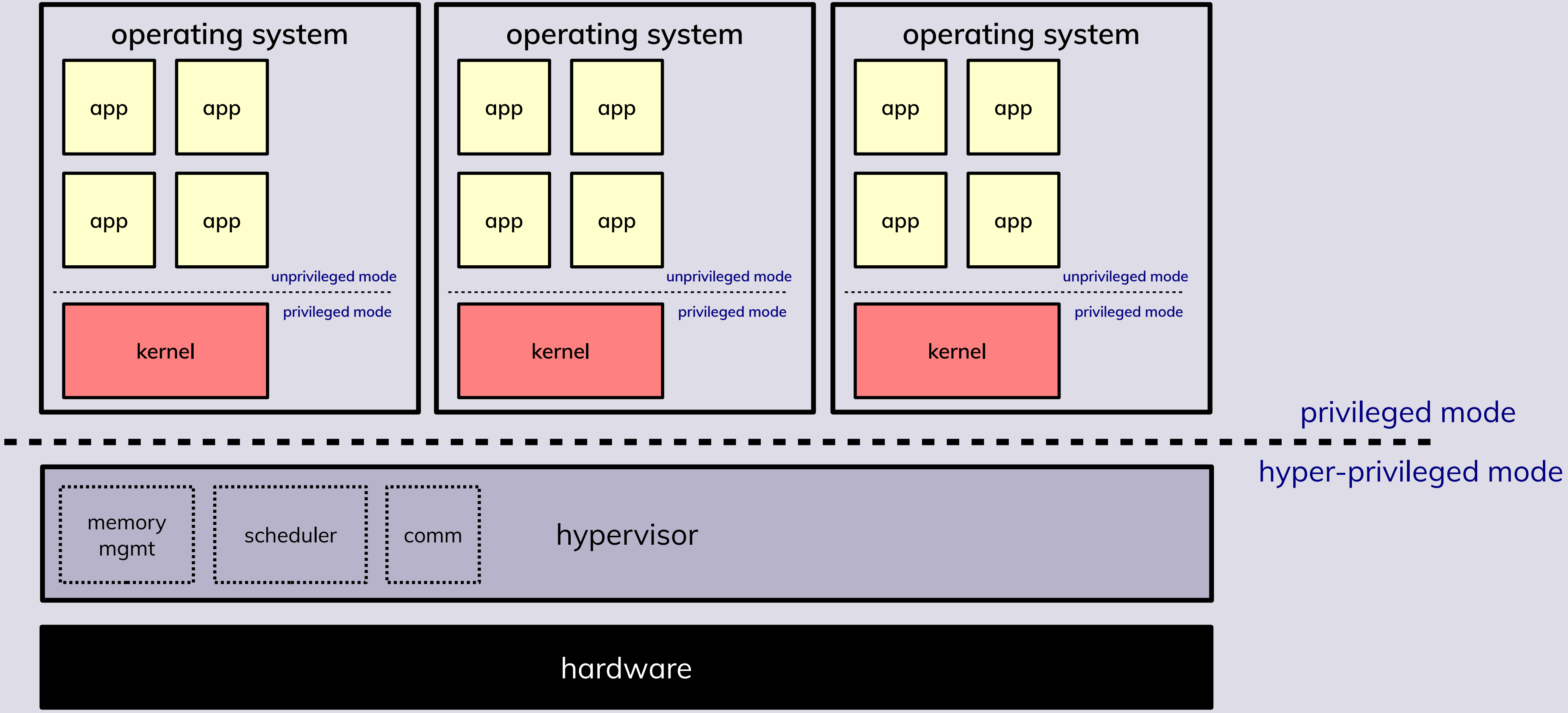
Advanced Operating Systems
Summer Semester 2023/2024

Martin Děcký

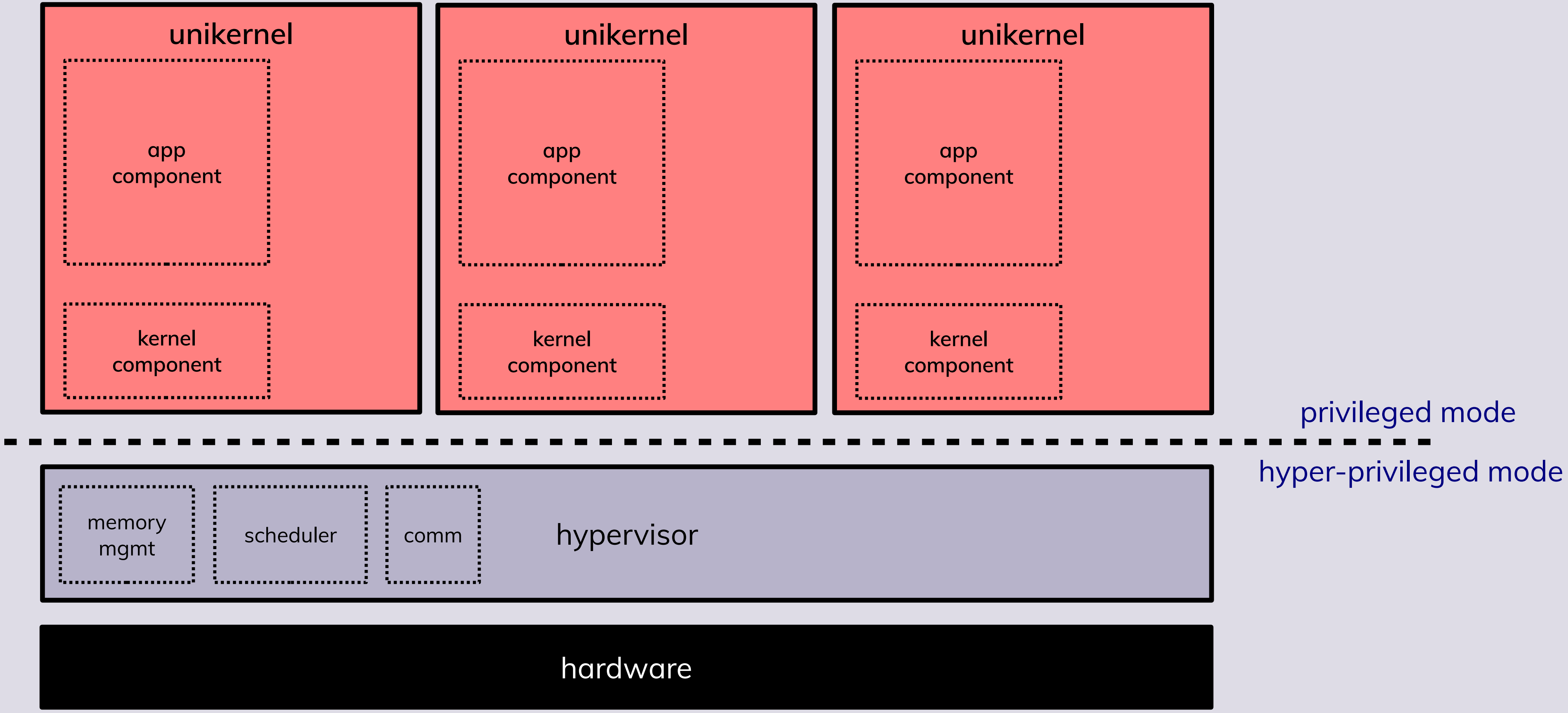
4

Virtualization

Hypervisor (Type 1)



Hypervisor (Type 1) with Unikernels



Effective Virtualization

- **Popek/Goldberg conditions on instruction set effective virtualization**
  - Assumes instruction sets with a privileged (kernel) and a non-privileged (user) mode
  - Definitions
    - ***Virtualizable*** instructions
      - Instructions that always trap when executed in non-privileged mode
    - ***State-altering*** instructions
    - ***State-affected*** instructions
  - ***Instruction set is virtualizable if every state-altering and state-affected instruction is also a virtualizable instruction***
    - Example: Classical IA-32 contains several ***critical*** instructions that do not meet this condition
      - **`SGDT`**, **`SIDT`**, **`SLDT`**, **`POPF`**, **`PUSHF`**, **`POP`**, **`PUSH`**, **`MOV`**, **`CALL`**, **`JMP`**, **`INT`**, **`RET`**

Virtualization without Effective Virtualization

- **Non-transparent virtualization**
  - Partitioning
    - "Shared kernel virtualization", "namespaces", "containers", "zones", etc.
      - Logical separation of user space tasks into isolated groups
      - No true VM abstractions
        - Traditional OS abstractions with additional layer of resource management and object visibility
  - Paravirtualization
    - Voluntary cooperation between VM and hypervisor
      - VM replaces ***state-altering*** instructions with hypercalls and adapts the output of ***state-affected*** instructions
      - Also usable as a performance improvement (e.g. I/O) for transparent virtualization

Virtualization without Effective Virtualization

- **Transparent virtualization**
  - Emulation
  - Dynamic translation
    - More efficient emulation that tries to separate ***critical*** and ***non-critical*** instructions
      - Whenever a code page is altered, ***critical*** instructions are replaced by explicit traps
      - VM usually provided with a read-only shadow copy to maintain integrity
      - Complicated by the fact that many non-effectively virtualizable instruction sets also do not provide other efficient features (e.g. non-executable pages)

Virtualization without Effective Virtualization

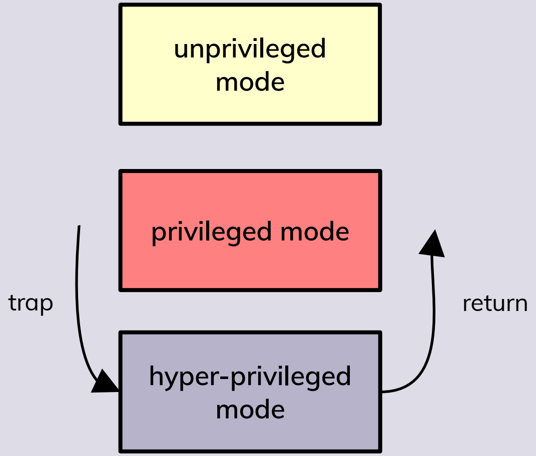
- **Transparent virtualization**
  - Special hardware privileged mode
    - Turning ***critical*** instructions into ***virtualizable*** instructions
    - Usually somewhat limited in scope (e.g. V86 on IA-32)
  - Hyper-privileged (hypervisor) mode
    - Mode that affects the behavior of the privileged mode (which is, in essence, not fully privileged)
      - Usually associated with an analogous set of control registers as the privileged mode
      - Instructions that might be ***critical*** w.r.t. non-privileged mode are ***virtualizable*** using the hyper-privileged mode
    - PL2 (ARM), EL2/EL3 (ARM64), M-mode (RISC-V)

Virtualization without Effective Virtualization

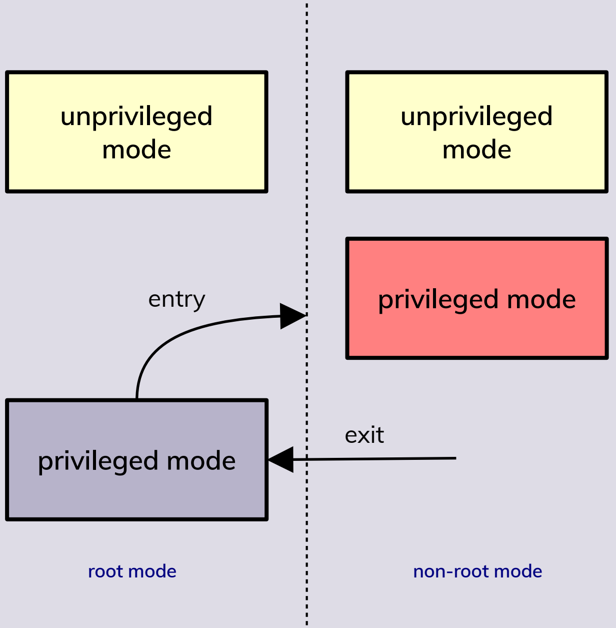
- **Transparent virtualization**
  - Orthogonal virtualization modes
    - Separate control registers (control structures) and control instructions
      - Nested virtualization possible if the control instructions are self ***non-critical***
  - No traditional traps, but VM exits (and VM entries)
  - Intel VT-x (VMX), AMD AMD-V (SVM)
    - Root mode (hypervisor)
    - Non-root mode (guest VM)
  - Hypervisor Extension (RISC-V)
    - HS-mode (hypervisor-extended supervisor mode)
    - VU-mode (virtual user mode), VS-mode (virtual supervisor mode)

Transparent Virtualization

- **Hyper-privileged mode**



- **Orthogonal modes**



Virtualization without Effective Virtualization

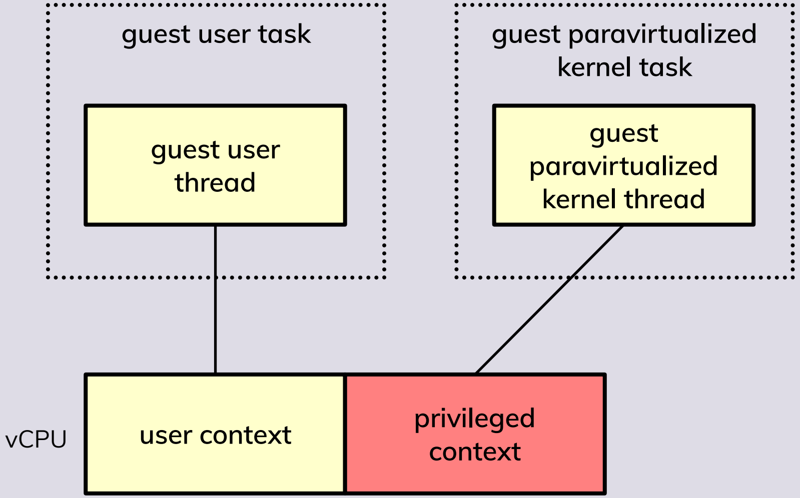
- **Side note: x86 CPU protection levels (rings)**
  - Compared to most other ISAs (except VAX, IA-64, MIPS*), there are 4 privilege levels
    - CPL 0 (kernel mode), CPL 1, CPL 2, CPL 3 (user mode)
    - Affects segmentation and I/O instructions, not paging (CPL 1 and 2 are privileged with respect to paging)
  - Legacy VMware and VirtualBox using dynamic translation executed the guest OS code in CPL 1
    - Harder to (accidentally) break the dynamic translation mechanism (via interrupt handling, etc.)
    - Easier to keep the actual user code in CPL 3
    - Entering CPL 1 instead of CPL 0 (and using different segments) is not transparent
      - Examining the CPL is a ***critical*** operation
  - Xen executed paravirtualized guests in CPL 1
  - OS/2 and VMS executed device drivers in CPL 2
    - Isolation both from the kernel and from the user space
    - Potentially challenging for virtualization

Operating System Virtualization Abstraction

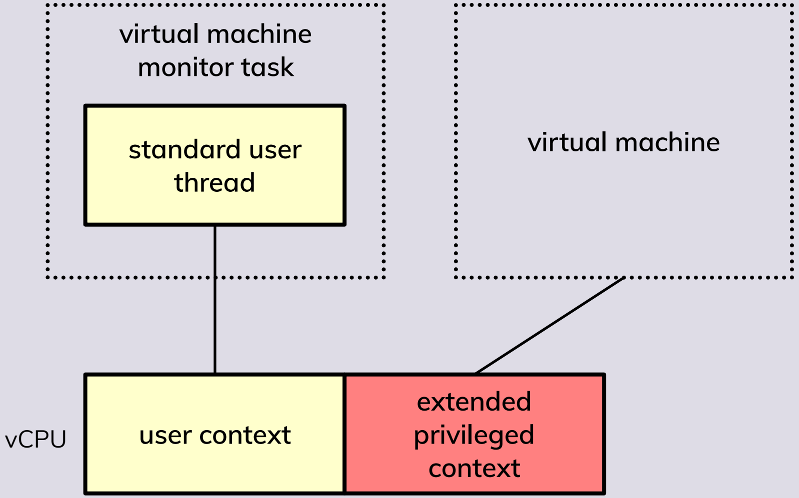
- **vCPU (virtual CPU)**
  - Logical extension of the (user) thread abstraction
    - Entity that keeps the computational context state
    - Besides the usual user context, it also tracks the privileged context
      - Paravirtualization
        - User context: Guest user thread running inside the VM
          - Exceptions, page faults, IRQs, IPC, etc., switch to the privileged context
        - Privileged context: Guest paravirtualized kernel (running in a different address space) that provides the environment for the guest user threads in the VM (including thread scheduling, etc.)
      - Transparent virtualization
        - User context: Virtual machine monitor (VMM, running in a different address space)
        - Privileged context: Context of the entire guest VM
          - Regular exceptions (including standard page faults) handled internally
          - IRQs, some state-altering instructions and other conditions switch to the user context (VM exit)

vCPU

- **Paravirtualization**



- **Transparent virtualization**



Intel VT-x (VMX)

- **Crucial instructions**
  - `VMXON` / `VMXOFF`
    - Enter / exit root mode
    - 4 KiB physical location for virtualization bookkeeping (opaque)
  - `VMPTRLD`
    - Load a Virtual-Machine Control Structure (VMCS) as current
      - 4 KiB physical location that stores the vCPU privileged context
        - Mostly opaque, fields accessed strictly via the **`VMREAD`** / **`VMWRITE`** instructions
        - *Control* fields (affecting the features / behavior of the virtualization, events that trigger VM exits, nested paging configuration, etc.)
        - *Guest* fields (context of the guest VM, i.e. privileged context of the vCPU)
          - **`RSP`**, **`RIP`**, **`RFLAGS`**, selectors, control registers, MSRs, interrupt/activity state
          - Does not store most of the GPRs
        - *Host* fields (context of the VMM, i.e. user context of the vCPU)
          - Analogy of the guests fields (for efficiently switching to the VMM)
        - *Read-only* fields (information about the VM exit)
          - VM exit reason, interruption (IDT vectoring) state, guest-physical address of a nested page fault, I/O instruction information, etc.

Intel VT-x (VMX)

- **Crucial instructions**
  - `VMLAUNCH` / `VMRESUME`
    - Launch / resume the current VMCS (i.e. execute a VM entry)
      - If there is no error on the VM entry, the instruction eventually transfers to the host state of the VMCS when a VM exit occurs
  - `INVEPT` / `INVVPID`
    - Invalidate the TLB for the nested paging based on the Extended Page Table root pointer or on the vCPU ID
  - `VMCALL`
    - Hypercall to the VMM
  - `VMFUNC`
    - Possible hardware acceleration of certain VMM operations (without a VM exit)
      - Currently only the Extended Page Table root pointer switching (among preset list of possible values)
      - Can be used to implement efficient hardware-assisted address space switching for IPC [1]

Intel VT-x (VMX)

- **Crucial VM exits**
  - Exception or NMI
  - External interrupt
  - Triple fault / INIT signal (i.e. reset) / start-up IPI
  - SMI events
  - Interrupt / NMI window (VM is in a state where it can handle the event)
  - Task switch, control register access, debug register access, `CPUID`, `RDMSR`, `WRMSR`, `GETSEC`, `HLT`, `INVD`, `INVLPG`, `MWAIT`, `MONITOR`, `PAUSE`, `XSETBV`, `XSAVES`, `XRSTORS`, `PCONFIG`, etc.
  - I/O instruction
  - APIC access
  - EPT violation
  - `VMCALL` (i.e. hypercall)
  - VMX instruction (i.e. nested virtualization)
  - Preemption timer
  - Page-modification log full

References

[1] Mi Z., Li D., Yang Z., Wang X., Chen H.: *SkyBridge: Fast and Secure Inter-Process Communication for Microkernels*, in Proceedings of the 14th EuroSys Conference, ACM, 2019, https://dl.acm.org/doi/10.1145/3302424.3303946

Thank you!

Questions?