

Networks:

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

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

Communication between cells

Gap junctions

Synaptic transmission
graded
spike-triggered

Spike-triggered synaptic transmission

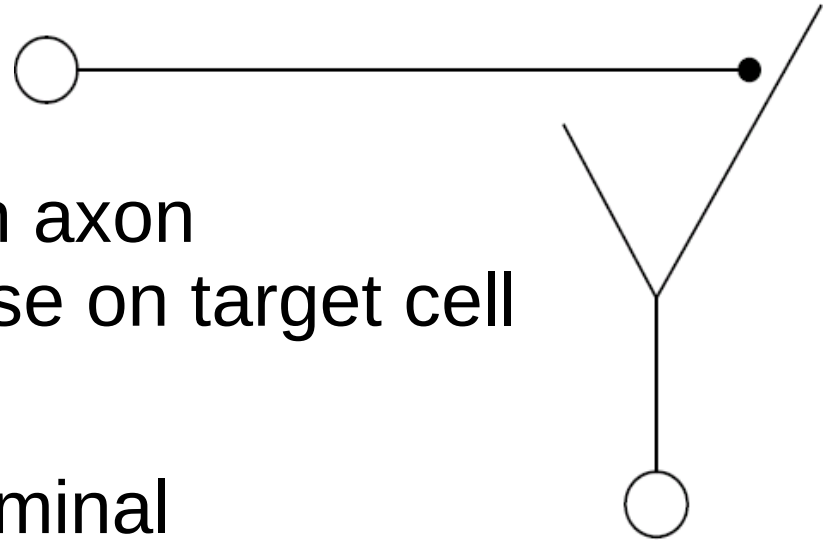
Physical system:

Presynaptic neuron with axon
that projects to synapse on target cell

Conceptual model:

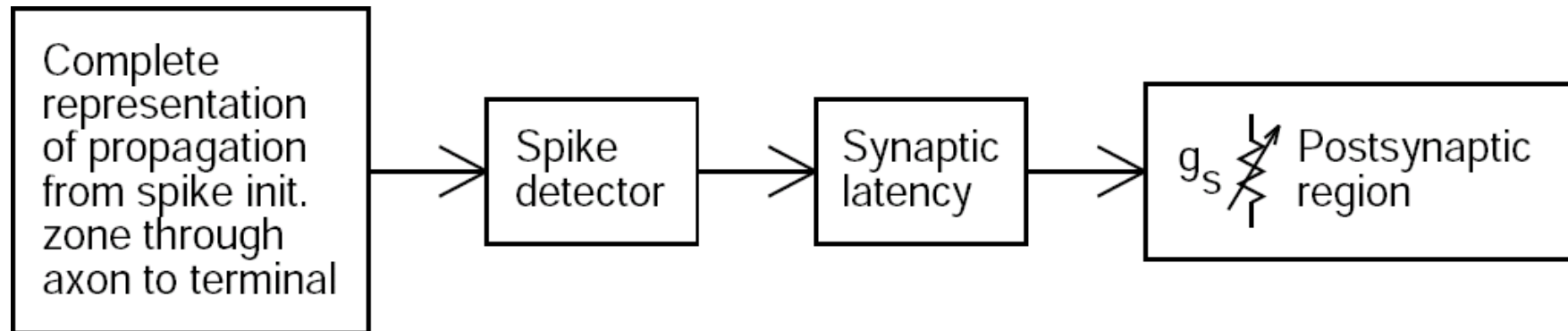
Spike in presynaptic terminal
triggers transmitter release;
presynaptic details unimportant

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

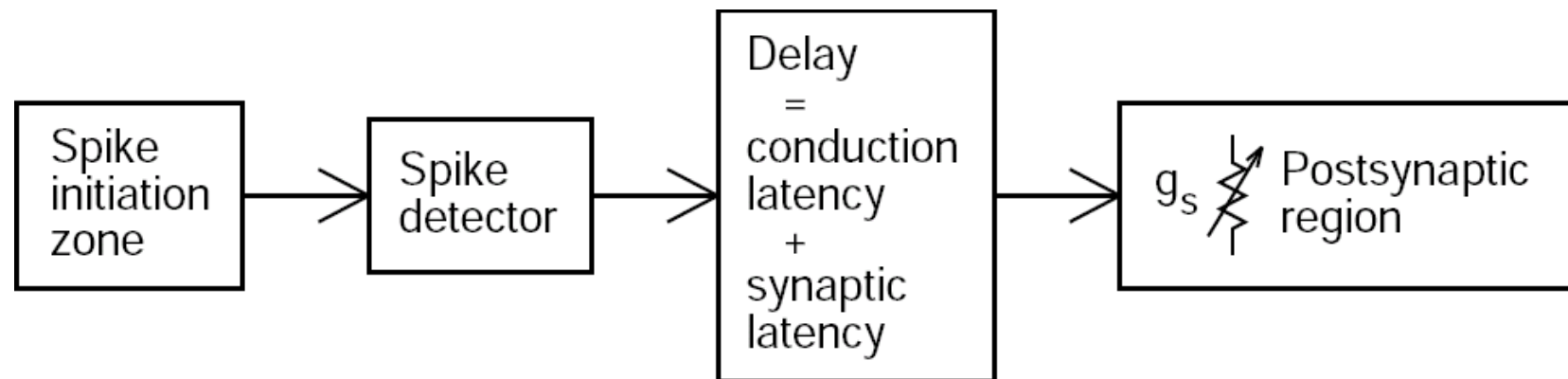


Spike-triggered transmission: computational implementation

Basic idea



More efficient: "virtual spike propagation"



The NetCon class

Python usage

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

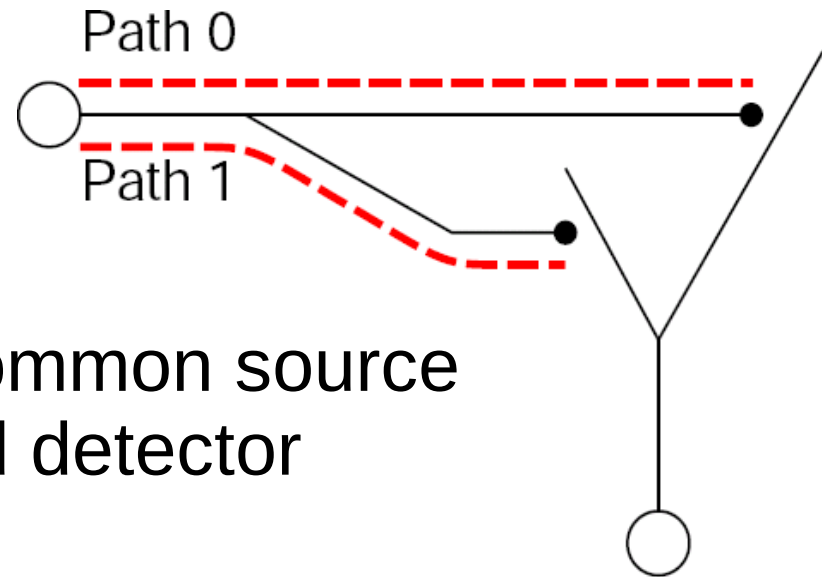
Defaults

```
nc.threshold = 10.0
nc.delay = 1.0 # must be >= 0
nc.weight[0] = 0.0 # weight is an array
```

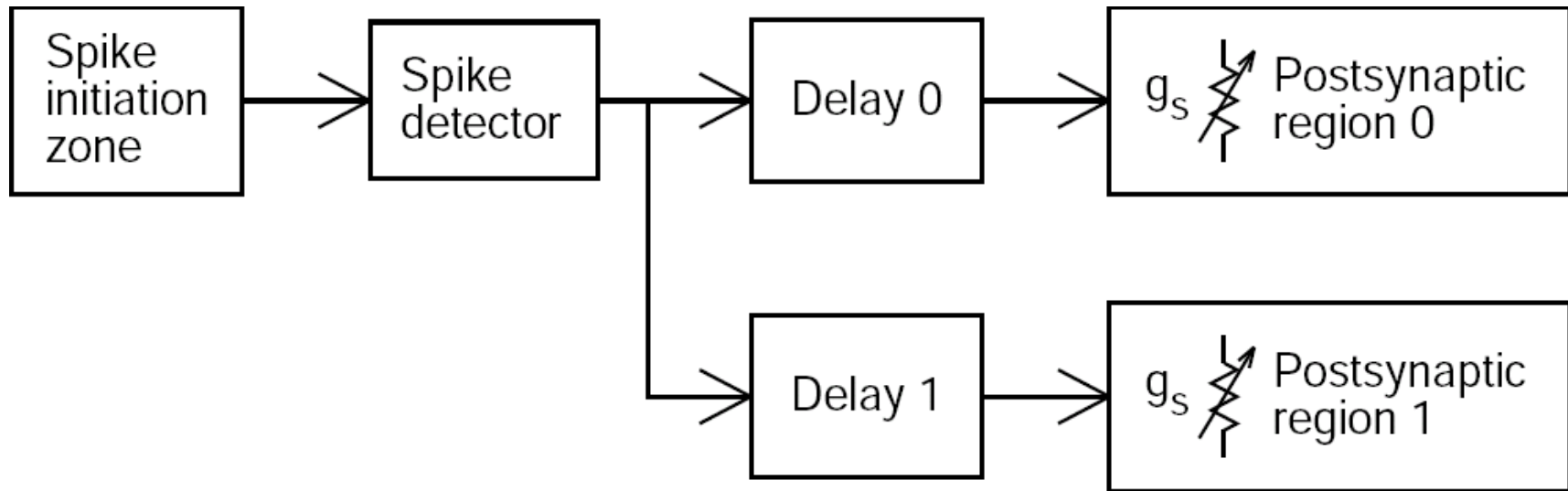
NMODL specification of synaptic mechanism

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

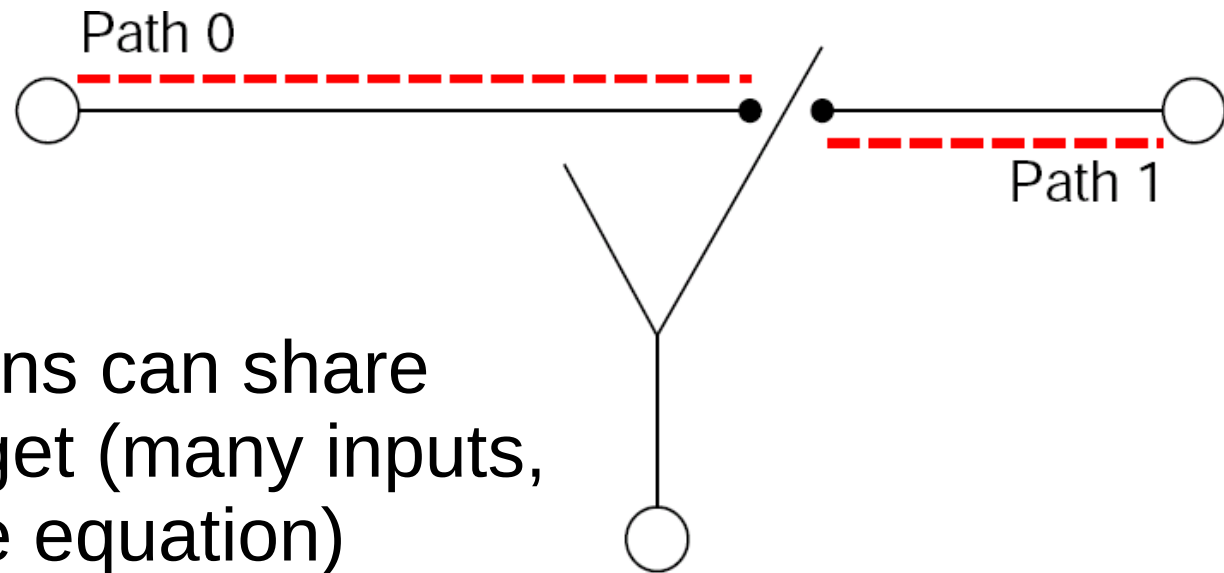
Efficient divergence



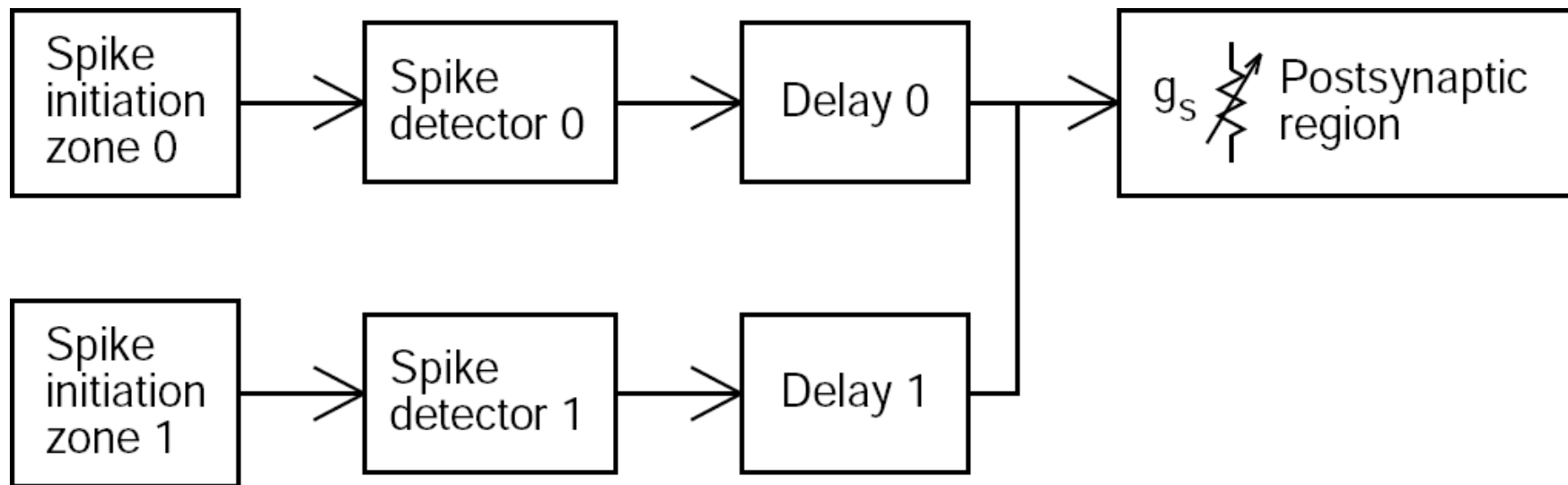
Multiple NetCons with a common source
share a single threshold detector



Efficient convergence



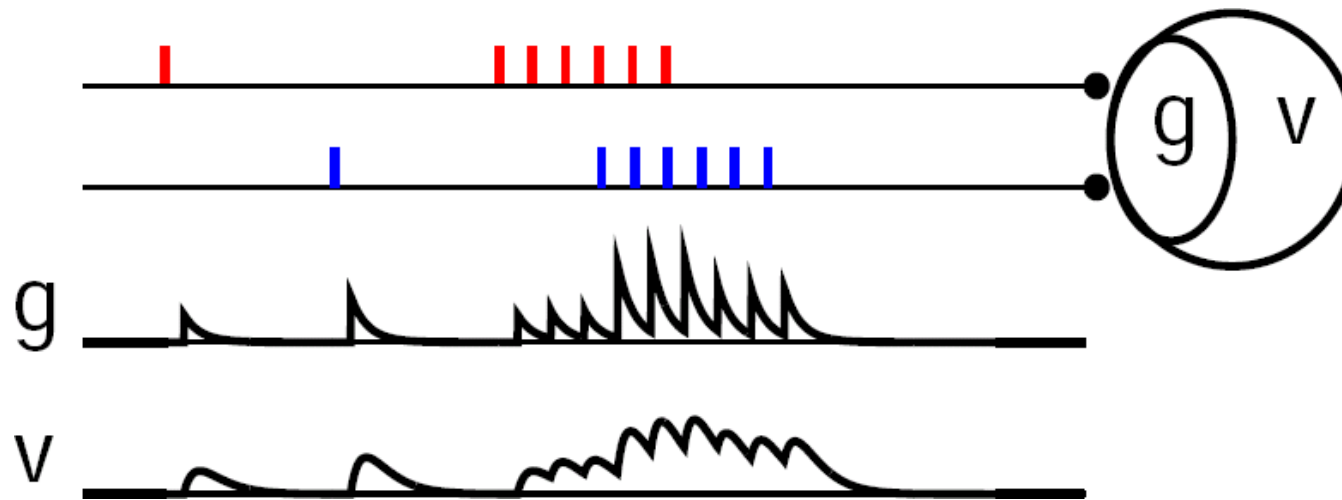
Multiple NetCons can share
a single target (many inputs,
but only one equation)



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 }
```

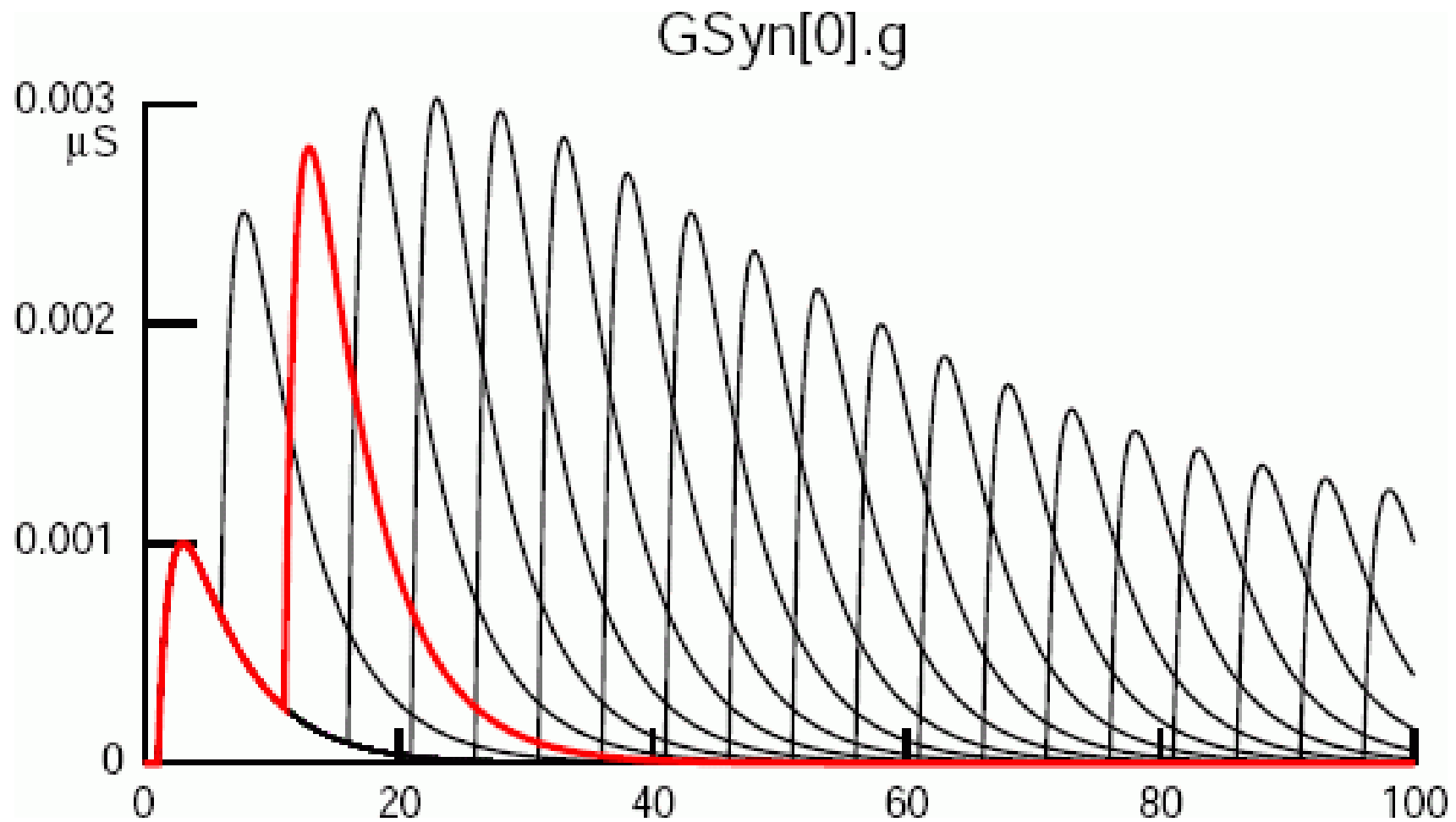

g_s with fast rise and exponential decay
continued



```

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

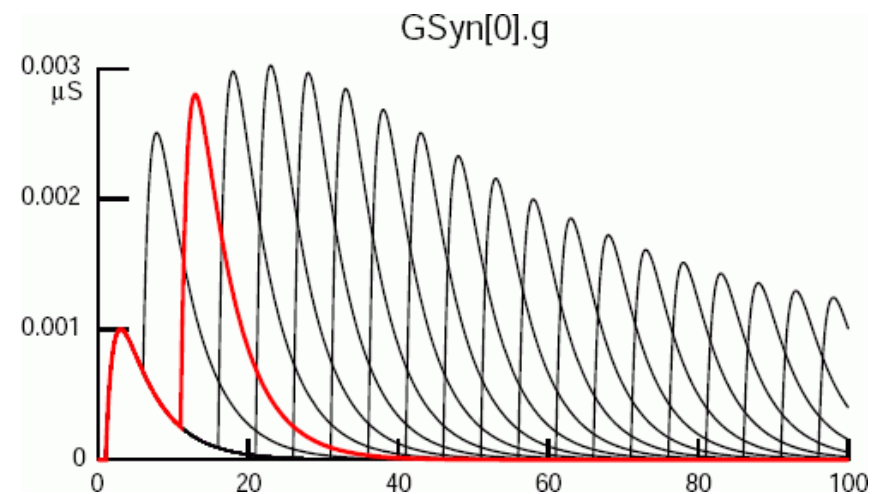


Use-dependent synaptic plasticity *continued*

```
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

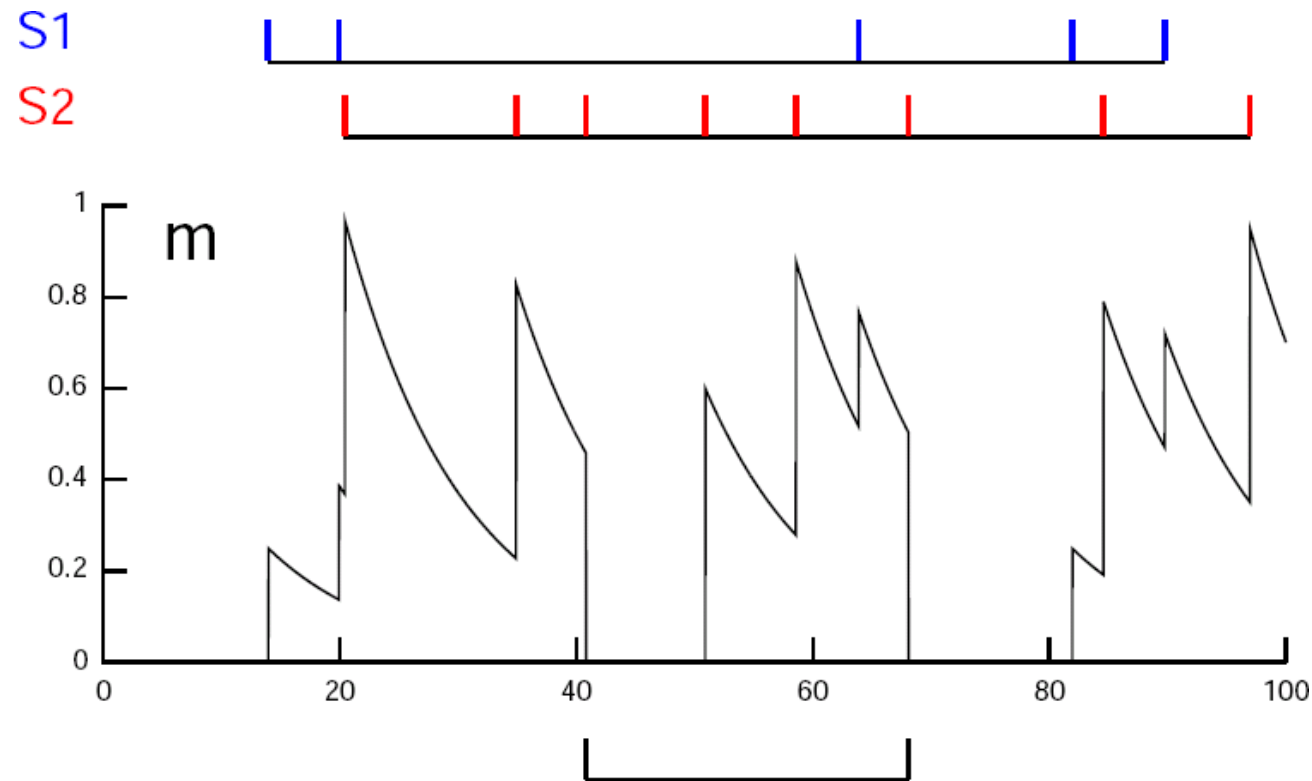
Event-driven simulation run time is

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  
    }  
}
```

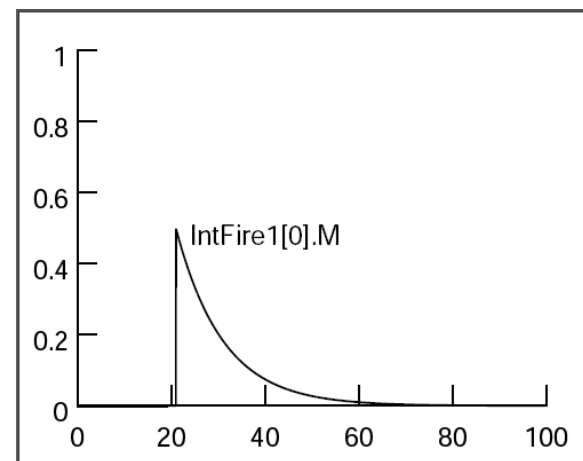
IntFire1

IntFire1[0]

tau (ms) 10

refrac (ms) 5

m 0



IntFire2

IntFire2[0]

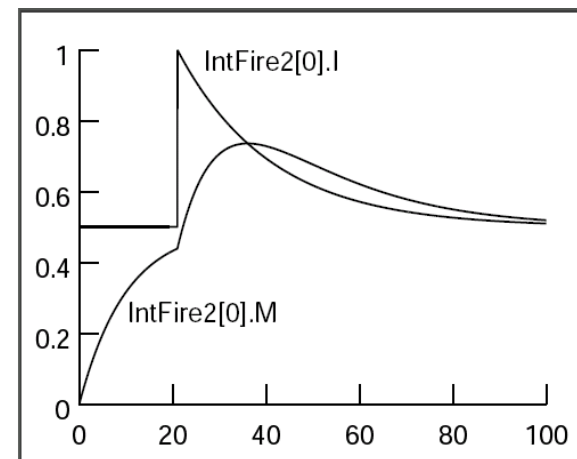
taus (ms) 20

taum (ms) 10

ib 0.5

i 0

m 0



IntFire4

IntFire4[0]

taue (ms) 5

taui1 (ms) 10

taui2 (ms) 20

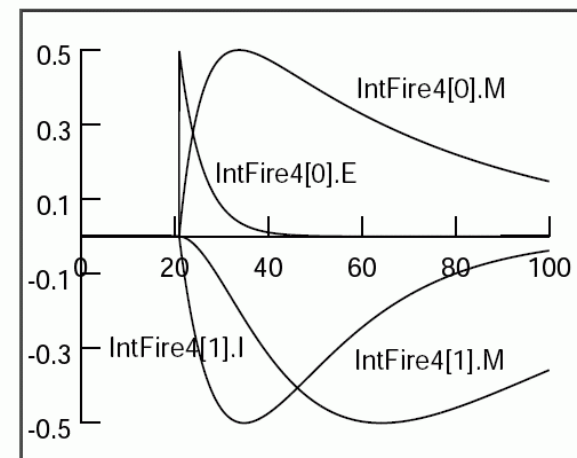
taum (ms) 50

e 0

i1 0

i2 0

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

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)
```