Neuromorphic computing

Human and animal models in biorobotics
M.Sc. programme in Bionics Engineering

Lorenzo Vannucci
lorenzo.vannucci@santannapisa.it

November 12th, 2019

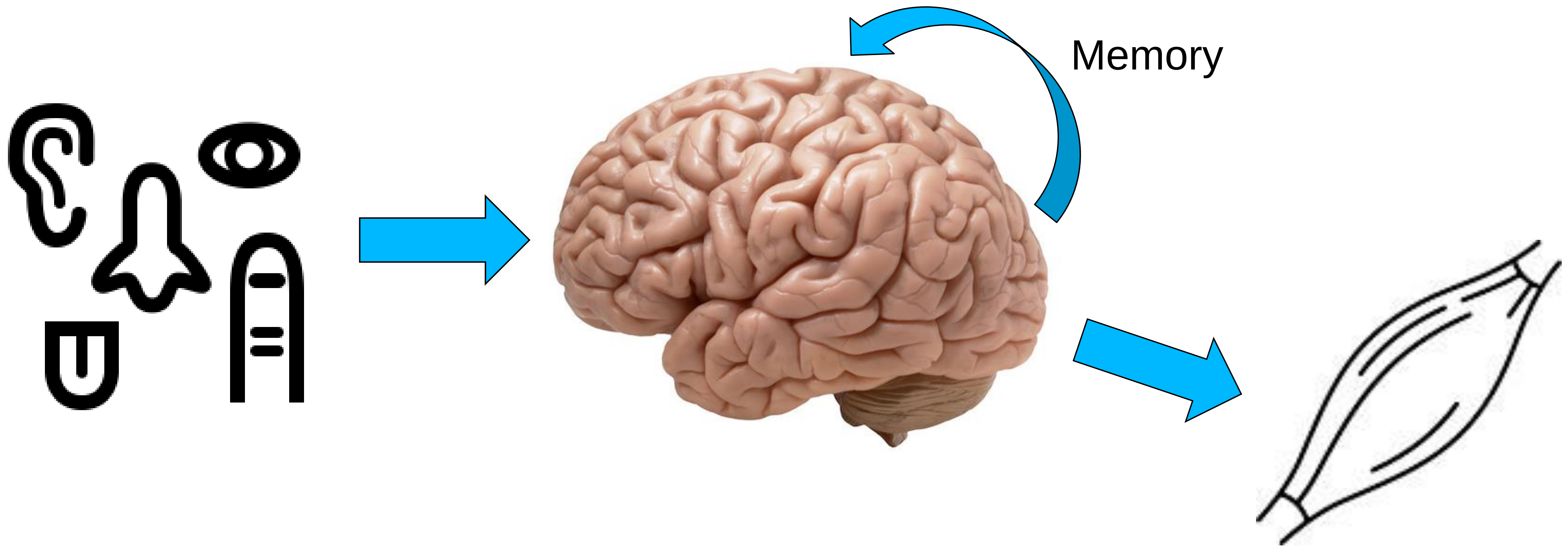
Outline

Outline

What is neuromorphic computing?

We can define **neuromorphic computing** as the act of performing a computation in a manner similar to the brain.

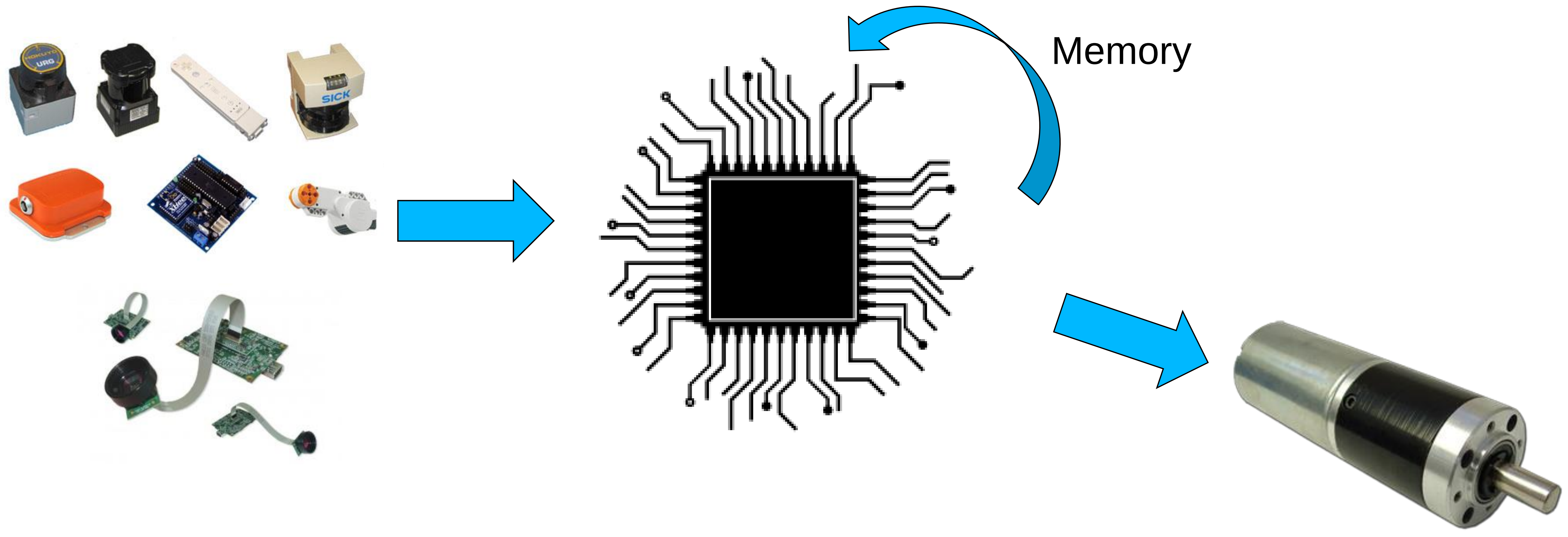
Our brain elaborates inputs coming from our sensors and produces outputs in term of generated motions and stored information.



Classic computing

This kind of computing is very similar to what can be found in a robotic controller.

But the sensors and actuators are completely different, compared to the ones of humans and animals, thus the brain is substituted by a computer.



Brain vs machines

Getting to know your Brain

- 1.3 Kg, about 2% of body weight
- 10^{11} neurons
- neuron growth:
  - 250,000 / min (early pregnancy)
  - -1 neuron/s (adult life)

Brain vs machines

Getting to know your Brain

- 1.3 Kg, about 2% of body weight
- 10^{11} neurons
- neuron growth:
  - 250,000 / min (early pregnancy)
  - -1 neuron/s (adult life)

Getting to know your CPU

- 50g
- 10^{10} transistors (Ryzen 9)
- no modification over lifetime

Brain vs machines

Getting to know your Brain

- 1.3 Kg, about 2% of body weight
- 10^{11} neurons
- neuron growth:
  - 250,000 / min (early pregnancy)
  - -1 neuron/s (adult life)

Getting to know your CPU

- 50g
- 10^{10} transistors (Ryzen 9)
- no modification over lifetime

“Operating mode” of Neurons

- analog computation in the soma
- digital pulses along axons
- 10^{14} stochastic synapses
- typical operating frequency:
  - < 100Hz, asynchronous

Brain vs machines

Getting to know your Brain

- 1.3 Kg, about 2% of body weight
- 10^{11} neurons
- neuron growth:
  250,000 / min (early pregnancy)
  -1 neuron/s (adult life)

Getting to know your CPU

- 50g
- 10^{10} transistors (Ryzen 9)
- no modification over lifetime

“Operating mode” of Neurons

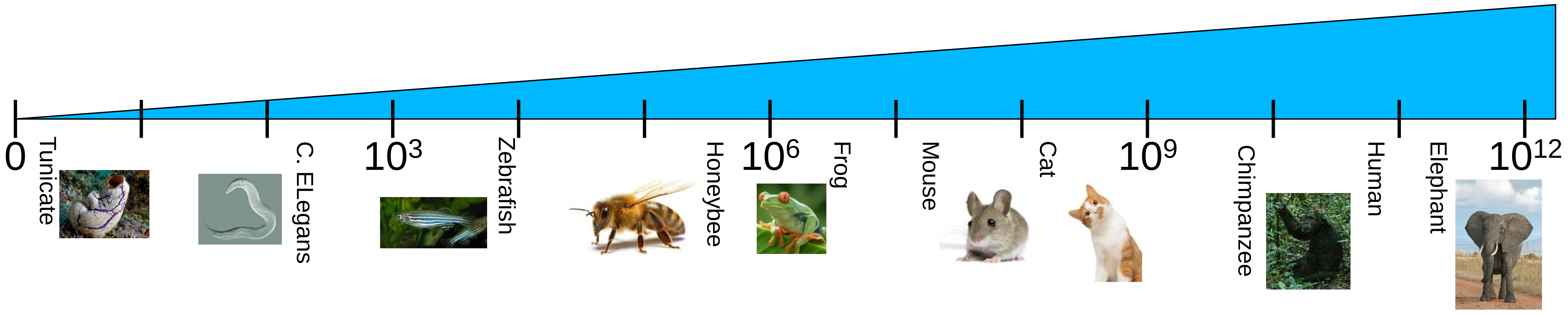
- analog computation in the soma
- digital pulses along axons
- 10^{14} stochastic synapses
- typical operating frequency:
  < 100Hz, asynchronous

“Operating mode” of CPUs

- digital Boolean logic processing
- digital signal propagation
- reliable storage of data
- typical operating frequency:
  GHz, synchronous

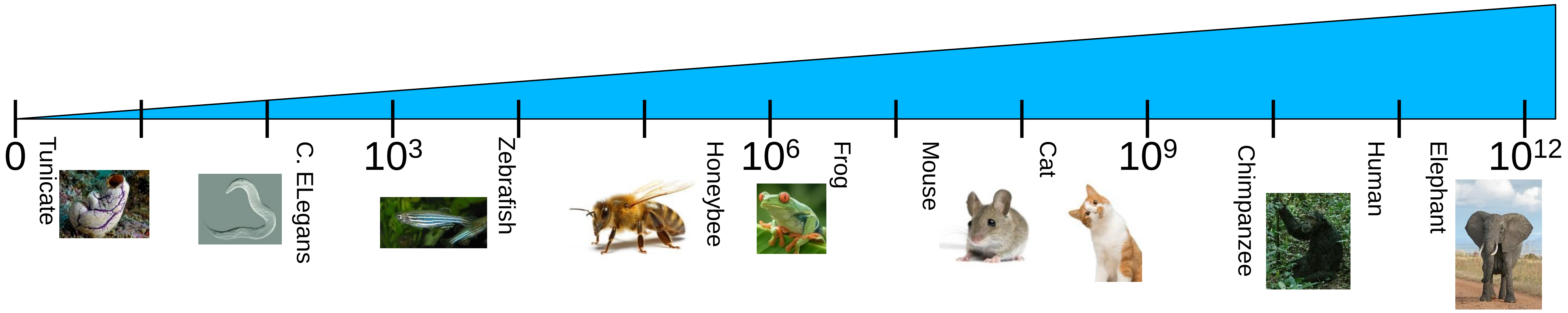
Why neuromorphic computing (in robotics)?

A brain is what defines a living being

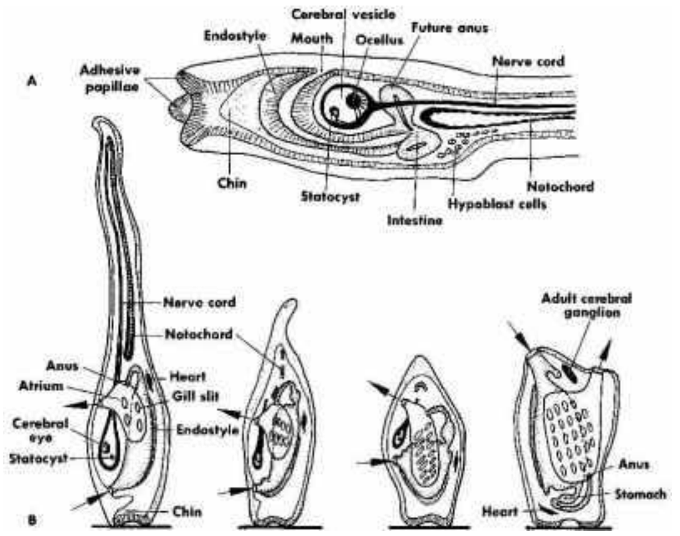


Why neuromorphic computing (in robotics)?

A brain is what defines a living being

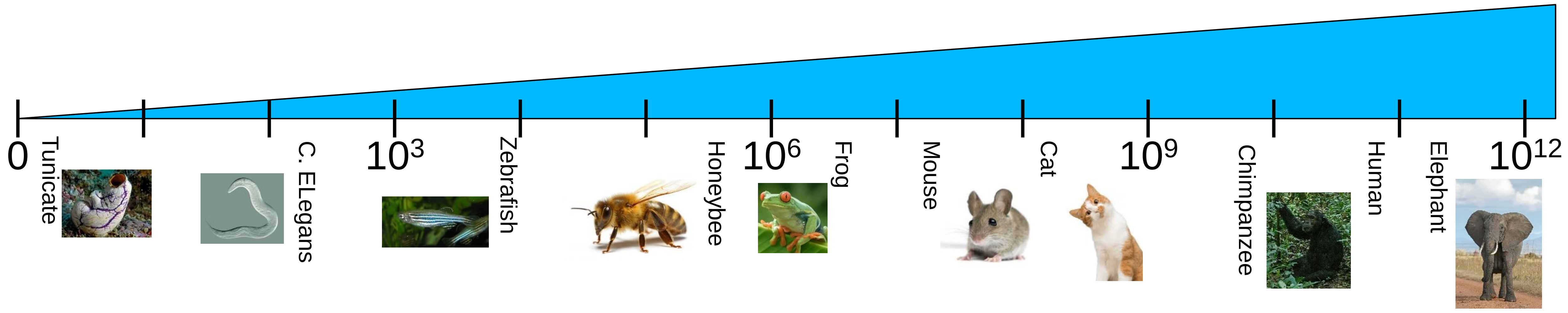


A brain is what make us be active in the environment (cfr. sea squirt).

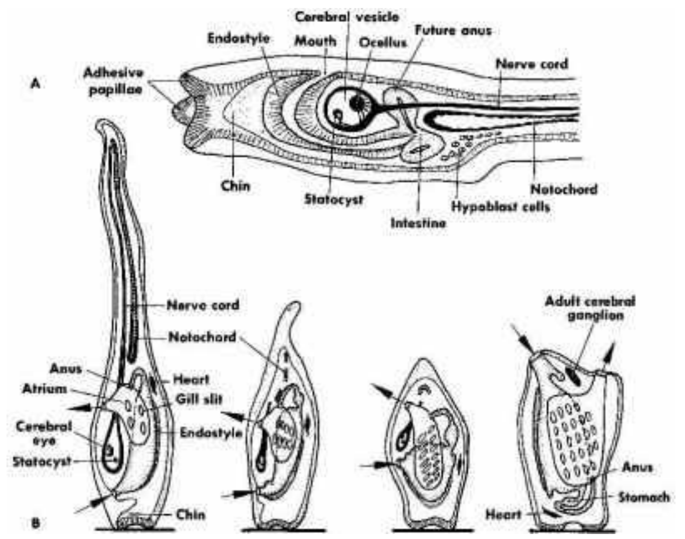


Why neuromorphic computing (in robotics)?

A brain is what defines a living being



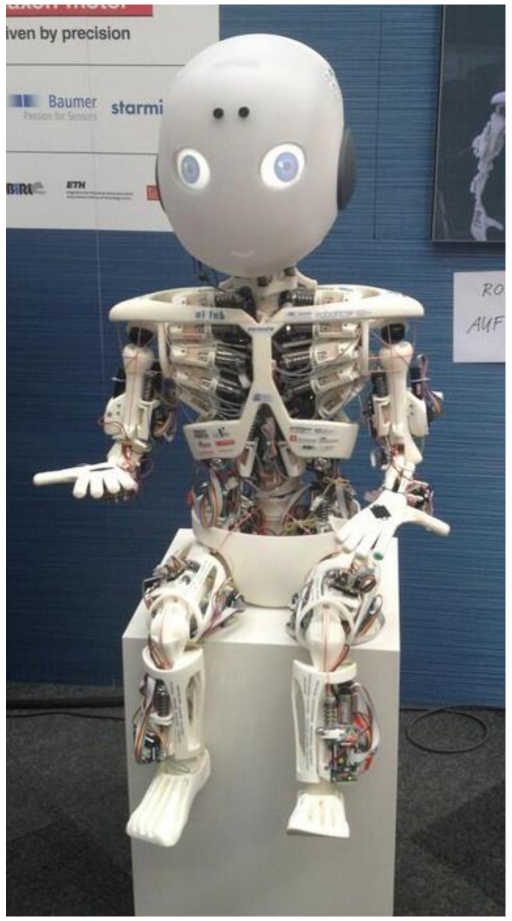
A brain is what make us be active in the environment (cfr. sea squirt).



Thus, we can apply neuromorphic computing to **give brains to robots**.

Why neuromorphic computing (in robotics)?

Today, bio-inspired sensing and actuation technologies are starting to emerge.

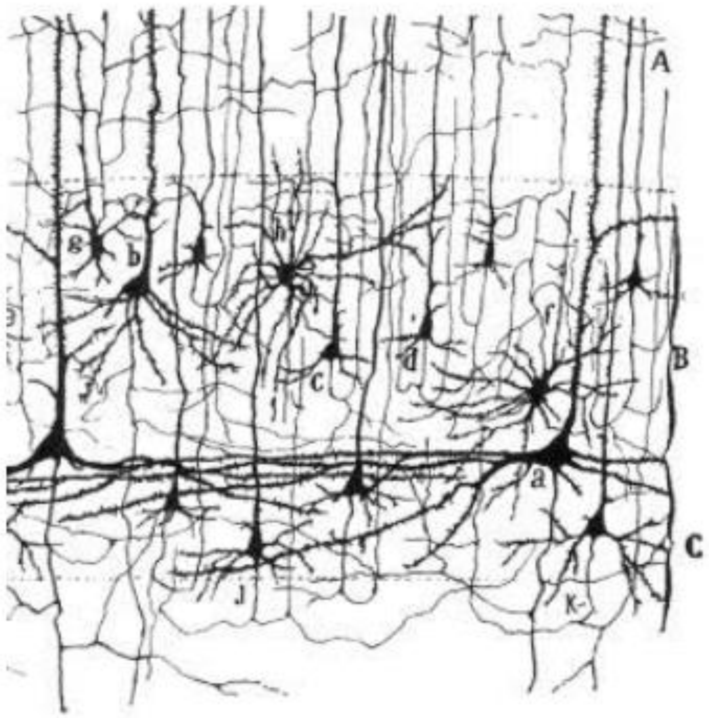


As such, neuromorphic computing can be used to control this new kind of robots.

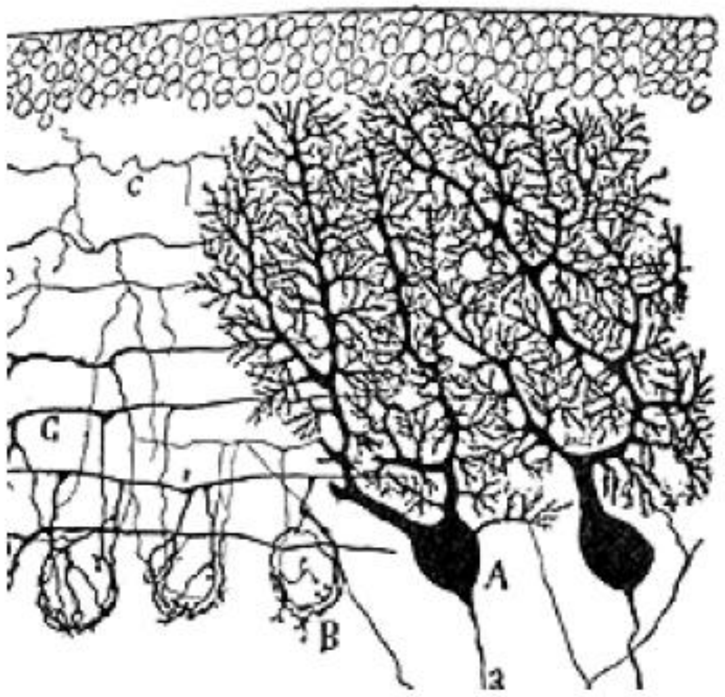
Outline

Neuronal physiology

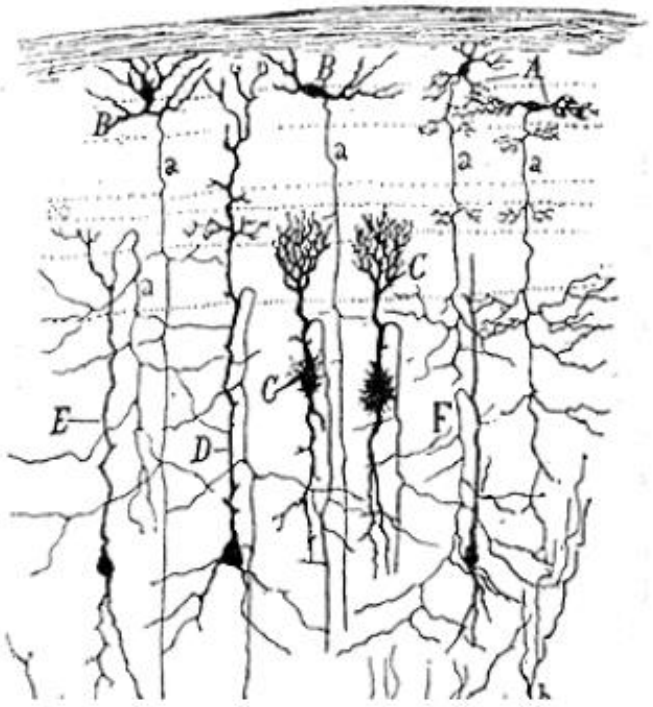
The neuron is the fundamental structural and functional unit of the brain.



Visual Cortex



Cerebellum

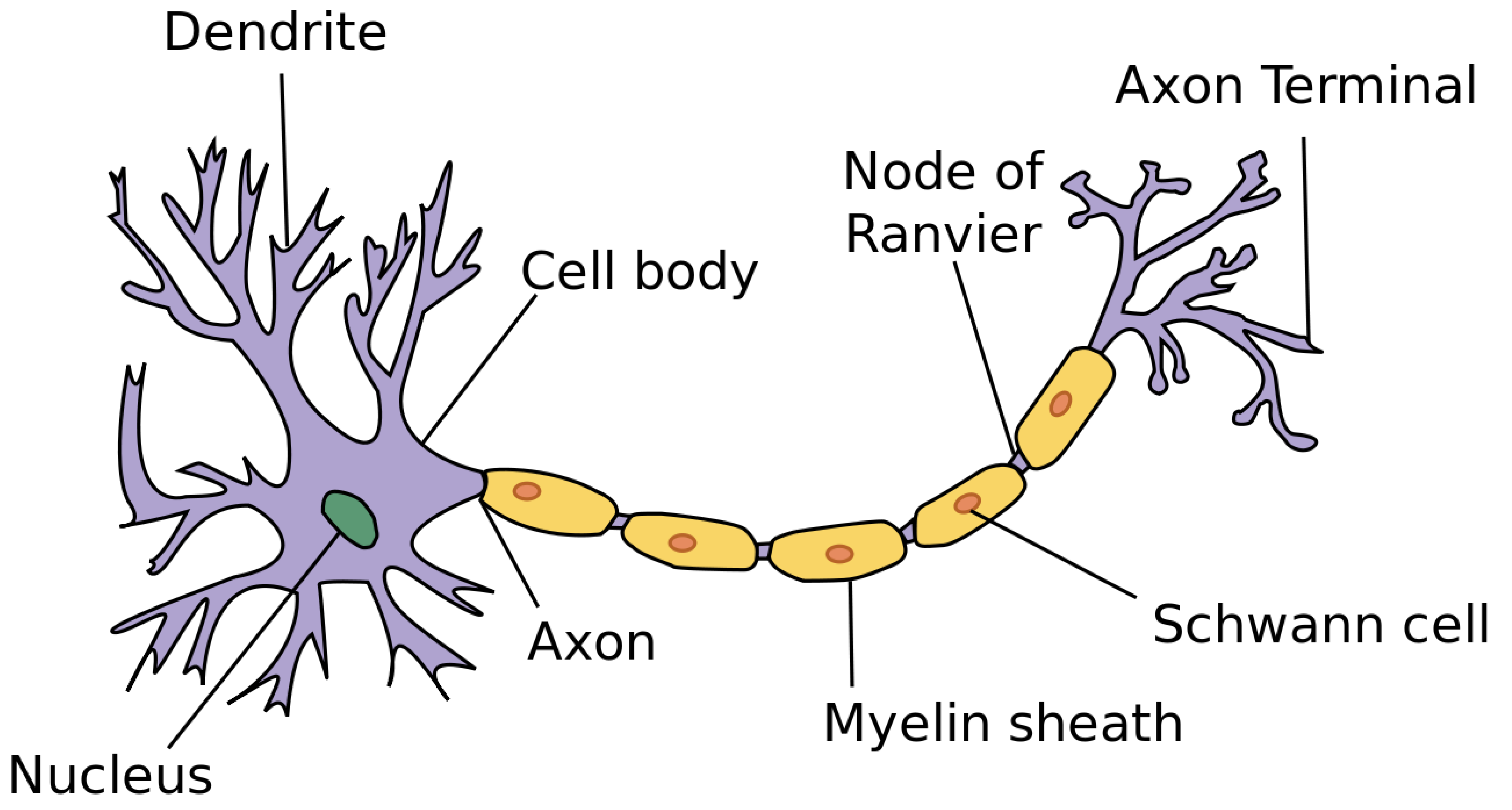


Optic Tectum

(Drawings by Ramón y Cajal, c. 1900)

Neuronal physiology

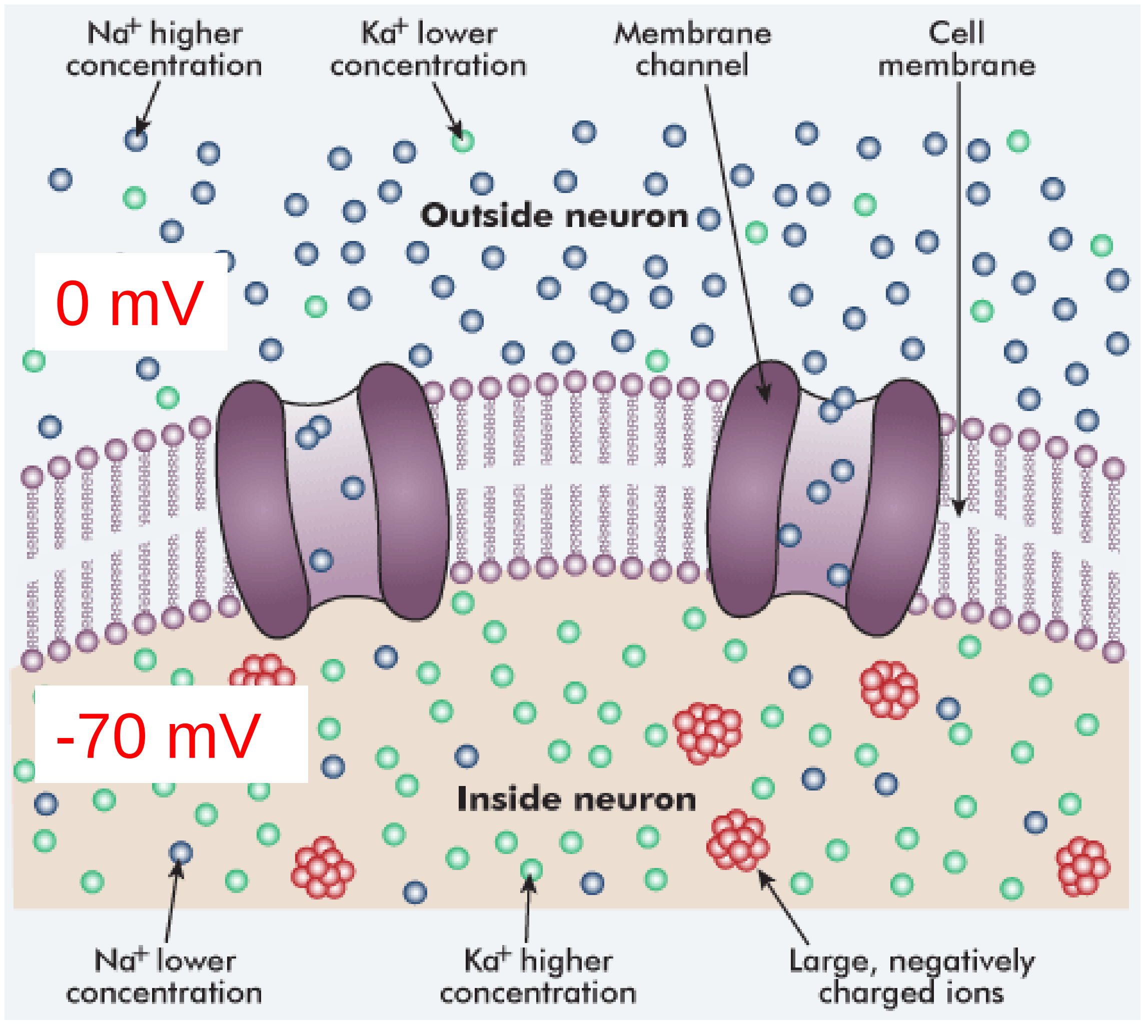
Many kind of neurons share the same cellular physiology.



Neuronal physiology

Neuronal electrophysiological activity lies on the cell membrane.

- Lipid bilayer, impermeable to charged ions.
- Ionic channels allow ions to flow in or out, selectively.
- The neuron maintains a **potential difference** across it membrane via the ionic pumps (expelling Na^+ and allowing K^+ in).
- When no external stimulus is present, we can refer to it as **resting potential**.



Action potentials

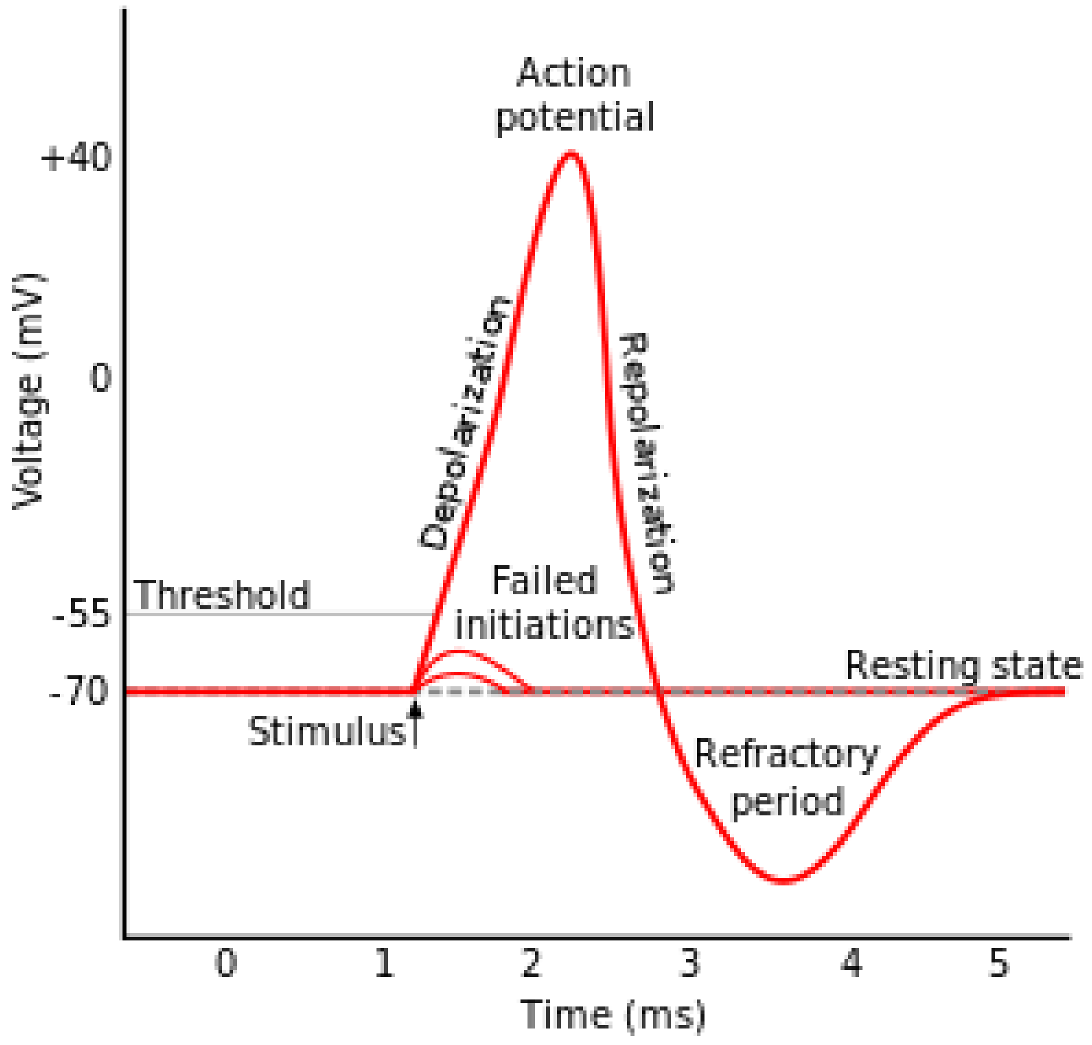
The activity of a neuron (its “output”) is the **action potential** (or spike), generated by voltage-gated ionic channels.

1. An external electric stimulus reach the membrane, depolarizing it.

2. Depolarization of the membrane opens Na^+ channels (→ even more depolarization).

3. If membrane potential exceeds the **threshold potential**, an action potential occurs.

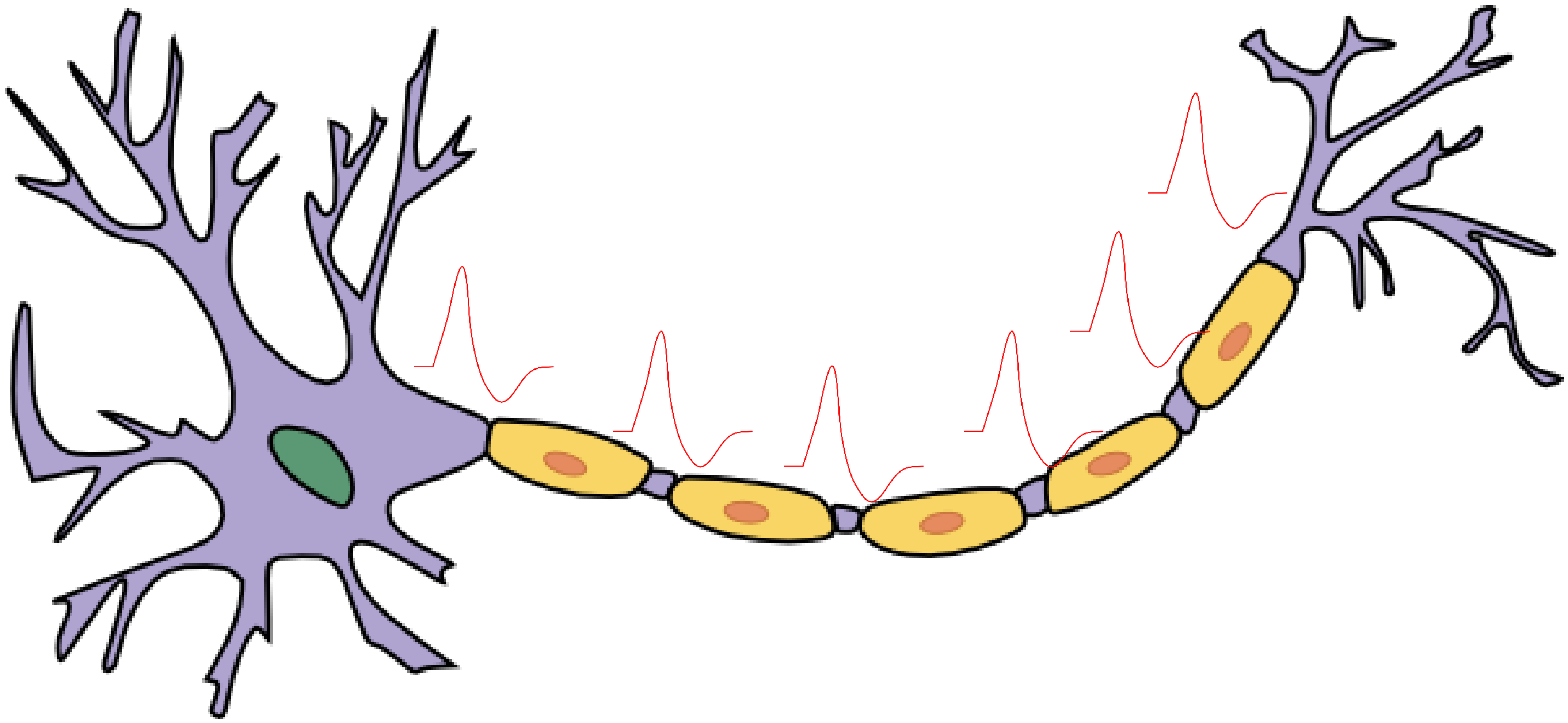
4. Afterwards, the membrane repolarize by expelling K^+ ions and the neuron enters the refractory period.



Action potentials

The action potential is transmitted through the axon towards other neurons.

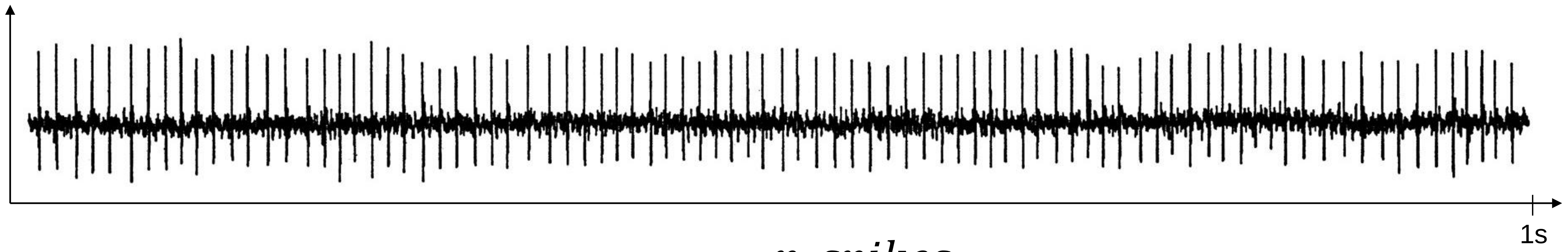
Each non-myelinated section (node of Ranvier) replicates the spike.



Propagation speed ranges from 1 to 100 m/s.

Action potentials

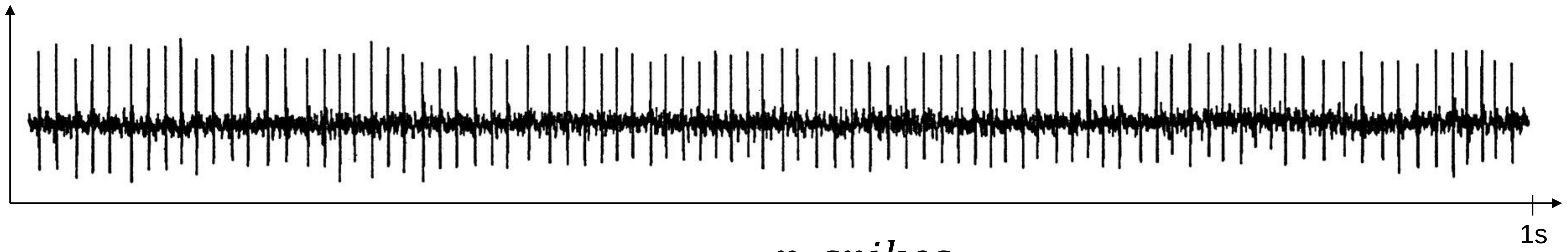
The activity of a neuron is measured by computing its **firing rate**, expressed as the mean number of spikes per second.



$$rate = \frac{n.\,spikes}{time}$$

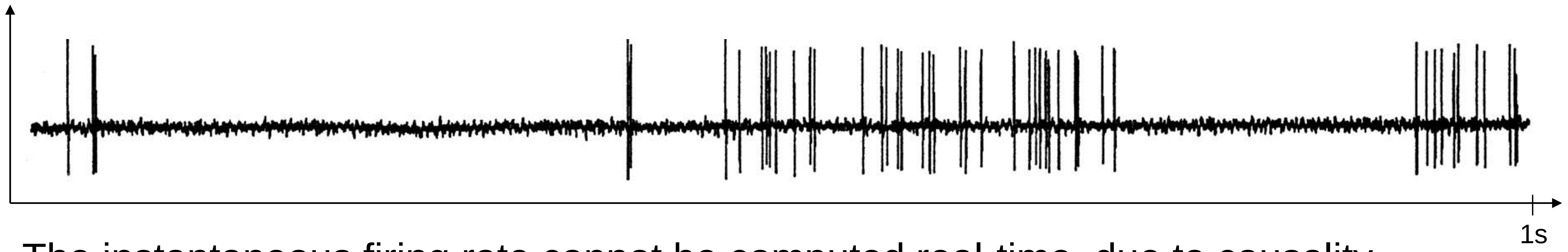
Action potentials

The activity of a neuron is measured by computing its **firing rate**, expressed as the mean number of spikes per second.



$$rate = \frac{n.\,spikes}{time}$$

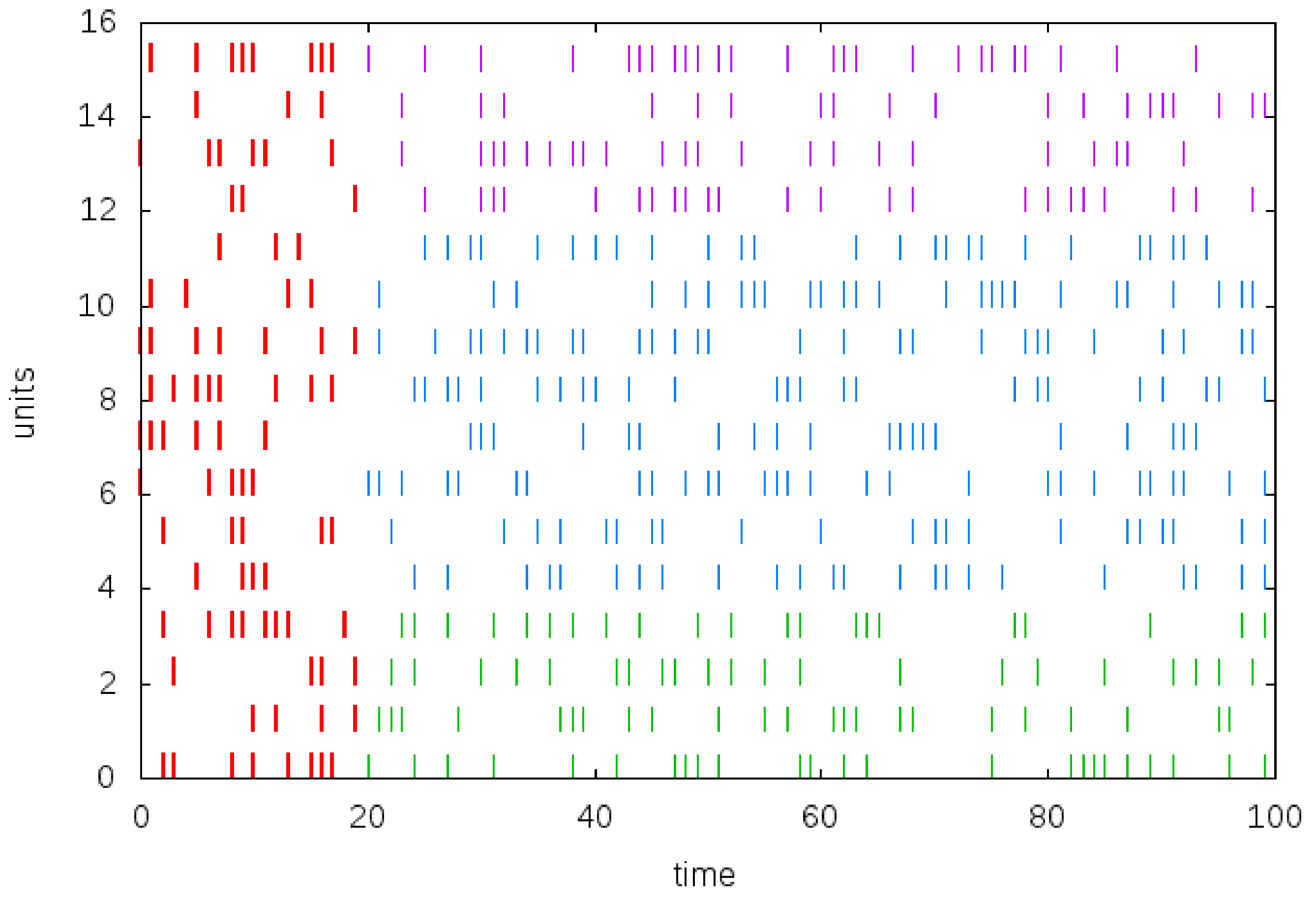
It is not always an easy task!

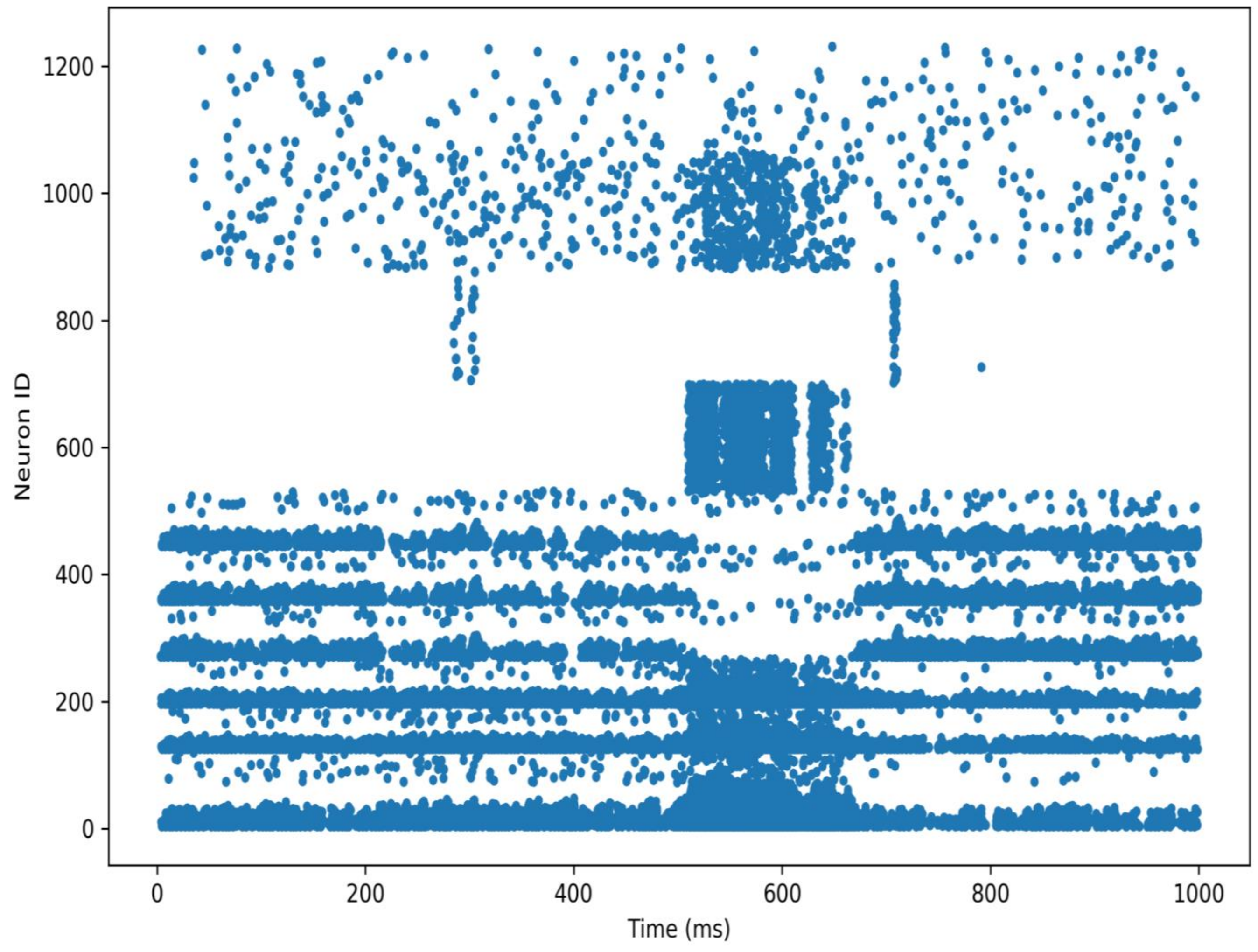


The instantaneous firing rate cannot be computed real-time, due to causality.

Action potentials

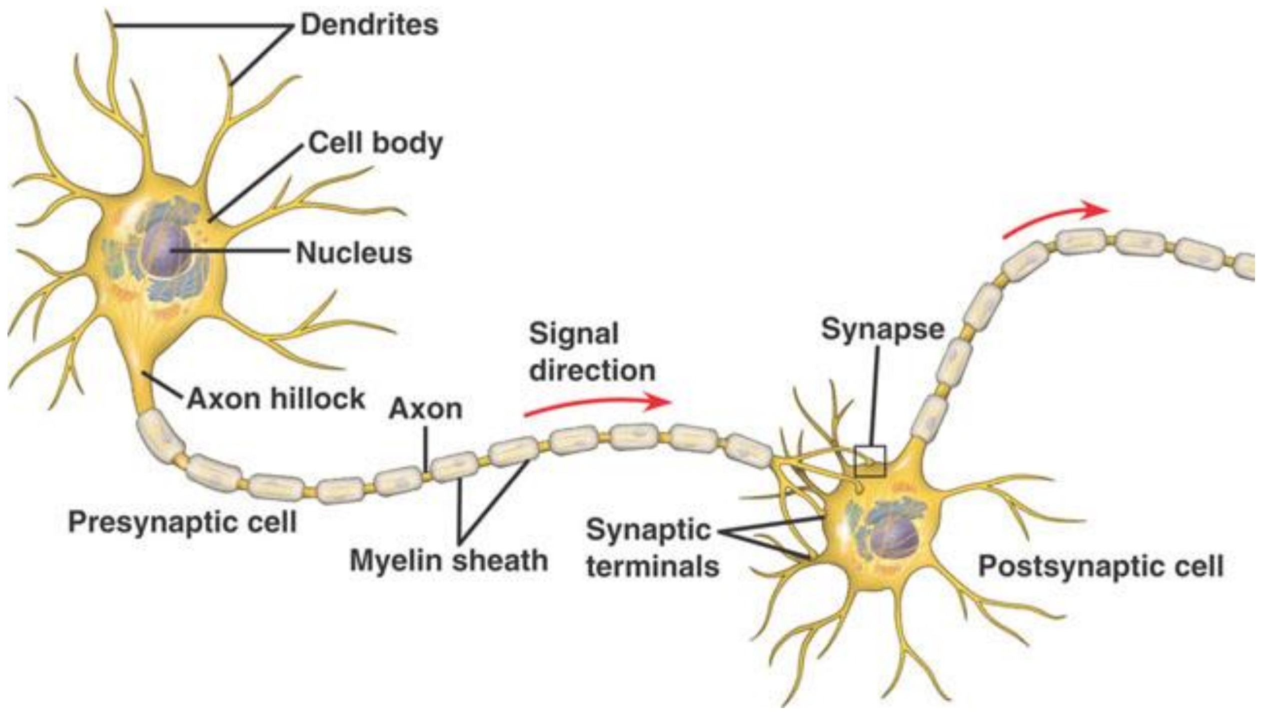
Usually, we are interested in looking at the spike events, instead of the membrane potential, and for a high number of neurons (a population).
We can do so with raster plots.

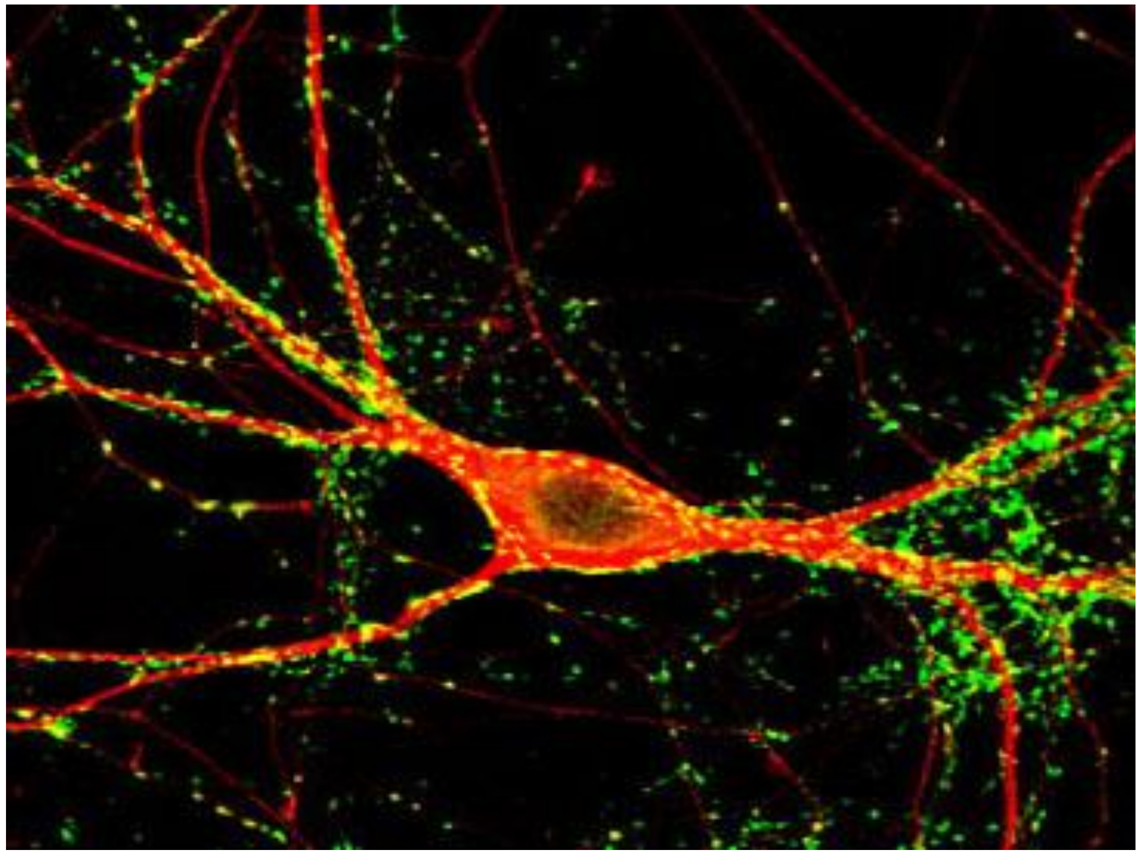




Synapses

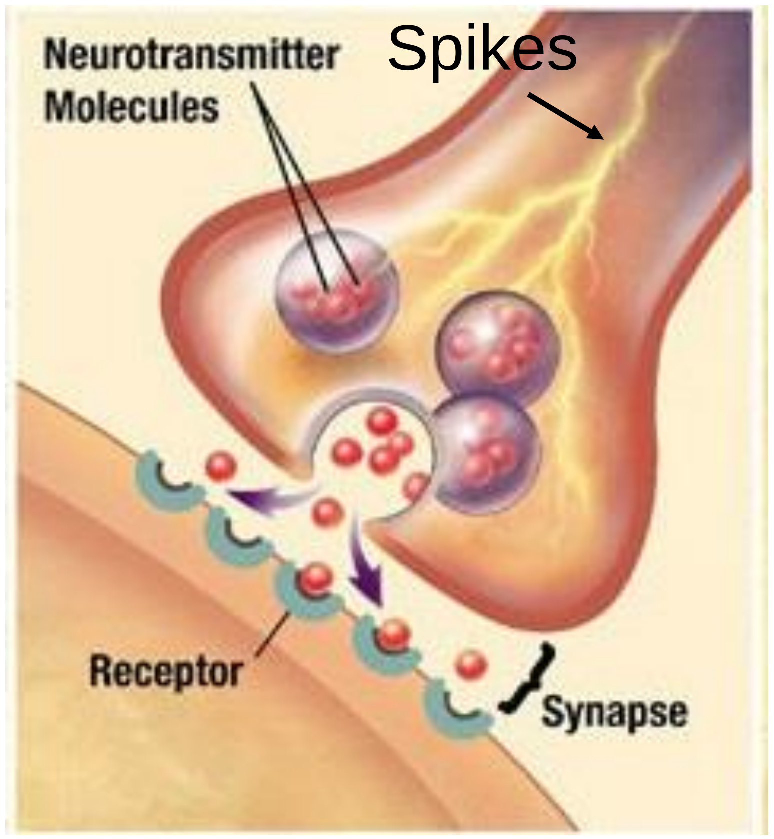
Axons and dendrites are connected through **synapses**. Each neuron has roughly 1000-10000 synapses.



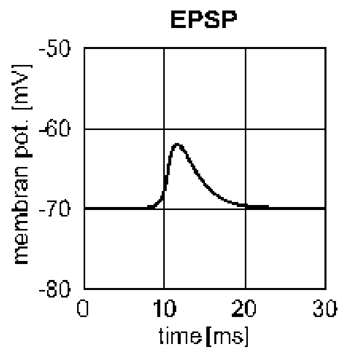


Synapses

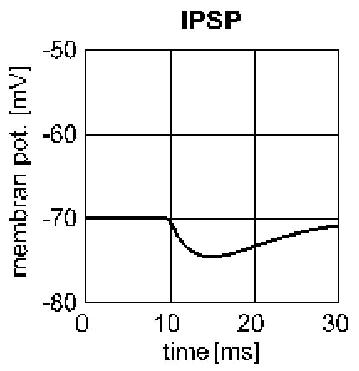
Synapses can be chemical or electrical, excitatory or inhibitory:



- a chemical **excitatory** synapse releases Glutamate → opening of ion channels for Na$^+$ influx → membrane depolarization (membrane potential *increases*);



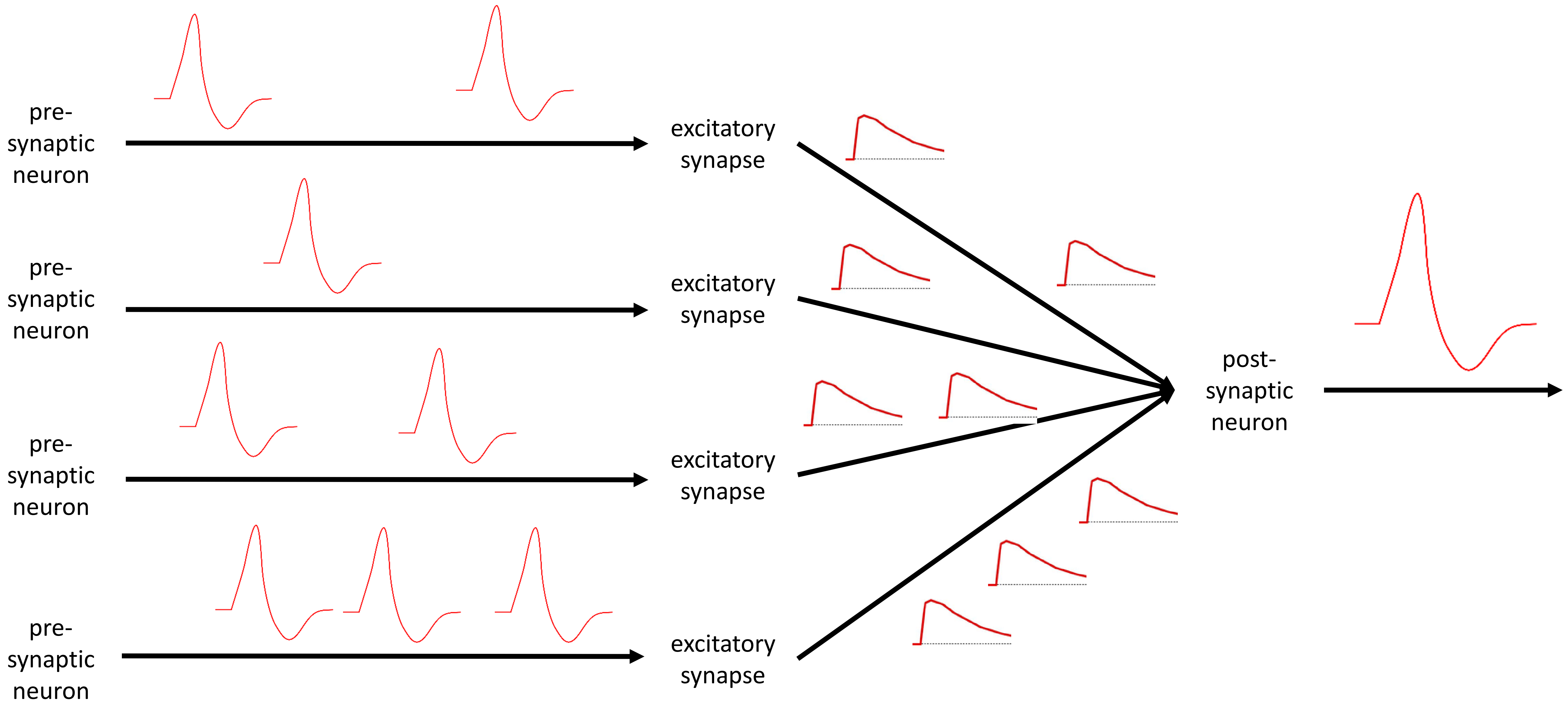
- a chemical **inhibitory** synapse releases GABA neurotransmitter → K+ leaves cell through ion channels → membrane hyperpolarization (membrane potential *decreases*).



Every synapse, once reached by an action potential, generates a *postsynaptic current* (PSC) which turns in a *postsynaptic potential* (PSP).

Synapses and action potentials

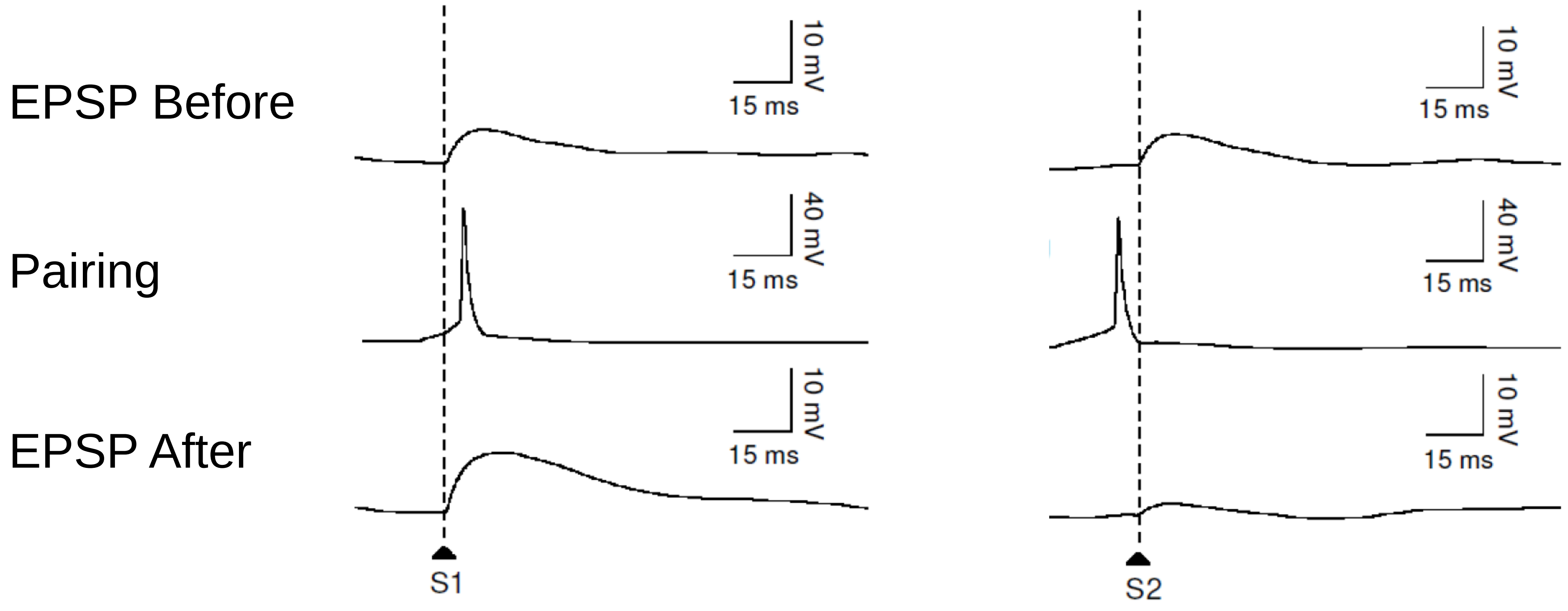
Each spike coming for pre-synaptic neurons and activating excitatory synapses contributes to the generation of an action potential in the post-synaptic neuron.



Synaptic plasticity

Synapses are the basis for memory and **learning**.

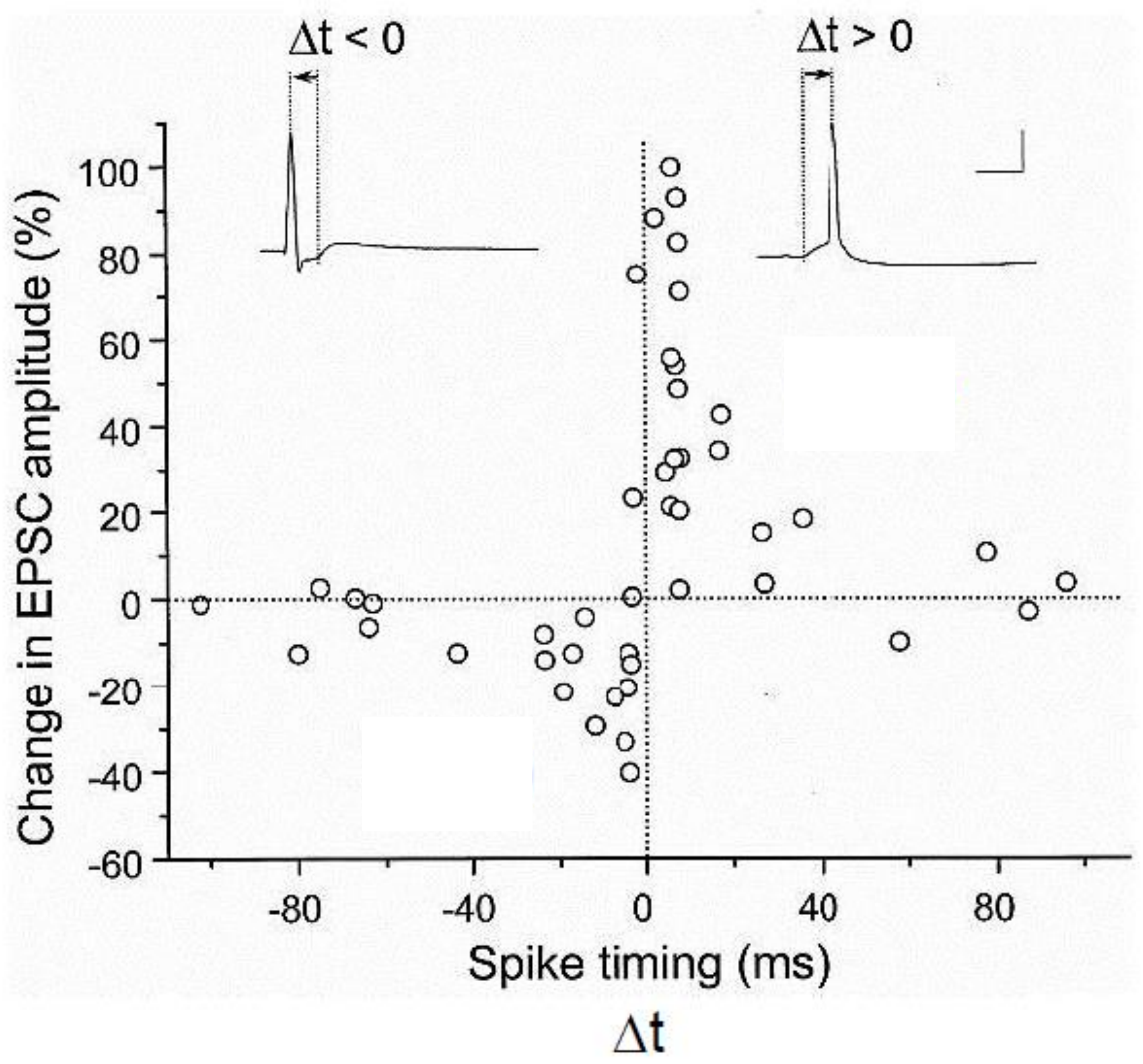
If neuron A repeatedly takes part in making neuron B spike, then the synapse from a to B is strengthened and vice versa. This leads to two phenomena:



Long term potentiation (LTP)   Long term depression (LTD)

Synaptic plasticity

This adaptation mechanism depends on the timing of the EPSP and the action potential. Thus, it is called **Spike-Timing Dependent Plasticity** (STDP).



Neural coding

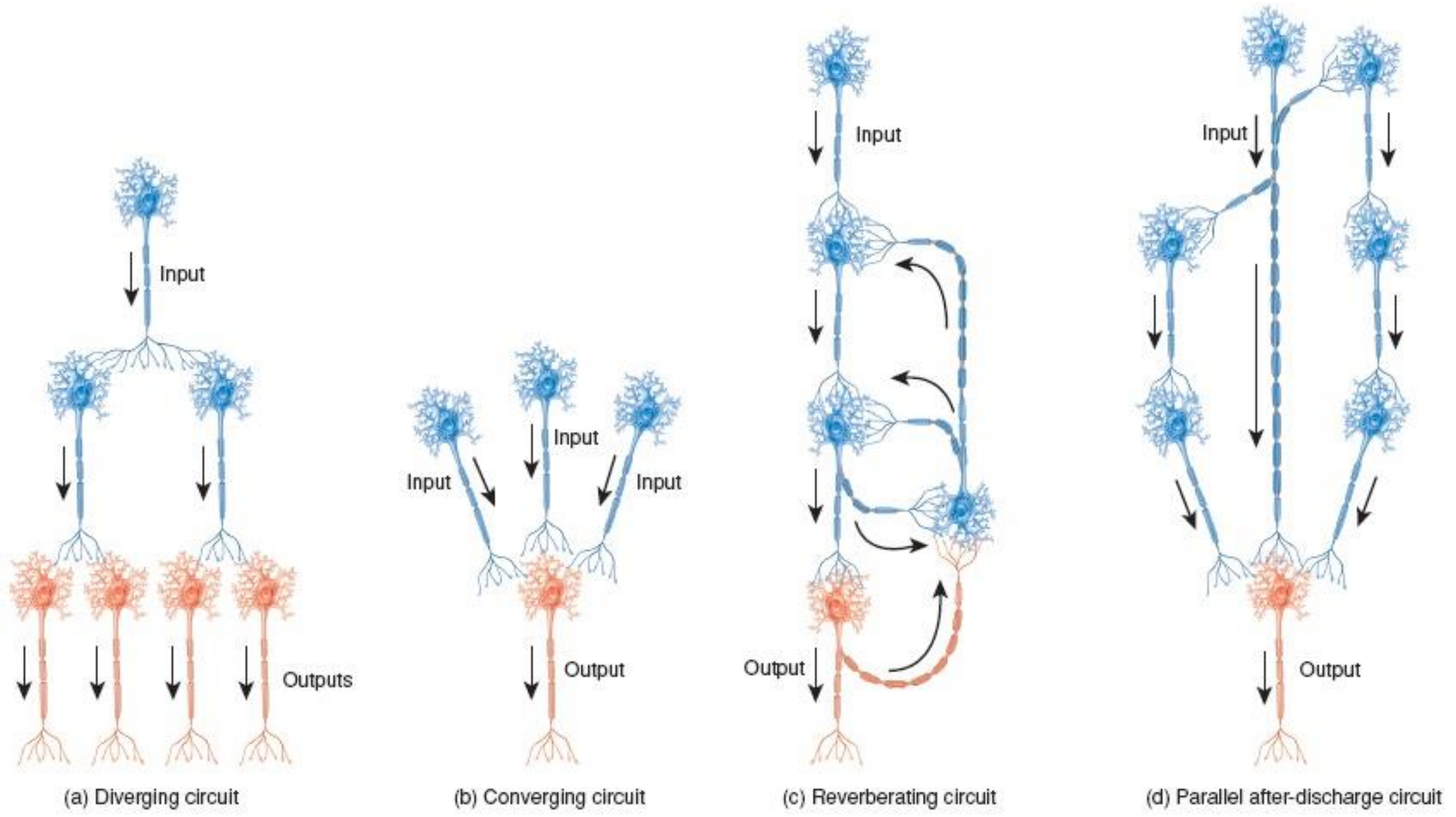
What kind of information can a neuron represent?

Neural information processing

What kind of information can a neuron process?

None! (by himself)

Information is stored in the network topology and synaptic properties.

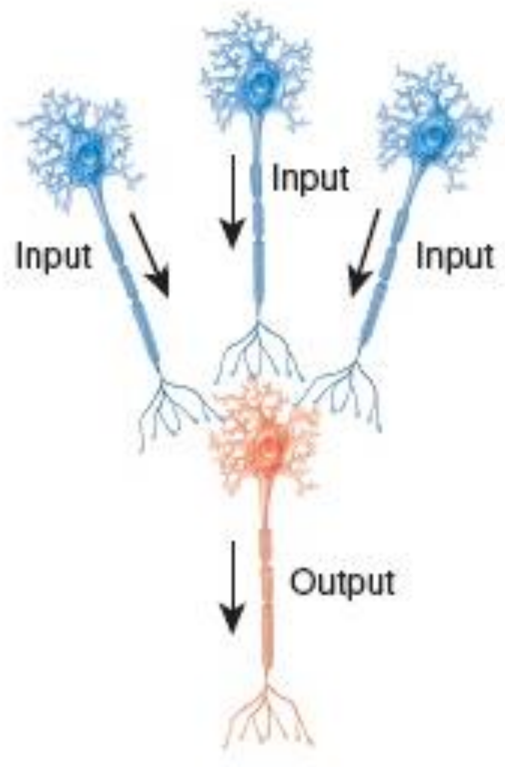


Receptive fields

A simple way of encoding information is the **receptive field** topology.

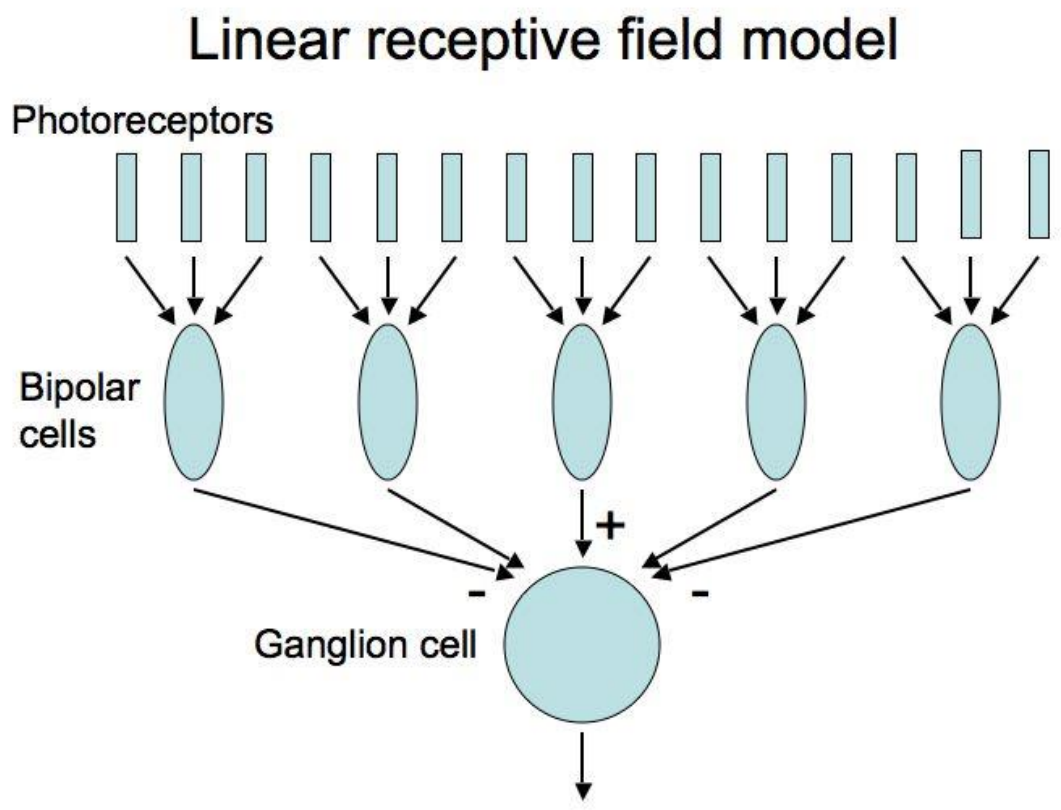
Each receptive field is made up of several input neurons and one output neuron that modulates the combination of their responses.

Receptive fields have been identified in the human brain to encode sensory information (auditory system, somatosensory system, visual system).



Receptive fields

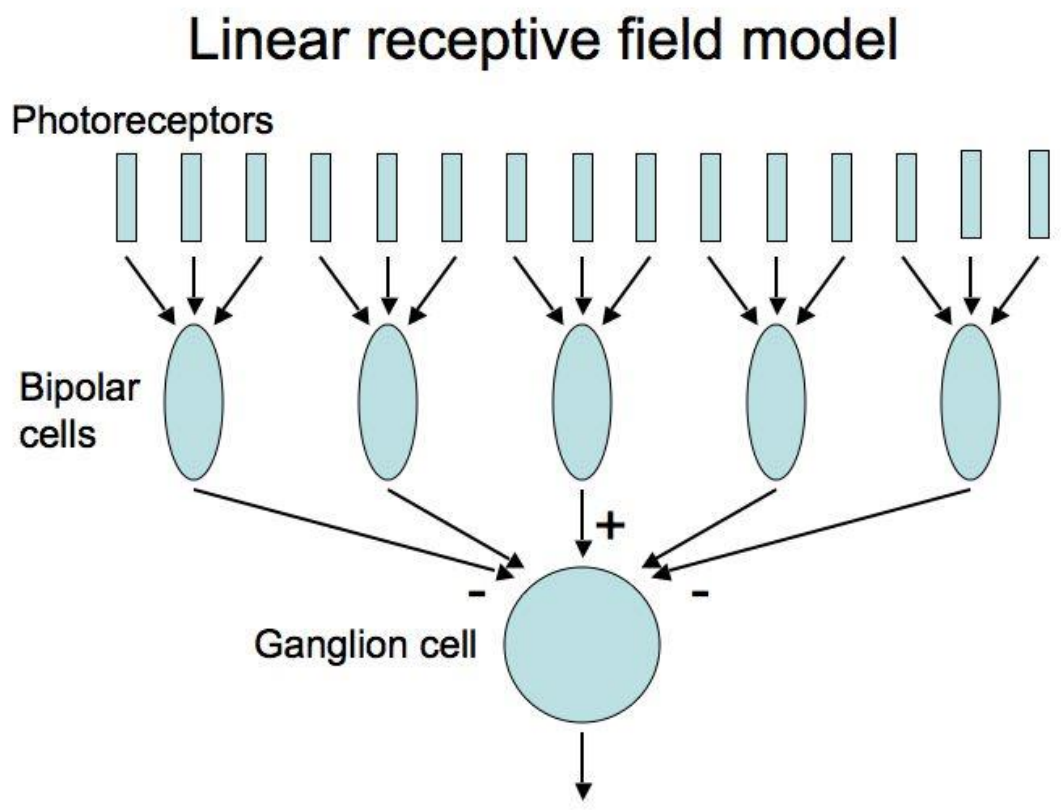
The retinal circuit implements receptive fields to process the image.



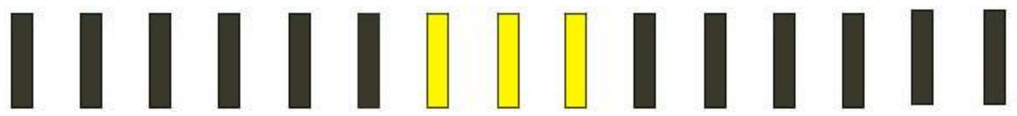
Preferred stimulus?

Receptive fields

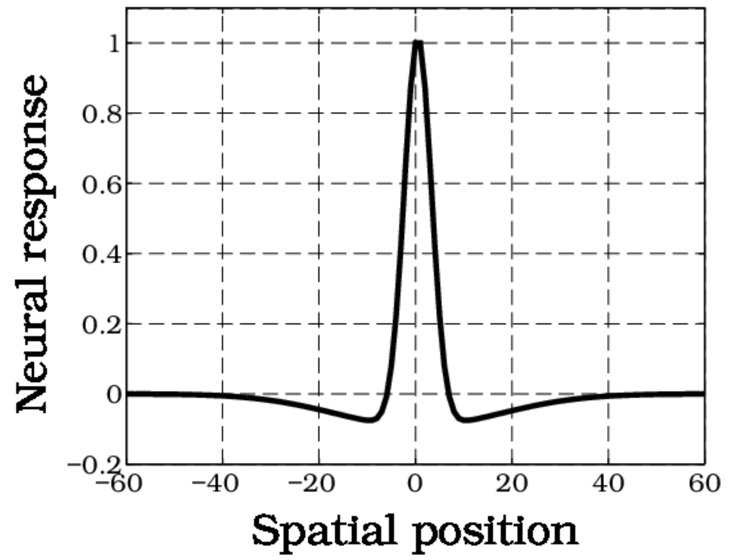
The retinal circuit implements receptive fields to process the image.



Preferred stimulus

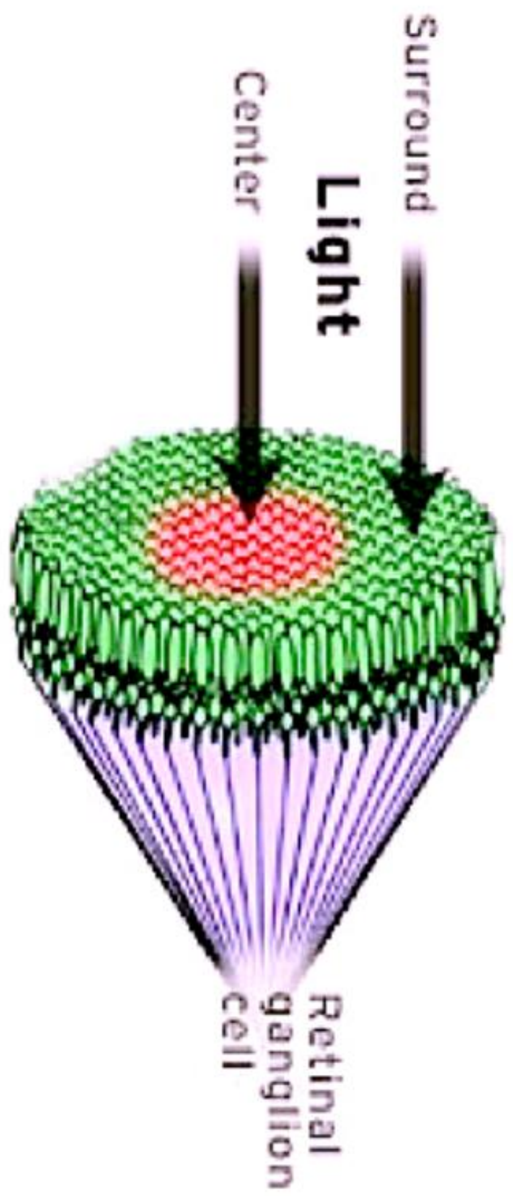


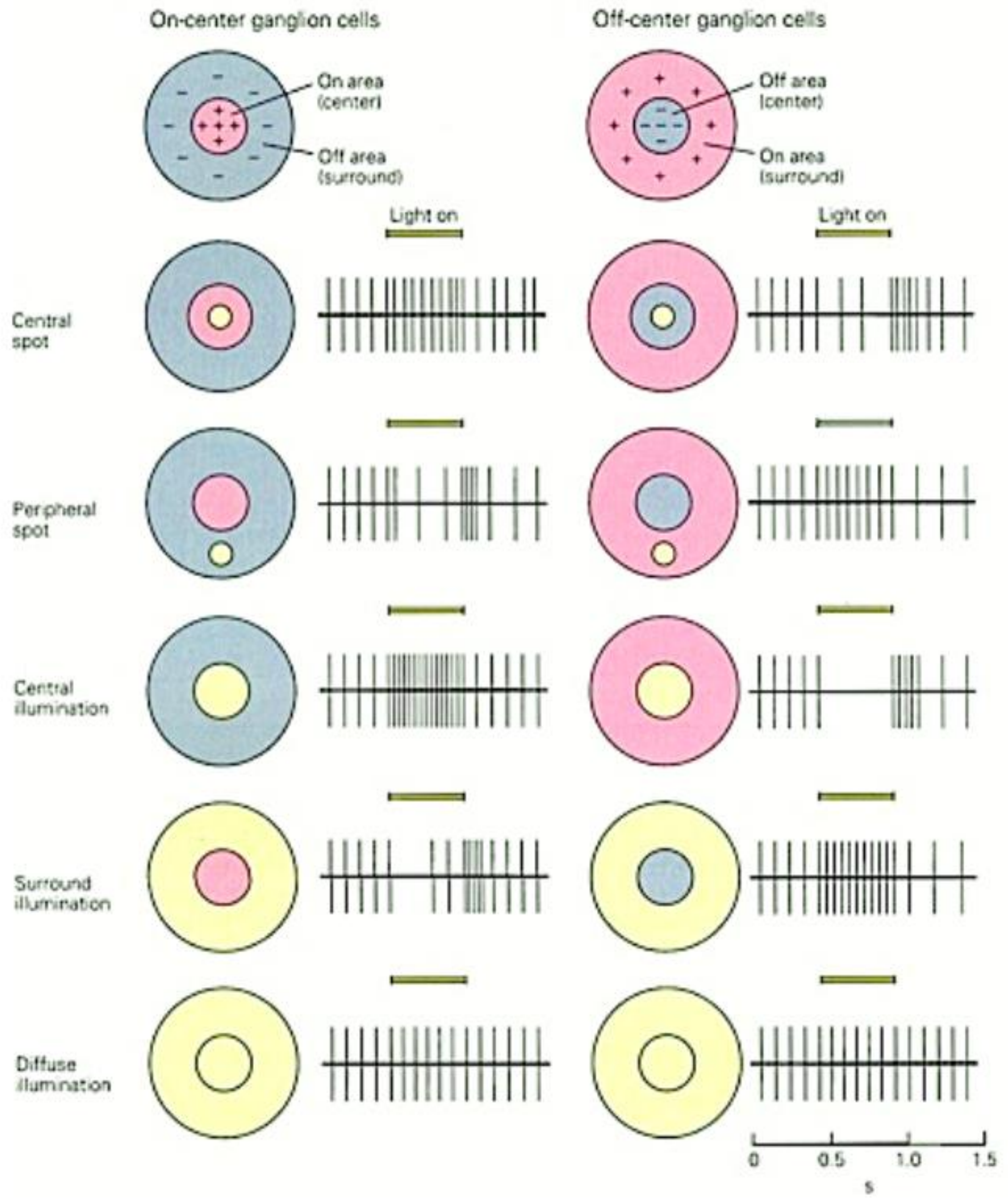
Neural response



Receptive fields

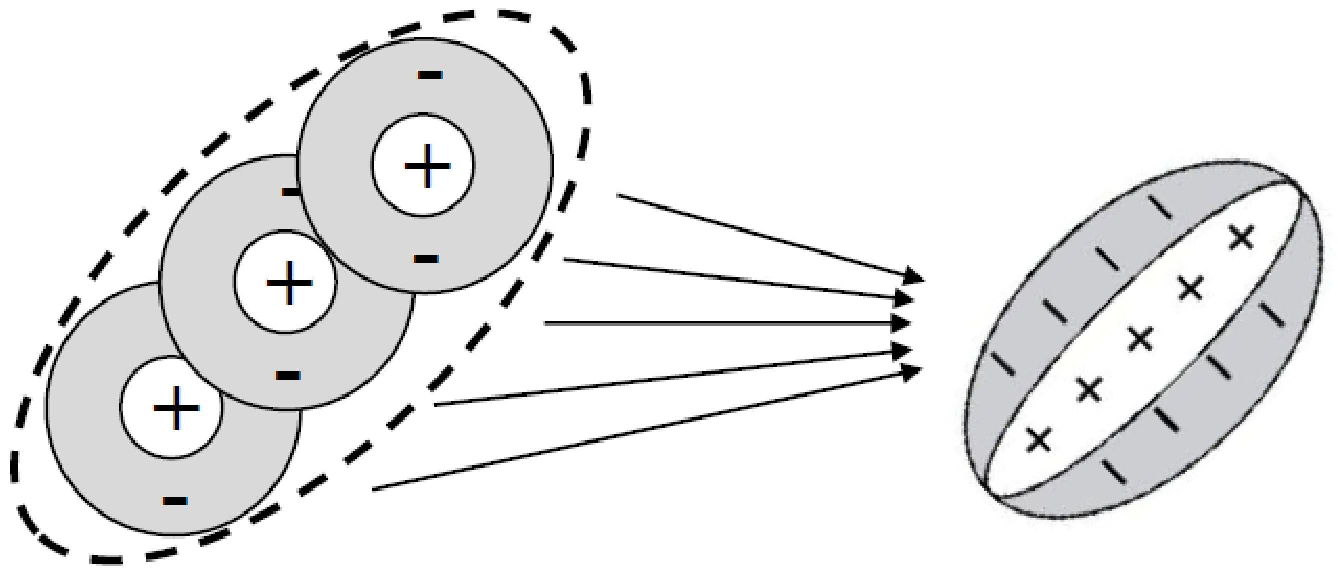
The retinal circuit implements receptive fields to process the image.

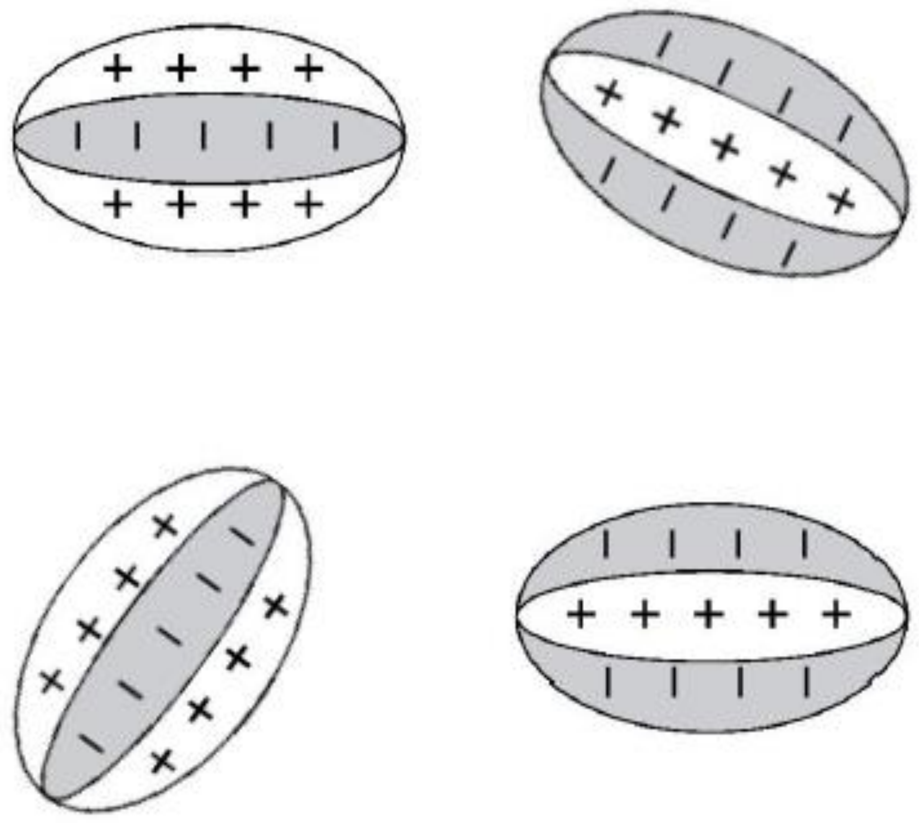




Receptive fields

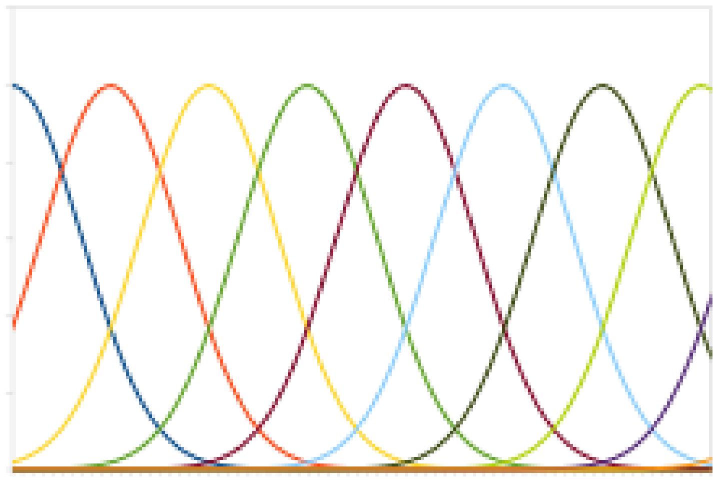
Receptive fields from the retina are in turn used to create oriented receptive fields in the visual cortex.

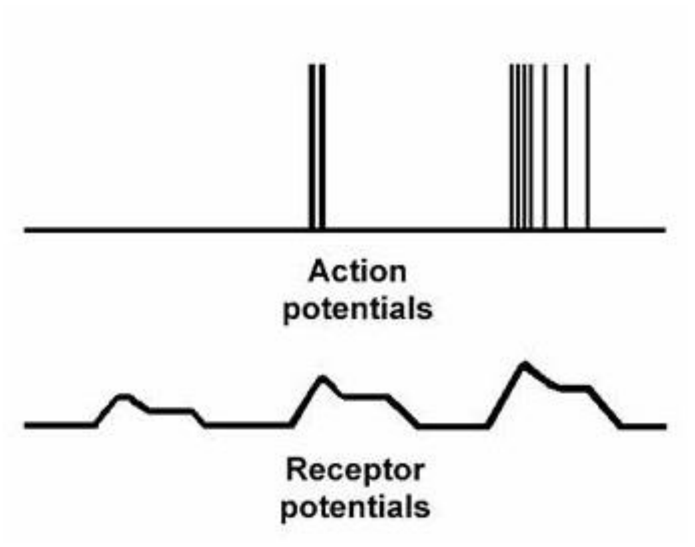




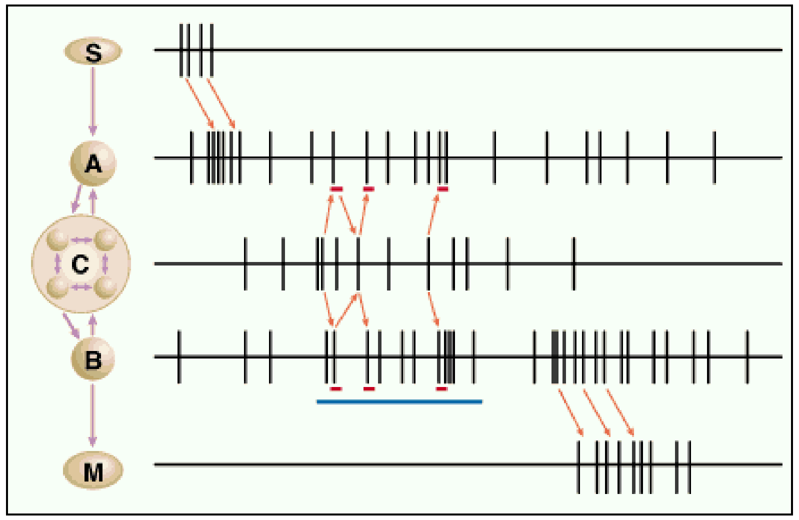
Neural coding

Each sensory input has its own dedicated brain areas that encode the information received. Different types of encoding are being used in the brain. The most well-understood are the following three:


population coding



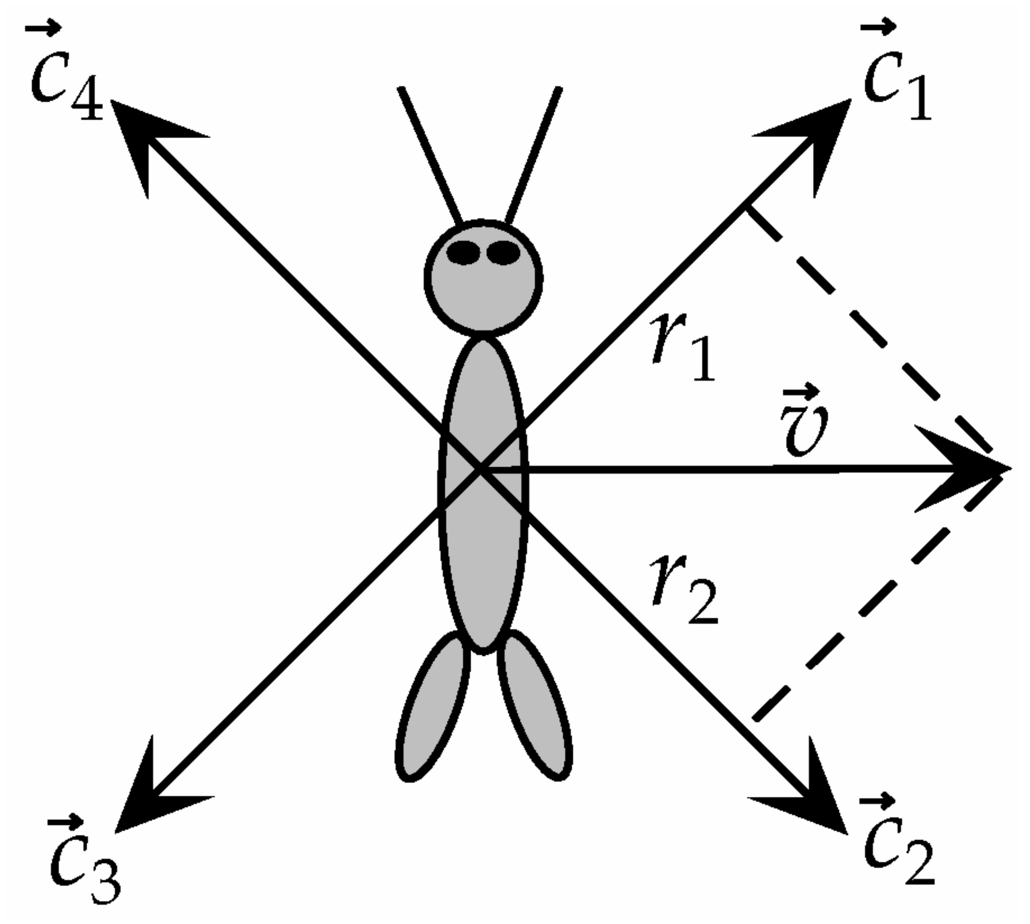
rate coding

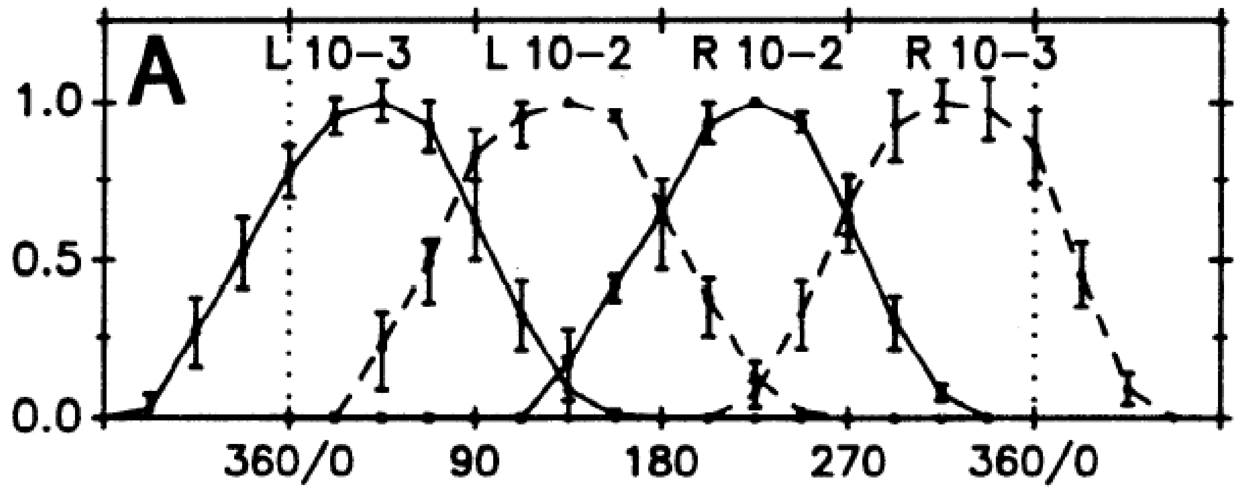


temporal coding

Neural coding

In **population coding**, there is a population of neurons that respond differently to different values of the same sensory information. E.g. cricket cercal cells, some visual areas in the human brain.



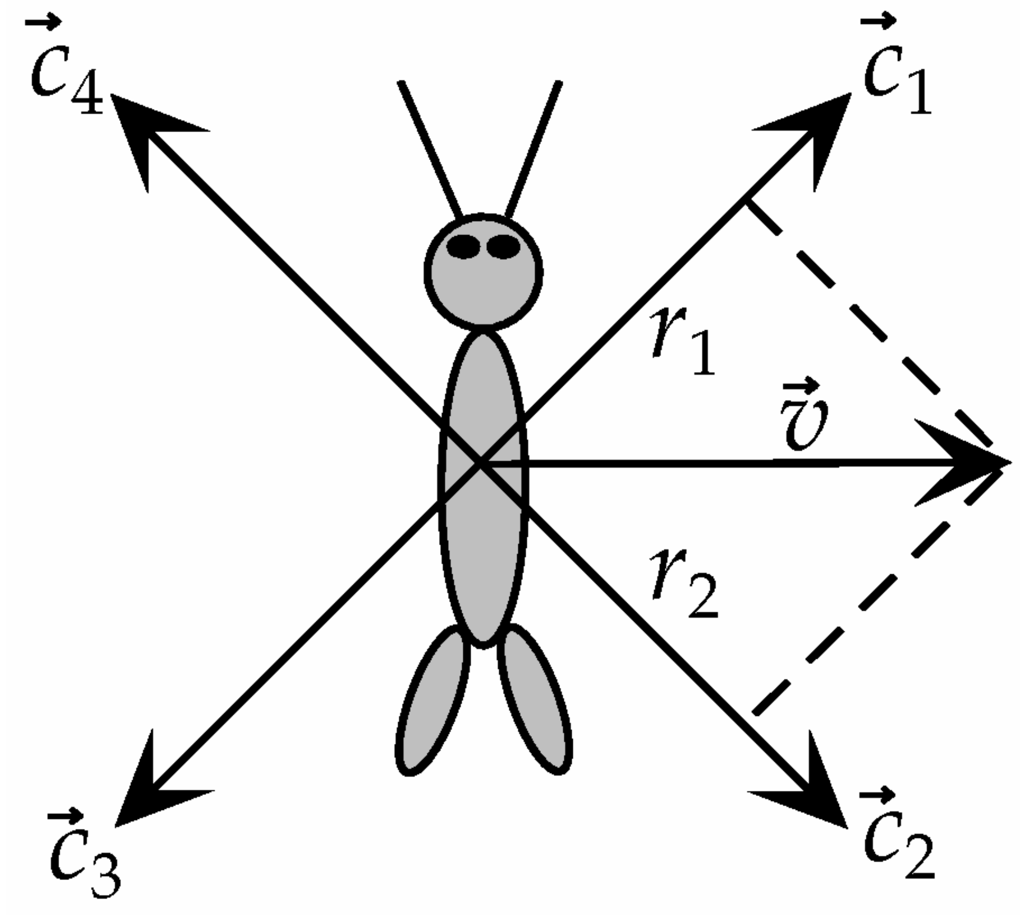


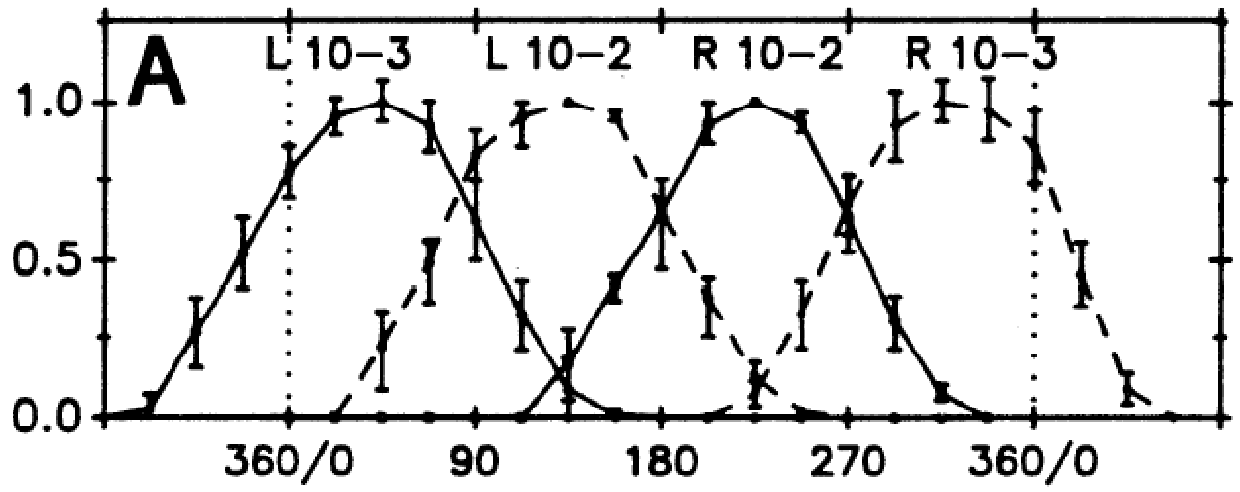
$$r_i = \vec{v} \cdot \overrightarrow{c_i}$$

The resulting value *r* is called the **population vector**.

Neural coding

In **population coding**, there is a population of neurons that respond differently to different values of the same sensory information. E.g. cricket cercal cells, some visual areas in the human brain.





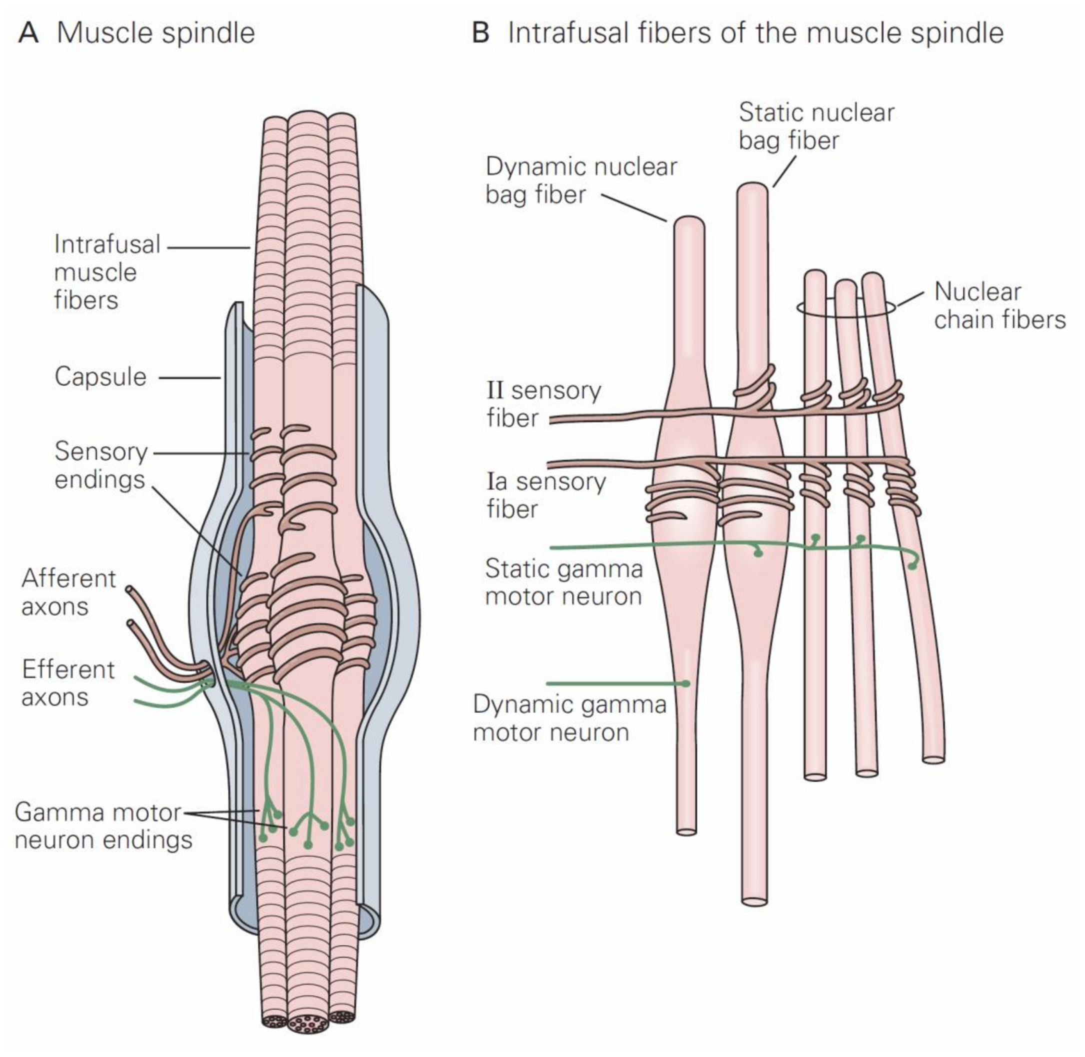
$$r_i = \vec{v} \cdot \overrightarrow{c_i}$$

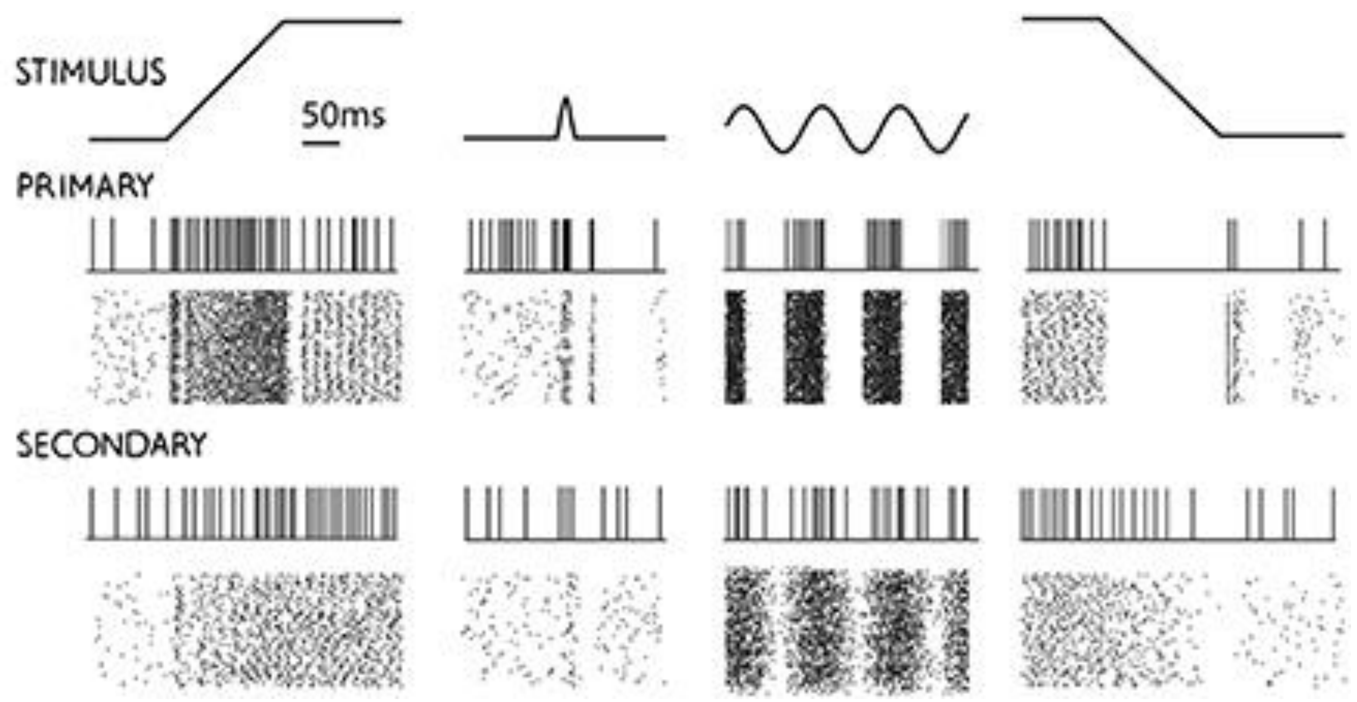
The resulting value *r* is called the **population vector**.

Is the representation efficient?
Aren't c_1 and c_2 enough?

Neural coding

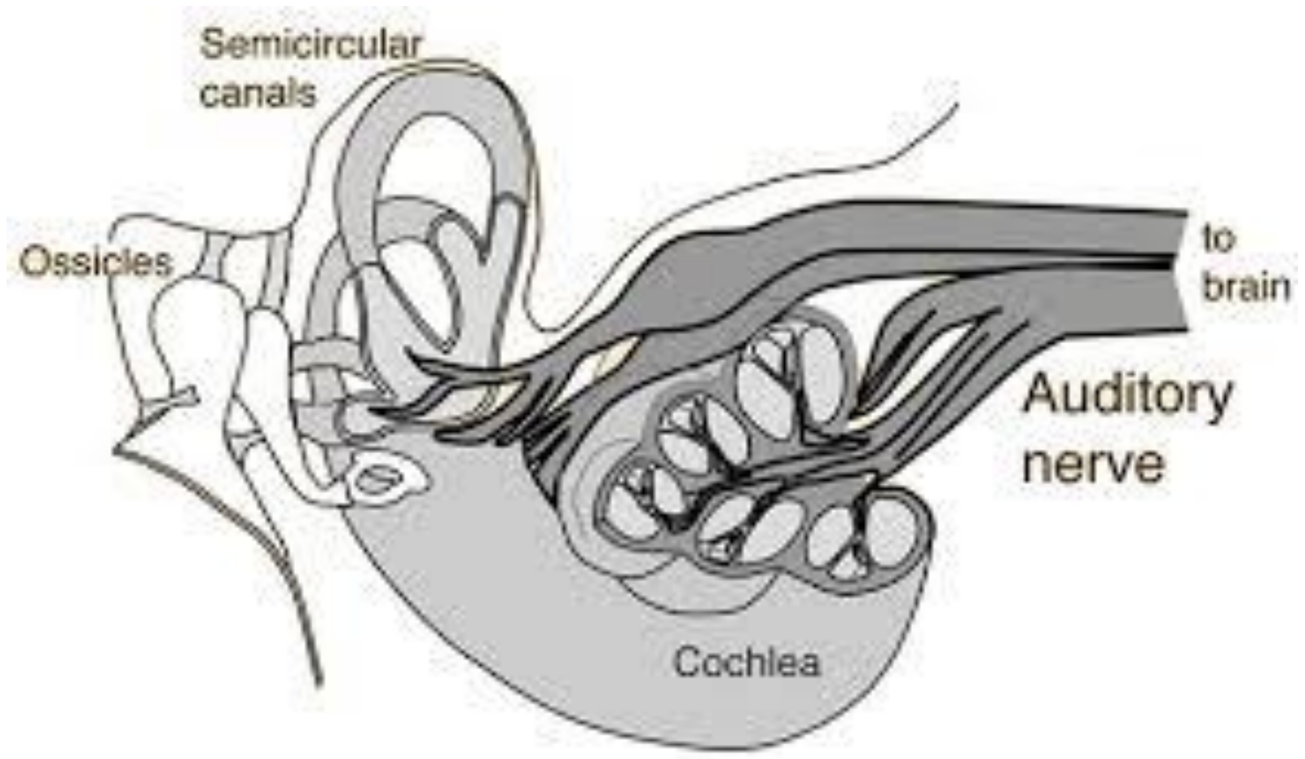
In **rate coding**, all the information is encoded by directly translating it into firing rates. Thus, all neurons in the same population respond in the same manner to the same stimulus. This is common in many sensory afferents, e.g. mammalian muscle spindles.

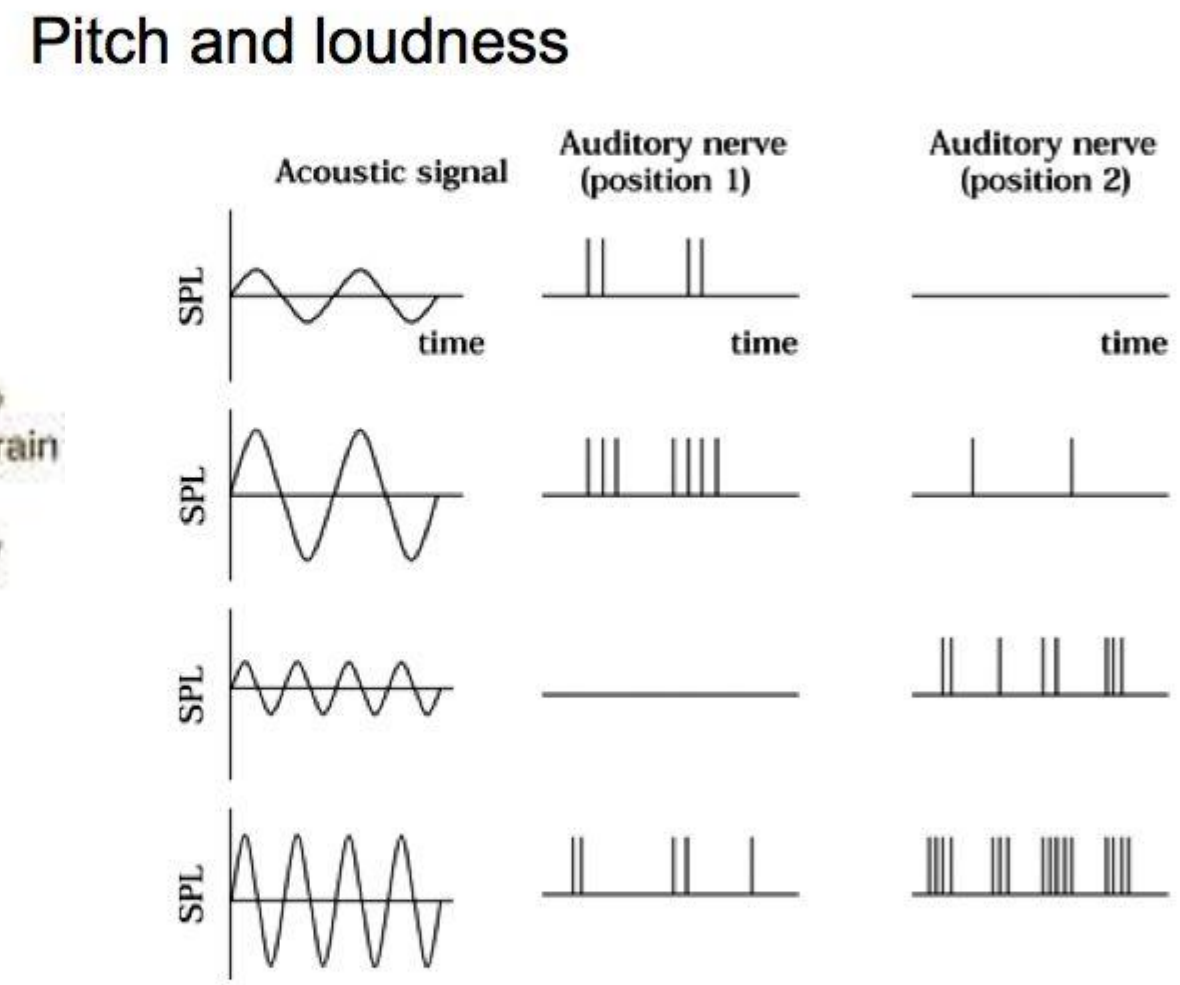




Neural coding

A more complex encoding mechanism is **temporal coding**, where absolute or relative spike times are used. There are evidence for this kind of encoding in the auditory and gustative systems.



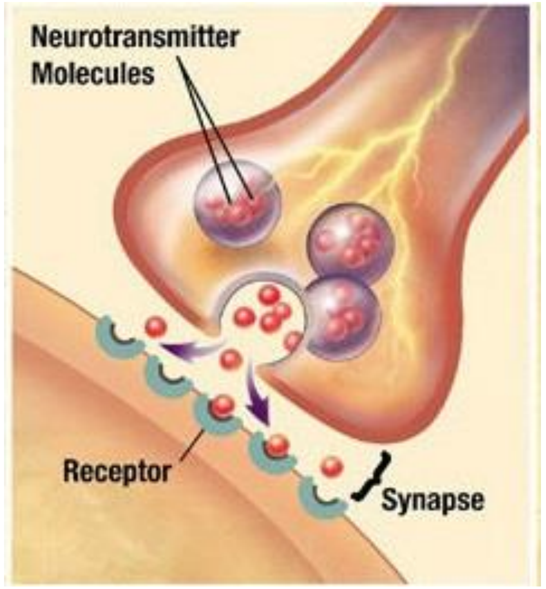


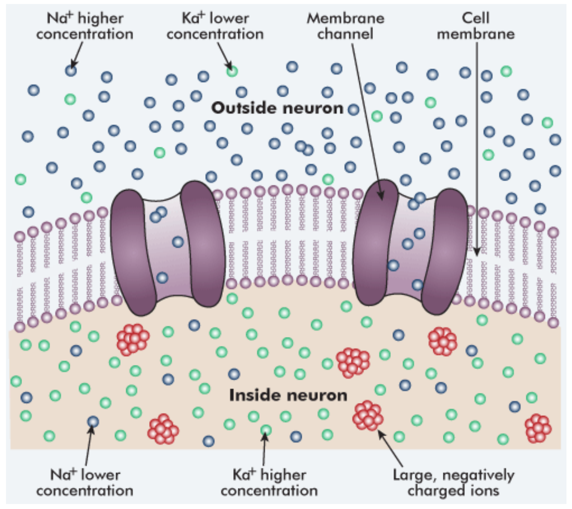
Outline

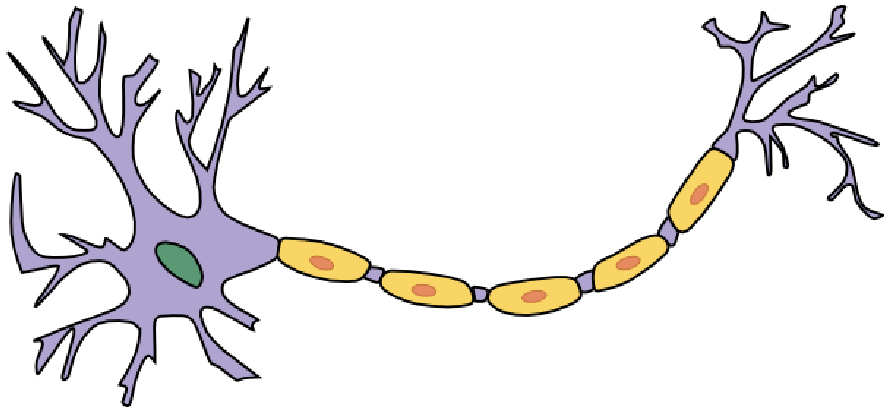
Neuron abstractions

In order to simulate the behaviour of neural circuits we have to model the neuron dynamics.

Thus, we have to translate neurophysiologic properties into equations that we can implement.







Abstract neuron models

- Rate-based
- Point neuron
- Detailed neuron

Detailed neural abstraction

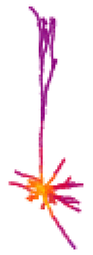
In these kind of models every aspect of the cell morphology is taken into account: diameter of the soma, length of the axons, position of synapses on the dendrites, distribution of ionic channels, neurotransmitter types, etc…

Pros:

- very accurate
- can model any aspect of neural activity

Cons:

- much knowledge is needed to model networks
- simulation times are high



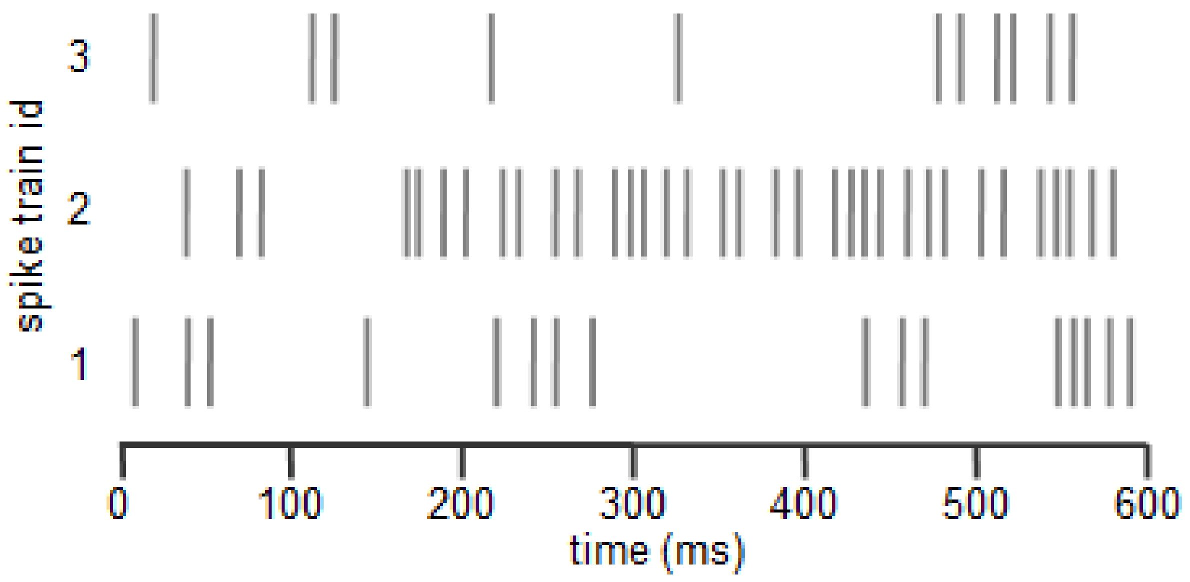
Some detailed neural simulators exist, i.e. NEURON (www.neuron.yale.edu/neuron).

Too little abstraction!

Rate-based abstractions

Each neuron produces spikes with a mean firing rate (in a time interval).

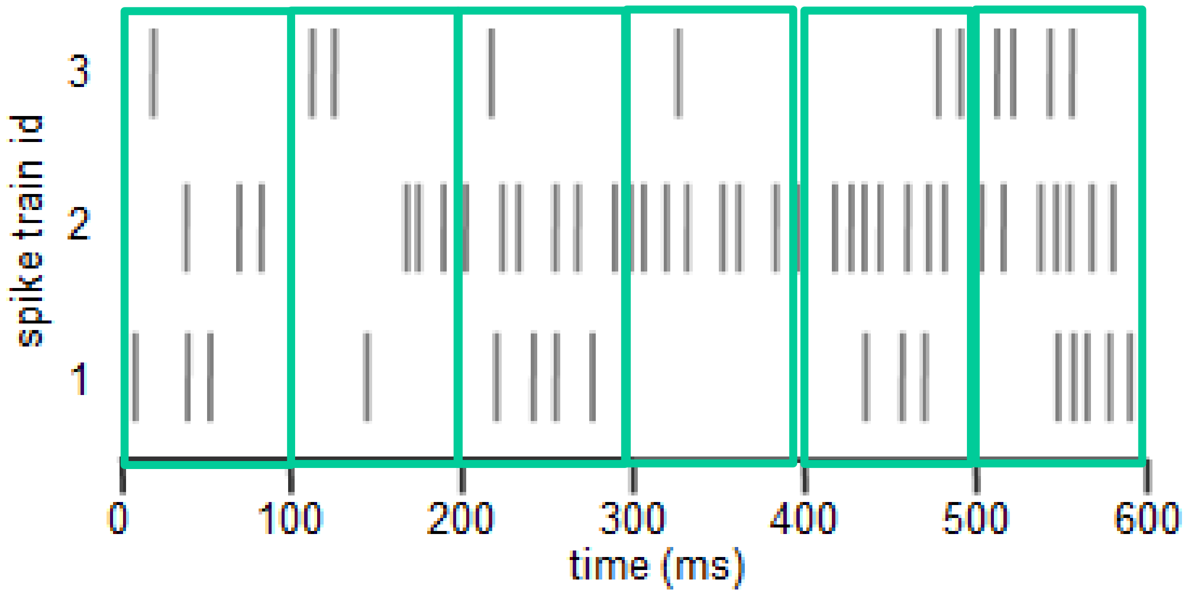
We can sample the firing rate by dividing spikes into bags:



Rate-based abstractions

Each neuron produces spikes with a mean firing rate (in a time interval).

We can sample the firing rate by dividing spikes into bags:



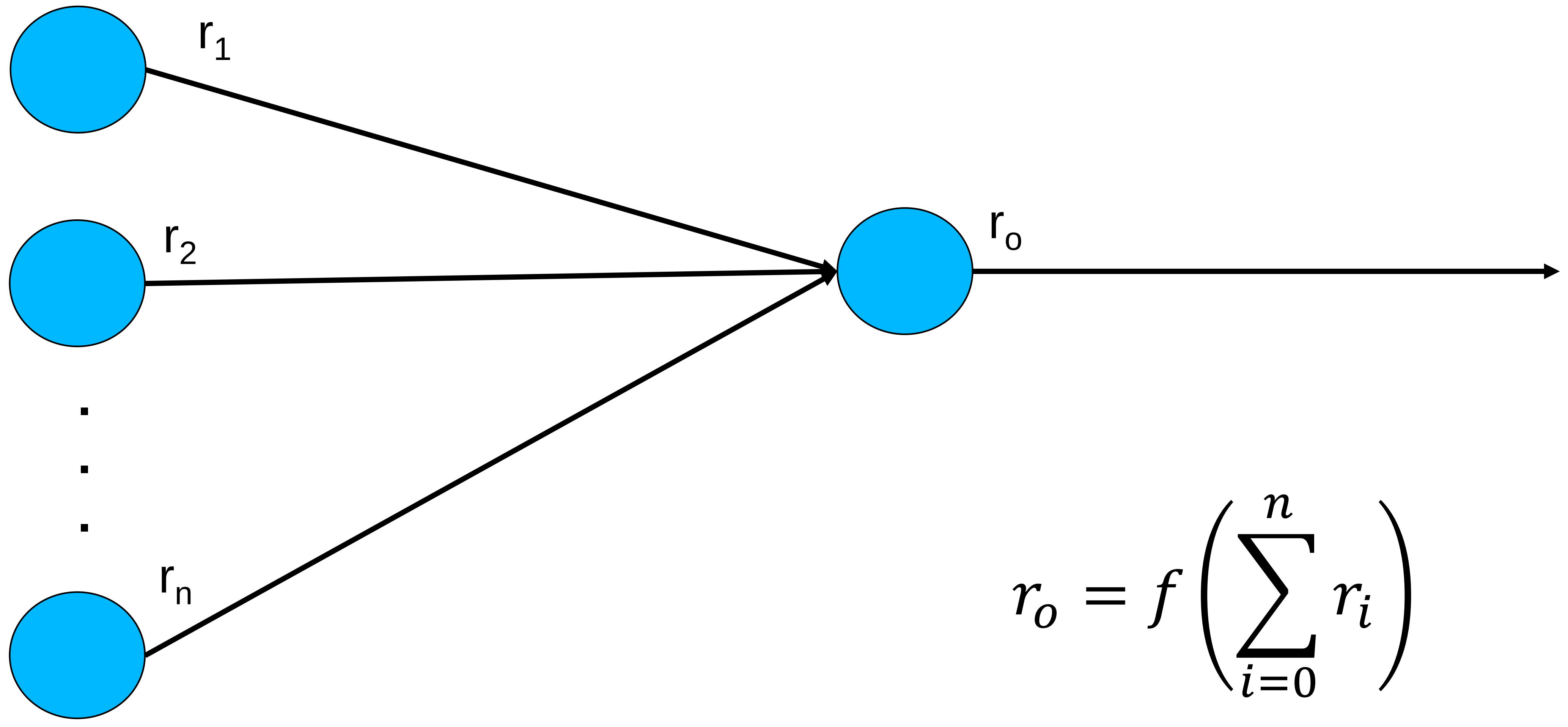
10	20	10	10	20	40
30	30	60	80	70	70
30	10	40	0	30	50

By doing so, we are:

- discretizing time
- forgetting about single action potential events

Rate-based abstractions

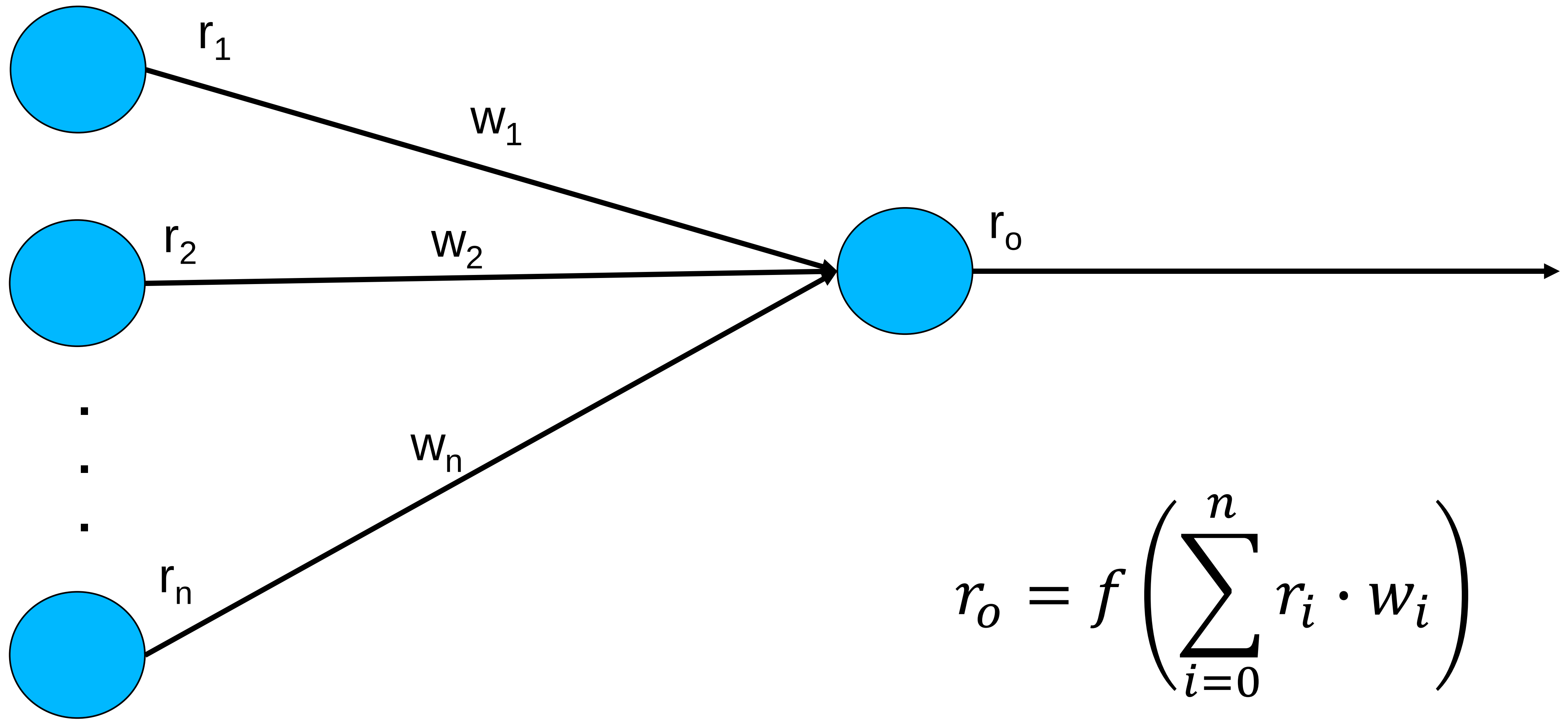
Activity of a post-synaptic neuron can be computed as a function of the rates of pre-synaptic neurons.



$$r_o = f\left(\sum_{i=0}^{n} r_i\right)$$

Rate-based abstractions

What about synapses? We can add weights on the connections.



$$r_o = f\left(\sum_{i=0}^{n} r_i \cdot w_i\right)$$

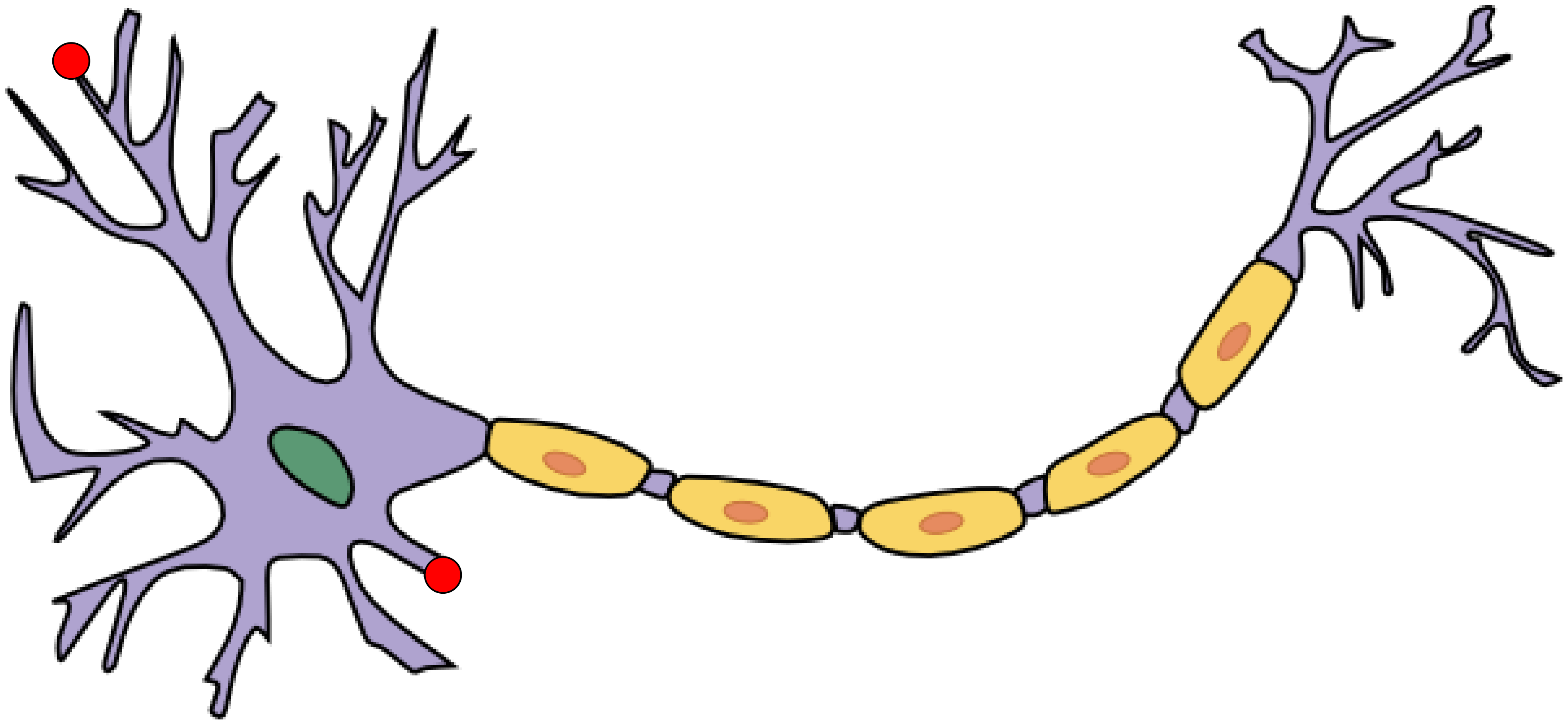
→ Rosenblatt's perceptron and Artificial Neural Networks.

Too much abstraction!

Point neuron abstractions

Why are these called point-neuron abstractions?

Because we do not take into account the neuron morphology. Each neuron is dimensionless and currents propagate instantaneously from all the receiving synapses.



Point neuron abstractions – neuron models

The neuron electrical properties can be described through electrical circuits:

- the lipidic membrane acts as a capacitor (C_m);
- all PSP can be summed up and represented as an external current generator (I_{ext}).

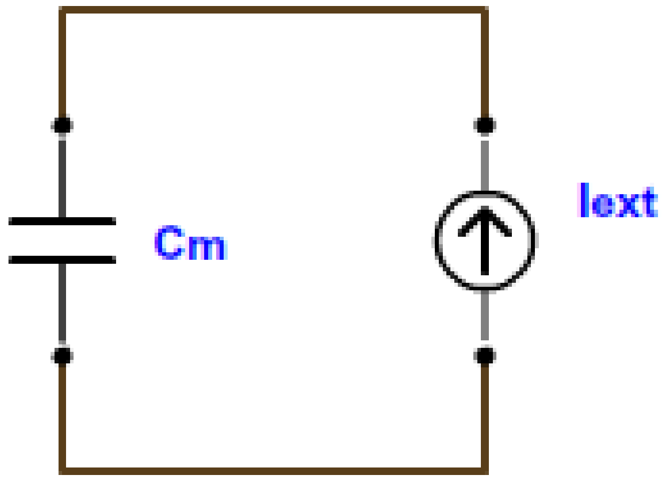
We are interested in the voltage between the two termination of the capacitor (membrane potential, V_m) and we also add the action potential rule:

If $V_m > V_{th}$ then V_m resets to V_{reset} and a spike is emitted.

Point neuron abstractions – neuron models

A first circuit representing neural activity is the **Integrate and fire** model (IAF).

Kirchhoff's law: $I_C(t) = I_{ext}(t)$



Point neuron abstractions – neuron models

A first circuit representing neural activity is the **Integrate and fire** model (IAF).

Kirchhoff's law:

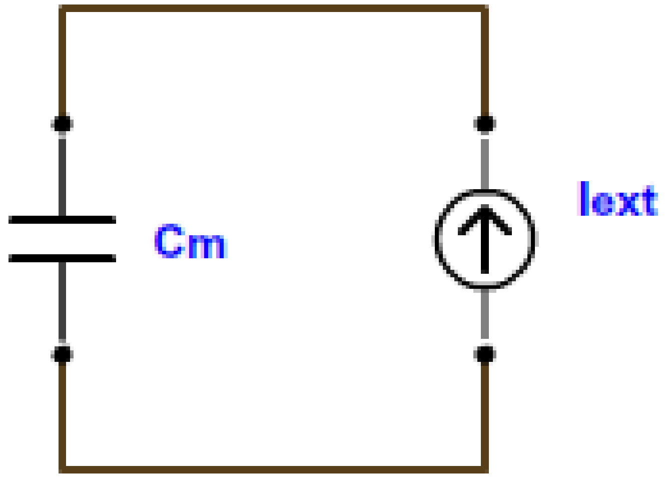
$$I_C(t) = I_{ext}(t)$$

$$Q(t) = C_m V_m(t)$$

$$\downarrow$$

By deriving the law of capacitance:

$$I_C(t) = C_m \frac{dV_m(t)}{dt}$$

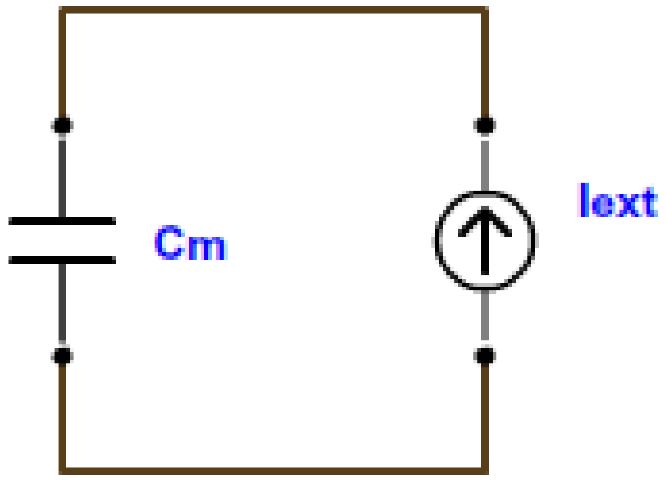


Point neuron abstractions – neuron models

A first circuit representing neural activity is the **Integrate and fire** model (IAF).

Kirchhoff's law:

$$I_C(t) = I_{ext}(t)$$



By deriving the law of capacitance:

$$Q(t) = C_m V_m(t)$$

$$\downarrow$$

$$I_C(t) = C_m \frac{dV_m(t)}{dt}$$

Thus, we obtain:

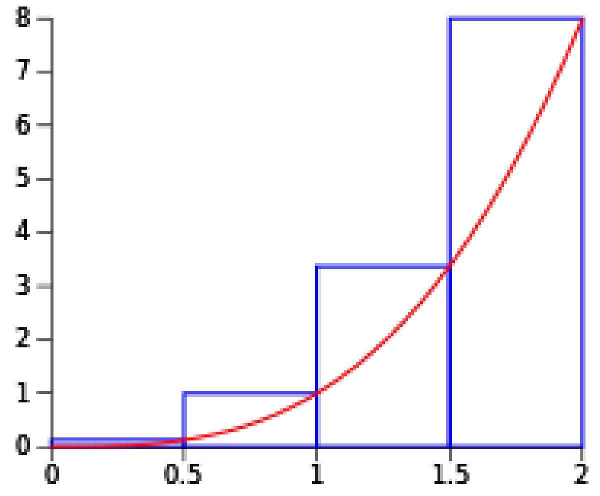
$$\boxed{\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m}}$$

Point neuron abstractions – simulation loop (I)

We can employ the differential equation to compute the dynamics of the membrane in a simulation loop, by discretizing time in small intervals.

```
T = 2000.0 // total simulation time, ms
time = 0.0
V = 0.0
dt = 1.0  // simulation step, ms

while (time < T) {

  Iext = sum_external_currents()

  dVm = membrane_update(Iext)

  V += dVm * dt  // discrete integration

  if (V > Vth) emit_spike();

  time += dt

}
```

$$\boxed{\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m}}$$



Let's try it out!

Point neuron abstractions – neuron models

Neurons have the **refractory period**, that must be taken into account for an accurate simulation. Otherwise, the firing rate will rise indefinitely.

without: $r(I) = \frac{I}{C_m(V_{th} - V_{reset})}$ $\lim_{I \to +\infty} r(I) = +\infty$

Point neuron abstractions – neuron models

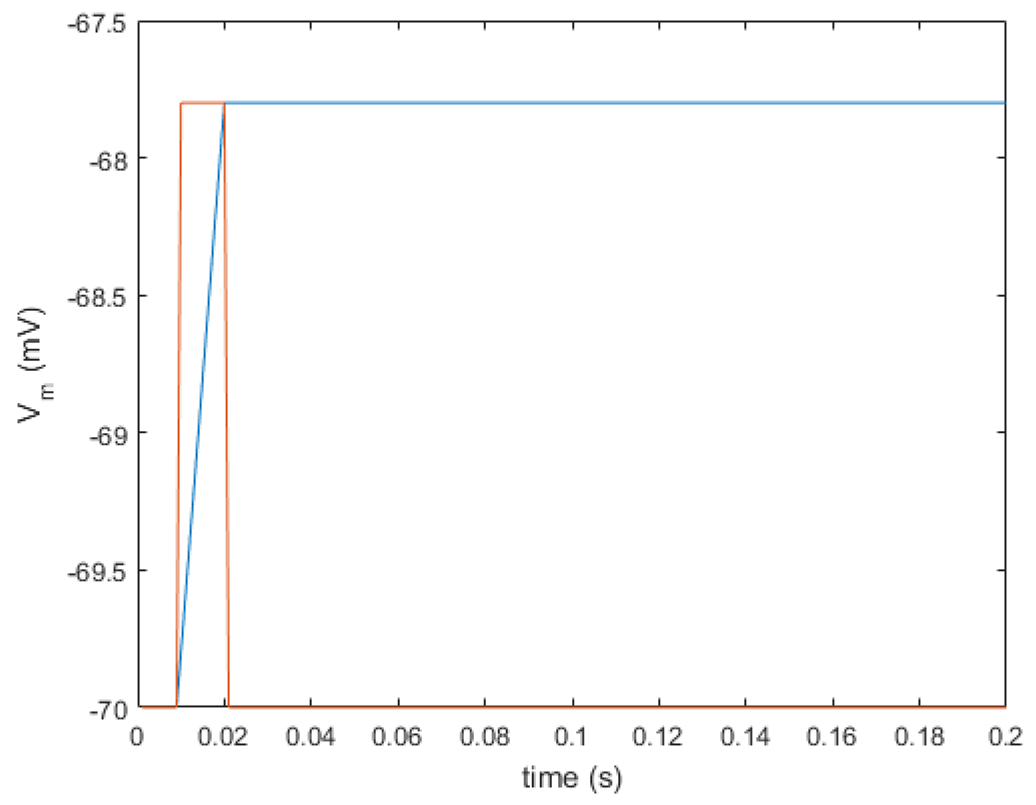
Neurons have the **refractory period**, that must be taken into account for an accurate simulation. Otherwise, the firing rate will rise indefinitely.

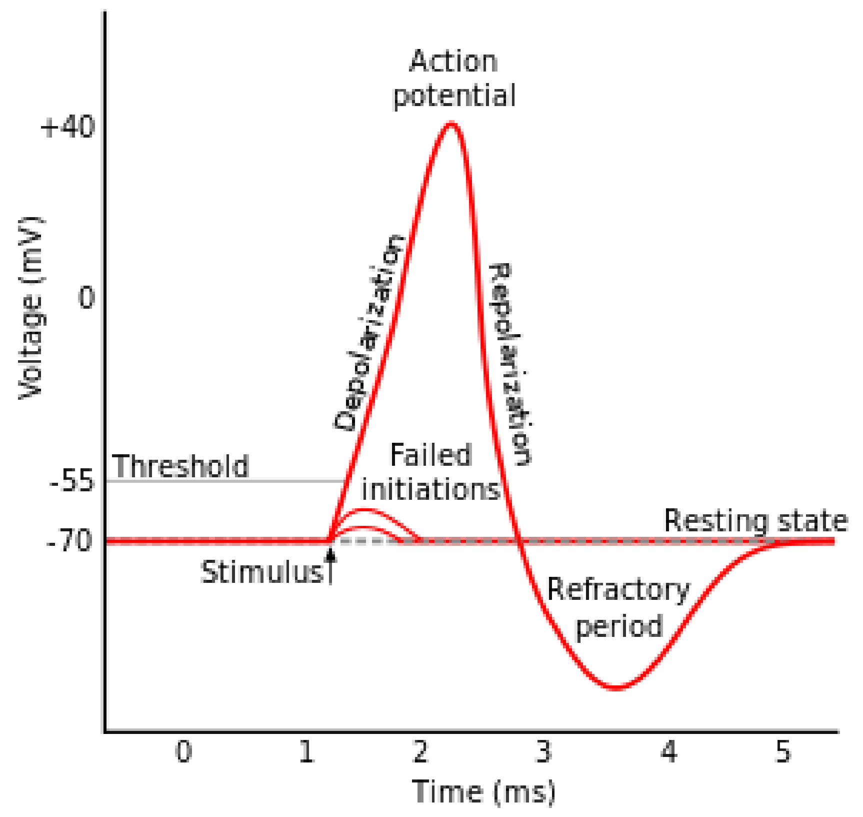
without: $$r(I) = \frac{I}{C_m(V_{th} - V_{reset})} \qquad \lim_{I \to +\infty} r(I) = +\infty$$

with: $$r(I) = \frac{I}{C_m(V_{th} - V_{reset}) + t_{ref} I} \qquad \lim_{I \to +\infty} r(I) = \frac{1}{t_{ref}}$$

Point neuron abstractions – neuron models

In the IAF model, the membrane continues to keep the gained potential, even if there is no external input current and the spike threshold is not reached. This is not true for the biological neuron.

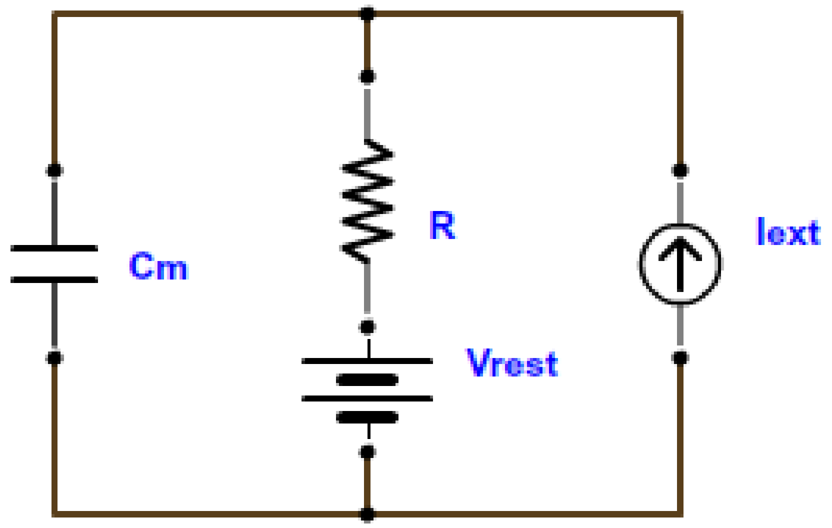




Point neuron abstractions – neuron models

The **Leaky integrate and fire** model (LIAF) adds a resistance in the circuit in order to model the leakage of charge. Moreover, a battery is added to represent the equilibrium potential of the cell membrane.

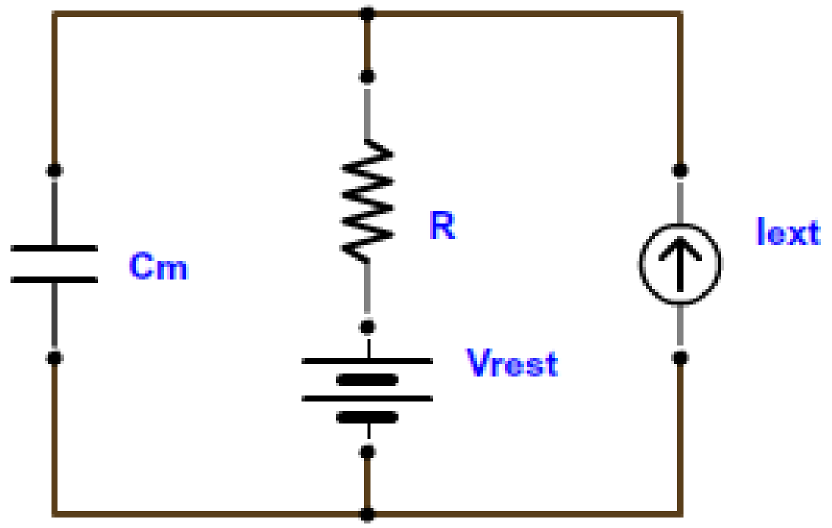
Kirchhoff's law: $I_C(t) + I_R(t) = I_{ext}(t)$



Point neuron abstractions – neuron models

The **Leaky integrate and fire** model (LIAF) adds a resistance in the circuit in order to model the leakage of charge. Moreover, a battery is added to represent the equilibrium potential of the cell membrane.

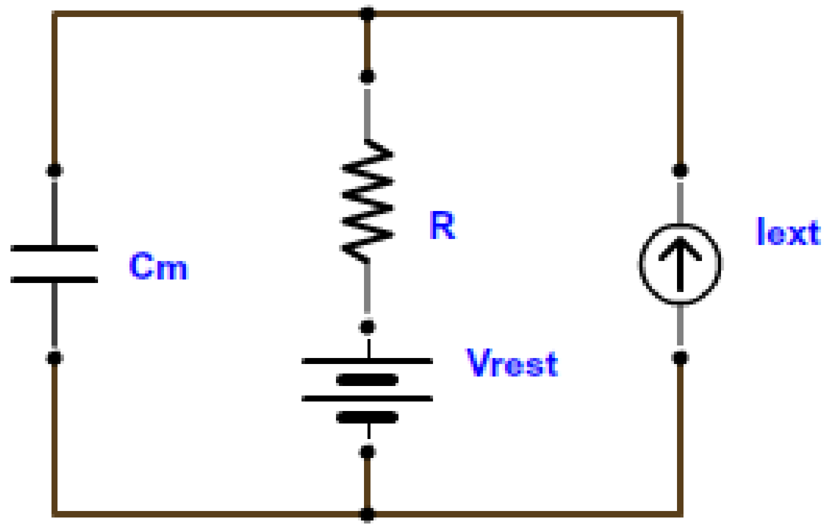
Kirchhoff's law: $I_C(t) + I_R(t) = I_{ext}(t)$



Ohm's law: $I_R(t) = \frac{(V_m(t) - V_{rest})}{R}$

Point neuron abstractions – neuron models

The **Leaky integrate and fire** model (LIAF) adds a resistance in the circuit in order to model the leakage of charge. Moreover, a battery is added to represent the equilibrium potential of the cell membrane.



Kirchhoff's law: $$I_C(t) + I_R(t) = I_{ext}(t)$$

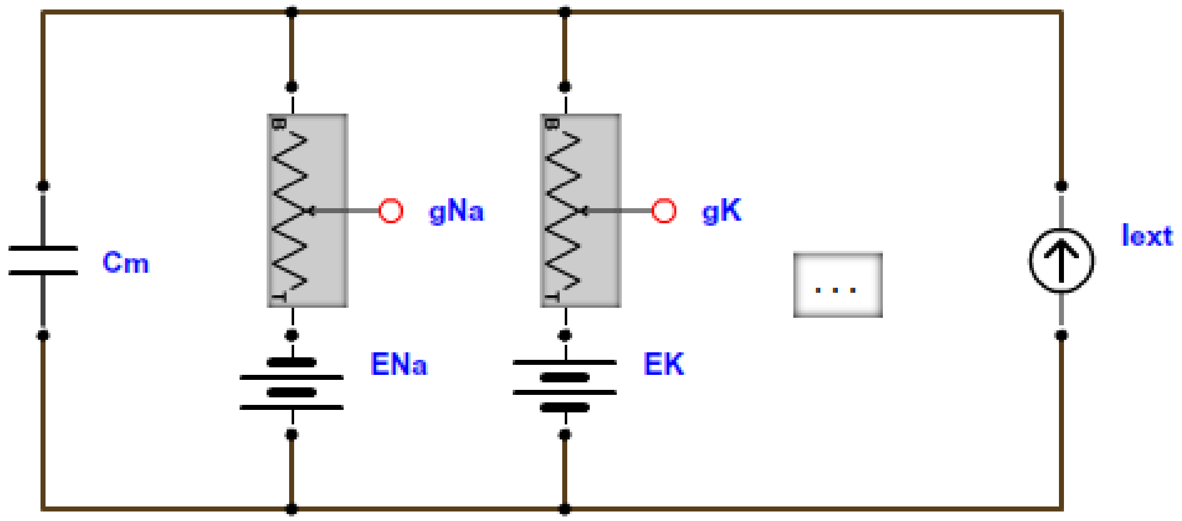
Ohm's law: $$I_R(t) = \frac{(V_m(t) - V_{rest})}{R}$$

Thus, we obtain: $$\boxed{\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m} - \frac{(V_m(t) - V_{rest})}{C_m R}}$$

Point neuron abstractions – neuron models

There are many others neuron models:

Hodgkin–Huxley: each ionic channel is modelled as a resistance-battery parallel circuit, with a probabilistic conductance.

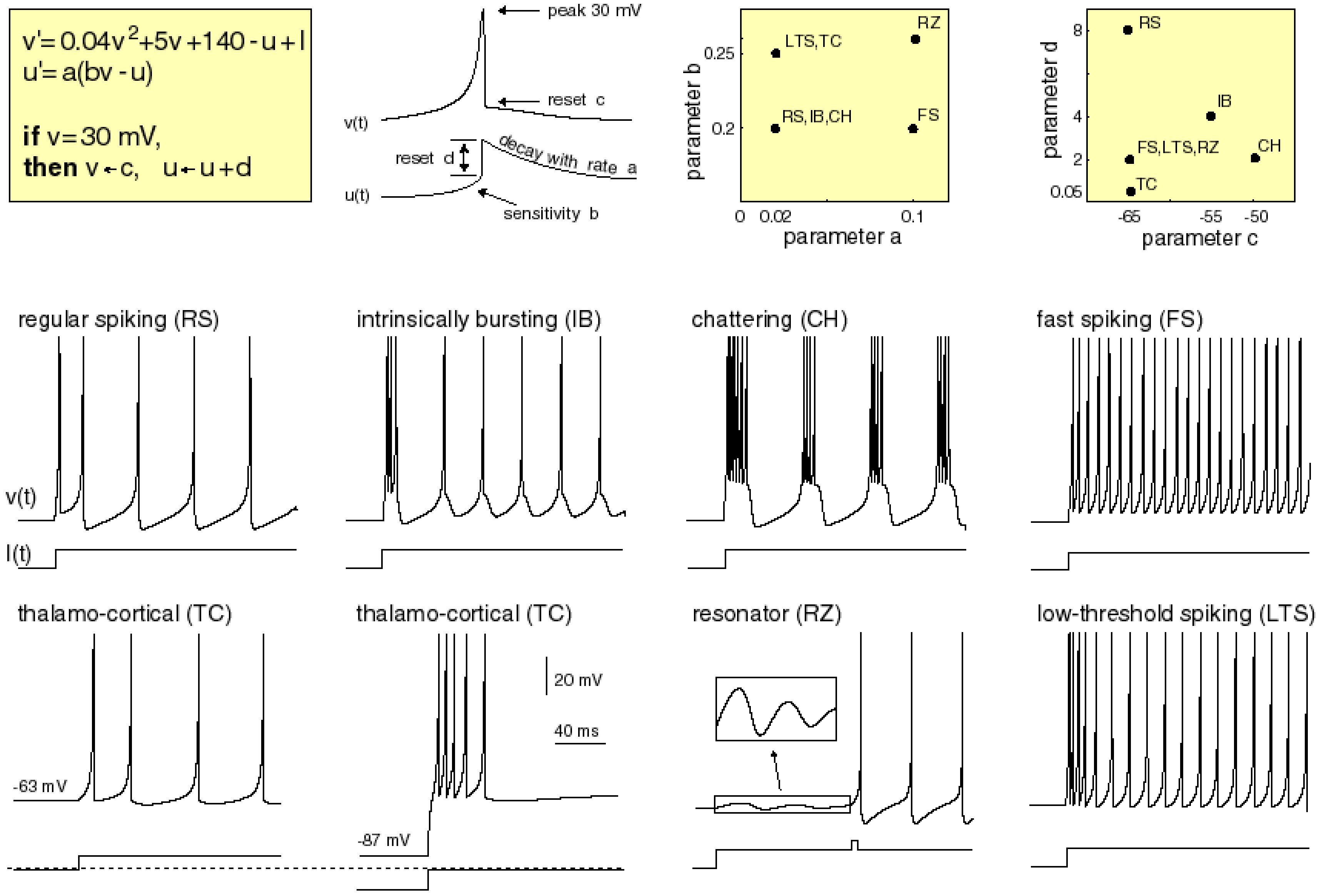


$$\boxed{\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m} - \frac{1}{C_m}\sum_i g_i(V_m(t) - E_i)}$$

Point neuron abstractions – neuron models

There are many others neuron models:

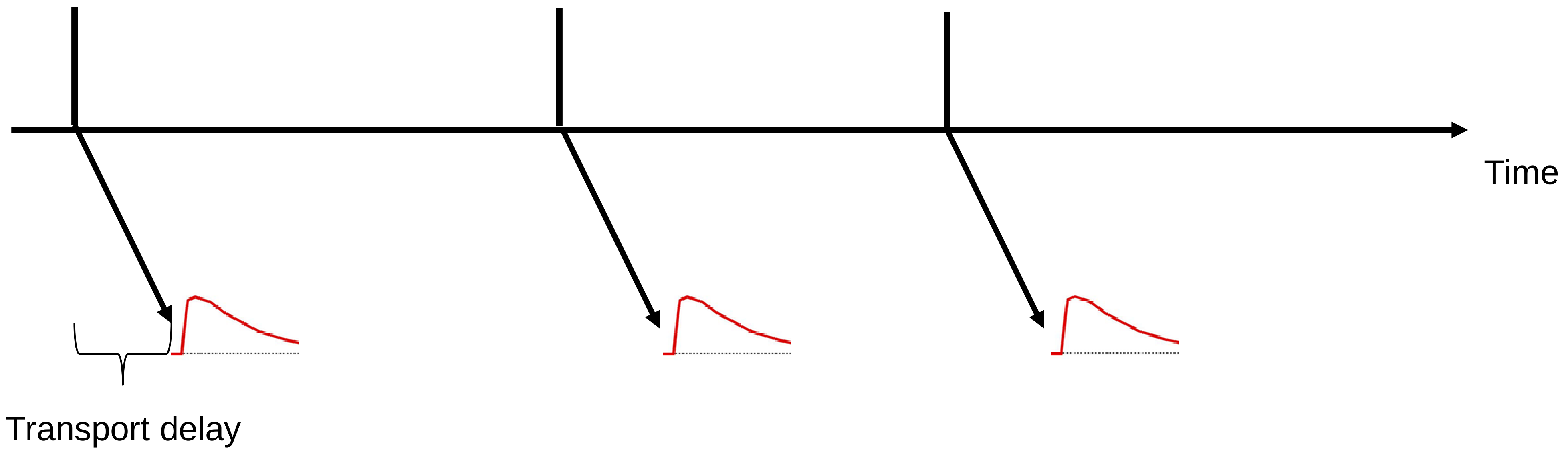
Izhikevich: two differential equations can model many different neuron behaviours.



www.izhikevich.com

Point neuron abstractions – synapses models

Each action potential is transmitted as an event to all postsynaptic neurons connected, after a transmission delay (travel time on the axon). When such event is received a proper EPSC or IPSC is generated and added to the total input current.



Amongst the most common PSC types there is the **alpha-shaped** one:

$$I(t) = \frac{t}{\tau_s} e^{-\frac{t}{\tau_s}}$$

Point neuron abstractions – synapses models

Each synapse has a weight that has two roles:

1. distinguishing between inhibitory and excitatory synapses by being negative or positive;
2. representing the strength of the connection between the two neurons.

Synaptic weights can be changed via rules implementing STDP, for example:

$$\Delta w_{ij} = \sum_{f} \sum_{n} W(t_i^f - t_j^n) \qquad W(x) = \begin{cases} A_+ e^{\left(-\frac{x}{\tau_+}\right)} \, for \; x > 0 \\ -A_- e^{\left(-\frac{x}{\tau_-}\right)} \, for \; x < 0 \end{cases}$$

Point neuron abstractions – simulation loop (II)

Given the previous equations we could in principle create a network simulation loop like the following:

```
while (time < T) {
  foreach (n : neurons) {
    Iext = n.sum_external_currents(n.received_spikes)
    dVm = n.membrane_update(Iext, n.V)
    n.V += dVm * dt
    if (n.V > n.Vth) {
        n.send_spike()
        n.adjust_weights(n.received_spikes)
    }
    time += dt
  }
}
```

Send spike through delayed and weighted connection

Outline

Point neuron simulators - NEST

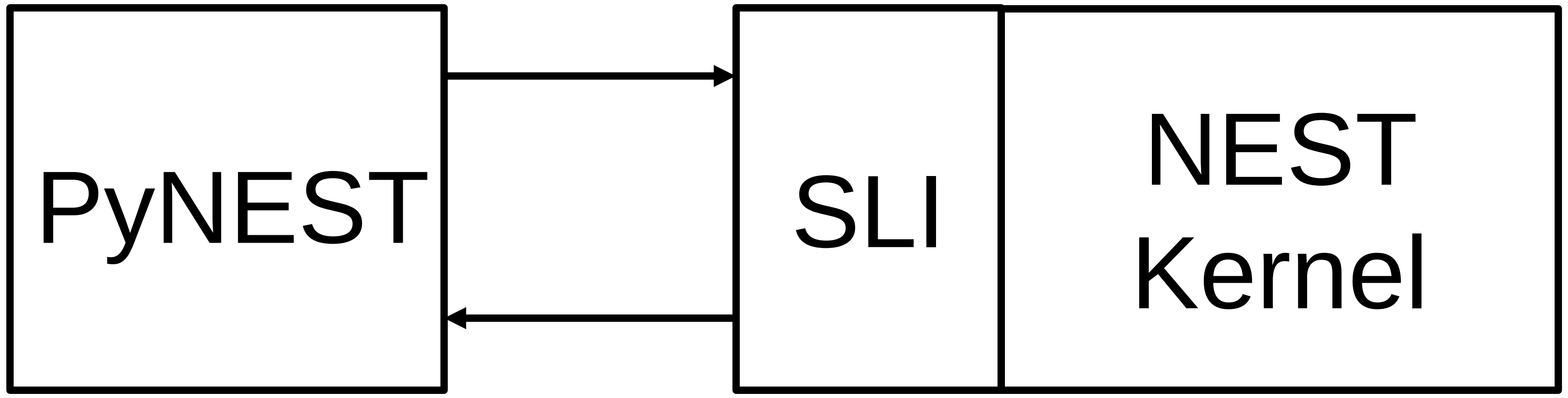
We don't have to implement a whole simulator by ourselves, several already exist!

Among these, a popular choice is NEST (NEural Simulation Tool), an open source spiking neural network simulator developed by the NEST initiative (www.nest-simulator.org). Among its features, there are:

- over 50 neuron models (including LIAF, Hodgkin-Huxley and Izhikevich)
- over 10 synapse models (including STDP)
- minimal dependencies
- open source (GNU GPLv2)
- “easily” extendable

Point neuron simulators - NEST

NEST has a simulation kernel (written in C++) and two layers of interface towards it.



The kernel cannot be directly accessed. In fact, the executable launches the Simulation Language Interpreter to which one can send commands to create the network.

Why PyNEST? Because SLI is basically PostScript!

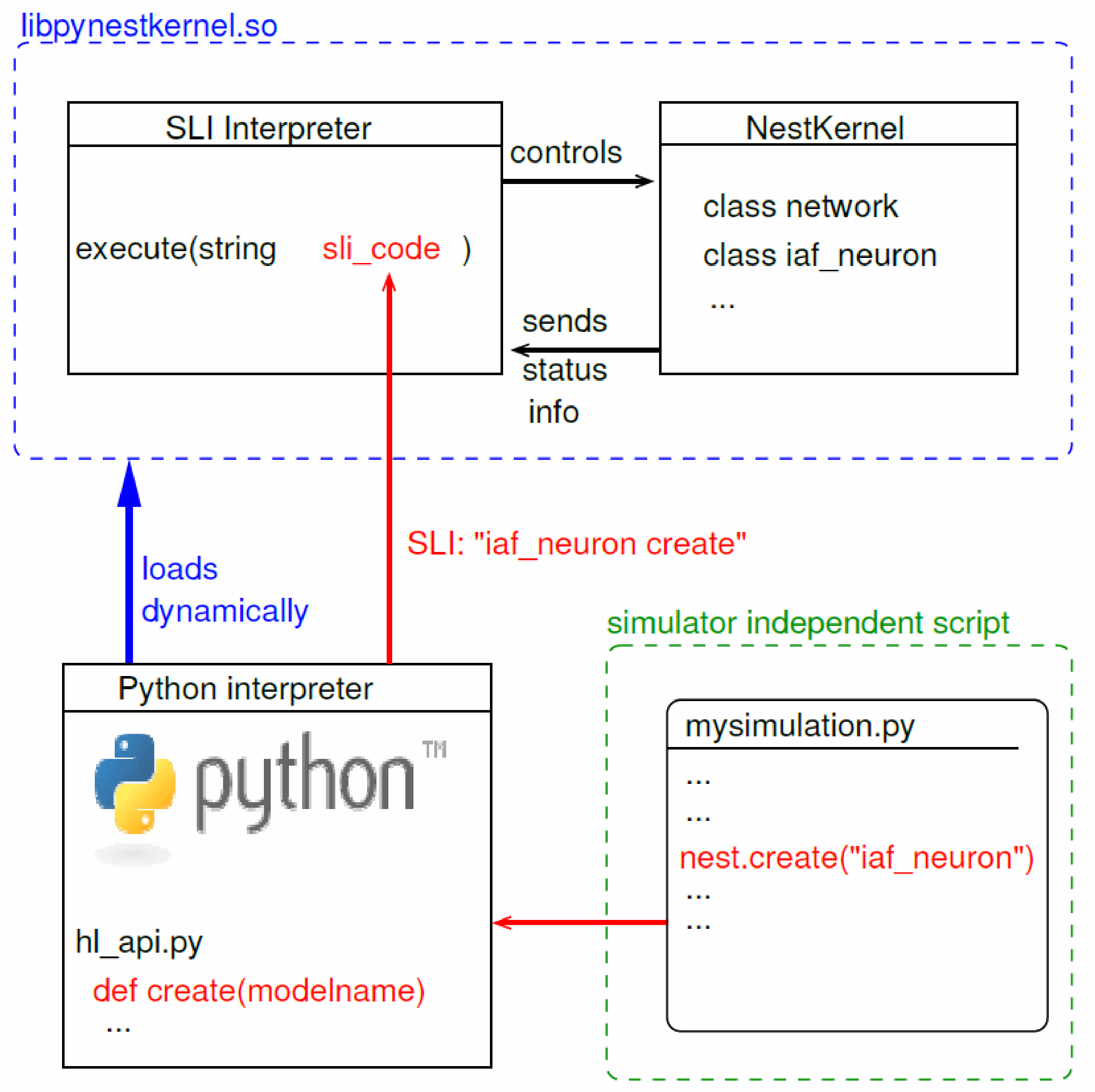
```
/iaf_neuron Create /n Set
/poisson_generator Create /pg Set
pg << /rate 220.0 Hz >> SetStatus
pg n Connect
```

→

```
n = nest.Create('iaf_neuron')
pg = nest.Create('poisson_generator')
nest.SetStatus(pg, {'rate': 220.0})
nest.Connect(pg, n)
```

Point neuron simulators - NEST

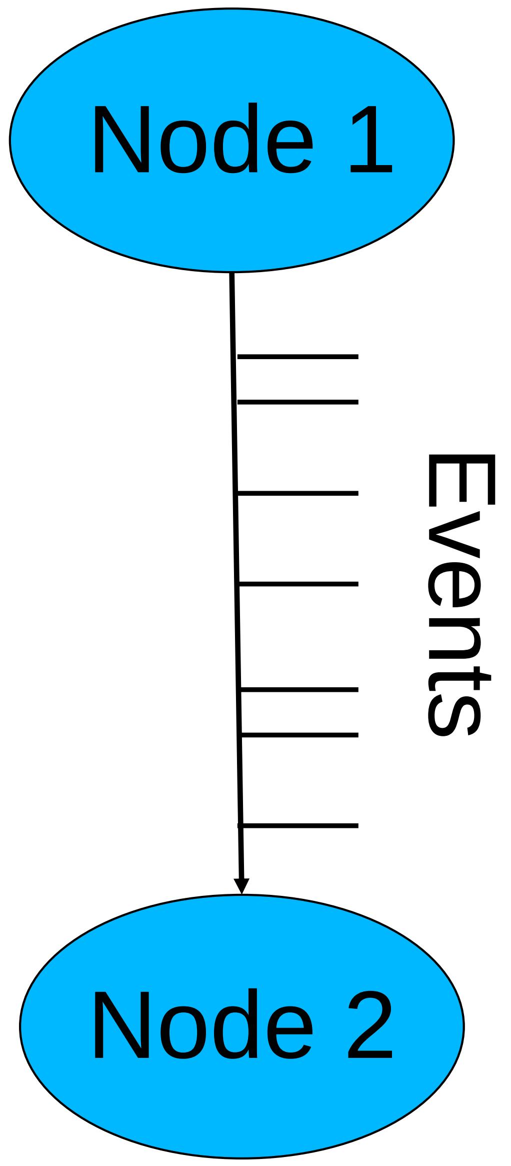
PyNEST provides an usable interface towards SLI.



Point neuron simulators - NEST

A NEST network is a *directed weighted graph*:

- **Nodes**
  - neurons, devices, sub-networks
  - have a dynamic state that changes over time and can be influenced by events
- **Events**
  - pieces of information of a particular type (e.g. spike, voltage or current event)
- **Connections**
  - communication channels for the exchange of events
  - directed (pre to post)
  - weighted (synaptic weights)
  - delayed (delay must be greater than 0!)



Point neuron simulators - NEST

The simulation is discretized into time steps of a certain duration (Δt). The simulation loop works as follows:

1. PSC for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δt

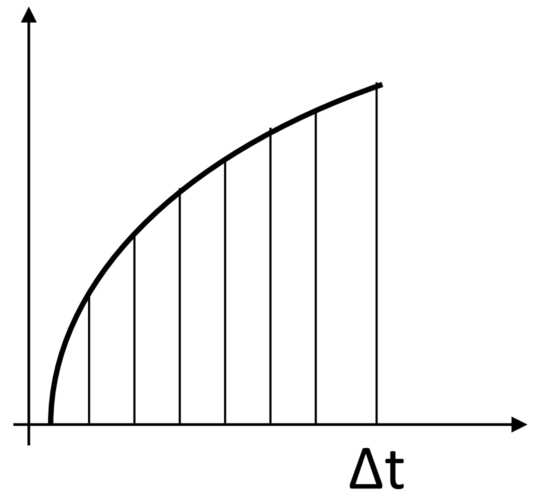
Point neuron simulators - NEST

The simulation is discretized into time steps of a certain duration (Δt). The simulation loop works as follows:

1. PSC for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δt

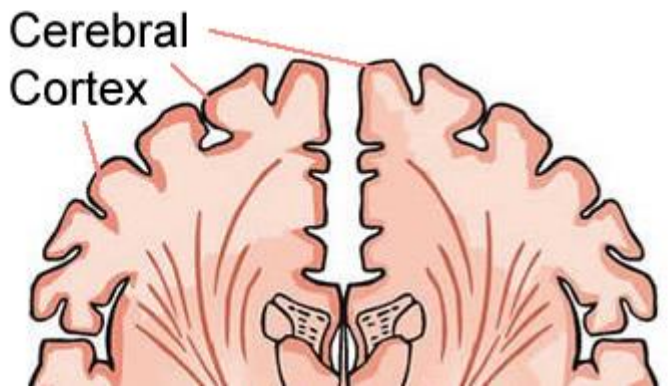
Notes:

- actually 1 and 2 occur inside an inner loop with a time step < Δt!
- delay of connections must be >= Δt
- during the time step, the node is isolated from the rest



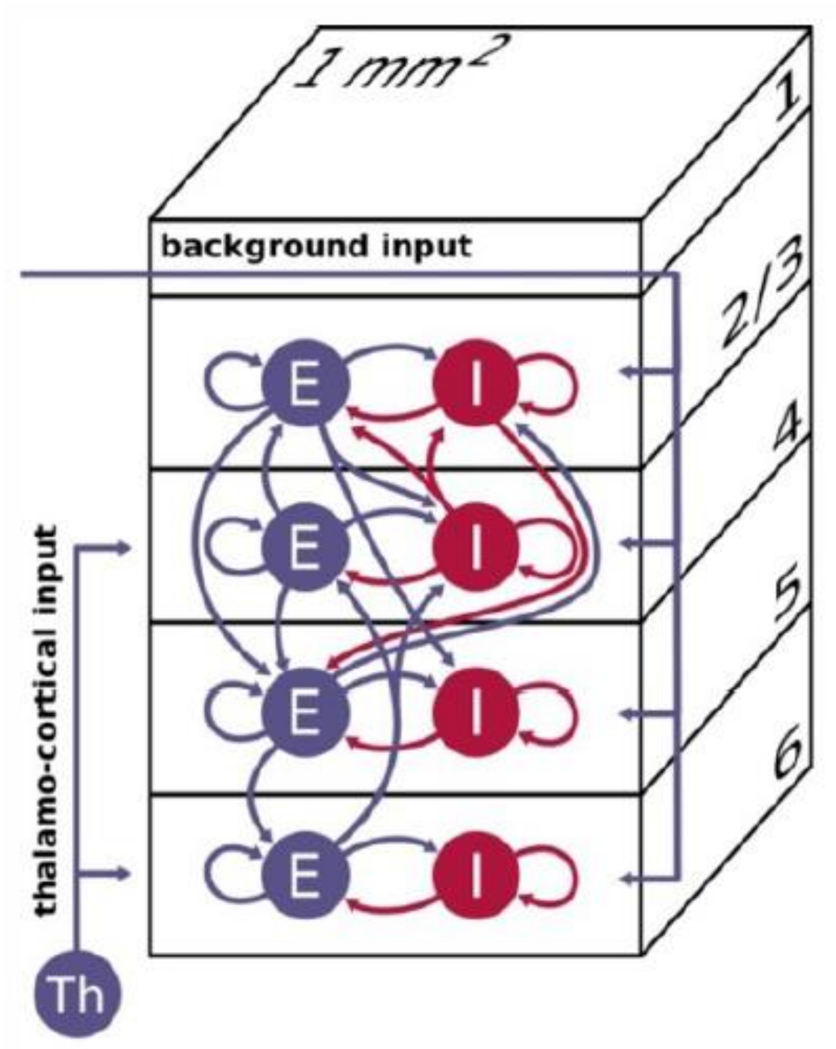
NEST example – cortical microcircuit

We want to simulate (a layer of) the cerebral cortex:



- 1mm^2
- 0.3 billion synapses, 80000 neurons
- 6 layers
- 2 population of LIAF neurons per layer

Potjans, Tobias C., and Diesmann, Markus. “The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model.” Cerebral Cortex 24.3 (2014): 785-806



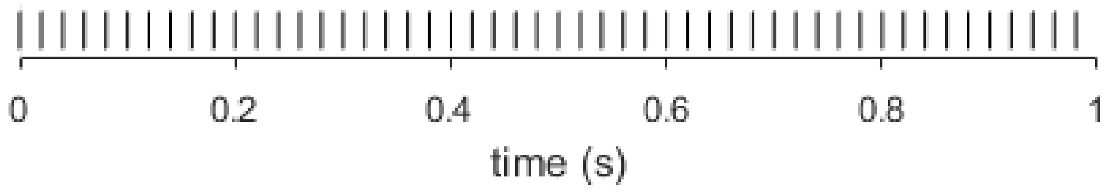
Let’s try it out!

NEST examples – Poisson generators

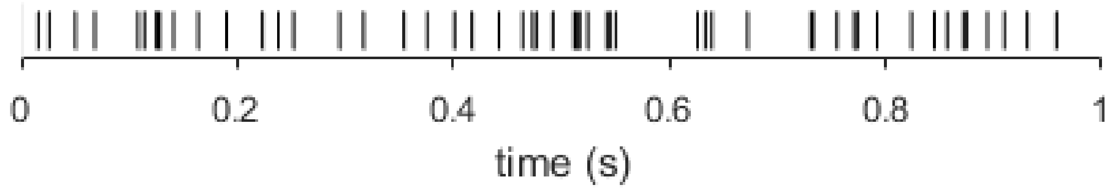
In order to give inputs to the system, representing activity of brain areas not modelled or sensory information, we need to **generate spikes** without actually simulate neurons.

It has been observed that most of the times (excluding when time encoding mechanisms are in action) the timing of successive action potential is highly irregular, probably because of stochastic forces.

Thus, when generating spikes, we want to avoid generating uniformly spaced action potentials.



Bad



Good

NEST examples – Poisson generators

To generate irregular spikes we can assume that every spike is independent from the previous one and that the generation depends solely on the instantaneous firing rate.

Upon this hypothesis we can generate spikes using a Poisson process:

$$P\{n\ spikes\ during\ \Delta t\} = e^{-r\Delta t}\frac{(r\Delta t)^n}{n!}$$

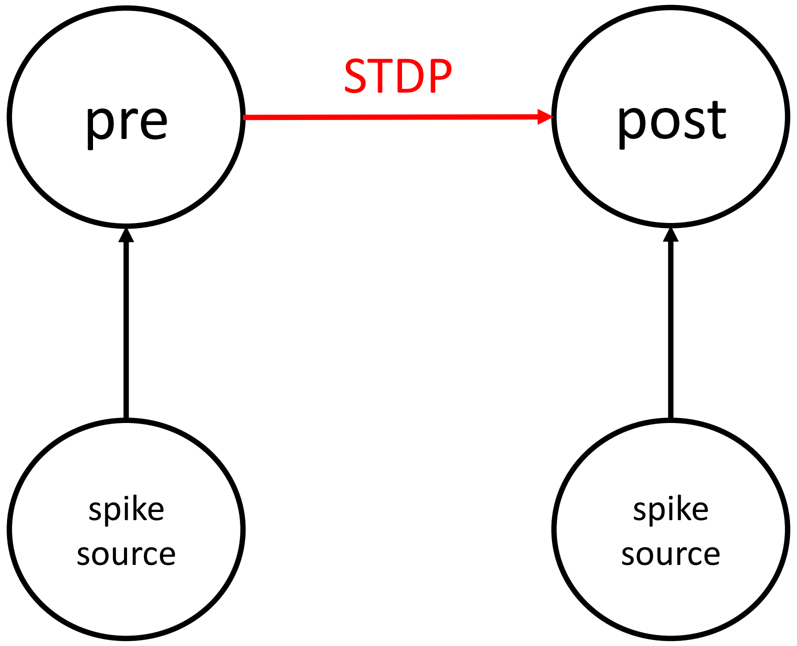
$$P\{1\ spike\ during\ \Delta t\} \approx r\Delta t\ ^*$$

* For a sufficiently short Δt

NEST example – STDP

Let’s see an example of synaptic plasticity.

- two populations, connected with STDP-enabled synapses
- external spike sources that trigger activity
- execution phases and expected results:
    1. only pre stimulation → no post activity
    2. pre and post stimulation → plasticity
    3. only pre stimulation → also post activity

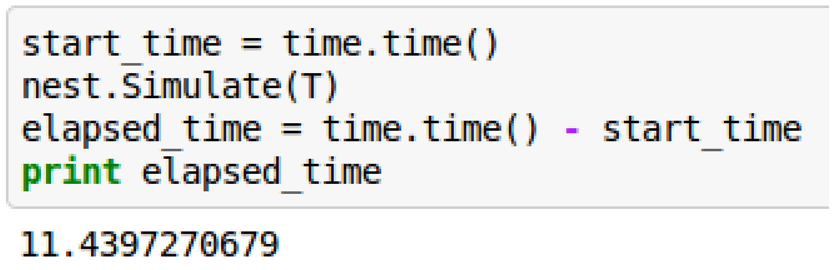


Let’s try it out!

NEST examples

What did we learn from these example?

- using nest is very easy to set up neural simulation
- nice syntactic sugar for randomized connection and weights
- useful spike recording utilities
- but it can take more than 10 seconds to simulate 1!



```
start_time = time.time()
nest.Simulate(T)
elapsed_time = time.time() - start_time
print elapsed_time
```

```
11.4397270679
```

We need to find a way to *speed up* the simulation.

NEST – parallel simulations

Let’s recall the kernel simulation loop:

1. PSP for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δ

MPI thread

MPI process

Inside a single time step, each neuron is decoupled from the others, thus the simulation of a single time step is an embarrassingly parallel problem.

In fact, NEST natively supports MPI and the parallelization of the loop.

Moreover, MPI is supported on High Performance Computing platforms!

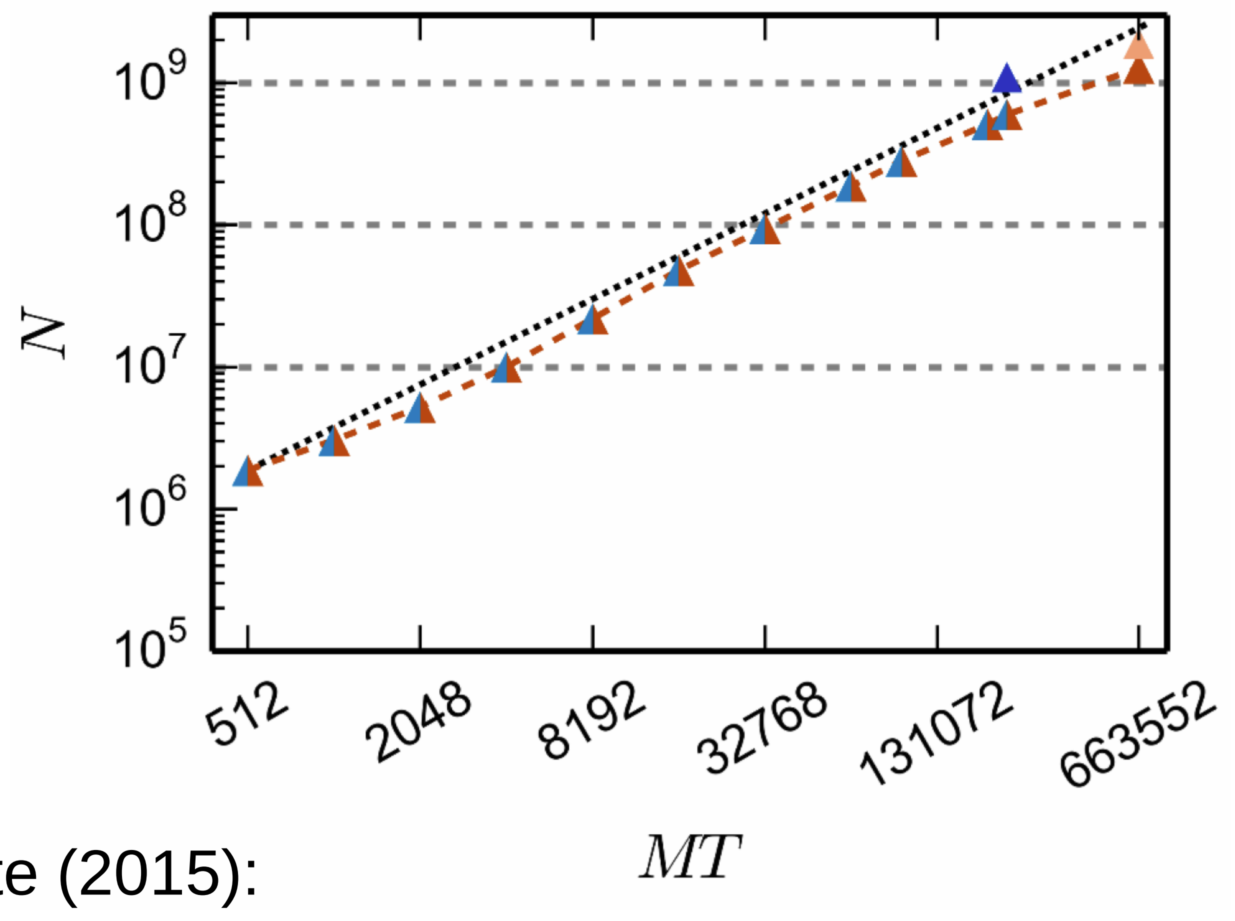
NEST – HPC

How well does nest perform on supercomputers?

Legend:

K - RIKEN, Japan
663,552 nodes, 4th

JUQUEEN - Jülich, Germany
229,376, 11th



Largest network simulation performed to date (2015):

1.86x10^9 neurons, 6000 synapses each

1.08x10^9 neurons, 6000 synapses each

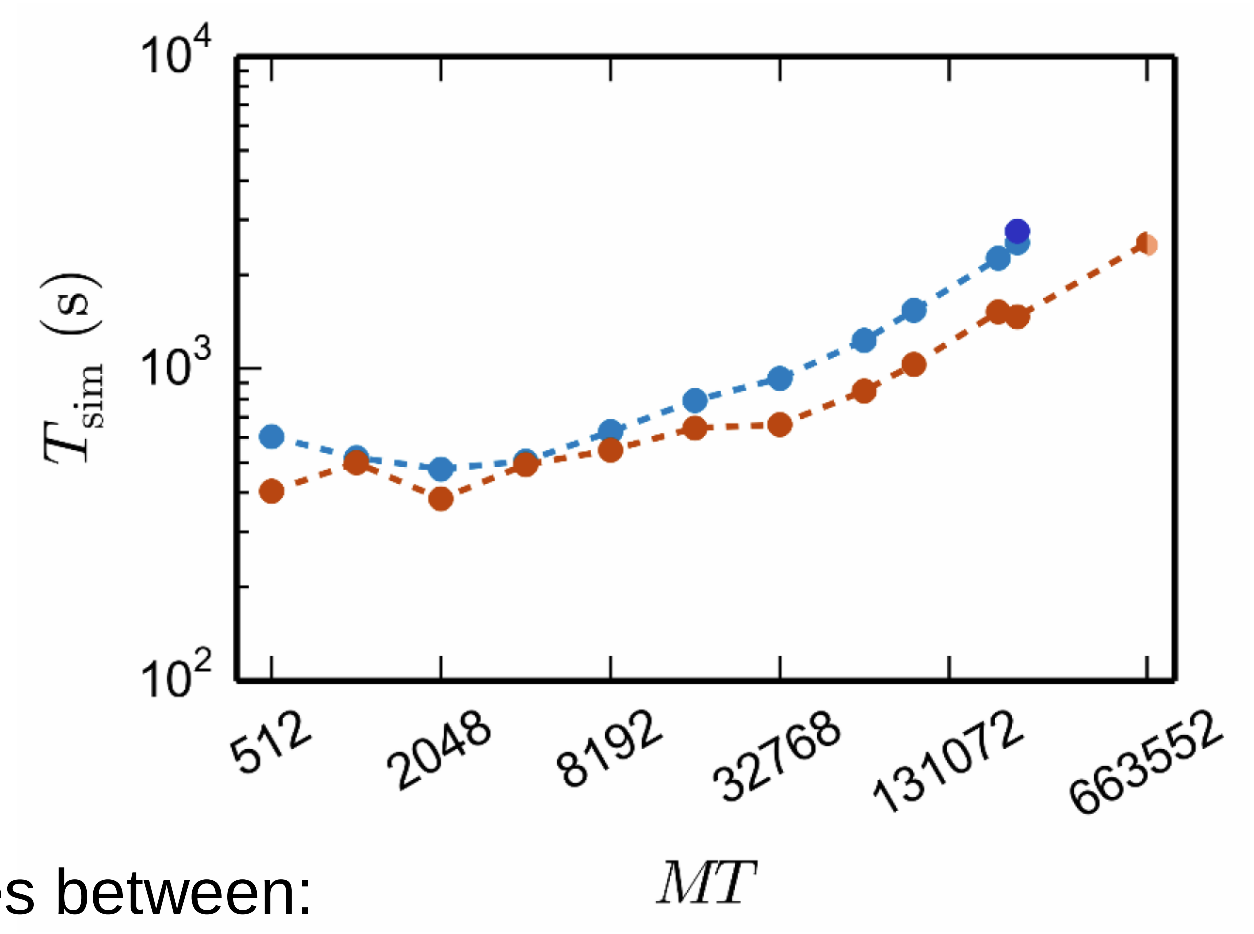
NEST – HPC

How well does nest perform on supercomputers?

Legend:

K - RIKEN, Japan
663,552 nodes, 4th

JUQUEEN - Jülich, Germany
229,376, 11th



Simulation time of 1 second of real time varies between:

between 6 and 42 minutes

between 8 and 41 minutes

NEST – Final Remarks

Is this a viable solution for physical robotics? Not really.

- even if we would like to simulate smaller networks, simulations will not be **real-time**
- usually robotics labs do not have supercomputers
- supercomputers work with job systems and as of today no interactive job mechanism exists
- latencies between the supercomputer and the robot
- power consumption (9.89 MW for K supercomp.)

However, NEST can be coupled with **robotics simulations** (more on this later).

Neuromorphic hardware

A new kind of processors, specifically designed to compute neural dynamics, have been developed in the last few years. This is what is called **neuromorphic hardware**.

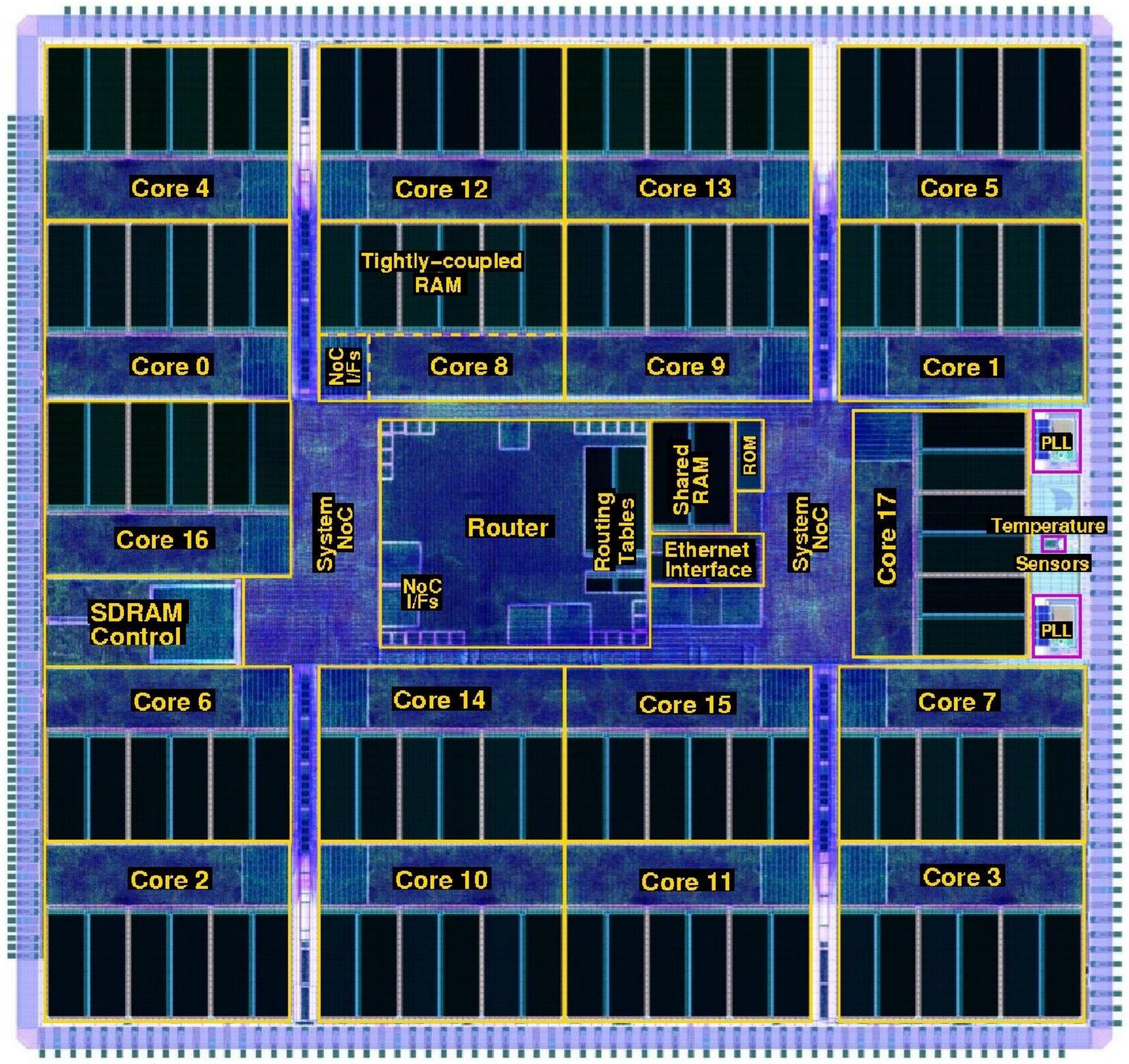
Usually, these kind of processors have these characteristics:

- massively parallel computation
- energy efficiency
- fault tolerance
- self organization of the network
- fast simulation times
- compactness

Neuromorphic hardware – SpiNNaker

SpiNNaker is a neuromorphic hardware platform developed by the University of Manchester.

- 1W chip
- 18 ARM-968 cores
- 1Gbit DDR-2 SDRAM
- 240MHz
- 6 bi-directional links
- optimized for 10million 32-bit packets/s

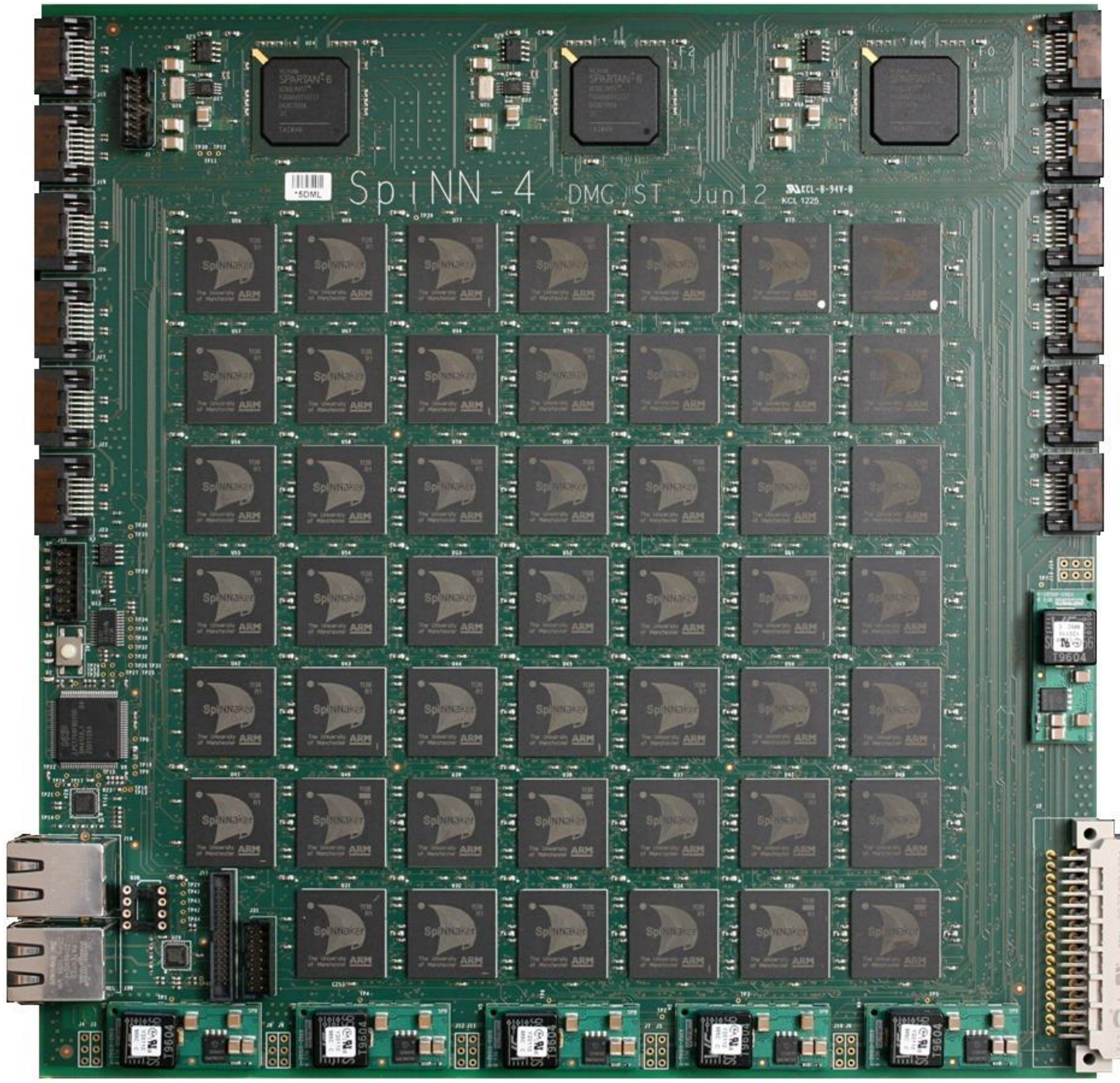


Neuromorphic hardware – SpiNNaker

SpiNNaker cores are arranged on 48 chips boards.

- 1000 neurons per core (theoretical)
- 18000 neurons per chip
- 864000 neurons per board

- 3.1Gbps SATA connections for connecting to other boards
- two 100Mbps Ethernet for control
- max 70W consumption, low temp



Neuromorphic hardware – SpiNNaker

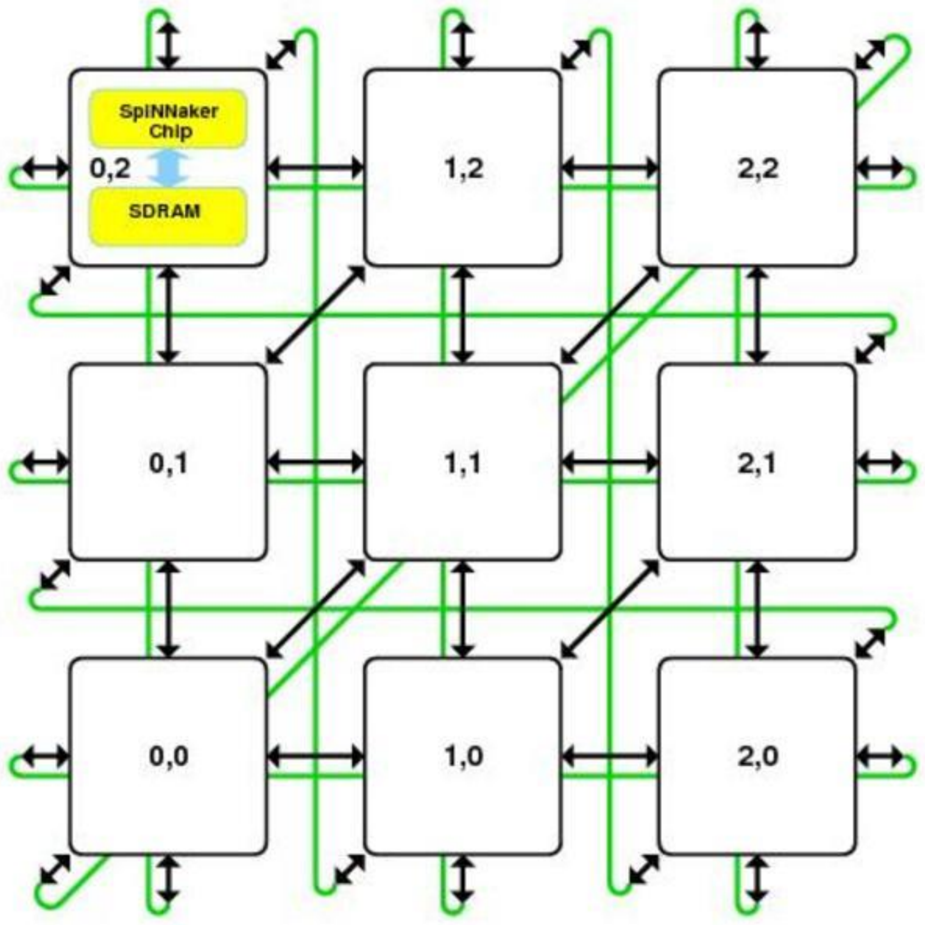
Multiple boards can be connected through SATA to allow further parallel processing exploitation.

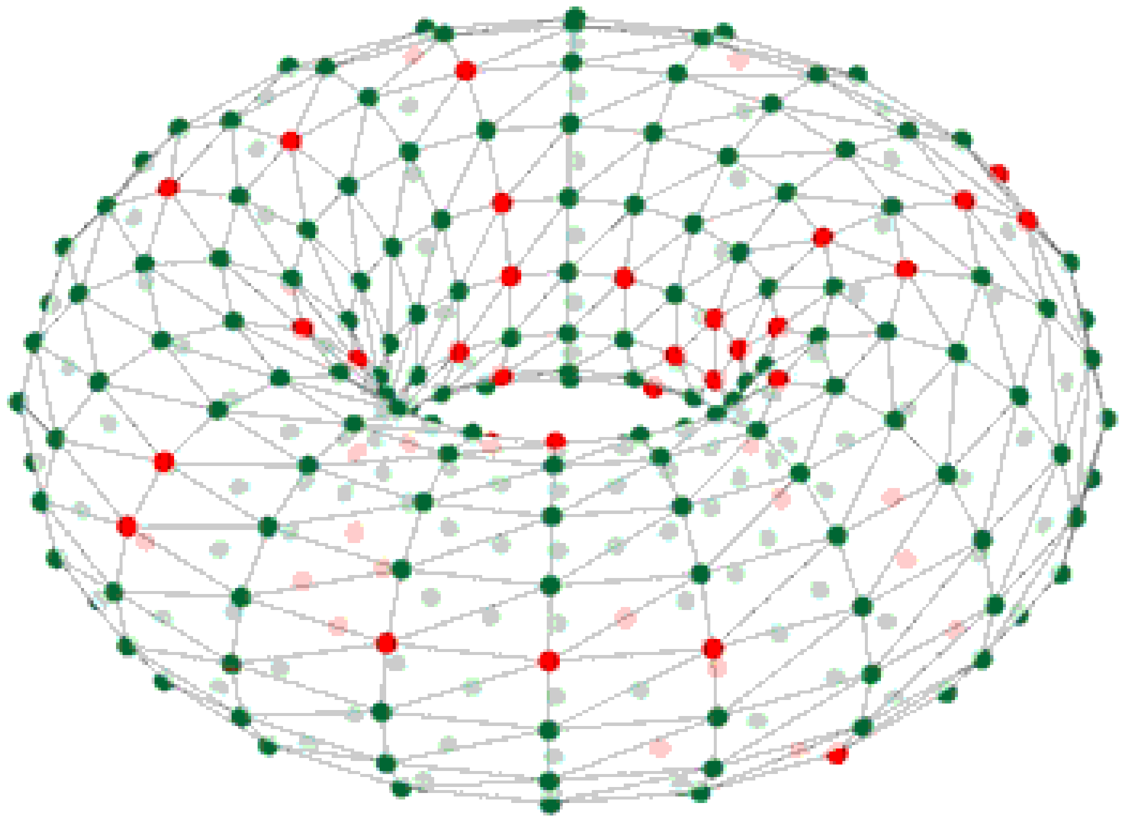
Current largest setup:

- 120 boards per cabinet
- ~1,000,000 cores
- 50 kW peak consumption
- up to a billion neurons

Neuromorphic hardware – SpiNNaker

In order to provide fast spike transmission between cores, a proper connectivity method must be exploited.





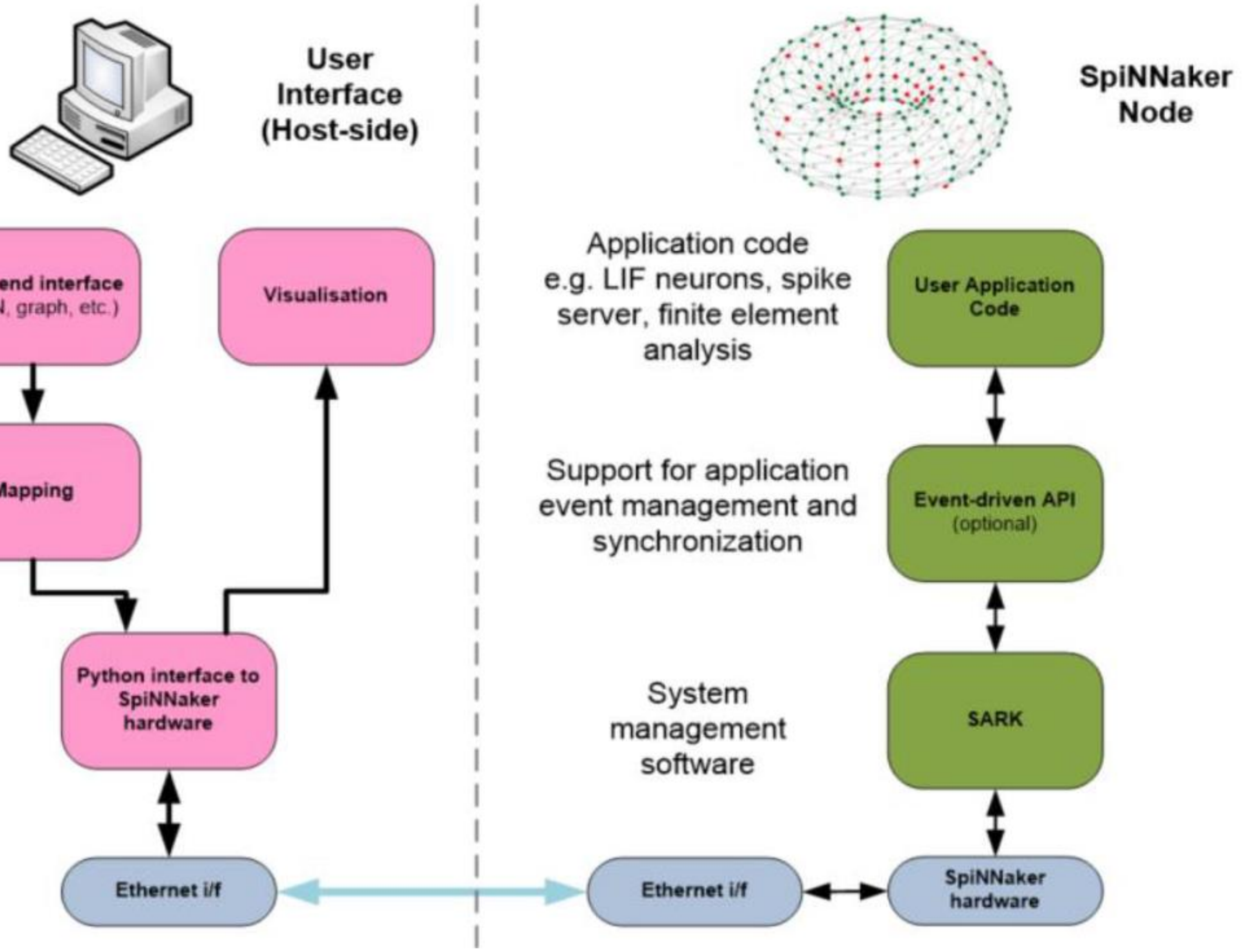
Toroidal connectivity ensures fast spike delivery among chips.

Neuromorphic hardware – SpiNNaker

How do one use these boards? There is a Python library that we can use to set up the network on the SpiNNaker cores: PyNN.

PyNN is a frontend for different neural simulators (including SpiNNaker and NEST)

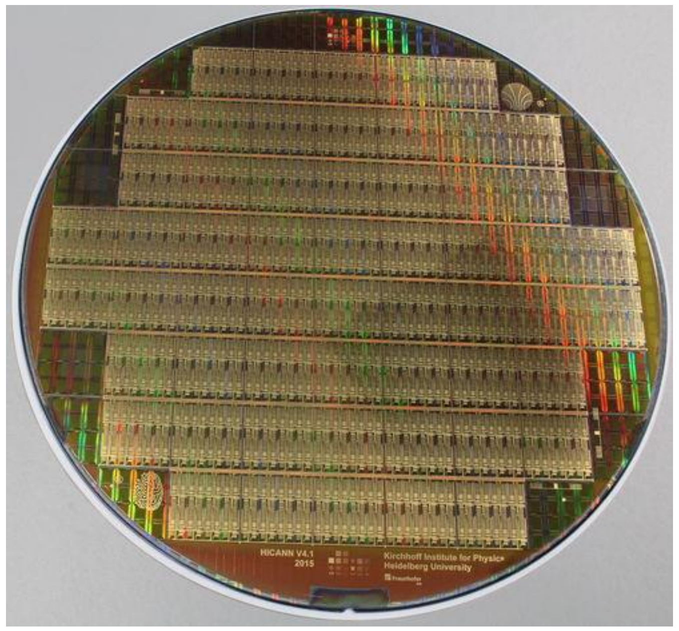
neuralensemble.org/PyNN

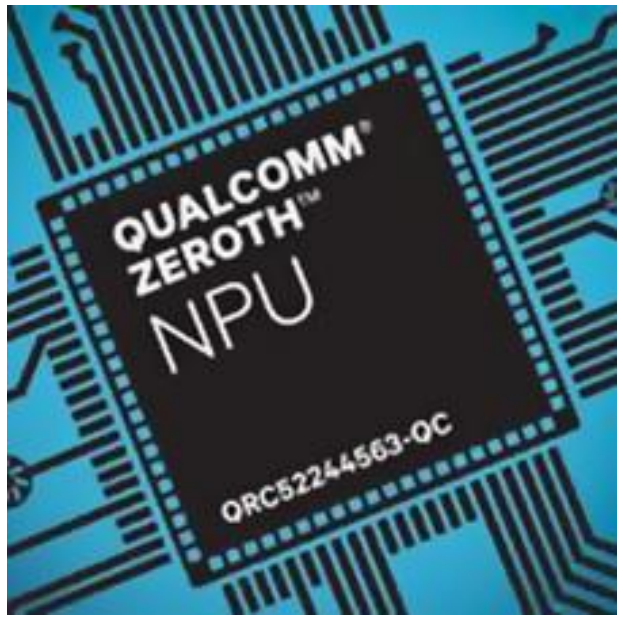

User Interface (Host-side)
SpiNNaker Node
Front end interface (PyNN, graph, etc.)
Visualisation
Mapping
Python interface to SpiNNaker hardware
Ethernet i/f
Application code e.g. LIF neurons, spike server, finite element analysis
User Application Code
Support for application event management and synchronization
Event-driven API (optional)
System management software
SARK
Ethernet i/f
SpiNNaker hardware

Neuromorphic hardware

SpiNNaker is not the only neuromorphic hardware platform:

Name	Developer	Features
TrueNorth	IBM	Custom processor, 4096 cores with 256 neurons each
BrainScaleS	Heidelberg	Physical model, accelerated simulation time
Brainstorm	Stanford	Physical model, real time
Zeroth	Qualcomm	Deep learning on Snapdragon





Neuromorphic hardware – Final remarks

Pros:

- **real time** neural simulation
- low power consumption
- portable (can be embedded on robots)

Cons:

- cost, availability
- limited number of neurons and connections by design
- still in development
- can lose spikes if firing rates are too high

Suitable to be embedded on a **physical robotic platform**.

Outline

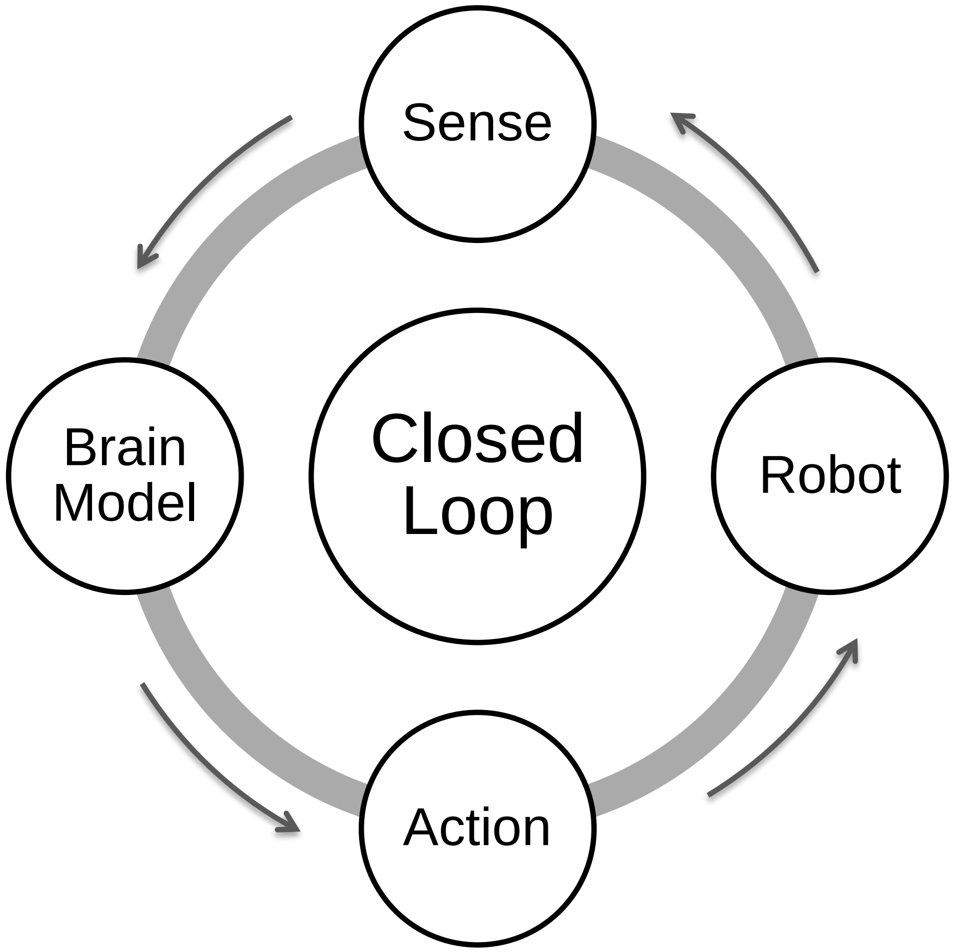
Robotic applications

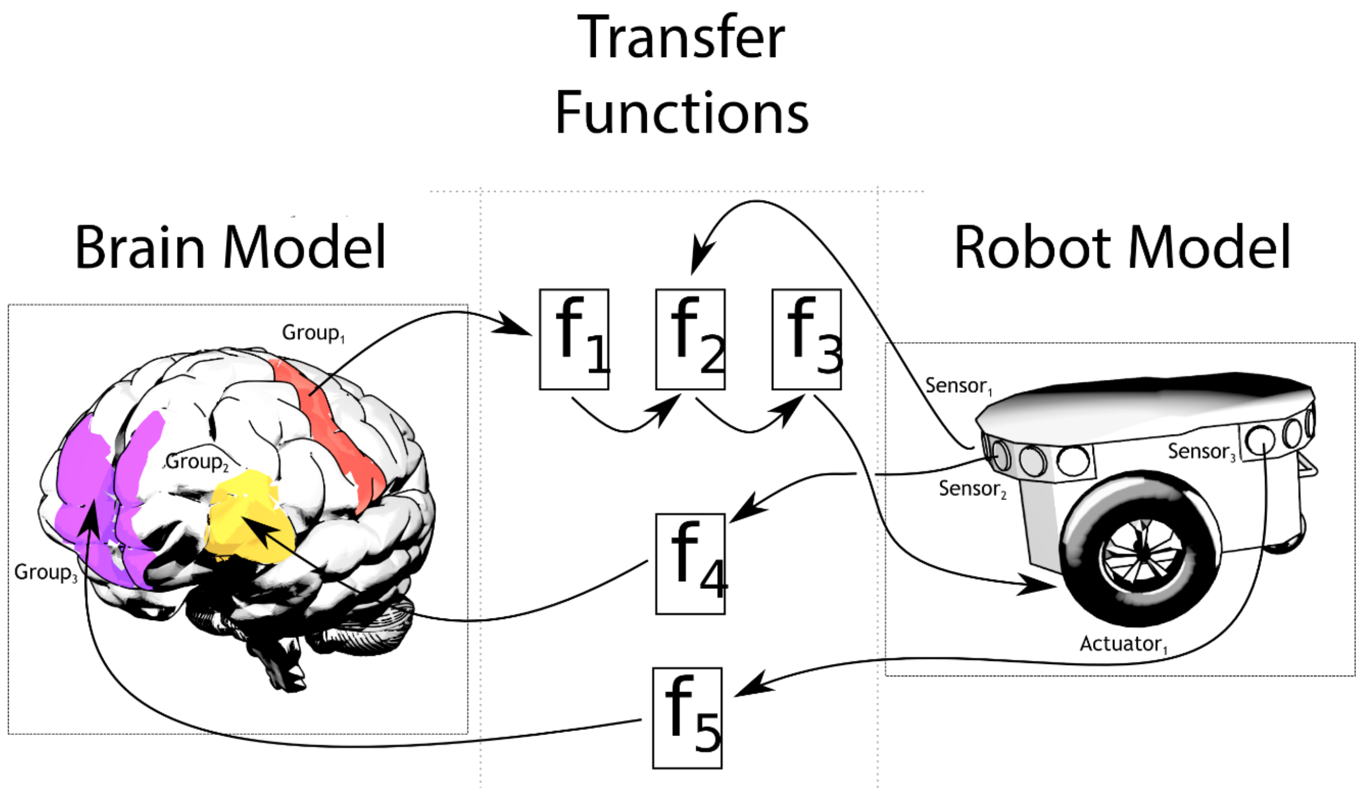
How can we integrate brain models with robotic platforms?

- spiking neural network can be integrated alongside classic robot controllers, relieving them of some computation
- bio-inspired brain models works well for processing of data coming from bio-inspired sensors
- bio-inspired brain models works well for bio-inspired actuators (tendon driven robots, muscle like actuators)
- if connected to a robot, the neural simulation must run in real-time
- if real-time neural simulation is not possible we have to simulate also the robot

The Neurorobotic Closed Loop

One way of integrating neuromorphic computing and robotics is implementing a *closed loop*, a complete action-perception mechanism that involves exchanging information between a robot and a brain model. Information between the two must be properly processed and converted.



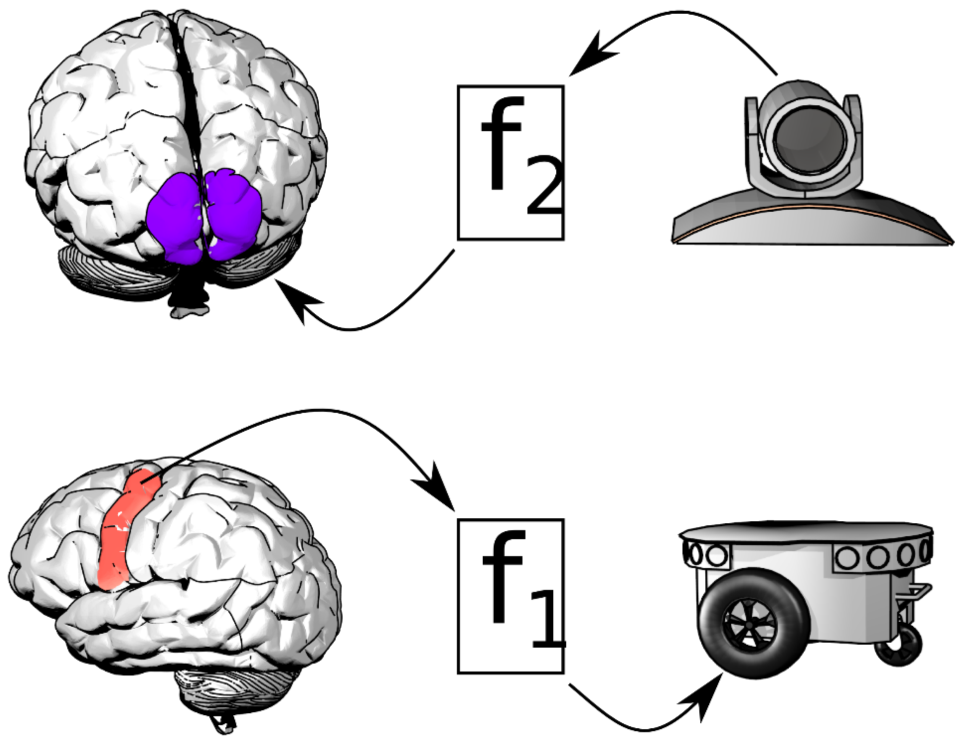


The Neurorobotic Closed Loop

Information between the robot and the brain model must be properly converted and exchanged.

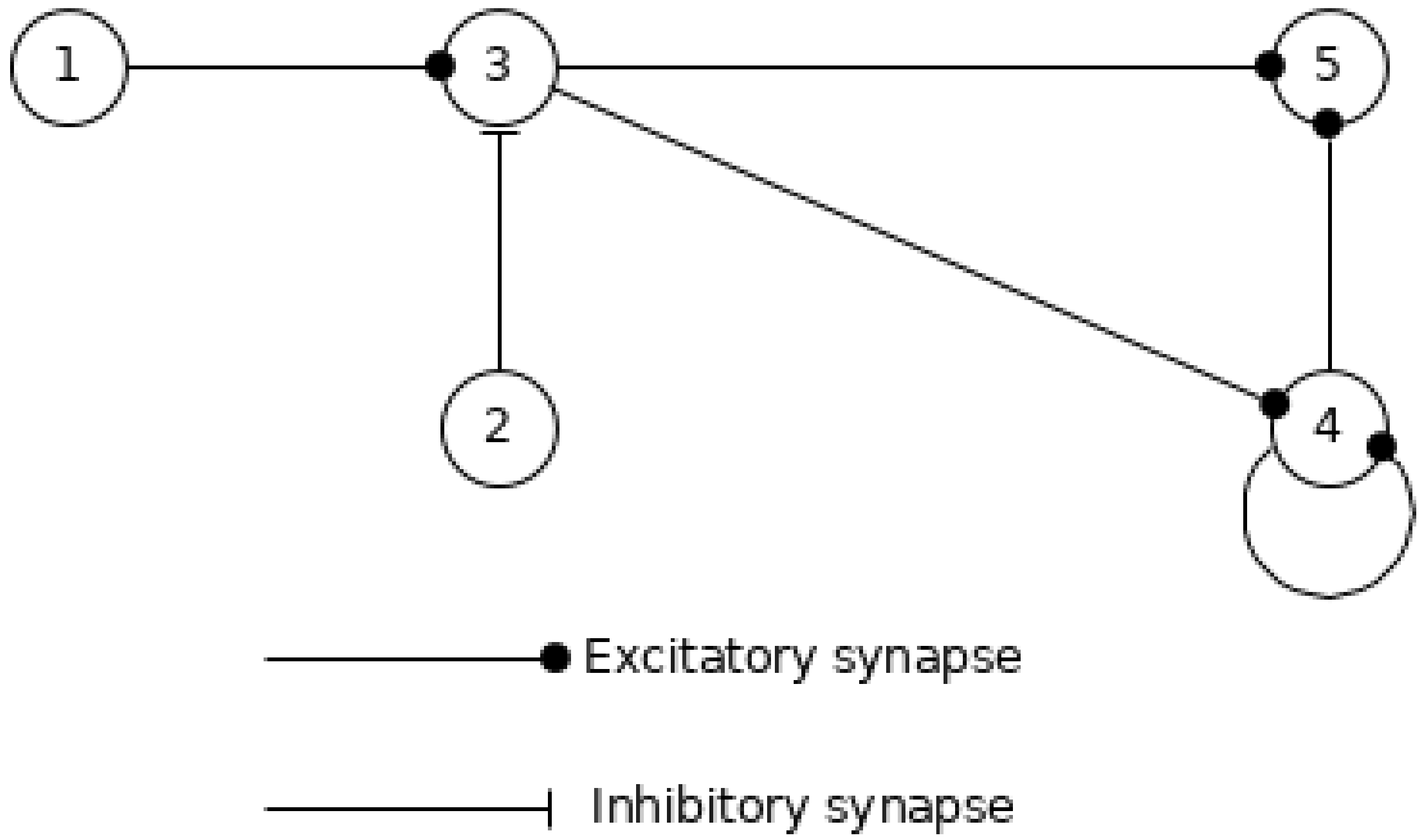
- **robot to neuron**: translate sensory information into spikes and current amplitudes, performing some *encoding*

- **neuron to robot**: take measurements on the neural network (spike rate, membrane potential) and transform them into robot commands, thus performing some *decoding*



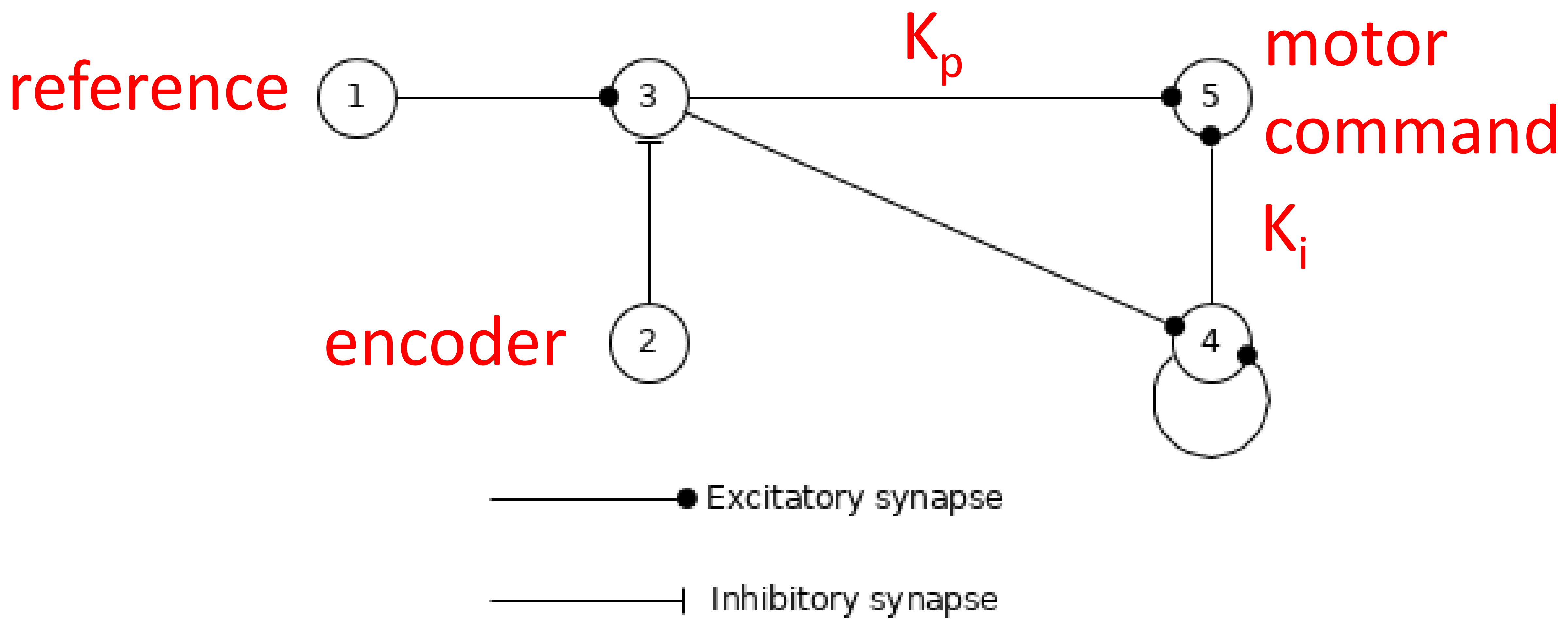
Neurorobotic Closed Loop example

Describe the behaviour of the network below (i.e. what is computed by neuron 5 with respect to neurons 1 and 2). How can such a network be used to implement a low level controller for a motor-actuated robot joint?



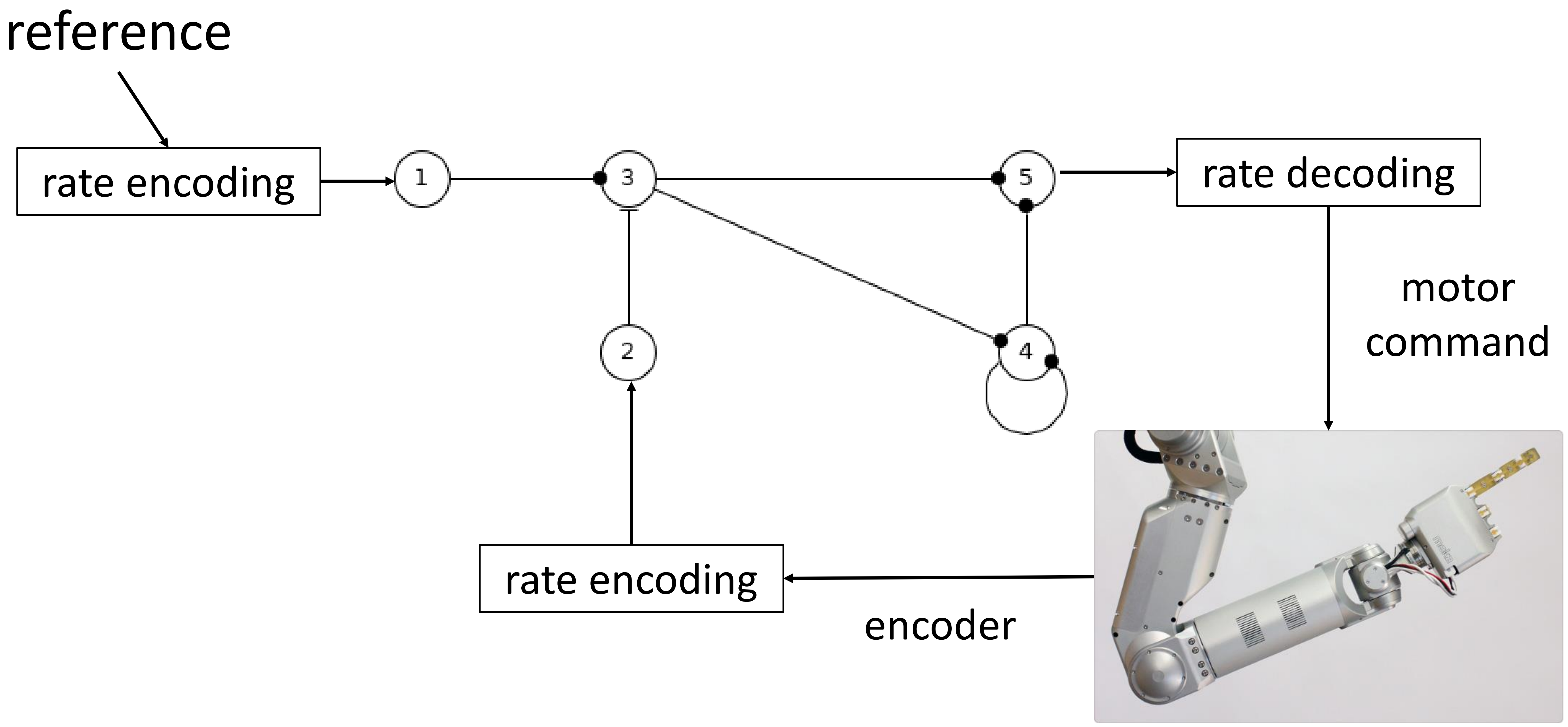
Neurorobotic Closed Loop example

Solution: a PI controller.



Neurorobotic Closed Loop example

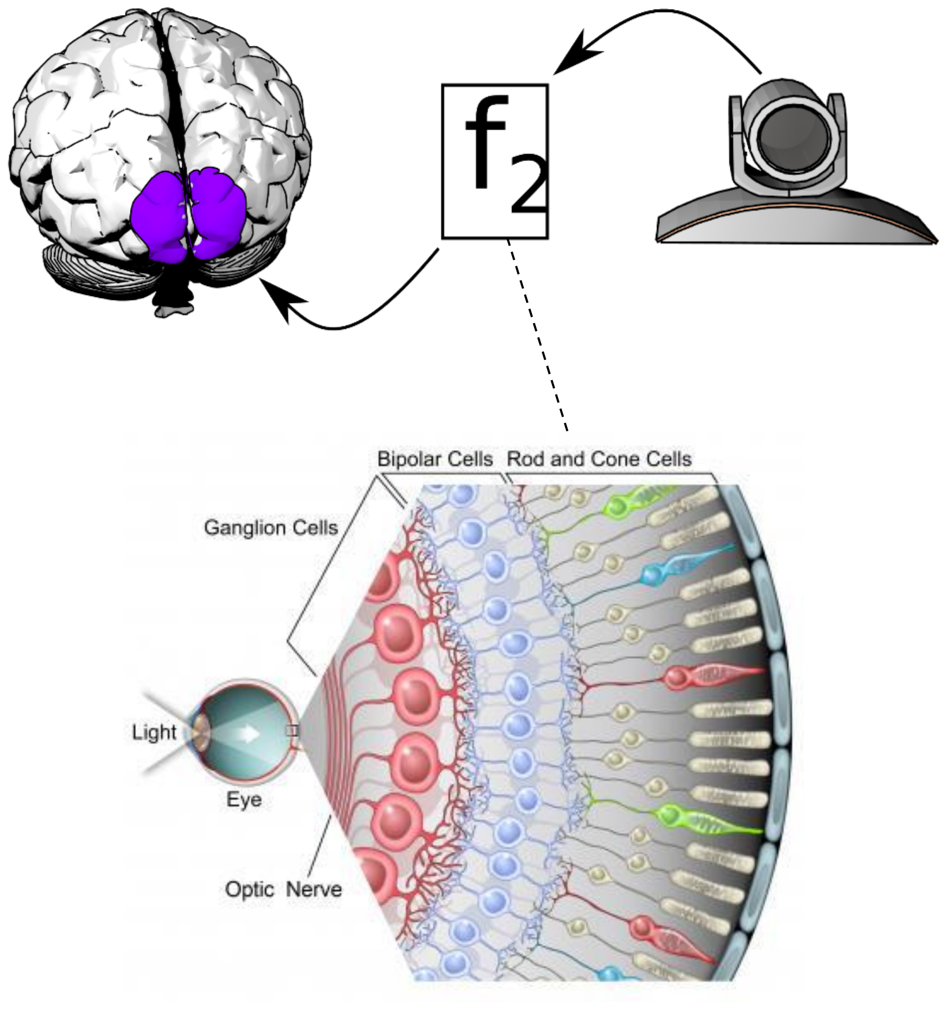
Solution: a PI controller.

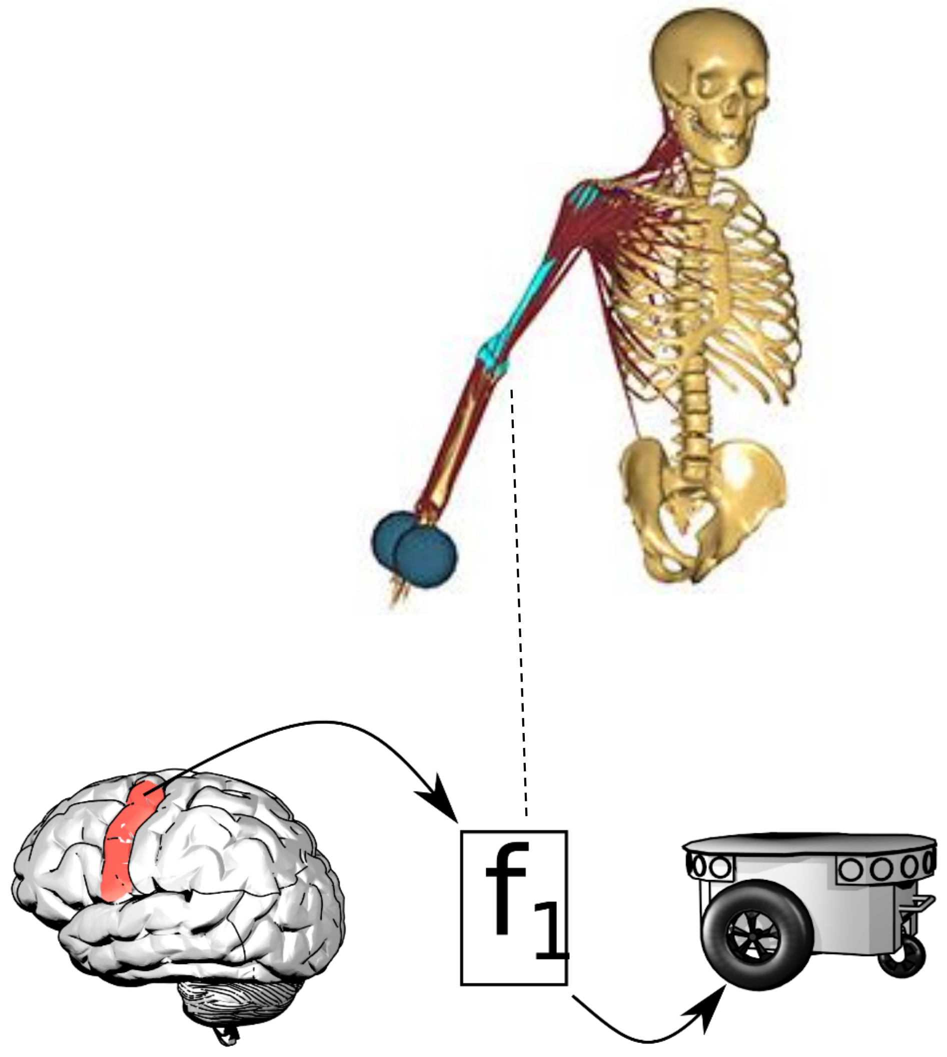


The Neurorobotic Closed Loop

Can we find some general methods to translate information between the two worlds?

Perhaps we could use biological models to do so.

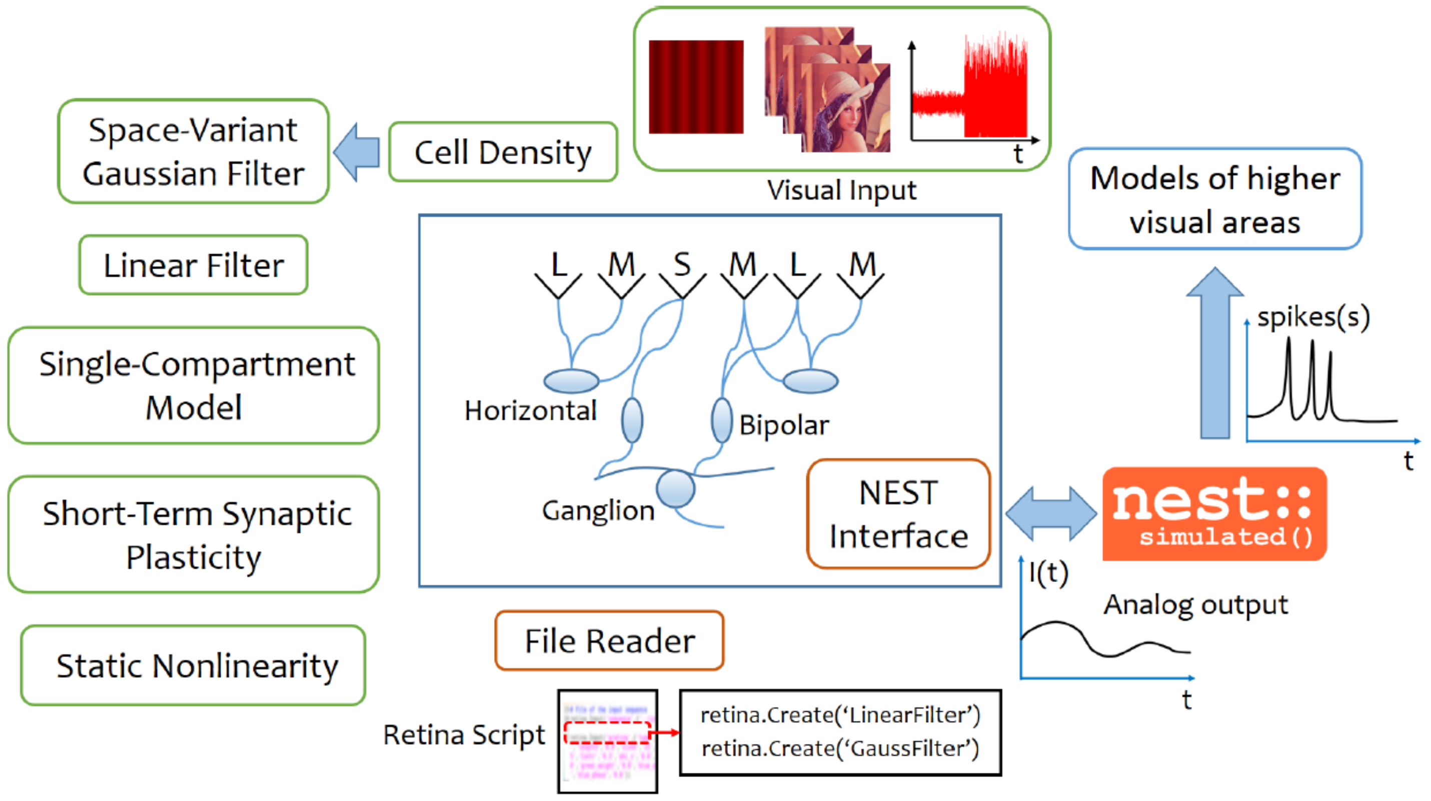




Retinal visual tracking

In this work we integrated bio-inspired sensing with spiking neural network in order to perform a visual tracking task.

We used the same setup as before where we also integrated a retina simulation as a robot to neuron transfer function.



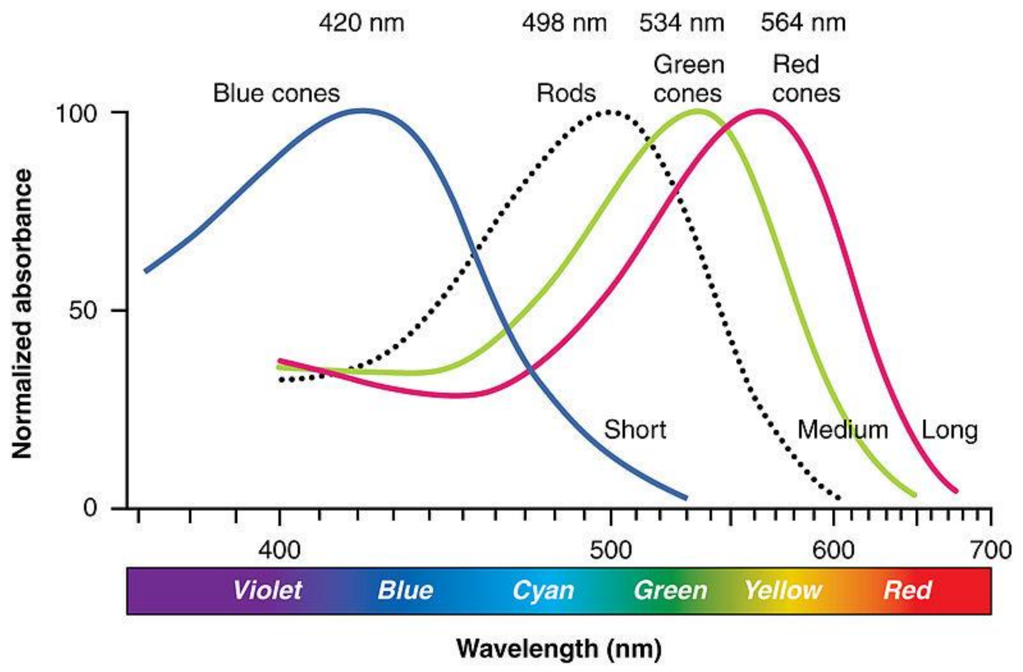
COREM simulator, developed by University of Granada.

- custom retina models
- based on linear/non-linear analysis
- produces an output compatible with NEST, but <u>not</u> spiking

Ambrosano, Alessandro, Lorenzo Vannucci, Ugo Albanese, Murat Kirtay, Egidio Falotico, Georg Hinkel, Jacques Kaiser et al. "Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform." In *Conference on Biomimetic and Biohybrid Systems*, pp. 16-27. Springer International Publishing, 2016.

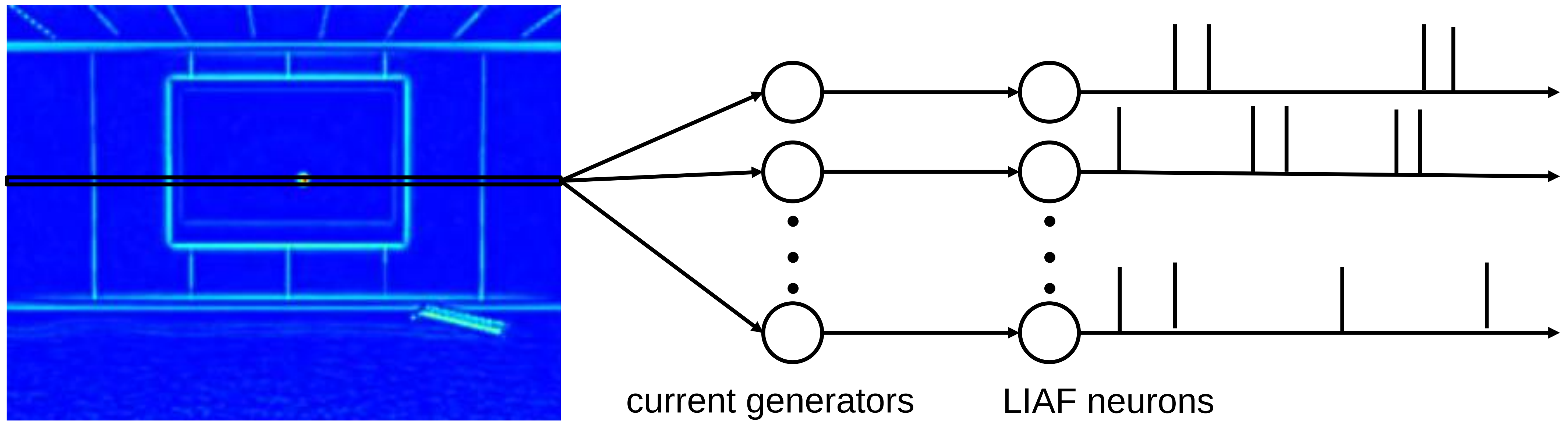
Retinal visual tracking

At first we performed a target detection via retinal image processing.



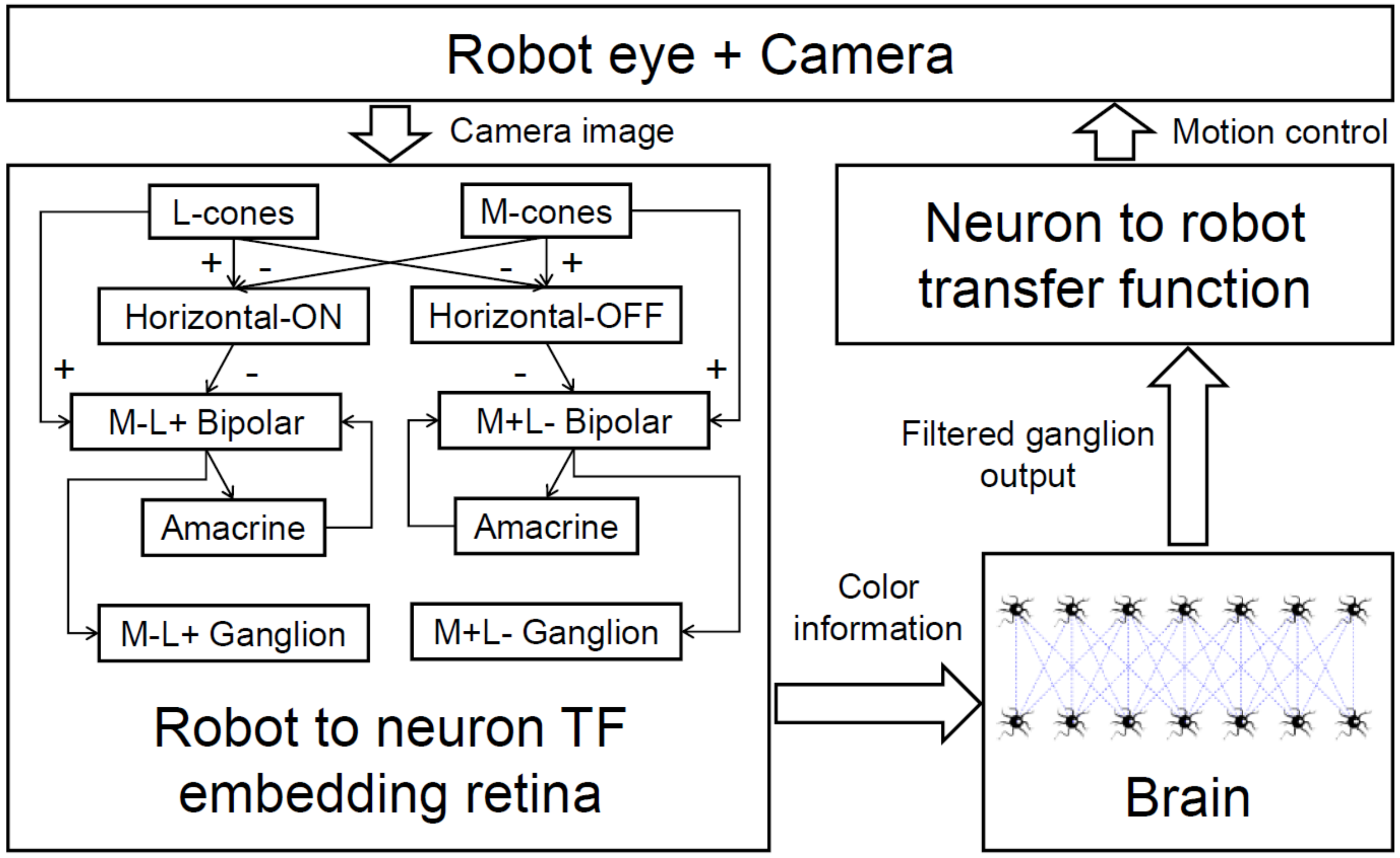
We used a complete retina model, but only with the pathway coming from M-cones (more sensitive to green).

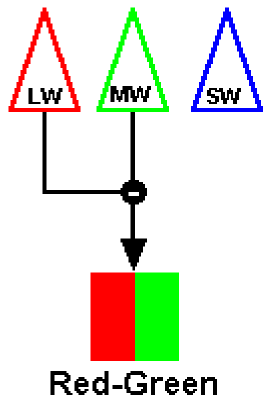
Analogue output from the ganglion cells is sent to a LIAF neuron layer via current generators devices.



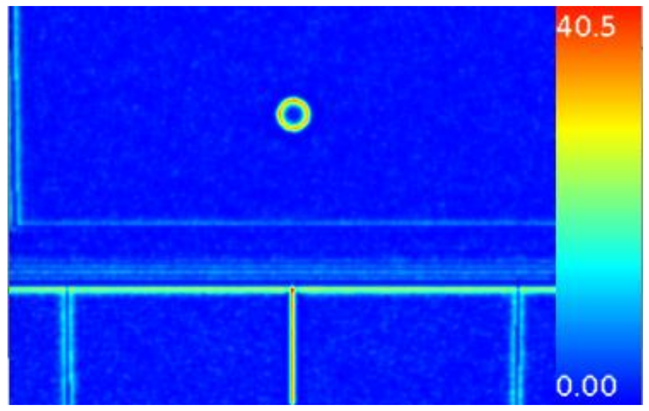
Retinal visual tracking

Then, we switched to a more sophisticated retina model, based on red-green opponency.



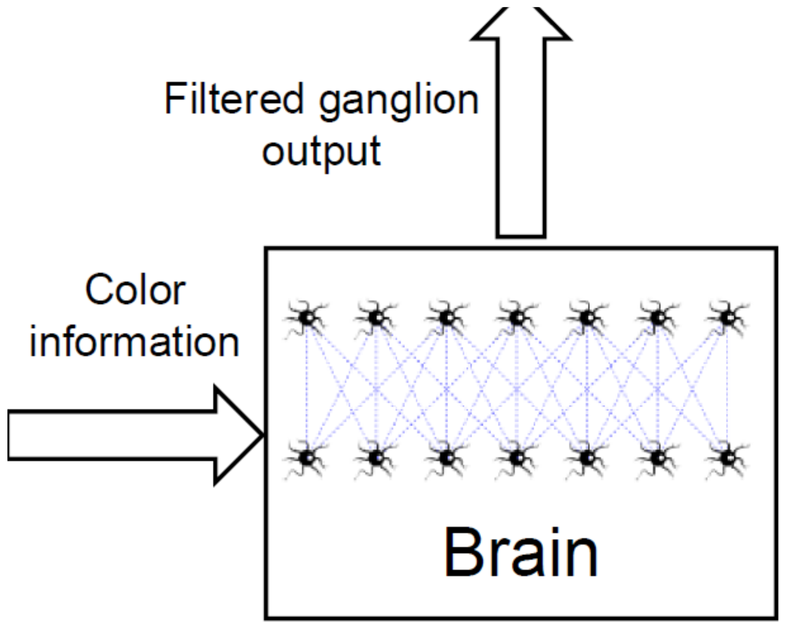


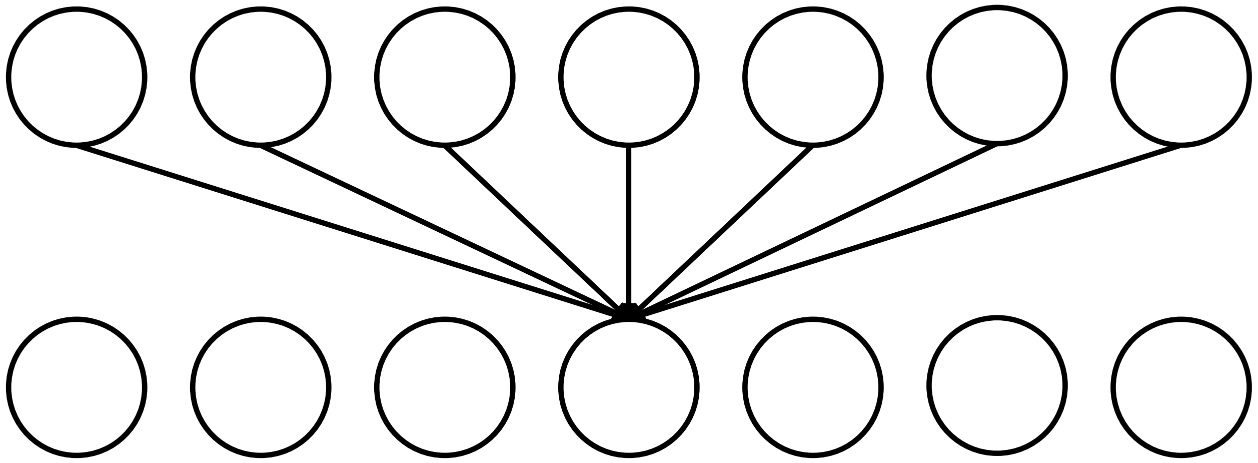
The output value of this retina model is higher for edges of the target.



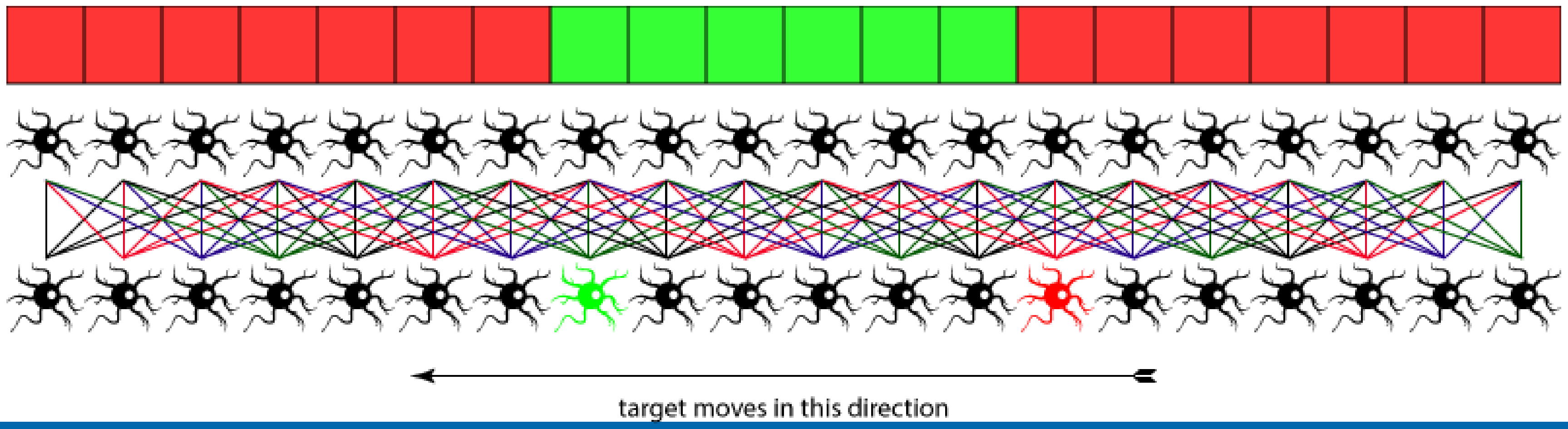
Retinal visual tracking

In order to filter out the noise from the ganglion output and retain only the target information, a two layer spiking neural network was used.



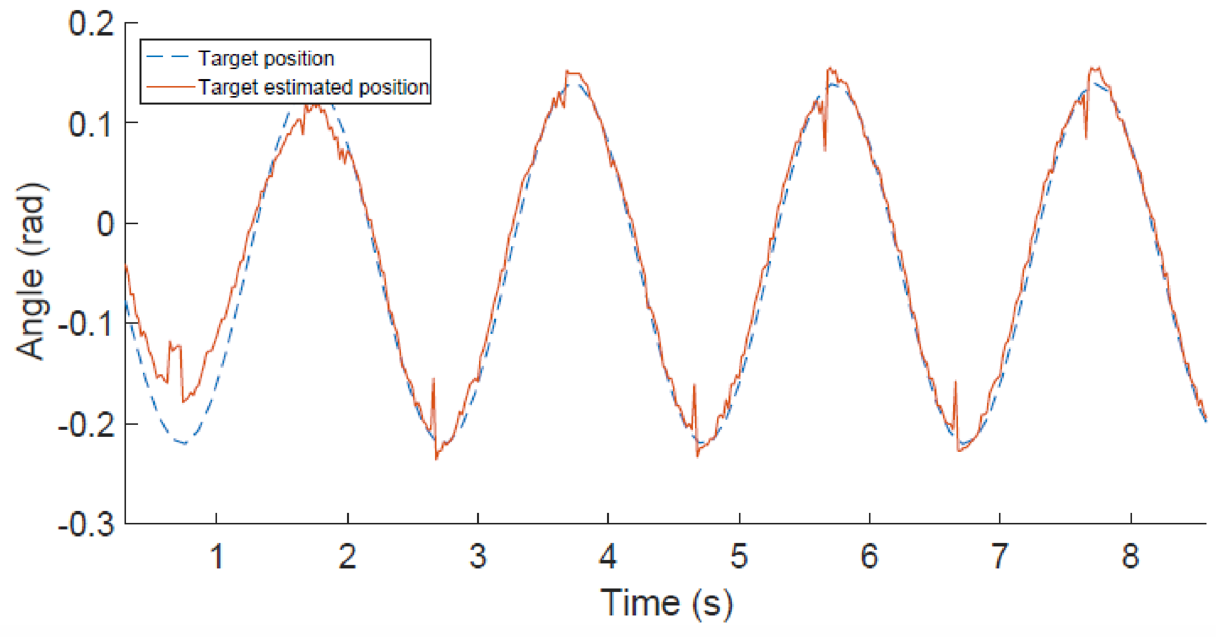


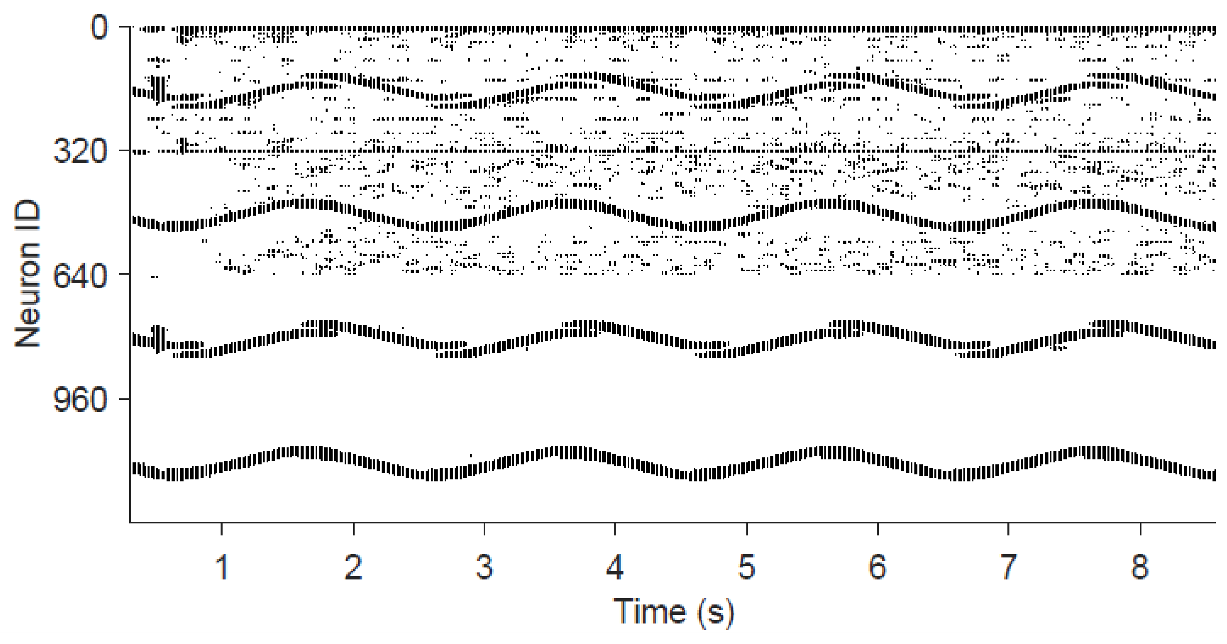
The first layer is a current to spike converter. Neurons in the second layer receives spikes from a *receptive field* of 7 neurons (pixels).



Retinal visual tracking

Using output of the neural network we can estimate the target centroid and use this information to generate motor commands for the robot eye.

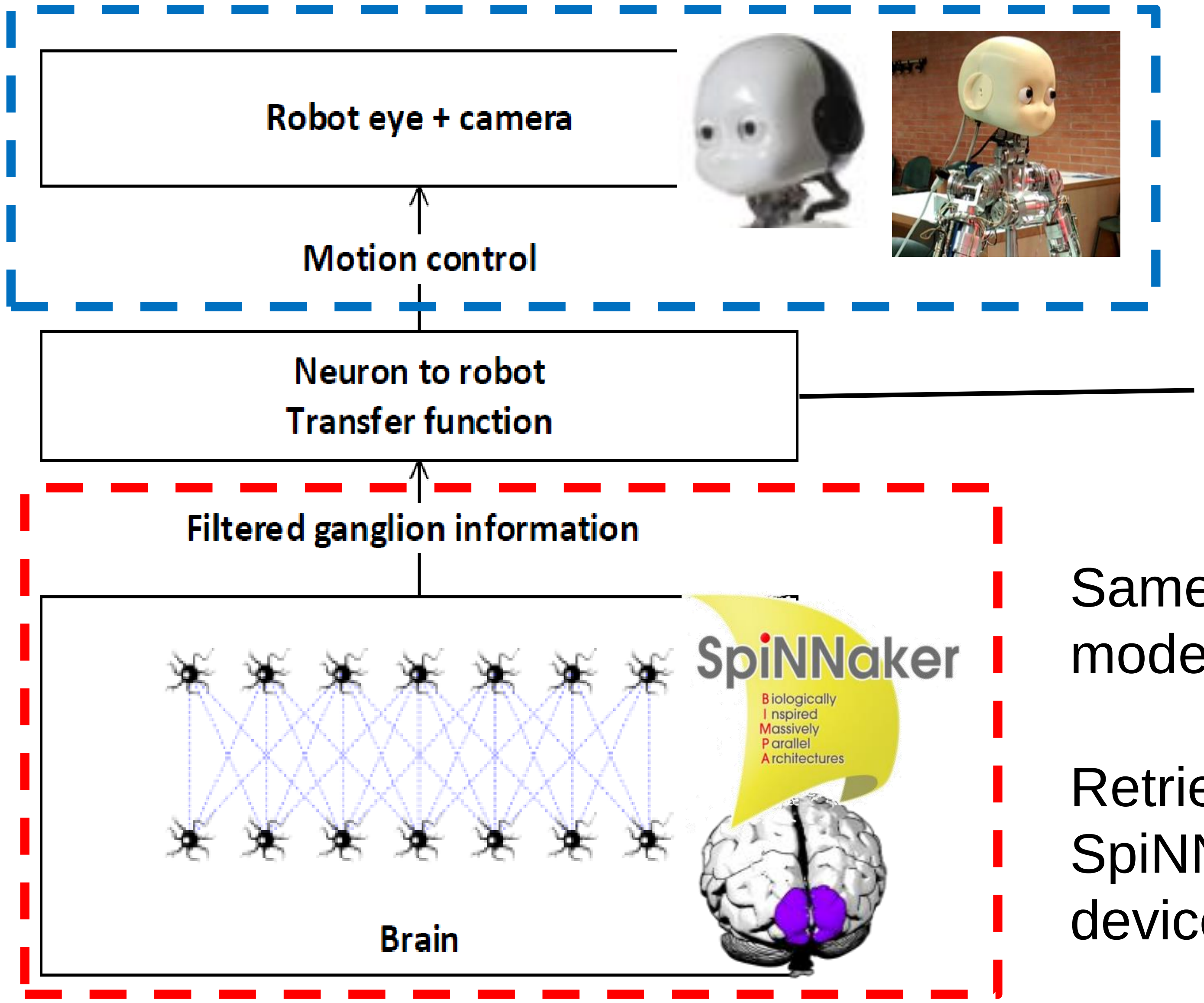




The robot is able to follow the target thanks to the neural filtering of the retinal output.

Visual tracking with SpiNNaker

The same controller was also implemented on the real iCub robotic platform, using neuromorphic hardware for real-time neural simulation.



Real iCub robot,
accessed via YARP

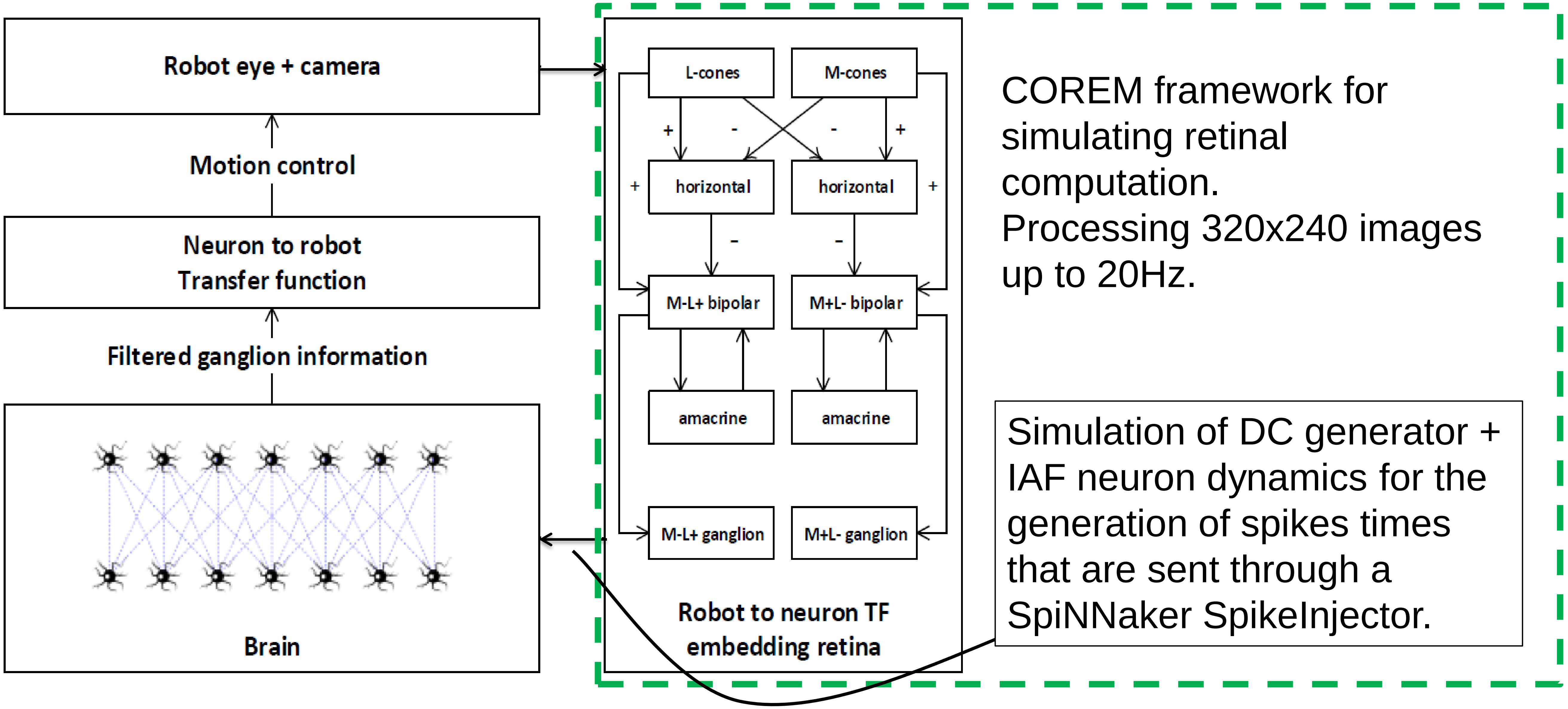
Same transfer function

Same PyNN brain model

Retrieval of spikes from SpiNNaker receiver devices

Visual tracking with SpiNNaker

The same controller was also implemented on the real iCub robotic platform, using neuromorphic hardware for real-time neural simulation.

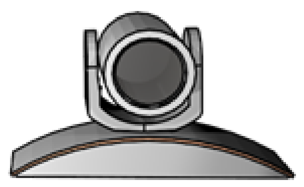


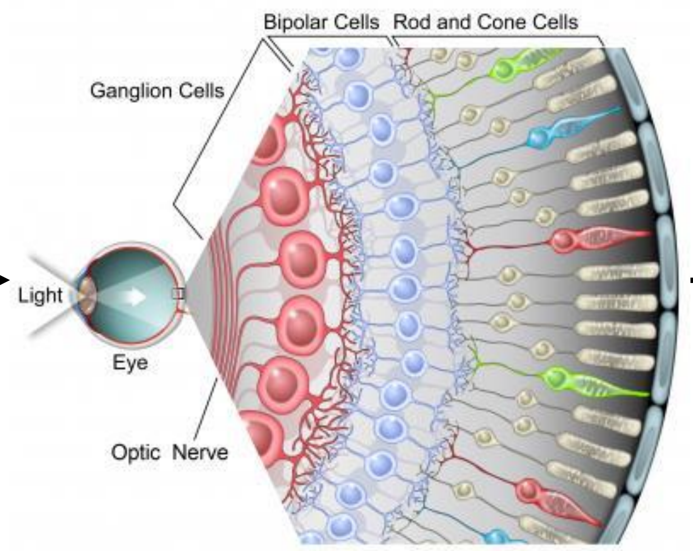
COREM framework for simulating retinal computation. Processing 320x240 images up to 20Hz.

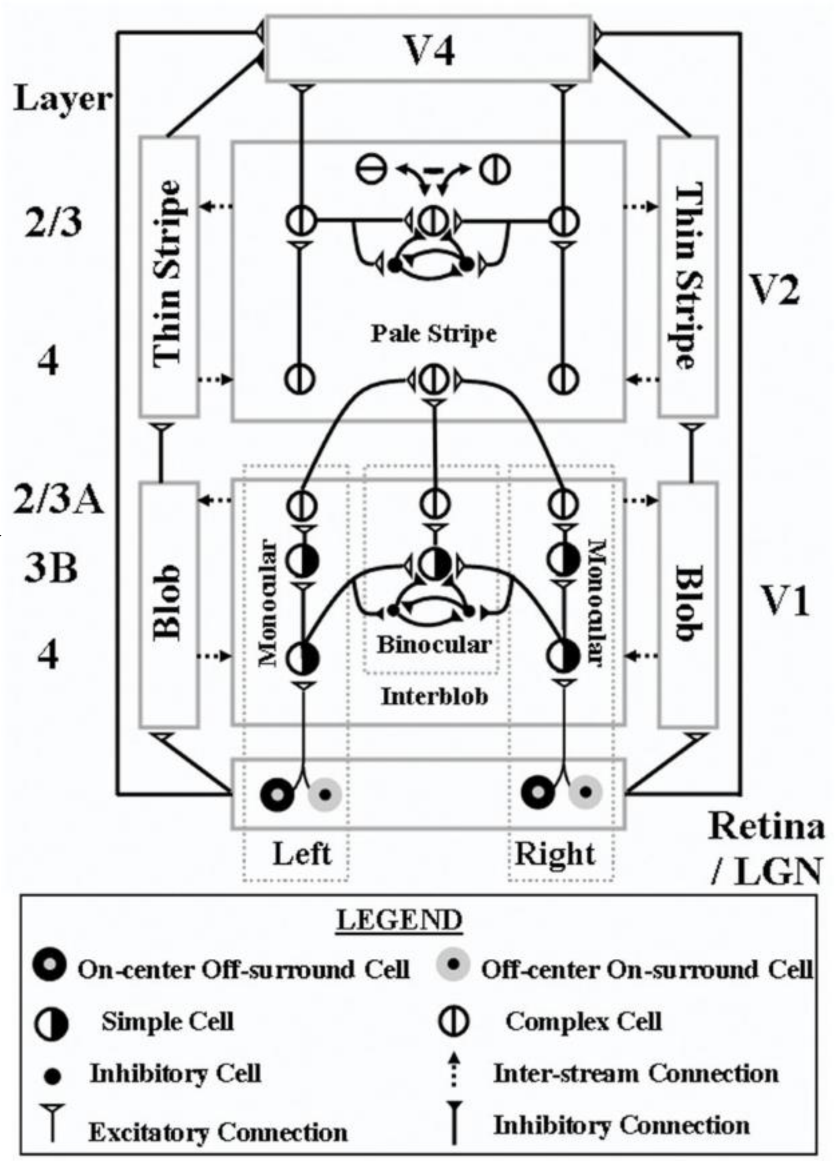
Simulation of DC generator + IAF neuron dynamics for the generation of spikes times that are sent through a SpiNNaker SpikeInjector.

Retina as a generic translation mechanism

This kind of translation is actually generic and in fact it was employed with a more complex visual cortex model.

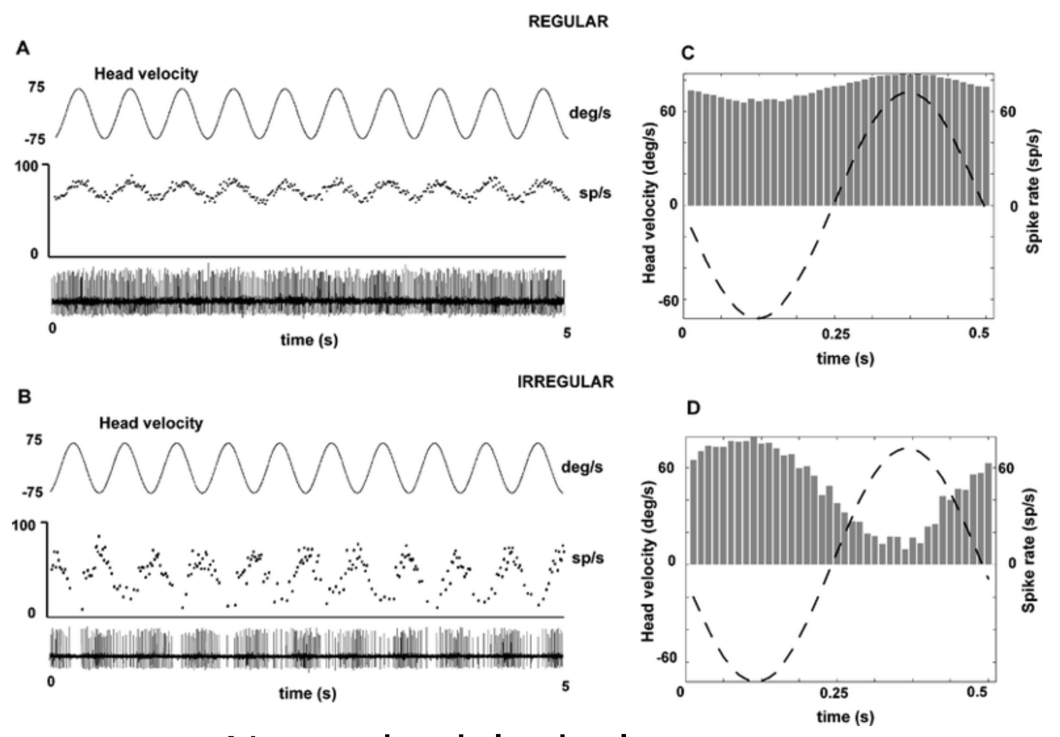




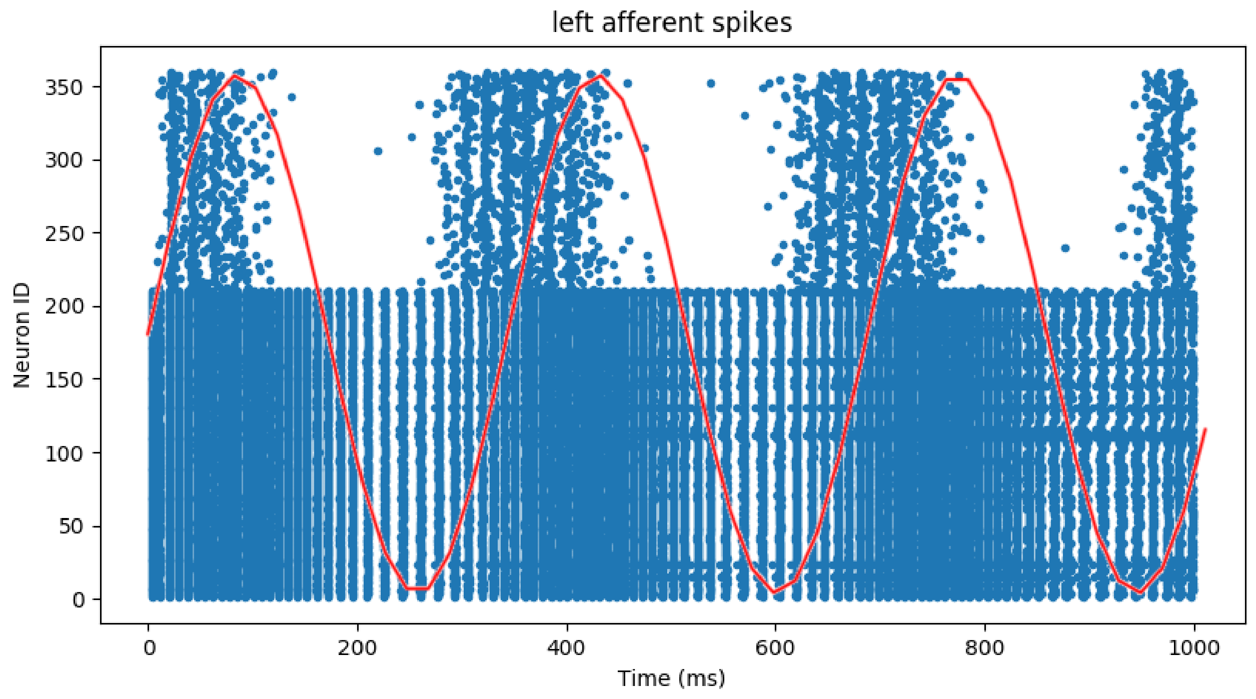


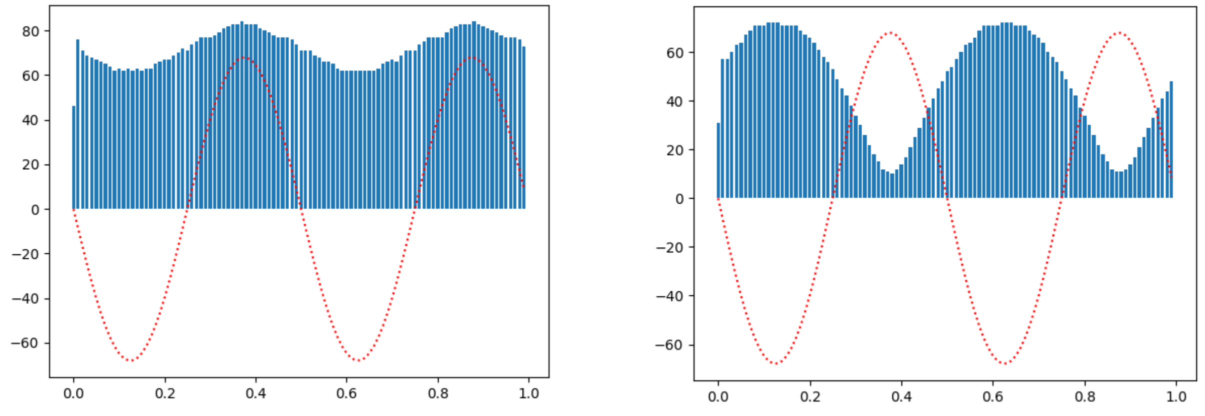
Neuromorphic model of vestibular afferents

In order to translate information coming from inertial sensors, we developed a neuromorphic model of vestibular afferents that comprises of both **regular** and **irregular** afferents.



Neurophysiological recordings





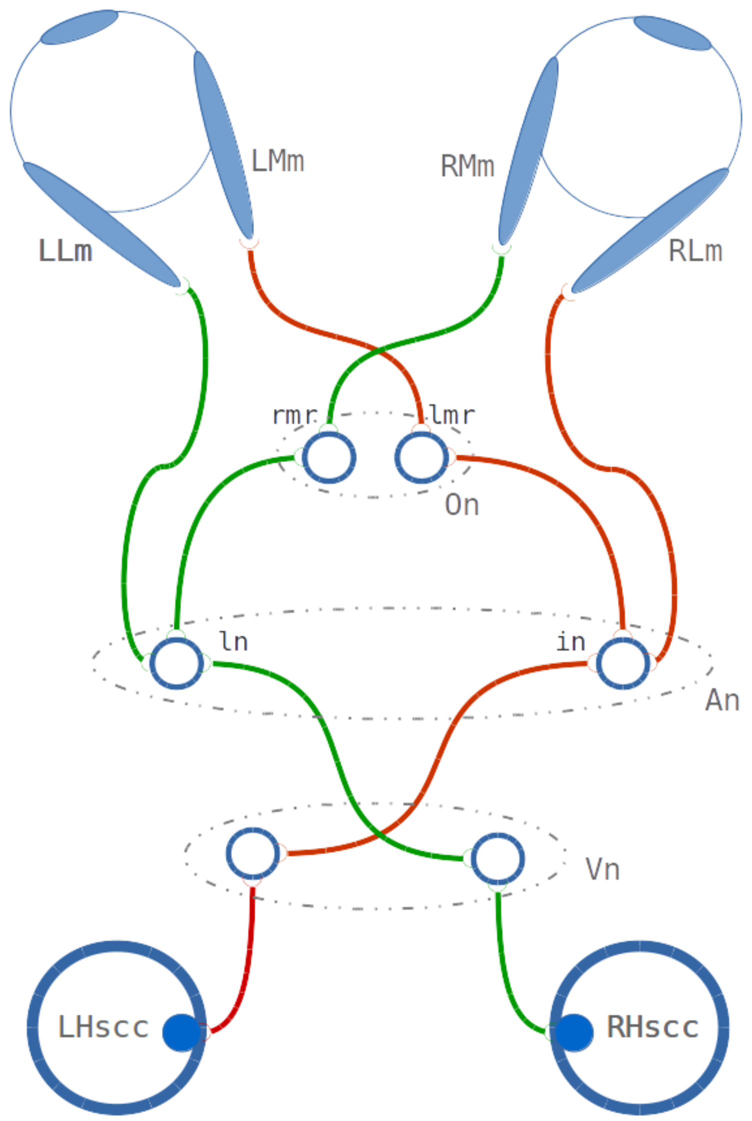
NEST implementation

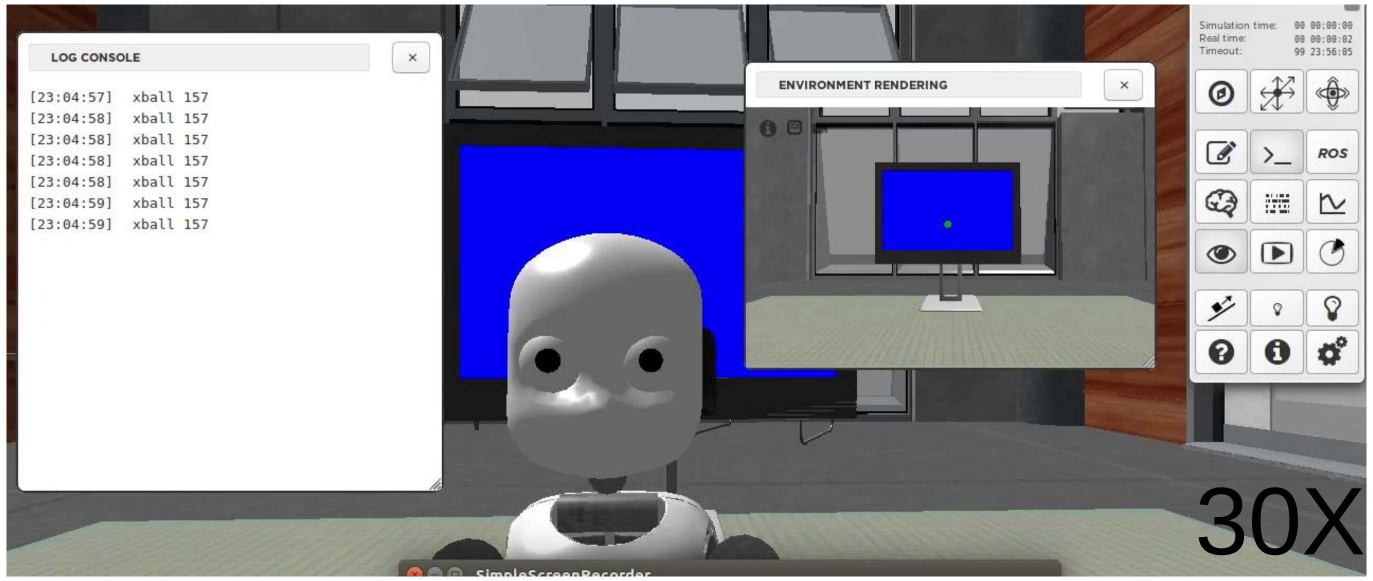
$$I(t) = G_H \cdot HV(t) - G_A \cdot X_A(t) + I_{bias} + \sigma\epsilon(t)$$

$$\frac{dV}{dt} = \frac{V(t) + I(t)}{\tau_V}$$

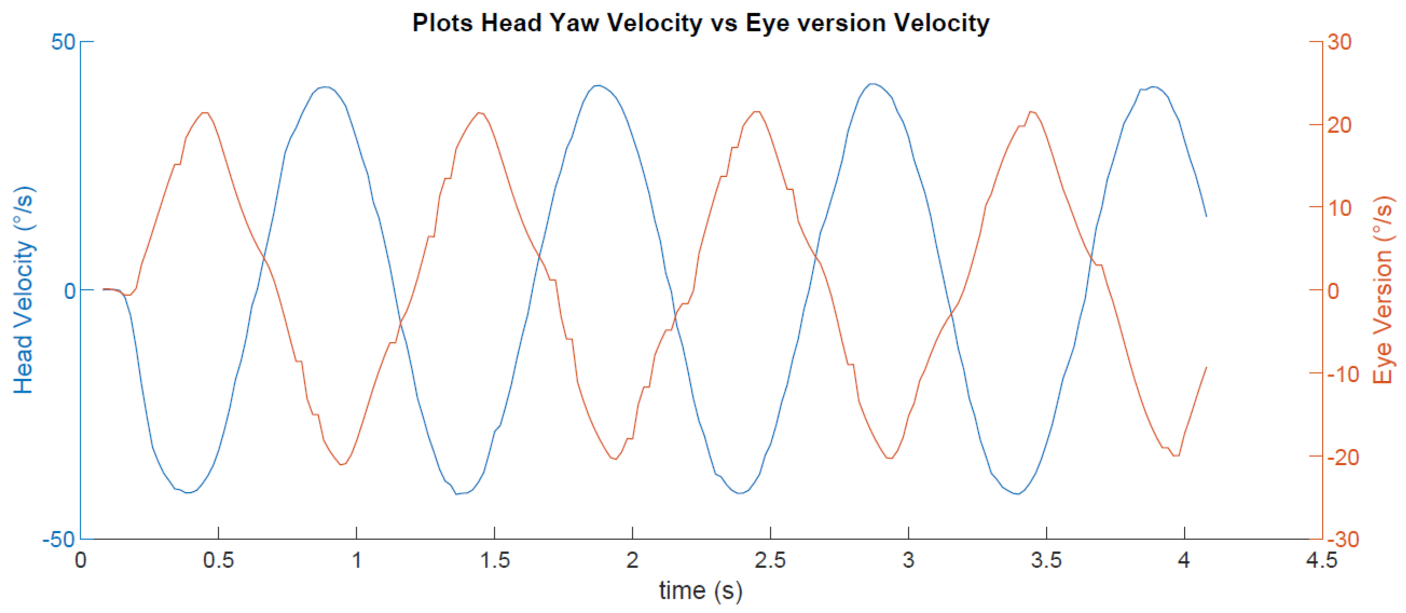
Neuromorphic model of vestibular afferents

To test the effectiveness of the model, a complete spiking network implementing the VOR circuit was for the iCub robot.



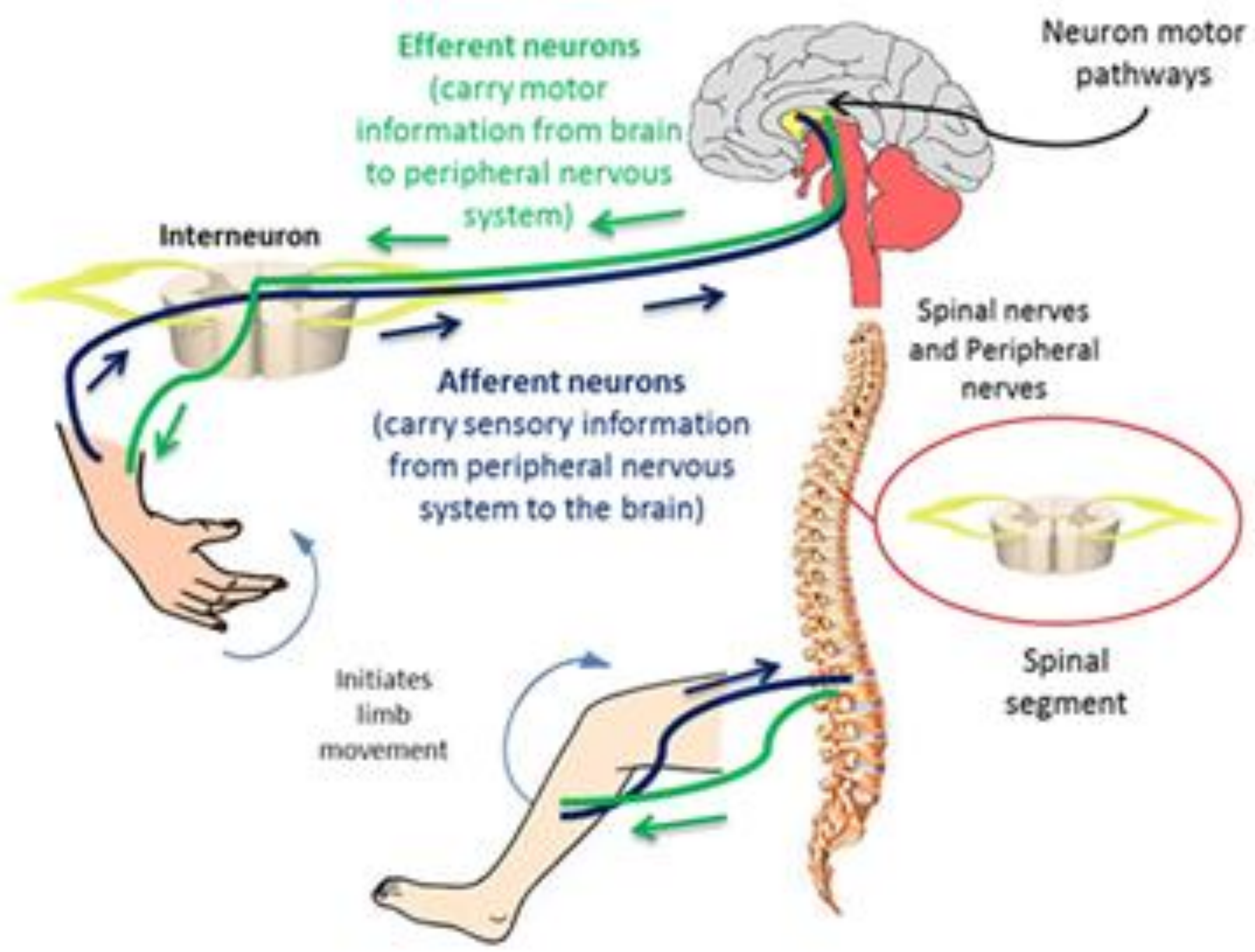


```
[23:04:57]  xball 157
[23:04:58]  xball 157
[23:04:58]  xball 157
[23:04:58]  xball 157
[23:04:58]  xball 157
[23:04:59]  xball 157
[23:04:59]  xball 157
```



Robot-brain connection through a spinal cord model

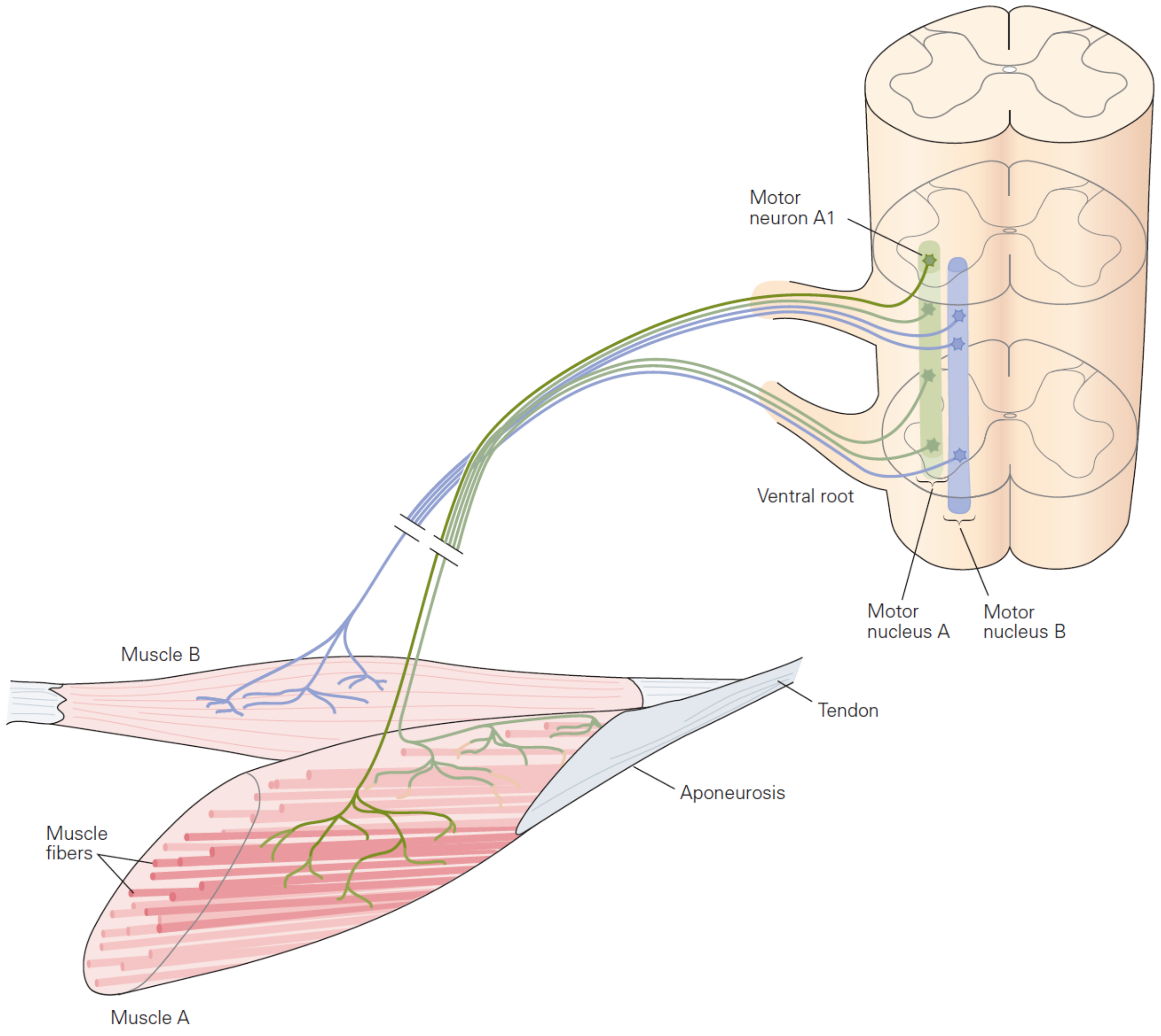
In most animals, motor commands from the brain cortex are not directly sent to the muscles, but they are transmitted through a series of hierarchically organized neural circuits. At the lowest level of this hierarchy lies the spinal cord.

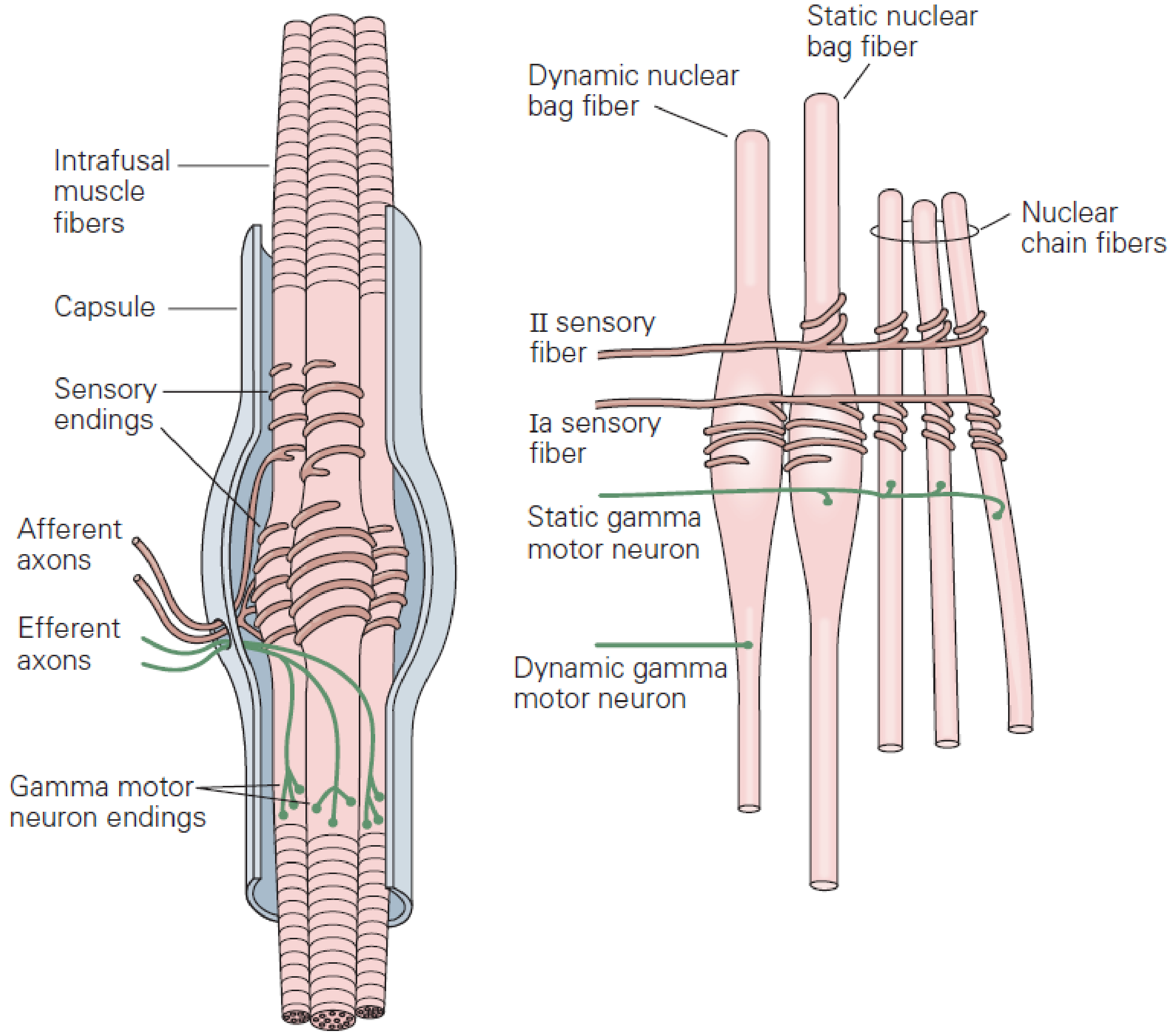


Therefore, we can think of implementing a spinal cord model that performs the translation of proprioceptive feedback and the generation of motor commands.

Robot-brain connection through a spinal cord model

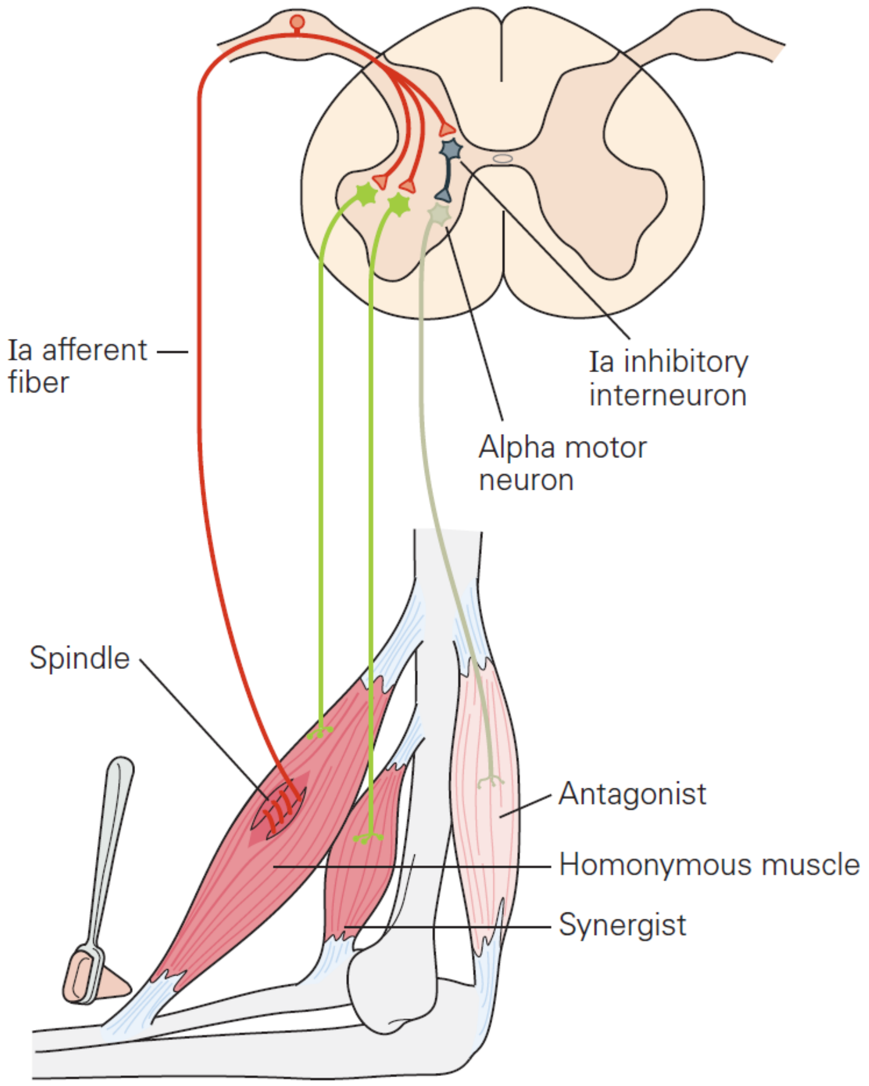
The spinal cord contains *α-motoneurons* that directly activate the muscle fibres, as well as sensory feedback endings such as *Ia* and *II* afferents from *muscle spindles*.

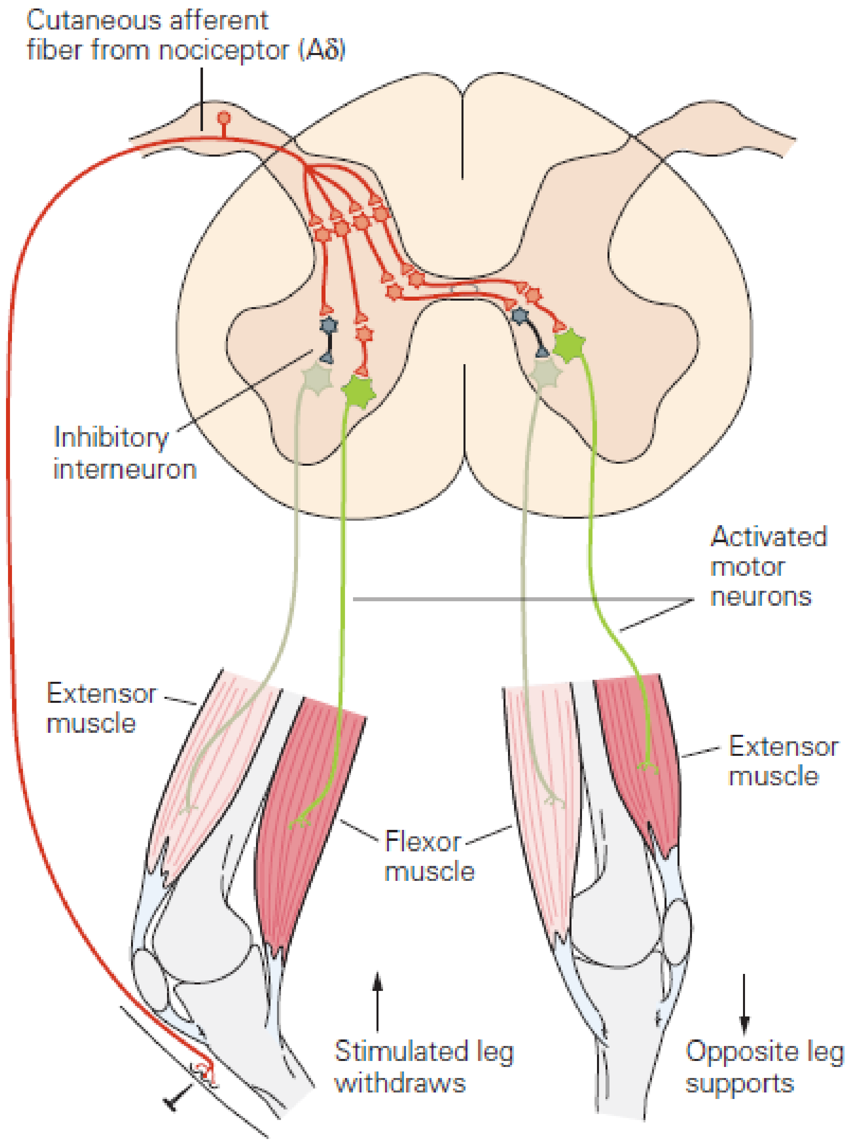




Robot-brain connection through a spinal cord model

The spinal cord is not only responsible for the activation of muscles and for the forwarding of proprioceptive information, but it also includes many local circuits for the generation of reflexes.



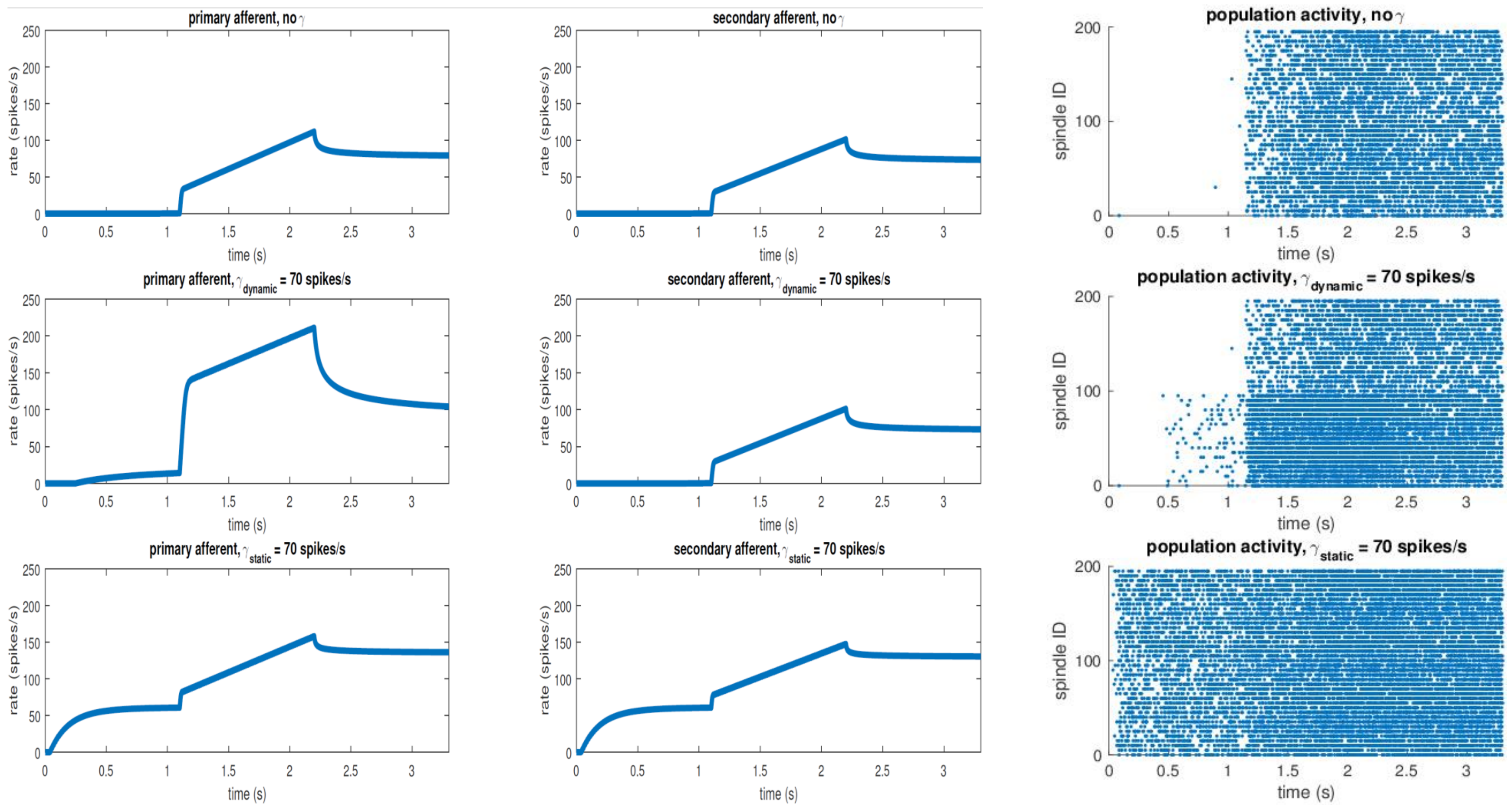


Robot-brain connection through a spinal cord model

We started by implementing in NEST (and on SpiNNaker) a bioinspired model of muscle spindle that simulates Ia and II afferent activities during a muscle stretch.

$$rate \propto T$$

$$\frac{dT}{dt} = \mathrm{f}\left(\mathrm{L}, \frac{dL}{dt}, \gamma_{st}, \gamma_{dyn}\right)$$

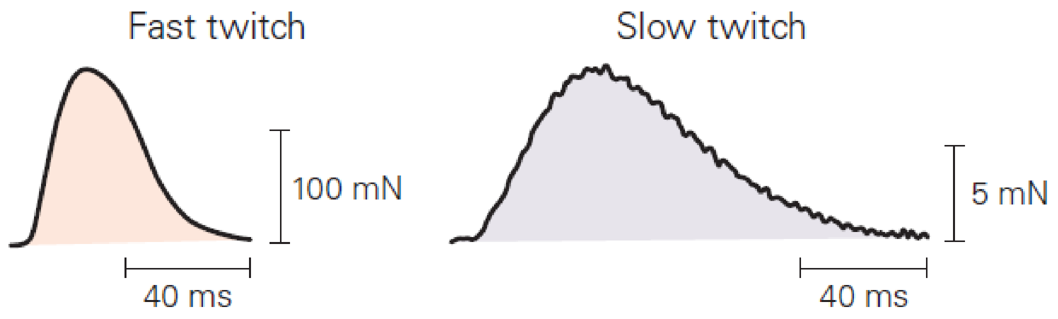


Vannucci, Lorenzo, Egidio Falotico, and Cecilia Laschi. "Proprioceptive Feedback through a Neuromorphic Muscle Spindle Model." Frontiers in Neuroscience 11 (2017): 341.

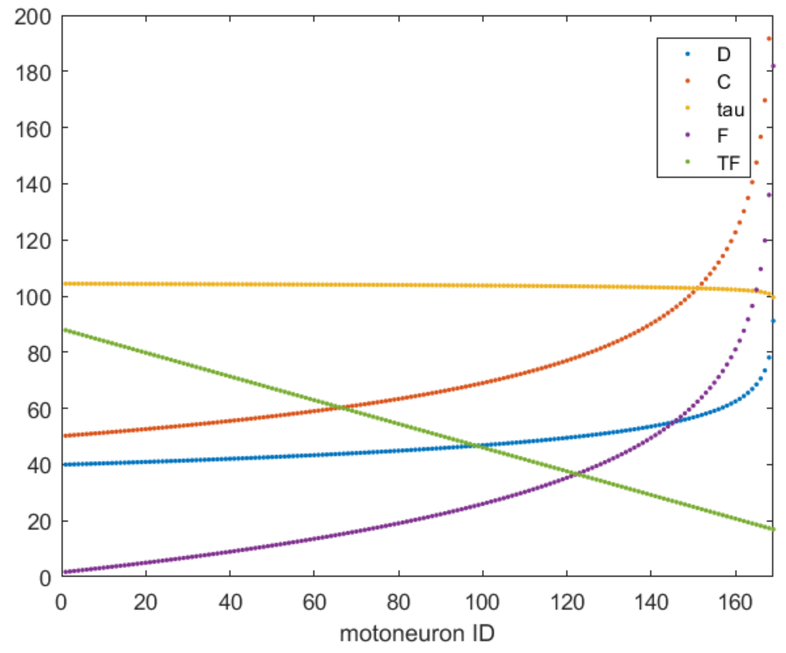
Robot-brain connection through a spinal cord model

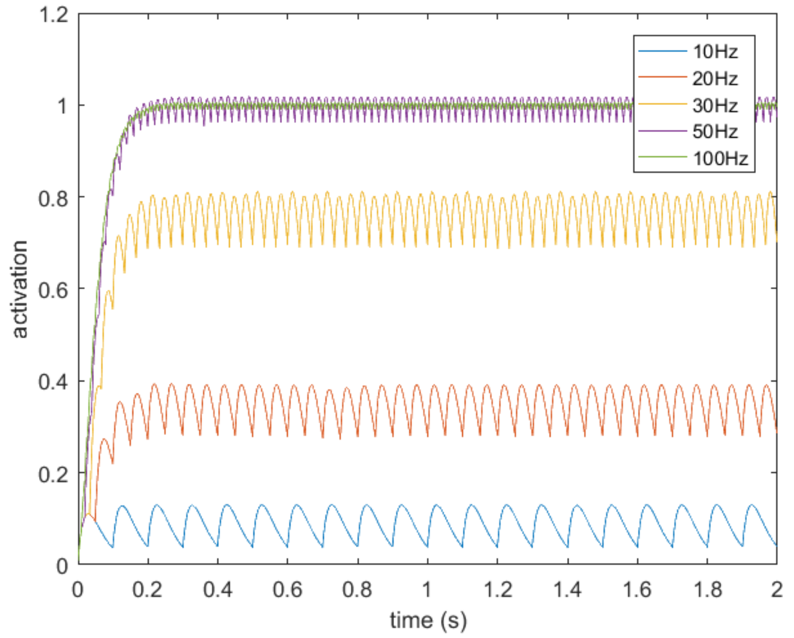
We then implemented in NEST a muscle activation model that includes motoneurons recruitment and twitches integration.

- motoneurons are activated from the weakest to the strongest
- each activation produces a twitch of some fibres



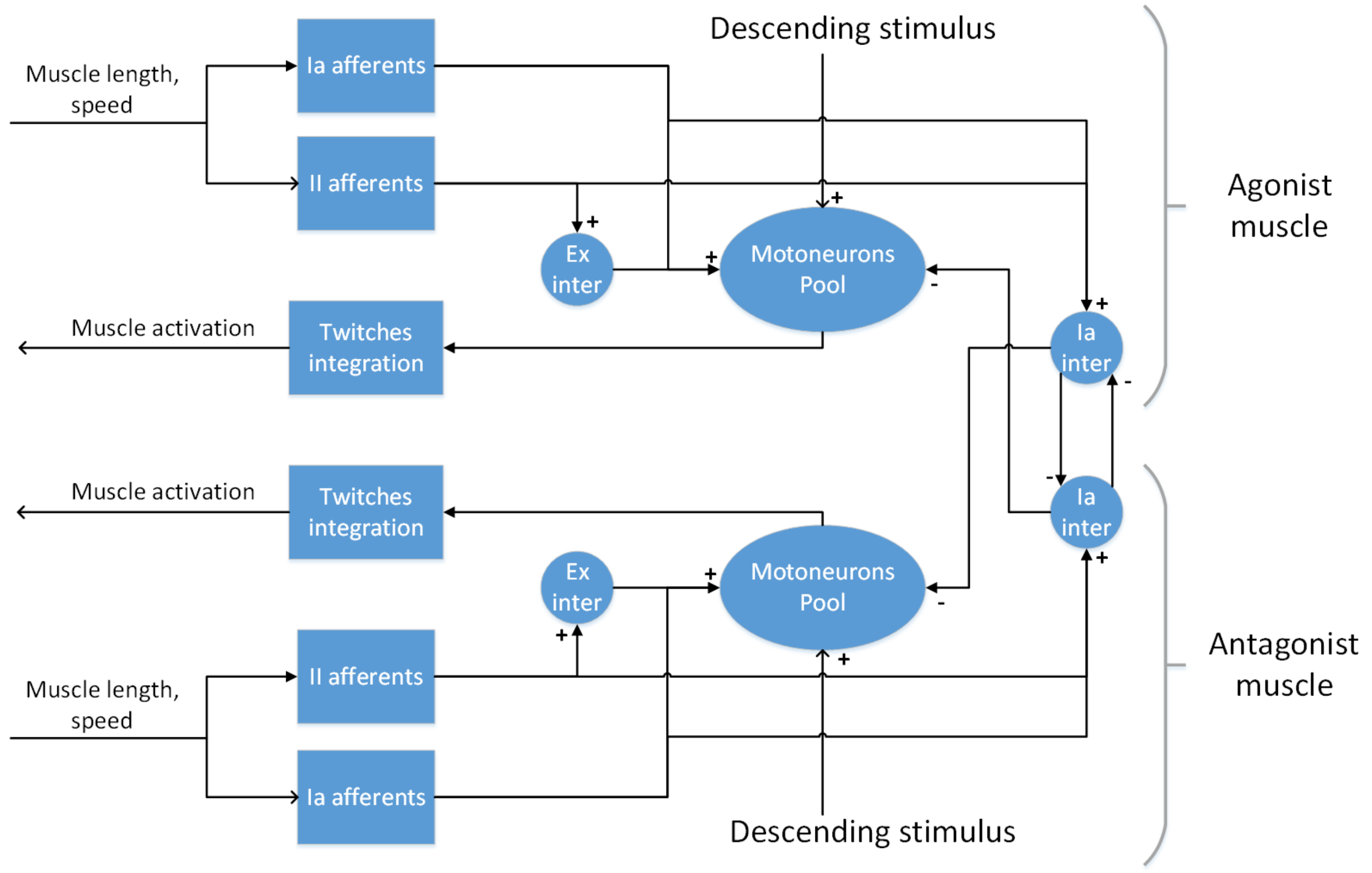
- all the twitches are summed up to compute the total muscle activation
- output can be normalized between 0 and 1

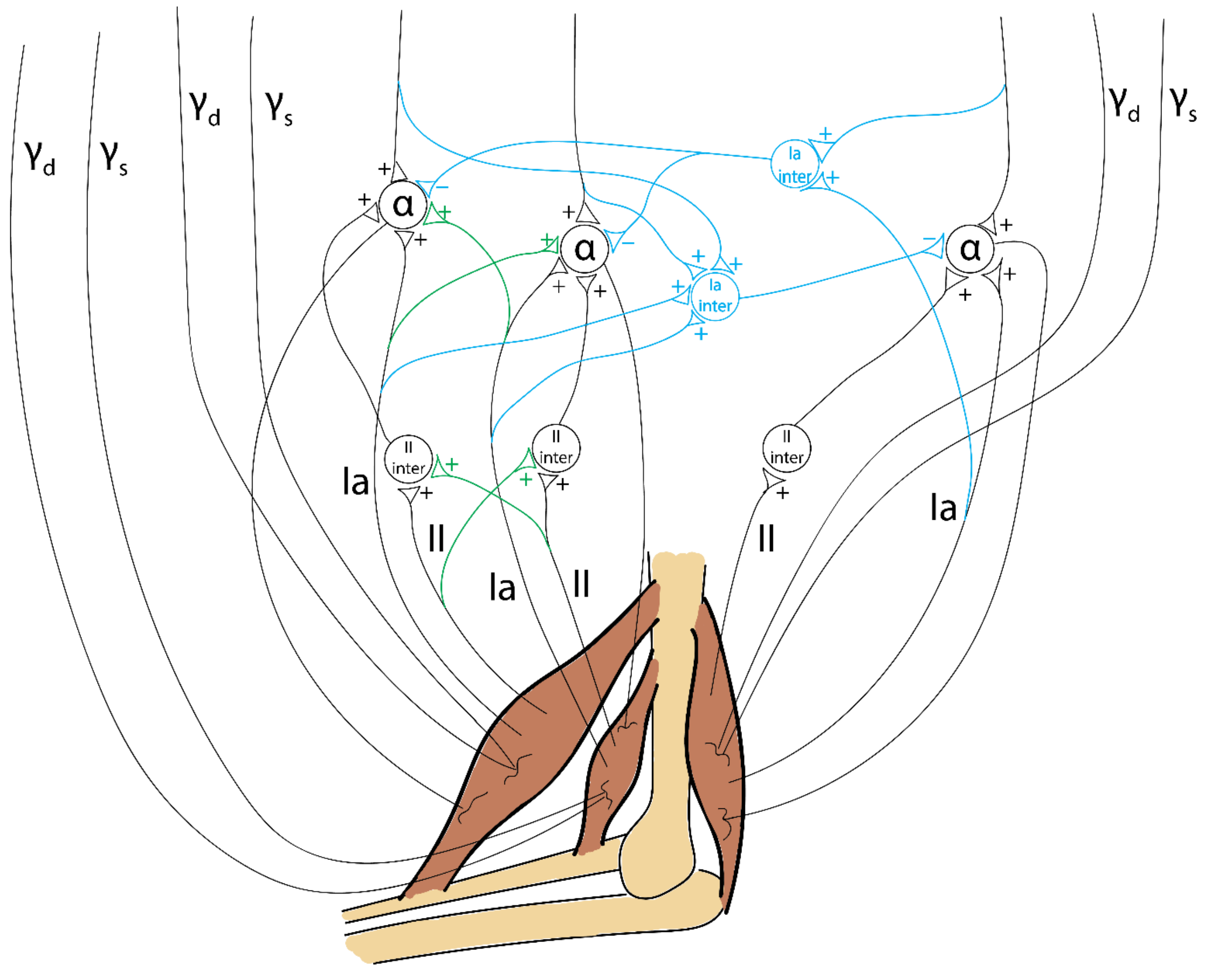




Robot-brain connection through a spinal cord model

Once we have the basic components we can assemble a fairly complete spinal cord model.

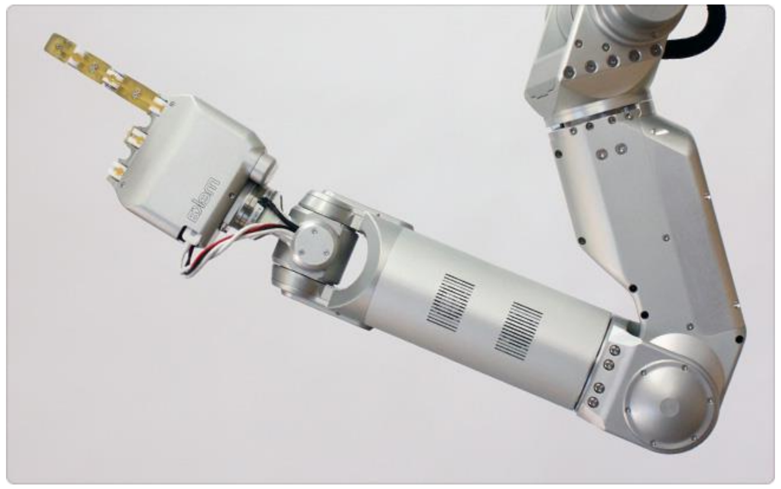




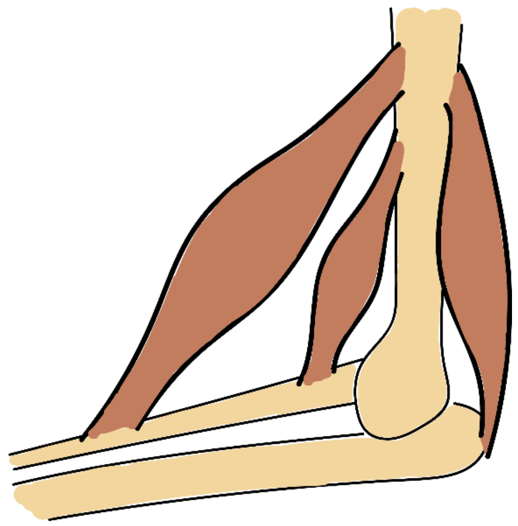
But we don't know yet how to connect it to a robot…

Robot-brain connection through a spinal cord model

In order to connect it to a robot, the more natural way is to add musculoskeletal system to the robot. Let’s consider a single joint of the robot (elbow joint).



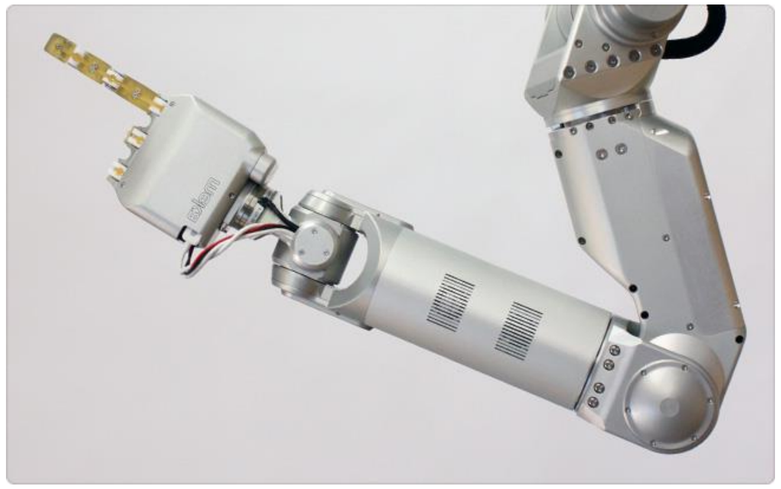
+



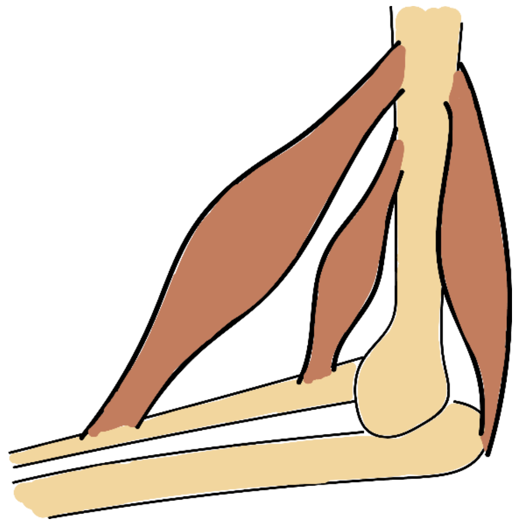
+

Robot-brain connection through a spinal cord model

In order to connect it to a robot, the more natural way is to add musculoskeletal system to the robot, via a simulation. Let's consider a single joint of the robot (elbow joint).

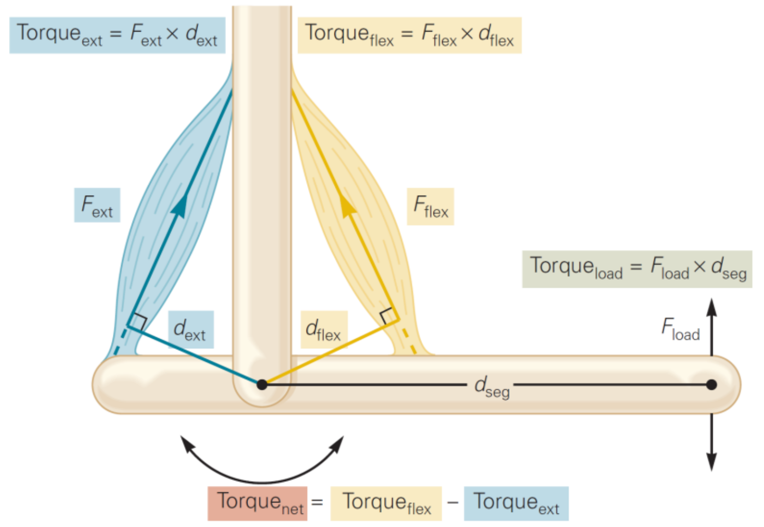


+



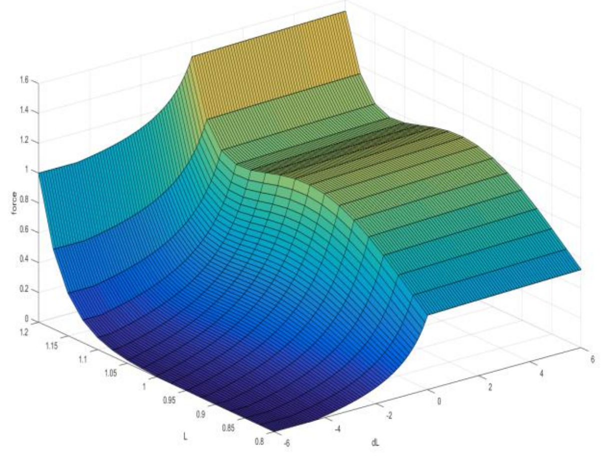
+

- joint torque from muscle forces
- muscle lengths from joint angle



$\text{Torque}_{ext} = F_{ext} \times d_{ext}$
$\text{Torque}_{flex} = F_{flex} \times d_{flex}$
F_{ext}
F_{flex}
$\text{Torque}_{load} = F_{load} \times d_{seg}$
d_{ext}
d_{flex}
F_{load}
d_{seg}
$\text{Torque}_{net} = \text{Torque}_{flex} - \text{Torque}_{ext}$

- muscle forces computed from activations via a Hill model
- muscle lengths sent to spindles

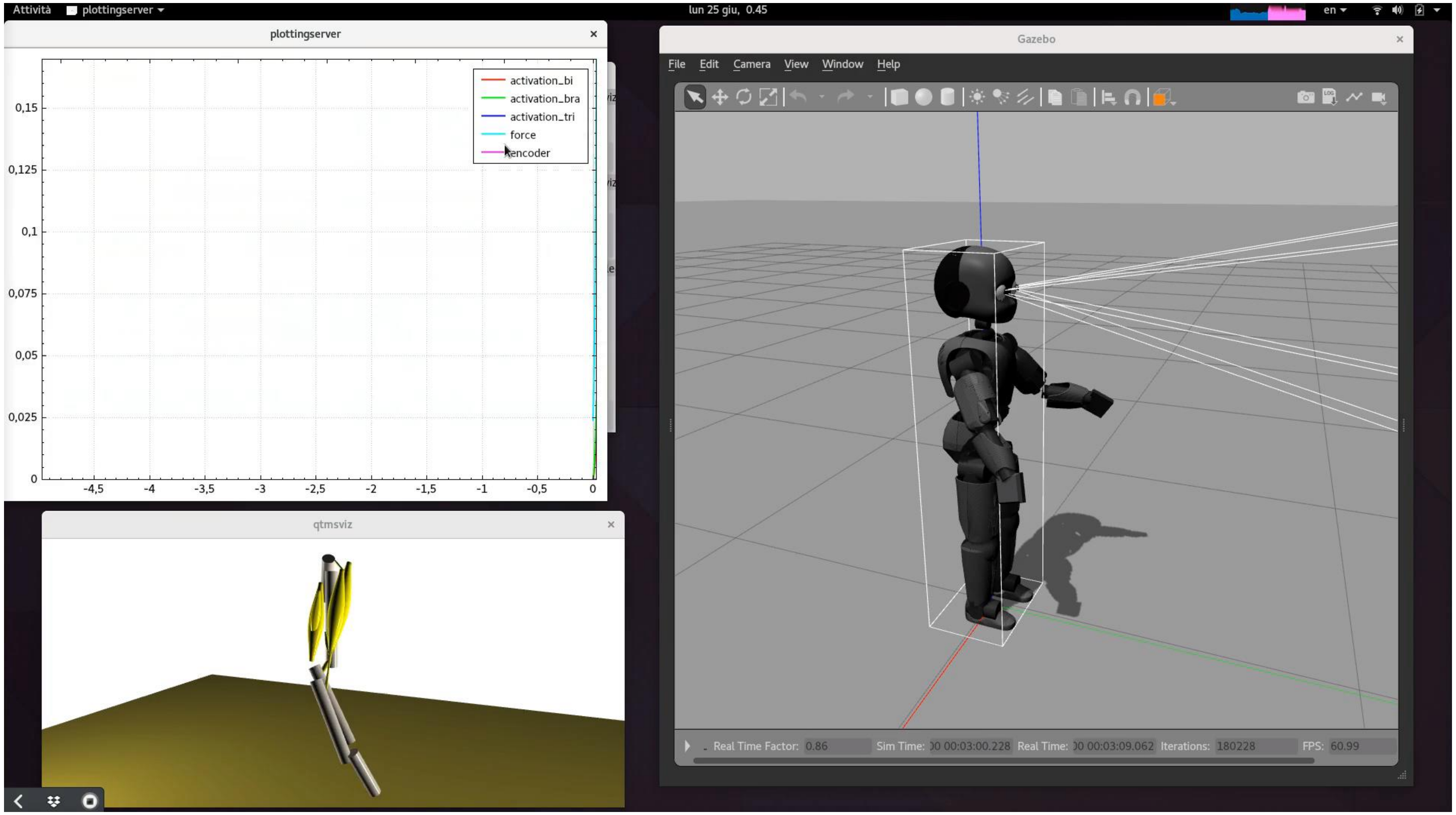


Robot-brain connection through a spinal cord model

If the robot has already muscle like-actuators, we do not need to employ the musculoskeletal simulation.

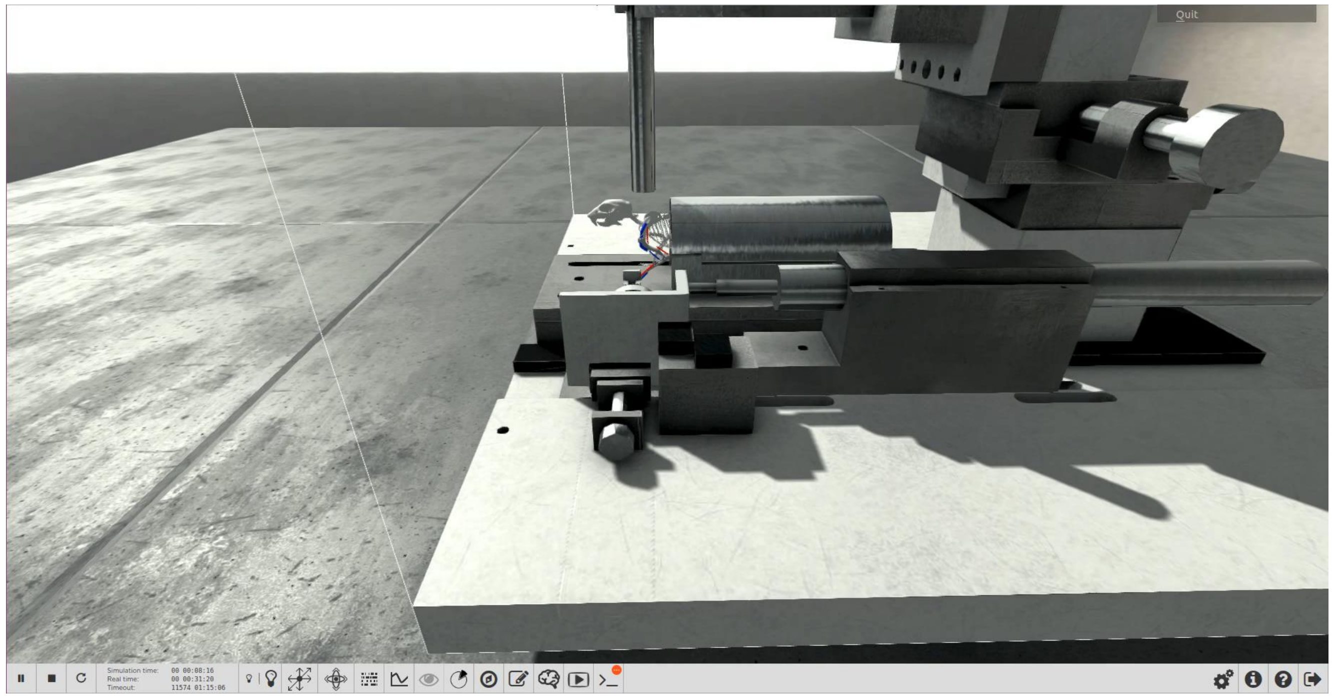
Spinal cord control example

The same muscular model, with adapted kinematic parameters, was employed to control, in a feedforward manner, the iCub elbow joint.



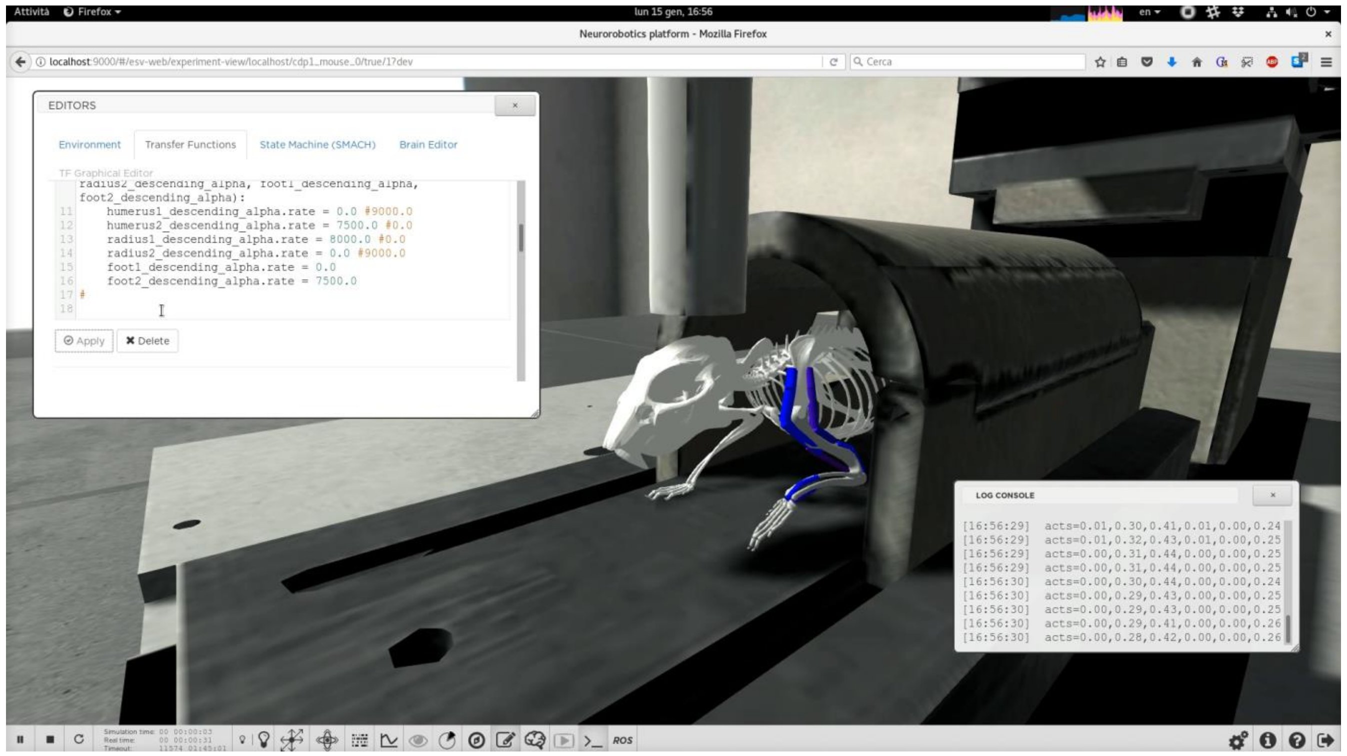
Spinal cord model as a general translation mechanism

The spinal cord model has been employed for (partially) reproducing real neuroscientific experiments.



←Motor rehabilitation experiment

Reaching experiment→

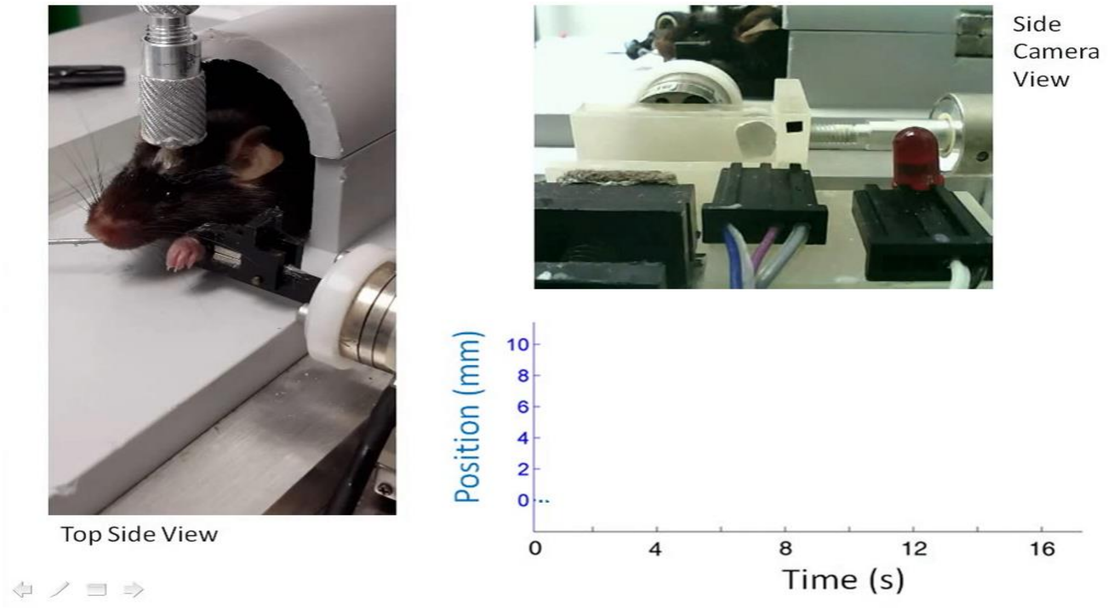


```
    radius2_descending_alpha, foot1_descending_alpha,
    foot2_descending_alpha):
        humerus1_descending_alpha.rate = 0.0 #9000.0
        humerus2_descending_alpha.rate = 7500.0 #0.0
        radius1_descending_alpha.rate = 8000.0 #0.0
        radius2_descending_alpha.rate = 0.0 #9000.0
        foot1_descending_alpha.rate = 0.0
        foot2_descending_alpha.rate = 7500.0
#
```

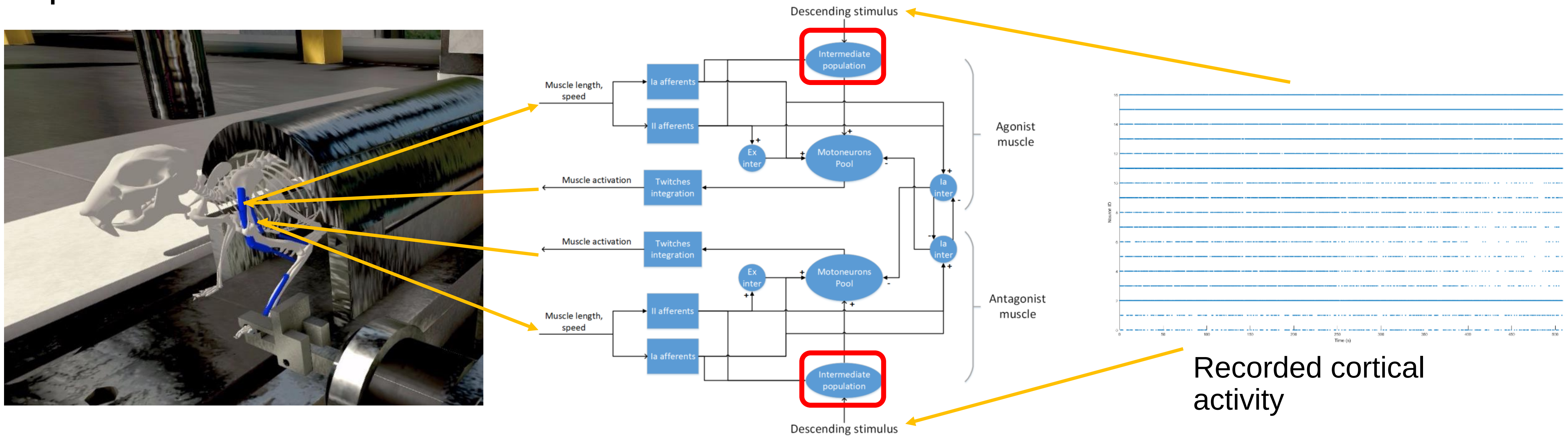
```
[16:56:29]  acts=0.01,0.30,0.41,0.01,0.00,0.24
[16:56:29]  acts=0.01,0.32,0.43,0.01,0.00,0.25
[16:56:29]  acts=0.00,0.31,0.44,0.00,0.00,0.25
[16:56:29]  acts=0.00,0.31,0.44,0.00,0.00,0.25
[16:56:30]  acts=0.00,0.30,0.44,0.00,0.00,0.24
[16:56:30]  acts=0.00,0.29,0.43,0.00,0.00,0.25
[16:56:30]  acts=0.00,0.29,0.43,0.00,0.00,0.25
[16:56:30]  acts=0.00,0.29,0.41,0.00,0.00,0.26
[16:56:30]  acts=0.00,0.28,0.42,0.00,0.00,0.26
```

Post-stroke rehabilitation simulation

Learning of a forelimb pulling task, then study of motor task re-training in rodent model after induction of photothrombotic stroke with simultaneous intracranial recording.



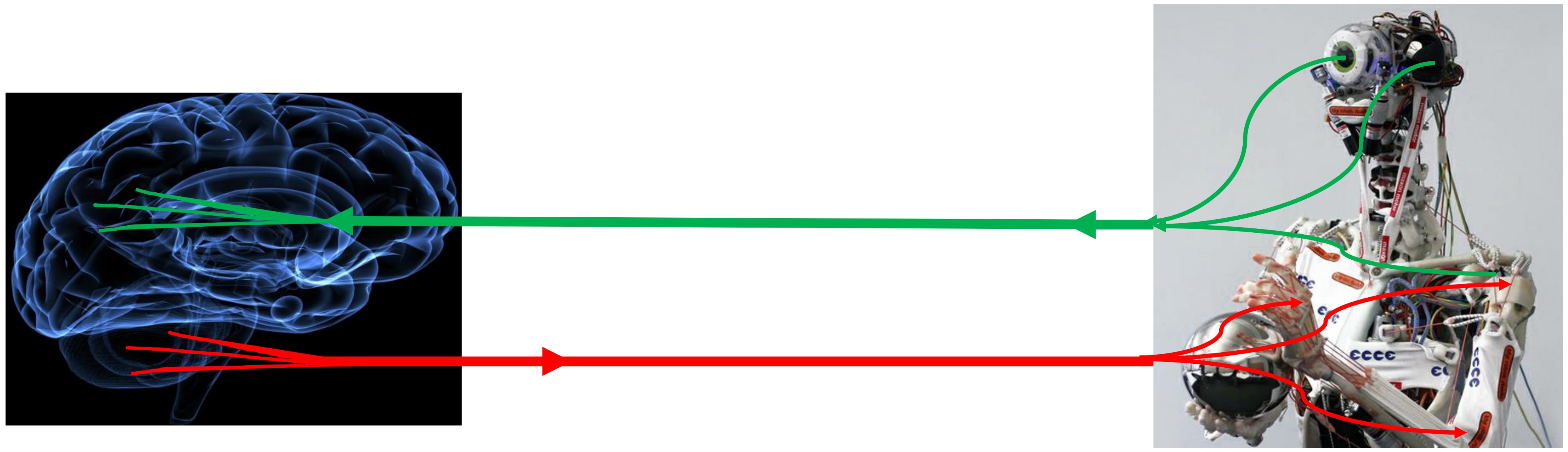
Reproduction *in silico*:



Recorded cortical activity

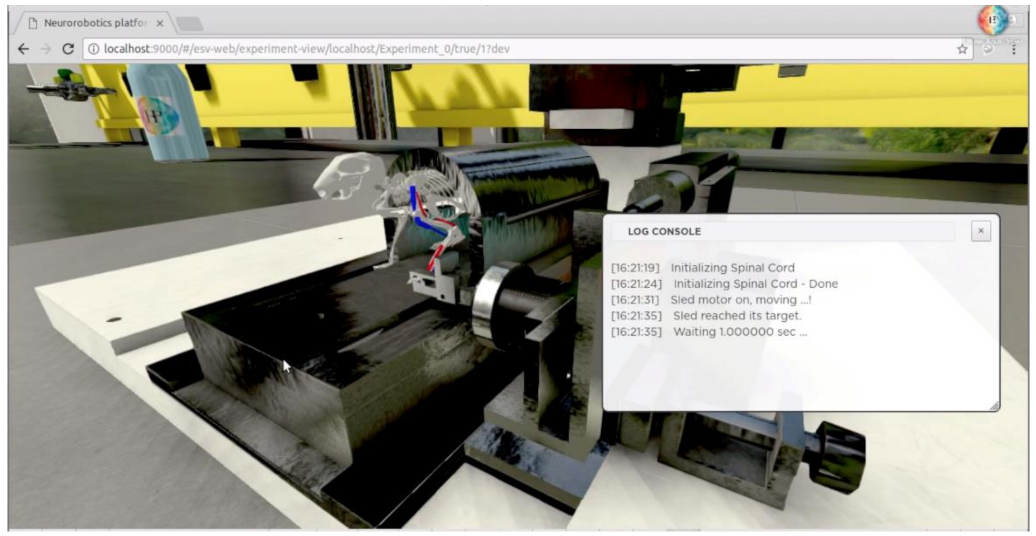
The Neurorobotics Platform

The Neurorobotics Platform is a simulation toolkit that aims at providing synchronized neural and robotic simulations, and data transfer from robot sensors/actors to brain areas and vice versa.

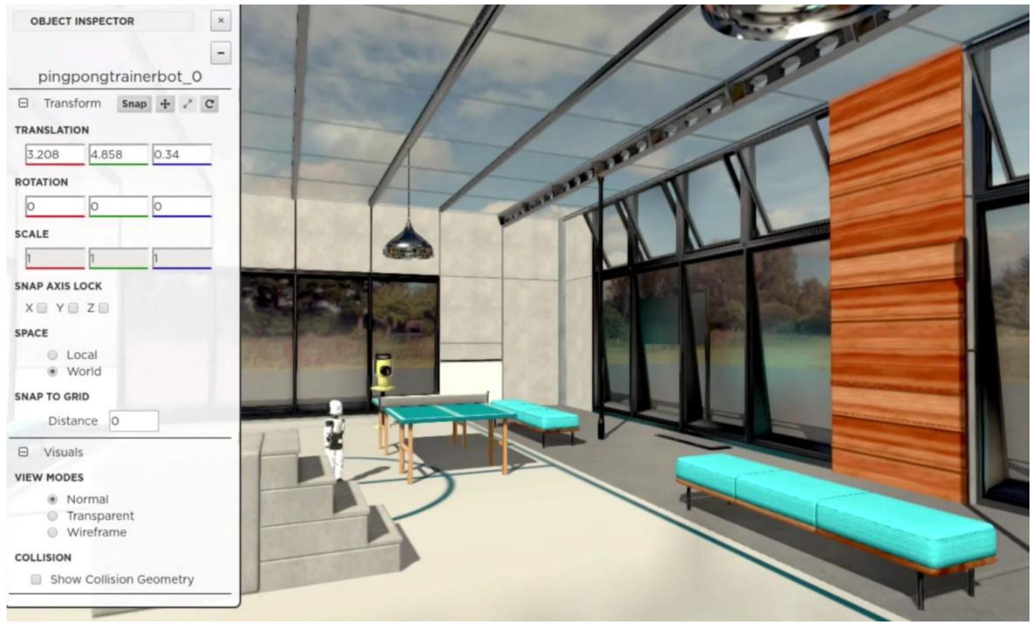


The Neurorobotics Platform

- transfer functions connect the neural and physical simulators
- neural simulation is provided by NEST, through the PyNN interface
- physical and robotic simulations are provided by Gazebo, via the ROS middleware
- web-based frontend for visualization and environment creation


Neurorobotics platfor
localhost:9000/#/esv-web/experiment-view/localhost/Experiment_0/true/1?dev
LOG CONSOLE

```
[16:21:19]   Initializing Spinal Cord
[16:21:24]   Initializing Spinal Cord - Done
[16:21:31]   Sled motor on, moving ...!
[16:21:35]   Sled reached its target.
[16:21:35]   Waiting 1.000000 sec ...
```



pingpongtrainerbot_0

TRANSLATION 3.208 4.858 0.34
ROTATION 0 0 0
SCALE 1 1 1
SNAP AXIS LOCK X Y Z
SPACE Local [✓ World]
SNAP TO GRID Distance 0
VIEW MODES [✓ Normal] Transparent Wireframe
COLLISION Show Collision Geometry

gazebosim.org

ros.org

Neuromorphic computing resources

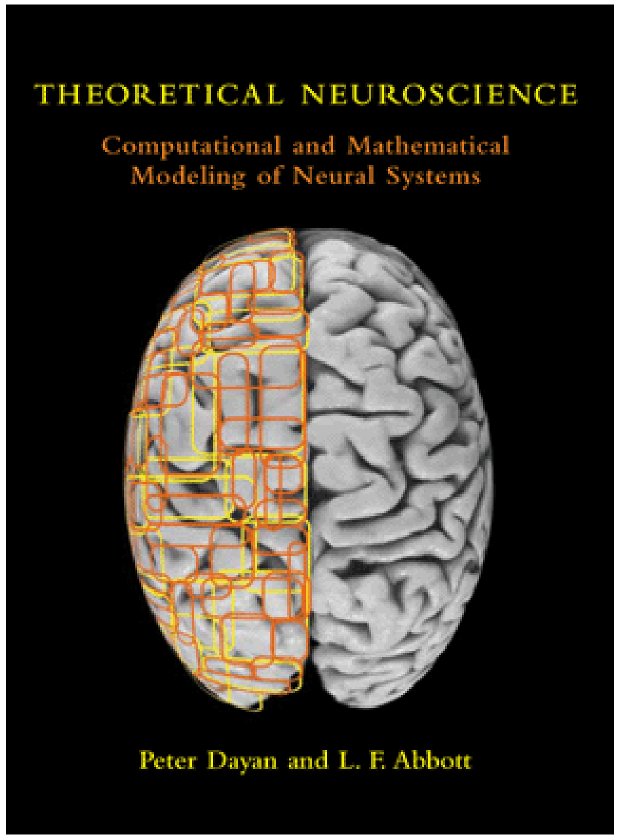
Neurology:
- "Principles of Neural Science" by Kandel et al.

Computational neuroscience:
- "Theoretical Neuroscience: computational and mathematical modeling of neural systems" by Peter Dayan and Larry Abbott
- Computational Neuroscience on Coursera

NEST:
- www.nest-simulator.org

SpiNNaker:
- spinnakermanchester.github.io
- apt.cs.manchester.ac.uk/projects/SpiNNaker

Neurorobotics Platform:
- neurorobotics.net


THEORETICAL NEUROSCIENCE
Computational and Mathematical Modeling of Neural Systems
Peter Dayan and L. F. Abbott

Contacts:
lorenzo.vannucci@santannapisa.it