VMS Software

What we Learned From the Port

Berlin, April 2024

Camiel Vanderhoeven | Chief Architect & Strategist

Vision

By linking the past to the future, we help OpenVMS users to protect and realize the full value of their application investments.

Mission

We combine leading edge technology and new industry standards with OpenVMS systems to provide our customers and partners with choice and opportunity to profitably prioritize business needs.

Agenda

1	Masquerade
2	Why Probe?
3	“Unified” Extensible Firmware Interface
4	Let’s be different!
5	Developing on a VM

VMS Software

Developing Using VMs

Masquerade

4 modes

- 4 pagetables per process
- Complex code for transitioning between modes

8k pages

- Native pages on x86 are 4k
- VMS pages are 8k
- Memory needed for UEFI may be in adjacent 4k pages

Queue Instructions

Probe Instructions

VMS Software

Why Probe?

Why Probe?

System service calls

- Verify caller has access
- Avoid pagefaults at elevated IPL

Interrupts / exceptions

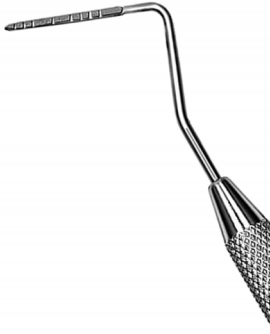
- Avoid pagefaults at elevated IPL
- Invalid stack pointers

Why Probe?

No probe instructions on X86

- Walk the page tables “by hand”
- Validate input parameters
- ~ 900 instructions executed

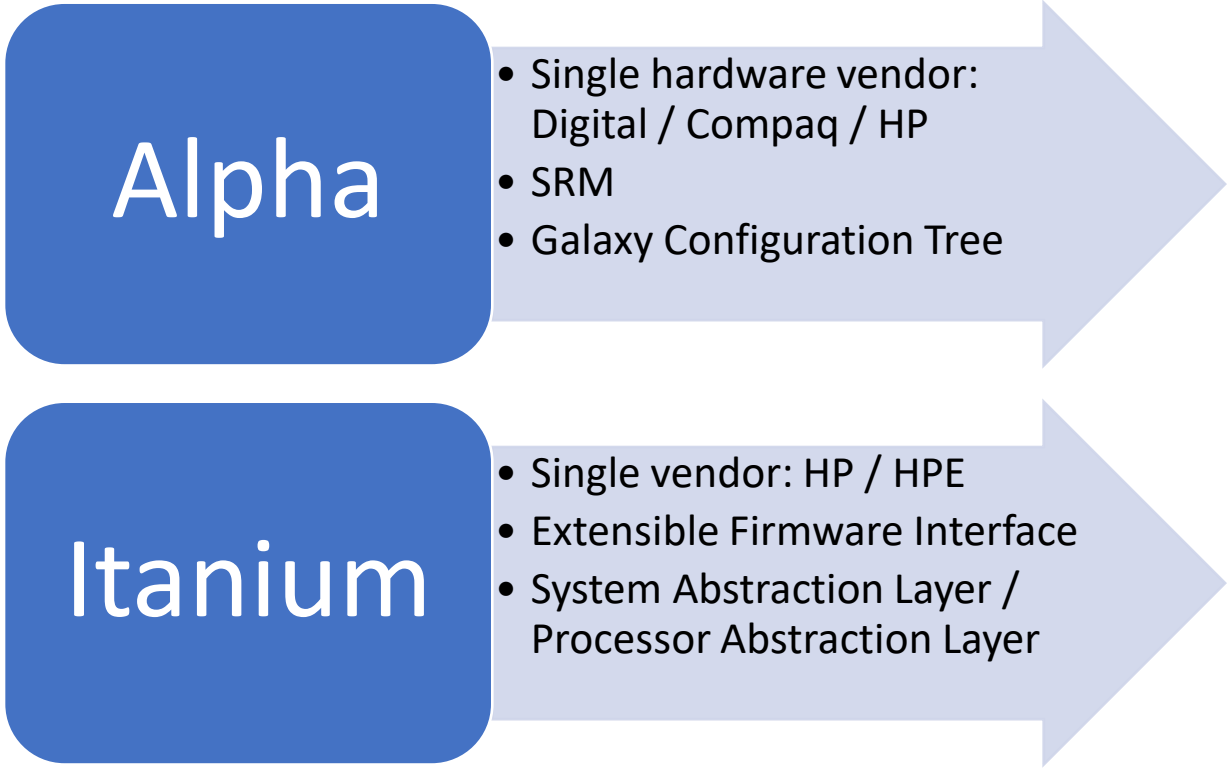
Expensive



VMS Software

“Unified” Extensible Firmware Interface

Where we came from



Where we ended up on X86

- Multiple vendors / hypervisors
- UEFI (without SAL or PAL)
- ACPI (advanced Configuration and Power Interface)
- Every hypervisor is different
  - Sometimes even between versions of the same Hypervisor
- Bare-metal hardware is different again

“Unified” Extensible Firmware Interface

BERT	BOOT	**BGRT**	CPEP	CSRT	DBG2	DBGP
DSDT	DMAR	DPPT	DRTM	ECDT	EINJ	ERST
ETDT	**FACS**	**FADT**	FPDT	GTDT	HEST	HMAT
HPET	IBFT	IORT	IVRS	LPIT	**MADT**	**MCFG**
MCHI	MPST	MSCT	MSDM	NFIT	OEMx	PCCT
PDTT	PMTT	PPTT	PSDT	RASF	**RSDP**	RSDT
SBST	SDEI	SDEV	SLIC	SLIT	SPCR	SPMI
SRAT	**SSDT**	STAO	TCPA	TPM2	UEFI	WAET
WDAT	WDDT	WDRT	WPBT	WSMT	XENV	**XSDT**

VMS Software

Let’s be Different!

VMware, VirtualBox, KVM

Virtual machines pretend to be real hardware

- Provide emulated devices and controllers – SCSI, SATA, Intel NIC, Chipset
- *Optionally* provide higher speed, lower latency virtualized I/O interfaces
- Guest OS can use standard device drivers

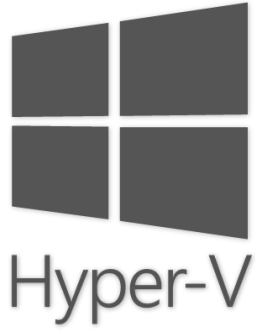
Able to accommodate OS'es that know nothing about Virtual Machines

Microsoft Hyper-V

Virtual machines are unapologetically virtual constructs

- Does not provide emulated devices or controllers
- *Only* provide virtualized I/O interfaces
- Guest required to use Hyper-V specific device drivers

Requires the OS to know how to run on Hyper-V



VMS Software

Developing using VMs

Developing on VM's

Versatility

- Test on the same system you use for other things

Resiliency

- Snapshots
- Pre-built appliances
- Crash? Easy roll-back

Flexibility

- More memory?
- More CPU cores?
- Different NIC?

Debugging

- Post-mortem

VMS Software

Thank you