

Advanced Systems Lab  
Tutorial IV  
Queuing Systems

G. Alonso

Systems Group

<http://www.systems.ethz.ch>

# Operational Laws

# Operational laws

- Operational laws are relationships that apply to certain measurable parameters in a computer system
- They are independent of the distribution of arrival times or service rates
- The parameters are observed for a finite time  $T_i$ , yielding operational quantities:
  - Number of arrivals  $A_i$
  - Number of completions  $C_i$
  - Busy time  $B_i$

# Job flow balance

- The job flow balance assumption implies:
- Number of arrivals  $A_i$  (jobs arriving during  $T_i$ )
- Number of completions  $C_i$  (jobs completed during  $T_i$ )

$$A_i = C_i$$

- Correct use heavily depends on semantics of “jobs completed”

# Derivations

- Arrival rate  $\lambda_i = A_i / T_i$
- Throughput  $X_i = C_i / T_i$
- Utilization  $U_i = B_i / T_i$
- Mean Service Time  $S_i = B_i / C_i$
- These are variables that change from observational period to observational period
- Some relationships hold for each observational period

# Utilization Law

$$U_i = \frac{B_i}{T_i} = \frac{C_i}{T_i} \times \frac{B_i}{C_i} = X_i \cdot S_i$$

The utilization law is simply based on the observation that the service time indicates how long the server needs to complete a job. Hence, the utilization of the server is the number of jobs completed multiplied the time that it takes to complete each one of them

# Example Utilization Law

- A disk serves 40 requests per second. Each request requires 0.0225 seconds
- Utilization:  $40 \times 0.0225 = 90\%$
- Instrumentation shows that a disk is busy 80% of the time when an application runs. We know that the application produces 20 requests per second
- The time taken per request is 40 ms

# Forced Flow Law

- Assume a closed system several devices are connected as a queuing network. The number of completed jobs is  $C_0$
- Assume job flow balance
- If each job makes  $V_i$  (visit ratio) requests of the  $i_{\text{th}}$  device, then  $C_i = C_0 V_i$

$$X_i = \frac{C_i}{T_i} = \frac{C_i}{C_0} \times \frac{C_0}{T_i} = X V_i$$



# Bottleneck device

- Utilization of the device

$$U_i = X_i S_i = X V_i S_i$$

With the demand for the device  $D_i = V_i S_i$

$$U_i = X D_i$$

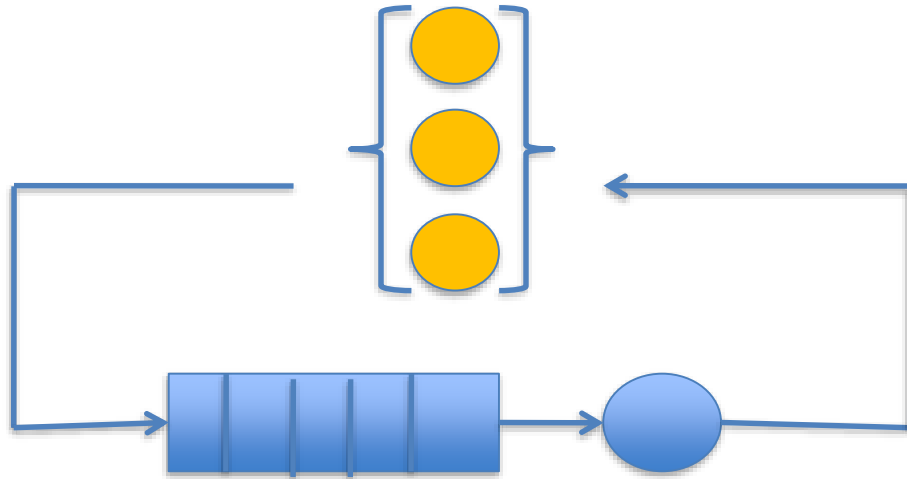
The device with the highest demand is the bottleneck device

# Little's Law

- Assuming job flow balance:
- $R_i$  is the response time of the device
- $Q_i$  is the number of jobs in the device
- $\lambda_i = X_i$  (arrival rate equals throughput)

$$Q_i = \lambda_i R_i = X_i R_i$$

# Interactive Response Time Law



- Users submit a request, when they get a response, they think for a time  $Z$ , and submit the next request

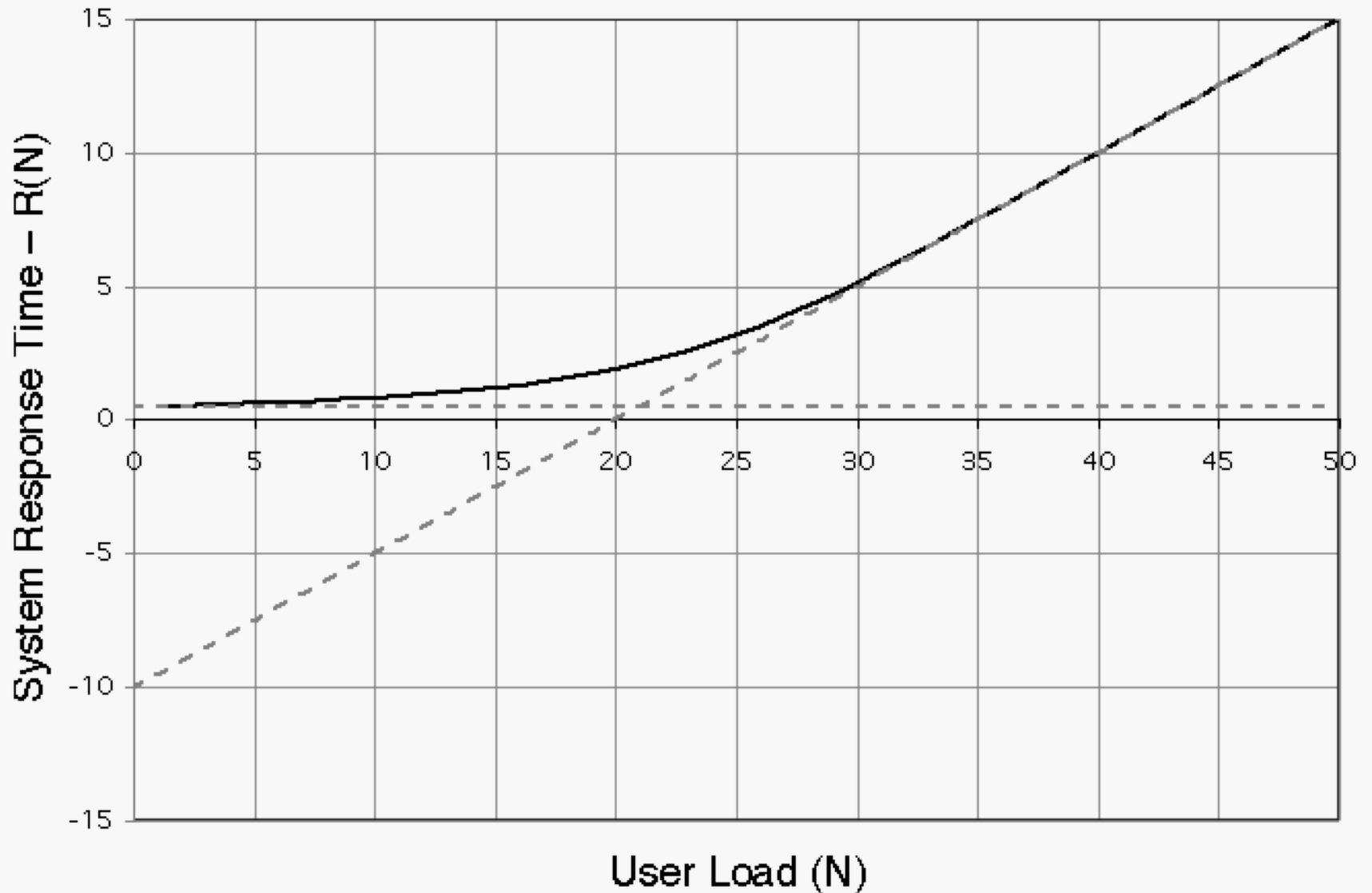
# Interactive, closed systems

- Response time  $R$
- Total cycle time  $R+Z$
- Each user generates  $T/(R+Z)$  requests in time  $T$
- There are  $N$  users

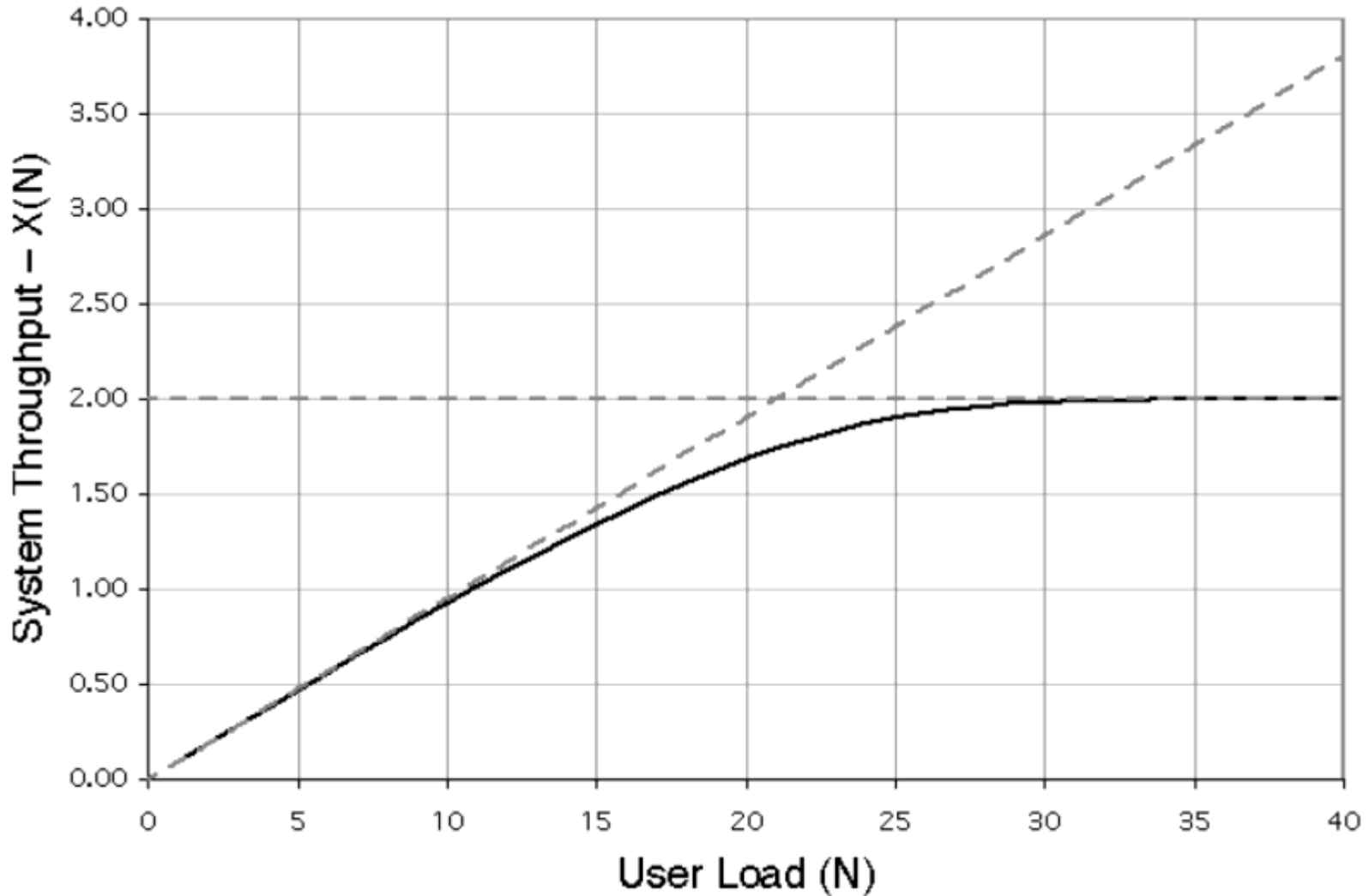
$$X = \frac{\text{Jobs}}{\text{Time}} = \frac{N \frac{T}{R+Z}}{T} = \frac{N}{R+Z}$$

$$R = \frac{N}{X} - Z$$

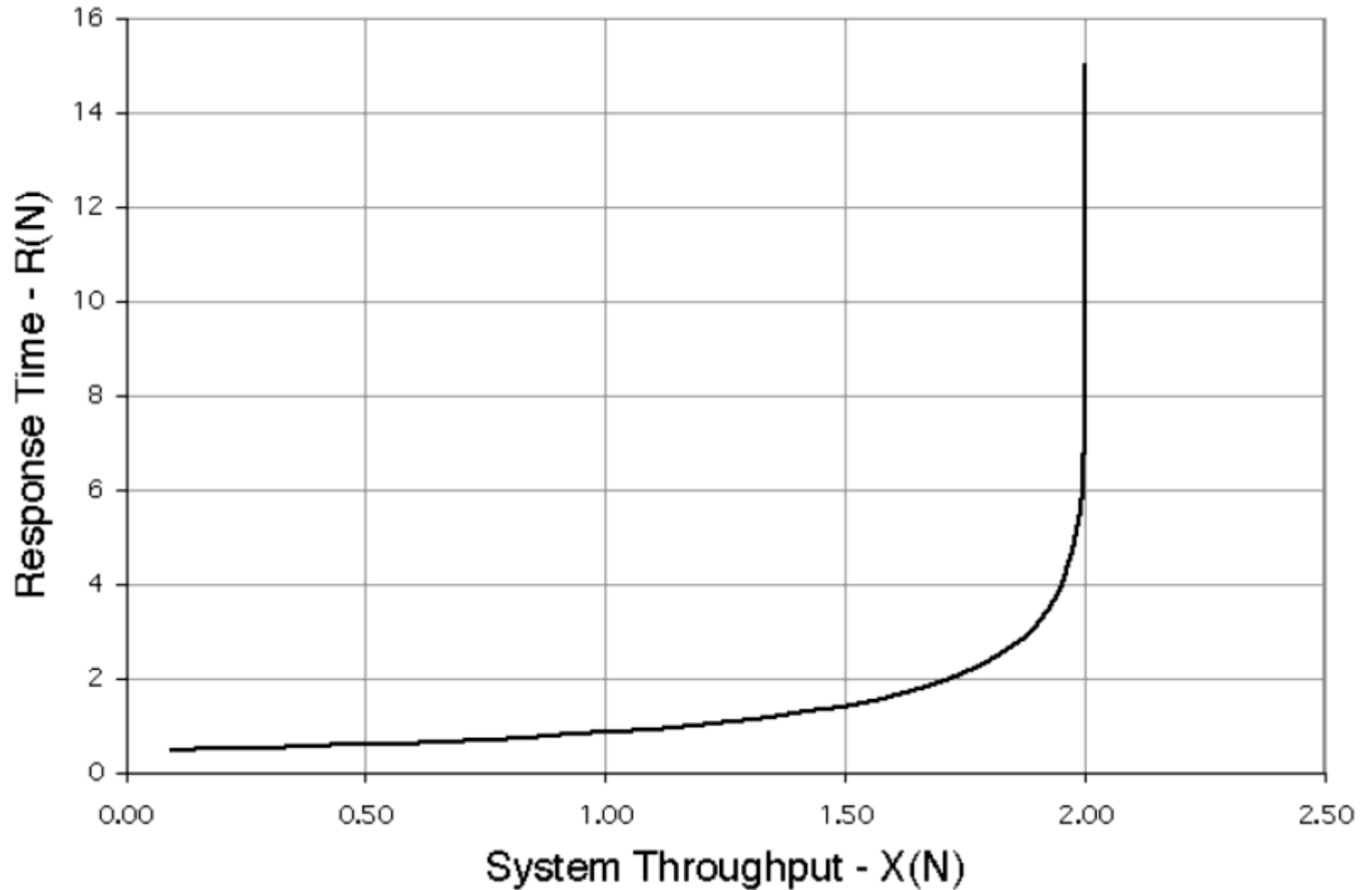
# Interactive Law in practice



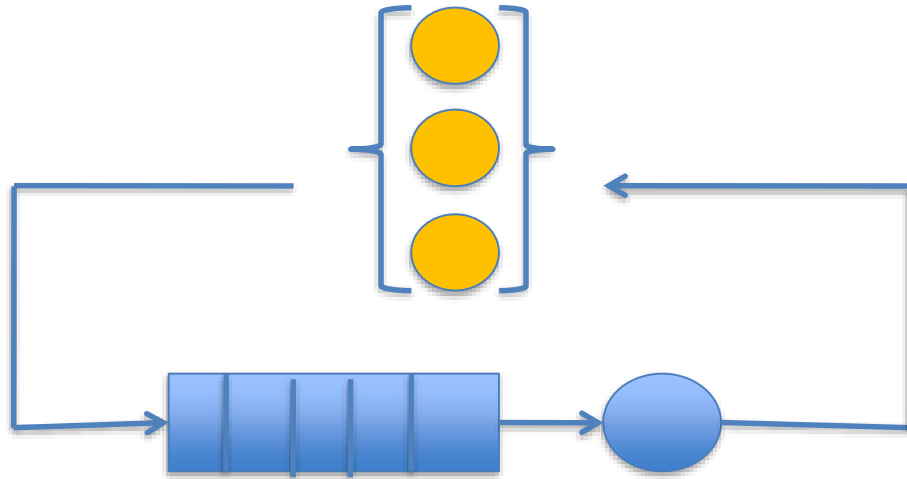
# Interactive law in practice



# Interactive law in practice



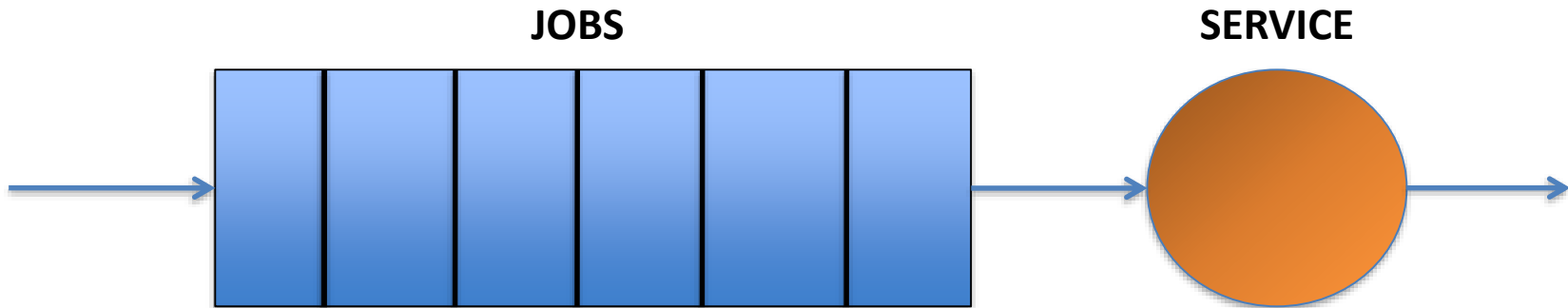
# Warning!



- Real systems are not ideal
  - Network effects
  - Processing overhead(not only wait)
  - Exceptions and not “normal” return values



# A queue



- Very useful model for many computer systems
- Basic principles of queuing theory
- Understand the limitations of queues as models

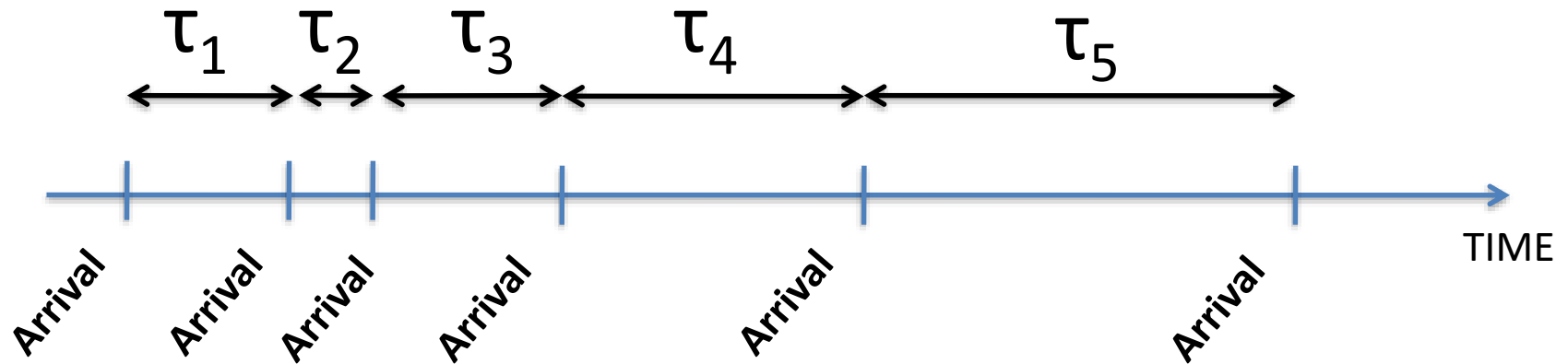
# Queuing system

# Characterizing a queuing system

- Arrival rate
  - Service time
  - Service discipline
  - System capacity
  - Number of servers
  - Population size
- A/S/m/C/P/SD
    - A = arrival distribution
    - S = service distribution
    - m = number of servers
    - C = buffer capacity
    - P = population size (input)
    - SD = service discipline
  - Notation not standardized

# ARRIVAL RATE DISTRIBUTION

# Interarrival time



- The interarrival times are assumed to form a sequence of Independent and Identically Distributed (IID) random variables
- Common assumption is a Poisson distribution

# Mean arrival rate

- Mean interarrival time =  $E[\tau]$
- Mean arrival rate:  $\lambda = 1 / E[\tau]$
- $\lambda$  is not a random variable!
- Examples;
  - a single client submits a query every 200 ms, then  $\lambda$  is 5 queries/second
  - 10 clients submit a query each every 500 ms, then  $\lambda$  is 20 queries/second
  - These are the queries submitted to the system

# Assumptions

- Queuing systems assume an arrival rate
  - state independent (does not depend on number of previous jobs)
  - stationary (does not change in time)
- These assumptions do not hold in real systems
  - Burstiness / batch jobs
  - Flash crowds (popularity)
  - Social effects (time of day variant load)

# **SERVICE RATE DISTRIBUTION**



# Service time per job

- The time it takes to process a job (only the time it takes to process it, not including the time it has been waiting in the queue) =  $s$
- Mean service rate:  $\mu = 1/E[s]$
- If there are  $m$  servers, mean service rate is  $m\mu$
- $\mu$  is not a random variable!
- Example:
  - Printer takes on average 20 seconds per job, then  $\mu = 0.05$  jobs/second = 3 jobs/minute

# Throughput

- Sometimes  $\mu$  is called the system's throughput
- Careful with the notion of throughput
- This is correct only in some cases
  - There are always jobs ready when a job is finished
  - No overhead in switching to new job
  - All jobs complete correctly
  - Service rate is state independent (does not depend on the number of jobs in the queue)
  - Service rate is stationary (does not change with time)

# **SERVICE DISCIPLINE**

# Queue discipline

- FCFS = First Come – First Served
  - Ordered queue
- LCFS = Last Come – First Served
  - Stack
- RR = Round Robin
  - CPU allocation to processes
- RSS = random
- Priority Based

# OTHER PARAMETERS

# System capacity

- The system (or buffer) capacity is the maximum number of jobs that can be waiting for service
- System capacity includes jobs waiting and jobs receiving service
- In reality = finite
- Analysis = assume infinite capacity
- Finite buffers very important in practice

# Number of Servers

- The service can be provided by one or more servers
- Assume work in parallel and independent
- Servers do not interfere with each other
- Total service rate is aggregation of each individual service rate

# Population

- The total number of potential jobs that can be submitted to the system:
- Analysis = assume infinite
- In practice:
  - Very large (assume infinite), e.g., number of clicks on a page
  - Finite, number of homework submissions for this lecture
  - Closed systems (output determines input)



# GENERAL RESULTS

**G/G/1    G/G/m**

# Offered load

- The offered load or traffic intensity is

$$\rho = \lambda / (m\mu) = \lambda \cdot E[s] / m$$

- The system is stable if

$$\rho < 1 \Rightarrow \lambda < m\mu$$

- In other words, the system is stable if the mean arrival rate ( $\lambda$ ) is less than the mean service rate ( $m\mu$ ), otherwise the queue grows without bounds

$$\rho = 1$$

- Unless arrivals and service are deterministic and exactly scheduled,  $\rho = 1$  does not lead to a steady system
  - Randomness prevents queuing from emptying
  - Server cannot catch up
  - Queue grows without bound
- One way to avoid this is flow control (drop jobs when load too high)

# Examples

Instrumentation shows that a disk is serving 50 I/O operations per second and the average I/O time is 10 ms. What is the disk utilization?

$$\rho = \lambda \cdot E[s]/m, \text{ with } m = 1$$

$$\rho = 50 \times 0.010 = 0.5 = 50\%$$

Application A generates about 50 I/O requests/s, if the disk is 85 % utilized, what is the average time needed for every I/O? ... 17ms

# Further examples

We have allocated 60% of the disk to one application. If we want to maintain an average response time for every I/O operations of 12 ms, what is the maximum number of I/O requests per second that the application can generate?

$$0.6 = \lambda \cdot 12 \text{ ms} \Rightarrow \lambda = 50 \text{ req/s}$$

# Some more notation

- $n = n_s + n_q$ , where
  - $n$  is the number of jobs in the system (queue)
  - $n_s$  is the number of job in the service
  - $n_q$  is the number of jobs waiting for service
- $w = w_q + s$ , where
  - $w$  is the total time in the system
  - $w_q$  is the time waiting in the queue
  - $s$  is the time in the service
- These are all random variables

# Little's Law

- For the queuing system:

$$E[n] = \lambda \cdot E[w]$$

- For the queue

$$E[n_q] = \lambda \cdot E[w_q]$$

- With

$$E[n] = E[n_q] + E[n_s]$$

$$E[w] = E[w_q] + E[s] = E[w_q] + 1/\mu$$

# Example

Instrumentation shows that the average time to respond to a request was 100 ms and the server received about 100 requests/second. If each active request requires 5 KB of memory, how much memory needs to be reserved for the average number of requests in the system?

Jobs in the system =  $100 \cdot 0.1 = 10$  jobs (Little's)

Memory needed =  $10 \cdot 5 \text{ KB} = 50 \text{ KB}$



# Jobs in service

- Using Little's Law, one can derive:

$$E[n_s] = \lambda / \mu = \lambda \cdot E[s]$$

- For a queuing system with  $m$  servers

$$E[n_s] = m \cdot \rho$$

that is, the average number of jobs in service is  $m$  times the arrival rate divided by the mean service rate

# **BIRTH-DEATH PROCESSES**

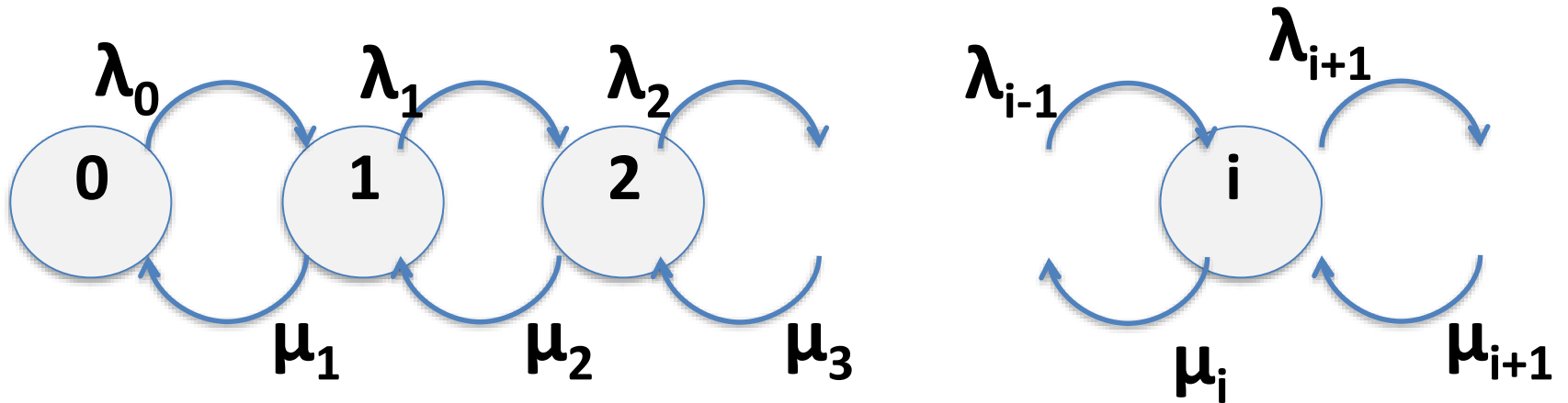
# Stochastic processes

- Many of the values in a queuing system are random variables function of time (e.g., the waiting time at a queue)
- Such random functions of times are called stochastic processes
- If the values a process can take are finite or countable, it is a discrete process or a stochastic chain

# Markov Processes

- If the future states of a process depend only of the current state and not on past states, the process is a Markov process
- Discrete Markov processes are Markov chains
- A Markov chain in which the transition between states is limited to neighboring states is called a birth-death process

# Steady state probability



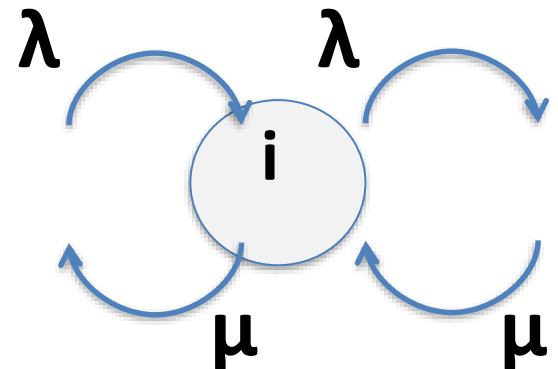
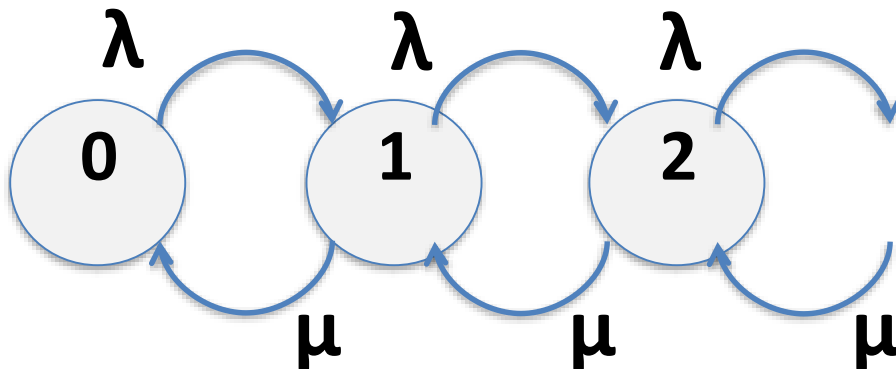
- Probability of being in state n is:

$$P_n = \frac{\lambda_0 \lambda_1 \dots \lambda_{n-1}}{\mu_1 \mu_2 \dots \mu_n} P_0$$

**M/M/1**

# M/M/1

- Memoryless distribution for arrival and service
- Single server
- Infinite buffers and FCFS
- Mean arrival rate:  $\lambda$
- Mean service rate:  $\mu$



# Basics M/M/1

- From the state probability of a birth-death process:

$$p_n = (\lambda/\mu)^n p_0, \quad n = 1, 2, \dots, \infty$$

- or

$$p_n = \rho^n p_0$$

- Since all probabilities must add to 1

$$p_0 = 1 - \rho \quad \text{and} \quad p_n = (1 - \rho) \rho^n$$



# Utilization

- Utilization: probability that there is one or more jobs in the system

$$U = 1 - p_0 = \rho$$

# M/M/1 behavior

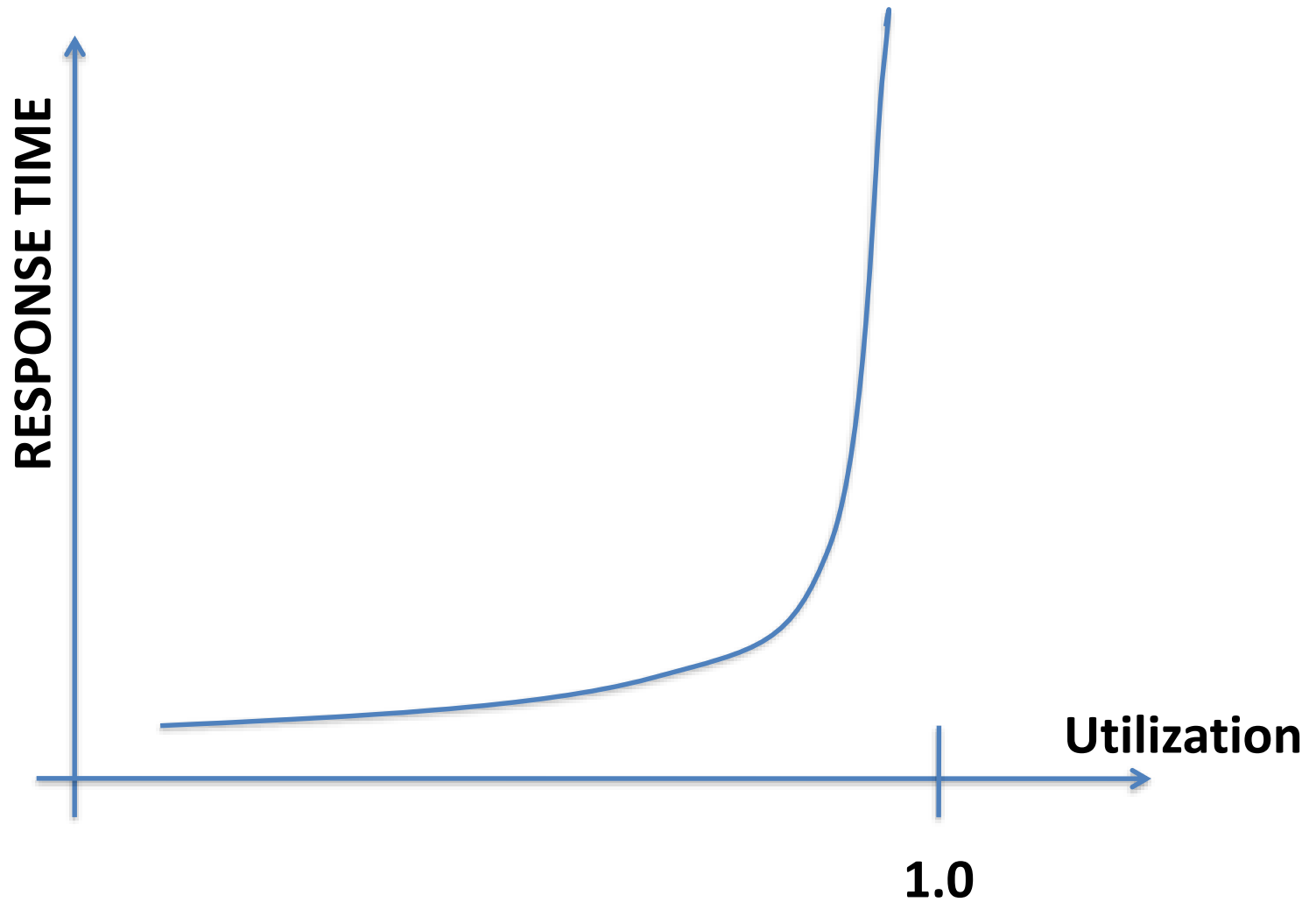
- The mean number of jobs  $E[n]$  is

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

- Applying Little's Law we get the response time

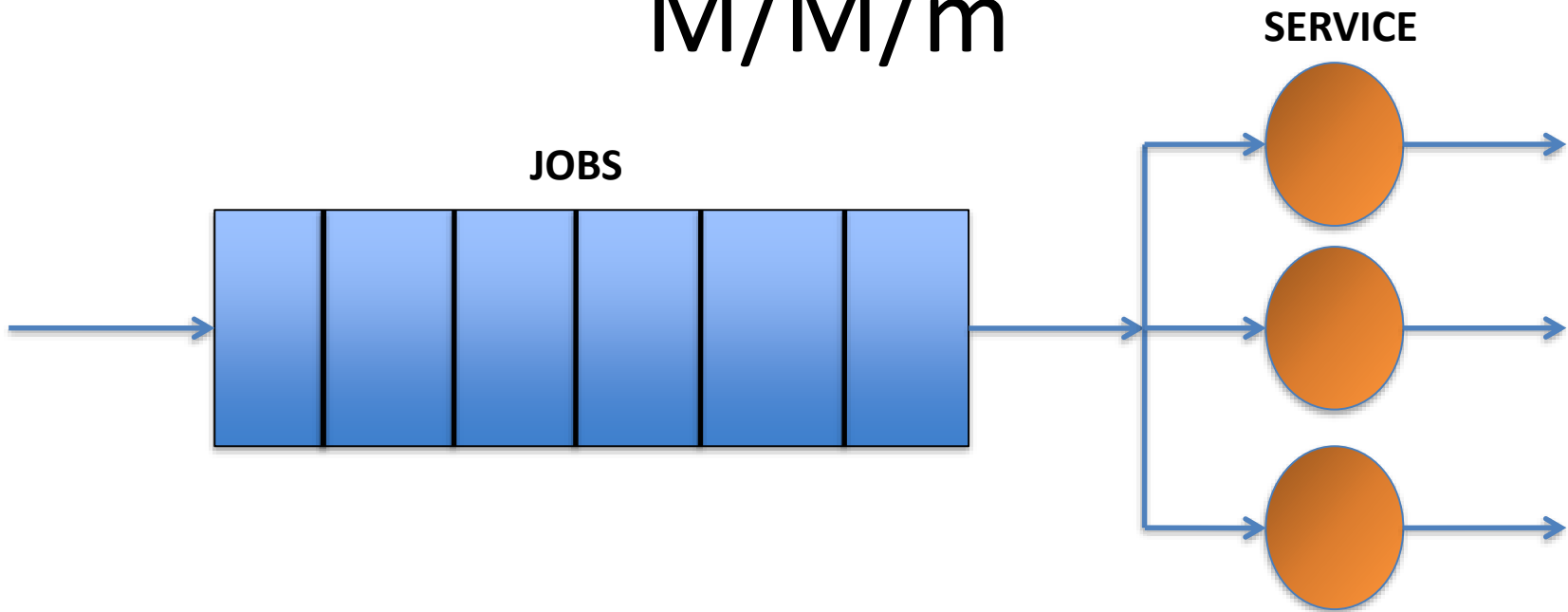
$$E[w] = \frac{1/\mu}{1-\rho}$$

# Response time in M/M/1



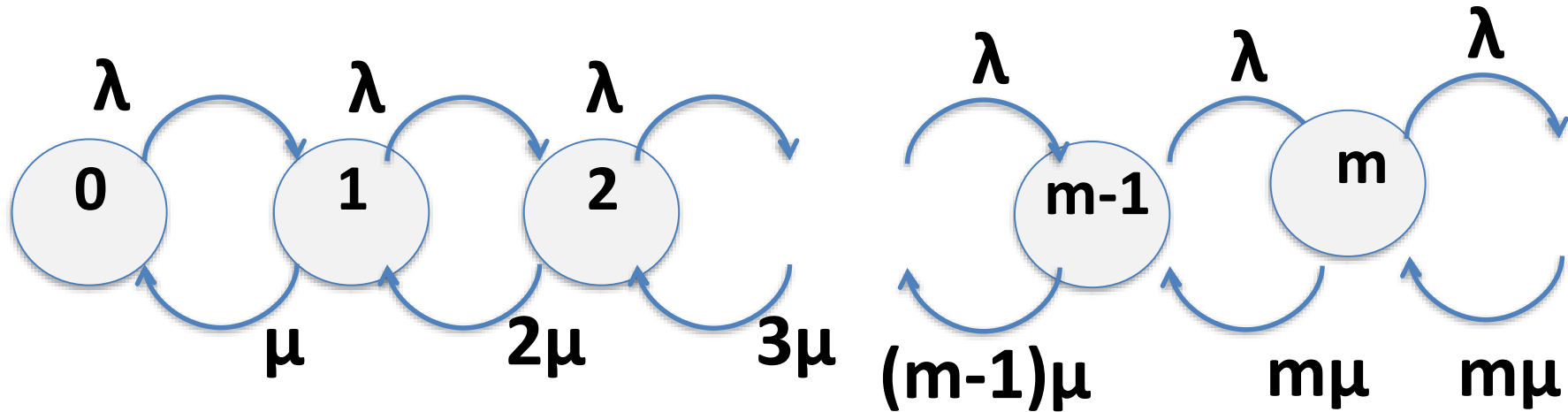
**M/M/m and M/M/m/B**

# M/M/m



- Each server serves  $\mu$  jobs per unit of time
- Jobs get service right away if less than  $m$  jobs in system, otherwise they wait in queue.

# Probabilities in M/M/m



- Number of jobs in a M/M/m system is a birth death process. Hence:

$$P_n = \frac{\lambda_0 \lambda_1 \dots \lambda_{n-1}}{\mu_1 \mu_2 \dots \mu_n} P_0$$

# Resolving for M/M/m

$$P_n = \frac{\lambda^n}{n! \mu^n} P_0 \quad \text{for } n = 0, 1, 2, \dots, m-1$$

$$P_n = \frac{\lambda^n}{m! m^{n-m} \mu^n} P_0 \quad \text{for } n = m, m+1, \dots, \infty$$

# With traffic intensity

- $\rho = \lambda / (m\mu)$

$$P_n = \frac{(m\rho)^n}{n!} P_0 \text{ for } n = 0, 1, 2, \dots, m-1$$

$$P_n = \frac{\rho^n m^m}{m!} P_0 \text{ for } n = m, m+1, \dots, \infty$$



# M/M/m queue

- The rest of the results for this system can be derived from the state probabilities (see in the text book)

# m times M/M/1 vs M/M/m

- What is better?
  - m queues of the form M/M/1 with arrival rate  $\lambda/m$
  - a single system of the form M/M/m with arrival rate  $\lambda$
- In general, M/M/m will be better because it leads to less waiting time (jobs waiting in a queue do not benefit if a server in another queue is free)

# M/M/m/B

- The queuing system has limited buffer capacity. After B buffers are full, jobs are no longer admitted
- State transition diagram is similar to that of M/M/m but it finishes with B (as opposed to having  $\infty$  states)
- As before

$$P_n = \frac{\lambda_0 \lambda_1 \dots \lambda_{n-1}}{\mu_1 \mu_2 \dots \mu_n} P_0$$

# State probabilities

$$P_n = \frac{\lambda^n}{n! \mu^n} P_0 = \frac{(m\rho)^n}{n!} P_0$$

for  $n < m$

$$P_n = \frac{\lambda^n}{m! m^{n-m} \mu^n} P_0 = \frac{\rho^n m^m}{m!} P_0$$

for  $n = m, m+1, \dots, B$

# M/M/m/B

- All the other parameters can be computed from these probabilities (see the textbook)
- Effective arrival rate:
  - Arrival rate  $\lambda$
  - After B jobs, no more jobs enter the system
  - $\lambda' = \lambda (1-p_B)$  effective arrival rate
  - $\lambda - \lambda' = \lambda p_B$  packet loss rate
- Apply effective arrival rate to Little's Law

Advanced Systems Lab  
Tutorial V  
Analyzing a system

G. Alonso

Systems Group

<http://www.systems.ethz.ch>

# Analytical treatment of a system

- Find ways to predict the behavior of a system through a model
  - Know what will happen without having to run an experiment
  - Use it to predict behavior
  - Gain more understanding about the system
  - Extrapolate behavior and results beyond what is feasible to test empirically
- Use experimental data to refine and validate the model

# Tools

- Statistics
- Queuing theory and models
- Operational laws
- Intuition and insight on how the system works



# For milestone 3

- A template will be made available
- Many possibilities for building models
  - Depends on the architecture of the system
  - Depends on what has been measured
  - Depends on what works
  - Depends on what aspects of the system are relevant
- Choosing what to model and how is part of the exercise

# Example 1

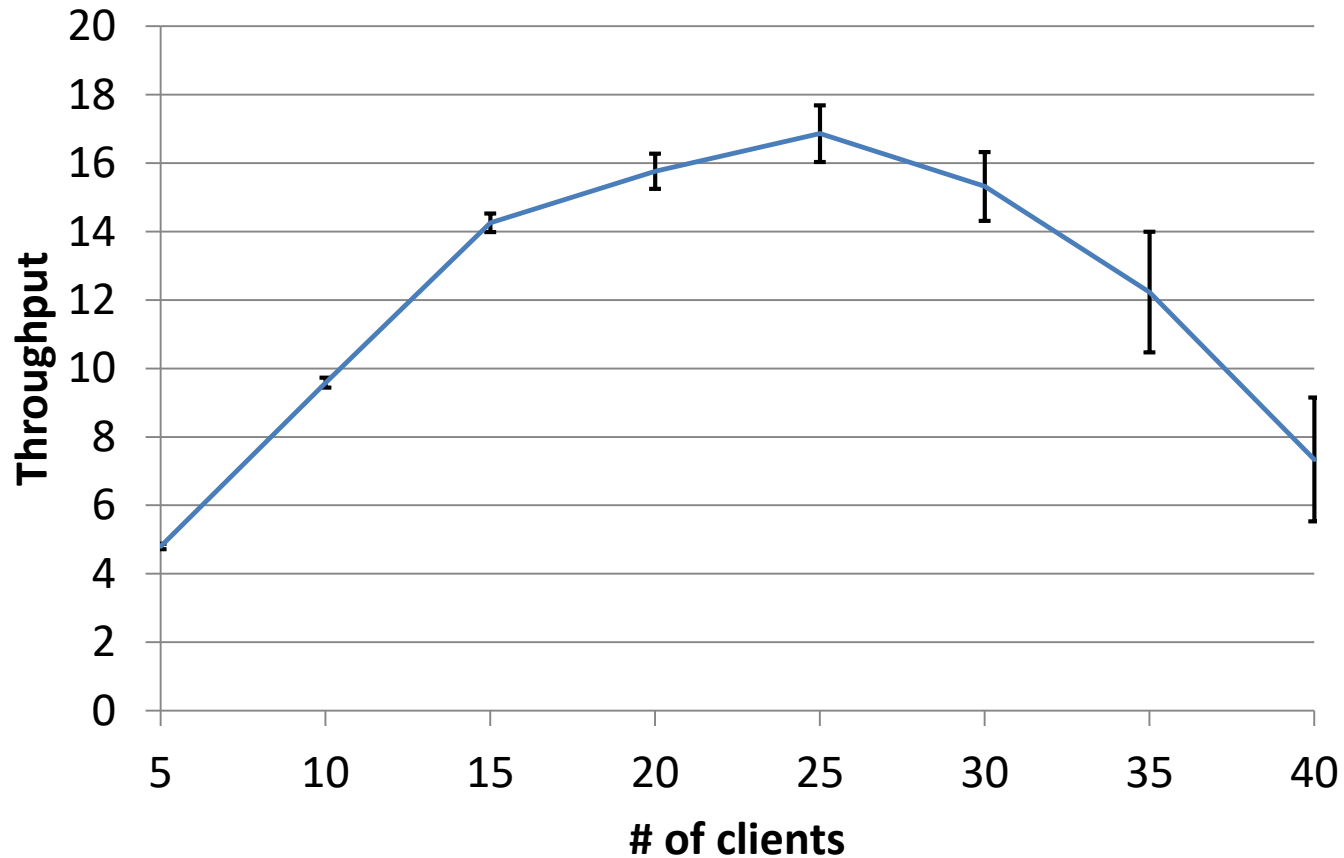
- For each load level (number of clients), five experiments are run and the table shows the throughput for each experiment in operations per second, the average, and the standard deviation. Clients have a think time of 1 second and the experiments are run as a closed system.

# Experimental data

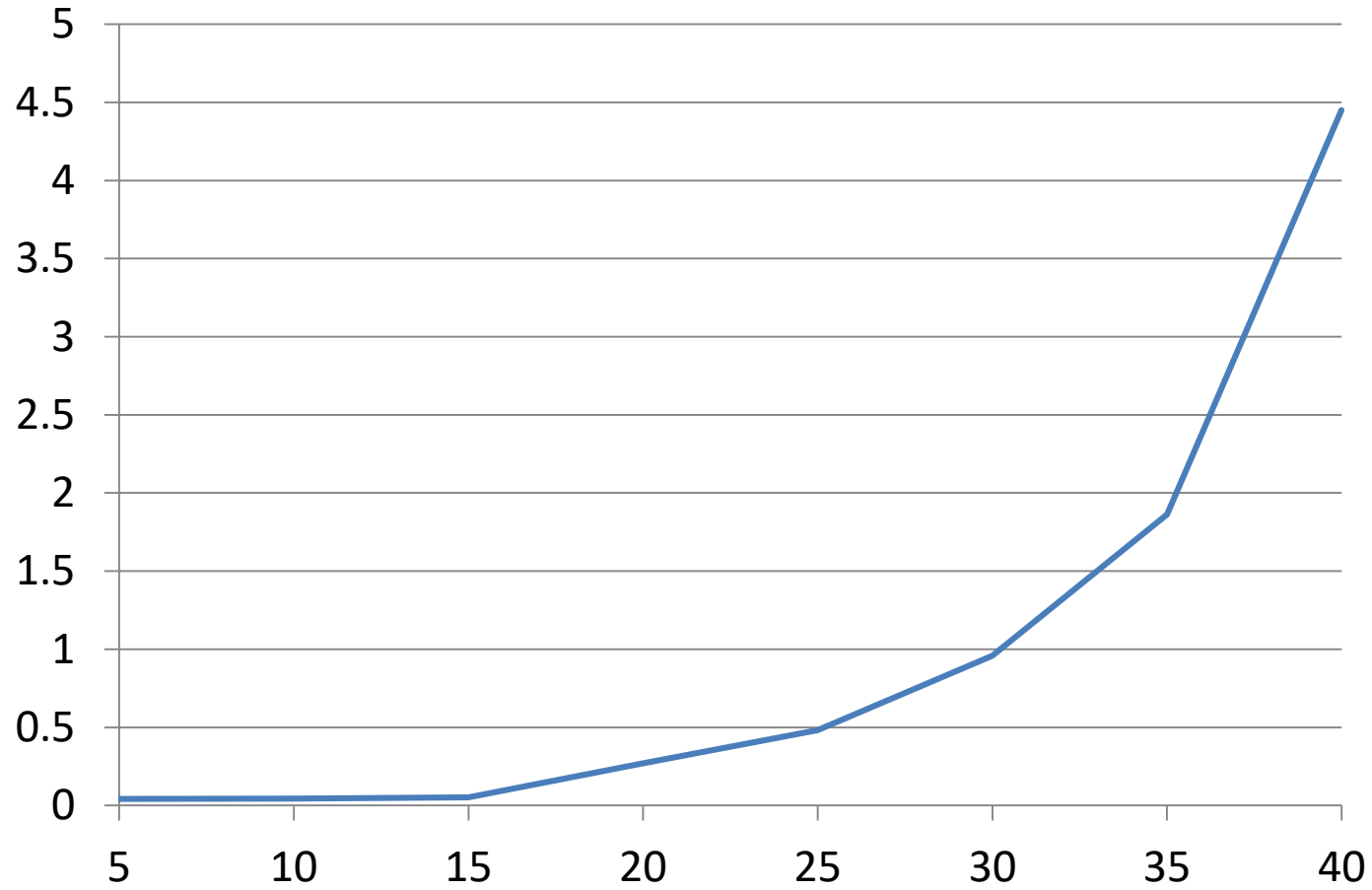
Throughput *Repetition*

<b>Load</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>AVG</b>	<b>STD</b>
<b>5</b>	4,60	4,90	5,00	4,70	4,80	4,80	0,16
<b>10</b>	9,40	9,60	9,20	9,92	9,80	9,58	0,29
<b>15</b>	14,20	13,40	14,90	14,40	14,40	14,26	0,55
<b>20</b>	16,10	15,30	17,40	14,80	15,20	15,76	1,03
<b>25</b>	17,20	14,90	17,01	15,90	19,30	16,86	1,65
<b>30</b>	16,50	17,00	12,80	16,80	13,50	15,32	2,00
<b>35</b>	8,30	14,10	10,25	11,20	17,30	12,23	3,52
<b>40</b>	12,50	4,30	5,20	4,90	9,80	7,34	3,62

# Throughput (just plot it)



# Response time



# Pay attention to what the numbers say

LOAD	X	RT
5	4,80	0,041667
10	9,58	0,043406
15	14,26	0,051893
20	15,76	0,269036

At 20 clients the response time is 5 times higher  
(system is 5 times slower)

# Service rate

- Based on those numbers:
  - What would be the service rate of the system (modeled as an M/M/1)
  - How would you calculate the number of jobs in the system?
    - 5      0,2
    - 10     0,416
    - 15     0,74
    - 20     4,24
    - 25     8,138

# And now what?

- You have characterized the system
  - Maximum performance
  - Inflexion points
  - Service rates at different loads
  - Queue length
  - ...
- Can you explain why these things are the way they are? => map to the design



# Further analysis

- Start with system as one block, then divide in smaller parts and see what you can say about each one of them
  - Where is the bottleneck (the database or the middleware?)
  - In the middleware, where is the bottleneck?
- Use 2k experiments approach to evaluate more aspects of the system once you know where the bottlenecks are