Discrete Event Simulation

with SimPy

Tran, Van Hoai (hoai@hcmut.edu.vn)
Le, Hong Trang (lhtrang@hcmut.edu.vn)

Faculty of Computer Science & Engineering
HCMC University of Technology

2017-2018/Semester 1

Outline

Outline

What is Discrete-Event Simulation (DES) ?

Discrete-Event Simulation is

- Discrete (in state)
- Dynamic (in time)
- Stochastic

DES mostly applied to queueing systems (but not limited to)

- Factory workflow
- Freeway traffic simulation
- Network traffic simulation
- ...

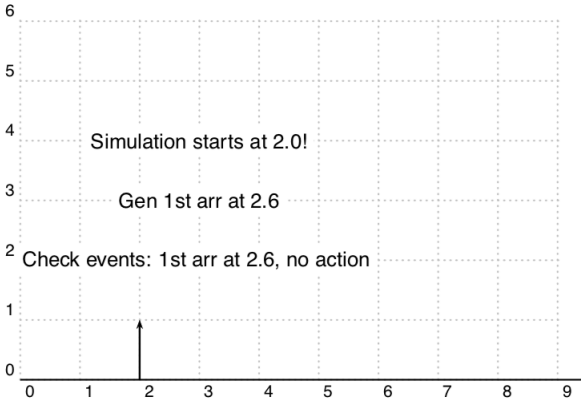
DES categories

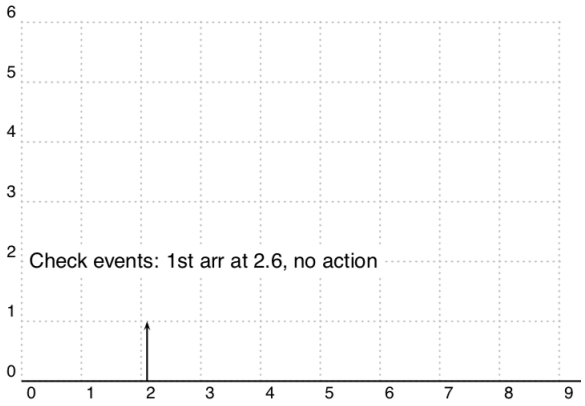
- Activity-oriented
  - fixed increment of time
  - time-consuming
- Event-oriented
  - on each event, generate next event and put into event queue and sort
  - simulation time advances to next closed event
  - faster than activity-oriented
- Process-oriented
  - abstract one object into a process
  - easier to maintain as object-oriented approach

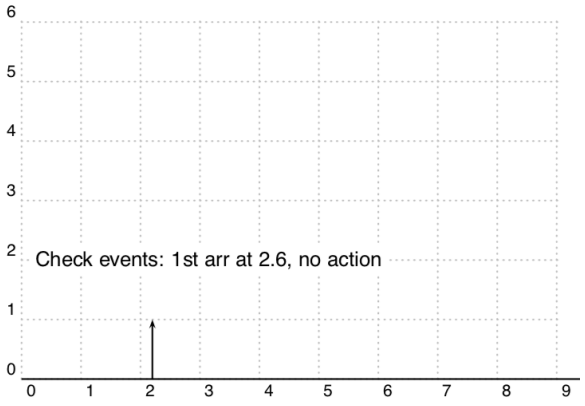
DES categories

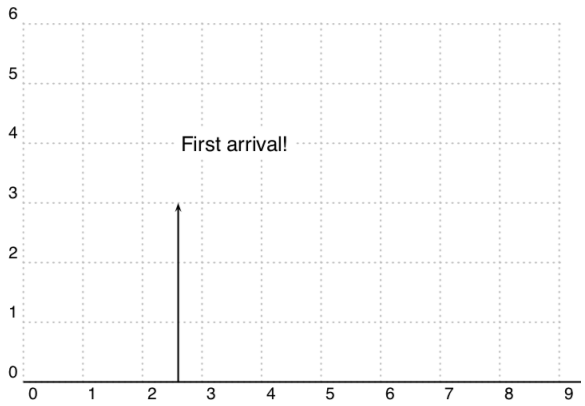
- Activity-oriented
  - fixed increment of time
  - time-consuming
- Event-oriented
  - on each event, generate next event and put into event queue and sort
  - simulation time advances to next closed event
  - faster than activity-oriented
- Process-oriented
  - abstract one object into a process
  - easier to maintain as object-oriented approach
  - **SimPy is here**

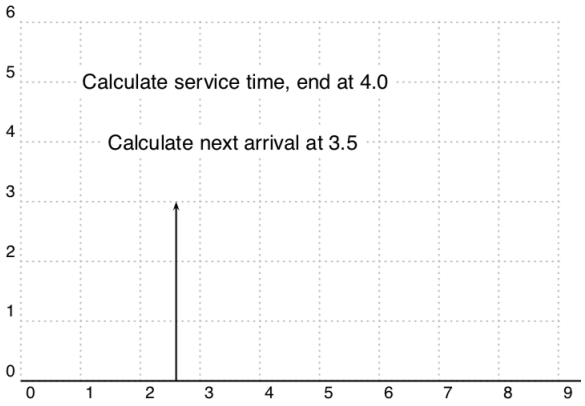
Activity-oriented

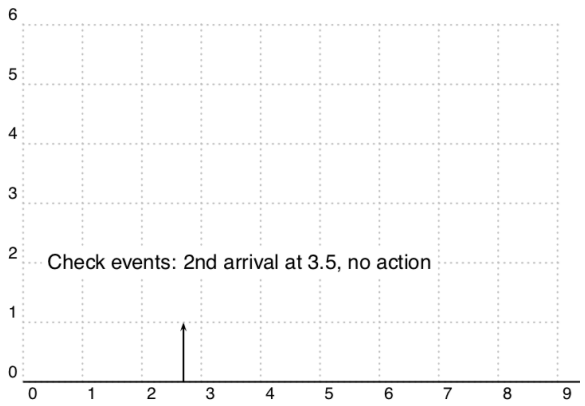




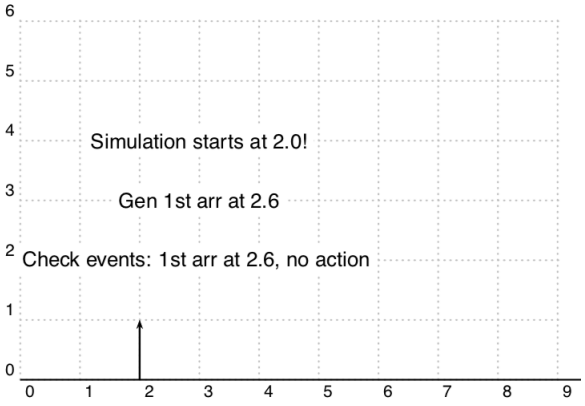


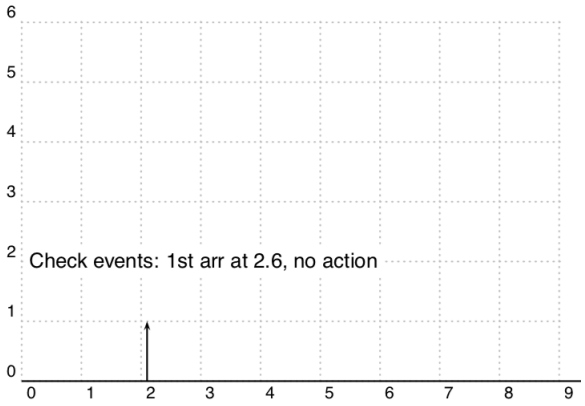


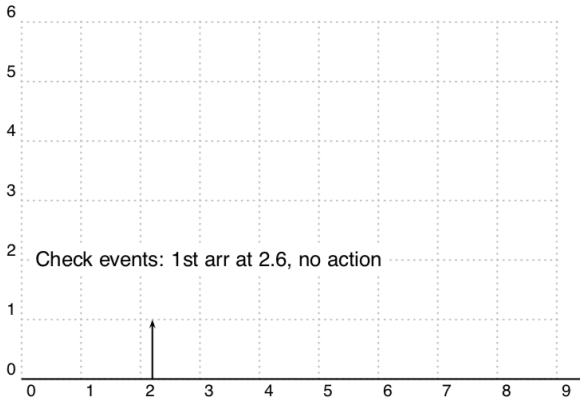


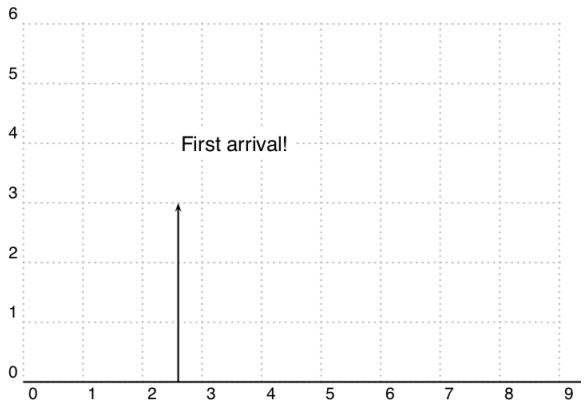


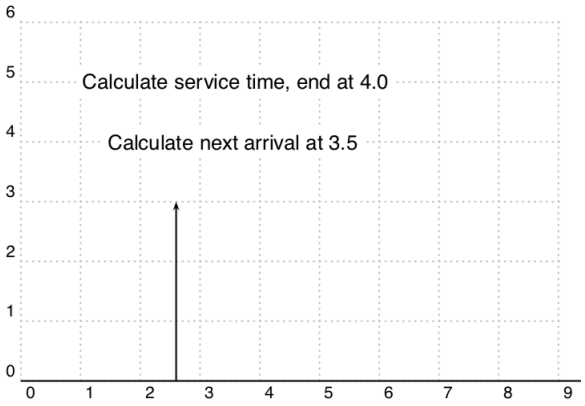
Activity-oriented

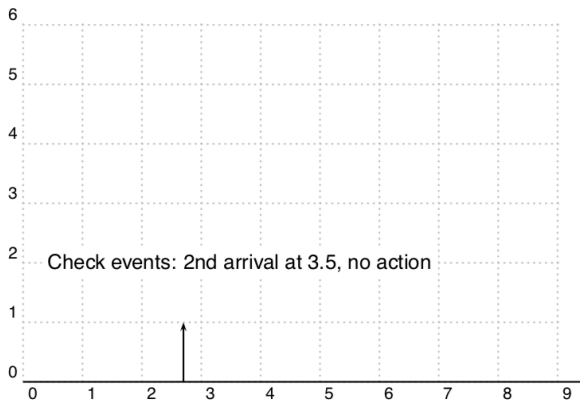






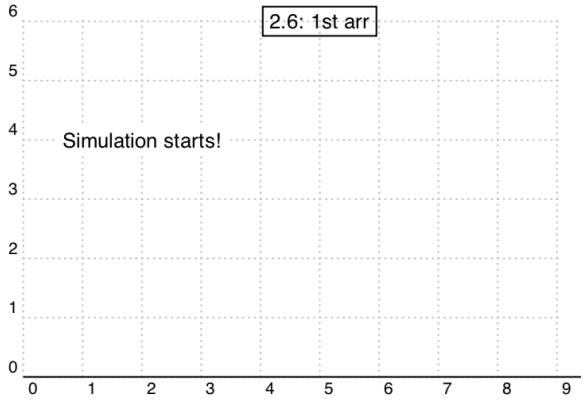


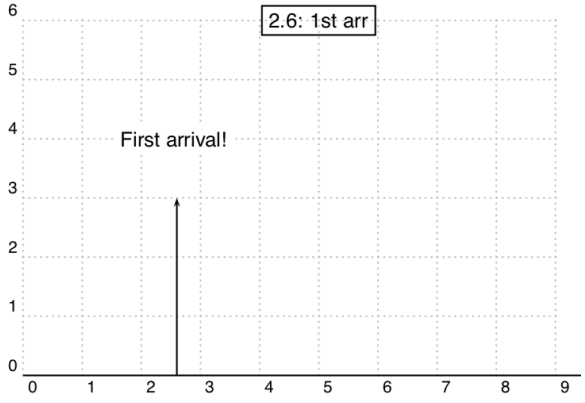


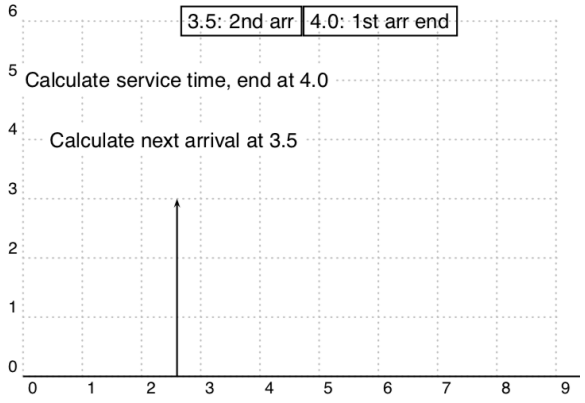


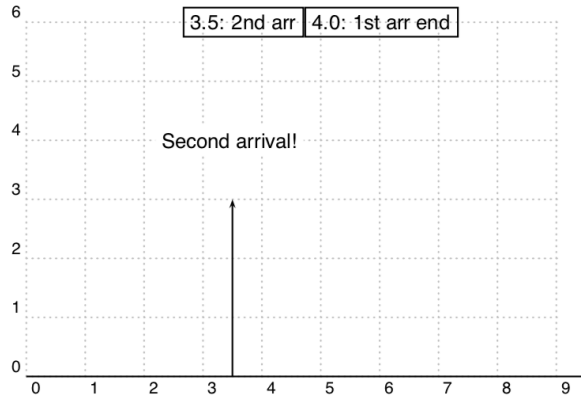
Much time for redundant checks of the unchanged state

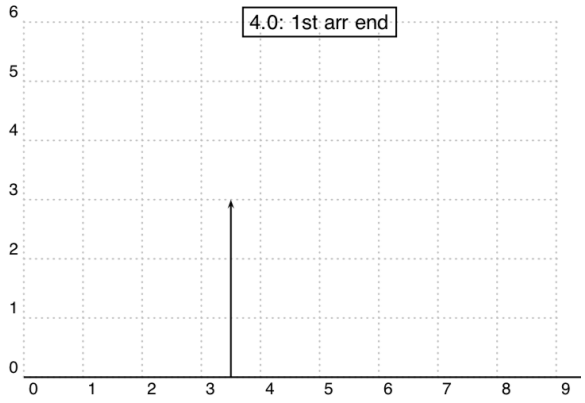
Event-oriented

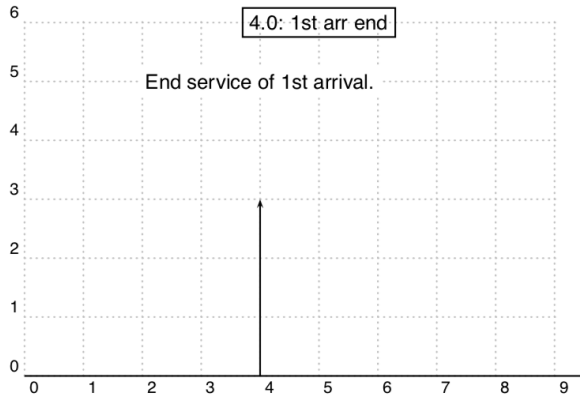




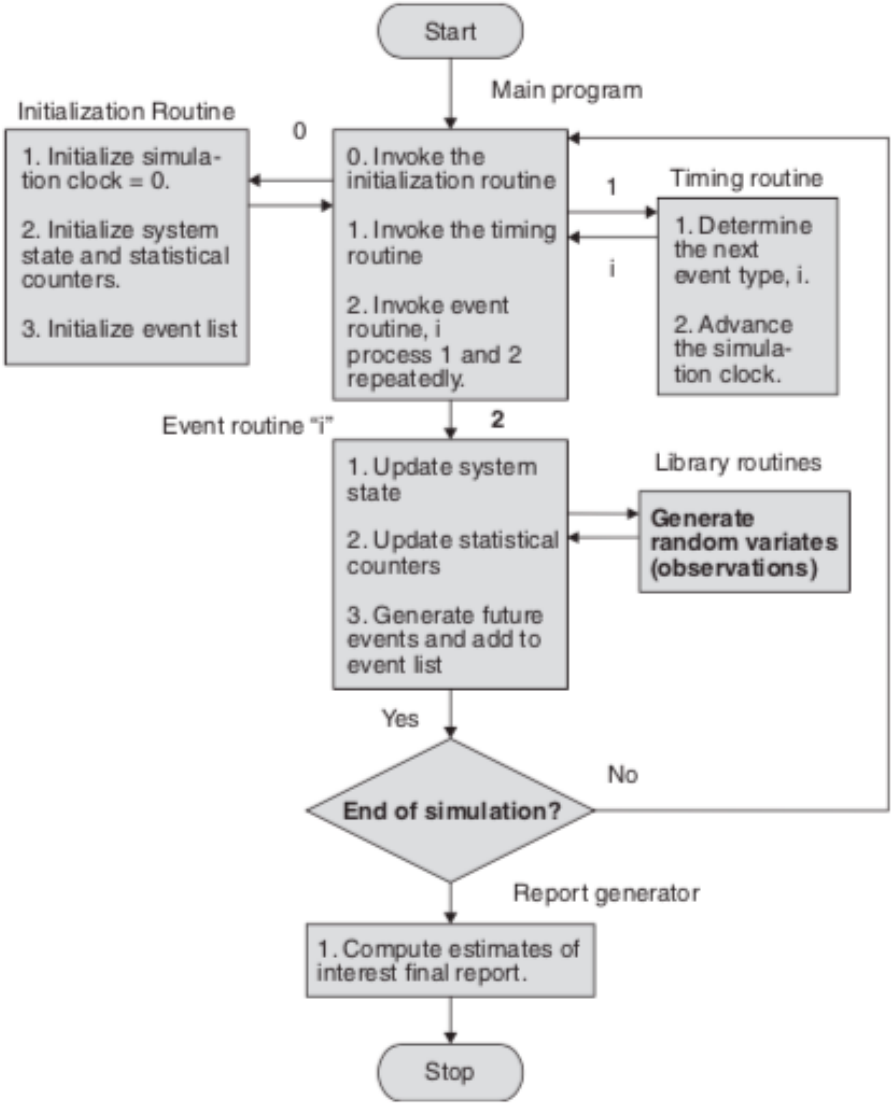








Event-oriented DES architecture



Process-oriented (1)

- Based on event-oriented
- Designed into separate processes

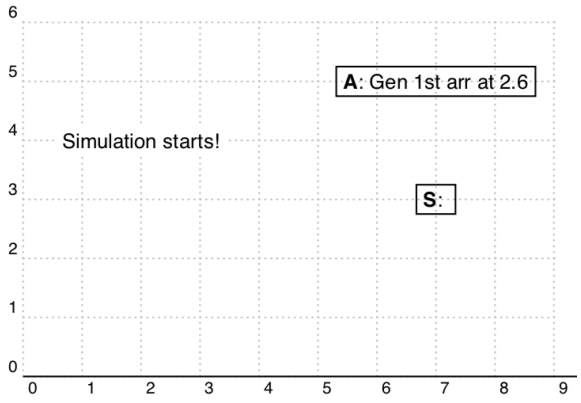
Process-oriented (1)

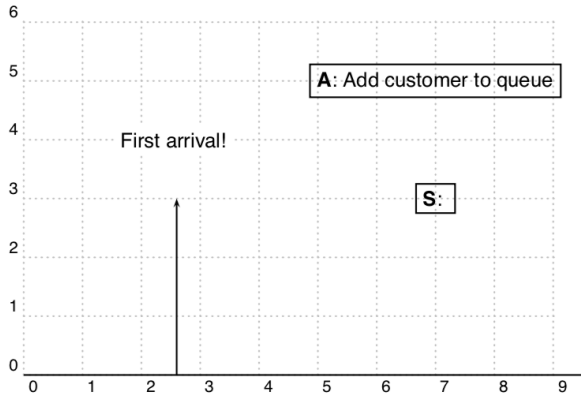
- Based on event-oriented
- Designed into separate processes

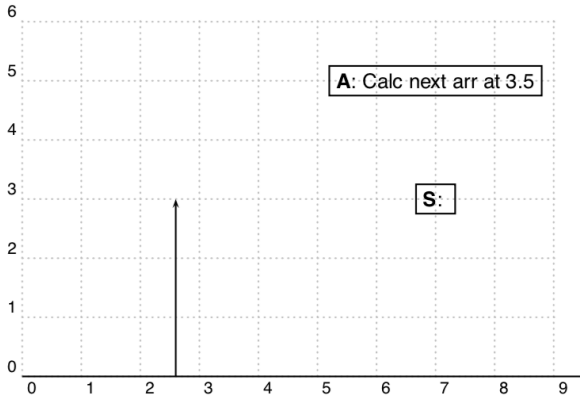
A single queue in process-oriented DES

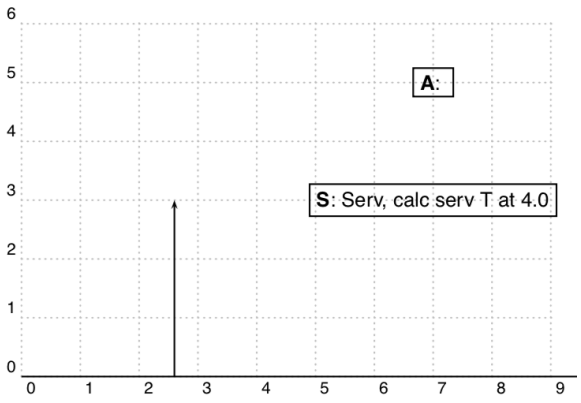
- **Arrival process**: an infinite loop of the following
  - Calculate next arrival time
  - Sleep until next arrival
  - Add new job to queue
- **Service process**: an infinite loop of the following
  - Sleep until waken by new jobs
  - Serve the queued jobs on waken until there is no job in queue

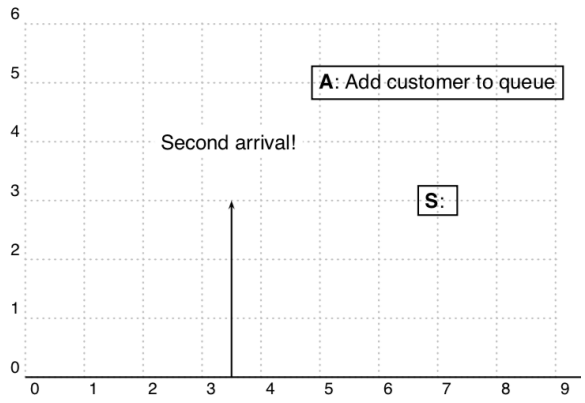
Process-oriented (2)

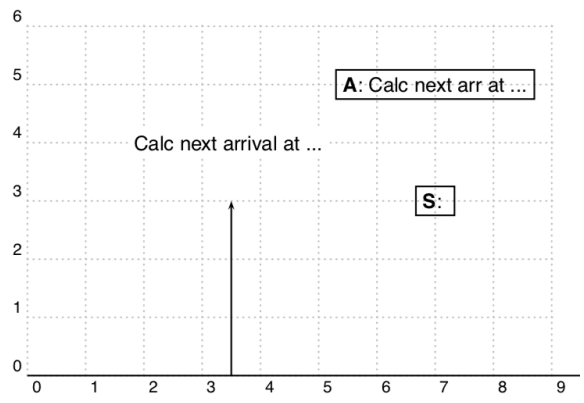


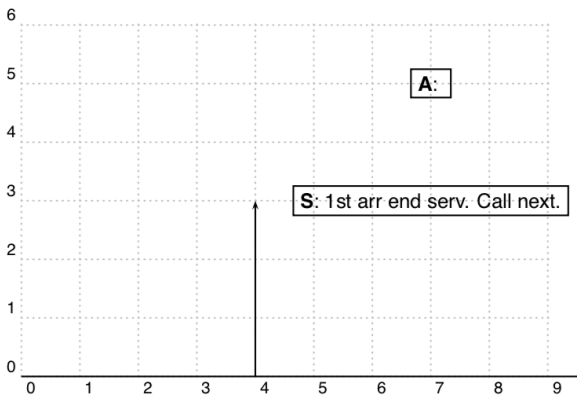


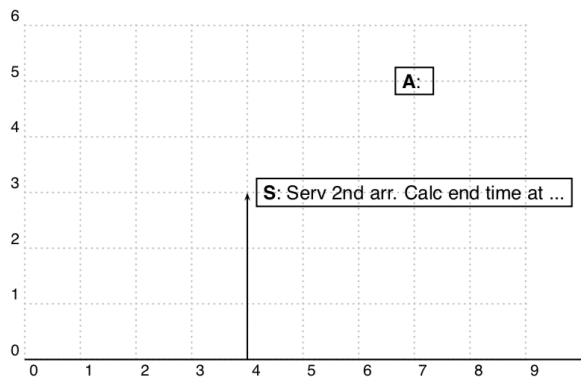












DES implementation

- Using C++
- Using generalized simulation library (language)
  - SIMULA programming language
  - C++SIM or JavaSIM
  - SimEvents in Simulink/MATLAB
- Using special purpose simulation packages
  - ns-3 for network simulation
  - OPNET for network simulation

DES implementation

- Using C++
- Using generalized simulation library (language)
  - SIMULA programming language
  - C++SIM or JavaSIM
  - SimEvents in Simulink/MATLAB
  - **SimPy is here**
- Using special purpose simulation packages
  - ns-3 for network simulation
  - OPNET for network simulation

Outline

“Hello world” with Python

- Python is a scripting language (as MATLAB, R, ...)
- Python is free with a huge community
- Python is easy to write

```
>>> print( 'Hello world' )
```

"Hello world" with Python

- Python is a scripting language (as MATLAB, R, ...)
- Python is free with a huge community
- Python is easy to write

```
>>> print( 'Hello world' )
```

- Python is used for web development (server side)
- Python is used for mathematics
- Python can be treated in a **procedural way**, an **object-orientated way** or a **functional way**

Python syntax

Indentation

```
if 5 > 2:
  print( "Five > Two" )
```

Variable

```
y = 'John'
print( y )
```

Numeric types

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Operators

Arithmetic, Assignment, Comparison, Logical, Identity, Membership, Bitwise

Collection datatypes

List, Tuple, Set, Dictionary

Conditional clauses

```
if a > b:
  print( 'a > b' )
else:
  print( 'a <= b' )
```

Loop

```
i = 1
while i < 6:
  print( i )
  i += 1
```

Function

```
def myfunc():
  print( 'My function' )
```

Class

```
class Person:
  count = 0
  def __init__(self,name:
    self.name = name
P1 = Person('Hoai')
print( P1.name )
```

What is SimPy ?
(based on slides of Stefan Scherfke)

- Environment
- Process
- Event
- Resource

```
>>> import simpy
>>>
>>> def clock(env, name, tick):
...     while True:
...         print(name, env.now)
...         yield env.timeout(tick)
...
>>> env = simpy.Environment()
>>>
>>> env.process(clock(env, 'fast', 0.5))
<Process(clock) object at 0x...>
>>> env.process(clock(env, 'slow', 1))
<Process(clock) object at 0x...>
>>>
>>> env.run(until=2)
fast 0
slow 0
fast 0.5
slow 1
fast 1.0
fast 1.5
```

Start simpy environment

Register processes with their events

Run simulation until time of 2

Timeout and processes
(based on slides of Stefan Scherfke)

Timeout is event


```
def speaker(env, start):
    until_start = start - env.now
    yield env.timeout(until_start)

    yield env.timeout(30)
```

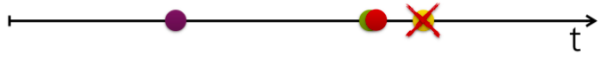
Process is event, too


```
def speaker(env):
    yield env.timeout(30)
    return 'handout'


def moderator(env):
    for i in range(3):
        val = yield env.process(speaker(env))
        print(val)
```

Synchronization
(based on slides of Stefan Scherfke)

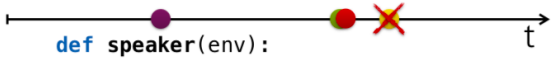
Asynchronous interrupt




```
def speaker(env):
    try:
        yield env.timeout(randint(25, 35))
    except simpy.Interrupt as interrupt:
        print(interrupt.cause)


def moderator(env):
    for i in range(3):
        speaker_proc = env.process(speaker(env))
        yield env.timeout(30)
        speaker_proc.interrupt('No time left')
```

Condition event




```
def speaker(env):
    try:
        yield env.timeout(randint(25, 35))
    except simpy.Interrupt as interrupt:
        print(interrupt.cause)


def moderator(env):
    for i in range(3):
        speaker_proc = env.process(speaker(env))
        results = yield speaker_proc | env.timeout(30)

        if speaker_proc not in results:
            speaker_proc.interrupt('No time left')
```

Outline

FIFO queue

FIFO queue

- **Job**: an object of a class

FIFO queue

- **Job**: an object of a class
- **Queue of jobs**: list of objects

FIFO queue

- **Job**: an object of a class
- **Queue of jobs**: list of objects
- **Arrival process**: an infinite loop of the following
  - Calculate next arrival time
  - Sleep until next arrival
  - Add new job to queue

FIFO queue

- **Job**: an object of a class
- **Queue of jobs**: list of objects
- **Arrival process**: an infinite loop of the following
  - Calculate next arrival time
  - Sleep until next arrival
  - Add new job to queue
- **Service process**: an infinite loop of the following
  - Sleep until waken by new jobs
  - Serve the queued jobs on waken until there is no job in queue

FIFO queue
Job

```
class Job:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration
```

More attributes could be added in order to collect statistical data, such as

- arrival time
- start time of service

FIFO queue

A queue with single server

```
class Server:
    def __init__(self, env):
        self.Jobs = list(())
        env.process( self.serve(env) )

    def serve(self, env):
        while True:
            if len( self.Jobs ) == 0 :
                self.serversleeping = env.process( self.waiting( env ))
                yield self.serversleeping
            else:
                j = self.Jobs.pop( 0 )
                yield env.timeout( j.duration )

    def waiting(self, env):
        try:
            print( 'Server is idle at %d' % env.now )
            yield env.timeout(1000)
        except simpy.Interrupt as i:
            print('A new job comes. Server waken up and works now at %d'
                  % env.now )
```

FIFO queue
Job generator

```
class JobGenerator:
  jcnt = 0

  def __init__(self, env, server):
    self.server = server
    env.process( self.jobgen(env) )

  def jobgen(self, env):
    while True:
      job_interarrival = random.randint(1,5)
      yield env.timeout( job_interarrival )

      job_duration = random.randint(2,5)
      self.jcnt += 1
      self.server.Jobs.append( Job('Job %s' %self.jcnt, job_duration) )
      print( 'job %d: t = %d, l = %d' %( self.jcnt, env.now, job_duration ))

      if not self.server.serversleeping.triggered:
        self.server.serversleeping.interrupt( 'Wake up, please.' )
```

```
env = simpy.Environment()
MyServer = Server( env )
MyJobGenerator = JobGenerator( env, MyServer )
env.run( until = 20 )
```

And results

```
Server is idle at 0
job 1: t = 4, l = 2
A new job comes. Server back to work at 4 by 'Wake up, please.'
Server is idle at 6
job 2: t = 9, l = 5
A new job comes. Server back to work at 9 by 'Wake up, please.'
job 3: t = 13, l = 5
job 4: t = 14, l = 3
job 5: t = 19, l = 4
```

M/M/1 queue

Simulation parameters

- Inter-arrival rate: $\lambda = 0.08$
- Service rate: $\mu = 0.1$
- Simulation time: 500

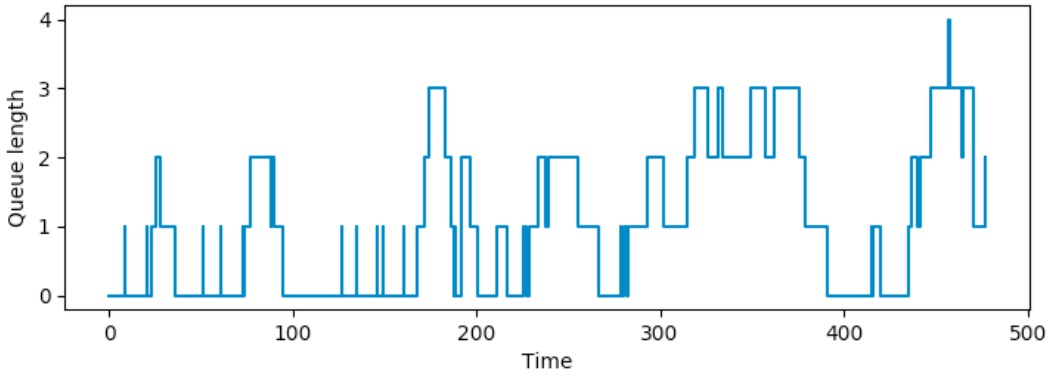
M/M/1 queue

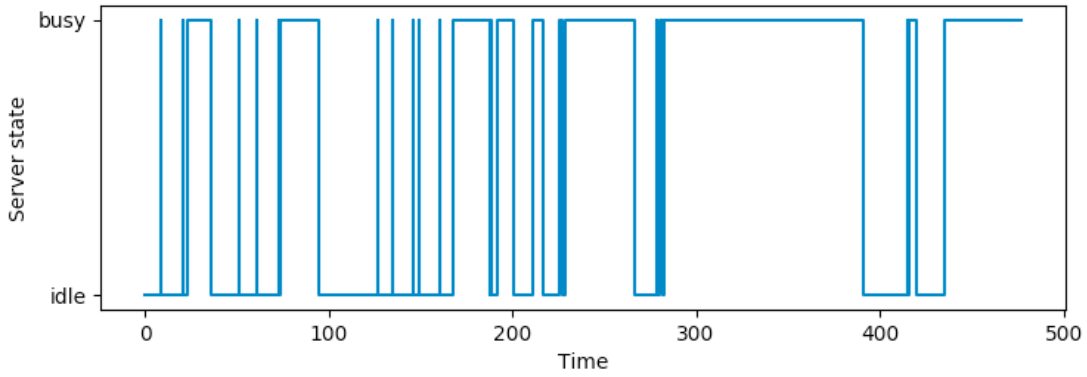
Simulation parameters

- Inter-arrival rate: $\lambda = 0.08$
- Service rate: $\mu = 0.1$
- Simulation time: 500

Performance

- Analytical modeling: $\overline{W} = 40.0$ (for validation)
- Simulation modeling: $\overline{W} = 10.86$ (???)





M/M/1 queue
What else should be doned ?

- Simulation functions

- Model enhancement

M/M/1 queue
What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)

- Model enhancement

M/M/1 queue
What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)

- Model enhancement

M/M/1 queue
What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)
  - Simulation logging
- Model enhancement

M/M/1 queue
What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)
  - Simulation logging
  - Simulation plotting
- Model enhancement

M/M/1 queue
What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)
  - Simulation logging
  - Simulation plotting
- Model enhancement
  - Different types of single queue in Kendall's notation M/M/c/K/N/D
    - Queueing strategies (FIFO, SJF, priority, round-robin,...)
    - Different types of workload

M/M/1 queue
What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)
  - Simulation logging
  - Simulation plotting
- Model enhancement
  - Different types of single queue in Kendall's notation M/M/c/K/N/D
    - Queueing strategies (FIFO, SJF, priority, round-robin,...)
    - Different types of workload
  - Queueing networks