EEE413 Data Communication and Communication Networks

Dr Kyeong Soo (Joseph) Kim

Department of Electrical and Electronic Engineering
Xi'an Jiaotong-Liverpool University

16 September 2019

Outline

Next . . .

Basic Queueing Theory
- **Little's Theorem**
- **Poisson Process**
- **M/M/1 Queueing System**
- **Other Markov Systems**

Network Simulation
- **Ergodicity**
- **Time-Stepped vs. Discrete-Event Simulation**
- **Discrete-Event Simulation**
- **SimPy: Python-Based Discrete-Event Simulation Framework**

Basic Queueing Theory

Example: Queueing in An Airport

Example: Queueing in An Airport

Next . . .

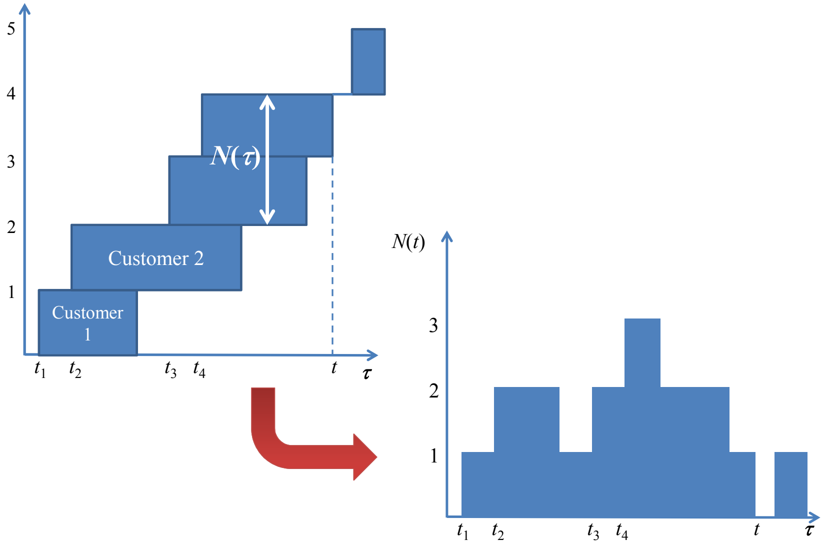
Little's Theorem: Definitions - State Variables

- $N(t)$: Number of customers in the system at time t.
- $\alpha(t)$: Number of customers arrived during the interval $[0,\ t]$.
- T_i: Time spent in the system by the ith customer.

Little's Theorem: Definitions - Steady States

- Steady-state number of customers: $N \triangleq \lim_{t\to\infty} N_t$,
  where $N_t = \frac{1}{t}\int_0^t N(\tau)d\tau$.
- Steady-state arrival rate: $\lambda \triangleq \lim_{t\to\infty} \lambda_t$,
  where $\lambda_t = \frac{\alpha(t)}{t}$.
- Steady-state customer delay: $T \triangleq \lim_{t\to\infty} T_t$,
  where $T_t = \frac{\sum_{i=1}^{\alpha(t)} T_i}{\alpha(t)}$.

Little’s Theorem: Graphical Proof I



Little's Theorem: Graphical Proof II

- Because the shaded area (up to time t) can be expressed as either $\int_0^t N(\tau)d\tau$ (in the right figure) or $\sum_{i=1}^{\alpha(t)} T_i$ (in the left figure), we can prove the Little's theorem as follows:
  - Equate both sides: $\int_0^t N(\tau)d\tau = \sum_{i=1}^{\alpha(t)} T_i$.
  - Divide them by t: $\frac{\int_0^t N(\tau)d\tau}{t} = \frac{\sum_{i=1}^{\alpha(t)} T_i}{t}$.
  - Take the limit of t: $\lim_{t\to\infty} \frac{\int_0^t N(\tau)d\tau}{t} = \lim_{t\to\infty} \frac{\sum_{i=1}^{\alpha(t)} T_i}{t}$.
  - Note that $\lim_{t\to\infty} \frac{\int_0^t N(\tau)d\tau}{t} = N$ and $\lim_{t\to\infty} \frac{\sum_{i=1}^{\alpha(t)} T_i}{t} = \lim_{t\to\infty} \left[\left(\frac{\sum_{i=1}^{\alpha(t)} T_i}{\alpha(t)}\right) \times \left(\frac{\alpha(t)}{t}\right)\right] = T\lambda$.
  - Finally, we have $N = \lambda T$.

Little's Theorem: Applications

- If N_Q is the average number of customers waiting in the queue (but not under service) and W is the average customer waiting time in the queue, then

$$N_Q = \lambda W.$$

- If $\overline{X}$ is the average service time and ρ is the average number of customers under service (also called "utilization factor" in the communication system), then

$$\rho = \lambda \overline{X}.$$

Next . . .

Poisson Process: Introduction

- In queueing theory, the arrival process of customers often can be described by a Poisson process.
- In communication networks, the *customers* may be calls or packets. Poisson process is a viable model when the calls or packets originate from a large population of independent users.
- Poisson process is mathematically described by a counting process $A(t)$, i.e., for $t > 0$ and $s > t$,
  - $A(0) = 0$
  - $A(t)$ = the # of arrivals during the interval $[0, t]$.
  - $A(t - s) \triangleq A(t) - A(s)$ = the # of arrivals during the interval $(s, t]$.

Poisson Process: Definition 1 - Pure Birth Process

A ***Poisson process*** *with rate* λ can be defined in the three different but *equivalent* ways.

- In an infinitesimal time interval δ, there may occur only one arrival with the probability $\lambda\delta$, i.e., for every $t \geq 0$ and $\delta \geq 0$,

$$
\begin{aligned}
P\{A(t+\delta) - A(t) = 0\} &= 1 - \lambda\delta + o(\delta), \\
P\{A(t+\delta) - A(t) = 1\} &= \lambda\delta + o(\delta), \\
P\{A(t+\delta) - A(t) \geq 2\} &= o(\delta),
\end{aligned}
$$

  where $o(\delta)$ is a function such that $\lim_{\delta \to 0} \frac{o(\delta)}{\delta} = 0$.
- The arrival is independent of arrivals outside the interval.

Poisson Process: Definition 2 - Process of Independent Increments

- The number of arrivals in any finite interval of length t follows a Poisson distribution with the average of λt, i.e.,

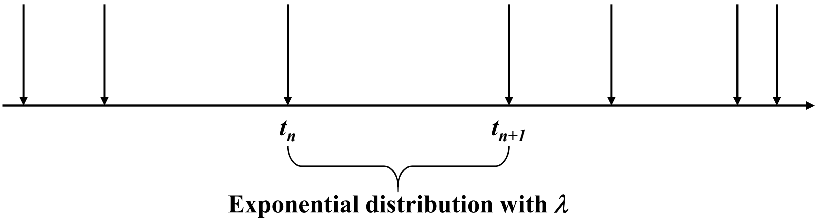
$$P\{A(t) = n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!},\ n = 0, 1, \ldots$$

- The number of arrivals in disjoint intervals are independent.

Poisson Process: Definition 3 - Exponential Interarrival Time

- Interarrival times are independent and exponentially distributed with parameter λ; if t_n denotes the time of the nth arrival, the intervals $\tau_n = t_{n+1} - t_n$ have the probability distribution

$$P\{\tau_n \leq s\} = 1 - e^{-\lambda s},\ s \geq 0.$$



Poisson Process: Equivalence of Definitions - $1 \to 2$ I

- From the definition 1, we have

$$P\left\{A(t+\delta) - A(t)\right\} = \lambda\delta + o(\delta).$$

- Consider the probability generating function $G_t(z)$:

$$\begin{aligned}
G_t(z) &= \mathrm{E}\left[z^{A(0,t)}\right], \\
G_{t+\delta}(z) &= \mathrm{E}\left[z^{A(0,t+\delta)}\right] = \mathrm{E}\left[z^{A(0,t)+A(t,t+\delta)}\right] \\
&= \mathrm{E}\left[z^{A(0,t)}\right]\mathrm{E}\left[z^{A(t,t+\delta)}\right] = G_t(z)\left(1 - \lambda\delta + \lambda\delta z\right) \\
&= G_t(z) - \lambda\delta(1-z)G_t(z) \\
\frac{G_{t+\delta}(z) - G_t(z)}{\delta} &= \lambda(z-1)G_t(z).
\end{aligned}$$

Poisson Process: Equivalence of Definitions - $1 \rightarrow 2$ II

- Taking limit (i.e., $\delta \rightarrow 0$) provides the following differential equation:

$$\frac{d}{dt} G_t(z) = \lambda(z-1) G_t(z),$$
$$G_t(z) = e^{(z-1)\lambda t},$$

  where the last equation is the probability generating function of the Poisson distribution.

Poisson Process: Equivalence of Definitions - $2 \rightarrow 3$

- Note that the the following two events are identical:
  - An interarrival time is greater than t.
  - There is no arrival during the time interval of length t.
- If the interarrival time is for nth and $(n+1)$th arrivals (i.e., $\tau_n = t_{n+1} - t_n$), we have

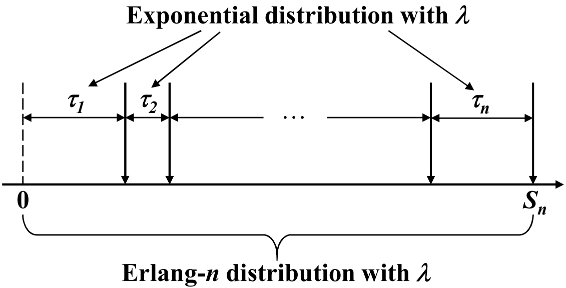
$$P\{\tau_n > t\} = P\{A(t_n + t, t_n) = 0\} = e^{-\lambda t},$$

which shows the interarrival time follows the exponential distribution.

Poisson Process: Equivalence of Definitions - $3 \to 1$ I

- If the interarrival times follow the exponential distribution with parameter λ, the nth arrival time $S_n \triangleq \sum_{i=1}^{n} \tau_i$ follows Erlang-n distribution[1], i.e.,

$$P\{S_n \le t\} = 1 - \sum_{i=0}^{n-1} \frac{e^{-\lambda t}(\lambda t)^i}{i!}.$$



Poisson Process: Equivalence of Definitions - $3 \rightarrow 1$ II

- For infinitesimal time δ, therefore, we have the following probabilities:

$$P\{A(t+\delta) - A(t) = 0\} = P\{\tau_n > \delta\} = e^{-\lambda\delta} = 1 - \lambda\delta + o(\delta),$$
$$P\{A(t+\delta) - A(t) \geq 2\} = P\{S_2 \leq \delta\} = 1 - \sum_{i=0}^{1} \frac{e^{-\lambda\delta}(\lambda\delta)^i}{i!} = o(\delta),$$

  where we apply Maclaurin series (i.e., Taylor series centered at zero) expansion of the exponential function.

  - $P\{A(t+\delta) - A(t) = 1\}$ can be derived from the above and the probability normalization condition.

[1] http://www.math.unl.edu/~scohn1/428s05/queue3.pdf

Poisson Process: Equivalence of Definitions (Optional) - 2 $\to$ 1

If the counting process $A(t)$ follows the Poisson distribution, i.e., $P\{A(t)=n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!}$, then

$$P\{A(\delta) = 0\} = e^{-\lambda\delta} = 1 - \lambda\delta + o(\delta),$$
$$P\{A(\delta) = 1\} = \frac{\lambda\delta}{1!}e^{-\lambda\delta} = \lambda\delta + o(\delta).$$

Next . . .

Queueing Theory Nomenclature

$$M/M/m/m$$

Nature of arrival process

- *M: Memoryless*
- *G: General*
- *D: Deterministic*

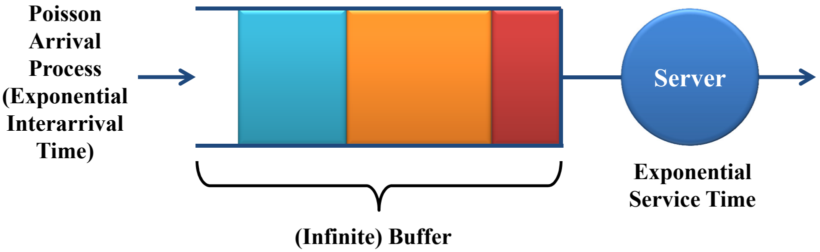
Service time distribution

- *M: Memoryless*
- *G: General*
- *D: Deterministic*

Number of servers

Maximum number of customers in the system

M/M/1 Queueing System: Overview



- Note that successive interarrival and service times are assumed to be ***statistically independent of each other***.

M/M/1 Queueing System: Arrival Statistics

- The customer arrivals follow a Poisson process with rate λ.
  - The interarrival times, therefore, follow an exponential distribution with parameter λ, i.e.,

    $$P\{\tau_n \leq t\} = 1 - e^{-\lambda t},\ t \geq 0,$$

    where τ_n is the interarrival time between the nth and the $(n+1)$th customers.
- The interarrival times τ_n are mutually independent and also independent of all interarrival times.
- The parameter λ is called the ***arrival rate*** and represents the average rate at which the customer arrives at the system.

M/M/1 Queueing System: Service Statistics

- The customer service times follow an exponential distribution with parameter μ, i.e.,

$$P\{s_n \leq t\} = 1 - e^{-\mu t},\ t \geq 0,$$

  where s_n is the service time of the nth customer.
- The service times s_n are mutually independent and also independent of all service times.
- The parameter μ is called the ***service rate*** and represents the average rate at which the server operates when busy.

M/M/1 Queueing System: Memoryless Property

- The exponential distribution has a memoryless property which is expressed as follows (for the interarrival and service times τ_n and s_n, respectively):

$$P\{\tau_n > r + t | \tau_n > t\} = P\{\tau_n > r\}, \quad \text{for } r, t \geq 0,$$
$$P\{s_n > r + t | s_n > t\} = P\{s_n > r\}, \quad \text{for } r, t \geq 0.$$

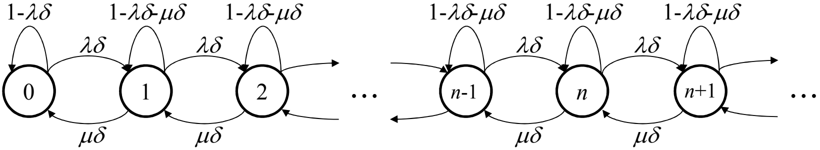
M/M/1 Queueing System: Markov Chain Formulation I

- The memoryless property allows the use of ***Markov chains*** (here we use a discrete-time version for simplicity):
  - We focus only on the number of customers in the system at discrete times $0, \delta, 2\delta, \ldots$ (δ is a small positive number).
  - The set $\{N_k | k=0, 1, \ldots\}$ forms a discrete-time Markov chain with transition probabilities $P_{i,j}=P\{N_k=j | N_{k-1}=i\}$, where N_k is defined as $N(k\delta)$.

M/M/1 Queueing System: Markov Chain Formulation II

- Now the transition probabilities are given by

$$
\begin{aligned}
P_{0,0} &= 1 - \lambda\delta + o(\delta), && \\
P_{i,i} &= 1 - \lambda\delta - \mu\delta + o(\delta), && i \geq 1, \\
P_{i,i+1} &= \lambda\delta + o(\delta), && i \geq 0, \\
P_{i,i-1} &= \mu\delta + o(\delta), && i \geq 1, \\
P_{i,j} &= o(\delta), && i \text{ and } j \neq i, i+1, i-1.
\end{aligned}
$$



M/M/1 Queueing System: Stationary Distribution I

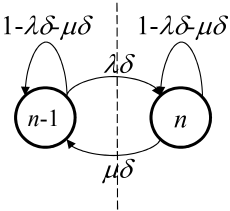
- Define steady-state probabilities

$$p_n = \lim_{k\to\infty} P\{N_k = n\} = \lim_{t\to\infty} P\{N(t) = n\}.$$

- A local balance (of flows) equation is given by

$$p_{n-1}\lambda\delta + o(\delta) = p_n\mu\delta + o(\delta).$$

- By dividing both sides by δ and taking the limit $\delta \to 0$, we obtain $p_{n-1}\lambda = p_n\mu$.



M/M/1 Queueing System: Stationary Distribution II

- Now we can obtain $p_n = \rho^n (1-\rho)$ for $n \geq 0$ from the following two equations (if $\rho < 1$):

$$p_n = \rho^n p_0, \qquad n = 1, 2, \ldots,$$

$$1 = \sum_{n=0}^{\infty} p_n = \sum_{n=0}^{\infty} \rho^n p_0 = \frac{p_0}{1-\rho},$$

where $\rho = \frac{\lambda}{\mu}$.

M/M/1 Queueing System: Main Results

- Average number of customers in the system

$$\overline{N} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

- Average delay

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\mu - \lambda}$$

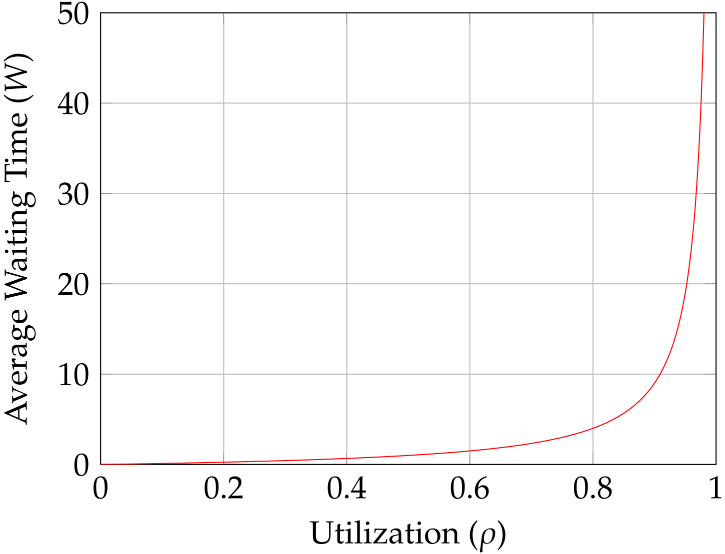
- Average waiting time in the queue

$$\overline{W} = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}$$

- Average number of customers in the queue

$$\overline{N_Q} = \lambda \overline{W} = \frac{\rho^2}{1-\rho}$$

M/M/1 Queueing System: Average Delay



Next . . .

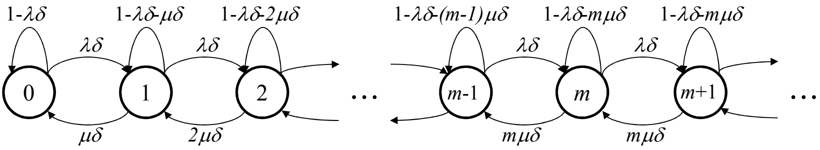
Basic Queueing Theory

- Little's Theorem
- Poisson Process
- M/M/1 Queueing System
- **Other Markov Systems**

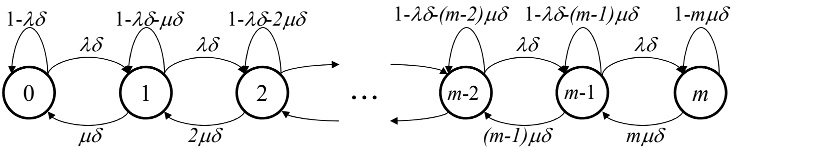
Network Simulation

- Ergodicity
- Time-Stepped vs. Discrete-Event Simulation
- Discrete-Event Simulation
- SimPy: Python-Based Discrete-Event Simulation
- Framework

Other Markov Systems



M/M/m – The System with *m* Servers



M/M/m/m – The *m*-Server Loss System

M/M/m Queueing System: Stationary Distribution I

- From the local balance equations, we get the following (if $\rho \triangleq \lambda/(m\mu) < 1$):

$$\begin{cases} p_n = \dfrac{\lambda}{n\mu} p_{n-1}, & \text{for } 1 \le n < m, \\ p_n = \dfrac{\lambda}{m\mu} p_{n-1}, & \text{for } n \ge m. \end{cases}$$

- Now we can obtain p_n as follows:

$$\begin{cases} p_n = \dfrac{(m\rho)^n}{n!} p_0, & \text{for } 0 \le n < m, \\ p_n = \dfrac{m^m \rho^n}{m!} p_0, & \text{for } n \ge m. \end{cases}$$

M/M/m Queueing System: Stationary Distribution II

- Because $\sum_{n=0}^{\infty} p_n = 1$, we obtain

$$p_0 = \frac{1}{\sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho}}$$

- We first obtain p_{m+}, which will be used in deriving main results:

$$p_{m+} \triangleq P\{N \geq m\} = \sum_{n=m}^{\infty} p_n = \frac{(m\rho)^m}{m!(1-\rho)} p_0$$

  - p_{m+} is the probability that all servers are busy and the customers has to wait in the queue (also called ***Erlang C*** formula).

M/M/m Queueing System: Main Results

- Average number of customers in the system

$$\overline{N} = m\rho + \frac{\rho}{1-\rho} p_{m+}$$

- Average number of customers in the queue

$$\overline{N_Q} = \frac{\rho}{1-\rho} p_{m+}$$

- Average waiting time in the queue

$$\overline{W} = \frac{\overline{N_Q}}{\lambda} = \frac{\rho}{\lambda(1-\rho)} p_{m+}$$

- Average delay

$$\overline{T} = \overline{W} + \frac{1}{\mu} = \frac{\rho}{\lambda(1-\rho)} p_{m+} + \frac{1}{\mu}$$

Next . . .

Network Simulation

Next . . .

Ergodicity: Introduction

In econometrics and signal processing, a stochastic process is said to be *ergodic* if its statistical properties can be deduced from a single, sufficiently long, random sample of the process.[2]

- The reasoning is that any collection of random samples from a process must represent the average statistical properties of the entire process.
- In other words, regardless of what the individual samples are, a birds-eye view of the collection of samples must represent the whole process.
- Conversely, a process that is not ergodic is a process that changes erratically at an inconsistent rate.

[2] The contents on ergodicity are from Wikipedia.

Ergodicity: Mean-Ergodic Process

The process $X(t)$ is said to be *mean-ergodic* or *mean-square ergodic in the first moment* if the time average estimate

$$\hat{\mu}_X = \frac{1}{T}\int_0^T X(t)\,dt$$

converges in squared mean to the ensemble average μ_X as $T \to \infty$.

Ergodicity: Autocovariance-Ergodic Process

Likewise, the process is said to be *autocovariance-ergodic* or *mean-square ergodic in the second moment* if the time average estimate

$$\hat{r}_X(\tau) = \frac{1}{T}\int_0^T [X(t+\tau) - \mu_X][X(t) - \mu_X]\, dt$$

converges in squared mean to the ensemble average $r_X(\tau)$, as $T \to \infty$.

- A process which is ergodic in the mean and autocovariance is sometimes called *ergodic in the wide sense*.

Next . . .

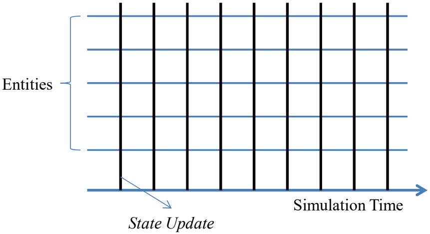
Time-Stepped vs. Discrete-Event Simulation

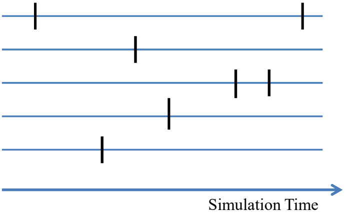
Time-Stepped

- System states are globally updated at periodic time instances with increment *dt*.
- Entities exchanges state updates via *messages*.

Discrete-Event

- System states are locally and asynchronously updated at different times.
- Entities exchange *events* for state updates.





Next . . .

Discrete-Event Simulation: Components

- **Event list**: List of future events sorted by their event times
- **Clock**: Unlike real-time simulation, time hops from one event to another.
- **Random-number generators**: For generating random variables of various kinds based on PRNGs.
- **Statistics**: Mean waiting time, queue length, ...
- **Ending condition**: Rum time (e.g., 3 hours) or the number events (e.g., after 10,000 packets generated).

Discrete-Event Simulation: Program Structure

- Start
  - Set *Ending Condition* to FALSE.
  - Initialize clock (typically set to zero) and variables for system states and statistics.
  - Schedule *the first event* to trigger the simulation.
- While *Ending Condition* is **FALSE**, do the following:
  - Fetch *the next event* from the event list.
  - Set the clock to the time of the fetched event.
  - Process the event.
    - Update variables for system states and statistics.
    - Schedule/cancel future events.
- End
  - Generate statistical report

Next . . .

SimPy: Overview

SimPy[3] is a process-based discrete-event simulation framework based on standard Python[4].

- *Processes* are defined by Python *generator functions*[5] and may, for example, be used to model active components like packets and customers.
- *Shared resources* are also provided to model limited capacity congestion points like servers and checkout counters.

---

[3] https://simpy.readthedocs.io/

[4] https://www.python.org/

[5] https://wiki.python.org/moin/Generators

SimPy: An Illustrating Example

Simulates two clocks ticking in different time intervals.

```
>>> 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
```

Generator I

```
>>> itr = [x*x for x in range(10)]
>>> itr
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> len(itr)
10
>>> gen = (x*x for x in range(10))
>>> gen
<generator object <genexpr> at 0x000001532A8B9BF8>
>>> len(gen)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'generator' has no len()
```

The generator expression for `gen` above is equivalent to the following generator implementation:

```
def squared_series():
    for x in range(10):
        yield x*x
```

Generator II

Can we make an infinite sequence? Yes, sort of ...

```python
def pi_series():
    sum = 0
    i = 1.0; j = 1
    while(1):
        sum = sum + j/i
        yield 4*sum
        i = i + 2; j = j * -1
```

This generator generates a series converges to $\pi/4$.

Generator III

How can we use a generator?

- By next() call.

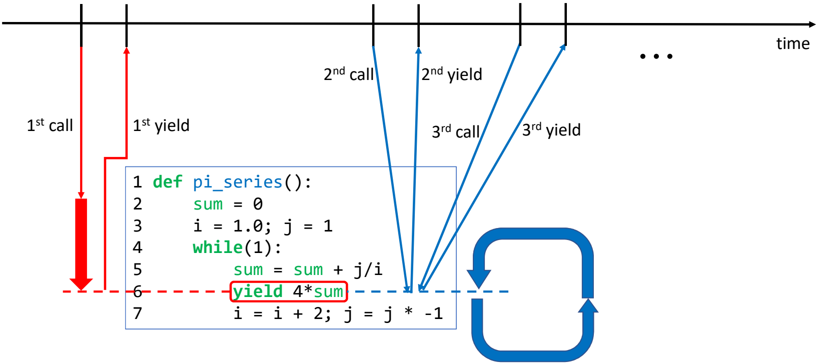
```
>>> gen = (x*x for x in range(10))
>>> next(gen)
0
>>> next(gen)
1
>>> next(gen)
4
```

- Inside a loop.

```
s = squared_series()
for i in s:
    print(i)
```

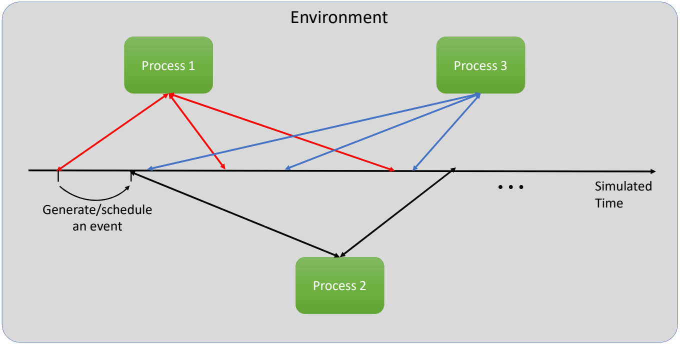
Generator IV

The following figure illustrates the execution of a generator function during its 1st call and calls thereafter.



SimPy: Basic Concepts I

- The behavior of an active component (e.g., a packet) is modeled with processes. All processes live in an environment. They interact with the environment and with each other via events.



SimPy: Basic Concepts II

- Processes are described by simple Python generators. During their lifetime, they create events and yield them in order to wait for them to be triggered.
- When a process yields an event, the process gets suspended. SimPy resumes the process, when the event occurs (we say that the event is triggered).
- An important event type is the Timeout. Events of this type are triggered after a certain amount of (simulated) time has passed. They allow a process to sleep (or hold its state) for the given time. A Timeout and all other events can be created by calling the appropriate method of the Environment that the process lives in (Environment.timeout() for example).

SimPy: Process Interaction

The Process instance that is returned by Environment.process() can be utilized for process interactions, including

- Waiting for another process to terminate
- Interrupting another process

SimPy: Shared Resources

SimPy offers three types of resources that help you modeling problems, where multiple processes want to use a resource of limited capacity (e.g., packets at a queueing system with a limited number of servers) or classical producer-consumer problems.

- *Resources* can be used by a limited number of processes at a time (e.g., a gas station with a limited number of fuel pumps).
- *Containers* model the production and consumption of a homogeneous, undifferentiated bulk. It may either be continuous (like water) or discrete (like apples).
- *Stores* allow the production and consumption of Python objects.

Example: M/M/1 Queue I

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     mm1.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2016-09-27
#
# @brief    Simulate M/M/1 queueing system
#
# @remarks  Copyright (C) 2016 Kyeong Soo (Joseph) Kim. All rights reserved.
#
# @remarks  This software is written and distributed under the GNU General
#           Public License Version 2 (http://www.gnu.org/licenses/gpl-2.0.html).
#           You must not remove this notice, or any other, from this software.
#

import argparse
import numpy as np
import random
import simpy
import sys


def source(env, mean_ia_time, mean_srv_time, server, wait_times, number, trace):
    """Generates packets with exponential interarrival time."""
    for i in range(number):
        ia_time = random.expovariate(1.0 / mean_ia_time)
        srv_time = random.expovariate(1.0 / mean_srv_time)
        pkt = packet(env, 'Packet-%d' % i, server, srv_time, wait_times, trace)
        env.process(pkt)
        yield env.timeout(ia_time)
```

Example: M/M/1 Queue II

```python


def packet(env, name, server, service_time, wait_times, trace):
    """Requests a server, is served for a given service_time, and leaves the server."""
    arrv_time = env.now
    if trace:
        print('t=%.4Es: %s arrived' % (arrv_time, name))

    with server.request() as request:
        yield request

        wait_time = env.now - arrv_time
        wait_times.append(wait_time)
        if trace:
            print('t=%.4Es: %s waited for %.4Es' % (env.now, name, wait_time))
        yield env.timeout(service_time)
        if trace:
            print('t=%.4Es: %s served for %.4Es' %
                  (env.now, name, service_time))


def run_simulation(mean_ia_time, mean_srv_time, num_packets=1000, random_seed=1234, trace=True):
    """Runs a simulation and returns statistics."""
    print('M/M/1 queue\n')
    random.seed(random_seed)
    env = simpy.Environment()

    # start processes and run
    server = simpy.Resource(env, capacity=1)
    wait_times = []
```

Example: M/M/1 Queue III

```
    env.process(source(env, mean_ia_time,
                       mean_srv_time, server, wait_times, number=num_packets, trace=trace))
    env.run()

    # return statistics (i.e., mean waiting time)
    return np.mean(wait_times)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-A",
        "--mean_ia_time",
        help="mean packet interarrival time [s]; default is 1.0",
        default=1.0,
        type=float)
    parser.add_argument(
        "-S",
        "--mean_srv_time",
        help="mean packet service time [s]; default is 0.1",
        default=0.1,
        type=float)
    parser.add_argument(
        "-N",
        "--num_packets",
        help="number of packets to generate; default is 1000",
        default=1000,
        type=int)
    parser.add_argument(
        "-R",
```

Example: M/M/1 Queue IV

```
            "--random_seed",
            help="seed for random number generation; default is 1234",
            default=1234,
            type=int)
    parser.add_argument('--trace', dest='trace', action='store_true')
    parser.add_argument('--no-trace', dest='trace', action='store_false')
    parser.set_defaults(trace=True)
    args = parser.parse_args()

    # set variables using command-line arguments
    mean_ia_time = args.mean_ia_time
    mean_srv_time = args.mean_srv_time
    num_packets = args.num_packets
    random_seed = args.random_seed
    trace = args.trace

    # run a simulation
    mean_waiting_time = run_simulation(
        mean_ia_time, mean_srv_time, num_packets, random_seed, trace)

    # print statistics from the simulation
    print("Average waiting time = %.4Es\n" % mean_waiting_time)
```

Development Environment

- Download and install WinPython distribution.
- Download a sample program "mm1.py" from the ICE.
- Open "WinPython Command Prompt" and type the following command:
  - "python mm1.py"
- Congratulations! You just installed SimPy and run your first simulation program.
- For a better development and debugging, we would recommend the following:
  - ipython within the "WinPython Command Prompt"
  - Spyder, a graphical IDE

References I

[1] L. Kleinrock, *Queueing Systems - Volume I; Theory*. John Wiley & Sons, 1974.