Chapter 12

Virtualization

Virtualization…

- …is used in many contexts in computer systems
- In Operating Systems (already covered in the lecture):
  - Virtualization of memory
  - Virtualization of block devices (or I/O devices in general)
  - Virtualization of processors
  - …
  - Virtualization of whole machines → topic of this lecture

- Focus of this lecture:
  Virtualization of whole machines in the sense that the user can access some (or many) instances of possibly different operating systems on one physical machine, i.e., getting the illusion of some (or many) different machines with possibly different operating systems.

Virtualization

- Not focus of this lecture:
  - Everything mentioned above except virtual machines
  - Virtual execution environments such as Java VM
  - Market overview for virtualization products

Motivation (Examples)

- Access to more than one OS on one machine
- Consolidation of servers in computation centers, with additional features such as improved fault tolerance and load balancing
- Isolation of different administrative domains (e.g., in hosting business)
- Execution environment for legacy software
- Execution environment for future software
- Teaching
- Debugging, Testing, Fault injection
- OS development
- Network programming
- Secure test environment
- Migration of virtual machines is much simpler than migration of individual applications
- Cloud computing, especially infrastructure as a service (IaaS)

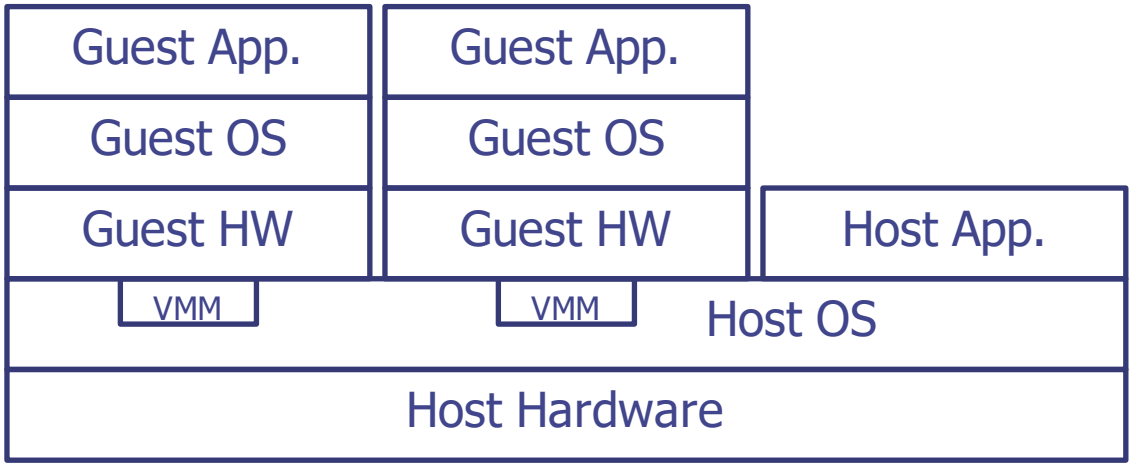
Goals

- "There are three properties of interest when any arbitrary program is run while the control program (VMM, virtual machine monitor) is resident: efficiency, resource control, and equivalence.
  - The efficiency property. All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program.
  - The resource control property. It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocator of the control program is to be invoked upon any attempt.
  - The equivalence property. Any program *K* executing with a control program resident, with two possible exceptions, performs in a manner indistinguishable from the case when the control program did not exist and *K* had whatever freedom of access to privileged instructions that the programmer had intended."

- Popek und Goldberg 1974:
  "Formal Requirements for Virtualizable Third Generation Architectures"
  Communications of the ACM 17 (7): 412-421.

Terminology

- VMM Virtual Machine Monitor, provides execution environment for guest and manages it. Also called "hypervisor".
- "VM" and "guest" are used as synonyms in this lecture



Example Architecture

History

- Virtualization was not invented by VMware, Intel or some other company in recent time.
- Virtualization is part of IBM mainframe systems since 1972!
  - First predecessors in 1966
  - First release of "hypervisor" was VM/370 (Virtual Machine Facility/370) for System/370) in 1972
  - Current version: z/VM
  - Can execute all other mainframe OS as guest (including Linux today)

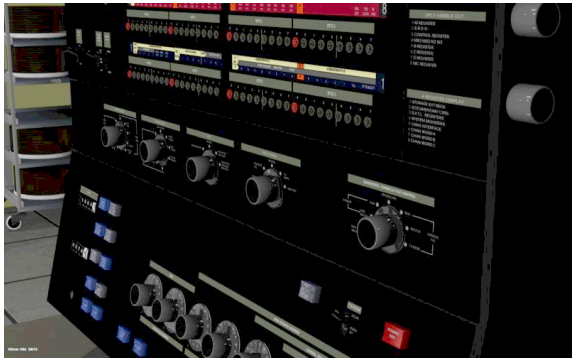
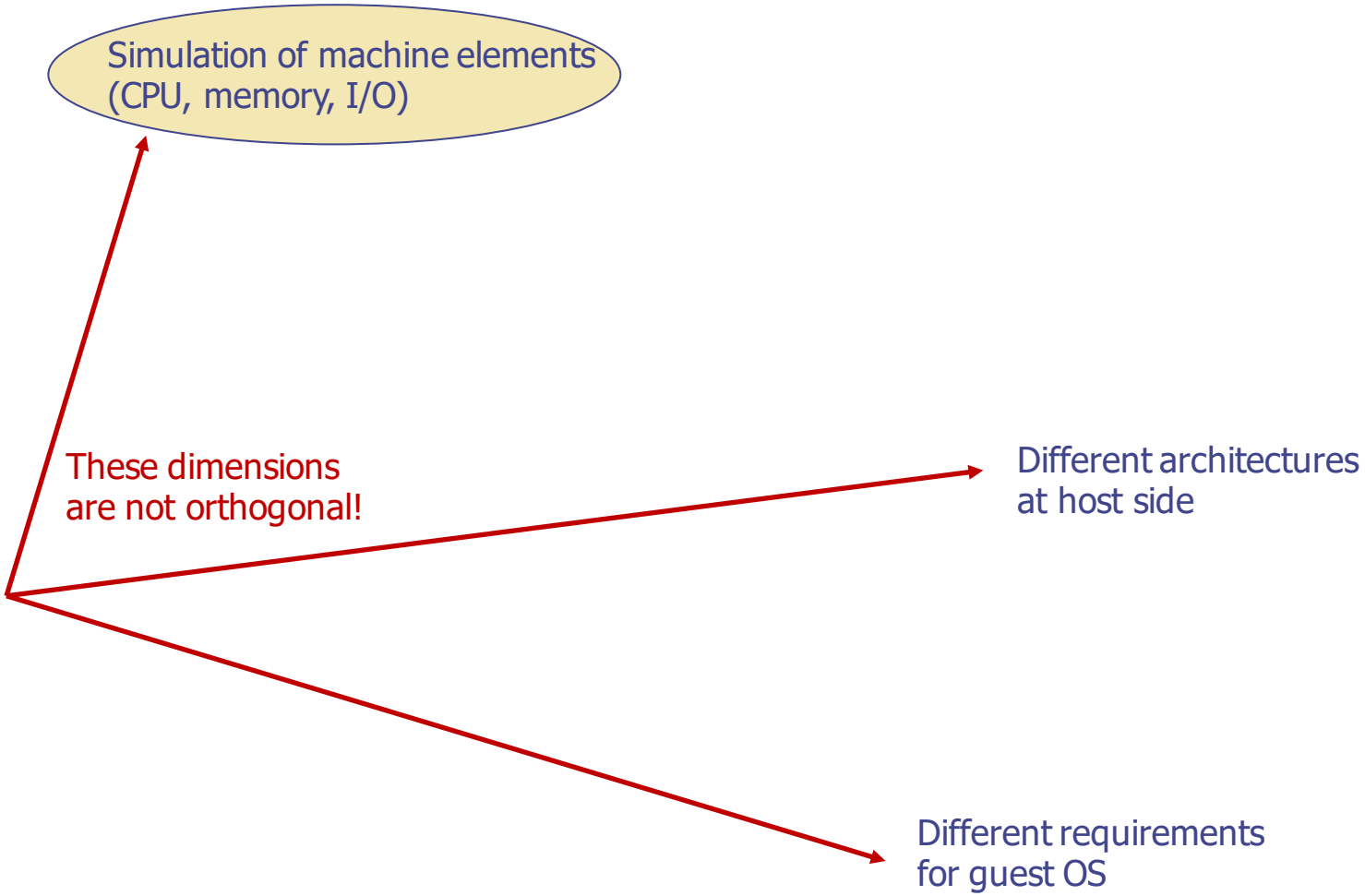


Image: Wikipedia

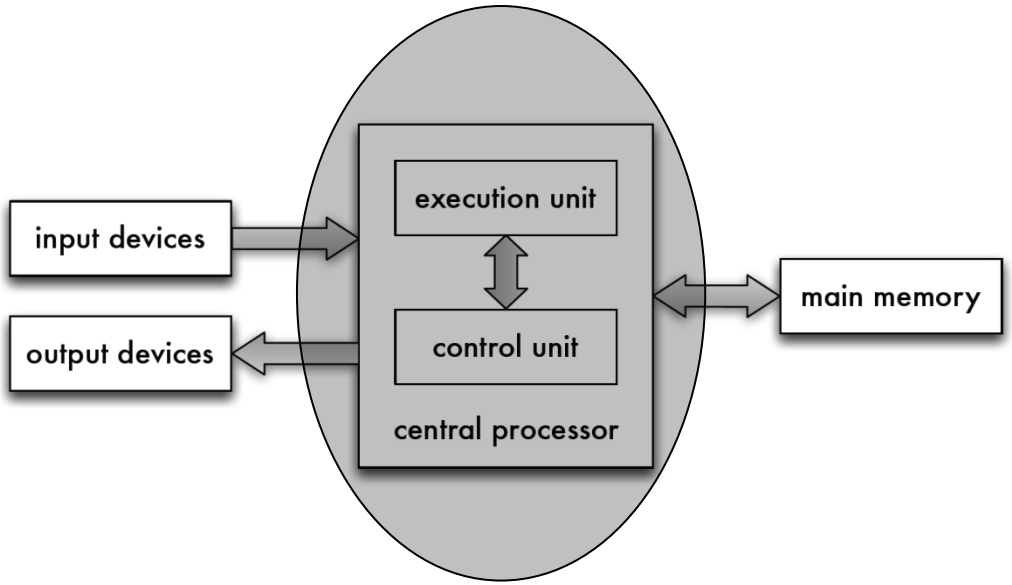
- x86 virtualization is more complicated due to the instruction set and its limitations.
  - Software solutions appeared in the late 90s and early 2000s
  - Hardware support was introduced in 2005/2006
  - Today many alternative solutions available and widely used

- Currently: Virtualization solutions for mobile systems such as smart phones

Dimensions of consideration



Simulating a von-Neumann Machine



We need simulation of

- Processor
- Memory
- Input and Output

realized in

- Software
- Software with different levels of hardware support

Processor: Different Instruction Set

Freie Universität Berlin

- Implementing a "processor simulator" in user space: Emulator
  - Fetch-execute cycle is easy to implement
  - Case-statement to implement individual opcodes
  - Register etc. as variables

- Problem: Poor performance
  - Software does not allow effective implementations of hardware mechanisms such as interrupts (theoretically, check for interrupt is necessary after each instruction).
  - Each guest instruction is executed by several host instructions.

- Popek and Goldberg:
  [..] is efficiency. It demands that a statistically dominant subset of the virtual processor's instructions be executed directly by the real processor, with no software intervention by the VMM. This statement rules out traditional emulators and complete software interpreters (simulators) from the virtual machine umbrella.

Processor: Different Instruction Set

- Possible optimizations
  - Just in time compilation (JIT) similar to JIT in JVM
    - Guest code is on the fly compiled to host code
    - Problematic code is still emulated
    - Cache for reusing already compiled parts
  - Compilation from guest to host code
    - Considerations for “same instruction set” apply
  - Check for interrupts only “once in a while”

Processor: Same Instruction Set

- Emulation is possible but not wise: Why realize one instruction by many instructions of the same instruction set?
- Instead of emulation, use the host processor directly: VM becomes process on host, host processor executes its instructions.
- Problem: Some instructions may affect the host (e.g., blocking IRQ)
- Under which conditions is this possible in a simple way?

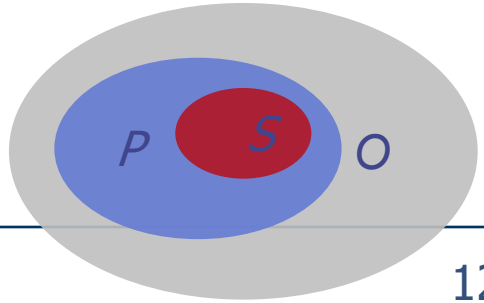
→ Popek and Goldberg 1974:
"Formal Requirements for Virtualizable Third Generation Architectures". Communications of the ACM 17 (7): 412 –421.

Popek and Goldberg 1974

- The set $O = P \cup S \cup I$ of instructions of an instruction set can be separated into
  - P – Privileged instructions: Execution in system mode is possible, execution in user mode traps
  - S – Sensitive instructions $S = C \cup B$ with
    - C – Control sensitive instructions: Change configuration of system resources
    - B – Behavior sensitive instructions: Behavior depends on configuration of system resources
  - I – Insensitive instructions: All remaining instructions

- Theorem of Popek and Goldberg (slightly adapted):

  The construction of an (efficient) virtual machine monitor is possible if $S \subseteq P$ applies for a computer



Processor: Same Instruction Set

- If Popek and Goldberg criterion is met: “Trap and emulate”
  - Execute VM in user mode
  - Sensitive (and privileged) instructions trap
  - Emulate them by an appropriate handler in VMM, then return to VM
  - VMM maintains data structures representing state of VM
  - This way, virtual system mode is realized in real user mode
- On machines not fulfilling the criterion, this simple approach does not work
- Attention!
  The criterion contains an implication, not an equivalence!
  $S \subseteq P$ → Construction of VMM is possible

Processor: Same Instruction Set

- Some instruction sets do not meet the criterion $S \subseteq P$.
- x86: 17 instructions are sensitive but not privileged (ISA: P54C)
- Code example ARMv4: Return from context switch

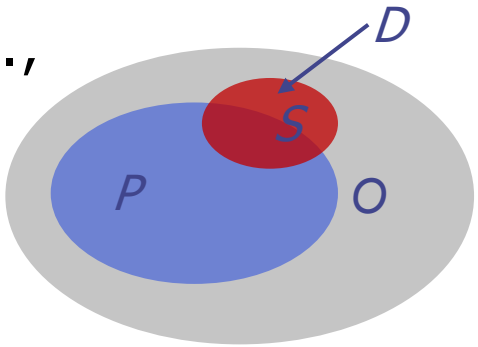
```
; save process state onto stack
STMFD    SP!, {r14}          ; link register for interrupt
STMFD    SP!, {r0-r14}^      ; user registers
MRS      r2, spsr            ; saved CPU state into R2
STMFD    SP!, {r2}                   ; and then to stack
STR      SP, [r0]                    ; pcb->cpu_state = SP
; switch to other process
LDR      SP, [r1]                    ; SP = next_pcb->cpu_state
; restore context
LDMFD    SP!, {r2}                   ; CPU state to R2
MSR      spsr, r2            ; and then into saved state
LDMFD    SP!, {r0-r14}^      ; user registers
LDMFD    SP!, {pc}^          ; link register for return
                             ; from interrupt
```

Behavior sensitive but not privileged!

The instruction set contains more of such instructions (altogether, 24 sensitive but not privileged instructions)

Processor: Same Instruction Set

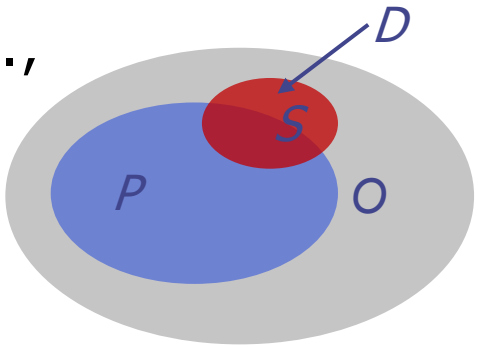
- If Popek and Goldberg criterion is not met, i.e., $D = \{i \in O \mid i \in S, i \notin P\} \neq \emptyset$



- Different approaches are possible, selection:
  - Approach by Popek and Goldberg: "Hybrid VM"
    - Basic idea: Instructions may be sensitive in user and/or system mode
    - If the set of user sensitive instructions is a subset of privileged instructions, a "hybrid virtual machine monitor" can be constructed
    - This leaves the problem of sensitive instructions in virtual system mode
    - Solution: Instructions in system mode are always emulated
      → Not very efficient but better than a pure emulator
  - Adapt VM in a way that no instructions from *D* are executed
  - Adapt hardware in a way that it fulfills the criterion

Processor: Same Instruction Set

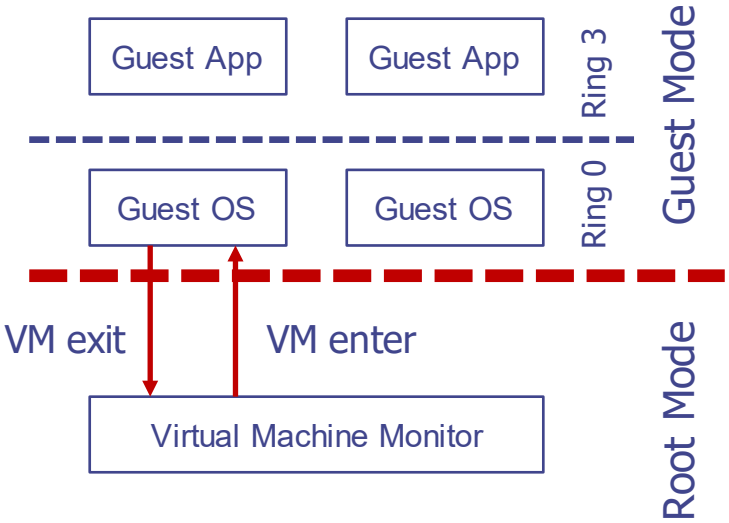
- If Popek and Goldberg criterion is not met, i.e., $D = \{i \in O \mid i \in S, i \notin P\} \neq \emptyset$



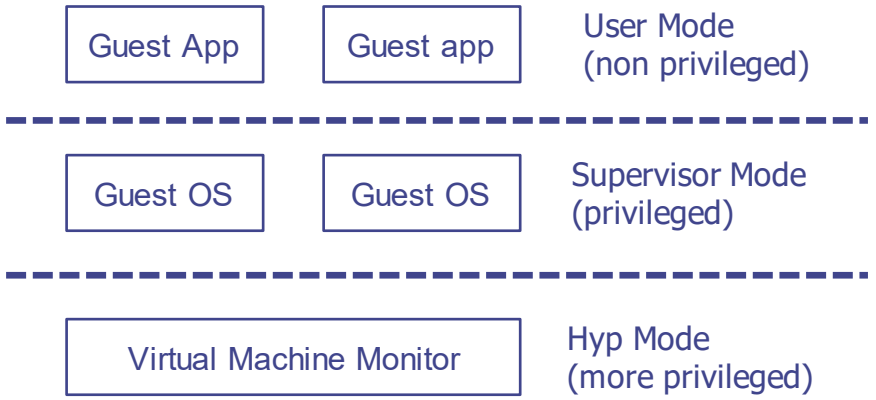
- Different approaches are possible, selection:
  - Approach by Popek and Goldberg: "Hybrid VM"
  - Adapt VM in a way that no instructions from *D* are executed
    - Instructions are replaced by VMM at runtime (binary translation)
    - Instructions are replaced before runtime
    - Adapting the guest OS (paravirtualization)
  - Adapt hardware in a way that it fulfills the criterion
    - Hardware becomes virtualization-aware
      - Add new instructions for virtualization
      - Add new mode for virtualization
        - Orthogonal to traditional modes or
        - New mode "above" system mode (even more privileged)
    - Example: AMD-V, Intel VT-x, ARM Virtualization Extensions as an extension of ARMv7

Examples

- Intel VT-x and AMD-V



- ARMv7-A Virtualization Extensions



Examples: VT-x and AMD-V in Linux

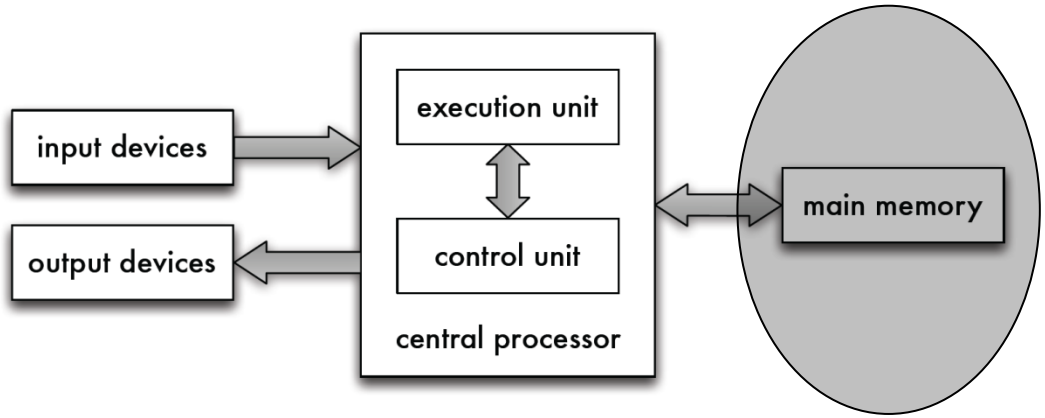
- Intel Xeon X3470

```
$ cat /proc/cpuinfo | grep flags | head -1
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss
ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc aperfmperf pni dtes64 monitor
ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 popcnt
lahf_lm ida dtherm tpr_shadow vnmi flexpriority ept vpid
```

- AMD Opteron 8435

```
$ cat /proc/cpuinfo | grep flags | head -1
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall
nx mmxext fxsr_opt pdpe1gb rdtscp lm 3dnowext 3dnow constant_tsc
rep_good nopl nonstop_tsc extd_apicid pni monitor cx16 popcnt
lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse
3dnowprefetch osvw ibs skinit wdt npt lbrv svm_lock nrip_save
pausefilter
```

Simulating a von-Neumann Machine



We need simulation of

- Processor
- Memory
- Input and Output

realized in

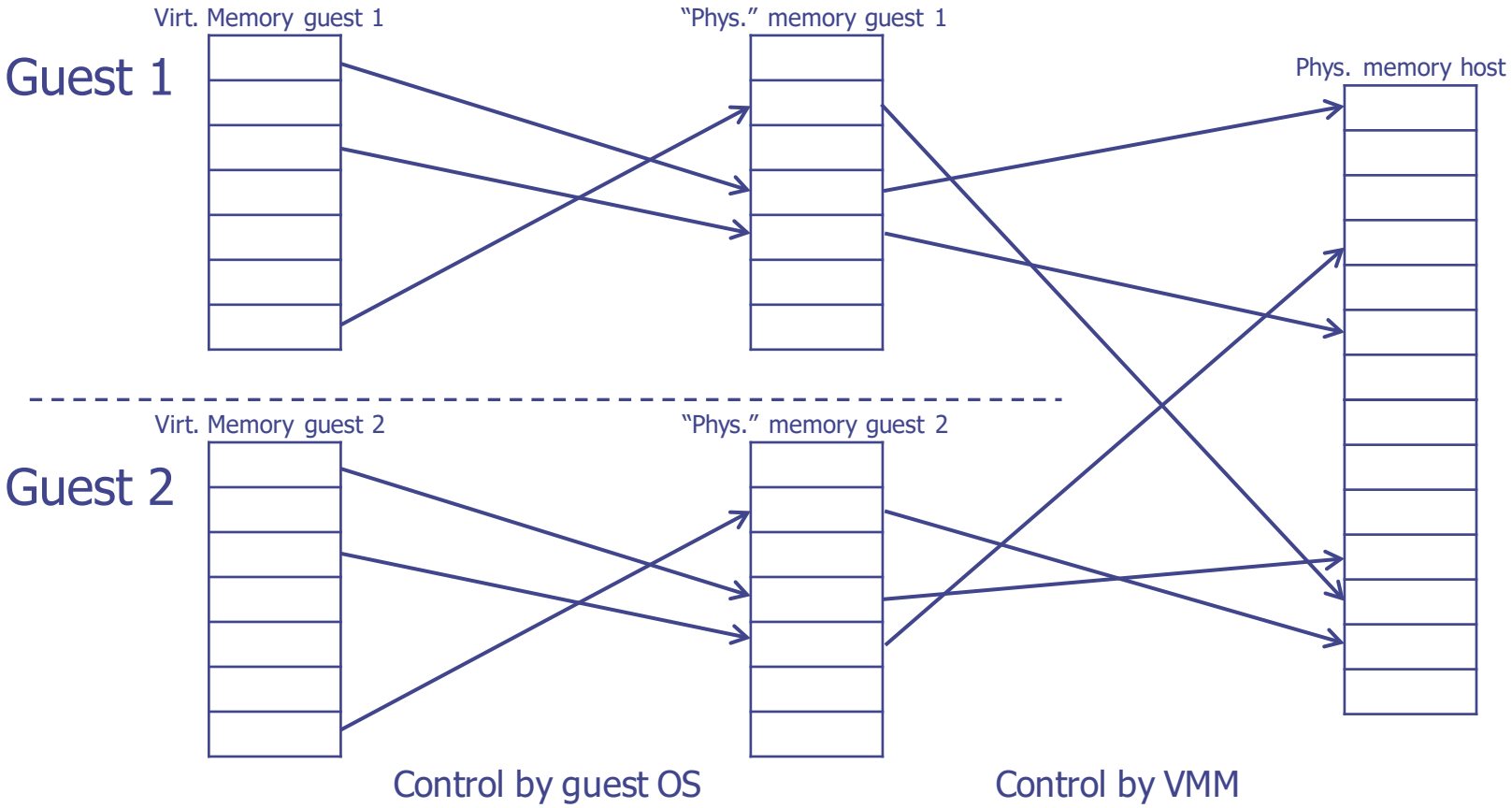
- Software
- Software with different levels of hardware support

Memory

- Memory of the VM is just a large memory region of the host.
  - Instructions of VM access this memory
  - MMU of host can be used for address mapping
- Problem: Virtual memory and memory protection inside the VM
  - Directly accessing the MMU by the VM would influence the host
  - Pure Software solution
    - MMU behavior is emulated for the VM
    - Guest page tables are data structure of VMM
    - VMM creates shadow page tables for host MMU
    - High overhead due VMM interventions
  - Hardware support: Second Level Address Translation (SLAT)
    - Separating memory management of VM and VMM in hardware
    - Guest can change its page tables without VMM intervention
    - Virtualization-aware TLB important for performance
    - Supported by the second generation of AMD-V and Intel VT-x, called rapid virtualization indexing (formerly nested pagetables) (AMD) or extended pagetables (Intel).

Second Level Address Translation

Who deals with page faults?



Examples: VT-x and AMD-V in Linux

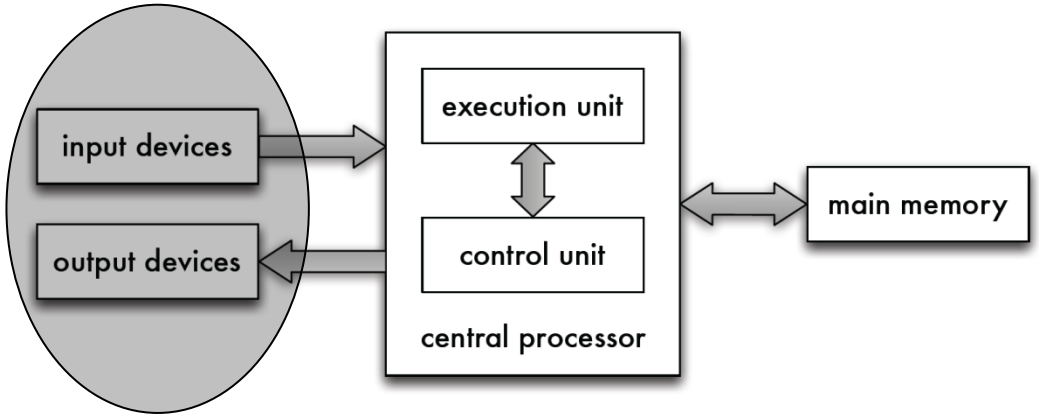
- Intel Xeon X3470

```
$ cat /proc/cpuinfo | grep flags | head -1
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss
ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc aperfmperf pni dtes64 monitor
ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 popcnt
lahf_lm ida dtherm tpr_shadow vnmi flexpriority ept vpid
```

- AMD Opteron 8435

```
$ cat /proc/cpuinfo | grep flags | head -1
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall
nx mmxext fxsr_opt pdpe1gb rdtscp lm 3dnowext 3dnow constant_tsc
rep_good nopl nonstop_tsc extd_apicid pni monitor cx16 popcnt
lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse
3dnowprefetch osvw ibs skinit wdt npt lbrv svm_lock nrip_save
pausefilter
```

Simulating a von-Neumann Machine



We need simulation of

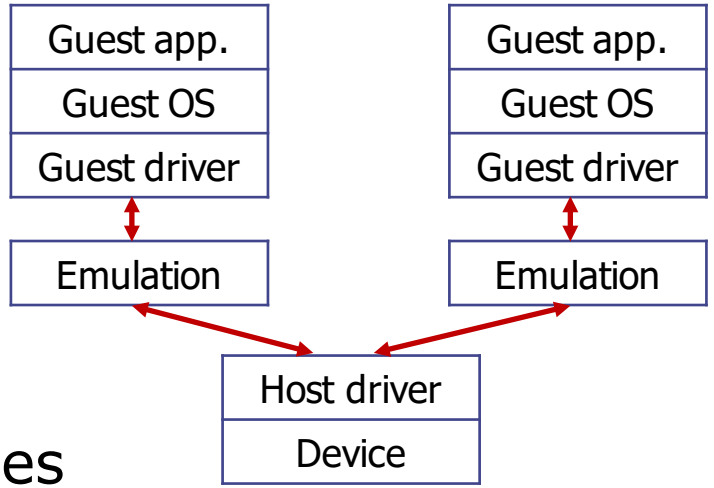
- Processor
- Memory
- Input and Output

realized in

- Software
- Software with different levels of hardware support

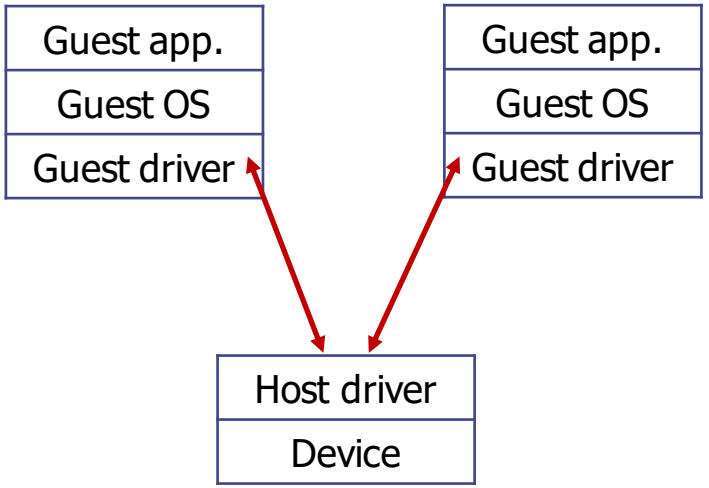
Input/Output: Emulation

- Devices are emulated with all functionality
  - Busses
  - Interrupt mechanisms
  - Memory mappings
  - …
- Allows arbitrary devices independent from physical host hardware
- Virtualization software maps accesses to real devices



Input/Output: Paravirtualization

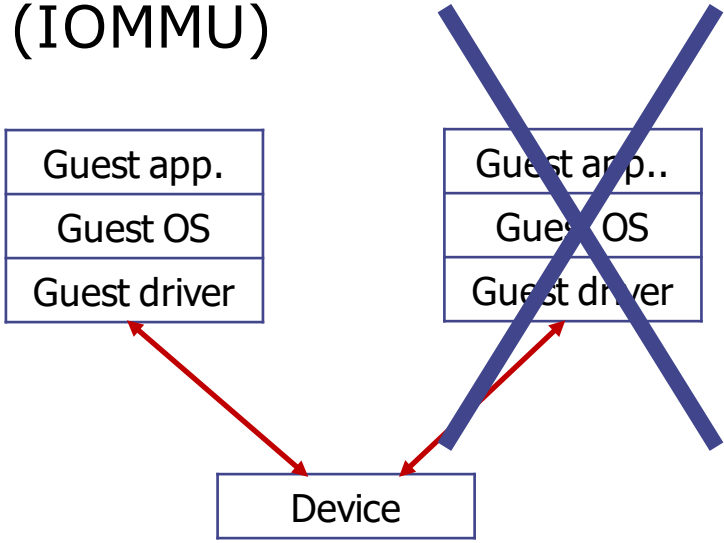
- Avoiding the overhead of simulation by saving "translations"
- VMM offers VM a special device
  - Maps I/O operations directly to host I/O
  - VM has special drivers for device, usually much simpler than driver for real hardware



- Most often used for
  - Network
  - Mass storage
  - Graphics
- Additionally special functions for interaction with VMM
  - Dynamic changes of memory size of VM

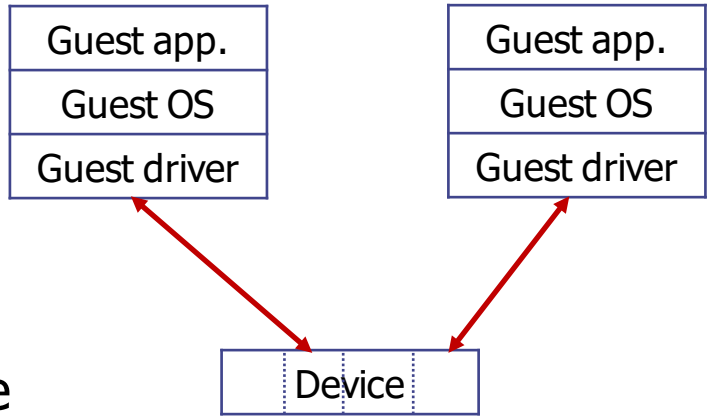
Input/Output: Access to Host Devices

- Devices of host are mapped exclusively into a VM
- Needs hardware support for mapping of memory locations and interrupts (IOMMU)
  - Intel: VT-d
  - AMD: AMD-Vi
- Native device drivers are used by VM
- Host should not access the device



Input/Output: Access to Virtualizable Devices of Host

- As before, but
  - Device is virtualizable and offers multiple instances that can be mapped into different VM
  - Guest accesses its instance by appropriate drivers
  - Cooperation of instances is managed by device, sometimes with help of driver at host



- Currently used for
  - Professional network cards (multiple HW queues are mapped into VM)
  - Graphic boards for GPGPU or desktop virtualization

Linux KVM, Guest View to PCI

Freie Universität Berlin

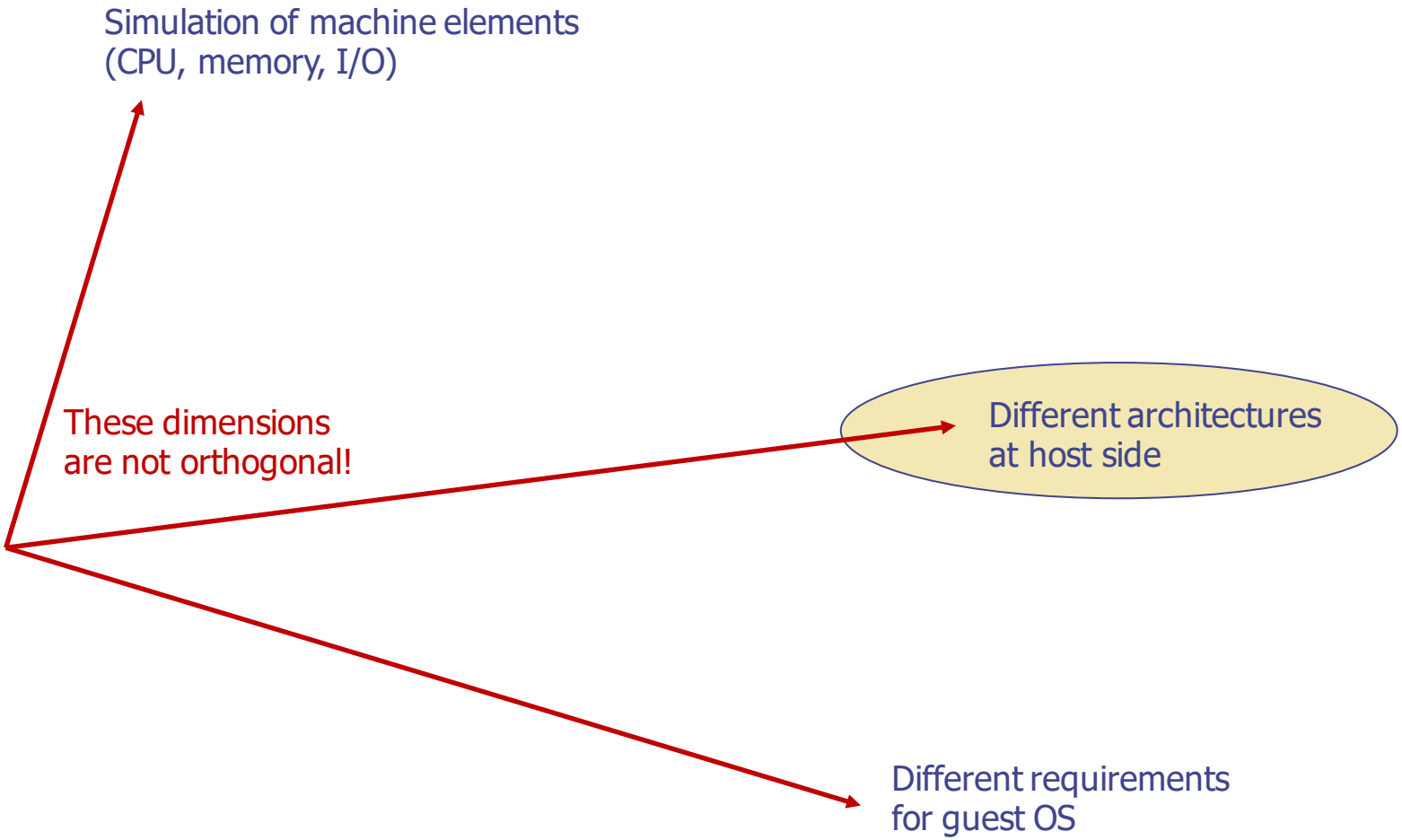
```
lspci
   00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
   00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
   00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
   00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
   00:02.0 VGA compatible controller: Cirrus Logic GD 5446
   00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet
   Controller (rev 03)
   00:04.0 Ethernet controller: Red Hat, Inc Virtio network device
   00:05.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
   00:06.0 SCSI storage controller: Red Hat, Inc Virtio block device

   emulated / paravirtualized / Interaction with VMM
```

VMware ESX, Guest View to PCI

Freie Universität Berlin

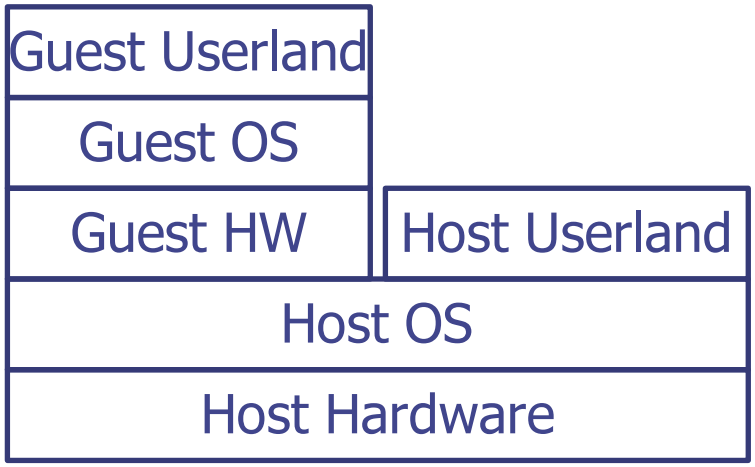
```
lspci
   00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge
   (rev 01)
   00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge
   (rev 01)
   00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
   00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
   00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
   00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev
   10)
   00:0f.0 VGA compatible controller: VMware SVGA II Adapter
   00:11.0 PCI bridge: VMware PCI bridge (rev 02)
   00:15.0 PCI bridge: VMware PCI Express Root Port (rev 01)
[…]
   02:01.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
   Controller (Copper)
   03:00.0 Serial Attached SCSI controller: VMware PVSCSI SCSI Controller (rev
   02)
   0b:00.0 Ethernet controller: VMware VMXNET3 Ethernet Controller (rev 01)

   emulated / paravirtualized / Interaction with VMM
```

Dimensions of Consideration



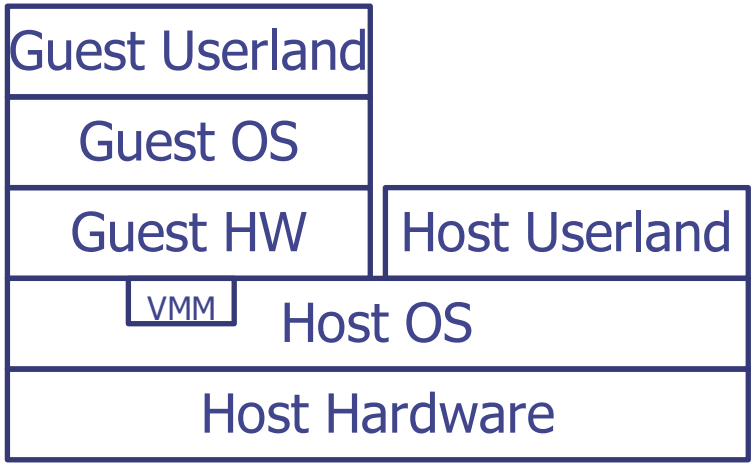
VM as Userland Process of Host OS

- No special kernel support
- Suitable for emulators
- Problematic for hardware-supported solutions because privileged mode requires OS intervention
- VM is subject to usual CPU scheduling
- Example: QEMU, Frodo, …



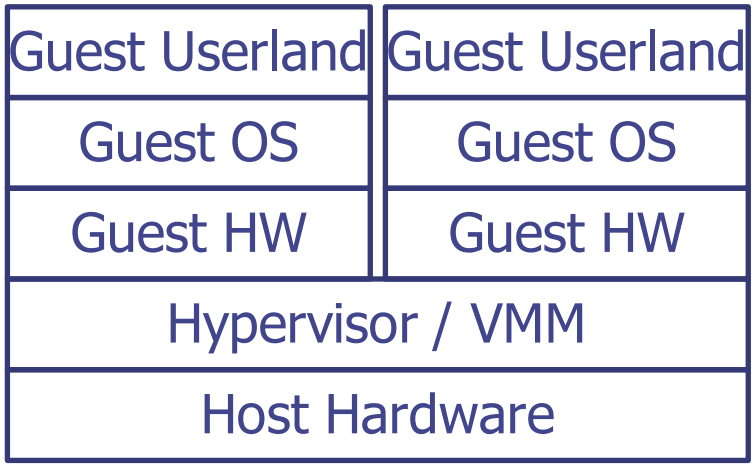
VM as Userland Process of Host OS with VMM in Kernel

- Management of VM inside kernel (integrated hypervisor)
- Kernel driver for privileged components
- Suitable for hardware-supported virtualization both without and with special instruction sets
  - Kernel driver for virtualization extensions in the latter case
- VM is subject to usual CPU scheduling
- Also called "Type 2 hypervisor"
- Example: Virtualbox, VMware Player and Workstation, KVM, …

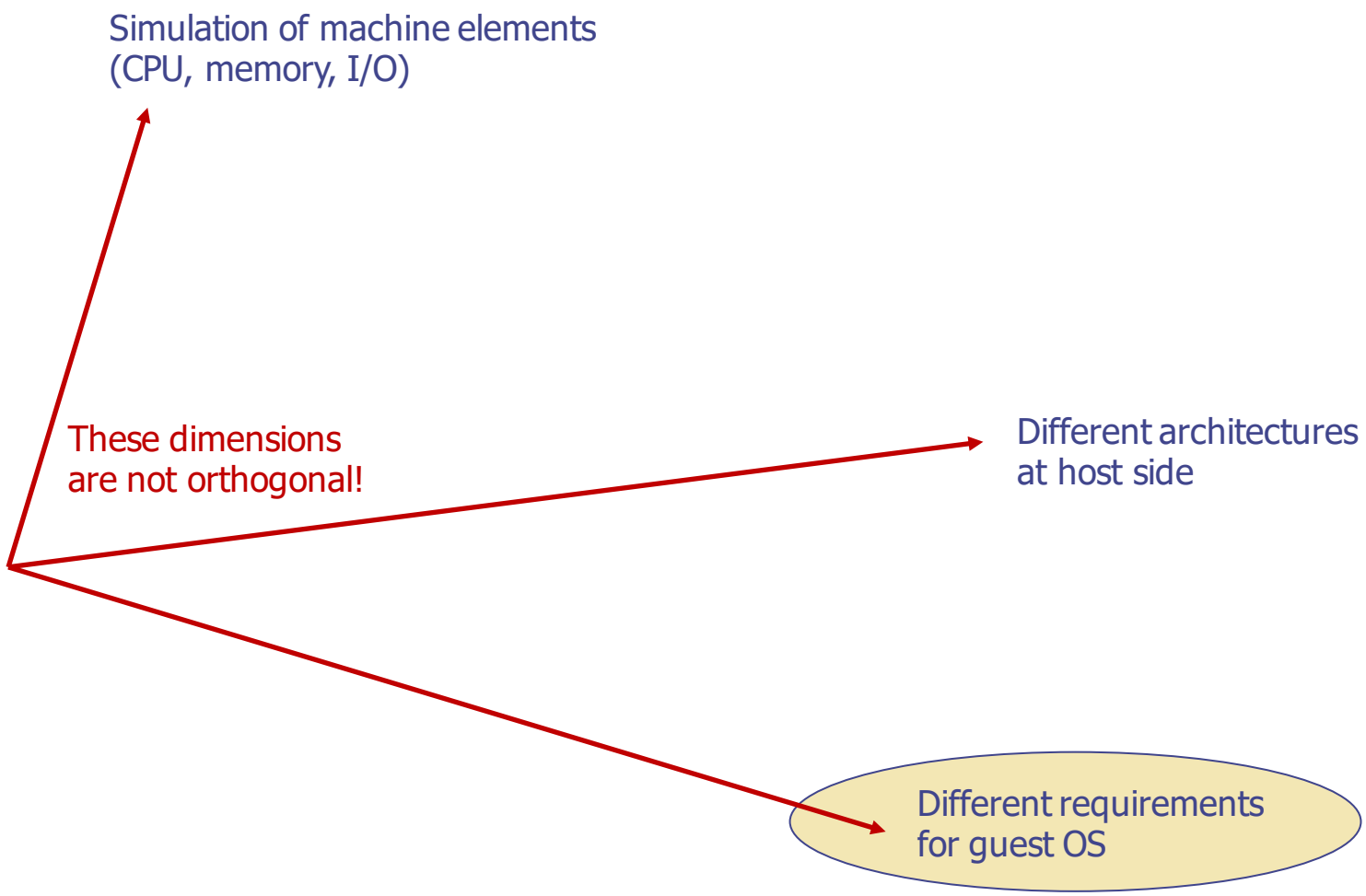


Dedicated OS only for Execution of VMs (hypervisor)

- Hypervisor runs natively on the machine (as a kind of OS)
- Virtualization drivers are integrated into hypervisor
- VM are scheduled by special scheduler that is aware of virtualization
- Also called "Type 1 hypervisor"
- Example: Xen, VMware ESX, Microsoft HyperV, …



Dimensions of Consideration



Unmodified Guest Operating Systems

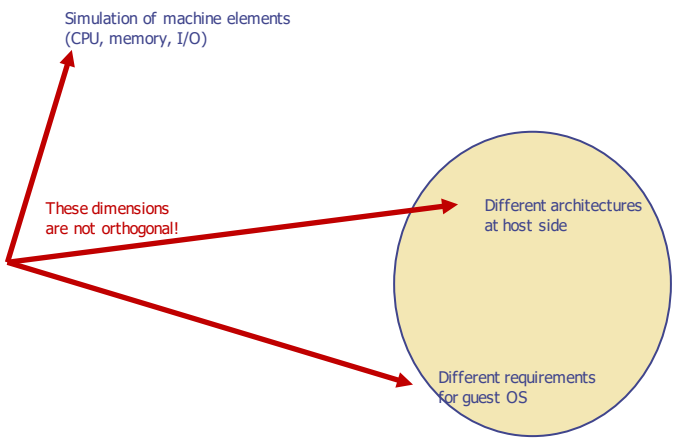
Freie Universität Berlin

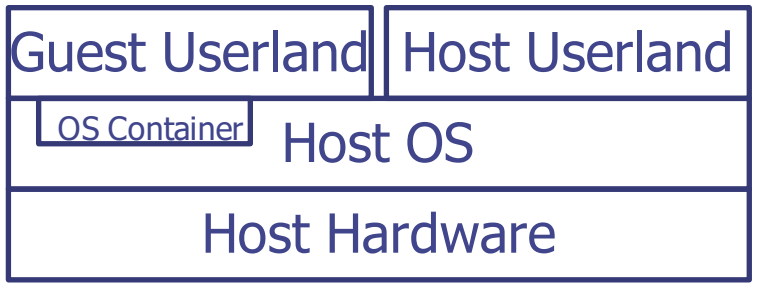
- If the virtualized hardware behaves the same way as real hardware (with exception of timing and available resources), an unmodified guest OS can be used.
  - Requires emulation of at least some I/O devices → low performance
  - Existing drivers can be used
  - Closed source systems run "out of the box"
- Also called "full virtualization" in case that a significant amount if instructions is not emulated
- Example: Nearly all virtualization products

Paravirtualization

- Alternative option: Guest OS is aware of virtual environment
  - No need for emulation because guest accesses real functionality in a more direct way (mapping to functions of host OS).
- Paravirtualized OS
  - Guest OS is adapted to be aware of virtualization.
  - Code that uses problematic instructions is replaced by code directly accessing the appropriate hypervisor functionality.
  - Guest OS is no longer build for the original processor but specifically for the appropriate hypervisor.
  - Example: Xen, KVM (both optional)
- Paravirtualized I/O → slide 26 on input/output
  - Paravirtualized device drivers can also be used in an unmodified guest OS

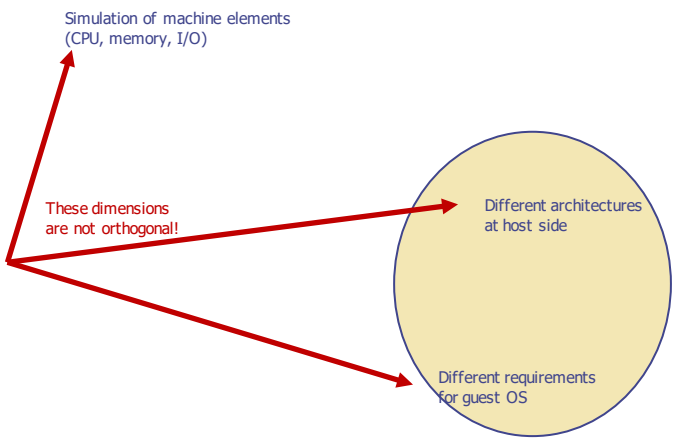
Containers

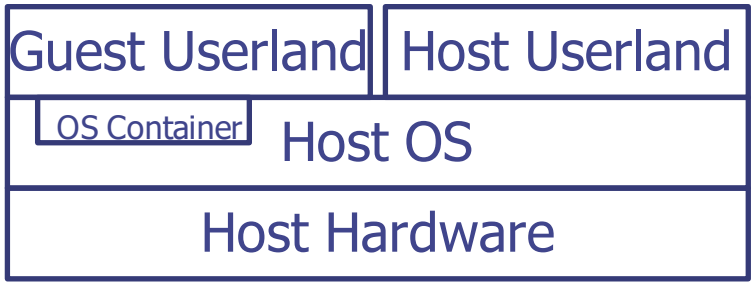




- Strictly seen, no "real" virtualization
- Implementation inside host kernel to provide necessary functionality, esp. separation.
- All processes of all guests are visible to host OS.
- Host kernel is used for host and all guests.
  → only same OS possible
- Guest only has own userland
  → e.g., different Linux distributions
- Separation is done inside host OS by creating isolated sets of processes
  - Behave similar to the userland of a real OS.

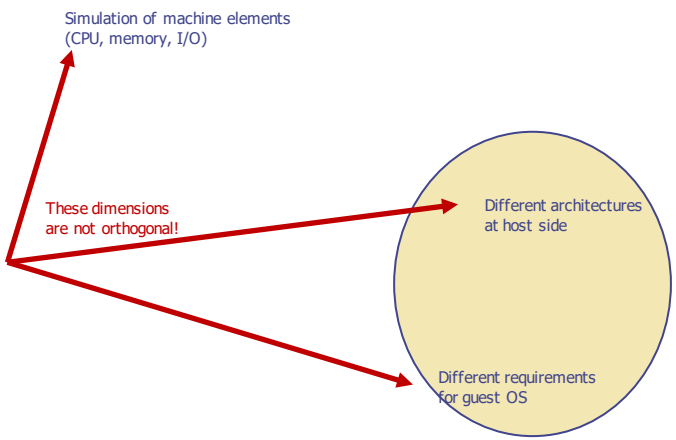
Containers

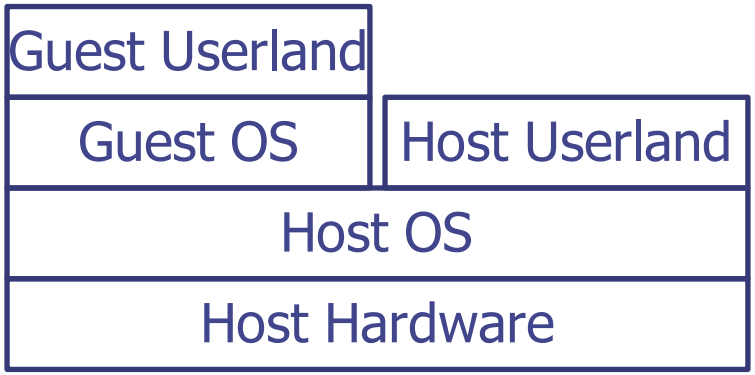




- Resources belong to the host, usually limited for individual guests.
  - Number of processes, memory, CPU shares for scheduling
- Mapping of user IDs to enable root inside guest.
- I/O is realized by the host kernel by offering virtualized view to real I/O.
  - Filesystem may be a directory of host filesystem or just an image formatted with some filesystem.
  - Network access by simple devices transmitting data to the host kernel (Linux: e.g., tun/tap infrastructure).
- Example: Solaris zones, Linux containers, OpenVZ, …

User Mode Virtualization





- Strictly seen, no "real" virtualization
- Combination of concepts from paravirtualization and container
- Guest OS…
  - does not run on emulated hardware or special paravirtualized devices
  - runs directly as process on top of host OS.
  - is therefore adapted to use the API of the host OS instead of accesses to (real or emulated) hardware.
  - is visible as one process to host OS.
- Example: User mode linux

Live Migration

- In larger installations with more than one host, migration of VMs is a useful option for many reasons.
  - Load balancing
  - Maintenance
  - Adaption to changes in requirements (e.g., faster host)
  - Energy-awareness (only necessary hosts are powered, dynamic adaption by starting/stopping hosts)
- Assumptions
  - Hosts are sufficiently equal
  - Hosts are both up and connected by fast network
- Basic idea
  - Copy VM memory from source to target iteratively (watching changes) while VM is running
  - Stop VM on source host, copy last changes and state
  - Start VM on target host

Live Migration

- Problems
  - Access to devices has to be possible from both hosts
    - Network storage, same network for VM on both hosts, …
  - Direct access to host devices for VM
    - Special handling or using emulated devices supporting migration
- Many optimizations are possible
- "Snapshots" are also interesting for other purposes (backup, experiments, …)

Further Aspects

- Multiprocessor VMs
  - Basic principle can also be applied to multiple virtual CPU (vCPU)
  - Problems
    - Violates assumption that a CPU is always present
    - Flexible VM vs. detailed knowledge on execution environment
    - …
  - Solutions
    - Improved hardware support
    - Adapted scheduling of vCPU by VMM
- Virtual networks
  - Virtual network adapters of VM are networked to virtual infrastructures
  - Connection to physical networks

Further Aspects

- Recursive Virtualization
  - Hypervisor is guest of another hypervisor
  - Challenges are identical to those of single level virtualization
  - Hardware support usually only supports one layer → Emulation necessary
- Cloud computing
  - "Infrastructure as a service" (IaaS)
  - VM as container for applications that can easily be handled
  - Resources can be adapted at any time
- Management and Orchestration
  - Container
    - Docker
    - Kubernetes