Python + NEURON

Python + NEURON

All legacy models must work.

Python + NEURON

All legacy models must work.

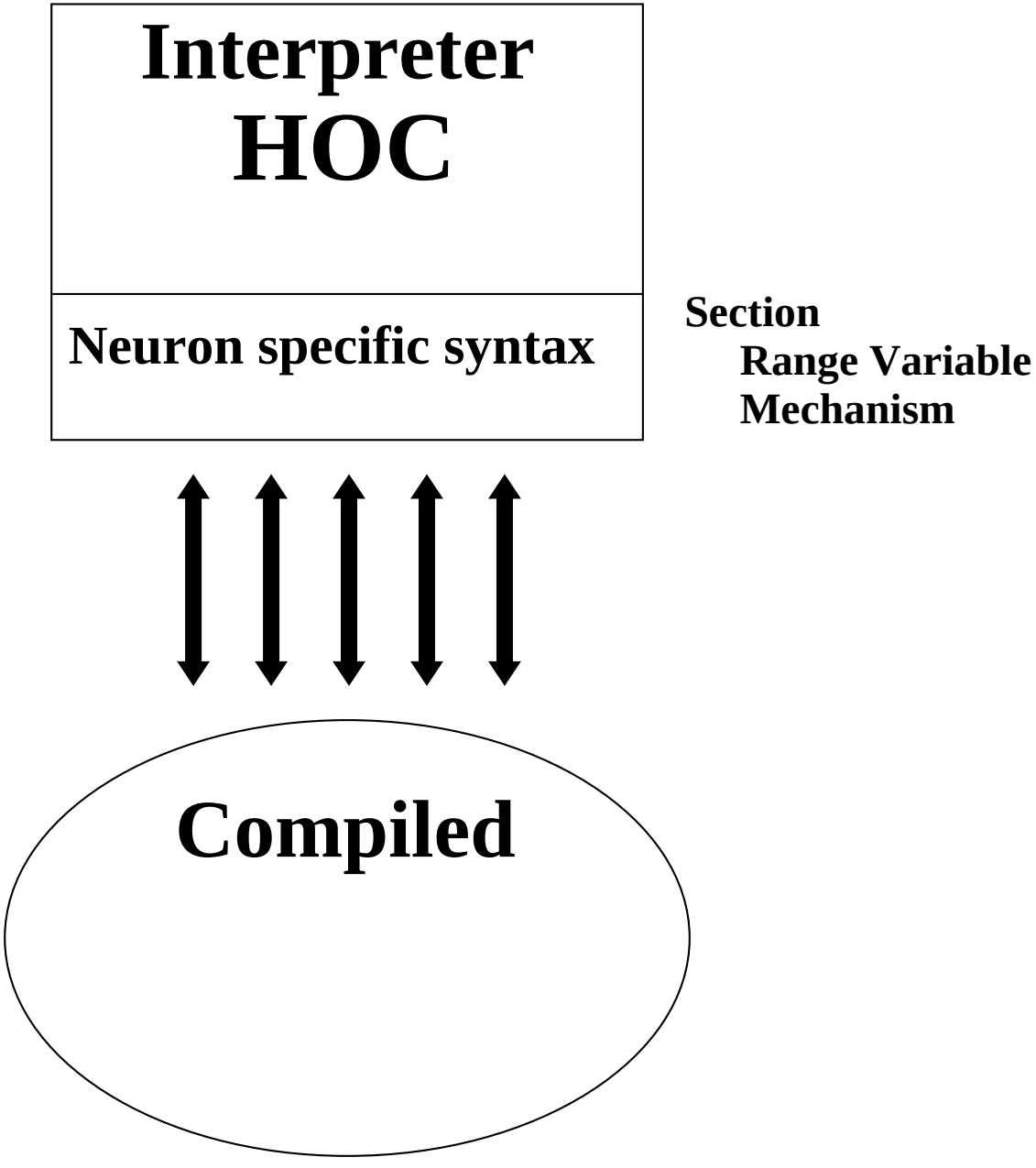
Superior representation of underlying concepts.

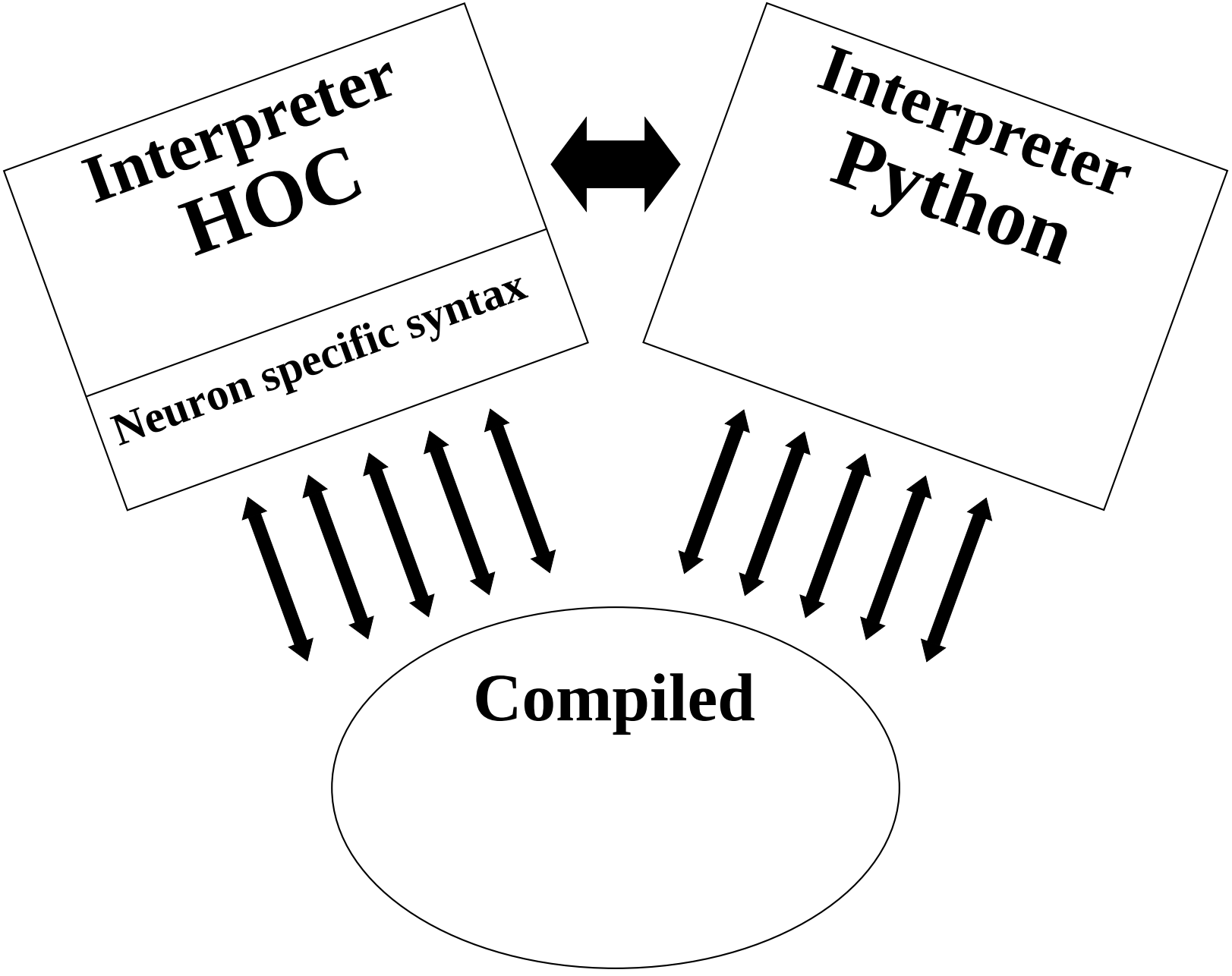
Python + NEURON

All legacy models must work.

Superior representation of underlying concepts.

No extra installation difficulty.


Interpreter
HOC
Neuron specific syntax
Section
Range Variable
Mechanism
Compiled


Interpreter
HOC
Neuron specific syntax
Interpreter
Python
Compiled

Installation

```
>>> import neuron
```

Linux: i686, x86_64

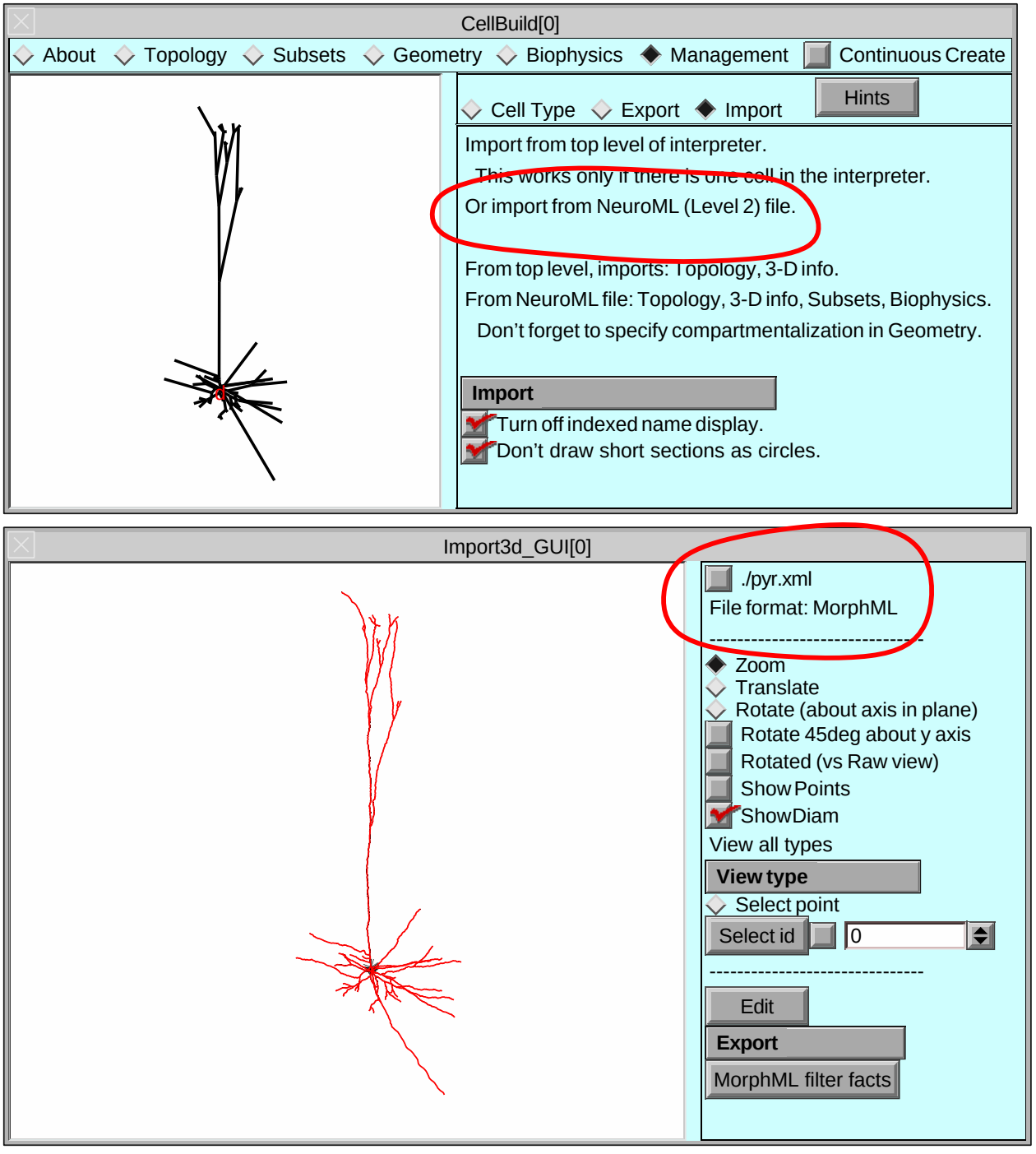
Mac: OS X 10.4, 10.5, 10.6

MSWin: Cygwin, MinGW

Launch: NEURON, Python

Python: 2.3, 2.4, 2.5, 2.6, 3.1

NumPy


CellBuild[0]
About
Topology
Subsets
Geometry
Biophysics
Management
Continuous Create
Cell Type
Export
Import
Hints
Import from top level of interpreter.
This works only if there is one cell in the interpreter.
Or import from NeuroML (Level 2) file.
From top level, imports: Topology, 3-D info.
From NeuroML file: Topology, 3-D info, Subsets, Biophysics.
Don't forget to specify compartmentalization in Geometry.
Import
Turn off indexed name display.
Don't draw short sections as circles.
Import3d_GUI[0]
./pyr.xml
File format: MorphML
Zoom
Translate
Rotate (about axis in plane)
Rotate 45deg about y axis
Rotated (vs Raw view)
Show Points
ShowDiam
View all types
View type
Select point
Select id
0
Edit
Export
MorphML filter facts

```
$ nrniv -python
NEURON -- VERSION 7.2 ...
```

```
$ nrniv -python
NEURON -- VERSION 7.2 ...

>>> from neuron import h
>>> print h
<hoc.HocObject object at 0x2b4f1b81e030>
>>> print h.hname()
TopLevelHocInterpreter
```

```
>>> h(’’’
... x = 5
... strdef s
... s = "hello"
... func square() { return $1*$1 }
... ’’’)
1
```

```
>>> h('''
... x = 5
... strdef s
... s = "hello"
... func square() { return $1*$1 }
... ''')
1
>>> print h.x, h.s, h.square(4)
5.0 hello 16.0
```

```
>>> v = h.Vector(4).indgen().add(10)
>>> print v, len(v), v.size(), v.x[2], v[2]
Vector[1] 4 4.0 12.0 12.0
```

```
>>> v = h.Vector(4).indgen().add(10)
>>> print v, len(v), v.size(), v.x[2], v[2]
Vector[1] 4 4.0 12.0 12.0
>>> v.printf()
10      11      12      13
4.0
>>> for x in v: print x
...
10.0
11.0
12.0
13.0
>>>
```

```
>>> import numpy
>>> na = numpy.arange(0, 10, 0.00001) # 0.0131
>>> v = h.Vector(na)                  # 0.0197
>>> v.size()
1000000.0
>>> nb = numpy.array(v)               # 0.0125
>>> nb[999999]
9.9999900000000004
>>> b = list(v)                       # 0.0717
>>> for i in xrange(0, len(nb)):
...   v.x[i] = na[i]
...                                   # 3.7497
```

```
>>> def callback(a = 1, b = 2):
...   print "callback: a=%d b=%d" % (a, b)
...
>>> fih = h.FInitializeHandler(callback)
>>> h.finitialize()
callback: a=1 b=2
1.0
```

```
>>> def callback(a = 1, b = 2):
...   print "callback: a=%d b=%d" % (a, b)
...
>>> fih = h.FInitializeHandler(callback)
>>> h.finitialize()
callback: a=1 b=2
1.0
>>> fih = h.FInitializeHandler((callback,\
... (4, 5)))
>>> h.finitialize()
callback: a=4 b=5
1.0
>>>
```

```
# assume hh soma model

vvec = h.Vector()
vvec.record(soma(.5)._ref_v, sec=soma)
```

```
# assume hh soma model

vvec = h.Vector()
vvec.record(soma(.5)._ref_v, sec=soma)

tvec = h.Vector()
tvec.record(h._ref_t, sec=soma)

h.run()
```

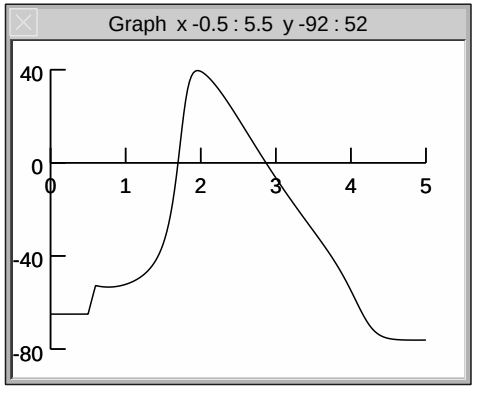
```
# assume hh soma model

vvec = h.Vector()
vvec.record(soma(.5)._ref_v, sec=soma)

tvec = h.Vector()
tvec.record(h._ref_t, sec=soma)

h.run()

g = h.Graph()
g.size(0, 5, −80, 40)
vvec.line(g, tvec)
```



```
>>> from neuron import h
>>> soma = h.Section(name = 'soma')
>>> axon = h.Section()
>>> axon.connect(soma(1))
>>> axon.nseg = 5
>>> h.topology()

|-|       soma(0-1)
   `----|       PySec_2b371cd17190(0-1)

1.0
```

```
>>> axon.L = 1000
>>> axon.diam = 1

>>> for sec in h.allsec():
...   sec.cm = 1
...   sec.Ra = 100
...   sec.insert('hh')
...
```

```
>>> axon.gnabar_hh = .1
>>> axon(.5).hh.gnabar = .09
>>> for seg in axon:
...   print seg.x, seg.hh.gnabar
...
0.1 0.1
0.3 0.1
0.5 0.09
0.7 0.1
0.9 0.1
```

```
>>> stim = h.IClamp(soma(.5))
>>> stim.delay = .5
>>> stim.dur = .1
>>> stim.amp = .4
```

```
class Cell(object):
  def __init__(self):
    self.topology()
    self.subsets()
    ...
```

```
class Cell(object):
  def __init__(self):
    self.topology()
    self.subsets()
    ...
  def topology(self):
    self.soma = h.Section(cell = self)
    self.dend = h.Section(cell = self)
    self.dend.connect(self.soma)
    ...
```

```
class Cell(object):
  def __init__(self):
    self.topology()
    self.subsets()
    ...
  def topology(self):
    self.soma = h.Section(cell = self)
    self.dend = h.Section(cell = self)
    self.dend.connect(self.soma)
    ...
  def subsets(self):
    self.all = h.SectionList()
    self.all.wholetree(sec=self.soma)
```