Question 13.2
In this problem you, can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with $\lambda_1 = 5$ per minute (i.e., mean interarrival rate $\mu_1 = 0.2$ minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate $\mu_2 = 0.75$ minutes. [Hint: model them as one block that has more than one resource.] After that, the passengers are assigned to the shortest of the several personal-check queues, where they go through the personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute).
Use the Arena software (PC users) or Python with SimPy (PC or Mac users) to build a simulation of the system, and then vary the number of ID/boarding-pass checkers and personal-check queues to determine how many are needed to keep average wait times below 15 minutes. [If you're using SimPy, or if you have access to a non-student version of Arena, you can use $\lambda_1 = 50$ to simulate a busier airport.]

<My Python code for the simulation>

```python
import simpy
import random
import numpy as np
import pandas as pd
import plotly.express as px
```

```python
class SecurityCheckpoint:
    def __init__(self, env, id_stations, scan_stations, id_check_server_time, scan_time_range):
        self.env = env
        self.id_check = simpy.Resource(env, id_stations)
        self.scan = simpy.Resource(env, scan_stations)
        self.id_check_server_time = id_check_server_time
        self.scan_time_range = scan_time_range

    def check_id(self, traveler_id):
        id_check_duration = random.expovariate(1 / self.id_check_server_time)
        yield self.env.timeout(id_check_duration)
        # print(f'A traveler {traveler_id} leaves the id check station at {env.now:.2f}.')

    def scan_traveler(self, traveler_id):
        scan_duration = random.uniform(self.scan_time_range[0], self.scan_time_range[1])
        yield self.env.timeout(scan_duration)
        # print(f'A traveler {traveler_id} leaves the scanning station at {env.now:.2f}.')

def traveler(env, checkpoint, traveler_id):
    arrival_time = env.now

    with checkpoint.id_check.request() as request:
        yield request  # a traveler enters id check station
        yield env.process(checkpoint.check_id(traveler_id))  # the traveler leaves id check station

    with checkpoint.scan.request() as request:
        yield request  # a traveler enters scanning station
        yield env.process(checkpoint.scan_traveler(traveler_id))  # the traveler leaves id check station
    # print(f"now ({env.now}) - arrival_time ({arrival_time}) = {env.now - arrival_time}")
    wait_durations.append(env.now - arrival_time)


def security_checkpoint_operation(env, id_stations, scan_stations, id_check_server_time, scan_time_range):
    checkpoint = SecurityCheckpoint(env, id_stations, scan_stations, id_check_server_time, scan_time_range)

    traveler_id = 1

    # Create 1 initial traveler
    for _ in range(1):
        env.process(traveler(env, checkpoint, f"{traveler_id}"))

    while True:
        yield env.timeout(random.expovariate(1 / MEAN_INTERARRIVAL_RATE))
        traveler_id += 1
        env.process(traveler(env, checkpoint, f"{traveler_id}"))

def average_wait_time(durations):
    mean_wait = np.mean(durations)
    return mean_wait
```

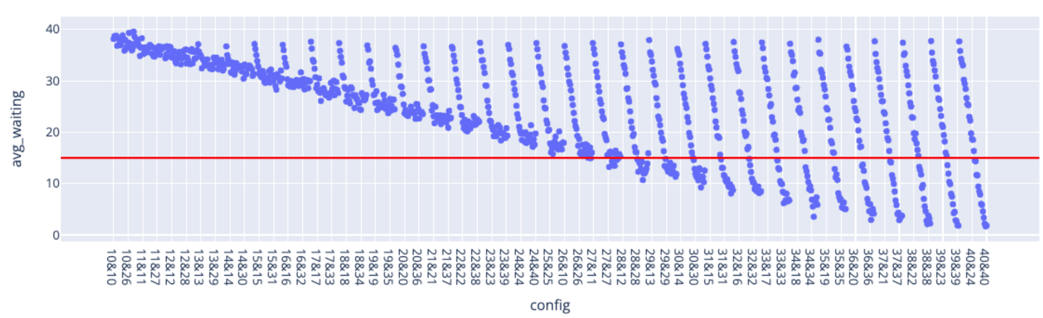
```python
SEED = 1
ID_STATIONS = 20  # will vary
ID_CHECK_SERVER_TIME = 0.75  # minutes

SCAN_STATIONS = 20  # will vary
SCAN_TIME_RANGE = [0.5, 1]  # minutes

# Passengers arrive according to a Poisson distribution with λ1 / per minute
lambda1 = 50  # λ1 = 50 to simulate a busier airport
MEAN_INTERARRIVAL_RATE = 1 / lambda1  # minutes
```

```python
# Getting unique combinations of two lists `id_check_station_time` and `scan_station_time`
import itertools

id_check_station_time = list(range(10, 41, 1))
scan_station_time = list(range(10, 41, 1))
combinations = list(itertools.product(id_check_station_time, scan_station_time))
```

```python
%%time
various_configs = list()
random.seed(SEED)
for comb in combinations:
    wait_durations = []
    ID_STATIONS = comb[0]
    SCAN_STATIONS = comb[1]
    env = simpy.Environment()
    env.process(security_checkpoint_operation(env, ID_STATIONS, SCAN_STATIONS, ID_CHECK_SERVER_TIME, SCAN_TIME_RANGE))
    env.run(until=100)
    avg_waiting = average_wait_time(wait_durations)
    various_configs.append(
        {
            "num_id_stations": ID_STATIONS,
            "num_scan_stations": SCAN_STATIONS,
            "avg_waiting": avg_waiting,
            "config": str(ID_STATIONS) + "&" + str(SCAN_STATIONS)
        }
    )
    del env

df_various_configs = pd.DataFrame(various_configs)
```

```
CPU times: user 1min 31s, sys: 15.6 ms, total: 1min 31s
Wall time: 1min 31s
```

```python
fig = px.scatter(df_various_configs.sort_values(by=["num_id_stations", "num_scan_stations"]), x="config", y="avg_waiting")
fig.add_hline(y=15, line_color="red")
fig.show()
```



Observations: To vary the number of ID/boarding-pass check stations and personal scanning stations, I iterated over combinations of different numbers of both types of stations. The range for each station type is between 10 and 40. Among 961 combinations, there are 185 combinations that result in an average travelers' waiting time of 15 minutes or less. The optimal number of each station can be determined based on external factors such as operation cost. However, if we assume the costs are the same, then the optimal setup includes 27 ID stations and 28 scanning stations, resulting in an average travelers' waiting time of approximately 13 minutes.

	num_id_stations	num_scan_stations	avg_waiting	config
522	26	36	14.988471	26&36
525	26	39	14.979172	26&39
545	27	28	13.235261	27&28
548	27	31	14.326190	27&31
549	27	32	13.821256	27&32
...	...	...	...	...
956	40	36	4.603181	40&36
957	40	37	3.373735	40&37
958	40	38	2.178493	40&38
959	40	39	1.693107	40&39
960	40	40	1.837002	40&40

185 rows × 4 columns

```python
optimals["stations_total"] = optimals["num_id_stations"] + optimals["num_scan_stations"]
```

```python
optimals.loc[optimals["avg_waiting"] <= 15].sort_values(by=["stations_total"], ascending=True).head(5)
```

	num_id_stations	num_scan_stations	avg_waiting	config	stations_total
545	27	28	13.235261	27&28	55
606	29	27	13.434151	29&27	56
576	28	28	14.727944	28&28	56
607	29	28	14.784725	29&28	57
577	28	29	12.306962	28&29	57