UNSW SYDNEY | Australia's Global University

School of Computer Science & Engineering

COMP3891/9283 Extended Operating Systems

2025 T2 Week 08

Virtual Machines

Gernot Heiser

UNSW SYDNEY

Copyright Notice

Learning Outcomes

- An appreciation that the abstract interface to the system can be at different levels.
  - Virtual machine monitors (VMMs) provide a low-level interface
- An understanding of trap and emulate
- Understanding the difference between Type-1 (native) and Type-2 VMMs (hosted)

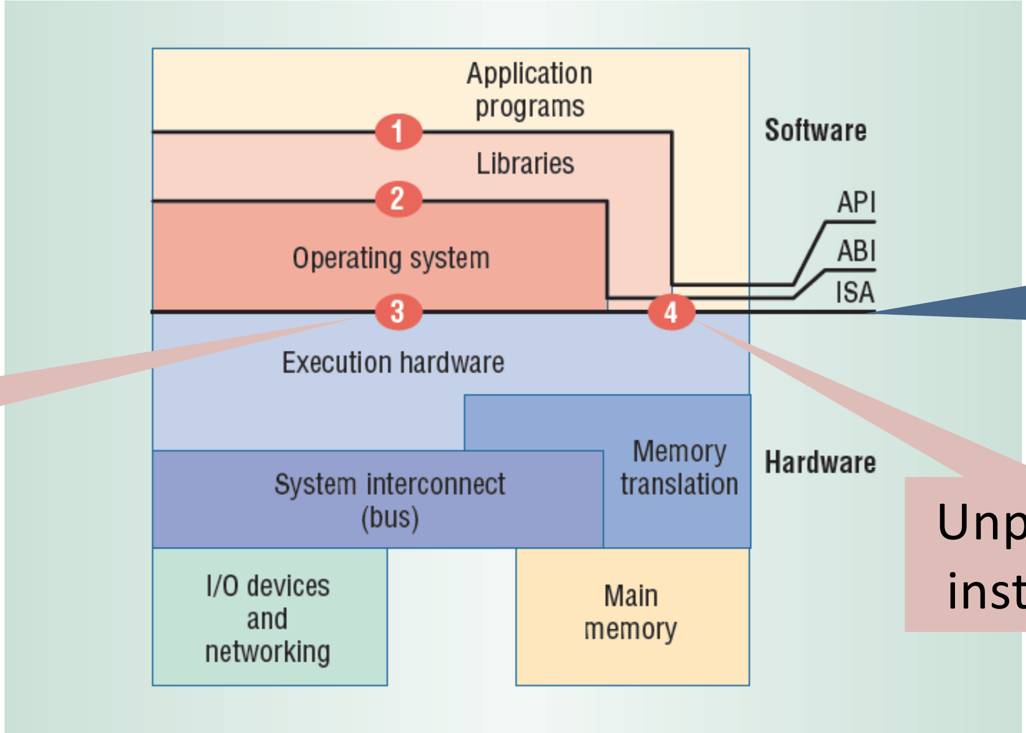
Virtual Machines: References

- Short version: Smith, J.E.; Ravi Nair, "The architecture of virtual machines," Computer, **38**(5), pp. 32- 38, May 2005
- Longer version: Textbook “Modern Operating Systems”, 5th ed, Ch 7–7.3
- If you’re keen: Rest of chapter 7.

Observations on Interfaces

- Operating systems provide well defined interfaces
  - Abstract hardware details
    - Simplify
    - Enable portability across hardware differences
- Hardware instruction set architectures are another well defined interface
  - Example AMD and Intel both implement (mostly) the same ISA
  - Same software can run on both

Instruction Set Architecture (ISA)



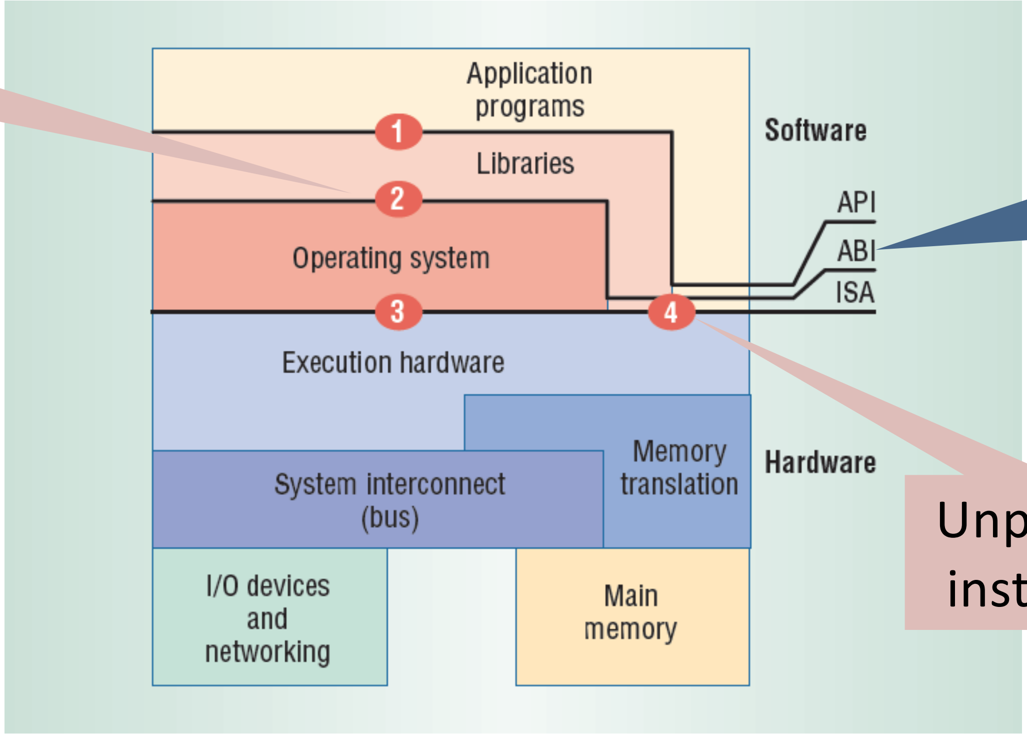
Privileged instructions

ISA= HW-SW interface

Unprivileged instructions

Portability across processors sharing ISA

Application Binary Interface (ABI)



OS system calls

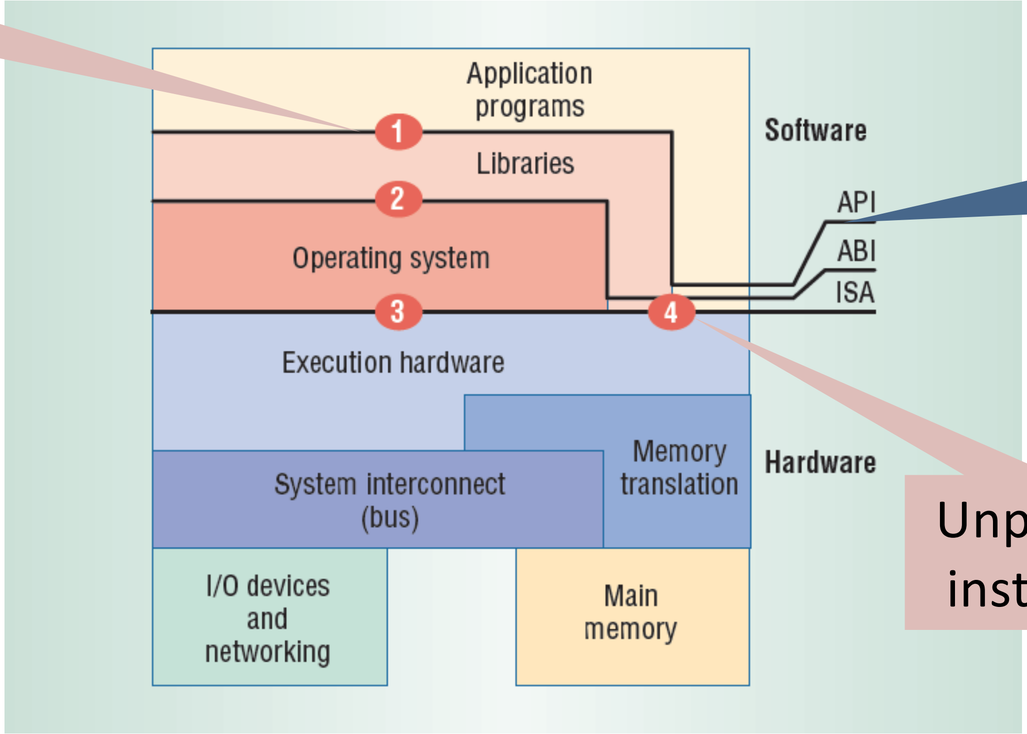
ABI: Interface seen by applications

Unprivileged instructions

Privilege boundary, Portability within ISA

Application Programming Interface (API)

Library calls (incl syscalls)



API: Source-level Interface

Unprivileged instructions

Abstracts ABI, source-code portability across ISAs

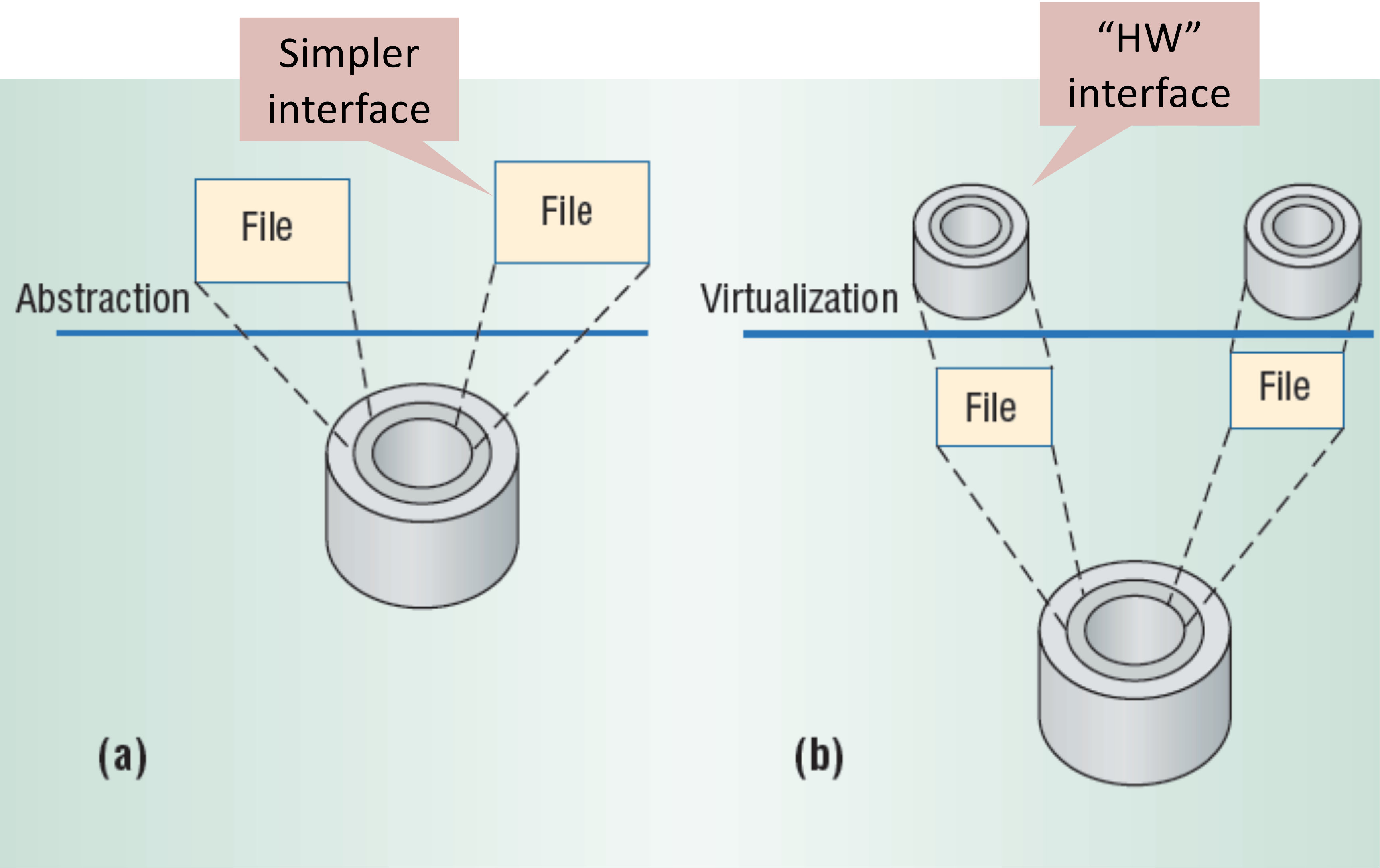
Interface Goals

- Portability of software across all computing platforms
- Secure sharing of hardware resources.
  - E.g. cloud computing

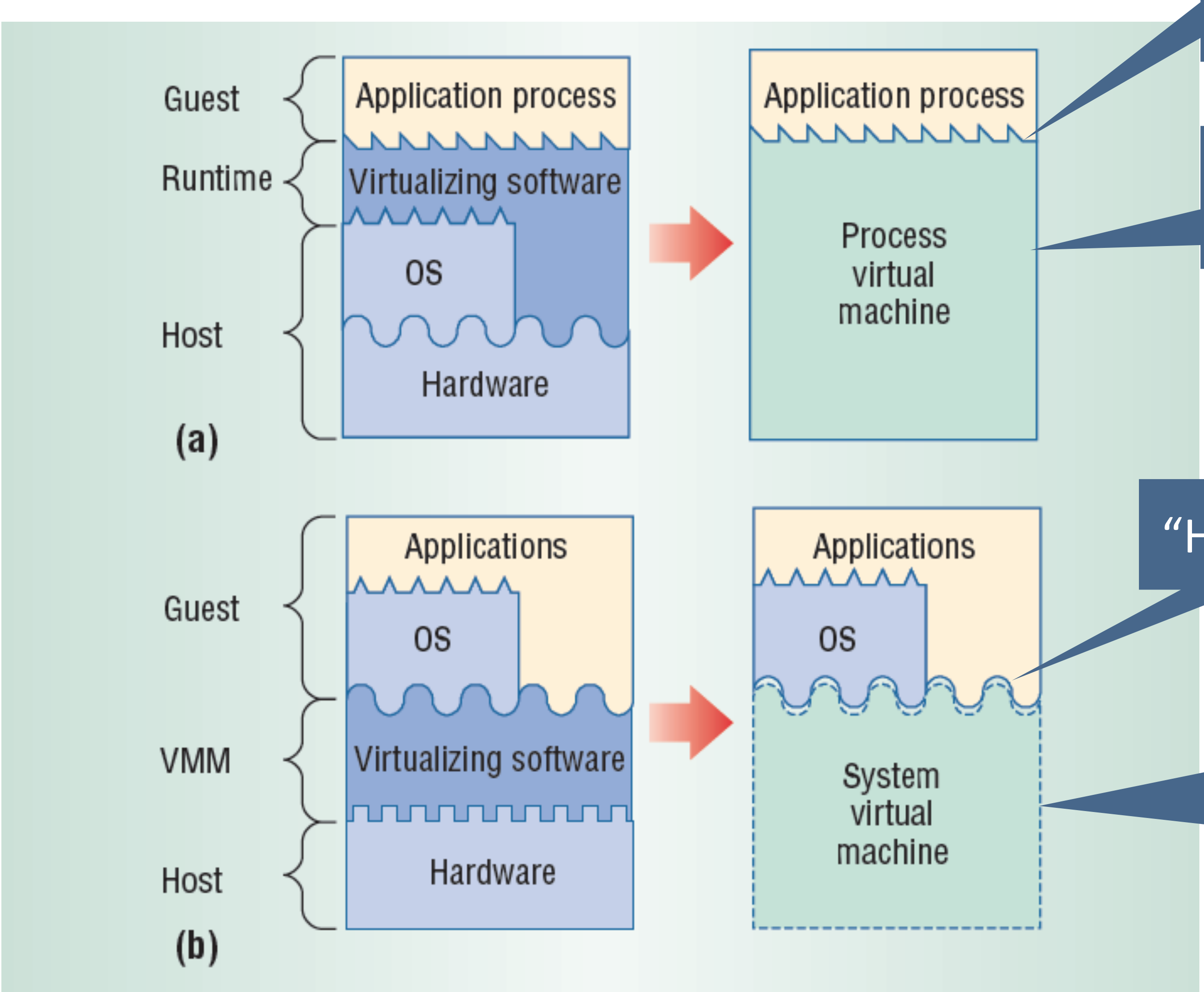
OS as a Virtual Machine

- Multiplexes the physical machine between applications
  - Time sharing, multitasking, batching
- … with a changed (more high-level) interface for
  - Ease of use
  - Portability
  - Efficiency
  - Security
  - Etc….

Abstraction versus Virtualisation



Process versus *System* Virtual Machine



High-level API

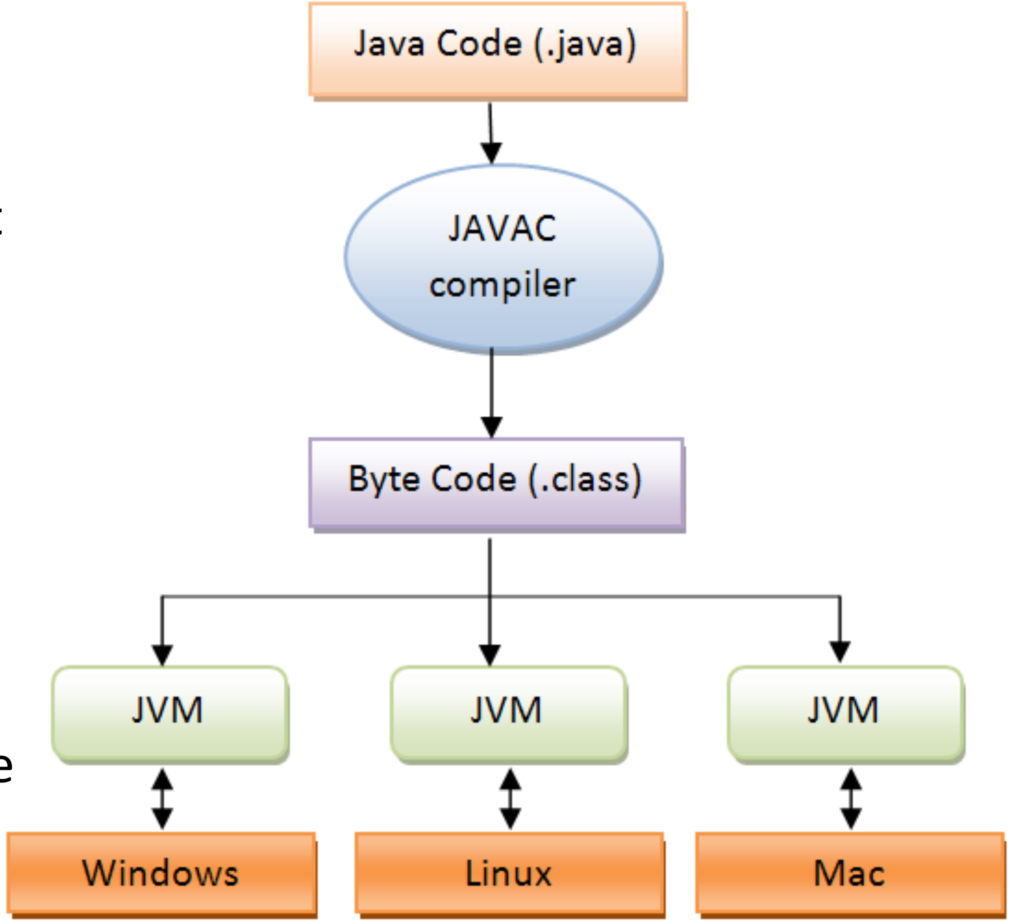
Lifetime is process

“HW” API

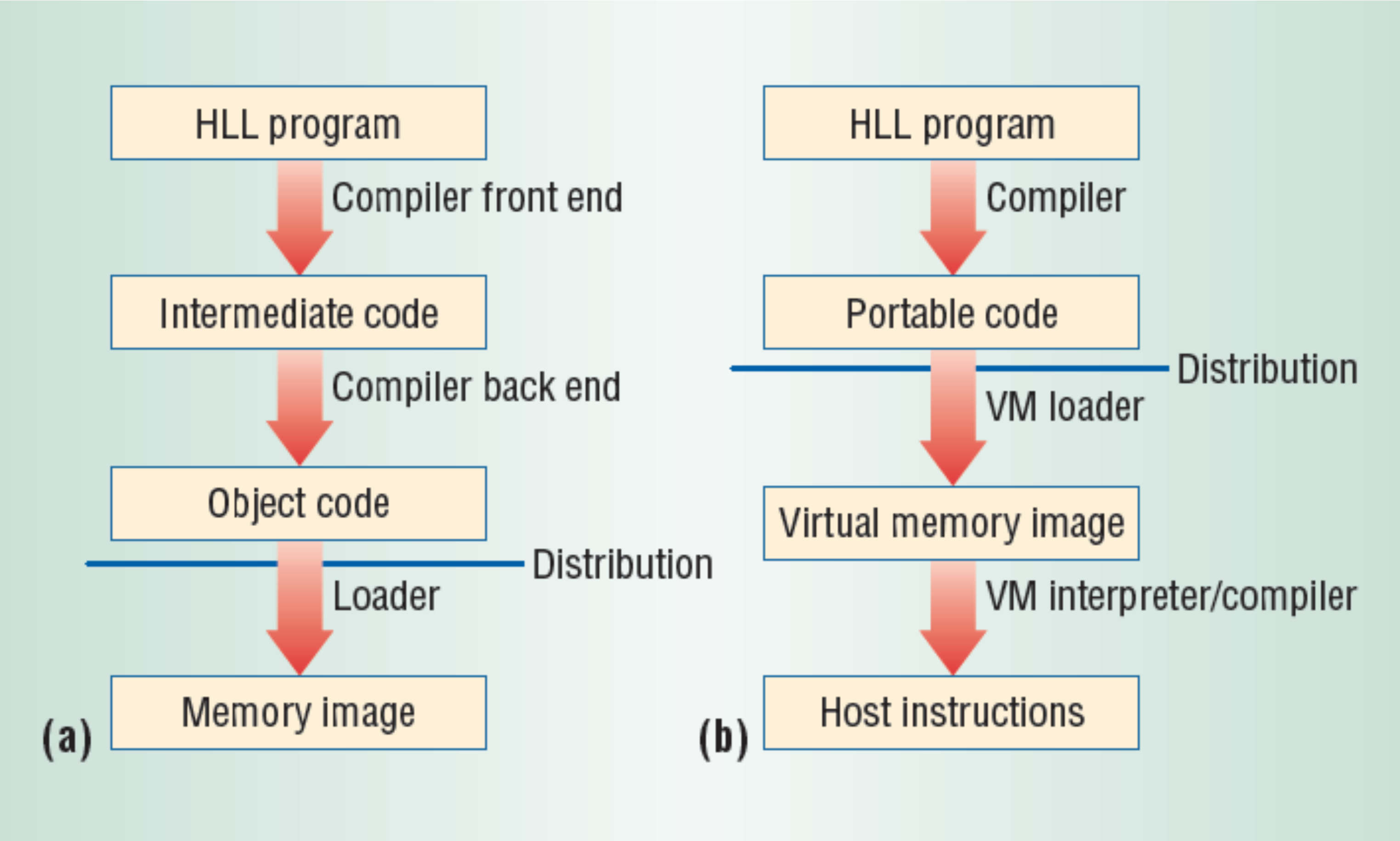
Lifetime exceeds apps

JAVA – Process Virtual Machine

- Write a program once, and run it anywhere
  - Architecture independent
  - Operating System independent
- Language itself is clean, robust, garbage collection
- Program compiled into bytecode
  - Interpreted or just-in-time compiled.
  - Lower than native performance



Native Execution vs Emulation/Translation



JAVA and the Interface Goals

- Support deploying software across all computing platforms. ✓
- Provide a platform to securely share hardware resources. ✗

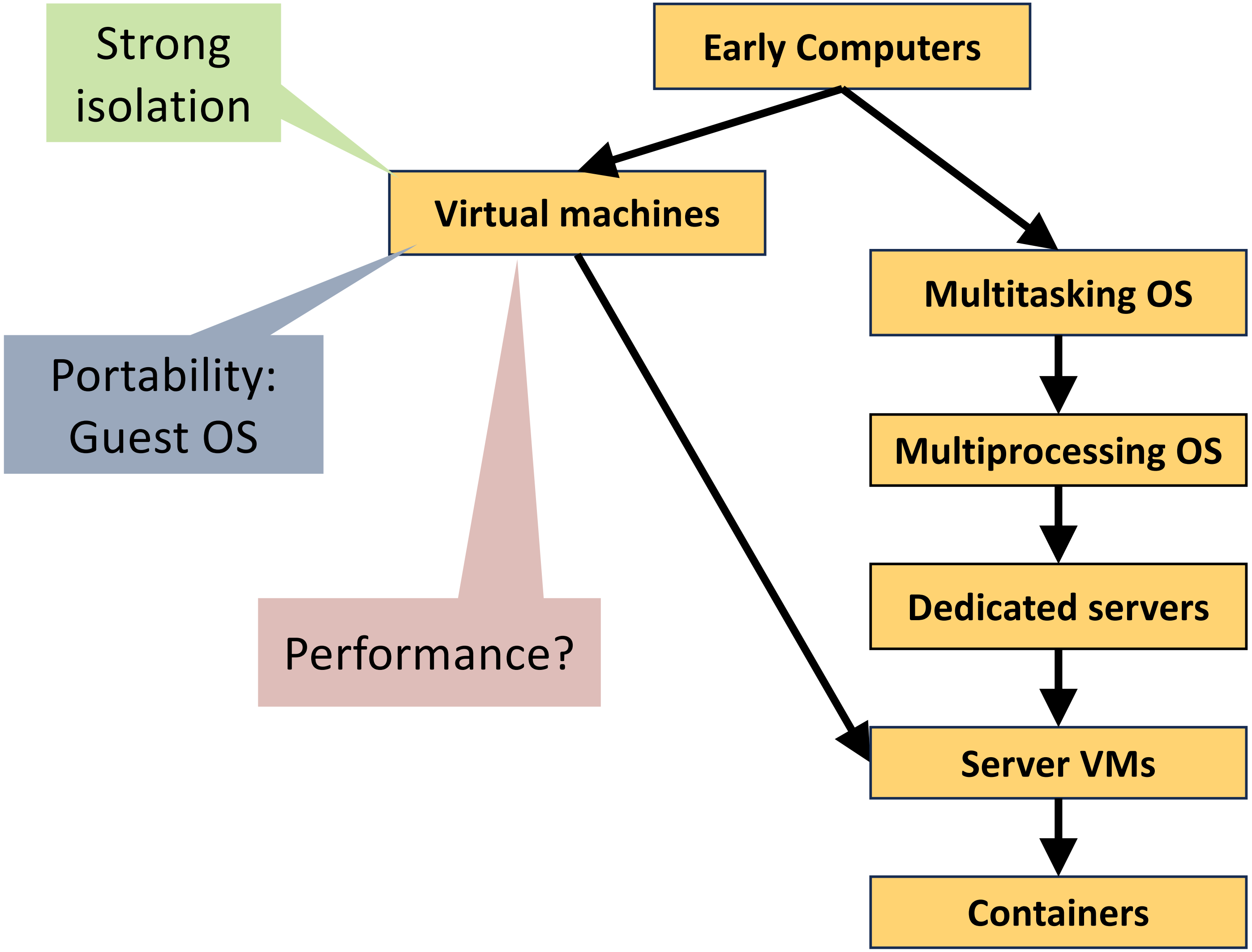
Issues

- Legacy applications
- No isolation nor resource management between applets
- Security
  - Trust JVM implementation? Trust underlying OS?
- Performance compared to native?

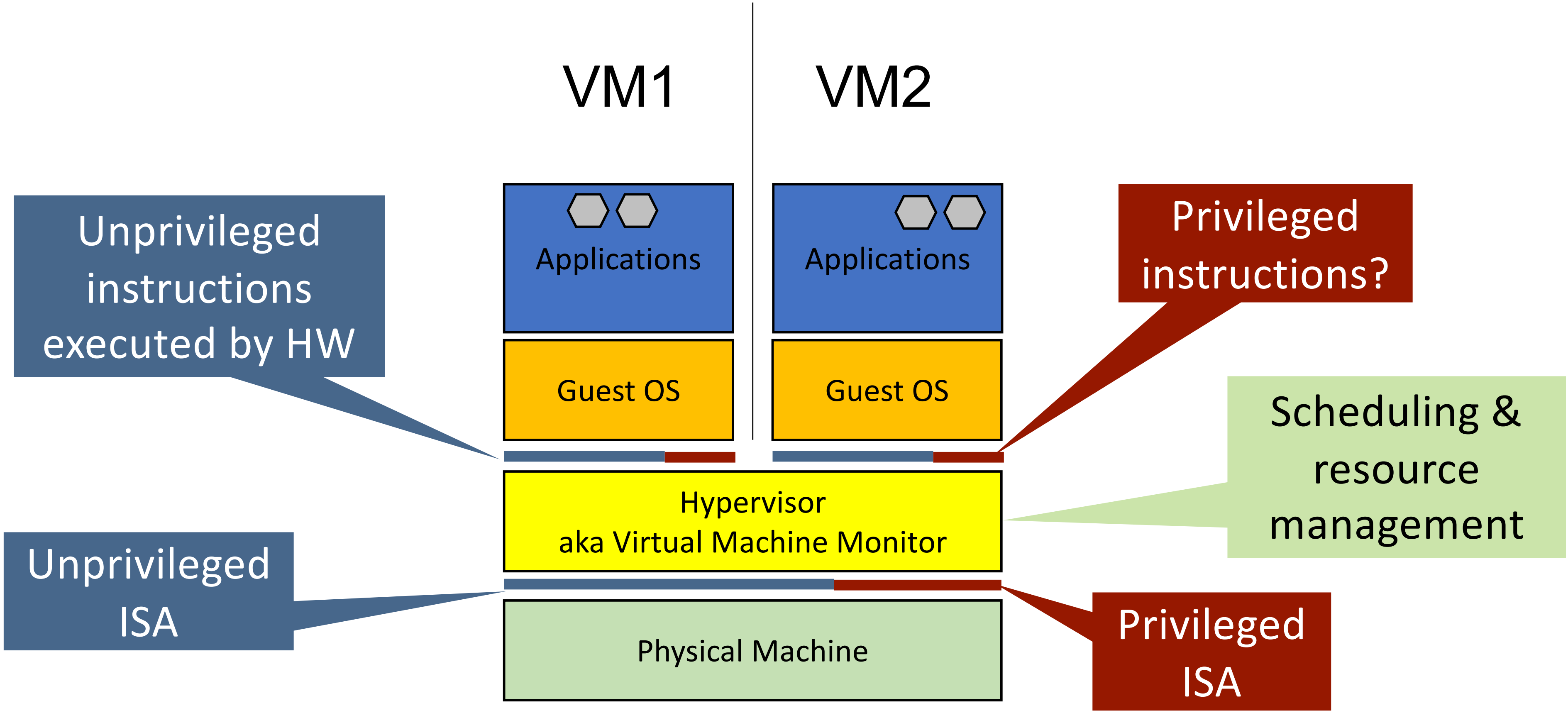
Isn’t This The Job Of The OS?

- Security
  - Trust the underlying OS?
- Legacy application and OSs
- Resource management of existing systems suitable for all applications?
  - Performance isolation?
- What about activities requiring “root” privileges

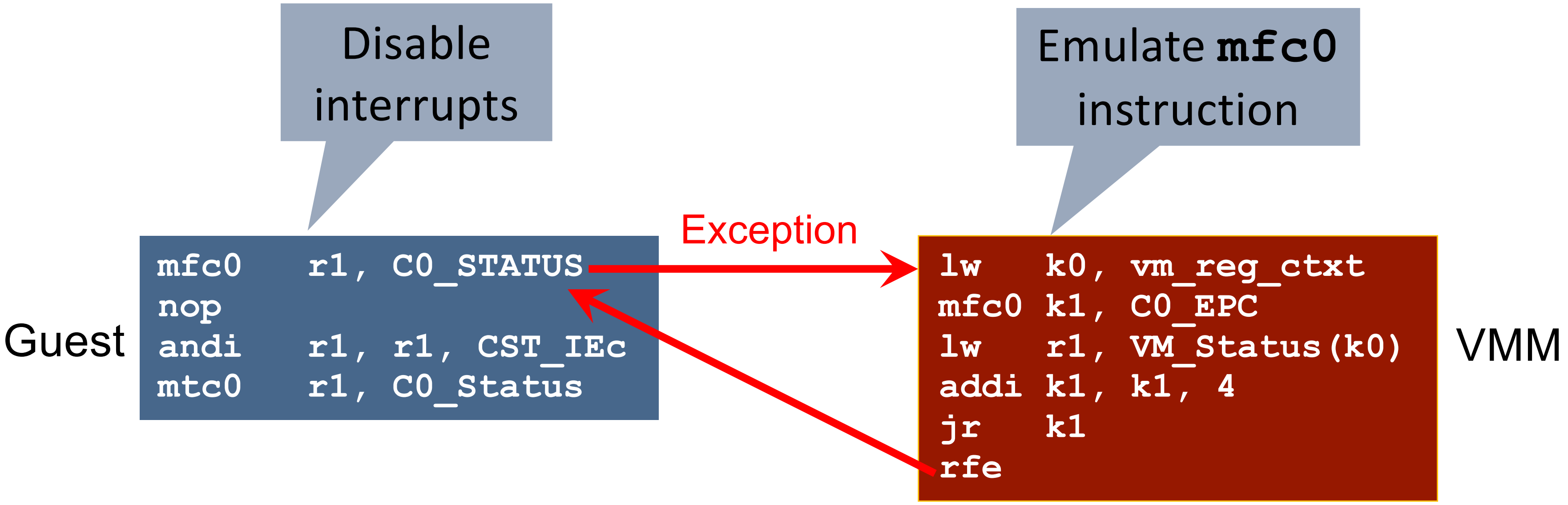
Remember: History of Processes



Virtual Machine: Hypervisor



Privileged Instruction: Trap-and-Emulate



Disable interrupts

Emulate **`mfc0`** instruction

Guest

```
mfc0   r1, C0_STATUS
nop
andi   r1, r1, CST_IEc
mtc0   r1, C0_Status
```

Exception

VMM

```
lw    k0, vm_reg_ctxt
mfc0  k1, C0_EPC
lw    r1, VM_Status(k0)
addi  k1, k1, 4
jr    k1
rfe
```

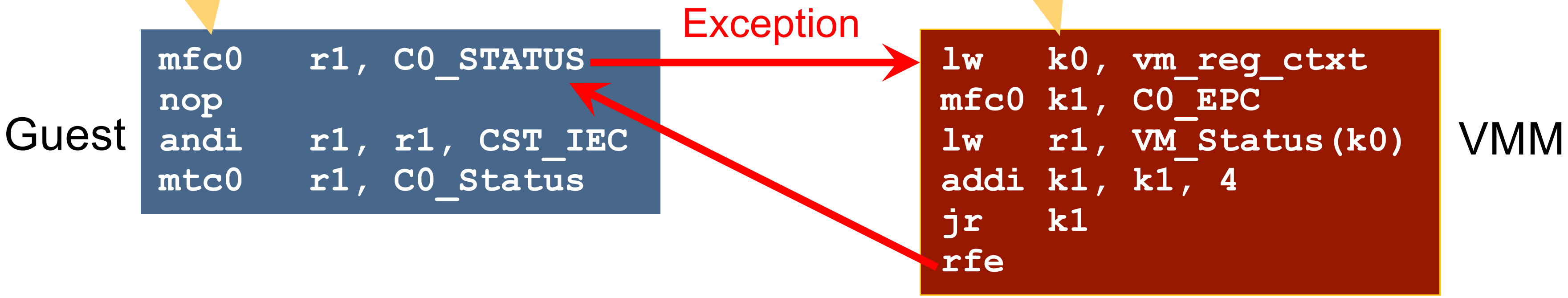
Most instructions do not trap

- prerequisite for efficient virtualisation
- requires VM ISA (almost) same as processor ISA

Trap-and-Emulate Limitations

What if reading privileged state doesn't trap (is a **`nop`**)?

What if the guest uses **`k0`**?



```
mfc0   r1, C0_STATUS
nop
andi   r1, r1, CST_IEC
mtc0   r1, C0_Status
```

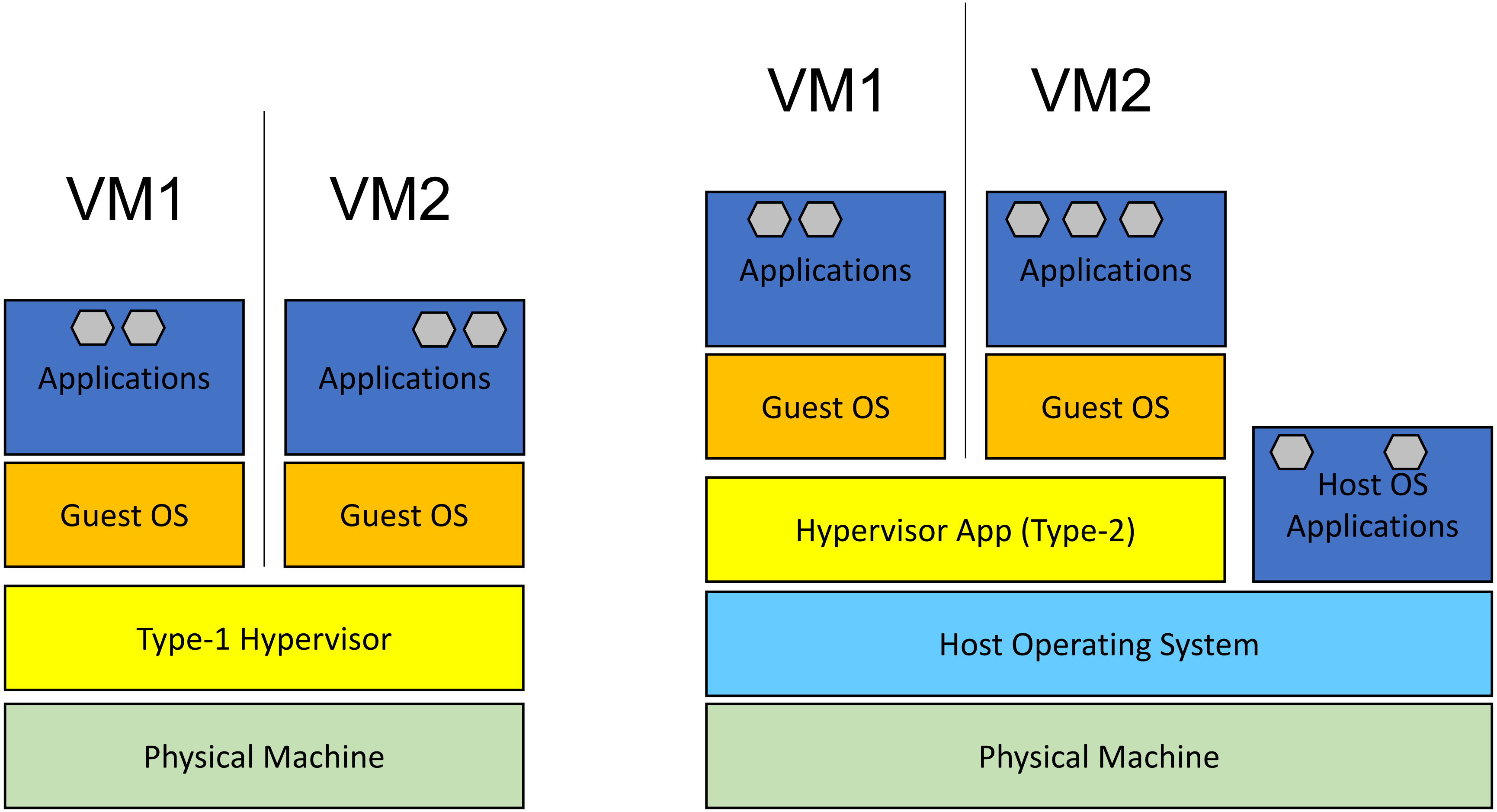
```
lw    k0, vm_reg_ctxt
mfc0  k1, C0_EPC
lw    r1, VM_Status(k0)
addi  k1, k1, 4
jr    k1
rfe
```

Many ISAs not trap&emulate virtualisable!

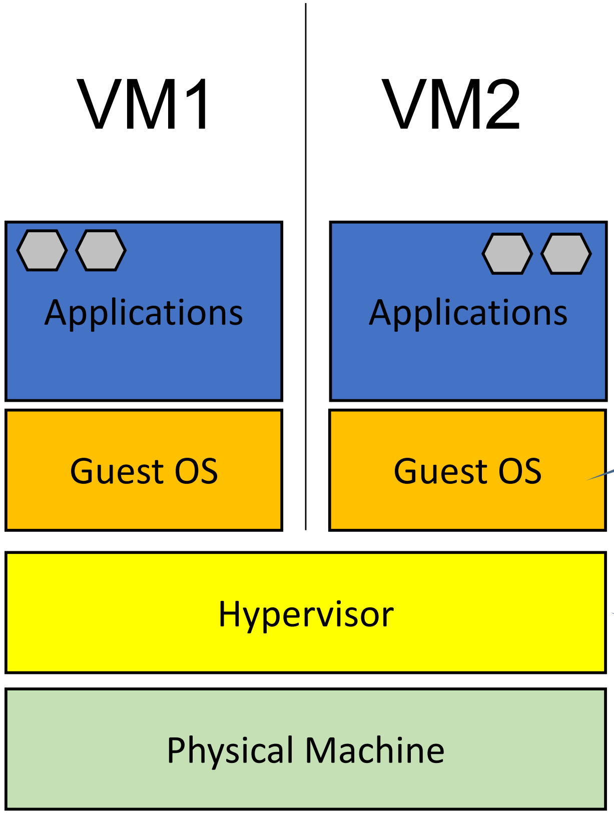
- MIPS k0, k1
- x86 **`popf`** – no-op from user mode
- Original Arm, …

Virtualisation ISA extensions

Native (Type-1) vs. Hosted (Type-2) Hypervisor



Type-1 (Native) Hypervisor

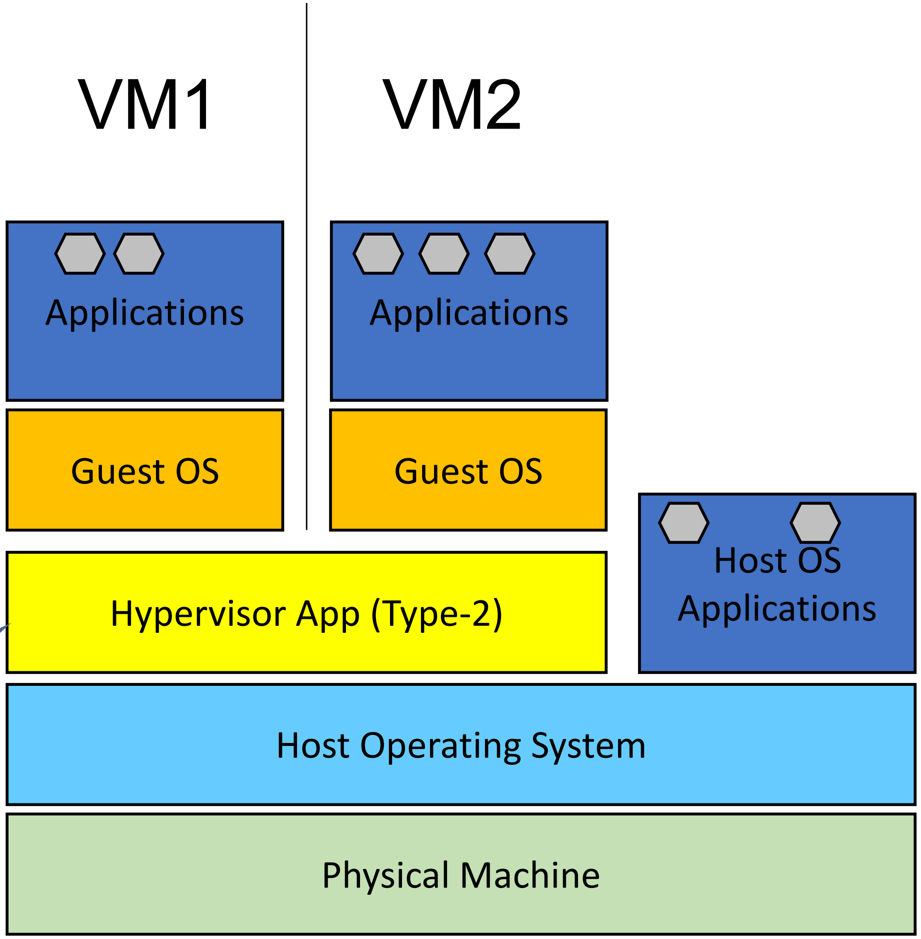


Runs in un-privileged mode

- Runs in privileged mode
- implements virtual kernel/user mode

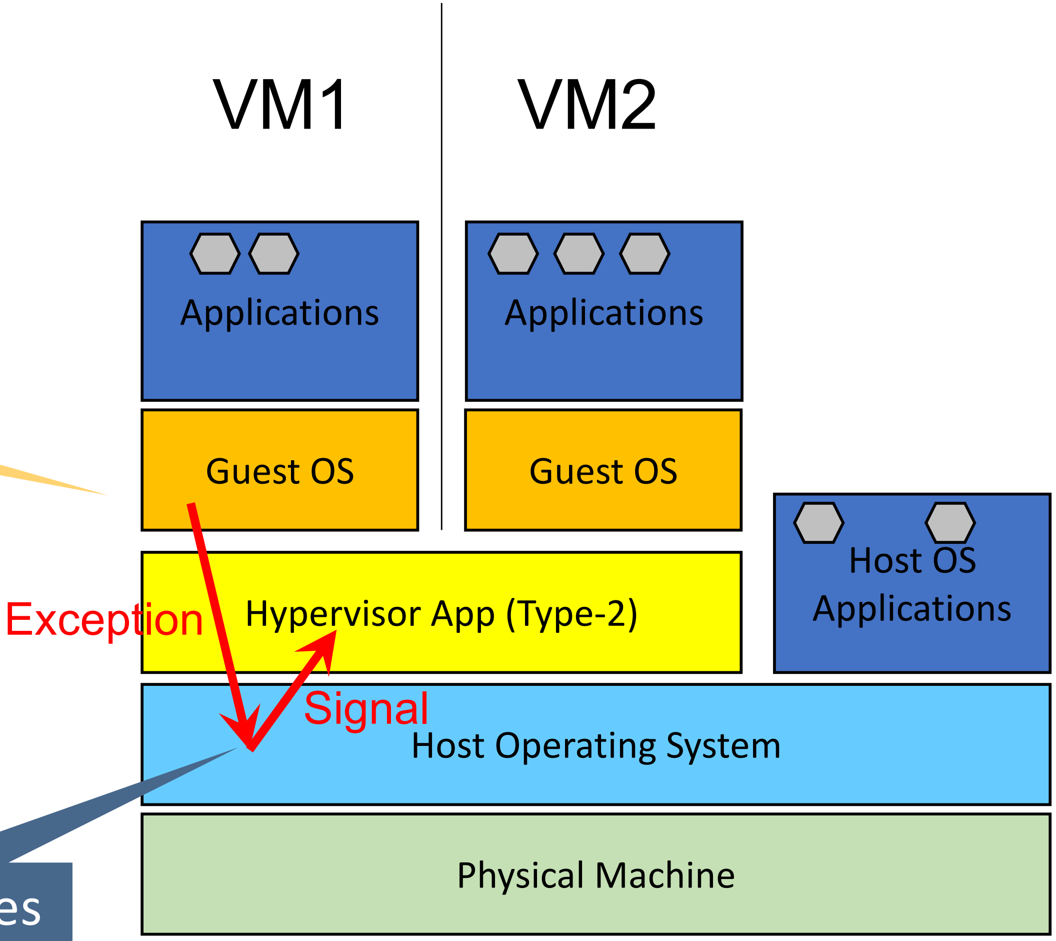
Type-2 (Hosted) Hypervisor

How trap guest instructions?



- Runs in un-privileged mode
- implements virtual kernel/user mode
- uses host for I/O etc

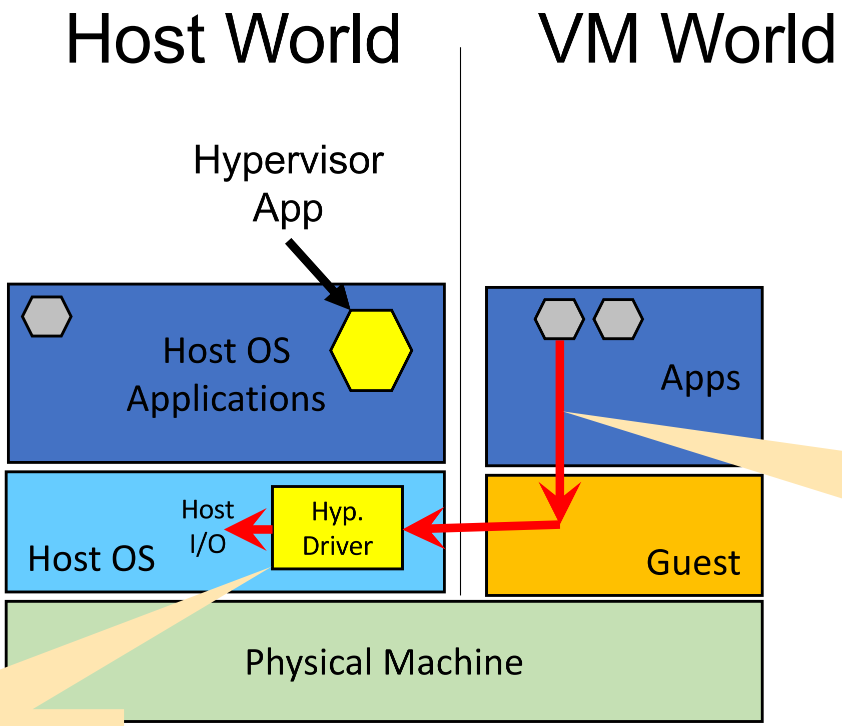
Type-2 Hypervisor: Trap&Emulate



Alternative: Re-write guest binary to invoke hypervisor directly

Host emulates trap handling

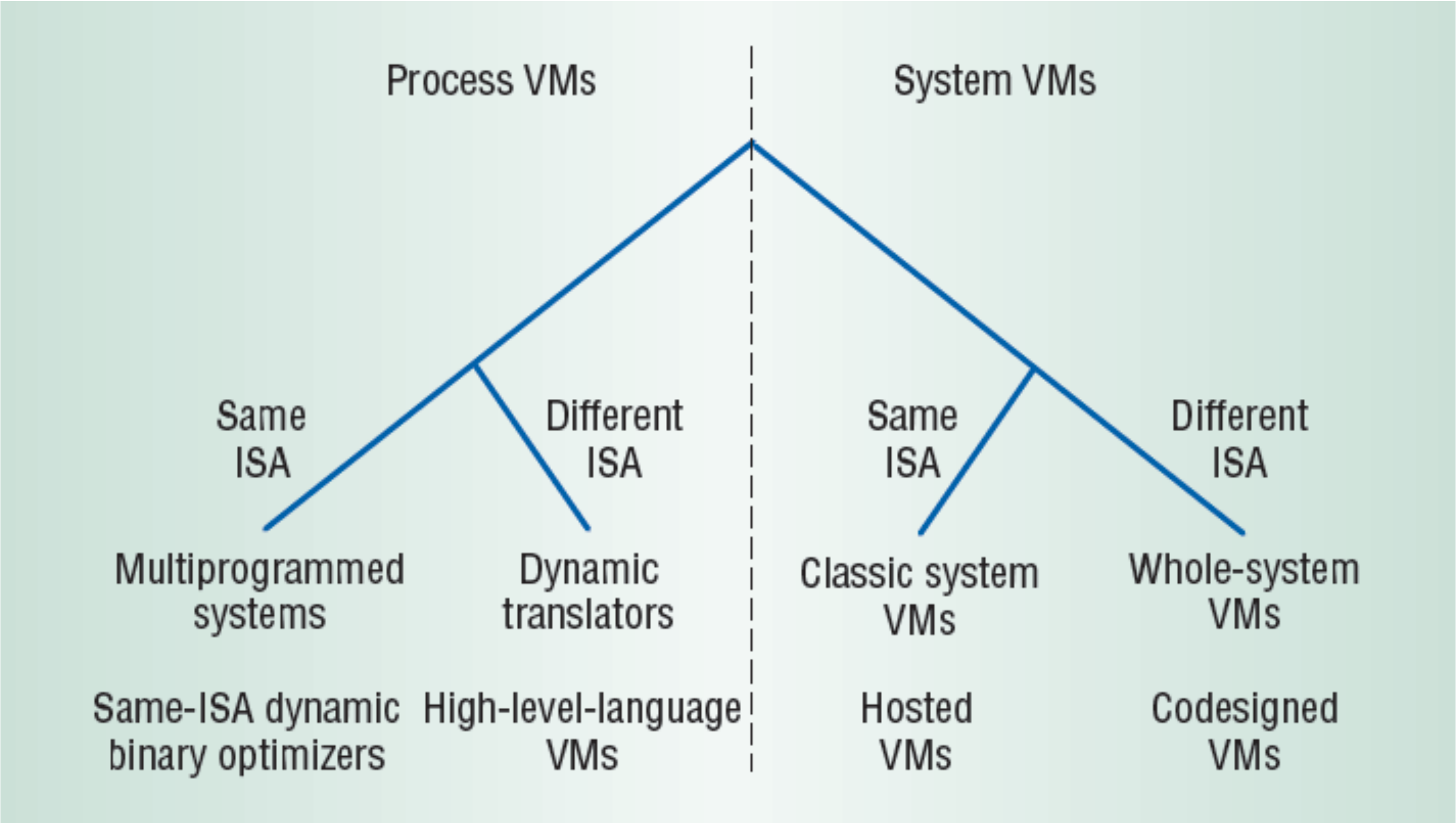
Type-2 Hypervisor: I/O



VM I/O re-directs to Host I/O via hypervisor driver

Hypervisor app installs driver in host

Taxonomy of Virtual Machines


Process VMs
System VMs
Same ISA
Different ISA
Same ISA
Different ISA
Multiprogrammed systems
Dynamic translators
Classic system VMs
Whole-system VMs
Same-ISA dynamic binary optimizers
High-level-language VMs
Hosted VMs
Codesigned VMs