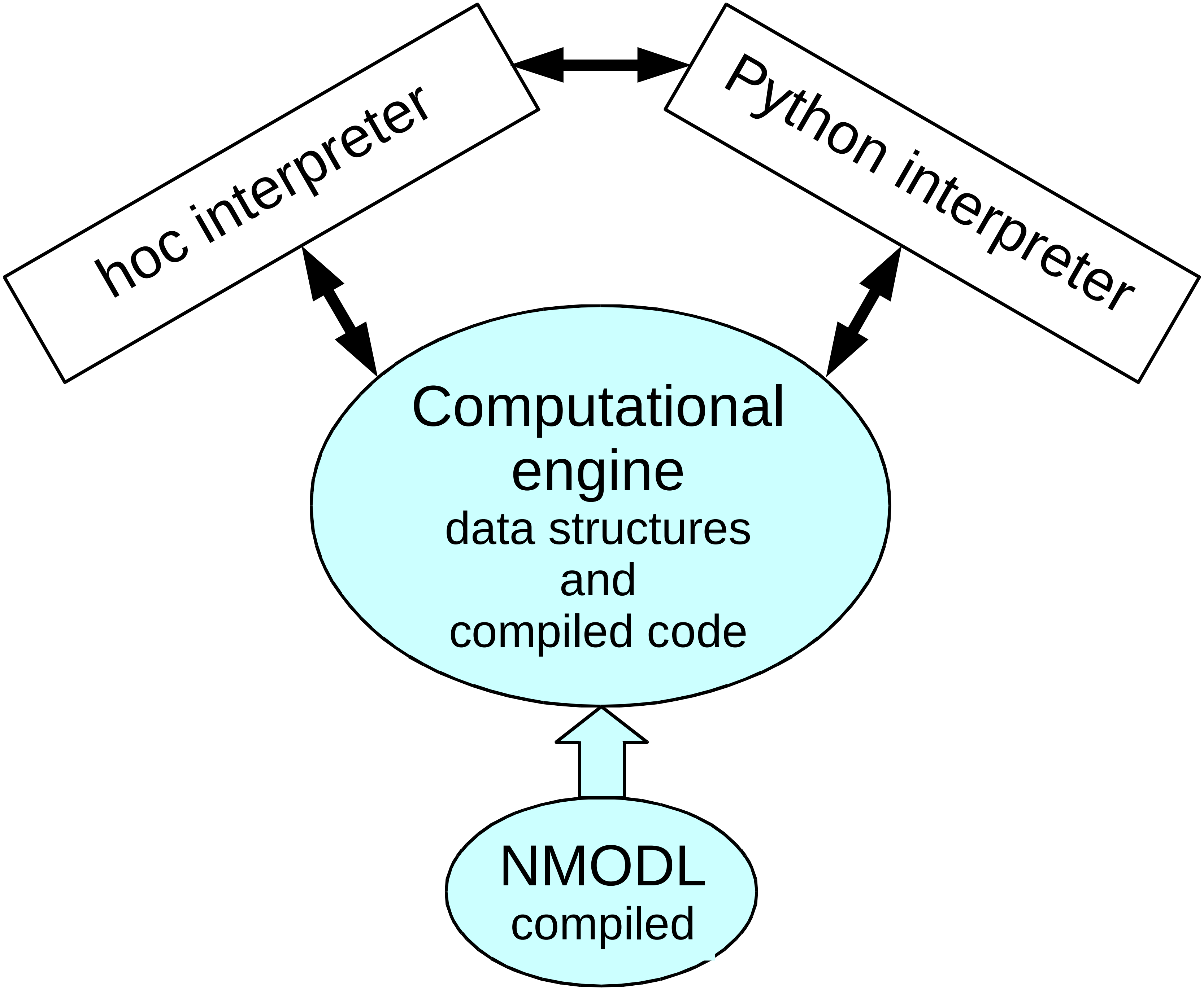
The NEURON Model Description Language

Used to add:

- ion channels
- accumulation, diffusion, transport
- reactions described by ODEs, kinetic schemes
- algebraic equations, e.g. waveform generators
- synaptic mechanisms
- events, state machines, artificial spiking cells

Advantages

- Specification only--independent of solution method
- Efficient--translated into C
- Compact
  - One NMODL statement $\rightarrow$ many C statements
  - Interface code automatically generated
- Consistent ion current / concentration interactions
- Consistent units


hoc interpreter
Python interpreter
Computational
engine
data structures
and
compiled code
NMODL
compiled

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 = 0.036 (S/cm2) <0, 1e9> . . . }
STATE { n . . . }
ASSIGNED { ik (mA/cm2) . . . }
```

Default initial values for states

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

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

(and specify how states are integrated)

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

State equations

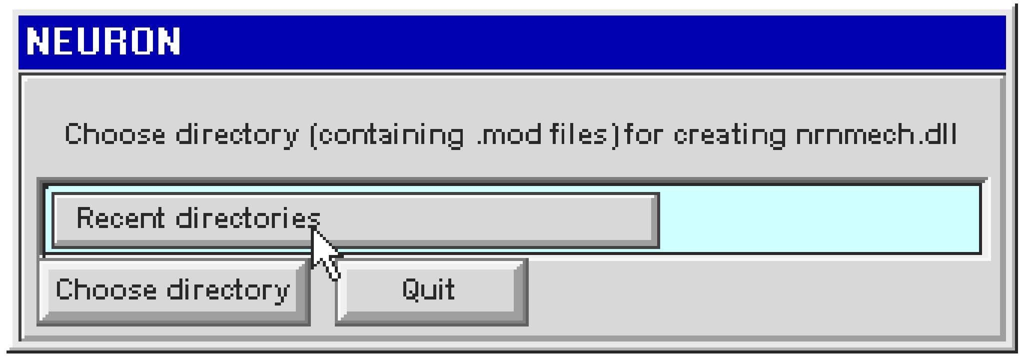
```
DERIVATIVE deriv {
   rates(v)
   n' = (ninf - n)/ntau
}
```

Functions and procedures

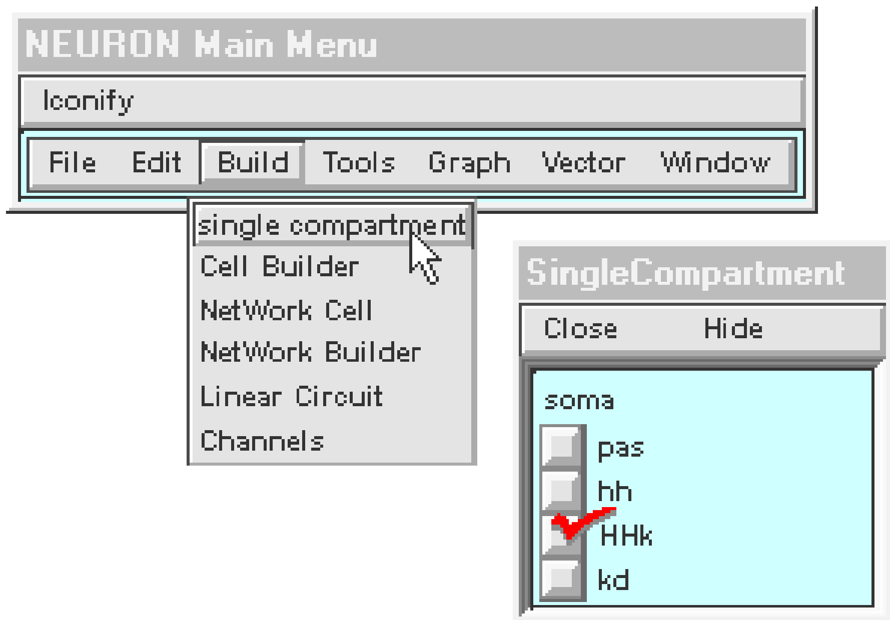
```
PROCEDURE rates(v(mV)) {
   . . .
}
```

Any OS `nrnivmodl`

MSWin only



Result: NEURON has a new mechanism



Density mechanism

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

PARAMETER {
   g = 0.001 (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 = (0.001)*(v − e)/r
}
```

NMODL

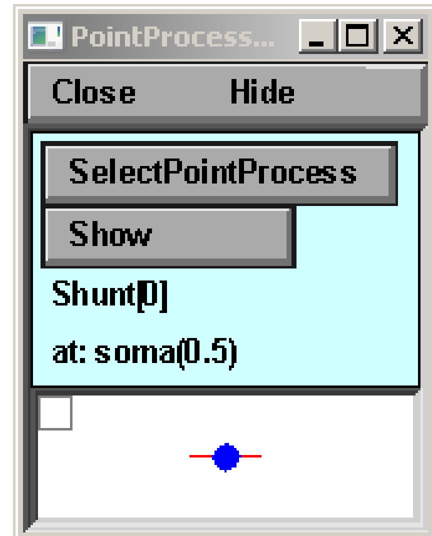
Density mechanism

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

Point Process

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

GUI



Interpreter

hoc:

```
soma {
    insert leak
    g_leak = 1e-4
}
print soma.i_leak(0.5)
```

```
objref s
soma s = new Shunt(0.5)
s.r = 2

print s.i
```

python:

```
soma.insert.h.leak
soma.leak.g = 1e-4
print(soma(0.5).leak.i)
```

```
s = h.Shunt(soma(0.5))
s.r = 2.0
print(s.i)
```

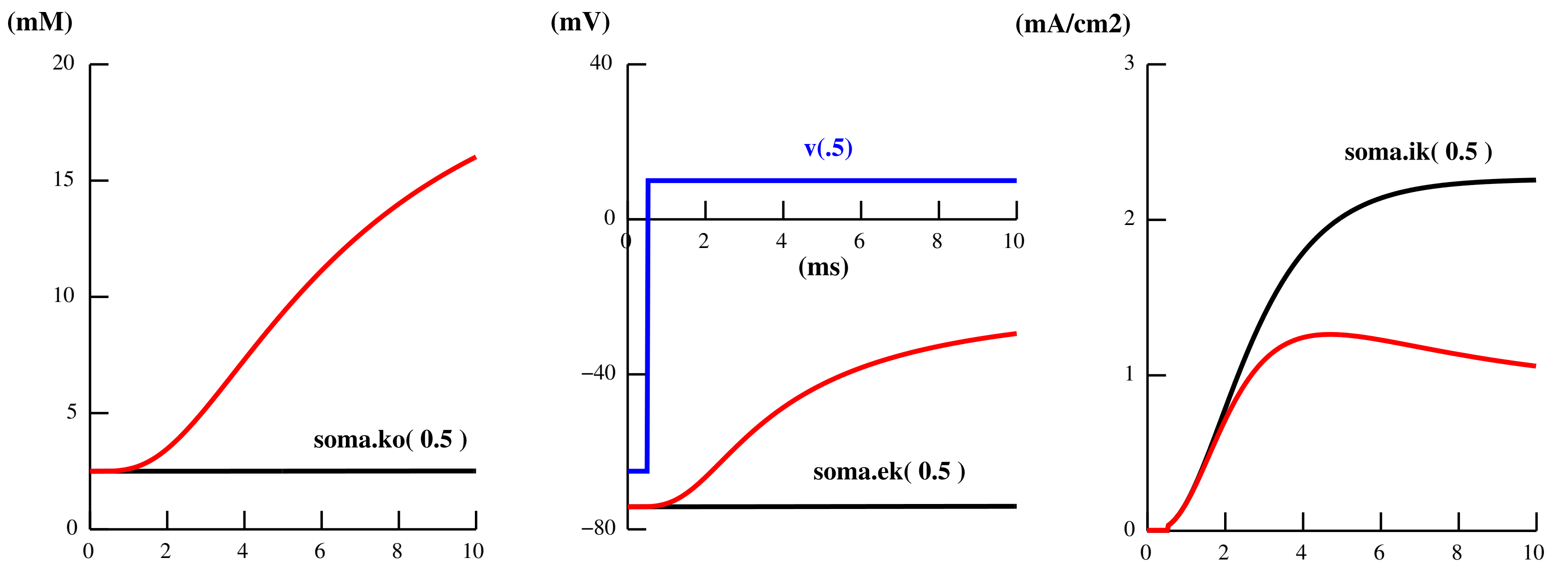
Ion Channel

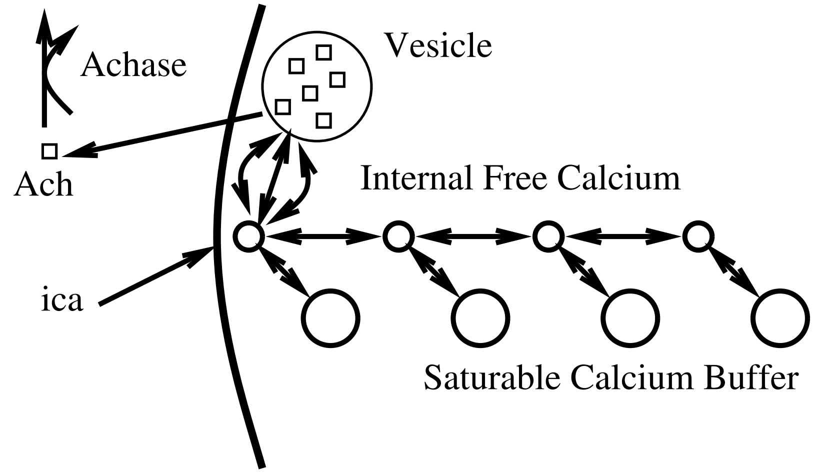
```
NEURON {
  USEION k READ ek WRITE ik
}
BREAKPOINT {
  SOLVE states METHOD cnexp
  ik = gbar*n*n*n*n*(v - ek)
}
DERIVATIVE states {
  rate(v*1(/mV))
  n' = (inf - n)/tau
}
```

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 = (0.001)*(v - e)/r
```

What conversion factor will make the following consistent?

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

Where to find mod files?

NEURON's source code from **github.com/neuronsimulator/nrn**

look in **nrn/src/nrnoc**

ModelDB **modeldb.yale.edu** | **modeldb.science**

"but be careful"

Hines, M.L. and Carnevale, N.T. Expanding NEURON's Repertoire of Mechanisms with NMODL. Neural Computation 12:995-1007, 2000. Get the enhanced preprint **https://neuron.yale.edu/neuron/static/papers/nc2000/nmodl400.pdf**

Chapters 9 and 10 of The NEURON Book

"Why not just write my own?"

- start with something close to what you want
- make small changes and check results

Or resort to the Channel Builder.

Learn more about NMODL

(URLs relative to **`https://neuron.yale.edu/neuron/static/`** unless otherwise noted)

Hines, M.L. and Carnevale, N.T. Expanding NEURON's Repertoire of Mechanisms with NMODL. Neural Computation 12:995-1007, 2000. Get the enhanced preprint **`papers/nc2000/nmodl400.pdf`**

Chapters 9 and 10 of The NEURON Book

"Integration methods for SOLVE statements"
`https://neuron.yale.edu/phpBB/viewtopic.php?f=28&t=592`

Programmer's Reference documentation of NMODL
`py_doc/modelspec/programmatic/mechanisms/nmodl.html`

and the NEURON block in particular
`py_doc/modelspec/programmatic/mechanisms/nmodl2.html`

Future developments: **`https://github.com/BlueBrain/nmodl`**

Homework: virtual molecular biology!

In this experiment you will use a computational model to perform a virtual knockout and rescue experiment.

First, you will create a "control" model cell with Hodgkin-Huxley ion channels and verify that it can generate a spike.

Then you will "knock out" its potassium channels (by reducing the hh mechanism's gkbar to 0), and see what that does to its electrical activity.

Finally, you will "rescue" the cell's excitability by making it "express" a potassium channel that replaces the one that is bundled with the hh mechanism.

Part 1. Create a "control" model cell and verify that it can generate a spike.

1. Copy

```
https://www.neuron.yale.edu/ftp/neuron/
        2021_NEURON_Online_Course/hhkchan.mod
```

into an empty directory.

2. In a terminal, navigate to the directory that contains `hhkchan.mod` and execute

```
nrnivmodl
```

3. In that same directory, start Python and then

```
from neuron import h, gui
```

Part 1 *continued*

4. Use a CellBuilder to create a single compartment model with these properties:

    surface area 100 um2

    Ra = 100 ohm cm, cm = 1 uf/cm2

    hh channels with default channel densities

    HHk channels with gkbar set to 0

5. Set up a user interface that includes

    a RunControl panel

    a voltage axis graph (plot of v at soma(0.5) vs. t)

    a PointProcessManager configured as an IClamp with del 1 ms, dur 0.1 ms, and amp 0.1 nA.

6. Run a simulation.

Do you see a normal hh action potential?

Part 2. "Knock out" the hh potassium channels.

Knock out the hh potassium channels by using the CellBuilder to set gkbar_hh to 0 S/cm2

Without changing the IClamp's parameters, run a new simulation. Do you get a spike? Can you elicit a spike by adjusting the IClamp's dur or amp parameters?

When you are finished exploring the effects of changing the IClamp's dur and amp, restore these parameters to 0.1 ms and 0.1 nA, respectively.

Part 3. "Rescue" excitability.

Change gkbar_HHk to 0.036 S/cm2. Run a new simulation to verify that the model generates a normal action potential waveform.

Consider using Keep Lines and Color/Brush to generate a figure that confirms that the control and rescued action potentials have the same waveform.