Kai Schwidder
Distinguished IT Architect

Enterprise IT Architectures

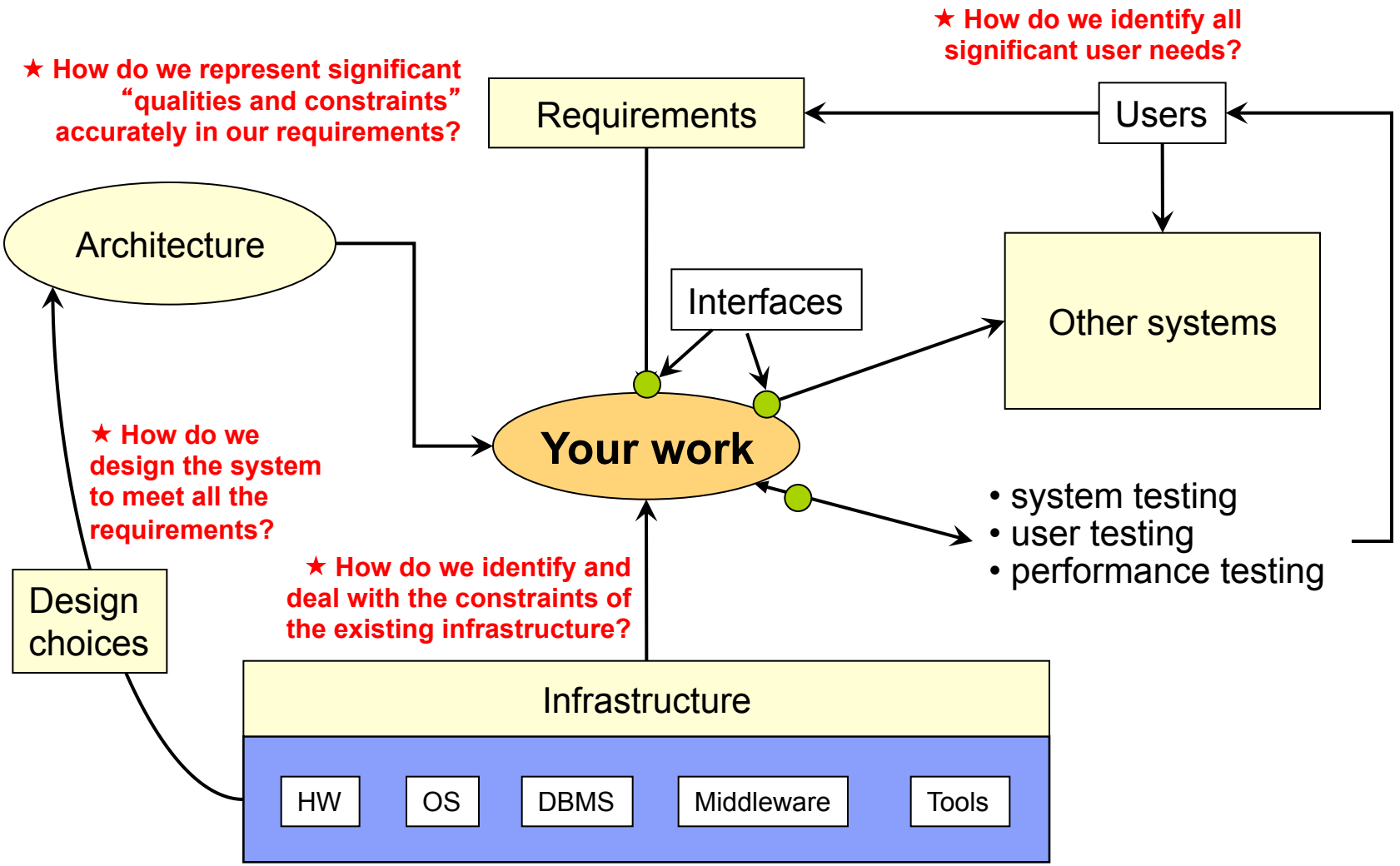
Solution IT Architectures – Key Elements 2

Quality and Constraints

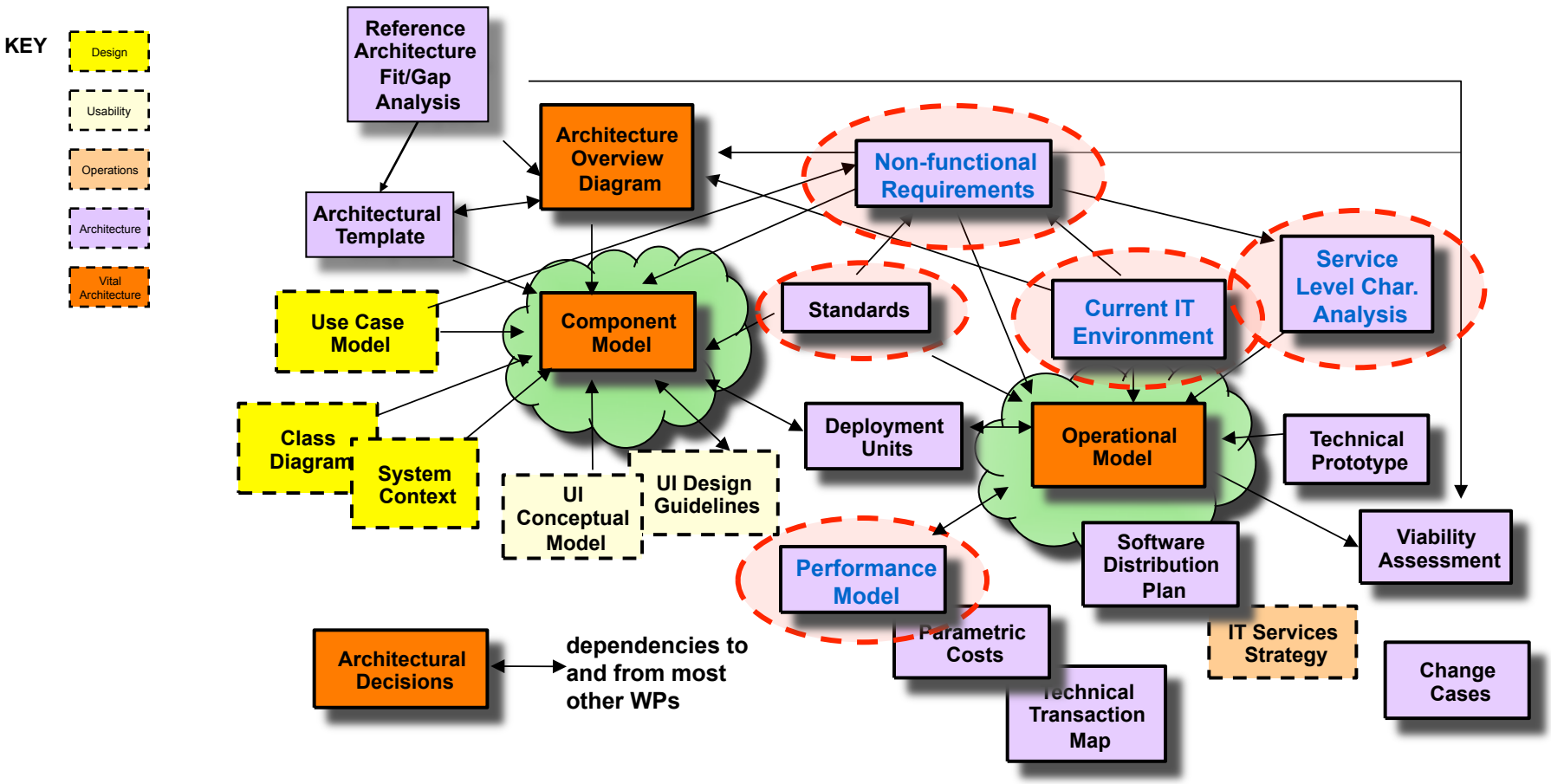
Agenda (Part I - today)

- ***Qualities & Constraints in IT Architecture – overview***
  - What are "qualities and constraints" in IT Architecture?
  - Non-Functional Requirements and their quality
- **Focus on *Availability***
  - Availability modelling
  - Availability design techniques
- **Focus on *Performance***
  - The Performance Engineering Lifecycle
  - Volumetrics
  - Estimation and Modelling
  - Optional exercise

"The wider context"



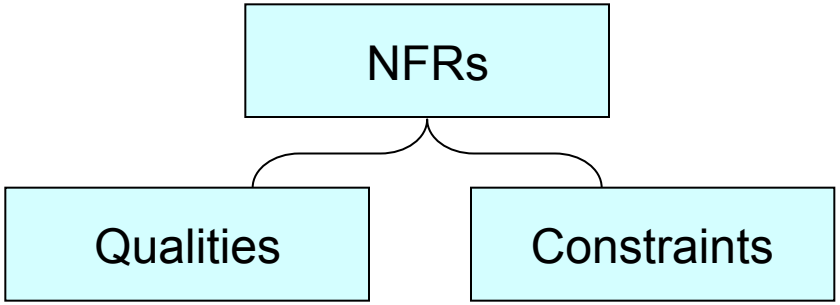
Architectural Thinking — Method support



key "qualities & constraints" related work products highlighted

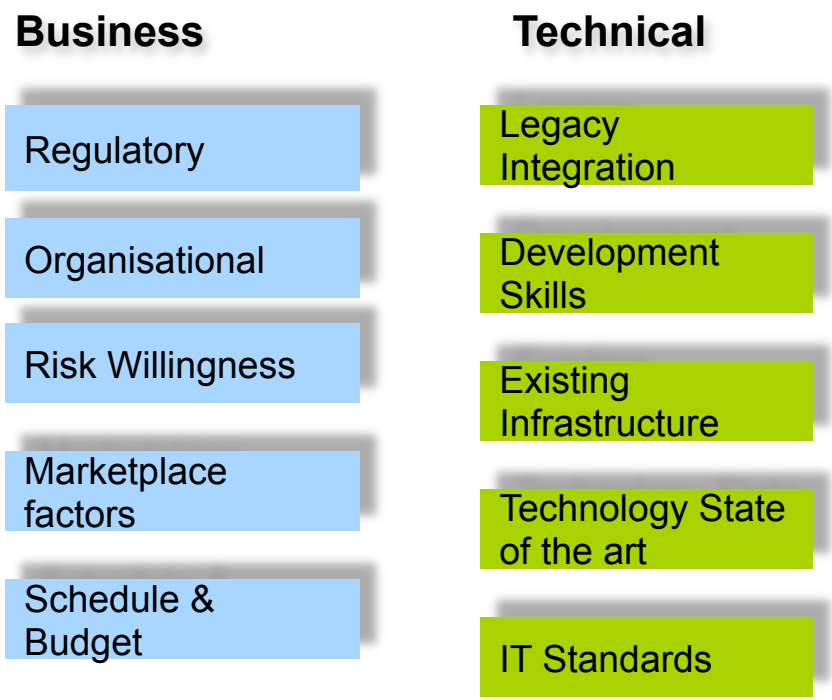
Qualities and Constraints are often referred to as '*Non-Functional Requirements*'

- **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**



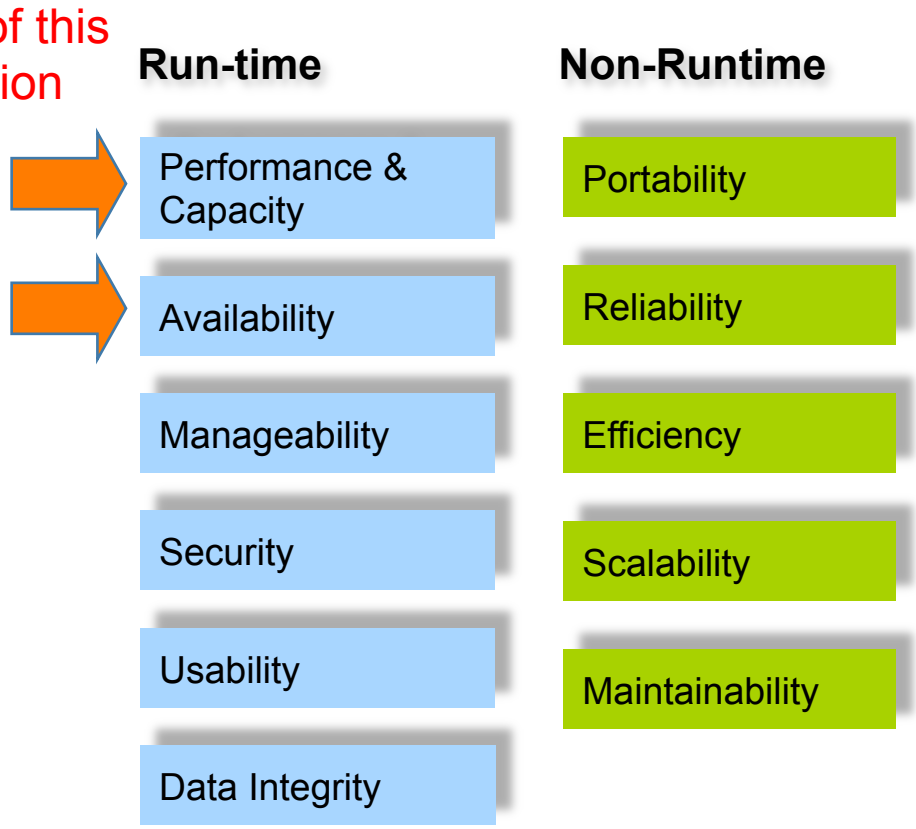
Constraints

- **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**



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.**

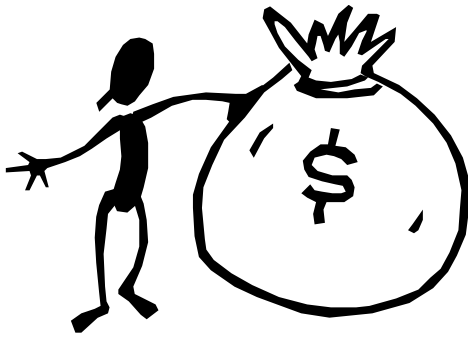


Quality-of-Service "metrics" have an impact on a company's bottom line – consider online services …

- Tangible metrics are ones which can be quantified as a measure of "Loss per transaction":
  - In the online world it's important to do a great job with buyers
  - People leave ".com" sites because of pages being unavailable or too slow
  - Even if a site is available and fast – it is *usable*?
  - Slow sites and/or poor navigation techniques cost companies sales
- Intangible metrics are less quantifiable and require estimation:
  - Consider a web site to be really just an extension of a company's BRAND
  - Visiting a web site is the same as visiting a store with the company's logo on it
  - Even if the experience produces no revenue, it can have an impact on return visits
  - Ideally, a customer should develop a mechanism for taking into account these "soft" costs in order to work out their quality of service requirements

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 …



(This is not a requirements engineering module, but …)
Common problems with Non-Functional "requirements"

- **Requirements are often vague or unactionable**
  - **They need further elaboration, clarification, investigation (and possibly rejection)**
  - **It may be possible to derive clear, actionable intentions from them**
- **Requirements can be statements of principle or good intention but come with little enforcement**
  - **The organisation's governance models are central**
- **Once captured, requirements are often treated as "musts" or "givens" whereas in fact they are "tradable" and may need to be challenged**
  - **Classic example is "given" technology standards (e.g. "all applications in .NET") or infrastructure constraints ("64kbps links to offices")**
- **Requirements are often of poor quality**
  - **Watch out for these issues: Unrepresentative, unclear, inaccurate, inconsistent, incomplete or unnecessarily constraining**
- **NFRs documents often become "dumping grounds" for things which don't have another home**
  - **(regardless of quality or suitability)**

In reality, "requirements" are actually "influences" whose characteristics we have to be clear about

- **A "requirement" in the widest sense …**

  ***stems from many sources* … [Context]**

  ***is either an aspiration or a constraint* …[Polarity]**

  ***may be negotiable (i.e. varying in importance)* … [Strength]**

  ***may be generic or specific* … [Level of generality]**

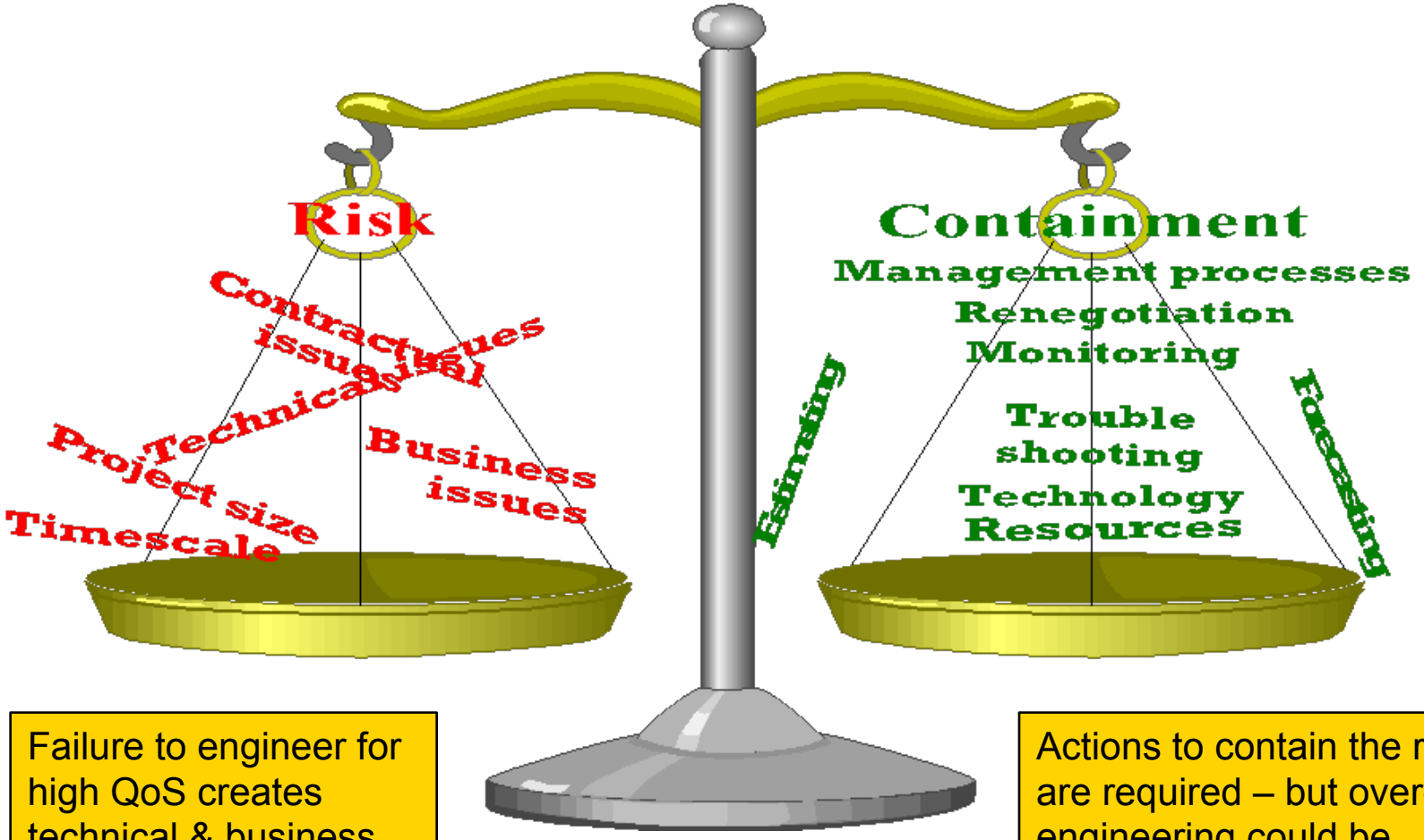
  ***may be directly actionable or difficult to interpret* … [Actionability]**

  ***may affect many components of the Solution* … [Affected objects]**

  ***may be helpful (good quality) or unhelpful (poor quality)* [Quality]**

- **Unless we understand the real context and importance of each requirement, we risk producing the wrong solution**

Beware: a BALANCE must be maintained between *risk* and *cost*



Failure to engineer for high QoS creates technical & business risks

Actions to contain the risk are required – but over-engineering could be unnecessarily costly

Availability

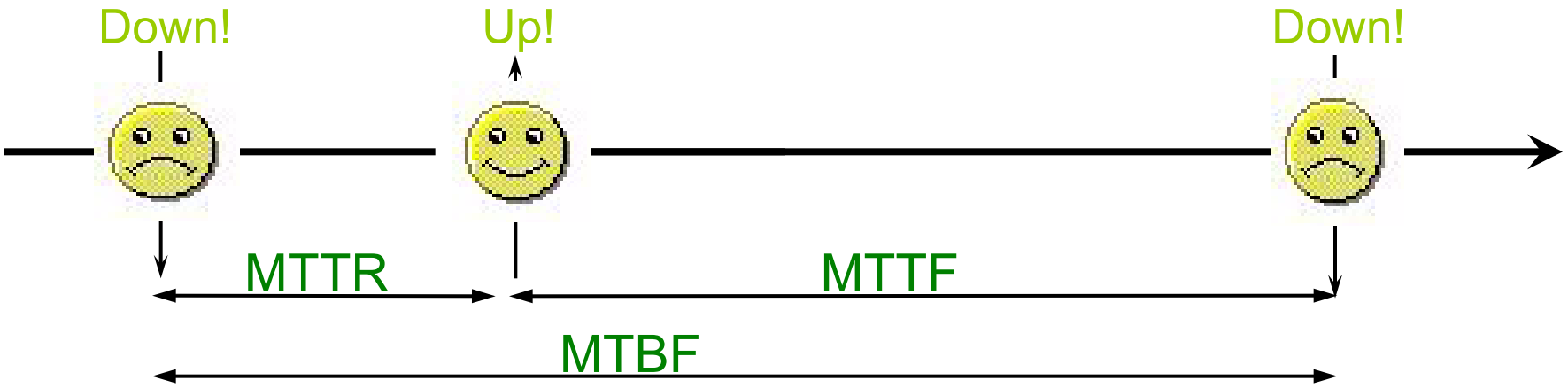
The reality of Availability is that customers directly relate it to the End User experience

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

- **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 …



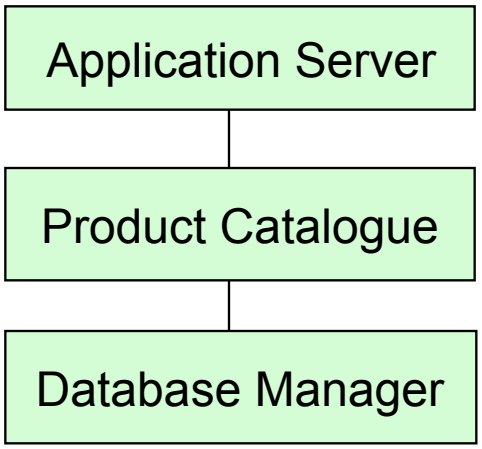
- **Mean Time to Recover (MTTR) is the typical time that it takes to recover (includes repair) a component, sub-system or a system.**
- **Mean Time to Failure (MTTF) is the mean time between successive failures of a given component, sub-system or system.**
- **Mean Time between Failure (MTBF) is the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

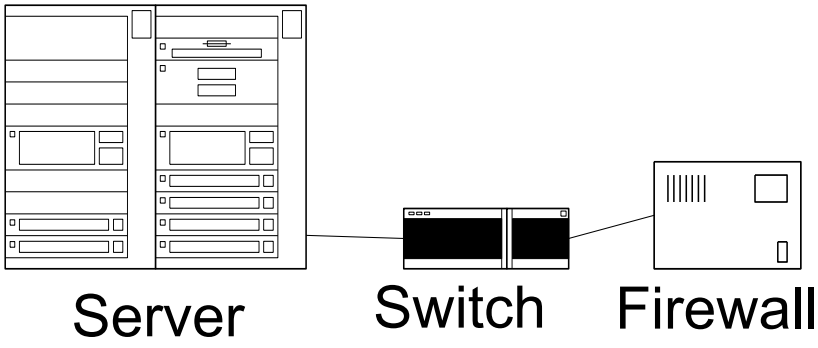
Availability (A) = A^1 x A^2 x A^3

- 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

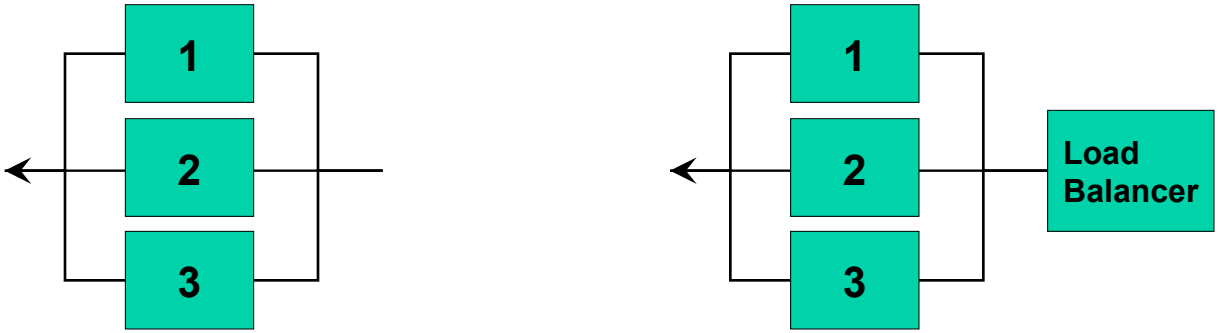
Functional



Operational



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

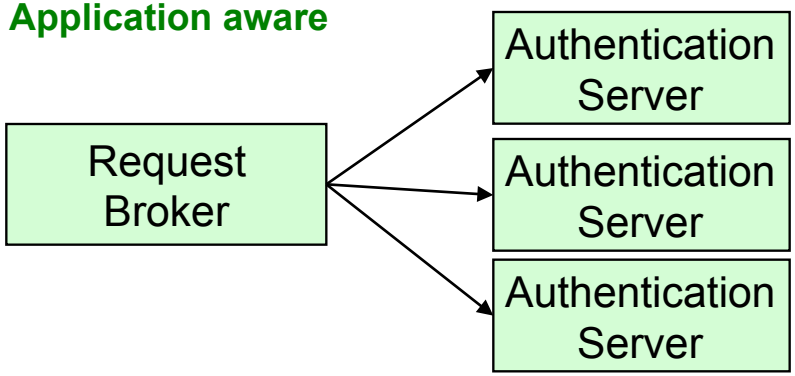


Availability = 1-[(1-A(1))x(1- A(2))x (1-A(3))]

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

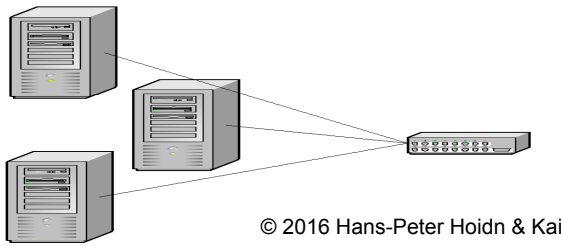
Functional

Application aware



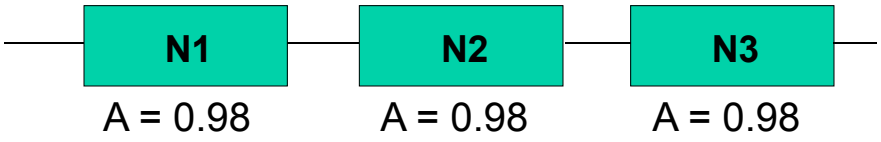
Operational

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

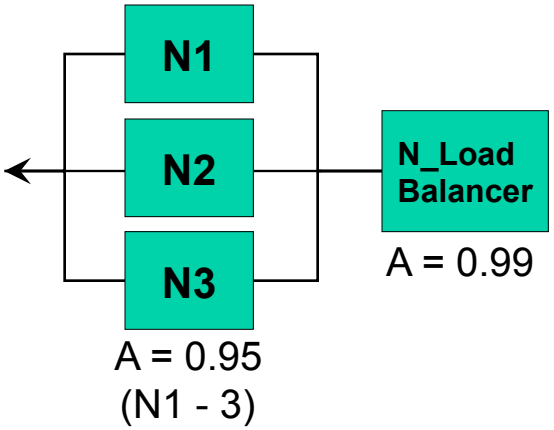


Exercise 2 – Serial vs. Parallel Availability

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



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



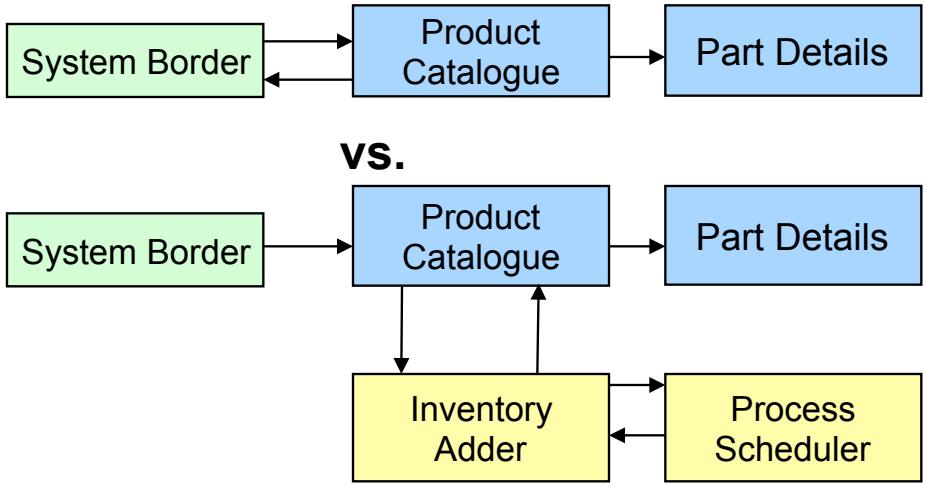
- **5 minutes**

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

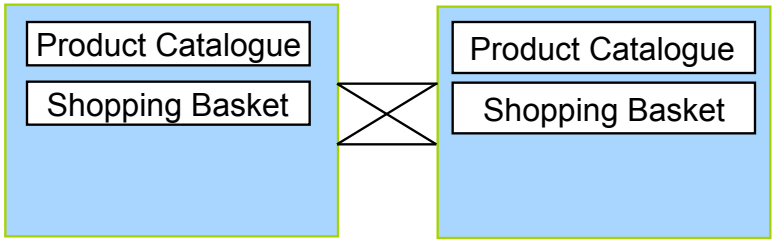
- The separation of components with regard to business importance and their availability characteristics

Functional

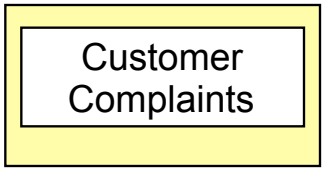
- **Loose coupling of HA Components**



Operational

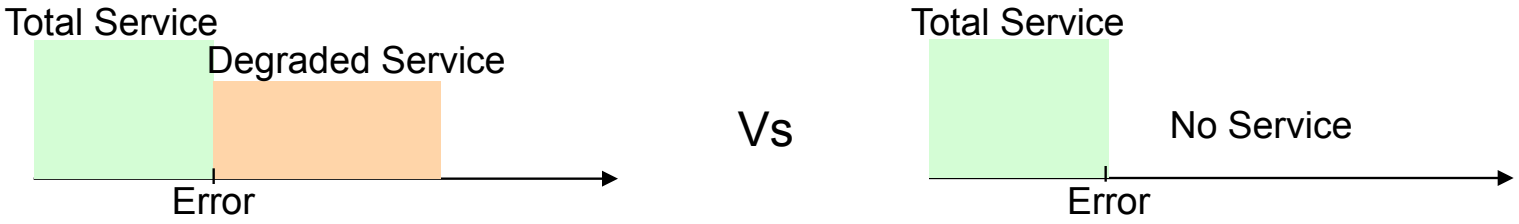


HA-focused Nodes



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**

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
- E.g. someone says the:
  - Root cause 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

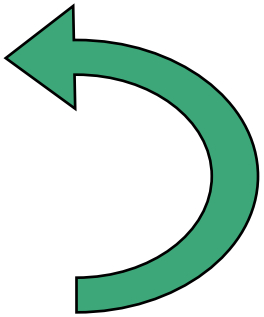
Performance

What is Performance?

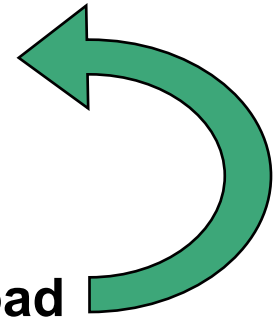
- **Definition**
  - **“Performance. The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage.” [IEEE-610.12]**
- **In general**
  - ***Timeliness* of response, and *predictability*, are the two main goals**
  - **“Faster” is not always enough, as in for example, a real time system requires extremely consistent performance**
- **An (old) quote (from ICCM):**
  - **“A manager's goal should always be to strike the right balance between system function, processing costs, people costs, and performance. This is why the technical aspects of performance can never be entirely divorced from organizational politics”**

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

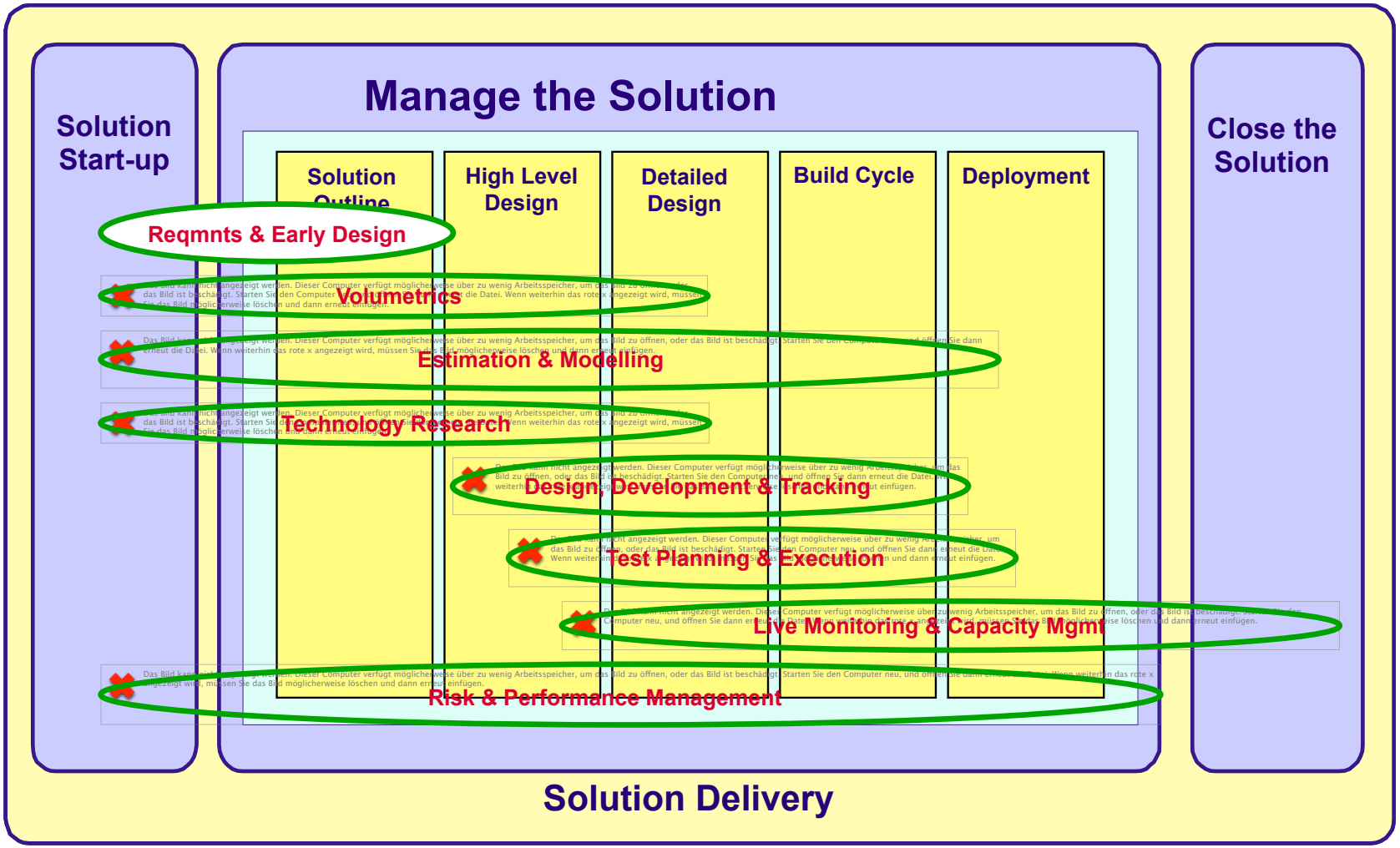


Must have adequate throughput to avoid poor response times



Sufficient capacity is required to meet throughput requirements

Major activities a Performance Engineer executes across the project lifecycle



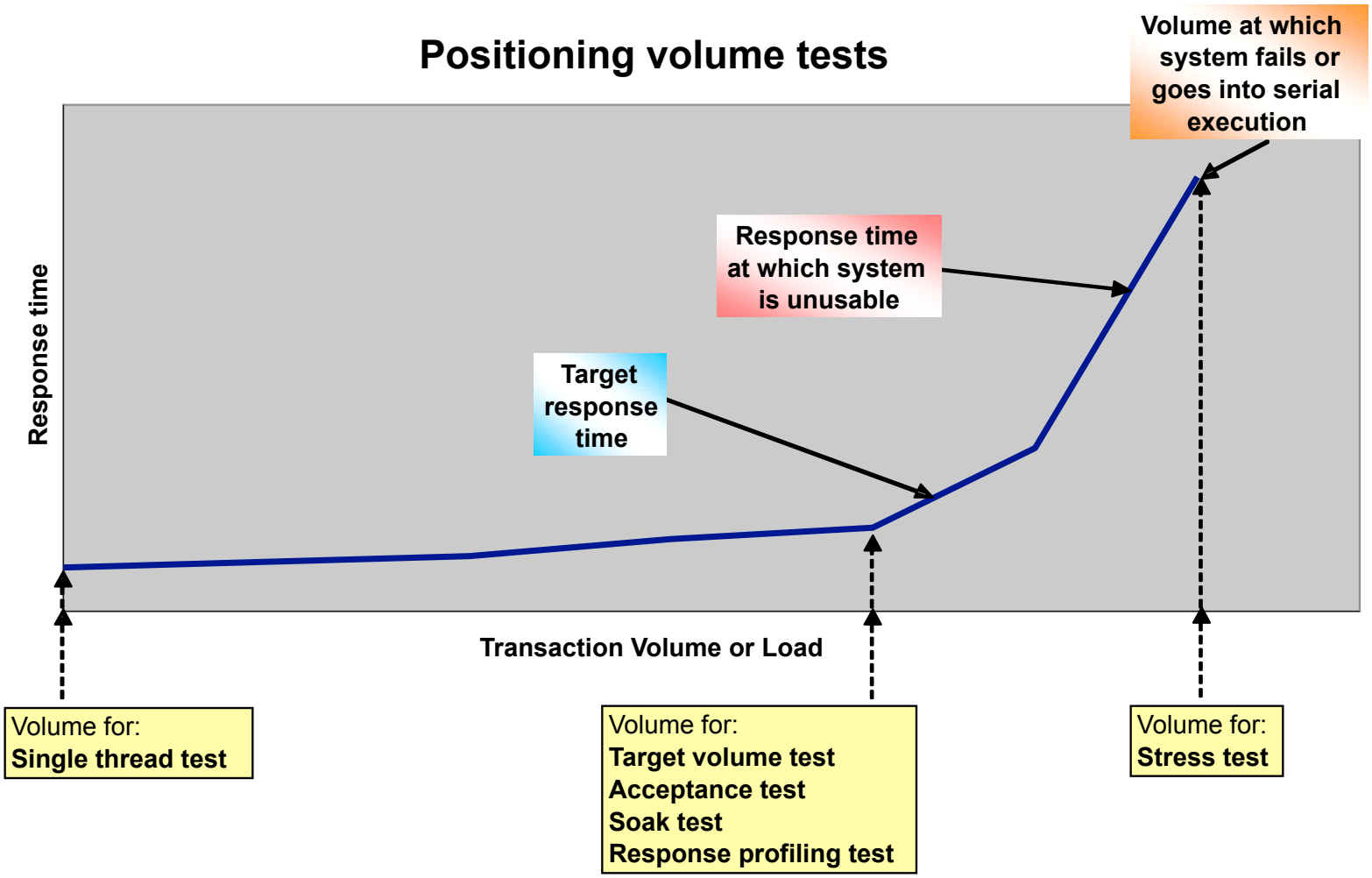
Technology research is a vital part of performance engineering

- Purpose: to determine sufficient information about components to be able to build an effective Performance Model
- Can we reduce risks by researching?
  - Has this been done before?
  - Are there any relevant benchmarks?
  - Is there a recognised Centre of Competence?
  - Have any prototypes been built? Is one required?
  - Is there an accessible reference site?

- For each key component and transaction type, identify and seek required information, e.g.
  - Structural understanding and behaviour of components
  - Parametric cost of operation (CPU required, number of I/Os, …)
- Focus deep technology research on the key components
  - New technology
  - New usage of existing technology
  - Unknown performance characteristics
  - Early performance and capacity estimates indicate component is performance critical
- Treat all sources of data with great care
  - Especially benchmarks
  - What was their exact configuration?
  - How relevant is the data for YOUR system?

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)

- 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 …)

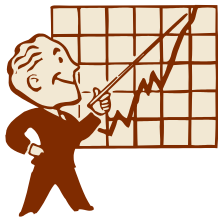
- As with initial performance modelling, the capacity plan needs cover all resources which could cause a system to perform poorly
  - Performance bottlenecks can occur at any part of the chain
  - Incentives to ensure the system makes optimum use of the available resources

- 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

Performance ::
Volumetrics

The Importance of Numbers

Performance Architects rely on **VOLUMETRIC DATA** and **ASSUMPTIONS** in order to ….



What do you do when these are vague or difficult to get?

Feed **performance and capacity models**, in order to ….

Or difficult to map down to the technical level?

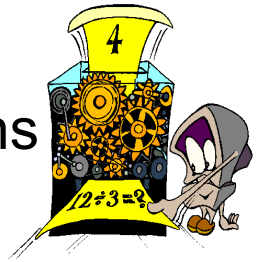
Predict system performance
- online and batch

Size systems

Evaluate & improve designs

Plan capacity

Plan testing



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

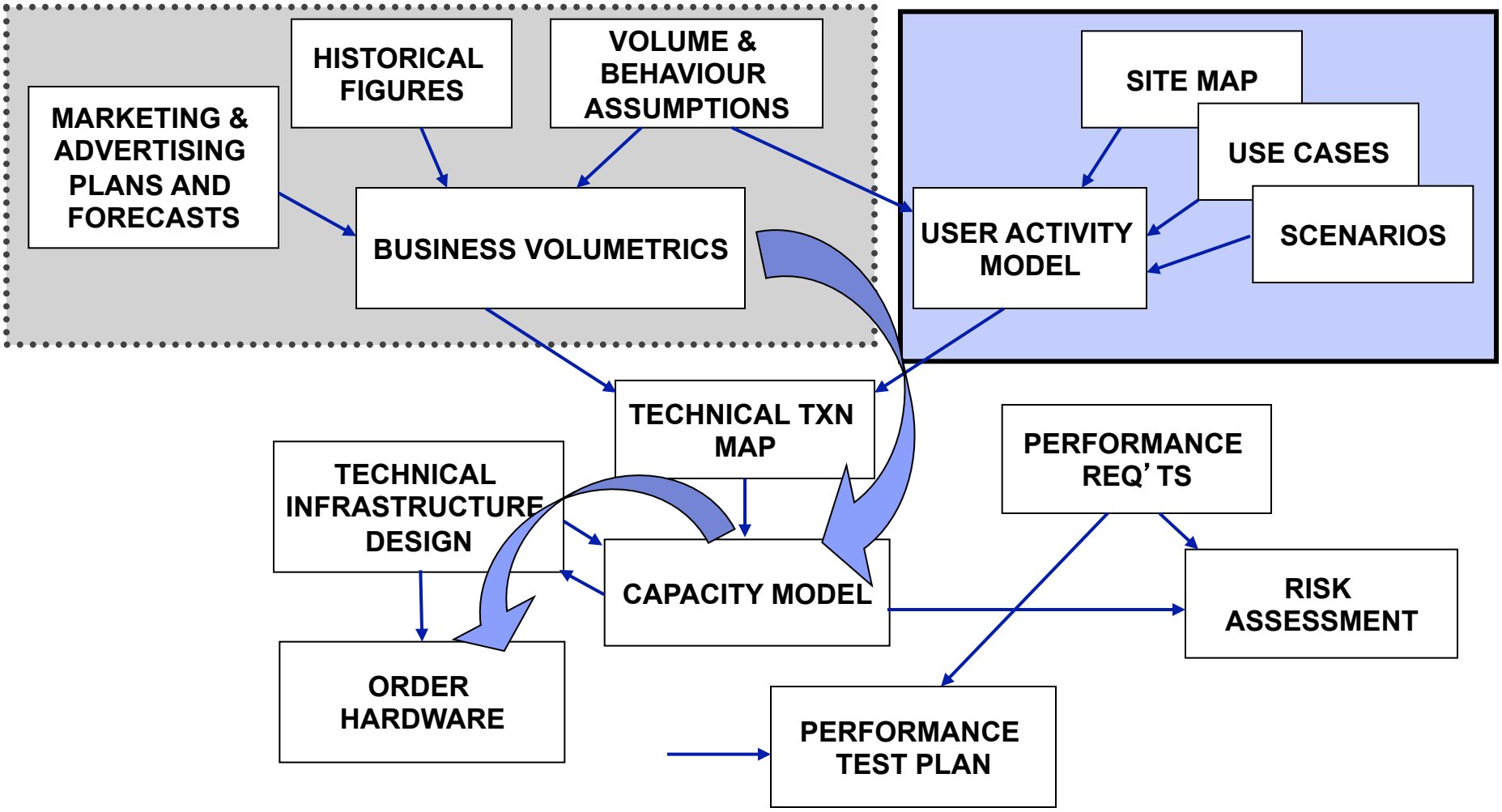
- ***"We're just about to spend £20m on advertising our new brand. How many web servers do we need?"*** **- Insurance company**

- ***"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**

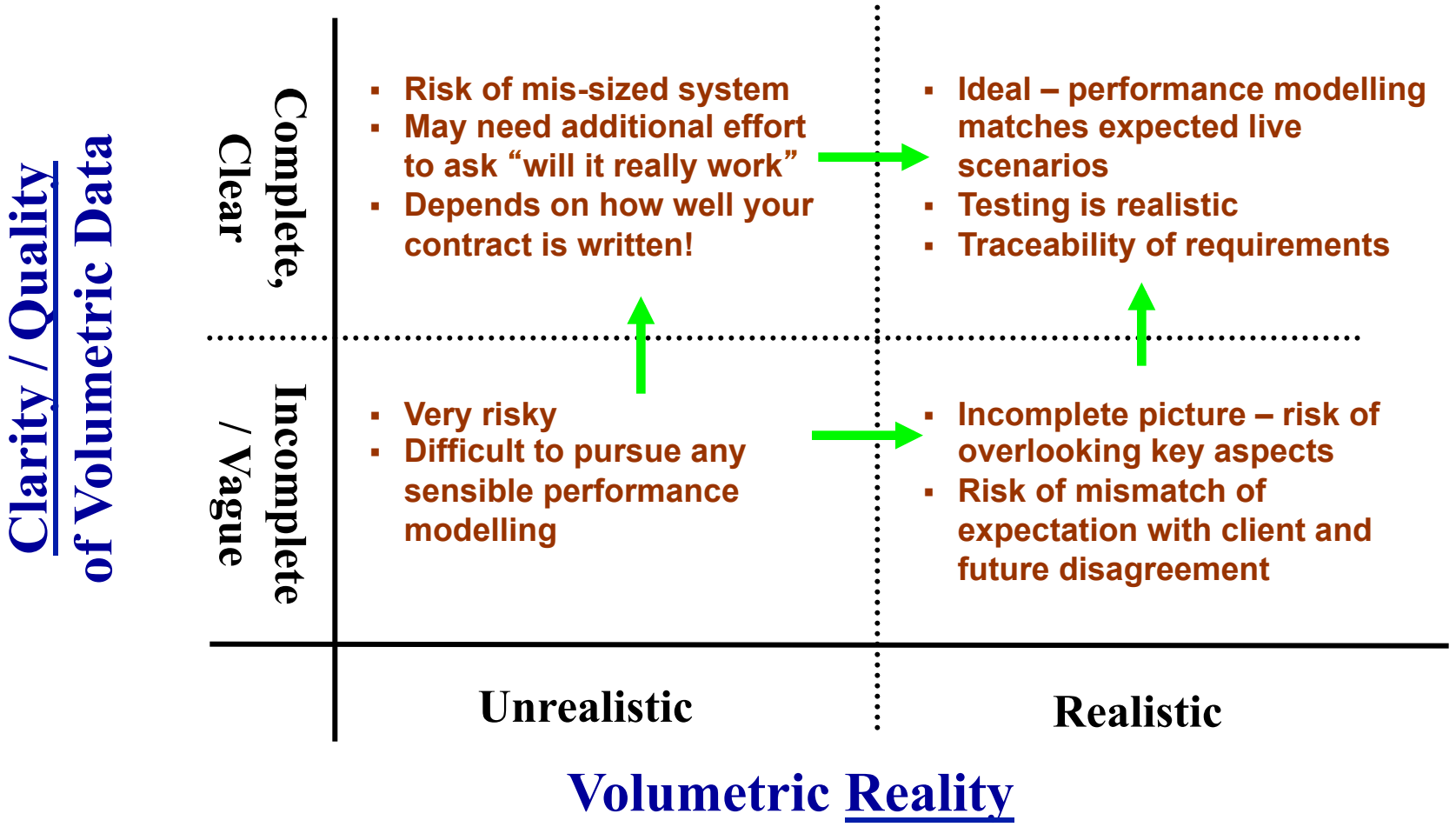
- ***"How fast is the Internet?"*** **– Offshore bank**

Volumetric data can be traced from various sources

An example "volumes map" used on an engagement



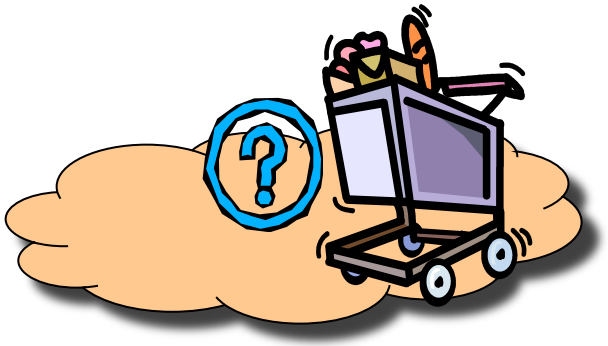
The "Volumetrics Magic Quadrant"
Relationship to Contracts and Targets



Exercise 3 - Volumetric estimation

- **Q1. Estimate, using your own knowledge and means, the number of orders per day the UK's 3rd largest supermarket has through its online shopping site**
  - **Document your assumptions**
  - **Be prepared to outline your thinking process**

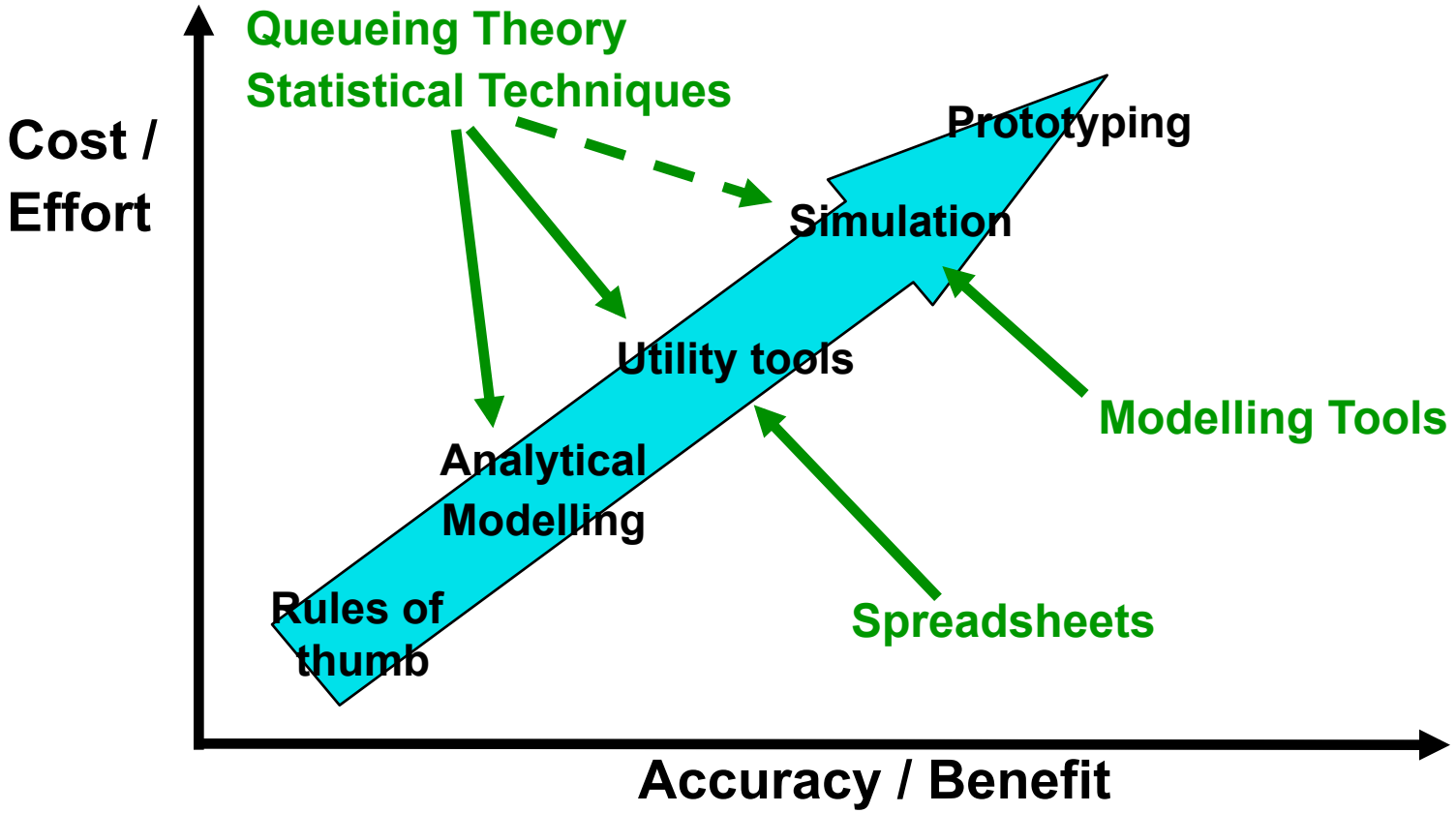
- **10 minutes**



Performance :: Estimation and Modelling

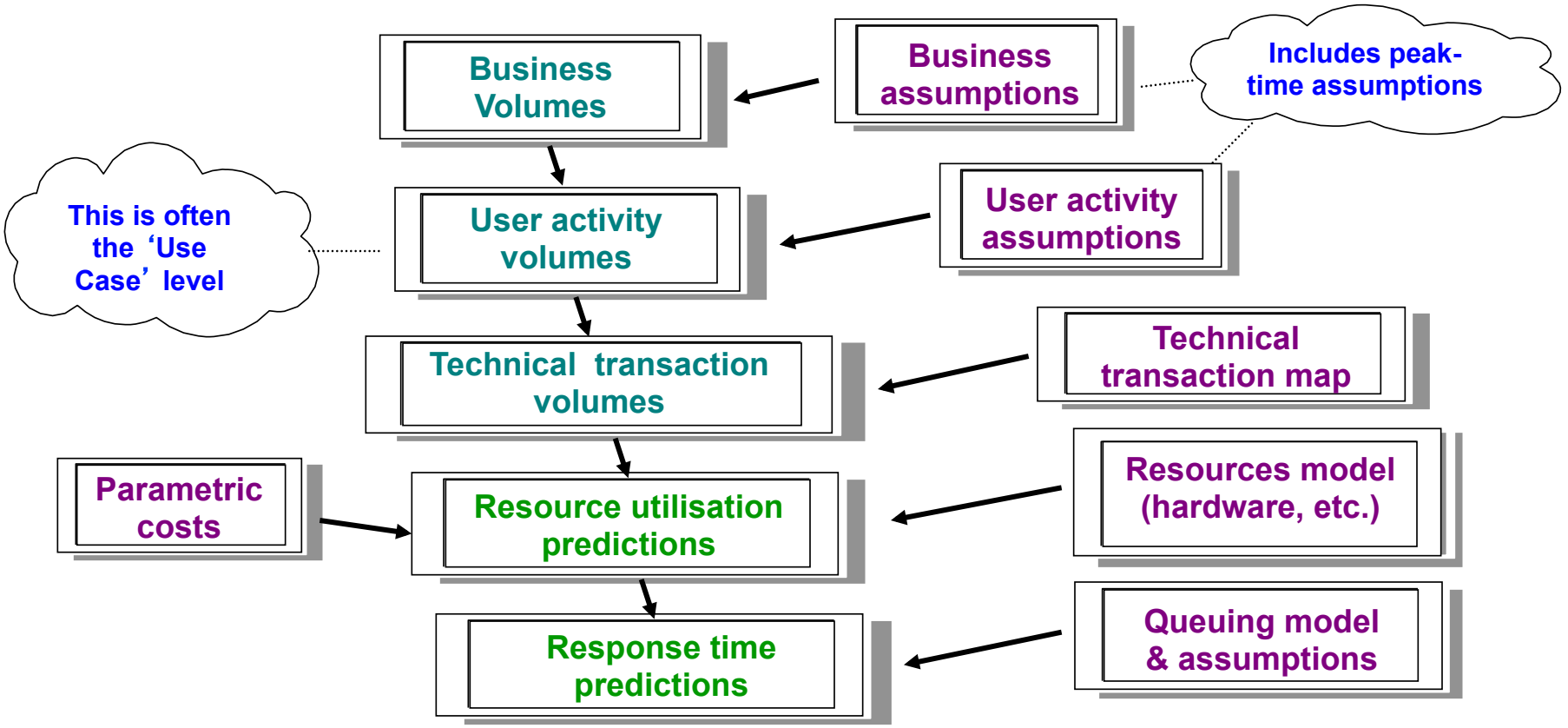
Performance characteristics of a system can be investigated in more detail by creating a model

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



Analytical performance model typical structure

This is a outline (simplified) view of the relationships of the data sets and analysis steps required to build a analytic model

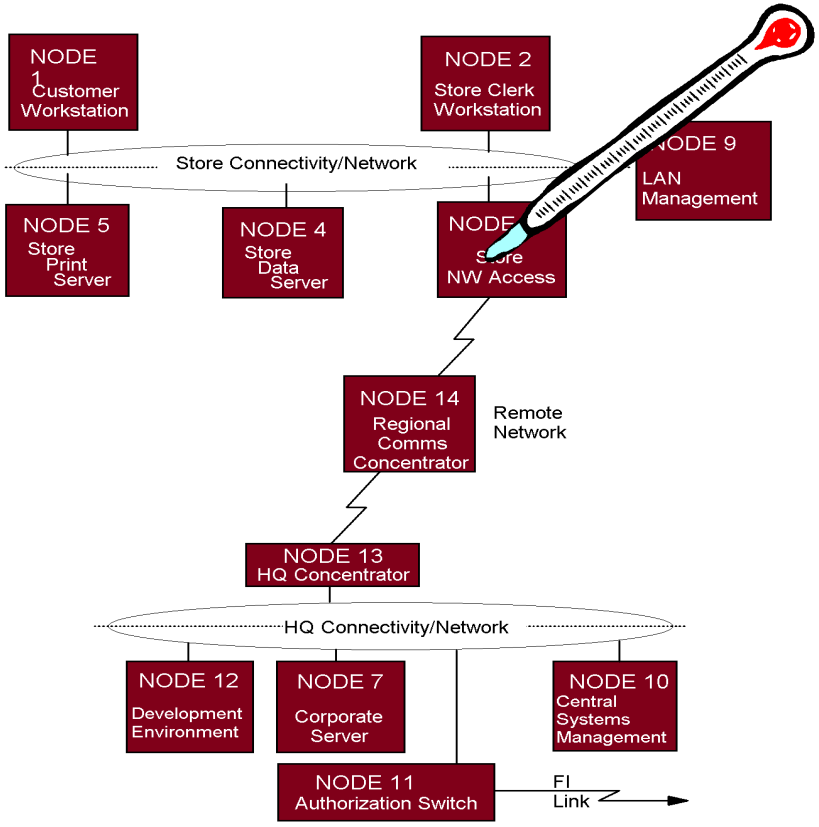


The ‘hotspot' concept is a good way of understanding where to conduct detailed performance analysis

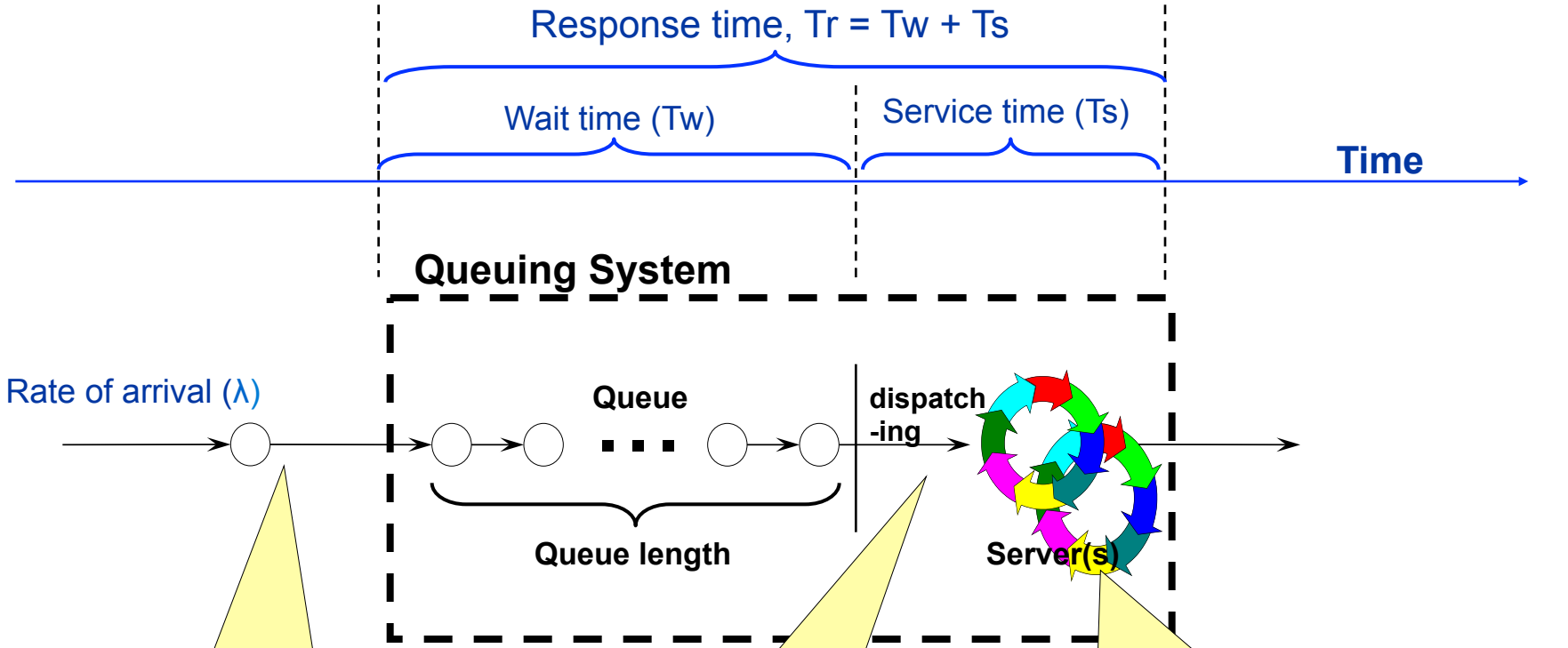
- **Typical questions you should seek the answers to …**
  - **Where are important functions (s/w execution) and data going to be located?**
  - **What rate of transactions will be required?**
  - **Are there any large volumes of data being sent over low bandwidth links?**
  - **What is the impact of multiple instances?**

    e.g. many branches to one data-centre

➢ ***Which parts of the architecture may prove to be ‘bottlenecks’?***



Characteristics and principles of Queuing Systems

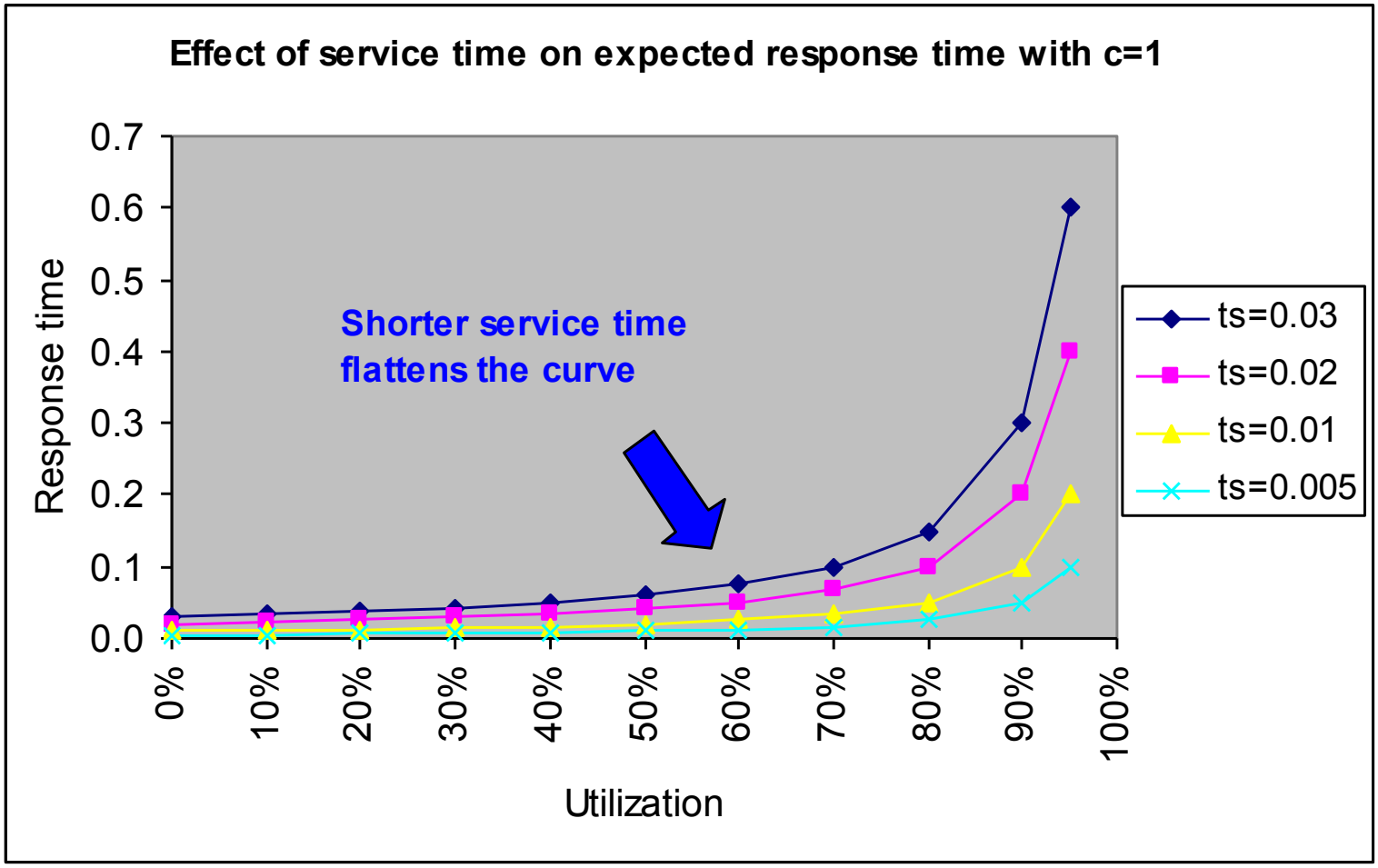


Arriving units of work:
- How many tasks enter the queue per unit of time?
- What is the interarrival time distribution?
- From finite or infinite population?

Dispatching policy:
- FCFS – standard queue
- LCFS – 'stack'
- Priorities
- 'Round robin' or straight through?

Servers and Service time:
- How many servers?
- What is the 'service time' – i.e. average to processing time for one work unit?
- What is the service time distribution?

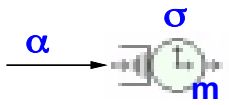
Response times degrade severely at high utilisations



Notation for major types of queues

Formal notation for queues: $\alpha/\sigma/m$, where:

- α Type of probability distribution that represents the periods between arrivals into the queue (M = Exponential, D = Deterministic, G = General)
- σ Type of probability distribution that represents the periods required to service each request in the queue (values as above)
- **m** Number of servers at the queuing center



Note: Other factors can be specified which define more advanced queue types, including:

- Buffer Size or storage capacity in the queue
- The allowed population size, which may be finite or infinite
- The type of service policy e.g. FIFO, LIFO, RR, PS

M/M/1

- Known as the 'Poisson process'
- Exponentially distributed interarrival and service times around known averages; single server
- Reasonable approximation to most single server queues
- Mathematics are manageable

$M/M/k$

- Exponentially distributed interarrival and service times around known averages
- Multiple servers
- Mathematics more complicated

Others, e.g. M/G/k

- Generalised distributions of either interarrival and service times
- Multiple servers
- Mathematics beyond most of us
- If important, consider specialist tooling or simulation

Even some simple equations can provide some useful results

Utilization Law: $U_i = X_i * S_i$

Utilization	Throughput (tps)	Average service time (s)
10%	0.5	0.2

Forced Flow Law: $X_i = V_i * X_o$

Queue i's throughput	Average # of visits to queue i	System throughput
12	3	4

Service Demand Law: $D_i = U_i / X_o$

Service demand at resource i	Resource i's utilization	System Throughput
0.2	0.4	2

Little's Law: $N = X * R$

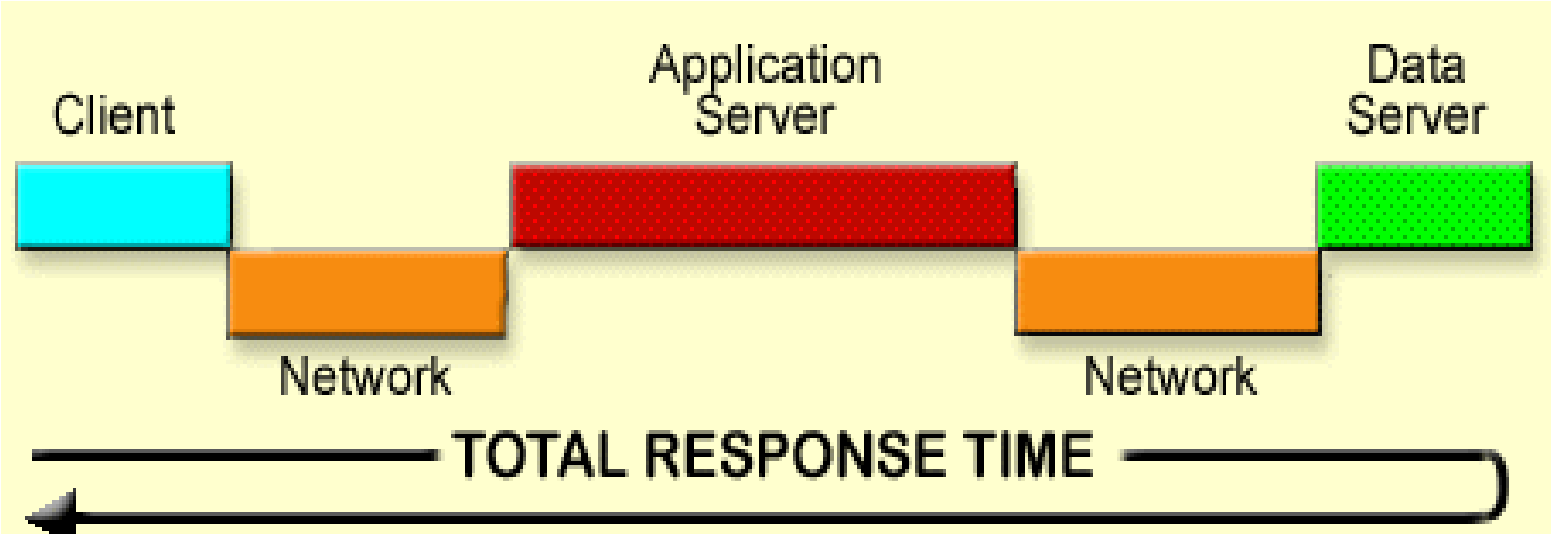
Average # in the Node	Throughput of the node (tps)	Average time in the node (s)
3	10	0.3

Example M/M/1 queue calculation

- Requests arrive at the rate of λ per second — $\lambda = 10 / s$
- Average service time is Ts — $Ts = 0.08s$
- Server utilisation U — $U = 10 * 0.08 = 0.8$ (or 80%)
- Average queuing time formula for M/M/1 queues: $Tw = Ts * U / (1-U)$ — $Tw = 0.08 * (0.8 / (1 - 0.8)) = 0.08 * 4 = 0.32s$
- Average Response time $Tr = Ts + Tw$ — $Tr = 0.08 + 0.32 = 0.40s$
- Average population in the system (Little's Law), N — $N = 10 * 0.40 = 4.0$

In actual models we need to sum response times for all components an end-to-end transaction relies on

- **Utilisation of each resource based on the total workload and workmix**
- **End-to-end response times based on multiple steps in the end-to-end transaction path**

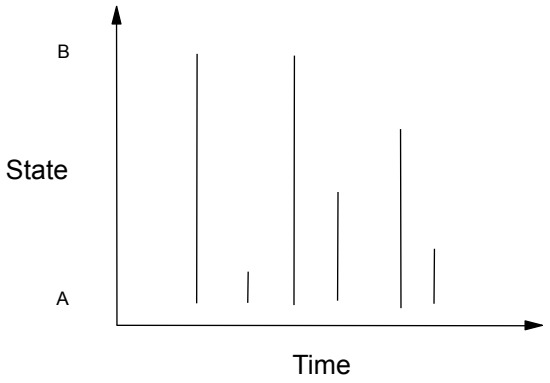


Simulation Modelling

- **A technique for modelling system behaviour in which:**
  - **the model is a *simplified representation* (abstraction) of a system**
  - **the model is *executable* and *dynamically simulates events* within the system**
  - **events and relationships within a system are processed (simulated) and *displayed over time***

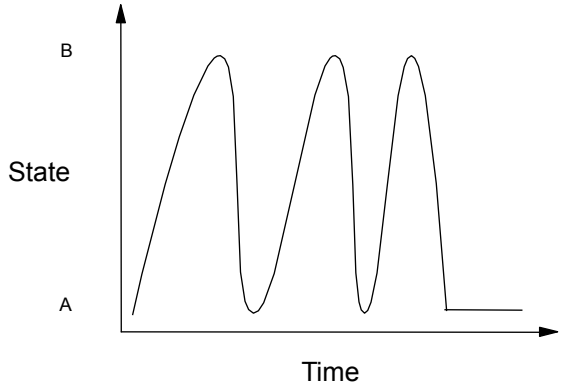
DISCRETE EVENT SIMULATION

• System behaviour modelled as a series of discrete events
• Times at which the system state does not change are skipped
• Particularly good for modelling computer hardware and software



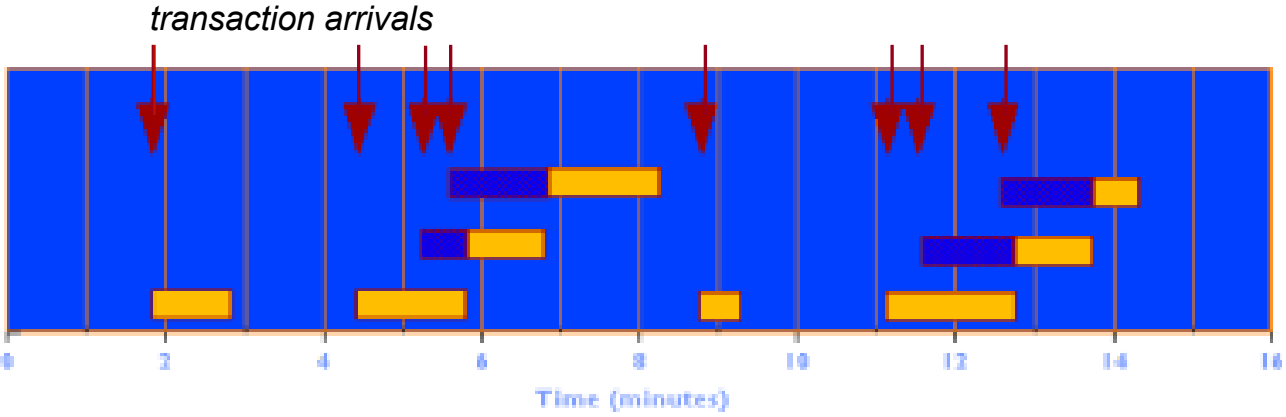
CONTINUOUS SIMULATION

• System behaviour modelled as a series of equations
• System state is updated at all times
• Particularly good for modelling growth and step-changes (for example, continuous flow chemical plants)



How does discrete event simulation work?

- **This diagram shows how a discrete event simulation of a queuing model works internally**



Event status

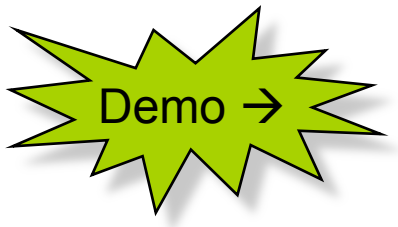
Time	Average Utilisation	Current Queue Length	Average Response Time	Average Service Time
1.8	0	0	0	0
2.8	0.36	0	1.0	1.0
4.5	0.22	0	1.0	1.0
5.2	0.33	1	1.0	1.0
5.5	0.36	2	1.0	1.0
5.8	0.41	1	1.18	1.18
6.8	0.48	0	1.29	1.10
8.2	0.57	0	1.65	1.18
.....	.....	.....	.....	.....
16.0	0.52	0	1.59	1.04

Transaction results

Transaction Number	Ts	Tw	Tr
1	1.00	0	1.00
2	1.35	0	1.35
3	0.94	0.60	1.53
4	1.41	1.30	2.71
5	0.47	0	0.47
6	1.65	0	1.65
7	0.94	1.24	2.18
8	0.53	1.29	1.82

To demonstrate the capabilities of simulation as a technique, we will now look at an example model

- The demonstration models customers in a supermarket. It shows:
  - A method of modelling a system
  - The power of a model in 'what if' analysis
- A supermarket is used as it has many of the features that we need to consider in an IT system
  - Features
    - Static resource = shopping carts
    - Server = checkout
    - Delay = time spent shopping
    - Queues for carts and checkouts
  - Output
    - Utilisation - of carts and checkouts
    - Time - overall shopping time
    - Queue length - for carts and checkouts



- 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)

Application to real IT Systems

- We use the same concepts to model an IT system
  - Static resources = memory, thread pools, …
  - Server = processor time
  - Delay = wait time (e.g. disk I/O not modelled)
  - Queues = for resources and servers
- Output
  - Utilisation - resources and servers
  - Time - overall response time
  - Queue length - for resources and servers
- However, a general purpose package has significant costs:
  - Steep learning curve
  - Needs detailed understanding of problem
  - Have to model all elements from scratch
  - Needs careful calibration
- Commercial simulation modelling tools provide significant usability
  - They are adapted to the problem space
  - Provide pre-canned, pre-calibrated modelling components
  - e.g. HyPerformix, OpNet

Simulation modelling has significant advantages … but beware …

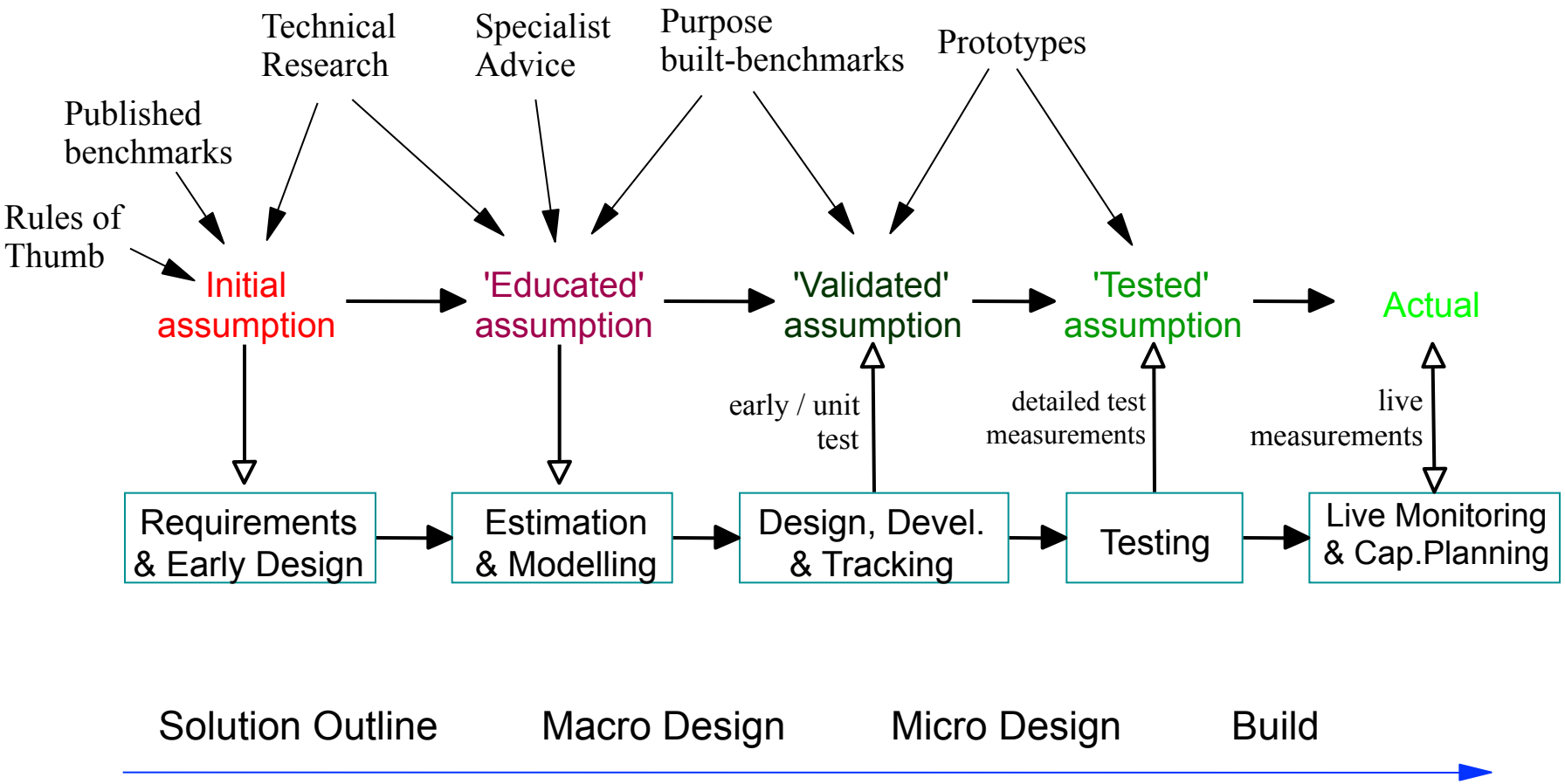
- Provides a safe environment in which to understand the effects of change (an environment for experimentation)
  - Parameterise models to ask any number of "what-if" questions
  - E.g. Test out different placement and configuration options
- Powerful and flexible modelling capabilities
  - Model complex 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

Exercise 4 - Estimating system performance

- **Inputs:**
  - **'AmGro-from-Home' Case Study document (handout)**
  - **Consider the 'Year 2' Scenario only**
- **Q1: Calculate the likely rate of the technical transaction "Serve HTTP request" in the peak period**
- **Q2: Estimate the utilisation of the web ('Presentation') server node (PN5)**
  - **How many such nodes will be required?**
- **Q3: Estimate the end-to-end response time for the user 'Search/Browse for Item' transaction**
  - **Complete the response time breakdown table at the end of the document**
- **Q4: Given your results, what recommendations would you make for the design of either the infrastructure or the AmGro-from-Home Order Management Application?**

- **45-60 minutes**

Assumptions traceability
The Technical Assumptions and Metrics 'Lifecycle'



Summary

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*