Advanced Systems Lab

G. Alonso, D. Kossmann
Systems Group
http://www.systems.ethz.ch

Reading

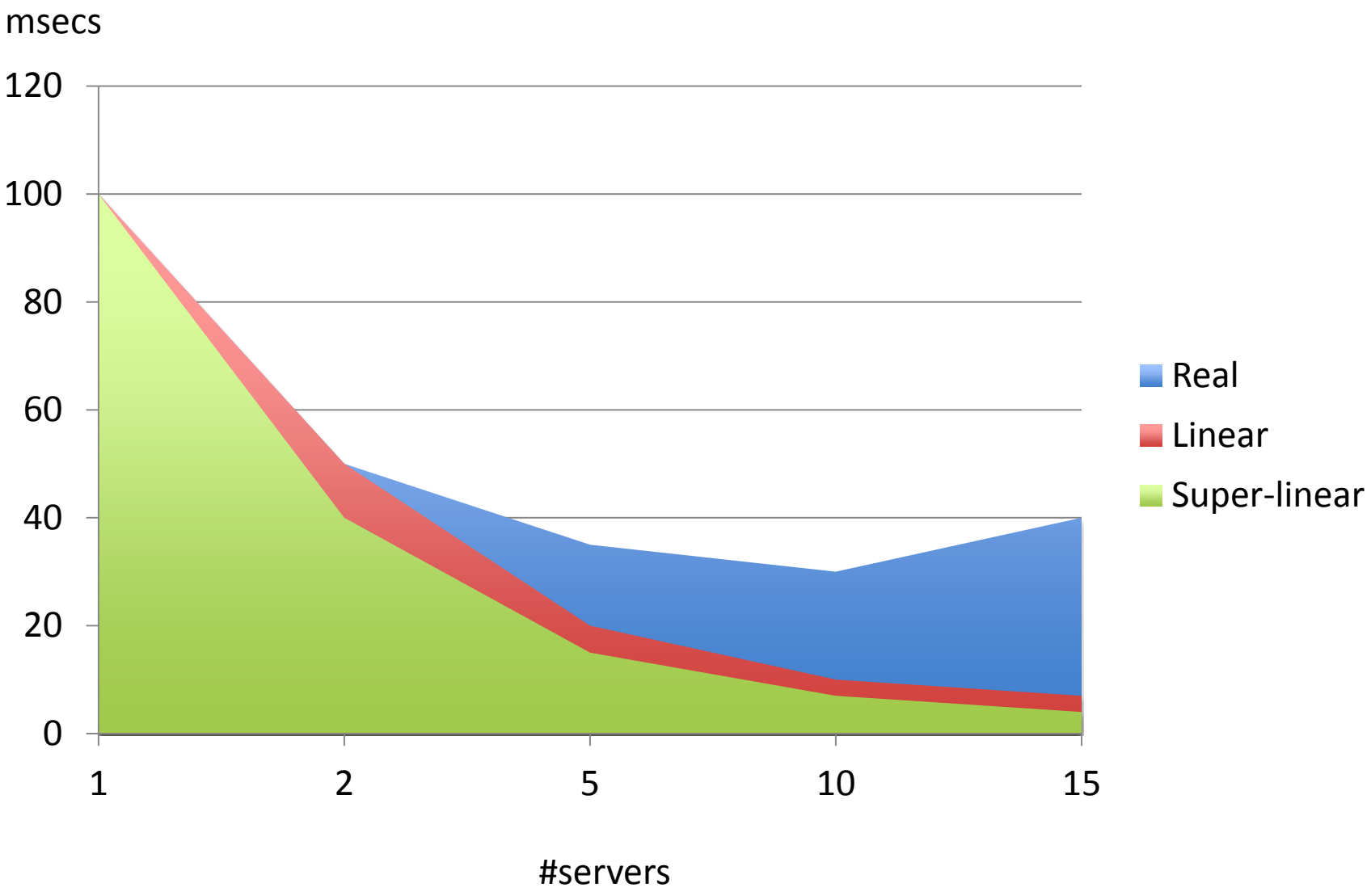
- Read Chapter 4, 5, and 6 in the text book

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

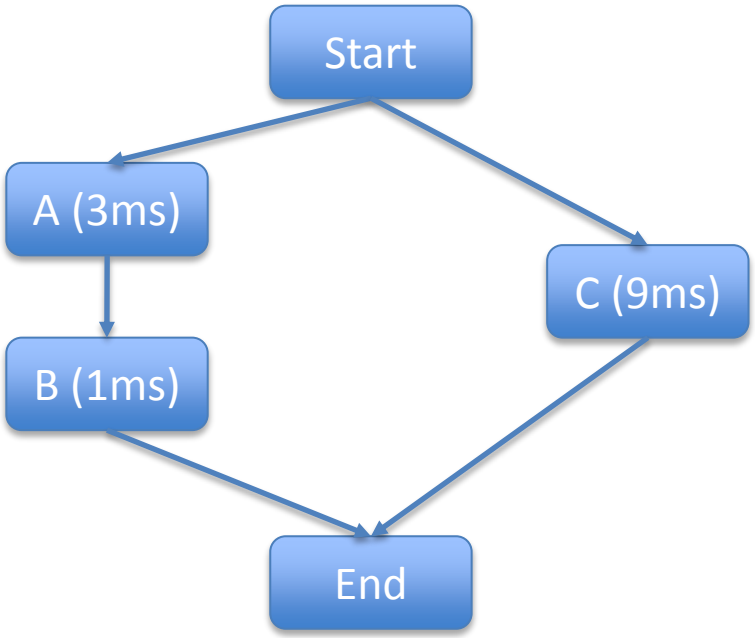


Why are response times long?

- Because operations take long
  - cannot travel faster than light
  - delays even in „single-user“ mode
  - possibly, „parallelize“ long-running operations
    - „intra-request parallelism“
- 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
    - „inter-request parallelism“

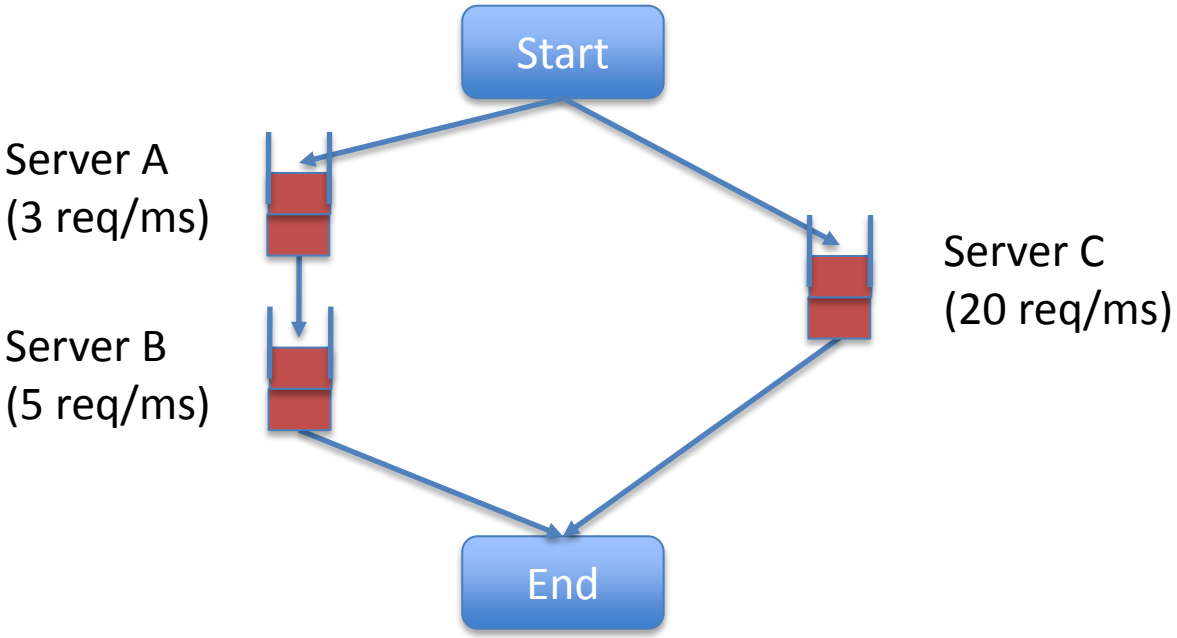
Critical Path

- Directed Graph of Tasks and Dependencies
  - response time = max { length of path }
  - assumptions: no resource contention, no pipelining, ...
- Which tasks would you try to optimize here?



Queueing Network Models

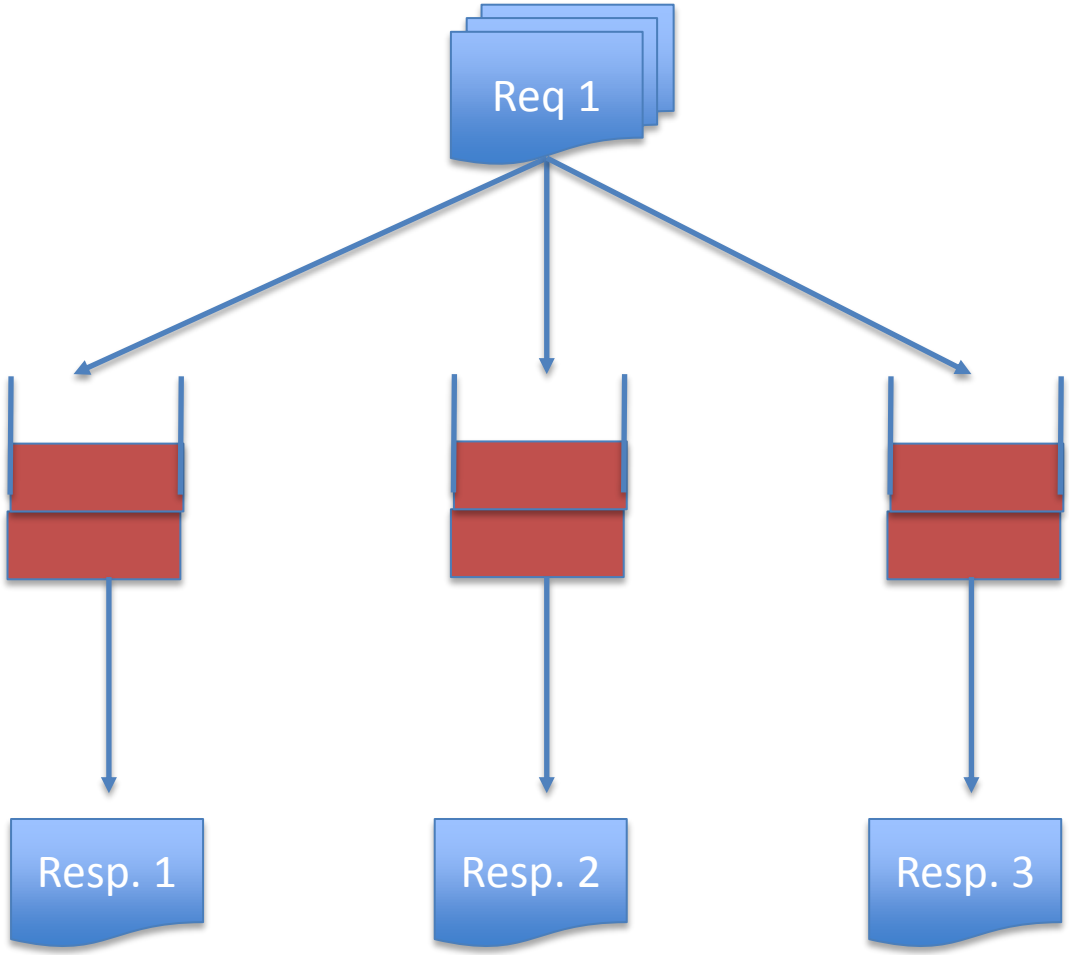
- Graph of resources and flow of requests
- Bottleneck=Resource defines tput of whole system
  - (analysis techniques described later in the course)



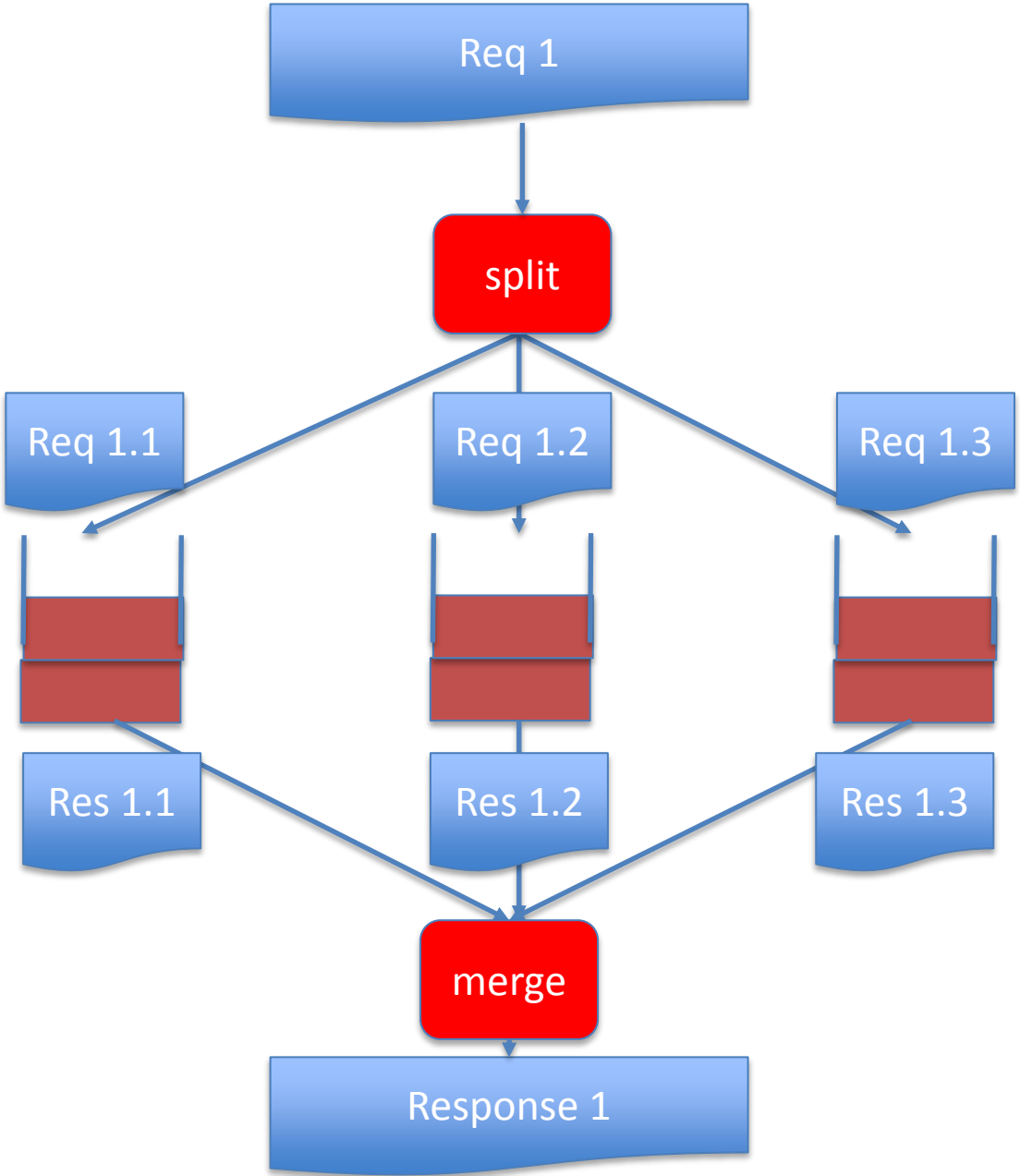
Forms of Parallelism

- Inter-request Parallelism
  - several requests handled at the same time
  - principle: replicate resources
  - e.g., ATMs
- (Independent) Intra-request Parallelism
  - principle: divide & conquer
  - e.g., print pieces of document on several printers
- Pipelining
  - each „item“ is processed by several resources
  - process „items“ at different resources in parallel
  - can lead to both inter- & intra-request parallelism

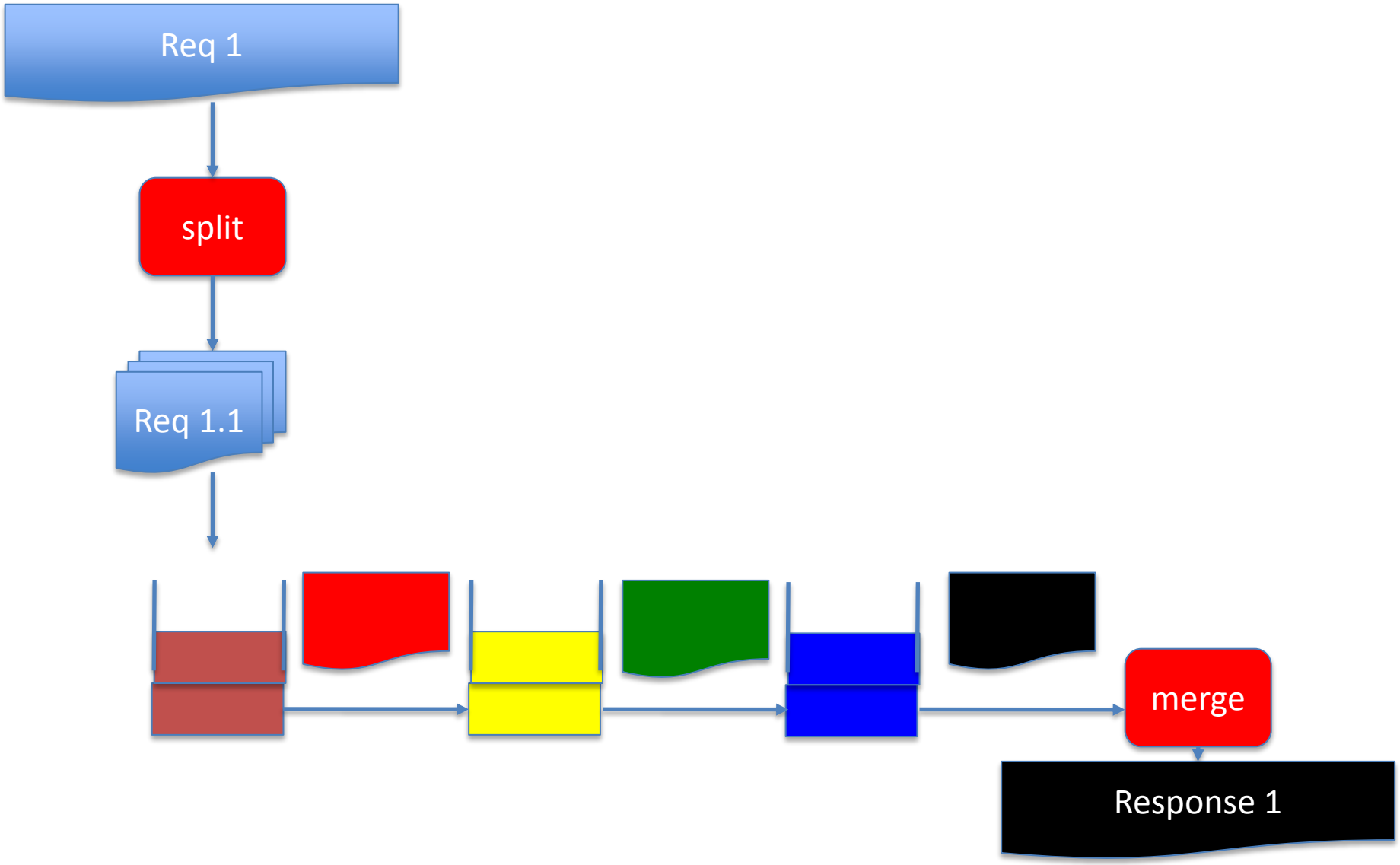
Inter-request Parallelism



Intra-request Parallelism



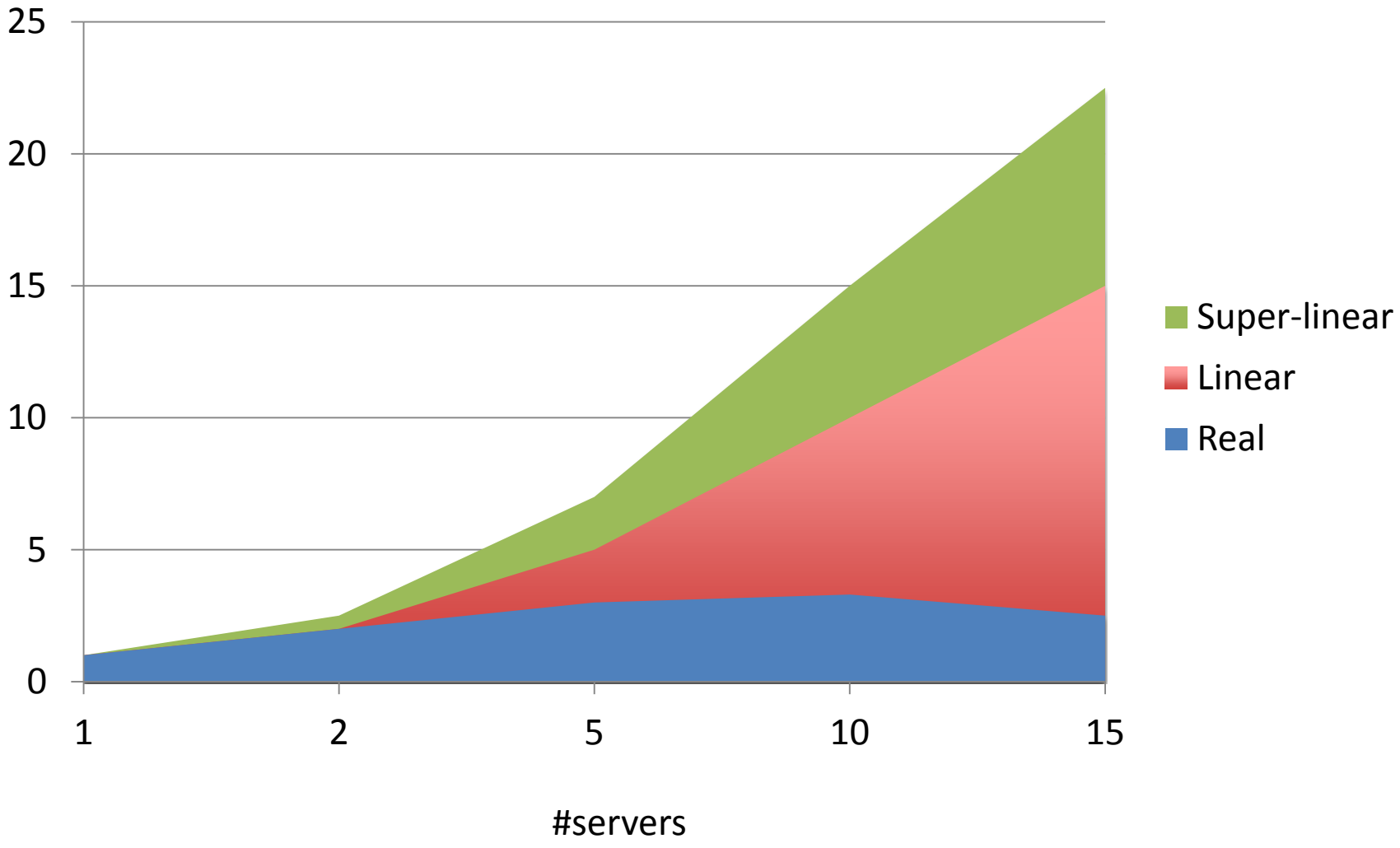
Pipelining (Intra-request)



Speed-up

- Metric for intra-request parallelization
- Goal: test ability of SUT to reduce response time
  - 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?

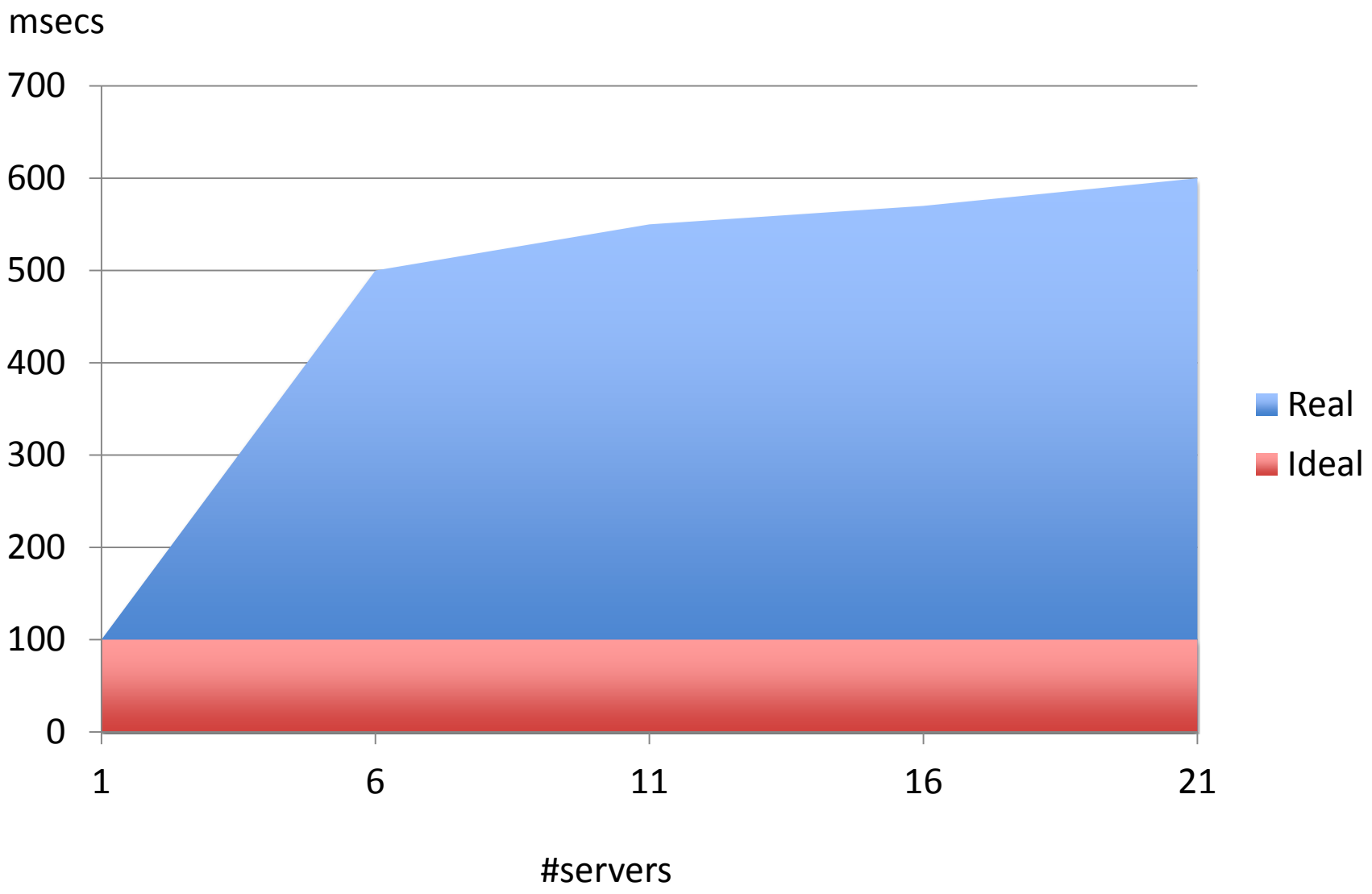
Speed Up



Scale-up

- Test how SUT scales with size of the problem
  - measure response time with 1 server, unit problem
  - measure response time with N servers, N units problem
  - ScaleUp(N) = RT(1) / RT(N)
- Ideal
  - ScaleUp(N) is a constant function
  - Can you imagine super scale-up?

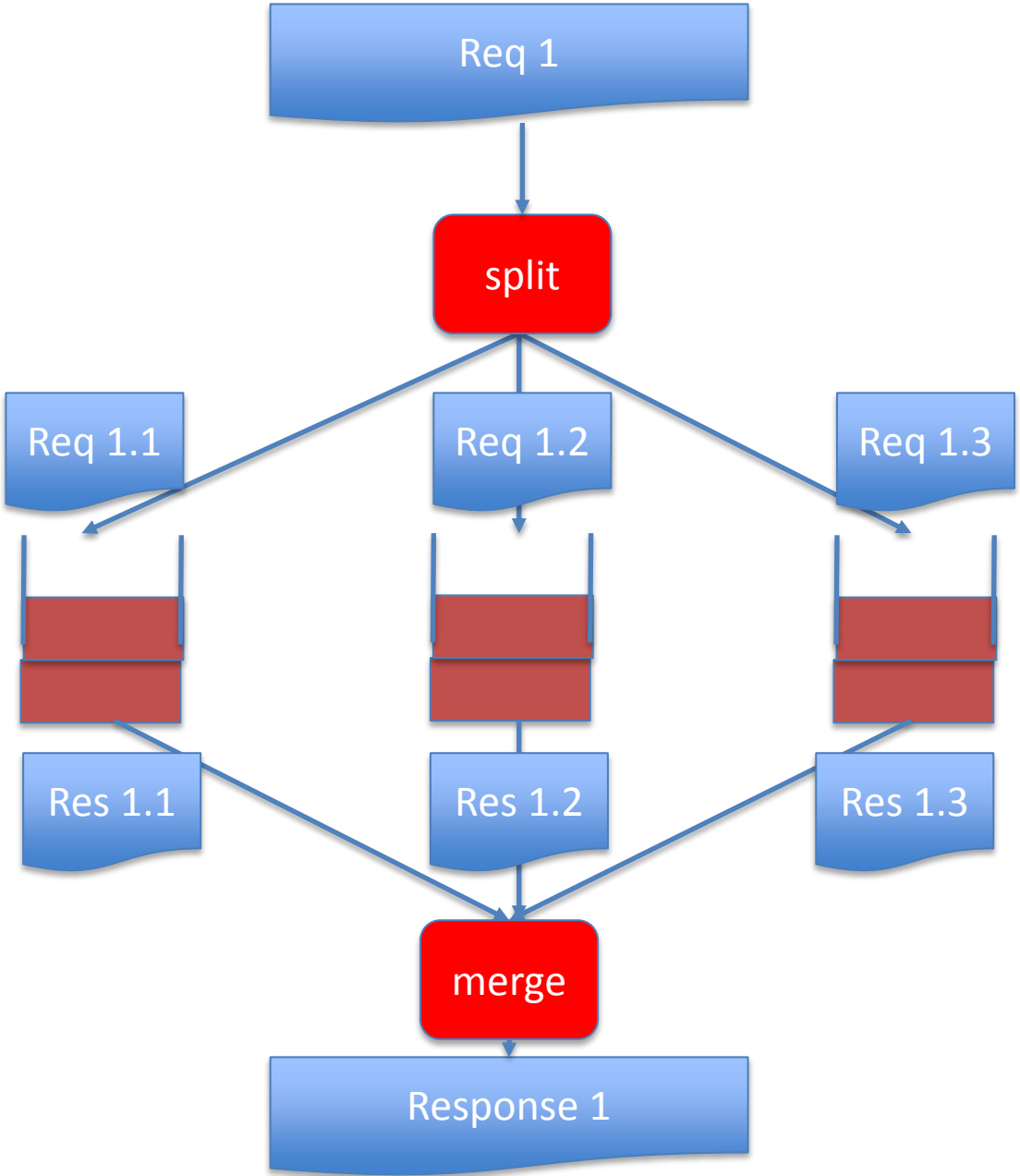
Scale Up Exp.: Response Time



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)
- Scale-out and down in Cloud Computing
  - the ability of a system to adapt to changes in load
  - often measured in $ (or at least involving cost)

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

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

Metrics and workloads

Metrics and Workloads

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

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

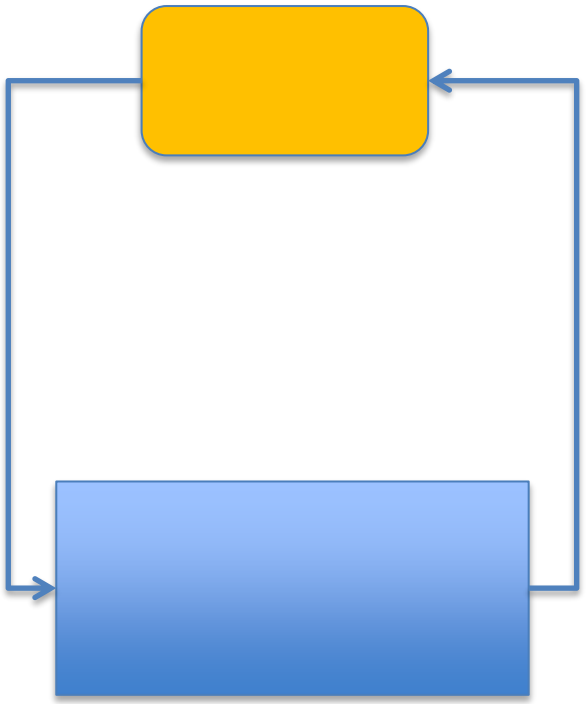
- Characterized by its API (services)
  - set of functions with parameters and result types
- Characterized by a set of parameters
  - Hardware characteristics
    - E.g., network bandwidth, number of cores, ...
  - Software characteristics
    - E.g., consistency level for a database system
- Observable 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!

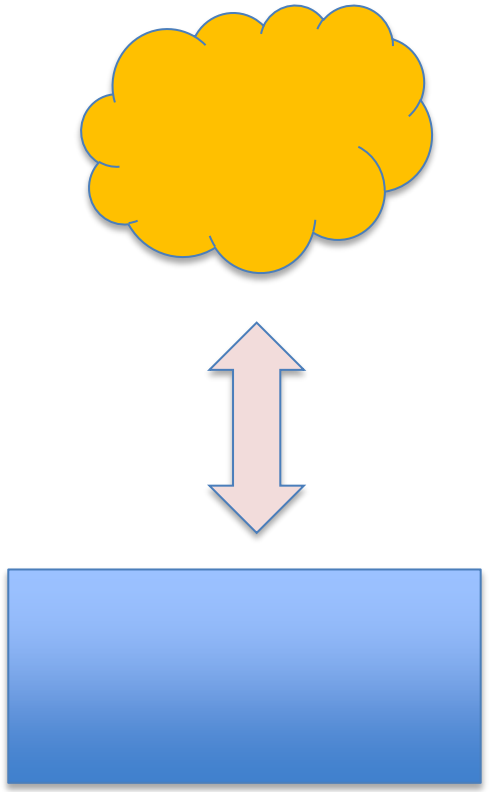
Closed system

- 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

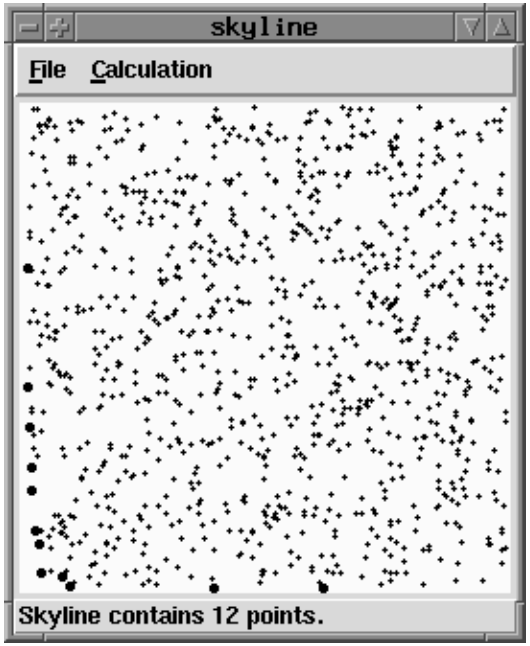
Test Database

- 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.

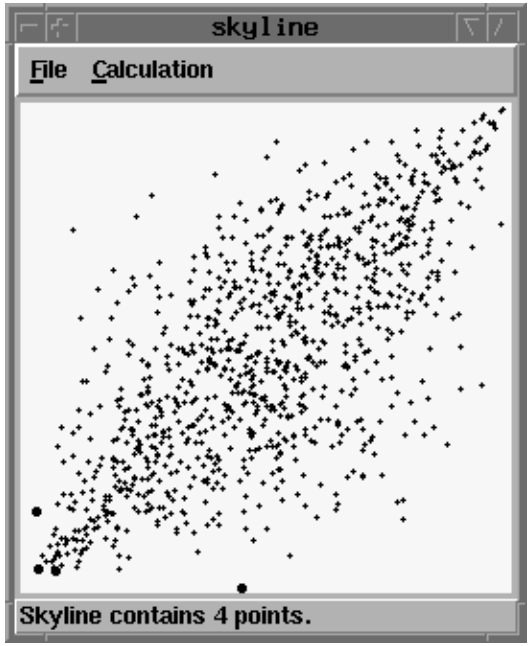
Popular Distributions

- 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

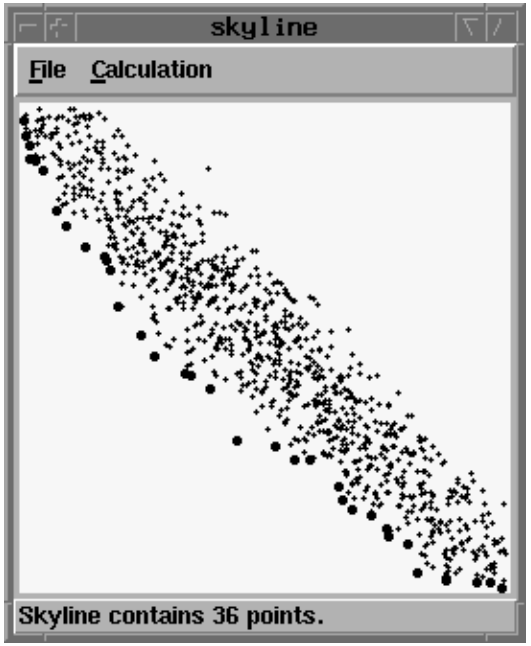
Multi-dimensional Distributions



Independent



Correlated



Anti-correlated

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

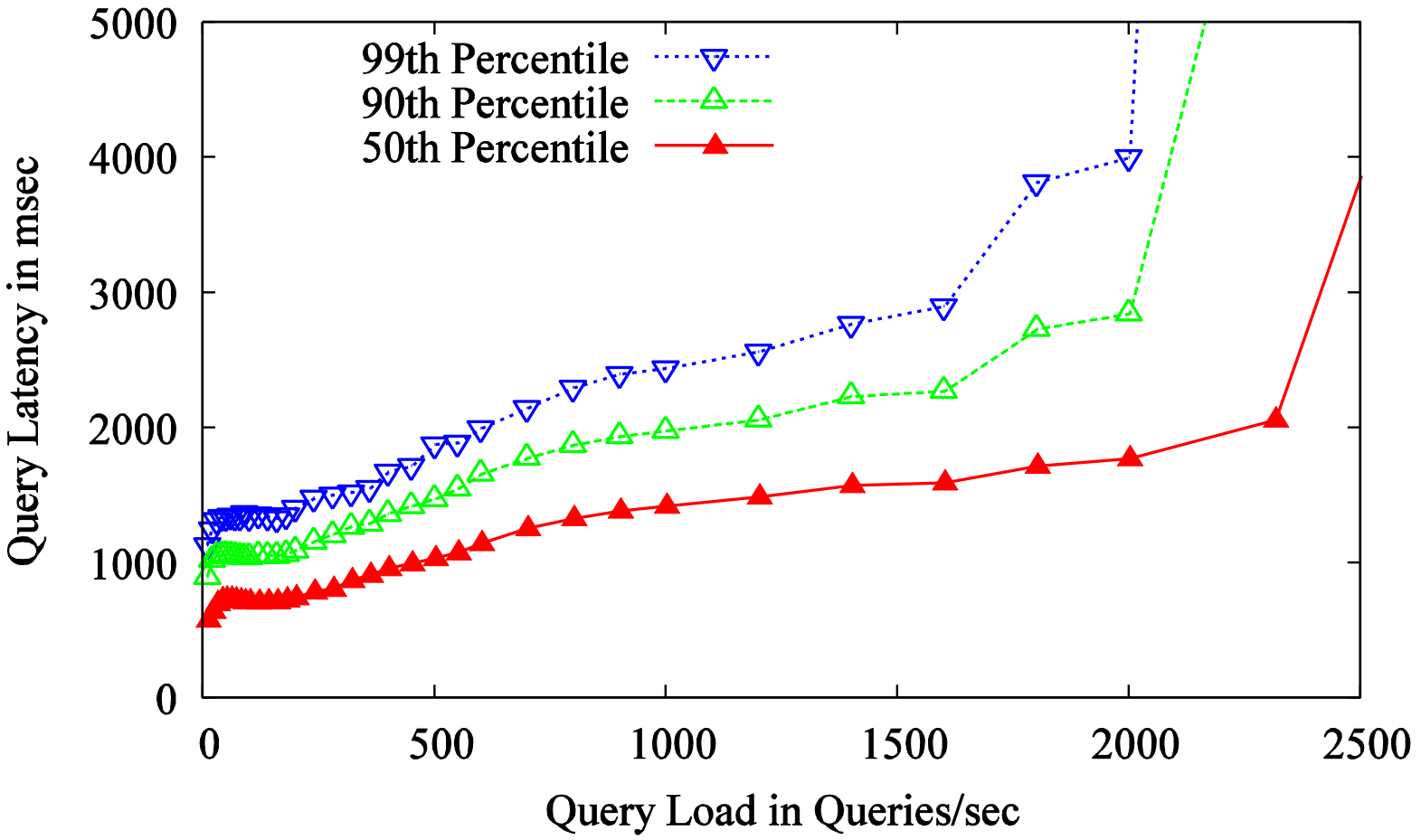
Metrics

- 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!!!

Where does all this come from?

- Real workloads
  - Use traces from existing (production) system
  - Use real databases from production system
- Synthetic workloads
  - Use standard benchmark
  - Invent something yourself
- Tradeoffs
  - Real workload is always relevant
  - Synthetic workload good to study „corner cases“
    - Makes it possible to vary all „workload parameters“
  - If possible, use both!

Benchmarks

- Specify the whole experiment **except** SUT
  - Sometimes specify settings of „system parameters“
    - E.g., configure DBMS to run at isolation level 3
- Designed for „is System A better than B“ questions
  - Report one or two numbers as metrics only
  - Use complex formula to compute these numbers
  - Zero or one workload parameters only
  - Standardization and notaries to publish results
- Misused by research and industry
  - Implement only a subset
  - Invent new metrics and workload parameters
  - Violation of „system parameter“ settings and fineprint

Benchmarks: Good, bad, and ugly

- Good
  - Help define a field: give engineers a goal
  - Great for marketing and sales people
  - Even if misused, great tool for research and teaching
- Bad
  - Benchmark wars are not productive
  - Misleading results – huge damage if irrelevant
- Ugly
  - Expensive to be compliant (legal fineprint)
  - Irreproducible results due to complex configurations
  - Vendors have complex license agreements (DeWitt clause)
  - Single number result favors „elephants“
    - Difficult to demonstrate advantages in the „niche“

Benchmarks

- Conjecture

  „Benchmarks are a series of tests in order to obtain **prearranged results** not available on competitive systems.“ (S. Kelly-Bootle)

- Corollary

  „I only trust statistics that I have invented myself.“ (folklore)

Example Benchmarks

- CPU
  - E.g., „g++“, Ackermann, SPECint
- Databases (www.tpc.org)
  - E.g., TPC-C, TPC-E, TPC-H, TPC-W, ...
- Parallel Systems
  - NAS Parallel Benchmark, Splash-2
- Other
  - E.g., CloudStone, LinearRoad
- Microbenchmarks
  - E.g., LMBench

SPECint

- Goal: Study CPU speed of different hardware
- SPEC = Standard Performance Eval. Corp.
  - www.spec.org
- Long history of CPU benchmarks
  - First version CPU92
  - Current version: SPECint2006
- SPECint2006 involves 12 tests (all in C/C++)
  - perlbench, gcc, bzip2, ..., xalancbmk
- Metrics
  - Compare running time to „reference machine“
    - E.g., 2000secs vs. 8000secs for gcc gives score of 4
  - Overall score = geometric mean of all 12 scores

SPECint Results

- Visit:

http://www.spec.org/cpu2006/results/cint2006.html

TPC-H Benchmark

- Goal: Evaluate DBMS + hardware for OLAP
  - Find the „fastest“ system for a given DB size
  - Find the best „speed / $“ system for a given size
  - See to which DB sizes the systems scale
- TPC-H models a company
  - Orders, Lineitems, Customers, Products, Regions, ...
- TPC-H specifies the following components
  - Dbgen: DB generator with different scaling factors
  - Scaling factor of DB is the only workload parameter
  - Mix of 22 queries and 2 update functions
  - Execution instructions and metrics

TPC-H Fineprint

- Physical Design
  - E.g., you must not vertically partition DB
  - (many results violate that, i.e., all column stores)
- Execution rules
  - Specify exactly how to execute queries and updates
  - Specifiy exactly which SQL variants are allowed
- Results
  - Specifies exactly how to compute metrics and how to publish results
- The specification is about 150 pages long (!)

TPC-H Results

- Visit:

http://www.tpc.org/tpch/results/tpch_price_perf_results.asp

Microbenchmarks

- Goal: „Understand full behavior of a system“
  - Not good for decision „System A vs. System B“
  - Good for component tests and unit tests
- Design Principles
  - Many small and simple experiments, many workload parameters
  - Report all results (rather than one big number)
  - Each experiment tests a different feature (service)
    - E.g., table scan, index scan, join for DB
    - E.g., specific function calls, representative parameter settings
  - Isolate this feature as much as possible
  - Design requires knowledge of internals of SUT
    - Designed for a specific study, benchmark not reusable

How to improve performance?

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