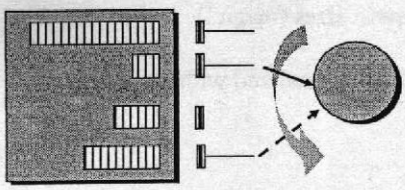
Processor Sharing

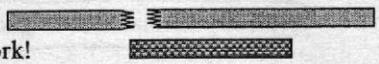


Model

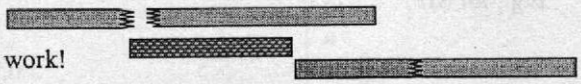
- 1 processor
- infinite buffer capacity
- processor can switch between requests, switchover times assumed neglibible

Processor Sharing (2)

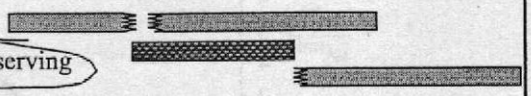
Switching between requests can take various forms:



• preemption with *abandonment* – destroys work!



• …with *rollback* – creates work!



• …with *resumption* – work-conserving

Problems:

- feasibility: service may not be "divisible"
- cost: switchover times may be significant, storage of requests' attained service

Processor Sharing (3)

Why ask for trouble?

- stronger delay differentiation – favoring small-size requests
- ... possibly without information on request sizes (magic!)

Basic evaluation criterion: mean waiting delay normalized with respect to requested service time.

For a request with service time x it is $\frac{w_m^{(x)}}{x}$.

In the case of time sharing, $w_m^{(x)} \geq \tau_m$ (waiting can't be shorter than residual service found in progress on arrival), so

$$\lim_{x \to 0} \frac{w_m^{(x)}}{x} = \infty$$

and this can't be overcome by whatever sophisticated QD we devise.

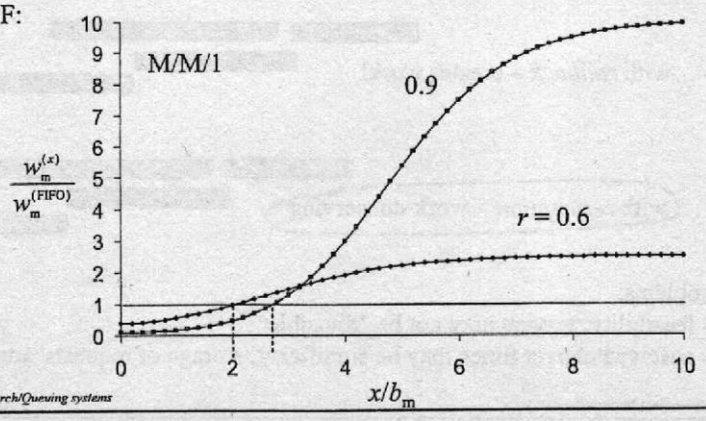
Processor Sharing (4)

Basic operational characteristic: $w_m^{(x)}$

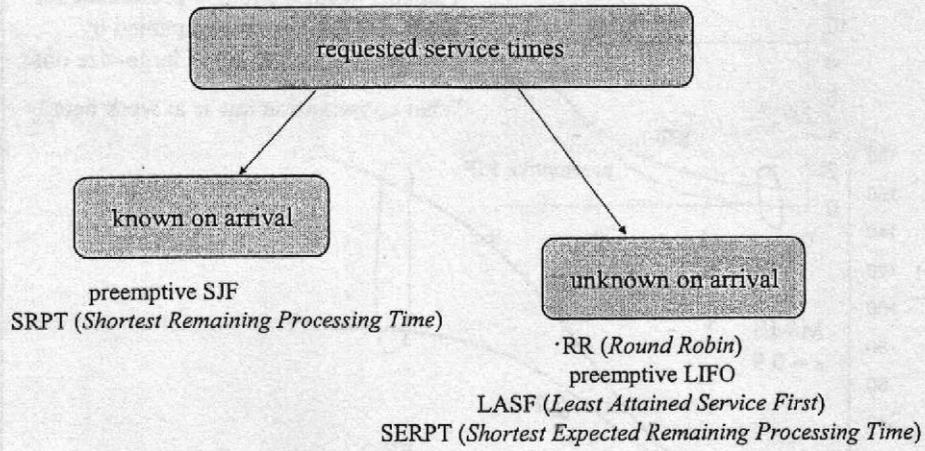
typically normalized to τ_m or $w_m^{(FIFO)}$

typically normalized to b_m

E.g., for SJF:



Processor Sharing (5)



preemptive SJF
SRPT (*Shortest Remaining Processing Time*)

·RR (*Round Robin*)
preemptive LIFO
LASF (*Least Attained Service First*)
SERPT (*Shortest Expected Remaining Processing Time*)

Preemptive HOL / SJF

Preemptive HOL:

- FIFO within class
- arrival of higher-priority request preempts lower-priority service
- upon service completion, class queues inspected from highest priority
  same as in HOL

Class i requests "don't feel" classes $j < i$:

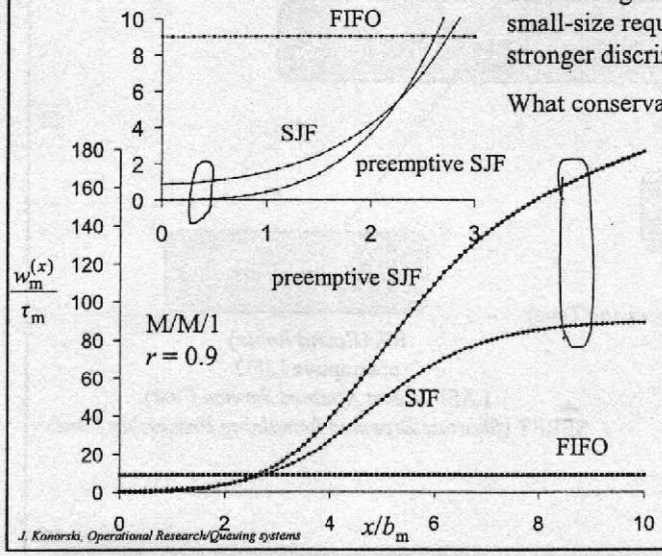
$$d_{\mathrm{m}}^{(i)} = w_{\mathrm{m}}^{(\mathrm{FIFO})}\Big|_{r^{(1)}+\ldots+r^{(i)}} + \frac{b_{\mathrm{m}}^{(i)}}{v} + \underbrace{\sum_{j<i} \frac{d_{\mathrm{m}}^{(i)}}{a_{\mathrm{m}}^{(j)}} \frac{b_{\mathrm{m}}^{(j)}}{v}}_{\text{service of preempting requests}} \qquad r^{(j)}$$

Preemptive SJF – continuous version: class $x \equiv B \in (x, x+\Delta)$

Preemptive HOL / SJF (2)

Tradeoff again – stronger preference for small-size requests accompanied by stronger discrimination of large-size ones.

What conservation law is at work here?



Processor Sharing: Conservation Law

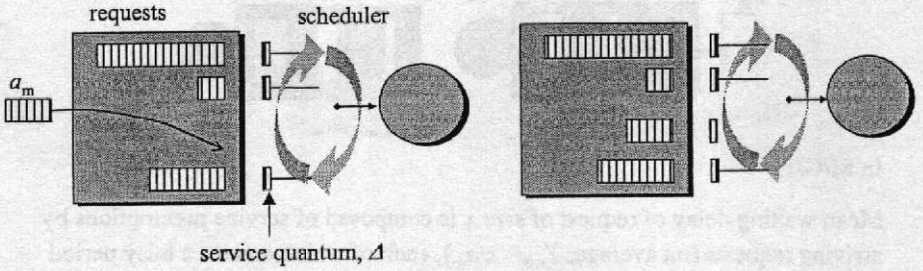
Again makes use of $U(t)$ invariance with respect to QD.
For M/G/1,

$$\int_0^\infty \frac{w_m^{(x)}}{\tau_m} P(x)dx = w_m^{(\mathrm{FIFO})}$$

$\Pr[B \geq x]$ – complementary distribution function of request size

$P(x)$ decreases in x, so reduction of $w_m^{(x)}$ for small x causes a stronger still growth for large x.

RR



As $\Delta \to 0$:

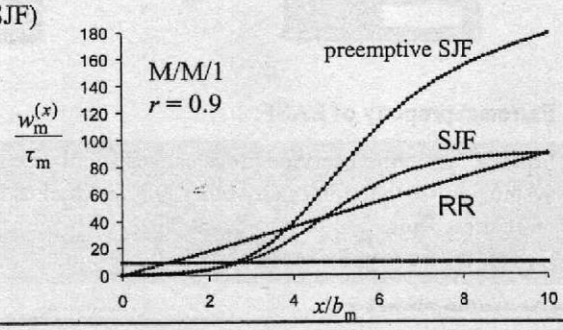
- processor shared evenly among present requests
- mean waiting delay ~ number of necessary scheduler "rounds"
  ~ number of requested service quanta ~ x
- $w_m^{(x)} = const \cdot x$

RR (2)

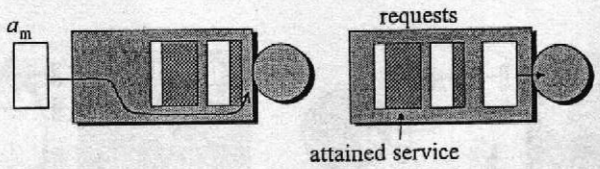
For M/G/1, the *const* follows from the conservation law, P-K formula, and distribution of mean residual time. Finally,

$$w_m^{(x)} = \frac{r}{1-r} x$$

- insensitivity to the *shape* of request size distribution!
- $w_m^{(x)} / x$ is identical for all x
- discourages artificial request aggregation / decomposition !
  (in contrast with FIFO, SJF)



Preemptive LIFO



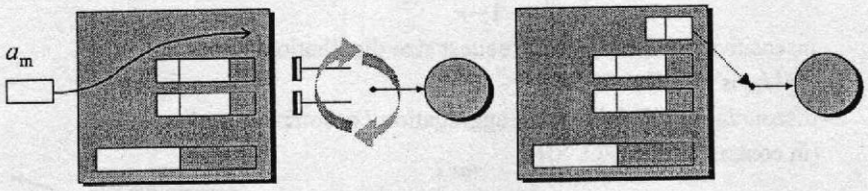
In M/G/1:

Mean waiting delay of request of size x is composed of service preemptions by arriving requests (on average, $Y_{x,m} = x/a_m$), each of which creates a busy period of mean duration $\frac{\tau_m}{1-r}$.

- $w_m^{(x)} = \frac{x}{a_m} \cdot \frac{\tau_m}{1-r} = \frac{r}{1-r} \cdot x$, same as in RR !
- mean number of request preemptions = $Y_{B/v,m} = \tau_m / a_m = r < 1$, what about RR?!
- yet delay variation is *much* larger than in RR

LASF

- at any time, processor serves request with minimum attained service time
- ties resolved by RR with $\Delta \approx 0$

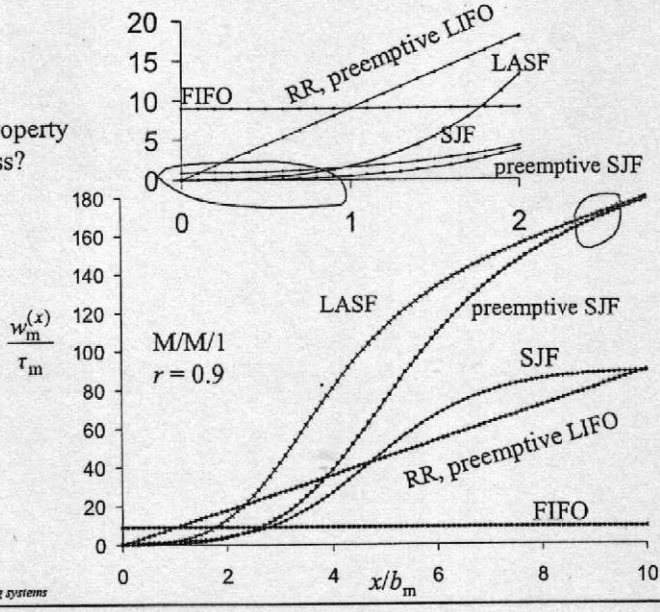


Extremal property of LASF:

Equalizing attained service times causes small-size requests to leave the system earlier than under any other QD that makes no use of information on request sizes.
Hence, favors small-size requests in the strongest way possible.

LASF (2)

What extremal property does FIFO possess?



That's that done, then!

Thank you for your attention (for good).

Jerzy Konorski

Operational Research

Queuing Systems 1: Description and Operation

Jerzy Konorski
Room 139 (old bldg)
office hours: tba
jekon@eti.pg.gda.pl

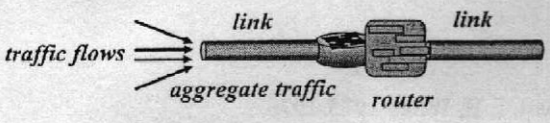
Recommended Reading

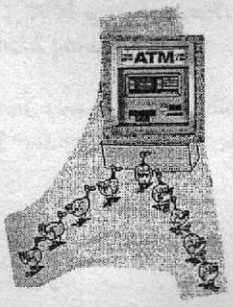
- L. Kleinrock: *Queuing systems*, vol. I, II, Wiley 1975-1976
- D. Gross, C.M. Harris: *Fundamentals of Queuing Theory*, Wiley 1998
- Joti Lal Jain, W. Boehm, Sri Gopal Mohanty: *A Course on Queuing Models*, Chapman & Hall 2006
- G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi: *Queueing Networks and Markov Chains. Modeling and Performance Evaluation with Computer Science Applications*, 2nd Ed., Wiley-Interscience 2006

Queuing (Stochastic Service) Systems: Examples

- **computer system** (mainframe / call center / database / Web server) : interruptions / system tasks / queries / transactions wait to be processed when operators / processors / data storage released
- **communication device** (network card / telephone exchange / link multiplexer): data frames / subscriber calls wait for free capacity
- **transport infrastructure** (toll gate / gas station / harbor quay / runway): vehicles await a free "service slot"
- **service access point** (ATM / supermarket checkout / public office): customers / shoppers wait to be served / attended to by clerk / till lady / server

Queuing Systems(2)



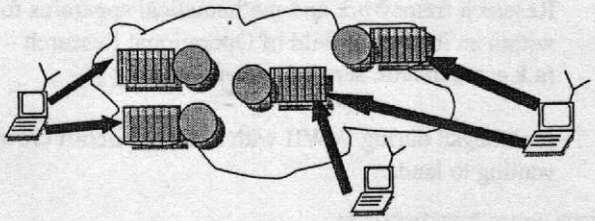


Queues everywhere!
Must be accounted for in systems design.

Queuing Systems(3)

- **system** has **resources** – limited, reusable zasoby ograniczone
- perceives events in the form of **request** arrivals = some entity demands access to the resources
- in response, system assigns resources to request enabling their consumption for a prespecified **service time**
- resources capable of serving requests = **processors**
- arriving request may find all processors busy serving other requests; then is stored in a **buffer** = waiting area for queuing requests until processor becomes free and service can commence

jest zapas do wchodz. do bufora



QueuingTheory: Mission

Population of requests / request sources usually very large.

Renders pointless optimization of specific request arrival scenarios e.g., scheduling for earliest termination or minimum processor usage.

Only meaningful is analysis and design of service systems whose input is an **arrival stream** = unpredictable on-the-fly arrivals of successive requests.

To this end we study trajectories of various queue characteristics over time = **queuing (service) processes.**

QueuingTheory: Mission (2)

With a large request population, instantaneous demand often exceeds instantaneous service supply – this is how queues form.

System designers are supposed to keep resulting damage under control e.g.,

- customer dissatisfaction due to delays / rejections, balking (refusing to join a long queue),
- buffer and queue management burden,

with a view of the economics of processor usage.

Research framework and mathematical apparatus for that were developed within an important field of Operational Research – **queuing theory** (a.k.a. **stochastic service systems theory**).

It all began during WWII with bomber aircraft crowding over the airfield, waiting to land…

Simplest Model

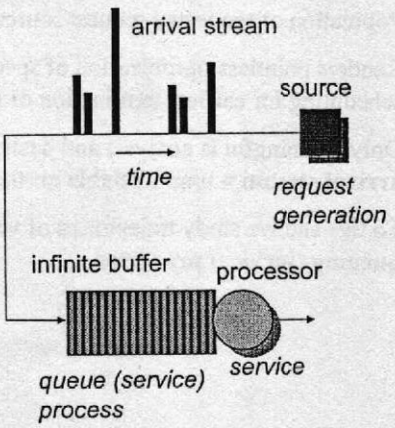
The simplest model of a queuing system consists of:

- a processor,
- a buffer, and
- an arrival stream.

Characteristics of service process depend on those of arrival stream and the way buffer & processor system operates.

How?

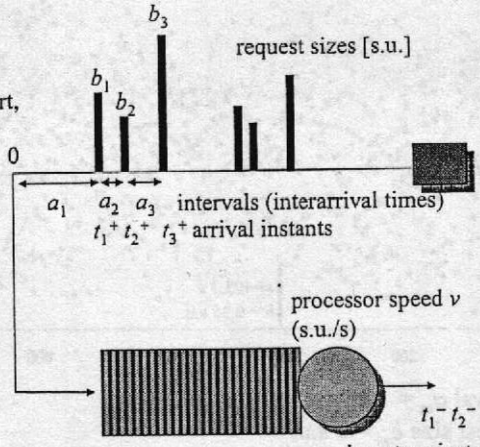
This is what queuing theory is about.



Simplest Model (2)

Which characteristics of the arrival stream and which rules of queuing system operation are relevant?

s.u.: **service unit** (byte, FLOP, item in shopping cart, time unit, ...)



b – wymagane [s.u.] zasoby
a – odstęp = czas od wcześniejszego zgłoszenia
s.u. – jednostki natury → b
$V = \frac{s.u.}{s} = \frac{b}{s}$

wyjście z procesu – opuszczenie procesu? czas

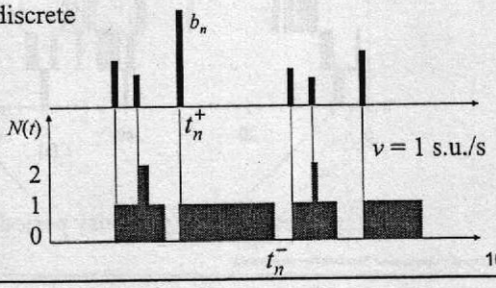
Simplest Model (3)

Remarks:

- $t_n^+ = a_1 + \ldots + a_n$ — czas pojawienia się n-tego żądania
- $b_n/v = \tau_n$ = request size / processor speed = requested service time n — czas potrzebny na obsługę żądania; b_n – wymagane zasoby; v – speed procesora
- $d_n = t_n^- - t_n^+$ = **system delay** of request n — opóźnienie systemowe, różnica pomiędzy czasem pojawienia się żądania a czasem opuszczenia systemu przez żądanie
- $w_n = d_n - b_n/v$ = **waiting (buffering) delay** of request n (wasted time) — czas buforowania, czas spędzony na czekanie na zasoby i obsłużenie
- $N(t) = \#\{n \mid t_n^+ \le t \le t_n^-\}$ = **number of requests in system** at time t — liczba zgłoszeń w systemie, tj. w buforze + te co są teraz obsługiwane

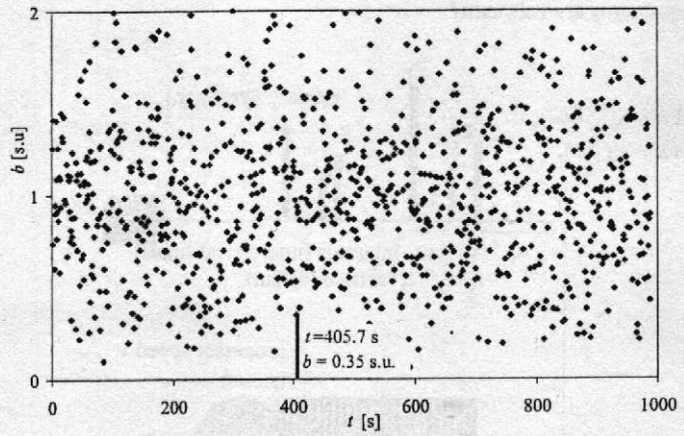
The continuous process ($N(t)$) and the discrete process (w_n) are determined by:
- processor speed v,
- **queuing discipline** (order of service)

FIFO: $t_n^- = \max\{t_n^+, t_{n-1}^-\} + b_n/v$



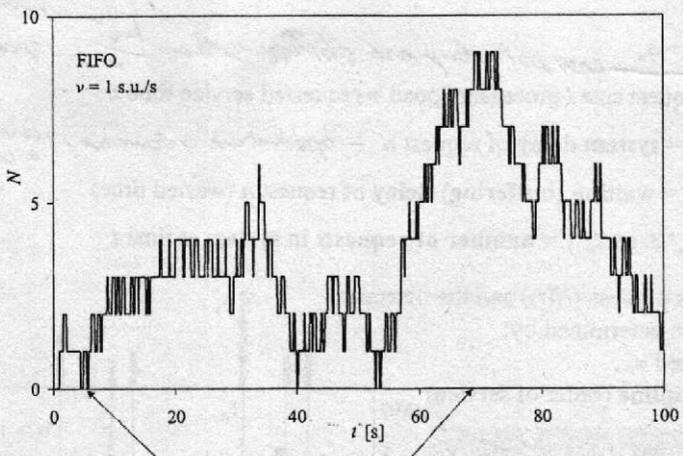
kiedy my wyjdziemy
czas wyjścia poprzednio
τ
czas obsługi w CPU
lub nasz czas przyjścia

Arrival Stream and Service Process: Example



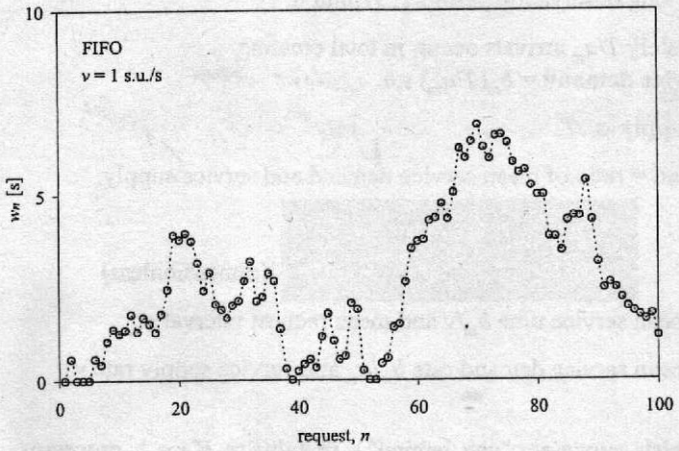
mean interval $a_m = 1$ s
mean request size $b_m = 1$ s.u.

Arrival Stream and Service Process: Example (2)



processor's idle and busy periods

Arrival Stream and Service Process: Example (3)



Properties of "Good" Service Process

- from requests' viewpoint:
  small waiting delays, rare buffer overflows
- from system operator's viewpoint:
  high processor utilization (rare idle periods)

These are contradictory! Rare idle periods imply:

- occurrences of queuing
- long queues becoming prevalent
- systematic queue growth (instability) / avalanche of buffer overflows – processor is "getting behind" in the service – unable to work in real time!

Relationship between arrival stream characteristics and processor speed determines an important parameter, **offered load**.

What information about the system operation does it give?
What offered load causes processor to "get behind"?

Offered Load

Consider a long observation period T. Within it,

- approximately T/a_m arrivals occur, in total creating mean **service demand** = $b_m(T/a_m)$ s.u.
- **service supply** is vT.

Offered load = ratio of mean service demand and service supply:

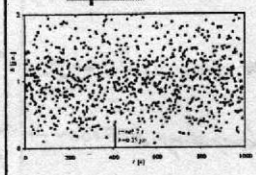
$$r = \frac{b_m(T/a_m)}{vT} = \frac{b_m/v}{a_m} = \frac{b_m/a_m}{v}$$ (dimensionless)

= ratio of mean service time b_m/v and mean request interval a_m,

= ratio of mean service demand rate b_m/a_m and service supply rate v

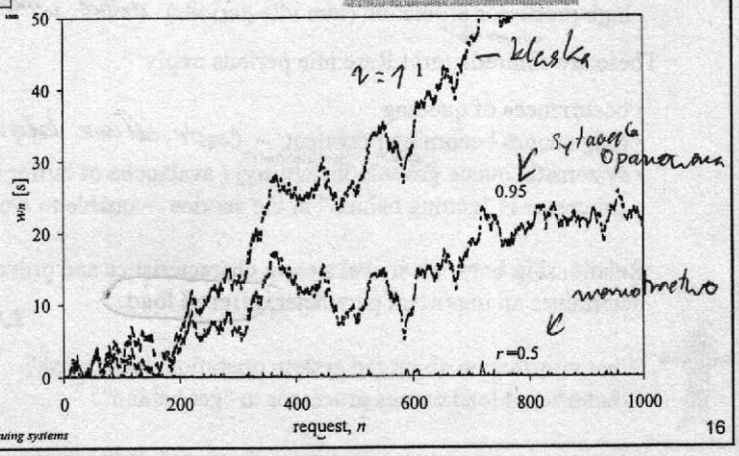
If $r > 1$ persists, processor "gets behind" – **instability**. If $r < 1$, processor "keeps pace" – system stable. What happens under $r = 1$?

Offered Load (2)



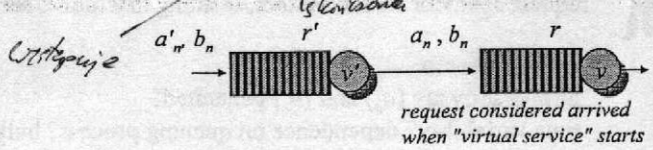
same arrival stream ($a_m = 1$ s, $b_m = 1$ s.u.)
decreasing v

instability at $r = 1$!



Impact of Input Speed

So far immediate input assumed of requests from source to system (arbitrary a_n).
In reality, request transfer from source occurs at finite speed.

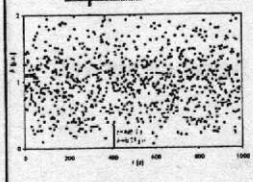


Can be modeled as a "virtual" input queuing system with processor speed $v' < \infty$, and arrival stream with b_n and arbitrarily small a'_n; offered load $r' = (b_m/a'_m)/v'$.

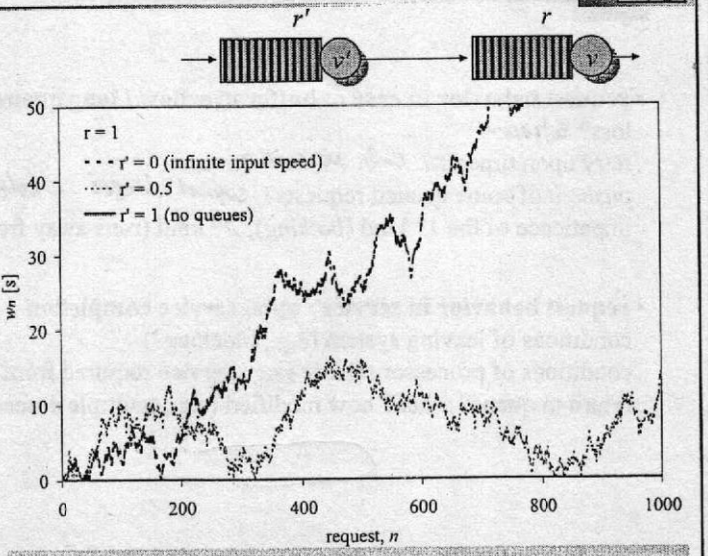
Arrival stream at the real system has $a_n \geq b_n/v'$; offered load $r = (b_m/a_m)/v$.
Clearly, $a_m = a'_m$, so $r' = (v/v')r$.

$r' = 0$
$r' = 0.5r$
$r' = r$
} corresponds to {
$v' = \infty$ (infinite input speed),
$v' = 2v$
$v' = v$ (no queues).

Impact of Input Speed (2)



same arrival stream
($a_m = 1$ s, $b_m = 1$ s.u.)



At $r = 1$ with finite input speed, system can still remain stable.

przy $r = 1$
i skończonej prędkości przychodzących
to jest szansa, że pozostanie stabilne

Towards Richer Models

How is queuing process affected by other characteristics of the arrival stream, request behavior within system, queuing discipline, service rules?

- **arrival stream**
  how exactly are (a_n) and (b_n) generated?
  time variability? dependence on queuing process? bulk arrivals?
  request sizes b_n – known/unknown on arrival?
- **buffer**
  finite capacity
  limited accessibility? when is a request rejected – *drop-tail*, …?

Towards Richer Models (2)

- **request behavior in case of buffer overflow / long queue found on arrival**
  loss?
  retry upon timeout?
  pushout of some queued requests?
  impatience of the 1st kind (*balking*), 2nd kind (runs away from queue)
- **request behavior in service / upon service completion**
  conditions of leaving system (e.g., blocking?)
  conditions of processor release (e.g., service required from other processors?)
  return to queue? when? how modified (e.g., multiple descendants)?

Towards Richer Models (3)

- **queuing discipline – what decides the order of service**
  - arrival instant (FIFO, LIFO)?
  - pure chance (RANDOM)?
  - request size / current service advancement (SJF, LASF)?
  - predefined order (RR)?
  - special requirements of requests e.g., deadlines?
  - request classification for priority, fairness (HOL, WFQ)?
- **service mode**
  - work-conserving (busy period lasts exactly $\Sigma b_i/v$)
  - or non-conserving (processor breakdowns / "vacations", background arrivals, bulk service, service preemption with abandonment / *rollback*)?
  - time or processor sharing (requests served one by one or switched between?)
  - processor-bound or parallel service?
  - multiple processors – availability? *grading* (specific processors demanded)?

Can a universal queuing systems simulator ever be written?

Steady State

In a **stationary** queuing system:

- mechanism governing request generation remains invariable in time (characteristics of the arrival stream do not change)
- processor speed v remains constant.

A *well-designed* stationary queuing system (where $r < 1$) tends to **steady state** (characteristics of the queuing process exhibit asymptotic behavior as the observation period lengthens).

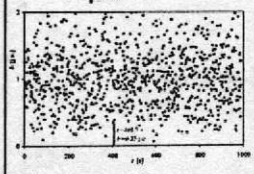
In particular,

$$\frac{|\{t \le T \mid N(t) = k\}|}{T} \to p_k \text{ as } T \to \infty$$

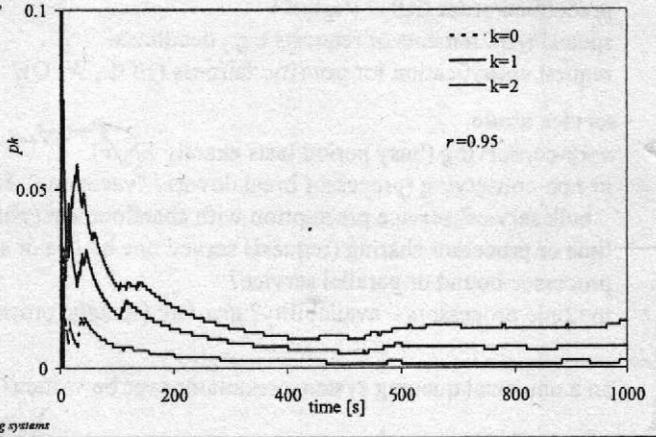
Stan ustalony

System dobrze zaprojektowany → przy stanie ustalonym, wraz ze wzrostem czasu obserwacji średnia liczba zgłoszeń w systemie nie rośnie, a stabilizuje się wokół pewnej liczby

Steady State (2)

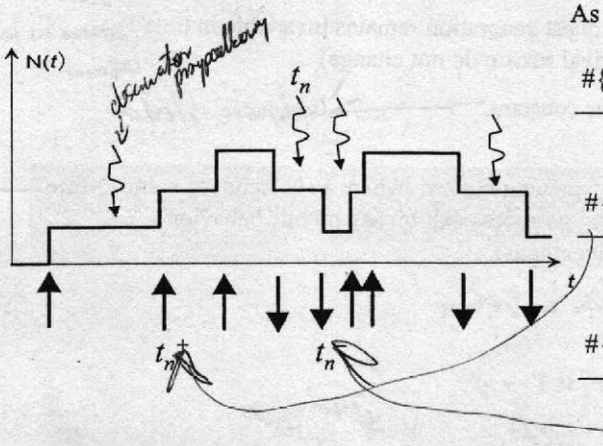


same arrival stream (a_m = 1 s, b_m = 1 s.u.)



Steady State (3)

Observation of $N(t)$ – viewpoint matters!



As $n_0 \to \infty$:

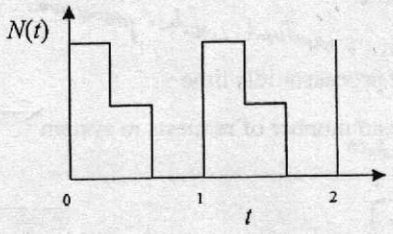
$$\frac{\#\{n \le n_0 \mid N(t_n) = k\}}{n_0} \to p_k$$ random observer

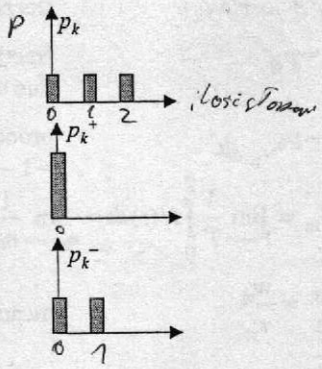
$$\frac{\#\{n \le n_0 \mid N(t_n^+ - 0) = k\}}{n_0} \to p_k^+$$ arriving request

$$\frac{\#\{n \le n_0 \mid N(t_n^- + 0) = k\}}{n_0} \to p_k^-$$ departing request

Steady State (4)

"Practitioners", beware!

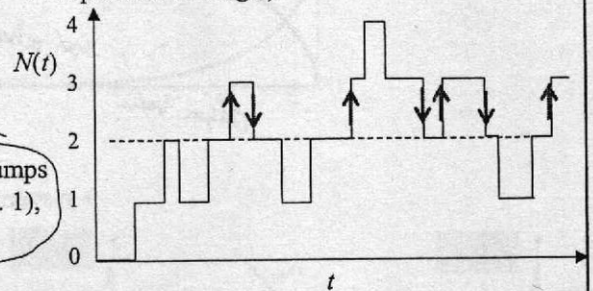




Steady State (5)

Unit jump argument: if $N(t)$ only changes by +1 then $p_k^+ \equiv p_k^-$

(% arrivals finding k in system = % departures leaving k)



over any time period,
#upward jumps – #downward jumps
= queue length (finite with prob. 1),
and $\#\uparrow - \#\downarrow \le 1$

Remark: Only *accepted* requests count as arrivals.

If $Q < \infty$ (some requests can be rejected due to buffer overflow) then

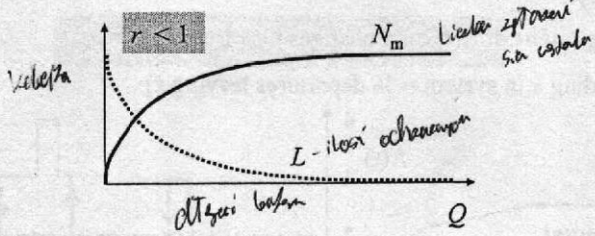
$$\frac{p_k^+}{1-p_Q^+} = p_k^-,\ k = 0,\dots,Q-1$$

Steady State (6)

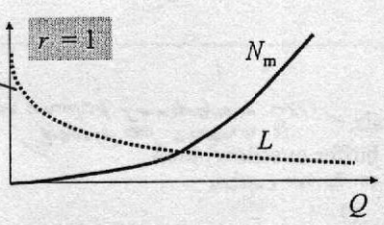
Relevant evaluation criteria:

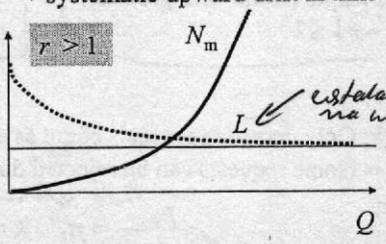
$p_1^+ + \ldots + p_{Q-1}^+$	fraction of buffered requests
$L = p_Q^+$	fraction of requests rejected (lost if no retries) due to buffer overflow
$1 - p_0$	processor utilization = 1 − proportion of processor idle time
$N_m = \lim_{T\to\infty} \frac{1}{T}\int_0^T N(t)dt = \lim_{n_0\to\infty} \frac{1}{n_0}\sum_{n=1}^{n_0} N(t_n)$	mean number of requests in system
$\frac{w_m}{b_m/\nu} = \frac{w_m}{\tau_m}$	mean waiting delay, normalized to mean service time
$\frac{d_m}{\tau_m}$	mean system delay, normalized to mean service time

Steady State (7)

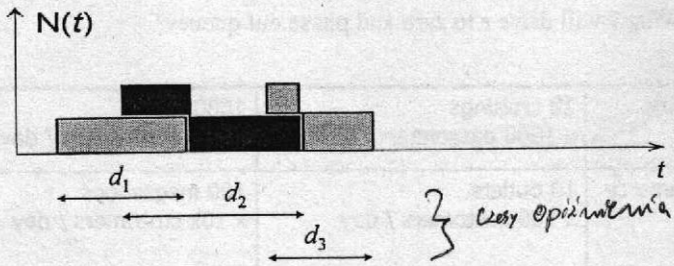


+ systematic upward drift in time





Little's Theorem



$$\text{as } n, T \to \infty, \quad \int_0^T N(t)dt = d_1 + \ldots + d_n$$

TN_m

$(T/a_m)(1-L)d_m$

Little's Theorem (2)

$$N_m = \frac{1-L}{a_m} d_m$$

mean number of requests in system = (mean system thruput) x (mean system delay)

mean population = (mean circulation) x (mean lifetime)

Valid for any part of system:

- processor: $1 - p_0 = \frac{1-L}{a_m}\tau_m = (1-L)r$ **flow conservation equation**
- buffer: $N_m - (1-p_0) = \frac{1-L}{a_m} w_m$

Is Queuing Theory Losing Momentum?

Growing v will drive r to zero and phase out queues?

transatlantic transport	10 cruisings x 1000 passengers /20 days	1000 flights x 250 passengers / day	x500
retail commerce	10 outlets x 100 customers / day	100 megashops x 10k customers / day	x1000
Internet access link	10 kb/s	100 Mb/s	x10k
processing power cost	$15m/GFLOPS (1984)	$0.5/GFLOPS (2007) http://en.wikipedia.org/wiki/FLOPS	x30m

Yet the answer is No.

Is Queuing Theory Losing Momentum? (2)

First, airliners, supermarkets, Internet links, and mainframe computers seem more crowded than ever. Same for online banking, toll-free numbers, hub airports etc.

Service demand rate b_m/a_m grows in step with service supply rate v, and so r isn't dropping any!

Is Queuing Theory Losing Momentum? (3)

Second, how do queues form anyway?

Not only because of $\nu < \infty$, but above all because of variability of a_n (arythmic arrivals) and b_n (capricious demand) exhibited by request source!

Arythmic arrivals cause *instantaneous* offered load to vary between 0 and ∞.
To get rid of queues, even occasional, one needs $\nu = \infty$.

Under $b_m/a_m < \infty$ this gives $r = 0$ i.e, zero processor utilization !!
Highly uneconomical, no matter what progress technology and management make.

What *is* economical? Keep $r < 1$ i.e., $\nu > b_m/a_m$, *but not much.*
Meaning, allow queues at times.

Comments on Queuing Theory

Queuing theory is a mathematical analysis tool. When designing a queueuing system, perhaps one could do better with a prototype / simulator?

- credible estimates of troublesome characteristics
  rare events – queue length crosses threshold,
  long busy period – do we have instability here?
- qualitative (rather than scenario-specific) influence of parameter settings upon relevant characteristics of queuing process
  – saves a lot of unnecessary experimenting
- ! - universal (qualitatively, often also quantitatively) impact of results for simple models – carry over to much more realistic ones

Contrary to what might seem, mathematical analysis is very costly.
Only pays off if provides answers that would be hard to get otherwise.

Comments on Queuing Theory (2)

Agner K. Erlang (1878-1929)

Danish mathematician and engineer,
was the first to appreciate that modern telephony cannot do without probability

today's teletraffic unit = 1 *erlang*

Comments on Queuing Theory (3)

How is queuing theory related to the theories of:

job scheduling
finding an optimum schedule for a fixed job set
vs. unpredictable on-the-fly arrivals of requests

concurrent processes
deterministic analysis of specific event scenarios
vs. massive population of random events, where only statistical characteristics are worth studying

stochastic processes
similar calculus
queuing process = nonlinear, infinite-memory transformation of arrival stream

Operational Research

Queuing Systems 2: Stochastic Models and Characteristics

Jerzy Konorski
Room 139 (old bldg)
jekon@eti.pg.gda.pl

Random Variables and Stochastic Arrival Streams

For an arrival stream observed over a finite time, a_m, b_m and any other useful characteristics can be calculated.

Yet for prediction of characteristics over infinite observation periods, or computer imitation of the arrival stream, one needs a model of generation of (a_n) and (b_n).

With rather impractical exceptions, deterministic models are of no interest:

- impractical – arrival instant and size of the next request rarely known in advance,
- carry no information (what *is* known in advance doesn't ever come as a surprise),
- pose no design challenge.

From now on we focus on **stochastic** (models of) **arrival streams** i.e., consider relevant quantities to be **random variables**:

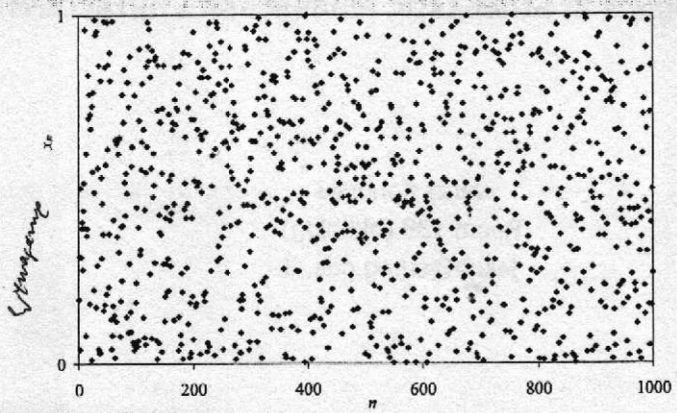
- described by a **probability distribution** over a set of values (**realizations**),
- this probability distribution exists, is either known or can be derived somehow (the **Bayesian approach**).

Opis stochastyczny

Random Variables and Stochastic Arrival Streams (2)

An example of a random variable is value returned by the function `random` (if we are deliberately oblivious to the algorithm of pseudorandom number generation). Its probability distribution is uniform on [0,1).

The first 1000 observed realizations:

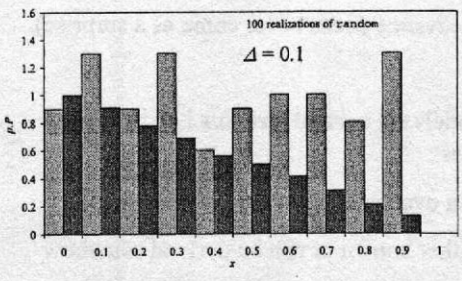


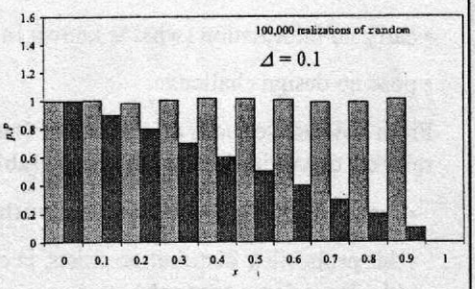
Empirical Distributions

Having realizations $x_1,\ldots,x_N$ one constructs a **histogram**:

- divide the range of possible realizations into bins of length Δ,
- count realizations falling into ith bin: $k_i = \#\{n \mid i\Delta \le x_n \le (i+1)\Delta\}$,
- at $i\Delta$, draw a bar of width Δ and height $p_i = (k_i/N)/\Delta$.

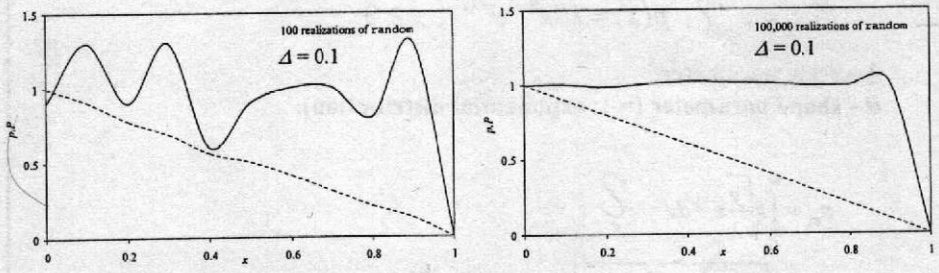
Cumulative histogram constructed analogously, with bars of height $C_i = \Sigma_{j \ge i} p_j$.





Empirical Distributions (2)

...or for readability, use smooth lines instead of bars:



Mean value: $x_{\mathrm{m}} = \dfrac{x_1 + \ldots + x_N}{N}$

Standard deviation (dispersion around mean): $\sigma_x = \sqrt{\dfrac{1}{N-1}\sum_{n=1}^{N}(x_n - x_{\mathrm{m}})^2}$

odchylenie standardowe

Theoretical Distributions

Probability density function and **complementary distribution function** – histograms one would obtain taking $N \to \infty$ and $\Delta \to 0$.

Probability density function: $p(x) = \lim_{\Delta \to 0} \dfrac{\Pr[x \le X < x + \Delta]}{\Delta}$

For any x' and x'', $\Pr[x' \le X < x''] = \int_{x'}^{x''} p(x)dx$.

Complementary distribution function: $P(x) = \Pr[X \ge x] = \int_{x}^{\infty} p(y)dy$

$x_{\mathrm{m}} = \int_{-\infty}^{\infty} xp(x)dx, \quad \sigma_x = \sqrt{\int_{-\infty}^{\infty}(x - x_{\mathrm{m}})^2 p(x)dx}$

Theoretical Distributions (2)

Modeling for engineering applications often uses **Weibull distribution**:

$$P(x) = e^{-\lambda x^{\theta}}, \quad p(x) = \lambda\theta x^{\theta-1} e^{-\lambda x^{\theta}}, \quad x \geq 0$$

λ – **scale parameter,**
θ – **shape parameter** (= 1: **exponential distribution**).

$$x_m = \int_0^{\infty} \sqrt[\theta]{\frac{y}{\lambda}} e^{-y} dy$$

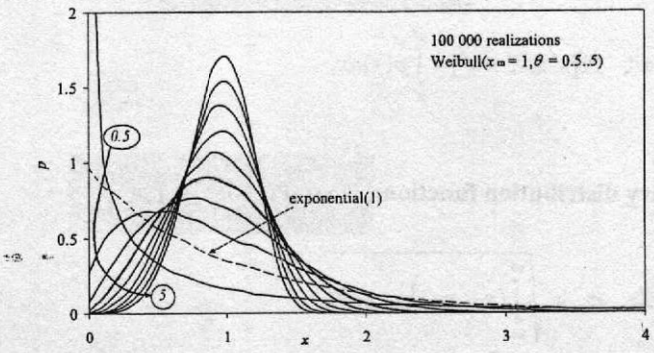
$$\sigma_x = \sqrt{\int_0^{\infty} \sqrt[\theta]{\frac{y^2}{\lambda^2}} e^{-y} dy - x_m^2}$$

Computer Generation of Arrival Streams

Given `random` generator that returns values (z_n),
how to generate pseudorandom numbers (x_n) with arbitrary $P(x)$?

x_n solves $P(x) = z_n$ e.g., for Weibull distribution: $x_n = \sqrt[\theta]{-\frac{\ln z_n}{\lambda}}$

(method of **inverted distribution function**; many others exist).

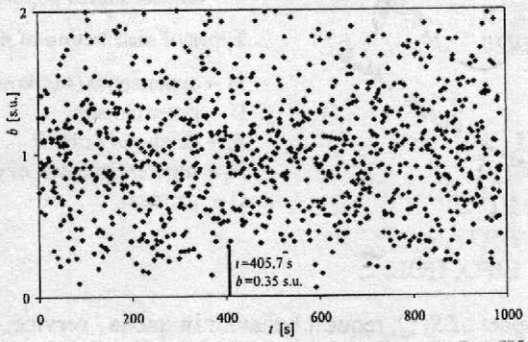


Computer Generation of Arrival Stream (2)

Introduce random variables:

A – interarrival interval (realizations: a_n)

B – request size (b_n)

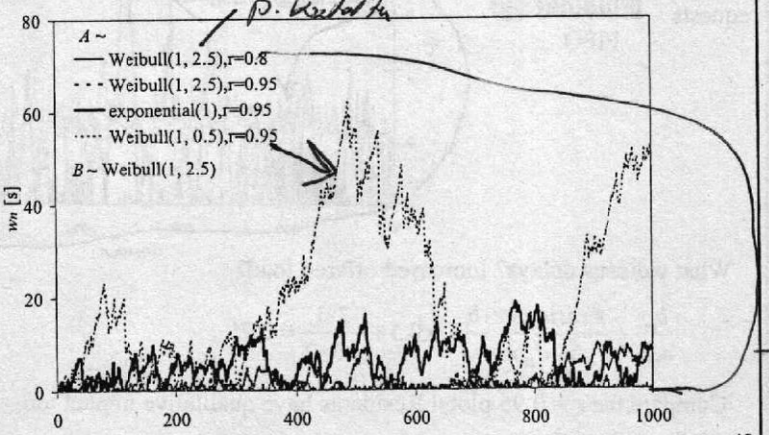


Arrival stream we met before had been generated as A, B ~ Weibull(1, 2.5)

Impact of Distribution of *A* and *B*

We have seen the impact of mean values a_m and b_m upon the queuing process (through offered load).

Is the *shape* of distributions of A and B equally relevant ?



very much so!

Kendall Notation

A/B/S/Q/J ...

Examples:

- M/M/1
- M/D/1/10
- $E_3/H_2/5/10$
- M/M/5/20
- M/G/*S*
- G/G/*S*
- $M^X/G/5$
- MMPP/G/1
- D+M/M/1//20
- M/D/1/*P***Q*
- M/G/1 LIFO, HOL, ...

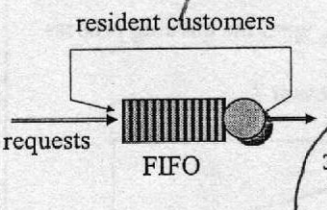
S – # processors
Q – buffer capacity
J – request source population size

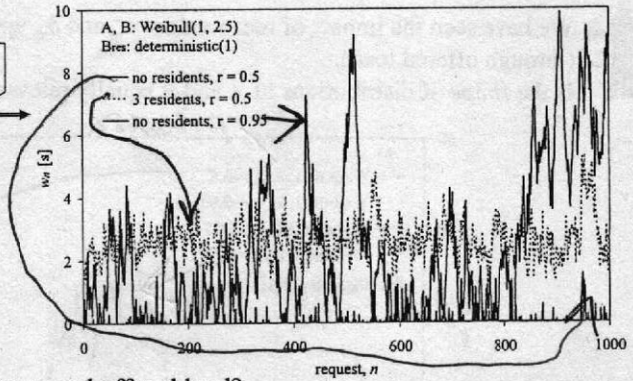
Types of distribution of *A*, *B* :

M – exponential (Markovian)
D – deterministic
E_k – Erlang of order *k*
H_k – hiperexponential of order *k*
G – general

What is the impact of *S*, *Q*, request behavior in queue / service, service rules?

Resident Customers





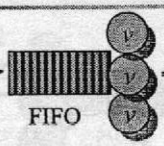
What worsens delays? Increased offered load?

$$\frac{b_m}{a_m v} + \frac{\#residents \cdot b_m}{d_m v} \approx 0.5 + \frac{3 \cdot 1}{3.5 \cdot 2} = 0.95$$

Compare the $r = 0.95$ plots! Residents have qualitative impact too.

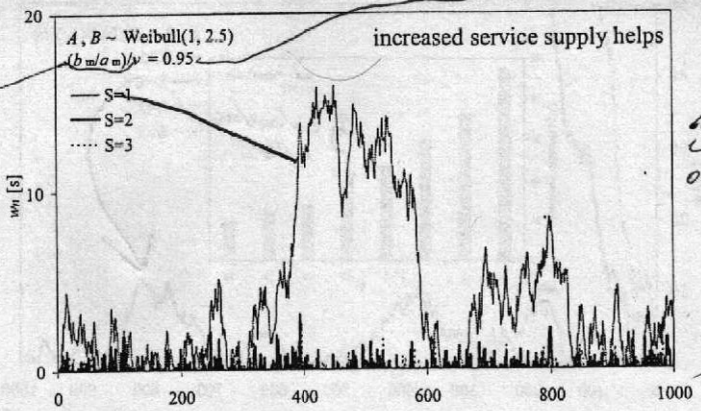
Multiple Servers: *A/B/S*

no grading
(any processor will do)



full availability
processor-bound service

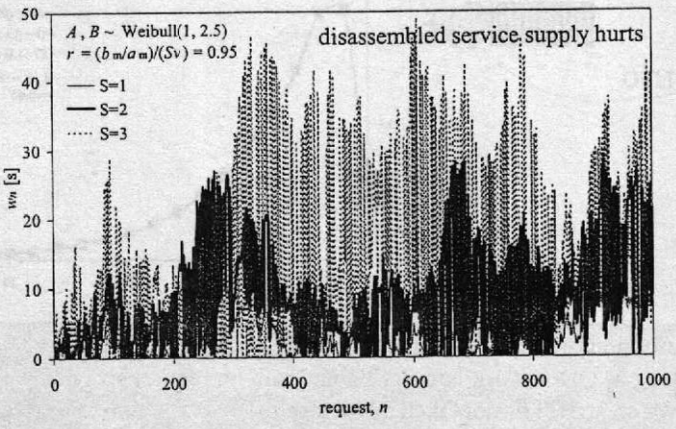
Keep service demand constant:



busy-hour load: created by the total demand if only one processor working

Multiple Servers: *A/B/S* (2)

Fairer plot: keep offered load constant. And?!



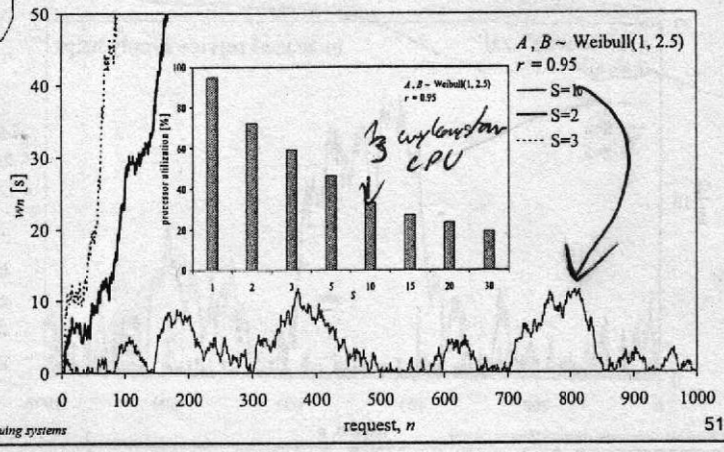
A/B/S with Grading and FIFO Blocking

grading:
specific processor

full availability
processor-bound service

motivation:
front-end processor
(switching fabric,
load balancer,...)

FIFO



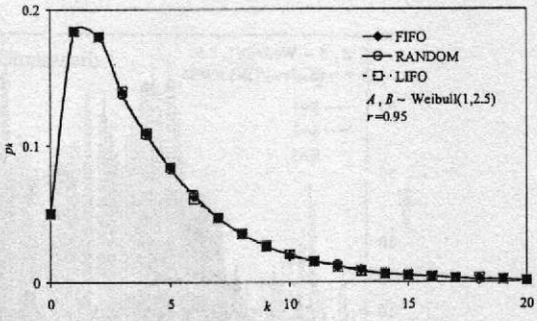
Queuing Discipline

RANDOM

FIFO

requests

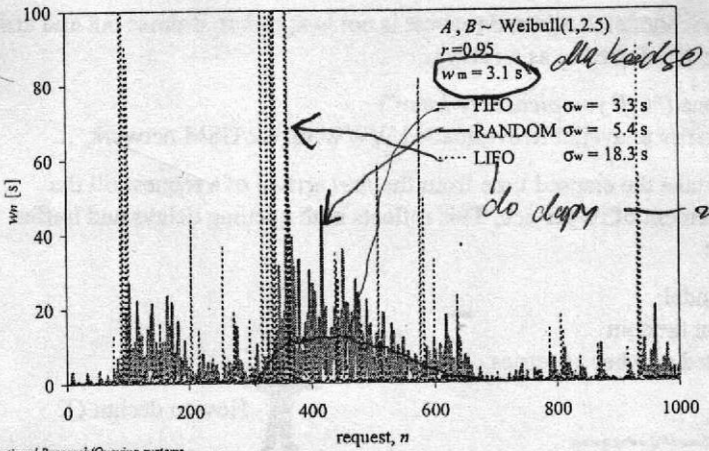
LIFO



Distribution of $N(t)$ is QD invariant!
(as long as QD is work-conserving and info on request size not known on arrival or not taken advantage of!)

Queuing Discipline (2)

Which QD cause largest w_m? largest σ_w?



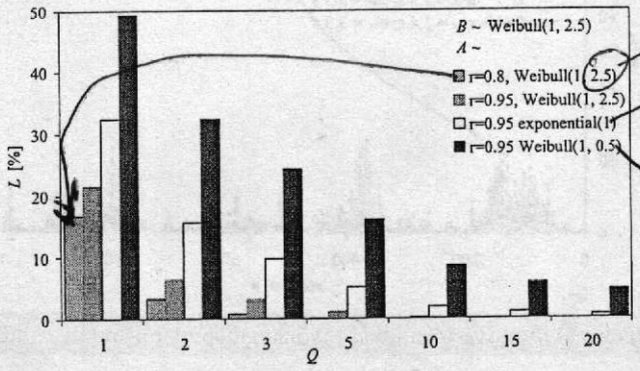
Finite Buffer Capacity: A/B/1/Q

maximum Q requests (in-service included)

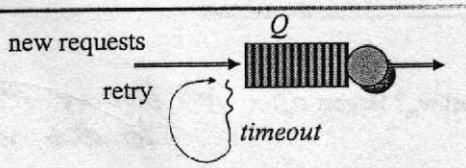
requests → *drop-tail rejection*

Motivation:
- memory cost?!
- delays, management cost!

Low waiting delays (w_n) traded for increased loss L due to buffer overflow.



A/B/1/Q with Retries



In a realistic model, a rejected request is not lost; rather, it times out and arrives again (this is referred to as a **retry**).

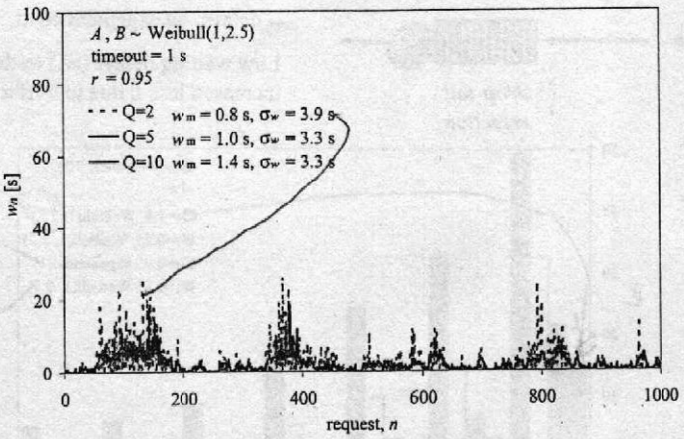
- busy tone (*"will you please try later"*)
- temporarily unavailable/overloaded WWW server/GSM network, ...

As w_n we take the elapsed time from the *first* arrival of a request till the commencement of its service. This reflects both queuing delays and buffer overflows.

Simple model:
- constant timeout
- unlimited number of retries.

How to design Q?

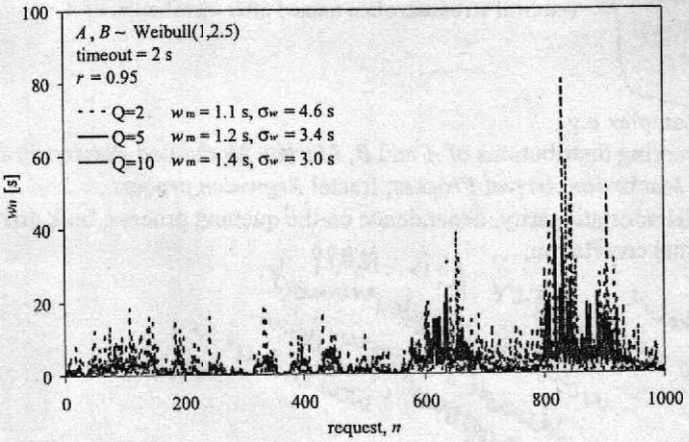
A/B/1/Q with Retries (2)



Smaller Q improve mean wait, worsen dispersion and distribution tail.

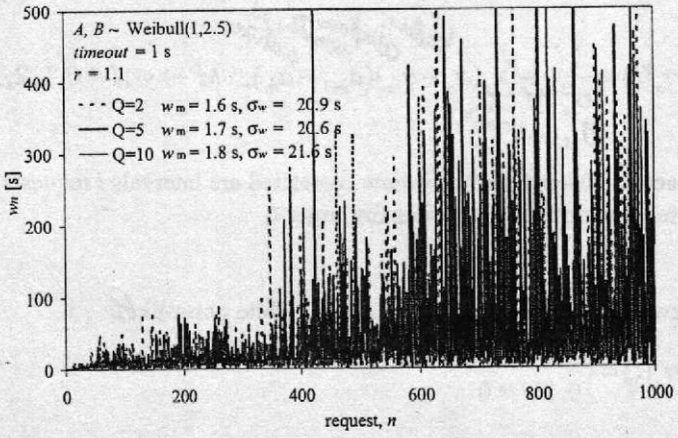
A/B/1/Q with Retries (3)

This becomes even more visible with more "patient" requests…



A/B/1/Q with Retries (4)

…whereas at heavier offered load, distribution tails become dramatic:



Common Models of Arrival Stream

- Weibull
- Bernouilli
- Erlang
- gamma
- …

(a_n) are **iid** – *independent, identically distributed* **renewal streams**, often named after distribution of A

- more complex e.g., time-varying distributions of A and B, *Markov Modulated Poisson Process*, *Batch Markovian Arrival Process*, fractal *Brownian process* , …
  – model nonstationarity, dependence on the queuing process, bulk arrivals, internal correlation, …

Arrival Stream: Impact of Autocorrelation

Are distributions of A and B enough to determine the queuing process (given fixed ν), or do we need information on the internal correlation in (a_n)?

$$corr_a(l) = \frac{1}{\sigma_a^2}\frac{1}{M}\sum_{n=1}^{M}(a_n - a_m)(a_{n+l} - a_m), \quad M \to \infty, l = 0,1,2,...$$

(**autocorrelation function** = how correlated are intervals l requests apart; correlation normally vanishes for larger l)

Renewal arrival stream is uncorrelated (white noise-like):

$$corr_a(l) = \begin{cases} 1, \text{if } l = 0 \\ 0, \text{if } l \neq 0 \end{cases}$$

Arrival Stream: Impact of Autocorrelation (2)

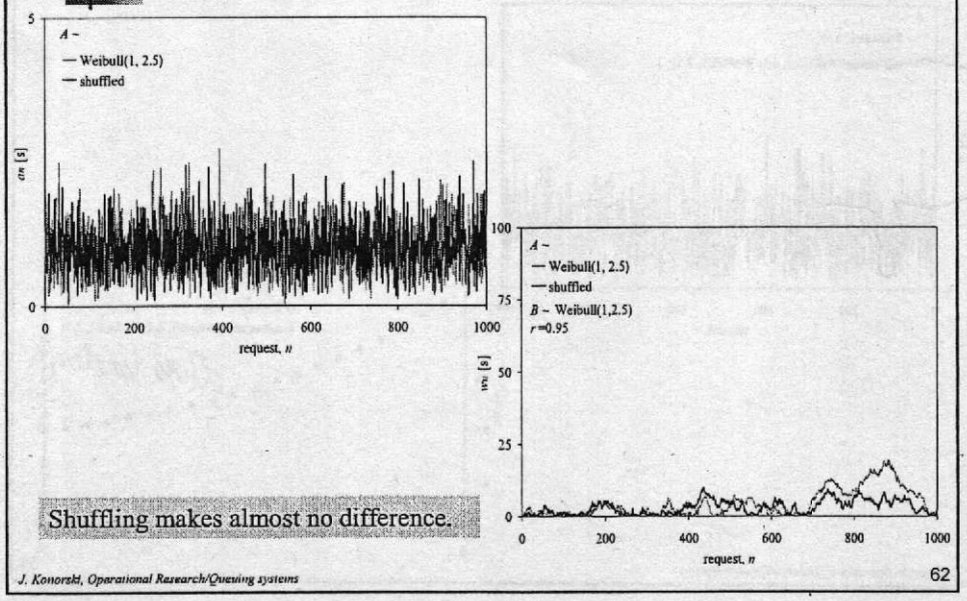
Experiment 1

Generate (a_n) and (b_n) according to Weibull(1, 2.5) distribution using the method of inverted distribution function. Input the obtained renewal arrival stream to a queuing system with $r = 0.95$.

Observe the queuing process (w_n).

Next, **shuffle** i.e., apply random permutation to the (a_n), use the same (b_n) and again observe (w_n).

Arrival Stream: Impact of Autocorrelation (3)



Shuffling makes almost no difference.

Arrival Stream: Impact of Autocorrelation (4)

Experiment 2

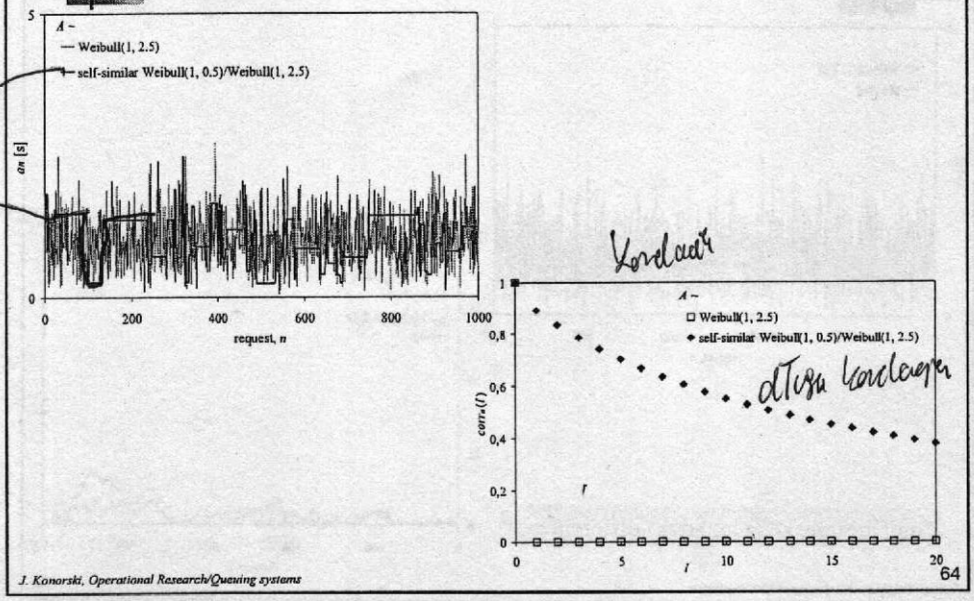
Take $A \sim$ Weibull(1, 2.5), and generate (a_n):

- as iid intervals from successive random numbers – renewal stream,
- by repeating each successive interval R times, where R is drawn from Weibull(1, 0.5) distribution.

In both variants, distribution of A is the same.
However, the second variant yields (a_n) with long-range autocorrelation – a **self-similar** arrival stream.

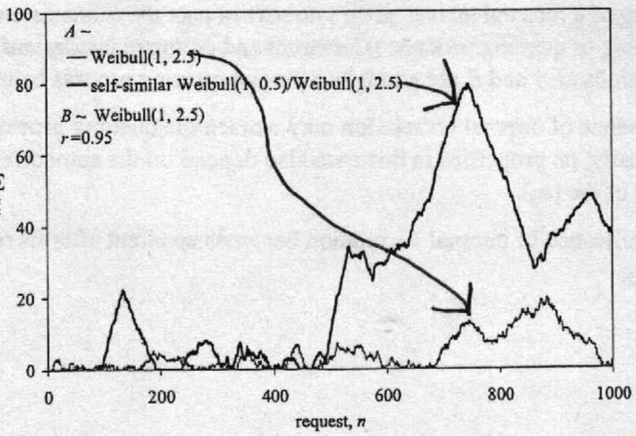
Using the same (b_n) as before, input the obtained arrival stream to a queuing system with $r = 0.95$.

Arrival Stream: Impact of Autocorrelation (5)



Arrival Stream: Impact of Autocorrelation (6)

Comparison of the queuing process for renewal and self-similar arrivals is quite spectacular… (note again: distributions of A and B are same in both variants!)

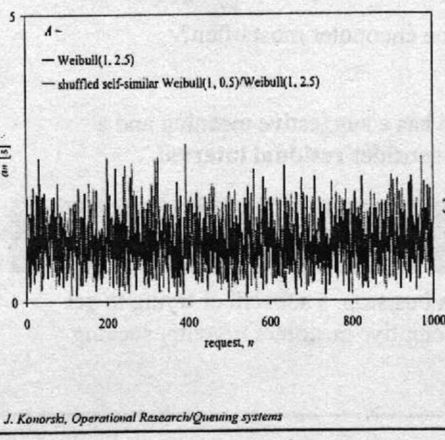


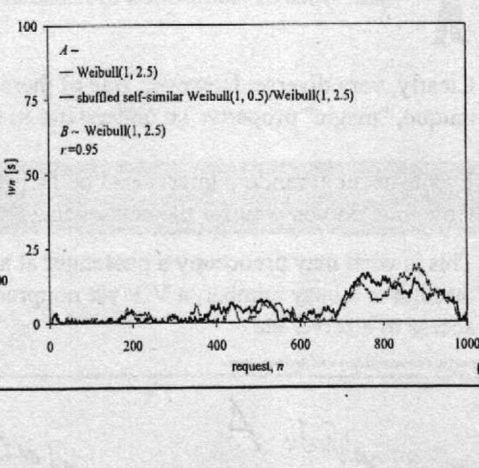
Arrival Stream: Impact of Autocorrelation (7)

Experiment 3

Shuffle the obtained self-similar (a_n), which of course removes long-range autocorrelation, but preserves distributions of A and B.

Use the same (b_n) and again compare with the renewal stream variant…





Arrival Stream: Impact of Autocorrelation (8)

Conclusions:

- Shuffling of a renewal arrival stream doesn't impact the (nonexistent) internal correlation, or queuing process. (Construct and compare histograms to be sure.) Distributions of A and B are enough to predict queuing process behavior.
- The presence of internal correlation may worsen the queuing process dramatically, its properties in this case also depend on the autocorrelation function of the (a_n).
- The significance of internal correlation becomes apparent after its removal by shuffling.

Renewal Arrival Stream: Residual Interval

Except special fields of research (heartbeat anomalies, overflows of the Nile, Web traffic analysis etc.), renewal streams model real-world arrival streams adequately. *Henceforth we focus upon them.*

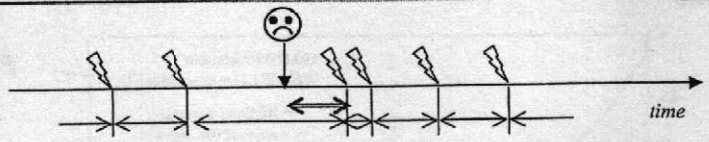
What types of distribution of A can one encounter most often?

Clearly, very diverse. However, one of them has a suggestive meaning and a unique, "magic" property. To understand it, consider **residual interval**.

Events occur at random intervals. You arrive at a random instant.
How long do you wait for the next event?

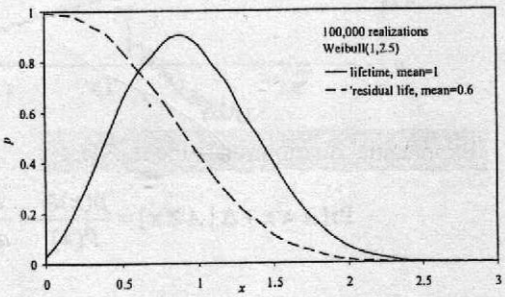
This is what may preoccupy a passenger at a bus stop, a subscriber trying to get through to a busy number, a VIP yet nonpreemptive customer urgently seeking access to a server etc.

Renewal Arrival Stream: Residual Interval (2)

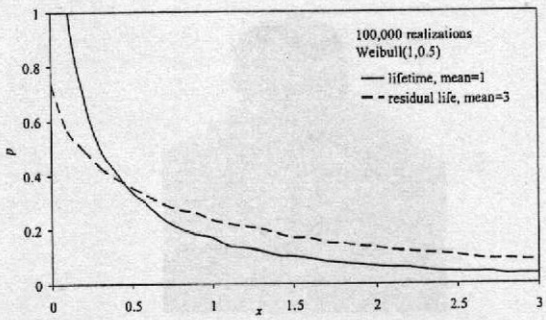


Buses run every hour, on average. "Statistical passenger" waits for half an hour?

Given the distribution of inter-bus interval (lifetime), what distribution does the residual interval have? Shifted towards smaller realizations?



Renewal Arrival Stream: Residual Interval (3)

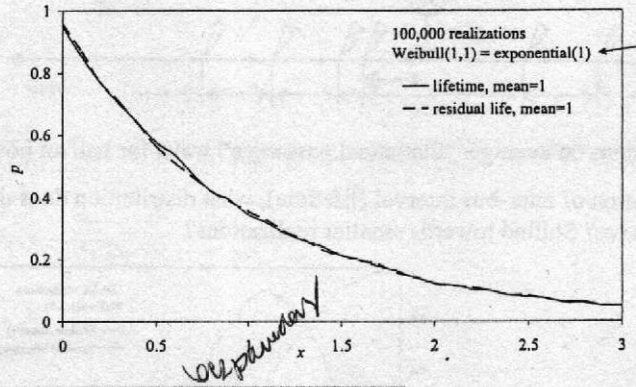


Paradox of residual life:

$$\bar{a}_{\mathrm{m}} = \frac{a_{\mathrm{m}}}{2}(1+c_a^2) \qquad c_a = \frac{\sigma_a}{a_{\mathrm{m}}} \text{ – coefficient of variation of } A$$

If $A \sim P(x)$ then $\tilde{A} \sim \tilde{p}(x) = P(x)/a_{\mathrm{m}}$

Renewal Arrival Stream: Residual Interval (4)



$P(x) = e^{-x/a_m}$

Exponential distribution is memoryless:

$$\Pr[A < x+\Delta \mid A \geq x] = \frac{p(x)\Delta}{P(x)} = \frac{\Delta}{a_m} \quad \text{regardless of } x$$

Poisson Arrival Stream

Simeon-Denis Poisson (1781-1840)

French mathematician, physicist and astronomer

investigated memoryless stochastic processes, now named after him, that model today's telecommunication and computer generated traffic

Poisson Arrival Stream (2)

At any instant of time, new request arrival occurs with constant probability.
In Kendall notation: M/... systems.

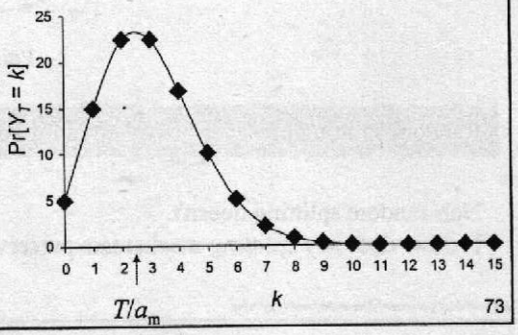
$$\Pr[\text{new request in } (t, t+\Delta)] = \Delta/a_m + o(\Delta)$$

Y_T – number of requests arriving during interval of length T

$$Y_{Tm} = T/a_m$$

$$\Pr[Y_T = k] = \frac{(T/a_m)^k}{k!} e^{-T/a_m}$$

Poisson distribution



Poisson Arrival Stream (3)

For stationary stochastic models, time average-type characteristics of queuing processes are determined using probability theory.

Steady state is then referred to as **statistical equilibrium**.
Consists in the time averages of interest stabilizing over time e.g., system state probabilities, loss probability, waiting time distributions etc.

PASTA (*Poisson Arrivals See Time Averages*): for Poisson arrivals, $p_k^+ \equiv p_k$

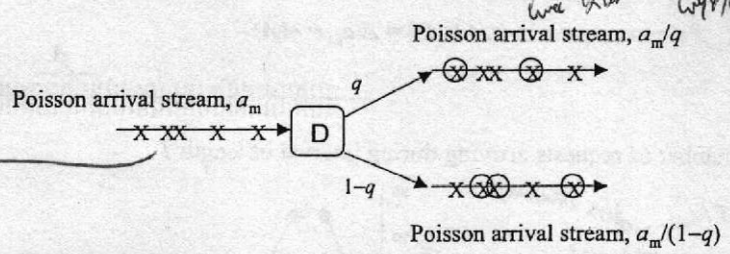
(requests arriving according to a Poisson stream "see" the same queue length distribution as does a random observer).

Proof: $$\Pr[N(t) = k \mid \text{request arrival in } (t, t+\Delta)] = \frac{\Pr[N(t) = k]\frac{\Delta}{a_m}}{\frac{\Delta}{a_m}} = \Pr[N(t) = k]$$

Hence, in M/G/*S*/*Q*: loss probability due to buffer overflow equals $L = p_Q^+ \equiv p_Q$.

Poisson Arrival Stream (4)

Random splitting:

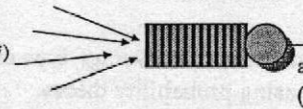


Random splitting preseves arrival stream's Poissonian nature!

Non-random splitting doesn't.
Neither does any splitting mechanism preserve the nature of non-Poisson arrivals.

Aggregation of Renewal Arrival Streams

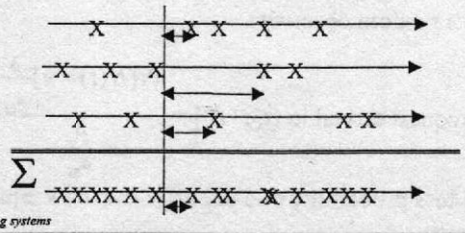
J independent components – renewal arrival streams (Web surfers, phone subscribers, mobile terminals, …)



aggregated arrival stream (input to backbone link, mainframe / Web server…)

$$\frac{T}{a_m} = \sum_{j=1}^{J} \frac{T}{a_m^{(j)}}$$ (in particular, for identical components, $a_m = \frac{a_m^{(j)}}{J}$)

What interval distribution does the system "see"? At least a renewal stream?
Not necessarily. In general, analytic calculation difficult if not impossible.

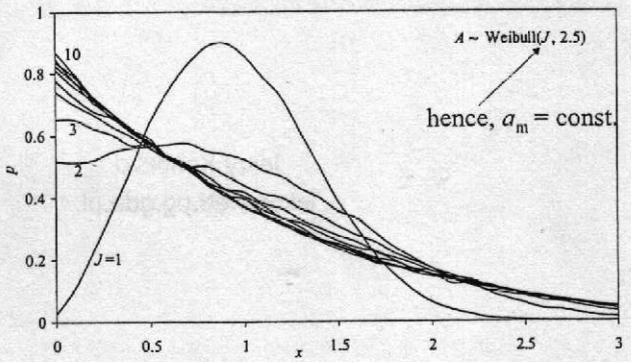


residual intervals:
$\tilde{A} = \min\{\tilde{A}^{(1)}, \ldots, \tilde{A}^{(J)}\}$

Aggregation of Renewal Arrival Streams (2)

With $J \to \infty$, but $a_m > 0$ (a practical model) and independent component streams:

Aggregated arrival stream is Poisson (**Palm theorem**).



Aggregation of Renewal Arrival Streams (3)

Proof: omitted :)

$$\tilde{A} = \min\{\tilde{A}^{(1)},\ldots,\tilde{A}^{(J)}\},\ \text{so}\ \Pr[\tilde{A} \geq x] = \prod_{j=1}^{J} \Pr[\tilde{A}^{(j)} \geq x]$$

$J \to \infty$, but $a_m > 0$ (a practical model). That is, $a_m^{(j)} \to \infty$.

For any finite x, $x / a_m^{(j)} \to 0$ and we can neglect $Y_x^{(j)} > 1$.

$$\Pr[\tilde{A}^{(j)} \geq x] = 1 - \Pr[Y_x^{(j)} > 0] \approx 1 - \Pr[Y_x^{(j)} = 1] \approx 1 - Y_{x,m}^{(j)} \approx 1 - \frac{x}{a_m^{(j)}}$$

Finally,

$$\Pr[\tilde{A} \geq x] \approx \prod_{j=1}^{J}\left(1 - \frac{x}{a_m^{(j)}}\right) \approx \exp\left(-\sum_{j=1}^{J}\frac{x}{a_m^{(j)}}\right) = e^{-x/a_m}$$

since for $x, y, z, \ldots << 1$, the following holds: $(1-x)(1-y)(1-z)\ldots \approx e^{-(x+y+z+\ldots)}$

Since residual interval in the aggregated stream is exponentially distributed, so is interval itself!

Operational Research
Queuing Systems 3: Markovian Models

Jerzy Konorski
jekon@eti.pg.gda.pl

Markovian Systems

Recall that queuing theory deals with systems and processes that can be observed, measured, and simulated.

Mathematical analysis may be useful too, but only if leads to *simple* and *insightful* results.

Take an *A/B/...* system. Is it easy to predict its characteristic theoretically?

Yes, if necessary simplifications are made:
- not too drastic
  keep models close to reality!
  (or else face charges of shaping the lock to fit the key!)
- yet bold enough
  keep problems tractable!
  get universal insight!

Example: **Markovian queuing systems.**

Markovian Systems (2)

Exhibit the apparently unusual, but most useful **Markov property.**

(which, however, they share with a huge number of real-world dynamical systems – technical, physical, social, biological, economic, ...)

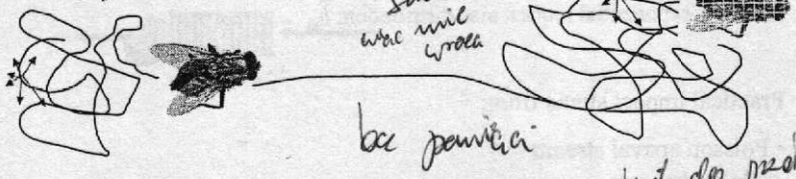
$state(t + \Delta) = f[state(t), \phi]$

"noise" (random external input at time t, in general dependent on the current state, but independent of earlier ones)

Markovian Systems (3)

- The fly...

selection of momentary direction



- card shuffling: card order in the deck (*selection of cut point*)
- gambler's capital / population (*current interest / growth rate*)
- $trend(t + \Delta) = (1 - c) \cdot trend(t) + c \cdot \phi(t)$ (*current observation*)
- $market\_share(t + \Delta) = \phi \cdot market\_share(t)[1 - market\_share(t)]$ (*current management performance*)
- Internet topology (*number and points of attachment of new networks*)

Markovian Systems (4)

Markow - pawee' ograniczona

Andrei A. Markov (1856-1922)

Russian mathematician

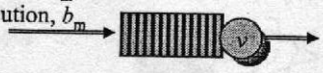
investigated stochastic processes of finite memory, now named after him, that model many natural and man-made phenomena

Markovian Systems (5)

...are those queuing systems encoded as **M/M/...**

wykładnicze pamięta tylko chwilę przed
dlatego Markow

Poisson arrival stream, a_m
exponential request size distribution, b_m



Practical impact stems from:

- Poisson arrival stream
  - Palm theorem
  - random splitting
  - PASTA
  - pessimistic (meaning: fail-safe) performance characteristics — bezpieczny, ale nie wybitnie pesymistyczny przypadek
- exponential request size distribution: crude approximation of
  - call holding time, Web / P2P file transfer
  - batch processing time
  - ...

Markovian Systems (6)

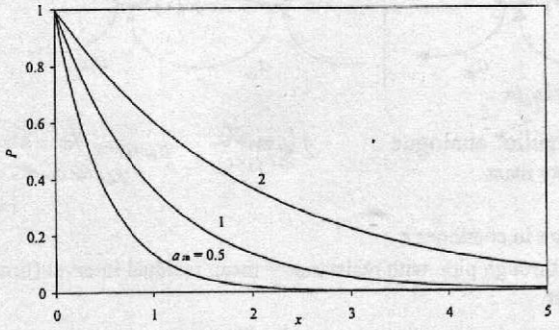
1 proc

v – (constant) processor speed

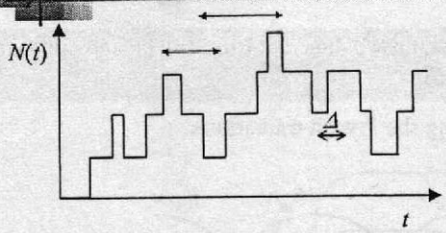
$A \sim \mathrm{M} = \text{exponential}(a_\mathrm{m})$: $\Pr(A \geq x) = e^{-x/a_\mathrm{m}}$

$B \sim \mathrm{M} = \text{exponential}(b_\mathrm{m})$: $\Pr(B \geq x) = e^{-x/b_\mathrm{m}}$

$\Pr[\text{service time} \geq x] = e^{-x/(b_\mathrm{m}/v)}$



M/M/1 Queuing System



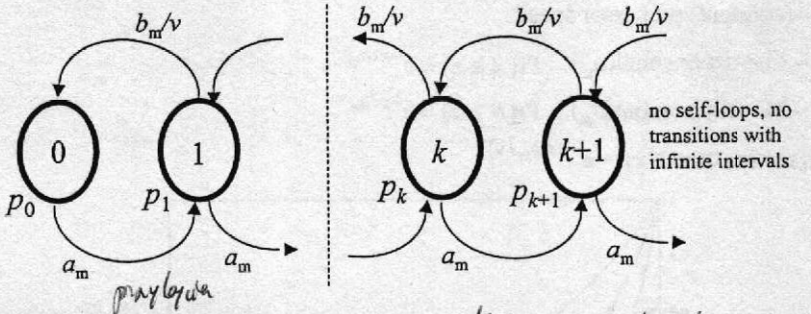
$N(t)$ – **birth and death** process
a_m – mean **birth interval**
b_m/v – mean **death interval**

Let $p_k(t) = \Pr[N(t) = k]$. We know that $\lim_{t \to \infty} p_k(t) = p_k$ (**statistical equilibrium**).
State distribution (p_k) can be derived from **birth and death equations.**

- Exponential distribution has no probability mass at 0
  $\Rightarrow$ suppose $N(t) = k$, what can happen between t and $t + \Delta$?
  practically, only one of the following: nothing / 1 birth / 1 death (if $k > 0$),
  (state transitions between neighbor states only)

- Exponential distribution is memoryless (Markov property)
  $\Rightarrow$ residual interval (time to occurrence) of next birth / death is statistically the same as the interval between consecutive births / deaths

M/M/1 Queuing System (2)

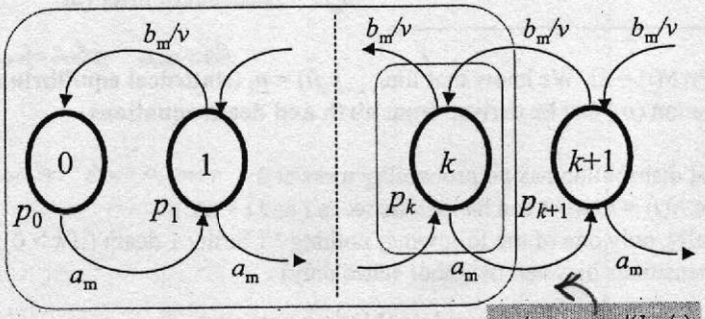


Suggestive "hydraulic" analogue :
- liquid ~ probability mass
- state ~ container
- p_k ~ liquid pressure in container k
- transition ~ flow through pipe with resistance = mean residual interval (time to occurrence) of respective event

M/M/1 Queuing System (3)

In statistical equilibrium, in- and outflow must balance out for *any* closed contour. (These are our birth and death equations.)

For convenience, select contours crossing the fewest transitions!



$$p_k/a_m = p_{k+1}/(b_m/\nu)$$

$$\Rightarrow \frac{p_{k+1}}{p_k} = \frac{b_m}{a_m \nu} = r$$

M/M/1 Queuing System (4)

$p_k = p_0 \cdot r^k \qquad p_0 + p_1 + p_2 + \ldots = 1 \Rightarrow p_0 = 1 - r$

Hence mean queue length and further, by Little's theorem, mean waiting delay:

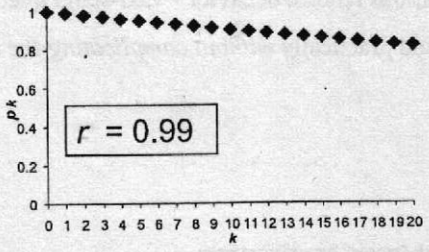
$$N_m = \frac{r}{1-r}$$

$$d_m = a_m N_m = \tau_m \cdot \frac{1}{1-r}$$

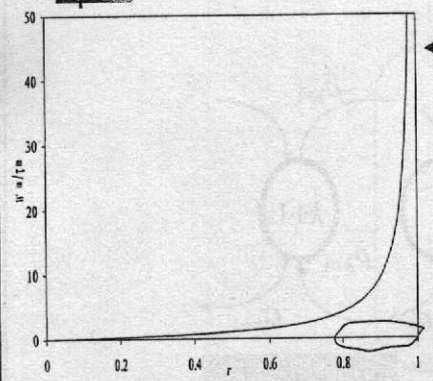
$$w_m = d_m - \tau_m$$

very suggestive!

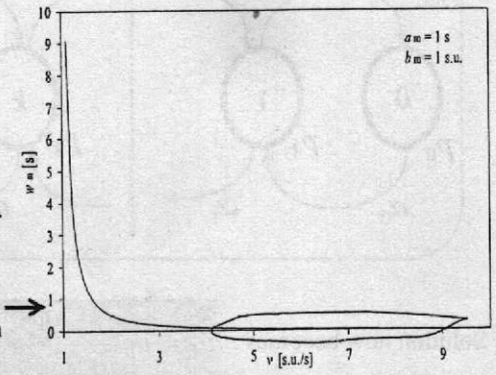
The most frequent queue length?!



M/M/1 Queuing System (5)



w_m/τ_m vs. r:
processor overload region



w_m vs. v:
processor overdimensioning region

Distribution of waiting time does depend on QD.
For FIFO, will be found with a little richer math… shortly.

M/M/... Systems

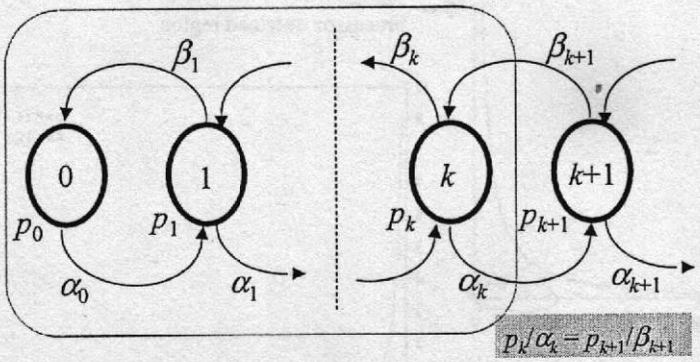
Birth-and-death processes help to analyze far richer and more realistic Markovian models of queuing systems featuring e.g.,

- finite buffer capacity (no-retry, *drop-tail*)
- multiple processors (no grading), perhaps in a queue-dependent number,
- queue-dependent arrival stream (intelligent terminal-type request sources)
- various request behavior – taxi-stand queue / token bucket, impatience, ...

...and practically *without complicating the math*!.

M/M/... Systems (2)

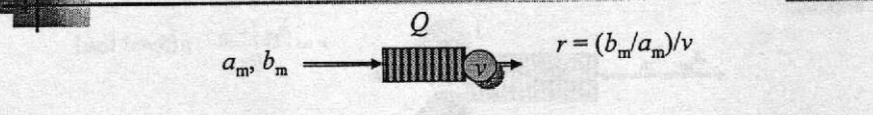
Make mean transition interval state-dependent:

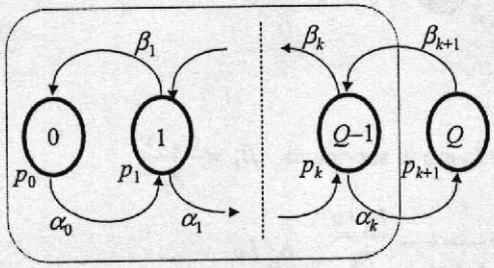


Solution now becomes : $p_k = p_0 \frac{\beta_1 \ldots \beta_k}{\alpha_0 \ldots \alpha_{k-1}}, \quad k = 0,1,2,\ldots$

whence all the interesting characteristics: L, p_0, N_m, d_m, w_m,...

M/M/1/Q System





Arrivals stop at $k = Q$:

$$\alpha_k = \begin{cases} a_m, \text{ if } k < Q \\ \infty, \text{ if } k \geq Q \end{cases} \qquad \beta_k \equiv b_m / v$$

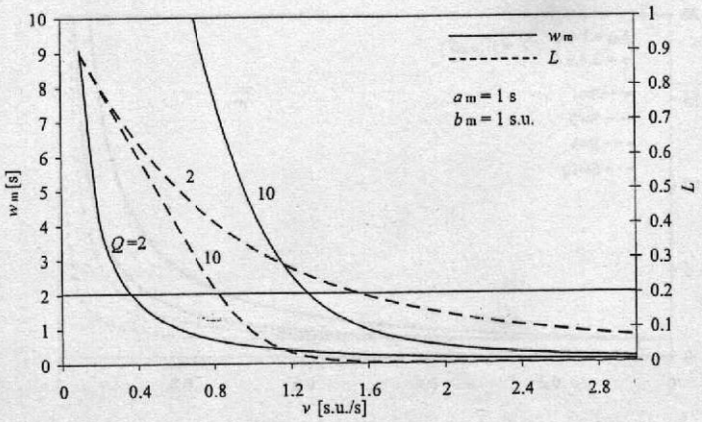
$$p_k = p_0 r^k, \quad k = 0,\ldots,Q$$

$$L = p_Q^{+} \underset{\text{PASTA}}{=} p_Q = \begin{cases} \dfrac{1-r}{1-r^{Q+1}} r^Q, \text{ if } r \neq 1 \\ \dfrac{1}{Q+1}, \text{ if } r = 1 \end{cases} \qquad d_m = \frac{N_m}{\frac{1-L}{a_m}} \qquad w_m = d_m - b_m / v$$

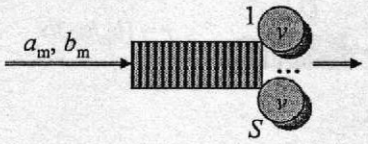
M/M/1/Q System (2)

Example: Design processor speed v to ensure $w_m \leq 2$ s and $L \leq 20\%$.

Which requirement is the more stringent? What Q value is the best?



M/M/S System

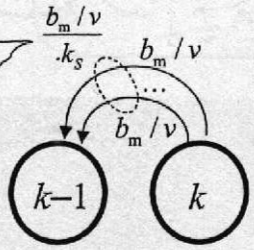


$r = \frac{b_m / a_m}{S\nu}$ offered load

$a_k \equiv a_m$

Of k requests in system, $\min\{k, S\} = k_S$ are in service $\Rightarrow$ $\beta_k = \frac{b_m / \nu}{k_S}$

k_S transitions,
each with mean time to occurrence = b_m/ν

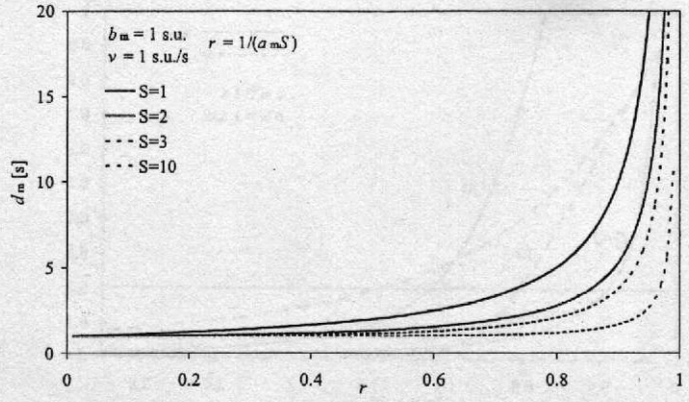


M/M/S System (2)

$$p_k = p_0 r^k \frac{S^{k_S}}{k_S!}$$

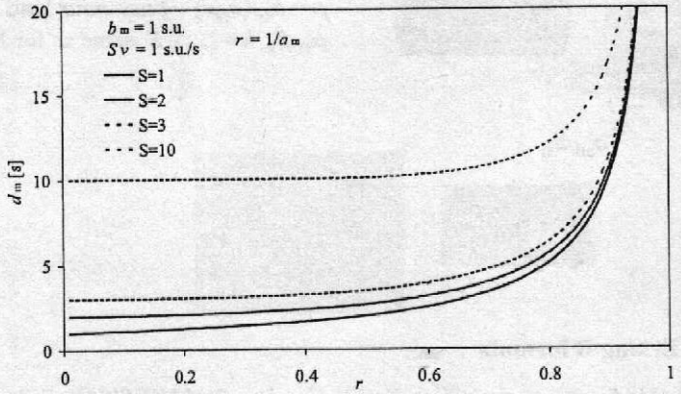
Hence N_m; d_m follows from Little's theorem.

Constant speed of each processor:



M/M/*S* System (3)

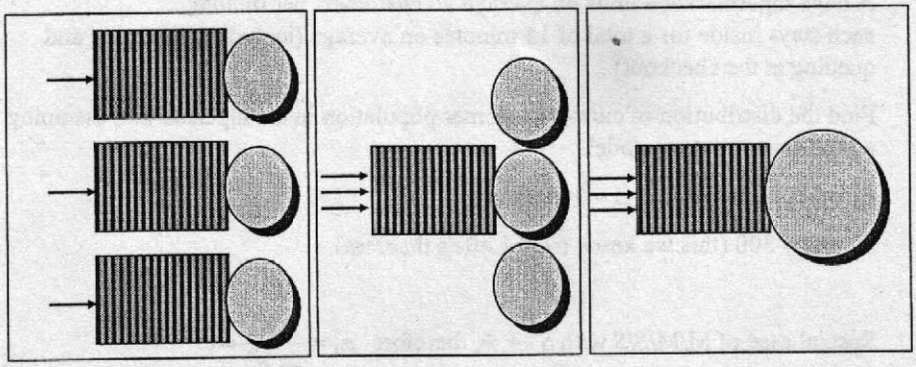
Constant speed of the processor *pool*:



(We have seen already that disassembled service supply hurts!)

On Benefits of Buffer Sharing and Pooled Service Supply

$r = 0.667$

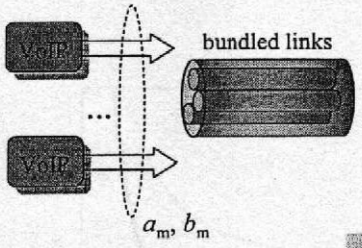


3 x M/M/1 — $d_m = 1$ s

M/M/3 — $d_m = 0.48$ s

M/M/1 — $d_m = 0.33$ s

M/M/*S*/*S* System



no waiting room

v – single link capacity

$\rho = b_m/(a_m v)$ – *busy-hour* load

α_k, β_k, $k = 0,\ldots,S$ same as for M/M/*S*

$$p_k = p_0 \frac{\rho^k}{k!} \qquad p_S = L = \frac{\frac{\rho^S}{S!}}{\sum_{k=0}^{S} \frac{\rho^k}{k!}}$$

- famous **Erlang B formula**
- magic: holds for *any* request size distribution i.e., for M/G/*S*/*S* (!)
- online calculators exist (`www.voip-calculator.com/calculator/`)

M/M/∞ System

A huge hipermarket admits on average 20 customers per minute, each stays inside for a total of 15 minutes on average (including shopping and queuing at the checkout) .

Find the distribution of current customer population in the hipermarket, assuming a Markovian system model.

$a_m = 3$ s, $b_m/v = 900$ s, $\rho = 300$ erlangs

$N_m = \rho = 300$ (this we know from Little's theorem).

Special case of M/M/*S*/*S* with $S \to \infty$, therefore $p_0 = \frac{1}{\sum_{k=0}^{\infty} \frac{\rho^k}{k!}} = e^{-\rho}$

$p_k = e^{-\rho} \frac{\rho^k}{k!}$ – Poisson distribution (!)