BCS Bristol Branch

Putting multi-core systems to use

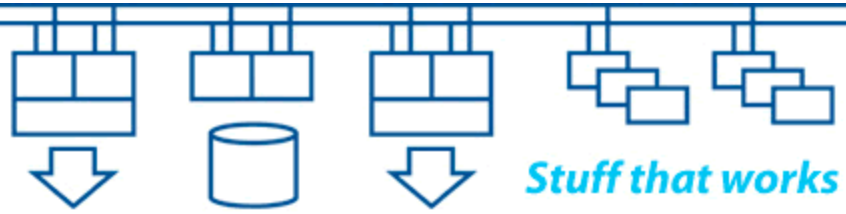
Colin Butcher CEng FBCS CITP

Technical director, XDelta Limited

www.xdelta.co.uk

Agenda

Introduction
"Down the rabbit hole" – from top to bottom
Performance - principles
Hardware platforms
Operating systems
Virtualisation
Software and compilers
Multi-core considerations
Summary



Personal background

- Systems architect specialising in mission critical systems
- Engineering background
- Wide range of experience (satellite flight control, air traffic monitoring, finance data, healthcare, etc.)
- Started XDelta in 1996

XDelta – what we do

- Lead mission-critical systems projects
- Deliver world class services in demanding environments
- Strategic planning, technical leadership and project direction with clarity of vision and an eye for detail
- Systems engineering for availability and performance
- Ensure long term success through skills transfer

What are multi-core systems ?

Multi-core systems are machines built around microprocessors that contain multiple CPU elements and associated elements (eg: caches) on a single chip. A single processor socket contains multiple CPUs. We can build much higher CPU count machines within a smaller physical footprint.

It's all about exploiting the inherent parallelism in the hardware and understanding where the performance gains and limits are.

Down the rabbit-hole …

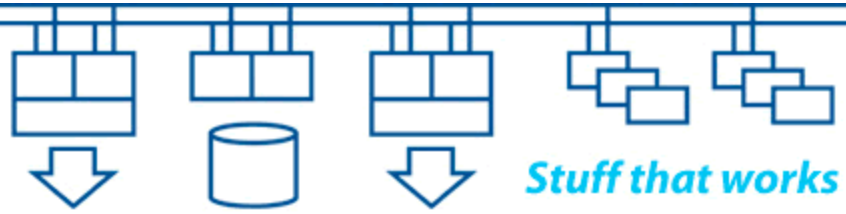
A guide to how a whole system functions, from application software through all the layers down to the underlying hardware.

Let’s start at the top …

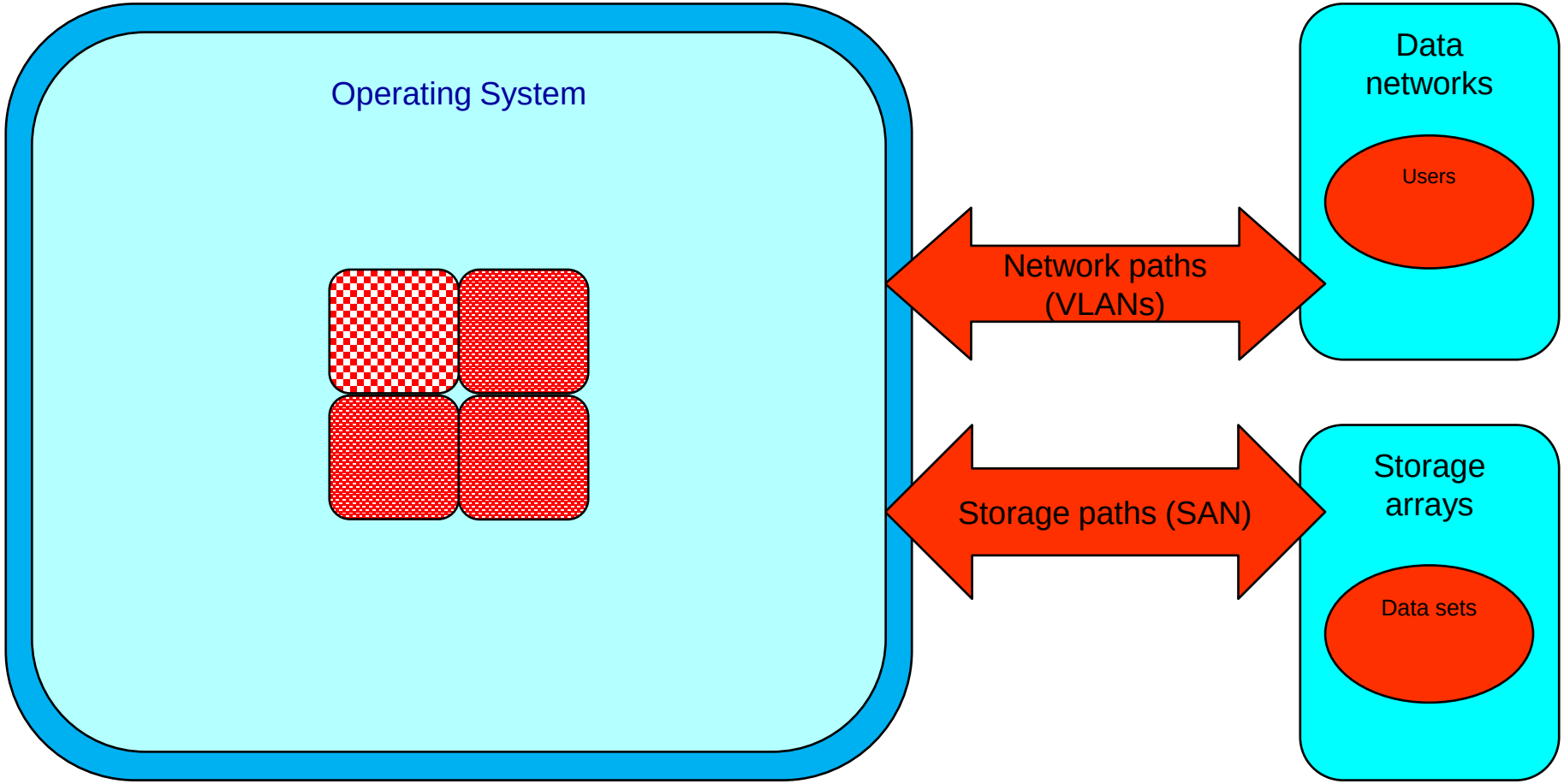
Down the rabbit-hole

Conventional server computing:

- A small machine, no virtualisation.
- Let’s look at what’s happening inside …



Conventional server computing



Conventional server computing

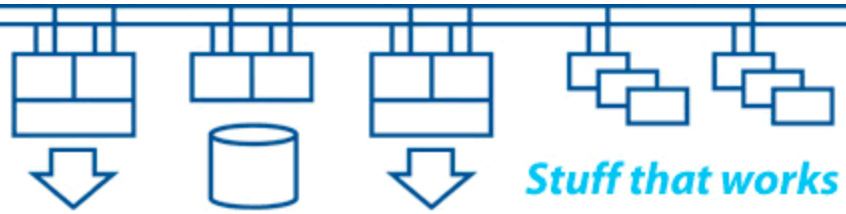
- Several applications each with single stream of execution spread across available CPUs
- Primary CPU runs core operating system functions
- NUMA and “locality” do not apply
- One operating system inside a physical machine
- Interconnects from the machine to the data network switches
- Alternate paths for network data flows (NIC teaming)
- Interconnects from the machine to the SAN fabric
- Interconnects from the SAN fabric to the storage arrays
- Alternate paths for storage data flows (SAN dependent)

Down a different rabbit-hole

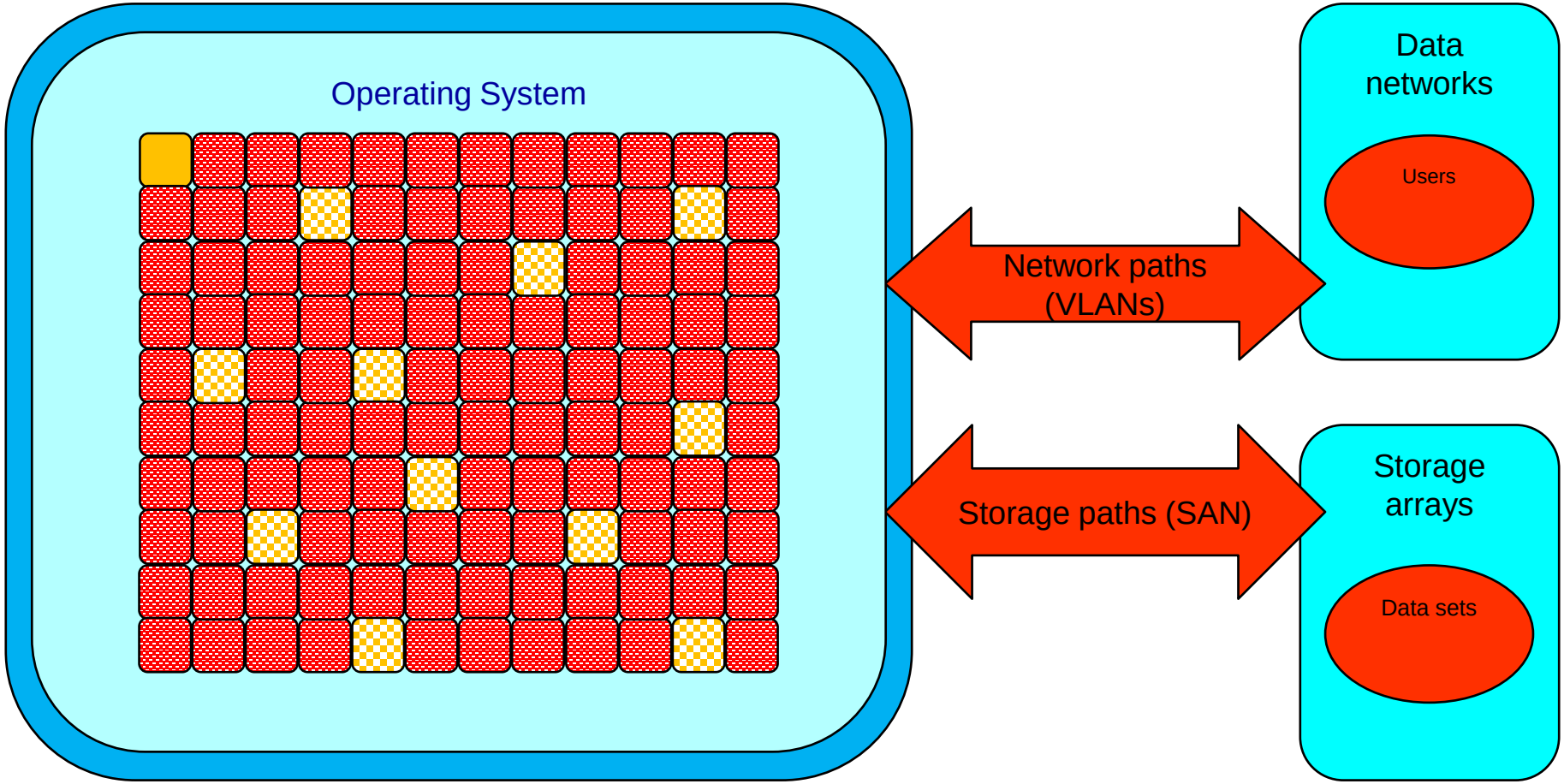
High-performance computing:

- A high CPU count machine with a large amount of shared memory, running as a single system image
- Algorithm design is extremely important to maximise performance by exploiting parallelism
- Some CPUs dedicated to operating system and IO activities
- NUMA (non-uniform memory access) and “locality” matter

- Let’s look at what’s happening inside …

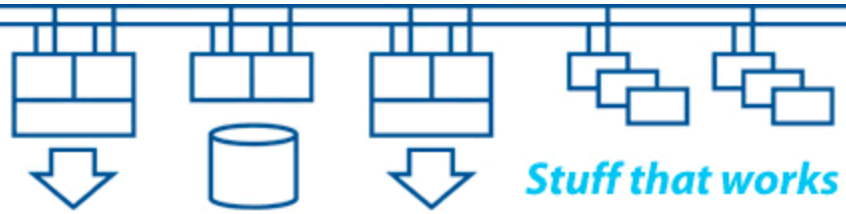


High performance computing



High performance computing

- Single application with multiple streams of execution spread across many CPUs using shared memory
- One operating system inside a physical machine
- NUMA and “locality” effects need to be catered for
- Interconnects from the chassis to the data network switches
- Many parallel paths for network data flows
- Interconnects from the chassis to the SAN fabric
- Interconnects from the SAN fabric to the storage arrays
- Many parallel paths for storage data flows (SAN dependent)



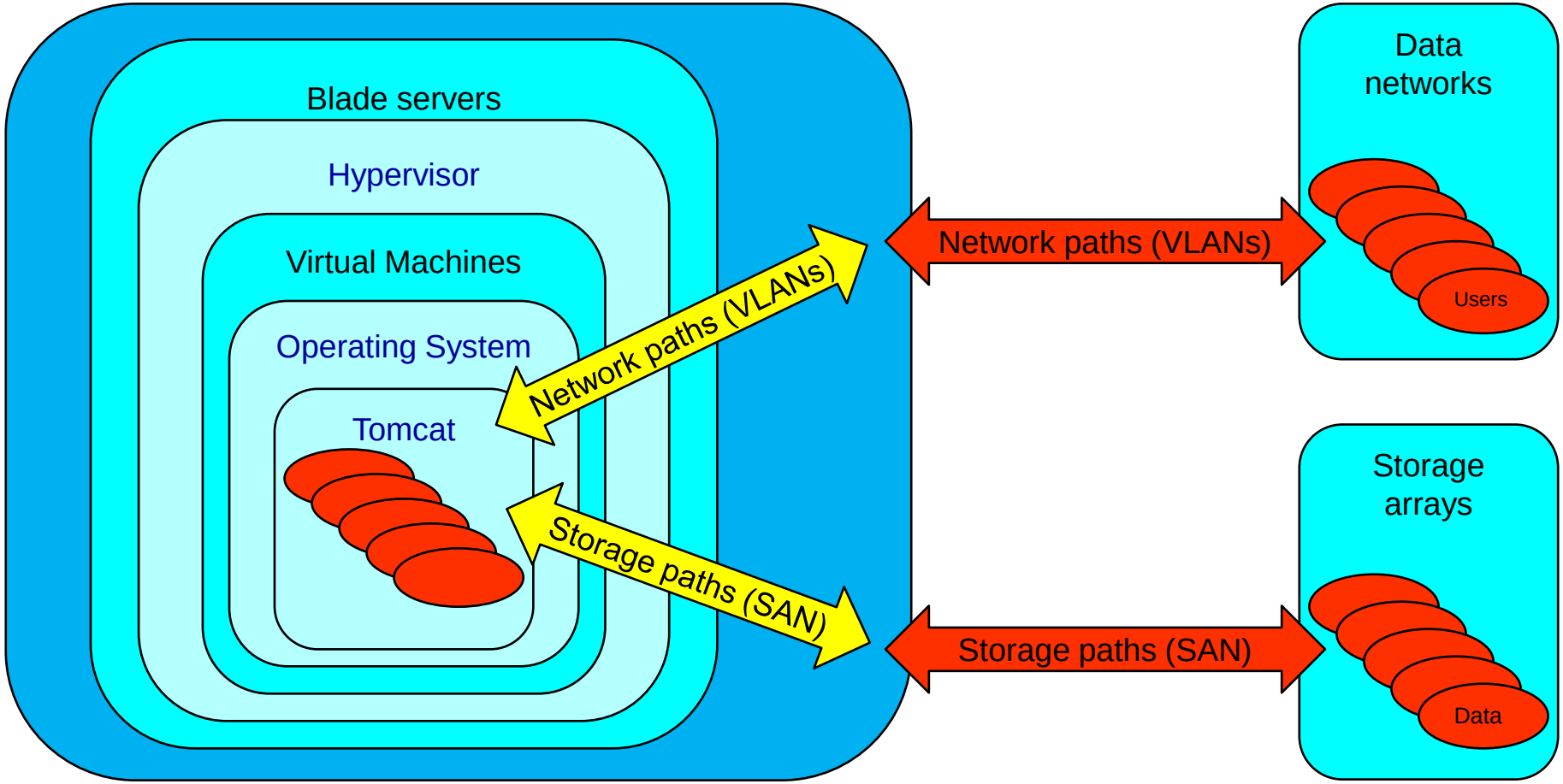
Down a complex rabbit-hole

Here's a typical example:

- An application written in Java, running under Tomcat, running on an operating system, in a virtual machine, under a hypervisor, on a multi-core blade server, in a chassis, connected to remote users over the network, using lots of data held on a fibrechannel storage array, all data replicated to another site, with rapid failover in the event of failure.

- Let's look at what's happening inside …

Virtual server computing



Virtual server computing

- Multiple Java programs under Tomcat
- One instance of Tomcat under an operating system
- One operating system inside a virtual machine
- Multiple virtual machines under a hypervisor
- One hypervisor on the hardware platform (blade server)
- Multiple blade servers in a chassis
- Interconnects from the chassis to the data network switches
- Separation of network data flows (VLANs)
- Interconnects from the chassis to the SAN fabric
- Interconnects from the SAN fabric to the storage arrays
- Separation of storage data flows (SAN dependent)

Performance and availability

A system that doesn't meet its performance requirements is a system that's not working properly, so it becomes unavailable.

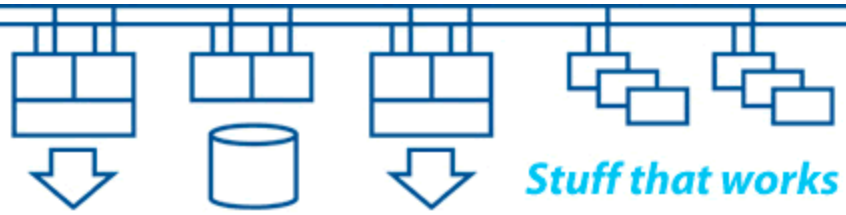
Performance related failures are often transient and exceedingly difficult to fully understand and resolve. The systems have to have sufficient capacity and performance to deal with the workload in an acceptable period of time under normal, failure and recovery conditions.

Basic limits to performance

- Speed of light
- Feature size in “chip”
- Signal degradation (frequency, distance)
- Power consumption and heat dissipation
- Manufacturing processes: impurities, poor contacts, etc.
- Hardware platform and infrastructure design
- Operating system design and configuration
- Compilers and optimisation techniques
- Application and algorithm design

What is “fast” and “slow” ?

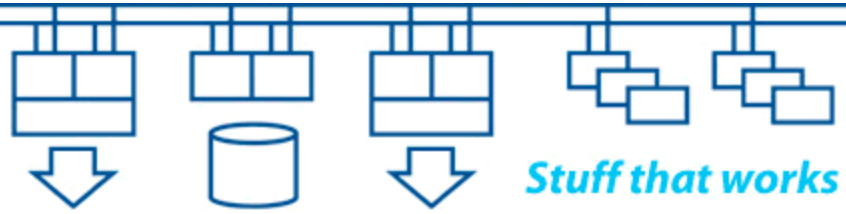
- Bandwidth – determines throughput
- Latency – determines response time
- Jitter – “diff latency” (variation of latency with respect to time) – determines predictability of response



Bandwidth

Bandwidth – determines throughput

- Worst case is the maximum capacity of the smallest path in the system (the “bottleneck”)
- It’s not just “speed” (flow rate), it’s throughput in terms of “units of stuff per second”



Latency

Latency – determines response time

- Worst case is the sum of all the delays in the system
- It's not just "speed" (responsiveness), it's the time taken to interact with the system
- Latency determines how much data is in transit through the system at any given instant in time - "data in transit" is at risk if there is a failure

Jitter (“diff latency”)

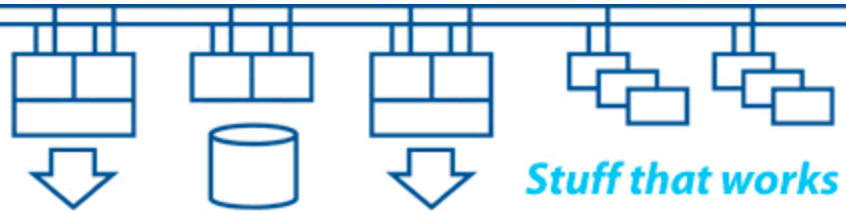
Jitter (“diff latency” - variation of latency with respect to time)

- Worst case latency is used to set timeout values etc.
- Ideally zero, so completely consistent responsiveness
- Latency fluctuations will cause transient system failures under peak load, especially spikes in workload

Capacity and scalability

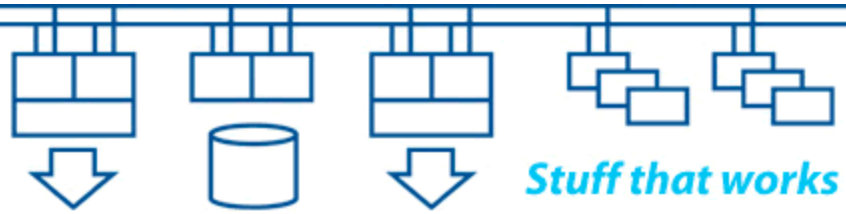
- Contention and saturation – running out of capacity:
  - Queuing theory
  - What else are we sharing our capacity with ?

- 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)



Parallelism and workloads

- 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

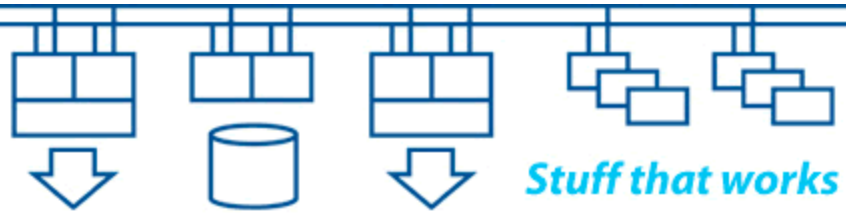


Designing for performance

- Size systems to cope with peaks in workload

- Minimise “wait states” (caches, parallelism, contention)
  - Don’t make it go faster, stop it going slower
- Maximise “user mode”, minimise the other modes:
  - The fastest code is the code you don’t execute
- Maximise “IO throughput”:
  - The fastest IO is the IO you don’t do

Designing and writing very good code requires very good programmers.



Hardware platforms

A hardware platform (machine, computer, server, etc.) comprises a box with stable power, cooling, monitoring, console interface, CPUs, memory, data paths and IO devices. It may also include local storage devices and graphics capability. The main component is the processor, whose architecture is defined by the instruction set.

CPUs (1)

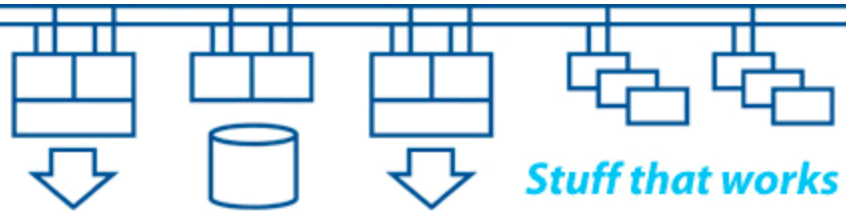
- Processes instructions (I stream)
- Manipulates data (D stream)
- Instruction set (processor architecture)
  - Integer arithmetic
  - Floating point arithmetic
  - Character string manipulation
- Microcode (processor implementation)
- Clocking
- “bit width” (16bit, 32bit, 64bit …) and alignment boundaries
- Protection mechanisms (processor modes)

CPUs (2)

- Performance:
  - Registers
  - Caches
  - Pipelines
  - Predictive branch
  - Instruction re-ordering
  - Parallel execution
  - Multiple cores
  - Hyperthreading
- CISC / RISC / EPIC
- Compilers and optimisation are key to performance

Memory subsystem

- Used to be small, slow and expensive, so tried hard to minimise memory usage
- Now plentiful, fast and relatively cheap
- Error detection and correction
- Can use memory to gain performance
- Caches and synchronisation across multiple processors
- Memory management: page tables and translation buffer
- Memory interconnects to CPUs and “locality” to IO devices
- NUMA (non-uniform memory access)



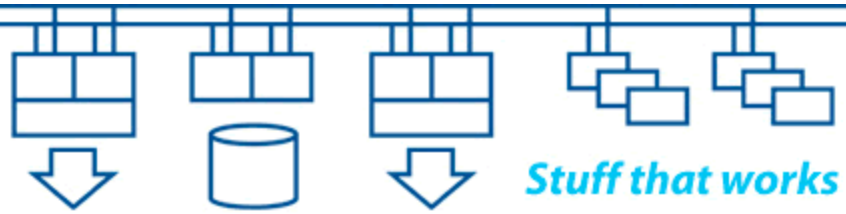
IO (Input / Output) subsystem

- Provides interface with outside world
- Connects to network devices, storage subsystems, etc.
- Interface bandwidth and latency (memory, cache etc.)
- IO controllers and “locality” to CPUs / memory
- IO device operation (interrupts, registers, DMA transfers)
- IO interconnects to CPU and memory subsystems
- Device detection (ACPI)

- Typical storage latency for FC SAN = 0.5 msec (very good)
- Typical storage latency for 5k4 laptop drive = 13+ msec
- Compare with CPU cycle time and memory latency!

Multiprocessing

- Asymmetric Multiprocessing (attached processor, eg: GPU)
- Symmetric Multiprocessing (SMP)
- Interconnections (switch, mesh, toroid etc.)
- Latency and bandwidth to memory and IO devices (NUMA, “locality”)
- Caches and cache coherency
- Synchronisation and serialisation
- IO processing (interrupt handling)
- Multi-core
- Hyper-threading
- Partitioning



Multiprocessing - trends over time

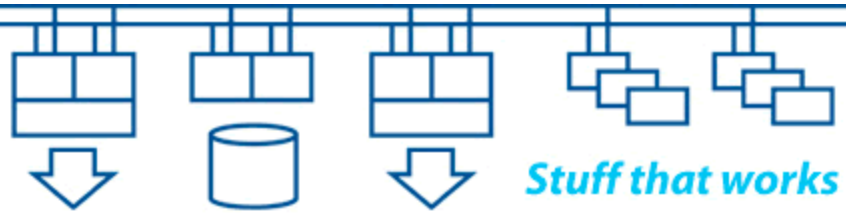
- Increasing use of parallelism and caches over time
- Increasing clock rates affect signal distances
- Increasing awareness of NUMA

- Uniprocessor where CPU takes several boards
- Uniprocessor where CPU is a dedicated microprocessor
- Attached processors (FPUs, co-processors, GPUs)
- Multi-processor with common memory and IO bus
- Multi-processor with multiple buses
- Multi-core processors (2,4,8, …)

Blade servers and chassis

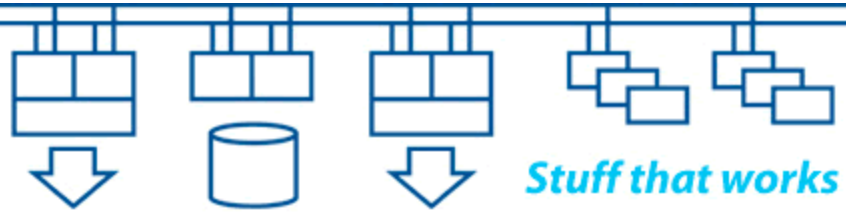
Blade technology brings virtualisation of the system infrastructure (chassis components):

- Virtual connections from processing components over backplane channels – think about how WWIDs and MAC addresses are presented
- Modular systems provide great flexibility of configuration and interchangeability of components



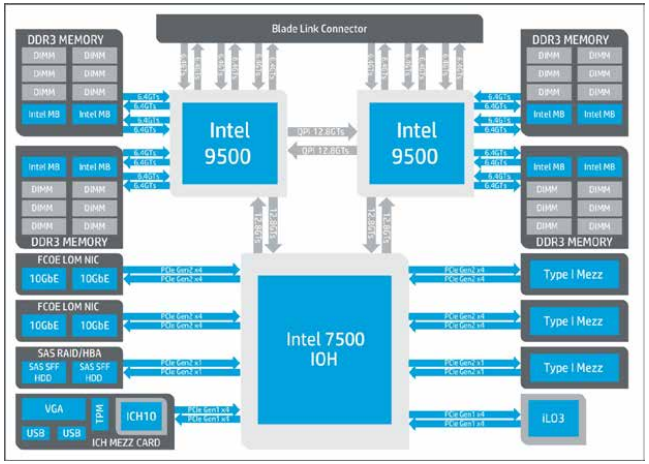
Stand-alone systems

- Big high-end multiprocessor systems are needed for certain workloads
- Stand-alone systems are still needed, eg: highly secure or mission-critical / safety-critical environments (less complex hardware infrastructure = reduced probability of errors)

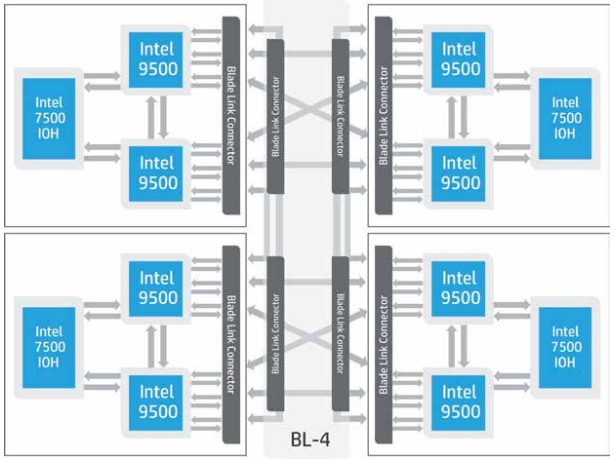


Example: HP BL890c-i4 architecture

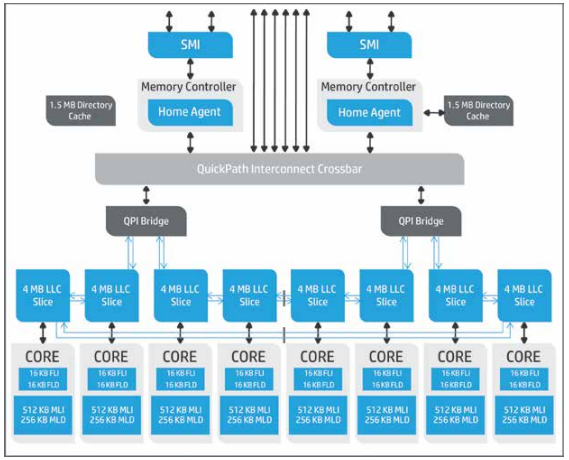
Blade



Links



CPU



Quad-width blade
64 cores, 1.5TB
16x 10GigE/FCoE
+ 12x IO modules

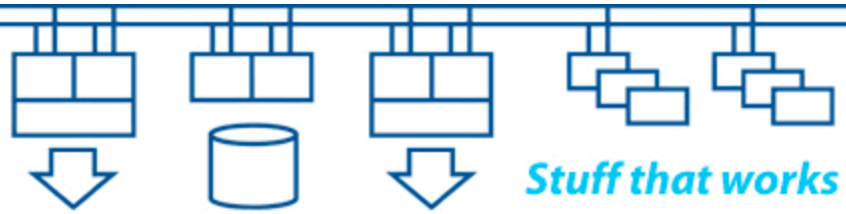
Operating systems

Operating systems and software are the things we interact with and which turn a pile of hardware into something we can use

Operating systems have to be well structured and designed for performance and availability on a wide range of hardware.

Memory management and virtual memory

- Each process can use the full address space capability of the machine as permitted by the operating system
- Process address space is ‘mapped’ into physical memory – translation buffer hardware and page table entries are where the microprocessor and operating system meet
- Physical memory usage per process is controlled to maintain overall performance and prevent interactions
- Per process memory that exceeds the permitted physical memory quotas is ‘parked’ in the page file ready for use
- Cache synchronisation (eg: cache invalidation)



Scheduling – processes and “threads”

- The scheduler allocates computable streams of execution to CPUs for a period of time, or until a blocking event occurs (eg: IO issue, synchronisation, etc.)
- Process creation / deletion is expensive
- Context switching is expensive
- User-mode threads are a lighter-weight schedulable entity than processes and exist within the context of a process
- Cache synchronisation and kernel data structure access events across multiple CPUs may need to “stall” the entire machine
- Awareness of NUMA and “locality”

IO handling

- The IO subsystem presents a consistent interface to the application software
- A “computable entity” (process, thread, etc.) cannot be computable while waiting for IO completion before it can continue
- Interrupt handling in multiprocessor systems – best done on CPU nearest the IO controller (“locality”)
- DMA (direct memory access) data transfer – best done to memory nearest the IO controller (NUMA)
- Synchronisation and serialisation of IO operations

The file system and storage devices

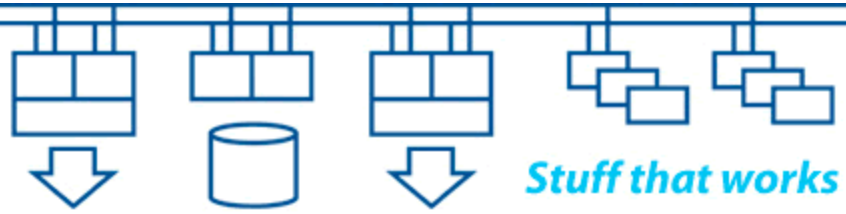
- Devices are “block structured”
- Volumes are “file structured”
- Data are “record structured”
- Arbitrate access to control data structures (file system) and the file content (record level locking)

Virtualisation

The hypervisor is a minimalist operating system sharing out the physical resources amongst the virtual machine guests.

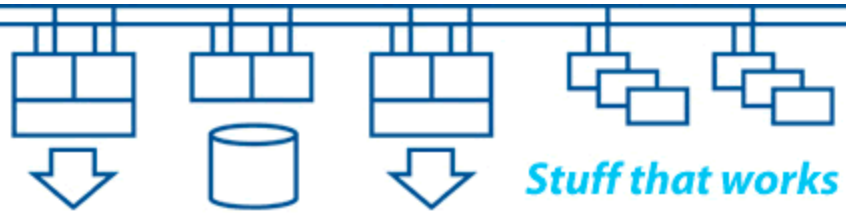
It hides the underlying hardware so that we can use it without having to think about it.

It's a way of utilising parallelism in the hardware without having to re-design applications.



What can CPU virtualisation do for us?

- Improved utilisation of hardware resources
- Improved high availability and disaster tolerance
- Multiple run-time environments
- Put all our eggs in one basket!

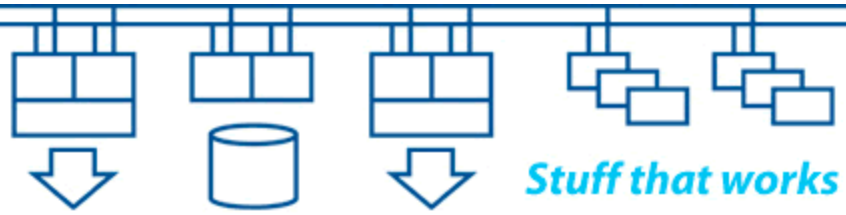


Virtual machines

A CPU cannot exist in isolation. We need:

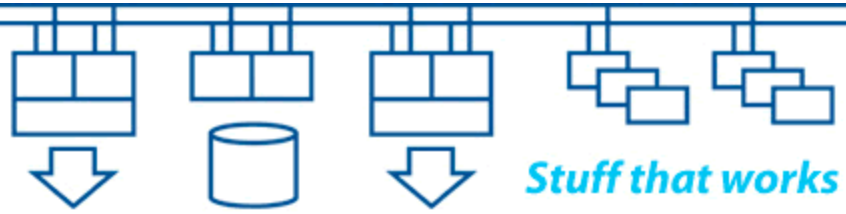
- CPU – processing capability
- Console interface
- Memory – stores data and instructions ready for use
- IO devices – move data into / out of memory

- IO from within a virtual machine has much greater latency than direct physical IO, hence techniques such as paravirtualisation with dedicated IO devices



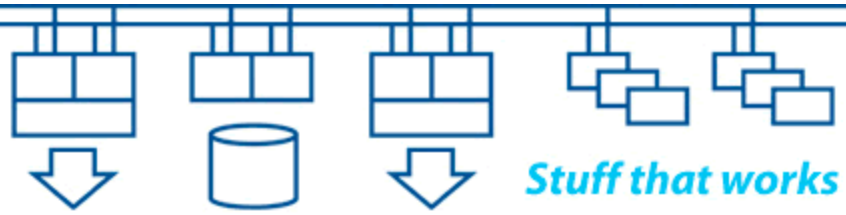
Creating a Virtual Machine (1)

- The VM is a multi-threaded application running under the control of the hypervisor
- CPU scheduling in the guest OS maps to threads scheduled by the hypervisor
- Physical memory in the guest OS maps to virtual memory in the hypervisor – lots of physical memory in the hardware platform is a good thing



Creating a Virtual Machine (2)

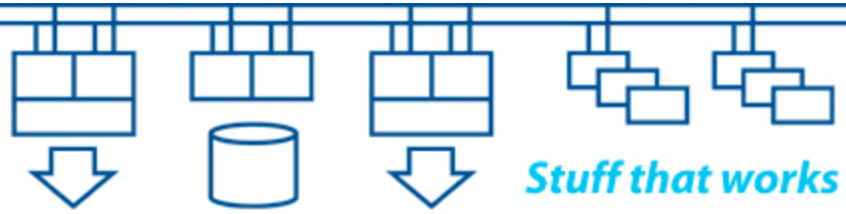
- Devices presented to the operating system running inside the VM have to ‘map through’ to devices managed by the hypervisor:
- Storage devices typically map to container files, not physical hardware
- Network devices typically map to 802.1Q tagged VLANs, not physical hardware



Parallelism through virtualisation

- It's another way to achieve parallelisation of processing without having to invest time in writing code that works well on a multi-processor system.

- However, we still need to think about concurrent access to data from multiple "virtual machine instances" and providing access to the multiple "virtual machine instances".



VLANs – virtualisation of data networks

VLANs are used to segment a data network:

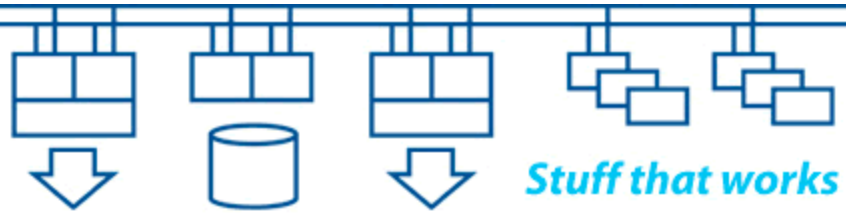
- Implemented within core switches
- Connectivity between VLANs
- VLAN tagging of packets (802.1Q)
- VLAN tagging of packets out of NICs
- QoS (Quality of Service) and bandwidth reservation

Storage arrays – virtualisation of discs

Storage array controllers:

- The array “hides” the behaviour of the discs from you
- The array “levels” the storage to provide best throughput for the access pattern to the entire array (or disk group)
- The array controller caches much of the data

- The only real performance issue is bandwidth to and from the array controller pair and what else is contending for that bandwidth



Workloads in a virtual world

- To the hypervisor, each and every virtual machine is a workload needing physical hardware resources
- Within a virtual machine, each application (and the operating system overhead) is a workload

- What level of interaction is there between the virtual machines that run your applications ?
- What happens when your host hardware runs out of resources or when virtual machines are moving to another hardware host ?

Software and compilers

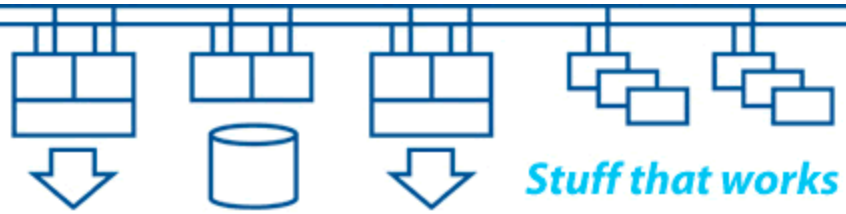
Pick an appropriate language for the task at hand.

Limits to software performance

- Choice of languages
- Ability to make use of parallelism
- Granularity of data structures
- Synchronisation techniques
- Serialisation techniques
- Scalability techniques
- Compilers
- Application and algorithm design

Compilers

- Different types of instructions and data are split out into separate sections (shared data, read-only data, local read-write data etc.) for use by the linker
- Generate code for a specific machine architecture
- Optimisation re-orders the code to take advantage of hardware parallelism and processing efficiencies
- Generate debug information
- Linker lays out the image address space and provides hooks for the image activator
- Warnings are the compiler's way of telling you that it's guessing! Don't ignore them – fix your code.

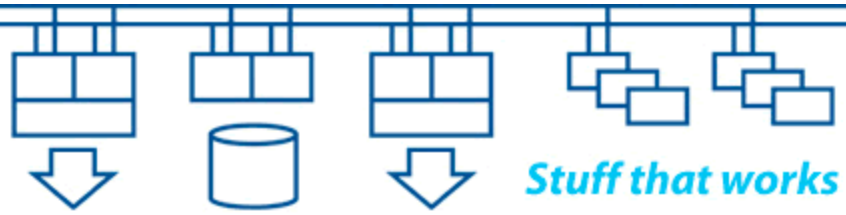


Multi-core considerations

At long last ...

Current technology trends - parallelism

- Move from low core count, high clock rate processors to high core count, low clock rate processors
- Implicit assumption is that parallelism can be achieved
- How does our workload break down into parallel streams of execution ?
- Can compilers, operating systems and microprocessors find inherent parallelism and exploit it for us ?



Where does multi-core fit in all this?

- High core counts and shared on-chip caches boost performance:
  - If all cores are working on the same data
  - If not, increases overhead with cache synchronisation
- More CPUs in the same physical footprint
- Reduced signal distances improve memory and IO latency Reduced power consumption
- Hardware assist for CPU virtualisation
- Graphics processors (GPUs) are a good example of high core count processors for specific tasks

Things to consider

- Maximum achievable throughput (bandwidth)
- Effects of latency and jitter
- Monitoring and management techniques
- Scheduling techniques
- Availability, reliability and complexity
- Planning downtime for maintenance on shared-use hardware with many “virtual machine instances”
- Realistic testing for scale and performance
- Licensing policies and “pay per use” costs

Discussion

BCS Bristol Branch

Thank you for your participation

Colin Butcher CEng FBCS CITP

Technical director, XDelta Limited

www.xdelta.co.uk