SimPy – A Discrete Event Simulation Package in Python

Dr. Klaus G. Müller (kgmuller at xs4all.nl)

EuroSciPy 2008 ***Leipzig, 26-27 July, 2008***

SimPy

About SimPy

- A Discrete Event Simulation (DES) package
  - Process based
  - Object oriented
  - Modeled after Simula and Simscript
- Written in Python
  - The **only** DES package in Python
- Open Source (LGPL)
- Development since 2002
  - Prof. Tony Vignaux (Lead user specs, documentation), Dr. Klaus Müller (Lead software)
  - International community of contributors

SimPy

Discrete Event Simulation

- Discrete-event simulation (DES):
  - operation of a system is modeled as a chronological sequence of events
  - Each event occurs at an instant in time and marks a change of state in the system
- Used to perform “what if” studies (experiments) of existing or planned systems by executing model on computer

SimPy

What has SimPy been used for?

- Teaching system simulation and Operations Research courses (universities in New Zealand, US, Canada, Venezuela, Spain, . . .)
- Simulation of epidemics
- Simulation of communications systems
- Simulation of computer hardware performance
- Simulation of nuclear processing facilities in support of designing safeguards for nuclear non-proliferation
- Comparison of Personal Rapid Transit (PRT) and classic rapid transit systems
- Simulation of air space surveillance
- Simulation of telescope management in observatory
- . . . .
- ? ? ? ?

SimPy

SimPy benefits

- Open Source
- Totally written in Python
- Source code and –documentation part of distribution
  - Insight
  - Extensibility
- Clean, small API
- Easy to use, flat learning curve
- Extensive documentation, course-tested

SimPy

Quotes from a user

- ☺“coded in less than an hour; more than 8 hours for a similar Java SSJ simulation”
- ☺“31 lines for a complete queuing simulation with statistics”
- ☺“more than 200 lines for a similar Java SSJ simulation”
- ☹“performance is an order less than a similar Java SSJ simulation”

*From www.sic.rma.ac.be/~flapierr/divers/**SimPy**.pdf*

SimPy and Co-Routines in Python

- SimPy basis: co-routines allow cooperative multitasking
- Co-routines
  - multiple entry points; suspending and resuming of execution at certain locations
- Python has generators
  - generator looks like a function but behaves like an iterator
- `yield` statement passes a value back to a parent routine
- Co-routines in SimPy
  - generators+dispatcher routine

SimPy Models Processes by Co-Routines

- SimPy simulations
  - Interactions/synchronizations between process entities over time
  - At every synchronization point (event), process entity releases control to dispatcher by `yield` with payload
    - Example: **`yield`** `hold,self,tDelay`
- SimPy process entity
  - Class instance data = process state variables
  - Generator = Process Execution Method
    - A process' lifecycle

SimPy

A SimPy Process and its Activation

```
 1  import SimPy.Simulation as Sim
 2  class Car(Sim.Process):
 3      def drive(self):
 4          """This is the Process Execution Method, a generator"""
 5          while True:
 6              #drive for 10 time units
 7              yield Sim.hold,self,10
 8              #go to sleep
 9              yield Sim.passivate,self
10
11  #create a Car instance
12  c=Car()
13  #activate this car's process in 15 time units
14  Sim.activate(c,c.drive(),at=15)
```

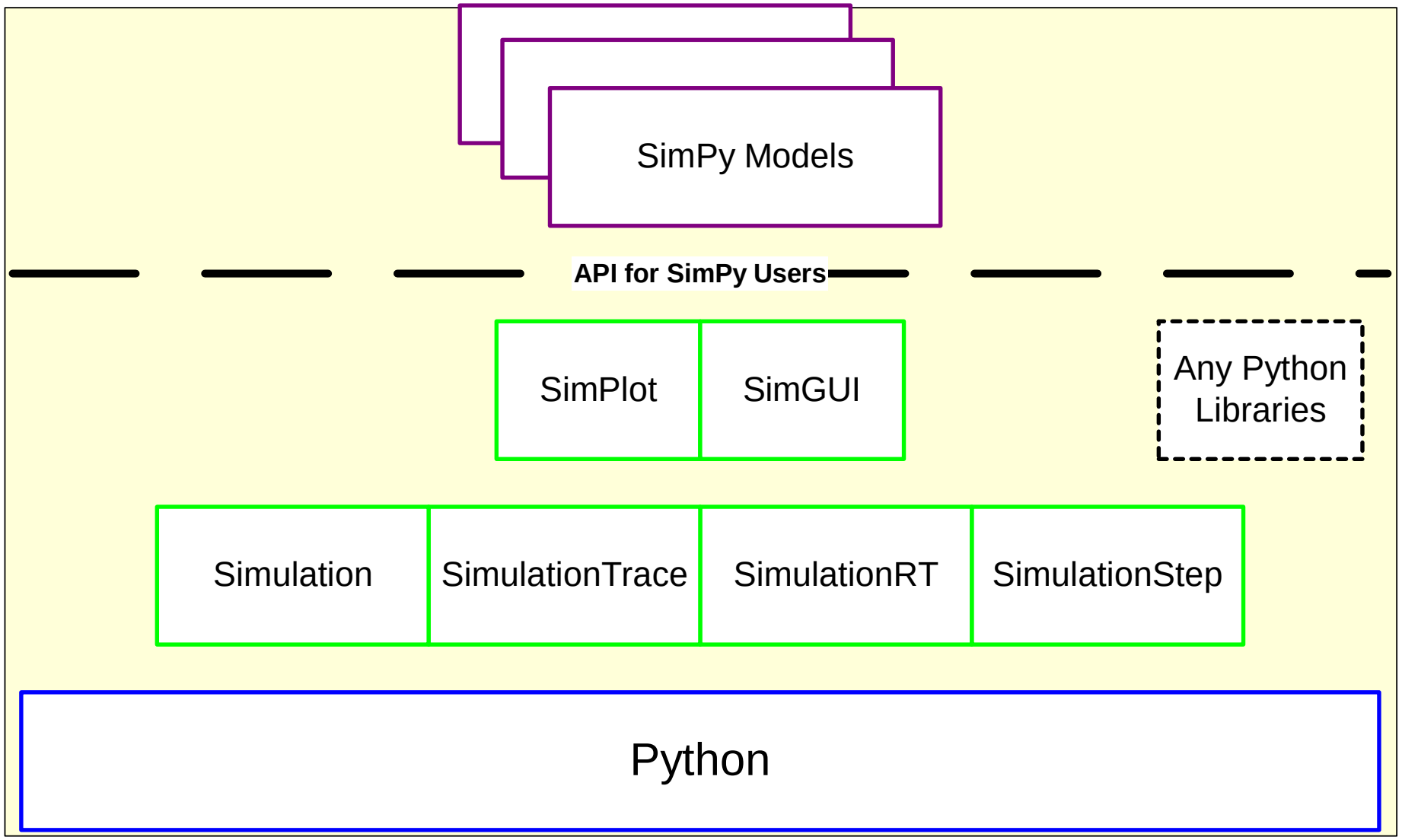
SimPy Co-Routine Machinery

```
hold=0; passivate=1 ## and many more function codes
events=[]
tNow=0.0
def post(who,when): ##Schedule next activation for process 'who'
    events.append((when,who)); events.sort()
def activate(who,generator,when=0): ## Make new process runnable
    who.nextpoint=generator
    post(who,when) ## first event for "who"
def holdfun(who,delay):
    post(who,tNow+delay) ## next event for "who"
def passivatefun(who,ignored=None):
    pass ## no scheduled event for "who"; wait for wakeup
def simulate(tUntil): ## Activate co-routines in time order
    global tNow
    dispatch={hold:holdfun,passivate:passivatefun} #etc
    while events and tNow<tUntil:
        activationTime,who=events.pop(0)
        tNow=activationTime
        try:
            nextEvent=who.nextpoint.next() ## pass control to "who"
            actioncode,who,actionpar=nextEvent ## the yield payload
            dispatch[actioncode](who,actionpar) ## execute command
        except StopIteration: who.nextpoint=None ## "who" terminated
```

Process Synchronizations in SimPy

- A SimPy process can wait for
  - fixed time period (a delay)
  - re-activation by another process
  - resource to become available
  - event to be signaled
  - general condition (predicate of state variables)
- It can
  - (Re-)activate another process
  - Put itself to sleep
  - Put another process to sleep
  - Interrupt another process
  - Preempt another process queuing for a resource

SimPy Module Structure and User API



SimPy Modules/Libraries

- Simulation modules
  - SimPy.Simulation : Discrete Event Simulation (DES)
  - SimPy.SimulationTrace: DES with event trace
  - SimPy.SimulationRT: DES with real time synchronization
  - SimPy.SimulationStep: DES with event stepping under user control
- SimPy utility modules
  - SimPy.SimGUI: Simulation GUI API
  - SimPy.SimPlot: Basic plotting API

SimPy

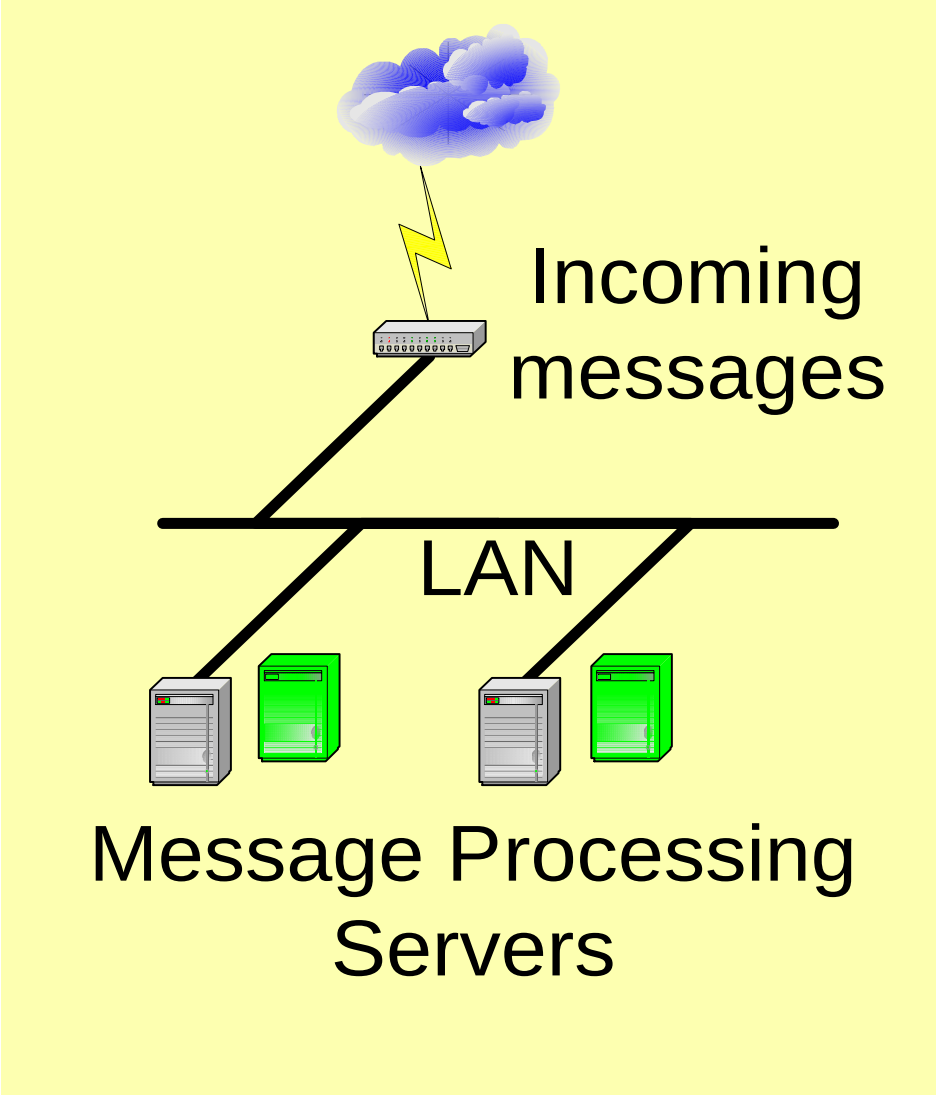
SimPy.Simulation: Discrete Event Simulation

Needed for DES	SimPy.Simulation provides
Entities	**class** `Process`, **class** `Resource`, **class** `Store`, **class** `Level`
Data collectors	**class** `Monitor`, **class** `Tally`
Stochastic distributions	Python's `Random` library
Scheduler entity	Dispatch loop in function `simulate`
Quasi-parallelism	Co-routines, based on Python generators

SimPy

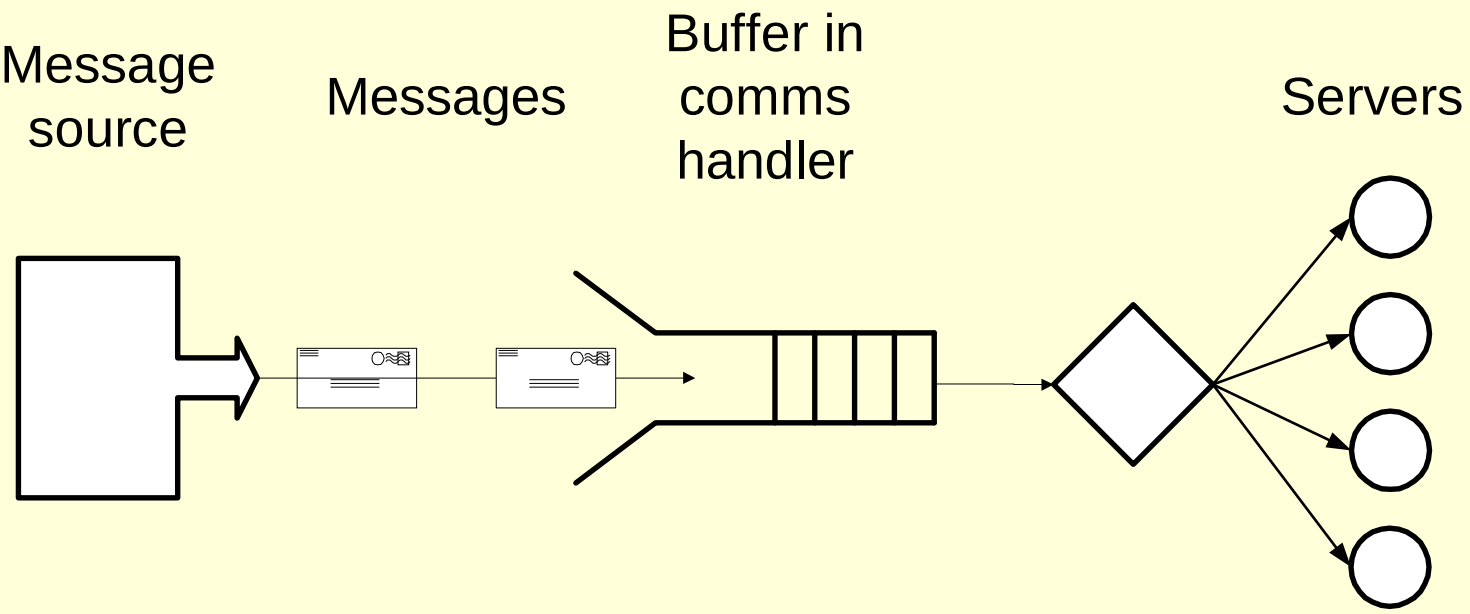
A SimPy Example: Problem Scenario

- Messages arrive in a LAN at a rate of about 1/minute (calculated from 1000 data points collected)
- They are processed by 2 **expensive** servers at 1 minute/message
- ***To investigate:*** Could 4 **cheaper, slower** servers (2 min./message) do at least the same job?
  - **Figures of merit:**
    - **Message delays**
    - **Nr of messages delayed**



Mapping Example
Model=>SimPy Objects

Abstract model



def generateMessages()

Resource(capacity=nrServers)

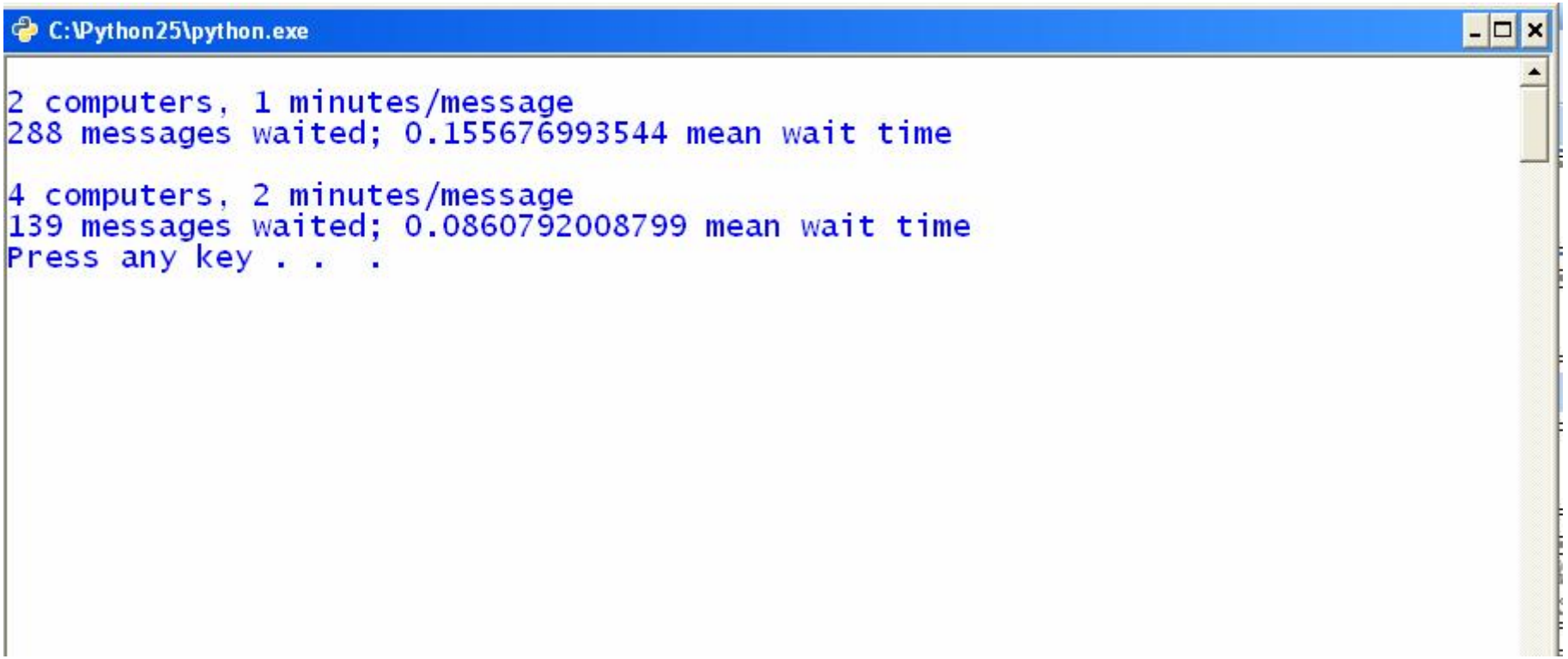
class Message(Process)

SimPy objects

```
 1  import SimPy.Simulation as Sim
 2  def generateMessages(datafile,messageHandler,servtime):
 3      for item in datafile:
 4          p=Message()
 5          Sim.activate(p,p.action(messageHandler,servtime),at=float(item))
 6  class Message(Sim.Process):
 7      def action(self,messageHandler,servtime):
 8          tstart=Sim.now()
 9          yield Sim.request,self,messageHandler
10          waited=Sim.now()-tstart
11          obs.append((Sim.now(),waited))
12          yield Sim.hold,self,servtime
13          yield Sim.release,self,messageHandler
14  def run():
15      global obs,computers
16      for computers,servtime in ((2,1),(4,2)):
17          obs=[]
18          infile=open(r".\arrFile.txt","r")
19          Sim.initialize()
20          messageHandler=Sim.Resource(capacity=computers,name="servers")
21          generateMessages(infile,messageHandler,servtime)
22          Sim.simulate(until=2000)
23          print "\n%s computers, %s minutes/message" %(computers,servtime)
24          print "%s messages waited; %s mean wait time" \
25              %(len([x for x in obs if x[1]>0]),
26              sum([y[1] for y in obs])/len(obs))
27  run()
28  raw_input("Press any key . . .")
```

Run!

SimPy.Simulation
Example Output



SimPy

SimPy.SimulationTrace: Opening the Black Box

- Insight into parallel processes and their interaction difficult
  - same problem occurs with simulation
- Module **SimulationTrace** traces all events to show what is happening behind the scenes
- Just replace

  from SimPy.Simulation **import** *

  with

  from SimPy.SimulationTrace **import** *
- Trace output can be redirected to a file
- Event types to be traced can be selected

```
 1  import SimPy.SimulationTrace as Sim
 2  def generateMessages(datafile,messageHandler,servtime):
 3      for item in datafile:
 4          p=Message("Message %s"%float(item))
 5          Sim.activate(p,p.action(messageHandler,servtime),at=float(item))
 6  class Message(Sim.Process):
 7      def action(self,messageHandler,servtime):
 8          tstart=Sim.now()
 9          yield Sim.request,self,messageHandler
10          waited=Sim.now()-tstart
11          obs.append((Sim.now(),waited))
12          yield Sim.hold,self,servtime
13          yield Sim.release,self,messageHandler
14  def run():
15      global obs,computers
16      for computers,servtime in ((2,1),(4,2)):
17          Sim.trace.tchange(outfile=open(r".\message_trace.txt","w"))
18          obs=[]
19          infile=open(r".\arrFile.txt","r")
20          Sim.initialize()
21          messageHandler=Sim.Resource(capacity=computers,name="servers")
22          generateMessages(infile,messageHandler,servtime)
23          Sim.simulate(until=2000)
24          print "\n%s computers, %s minutes/message"%(computers,servtime)
25          print "%s messages waited; %s mean wait time"\
26              %(len([x for x in obs if x[1]>0]),
27              sum([y[1] for y in obs])/len(obs))
28  run()
29  raw_input("Press any key . . .")
```

View trace

SimPy.SimulationTrace Example Output

```
0 activate <Message 0.0580196497695> at time: 0.0580196497695 prior: False
0 activate <Message 1.20632086606> at time: 1.20632086606 prior: False
0 activate <Message 1.47841256495> at time: 1.47841256495 prior: False
0 activate <Message 2.84252523542> at time: 2.84252523542 prior: False
0 activate <Message 6.84300934018> at time: 6.84300934018 prior: False
0 activate <Message 7.65546313948> at time: 7.65546313948 prior: False
0 activate <Message 7.68086372638> at time: 7.68086372638 prior: False
0 activate <Message 7.75521885492> at time: 7.75521885492 prior: False
0 activate <Message 8.27563476256> at time: 8.27563476256 prior: False
0 activate <Message 10.0767483281> at time: 10.0767483281 prior: False
0 activate <Message 11.999519201> at time: 11.999519201 prior: False
0 activate <Message 12.7071946816> at time: 12.7071946816 prior: False
0 activate <Message 12.9183245122> at time: 12.9183245122 prior: False
0 activate <Message 15.1756150312> at time: 15.1756150312 prior: False
0 activate <Message 15.1882493038> at time: 15.1882493038 prior: False
0 activate <Message 17.6958705982> at time: 17.6958705982 prior: False
0 activate <Message 18.2389977743> at time: 18.2389977743 prior: False
0 activate <Message 18.6258668918> at time: 18.6258668918 prior: False
0 activate <Message 19.703615543> at time: 19.703615543 prior: False
0 activate <Message 20.270252137> at time: 20.270252137 prior: False
0 activate <Message 21.4333596365> at time: 21.4333596365 prior: False
0 activate <Message 22.2700164406> at time: 22.2700164406 prior: False
0 activate <Message 22.310170991> at time: 22.310170991 prior: False
0 activate <Message 24.2933642542> at time: 24.2933642542 prior: False
0 activate <Message 25.8038811791> at time: 25.8038811791 prior: False
0 activate <Message 26.0836116412> at time: 26.0836116412 prior: False
0 activate <Message 29.9211289433> at time: 29.9211289433 prior: False
0 activate <Message 34.8666724693> at time: 34.8666724693 prior: False
0 activate <Message 37.4417294622> at time: 37.4417294622 prior: False
```

SimPy.SimulationRT: Synchronize With Real Time

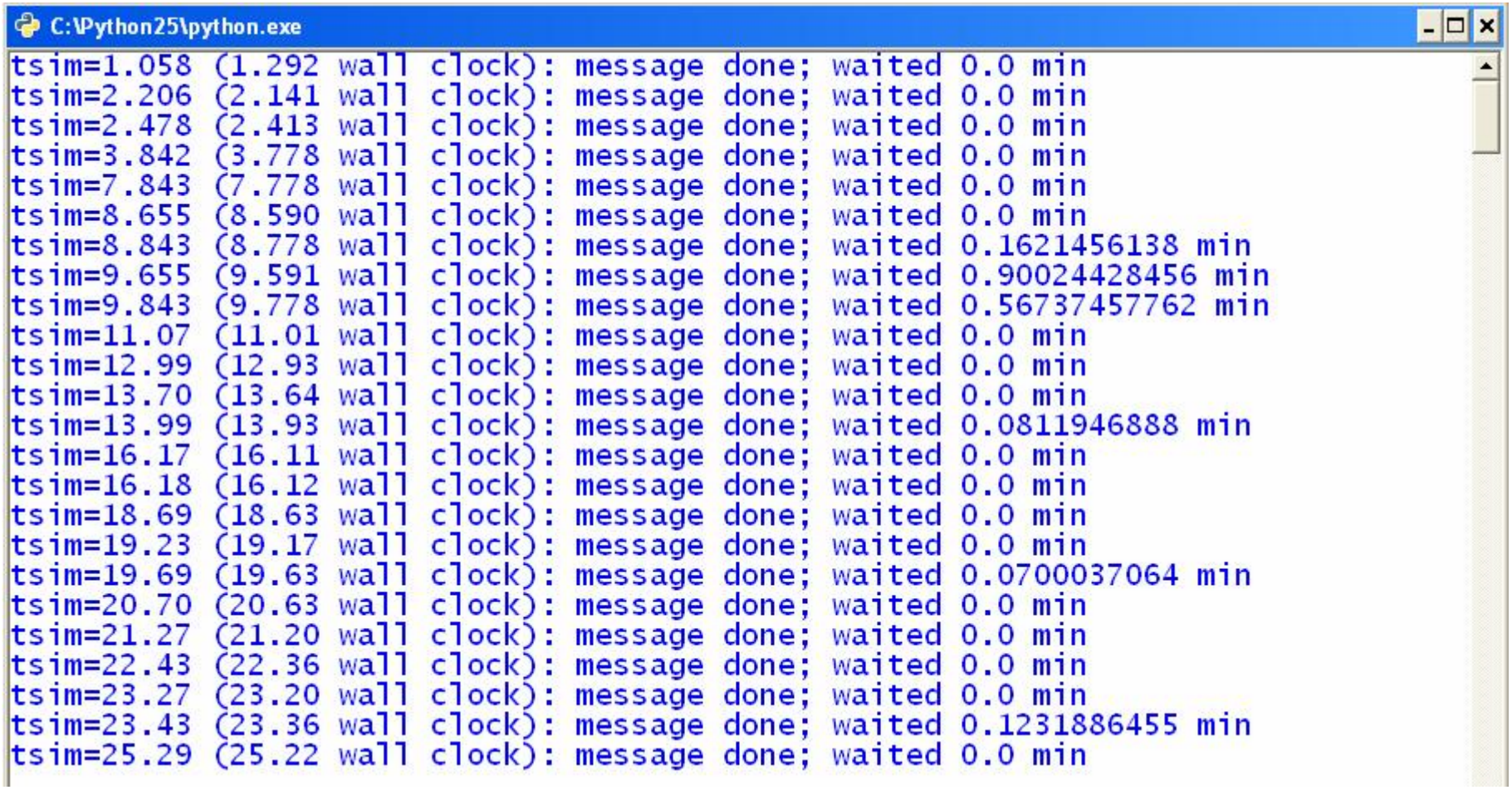
- For simulation user interaction e.g. in game applications or animation, events should appear to be spaced in real time
- ***SimulationRT*** allows tying simulation time to wall clock time
- Example:

  simulate(real_time=True,relSpeed=0.2,until= …) runs 1 simulation time unit in 5 real seconds
- Module works better under Windows than under Unix or Linux
  - Problem stems from time.clock( )

```
 1  import SimPy.SimulationRT as Sim
 2  import time
 3  def generateMessages(datafile,messageHandler,servtime):
 4      for item in datafile:
 5          p=Message()
 6          Sim.activate(p,p.action(messageHandler,servtime),at=float(item))
 7  class Message(Sim.Process):
 8      def action(self,messageHandler,servtime):
 9          tstart=Sim.now()
10          yield Sim.request,self,messageHandler
11          waited=Sim.now()-tstart
12          obs.append((Sim.now(),waited))
13          yield Sim.hold,self,servtime
14          yield Sim.release,self,messageHandler
15          print "tsim=%5.5s (%5.5s wall clock): message done; waited %s min"\
16                %(Sim.now(),time.clock()-tbegin,waited)
17  def run():
18      global obs,computers,tbegin
19      for computers,servtime in ((2,1),(4,2)):
20          obs=[]
21          infile=open(r".\arrFile.txt","r")
22          Sim.initialize()
23          messageHandler=Sim.Resource(capacity=computers,name="servers")
24          generateMessages(infile,messageHandler,servtime)
25          tbegin=time.clock()
26          Sim.simulate(until=2000,real_time=True,rel_speed=1)#1 min=1 sec RT
27          print "\n%s computers, %s minutes/message"%(computers,servtime)
28          print "%s messages waited; %s mean wait time"\
29              %(len([x for x in obs if x[1]>0]),
30              sum([y[1] for y in obs])/len(obs))
31  run()
32  raw_input("Press any key . . .")
```

Run!

SimPy.SimulationRT Example Output



SimPy

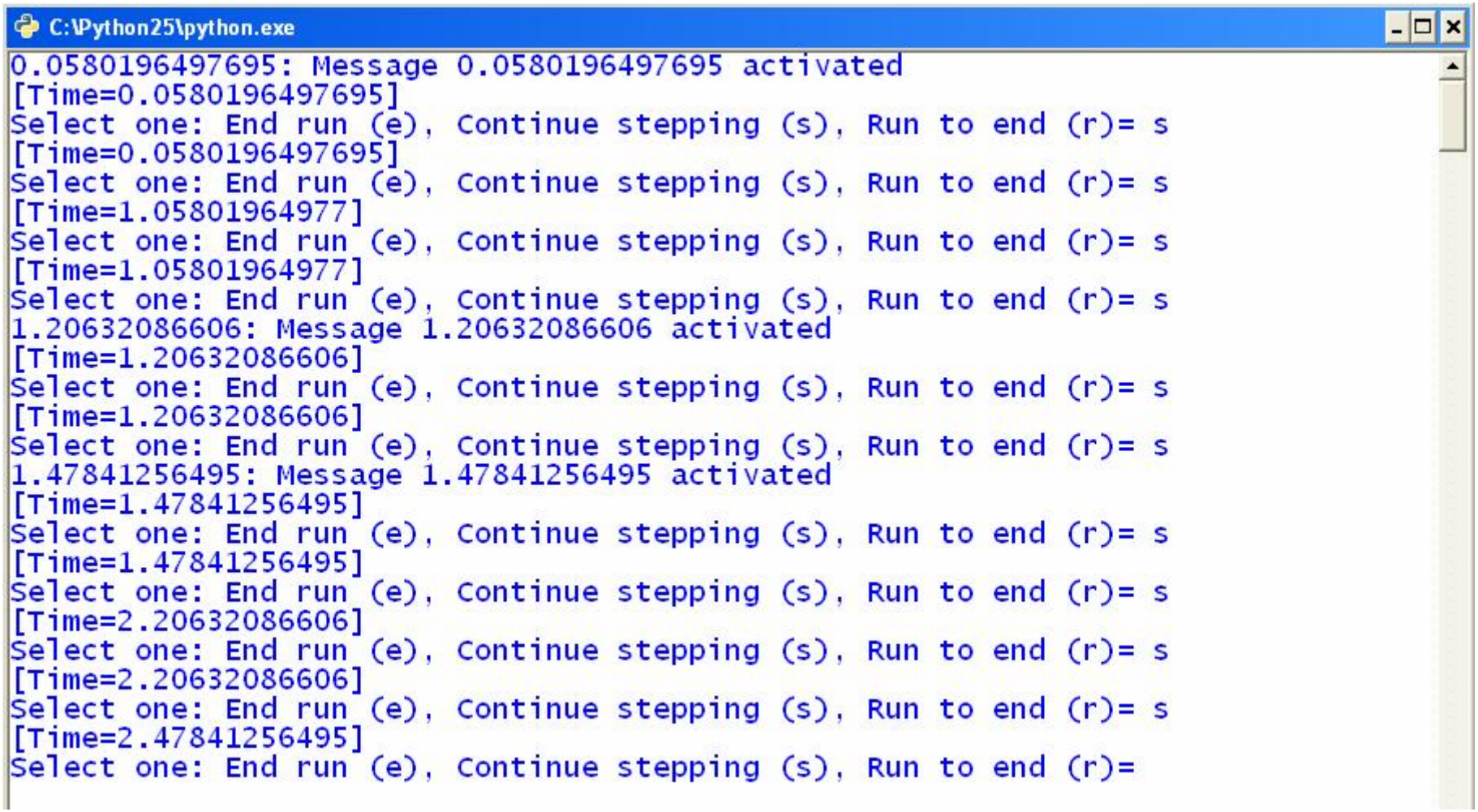
SimPy.SimulationStep: Stepping Event by Event

- For event by event user control
  - Debugging
  - Insertion of events or parameter changes by user
- User-provided callback function gets called for every event

```
 1  import SimPy.SimulationStep as Sim
 2  def generateMessages(datafile,messageHandler,servtime):
 3      for item in datafile:
 4          p=Message("Message %s"%float(item))
 5          Sim.activate(p,p.action(messageHandler,servtime),at=float(item))
 6  class Message(Sim.Process):
 7      def action(self,messageHandler,servtime):
 8          print "%s: %s activated"%(Sim.now(),self.name)
 9          . . . .
10  def callbackUserControl():
11      while True:
12          a=raw_input("[Time=%s]\nSelect one: End run (e), "\
13                      "Continue stepping (s), Run to end (r)= "%Sim.now())
14          if a=="e": Sim.stopSimulation();break
15          elif a=="s": return
16          elif a=="r": Sim.stopStepping();break
17          else: print "'%s' illegal command"%a
18  def run():
19      global obs,computers
20      for computers,servtime in ((2,1),(4,2)):
21          . . . .
22          Sim.startStepping()
23          Sim.simulate(callback=callbackUserControl,until=2000)
24          print "\n%s computers, %s minutes/message"%(computers,servtime)
25          print "%s messages waited; %s mean wait time"\
26              %(len([x for x in obs if x[1]>0]),
27              sum([y[1] for y in obs])/len(obs))
28  run()
29  raw_input("Press any key . .  .")
```

Run!

SimPy.SimulationStep Example Output



SimPy

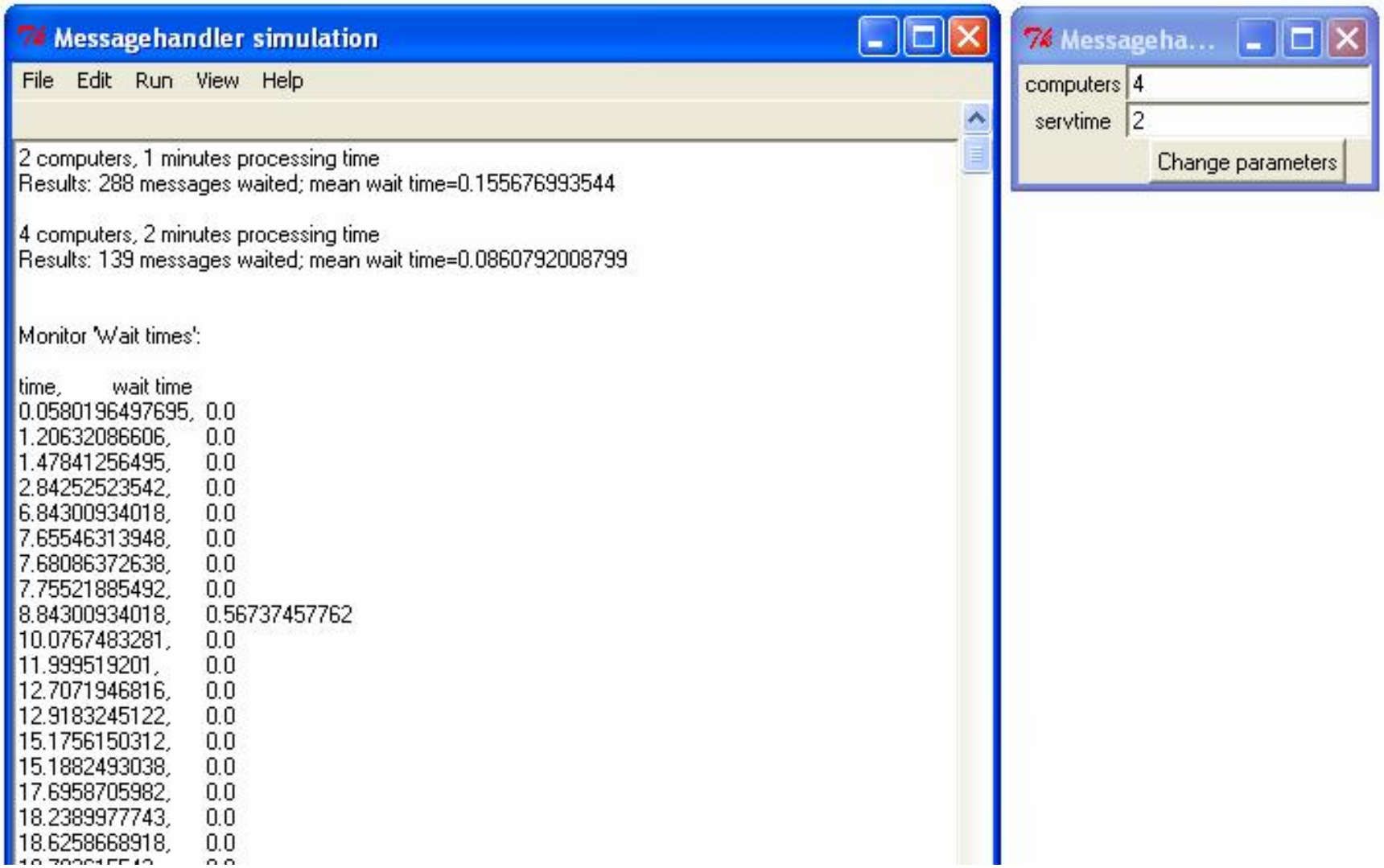
SimPy.SimGUI: Graphical User Interface

- Support for developing simulation user GUI
  - Point-and-click
  - Simulation start
  - Parameter changes
  - Display/saving of results (raw and analyzed)
  - Model description
- Tkinter-based simulation GUI framework

```
__doc__="""Messagehandler simulation. Messages arrive in a LAN
and are processed by 'computers' servers at a rate of 'servtime'
minutes per message.
"""
import SimPy.Simulation as Sim
from SimPy.SimGUI import *

def generateMessages(datafile,messageHandler,servtime):
    . . . .
class Message(Sim.Process):
    def action(self,messageHandler,servtime):
        . . . .
def run():
    gui.observations=Sim.Monitor("Wait times",ylab="wait time",tlab="time")
    computers=gui.params.computers
    infile=open(r".\arrFile.txt","r")
    Sim.initialize()
    messageHandler=Sim.Resource(capacity=computers,name="servers")
    generateMessages(infile,messageHandler,gui.params.servtime)
    Sim.simulate(until=2000)
    gui.noRunYet=False
    gui.writeConsole(
    "%s computers, %s minutes processing time\nResults: %s messages waited; mean wait time=%s\n" \
            %(gui.params.computers,gui.params.servtime,
            len([x for x in gui.observations if x[1]>0]),
            gui.observations.mean()))
class MyGUI(SimGUI):
    def __init__(self,win,**par):
        SimGUI.__init__(self,win,**par)
        self.run.add_command(label="Run messagehandler simulation",
                             command=run,underline=0)
        self.params=Parameters(computers=2,servtime=1)
root=Tk()
gui=MyGUI(root,title="Messagehandler simulation",doc=__doc__,consoleHeight=40)
gui.mainloop()
```

SimPy.SimGUI
Example Output

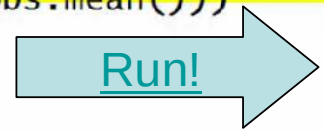


SimPy

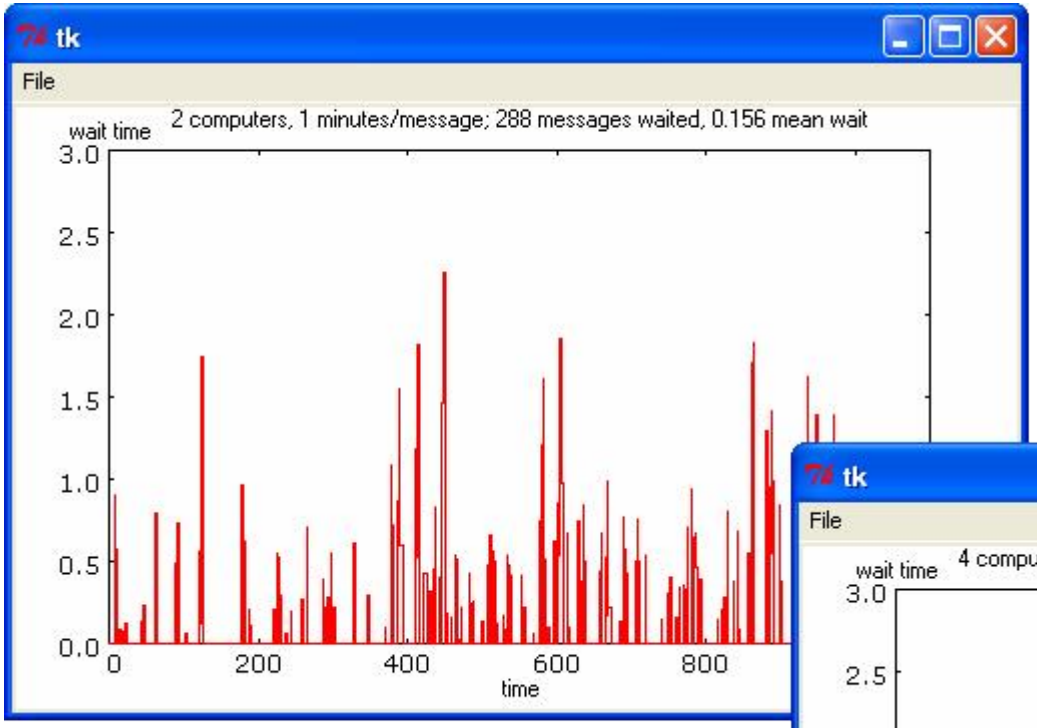
SimPy.SimPlot: Plotting Simulation Results

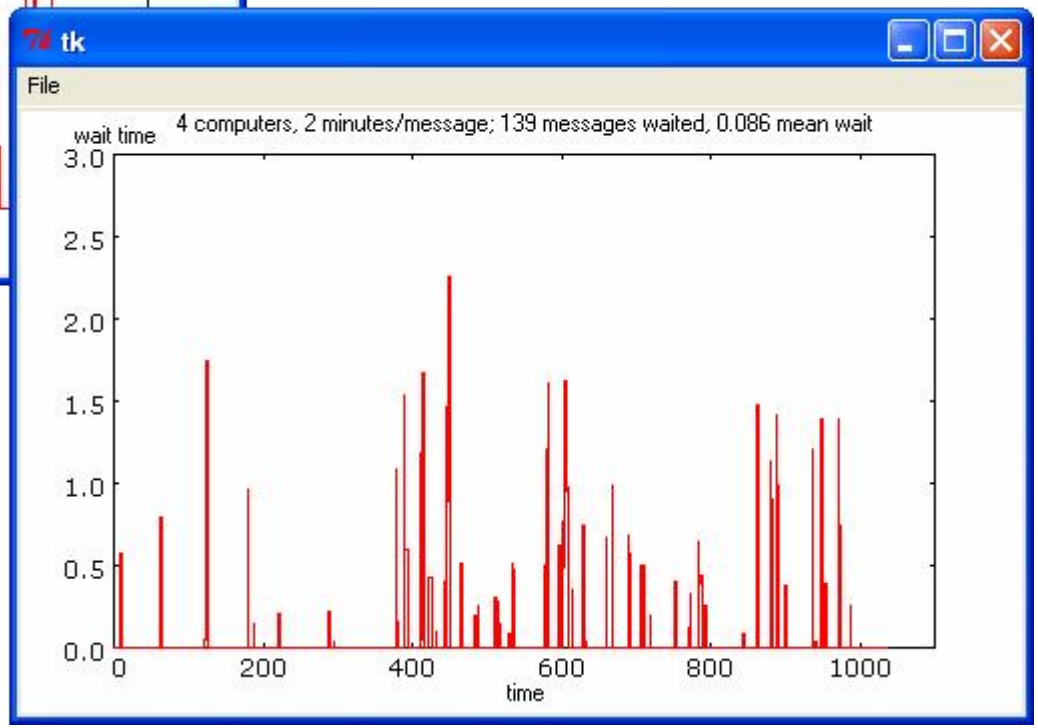
- Simple, out-of-the-box plotting package
- Tkinter-based
  - Derived from Konrad Hinsen's plotting module
- Plots (time,value) time-series data from Monitor instances
- Plot types:
  - Line, bar, step, histogram, scatter
- Basic and advanced API

```
 1  import SimPy.Simulation as Sim
 2  import SimPy.SimPlot as Plot
 3  def generateMessages(datafile,messageHandler,servtime):
 4      for item in datafile:
 5          p=Message("Message %s"%float(item))
 6          Sim.activate(p,p.action(messageHandler,servtime),at=float(item))
 7  class Message(Sim.Process):
 8      def action(self,messageHandler,servtime):
 9          tstart=Sim.now()
10          yield Sim.request,self,messageHandler
11          waited=Sim.now()-tstart
12          obs.observe(t=Sim.now(),y=waited)
13          yield Sim.hold,self,servtime
14          yield Sim.release,self,messageHandler
15  def run():
16      global obs,computers
17      for computers,servtime in ((2,1),(4,2)):
18          obs=Sim.Monitor(name="Wait time data",ylab="wait time",tlab="time")
19          infile=open(r".\arrFile.txt","r")
20          Sim.initialize()
21          messageHandler=Sim.Resource(capacity=computers,name="servers")
22          generateMessages(infile,messageHandler,servtime)
23          Sim.simulate(until=2000)
24          plt=Plot.SimPlot()
25          plt.plotStep(obs,color="red",
26          title="%s computers, %s minutes/message; %s messages waited, %4.3f mean wait"\
27          %(computers,servtime,len([x for x in obs.yseries() if x>0]),obs.mean()))
28          plt.mainloop()
29  run()
30  raw_input("Press any key . . .")
```



SimPy.SimPlot
Example Output





SimPy

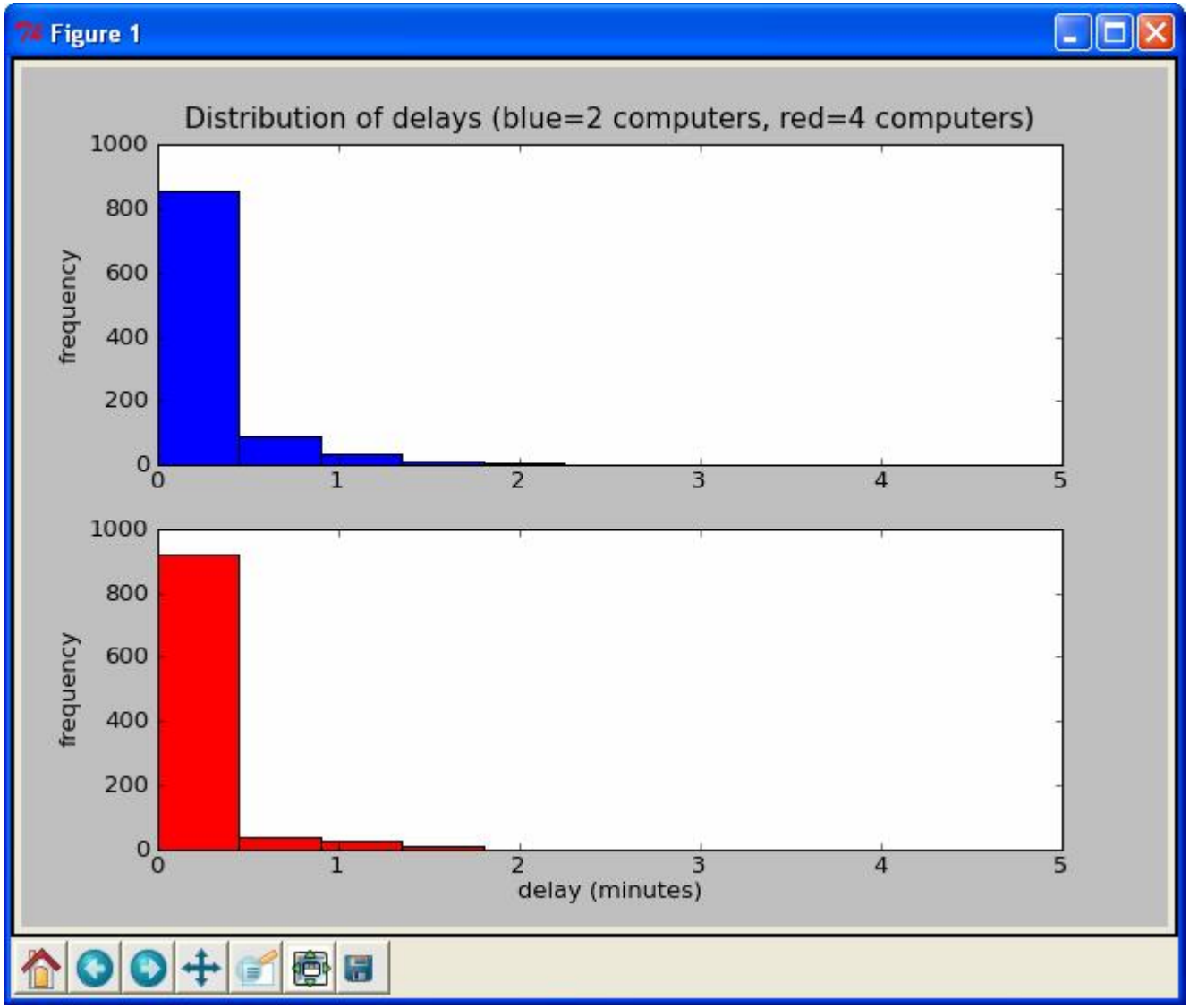
Publication-Quality Plotting with Matplotlib

- SimPlot is only intended as quick, first-level analysis tool
- Not optimal for publication quality plots
  - File format limited to Postscript
  - Limited plot types
- SciPy's Matplotlib is recommended
  - Easy interface to SimPy
  - Great range of plot types/formats
  - Many file formats (PNG, EMF, EPS, PDF, PS, RAW, SVG)

```
 1  import SimPy.Simulation as Sim
 2  import pylab
 3  def generateMessages(datafile,messageHandler,servtime):
 4      . . . .
 5  class Message(Sim.Process):
 6      def action(self,messageHandler,servtime):
 7          . . . .
 8  def run():
 9      global obs,computers
10      obslist=[]
11      for computers,servtime in ((2,1),(4,2)):
12          obs=(Sim.Monitor())
13          . . . .
14          Sim.simulate(until=2000)
15          obslist.append(obs)
16      pylab.subplot(211)
17      pylab.title("Distribution of delays (blue=2 computers, red=4 computers)")
18      n, bins, patches = pylab.hist(obslist[0].yseries(), 5,fc="b")
19      pylab.xlim(0,5);pylab.ylim(0,1000)
20      pylab.ylabel("frequency")
21      pylab.subplot(212)
22      n, bins, patches = pylab.hist(obslist[1].yseries(), 5,fc='r')
23      pylab.xlim(0,5);pylab.ylim(0,1000)
24      pylab.ylabel("frequency");pylab.xlabel("delay (minutes)")
25      pylab.show()
26  run()
27  raw_input("Press any key . .  .")
```

SimPy

Matplotlib Example Output



SimPy Release 1.9.1 Documentation in Distribution

- User manual
- “Once over lightly” user manual
- Cheat sheet
- Two tutorials
- Manuals for all simulation and utility libraries
- Source code documentation in HTML (automatically generated by Epydoc)
- Many SimPy simulation models

SimPy and SciPy

- Collaboration with/visibility in SciPy community sought
- Future SimPy versions will have NumPy, Matplotlib interfaces and documentation
- Inclusion in Enthought SciPy distribution sought

SimPy Web Resources

- SimPy web site http://SimPy.SourceForge.Net
- Outstanding online simulation textbook by Prof. Norman Matloff (U. of California, Davis, U.S.)
  http://heather.cs.ucdavis.edu/~matloff/simcourse.html
- SimPy course notes by Prof. Tony Vignaux (Victoria U., Wellington, New Zealand)
  http://www.mcs.vuw.ac.nz/courses/OPRE352/2008T2/Lecture-Notes/
- Downloads from SimPy web site
- SimPy wiki http://www.mcs.vuw.ac.nz/cgi-bin/wiki/SimPy
- Mailing lists for users, developers, CVS commits
- CVS repository on SourceForge
  http://sourceforge.net/cvs/?group_id=62366