Partial Menu of Presentations

NTC	Ted Carnevale
MLH	Michael Hines
WWL	Bill Lytton
FS	Felix Schürmann

CNS*2007 NEURON Course

Toronto, Canada
Wednesday, July 11, 2007

Michael Hines
Ted Carnevale
Bill Lytton
Felix Schürmann

Supported by NINDS

The What and the Why of Neural Modeling

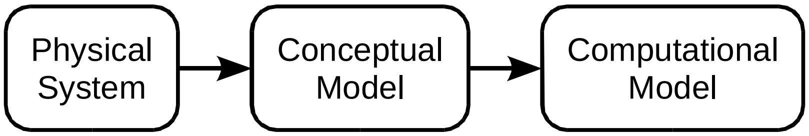
The moment-to-moment processing of information in the nervous system involves the propagation and interaction of electrical and chemical signals that are distributed in space and time.

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

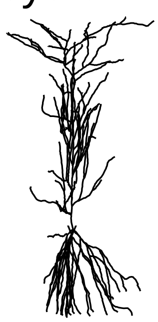
a simplified representation of the physical system

Computational model

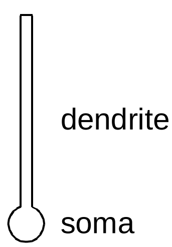
an accurate representation of the conceptual model

From Physical System to Computational Model

Physical system	Conceptual model	Computational model



Ca1 pyramidal cell



ball and stick

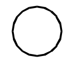
```
create soma, dendrite
connect dendrite(0), soma(1)
```

hoc code

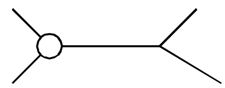
Hierarchies of Complexity

Structure

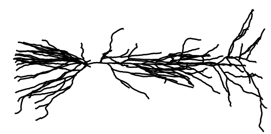
Single compartment



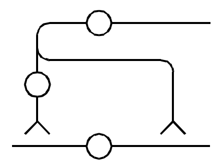
Stylized



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

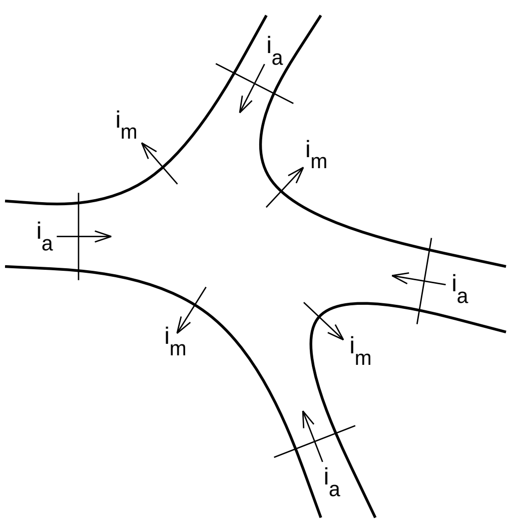
Diffusion, buffers, transport & exchange

Artificial spiking cells ("integrate & fire")

Fundamental Concepts in NEURON

Signals	**Flux**	**Driving force**	**What is conserved**
Electrical	current	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

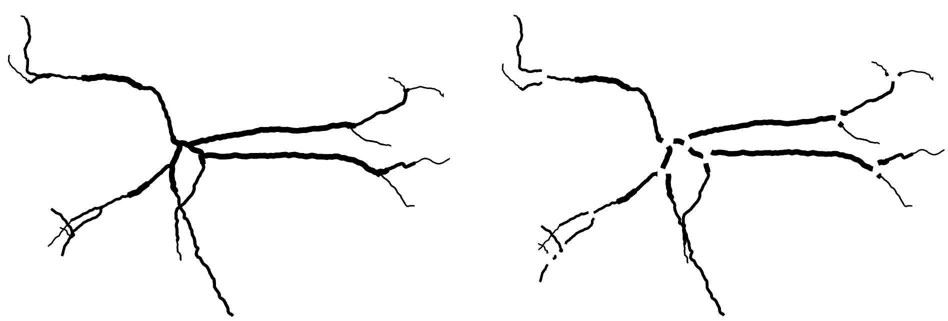
c_j membrane capacitance of compartment j

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

Range Variables

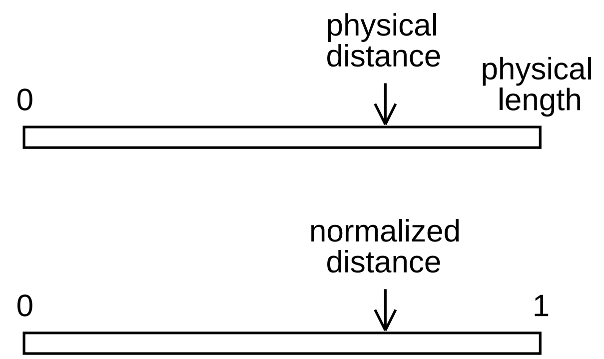
Name	Meaning	Units
`diam`	diameter	[µm]
`cm`	specific membrane capacitance	[µf/cm^2]
`g_pas`	specific conductance of the `pas` mechanism	[siemens/cm^2]
`v`	membrane potential	[mV]

range

normalized position along the length of a section

0 ≤ range ≤ 1

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



Syntax:

```
sectionname.rangevar(range)
```

returns or sets the value of rangevar
at the location corresponding to range

Examples:

```
dend.v(0.5)
```

returns membrane potential at the 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

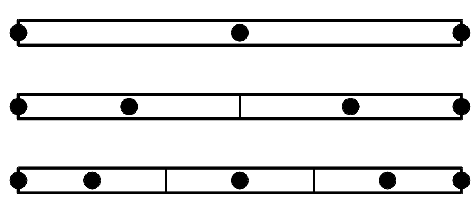
nseg

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

nseg=1

nseg=2

nseg=3



Example: `axon nseg = 3`

To test spatial resolution

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

and repeat the simulation

Category	**Variable**	**Units**
Time	`t`	[ms]
Voltage	`v`	[mV]
Current	`i`	[mA/cm^2] (density) [nA] (point process)
Concentration	`nai` etc.	[mM]
Specific capacitance	`cm`	[µf/cm^2] (density)
Length	`diam, L`	[µm]
Conductance	`g`	[S/cm^2] (density) [µS] (point process)
Cytoplasmic resistivity	`Ra`	[Ω cm]
Resistance	`ri`	[10^6 Ω]

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.

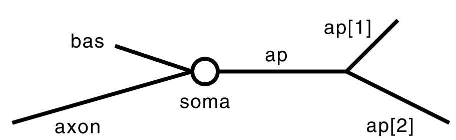
Step 0: Conceptualize the task

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

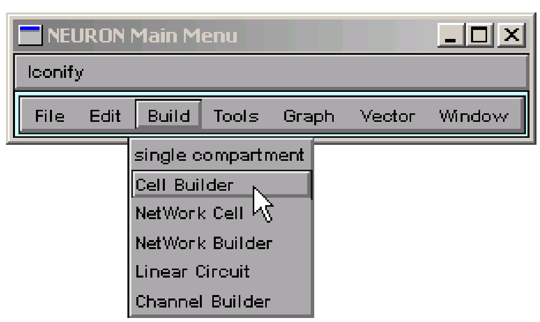
Throughout the cell Ra = 160 Ω cm, cm = 1 µf / cm^2

Launch NEURON with its library of graphical tools

UNIX/Linux `nrngui`

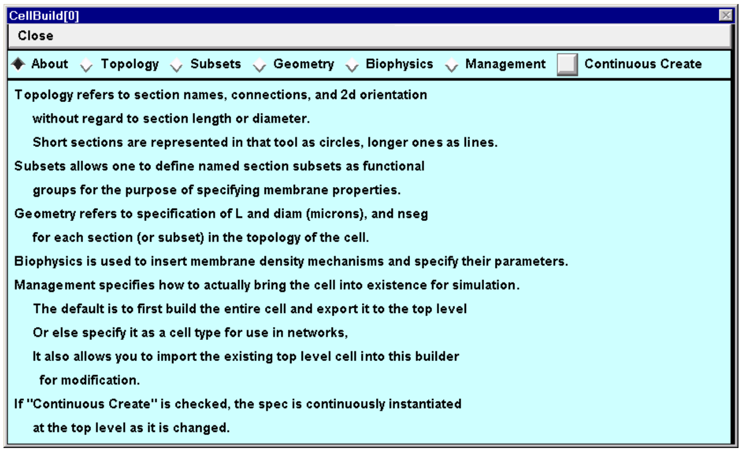
MSWin or OS X

Bring up a CellBuilder



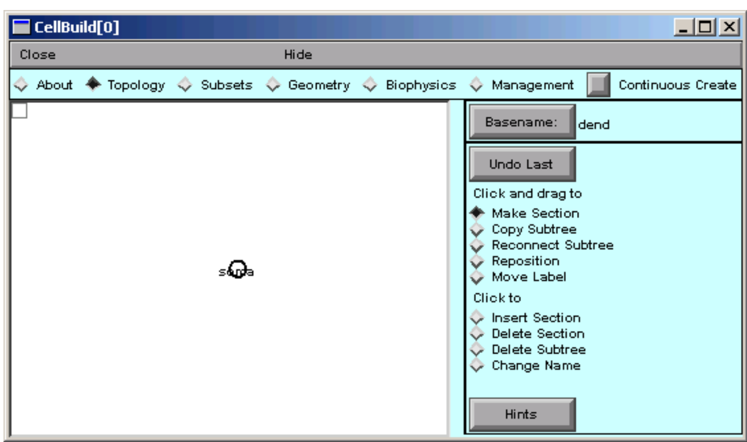
NEURON Main Menu / Build / Cell Builder

The CellBuilder



Use buttons from left to right.

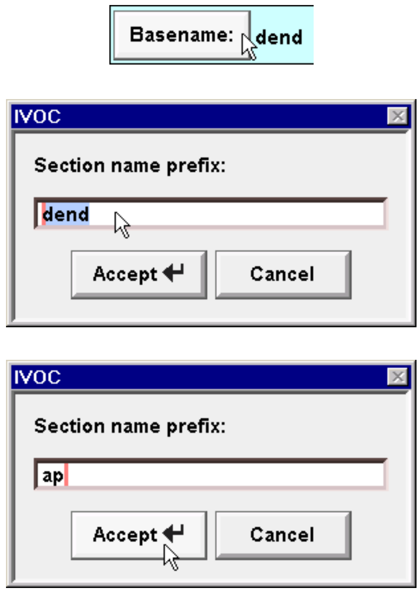
Topology



CB starts with a "soma" section.

We want to create new sections.

Specifying the "Basename"



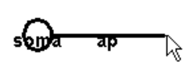
Making a new section

Place cursor near end of existing section

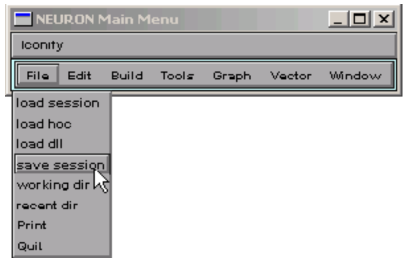
Click to start new section

Drag to desired length

Release mouse button

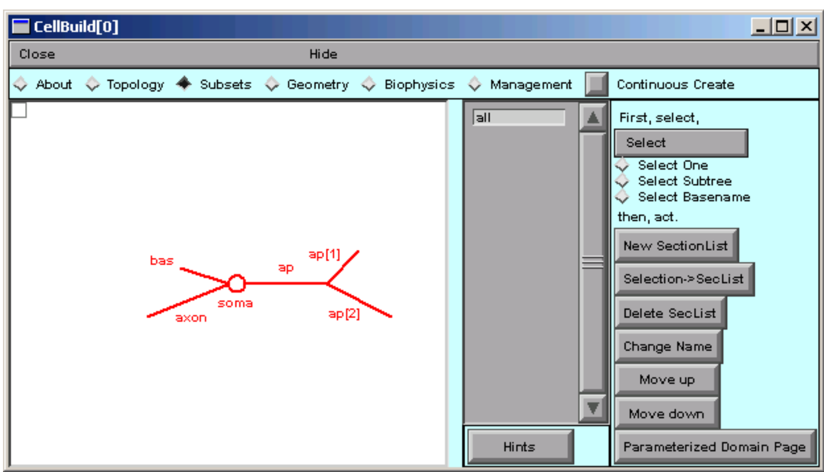


Save your work as you make progress!



NEURON Main Menu / File / save session

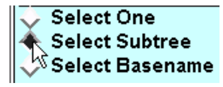
Subsets



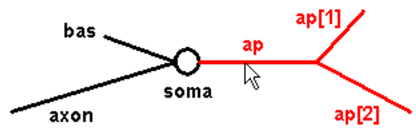
Group sections that have shared properties.
We want to make an "apicals" subset.

Making a new subset

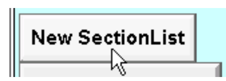
Click "Select Subtree"



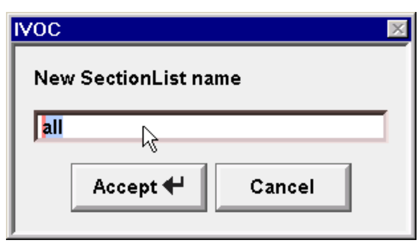
Click root of apical tree . . .

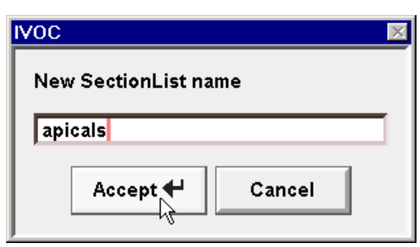


. . . then "New SectionList"

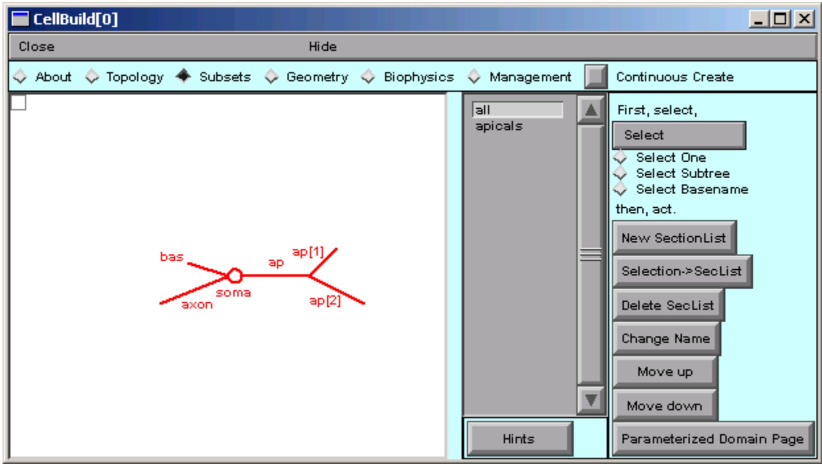


Making a new subset *continued*





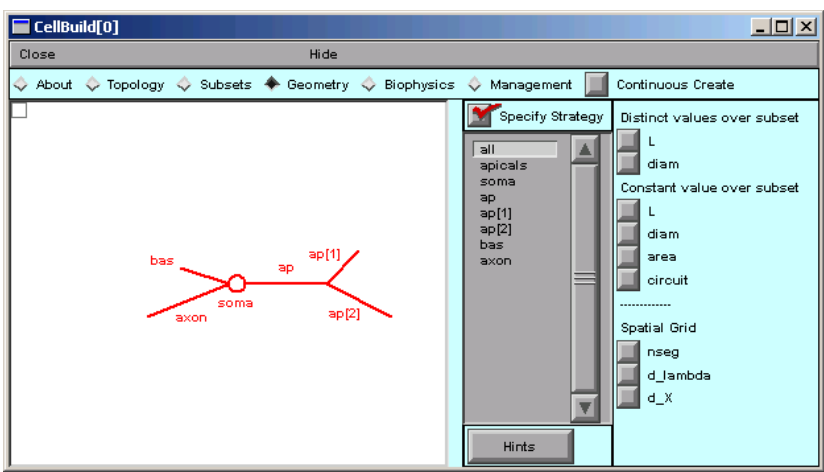
Subsets finished



Note "apicals".

Time to save a new session file.

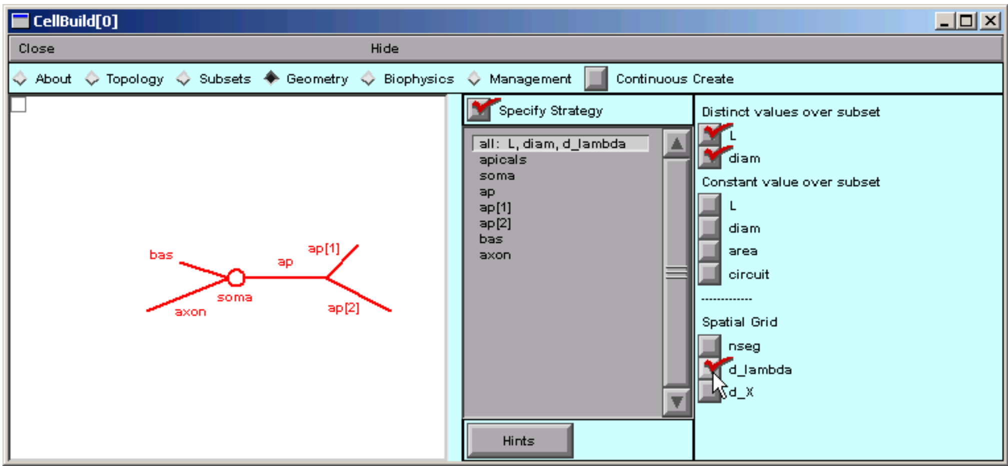
Geometry



"Specify Strategy" is ON.

A good strategy is a concise strategy.

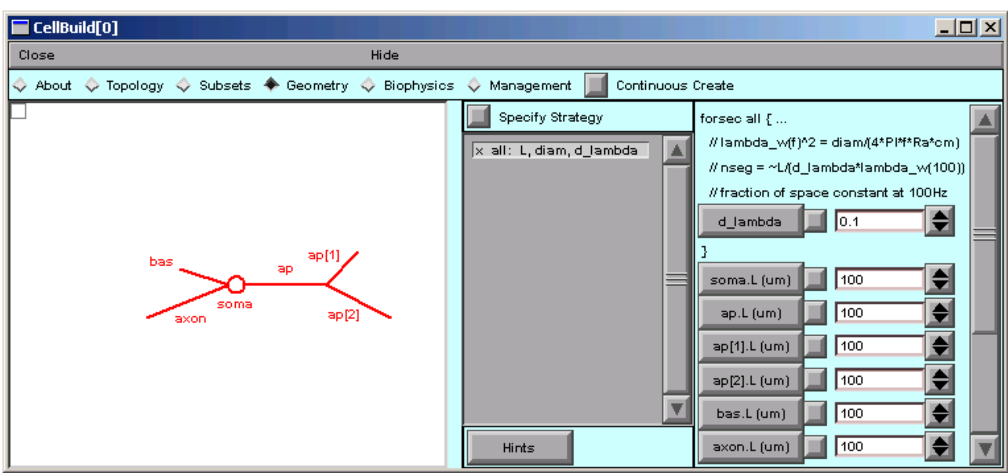
Geometry strategy



Each section has a different L and diam.

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

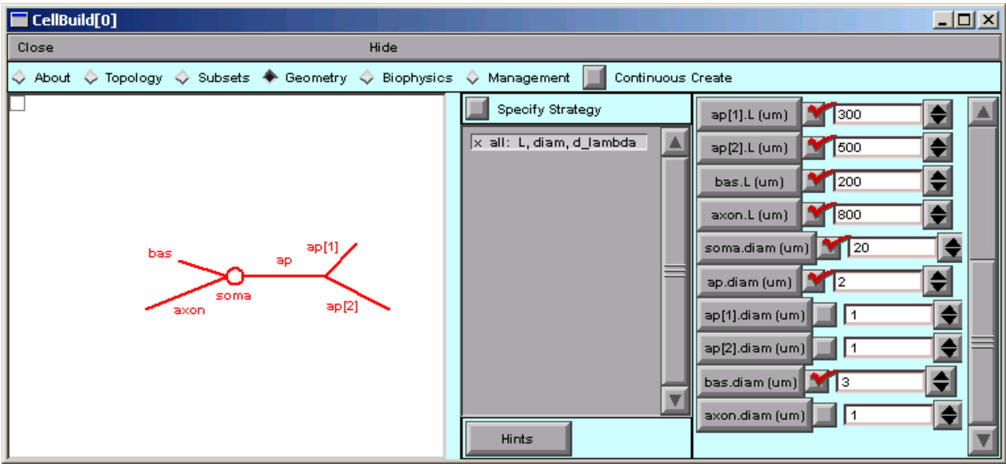
Implementing geometry strategy



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.

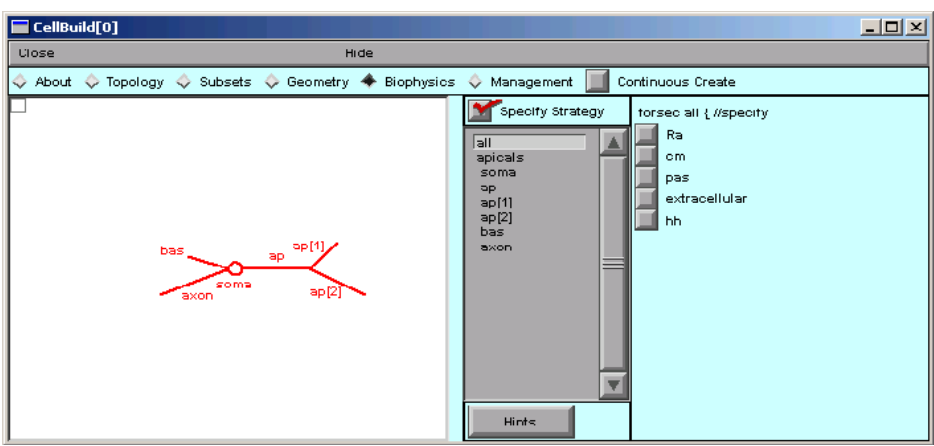
Implementing geometry *continued*



Set L and diam for all sections.

Time to save to a session file!

Biophysics

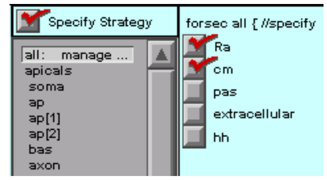


"Specify Strategy" is ON.

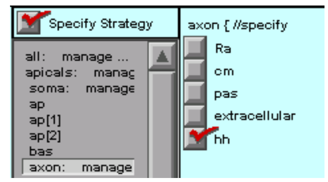
Base the plan on shared properties.

Biophysics strategy

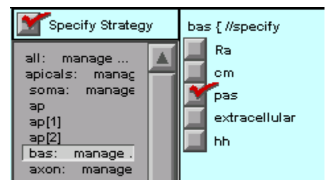
Ra and cm are homogeneous



apicals, soma and axon have hh

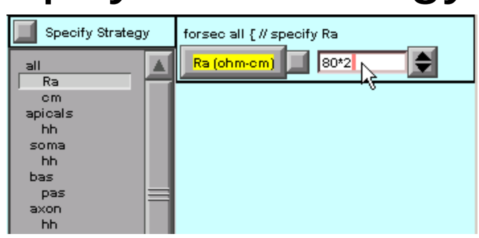


bas has pas

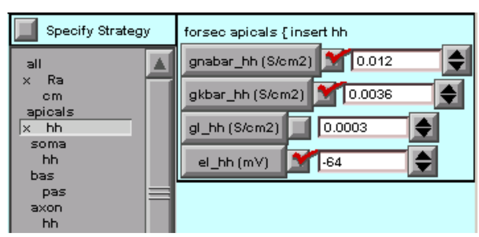


Implementing biophysics strategy

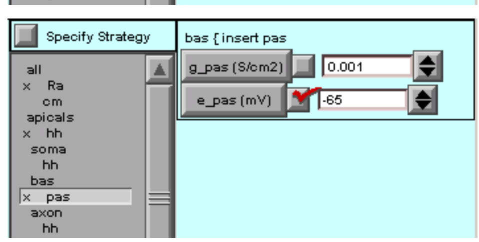
Double Ra



Fix apicals hh params



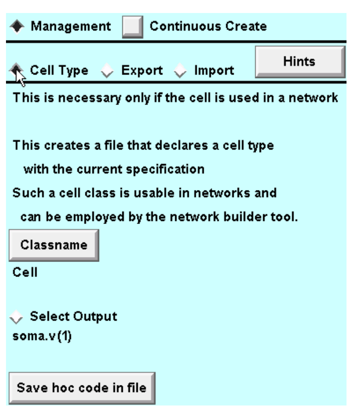
Shift e_pas in bas



Save another session file!!

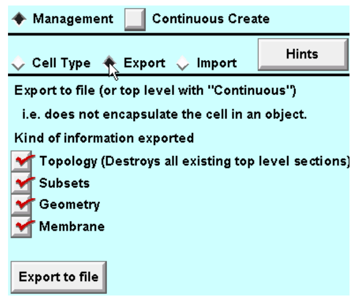
Management

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



Management *continued*

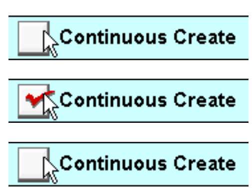
Option 2: save as hoc file



Management *continued*

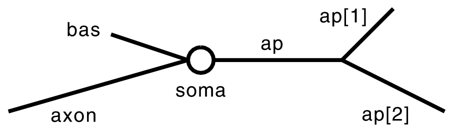
Option 3: export to interpreter

Toggle Continuous Create ON and OFF



or just leave it ON all the time.

Step 2: creating and using an interface for running simulations



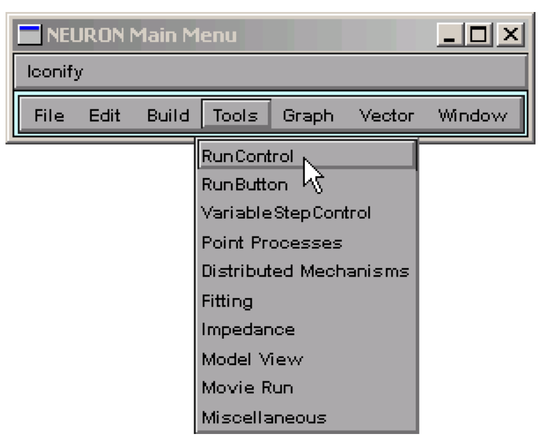
We want to

- 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

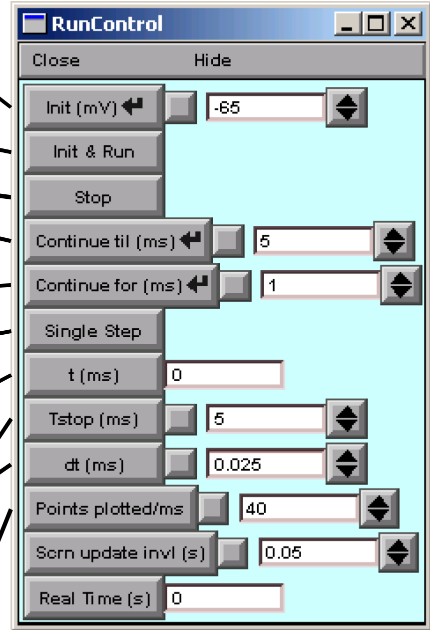
Single step advances by 1/(**Points plotted/ms**)

t numeric field shows model time

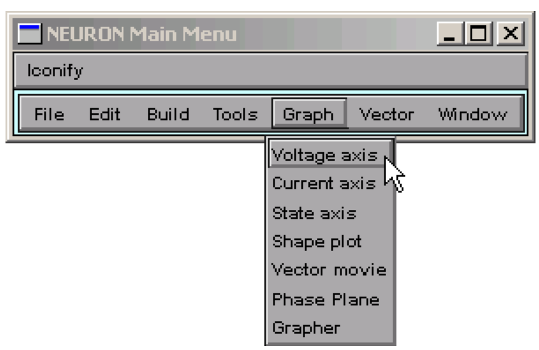
Tstop specifies when simulation ends

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

Points plotted/ms is plotting interval

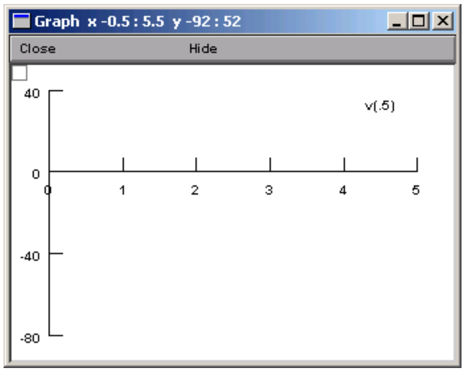


We need to plot Vm(t) at soma



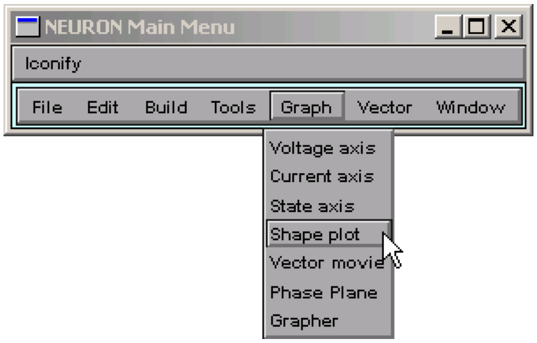
NEURON Main Menu / Graph / Voltage axis

Graph window



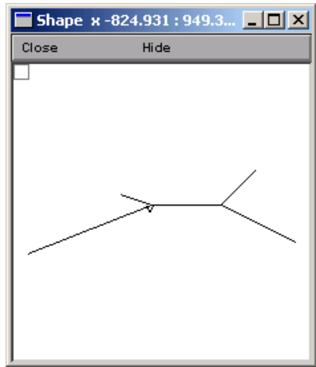
v(.5) is Vm at middle of default section
(soma in this example)

We need to plot Vm along a path



NEURON Main Menu / Graph / Shape plot

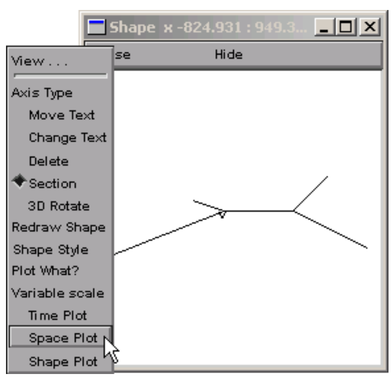
Bringing up a space plot



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

Click on its "menu box" . . .

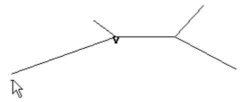
Bringing up a space plot *continued*



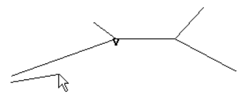
. . . and scroll down to "Space Plot".

Bringing up a space plot *continued*

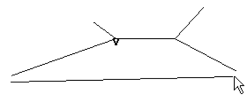
Click just left of the shape



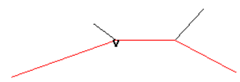
Hold button down while dragging
from left . . .



. . . to right . . .

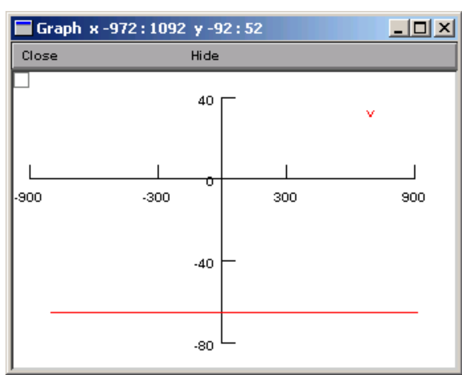


. . . then release button.



This pops up a . . .

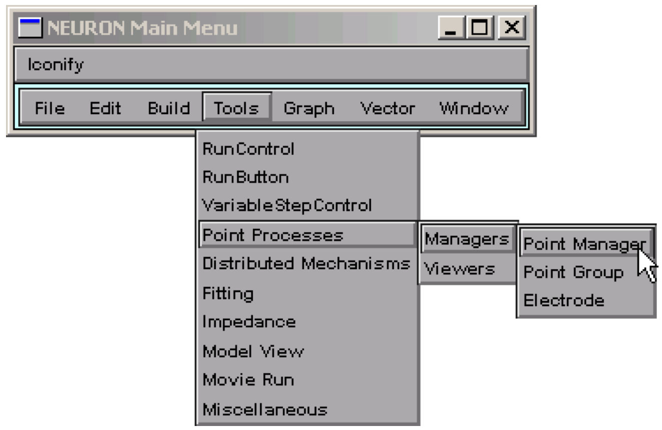
Space plot



A plot of Vm vs. distance along a path.

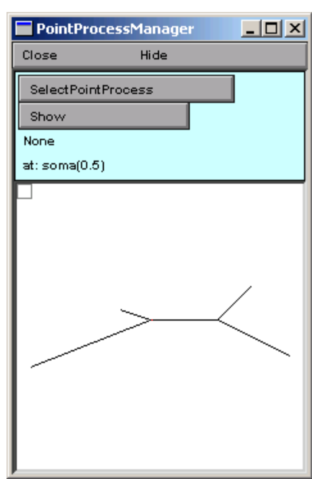
Better save a session file.

We need a stimulator



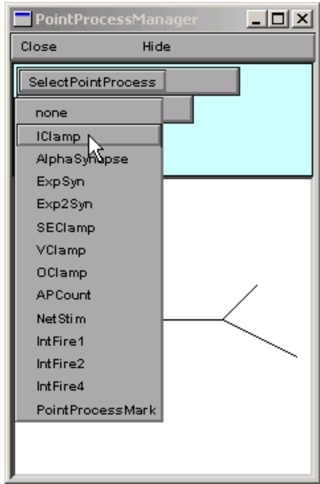
NEURON Main Menu / Tools / Point Processes / Managers / Point Manager

PointProcessManager window



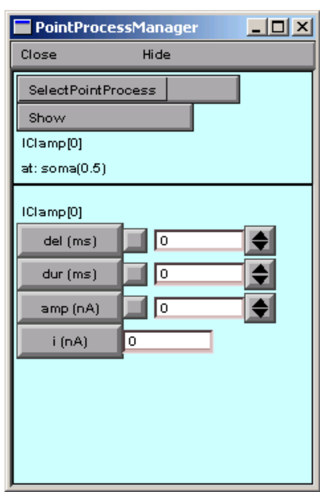
To make this an IClamp . . .

Creating an IClamp



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

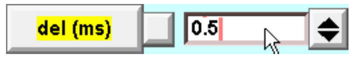
IClamp parameter panel



Next: set parameter values.

Entering values into numeric fields

Direct entry

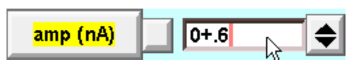


Note yellow highlight on button

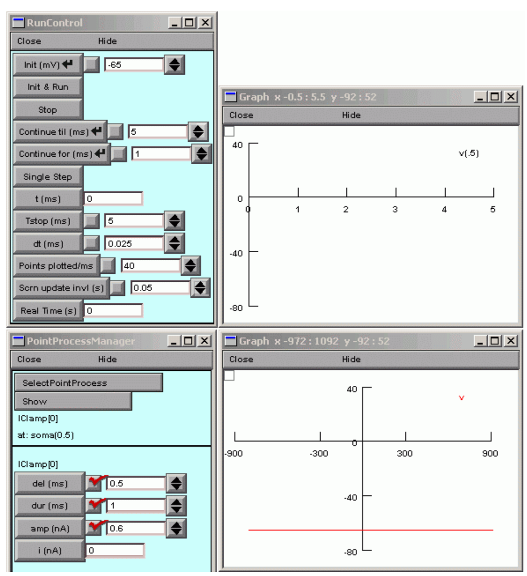
Spinner

Red check means value has been changed from default

Mathematical expression

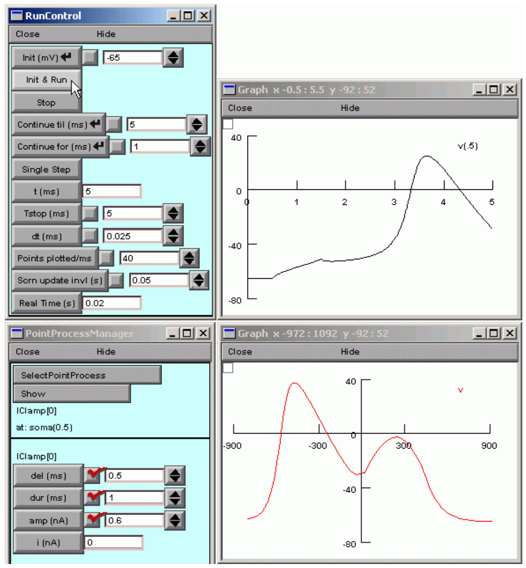


Our user interface

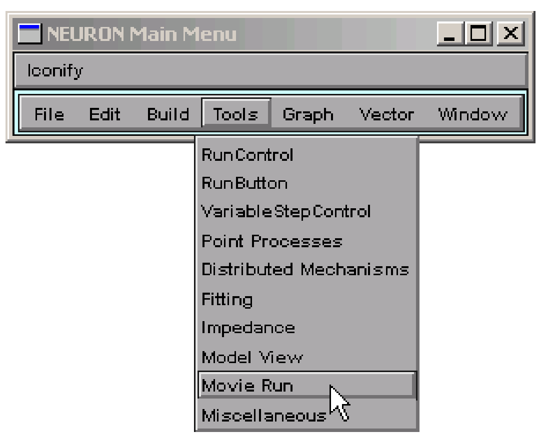


Time to save to a new session file!

It works!



How to get nice space plot "movies"

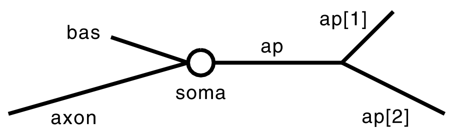


NEURON Main Menu / Tools / Movie Run

Space plot "movies" *continued*

Movie Run / Init & Run

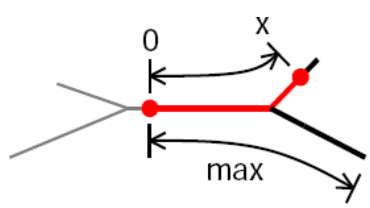
What if hh is nonuniform over the apicals?



Suppose gnabar_hh, gkbar_hh, and gl_hh
all derease linearly with distance
from the origin of the apical tree.

Details:

1. All have full density at origin of apical tree.
2. Density falls to 0% at most the most distant termination.
3. For uniform -65 mV resting potential, el_hh = -54.3 mV.



This example:

gnabar_hh = 0.12 * (1 - p) where p = L_{0x}/L_{max}
(normalized path distance from location x
to origin 0 of apical tree)

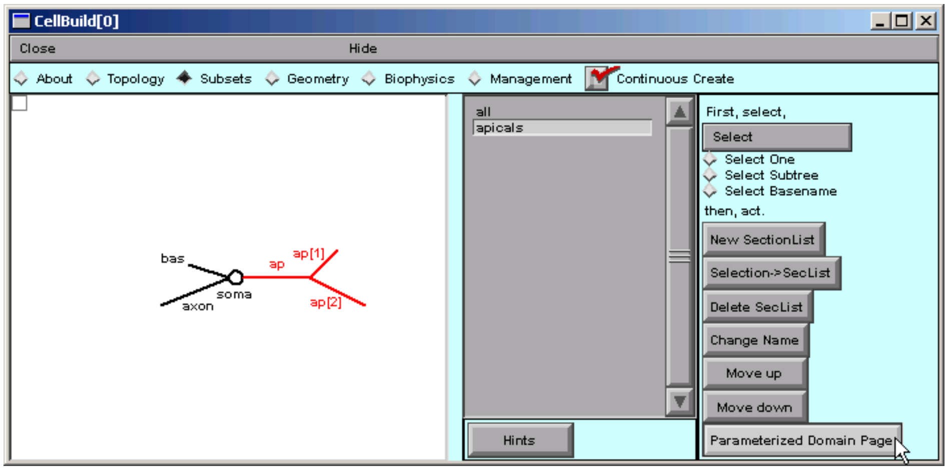
The general task: param = f(p), where f can be any function
and p is one of these "distance metrics":

- path length from a reference point
- radial distance from a reference point
- distance from a plane ("3D projection onto a line")

An equivalent hoc idiom:

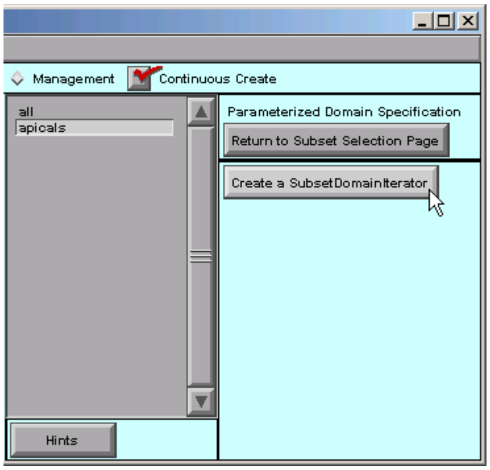
forsec subset for (x,0) { rangevar_suffix(x) = f(p(x)) }

Setting up a SubsetDomainIterator



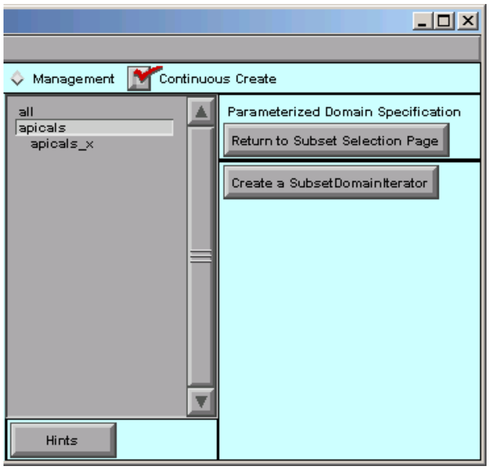
Select a subset, then click on "Parameterized Domain Page"

SubsetDomainIterator *continued*



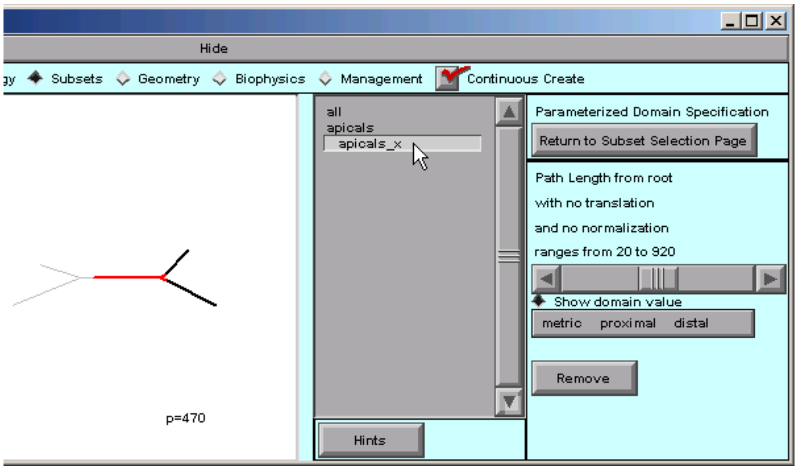
Click on "Create a SubsetDomainIterator"

SubsetDomainIterator *continued*



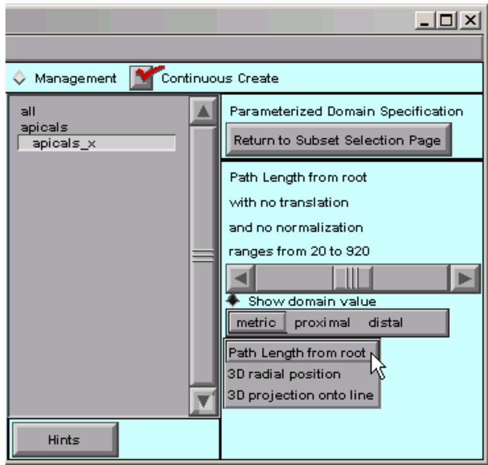
Note "apicals_x" in middle panel.
Click on it . . .

SubsetDomainIterator *continued*



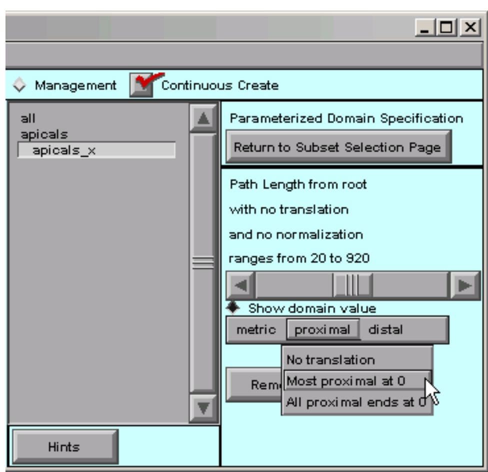
. . . to see controls for specifying the distance metric.

SubsetDomainIterator *continued*



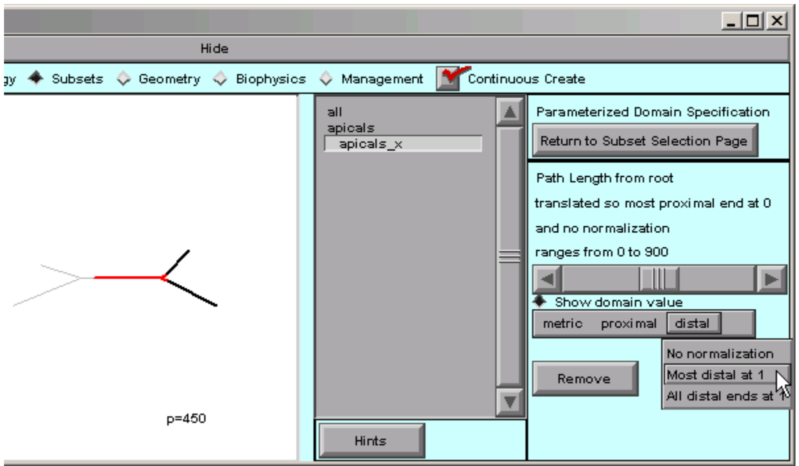
"metric" offers the three basic choices

SubsetDomainIterator *continued*



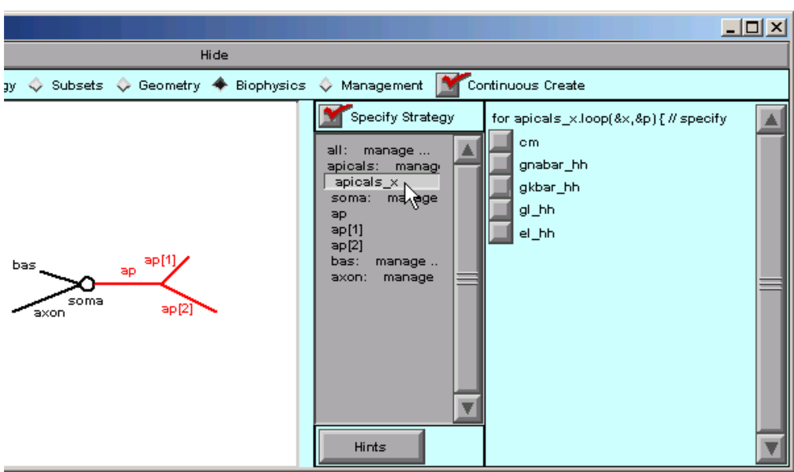
proximal / Most proximal at 0
makes distance start at root of apical tree

SubsetDomainIterator *continued*



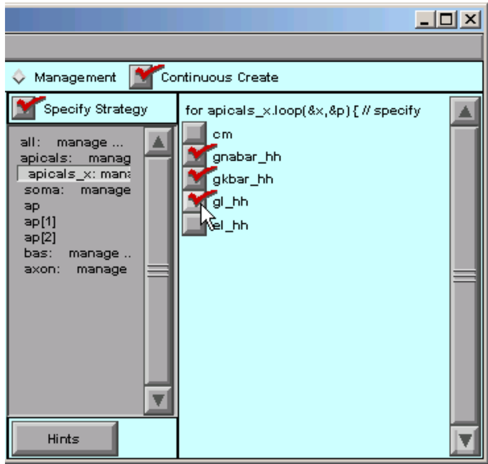
distal / Most distal at 1
finishes "normalization" of distance

Back to Biophysics Strategy



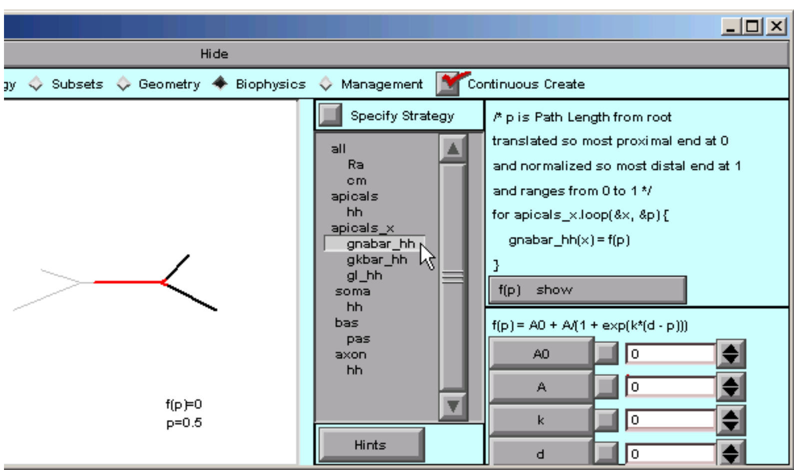
Click on apicals_x,
then select the parameters it will control.

Biophysics Strategy *continued*



We want gnabar_hh, gkbar_hh, and gl_hh to be inhomogeneous.

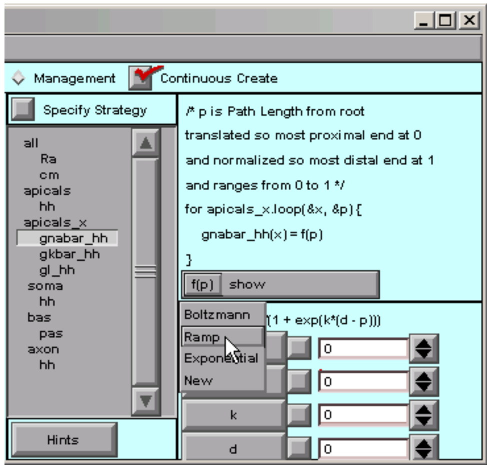
Implement the strategy



Click on one of the inhomogeneous parameters.

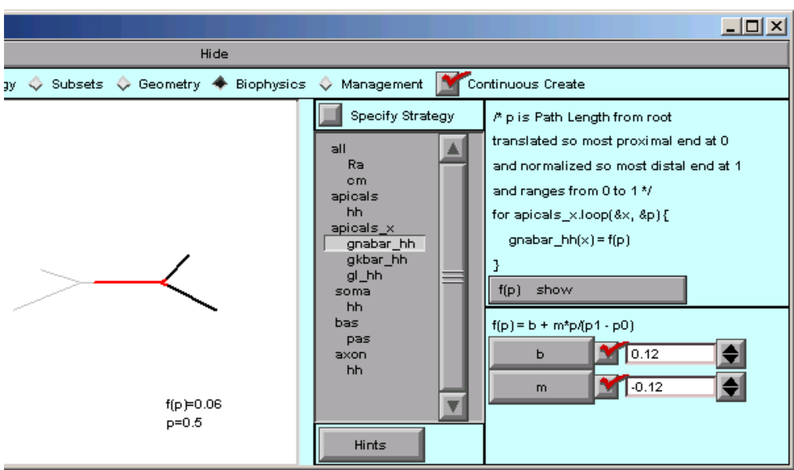
Note that default f( ) is Boltzmann.

Implement the strategy *continued*



f(p) / Ramp
selects linear function

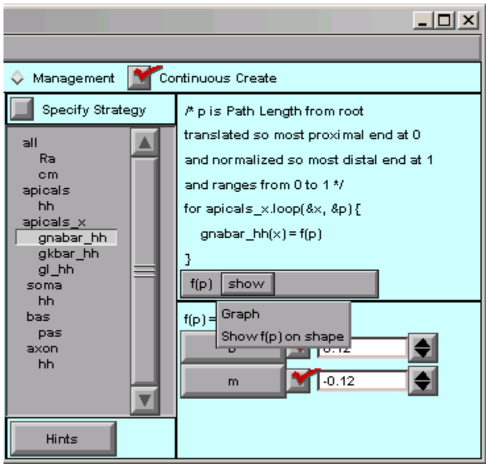
Implement the strategy *continued*



After setting intercept b and slope m for gnabar_hh

Save another session file!!

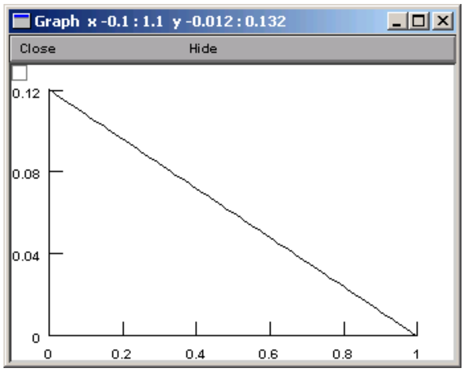
Verify the implementation



show / graph

show / Show f(p) on shape

Verify the implementation *continued*



"show / graph" results:

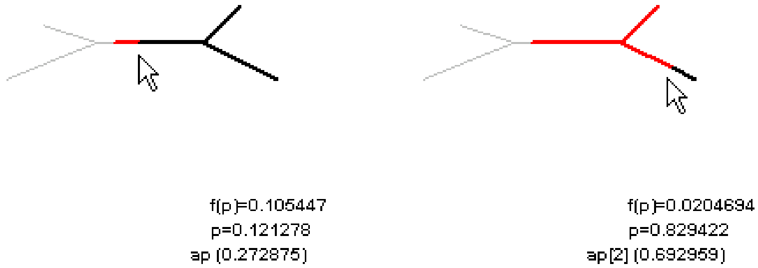
x axis: normalized distance from origin of apicals

y axis: gnabar_hh

Verify the implementation *method 2*

1. show / Show f(p) on shape
2. Click next to shape and drag . . .

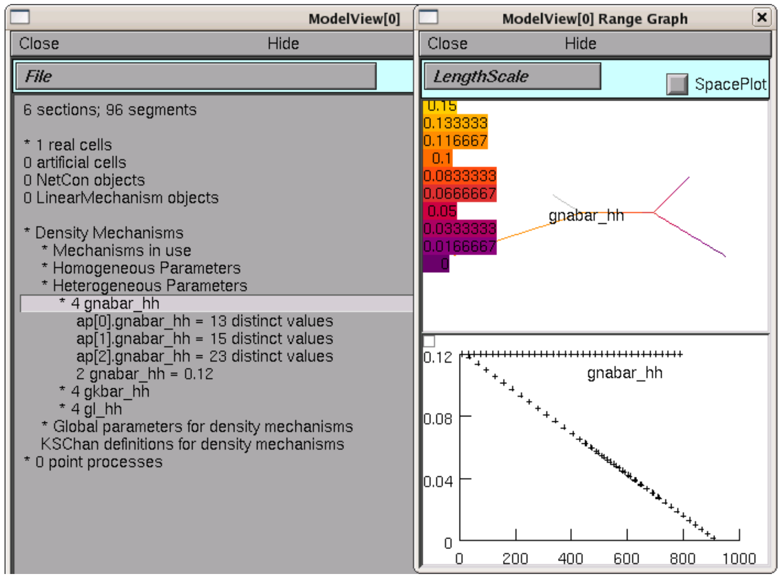
. . . from left . . .   . . . to right . . .



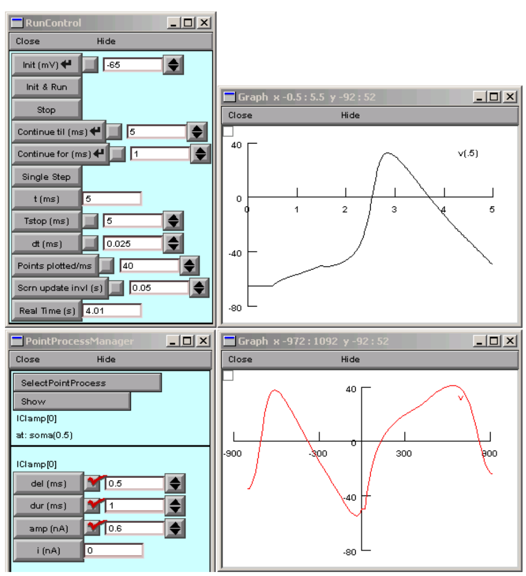
. . . while watching the values of p and f(p)

Verify the implementation *method 3*

NEURON Main Menu / Tools / Model View



A simulation with the revised model



The Channel Builder

Voltage- and ligand-gated channels

Kinetic schemes, HH-style differential equations

Optional stochastic gating mode for point processes

Faster than equivalent NMODL mechanisms

Much easier to use than writing NMODL code

Limited to channels
- NMODL needed for pumps, buffers, diffusion, event-driven synaptic mechanisms, artificial spiking cells

Tutorial: see Documentation at NEURON's home page
- http://www.neuron.yale.edu/

Conceptualize the task

Ion selectivity	
I/V relationship	ohmic / GHK (constant field)
Description of dynamics	HH style / kinetic scheme
Gates	independent identical subunits fractional openness
Sensitivity	voltage / ligand
Transition style	alpha, beta / inf, tau functions / tables

Implementing the HH i_{Na} with the Channel Builder

$i_{Na} = g_{Na} (V - E_{Na})$ where

$g_{Na} = gbar_{Na}\ m^3h$

$gbar_{Na} = 0.12\ S/cm^2$

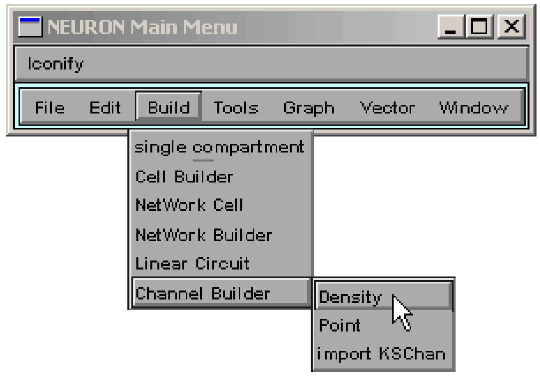
m and h are described by DEs of the form

dx/dt = alpha (1 - x) - beta x

How to proceed

1. Bring up a Channel Builder
2. Specify channel's basic properties
3. Specify channel gating
   - states
   - transitions (if a kinetic scheme)
   - effects of voltage and ligands

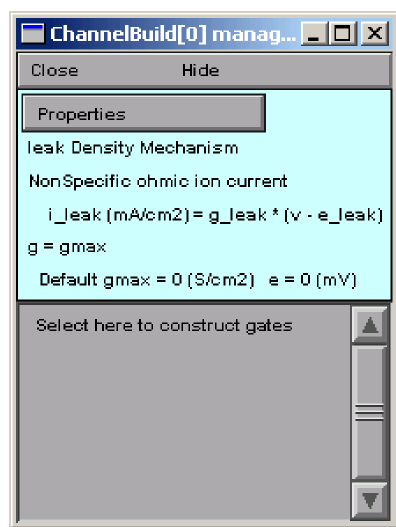
1. Bring up a Channel Builder



NEURON Main Menu / Build
/ Channel Builder / Density

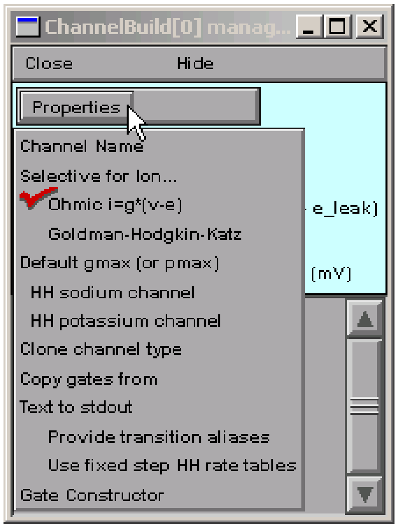
The Channel Builder

We need to change its name,
ion selectivity,
default conductance,
and equilibrium potential



2. Specify channel's basic properties

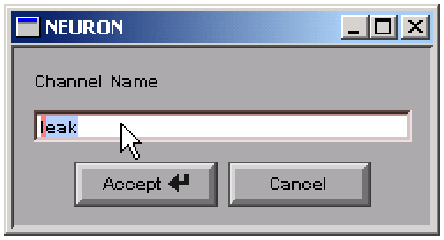
Click on Properties,
then select item to change

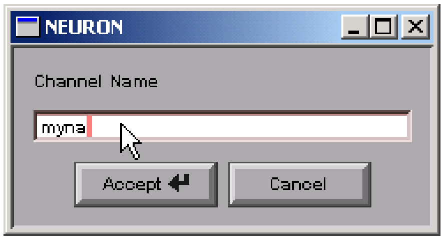


Name

Properties / Channel Name

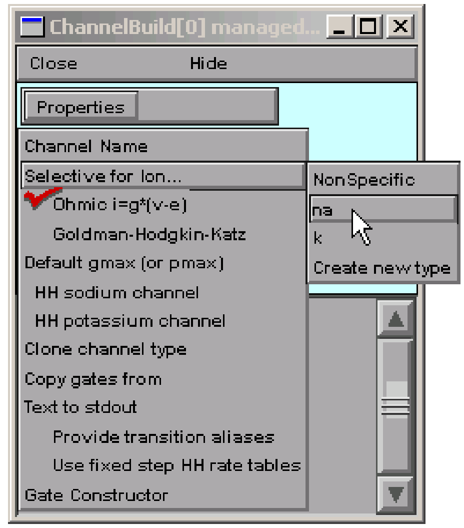
Then change leak to myna





Ion selectivity

Properties
/ Selective for Ion... / na



Default conductance and equilibrium potential

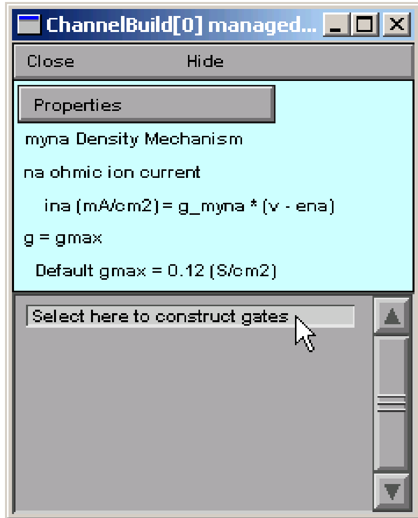
Properties / Default gmax

Specify 0.12 S/cm2

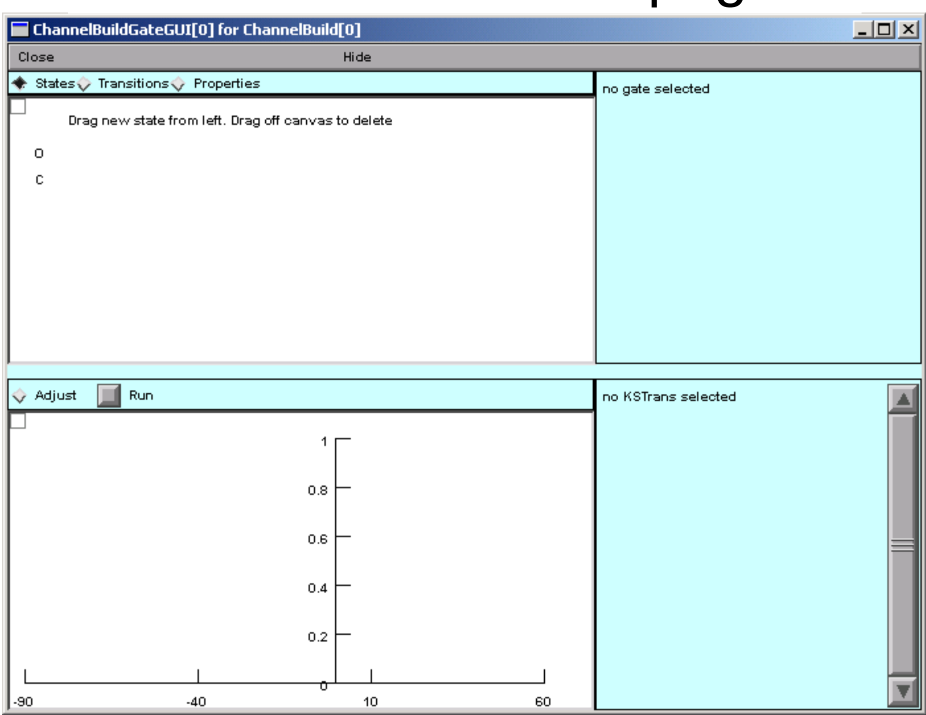
Equilibrium potential:
na has its own ena,
so nothing to do!

3. Specify channel gating

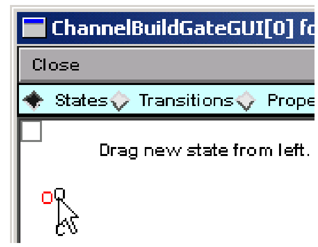
"Select here to construct gates"



"GateGUI": States page



Spawn states



Click and drag O ("open")
from palette . . .

. . . to canvas.

Repeat for C ("closed")

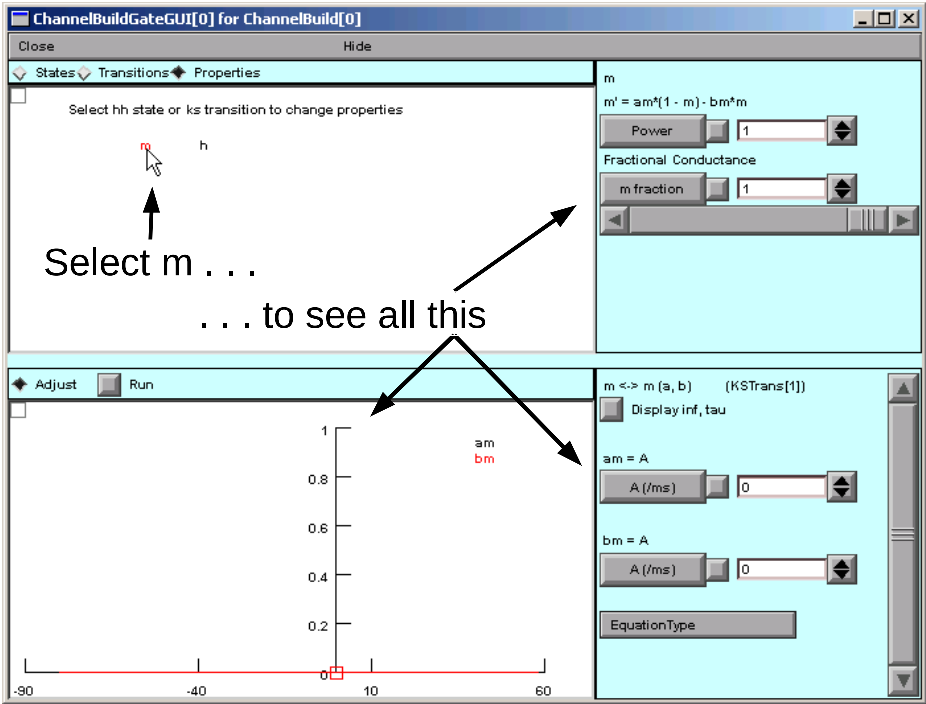
Rename states

Click O without dragging

Change to m

Change C to h

"GateGUI": Properties page

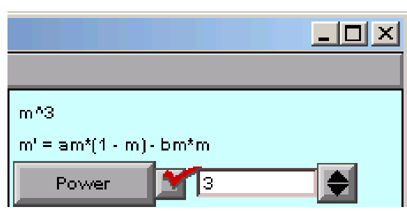


Select m . . .

. . . to see all this

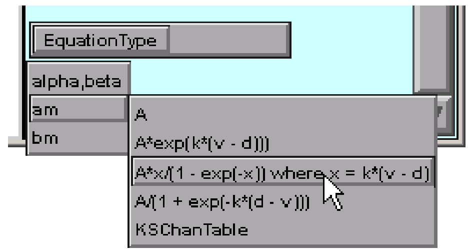
Set m exponent

Change Power to 3



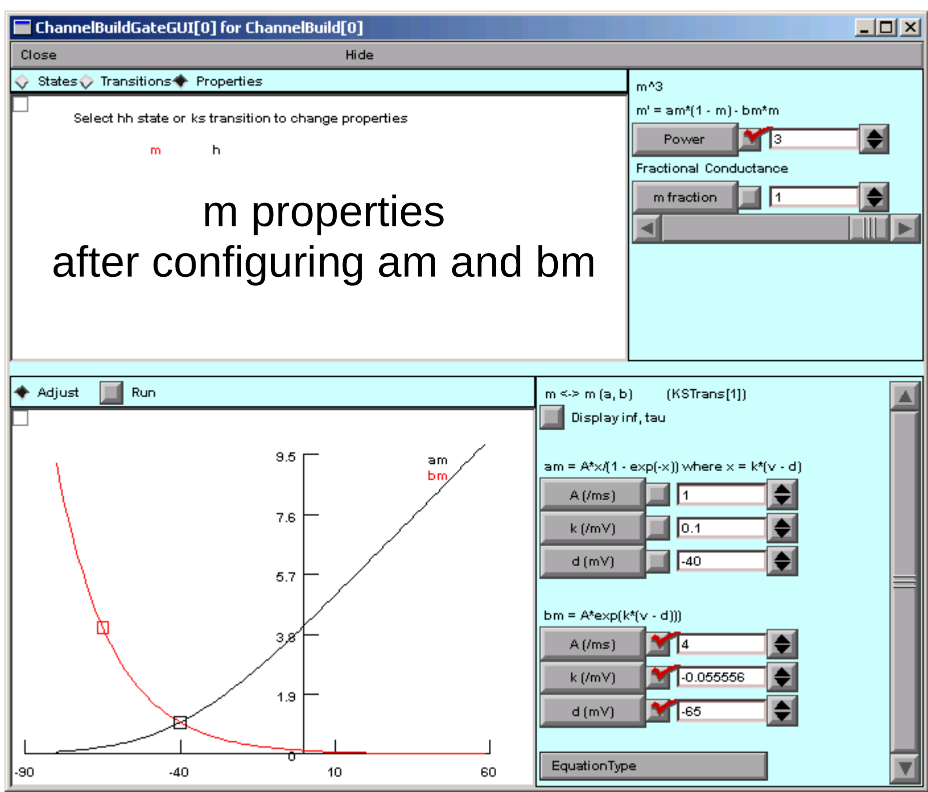
Specify voltage dependence of am and bm

Choose functional form for am

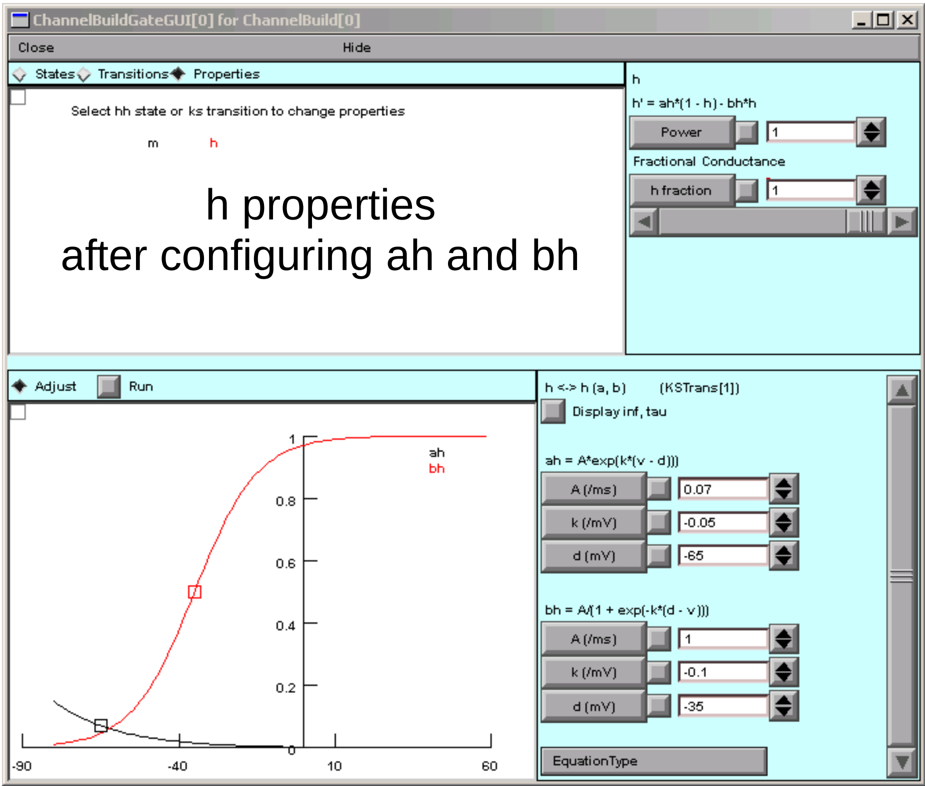


Set parameter values

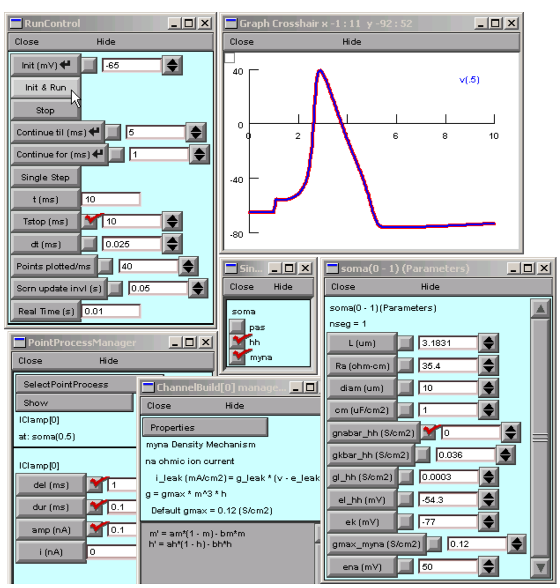
Do same for bm



m properties
after configuring am and bm


ChannelBuildGateGUI[0] for ChannelBuild[0]
Close
Hide
States
Transitions
Properties
Select hh state or ks transition to change properties
m
h
h properties
after configuring ah and bh
h
h' = ah*(1 - h) - bh*h
Power
1
Fractional Conductance
h fraction
1
Adjust
Run
h <-> h (a, b) (KSTrans[1])
Display inf, tau
ah = A*exp(k*(v - d))
A (/ms)
0.07
k (/mV)
-0.05
d (mV)
-65
bh = A/(1 + exp(-k*(d - v)))
A (/ms)
1
k (/mV)
-0.1
d (mV)
-35
EquationType
ah
bh
1
0.8
0.6
0.4
0.2
0
-90
-40
10
60

Testing


RunControl
Init (mV)
-65
Init & Run
Stop
Continue til (ms)
5
Continue for (ms)
1
Single Step
t (ms)
10
Tstop (ms)
10
dt (ms)
0.025
Points plotted/ms
40
Scrn update invl (s)
0.05
Real Time (s)
0.01
Graph
v(.5)
40
0
-40
-80
0
2
6
8
10
PointProcessManager
SelectPointProcess
Show
IClamp[0]
at: soma(0.5)
IClamp[0]
del (ms)
1
dur (ms)
0.1
amp (nA)
0.1
i (nA)
0
soma
pas
hh
myna
ChannelBuild[0] manage
Properties
myna Density Mechanism
na ohmic ion current
i_leak (mA/cm2) = g_leak * (v - e_leak)
g = gmax * m^3 * h
Default gmax = 0.12 (S/cm2)
m' = am*(1 - m) - bm*m
h' = ah*(1 - h) - bh*h
soma(0 - 1) (Parameters)
nseg = 1
L (um)
3.1831
Ra (ohm-cm)
35.4
diam (um)
10
cm (uF/cm2)
1
gnabar_hh (S/cm2)
0
gkbar_hh (S/cm2)
0.036
gl_hh (S/cm2)
0.0003
el_hh (mV)
-54.3
ek (mV)
-77
gmax_myna (S/cm2)
0.12
ena (mV)
50

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
}
```

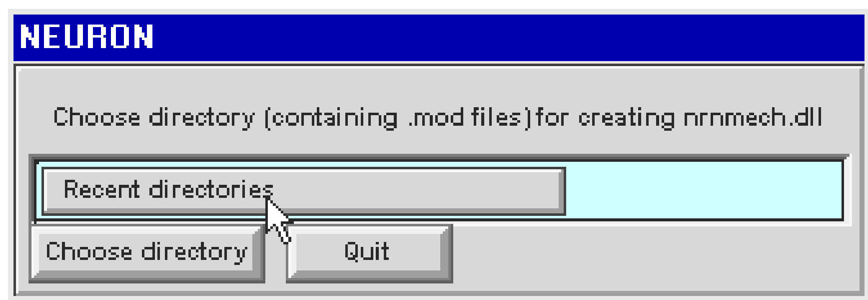
Functions and procedures

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

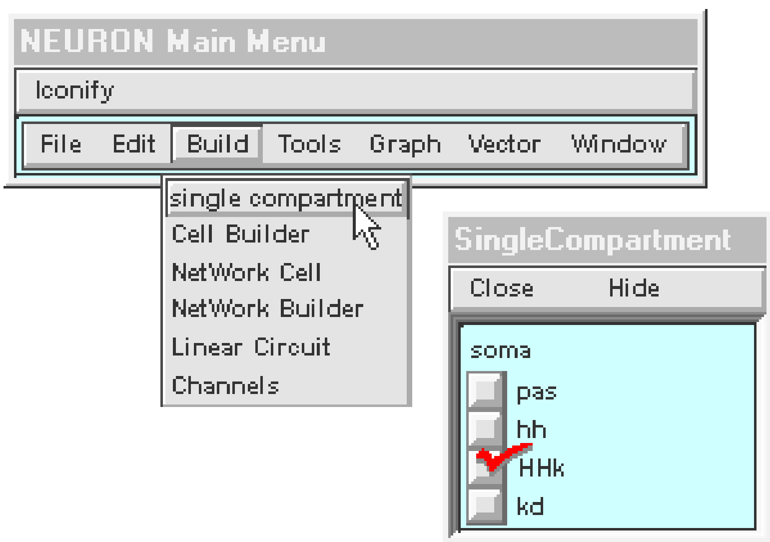
UNIX

```
nrnivmodl
nrngui
```

MSWIN



Select NEURON Main Menu / Build / single compartment



Density mechanism / Point Process

NMODL

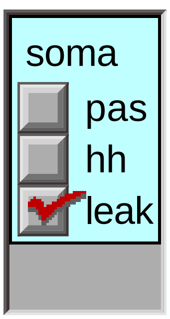
Density mechanism:

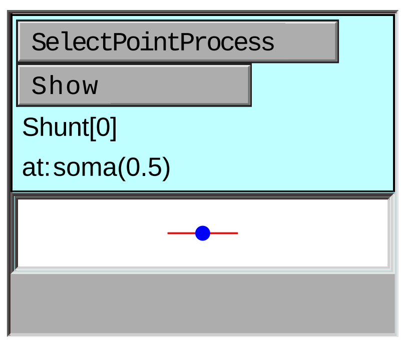
```
NEURON {
    SUFFIX leak
    NONSPECIFIC_CURRENT i
    RANGE i, e, g
}

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
}
```

GUI





Interpreter

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

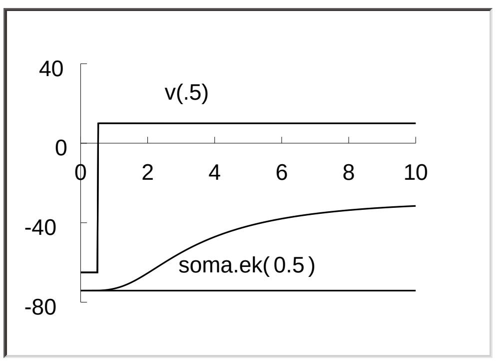
```
objref s
soma s = new Shunt(.5)
s.r = 2
```

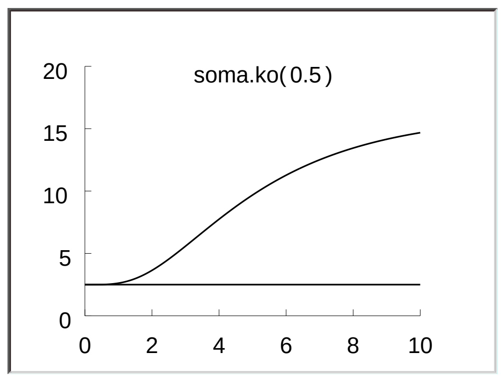
Ion Channel

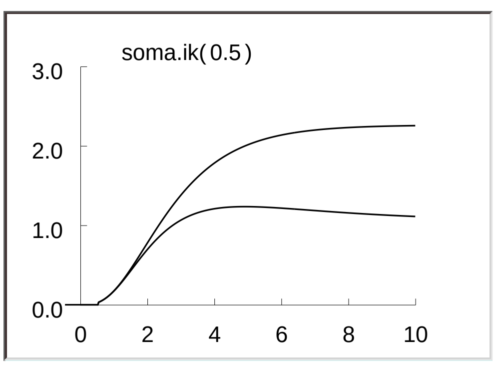
```
NEURON {
  USEION k READ ek WRITE ik
}
BREAKPOINT {
  SOLVE states METHOD cnexp
  ik = gbar*n*n*n*n*(v - ek)
}
DERIVATIVE states {
  rate(v*1(/mV))
  n' = (inf - n)/tau
}
```

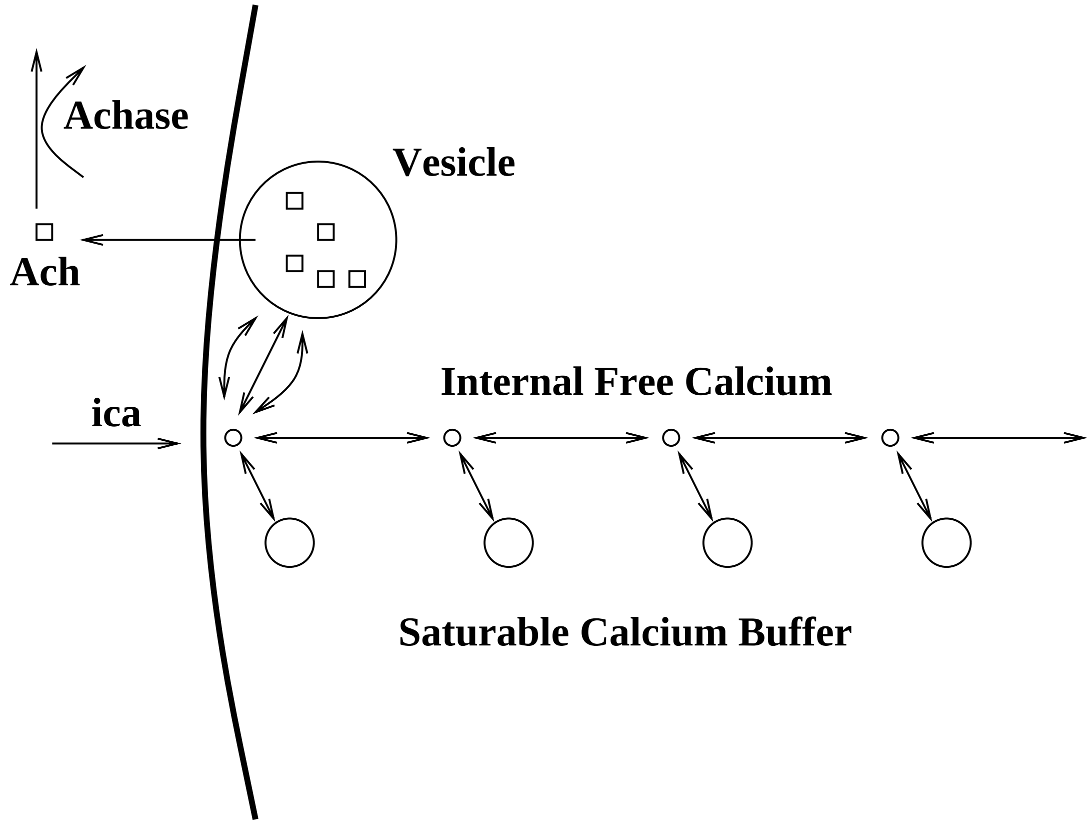
Ion Accumulation

```
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.
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)
```

The Linear Circuit Builder

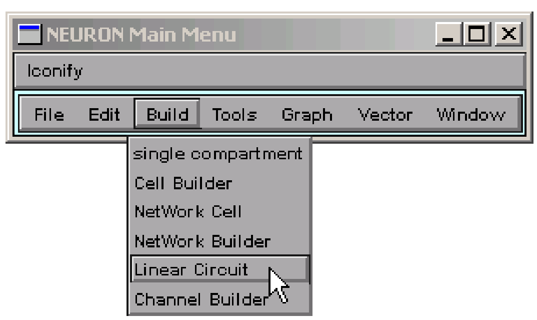
For building models that have linear circuit elements and may also involve neurons

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

Potential applications include

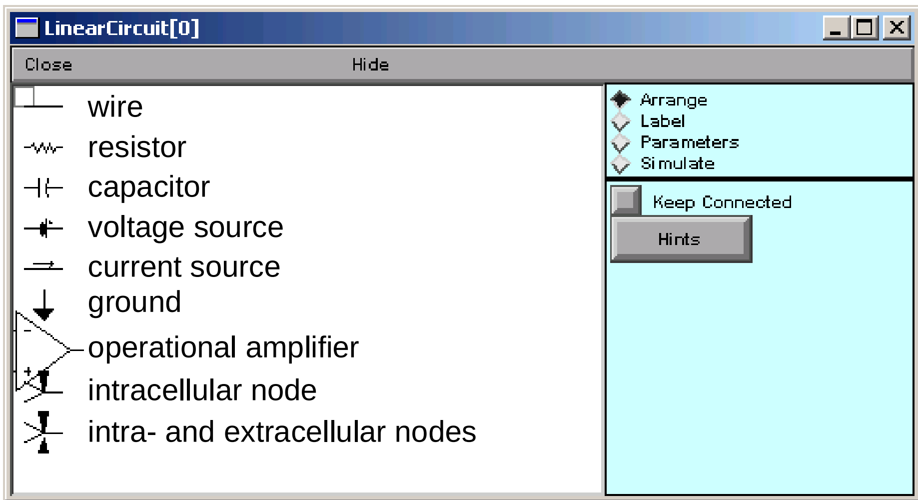
- effects and compensation of electrode R & C
- two-electrode voltage clamp
- ohmic and nonlinear gap junctions

1. Bring up a Linear Circuit Builder



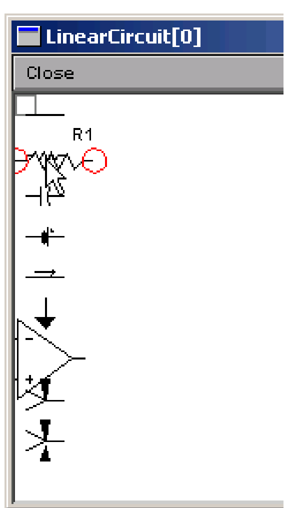
NEURON Main Menu / Build / Linear Circuit

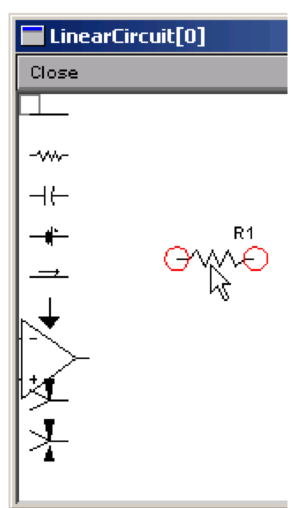
The Linear Circuit Builder



Arrange: spawn components

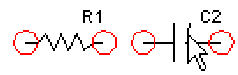
Click on palette and drag onto canvas



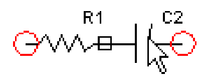


Arrange: connect components

Click and drag to
overlap red circles



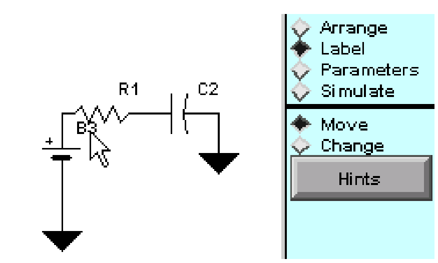
Black square is
"solder joint"

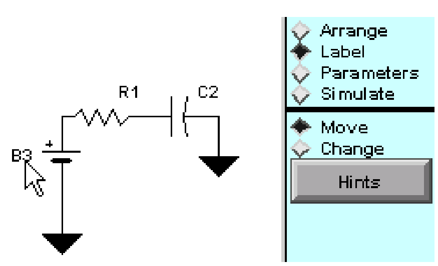


Pull apart to break connection

Label: move labels

Click and drag
to new location

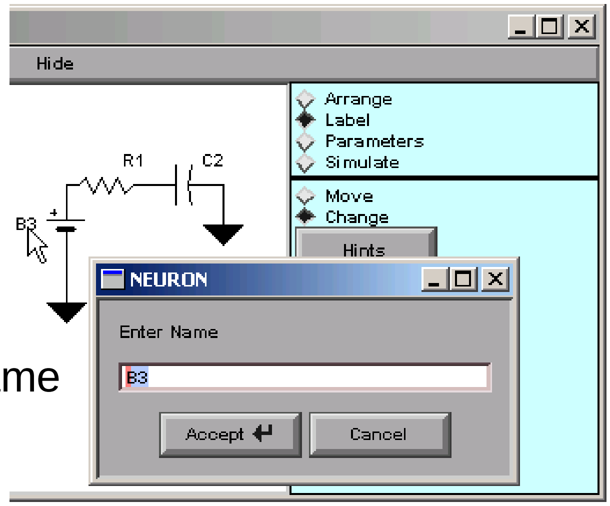




Label: change labels 1

Click on a label . . .

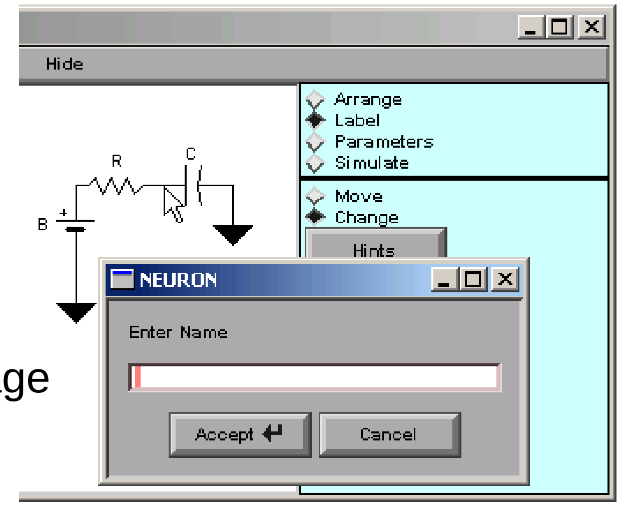
. . . to change its name



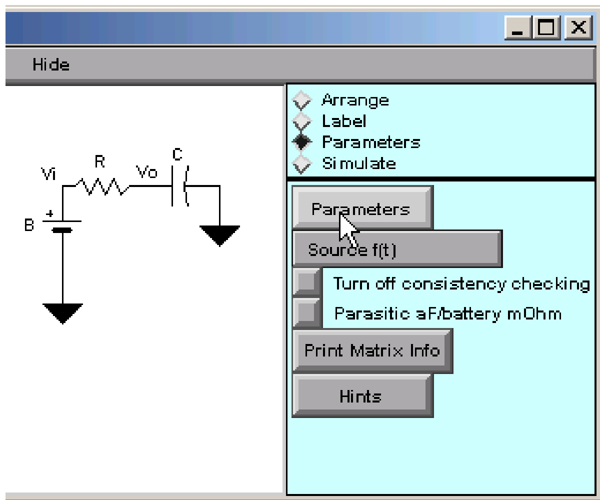
Label: change labels 2

Click on a node . . .

. . . to label a voltage



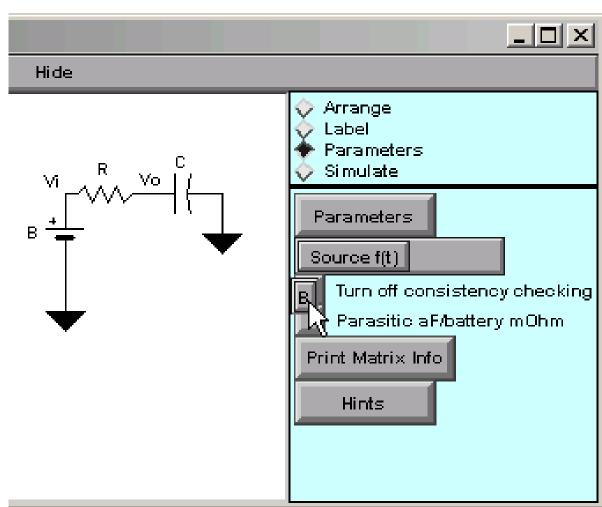
Parameters: non-source elements



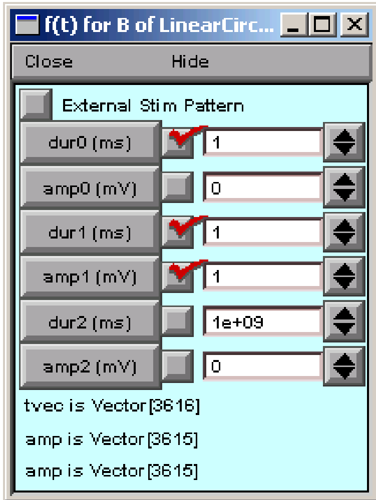
Click on
"Parameters"

Parameters: signal sources

Source f(t) / B



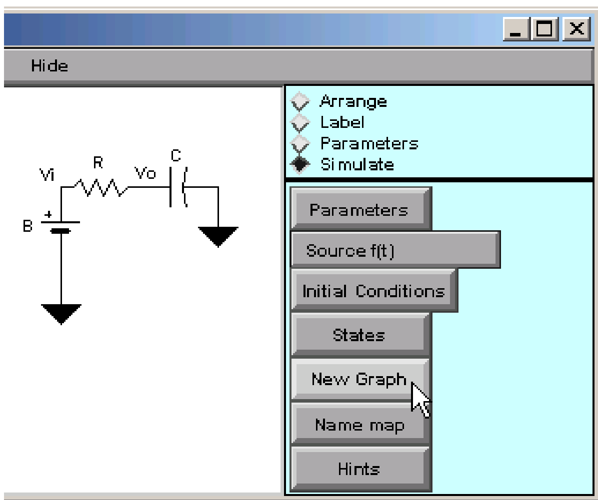
Parameters: signal sources *continued*



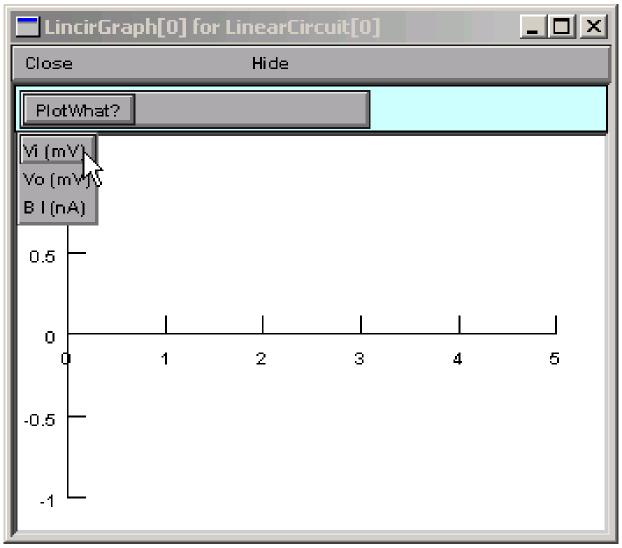
Configured

Simulate: creating a graph

New Graph

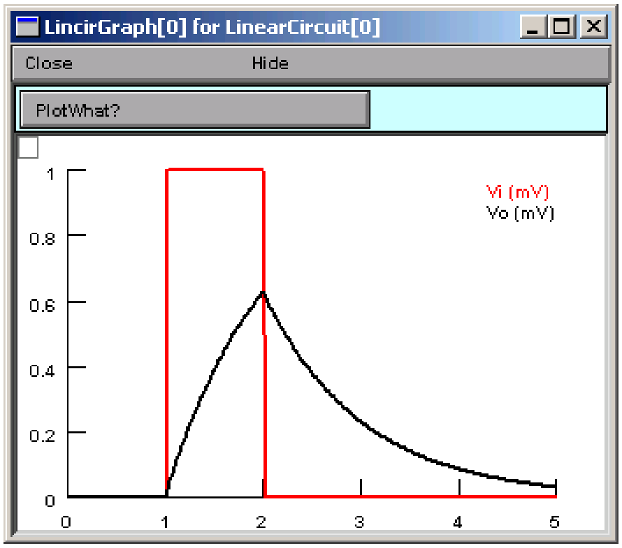


Simulate: specifying what to plot



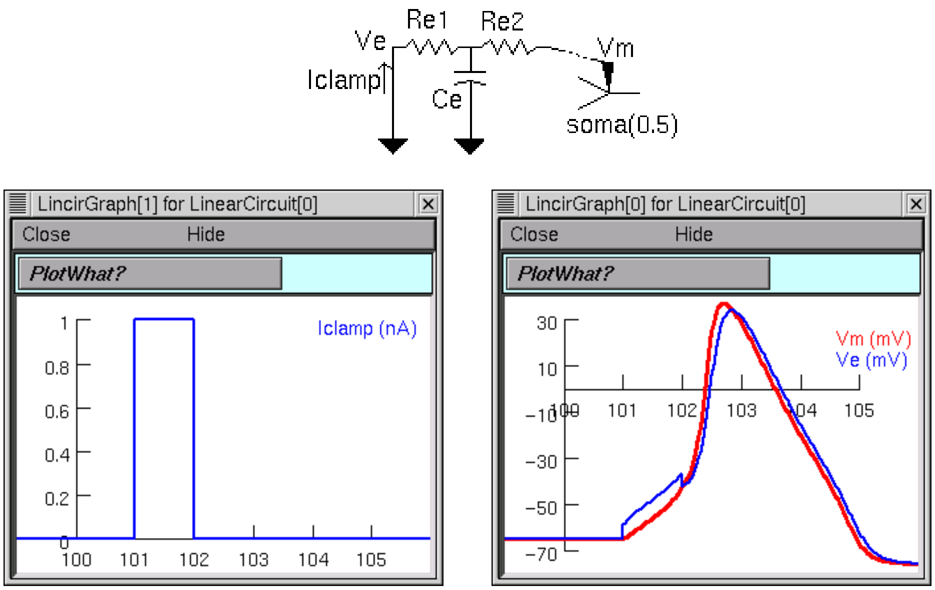
PlotWhat? / *variable_label*

Simulate: simulation results

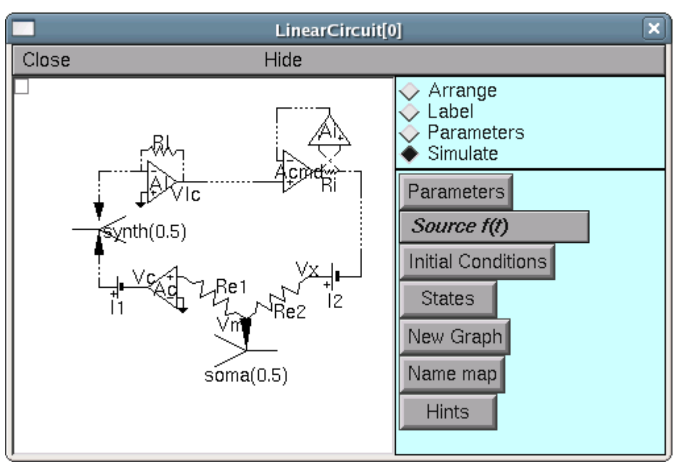


After minor cosmetic changes

Patch clamp with electrode R and C



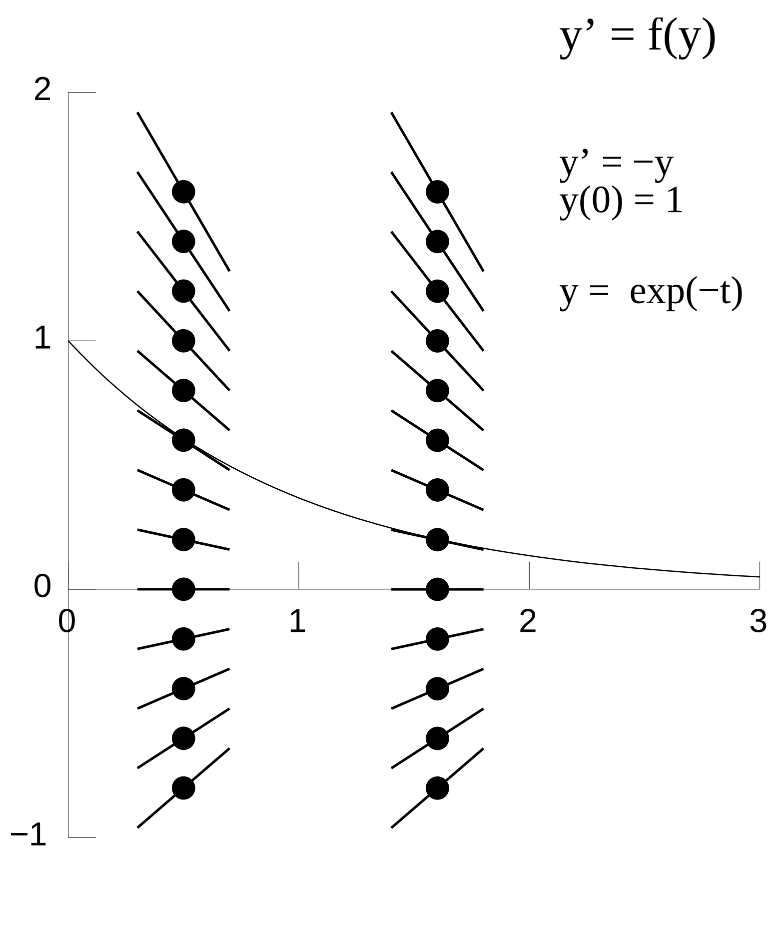
NEURON demo: dynamic clamp



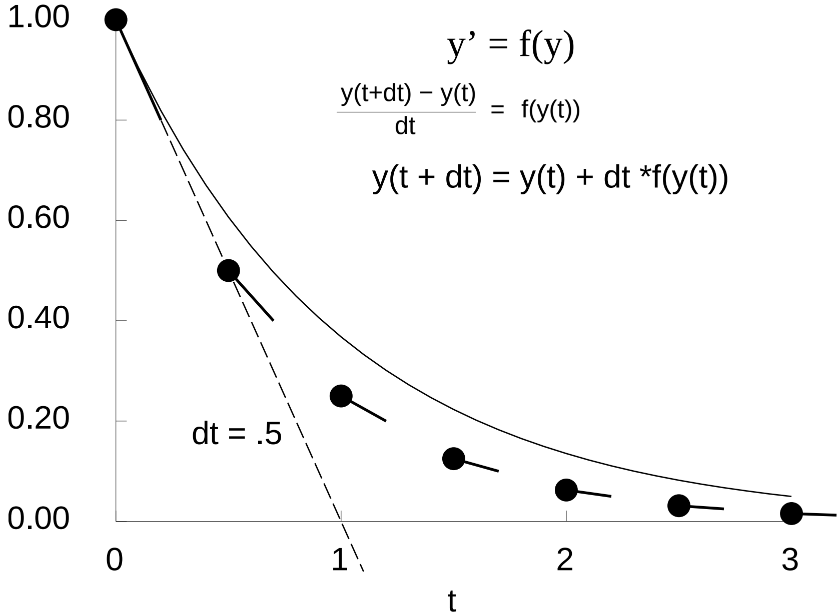
Compartmental Modeling

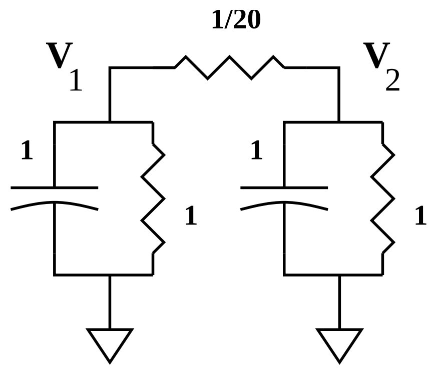
Not much mathematics required.

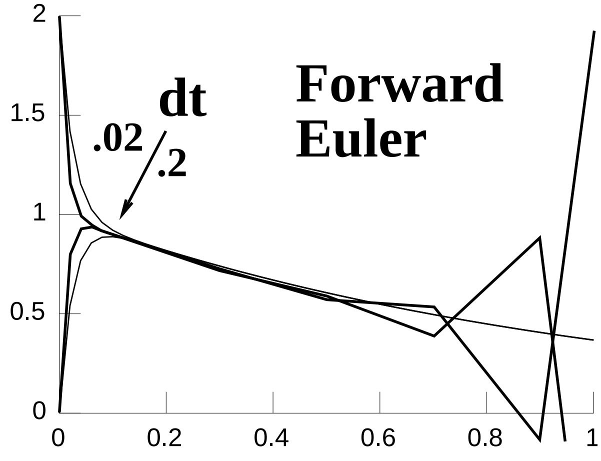
Good judgment essential!



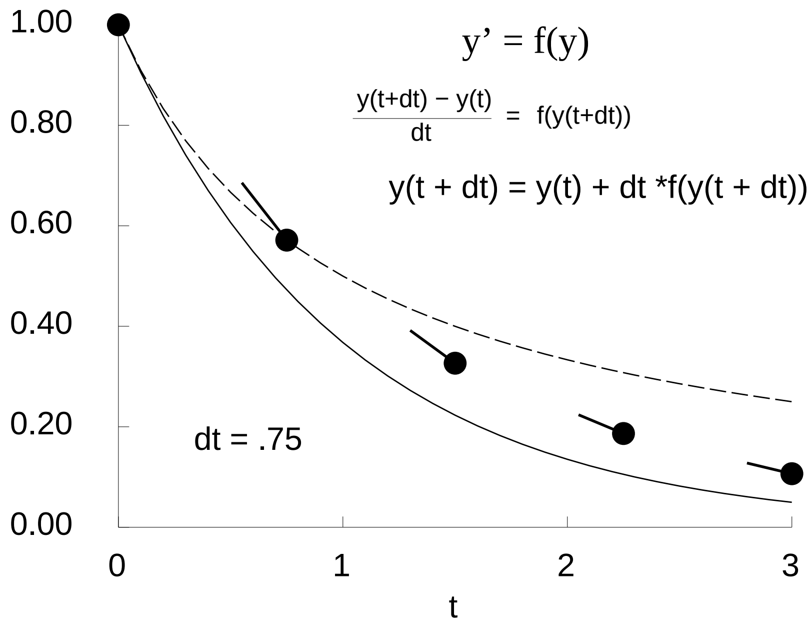
Forward Euler

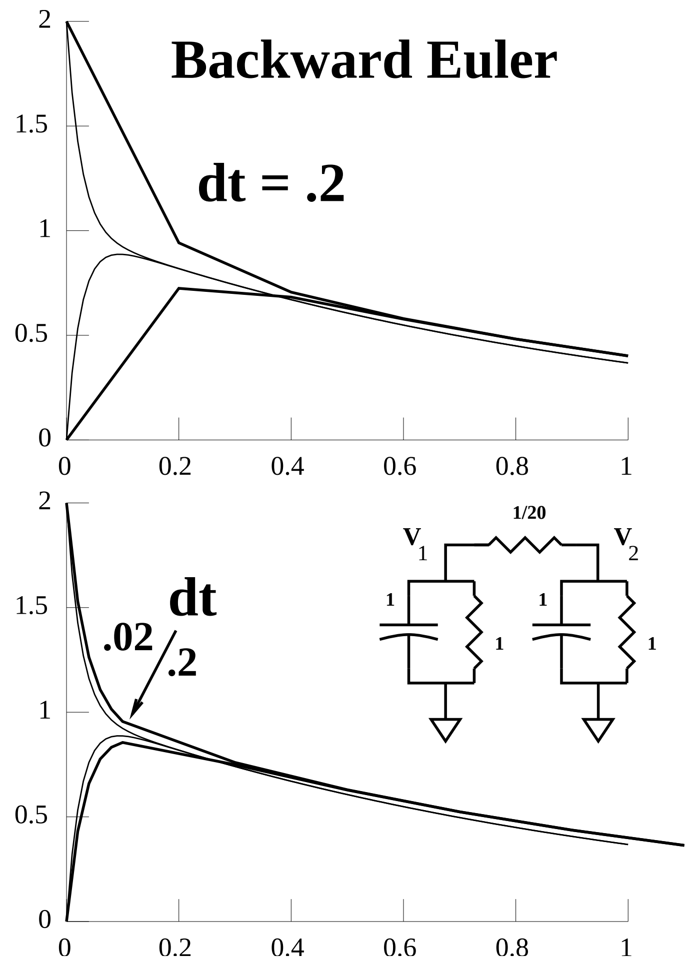




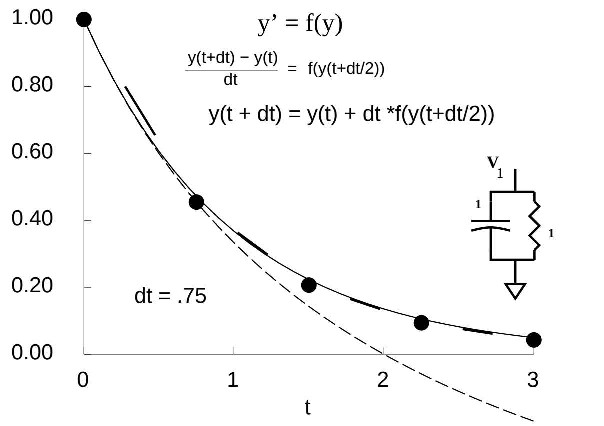


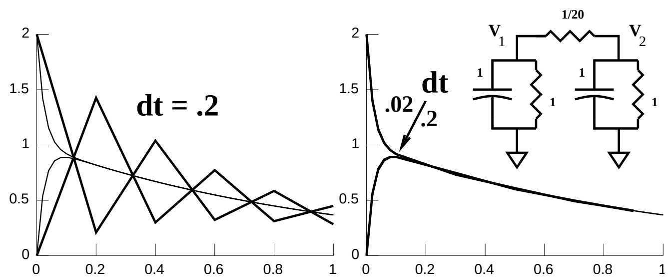
Backward Euler

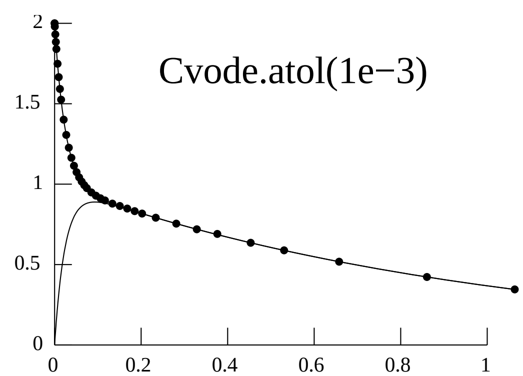


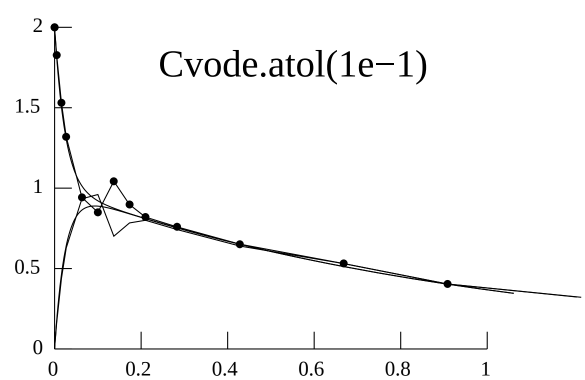


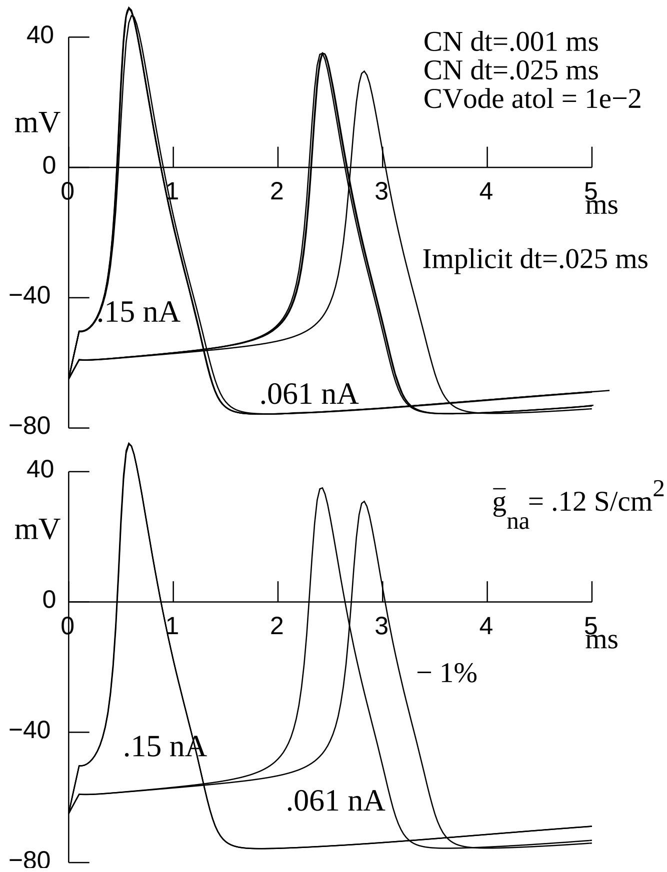
Crank−Nicholson










40
mV
0
−40
−80
0
1
2
3
4
5
ms
CN dt=.001 ms
CN dt=.025 ms
CVode atol = 1e−2
Implicit dt=.025 ms
.15 nA
.061 nA
40
mV
0
−40
−80
0
1
2
3
4
5
ms
$\bar{g}_{na}$ = .12 S/cm^2
− 1%
.15 nA
.061 nA

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

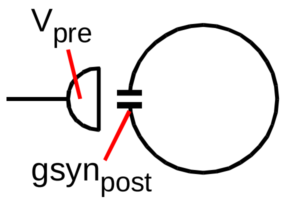
Synaptic transmission
- graded
- spike-triggered

Graded synaptic transmission

Physical system:

A presynaptic variable governs
continuous transmitter release

Transmitter modulates
a postsynaptic property



$gsyn_{post} = f(V_{pre})$

Problem: how does postsynaptic cell know V_{pre}?

Graded synaptic transmission *continued*

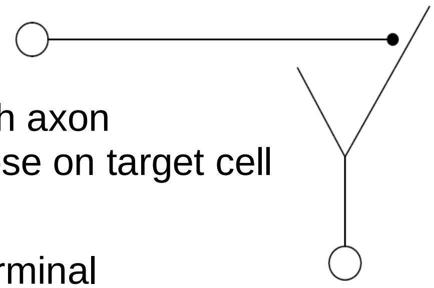
POINTER links postsynaptic variable
to presynaptic variable

```
NEURON {
   POINT_PROCESS Syn
   POINTER v_pre
}
```

hoc usage

```
objref syn
dend syn = new Syn(0.5)
setpointer syn.v_pre, precell.axon.v(1)
```

Spike-triggered synaptic transmission

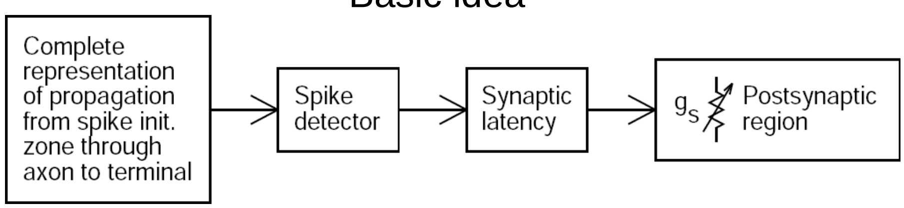


Physical system:
- Presynaptic neuron with axon that projects to synapse on target cell

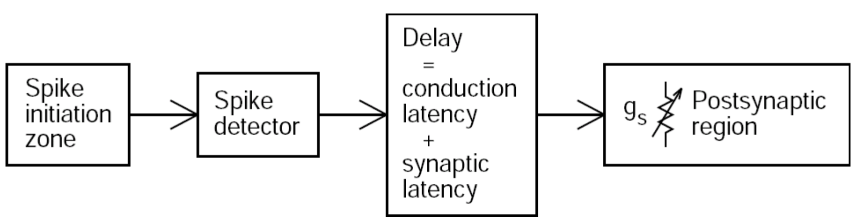
Conceptual model:
- Spike in presynaptic terminal triggers transmitter release; presynaptic details unimportant
- Postsynaptic effect described by DE or kinetic scheme that is perturbed by occurrence of a presynaptic spike

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)
```

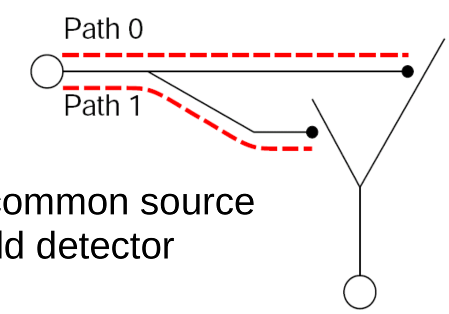
Defaults

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

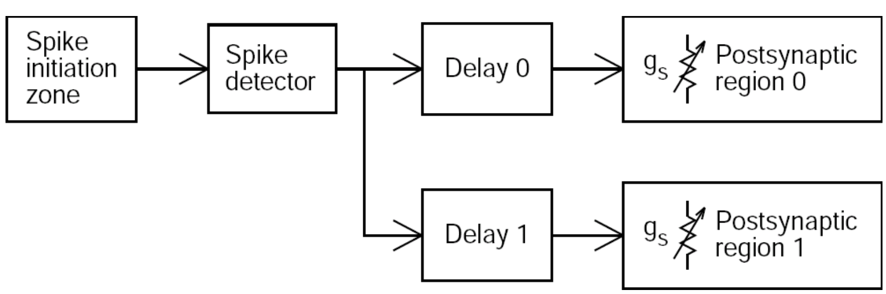
NMODL specification of synaptic mechanism

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

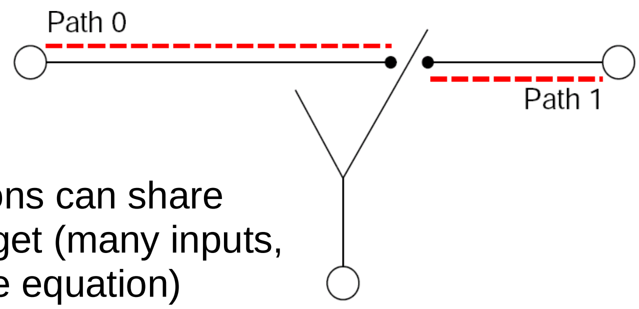
Efficient divergence



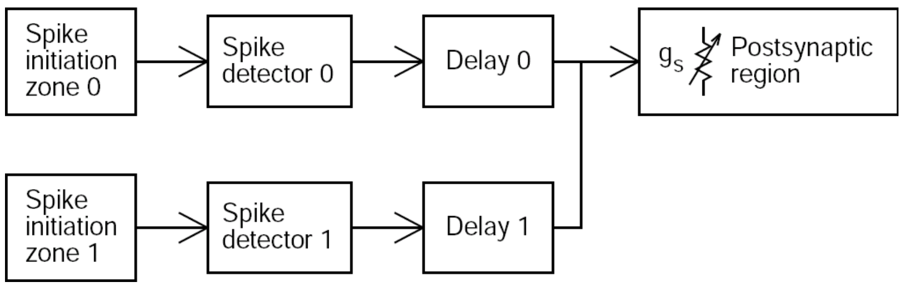
Multiple NetCons with a common source
share a single threshold detector



Efficient convergence



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

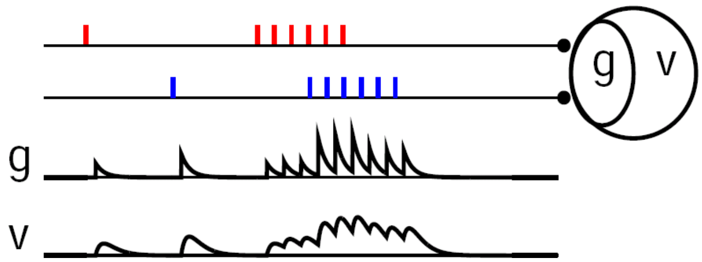


Example: g_s with fast rise and exponential decay

```
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 }
```

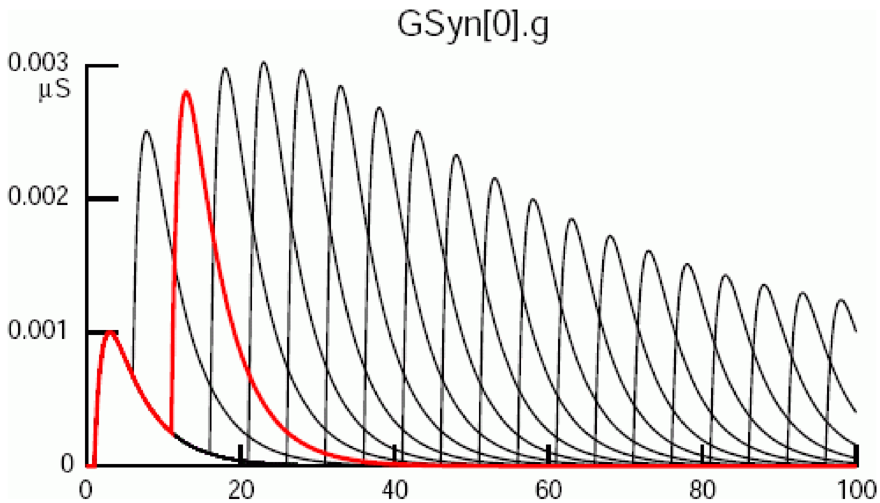
g_s with fast rise and exponential decay

continued



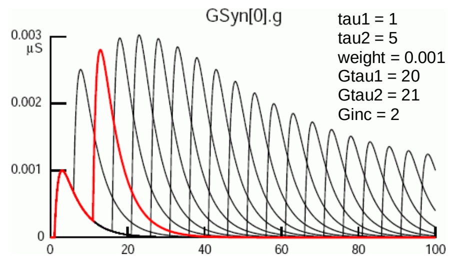

```
BREAKPOINT {
  SOLVE state METHOD cnexp
  i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }
```

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

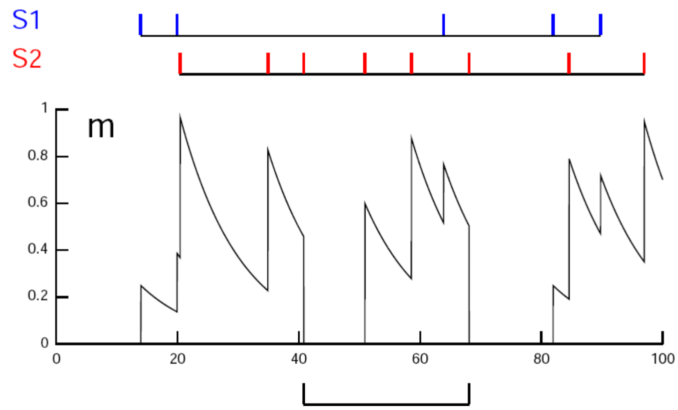
Prerequisite: all state variables must be analytically computable from a new initial condition

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

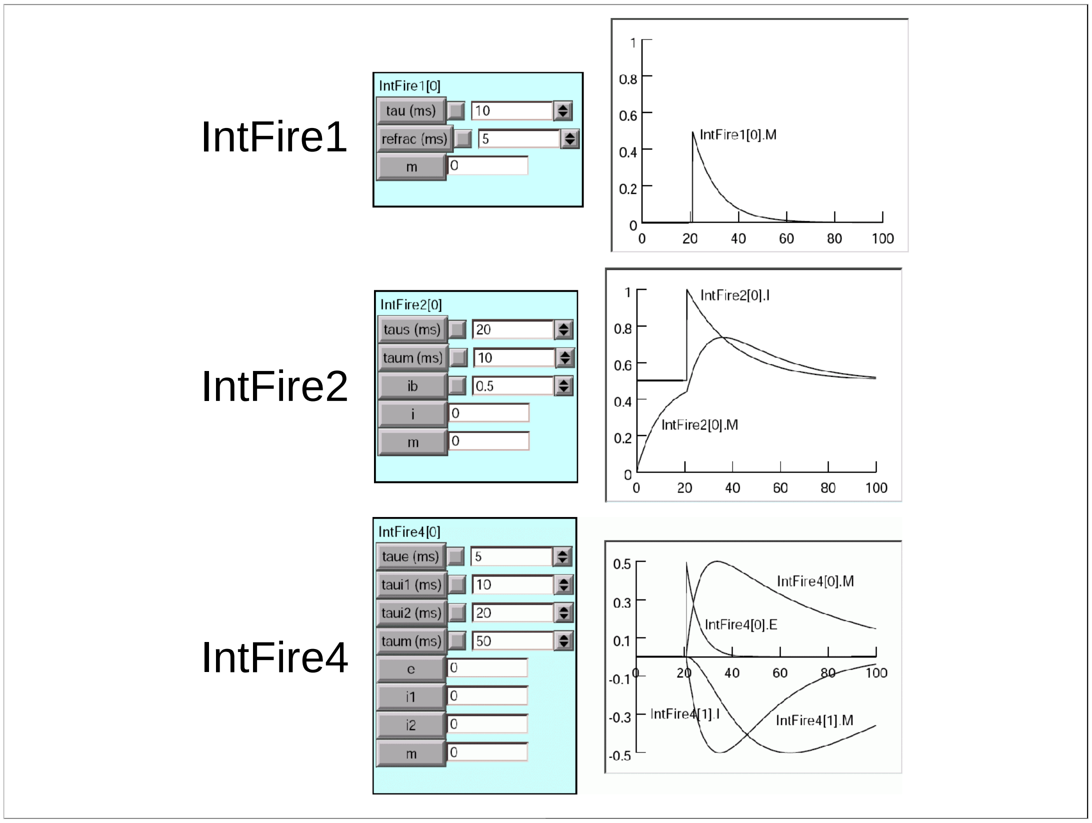
Hybrid networks

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
  }
}
```



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

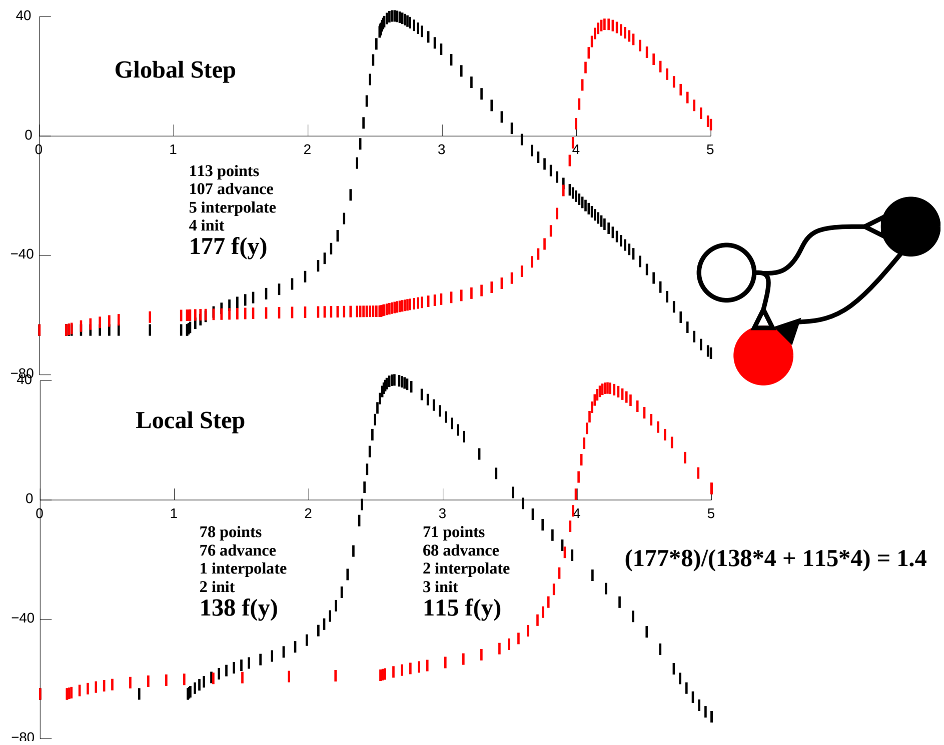
Encapsulate in a class

```
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

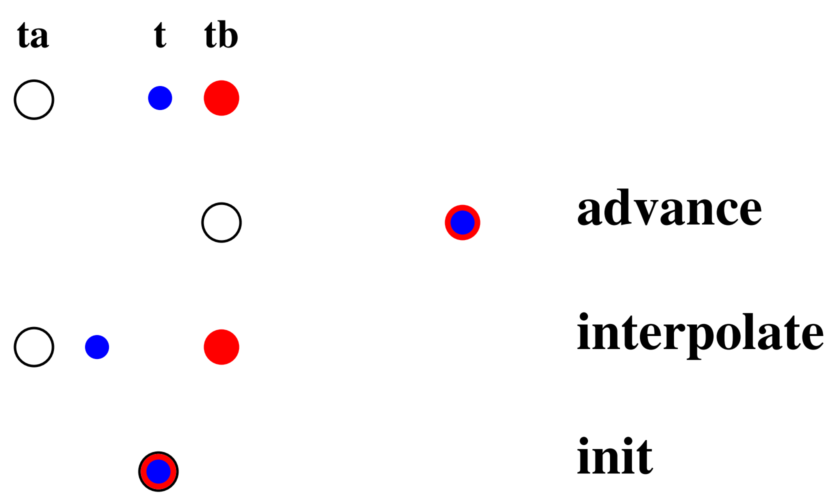
Hint: outer loop should iterate over targets

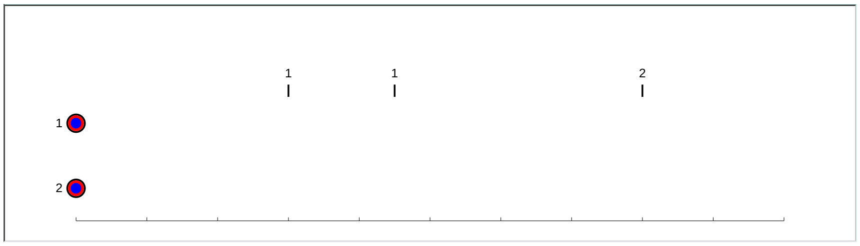
```
for each target cell {
    for each source that projects to this target cell {
        set up a NetCon that connects source to target
    }
}
```

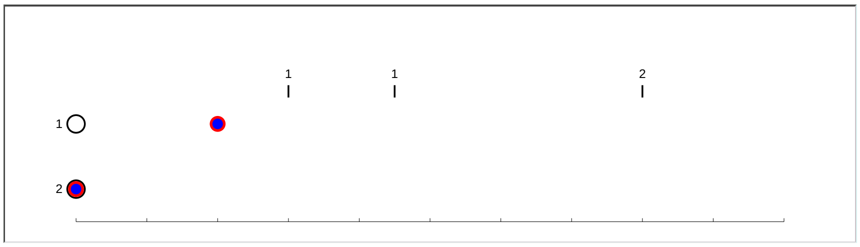


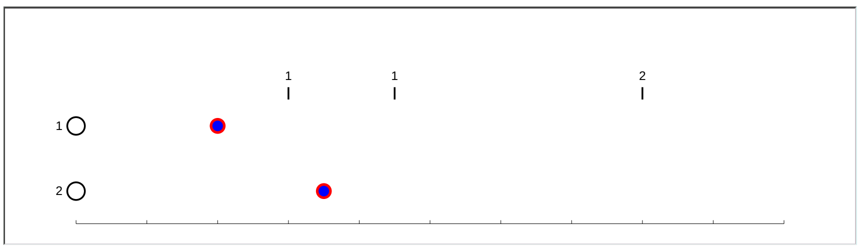
One integrator instance per cell

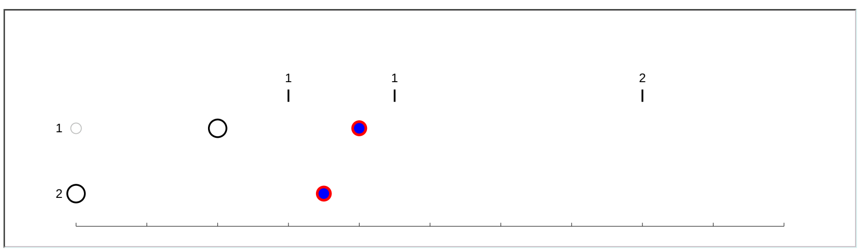
$\forall i, j: \quad ta_i \le tb_j$

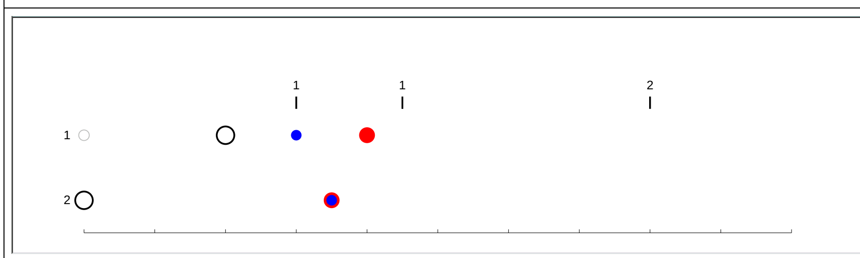


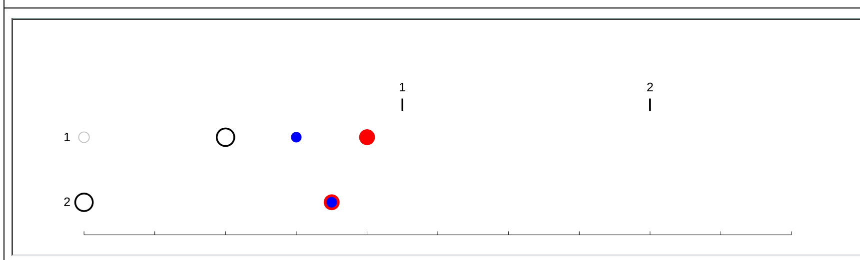

1
1
2
1
2

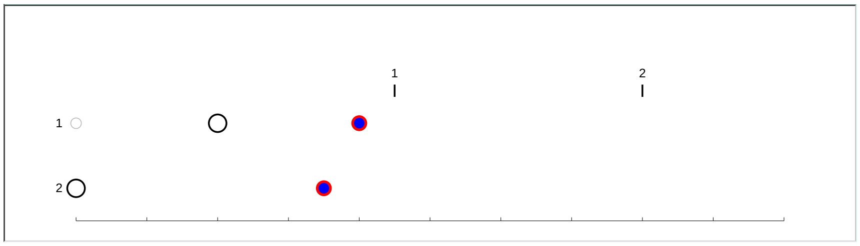

1
1
2
1
2

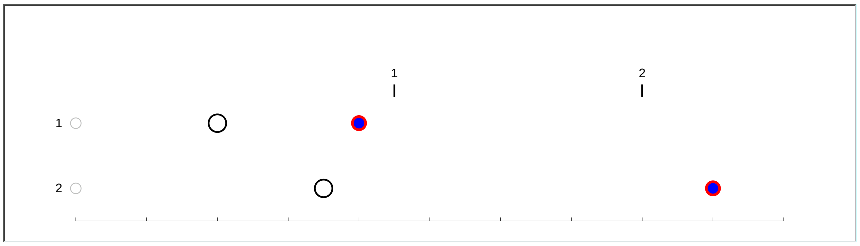

1
1
2
1
2

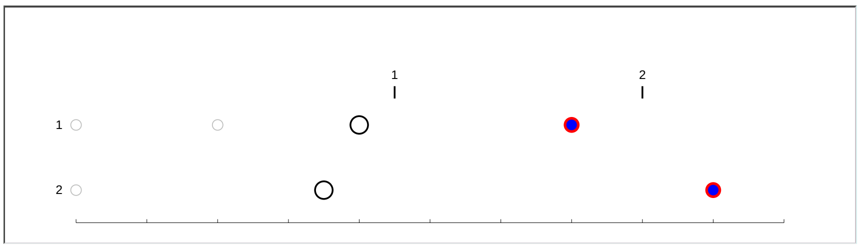

1
1
2
1
2

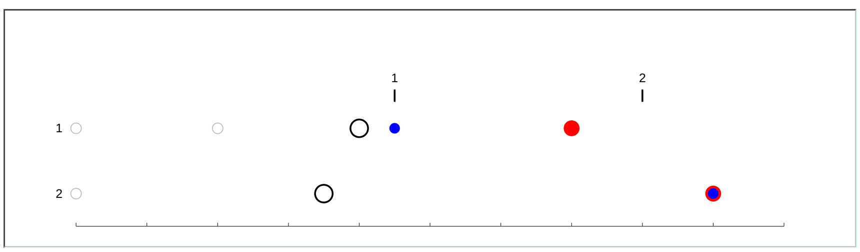

1
1
2
1
2

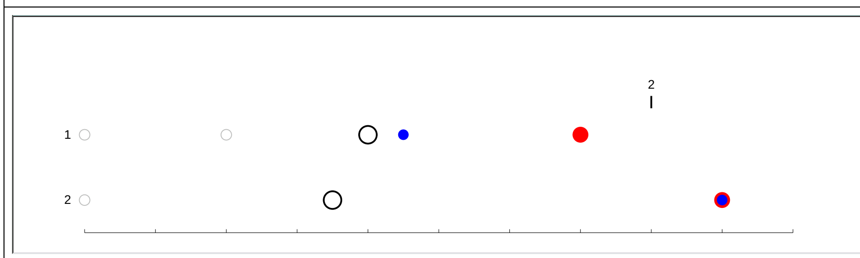

1
2
1
2

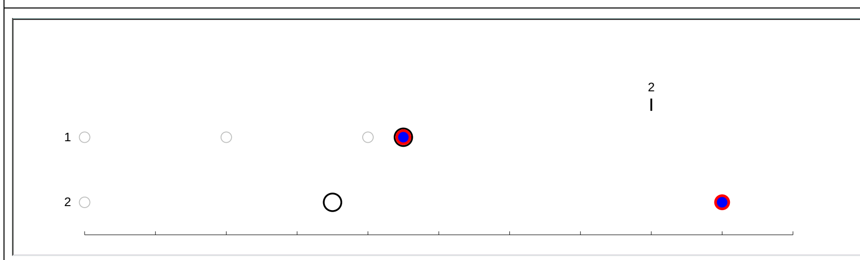

1
2
1
2

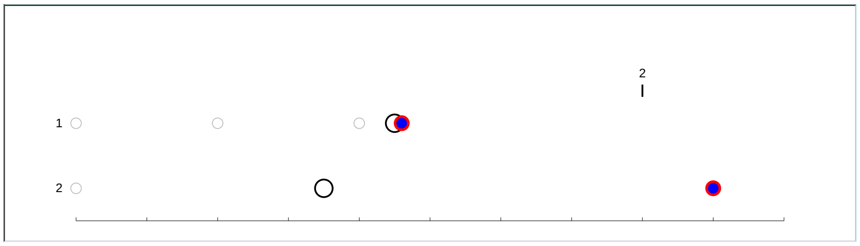

1
2
1
2

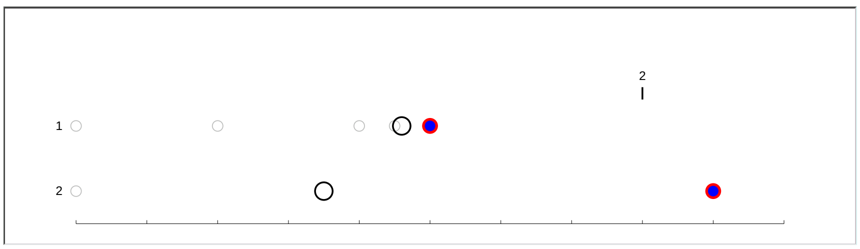

1
2
1
2

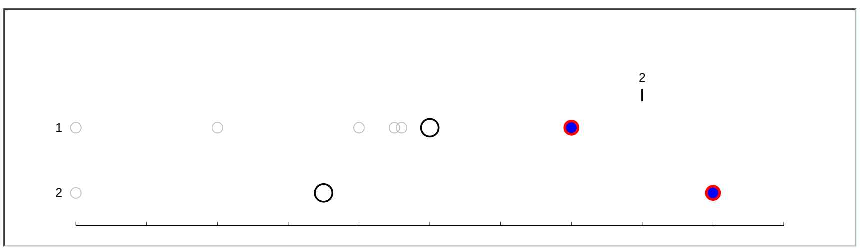

1
2
1
2

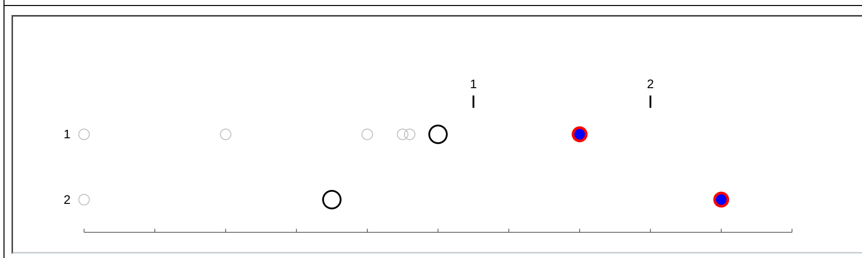

2
1
2

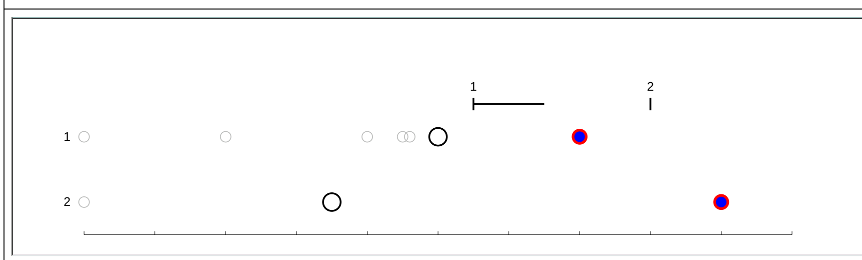

2
1
2

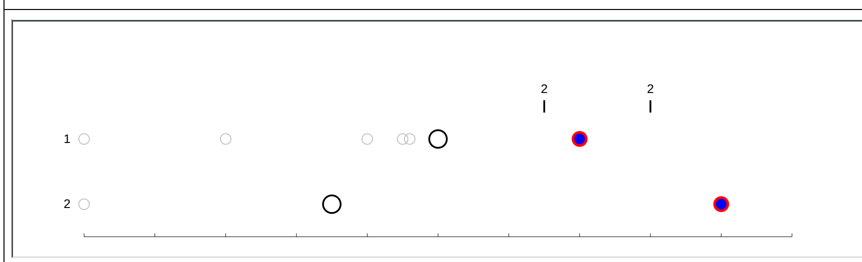

2
1
2

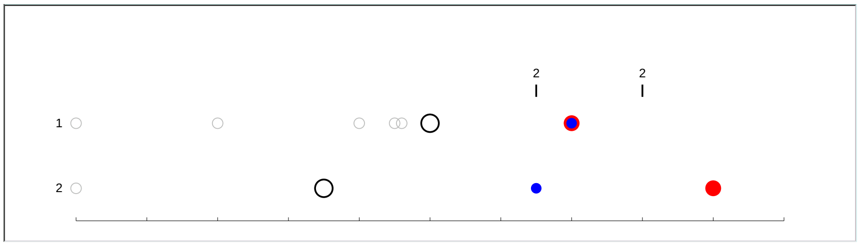

2
1
2

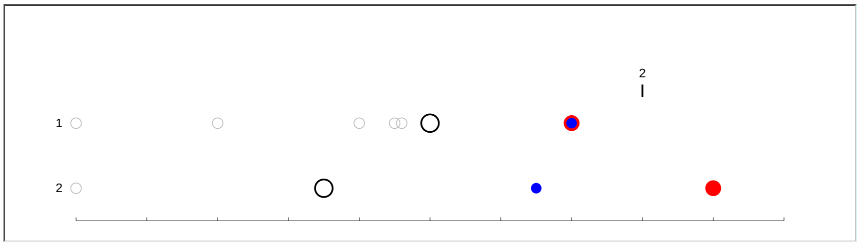

2
1
2

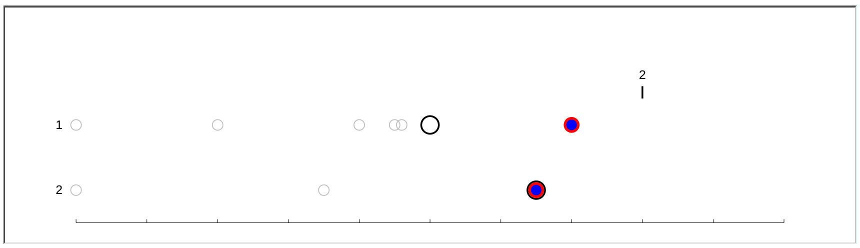

1
2
1
2


1
2
1
2


2
2
1
2


2
2
1
2


2
1
2


2
1
2

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?

8192 processor EPFL IBM BlueGene

1 hour at 700MHz

3 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

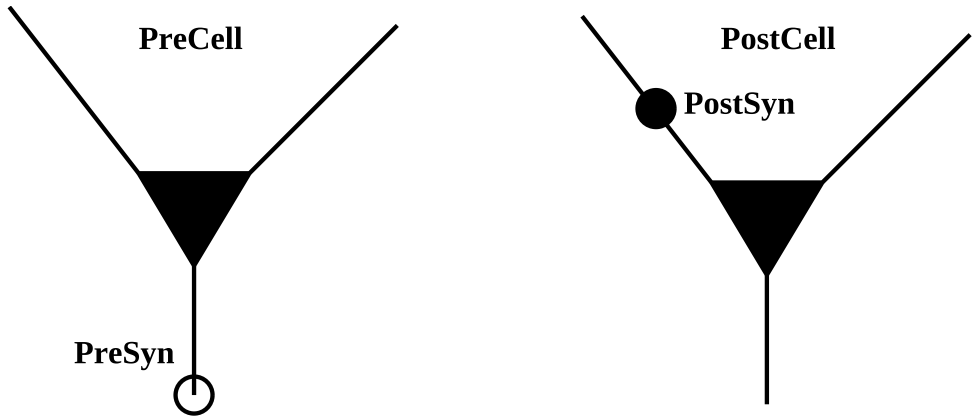
Parallel Computation

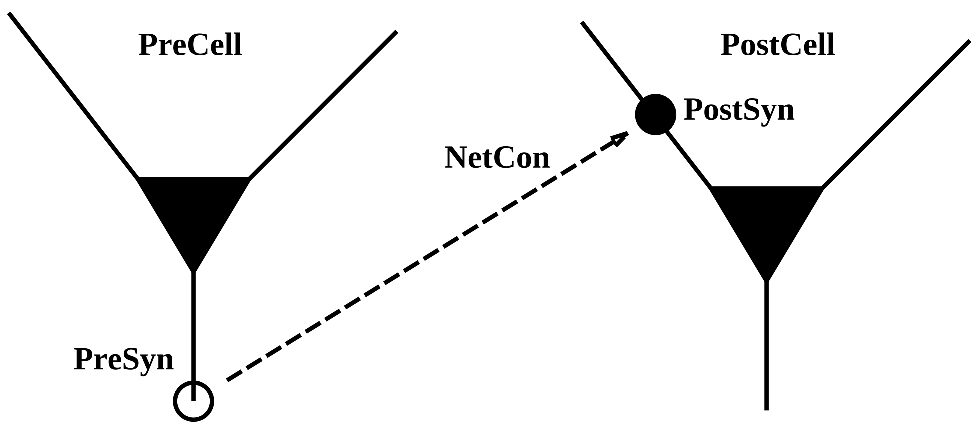
A simulation run takes hours.

want to spread the problem over several machines.

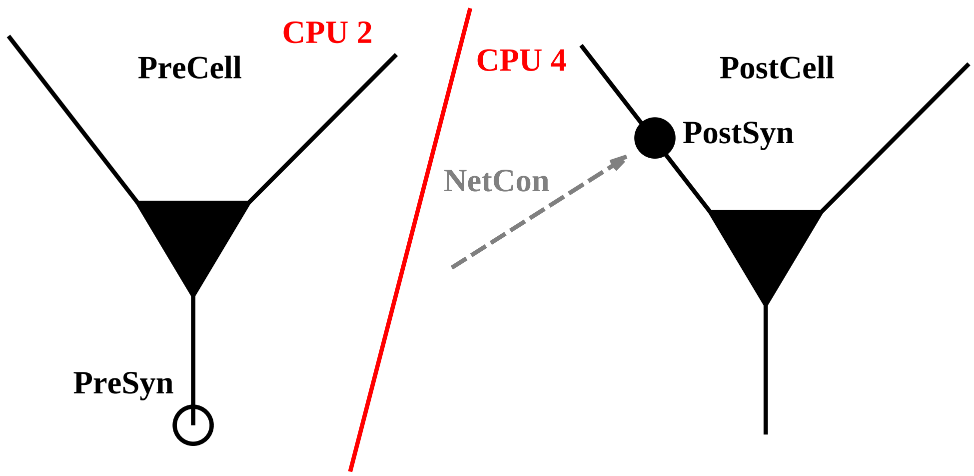
Single cells

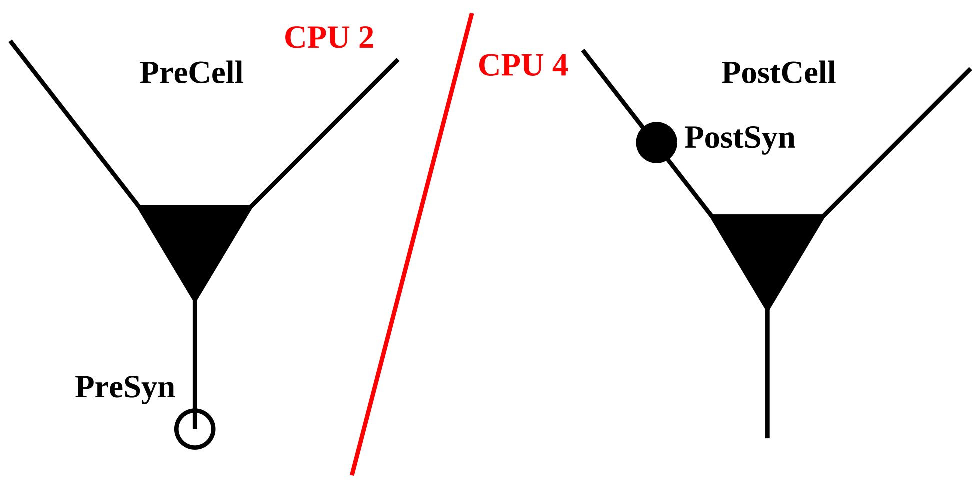
- portions of the tree cable equation on different machines.



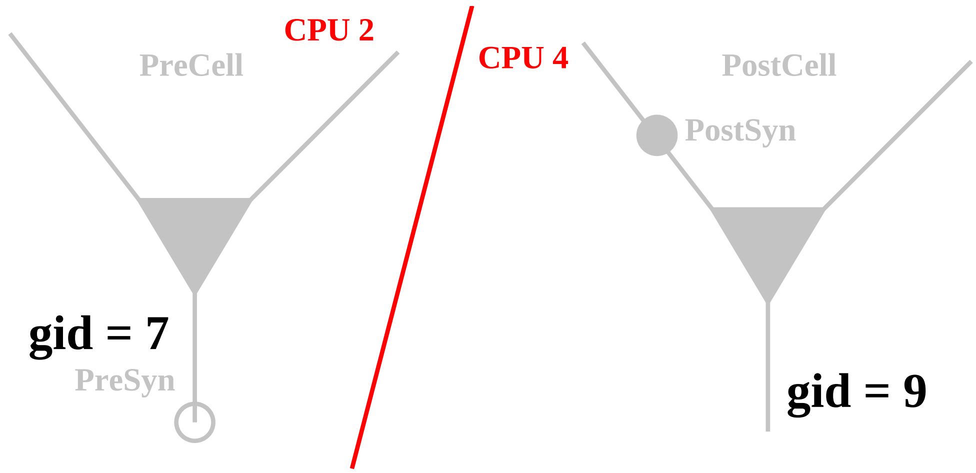



```
nc = new NetCon(PreSyn, PostSyn)
```






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



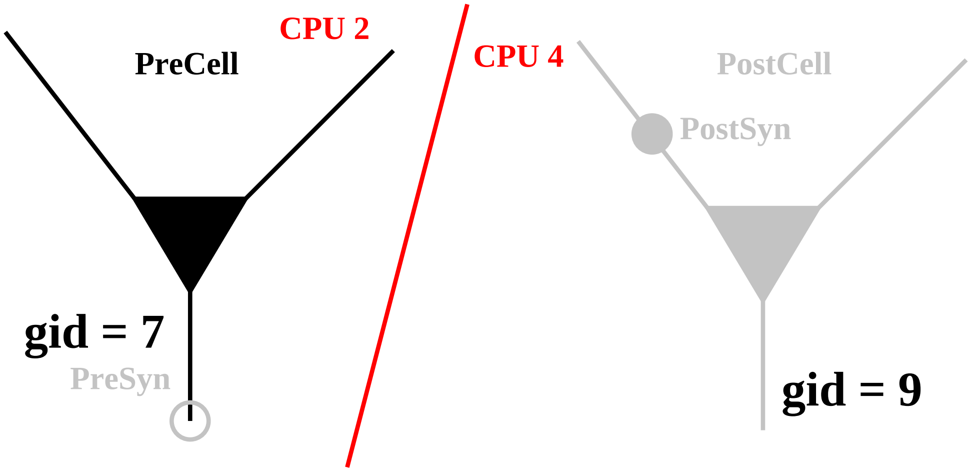
Every spike source (cell) must have a global id number.

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:

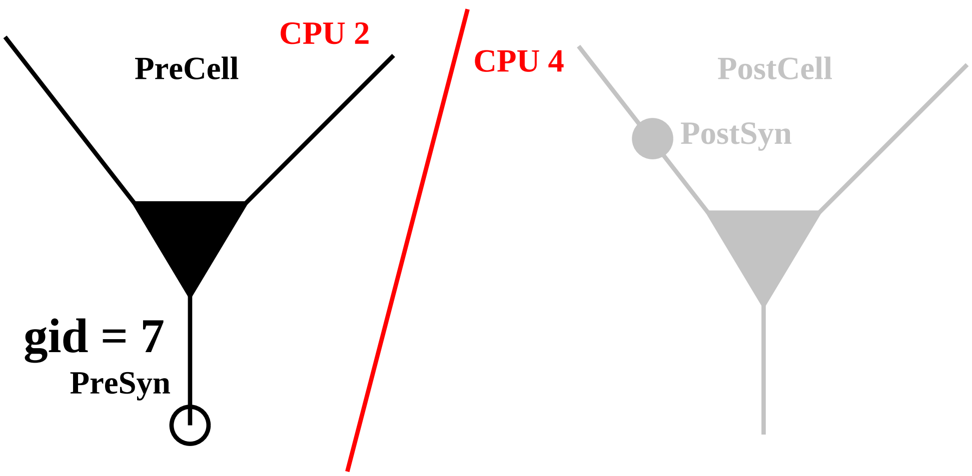
```
for (gid = pc.id; gid < ncell; gid += pc.nhost)
    pc.set_gid2node(gid, pc.id)
    ...
}
```

body executed only ncell/nhost times, not ncell.



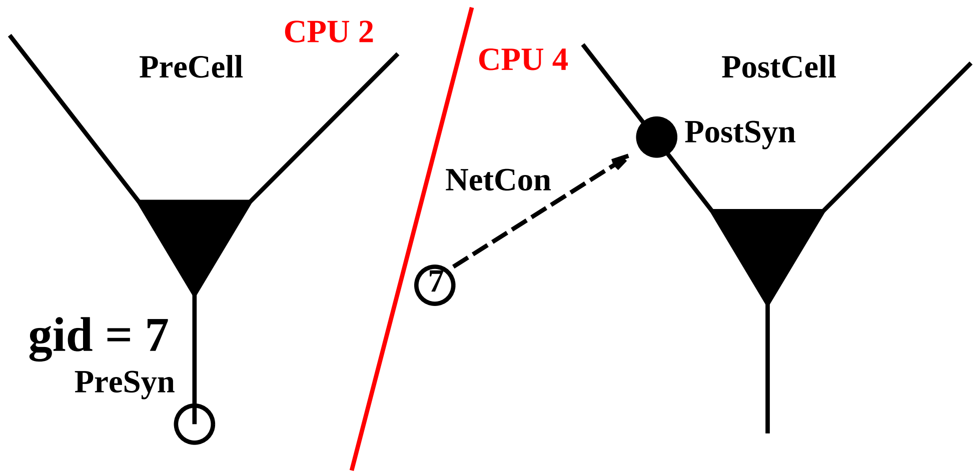
Create cell only where the gid exists.

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



Associate gid with spike source.

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



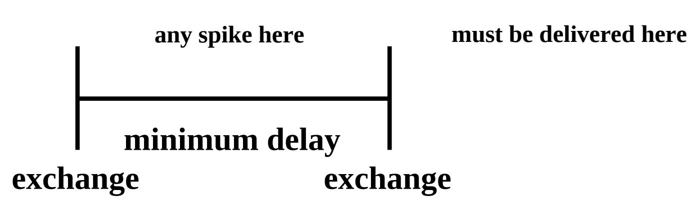
Create NetCon on CPU where target exists.

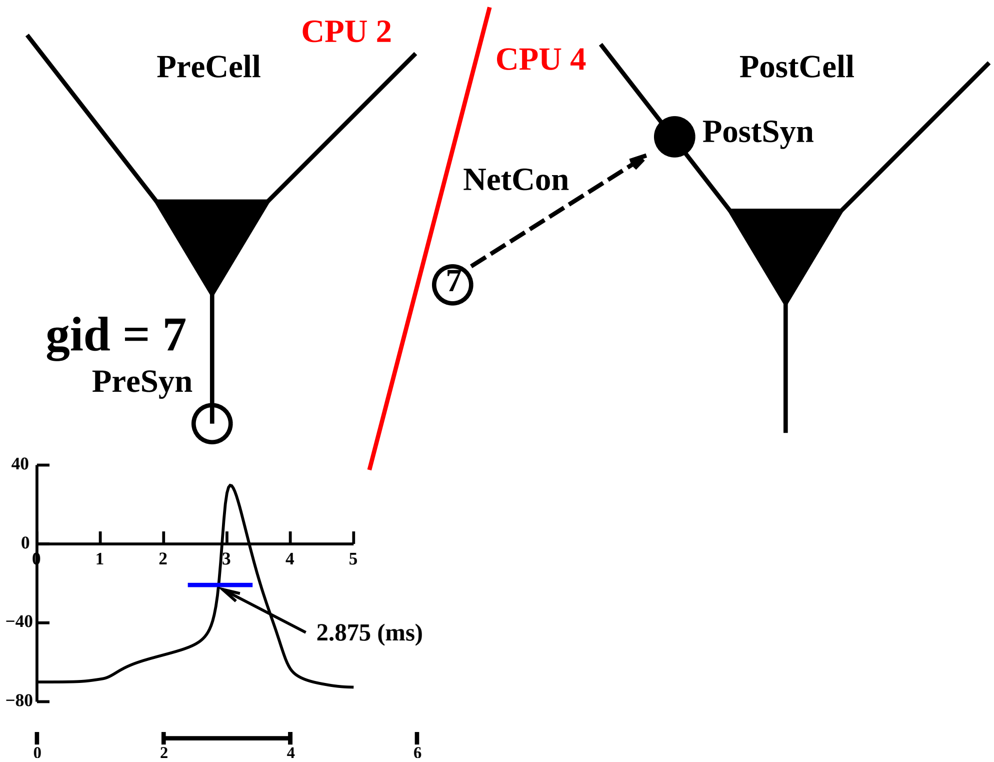
```
nc = pc.gid_connect(7, PostSyn
```

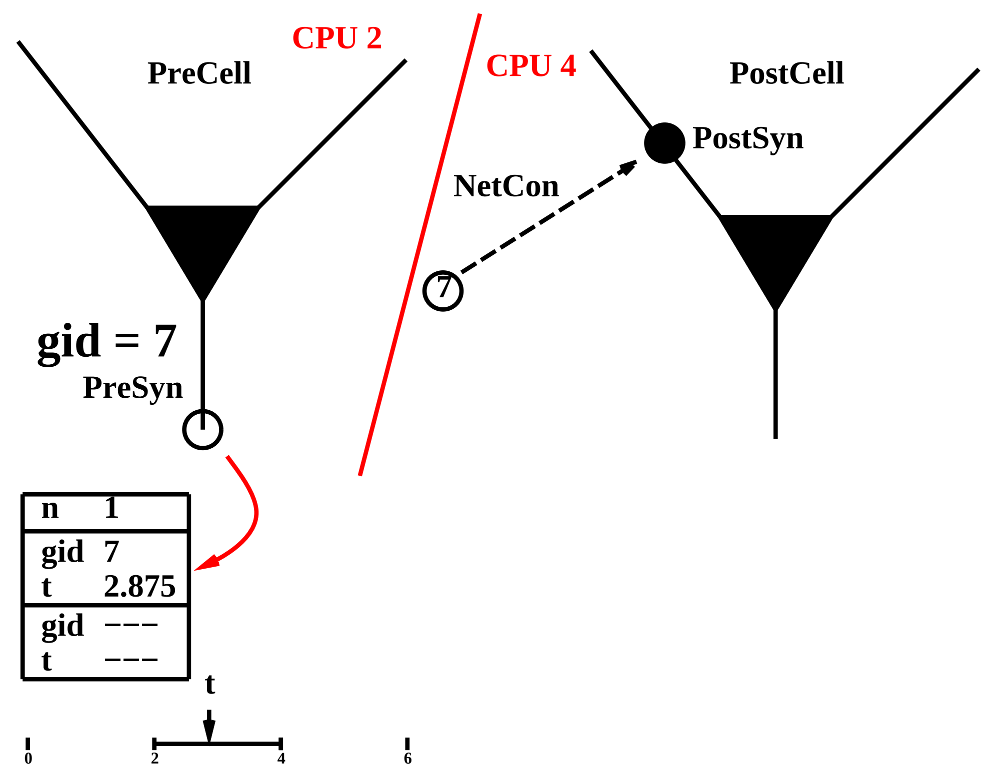
Run using the idiom

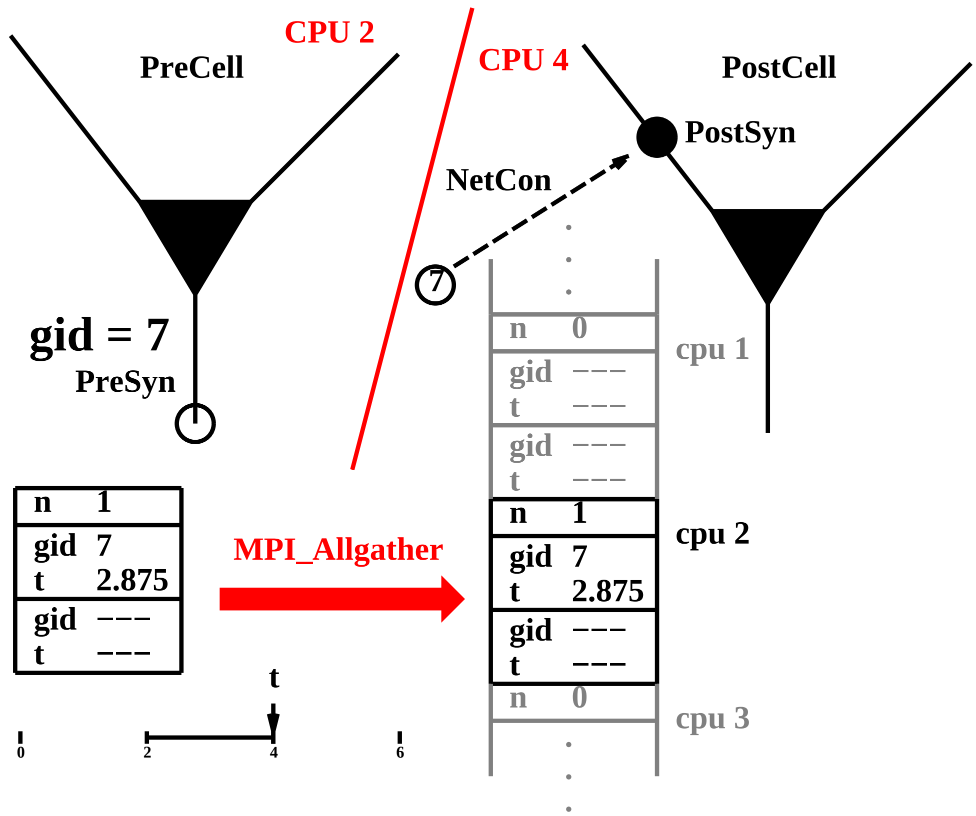
pc.set_maxstep(10)
stdinit()
pc.psolve(tstop)

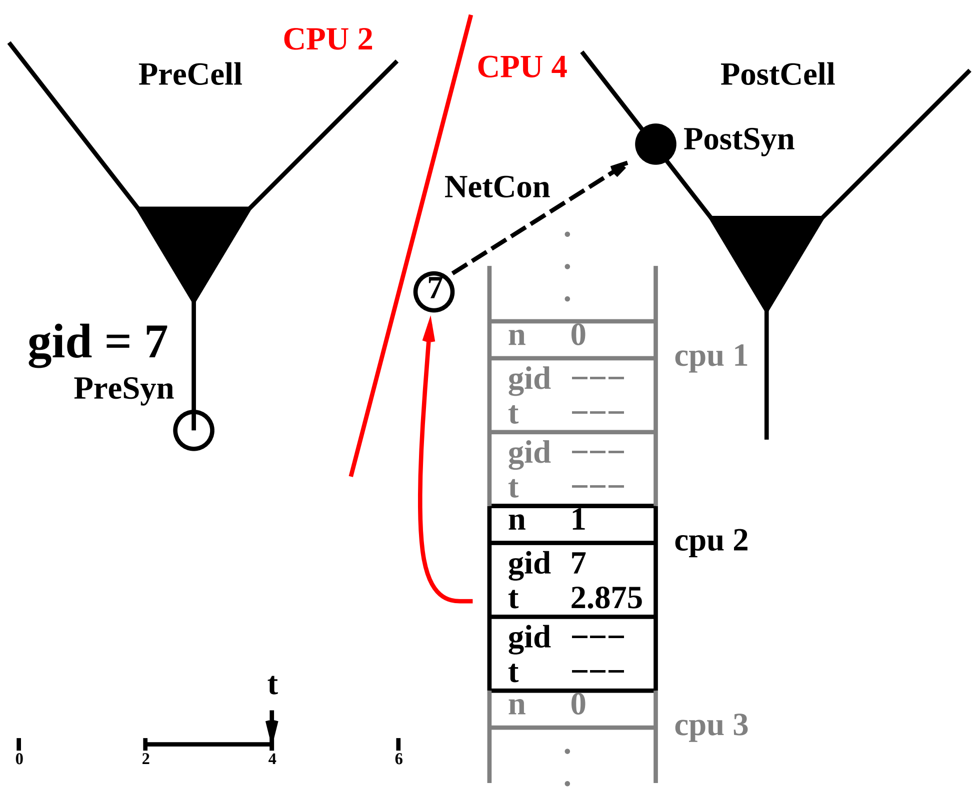
pc.set_maxstep() uses
MPI_Allreduce
to determine minimum delay.

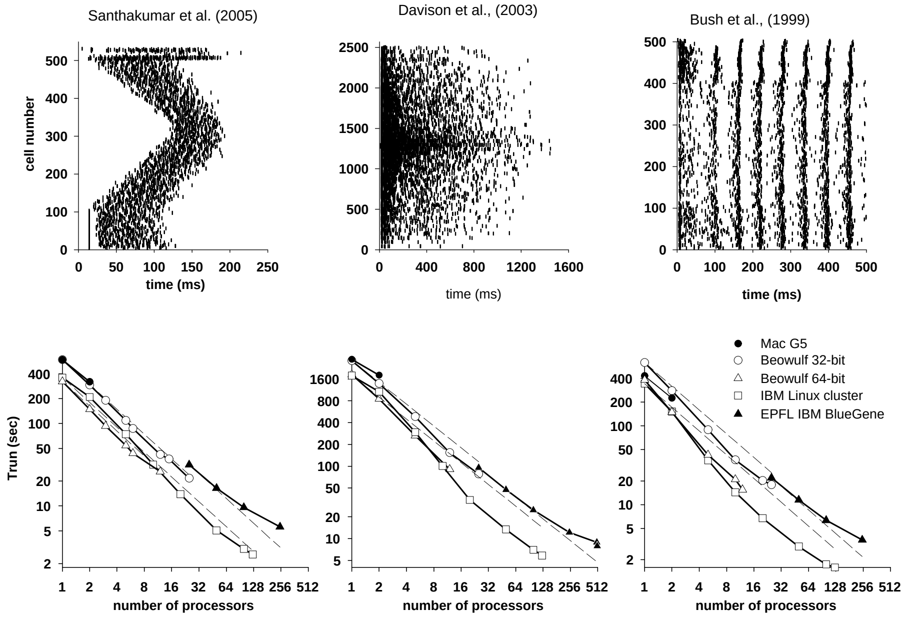



CPU 2
PreCell
CPU 4
PostCell
PostSyn
NetCon
7
gid = 7
PreSyn
40
0
−40
−80
0
1
2
3
4
5
2.875 (ms)
0
2
4
6

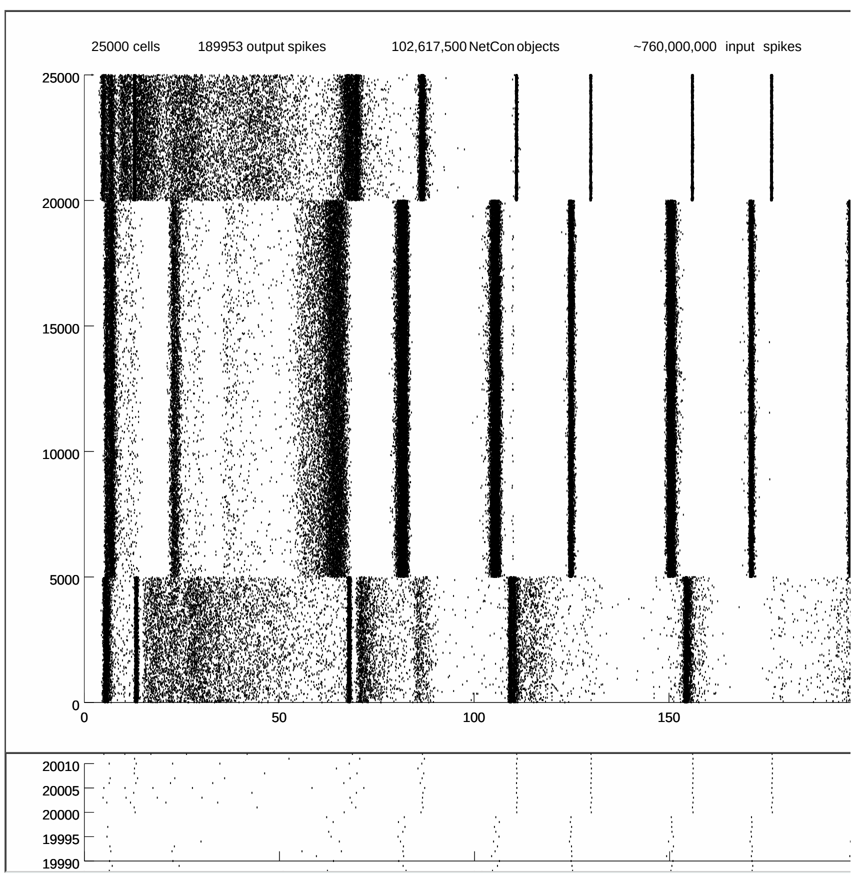

CPU 2
PreCell
CPU 4
PostCell
PostSyn
NetCon
7
gid = 7
PreSyn
n 1
gid 7
t 2.875
gid ---
t ---
t
0
2
4
6


CPU 2
PreCell
CPU 4
PostCell
PostSyn
NetCon
7
gid = 7
PreSyn
n 1
gid 7
t 2.875
gid ---
t ---
MPI_Allgather
t
0
2
4
6
n 0
gid ---
t ---
gid ---
t ---
cpu 1
n 1
gid 7
t 2.875
gid ---
t ---
cpu 2
n 0
cpu 3


CPU 2
PreCell
CPU 4
PostCell
PostSyn
NetCon
7
gid = 7
PreSyn
t
0
2
4
6
n 0
gid ---
t ---
gid ---
t ---
cpu 1
n 1
gid 7
t 2.875
gid ---
t ---
cpu 2
n 0
cpu 3



Santhakumar et al. (2005)
Davison et al., (2003)
Bush et al., (1999)
cell number
time (ms)
Trun (sec)
number of processors
Mac G5
Beowulf 32-bit
Beowulf 64-bit
IBM Linux cluster
EPFL IBM BlueGene



25000 cells
189953 output spikes
102,617,500 NetCon objects
~760,000,000 input spikes

More Efficient Spike Management

Spike exchange buffer compression (Requires fixed step method)

Reduce integration interval to < 256 dt steps, code the double spiketime as a byte.

If there are < 256 cells on each CPU code the int gid as a char local_id.

Select reasonable MPI_Allgather buffer size to send n spikes before requiring an MPI_Allgatherv overflow message.

Bin Queue (Requires fixed step method)

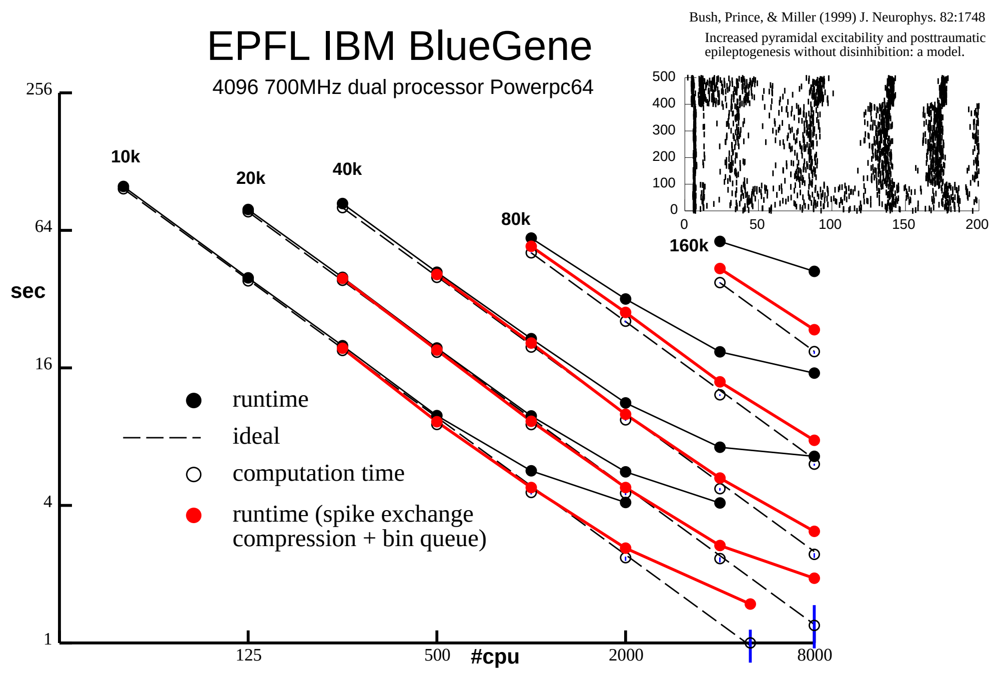
need at least maximum NetCon delay / dt bins

ARTIFICIAL_CELL SelfEvents bypass queue

(Requires the integration interval be <= the positive global minimum NetCon delay)

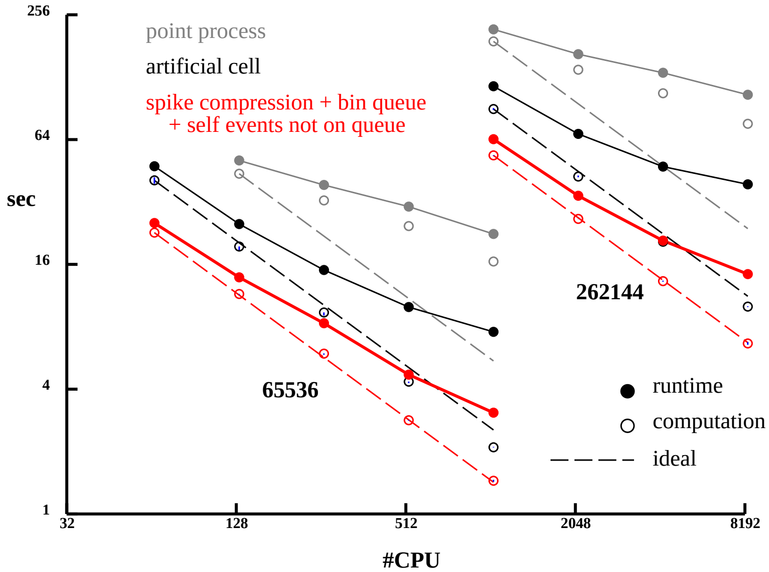
On every incoming NetCon event check to see if SelfEvent < t

After each integration interval iterate over outstanding SelfEvents to deliver all that are < t.



#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

Artificial Spiking Net Performance



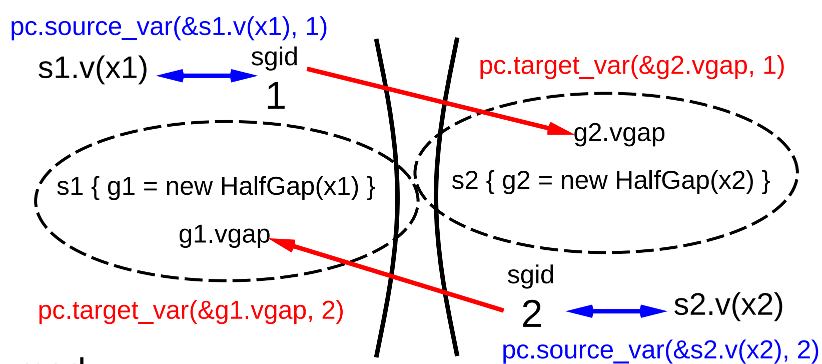
Each cell fires randomly every 10 to 20 ms.
65K cells, 1000 random connections per cell
256K cells, 10,000 random connections per cell

tstop = 200(ms)
delay = 1(ms)
weight = 0

Gap Junction Specification

Continuous Voltage Exchange

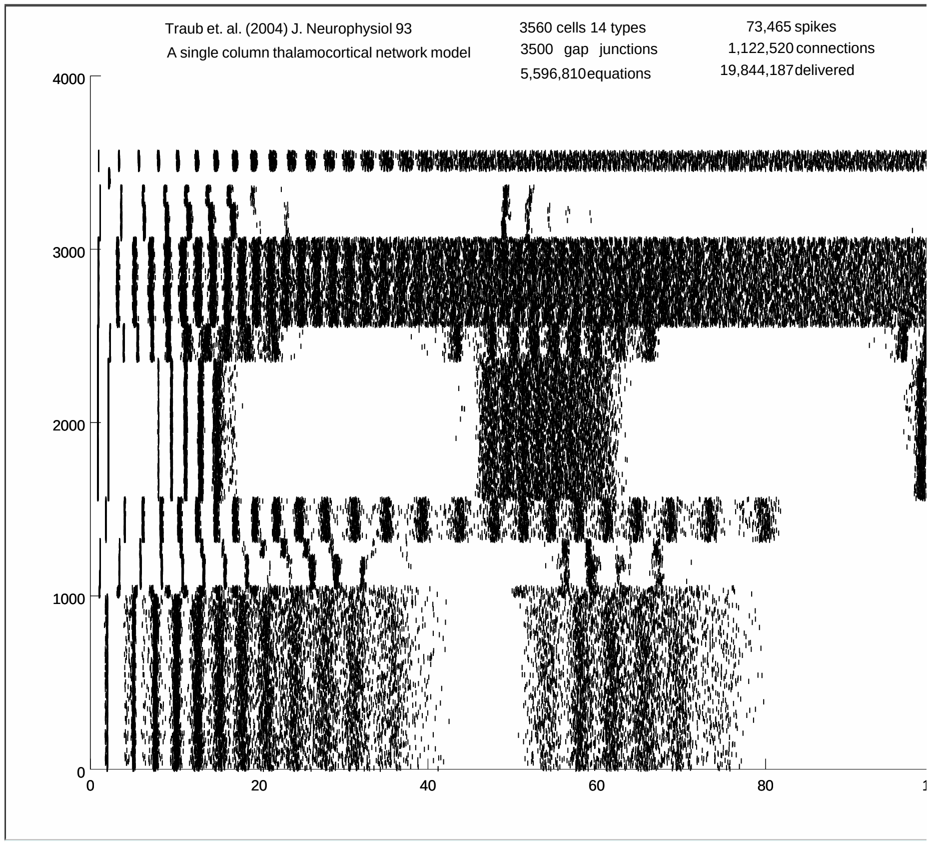
pc.source_var(&source_var, sgid)
pc.target_var(&target_var, sgid)



gap.mod

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

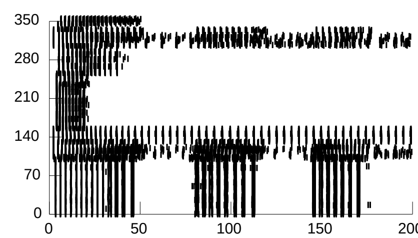
```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
BREAKPOINT { i = (vgap - v)/r }
```



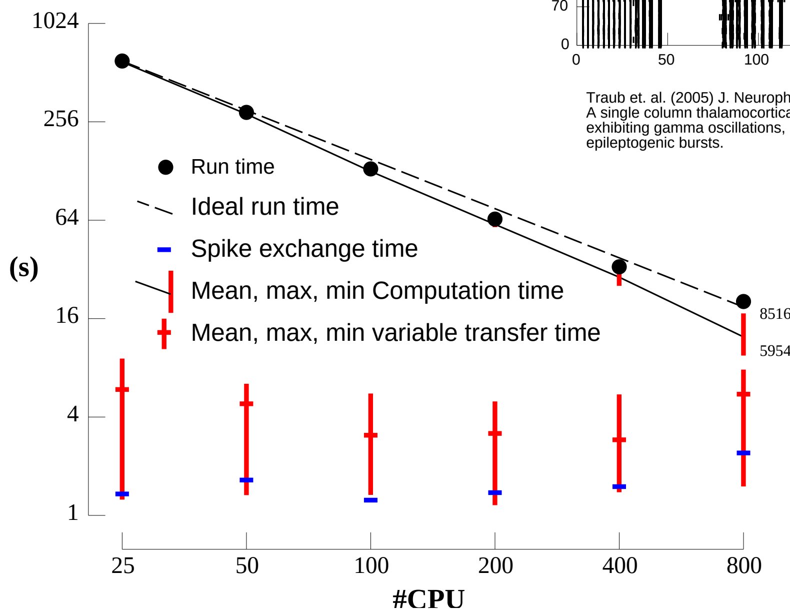
Pittsburgh Supercomputing Center

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.

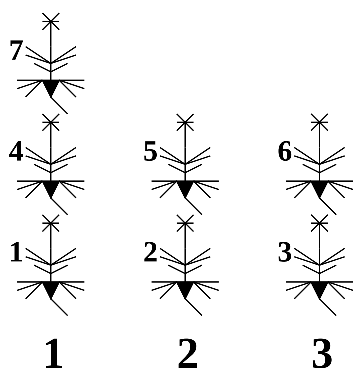


3560 cells 14 types
3500 gap junctions
5,596,810 equations
73,465 spikes
1,122,520 connections
19,844,187 delivered

Load Balance

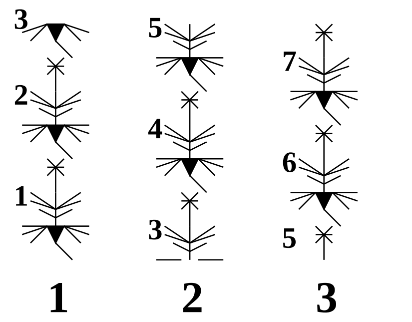
7 cells 3 cpus (or heterogeneous cells)

Round Robin



Fill cpu i ...

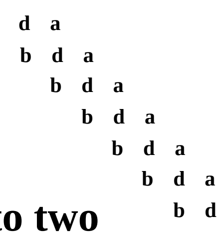
up to (total complexity) / ncpu

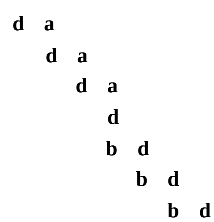


overflow on cpu i + 1

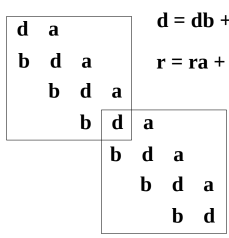
but... what is the overhead of splitting a cell?

Optimal Gaussian elimination:
Triangularize from leaves to root.



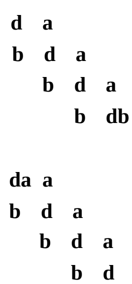


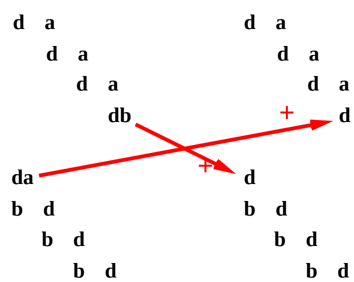
Any tree can be split into two subtrees with a shared root node.



d = db + da

r = ra + rb



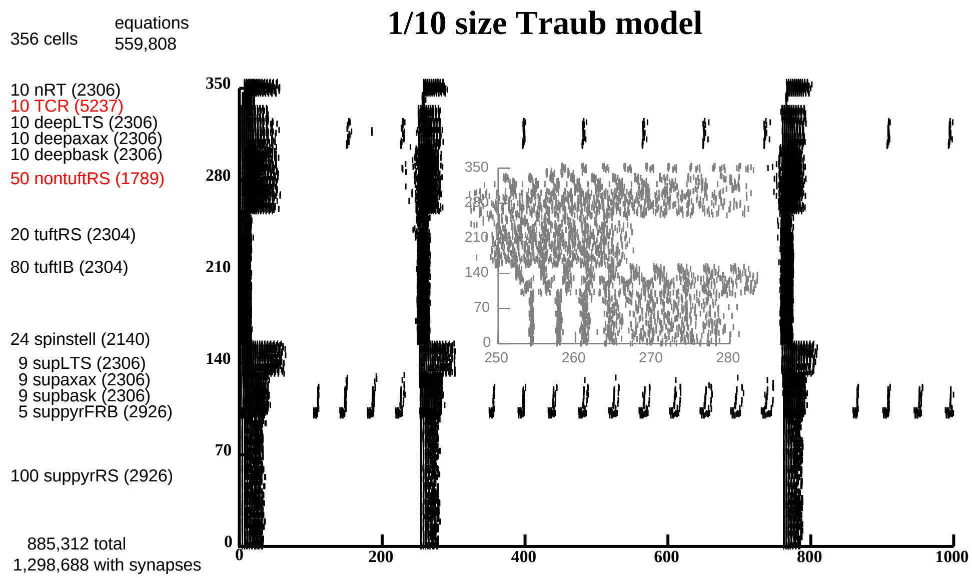


No change in Stability

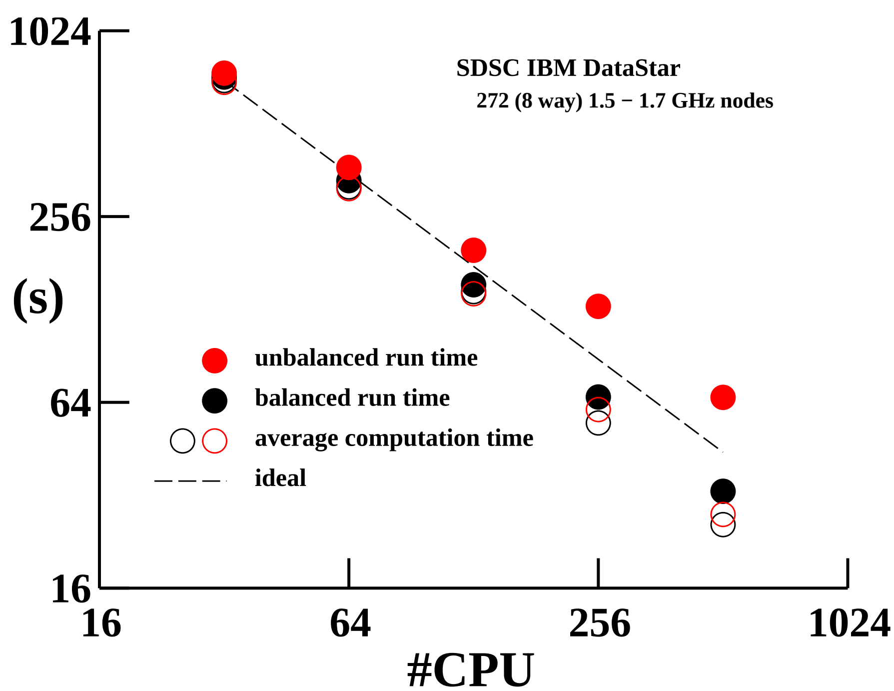
Accuracy

Complexity

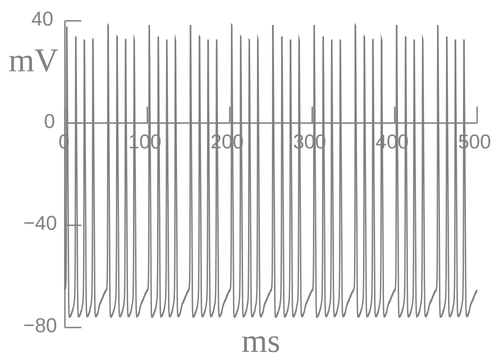
Also exchange ra and rb.

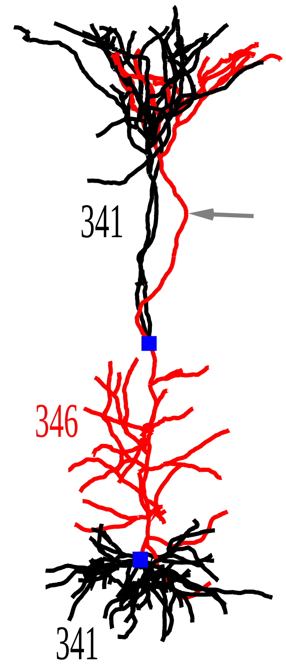


ncpu	avgload	round robin maxload		metis	split cell maxload	
32	40584	42870	6%	1%	40963	0%
64	20292	23310	15%	5%	20496	1%
128	10146	13350	32%	18%	10260	1%
256	5073	9461	86%	57%	5237	3%
512	2536	5475	116%	116%	2839	11%

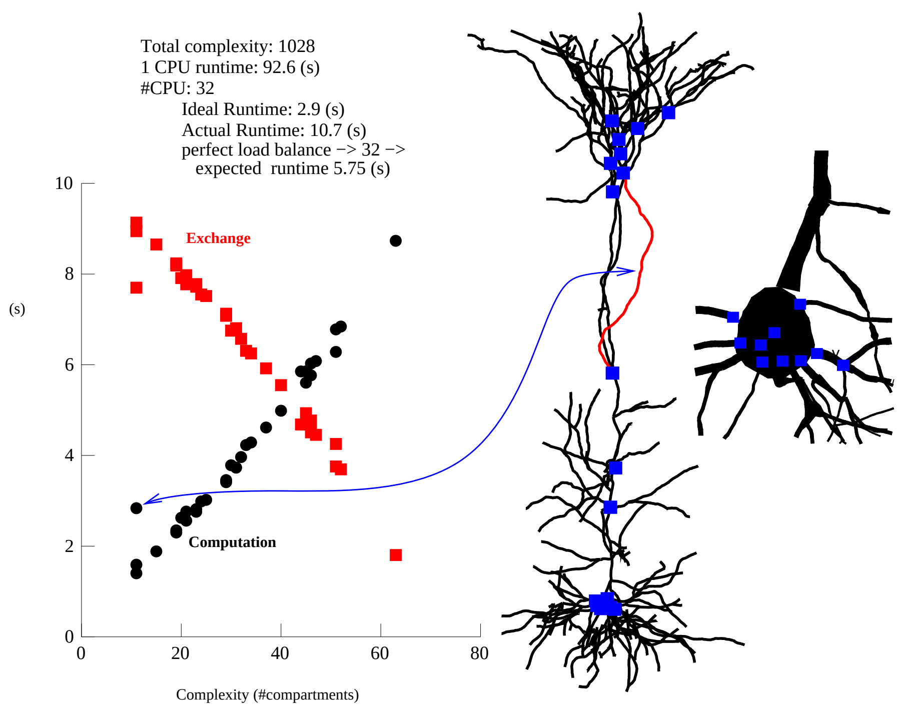


#CPU	Runtime (s)
1	19.1
3	6.5






```
rank  section                       x sid exact
0    dend[113] pc.multisplit( 0, 0, 1)

1    apic[67]  pc.multisplit( 0, 1, 1)

2    soma      pc.multisplit(.5, 0, 1)
2    apic[65]  pc.multisplit( 1, 1, 1)
```



Acknowledgements

Supported by NINDS NS11613

Henry Markram for the use of the EPFL IBM BlueGene and development support under the auspices of the Blue Brain project.

Roger Traub for making his model available in ModelDB.

Terry Sejnowski for the use of the SDSC DataStar.

Pittsburgh Supercomputer Center for the use of the Cray XT3.