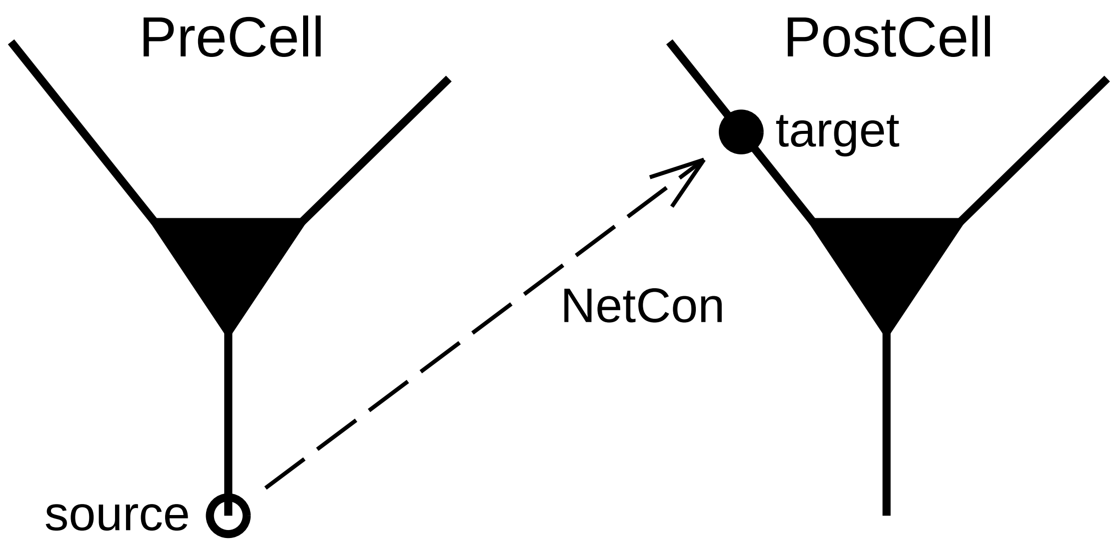
Parallelizing Serial Network Models with NEURON

Definitions

- # of processing units = # of programs that can be executed simultaneously
  - Synonyms: processing unit, **processor**, CPU, core, node, host

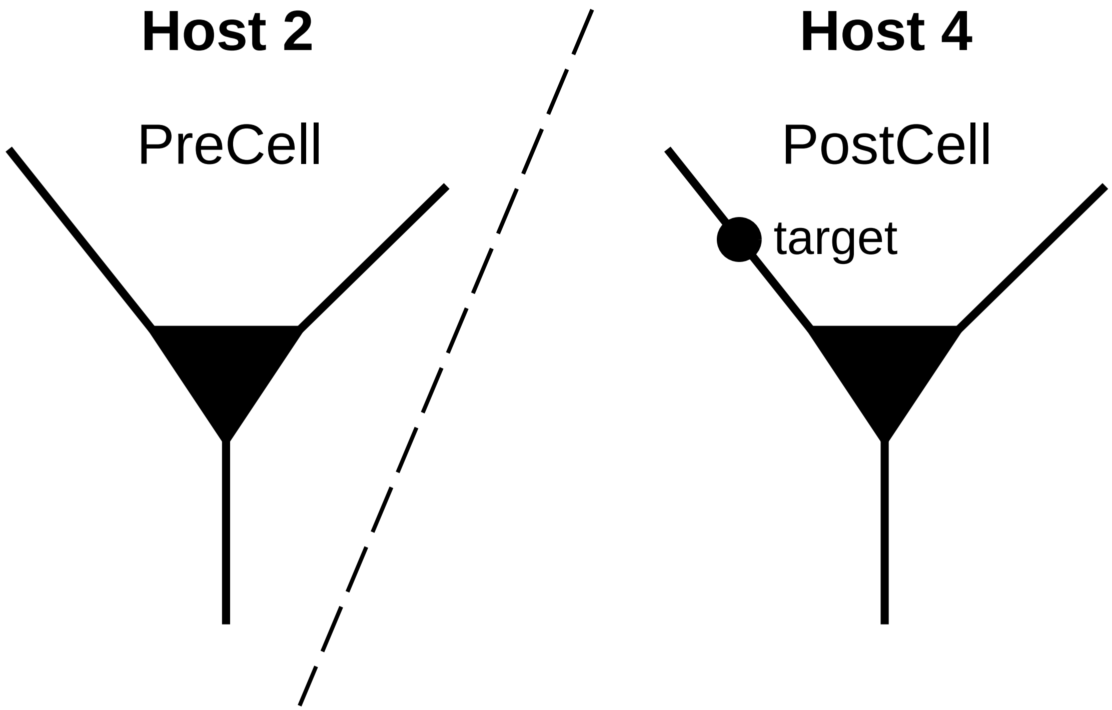
Serial implementation



All cells exist on a single processor.

```
nc = new NetCon(source,
  target)
```

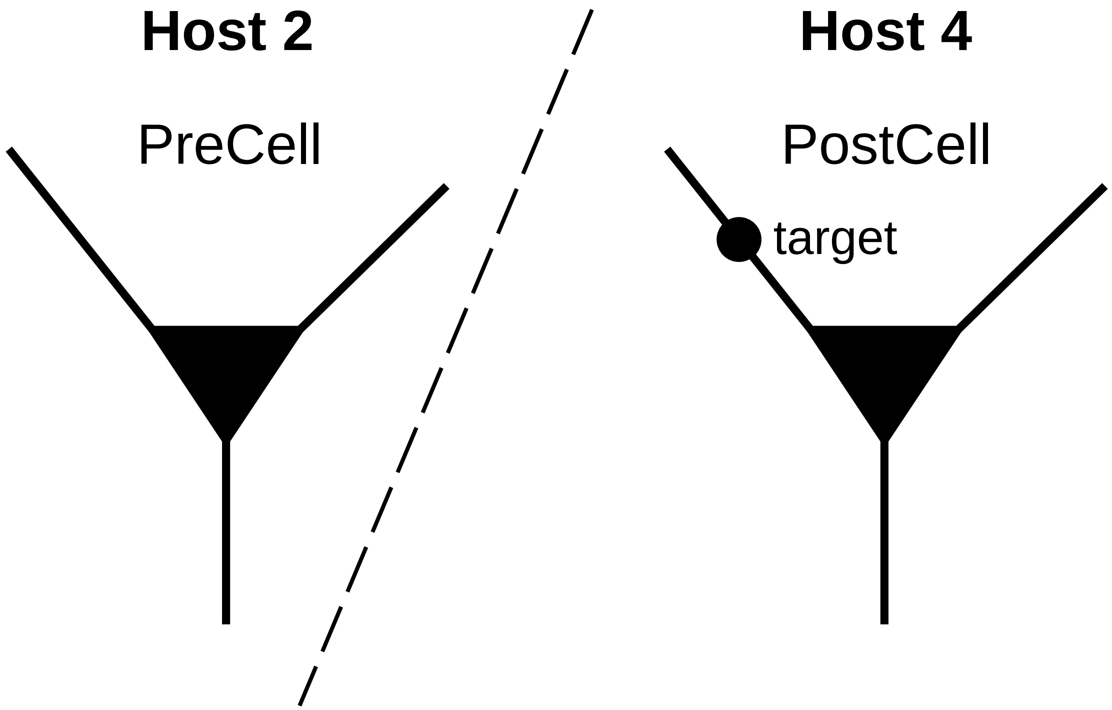
Parallel implementation



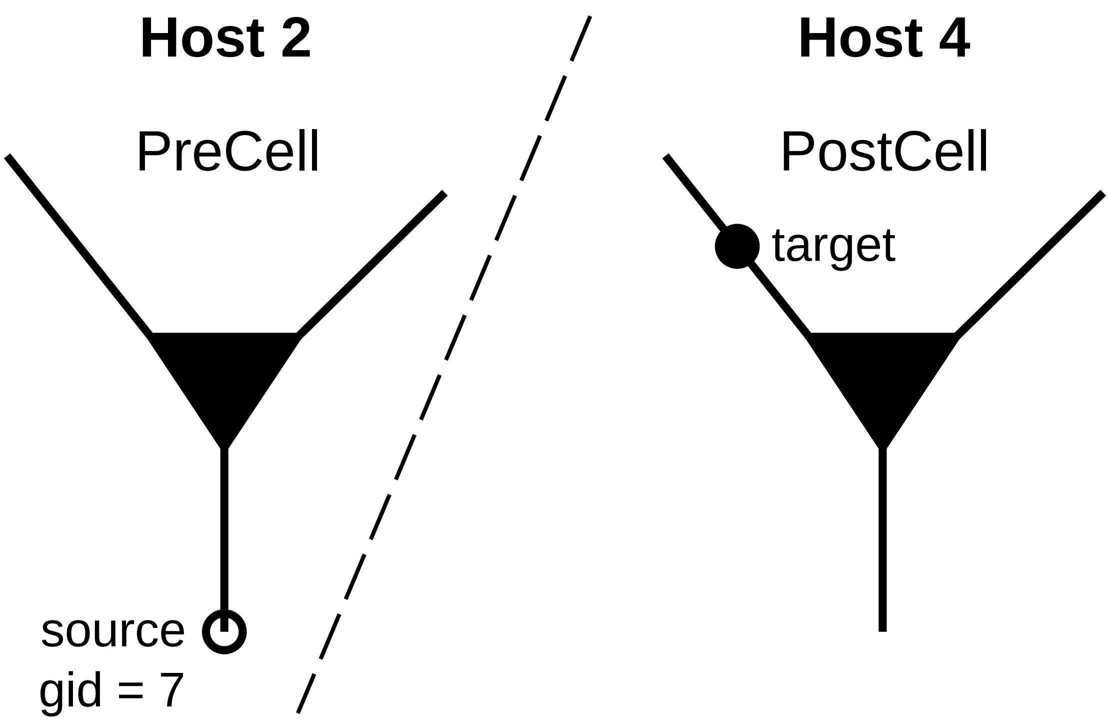
Multiple processors, each with its own cells.

Problem: spike sources and targets usually not on the same processor

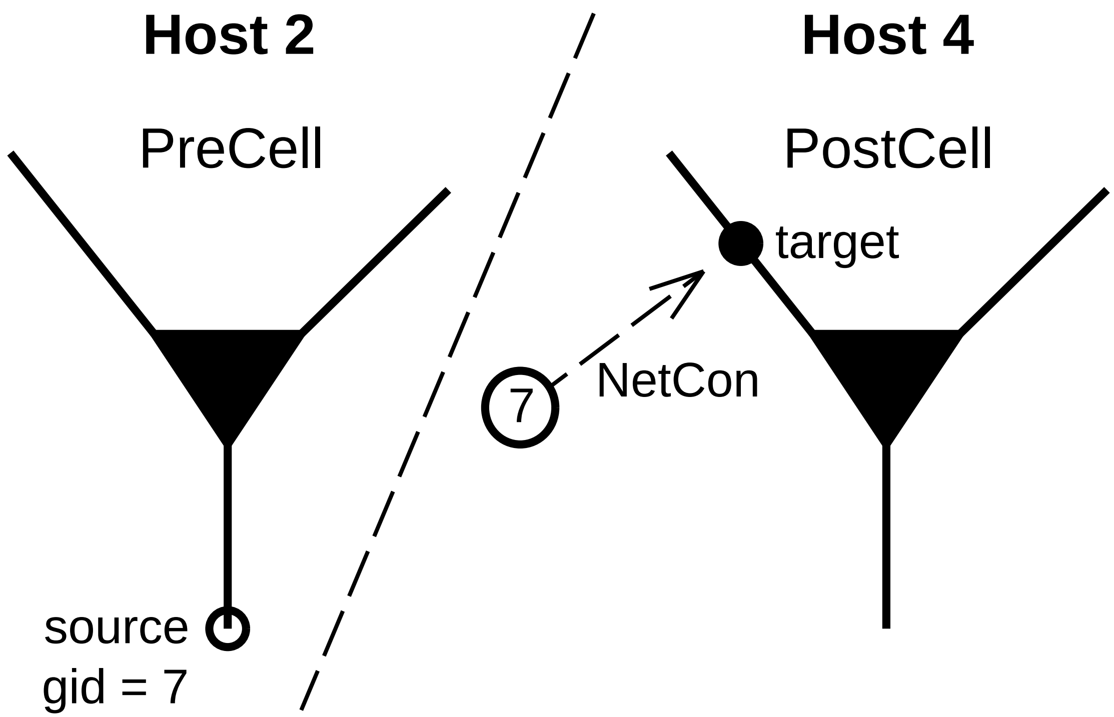
Parallel implementation



Solution: give each spike source a unique global identifier.




```
pc = new ParallelContext // all processors
 . . .
// processor with presynaptic cell
nc = new NetCon(source, nil) // attach spike detector
                             //   to spike trigger zone
pc.cell(7, nc) // associate spike detector with gid 7
```



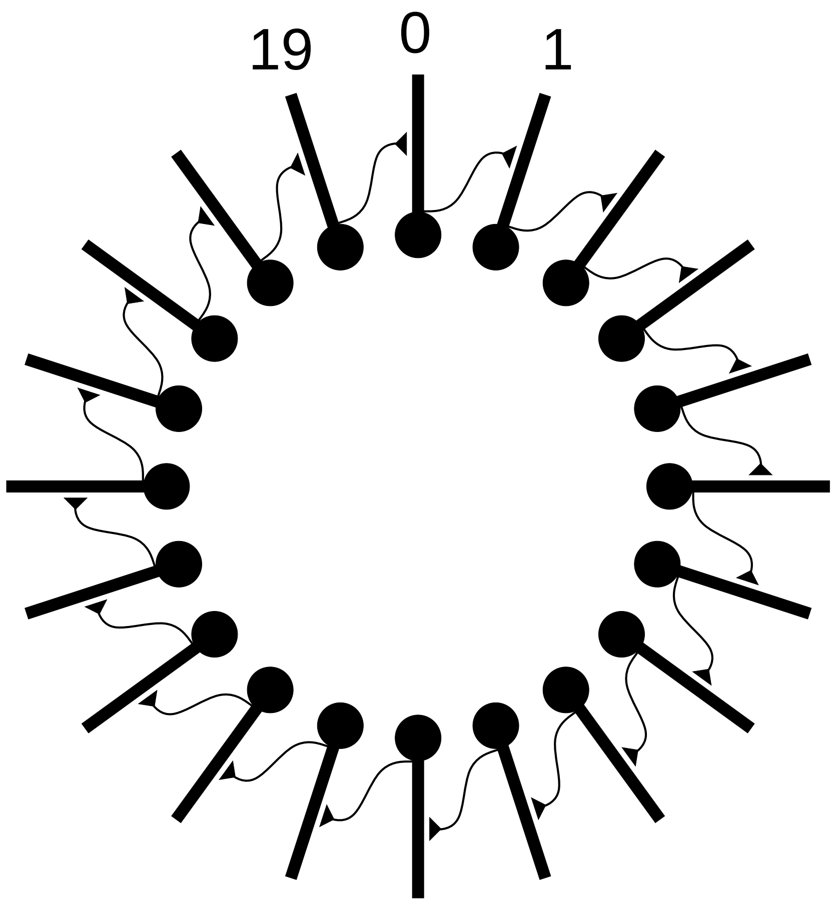

```
// processor with postsynaptic cell
pc.gid_connect(7, target)
```

On each processor:
for each presynaptic cell
    attach a spike detector to this cell
    associate spike detector with a unique gid

Target-centric strategy

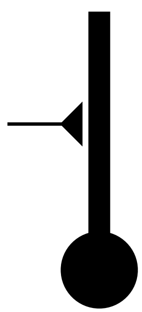
On each processor:
for each cell *c* on this proessor {
    for each spike source *s* that targets *c* {
        set up a conection from *s* to the proper
          target synapse on *c*
    }
}

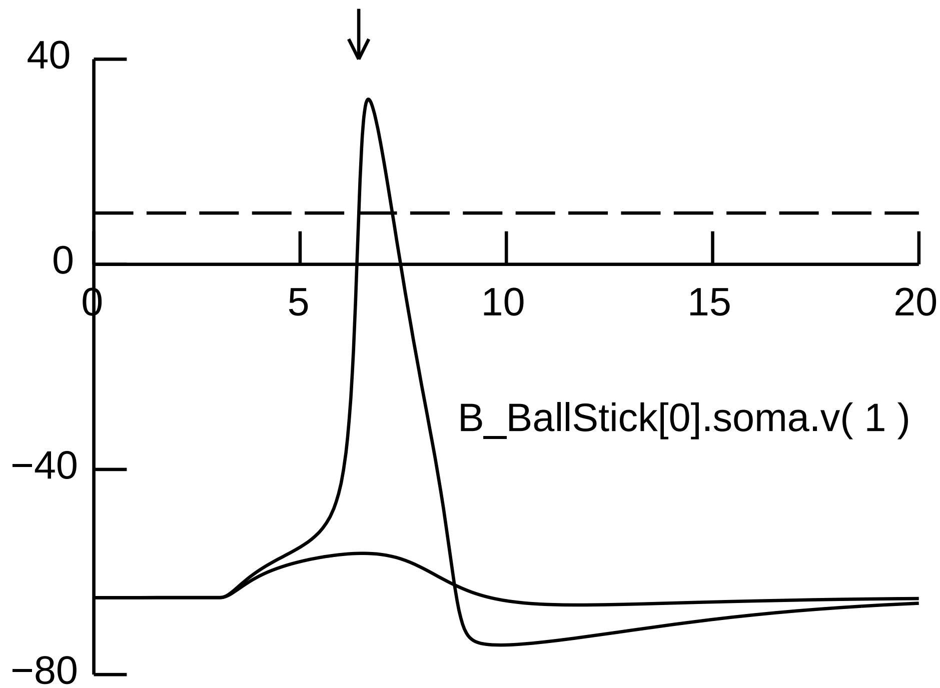
Example: ring network



Source code from Hines and Carnevale J. Neurosci. Methods 169:425-455, 2008.
http://senselab.med.yale.edu/modeldb/ShowModel.asp?model=96444

Ring network cells





Serial implementation: `ringser.hoc`

Organization of program--

- Set up network
  - Define cell classes
  - Create cells
  - Connect cells
- Specify instrumentation
- Specify simulation control
- Report results

Define cell classes

cell.hoc

```
obfunc connect2target() { localobj nc
  soma nc = new NetCon(&v(1), $o1)
  nc.threshold = 10
   . . .
```

Analyzing `ringser.hoc`

Get quick overview by skipping over

```
proc mkring() { . . .
proc mkcells() { . . .
proc connectcells() { . . .
```

to find

```
mkring(NCELL)
```

then go back to see how it all works.

Create cells

```
proc mkring() {
  mkcells($1)  // create cells
  connectcells()  // connect them together
}

// create cells and append to a List
// argument is how many to create
proc mkcells() {local i  localobj cell
  cells = new List()
  for i=0, $1-1 {
    cell = new B_BallStick()
    cells.append(cell)
  }
}
```

Connect cells

```
// connect cells, append NetCons to nclist
proc connectcells() {local i  localobj src, target, syn, nc
  nclist = new List()
  for i=0, cells.count-1 {  // iterates over sources
    src = cells.object(i)
    target = cells.object((i+1)%cells.count)
    syn = target.synlist.object(0)  // first obj in synlist
        // is an ExpSyn with e = 0 (excitatory)
    nc = src.connect2target(syn)
    nclist.append(nc)
    nc.delay = 1
    nc.weight = 0.01
  }
}
```

Specify instrumentation

```
proc mkstim() {
  stim = new NetStim()
   . . .
  ncstim = new NetCon(stim, cells.object(0).synlist.object(0))
   . . .
}

mkstim()

objref tvec, idvec  // to record spike times and cell ids
proc spikerecord() {local i  localobj nc, nil
  tvec = new Vector()
  idvec = new Vector()
  for i=0, nclist.count-1 {
    nc = cells.object(i).connect2target(nil)
    nc.record(tvec, idvec, i)
  }
}
```

Specify simulation control and report results

```
tstop = 100
run()

proc spikeout() { local i
  printf("\ntime\t cell\n")
  for i=0, tvec.size-1 {
    printf("%g\t %d\n", tvec.x[i], idvec.x[i])
  }
}
```

Output:

```
time     cell
2.05     0
5.1      1
8.15     2
 . . .
96.6     11
99.65    12
```

Parallel implementation: `ringpar.hoc`

Use `{ }` to suppress printing to stdout

```
{load_file("nrngui.hoc")}
```

Enable use of gids to refer to
source and target cells

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

Create cells (parallel)

```
proc mkcells() {local i  localobj cell, nc, nil
  cells = new List()
  // each host gets every nhost'th cell,
  // starting from the id of the host
  // and continuing until all cells have been dealt out
  for (i=pc.id; i < $1; i += pc.nhost) {
    cell = new B_BallStick()
    cells.append(cell)
    // associate gid i with this host
    pc.set_gid2node(i, pc.id)
    // attach spike detector to cell
    nc = cell.connect2target(nil)
    // associate gid i with spike detector
    pc.cell(i, nc)
  }
}
```

host id	gid
0	0, N, 2N . . .
1	1, 1+N, 1+2N . . .
. . .	
N-1	N-1, 2N-1, 3N-1 . . .

Connect cells (parallel)

```
// connect cells, append NetCons to nclist
proc connectcells() {local i, targid  localobj src, target, syn, nc
  nclist = new List()
  for i=0, NCELL - 1 {  // iterating over source gids
    targid = (i+1)%NCELL
    if (!pc.gid_exists(targid)) { continue }
    target = pc.gid2cell(targid)
    syn = target.synlist.object(0)  // first obj in synlist
        // is an ExpSyn with e = 0 (excitatory)
    nc = pc.gid_connect(i, syn)
    nclist.append(nc)
    nc.delay = 1
    nc.weight = 0.01
  }
}
```

Specify instrumentation (parallel)

```
proc mkstim() {
  // exit if first cell in the net does not exist on this host
  if (!pc.gid_exists(0)) { return }
  stim = new NetStim()
   . . .
  ncstim = new NetCon(stim, pc.gid2cell(0).synlist.object(0))
   . . .
}

mkstim()

objref tvec, idvec  // to record spike times and cell ids
proc spikerecord() {local i  localobj nc, nil
  tvec = new Vector()
  idvec = new Vector()
  for i=0, cells.count-1 {
    nc = cells.object(i).connect2target(nil)
    nc.record(tvec, idvec, nc.srcgid)
  }
}
```

Specify simulation control (parallel)

```
tstop = 100
{pc.set_maxstep(10)}
stdinit()
{pc.psolve(tstop)}
```

Report results (parallel)

```
proc spikeout() { local i, rank
  pc.barrier()  // wait for all hosts to get to this point
  if (pc.id==0) printf("\ntime\t cell\n")  // print header once
  for rank=0, pc.nhost-1 {  // host 0 first, then 1, 2, etc.
    if (rank==pc.id) {
      for i=0, tvec.size-1 {
        printf("%g\t %d\n", tvec.x[i], idvec.x[i])
      }
    }
    pc.barrier()  // wait for all hosts to get to this point
  }
}
```

Output:

```
time     cell
2.05     0
14.25    4
26.45    8
38.65    12
 . . .
72.2     3
84.4     7
96.6     11
```

Sort and compare results (parallel)

Capture outputs to files, sort by spike time and cell id, then compare.

```
$ # numeric sort on spike time, then cell id
$ sort -n -k 1 -k 2 serout.txt > sorted_serout.txt
$ sort -n -k 1 -k 2 parout.txt > sorted_parout.txt
$ cmp sorted_serout.txt sorted_parout.txt
$
```

Exercises

```
Download source code from ModelDB (accession # 96444).
Run ringser.hoc and ringpar.hoc.
Use sort to compare serial and parallel outputs.

Copy ringpar.hoc to testringpar.hoc.
In testringpar.hoc, comment out everything after spikerecord(),
then embed print statements of the form
printf("host %d created cell %s and assigned gid %d\n", \
        pc.id, cell, id)
as the last statement inside the for loop of proc mkcells(),
then use NEURON to execute your code.
Does the printed output make sense?

Do something similar in proc connectcells() to verify that spike
sources and targets have been connected correctly.

Use printf statements to verify that proc mkstim() and
proc spikerecord() work properly.
```