Schedule of Presentations

NTC	Ted Carnevale
MLH	Michael Hines
WWL	Bill Lytton
GMS	Gordon Shepherd

Morning session

Afternoon session

2:00	MLH	Python + NEURON	95
3:15	Coffee Break		
3:30	GMS	Databases for computational neuroscience	103
4:00	WWL	Reaction-diffusion	supplement
4:45	MLH	Future directions	
5:00		End of afternoon session	

Receipt and Survey **last two pages**

We value your opinions and suggestions for improving this course. Please take a moment to fill out and hand in the survey.

Satellite Symposium, Society for Neuroscience

Using NEURON to Model Cells and Networks

San Diego, CA

Friday, November 8, 2013

Ted Carnevale
Michael Hines
Bill Lytton
Gordon Shepherd

Supported by NINDS

NEURON http://neuron.yale.edu/

NEURON: a brief tour

A tool for empirically-based models of neurons and neural circuits

Open source project directed by Michael Hines

Active development and user support

- Documentation, tutorials, and forum at http://www.neuron.yale.edu/
- Courses
  - SFN meetings
  - summer course at UCSD
  - other courses

The NEURON user community

Used by experimentalists, theoreticians, and educators for neuroscience research and teaching

As of October 2012

- more than 1180 publications
- more than 1500 subscribers to mailing list and forum http://www.neuron.yale.edu/phpBB/
- source code for 355 published models at ModelDB http://modeldb.yale.edu/

Specifying and using models with NEURON

- Model specifications written in hoc and/or Python
  - and/or
- created with GUI tools (work via hoc)
  - CellBuilder, Channel Builder,
  - Network Builder, Linear Circuit Builder
- Add new functionality with NMODL (compiled)
  - new density mechanisms and point processes
    - described by ODEs, kinetic schemes,
    - algebraic equations
  - events, state machines, artificial spiking cells

Not model specification, but necessary

- Instrumentation
  - stimulators, current or voltage clamps
  - plotting and recording variables
- Simulation control
  - default and custom initializations
  - integration methods
    - fixed time step
    - adaptive integration
  - event system useful for implementing
    - "experimental protocols"
- User interface

Other features

- Parallel simulation
  - multithreaded execution
  - embarrassingly parallel problems
  - distributed models
- Optimization tools
- Model analysis
  - Impedance tools
  - ModelView
- Import3D for detailed morphometric data

Where to learn more

- The NEURON Book
- The WWW page http://www.neuron.yale.edu/
  - Documentation
    - hints and tutorials
    - link to FAQ list
    - links to key papers about NEURON
  - Programmer's Reference
  - Courses
- The Forum http://www.neuron.yale.edu/phpBB/
  - Getting started
  - Hot tips

The What and the Why of Neural Modeling

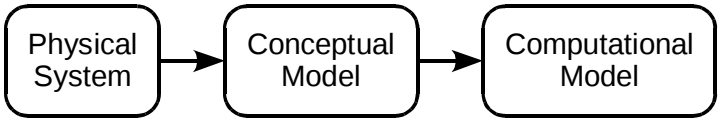
The moment-to-moment processing of information in the nervous system involves the propagation and interaction of electrical and chemical signals that are distributed in space and time.

Empirically-based modeling is needed to test hypotheses about the mechanisms that govern these signals and how nervous system function emerges from the operation of these mechanisms.

Topics

1. How to create and use models of neurons and networks of neurons
   - How to specify anatomical and biophysical properties
   - How to control, display, and analyze models and simulation results
2. How NEURON works
3. How to add user-defined biophysical mechanisms

From Physical System to Computational Model



Conceptual model

a simplified representation of the physical system

Computational model

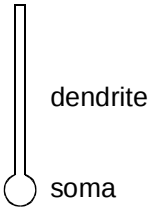
an accurate representation of the conceptual model

From Physical System to Computational Model

Physical system

Ca1 pyramidal cell

Conceptual model



ball and stick

Computational model

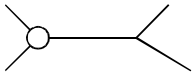
```
create soma, dendrite
connect dendrite(0), soma(1)
```

hoc code

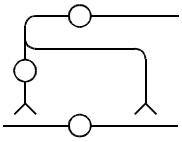
Hierarchies of Complexity

Structure

Single compartment 

Stylized 

Anatomically detailed 

Network 

Hierarchies of Complexity

Mechanism

Passive and Active currents
- HH-style
- kinetic scheme

Synaptic transmission
- continuous
- spike-triggered

Gap junctions

Extracellular fields, Linear circuits

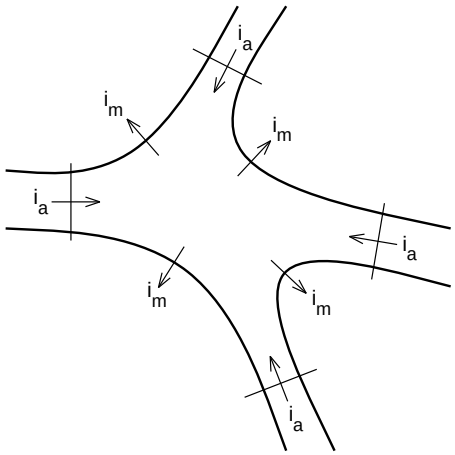
Diffusion, buffers, transport & exchange

Artificial spiking cells ("integrate & fire")

Fundamental Concepts in NEURON

Signals	What moves	Driving force	What is conserved
Electrical	charge carriers	voltage gradient	charge
Chemical	solute	concentration gradient	mass

Conservation of Charge



$$c_m \frac{dV_m}{dt} + i_{ion} = \sum i_a$$

The Model Equations

$$c_j \frac{dv_j}{dt} + i_{ion_j} = \sum_k \frac{v_k - v_j}{r_{jk}}$$

v_j membrane potential in compartment j

i_{ion_j} net transmembrane ionic current in compartment j

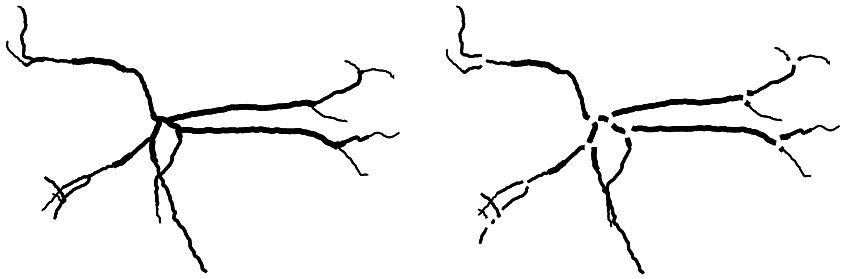
c_j membrane capacitance of compartment j

r_{jk} axial resistance between the centers of
compartment j
and
adjacent compartment k

Separating Anatomy and Biophysics from Purely Numerical Issues

section

a continuous length of unbranched cable



Anatomical data from A.I. Gulyás

Mathematical description of a section

What we want:

$$c_j \frac{dv_j}{dt} + i_{ion_j} = \sum_k \frac{v_k - v_j}{r_{jk}}$$

What a new section gives us:

$$c_j \frac{dv_j}{dt} = \sum_k \frac{v_k - v_j}{r_{jk}}$$

i.e. membrane capacitance and axial resistance,
but no ionic current.

How can we put ion channels in the membrane?

Adding mechanisms to sections

Density mechanisms
- distributed channels
- ion accumulation

Point processes
- electrodes, synapses

Described by
- differential equations
- kinetic schemes
- algebraic equations

Constructed with
- NMODL
- Channel Builder

```
create soma, dend
connect dend(0), soma(1)

soma {
  L = 50  // [um] length
  diam = 50  // [um] diameter
  insert hh  // Hodgkin-Huxley mechanism
  nseg = 1
}

dend {
  L = 200
  diam = 2
  insert pas // passive channels
  nseg = 3
}
```

Range Variables

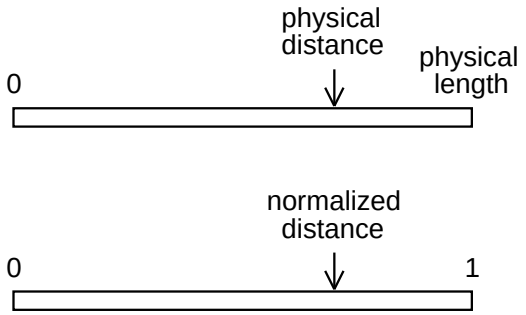
Name	Meaning	Units
diam	diameter	[µm]
cm	specific membrane capacitance	[µf/cm^2]
g_pas	specific conductance of the pas mechanism	[siemens/cm^2]
v	membrane potential	[mV]

range

normalized position along the length of a section

$0 \leq \text{range} \leq 1$

any variable name can be used for range, e.g. x



Syntax:

`sectionname.rangevar(range)`

returns or sets value of rangevar
at location that corresponds to range

Examples:

`dend.v(0.5)`

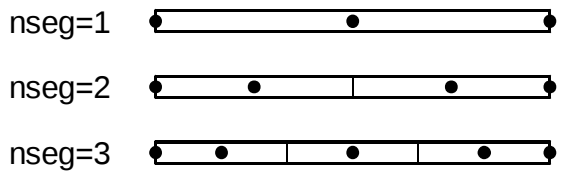
returns v at middle of `dend`
Shortcut: `dend.v`

`dend for (x) print x*L, v(x)`

prints physical distance and v
at each point in `dend` where v was calculated

nseg

the number of points in a section section where membrane current and potential are computed



Example: `axon nseg = 3`

To test spatial resolution

```
forall nseg = nseg*3
```

and repeat the simulation

Category	**Variable**	**Units**
Time	`t`	[ms]
Distance	`diam, L`	[µm]
Voltage	`v`	[mV]
Current		
specific	`i`	[mA/cm^2] (density)
absolute		[nA] (point process)
Capacitance		
specific	`cm`	[µf/cm^2]
absolute		[nf] (point process)
Conductance		
specific	`g`	[S/cm^2] (density)
absolute		[µS] (point process)
Cytoplasmic resistivity	`Ra`	[Ω cm]
Resistance	`SEClamp.rs`	[10^6 Ω]
Concentration	`cai`, `nao`, etc.	[mM]

Model specification summary

Topology: create and connect sections

Geometry: stylized (L & diam) or 3D (x,y,z,diam)

Biophysics: insert density mechanisms,
attach "biological" point processes (synapses)

Network models
- Create cells
- Connect cells

Construction and Use of Models

1. Specify the model ("virtual organism").
2. Specify the user interface ("virtual lab rig").
3. Tests
   - structural integrity
   - spatial grid
   - time steps

Example: using the GUI to build and exercise a stylized model

1. How to use the CellBuilder to create and manage a model cell.
2. How to use NEURON's graphical tools to make an interface for running simulations.

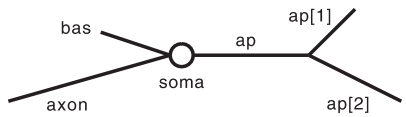
Step 0: Conceptualize the task

Shape
- stick figure / detailed

Channel distribution
- uniform / nonuniform
- whole cell / region / individual neurite

Creation
- single cell / use in a network

Step 1: using the CellBuilder to make a stylized model



Section	L	diam	Biophysics
soma	20 µm	20 µm	hh
ap[0]	400	2	reduced hh *
ap[1]	300	1	reduced hh *
ap[2]	500	1	reduced hh *
bas	200	3	pas §
axon	800	1	hh

* - gnabar_hh and gkbar_hh reduced to 10%, el_hh = - 64 mV

§ - e_pas = - 65 mV

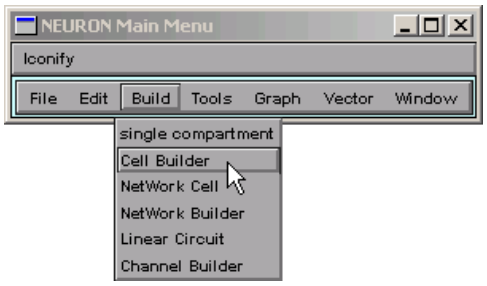
Throughout the cell Ra = 160 Ω cm, cm = 1 µf / cm^2

Launch NEURON with its library of graphical tools

UNIX/Linux `nrngui`

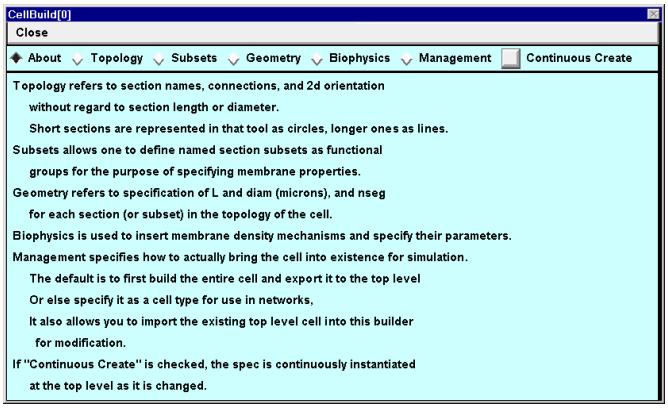
MSWin or OS X

Bring up a CellBuilder



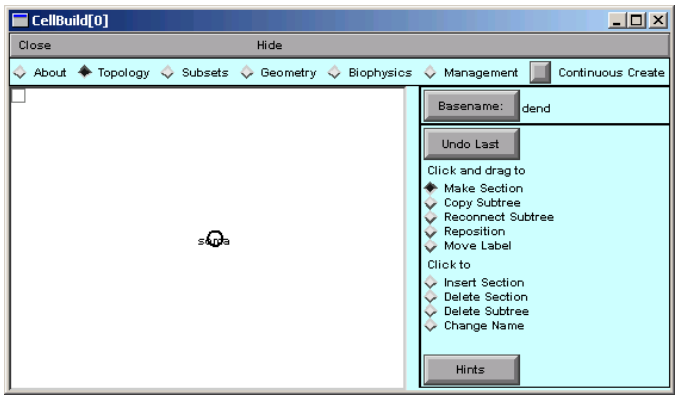
NEURON Main Menu / Build / Cell Builder

The CellBuilder



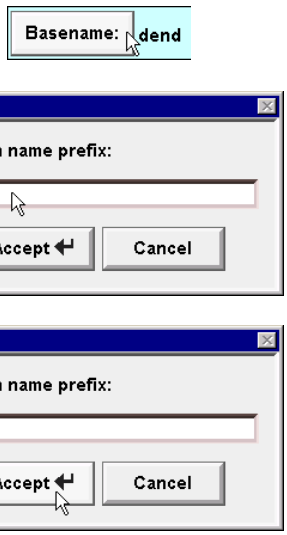
Use buttons from left to right.

Topology



CB starts with a "soma" section.
We want to create new sections.

Specifying the "Basename"



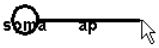
Making a new section

Place cursor near end of existing section

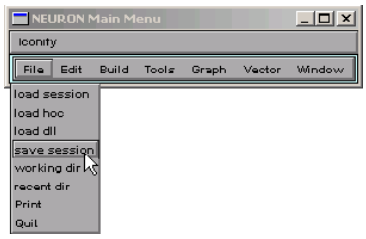
Click to start new section

Drag to desired length

Release mouse button

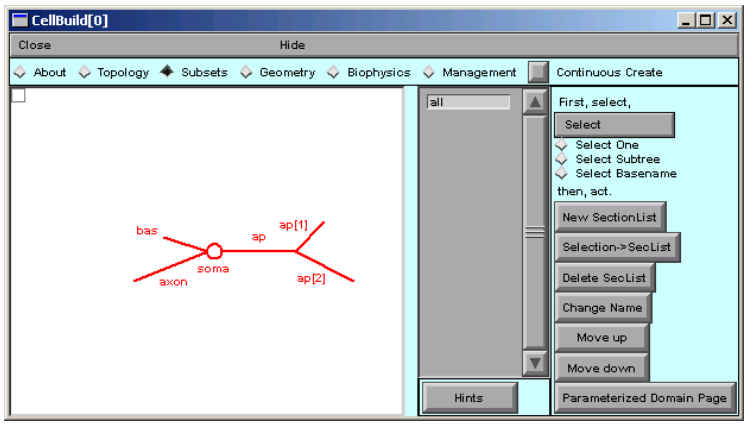


Save your work as you make progress!



NEURON Main Menu / File / save session

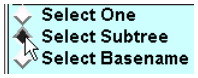
Subsets



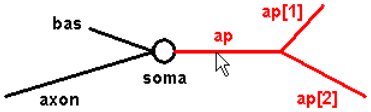
Group sections that have shared properties.
We want to make an "apicals" subset.

Making a new subset

Click "Select Subtree"

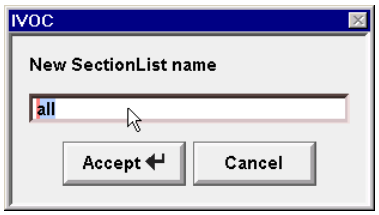


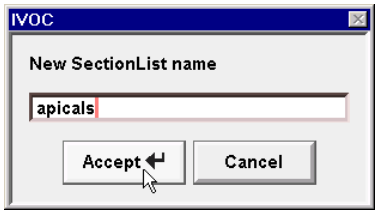
Click root of apical tree . . .



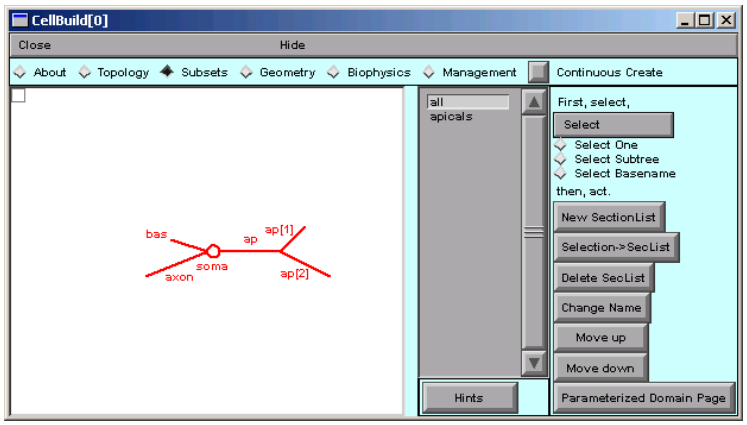
. . . then "New SectionList"

Making a new subset *continued*





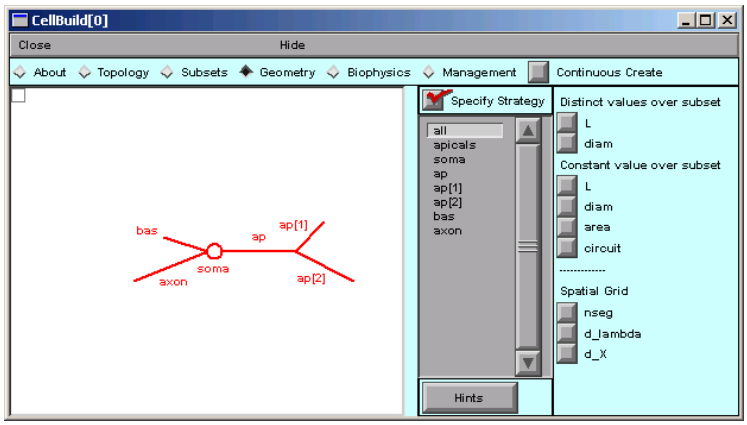
Subsets finished



Note "apicals".

Time to save a new session file.

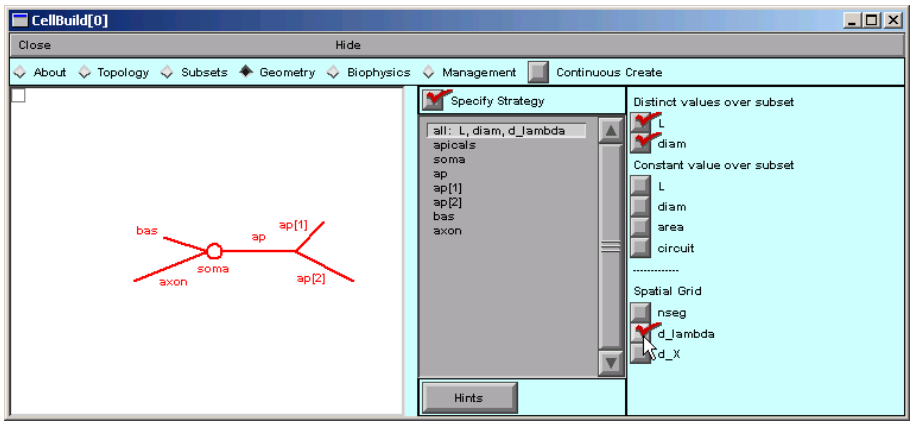
Geometry



"Specify Strategy" is ON.

A good strategy is a concise strategy.

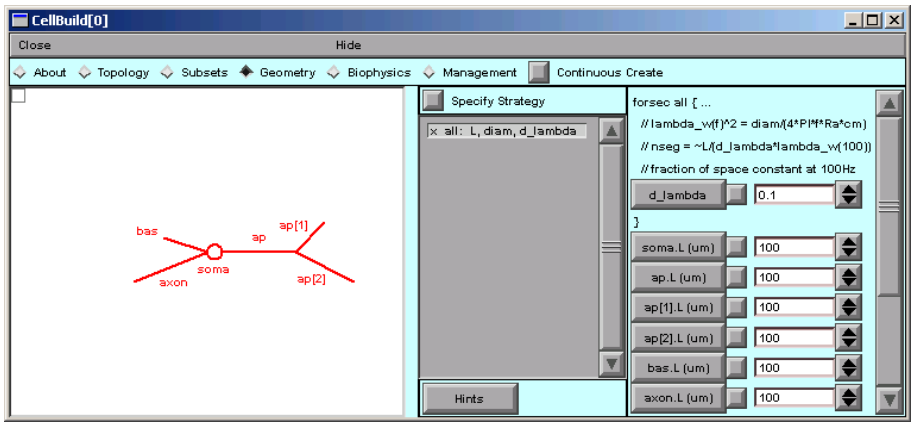
Geometry strategy



Each section has a different L and diam.

Compartmentalize according to $\lambda_{100\ Hz}$ (d_lambda rule).

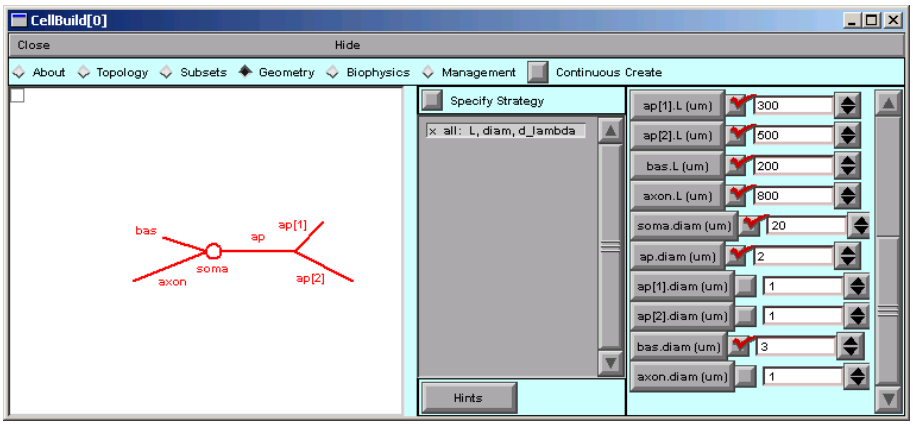
Implementing geometry strategy



When strategy is complete, turn "Specify Strategy" OFF and start assigning values to parameters.

d_lambda = 0.1 at 100 Hz usually gives good spatial accuracy.

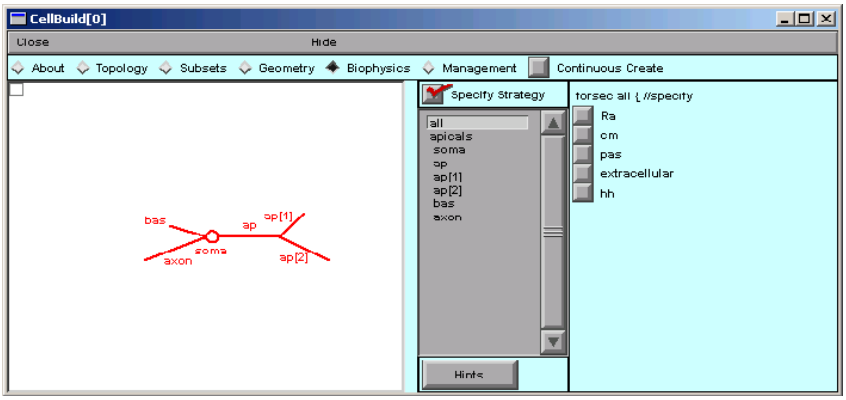
Implementing geometry *continued*



Set L and diam for all sections.

Time to save to a session file!

Biophysics

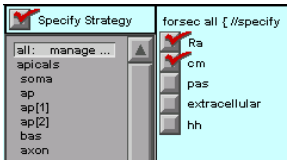


"Specify Strategy" is ON.

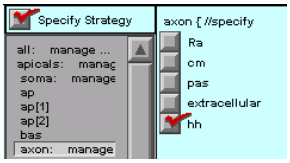
Base the plan on shared properties.

Biophysics strategy

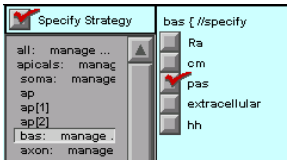
Ra and cm are homogeneous



apicals, soma and axon have hh

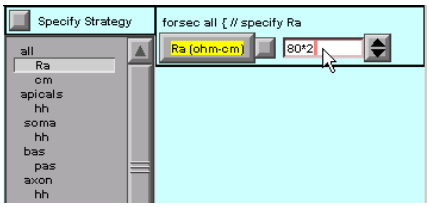


bas has pas

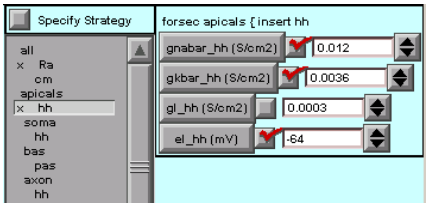


Implementing biophysics strategy

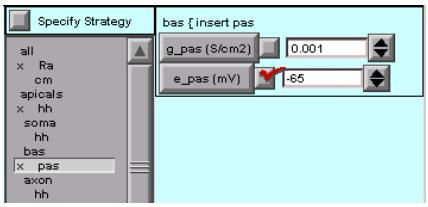
Double Ra



Fix apicals hh params



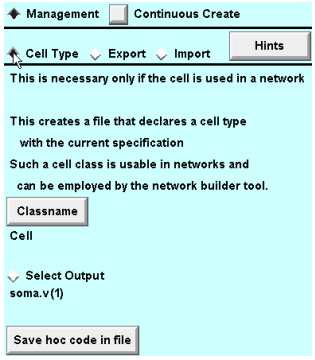
Shift e_pas in bas



Save another session file!!

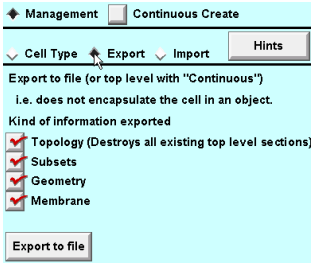
Management

Option 1: save as a Cell Type
for use in a network



Management *continued*

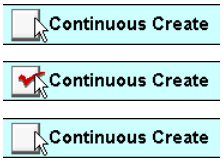
Option 2: save as hoc file



Management *continued*

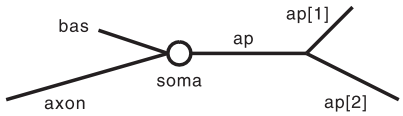
Option 3: export to interpreter

Toggle Continuous Create ON and OFF



or just leave it ON all the time.

Step 2: creating and using an interface for running simulations



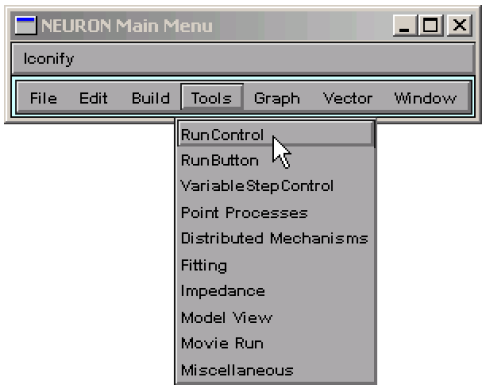
We want to

- attach a stimulating electrode
- evoke an action potential
- show time course of Vm at soma
- show Vm along a path from one end of the cell to the other

We need

- a "Run" button
- graphs to plot results
- a stimulator

Get a "Run" button



NEURON Main Menu / Tools / RunControl

RunControl panel

Init sets time to 0,
Vm to displayed value, and
conductances to steady-state

Init & Run does an Init,
then starts a simulation

Stop interrupts the simulation

Continue til runs until displayed time

Continue for runs for displayed
interval

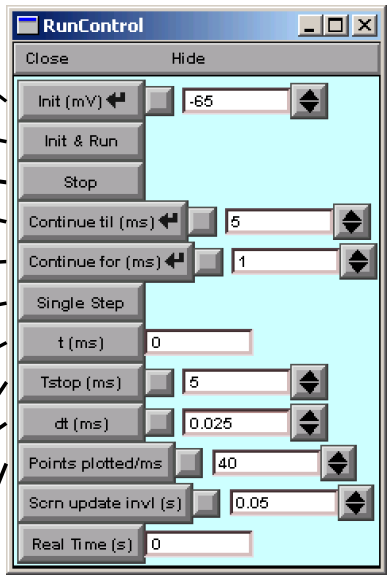
Single step advances by
1/(**Points plotted/ms**)

t numeric field shows model time

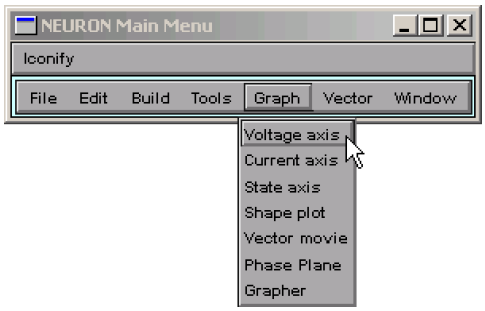
Tstop specifies when simulation ends

dt is integration time step;
must be integer fraction of
1/(**Points plotted/ms**)

Points plotted/ms is plotting interval

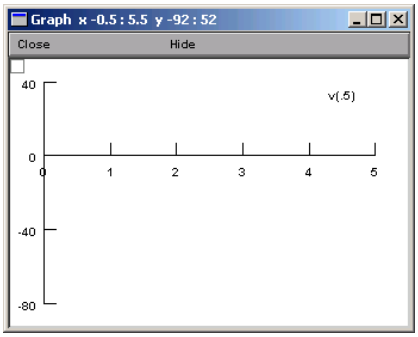


We need to plot Vm(t) at soma



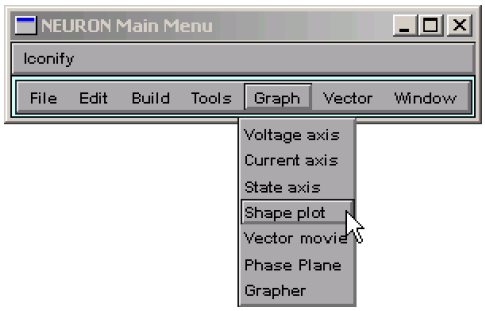
NEURON Main Menu / Graph / Voltage axis

Graph window



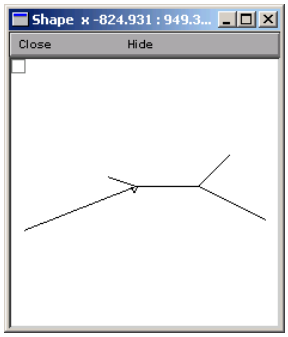
v(.5) is Vm at middle of default section (soma in this example)

We need to plot Vm along a path



NEURON Main Menu / Graph / Shape plot

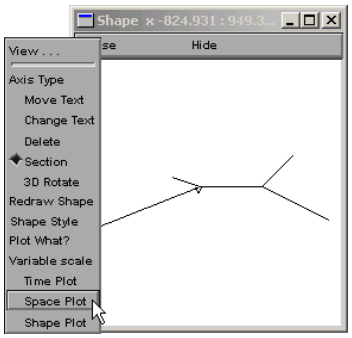
Bringing up a space plot



Use this "shape plot" to create a "space plot".

Click on its "menu box" . . .

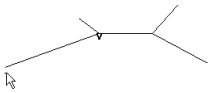
Bringing up a space plot *continued*



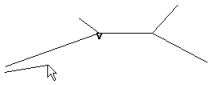
. . . and scroll down to "Space Plot".

Bringing up a space plot *continued*

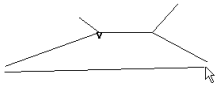
Click just left of the shape



Hold button down while dragging from left . . .



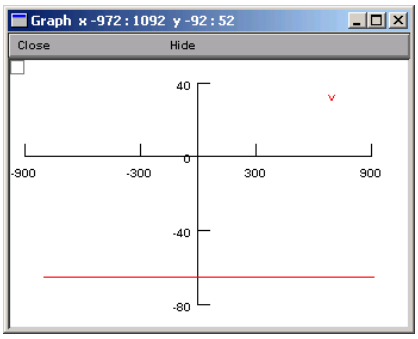
. . . to right . . .



. . . then release button.

This pops up a . . .

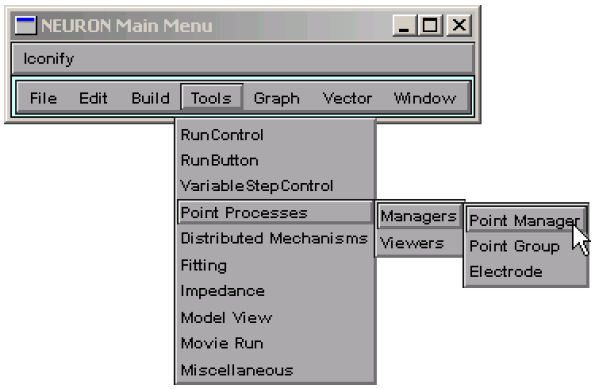
Space plot



A plot of Vm vs. distance along a path.

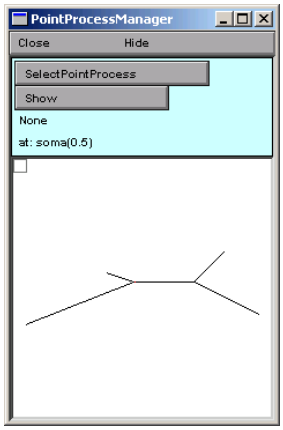
Better save a session file.

We need a stimulator



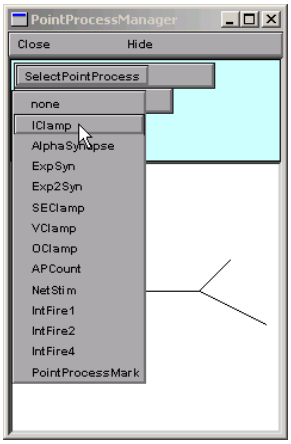
NEURON Main Menu / Tools / Point Processes / Managers / Point Manager

PointProcessManager window



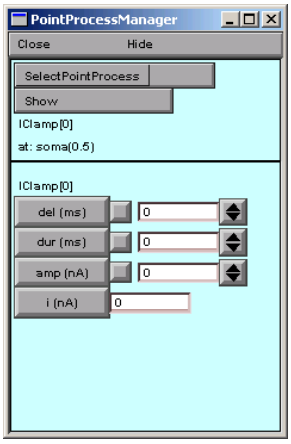
To make this an IClamp . . .

Creating an IClamp



. . . click on SelectPointProcess
and scroll down to IClamp.

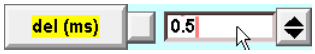
IClamp parameter panel



Next: set parameter values.

Entering values into numeric fields

Direct entry

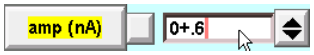


Note yellow highlight on button

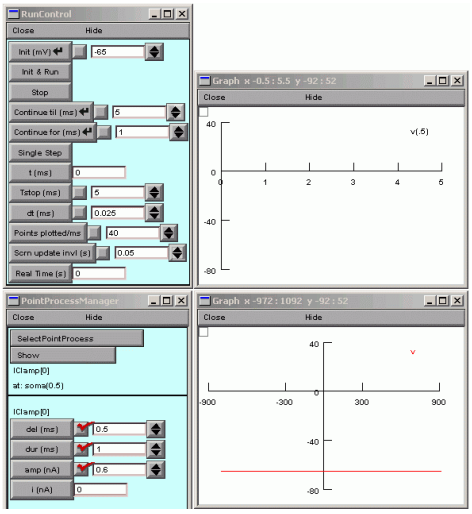
Spinner

Red check means value has been changed from default

Mathematical expression

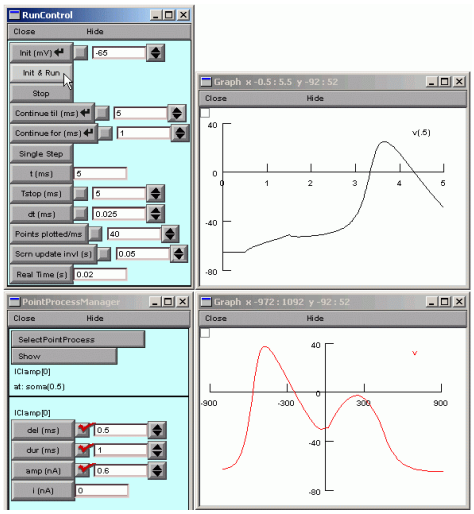


Our user interface

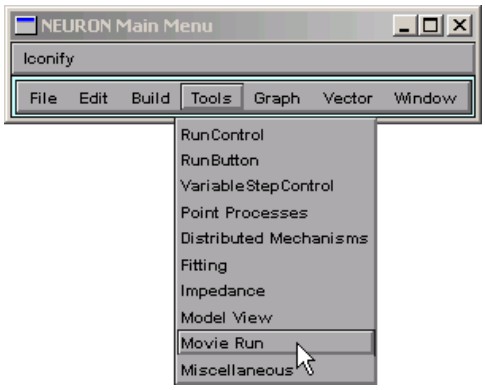


Time to save to a new session file!

It works!

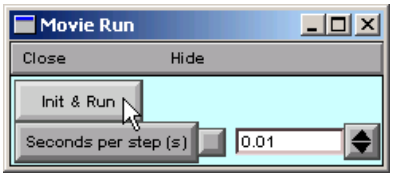


How to get nice space plot "movies"



NEURON Main Menu / Tools / Movie Run

Space plot "movies" *continued*



Movie Run / Init & Run

What if channel density in the apical tree varies systematically with position, e.g. distance from the soma?

See "Specifying parameterized variation of biophysical properties" in the CellBuilder tutorial at http://neuron.yale.edu/neuron/docs

The Linear Circuit Builder

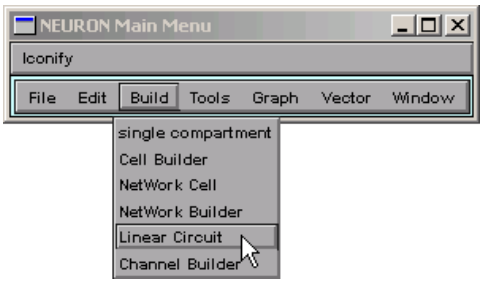
For building models that have linear circuit elements and may also involve neurons

Circuit elements include ground, current & voltage source, R, C, op amp

Potential applications include

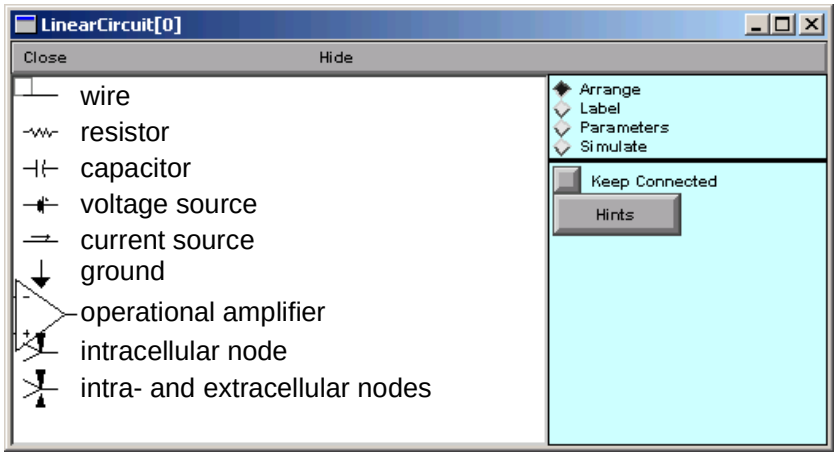
- effects and compensation of electrode R & C
- two-electrode voltage clamp
- ohmic and nonlinear gap junctions

1. Bring up a Linear Circuit Builder



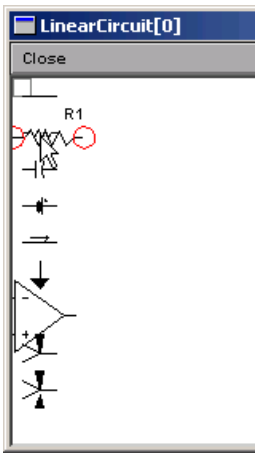
NEURON Main Menu / Build / Linear Circuit

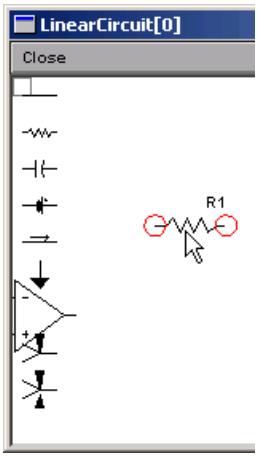
The Linear Circuit Builder



Arrange: spawn components

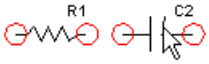
Click on palette and drag onto canvas



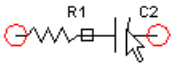


Arrange: connect components

Click and drag to overlap red circles



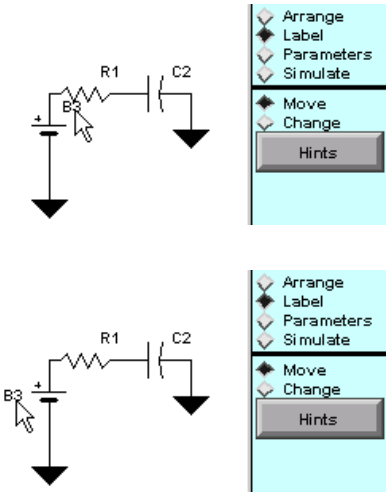
Black square is "solder joint"



Pull apart to break connection

Label: move labels

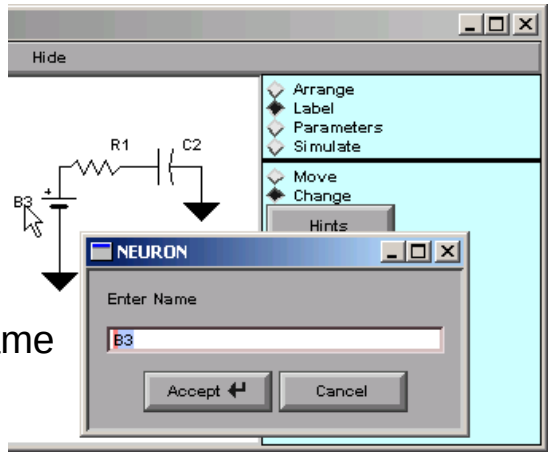
Click and drag to new location



Label: change labels 1

Click on a label . . .

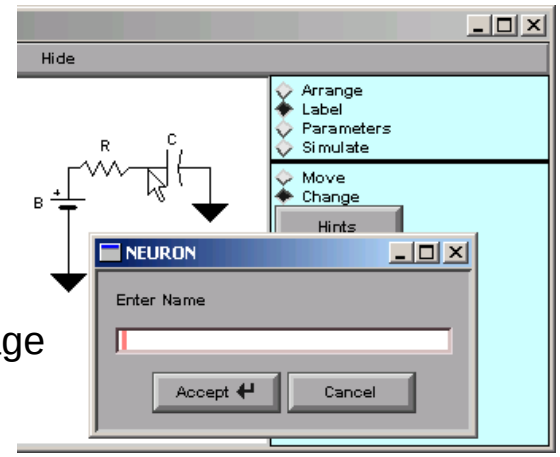
. . . to change its name



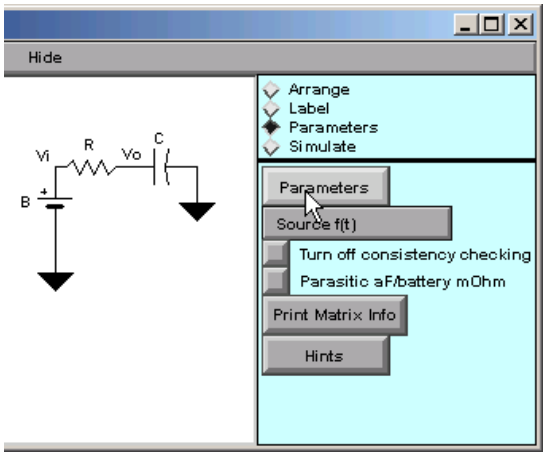
Label: change labels 2

Click on a node . . .

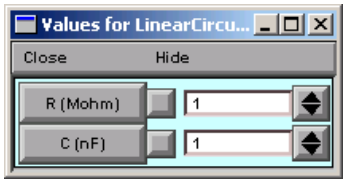
. . . to label a voltage



Parameters: non-source elements

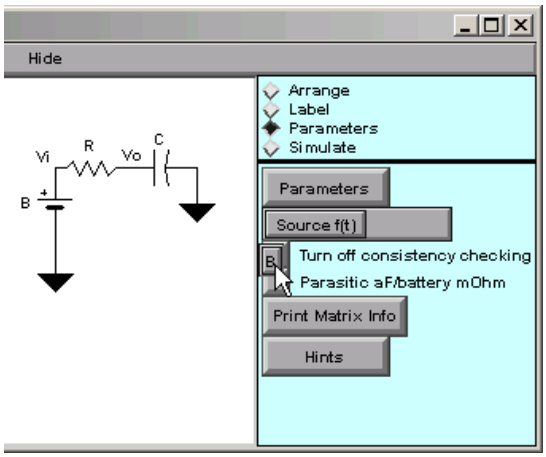


Click on "Parameters"

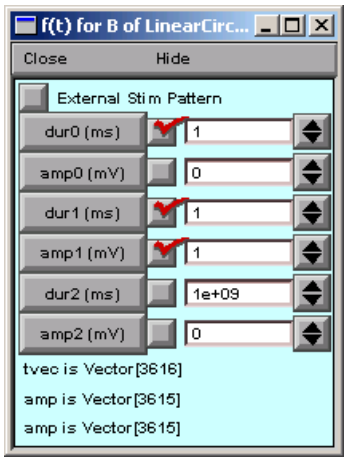


Parameters: signal sources

Source f(t) / B



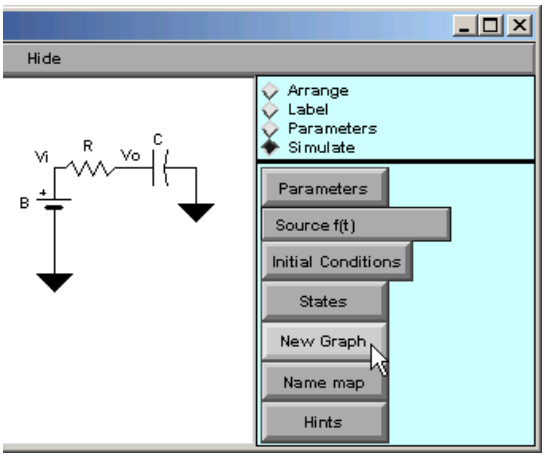
Parameters: signal sources *continued*



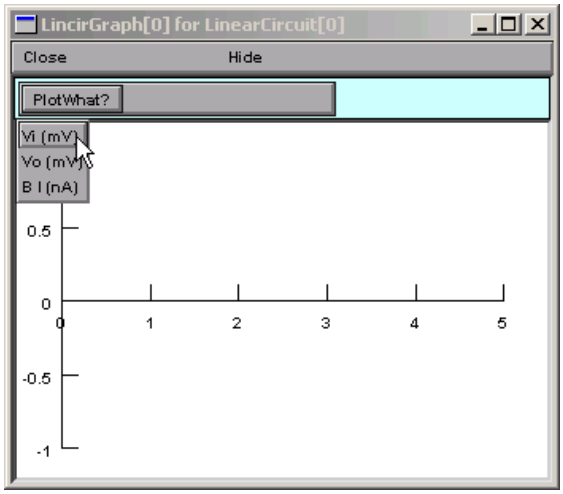
Configured

Simulate: creating a graph

New Graph

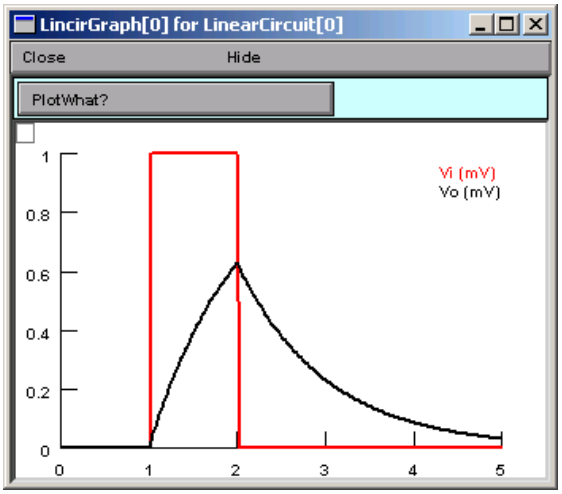


Simulate: specifying what to plot



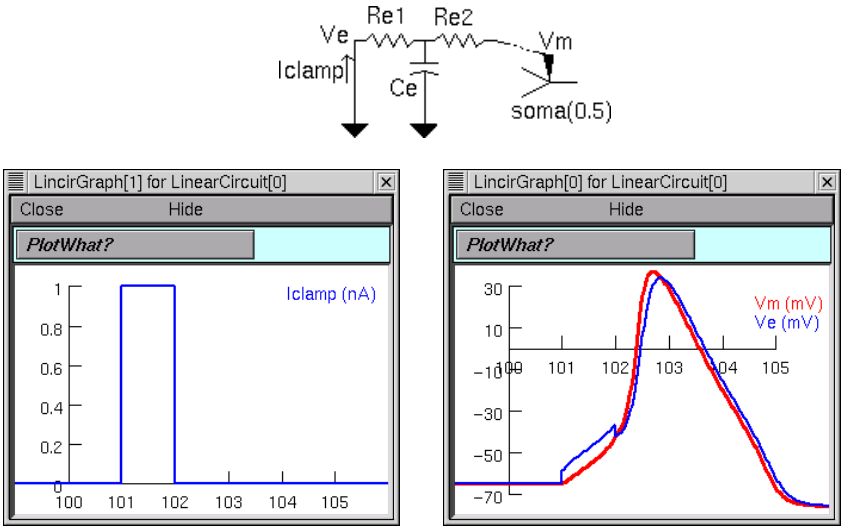
PlotWhat? / *variable_label*

Simulate: simulation results

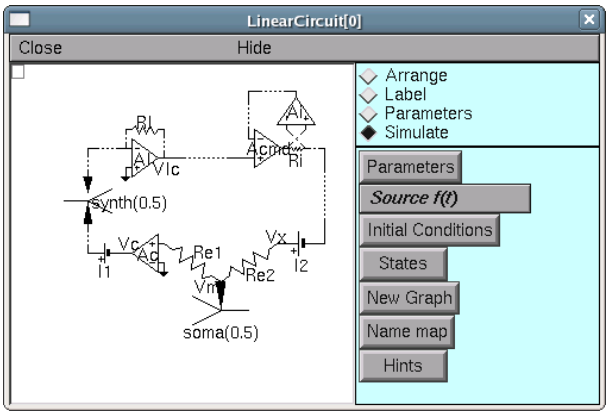


After minor cosmetic changes

Patch clamp with electrode R and C



NEURON demo: dynamic clamp



NMODL

NEURON Model Description Language

Add new membrane mechanisms to NEURON

Density mechanisms

- Distributed Channels
- Ion accumulation

Point Processes

- Electrodes
- Synapses

Described by

- Differential equations
- Kinetic schemes
- Algebraic equations

Benefits

- Specification only −− independent of solution method.
- Efficient −− translated into C.
- Compact
  - One NMODL statement −> many C statements.
  - Interface code automatically generated.
- Consistent ion current/concentration interactions.
- Consistent Units

NMODL general block structure

What the model looks like from outside

```
NEURON {
    SUFFIX kchan
    USEION k READ ek WRITE ik
    RANGE gbar, ...
}
```

What names are manipulated by this model

```
UNITS { (mV) = (millivolt) ... }
PARAMETER { gbar = .036 (mho/cm2) <0, 1e9>... }
STATE { n ... }
ASSIGNED { ik (mA/cm2) ... }
```

Initial default values for states

```
INITIAL {
    rates(v)
    n = ninf
}
```

Calculate currents (if any) as function of v, t, states

(and specify how states are to be integrated)

```
BREAKPOINT {
    SOLVE deriv METHOD cnexp
    ik = gbar * n^4 * (v − ek)
}
```

State equations

```
DERIVATIVE deriv {
    rates(v)
    n’ = (ninf − n)/ntau
}
```

Functions and procedures

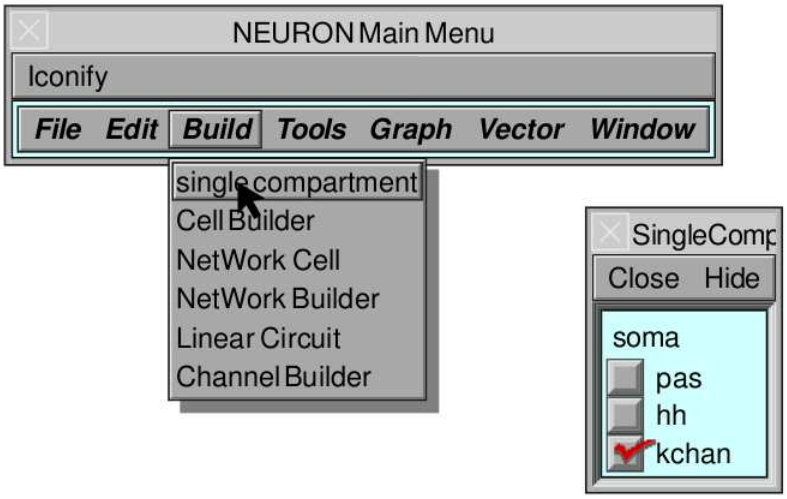
```
PROCEDURE rates(v(mV)) {
    ...
}
```


UNIX

```
nrnivmodl
nrngui
```

MSWIN

Select NEURON Main Menu / Build / single compartment



Density mechanism

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

PARAMETER {
    g = .001 (mho/cm2) <0, 1e9>
    e = −65 (millivolt)
}

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

BREAKPOINT {
    i = g*(v − e)
}
```

Point Process

```
NEURON {
    POINT_PROCESS Shunt
    NONSPECIFIC_CURRENT i
    RANGE i, e, r
}

PARAMETER {
    r = 1 (gigaohm) <1e−9,1e9>
    e = 0 (millivolt)
}

ASSIGNED {
    i (nanoamp)
    v (millivolt)
}

BREAKPOINT {
    i = (.001)*(v − e)/r
}
```

Density mechanism

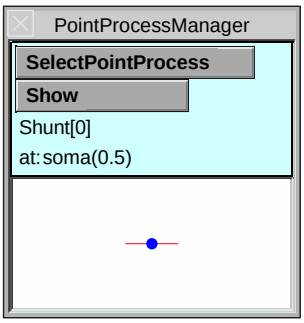
Point Process

NMODL

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

```
NEURON {
    POINT_PROCESS Shunt
    NONSPECIFIC_CURRENT i
    RANGE i, e, r
}
```

GUI



Interpreter

```
soma {
    insert leak
    g_leak = .0001
}
print soma.i_leak(.5)
```

```
objref s
soma s = new Shunt(.5)
s.r = 2
```

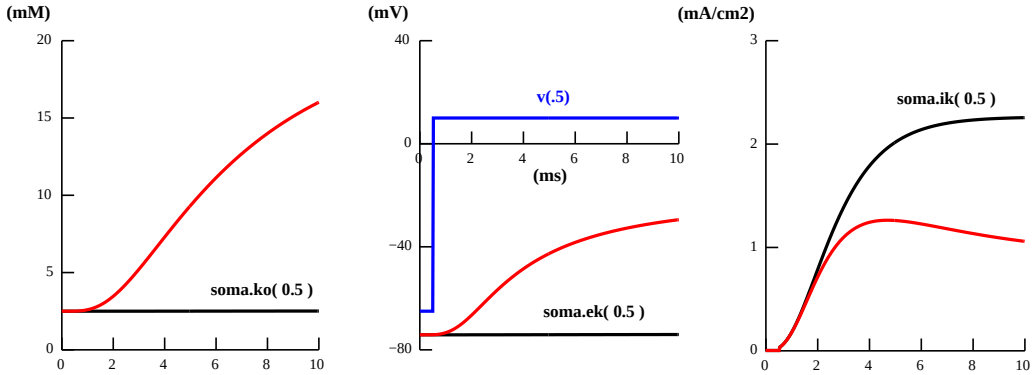
Ion Channel

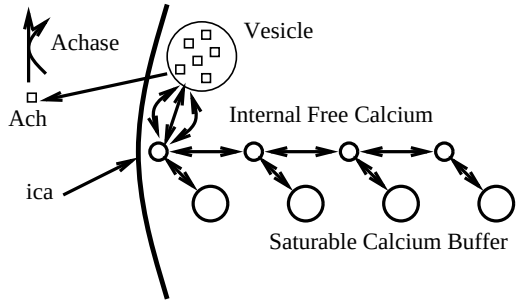
```
NEURON {
  USEION k READ ek WRITE ik
}
BREAKPOINT {
  SOLVE states METHOD cnexp
  ik = gbar*n*n*n*n*(v - ek)
}
DERIVATIVE states {
  rate(v*1(/mV))
  n' = (inf - n)/tau
}
```

Ion Accumulation

```
NEURON {
  USEION k READ ik WRITE ko
}
BREAKPOINT {
  SOLVE state METHOD cnexp
}

DERIVATIVE state {
  ko' = ik/fhspace/F*(1e8)
        + k*(kbath - ko)
}
```






```
STATE {
   Vesicle Ach Achase Ach2ase X Buffer[N] CaBuffer[N] Ca[N]
}
KINETIC calcium_evoked_release  {
   : release
 ~ Vesicle + 3Ca[0] <-> Ach  (Agen, Arev)
 ~ Ach + Achase <-> Ach2ase  (Aase2, 0) : idiom for enzyme reaction
 ~ Ach2ase <-> X + Achase    (Aase2, 0) : requires two reactions
   : Buffering
   FROM i = 0 TO N-1 {
     ~ Ca[i] + Buffer[i] <-> CaBuffer[i]  (kCaBuffer, kmCaBuffer)
   }
   : Diffusion
   FROM i = 1 TO N-1 {
     ~ Ca[i-1] <-> Ca[i]     (Dca*a[i-1], Dca*b[i])
   }
   : inward flux
 ~ Ca[0] <<      (ica)
}
```


UNITS Checking

```
NEURON { POINT_PROCESS Shunt ... }
PARAMETER {
    e = 0 (millivolt)
    r = 1 (gigaohm) <1e−9,1e9>
}
ASSIGNED {
    i (nanoamp)
    v (millivolt)
}
BREAKPOINT {
    i = (v − e)/r
}
```

Units are incorrect in the "i = ..." current assignment.

```
BREAKPOINT {
    i = (v − e)/r
}
```

The output from
modlunit shunt
is:

```
Checking units of shunt.mod
The previous primary expression with units: 1−12 coul/sec
is missing a conversion factor and should read:
  (0.001)*()
at line 14 in file shunt.mod
       i = (v − e)/r<>
```

To fix the problem replace the line with:

```
i = (.001)*(v − e)/r
```

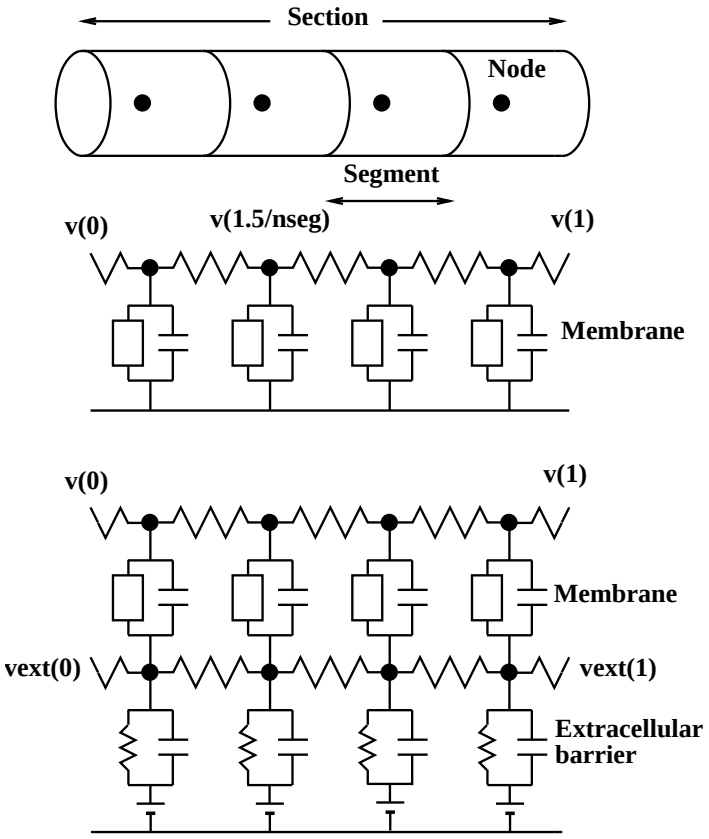
What conversion factor will make the following consistent?

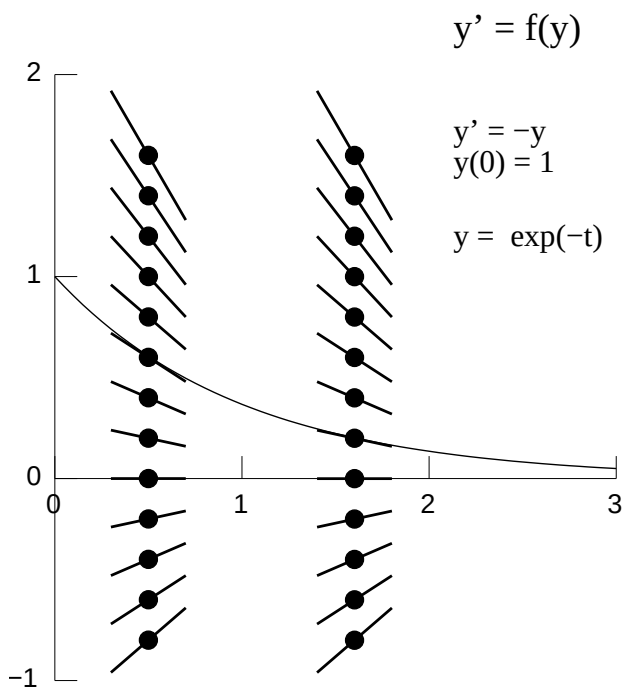
nai' = ina / FARADAY * (c/radius)
(uM/ms) (mA/cm2) / (coulomb/mole) / (um)

Compartmental Modeling

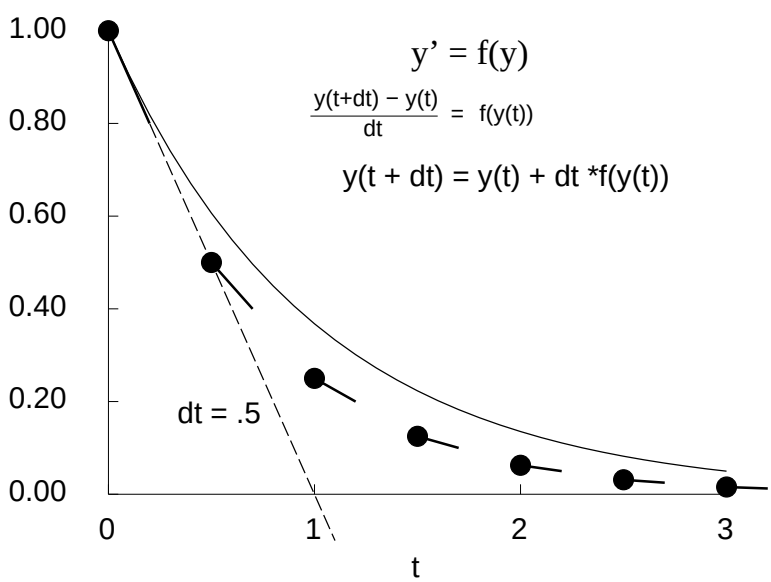
Not much mathematics required.

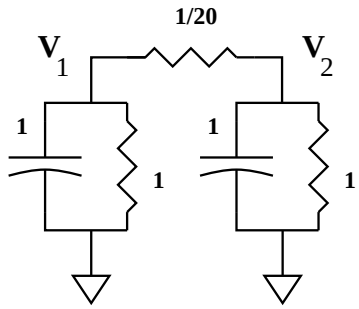
Good judgment essential!

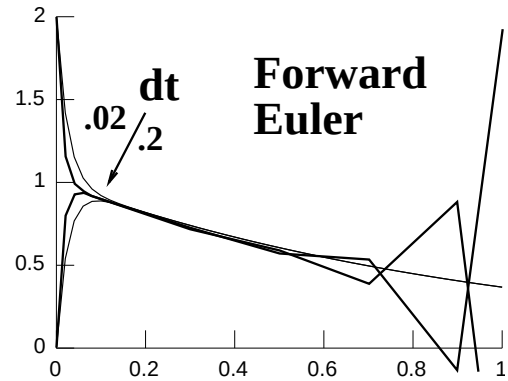




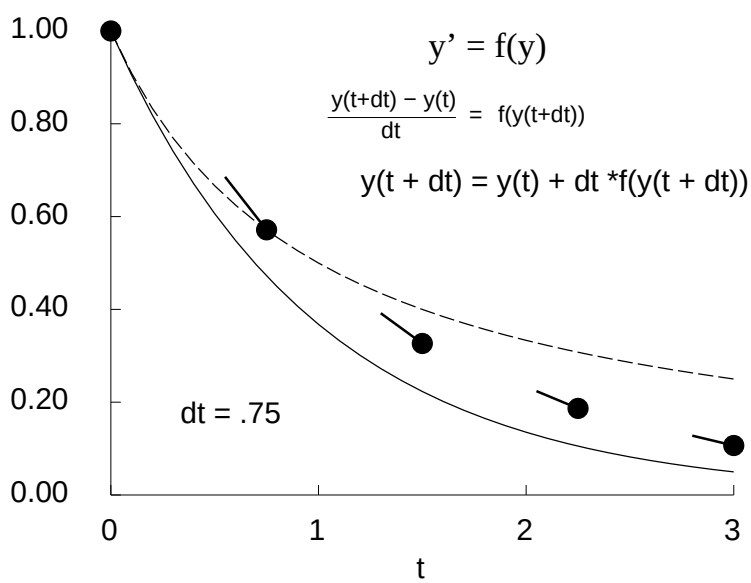
Forward Euler

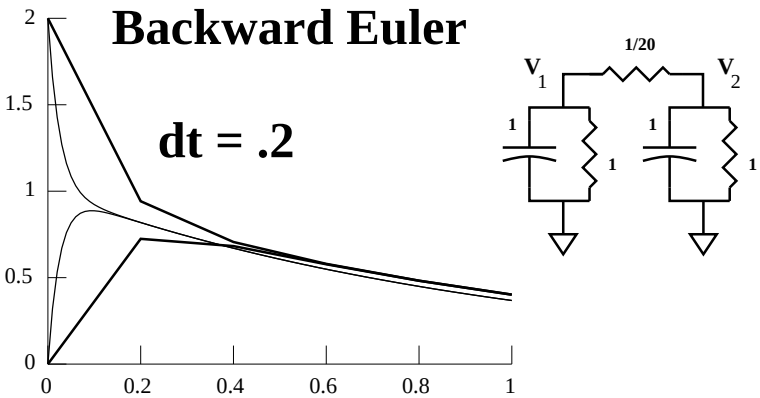


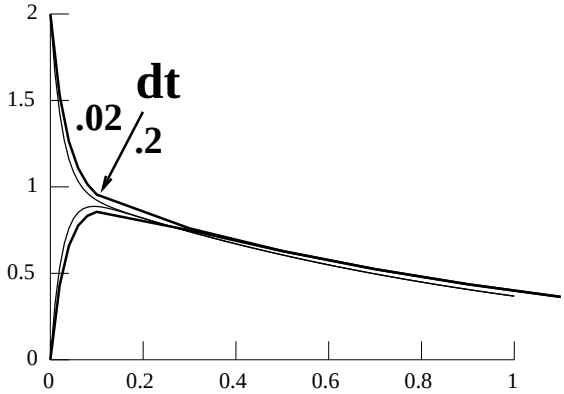


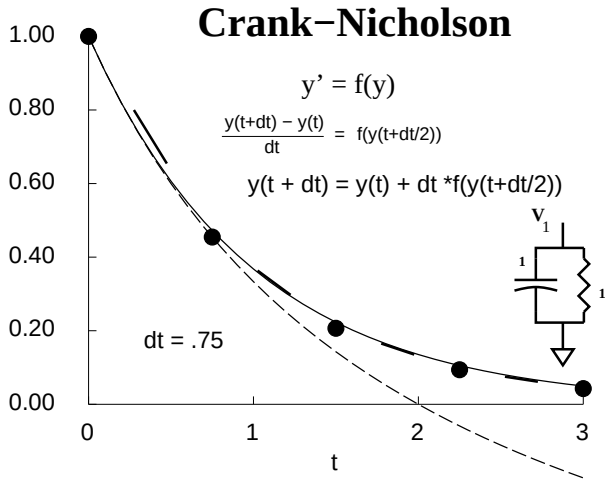


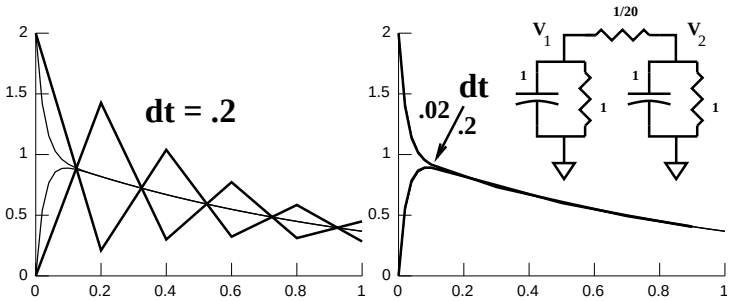
Backward Euler

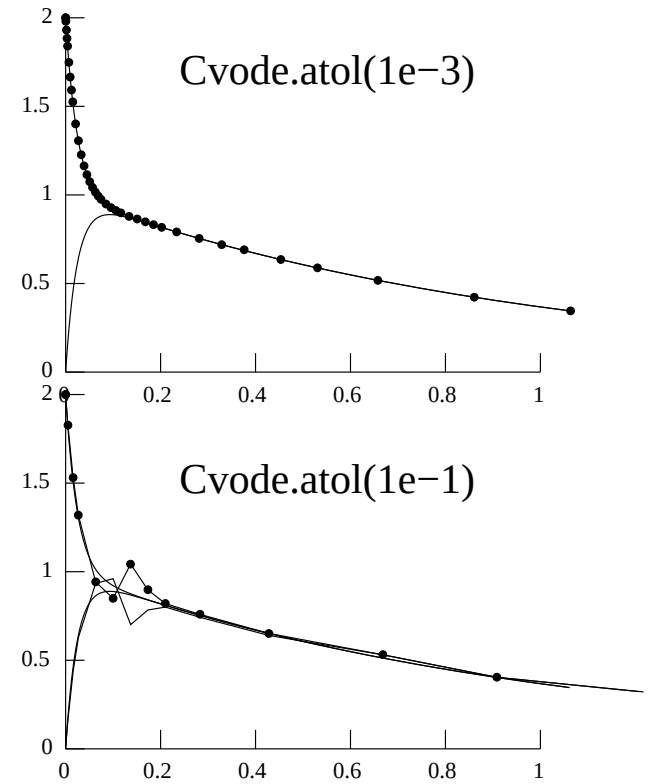


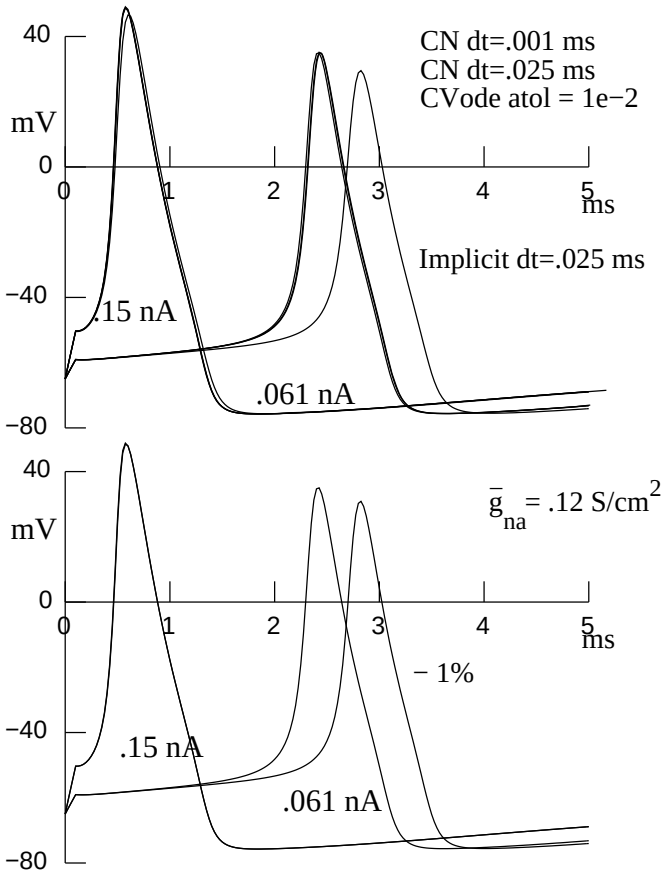












Networks: spike-triggered synaptic transmission, events, and artificial spiking cells

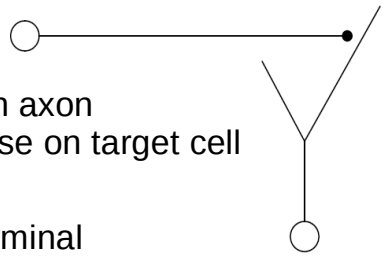
1. Define the types of cells
2. Create each cell in the network
3. Connect the cells

Communication between cells

Gap junctions

Synaptic transmission
- graded
- spike-triggered

Spike-triggered synaptic transmission

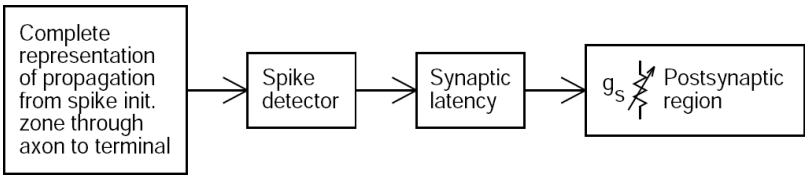


Physical system:
- Presynaptic neuron with axon that projects to synapse on target cell

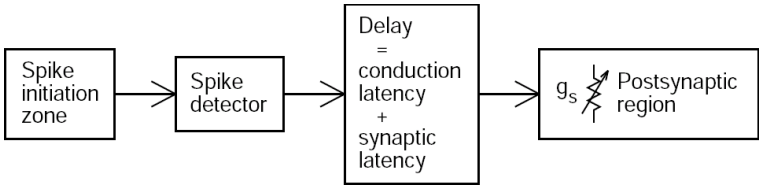
Conceptual model:
- Spike in presynaptic terminal triggers transmitter release; presynaptic details unimportant
- Postsynaptic effect described by DE or kinetic scheme that is perturbed by occurrence of a presynaptic spike

Spike-triggered transmission: computational implementation

Basic idea



More efficient: "virtual spike propagation"



The NetCon class

hoc usage

```
netcon = new NetCon(source, target)
presection netcon = new NetCon(&v(x), \
    target, threshold, delay, weight)
```

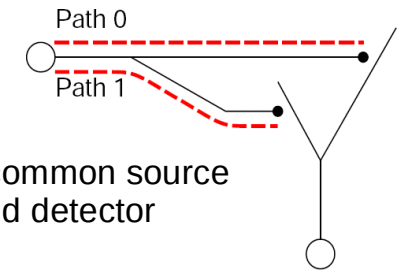
Defaults

```
threshold = 10
delay = 1 // must be >= 0
weight = 0
```

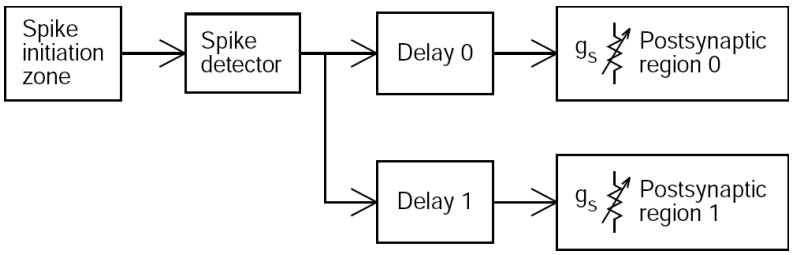
NMODL specification of synaptic mechanism

```
NET_RECEIVE(weight(microsiemens)) {
    . . .
}
```

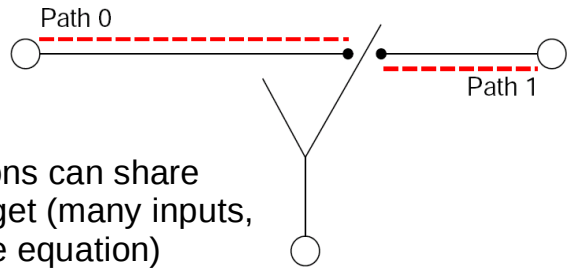
Efficient divergence



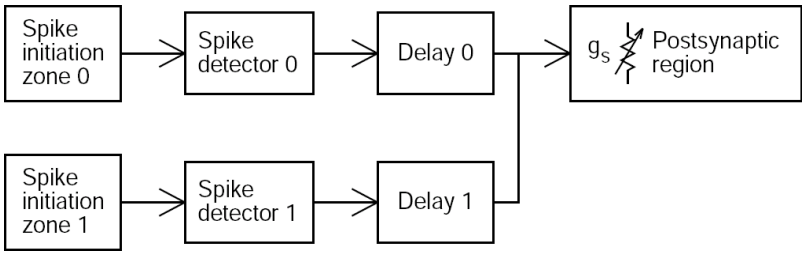
Multiple NetCons with a common source share a single threshold detector



Efficient convergence



Multiple NetCons can share a single target (many inputs, but only one equation)

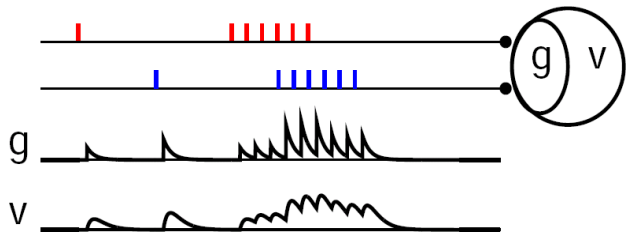


Example: g_s with fast rise and exponential decay

```
NEURON {
  POINT_PROCESS ExpSyn
  RANGE tau, e, i
  NONSPECIFIC_CURRENT i
}
  . . . declarations . . .
INITIAL { g = 0 }
BREAKPOINT {
  SOLVE state METHOD cnexp
  i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }
```

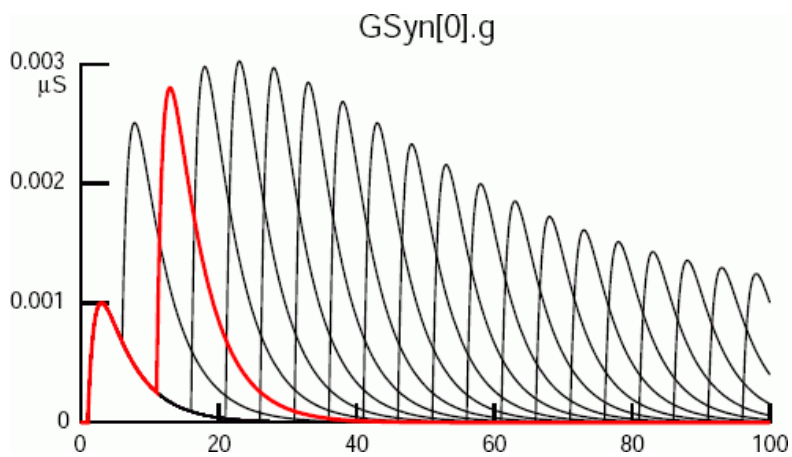
g_s with fast rise and exponential decay

continued



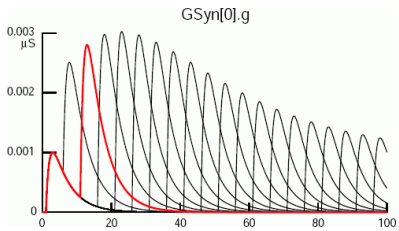

```
BREAKPOINT {
  SOLVE state METHOD cnexp
  i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }
```

Example: use-dependent synaptic plasticity



Use-dependent synaptic plasticity *continued*

```
BREAKPOINT {
  SOLVE state METHOD cnexp
  g = B - A
  i = g*(v-e)
}
DERIVATIVE state {
  A' = -A/tau1
  B' = -B/tau2
}
```




```
NET_RECEIVE(weight (uS), w, G1, G2, t0 (ms)) {
  INITIAL {w=0 G1=0 G2=0 t0=t}
  G1 = G1*exp(-(t-t0)/Gtau1)
  G2 = G2*exp(-(t-t0)/Gtau2)
  G1 = G1 + Ginc*Gfactor
  G2 = G2 + Ginc*Gfactor
  t0 = t
  w = weight*(1 + G2 - G1)
  g = g + w
  A = A + w*factor
  B = B + w*factor
}
```

Artificial spiking cells

"Integrate and fire" cells

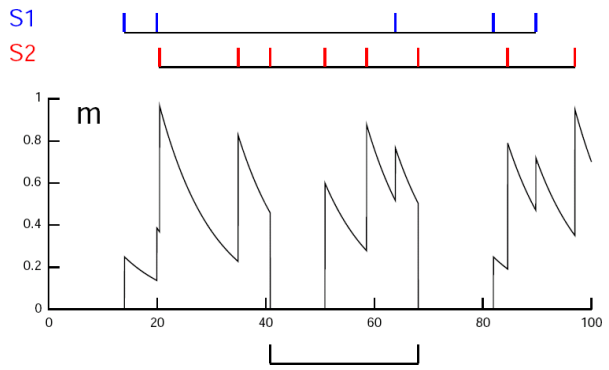
Prerequisite: all state variables must be analytically computable from a new initial condition

Orders of magnitude faster than numerical integration

Event-driven simulation run time is
- *proportional* to # of received events
- *independent* of # of cells, # of connections, and problem time

Hybrid networks

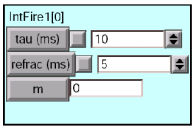
Example: leaky integrate and fire model

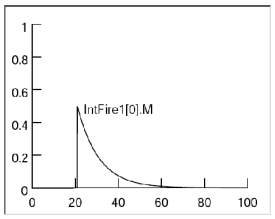


Leaky integrate and fire model *continued*

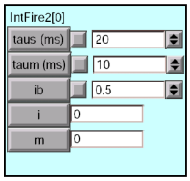
```
NEURON {
  ARTIFICIAL_CELL IntFire
  RANGE tau, m
}
  . . . declarations . . .
INITIAL { m = 0   t0 = t }
NET_RECEIVE (w) {
  m = m*exp(-(t-t0)/tau)
  t0 = t
  m = m + w
  if (m > 1) {
    net_event(t)
    m = 0
  }
}
```

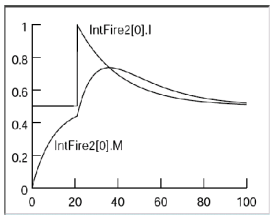

IntFire1



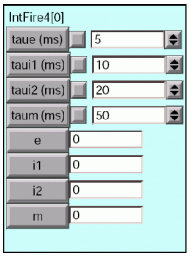


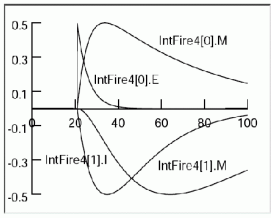
IntFire2





IntFire4





Defining the types of cells

Artificial spiking cells

- ARTIFICIAL_CELL with a NET_RECEIVE block that calls net_event
- NetStim, IntFire1, IntFire2, IntFire4

Biophysical model cells

- "Real" model cells
- Sections and density mechanisms
- Synapses are POINT_PROCESSes that affect membrane current and have a NET_RECEIVE block, e.g. ExpSyn, Exp2Syn

Defining types of biophysical model cells

Encapsulate in a class

```
begintemplate Cell
  public soma, E, I
  create soma
  objref E, I
  proc init() {
    soma {
      insert hh
      E = new ExpSyn(0.5)
      I = new Exp2Syn(0.5)
      I.e = -80
    }
  }
endtemplate Cell

objref bag_of_cells
bag_of_cells = new List()
for i = 1,1000 bag_of_cells.append(new Cell())
```

Connecting cells

Which setup strategy is more efficient?

Iterate over sources

```
for each cell {
  connect this cell to its targets
}
```

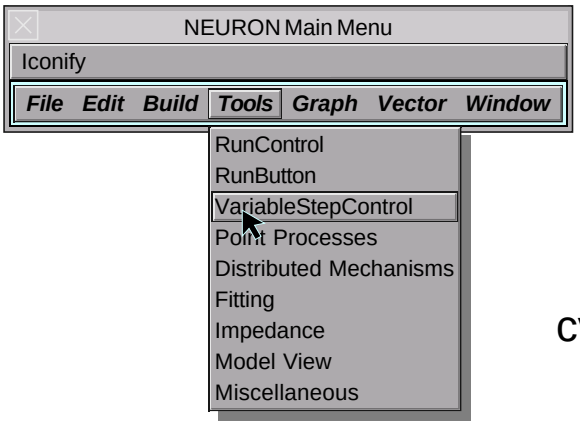
or iterate over targets?

```
for each cell {
  connect sources to this cell
}
```

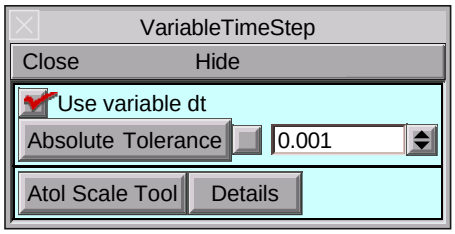
Connecting cells

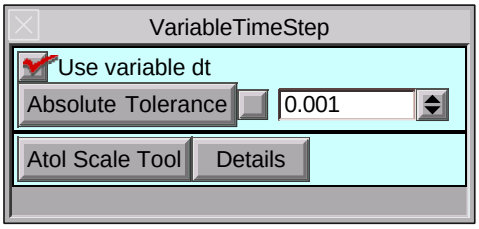
For a net distributed over multiple CPUs,
it is most efficient to iterate over targets first.

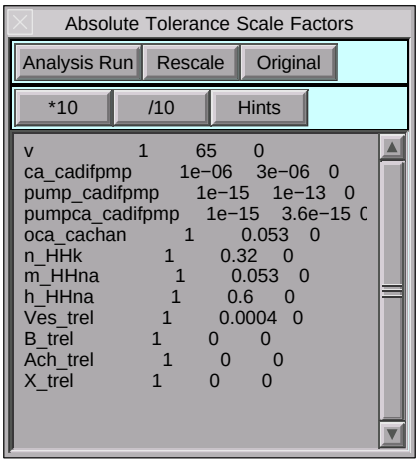
```
for each cell {
  connect sources to this cell
}
```

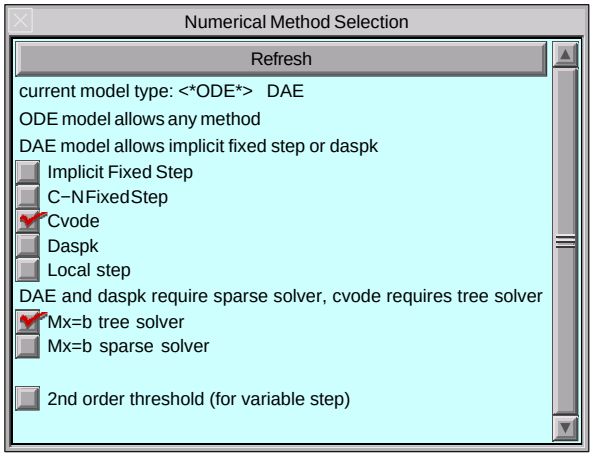


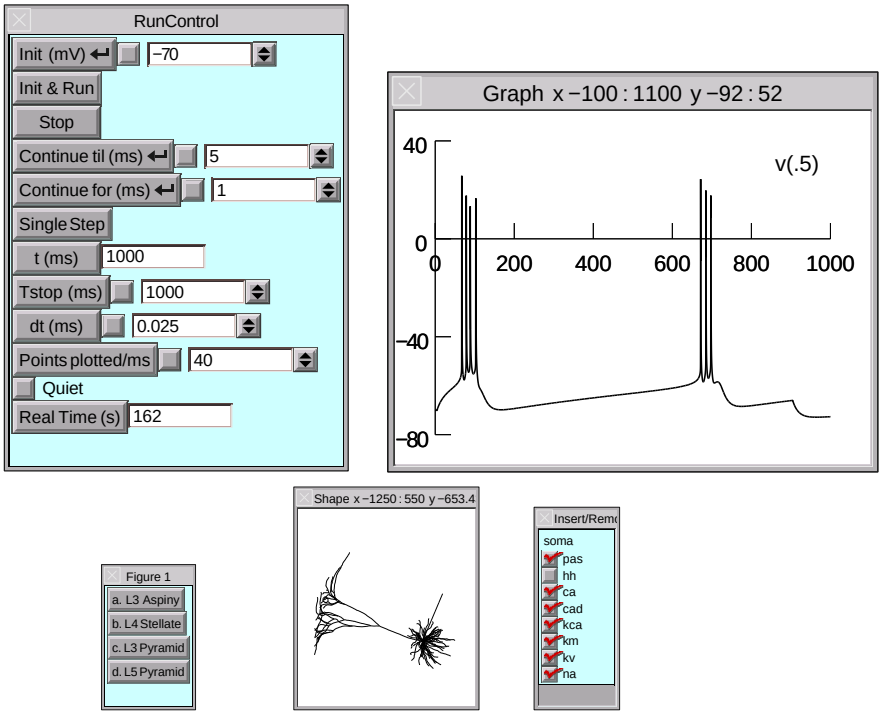
cvode_active(1)

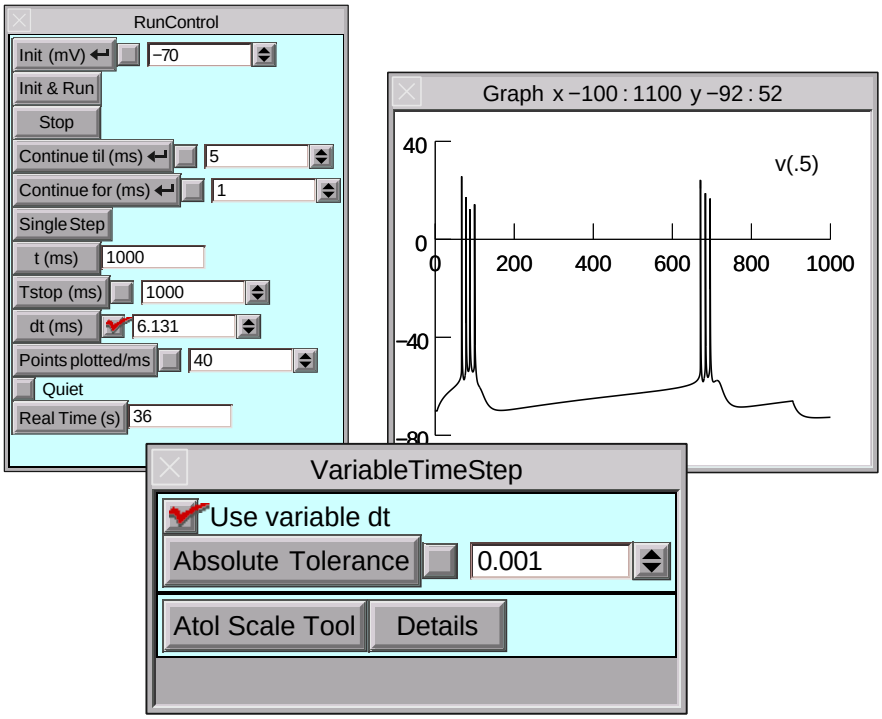


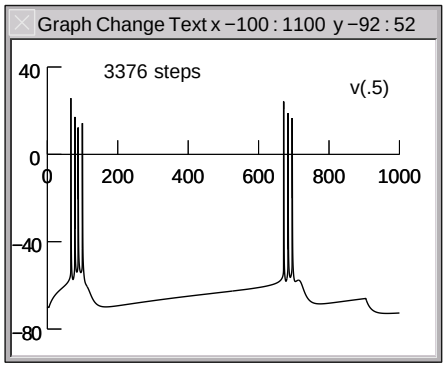


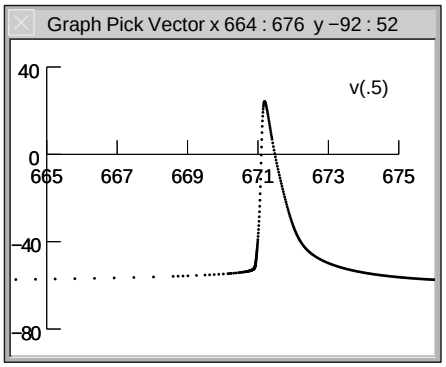


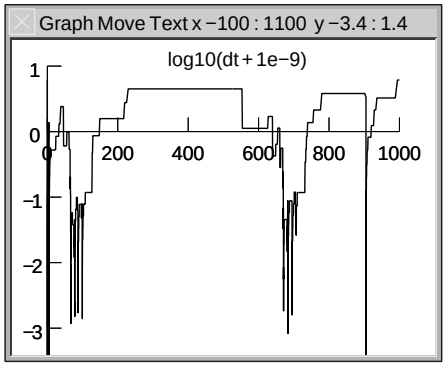


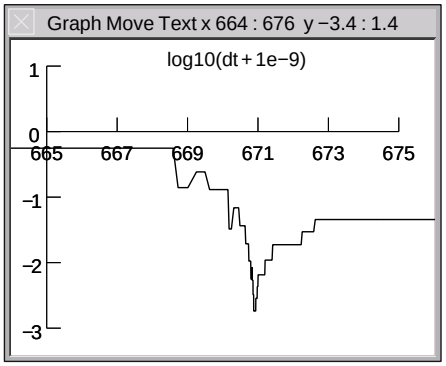


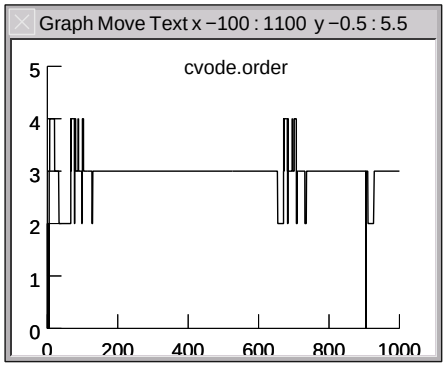


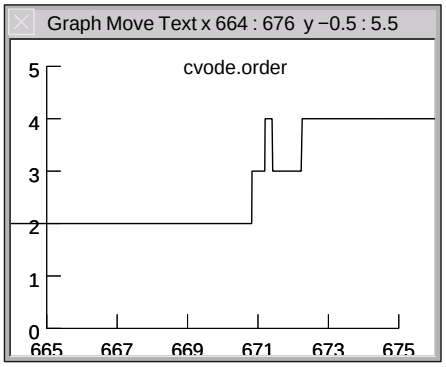


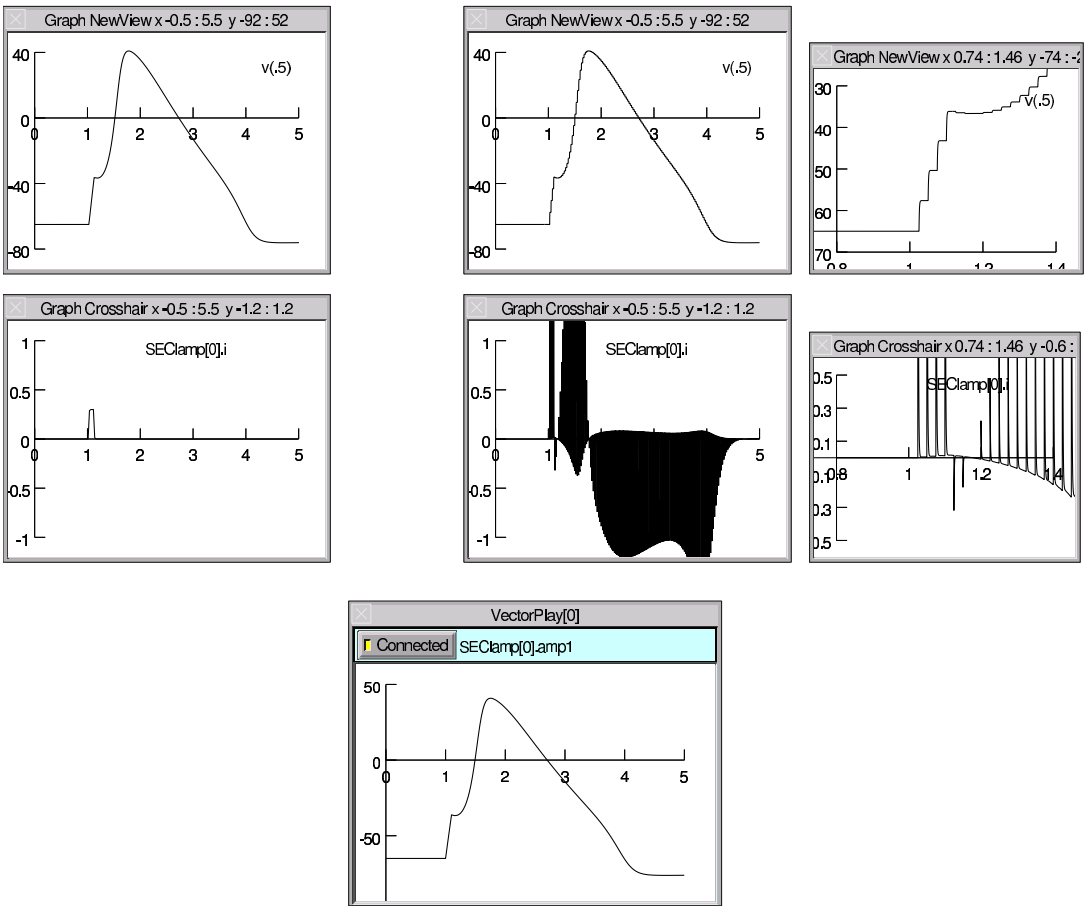


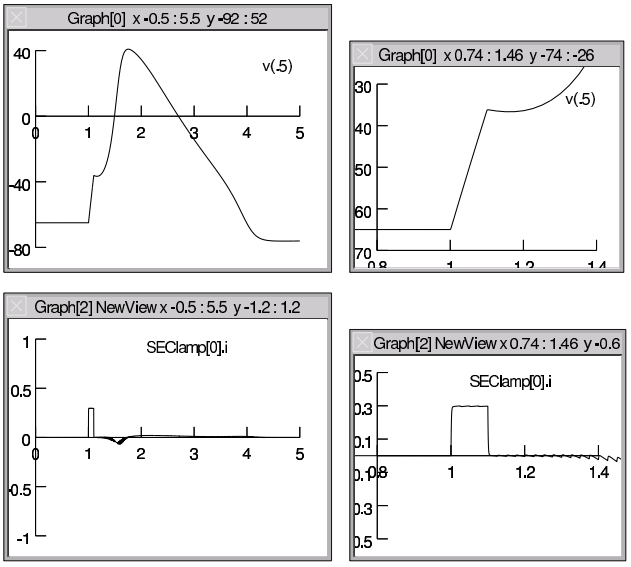













```
soma vvec.play(&SEClamp[0].amp1, tvec, 1)
```


Parallel Computation

"Faster" is the only reason

But...

greater programming complexity

new kinds of bugs

...and not much help for fixing them.

Can the day or week of user effort be recovered?

16384 core EPFL IBM BlueGene/P

1 hour at 850MHz

6 months at 3GHz

Parallel Computation

A simulation run takes about a second

want to do 1000's of them,

varying a dozen or so parameters.

- Screensaver Calin−Jageman and Katz, 2006
- Bulletin−board (Linda)

A simulation run takes hours.

want to spread the problem over several machines.

Parallel Computation

A simulation run takes hours.

want to spread the problem over several machines.

Network

Subnets on different machines

Cells communicate by:

logical spike events with significant axonal, synaptic delay.

postsynaptic conductance depends continuously on presynaptic voltage.

gap junctions

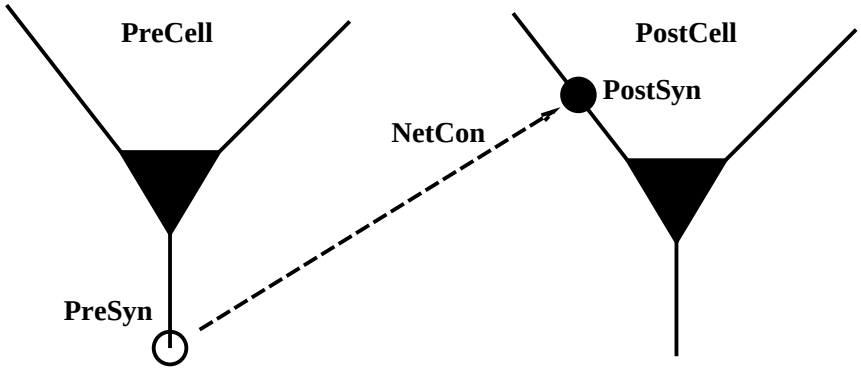
Parallel Computation

A simulation run takes hours.

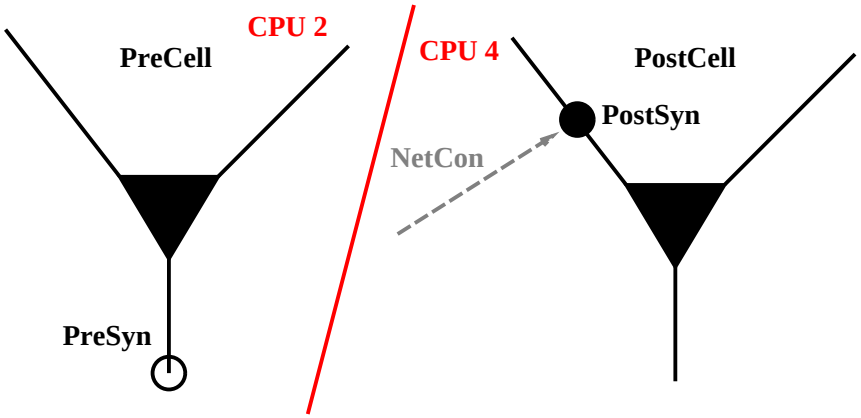
want to spread the problem over several machines.

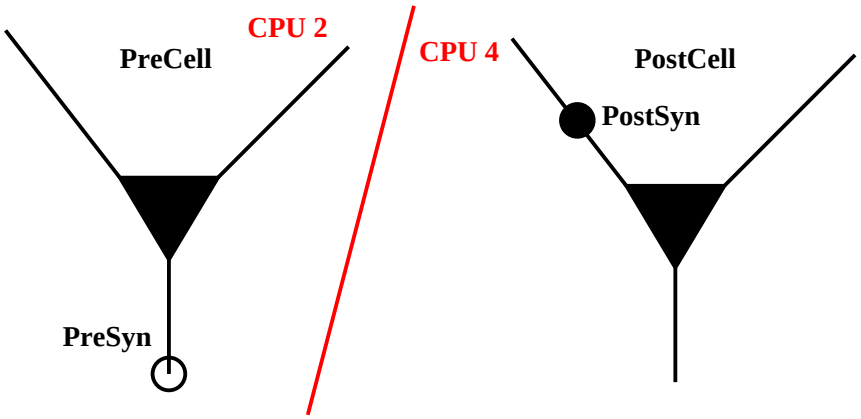
Single cells

portions of the tree cable equation on different machines.

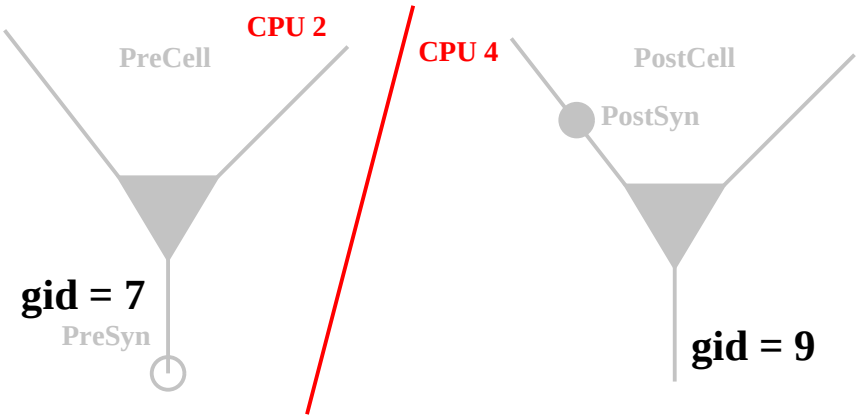



```
nc = new NetCon(PreSyn, PostSyn)
```






```
pc = new ParallelContext()
```



Every spike source (cell) must have a global id number.

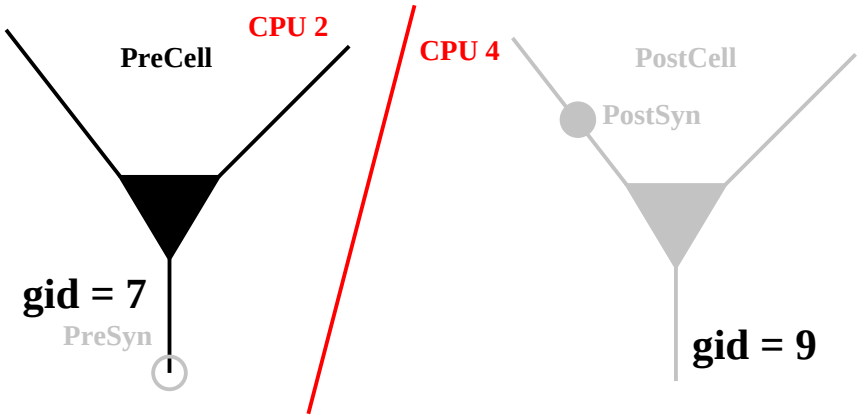
CPU 0		CPU 3		CPU 4	
pc.id	0	pc.id	3	pc.id	4
pc.nhost	5	pc.nhost	5	pc.nhost	5
ncell	14	ncell	14	ncell	14
gid		gid		gid	
0		3		4	
5		8		9	
10		13			

…

An efficient way to distribute:

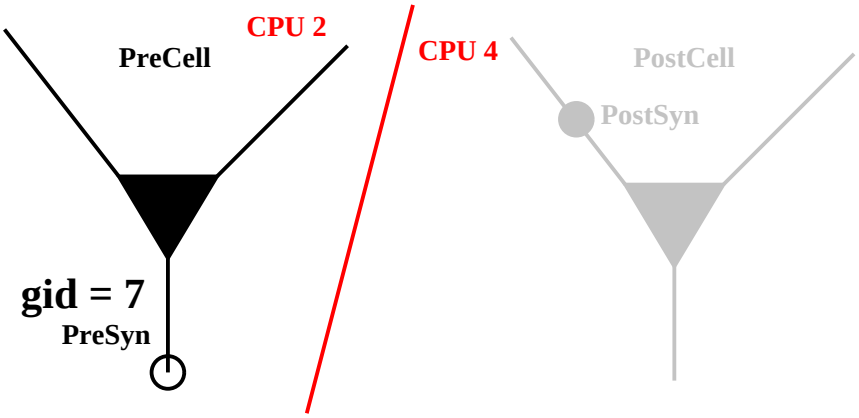
```
for (gid = pc.id; gid < ncell; gid += pc.nhost)
    pc.set_gid2node(gid, pc.id)
    ...
}
```

body executed only ncell/nhost times, not ncell.



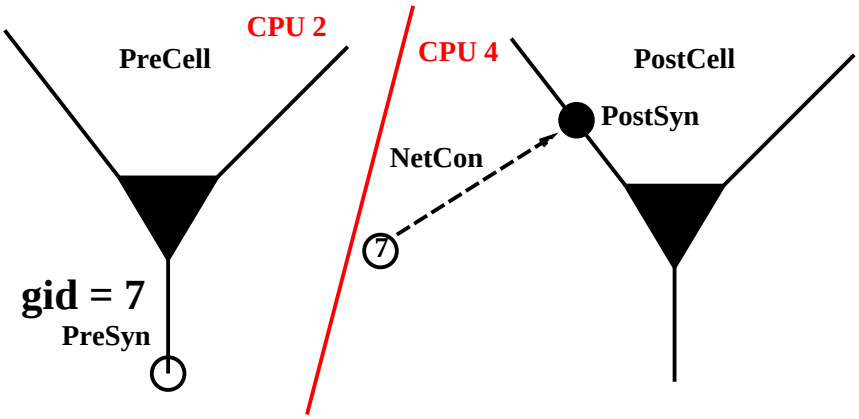
Create cell only where the gid exists.

```
if (pc.gid_exists(7)) {
    PreCell = new Cell()
}
```

Associate gid with spike source.

```
nc = new NetCon(PreSyn, nil)
pc.cell(7, nc)
```



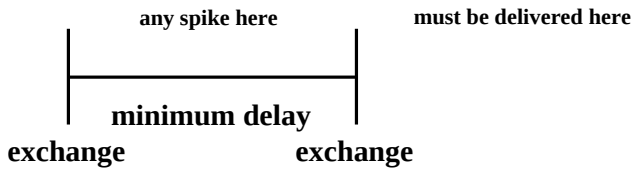
Create NetCon on CPU where target exists.

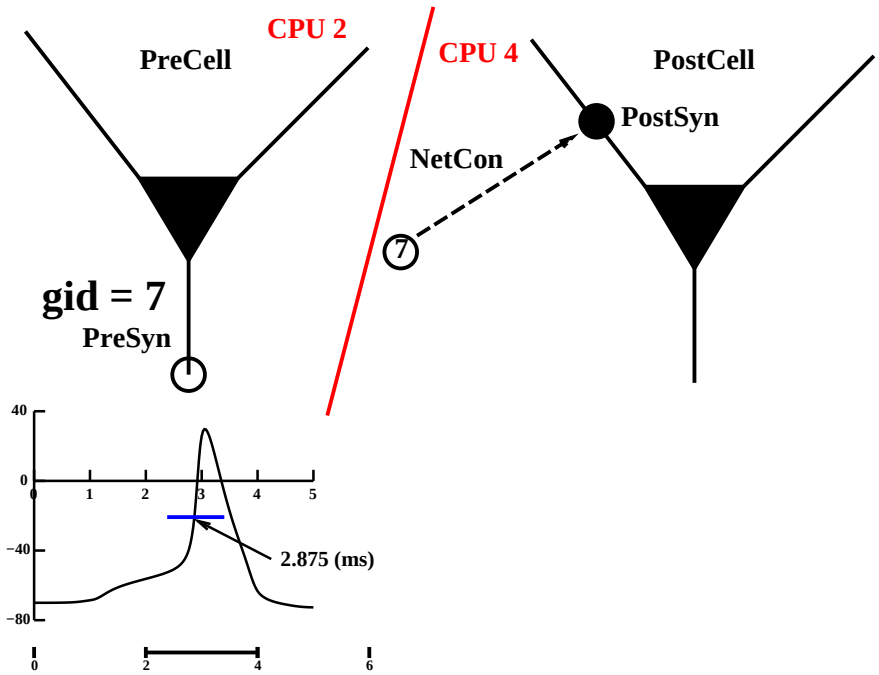
```
nc = pc.gid_connect(7, PostSyn)
```

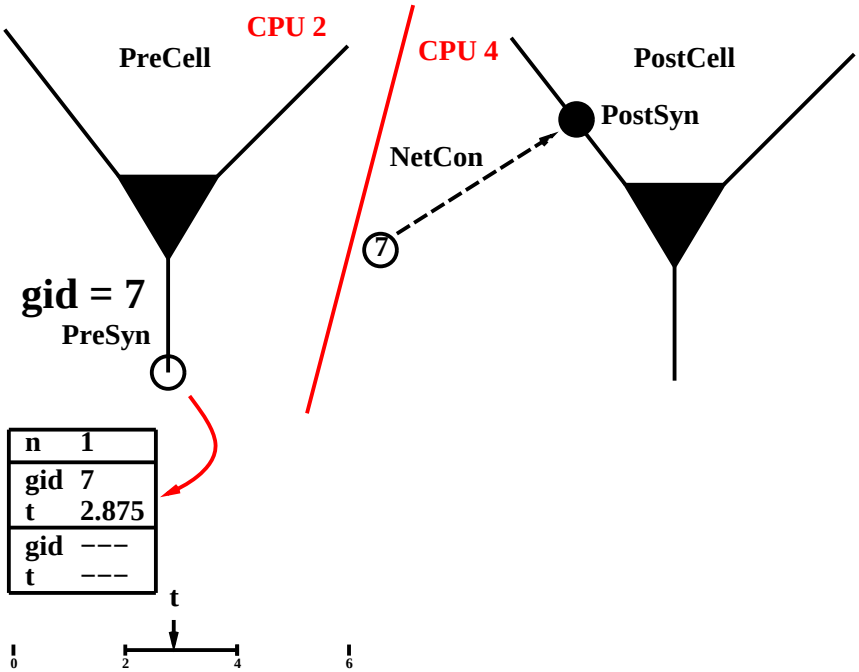

Run using the idiom

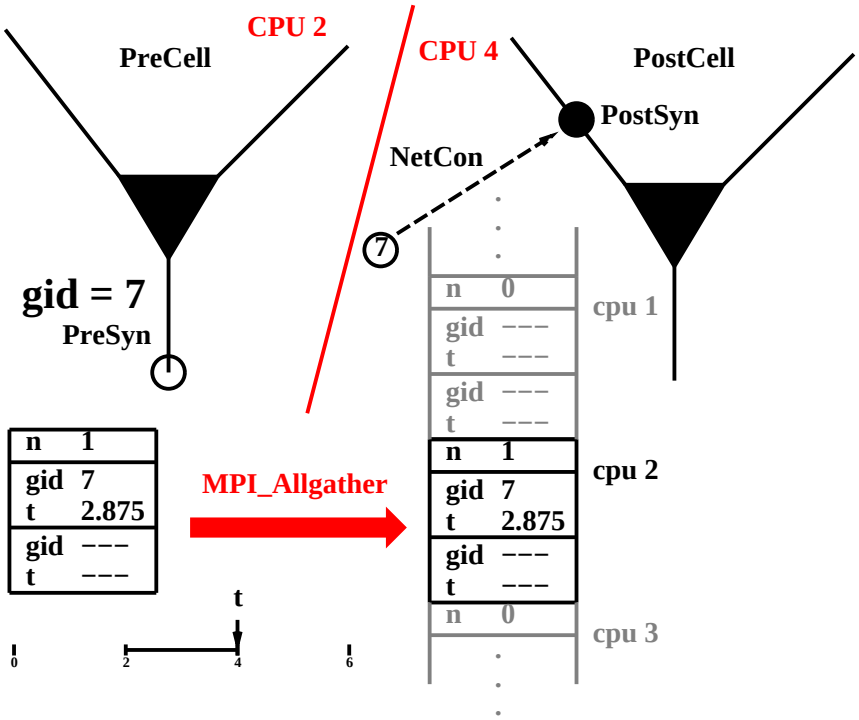
```
pc.set_maxstep(10)
stdinit()
pc.psolve(tstop)
```

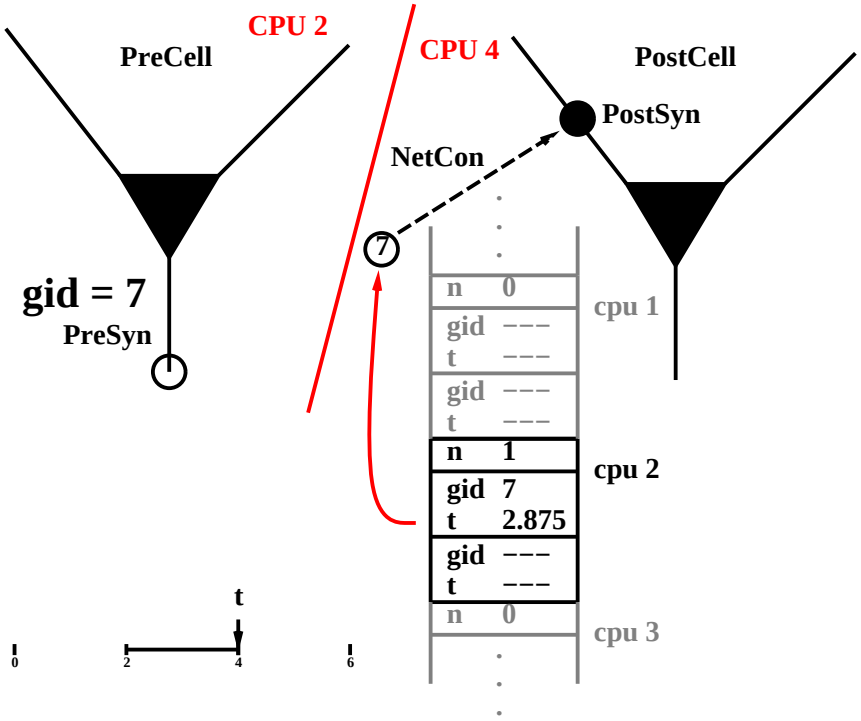
pc.set_maxstep() uses MPI_Allreduce to determine minimum delay.



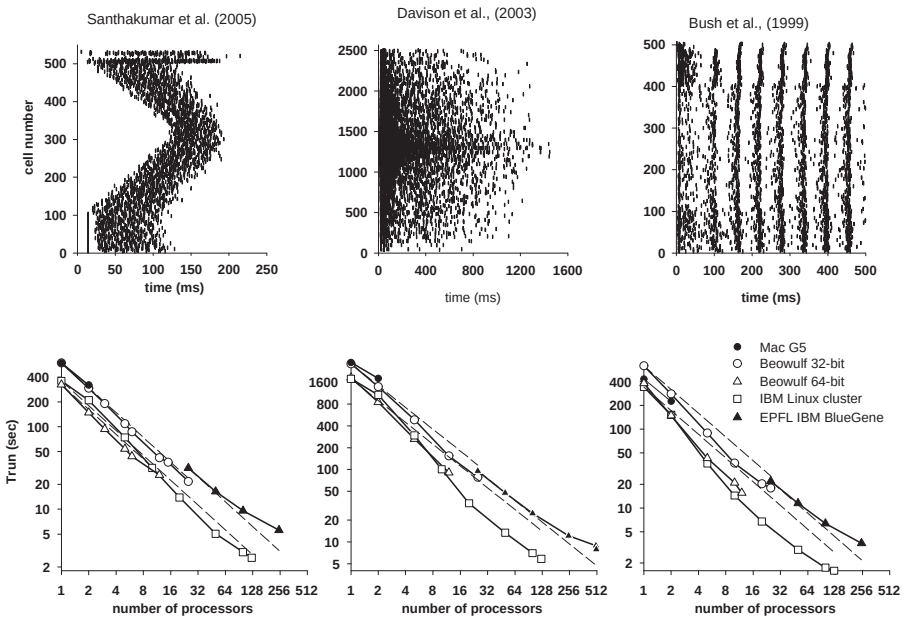


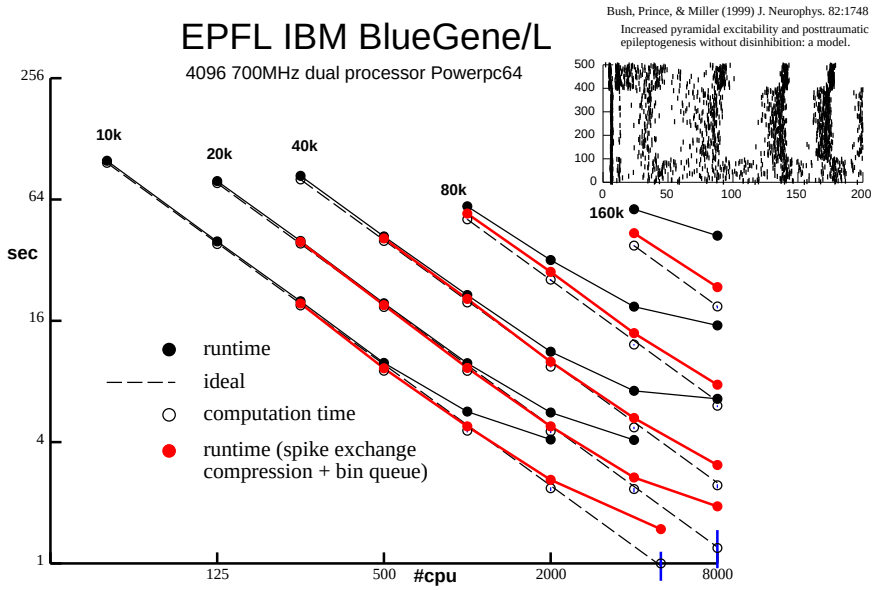




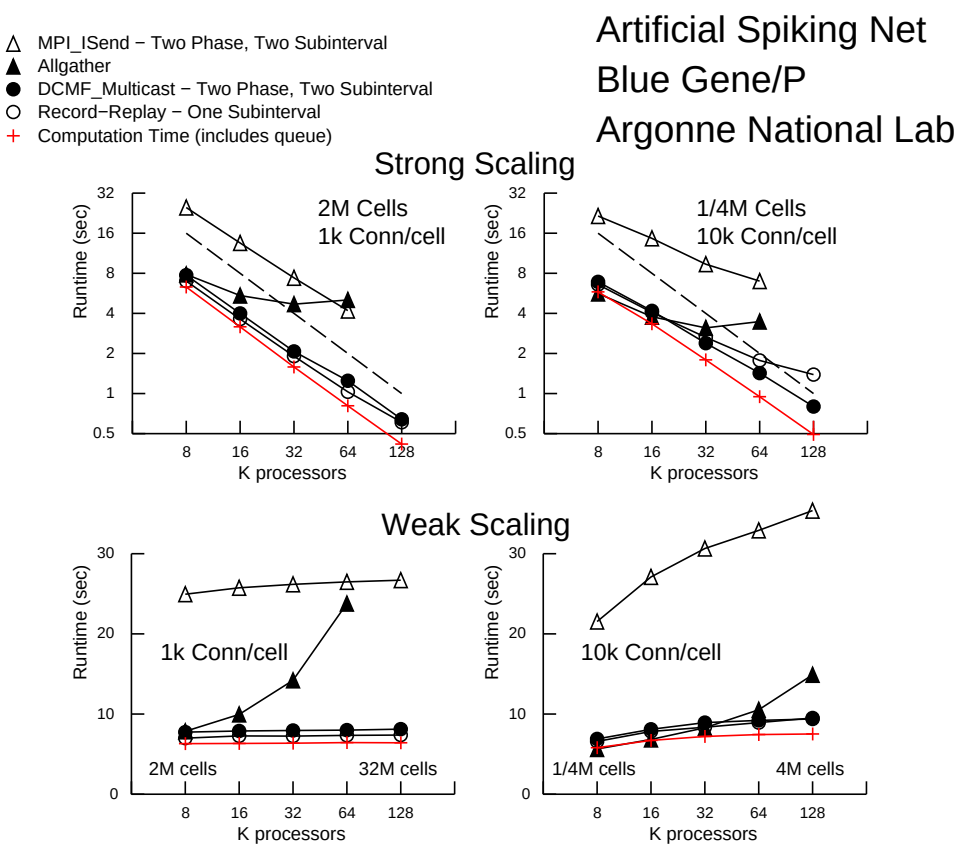


Migliore et al (2006) J. Comput. Neurosci. 21(2):119

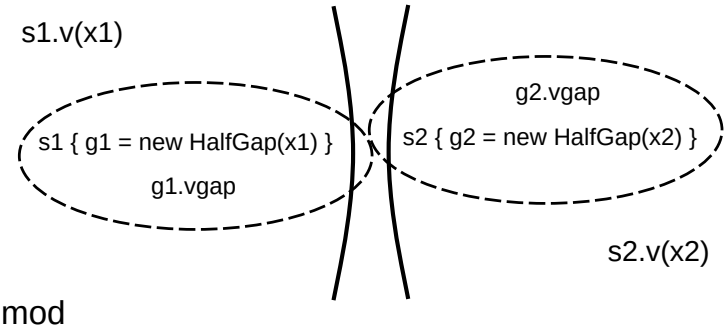




#cells	#states	#netcons	#outputSpikes	#spikedeliver
10k	444,664	17,167,785	31,118	52,040,794
20k	888,664	68,655,566	33,960	110,634,002
40k	1,777,664	274,591,128	69,529	452,987,907
80k	3,553,664	1,098,302,267	294974	2,147,483,647 *
160k	7,107,664	4,393,084,577	844,175	22,847,784,937



Continuous Voltage Exchange



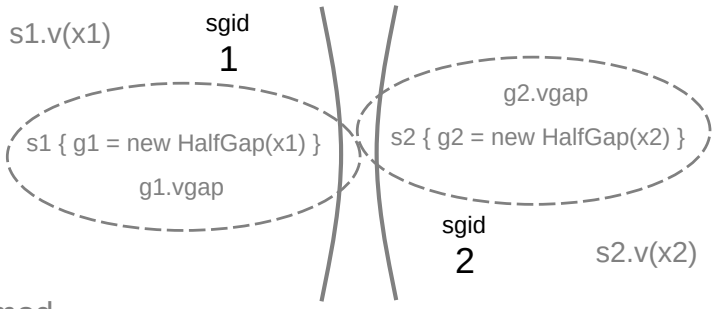
gap.mod

```
NEURON {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap − v)/r }
```

Continuous Voltage Exchange

pc.source_var(&source_var, sgid)



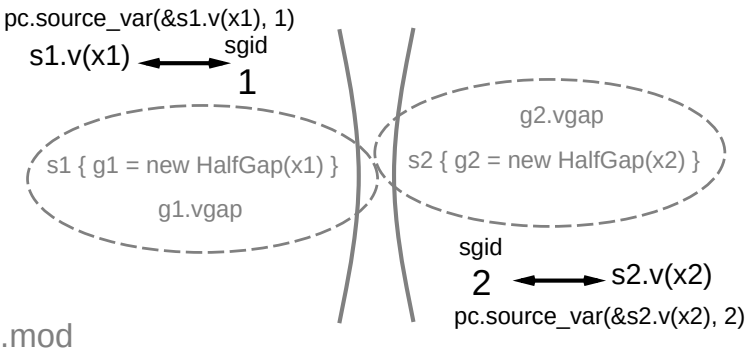
gap.mod

```
NEURON {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap − v)/r }
```


Continuous Voltage Exchange

pc.source_var(&source_var, sgid)



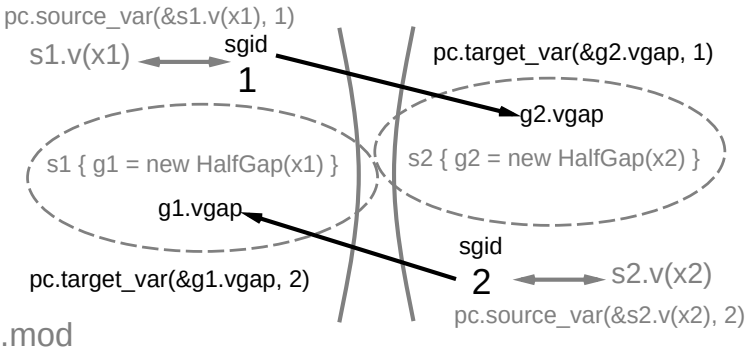
gap.mod

```
NEURON {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap − v)/r }
```

Continuous Voltage Exchange

pc.source_var(&source_var, sgid)
pc.target_var(&target_var, sgid)

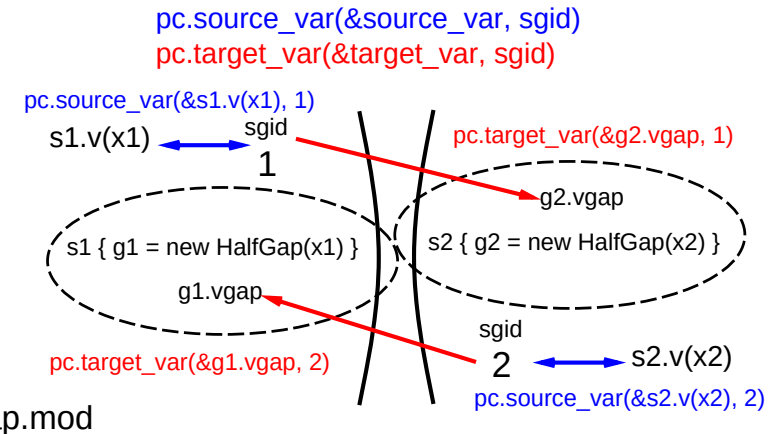


gap.mod

```
NEURON {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap − v)/r }
```


Continuous Voltage Exchange



gap.mod

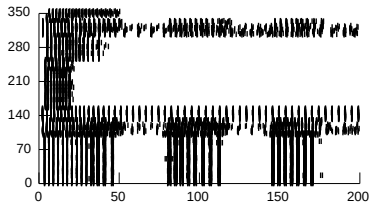
```
NEURON {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap - v)/r }
```

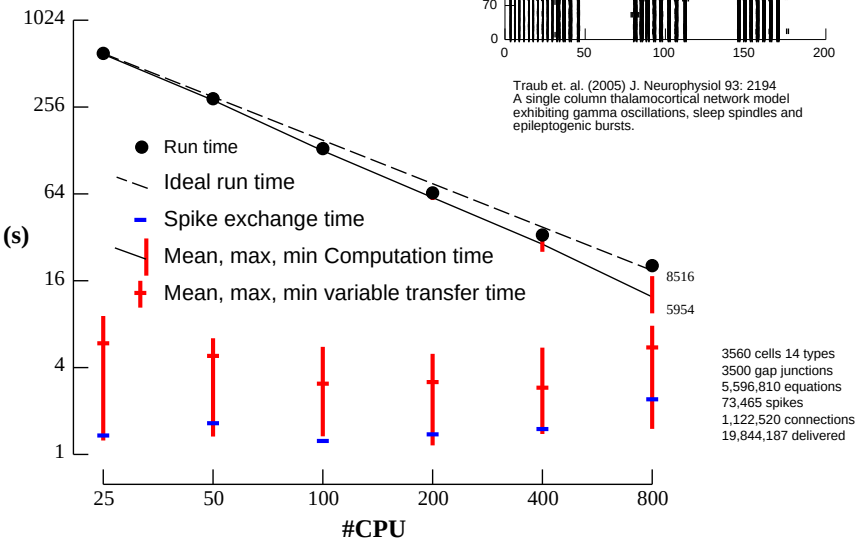
Pittsburgh Supercomputing Center

Bigben Cray XT3

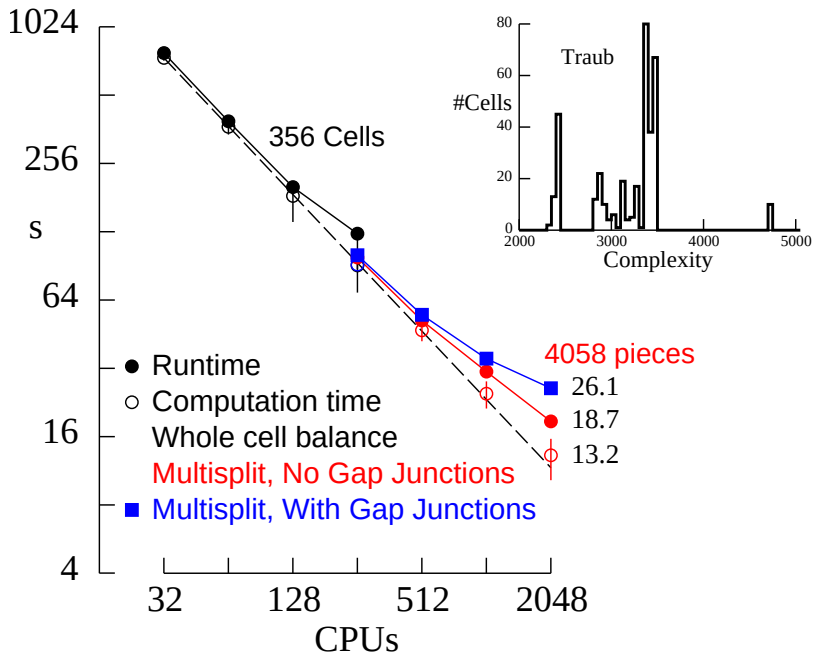
2068 2.4 GHz Opteron Processors



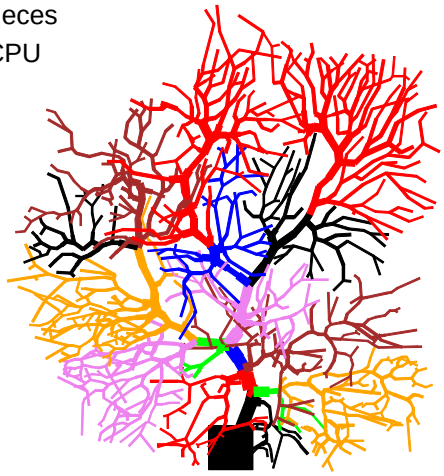
Traub et. al. (2005) J. Neurophysiol 93: 2194
A single column thalamocortical network model exhibiting gamma oscillations, sleep spindles and epileptogenic bursts.

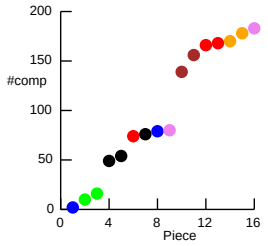


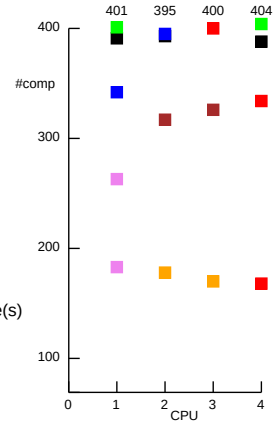
3560 cells 14 types
3500 gap junctions
5,596,810 equations
73,465 spikes
1,122,520 connections
19,844,187 delivered



16 Pieces
4 CPU



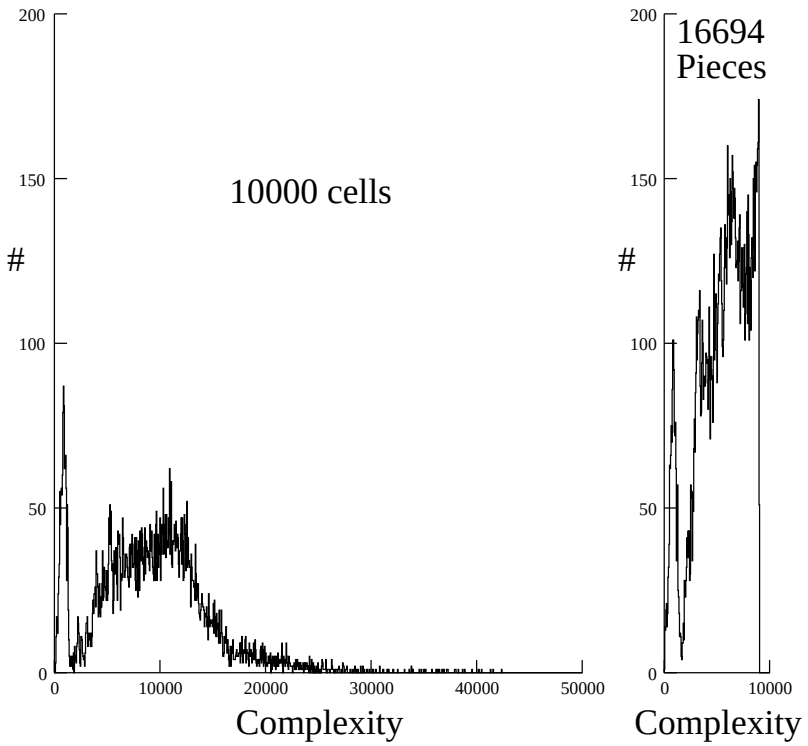


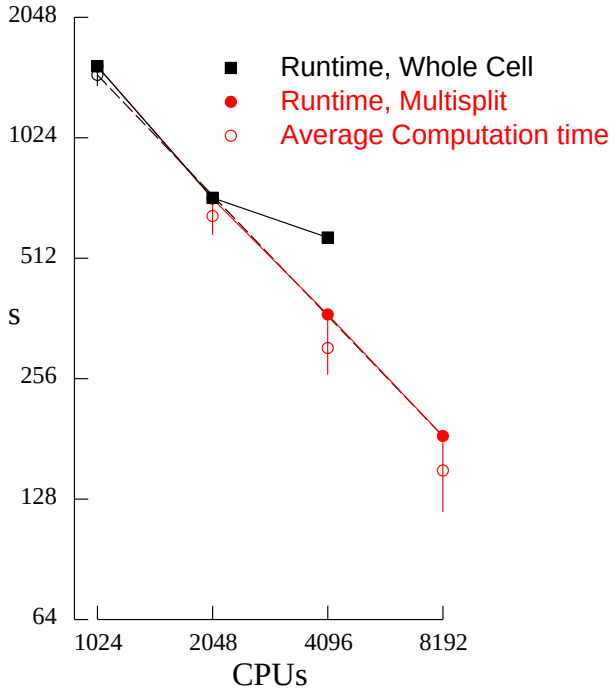


Time (s)

CPU	Computation	Exchange
0	13.82	0.56
1	13.35	1.03
2	13.47	0.90
3	13.56	0.82

	Runtime(s)
16 pieces, 1 cpu	55.0
wholecell, 1 cpu	56.2
16 pieces, 4 cpu	14.4



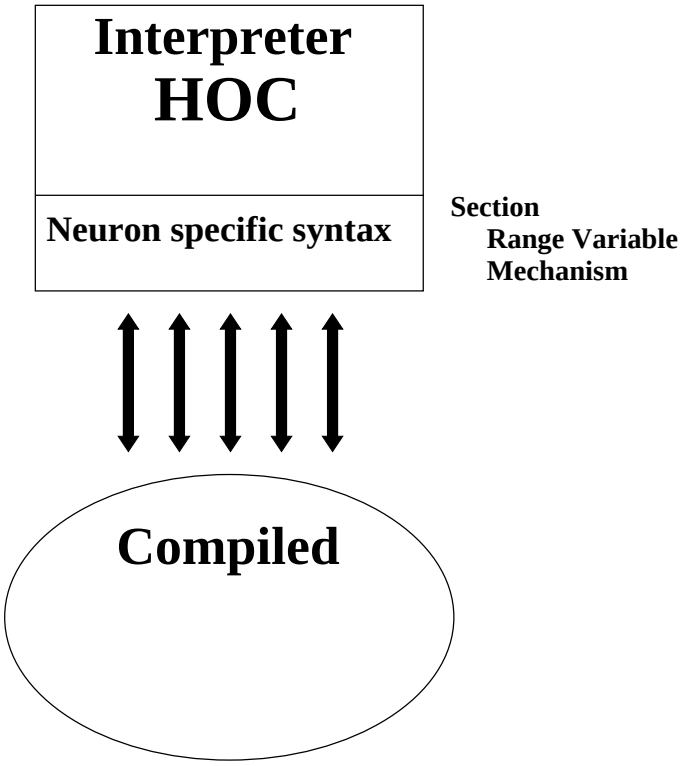


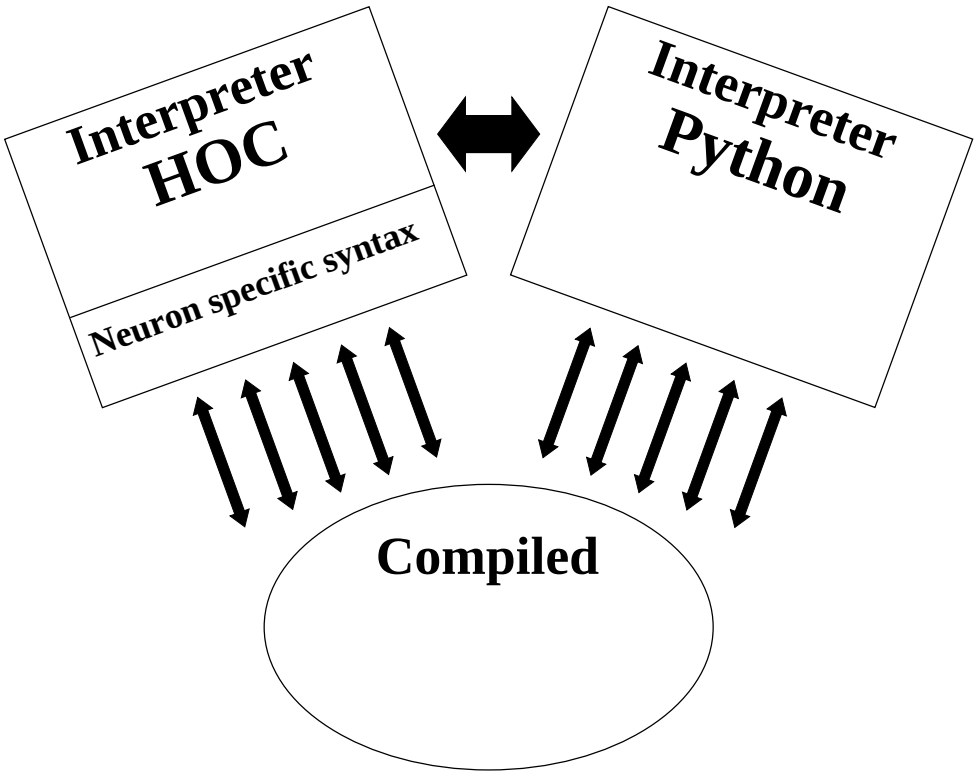
Python + NEURON

All legacy models must work.

Superior representation of underlying concepts.

No extra installation difficulty.

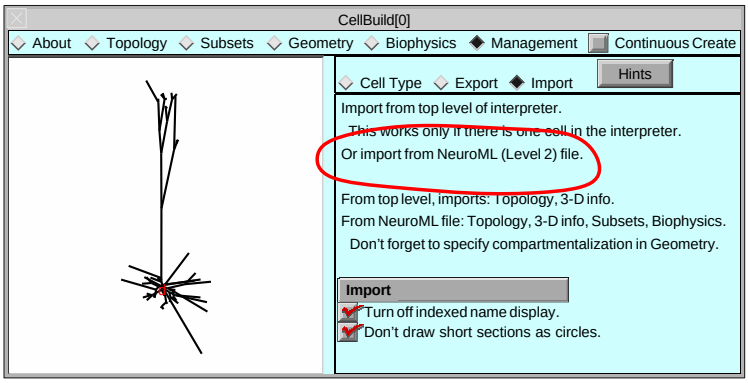


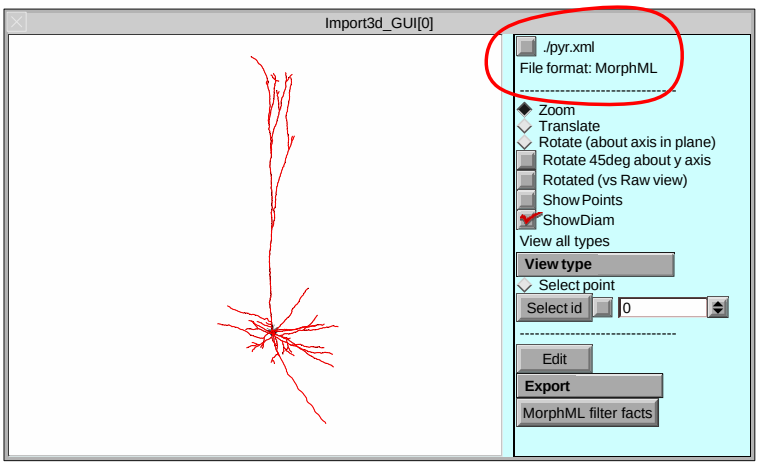


Installation

>>> import neuron

Linux	i686 x86_64		
Mac	OS X 10.5−8	Python	2.3−4 2.5−7 3.0−2
MSWin	Cygwin MinGW		
Launch	NEURON Python	NumPy	






```
$ nrniv -python
NEURON -- VERSION 7.3 ...

>>> from neuron import h
>>> print h
<hoc.HocObject object at 0x2b4f1b81e030>
>>> print h.hname()
TopLevelHocInterpreter
>>> h('''
... x = 5
... strdef s
... s = "hello"
... func square() { return $1*$1 }
... ''')
1
>>> print h.x, h.s, h.square(4)
5.0 hello 16.0
```



```
>>> v = h.Vector(4).indgen().add(10)
>>> print v.hname(), len(v), v.size(), v.x[2], v[2]
Vector[1] 4 4.0 12.0 12.0
>>> v.printf()
10      11      12      13
4.0
>>> for x in v: print x
...
10.0
11.0
12.0
13.0
>>>
```

```
>>> import numpy
>>> na = numpy.arange(0, 10, 0.00001) # 0.0131
>>> v = h.Vector(na)                  # 0.0197
>>> v.size()
1000000.0
>>> nb = numpy.array(v)               # 0.0125
>>> nb[999999]
9.9999900000000004
>>> b = list(v)                       # 0.0717
>>> for i in xrange(0, len(nb)):
...   v.x[i] = na[i]
...                                   # 3.7497
>>> nc = v.as_numpy()
>>> v.x[20] = 50.0
>>> nc[20]
50.0
```



```
>>> def callback(a = 1, b = 2):
...   print "callback: a=%d b=%d" % (a, b)
...
>>> fih = h.FInitializeHandler(callback)
>>> h.finitialize()
callback: a=1 b=2
1.0
>>> fih = h.FInitializeHandler((callback,\
... (4, 5)))
>>> h.finitialize()
callback: a=4 b=5
1.0
>>>
```

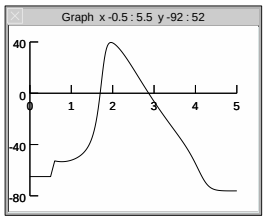
```
# assume hh soma model

vvec = h.Vector()
vvec.record(soma(.5)._ref_v, sec=soma)

tvec = h.Vector()
tvec.record(h._ref_t, sec=soma)

h.run()

g = h.Graph()
g.size(0, 5, -80, 40)
vvec.line(g, tvec)
```




```
>>> from neuron import h
>>> soma = h.Section(name = 'soma')
>>> axon = h.Section()
>>> axon.connect(soma(1))
>>> axon.nseg = 5
>>> h.topology()

|-|       soma(0-1)
   `----|       PySec_2b371cd17190(0-1)

1.0
>>> axon.L = 1000
>>> axon.diam = 1

>>> for sec in h.allsec():
...   sec.cm = 1
...   sec.Ra = 100
...   sec.insert('hh')
...
>>> axon.gnabar_hh = .1
>>> axon(.5).hh.gnabar = .09
>>> for seg in axon:
...   print seg.x, seg.hh.gnabar
...
0.1 0.1
0.3 0.1
0.5 0.09
0.7 0.1
0.9 0.1
```

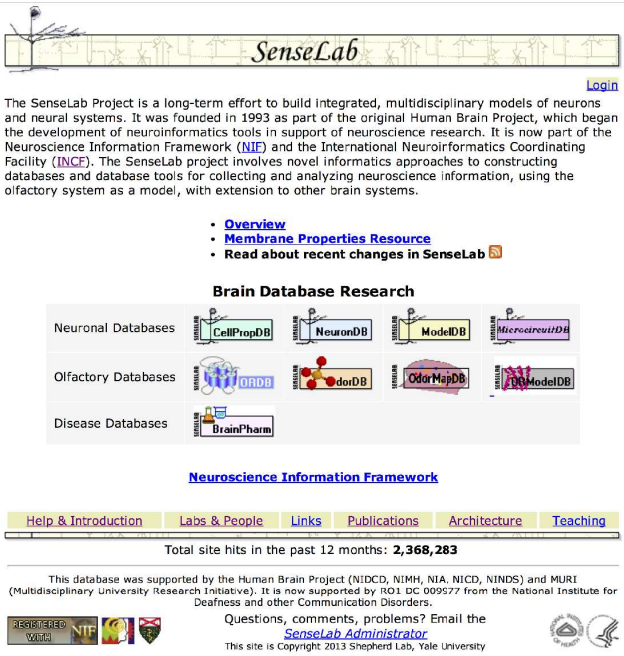


```
>>> stim = h.IClamp(soma(.5))
>>> stim.delay = .5
>>> stim.dur = .1
>>> stim.amp = .4
```

```
class Cell(object):
  def __init__(self):
    self.topology()
    self.subsets()
    ...
  def topology(self):
    self.soma = h.Section(cell = self)
    self.dend = h.Section(cell = self)
    self.dend.connect(self.soma)
    ...
  def subsets(self):
    self.all = h.SectionList()
    self.all.wholetree(sec=self.soma)
```


SenseLab

Neuron Course SfN 2013
Gordon Shepherd
Friday November 8th



NeuronDB

Back to SenseLab User: **Public**

NeuronDB provides a dynamically searchable database of three types of neuronal properties: voltage gated conductances, neurotransmitter receptors, and neurotransmitter substances. It contains tools that provide for integration of these properties in a given type of neuron and compartment, and for comparison of properties across different types of neurons and compartments. Read the tutorial for searching for Neuron Properties in NeuronDB

This resource is intended to:

- Support the genomics and proteomics of neuron types
- Support research on neuron properties
- Facilitate the creation of computational neuronal models
- Identify receptors across neuron types to aid in drug development
- Serve as a teaching aid

Search the Database By:

- Neuron List Alphabetically
- Neuron List By Brain Regions
- Membrane Properties Comprehensive Inventory: Channels, Receptors, Neurotransmitters/Neuromodulators
- Membrane Properties for NeuronDB: Currents, Receptors, Neurotransmitters/Neuromodulators
- Canonical forms of neurons (see explanation)
- Bibliographic citations

- Give us your feedback
- Deposit to the Database
- FAQ and related Links
- NeuronDB Login

This database is being developed by Luis N. Marenco[1], Rixin Wang[1], Thomas M. Morse[2], Perry L. Miller[1] and Gordon M. Shepherd[2], [1]*Center for Medical Informatics,* [2]*Department of Neurobiology, Yale University School of Medicine, New Haven, CT 06510.*

Some of the data, together with the graphics on the NeuronDB banner, are taken from The Synaptic Organization of the Brain, edited by G.M. Shepherd, New York: Oxford University Press (Second to Fourth Editions: 1979, 1990, 1998).

This database was supported by the Human Brain Project (NIDCD, NIMH, NIA, NICD, NINDS) and MURI (Multidisciplinary University Research Initiative). It is now supported by RO1 DC 009977 from the National Institute for Deafness and other Communication Disorders.

NeuronDB

Back User: **Public**

Olfactory bulb mitral cell

Mode: Overview Data/Search plus Connectivity **plus Classical References/Notes** Models **BrainPharm**

Region: Distal apical dendrite | Middle apical dendrite | Proximal apical dendrite | Distal basal dendrite | Middle basal dendrite | Proximal basal dendrite | Soma | Axon hillock | Axon fiber | Axon terminal | **All Compartments**

Properties: Receptors Channels Transmitters **All Properties**

Are: Present Absent

Neuron Type: principal

Organism: Vertebrates

ElectroPhysiology: NeuroElectro.org

Genes: Allen Brain Atlas - Links

	Input Receptors		Intrinsic Currents	Output Transmitters	
Distal apical dendrite	Glutamate from olfactory receptor neuron axon terminals	AMPA	I h (Angelo K and Margrie TW, 2011[560]) report the presence and function of Ih. I A (Bischofberger J and Jonas P, 1997[397]). I Na,t Implied by recording of fast prepotential. Dual patch recordings provide evidence for both backpropagating and forward-propagating impulses in the primary dendrite (Mori K, Nowycky M and Shepherd GM, 1982 [turtle][19] ; Chen et al 1997). Dendritic patch recordings showed an even density of Na channels (120pS um-2) up to 350 um from the soma along the primary dendrite to the origin of the glomerular tuft (Bischofberger J and Jonas P, 1997[397]). By	Glutamate	Mitral cell glomerular tuft and PG cell dendrites in the glomerulus (presumably) Implied by Glu released by other compartments of the mitral cell (Dale's law). Target (destination) is presumably PG cell dendrites in the glomerulus (van den Pol AN, 1995 [rat][1]).
	Intracellular recordings: CNQX blocks early component of EPSP elicited by olfactory nerve volley nerve volley (Berkowicz D and Trombley PQ and Shepherd GM, 1994 [turtle][10]). Electrophysiology data: DNQX attenuates early and late excitatory components in peristimulus time histograms of mitral cell unit responses to olfactory nerve volleys (Ennis and Zimmer LA and Shipley MT, 1996 [rat][20]). Intracellular recordings: CNQX blocks early component of EPSP response to olfactory nerve volley (Chen WR and Shepherd GM, 1997 [rat][16]). Paired whole-cell recording revealed reciprocal excitatory connections between mitral cells. Pharmacological analysis suggested that it could be mediated by both AMPA and NMDA receptors (Urban NN and Sakmann B, 2002[469]).				
	(autoreceptors) mitral cell glomerular tuft (Dad)	mGluR1			
	Auto-activation from glutamate released by mitral cell secondary dendrites (van den Pol AN, 1995 [rat][1]).				

SenseLab ModelDB

Models that contain the Neuron : *Olfactory bulb mitral cell*

Models

A two-layer biophysical olfactory bulb model of cholinergic neuromodulation (Li and Cleland 2013)
Compartmental model of a mitral cell (Popovic et al. 2005)
Dynamical model of olfactory bulb mitral cell (Rubin, Cleland 2006)
Functional structure of mitral cell dendritic tuft (Djurisic et al. 2008)
I&F models of ACh modulation in the olfactory bulb and piriform cortex (de Almeida et al. 2012)
Large scale model of the olfactory bulb (Yu et al., 2013)
Lateral dendrodenditic inhibition in the Olfactory Bulb (David et al. 2008)
Olfactory Bulb Network (Davison et al 2003)
Olfactory Computations in Mitral-Granule cell circuits (Migliore & McTavish 2013)
Olfactory Mitral Cell (Bhalla, Bower 1993)
Olfactory Mitral Cell (Davison et al 2000)
Olfactory Mitral Cell (Shen et al 1999)
Olfactory Mitral Cell: I-A and I-K currents (Wang et al 1996)
Olfactory Mitral cell: AP initiation modes (Chen et al 2002)
Olfactory bulb cluster formation (Migliore et al. 2010)
Olfactory bulb granule cell: effects of odor deprivation (Saghatelyan et al 2005)
Olfactory bulb mitral and granule cell column formation (Migliore et al. 2007)
Olfactory bulb mitral and granule cell: dendrodendritic microcircuits (Migliore and Shepherd 2008)
Olfactory bulb mitral cell gap junction NN model: burst firing and synchrony (O`Connor et al. 2012)
Olfactory bulb mitral cell: synchronization by gap junctions (Migliore et al 2005)
Olfactory bulb network model of gamma oscillations (Bathellier et al. 2006; Lagier et al. 2007)
Olfactory bulb network: neurogenetic restructuring and odor decorrelation (Chow et al. 2012)
Synchrony by synapse location (McTavish et al. 2012)

Re-display model names with descriptions

ModelDB Home SenseLab Home Help
Questions, comments, problems? Email the *ModelDB Administrator*
How to cite ModelDB

SenseLab ModelDB

Models that contain the Neuron : *Olfactory bulb mitral cell*

Models	Description
A two-layer biophysical olfactory bulb model of cholinergic neuromodulation (Li and Cleland 2013)	This is a two-layer biophysical olfactory bulb (OB) network model to study cholinergic neuromodulation. Simulations show that nicotinic receptor activation sharpens mitral cell receptive field, while muscarinic receptor activation enhances network synchrony and gamma oscillations. This general model suggests that the roles of nicotinic and muscarinic receptors in OB are both distinct and complementary to one another, together regulating the effects of ascending cholinergic inputs on olfactory bulb transformations.
Compartmental model of a mitral cell (Popovic et al. 2005)	Usage of a morphologically realistic compartmental model of a mitral cell and data obtained from whole-cell patch-clamp and voltage-imaging experiments in order to explore passive parameter space in which reported low EPSP attenuation is observed.
Dynamical model of olfactory bulb mitral cell (Rubin, Cleland 2006)	This four-compartment mitral cell exhibits endogenous subthreshold oscillations, phase resetting, and evoked spike phasing properties as described in electrophysiological studies of mitral cells. It is derived from the prior work of Davison et al (2000) and Bhalla and Bower (1993). See readme.txt for details.
Functional structure of mitral cell dendritic tuft (Djurisic et al. 2008)	The computational modeling component of Djurisic et al. 2008 addressed two primary questions: whether amplification by active currents is necessary to explain the relatively mild attenuation suffered by tuft EPSPs spreading along the primary dendrite to the soma; what accounts for the relatively uniform peak EPSP amplitude throughout the tuft. These simulations show that passive spread from tuft to soma is sufficient to yield the low attenuation of tuft EPSPs, and that random distribution of a biologically plausible number of excitatory synapses throughout the tuft can produce the experimentally observed uniformity of depolarization.
I&F models of ACh modulation in the olfactory bulb and piriform cortex (de Almeida et al. 2012)	This matlab code was used in the paper "A model of cholinergic modulation in olfactory bulb and piriform cortex" (de Almeida, Idiart and Linster, 2013). This work uses a computational model of the OB and PC and their common cholinergic inputs to investigate how bulbar cholinergic modulation affects cortical odor processing.
Large scale model of the olfactory bulb (Yu et al., 2013)	The readme file currently contains links to the results for all the 72 odors investigated in the paper, and the movie showing the network activity during learning of odor k3-3 (an aliphatic ketone).

SenseLab ModelDB

A single column thalamocortical network model (Traub et al 2005)

Accession: 45539

To better understand population phenomena in thalamocortical neuronal ensembles, we have constructed a preliminary network model with 3,560 multicompartment neurons (containing soma, branching dendrites, and a portion of axon). Types of neurons included superficial pyramids (with regular spiking [RS] and fast rhythmic bursting [FRB] firing behaviors); RS spiny stellates; fast spiking (FS) interneurons, with basket-type and axoaxonic types of connectivity, and located in superficial and deep cortical layers; low threshold spiking (LTS) interneurons, that contacted principal cell dendrites; deep pyramids, that could have RS or intrinsic bursting (IB) firing behaviors, and endowed either with non-tufted apical dendrites or with long tufted apical dendrites; thalamocortical relay (TCR) cells; and nucleus reticularis (nRT) cells. To the extent possible, both electrophysiology and synaptic connectivity were based on published data, although many arbitrary choices were necessary.

References:

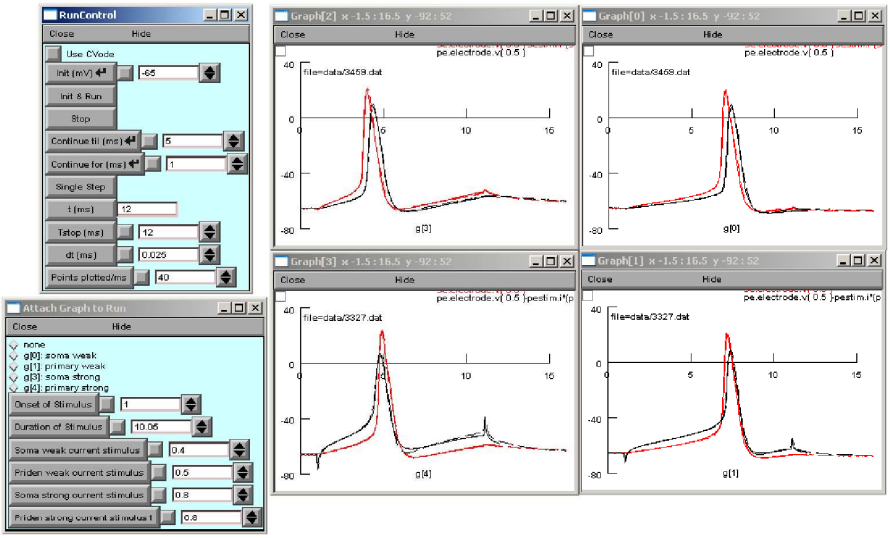
1. Traub RD, Contreras D, Cunningham MO, Murray H, Lebeau FE, Roopun A, Bibbig A, et al (2005) A single-column thalamocortical network model exhibiting gamma oscillations, sleep spindles and epileptogenic bursts: Appendix B. *J Neurophysiol* **93(4)**:2194-232 [PubMed]
2. Traub RD, Contreras D, Whittington MA (2005) Combined experimental-simulation studies of cellular and network mechanisms of epileptogenesis in vitro and in vivo. *J Clin Neurophysiol* **22**:330-42 [PubMed]

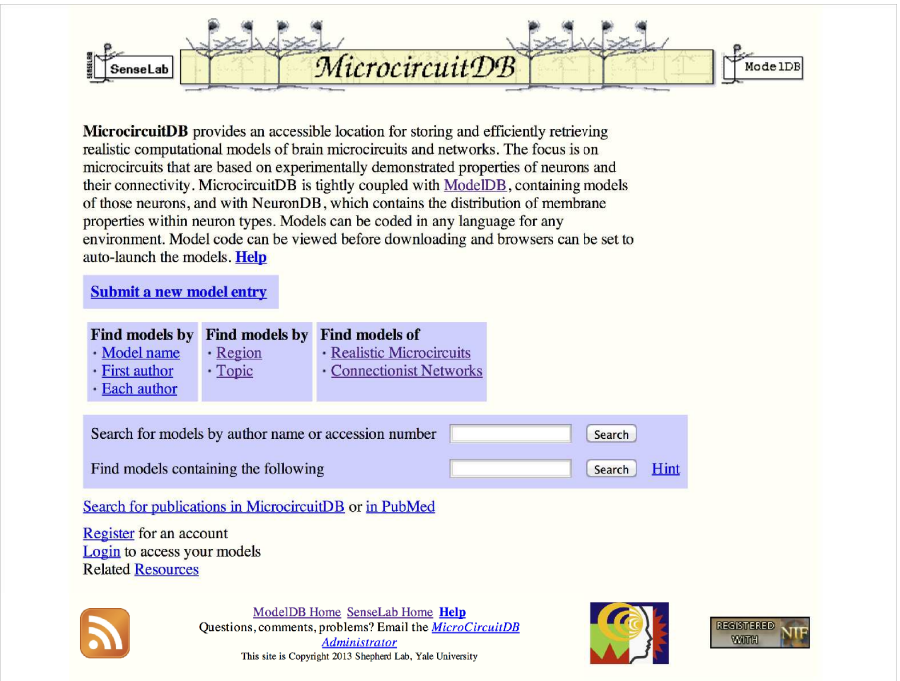
Citations Citation Browser

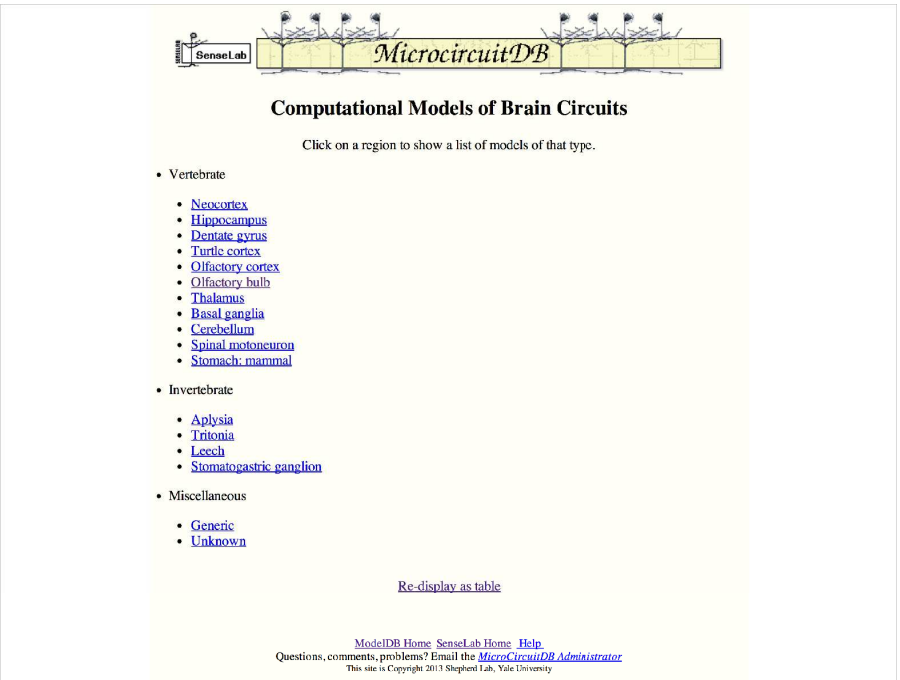
Model Information (*Click on a link to find other models with that property*)

Model Type:	Network;
Brain Region(s)/Organism:	Neocortex; Thalamus;
Cell Type(s):	Thalamic relay neuron; Thalamic reticular neuron; Neocortical pyramidal neuron: deep; Neocortical pyramidal neuron: superficial; Neocortical Fast Spiking (FS) interneuron; Neocortical Regular Spiking (RS) neuron; Neocortical Low Threshold Spiking (LTS) neuron;
Channel(s):	I Na,p; I Na,t; I L high threshold; I T low threshold; I A; I K; I M; I h; I K,Ca; I Calcium; I A, slow;
Gap Junctions:	Gap junctions;
Receptor(s):	GabaA; AMPA; NMDA;
Gene(s):	
Transmitter(s):	
Simulation Environment:	Neuron; FORTRAN;
Model Concept(s):	Activity Patterns; Bursting; Temporal Pattern Generation; Oscillations; Simplified Models; Epilepsy; Sleep;
Implementer(s):	Traub, Roger D.;

Search NeuronDB for information about: Thalamic relay neuron; Thalamic reticular neuron; Neocortical pyramidal neuron: deep; Neocortical pyramidal neuron: superficial; GabaA; AMPA; NMDA; I Na,p; I Na,t; I L high threshold; I T low threshold; I A; I K; I M; I h; I K,Ca; I Calcium; I A, slow;







SenseLab MicrocircuitDB

Circuits that contain the Region : *Neocortex*

Models

A Fast Rhythmic Bursting Cell: in vivo cell modeling (Lee 2007)
A Neural mass computational model of the Thalamocorticothalamic circuitry (Bhattacharya et al. 2011)
A fast model of voltage-dependent NMDA Receptors (Moradi et al. 2012)
A microcircuit model of the frontal eye fields (Heinzle et al. 2007)
A neurocomputational model of classical conditioning phenomena (Moustafa et al. 2009)
A single column thalamocortical network model (Traub et al 2005)
A spiking model of cortical broadcast and competition (Shanahan 2008)
AP back-prop, explains threshold variability and rapid rise (McCormick et al. 2007, Yu et al. 2008)
Accurate and fast simulation of channel noise in conductance-based model neurons (Linaro et al 2011)
Action potential-evoked Na+ influx are similar in axon and soma (Fleidervish et al. 2010)
Asynchronous irregular and up/down states in excitatory and inhibitory NNs (Destexhe 2009)
Axonal Projection and Interneuron Types (Helmstaedter et al. 2008)
Biophysically realistic neural modeling of the MEG mu rhythm (Jones et al. 2009)
CONFIGR: a vision-based model for long-range figure completion (Carpenter et al. 2007)
Compartmentalization of GABAergic inhibition by dendritic spines (Chiu et al. 2013)
Composite spiking network/neural field model of Parkinsons (Kerr et al 2013)
Computational Surgery (Lytton et al. 2011)
Computational aspects of feedback in neural circuits (Maass et al 2006)
Cortical network model of posttraumatic epileptogenesis (Bush et al 1999)
Dendritic Discrimination of Temporal Input Sequences (Branco et al. 2010)
Dendritic Na+ spike initiation and backpropagation of APs in active dendrites (Nevian et al. 2007)
Development of orientation-selective simple cell receptive fields (Rishikesh and Venkatesh, 2003)
Efficient simulation environment for modeling large-scale cortical processing (Richert et al. 2011)
Emergence of physiological oscillation frequencies in neocortex simulations (Neymotin et al. 2011)
Engaging distinct oscillatory neocortical circuits (Vierling-Claassen et al. 2010)
Event-related simulation of neural processing in complex visual scenes (Mihalas et al. 2011)
Excitability of PFC Basal Dendrites (Acker and Antic 2008)

Receipt

Received: $110

From:

For: Using the NEURON Simulation Environment
Held Nov. 8, 2013 in San Diego, CA
http://www.neuron.yale.edu/neuron/static/courses/sd2013/sd2013.html

By: N.T. Carnevale
Director, Using the NEURON Simulation Environment
203-494-7381
ted.carnevale@yale.edu

For deposit in: Yale University account "NNC--Fees"

Survey

We'd appreciate your frank opinions and suggestions to help us refine this course and design future offerings on related subjects.

Please score these	**. . . . .**	**according to this scale**	
Overall impression	_____	no opinion	0
Relevance to my research	_____	poor, not helpful	1
Didactic presentations	_____	fair	2
Written handouts	_____	good	3
Overhead transparencies	_____	excellent, very helpful	4
Computer projection	_____		
Classroom	_____		
Food	_____		

Best feature _____

Weakest feature _____

_____

Additional topics that should be covered, topics that should receive more or less coverage, or other suggestions for improvement.

_____

_____

_____

Circle one

Y N I would recommend this course to others who are interested in neural modeling.

Y N I have developed my own modeling software using a high-level language (FORTRAN, C/C++ etc.).

Y N I have created my own models using modeling software.

Which software? _____

My primary area of research interest is _____

To help us better meet the needs of NEURON users, please circle all platforms that you plan to use for modeling.

Hardware Mac PC Other _____

OS MacOS X Win XP | Vista | 7 | 8 UNIX | Linux | OS X | BSD

If Linux, which distribution? _____