Advanced Systems Lab
Tutorial II
Experimental design

G. Alonso

Systems Group

http://www.systems.ethz.ch

THINKING IN ADVANCE

Quantitative questions about systems

- Absolute or comparative performance analysis
  - How many operations can a system run per second? How long does an operation take?
  - How many concurrent clients does a system support?
  - Do SSDs make an application faster than hard disks?
  - Should I use quick sort instead of merge sort for an online catalogue?
  - Where is the bottleneck in the system?

How to answer such questions?

- Experiments
  - You implement / install „system(s) under test“ (SUT)
  - You run benchmarks and measure observable results
- Modeling
  - You build a model of the „system(s) under test“
  - You calculate results with model
- Simulation
  - You implement a system that behaves like SUT
  - You run benchmarks and measure computed results

Experiments vs. Modeling

- Experiments
  - Often expensive to implement
  - Specific to environment (e.g., hardware used)
  - Accurate (quantitative) results
  - Sometimes misleading
- Modeling
  - Typically cheap
  - General
  - Qualitative results
  - You always learn something
- Use modeling whenever you can
  - Unfortunately, modern systems are too complex

Methodology

1. Ask the right question
   - Define the „system(s) under test“
   - Define what to measure and understand why
   - Define relevant workloads, understand parameters
2. Make a hypothesis
   - „A good scientist predicts the results and explains later why something totally different happened.“
3. Carry out experiment (real system, model)
   - Run workloads, measure metrics
4. Report results, analyze results, gotoStep 1
   - Give answer to question, possibly refine question

Making a Hypothesis

- Use the same format as the final results
  - Draw graphs with expected results
  - Even try to predict variance and statistical properties
  - Make bullet points with explanations
  - Use „modeling“ to make hypothesis
- Share hypothesis with your customer
  - Validates whether you are asking the right question
  - i.e., can you make decisions if results turn out like that
- Comparison of expected vs. real results
  - Essential to find bugs in your experiments
  - Essential to understand real results

Hint for project

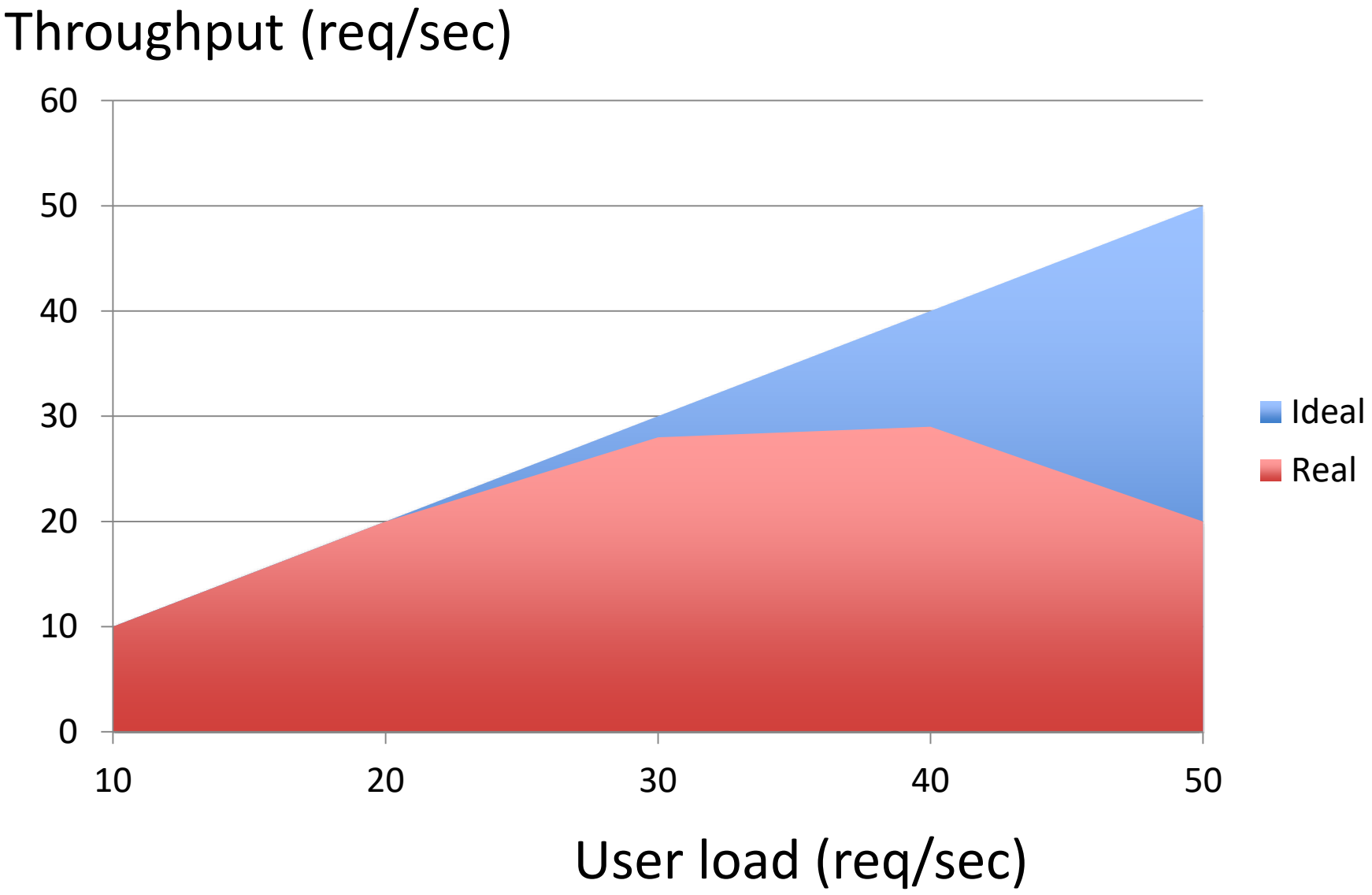
Describing your hypothesis, the experiments, comparing both, and explaining the similarities or differences is a good way to write the report for milestone 2

Make the hypothesis before you run the experiment

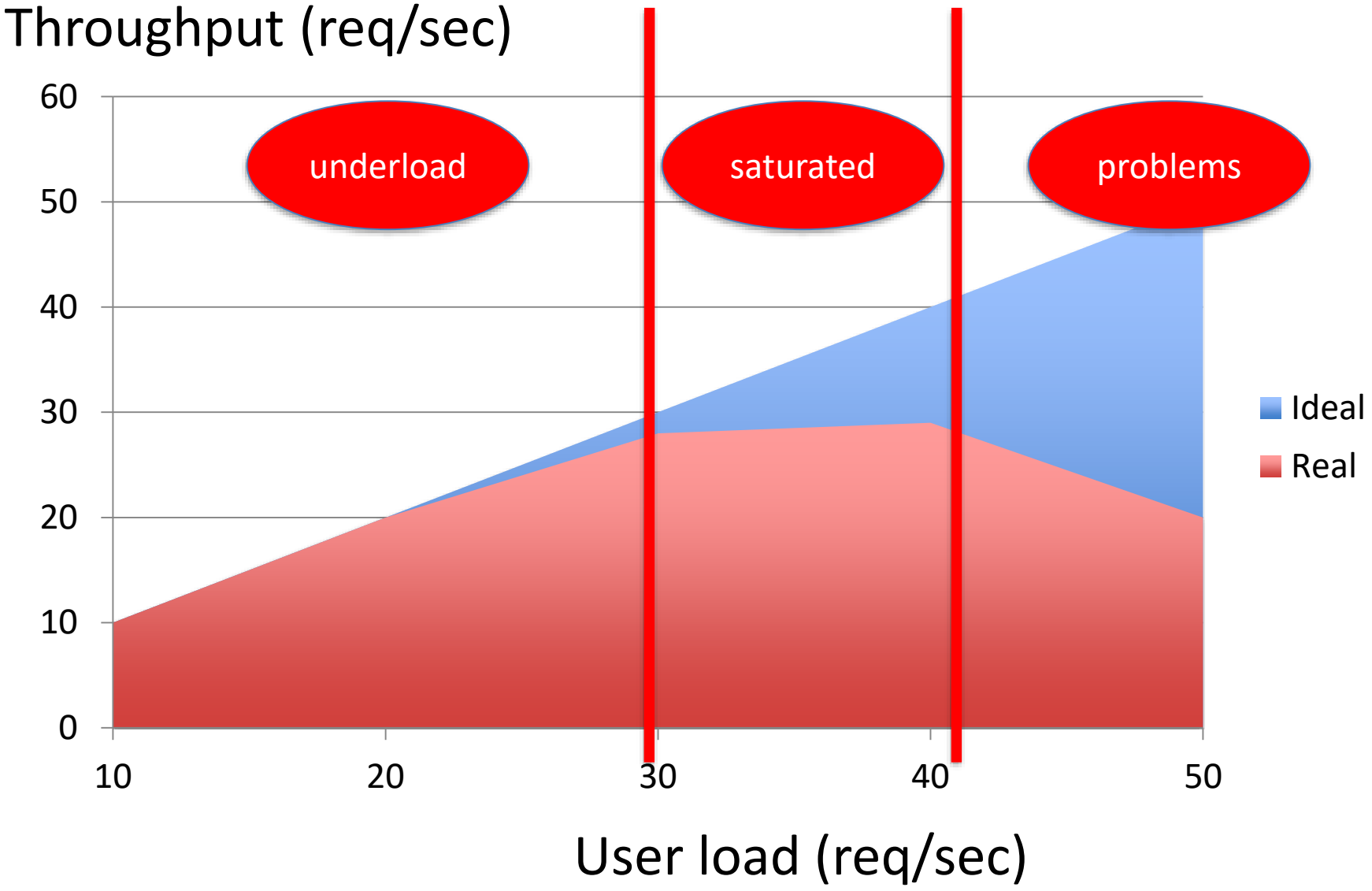
Example

- Metric 1: Throughput (y-axis of graphs)
  - requests completed per unit of time (secs)
  - count only “successful” requests (no error, < timeout)
- Metric 2: Response Time (y-axis of graphs)
  - max/min/avg time (secs) to complete a request
- Parameter: User Load (x-axis of graphs)
  - number of requests arriving per unit of time (secs)

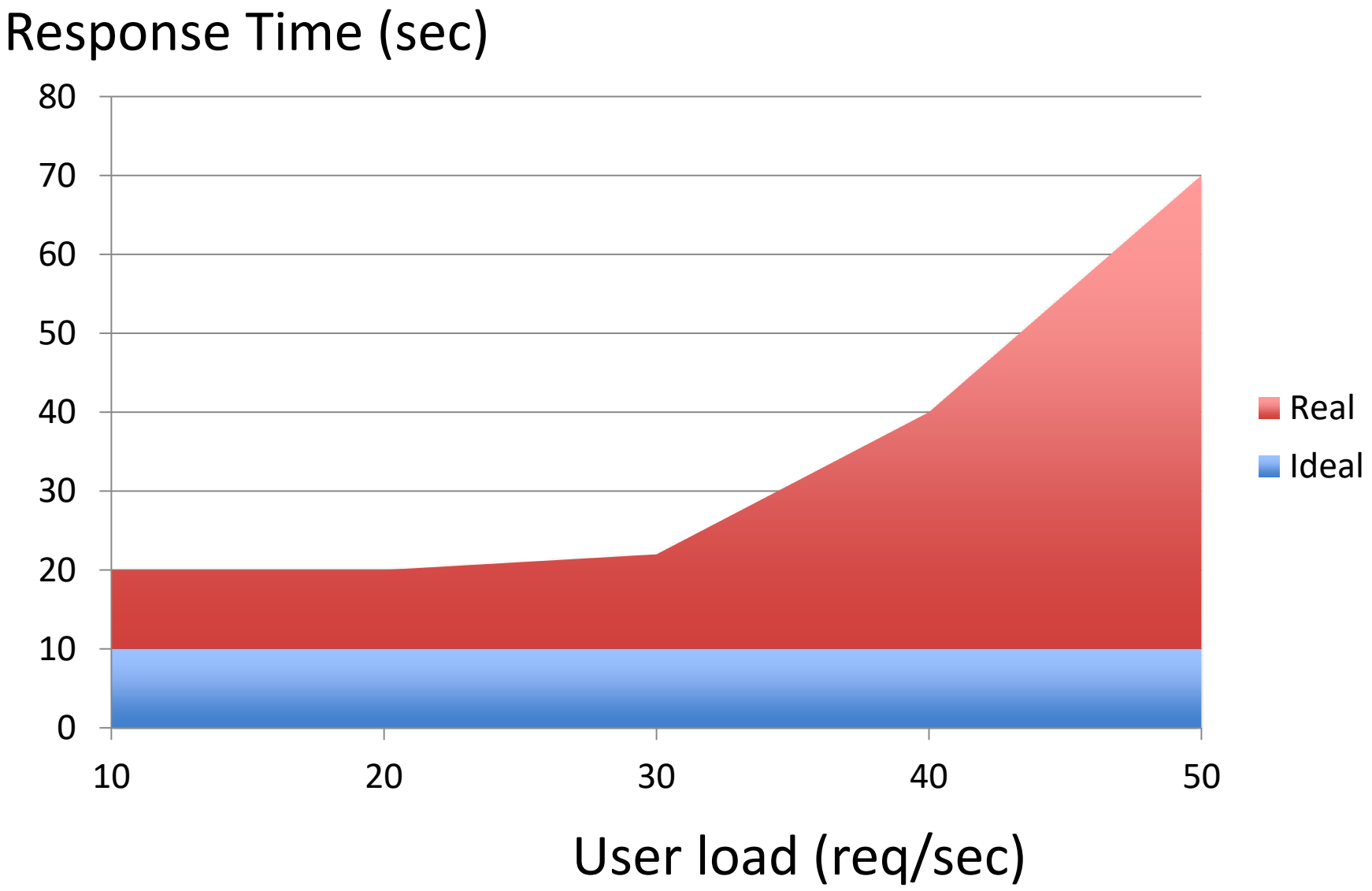
Example: Throughput



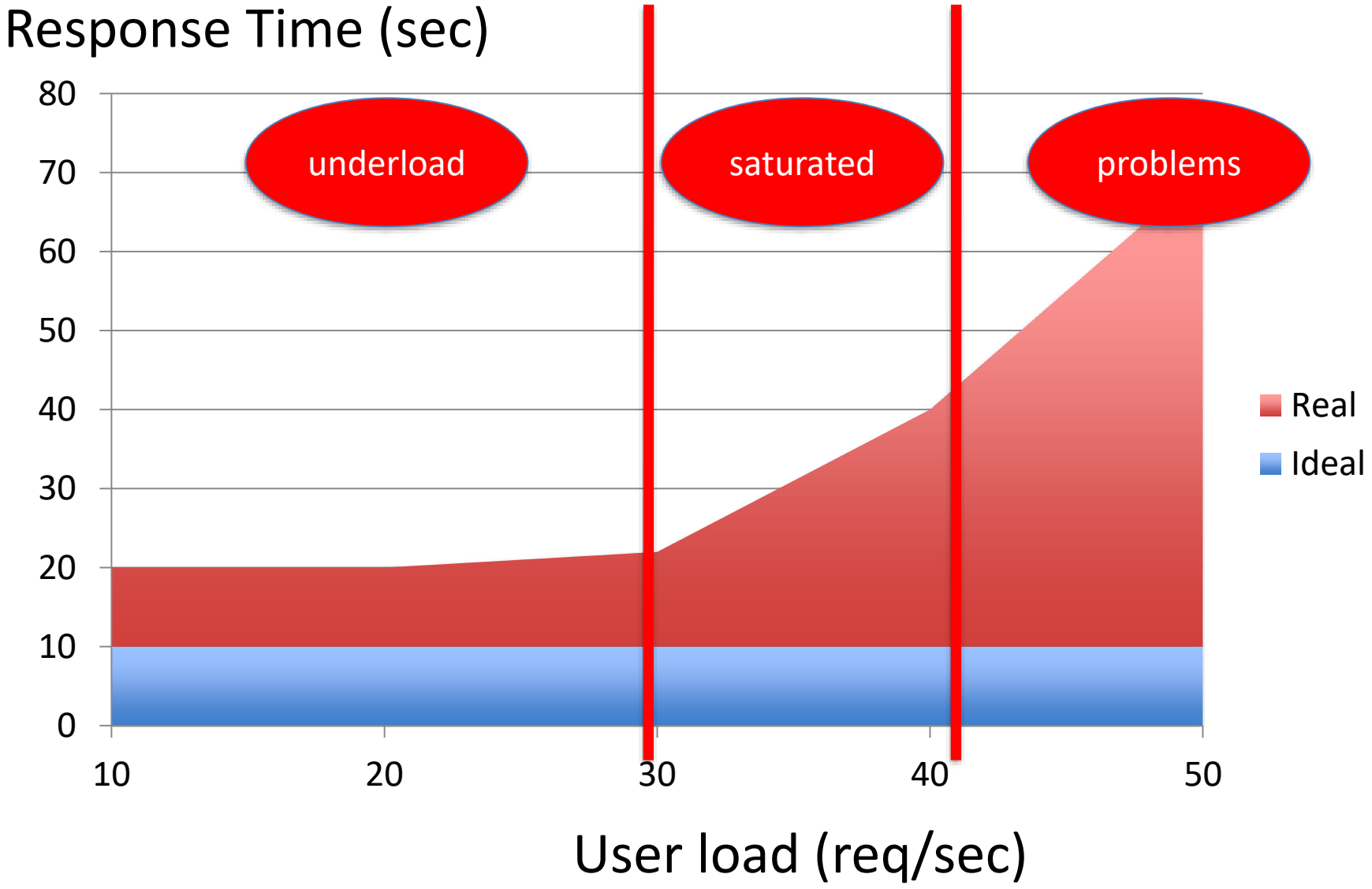
Example: Throughput Analysis



An Example: Avg. Response Time



An Example: Avg. Response Time



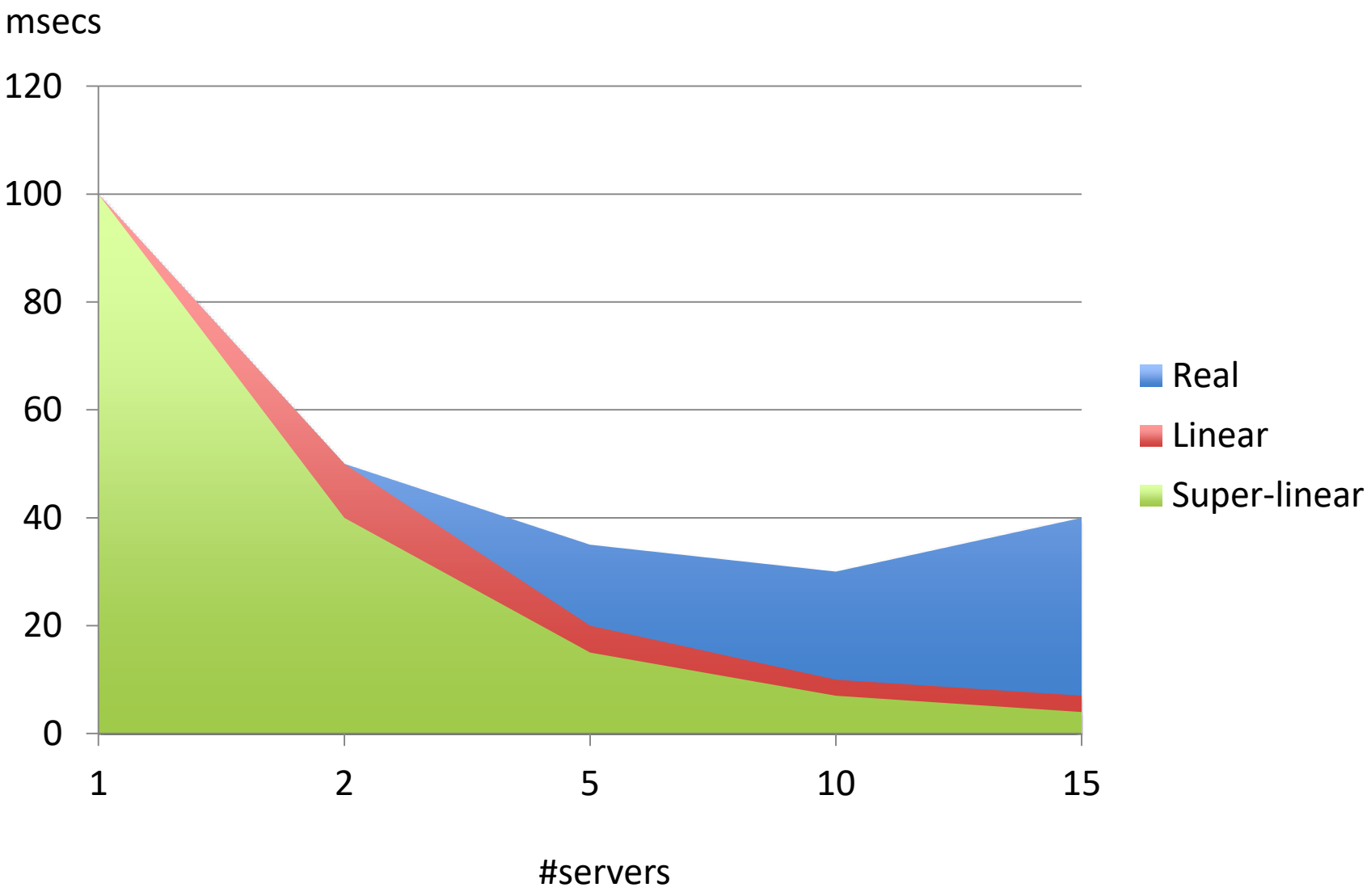
Read
chapters 1, 2, and 3
from the text book

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



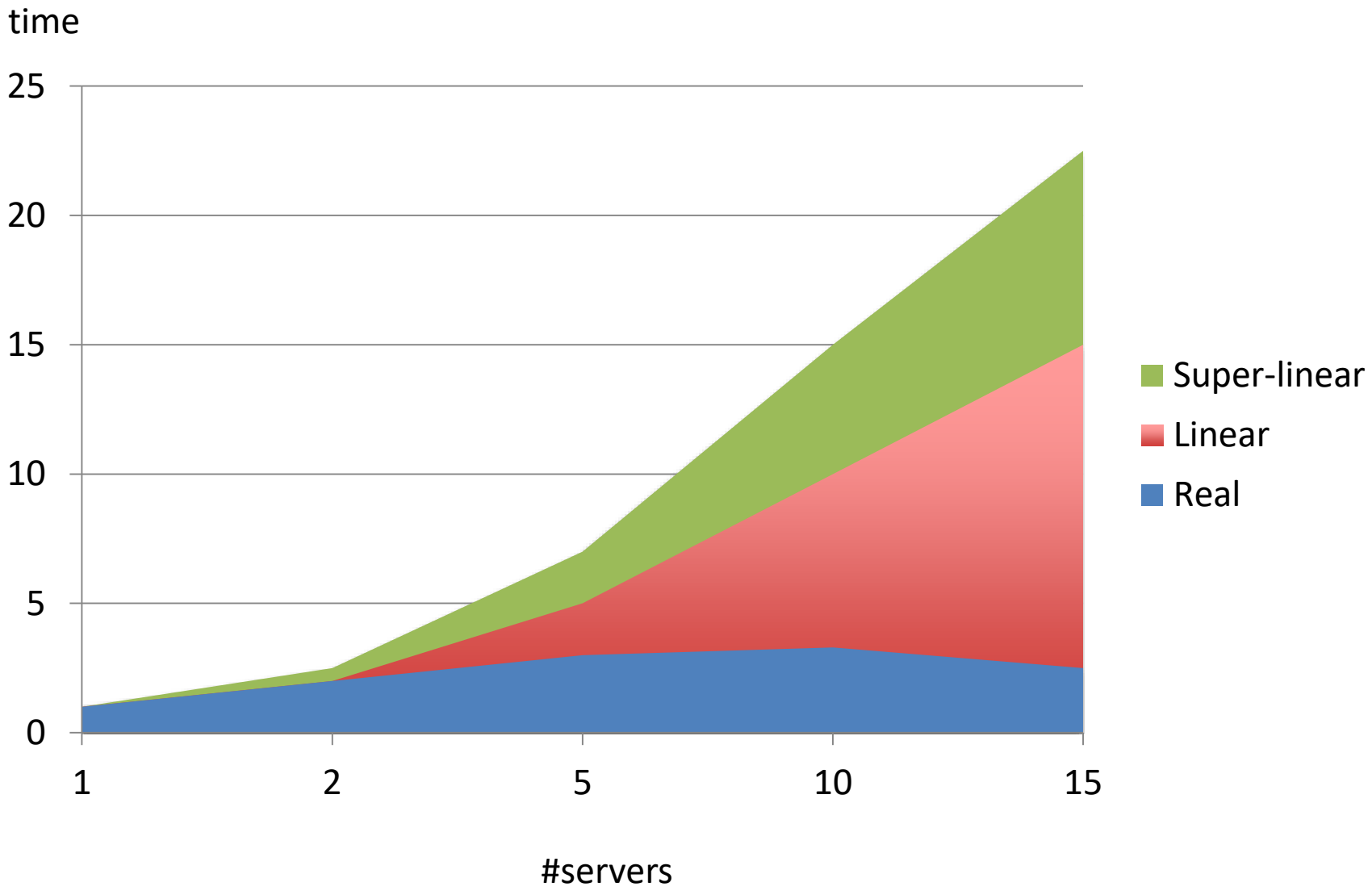
Why are response times long?

- 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?

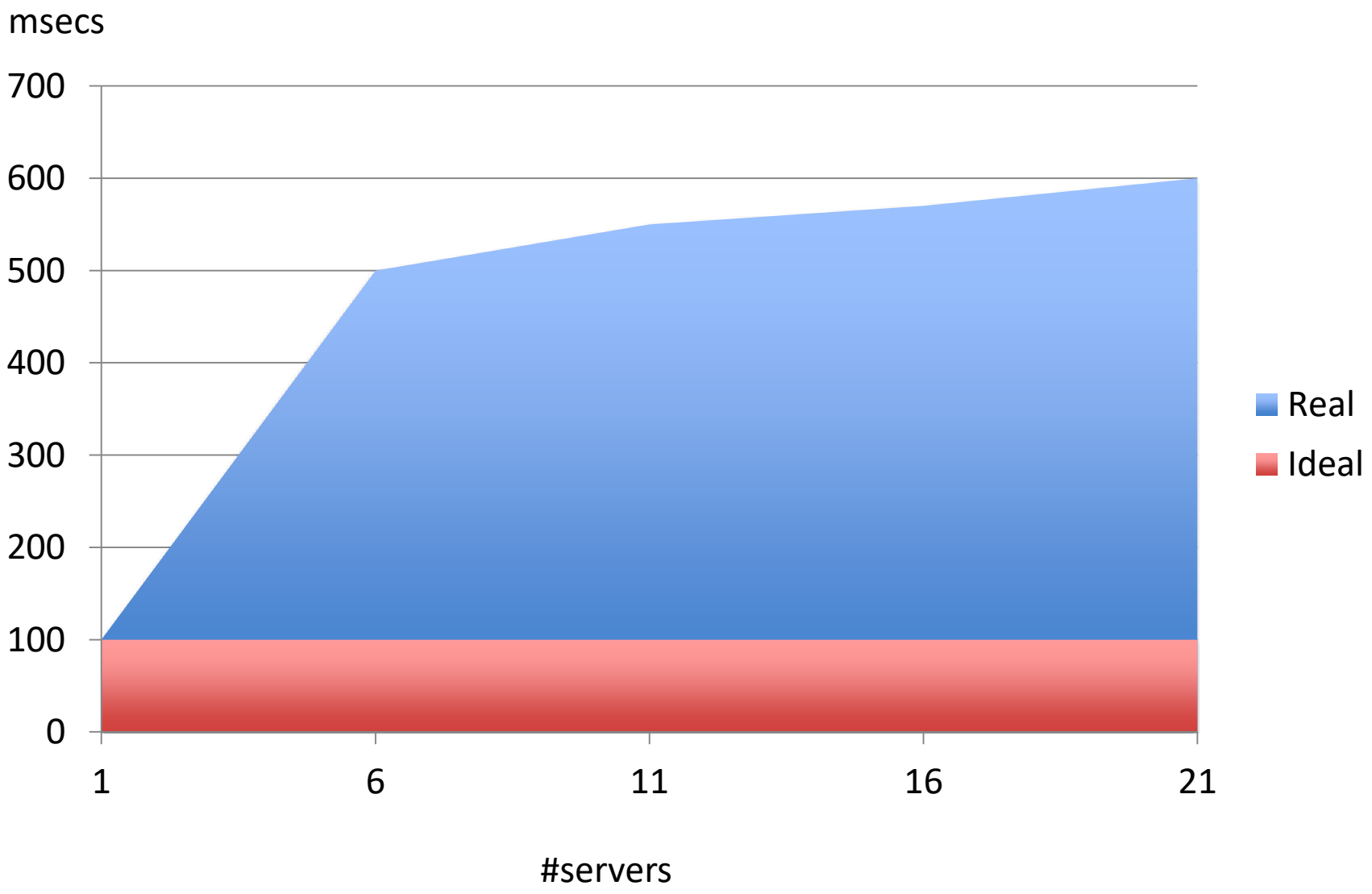
Speed Up



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

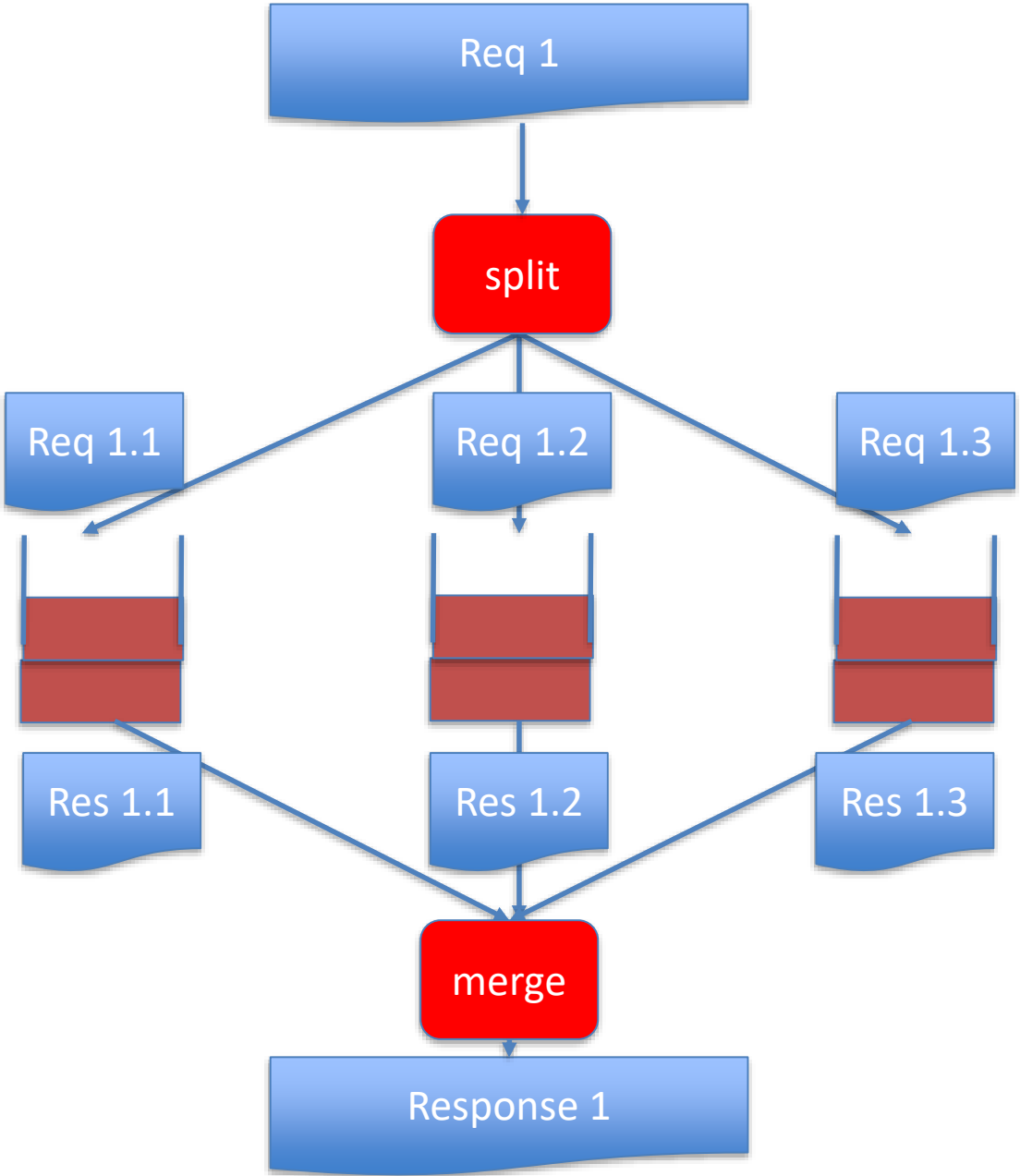
Scale Up Exp.: Response Time



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?

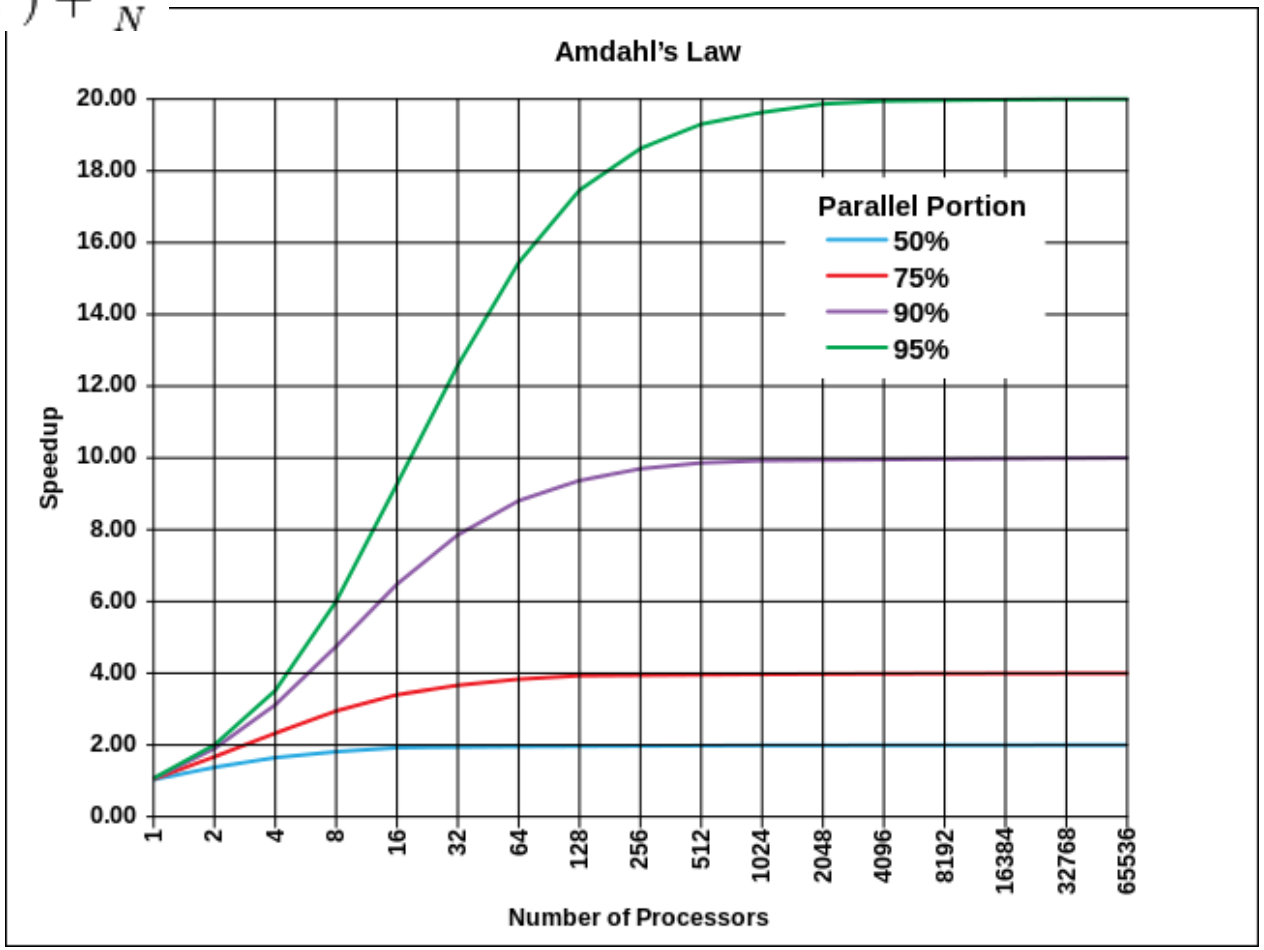


Why is speed-up sub-linear?

- 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

Hypothesis and questions

- The KVS you will build uses both partition and replication
- Formulate a number of questions on what you expect to see, develop a hypothesis around the behavior that is expected, explain the hypothesis, and write it all down
- Compare with what you get in reality when you implement and run the system for the first time.

Metrics and workloads

Metrics and Workloads

- Defining more terms
  - Workload
  - Parameters
  - …
- Example Benchmarks
  - TPC-H, etc.
  - Learn more metrics

Ingredients of an Experiment (rev.)

- System(s) Under Test
  - The (real) systems we would like to explore
- Workload(s) = User model
  - Typical behavior of users / clients of the system
- Parameters
  - The „it depends“ part of the answer to a perf. question
  - System parameters vs. Workload parameters
- Test database(s)
  - For database workloads
- Metrics
  - Defining what „better“ means: speed, cost, availability, ...

System under Test

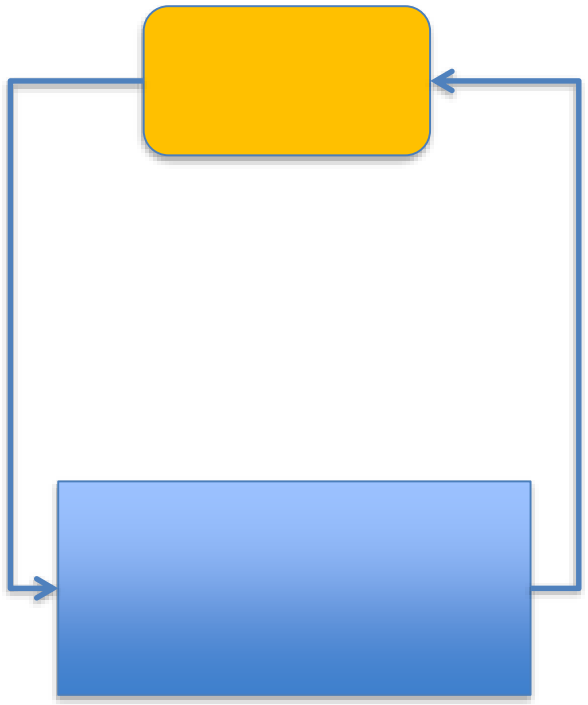
- Characterized by its API (services)
  - set of functions with parameters and result types
- Characterized by a set of parameters
  - Hardware characteristics
    - E.g., network bandwidth, number of cores, ...
  - Software characteristics
    - E.g., consistency level for a database system
- Observable outcomes
  - Dropped requests, latency, system utilization, ...
  - (results of requests / API calls)

Workload

- A sequence of requests (i.e., API/service calls)
  - Including parameter settings of calls
  - Possibly, correlation between requests (e.g., sessions)
  - Possibly, requests from different geographic locations
- Workload generators
  - Simulate a client which issues a sequence of requests
  - Specify a „thinking time“ or arrival rate of requests
  - Specify a distribution for parameter settings of requests
- Open vs. Closed System
  - Number of „active“ requests is a constant or bound
  - Closed system = fixed #clients, each client 0,1 pending req.
  - Warning: Often model a closed system without knowing!

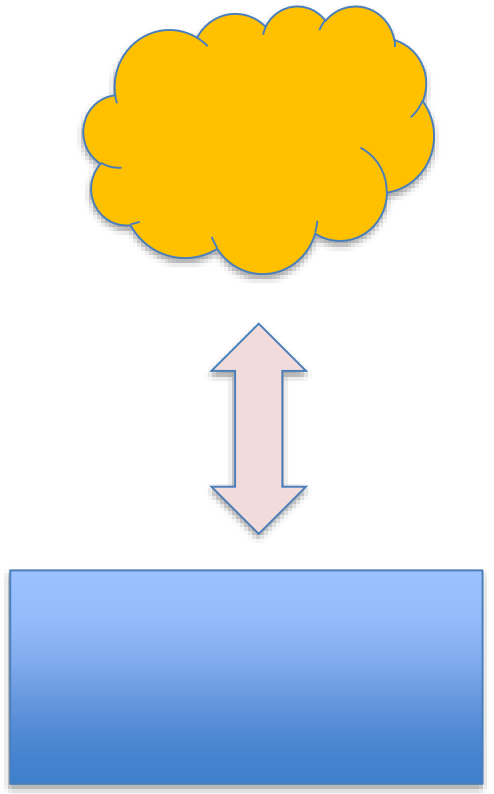
Closed system

- Load comes from a limited set of clients
- Clients wait for response before sending next request
- Load is self-adjusting
- System tends to stability
- Example: database with local clients



Open system

- Load comes from a potentially unlimited set of clients
- Load is not limited by clients waiting
- Load is not self-adjusting (load keeps coming even if SUT stops)
- Tests system's stability
- Example: web server



Parameters

- Many system and workload parameters
  - E.g., size of cache, locality of requests, ...
- Challenge is to find the ones that matter
  1. understanding the system + common sense
  2. Compute the standard deviation of metric(s) when varying a parameter
     - if low, the parameter is not significant
     - if high, the parameter is significant
     - important are parameters which generate „cross-over points“ between System A and B when varied.
     - Careful about correlations: vary combinations of params

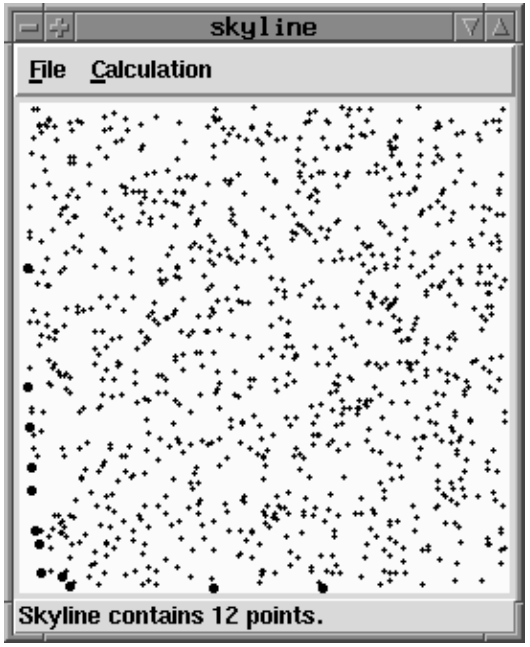
Test Database

- Many systems involve „state“
  - Behavior depends on state of database
  - E.g., long response times for big databases
- Database is a „workload parameter“
  - But very complex
  - And with complex implications
- Critical decisions
  - Distribution of values in the database
  - Size of database (performance when generating DB)
  - Ref.: J. Gray et al.: SIGMOD 1994.

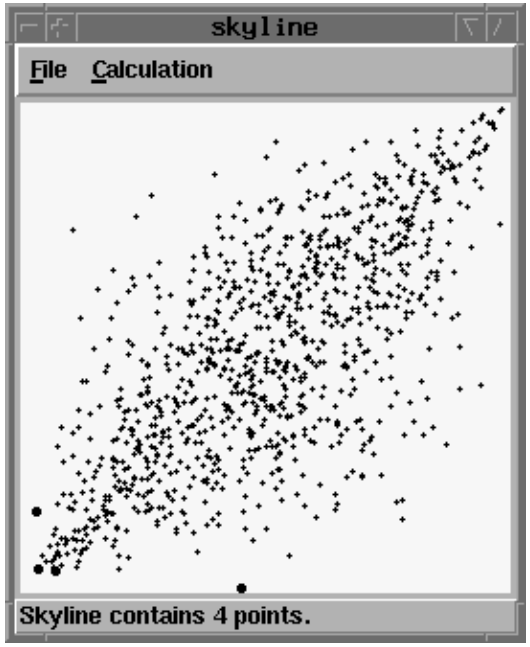
Popular Distributions

- Uniform
  - Choose a range of values
  - Each value of range is chosen with the same prob.
- Zipf (self-similarity)
  - Frequency of value is inverse proportional to rank
  - F(V[1]) ~ 2 x F(V[2]) ~ 4x F(V[4]) ...
  - Skew can be controlled by a parameter ζ
    - Default: $\zeta=1$; uniform: $\zeta=0$; high ζ corresponds to high skew
- Independent vs. Correlations
  - In reality, the values of 2 (or more) dim. are correlated
  - E.g., people who are good in math are good in physics
  - E.g., a car which is good in speed is bad in price

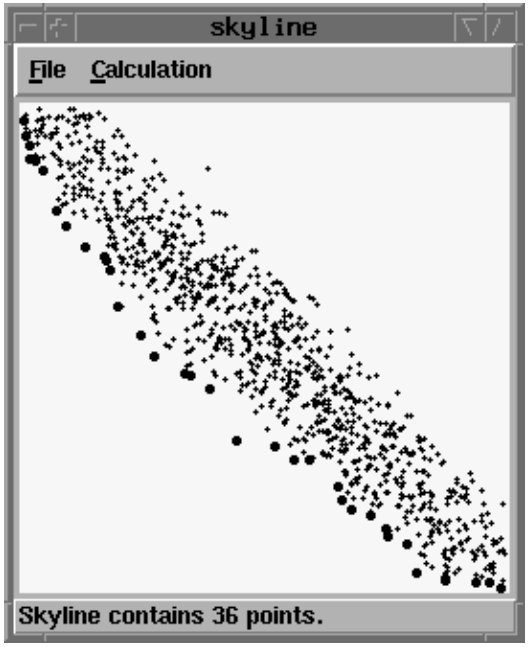
Multi-dimensional Distributions



Independent



Correlated



Anti-correlated

Ref.: Börszönyi et al.: „The Skyline Operator“, ICDE 2001.

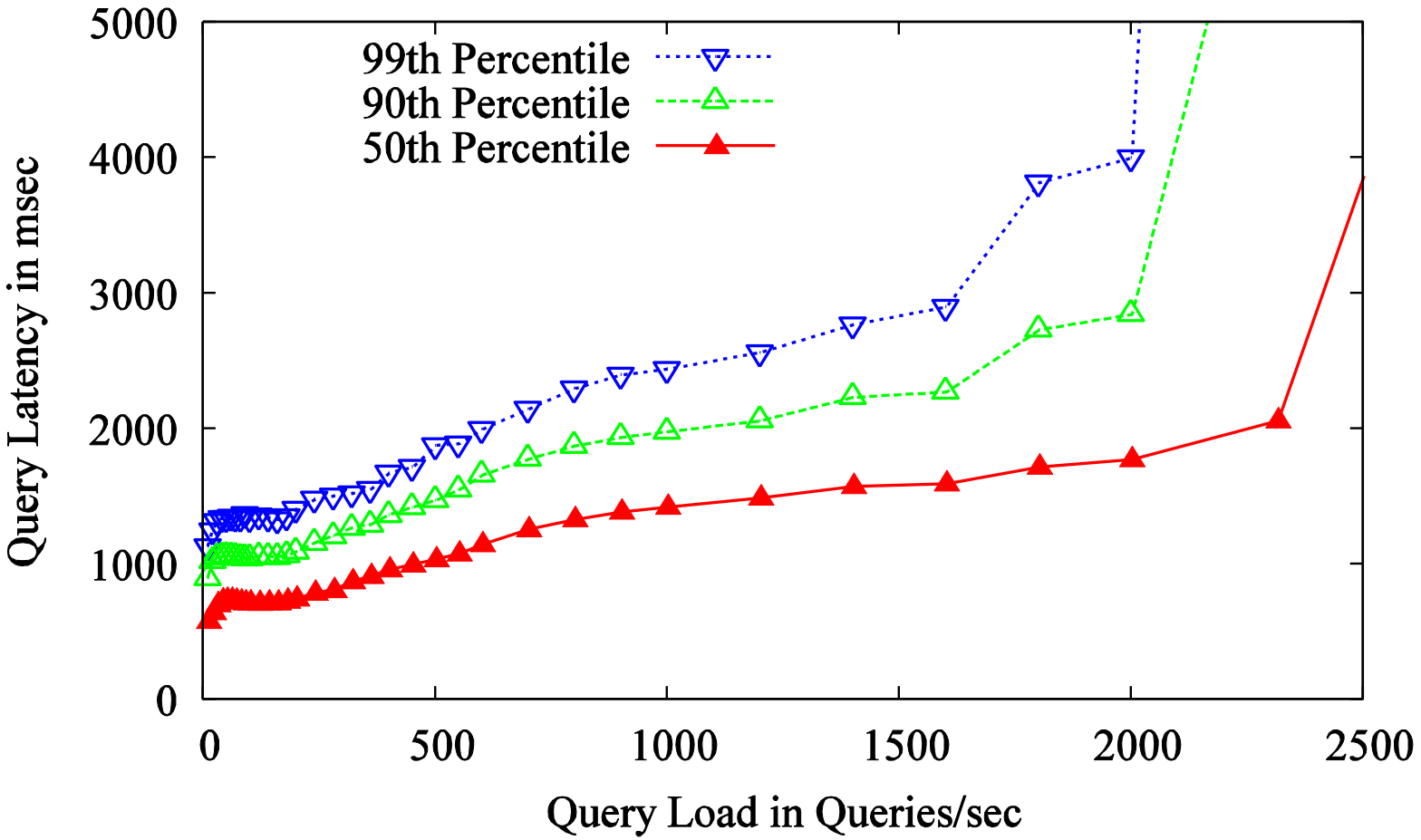
Metrics

- Performance; e.g.,
  - Throughput (successful requests per second)
  - Bandwidth (bits per second)
  - Latency / Response Time
- Cost; e.g.,
  - Cost per request
  - Investment
  - Fix cost
- Availability; e.g.,
  - Yearly downtime of a single client vs. whole system
  - % dropped requests (or packets)

Metrics

- How to aggregate millions of measurements
  - classic: median + standard deviation
  - Why is median better than average?
  - Why is standard deviation so important?
- Percentiles (quantiles)
  - V = Xth percentile if X% of measurments are < V
  - Max ~ 100th percentile; Min ~ 0th percentile
  - Median ~ 50th percentile
  - Percentiles good fit for Service Level Agreements
- Mode: Most frequent (probable) value
  - When is the mode the best metric? (Give an example)

Percentile Example



Amazon Example (~2004)

- Amazon lost about 1% of shopping baskets
  - Acceptable because incremental cost of IT infrastructure to secure all shopping baskets much higher than 1% of the revenue
- Some day, somebody discovered that they lost the *largest* 1% of the shopping baskets
  - Not okay because those are the premium customers and they never come back
  - Result in much more than 1% of the revenue
- Be careful with correlations within results!!!

How to improve performance?

- Find bottleneck
- Throw additional resources at the bottleneck
- Find the new bottleneck
- Throw additional resources at the bottleneck
- …

Reading

- Read Chapter 4, 5, and 6 in the text book