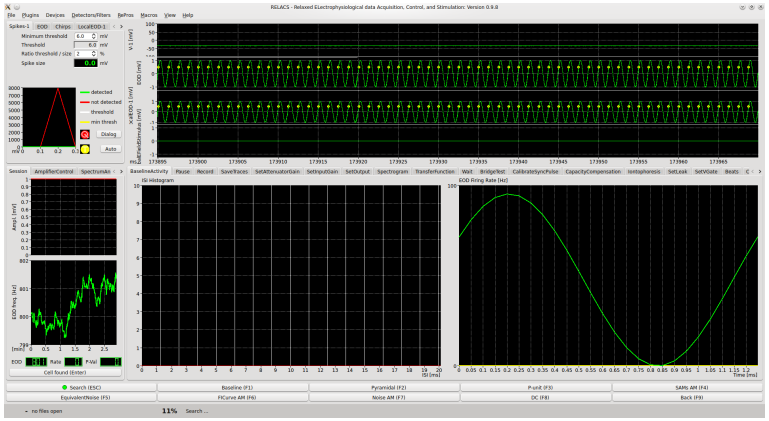
Integration of *NIX* in RELACS

RELACS overview



Integration of *NIX* in RELACS

RELACS overview



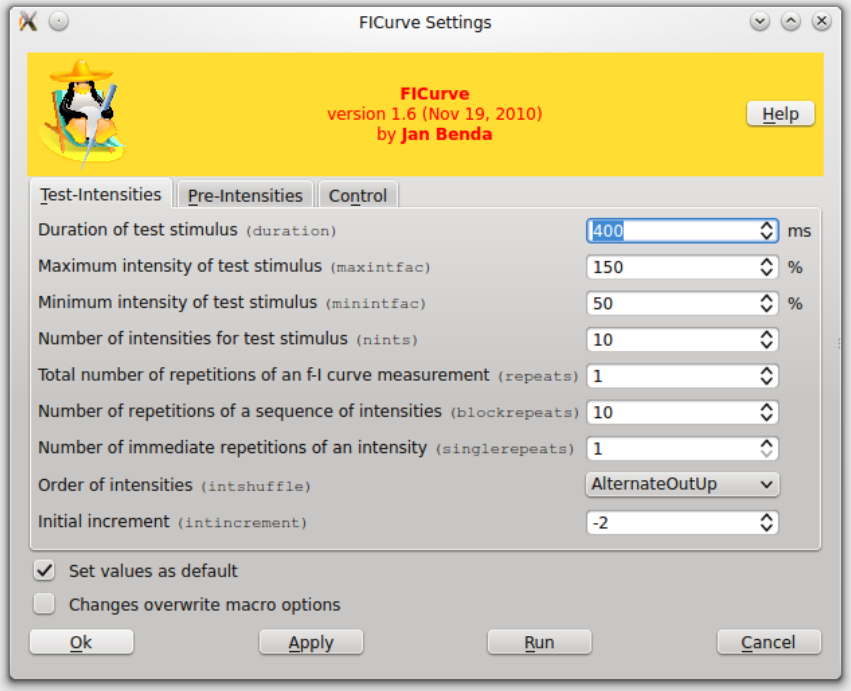
Storing various datasets.

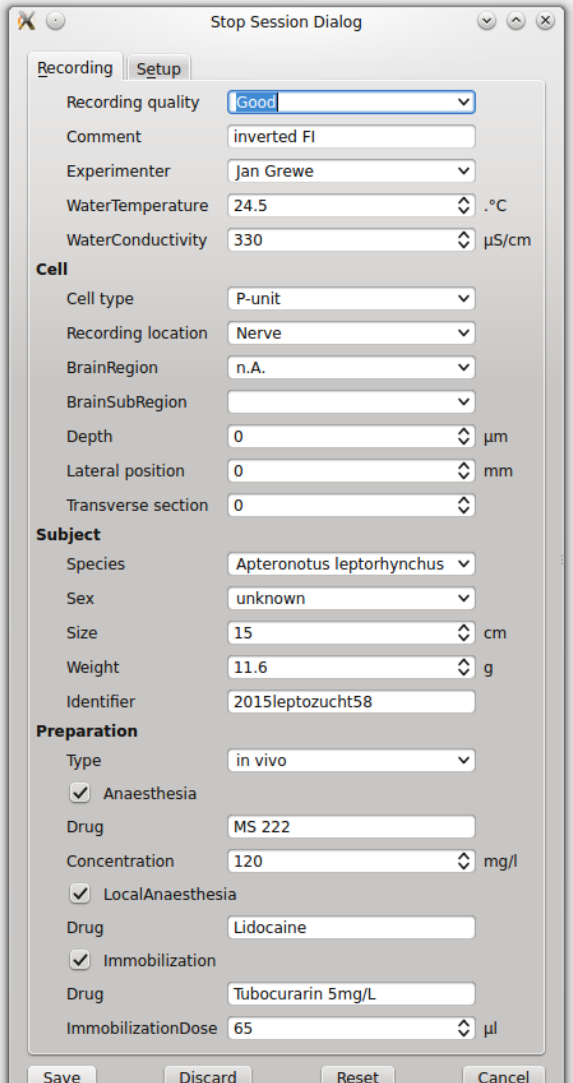
1. The membrane voltage (V-1).
2. The fish's electric organ discharge (EOD), global measurement.
3. The local measurement of the EOD (LocalEOD-1).
4. The stimulus put into the tank (GlobalEFieldStimulus).
5. The times of detected spikes (Spikes-1).
6. Sometimes the times of EOD discharges and detected chirps.

Integration of *NIX* in RELACS

RELACS overview

1. Relacs acquires data and controls the stimulus output.
2. knows a lot of metadata.

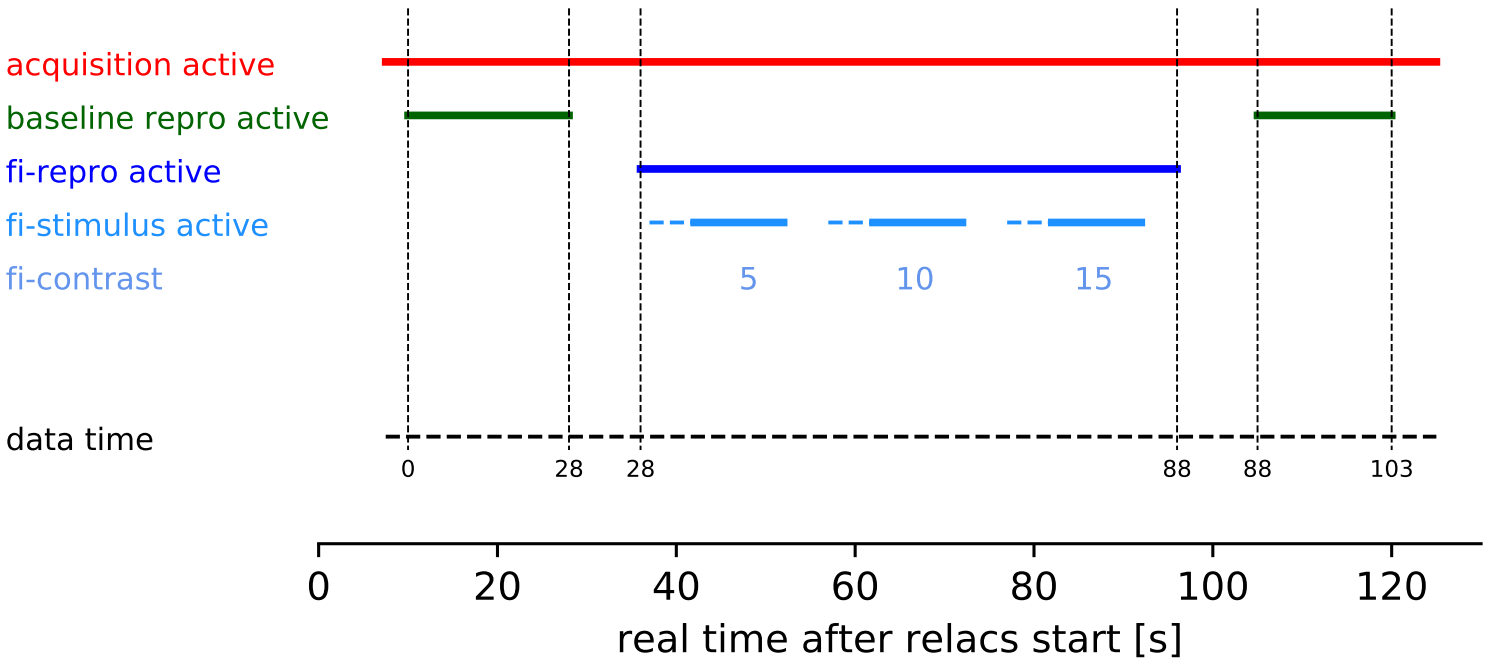




Integration of *NIX* in RELACS

A recording session in RELACS

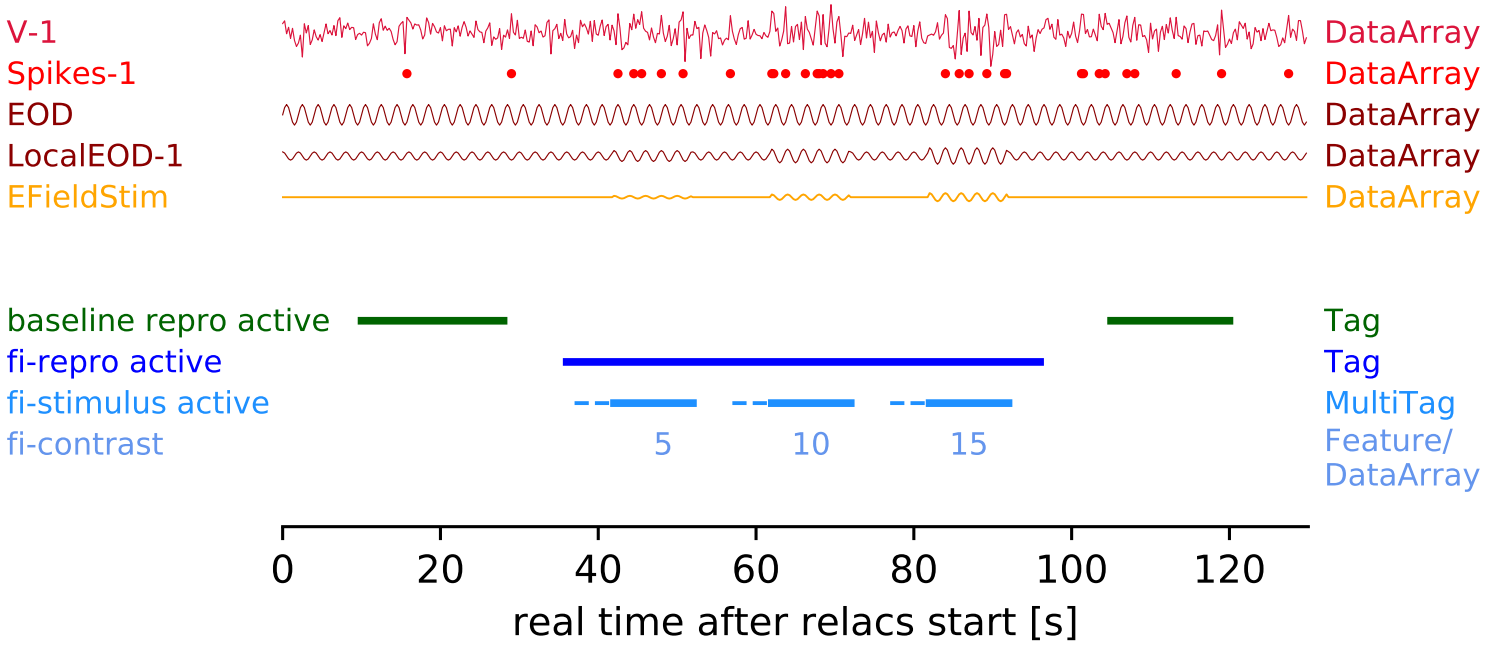
Once we press “Enter”, we may run some macros that in turn run research protocols (repros) with various parameter sets and put out certain stimuli.



Data is not always dumped to file. The “data time” does not necessarily match the real time.

Integration of *NIX* in RELACS

Representing RELACS data in *NIX*



Integration of *NIX* in RELACS

Representing RELACS data in *NIX*

- For each recording (pressing of enter) we create a new dataset.
- In each dataset we find only a single **Block** that contains all data.
- We always have as many **DataArray**s as there are input channels and event channels (from Filters and Detectors).
- For each run of a repro, there will be one **Tag** (usually named according to the repro name with a number suffix).
- For each repro we have a **MultiTag** to store the stimulus-on segments.
- Repro settings go to the metadata into a **Section** that has the same name as the repro **Tag**.
- If the stimulus settings change, a new **MultiTag** will be created.
- Except if the chaning setting is set as “mutable”, then it will become a **Feature** of the respective **MultiTag**.

Working with relacs-flavored nix-files

Working with relacs-flavored *NIX* files

Basic structure of the file

Any *NIX* file has two top-level "folders" blocks, and sections. For data and metadata, respectively. There will be only one **Block** that represents the recording session.

```
import nixio as nix

f = nix.File.open("../2018-05-17-ab.nix", nix.FileMode.ReadOnly)

print("file format: %s \t format version: %s, library version %s"
      % (f.format, f.version, nix.__version__))

print("Blocks:")
for b in f.blocks:
    print("\t%s" % b.name)

print("Metadata sections:")
for s in f.sections:
    print("\t%s" % s.name)

f.close()
```

Working with relacs-flavored *NIX* files

Accessing DataArrays

```
f = nix.File.open("../2018-05-17-ab.nix", nix.FileMode.ReadOnly)
b = f.blocks[0]

eod = b.data_arrays["EOD"]
dim = eod.dimensions[0]

plt.plot(dim.axis(1000), eod[:1000], label=eod.name)
plt.xlabel("%s [%s]" % (dim.label, dim.unit))
plt.ylabel("%s [%s]" % (eod.label, eod.unit))
plt.legend()
plt.show()

f.close()
```

Working with relacs-flavored *NIX* files

Accessing DataArrays

```
b = f.blocks[0]

tag = b.tags["ReceptiveField_0"]
mtag = b.multi_tags["ReceptiveField-1"]

eod_array = tag.references["LocalEOD-2"]
time_dimension = eod_array.dimensions[0]
eod = tag.retrieve_data("LocalEOD-2")[:]
time = time_dimension.axis(len(eod)) + tag.position[0]

ax.plot(time[::10], eod[::10], lw=0.5, label=eod_array.name)
ax.set_xlabel("%s [%s]" % (time_dimension.label, time_dimension.unit))
ax.set_ylabel("%s [%s]" % (eod_array.label, eod_array.unit))

for i, (s, e) in enumerate(zip(mtag.positions[:], mtag.extents[:])):
    x_pos = mtag.retrieve_feature_data(i, "ReceptiveField-1_x_pos")[:]
    ax.plot([s, s+e], [2, 2], lw=2, color="red")
    ax.text(s, 2.25, str(x_pos[0]), fontsize=6, color="red")
```

Working with relacs-flavored *NIX* files

What else?

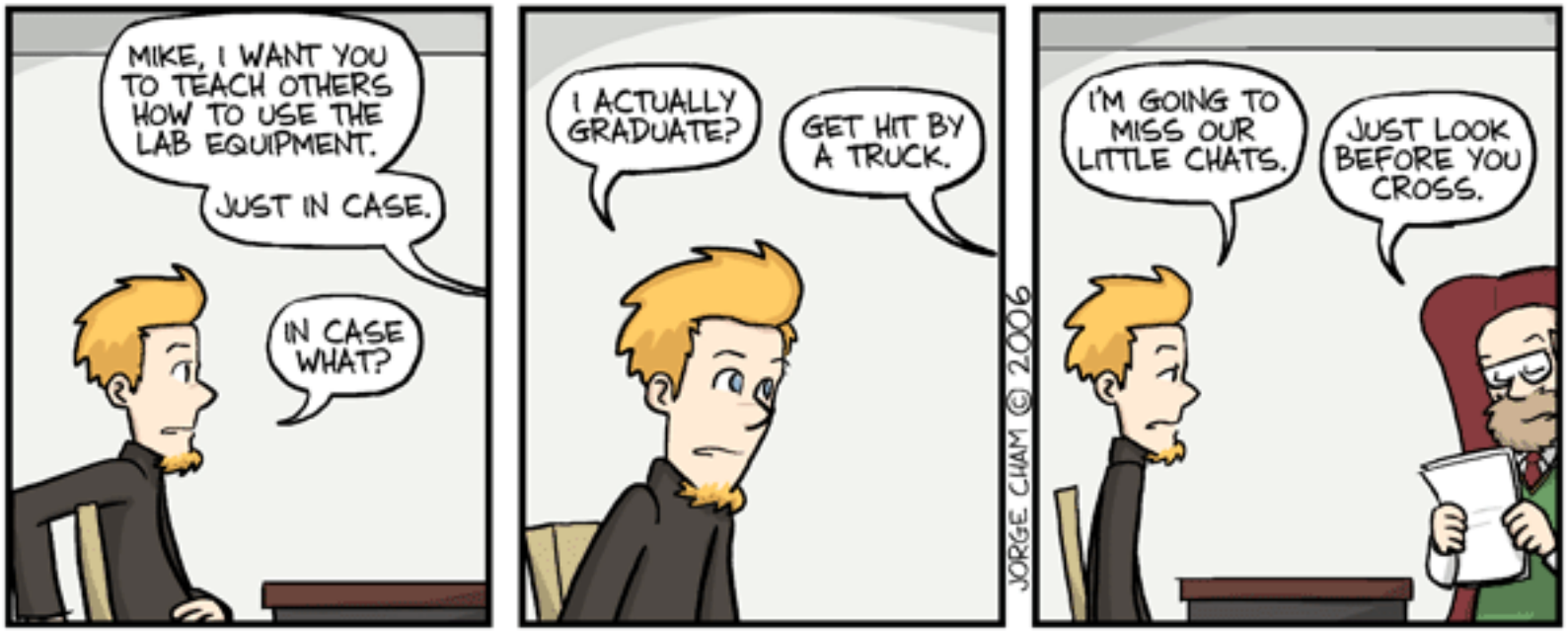
1. With the use of the nix-files we achieve a (more or less) common data storage.
2. Data and metadata reside within the same container.
3. We can hope for a common toolbox for accessing the recorded data.

- More examples required?
- How can we improve the storing of data?
- What functionality should be provided by the nix-libraries?

People

Quite a few people are, or have been, involved in the development:

Jan Benda
Christian Garbers
Jan Grewe
Christian Kellner
Achilleas Koutsou
Balint Morvai
Michael Sonntag
Adrain Stoewer
Andrey Sobolev
Thomas Wachtler
Alexander Ott
Lorand Madai
Fabian Woltermann
@matham
@sudoankit
@Prabhanit
@digaru19
@sOnskar
@saurabhiiit
@sujithvm


MIKE, I WANT YOU TO TEACH OTHERS HOW TO USE THE LAB EQUIPMENT.
JUST IN CASE.
IN CASE WHAT?
I ACTUALLY GRADUATE?
GET HIT BY A TRUCK.
I'M GOING TO MISS OUR LITTLE CHATS.
JUST LOOK BEFORE YOU CROSS.
JORGE CHAM © 2006
WWW.PHDCOMICS.COM