# **mozaik**: integrative work-flow framework for large-scale spiking network simulations

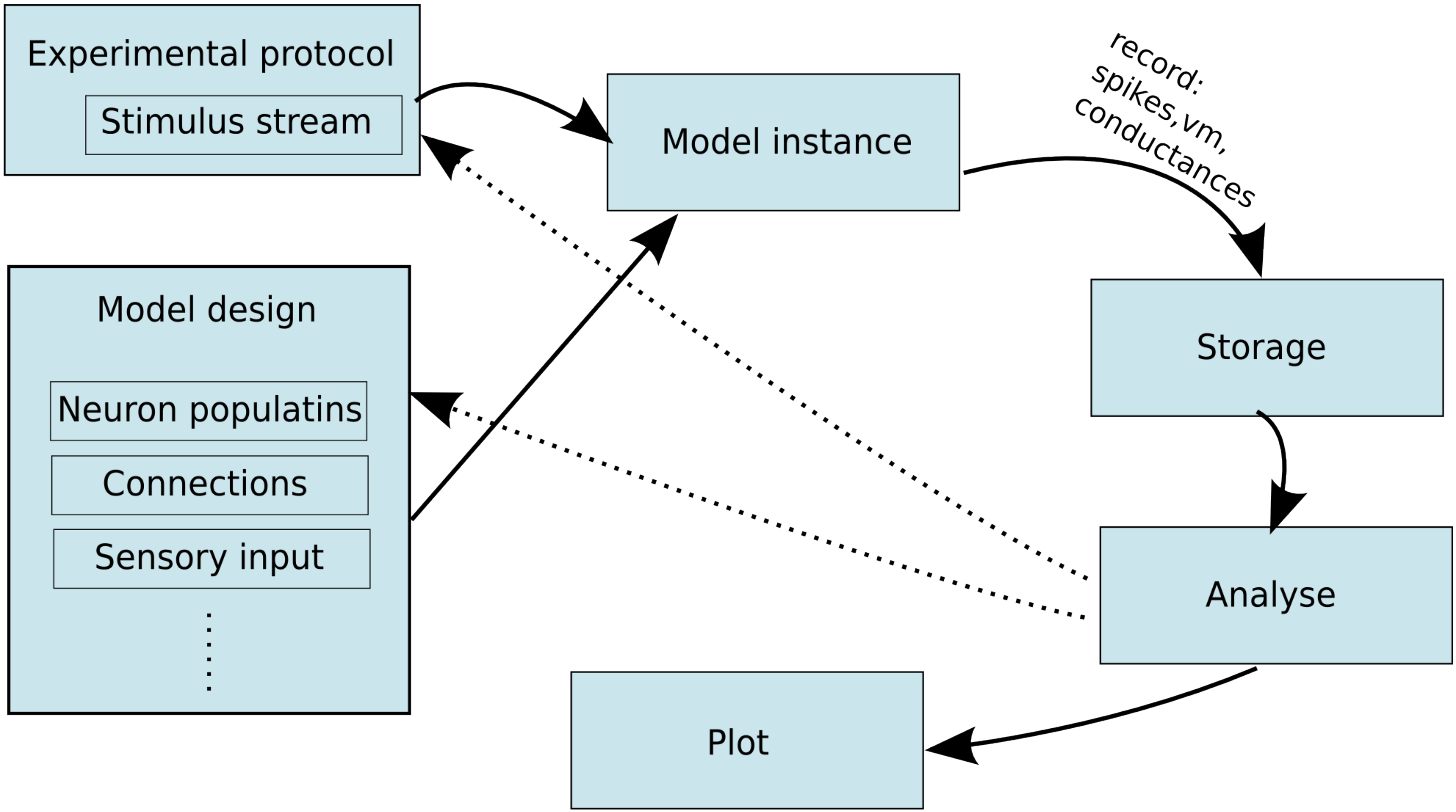
Jan Antolik, Andrew Davison, Yves Frégnac

Unité de Neurosciences Information et Complexité, CNRS, Gif-sur-Yvette, France

# Motivation

- Most current spiking network simulations:
  - Small to medium scale
  - Simple homogeneous architectures
  - Focus on statistical rather then functional properties
  - Simple experimental and stimulation protocols
- Recent shift to:
  - Large-scale
  - Heterogeneous architectures reflecting anatomy
  - Investigation of functional properties
  - Involved experimental and stimulation protocols reflecting biological experiments

# Typical work-flow cycle



# Motivation

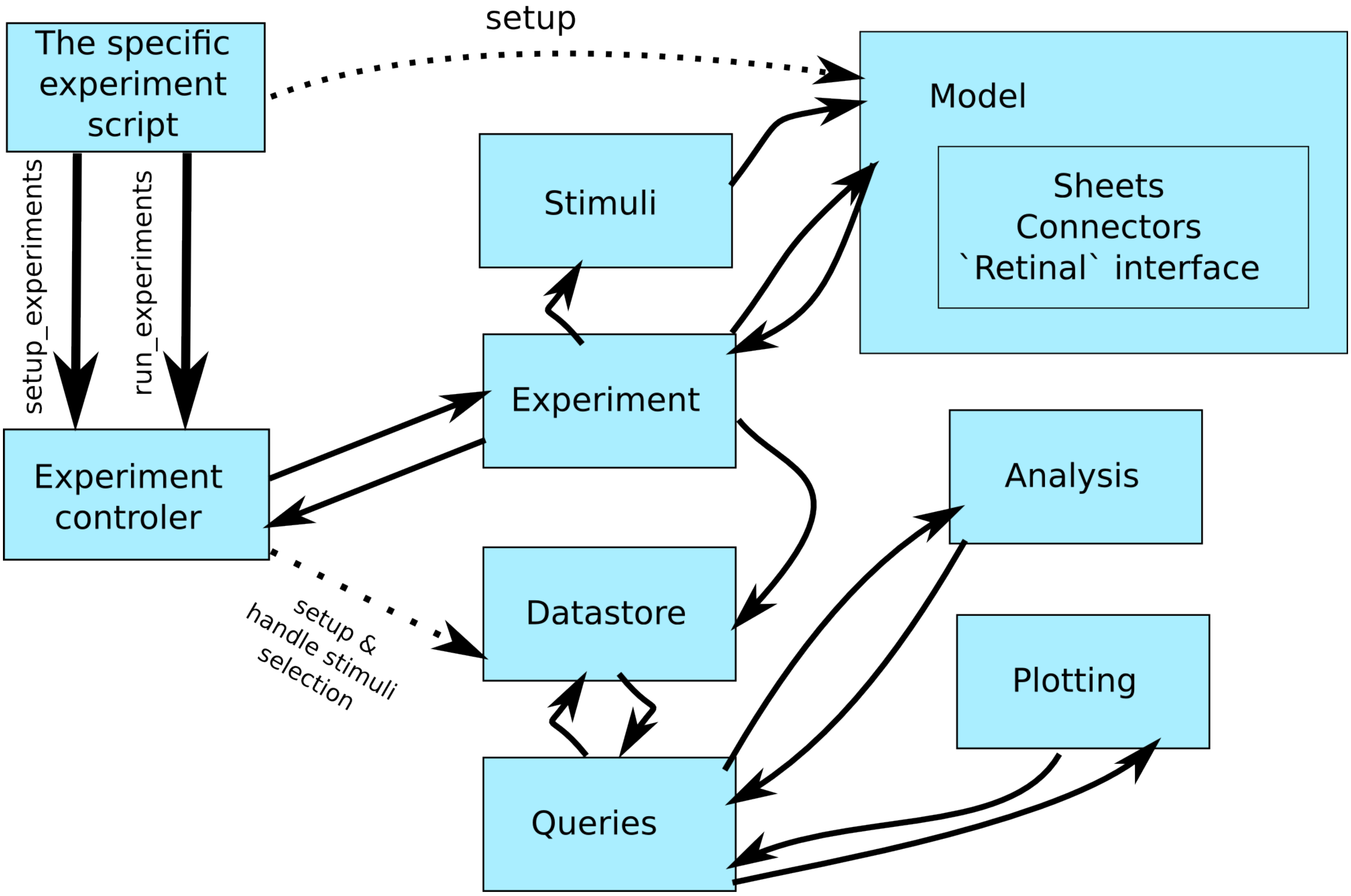
- Number of tools doing parts of the work-flow (nest, neuron, pyNN, neo, NeuroTools, ...)
- Some areas not covered
- No unifying framework connecting all these tools into single work-flow

# mozaik

- Python set of modules offering high-level integration of the work-flow
- Integrates
  - Model design
  - Experimental protocol design
  - Stimulus handling
  - Data storage and high-level manipulation
  - Analysis
  - Plotting

# **mozaik** design

- Built on top of existing tools
  - simulator ↔ pyNN (NEST*)
  - storage ↔ neo
  - stimuli ↔ topographica *
  - analysis ↔ neurotools *
  - plotting ↔ matplotlib
- For now focused on cortex and visual modality
- Written in modular way
- All pure python


The specific experiment script
setup
Model
Sheets
Connectors
`Retinal` interface
Stimuli
setup_experiments
run_experiments
Experiment
Experiment controler
Analysis
Datastore
setup & handle stimuli selection
Plotting
Queries

# Model design

- Fully parametrized via *neurotools ParameterSet*
- Model is composed of
  - 2D sheets of neurons (layers)
    - various distributions of neurons in space
    - handles cortical magnification
  - Projections between sheets
    - supports all the basic types in pyNN
    - adds gabor receptive fields
    - push-pull connectivity
    - afferent input parametrized with maps

# Model design

- Sensory input sheet
  - currently retina supported
  - handling of visual space
- Model handles retrieval of data from simulator (via pyNN)
- Components add annotations to datastore such as neural positions, initialized orientation preferences etc.

# Stimuli

- System for identifying stimuli and their parameters
- Used through the **mozaik** to track the origin of any recordings
- Currently very simple high level abstract API
- Makes it trivial to re-use stimuli from topographica.
-

# Experiment control

- Currently very simple principle
    - Lot of placeholders for potentially more involved logic
- User defines order of experiments
- Each experiment
    - Defines a stream of stimuli
    - Any analysis to be performed on the recorded data
- The experimental controller:
    - Goes experiment by experiment
    - Presents the list of stimuli to the model
    - Shows blank stimuli for x time between each stimuli
    - Executes analysis
    - Stores both recording and analysis results into datastore

# Storage

- Storage module aggregates all the info
  - Recordings (spikes, vm, conductances)
  - Neuron positions
  - Stimuli
  - Additional data annotations
- Stores them in neo format
  - pickled format
  - hdf5 (not tested)
- Allows for doing views into the datastore

# Queries

- Queries allow filtering data in datastore
- (datastore/datastore view) → datastore view
- This allows for powerful customization of analysis and plotting tools
- Available query examples:
  - constrain data to single sheet
  - constrain data to certain type of stimuli
  - splice data based on stimulus parameter value
- Contains both function level and ParameterSet interface

# Analysis

- Trivial high level interface:
  - Input is datastore view
  - Each analysis should know how to filter out the most general data it is applicable to (using queries)
  - As a results it creates AnalysisDataStructure and returns it
  - This gets stored back into Datastore
- This way analysis should be highly customizable just by pre-filtering the data it is passed

# Plotting

- Built on top of matplotib
- Similar philosophy as analysis
- Input is a datastore/view
- Given plotting algorithm filters out the most general set of data it can handle and plots it
- Parametrized via ParameterSet
- High-level plotting API that allows for hierarchical definition of figures
- Uses the prior knowledge of the **mozaik** primitives to automatize lot of plotting code

# Limitations

- Very early pre-alpha stage
- Non-interactive
- Currently all data in memory
- Only visual cortex scope
- Documentation
- No tests

# Future work

- Random data access on the hard-drive
- Higher-level model components (cortical layers, cortical areas, higher-level connectors)
- Project tracking (Sumatra)
- Automatic checking of experiment order sanity
- Support of FACETS like benchmarks within the Experimental control framework
- GUI for browsing and plotting of data in datastore
- You tell me!!!

https://github.com/antolikjan/mozaik