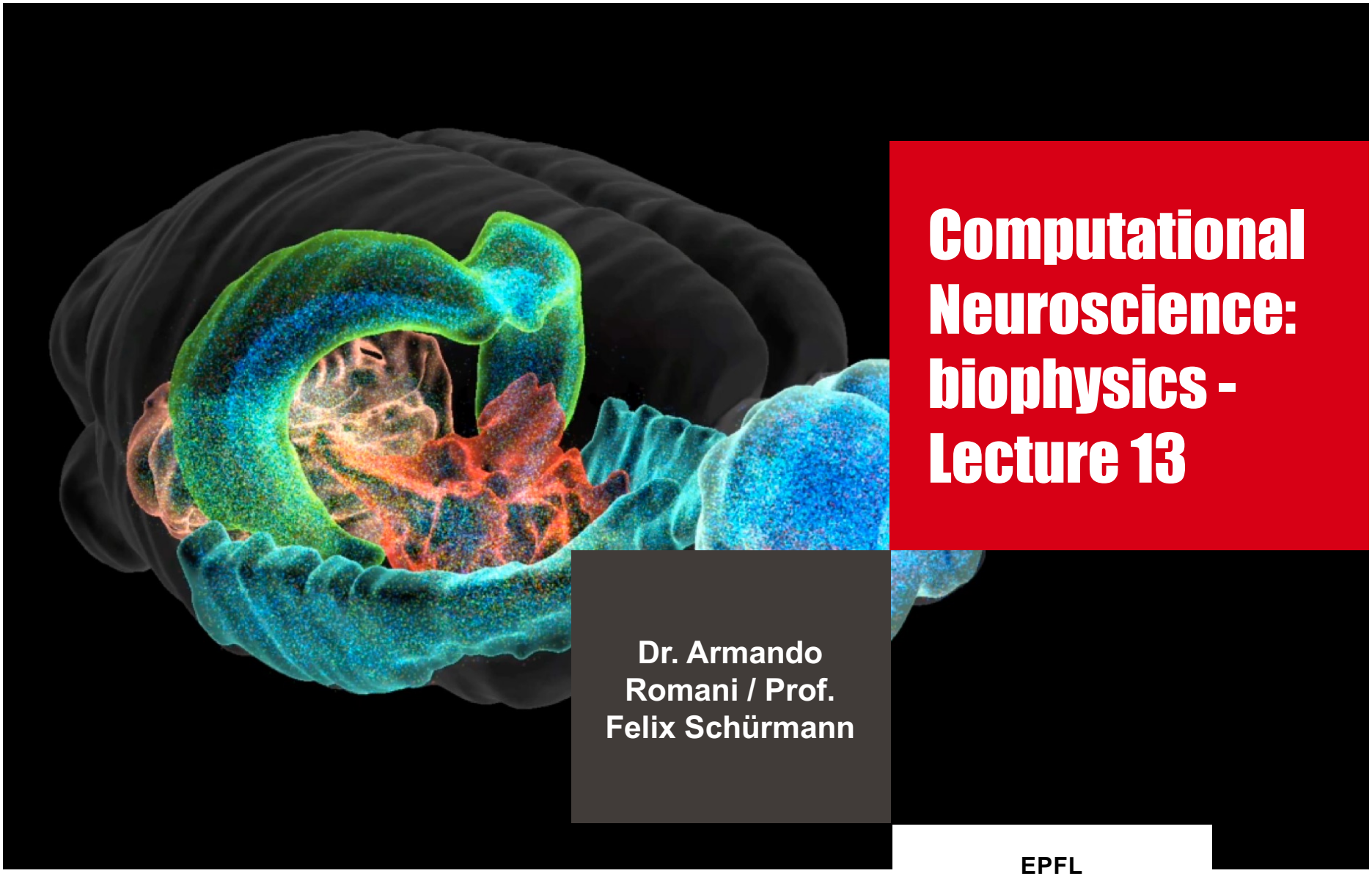
EPFL



Computational Neuroscience: biophysics - Lecture 13

Dr. Armando Romani / Prof. Felix Schürmann

Simulation & Scientific Computing

EPFL

Lecture Overview

- Scope
- Approaches
- Applications

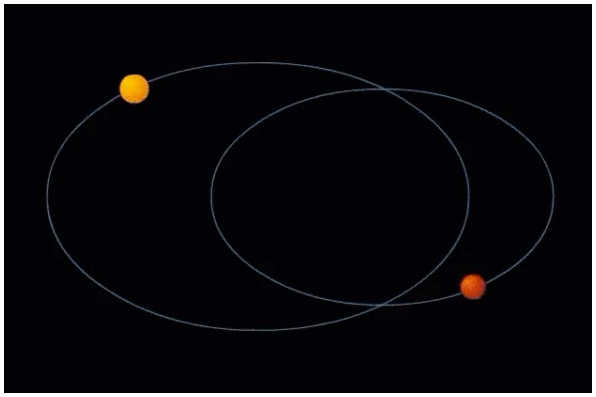
EPFL

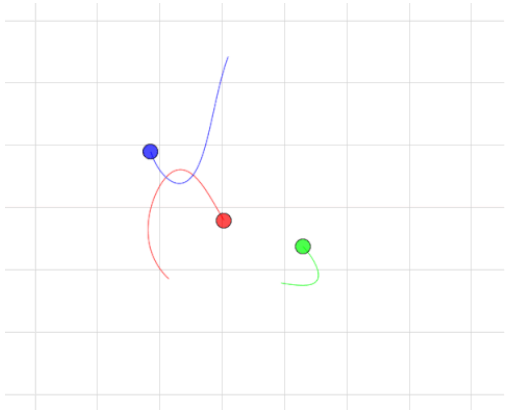
Lecture Overview

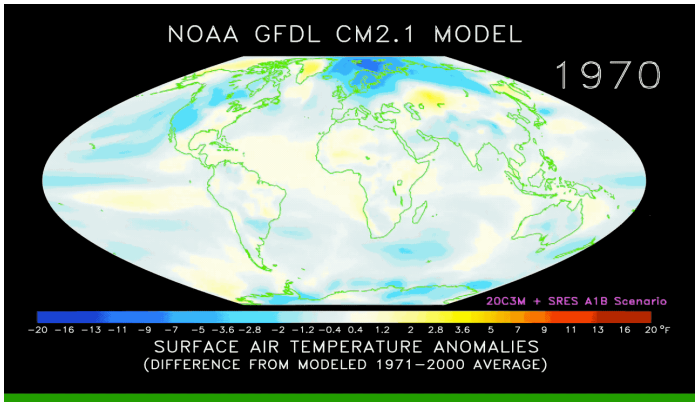
- **Scope**
- Approaches
- Applications

EPFL

Of Seemingly Simple and Complex Systems







- Theory is the starting point
- Many of those models can only be solved using numerical integration
- Some models are (much) more complex/larger than others

EPFL

Beginnings of Scientific Computing

NUMERICAL INVERTING OF MATRICES OF HIGH ORDER

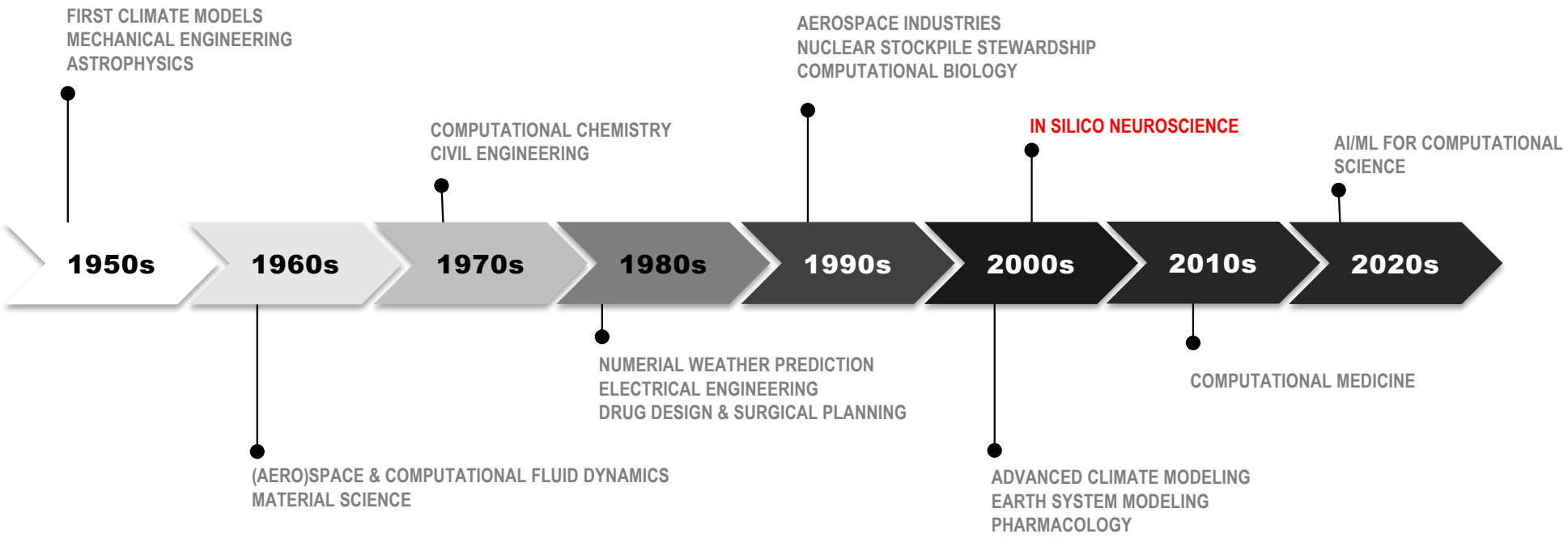
JOHN VON NEUMANN AND H. H. GOLDSTINE

(Bulletin of the AMS, Nov. 1947)

- First testimony of the synergy of
  - programmable electronic computer
  - mathematical analysis
  - opportunity and need to solve large and complex problems in applications

EPFL

Brief History of Computational Science



EPFL

Computing – the most successful scaling story in human history

ENIAC

Glen Beck (background) (foreground) program… building 328. (U.S. Army p…

~18,000 va…
~27tons
~167…
150kW…
Per secon…
5000 additions or
357 multiplications or
38 divisions

"ENIAC": ~10^3 FLOPS

75 years
10^{15}x increase

Main drivers
Transistor 1948
Integrated Circuits 1958
CMOS 1963

OLCF – "Frontier"

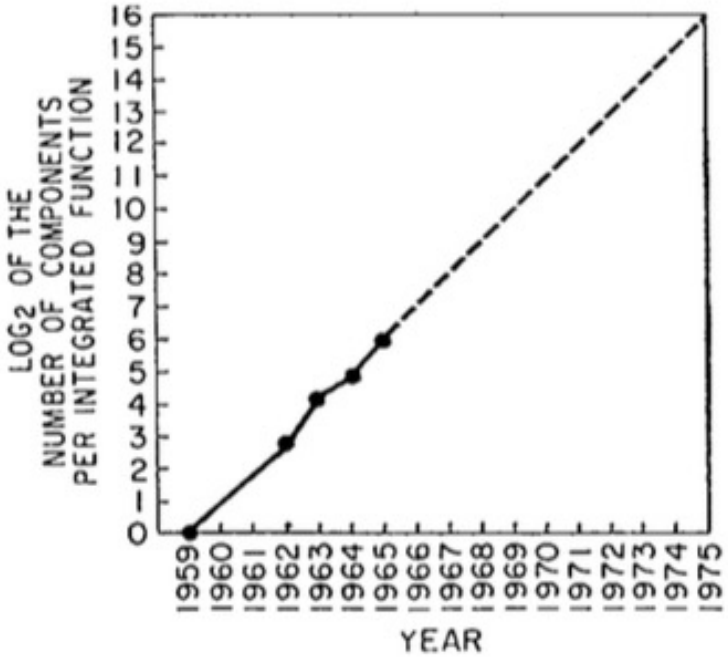
Operators	Oak… Laboratory and …of Energy
Locatio…	…eadership Computing
…	…W
Op… syste…	…HPE Cray OS
Space	680 m^2 (7,300 sq ft)
Speed	1.102 exaFLOPS (Rmax) / 1.685 exaFLOPS (Rpeak)[1]
Cost	US$600M (estimated cost)
Purpose	Scientific research
Website	https://www.olcf.ornl.gov/frontier/

"Frontier": ~10^{18} FLOPS

EPFL

Moore's Law

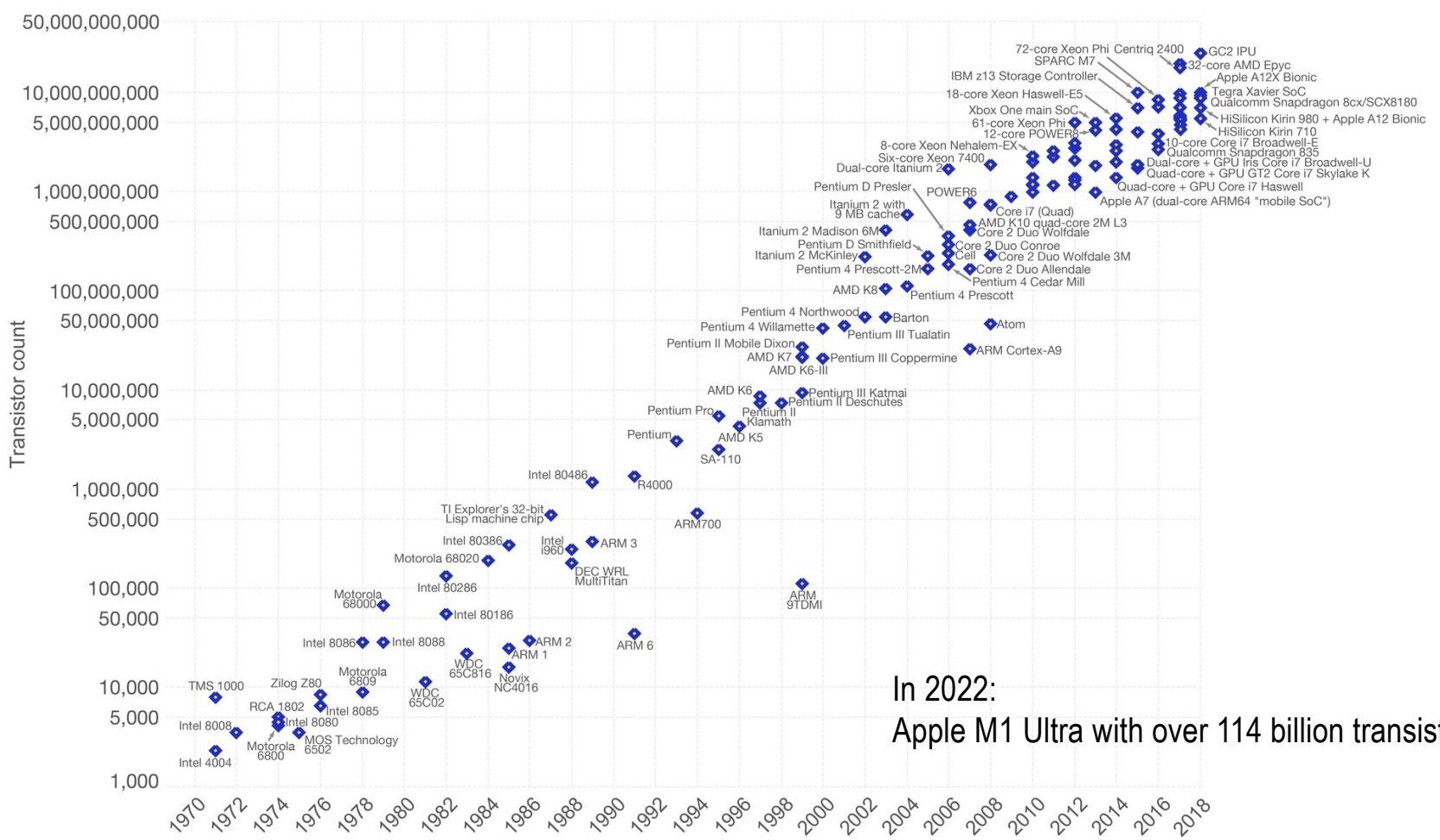
- Describes the observation that component counts double every year in integrated circuits
- Revised in 1975 to double every 2 years



Gordon E. Moore, 1965

Gordon E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics,* pp. 114–117, April 19, 1965.

Moore's Law 1971-2018



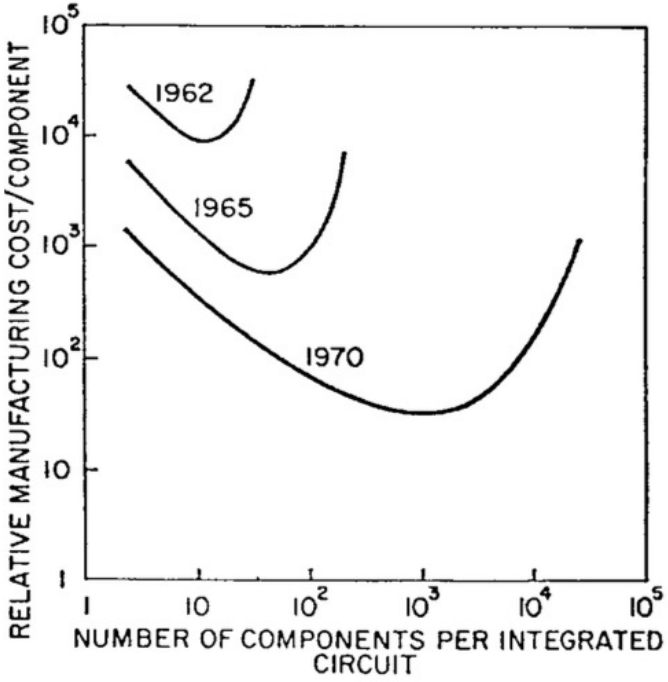
In 2022:
Apple M1 Ultra with over 114 billion transistors

Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

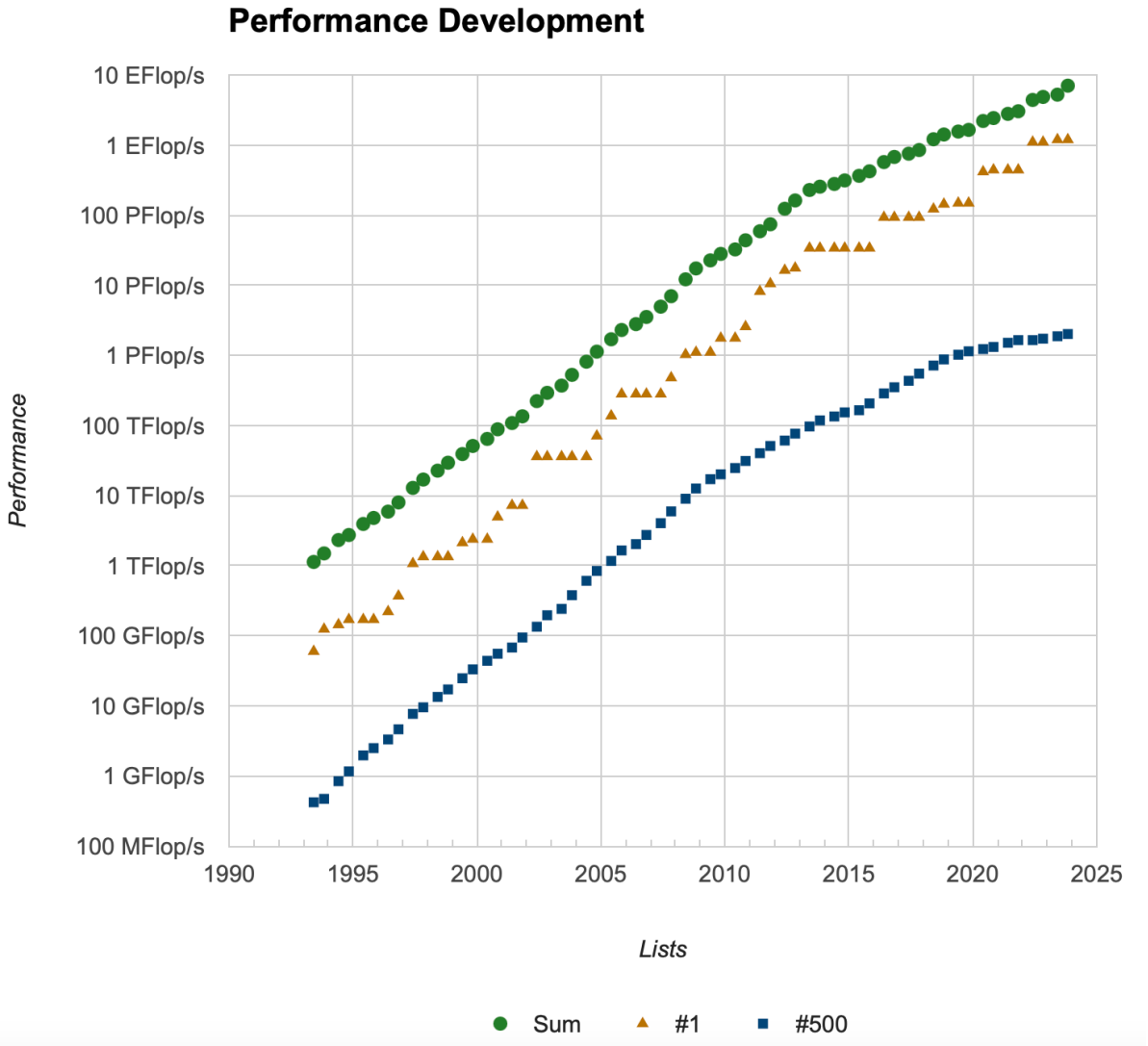
Economics!

- At any time a manufacturing technology allows
  - Integration of few to many components in an IC
  - for simple circuits the cost per component is nearly ~1/#components
  - At some point the decreased yield makes it economically uninteresting

- Yield:
  - Yield is the percentage of chips on a finished wafer that pass all tests and function properly.
  - Production of integrated circuits leads to non-functional variations and faults (and surely: more surface more faults!)



Top500 Supercomputers over time



From: top500.org

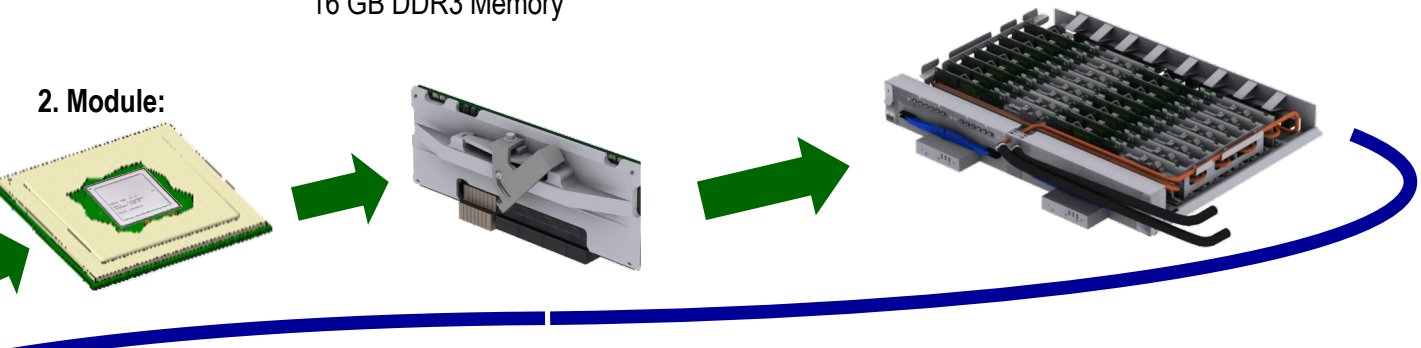
Buil-Up of a Massively Parallel Supercomputer

1. Chip:
16 cores

2. Module:

3. Compute Card ("Node"):
One BQC Module (1x1x1x1x1),
16 GB DDR3 Memory

4. Node Card ("Node Board"):
32 Compute Cards (2x2x2x2x2),
Optical Modules, BQL Link Chips, Torus

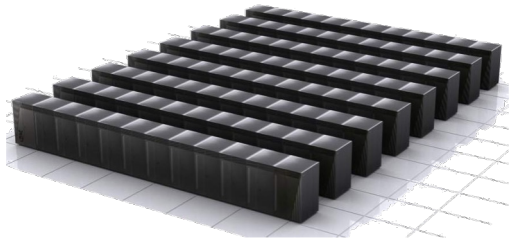


5a. Midplane:
16 Node Cards (4x4x4x4x2)

5b. I/O drawer (1,2 or 4 per rack):
* 8 I/O cards @ 16 GB,
* 8 PCIe gen2 x8 slots (IB, 10GbE)

6. Rack: 2 Midplanes (4x4x4x8x2)

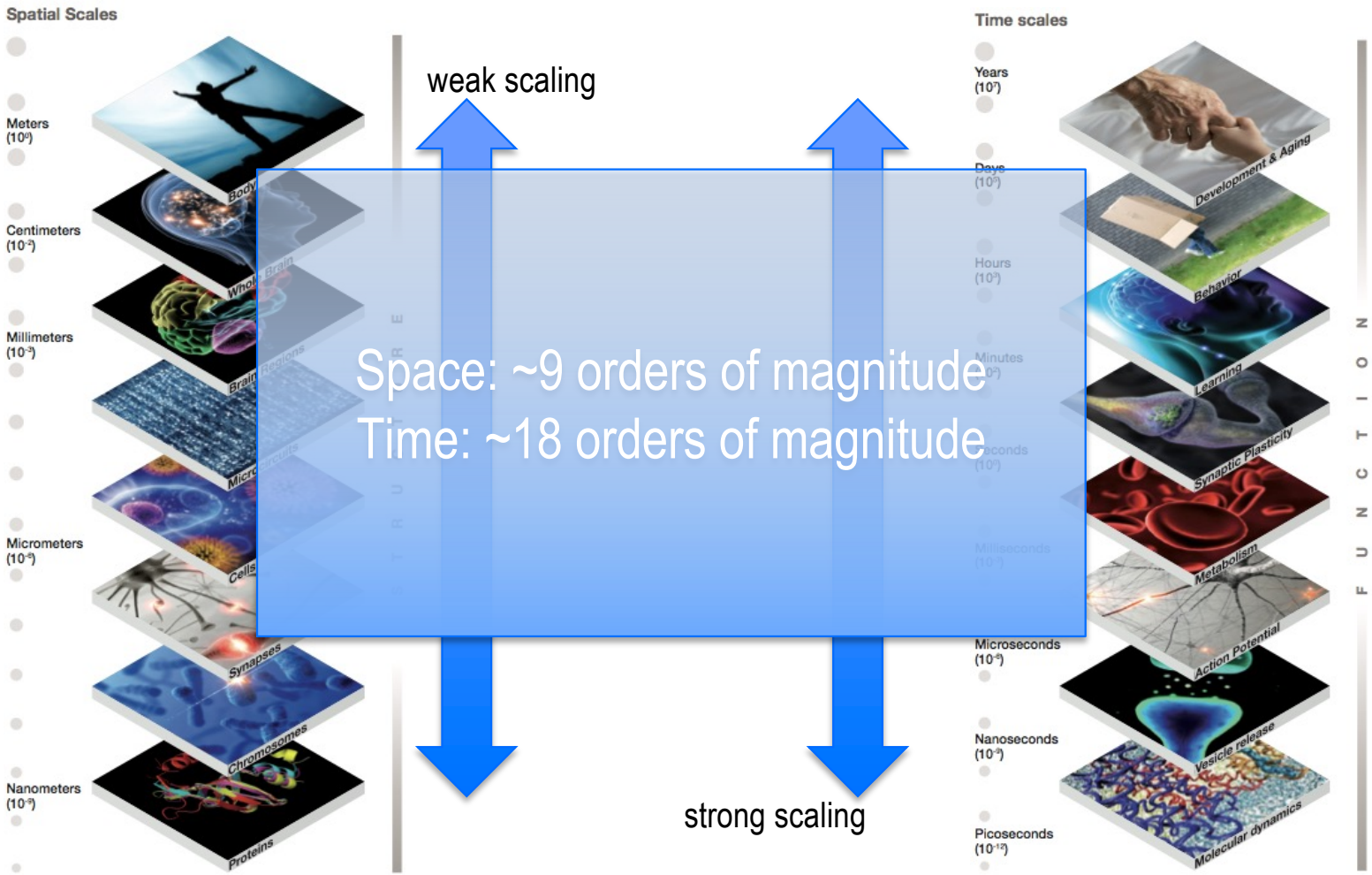
7. System:
e.g. 8 racks (8x8x8x8x2) = 1.7 PF/s
e.g. 96 racks (16x12x16x16x2) = 20 PF/s



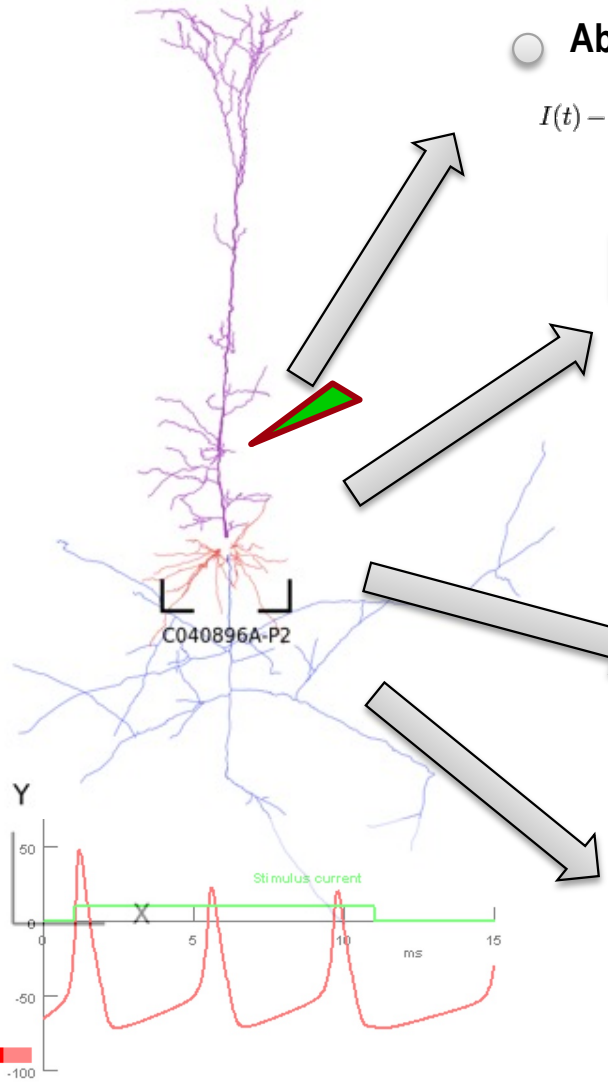
•Sustained single node perf: 10x BG/P, 20x BG/L
• MF/Watt: 6x BG/P, 10x BG/L (~2GF/Watt)

Source: IBM

The Brain - Relevant Scales



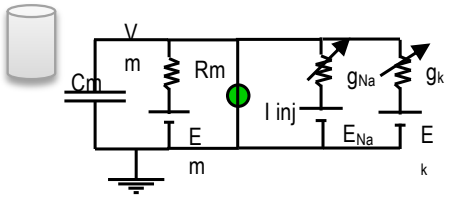
Many Ways to Modeling a Neuron



Abstract Model: Point neuron, e.g. Leaky Integrate and Fire

$$I(t) - \frac{V_{\mathrm{m}}(t)}{R_{\mathrm{m}}} = C_{\mathrm{m}}\frac{dV_{\mathrm{m}}(t)}{dt} \qquad f(I) = \begin{cases} 0, & I \le I_{\mathrm{th}} \\ [t_{\mathrm{ref}} - R_{\mathrm{m}}C_{\mathrm{m}}\log(1-\frac{V_{\mathrm{th}}}{IR_{\mathrm{m}}})]^{-1}, & I > I_{\mathrm{th}} \end{cases}$$

Simplified Model: Single Compartment and ion channel formalism



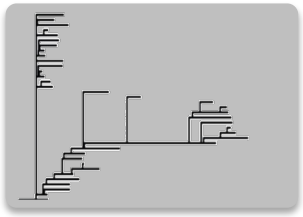
$$\frac{C_m dV_m}{dt} = \frac{E_m - V_m}{R_m} + I_{channels}$$

$$\frac{d\mathrm{m}}{dt} = \alpha_{\mathrm{m}}(V_m)(1-\mathrm{m}) - \beta_{\mathrm{m}}(V_m)\mathrm{m}$$

$$\frac{d\mathrm{h}}{dt} = \alpha_{\mathrm{h}}(V_m)(1-\mathrm{h}) - \beta_{\mathrm{h}}(V_m)\mathrm{h}$$

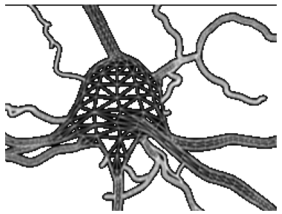
$$I_{channel} = \mathrm{m}^n \mathrm{h} g_{channel}(V_m - E_{channel})$$

Cellular Model: Cable and ion channel formalism



$$\frac{C_m dV_m}{dt} = \frac{E_m - V_m}{R_m} + I_{channels} + \frac{2(V_{m_{i+1}} - V_{m_i})}{R_{a_{i+1}} + R_a} + \frac{2(V_{m_{i-1}} - V_{m_i})}{R_{a_{i-1}} + R_a}$$

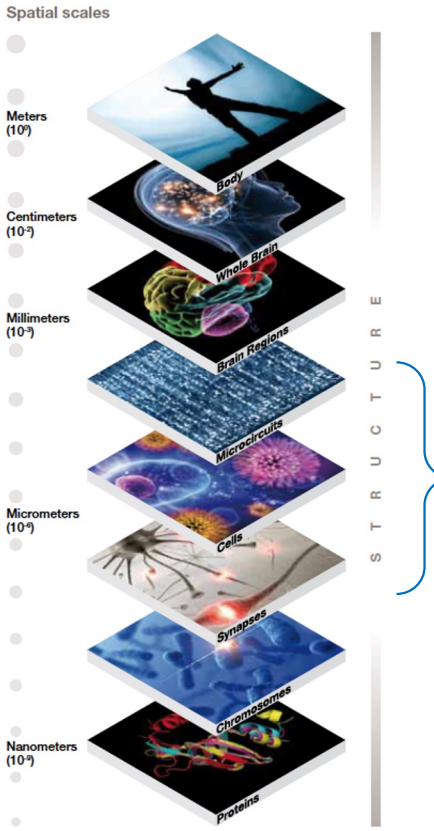
Subcellular Model: Reaction-Diffusion formalism



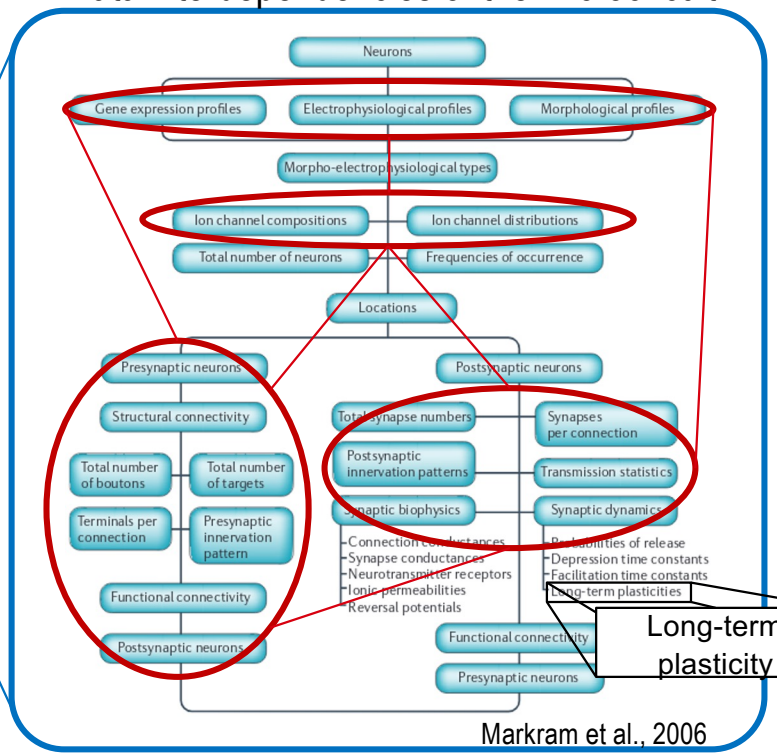
$$\dot{p}(\mathbf{x};t) = -p(\mathbf{x};t)\sum_{\mu=1}^{M} a_\mu(\mathbf{x}) + \sum_{\mu=1}^{M} p(\mathbf{x}-s_\mu;t)a_\mu(\mathbf{x}-s_\mu)$$

How to understand neural circuits across levels?

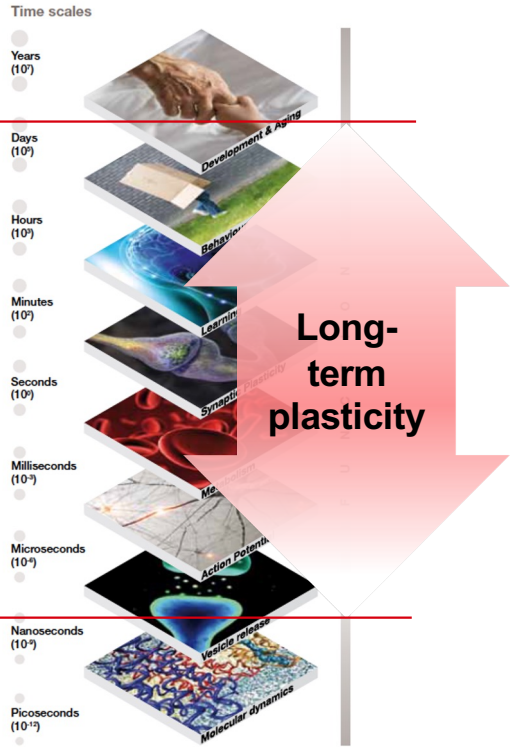
Example: Microcircuit plasticity, a synaptic correlate of learning



Data interdependencies of the microcircuit



Markram et al., 2006



Challenges

- Computing at the “frontier of what is computable” by definition is a changing problem, requiring adaptation to new technology and its limitations
- Navigating this potential requires knowledge from the fields of computational neuroscience, applied mathematics and computer science and engineering
- Leveraging the computational power available in modern supercomputers requires appropriate computational methods, tools and practices

EPFL

Summary 1

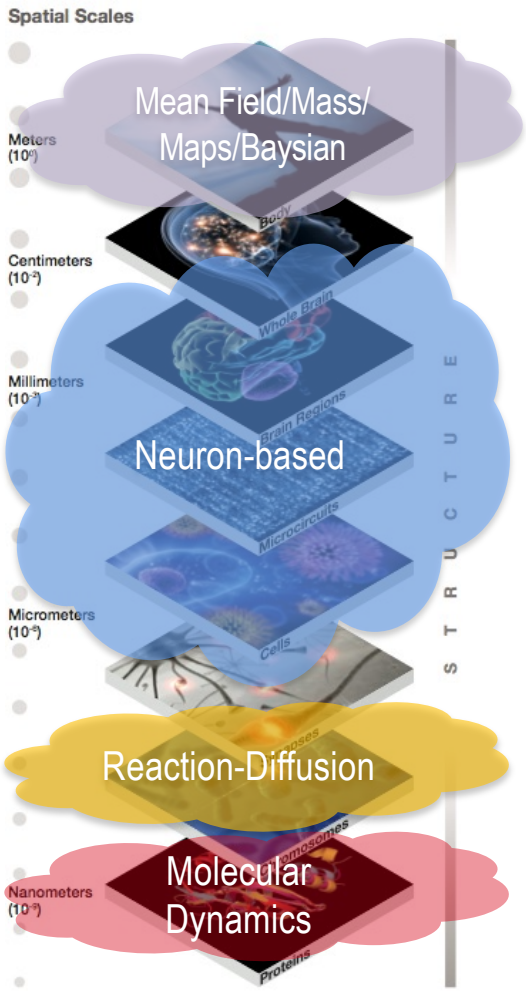
- Scientific computing describes the opportunity and need to solve large and complex problems using programmable electronic computers and mathematical analysis
- The performance increase in electronic computers since the beginnings in the 1940s is possibly the largest continual improvement in any man-made technology
- Biophysical and biochemical computational modeling is a useful approach to study the multi-scale nature of the brain and its phenomena requiring appropriate numerical methods and large-scale computers

Lecture Overview

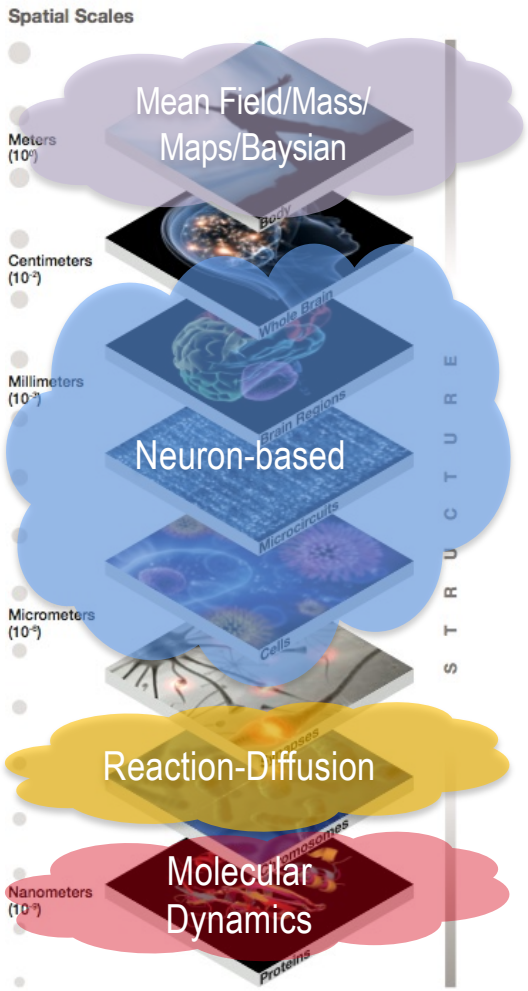
- Scope
- **Approaches**
- Resources/Applications

EPFL

Many Ways to Modeling (pieces of) a Brain



Many Neuro Simulator Resources



Topographica, MIIND, …

NEST, BRIAN, CSIM, MVASpike, C2, …

NEURON/CoreNeuron, MOOSE, GENESIS, Arbor, …

STEPS, MCELL, VCELL, NEURORD, CDS, …

LAMMPS, NAMD, AMBER, GROMACS, ESPRESSO, …
Gaussian, CPMD, CP2K, …

EPFL

A good starting point

J Comput Neurosci (2007) 23:349–398
DOI 10.1007/s10827-007-0038-6

TOPICAL REVIEW ON TECHNIQUES

Simulation of networks of spiking neurons: A review of tools and strategies

Romain Brette · Michelle Rudolph · Ted Carnevale · Michael Hines · David Beeman · James M. Bower · Markus Diesmann · Abigail Morrison · Philip H. Goodman · Frederick C. Harris, Jr. · Milind Zirpe · Thomas Natschläger · Dejan Pecevski · Bard Ermentrout · Mikael Djurfeldt · Anders Lansner · Olivier Rochel · Thierry Vieville · Eilif Muller · Andrew P. Davison · Sami El Boustani · Alain Destexhe

Received: 29 November 2006 / Revised: 2 April 2007 / Accepted: 12 April 2007 / Published online: 12 July 2007

Abstract We review different aspects of the simulation of spiking neural networks. We start by reviewing the different types of simulation strategies and algorithms that are currently implemented. We next review the precision of those simulation strategies, in particular in cases where plasticity depends on the exact timing of the spikes. We overview different simulators and simulation environments presently available (restricted to those freely available, open source and documented). For each simulation tool, its advantages and pitfalls are reviewed, with an aim to allow the reader to identify which simulator is appropriate for a given task. Finally,

EPFL

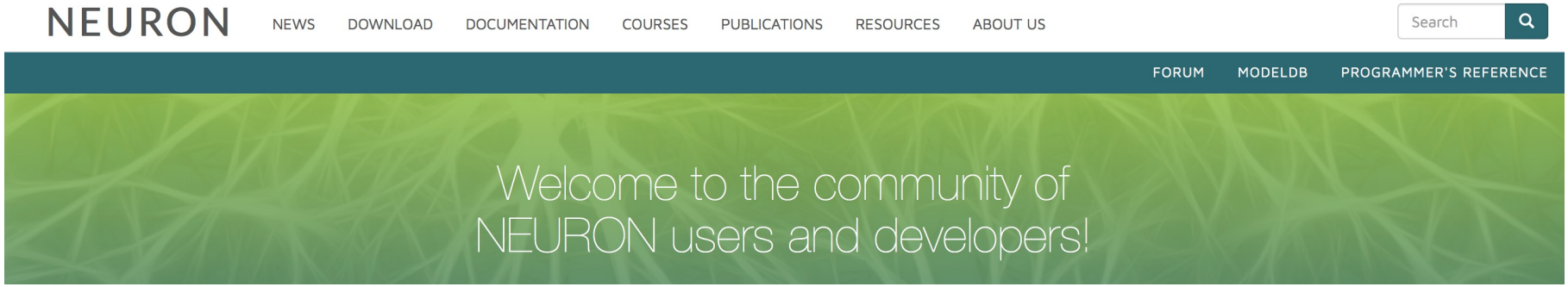
Comparison

Table 1 Comparison of features of the different simulators

Question	NEURON	GENESIS	NEST	NCS	CSIM	XPP	SPLIT	Mvaspike
HH	B.I.	B.I.	YES	B.I.	B.I.	YES	B.I.	POSS
LIF	B.I.	POSS	YES	B.I.	B.I.	YES	POSS**	B.I.
Izhikevich IF	YES	B.I.	YES	NO	B.I.	YES	POSS**	POSS**
Cable eqs	B.I.	B.I.	NO	NO	NO	YES	B.I.	NO
ST plasticity	YES	B.I.	YES	B.I.	B.I.	YES	B.I.	YES
LT Plasticity	YES	YES	YES	B.I.	B.I.	YES	NO**	YES
Event-based	B.I.	NO	YES	NO	NO	YES	NO	YES
Exact	B.I.	–	YES	–	–	NO	–	YES
Clock-based	B.I.	B.I.	YES	B.I.	YES	YES	YES	POSS**
Interpolated	B.I.	NO	YES	NO	NO	YES	B.I.	POSS
G synapses	B.I.	B.I.	YES	B.I.	B.I.	YES	B.I.	POSS**
Parallel	B.I.	YES	B.I.	B.I.	NO**	NO	B.I.	NO**
Graphics	B.I.	B.I.	NO(*)	NO(*)	NO(*)	YES	NO	NO
Simple analysis	B.I.	B.I.	YES	NO(*)	NO(*)	YES	NO	NO
Complx analysis	B.I.	YES	NO(*)	NO(*)	NO(*)	YES	NO	NO
Development	YES	YES	YES	YES	YES	YES	YES	YES
How many p.	3	2–3	4	2–3	2	1	2	1
Support	YES	YES	YES	YES	YES	YES	YES	YES
Type	e,p,c	e	e	e	e	e	e	e
User forum	YES	YES	YES	NO	NO	YES	YES	NO
Publ list	YES	YES	YES	YES	YES	NO	NO	NO
Codes	YES	YES	YES	YES	YES	YES	NO	NO
Online manual	YES	YES	YES	YES	YES	YES	YES	YES
Book	YES	YES	NO	NO	NO	YES	NO	NO
XML import	NO**	POSS	NO**	NO**	NO	YES	NO	NO**
XML export	B.I.	NO**	NO**	NO**	NO	NO	NO	NO**
Web site	YES	YES	YES	YES	YES	YES	YES	YES
LINUX	YES	YES	YES	YES	YES	YES	YES	YES
Windows	YES	YES	YES	YES	YES	YES	NO	NO
Mac-Os	YES	YES	YES	NO	NO	YES	NO	NO
Interface	B.I.	B.I.	POSS	B.I	YES	POSS	POSS	POSS
Save option	B.I.	YES	NO**	B.I.	NO	NO	NO	NO

EPFL

The NEURON Simulation Environment



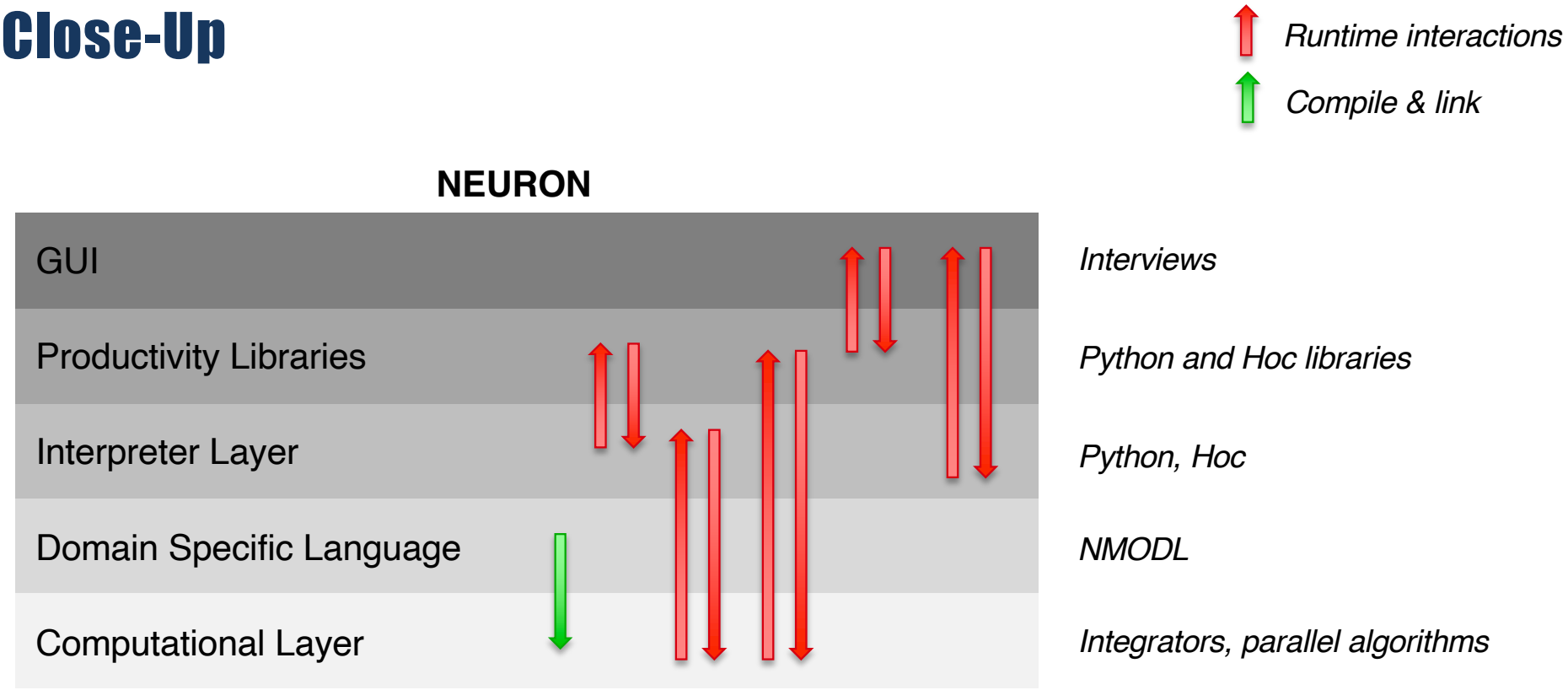
Core authors: **Michael Hines**, Robert McDougal, Ted Carnevale @ Yale University

Open source, http://www.neuron.yale.edu

- Efficient, domain-specific simulator for multi-compartment, conductance-based simulations of neurons and networks
- De-facto standard for cellular-level models: more than 2300 scientific publications using NEURON

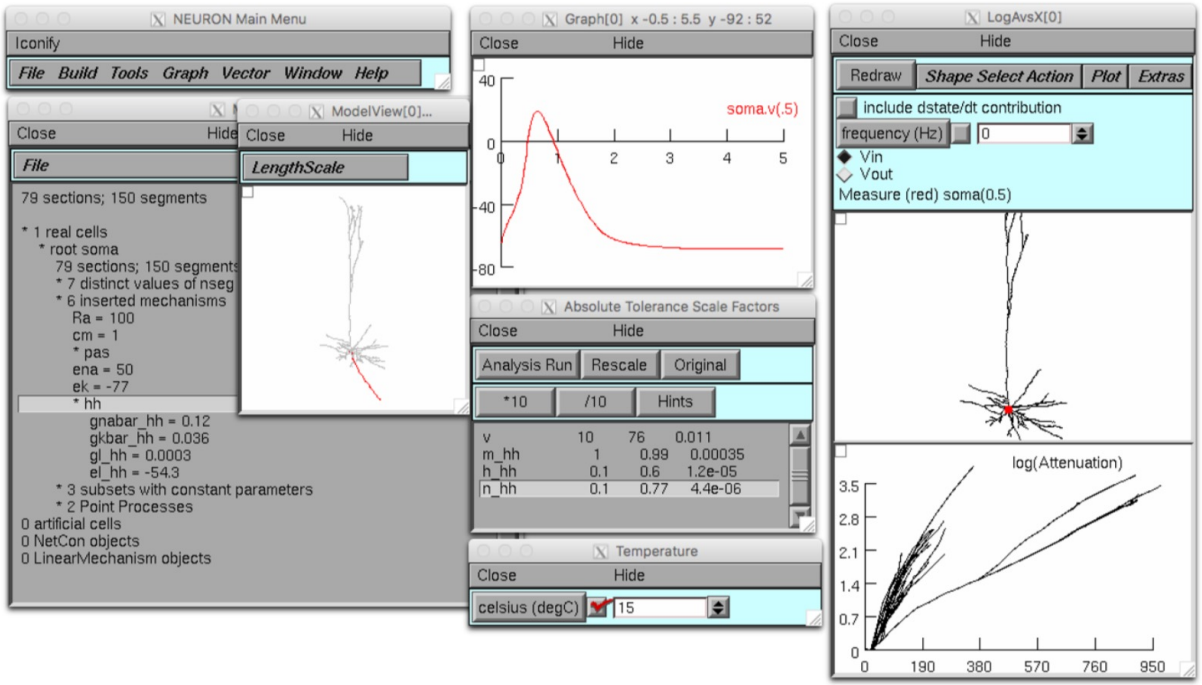
EPFL

NEURON Close-Up



Graphical User Interface

- When installed locally, NEURON comes with a powerful graphical user interface
- This provides access to model building tools (Import3D, CellBuilder, ChannelBuilder, NetworkBuilder), parameters, visualization of neurons and simulation results without the need to program
- The UI is customizable for your own purposes, allowing to build tailor-made “applications”




```
$ python
>>> from neuron import h, gui
```

Productivity Libraries

- NEURON provides specialized functionality for complex workflows, e.g.
  - Morphology import (Import3D) allowing data input and model output, analysis and repair
  - Pseudo random number generators (such as Random123)
  - Reaction-Diffusion semantics (rxd) allowing to declare domains, molecules, diffusion, reactions, 1D, 3D
  - SaveState/BBSaveState allowing to store the state of a neuron model for e.g. simulating different trajectories
  - ParallelContext allowing simple bulletin board style parallelization
  - Etc.

EPFL

Interpreter Layer

- NEURON provides an interpreter layer, ie. a way to script NEURON without the need of compiling the code
- This is the most powerful way of interacting with NEURON and its functionality
- Traditionally, NEURON used the hoc (higher order calculator) language as its primary scripting language
- Since 2009, NEURON supports python, which is now the standard (however, you will still find a lot of hoc code around)

frontiers in
NEUROINFORMATICS

ORIGINAL RESEARCH ARTICLE
published: 28 January 2009
doi: 10.3389/neuro.11.001.2009

NEURON and Python

Michael L. Hines[1], Andrew P. Davison[2]* and Eilif Muller[3]

[1] *Computer Science, Yale University, New Haven, CT, USA*
[2] *Unité de Neurosciences Intégratives et Computationelles, CNRS, Gif sur Yvette, France*
[3] *Laboratory for Computational Neuroscience, Ecole Polytechnique Fédérale de Lausanne, Switzerland*

Edited by:
Rolf Kötter, Radboud University, Nijmegen, The Netherlands

Reviewed by:
Felix Schürmann, Ecole Polytechnique Fédérale de Lausanne, Switzerland
Volker Steuber, University of Hertfordshire, UK
Arnd Roth, University College London, UK

****Correspondence:***
Andrew Davison, UNIC, Bât. 32/33, CNRS, 1 Avenue de la Terrasse, 91198 Gif sur Yvette, France.
e-mail: andrew.davison@unic.cnrs-gif.fr

The NEURON simulation program now allows Python to be used, alone or in combination with NEURON's traditional Hoc interpreter. Adding Python to NEURON has the immediate benefit of making available a very extensive suite of analysis tools written for engineering and science. It also catalyzes NEURON software development by offering users a modern programming tool that is recognized for its flexibility and power to create and maintain complex programs. At the same time, nothing is lost because all existing models written in Hoc, including graphical user interface tools, continue to work without change and are also available within the Python context. An example of the benefits of Python availability is the use of the `xml` module in implementing NEURON's Import3D and CellBuild tools to read MorphML and NeuroML model specifications.

Keywords: Python, simulation environment, computational neuroscience

```
# ---------------------- Model specification ----------------------

# topology
soma = Section()                    #       create soma, apical, basilar, axon
apical = Section()
basilar = Section()
axon = Section()

apical.connect(soma, 1, 0)          #       connect apical(0), soma(1)
basilar.connect(soma, 0, 0)         #       connect basilar(0), soma(0)
axon.connect(soma, 0, 0)            #       connect axon(0), soma(0)

# geometry
                                    #       soma {
soma.L = 30                         #           L = 30
soma.nseg = 1                       #           nseg = 1
soma.diam = 30                      #           diam = 30
                                    #       }
                                    #       apical {
apical.L = 600                      #           L = 600
apical.nseg = 23                    #           nseg = 23
apical.diam = 1                     #           diam = 1
                                    #       }
```

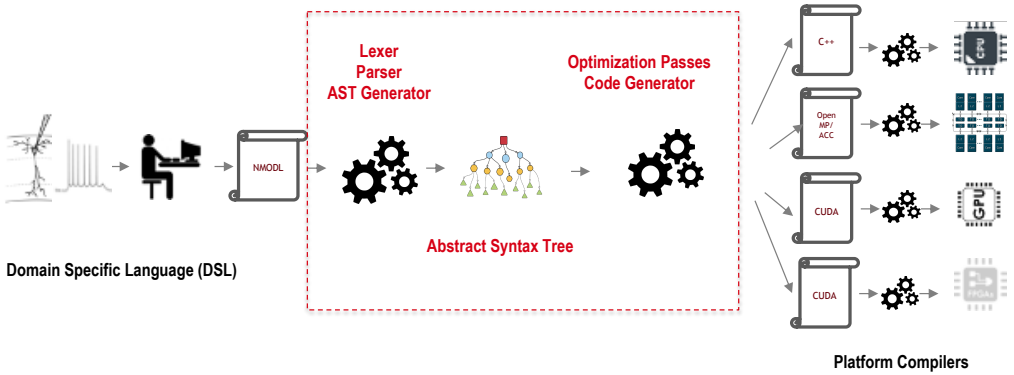
EPFL

Domain Specific Language - NMODL

- NEURON provides a Domain Specific Language (DSL) called NMODL that allows to describe models in terms of equations and kinetic schemes
- Behind the scenes, NEURON translates the NMODL constructs into C which then get compiled and linked to the executable
- This provides the most efficient way to implement models as it avoids the interpreter, used for channels/synapses etc.
- DSLs have recently become a major tool in computational science to bridge domain science with high performance execution

```
: A passive leak current

NEURON {
      SUFFIX leak
      NONSPECIFIC_CURRENT i
      RANGE i, e, g
}

PARAMETER {
      g = 0.001   (siemens/cm2)  < 0, 1e9 >
      e = -65     (millivolt)
}

ASSIGNED {
      i  (milliamp/cm2)
      v  (millivolt)
}

BREAKPOINT { i = g*(v - e) }
```

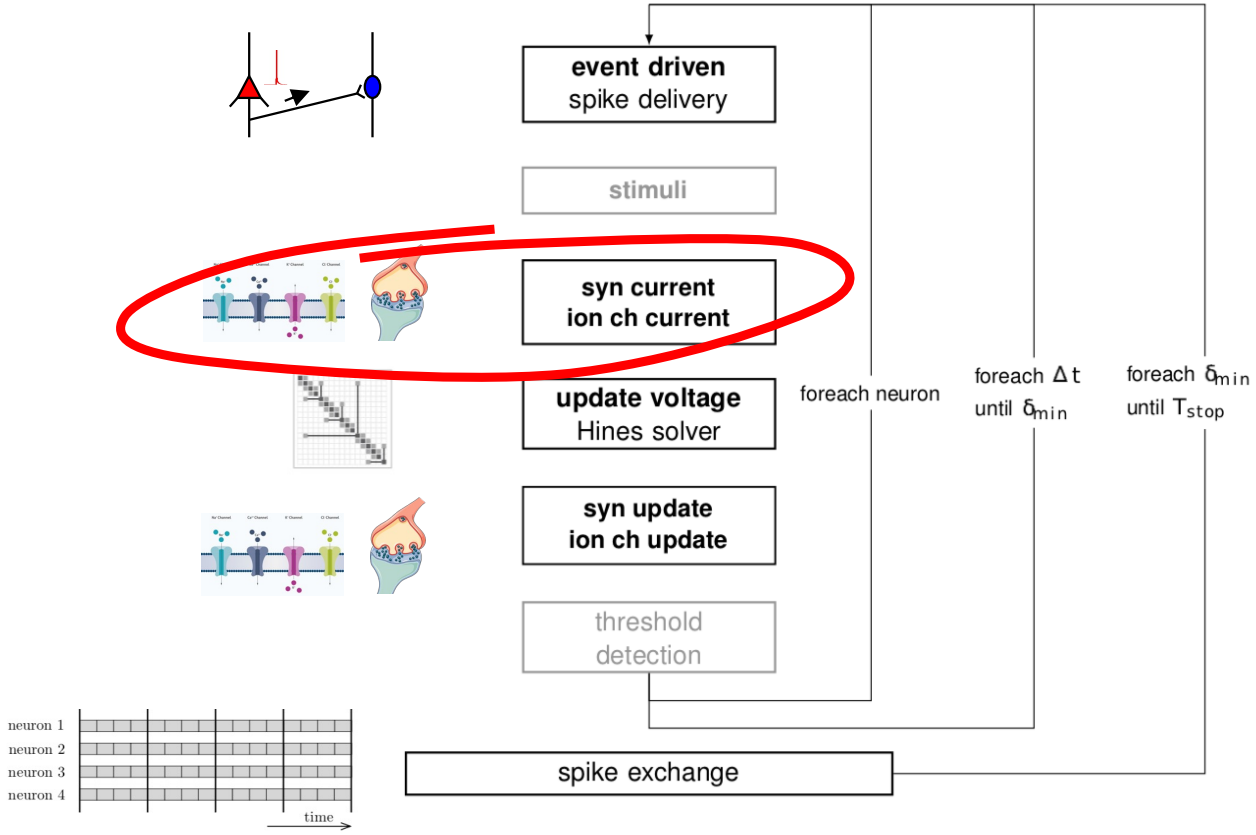
Hines and Carnevale; Expanding NEURON with NMODL, 2000



Kumbhar et al.; An optimizing multi-platform source-to-source compiler framework for the NEURON MODeling Language, ICCS, 2020

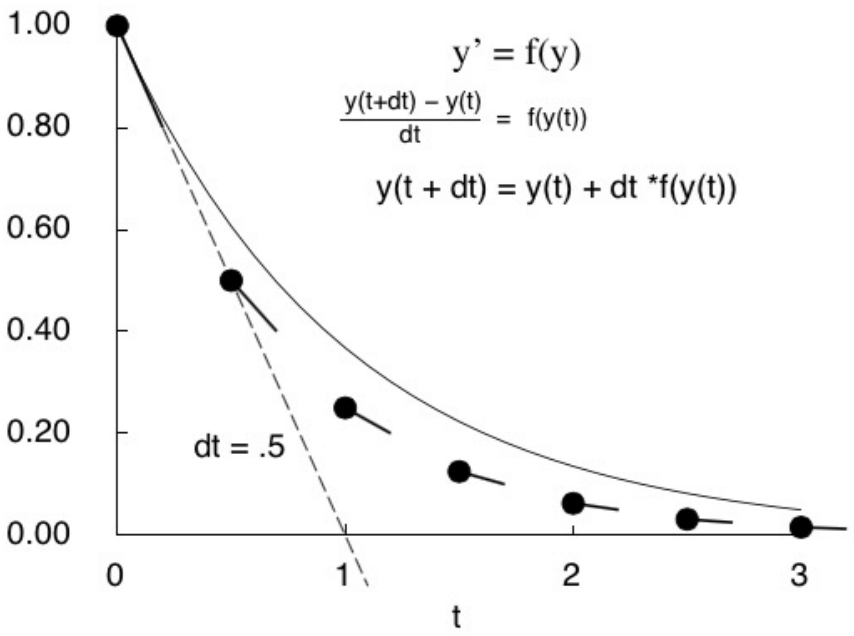
EPFL

Simulation Loop for Biophysically Detailed Neurons/Networks



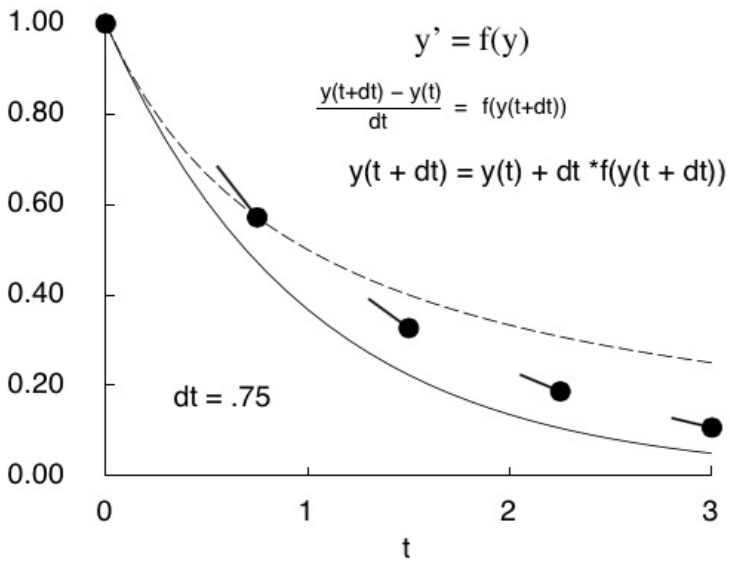
Forward Euler

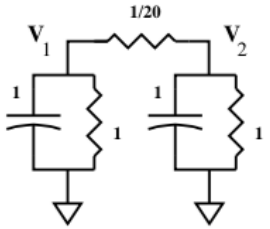
- Most basic explicit method for solving ODEs with initial value
- (simplest Runge-Kutta method)
- First order method → global error ~ dt
- Unstable for stiff equations,
  e.g. something as simple as y'=-2.3y
  (unstable for step size =1; stable for step size 0.7)
- NOT used in NEURON

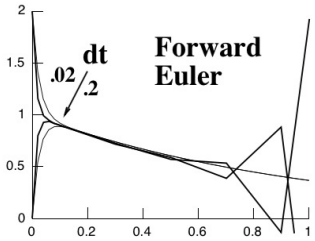


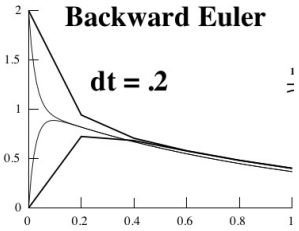
Backward Euler

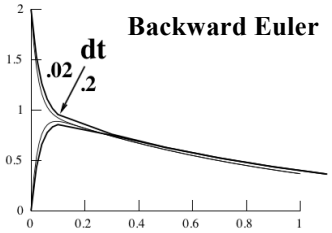
- Simplest implicit method for solving ODEs with initial value
- Similar to forward Euler
- Inaccurate but stable
- net error proportional to dt (useful for large time steps!)
- Standard integration method for example in NEURON





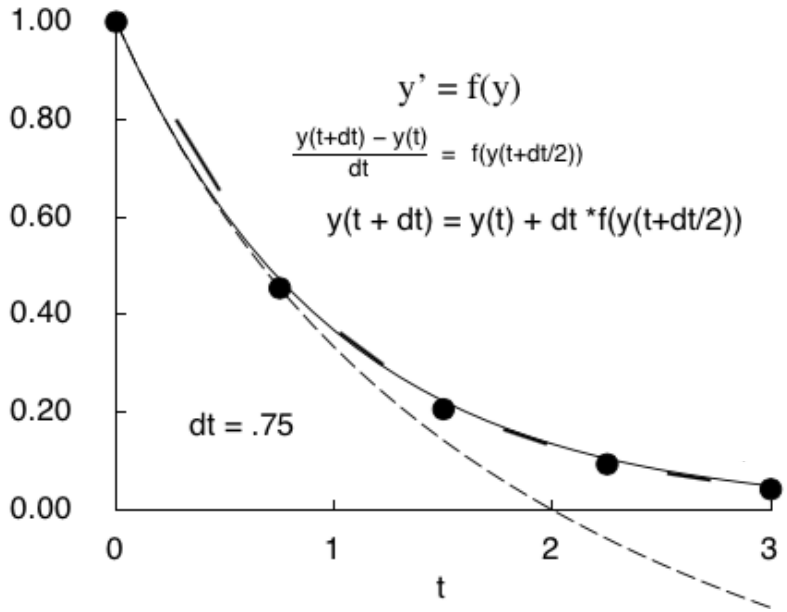


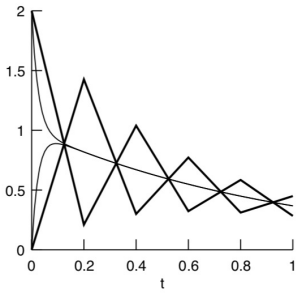


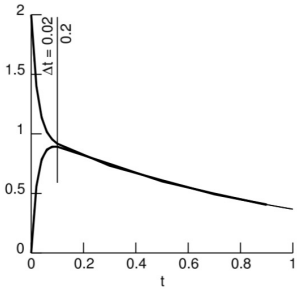


Crank-Nicolson

- Implicit method
- Combination of forward Euler & backward Euler (half step each)
- Error proportional to dt^2 (useful for small time steps)
- Well suited for HH-style ionic currents

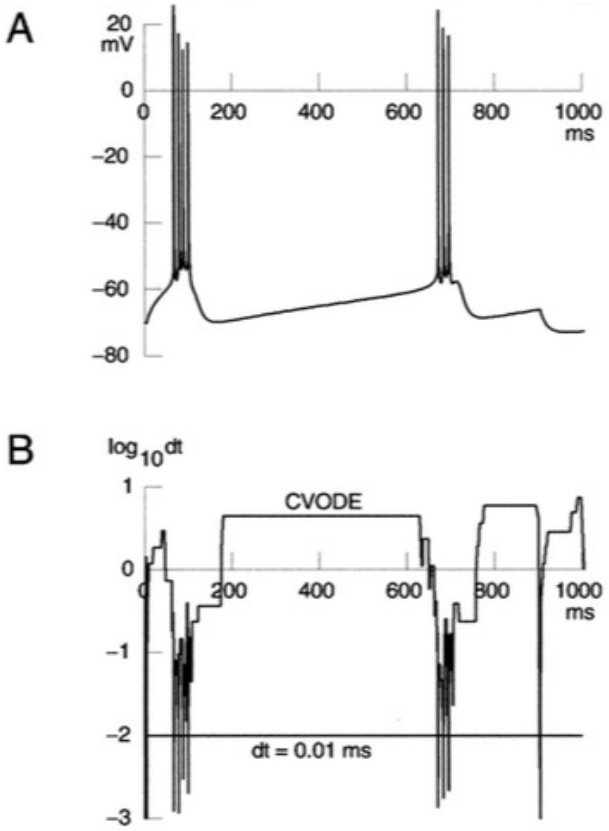






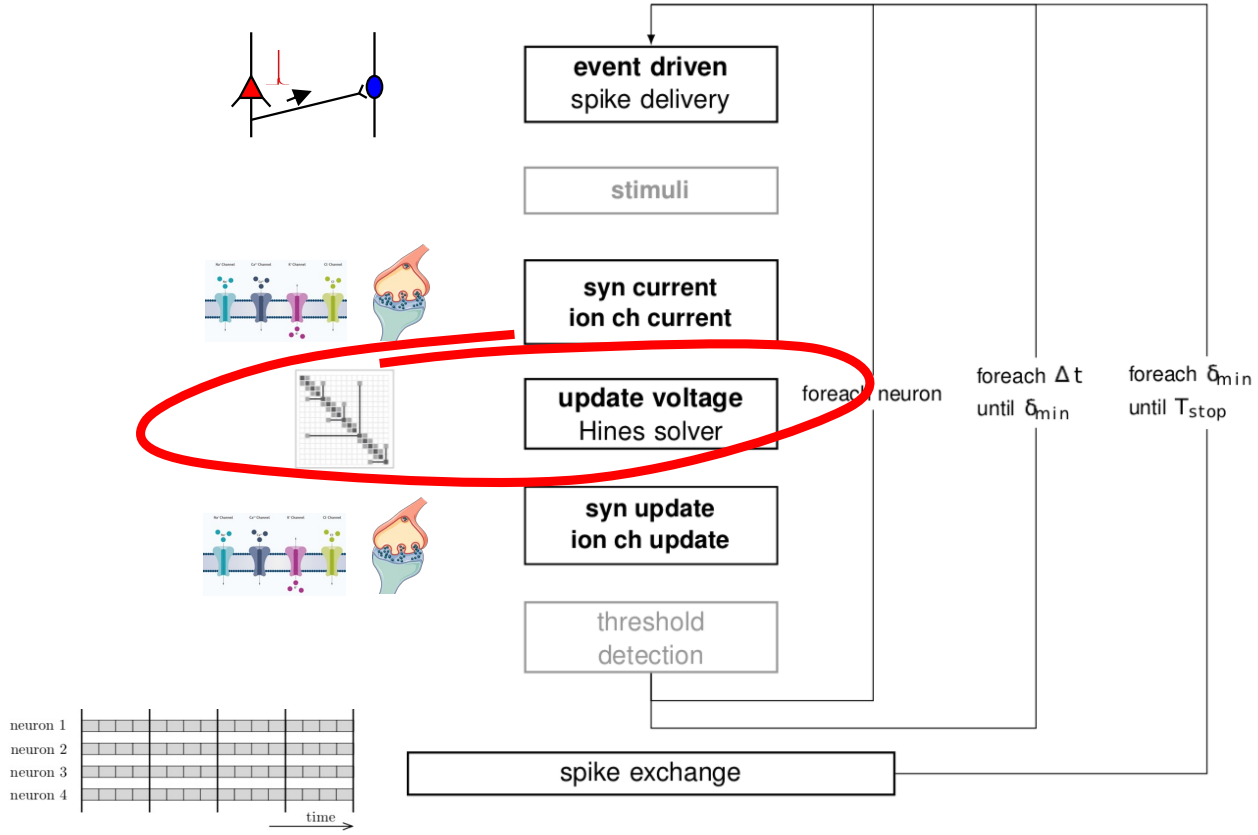
Adaptive Time Step Methods

- Define a maximum allowable absolute error rather than a time step
- Integrator adjusts time step dt so that the estimated local error is always less then the absolute error
- Can be 10x faster than fixed time even though more complex
- E.g. NEURON uses CVODE (Cohen & Hindmarsh 1994)

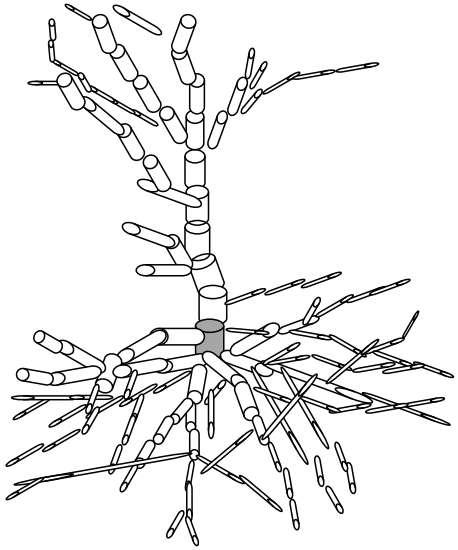


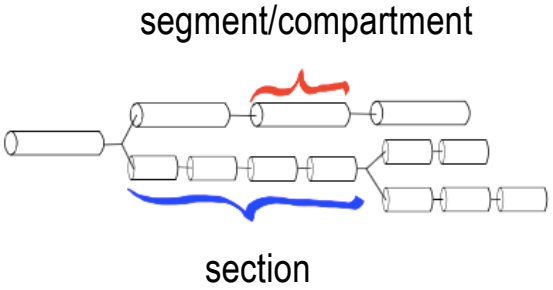
Hines and Carnevale, 2001

Simulation Loop for Biophysically Detailed Neurons/Networks



Multi-Compartment Modeling

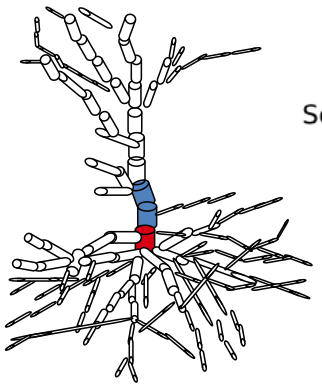


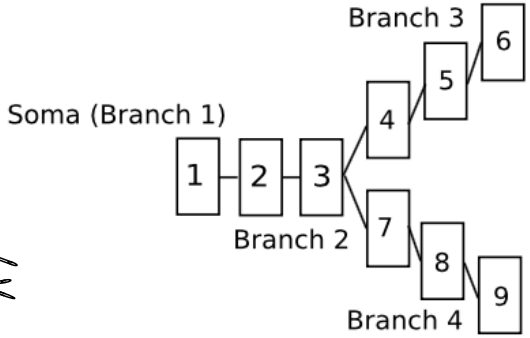


For each compartment *i*:

$$C_{m_i}\frac{dV_i}{dt} = g_{m_i}(E_{leak_i} - V_i) + \sum_k g_{k_i}(E_{k_i} - V_i) + \sum_j \frac{V_j - V_i}{(r_{a_j} + r_{a_i})/2}$$

Multi-compartment Modeling - Naïve Numbering





$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_{2,1} & A_{2,2} & A_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{3,2} & A_{3,3} & A_{3,4} & 0 & 0 & A_{3,7} & 0 & 0 \\ 0 & 0 & A_{4,3} & A_{4,4} & A_{4,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{5,4} & A_{5,5} & A_{5,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & A_{6,5} & A_{6,6} & 0 & 0 & 0 \\ 0 & 0 & A_{7,3} & 0 & 0 & 0 & A_{7,7} & A_{7,8} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & A_{8,7} & A_{8,8} & A_{8,9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_{9,8} & A_{9,9} \end{bmatrix}$$

Requires general Gaussian elimination of complexity $O(n^3)$

EPFL

Hines Algorithm

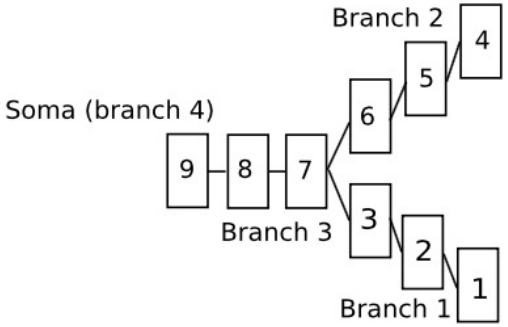
EFFICIENT COMPUTATION OF BRANCHED NERVE EQUATIONS

MICHAEL HINES

Department of Physiology, Duke University Medical Center, Durham, N.C. 27710 (U.S.A.)

(Received 19 March, 1983)

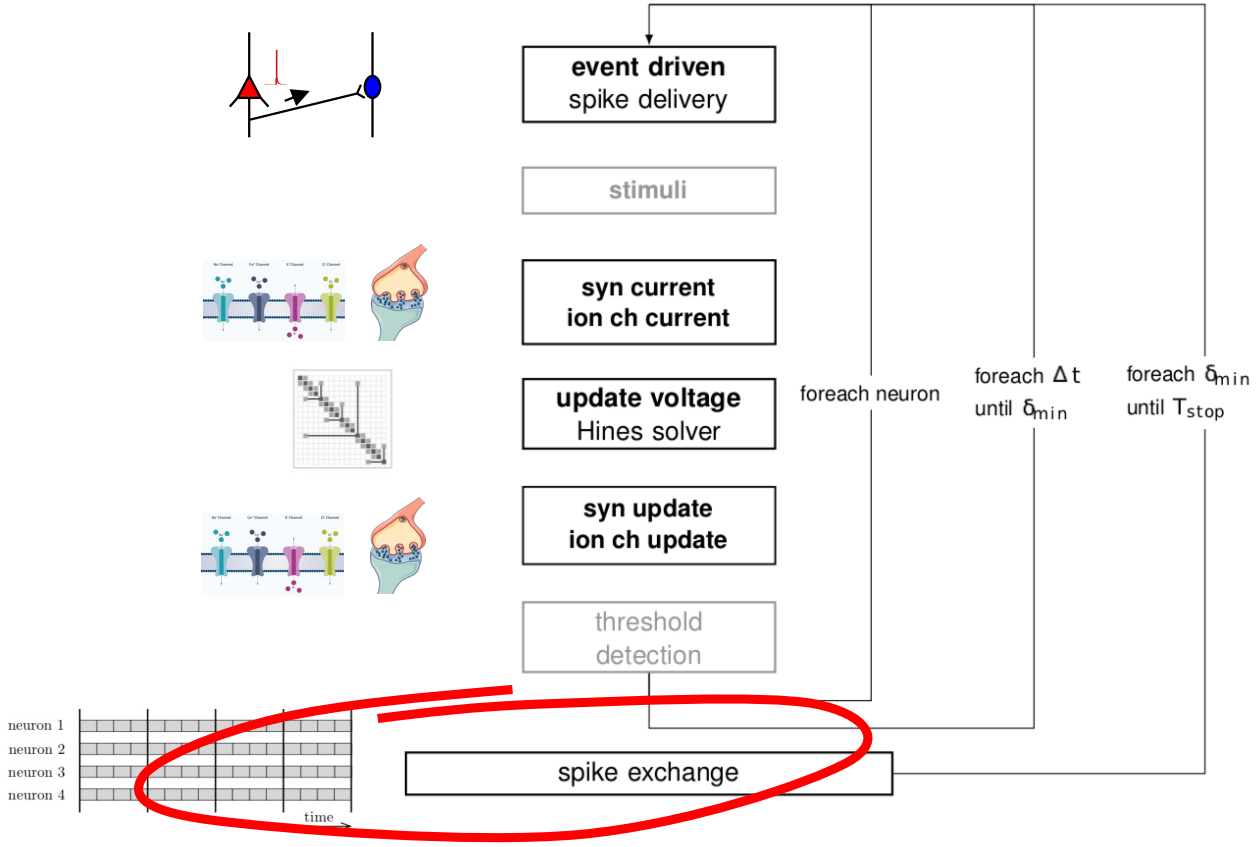
Int J Biomed Comput. 1984 Jan-Feb;15(1):69-76.



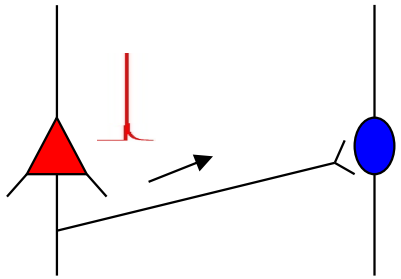
$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_{2,1} & A_{2,2} & A_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{3,2} & A_{3,3} & 0 & 0 & 0 & A_{3,7} & 0 & 0 \\ 0 & 0 & 0 & A_{4,4} & A_{4,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{5,4} & A_{5,5} & A_{5,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & A_{6,5} & A_{6,6} & A_{6,7} & 0 & 0 \\ 0 & 0 & A_{7,3} & 0 & 0 & A_{7,6} & A_{7,7} & A_{7,8} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & A_{8,7} & A_{8,8} & A_{8,9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_{9,8} & A_{9,9} \end{bmatrix}$$

Can be solved with a slightly modified Tridiagonal solver! O(n)

Simulation Loop for Biophysically Detailed Neurons/Networks



Minimal Axonal Delay



No spike evoked at time *t* can arrive at any postsynaptic cell sooner than $t+\min\{t_{axonaldelay}\}$

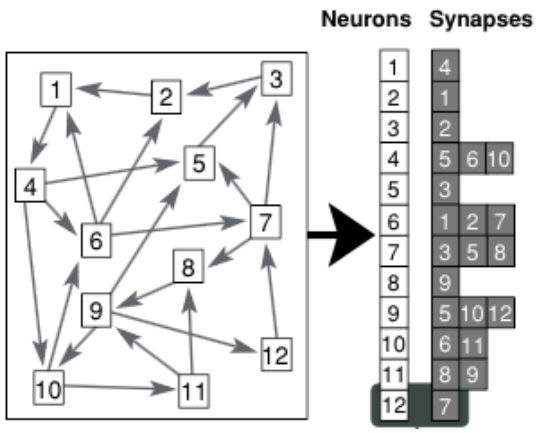
→ Each cell can be integrated independently for $\min\{t_{axonaldelay}\}$

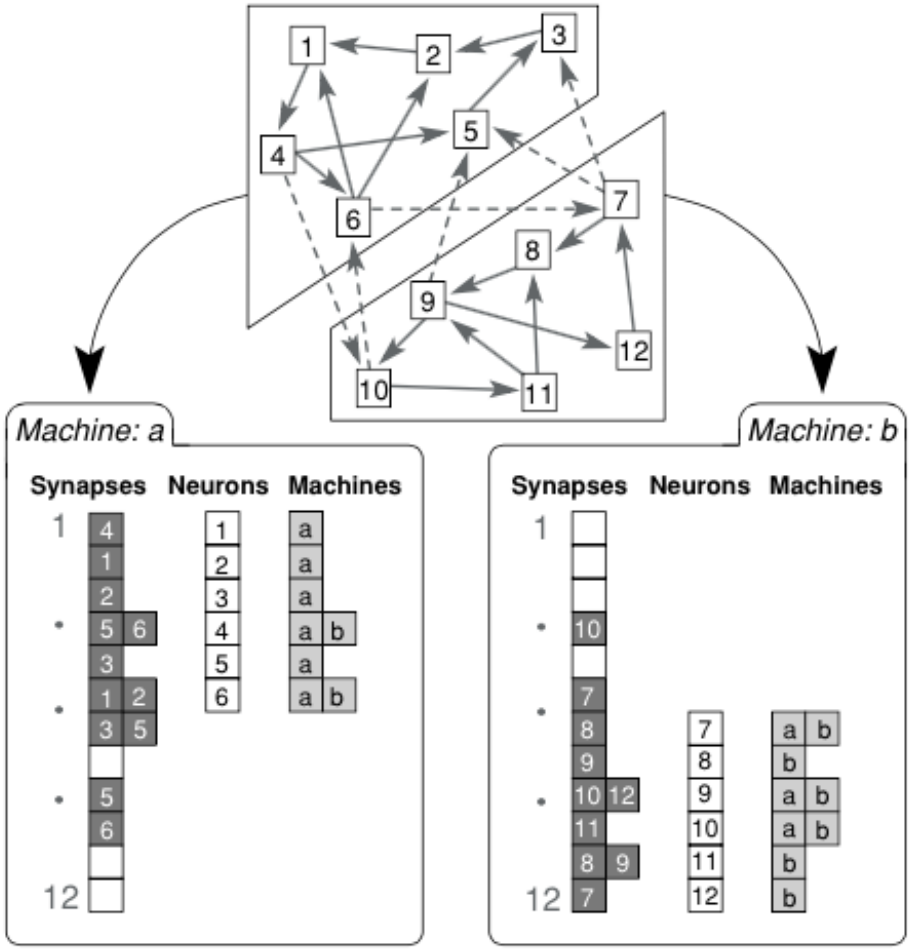
Morrison et al., Advancing the Boundaries of High-Connectivity Network Simulation with Distributed Computing, *Neural Computation* **17, 1776–1801** (2005)

Migliore et al., Parallel network simulations with NEURON, J Comput Neurosci. 2006 Oct;21(2):119-29.

EPFL

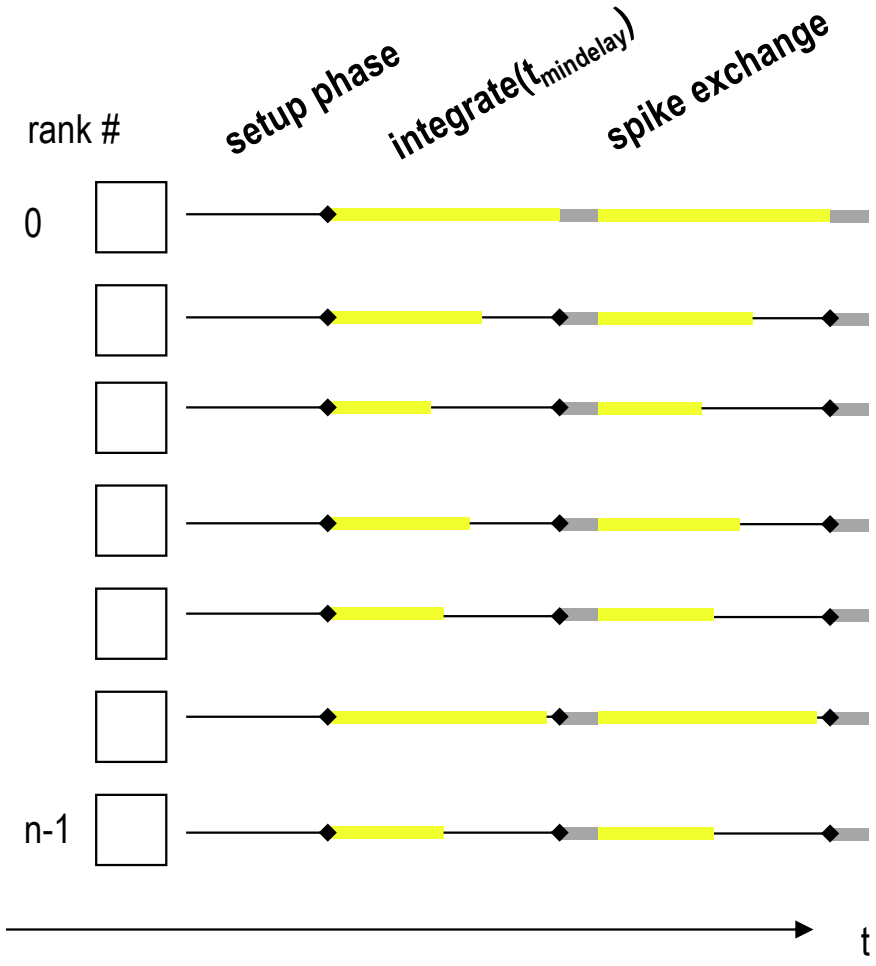
Parallelizing a Network





Morrison et al., Advancing the Boundaries of High-Connectivity Network Simulation with Distributed Computing, *Neural Computation* **17, 1776–1801** (2005)

Parallel Execution of Neural Networks



EPFL

Summary 2

- Simulation engines for simulation of neurons and networks are essential tools for computational neuroscience research
- There exist many neurosimulators and most likely you do not need to (and should not!) write your own
- Depending on your level of detail in your model, NEURON may be the tool of choice (especially for multi-compartment models)
  - It offers an easy-to-use UI as well as scriptable and efficient executables
  - It offers all the numerical methods needed (numerical integration, linear algebra, random numbers etc)
  - It runs on your desktop as well as on massively parallel supercomputers

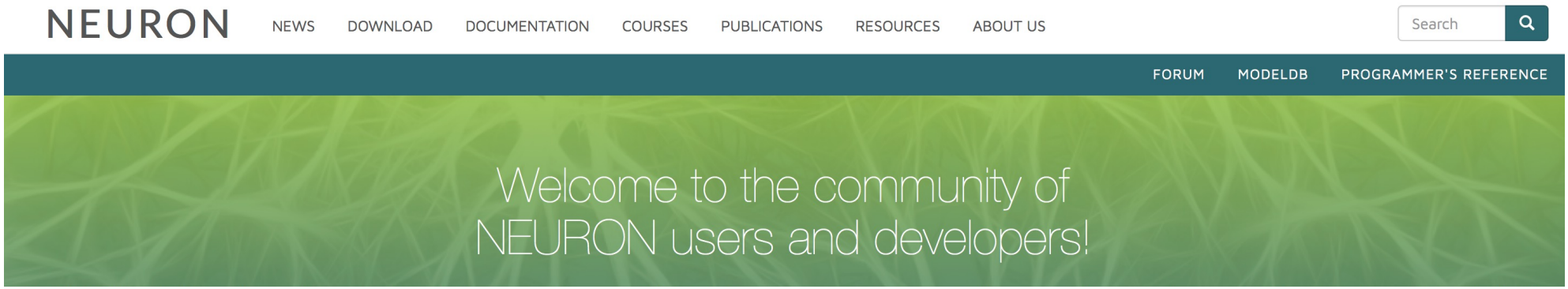
EPFL

Lecture Overview

- Scope
- Approaches
- **Applications**

EPFL

The NEURON Simulation Environment (neuron.yale.edu)



- Efficient, domain-specific simulator for multi-compartment, conductance-based simulations of neurons and networks
- De-facto standard for cellular-level models: more than 2300 scientific publications using NEURON
- Python support and extensible
- Extreme performance, runs on desktop and largest supercomputers

EPFL

The NEURON Book, Carnevale, N. T., and Hines M. L. , (2006)

Research Using Simulation

>2300 publications using NEURON

Please cite this article in press as: Billeh et al., Systematic Integration of Structural and Functional Data into Multi-scale Models of Mouse Primary Visual Cortex, Neuron (2020), https://doi.org/10.1016/j.neuron.2020.01.040

Neuron
NeuroResource

CellPress

Systematic Integration of Structural and Functional Data into Multi-scale Models of Mouse Primary Visual Cortex

Yazan N. Billeh,[1,*] Binghuang Cai,[1] Sergey L. Gratiy,[1] Kael Dai,[1] Ramakrishnan Iyer,[1] Nathan W. Gouwens,[1] Reza Abbasi-Asl,[1,2] Xiaoxuan Jia,[1] Joshua H. Siegle,[1] Shawn R. Olsen,[1] Christof Koch,[1] Stefan Mihalas,[1] and Anton Arkhipov[1,3,*]

[1]Allen Institute for Brain Science, Seattle, WA, USA
[2]UCSF Weill Institute for Neurosciences, Department of Neurology, University of California, San Francisco, CA, USA
[3]Lead Contact
*Correspondence: yazanb@alleninstitute.org (Y.N.B.), antona@alleninstitute.org (A.A.)
https://doi.org/10.1016/j.neuron.2020.01.040

Cell

Resource

Reconstruction and Simulation of Neocortical Microcircuitry

Henry Markram,[1,2,19,*] Eilif Muller,[1,19] Srikanth Ramaswamy,[1,19] Michael W. Reimann,[1,19] Marwan Abdellah,[1] Carlos Aguado Sanchez,[1] Anastasia Ailamaki,[16] Lidia Alonso-Nanclares,[6,7] Nicolas Antille,[1] Selim Arsever,[1] Guy Antoine Atenekeng Kahou,[1] Thomas K. Berger,[2] Ahmet Bilgili,[1] Nenad Buncic,[1] Athanassia Chalimourda,[1] Giuseppe Chindemi,[1] Jean-Denis Courcol,[1] Fabien Delalondre,[1] Vincent Delattre,[2] Shaul Druckmann,[4,5] Raphael Dumusc,[1] James Dynes,[1] Stefan Eilemann,[1] Eyal Gal,[4] Michael Emiel Gevaert,[1] Jean-Pierre Ghobril,[2] Albert Gidon,[3] Joe W. Graham,[1] Anirudh Gupta,[2] Valentin Haenel,[1] Etay Hay,[3,4] Thomas Heinis,[1,16,17] Juan B. Hernando,[8] Michael Hines,[12] Lida Kanari,[1] Daniel Keller,[1] John Kenyon,[1] Georges Khazen,[1] Yihwa Kim,[1] James G. King,[1] Zoltan Kisvarday,[13] Pramod Kumbhar,[1] Sébastien Lasserre,[1,15] Jean-Vincent Le Bé,[2] Bruno R.C. Magalhães,[1] Angel Merchán-Pérez,[6,7] Julie Meystre,[2] Benjamin Roy Morrice,[1] Jeffrey Muller,[1] Alberto Muñoz-Céspedes,[6,7] Shruti Muralidhar,[2] Keerthan Muthurasa,[1] Daniel Nachbaur,[1] Taylor H. Newton,[1] Max Nolte,[1] Aleksandr Ovcharenko,[1] Juan Palacios,[1] Luis Pastor,[9] Rodrigo Perin,[2] Rajnish Ranjan,[1,2] Imad Riachi,[1] José-Rodrigo Rodríguez,[6,7] Juan Luis Riquelme,[1] Christian Rössert,[1] Konstantinos Sfyrakis,[1] Ying Shi,[1,2] Julian C. Shillcock,[1] Gilad Silberberg,[18] Ricardo Silva,[1] Farhan Tauheed,[1,16] Martin Telefont,[1] Maria Toledo-Rodriguez,[14] Thomas Tränkler,[1] Werner Van Geit,[1] Jafet Villafranca Díaz,[1] Richard Walker,[1] Yun Wang,[10,11] Stefano M. Zaninetta,[1] Javier DeFelipe,[6,7,20] Sean L. Hill,[1,20] Idan Segev,[3,4,20] and Felix Schürmann[1,20]

The microcircuits of striatum in silico

J. J. Johannes Hjorth[a,1], Alexander Kozlov[a,b,1], Ilaria Carannante[a,2], Johanna Frost Nylén[b,2], Robert Lindroos[b,2], Yvonne Johansson[b,3], Anna Tokarska[b,3], Matthijs C. Dorst[b,3], Shreyas M. Suryanarayana[b], Gilad Silberberg[b], Jeanette Hellgren Kotaleski[a,b,4,5], and Sten Grillner[b,4,5]

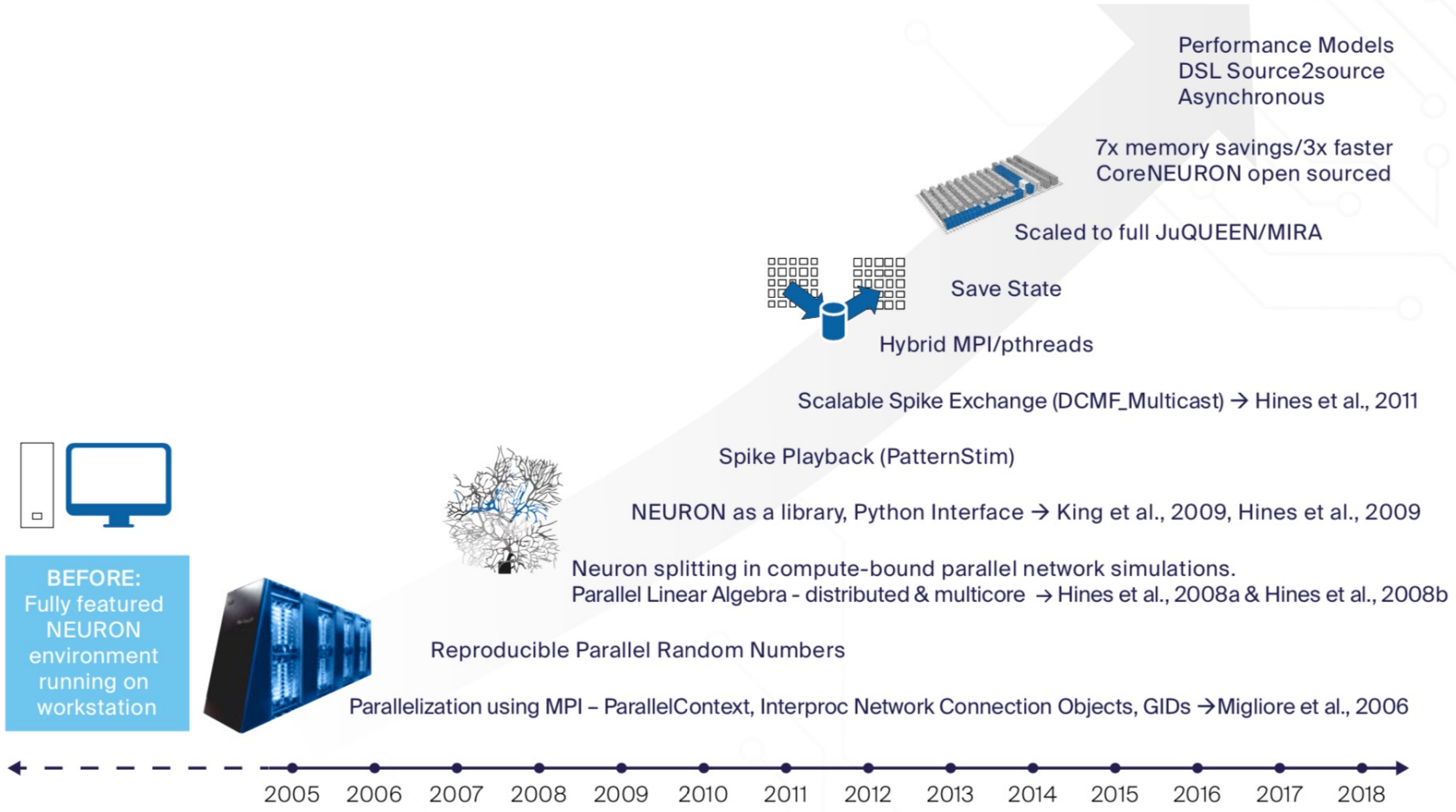
[a]Science for Life Laboratory, School of Electrical Engeneering and Computer Science, Royal Institute of Technology, SE-10044 Stockholm, Sweden; and [b]Department of Neuroscience, Karolinska Institutet, SE-17165 Stockholm

9554–9565 | PNAS | April 28, 2020 | vol. 117 | no. 17

EPFL

Research on Simulation

Modernizing the NEURON simulator for sustainability, portability, and performance
Awile et al. 2022



Research Software sticks around: almost 40 years and counting

frontiers | Frontiers in Neuroinformatics

TECHNOLOGY AND CODE
published: 27 June 2022
doi: 10.3389/fninf.2022.884046

Check for updates

Modernizing the NEURON Simulator for Sustainability, Portability, and Performance

Omar Awile[1†], Pramod Kumbhar[1†], Nicolas Cornu[1], Salvador Dura-Bernal[2,3], James Gonzalo King[1], Olli Lupton[1], Ioannis Magkanaris[1], Robert A. McDougal[4,5,6], Adam J. H. Newton[2,4], Fernando Pereira[1], Alexandru Săvulescu[1], Nicholas T. Carnevale[7‡], William W. Lytton[3‡], Michael L. Hines[7‡] and Felix Schürmann[1‡]*

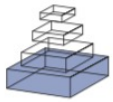
[1] Blue Brain Project, École Polytechnique Fédérale de Lausanne (EPFL), Geneva, Switzerland, [2] Department Physiology and Pharmacology, SUNY Downstate, Brooklyn, NY, United States, [3] Center for Biomedical Imaging and Neuromodulation, Nathan Kline Institute for Psychiatric Research, Orangeburg, NY, United States, [4] Department of Biostatistics, Yale School of Public Health, New Haven, CT, United States, [5] Program in Computational Biology and Bioinformatics, Yale University, New Haven, CT, United States, [6] Yale Center for Medical Informatics, Yale University, New Haven, CT, United States, [7] Department of Neuroscience, Yale University, New Haven, CT, United States

EPFL

Linden et al. 2014

frontiers in
NEUROINFORMATICS

ORIGINAL RESEARCH ARTICLE
published: 16 January 2014
doi: 10.3389/fninf.2013.00041



LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons

Henrik Lindén[1,2†], Espen Hagen[1†], Szymon Łęski[1,3], Eivind S. Norheim[1], Klas H. Pettersen[1,4] and Gaute T. Einevoll[1*]

[1] *Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Ås, Norway*
[2] *Department of Computational Biology, School of Computer Science and Communication, Royal Institute of Technology (KTH), Stockholm, Sweden*
[3] *Department of Neurophysiology, Nencki Institute of Experimental Biology, Warsaw, Poland*
[4] *CIGENE, Norwegian University of Life Sciences, Ås, Norway*

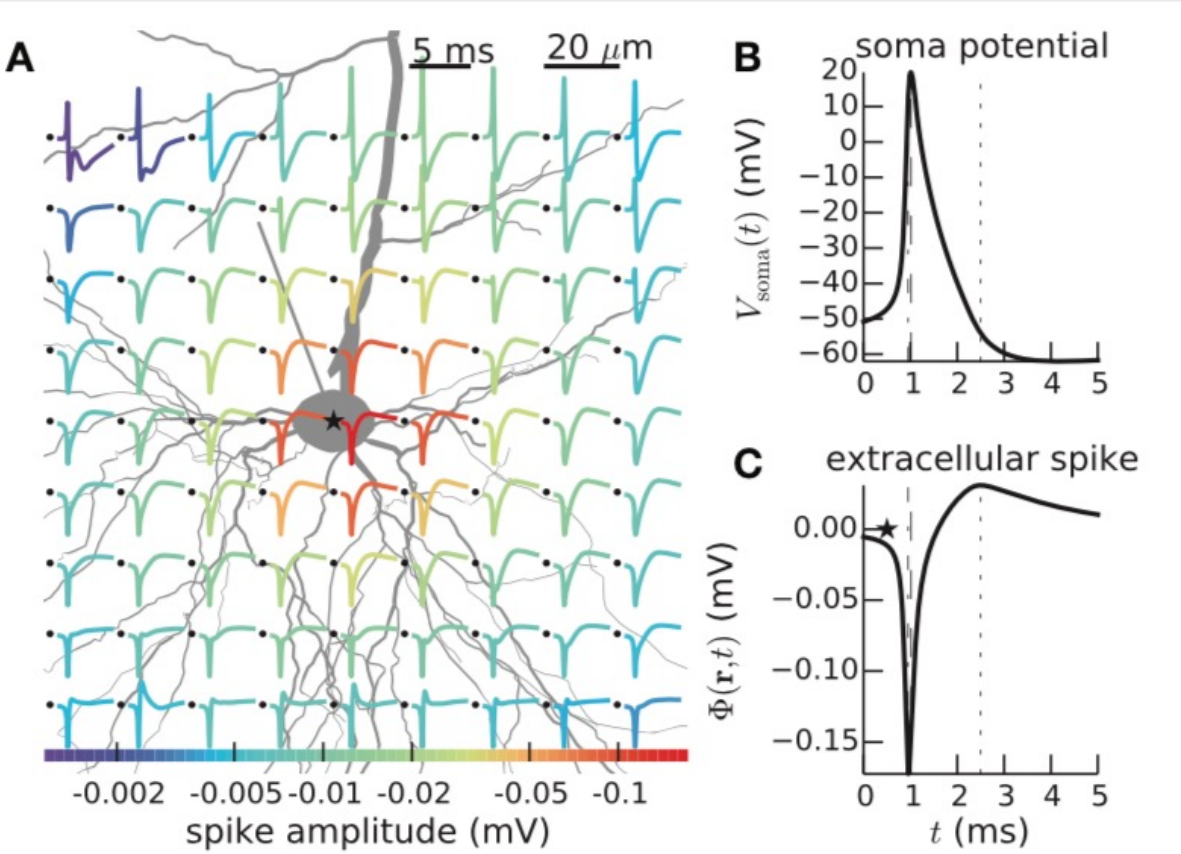
Extracellular Potentials

- The study aims to computationally derive the extracellular signature of neuronal activity in brain tissue models
- These extra-cellular potentials are generated by the summed distance-dependent contributions from transmembrane currents
- There are two main electrical signals recorded in experimental literature
  - High-frequency signals (>= 500 Hz) referred to as Multi-unit activity mostly reflecting spiking activity tens of micrometers from the electrode contact
  - Low-frequency signals referred to as local field potential mostly reflecting synaptic integration in populations of neurons within radii of hundreds of micrometers from the electrode contact
- Follow-up paper (Hagen et al., 2018) extends this also to other signals such as ECoG, EEG, MEG

EPFL

Calculating Extracellular Potentials

Position-dependent extracellular spike waveforms during an action potential in a rat L5b pyramidal-cell model (Hay et al., 2011). Black dots correspond to the positions of the (virtual) electrode contact points. Spike traces at each position are normalized and color coded according to the magnitude of the negative peak.



From Neuron models to Extracellular Potential

1.Calculation of transmembrane currents of each neuron, using multicompartmental model neurons derived from detailed morphological reconstructions of neurons within NEURON simulation environment (Carnevale and Hines, 2006; Carnevale, 2007).

2.Calculation of the extracellular potential from the trans-membrane currents using a biophysical forward-modeling formalism derived within so called *volume-conductor theory* (Hämäläinen et al., 1993; Nunez and Srinivasan, 2006).

EPFL

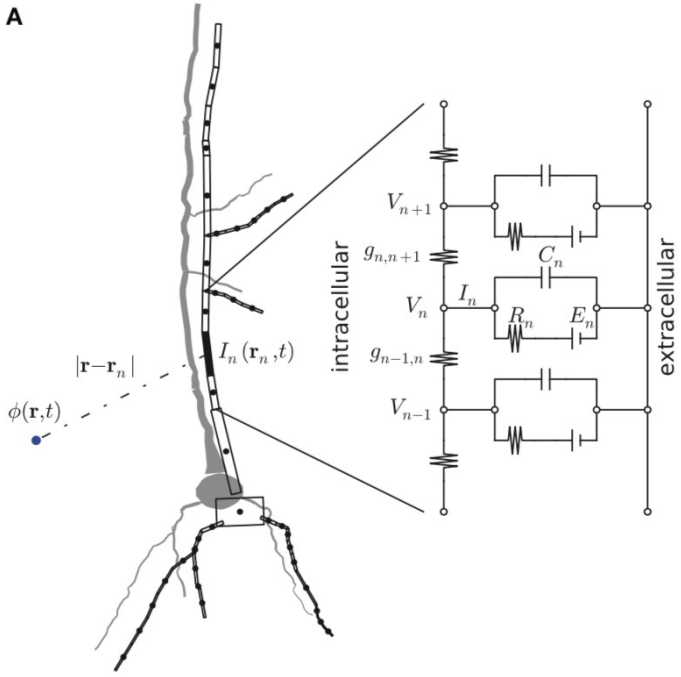
From Transmembrane Currents to Electrical Potential

Currents in a single neurons:

$$g_{n,\,n+1}(V_{n+1} - V_n) - g_{n-1,\,n}(V_n - V_{n-1}) = C_n \frac{dV_n}{dt} + \sum_j I_n^j$$

Forward Model for Extracellular potential[1]:

$$\phi(\mathbf{r},\ t) = \frac{1}{4\pi\sigma} \frac{I_0(t)}{|\mathbf{r} - \mathbf{r}_0|}$$



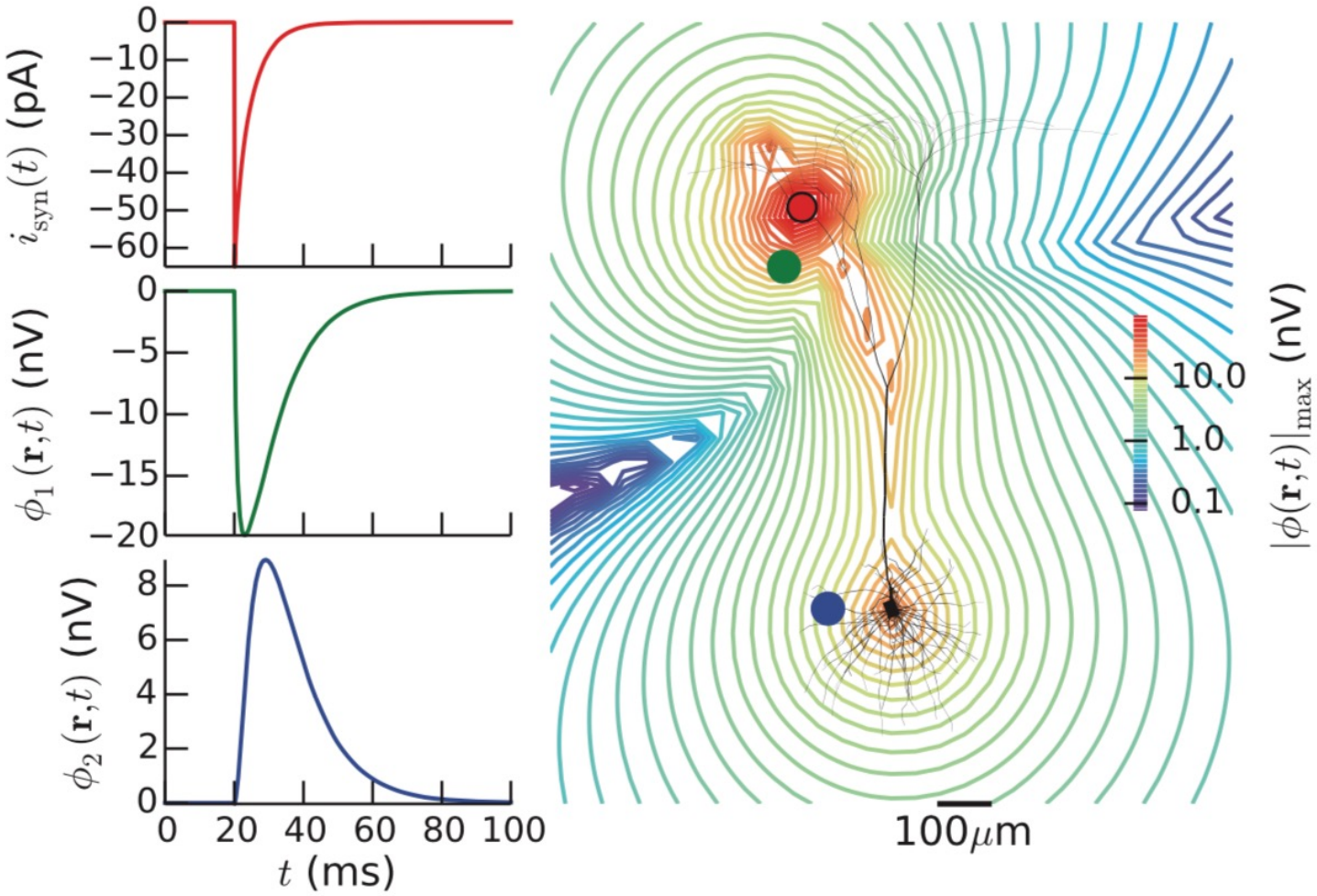
Assuming:

- Quasi-static approximation (E and B fields effectively decouple)
- linear, isotropic, homogeneous and ohmic extracellular medium

1) In practice, the LFPy does use the line source approximation for neuronal processess and sphere for soma

Results

FIGURE 3 | Extracellular potential generated by a single synaptic input produced by executing `example1.py`. Extracellular potentials (**middle** and **bottom left** panels) generated at positions marked by green and blue dot, respectively, by a synaptic input current (**upper left** panel) injected in the apical dendrite at the position marked by red dot. The pyramidal cell corresponds to a layer-5 pyramidal cell from cat visual cortex with passive membranes but without adjustment of the membrane area to compensate for spines (Mainen and Sejnowski, 1996). The contour plot shows equipotential lines for the maximum magnitude of the extracellular potential in the *xz*-plane.



Results

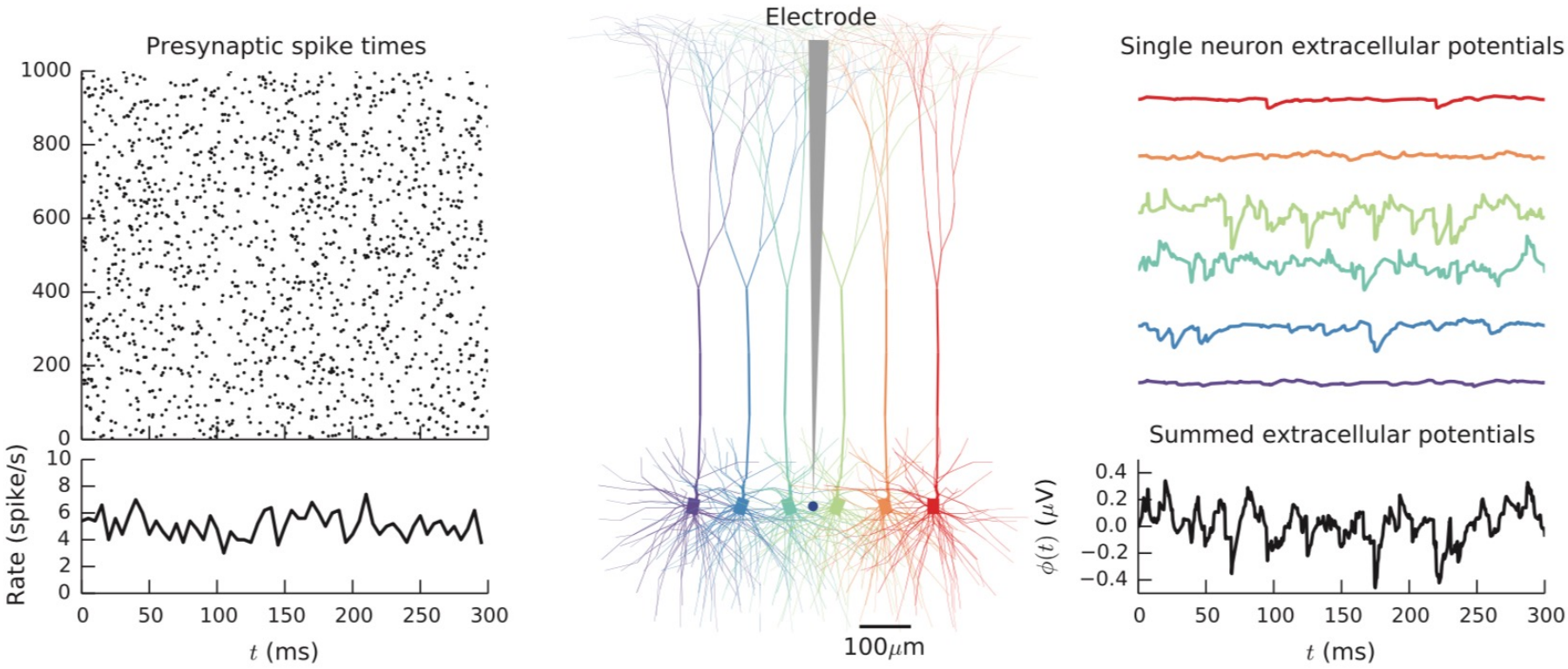


FIGURE 4 | Simulation of extracellular potentials from a population of neurons using MPI. A population of pyramidal cells (**Middle** panel) receiving input spikes from a presynaptic pool of spike trains (**Left** panel) is simulated by distributing cells on different MPI processes and by collecting their individual contributions in the root MPI process. Summation of the individual contribution then gives the total population potential (**Right** panel). Results shown come from executing `example3.py`.

Take Home Messages from Linden et al. 2014

- Example of how the simulation results of biophysically detailed neuron models done in NEURON can be used to derive additional biophysical signals through a forward model
- Specifically, LFPy sums up and geometrically weighs the trans-membrane currents of neuron models as computed by NEURON and calculates the electrical potential
- Under the hood, LFPy invokes NEURON to calculate the transmembrane currents

EPFL

Dura-Bernal et al. 2019

eLIFE elifesciences.org TOOLS AND RESOURCES

NetPyNE, a tool for data-driven multiscale modeling of brain circuits

Salvador Dura-Bernal[1]*, Benjamin A Suter[2†], Padraig Gleeson[3], Matteo Cantarelli[4], Adrian Quintana[5], Facundo Rodriguez[1,4], David J Kedziora[6], George L Chadderdon[1‡], Cliff C Kerr[6], Samuel A Neymotin[1,7], Robert A McDougal[8,9], Michael Hines[8], Gordon MG Shepherd[2], William W Lytton[1,10]

Abstract Biophysical modeling of neuronal networks helps to integrate and interpret rapidly growing and disparate experimental datasets at multiple scales. The NetPyNE tool (www.netpyne.org) provides both programmatic and graphical interfaces to develop data-driven multiscale network models in NEURON. NetPyNE clearly separates model parameters from implementation code. Users provide specifications at a high level via a standardized declarative language, for example connectivity rules, to create millions of cell-to-cell connections. NetPyNE then enables users to generate the NEURON network, run efficiently parallelized simulations, optimize and explore network parameters through automated batch runs, and use built-in functions for visualization and analysis – connectivity matrices, voltage traces, spike raster plots, local field potentials, and information theoretic measures. NetPyNE also facilitates model sharing by exporting and importing standardized formats (NeuroML and SONATA). NetPyNE is already being used to teach computational neuroscience students and by modelers to investigate brain regions and phenomena.

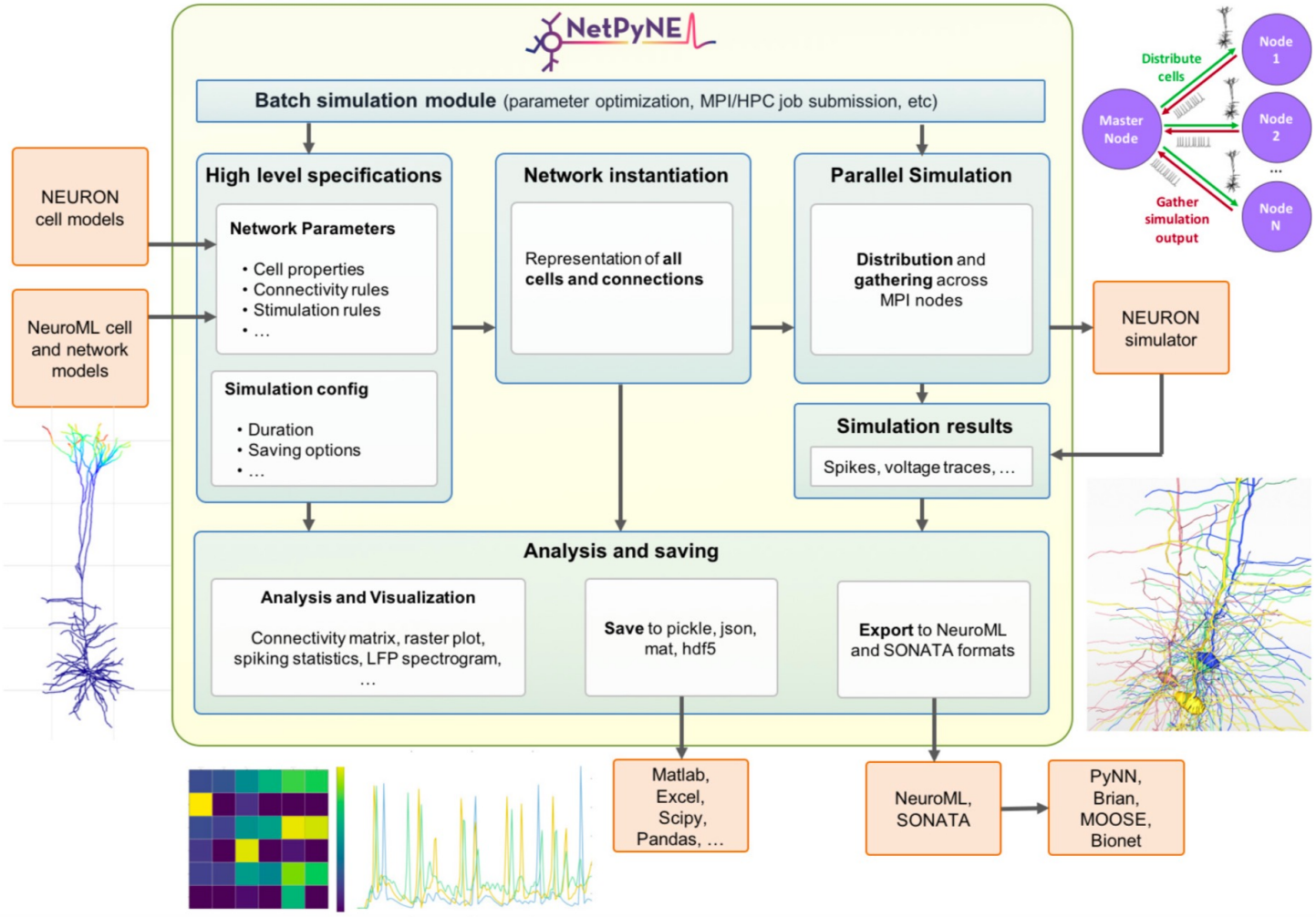
Dura-Bernal *et al.* eLife 2019;8:e44494. DOI: https://doi.org/10.7554/eLife.44494

Premise

- NetPyNE = Networks using Python and NEURON
- Simplify the work with large-scale neural networks, in particular
  - (1) flexible, rule-based, high-level standardized specifications covering scales from molecule to cell to network;
  - (2) efficient parallel simulation both on stand-alone computers and in high-performance computing (HPC) clusters;
  - (3) automated data analysis and visualization (e.g. connectivity, neural activity, information theoretic analysis);
  - (4) standardized input/output formats, importing of existing NEURON cell models, and conversion to/from NeuroML (Gleeson et al., 2010; Cannon et al., 2014);
  - (5) automated parameter tuning across multiples scales (molecular to network) using grid search and evolutionary algorithms.
  - All tool features are available programmatically or via an integrated graphical user interface (GUI).

EPFL

NetPyNE - Schema

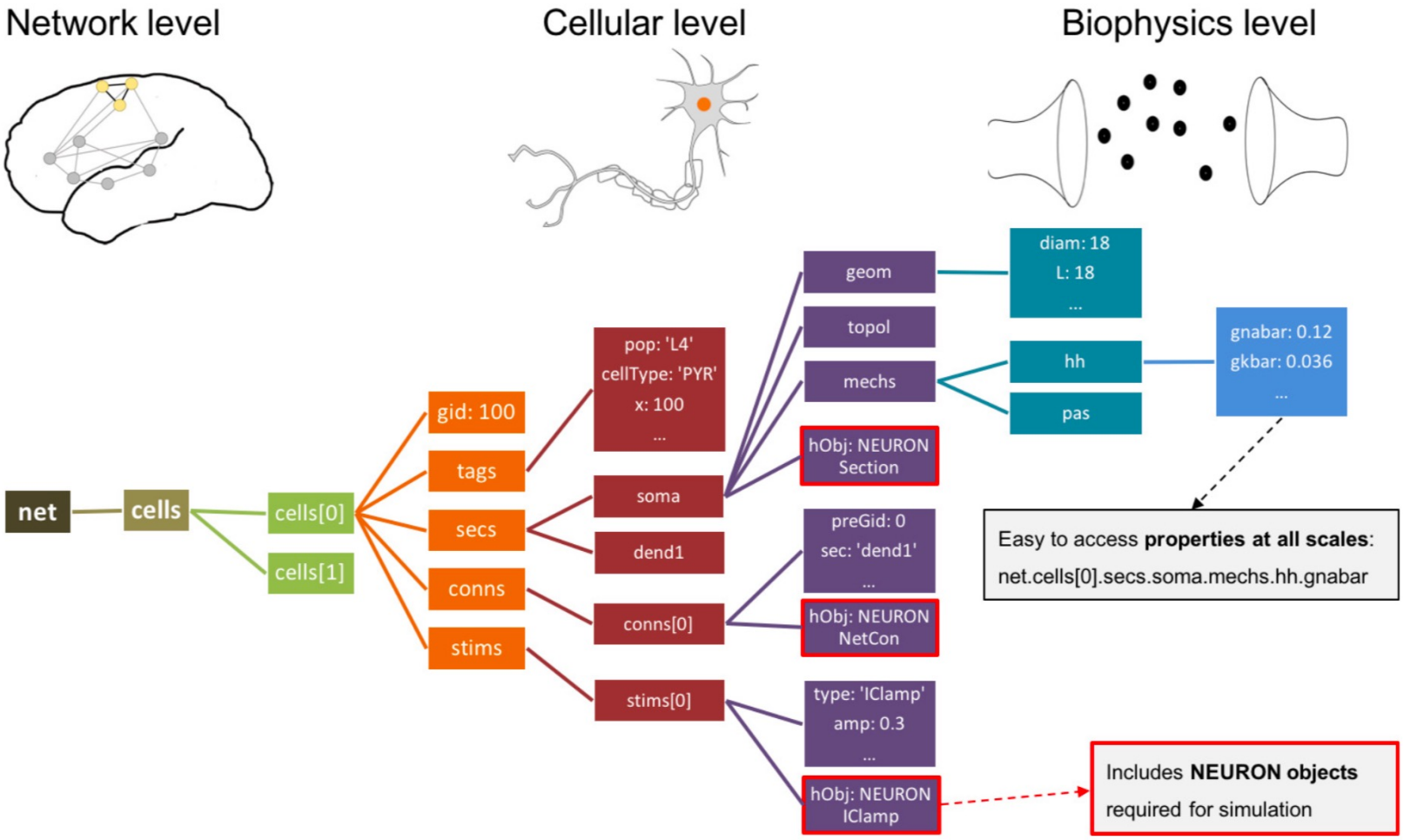


Related work

- neuroConstruct (Gleeson et al., 2007)
- PyNN (Davison, 2008)
- Topographica (Bednar, 2009)
- ARACHNE (Aleksin et al., 2017)
- BioNet (Gratiy et al., 2018)
- …

EPFL

Python Network Data-Structure



Less is More

Table 1. Number of lines of code in the original models and the NetPyNE reimplementations.

Model description (reference)	Original language	Original num lines	NetPyNE num lines
Dentate gyrus (***Tejada et al., 2014***)	NEURON/hoc	1029	261
CA1 microcircuits (***Cutsuridis et al., 2010***)	NEURON/hoc	642	306
Epilepsy in thalamocortex (***Knox et al., 2018***)	NEURON/hoc	556	201
EEG and MEG in cortex/HNN model (***Jones et al., 2009***)	NEURON/Python	2288	924
Motor cortex with RL (***Dura-Bernal et al., 2017***)	NEURON/Python	1171	362
Cortical microcircuits (***Potjans and Diesmann, 2014***)	PyNEST	689	198

EPFL

LFP Recordings

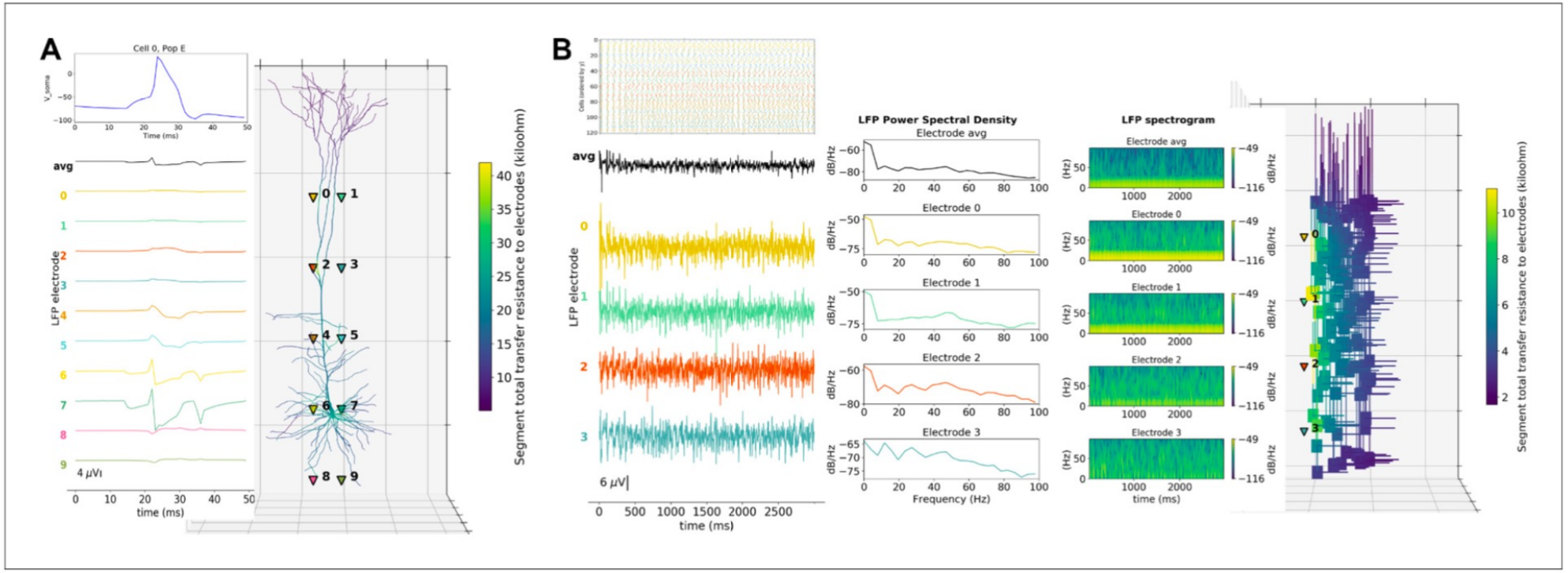
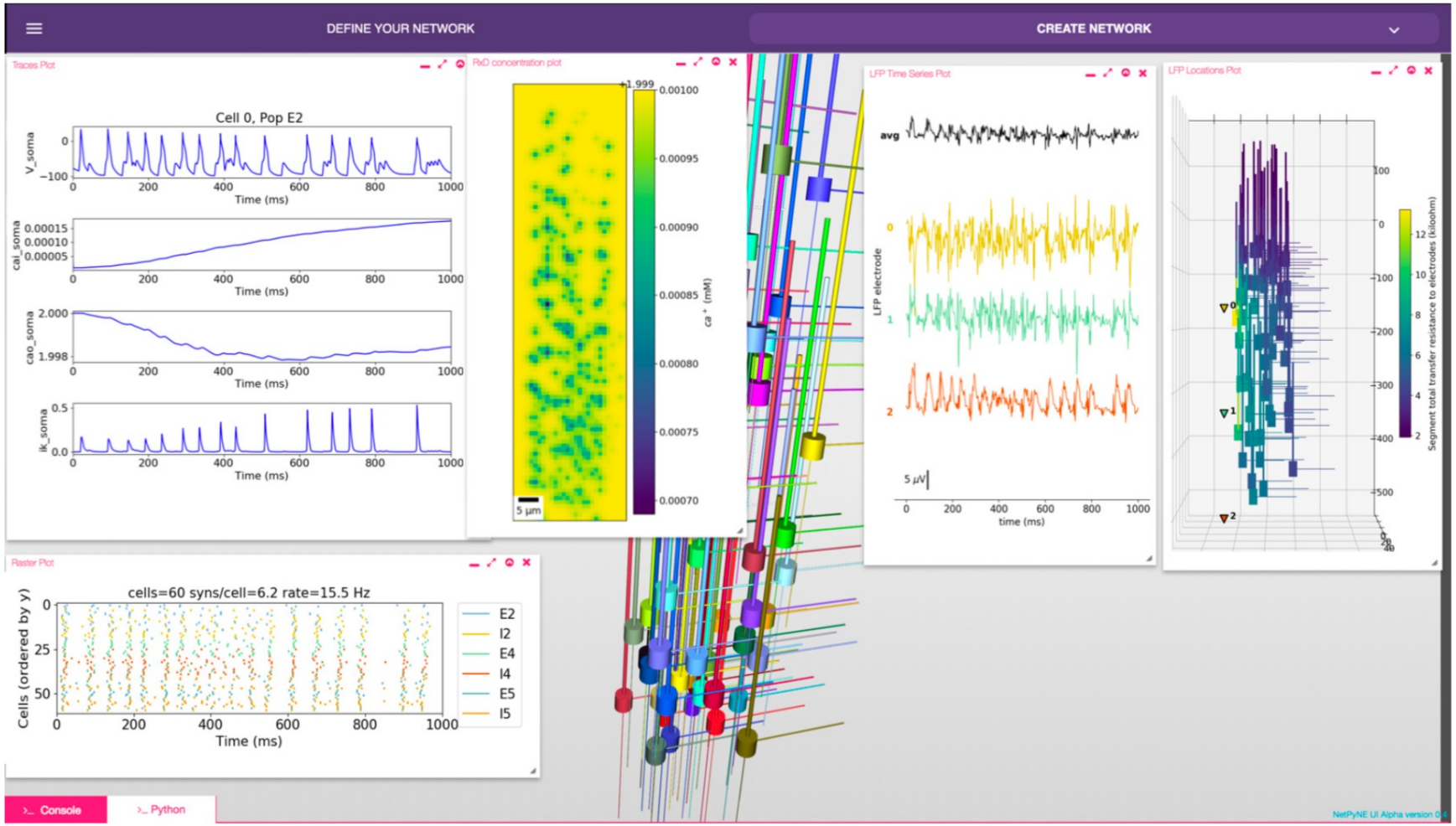


Figure 6. LFP recording and analysis. (A) LFP signals (left) from 10 extracellular recording electrodes located around a morphologically detailed cell (right) producing a single action potential (top-right). (B) LFP signals, PSDs and spectrograms (left and center) from four extracellular recording electrodes located at different depths of a network of 120 five-compartment neurons (right) producing oscillatory activity (top-left).

DOI: https://doi.org/10.7554/eLife.44494.008

EPFL

Graphical User Interface



Take Home Messages from Dura-Bernal et al. 2019

- NetPyNE is a high-level Python interface to the NEURON simulator that facilitates the definition, parallel simulation, optimization and analysis of data-driven brain circuit models.
- It provides multi-scale specifications using a declarative language; from molecular level chemo-physiology to network scale
- Integrated parameter optimization
- It can lead to more compact model specifications increasing clarity and reducing likeliness of bugs

EPFL

Summary 3

- For biophysically detailed models of neurons and networks, the NEURON simulation environment is the defacto standard
- Thousands of scientific studies have used NEURON, including the largest and most detailed brain tissue models to-date
- Furthermore, NEURON is used as a building block for multi-scale, multi-physics simulations
- Lastly, there is a rich eco-system of software facilitating the use of NEURON
- For other scales and scopes (e.g. stochastic reaction-diffusion, spiking neural networks), other simulators are to be considered as they have optimizations relevant at those scales

EPFL

Lecture Summary

- Scientific computing describes the opportunity and need to solve large and complex problems using programmable electronic computers and mathematical analysis
- Methods and algorithms for the simulation of neurons and networks (of diverse levels of detail) are the ingredients for scientific computing in neuroscience
- Several simulation packages have emerged over the last 40 years that are the de facto standard for simulation (NEURON for biophysically detailed simulations, NEST for large scale spiking neural networks, BRIAN for smaller spiking neural networks)
- These packages are under constant development to keep up with the development of computers and to advance the type of scientific studies that can be done

EPFL