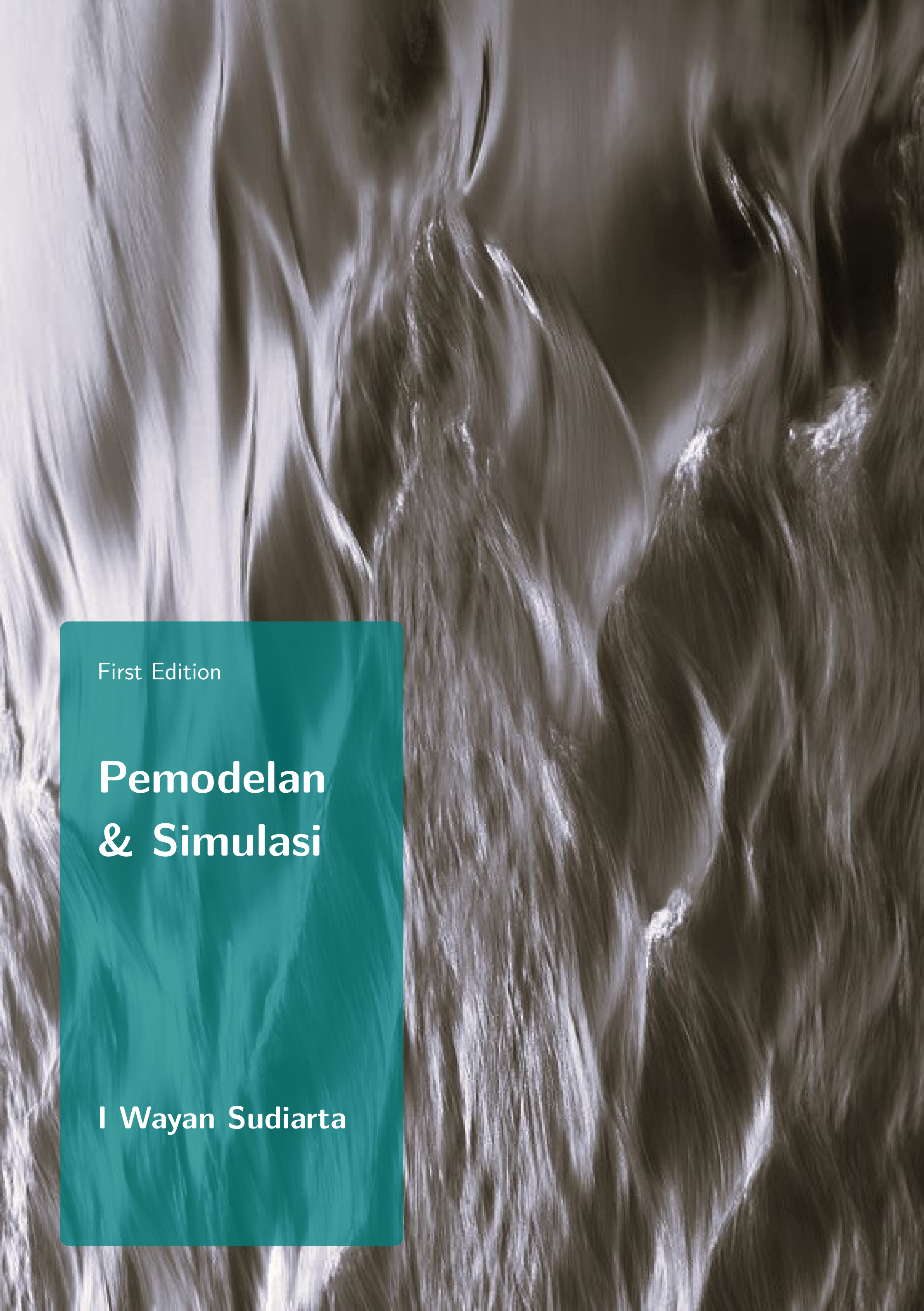


First Edition

Pemodelan & Simulasi

I Wayan Sudiarta

Pemodelan dan Simulasi
Author: I Wayan Sudiarta

Halaman awal: 14
Halaman isi: hlm.
Book Size: 148 × 210 mm
Tata letak buku ini dibuat dengan LaTeX
MikTeX 2019, editor TeXMaker
website: http://s.id/sudiarta
https://github.com/wayansudiarta/pemodelan-simulasi
e-mail: wayan.sudiarta@unram.ac.id

Cetakan pertama: tahun 2024
ISBN: 000-000-0000-00-0
Publisher: LPPM Press

Daftar Isi

Bab 1

Discrete Event Simulations

"Your exit criteria create a discrete event, ending the position and preventing the continuous process from going on and on." – Jim Paul

Bab ini membahas secara singkat tentang discrete event simulation (DES) atau simulasi kejadian diskrit khususnya penerapan untuk kasus antrian dan kemudian menggunakan pemrograman Python dengan modul Simpy untuk simulasi antrian.

Teori tentang antrian juga diungkapkan ketika membandingkan dengan hasil simulasi.

1.1 Langkah-langkah Projek Simulasi

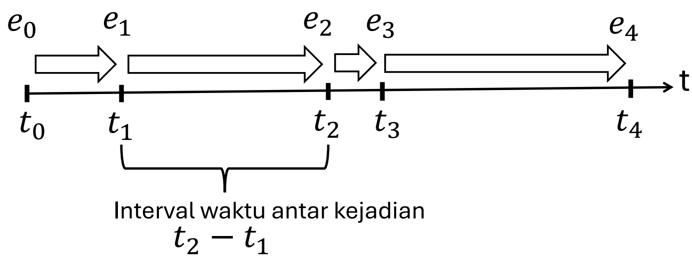
Dalam sebuah projek atau penelitian tentang simulasi atau model, beberapa langkah yang perlu dilakukan yaitu:

1. Menyusun struktur and desain simulasi/model
2. Menentukan/mengestimasi parameter yang akan digunakan, diperoleh dari dari data hasil eksperimen atau observasi
3. Menetapkan cara implementasi simulasi/model
4. Menganalisis hasil simulasi/model
5. Mempresentasikan hasil dan melaporkannya, memvisualisasi dan menarik kesimpulan.

[Perlu tambahan penjelasan tentang langkah-langkah di atas]

1.2 Pengenalan Discrete Event Simulations

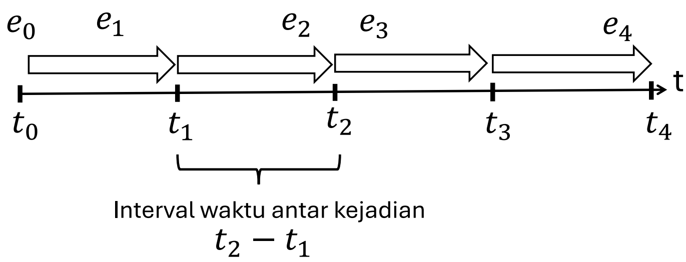
Discrete-Event Simulation (DES) sesuai dengan namanya, menggunakan simulasi kejadian atau peristiwa dari suatu proses atau operasi pada sebuah sistem berbentuk deretan ("sequence") kejadian pada waktu-waktu tertentu (disktrit). Setiap kejadian berada pada saat perubahan yang terjadi pada sistem. Dengan kata lain, pada DES terdapat lompatan kejadian-kejadian dari suatu waktu ke waktu berikutnya.



Gambar 1.1: *Waktu simulasi ($t_0 \ldots t_4$) digunakan untuk kejadian-kejadian ($e_0 \ldots e_4$) dengan teknik Next-Event Time Advance (NETA)*

Agar lebih jelas perhatikan Gambar 1.1, dalam simulasi DES kita dapat melompat dari waktu ke waktu sesuai kejadian atau yang disebut dengan teknik Next-Event Time Progression atau Next-Event Time Advance (NETA). Dengan kata lain, kita mengabaikan waktu di antara kejadian.

Cara lain yang bisa dilakukan adalah dengan menggunakan perubahan waktu yang tetap atau *Incremental Time Progression* (perhatikan Gambar 1.2. Setelah interval waktu, keadaan sistem diperbaharuai sesuai peristiwa yang sudah terjadi. Cara ini juga disebut dengan *Fixed-Increment Time Advance* (FITA).



Gambar 1.2: *Setiap keadaan sistem pada waktu tertentu dengan interval konstan atau teknik Next-Event Time Advance (FITA)*

DES banyak digunakan untuk mempelajari proses antrian. Pada kuliah ini kita membahas simulasi proses antrian pada berbagai kasus.

1.3 Notasi Kendal

Untuk membedakan dan memberikan label pada kasus sistem antrian, notasi Kendal biasanya digunakan. Notasi ini dituliskan seperti $A/B/m/K$ dengan arti sebagai berikut:

- A menyatakan distribusi apa yang digunakan untuk interval waktu antar kedatangan (*interarrival time*)

- B menyatakan distribusi interval waktu servis

- m adalah jumlah server
- K adalah kapasitas penampungan pada sistem, jika tidak tertulis maka nilai $K = \infty$.

Hal yang perlu kita ingat bahwa notasi ini tidak memberikan proses apa terjadi internal (operating policies) yang ada pada sistem. Juga tidak memberikan tipe entitas yang digunakan.

Simbol untuk distribusi probabilitas yang digunakan tertera pada dua huruf pertama notasi Kendal dengan arti sebagai berikut:

- G berarti *General distribution* (distribusi umum).
- GI berarti *General distribution with Independent interarrival time.*
- D menyatakan *Deterministic case*, berarti interval waktu tetap, konstan.
- M menyatakan distribusi untuk kasus Markovian, dengan distribusi probabilitas exponensial atau Poisson.

Perhatikan contoh: $M/M/1$ menyatakan bahwa sistem antrian dengan distribusi Markovian untuk *interarrival time* dan inteval servis.

1.4 Simulasi Antrian

Dalam simulasi dengan metode DES, kita perlu menentukan:

1. spesifikasi model stokastik (stochastic models) untuk interval waktu kedatangan dan servis. Perlu menentukan distribusi probabilitas.

2. parameter struktur (structural parameters) sistem seperti jumlah server, kapasitas penampungan pada antrian, lama delay pada proses.

3. operating policies: kondisi atau syarat yang berlaku pada proses atau peristiwa atau aktivitas pada sistem.

Biasanya kuantitas laju kedatangan atau banyaknya entitas per satuan waktu menggunakan simbol λ dan laju servis atau banyaknya entitas yang dilayani per satuan waktu (μ).

Simulasi untuk antrian kita perlu menentukan tiga hal yaitu

1. Entity : Has Attribute, track their journey

2. Resource: Capacity, Entities can visit resources

3. Source: Feed entities into simulation, Define interarrival time

Contoh Antrian single queue single server:

Gambar di sini

DES dapat dibagi menjadi tiga tipe: (1) *Event-scheduling*, (2) *Process-interaction* dan (3) *Activity-scanning.*

Pada bab ini kita menggunakan modul SimPy untuk simulasi. Simpy menerapkan process-oriented simulation scheme. Setiap entity melalui proses berikut:

1. entity datang ke sistem.

2. entity masuk ke antrian.

3. entity me"requests" servis dari server; jika server dalam keadaan idle (dapat digunakan), entity mendapatkan resource, jika tidak entity tetap dalam antrian sampai server menjadi idle.

4. entity setelah mendapat servis, entity tetap dalam servis selama periode tertentu sesuai dengan interval waktu servis.

5. setelah servis selesai, entity mengembalikan “releases” server.

6. entity meninggalkan sistem.

Sebuah proses pada DES berupa deretan fungsi-fungsi. Fungsi dibedakan menjadi tipe:

1. Fungsi Logika: Instantaneous actions taken by the entity that triggers this function in its process. For example, checking a condition such as “is the server idle?” or updating a data structure, such as recording the arrival time of the entity.

2. Fungsi waktu delay: The entity is held by that function for some period of time.

Dalam proses antrian dianalisis dengan memperhatikan beberapa ukuran atau perfomance measures yaitu:

1. rata-rata waiting time pada saat steady state, $E[W]$,

2. rata-rata system time pada saat steady state, $E[S]$,

3. rata-rata panjang antrian pada saat steady state, $E[X]$, yang diinginkan tentunya panjang antrian pendek.

4. utilisasi system, khususnya prosentase waktu penggunaan server.

5. luaran dari sistem, jumlah servis yang dapat dijalankan selama waktu tertenu.

Sebagai ukuran suatu antrian, intensitas trafik atau *traffic intensity* ρ didefinisikan

$$\rho \equiv \frac{\text{[average arrival rate]}}{\text{[average service rate]}} \tag{1.1}$$

atau

$$\rho \equiv \frac{\lambda}{\mu} \tag{1.2}$$

Jika ada sebanyak m server identik tersedia, *traffic intensity* menjadi

$$\rho \equiv \frac{\lambda}{m\mu} \tag{1.3}$$

Contoh kasus antrian:

Gambar

Selain itu, waktu sistem rata-rata per *entity* (contohnya customer) sampai pada waktu t dinotasikan dengan $\bar{s}(t)$ dapat diperoleh dengan

$$\bar{s}(t) = \frac{u(t)}{n(t)} \tag{1.4}$$

$u(t)$ adalah waktu total semua *entity* menghabiskan waktu di sistem sampai waktu t dan $n(t)$ adalah jumlah *entity* yang telah datang.

Hukum Little menyatakan bahwa pada keadaan *steady state* panjang antrian rata-rata $E[X]$ sebanding dengan lama waktu rata-rata berada pada sistem $E[S]$ mempunyai hubungan

$$E[X] = \lambda E[S] \tag{1.5}$$

dengan konstanta proporsionalitas yaitu laju kedatangan per satuan waktu, λ.

1.5 Penerapan Python dan Simpy

Python Generator

Sebelum membahas tentang simulasi menggunakan Simpy, mari kita perhatikan terlebih dahulu konsep generator Python pada fungsi berikut.

In[1]:
```
def simple_generator(n):
    i = 0
    while i < n:
        yield i
        i += 1
```

Bagian kode yang membedakan sebuah fungsi `simple_generator` sebagai sebuah generator adalah kata kunci **yield**. Ketika fungsi ini dipanggil, fungsi `simple_generator` akan memberikan angka sesuai perintah `yield i` yaitu 0. Jika kita panggil lagi fungsi ini akan memberikan nilai berikutnya yaitu 1 karena sudah ditambahkan 1 pada bagian `i += 1`. Begitu seterusnya jika fungsi ini dipanggil lagi. Fungsi generator ini menyimpan nilai value.

Mari perhatikan contoh penggunaan fungsi `simple_generator` berikut:

In[2]:
```
sg = simple_generator(10)
```

In [3]:
```
print(next(sg))
```

```
0
```

In [4]:
```
print(next(sg))
```

```
1
```

In [5]:
```
print(next(sg))
```

```
2
```

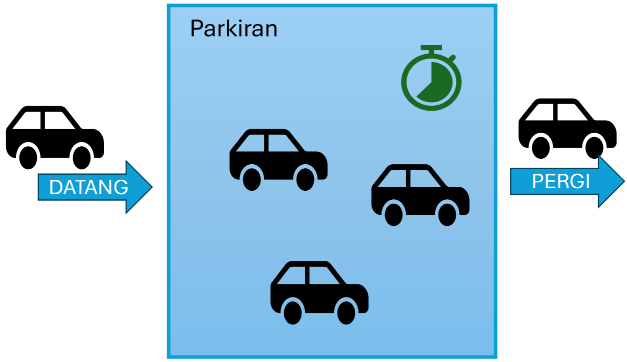
Simulasi Mobil Parkir Sederhana

Mari kita membuat simulasi sederhana. Simulasi berupa mobil datang ke parkiran secara berurutan dengan interval waktu 1 jam dan parkir selama 5 jam (lihat Gambar 1.3. Kita mengasumsikan tempat parkiran cukup luas jadi tidak ada batasan jumlah mobil yang parkir.

In [1]:
```
import simpy
```

In [2]:
```
def mobil(env, nama, waktu_datang):
    yield env.timeout(waktu_datang)
    print("%s datang pada waktu: %d" % (nama, env.
↪now))

    yield env.timeout(5)
    print("%s pergi pada waktu: %d" % (nama, env.
↪now))
```

In [3]:
```
env = simpy.Environment()
```



Gambar 1.3: *Simulasi mobil parkir*

In[6]:
```
for i in range(10):
    env.process(mobil(env, 'Mobil ' + str(i), i))
```

In[7]:
```
env.run()
```

```
Mobil 0 datang pada waktu: 9
Mobil 1 datang pada waktu: 10
Mobil 2 datang pada waktu: 11
Mobil 3 datang pada waktu: 12
Mobil 4 datang pada waktu: 13
Mobil 5 datang pada waktu: 14
Mobil 0 pergi pada waktu: 14
Mobil 6 datang pada waktu: 15
Mobil 1 pergi pada waktu: 15
Mobil 7 datang pada waktu: 16
Mobil 2 pergi pada waktu: 16
Mobil 8 datang pada waktu: 17
Mobil 3 pergi pada waktu: 17
```

```
Mobil 9 datang pada waktu: 18
Mobil 4 pergi pada waktu: 18
Mobil 5 pergi pada waktu: 19
Mobil 6 pergi pada waktu: 20
Mobil 7 pergi pada waktu: 21
Mobil 8 pergi pada waktu: 22
Mobil 9 pergi pada waktu: 23
```

Simulasi Mobil Parkir Terbatas

Melanjutkan simulasi sebulumnya, mobil datang, parkir dan kemudian pergi, tetapi dengan jumlah tempat parkir terbatas. Untuk simulasi kita menambahkan bagian resource yaitu parkiran dengan kapasitas 3 mobil seperti kode berikut.

In[15]:
```
parkiran = simpy.Resource(env, capacity=3)
```

Kemudian parkiran diinput ke fungsi mobil yang di dalamnya kita me"request" resource. Jika resource tersedia, request diterima dan menghasilkan tempat parkir. Jika tidak tersedia, mobil menunggu sampai ada tempat yang kosong. Bagian kode untuk request berikut ini.

In[13]:
```
    with parkiran.request() as req:
        yield req

        yield env.timeout(5)
        print("%s pergi pada waktu: %d" % (nama,␣
↪env.now))
```

Keseluruhan kode simulasi diberikan di bawah ini.

In[12]:
```python
import simpy
```

In[13]:
```python
def mobil(env, parkiran, nama, waktu_datang):
    yield env.timeout(waktu_datang)
    print("%s datang pada waktu: %d" % (nama, env.
↪now))

    with parkiran.request() as req:
        yield req

        yield env.timeout(5)
        print("%s pergi pada waktu: %d" % (nama,␣
↪env.now))
```

In[14]:
```python
env = simpy.Environment()
```

In[15]:
```python
parkiran = simpy.Resource(env, capacity=3)
```

In[16]:
```python
for i in range(10):
    env.process(mobil(env, parkiran, 'Mobil ' +␣
↪str(i), i))
```

In[17]:
```python
env.run()
```

```
Mobil 0 datang pada waktu: 0
Mobil 1 datang pada waktu: 1
Mobil 2 datang pada waktu: 2
Mobil 3 datang pada waktu: 3
Mobil 4 datang pada waktu: 4
Mobil 5 datang pada waktu: 5
Mobil 0 pergi pada waktu: 5
```

```
Mobil 6 datang pada waktu: 6
Mobil 1 pergi pada waktu: 6
Mobil 7 datang pada waktu: 7
Mobil 2 pergi pada waktu: 7
Mobil 8 datang pada waktu: 8
Mobil 9 datang pada waktu: 9
Mobil 3 pergi pada waktu: 10
Mobil 4 pergi pada waktu: 11
Mobil 5 pergi pada waktu: 12
Mobil 6 pergi pada waktu: 15
Mobil 7 pergi pada waktu: 16
Mobil 8 pergi pada waktu: 17
Mobil 9 pergi pada waktu: 20
```

Simulasi Mobil Terbatas Tercatat

Simulasi bagian ini sama seperti sebelumnya dengan penambahan kode untuk mencatat setiap kejadian dan waktunya. Kemudian hasilnya dianalisis dan ditampilkan. Perhatikan bagian untuk mencatat yaitu

In[2]:
```
log = []
```

Di bagian ini, pencatatan dilakukan dengan cara membuat variabel global `log` dengan tipe data list. Setiap kejadian ditambahkan dengan perintah `append`. Agar lebih lengkap dan mudah dimodifikasi, data setiap kejadian dicatat dalam tipe dictionary.

In[3]:
```
    data = {'nama': nama, 'waktu datang': env.now}

        data['waktu pergi'] = env.now
```

```
        log.append(data)
```

Kode lengkap diberikan berikut ini.

In[1]:
```
import simpy
```

In[2]:
```
log = []
```

In[3]:
```
def mobil(env, parkiran, nama, waktu_datang):
    yield env.timeout(waktu_datang)
    print("%s datang pada waktu: %d" % (nama, env.
 ↪now))
    data = {'nama': nama, 'waktu datang': env.now}

    with parkiran.request() as req:
        yield req

        yield env.timeout(5)
        print("%s pergi pada waktu: %d" % (nama,␣
 ↪env.now))
        data['waktu pergi'] = env.now
        log.append(data)
```

In[4]:
```
env = simpy.Environment()
```

In[5]:
```
parkiran = simpy.Resource(env, capacity=3)
```

In[6]:
```
for i in range(10):
    env.process(mobil(env, parkiran, 'Mobil ' +␣
 ↪str(i), i))
```

In[7]:
```
env.run()
```

```
Mobil 0 datang pada waktu: 0
Mobil 1 datang pada waktu: 1
Mobil 2 datang pada waktu: 2
Mobil 3 datang pada waktu: 3
Mobil 4 datang pada waktu: 4
Mobil 5 datang pada waktu: 5
Mobil 0 pergi pada waktu: 5
Mobil 6 datang pada waktu: 6
Mobil 1 pergi pada waktu: 6
Mobil 7 datang pada waktu: 7
Mobil 2 pergi pada waktu: 7
Mobil 8 datang pada waktu: 8
Mobil 9 datang pada waktu: 9
Mobil 3 pergi pada waktu: 10
Mobil 4 pergi pada waktu: 11
Mobil 5 pergi pada waktu: 12
Mobil 6 pergi pada waktu: 15
Mobil 7 pergi pada waktu: 16
Mobil 8 pergi pada waktu: 17
Mobil 9 pergi pada waktu: 20
```

In[8]:
```python
from pprint import pprint
pprint(log)
```

```
[{'nama': 'Mobil 0', 'waktu datang': 0, 'waktu␣
↪pergi': 5},
 {'nama': 'Mobil 1', 'waktu datang': 1, 'waktu␣
↪pergi': 6},
 {'nama': 'Mobil 2', 'waktu datang': 2, 'waktu␣
↪pergi': 7},
 {'nama': 'Mobil 3', 'waktu datang': 3, 'waktu␣
↪pergi': 10},
```

```
{'nama': 'Mobil 4', 'waktu datang': 4, 'waktu␣
↪pergi': 11},
{'nama': 'Mobil 5', 'waktu datang': 5, 'waktu␣
↪pergi': 12},
{'nama': 'Mobil 6', 'waktu datang': 6, 'waktu␣
↪pergi': 15},
{'nama': 'Mobil 7', 'waktu datang': 7, 'waktu␣
↪pergi': 16},
{'nama': 'Mobil 8', 'waktu datang': 8, 'waktu␣
↪pergi': 17},
{'nama': 'Mobil 9', 'waktu datang': 9, 'waktu␣
↪pergi': 20}]
```

In [9]:
```
import pandas as pd
df = pd.DataFrame.from_dict(log, orient='columns')
```

In [10]:
```
df
```

Out[10]:

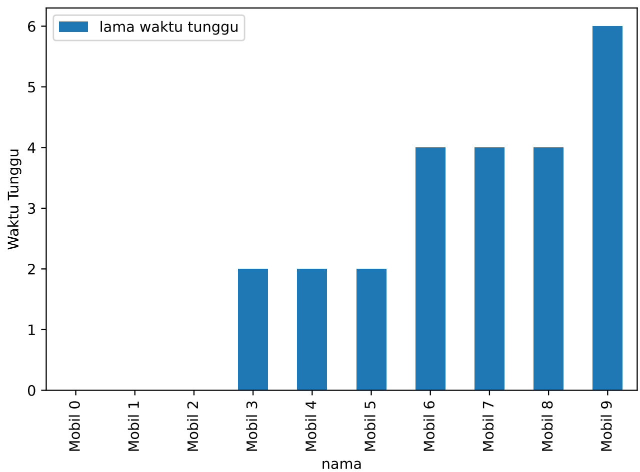
	nama	waktu datang	waktu pergi
0	Mobil 0	0	5
1	Mobil 1	1	6
2	Mobil 2	2	7
3	Mobil 3	3	10
4	Mobil 4	4	11
5	Mobil 5	5	12
6	Mobil 6	6	15
7	Mobil 7	7	16
8	Mobil 8	8	17
9	Mobil 9	9	20

In [11]:
```
df['lama waktu tunggu'] = df.loc[:, 'waktu pergi']␣
↪- df.loc[:, 'waktu datang'] - 5
```

In[12]:
```
df
```

Out[12]:
```
        nama  waktu datang  waktu pergi  lama waktu␣
↪tunggu
0  Mobil 0             0            5              ␣
↪     0
1  Mobil 1             1            6              ␣
↪     0
2  Mobil 2             2            7              ␣
↪     0
3  Mobil 3             3           10              ␣
↪     2
4  Mobil 4             4           11              ␣
↪     2
5  Mobil 5             5           12              ␣
↪     2
6  Mobil 6             6           15              ␣
↪     4
7  Mobil 7             7           16              ␣
↪     4
8  Mobil 8             8           17              ␣
↪     4
9  Mobil 9             9           20              ␣
↪     6
```

In[13]:
```
df.loc[:,['nama','lama waktu tunggu']].
↪plot(kind='bar', x = 'nama', y = 'lama waktu␣
↪tunggu', ylabel='Waktu Tunggu');
```



Mari kita tinjau hasil yang kita peroleh dengan simulasi ini. Parameter yang digunakan yaitu: laju kedatangan atau jumlah mobil per satuan waktu adalah 1 mobil/jam, dan lama parkir 5 jam setiap mobil (atau lama waktu servis) dan ada 3 tempat parkir (atau server). Kita mendapatkan parameter DES untuk single queue $\lambda = 1$, $\mu = 1/5$ dan $m = 3$.

Intensitas trafik (traffic intensity) untuk kasus ini

$$\rho = \frac{\lambda}{m\mu} = \frac{1}{3(1/5)} = \frac{5}{3} \tag{1.6}$$

Nilai intensitas trafik ini lebih besar dari 1, sehingga kita perhatikan bahwa lama waktu tunggu semakin lama seiring dengan bertambahnya mobil yang datang. Agar intensitas trafik kurang dari atau sama dengan 1, kapasitas parkiran dapat diperbanyak minimal dapat menampung 5 mobil.

1.6 Kasus-Kasus DES

Kasus-kasus DES untuk sementara belum dimasukkan di buku ini. Bacalah, pahami dan cobalah dokumen jupyter notebook yang menyertai buku ini.

Soal-soal

Soal 1.1. Buatlah kode fungsi generator Python untuk menghasilkan bilangan Fibonacci dengan persamaan berikut.

$$F(n) = \begin{cases} 0 & \text{jika } n = 0 \\ 1 & \text{jika } n = 1 \\ F(n-1) + F(n-2) & \text{jika n} > 1 \end{cases} \tag{1.7}$$

Soal 1.2. Simulasikan kejadian pada lampu lalulintas (*traffic light*) berubah dari warna merah, ke hijau dan kemudian kuning berulang-ulang.

Soal 1.3. Untuk kasus simulasi mobil parkir terbatas, apa yang terjadi ketika kapasitas tempat parkir ditambah? Pada kapasitas berapa lama antrian menjadi nol?

Soal 1.4. Ubah simulasi mobil parkir dengan cara mengganti (1) interval waktu kedatangan menggunakan distribusi eksponensial dengan laju kedatangan rata-rata 1 mobil per jam dan (2) interval waktu parkir dengan distribusi normal (rata-rata = 4 jam dan deviasi standar = 1 jam)

Soal 1.5. Seorang dalam keadaan mabuk (pemabuk) ingin pulang ke rumah. Umpama pemabuk ini hanya bergerak dengan melangkah maju dan mundur. Probabilitas gerak maju 3 kali lebih besar dari probabilitas gerak mundur. Buatlah simulasi gerak pemabuk ini dengan modul Simpy dan Random. Asumsi satu langkah adalah $0,5$ meter. Setelah berapa langkah pemabuk ini mencapai jarak 10 m?

Simulasi Klinik Kasus 1: Creating a patient arrival process

I Wayan Sudiarta, Ph.D.

Mengikuti Kode diberikan pada Ref. 1.

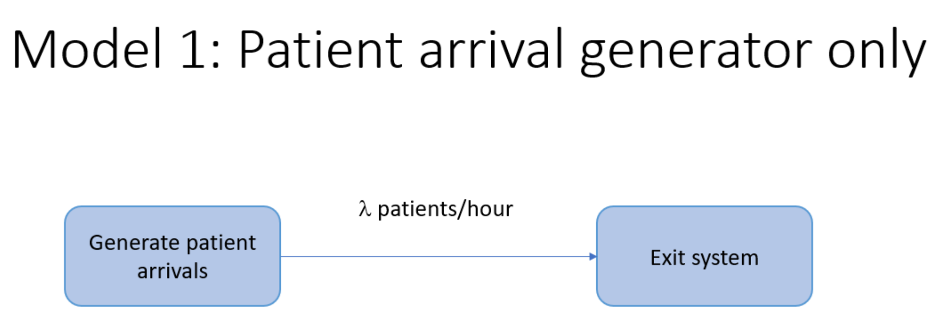
References:

1. https://bitsofanalytics.org/posts/simpy-vaccine-clinic-part1/simpy_getting_started_vaccine_clinic.html

In [1]:
```
# Import Modul Python
# Manipulasi Data
import numpy as np
import pandas as pd
from numpy.random import default_rng
from scipy import optimize

# Visualisasi
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image

# Simulasi
import simpy

# output inline
%matplotlib inline
```

In [2]:
```
Image("patient-arrival.png", width=500) # gambar dari Ref. 1.
```

Out[2]:



Kasus Interval Waktu Tetap

In [3]:
```
# Ref. 1.
def patient_arrivals(env, interarrival_time=5.0):
    """Generate patients according to a fixed time arrival process"""
```

```python
    # Create a counter to keep track of number of patients generated and
    # to serve as unique patient id
    patient = 0

    # Infinite loop for generating patients
    while True:

        # Generate next interarrival time (this will be more complicated later)
        iat = interarrival_time

        # This process will now yield to a 'timeout' event. This process will
        # resume after iat time units.
        yield env.timeout(iat)

        # Okay, we're back. :) New patient generated = update counter of patients
        patient += 1

        print(f"Patient {patient} created at time {env.now}")
```

In [4]:

```python
# Membuat environmnet
env = simpy.Environment()
```

In [5]:

```python
# Interval waktu kedatangan
interarrival_time = 2

# Buat Generator patient_arrivals
arrival_generator = patient_arrivals(env, interarrival_time)

# Daftarkan/Masukkan proses ke environment
env.process(arrival_generator)
```

Out[5]:

```
<Process(patient_arrivals) object at 0x190bf22cdf0>
```

Ada dua hal yang dilakukan kode di atas:

1. Bagian **patient_arrivals (env, interarrival_time)** memanggil fungsi generator patient_arrivals dan mendapatkan kembali generator Python.
2. Bagian **env.process()** mendaftarkan generator ini dengan lingkungan simulasi (**env** dalam contoh ini).

Untuk setiap fungsi generator yang Anda tulis (yaitu fungsi apa pun yang berisi pernyataan yield), Anda HARUS mendaftarkannya dengan enironment simulasi dengan melewatkan generator dengan fungsi process. [Diambil dari Ref. 1]

In [6]:

```python
# Jalankan simulasi selama runtime
runtime = 30
env.run(until=runtime)
```

```
Patient 1 created at time 2
Patient 2 created at time 4
Patient 3 created at time 6
Patient 4 created at time 8
Patient 5 created at time 10
Patient 6 created at time 12
Patient 7 created at time 14
Patient 8 created at time 16
Patient 9 created at time 18
Patient 10 created at time 20
Patient 11 created at time 22
Patient 12 created at time 24
Patient 13 created at time 26
Patient 14 created at time 28
```

Model Poisson

Dari perspektif pemodelan, pasien yang tiba pada waktu interarrival dengan interval yang sama sangat tidak realistis. Jika kita memodelkan "klinik berjalan" di mana tidak ada pasien yang menjadwalkan appointment, akan lebih tepat untuk memodelkan kedatangan pasien dengan sesuatu yang dikenal sebagai proses kedatangan Poisson. Proses-proses ini ditandai oleh:

1. tingkat kedatangan rata-rata konstan (biasanya dilambangkan dengan λ),
2. waktu antara kedatangan individu didistribusikan secara eksponensial,
3. kedatangan tidak tergantung satu sama lain,
4. jumlah kedatangan adalah setiap interval waktu panjang, adalah Poisson didistribusikan dengan rata-rata.

Proses Poisson biasanya digunakan untuk memodelkan hal-hal seperti panggilan ke Call Centre, kedatangan pasien ke ruang gawat darurat, dan bahkan hal-hal seperti kejadian angin topan.

Karena proses kedatangan Poisson memiliki waktu interarrival yang didistribusikan secara eksponensial, kita perlu menghasilkan variasi acak eksponensial dalam fungsi **patient_arrivals** kita. Untuk ini, kita akan menggunakan numpy. Baru-baru ini, numpy telah memperbarui rutinitas pembuatan variabel acak mereka - detailnya ada di https://numpy.org/doc/stable/reference/random/index.html.

In [7]:
```
from numpy.random import default_rng
rg = default_rng(seed=11344)
```

In [8]:
```
# Tambahkan list untuk mencatat event
event_log = []
```

In [9]:
```
def patient_arrivals_random_1(env,
                              mean_interarrival_time=5.0, rg=default_rng(0)):
    """Generate patients according to a Poisson arrival process"""
```

```python
    # Create a counter to keep track of number of patients generated
    # and to serve as unique patient id
    patient = 0

    # Infinite loop for generating patients
    while True:

        # Generate next interarrival time from exponential distribution
        iat = rg.exponential(mean_interarrival_time)

        # This process will now yield to a 'timeout' event. This process
        # will resume after iat time units.
        yield env.timeout(iat)

        # Update counter of patients
        patient += 1

        print(f"Patient {patient} created at time {env.now}")

        # tambahkan pada event_log
        event_log.append({'patient': patient,
                          'time': env.now})
```

In [10]:

```python
# Initialize a simulation environment
env2 = simpy.Environment()

# Create a process generator and start it and add it to the env
# env.process() starts and adds it to env
runtime = 25
interarrival_time = 3.0

arrival_random = patient_arrivals_random_1(env2, interarrival_time)
env2.process(arrival_random)

# Run the simulation
env2.run(until=runtime)
```

```
Patient 1 created at time 2.039795711906729
Patient 2 created at time 5.098587016304323
Patient 3 created at time 5.158007004071489
Patient 4 created at time 5.164814984115174
Patient 5 created at time 6.815843602032318
Patient 6 created at time 11.705664906007474
Patient 7 created at time 13.72641376400917
Patient 8 created at time 15.992317837485343
Patient 9 created at time 24.44267577471252
```

In [11]:

```python
event_log
```

Out[11]:
```
[{'patient': 1, 'time': 2.039795711906729},
 {'patient': 2, 'time': 5.098587016304323},
 {'patient': 3, 'time': 5.158007004071489},
 {'patient': 4, 'time': 5.164814984115174},
 {'patient': 5, 'time': 6.815843602032318},
 {'patient': 6, 'time': 11.705664906007474},
 {'patient': 7, 'time': 13.72641376400917},
 {'patient': 8, 'time': 15.992317837485343},
 {'patient': 9, 'time': 24.44267577471252}]
```

In [ ]:

Simulasi Klinik Kasus 2: Simplified vaccine clinic with delay processes and one resource

I Wayan Sudiarta, Ph.D.

Menggunakan Ref. 1.

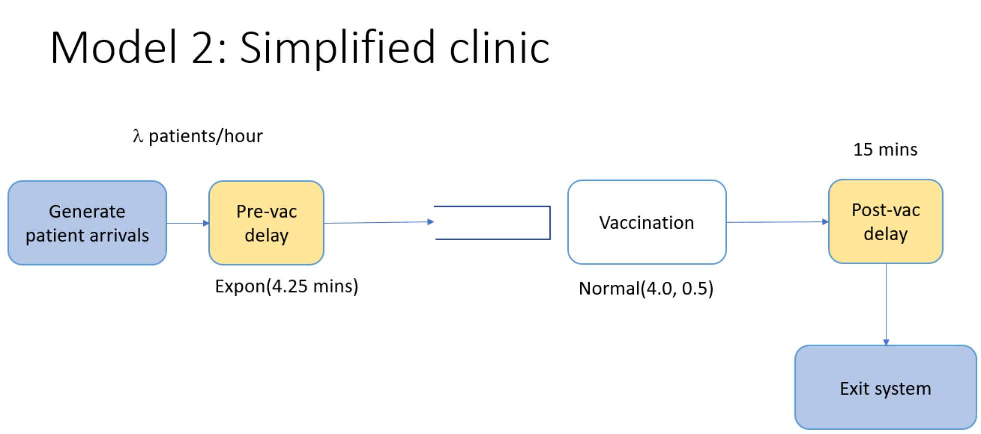
References:

1. https://bitsofanalytics.org/posts/simpy-vaccine-clinic-part1/simpy_getting_started_vaccine_clinic.html

In [1]:
```
# Import Modul Python
# Manipulasi Data
import numpy as np
import pandas as pd
from numpy.random import default_rng
from scipy import optimize

# Visualisasi
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image

# Simulasi
import simpy

# output inline
%matplotlib inline
```

In [2]:
```
Image("model-klinik-2.png", width=500) # gambar dari Ref. 1.
```

Out[2]:



Kita mensimulasikan proses vaksinasi yang sederhana.

Asumsi digunakan pada simulasi ini:

1. Pasien datang secara acak dengan distribusi exponensial dengan rata-rata interval waktu $mean_interarrival_time = 3$ menit
2. Pasien melalui proses pre-vaksinasi dengan delay tertentu, distribusi eksponensial dengan rata-rata 4.25 menit.
3. Pasien kemudian antri pada ruang tunggu sebelum mendapatkan vaksinasi.
4. Durasi vaksinasi menggunakan distribusi normal dengan rata-rata 4 menit dan standar deviasi 0.5 menit.

Kode yang perlu kita perhatikan:

```
with vaccinator.request() as request:
    yield request
    yield env.timeout(rg.normal(mean_vac_time, 0.5)
```

Kita menggunakan **with vaccinator.request()** untuk permintaan ke "context manager" **SimPy**. Bagian ini akan menunggu sampai resource tersedia.

In [3]:
```
from numpy.random import default_rng
rg = default_rng(seed=11344)
```

In [4]:
```
# Ref. 1.
def simplified_vac_process(env, name, mean_prevac_time,
                           mean_vac_time, mean_postvac_time, vaccinator):
    """Process function modeling how a patient flows through system."""

    print(f"{name} entering vaccination clinic at {env.now:.4f}")

    # Yield for the pre-vac time
    yield env.timeout(rg.exponential(mean_prevac_time))

    # Request vaccination staff to get vaccinated
    with vaccinator.request() as request:
        print(f"{name} requested vaccinator at {env.now:.4f}")
        yield request
        print(f"{name} got vaccinator at {env.now:.4f}")
        yield env.timeout(rg.normal(mean_vac_time, 0.5))

    # Yield for the post-vac time
    yield env.timeout(mean_postvac_time)

    # The process is over, we would exit the clinic
    print(f"{name} exiting vaccination clinic at {env.now:.4f}")
```

In [5]:
```
# Ref 1.
def patient_arrivals_random_2(env, mean_interarrival_time,
                              mean_prevac_time, mean_vac_time,
                              mean_postvac_time, vaccinator, rg=default_rng(0)):
    """Generate patients according to a Poisson arrival process"""

    # Create a counter to keep track of number of patients generated
    # and to serve as unique patient id
```

```python
    patient = 0

    # Infinite loop for generating patients
    while True:

        # Generate next interarrival time
        iat = rg.exponential(mean_interarrival_time)

        # This process will now yield to a 'timeout' event.
        # This process will resume after iat time units.
        yield env.timeout(iat)

        # Update counter of patients
        patient += 1

        print(f"Patient{patient} created at time {env.now}")

        # Create and register the simplifed vaccation process process.
        # Create a new patient delay process generator object.
        patient_visit = simplified_vac_process(
            env, 'Patient{}'.format(patient), mean_prevac_time,
            mean_vac_time, mean_postvac_time, vaccinator)

        # Register the process with the simulation environment
        env.process(patient_visit)
```

In [6]:

```python
# Initialize a simulation environment
env = simpy.Environment()

# Set input values
mean_interarrival_time = 3.0
mean_prevac_time = 4.25
mean_vac_time = 4.0
mean_postvac_time = 15.0
num_vaccinators = 2

# Create vaccinator resource
vaccinator = simpy.Resource(env, num_vaccinators)

# Register our new arrivals process
env.process(patient_arrivals_random_2(
    env, mean_interarrival_time, mean_prevac_time,
    mean_vac_time, mean_postvac_time, vaccinator))

# Run the simulation
runtime = 50
env.run(until=runtime)
```

```
Patient1 created at time 2.039795711906729
Patient1 entering vaccination clinic at 2.0398
Patient1 requested vaccinator at 2.1466
Patient1 got vaccinator at 2.1466
Patient2 created at time 5.098587016304323
Patient2 entering vaccination clinic at 5.0986
Patient3 created at time 5.158007004071489
Patient3 entering vaccination clinic at 5.1580
Patient4 created at time 5.164814984115174
Patient4 entering vaccination clinic at 5.1648
Patient4 requested vaccinator at 5.5334
Patient4 got vaccinator at 5.5334
Patient3 requested vaccinator at 5.8842
Patient2 requested vaccinator at 6.2385
Patient3 got vaccinator at 6.3154
Patient5 created at time 6.815843602032318
Patient5 entering vaccination clinic at 6.8158
Patient5 requested vaccinator at 7.4472
Patient2 got vaccinator at 9.3710
Patient5 got vaccinator at 10.0824
Patient6 created at time 11.705664906007474
Patient6 entering vaccination clinic at 11.7057
Patient6 requested vaccinator at 12.0489
Patient6 got vaccinator at 13.5589
Patient7 created at time 13.72641376400917
Patient7 entering vaccination clinic at 13.7264
Patient8 created at time 15.992317837485343
Patient8 entering vaccination clinic at 15.9923
Patient7 requested vaccinator at 16.3304
Patient7 got vaccinator at 16.3304
Patient8 requested vaccinator at 18.4969
Patient8 got vaccinator at 18.4969
Patient1 exiting vaccination clinic at 21.3154
Patient4 exiting vaccination clinic at 24.3710
Patient9 created at time 24.44267577471252
Patient9 entering vaccination clinic at 24.4427
Patient3 exiting vaccination clinic at 25.0824
Patient2 exiting vaccination clinic at 28.5589
Patient9 requested vaccinator at 28.5605
Patient9 got vaccinator at 28.5605
Patient5 exiting vaccination clinic at 29.7276
Patient6 exiting vaccination clinic at 32.2030
Patient7 exiting vaccination clinic at 35.1510
Patient8 exiting vaccination clinic at 36.5380
Patient10 created at time 42.61593501604024
Patient10 entering vaccination clinic at 42.6159
Patient9 exiting vaccination clinic at 48.2462
```

In [ ]:

Simulasi Klinik Kasus 3: The vaccine clinic model

I Wayan Sudiarta, Ph.D.

Menggunakan Ref. 1.

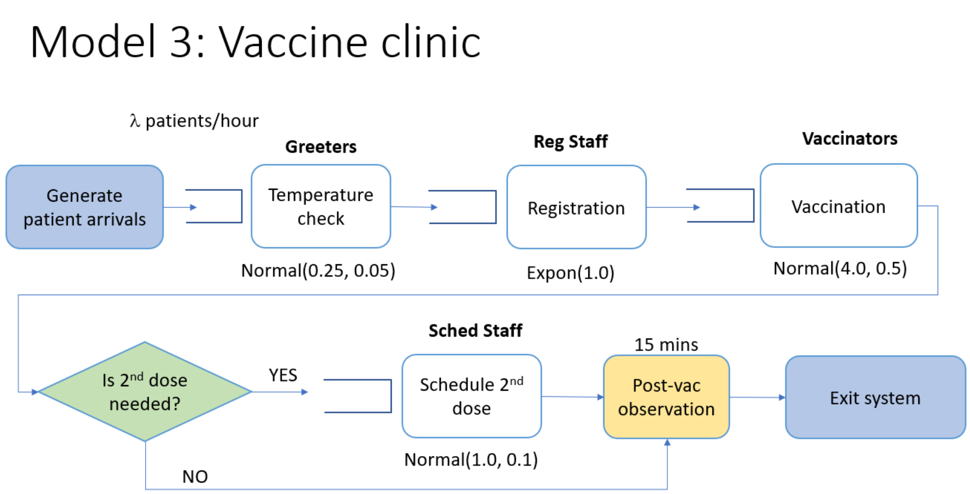
References:

1. https://bitsofanalytics.org/posts/simpy-vaccine-clinic-part1/simpy_getting_started_vaccine_clinic.html

In [1]:
```
# Import Modul Python
# Manipulasi Data
import numpy as np
import pandas as pd
from numpy.random import default_rng
from scipy import optimize

# Visualisasi
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image

# Simulasi
import simpy

# output inline
%matplotlib inline
```

In [2]:
```
Image("model-klinik-3.png", width=500) # gambar dari Ref. 1.
```

Out[2]:



In [3]:
```
from numpy.random import default_rng
rg = default_rng(seed=4470)
```

In [4]:
```python
# Ref. 1.
class VaccineClinic(object):
    def __init__(self, env, num_greeters, num_reg_staff,
                 num_vaccinators, num_schedulers, rg):
        # Simulation environment
        self.env = env
        self.rg = rg

        # Create list to hold timestamps dictionaries (one per patient)
        self.timestamps_list = []
        # Create lists to hold occupancy tuples (time, occ)
        self.postvac_occupancy_list = [(0.0, 0.0)]
        self.vac_occupancy_list = [(0.0, 0.0)]

        # Create resources
        self.greeter = simpy.Resource(env, num_greeters)
        self.reg_staff = simpy.Resource(env, num_reg_staff)
        self.vaccinator = simpy.Resource(env, num_vaccinators)
        self.scheduler = simpy.Resource(env, num_schedulers)

    # Create process methods - hard coding processing time distributions for now
    # TODO - remove hard coding
    # The patient argument is just a unique integer number
    def temperature_check(self, patient):
        yield self.env.timeout(self.rg.normal(0.25, 0.05))

    def registration(self, patient):
        yield self.env.timeout(self.rg.exponential(1.0))

    def vaccinate(self, patient):
        yield self.env.timeout(self.rg.normal(4.0, 0.5))

    def schedule_dose_2(self, patient):
        yield self.env.timeout(self.rg.normal(1.0, 0.25))

    # We assume all patients wait at least 15 minutes post-vaccination
    # Some will choose to wait longer. This is the time beyond 15 minutes
    # that patients wait.
    def wait_gt_15(self, patient):
        yield self.env.timeout(self.rg.exponential(0.5))
```

In [5]:
```python
def get_vaccinated(env, patient, clinic, pct_first_dose, rg):
    # Patient arrives to clinic - note the arrival time
    arrival_ts = env.now

    # Request a greeter for temperature check
    # By using request() in a context manager, we'll automatically
    # release the resource when done
    with clinic.greeter.request() as request:
        yield request
        # Now that we have a greeter, check temperature. Note time.
        got_greeter_ts = env.now
        yield env.process(clinic.temperature_check(patient))
        release_greeter_ts = env.now
```

```python
    # Request reg staff to get registered
    with clinic.reg_staff.request() as request:
        yield request
        got_reg_ts = env.now
        yield env.process(clinic.registration(patient))
        release_reg_ts = env.now

    # Request clinical staff to get vaccinated
    with clinic.vaccinator.request() as request:
        yield request
        got_vaccinator_ts = env.now
        # Update vac occupancy - increment by 1
        prev_occ = clinic.vac_occupancy_list[-1][1]
        new_occ = (env.now, prev_occ + 1)
        clinic.vac_occupancy_list.append(new_occ)
        yield env.process(clinic.vaccinate(patient))
        release_vaccinator_ts = env.now
# Update vac occupancy - decrement by 1 - more compact code
# Note that clinic.vac_occupancy_list[-1] is the last tuple in the list
#  and that clinic.vac_occupancy_list[-1][1] is referencing the occupancy
#  value in the tuple (remember tuple elements are indexed starting with 0, so
#  the timestamp is at [0] and the occupancy is at [1]).
#  BTW, this suggests that perhaps using something known as "namedtuples" might
#  make our code more readable. See https://realpython.com/python-namedtuple/
#  for a good introduction to namedtuples.
        clinic.vac_occupancy_list.append((
            env.now, clinic.vac_occupancy_list[-1][1] - 1))

        # Update postvac occupancy - increment by 1
        clinic.postvac_occupancy_list.append((
            env.now, clinic.postvac_occupancy_list[-1][1] + 1))

    # Request scheduler to schedule second dose if needed
    if rg.random() < pct_first_dose:
        with clinic.scheduler.request() as request:
            yield request
            got_scheduler_ts = env.now
            yield env.process(clinic.schedule_dose_2(patient))
            release_scheduler_ts = env.now
    else:
        got_scheduler_ts = pd.NA
        release_scheduler_ts = pd.NA

    # Wait at least 15 minutes from time we finished getting vaccinated
    post_vac_time = env.now - release_vaccinator_ts
    if post_vac_time < 15:
        # Wait until 15 total minutes post vac
        yield env.timeout(15 - post_vac_time)
        # Wait random amount beyond 15 minutes
        yield env.process(clinic.wait_gt_15(patient))
        exit_system_ts = env.now

        # Update postvac occupancy - decrement by 1
        clinic.postvac_occupancy_list.append((
            env.now, clinic.postvac_occupancy_list[-1][1] - 1))
```

```python
    exit_system_ts = env.now
    #print(f"Patient {patient} exited at time {env.now}")

    # Create dictionary of timestamps
    timestamps = {'patient_id': patient,
                  'arrival_ts': arrival_ts,
                  'got_greeter_ts': got_greeter_ts,
                  'release_greeter_ts': release_greeter_ts,
                  'got_reg_ts': got_reg_ts,
                  'release_reg_ts': release_reg_ts,
                  'got_vaccinator_ts': got_vaccinator_ts,
                  'release_vaccinator_ts': release_vaccinator_ts,
                  'got_scheduler_ts': got_scheduler_ts,
                  'release_scheduler_ts': release_scheduler_ts,
                  'exit_system_ts': exit_system_ts}

    clinic.timestamps_list.append(timestamps)
```

In [6]:
```python
def run_clinic(env, clinic, mean_interarrival_time, pct_first_dose, rg,
               stoptime=simpy.core.Infinity, max_arrivals=simpy.core.Infinity):

    # Create a counter to keep track of number of patients
    # generated and to serve as unique patient id
    patient = 0

    # Loop for generating patients
    while env.now < stoptime and patient < max_arrivals:

        # Generate next interarrival time
        iat = rg.exponential(mean_interarrival_time)

        # This process will now yield to a 'timeout' event.
        # This process will resume after iat time units.
        yield env.timeout(iat)

        # New patient generated = update counter of patients
        patient += 1

        #print(f"Patient {patient} created at time {env.now}")

        env.process(get_vaccinated(env, patient, clinic, pct_first_dose, rg))
```

In [7]:
```python
# For now we are hard coding the patient arrival rate (patients per hour)
patients_per_hour = 180
mean_interarrival_time = 1.0 / (patients_per_hour / 60.0)
pct_first_dose = 0.50

# Create a random number generator
rg = default_rng(seed=4470)

# For now we are going to hard code in the resource capacity levels
num_greeters = 5
num_reg_staff = 5
num_vaccinators = 13
```

```
num_schedulers = 5

# Hours of operation
stoptime = 600 # No more arrivals after this time

# Create a simulation environment
env = simpy.Environment()
# Create a clinic to simulate
clinic = VaccineClinic(env, num_greeters, num_reg_staff,
                       num_vaccinators, num_schedulers, rg)

# Register the run_clinic (generator) function
env.process(run_clinic(env, clinic, mean_interarrival_time,
                       pct_first_dose, rg, stoptime=stoptime))
# Actually run the simulation
env.run()

# The simulation is over now, let's create the output csv files from
# the dataframes created by running the simulation model.

# Output log files
clinic_patient_log_df = pd.DataFrame(clinic.timestamps_list)
clinic_patient_log_df.to_csv('clinic_patient_log_df.csv', index=False)

vac_occupancy_df = pd.DataFrame(clinic.vac_occupancy_list, columns=['ts', 'occ'])
vac_occupancy_df.to_csv('vac_occupancy_df.csv', index=False)

postvac_occupancy_df = pd.DataFrame(
    clinic.postvac_occupancy_list, columns=['ts', 'occ'])
postvac_occupancy_df.to_csv('postvac_occupancy_df.csv', index=False)

# Note simulation end time
end_time = env.now
print(f"Simulation ended at time {end_time}")
```

```
Simulation ended at time 623.526585622315
```

Visualisasi

In [8]:

```
clinic_patient_log_df = pd.read_csv('clinic_patient_log_df.csv')
clinic_patient_log_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1798 entries, 0 to 1797
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   patient_id              1798 non-null   int64
 1   arrival_ts              1798 non-null   float64
 2   got_greeter_ts          1798 non-null   float64
 3   release_greeter_ts      1798 non-null   float64
 4   got_reg_ts              1798 non-null   float64
 5   release_reg_ts          1798 non-null   float64
 6   got_vaccinator_ts       1798 non-null   float64
 7   release_vaccinator_ts   1798 non-null   float64
 8   got_scheduler_ts        887 non-null    float64
 9   release_scheduler_ts    887 non-null    float64
 10  exit_system_ts          1798 non-null   float64
dtypes: float64(10), int64(1)
memory usage: 154.6 KB
```

In [9]:
```python
clinic_patient_log_df.head()
```

Out[9]:

	patient_id	arrival_ts	got_greeter_ts	release_greeter_ts	got_reg_ts	release_reg_ts	got_vaccinatc
0	7	0.827865	0.827865	1.128493	1.414543	1.649776	1.64§
1	2	0.288373	0.288373	0.565944	0.565944	1.484153	1.484
2	9	0.962394	0.962394	1.273385	1.498804	2.809295	2.80§
3	10	1.163169	1.163169	1.381548	1.649776	1.884211	1.884
4	1	0.259164	0.259164	0.434629	0.434629	1.139493	1.13§

In [10]:
```python
def compute_durations(timestamp_df):
    timestamp_df['wait_for_greeter'] = \
        timestamp_df.loc[:, 'got_greeter_ts'] - timestamp_df.loc[:, 'arrival_ts']
    timestamp_df['wait_for_reg'] = \
        timestamp_df.loc[:, 'got_reg_ts'] - timestamp_df.loc[:, 'release_greeter_ts
    timestamp_df['wait_for_vaccinator'] = \
        timestamp_df.loc[:, 'got_vaccinator_ts'] - timestamp_df.loc[:, 'release_reg
    timestamp_df['vaccination_time'] = \
        timestamp_df.loc[:, 'release_vaccinator_ts'] - timestamp_df.loc[:, 'got_vac
    timestamp_df['wait_for_scheduler'] = \
        timestamp_df.loc[:, 'got_scheduler_ts'] - timestamp_df.loc[:, 'release_vacc
    timestamp_df['post_vacc_time'] = \
        timestamp_df.loc[:, 'exit_system_ts'] - timestamp_df.loc[:, 'release_vaccin
    timestamp_df['time_in_system'] = \
        timestamp_df.loc[:, 'exit_system_ts'] - timestamp_df.loc[:, 'arrival_ts']

    return timestamp_df
```

In [11]:
```python
clinic_patient_log_df = compute_durations(clinic_patient_log_df)
clinic_patient_log_df
```

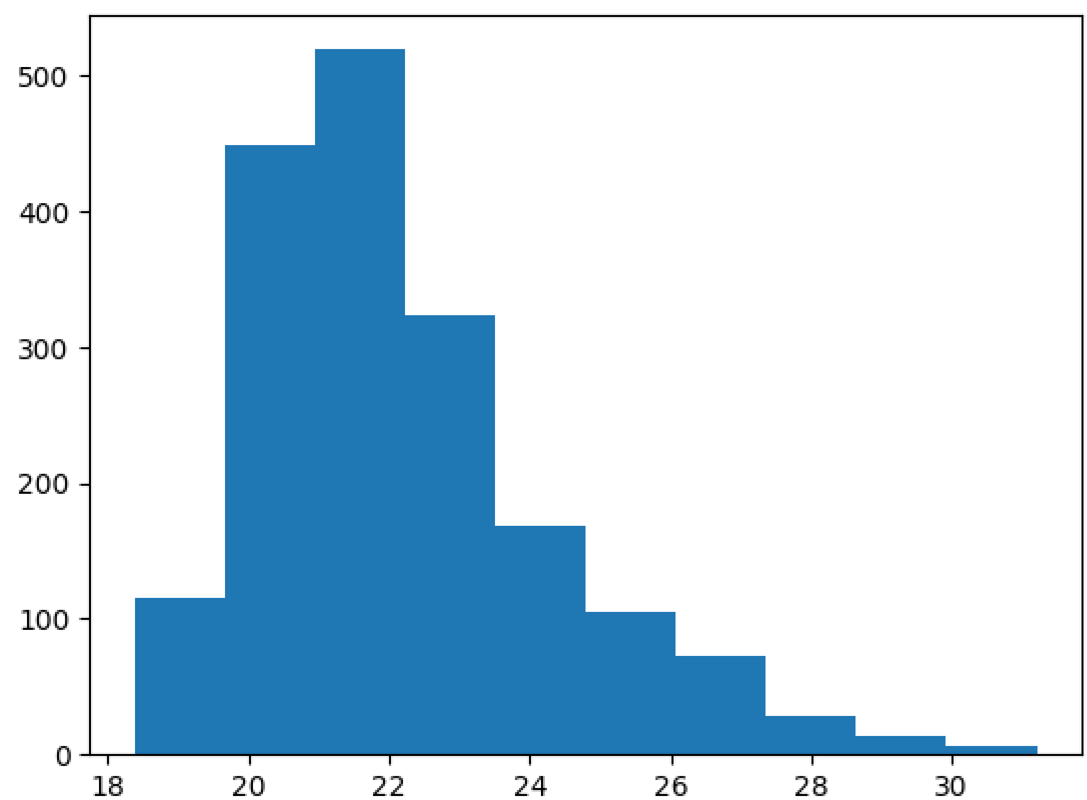
Out[11]:

	patient_id	arrival_ts	got_greeter_ts	release_greeter_ts	got_reg_ts	release_reg_ts	got_vac
0	7	0.827865	0.827865	1.128493	1.414543	1.649776	
1	2	0.288373	0.288373	0.565944	0.565944	1.484153	
2	9	0.962394	0.962394	1.273385	1.498804	2.809295	
3	10	1.163169	1.163169	1.381548	1.649776	1.884211	
4	1	0.259164	0.259164	0.434629	0.434629	1.139493	
...	...	...	...	...	...	...	
1793	1792	596.193904	596.193904	596.471168	596.471168	598.219305	5
1794	1797	599.113610	599.113610	599.430689	599.430689	600.715132	6
1795	1796	598.985033	598.985033	599.329011	599.329011	600.552033	6
1796	1795	598.914456	598.914456	599.119148	599.119148	601.946829	6
1797	1798	600.282169	600.282169	600.510972	600.510972	603.257923	6

1798 rows × 18 columns

In [12]:

```
plt.hist(clinic_patient_log_df['time_in_system']);
```



In [13]:

```
clinic_patient_log_df.loc[:, ['wait_for_greeter', 'wait_for_reg',
                              'wait_for_scheduler', 'wait_for_vaccinator',
                              'time_in_system']].describe()
```

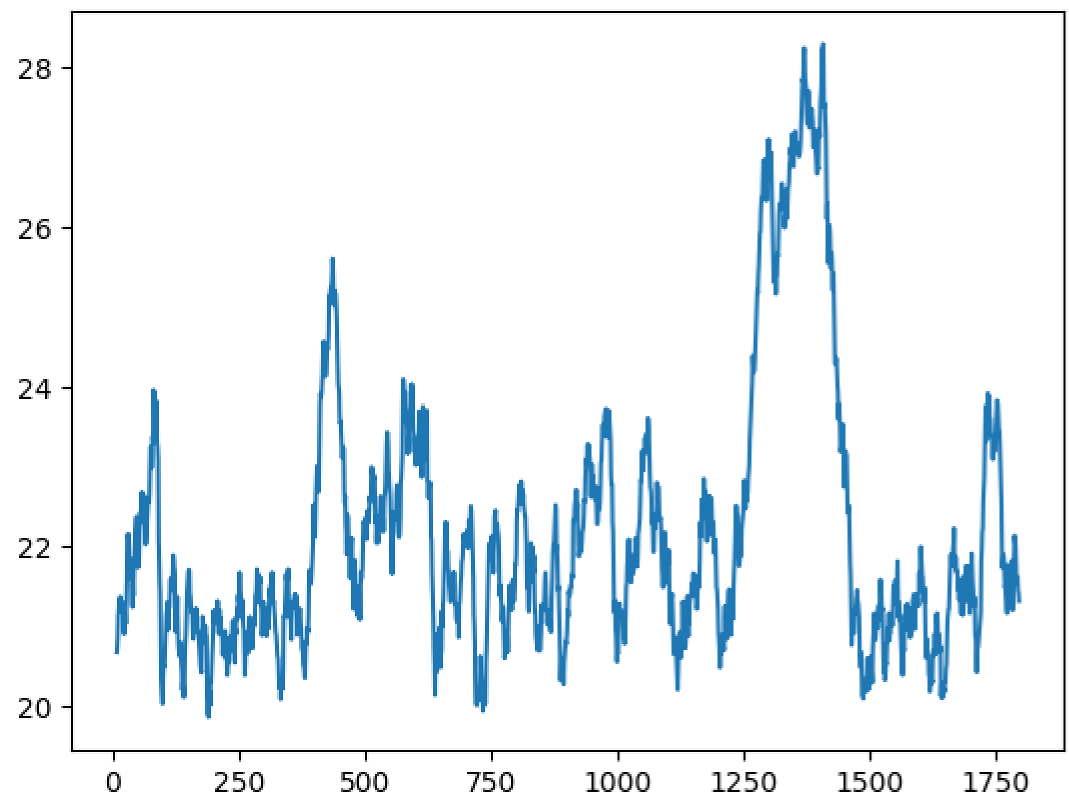
Out[13]:

	wait_for_greeter	wait_for_reg	wait_for_scheduler	wait_for_vaccinator	time_in_system
count	1798.0	1798.000000	887.000000	1798.000000	1798.000000
mean	0.0	0.116159	0.002736	1.303455	22.191459
std	0.0	0.335467	0.029372	1.717164	2.125317
min	0.0	0.000000	0.000000	0.000000	18.374674
25%	0.0	0.000000	0.000000	0.000000	20.671008
50%	0.0	0.000000	0.000000	0.734681	21.732092
75%	0.0	0.000000	0.000000	1.816202	23.255448
max	0.0	2.426057	0.525315	7.407274	31.207599

In [14]:

```
y = clinic_patient_log_df['time_in_system'].rolling(10, 10).mean()
plt.plot(y)
```

Out[14]: [<matplotlib.lines.Line2D at 0x1f2df208bb0>]



In [15]:

```
postvac_occupancy_df = pd.read_csv('postvac_occupancy_df.csv')
vac_occupancy_df = pd.read_csv('vac_occupancy_df.csv')
postvac_occupancy_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3597 entries, 0 to 3596
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   ts      3597 non-null   float64
 1   occ     3597 non-null   float64
dtypes: float64(2)
memory usage: 56.3 KB
```

In [16]:

```
postvac_occupancy_df.iloc[250:275,]
```

Out[16]:

	ts	occ
250	54.405398	52.0
251	54.608946	51.0
252	54.616138	50.0
253	54.668165	51.0
254	54.736020	50.0
255	55.001188	51.0
256	55.072893	50.0
257	55.155457	49.0
258	55.165267	48.0
259	55.512725	49.0
260	55.591244	48.0
261	55.741194	49.0
262	55.852800	48.0
263	55.924815	49.0
264	56.632630	50.0
265	56.848482	51.0
266	56.854441	50.0
267	57.229431	51.0
268	57.323785	50.0
269	57.361174	49.0
270	57.428900	50.0
271	57.437393	51.0
272	57.462332	52.0
273	57.703675	51.0
274	58.127201	50.0

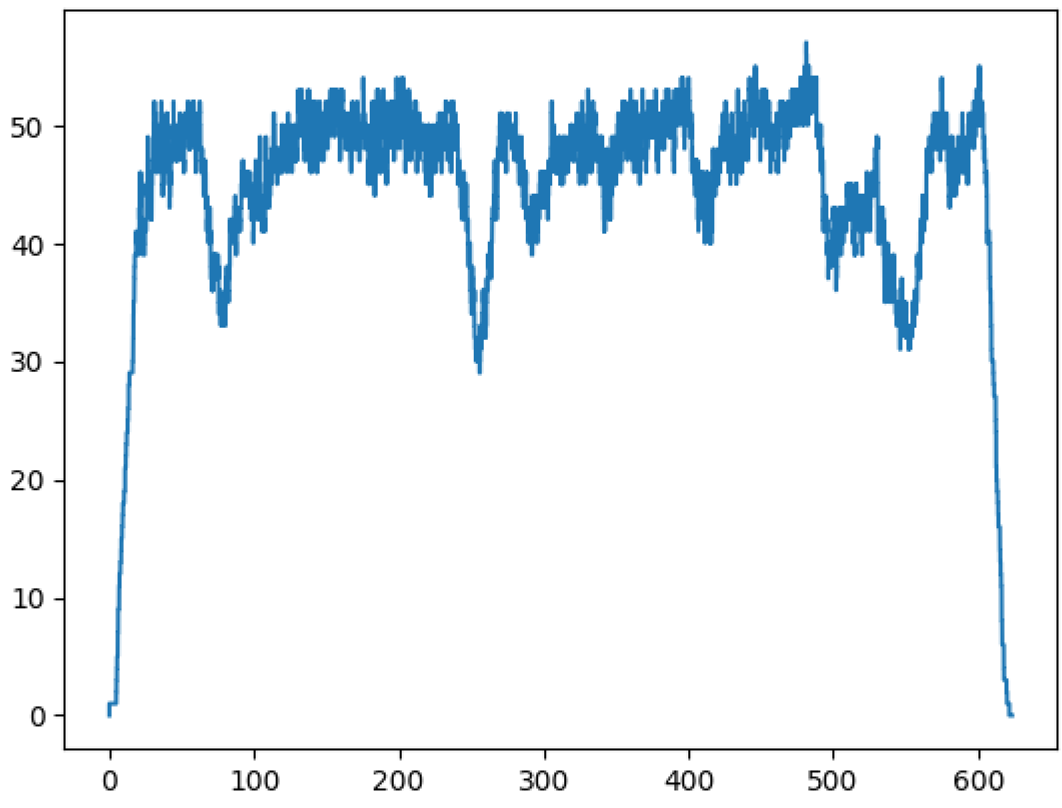
In [17]:
```
# Compute difference between ts[i] i and ts[i + 1]
# (in pandas this corresponds to periods=-1 in diff() function)
postvac_occupancy_df['occ_weight'] = -1 * postvac_occupancy_df['ts'].diff(periods=-
vac_occupancy_df['occ_weight'] = -1 * postvac_occupancy_df['ts'].diff(periods=-1)

# Need last occ_weight to compute weight for last row
last_weight = end_time - postvac_occupancy_df.iloc[-1, 0]
postvac_occupancy_df.fillna(last_weight, inplace=True)

last_weight = end_time - vac_occupancy_df.iloc[-1, 0]
vac_occupancy_df.fillna(last_weight, inplace=True)
```

In [18]:
```
np.average(a=postvac_occupancy_df['occ'], weights=postvac_occupancy_df['occ_weight'
```

Out[18]: 44.72164011074404

In [19]:
```
plt.step(postvac_occupancy_df['ts'], postvac_occupancy_df['occ'])
```

Out[19]: [<matplotlib.lines.Line2D at 0x1f2df26f730>]



In [20]:
```
available_capacity = end_time * num_vaccinators
available_capacity
```

Out[20]: 8105.845613090095

In [21]:
```
used_capacity = clinic_patient_log_df['vaccination_time'].sum()

vaccinator_utilization = used_capacity / available_capacity

print(f"Vaccinator utilization: {vaccinator_utilization:0.3f}")
```

```
Vaccinator utilization: 0.888
```

In [ ]:

```

```

Simulasi Klinik Kasus 4: patient flow modeling

I Wayan Sudiarta, Ph.D.

Menggunakan Ref. 1.

References:

1. https://bitsofanalytics.org/posts/simpy-getting-started-patflow-model/simpy-getting-started
2. https://bitsofanalytics.org/posts/simpy-oo-patflow-model/simpy-oo-patflow-model

In [1]:
```
# Import Modul Python
# Manipulasi Data
import numpy as np
import pandas as pd
from numpy.random import default_rng
from scipy import optimize

# Visualisasi
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image
from numpy.random import RandomState

# Simulasi
import simpy

# output inline
%matplotlib inline
```

Simple OB patient flow

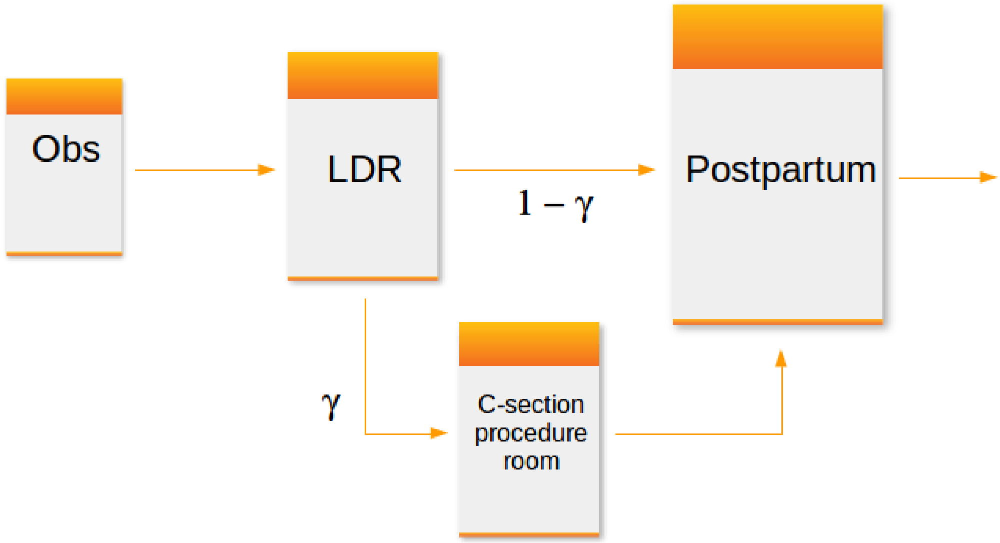
Kode diperoleh dari Ref. 1.

Model yang digunakan (Ref. 1):

- Generate arrivals via Poisson process
- Uses one Resource objects to model OBS, LDR, and PP.
- Arrival rates and mean lengths of stay hard coded as constants. Later versions will read these from input files.
- Additional functionality added to arrival generator (initial delay and arrival stop time).

In [2]:
```
Image("model-klinik-4.png", width=500) # gambar dari Ref. 1.
```

Out[2]:



In [3]:
```
from numpy.random import default_rng
rg = default_rng(seed=113366)
```

In [4]:
```
# Simulation parameters

ARR_RATE = 0.4
MEAN_LOS_OBS = 3
MEAN_LOS_LDR = 12
MEAN_LOS_PP = 48

CAPACITY_OBS = 2
CAPACITY_LDR = 6
CAPACITY_PP = 24

RNG_SEED = 6353
```

In [5]:
```
def patient_generator(env, arr_stream, arr_rate, initial_delay=0,
                      stoptime=simpy.core.Infinity, prng=RandomState(0)):
    """Generates patients according to a simple Poisson process

        Parameters
        ----------
        env : simpy.Environment
            the simulation environment
        arr_rate : float
            exponential arrival rate
        initial_delay: float (default 0)
            time before arrival generation should begin
        stoptime: float (default Infinity)
            time after which no arrivals are generated
        prng : RandomState object
            Seeded RandomState object for generating pseudo-random numbers.
            See https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.R

    """

    patients_created = 0
```

```python
    # Yield for the initial delay
    yield env.timeout(initial_delay)

    # Generate arrivals as long as simulation time is before stoptime
    while env.now < stoptime:

        iat = prng.exponential(1.0 / arr_rate)

        # Sample los distributions
        los_obs = prng.exponential(MEAN_LOS_OBS)
        los_ldr = prng.exponential(MEAN_LOS_LDR)
        los_pp = prng.exponential(MEAN_LOS_PP)


        # Create new patient process instance
        patients_created += 1
        obp = obpatient_flow(env, 'Patient{}'.format(patients_created),
                             los_obs=los_obs, los_ldr=los_ldr, los_pp=los_pp)

        env.process(obp)

        # Compute next interarrival time

        yield env.timeout(iat)
```

In [6]:

```python
def obpatient_flow(env, name, los_obs, los_ldr, los_pp):
    """Process function modeling how a patient flows through system.

        Parameters
        ----------
        env : simpy.Environment
            the simulation environment
        name : str
            process instance id
        los_obs : float
            length of stay in OBS unit
        los_ldr : float
            length of stay in LDR unit
        los_pp : float
            length of stay in PP unit
    """

    # Note the repetitive code and the use of separate request objects for each
    # stay in the different units.

    # OBS
    print("{} trying to get OBS at {:.4f}".format(name, env.now))
    bed_request_ts = env.now
    bed_request1 = obs_unit.request() # Request an OBS bed
    yield bed_request1
    print("{} entering OBS at {:.4f}".format(name, env.now))
    if env.now > bed_request_ts:
        print("{} waited {:.4f} time units for OBS bed".format(name, env.now-  bed_
    yield env.timeout(los_obs) # Stay in obs bed

    print("{} trying to get LDR at {:.4f}".format(name, env.now))
```

```
    bed_request_ts = env.now
    bed_request2 = ldr_unit.request()  # Request an LDR bed
    yield bed_request2

    # Got LDR bed, release OBS bed
    obs_unit.release(bed_request1)  # Release the OBS bed
    print("{} leaving OBS at {}".format(name, env.now))

    # LDR stay
    print("{} entering LDR at {:.4f}".format(name, env.now))
    if env.now > bed_request_ts:
        print("{} waited {:.4f} time units for LDR bed".format(name, env.now - bed_
    yield env.timeout(los_ldr) # Stay in LDR bed

    print("{} trying to get PP at {:.4f}".format(name, env.now))
    bed_request_ts = env.now
    bed_request3 = pp_unit.request()  # Request a PP bed
    yield bed_request3

    # Got PP bed, release LDR bed
    ldr_unit.release(bed_request2)  # Release the obs bed
    print("{} leaving LDR at {:.4f}".format(name, env.now))

    # PP stay
    print("{} entering PP at {:.4f}".format(name, env.now))
    if env.now > bed_request_ts:
        print("{} waited {:.4f} time units for PP bed".format(name, env.now - bed_r
    yield env.timeout(los_pp) # Stay in PP bed
    pp_unit.release(bed_request3)  # Release the PP bed

    print("{} leaving PP and system at {:.4f}".format(name, env.now))
```

In [7]:

```
prng = RandomState(RNG_SEED)

rho_obs = ARR_RATE * MEAN_LOS_OBS / CAPACITY_OBS
rho_ldr = ARR_RATE * MEAN_LOS_LDR / CAPACITY_LDR
rho_pp = ARR_RATE * MEAN_LOS_PP / CAPACITY_PP

# Initialize a simulation environment
env = simpy.Environment()

# Declare Resources to model all units
obs_unit = simpy.Resource(env, CAPACITY_OBS)
ldr_unit = simpy.Resource(env, CAPACITY_LDR)
pp_unit = simpy.Resource(env, CAPACITY_PP)
```

In [8]:

```
# Run the simulation for a while. Let's shut arrivals off after 50 time units.
runtime = 75
stop_arrivals = 50
env.process(patient_generator(env, "Type1", ARR_RATE, 0, stop_arrivals, prng))
env.run(until=runtime)
```

```
Patient1 trying to get OBS at 0.0000
Patient1 entering OBS at 0.0000
Patient1 trying to get LDR at 0.3475
Patient1 leaving OBS at 0.3474891089551544
Patient1 entering LDR at 0.3475
Patient2 trying to get OBS at 3.7668
Patient2 entering OBS at 3.7668
Patient3 trying to get OBS at 5.6439
Patient3 entering OBS at 5.6439
Patient1 trying to get PP at 6.1449
Patient1 leaving LDR at 6.1449
Patient1 entering PP at 6.1449
Patient4 trying to get OBS at 7.2625
Patient5 trying to get OBS at 8.9829
Patient3 trying to get LDR at 10.8140
Patient3 leaving OBS at 10.813985162144862
Patient3 entering LDR at 10.8140
Patient4 entering OBS at 10.8140
Patient4 waited 3.5515 time units for OBS bed
Patient4 trying to get LDR at 10.8690
Patient4 leaving OBS at 10.869006908518232
Patient4 entering LDR at 10.8690
Patient5 entering OBS at 10.8690
Patient5 waited 1.8861 time units for OBS bed
Patient4 trying to get PP at 11.4351
Patient4 leaving LDR at 11.4351
Patient4 entering PP at 11.4351
Patient2 trying to get LDR at 12.9060
Patient2 leaving OBS at 12.90600490911474
Patient2 entering LDR at 12.9060
Patient6 trying to get OBS at 16.8153
Patient6 entering OBS at 16.8153
Patient7 trying to get OBS at 17.8737
Patient8 trying to get OBS at 18.7157
Patient5 trying to get LDR at 18.8306
Patient5 leaving OBS at 18.830564537083035
Patient5 entering LDR at 18.8306
Patient7 entering OBS at 18.8306
Patient7 waited 0.9569 time units for OBS bed
Patient7 trying to get LDR at 19.0556
Patient7 leaving OBS at 19.05561983104953
Patient7 entering LDR at 19.0556
Patient8 entering OBS at 19.0556
Patient8 waited 0.3400 time units for OBS bed
Patient7 trying to get PP at 19.0901
Patient7 leaving LDR at 19.0901
Patient7 entering PP at 19.0901
Patient6 trying to get LDR at 19.4866
Patient6 leaving OBS at 19.48663625825889
Patient6 entering LDR at 19.4866
Patient9 trying to get OBS at 20.4913
Patient9 entering OBS at 20.4913
Patient8 trying to get LDR at 23.1856
Patient8 leaving OBS at 23.185576349161767
Patient8 entering LDR at 23.1856
Patient8 trying to get PP at 23.5282
```

```
Patient8 leaving LDR at 23.5282
Patient8 entering PP at 23.5282
Patient2 trying to get PP at 25.7847
Patient2 leaving LDR at 25.7847
Patient2 entering PP at 25.7847
Patient5 trying to get PP at 26.4507
Patient5 leaving LDR at 26.4507
Patient5 entering PP at 26.4507
Patient10 trying to get OBS at 26.8687
Patient10 entering OBS at 26.8687
Patient10 trying to get LDR at 27.6307
Patient10 leaving OBS at 27.63070937211375
Patient10 entering LDR at 27.6307
Patient11 trying to get OBS at 27.6876
Patient11 entering OBS at 27.6876
Patient6 trying to get PP at 28.3114
Patient6 leaving LDR at 28.3114
Patient6 entering PP at 28.3114
Patient9 trying to get LDR at 30.4208
Patient9 leaving OBS at 30.420840322655998
Patient9 entering LDR at 30.4208
Patient11 trying to get LDR at 30.4652
Patient11 leaving OBS at 30.465221449496795
Patient11 entering LDR at 30.4652
Patient11 trying to get PP at 30.5128
Patient11 leaving LDR at 30.5128
Patient11 entering PP at 30.5128
Patient12 trying to get OBS at 31.4872
Patient12 entering OBS at 31.4872
Patient13 trying to get OBS at 31.9981
Patient13 entering OBS at 31.9981
Patient12 trying to get LDR at 32.2156
Patient12 leaving OBS at 32.215634051591636
Patient12 entering LDR at 32.2156
Patient13 trying to get LDR at 32.5002
Patient13 leaving OBS at 32.500238016146994
Patient13 entering LDR at 32.5002
Patient14 trying to get OBS at 33.5398
Patient14 entering OBS at 33.5398
Patient7 leaving PP and system at 34.5316
Patient12 trying to get PP at 35.1766
Patient12 leaving LDR at 35.1766
Patient12 entering PP at 35.1766
Patient15 trying to get OBS at 35.6094
Patient15 entering OBS at 35.6094
Patient13 trying to get PP at 35.8941
Patient13 leaving LDR at 35.8941
Patient13 entering PP at 35.8941
Patient16 trying to get OBS at 36.5130
Patient17 trying to get OBS at 36.5369
Patient15 trying to get LDR at 36.6000
Patient15 leaving OBS at 36.59999969293714
Patient15 entering LDR at 36.6000
Patient16 entering OBS at 36.6000
Patient16 waited 0.0870 time units for OBS bed
Patient16 trying to get LDR at 37.4484
```

```
Patient16 leaving OBS at 37.44839668463982
Patient16 entering LDR at 37.4484
Patient17 entering OBS at 37.4484
Patient17 waited 0.9115 time units for OBS bed
Patient11 leaving PP and system at 38.1145
Patient18 trying to get OBS at 39.6783
Patient16 trying to get PP at 39.7335
Patient16 leaving LDR at 39.7335
Patient16 entering PP at 39.7335
Patient17 trying to get LDR at 39.9760
Patient17 leaving OBS at 39.97599817647182
Patient17 entering LDR at 39.9760
Patient18 entering OBS at 39.9760
Patient18 waited 0.2977 time units for OBS bed
Patient9 trying to get PP at 40.7127
Patient9 leaving LDR at 40.7127
Patient9 entering PP at 40.7127
Patient14 trying to get LDR at 41.0861
Patient14 leaving OBS at 41.086116528209494
Patient14 entering LDR at 41.0861
Patient19 trying to get OBS at 42.1910
Patient19 entering OBS at 42.1910
Patient10 trying to get PP at 42.5130
Patient10 leaving LDR at 42.5130
Patient10 entering PP at 42.5130
Patient18 trying to get LDR at 42.7172
Patient18 leaving OBS at 42.71719377930215
Patient18 entering LDR at 42.7172
Patient1 leaving PP and system at 42.8634
Patient17 trying to get PP at 44.1436
Patient17 leaving LDR at 44.1436
Patient17 entering PP at 44.1436
Patient18 trying to get PP at 44.4498
Patient18 leaving LDR at 44.4498
Patient18 entering PP at 44.4498
Patient15 trying to get PP at 44.6092
Patient15 leaving LDR at 44.6092
Patient15 entering PP at 44.6092
Patient20 trying to get OBS at 44.6790
Patient20 entering OBS at 44.6790
Patient3 trying to get PP at 44.9184
Patient3 leaving LDR at 44.9184
Patient3 entering PP at 44.9184
Patient20 trying to get LDR at 45.2709
Patient20 leaving OBS at 45.270857202928546
Patient20 entering LDR at 45.2709
Patient21 trying to get OBS at 46.6338
Patient21 entering OBS at 46.6338
Patient21 trying to get LDR at 47.8777
Patient21 leaving OBS at 47.8776934769442
Patient21 entering LDR at 47.8777
Patient19 trying to get LDR at 48.7520
Patient19 leaving OBS at 48.75197164898342
Patient19 entering LDR at 48.7520
Patient14 trying to get PP at 49.3856
Patient14 leaving LDR at 49.3856
```

```
Patient14 entering PP at 49.3856
Patient19 trying to get PP at 49.7737
Patient19 leaving LDR at 49.7737
Patient19 entering PP at 49.7737
Patient22 trying to get OBS at 49.9682
Patient22 entering OBS at 49.9682
Patient22 trying to get LDR at 50.6934
Patient22 leaving OBS at 50.69343889384862
Patient22 entering LDR at 50.6934
Patient21 trying to get PP at 50.6969
Patient21 leaving LDR at 50.6969
Patient21 entering PP at 50.6969
Patient20 trying to get PP at 55.7203
Patient20 leaving LDR at 55.7203
Patient20 entering PP at 55.7203
Patient16 leaving PP and system at 56.0439
Patient18 leaving PP and system at 57.0161
Patient12 leaving PP and system at 58.3965
Patient22 trying to get PP at 58.5375
Patient22 leaving LDR at 58.5375
Patient22 entering PP at 58.5375
Patient10 leaving PP and system at 59.8131
Patient9 leaving PP and system at 60.4868
Patient5 leaving PP and system at 63.4959
Patient17 leaving PP and system at 69.8364
Patient14 leaving PP and system at 69.9692
Patient15 leaving PP and system at 70.5547
Patient22 leaving PP and system at 72.2739
Patient20 leaving PP and system at 73.4076
```

In [ ]:

```

```

Simulasi Mobil Listrik dengan Simpy

I Wayan Sudiarta, Ph.D.

References:

1. https://simpy.readthedocs.io/en/latest/contents.html
2. https://simpy.readthedocs.io/en/latest/simpy_intro/shared_resources.html

Kode Python di bawah ini mengikuti Ref 2 (Dokumentasi Modul Simpy).

Bagian ini menyimulasikan mobil listrik datang ke Stasiun Pengisian Kendaraan Listrik Umum (SPKLU).

Parameter yang digunakan untuk simulasi:

- Sekali charge butuh interval waktu **CHARGE_INTERVAL** satuan waktu.
- Interval kedatangan mobil listrik yaitu konstan, **ARRIVAL_INTERVAL** satuan waktu.
- Terdapat sebanyak **CHARGING_UNITS** unit pengisian baterai.

Model Antrian D/D/2

In [8]:
```
import simpy
import numpy as np
```

In [31]:
```
CHARGE_INTERVAL = 5
ARRIVAL_INTERVAL = 2
CHARGING_UNITS = 3
```

In [32]:
```
# Definisikan fungsi generator untuk mobil listrik "car"
def car(env, name, bcs, driving_time, charge_duration):
    # Simulate driving to the BCS
    yield env.timeout(driving_time)

    # Request one of its charging spots
    print('%s arriving at %f' % (name, env.now))
    with bcs.request() as req:
        yield req

        # Charge the battery
        print('%s starting to charge at %s' % (name, env.now))
        yield env.timeout(charge_duration)
        print('%s leaving the bcs at %s' % (name, env.now))
```

In [33]:
```
# Inisialisasi simulasi
env = simpy.Environment(initial_time = 0)
bcs = simpy.Resource(env, capacity=CHARGING_UNITS)
```

In [34]:
```
# Process 20 cars
```

```python
waktu = 0.0
for i in range(20):
    waktu += np.random.exponential(ARRIVAL_INTERVAL)
    env.process(car(env, 'Car %d' % i, bcs, waktu, CHARGE_INTERVAL))
```

In [35]:
```python
# Jalankan "run" simulasi
env.run()
```

```
Car 0 arriving at 3.315333
Car 0 starting to charge at 3.3153330385305804
Car 1 arriving at 3.951845
Car 1 starting to charge at 3.9518446705006753
Car 2 arriving at 5.586485
Car 2 starting to charge at 5.586485109672431
Car 3 arriving at 5.684059
Car 4 arriving at 6.033369
Car 5 arriving at 7.375742
Car 6 arriving at 7.889483
Car 0 leaving the bcs at 8.31533303853058
Car 3 starting to charge at 8.31533303853058
Car 7 arriving at 8.389092
Car 1 leaving the bcs at 8.951844670500675
Car 4 starting to charge at 8.951844670500675
Car 2 leaving the bcs at 10.58648510967243
Car 5 starting to charge at 10.58648510967243
Car 8 arriving at 12.435562
Car 3 leaving the bcs at 13.31533303853058
Car 6 starting to charge at 13.31533303853058
Car 4 leaving the bcs at 13.951844670500675
Car 7 starting to charge at 13.951844670500675
Car 9 arriving at 14.547271
Car 10 arriving at 14.574801
Car 11 arriving at 15.509805
Car 5 leaving the bcs at 15.58648510967243
Car 8 starting to charge at 15.58648510967243
Car 6 leaving the bcs at 18.315333038530582
Car 9 starting to charge at 18.315333038530582
Car 12 arriving at 18.473902
Car 7 leaving the bcs at 18.951844670500677
Car 10 starting to charge at 18.951844670500677
Car 8 leaving the bcs at 20.58648510967243
Car 11 starting to charge at 20.58648510967243
Car 9 leaving the bcs at 23.315333038530582
Car 12 starting to charge at 23.315333038530582
Car 10 leaving the bcs at 23.951844670500677
Car 13 arriving at 24.447835
Car 13 starting to charge at 24.44783515316827
Car 11 leaving the bcs at 25.58648510967243
Car 14 arriving at 27.139374
Car 14 starting to charge at 27.139374120984733
Car 12 leaving the bcs at 28.315333038530582
Car 15 arriving at 28.534311
Car 15 starting to charge at 28.53431127432374
Car 13 leaving the bcs at 29.44783515316827
Car 14 leaving the bcs at 32.13937412098473
Car 16 arriving at 32.877139
Car 16 starting to charge at 32.87713904436833
Car 15 leaving the bcs at 33.53431127432374
Car 17 arriving at 34.916470
Car 17 starting to charge at 34.916470154382736
Car 18 arriving at 35.052920
Car 18 starting to charge at 35.05291982852727
Car 19 arriving at 35.756756
Car 16 leaving the bcs at 37.87713904436833
```

```
Car 19 starting to charge at 37.87713904436833
Car 17 leaving the bcs at 39.916470154382736
Car 18 leaving the bcs at 40.05291982852727
Car 19 leaving the bcs at 42.87713904436833
```

In [ ]:

```
```

In [ ]:

```
```

Simulasi Mobil Listrik - Simpy dan Simpy_helpers

I Wayan Sudiarta, Ph.D.

References:

1. https://github.com/bambielli/simpy_helpers

"The `simpy_helpers` package was written to make building simulations and collecting statistics about simulations using the Simpy framework simpler." Ref.1.

Ada empat class yang perlu diperhatikan:

1. Entity
2. Resource
3. Source
4. Stats

In [1]:
```
import simpy
from simpy_helpers import Entity, Resource, Source, Stats

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
# Parameter
CHARGE_INTERVAL = 5
ARRIVAL_INTERVAL = 2
CHARGING_UNITS = 2
```

In [3]:
```
## Here is the Entity Subclass
class Car(Entity):
    def process(self):
        yield self.wait_for_resource(charge_station)
        yield self.process_at_resource(charge_station)
        self.release_resource(charge_station)
```

In [4]:
```
# Here is the Resource subclass
class ChargeStation(Resource):
    def service_time(self, entity):
         return CHARGE_INTERVAL  # waktu dibutuhkan untuk charge
```

In [5]:
```
# Here is the Source subclass
class CarSource(Source):
    def interarrival_time(self):
        return ARRIVAL_INTERVAL # setiap 2 satuan, car arrive
```

```python
    def build_entity(self):
        # jenis mobil listrik - untuk tambahan atribut
        # digunakan untuk simulasi selanjutnya
        car_type = np.random.choice(["Honda", "Suzuki"], p=[0.5, 0.5])
        attributes = {
            "car_type": car_type
        }
        return Car(env, attributes)
```

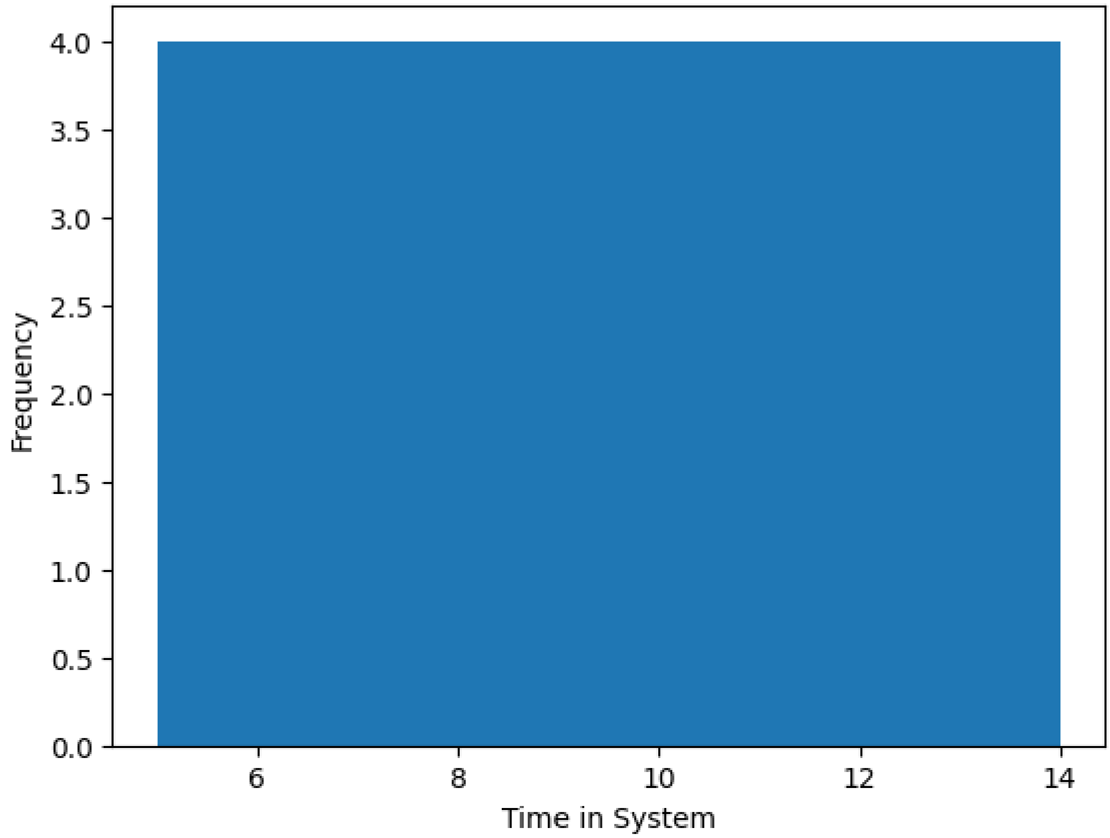
In [6]:
```python
# Now construct instances of Resource and Source.
np.random.seed(112233) # set seed

env = simpy.Environment(initial_time = 0)
charge_station = ChargeStation(env, capacity=CHARGING_UNITS) # configure 2
car_source = CarSource(env, number=20) # 20 cars
env.process(car_source.start(debug=False))
# if you want to see printed output for simulation, set debug=True

# Jalankan simulasi
env.run()
```

In [7]:
```python
# Now that the simulation has ended, we can use the Stats class to view summary sta
system_time = Stats.get_total_times()
print("total_time:", Stats.get_total_times())
print("total waiting_time:", Stats.get_waiting_times())
print("total processing_time:", Stats.get_processing_times(), "\n")

print("waiting time for charge_station resource", Stats.get_waiting_times(charge_st
print("processing time for charge_station resource", Stats.get_processing_times(cha
print("total time at charge_station resource", Stats.get_total_times(charge_station

print("charge_station queue size over time", Stats.queue_size_over_time(charge_stat
print("charge_station utilization over time", Stats.utilization_over_time(charge_st

print("entities that were not disposed", Stats.get_total_times(attributes={"dispose
```

```
total_time: [5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 13, 13, 14, 14]
total waiting_time: [0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9]
total processing_time: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5]

waiting time for charge_station resource [0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6,
6, 7, 7, 8, 8, 9, 9]
processing time for charge_station resource [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5]
total time at charge_station resource [5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11, 1
1, 12, 12, 13, 13, 14, 14]

charge_station queue size over time [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 4, 3, 4, 3, 4, 4,
3, 3, 2, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0]
charge_station utilization over time [0, 0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 0.5, 0.0]
entities that were not disposed []
```

In [8]:

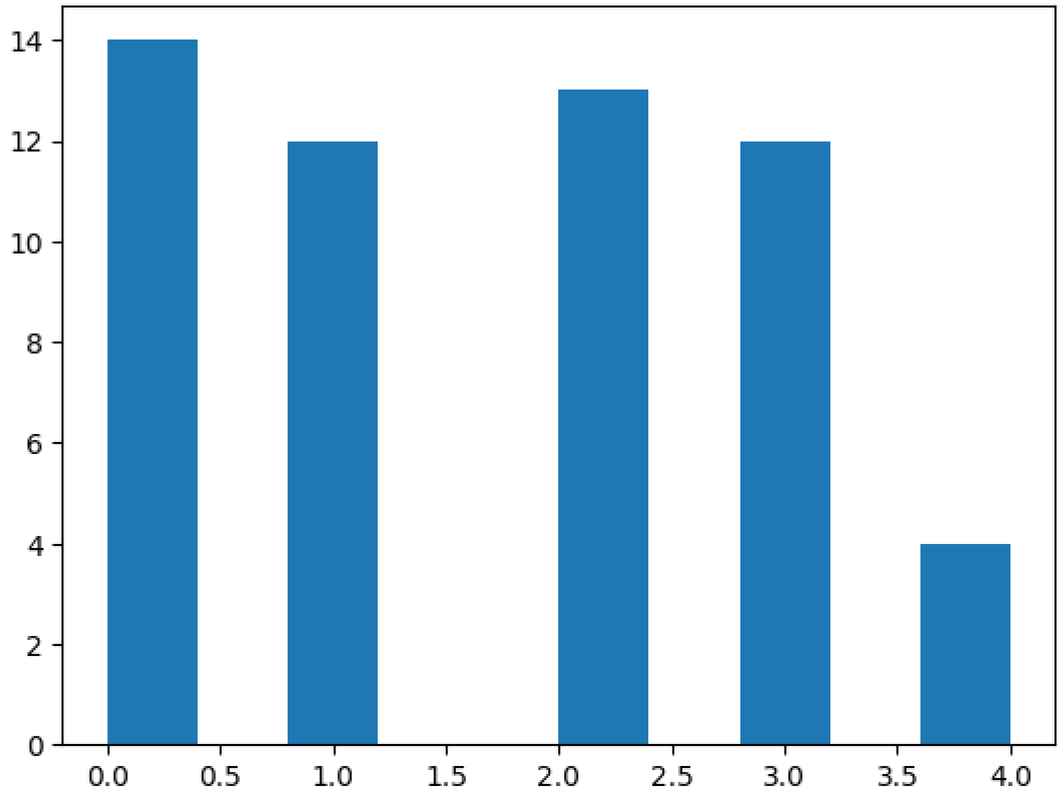
```
# we can use the return values from these statistics methods to create charts about
plt.hist(system_time,bins=5)
plt.ylabel('Frequency')
plt.xlabel('Time in System')
plt.show()
```



In [9]:

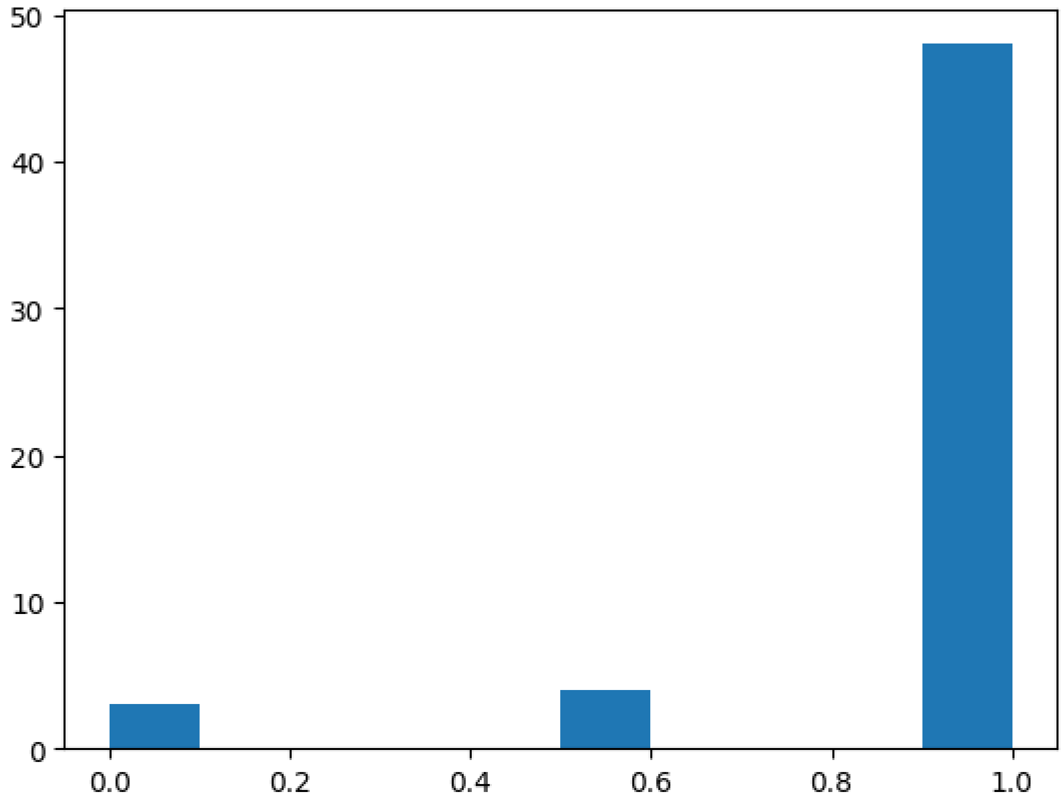
```
"""
Histogram of queue over time
"""
charge_station_queue = Stats.queue_size_over_time(charge_station)
plt.hist(charge_station_queue)
print(f"Average number in queue: {np.mean(charge_station_queue)}")
print(f"Max in queue: {np.max(charge_station_queue)}")

# clearly there are some problems with our current setup based on expected customer
```

```
Average number in queue: 1.6363636363636365
Max in queue: 4
```



In [10]:
```
charge_station_utilization = Stats.utilization_over_time(charge_station)
number_of_car_being_processed_over_time = Stats.number_being_processed_over_time(ch
print(f"number being processed at call center over time: {number_of_car_being_proce
plt.hist(charge_station_utilization)
print(f"Average utilization: {np.mean(charge_station_utilization)}")
print(f"Max utilization: {np.max(charge_station_utilization)}")
```

```
number being processed at call center over time: [0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 0]
Average utilization: 0.9090909090909091
Max utilization: 1.0
```



In [ ]:

SimPy Documentation

Release 4.0.1

Team SimPy

Apr 15, 2020

CHAPTER 4

Examples

All theory is grey. In this section, we present various practical examples that demonstrate how to uses SimPy's features.

Here is a list of examples grouped by the features they demonstrate.

4.1 Condition events

- *Bank Renege*
- *Movie Renege*

4.2 Interrupts

- *Machine Shop*

4.3 Monitoring

4.4 Resources: Container

- *Gas Station Refueling*

4.5 Resources: Preemptive Resource

- *Machine Shop*

4.6 Resources: Resource

- *Bank Renege*
- *Carwash*
- *Gas Station Refueling*
- *Movie Renege*

4.7 Resources: Store

- *Event Latency*
- *Process Communication*

4.8 Shared events

- *Movie Renege*

4.9 Waiting for other processes

- *Carwash*
- *Gas Station Refueling*

4.10 All examples

4.10.1 Bank Renege

Covers:

- Resources: Resource
- Condition events

A counter with a random service time and customers who renege. Based on the program bank08.py from TheBank tutorial of SimPy 2. (KGM)

This example models a bank counter and customers arriving at random times. Each customer has a certain patience. She waits to get to the counter until she's at the end of her tether. If she gets to the counter, she uses it for a while before releasing it.

New customers are created by the `source` process every few time steps.

```
"""
Bank renege example

Covers:

- Resources: Resource
```

(continues on next page)

(continued from previous page)

```
- Condition events

Scenario:
  A counter with a random service time and customers who renege. Based on the
  program bank08.py from TheBank tutorial of SimPy 2. (KGM)

"""
import random

import simpy


RANDOM_SEED = 42
NEW_CUSTOMERS = 5  # Total number of customers
INTERVAL_CUSTOMERS = 10.0  # Generate new customers roughly every x seconds
MIN_PATIENCE = 1  # Min. customer patience
MAX_PATIENCE = 3  # Max. customer patience


def source(env, number, interval, counter):
    """Source generates customers randomly"""
    for i in range(number):
        c = customer(env, 'Customer%02d' % i, counter, time_in_bank=12.0)
        env.process(c)
        t = random.expovariate(1.0 / interval)
        yield env.timeout(t)


def customer(env, name, counter, time_in_bank):
    """Customer arrives, is served and leaves."""
    arrive = env.now
    print('%7.4f %s: Here I am' % (arrive, name))

    with counter.request() as req:
        patience = random.uniform(MIN_PATIENCE, MAX_PATIENCE)
        # Wait for the counter or abort at the end of our tether
        results = yield req | env.timeout(patience)

        wait = env.now - arrive

        if req in results:
            # We got to the counter
            print('%7.4f %s: Waited %6.3f' % (env.now, name, wait))

            tib = random.expovariate(1.0 / time_in_bank)
            yield env.timeout(tib)
            print('%7.4f %s: Finished' % (env.now, name))

        else:
            # We reneged
            print('%7.4f %s: RENEGED after %6.3f' % (env.now, name, wait))


# Setup and start the simulation
print('Bank renege')
random.seed(RANDOM_SEED)
env = simpy.Environment()
```

(continues on next page)

(continued from previous page)

```
# Start processes and run
counter = simpy.Resource(env, capacity=1)
env.process(source(env, NEW_CUSTOMERS, INTERVAL_CUSTOMERS, counter))
env.run()
```

The simulation's output:

```
Bank renege
 0.0000 Customer00: Here I am
 0.0000 Customer00: Waited  0.000
 3.8595 Customer00: Finished
10.2006 Customer01: Here I am
10.2006 Customer01: Waited  0.000
12.7265 Customer02: Here I am
13.9003 Customer02: RENEGED after  1.174
23.7507 Customer01: Finished
34.9993 Customer03: Here I am
34.9993 Customer03: Waited  0.000
37.9599 Customer03: Finished
40.4798 Customer04: Here I am
40.4798 Customer04: Waited  0.000
43.1401 Customer04: Finished
```

4.10.2 Carwash

Covers:

- Waiting for other processes
- Resources: Resource

The *Carwash* example is a simulation of a carwash with a limited number of machines and a number of cars that arrive at the carwash to get cleaned.

The carwash uses a `Resource` to model the limited number of washing machines. It also defines a process for washing a car.

When a car arrives at the carwash, it requests a machine. Once it got one, it starts the carwash's *wash* processes and waits for it to finish. It finally releases the machine and leaves.

The cars are generated by a *setup* process. After creating an intial amount of cars it creates new *car* processes after a random time interval as long as the simulation continues.

```
"""
Carwash example.

Covers:

- Waiting for other processes
- Resources: Resource

Scenario:
  A carwash has a limited number of washing machines and defines
  a washing processes that takes some (random) time.

  Car processes arrive at the carwash at a random time. If one washing
```

(continues on next page)

(continued from previous page)

```
  machine is available, they start the washing process and wait for it
  to finish. If not, they wait until they an use one.

"""
import random

import simpy


RANDOM_SEED = 42
NUM_MACHINES = 2  # Number of machines in the carwash
WASHTIME = 5      # Minutes it takes to clean a car
T_INTER = 7       # Create a car every ~7 minutes
SIM_TIME = 20     # Simulation time in minutes


class Carwash(object):
    """A carwash has a limited number of machines (``NUM_MACHINES``) to
    clean cars in parallel.

    Cars have to request one of the machines. When they got one, they
    can start the washing processes and wait for it to finish (which
    takes ``washtime`` minutes).

    """
    def __init__(self, env, num_machines, washtime):
        self.env = env
        self.machine = simpy.Resource(env, num_machines)
        self.washtime = washtime

    def wash(self, car):
        """The washing processes. It takes a ``car`` processes and tries
        to clean it."""
        yield self.env.timeout(WASHTIME)
        print("Carwash removed %d%% of %s's dirt." %
              (random.randint(50, 99), car))


def car(env, name, cw):
    """The car process (each car has a ``name``) arrives at the carwash
    (``cw``) and requests a cleaning machine.

    It then starts the washing process, waits for it to finish and
    leaves to never come back ...

    """
    print('%s arrives at the carwash at %.2f.' % (name, env.now))
    with cw.machine.request() as request:
        yield request

        print('%s enters the carwash at %.2f.' % (name, env.now))
        yield env.process(cw.wash(name))

        print('%s leaves the carwash at %.2f.' % (name, env.now))


def setup(env, num_machines, washtime, t_inter):
```

(continues on next page)

(continued from previous page)

```
    """Create a carwash, a number of initial cars and keep creating cars
    approx. every ``t_inter`` minutes."""
    # Create the carwash
    carwash = Carwash(env, num_machines, washtime)

    # Create 4 initial cars
    for i in range(4):
        env.process(car(env, 'Car %d' % i, carwash))

    # Create more cars while the simulation is running
    while True:
        yield env.timeout(random.randint(t_inter - 2, t_inter + 2))
        i += 1
        env.process(car(env, 'Car %d' % i, carwash))


# Setup and start the simulation
print('Carwash')
print('Check out http://youtu.be/fXXmeP9TvBg while simulating ... ;-)')
random.seed(RANDOM_SEED)  # This helps reproducing the results

# Create an environment and start the setup process
env = simpy.Environment()
env.process(setup(env, NUM_MACHINES, WASHTIME, T_INTER))

# Execute!
env.run(until=SIM_TIME)
```

The simulation's output:

```
Carwash
Check out http://youtu.be/fXXmeP9TvBg while simulating ... ;-)
Car 0 arrives at the carwash at 0.00.
Car 1 arrives at the carwash at 0.00.
Car 2 arrives at the carwash at 0.00.
Car 3 arrives at the carwash at 0.00.
Car 0 enters the carwash at 0.00.
Car 1 enters the carwash at 0.00.
Car 4 arrives at the carwash at 5.00.
Carwash removed 97% of Car 0's dirt.
Carwash removed 67% of Car 1's dirt.
Car 0 leaves the carwash at 5.00.
Car 1 leaves the carwash at 5.00.
Car 2 enters the carwash at 5.00.
Car 3 enters the carwash at 5.00.
Car 5 arrives at the carwash at 10.00.
Carwash removed 64% of Car 2's dirt.
Carwash removed 58% of Car 3's dirt.
Car 2 leaves the carwash at 10.00.
Car 3 leaves the carwash at 10.00.
Car 4 enters the carwash at 10.00.
Car 5 enters the carwash at 10.00.
Carwash removed 97% of Car 4's dirt.
Carwash removed 56% of Car 5's dirt.
Car 4 leaves the carwash at 15.00.
Car 5 leaves the carwash at 15.00.
Car 6 arrives at the carwash at 16.00.
```

(continues on next page)

(continued from previous page)

```
Car 6 enters the carwash at 16.00.
```

4.10.3 Machine Shop

Covers:

- Interrupts
- Resources: PreemptiveResource

This example comprises a workshop with *n* identical machines. A stream of jobs (enough to keep the machines busy) arrives. Each machine breaks down periodically. Repairs are carried out by one repairman. The repairman has other, less important tasks to perform, too. Broken machines preempt theses tasks. The repairman continues them when he is done with the machine repair. The workshop works continuously.

A machine has two processes: *working* implements the actual behaviour of the machine (producing parts). *break_machine* periodically interrupts the *working* process to simulate the machine failure.

The repairman's other job is also a process (implemented by *other_job*). The repairman itself is a `PreemptiveResource` with a capacity of *1*. The machine repairing has a priority of *1*, while the other job has a priority of *2* (the smaller the number, the higher the priority).

```
"""
Machine shop example

Covers:

- Interrupts
- Resources: PreemptiveResource

Scenario:
  A workshop has *n* identical machines. A stream of jobs (enough to
  keep the machines busy) arrives. Each machine breaks down
  periodically. Repairs are carried out by one repairman. The repairman
  has other, less important tasks to perform, too. Broken machines
  preempt theses tasks. The repairman continues them when he is done
  with the machine repair. The workshop works continuously.

"""
import random

import simpy


RANDOM_SEED = 42
PT_MEAN = 10.0         # Avg. processing time in minutes
PT_SIGMA = 2.0         # Sigma of processing time
MTTF = 300.0           # Mean time to failure in minutes
BREAK_MEAN = 1 / MTTF  # Param. for expovariate distribution
REPAIR_TIME = 30.0     # Time it takes to repair a machine in minutes
JOB_DURATION = 30.0    # Duration of other jobs in minutes
NUM_MACHINES = 10      # Number of machines in the machine shop
WEEKS = 4              # Simulation time in weeks
SIM_TIME = WEEKS * 7 * 24 * 60  # Simulation time in minutes
```

(continues on next page)

(continued from previous page)

```
def time_per_part():
    """Return actual processing time for a concrete part."""
    return random.normalvariate(PT_MEAN, PT_SIGMA)


def time_to_failure():
    """Return time until next failure for a machine."""
    return random.expovariate(BREAK_MEAN)


class Machine(object):
    """A machine produces parts and my get broken every now and then.

    If it breaks, it requests a *repairman* and continues the production
    after the it is repaired.

    A machine has a *name* and a numberof *parts_made* thus far.

    """
    def __init__(self, env, name, repairman):
        self.env = env
        self.name = name
        self.parts_made = 0
        self.broken = False

        # Start "working" and "break_machine" processes for this machine.
        self.process = env.process(self.working(repairman))
        env.process(self.break_machine())

    def working(self, repairman):
        """Produce parts as long as the simulation runs.

        While making a part, the machine may break multiple times.
        Request a repairman when this happens.

        """
        while True:
            # Start making a new part
            done_in = time_per_part()
            while done_in:
                try:
                    # Working on the part
                    start = self.env.now
                    yield self.env.timeout(done_in)
                    done_in = 0  # Set to 0 to exit while loop.

                except simpy.Interrupt:
                    self.broken = True
                    done_in -= self.env.now - start  # How much time left?

                    # Request a repairman. This will preempt its "other_job".
                    with repairman.request(priority=1) as req:
                        yield req
                        yield self.env.timeout(REPAIR_TIME)

                    self.broken = False
```

(continues on next page)

(continued from previous page)

```
            # Part is done.
            self.parts_made += 1

    def break_machine(self):
        """Break the machine every now and then."""
        while True:
            yield self.env.timeout(time_to_failure())
            if not self.broken:
                # Only break the machine if it is currently working.
                self.process.interrupt()


def other_jobs(env, repairman):
    """The repairman's other (unimportant) job."""
    while True:
        # Start a new job
        done_in = JOB_DURATION
        while done_in:
            # Retry the job until it is done.
            # It's priority is lower than that of machine repairs.
            with repairman.request(priority=2) as req:
                yield req
                try:
                    start = env.now
                    yield env.timeout(done_in)
                    done_in = 0
                except simpy.Interrupt:
                    done_in -= env.now - start


# Setup and start the simulation
print('Machine shop')
random.seed(RANDOM_SEED)  # This helps reproducing the results

# Create an environment and start the setup process
env = simpy.Environment()
repairman = simpy.PreemptiveResource(env, capacity=1)
machines = [Machine(env, 'Machine %d' % i, repairman)
            for i in range(NUM_MACHINES)]
env.process(other_jobs(env, repairman))

# Execute!
env.run(until=SIM_TIME)

# Analyis/results
print('Machine shop results after %s weeks' % WEEKS)
for machine in machines:
    print('%s made %d parts.' % (machine.name, machine.parts_made))
```

The simulation's output:

```
Machine shop
Machine shop results after 4 weeks
Machine 0 made 3251 parts.
Machine 1 made 3273 parts.
Machine 2 made 3242 parts.
Machine 3 made 3343 parts.
```

(continues on next page)

(continued from previous page)

```
Machine 4 made 3387 parts.
Machine 5 made 3244 parts.
Machine 6 made 3269 parts.
Machine 7 made 3185 parts.
Machine 8 made 3302 parts.
Machine 9 made 3279 parts.
```

4.10.4 Movie Renege

Covers:

- Resources: Resource
- Condition events
- Shared events

This examples models a movie theater with one ticket counter selling tickets for three movies (next show only). People arrive at random times and try to buy a random number (1–6) of tickets for a random movie. When a movie is sold out, all people waiting to buy a ticket for that movie renege (leave the queue).

The movie theater is just a container for all the related data (movies, the counter, tickets left, collected data, . . .). The counter is a `Resource` with a capacity of one.

The *moviegoer* process starts waiting until either it's his turn (it acquires the counter resource) or until the *sold out* signal is triggered. If the latter is the case it reneges (leaves the queue). If it gets to the counter, it tries to buy some tickets. This might not be successful, e.g. if the process tries to buy 5 tickets but only 3 are left. If less than two tickets are left after the ticket purchase, the *sold out* signal is triggered.

Moviegoers are generated by the *customer arrivals* process. It also chooses a movie and the number of tickets for the moviegoer.

```
"""
Movie renege example

Covers:

- Resources: Resource
- Condition events
- Shared events

Scenario:
  A movie theatre has one ticket counter selling tickets for three
  movies (next show only). When a movie is sold out, all people waiting
  to buy tickets for that movie renege (leave queue).

"""
import collections
import random

import simpy


RANDOM_SEED = 42
TICKETS = 50  # Number of tickets per movie
SIM_TIME = 120  # Simulate until

```

(continues on next page)

(continued from previous page)

```
def moviegoer(env, movie, num_tickets, theater):
    """A moviegoer tries to by a number of tickets (*num_tickets*) for
    a certain *movie* in a *theater*.

    If the movie becomes sold out, she leaves the theater. If she gets
    to the counter, she tries to buy a number of tickets. If not enough
    tickets are left, she argues with the teller and leaves.

    If at most one ticket is left after the moviegoer bought her
    tickets, the *sold out* event for this movie is triggered causing
    all remaining moviegoers to leave.

    """
    with theater.counter.request() as my_turn:
        # Wait until its our turn or until the movie is sold out
        result = yield my_turn | theater.sold_out[movie]

        # Check if it's our turn or if movie is sold out
        if my_turn not in result:
            theater.num_renegers[movie] += 1
            return

        # Check if enough tickets left.
        if theater.available[movie] < num_tickets:
            # Moviegoer leaves after some discussion
            yield env.timeout(0.5)
            return

        # Buy tickets
        theater.available[movie] -= num_tickets
        if theater.available[movie] < 2:
            # Trigger the "sold out" event for the movie
            theater.sold_out[movie].succeed()
            theater.when_sold_out[movie] = env.now
            theater.available[movie] = 0
        yield env.timeout(1)


def customer_arrivals(env, theater):
    """Create new *moviegoers* until the sim time reaches 120."""
    while True:
        yield env.timeout(random.expovariate(1 / 0.5))

        movie = random.choice(theater.movies)
        num_tickets = random.randint(1, 6)
        if theater.available[movie]:
            env.process(moviegoer(env, movie, num_tickets, theater))


Theater = collections.namedtuple('Theater', 'counter, movies, available, '
                                            'sold_out, when_sold_out, '
                                            'num_renegers')


# Setup and start the simulation
print('Movie renege')
```

(continues on next page)

(continued from previous page)

```
random.seed(RANDOM_SEED)
env = simpy.Environment()

# Create movie theater
counter = simpy.Resource(env, capacity=1)
movies = ['Python Unchained', 'Kill Process', 'Pulp Implementation']
available = {movie: TICKETS for movie in movies}
sold_out = {movie: env.event() for movie in movies}
when_sold_out = {movie: None for movie in movies}
num_renegers = {movie: 0 for movie in movies}
theater = Theater(counter, movies, available, sold_out, when_sold_out,
                  num_renegers)

# Start process and run
env.process(customer_arrivals(env, theater))
env.run(until=SIM_TIME)

# Analysis/results
for movie in movies:
    if theater.sold_out[movie]:
        print('Movie "%s" sold out %.1f minutes after ticket counter '
              'opening.' % (movie, theater.when_sold_out[movie]))
        print('  Number of people leaving queue when film sold out: %s' %
              theater.num_renegers[movie])
```

The simulation's output:

```
Movie renege
Movie "Python Unchained" sold out 38.0 minutes after ticket counter opening.
  Number of people leaving queue when film sold out: 16
Movie "Kill Process" sold out 43.0 minutes after ticket counter opening.
  Number of people leaving queue when film sold out: 5
Movie "Pulp Implementation" sold out 28.0 minutes after ticket counter opening.
  Number of people leaving queue when film sold out: 5
```

4.10.5 Gas Station Refueling

Covers:

- Resources: Resource
- Resources: Container
- Waiting for other processes

This examples models a gas station and cars that arrive at the station for refueling.

The gas station has a limited number of fuel pumps and a fuel tank that is shared between the fuel pumps. The gas station is thus modeled as `Resource`. The shared fuel tank is modeled with a `Container`.

Vehicles arriving at the gas station first request a fuel pump from the station. Once they acquire one, they try to take the desired amount of fuel from the fuel pump. They leave when they are done.

The gas stations fuel level is regularly monitored by *gas station control*. When the level drops below a certain threshold, a *tank truck* is called to refuel the gas station itself.

```
"""
Gas Station Refueling example

Covers:

- Resources: Resource
- Resources: Container
- Waiting for other processes

Scenario:
  A gas station has a limited number of gas pumps that share a common
  fuel reservoir. Cars randomly arrive at the gas station, request one
  of the fuel pumps and start refueling from that reservoir.

  A gas station control process observes the gas station's fuel level
  and calls a tank truck for refueling if the station's level drops
  below a threshold.

"""
import itertools
import random

import simpy


RANDOM_SEED = 42
GAS_STATION_SIZE = 200      # liters
THRESHOLD = 10              # Threshold for calling the tank truck (in %)
FUEL_TANK_SIZE = 50         # liters
FUEL_TANK_LEVEL = [5, 25]   # Min/max levels of fuel tanks (in liters)
REFUELING_SPEED = 2         # liters / second
TANK_TRUCK_TIME = 300       # Seconds it takes the tank truck to arrive
T_INTER = [30, 300]         # Create a car every [min, max] seconds
SIM_TIME = 1000             # Simulation time in seconds


def car(name, env, gas_station, fuel_pump):
    """A car arrives at the gas station for refueling.

    It requests one of the gas station's fuel pumps and tries to get the
    desired amount of gas from it. If the stations reservoir is
    depleted, the car has to wait for the tank truck to arrive.

    """
    fuel_tank_level = random.randint(*FUEL_TANK_LEVEL)
    print('%s arriving at gas station at %.1f' % (name, env.now))
    with gas_station.request() as req:
        start = env.now
        # Request one of the gas pumps
        yield req

        # Get the required amount of fuel
        liters_required = FUEL_TANK_SIZE - fuel_tank_level
        yield fuel_pump.get(liters_required)

        # The "actual" refueling process takes some time
        yield env.timeout(liters_required / REFUELING_SPEED)
```

(continues on next page)

(continued from previous page)

```
        print('%s finished refueling in %.1f seconds.' % (name,
                                                          env.now - start))


def gas_station_control(env, fuel_pump):
    """Periodically check the level of the *fuel_pump* and call the tank
    truck if the level falls below a threshold."""
    while True:
        if fuel_pump.level / fuel_pump.capacity * 100 < THRESHOLD:
            # We need to call the tank truck now!
            print('Calling tank truck at %d' % env.now)
            # Wait for the tank truck to arrive and refuel the station
            yield env.process(tank_truck(env, fuel_pump))

        yield env.timeout(10)  # Check every 10 seconds


def tank_truck(env, fuel_pump):
    """Arrives at the gas station after a certain delay and refuels it."""
    yield env.timeout(TANK_TRUCK_TIME)
    print('Tank truck arriving at time %d' % env.now)
    ammount = fuel_pump.capacity - fuel_pump.level
    print('Tank truck refuelling %.1f liters.' % ammount)
    yield fuel_pump.put(ammount)


def car_generator(env, gas_station, fuel_pump):
    """Generate new cars that arrive at the gas station."""
    for i in itertools.count():
        yield env.timeout(random.randint(*T_INTER))
        env.process(car('Car %d' % i, env, gas_station, fuel_pump))


# Setup and start the simulation
print('Gas Station refuelling')
random.seed(RANDOM_SEED)

# Create environment and start processes
env = simpy.Environment()
gas_station = simpy.Resource(env, 2)
fuel_pump = simpy.Container(env, GAS_STATION_SIZE, init=GAS_STATION_SIZE)
env.process(gas_station_control(env, fuel_pump))
env.process(car_generator(env, gas_station, fuel_pump))

# Execute!
env.run(until=SIM_TIME)
```

The simulation's output:

```
Gas Station refuelling
Car 0 arriving at gas station at 87.0
Car 0 finished refueling in 18.5 seconds.
Car 1 arriving at gas station at 129.0
Car 1 finished refueling in 19.0 seconds.
Car 2 arriving at gas station at 284.0
Car 2 finished refueling in 21.0 seconds.
```

(continues on next page)

(continued from previous page)

```
Car 3 arriving at gas station at 385.0
Car 3 finished refueling in 13.5 seconds.
Car 4 arriving at gas station at 459.0
Calling tank truck at 460
Car 4 finished refueling in 22.0 seconds.
Car 5 arriving at gas station at 705.0
Car 6 arriving at gas station at 750.0
Tank truck arriving at time 760
Tank truck refuelling 188.0 liters.
Car 6 finished refueling in 29.0 seconds.
Car 5 finished refueling in 76.5 seconds.
Car 7 arriving at gas station at 891.0
Car 7 finished refueling in 13.0 seconds.
```

4.10.6 Process Communication

Covers:

- Resources: Store

This example shows how to interconnect simulation model elements together using “resources.Store” for one-to-one, and many-to-one asynchronous processes. For one-to-many a simple BroadCastPipe class is constructed from Store.

When Useful: When a consumer process does not always wait on a generating process and these processes run asynchronously. This example shows how to create a buffer and also tell is the consumer process was late yielding to the event from a generating process.

This is also useful when some information needs to be broadcast to many receiving processes

Finally, using pipes can simplify how processes are interconnected to each other in a simulation model.

Example By: Keith Smith

```
"""
Process communication example

Covers:

- Resources: Store

Scenario:
  This example shows how to interconnect simulation model elements
  together using :class:`~simpy.resources.store.Store` for one-to-one,
  and many-to-one asynchronous processes. For one-to-many a simple
  BroadCastPipe class is constructed from Store.

When Useful:
  When a consumer process does not always wait on a generating process
  and these processes run asynchronously. This example shows how to
  create a buffer and also tell is the consumer process was late
  yielding to the event from a generating process.

  This is also useful when some information needs to be broadcast to
  many receiving processes

  Finally, using pipes can simplify how processes are interconnected to
  each other in a simulation model.
```

(continues on next page)

(continued from previous page)

```
Example By:
  Keith Smith

"""
import random

import simpy


RANDOM_SEED = 42
SIM_TIME = 100


class BroadcastPipe(object):
    """A Broadcast pipe that allows one process to send messages to many.

    This construct is useful when message consumers are running at
    different rates than message generators and provides an event
    buffering to the consuming processes.

    The parameters are used to create a new
    :class:`~simpy.resources.store.Store` instance each time
    :meth:`get_output_conn()` is called.

    """
    def __init__(self, env, capacity=simpy.core.Infinity):
        self.env = env
        self.capacity = capacity
        self.pipes = []

    def put(self, value):
        """Broadcast a *value* to all receivers."""
        if not self.pipes:
            raise RuntimeError('There are no output pipes.')
        events = [store.put(value) for store in self.pipes]
        return self.env.all_of(events)  # Condition event for all "events"

    def get_output_conn(self):
        """Get a new output connection for this broadcast pipe.

        The return value is a :class:`~simpy.resources.store.Store`.

        """
        pipe = simpy.Store(self.env, capacity=self.capacity)
        self.pipes.append(pipe)
        return pipe


def message_generator(name, env, out_pipe):
    """A process which randomly generates messages."""
    while True:
        # wait for next transmission
        yield env.timeout(random.randint(6, 10))

        # messages are time stamped to later check if the consumer was
        # late getting them.  Note, using event.triggered to do this may
```

(continues on next page)

(continued from previous page)

```
        # result in failure due to FIFO nature of simulation yields.
        # (i.e. if at the same env.now, message_generator puts a message
        # in the pipe first and then message_consumer gets from pipe,
        # the event.triggered will be True in the other order it will be
        # False
        msg = (env.now, '%s says hello at %d' % (name, env.now))
        out_pipe.put(msg)


def message_consumer(name, env, in_pipe):
    """A process which consumes messages."""
    while True:
        # Get event for message pipe
        msg = yield in_pipe.get()

        if msg[0] < env.now:
            # if message was already put into pipe, then
            # message_consumer was late getting to it. Depending on what
            # is being modeled this, may, or may not have some
            # significance
            print('LATE Getting Message: at time %d: %s received message: %s' %
                    (env.now, name, msg[1]))

        else:
            # message_consumer is synchronized with message_generator
            print('at time %d: %s received message: %s.' %
                    (env.now, name, msg[1]))

        # Process does some other work, which may result in missing messages
        yield env.timeout(random.randint(4, 8))


# Setup and start the simulation
print('Process communication')
random.seed(RANDOM_SEED)
env = simpy.Environment()

# For one-to-one or many-to-one type pipes, use Store
pipe = simpy.Store(env)
env.process(message_generator('Generator A', env, pipe))
env.process(message_consumer('Consumer A', env, pipe))

print('\nOne-to-one pipe communication\n')
env.run(until=SIM_TIME)

# For one-to many use BroadcastPipe
# (Note: could also be used for one-to-one,many-to-one or many-to-many)
env = simpy.Environment()
bc_pipe = BroadcastPipe(env)

env.process(message_generator('Generator A', env, bc_pipe))
env.process(message_consumer('Consumer A', env, bc_pipe.get_output_conn()))
env.process(message_consumer('Consumer B', env, bc_pipe.get_output_conn()))

print('\nOne-to-many pipe communication\n')
env.run(until=SIM_TIME)
```

The simulation's output:

```
Process communication

One-to-one pipe communication

at time 6: Consumer A received message: Generator A says hello at 6.
at time 12: Consumer A received message: Generator A says hello at 12.
at time 19: Consumer A received message: Generator A says hello at 19.
at time 26: Consumer A received message: Generator A says hello at 26.
at time 36: Consumer A received message: Generator A says hello at 36.
at time 46: Consumer A received message: Generator A says hello at 46.
at time 52: Consumer A received message: Generator A says hello at 52.
at time 58: Consumer A received message: Generator A says hello at 58.
LATE Getting Message: at time 66: Consumer A received message: Generator A says hello␣
↪at 65
at time 75: Consumer A received message: Generator A says hello at 75.
at time 85: Consumer A received message: Generator A says hello at 85.
at time 95: Consumer A received message: Generator A says hello at 95.

One-to-many pipe communication

at time 10: Consumer A received message: Generator A says hello at 10.
at time 10: Consumer B received message: Generator A says hello at 10.
at time 18: Consumer A received message: Generator A says hello at 18.
at time 18: Consumer B received message: Generator A says hello at 18.
at time 27: Consumer A received message: Generator A says hello at 27.
at time 27: Consumer B received message: Generator A says hello at 27.
at time 34: Consumer A received message: Generator A says hello at 34.
at time 34: Consumer B received message: Generator A says hello at 34.
at time 40: Consumer A received message: Generator A says hello at 40.
LATE Getting Message: at time 41: Consumer B received message: Generator A says hello␣
↪at 40
at time 46: Consumer A received message: Generator A says hello at 46.
LATE Getting Message: at time 47: Consumer B received message: Generator A says hello␣
↪at 46
at time 56: Consumer A received message: Generator A says hello at 56.
at time 56: Consumer B received message: Generator A says hello at 56.
at time 65: Consumer A received message: Generator A says hello at 65.
at time 65: Consumer B received message: Generator A says hello at 65.
at time 74: Consumer A received message: Generator A says hello at 74.
at time 74: Consumer B received message: Generator A says hello at 74.
at time 82: Consumer A received message: Generator A says hello at 82.
at time 82: Consumer B received message: Generator A says hello at 82.
at time 92: Consumer A received message: Generator A says hello at 92.
at time 92: Consumer B received message: Generator A says hello at 92.
at time 98: Consumer B received message: Generator A says hello at 98.
at time 98: Consumer A received message: Generator A says hello at 98.
```

4.10.7 Event Latency

Covers:

- Resources: Store

This example shows how to separate the time delay of events between processes from the processes themselves.

When Useful: When modeling physical things such as cables, RF propagation, etc. it better encapsulation to keep this propagation mechanism outside of the sending and receiving processes.

Can also be used to interconnect processes sending messages

Example by: Keith Smith

```
"""
Event Latency example

Covers:

- Resources: Store

Scenario:
  This example shows how to separate the time delay of events between
  processes from the processes themselves.

When Useful:
  When modeling physical things such as cables, RF propagation, etc.  it
  better encapsulation to keep this propagation mechanism outside of the
  sending and receiving processes.

  Can also be used to interconnect processes sending messages

Example by:
  Keith Smith

"""
import simpy


SIM_DURATION = 100


class Cable(object):
    """This class represents the propagation through a cable."""
    def __init__(self, env, delay):
        self.env = env
        self.delay = delay
        self.store = simpy.Store(env)

    def latency(self, value):
        yield self.env.timeout(self.delay)
        self.store.put(value)

    def put(self, value):
        self.env.process(self.latency(value))

    def get(self):
        return self.store.get()


def sender(env, cable):
    """A process which randomly generates messages."""
    while True:
        # wait for next transmission
        yield env.timeout(5)
        cable.put('Sender sent this at %d' % env.now)
```

(continues on next page)

(continued from previous page)

```
def receiver(env, cable):
    """A process which consumes messages."""
    while True:
        # Get event for message pipe
        msg = yield cable.get()
        print('Received this at %d while %s' % (env.now, msg))


# Setup and start the simulation
print('Event Latency')
env = simpy.Environment()

cable = Cable(env, 10)
env.process(sender(env, cable))
env.process(receiver(env, cable))

env.run(until=SIM_DURATION)
```

The simulation's output:

```
Event Latency
Received this at 15 while Sender sent this at 5
Received this at 20 while Sender sent this at 10
Received this at 25 while Sender sent this at 15
Received this at 30 while Sender sent this at 20
Received this at 35 while Sender sent this at 25
Received this at 40 while Sender sent this at 30
Received this at 45 while Sender sent this at 35
Received this at 50 while Sender sent this at 40
Received this at 55 while Sender sent this at 45
Received this at 60 while Sender sent this at 50
Received this at 65 while Sender sent this at 55
Received this at 70 while Sender sent this at 60
Received this at 75 while Sender sent this at 65
Received this at 80 while Sender sent this at 70
Received this at 85 while Sender sent this at 75
Received this at 90 while Sender sent this at 80
Received this at 95 while Sender sent this at 85
```

You have ideas for better examples? Please send them to our mailing list or make a pull request on GitLab.