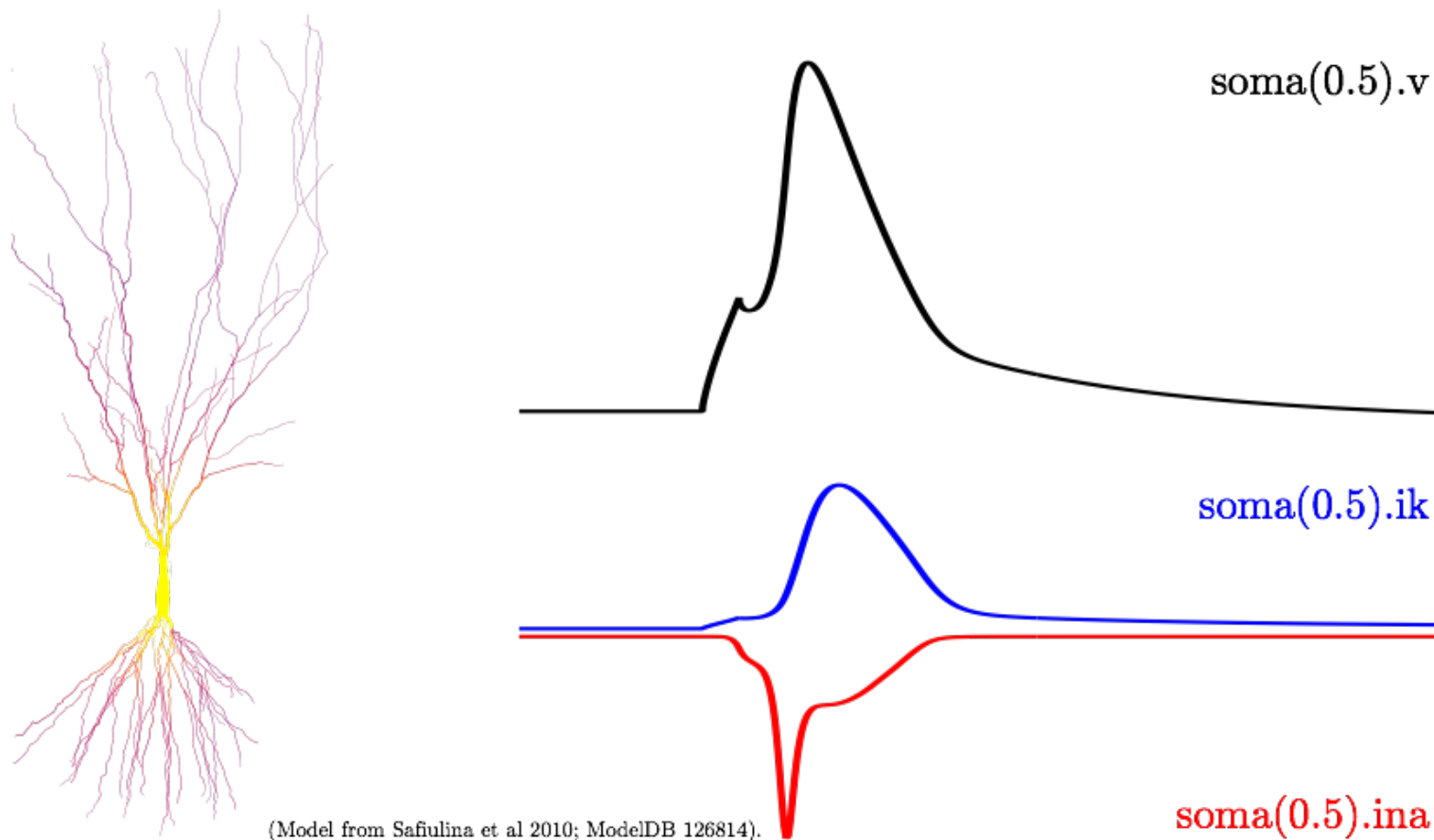


# Modeling neuronal reaction-diffusion

Robert A McDougal

10 November 2017

Neurons generate action potentials by moving ions across their membrane.

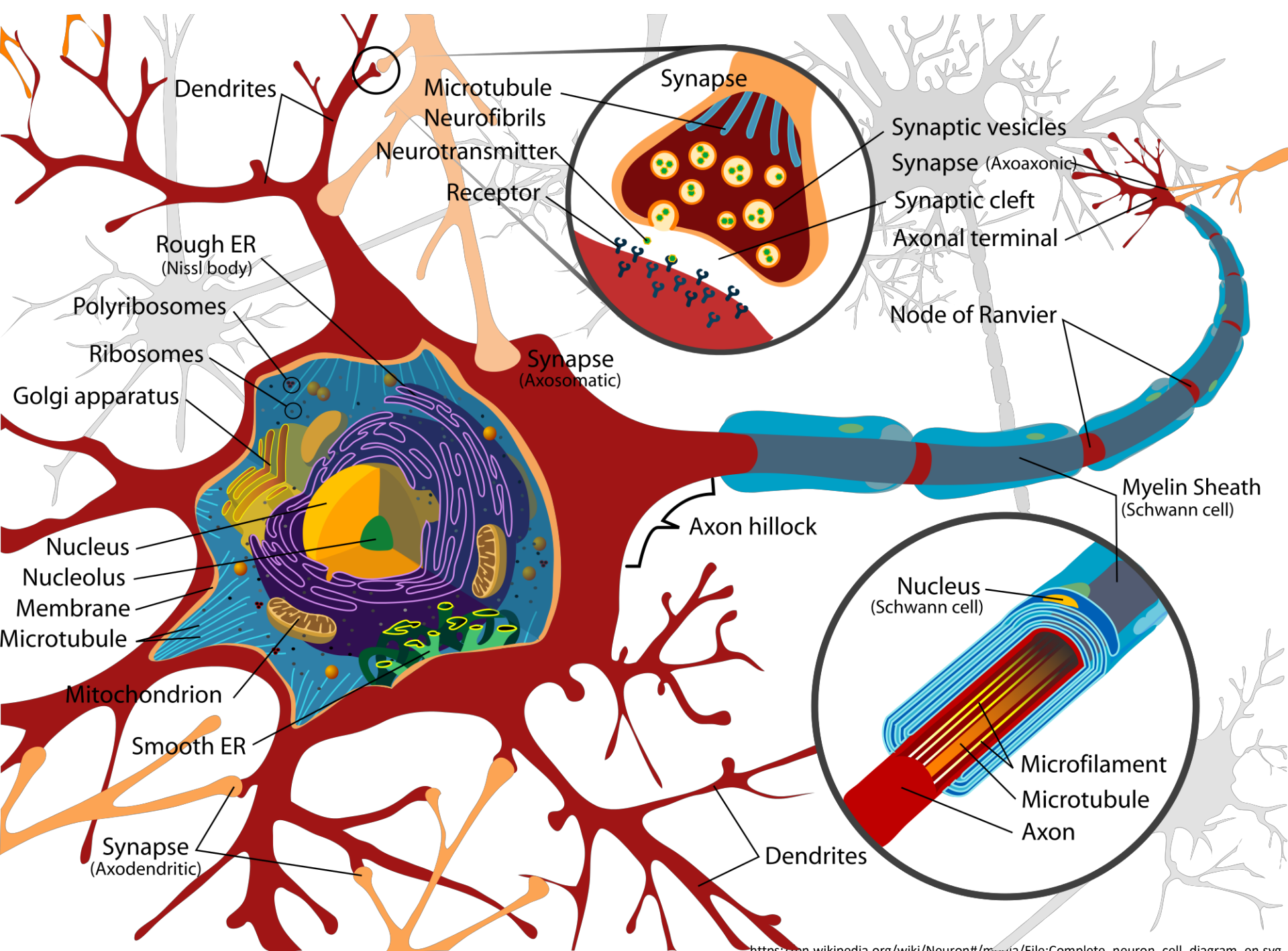


# A neuron is not a transistor



$\neq$

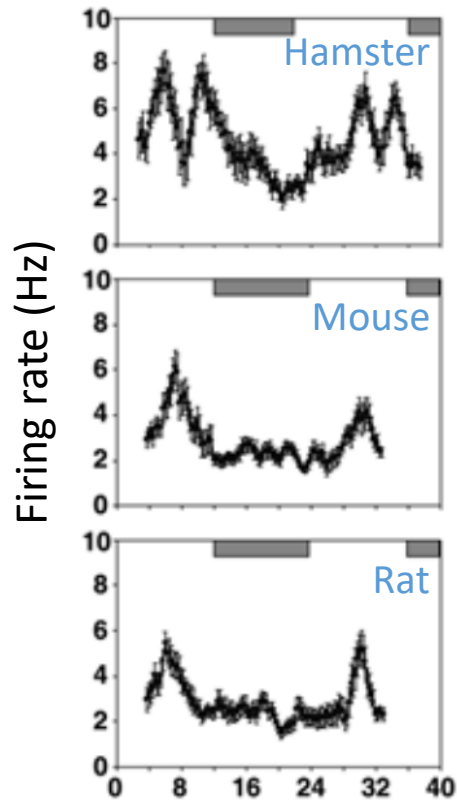




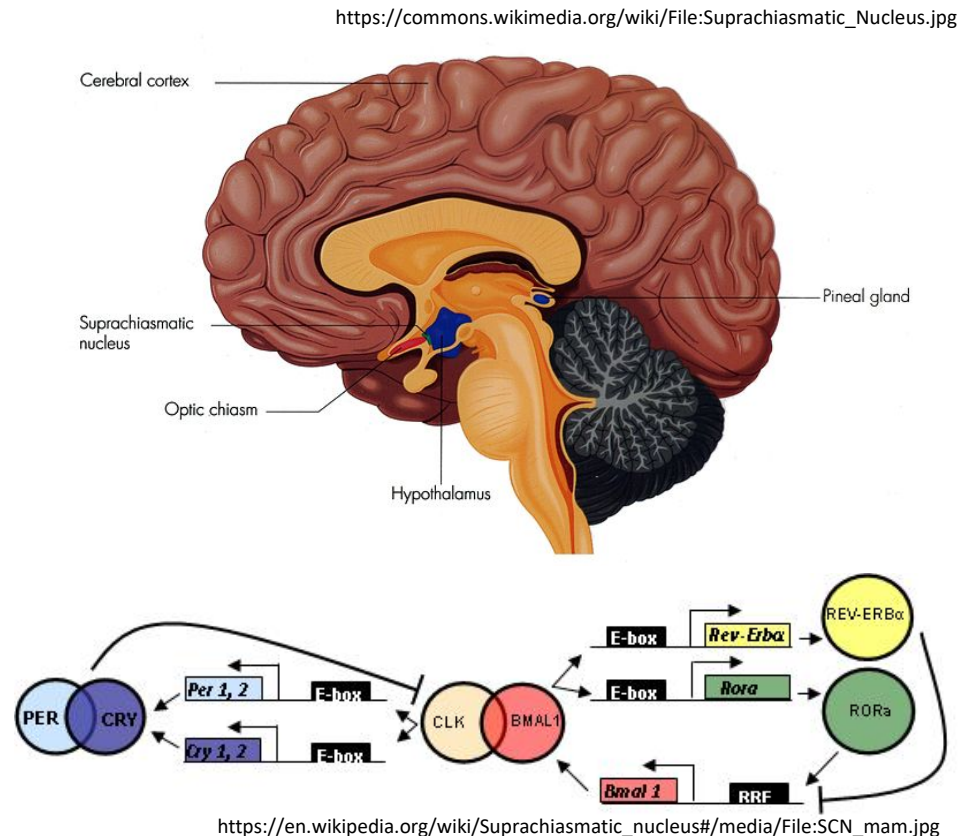


# Neurons have state

(example: protein oscillations in the SCN)

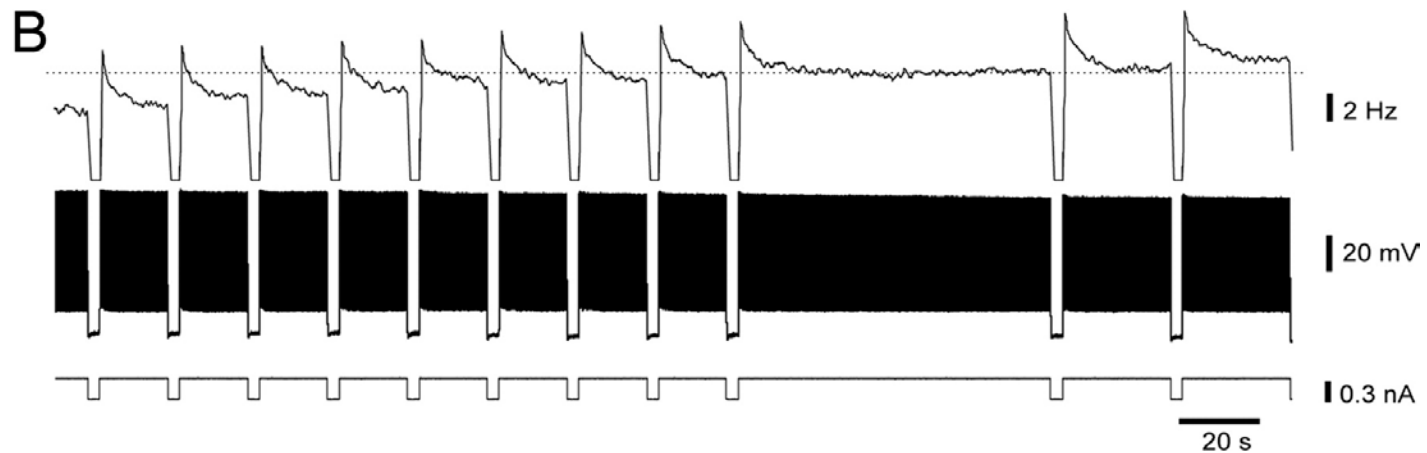
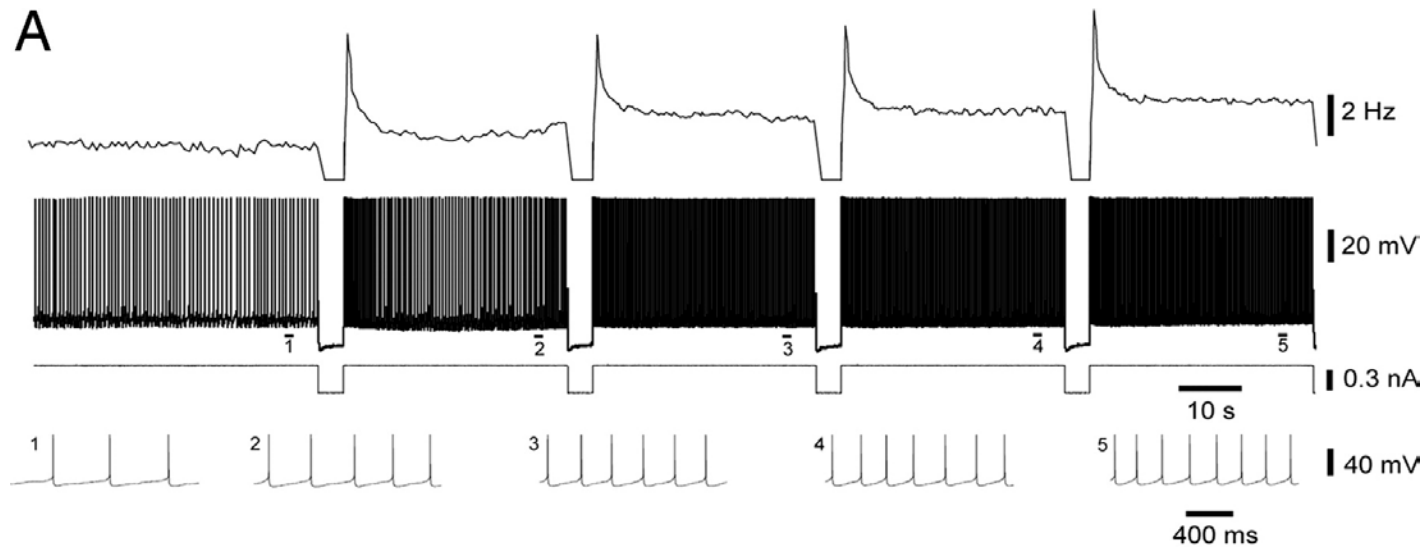


Burgoon et al 2004



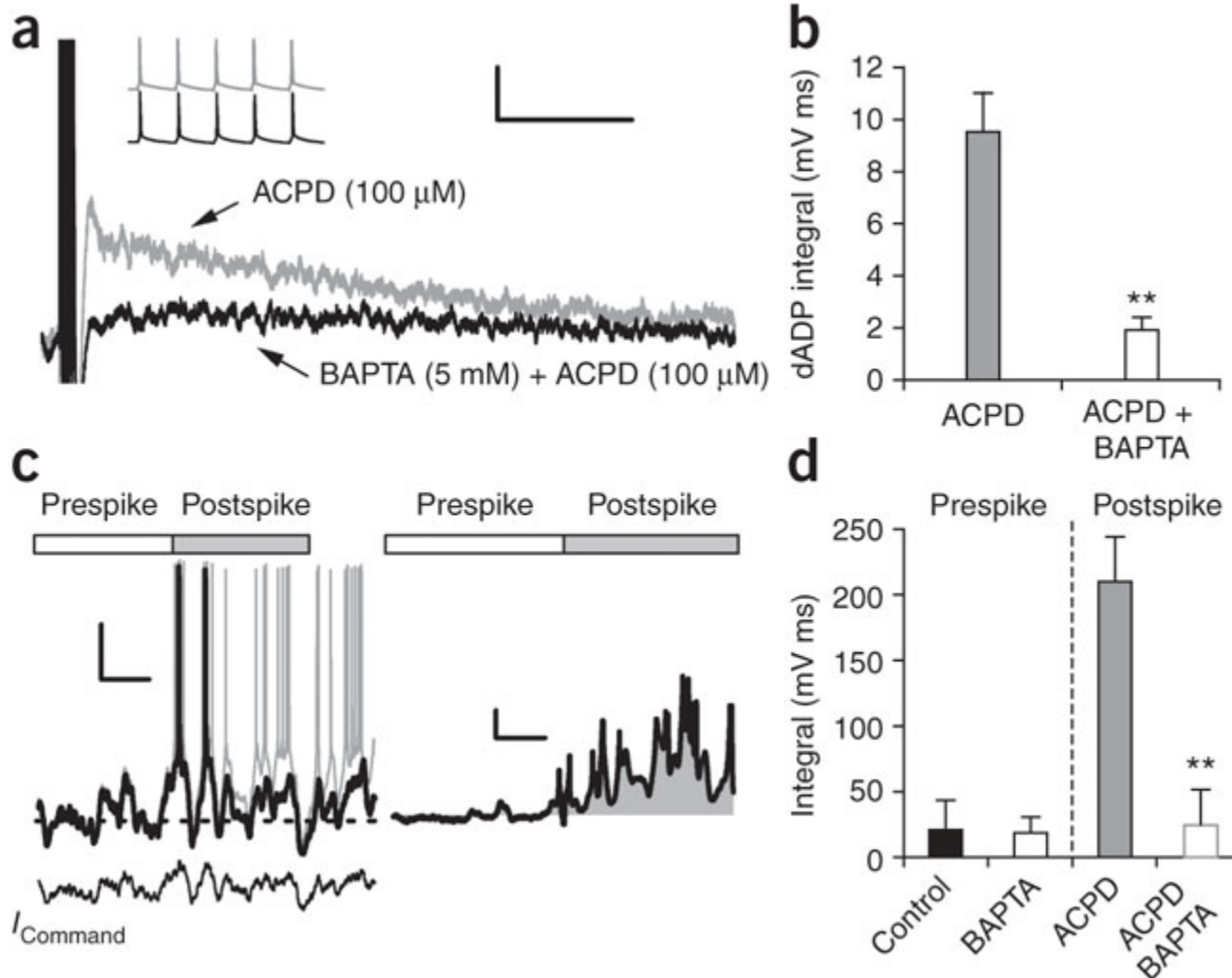
# Neurons have state

(example: HAGPA in PFC)

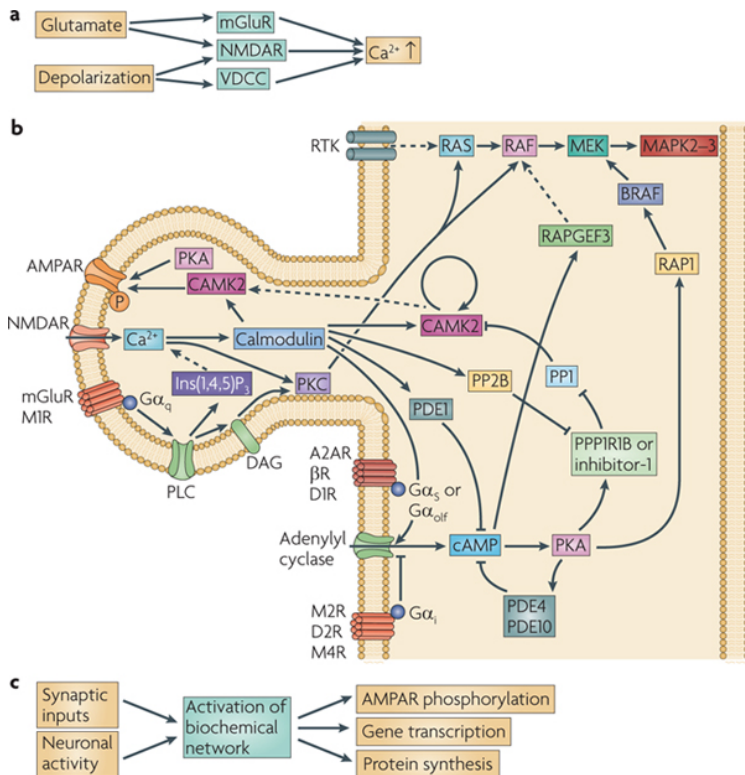


# Neurons have state

(example: intracellular calcium)

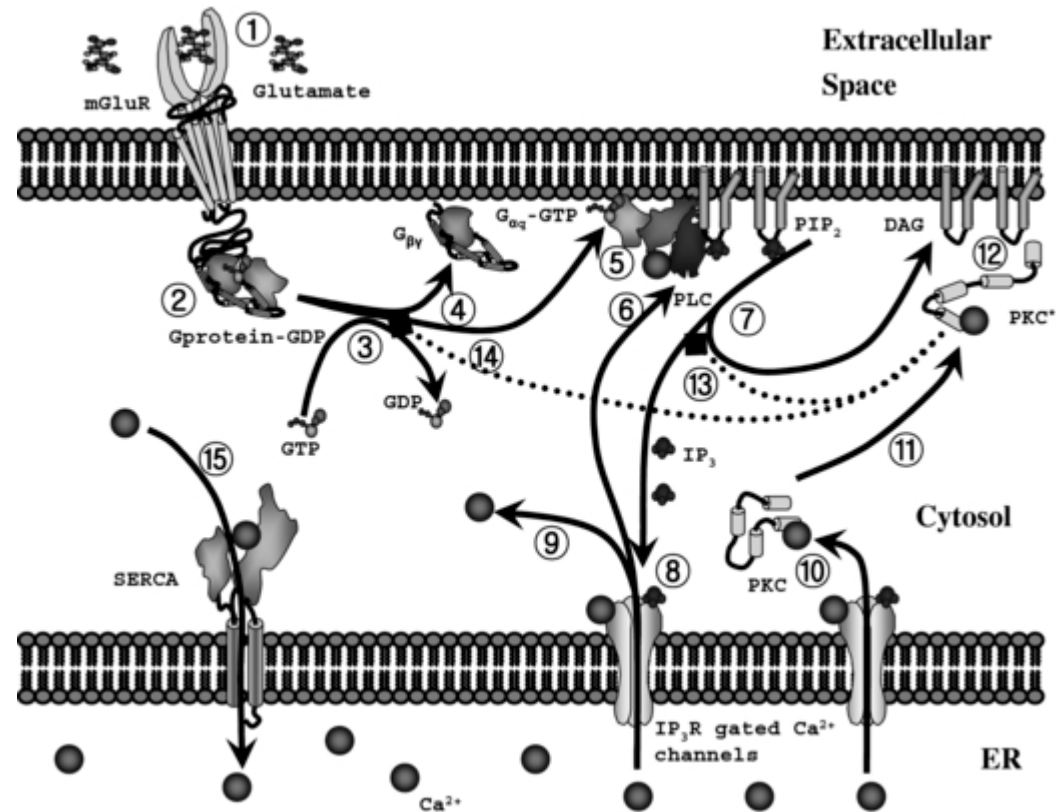


# Neurons have state (example: synaptic pathways)



Nature Reviews | Neuroscience

Jeanette Hellgren Kotaleski & Kim T. Blackwell 2010.



Minchul Kang and Hans G Othmer 2007.

How do we model this?

“**Reaction–diffusion systems** are mathematical models which explain how the **concentration** of one or more substances distributed in space changes under the influence of two processes: **local chemical reactions** in which the substances are transformed into each other, and **diffusion** which causes the substances to spread out over a surface in space.”

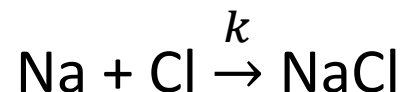
# Mass-Action kinetics

## The model

- A reaction's product is formed at a rate proportional to the concentration of the reactants.

## Example

- Consider the reaction



- Then:

$$[\text{Na}]' = -k[\text{Na}][\text{Cl}]$$

$$[\text{Cl}]' = -k[\text{Na}][\text{Cl}]$$

$$[\text{NaCl}]' = k[\text{Na}][\text{Cl}]$$

### Conservation of mass.

Matter is neither created nor destroyed by reactions.

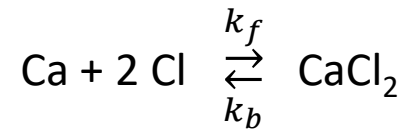
In our equations, this means:

$$[\text{Na}] + [\text{NaCl}] = \text{constant}$$

$$[\text{Cl}] + [\text{NaCl}] = \text{constant}$$

# Example

Using the law of mass-action, we can write a system of equations describing the formation of *calcium chloride*:

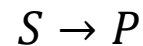


$$\begin{aligned} [\text{Ca}]' &= -k_f [\text{Ca}][\text{Cl}]^2 + k_b [\text{CaCl}_2] \\ [\text{Cl}]' &= -2k_f [\text{Ca}][\text{Cl}]^2 + k_b [\text{CaCl}_2] \\ [\text{CaCl}_2]' &= 2k_f [\text{Ca}][\text{Cl}]^2 - k_b [\text{CaCl}_2] \end{aligned}$$

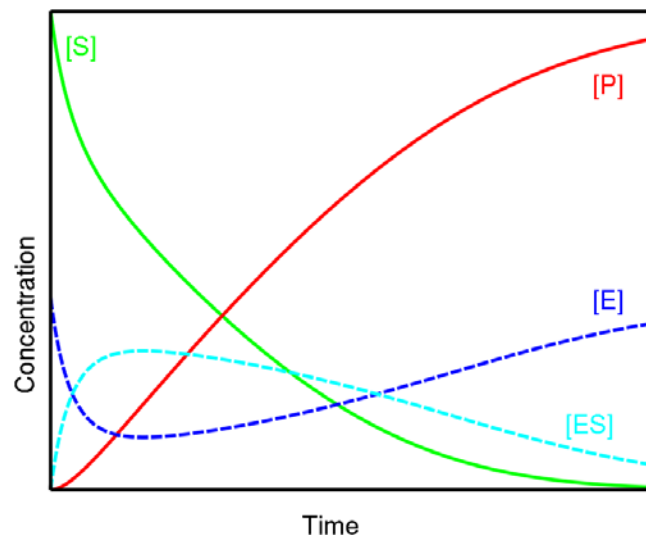
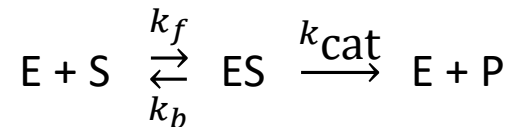


# Enzyme kinetics

It is generally **not** the case that a substrate transforms directly into a product:



Instead, an enzyme is often involved:



# Michaelis-Menten

**If** we can assume either:

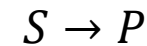
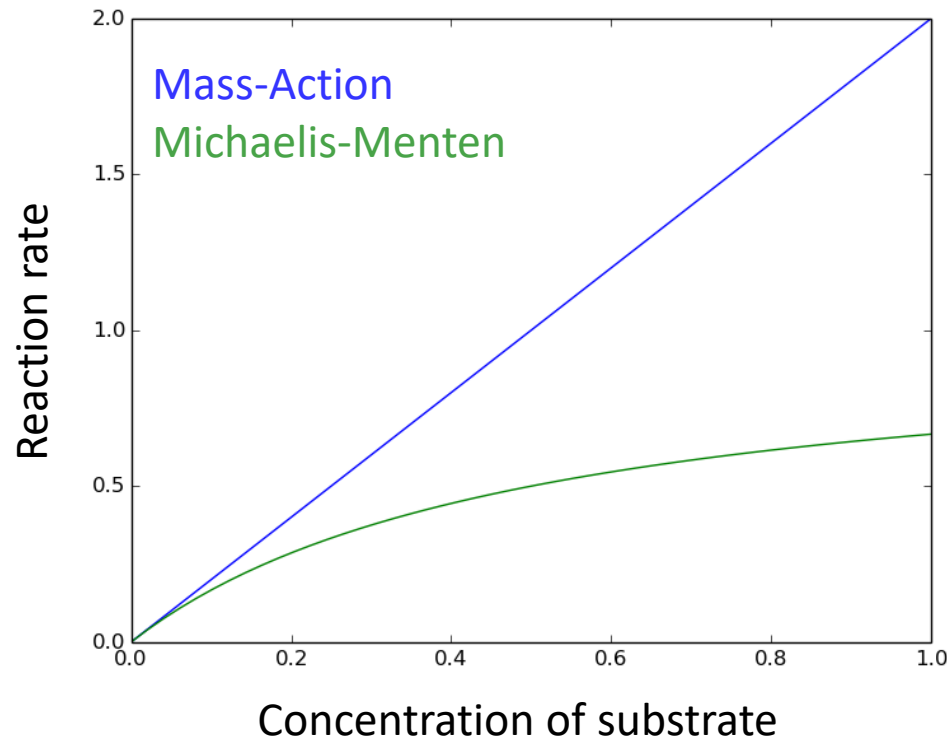
- the substrate (S) and the complex (ES) are in instantaneous equilibrium, or
- the concentration of the complex (ES) does not change on the time-scale of product formation

**Then** the rate of the enzymatic reaction reduces to:

$$\frac{V_{max} [S]}{K_M + [S]}$$

$K_M$  is called the *Michaelis constant*. It is the concentration at which the reaction proceeds at half its maximum rate.

# Michaelis-Menten vs Mass-Action



Both curves on the left have the same rate of reaction when the substrate concentration is low, but the Michaelis-Menten rate levels off (due to limited enzyme availability) as concentrations increase.

$$y = 2x$$

$$y = \frac{x}{x + 0.5}$$

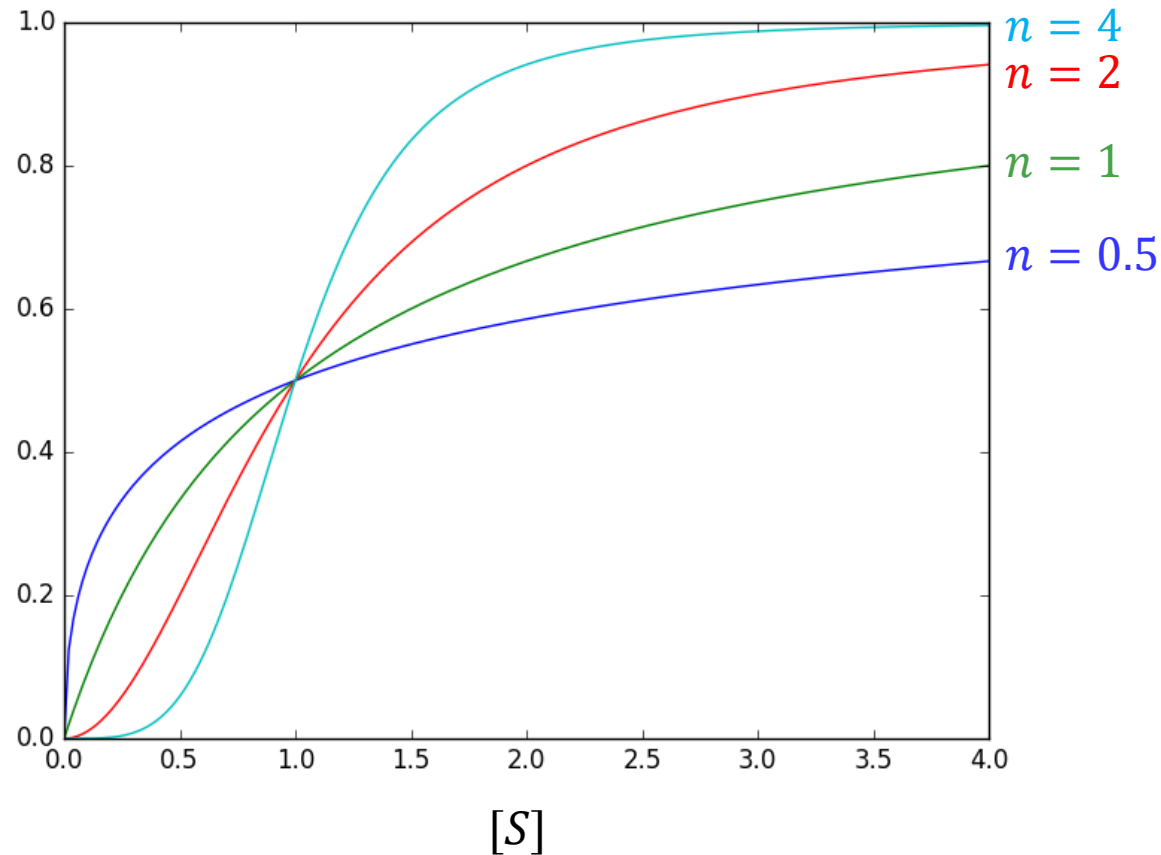
# Hill equation: cooperative binding

$$\frac{V_{max} [S]^n}{[k_A]^n + [S]^n}$$

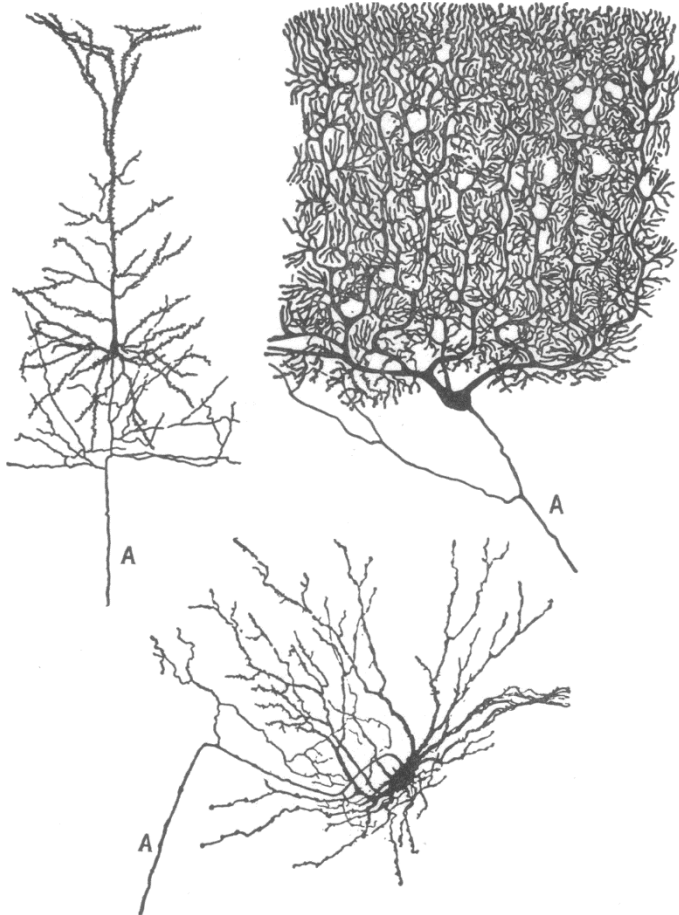
If  $n > 1$ , positive cooperativity.

If  $n < 1$ , negative cooperativity.

$$\frac{[S]^n}{1^n + [S]^n}$$



# Neurons have spatial extent



Cajal 1909 as reproduced in Rall 1962.

## Effects of non-point-ness:

- Ion and protein concentrations vary with space.
- Cellular mechanisms (ER, ion channels, etc) vary with space.

## Concentrations at different locations affect each other:

- Transport
- Diffusion

# Fick's First Law and the diffusion equation

## **Fick's First Law:**

- Diffusive flux is proportional to the concentration gradient.

$$J = -D \nabla \varphi$$

- Here  $D$  is called the *diffusion coefficient*.

## **Fick's Second Law** (the diffusion equation):

$$\frac{\partial \varphi}{\partial t} = \nabla \cdot (D \nabla \varphi) = D \nabla^2 \varphi$$

where the last equality only holds if  $D$  is constant.

# Practical limits of pure diffusion

The expected time  $E[t]$  for a molecule with diffusion constant  $D$  to diffuse a distance  $x$  is:

$$E[t] = \frac{x^2}{2D}$$

So in particular, if

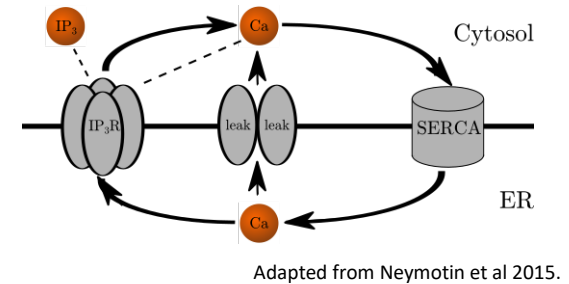
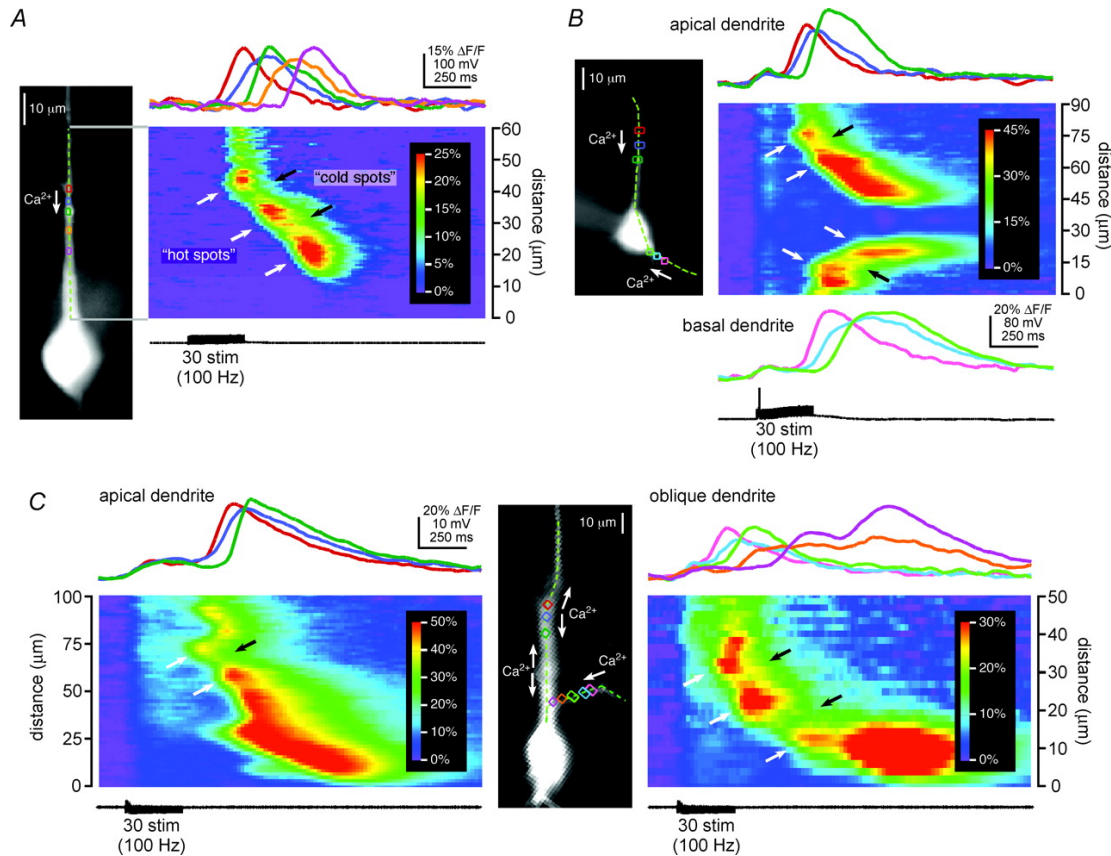
$$D = 1 \mu\text{m}^2/\text{ms} \text{ and}$$

$$x = 100 \mu\text{m},$$

Then

$$E[t] = \frac{100^2}{2} = 5000 \text{ ms}.$$

# Diffusion with regenerative dynamics can quickly spread signals



Fitzpatrick et al 2009.



# Where does diffusion occur?

- Cytosol
  - But not full cross section because of organelles
- Organelles (e.g. ER)
- Extracellular space
  - Tortuosity
  - Anisotropy
  - Volume fraction

# Calcium in spines



<http://dx.doi.org/10.6084/m9.figshare.1266444>

A typical dendritic spine head may have a volume of  **$0.5 \mu\text{m}^3$** .

A typical cytosolic calcium concentration is  **$100 \text{ nM}$** .

At these levels, how many molecules of calcium are in a dendritic spine head? What is the percentage change in concentration if one molecule leaves the spine head?

# Reaction-diffusion in NEURON

# Why use NEURON's rxd module?

## Reduces typing

- In 2 lines: declare a domain, then declare a molecule, allowing it to diffuse and respond to flux from ion channels.  
    `all = rxd.Region(h.allsec(), nrn_region= 'i')`  
    `ca = rxd.Species(all, name= 'ca', d= 1, charge= 2)`
- **Reduces** the risk for **errors** from typos or misunderstandings.

## Allows arbitrary domains

NEURON traditionally only identified concentrations just inside and just outside the plasma membrane. The rxd module allows you to **declare your own regions** of interest (e.g. ER, mitochondria, etc).

# rxn module overview

terministic vs stochastic.



# Declare a region: `rxd.Region`

## Basic Usage

```
cyt = rxd.Region(seclist)
```

`seclist` may be any iterable of sections; e.g. a `SectionList` or a Python list.

## Identify with a standard region

```
cyt = rxd.Region(seclist, nrn_region='i')
```

`nrn_region` may be `i` or `o`, corresponding to the locations of e.g. `nai` vs `nao`.

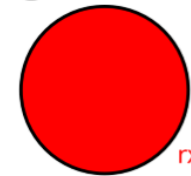
## Specify the cross-sectional shape

```
cyt = rxd.Region(seclist, geometry=rxd.Shell(0.5, 1))
```

The default geometry is `rxd.inside`.

The `geometry` and `nrn_region` arguments may both be specified.

geometry:



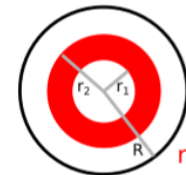
`rxd.inside`



`rxd.membrane`



`rxd.FractionalVolume(  
volume_fraction=f1,  
surface_fraction=f2)`



`rxd.Shell(r1/R, r2/R)`

Adapted from:  
McDougal et al 2013.

# rxd.Region tips

## Specify `nrn_region` if concentrations interact with NMODL

If NMODL mechanisms (ion channels, point processes, etc) depend on or affect the concentration of a species living in a given region, that region must declare a `nrn_region` (typically 'i').

## To declare a region that exists on all sections

```
r = rxd.Region(h.allsec())
```

## Use list comprehensions to select sections

```
r = rxd.Region([sec for sec in h.allsec() if 'apical' in sec.name()])
```

# Declare ions & proteins: rxd.Species

## Basic usage

```
protein = rxd.Species(region, d=16)
```

$d$  is the **diffusion constant** in  $\mu\text{m}^2/\text{ms}$ .  $\text{region}$  is an `rxd.Region` or an iterable of `rxd.Region` objects.

## Initial conditions

```
protein = rxd.Species(region, initial=value)
```

$\text{value}$  is in mM. It may be a constant or a function of the node.

## Connecting with HOC

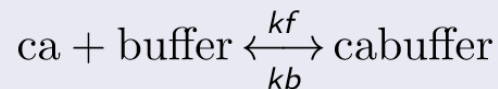
```
ca = rxd.Species(region, name='ca', charge=2)
```

If the `nrn_region` of `region` is `"i"`, the concentrations of this species will be stored in `cai`, and its concentrations will be affected by `ica`.



# Specifying dynamics: rxd.Reaction

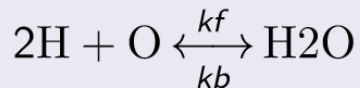
## Mass-action kinetics



`buffering = rxd.Reaction(ca + buffer, cabuffer, kf, kb)`

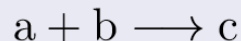
kf is the forward reaction rate, kb is the backward reaction rate. kb may be omitted if the reaction is unidirectional.  
In a mass-action reaction, the reaction rate is proportional to the product of the concentrations of the reactants.

## Repeated reactants



`water_reaction = rxd.Reaction(2 * H + O, H2O, kf, kb)`

## Arbitrary reaction formula, e.g. Hill dynamics



`hill_reaction = rxd.Reaction(a + b, c, a ^ 2 / (a ^ 2 + k ^ 2), mass_action=False)`

Hill dynamics are often used to model cooperative reactions.

# rxd.Rate and rxd.MultiCompartmentReaction

## rxd.Rate

Use `rxd.Rate` to specify an explicit contribution to the rate of change of some concentration or state variable.

```
ip3degradation = rxd.Rate(ip3, -k * ip3)
```

## rxd.MultiCompartmentReaction

Use `rxd.MultiCompartmentReaction` when the dynamics span multiple regions; e.g. a pump or channel.

```
ip3r = rxd.MultiCompartmentReaction(ca[er], ca[cyt], kf, kb,  
                                     membrane=cyt_er_membrane)
```

The rate of these dynamics is proportional to the membrane area.

# Manipulating nodes

## Getting a list of nodes

- `odelist = protein.nodes`

## Filtering a list of nodes

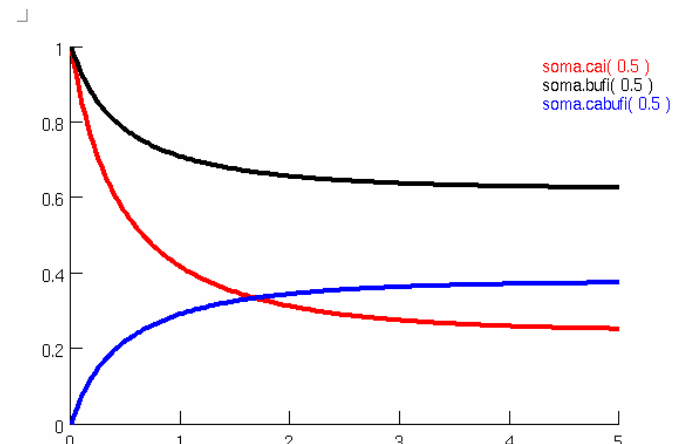
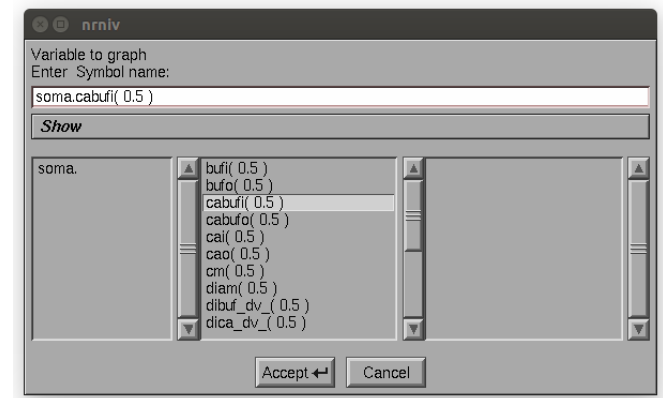
- `odelist2 = oodelist(region)`
- `odelist2 = oodelist(0.5)`
- `odelist2 = oodelist(section)(region)(0.5)`

## Other operations

- `odelist.concentration = value`
- `values = oodelist.concentration`
- `surface_areas = oodelist.surface_area`
- `volumes = oodelist.volume`
- `node = oodelist[0]`

# Example: Calcium buffering

Use the GUI to create a graph and run the simulation.



abuf, 1, 0.1)

# Concentration pointers

To get a pointer to a concentration, use `node._ref_concentration`:

## Recording traces

```
v = h.Vector()
v.record(ca.nodes[0]._ref_concentration)
```

## Plotting

```
g = h.Graph()
g.addvar('ca[er][dend](0.5)', ca.nodes(er)(dend)(0.5)[0]._ref_concentration)
h.graphList[0].append(g)
```

# Tips

To find out what properties and methods are available, use `dir`; e.g.

```
dir(ca.nodes)
```

NEURON's variable step solver has a default absolute tolerance of 0.001.

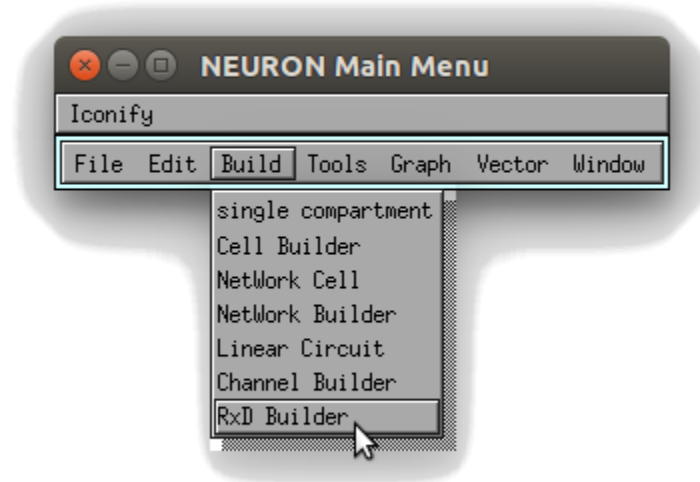
Since NEURON measures concentration in mM and some cell biology concentrations (e.g. calcium) are in  $\mu\text{M}$ , this tolerance may be too high. Compensate by using an `atolscale` in the constructor\*, e.g.

```
ca = h.Species(cyt, atolscale=1e-6)
```

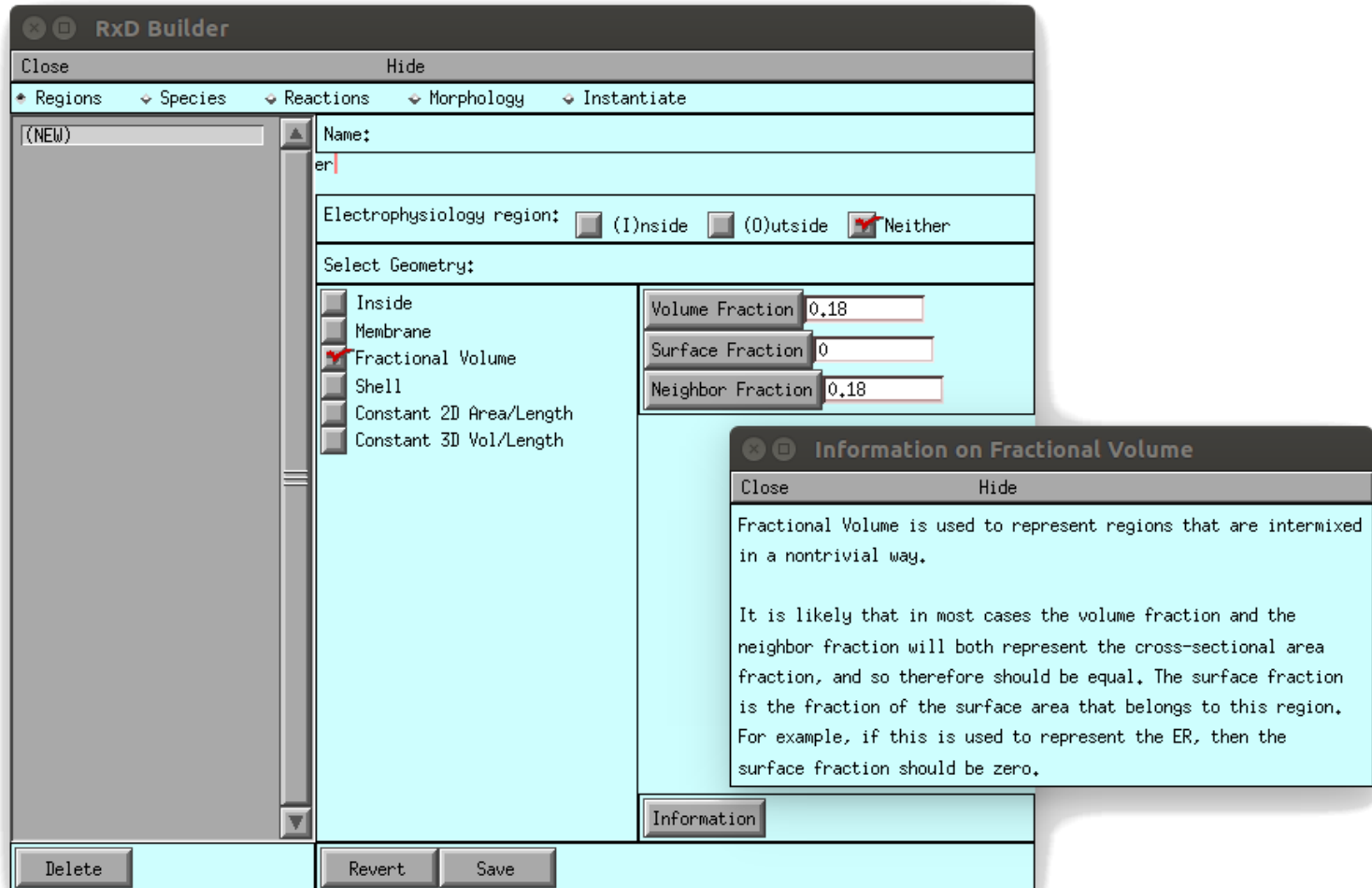
\* `atolscale` is only supported in the development version; on older versions of NEURON, change the scale globally, e.g. `h.Cvode().atol(1e-8)`.

# GUI-based specification

Reaction-diffusion dynamics can also be specified using the GUI. This option appears only when rxd is supported in your install (Python and scipy must be available).

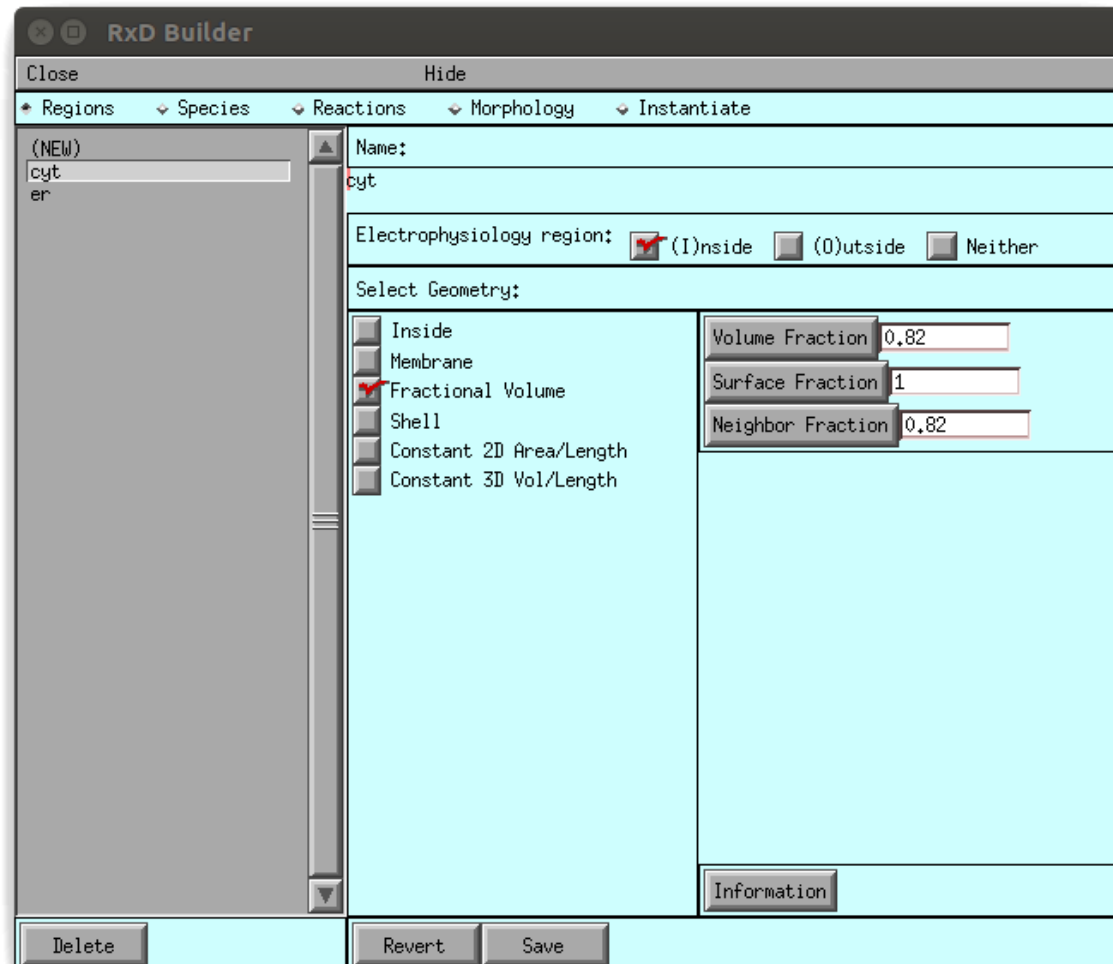


# GUI-based specification

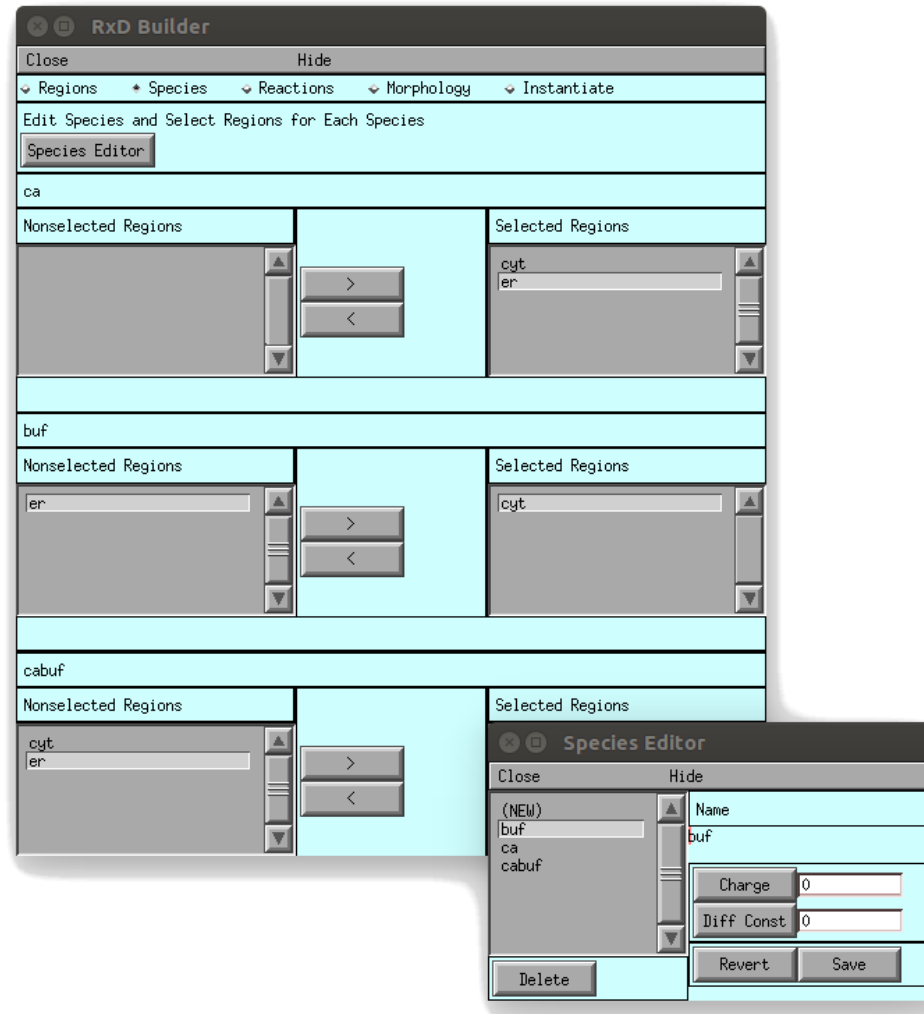




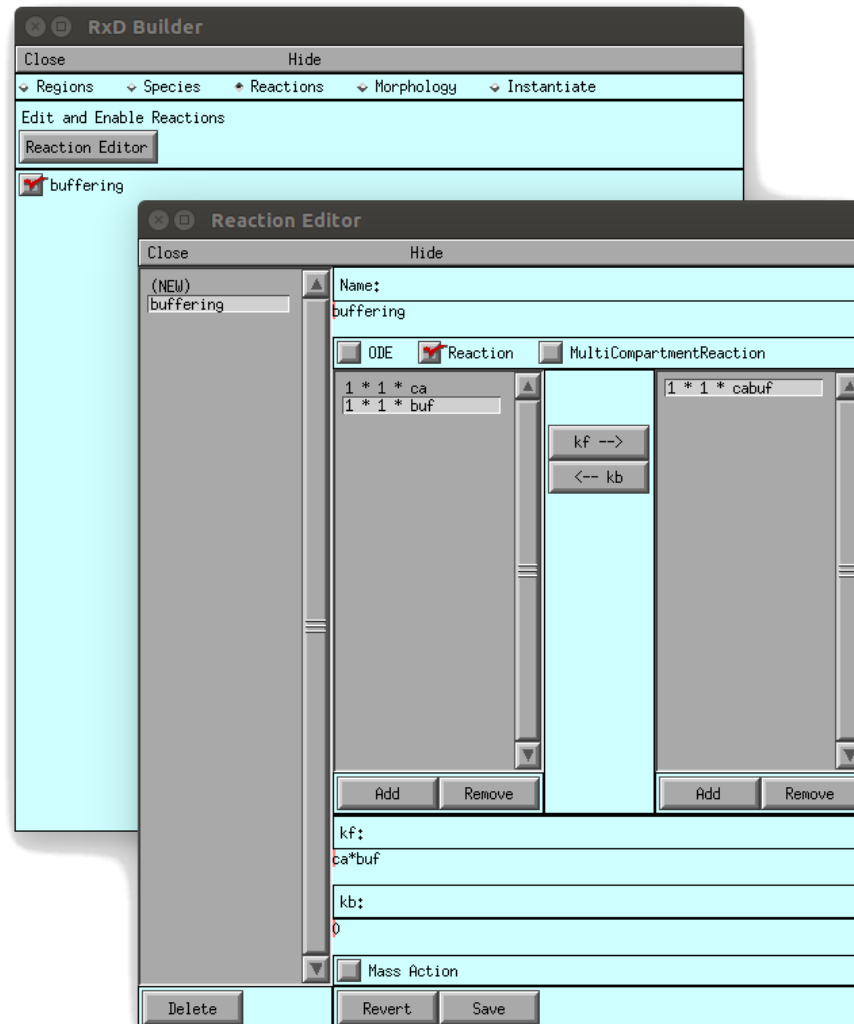
# GUI-based specification



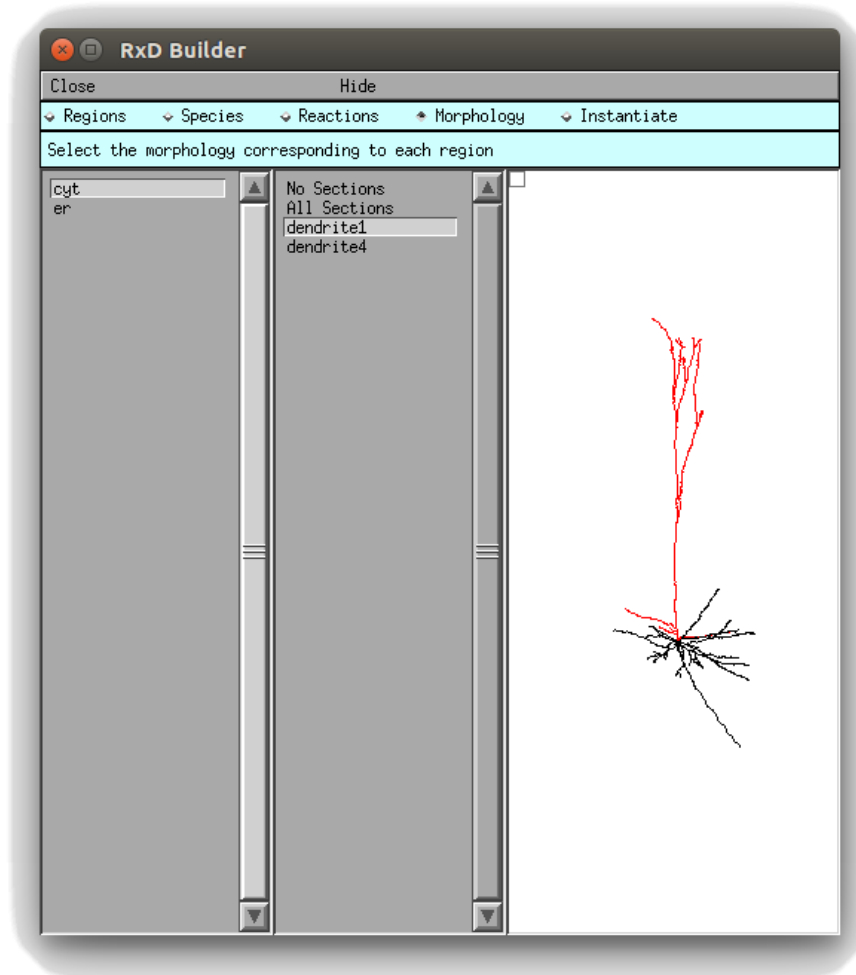
# GUI-based specification



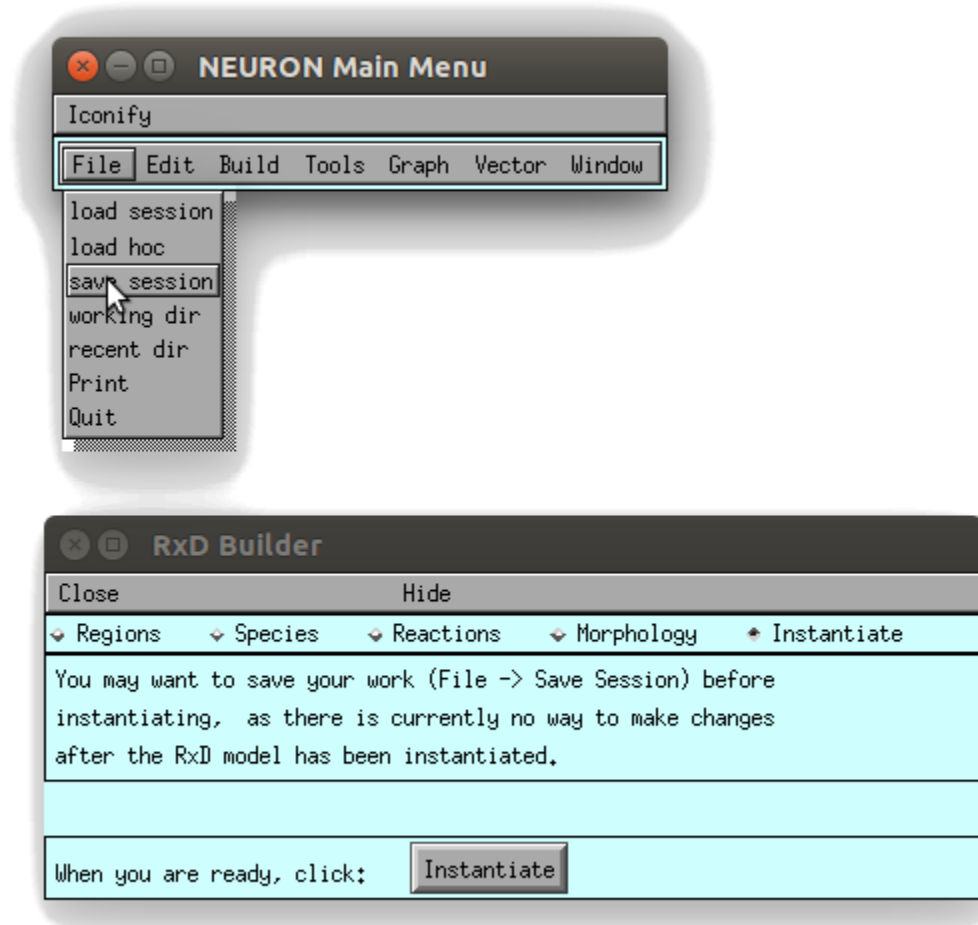
# GUI-based specification



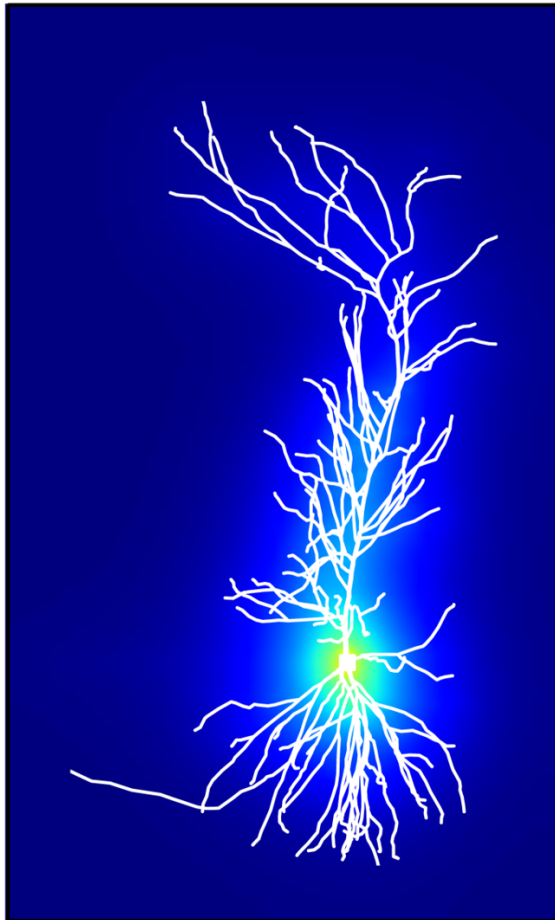
# GUI-based specification



# GUI-based specification



# Extracellular diffusion\*

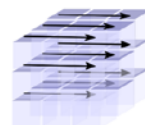


New region type:

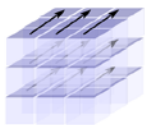
```
ecs = rxd.Extracellular(xlo, ylo, zlo, xhi, yhi, zhi,
                        dx=dx, tortuosity=1, volume_fraction=1)
```

Setting/getting extracellular concentrations:

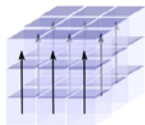
```
ca[ecs].states3d[5:15, 5:15, :] = 1
pyplot.imshow(ca[ecs].states3d[:, :, 0],
               interpolation='nearest', vmin=0, vmax=1,
               extent=ca[ecs].extent('xy'), origin='lower')
```



$$1 - \frac{r_x}{2} \nabla_x^2 \phi^{(t+\frac{1}{3})} = \left( \frac{r_x}{2} \nabla_x^2 + r_y \nabla_y^2 + r_z \nabla_z^2 \right) \phi^{(t)}$$



$$1 - \frac{r_y}{2} \nabla_y^2 \phi^{(t+\frac{2}{3})} = -\frac{r_y}{2} \nabla_y^2 \phi^{(t+\frac{1}{3})}$$



$$1 - \frac{r_z}{2} \nabla_z^2 \phi^{(t+1)} = -\frac{r_z}{2} \nabla_z^2 \phi^{(t+\frac{2}{3})}$$

We use a finite-volume method, the Douglas-Gunn Alternating Direction Implicit algorithm is **unconditionally stable**.

Each time-step is divided into an x-, y- and z-direction and requires solving diagonally dominant tridiagonal systems of equations. This is solved with the **Thomas algorithm**, so the runtime scales linearly with the number of voxels.

We currently support zero-flux Neumann boundary conditions which conserves the total concentration.

\* Extracellular diffusion support is currently only available in the alpha version.

# 3D Simulations

## Specifying 3D Simulations

<sup>2</sup> `rx.d.set_solve_type` can optionally take a list of sections as its first argument; in that case only the specified sections will be simulated in three dimensions.



# Example: wave curvature

```
from neuron import h, gui, rxd
import volume_slicer

sec1, sec2 = h.Section(), h.Section()
h.pt3dadd(2, 0, 0, 2, sec=sec1)
h.pt3dadd(9.9, 0, 0, 2, sec=sec1)
h.pt3dadd(10, 0, 0, 2, sec=sec1)
h.pt3dadd(10, 0, 0, 10, sec=sec2)
h.pt3dadd(18, 0, 0, 10, sec=sec2)

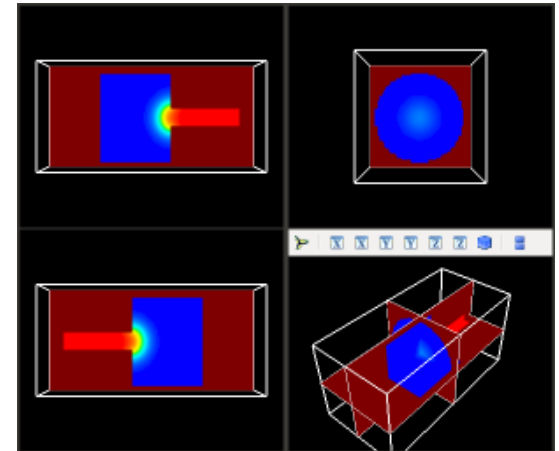
def do_init(node):
    return 1 if node.x3d < 8 else 0

all3d = rxd.Region(h.allsec(), dimension=3)
ca = rxd.Species(all3d, initial=do_init, d=0.05)
r = rxd.Rate(ca, -ca * (1 - ca) * (0.1 - ca))

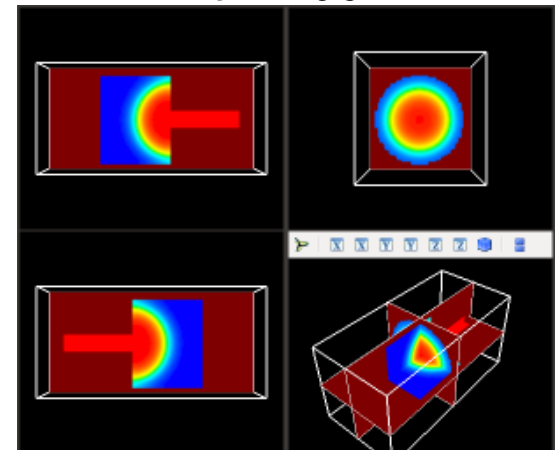
def plot_it():
    graph = volume_slicer.VolumeSlicer(
        data=ca.nodes.value_to_grid(),
        vmin=0, vmax=1)
    graph.configure_traits()

h.finitialize()
for t in [30, 60]:
    h.continuerun(t)
    plot_it()
```

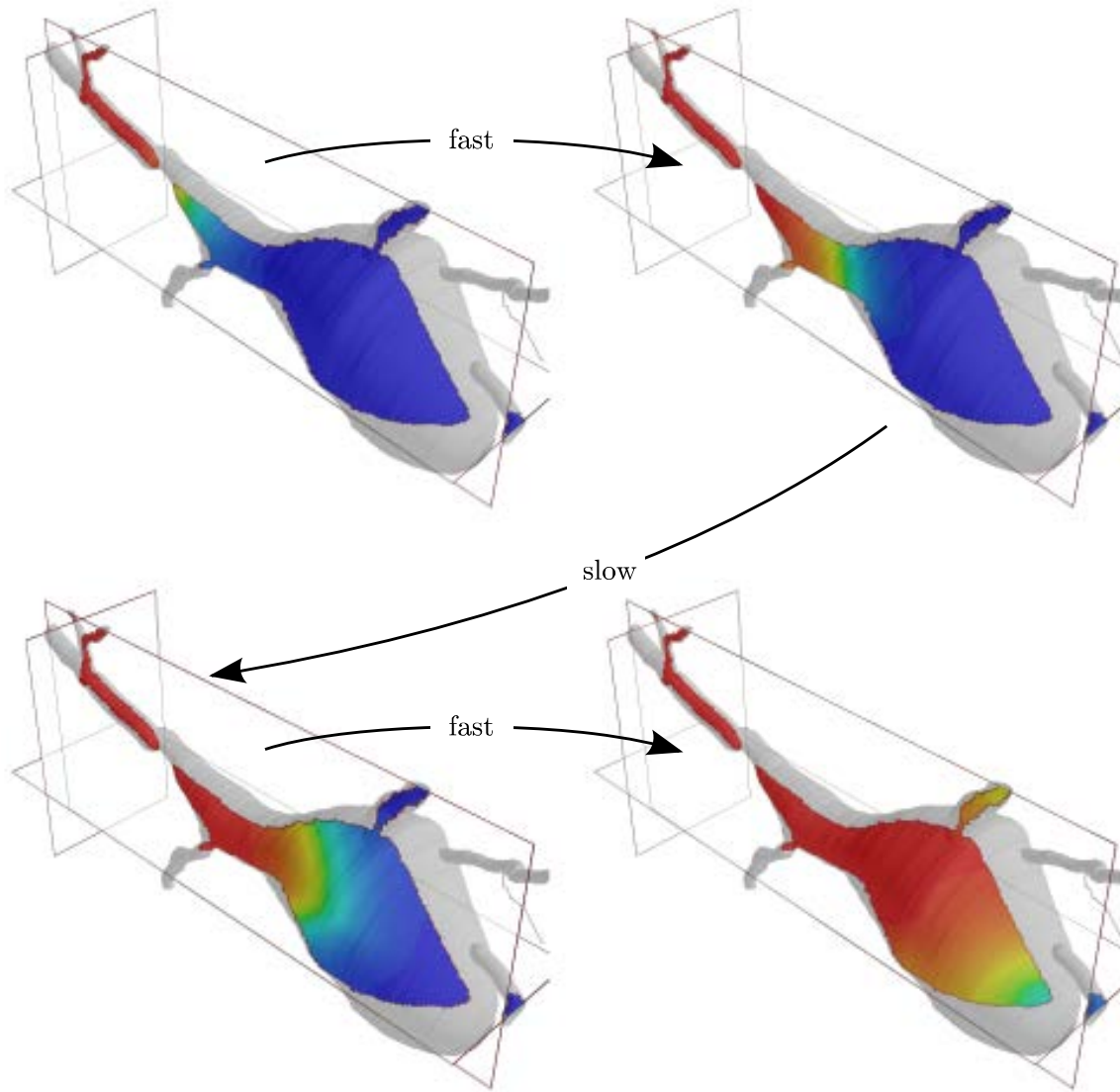
t = 30



t = 60



# Wave curvature at soma entry



# Under development

- Enhancements to extracellular diffusion.
- Stochastic reaction-diffusion.
- SBML support.
- Better reaction-diffusion performance.
- Parallel reaction-diffusion.

Contact us if you would like to alpha test any of these features.

# For more information

Journal Articles on Reaction-Diffusion in NEURON

ammer's Reference

