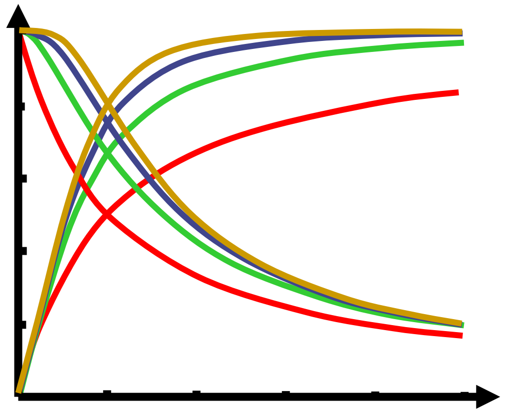
Chapter 11

Performance Modeling



11.1 Problem Statement

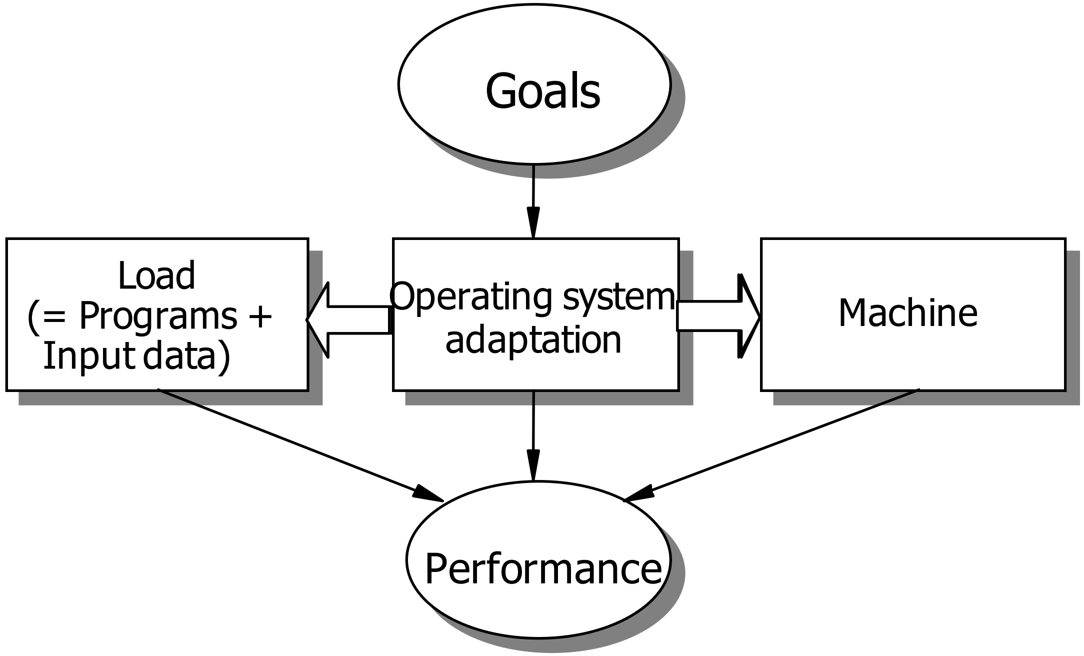
- Whether a computer system consisting of hard- and software really achieves the desired performance turns out during operation at the latest.
- Then, however, it is too late.
- Similar to other areas of engineering (e.g. architecture, aviation) quantitative issues (performance, capacity) need to accompany the design process and be interwoven with it.
- Whereas the performance of hardware components can be determined relatively easily, the performance of a complete computer system depends on the complex interplay of software and hardware components.
- It is the responsibility of the operating system to organize this interplay in the most efficient way to achieve the maximum performance.

Impact factors on the performance

- The performance data of the machine (MIPS per core, number of cores, memory capacity, bus bandwidth,...) set upper bounds for the performance of the computer system.
- To what extent this performance capacity can be exploited depends on the program load that more or less fits the properties of the machine.
- The operating system tries by using appropriate strategies and a suitable program mix to provide all active components with useful work.

Impact factors on the performance

- Finally, the user or operator/owner of the computing system defines the ultimate performance goals from which particular performance measures can be derived.



What is performance?

- Depending on the perspective and the application area, different measures for performance are useful.
- Performance measures are usually based on the intuitive physical notion of power:

  power (=performance) = work per time

- It is measured
  - how long some action takes

    (memory cycle time, block transfer time, response time, program runtime, packet delay,....)
  - how many actions per time unit are performed

    (MIPS, MFLOPS, transactions/sec, Jobs/h, Mbit/sec, SPECmarks…)
- As can be seen from the examples, pure hardware measures and measures for complete HW/SW systems are used.

How can performance be assessed?

Freie Universität Berlin

- At the real system
  - If the system to be evaluated is at disposal, we can use **measurements**.
  - A device to measure the system’s behavior is called **monitor**.
  - Monitors can be realized in hardware or in software.
- Hardware monitor
  - A monitor device consists of several probes, that are connected at those places where something is to be measured.
  - Typical components :

Counter	counting of specific events
Logic elements	combination of special signals
Comparer	Recognition of particular signal values (e.g. specific address)
Clock	To provide logged events with a timestamp
Disk / Tape	recording of signals or events

  - Measures hardware quantities, e.g. addresses, instruction execution times, bus assignment, cache misses, etc.
  - High time resolution (nsec), high sampling rate, usually no performance impact.

How can performance be assessed?

- Software Monitor
  - **A program system embedded into an application system or operating system**
  - Measures software quantities, e.g. operating system calls, procedure execution times, working set sizes
  - Uses system resources, i.e. influences performance and may distort measurements to some degree
  - Time resolution dependent on system clock
  - System specific
  - Operation modes
    - On-line-operation: Representation of measurements on-line during operation
    - Off-line-operation: Recording of events (trace) on secondary memory for later post-processing and evaluation

Monitors

- Monitors are useful for bottleneck analysis and optimization of existent hard-/software systems.
- If we can measure the call frequencies of particular operating system modules, we know at which place further code optimization is profitable.
- If we can measure statistical profile of memory requests, we can optimize the memory management scheme towards this profile.
- If we know the block access frequencies at the disk storage we tune the track allocation to minimize head movements and thus access times.
- If we know the behavior of individual programs (e.g. compute bound vs. I/O-bound), we can achieve high utilization by a suitable program mix.
- If the performance of a system is poor, it may not be the processor to be blamed:
- The bottleneck may be
  - the memory (too much paging)
  - the cache (too low hit ratio)
  - the bus (too many bus conflicts)
- All this can be found out using monitors.

Characterizing the Program Load

- Processor performance indicators such as MIPS or MFLOPS are based on a weighted mix of individual instructions.
- They only tell you something about performance if the processor is really the bottleneck.
- I/O behavior and possible performance loss due to the operating system remains unconsidered.

- Example: Gibson Instruction Mix

Instruction type	**percentage**
Load and Store	31.2
Fixed-Point Add and Subtract	6.1
Compares	3.8
Branches	16.6
Floating Point Add and Subtract	6.9
Floating Multiply	3.8
Floating Divide	1.5
Fixed-Point Multiply	0.6
Fixed-Point Divide	0.2
Shifting	4.4
Logical, And, Or	1.6
Instructions not using registers	5.3
Indexing	18.0
	100.0

Characterizing the Program Load

- **Synthetic Programs**
  - A **synthetic program** is a special small test program in a higher programming language that mainly consists of operating system calls and I/O operations.
  - It is supposed to mimic program behavior in a condensed way.
  - It is easily portable and adaptable.
  - Effects resulting from multiprogramming and high program load (paging) cannot be modeled adequately.

- **Benchmarks**
  - To determine the real performance, we have to measure real programs.
  - A **benchmark** is a program or a set of programs that represent a typical load profile (workload).

Application-specific Benchmarks

Freie Universität Berlin

- Since the performance requirements are different in different areas of computing, application specific benchmarks have proven useful.

- Example
  - Linpack
    - Dedicated to scientific computing. High fraction of floating point operations.
    - Most of the time are spent in BLAS subroutines (Basic Linear Algebra Subpackage).
  - Dhrystone
    - Specialized for system software. Many procedure calls, many string operations.
    - Good for integer performance. I/O- and floating-point performance are not covered.
  - Debit-Credit Benchmark
    - Dedicated to transaction systems (Banking applications)
  - SPEC Benchmark Suite (Systems Performance Evaluation Cooperative)
    - Sort of standard, which leading manufacturers have agreed on.
    - Measures primarily CPU performance (integer and floating-point).
    - Consists of 10 selected applications from science and engineering.

Benchmark Comparison

Freie Universität Berlin

Table A. Statement distribution in percentages. *

Statement	Dhrystone	Whetstone	Linpack/saxpy
Assignment of a variable	20.4	14.4	-
Assignment of a constant	11.7	8.2	-
Assignment of an expression (one operator)	17.5	1.4	-
Assignment of an expression (two operators)	1.0	24.3	48.5
Assignment of an expression (three operators)	1.0	1.6	-
Assignment of an expression (>three operators)	-	6.8	-
One-sided if statement, "then" part executed	2.9	0.5	-
One-sided if statement, "then" part not executed	3.9	0.1	2.2
Two-sided if statement, "then" part executed	4.9	4.0	-
Two-sided if statement, "else" part executed	1.9	4.0	-
For statement (evaluation)	6.8	17.3	49.3
Goto statement	-	0.5	-
While/repeat statement (evaluation)	4.9	-	-
Switch statement	1.0	-	-
Break statement	1.0	-	-
Return statement (with expression)	4.9	-	-
Call statement (user procedure)	9.7	11.9	-
Call statement (user function)	4.9	-	-
Call statement (system procedure)	1.0	-	-
Call statement (system function)	1.0	4.7	-
	100	100	100

*Because of rounding, all percentages can add up to a number slightly below or above 100.

From: Weicker, R.P.: An Overview of Common Benchmarks. IEEE Computer, Dec. 1990

SPEC-Benchmark

Freie Universität Berlin

- Original SPEC-Benchmark:

Table 4. SPEC benchmark programs.

Acronym	Short Characterization	Language	Main Data Types
gcc	GNU C compiler	C	Integer
espresso	PLA simulator	C	Integer
spice 2g6	Analog circuit simulation	Fortran	Floating point
doduc	Monte Carlo simulation	Fortran	Floating point
nasa7	Collection of several numerical "kernels"	Fortran	Floating point
li	Lisp interpreter	C	Integer
eqntott	Switching-function minimization, mostly sorting	C	Integer
matrix300	Various matrix multiplication algorithms	Fortran	Floating point
fpppp	Maxwell equations	Fortran	Floating point
tomcatv	Mesh generation, highly vectorizable	Fortran	Floating point

- The current SPEC CPU2017 suite includes applications from these areas:

AI game theory	bioinformatics	chemistry	compilers
interpreters	data compression	fluid dynamics	physics
speech recognition	video processing	weather prediction	

SPEC-Benchmark

- Meanwhile, SPEC is only the umbrella organization, under which different groups are developing specific benchmarks:
  - Open Systems Group (OSG)
  - High Performance Group (HPC)
  - Graphics Performance Group (GPG)

- Currently, SPEC benchmarks are available for:

• CPU,	Graphics,	MPI/OMP,
• Java Client/Server,	Mail Server,	NFS,
• Power,	SIP,	SOA,
• Virtualization,	Web Servers,	Cloud/IaaS

Performance evaluation a priori

- To assess and evaluate strategy variants during the design phase, we cannot rely on measurements since the system does not exist yet.
- Frequently, we also want to abstract from machine details to exclude side effects.
- In this case we have to **model** the system and its behavior.
- To do this, we have two alternatives:
  - **Analytical models**
    - The system behavior is described by mathematical quantities and functional relations between them.
  - **Simulation models**
    - The computing system with all its components and its behavior is simulated on the computer.

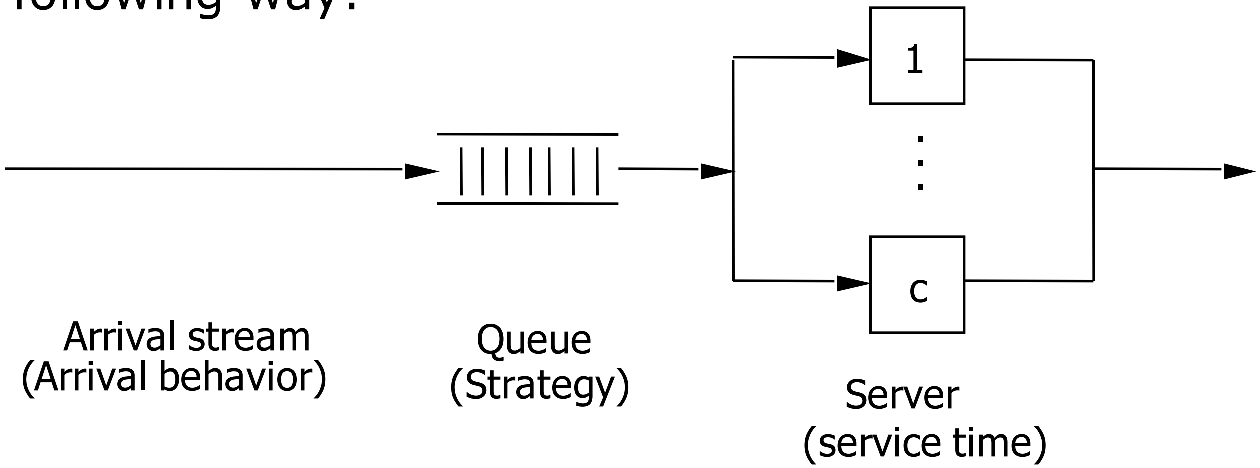
Simulation vs. Analytical Modeling

- **Simulation models** are almost unlimited concerning their accuracy and their application areas.
- Modeling can be done with an arbitrary level of detail.
- The cost is correspondingly high:
  - The development of the simulation models is costly.
  - Carrying out simulations runs is extremely compute intensive.
- To find out the functional relationship between two system quantities, we need to perform a complete set of simulation runs.

- **Analytical models** rely on assumptions that in real world are often not met (e.g. assumptions about distributions).
- The computational overhead is very low compared to simulation.
- Functional relationships can be derived directly from the model.
- The application range is limited due to the mathematical assumptions.

11.2 Queuing models: Introduction

- Queuing models consist of one or more service stations that are built in the following way:



- An arrival stream, described by the distribution of the interarrival time, feeds an input queue with objects that may be customers, requests, processes, packets etc. depending on the application.
- From there the customers get to one of the identical service stations or servers, if it is idle.
- The selection from the queue is done according to some given strategy.
- The time the customer spends at the service station is described by the probability distribution of the service time.

Characterization of queuing systems

- To describe a given queuing system, the major parameters are composed in the following characteristic way (Kendall's notation):

  A | B | c | K | P | S

- The letters have the following meaning:

  A: Distribution of the interarrival time
  B: Distribution of the service time
  c: Number of service stations
  K: Capacity of queue
  P: Size of population (maximum number of customers)
  S: Selection strategy

Characterization of queuing systems

A | B | c | K | P | S

A: Distribution of the interarrival time

Examples:

D	Deterministic
M	Markov (exponential)
E_r	Erlang stage r
H_r	Hyperexponential degree r
G	General (unspecified)

B: Distribution of the service time

Possible specifications as with A

S: Selection strategy

Examples:

FCFS	First Come First Served
LCFS	Last Come First Served
PS	Processor Sharing

Characterization of queuing systems

A | B | c | K | P | S

- Usually, the parameters queue size K and population P are not limited.
- In those cases the quantities are not indicated.
- If nothing is said about S, the assumption is FCFS (default strategy).

- Typical descriptions are:

  D | D | 1

  M | M | c

  M | M | 1 | K

  E_r | M | 1

  M | G | c

  G | G | 1

  M | M | 3 | 20 | 1000 | FCFS

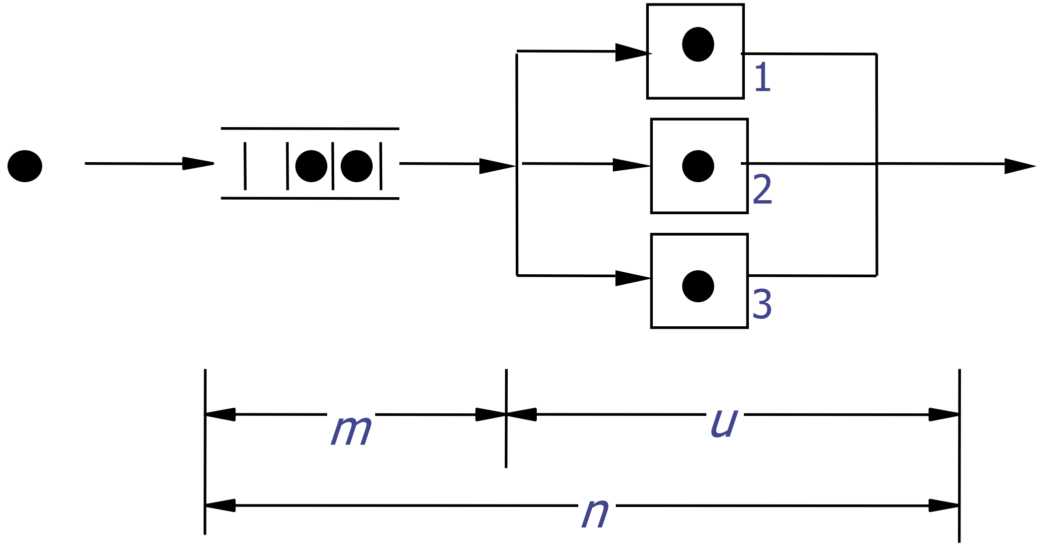
Important quantities

- **Number quantities** (stochastic variables)

  m number of customers in the queue

  u number of customers currently being served

  n number of customers in the system



Important quantities

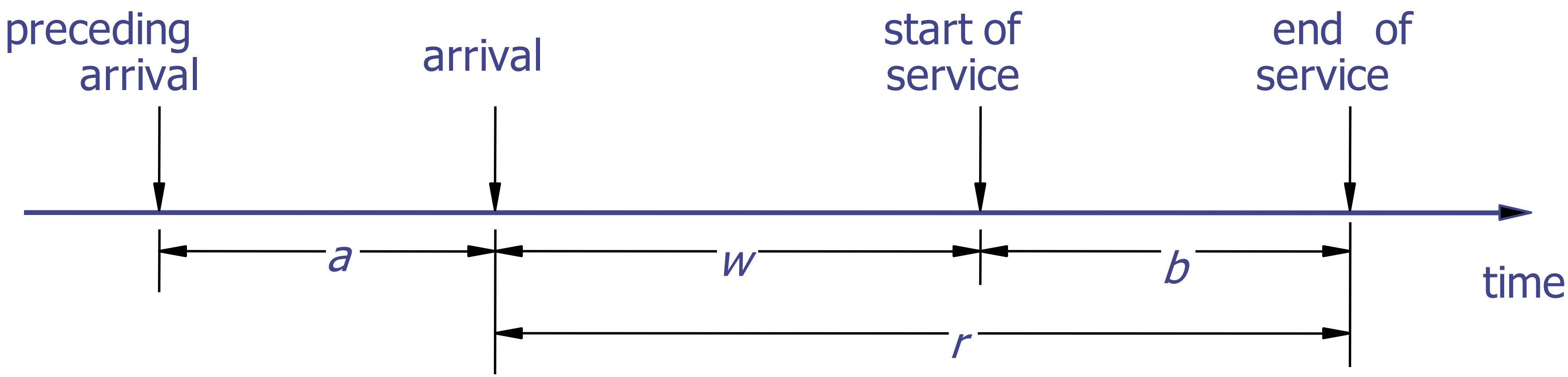
- **Time quantities** (stochastic variables)

  a interarrival time

  w waiting time (time spent in queue)

  b service time (time spent at service station)

  r response time (time spent in the system), a.k.a. residence time



- **Rates** (Parameter of distribution)

  λ arrival rate ($E[a]=1/\lambda$)

  μ service rate ($E[b]=1/\mu$)

- **Stability criterion**
  - If *c* denotes the number of service stations (or simply: servers) the following must hold:

    $$\lambda < c\ \mu$$

    "On the average, not more customers may arrive than can be processed (= served)"

  - Especially in systems with only one server ($c = 1$) we use

    $\rho := \lambda / \mu$ (traffic intensity)

  - We get as stability condition: $\rho < 1$

Elementary relations

- If the stability criterion is violated, we get (in case of unlimited population and unlimited queue size) infinite queue lengths ($m = \infty$).
- By limiting the input buffer (queue), however, the system remains (mathematically) stable.
- If there is a buffer overflow, customers (requests) get lost.

Elementary relations

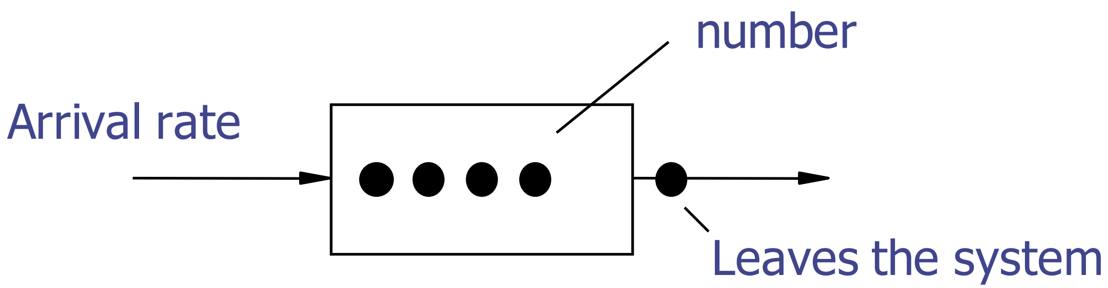
- **Numbers**
  - The following holds: $m + u = n$
  - The number of requests in the system is composed of the number in the queue and the number in service.
  - This also holds for the expectations:
    $E[m] + E[u] = E[n]$

- **Times**
  - The following holds: : $w + b = r$
  - The response time is composed of waiting time and service time.
    $E[w] + E[b] = E[r]$

- If the service rate is independent of the queue length, the additive relation is also applicable to the variances:
  $\mathrm{var}[m] + \mathrm{var}[u] = \mathrm{var}[n]$ and $\mathrm{var}[w] + \mathrm{var}[b] = \mathrm{var}[r]$

Little's Law

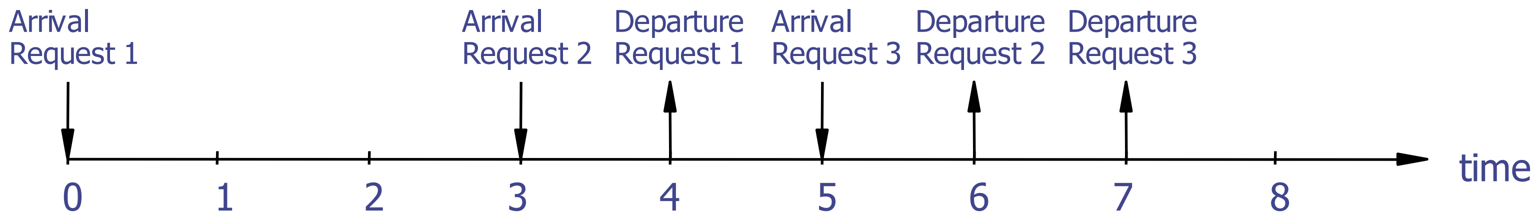
- Relation between numbers and times in arbitrary (sub)systems
- **„Mean number = arrival rate x mean residence time"**



- **Idea for proof:**
  - If you look at the system exactly when a requests leaves, then there are in the system exactly those that have arrived during the residence time of the leaving request.
  - The number of requests in the system is *N*, and the number of requests arrived during a period *R* divided by *R* is the arrival rate.
  - By taking the average we get the above relation.

Little‘s Law

- We observe the system over a long interval [0,T].



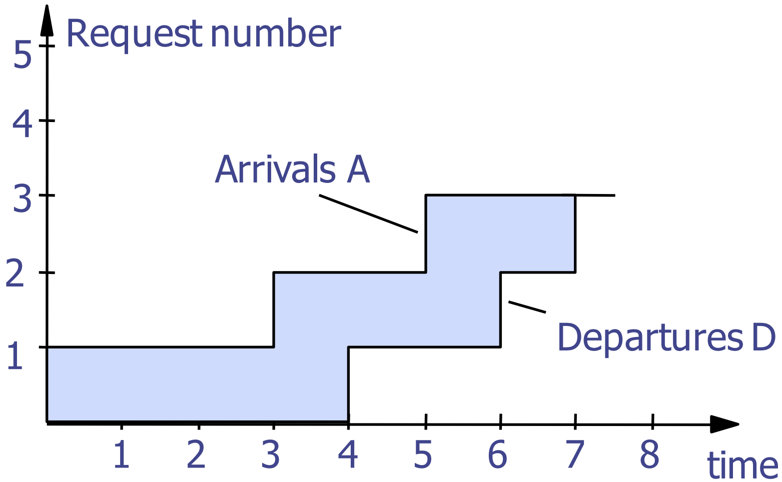
- *A(t):* Number arrivals in [0,t], t<T
- *D(t):* Number departures in [0,t], t<T

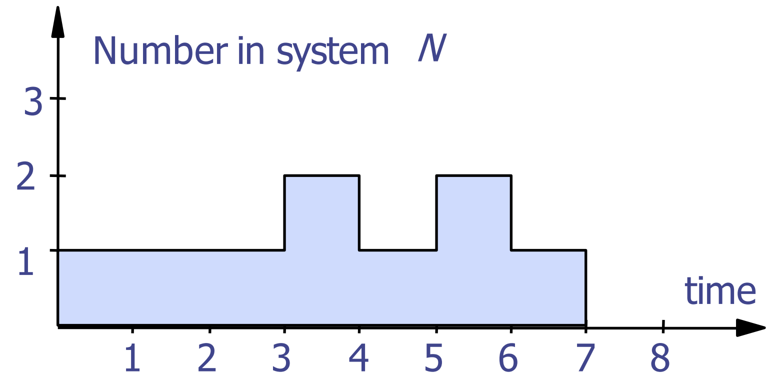
- Due to the stability condition the following approximately holds:
  A(T) = D(T).

- We obtain as arrival rate:
  Arrival rate = A(T) / T = D(T) / T = Departure rate

Little's Law

- Moreover, we get for the number of requests *N(t)* in the system:
  $N(t) = A(t) - D(t)$





- The filled area in both diagrams has the same size. It can be calculated as

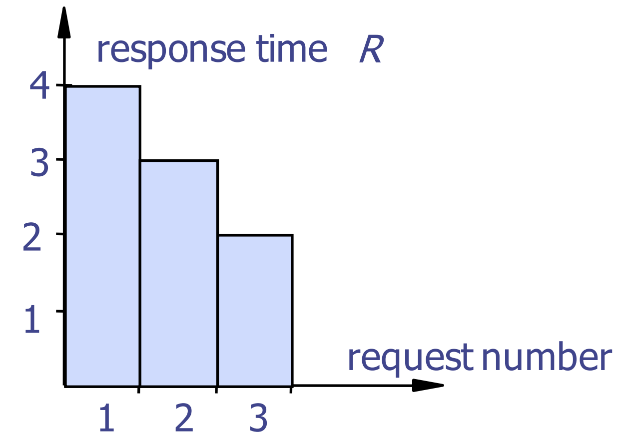
$$J = \int_0^T N(t)\,dt = \int_0^T A(t) - D(t)\,dt = \int_0^T A(t)\,dt - \int_0^T D(t)\,dt$$

and indicates the accumulated residence time of all requests in the system.

Little's Law

- J can be obtained also this way (see diagram at the right):

$$J = \sum_{i=1}^{A(T)} R_i$$



- Dividing J by the number of arrived (or departed) requests yields the mean residence (response) time:

$$\overline{R} = \frac{J}{A(T)}$$

- We obtain as the mean number of requests in the system:

$$\overline{N} = \frac{1}{T}\int_0^T N(t)\,dt = \frac{J}{T}$$

$$= \frac{A(T)}{T} \times \frac{J}{A(T)}$$

- This last equation is again Little's Law.

Little's Law

- Mean number = arrival rate × mean residence time

- The law is applicable
  - for the complete queuing station:

    $E[n] = \lambda\, E[r]$ number in system = arrival rate × response time

  - for the server alone:

    $E[u] = \lambda\, E[b]$ number in service = arrival rate × service time

  - for the queue:

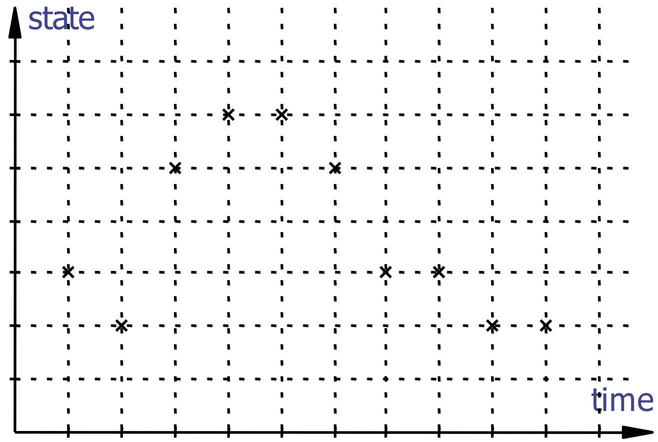
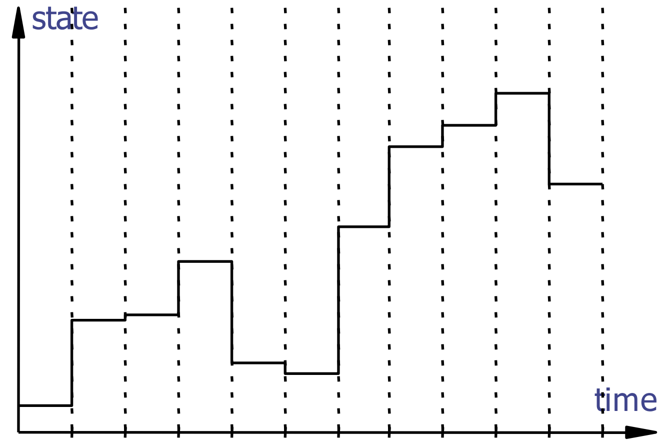
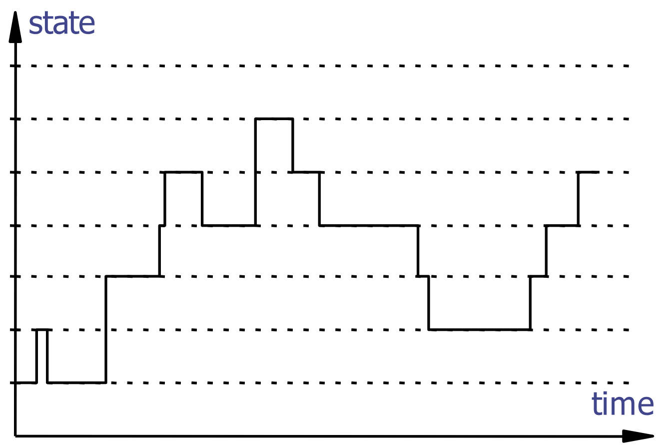
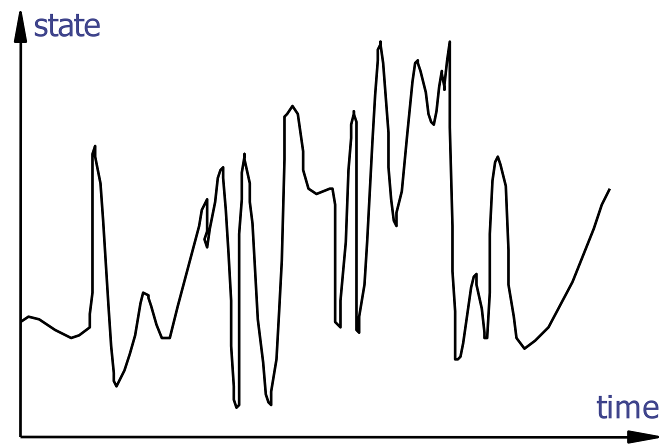
    $E[m] = \lambda\, E[w]$ number in queue = arrival rate × waiting time

11.3 Stochastic Processes

Freie Universität Berlin

- If we look at a system quantity such as the queue length *m* at different times, we will observe different values.
- They can be regarded as **stochastic variables over the time**.
- Let T be a set and let $X(t)$ be a stochastic variable for each $t \in T$.
- The collection of all stochastic variables $X(t), t \in T$ is called a *stochastic process*.
- Types of stochastic processes:
  - If the set of values that $X(t)$ can take is finite or countable, the process is called **discrete-state**, otherwise **continuous-state**.
  - If the set T is finite or countable, the process is called **discrete-time**, otherwise **continuous-time**.

Examples

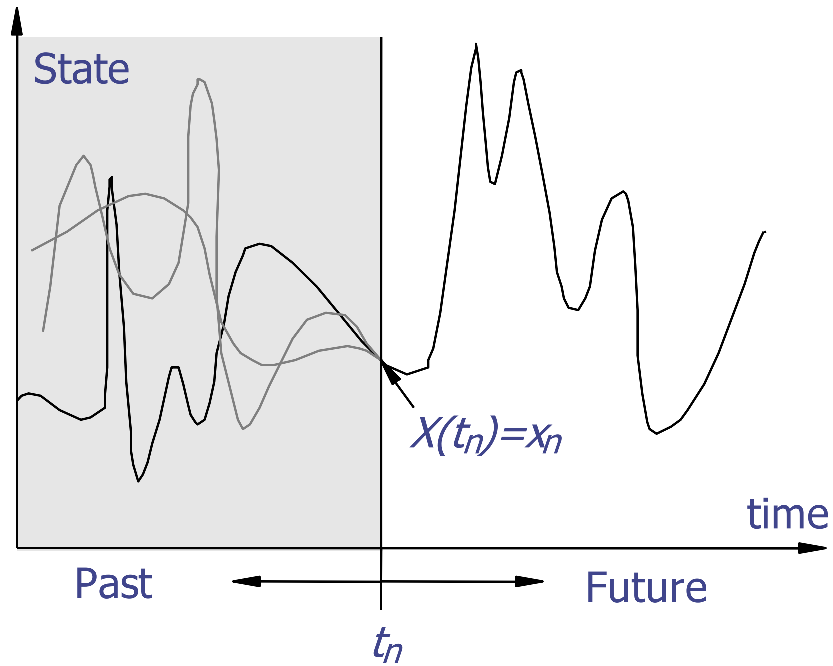
	discrete-state	continuous-state
discrete-time		
continuous-time		

Freie Universität Berlin

- **Definition**
  - A stochastic process $\{X(t),\ t \in T\}$ is called **Markov process**, if for each subset of $n+1$ values $t_1 < t_2 < \ldots < t_{n+1}$ of the index set T and for each set of $n+1$ states $\{x_1, x_2, \ldots x_{n+1}\}$ the following holds:

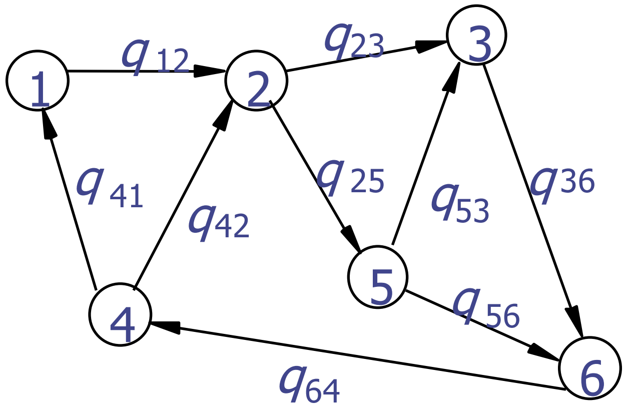
$$P\left[X(t_{n+1}) = x_{n+1} \middle| X(t_1) = x_1, X(t_2) = x_2, \ldots, X(t_n) = x_n\right]$$
$$= P\left[X(t_{n+1}) = x_{n+1} \middle| X(t_n) = x_n\right]$$

- That means that for a Markov process the next state only depends on the current state independent of from where we entered that state.
- The complete history of the process is condensed or summarized in the current state.
- A discrete-state Markov process is also called **Markov chain**.



Markov Chains

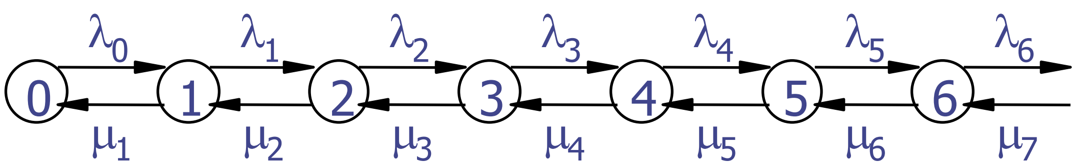
- A Markov chain switches states at some times.
- If the probability for the state change does not depend on the time, the chain is called **homogeneous**.
- A homogeneous Markov chain can be described by its state change behavior.
- We denote with q_{ij} the **transition rate** from state *i* into state *j*.
- The Markov chain can be represented as a (possibly infinite) graph with the states as vertices and the possible transitions as edges.



- Under some assumptions the process shows a so-called **stationary behavior**, i.e. for $t \to \infty$ the process exhibits an "average" behavior that is independent of its initial state.
- Then we can calculate the probability that the process is in some state *i*.

Special Markov Chains

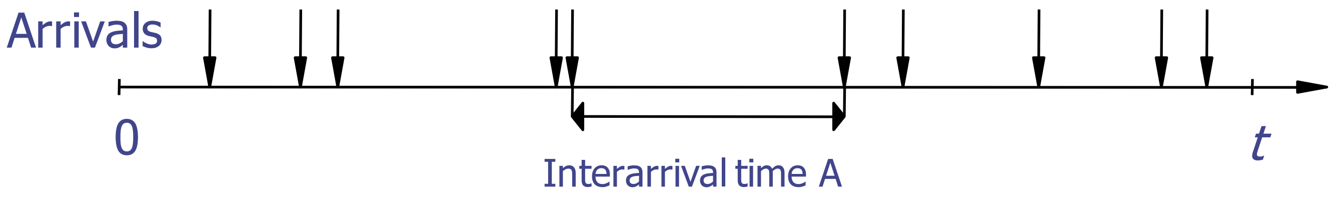
- If in a discrete, one-dimensional state space only transitions between neighboring states are possible the Markov chain is called **birth-and-death process**.
- The birth-and-death (BD) process can be described by the following state diagram:



- The λ_i are called **birth rates**, the μ_i are **death rates**.
- Applying BD processes to queuing systems the λ_i are called **arrival rates**, the μ_i are **service rates**.
- The state space may be finite or infinite.
- There are also pure birth processes ($\mu_i = 0 \; \forall i$) and pure death processes ($\lambda_i = 0 \; \forall i$). (In queuing systems: arrival process and departure process.)
- In many cases the rates are independent of the state, i.e. $\lambda_i = \lambda \; \forall i$ or $\mu_i = \mu \; \forall i$.

The Poisson Process (Poisson Stream)

- The pure birth process counts the births or arrivals, respectively, and is called **Poisson Process**.



- The following properties hold:
  - The probability for an arrival at time *t* is independent of the time of the previous arrival ("*Memorylessness*" or *Markov-property*).
  - The interarrival time is exponentially distributed, i.e.

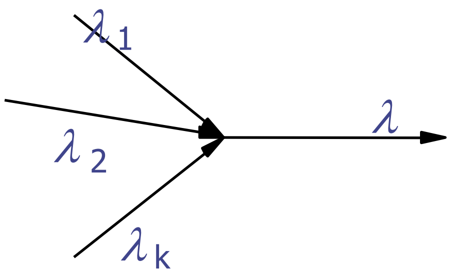
$$A(t) = 1 - e^{-\lambda t}$$

  - The number of arrivals (state of Poisson Process) *X(t)* in interval [0,t] follows a Poisson distribution, i.e.

$$P[X(t) = k] = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$$

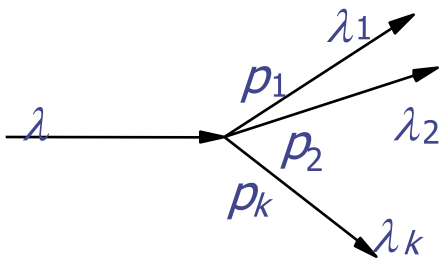
Poisson Streams

- **Joint of Poisson Streams**
  If several Poisson arrival streams are combined, the resulting combined process is again a Poisson process with rate



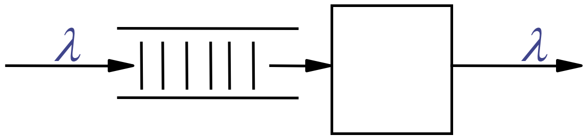
$$\lambda = \sum_{i=1}^{k} \lambda_i$$

- **Fork of Poisson Streams**
  If a Poisson stream forks into several substreams with forking probabilities $p_i, \sum p_i = 1$, the resulting substreams are also of Poisson type and the following holds:



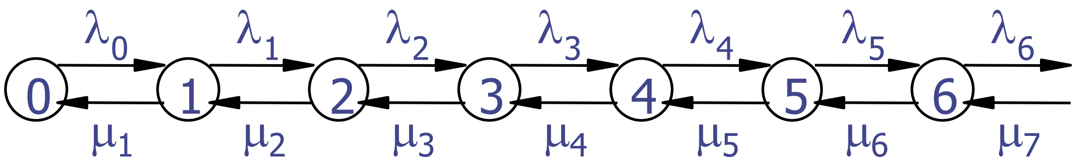
$$\lambda_i = p_i \lambda$$

- **Departures**
  If a M|M|k-Station is fed by a Poisson stream then the departure stream is also of Poisson type.



11.4 The M|M|1-Station

- The birth-and-death process exactly describes the behavior of the most simple service station, the M|M|1-Station.



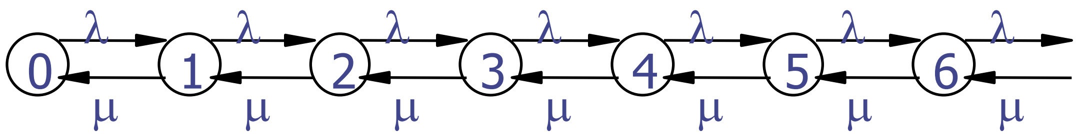
- The state of the process indicates the number *N* of requests at the station.
- The following generally holds for the BD-process:

$$P[N = k] = \pi_k = \frac{\lambda_{k-1}}{\mu_k}\frac{\lambda_{k-2}}{\mu_{k-1}}\cdots\frac{\lambda_0}{\mu_1}\pi_0 \qquad for\ k > 0$$

- The idle probability π_0 follows from the normalization condition:

$$\sum_{i=0}^{\infty} \pi_i = 1 \qquad \rightarrow \qquad \pi_0 = \left(1 + \sum_{k=1}^{\infty}\prod_{i=0}^{k-1}\frac{\lambda_i}{\mu_{i+1}}\right)^{-1}$$

Steady state probabilities



- If all rates are constant ($\lambda_i = \lambda$, $\mu_i = \mu$) we get:

$$P[N=k] = \pi_k = \frac{\lambda^k}{\mu^k}\pi_0 \quad \text{with} \qquad \pi_0 = \left(1 + \sum_{i=1}^{\infty} \frac{\lambda^k}{\mu^k}\right)^{-1} = \left(\sum_{i=0}^{\infty} \frac{\lambda^k}{\mu^k}\right)^{-1} = 1 - \frac{\lambda}{\mu}$$

- Usually, we set ρ **:=** λ **/** μ and call ρ **traffic intensity**. Then we have:

$$\pi_0 = 1 - \rho$$

- For stability reasons we require: $\rho < 1$, i.e. $\lambda < \mu$.

- Then we get for the state probabilities

$$P[N=k] = \pi_k = \rho^k(1-\rho)$$

- The number of requests in the system in a M|M|1-Station is therefore **geometrically distributed**.

Other Quantities

- Once we know the distribution, we can compute many relevant quantities:
  - Mean number of requests in the system:

$$\overline{N} = E[N] = \sum_{i=0}^{\infty} i\,\pi_i = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$$

  - Probability that there are at least *k* requests in the system:

$$P[N \geq k] = \sum_{i=k}^{\infty} \pi_k = \sum_{i=k}^{\infty} \rho^i (1-\rho) = \rho^k$$

  - Mean response time

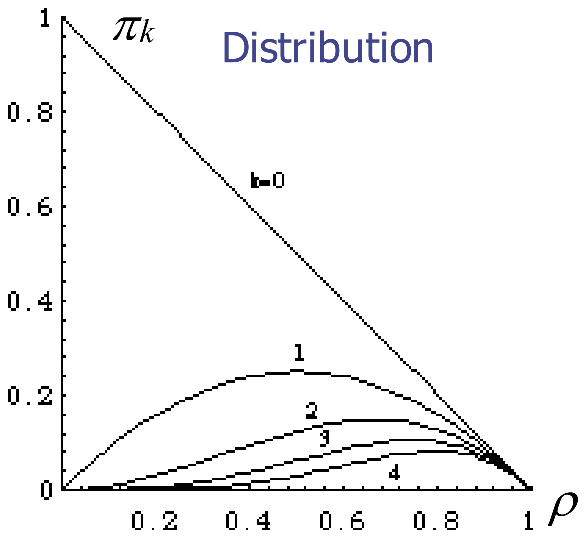
    Using Little's formula we obtain the mean response time from the mean number of requests:

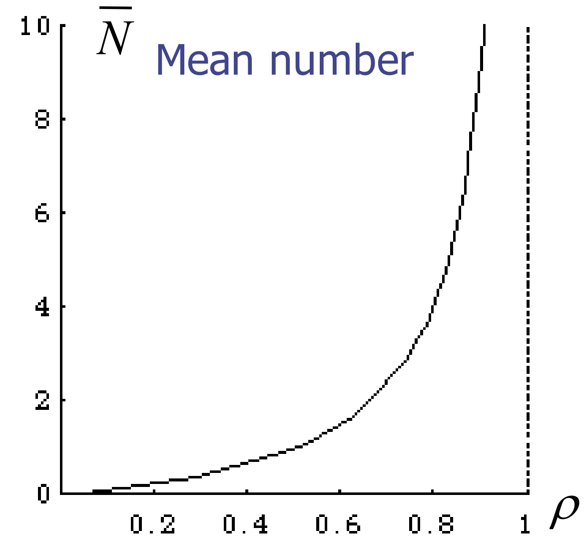
$$\overline{N} = \lambda\,\overline{R} \Rightarrow \overline{R} = \frac{\overline{N}}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu(1-\rho)} = \frac{1}{\mu-\lambda}$$

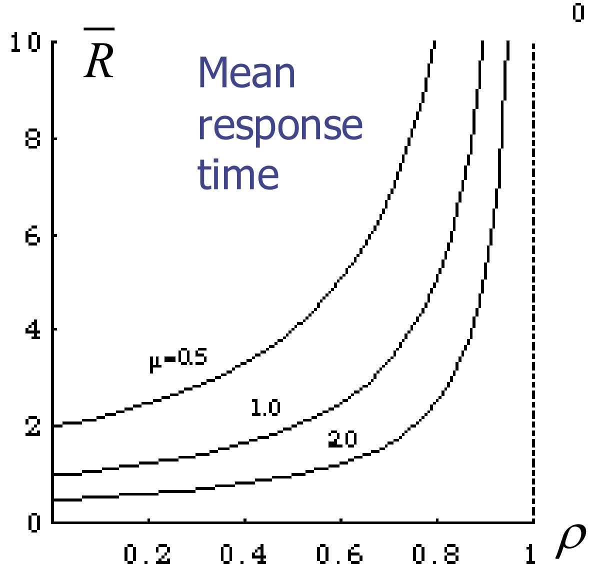
  - **Utilization** defined as the probability that the system is working:

$$\eta = P[\text{system not idle}] = 1 - \pi_0 = \rho$$

Quantities as Functions of Utilization







Example

- At a gateway arrive on the average 125 packets per second.
- The gateway needs 2 msec to forward a packet.

  a) What is the probability for a packet loss, if the gateway has buffer capacity for exactly 13 packets?

  b) How many buffer slots do we need if we want to loose at most one out of one million packets?
- Arrival rate λ = 125 pps
- Service rate μ = 1/0.002 = 500 pps
- Traffic intensity $\rho = \lambda / \mu$ = 0.25
- State probability P[k Packets at gateway] = 0.75 $(0.25)^k$
- Mean number packets at gateway: $\overline{N} = \frac{\rho}{1-\rho} = \frac{0.25}{0.75} = 0.33$
- Mean packet residence time at gateway: $\overline{R} = \frac{1}{\mu(1-\rho)} = \frac{1}{500(1-0.25)} = 2.66 \text{ msec}$
- Probability for buffer overflow (Packet loss) $P[N \geq 13] = \rho^{13} = 0.25^{13} = 1.49 \times 10^{-8}$
- Limitation of loss probability to 10^{-6}:

$$\rho^k \leq 10^{-6} \Rightarrow k \geq log\left(10^{-6}\right) / log(0.25) \approx 9.96$$

i.e. 10 buffer slots are sufficient.

Other Quantities

Freie Universität Berlin

- Mean number in service $\overline{U}$

$$\overline{U} = E[U] = 0 \cdot \pi_0 + 1 \cdot (1 - \pi_0) = \rho$$

- Mean number in queue $\overline{M}$

  Since the number of request in the station N is composed of the number in service U and the number in the queue M, we get:

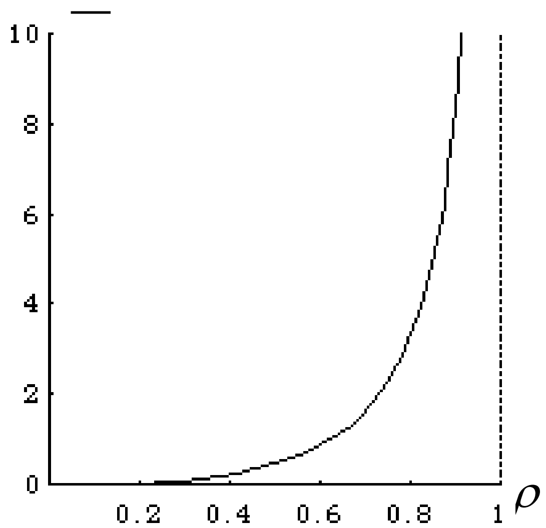
$$\overline{M} = \overline{N} - \overline{U} = \frac{\rho}{1-\rho} - \rho = \frac{\rho - (\rho - \rho^2)}{1-\rho} = \frac{\rho^2}{1-\rho} = \rho\overline{N}$$

- Mean waiting time $\overline{W}$

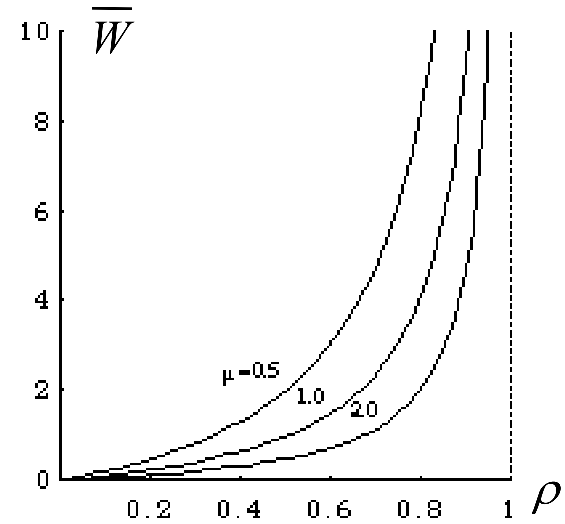
  The mean waiting time can be obtained by using Little's law :

$$\overline{W} = \overline{M} / \lambda = \frac{\rho^2}{\lambda(1-\rho)} = \frac{1}{\mu} \frac{\rho}{(1-\rho)}$$

Other Quantities



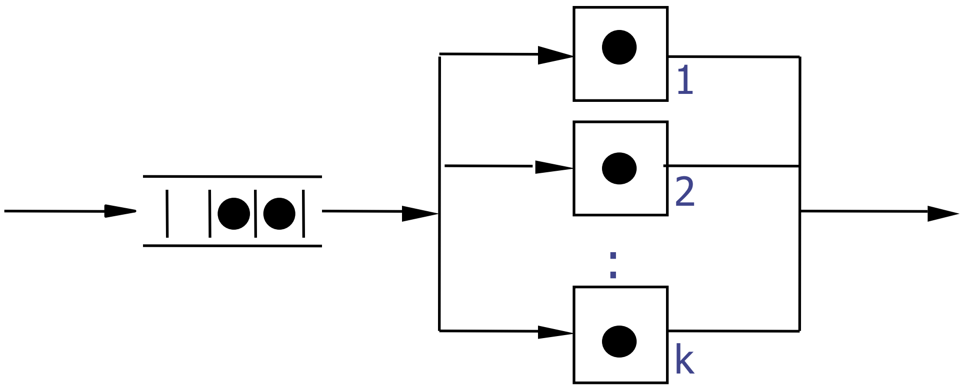
Number in queue



Waiting time

11.5 The M|M|k-Station

- In the M|M|k-Station we have *k* servers that work in parallel. Each of these *k* servers works at the same rate.



- If there are *k* or more requests at the station ($j \geq k$), then all servers are busy and provide a joint service rate of $k \times \mu$.

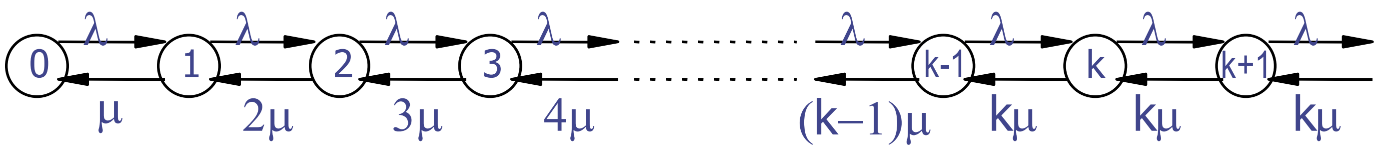
- If there are less than *k* requests at the station ($j < k$), then all these *j* requests are currently processed, i.e. *j* servers are busy and provide an accumulated service rate of $j \times \mu$.

M|M|k-Station

- Therefore, the service rate of the M|M|k-station is state-dependent:

$$\mu(j) = \begin{cases} j \cdot \mu, \text{ if } j < k \\ k \cdot \mu, \text{ if } j \geq k \end{cases}$$

- That leads to the following state transition diagram:



- The state probabilities can be derived from the general formula for the BD-process with state-dependent rates:

$$\pi_j = \begin{cases} \left(\frac{\lambda}{\mu}\right)^j \frac{1}{j!} \pi_0 & j < k \\ \left(\frac{\lambda}{\mu}\right)^j \frac{1}{k!\,k^{j-k}} \pi_0 & j \geq k \end{cases}$$

M|M|k-Station

- From these state probabilities we can (with some effort) calculate the mean values of different quantities:

$$\overline{N} = \frac{\lambda}{\mu} + \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!\left(1-\frac{\lambda}{k\mu}\right)^2}\frac{\lambda}{k\mu}\pi_0 \qquad \overline{M} = \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!\left(1-\frac{\lambda}{k\mu}\right)^2}\frac{\lambda}{k\mu}\pi_0$$

using

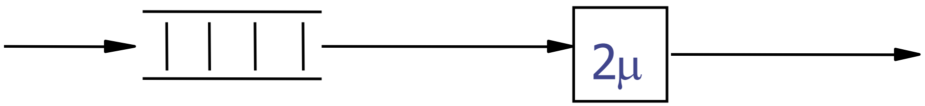
$$\pi_0 = \frac{1}{\sum_{j=0}^{k-1}\frac{1}{j!}\left(\frac{\lambda}{\mu}\right)^j + \frac{1}{k!}\left(\frac{\lambda}{\mu}\right)^k\left(\frac{1}{1-\frac{\lambda}{k\mu}}\right)}$$

- The corresponding times can be derived applying Little's law.

Example

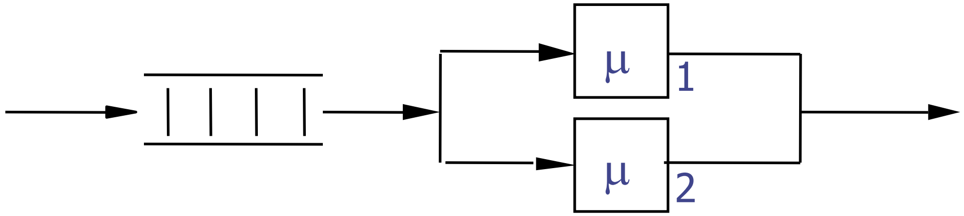
- A server works at a rate of μ.
- To improve the response times, one ponders the following alternative:
  - Alternative 1 (twice as fast)

    Replace the server by a new one with double speed, i.e. one with rate of 2μ.



  - Alternative 2 (two servers)

    Add to the existing server another one with the same speed, such that both can work in parallel and are fed by a joint input queue.

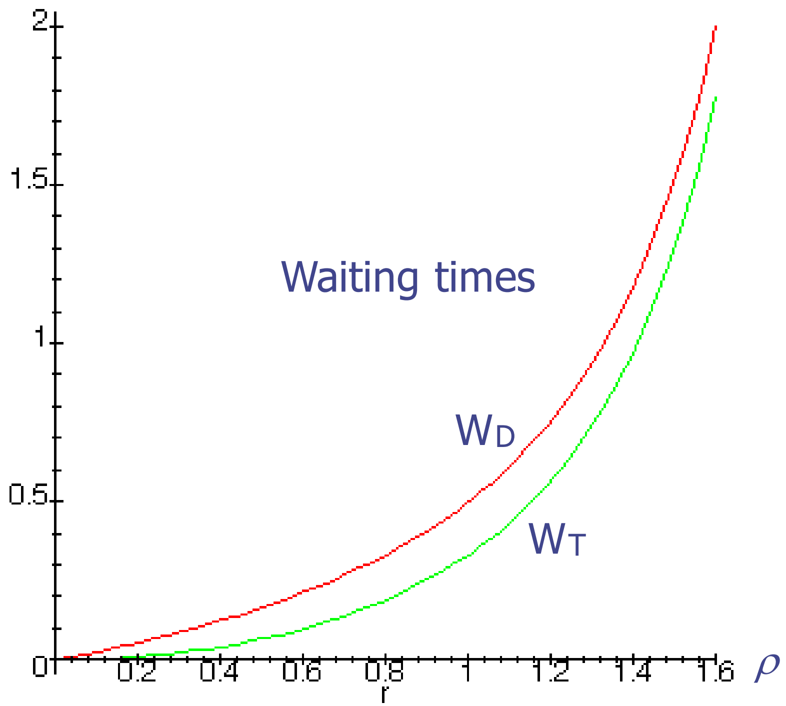


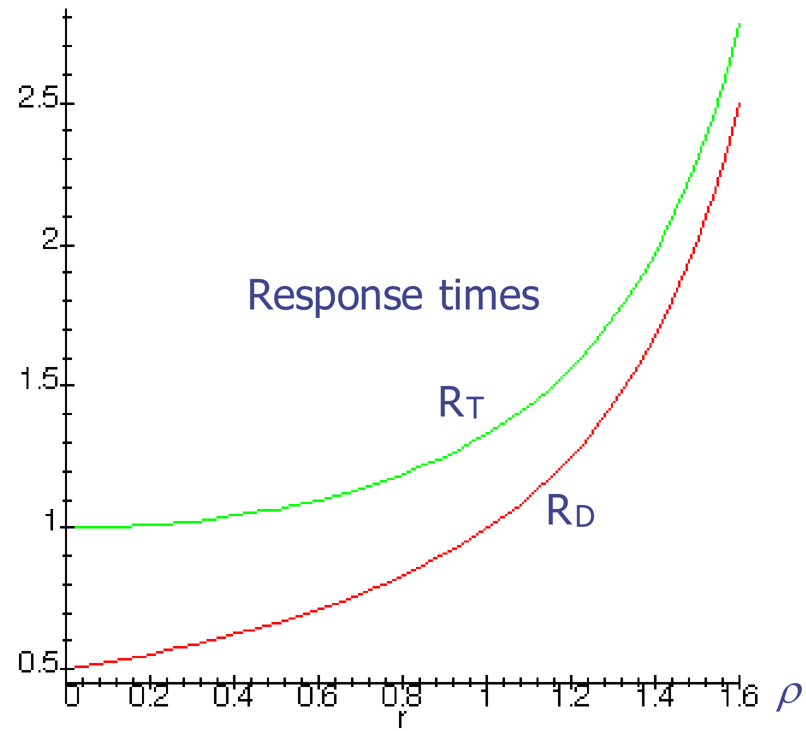
Which Alternative is better?

Freie Universität Berlin

One server with double speed	Two servers with single speed
$\overline{M}_D = \dfrac{\left(\frac{\rho}{2}\right)^2}{1-\frac{\rho}{2}}$	$\overline{M}_T = \dfrac{\left(\frac{\rho}{2}\right)^2}{1-\frac{\rho}{2}} \dfrac{\rho}{1+\frac{\rho}{2}}$
$\overline{N}_D = \dfrac{\frac{\rho}{2}}{1-\frac{\rho}{2}}$	$\overline{N}_T = \dfrac{\frac{\rho}{2}}{1-\frac{\rho}{2}} + \dfrac{\frac{\rho}{2}}{1+\frac{\rho}{2}}$
$\overline{W}_D = \dfrac{\frac{\rho}{2}}{1-\frac{\rho}{2}} \dfrac{1}{2\mu}$	$\overline{W}_T = \dfrac{\frac{\rho}{2}}{1-\frac{\rho}{2}} \dfrac{\rho}{1+\frac{\rho}{2}} \dfrac{1}{2\mu}$
$\overline{R}_D = \dfrac{1}{1-\frac{\rho}{2}} \dfrac{1}{2\mu}$	$\overline{R}_T = \left(\dfrac{1}{1-\frac{\rho}{2}} + \dfrac{1}{1+\frac{\rho}{2}} \right) \dfrac{1}{2\mu}$

Comparison of time

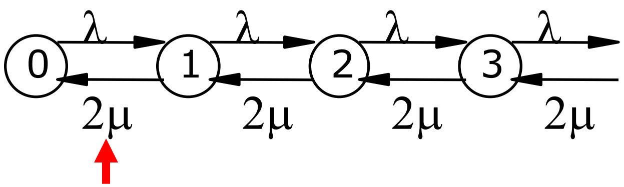




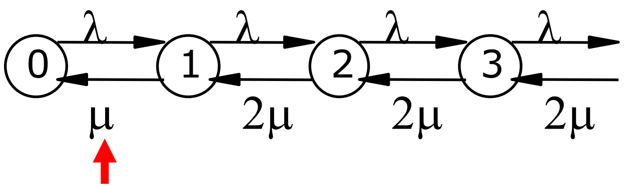
- The corresponding numbers *N* and *M* perform similarly due to Little's law.

Interpretation

- The system with two servers has lower waiting times, since for N=1 an arriving requests can be served immediately, while in the system with one double speed server it has to wait.
- The single double speed server can more than compensate the higher waiting time since a request is served in half the time.
- The single double speed server delivers its service rate of 2μ already for N=1, while the system with two single speed servers only for N=2 and more works at the rate of 2μ.



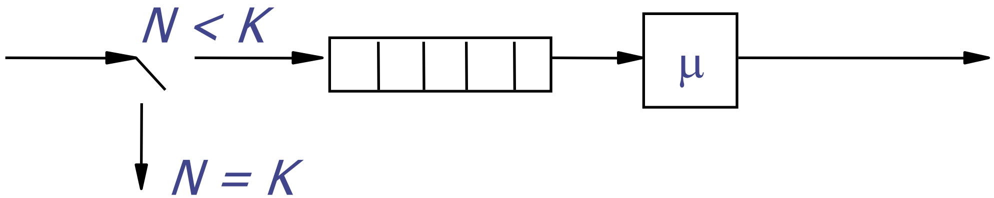
Single double speed server



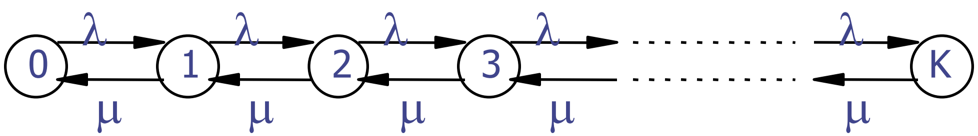
two single speed servers

11.6 The M|M|1|K-System (limited input buffer)

- If the system has a bounded queue capacity, arriving requests may be refused due to buffer overflow, i.e. they are lost.



- The state space in this case is finite:



$$\lambda_k = \begin{cases} \lambda & 0 \le k < K \\ 0 & k \ge K \end{cases}$$

$$\mu_k = \begin{cases} \mu & 0 < k \le K \\ 0 & k > K, k = 0 \end{cases}$$

The M|M|1|K-Station

- Due to the finiteness of the state space we can drop the stability condition $\rho < 1$:
- For the state probabilities we obtain

$$\pi_k = \frac{1-\rho}{1-\rho^{K+1}} \rho^k \qquad 0 \le k \le K, \qquad \rho \ne 1$$

- For $\rho = 1$ this leads to an undefined expression.
- Computing the limit $\rho \to 1$ yields (L' Hospital):

$$\pi_k = \frac{1}{K+1} \qquad 0 \le k \le K \qquad \rho = 1$$

- From that we can calculate as expectation the mean value:

$$\overline{N} = \frac{\rho}{1-\rho} - (K+1)\frac{\rho^{K+1}}{1-\rho^{K+1}}$$

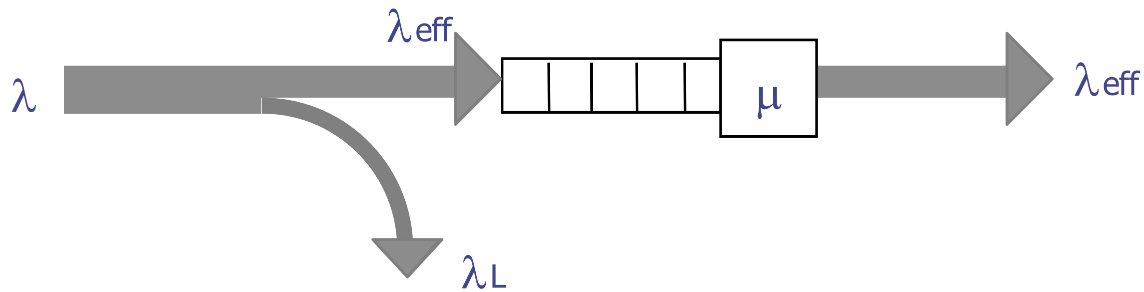
Loss due to buffer overflow

Freie Universität Berlin

- Of particular interest is the loss probability, i.e. the probability that a request is lost:

$$p_L = \pi_K = \frac{1-\rho}{1-\rho^{K+1}} \rho^K$$

- While at the system with infinite input queue the arrival rate is equal to the departure rate, here we have to consider some loss:



- The arrival stream splits into an effective stream and a loss or leakage stream: $\lambda = \lambda_{eff} + \lambda_L$
- The loss rate is $\lambda_L = p_L \lambda$
- Therefore the effective arrival rate is $\lambda_{eff} = (1-p_L) \lambda$

Remark

- Using the M|M|1|K-station we can calculate the example of slide 11-42 more accurately:
- We found: arrival rate λ = 125 pps
  service rate μ = 1/0.002 = 500 pps
  traffic intensity $\rho = \lambda / \mu$ = 0.25

- We wanted to calculate the loss probability for a buffer size of 13.
- The probability for buffer overflow (packet loss) was

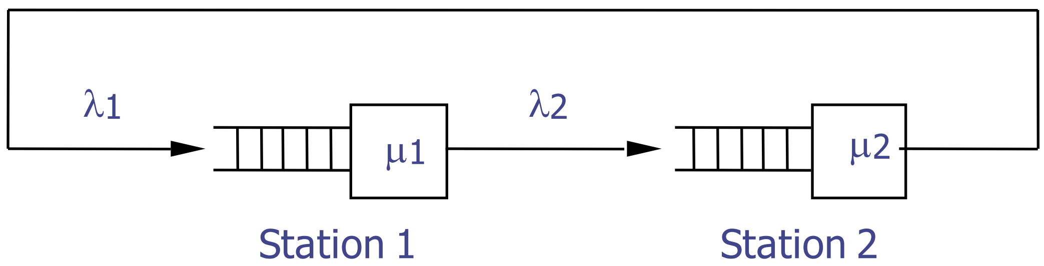
$$P[N \geq 13] = \rho^{13} = 0.25^{13} = 1.49 \times 10^{-8}$$

- This is only an approximate solution since the M|M|1| -system can also get into higher states (N>K), while the M|M|1|K-system remains in the state K.
- The M|M|1 -system should therefore calculate the loss probability to pessimistically.
- Actually, we obtain for the M|M|1|13-system a value of

$$p_L = \pi_{13} = \frac{1-0.25}{1-(0.25)^{14}} \; 0.25^{13} = 1.11 \times 10^{-8}$$

11.7 Closed System with two queuing stations

- In a multitasking system, for each process we can observe alternating compute phases and I/O phases.
- The I/O operations can be file accesses or paging activities.
- Modeling the processor and the disk each as a queuing station we get the following picture:



- We assume that in this closed system *K* processes circulate, i.e. *K* is the multiprogramming degree.
- The state of the system can be expressed solely by e.g. the number of requests (processes) *k* at the first station, because the number at the second station necessarily results to *K-k* .

Computing relevant quantities

- Unknown are the arrival rates at the particular stations.
- However, the arrival rate at the first station equals the departure rate at the second station, as long as there are requests at the second station (and vice versa):

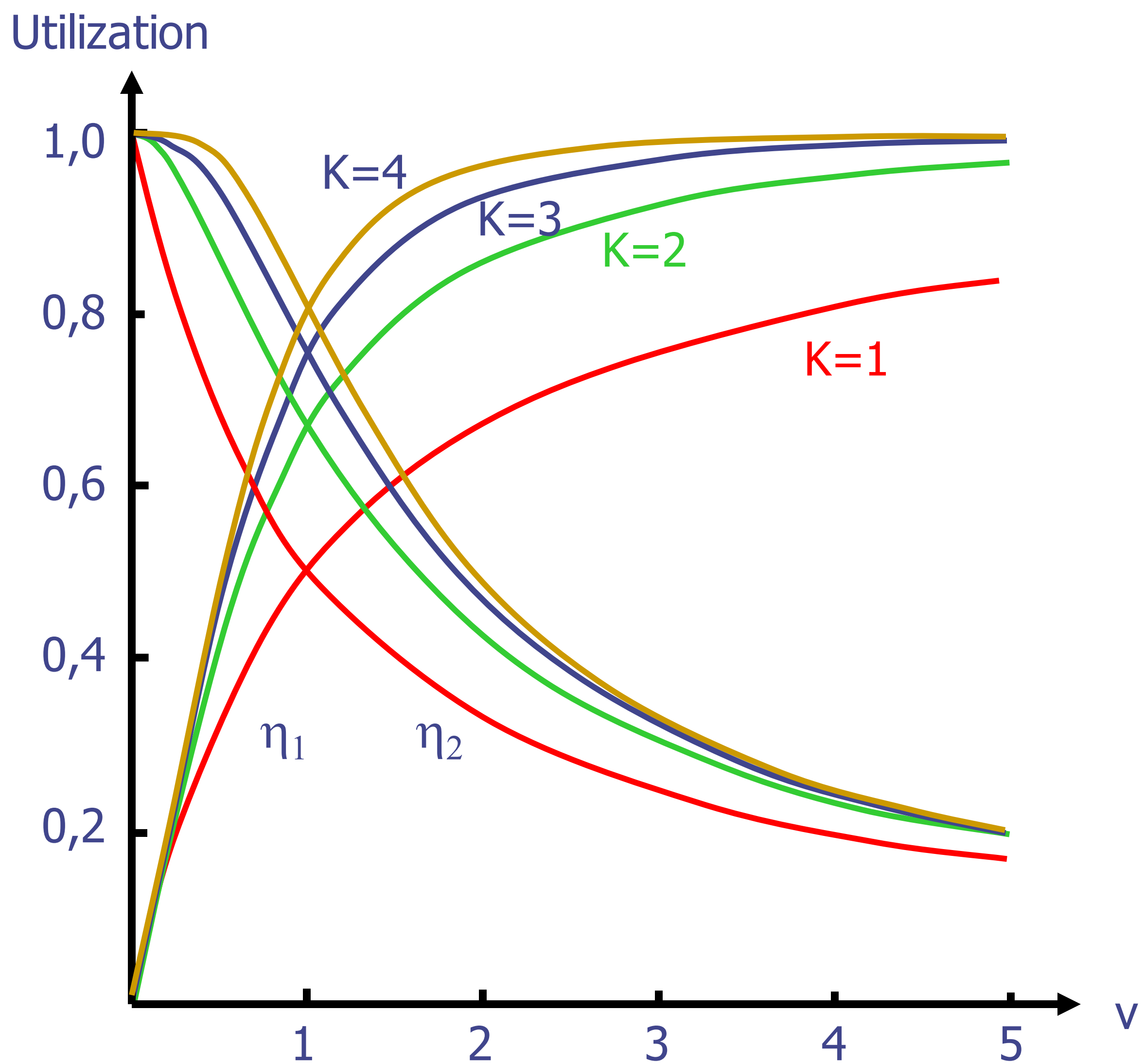
$$\lambda_1 = \begin{cases} \mu_2 & \text{for} \quad k < K \\ 0 & \phantom{\text{for}} \quad k \geq K \end{cases}$$

- These equations are exactly the same conditions as for the limited M|M|1|K-system of the last section.
- That means that the closed two-station system is exactly described by the equations of slide 11-53 if we write $v = \mu_2 / \mu_1$ instead of von $\rho = \lambda / \mu$.
- Interesting is e.g. the utilization of processor and disk:

$$\eta_1 = P[N \geq 1] = 1 - P[N = 0] = 1 - \pi_0 = 1 - \frac{1-v}{1-v^{K+1}} = v\,\frac{1-v^K}{1-v^{K+1}}$$

$$\eta_2 = P[N \leq K-1] = 1 - P[N = K] = 1 - \pi_K = 1 - \frac{v^K - v^{K+1}}{1-v^{K+1}} = \frac{1-v^K}{1-v^{K+1}}$$

Utilization in the 2-Server System



Modeling paging

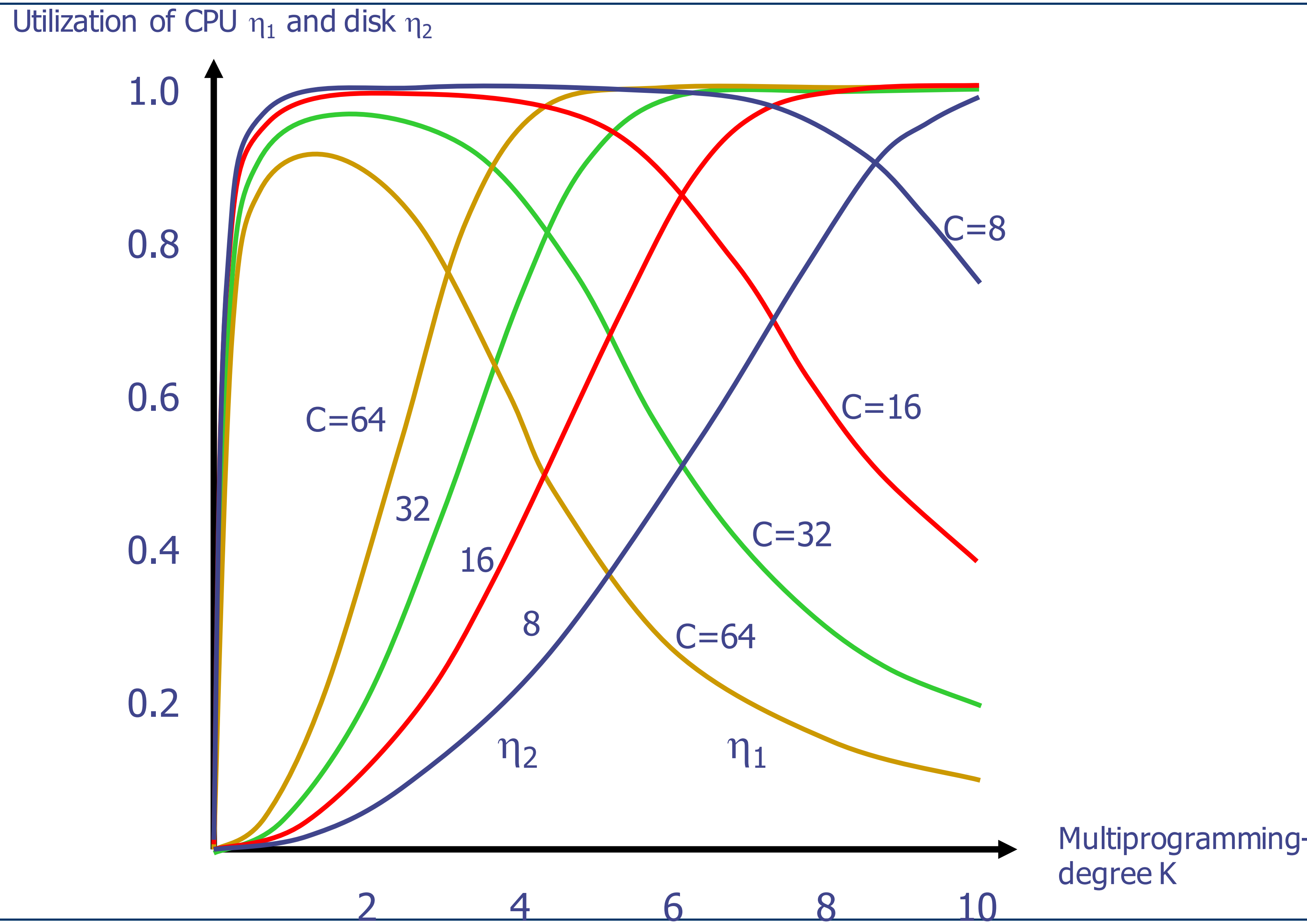
- We know from the discussion of paging that increasing the multiprogramming degree *K* reduces the number of page frames per process.
- By that, the time between two page faults gets smaller and smaller.
- As relationship between the multiprogramming degree and the interpage-fault time we know from empirical studies:

$$t_s = a\left(\frac{M}{K}\right)^2$$, where *M* is the memory size.

- We now can model this relationship by using a closed two-station system:
- Let station 1 be the CPU, station 2 the paging device.
- The interpage-fault time can be interpreted as the mean service time of the CPU:

$$t_s = \overline{B}_1 = 1/\mu_1 \quad \Rightarrow \quad \mu_1 = \frac{1}{aM^2}K^2 = C \cdot K^2$$

Thrashing


Utilization of CPU η_1 and disk η_2
1.0
0.8
0.6
0.4
0.2
C=8
C=16
C=32
C=64
C=64
32
16
8
η_2
η_1
Multiprogramming-degree K
2
4
6
8
10

Modeling paging

- We can derive the following:
  - For small *K* the CPU utilization (η_1) increases strongly and stays for some time at a high level.
  - The utilization of the paging disk (η_2) is initially very low.
  - With increasing *K* the disk utilization increases and at the same time the CPU utilization.
  - There is a point where congestion of the processes starts because of the intensive usage of the paging device (thrashing effect).
  - For large values of the constant *C* (= small memory size) the effect starts early and heavily.
  - If we have more memory available (small *C*), we can achieve high CPU utilization also for large numbers of processes K.

Remarks

- Understanding the single queuing station and applying the formulas can lead to useful insights into the behavior of an operating system.
- For more complex problems we need to
  - give up the Markov assumption
  - use other strategies as FIFO
  - regard the system to be modeled as a network of many queuing stations
- In such cases we may use modeling tools with graphical user interface to describe the system to modeled.
- Analysis and graphical presentation of the results can be done automatically.
- Some of these tools allow also the simulative analysis with statistic evaluation.

Further references

- Jain,R.: *The Art of Computer Performance Analysis*, John Wiley, 1991
- Kleinrock, L.: *Queuing Systems, Vol.1+2*, John Wiley, 1975
- Trivedi, K.: *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, John Wiley, 2002
- Weicker, R.P.: *An Overview of Common Benchmarks*. IEEE Computer, Dec. 1990
- Haverkort, B.: *Performance of Computer Communication Systems*, John Wiley, 1998
- Zeigler, B. et al.: *Theory of Modeling and Simulation* (2nd ed.) Academic Press, 2000