A Comparison of Software and Hardware Techniques for x86 Virtualization

Keither Adams, Ole Ageson
VMWare

Presented by: Benjamin Prosnitz

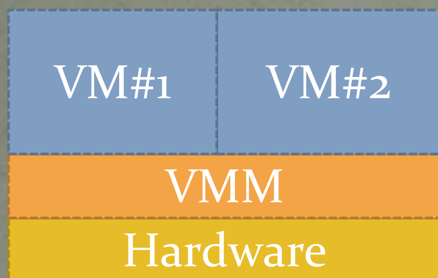
Overview of Virtual Machines

- Virtual Machine Monitors (VMM) provide virtual machine (VM) software with the impression of running directly on hardware

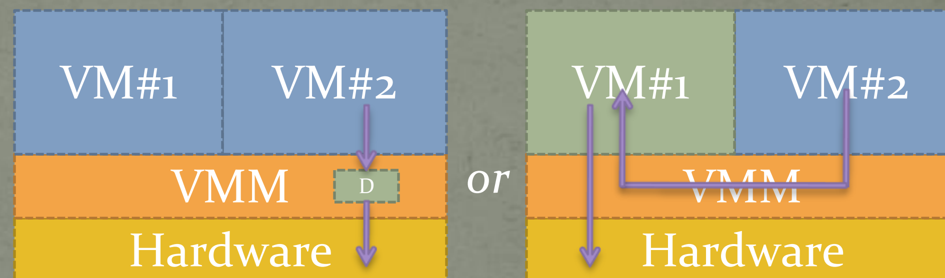
- Popek and Goldberg requirements for a VMM:
  - *Fidelity* (runs normally, as if there were no VMM)
  - *Performance* (not interpreted)
  - *Safety* (resources are protected as appropriate)

Types of VMMs

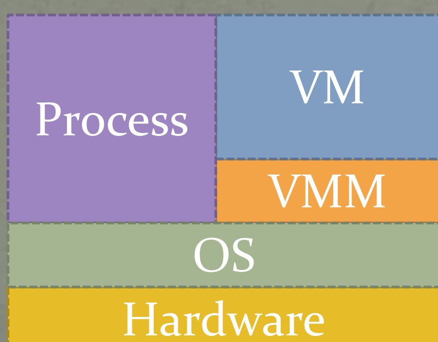
- "Type I": (hypervisor runs directly on hardware)



Drivers can either be in the hypervisor or in one of the VMs



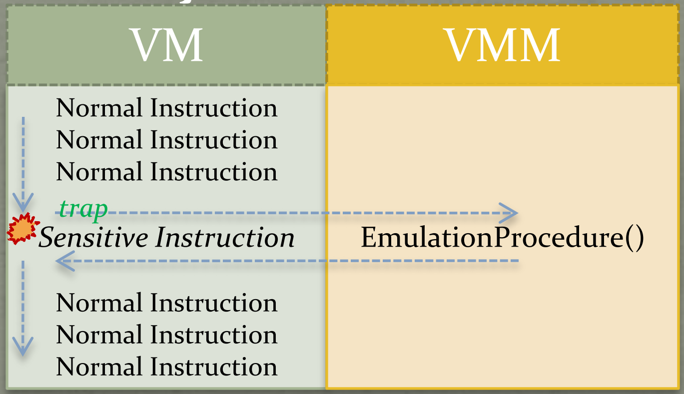
- "Type II": (hypervisor runs on a host OS)



Type II VMMs that use *hardware* virtualization require modifications to the host OS (patches, or Kernel Driver as in KVM) to perform privileged virtualization instructions

Classic Virtualizability: Trap & Emulate

- For an architecture to be classically virtualizable, all "sensitive" instructions must be privileged and trap so that they are emulatable by the VMM:



- Sensitive instructions include:
  - Processor mode changes
  - Hardware accesses
  - Instructions whose behavior is different in user/kernel mode

Classic Virtualizability on x86

- Historically x86 hasn't been virtualizable[1], since there are sensitive instructions that do not trap:
  - Instructions like popf, which has different kernel and user-mode behavior
  - SMSW, which stores the machine status
  - SGDT, SLDT for segment descriptors
- Recently, classic virtualizability has become possible due to two new (incompatible) architectures:
  - AMD SVM (secure virtual machine)
  - Intel VT (virtualization technology)

1. Robin, Irvine. *Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor.* Procedings of the 9th Usenix Security Symposium. 2000.

x86 Virtualization without Hardware Support

- A first attempt: direct fetch-and-decode emulation of all instructions
  - Slow
  - Not technically virtualization by Popek & Goldberg requirements
- A minor improvement: decoding instructions and saving them in a cache in an easy-to-process format:

Instruction	Operand	Operand
…	…	…
Instruction	Operand	Operand

  - Minor speed up (but still not technically virtualization)
  - Need to manage the cache

x86 Virtualization without Hardware Support

- Idea: Compile the instructions in the cache and run them directly
- Challenges:
  - Protection of the cache
  - Ensuring correctness of direct memory addresses
  - Relative memory addressing needs to still work, despite the fact that the cache may be structured differently
  - Sensitive instructions need to be handled
- Also: Need to compile on demand

Basic Blocks (Translation Units)

- Divide the instructions into blocks that end with a control-flow instruction:

```
isPrime:   mov %ecx, %edi
           mov %esi, $2
           cmp %esi, %ecx
           jge prime
nexti:     mov %eax, %ecx
           …
```

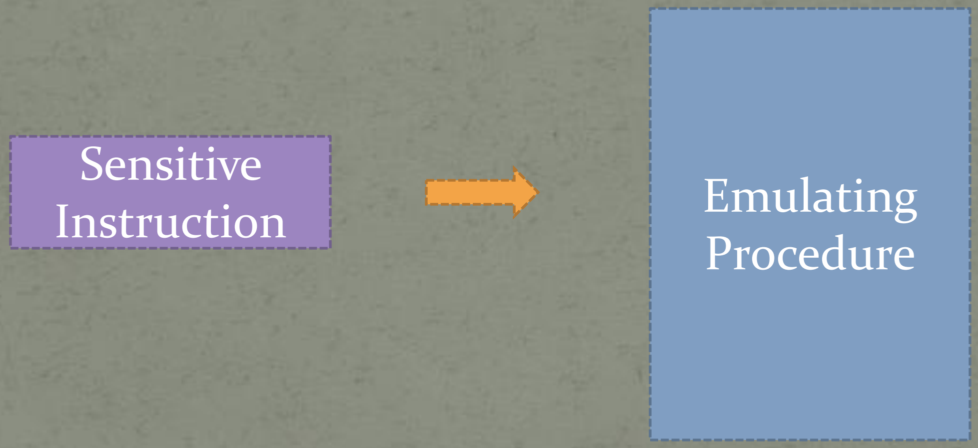
⇨

```
isPrime:   mov %ecx, %edi
           mov %esi, $2
           cmp %esi, %ecx
           jge [HANDLER1]
           jme [HANDLER2]
```

- The control flow instructions are replaced with jumps into VMM code which determines what to do next
  - If the destination code is already in the cache, the VMM uses it. Otherwise, it compiles it.
  - The VMM replaces the pointer to the handler in *isPrime* with a direct pointer to the destination code block in a process called *chaining*

Handling Sensitive Instructions

- If a sensitive instruction already traps, it can be handled when it traps
- If it doesn't trap, replace the sensitive instruction with a call to a procedure that performs the operation



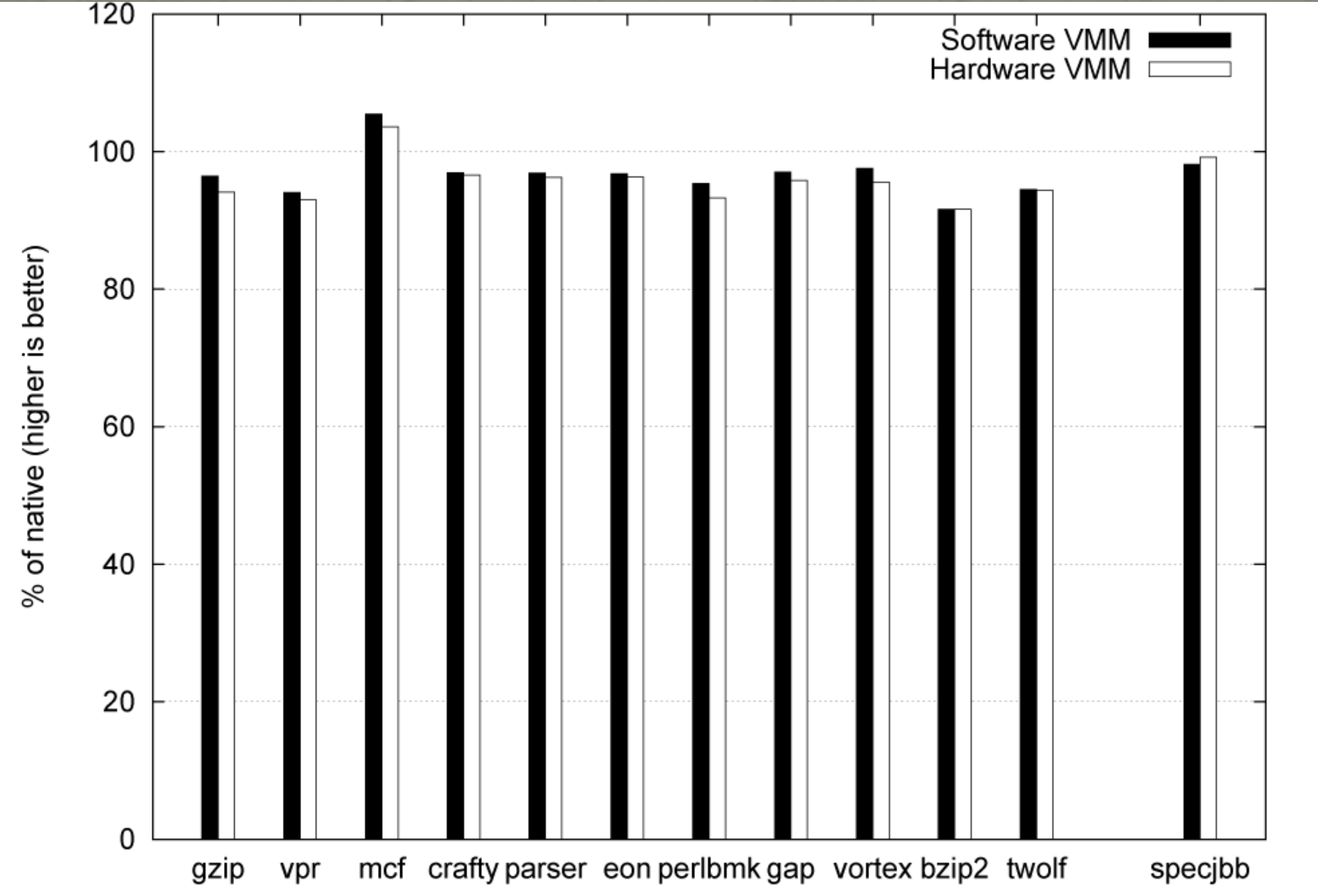
Reducing the Number of Traps

- Modify certain trapping instructions (like system calls) so that they run code in the cache and don't trap unless they have to (to access hardware or page tables)
- VMWare developed a method to adaptively identify pieces of code with which this is possible

x86 Hardware Virtualization

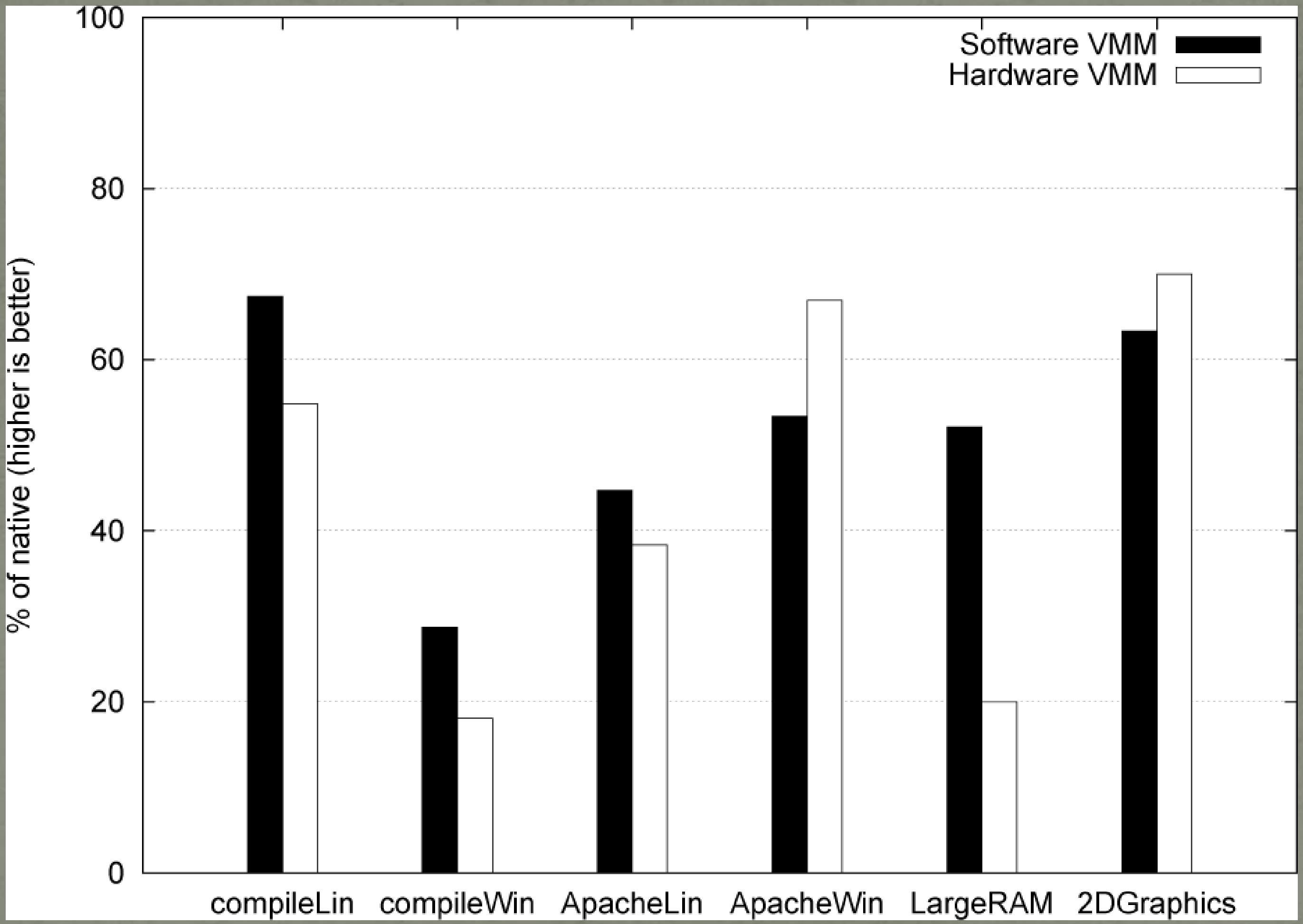
- Define a virtual machine control block (VMCB) which:
  - Includes control information (when should a VMEXIT be performed)
  - Includes VM state information (which is filled when a VMEXIT is performed)
- To run the virtual machine, perform a VMENTRY after the control block has been specified
- Many Caveats!
  - Intel VT doesn't handle real mode virtualizaton
  - Instruction emulation is needed for some instructions
  - No MMU virtualization – page table updates are slow
  - …

Performance: "Direct Execution"

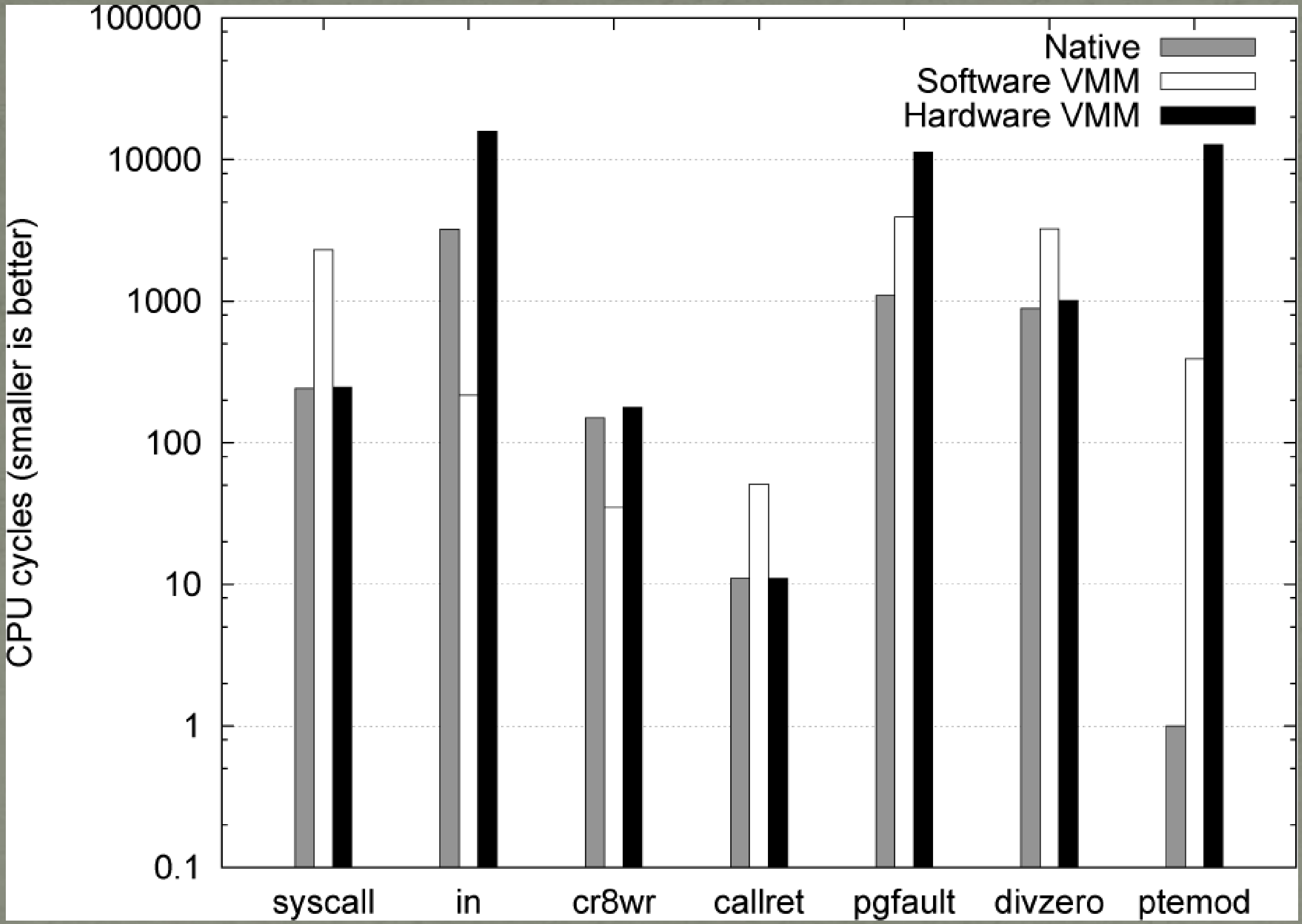


SPECint 2000 and SPECjbb 2005.

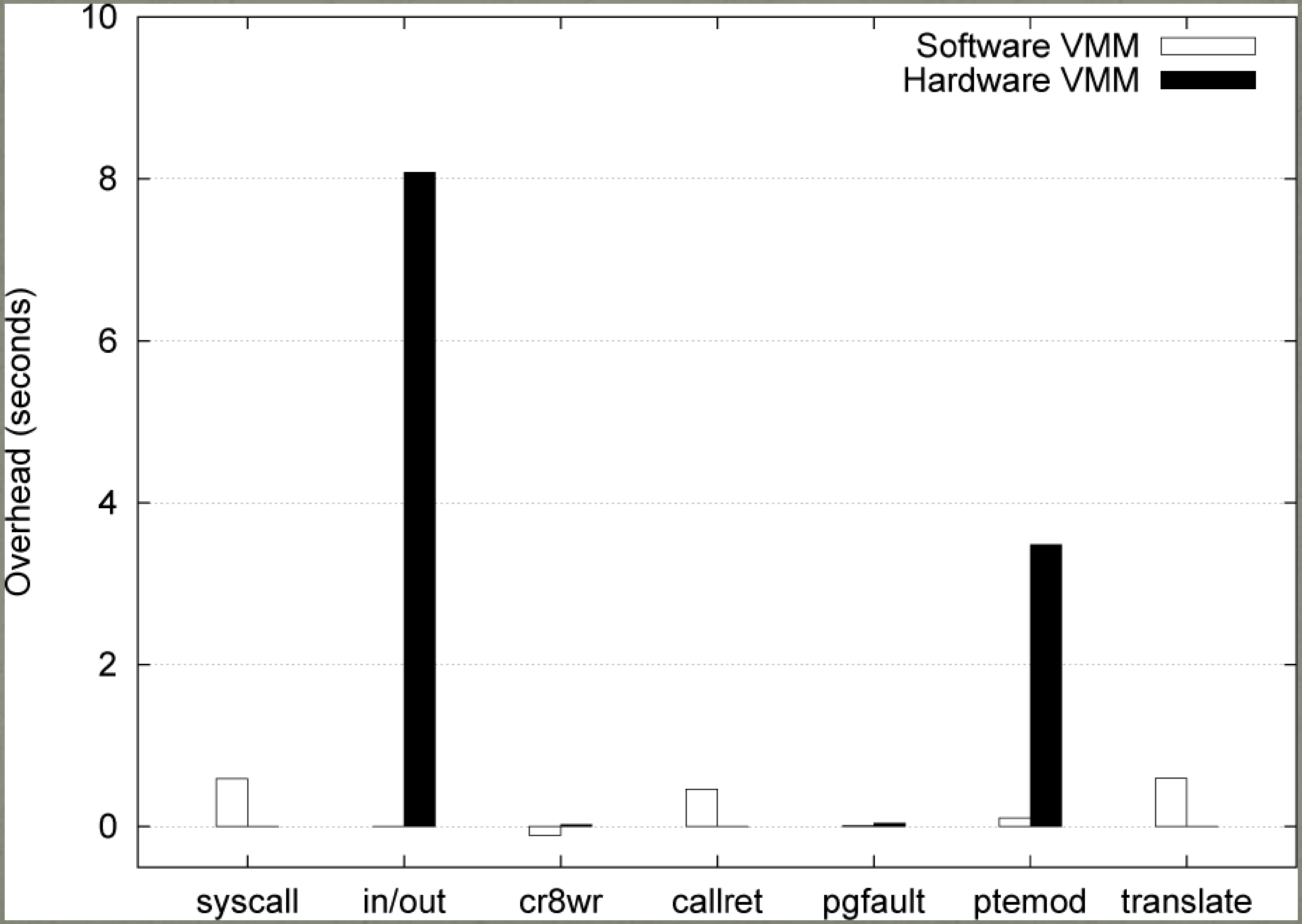
Performance: Macrobenchmarks



Performance: Nanobenchmarks



Performance: Overhead during XP boot



Hybrid VMMs

- It is possible to use both hardware virtualization and software virtualization at different times, switching between modes when heuristics indicate that one may get better performance than the other

Hardware Virtualization Improvements

- VMENTRY/VMEXIT performance
  - Already improvements in new processors
- MMU Virtualization
  - Nested page tables (extra gpa->hpa map)
- I/O Virtualization

Thank you!