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

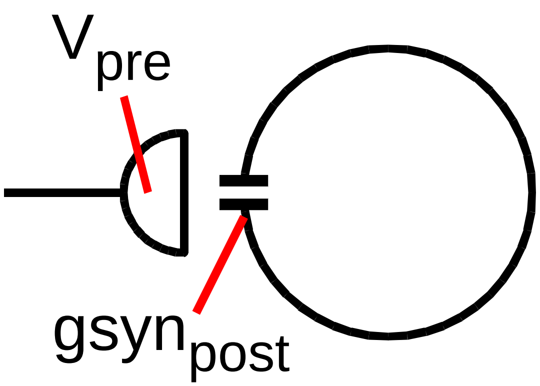
- graded
- spike-triggered

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*

Link postsynaptic variable to the presynaptic variable with a POINTER

NMODL specification of synaptic mechanism:

```
NEURON {
    POINT_PROCESS Syn
    POINTER vpre
}
```

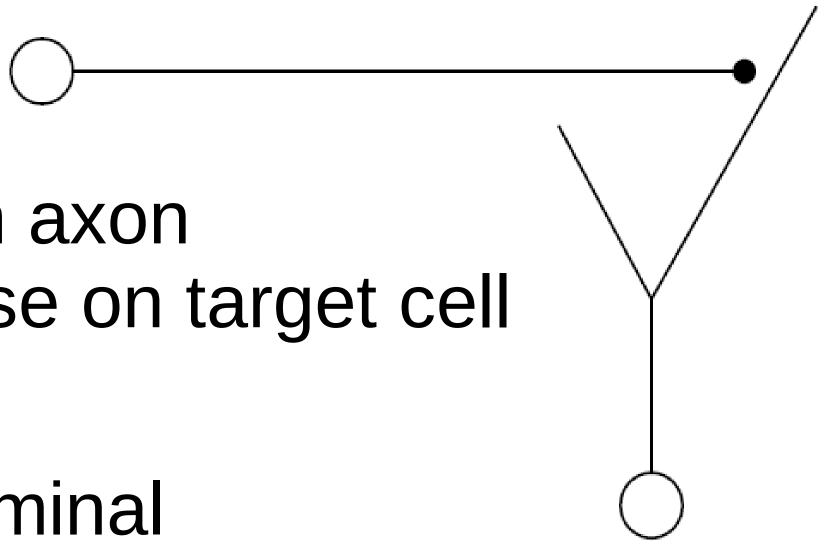
hoc usage

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

Python usage

```
syn = h.Syn(dend(0.5))
syn._ref_vpre = precell.axon(1)._ref_v
```

Spike-triggered synaptic transmission



Physical system:

Presynaptic neuron with axon
that projects to synapse on target cell

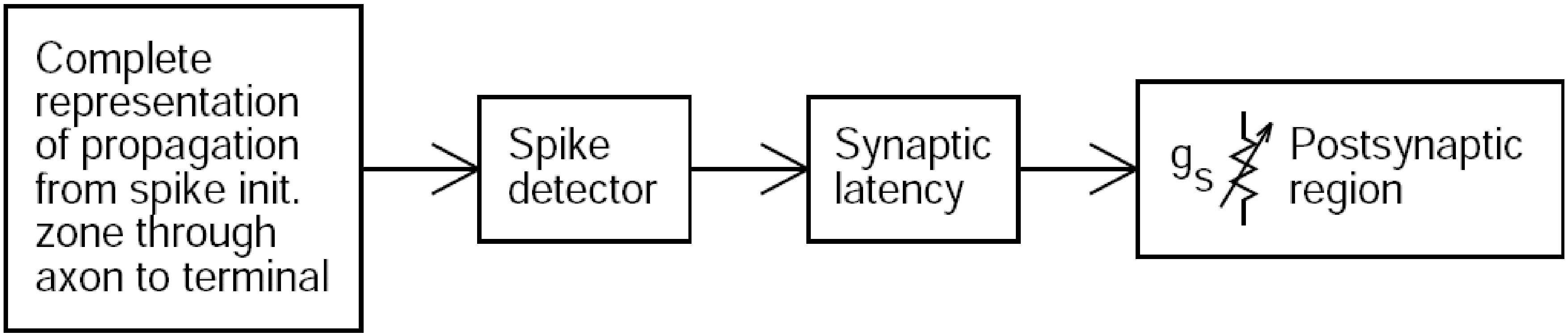
Conceptual model:

Spike in presynaptic terminal
triggers transmitter release;
presynaptic details unimportant

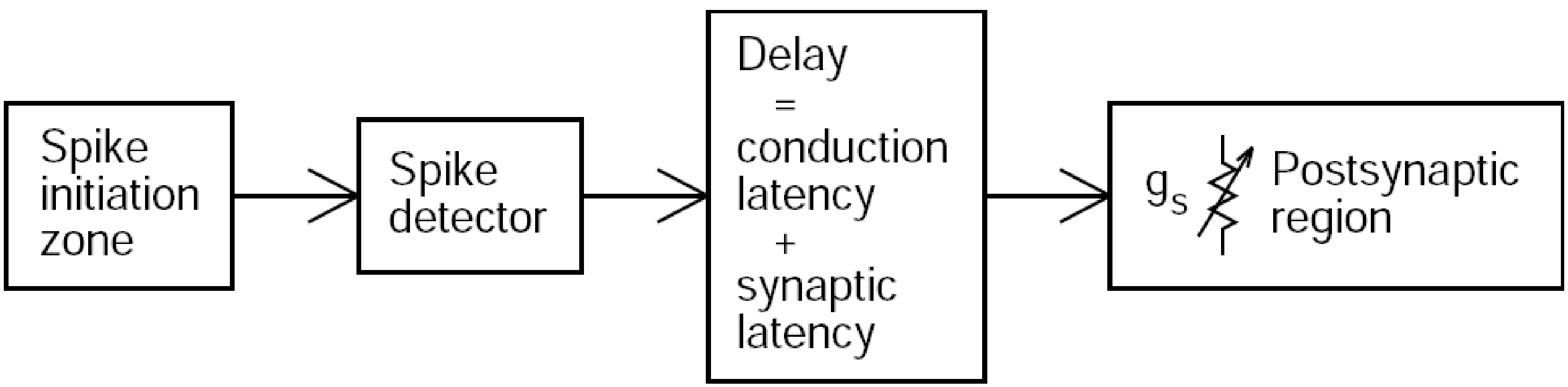
Postsynaptic effect described by
DE or kinetic scheme that is perturbed by
occurrence of a presynaptic spike

Spike-triggered transmission: computational implementation

Basic idea



More efficient: "virtual spike propagation"



The NetCon class

Python usage

```
nc = h.NetCon(source, target)
nc = h.NetCon(source_ref_v, target
     [, threshold, delay, weight],
     sec = section)
```

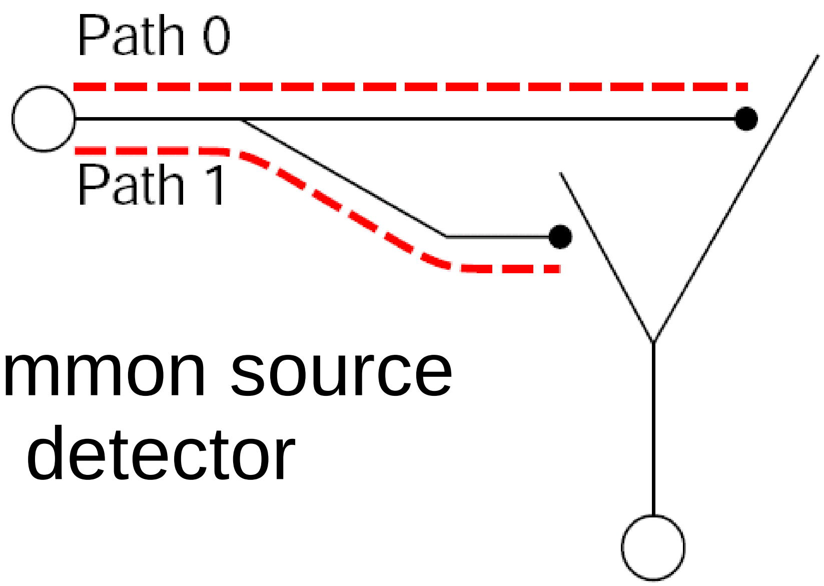
Defaults

```
nc.threshold = 10
nc.delay = 1 # must be >= 0
nc.weight[0] = 0 # weight is an array
```

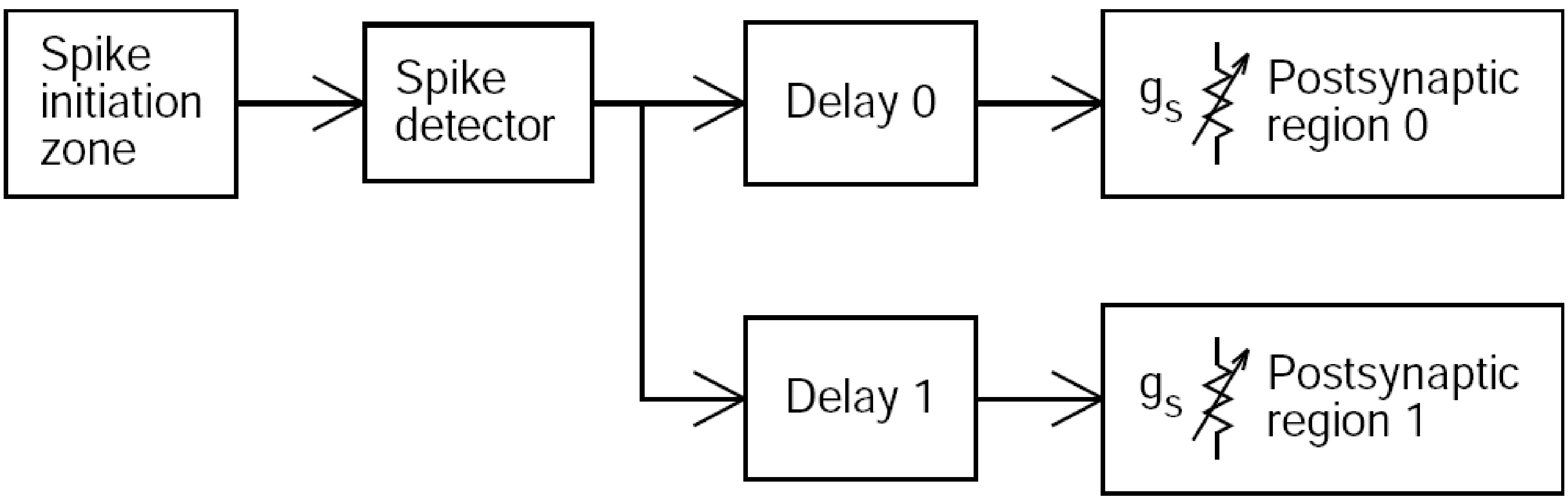
NMODL specification of synaptic mechanism

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

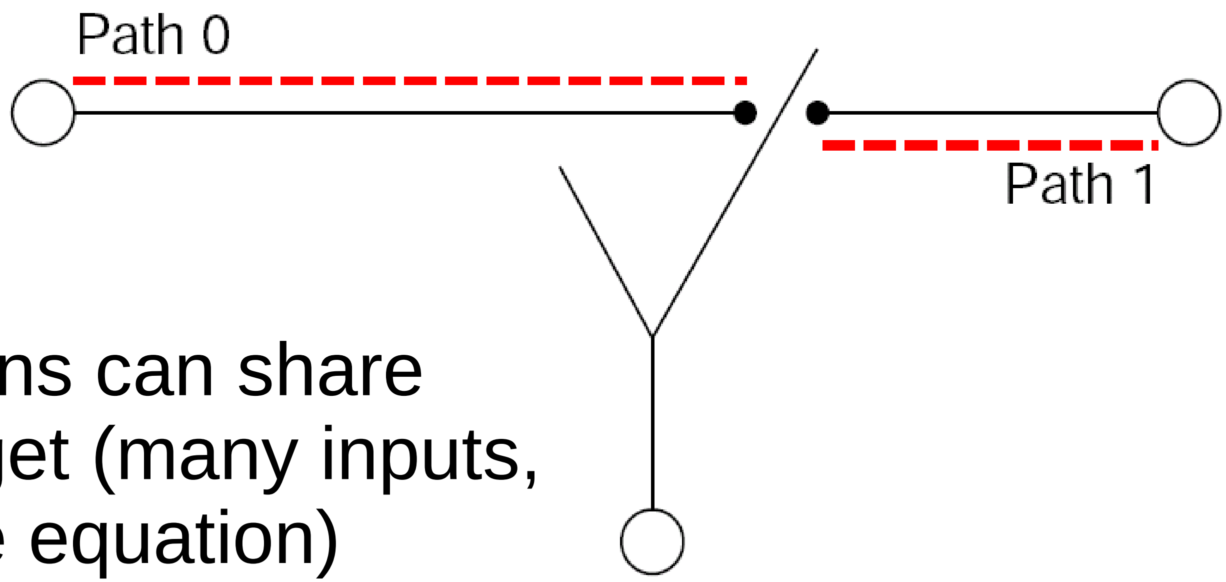
Efficient divergence



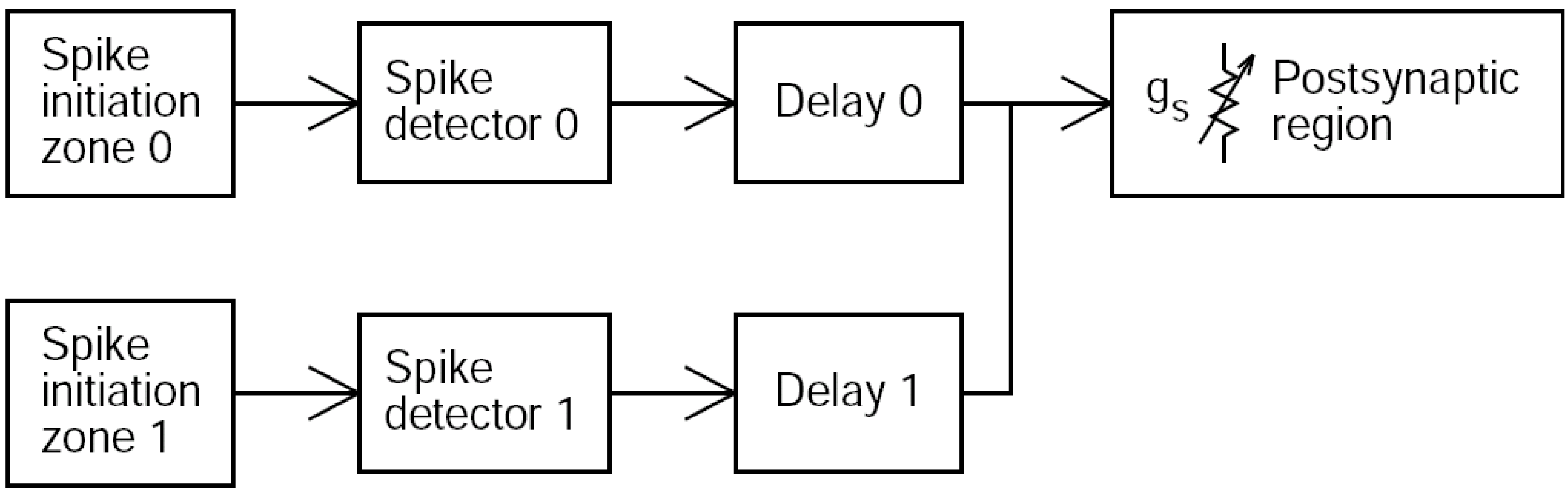
Multiple NetCons with a common source share a single threshold detector



Efficient convergence



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

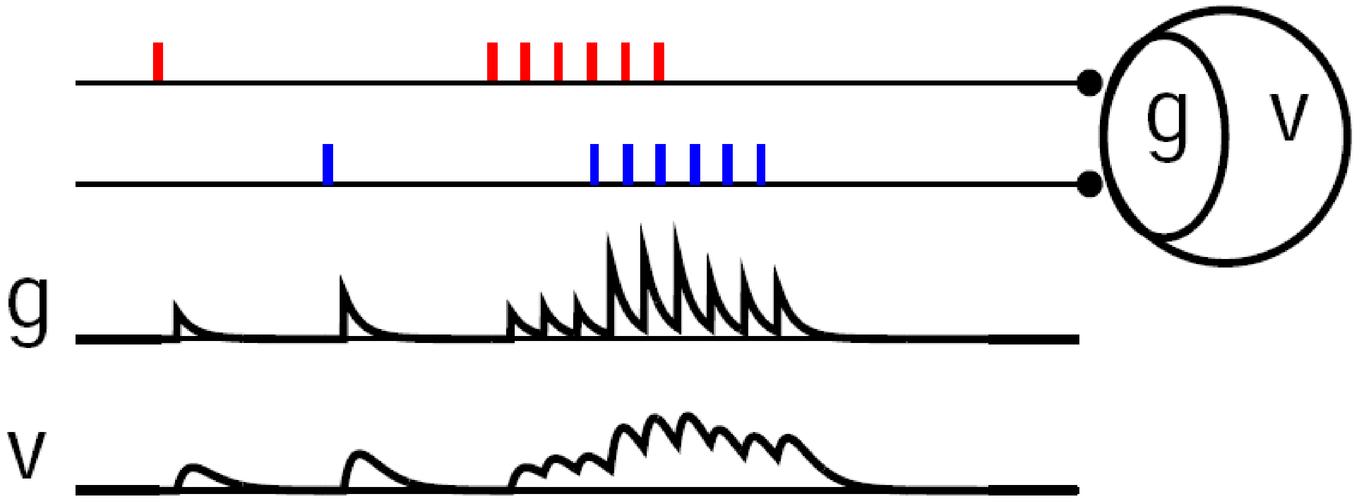


Example: g_s with fast rise and exponential decay

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

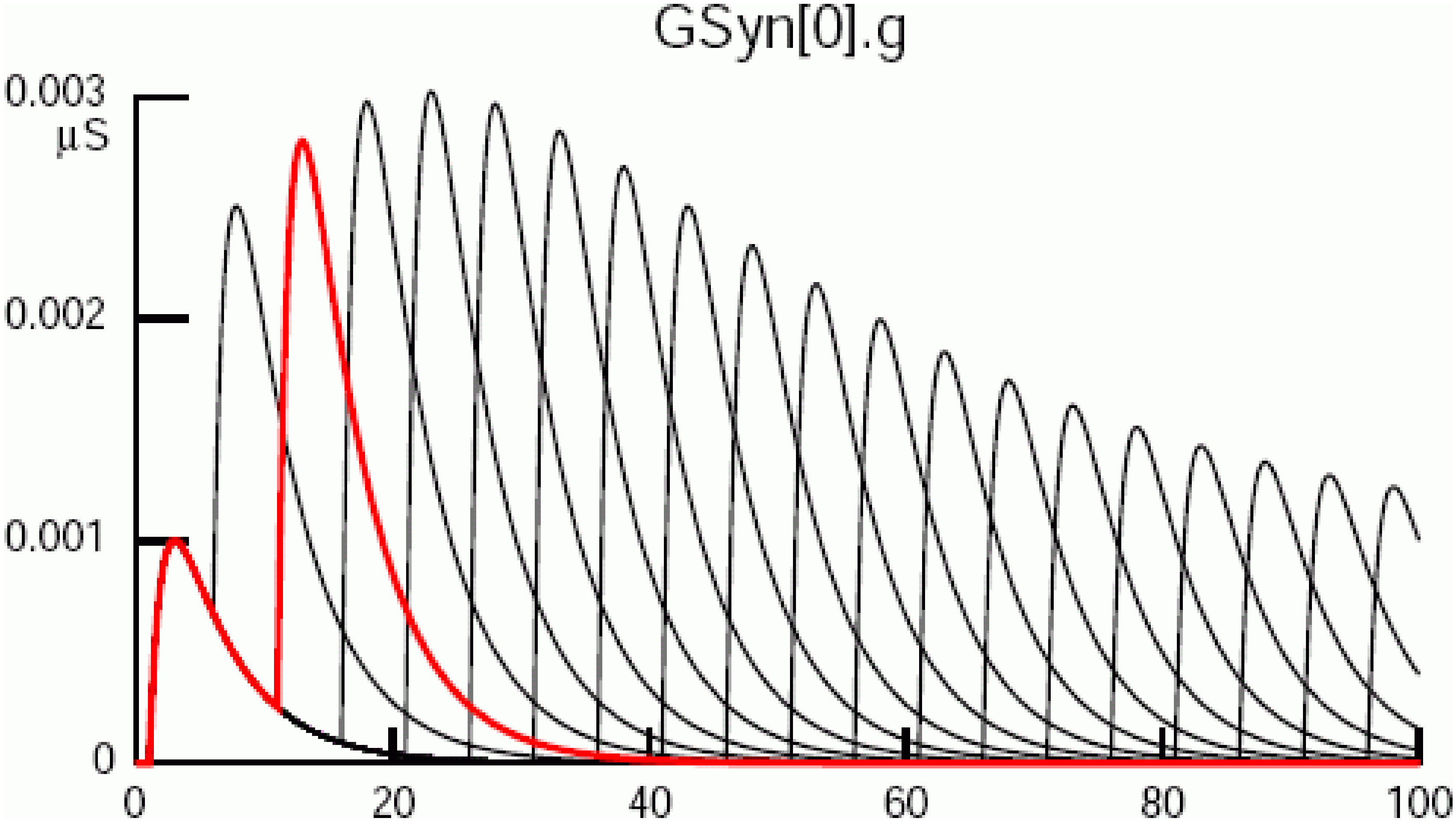
g_s with fast rise and exponential decay

continued



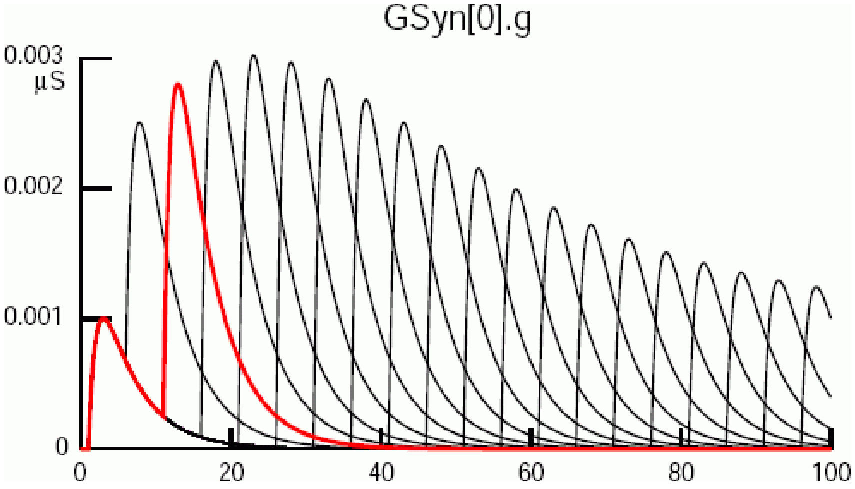

```
BREAKPOINT {
  SOLVE state METHOD cnexp
  i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }
```

Example: use-dependent synaptic plasticity



Use-dependent synaptic plasticity *continued*

```
BREAKPOINT {
  SOLVE state METHOD cnexp
  g = B - A
  i = g*(v-e)
}
DERIVATIVE state {
  A' = -A/tau1
  B' = -B/tau2
}
NET_RECEIVE(weight (uS), w, G1, G2, t0 (ms)) {
  INITIAL {w=0 G1=0 G2=0 t0=t}
  G1 = G1*exp(-(t-t0)/Gtau1)
  G2 = G2*exp(-(t-t0)/Gtau2)
  G1 = G1 + Ginc*Gfactor
  G2 = G2 + Ginc*Gfactor
  t0 = t
  w = weight*(1 + G2 - G1)
  g = g + w
  A = A + w*factor
  B = B + w*factor
}
```



Artificial spiking cells

"Integrate and fire" cells

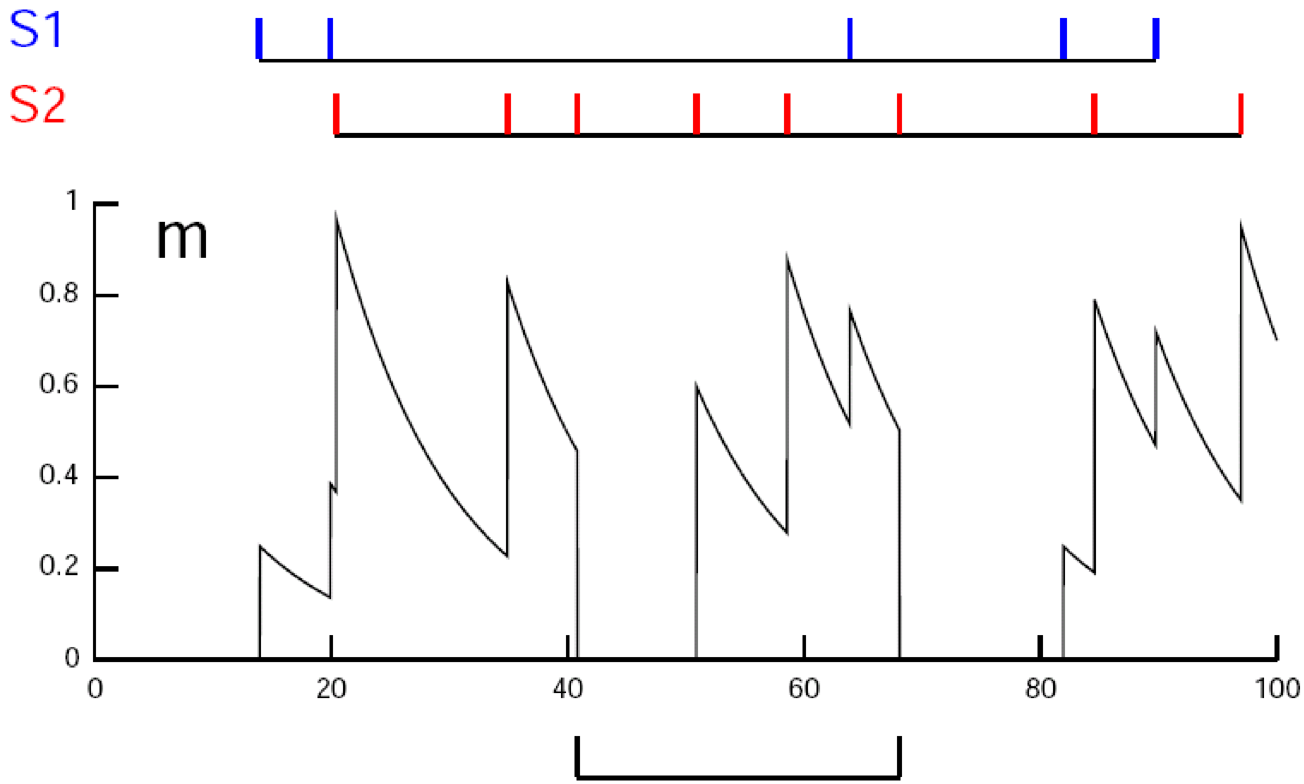
Prerequisite: all state variables must be
analytically computable from a new initial condition

Orders of magnitude faster than numerical integration

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

Hybrid networks

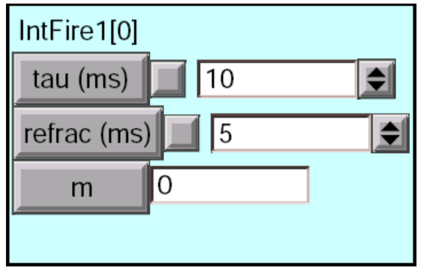
Example: leaky integrate and fire model

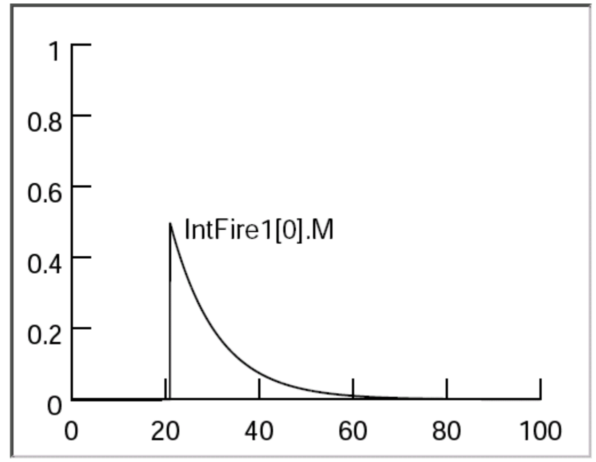


Leaky integrate and fire model *continued*

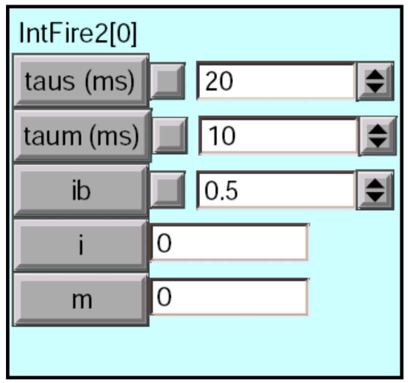
```
NEURON {
  ARTIFICIAL_CELL IntFire
  RANGE tau, m
}
  . . . declarations . . .
INITIAL { m = 0   t0 = t }
NET_RECEIVE (w) {
  m = m*exp(-(t-t0)/tau)
  t0 = t
  m = m + w
  if (m > 1) {
    net_event(t)
    m = 0
  }
}
```

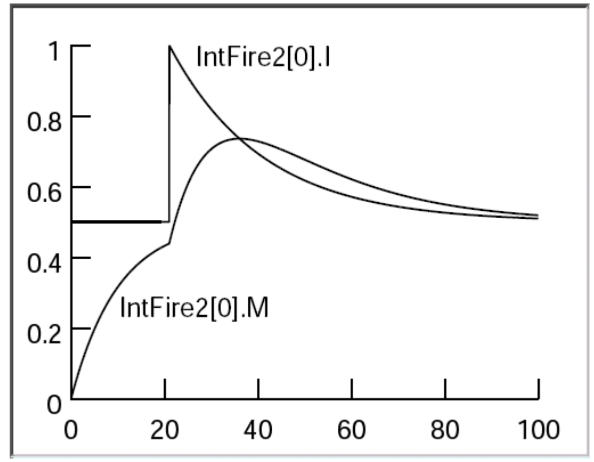
IntFire1



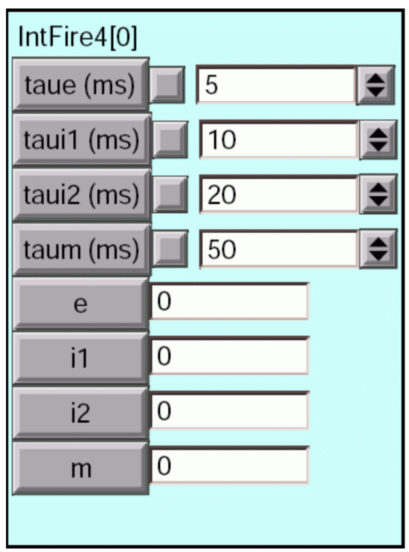


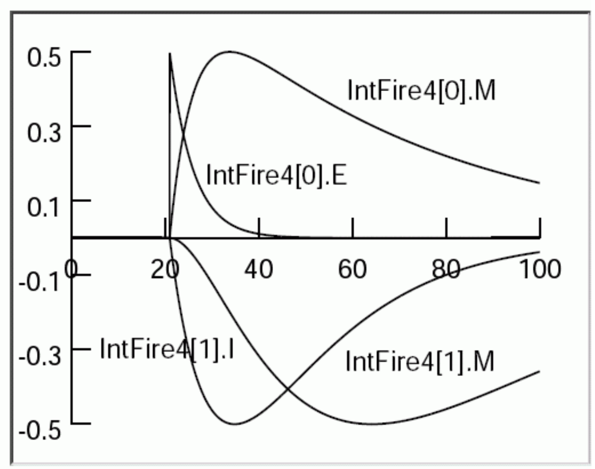
IntFire2





IntFire4





Defining the types of cells

Artificial spiking cells

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

Biophysical model cells

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

Defining types of biophysical model cells

Encapsulate in a class

Export hoc class definition from CellBuilder or Network Builder

or

write your own in Python.

```
class Cell:
  def __init__(self)
    # specify geom, topol, biophys
    soma = h.Section(name='soma')
    self.soma = soma
    ... etc. ...

cells[]
N = 1000
for i in range(N):
  cell = Cell() # h.Cell() if Cell is defined in hoc
  cells.append(cell)
```

Homework

Create a 1 section model cell called 'soma' with
- surface area 100 um2
- nseg 1
- pas channels with e -65 mV and g 5e-5 S/cm2
- (membrane time constant 20 ms)

Attach an ExpSyn with tau 3 ms, e 0 mV to soma(0.5).

Drive the ExpSyn with events from a NetStim with
- interval 10 ms
- number 1
- start 5 ms
- noise 0

Set the NetCon's delay to 1 ms.

Homework *continued*

Run a simulation for 100 ms. How big must the NetCon's weight[0] be to elicit a 1 mV EPSP at soma(0.5)? (2 significant figures)

Now uninsert pas and insert hh. What is the minimum positive weight[0] that triggers a spike?

Extra credit:

Using the model with hh, adjust weight[0] to a value that elicits a 1 mV EPSP.

Next change the NetStim's interval to 1 ms, number 1e9, and noise to 1. Run 100 simulations that include 1000 ms of synaptic input and record the number of spikes per run. Generate a histogram of number of spikes per run (binwidth = 1).