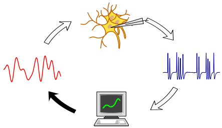
# *RELACS* — a modular software platform for closed-loop experiments

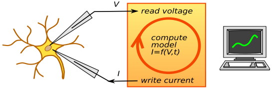
**Jan Benda**

Biozentrum Martinsried
Ludwig-Maximilians Universität München

# Content



## Closed-loop experiments

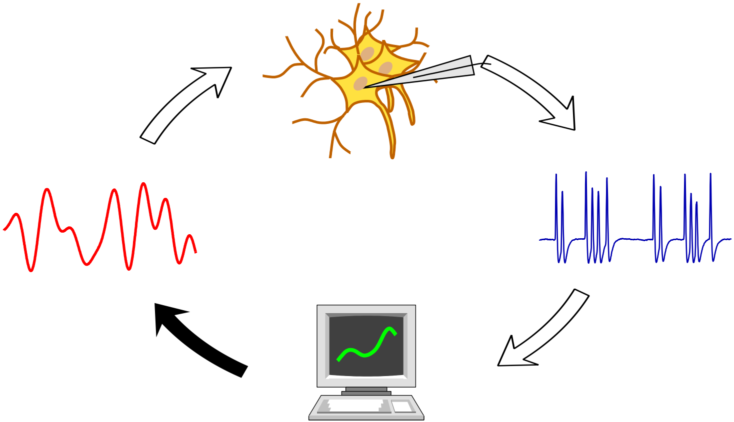



## Dynamic clamp

## *RELACS*

## Metadata

# Closed-loop experiments



# “Traditional” experiments

1. A set of stimuli and a more or less fixed experimental protocol are prepared

# “Traditional” experiments

1. A set of stimuli and a more or less fixed experimental protocol are prepared
2. The recordings are done on a few cells ($>$ one week of experimental work)
   - the raw voltage trace is the only feedback
   - changes during the running experiment often involve manual manipulations
     $\Rightarrow$ precious recording time is wasted

# “Traditional” experiments

1. A set of stimuli and a more or less fixed experimental protocol are prepared
2. The recordings are done on a few cells ($>$ one week of experimental work)
   - the raw voltage trace is the only feedback
   - changes during the running experiment often involve manual manipulations
     $\Rightarrow$ precious recording time is wasted
3. The data are analyzed offline

# “Traditional” experiments

1. A set of stimuli and a more or less fixed experimental protocol are prepared
2. The recordings are done on a few cells ($>$ one week of experimental work)
   - the raw voltage trace is the only feedback
   - changes during the running experiment often involve manual manipulations
     $\Rightarrow$ precious recording time is wasted
3. The data are analyzed offline
4. The stimuli and the protocol are modified

# “Traditional” experiments

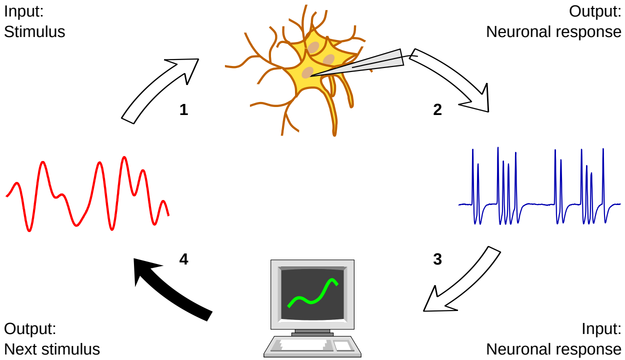
1. A set of stimuli and a more or less fixed experimental protocol are prepared
2. The recordings are done on a few cells ($>$ one week of experimental work)
   - the raw voltage trace is the only feedback
   - changes during the running experiment often involve manual manipulations
     $\Rightarrow$ precious recording time is wasted
3. The data are analyzed offline
4. The stimuli and the protocol are modified
5. A new set of recordings is made

# “Traditional” experiments

1. A set of stimuli and a more or less fixed experimental protocol are prepared
2. The recordings are done on a few cells ($>$ one week of experimental work)
   - the raw voltage trace is the only feedback
   - changes during the running experiment often involve manual manipulations
     $\Rightarrow$ precious recording time is wasted
3. The data are analyzed offline
4. The stimuli and the protocol are modified
5. A new set of recordings is made
6. After several iterations a paper is written

# Closed-loop experiments

1. Present a stimulus
2. Record the response
3. Immediately analyze and visualize the data
4. Generate the next stimulus



# Simple closed-loop experiments

- Online visualization of processed data:
  - General infos, e.g. quality of spike detection, sensitivity of the cell, temperature, condition of animal, ...
  - Specific results, e.g. spike raster, firing rates, spike-triggered averages, ...

$\Rightarrow$ Speeds up manual (“traditional”) closed-loop

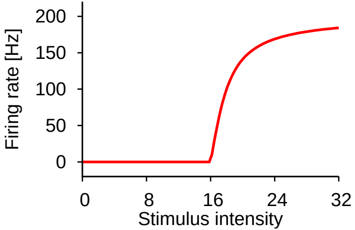
# Simple closed-loop experiments

- Online visualization of processed data:
  - General infos, e.g. quality of spike detection, sensitivity of the cell, temperature, condition of animal, ...
  - Specific results, e.g. spike raster, firing rates, spike-triggered averages, ...

  $\Rightarrow$ Speeds up manual (“traditional”) closed-loop
- Set stimuli relative to the neuron’s dynamic range
- Automatically control motorized electrodes (great for dual unit recordings!)
- Optimize tuning curve measurements
- ...

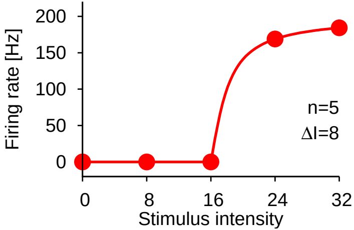
# Example: tuning curve measurement

Traditional:



# Example: tuning curve measurement

Traditional:

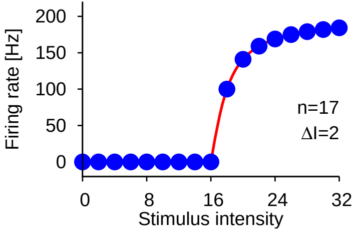



either:

fast $\longrightarrow$ low resolution

# Example: tuning curve measurement

Traditional:




either:

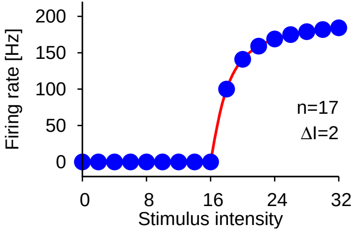
fast $\longrightarrow$ low resolution

or:

high resolution $\longrightarrow$ slow

# Example: tuning curve measurement

Traditional:



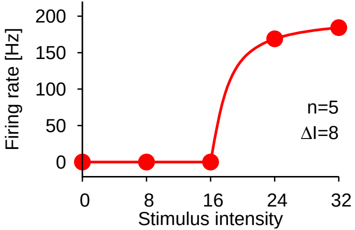

either:

fast $\longrightarrow$ low resolution

or:

high resolution $\longrightarrow$ slow

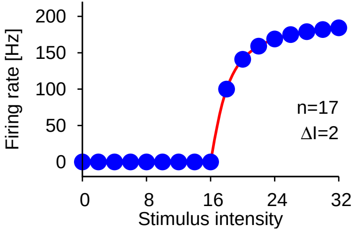
Closed loop:




1. start with low resolution

# Example: tuning curve measurement

Traditional:



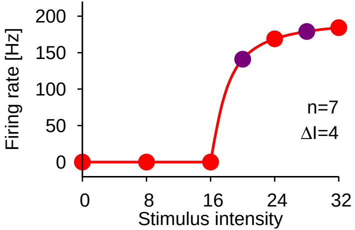

either:

fast → low resolution

or:

high resolution → slow

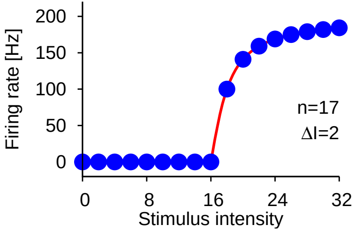
Closed loop:




1. start with low resolution
2. increase resolution where necessary!

# Example: tuning curve measurement

Traditional:



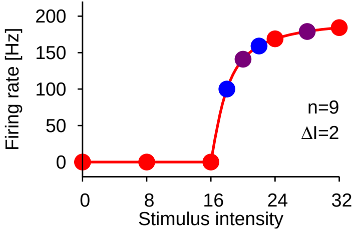

either:

fast ⟶ low resolution

or:

high resolution ⟶ slow

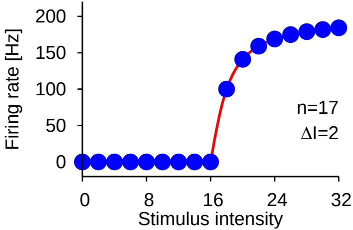
Closed loop:




1. start with low resolution
2. increase resolution where necessary!
3. further increase resolution

# Example: tuning curve measurement

Traditional:



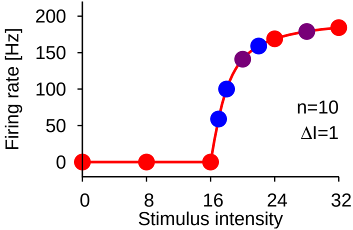

either:

fast → low resolution

or:

high resolution → slow

Closed loop:




1. start with low resolution
2. increase resolution where necessary!
3. further increase resolution

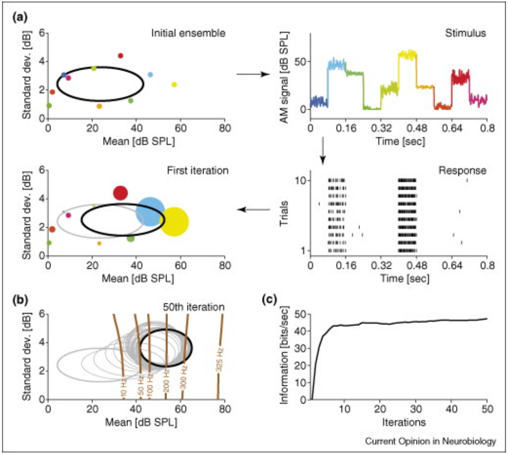
# Advanced closed-loop experiments

New experimental designs are possible:

- Optimal search for a neuron's receptive field.
- Search for stimuli that drive a neuron in an "optimal" way.
- Find set's of stimulus parameter that result in the same response (iso-response method).
- ...

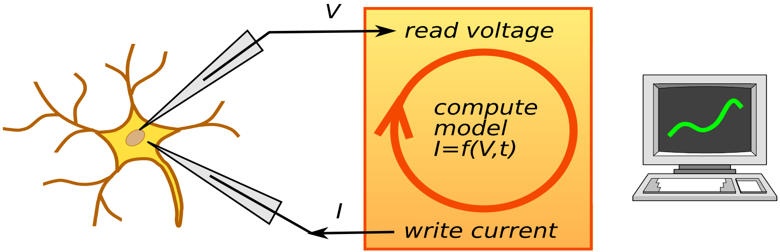
Benda et al. (2007): "From response to stimulus: adaptive sampling in sensory physiology." *Curr. Opin. Neurobiol.* **17**: 430–436.

# Example: optimal stimulus ensembles



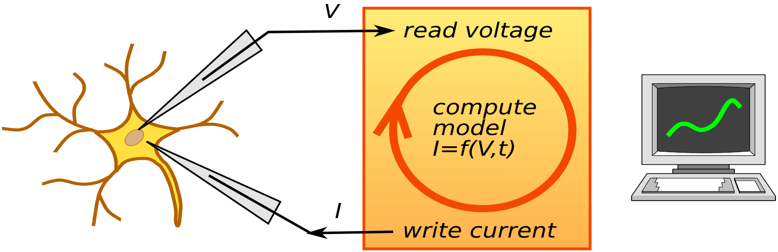

Machens et al. (2005) *Neuron* **17**: 47–56.

# Dynamic clamp



# Dynamic clamp

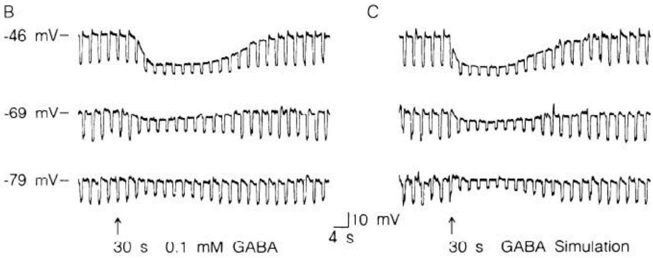
Current-clamp, with the current $I$ computed as a function of the measured membrane potential $V$.




Closed-loop at a per sample time scale (tens of kHz).

# Artificial conductances

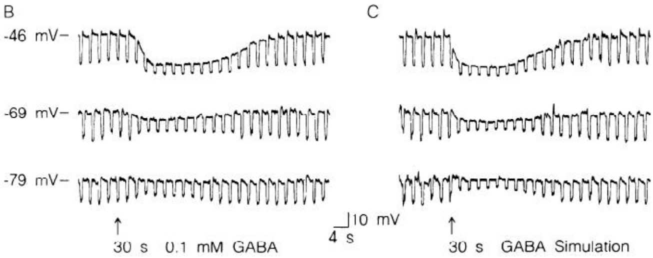
$$I = g(t) \cdot (V - E)$$




Andrew A. Sharp, Michael B. ONeil, L. F. Abbott, & Eve Marder (1993) *J Neurophysiol*

# Artificial conductances

$$I = g(t) \cdot (V - E)$$

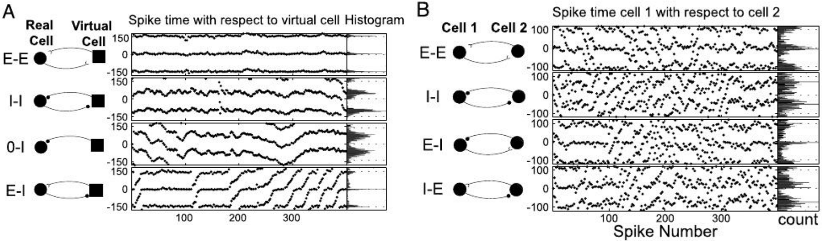



Andrew A. Sharp, Michael B. ONeil, L. F. Abbott, & Eve Marder (1993) *J Neurophysiol*

- Synaptic conductances
- Voltage-gated conductances

# Artificial networks

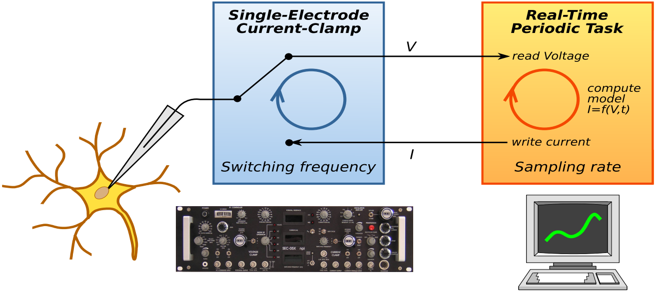
$$I_1 = g_{syn}(V_2) \cdot (V_1 - E) \qquad I_2 = g_{syn}(V_1) \cdot (V_2 - E)$$




Theoden I. Netoff, Matthew I. Banks, Alan D. Dorval, Corey D. Acker, Julie S. Haas, Nancy Kopell, & John A. White (2005) *J Neurophysiol*

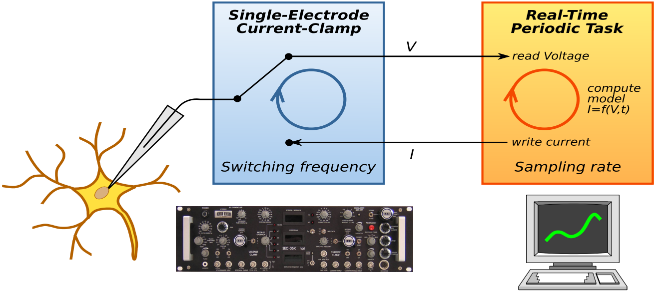
- Artificially couple real neurons
- Couple with simulated neurons

# Discontinuous CC and dynamic clamp




- Sampling rate $\leq$ switching frequency$/2$

# Discontinuous CC and dynamic clamp



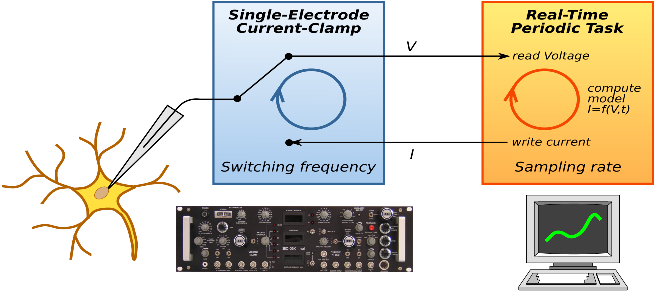

- Sampling rate $\leq$ switching frequency$/2$

$\Rightarrow$ Synchronize switching and dynamic clamp cycles!

Sampling rate $=$ switching frequency

(in collaboration with R. Polder, npi electronic)

# Discontinuous CC and dynamic clamp




- Sampling rate $\leq$ switching frequency$/2$

$\Rightarrow$ Synchronize switching and dynamic clamp cycles!
Sampling rate $=$ switching frequency
(in collaboration with R. Polder, npi electronic)

- NPI SEC: variable switching frequency $\gg$ 10 kHz independent of C compensation

# relacs

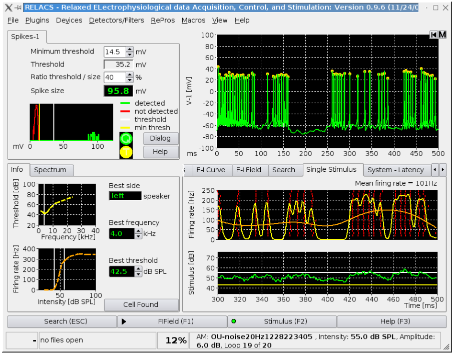
... enjoy your recordings

www.relacs.net

# *RELACS* ... enjoy your recordings

Relaxed Electrophysiological data Acquisition, Control, and Stimulation

*RELACS* is a framework for closed-loop experiments

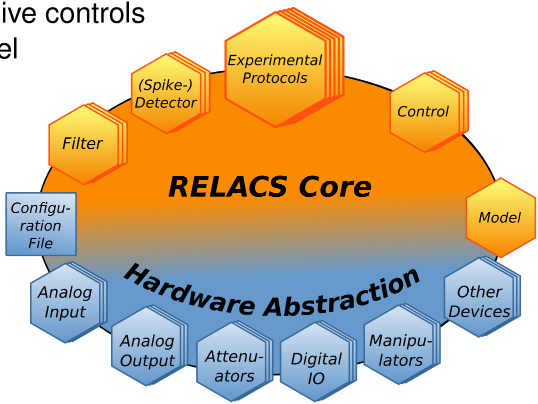



⇒ currently 13 scientific publications based on *RELACS* data in *Neuron, J Neurosci, PLoS Biol, Nat Neurosci, J Neurophysiol, etc.*

# Modular design

*RELACS* core with flexible C++ Plugins for

- hardware abstraction
- data pre-processsing (filter, spike detectors)
- **experimental protocols**
- passive controls
- model

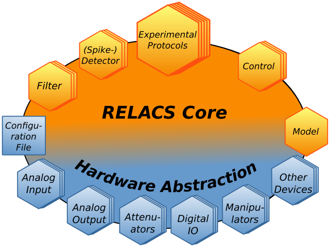


# Hardware independent protocols

*RELACS* integrates all hardware components.

**Experimental protocols** for *RELACS*

- are implemented independently of specific hardware
- can be used on all the different experimental setups in your lab without any modifications
- can be shared with other labs



# Free and open source software

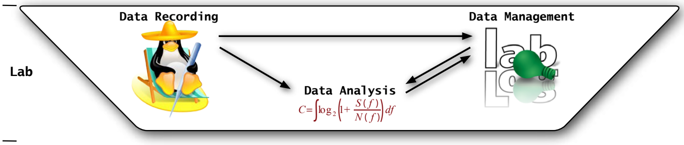
*RELACS* is open source and free software distributed under the GNU General Public License (GPL).

- No hassle with licenses of commercial software.
- Add whatever new feature you need directly to the program.
- Share the program and your specific experimental protocols with your collaborators.
- Know what the data-analysis algorithms are doing!

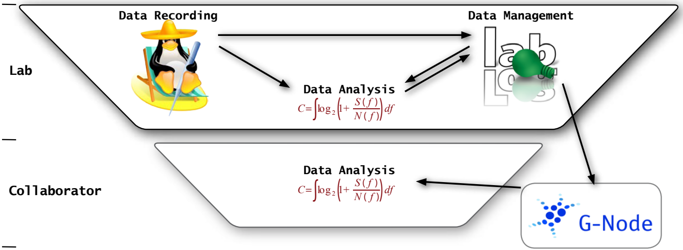
# Talking about data

## an extensible framework
## for metadata exchange

# The data-chain



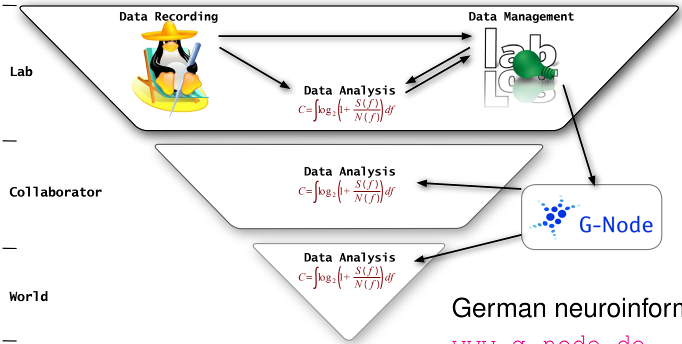
# The data-chain




German neuroinformatics node

www.g-node.de

# The data-chain




German neuroinformatics node

www.g-node.de

- All data transfer requires talking about data.
- How to exchange metadata?

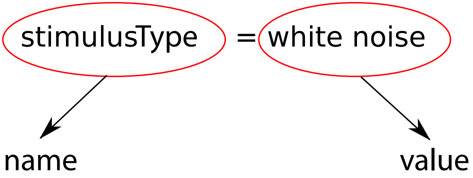
# Metadata

- is “data about data”.
- describe recording conditions.
- essential for data analysis, management, and sharing.

stimulusType = white noise

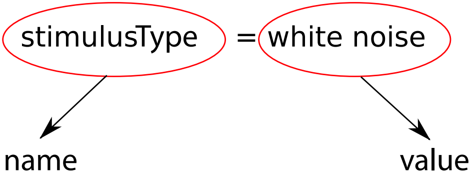
# Metadata

- is “data about data”.
- describe recording conditions.
- essential for data analysis, management, and sharing.



# The metadata problem

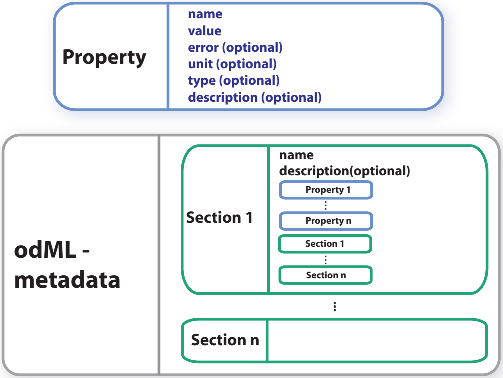
- is “data about data”.
- describe recording conditions.
- essential for data analysis, management, and sharing.




- What name to choose?
- What does it mean?
- How to organize metadata?

# odML — open metadata markup language

## Structure:




Implemented as the odML XML Schema

# odML — open metadata markup language

## Vocabularies: names & definitions

**HardwareSettings:**

Amplifier:

| name | type | description |
|---|---|---|
| Gain | float | The amplifier gain. |
| HighpassCutoff | float | The cutoff frequency of the amplifier's highpass filter. Given in Hz. |
| LowpassCutoff | float | The cutoff frequency of the amplifier's lowpass filter. Given in Hz. |
| Mode | string | The amplifier mode. E.g. Bridge, CC, VC etc. |

# How to use odML?

1. Assemble properties:

    - If you find an appropriate property in the odML-vocabularies, use it!
    - Ignore all properties that do not match.
    - Add your own properties that are not yet in the vocabulary, if possible with a description.

# How to use odML?

1. Assemble properties:
   - If you find an appropriate property in the odML-vocabularies, use it!
   - Ignore all properties that do not match.
   - Add your own properties that are not yet in the vocabulary, if possible with a description.
2. Write them into an odML XML file

# How to use odML?

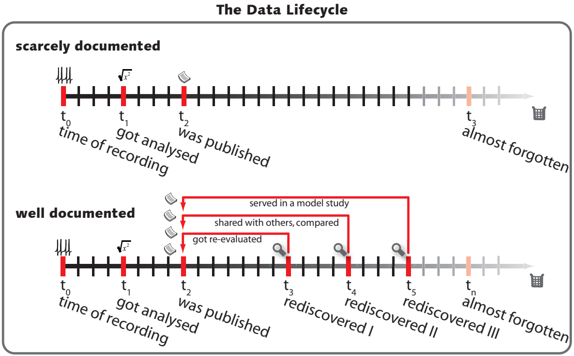
1. Assemble properties:

    - If you find an appropriate property in the odML-vocabularies, use it!
    - Ignore all properties that do not match.
    - Add your own properties that are not yet in the vocabulary, if possible with a description.

2. Write them into an odML XML file

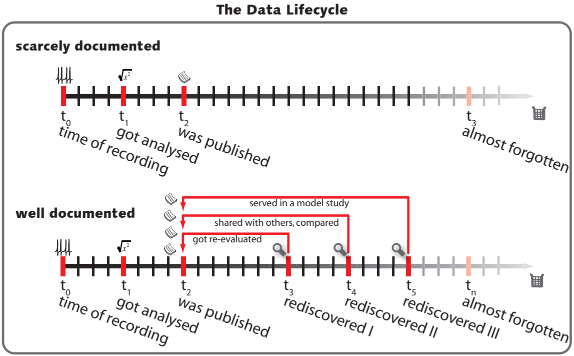
3. Transfer them to an analysis or database program

# The data life-cycle




- Meta information tends to vanish with time.
- Thus, re-using of old data is a tedious business.
- Data should be annotated as early as possible (preferentially at the time of acquisition, e.g. with *RELACS*).

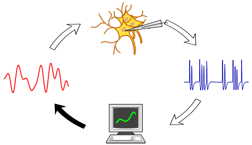
# The data life-cycle

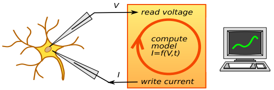



- odML provides a simple and flexible standard
- Well annotated data can be found and reused easily

$\Rightarrow$ Your data deserves it!

# Summary






## Closed-loop experiments

Novel experimental designs

## Dynamic clamp

Artificial conductances and hybrid networks

— R. Polder, npi electronic, Tamm

## *RELACS* www.relacs.net

Software platform for closed-loop and dynamic clamp experiments

— NeurOnline: S. Rotter, M. Ambard, A. Brandt, C. Boucsein, Freiburg

## Metadata — odML

A standard for sharing data

— J. Grewe & G-node, LMU Munich

# Experimental protocol example

```
int Example::main( void ) {
  // some initialization ...
  OutData signal;
  signal.setTrace( "LeftSpeaker" );
  signal.sineWave( frequency, duration, amplitude );
  SampleDataD rate( 0.0, duration, 0.001 );
  for ( int counter=0; counter<Repeats; counter++ ) {
    write( signal );
    sleep( duration + pause );
    EventData spikes( events( SpikeEvents[0] ),
                      events( SpikeEvents[0] ).signalTime(),
                      events( SpikeEvents[0] ).signalTime() + duration );
    double meanrate = spikes.rate( 0.3*duration, duration );
    spikes.addRate( rate, counter, GaussKernel( sigma ) );
    P.clear();
    P.plot( rate, 1000.0, Plot::Yellow, 2, Plot::Solid );
    P.draw();
    if ( meanrate < targetrate ) {
      amplitude *= 2.0;
      signal.sineWave( frequency, duration, amplitude );
    }
  }
}
```

# *RELACS* C++ library for data analysis

Data structures (classes, container):

- *Array* — Basic 1-D vector
- *SampleData* — 1-D data vector with regularly sampled time axis
- *Map* — Sequence of $x|y$ data pairs

Algorithms:

- basic statistics (moments, quartiles, histogram)
- power spectra, coherence, transfer function
- linear fits
- non-linear fits (Simplex, Levenberg-Marquardt)

# *RELACS* C++ library for data analysis

Data structures (classes, container):

- *EventData* — Spikes and other point process data
- *EventList* — Multi-trial spike trains

Algorithms:

- firing rates (mean, PSTH binned/kernel, 1/ISI)
- CV, Fano factor, ISI correlation
- vector strength, reliability, jitter
- mutual information (lower and upper bound)

# Current odML developments

Done:

- Schema definition converges to version 1.
- Java and MatLab library to read, write and manipulate odML-files.
- Editor for odML metadata.

# Current odML developments

Done:

- Schema definition converges to version 1.
- Java and MatLab library to read, write and manipulate odML-files.
- Editor for odML metadata.

On the way:

- Definition of the vocabularies.

Planned:

- Libraries for C/C++, Python ...
- collaboration with various initiatives (CRCNS, NIF, etc. )

# odML — Details

## odML-Property:

| element name | description | occurrence |
|---|---|---|
| name | The name of this property. | 1 |
| value | The value of this property. | 1 - ∞ |
| error | An error estimate of the value(s). | 0 - ∞ |
| unit | The unit of the value(s). | 0 - 1 |
| type | The datatype of the value e.g. int, float, string, date, time, binary, etc. | 1 |
| id | An identifier for each value, e.g. for a database. | 0 - ∞ |
| nameDefinition | Defines the property. | 0 - 1 |
| valueDefinition | Defines each individual value. | 0 - ∞ |
| parent | This property is only meaningful if the parent property exists. | 0 - 1 |
| parentValue | This property is only meaningful for a specific parent value. | 0 - 1 |

# odML — Details

**odML-Section:**

| element name | description | occurrence |
|---|---|---|
| name | The name of this section. | 1 |
| alias | An alias name for this section. | 0 - 1 |
| id | An identifier e.g. from a database by which the information of this section can be found. | 0 - 1 |
| definition | Defines the section. | 0 - 1 |
| vocabulary | The URI of the vocabulary which defines this section. | 0 - 1 |
| parent | This section might be meaningful only if it is child of a parent section. | 0 - 1 |
| parentURI | The URI of the parent section's definition. | 0 - 1 |
| odML-Property | A section can contain properties ... | 0 - ∞ |
| odML-Section | ... and subsections. | 0 - ∞ |