Advanced Systems Lab
Tutorial II
Life Cycle Experiment

G. Alonso

Systems Group

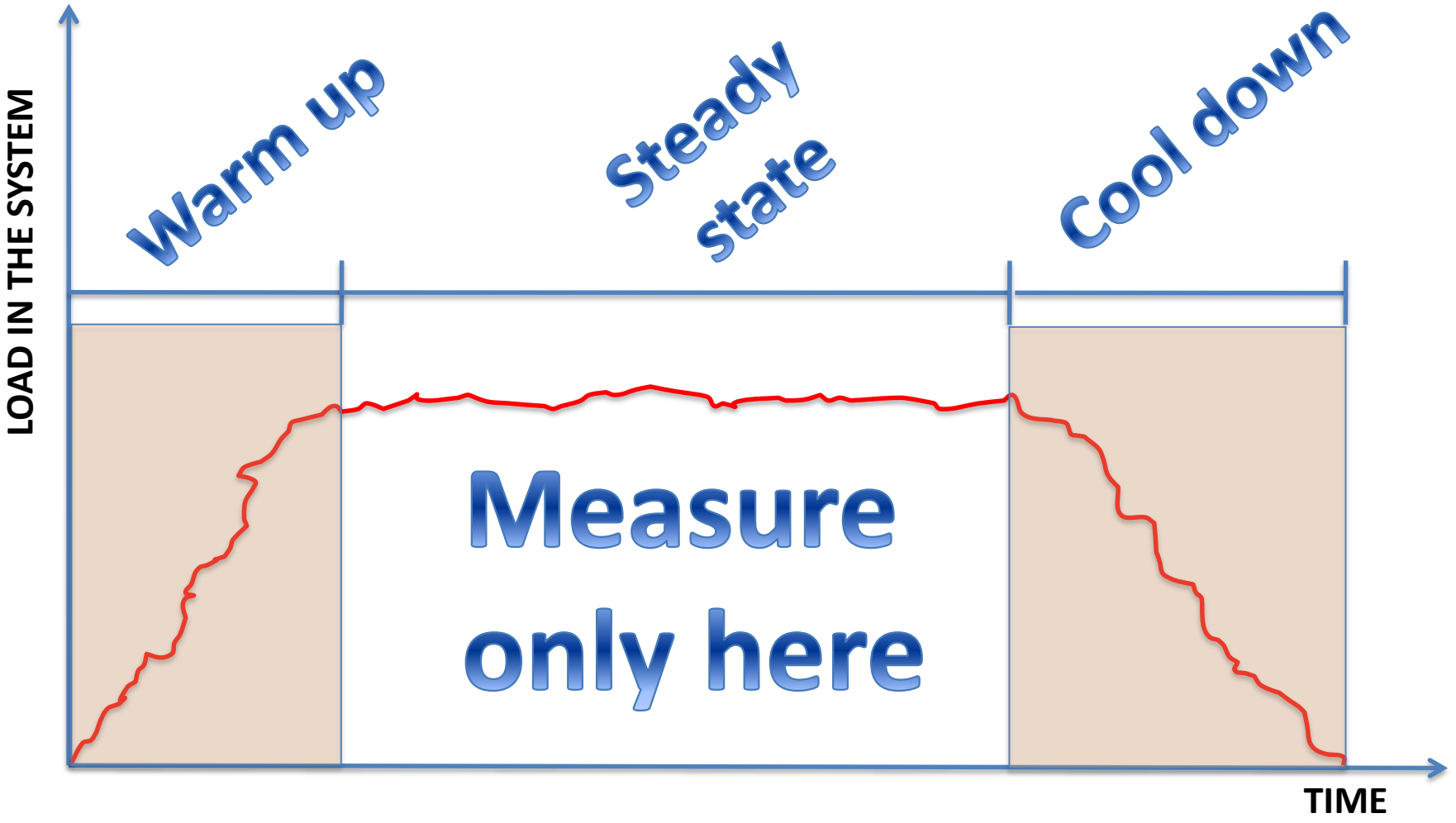
http://www.systems.ethz.ch

When to measure

What and when to measure

- Decide on the parameters to measure:
  - Throughput, response time, latency, etc.
- Design your experiment
  - Configuration, data, load generators, instrumentation, hypothesis
- Run the experiment and start measuring:
  - When to measure (life cycle of an experiment)
  - What to measure (sampling)

Life cycle of an experiment



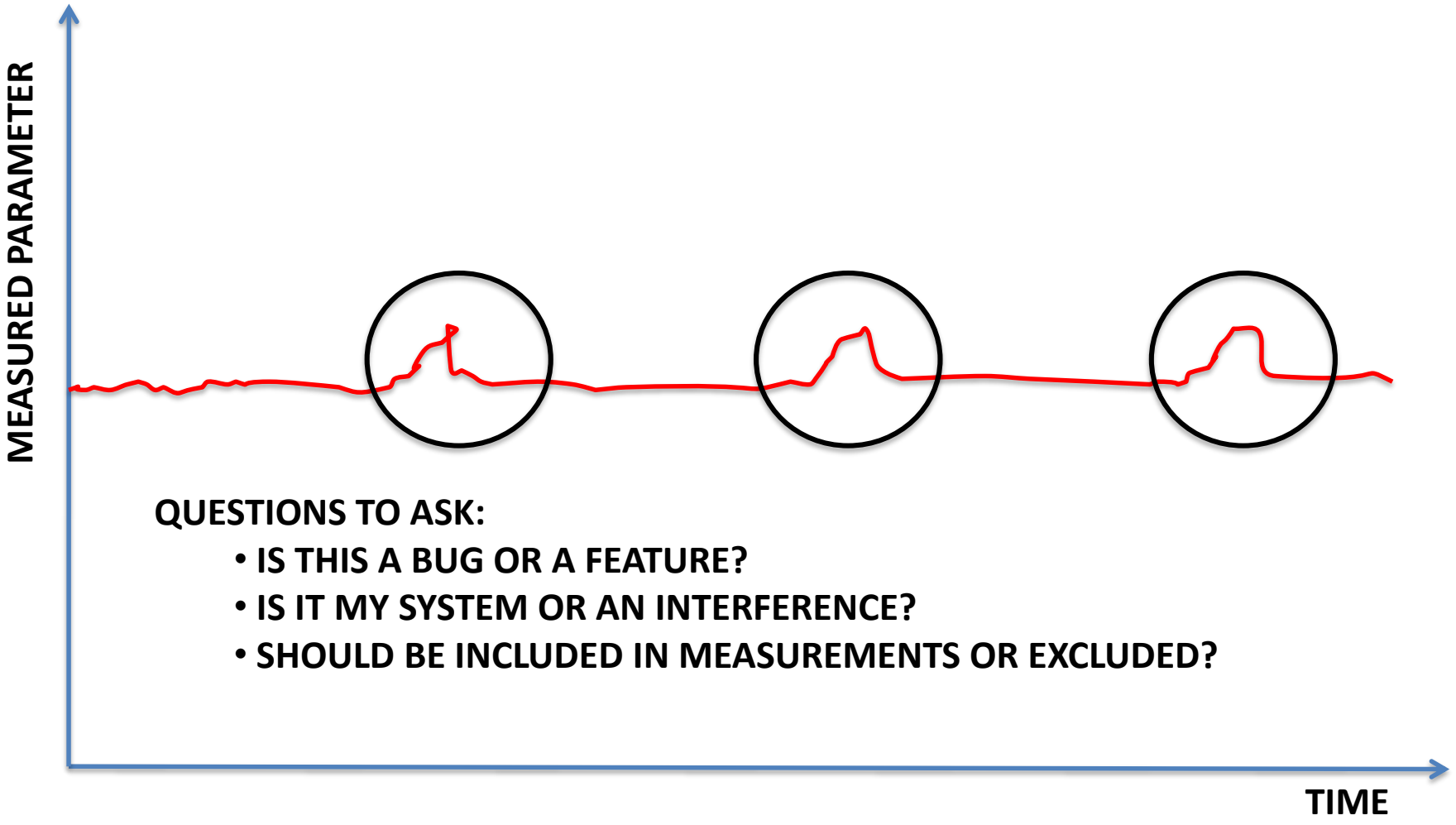
Warm up phase

- Warm up phase
  - Time until clients are all up, caches full (warm), data in main memory, etc.
  - Throughput lower than steady state throughput
  - Response time better than in steady state
- Detect by watching measured parameter changing with time
- Measure only in steady state

Cool down phase

- Cool down phase
  - Clients start finishing, resulting in less load in the system
  - Throughput is lower than in steady state
  - Response time better than in steady state
- Detect by observing when measured parameter suddenly changes behavior
- Stop measuring when clients no longer generate a steady load

Patterns to watch for - glitches

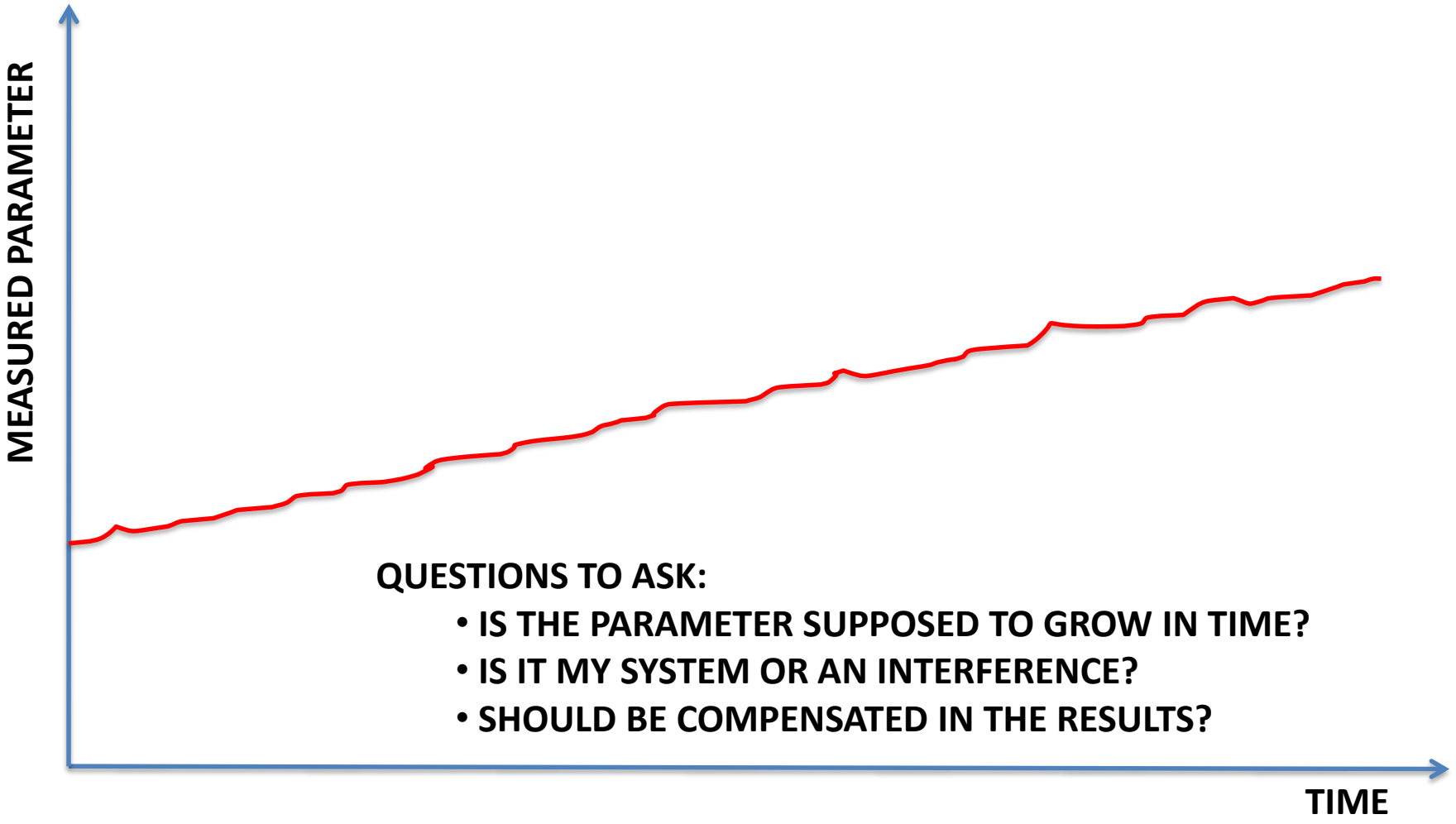


QUESTIONS TO ASK:

- **IS THIS A BUG OR A FEATURE?**
- **IS IT MY SYSTEM OR AN INTERFERENCE?**
- **SHOULD BE INCLUDED IN MEASUREMENTS OR EXCLUDED?**

ASSUME STEADY STATE MEASUREMENTS

Patterns to watch for - trends

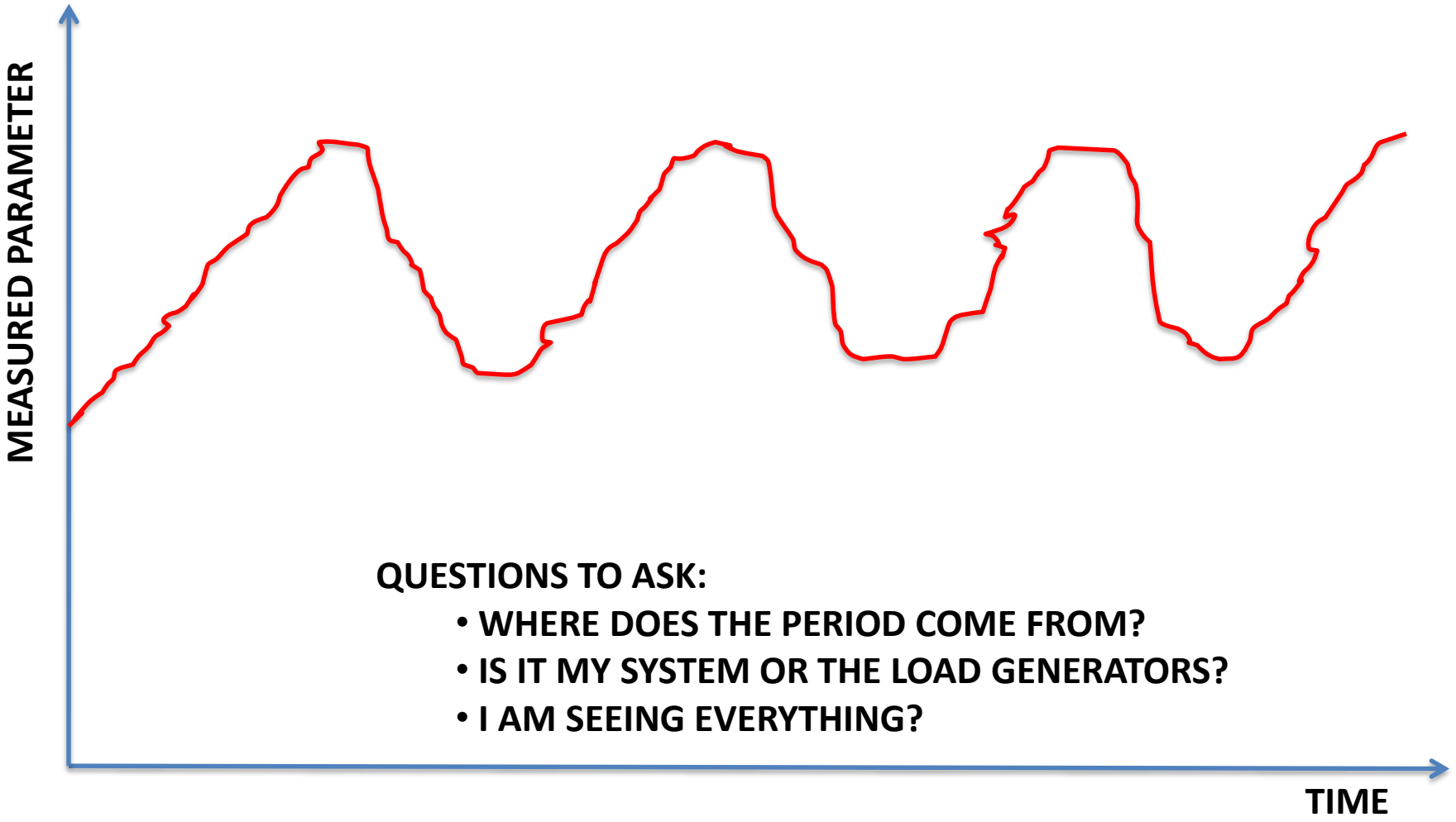


QUESTIONS TO ASK:

- **IS THE PARAMETER SUPPOSED TO GROW IN TIME?**
- **IS IT MY SYSTEM OR AN INTERFERENCE?**
- **SHOULD BE COMPENSATED IN THE RESULTS?**

ASSUME STEADY STATE MEASUREMENTS

Patterns to watch for - periodic



QUESTIONS TO ASK:

- **WHERE DOES THE PERIOD COME FROM?**
- **IS IT MY SYSTEM OR THE LOAD GENERATORS?**
- **I AM SEEING EVERYTHING?**

ASSUME STEADY STATE MEASUREMENTS

Why are these patterns relevant?

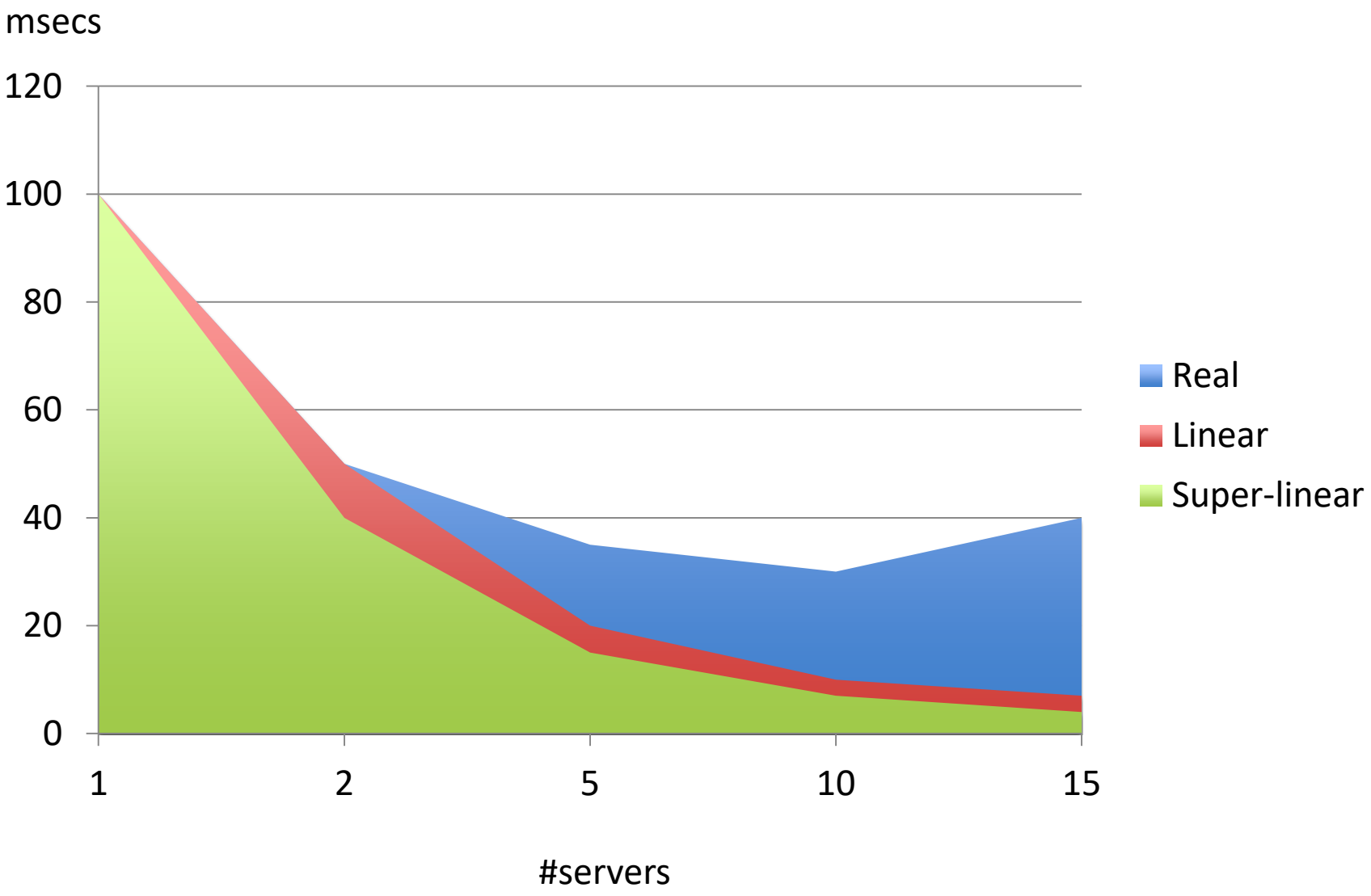
- Too few measurements and too short experiments are meaningless
  - May not capture system behavior
  - May not show pathological behavior
  - May not reflect real values
- Statistics are a way to address some of these issues by providing more information from the data and a better idea of the system behavior
  - but applying statistics to the wrong data will not help!

UNDERSTANDING THROUGHPUT AND RESPONSE TIME

Understanding Performance

- Response Time
  - critical path analysis in a task dependency graph
  - „partition“ expensive tasks into smaller tasks
- Throughput
  - queueing network model analysis
  - „replicate“ resources at bottleneck

Response Times



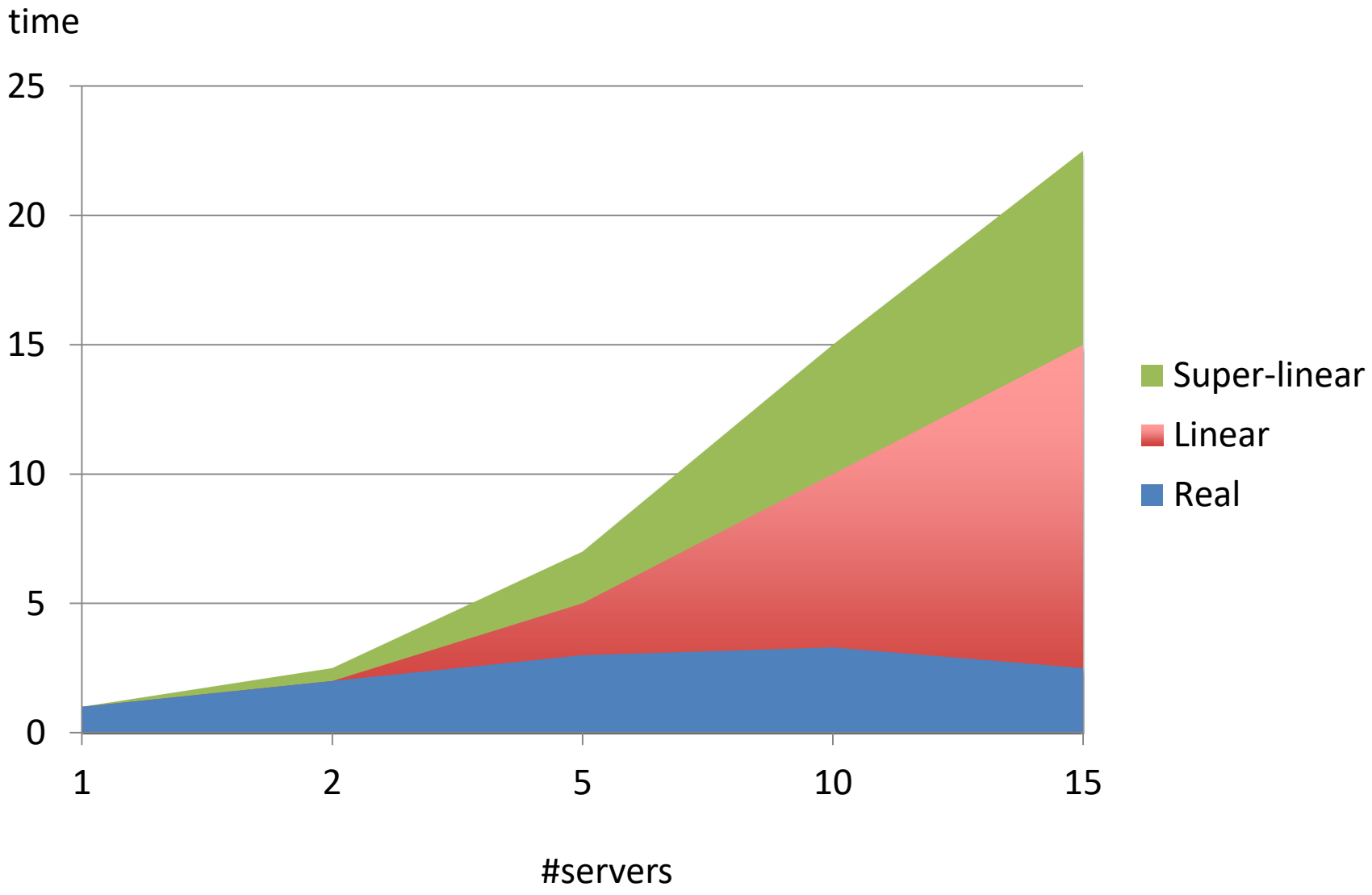
Why are response times long?

- Because operations take long
  - cannot travel faster than light
  - delays even in „single-user“ mode
- Because there is a bottleneck
  - contention of concurrent requests on a resource
  - requests wait in queue before resource available
  - add resources to parallelize requests at bottleneck

Speed-up

- Goal: test ability of SUT to reduce response time for the same load by adding resources
  - measure response time with 1 resource
  - measure response time with N resources
  - SpeedUp(N) = RT(1) / RT(N)
- Ideal
  - SpeedUp(N) is a linear function
  - can you imagine super-linear speed-ups?

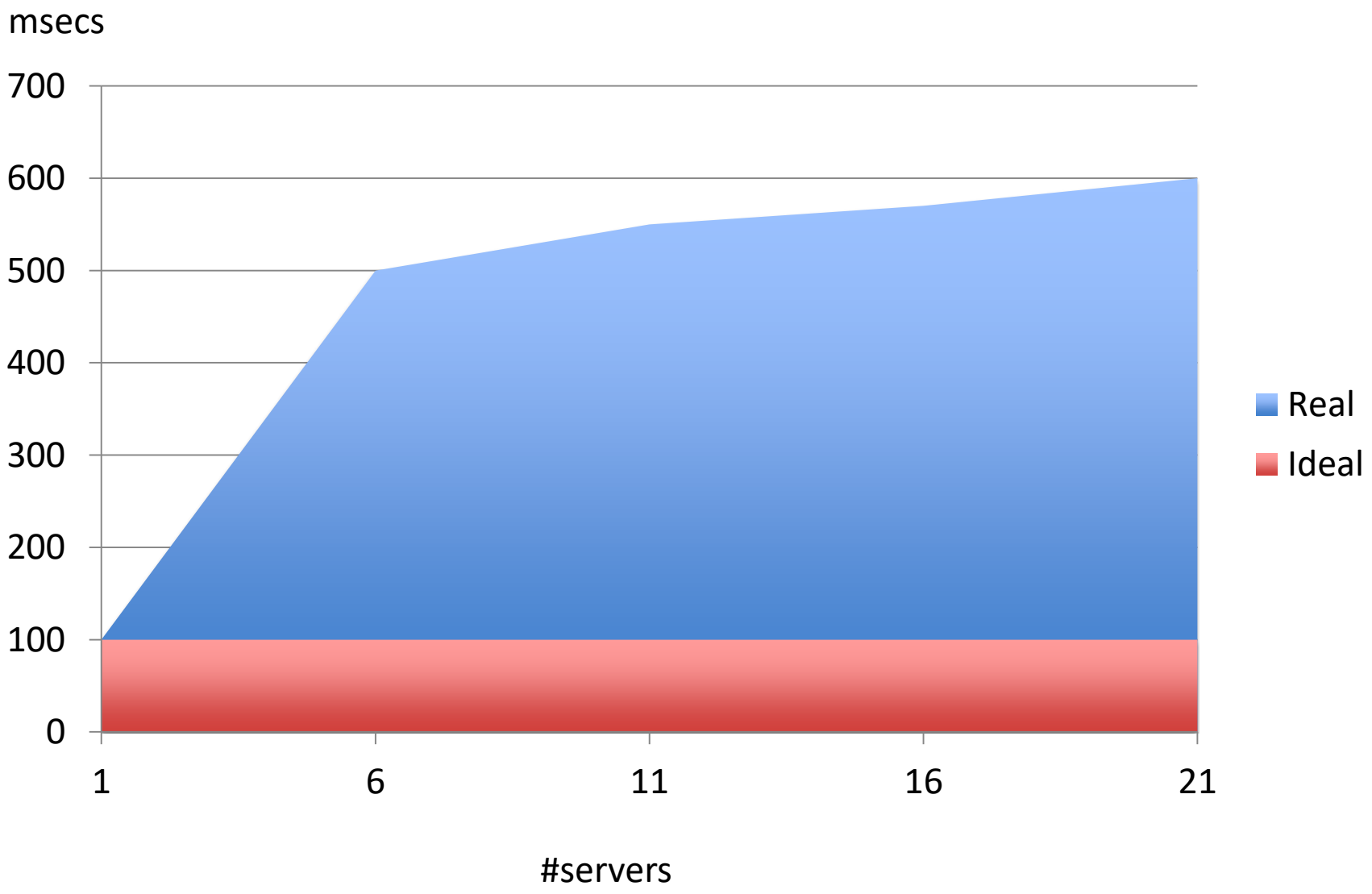
Speed Up



Scale-up

- Goal: test ability of SUT to deal with larger loads by adding resources
  - measure response time with 1 server, 1 unit problem
  - measure response time with N servers, N units problem
  - ScaleUp(N) = RT(1) / RT(N)
- Ideal
  - ScaleUp(N) is a constant function

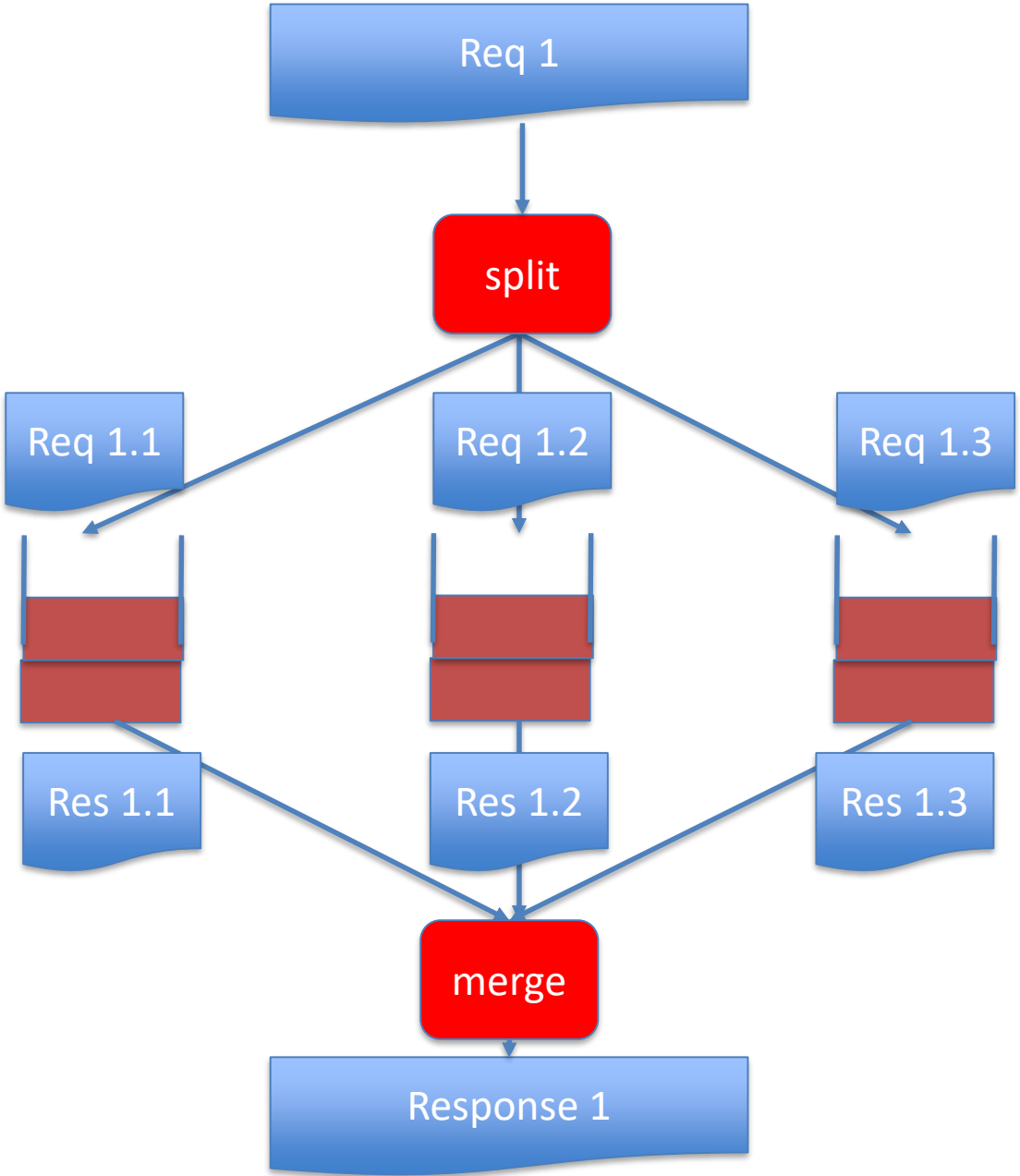
Scale Up Exp.: Response Time



Scale Out

- Test how SUT behaves with increasing load
  - measure throughput: 1 server, 1 user
  - measure throughput: N servers, N users
  - ScaleOut(N) = Tput(1) / Tput(N)
- Ideal
  - Scale-Out should behave like Scale-Up
  - (often terms are used interchangeably;
    but worth-while to notice the differences)

Why is speed-up sub-linear?

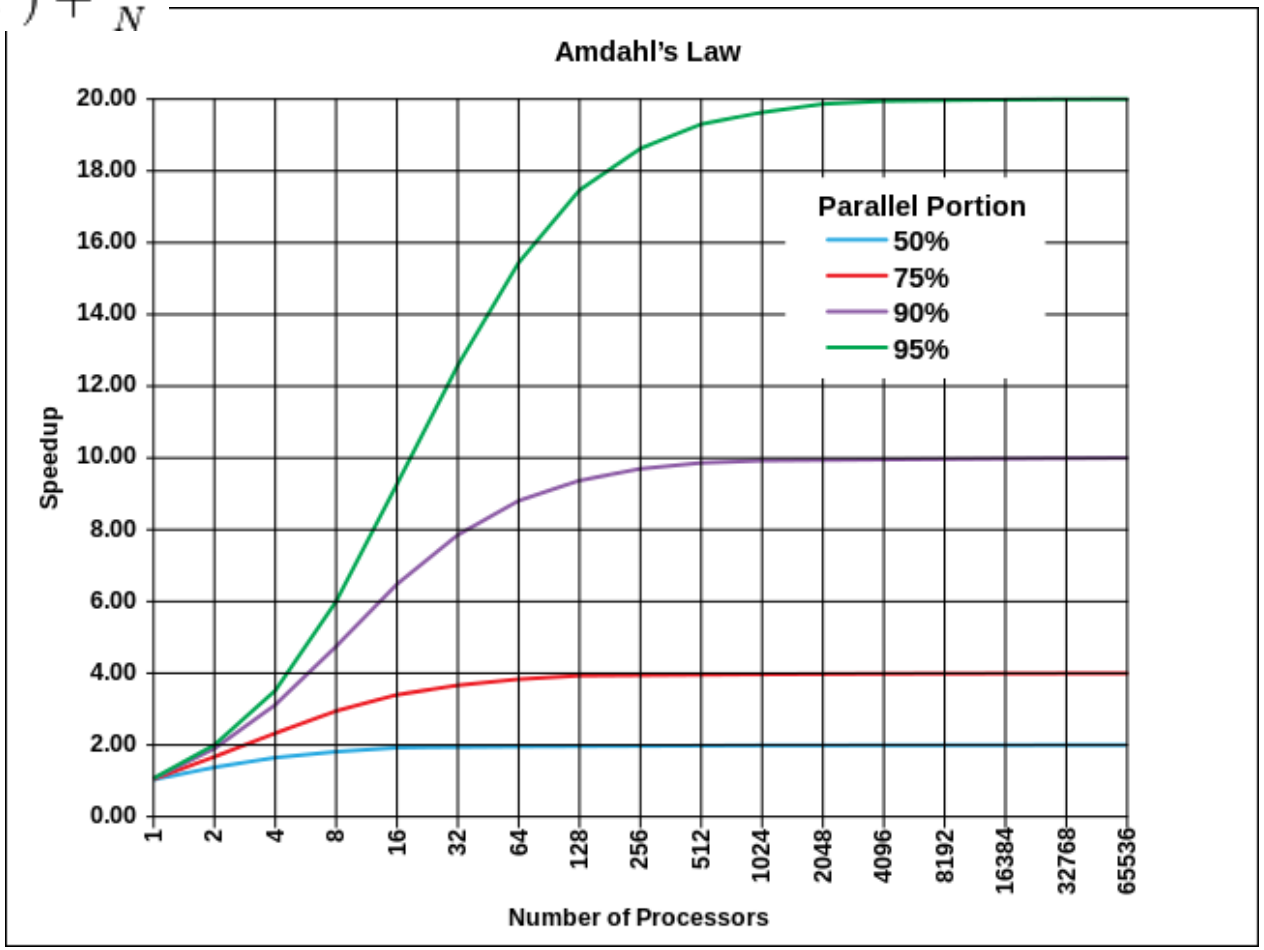


Why is speed-up sub-linear?

- Cost for „split“ and „merge“ operation
  - those can be expensive operations
  - try to parallelize them, too
- Interference: servers need to synchronize
  - e.g., CPUs access data from same disk at same time
  - shared-nothing architecture
- Skew: work not „split“ into equal-sized chunks
  - e.g., some pieces much bigger than others
  - keep statistics and plan better

Amdahl's Law

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}.$$



Summary

- Improve Response Times by „partitioning“
  - divide & conquer approach
  - Works well in many systems
- Improve Throughput by relaxing „bottleneck“
  - add resources at bottleneck
- Fundamental limitations to scalability
  - resource contention (e.g., lock conflicts in DB)
  - skew and poor load balancing
- Special kinds of experiments for scalability
  - speed-up and scale-up experiments