Cloud Computing Applications and Services

(Aplicações e Serviços de Computação em Nuvem)

Benchmarking

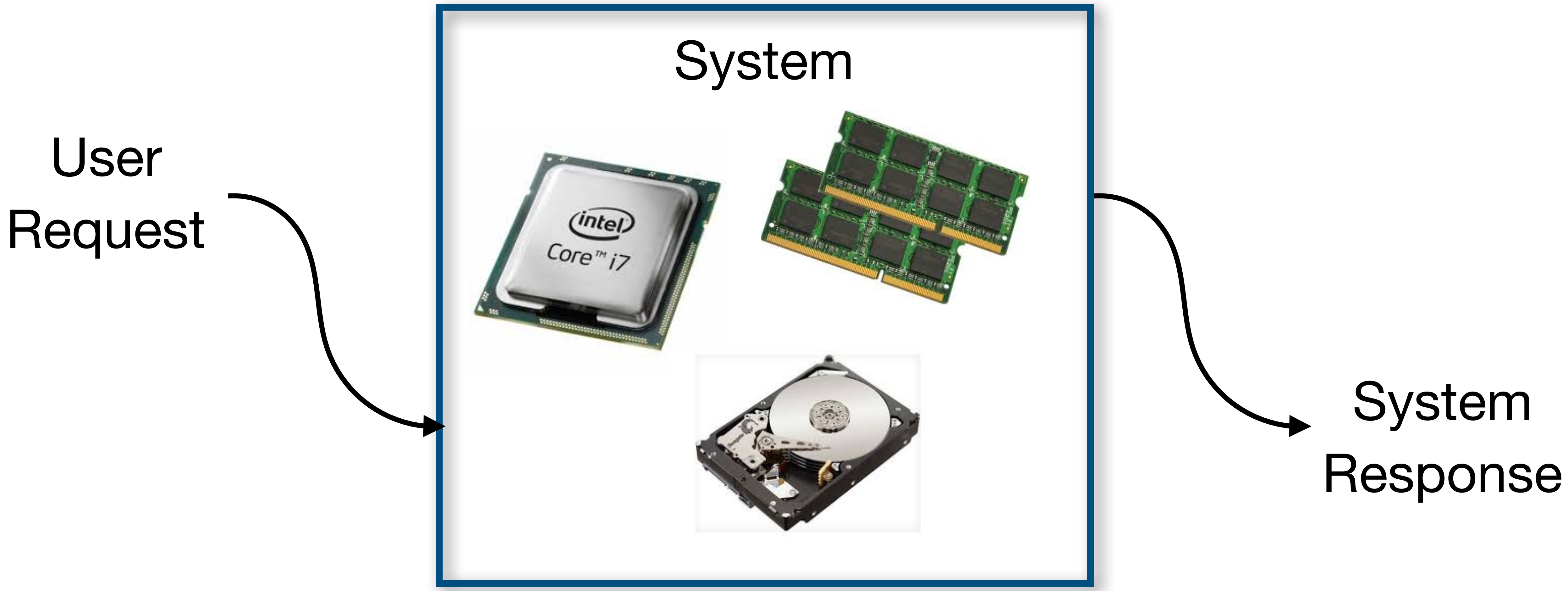
University of Minho

2024-2025

Software Systems

What do these do?

- Software systems can assume different forms (e.g., operating system, file system, database, web server, …)
- Systems perform useful work for users by receiving **requests**, handling these, and producing **responses**
- Systems use physical and logical resources with limited **capacity**
  - **Physical:** CPU, memory, disk, network, …
  - **Logical:** locks, caches, …



System Benchmarking

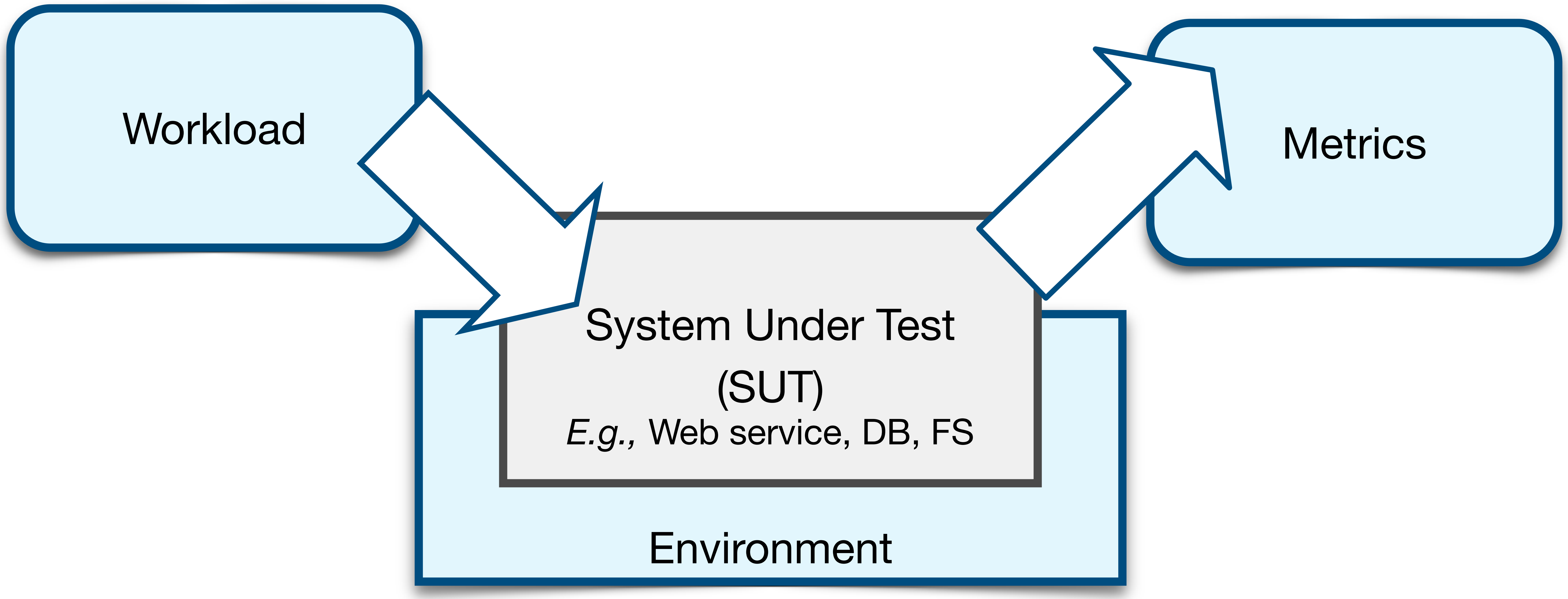
Why is it useful?

- How can one build efficient and reliable systems?
  - Follow good design and implementation guidelines
  - Review and validate the code and implementation (e.g., formal verification)
  - **Test** (i.e., **benchmark**) implementations!
- How can one know the system is using logical and physical resources efficiently?
  - **Benchmark** the system!
- How can one know that the environment where the system is running has enough resources?
  - **Benchmark** the system in that environment!

System Benchmarking

Ecosystem

- A Benchmarking ecosystem consists of
  - The **workload** (group of requests) being issued to the **System Under Test** (SUT)
  - The **environment** where the SUT is running
  - The **metrics** collected to measure the performance, efficiency, and/or reliability of the SUT



Workload

Traces vs synthetic workloads

- Testing systems in **production** may not be viable, or the best option
  - Imagine only knowing if your SUT works when deployed and serving users…
- One can use **traces of requests** from a real production setup instead
  - **Advantage:** requests extracted from **real workloads**
  - **Disadvantage: Hard to get (sometimes) and scale** (i.e., How can one scale a trace with 100 requests to millions of requests without losing realism?)
- Another option is to use **synthetic workloads**
  - Use a **subset of synthetically generated requests** (e.g., set of database queries, file system operations, web requests)
  - Generate parameters to **mimic different behaviors** (e.g., request type, size, parallelism)
    - Following a given distribution (e.g., sequential, uniform, zipf, Poisson …)

Environment

Hardware and software

- Knowing and setting up the right environment to run experiments is very important!
  - Testing your SUT in an unrealistic environment may lead to **wrong conclusions**
  - It is also important for others to be able to **reproduce** your results

- One must be able to **characterize the experimental environment**

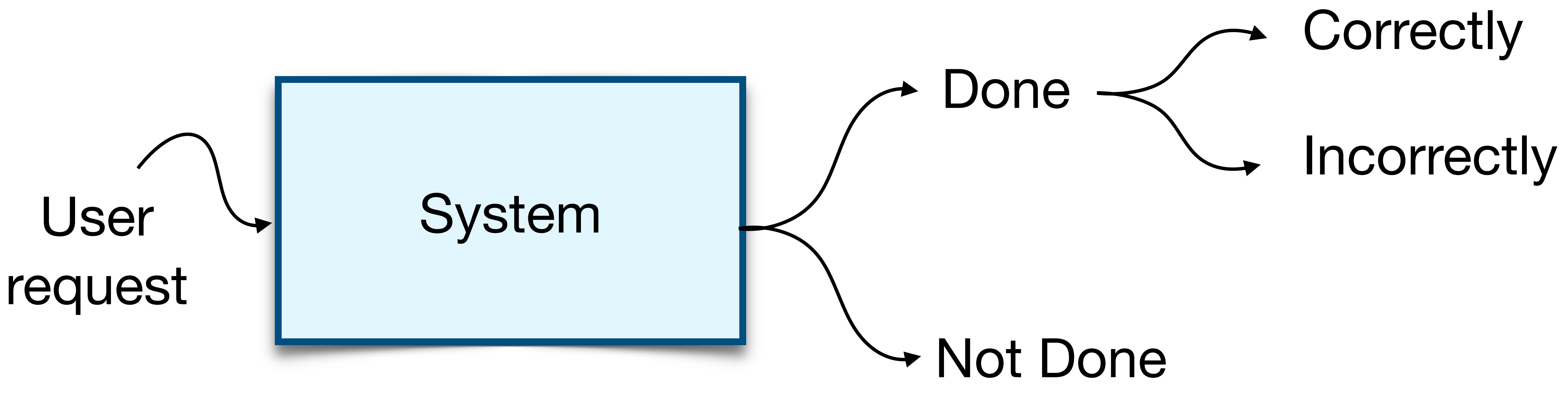
- **Hardware**
  - What CPU, RAM, GPU, and disk models are being used?
  - What hardware configurations are being used? (e.g., number of CPUs, amount of RAM, …)

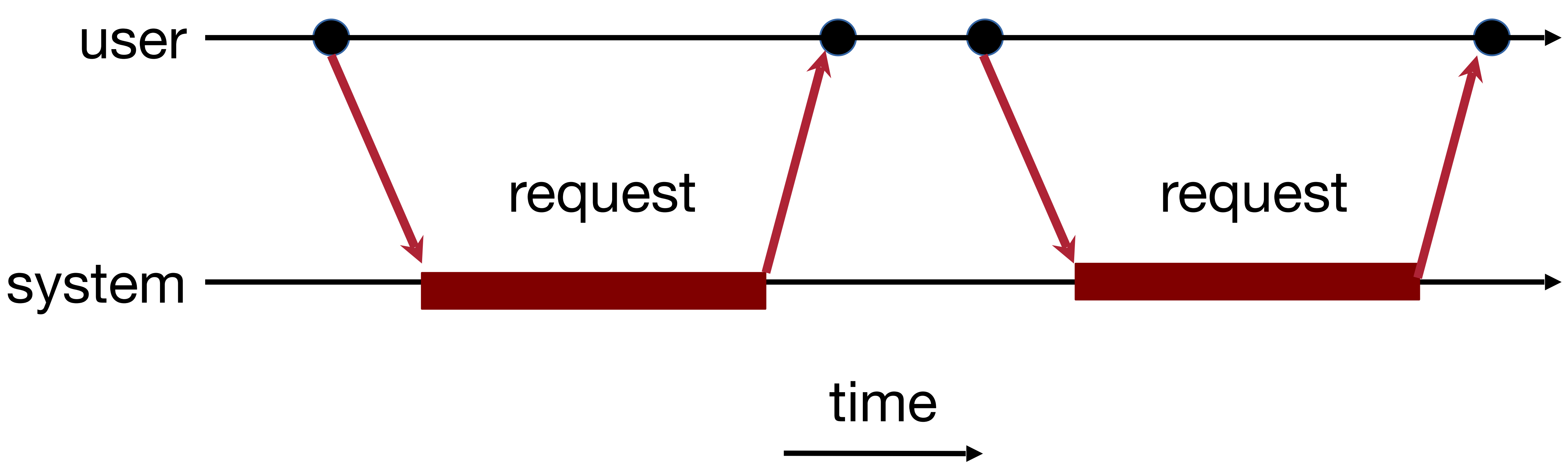
- **Software**
  - What operating system is being used? And the kernel version?
  - What about the libraries and corresponding versions?
  - And finally, what are the versions of the different SUT components?

Metrics

Let's start with performance

- The SUT will be serving **multiple user requests over time**
- The **response** varies
  - Some requests are served **correctly** or **incorrectly** (e.g., errors)
  - Other requests may **not be served**
    (e.g., the SUT rejects a request because it is too busy)

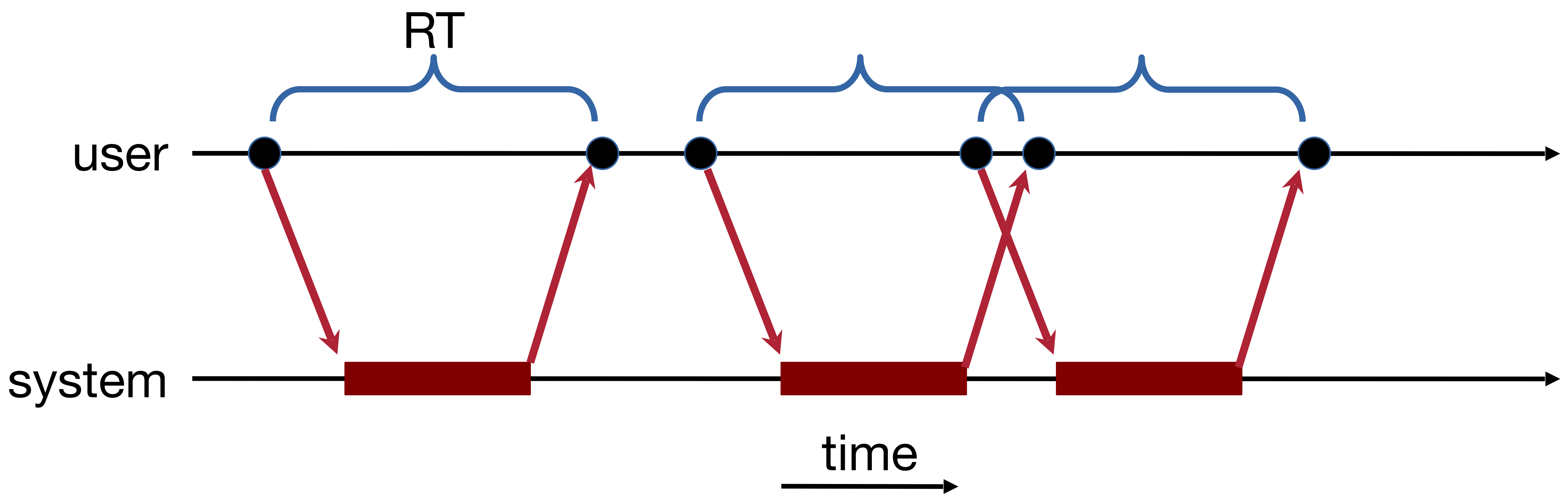


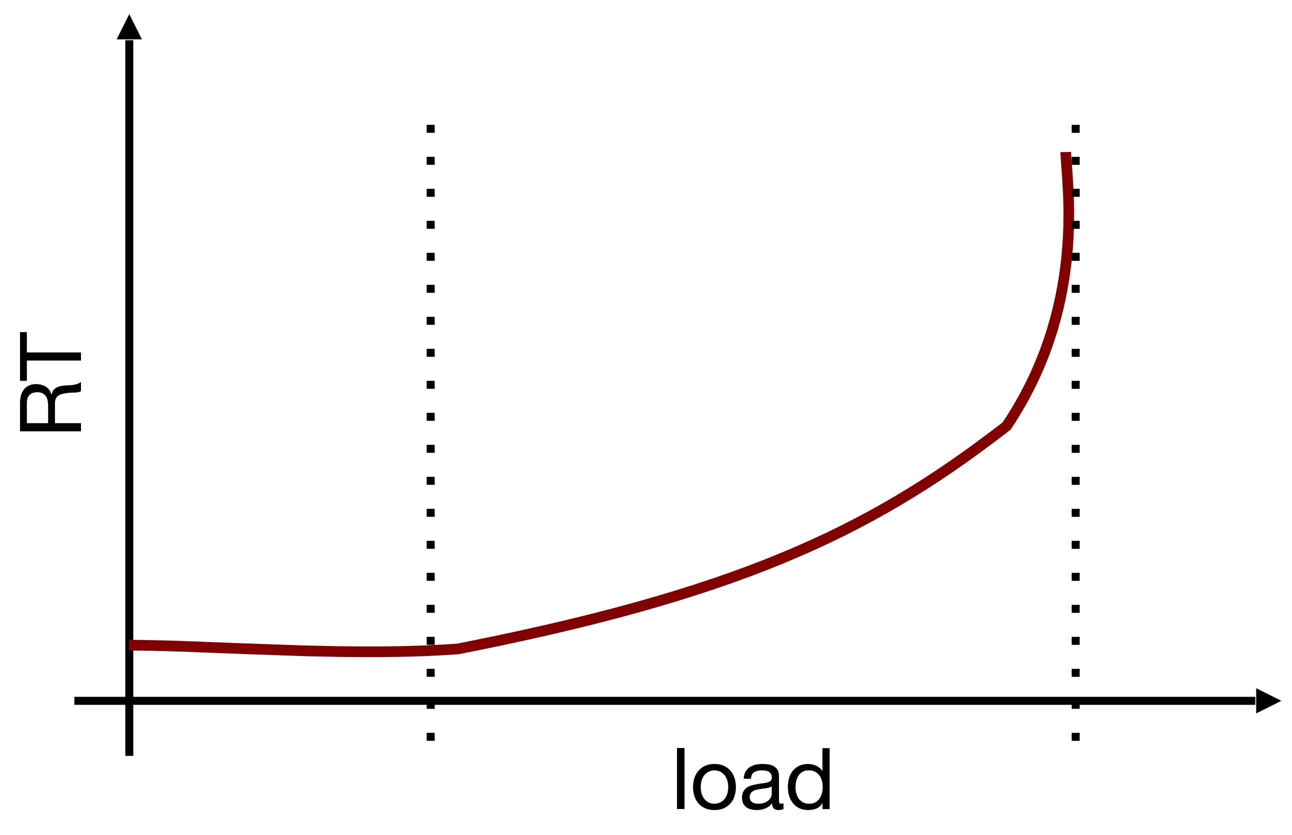


Performance

Metric: Response Time (RT)

- **RT:** The time interval between the user's request and the system's response
  - Sometimes also referred to as **latency**
- As the **load** in the system **increases**, the **RT** of requests tends also to **increase**

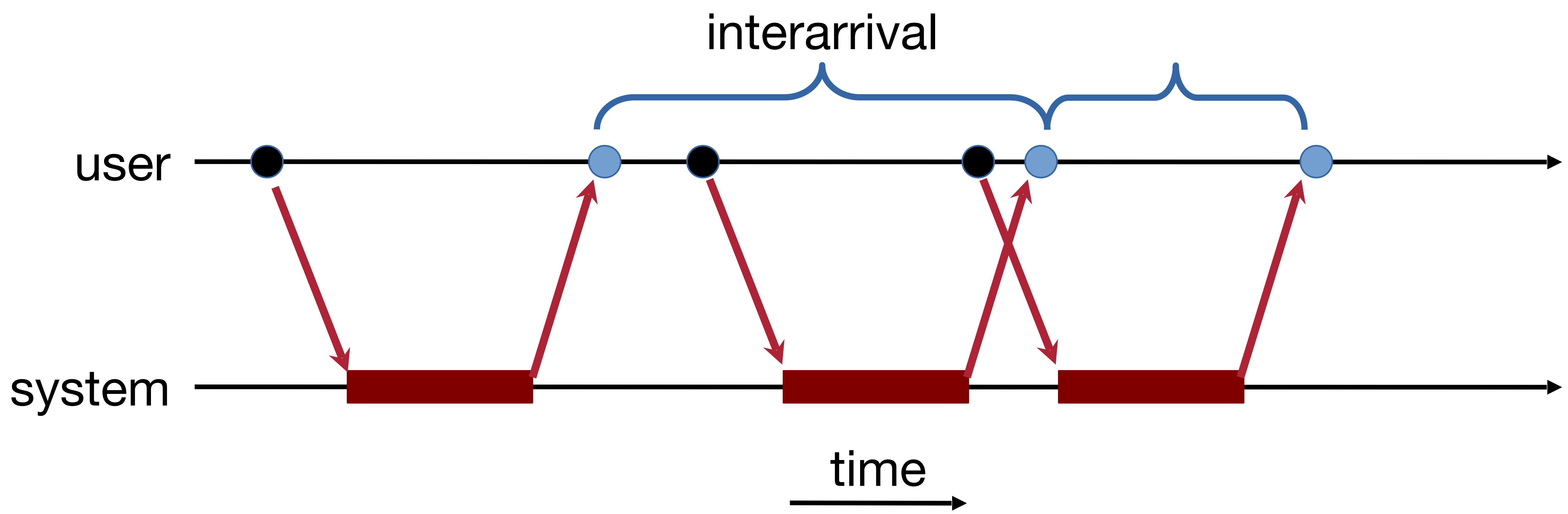


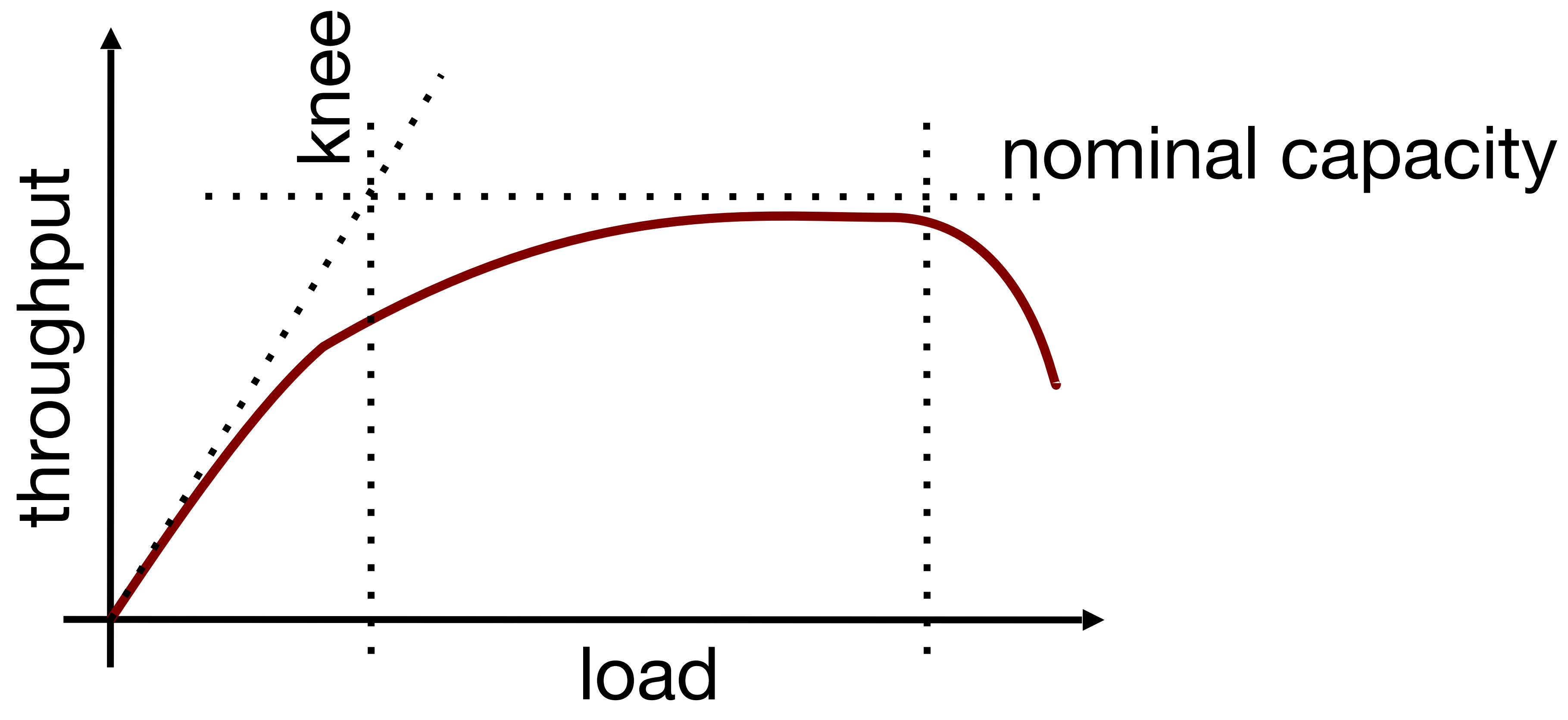


Performance

Metric: Throughput

- **Throughput:** rate at which user requests are serviced by the system (i.e., operations served per unit of time)
- As the **load** in the system **increases**, the **throughput** of requests tends to increase until a certain point, reaching the system's **nominal capacity**
  - When the system becomes **saturated**, the throughput starts decreasing

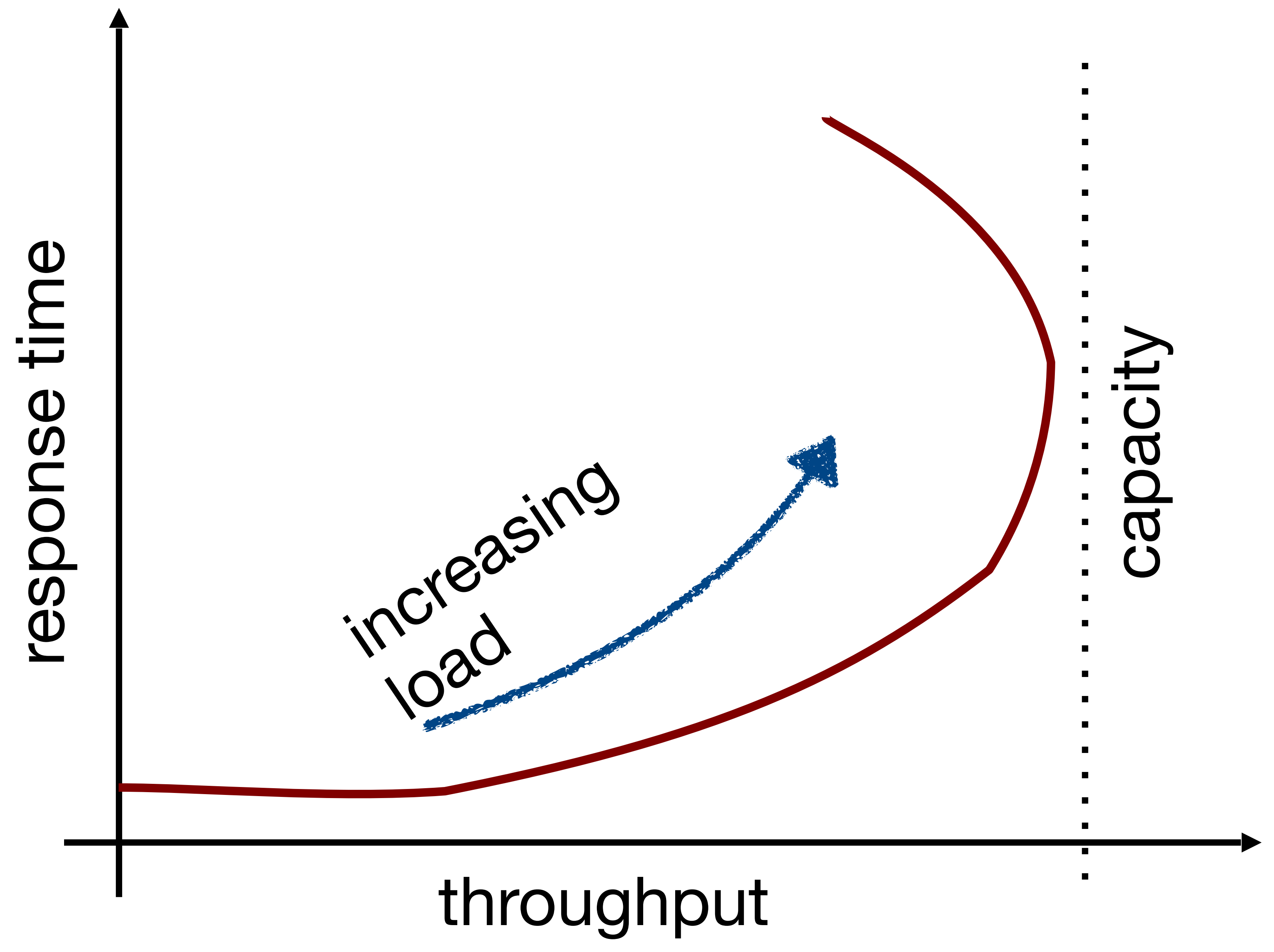




Response Time vs Throughput

Carefull!

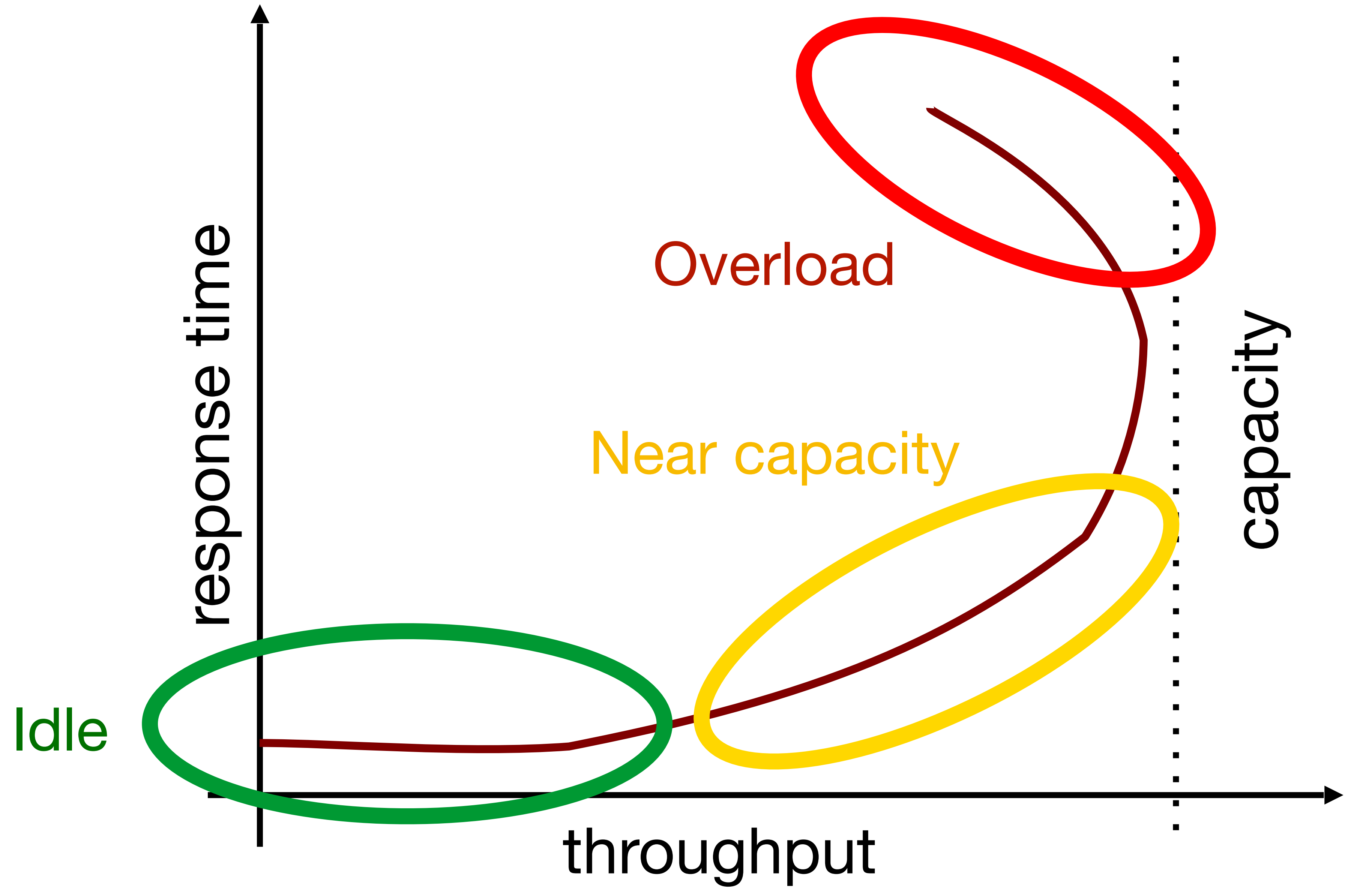
- A naive view (**RT = 1 / Throughput**) is false!
  - This is only true when the system is busy 100% of the time executing exactly 1 request!
- The relation between **RT** and **throughput** characterizes system performance



Response Time vs Throughput

Phases

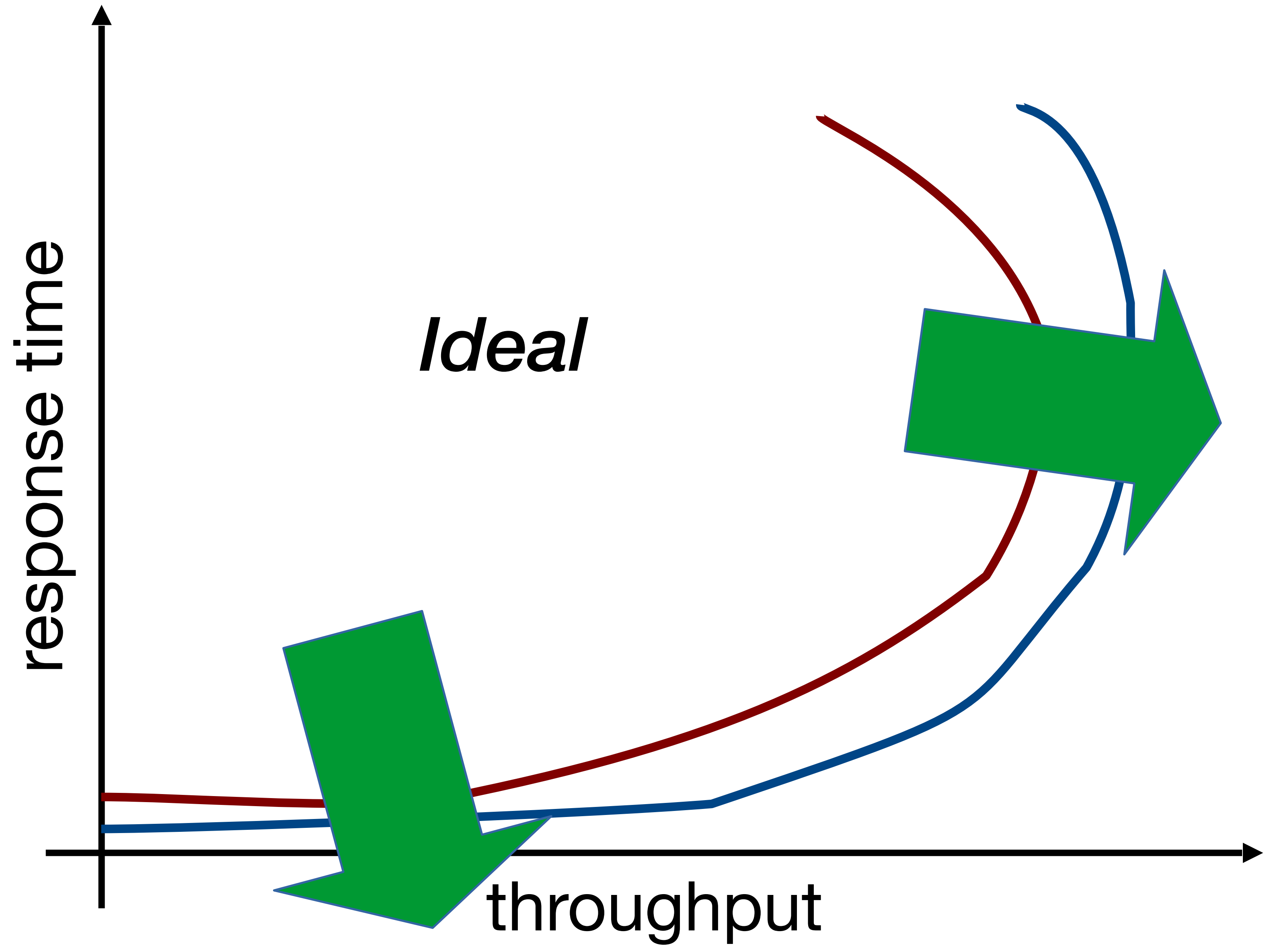
- In the figure below, one can observe **3 distinct phases** of the system
  - ***Idle***: requests are immediately handled as the system has spare capacity
  - **Near capacity:** requests are handled after a brief wait (throughput and RT increasing)
  - ***Overload***: resources become saturated (throughput decreases while RT increases)

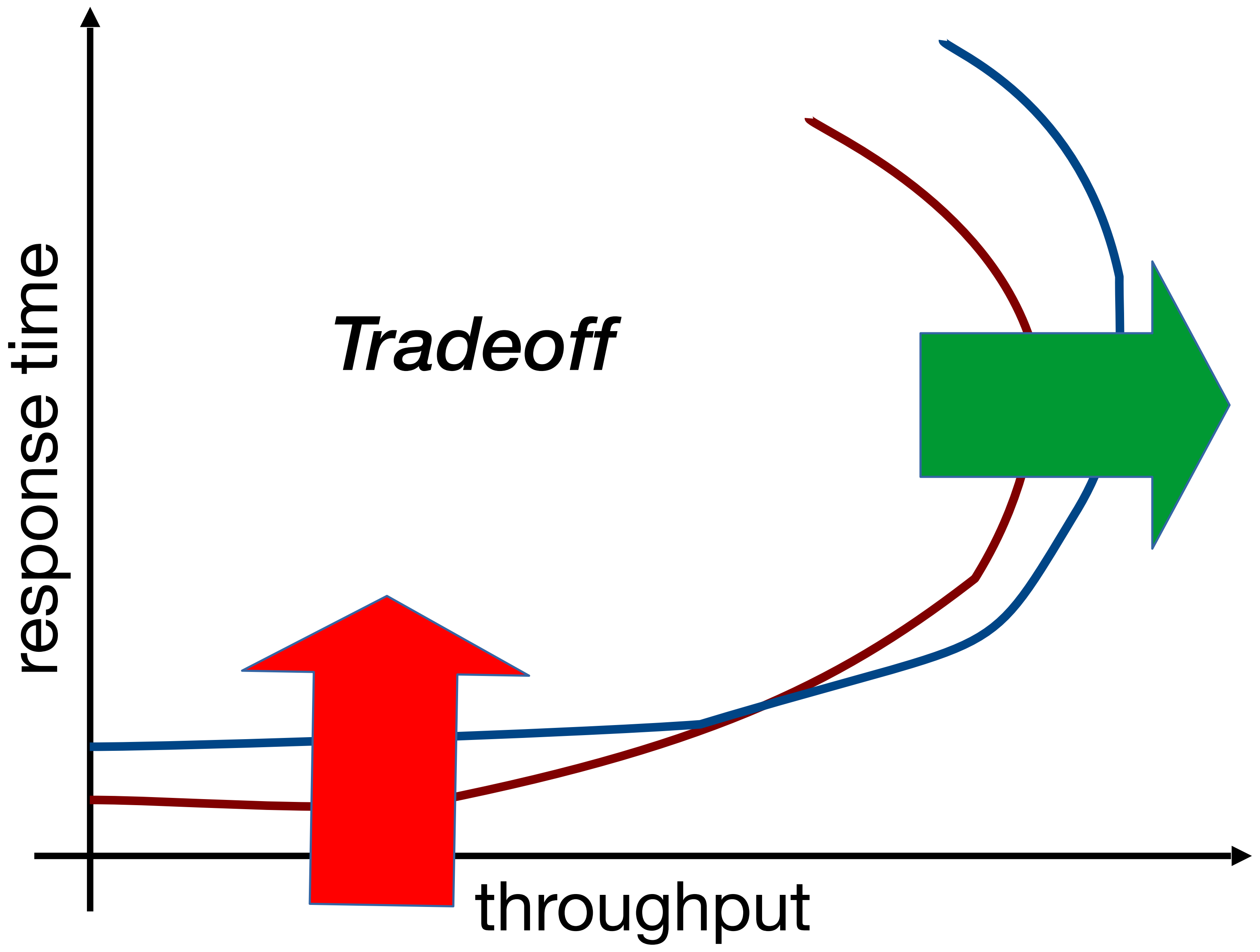


Response Time vs Throughput

Tradeoff

- Ideally, one would **optimize** a system to increase throughput and decrease RT
  - Hard to achieve!
- Most optimizations **trade** one metric for the other
  (e.g., batching requests usually improve throughput at the cost of RT)





More Metrics

Other than performance

- **Utilization** of resources
  (e.g., CPU, RAM, network, disk, energy)
- **Efficiency** of the system
  (e.g., the ratio between throughput and utilization, between throughput and energy consumption, …)
- **Reliability** of a system
  (e.g., number of errors, failures)
- **Availability** of a system
  (e.g., uptime, downtime)

Analysis

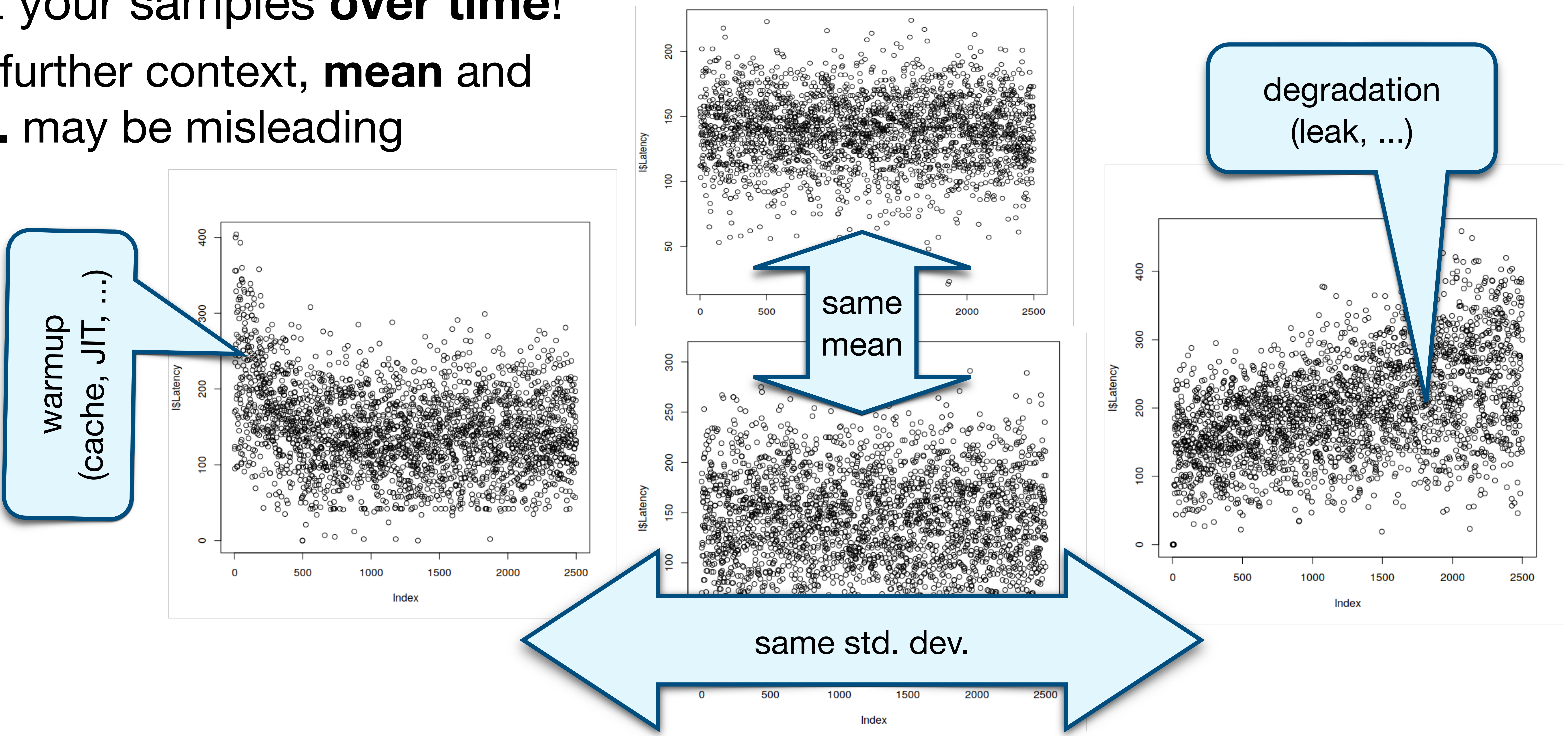
Of our metrics

- When running experiments, one must collect several **samples** for metrics of interest!
  - Repeat your experiments several times to identify **abnormal behavior** (outliers)
  - Remember that many workloads **may not be deterministic**!
- After collecting the samples, these must be **analyzed** and, ideally, **summarized**
  - Mean
  - Standard deviation (std. dev.)
  - Mode
  - Median
  - Percentiles
  - …

Sample Analysis

Versus time

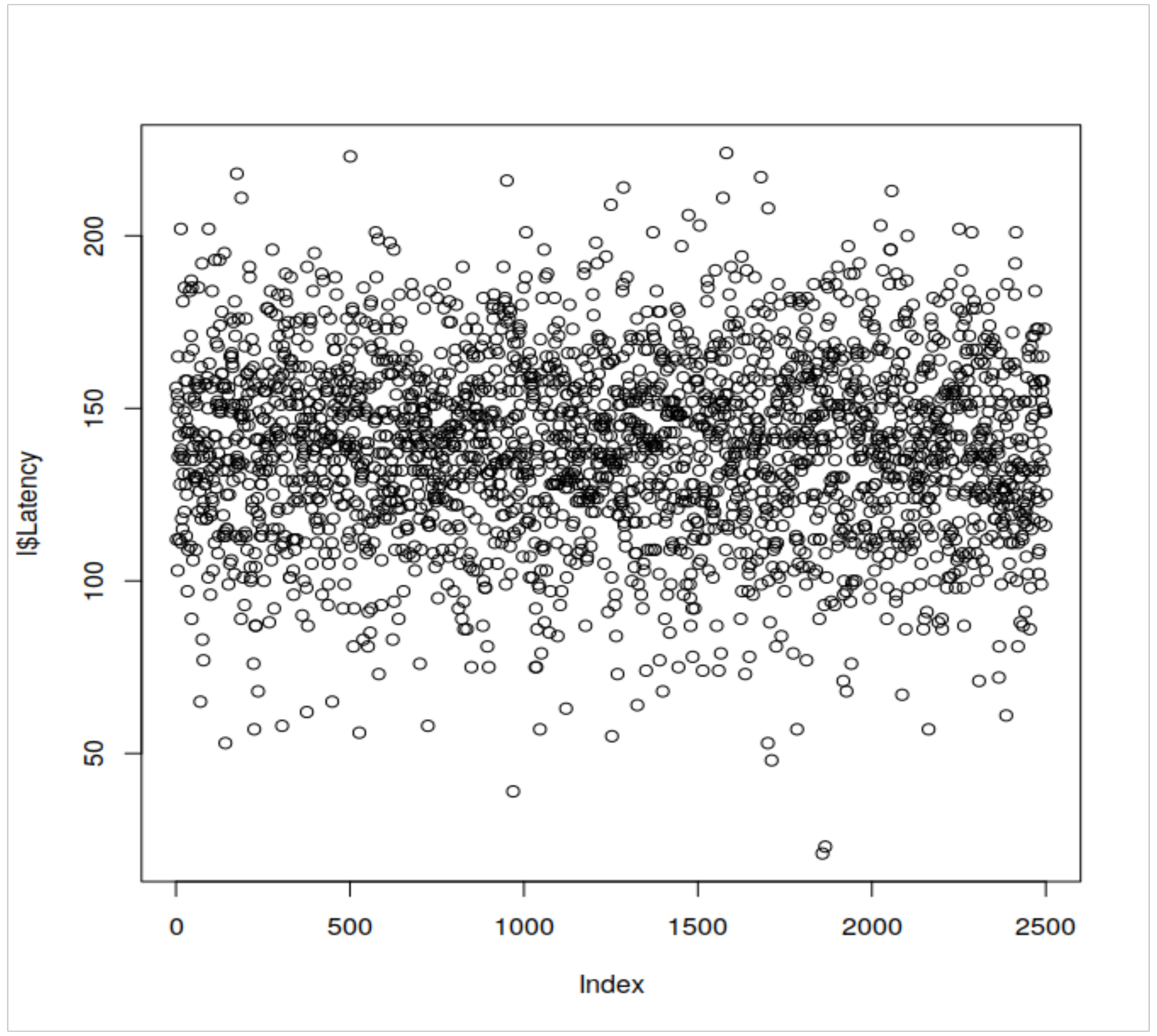
- Represent your samples **over time**!
  - Without further context, **mean** and **std. dev.** may be misleading

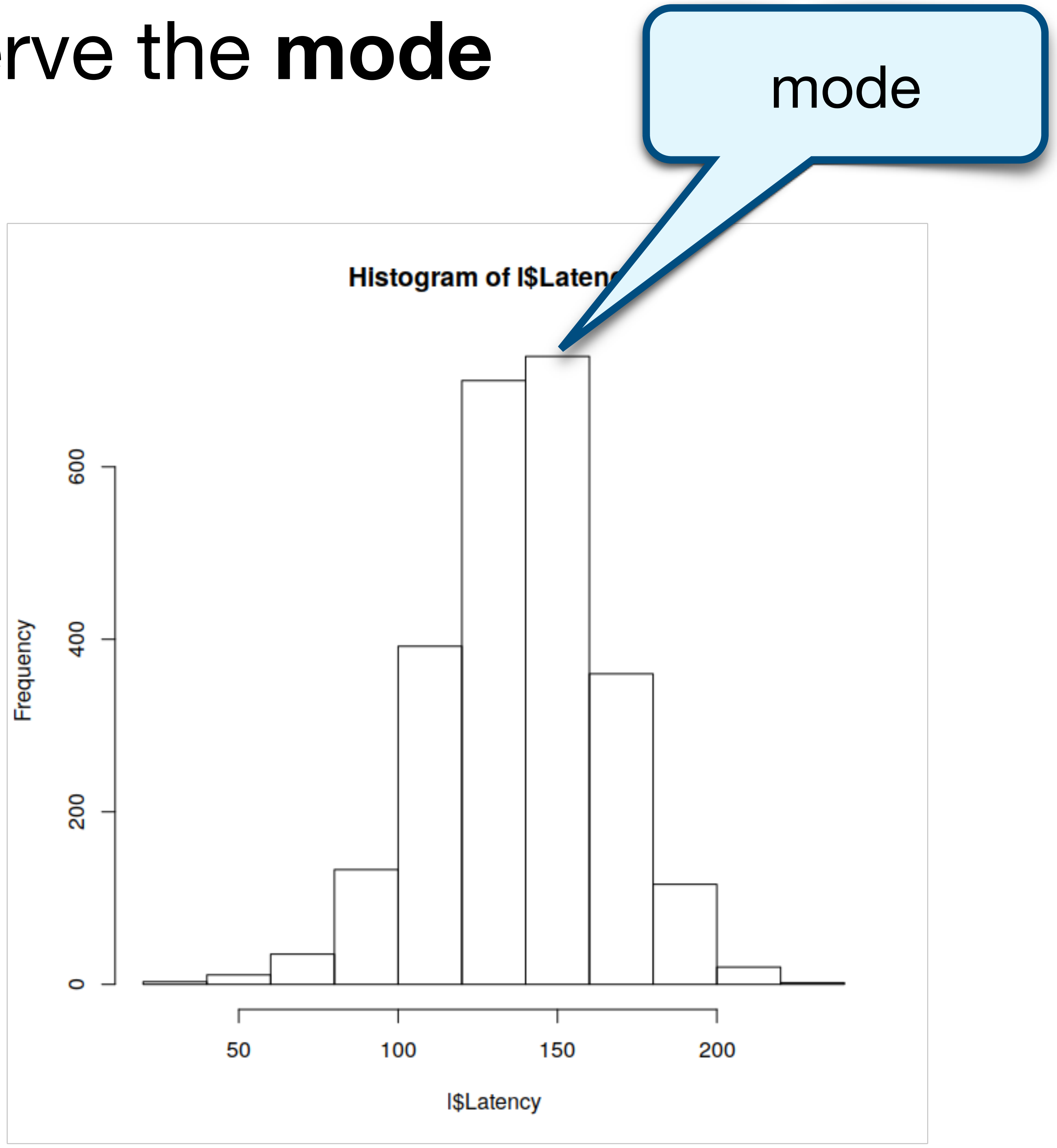


Sample Analysis

Versus frequency

- Represent the **frequency** of your samples!
  - E.g., through a **histogram** to easily observe the **mode** and results' symmetry

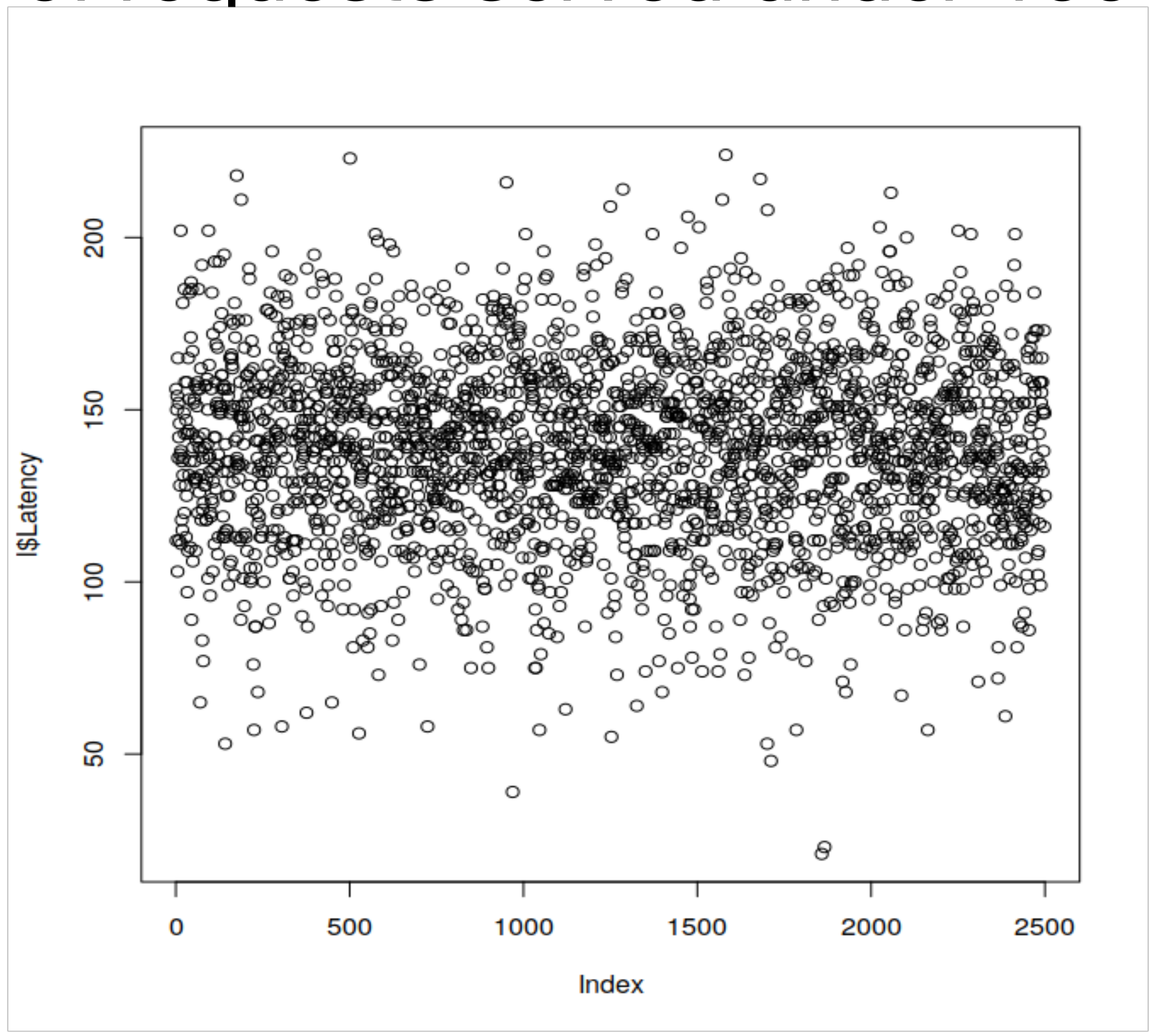


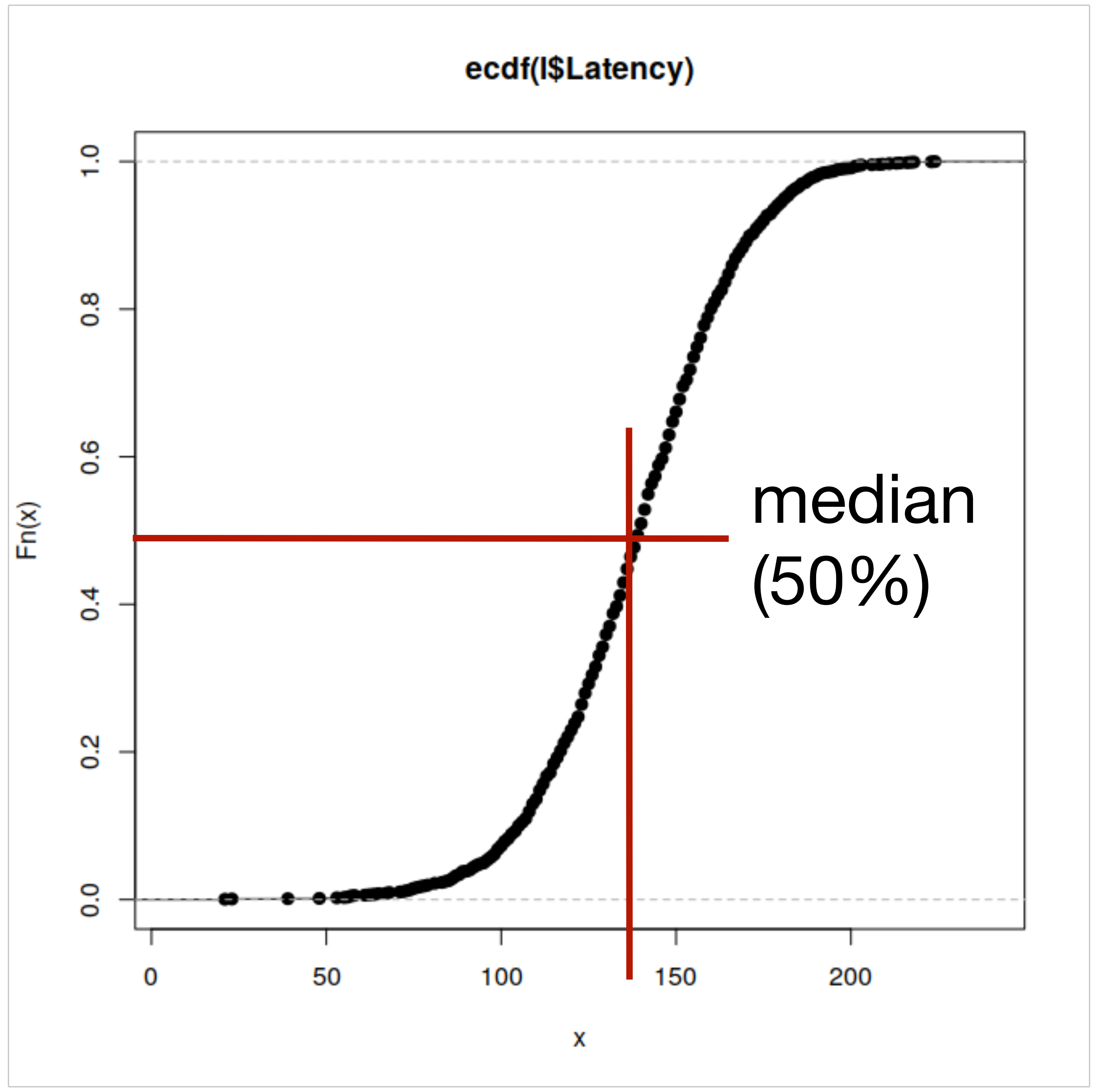


Sample Analysis

Versus frequency

- Or through an **Empirical Cumulative Distribution Function (ECDF)**
  - To easily observe the **median**, **percentiles**, **quartiles**, ….
- **Example:** Useful to observe Service Level Agreements (SLAs)
  E.g., 80% of requests served under 150 ms

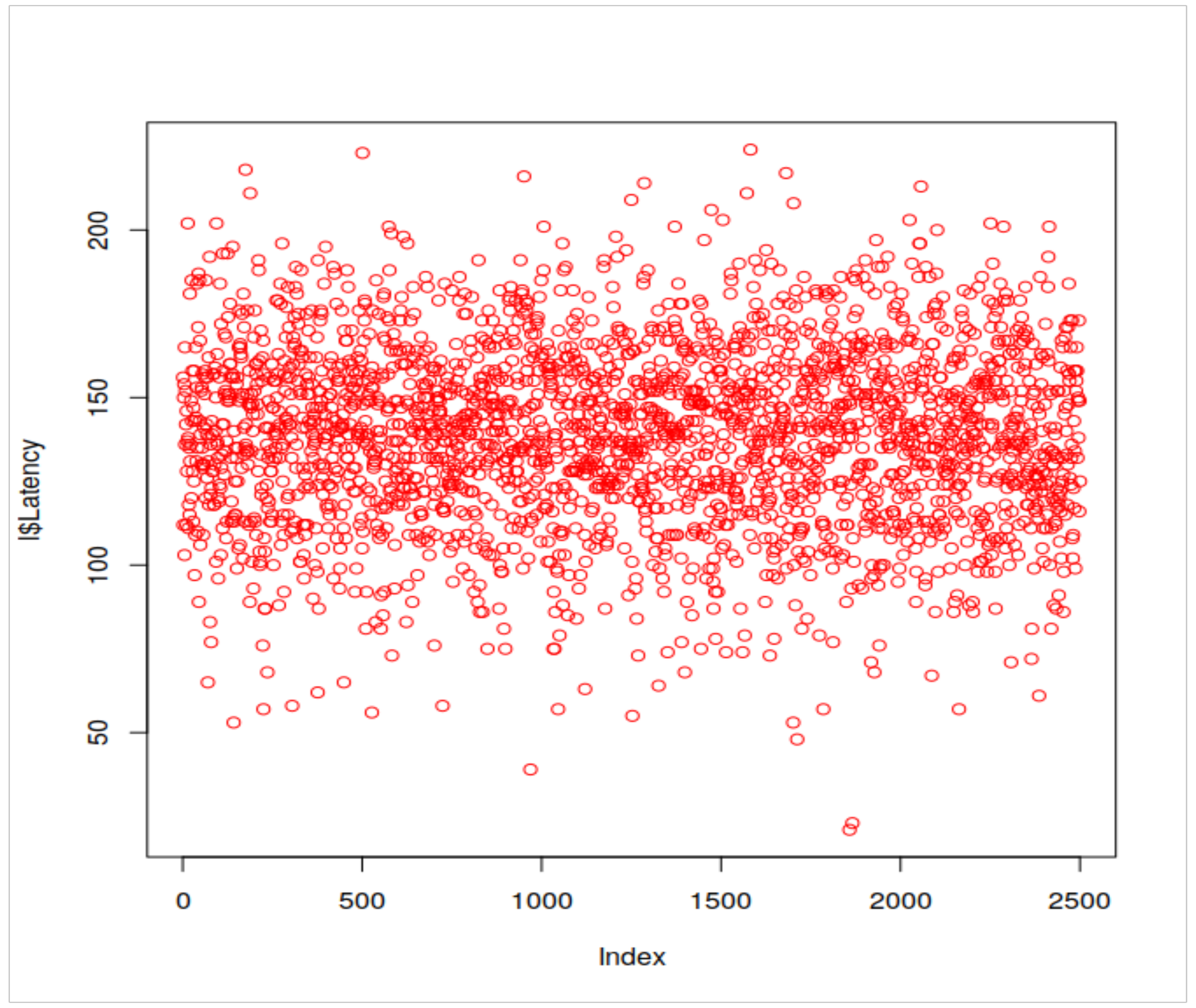


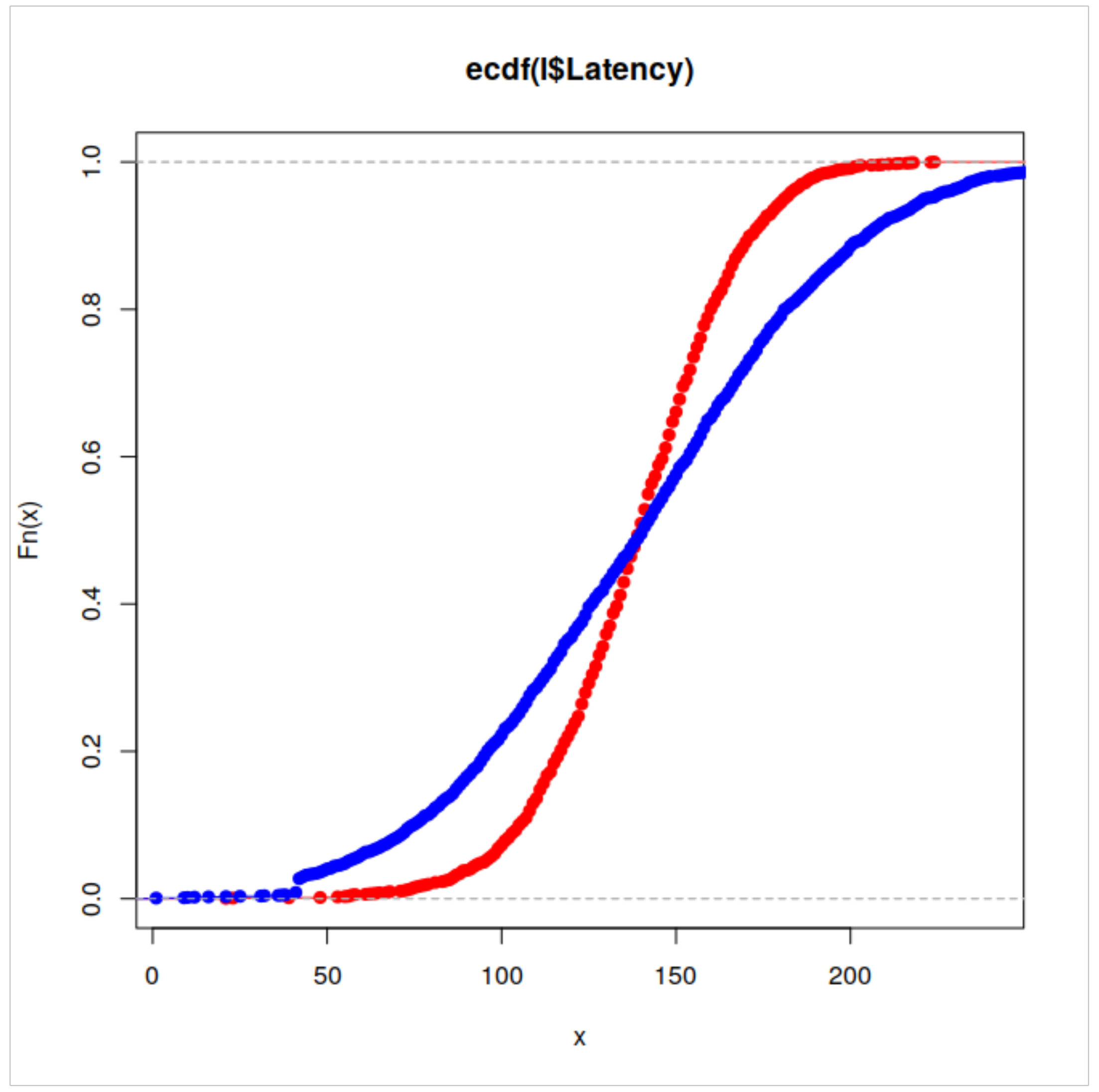


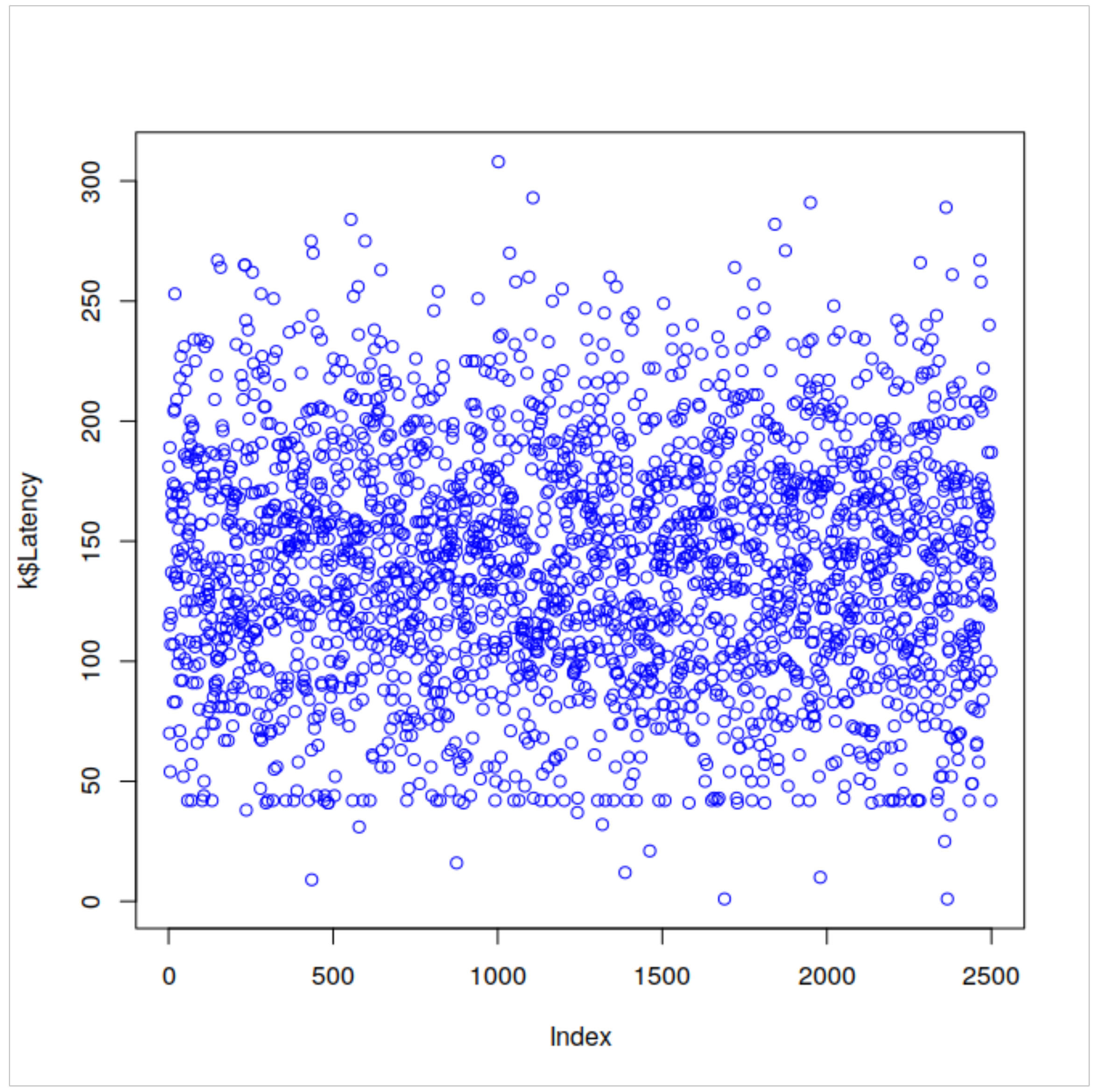
Sample Analysis

Versus frequency

- **ECDFs** are also useful for direct comparison of distributions
  - E.g., For up to 50% of requests, the blue distribution provides lower RT
  - E.g., the red distribution serves a higher percentage of overall requests with lower RT



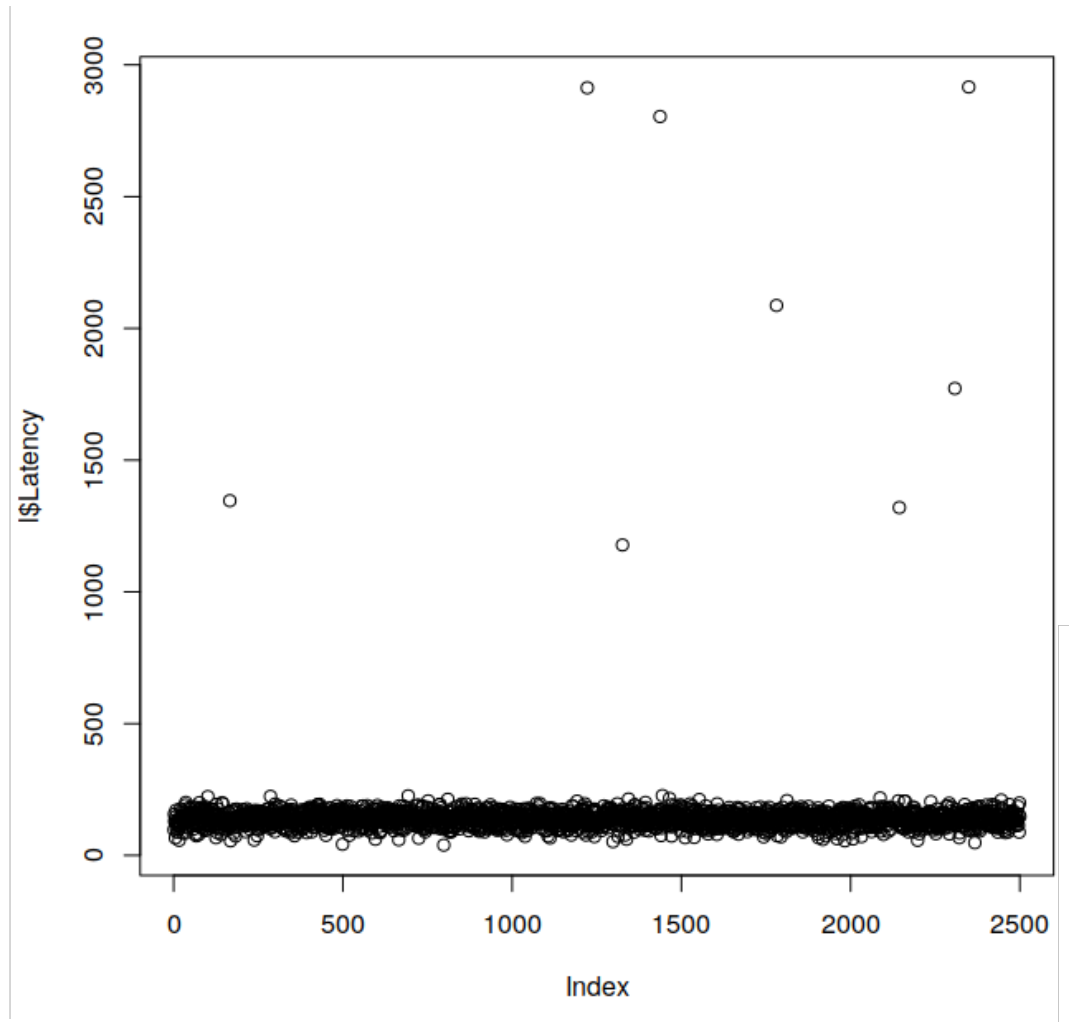


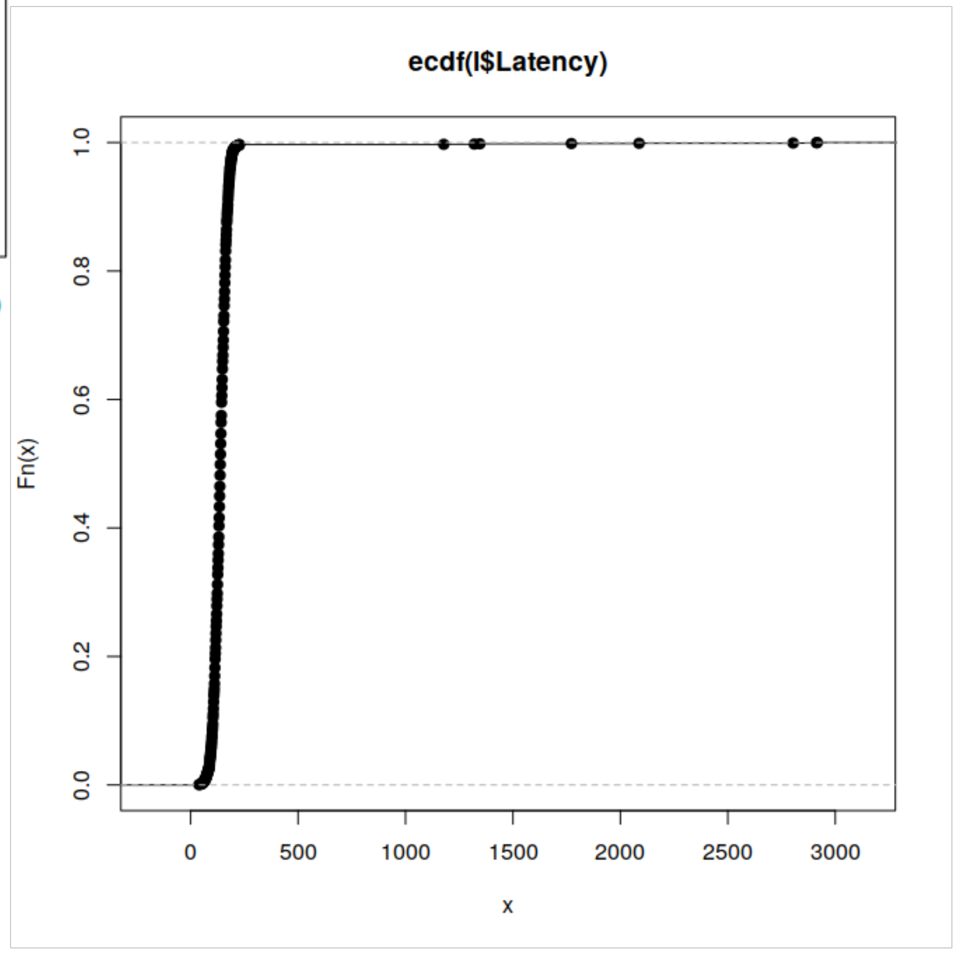


Sample Analysis

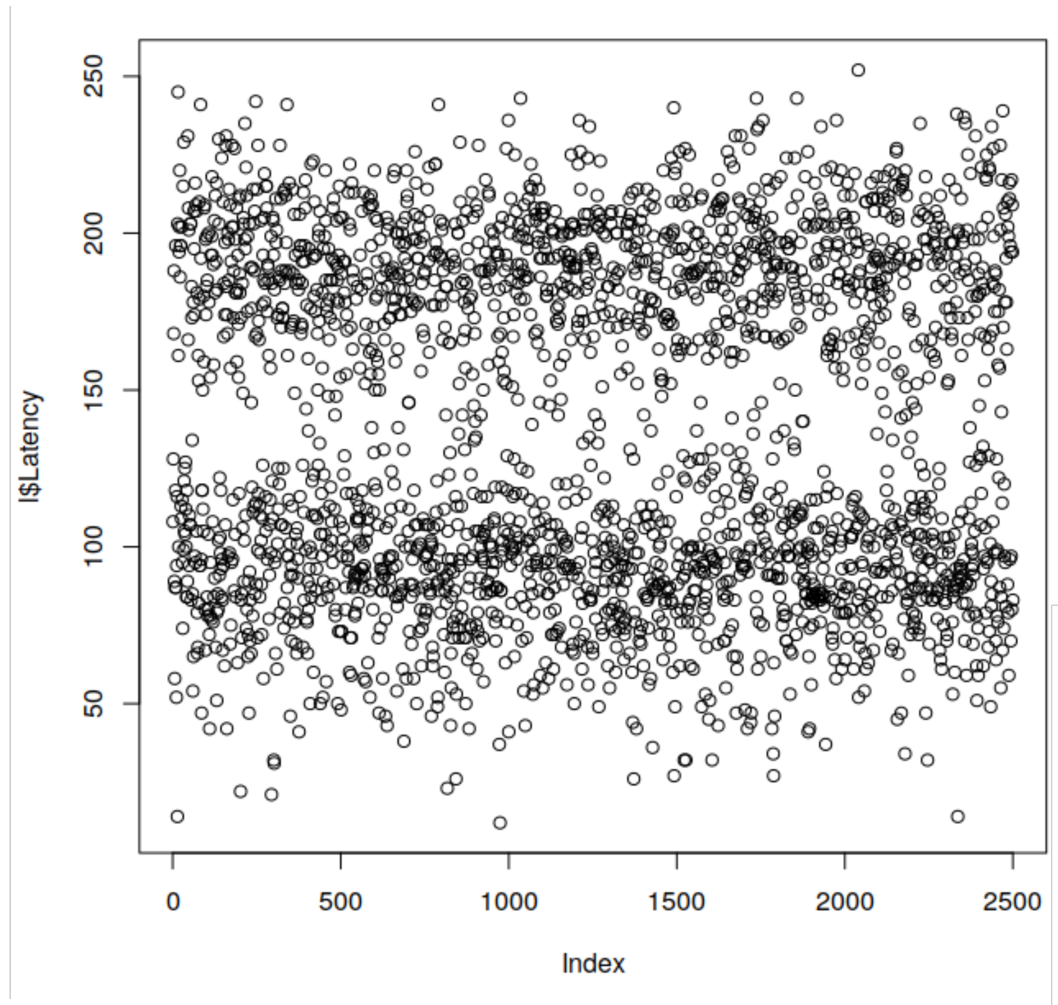
Versus frequency

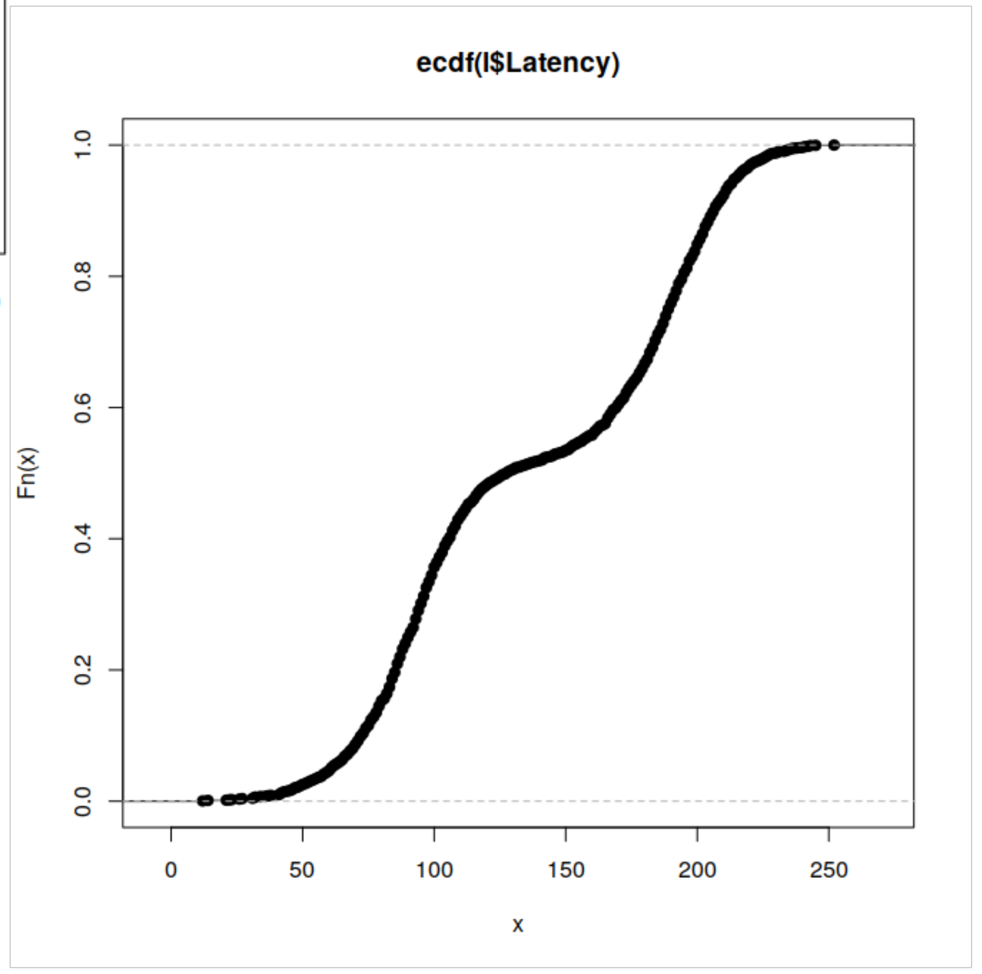
Long Tail distribution
e.g., caused by a GC pause





Bimodal distribution
e.g., caused by an “If” statement





Some Conclusions

Summarizing samples

- Use **mean**, **mode**, **median**, or **high percentiles**
  - Along with **confidence intervals (CI)**
- Combine mean and std. dev. to get the **coefficient of variation (C.O.V)**
  - **Std. dev. / mean** (usually expressed as %)
- Use **visual representations** to observe and better understand your samples!
  - **Time-series** plots, **histograms**, **ECDFs**, …

Benchmarking and Analysis Tools

Just a few examples

- Workload generator and sampling

- Data analysis

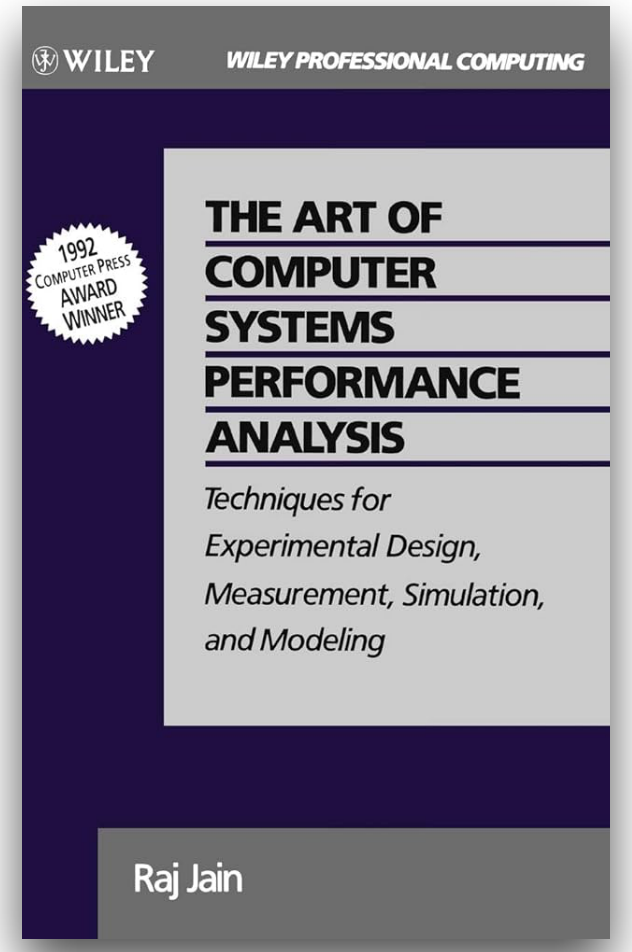
Common Mistakes

When benchmarking systems

- No goals or biased ones
  - Define the **evaluation questions** you want to answer with your experiments!
- Unsystematic approach - **Reproducibility is key!**
  - Set and document your workloads, experimental environment, and configurations!
- Unrepresentative workloads and metrics
  - Choose **realistic workloads** and experiments that help answer your questions
  - Avoid being biased (i.e., don't be afraid to show the **strengths** and **drawbacks** of the SUT)
- Wrong analysis and presentation of results (i.e., misunderstanding measurements)
  - **Inspect samples in time** for stability
    - Consider external phenomenons (e.g., cache warmups)
  - Inspect ECDFs for **distribution**
  - Then **summarize**…

Further Reading

- R. Jain, ***“The Art of Computer Systems Performance Analysis.”*** Wiley, 1991.
  - Chapters 1 to 5 and 12
  - Further reading:
    - Chapter 6, 9, 10, 11 and 13
- Coelho F, Paulo J, Vilaça R, Pereira J, Oliveira R. 2017. ***HTAPBench: Hybrid Transactional and Analytical Processing Benchmark***. International Conference on Performance Engineering (ICPE).
- Vangoor, B.K.R., Tarasov, V. and Zadok, E., 2017. ***To FUSE or not to FUSE: Performance of user-space file systems***. In 15th USENIX Conference on File and Storage Technologies (FAST 17) (pp. 59-72).



Questions?