Discrete Event Simulation Using SimPy

SYS-611: Simulation and Modeling

Paul T. Grogan, Ph.D.
Assistant Professor
School of Systems and Enterprises

Agenda

1. Simulation Software
2. Introduction to SimPy
3. Queuing System Model
4. Inventory System Model
5. Factory System Model

Reading: A.M. Law, “Simulation Software,” Ch. 3 in *Simulation Modeling and Analysis*, 5th Edition, 2013, pp. 181-213.

SimPy Reference Material: http://simpy.readthedocs.io

Developing Simulation Software

Simulation Software

Based on A.M. Law (2013), pp. 186-193

- Two main options for developing simulations:
  - General-purpose programming languages – simulation is a software engineering project
    - Java, C++, C most common in defense domain
  - Simulation packages – simulation is an application in an encompassing software framework/tool
    - General-purpose or application-oriented
- Choosing the best option will depend on the specifics of each problem

Simulation Package Examples

Based on A.M. Law (2013), pp. 186-193

- General-purpose:
  - AnyLogic
  - Arena
  - ExtendSim
  - GPSS (language)
  - Simio
  - Simula (language)
  - SIMUL8

- Application-oriented:
  - *Manufacturing*: FlexSim, ProModel
  - *Communications*: NetSim, OPNET++
  - *Healthcare*: FlexSim
  - *Multi-physics*: Fluent, COMSOL
  - *Controls*: Simulink

Desirable Software Features

Based on A.M. Law (2013), pp. 186-193

- Flexible modeling
- Hierarchical structure
- Usable / graphical user interface
- Command line / headless execution
- Debugging aids
- Rapid feedback (<1s)
- Import/export data
- Serialize/save state
- Low cost
- Cross-platform
- Permissive licenses for extension, distribution
- RNG, statistical tools
- Documentation and user community

Comparison of Simulation Software

Programming Lang.

- Ultimate flexibility
- Interoperability
- Higher performance (if done right)
- High level of control
- Lower infrastructure costs, higher developer costs

Simulation Package

- Easier verification
- Build on existing work
- Easier learning curve
- Execution often optimized
- Higher infrastructure costs, lower developer costs

Introduction to SimPy

SimPy Overview

- **SimPy** is a process interaction-based discrete event simulation package in standard Python
  - Provides application programming interface (API)
  - Distributed under MIT License (permissive)
- Install from Anaconda Command Prompt:

```
pip install simpy
```

- Define processes using Python generators:
  - Each active entity (e.g. customer) has a process
  - Allow shared resources (e.g. servers) and interrupts

SimPy History

- 2003-2008: Version 1.X
  - Inspired by Simula 67 and Simscript
  - Goal: native simulation in Python
- 2009-2012: Version 2.X
  - Object-oriented API
- 2013-Present: Version 3.X
  - Completely rewritten from scratch
  - New and easier-to-use API conforming to community guidelines (PEP 8)

Python: Iterables

http://stackoverflow.com/questions/231767/what-does-the-yield-keyword-do/231855#231855

- An **iterable** is a data structure that allows iteration:

```
my_list = [0,1,4]
for i in my_list:
    print i
```

- Iterables can also be defined with "list comprehension" notation:

```
my_list = [x*x for x in range(3)]
for i in my_list:
    print i
```

Python: Generators

http://stackoverflow.com/questions/231767/what-does-the-yield-keyword-do/231855#231855

- A **generator** is a one-time-use iterable

```
my_generator = (x*x for x in range(3))
for i in my_generator:
    print i
```

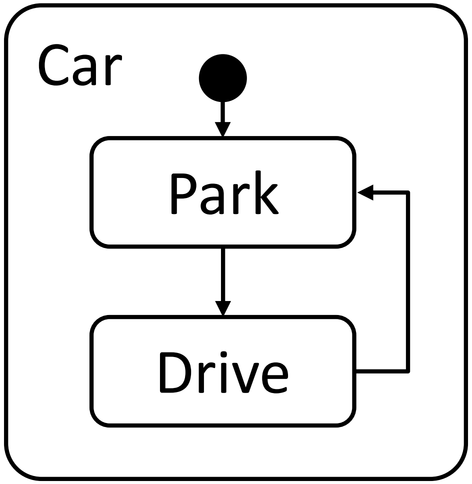
- Generators can also be defined in function notation using the *yield* keyword (similar to *return*):

```
def create_generator():
    for x in range(3):
        yield x*x
my_generator = create_generator()
for i in my_generator:
    print i
```

SimPy Processes

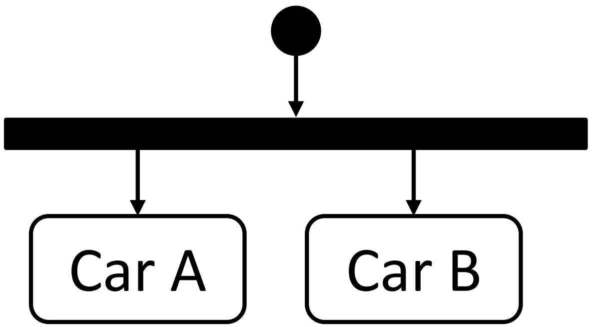
- SimPy uses generators to define processes
  - A process is an activity with > 0 duration

```
import simpy # simpy1.py

def car_process(env):
  while True:
    print 'parking @ {}'.format(env.now)
    yield env.timeout(5.0) # wait 5 hours
    print 'driving @ {}'.format(env.now)
    yield env.timeout(2.0) # wait 2 hours

env = simpy.Environment()
env.process(car_process(env))
env.run(until=15.0)
```



Multiple Processes

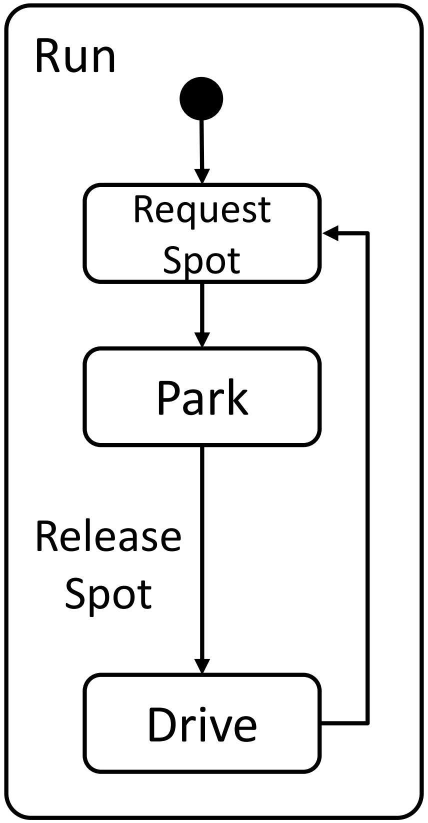
```
import simpy # simpy2.py

def car_process(env, name, t_park, t_drive):
  while True:
    print '{} parking @ {}'.format(
      name, env.now)
    yield env.timeout(t_park)
    print '{} driving @ {}'.format(
      name, env.now)
    yield env.timeout(t_drive)

env = simpy.Environment()
env.process(car_process(env, 'A', 5.0, 2.0))
env.process(car_process(env, 'B', 3.0, 3.0))
env.run(until=15.0)
```



Process-oriented SimPy

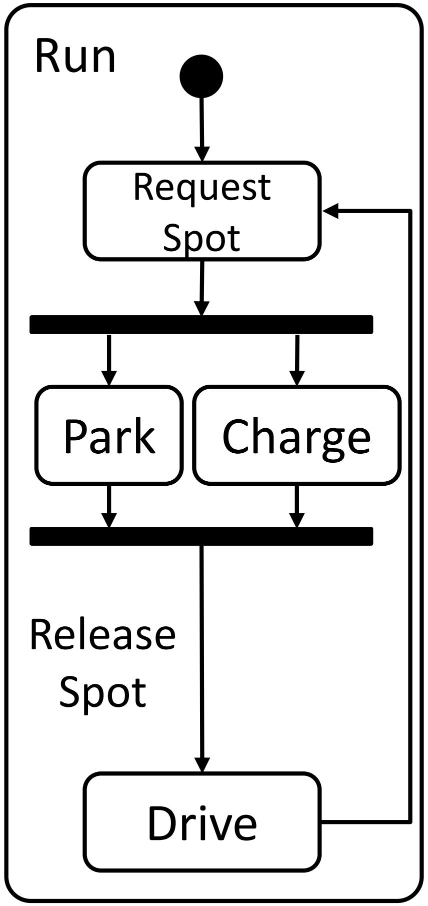
```
import simpy # simpy3.py

def car_park(env, name, duration):
  yield env.timeout(duration)

def car_drive(env, name, duration):
  yield env.timeout(duration)

def car_run(env, name, t_park, t_drive):
  while True:
    yield env.process(car_park(env, name, t_park))
    yield env.process(car_drive(env, name, t_drive))

env = simpy.Environment()
env.process(car_run(env, 'A', 5.0, 2.0))
env.process(car_run(env, 'B', 3.0, 3.0))
env.run(until=15.0)
```

Shared Resources




```
import simpy # simpy4.py

def car_park(env, name, duration):
  yield env.timeout(duration)

def car_drive(env, name, duration):
  yield env.timeout(duration)

def car_run(env, name, lot, t_park, t_drive):
  while True:
    with lot.request() as request:
      yield request
      yield env.process(car_park(env, name, t_park))
    yield env.process(car_drive(env, name, t_drive))

env = simpy.Environment()
single_lot = simpy.Resource(env, capacity=1)
env.process(car_run(env, 'A', single_lot, 5.0, 2.0))
env.process(car_run(env, 'B', single_lot, 3.0, 3.0))
env.run(until=15.0)
```

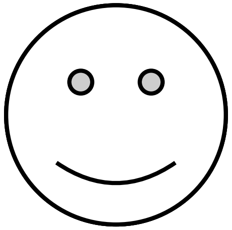
Process Interaction




```
import simpy # simpy5.py

def car_charge(env, name, duration):
  yield env.timeout(duration)

def car_park(env, name, duration):
  yield env.timeout(duration)

def car_drive(env, name, duration):
  yield env.timeout(duration)

def car_run(env, name, lot, t_park, t_drive, t_charge):
  while True:
    with lot.request() as request:
      yield request
      parking = env.process(car_park(env, name, t_park))
      charging = env.process(car_charge(env, name, t_charge))
      yield parking & charging
    yield env.process(car_drive(env, name, t_drive))

env = simpy.Environment()
single_lot = simpy.Resource(env, capacity=1)
env.process(car_run(env, 'A', single_lot, 5.0, 2.0, 6.0))
env.process(car_run(env, 'B', single_lot, 3.0, 3.0, 2.0))
env.run(until=15.0)
```

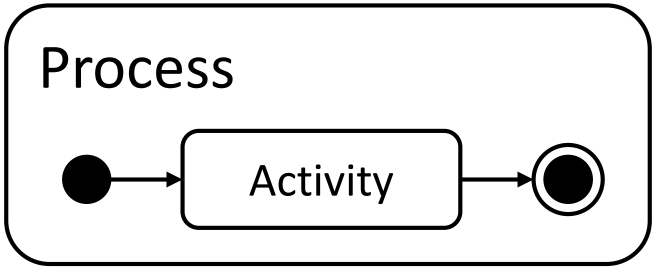
SimPy Example: Queuing Model

Queuing System

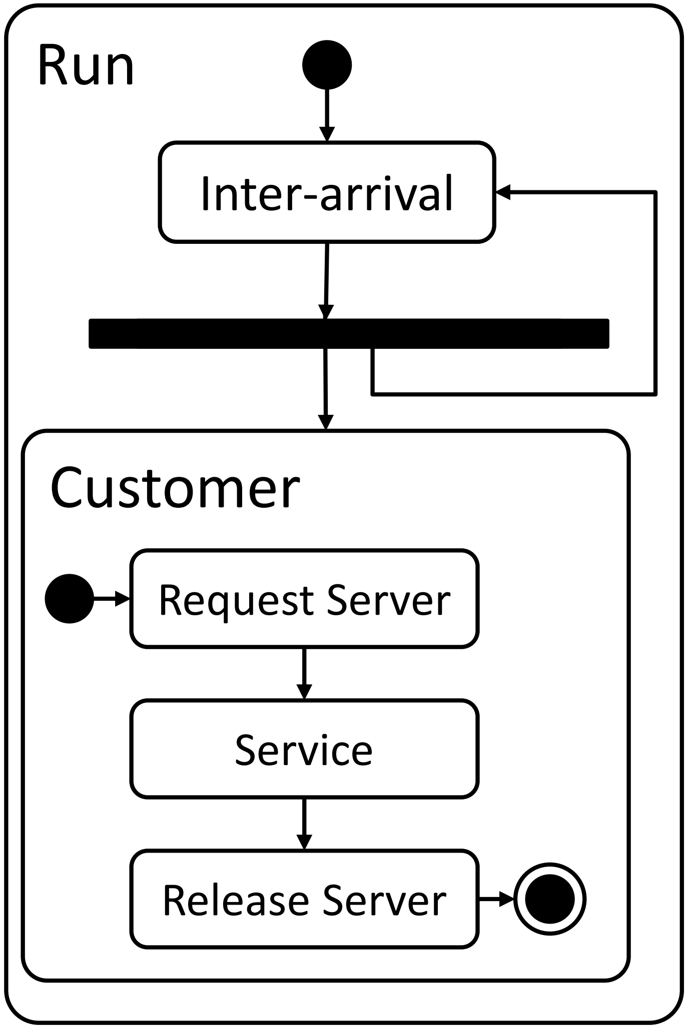


Queuing System Model

- Queuing system specification:
  - $n = 1$ server, $C = \infty$ queue capacity
  - Customer inter-arrival period $x \sim \text{exponential}(\lambda = 3)$
  - Service time $y \sim \text{exponential}(\mu = 4)$
- **What are the processes and activities?**



Queuing Model Activity Diagram




```
import simpy # QueuingSystem.py
import numpy as np

def cafe_run(env, servers, _lambda, _mu):
  while True:
    yield env.timeout(
        np.random.exponential(1./_lambda))
    env.process(customer(env, servers, _mu))

def customer(env, servers, _mu):
  with servers.request() as request:
    yield request
    yield env.timeout(
      np.random.expeonential(1./_mu))

env = simpy.Environment()
servers = simpy.Resource(env, capacity=1)
env.process(cafe_run(env, servers, 3.0, 4.0))
env.run(until=10)
```

Recording Observations

```
def cafe_run(env, servers, _lambda, _mu):
  while True:
    yield env.timeout(
        np.random.exponential(1./_lambda))
    env.process(customer(env, servers, _mu))

wait_q = []
wait_t = []

def customer(env, servers, _mu):
  with servers.request() as request:
    t_arrive = env.now
    yield request
    t_served = env.now
    wait_q.append(t_served - t_arrive)
    yield env.timeout(
      np.random.exponential(1./_mu))
    t_depart = env.now
    wait_t.append(t_depart - t_arrive)
```

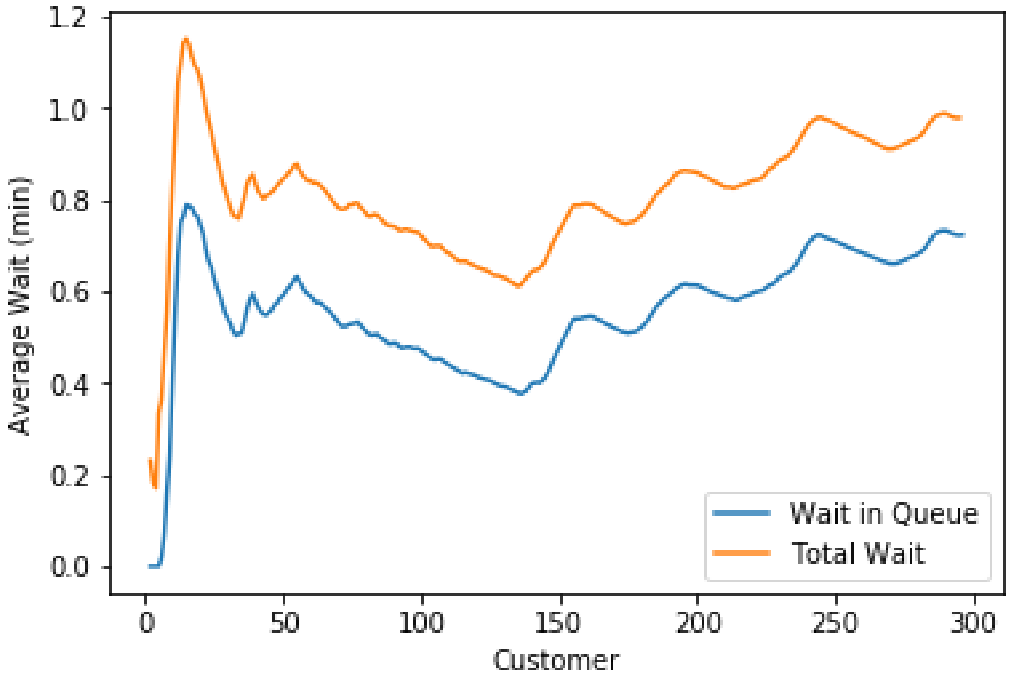
- Append to data lists during processes
- Example shown relies on local variables
- Better to pass in as generator arguments

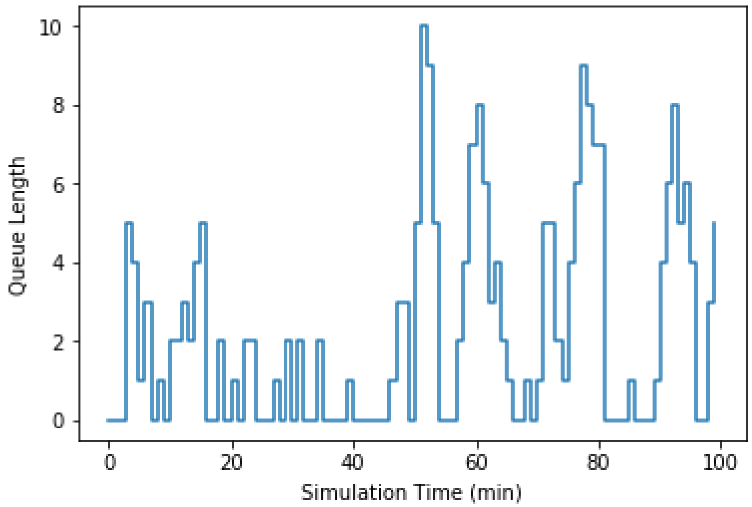
Process Monitoring

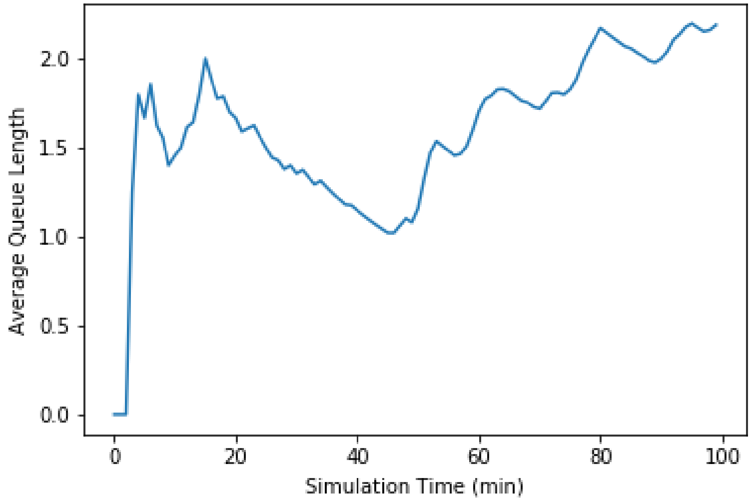
```
obs_time = []
queue_length = []

def observe(env, servers):
  while True:
    obs_time.append(env.now)
    queue_length.append(len(servers.queue))
    yield env.timeout(1.0)

env = simpy.Environment()
servers = simpy.Resource(env, capacity=1)
env.process(cafe_run(env, servers, 3.0, 4.0))
env.process(observe(env, servers))
env.run(until=10)
```

- Create simple process to take observations at fixed times
- Example shown relies on local variables
- Better to pass in as generator arguments

Queuing System Results

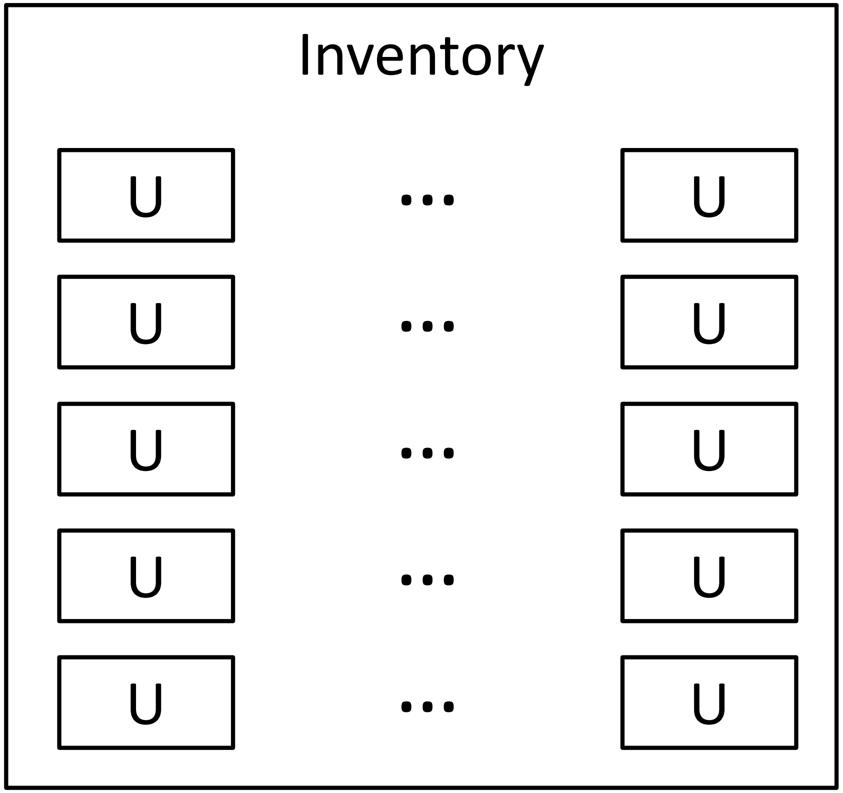


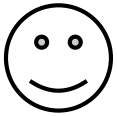


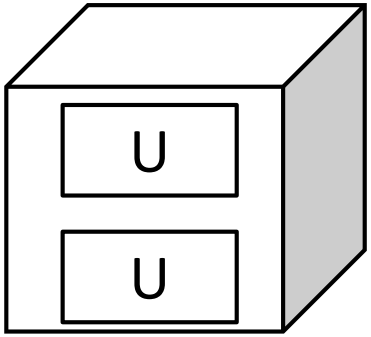


SimPy Example: Inventory Model

Inventory System



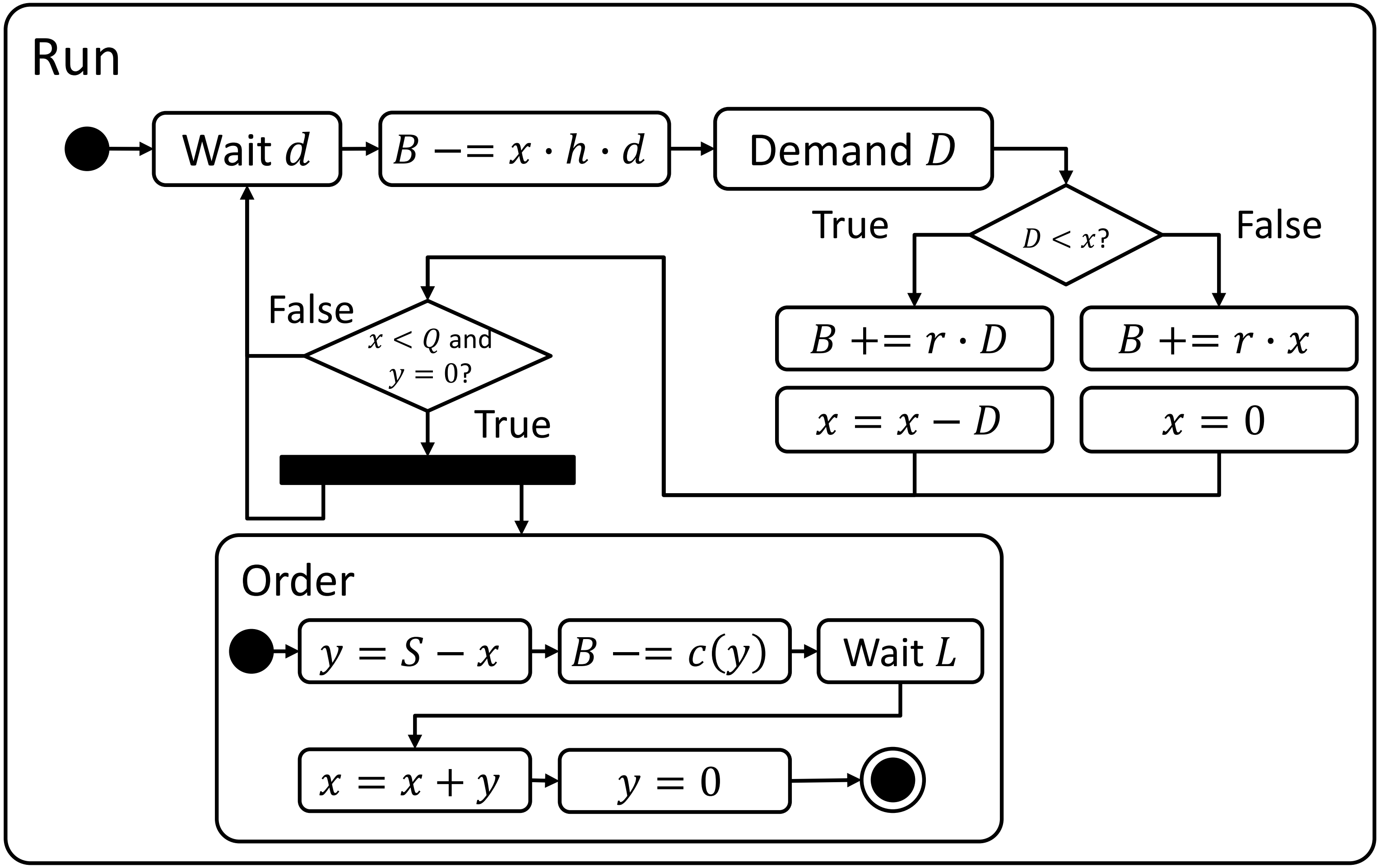




Inventory System Model

- Inventory system specification:
  - Sell products for $r = 100$ (can only sell those in stock)
  - Customer inter-arrival period $d \sim \text{exponential}(\lambda = 5)$
  - Each customer demands $D \sim \text{uniform}(1,4)$ products
  - Order policy (Q, S): if inventory is $x < Q$, order $y = S - x$
  - Costs $c(y) = 50y$ to order y units
  - Delay of $L = 2$ days until delivery
  - Holding cost of $h = 2$ per item per day
- **What are the processes and activities?**

Inventory Model Activity Diagram

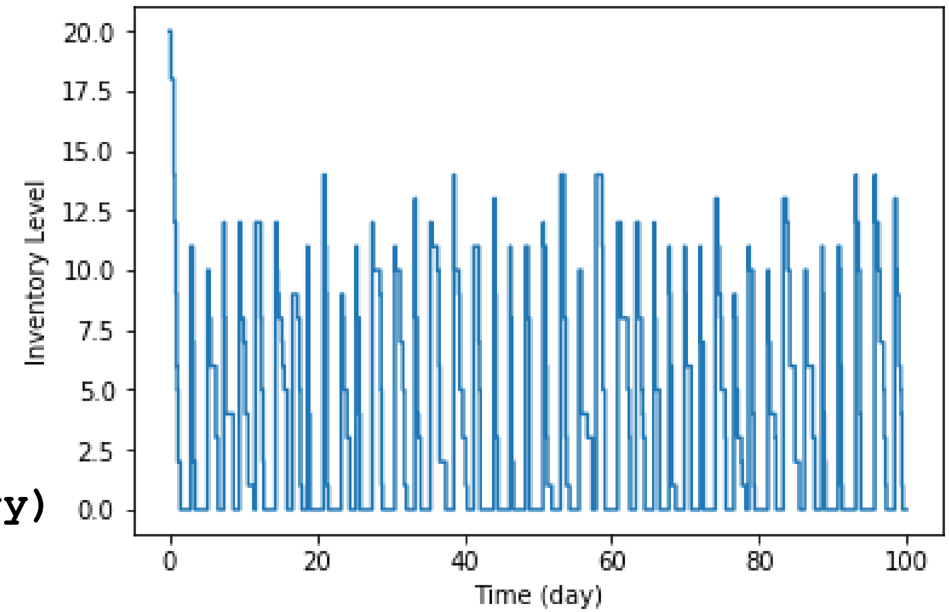


Inventory Model Python

```
import simpy # InventoryModel.py
import numpy as np

def warehouse_run(env, order_threshold, order_up_to):
  global inventory, balance, num_ordered
  inventory = order_target
  balance = 0
  num_ordered = 0
  while True:
      interarrival = np.random.exponential(1/5.0)
      yield env.timeout(interarrival)
      balance -= 2.00*inventory*interarrival
      demand = np.random.randint(1, 4+1)
      if inventory > demand: num_sold = demand
      else: num_sold = inventory
      balance += 100.00*num_sold
      inventory -= num_sold
      if inventory < order_threshold and num_ordered == 0:
        env.process(handle_order(env, order_up_to - inventory))
```

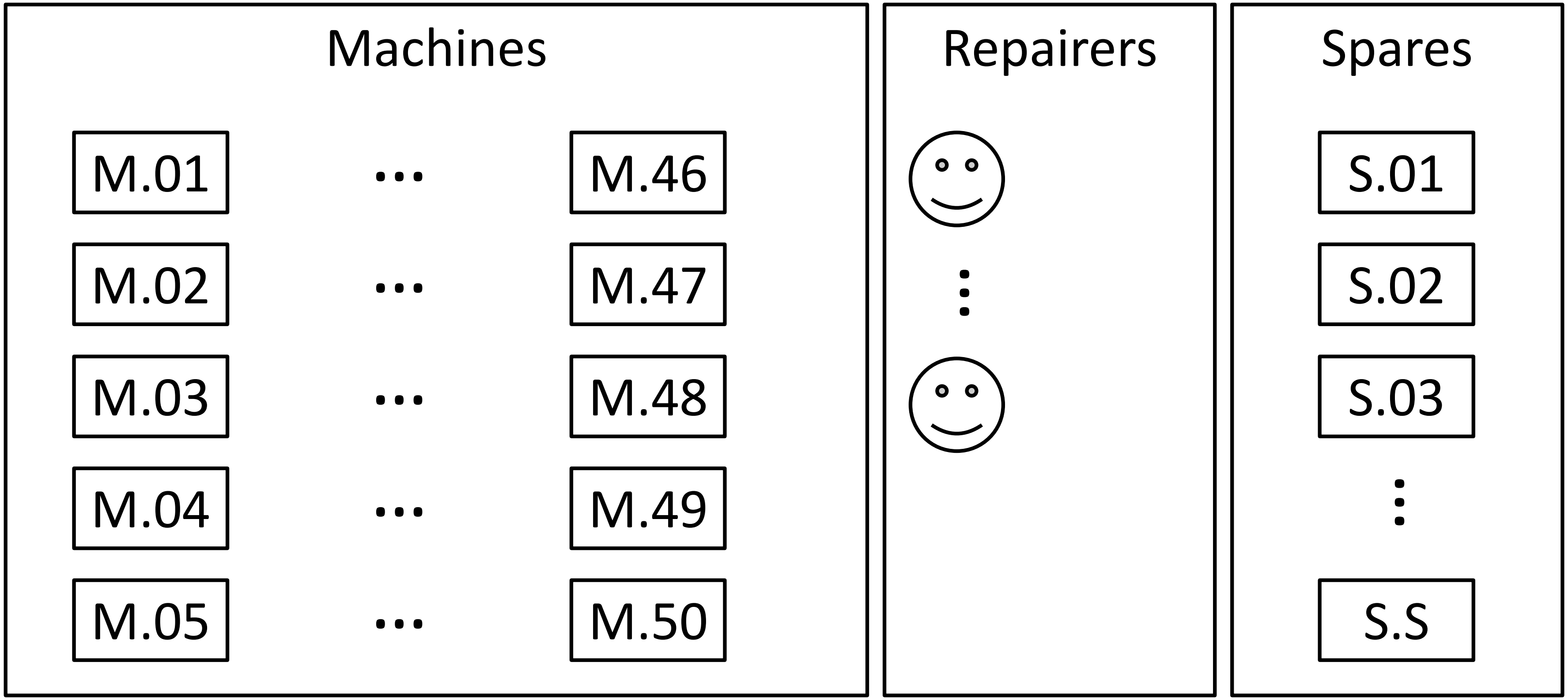
Inventory Model Python (cont.)

```
def handle_order(env, quantity):
  global inventory, balance, num_ordered
  num_ordered = quantity
  balance -= 50.00*quantity
  yield env.timeout(2.0)
  inventory += quantity
  num_ordered = 0

obs_time = []
inventory_level = []

def observe(env):
  while True:
    obs_time.append(env.now)
    inventory_level.append(inventory)
    yield env.timeout(0.1)

env = simpy.Environment()
env.process(warehouse_run(env, 10, 20))
env.process(observe(env))
env.run(until=5.0)
```



SimPy Example: Factory Model

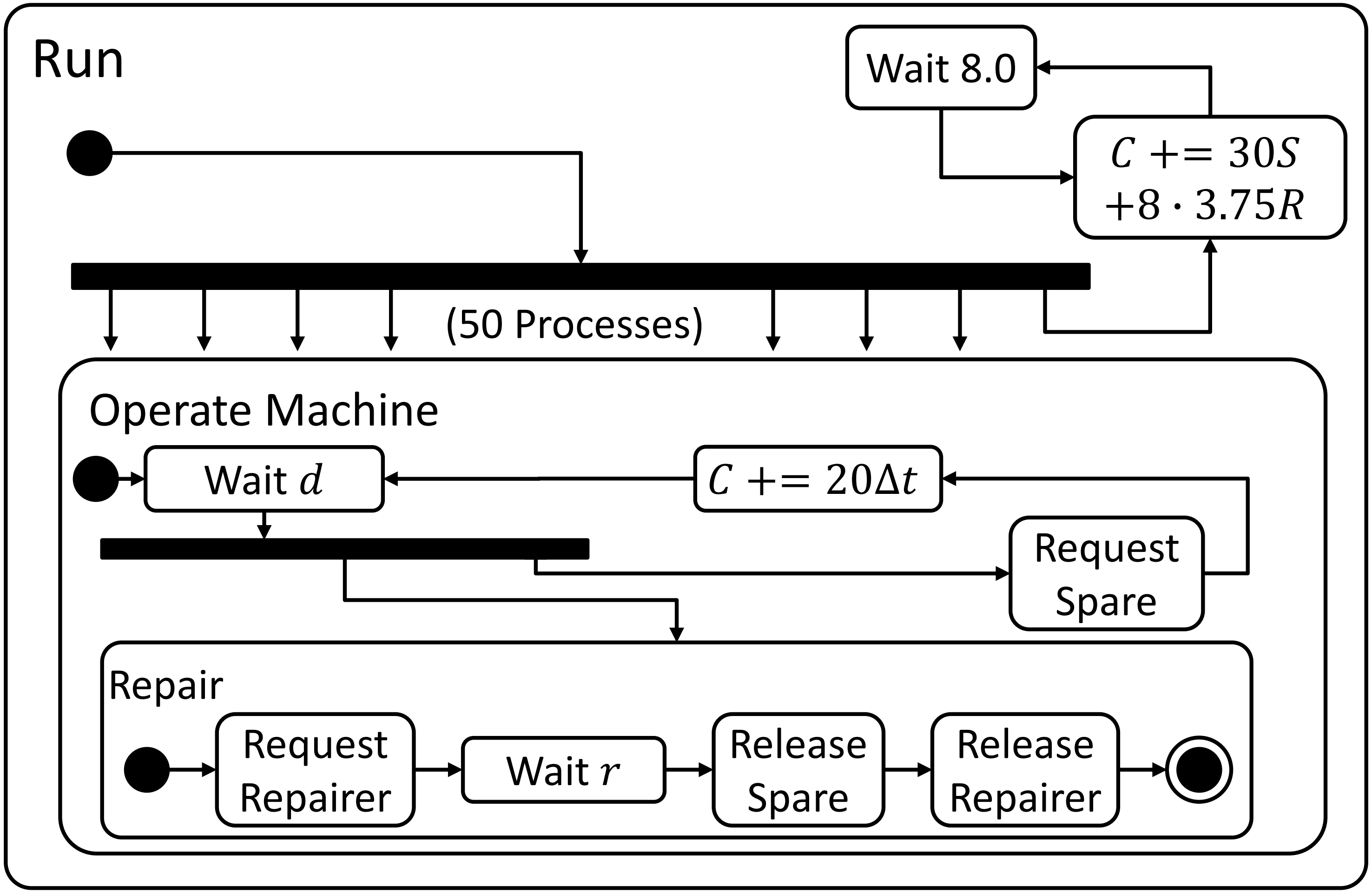
Factory System



Factory System Model

- Factory system specification:
  - $n = 50$ machines working 8 hours/day, 5 days/week
  - Each fails randomly after $d \sim \mathrm{uniform}(132{,}182)$ hours
  - Immediately replace with spare when available
  - Repairer fixes failed machine in $r \sim \mathrm{uniform}(4{,}10)$ hours, returns to spares
  - Hire R repairers at $3.75/hour
  - Purchase S spares at $30/day
  - Costs $20/hour/machine if out-of-service (no spares)
- **What are the processes and activities?**

Factory Model Activity Diagram



Factory Model Python

```
import simpy # FactorySystem.py
import numpy as np

def factory_run(env, repairers, spares):
  global cost
  cost = 0
  for i in range(50):
    env.process(operate_machine(env, i+1, repairers, spares))
  while True:
    cost += 3.75*8*repairers.capacity + 30*spares.capacity
    yield env.timeout(8.0)

def operate_machine(env, machine, repairers, spares):
  global cost
  while True:
    yield env.timeout(np.random.uniform(132, 182))
    time_broken = env.now
    env.process(repair_machine(env, repairers))
    yield spares.get(1)
    cost += 20*(env.now - time_broken)
```

Factory Model Python (cont.)

```
def repair_machine(env, repairers):
  with repairers.request() as request:
    yield request
    yield env.timeout(np.random.uniform(4,10))
    yield spares.put(1)

obs_time = []
obs_spares = []
def observer(env, spares):
  while True:
    obs_time.append(env.now)
    obs_spares.append(spares.level)
    yield env.timeout(1.0)

env = simpy.Environment
repairers = simpy.Resource(env, capacity=3)
spares = simpy.Container(env, init=20, capacity=20)
env.process(factory_run(env, repairers, spares))
env.process(observe(env, spares))
env.run(until=8*5*52)
```