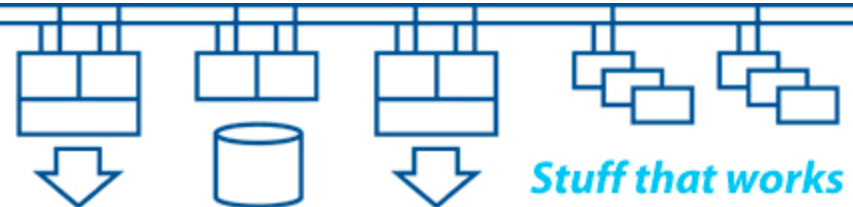


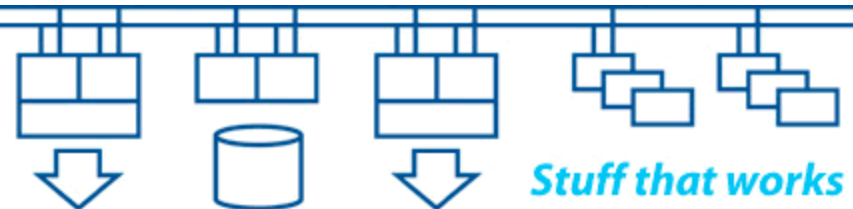
UKCMG - EuroTEC 2009

Performance testing and monitoring for mission-critical systems

Colin Butcher

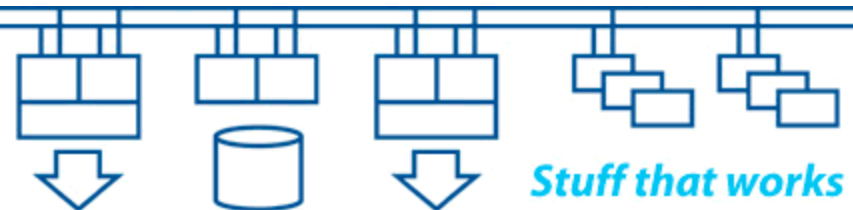


- Why is understanding performance so important in mission-critical systems?
- How does performance affect availability?
- What can we do to test a system before it goes into production?
- What can we do to test changes to a system before we implement them?
- What can we do to monitor a system once it's in production?

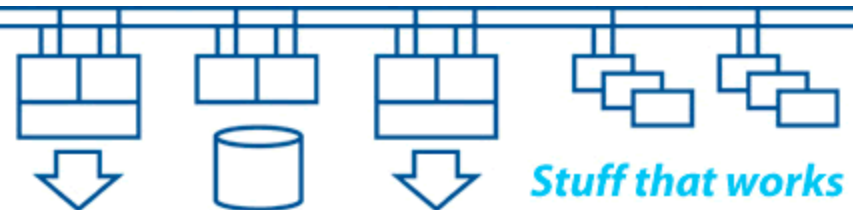


Mission critical systems need to be able to:

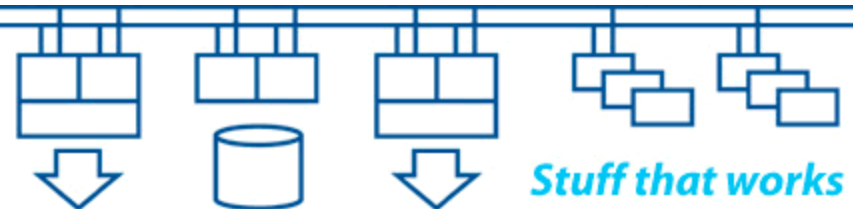
- Survive failures (resilience and failover)
 - Survive changes (adapt and evolve)
 - Survive people (simplify and automate)
 - Never corrupt or lose critical data (data integrity)
-
- Requirements never remain static over an extended period of time, so we need to be able to make changes during the operational lifetime of the system
 - Circumstances change, so we often need to be able to extend the operational lifetime and scope of a system



Cause of Outage:	Planned (Maintenance)	Unplanned (Failure)
Hardware	?	?
Operating System	?	?
Network Layer	?	?
Layered Products	?	?
Application Software	?	?
Application Data	?	?
Environment	?	?
People	?	?



- How long have we got?
- How much data can we afford to lose?
- How long have we got before we need to be ready for the next failure?
- How would we like the system to fail?
- Nothing happens instantaneously - there is always a “state transition”



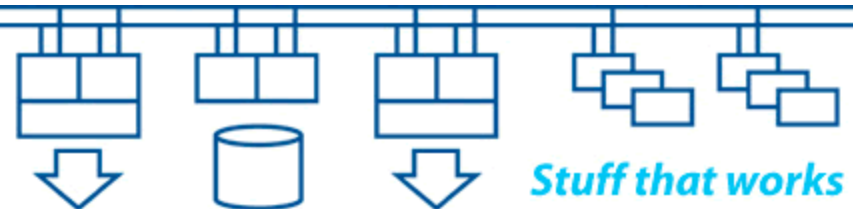
Availability:

- Probability of system being available for use at a given instant in time within the 'operational window'
- Function of both MTBF (reliability) and MTTR (repair time)

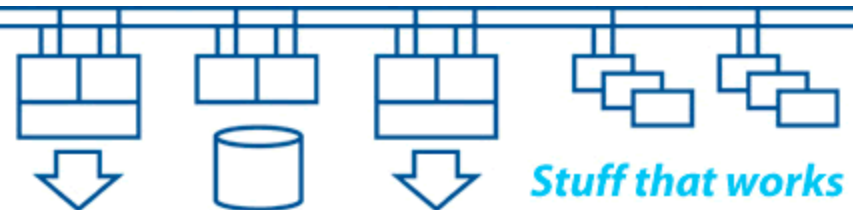
Performance:

- Performance issues are often the cause of transient system failures and disruption
- The systems have to have sufficient capacity and performance to deal with the workload in an acceptable period of time under normal, failure and recovery conditions

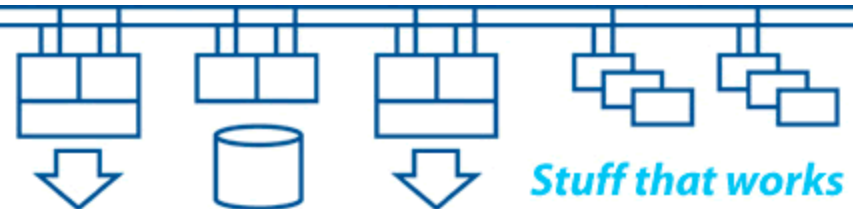
Availability is more important than performance



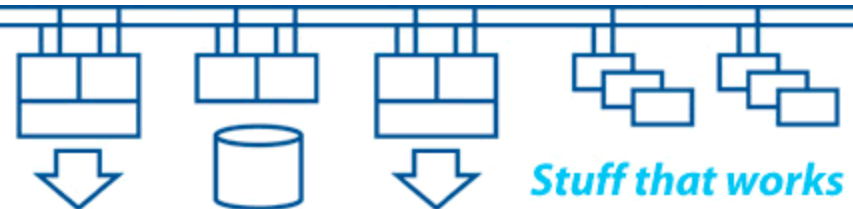
- **Bandwidth – determines throughput**
 - It's not just “speed”, it's throughput in terms of “units of stuff per second”
- **Latency – determines response time**
 - Determines how much “stuff” is in transit through the system at any given instant
 - “Stuff in transit” is the data at risk if there is a failure
- **Jitter (“div latency” or variation of latency with time) – determines predictability of response**
 - Understanding jitter is important for establishing timeout values
 - Latency fluctuations can cause system failures under peak load



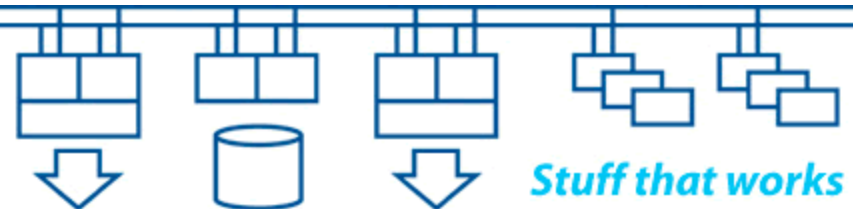
- Size systems to cope with peaks in workload
- Minimise “wait states” (caches, parallelism)
- Minimise contention for resources and data structures
- Understand the need for synchronisation and serialisation of access to data structures
- Maximise “User Mode”, minimise the other modes:
 - The fastest IO is the IO you don’t do
 - The fastest code is the code you don’t execute



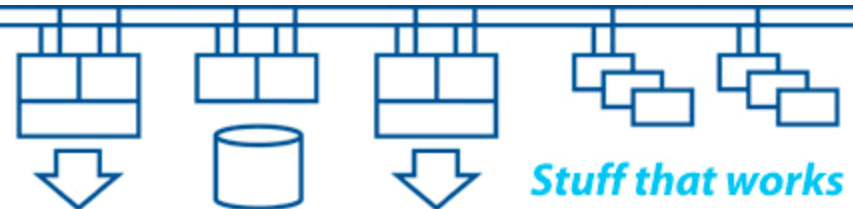
- Understand how the applications could break down into parallel streams of execution:
 - Some will be capable of being split into many small elements with little interaction between the parallel streams of execution
 - Some will require very high interconnectivity between the parallel streams of execution
 - Some will require high-throughput single-stream processing
- Understand scalability – do as much as possible once only, do little as possible every time



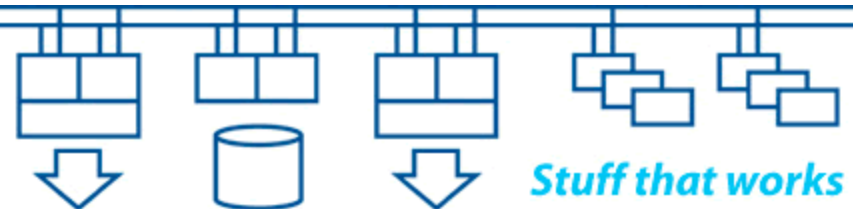
- We need to prove that service will continue with minimal disruption during failure and recovery
- We need to test scalability as well as functionality
- We need to test every aspect of the system and surrounding infrastructure under normal, failure and recovery conditions
- How will we generate a realistic load for testing?
- How will we instrument the system and infrastructure?
- We need to regularly rehearse and test our procedures and plans to ensure that we stay current



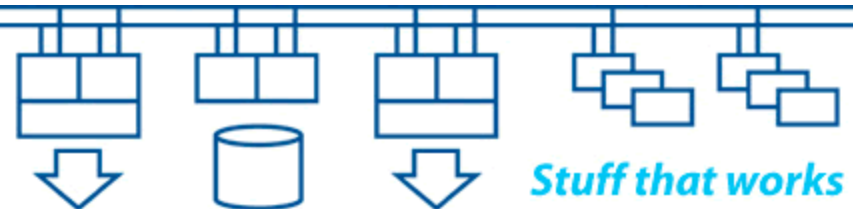
- Mission-critical systems hardly ever fail, so we need the people responsible for its operation to have a good understanding and ‘feel’ for the way it works
- We need to find out how the system behaves when it starts to fail
- We need to know the ‘warning signs’ of incipient failure
- We need to know how to return the system to its normal operational state without data loss or data corruption
- We must have a representative offline test environment



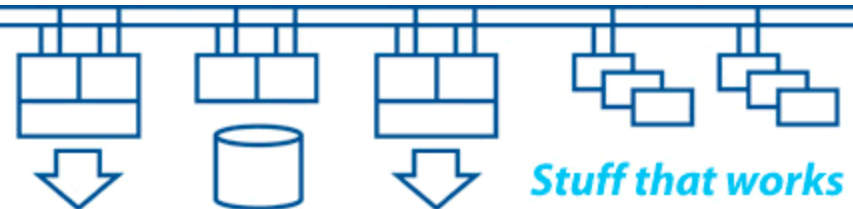
- Physical environment (power, cooling, etc.)
 - Hardware (revision levels, labelling, etc.)
 - Firmware versions
 - Operating system updates
 - Network infrastructure and configurations
 - Storage infrastructure and configurations
 - Database versions and configurations
 - Application software
-
- Interactions
 - Interoperability with mixed versions
 - Upgrade and backout actions
 - Data conversions



- How do we instrument the system?
 - Application level
 - Operating system level
 - Data storage level
 - Network infrastructure
- Time synchronisation
- How do we generate a typical workload?
- How do we generate representative data sets?



- Where can we do the testing?
 - Production environment
 - Pre-production Test environment
 - Pre-delivery Test environment
- How might a problem show up – and when?
- How can we find a problem, eg: data corruption?
- Can we recreate a problem in a test environment?
- Continual monitoring and event logging is essential
- Knowledge of the whole system is essential



Thank you for your participation

Colin Butcher

