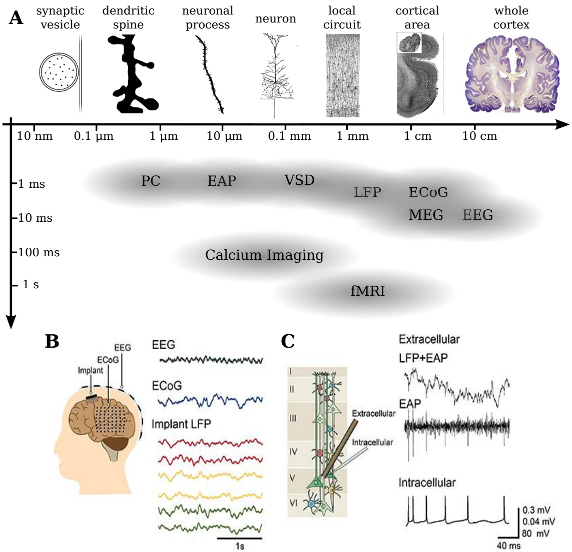
PART 1: INTRODUCTION TO NEST

Introduction to the simulation of structurally detailed large-scale neuronal networks

13 July 2019 | Alexander van Meegen, Dennis Terhorst | INM-6, IAS-6, INM-10; Jülich Research Centre

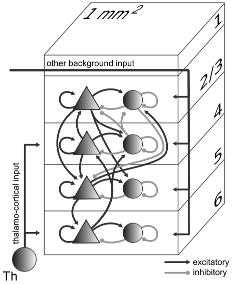
Multi-scale brain structure and dynamics



[Dahmen, 2017]

The microcircuit model

- 10^5 identical leaky-integrate and fire neurons
- $3 \cdot 10^8$ exponentially decaying synaptic currents
- four layers with one excitatory and one inhibitory population each
- size of populations and connection probabilities deduced from anatomical data sets

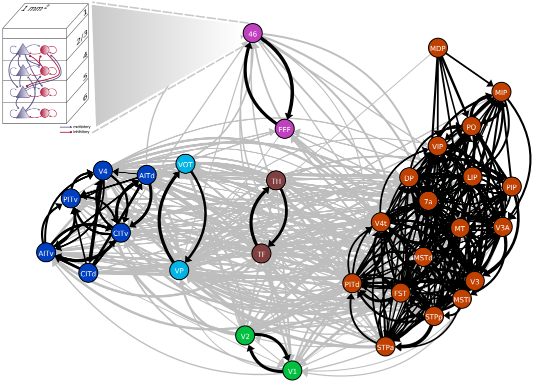


- asynchronous irregular and cell-type specific firing rates
- thalamic stimulation elicits flow of activity through cortical layers

Potjans and Diesmann (2014) The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. Cerebral Cortex 24(3):785-806

The multi-area model

- full-density model of macaque visual cortex
- axonal tracing data from the CoCoMac database, which are systematically refined using dynamical constraints
- stable asynchronous irregular ground state



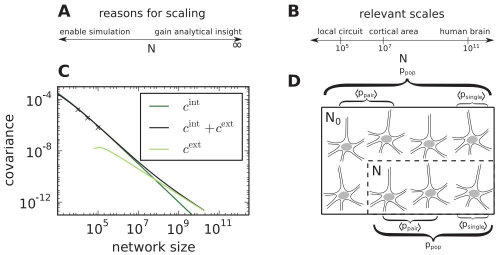
- produces realistic spiking statistics in V1
- functional connectivity compares to fMRI measurements

Schmidt et al. (2018) Multi-scale account of the network structure of macaque visual cortex. Brain Structure and Function 223(3):1409-1435
Schmidt et al. (2018) A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. PLOS CB 14(10):e1006359

JÜLICH Forschungszentrum

Importance of the correct network size

- under which conditions can a small network represent a subsampled larger network?
- analyzes scalability of binary and LIF neuron networks



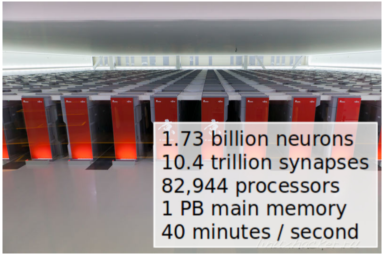
- mean activity can be preserved by adjusting the mean and variance of the input
- temporal structure of pairwise averaged correlations depends on the effective connectivity and cannot always be preserved

van Albada et al. (2015) Scalability of Asynchronous Networks Is Limited by One-to-One Mapping between Effective Connectivity and Correlations. PLOS CB 11(9):e1004490

NEST = NEural Simulation Tool

- Focus on the dynamics, size and structure of neural systems rather than on the exact morphology of individual neurons
- NEST runs on laptops (Linux, Mac OS X ($\geq$10.3), Windows via virtualization) as well as supercomputers $\rightarrow$ simulations of large-scale models
- NEST is a hybrid parallel (OpenMP+MPI) simulator for spiking neural networks, written in C++ with a Python front end

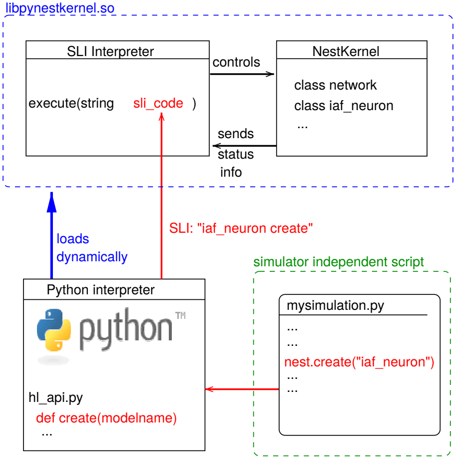
- Get publication and source code on http://nest-simulator.org



JÜLICH Forschungszentrum

Programming languages

- C++ kernel
- Built-in simulation language interpreter (SLI)
- Python-based user interface (PyNEST)

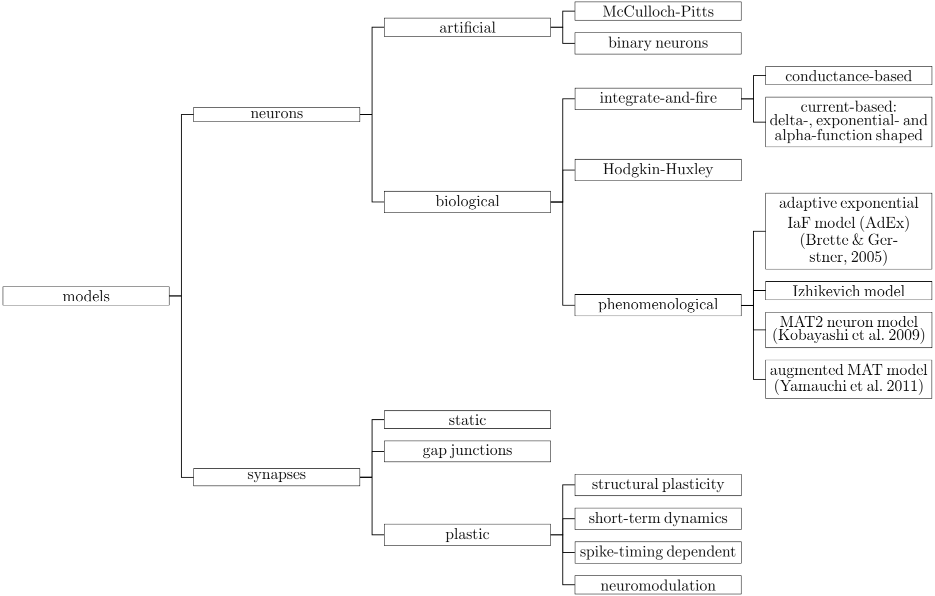


- Back end for the simulator-independent modeling tool PyNN
- Interface to the Multi Simulator Coordinator MUSIC

Three main components of a NEST simulation

- **Nodes**
  - Neurons – Devices (– Sub-networks)
  - Have dynamic state variable(s) that changes over time ($V_{\mathrm{m}}(t)$)
  - Can be affected by events (spikes)
- **Events**
  - Pieces of information of a particular type (e.g., spike, voltage or current event)
  - Recording devices: `‘spike_detector’`, `‘voltmeter’`, `‘multimeter’`
- **Connections**
  - Communication channels for the exchange of events
  - Directed (from source node to target node)
  - Weighted (how strongly does an event influence the target node)
  - Delayed (length of transmission duration between source and target)
  - Connections are created using one global `Connect` function

NEST neuron and synapse models



Event-driven vs. time-driven simulation

	Event-driven	Time-driven
Pros	■ more efficient for low input rates ■ 'correct' solution for invertible neuron models	■ more efficient for high input rates ■ works for all neuron models ■ scales well
Cons	■ only works for neurons with invertible dynamics ■ event queue does not scale well	■ only 'approximate' solution even for analytically solvable models ■ spikes can be missed due to discrete sampling of membrane potential

Event-driven vs. time-driven simulation

NEST: hybrid approach to simulation

- input events to neurons are frequent: time-driven algorithm
  - If the dynamics is nonlinear, we need a numerical method to solve it, e.g.:
    - Forward Euler: $y([i+1]h) = y(ih) + h \cdot \dot{y}(ih)$
    - Runge-Kutta (kth order)
    - Runge-Kutte-Fehlberg with adaptive step size
    - ...

  $\rightarrow$ Use a pre-implemented solver, for example, from the GNU Scientific Library (GSL).

  - If the dynamics is linear (e.g. LIF or MAT), we can solve it exactly.
- events at synapses are rare: event driven component
  - Exception: gap junctions

Exact integration of linear time-invariant systems

- consider time-invariant linear system $\dot{y} = Ay + x$

→ exact solution: $y(t) = e^{A(t-s)}y(s) + \int_{s_+}^{t} e^{A(t-\tau)}x(\tau)\,d\tau$

- time grid: $\{t_k = k \cdot h \,|\, k = 1, 2, \ldots\}$; spike train: $x(t) = \sum_k x_k \delta(t - t_k)$

→ general solution: $y_{k+1} = e^{Ah}y_k + x_{k+1}$

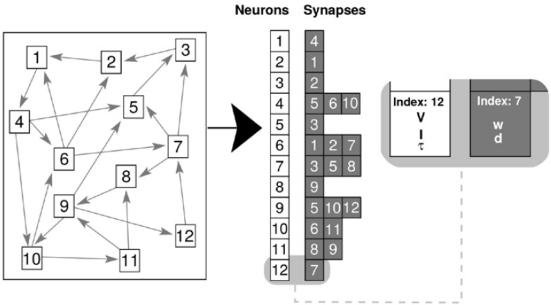
- iterative propagation of solution

$$
\begin{array}{ccccccccc}
 & & x_1 & & x_2 & & x_3 & & \cdots \\
 & & \downarrow & & \downarrow & & \downarrow & & \\
y_0 & \rightarrow & y_1 & \rightarrow & y_2 & \rightarrow & y_3 & \rightarrow & \cdots
\end{array}
$$

with propagator (matrix) $P(h) = e^{Ah}$

Rotter and Diesmann (1999) Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. Biological Cybernetics 81(5-6):381-402

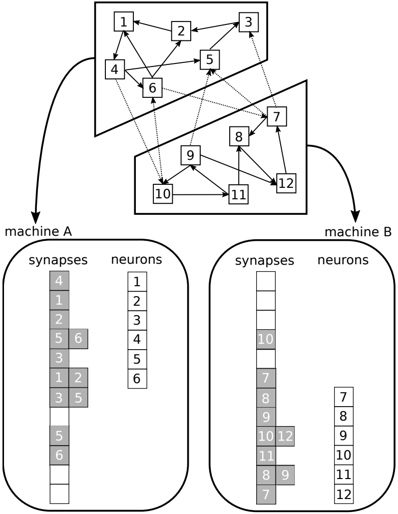
Representation of network structure: serial



- each neuron and synapse maintains its own parameters
- synapses save the index of the target neuron

Representation of network structure: distributed

- modulo operation distributes neurons
- one target list for every neuron on each machine
- synapse stored on machine that hosts the target neuron
- connections are established on each machine and the connectivity information subsequently propagated to other machines
  - → wiring is a parallelizable task

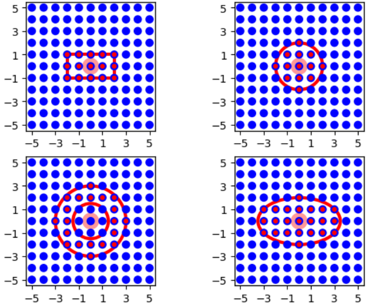


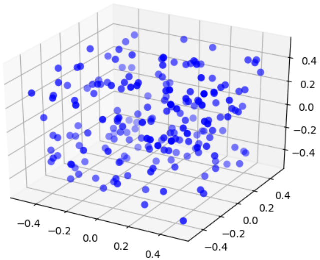
Creating custom models

- Discuss with developers via user mailing list
  - if your idea makes sense
  - if it has not yet been implemented
- Only create models for NEST versions > 2.10
  - Start from most similar existing model
  - It may end up in a release!
- **NESTML (NEST Modeling Language)**
  - New simplified language with syntax similar to Python (recommended)
  - `https://github.com/nest/nestml`
  - Currently enables developing neuron but not yet synapse models
- Extension modules (C++)
  - loaded dynamically
  - `http://nest.github.io/nest-simulator/extension_modules`
- Inside NEST (C++)

Topology

- Functionality
  - Lay out elements on grids or at arbitrary points in space (2D or 3D)
  - Elements can be neurons or combinations of neurons and devices
  - Connect neurons in a position- and distance-dependent manner
  - Set periodic boundary conditions
  - Choose whether to allow self-connections (autapses) or multiple connections (multapses)
  - Distance-dependent or random weights and delays
- User manual:
  `https://www.nest-simulator.org/documentation`
  $\rightarrow$ Topology





Why should I use NEST?

1. NEST provides over 50 neuron models
2. NEST provides over 10 synapse models, including short-term plasticity (Tsodyks & Markram) and different variants of STDP
3. NEST provides many examples that help you getting started
4. NEST lets you inspect and modify the state of each neuron and each connection at any time during a simulation
5. NEST is fast and memory efficient; it makes best use of your multi-core computer and compute clusters
6. NEST has a large and experienced developer community
7. NEST was first released in 1994 under the name SYNOD and has been extended and improved ever since
8. NEST is open source software and is licensed under the GPL 2

Anaconda, Miniconda, ...

User space package management system

- \+ isolates installations of different packages
- \+ includes system libraries (in contrast language specific managers like pip)
- \+ can export environment definition
- – interactions with base environment / system & language package managers
- – requires basic understanding of bash environment ($PATH, $PYTHONPATH, $LD_LIBRARY_PATH, ...)
- – non-trivial dependency management

Installation

1. Download `https://docs.conda.io/en/latest/miniconda.html`, (probably Miniconda3, 64-bit)
2. open new terminal window

Installation (conda)

```
conda env create --file environment.yml
```

with environment.yml defining the package environment

```
name: CNS2019_NEST
channels:
  - conda-forge
  - defaults
dependencies:
  - jupyter
  - cython
  - rise
  - numpy
  - nest-simulator=2.16.0
  - datrie
  - python=3.6
  - pygments
  - pip
  - pip:
    - docopt
    - snakemake
```

Installation (apt, experimental)

Add a personal package archive to find NEST package:

```
sudo add-apt-repository ppa:nest-simulator/nest
sudo apt-get update
sudo apt-get install nest
```

After the installation:

```
source /usr/bin/nest_vars.sh
```

or add this line to your .bashrc.
NEST was built with '-Dwith-python=3' and is installed to /usr! Python 2 will not know anything.

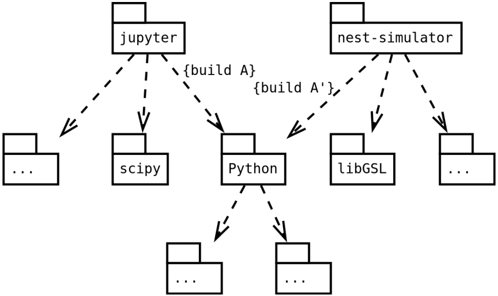
Now hands-on

1. **Install NEST**
   - Instructions: `https://www.nest-simulator.org/documentation/`
   - On Windows use a virtual machine (USB sticks with image available)
   - On Ubuntu you can also use a PPA: `ppa:nest-simulator/nest`
2. **Get the exercises**
   - Go to `https://github.com/alexvanmeegen/CNS2019_NEST_Tutorial`
   - Download as zip (or clone)
3. **Enjoy the ride**
   - Open `0_hello_world.ipynb` for a first glance
   - Get going with `1_first_steps.ipynb`

Background

Conda Dependencies...

```
conda create -n nest-tutorial
conda activate nest-tutorial
conda install jupyter
conda install -c conda-forge nest-simulator
...
```



$\Rightarrow$ replacing Python makes Jupyther loose it's kernel!

JÜLICH Forschungszentrum