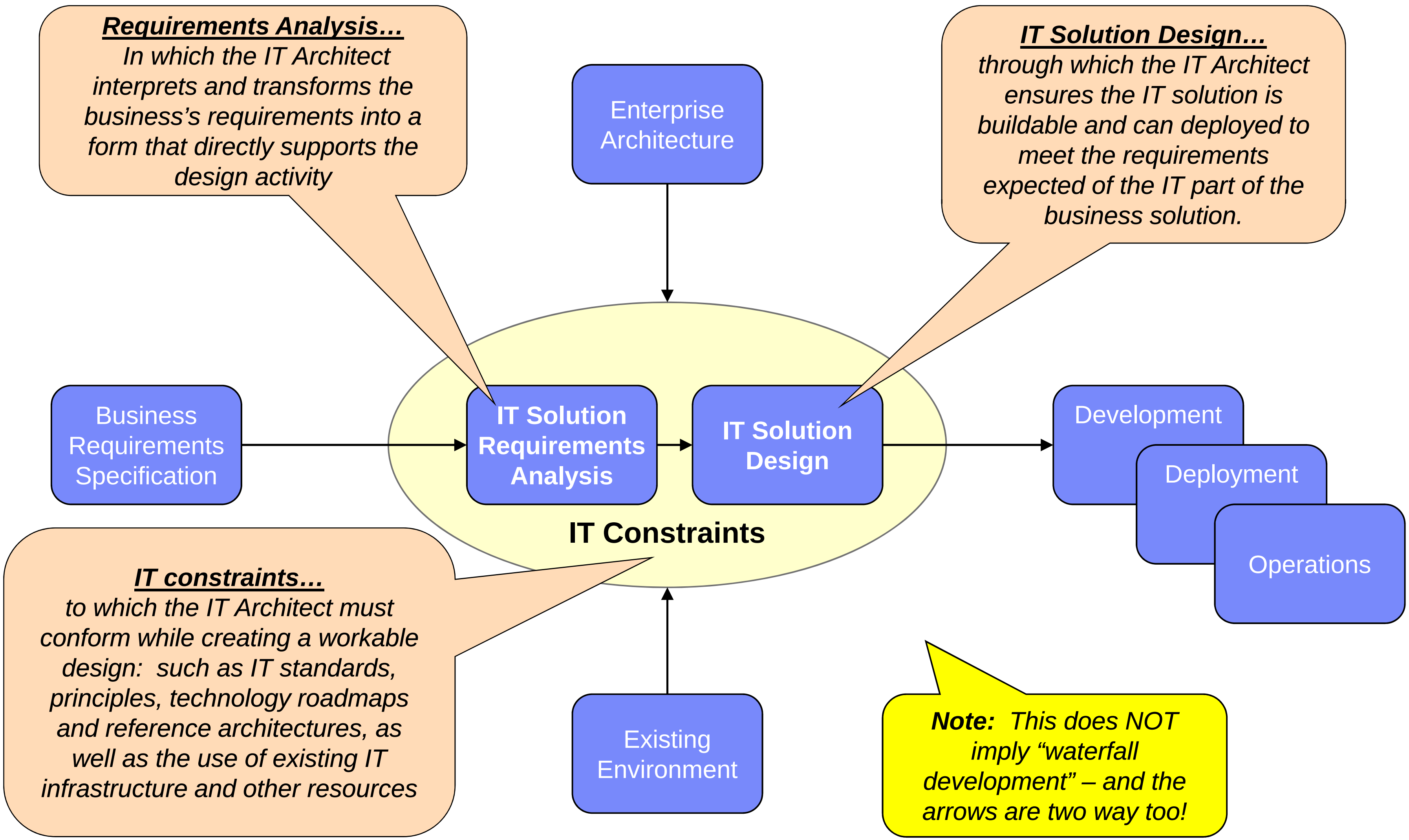
IT Architecture - Course Agenda

Day			Type	Name	Start	Finish	Duration
1	5.10.	Friday	Lecture	Uebersicht über die Vorlesung	12:15 PM	12:30 PM	00:15
			Lecture	What is IT Systems Architecture	12:30 PM	12:55 PM	00:25
			Lecture	What do IT Architects do all day	12:55 PM	1:20 PM	00:25
			Lecture	Why Architecture	1:20 PM	1:45 PM	00:25
2	12.10.	Friday	Lecture	Qualities and Constraints	12:15 PM	1:00 PM	00:45
			Lecture	Developing a solution's IT Architecture	1:00 PM	1:45 PM	00:45
3	19.10.	Friday	Case Study	Introduction; Scenario and Business Problem	12:15 PM	12:30 PM	00:15
			Case Study	Part 1: Functional Requirements	12:30 PM	1:30 PM	01:00
			Case Study	Part 2: Architecture Overview	1:30 PM	1:45 PM	00:15
4	26.10.	Friday	Case Study	Part 3: Functional Aspect	12:15 PM	1:00 PM	00:45
			Case Study	Part 4: Operational Aspect	1:00 PM	1:45 PM	00:45

IT Architecture ensures the business's requirements for an IT system are satisfied in a coherent and well structured way

An IT Architect achieves this goal by balancing many forces, helping to ensure the IT solution to a business requirement recognises "the art of the possible & affordable"

The IT Architect therefore focuses on three things



Qualities and Constraints in IT Architecture

Non-Functional Requirements

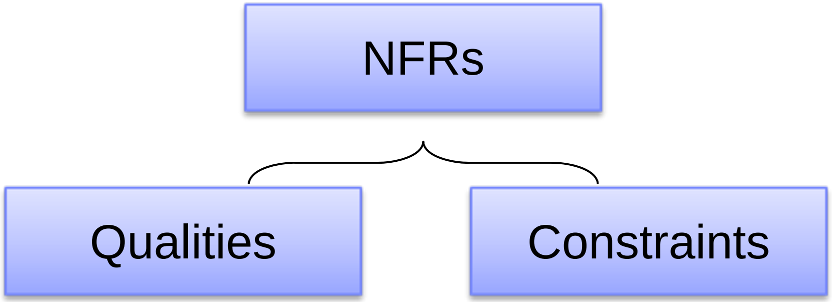
Examples: Availability and Performance

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

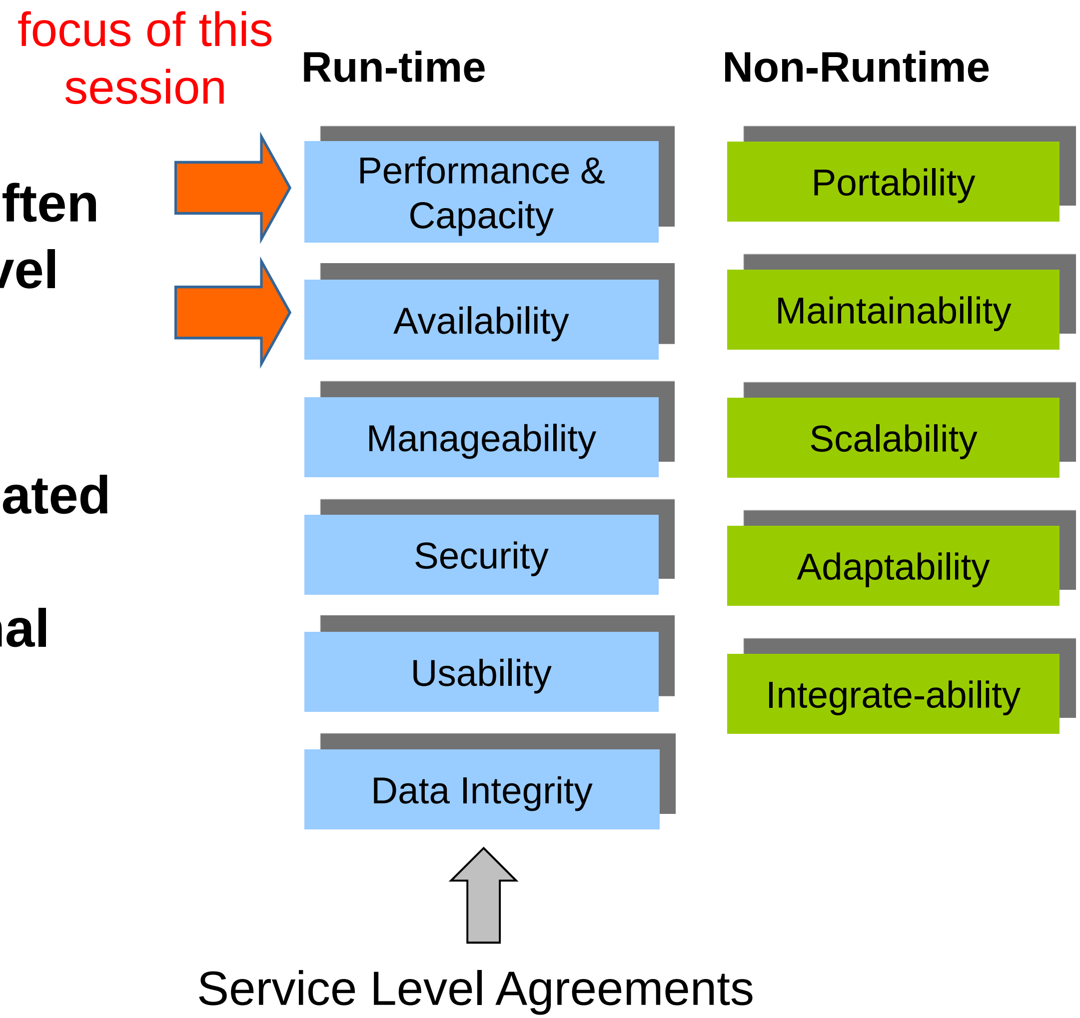
- **The business aspects of the project, customer's business environment or IT organization that influence the architecture**

- **The technical environment and prevailing standards that the system, and the project, need to operate within**

Business	**Technical**
Regulatory	Legacy Integration
Organisational	Development Skills
Risk Willingness	Existing Infrastructure
Marketplace factors	Technology State of the art
Schedule & Budget	IT Standards

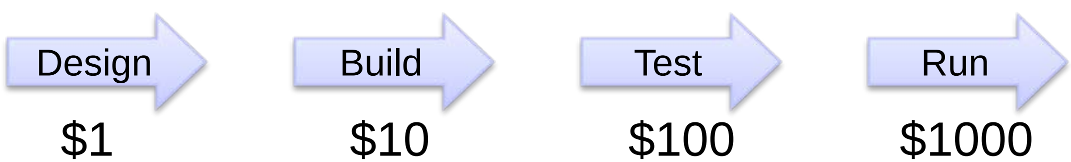
Qualities

- **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.**

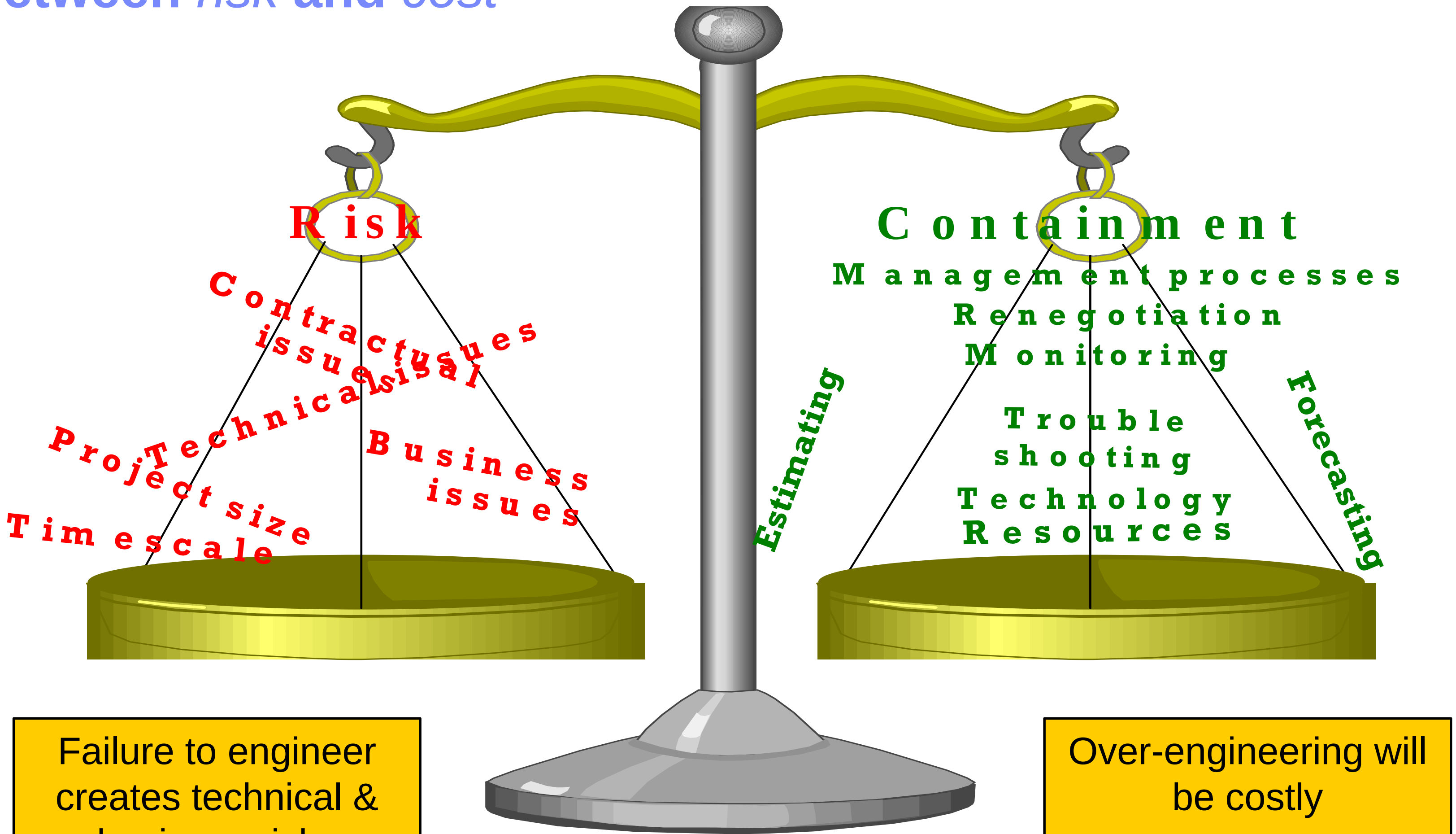


The best technique for reducing the risk of poor quality of service is to consider the qualities from the start

- 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

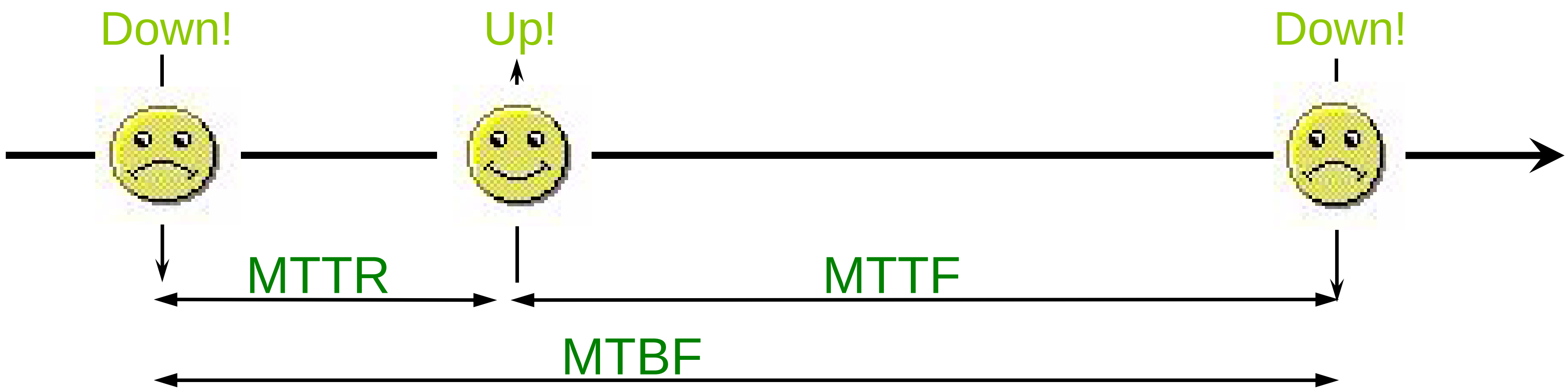
Over-engineering will be costly

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

Key Availability Terms



- **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
- **Recovery Time Objective (RTO)** is the duration of time and a service level within which a business process must be restored after a disaster (or disruption) in order to avoid unacceptable consequences associated with a break in business continuity.
- **Recovery Point Objective (RPO)** is the maximum tolerable period in which data might be lost from an IT service due to a major incident.

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

99% 99% 99%

1 — 2 — 3

Down

10.8 days / year

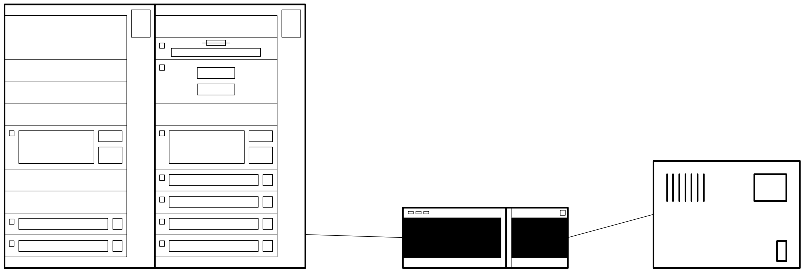
$$\text{Availability (A)} = A_1 \times A_2 \times A_3$$

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

Functional

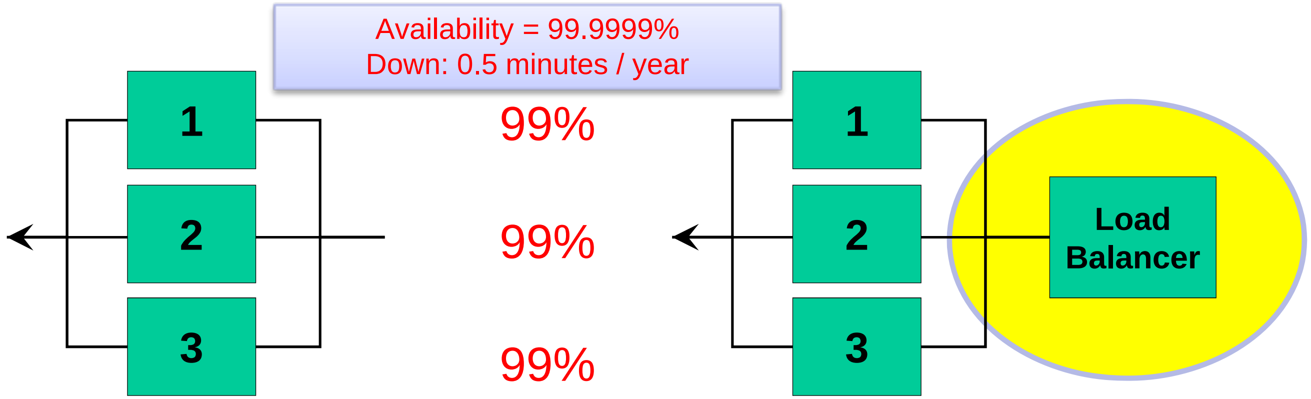
Application Server	99%
Product Catalogue	99%
Database Manager	99%

Operational



99% 99% 99%

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

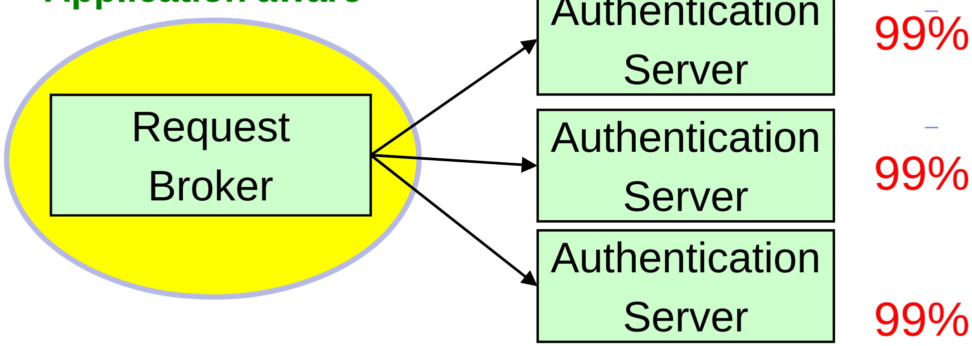


$$\text{Availability} = 1 - [\,(1-A_1) \times (1-A_2) \times (1-A_3)\,]$$

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

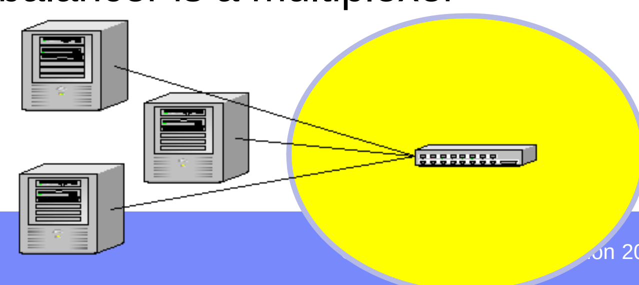
Functional

Application aware



Operational

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

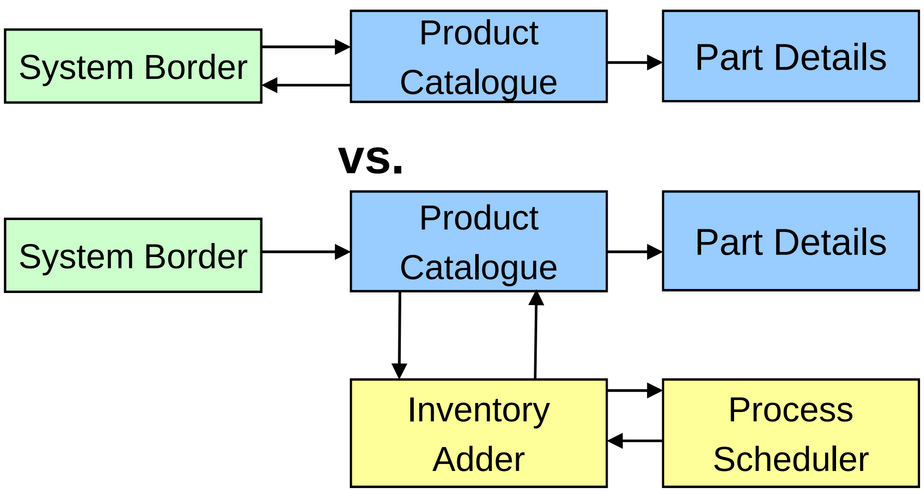


Separation of Concern is a technique that can be used to enable a loose coupling for components that provide critical services

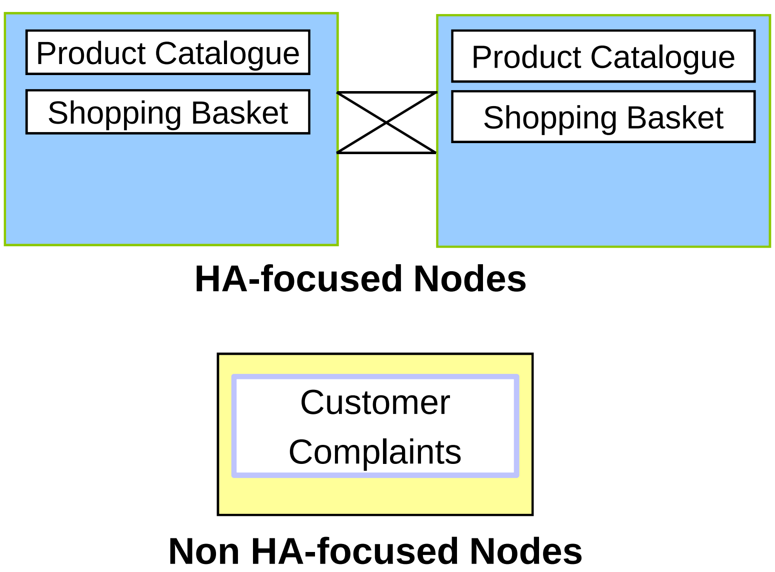
- The separation of components with regard to business importance and their availability characteristics

Functional

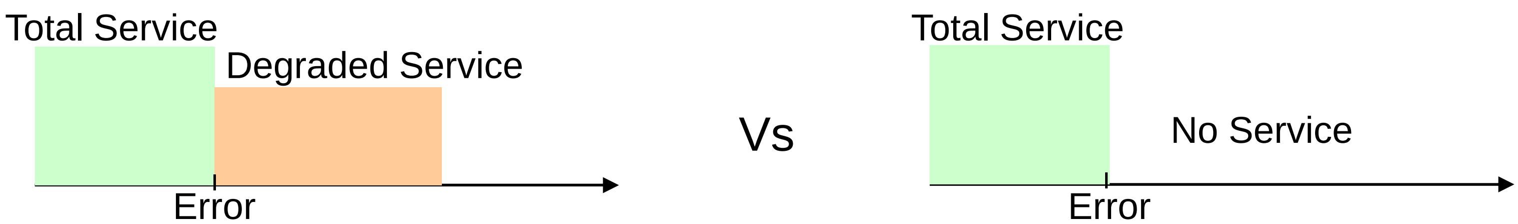
- Loose coupling of HA Components



Operational



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

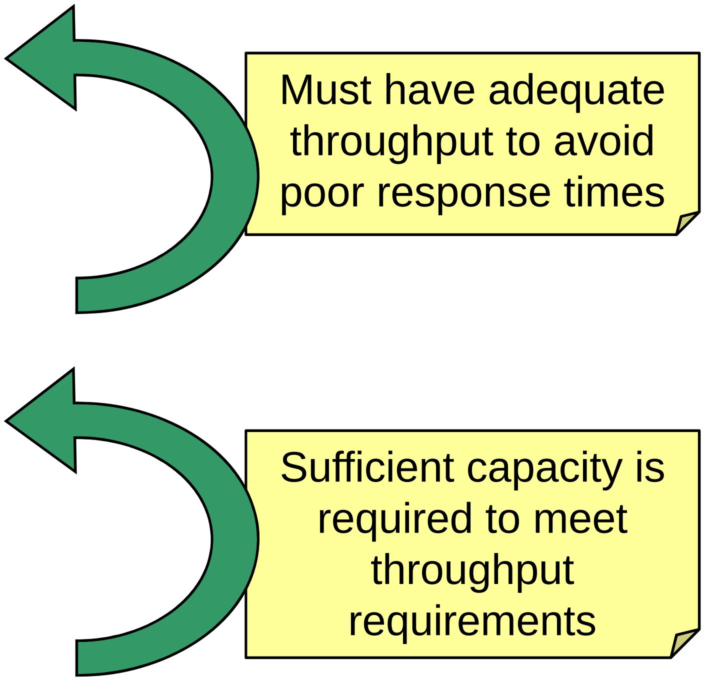
Availability – a final word

- 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

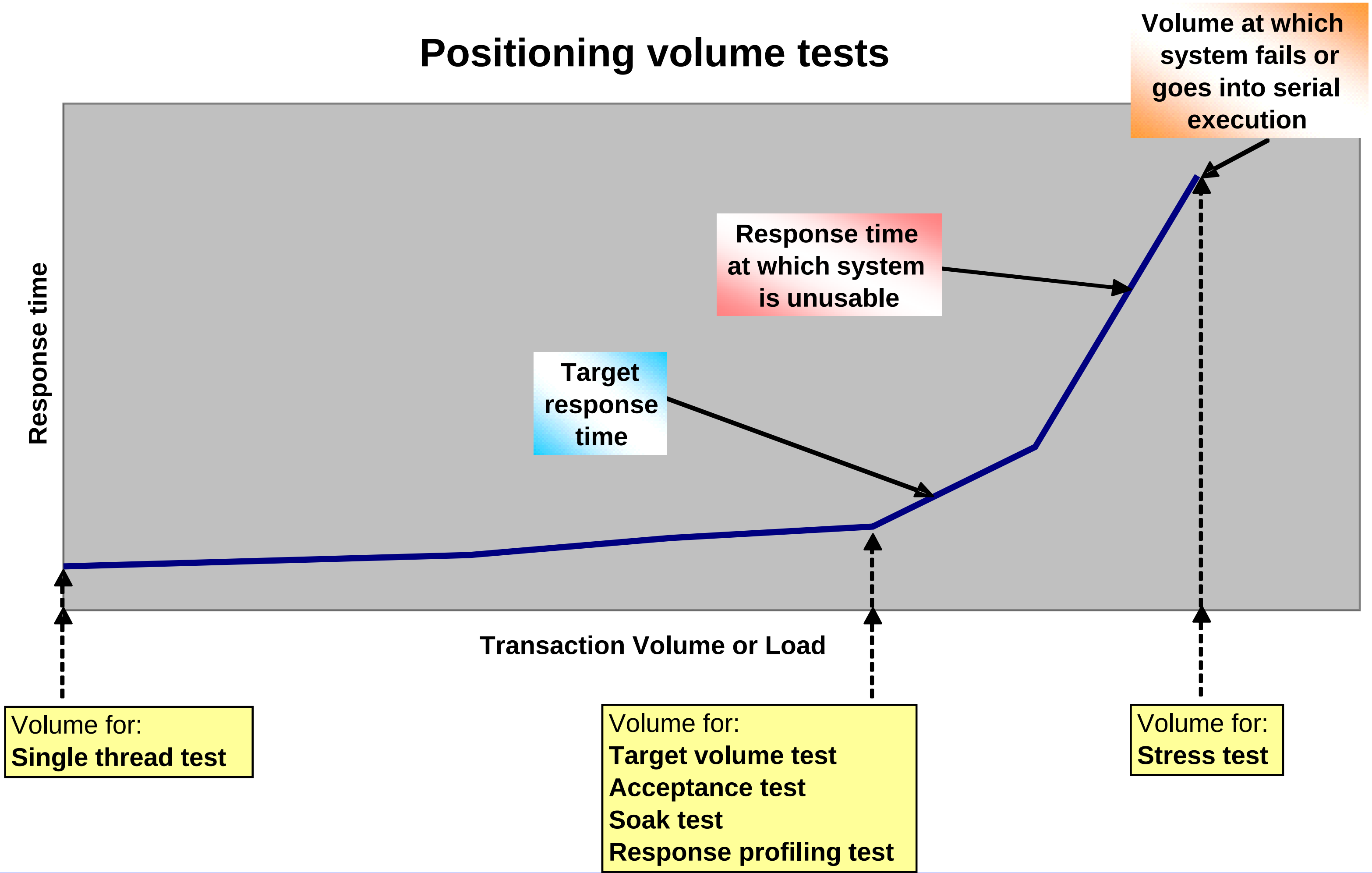
Performance

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

- Response Times
  - On-line response times
  - Batch run times
  - Even when the load increases
- Throughput
  - Transactions per second
  - Records processed per hour
  - Prerequisite for adequate response times
- Capacity
  - Component sizing to handle load
  - Scalability
  - Prerequisite for both sufficient throughput and adequate response times

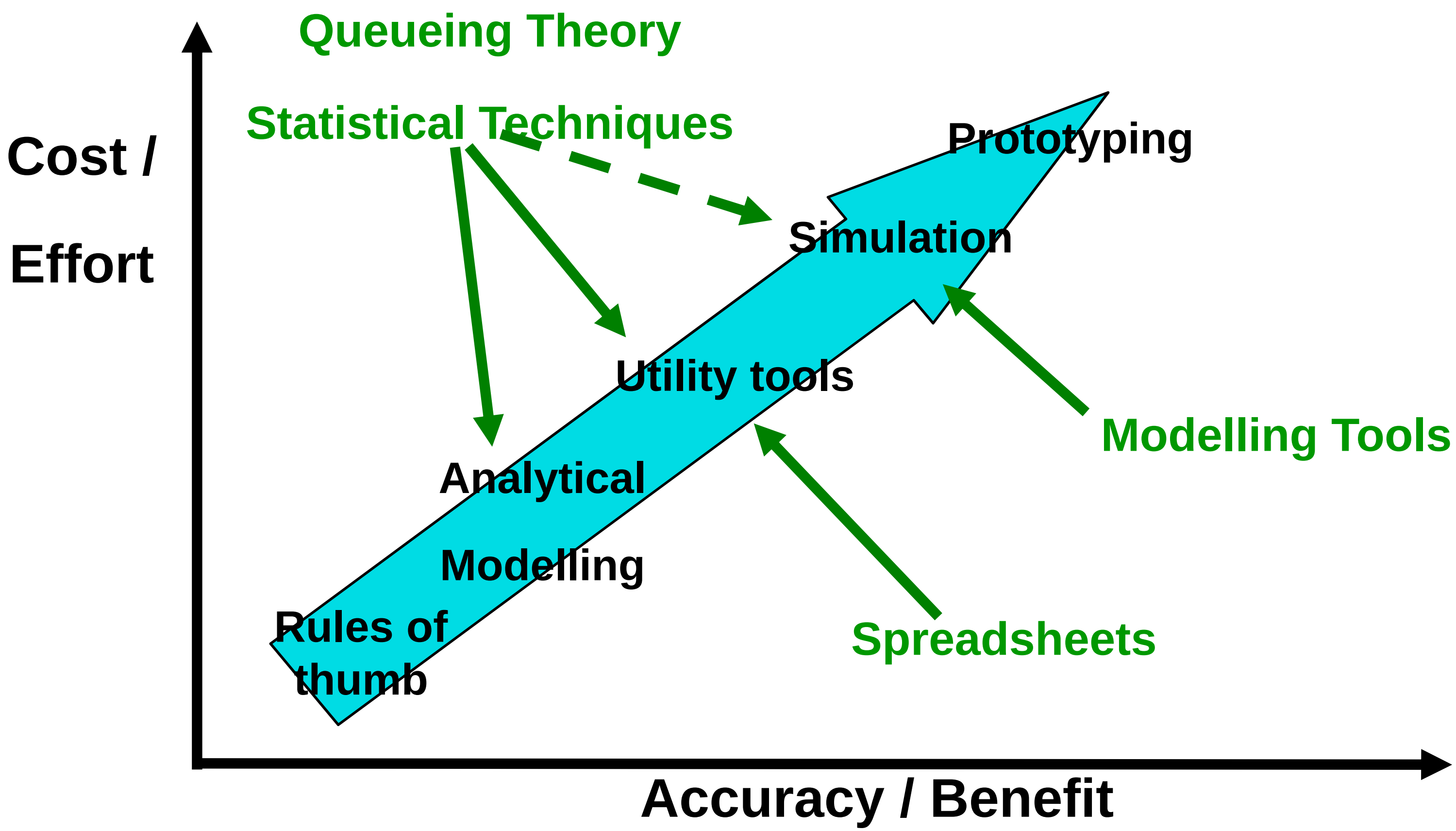


A range of Performance Test types are used for different purposes



Performance characteristics of a system can be investigated by creating a model

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



Example - Volumetric estimation supported by simulation

- 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)

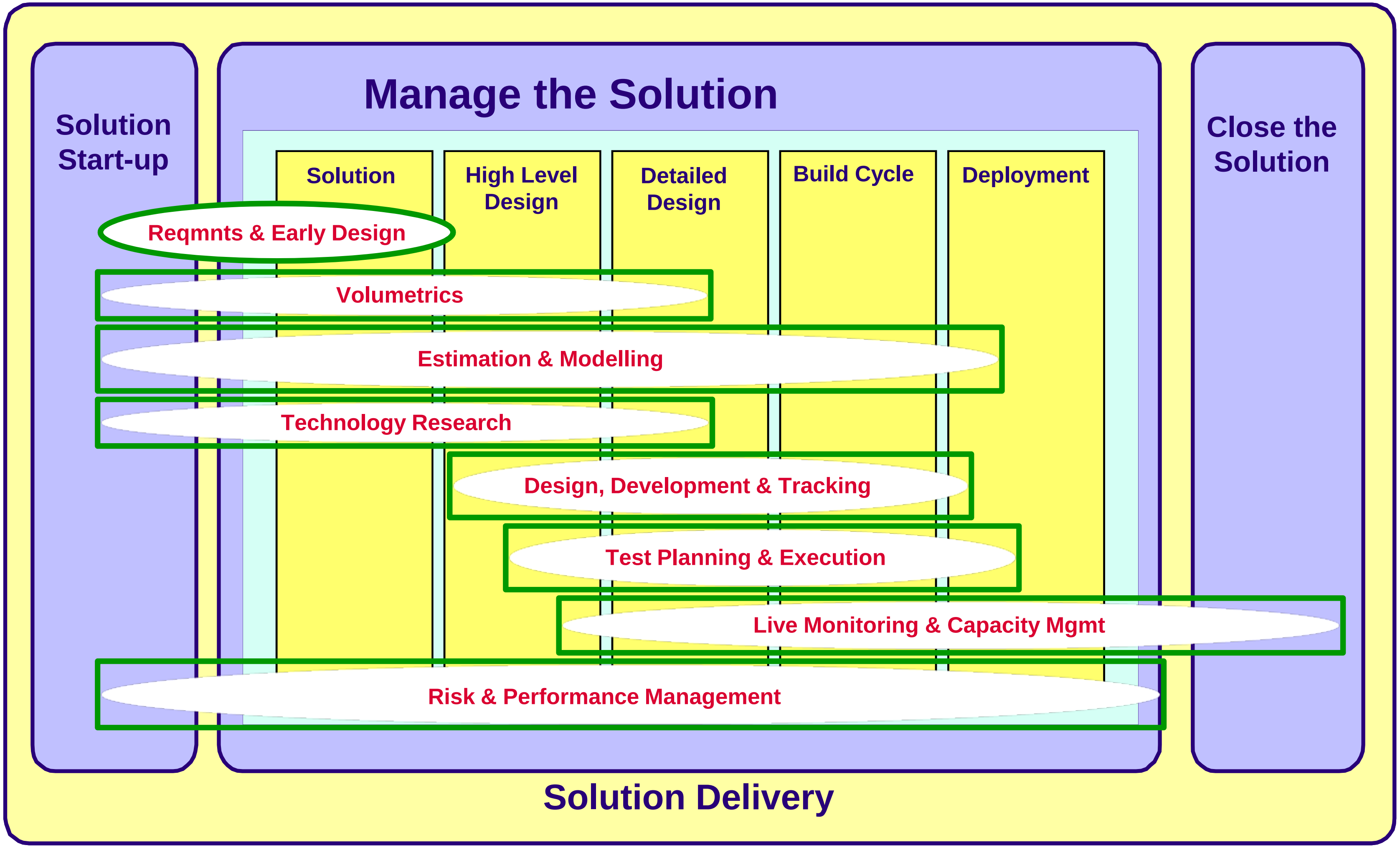
Simulation modelling has significant advantages … but beware …

- Provides a safe environment in which to understand the effects of change (an environment for experimentation)
  - Parameterise models to ask any number of "what-if" questions
  - E.g. Test out different placement and configuration options
- Powerful and flexible modelling capabilities
  - Model complex interactions between layers, components, subsystems, etc.
  - Use probability distributions for service times, arrival rates, etc.
  - Model different queue servicing disciplines (fcfs, round robin, priority ...)
  - Analyse time-dependent variations in incoming workloads
- Modeller does not need to know or use complex formulae

- Promotes real understanding of the system through visualisation and / or animation
  - See peaks, troughs, start-up, cool down periods
  - See times of specific events
- Promotes real understanding of end-to-end behaviour
  - model complex interactions between components, subsystems, etc.
  - model interaction between human and IT domains

- However:
  - has high start-up cost in both skills and resource
  - can be costly
  - requires detailed system knowledge and/or access to subject matter experts
  - **is only as accurate as inputs**
  - has a danger of false confidence
  - is only as good as the model

Major activities a Performance Engineer executes across the project lifecycle



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)

- 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*