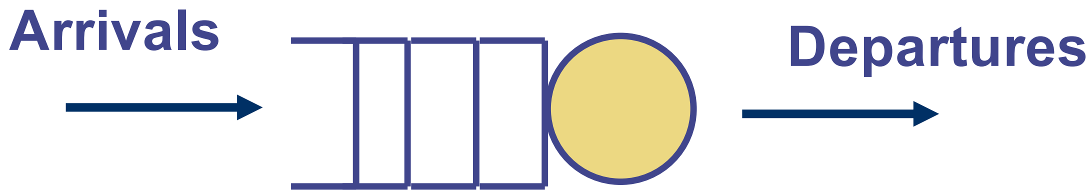
COMP9334
Capacity Planning for Computer Systems and Networks

Week 4B: Discrete event simulation (1)

Week 4A: Queues with general arrival & service time

- Queues with general inter-arrival and service time distributions

General Inter-arrivals time distribution
General service time distribution



- M/G/1 queue
  - Can calculate the mean waiting time W with the P-K formula

$$W = \frac{\lambda E[S^2]}{2(1-\rho)}$$

- G/G/1 queue
  - No explicit formula, get a bound or approximation

$$W \leq \frac{\lambda(\sigma_a^2 + \sigma_s^2)}{2(1-\rho)}$$

Analytical methods for queues

- You had learnt how to solve a number of queues analytically (= mathematically) given their
  - Inter-arrival time probability distribution
  - Service time probability distribution

- Queues that you can solve now include M/M/1, M/M/m, M/G/1, M/G/1 with priorities etc.
  - If you know the analytical solution, this is often the most straightforwad way to solve a queueing problem
- Unfortunately, *many queueing problems are still analytically intractable!*

- What can you do if we have an analytically intractable queueing problem?

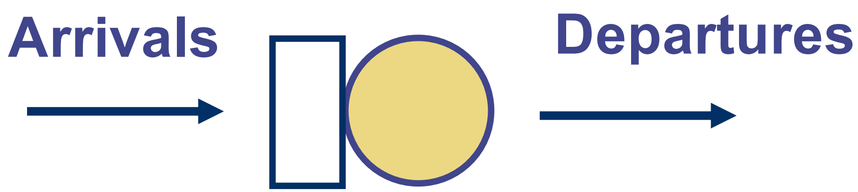
Lectures 4B, 5A, 5B: Discrete event simulation

- In these lectures, we look at the topic of using *discrete event simulation for queueing problems*
  - *Simulation is an imitation of the operation of real-life system over time.*

- The topics to be covered are
  - (4B) What are discrete event simulation?
  - (4B) How to structure a discrete event simulation?

  - For 5A and 5B
    - How to choose simulation parameters?
    - How to analyse data?
    - What are the pitfalls that you need to avoid?
    - How to generate pseudo-random numbers for simulation?
    - Reproducibility

Motivating example



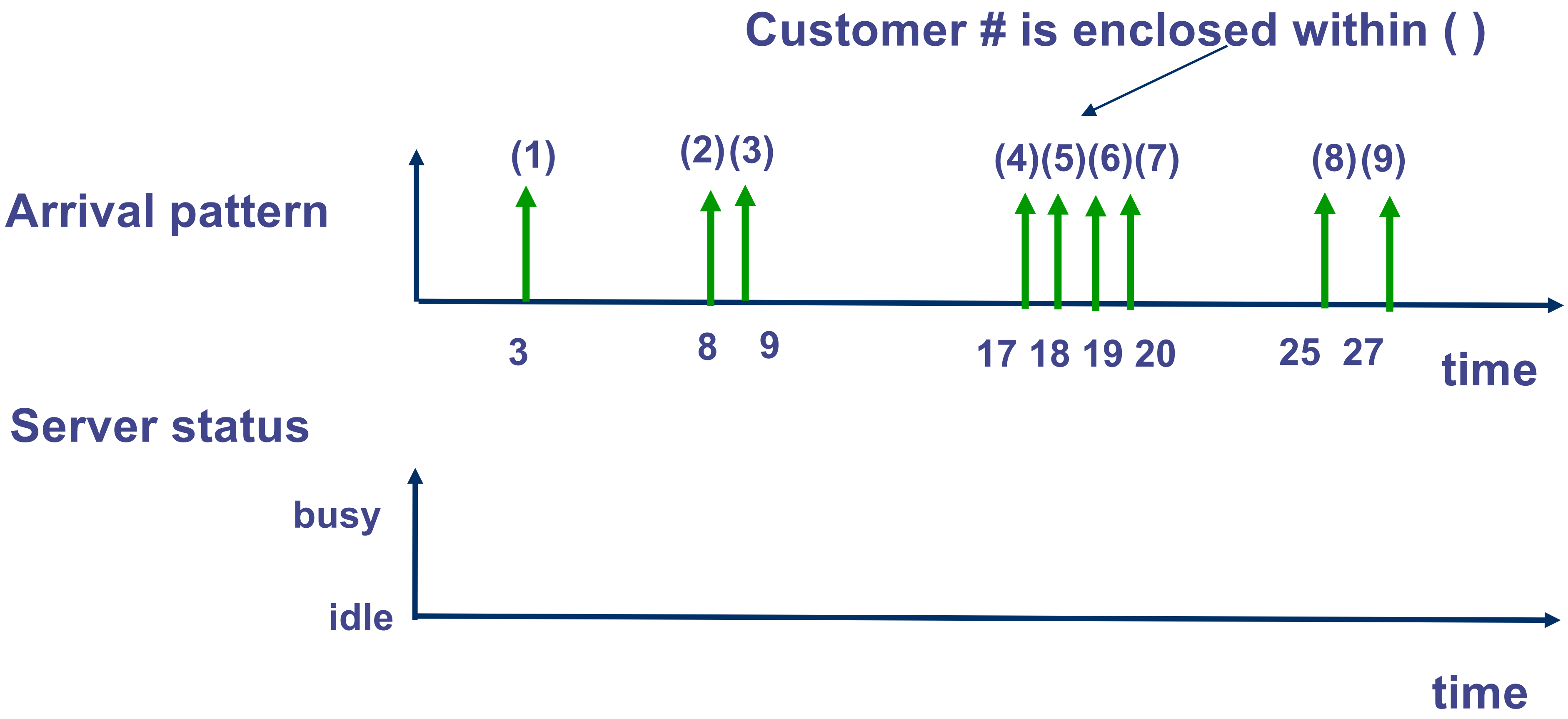
- Consider a single-server queue with only one buffer space (= waiting room)
- If a customer arrives when the buffer is occupied, the customer is rejected.
- Given the arrival times and service times in the table on the right, find
  - The mean response time
  - % of rejected customers

  Assuming an idle server at time = 0.

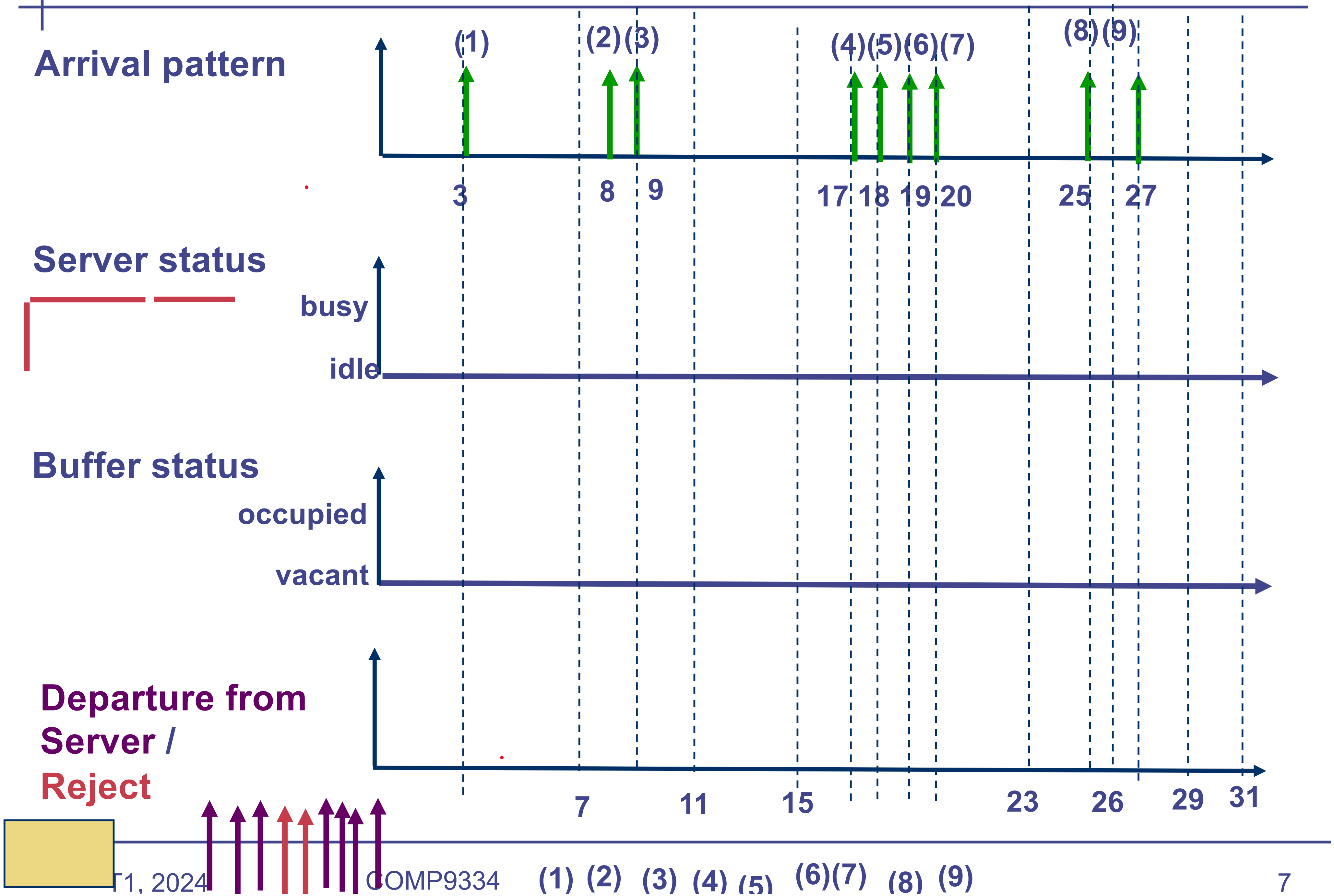
Customer number	Arrival time	Service time
1	3	4
2	8	3
3	9	4
4	17	6
5	18	3
6	19	2
7	20	2
8	25	3
9	27	2

Let us try a graphical solution

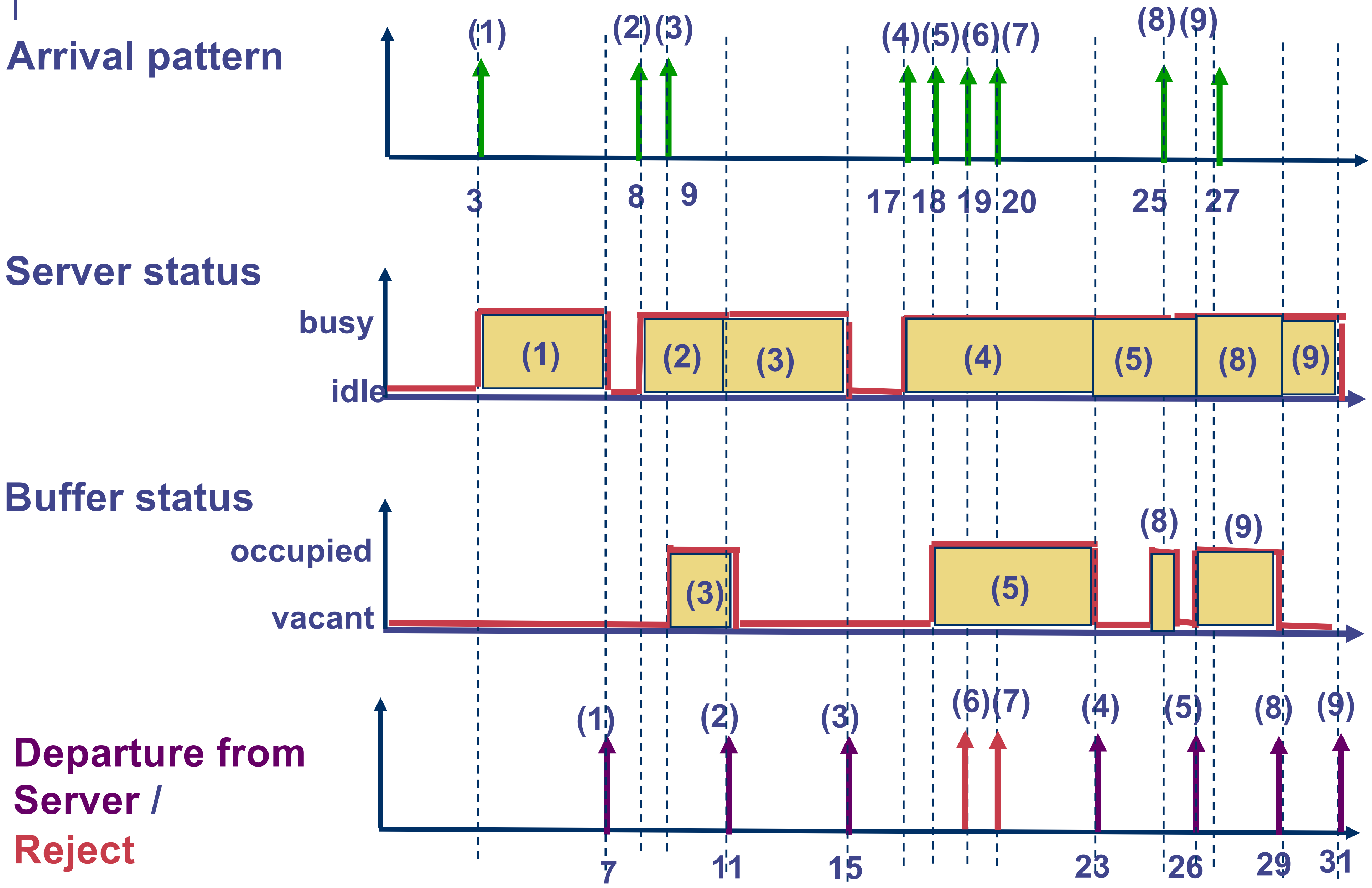
- In the graphical solution, we will keep track of
  - The status of the server: busy or idle
  - The status of the buffer: occupied or vacant



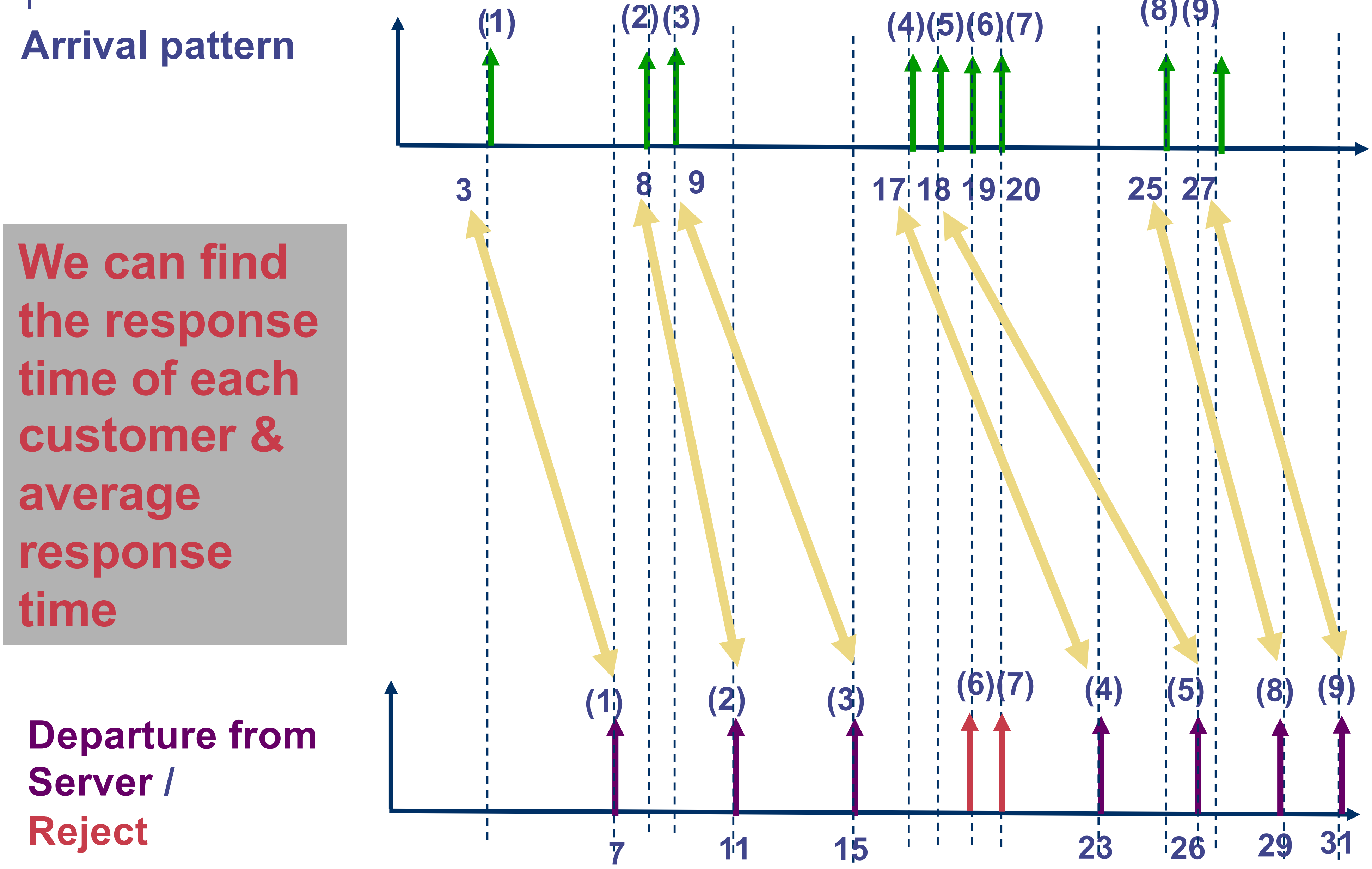
A graphical solution (To be completed)



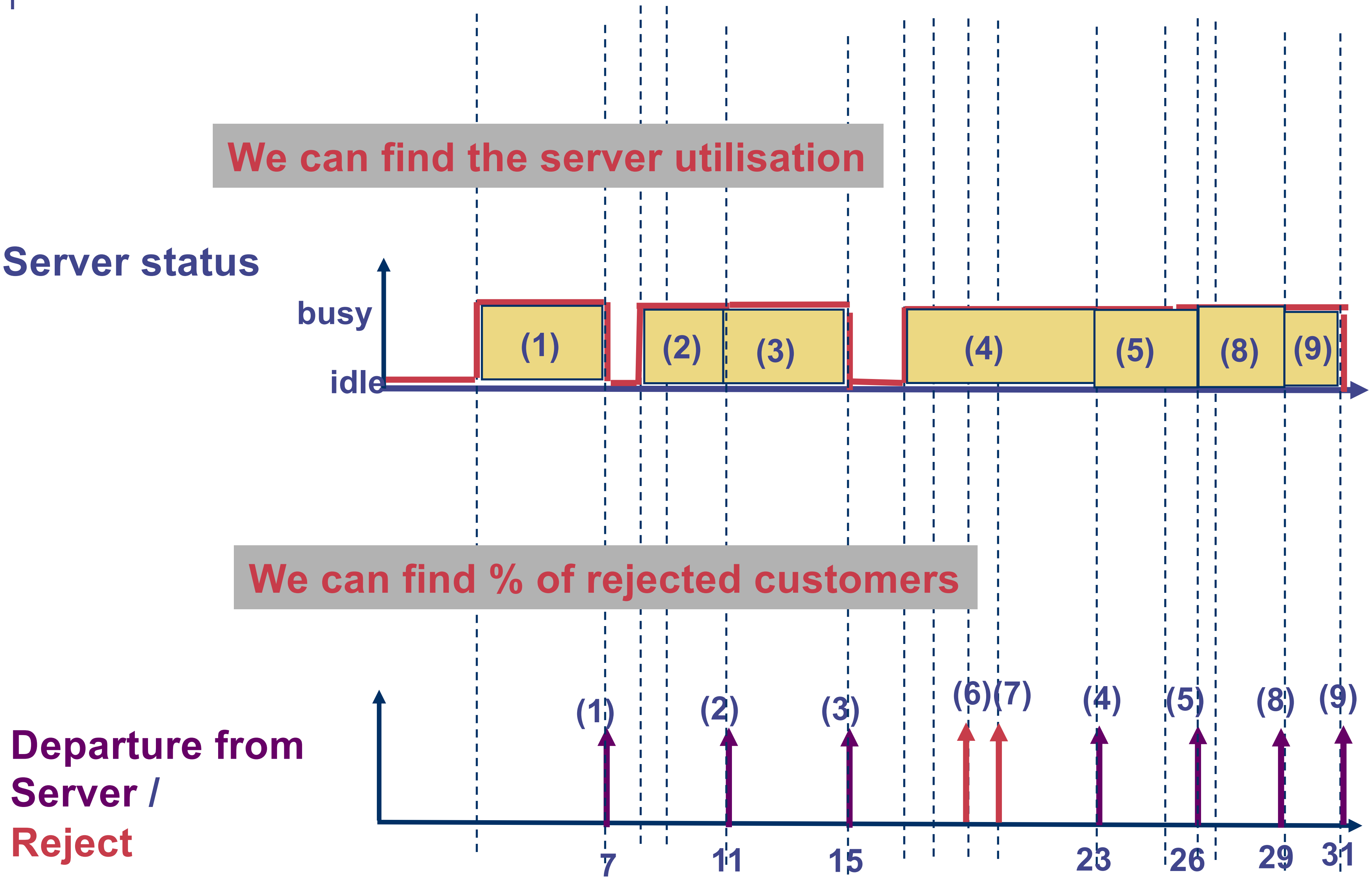
A graphical solution



Using the graphical solution (1)



Using the graphical solution (2)



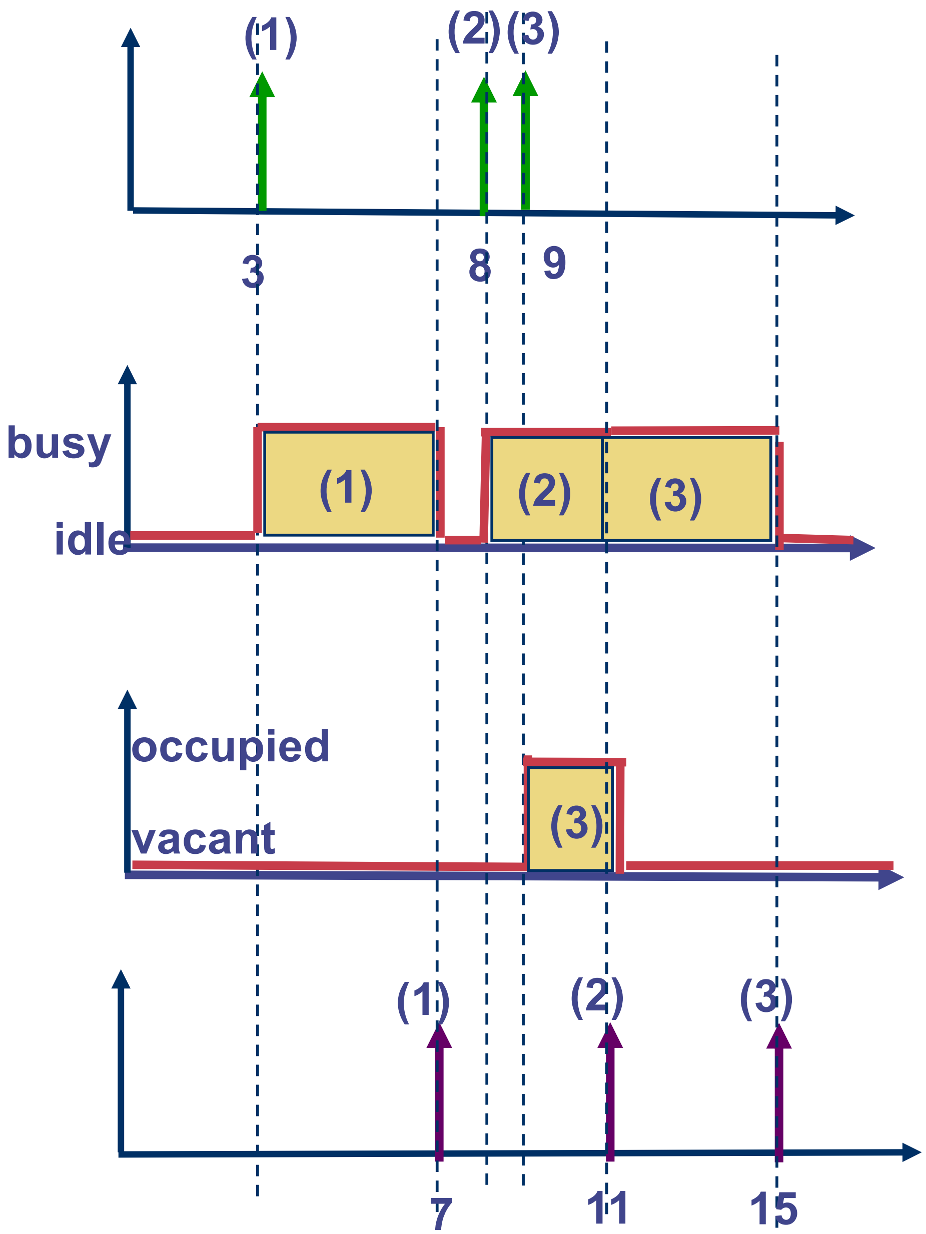
From graphical solution to computer solution (1)

- How can we turn this graphical solution into a computer solution, i.e. a computer program that can solve the problem for us

- We need to keep track of the status of the server and the status of the buffer,
  - This allows us to make decisions
  - E.g. If server is BUSY and buffer is OCCUIPIED, an arriving customer is rejected.
  - E.g. If server is BUSY and buffer is VACANT, an arriving customer goes to the buffer.
  - E.g. If server is IDLE, an arriving customer goes to the sever
- *What this means*: We need to keep track of the status of some variables in our computer solution.

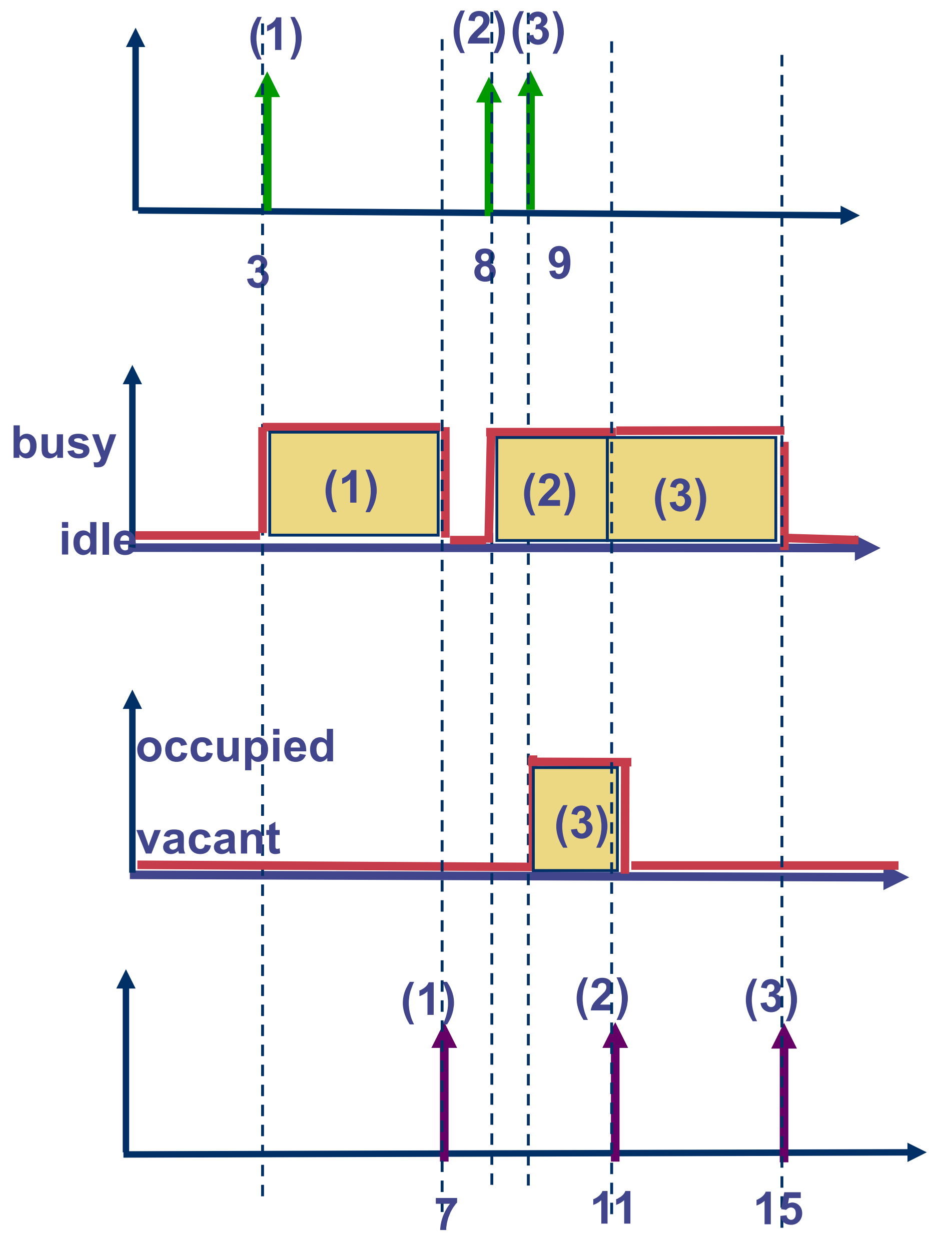
From graphical solution to computer solution (2)

- Observation #1:
  - An arriving or departing customer causes the server or buffer status to change
  - Examples:
    - At time = 3, the arrival of customer #1 causes the server to switch from IDLE to BUSY
    - At time = 7, the departure of customer #1 causes the server to switch from BUSY to IDLE
    - At time = 9, the arrival of customer #3 causes the buffer to switch from VACANT to OCCUPIED
    - Etc.



From graphical solution to computer solution (3)

- Let us call the arrival of a customer or the departure of a customer an ***event***
- Observation #2:
  - The status of the server and the status of the buffer remain the same between two consecutive events
- What this means:
  - We need to keep track of the timing of the events
    - Events can cause status transitions
    - In between events, status remain the same



From graphical solution to computer solution (4)

- In our computer solution, we will use a ***master clock*** to keep track of the current time
- We will advance the master clock from event to event
- In order to see how the computer solution works, let us try it out on paper first

On paper simulation

- In our simulation, we keep track of a number of variables
  - MC = Master clock
  - Status of
    - Server: 1 = BUSY, 0 = IDLE
    - Buffer: 1 = OCCUPIED, 0 = VACANT
  - Event time:
    - Next arrival event and service time of this arrival
    - Next departure event and arrival time of this departure
  - The (arrival time, service time) of the customer in buffer
  - In order to compute the response time, we keep track of
    - The cumulative response time (T)
    - Cumulative number of customers rejected (R)

MC	Next arrival		Next departure from the server		Server status	Buffer status + customer in buffer	T	R
	Arrival time	Service time	Departure time	Arrival time of this departure				

On paper simulation (To be completed)

MC	Next arrival		Next departure		Server status	Buffer status + Customer in buffer	T	R
	Arrival time	Service time	Departure time	Arrival time of this departure				
0	3	4	-	-	0	0	0	0
3	8	3	7	3	1	0	0	0
7	8	3	-	-	0	0	4	0

Can you continue?

(Arrival time, service time) of the customer in the buffer.

On paper simulation

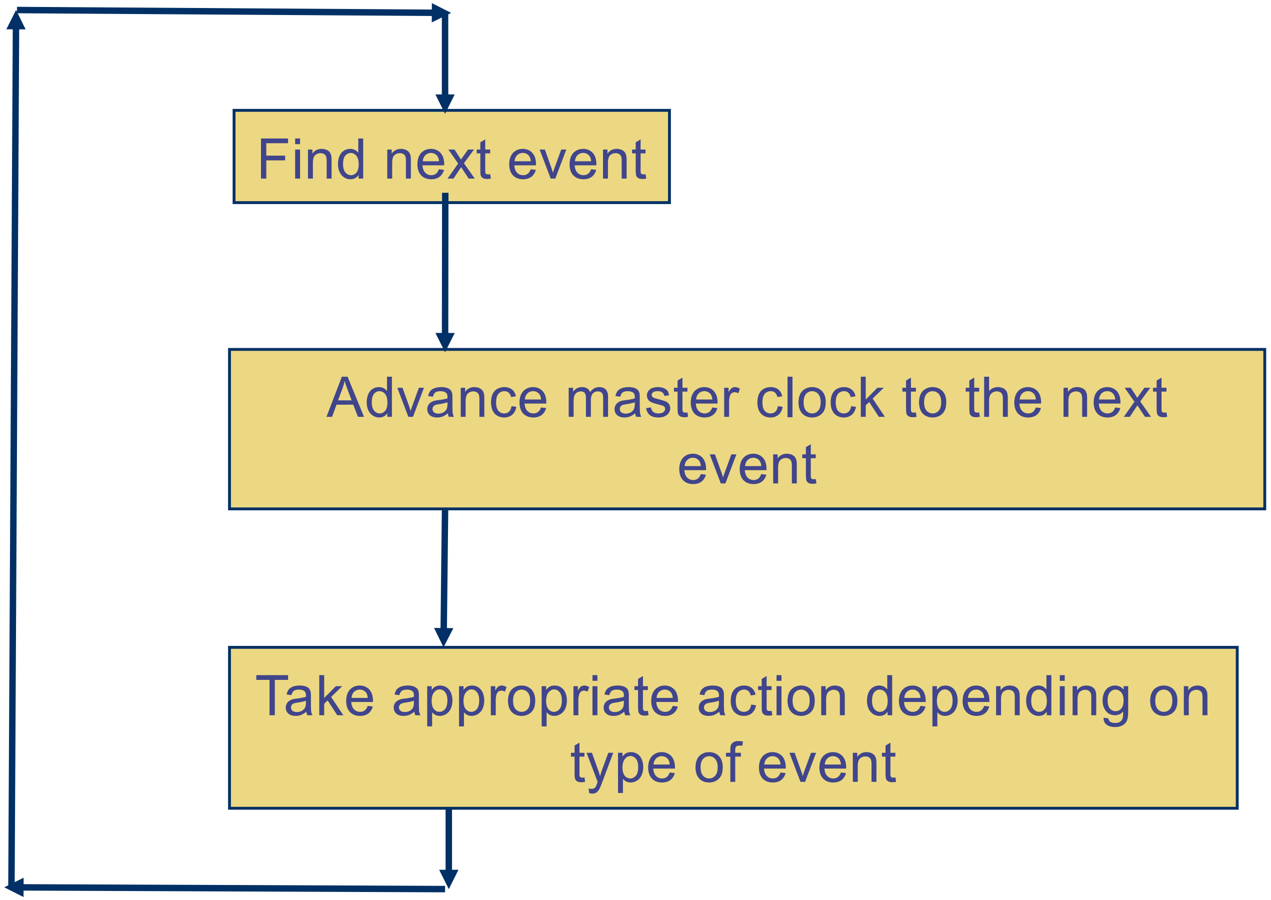
MC	Next arrival		Next departure		Server status	Buffer status + Customer in buffer	T	R
	Arrival time	Service time	Departure time	Arrival time of this departure				
0	3	4	-	-	0	0	0	0
3	8	3	7	3	1	0	0	0
7	8	3	-	-	0	0	4	0
8	9	4	11	8	1	0	4	0
9	17	6	11	8	1	1 (9,4)	4	0
11	17	6	15	9	1	0	7	0
15	17	6	-	-	0	0	13	0

Can you continue?

(Arrival time, service time) of the customer in the buffer.

Logic of the program (1)

- At each step, we advance to the next event that will take place



Handling an arrival event

Three cases according to the server and/or buffer status

Arrival event

Server IDLE

(Buffer VACANT)

- Add a departure event with departure time = current time + service time of the arrival
- Change server status to BUSY

Server BUSY

Buffer VACANT

- Change buffer status to OCCUPIED
- Store the arrival time and service time of this arrival with buffer information

Server BUSY

Buffer OCCUPIED

- Reject this customer
- Increment the cumulative number of rejected customers by one

---

- Look up the list of arrival to fill in the information for the next arrival event

Handling an departure event

Two cases according to the buffer status

Departure event

- Update the cumulative response time
  - T 🡐 T + current time - arrival time of the departing customer

Buffer VACANT

- Change server status to IDLE
- Next departure event becomes empty

Buffer OCCUPIED

- Update the departure event with information of the customer in the buffer
- Next departure time =

  current time + service time of the customer in the buffer
- Change buffer status to VACANT

Discrete event simulation

- The above computer program is an example of a discrete event simulation
- It allows you to solve a queueing problem with one server and one buffer space
- You can generalise the above procedure to
  - Multi-server
  - Finite or infinite buffer space
  - Different queueing disciplines
- Let us generalise it to the case of single-server with infinite buffer

Single server with infinite buffer simulation

- In this case, we will use buffer status to denote the number of customers in buffer
  - Buffer status = 0, 1, 2, 3, …
- We also need to store all the (arrival time, service time) of all the customers in the buffer
- Compare with the single-server single-buffer case, we only need to change the handling of
  - An arrival event
  - A departing event

Handling an arrival event

Two cases according to the server status

Arrival event

Server IDLE

- Add a departure event with departure time = current time + service time of the arrival
- Change server status to BUSY

Server BUSY

- Increment number of customers in the buffer by 1
- Store the arrival time and service time of this arrival with buffer information

- Look up the list of arrival to fill in the information for the next arrival

Handling an departure event

Two cases according to the buffer status

Departure event

- Update the cumulative response time
  - T 🡐 T + current time - arrival time of the departing customer

Buffer = 0

- Change server status to IDLE
- Departure event becomes empty

Buffer $\neq$ 0

- Update the departure event with first customer in the buffer
- Next departure time =

  current time + service time of the first customer in the buffer
- Delete first customer from buffer
- Decrement number of customers in the buffer by 1

One missing piece

- We know how to write a discrete event simulation program to simulate a single-server queue with infinite buffer assuming that we have the arrival times and service times

- Where do arrival times and service times come from?

- If we want to simulate an M/M/1 queue
  - The inter-arrival time is exponentially distributed
  - The service time is exponentially distributed

- We can get the arrival times and service times if we can generate exponentially distributed random numbers

The Python random library

- The random library can be used to generate random numbers from many probability distributions
- random.expovariate() can be used to generate exponentially distributed random numbers

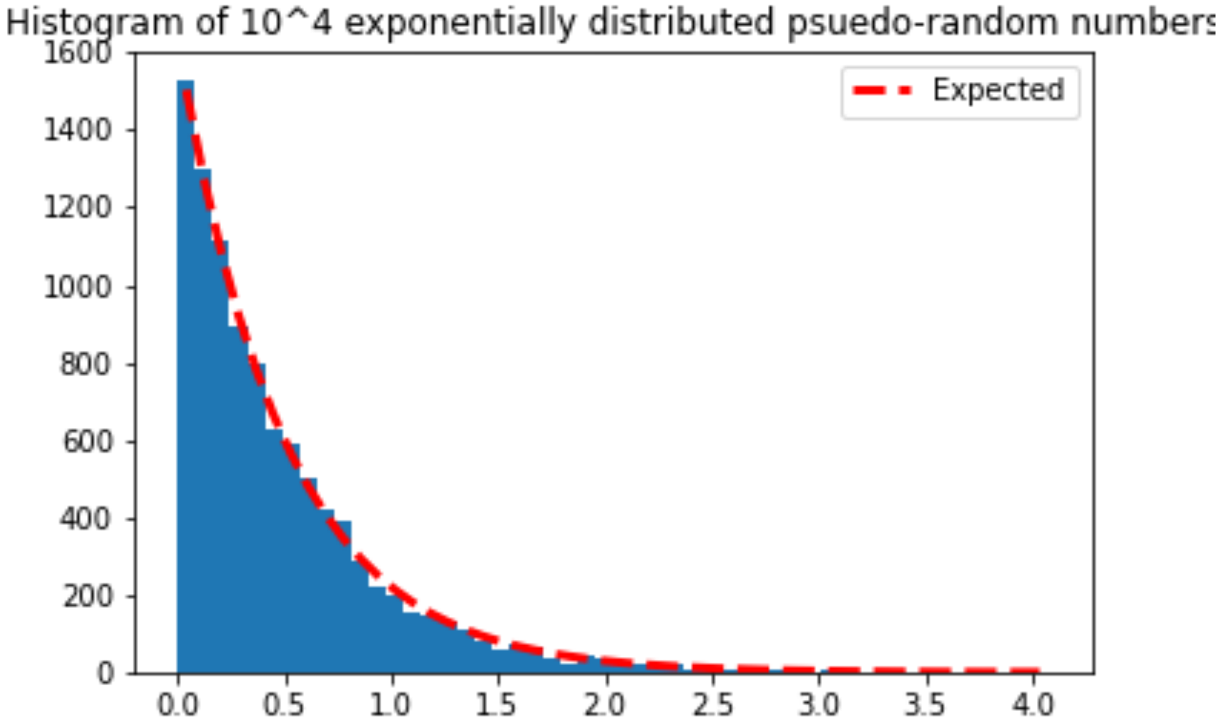
```
import random

lamb = 2
i1 = random.expovariate(lamb)
i2 = random.expovariate(lamb)
i3 = random.expovariate(lamb)
```

Name	Type	Size	Value
i1	float	1	0.886333566840348
i2	float	1	0.5129872509130181
i3	float	1	0.2597444444294557

Exponential distributed random numbers

```
# To produce 10,000 numbers that are exponentially distributed
lamb = 2
n = 10000
x = []
for i in range(n):
    x.append(random.expovariate(lamb))
```

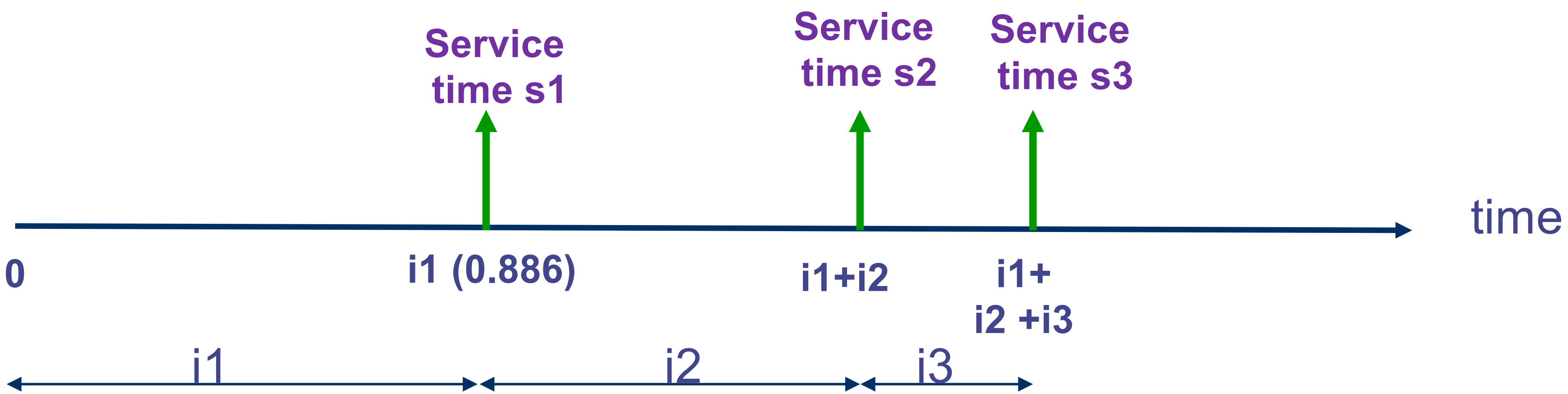
- Generate 10,000 exponentially distributed number and plot the histogram
- File: hist_random_expo.py
- Note: lambda is a Python keyword. Cannot use lambda as a variable name



Arrival and service times

```
import random

lamb = 2
i1 = random.expovariate(lamb)
i2 = random.expovariate(lamb)
i3 = random.expovariate(lamb)

mu = 1
s1 = random.expovariate(mu)
s2 = random.expovariate(mu)
s3 = random.expovariate(mu)
```

Name	Type	Size	Value
i1	float	1	0.886333566840348
i2	float	1	0.5129872509130181
i3	float	1	0.259744444294557
lamb	int	1	2
mu	int	1	1
s1	float	1	1.118666003053235
s2	float	1	5.088431468890377
s3	float	1	0.635335569495133



Simulating M/M/1 queue

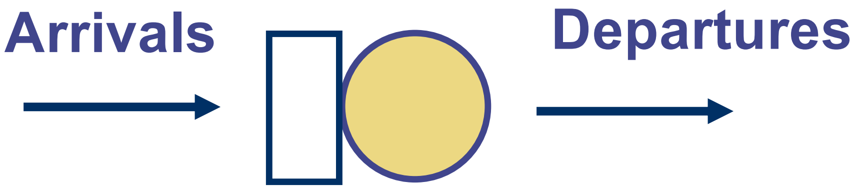
- In order to test how well our discrete event simulation program works, we will use it to simulate an M/M/1 queue and compare it with the expected result
- An M/M/1 simulation program is given in *sim_mm1.py* (available on the course web site)
- We will:
  - Take a look at the program
  - Run it and make some observations

Observations from running the simulation

- The mean response time from simulation can be close to (but not equal to) the theoretical mean response time

- Each simulation run gives a different mean response time

Trace driven simulation



- We considered this example in the beginning of this lecture
- We simulated using
  - A sequence (or trace) of arrival times
  - A sequence of service times
- We call this trace driven simulation
- Trace driven simulation is useful
  - You have a server and you have a log of the arrival time and service time of the job
  - You are considering changing to a new server
  - You can use the traces that you have and simulation to calculate the response time of the new server

Customer number	Arrival time	Service time
1	3	4
2	8	3
3	9	4
4	17	6
5	18	3
6	19	2
7	20	2
8	25	3
9	27	2

Trace driven simulation

- An example of trace driven simulation is in the file sim_1server_trace.py
  - Note that sim_1server_trace.py assumes infinite buffer rather than finite buffer
- Earlier we used random number generators to produce inter-arrival and service time
  - For trace driven simulation, the arrival time and service time are read from the supplied trace

Summary

- Simulation can be used when analytical solutions are not available
- Basic components of discrete event simulation
  - Master clock advancing from event to event
  - Event handling
    - Update the simulation variables according to the event
- Discrete event simulation can make use of
  - Randomly generated arrivals and service time
  - Trace collected from real operations
    - Trace driven simulation
- Generating exponentially distributed random number
- Simulation code for M/M/1 and trace driven simulation

References

- Discrete event simulation of single-server queue
  - Winston, “Operations Research”, Sections 23.1-23.2
  - Law and Kelton, “Simulation modelling and analysis”, Section 1.4
- Generation of random numbers
  - Raj Jain, “The Art of Computer Systems Performance Analysis”
    - Sections 26.1 and 26.2 on LCG
    - Section 28.1 on the inverse transform methods
- Note: We have only touched on the basic of discrete event simulations. For a more complete treatment, see
  - Law and Kelton, “Simulation modelling and analysis”
  - Harry Perros, “Computer Simulation Techniques: The definitive introduction”, an e-book that can be downloaded from
    - http://www4.ncsu.edu/~hp/files/simulation.pdf