



SimPy: System Simulation in Python

Dr. Klaus G. Müller

kgmuller@xs4all.nl



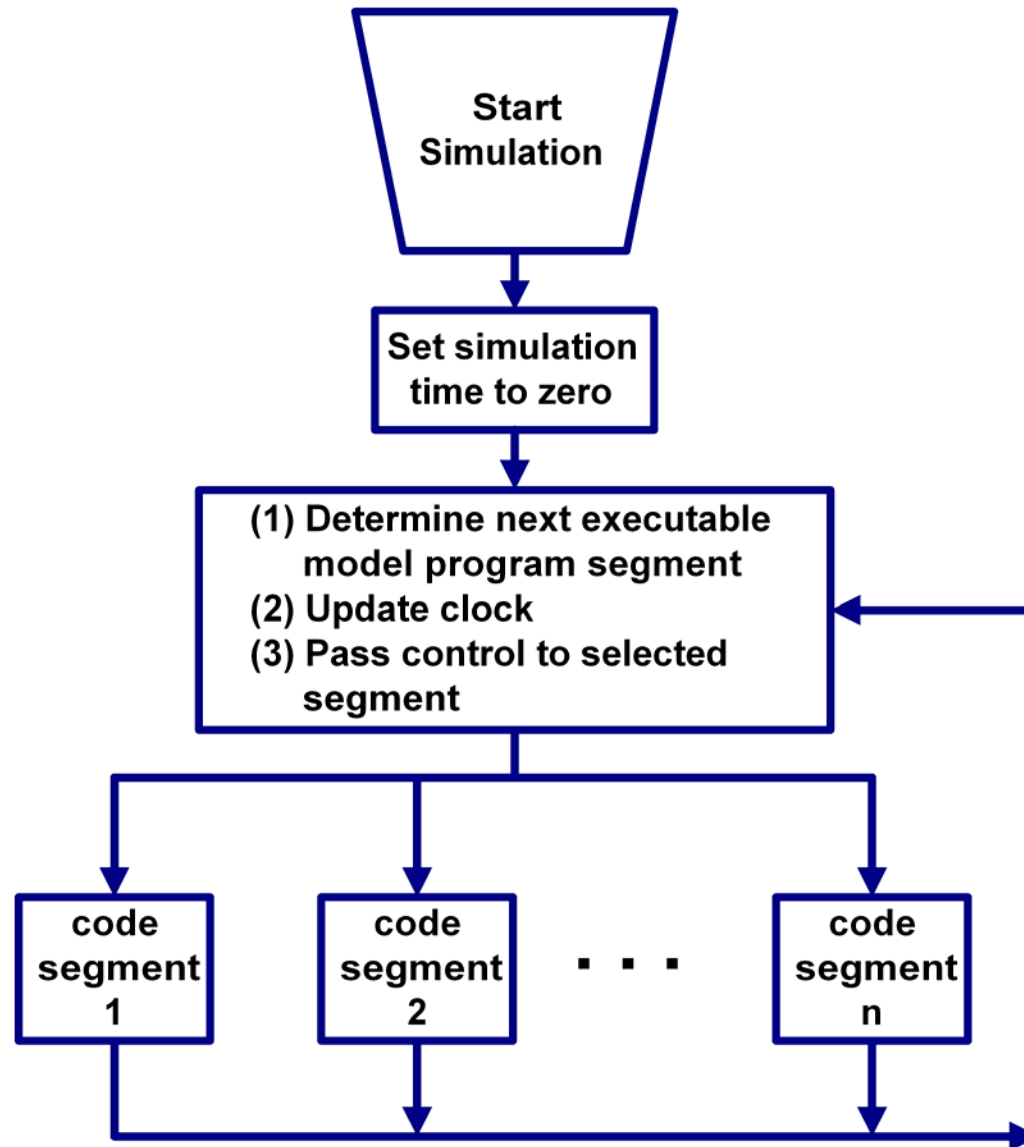
What Is SimPy?

- A Discrete Event Simulation (DES) package
 - Process based
 - Object oriented
 - Modeled after Simula
- Written in Python
- Open Source (GPL)

Process-Based Discrete Event Simulation in a Nutshell

- Models systems by
 - parallel *processes* == instances of execution of logically related activities
 - process *interactions*
- Discrete event simulation:
 - State variables change instantaneously only at specified time points (*events*)

Event Sequence Mechanism in DSE



Python Generators and Semi-Coroutines

- Python 2.2+'s generators (unlike functions/subroutines)
 - are resumable
 - can yield values across multiple invocations.
- Python generators are "semi-coroutines"
 - A generator is resumable and can branch control elsewhere
 - ... but it can only branch control back to its immediate caller

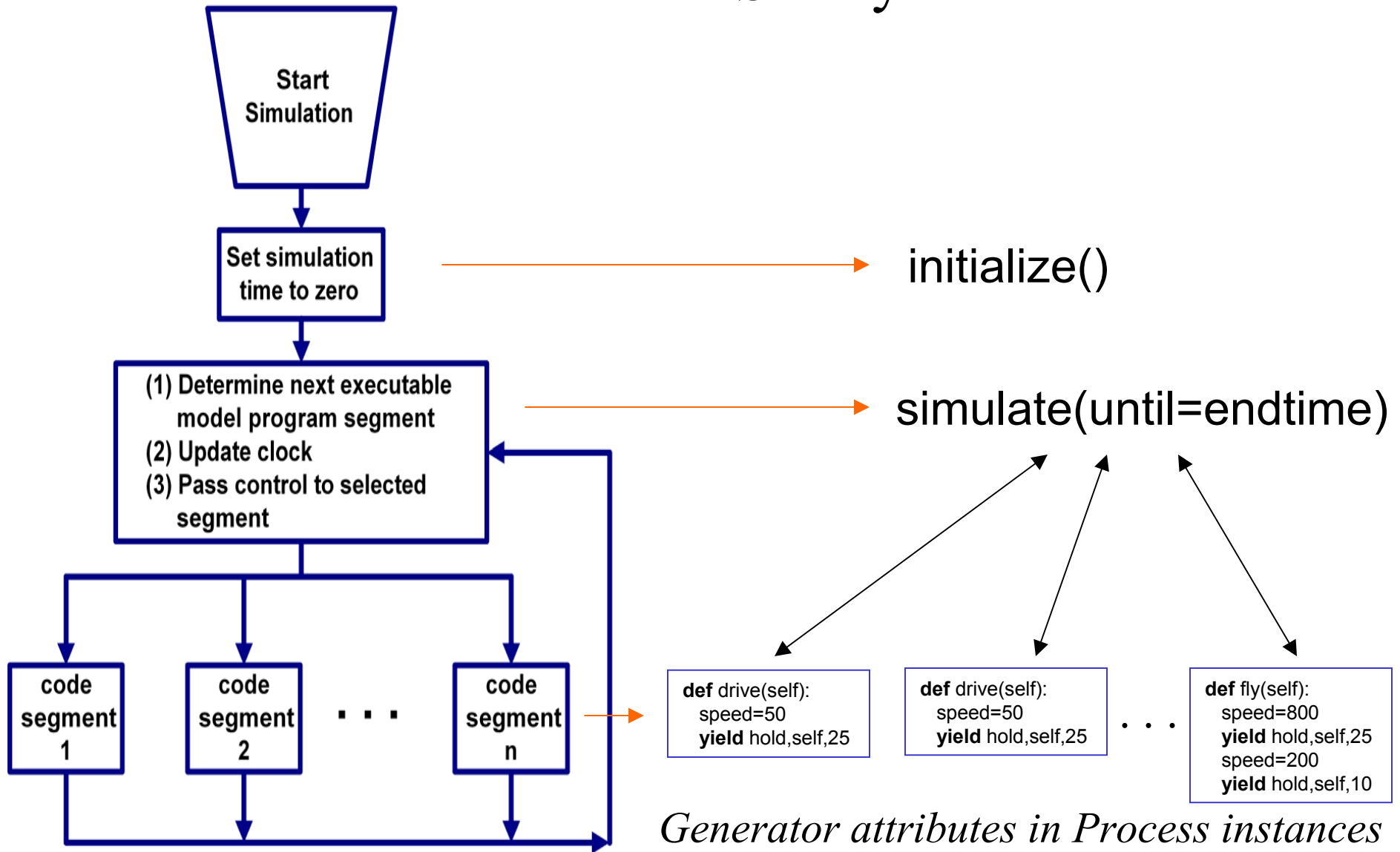
Example:

```
>>> from __future__ import generators
>>> def semi_coroutine(par):
...     for i in range(par):
...         yield i
>>> scr=semi_coroutine(4)
>>> scr.next()
0
>>> scr.next()
1
>>> scr.next()
2
>>> scr.next()
3
>>> scr.next()
```

```
Traceback (most recent call last): File
  "<interactive input>", line 1, in ?
StopIteration
```



Semi-Coroutines Simulate Parallel Processes in SimPy



Generator attributes in Process instances

Core SimPy API

- Object classes:

class Process

- Models an entity with activities

class Resource

- Models a group of shared entities used by processes

class Monitor

- Collects simulation data for analysis, statistics

- Scheduling statements:

yield hold,self,<holdtime>

= reactivate *self* after <holdtime>

yield passivate,self

= make *self* passive

yield request,self,<resource>

= request use of <resource>

(queue if <resource> not free)

yield release,self,<resource>

= give up use of <resource>



Core SimPy API (continued)

- Process interaction statements:

`activate(<pinstance>, <pinstance>.<generator>, <timing clause>)`

= make a Process instance active

`cancel(<pinstance>)`

= make a Process instance passive

`interrupt(<pinstance>)`

= interrupt <pinstance>'s current activity
(reactivate it before scheduled event)

- Simulation control statements:

`initialize()`

= set simulated time to zero

`simulate(until=<endtime>)`

= execute scheduled activities till <endtime>

`stopSimulation()`

= force immediate end of simulation



A Simple SimPy Simulation: No Process Interaction

Random driving:

```
from __future__ import generators # Python 2.2 only
from SimPy.Simulation import *
from random import uniform

class Car(Process):
    def __init__(self,id):
        Process.__init__(self)
        self.id=id

    def carDriving(self,driveTime):
        # the Process Execution Method (generator)
        print "%5.1f %s started" %(now(),self.id)
        yield hold,self,driveTime
        print "%5.1f %s arrived" %(now(),self.id)

initialize()
nrCars=10
for i in range(nrCars):
    c=Car(id="Car %s" %i)
    activate(c,c.carDriving(driveTime=uniform(1,90)))
simulate(until=100)
```

Output:

```
0.0 Car 0 started
0.0 Car 1 started
0.0 Car 2 started
0.0 Car 3 started
0.0 Car 4 started
0.0 Car 5 started
0.0 Car 6 started
0.0 Car 7 started
0.0 Car 8 started
0.0 Car 9 started
 5.9 Car 7 arrived
13.1 Car 2 arrived
27.7 Car 1 arrived
29.9 Car 5 arrived
30.6 Car 0 arrived
62.6 Car 4 arrived
73.7 Car 9 arrived
82.7 Car 3 arrived
88.7 Car 6 arrived
89.9 Car 8 arrived
```



Mutual Exclusion Synchronization

Can't park two cars in one space:

```
from __future__ import generators          # Python 2.2 only
from SimPy.Simulation import *
from random import uniform

class Car(Process):
    def __init__(self,id):
        Process.__init__(self)
        self.id=id

    def carParking(self,driveTime,parkTime):  # the PEM
        yield hold,self,driveTime
        yield request,self,parkingLot        # begin Mutex section
        print "%5.1f %s gets space" %(now(),self.id)
        yield hold,self,parkTime
        print "%5.1f %s releases space" %(now(),self.id)
        yield release,self,parkingLot        # end Mutex section

nrCars=10
initialize()
parkingLot=Resource(capacity=4,qType=FIFO)
for i in range(nrCars):
    c=Car(id="Car %s" %i)
    activate(c,c.carParking(driveTime=10,parkTime=uniform(5,20)))
simulate(until=100)
```

Output:

```
10.0 Car 0 gets space
10.0 Car 1 gets space
10.0 Car 2 gets space
10.0 Car 3 gets space
16.7 Car 1 releases space
16.7 Car 4 gets space
19.2 Car 0 releases space
19.2 Car 5 gets space
19.3 Car 3 releases space
19.3 Car 6 gets space
19.8 Car 2 releases space
19.8 Car 7 gets space
25.7 Car 4 releases space
25.7 Car 8 gets space
26.3 Car 6 releases space
26.3 Car 9 gets space
27.1 Car 7 releases space
32.0 Car 5 releases space
42.2 Car 9 releases space
44.0 Car 8 releases space
```



Producer/Consumer Synchronization

Can't pack more parts than have been made: Output:

```
...
class Widget: pass

class WidgetMaker(Process):          # The Producer
+ def __init__(self):
  def makeWidgets(self,inBuffer):
    while True:
      yield hold,self,uniform(2,3)
      print "%5.1d widget made" %now()
      inBuffer.append(Widget())
      if len(inBuffer) == 1: reactivate(p)

class Packer(Process):              # The Consumer
+ def __init__(self):
  def packWidgets(self,inBuffer):
    while True:
      if len(inBuffer) == 0:
        yield passivate,self
      else:
        inBuffer.pop(0)
        yield hold,self,uniform(1,6)
        print "%5.1d widget packed" %now()

inBuffer=[]
initialize()
w=WidgetMaker(); activate(w,w.makeWidgets(inBuffer))
p=Packer();      activate(p,p.packWidgets(inBuffer))
simulate(until=100)
```

```
2 widget made
5 widget made
5 widget packed
8 widget made
8 widget packed
10 widget made
10 widget packed
12 widget made
12 widget packed
15 widget made
15 widget packed
17 widget made
17 widget packed
19 widget made
...
92 widget packed
95 widget made
95 widget packed
97 widget made
97 widget packed
99 widget packed
```



Process Interrupts

Machines break down and get repaired:

```
.....
class Widget: pass
class WidgetMaker(Process):
    def __init__(self):
        Process.__init__(self)
        pr=Problem()
        activate(pr,pr.breakMaker())
    def makeWidgets(self,inBuffer):
        while True:
            yield hold,self,uniform(2,3)
            if self.interrupted():                # a breakdown
                print "%5.1d machine breakdown/lost widget" %now()
                yield hold,self,5 #repairtime
            else:
                print "%5.1d widget made" %now()
                inBuffer.append(Widget())
                if len(inBuffer) == 1: reactivate(p)

class Problem(Process):
    def __init__(self):
        Process.__init__(self)
    def breakMaker(self):
        while True:
            yield hold,self,uniform(10,20)
            self.interrupt(w)                    # cause breakdown

+ class Packer(Process):
.....
```

Output:

```
2 widget made
3 widget packed
5 widget made
7 widget packed
7 widget made
10 widget made
10 widget packed
11 machine breakdown/lost widget
12 widget packed
18 widget made
20 widget packed
.....
78 widget packed
81 machine breakdown/lost widget
83 widget packed
88 widget made
90 widget made
90 widget packed
92 widget packed
92 machine breakdown/lost widget
99 widget made
```

Simulation Statistics

class Monitor *does simulation statistics:*

Output:

```

...
from SimPy.Monitor import * ←
class Car(Process):
+ def init (self,id):
  def carDriving(self,driveTime,parkTime):
    yield hold,self,driveTime
    yield request,self,parking
    print "%5.1f %s gets parking space" %(now(),self.id)
    yield hold,self,parkTime
    wt.tally(parkTime) # accumulates observations
    print "%5.1f %s releases parking space" %(now(),self.id)
    yield release,self,parking

wt=Monitor() ←
nrCars=10
initialize()
parking=Resource(capacity=4)
for i in range(nrCars):
  c=Car(id="Car %s" %i)
  activate(c,c.carDriving(driveTime=10,parkTime=uniform(5,20)))
simulate(until=100)
print "Parking time: \tmean = %2d, \n\t\tvariance=
      %2d"%(wt.mean(),wt.var())
      ↑      ↑

```

```

10.0 Car 0 gets parking space
10.0 Car 1 gets parking space
10.0 Car 2 gets parking space
10.0 Car 3 gets parking space
17.8 Car 2 releases parking space
17.8 Car 4 gets parking space
22.7 Car 0 releases parking space
22.7 Car 5 gets parking space
26.0 Car 3 releases parking space
....
34.1 Car 9 gets parking space
35.2 Car 4 releases parking space
41.7 Car 5 releases parking space
46.8 Car 8 releases parking space
51.7 Car 9 releases parking space
Parking time: mean = 13,
              variance= 19

```

Graphical Output – Example Program

```
.....
from scipy import gplt
```

Use gplt from SciPy (<http://www.scipy.org>)

```
+ class Car(Process):
  class Observer(Process):
    def __init__(self):
      Process.__init__(self)
      self.q=[]
    def observe(self):
      while True:
        yield hold,self,1
        self.q.append(len(parking.waitQ))
```

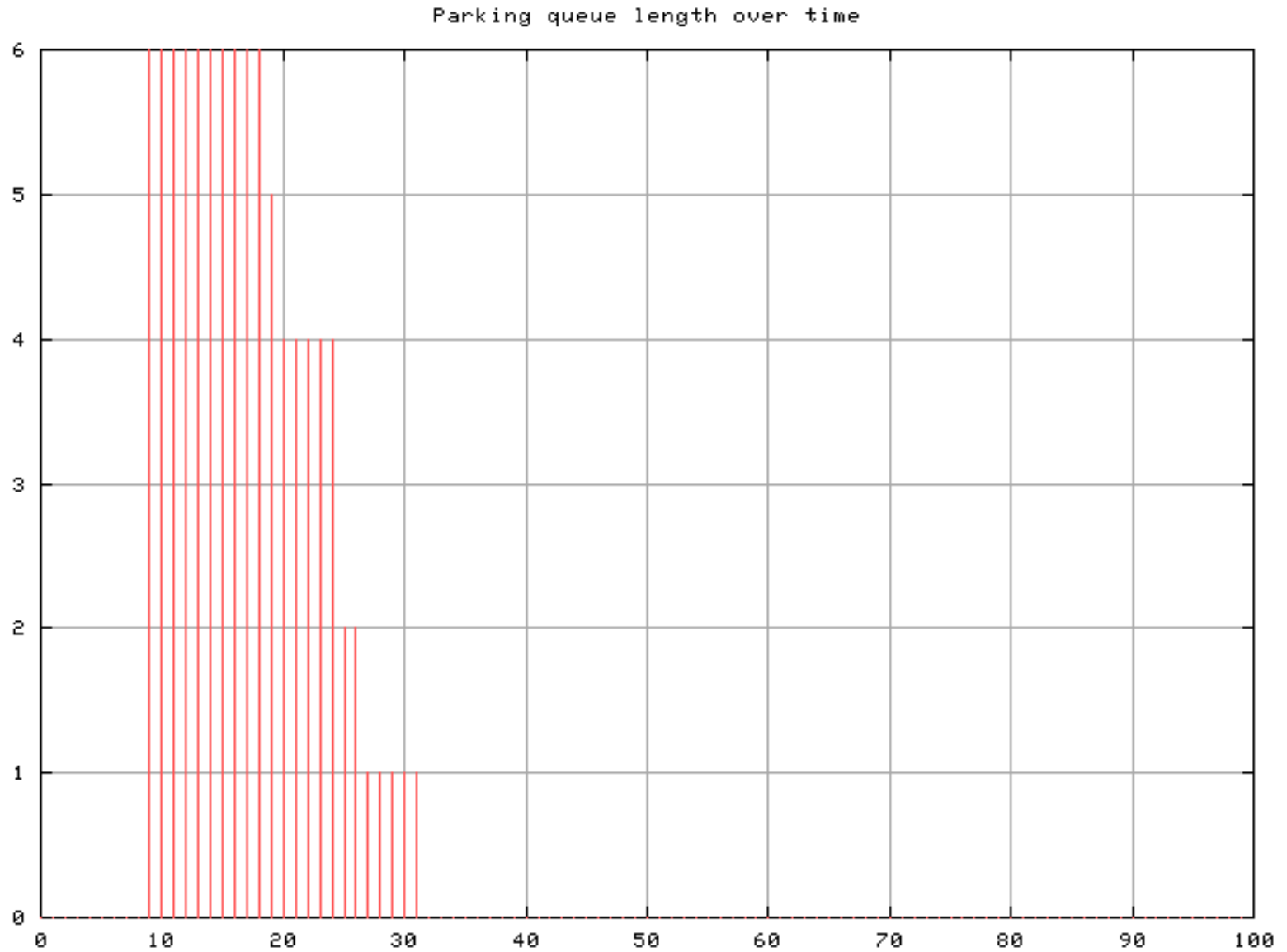
An observer process recording the parking.waitq length over time

```
nrCars=10
initialize()
ob=Observer()
activate(ob,ob.observe())
parking=Resource(capacity=4)
for i in range(nrCars):
  c=Car(id="Car %s" %i)
  activate(c,c.carDriving(driveTime=10,parkTime=uniform(5,20)))
simulate(until=100)
gplt.plot(ob.q,'notitle with impulses')
gplt.title("Parking queue length over time")
gplt.png("c:\\python22\\waitqueue.png")
```

Define plot style, title plot, output in PNG format.



Graphical Output -- Result





Graph/Network Simulation

Example: Simulation of an epidemic

```
...
from pajek import Pajek      ## http://vlado.fmf.uni-lj.si/pub/networks/pajek/

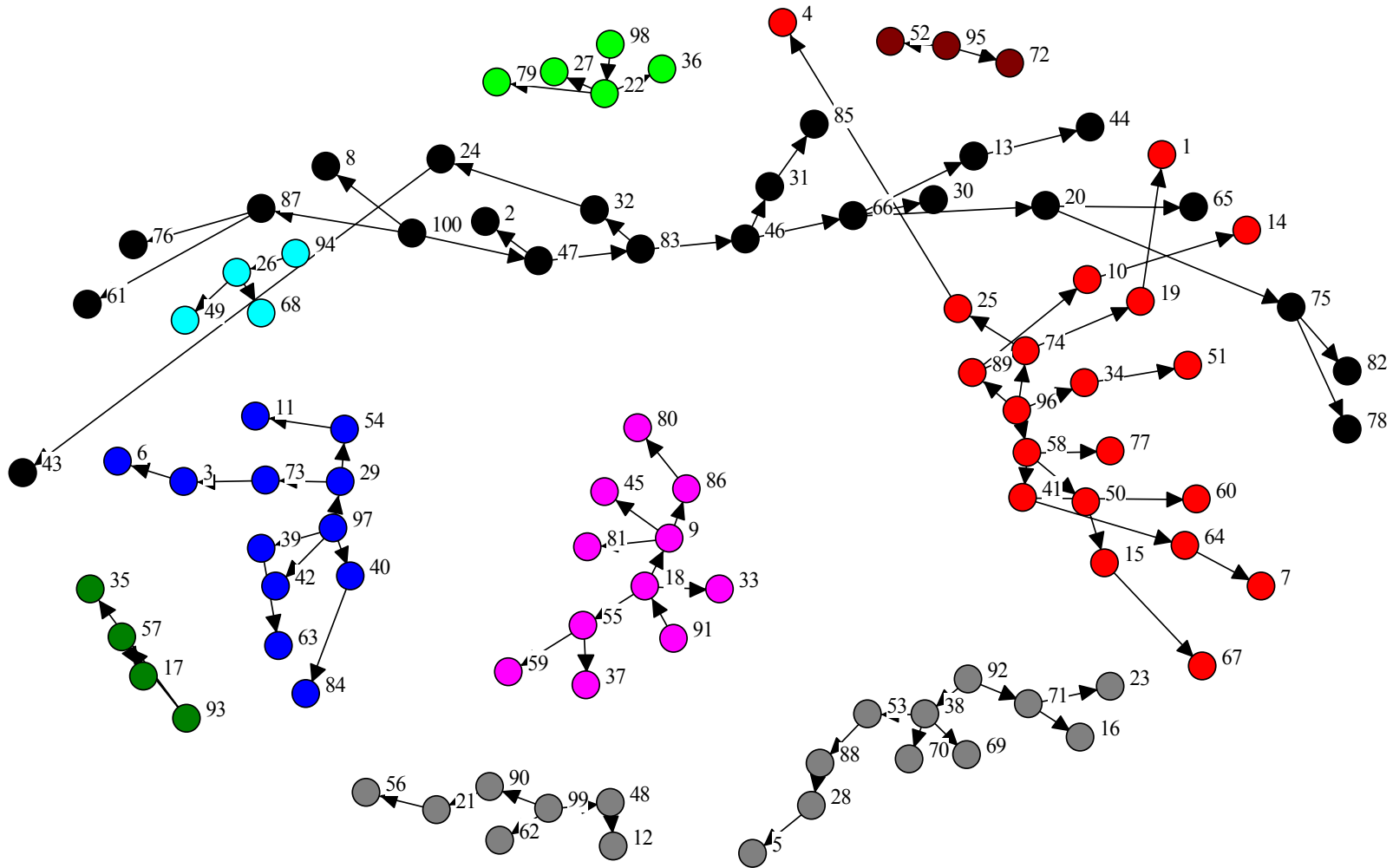
class Epidemic:
+ def __init__(self,infectProb,contactRate,rng=random.Random(1)):
+ class Network(Pajek):      ## A class to store a Pajek time events network
    class Individual(Process): ## Class of individuals which get infected and infect others
+ def __init__(self,id):
+ def contactInterval(self):
+ def chooseContact(self):

    def life(self):          ## Process Execution Method
        while self.infected==0:
            yield hold,self,self.contactInterval()
            B=self.chooseContact()
            if B.infected==1 and Epidemic.rng.random() <= Epidemic.infectProb:
                Epidemic.times.append(now())
                Epidemic.numInfected.append(Epidemic.numInfected[-1]+1)
                self.infected=1
                Epidemic.network.infectionEvent(B,self)
+ def model(nrIndividuals,contactRate,infectProb,initialInfected,runTime):
    model(nrIndividuals=100,contactRate=3,infectProb=0.003,initialInfected=10,runTime=1000)
```



Network Simulation Graphical Output

Pajek package output: “who infected whom?”



OO: Model Families Through Inheritance

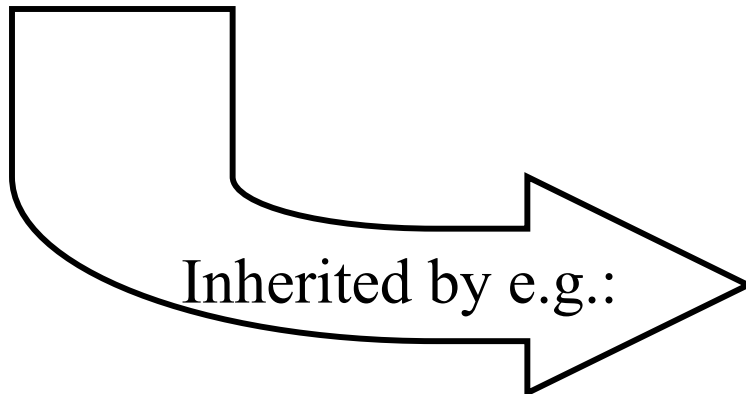
```

## CarsBase.py
from __future__ import generators
from SimPy.Simulation import *

class Car(Process):
    def __init__(self,id):
        Process.__init__(self)
        self.id=id

    def carDriving(self):
        raise "No PEM for Car defined"

```



```

from CarsBase import *

class Limo(Car):
    def __init__(self,id,cost):
        Car.__init__(self,id)
        self.costperhour=cost
    def bill(self):
        return costperhour*time/60.
    def carDriving(self):
        ...
        print bill()

class Compact(Car):
    def __init__(self,id,consumption):
        Car.__init__(self,id)
        self.consumption=consumption
    def gasUsed(self,distance):
        return self.consumption*distance
    def carDriving(self):
        ...
        print gasUsed(distance)

def model():
    initialize()
    ...
    simulate(until=...)
if __name__ == "__main__": model()

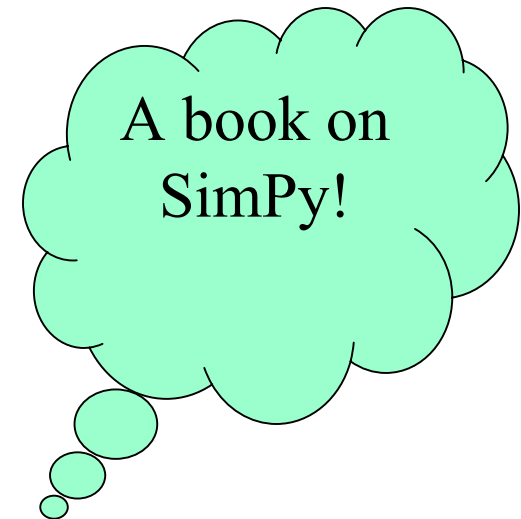
```

SimPy Application Areas So Far

- Industrial process optimization
- Performance modeling
- Computer hardware performance studies
- Air space surveillance planning
- Modeling/simulation of epidemics
- Teaching of simulation techniques

Status and Near Term Way Ahead for SimPy

- Current stable version -- SimPy 1.2:
 - Browsable documentation (manual, cheatsheet, external interface, sourcecode doc)
 - Wide range of demo models
 - Tutorial
- Planned capability extensions:
 - More synchronization constructs:
 - Producer/Consumer
 - waituntil(<general condition>)
 - Additional output analysis tools
 - Debugging tools
 - Teaching tools
- Planned API change:
 - Totally object oriented API



Future Plans

- Zope interface for web-based simulations
 - Depends on availability of version of Zope based on Python 2.2+
- SimPy in Jython
 - Depends on availability of future Jython version with generators



SimPy Resources

- SimPy web page:
 - SimPy.SourceForge.net
- SimPy project site (download):
 - sourceforge.net/projects/simpy/
- Tutorial page:
 - <http://heather.cs.ucdavis.edu/~matloff/simpy.html>
- Articles on SimPy:
 - <http://www-106.ibm.com/developerworks/linux/library/l-simpy.html?dwzone=linux>
 - <http://www.onlamp.com/pub/a/python/2003/02/27/simpy.html>