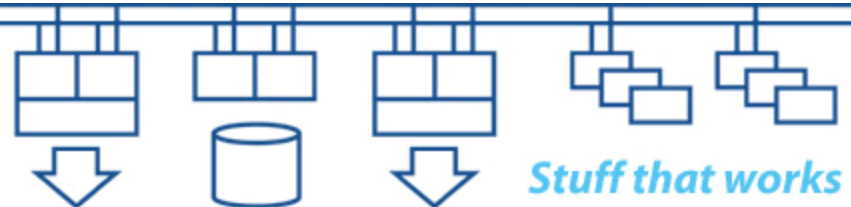


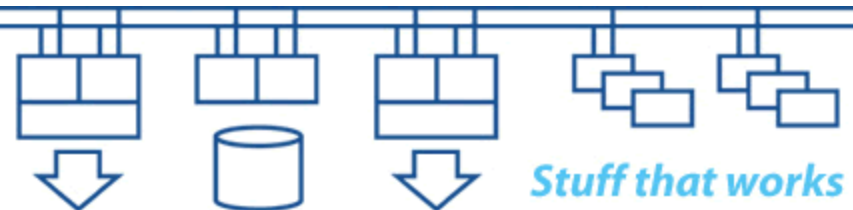
HPUG - OpenVMS day – March 2009

# Performance - getting the best from your new platform

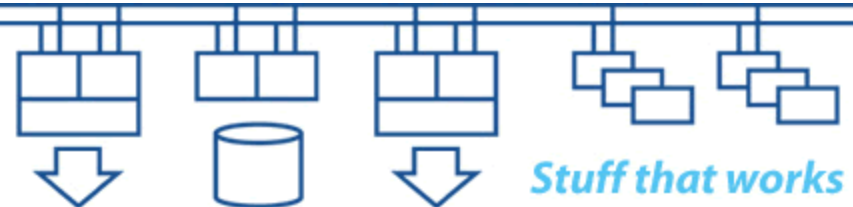
Colin Butcher



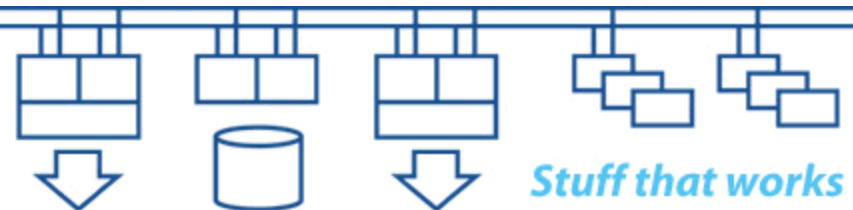
- Integrity is different from Alpha and VAX, so we should make appropriate changes for performance
- Setting the new systems up for good performance is one of the hidden aspects of porting applications
- We need systems to easily handle the normal workload
- We need systems to be capable of absorbing unexpected spikes in workload without problems
- We don't want to spend our time managing performance and doing tuning exercises



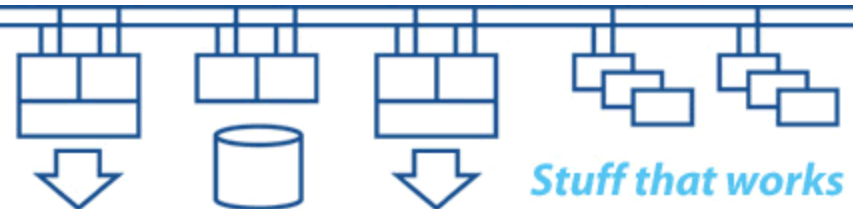
- Principles of performance
- Designing and implementing scalable software
- Integrity Server specific topics
- Making use of multiple CPUs (cores and hyperthreads)
- Making use of memory
- Making use of the storage IO subsystem
- Making use of the network IO subsystem
- Hardware platform – power-up / restart / dump
- Q & A



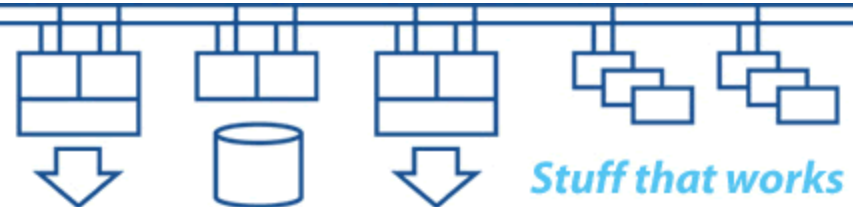
- **Bandwidth – determines throughput**
  - It's not just “speed”, it's throughput in terms of “units of stuff per second”
- **Latency – determines response time**
  - Determines how much “stuff” is in transit through the system at any given instant
  - “Stuff in transit” is the data at risk if there 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



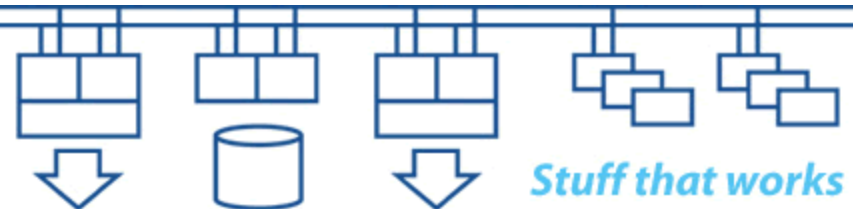
- Size systems to cope with peaks in workload
- Minimise “wait states” (caches, parallelism)
- Minimise contention for resources and data structures
- Understand the need for synchronisation and serialisation of access to data structures
- Maximise “User Mode”, minimise the other modes:
  - The fastest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



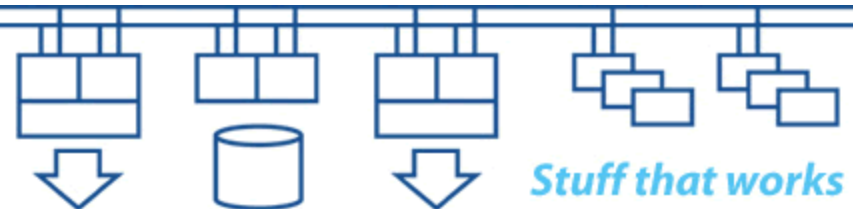
- Understand how the applications could break down into parallel streams of execution:
  - Some will be capable of being split into many small elements with little interaction between the parallel streams of execution
  - Some will require very high interconnectivity between the parallel streams of execution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



- Check for code making assumptions that the system is a uniprocessor machine:
  - Flags controlling access to an entire global section
  - Loops polling for flag status changes (spinlocks)
  - Data structures not protected from operations that may happen in parallel instead of sequentially
- Use the lock manager to serialise and synchronise access to data structures
- Minimise wait states by having appropriate granularity of access to data structures
- Take null locks out, then simply convert them as needed

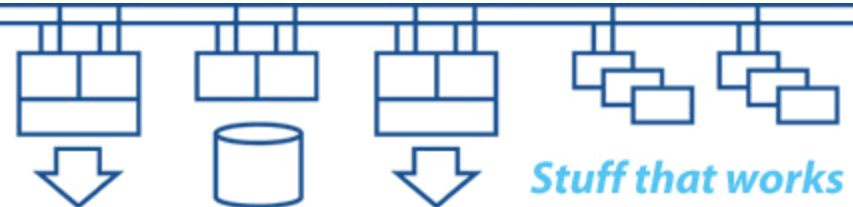


- Generate the Instruction and Data streams for processing by the system
- Different types of instructions and data are split out into separate sections (shared data, read-only data, local read-write data etc.) for use by the linker
- Generate code for a specific machine architecture
- Optimisation re-orders 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

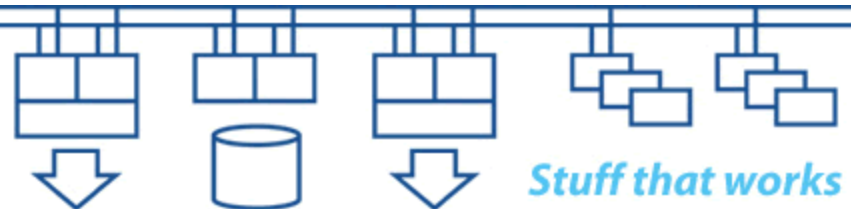




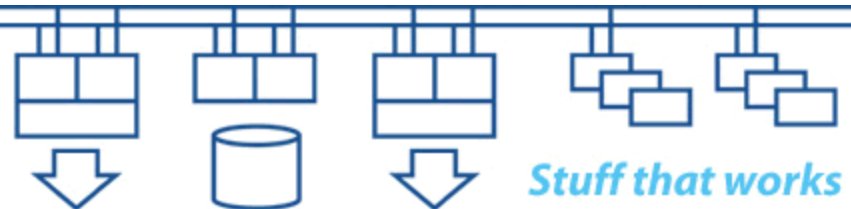
- Ability to make use of hardware parallelism
- Granularity of data structures
- Synchronisation techniques
- Serialisation techniques
- Scalability techniques
- Compilers
- Application design
- Designing and writing very good code requires very good programmers



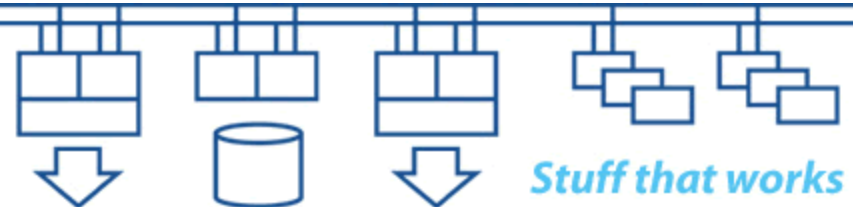
- Compilers are the key to performance
- Use the documented mechanisms provided by the operating system – read the release notes and new features, then use the current mechanisms
- Exception handling (`LIB$SIGNAL` etc.)
- Data alignment (unexpected alignment faults)
- Floating Point format (IEEE by default)
- Implicit assumptions (not Alpha, not VAX)
- Debugging - ELF & DWARF formats
- Integrity calling standard and register usage



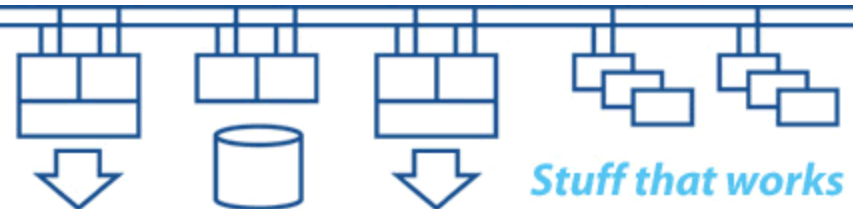
- Currently a mixture of Intel standard compilers and GEM back-end compilers, depending on the language
- Defaults are likely to be different on Integrity, so explicitly set the command line switches to get the behaviour you need
- Check the SPDs for the compilers you need
- Test compile some of your code early on – get to the latest compiler versions first if you can before you change platform
- DEC ADA'83 not available – use Gnat ADA Pro
- Standard C is Standard C
- Fortran DEC extensions are (mostly) supported



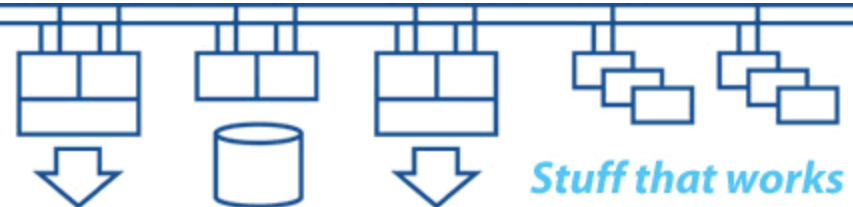
- Unexpected alignment faults on Integrity are expensive and affect the entire system performance, so eliminate them if you can. Use the appropriate compiler switches.
- MONITOR ALIGN will show if you have issues
- Home in on the problem areas using the SDA extension:
- Alignment fault trace (with SDA)...  
SDA> FLT START TRACE  
SDA> FLT SHOW TRACE /SUMMARY  
flt\_summary.txt
- Alignment fault trace (with SDA)...  
SDA> FLT START TRACE [/CALL]  
SDA> FLT SHOW TRACE  
flt\_trace.txt



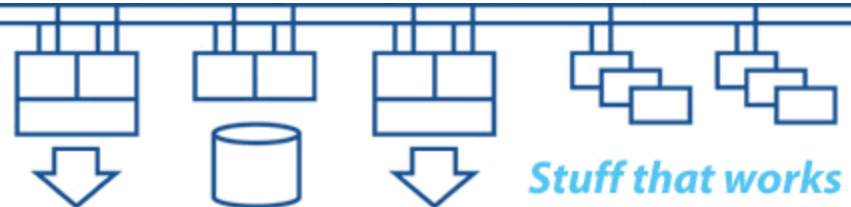
- Exception handling (LIB\$SIGNAL etc.) is a more expensive mechanism than it was on Alpha
- Consider alternatives if your code makes extensive use of exception handling as part of it's normal flow of control
- Increase stack space when using threads



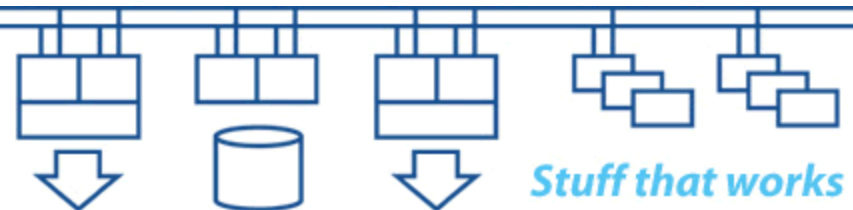
- Integrity default floating point format is IEEE
- VAX floating point is handled by converting to IEEE format, doing the processing, then converting the result back from IEEE format to VAX format, hence a small performance penalty
- Alphas do both VAX and IEEE, but the default is VAX
- Results with IEEE are 'slightly different', not by much
- Consider testing for a small difference, not equal
- Beware LIB\$WAIT(<real>)!
- Beware reading in data files / writing out data files and moving them between different systems
- What standard do you use for moving data?



- **Check for code making assumptions about hardware:**
  - “if not Alpha” is definitely a problem!
  - “if not VAX” should be less of a problem
  - “if VAX” / “if Alpha” / “if IA64” is safe
- **Check for code making assumptions about page sizes:**
  - Memory page sizes are variable on Integrity (and Alpha)
  - VAX is 512 bytes (same as disc block size)
  - Alpha is 8192 bytes
  - Integrity is currently 8192 bytes

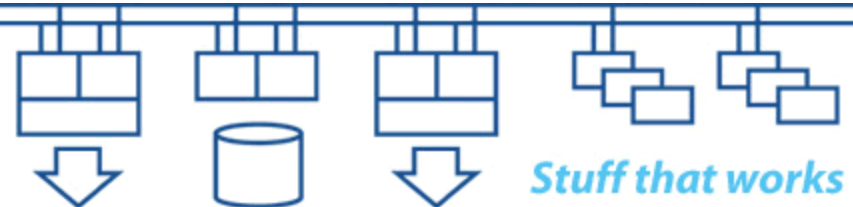


- **Check for code making assumptions that the system is a uniprocessor machine:**
  - Flag bits controlling access to an entire global section
  - Loops polling for flag bit status changes (spinlocks)
  - Data structures not protected from operations that may happen in parallel instead of sequentially
- **Use the lock manager to serialise and synchronise access to data structures**
- **Minimise wait states by having appropriate granularity of access to data structures**
- **Take null locks out, then simply convert them as needed**

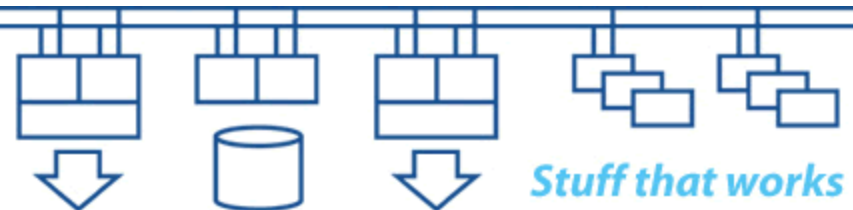




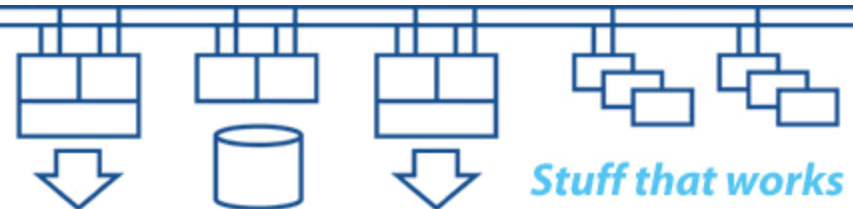
- Java is not a compiled language running native instructions – it uses an intermediate form known as “byte codes” which are processed by the Java run-time environment (Java Virtual Machine)
- The Java run-time environment uses significant amounts of memory and performs “garbage collection” intermittently to remove objects no longer in use
- Tuning a system to run Java well can be “interesting”
- Hotspot VM and JRAT on Integrity only
- Netbeans profiler



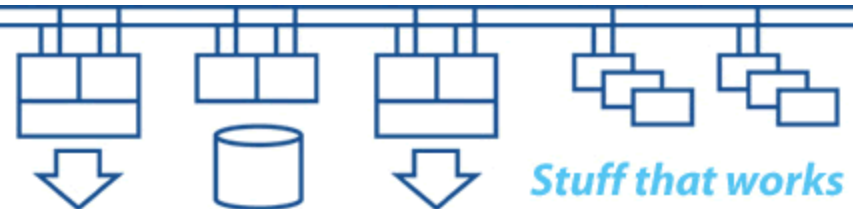
- Make a thorough audit and analysis of what you're doing at the moment
- Consider taking advantage of new features and new ways to do things
- There may be no direct equivalents
- Direct 'bug for bug' port or re-implement application?
- Are you going to have to continue to support non-migrated systems as well?
- **System infrastructure implications:**
  - storage subsystems (eg: SANs)
  - network connectivity (eg: multiple NICs)



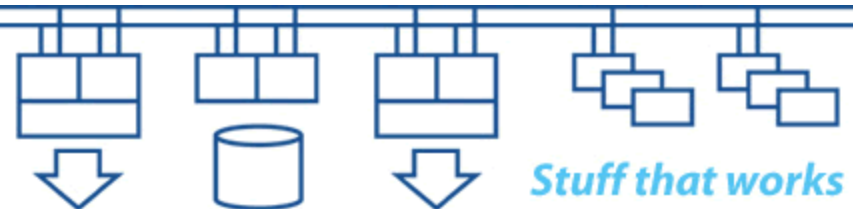
- Emulators can let you run existing binary code on an emulated hardware platform hosted on a replacement system platform (typically a Windows PC)
- Emulators are entirely dependent on the underlying system platform for their overall behaviour
- Emulators require IO support to be created specifically
- Emulators can give you long-term complications – carefully consider the long-term cost of ownership and the implications for supporting critical applications over time
  
- Binary translators will convert executable images “on the fly” into an executable image for the target processor, even VAX to Alpha to Integrity



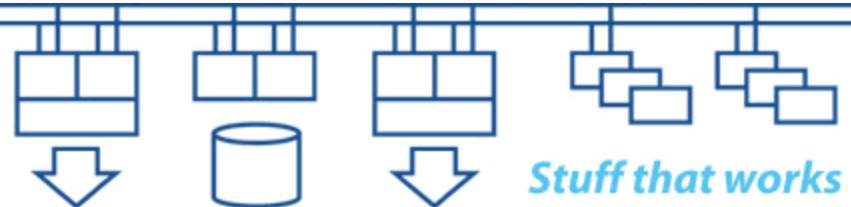
- Hyperthreading – affects entire box, very workload dependent
- Fastpath IO devices and distributed interrupt handling
- Introduce parallelism into your code and batch jobs where possible
- Dedicated CPU for lock manager (local locking)
- Compression and encryption, eg: SCS compression, BACKUP compression
- QUANTUM, workload dependent – many SYSGEN parameter defaults changed in V8.2
- Power management



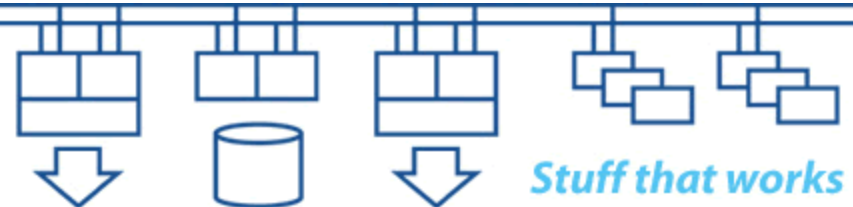
- Use memory (XFC, resident images, DECram etc.)
- Revisit working set sizes - WSMAX and process quotas (WSDEF / WSQUO / WSEXT / PGFLQUO)
- Use RMS global buffers
- Revisit RMS system defaults
- XFC - beware “double caching” and introducing performance penalties
- GH regions – map lots of memory with a small number of page table entries
- INSTALL /RESIDENT and GH region size
- 64bit P2 space and memory reservations
- DECram and HBVS to disc



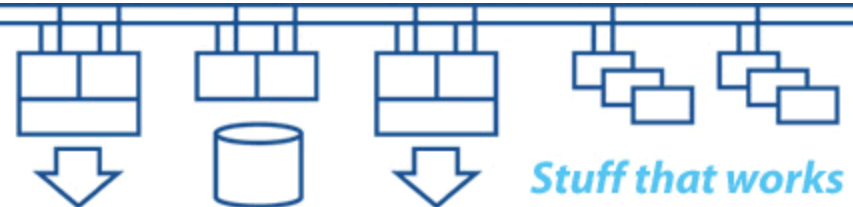
- FC bandwidth is important – what else are you sharing your storage bandwidth with? Why?
- EVA cache size and OpenVMS volume characteristics (cluster factor, extend quantity, dynamic volume expansion etc.)
- HBVS – only shadow what you really need to
- HBVS – many shadow sets let you control how rapidly shadow copying proceeds during recovery
- HBVS mini-copy and mini-merge policies
- HBVS block count to match EVA cache size



- Rotational latency no longer matters, nor does balancing the IO load to the spindles
- SAN zoning, preferred paths and inter-site links
- EVA configuration - disc groups, RSS groups, sparing, VRAID 0+1, levelling behaviour
- Virtual tape libraries
- EVA mirrorclones, snapshots etc.
- EVA CA – FC “fast write” behaviour

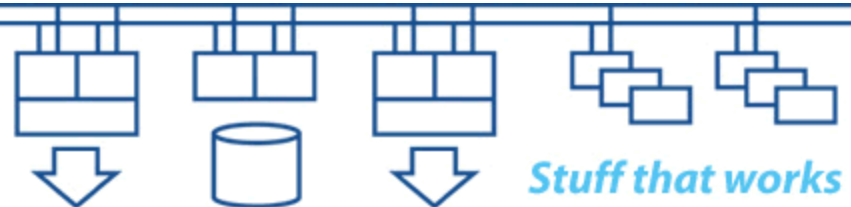


- Split protocols across adapters
- Jumbo frames (`LAN_FLAGS` and `LANCP`)
- SCS window size (`SCACP CALCULATE`)
- TCPIP “failsafe IP” and IP addressing scheme design
- 802.1Q VLAN tagging
- DECnet-Plus single rail with LAN failover v multiple rails with load balancing
- Understand latency effects of long-distance networks - check the real distance!
- Fast LAN transmit timeout (`LAN_FLAGS` bit 12)
- TCPIP - keepalive, delack, MTU discovery
- RMS block sizes for network IO





- Power-up / boot / dump / restart time
- Integrity Server memory tests on power-up
- Local disc and RAID controllers
- Where to put crash dumps? Local or FC?
- Where to put page / swap files? Local or FC?
- Compressed selective dumps
- EFI / MP configuration
- Remote monitoring / management
- Blade chassis and servers



# Thank you for your participation

## Q & A

