Qualities and Constraints in IT Architecture

Non-Functional Requirements

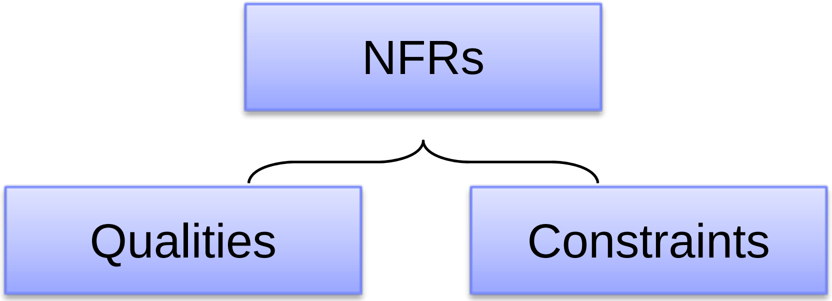
Examples: Availability and Performance

Dr. Marcel Schlatter
IBM Distinguished Engineer
Member of the IBM Academy of Technology
marcel.schlatter@ch.ibm.com

Non-functional requirements (or NFRs) define the desirable qualities of a system and the constraints within which the system must be built

- ***Qualities*** define the properties and characteristics which the delivered system should demonstrate

- ***Constraints*** are the limitations, standards and environmental factors which must be taken into account in the solution



Exercise – List Typical IT Project Constraints and NFRs

- List 5-10 types of constraints and qualities you would expect a typical medium to large IT project to have

- Constraints
  - Business
  - Technical

- Qualities
  - Runtime
  - Non-Runtime

Constraints

- **The business aspects of the project, customer's business environment or IT organization that influence the architecture**
- **The technical environment and prevailing standards that the system, and the project, need to operate within**

Business

- Regulatory
- Organisational
- Risk Willingness
- Marketplace factors
- Schedule & Budget

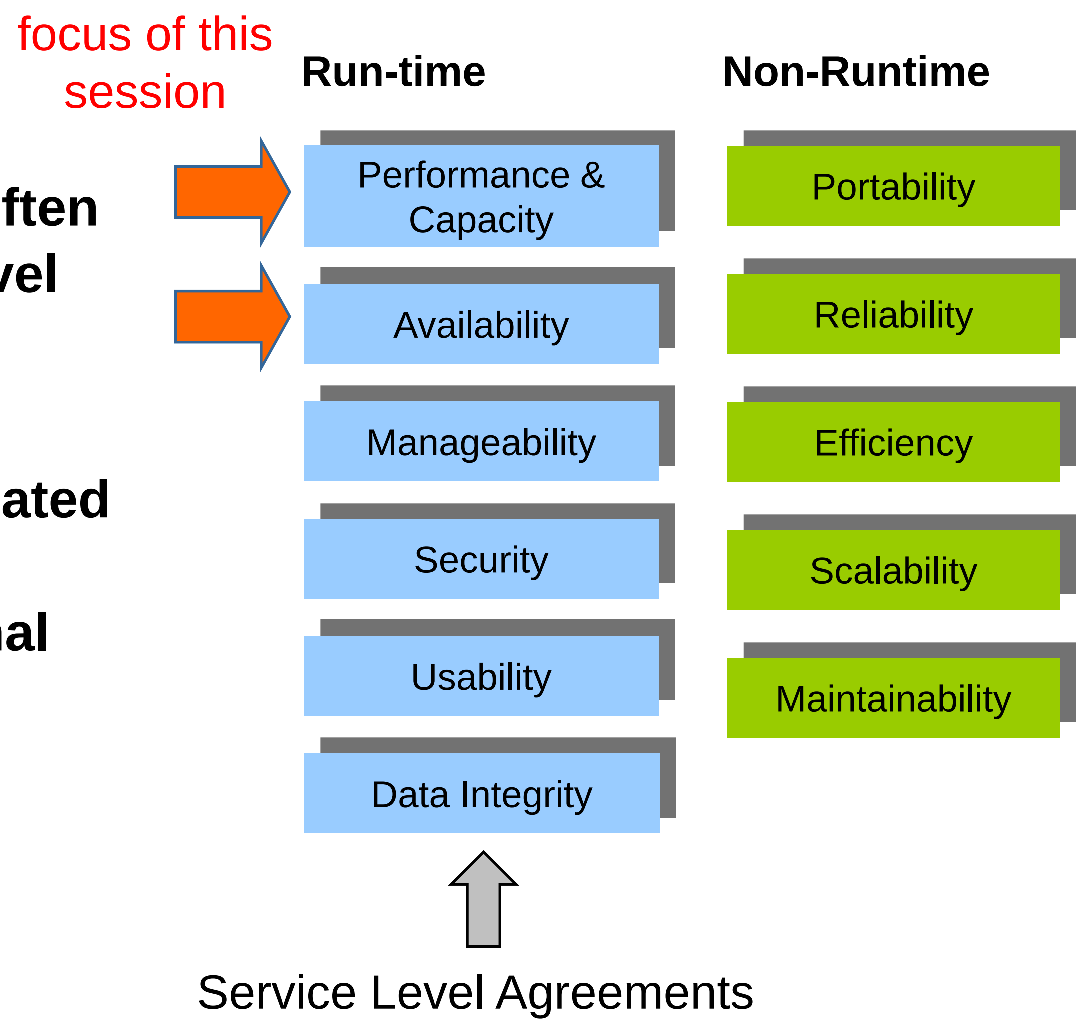
Technical

- Legacy Integration
- Development Skills
- Existing Infrastructure
- Technology State of the art
- IT Standards

Qualities

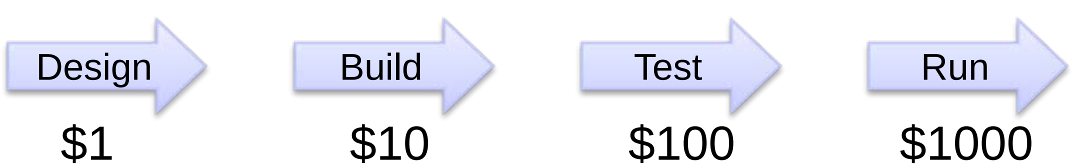
- **Runtime qualities are ‘measurable’ properties, often expressed as “Service Level Requirements”.**

- **Qualities might also be related to the development, maintenance, or operational concerns that are not expressed at runtime.**

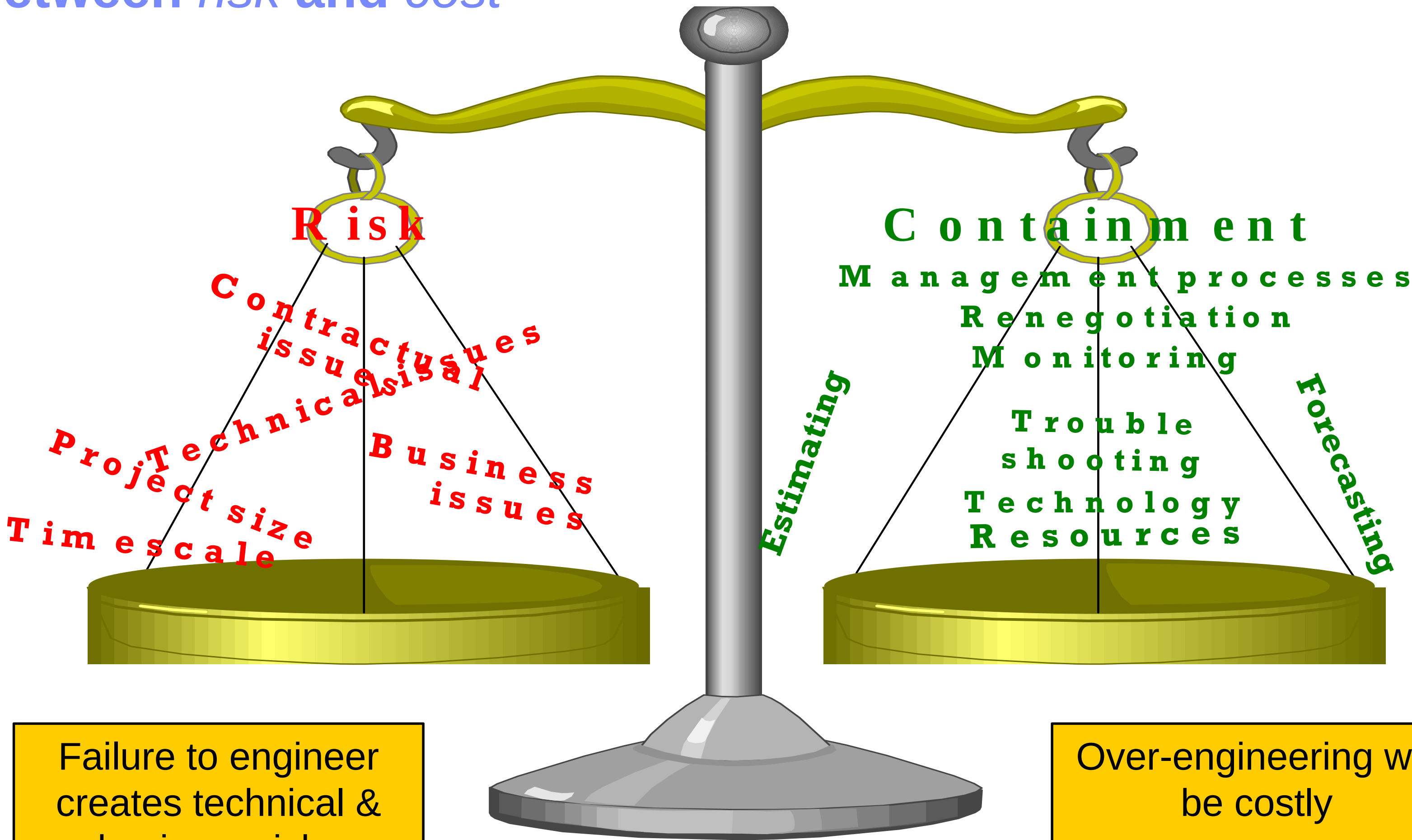


The best technique for reducing the risk of poor quality of service is to consider the qualities from the start

- Build ‘quality’ into the solution starting with early design
  - Understand the risks to the project
  - Conduct quality of service engineering from the first elaboration of the architecture model
  - **Set guidelines for the developers (software & infrastructure)**
  - **Test the application/system at each major stage of development**
  - **Make sure that the live support teams will be able to manage quality**
- Fix it early, and save money and problems later …



However a BALANCE must be maintained between *risk* and *cost*



Failure to engineer creates technical & business risks

Over-engineering will be costly

Availability

The reality of Availability is that customers directly relate it to the End User experience

The Availability of a system is a measure of its readiness for usage

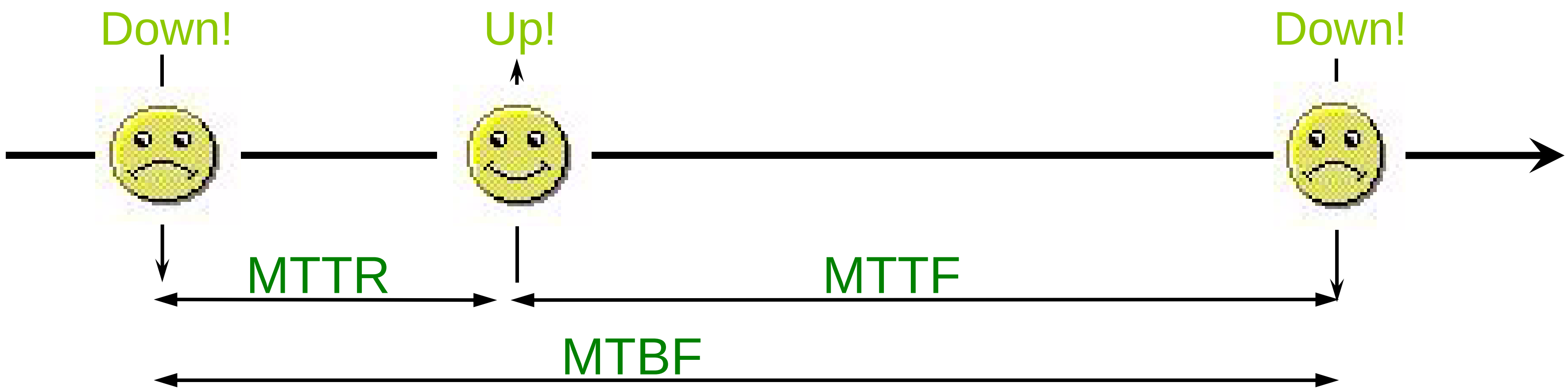
There are certain key terms that are used to define Availability-related concepts

Down

3.6 days / year

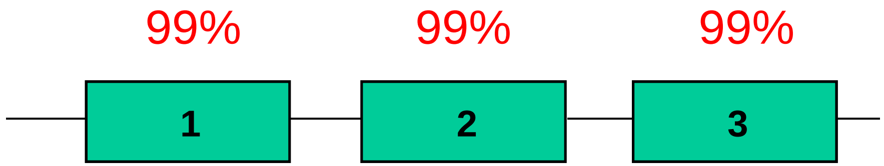
- **High Availability** is taken to mean a requirement for a system or service to be over 99% available – typically implies thorough design and may require redundant components
- **Disaster Recovery** means the recovery of essential services in the event of a major business disruption that has resulted from the occurrence of a disaster
- **Business Continuity** means the continued operation of business processes to a predetermined acceptable level in the event of a major business disruption
- **Unscheduled Outage** is a time period when the system is not ready for use and the users expect it to be. These are unplanned outages caused by ‘Random Events’
- **Scheduled Outage** is a time period when the system is not ready for use and the users do not expect it to be. These are planned outages driven by predefined events
- **Continuous Operations** is the requirement for perpetual operations 365 days per year 24 hours per day with perhaps very rare scheduled outages
- **Fault Tolerance** is that property of a component, sub-system or system that means that normal service continues even though a fault has occurred within the system
- **Reliability** is the probability that an item will perform its intended function for a specified interval under stated conditions
- **Maintainability** (or **Recoverability**) is the probability that using prescribed procedures and resources, an item can be retained in, or restored to, a specific condition within a given period

Key Availability terms – Mean Times …



- **Mean Time to Recover (MTTR)** is the typical time that it takes to recover (includes repair) a component, sub-system or a system.
- **Mean Time to Failure (MTTF)** is the mean time between successive failures of a given component, sub-system or system.
- **Mean Time between Failure (MTBF)** is the average time between successive failures of a given component, sub-system or system

One of the attributes of the design that should be understood for Availability Engineering is the effect of using components in series



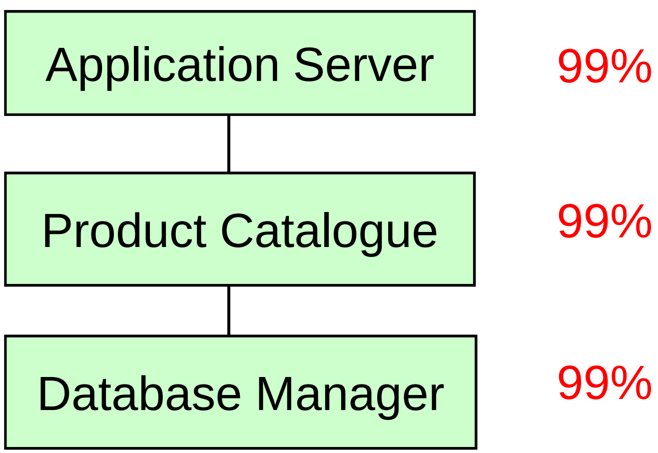
Down

10.8 days / year

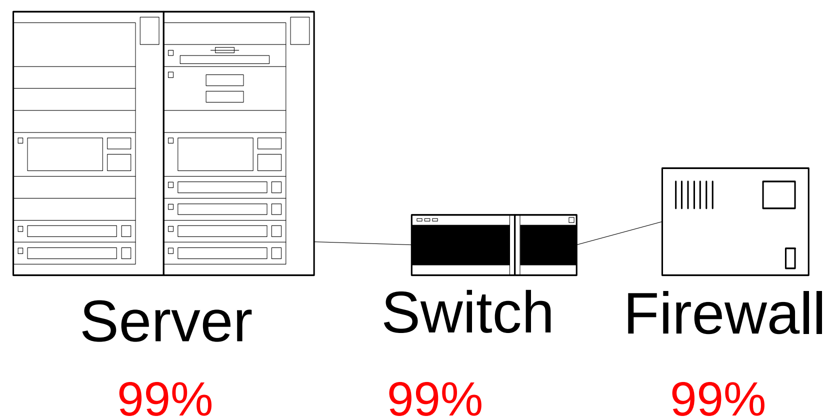
$$\text{Availability (A)} = A^1 \times A^2 \times A^3$$

- Components connected is a chain, relying on the previous component for availability
- The total availability is always lower than the availability of the weakest link

Functional

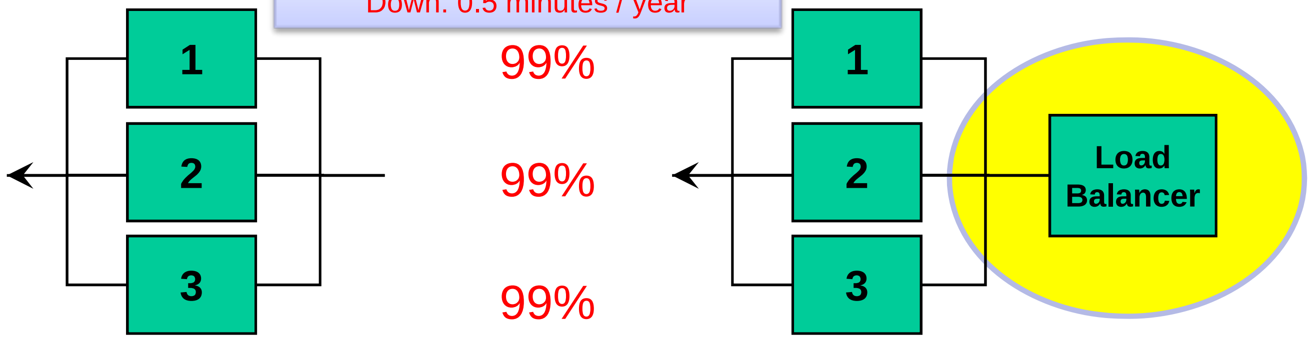


Operational



Another attribute of the design that should be understood for Availability Engineering is the effect of using components in parallel

Availability = 99.9999%
Down: 0.5 minutes / year

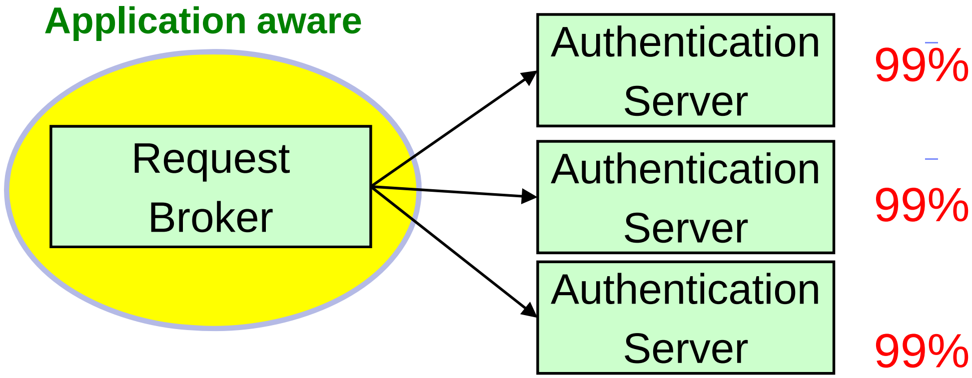


Availability = 1-[(1-A(1))x(1- A(2))x (1-A(3))]

- Component redundancy through duplication
- Total availability is higher than the availability of the individual links

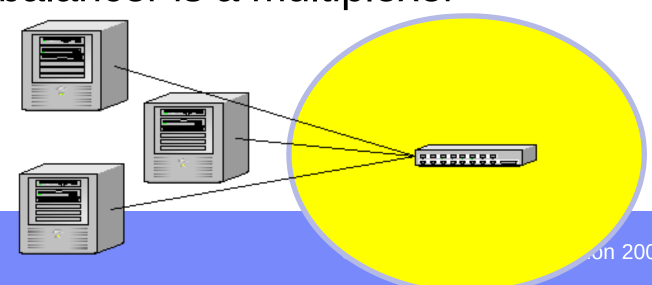
Functional

Application aware



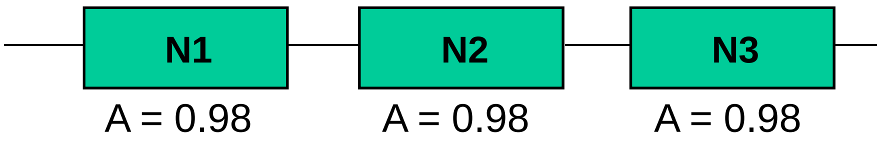
Operational

- Separate nodes all serving the same IP address
- Load balancer is a multiplexer

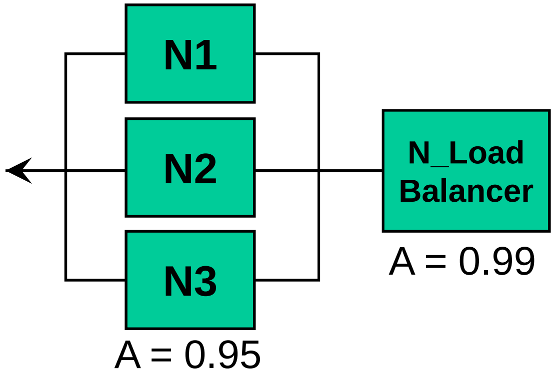


Exercise – Serial vs. Parallel Availability

- Q1. What is the overall availability of this serial structure of nodes?



- Q2. What is the overall availability of this combined structure of nodes?

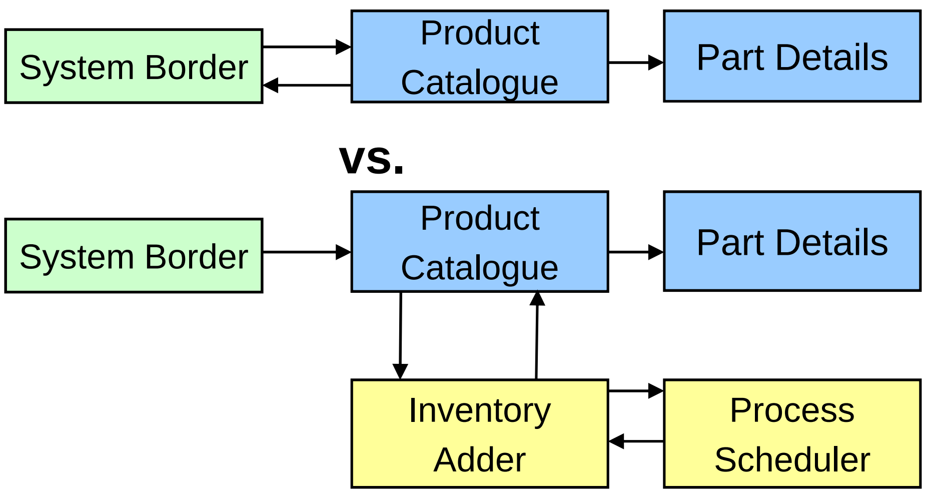


Separation of Concern is a technique that can be used to enable a loose coupling for components that provide critical services

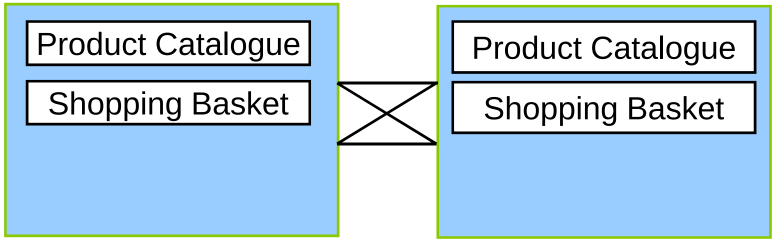
- The separation of components with regard to business importance and their availability characteristics

Functional

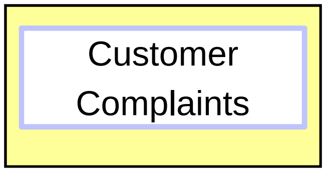
- Loose coupling of HA Components



Operational

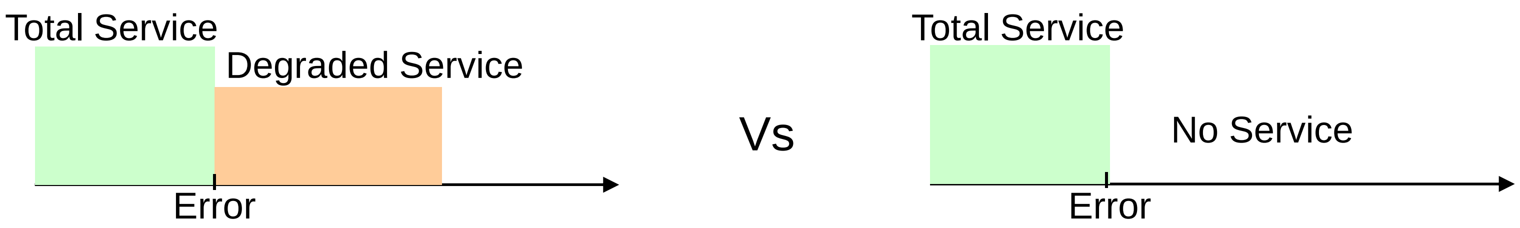


HA-focused Nodes



Non HA-focused Nodes

Fault Tolerance is a technique that can be used to enable the detection and correction of latent errors before they become effective



- Error Processing - Error processing is aimed at handling errors and exceptions, wherever possible, before the occurrence of a true failure.
- Error Treatment - Fault treatment is aimed at preventing previously activated faults from being reactivated.

Functional

- Use **try** and **catch** blocks throughout code
- Consider the case when **"Bad Data"** arrives and how to continue. E.g. put **"Bad Data"** in repair queues

Operational

- Achieved through duplications. For examples: Disk Mirroring, e.g. RAID (*)
- Specialised operations staff
- Autonomic Computing mechanisms

Redundant array of inexpensive / independent disks

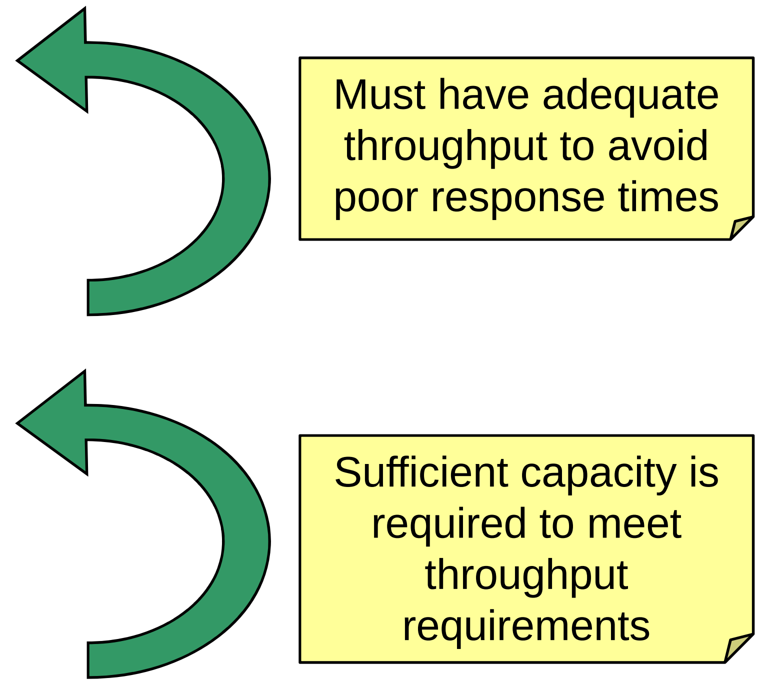
Availability – a final word

- It is estimated that
  - ~20% of your total availability is a function of your use of technology
  - ~80% is a function of your people and processes
- Someone may say:
  - The root cause of the system outage was that firewall logs were full
  - The real reason was there was insufficient process in place to monitor the logs and clear them down
- Technology and design is important, however don’t assume that is your only challenge

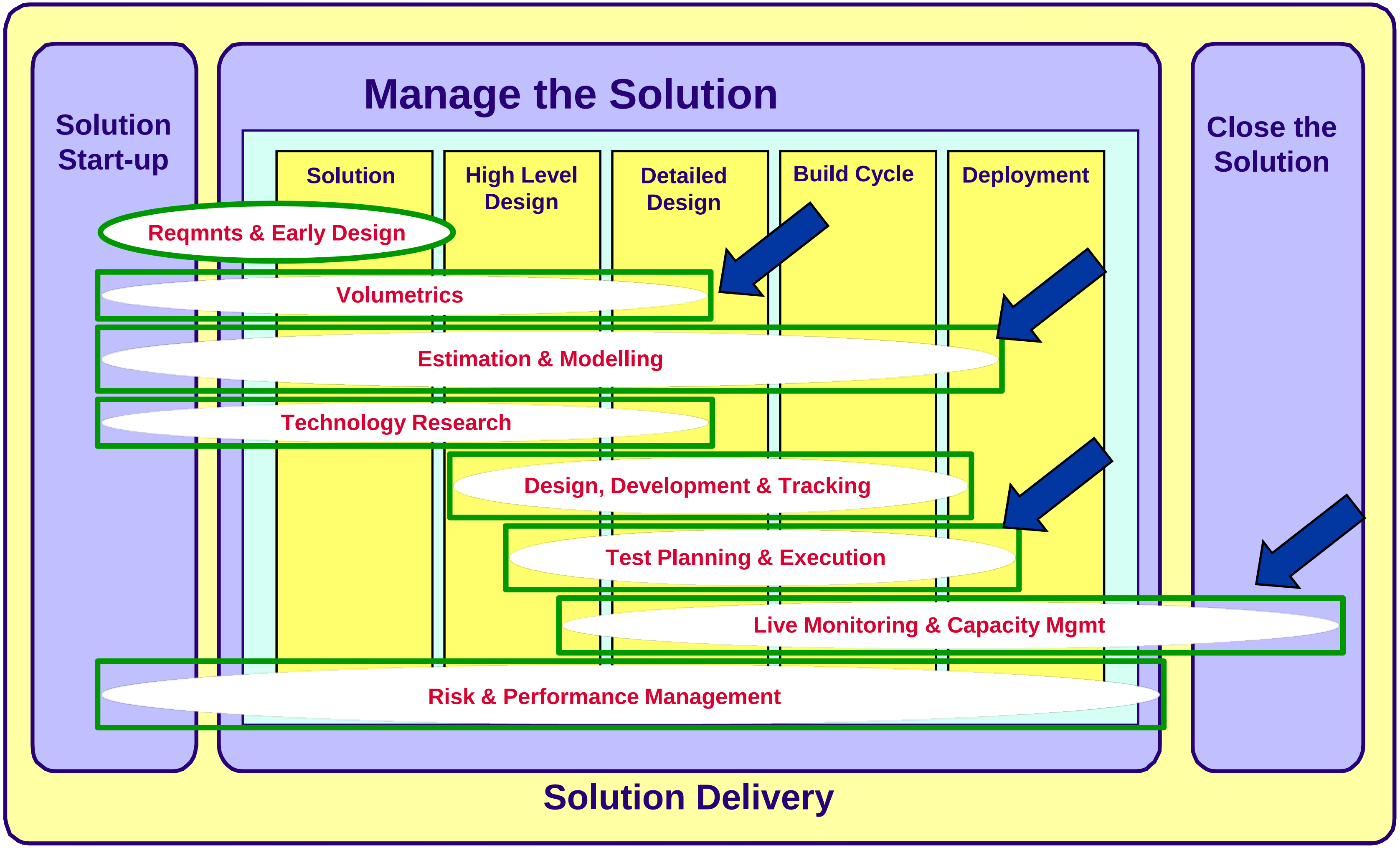
Performance

There are three main, heavily inter-related aspects of Performance to be considered

- Response Times
  - On-line response times
  - Batch run times
- Throughput
  - Transactions per second
  - Records processed per hour
- Capacity
  - Component sizing to handle load
  - Contingency and Scalability



Major activities a Performance Engineer executes across the project lifecycle



Enterprises often cannot provide detailed volumetric information – often, it has to be derived (or guessed!)

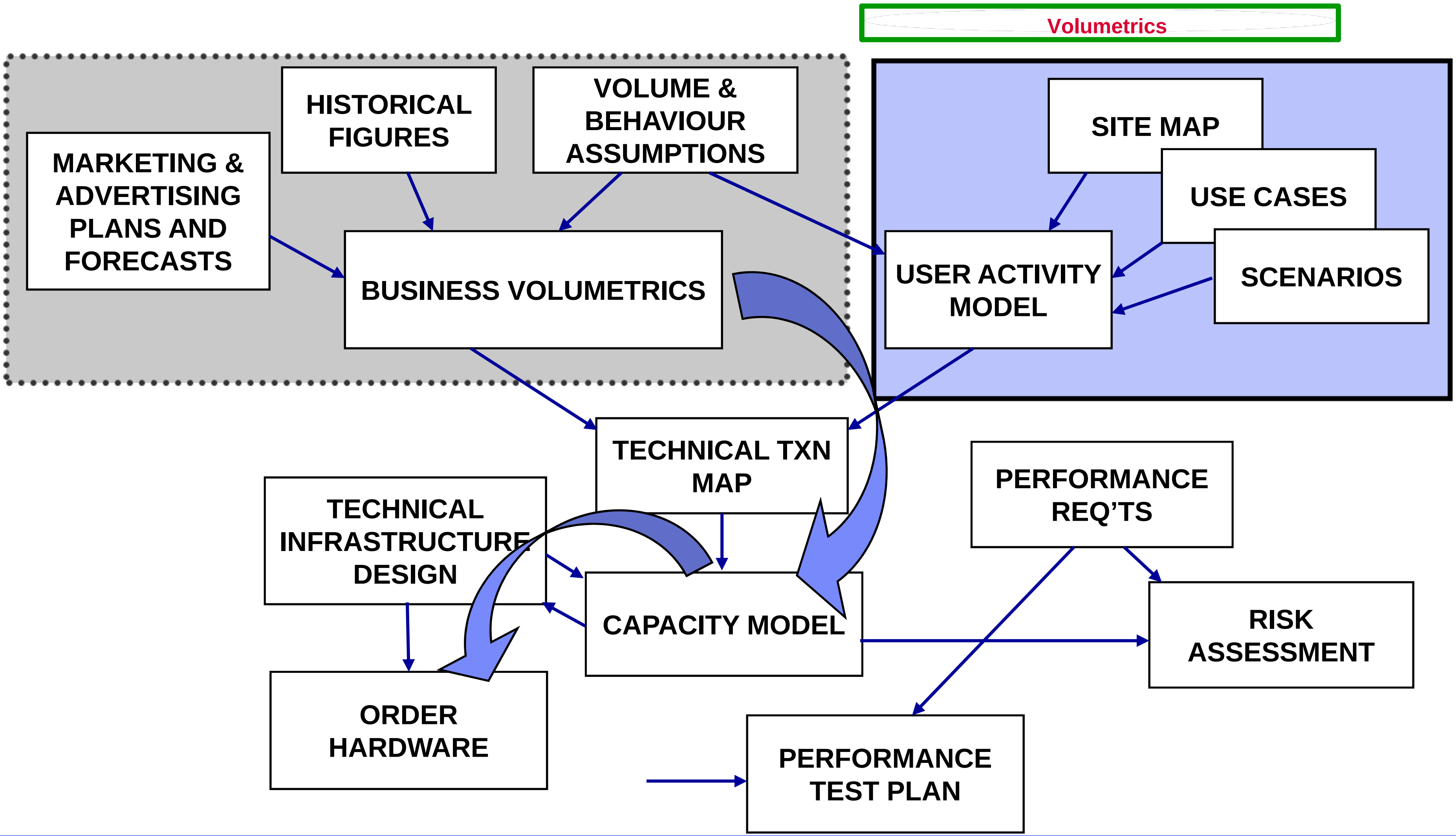
Real questions IBM Performance Engineers have been asked by customers

Volumetrics

- *"We're just about to spend £20m on advertising our new brand. How many web servers do we need?"* - Insurance company
- *"Will this new digital audio broadcasting solution perform OK, given we don't know how we are going to use it yet?"* – Public service radio broadcaster
- *"How fast is the Internet?"* – Offshore bank

Volumetric data can be traced from various sources

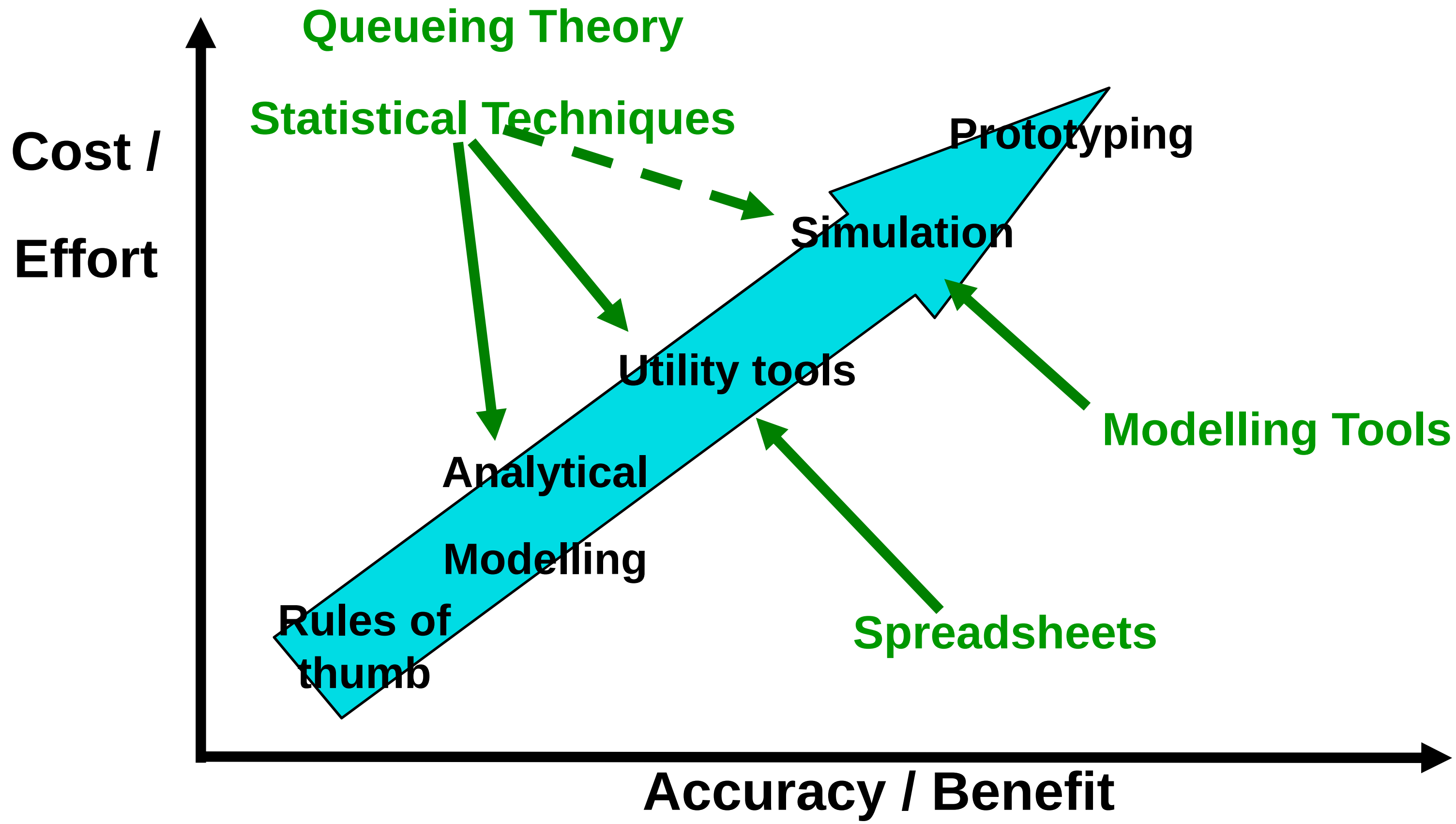
An example "volumes map" used on an engagement



Performance characteristics of a system can be investigated in more detail by creating a model

Estimation & Modelling

- Different techniques are available different levels of effort to provide answers with different levels of reliability



Estimation & Modelling

Exercise - Volumetric estimation

- Shop
  - In the peak hour, on the average, every 60 seconds a new shopper arrives (random arrivals, generated by a Poisson process)
  - Average shopping time: 10 minutes (random distribution)
  - Average time at the cashier: 2 minutes (random distribution)

- Estimate the minimum number of carts the shop must have to make sure that customers almost never have to wait for a cart

- Estimate the minimum number of cashiers required to make sure that the number of customers that must wait for a cashier is almost always at most 3

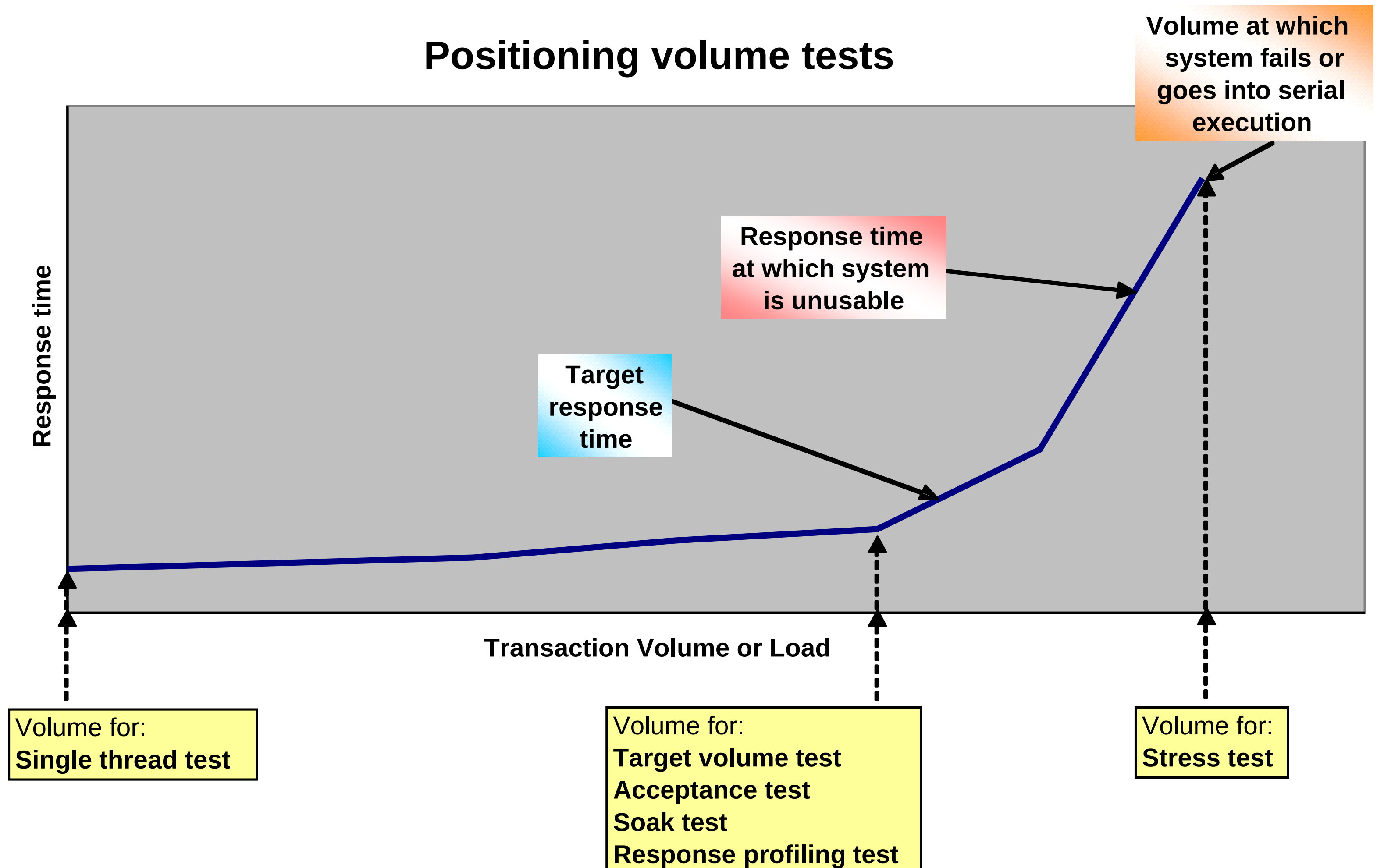
The demo uses the Ptolemy II simulation modelling tool

Open Source simulation toolkit written in Java available from http://ptolemy.eecs.berkeley.edu/ptolemyII

The model is a Discrete Event simulator. It has been extended with some custom actors (in porkbench.jar)

A range of Performance Test types are used for different purposes

Test Planning & Execution



Live Monitoring and Capacity Planning activities aim to ensure that the system continues to meet its performance targets once in live

- Once in live, there is the possibility of collecting real performance data, such as:
  - Real business volumetrics (volumes of events, business entity volumes)
  - Technical volumetrics (transaction volumes, data sizes, …)
  - Response times (at various tiers of the system)
  - Traffic profile information (peaks, distributions)

Live Monitoring & Capacity Mgmt

- Systems are subject to change from many perspectives:
  - Future business demand
  - Changes in user behavior (e.g. affecting workload mix)
  - Infrastructure change (network upgrade, hardware platform change, consolidations, …)
  - Application change (product upgrades, replacement of middleware, new functional requirements …)
- As with initial performance modelling, the capacity plan needs cover all resources which could cause a system to perform poorly
  - Performance bottlenecks can occur at any part of the chain
  - Incentives to ensure the system makes optimum use of the available resources
- This process starts at the design phase
  - Capacity planning will likely be the responsibility of a different group
  - The ability to record and report performance data must be considered during the design phase
  - Systems management design needs to support the capacity planning processes
  - Applications may have to be explicitly instrumented to record response time data

Summary of Topic

- Despite continuing advances in technology, IT Architects spend significant amounts of time engineering systems to account for Quality of Service requirements
  - In the context of often significant constraints
  - Software and infrastructure designs need to be iterated together to achieve goals
- Non-functional requirements & service levels may be contractually binding
  - Failure to achieve targets may result in financial penalties for the IT provider, and/or lost business for the customer
  - If a design cannot be established which meets requirements, this is top severity project issue
- Modelling theory, techniques and tools are available to assist with evaluating design alternatives
  - Employing them successfully requires understanding of the systems elements, management of assumptions and appropriate modelling skills
- Regardless of the quality of design, the quality of implementation must be validated through testing
  - QoS design should inform test strategy and test planning

- *The effort expended should always be proportionate to the risk involved*