Hospital Triage Simulation: Using a Min-Heap to Model ER Priority and Wait Times

ISABELLA CASTILLO

CS313E – ELEMENTS OF DATA ANALYTICS

Problem & Significance

- ER overcrowding is a global healthcare issue
- Delays can be life-threatening
- Staff and resource shortages make triage crucial

Goals

Methodolody

01

Simulate patient arrivals

02

Assign triage/priority levels

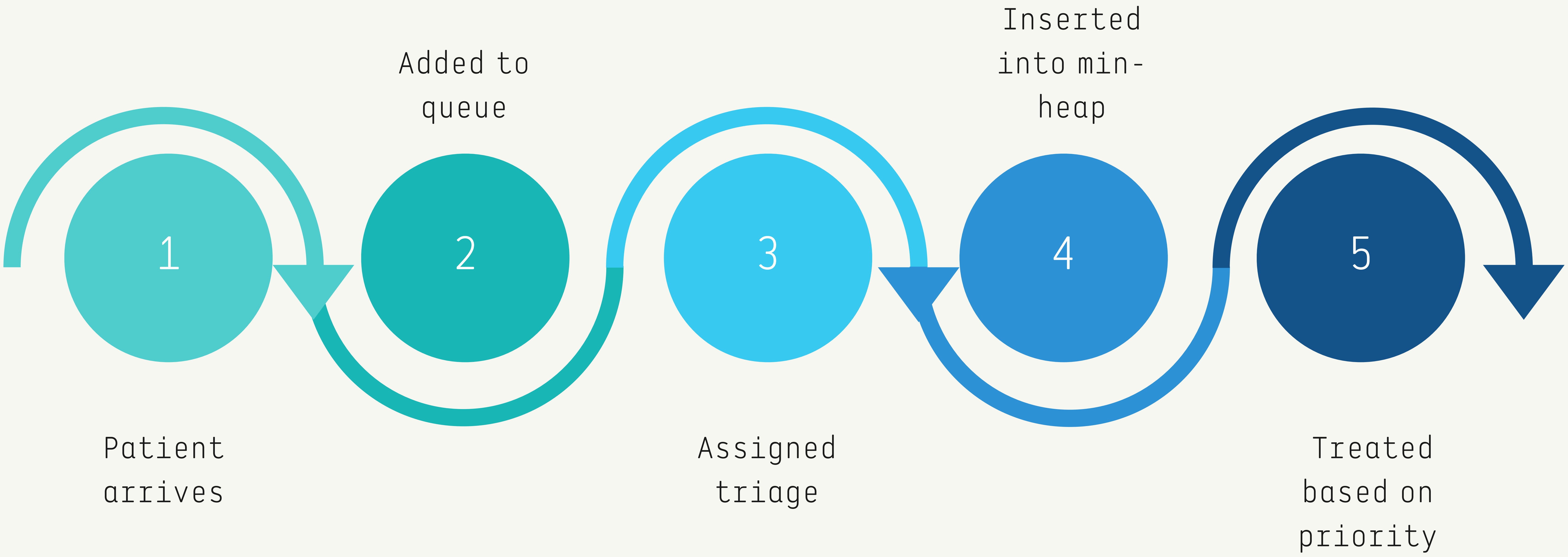
03

Use a min-heap to sort patients by urgency

04

Measure and analyze wait times

SYSTEM DESIGN



Patient arrives

Added to queue

Assigned triage

Inserted into min-heap

Treated based on priority

Data Structures & Algorithms

MIN-HEAP

for automatic patient prioritization

1 (most severe) ... 5 (least)

QUEUE

to manage patient arrivals

Poisson Distribution; severity probabilities: 1:5%, 2:10%, 3:25%, 4:30%, 5:30%

DICTIONARY

to store patient information

Paitent ID, arrival time, and severity

LIST

to store and analyze wait time data

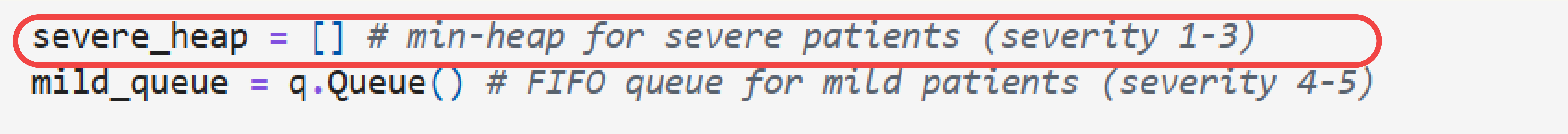
Queue lengths, wait times by severity, avg wait time

Algorithms Used:

- Priority-based scheduling using heap operations (heappush, heappop)
- A discrete-time simulation loop

Data Structures

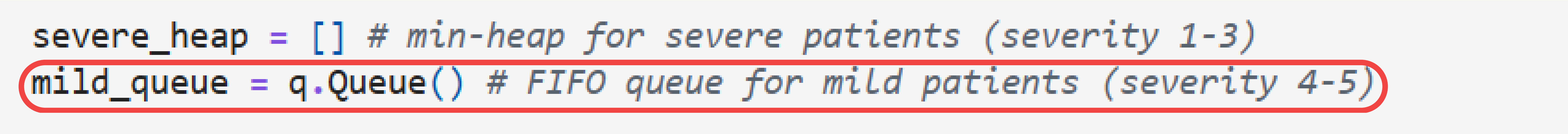
- **Initalized the heap**
- Severe patients added using heappush()
- Mild patients stored in FIFO queue
- Highest priority removed using heappop()
- Simulation runs 1 minute at a time



```
severe_heap = [] # min-heap for severe patients (severity 1-3)
mild_queue = q.Queue() # FIFO queue for mild patients (severity 4-5)
```

Data Structures

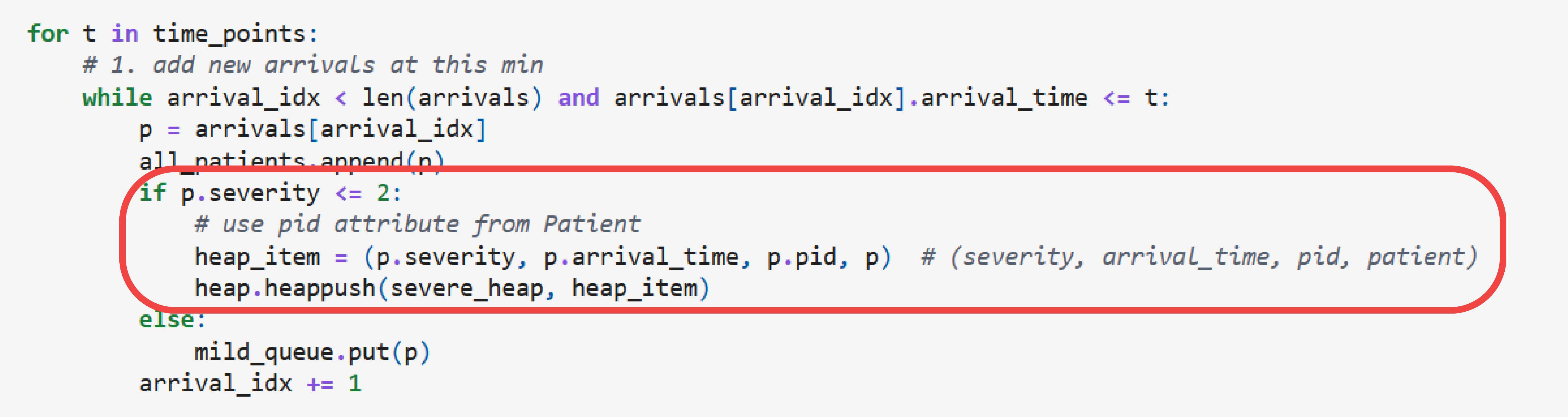
- Initalized the heap
- Severe patients added using heappush[]
- **Mild patients stored in FIFO queue**
- Highest priority removed using heappop[]
- Simulation runs 1 minute at a time



```
severe_heap = [] # min-heap for severe patients (severity 1-3)
mild_queue = q.Queue() # FIFO queue for mild patients (severity 4-5)
```

Data Structures

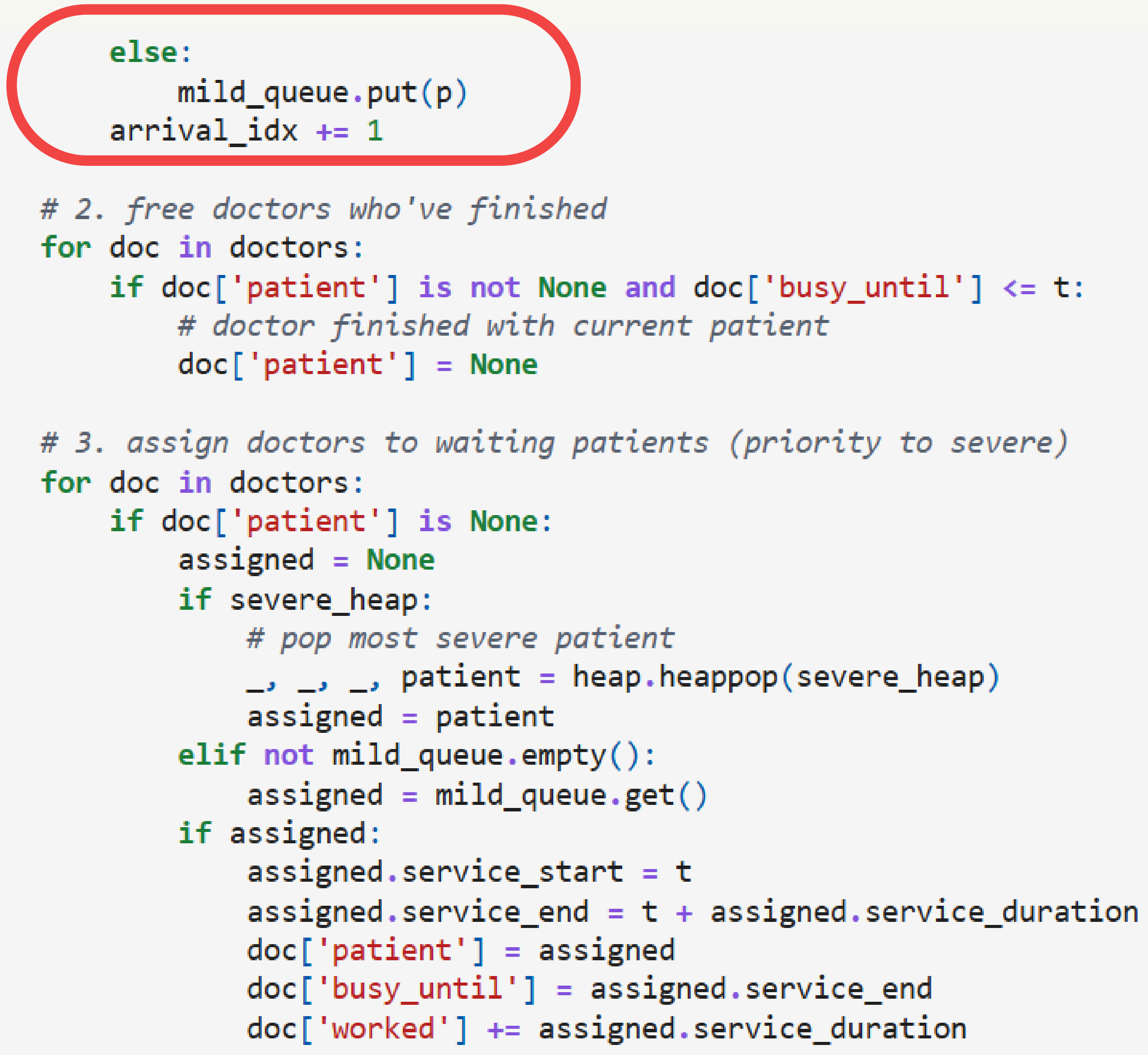
- Initalized the heap
- **Severe patients added using heappush()**
- Mild patients stored in FIFO queue
- Highest priority removed using heappop()
- Simulation runs 1 minute at a time



```
for t in time_points:
    # 1. add new arrivals at this min
    while arrival_idx < len(arrivals) and arrivals[arrival_idx].arrival_time <= t:
        p = arrivals[arrival_idx]
        all_patients.append(p)
        if p.severity <= 2:
            # use pid attribute from Patient
            heap_item = (p.severity, p.arrival_time, p.pid, p)  # (severity, arrival_time, pid, patient)
            heap.heappush(severe_heap, heap_item)
        else:
            mild_queue.put(p)
        arrival_idx += 1
```

Data Structures

- Initalized the heap
- Severe patients added using heappush[]
- **Mild patients stored in FIFO queue**
- Highest priority removed using heappop[]
- Simulation runs 1 minute at a time



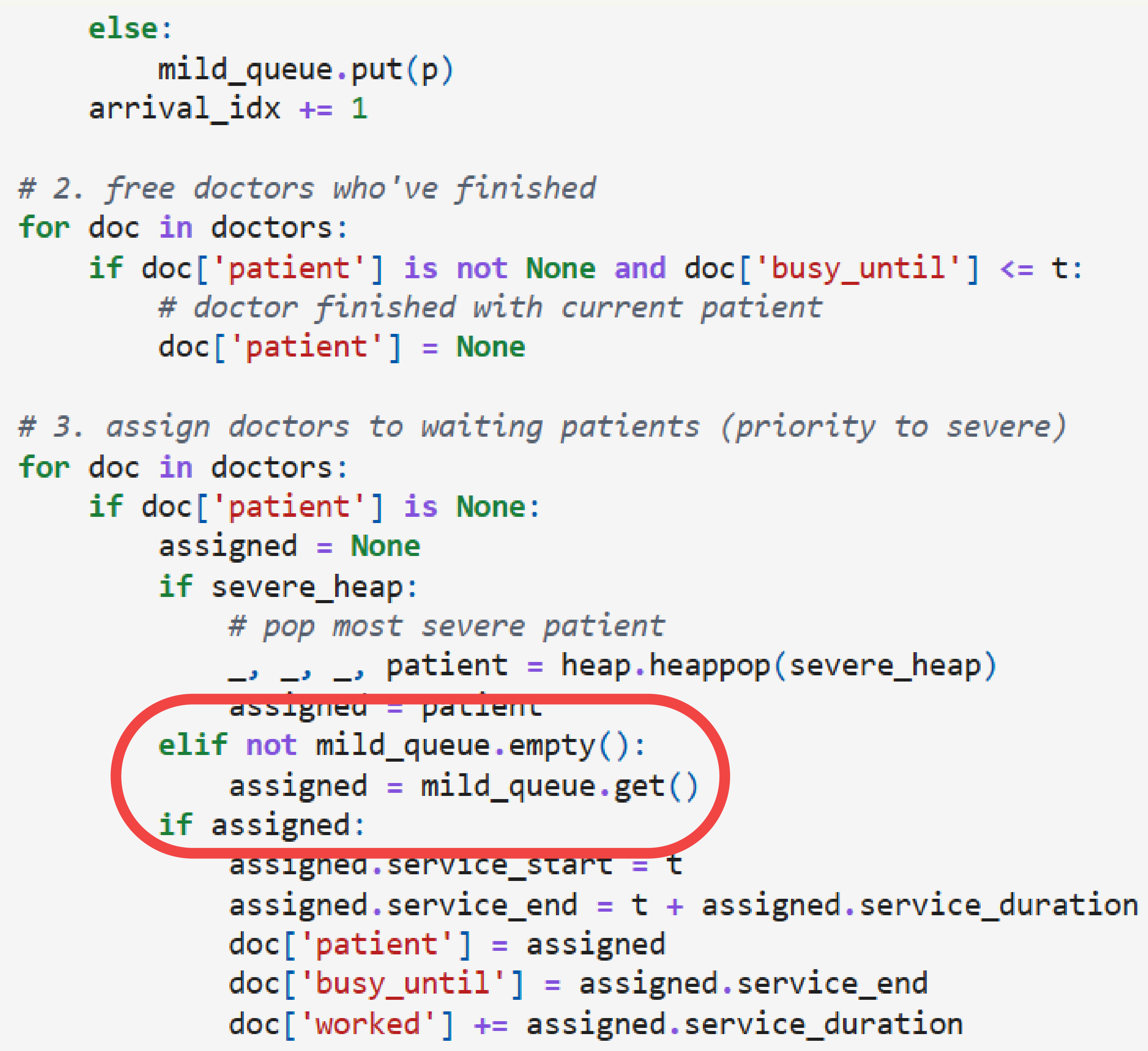
```
    else:
        mild_queue.put(p)
    arrival_idx += 1

# 2. free doctors who've finished
for doc in doctors:
    if doc['patient'] is not None and doc['busy_until'] <= t:
        # doctor finished with current patient
        doc['patient'] = None

# 3. assign doctors to waiting patients (priority to severe)
for doc in doctors:
    if doc['patient'] is None:
        assigned = None
        if severe_heap:
            # pop most severe patient
            _, _, _, patient = heap.heappop(severe_heap)
            assigned = patient
        elif not mild_queue.empty():
            assigned = mild_queue.get()
        if assigned:
            assigned.service_start = t
            assigned.service_end = t + assigned.service_duration
            doc['patient'] = assigned
            doc['busy_until'] = assigned.service_end
            doc['worked'] += assigned.service_duration
```

Data Structures

- Initalized the heap
- Severe patients added using heappush[]
- **Mild patients stored in FIFO queue**
- Highest priority removed using heappop[]
- Simulation runs 1 minute at a time



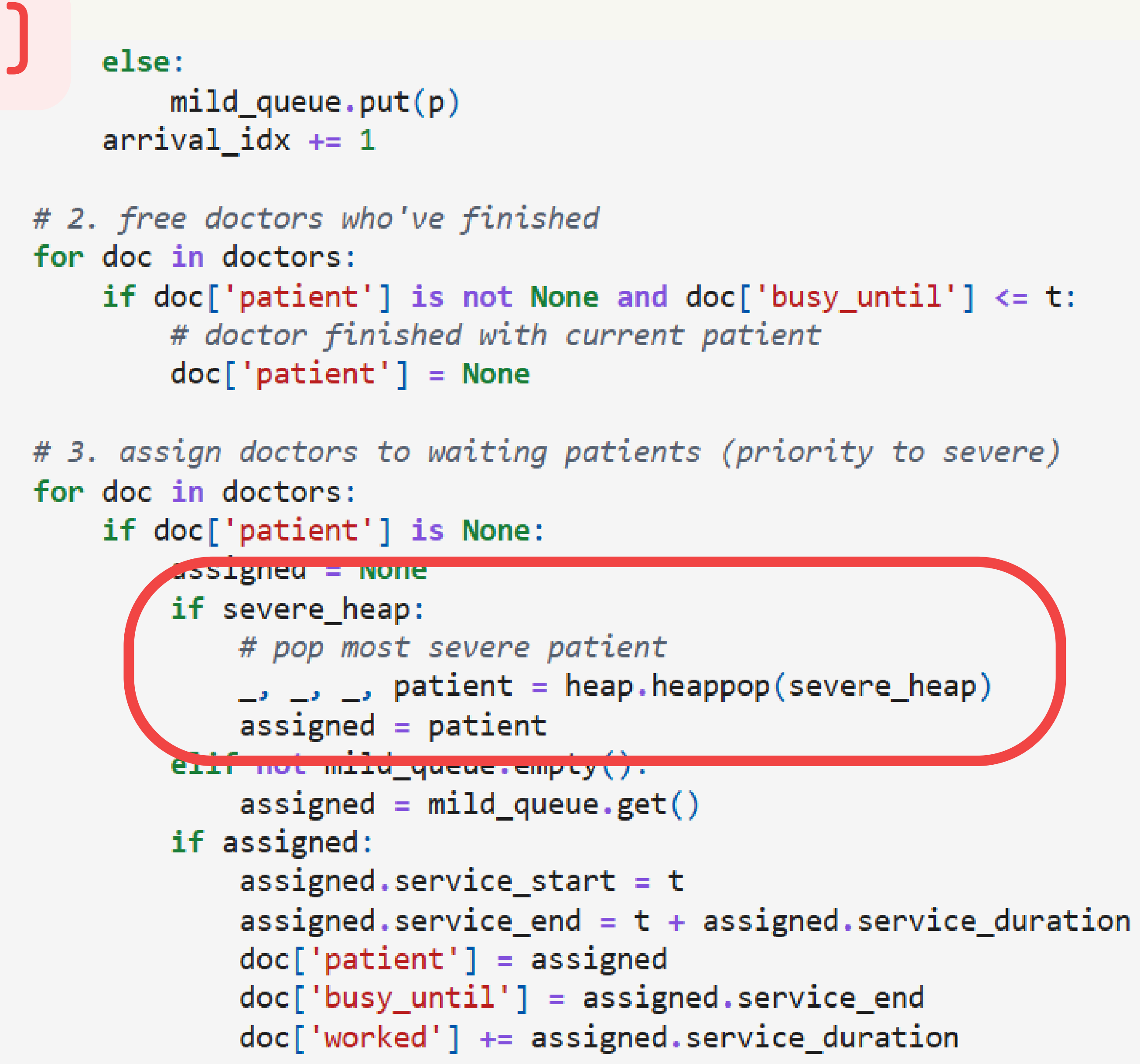
```
    else:
        mild_queue.put(p)
    arrival_idx += 1

# 2. free doctors who've finished
for doc in doctors:
    if doc['patient'] is not None and doc['busy_until'] <= t:
        # doctor finished with current patient
        doc['patient'] = None

# 3. assign doctors to waiting patients (priority to severe)
for doc in doctors:
    if doc['patient'] is None:
        assigned = None
        if severe_heap:
            # pop most severe patient
            _, _, _, patient = heap.heappop(severe_heap)
            assigned = patient
        elif not mild_queue.empty():
            assigned = mild_queue.get()
        if assigned:
            assigned.service_start = t
            assigned.service_end = t + assigned.service_duration
            doc['patient'] = assigned
            doc['busy_until'] = assigned.service_end
            doc['worked'] += assigned.service_duration
```

Data Structures

- Initalized the heap
- Severe patients added using heappush[]
- Mild patients stored in FIFO queue
- **Highest priority removed using heappop[]**
- Simulation runs 1 minute at a time



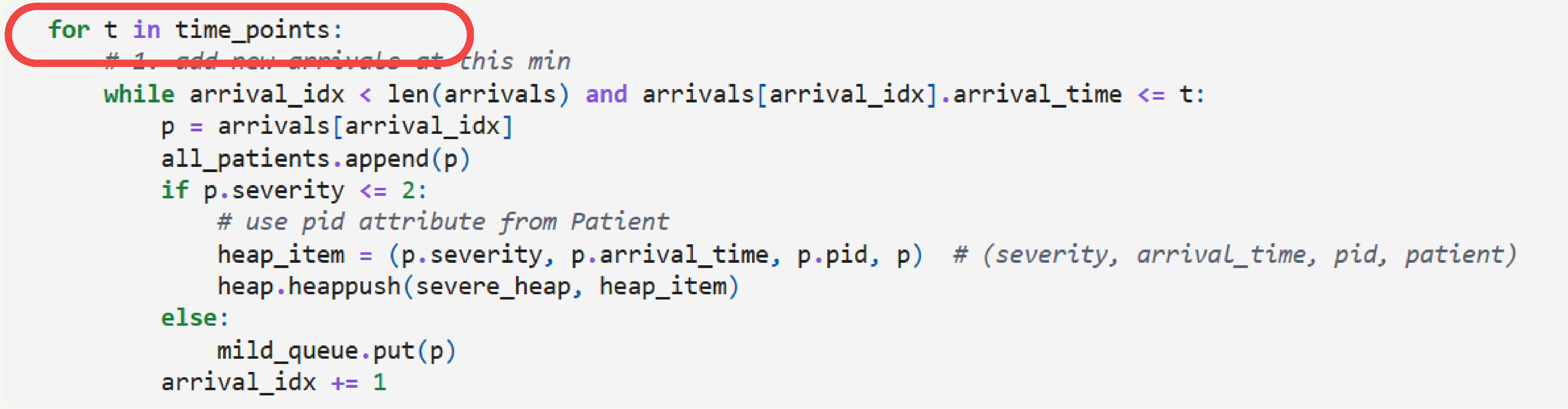
```
    else:
        mild_queue.put(p)
    arrival_idx += 1

# 2. free doctors who've finished
for doc in doctors:
    if doc['patient'] is not None and doc['busy_until'] <= t:
        # doctor finished with current patient
        doc['patient'] = None

# 3. assign doctors to waiting patients (priority to severe)
for doc in doctors:
    if doc['patient'] is None:
        assigned = None
        if severe_heap:
            # pop most severe patient
            _, _, _, patient = heap.heappop(severe_heap)
            assigned = patient
        elif not mild_queue.empty():
            assigned = mild_queue.get()
        if assigned:
            assigned.service_start = t
            assigned.service_end = t + assigned.service_duration
            doc['patient'] = assigned
            doc['busy_until'] = assigned.service_end
            doc['worked'] += assigned.service_duration
```

Data Structures

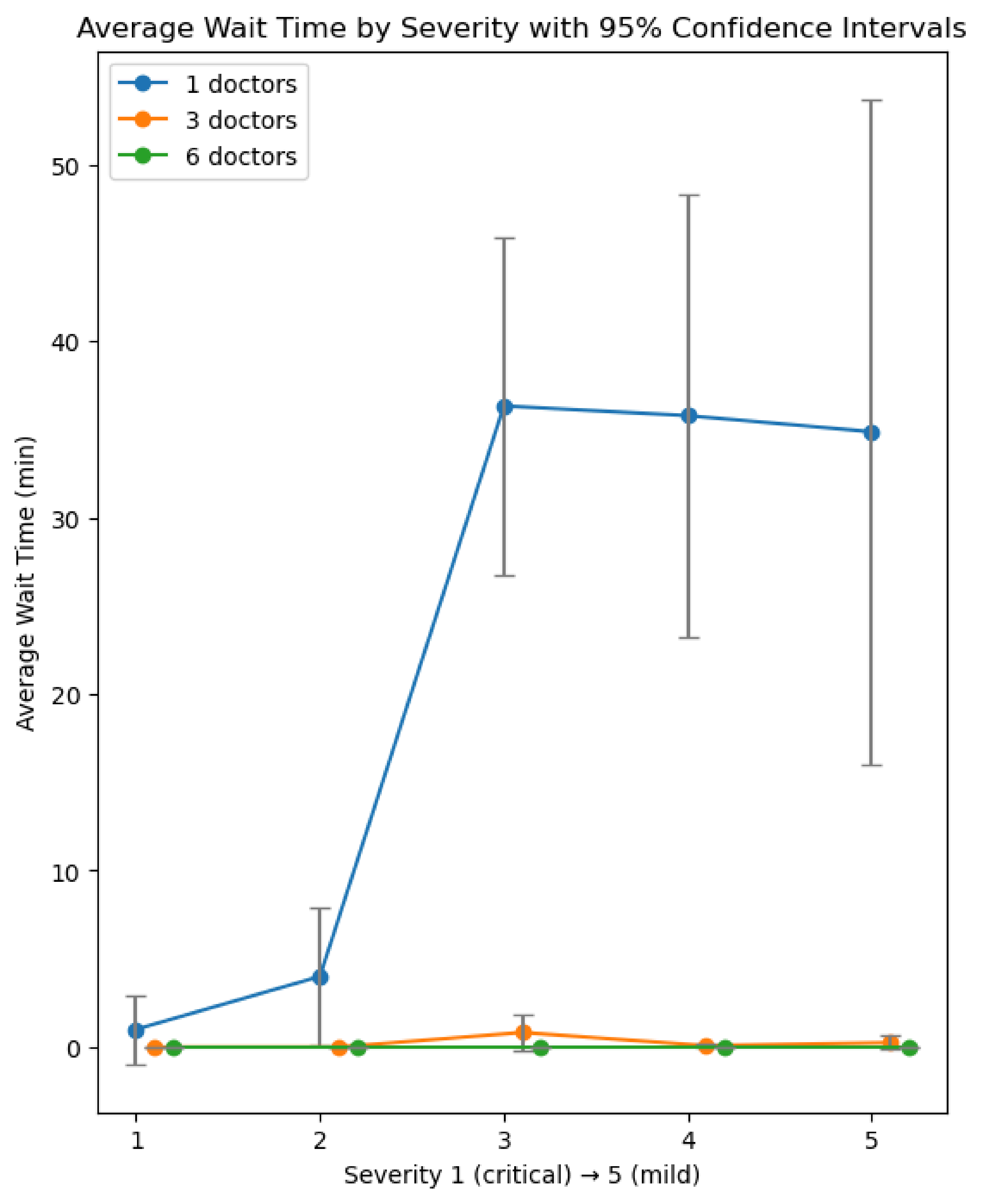
- Initalized the heap
- Severe patients added using heappush[]
- Mild patients stored in FIFO queue
- Highest priority removed using heappop[]
- **Simulation runs 1 minute at a time**



```
for t in time_points:
    # 1. add new arrivals at this min
    while arrival_idx < len(arrivals) and arrivals[arrival_idx].arrival_time <= t:
        p = arrivals[arrival_idx]
        all_patients.append(p)
        if p.severity <= 2:
            # use pid attribute from Patient
            heap_item = (p.severity, p.arrival_time, p.pid, p)  # (severity, arrival_time, pid, patient)
            heap.heappush(severe_heap, heap_item)
        else:
            mild_queue.put(p)
        arrival_idx += 1
```

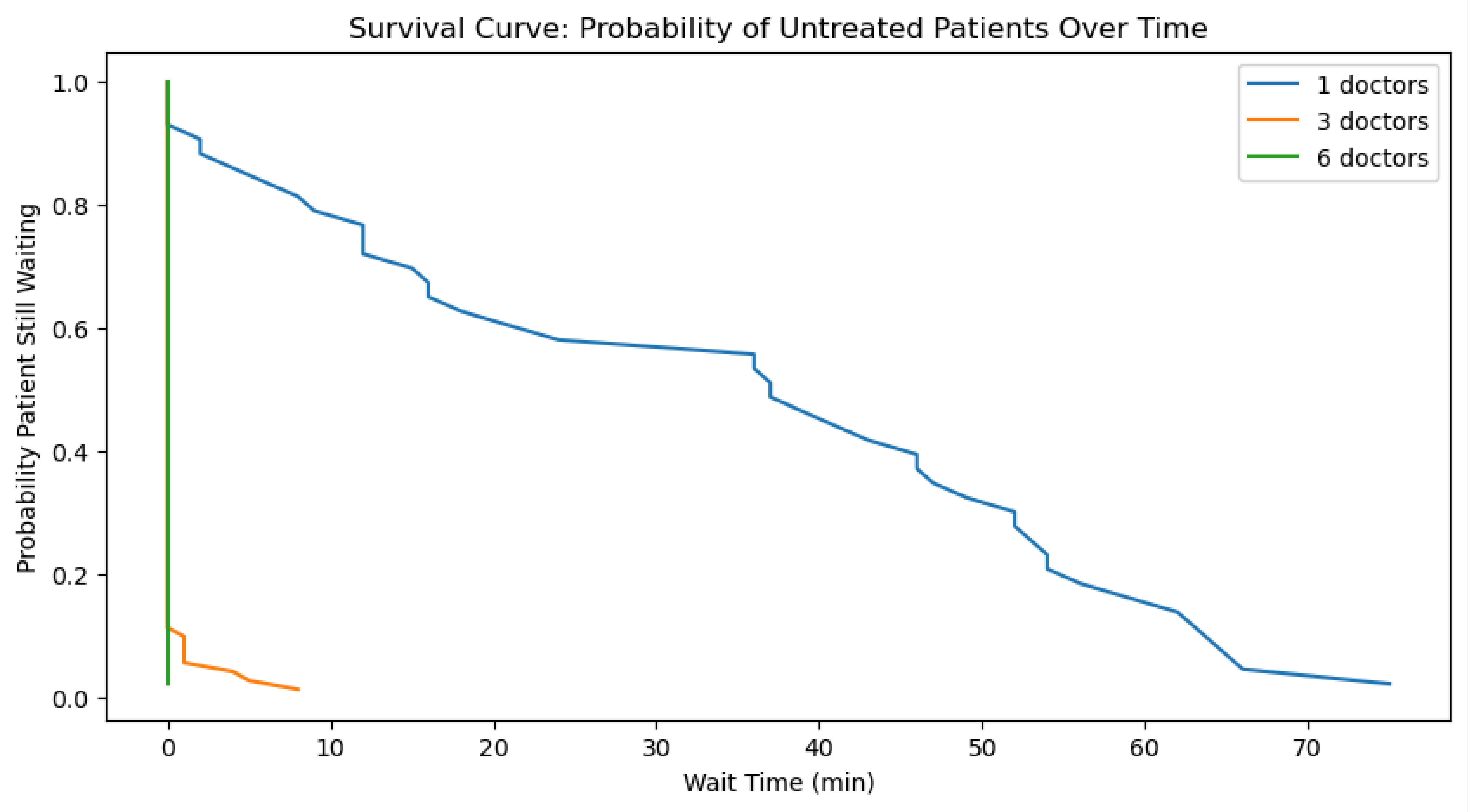
Outcome

- 1 doctor → long wait times and unsafe delays
- 3 doctors → minimal waits and 0% delayed
- 6 doctors → no waits but inefficient use of staff
- Shows tradeoff between safety & resources



Outcome

- Shows probability that patients remain untreated
- 1 doctor → untreated for 60+ minutes
- 3+ doctors → near immediate treatment



Limitations

- Uses simulated/random data
- Simplified hospital environment
- Assumes perfect triage decisions

Improvements

- Add real hospital datasets
- Model multiple departments
- Add staff shift changes
- Include machine learning for predictions

Thanks for your attention!