Virtualization (Recap)

- Assume basic virtualization support (Intel VT-x, AMD SVM, ARM VE)

⇒ All sensitive instructions either handled internally (i.e. modify virtual state instead of physical state) or cause a trap into HV

- MMU with nested page table support
  - First translations $GV \rightarrow GP$
  - Nested translation $GP \rightarrow P$

- But what about devices?

- *Usually* there is still only one graphics/network/.. card
Special case: Interrupt controller

- Interrupt controller signals to system software pending interrupts
- Device X triggers an IRQ → Trap into the hypervisor
- What now?
Special case: Interrupt controller

(Solution I: Paravirt):
- Set EIP/PC to exception vector in the guest
- Emulate IRQ controller interface and set interrupt pending
- Drawbacks: Many hypervisor traps → bad

Solution II: Virtualization Extensions
- Intel: Notification to guest via VMCB
  V_IRQ – If nonzero, virtual INTR is pending
- ARM: vGIC Virtual CPU interface
- Hypervisor can set interrupts pending for guests
- On next VM execution, hardware delivers IRQ
- Drawbacks: IRQ initially still delivered to hypervisor

Solution III: IRQ Controller with Virtualization support → Next Slide
Interrupt Controller with Virtualization support

- Intel: APICv / AMD: AVIC / ARM: -
- Hypervisor can configure interrupts to be delivered directly to guests
- Interesting for...
  - Inter-processor interrupts
  - Interrupts from assigned (pass-through) devices
Device Virtualization: Trap-and-emulate

- When guest tries to access the device pages → trap into hypervisor
- Hypervisor emulates the device
- Advantages:
  - Good solution for simple devices (e.g. UART)
- Drawbacks:
  - Complex logic (device emulator, driver, etc.) inside the hypervisor
  - Context switches take longer due to state saving
  - Might be impractical for complex devices (e.g. graphics card)
Device Virtualization: Driver VM

- One VM gets all devices, provides functionality to other VMs
- Advantages:
  - Easy to implement
  - Compatibility (Linux drivers are already there...)
- Drawbacks:
  - Driver VM becomes part of the TCB
  - Slow due to many context switches into and out of the driver VM
Device Virtualization: Hardware support

- Device supports holding the state for multiple VMs
- Usually only fancy server network cards can do this
- New PowerVR Series7XT/XE GPU can do it too

**Drawbacks:**
- Driver software might get complex
Quote: “Although you usually tend to hear people talking about virtualization in the context of CPUs, it is actually a system-level requirement that can only be implemented optimally if all the components in the chip support it.” — Imagination Technologies
Some devices can do DMA (Direct Memory Access)

Of course DMA controller
- Can be configured to do DMA for some devices
- Control register selects device to do DMA for (e.g. USB, UART, etc.)

But also other powerful devices, e.g. graphics card
- Graphics card has its own MMU (can potentially read/write entire main memory)
- No restriction through the CPU’s MMU
- Might be very dangerous in the wrong hands (Bug existed on Samsung Galaxy S2 and S3)
- Allowed for a privilege escalation through the GPU (gain root on Android) [1]

Problem even more delicate with virtualization!
If DMA capable device is set to pass-through to one VM it can write to other VMs (or even VMMs) address space

Attacks were proposed this way [2] to take over the Xen HV

Solution: No pass-through of powerful devices! (...or?)
Device Virtualization: IOMMU/SYSMMU

- Some x86 processors have an IOMMU (Intel VT-d or AMD AMD-Vi)
  - Matches devices on the PCI bus
  - Can be configured to allow individual PCI devices to only access parts of the main memory
- Some highend ARM boards have a SYSMMU in front of powerful BUS master devices
J. Danisevskis, M. Piekarska, and J.-P. Seifert.
Dark side of the shader: Mobile gpu-aided malware delivery.

J. Rutkowska and A. Tereshkin.
Bluepilling the xen hypervisor.