

# Virtual Machines

Gernot Heiser

COMP9242 2005/S2 Week 6

# VIRTUAL MACHINE

*A virtual machine (VM) is an efficient, isolated duplicate of a real machine [PG74].*

# VIRTUAL MACHINE

*A virtual machine (VM) is an efficient, isolated duplicate of a real machine [PG74].*

**Duplicate:** VM should behave identical to the real machine (programs cannot distinguish between execution on real or virtual hardware), except for:

- less resources available (and potentially different between executions)
- some timing differences (when dealing with devices)

**Isolated:** several VMs execute without interfering with each other

**Efficient:** VM should execute at a speed close to that of real hardware

- requires that most instructions are executed directly by real hardware

# VIRTUAL MACHINES, SIMULATORS AND EMULATORS

**Simulator** provides an *functionally accurate* software model of a machine

- ✓ may run on any hardware
- ✗ is typically slow (order of 1000 slowdown)

**Emulator** provides a *behavioural* model of a machine (and possibly its software)

- ✗ not fully accurate
- ✓ reasonably fast (order of 10 slowdown)

**Virtual machine** models a machine exactly and efficiently

- ✓ minimal slowdown
- ✗ needs to be run on the physical machine it virtualises

# TYPES OF VIRTUAL MACHINES

- Contemporary use of the term VM is more general

# TYPES OF VIRTUAL MACHINES

- Contemporary use of the term VM is more general
- Call virtual machines even if there is no correspondence to an existing real machine
  - e.g. *Java virtual machine*
  - can be viewed as virtualising at the ABI level
  - also called *process VM*

# TYPES OF VIRTUAL MACHINES

- Contemporary use of the term VM is more general
- Call virtual machines even if there is no correspondence to an existing real machine
  - e.g. *Java virtual machine*
  - can be viewed as virtualising at the ABI level
  - also called *process VM*
- We only concern ourselves with virtualising at the ISA level
  - ISA = *instruction-set architecture* (hardware-software interface)
  - also called *system VM* [SN05]
  - will later see subclasses of this

# WHY VIRTUAL MACHINES?

- Historically used for easier sharing of expensive mainframes
- Gone out of fashion in 80's

# WHY VIRTUAL MACHINES?

- Historically used for easier sharing of expensive mainframes
- Gone out of fashion in 80's
- Renaissance in recent years for improved isolation [RG05]
  - ★ improved QoS and security
  - ★ uniform view of hardware
  - ★ complete encapsulation
    - for replication, migration, checkpoint/restart, debugging
  - ★ total mediation

# WHY VIRTUAL MACHINES?

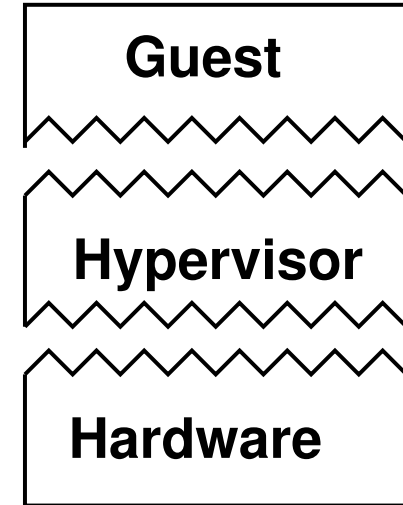
- Historically used for easier sharing of expensive mainframes
- Gone out of fashion in 80's
- Renaissance in recent years for improved isolation [RG05]
  - ★ improved QoS and security
  - ★ uniform view of hardware
  - ★ complete encapsulation
    - for replication, migration, checkpoint/restart, debugging
  - ★ total mediation
- Isn't that exactly the job description of an OS????

# WHY VIRTUAL MACHINES?

- Historically used for easier sharing of expensive mainframes
- Gone out of fashion in 80's
- Renaissance in recent years for improved isolation [RG05]
  - ★ improved QoS and security
  - ★ uniform view of hardware
  - ★ complete encapsulation
    - for replication, migration, checkpoint/restart, debugging
  - ★ total mediation
- Isn't that exactly the job description of an OS????
  - This really means that mainstream OSes suck!

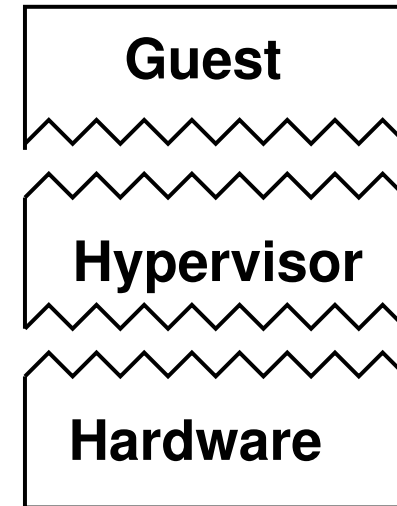
# VIRTUAL MACHINE MONITOR (VMM) AKA HYPERVISOR

- Program that runs on real hardware to implement the virtual machine
- Controls resources and schedules guests



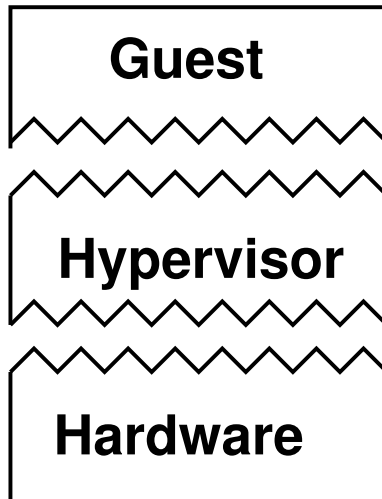
# VIRTUAL MACHINE MONITOR (VMM) AKA HYPERVISOR

- Program that runs on real hardware to implement the virtual machine
- Controls resources and schedules guests
- Implications:
  - hypervisor executes in privileged mode
  - guest software executes in unprivileged mode
  - *privileged instructions* in guest cause a trap into the hypervisor
  - hypervisor interprets/emulates them

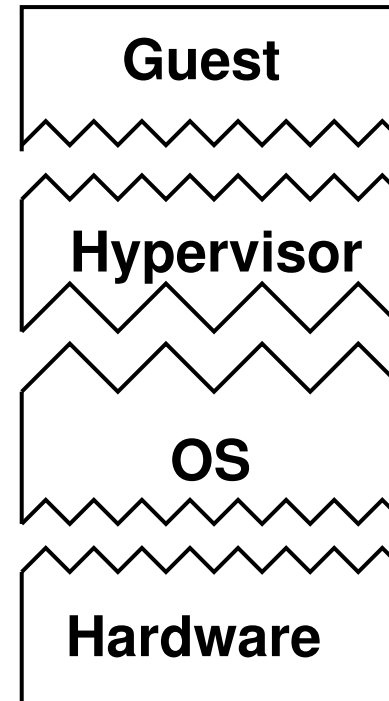


# NATIVE VS. HOSTED VMM

## NATIVE/CLASSIC/BARE-METAL

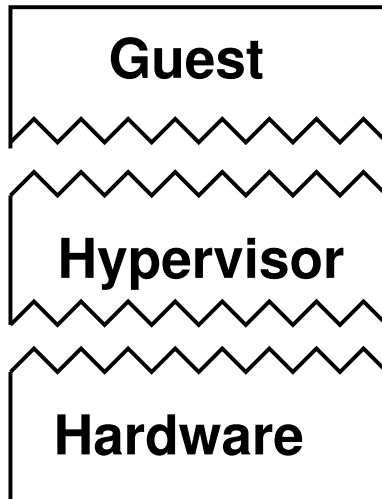


## HOSTED

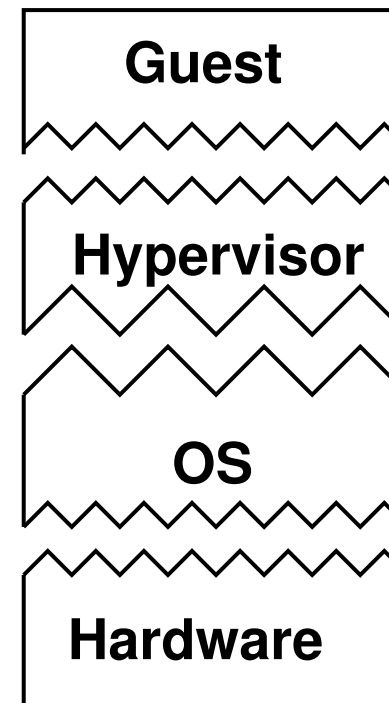


# NATIVE VS. HOSTED VMM

## NATIVE/CLASSIC/BARE-METAL



## HOSTED



- Hosted VMM can run besides native apps
  - ★ sandbox untrusted apps
  - ★ run second OS
  - ★ less efficient:
    - ☒ guest privileged instruction traps into OS, forwarded to hypervisor
    - ☒ return to guest requires a native SO system call

# VMM TYPES

**Classic** as above

**Hosted** e.g. VMware GSX Server

**Whole-system** virtual hardware and operating system

- really an emulation
- e.g. Virtual PC (for Macintosh)

**Physically partitioned** allocate actual processors to each VM

**Logically partitioned** time-share processors between VMs

**Co-designed** hardware specifically designed for VMM

- e.g. Transmeta Crusoe, IBM i-Series

# REQUIREMENTS FOR VIRTUALISATION

## DEFINITIONS

**Privileged instruction** executes in privileged mode, traps in user mode

→ Note: trap is required, NOOP is insufficient!

**Privileged state** determines resource allocation

→ includes privilege mode, addressing context, exception vectors, ...

# REQUIREMENTS FOR VIRTUALISATION

## DEFINITIONS

**Privileged instruction** executes in privileged mode, traps in user mode

→ Note: trap is required, NOOP is insufficient!

**Privileged state** determines resource allocation

→ includes privilege mode, addressing context, exception vectors, ...

**Sensitive instruction** control-sensitive or behaviour-sensitive

**control sensitive** *changes* privileged state

**behaviour sensitive** *exposes* privileged state

→ includes instructions which are NOOPs in user but not privileged mode

# REQUIREMENTS FOR VIRTUALISATION

## DEFINITIONS

**Privileged instruction** executes in privileged mode, traps in user mode

→ Note: trap is required, NOOP is insufficient!

**Privileged state** determines resource allocation

→ includes privilege mode, addressing context, exception vectors, ...

**Sensitive instruction** control-sensitive or behaviour-sensitive

**control sensitive** *changes* privileged state

**behaviour sensitive** *exposes* privileged state

→ includes instructions which are NOOPs in user but not privileged mode

**Innocuous instruction** not sensitive

# REQUIREMENTS FOR VIRTUALISATION

An architecture is *virtualizable* if all sensitive instructions are privileged (called *pure virtualisation*)

# REQUIREMENTS FOR VIRTUALISATION

An architecture is *virtualizable* if all sensitive instructions are privileged (called *pure virtualisation*)

- Can then achieve accurate, efficient guest execution
  - guest's sensitive instructions trap and are emulated by VMM
  - guest's innocuous instructions are executed directly
  - VMM controls resources
- Virtualised execution is indistinguishable from native, except:
  - resources more limited (running on *smaller* machine)
  - timing is different (if there is an observable time source)

# REQUIREMENTS FOR VIRTUALISATION

An architecture is *virtualizable* if all sensitive instructions are privileged (called *pure virtualisation*)

- Can then achieve accurate, efficient guest execution
  - guest's sensitive instructions trap and are emulated by VMM
  - guest's innocuous instructions are executed directly
  - VMM controls resources
- Virtualised execution is indistinguishable from native, except:
  - resources more limited (running on *smaller* machine)
  - timing is different (if there is an observable time source)
- Architecture is *recursively virtualizable* if VMM without timing dependency can be built

# VIRTUALISATION OVERHEADS

- VMM needs to maintain virtualised privileged machine state
  - processor status
  - addressing context
- VMM needs to simulate privileged instructions
  - synchronise virtual and real privileged state as appropriate
  - e.g. *shadow page tables* to virtualize hardware

# VIRTUALISATION OVERHEADS

- VMM needs to maintain virtualised privileged machine state
  - processor status
  - addressing context
- VMM needs to simulate privileged instructions
  - synchronise virtual and real privileged state as appropriate
  - e.g. *shadow page tables* to virtualize hardware
- Frequent virtualisation traps can be expensive
  - STI/CLI for mutual exclusion
  - frequent page table updates
  - MIPS `KSEG` addresses used for physical addressing in kernel

# UNVIRTUALISABLE ARCHITECTURES

- x86: lots of unvirtualisable features
  - e.g. sensitive PUSH of PSW is not privileged
  - segment and interrupt descriptor tables in virtual memory
  - segment descriptors expose privileged level

# UNVIRTUALISABLE ARCHITECTURES

- x86: lots of unvirtualisable features
  - e.g. sensitive PUSH of PSW is not privileged
  - segment and interrupt descriptor tables in virtual memory
  - segment descriptors expose privileged level
- Itanium: mostly virtualizable, but
  - interrupt vector table in virtual memory
  - THASH instruction exposes hardware page table address

# UNVIRTUALISABLE ARCHITECTURES

- x86: lots of unvirtualisable features
  - e.g. sensitive PUSH of PSW is not privileged
  - segment and interrupt descriptor tables in virtual memory
  - segment descriptors expose privileged level
- Itanium: mostly virtualizable, but
  - interrupt vector table in virtual memory
  - THASH instruction exposes hardware page table address
- MIPS: mostly virtualizable, but
  - kernel registers `k0`, `k1` (needed to save/restore state) user-accessible
  - performance issue with virtualising `KSEG` addresses

# UNVIRTUALISABLE ARCHITECTURES

- x86: lots of unvirtualisable features
  - e.g. sensitive PUSH of PSW is not privileged
  - segment and interrupt descriptor tables in virtual memory
  - segment descriptors expose privileged level
- Itanium: mostly virtualizable, but
  - interrupt vector table in virtual memory
  - THASH instruction exposes hardware page table address
- MIPS: mostly virtualizable, but
  - kernel registers `k0`, `k1` (needed to save/restore state) user-accessible
  - performance issue with virtualising `KSEG` addresses
- Most others have problems too

# IMPURE VIRTUALISATION

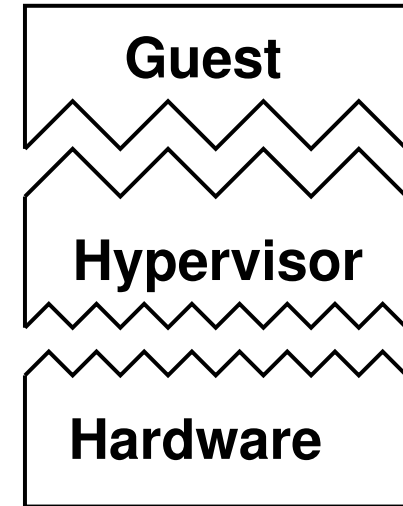
- Used for two reasons:
  - unvirtualisable architectures
  - performance problems of virtualisation

# IMPURE VIRTUALISATION

- Used for two reasons:
  - unvirtualisable architectures
  - performance problems of virtualisation
- Two standard approaches:
  - ① paravirtualisation
  - ② binary translation

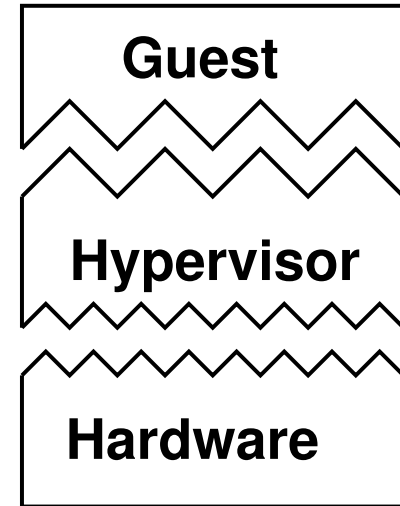
# PARAVIRTUALISATION

- New name, old technique
  - Used in Mach Unix server, L<sup>4</sup>Linux [HHL<sup>+</sup>97], Disco [BDGR97]
  - Name coined by Denali [WSG02], popularised by Xen [BDF<sup>+</sup>03]



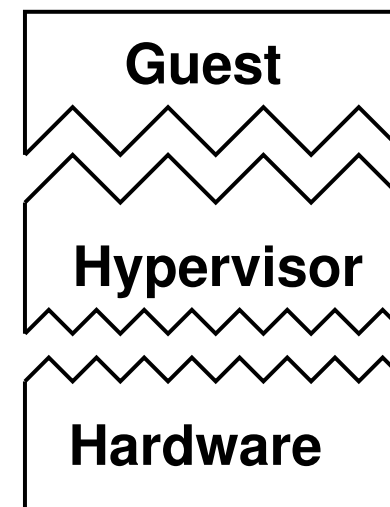
# PARAVIRTUALISATION

- New name, old technique
  - Used in Mach Unix server, L<sup>4</sup>Linux [HHL<sup>+</sup>97], Disco [BDGR97]
  - Name coined by Denali [WSG02], popularised by Xen [BDF<sup>+</sup>03]
- Manually port the guest OS to modified ISA
  - augmented by explicit hypervisor calls (*hypercalls*)
  - ✓ idea is to provide more high-level API to reduce the number of traps
  - ✓ remove unvirtualisable instructions
  - ✓ remove “messy” ISA features which complicate virtualisation



# PARAVIRTUALISATION

- New name, old technique
  - Used in Mach Unix server, L<sup>4</sup>Linux [HHL<sup>+</sup>97], Disco [BDGR97]
  - Name coined by Denali [WSG02], popularised by Xen [BDF<sup>+</sup>03]
- Manually port the guest OS to modified ISA
  - augmented by explicit hypervisor calls (*hypercalls*)
  - ✓ idea is to provide more high-level API to reduce the number of traps
  - ✓ remove unvirtualisable instructions
  - ✓ remove “messy” ISA features which complicate virtualisation
- Drawbacks:
  - ✗ significant engineering effort
  - ✗ needs to be repeated for each guest, ISA, hypervisor combination
  - ✗ paravirtualised guest needs to be kept in sync with native guest
  - ✗ requires source



# BINARY TRANSLATION

- Locate unvirtualisable instructions in guest binary and replace on-the-fly by emulation code or hypercall
- pioneered by VMware [RG05]

# BINARY TRANSLATION

- Locate unvirtualisable instructions in guest binary and replace on-the-fly by emulation code or hypercall
  - pioneered by VMware [RG05]
  - ✓ can also detect combinations of sensitive instructions and replace by single emulation
  - ✓ doesn't require source
  - ✓ may (safely) do some emulation in user space for efficiency
  - ✗ very tricky to get right (especially on x86!)
  - ✗ needs to make some assumptions on sane behaviour of guest

# VIRTUALISATION TECHNIQUES: MEMORY

- Shadow page tables
  - Guest accesses shadow PT
  - VMM detects changes (e.g. making them R/O) and syncs with real PT
  - can over-commit memory (similar to virtual-memory paging)

# VIRTUALISATION TECHNIQUES: MEMORY

- Shadow page tables
  - Guest accesses shadow PT
  - VMM detects changes (e.g. making them R/O) and syncs with real PT
  - can over-commit memory (similar to virtual-memory paging)
  - Note: Xen exposes hardware page tables

# VIRTUALISATION TECHNIQUES: MEMORY

- Shadow page tables
  - Guest accesses shadow PT
  - VMM detects changes (e.g. making them R/O) and syncs with real PT
  - can over-commit memory (similar to virtual-memory paging)
  - Note: Xen exposes hardware page tables
- Memory reclamation: *Ballooning* (VMware ESX Server)
  - load cooperating pseudo-device driver into guest
  - to reclaim, balloon driver requests physical memory from guest
  - VMM can then reuse that memory
  - guest determines which pages to release

# VIRTUALISATION TECHNIQUES: MEMORY

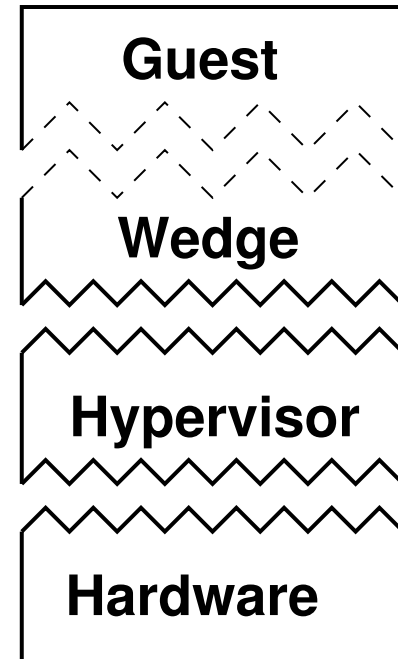
- Shadow page tables
  - Guest accesses shadow PT
  - VMM detects changes (e.g. making them R/O) and syncs with real PT
  - can over-commit memory (similar to virtual-memory paging)
  - Note: Xen exposes hardware page tables
- Memory reclamation: *Ballooning* (VMware ESX Server)
  - load cooperating pseudo-device driver into guest
  - to reclaim, balloon driver requests physical memory from guest
  - VMM can then reuse that memory
  - guest determines which pages to release
- Page sharing
  - VMM detects pages with identical content
  - establishes (copy-on-write) mappings to single page via shadow PT
  - significant savings when running many identical guest OSes

# VIRTUALISATION TECHNIQUES: DEVICES

- Drivers in VMM
  - maybe ported legacy drivers
- Host drivers
  - for hosted VMMs
- Legacy driver in separate driver VM
  - e.g. separate Linux guest for each device [LUSG04]
- Virtualisation-friendly devices
  - IBM channel architecture
  - can safely be accessed by guest drivers if physical memory access is restricted (I/O-MMU)

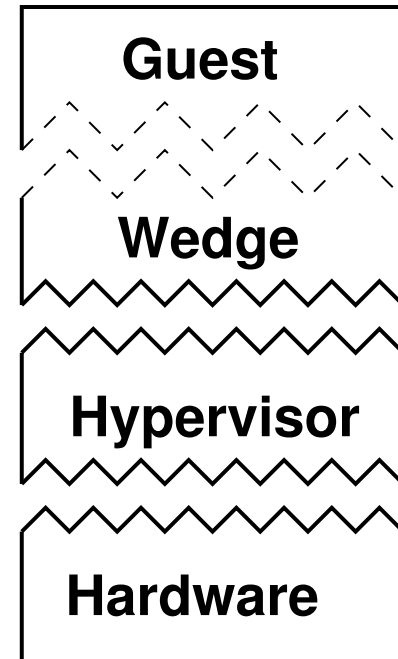
# PREVIRTUALISATION

- Combines advantages of pure and paravirtualisation



# PREVIRTUALISATION

- Combines advantages of pure and paravirtualisation
- Multi-stage process
  - ① During build, pad sensitive instructions with NOPs and keep record
  - ② During profiling run, trap sensitive memory operations (e.g. PT accesses) and record
  - ③ Redo build, also padding sensitive memory operations
  - ④ Link emulation library (*wedge*) with guest
  - ⑤ At load time, replace NOP-padded instructions by emulation code

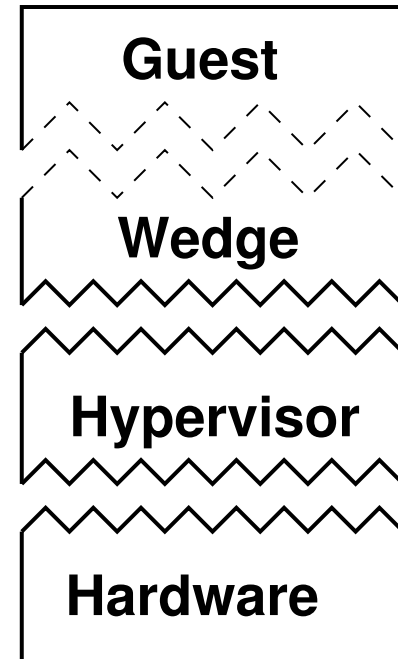


# PREVIRTUALISATION

- Combines advantages of pure and paravirtualisation

- Multi-stage process

- ① During build, pad sensitive instructions with NOPs and keep record
- ② During profiling run, trap sensitive memory operations (e.g. PT accesses) and record
- ③ Redo build, also padding sensitive memory operations
- ④ Link emulation library (*wedge*) with guest
- ⑤ At load time, replace NOP-padded instructions by emulation code



- Features:

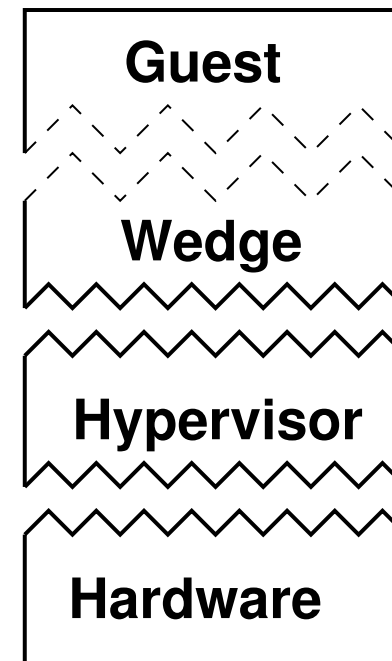
- ✓ significantly reduced engineering effort
- ✓ single binary runs on bare metal as well as *all* hypervisors
- ✗ requires source
- ✗ performance may require some paravirtualisation

# PREVIRTUALISATION

- Combines advantages of pure and paravirtualisation

- Multi-stage process

- ① During build, pad sensitive instructions with NOPs and keep record
- ② During profiling run, trap sensitive memory operations (e.g. PT accesses) and record
- ③ Redo build, also padding sensitive memory operations
- ④ Link emulation library (*wedge*) with guest
- ⑤ At load time, replace NOP-padded instructions by emulation code



- Features:

- ✓ significantly reduced engineering effort
- ✓ single binary runs on bare metal as well as *all* hypervisors
- ✗ requires source
- ✗ performance may require some paravirtualisation

→ See <http://l4ka.org/projects/virtualization/afterburn/>

# HARDWARE SUPPORT

- Intel VT-x/VT-i: virtualisation support for x86/Itanium [UNR<sup>+</sup>05]
  - ★ introduces new processor mode: *root mode* for hypervisor
  - ★ if enabled, all sensitive instructions in non-root mode trap to root mode
  - ★ VT-i (Itanium) also reduces virtual address-space size for non-root

# HARDWARE SUPPORT

- Intel VT-x/VT-i: virtualisation support for x86/Itanium [UNR<sup>+</sup>05]
  - ★ introduces new processor mode: *root mode* for hypervisor
  - ★ if enabled, all sensitive instructions in non-root mode trap to root mode
  - ★ VT-i (Itanium) also reduces virtual address-space size for non-root
- ARM Trustzone
  - similar super-privileged mode
  - also memory and device segregation

# HARDWARE SUPPORT

- Intel VT-x/VT-i: virtualisation support for x86/Itanium [UNR<sup>+</sup>05]
  - ★ introduces new processor mode: *root mode* for hypervisor
  - ★ if enabled, all sensitive instructions in non-root mode trap to root mode
  - ★ VT-i (Itanium) also reduces virtual address-space size for non-root
- ARM Trustzone
  - similar super-privileged mode
  - also memory and device segregation
- Aim is virtualisation of unmodified legacy OSes

# REFERENCES

- [BDF<sup>+</sup>03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *19th SOSP*, pages 164–177, Bolton Landing, NY, USA, Oct 2003.
- [BDGR97] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *Trans. Comp. Syst.*, 15:412–447, 1997.
- [HHL<sup>+</sup>97] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The performance of  $\mu$ -kernel-based systems. In *16th SOSP*, pages 66–77, St. Malo, France, Oct 1997.
- [LUSG04] Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *6th OSDI*, San Francisco, CA, USA, Dec 2004.
- [PG74] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *CACM*, 17(7):413–421, 1974.

- [RG05] Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. *IEEE Comp.*, 38(5):39–47, 2005.
- [SN05] James E. Smith and Ravi Nair. The architecture of virtual machines. *IEEE Comp.*, 38(5):32–38, 2005.
- [UNR<sup>+</sup>05] Rich Uhlig, Gil Neiger, Dion Rodgers, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kägi, Felix H Leung, and Larry Smith. Intel virtualization technology. *IEEE Comp.*, 38(5):48–56, May 2005.
- [WSG02] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the Denali isolation kernel. In *5th OSDI*, Boston, MA, USA, Dec 2002.