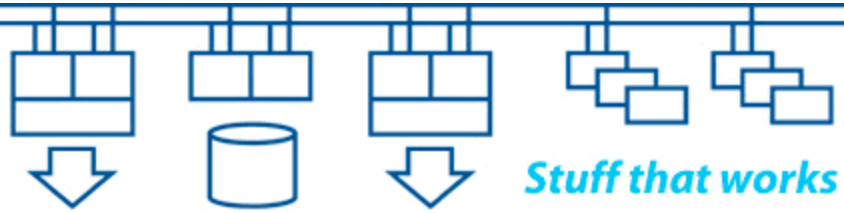
BCS Birmingham Branch – 19th July 2011

# Delivering mission critical systems

Colin Butcher

www.xdelta.co.uk





# Part 1 – Mission-critical:

- What are mission-critical systems ?

  Systems that someone somewhere relies on to get something done, without data loss or corruption and without stopping working when they need it to work.

- What makes them different from any other system ?

  Nothing! We just need an appropriate level of care and attention to detail.

- Objectives and Architectural design:
  - Have clear objectives. Think ahead as far as you can. Have a well-structured systems architecture. Understand the constraints. Focus on the core functions. Implement them as well as is possible.

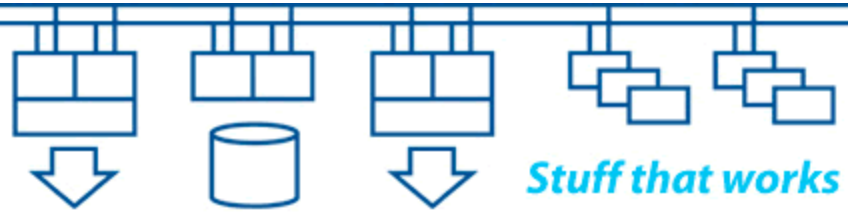
- Project leadership:
  - Ensure that everyone involved maintains a consistent understanding of the project. Plan ahead as best you can.

- Budget and Schedule:
  - They have to be appropriate for the problems you're trying to deal with. Don't set them first!

- Business Continuity:
  - It's not just the systems – it's everything
- Disaster tolerance:
  - Surviving major site outages without loss of service
- High availability:
  - Surviving failures at a site without loss of service
- Disaster recovery:
  - The process of restarting sufficient systems to run the business after loss of service, typically from another location.







Mission critical systems need to be able to:

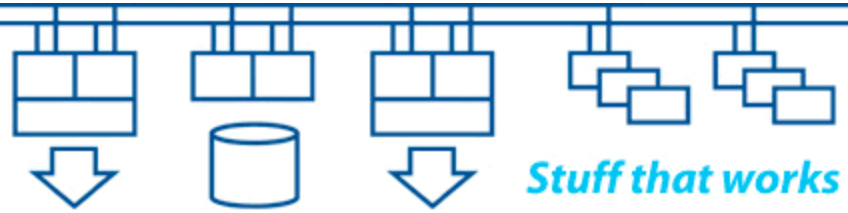
- Survive failures (resilience and failover)
- Survive changes (adapt and evolve)
- Survive people (simplify and automate)
- Never corrupt or lose critical data (data integrity)

- 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

- We're trying to reduce the probability of failure
- We're trying to minimise the consequences of a failure
- We're looking for critical components / people
- We're looking for critical stages during the project
- We need to understand how systems fail and what failure looks like when it happens
- We're paranoid about our data





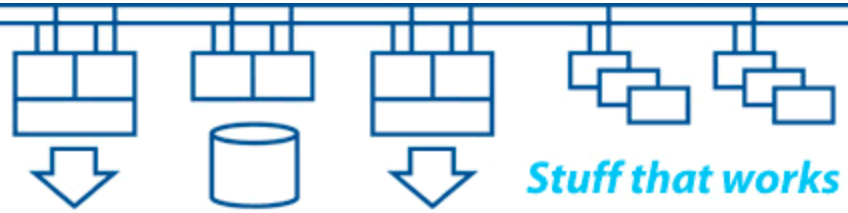


- Safety-critical systems (especially real-time monitoring and control systems such as air traffic control) require exceedingly high levels of availability. They also have to be fail-safe in order not to endanger lives.

- True 24x365 mission-critical systems are fairly rare. With these there is no "downtime window" to take backups, fix faults or to make changes. So, whatever you do has to be done "live" – and very carefully!







# "Survivability test" (1)



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

# "Survivability test" (2)

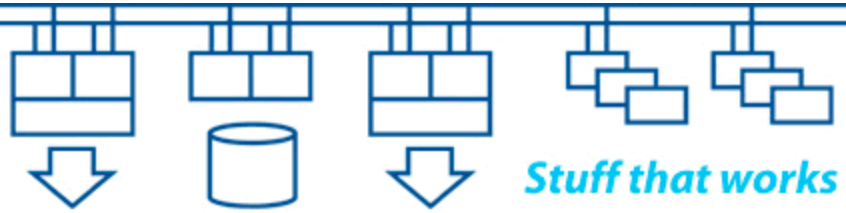
- How long have we got ?
- How much data can we afford to lose ?

- How can we model the whole system ?
- How can we predict the overall availability ?

- The closer you get to 100% uptime the more expensive a satisfactory solution will become







## RPO = Recovery Point Objective

- How much data can we tolerate losing ?
- How quickly do we need to react to a failure ?

## RTO = Recovery Time Objective

- What level of service outage can we tolerate ?
- How quickly do we need to recover ?
- How quickly do we need to be ready to deal with a subsequent failure ?

# Part 2 - The design process:

- Design is an abstract, yet structured process
- Start with the "ideal design"
- Know what constraints you have to deal with
- Build the whole system on paper first
- Understand the details and complexities
- Consider the possible interactions and plan ahead to avoid problems
- Remain flexible and adapt to changes

*"Entia non sunt multiplicanda praeter necessitatem"*

*"Entities should not be multiplied unnecessarily"*

"Keep things as simple as possible, but not so simple that they don't work"

William of Occam

- All design decisions are compromises and require you to exercise judgement
- Start by designing the "ideal system", eg: three data centres, even if we only implement one or two first
- Now consider the constraints and modify the overall design to implement it in stages
- How will you bring the new system into service ?
- Follow the data flows
- Don't make assumptions about how things will be implemented and let them constrain your thinking
- Document your decisions and the outline design

- Big decisions which have long-term implications and constraints
- Small decisions which seem big at the time
- There will be requirements and constraints you don't yet understand or know about
- Need to design in the ability to make changes
- Establish meaningful naming conventions

| | Before | Now | After |
|---|---|---|---|
| Environment | | | |
| System | | | |
| Component | | | |

See www.triz.co.uk for a lot more information!

# Part 3 – Naming conventions

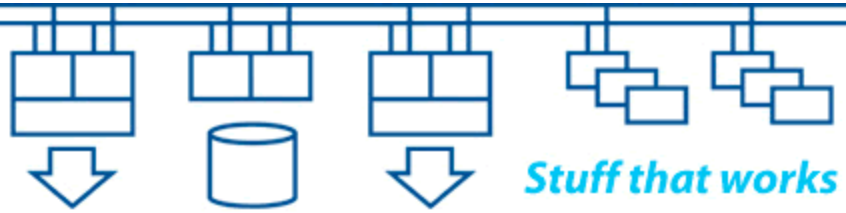
- Need to consider how things will change over time and how we can avoid changing everything later
- Node names, network addresses, etc. are almost impossible to change without major disruption
- Use a sparsely populated name space with room for expansion

# Entity naming

- Choose simple names based on function, not location
- Have a structured way of creating names and allow for future expansion and change
- Separate the different environments (eg: Development, Test, Pre-Production, Live Production)
- Do not embed real names of physical entities into the application
- Use logical names to map from functional entities in the system to physical entities – this allows change later with minimal impact



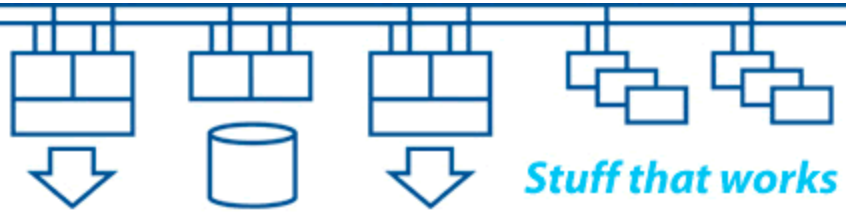




- Separate the data types, eg: bootable system, application executables, static data, dynamic data
- Allocate LUN values, identifiers, volume names, device names etc. in a logical manner
- Inside the storage array (eg: EVA Vdisks, Snapshots, Mirrorclones, Hosts etc.), have a convention that matches the naming used by the attached systems



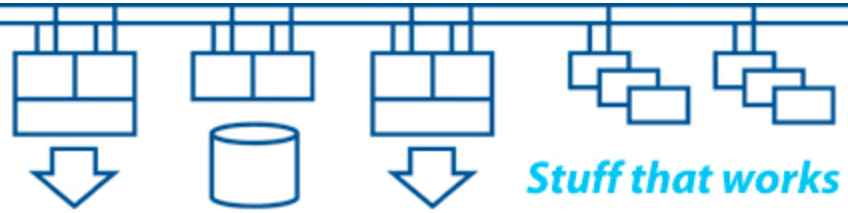




- Separate the data flows, eg: iLO traffic, inter-site storage traffic, user access, etc.
- Use 802.1Q VLANs to segment the network and separate out different types of traffic flow
- Allocate protocol addresses in a logical manner
- Map network addresses to functions, not machines





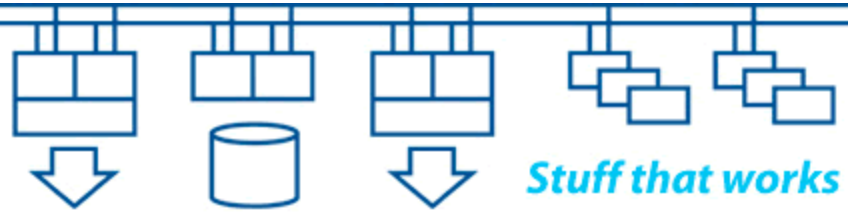


- Data file internal structures may change over time, so minimise risk of data corruption by tagging the file structure with a version identifier
- Packet / buffer formats for exchanging data between application components may change over time
- Application components may need to move between machines over time

# Part 4 - Availability and Performance:

- What is Availability ?
- Characterising Performance
- Scalability and Parallelism
- State Transitions





## 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

- 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 ?

# The systems "flight envelope"



- What are the systems required to do ?
- What are the consequences if the systems fail ?

- What happens if you push beyond the limits ?
- How far from the edge are you ?
- How do you know ?

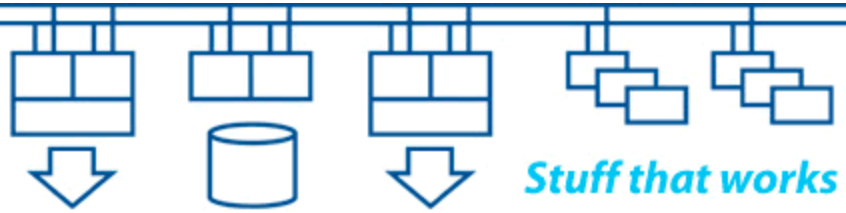
- What evidence can we look at ?
- What comparisons can we make ?
- What can we measure ?

- 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







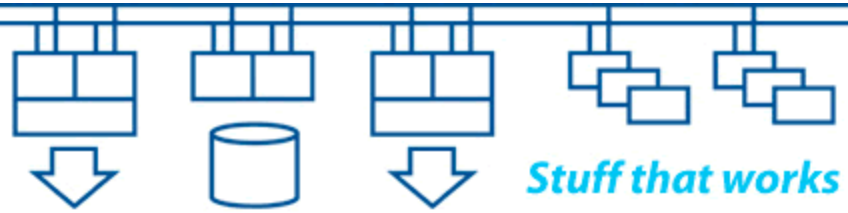
- Jitter ("div latency" or variation of latency with respect to time) – determines predictability of response
  - Understanding jitter is important for establishing timeout values
  - Latency fluctuations can cause system failures under peak load

- Contention and saturation – running out of capacity
  - Queuing theory predicts that latency will get worse and become more unpredictable as you approach the point of saturation
  - What else are we sharing our capacity with and what control (if any) can we have over it ?

- Increasing the capacity of the overall system:
  - "Scale up" or "vertical scaling" refers to increasing capacity by adding more resources to a machine or buying a bigger machine (CPU count, memory, I/O adapters, etc.)
  - "Scale out" or "horizontal scaling" refers to increasing capacity by adding more machines (eg: blades)
  - It depends on how your workloads break down into parallel streams of execution and on what level of availability you need to achieve







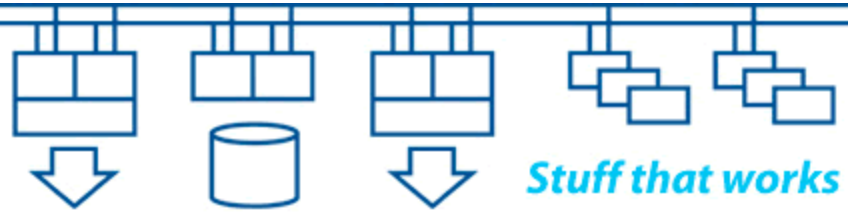
- Understand how your workload could break down into parallel streams of execution:
  - Some will be capable of being split into many elements with little interaction
  - Some will require very high levels of interconnectivity and interaction
  - Some will require high-throughput single-stream processing

- Size systems to cope with peaks in workload
- Minimise "wait states" (caches, parallelism)
  - Don't make it go faster, stop it going slower
- 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

- Which parts of the system are mission-critical ?
- Which parts of the system are safety-critical ?
- What kind of failure do we prefer ?
- What state transitions occur during both failure and recovery ?
- What happens if there's another failure part way through a state transition during a failure ?
- How can we recover from a failure without data loss or data corruption ?
- How can you make changes while it's working ?







# State transitions (1)

- Nothing happens instantaneously - there is always a "state transition"
- What possible states can a pair of machines be in ?

- How long do state transitions last for ?
- What can we do while a state transition is in progress ?

- Can we ensure that there are no timing windows or other flaws in the design ?

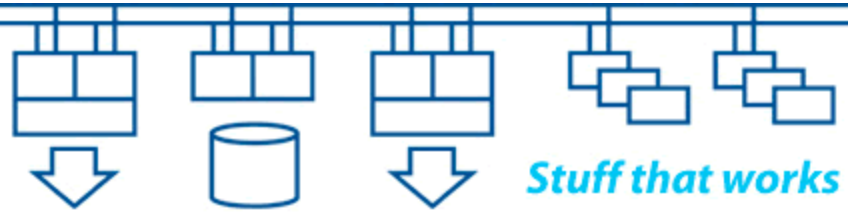
# State transitions (2)

| A | B |
|---|---|
| Off to Master | Off |
| Master | Off to Standby |
| Master to Off | Standby to Master |
| Master to Off | Off to Master |
| Master to Standby | Standby to Master |
| Master to Hung | Standby to Confused |
| … | |

- Effects of distance on network and storage protocols
- Symmetric or asymmetric operation – how good is your "crystal ball"?
- Avoid booting across inter-site links
- Remote access for management and operation
- Centralised (and duplicated) monitoring and alerting
- Quorum and voting schemes
- Data replication schemes
- Full environmental monitoring for lights-out sites
- Avoid automation of decision making when a site fails







# Part 5 - Risk:

- What is Risk ?
- Characterising risks

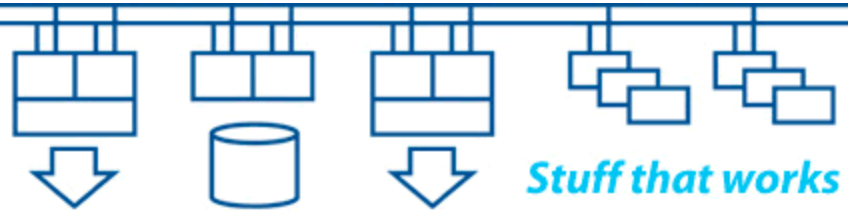
Risk is a fact of life. We have to deal with it as best we can – or at least well enough for the circumstances we find ourselves in.

Think about both project management and technical design as part of systems engineering, then apply techniques from other engineering disciplines to help us analyse the situation and guide our thinking.

Disaster-tolerant systems aim to minimise the risk of loss of service and loss of data as much as possible.







# The risk continuum

- What is the probability of a situation occurring ?
- What is the impact if that situation occurs ?
- What are the long-term consequences ?

- Most projects handle medium risk well enough
- Many projects over-specify to cater for what are in fact low risk issues
- Some projects under-specify and fail to cater for what are in fact high risk issues

- How can we start to identify what the risks might be ?
- How can we look for single points of failure ?
- How can we look for modes of failure ?
- How can we analyse how failures will ripple through ?
- Can we identify specific scenarios of interest ?
- Can we test all the conditions ?
- What happens to our data ?

There are many techniques which have evolved over the years and there are tools to help you apply them.

- Reliability Block Diagrams (RBD)
- Fault Tree Analysis (FTA)
- Failure Modes, Effects and Criticality Analysis (FMECA)

These (and many other) techniques can be applied with software tools available from a number of vendors.

# Part 6 – Operational service:

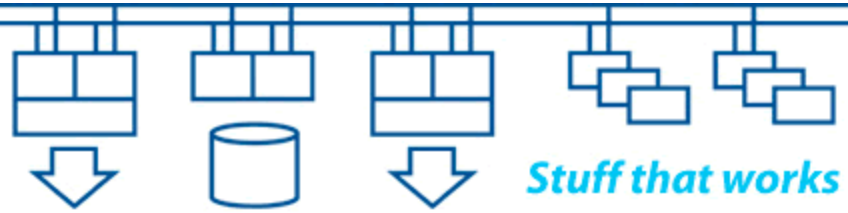
- Transition to the new systems
- Management
- Testing
- Monitoring
- Fault-finding and rectification

- Need to minimise risk of data loss
- Need to minimise risk of loss of service
- Need to migrate user connectivity
- Need to migrate live data, historic data (eg: backups for subsequent retrieval) and potentially convert data files

- How can we split transition into manageable steps ?
- Is anything a one-way step ?
- What is the least we have to do during cut-over ?
- How much can we get done in advance ?
- How could we revert to the original system ?







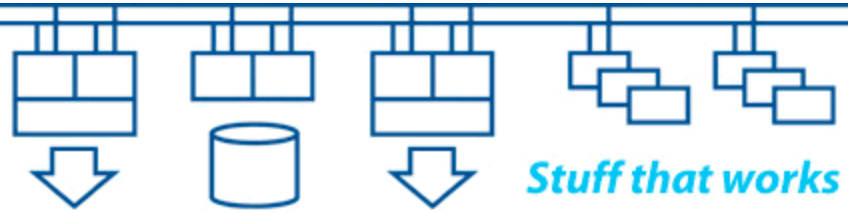
- Mission-critical systems hardly ever fail, so we need the people responsible for its operation to have a good understanding and 'feel' for the way it works

- 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 need a representative offline environment







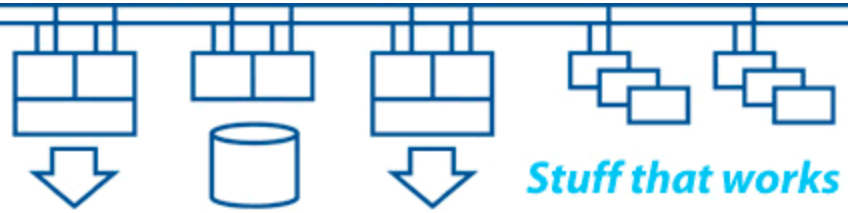
# Clear identification of equipment

- Colour-coding
- Labelling
- Lighting
- Physical layout of equipment within racks
- Physical connections between racks
- Cabling space and routes
- Power subsystems
- Cooling subsystems
- Access paths

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







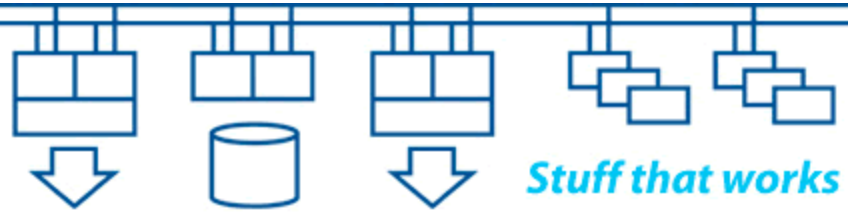
- We need to demonstrate that service will continue with minimal disruption during failure and recovery
- We need to regularly rehearse and test our procedures and plans to ensure that we stay current

- 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

- What are our acceptance criteria ?







- How do we instrument the system ?
  - Applications
  - Data storage (data files and integrity checking)
  - Operating system
  - Infrastructure (networks, storage, power, cooling, etc.)
- 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 ?

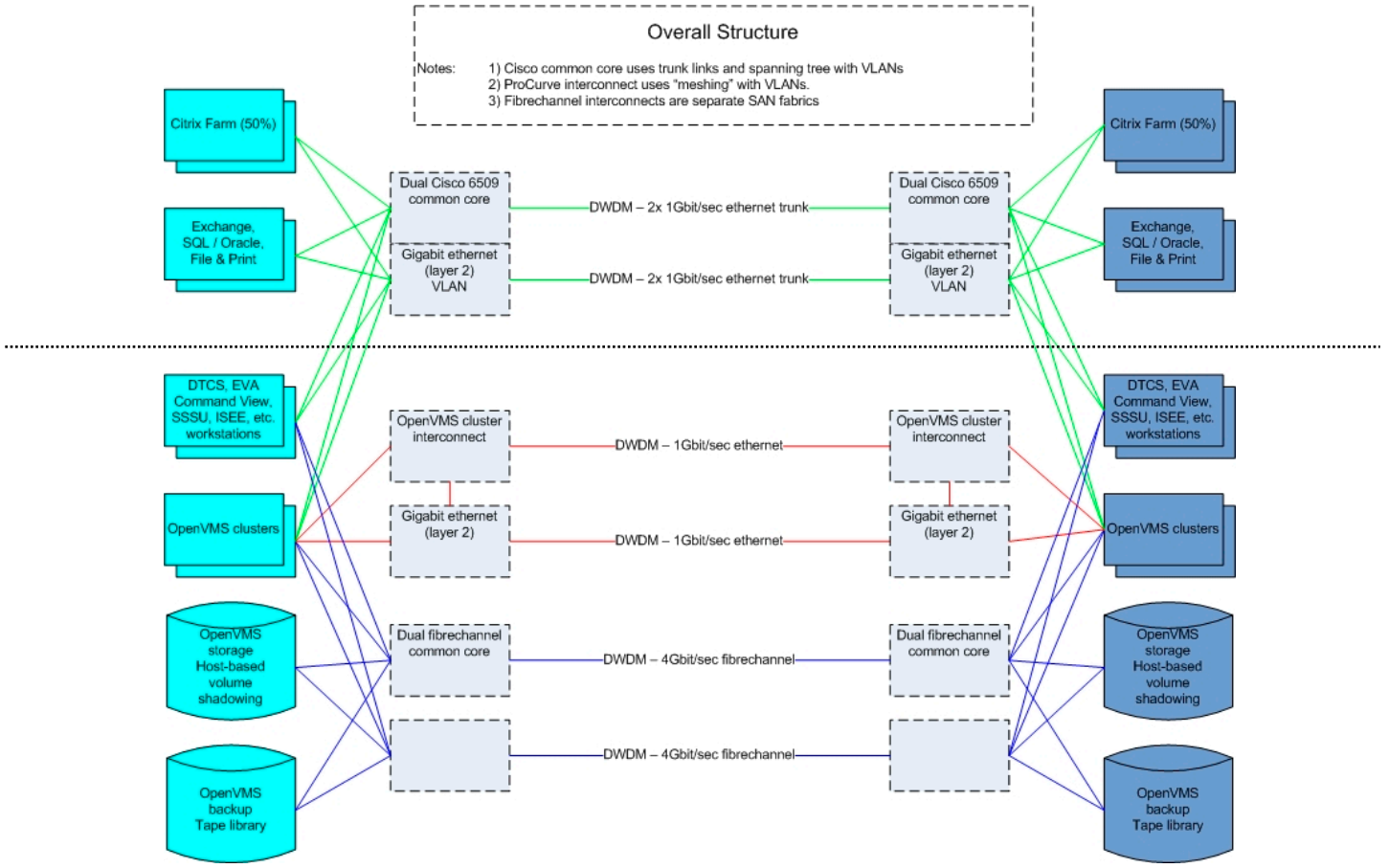
- Continual monitoring and event logging is essential
- Knowledge of the whole system is essential

# Part 7 - Example

- NHS Blood and Transplant "Pulse renewal":
- See http://www.xdelta.co.uk/news#nhsbtcasestudy

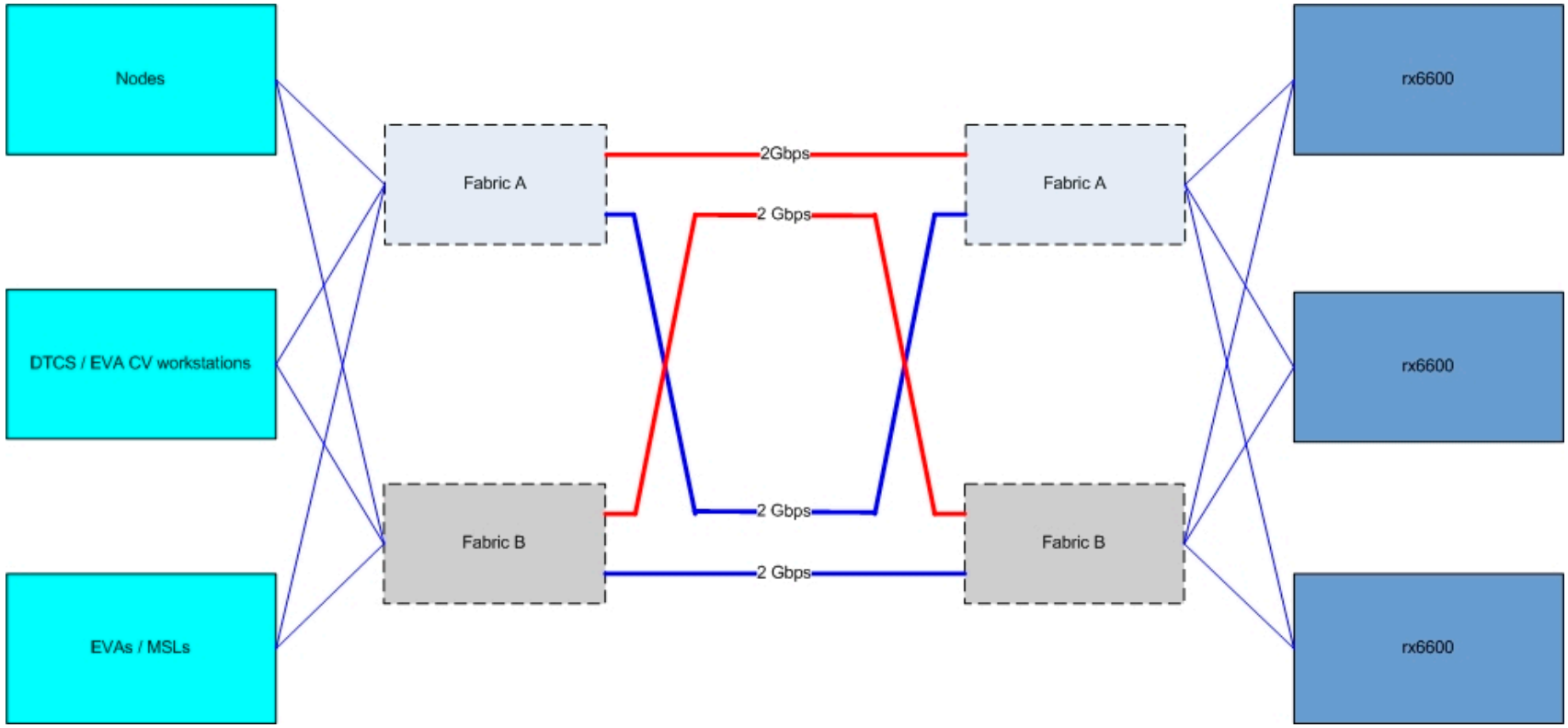
We have Production, Archive and Test environments with a shared common infrastructure, all of which is separated from the rest of the existing infrastructure.

# NHSBT Pulse: Overall structure



The systems are split up into:

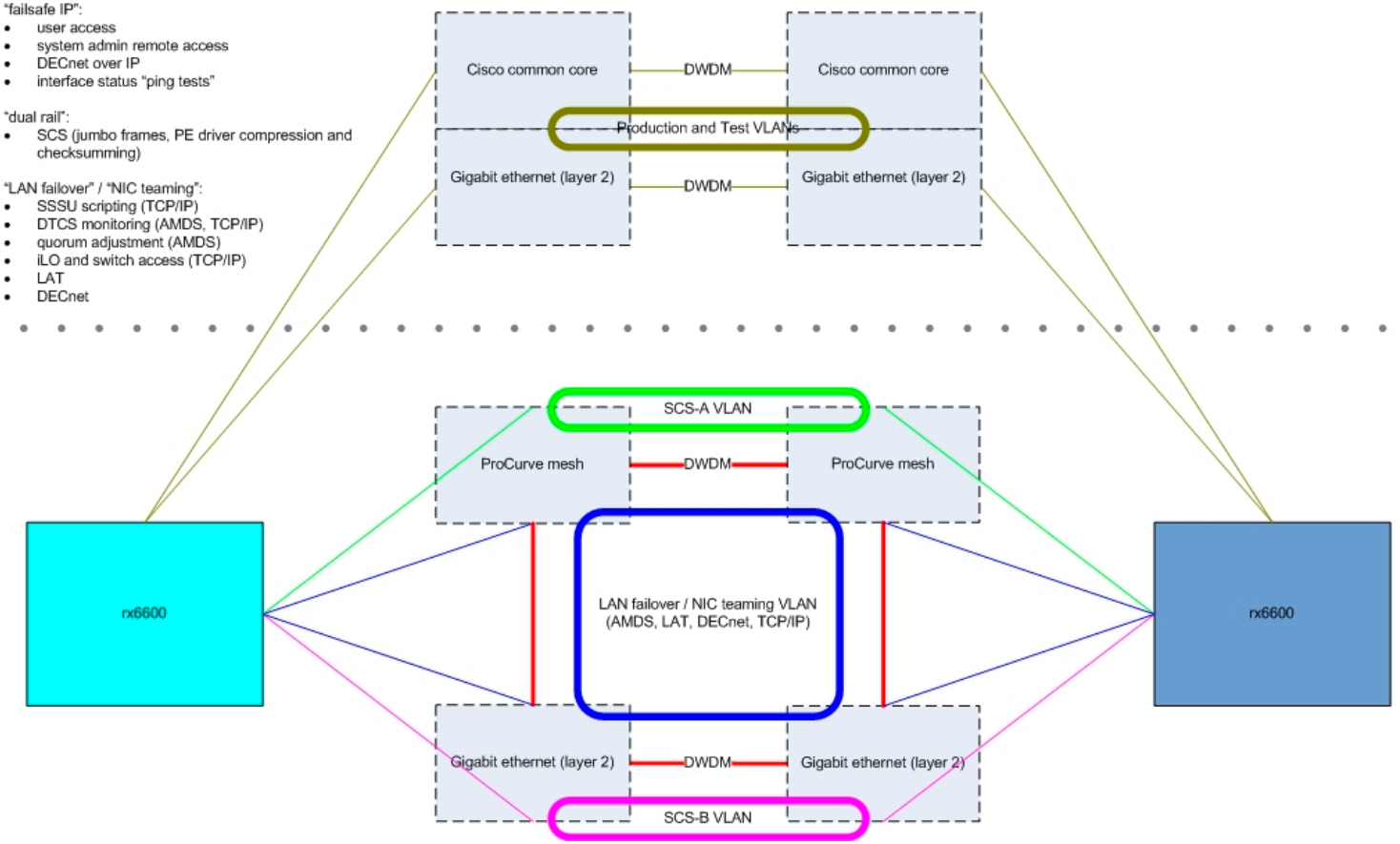
- Common infrastructure (SAN fabrics, private network interconnects etc.)
- Production environment (a split-site cluster with host-based volume shadowed storage)
- Test environment (a split-site cluster on a smaller scale)
- Archive environment (a single node at Site A)
- Duplicated monitoring and reporting facilities
- External connectivity for users



"failsafe IP":
- user access
- system admin remote access
- DECnet over IP
- interface status "ping tests"

"dual rail":
- SCS (jumbo frames, PE driver compression and checksumming)

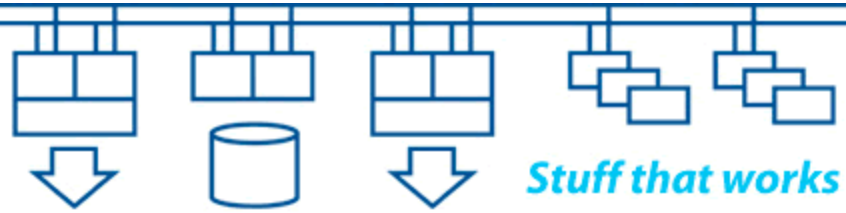
"LAN failover" / "NIC teaming":
- SSSU scripting (TCP/IP)
- DTCS monitoring (AMDS, TCP/IP)
- quorum adjustment (AMDS)
- iLO and switch access (TCP/IP)
- LAT
- DECnet



- Multi-site OpenVMS clusters give us "shared everything" access to data with protection from loss or corruption, even in the event of site failure
- Host-based volume shadowing (HBVS) ensures that data is consistent across all members of the shadow sets.
- The quorum scheme lets Site A continue if Site B fails and protects us from data corruption due to a partitioned cluster
- DTCS monitors the systems and (most important of all) controls the formation of storage shadow sets when the systems boot and when nodes rejoin the cluster







- DTCS is a set of HP and 3rd party products with installation, configuration and support services
- Remote console access, management and console output logging
- Integrated monitoring and quorum adjustment
- Rule based monitoring of individual systems / nodes
- Rule based SNMP polling of equipment
- Rule based TCP/IP "ping reachability" polling
- GUI and e-mail based alerting
- Scripting of failover and recovery actions across all systems / nodes and storage subsystems

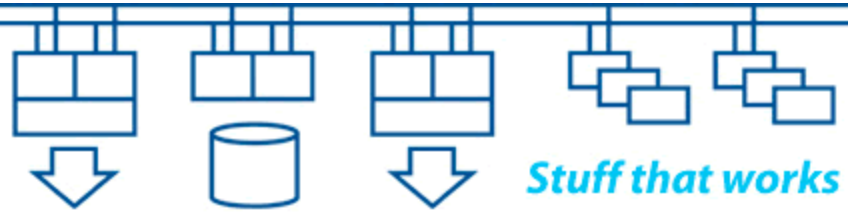
# Part 8 – Conclusion:

- How can we ensure successful delivery ?
  - We can't!
- How can we can maximise our chance of success ?
  - We can!

- You can't buy high-availability "off the shelf"
- Establish the "ideal design" and then adapt it to suit the constraints placed upon you
- Design, plan and test the transition to the new systems
- Design for maintainability and minimal risk of error
- Design for change on-the-fly with no loss of service
- Protect the data - ensure correctness and consistency
- Monitoring, information and automation
- Documentation and configuration control
- Test and check everything regularly
- Continual training and involvement



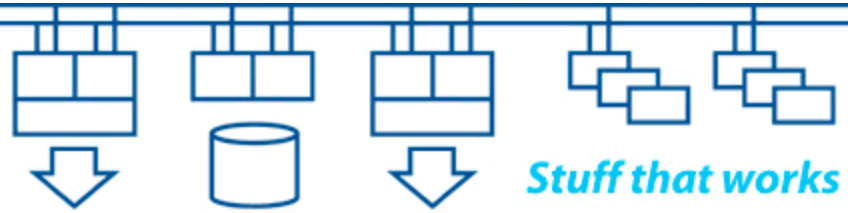




- Procurement – do enough work up front to understand what is technically feasible, what is necessary and to clearly establish the scope
- Have clear objectives and acceptance criteria
- Avoid split responsibility and be absolutely clear where the "duty of care" lies
- Build a "proof of concept" early on and learn from it
- Use small groups of excellent people who work well together and who can communicate effectively
- Attention to detail without losing the big picture
- Good engineering, leadership, project management and collaborative working will help you deliver







# Thank you for your participation

Colin Butcher

www.xdelta.co.uk