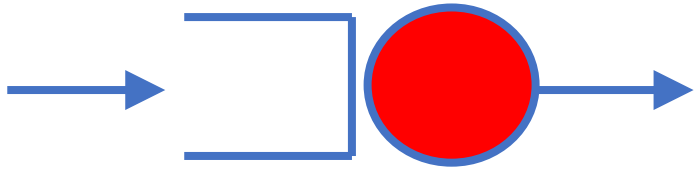
Birth-Death Processes

Birth-Death Model

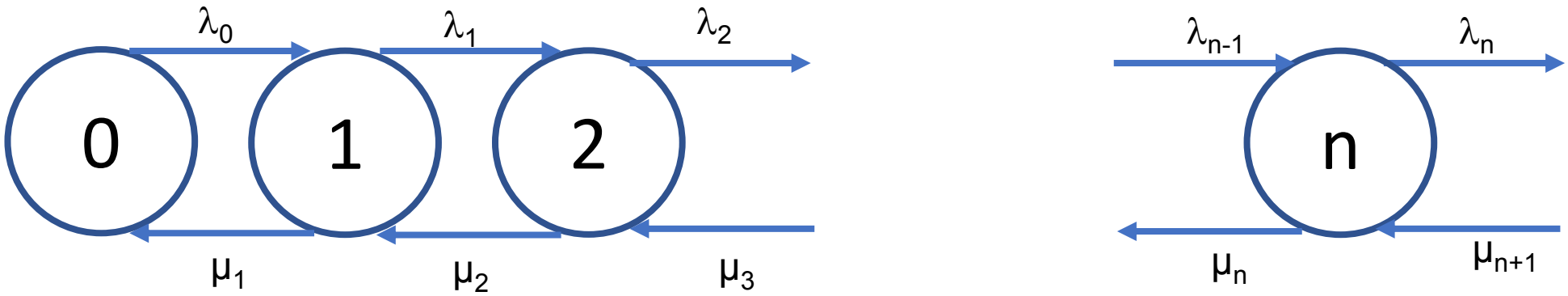


- Queueing system with one service facility (queue + server)
- State of system is number of jobs in system
- Interarrival times and service times exponentially distributed
- Birth-death process is a stochastic process where the number in system at each time t is a random variable n(t)
  - An arrival in state n is a birth and the state changes to n + 1
  - A departure in state n is a death and state changes to n - 1

Birth-Death Model

λ = arrival
μ = death

- Interarrival times in state n are exponentially distributed with interarrival rate λ_n
- Service times in state n are exponentially distributed with service rate μ_n



- State dependent arrival rate $\lambda_n = (N - n)\,\lambda$ is useful for modelling finite population and finite system capacity
- State dependent service rate $\mu_n = n\mu$ is useful for modelling multiple servers and infinite servers

Birth-Death Model – Steady State

- The steady state probability of a birth-death process being in state n is:

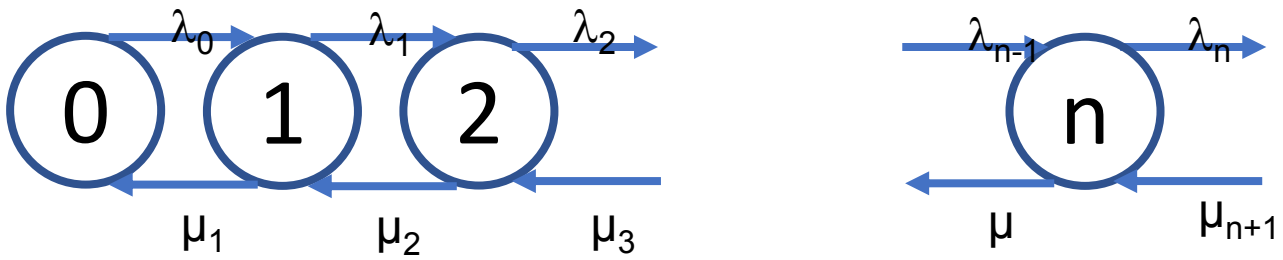
$$p_n = \frac{\lambda_0 \lambda_1 \lambda_2 \ldots \lambda_{n-1}}{\mu_1 \mu_2 \mu_3 \ldots \mu_n} p_0 \quad for\ n = 1, 2, \ldots, \infty$$

probability of ø state

$$Where\ p_0\ is\ the\ probability\ of\ being\ in\ the\ zero\ state$$

- This means that in the steady state, the likelihood of being in state n is the product of being in state zero, multiplied by all birth rates from states 0…n-1, divided by all death rates from states 1..n

Birth-Death Model – Balance Equations



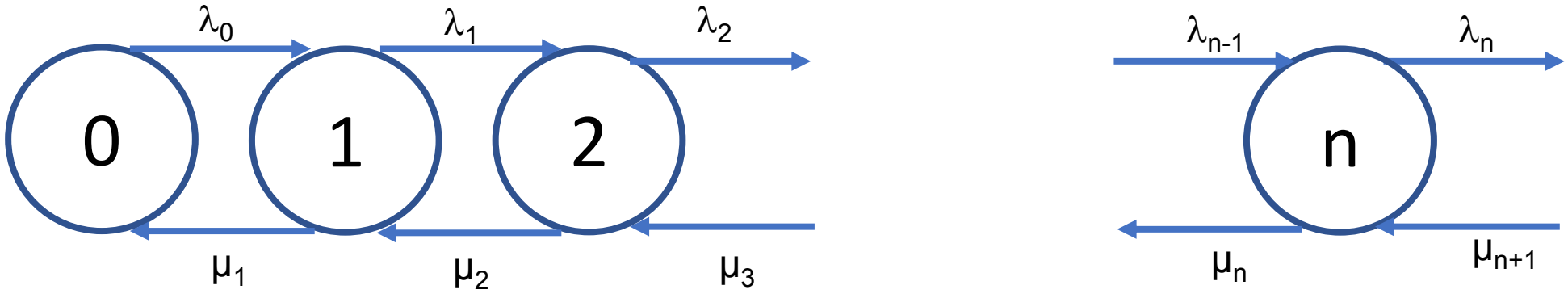
- When a balanced system is in steady state, the flow in from each state is equal to the flow out of the state.
  - So in state 0, the flow out is $\lambda_0 p_0$ and flow in is $\mu_1 p_1$ so $p_1 = \frac{\lambda_0}{\mu_1} p_0$
  - And for states n > 0, flow out is $(\lambda_n + \mu_n) p_n$ and the flow in is $\mu_{n+1} p_{n+1} + \lambda_{n-1} p_{n-1}$
  - So $p_{n+1} = \left(\frac{\mu_n + \lambda_n}{\mu_{n+1}}\right) p_n - \frac{\lambda_{n-1}}{\mu_{n+1}} p_{n-1} \quad for\ n > 0$

  Which leads to the steady state solution $p_n = \frac{\lambda_0 \lambda_1 \lambda_2 \ldots \lambda_{n-1}}{\mu_1 \mu_2 \mu_3 \ldots \mu_n} p_0 = p_0 \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}}$

  To get p_0 we note that the sum of all probabilities p_n is 1, so:

$$p_0 = \frac{1}{1 + \sum_{n=1}^{\infty} \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}}}$$

Birth-Death Model – Queuing Model



- $p_n = \frac{\lambda_0 \lambda_1 \lambda_2 \ldots \lambda_{n-1}}{\mu_1 \mu_2 \mu_3 \ldots \mu_n} p_0 = p_0 \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}}$
- $p_0 = \frac{1}{1 + \sum_{n=1}^{\infty} \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}}}$

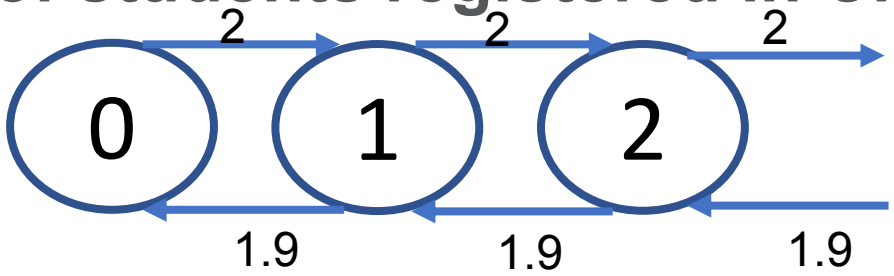
expected value

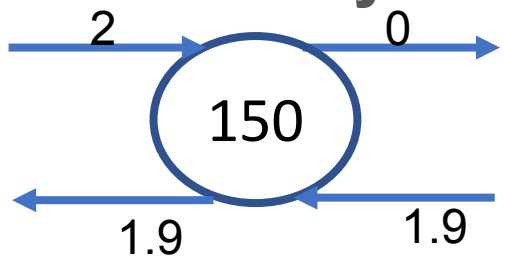
- The average number of customers in a queue is E(n) = $\sum_{n=0}^{\infty} n p_n$
- Mean arrival rate λ = $\sum_{n=0}^{\infty} \lambda_n p_n$
- Mean response time E(r) = E(n)/ λ (From Little's Law)

Exercise: Birth-Death Process

finite

- The number of students registering in CMPT 305 has an average registration rate of 2 students per day when the total registrants is between 0 and 150 then drops to 0 beyond 150. CMPT 305 has an average drop rate of 1.9 students per day. What is the average number of students registered in CMPT 305 in the steady state?





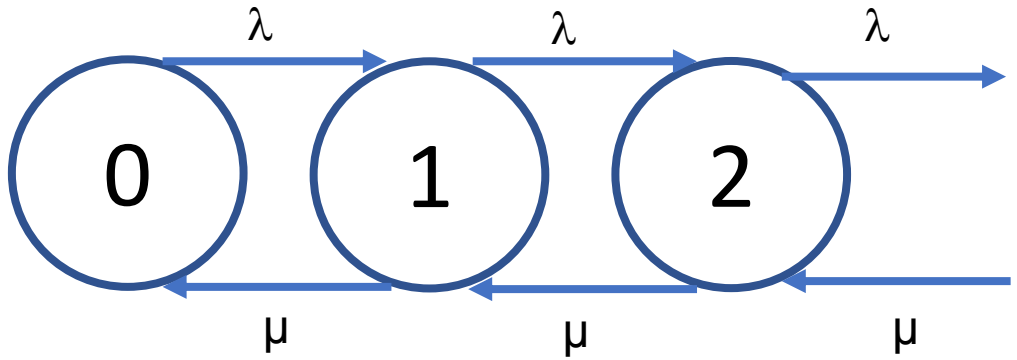
$E(n) = \sum_{n=0}^{\infty} np_n$

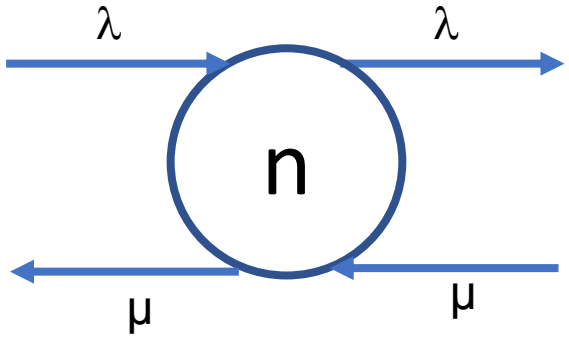
$$p_0 = \frac{1}{1+\sum_{n=1}^{\infty}\prod_{j=0}^{n-1}\frac{\lambda_j}{\mu_{j+1}}} = \frac{1}{1+\sum_{n=1}^{150}\prod_{j=0}^{n-1}\frac{2}{1.9}}$$ And $p_n = p_0\prod_{j=0}^{n-1}\frac{\lambda_j}{\mu_{j+1}} = \boldsymbol{p_0}\prod_{j=0}^{n-1}\frac{\mathbf{2}}{\mathbf{1.9}}$

Solution: 131 students

M/M/1 Queue

M/M/1 Queue

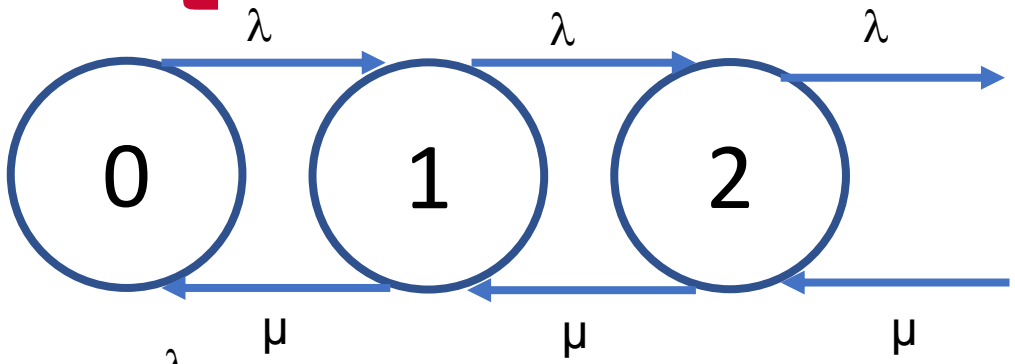


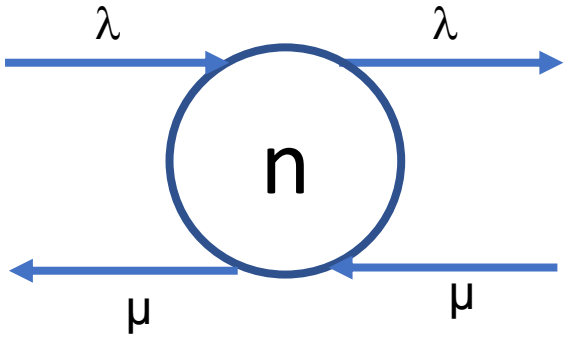


- Interarrival times and service times are both exponential.
- Infinite system capacity, infinite population
- Single Server
- Special case of birth-death model where $\lambda_n = \lambda$ for $n \geq 0$

  and $\mu_n = \mu$ for $n > 0$
- We can learn more information about M/M/1 performance metrics compared to general queues

M/M/1 Queue





Traffic Intensity $\rho = \frac{\lambda}{\mu}$ - system is stable if $\rho < 1$

- From birth-death equations:

$$p_n = p_0 \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}} = p_0 \left(\frac{\lambda}{\mu}\right)^n = p_0 \rho^n \quad for\ n = 1, 2, \ldots, \infty$$

$$p_0 = \frac{1}{1 + \sum_{n=1}^{\infty} \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}}} = \frac{1}{1 + \rho + \rho^2 + \cdots} = \frac{1}{\sum_{n=0}^{\infty} \rho^n} = \frac{1}{(1/(1-\rho))} = 1 - \rho \quad for\ \rho < 1$$

Maclaurin Series

M/M/1 Queue: Occupancy

$\rho = \frac{\lambda}{\mu}$, utilization

$p_n = (1-\rho)\rho^n \quad for\ n > 0$

Average number of customers in system

$E[n] = \bar{n}$

$$= \sum_{n=1}^{\infty} np_n = \sum_{n=1}^{\infty} n(1-\rho)\rho^n = (1-\rho)\sum_{n=1}^{\infty} n\rho^n = (1-\rho)\frac{\rho}{(1-\rho)^2} = \frac{\rho}{(1-\rho)}$$

$$Var[n] = E(n^2) - (E[n])^2 = \sum_{n=1}^{\infty} n^2(1-\rho)\rho^n - \frac{\rho^2}{(1-\rho)^2} = \frac{\rho}{(1-\rho)^2}$$

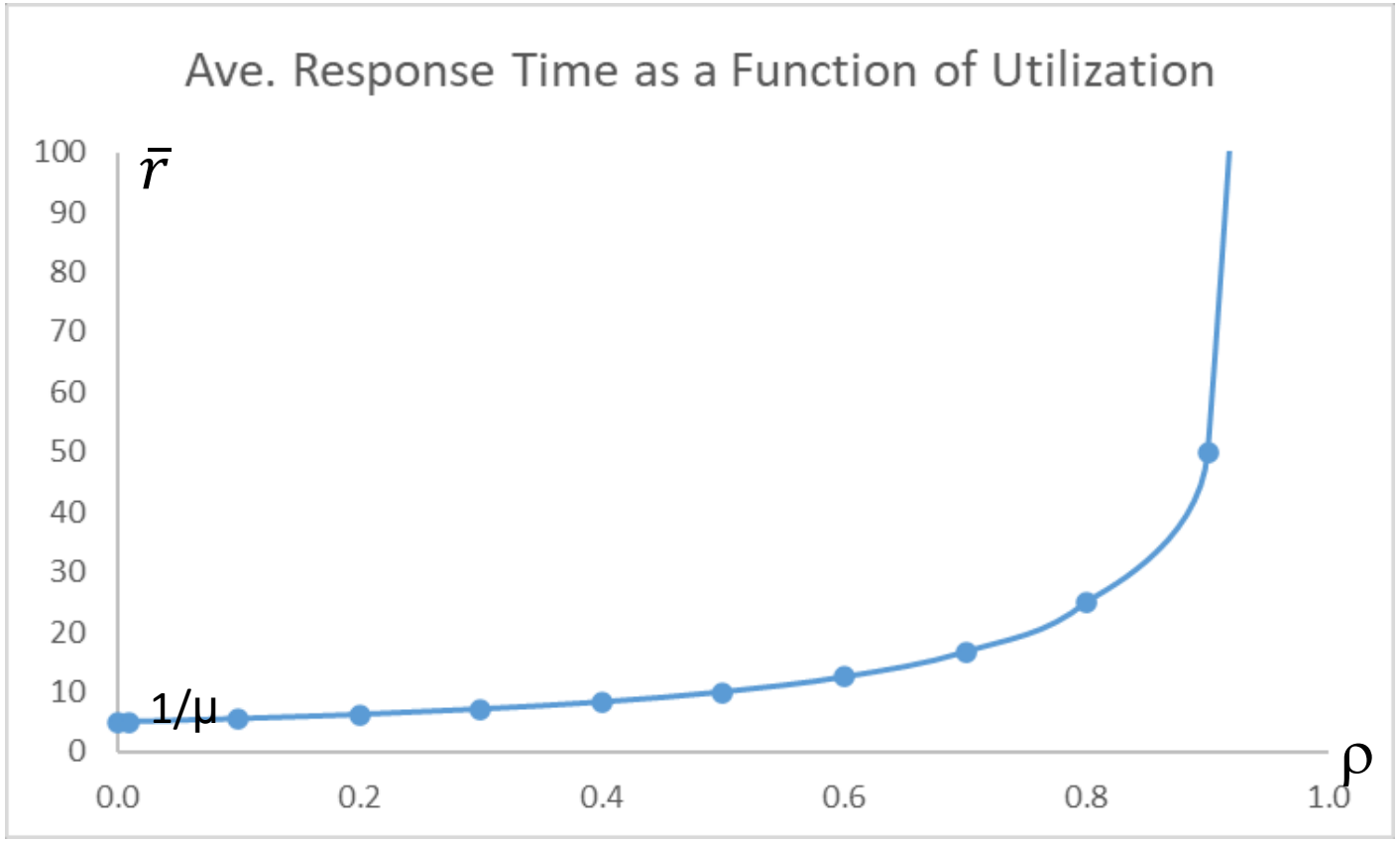
M/M/1 Queue: Response Time

Mean Response time (from Little's Law): $\bar{r} = \frac{\bar{n}}{\lambda} = \frac{\rho}{(1-\rho)\lambda} = \frac{1}{\mu(1-\rho)}$

throughput

Utilization = Fraction of time that system has at least 1 customer

$= 1 - p_0 = \rho$



M/M/1 Queue: Performance Metrics

Number of customers waiting in queue (all customers except the one being served)

$$\overline{n_q} = \sum_{n=1}^{\infty}(n-1)p_n = \sum_{n=1}^{\infty}(n-1)(1-\rho)\rho^n = (1-\rho)\sum_{n=1}^{\infty}(n-1)\rho^n = (1-\rho)\frac{\rho^2}{(1-\rho)^2} = \frac{\rho^2}{(1-\rho)}$$

Waiting time (From Little's Law): $\overline{w} = \frac{\overline{n_q}}{\lambda} = \frac{\rho}{\mu(1-\rho)}$

arrival rate = throughput

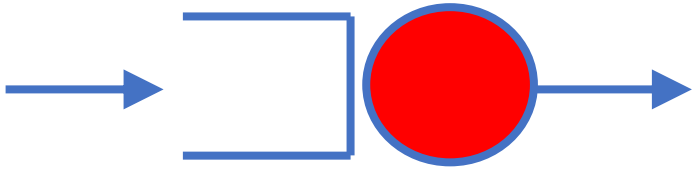
Probability of having at least n jobs in system = $\sum_{j=n}^{\infty} p_j = \sum_{j=n}^{\infty}(1-\rho)\rho^j = \rho^n$

Probability of having between m and n jobs in the system (m<n)

= Pr[having at least m jobs]-Pr[having at least n+1 jobs in the system]

= $\rho^m - \rho^{n+1}$

M/M/1 Queue: Example 1



- A convenience store has one checkout clerk. Both interarrival times and service times are exponentially distributed. On average, 20 customers arrive per hour to the store, and each customer takes 2 minutes on average to check out. What is the average number of customers in the store?

This is an M/M/1 queue with λ=20/hr and μ = ½ per minute = 60/2=30/hr

$$\rho = \frac{\lambda}{\mu} = \frac{20}{30} = \frac{2}{3} < 1$$ ← stable system!

μ ↑ service rate = 1/service time

$$E[n] = \frac{\rho}{(1-\rho)} = \frac{2/3}{1-2/3} = 2$$

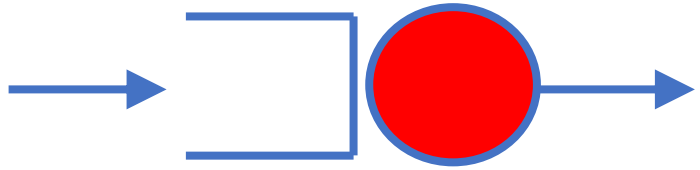
What percentage of time is the clerk busy? → utilization

$$U = \rho = \frac{2}{3} = 66.667\%$$ } percentage of time the server is busy

How much time is any customer waiting in line on average?

$$\bar{w} = \frac{\rho}{\mu(1-\rho)} = \frac{2/3}{30.(1-2/3)} = \frac{2}{30} hrs = \frac{(2)(60)}{30} = 4\ minutes$$

M/M/1 Queue: Example 2



- A small airport has one runway for all arriving flights. A plane cannot land in the runway until the previous plane has left the runway. Flight inter-arrival times are exponentially distributed with a mean of 4 minutes. The time from when a plane lands to when it leaves the runway is exponentially distributed with a mean of 3 minutes. What is the average number of planes waiting in the air to land?

m/m/1 queue

This is an M/M/1 queue with λ=1/4 per minute and μ = 1/3 per minute

↳ arrival rate = 1/mean interval time

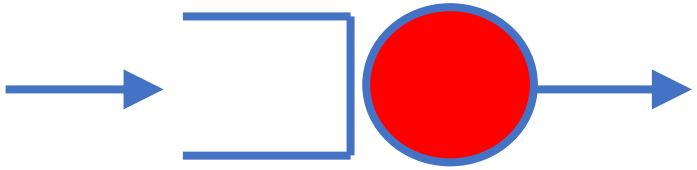
$$\rho = \frac{\lambda}{\mu} = \frac{1/4}{1/3} = 0.75 < 1$$ } stable system

$$\overline{n_q} = \frac{\rho^2}{(1-\rho)} = \frac{0.75^2}{1-0.75} = 2.25$$ So 2.25 planes are waiting in the air on average

How much time does each plane wait in the air, on average, before it lands?

$$\overline{w} = \frac{\rho}{\mu(1-\rho)} = \frac{0.75}{(1/3)(1-0.75)} = 9\ minutes$$

M/M/1 Queue: Example 3



- In a web server with a single CPU, the time between requests is exponential with an average arrival rate of 1,000 requests per second. The time to serve each request is exponential with an average service rate of 10,000 requests per second. What is the average response time for each request?

exponential arrival, exponential departure, single server

This is an M/M/1 queue with λ=1,000 req/sec and μ = 10,000 req/sec

$$\rho = \frac{\lambda}{\mu} = \frac{\mathbf{1000}}{\mathbf{10000}} = \mathbf{0.1 < 1}$$

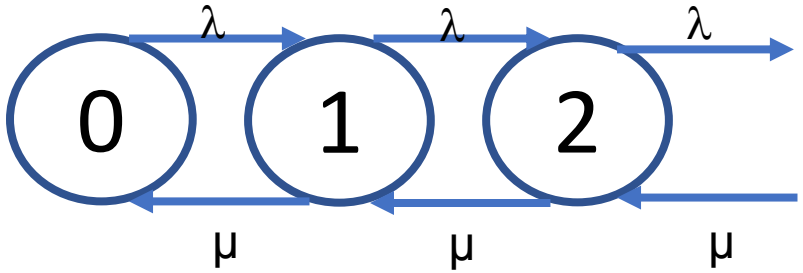
$$\bar{r} = \frac{1}{\mu(1-\rho)} = \frac{1}{(10000)(1-0.1)} = 0.000111\,\text{sec} = 0.111\,ms$$

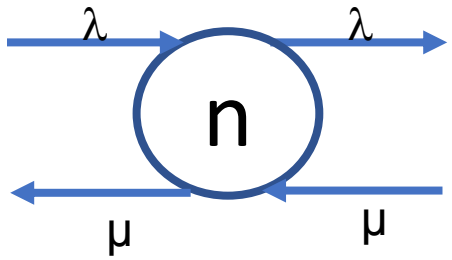
What is the probability of having between 2 and 4 requests getting served or waiting for service?

$$\Pr[\text{m} \leq requests \leq n] = \rho^m - \rho^{n+1}$$

$$\Pr[2 \leq requests \leq 4] = 0.1^2 - 0.1^5 = 0.00999$$

M/M/1 Queue Summary





Traffic Intensity $\rho = \frac{\lambda}{\mu}$ (λ → arrival rate / throughput; μ → service rate) - system is stable if $\rho < 1$

$p_0 = 1 - \rho \ \ for\ \rho < 1$ (Probability of having 0 customers in system)

$p_n = (1-\rho)\rho^n$ (Probability of having n customers in system)

$\boldsymbol{U} = \rho$ (Utilization)

$E[n] = \bar{n} = \frac{\rho}{(1-\rho)} \quad and \quad Var[n] = \frac{\rho}{(1-\rho)^2}$ (Average number in system and its variance)

$E[r] = \bar{r} = \frac{1}{\mu(1-\rho)}$ (Average response time)

$\overline{n_q} = \frac{\rho^2}{(1-\rho)}$ (Average number of customers waiting in queue)

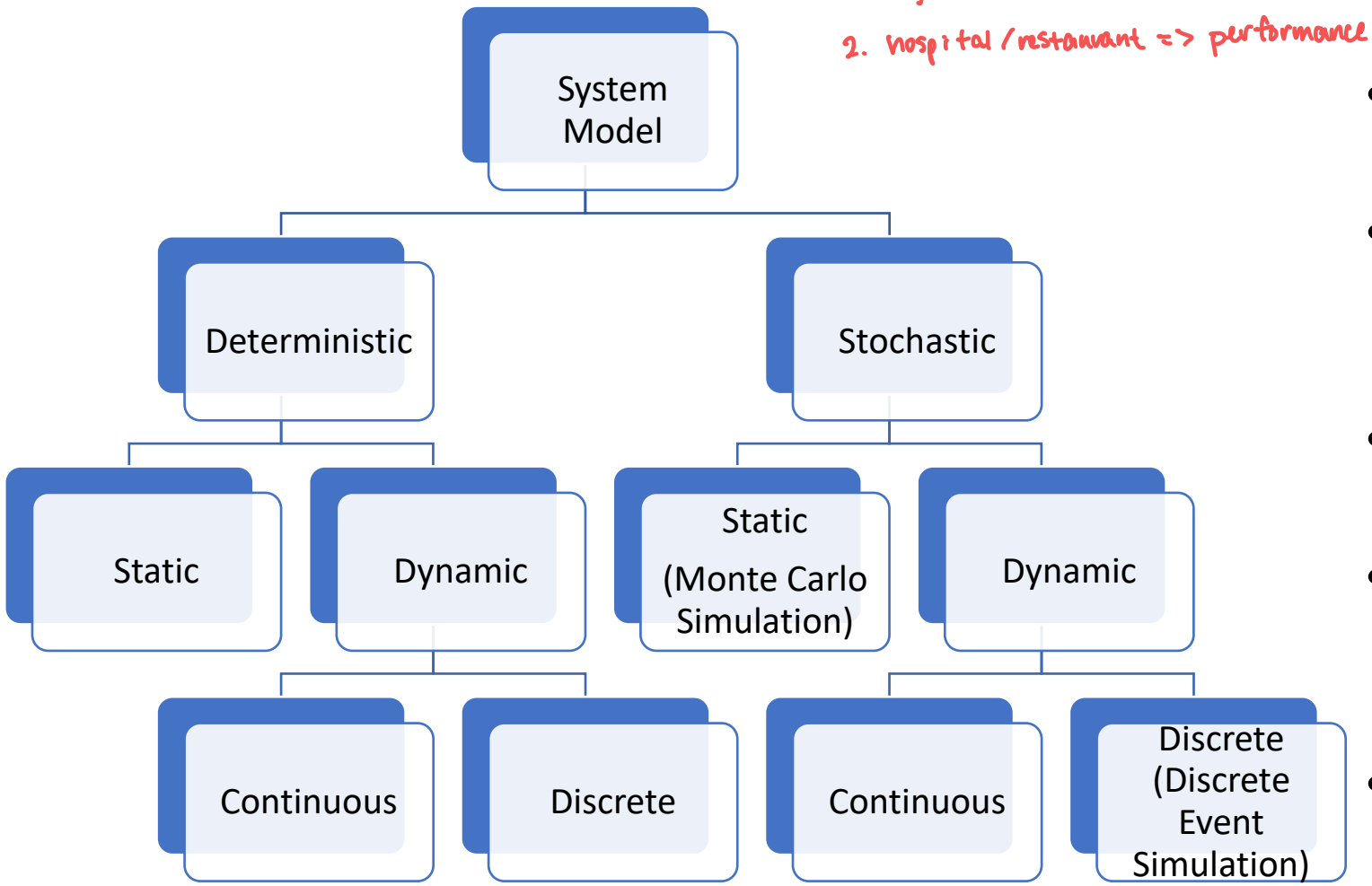
$\bar{w} = \frac{\rho}{\mu(1-\rho)}$ (Average wait time, i.e., queueing delay)

Introduction to Simulation

Simulation Taxonomy

Simulation
↳ a way of modelling a system given inputs and given some inputs, predicting the outcome of that system
1. functional
2. performance

1. flight simulator => functional
2. hospital / restaurant => performance



- **Deterministic:** Output can be predicted with certainty
- **Stochastic:** Part of the system state is random
- **Static:** Time is not a significant variable in simulation
- **Dynamic:** System state changes with time
- **Discrete State:** System state changes at discrete time instants → can only change as a function of something
- **Continuous State:** System state can change continuously

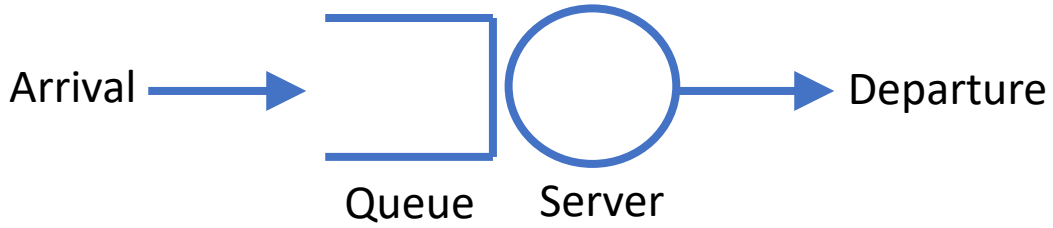
Simulation Models

- **Deterministic:** Output can be predicted with certainty vs. **Stochastic:** Part of the system state is random
- **Static:** Time is not a significant variable in simulation vs. **Dynamic:** System state changes with time
- **Discrete State:** System state changes at discrete time instants vs. **Continuous State:** System state can change continuously.
- **Discrete Time:** System state only defined at particular times vs. **Continuous Time:** System state defined at all times
- **Linear:** Output parameters are a linear function of input parameters vs. **Non-Linear:** otherwise
- **Open:** Input external to model and independent of it vs. **Closed:** No external input
- **Stable:** System achieves steady state independent of time vs. **Unstable:** Continuously-changing behavior.

Terminology

- Entity: Component of system that is explicitly represented in model
- Attribute: Property of an entity
- System state: Set of state variables that completely describes the system at a given point in time
- Event: An occurrence that changes the system state
- Activity: Period of time started and ended by related events
- Monte Carlo Simulation: Static stochastic simulation – used to model probabilistic phenomena for which the time evolution of state variables is unimportant
- Trace-Driven Simulation: Simulation that uses a time-ordered record of events on a real system (a trace) as input

Example: Single-Server Queue



- Entities: server, customer, system
- Attributes: service rate (server), service requirement (customer), interarrival time (system)
- System state: state of server (busy or idle), number of customers in system
- Event: arrival, start service, departure
- Activity: waiting in queue, receiving service
- HW 3 uses a trace-driven simulation from a randomly generated time-ordered record of arrival and service time events.

Discrete-Event Simulation

Components of a Discrete-Event Simulation

most important

1. Event Scheduler: Maintains list of events scheduled to occur in future. List usually implemented as a priority queue
   - Schedule event X at time t
   - Hold event X for time interval t
   - Cancel previously scheduled event X
   - Hold event X indefinitely (until scheduled by another event)
   - Schedule indefinitely held event X

scheduling events to simulate

2. Simulation Clock: Global variable for simulated time that is advanced by Event Scheduler. Two approaches:
   - Time-driven (unit time): Advance time by small increment and check for events that occur during the increment
   - Event-driven: Advance clock to earliest time that a scheduled future event occurs
3. State Variables: Global variables (number in system or queue)
4. Event Routines: One for each type of event. Updates system state variables and statistical counters, schedules other events
5. Sets: Collections of associated entities (e.g. jobs in queue)

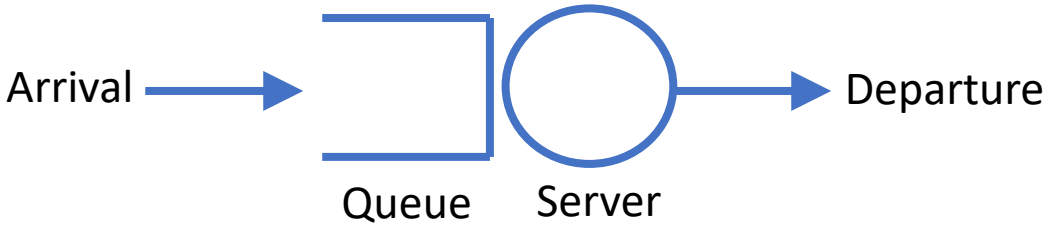
Components of a Discrete-Event Simulation (Cont.)

6. Input Routines: Get values of model parameters from user (mean service times, number of repetitions, termination conditions). Can specify several iterations (different sets of parameter values), each for several repetitions
7. Initialization Routines: Routines to initialize clock, state variables, statistical counters, and event list at beginning of simulation, beginning of iteration, and beginning of repetition
8. Report Generator: Calculate and output final statistics
9. Trace Routines: Debugging routines
10. Dynamic Memory Management: Garbage collection, free memory after use (automatic in some languages)
11. Main Program: Coordinates other routines: identify most imminent future event, advance clock to time of event, call event routine, repeat until termination conditions met

Trace-Driven Simulation

- **Trace:** Time-ordered record of events on a real system
  - Use trace as simulation input to explore trade-offs and different algorithms for a similar system
  - Trace must be independent of system under study (must not depend on any parameters that will be varied)
- Advantages
  - Credibility: more convincing than randomly-generated events
  - Validation: compare performance of real system to simulation
  - Accurate workload: preserves correlations and interactions
  - Fair comparison: compare alternatives with same input
- Disadvantages
  - Representativeness: Trace of a system might not be a good model of a different system or same system at different time
  - Single point of validation: algorithm for one trace might not be good for another trace $\Rightarrow$ Use several traces
  - Trade-offs: Difficult to change workload characteristics

Discrete-Event Simulation Example 1: Single-Server Queue

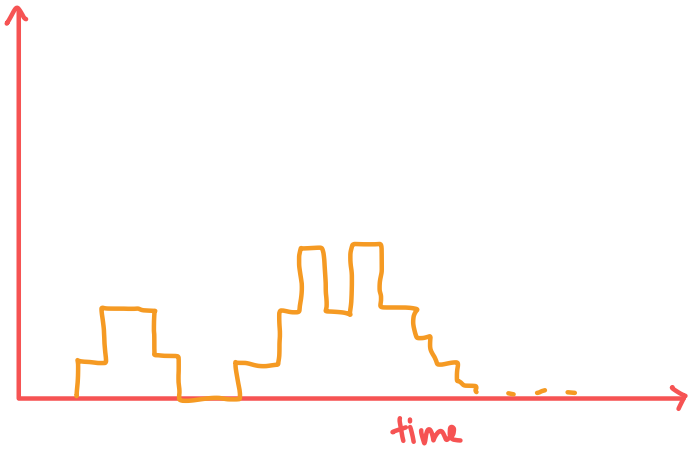


- Assumptions: (Similar to HW3)
  - Interarrival times independent of system state and IID (independent of each other and identically distributed)
  - Service times independent of system state and IID
  - FCFS scheduling
  - System empty at time 0
  - First customer arrives after first interarrival time
  - Simulation terminates when D customers depart

Discrete-Event Simulation Example: Single-Server Queue (Cont.)

- Input Parameters:
  - Interarrival time distribution
  - Service time distribution
  - Termination condition D
- Performance Measures:
  - Mean waiting time: w
  - Mean number in system: n
- State Variables:
  - clock: simulation clock
  - Server Status: busy or idle
  - n: number of customers in system

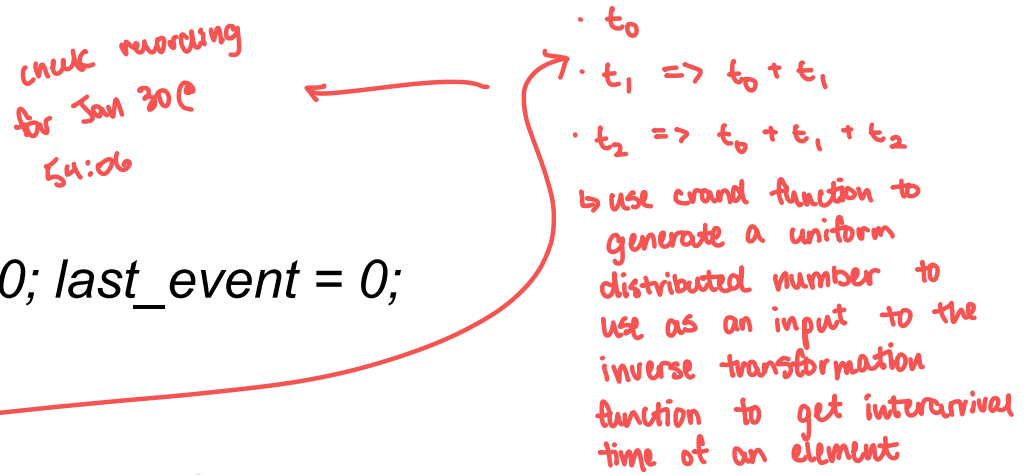
Discrete-Event Simulation Example: Single-Server Queue (Cont.)



- total number of customers multiplied by the time interval = response time

- Statistical Counters:
  - *nd*: number of departing customers
  - *sw*: sum of accumulated waiting times
  - *sa*: sum of accumulated areas (for calculating $\bar{n}$)
  - *last_event*: time of most recent event when accumulating *sa*
- Sets:
  - event list: list of future events
  - queue: customers in queue ordered by arrival time
- Event Types (and Event Routines):
  - Type 1: arrival
  - Type 2: start service
  - Type 3: departure

Single-Server Queue (Cont.): Simulation Program



- Initialization:
  - *clock = 0; status = idle; n = 0; nd = 0; sw = 0; sa = 0; last_event = 0;*
  - Initialize *queue* and *event_list* to empty
  - Fill up queue with *D* elements
  - Generate *D* inter-arrival times from inverse transformation of exponential dist., use these inter-arrival times to fill in arrival time of each of the *D* elements in queue
  - Generate *D* service times from inverse transformation of exponential distribution, use these service times to fill in service times of each of the *D* elements in queue
  - Determine first arrival time (*at1* of the head of queue). Schedule type 1 event at *clock+at1*
- Main program:
  - Remove earliest event from *event_list* (event type *i* at time *t*)
  - Set *clock = t; sa = sa+ (clock – last_event) * n; last_event = clock;*
  - Invoke event routine for type *i* event
  - Repeat previous steps until *nd=D* departures
  - Run Report Generator to print statistics (*mean waiting time = sw/nd;* $\bar{n}$*=sa/clock*)

Single Server Queue (Cont.): Event Routines

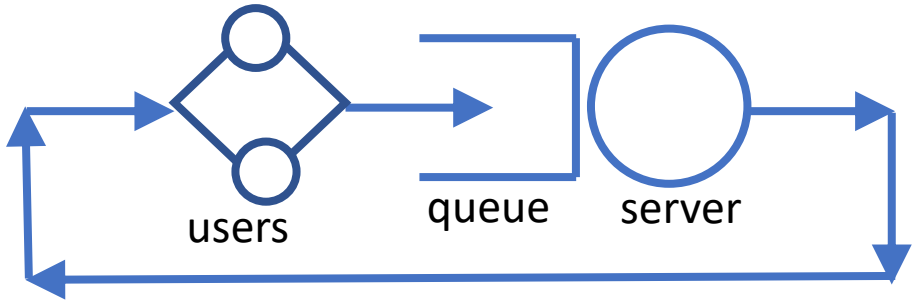
- Event Routine for type 1 event (arrival):
  - Determine arrival time of next element in queue, schedule an arrival event at that time
  - *n=n+1*
  - If *status* is idle; invoke routine for type 2 event (start service)
- Event Routine for type 2 event (start service):
  - Remove customer at the head of the queue, note its arrival_time and service_time, advance head to next
  - *sw = sw + (clock – arrival_time); Status = busy*
  - Schedule departure (type 3) event at time *clock+service_time*
- Event Routine for type 3 event (departure):
  - *n=n-1; Status = idle; nd = nd+1*
  - If *n>0,* invoke routine for start service (type 2) event

busy when servicing an element

arrival : 1s
start service: 5s
departure : 7s

wait time = 4s => sw++

Discrete-Event Simulation Example 2: Finite Population System



- Assumptions:
  - Think times are independent of system state and IID
  - Service times independent of system state and IID
  - FCFS scheduling
  - System empty at time 0
  - First request of each user submitted after the user's first think time
  - Simulation terminates at time *time_end*

Discrete-Event Simulation Example: Finite Population System (Cont.)

- Initialization:
  - *clock=0; status=idle; n=0*
  - Initialize *queue* and *event_list* to empty
  - For each user *j*, $1 \leq j \leq N$, determine a think time *think_t* and schedule an arrival event at time *clock+think_t*
  - Schedule an *end_simulation* event at time *time_end*
- Main program:
  - Remove earliest event from *event_list* (event type *i* at time *t*)
  - Set *clock = t; last_event = clock;*
  - Invoke event routine for type *i* event
  - Repeat previous steps until *event = end_simulation*
  - Run Report Generator to print statistics

Finite Population System (Cont.): Event Routines

- Event Routine for arrival
  - *n=n+1*
  - Add arriving user to *queue*
  - If *status* is idle; invoke routine for start service event
- Event Routine for start service:
  - Remove user at the head of the queue
  - *status = busy*
  - Determine service time of user *service_time*
  - Schedule departure event at time *clock+service_time*
- Event Routine for departure:
  - *n=n-1; Status = idle*
  - Determine the departing customer's think time *think_time*
  - Schedule an arrival event at time *clock+think_time*
  - If *n>0,* invoke routine for start service event

Managing Event List

- A significant part of discrete-event simulation is managing the event list
- Need to efficiently:
  - Schedule an event: Insert event in list
  - Determine next event: Find event with earliest start time (delete)
- A significant portion of the simulation is used to schedule events and determine next event, so we need an efficient data structure to manage events
- Linked list is not always the best option
  - For an ordered list, insertion is O(n) and deletion is O(1)
  - For an unordered list, insertion is O(1) and deletion is O(n)
  - For large numbers of events, either operation can become very inefficient
- Alternative data structures
  - Balanced Binary Search Tree: Both insertion and deletion are O(log n)
  - Heap: Insertion O(log n), deletion O(1)