THE NEURAL SIMULATION TOOL NEST

2nd HPAC Platform Training

November 25, 2019 | Jochen M. Eppler (j.eppler@fz-juelich.de) | SimLab Neuroscience

OUTLINE

Introduction

Neuronal simulations

Technological background

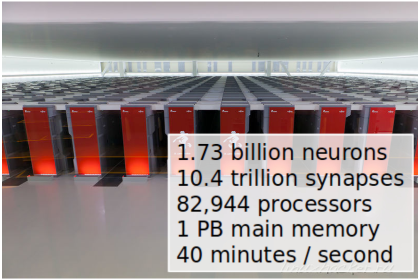
Developing new models

Performance

NEST = NEURAL SIMULATION TOOL

- Point neurons and neurons with few electrical compartments
- Phenomenological synapse models (STDP, STP)
  + gap junctions, neuromodulation and structural plasticity
- Frameworks for rate models and binary neurons
- Support for neuroscience interfaces (MUSIC, libneurosim)

- Highly efficient C++ core with a Python frontend
- Hybrid parallelization (OpenMP+MPI)
- Same code from laptops to supercomputers



NEST DESIGN GOALS

High accuracy and flexibility

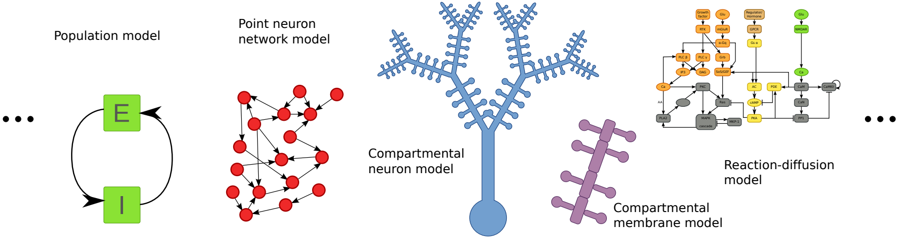
- Each neuron model is assigned an appropriate solver
- Exact integration is used for suitable neuron models
- Spikes are usually restricted to the computation time grid
- Spike interaction in continuous time for some models

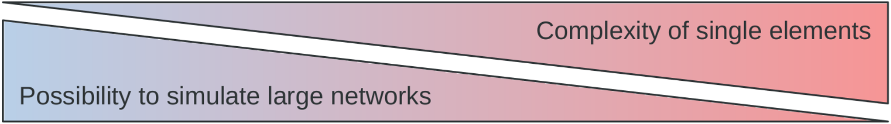
Constant quality assurance

- Automated unittest suite included in NEST build
- Continuous integration for all repository checkins
- Code review for all code contributions

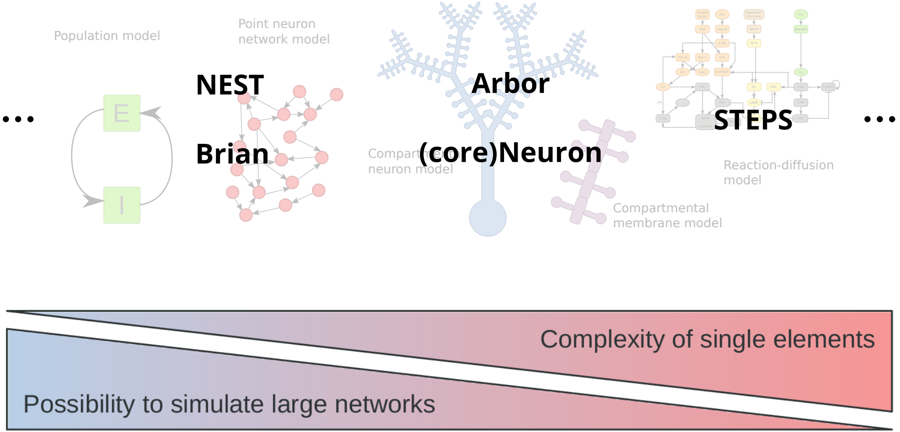
NEST's development is always driven by scientific needs

WHEN TO USE NEST?





WHEN TO USE NEST?



OBTAINING NEST

Download from `http://nest-simulator.org`

- Source code for official releases
- Virtual machine images (e.g. for use on Windows)

Open source development:

- `https://github.com/nest/nest-simulator`
- Direct access to current and future development
- Ability to fork and develop locally
- Pull requests for merging into the official version

From your distribution's package repository:

- PPA for Ubuntu and Debian
- Package in Neuro-Fedora

INSTALLING FROM SOURCE (LINUX)

1. **Download NEST and unpack** (in `$HOME` folder)**:**

```
wget https://git.io/vFxDo
tar -xzvf nest-2.18.0.tar.gz
```

2. **Create and enter build directory:**

```
mkdir nest-2.18.0-bld
cd nest-2.18.0-bld
```

3. **Configure, compile and install build:**

```
cmake -DCMAKE_INSTALL_PREFIX=$HOME/nest-2.18.0-inst ../nest-2.18.0
make -j4
make install
```

4. **Update environment** (in `$HOME/.bashrc` or similar file)**:**

```
. $HOME/nest-2.18.0-inst/bin/nest_vars.sh
```

NEST LIVE MEDIA USING VIRTUALBOX

1. **Download and install VirtualBox:** `http://virtualbox.org`
2. **Download NEST live media:** `http://nest-simulator.org/download`
   - Includes NEST, NEURON, Brian, PyNN, ...
3. **Start VirtualBox**:
   - File $\rightarrow$ Import Appliance $\rightarrow$ Appliance to import $\rightarrow$ Open
4. **Start VM, install VirtualBox Guest Additions CD image** (Devices $\rightarrow$). Follow instructions and restart guest OS
5. **Set up shared folders** (between host and guest):
   - Create shared folder in host OS, e.g. `vb_shared`
   - Devices $\rightarrow$ Shared Folders $\rightarrow$ Settings: add new
   - Uncheck 'Auto-mount' and 'Make permanent' $\rightarrow$ OK $\rightarrow$ OK
   - Create mount point in guest OS:

```
mkdir sharedir
sudo mount t vboxsf o uid=999,gid=999 vb_shared sharedir
```

HELP!

Within Python:

```
nest.help()
nest.helpdesk()
nest.help('iaf_psc_exp')
nest.help('Connect')
```

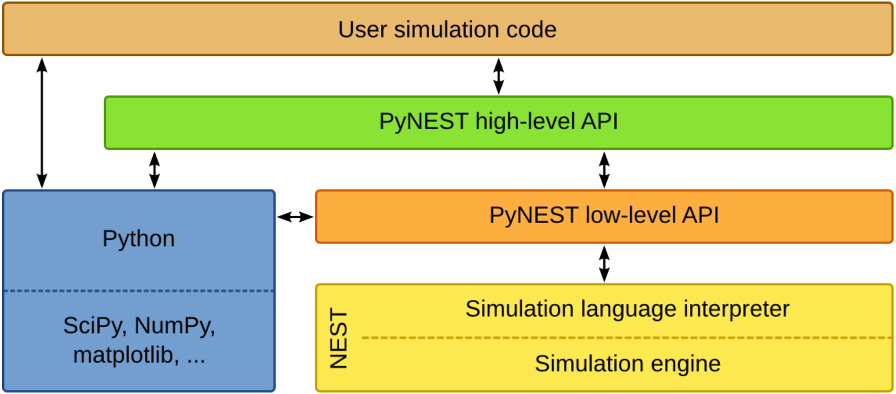
Online documentation:

`http://nest-simulator.org/documentation`

Community:

- NEST user mailing list
- Bi-weekly open video conference
- `http://nest-initiative.org/community`

HOW TO USE NEST?



Different user interfaces for maximum flexibility

HOW TO USE NEST?

Two different command line user interfaces:

- The built-in simulation language interpreter SLI

```
/n iaf_psc_alpha << /V_m -50.0 >> 5 Create def
/sd spike_detector Create def
n sd Connect
```

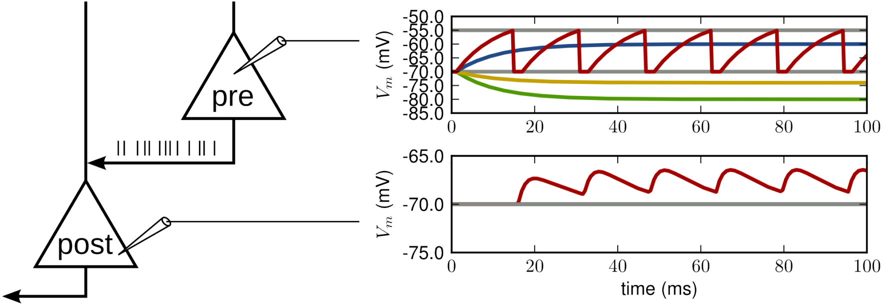
- The Python interface PyNEST

```
n = nest.Create("iaf_psc_alpha", 5, {"V_m": -50.0})
sd = nest.Create("spike_detector")
nest.Connect(n, sd)
```

NEST is also supported by the multi-simulator interface PyNN

NEURONAL SIMULATIONS IN NEST

A simulation in NEST mimics a neuroscientific experiment



NEURONAL SIMULATIONS IN NEST

- The network in NEST comprises a directed, weighted graph
  - Nodes represent either neurons or devices
  - Edges represent synapses between nodes
- Nodes are updated on a fixed-time grid, while spikes can also be in continuous time
- Neurons can be arbitrarily complex, not just point neurons
- Devices for stimulating neurons and recording their activity
- Synapse models to establish connections between nodes
- Parallelization and inter-process communication is handled transparently by NEST

JÜLICH Forschungszentrum

NEURON MODELS

- Integrate-and-fire models (`iaf_`)
  - Current-based (`iaf_psc`)
  - Conductance-based (`iaf_cond`)
  - Different post-synaptic shapes (`_alpha`, `_exp`, `_delta`)
- Single compartment Hodgin-Huxley models (`hh_`)
- Adaptive exponential integrate-and-fire models (`aeif_`)
- MAT2 neuron model (Kobayashi et al. 2009)
- Neuron models with few compartments

- Creation of neurons using the `Create` command:

```
Create(<model>, <num>, <params>)
```

STIMULATION DEVICES

Spike generators:

- `spike_generator` spikes at prescibed points in time
- `poisson_generator` spikes according to a Poisson distribution
- `gamma_sup_generator` spikes according to a Gamma distribution

Current generators

- `ac_generator` provides a sine-shaped current
- `dc_generator` provices a constant current
- `step_current_generator` provides a step-wise constant current
- `noise_generator` provides a random noise current

RECORDING DEVICES

- `spike_detector` records incoming spikes
- `multimeter` records analog quantities (potentials, conductances, ...)
- `voltmeter` records the membrane potential
- `correlation_detector` records pairwise cross-correlations between the spiking activity of neurons
- `weight_recorder` records the weight of connections

GENERAL PARAMETER ACCESS

All parameter access in NEST is carried out via dictionaries

- Retrieving the status of an element:

```
GetStatus(<element(s)>)
GetStatus(<element(s)>, <key(s)>)
```

- Setting properties of an element:

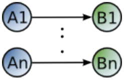
```
SetStatus(<element(s)>, <dict(s)>)
SetStatus(<element(s)>, <key(s)>, <value(s)>)
```

SPECIFICATION OF CONNECTIVITY

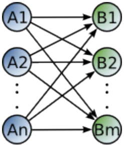
The Parameter conn_spec:

- defines the connection rule
- defines rule-specific parameter
- can be a string or a dictionary

```
A = Create('iaf_psc_alpha', n)
B = Create('spike_detector', n)
Connect(A, B, 'one_to_one')
```

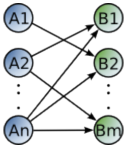



```
A = Create('iaf_psc_alpha', n)
B = Create('iaf_psc_alpha', m)
Connect(A, B)
```



SPECIFICATION CONNECTIVITY

```
A = Create("iaf_psc_alpha", n)
B = Create("iaf_psc_alpha", m)
conn_dict = {'rule': 'fixed_indegree',
             'indegree': N}
Connect(A, B, conn_dict)
```



Further rules and their keys:

- ’fixed_outdegree’, ’outdegree’
- ’fixed_total_number’, ’N’
- ’pairwise_bernoulli’, ’p’

SPECIFICATION OF SYNAPSE PROPERTIES

Using customized synapse model:

```
A = Create('iaf_psc_alpha', n)
B = Create('iaf_psc_alpha', n)
CopyModel('static_synapse','excitatory',
          {'weight':2.5, 'delay':0.5})
Connect(A, B, syn_spec='excitatory')
```

Insert synapse parameter directly into Connect():

```
syn_dict = {'model': 'static_synapse',
            'weight': 2.5, 'delay': 0.5}
Connect(A, B, syn_spec=syn_dict)
```

syn_spec defines the synapse model and synapse-specific parameters and can be a string or a dictionary

RANDOMIZATION OF SYNAPSE PROPERTIES

- specify distributed parameters as dictionaries

```
delay_dist = {'distribution': 'uniform',
              'low': 0.8, 'high': 2.5}

alpha_dist = {'distribution': 'normal_clipped',
              'low': 0.5, 'mu': 5.0,
              'sigma': 1.0}

syn_dict = {'model': 'stdp_synapse',
            'weight': 2.5,
            'delay': delay_dist,
            'alpha': alpha_dist}
```

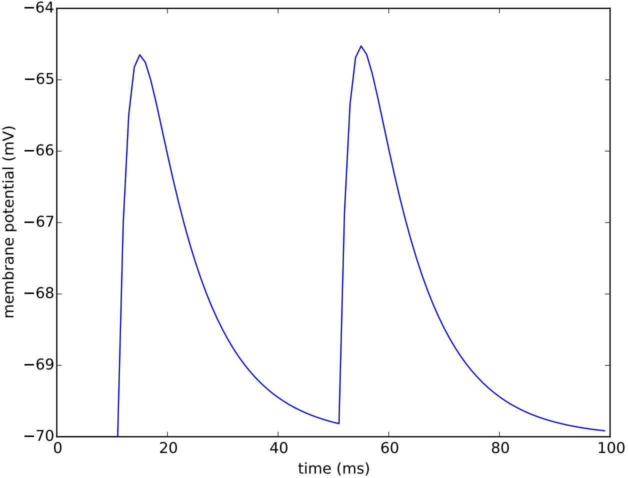
DISTRIBUTIONS

Distributions	Keys
’normal’	’mu’, ’sigma’
’normal_clipped’	’mu’, ’sigma’, ’low ’, ’high’
’lognormal’	’mu’, ’sigma’
’lognormal_clipped’	’mu’, ’sigma’, ’low’, ’high’
’uniform’	’low’, ’high’
’uniform_int’	’low’, ’high’
’binomial’	’n’, ’p’
’binomial_clipped’	’n’, ’p’, ’low’, ’high’
’exponential’	’lambda’
’exponential_clipped’	’lambda’, ’low’, ’high’
’gamma’	’order’, ’scale’
’gamma_clipped’	’order’, ’scale’, ’low’, ’high’
’poisson’	’lambda’
’poisson_clipped’	’lambda’, ’low’, ’high’

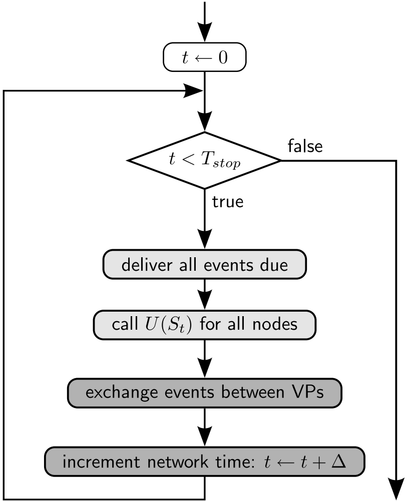
A FULL EXAMPLE

```
import nest                                   # import NEST module
neuron = nest.Create('iaf_psc_exp')           # create a neuron
voltmeter = nest.Create('voltmeter')          # create a voltmeter
spikegenerator = nest.Create('spike_generator') # create a spike generator
nest.SetStatus(spikegenerator, {'spike_times': [10., 50.]}) # let it spike

# connect spike generator and voltmeter to the neuron
nest.Connect(spikegenerator, neuron, syn_spec={'weight' : 1E3})
nest.Connect(voltmeter, neuron)

nest.Simulate(100.) # run the simulation

# read out recording time and voltage from voltmeter and plot them
times = nest.GetStatus(voltmeter)[0]['events']['times']
voltage = nest.GetStatus(voltmeter)[0]['events']['V_m']
pl.plot(times, voltage)
pl.xlabel('time (ms)'); pl.ylabel('membrane potential (mV)')
pl.show()
```

A FULL EXAMPLE



SIMULATION LOOP



- Simulation starts at $t = 0$
- We simulate for T_{stop} ms
- $U(S_t)$ propagates the neuron state S to time t
- VPs are virtual processes
- Δ is the minimal delay in the network

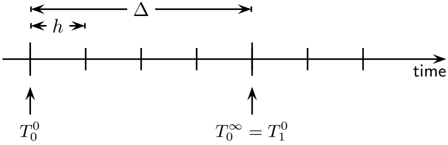
□ parallel on all threads
■ parallel on all processes

NETWORK UPDATE

- Neurons and devices are updated in the order of their creation
- During the run of the update function, all previous events are taken care of, and new events are created
- Spikes are buffered for local and remote delivery in the next time slice
- All other events are delivered immediately to local nodes
- Devices for stimulation and recording are replicated on each VP, which also deliver locally

NODE UPDATE

During an interval of the minimal transmission delay in the network (Δ), neurons are effectively decoupled.



- The update function of nodes (U) is called every Δ steps
- The nth time slice of length Δ starts at $T_n^0 = n \cdot \Delta$ and ends at $T_n^\infty = (n+1) \cdot \Delta$
- Internally, nodes use a time step of h (e.g. for solvers)

STRUCTURED NETWORKS USING TOPOLOGY

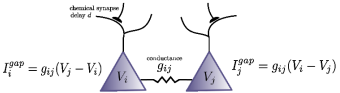
- **Invoke the topology module:**

  `from nest import topology`
- **Functionality:**
  - Set node positions on grids or arbitrary points in space (1D,2D,3D)
  - Nodes can be neurons or combinations of neurons and devices
  - Connect nodes in a position- and distance-dependent manner
  - Set boundary condition (periodic or not)
  - Enable/disable self-connections (autapses) or multiple connections (multapses)
- **Further reading:**

  `www.nest-simulator.org/documentation`

  $\rightarrow$ NEST user manual $\rightarrow$ Topological connections

GAP JUNCTIONS: IMPLEMENTATION



Neuron i (hh_psc_alpha_gap)

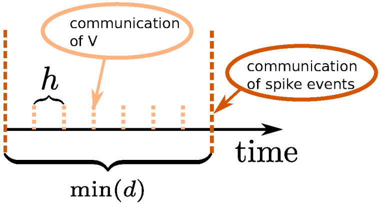
$y_i'(t) = f_i(y_i(t))\ ,\ y_i(t_0)$ given

$$\frac{V_i'}{C_m} = -I_i^{ionic}(V_i, m_i, h_i, n_i, p_i)$$

$$+I_i^{applied}(I_i^{ex}, I_i^{in})$$

$$+I_i^{gap}(V_i, V_j)$$

- at each time point neuron i needs membrane potential of neuron j
- large system of differential equations
- naïve: communication of V in each step
- better: Jacobi waveform relaxation

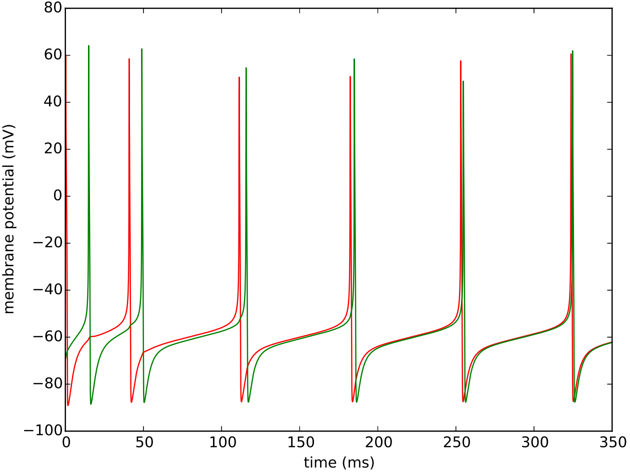


Hahne et al. (2015). A unified framework for spiking and gap-junction interactions in distributed neural network simulations. Frontiers in Neuroinformatics. 9:22

GAP JUNCTIONS: EXAMPLE

```
nest.SetKernelStatus({'max_num_prelim_iterations': 15,
                      'prelim_interpolation_order': 3,
                      'prelim_tol': 0.0001})

neuron = nest.Create('hh_psc_alpha_gap', 2, {'I_e': 100.})
nest.SetStatus([neuron[0]], {'V_m': -10.})
vm = nest.Create('voltmeter', { 'interval': 0.1})

syn_dic = {'model': 'gap_junction', 'weight': 0.5}
nest.Connect(neuron, neuron, syn_spec=syn_dic)
nest.Connect(vm, neuron)

nest.Simulate(351.)

vm_dict = nest.GetStatus(vm, 'events')
times_vm = vm_dict[0]['times']
V_vm = vm_dict[0]['V_m']
```

GAP JUNCTIONS: EXAMPLE



PARALLELIZATION IN NEST

Model developers and users (mostly) don't have to care about parallelization.

- A neuron n is created on the virtual process p, where

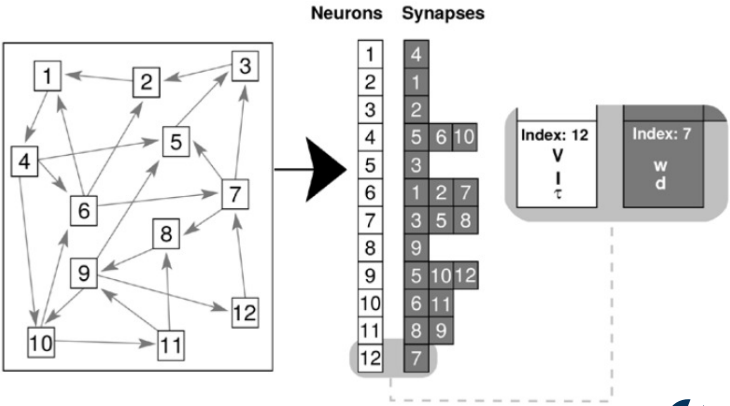
$$\mathrm{gid}(n) \mod \mathrm{N}_{\mathrm{MPI}} == p$$

- On all other VPs, a light-weight proxy is created
- Devices are replicated on each VP to distribute load

- There is one random number generator (RNG) per thread
- In addition, there is a global RNG that is kept synchronized

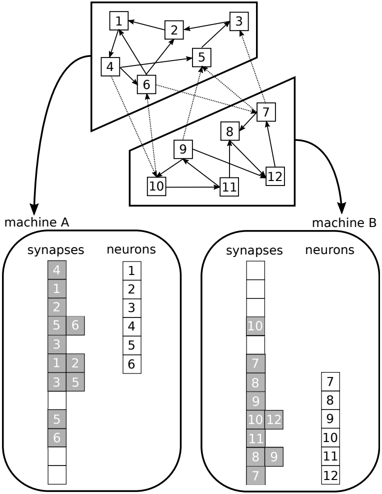
REPRESENTATION OF NETWORK STRUCTURE: SERIAL

- Each neuron and synapse maintains its own parameters
- Aynapses save the index of the target neuron



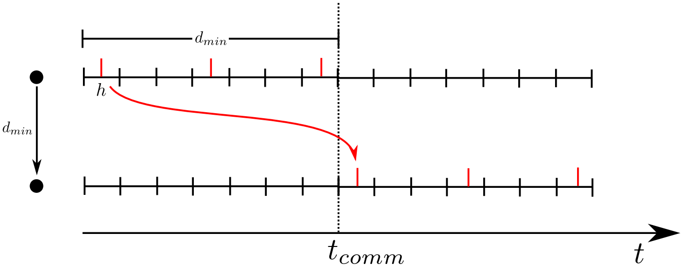
REPRESENTATION OF NETWORK STRUCTURE: DISTRIBUTED

- neurons are distributed round robin onto processes
- one target list for every neuron on each machine
- synapse stored on machine that hosts the target neuron
- wiring is a parallel operation



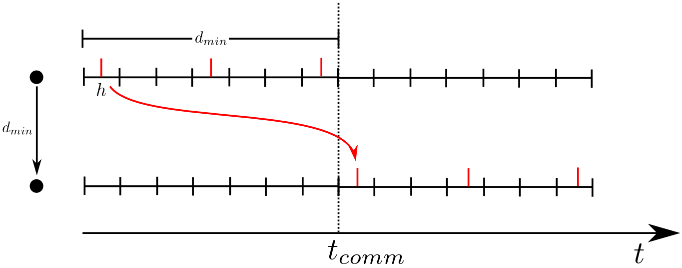
COMMUNICATION OF EVENTS

- communication only required in intervals of the minimal delay between neurons



COMMUNICATION OF EVENTS

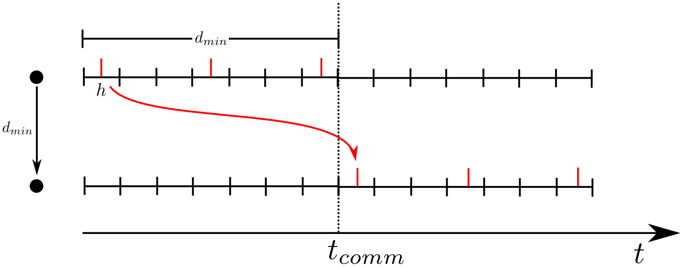
- communication only required in intervals of the minimal delay between neurons
- communication frequency independent of step size h



JÜLICH Forschungszentrum

COMMUNICATION OF EVENTS

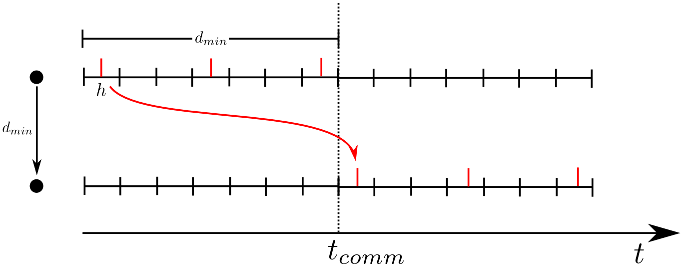
- communication only required in intervals of the minimal delay between neurons
- communication frequency independent of step size h
- less communications containing more data is more efficient due to overhead of communication between machines



JÜLICH Forschungszentrum

COMMUNICATION OF EVENTS

- communication only required in intervals of the minimal delay between neurons
- communication frequency independent of step size h
- less communications containing more data is more efficient due to overhead of communication between machines
- buffer sent to all machines (MPIAllgather)



EVENT-DRIVEN VS. TIME-DRIVEN

Event-driven simulation:

- Visit a neuron only when it receives an event (e.g. a spike)
- From $y(t_i)$, calculate $y(t_{i+1})$

Time-driven simulation:

- Visit each neuron in each time step h
- From $y(ih)$, calculate $y([i+1]h)$

EVENT-DRIVEN VS. TIME-DRIVEN

	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

NEST uses a 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.

EVENT-DRIVEN VS. TIME-DRIVEN

NEST uses a hybrid approach to simulation

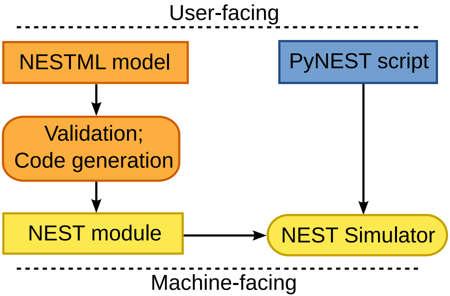
- input events to neurons are frequent: time-driven algorithm
  - If the dynamics is nonlinear, we need a numerical method to solve it, e.g.:
    - Forward Euler: $y([i+1]h) = y(ih) + h \cdot \dot{y}(ih)$
    - Runge-Kutta (kth order)
    - Runge-Kutte-Fehlberg with adaptive step size
    - …

  → 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

NESTML

NESTML is a domain-specific language for neuron and synapse models.



Using PyNEST, you instantiate and connect the models that you define in NESTML.

NESTML: DESIGN PRINCIPLES

- Concise; low on boilerplate
- Speak in the vernacular of the neuroscientist (keywords such as `neuron`, `synapse`)
- Easy (dynamical) equation handling coupled with imperative-style programming (`if V_m >= threshold: ...`)

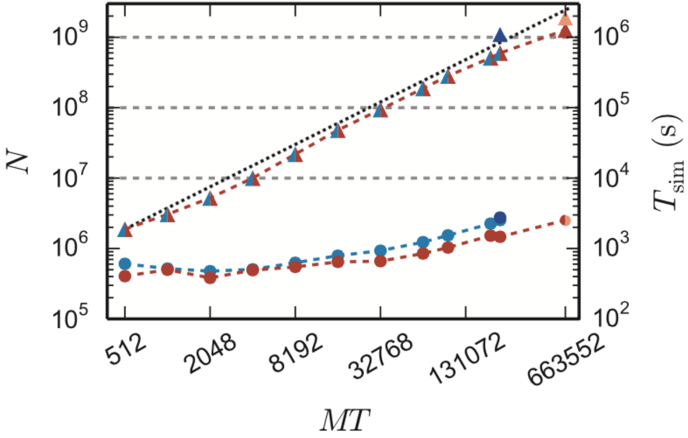
NESTML comes with a code generation toolbox.

- Code generation (model definition but not instantiation)
- Automated ODE analysis and solver selection
- Flexible addition of targets using Jinja2 templates

NESTML: EXAMPLE

```
neuron iaf_psc_exp:
  state:
    V_abs mV = 0 mV
  end

  equations:
    shape G = exp(-t / tau_syn)
    V_abs' = -V_abs / tau_m
           + (I_ext + convolve(G, spikes)) / C_m
  end

  parameters:
    C_m     pF = 250 pF
    tau_m   ms = 10 ms
    tau_syn ms = 2 ms
    V_threshold mV = 40 mV  # w.r.t. zero!
  end

  input:
    spikes pA <- spike
    I_ext pA <- current
  end

  update:
    integrate_odes()
    if V_abs > V_threshold:
      V_abs = 0
      emit_spike()
    end
  end
end
```

NEST PERFORMANCE



Maximum network size and corresponding run time as function of number of virtual processes on the K computer (red) and JUQUEEN (blue). Taken from Kunkel et al., (2014), Front Neuroinf. DOI: 10.3389/fninf.2014.00078

REFERENCES AND FURTHER READING

- The NEST Simulator homepage at https://www.nest-simulator.org
- Scientific publications about the technical side of the simulator
  $\Rightarrow$ nest-simulator.org/publications
- Our user mailing list for support and discussions
  $\Rightarrow$ nest-simulator.org/community.
- A bi-weekly open video conference
  $\Rightarrow$ nest-simulator.org/videoconference.
- An annual user and developer conference
  $\Rightarrow$ nest-simulator.org/conference.

Please tell us about problems. We only can fix what we know of!

MORE NEST IN HEIDELBERG

News and Features by Dennis Terhorst
Wednesday, November 27, 09:30-10:00

PyNEST tutorial by Håkon Mørk and Stine Brekke Vennemo
Wednesday, November 27, 13:30-14:30

NESTML tutorial by Charl Linssen
Wednesday, November 27, 16:00-16:30

Coupling NEST and TVB by Sandra Diaz
Wednesday, November 27, 16:30-17:00

NEST Desktop by Stefan Rotter and Sebastian Spreizer
Thursday, November 28, 09:00-10:30

ACKNOWLEDGMENTS

This presentation is based on previous work by many people.

- Hannah Bos
- David Dahmen
- Moritz Deger
- Jochen Martin Eppler
- Espen Hagen
- Abigail Morrison
- Jannis Schuecker
- Johanna Senk
- Tom Tetzlaff
- Sacha van Albada
- Charl Linssen