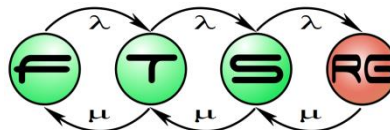


Performance Modelling 2

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group



Reminder

- **Stable state:**

- Calculating with average values
- $\lambda = X$ (arrival rate = throughput)

- **Maximum throughput (X^{max}):**

- Upper bound of the reachable throughput
- $X^{max} = \frac{K}{T}$ (in case of K resource instances)

- **Utilization (U):**

- Ratio of the actual and maximum throughputs
- $U = \frac{X}{K} \times T$ (in case of K resource instances)

Visitation number

Little's law

Zip's law

Changes in Workload

CONTENT

Visitation number

Little's law

Zip's law

Changes in Workload

VISITATION NUMBER

How do we do dimensioning?

- In general, resources are assigned to the activities
 - The (average) execution time of the activity is given
→ X^{max} of the activity can be computed
- E.g. in the Neptun system registering for a lecture utilize the DB server for 100 ms
 - $T = 100 \text{ ms}$
 - $X^{max} = \frac{1}{T} = 10 \frac{\text{registration}}{\text{sec}}$

Knowing the process model (e.g. the user behaviour), what is the maximum throughput of the whole system?

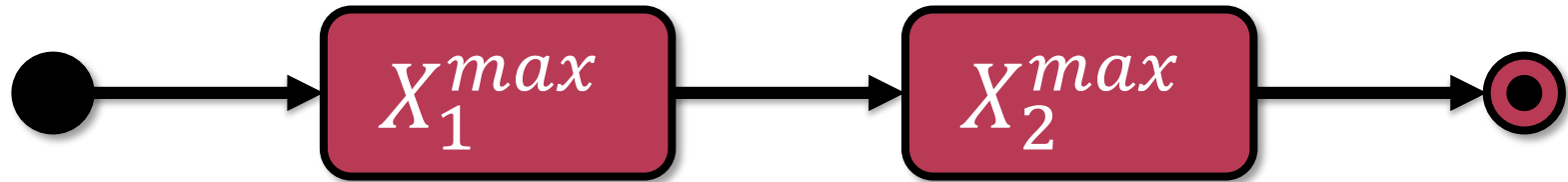
Definition: Visitation Number

The **visitation number** indicates the average number of times a given activity/subprocess runs in a single execution of the whole process.

- How many times the process visits/repeats the given activity during a single execution?
- During a single execution of a process one of its activities can run not at all, or once, or several times. (Decisions, loops!)
 - If a choice between different outputs is described with probabilities, then these probabilities also play a role in determining the visitation numbers.

Sequential Composition

χ^{max}



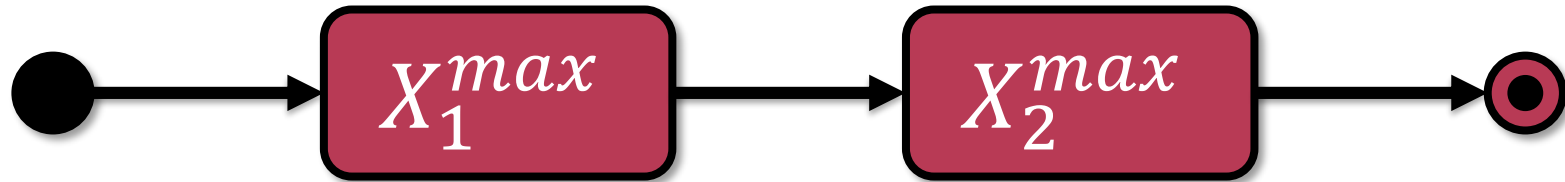
Each activity will be visited **once**.

It is no use if the one activity is fast, tokens will pile up at the other one.

E.g. in a citizen centre taking a ticket (300/hour), administration work (2/hour)

Sequential Composition

χ^{max}

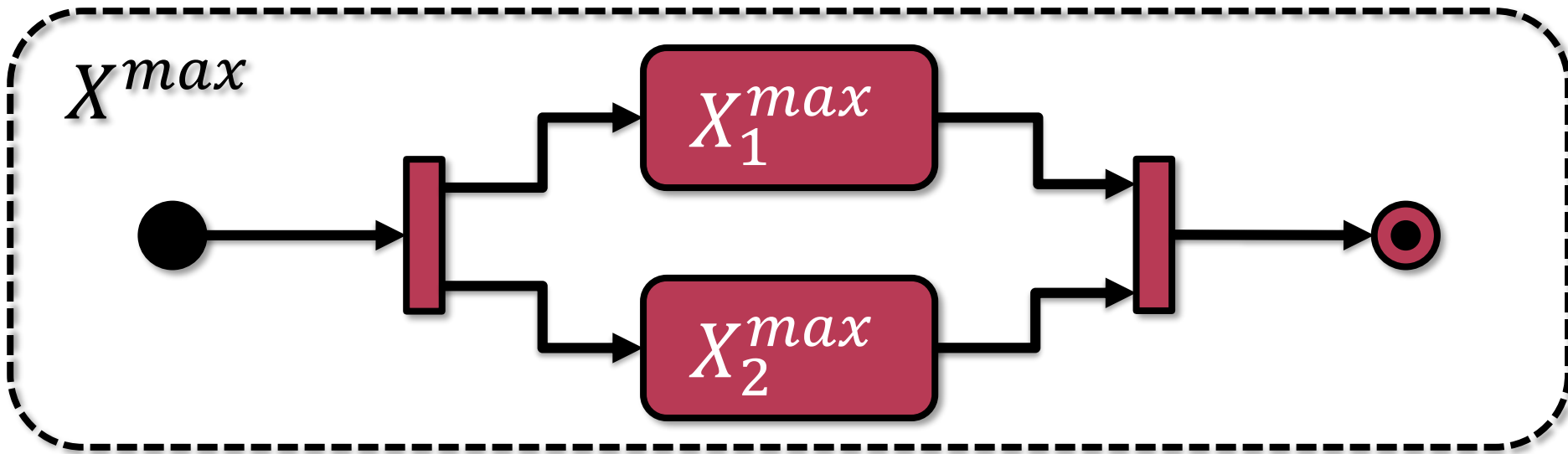


$$\chi^{max} = \min(X_1^{max}, X_2^{max})$$

Bottleneck:

The component with the minimum throughput (or the corresponding resource).

Parallel Composition

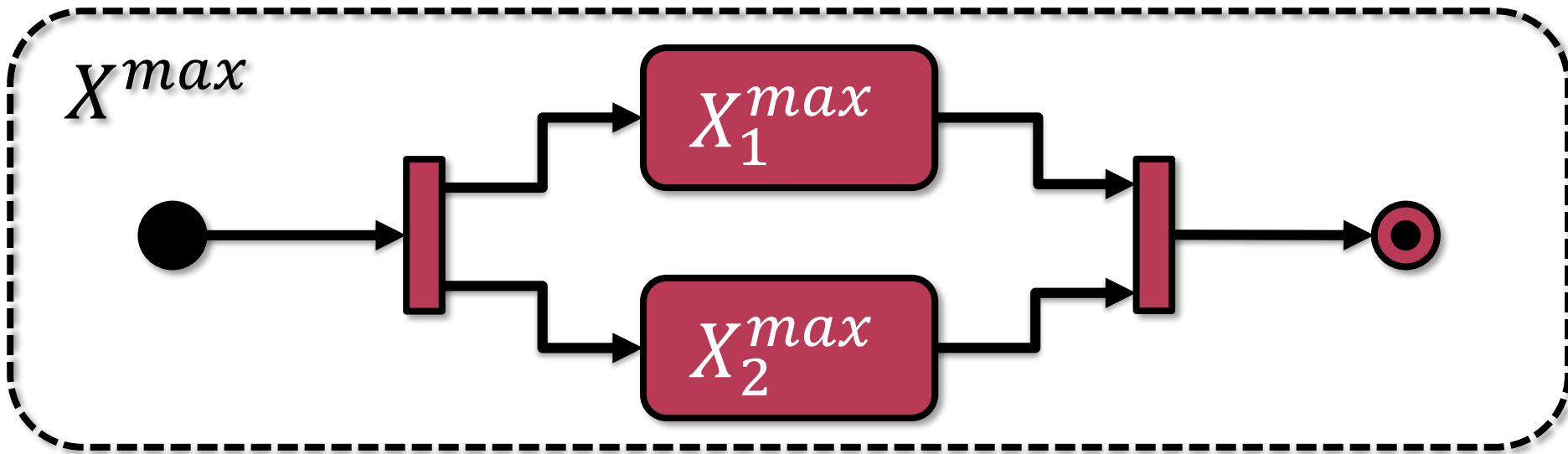


Each activity will be visited **once**.

It is no use if the one activity is fast, tokens will pile up at the other one.

E.g. at evaluating the exams
entry test (30/hour),
exercises (12/hour)

Parallel Composition

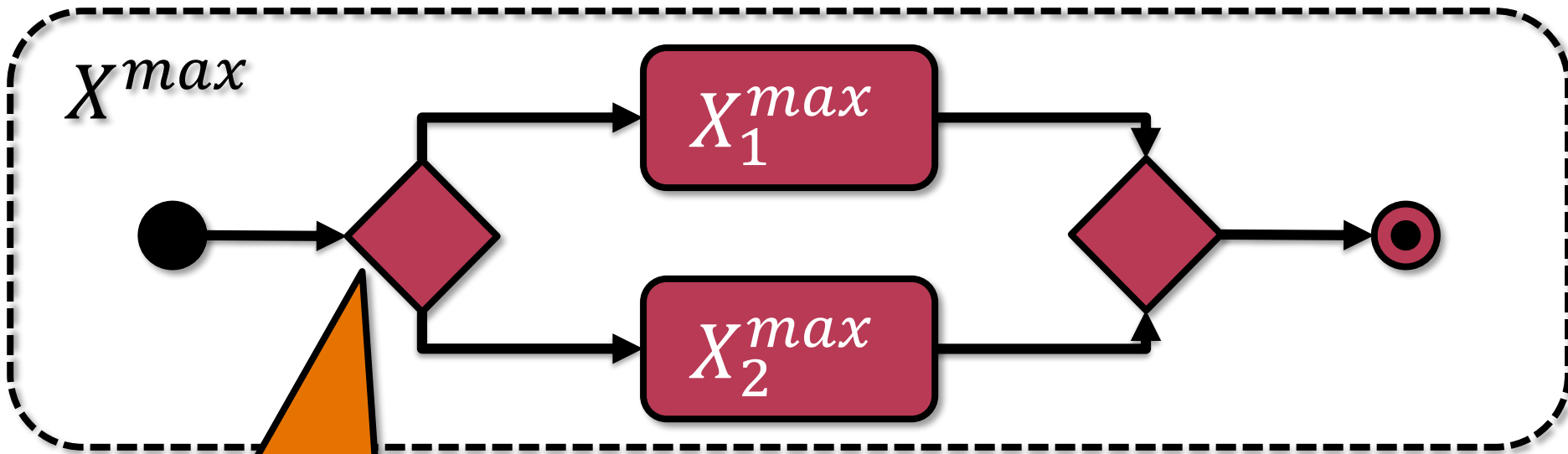


$$\chi^{max} = \min(\chi_1^{max}, \chi_2^{max})$$

Bottleneck:

The component with the minimum throughput (or the corresponding resource).

Composition of Free Choice

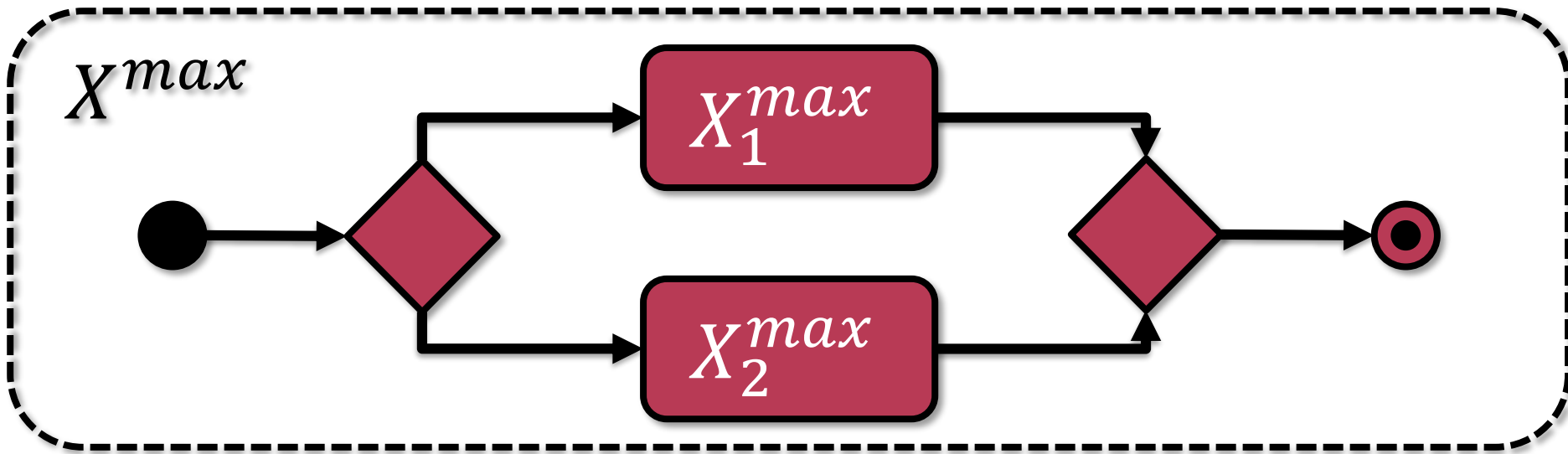


$$\chi^{max} = \chi_1^{max} + \chi_2^{max}$$

The tokens may choose a branch freely. If a branch is saturated, the other one may still be stable.

E.g. in a supermarket:
K cash-desks,
10 customer/h each

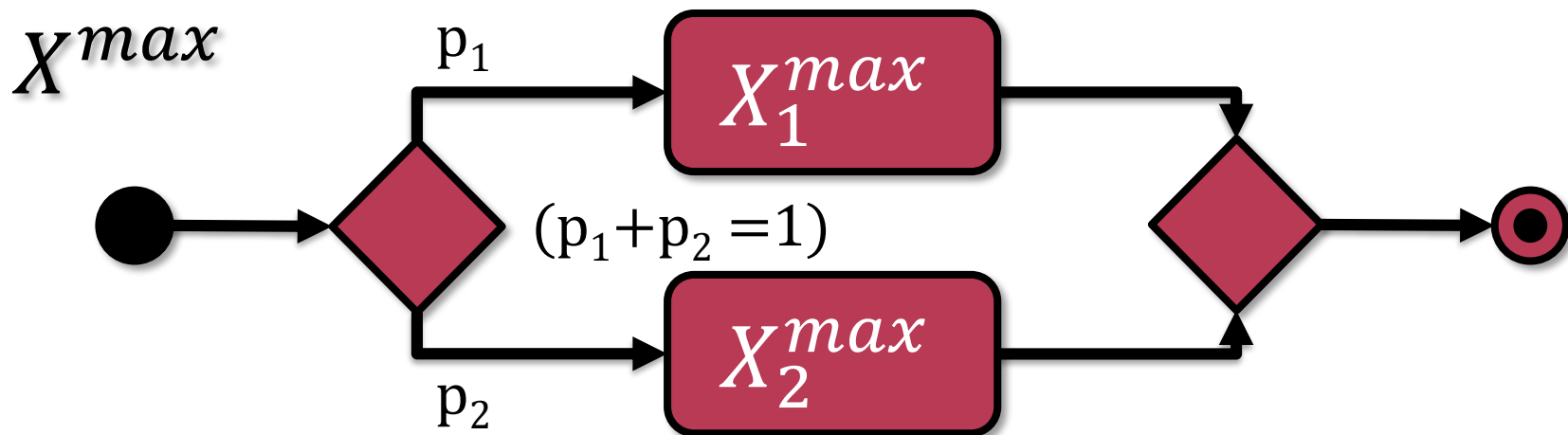
Composition of Free Choice



$$\chi^{max} = \chi_1^{max} + \chi_2^{max}$$

Note: Sometimes we assign the number of resources to the activities, and do not draw them several times explicitly. (see Simulation)
Condition: the resources finish the activity in the same time

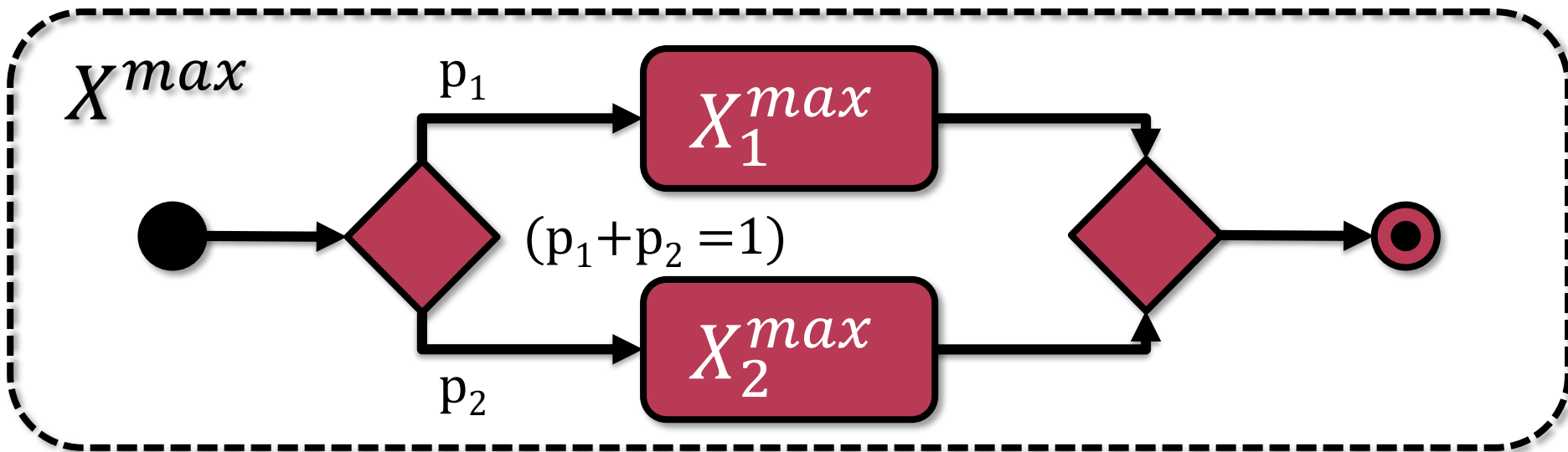
Composition of Stochastic Choice



Activity X_1 will be visited p_1 times in average, activity X_2 will be visited p_2 times in average.

The tokens choose the first or second branch with probability p_1 and p_2 , respectively. In the whole process one token out of $\frac{1}{p_1}$ or $\frac{1}{p_2}$ will visit the activities.

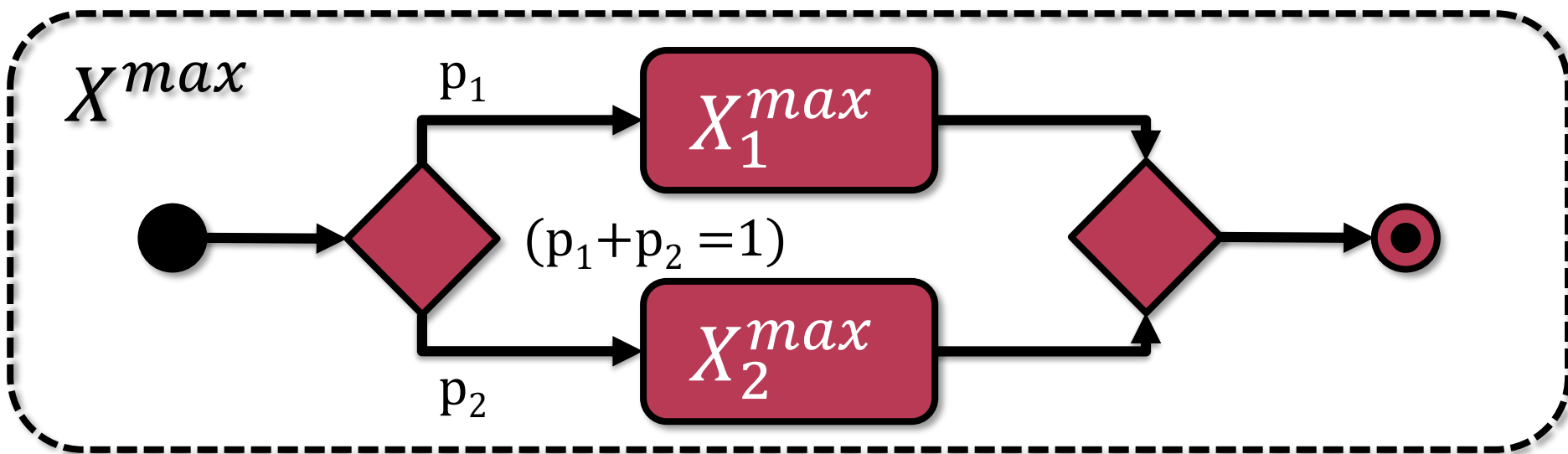
Composition of Stochastic Choice



$$X^{max} = \min\left(\frac{1}{p_1} \times X_1^{max}, \frac{1}{p_2} \times X_2^{max}\right)$$

E.g. the user of a web shop:
with 20% chance he buys the good (20/sec),
with 80% chance he cancels it (200/sec)

Composition of Stochastic Choice



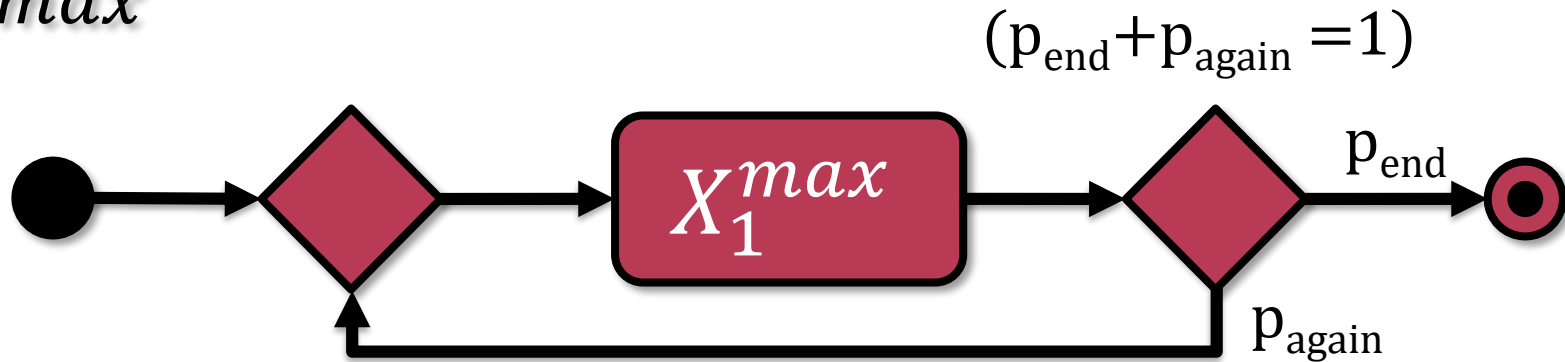
$$X^{max} = \min\left(\frac{1}{p_1} \times X_1^{max}, \frac{1}{p_2} \times X_2^{max}\right)$$

Bottleneck:

The component with the minimum throughput (or the corresponding resource).

Composition of Loop

χ^{max}

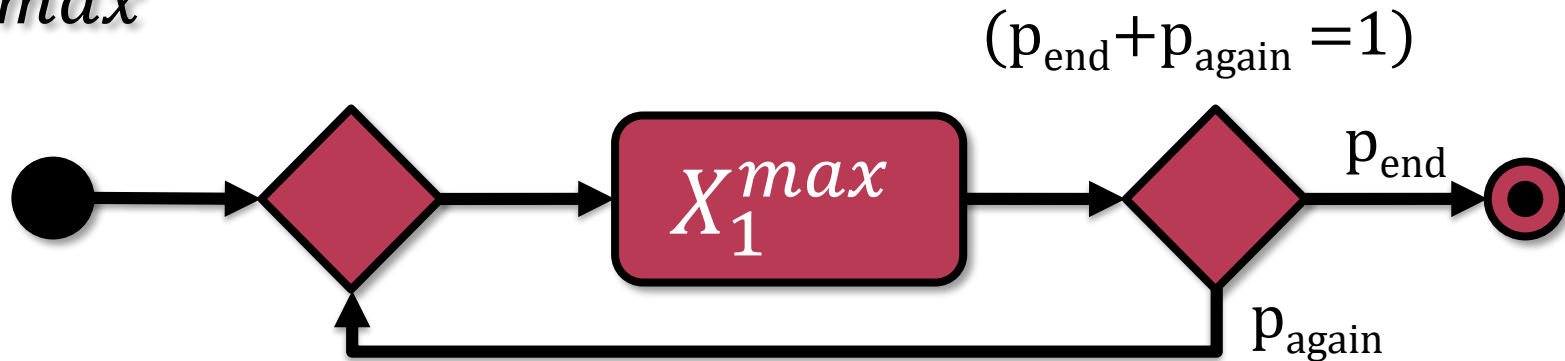


Activity X_1 will be visited $\frac{1}{p_{end}}$ times in average.

$\frac{1}{p_{end}}$ is the expected number of repetitions
(see in *Probability theory* lecture).

Composition of Loop

X^{max}



$$X^{max} = \frac{1}{p_{end}} \times X_1^{max} = p_{end} \times X_1^{max}$$

E.g. the user in a system:
With 10% he leaves, with 90% he asks again.
15 req/s, in avg. 10 req/session

Visitation Number

Choice:
$$X^{max} = \min(\frac{1}{p_1} \times X_1^{max}, \frac{1}{p_2} \times X_2^{max})$$

Loop:
$$X^{max} = \frac{1}{p_{end}} \times X_1^{max} = p_{end} \times X_1^{max}$$

- **Visitation number:** indicates the average number of times a given activity/subprocess runs in a single execution of the whole process.
 - In case of choice it is the decision possibility itself
 - In case of cycle it is the expected number of iterations

Visitation Number

Maximum throughput depending on the visitation number:

$$X^{max} = \frac{1}{v} \times X_1^{max}$$

- **Visitation number:** indicates the average number of times a given activity/subprocess runs in a single execution of the whole process.
 - In case of choice it is the decision possibility itself
 - In case of cycle it is the expected number of iterations

Visitation Number

Maximum throughput depending on the visitation number:

$$\frac{1}{x^{max}} = v \times \frac{1}{x_1^{max}}$$

- **Visitation number:** indicates the average number of times a given activity/subprocess runs in a single execution of the whole process.
 - In case of choice it is the decision possibility itself
 - In case of cycle it is the expected number of iterations

Visitation Number

Execution time depending on visitation number:

$$T_{process} = v \times T_{task}$$

- **Visitation number:** indicates the average number of times a given activity/subprocess runs in a single execution of the whole process.
 - In case of choice it is the decision possibility itself
 - In case of cycle it is the expected number of iterations

Visitation number

Little's law

Zip's law

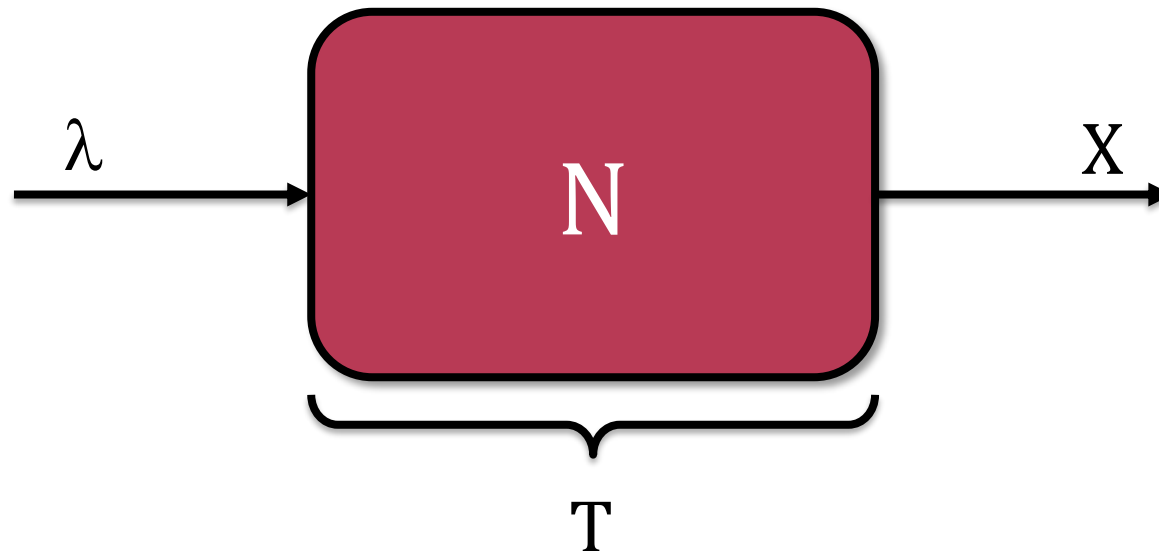
Changes in Workload

LITTLE'S LAW

The basic formula

Little's Law

- λ : arrival rate $\left[\frac{1}{s}\right]$ X : throughput $\left[\frac{1}{s}\right]$
- T : time spent in system $[s]$
- N : number of tokens in system $[1]$



Little's Law

- In stable state ($\lambda = X$) Little's law holds:

$$N = X \times T$$

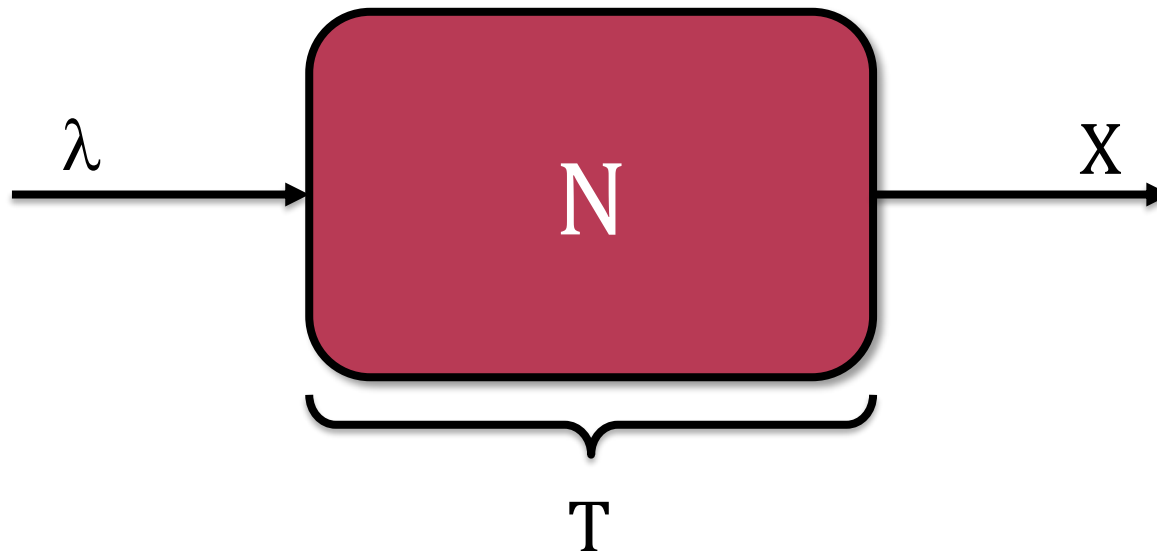


Illustration of Little's Law

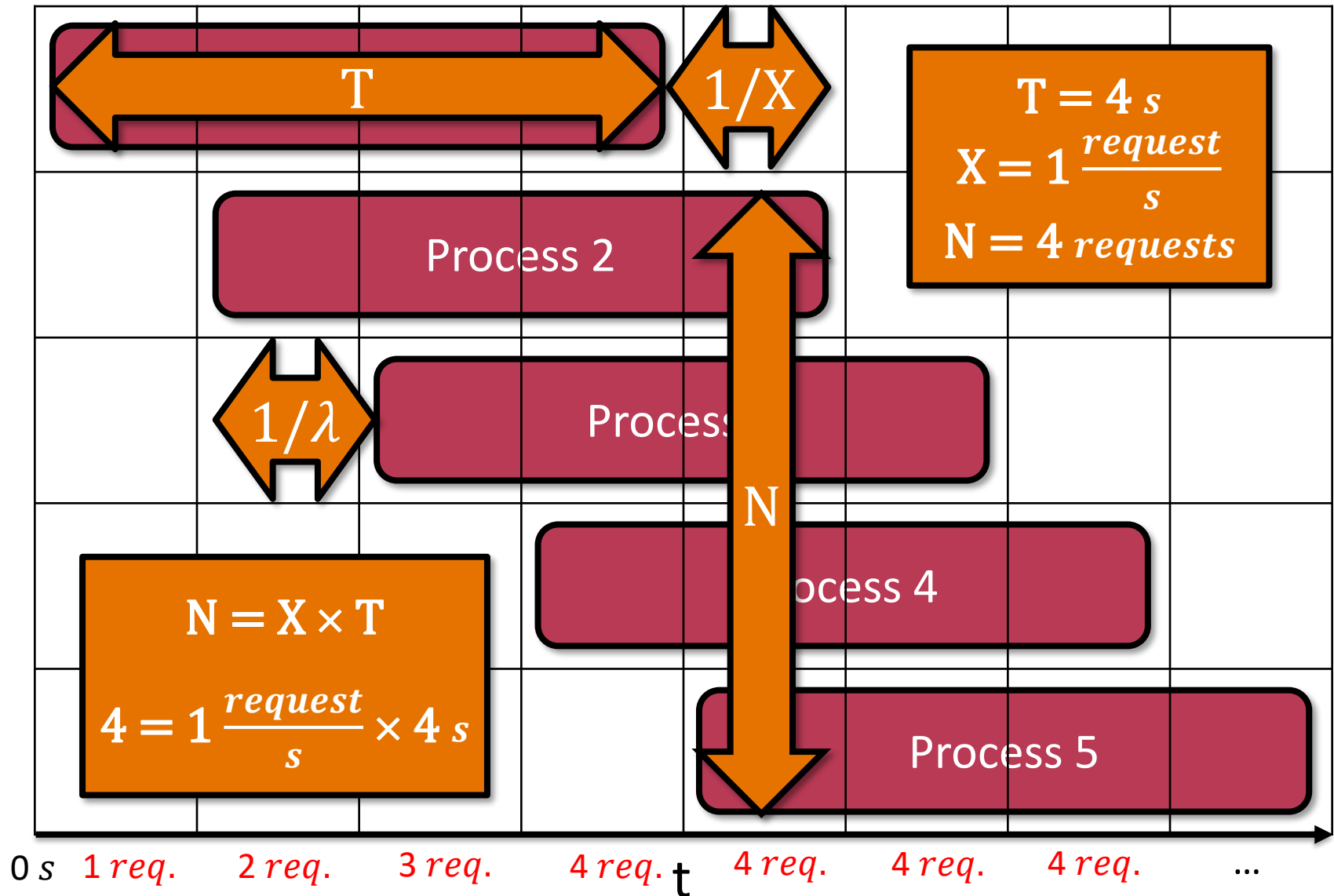
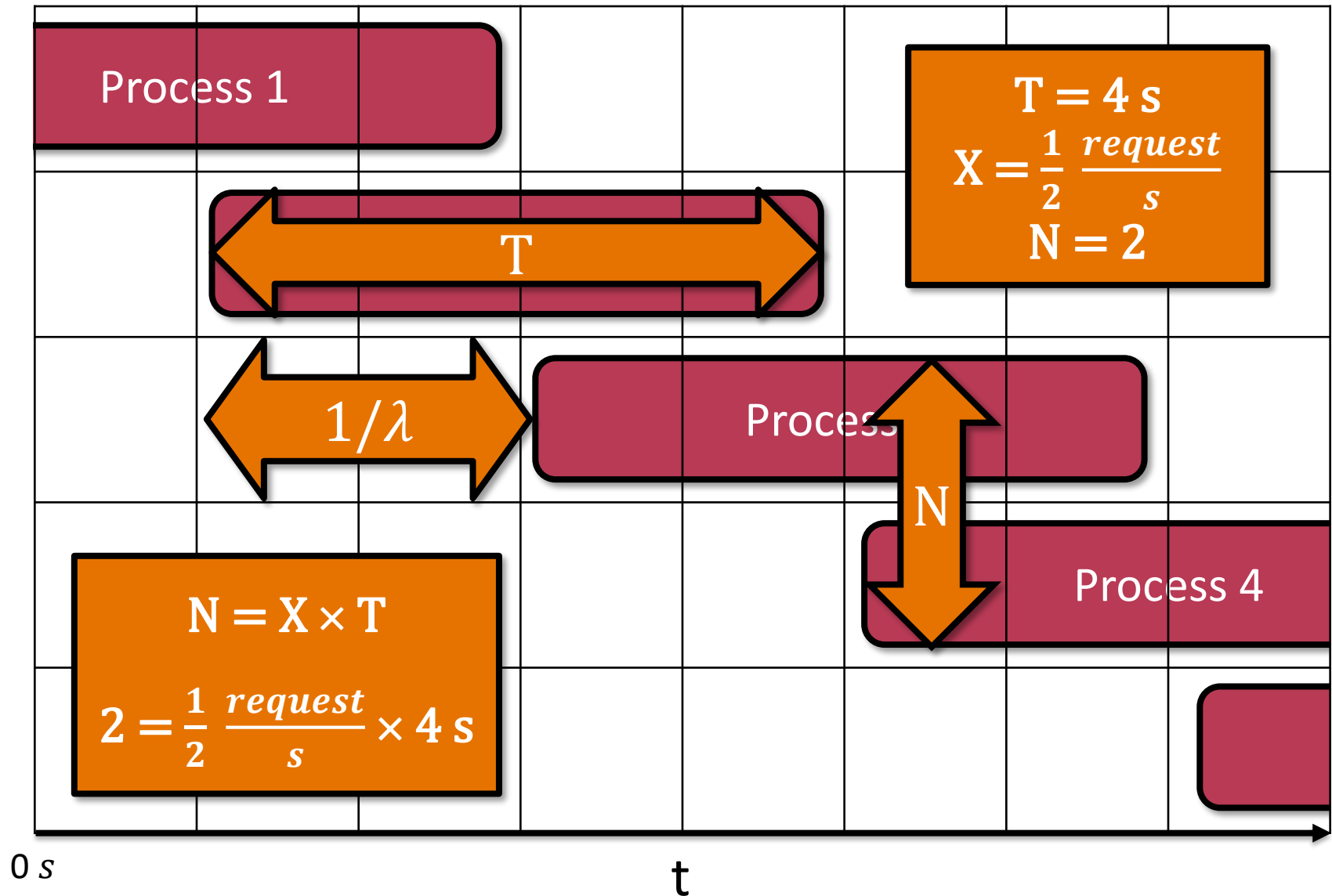


Illustration of Little's Law



Utilization and Little's Law

- K resource instances: maximum K process instances under execution at the same time
- Little's law:
number of process instances under execution (N)
- Average utilization can be derived as follows:

$$U = \frac{X}{K} \times T = \frac{X \times T}{K} = \frac{N}{K}$$

Utilization for K
resource instances

Little's law
($N = X \times T$)

Visitation number

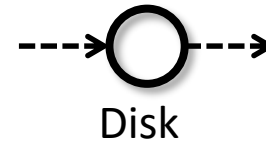
Little's law

Zip's law

Changes in Workload

LITTLE'S LAW: PRACTICAL EXAMPLES

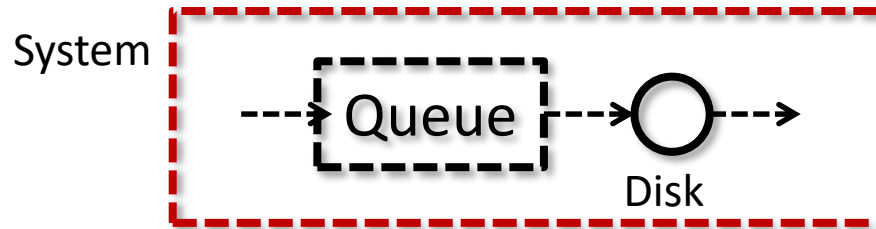
Little's Law – Example



- Resource: disk
- Serves 40 requests per second (no overlap)
- Serving 1 request takes up 0,0225 seconds on average
- How much is the utilization?

$$U = X \times T_{disk} = 40 \frac{\text{request}}{s} \times 0,0225 s = 0,9 = 90\%$$

Little's Law – Example

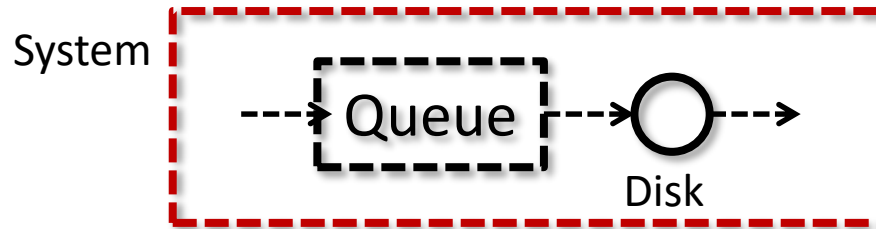


- Queuing before disk
- Disk: 40 *request/s*
- Average requests in system: 4

Average time a request spends in the system?
(T_{system})

Average queuing time? (T_{waiting})

Little's Law – Example



- Queuing before disk
- Disk: 40 *request/s*
- Average requests in system: 4

Queuing plus
disk serving time

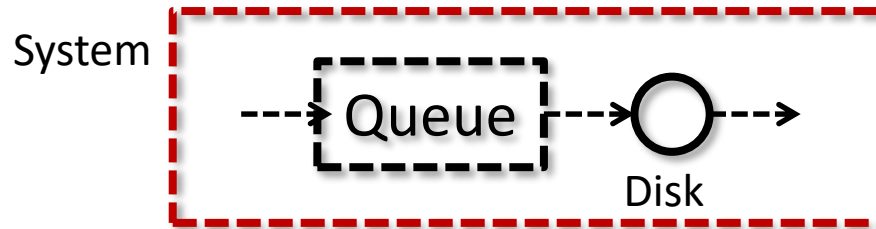
System

$$N = X \times T \rightarrow T_{\text{system}} = 4 \text{ requests} / 40 \frac{\text{request}}{\text{s}} = 0,1 \text{ s}$$

Average queuing time

$$(T_{\text{system}} - T_{\text{disk}}) = (0,1 \text{ s} - 0,0225 \text{ s}) = 0,0775 \text{ s}$$

Little's Law – Example



- Queuing before disk
- Disk: 40 *request/s*. In average 0,9 request
- Average requests in system: 4

Average number of requests in queue?

$$(N_{\text{system}} - N_{\text{disk}})$$

4 requests – 0,9 request = 3,1 requests

Little's Law in Practice

■ Simulation

- Dobson&Shumsky
- <https://youtu.be/UjzXQPGBaNA>

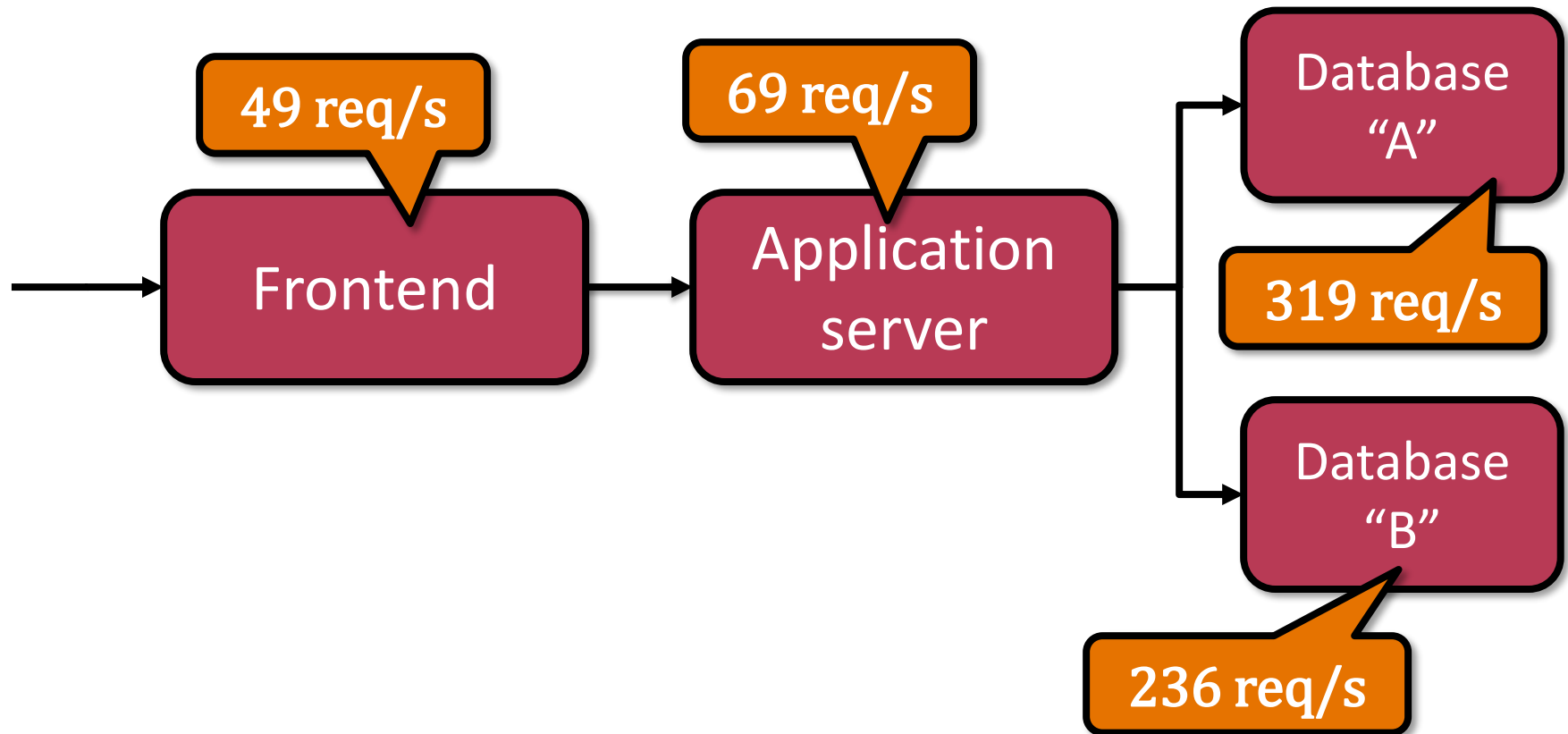
■ Why it is taught

- <http://pubsonline.informs.org/doi/pdf/10.1287/ited.7.1.106>

■ Examples

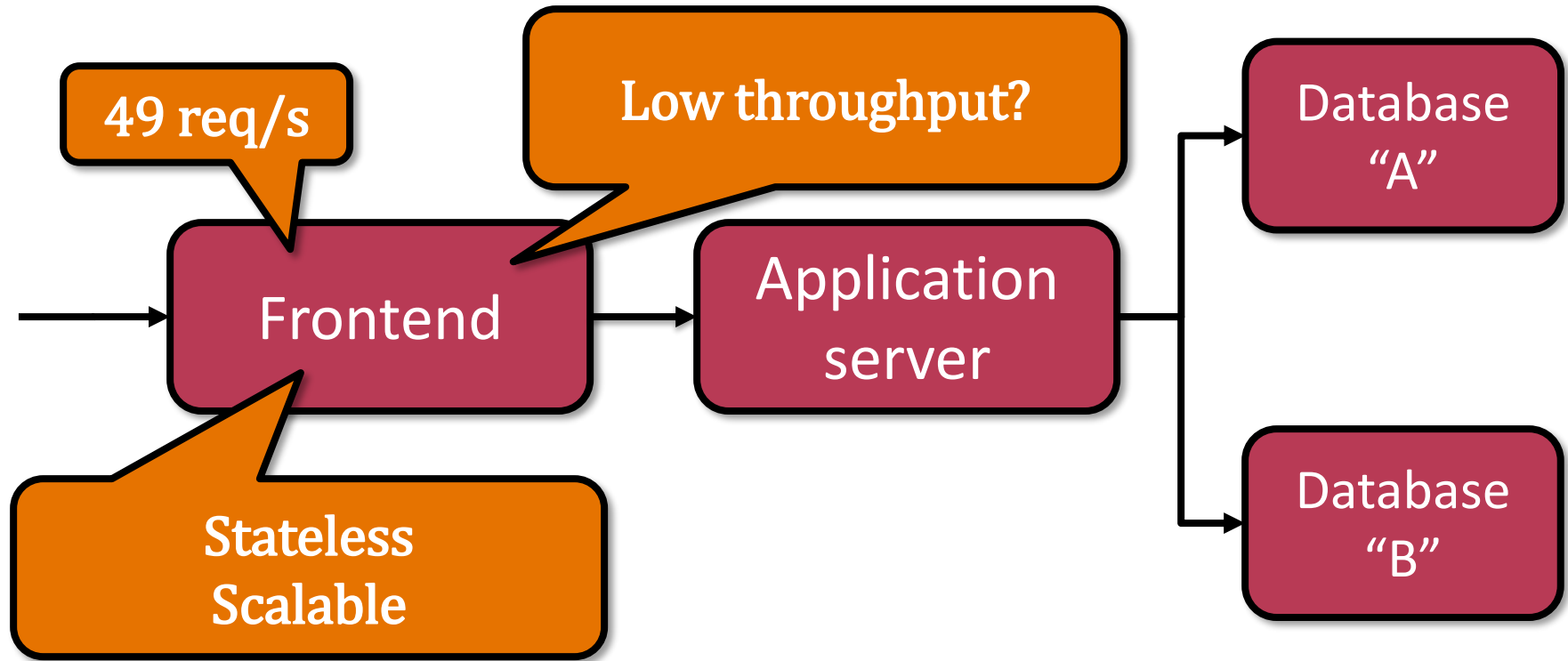
- <http://web.mit.edu/sgraves/www/papers/Little's%20Law-Published.pdf>
 - E.g.: How long do the wine bottles stay in the cellar?
 - The cellar is filled up to $\frac{2}{3}$ in average. (~160 bottles)
 - We bought 8 bottles per month in the last one year.
 - According to Little's law, the bottles stay in average $T=N/X$, that is $160/8=20$ months in the cellar.

Performance of 3-tier Architecture



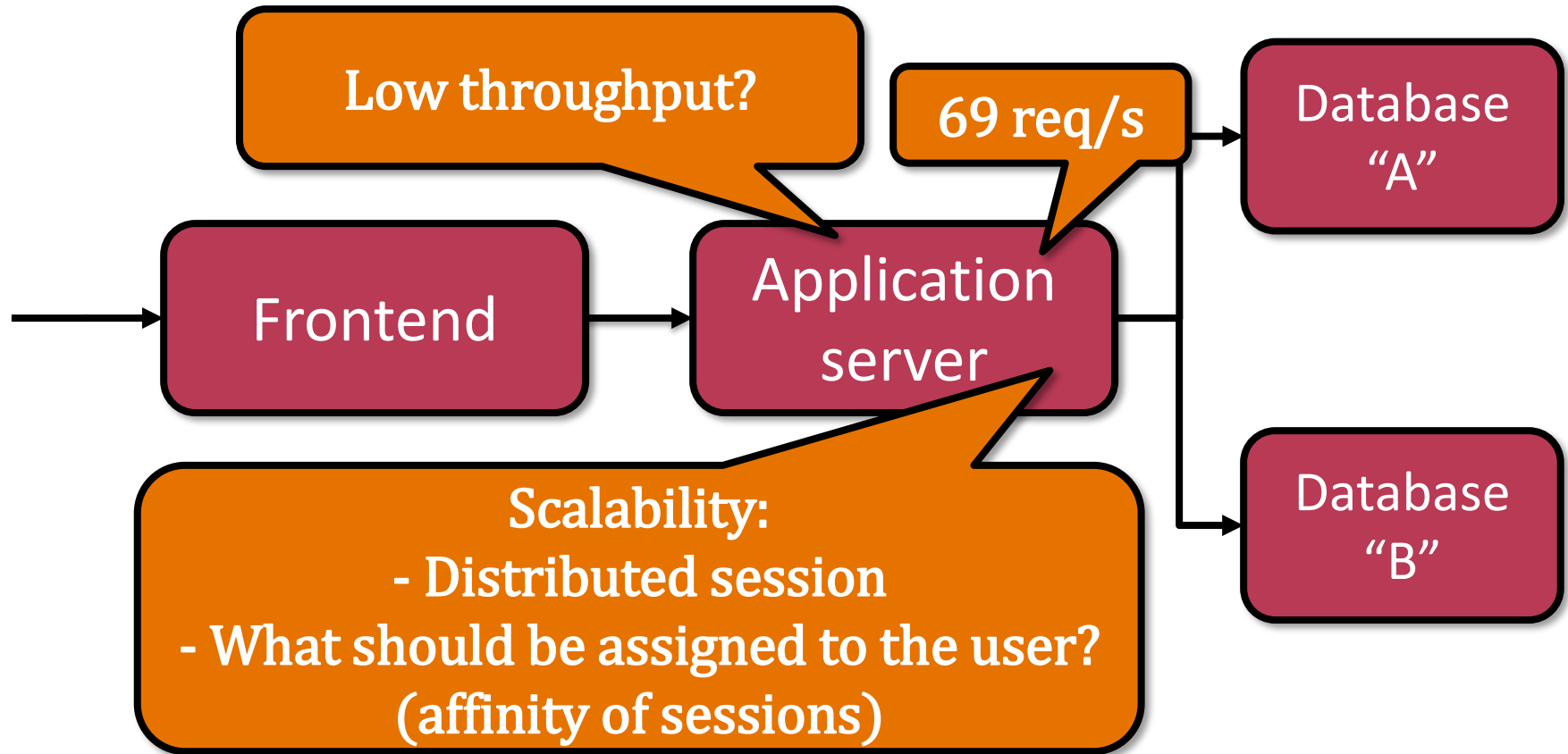
These metrics indicate the incoming load of the complete system! For instance „Database A” becomes the bottleneck if 319 requests arrive to the system each second.

Performance of 3-tier Architecture



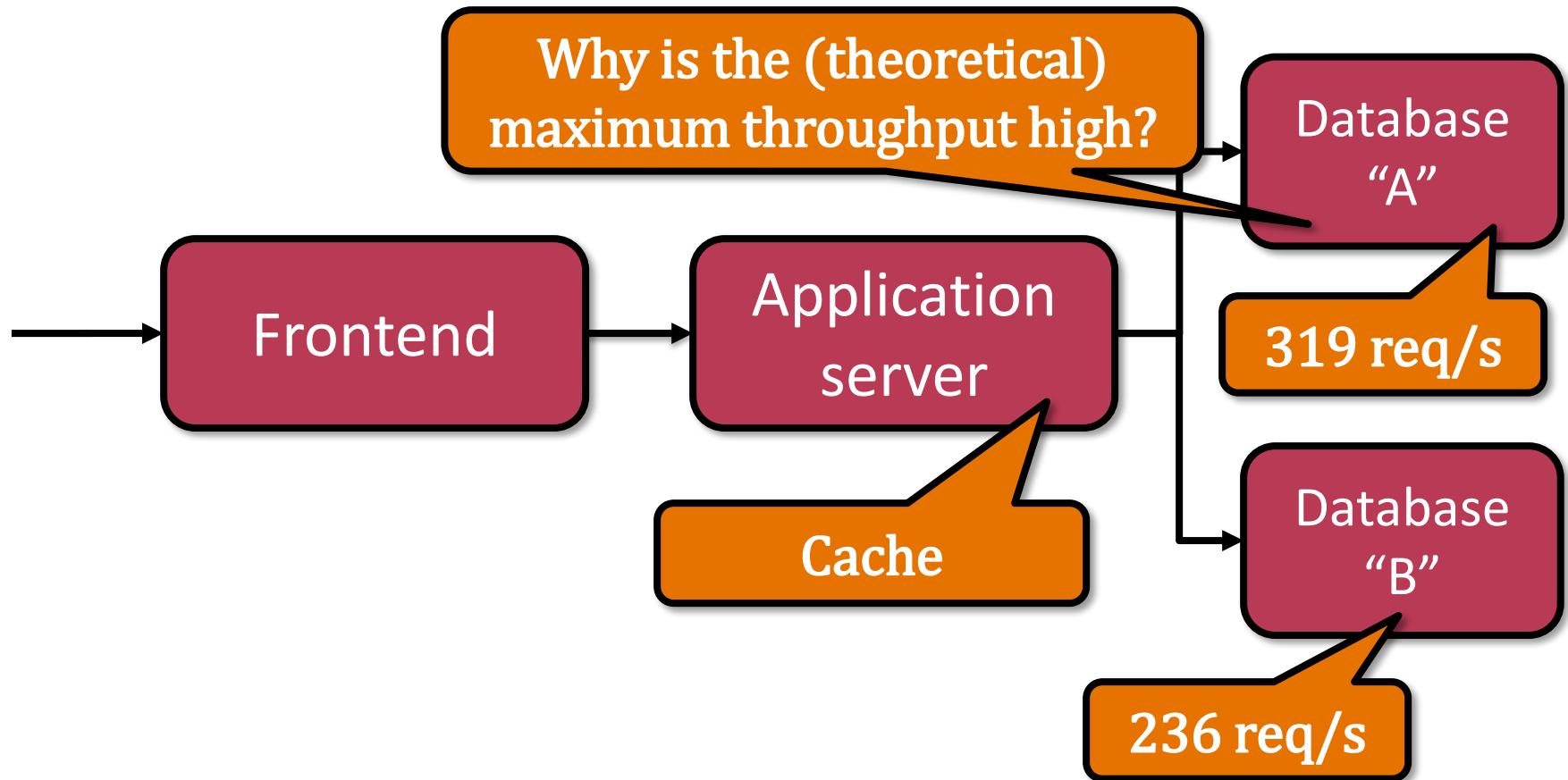
These metrics indicate the incoming load of the complete system!
For instance „Database A” becomes the bottleneck if 319 requests arrive to the system each second.

Performance of 3-tier Architecture



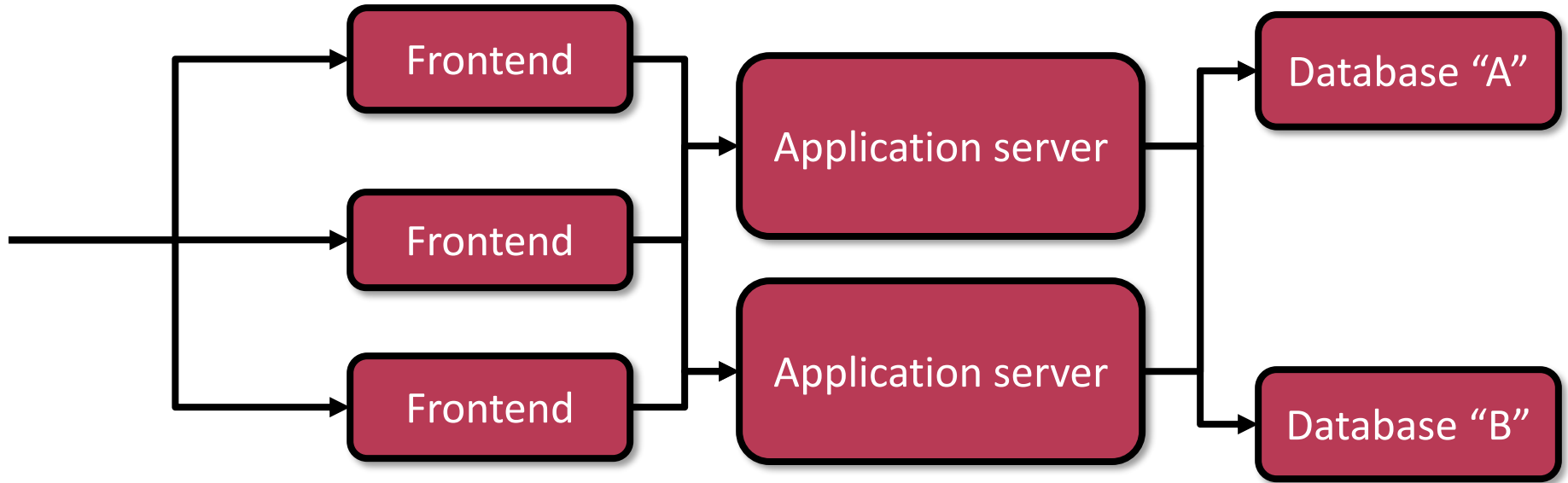
These metrics indicate the incoming load of the complete system!
For instance „Database A” becomes the bottleneck if 319 requests arrive to the system each second.

Performance of 3-tier Architecture



These metrics indicate the incoming load of the complete system! For instance „Database A” becomes the bottleneck if 319 requests arrive to the system each second.

3-tier Architecture in Reality



(Example: technological background for the interested)

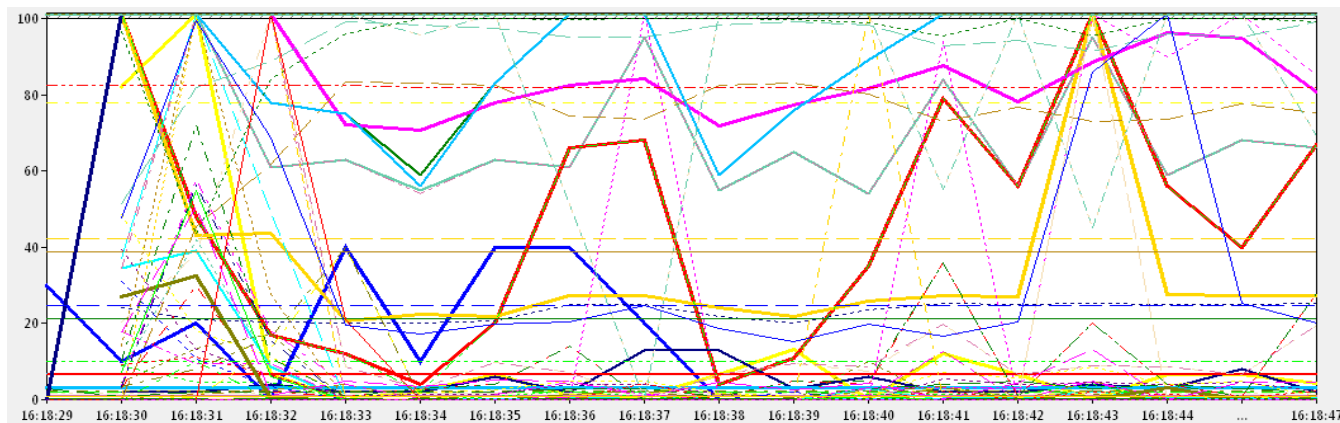
<http://www.projectclearwater.org/wp-content/uploads/2013/05/Clearwater-Deployment-Sizing-10-Apr-13.xlsx>

<http://www.projectclearwater.org/technical/clearwater-performance/>

What to Measure? / What is Important?

- Metrics “in small”

- E.g. Task manager, Resource monitor, the same on server-side...

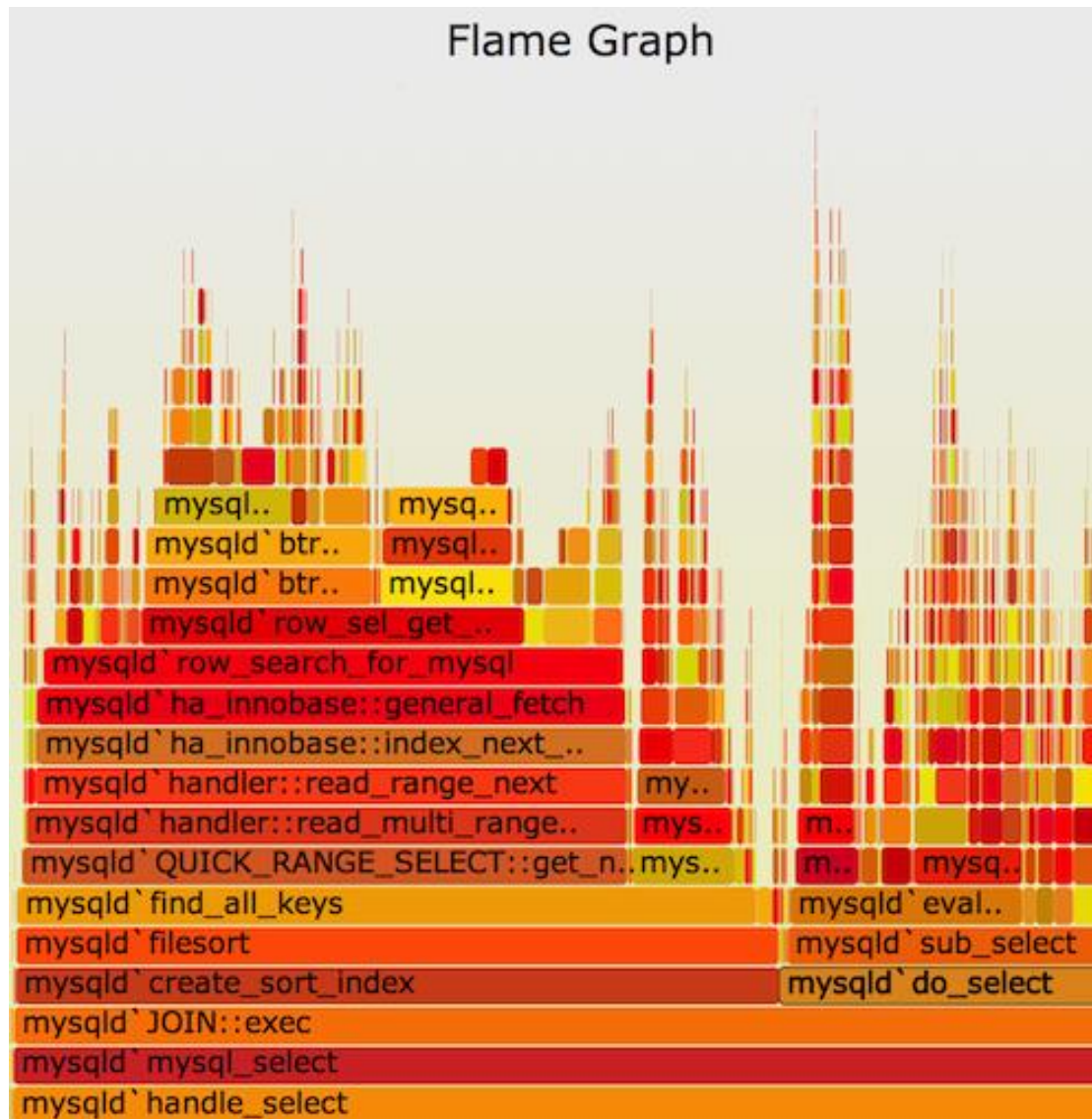


- Metrics “in big”

- E.g. virtualized systems

- Which metrics are interesting?

E.g.: What Takes so Much to Compute?

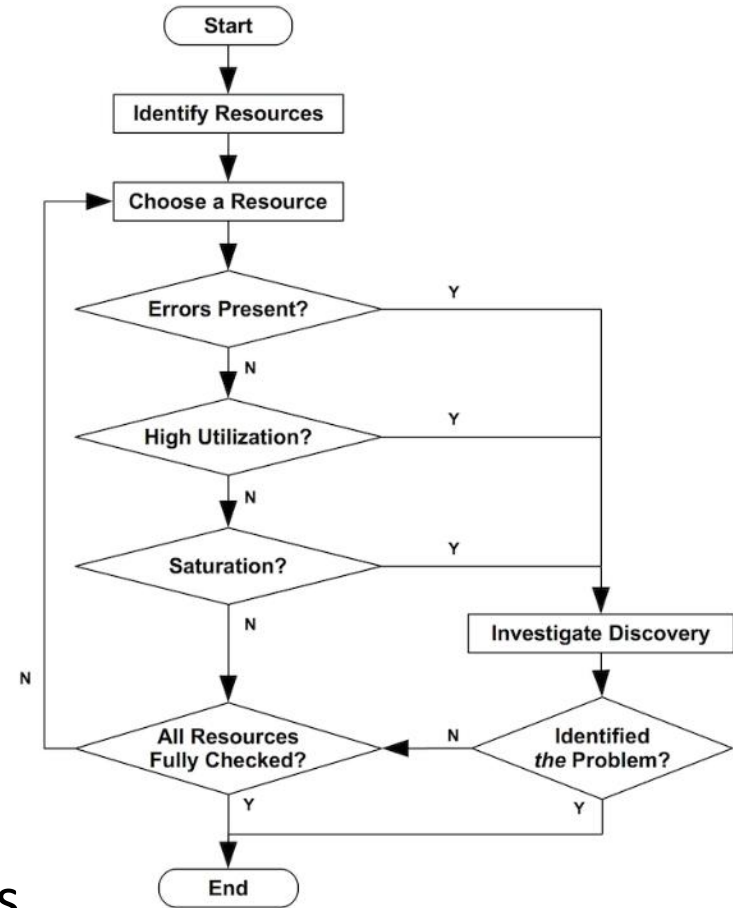


<http://www.brendangregg.com/flamegraphs.html>

Performance Modelling in the Practice

■ USE Method (from Brendan Gregg)

- Check **U**tilization, **S**aturation and **E**rrors for each resource
- Resources:
 - HW components
 - SW components
 - Communication connections
 - (from the structural model)
- How can USE be measured for the individual resources?
 - Saturation → long waiting queues



Where Do We Approximate?

- In practice the values are difficult to measure
 - (e.g. response time fluctuation, ...)
- Applications compete
 - ($2 * \lambda \neq \lambda + \lambda$)
- Decision between resources
 - Load balancer is also critical
 - E.g. requests of the same user to the same server
- We ignored the actual order/pattern of arrival
 - Advantage of Little's law
- Execution of a task may be data-dependent
- Structure/parameters of the system may change



“Watch my hands, because ...”
(picture: wikipedia)

Where Do We Approximate?

- In practice the values are difficult to measure

Everything is „lies and deception” only.

- (e.g., λ , X , ...)

- Applications compete

- ($2 * \lambda \neq \lambda + \lambda$)

- Decision between ...

→ Load balance

→ E.g. Request

- We ignored the ...

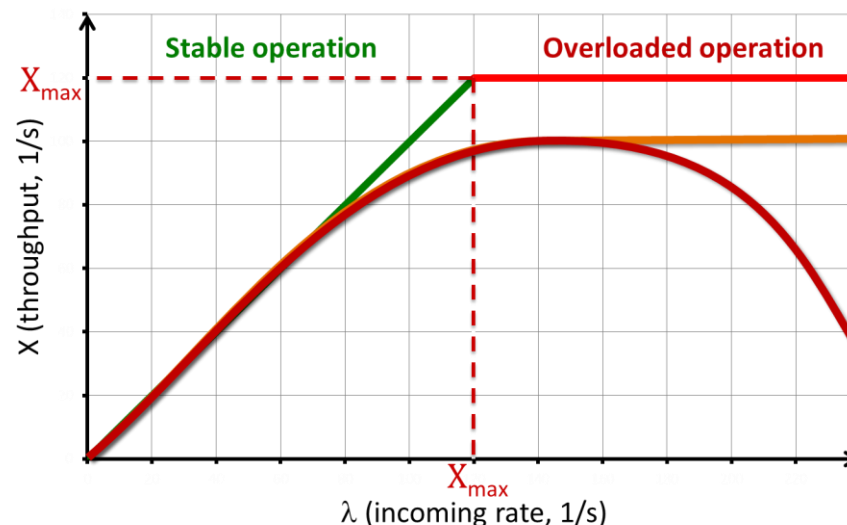
- Advantage of ...

- Execution of ...

- Structure/parameters of the system may change



...ands, because ...”
(media)



Visitation number

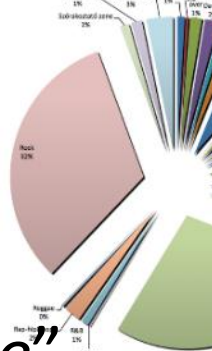
Little's law

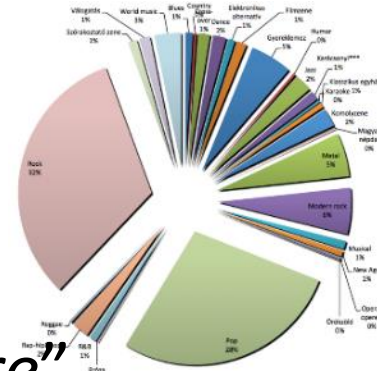
Zip's law

Changes in Workload

LOAD MODELS: ZIPF'S LAW

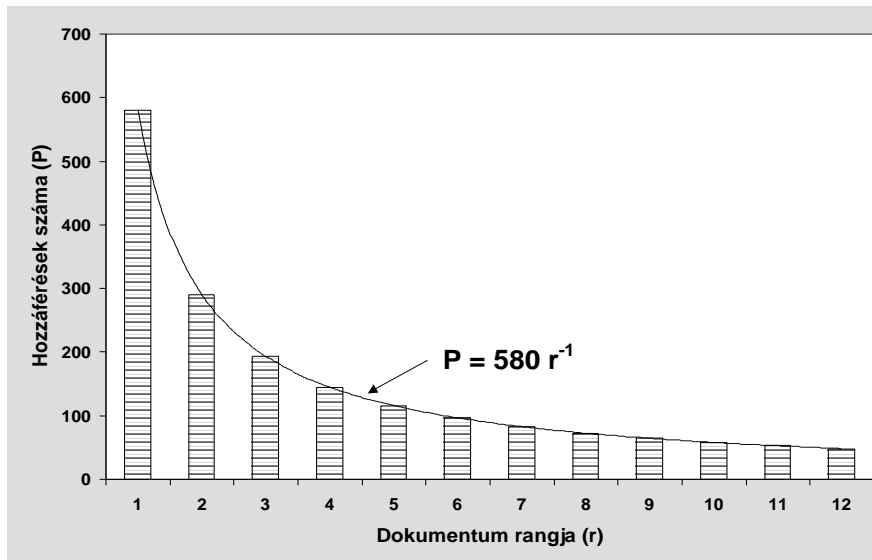
What is the Content of the Requests?

- Up to now: each requests are alike
 - “I need the details of a book”
 - Actually: requests have content
 - “I need the details of *Foundation and Empire*”
 - See Pareto principle (80% – 20%)
 - Majority of the requests concerns minority of data
 - Essential, because...
 - Has technical effects
 - Cache, pool size, static storage, ...
 - Concerns the system model
 - Special handling of frequent requests
- 



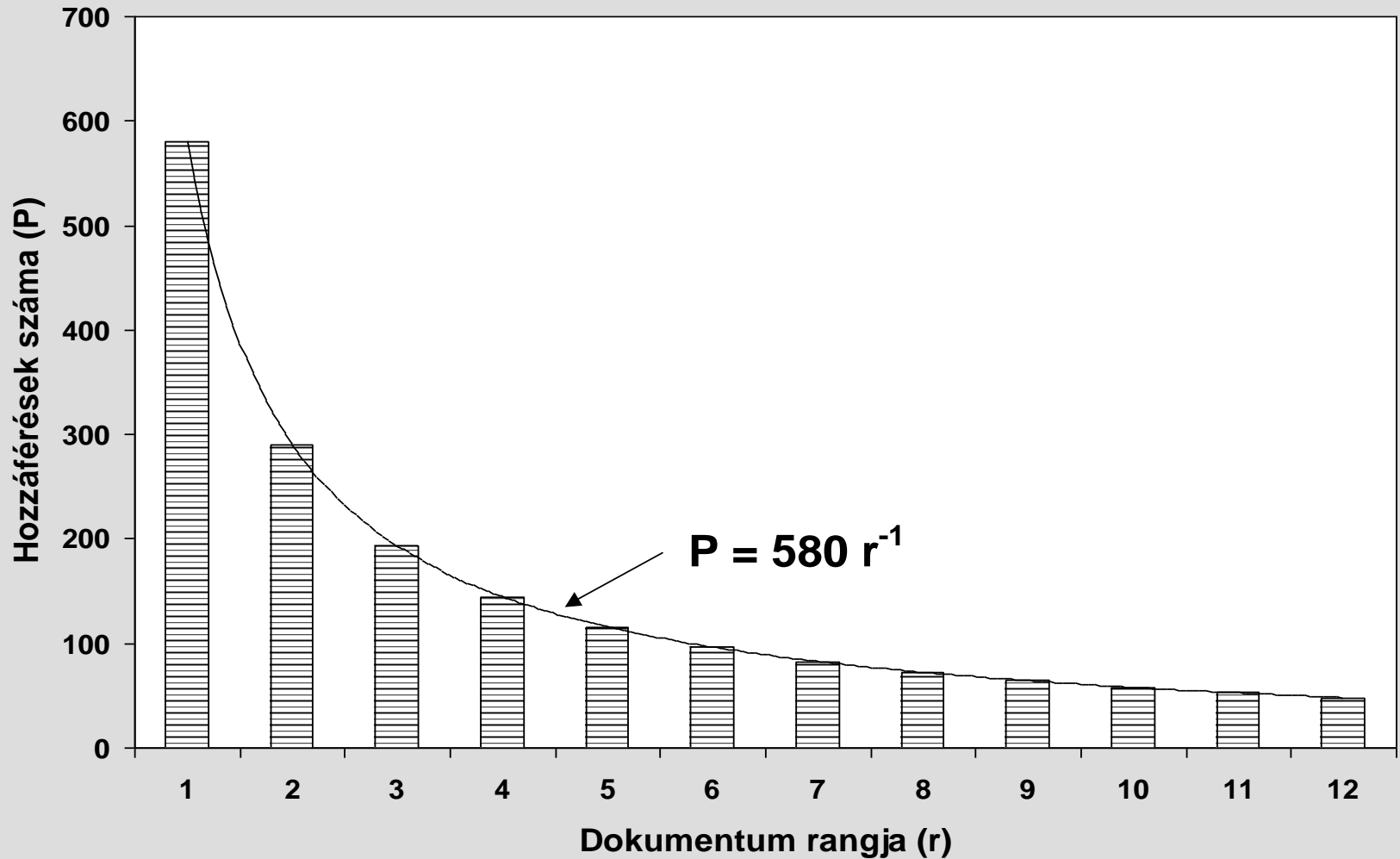
Zipf's Law

- Originally: number and frequency of words in *corpora* shows a characteristic distribution
 - True for not only language texts



George Kingsley Zipf
(1902–1950)
US American linguist
and philologist

Zipf's Law – Example



Zipf's Law - Examples

- Hit lists
- Population of cities by their ranks
- Characteristics of internet traffic
- Popularity of websites' subpages
- Evolution of open source systems

Zipf's Law - Formula

$$R_i \sim \frac{1}{i^\alpha}$$

$$f \sim \frac{1}{p}$$

- R_i – is the incidence of the i^{th} word
- α – a value characteristic of the corpus
 - close to 1
- Simplified ($\alpha = 1$):
 - f - frequency
 - p – popularity: rank of the text (decreasing order)

Zipf's Law – Example: Web Documents

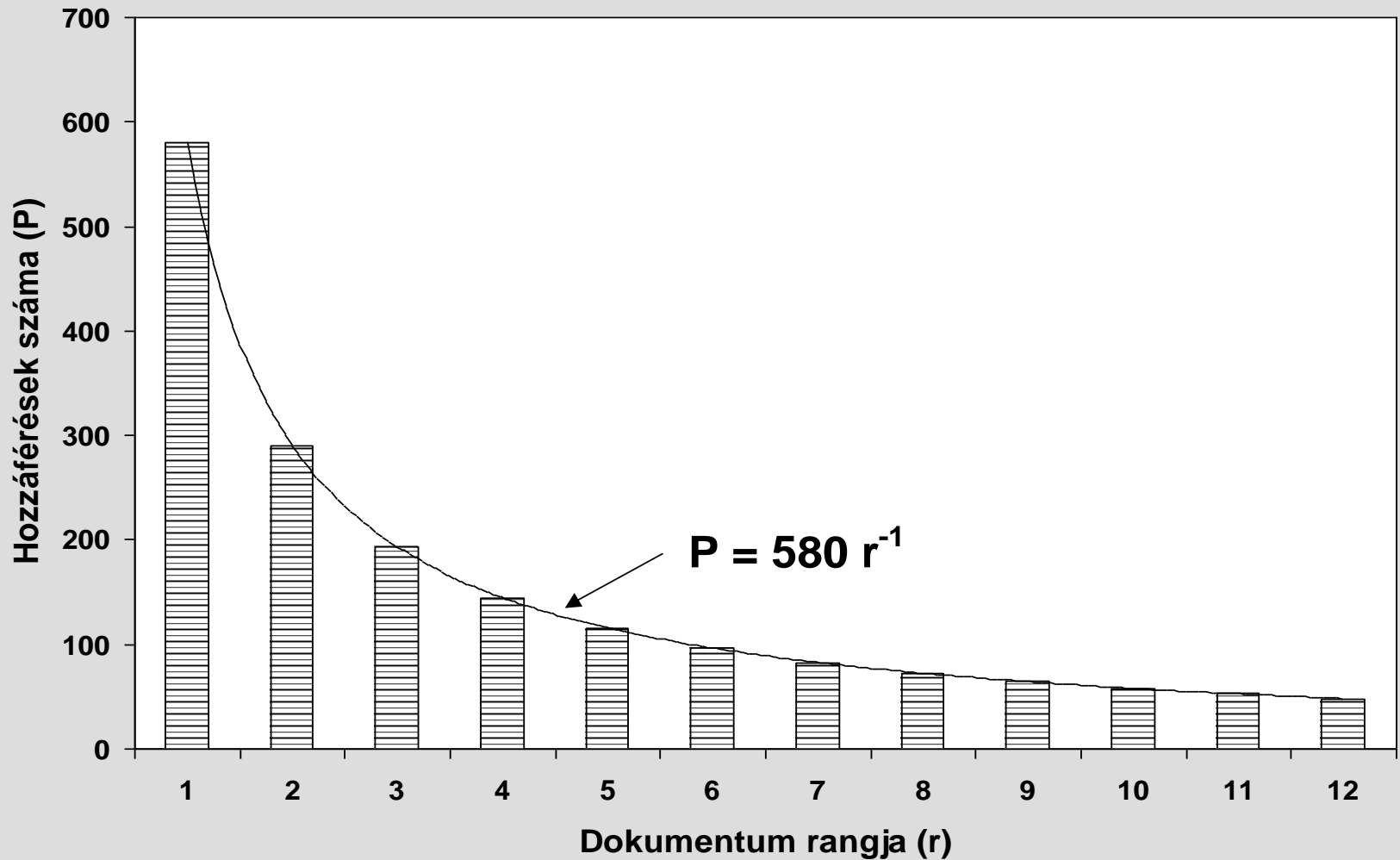
$$P = \frac{k}{r}$$

- P – references (hits)
- r – rank (1 = most frequent)
- k – positive constant

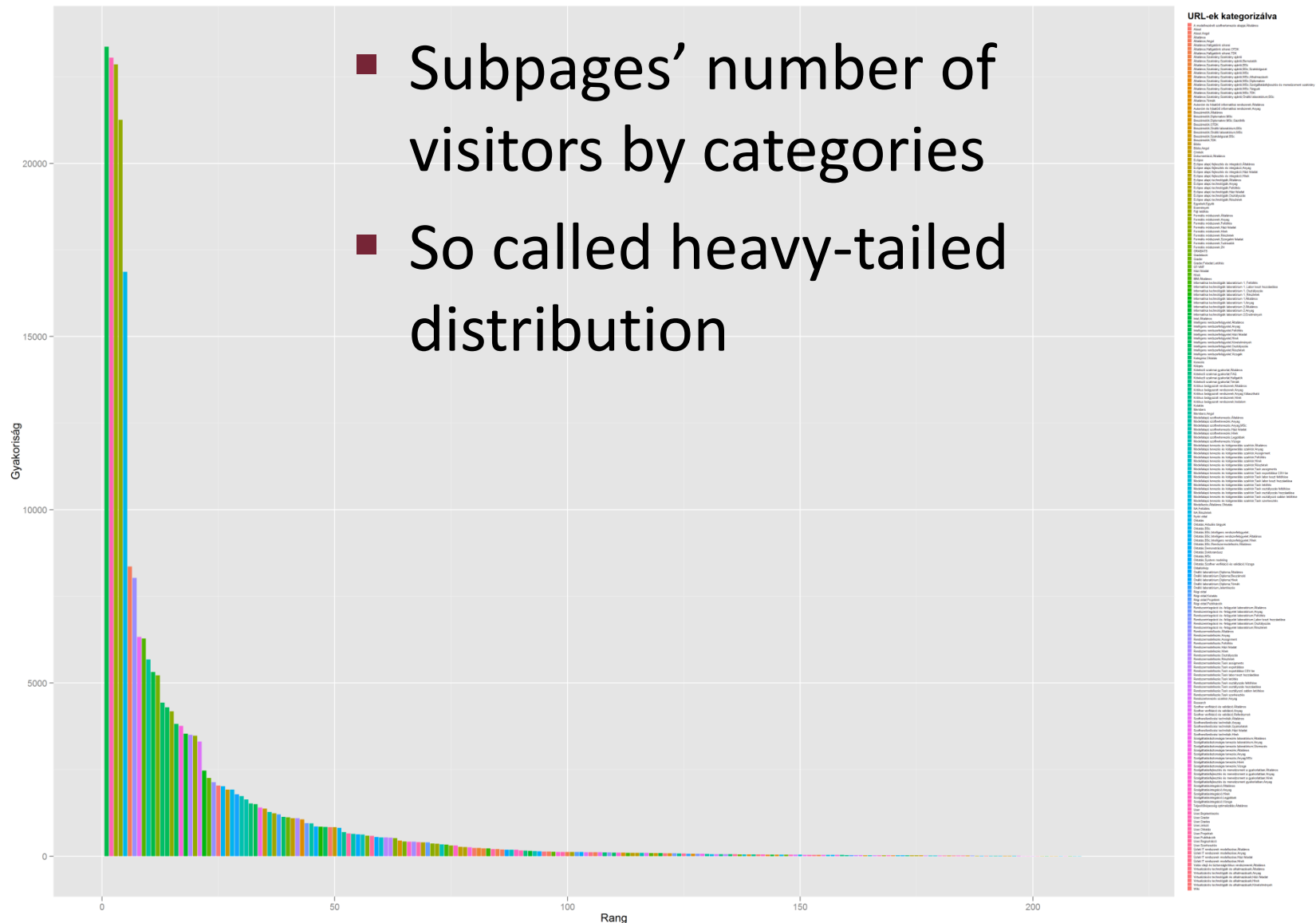
For more information see:

<http://www.hpl.hp.com/research/idl/papers/ranking/adamicglottometrics.pdf>

Zipf's Law – Example

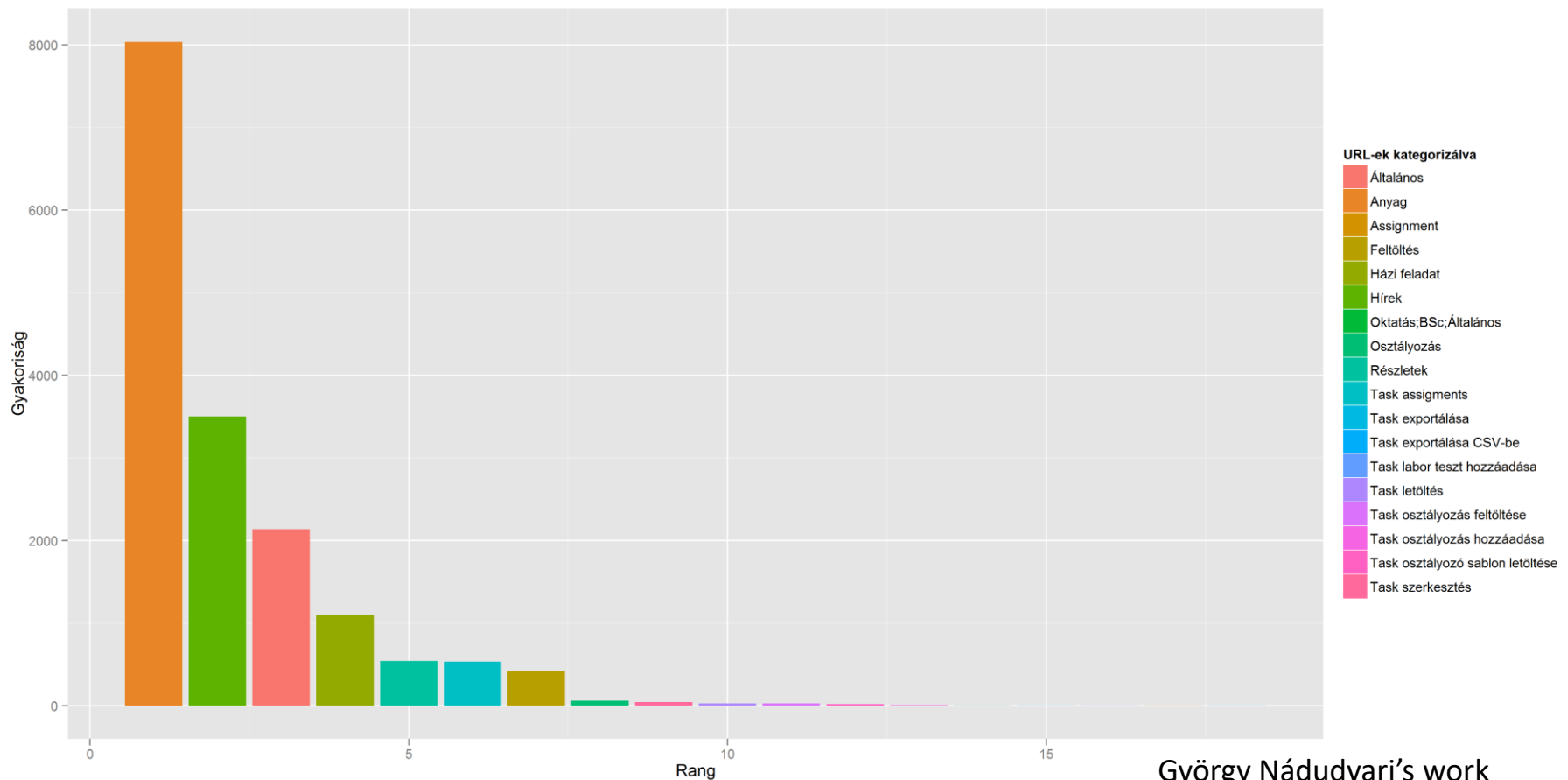


Zipf – Example: Website of our Group



Zipf – Example: Website of our Group

- Visitors of the webpages of the System Modelling course



György Nádudvari's work

Visitation number

Little's law

Zip's law

Changes in Workload

CHANGES IN THE WORKLOAD

What kind of workload?

- Up to now:
 - We calculated with average values
 - Regarded the system's behaviour depending on the *load (intensity)*
 - But: In reality the increase of the load is not necessarily predictable
- In reality
 - The behaviour of the system *changes over time*
 - This has technological effects
 - Switching between tasks, resource reservation, etc. (see: Operating systems)

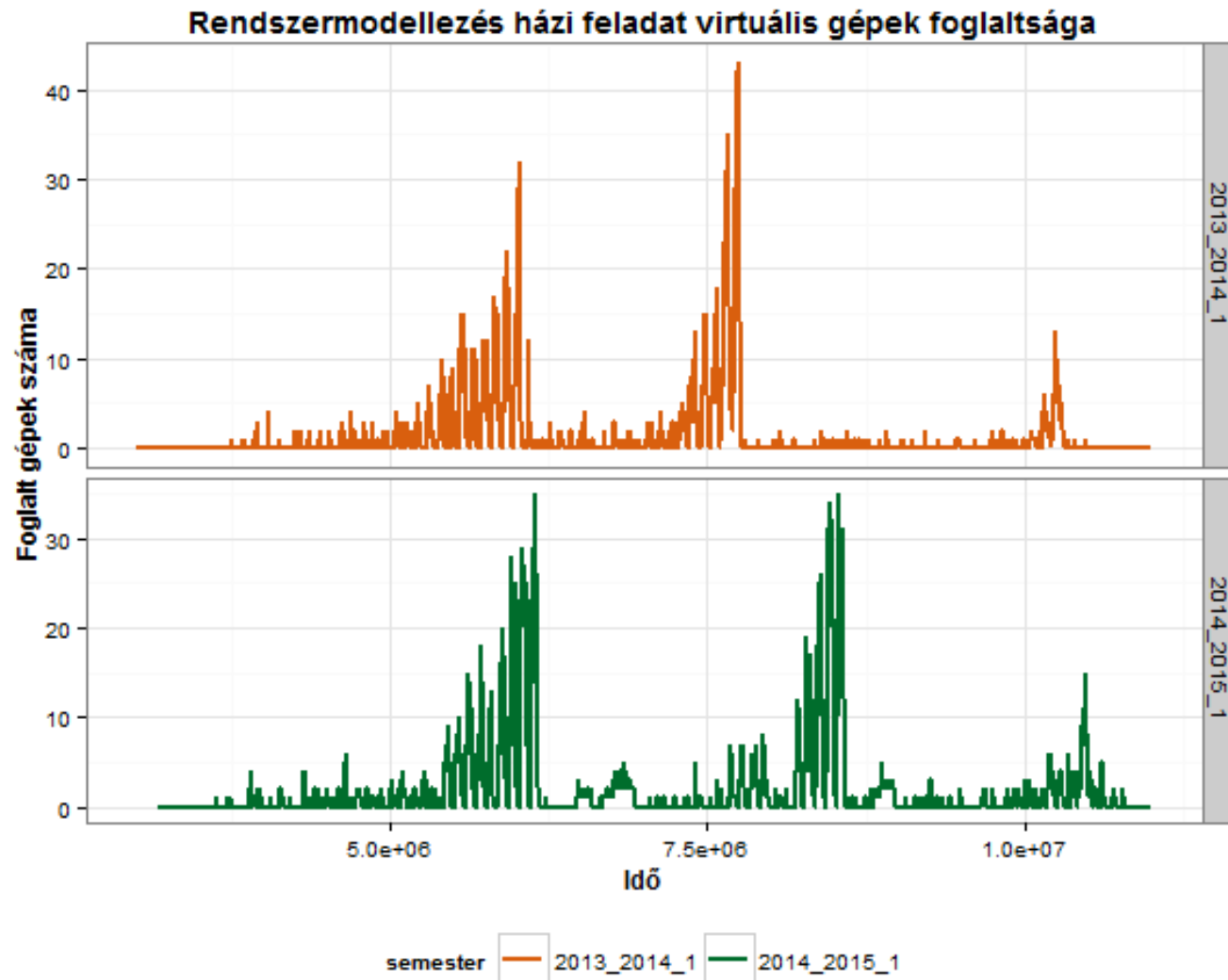
Changes in the Workload – Example

- Dimensioning a systems for producing the (at that time) new identity cards
 - It is predictable how many new cards will be applied for in a year. (expirations, next age group)
 - It is predictable how many hours there are in a year.
 - We have the avg. arrival rate of the applications [$card/h$]

Can it be used for dimensioning the system?

- Consider two different hours
 1. the 24th December 10-11PM
 2. the 15th June 4-5PM
(End of working day shortly before the main summer holiday time)

System Modelling (7th semest.) – in the cloud



System Modelling (7th semest.) – in the cloud

