CN510: Principles and Methods of Cognitive and Neural Modeling

Neurosimulators Overview
KInNeSS Design

Lecture 25

Instructor: Anatoli Gorchetchnikov <anatoli@bu.edu>
Teaching Fellow: Rob Law <nosimpler@gmail.com>

How to Avoid (Excessive) Coding?

Brian – spiking simulator in Python

- Allows quick simulation coding by writing equations in standard math notation

NEST – parallel large scale simulator primarily for point neurons

- Model description includes equations and solution methods

NEURON – simulator for morphologically correct neurons and networks of these neurons

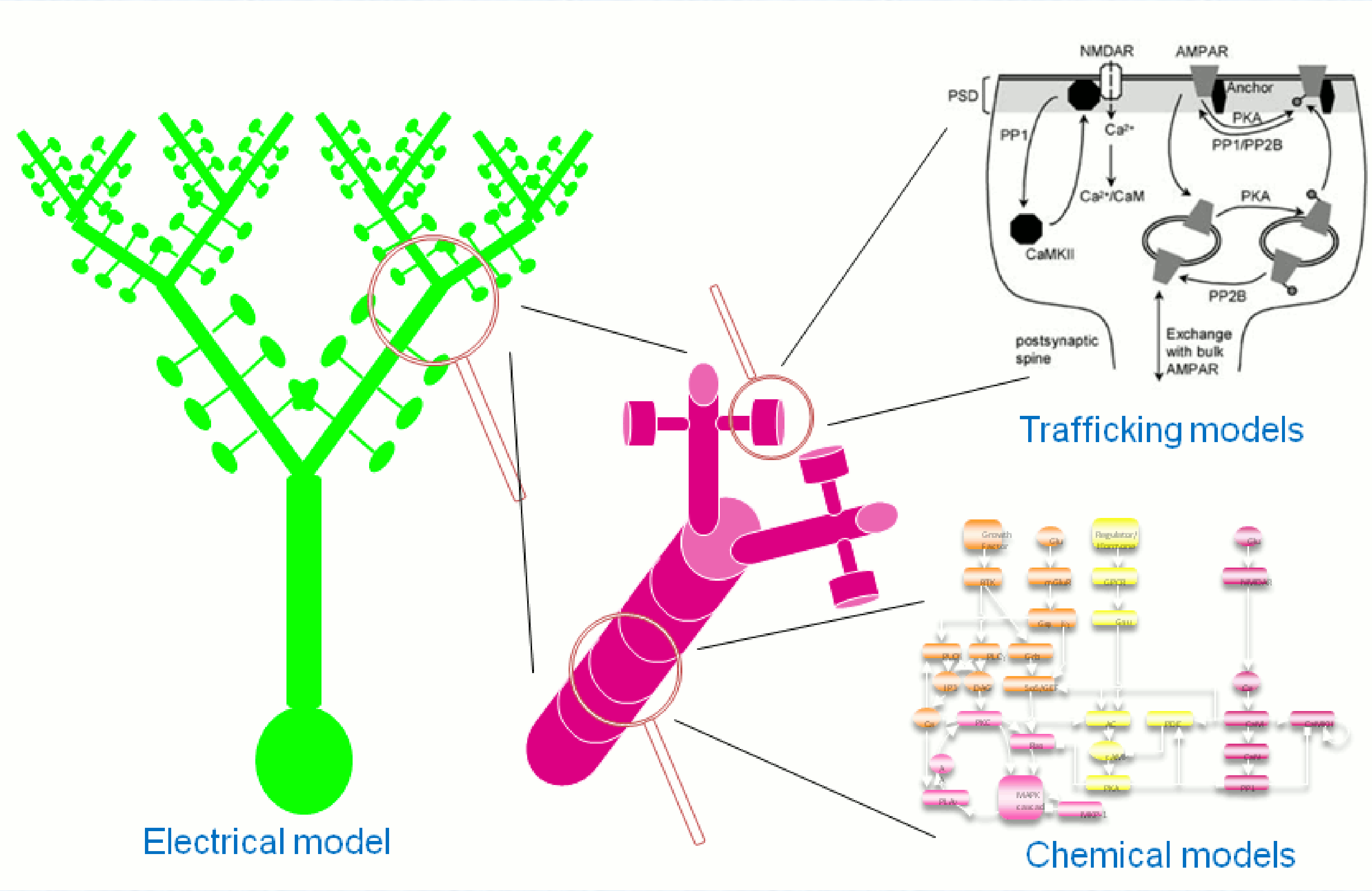
GENESIS – simulator for multi-compartmental neurons and networks of these neurons

PyNN – multi-simulator front end written in Python and allowing to run same model scripts on multiple simulators

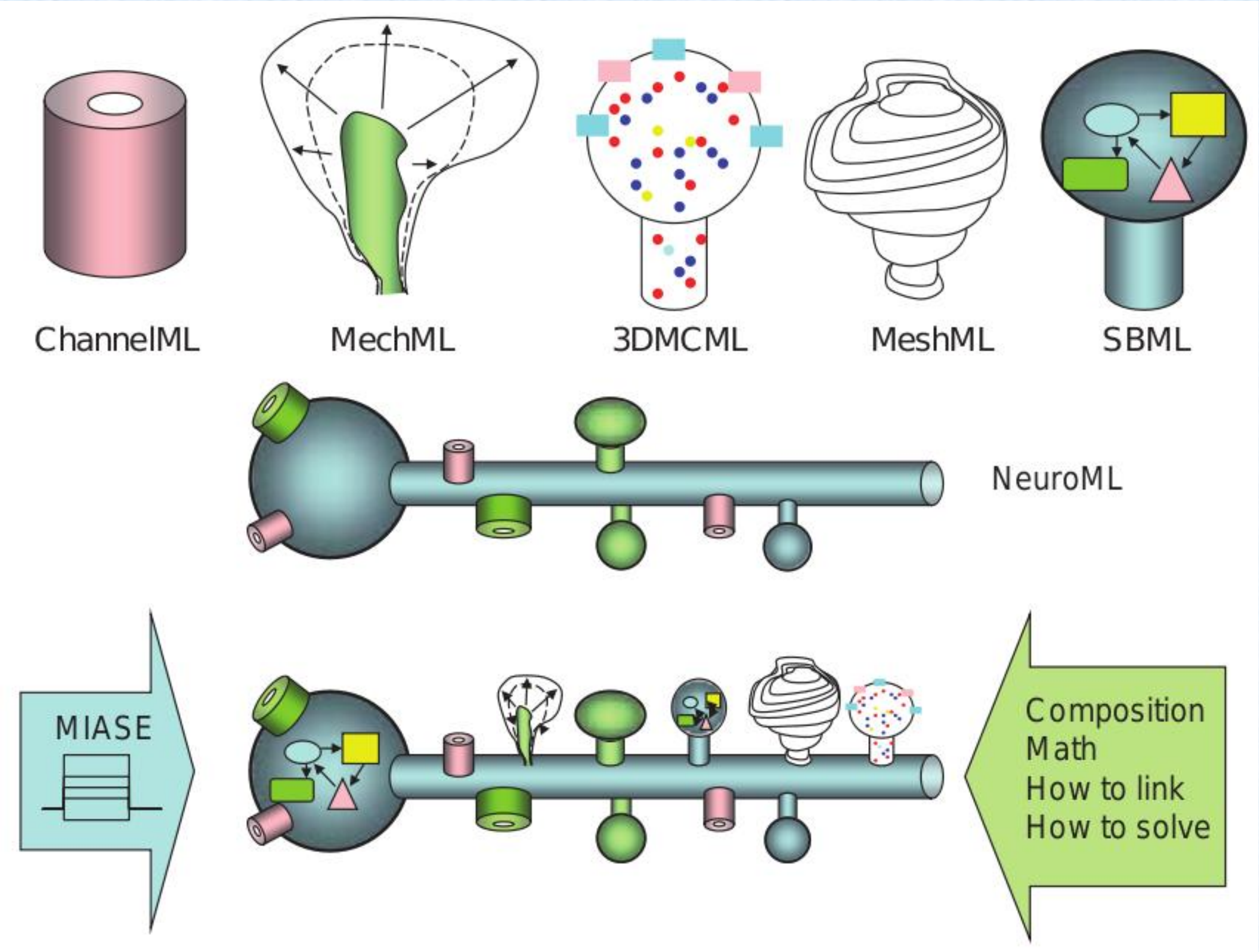
MOOSE – Multiscale Object Oriented Sim Env

MOOSE – multi-scale simulator designed to combine scales from molecules to synapses to neurons to networks

Models are scripted in Python



MOOSE – Using Multiple Standards



Other Simulation Packages

XPP – primarily designed for modeling any dynamical systems, including neural models

Cog Ex Machina – proprietary tool for simplified neural models interacting with the environment

KInNeSS – designed for large networks of few-compartmental neurons interacting with the environment

And many others that are even less known

And… 95% of neural models are still coded by hand in MatLab or other language of choice by researchers

NEURON

Simulates complex branched cell anatomy with various channel types and distributions

Network models are also possible and become more efficient

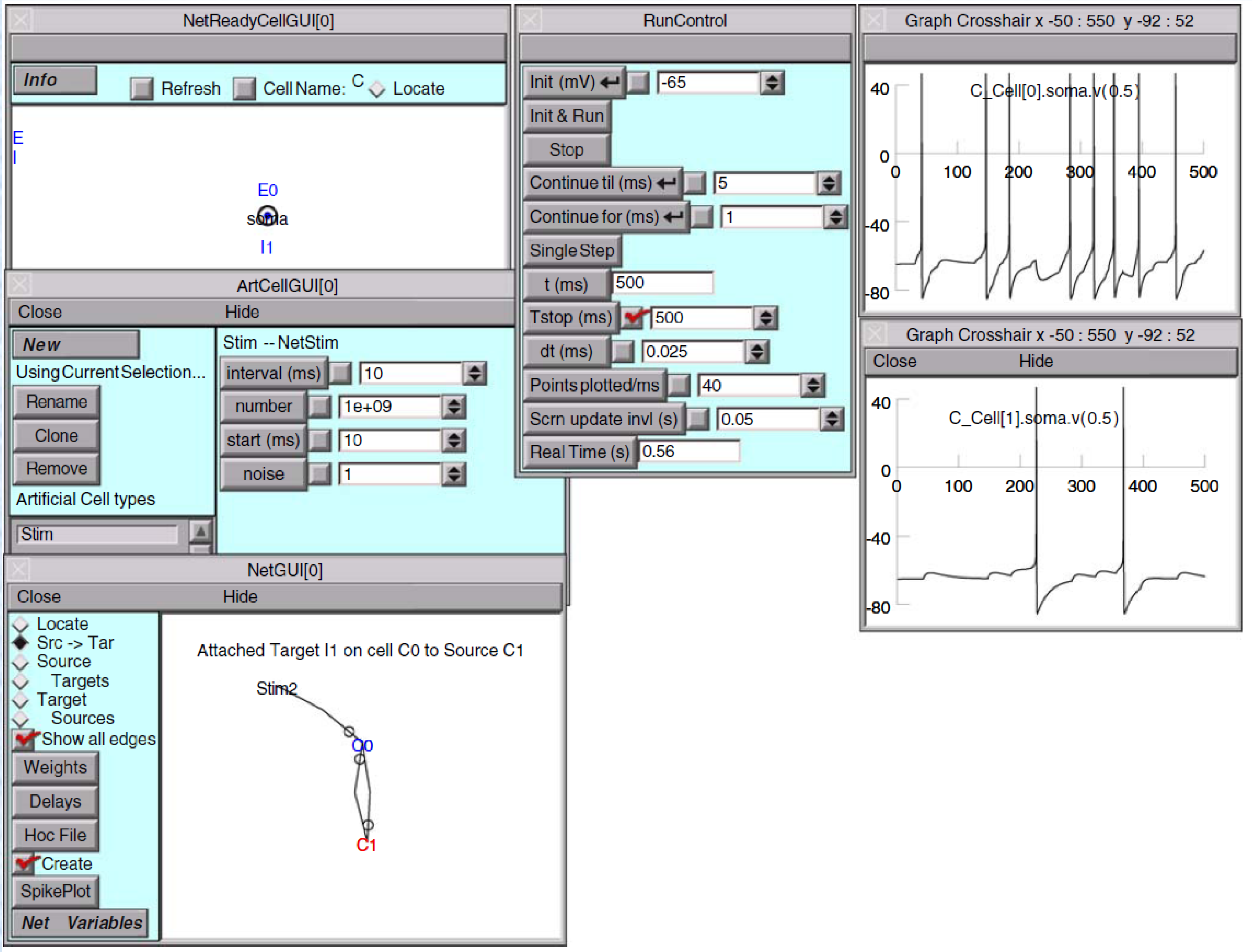
Hides the compartmentalization from the user: uses morphology definitions and internal criteria for discretization

But! Uses the same central difference approximation as compartmental simulators, so does not provide any extra benefits

Allows 0 delay spike delivery

Uses custom interpreted language for model descriptions

NEURON GUI



NetReadyCellGUI[0]

Cell Name: C

ArtCellGUI[0]

Stim -- NetStim

interval (ms) 10

number 1e+09

start (ms) 10

noise 1

Artificial Cell types

Stim

NetGUI[0]

Attached Target I1 on cell C0 to Source C1

RunControl

Init (mV) -65

Continue til (ms) 5

Continue for (ms) 1

t (ms) 500

Tstop (ms) ☑ 500

dt (ms) 0.025

Points plotted/ms 40

Scrn update invl (s) 0.05

Real Time (s) 0.56

Graph Crosshair x -50 : 550 y -92 : 52

NEURON Scalability

Parallel execution:

- One core – one simulation: good for batch mode
- Distributed networks with gap junctions: interesting as you have to solve one system across multiple processors
- Distributed cell model: hard to set up

Primary target – supercomputers (Cray, Blue Gene, etc)

Claim linear speed-up (fig. 7b)

Primary concentration: simulator is done, need more tools for network design and analysis

GENESIS

Realistic models of neurons and biological systems based on known anatomy and physiology

For cells – conductances and their distribution

For networks – realistic projection patterns

Uses precompiled objects for model components: 125 objects

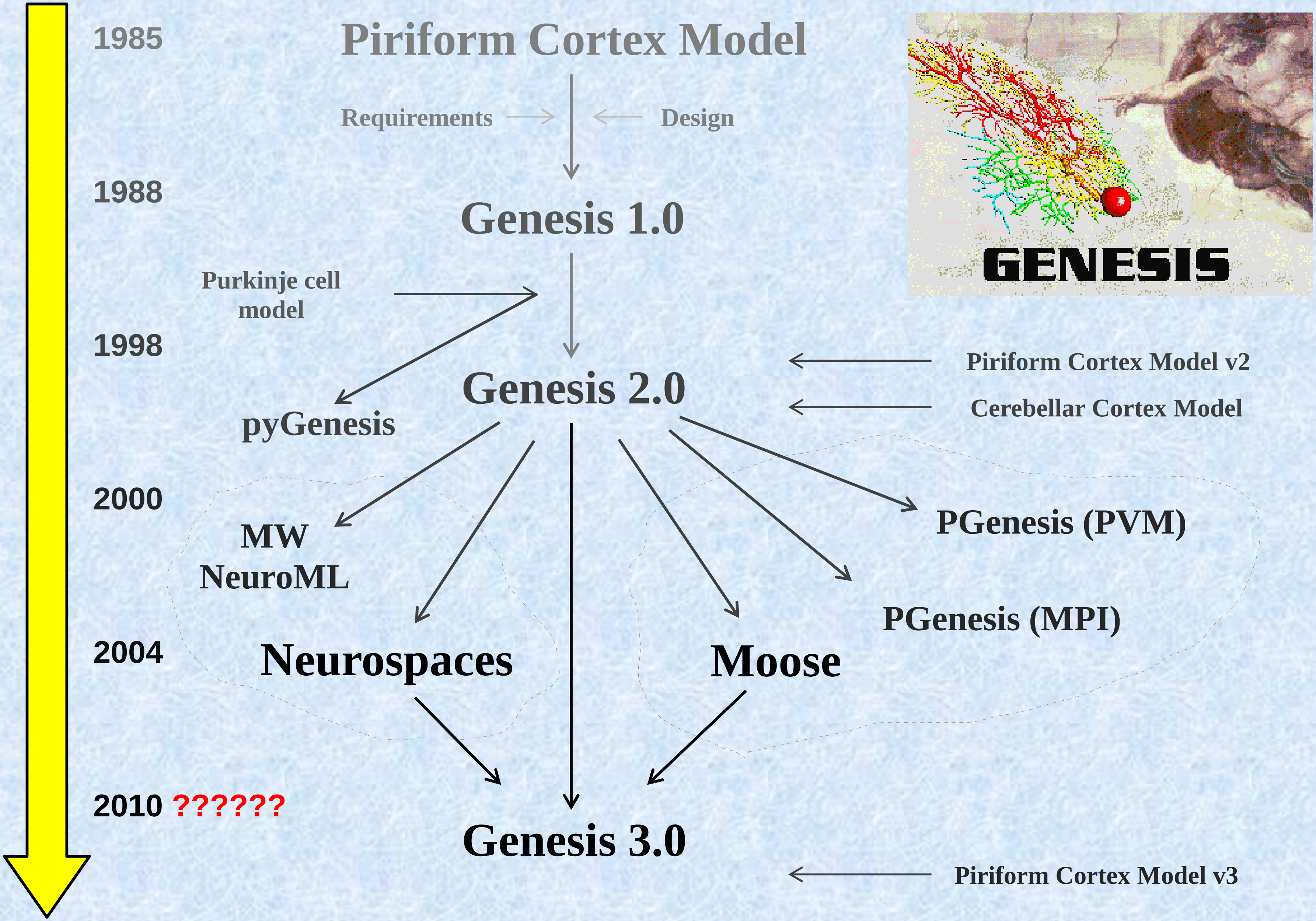
Custom interpreter language for model definitions: 268 commands

No scripting GUI builders for single cells and biochemical reactions

Next to no support for simplified models like Izhikevich and other IaF neurons

Future: GENESIS 3 in C++

History of The GENESIS Project



1985 Piriform Cortex Model

Requirements → ← Design

1988 Genesis 1.0

Purkinje cell model

1998 Genesis 2.0

pyGenesis

Piriform Cortex Model v2

Cerebellar Cortex Model

2000 MW NeuroML

PGenesis (PVM)

PGenesis (MPI)

2004 Neurospaces

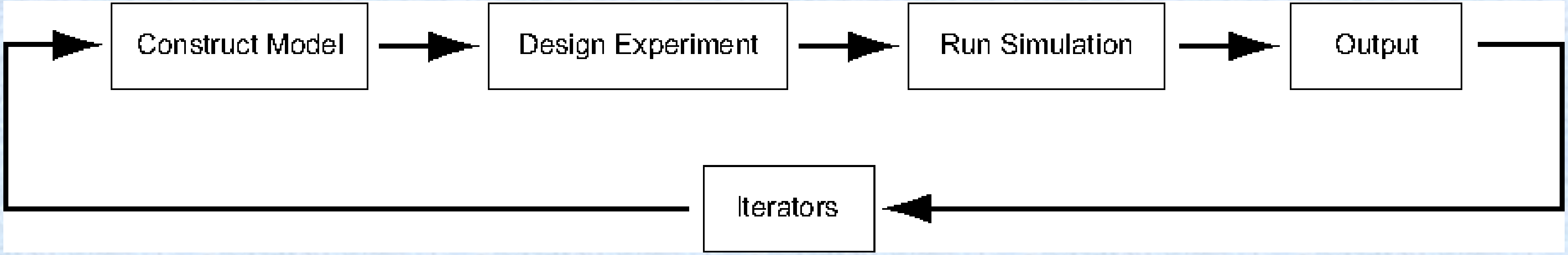
Moose

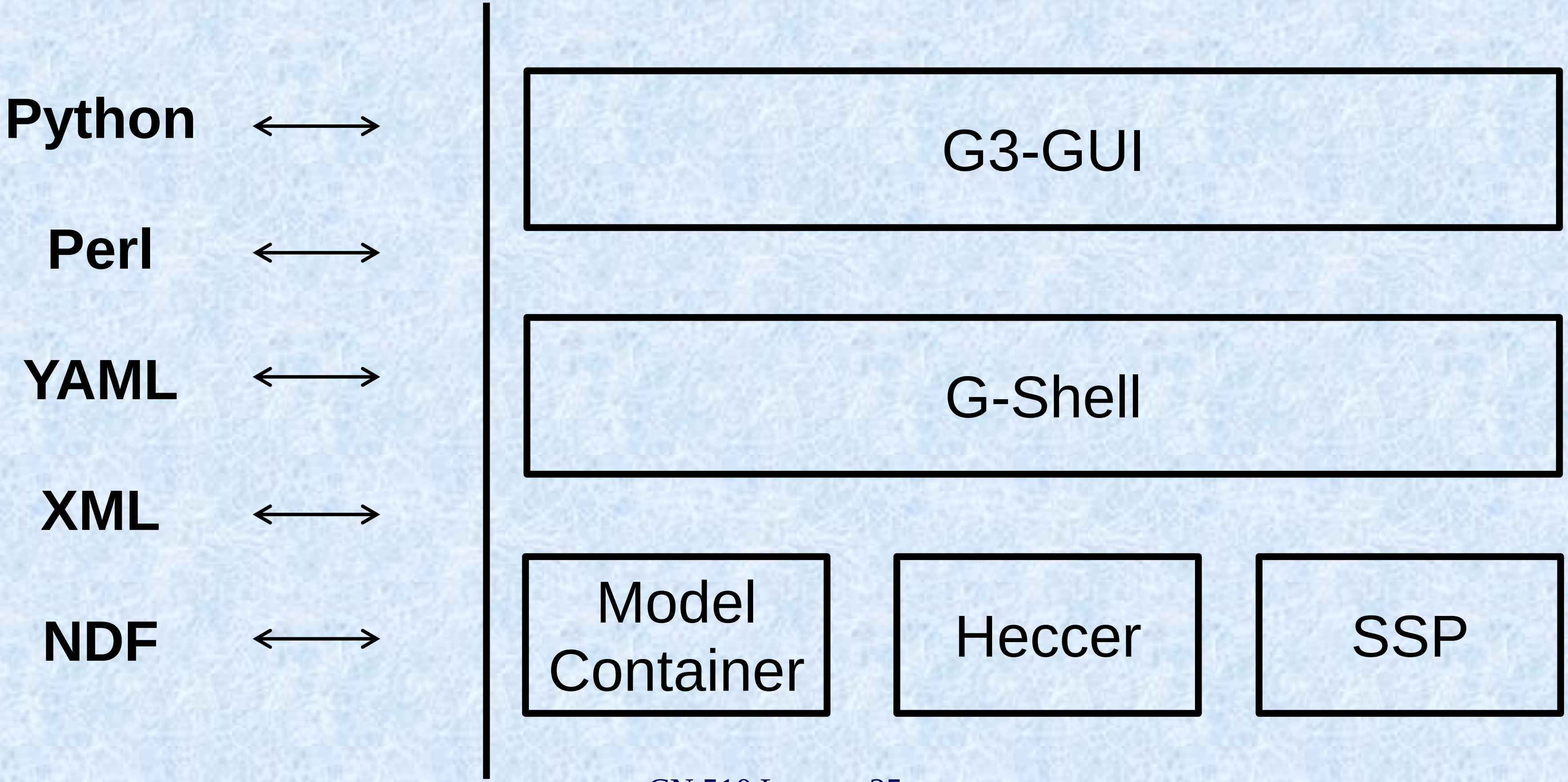
2010 ??????

Genesis 3.0

Piriform Cortex Model v3

GENESIS 3 User Workflow





Python ⟷

Perl ⟷

YAML ⟷

XML ⟷

NDF ⟷

G3-GUI

G-Shell

Model Container

Heccer

SSP

NEST

Simulation of networks with realistic sizes and complexity

Primary goal is scalability and parallelism

10^5 neurons with 10^8 synapses in 2005

One or few compartments, emphasis on efficient representation and update of synapses

Parallelizes both network construction and simulation

Pre-compiled C++ objects are combined into a model

Multi-threading and message passing

Every development is tested in a scientific project

Uses simulation language interpreter

Consider GUI useless for network simulations due to massive data arrays and procedural model specification

NEST

Communication overhead is minimized by only sending events at minimal delay intervals (not every time step)

Does not allow 0 delays, minimal delay puts upper bound on simulation time step

Synapses are stored on the receiving end

Reproducibility is achieved by splitting into virtual processes distributed across whatever machines are available

Uses exact integration (*Rotter-Diesmann*) wherever applicable

Links against GNU scientific library for more elaborate integrations

Implements *Morrison et al* STDP

NEST

Scales great: 40x synapses with 2x postsynaptic firing rate increase – 32x simulation time with STDP, missing bit is what was the increase in incoming firing rate…

Future: interactive mode for distributed simulations and a set of validation and test tools

Cog Ex Machina

Core idea: digital hardware is not suitable for solving differential equations with high speed and precision, it is suitable for algebraic computations

Strategy: simplify neural models to a set of algebraic computations, solve them in massively parallel way

Upside: really fast for conventional neural networks, convolutions, FFTs, etc

Downsides:

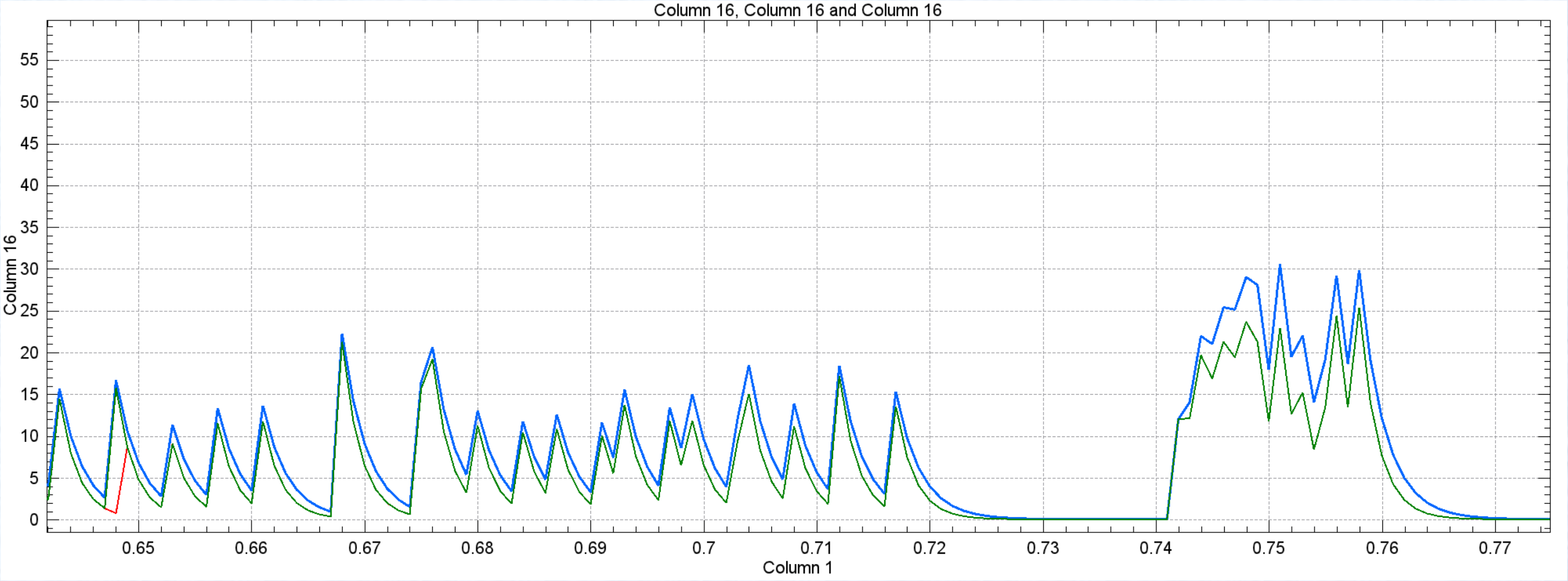
- – No real dynamics
- – No arbitrary or even random connectivity between neuronal populations
- – Proprietary

Cog Ex Machina: Adding Dynamics

Rotter-Diesmann approach would work perfectly, but it is limited to leaky IaF with current-based synapses

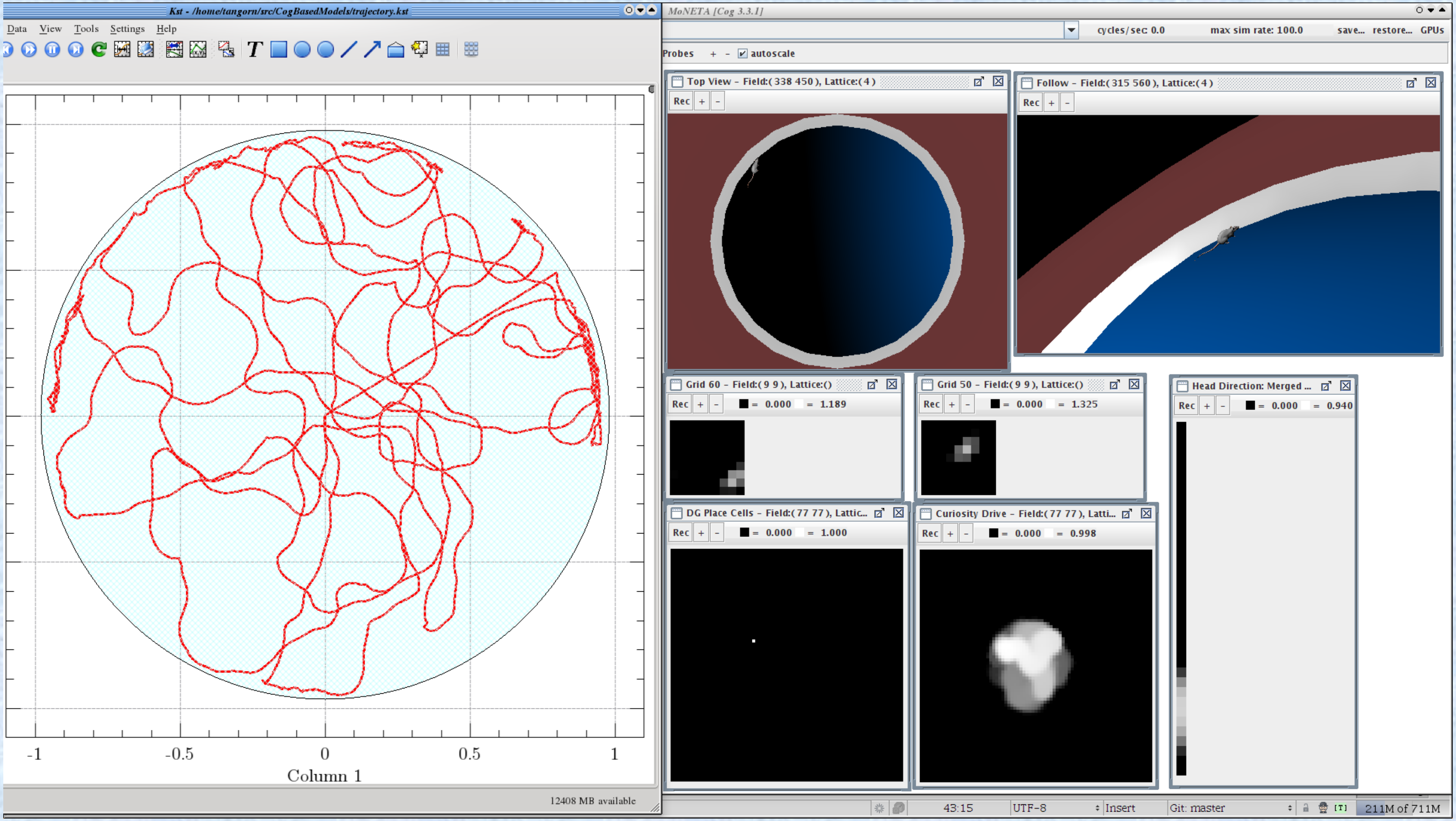
The only other way is Euler integration with a fixed step of 10ms – really bad precision

Here is a comparison of Euler (green) and RK4 (blue) with 1ms step for leaky integrator with delta synapses



Cog Ex Machina

Important upside: allows to integrate model with behavior


Kst - /home/tangorn/src/CogBasedModels/trajectory.kst
Data View Tools Settings Help
-1
-0.5
0
0.5
1
Column 1
12408 MB available
MoNETA [Cog 3.3.1]
cycles/sec 0.0
max sim rate: 100.0
save... restore... GPUs
Probes + - ☑ autoscale
Top View - Field:(338 450), Lattice:(4)
Rec + -
Follow - Field:(315 560), Lattice:(4)
Rec + -
Grid 60 - Field:(9 9), Lattice:()
Rec + - ■ = 0.000 □ = 1.189
Grid 50 - Field:(9 9), Lattice:()
Rec + - ■ = 0.000 □ = 1.325
Head Direction: Merged ...
Rec + - ■ = 0.000 □ = 0.940
DG Place Cells - Field:(77 77), Lattic...
Rec + - ■ = 0.000 □ = 1.000
Curiosity Drive - Field:(77 77), Latti...
Rec + - ■ = 0.000 □ = 0.998
43:15
UTF-8
Insert
Git: master
211M of 711M

Important Downside for All These Systems

They all require programming/scripting:

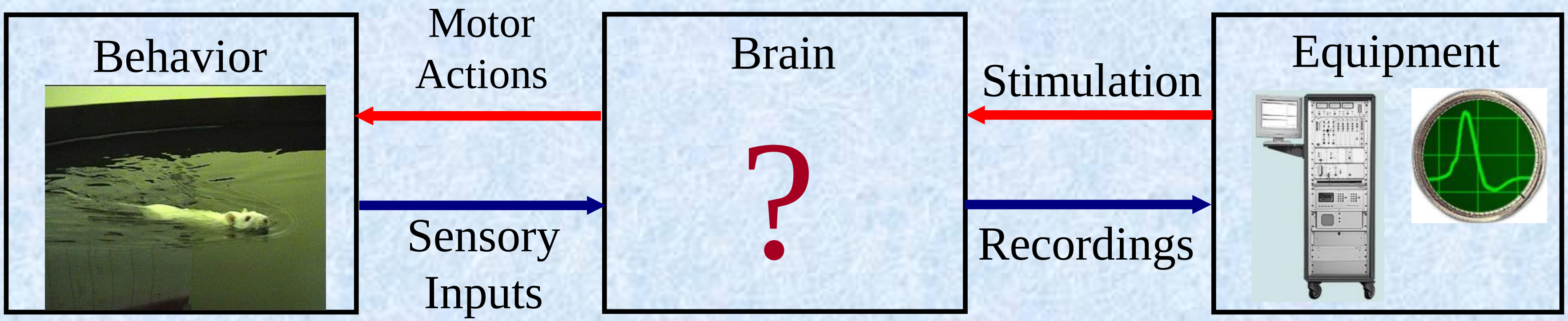
- XPP: scripts are directly interpreted
- Genesis 3, NEST: model is described in script formats, then interpreted by the system, C++ components are linked and executed
- NEURON, Cog Ex Machina: model is written in programming or scripting language, then (under the hood) compiled and executed

Although it is expected from computational neuroscientist to know programming, such design often shifts the focus from modeling to debugging your code

Alternative: leave debugging to pros – build model through GUI from extensive set of precompiled components

What Do We Model?

Experimental design in behavioral neuroscience:



Behavior

Motor Actions

Sensory Inputs

Brain

?

Stimulation

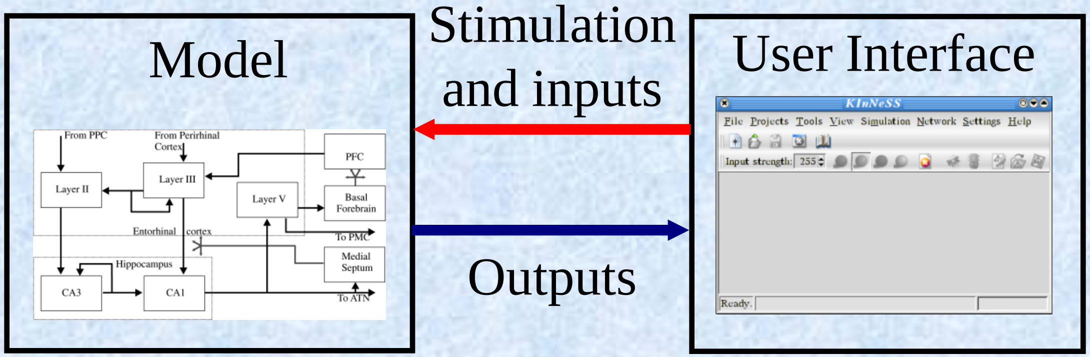
Recordings

Equipment

Usual modeling design in computational neuroscience:

often lump sensory inputs and stimulation

rarely model direct motor actions, so basically they are also lumped with recordings



Model

Stimulation and inputs

Outputs

User Interface

Input strength: 255

Ready.

What Is Missing

Behavior is out of the picture, we deduce it but we do not model it

This approach is fine in case of short stimulus-response simulations

Modeling of long behavioral sequences requires the mechanism to deduce next set of inputs given the current set of inputs and produced motor action

Example: visual stream during spatial navigation changes with position and head direction of the animal

Solution: create a virtual environment such that motor actions affect the environment and sensory inputs are produced by environment

Experimental design in behavioral neuroscience:

Behavior

Motor Actions

Sensory Inputs

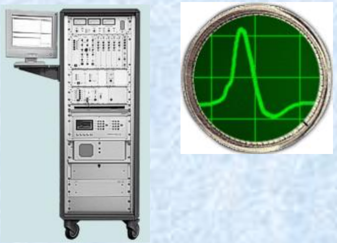
Brain

?

Stimulation

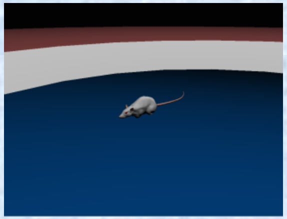
Recordings

Equipment



Suggested modeling design:

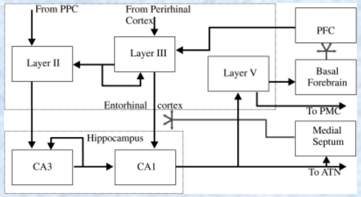
Virtual environment



“Motor Actions”

“Sensory Inputs”

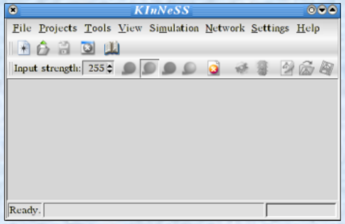
Model



Stimulation

“Recordings”

User Interface

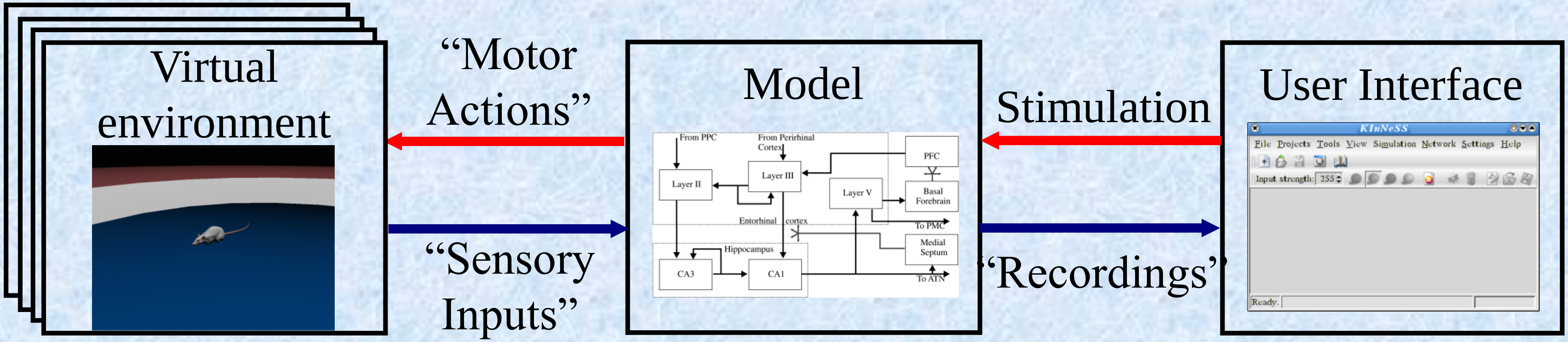


Input strength: 255

Ready.

Environment can be fully automatic and can visualize the behavior

Additional Considerations



Virtual environment

“Motor Actions”

“Sensory Inputs”

Model

Stimulation

“Recordings”

User Interface

Input strength: 255

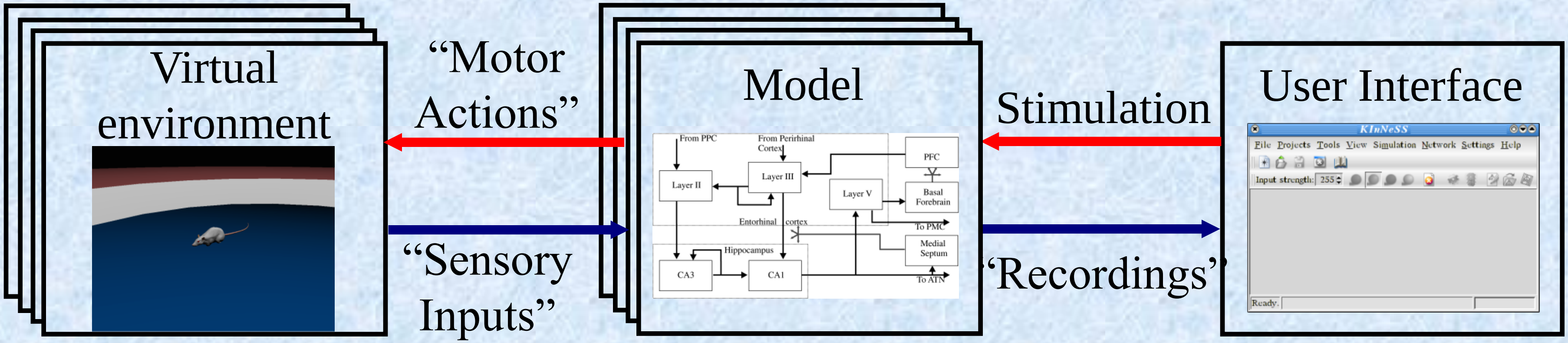
Ready.

There is a variety of different input/output pairs we would like to simulate including the conventional scheme with no behavior

It is inconvenient to have different software packages for simulations of different input/output combinations

Better solution is to allow different plugins within a single software framework and let the user choose the appropriate one

Additional Considerations



Virtual environment

"Motor Actions"

"Sensory Inputs"

Model

Stimulation

"Recordings"

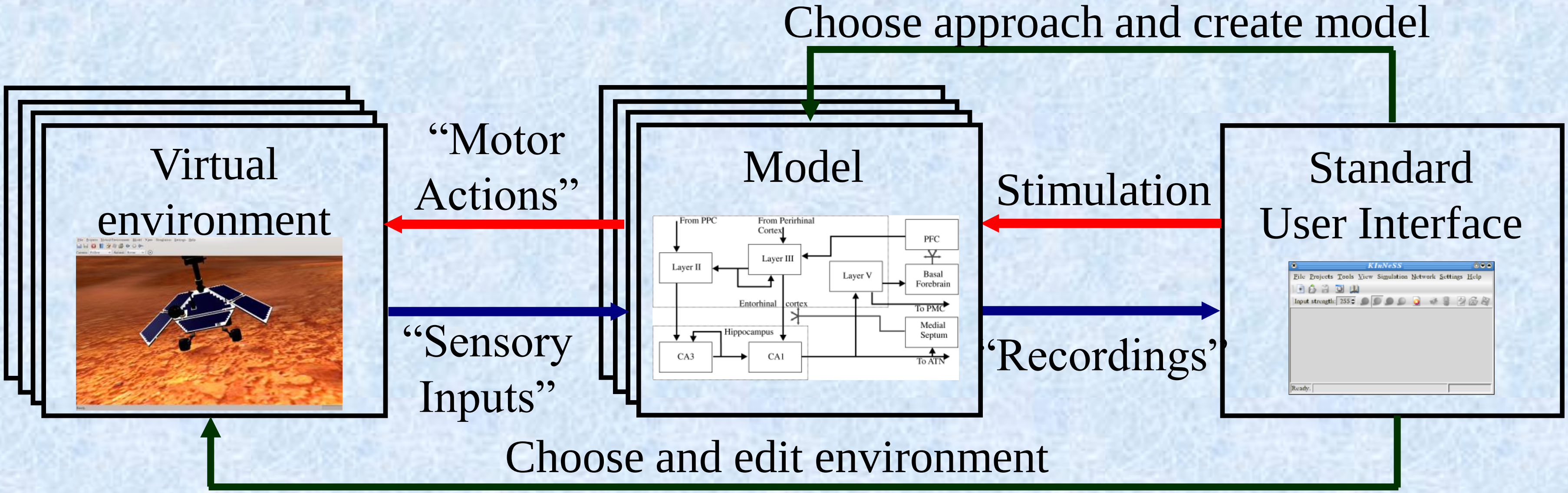
User Interface

Furthermore, there is a variety of modeling approaches

It would be great to give the user a chance to choose which approach to use in any given situation, or to try different approaches in any given study

Finally, the good software framework will allow the user to combine freely the modeling approach and the experimental environment

The Resulting Structure



Choose approach and create model

Virtual environment

“Motor Actions”

“Sensory Inputs”

Model

Stimulation

“Recordings”

Standard User Interface

Choose and edit environment

Standard user interface

- allows new users to learn the software package faster
- simplifies the creation and use of new environments and modeling frameworks

KDE Integrated NeuroSimulation Software (KInNeSS) is designed to implement this framework