School of Computer Science & Engineering

COMP3891/9283 Extended Operating Systems

2026 T2 Week 04

Process Abstractions IV: Virtual Machines

Gernot Heiser

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)

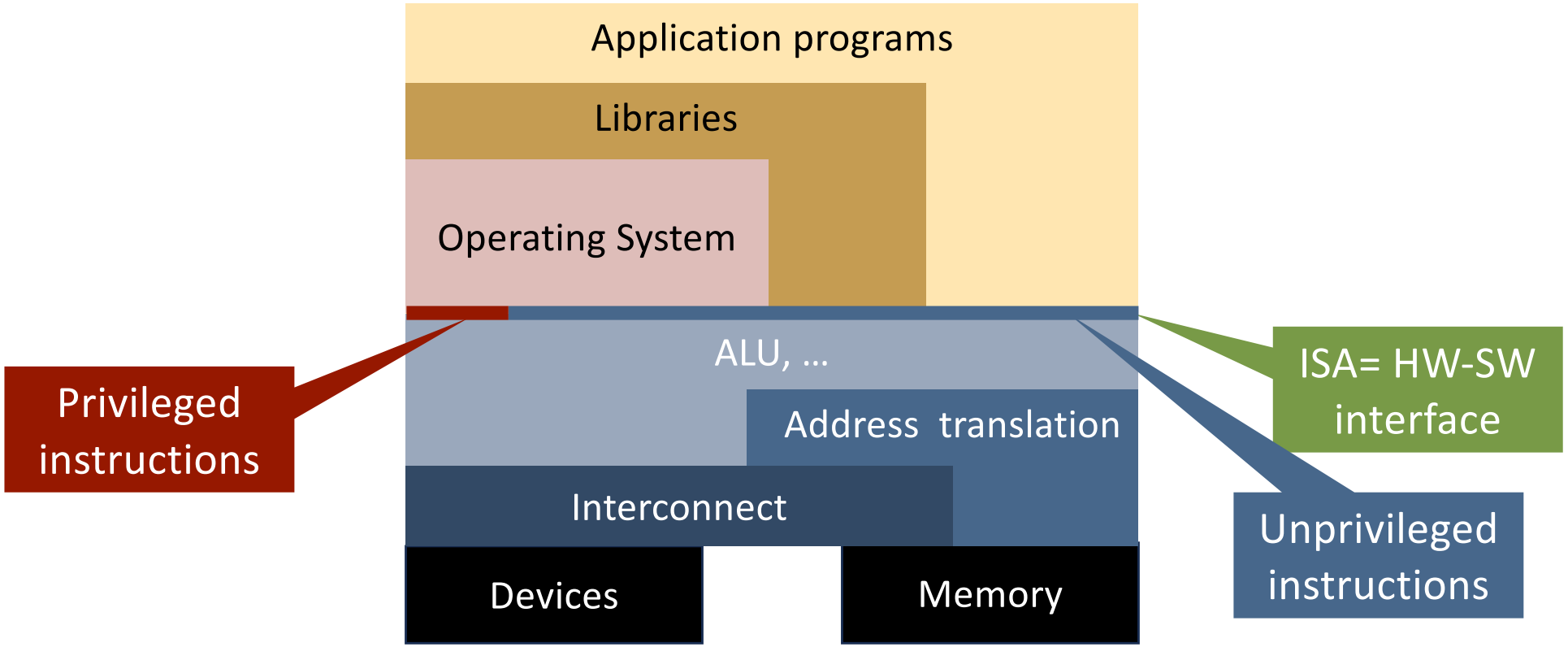
Virtual Machines: References

- 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

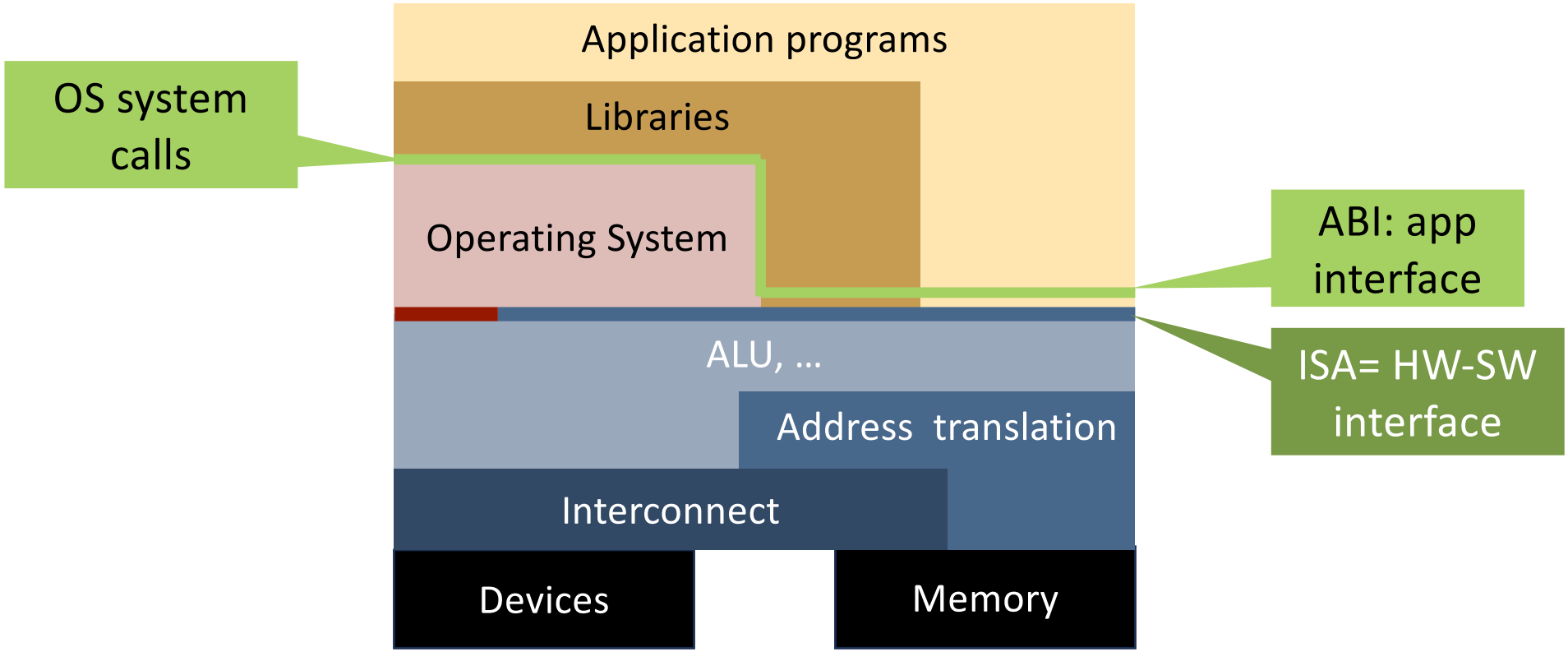
- Operating systems provide well-defined interfaces
  - Abstract hardware details
    - Simplify
    - Enable application portability across hardware differences
- Hardware instruction set architectures are another well-defined interface
  - Example AMD and Intel both implement (mostly) the same ISA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

Instruction Set Architecture (ISA)



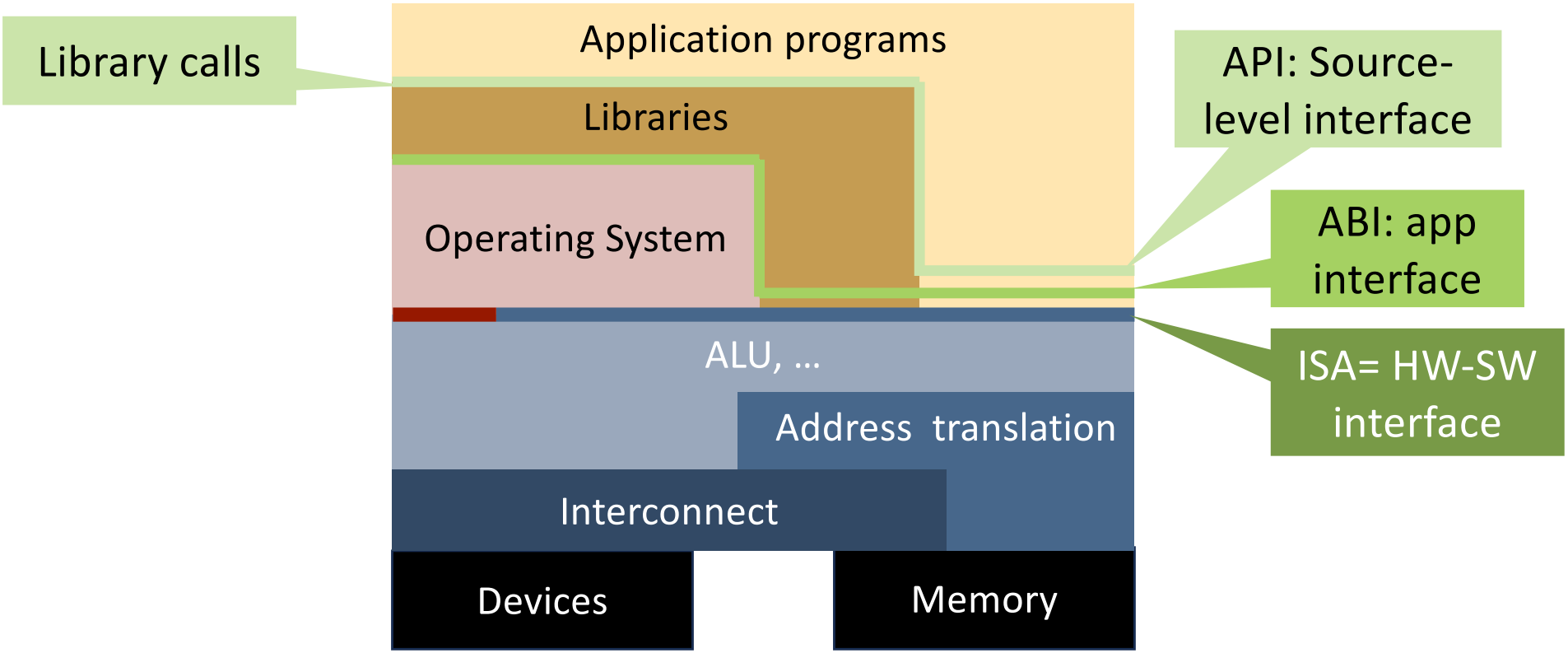
Portability across processors sharing ISA

Application Binary Interface (ABI)



Privilege boundary, Portability within ISA

Application Programming Interface (API)



Abstracts ABI, source-code portability across ISAs

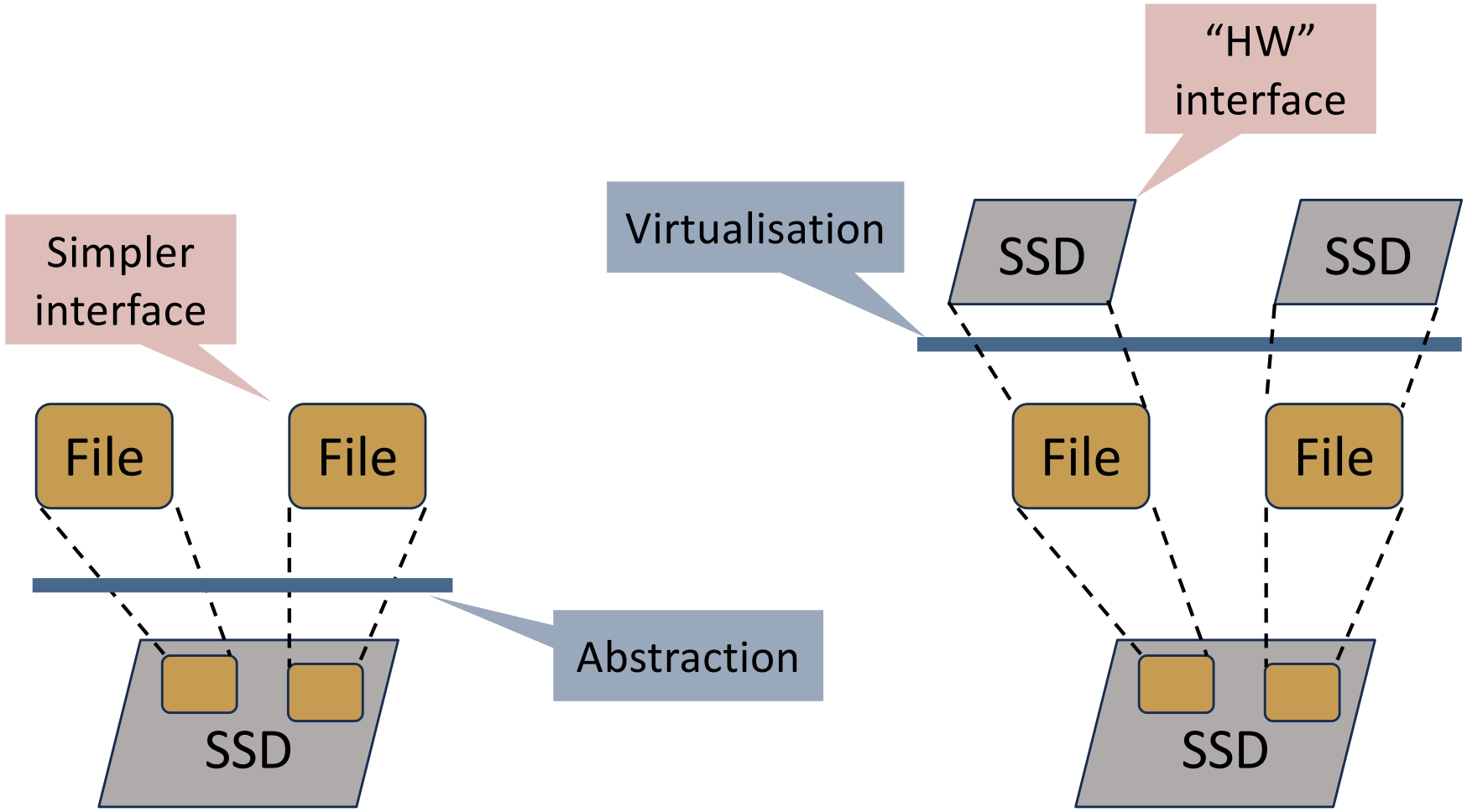
Interface Goals

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

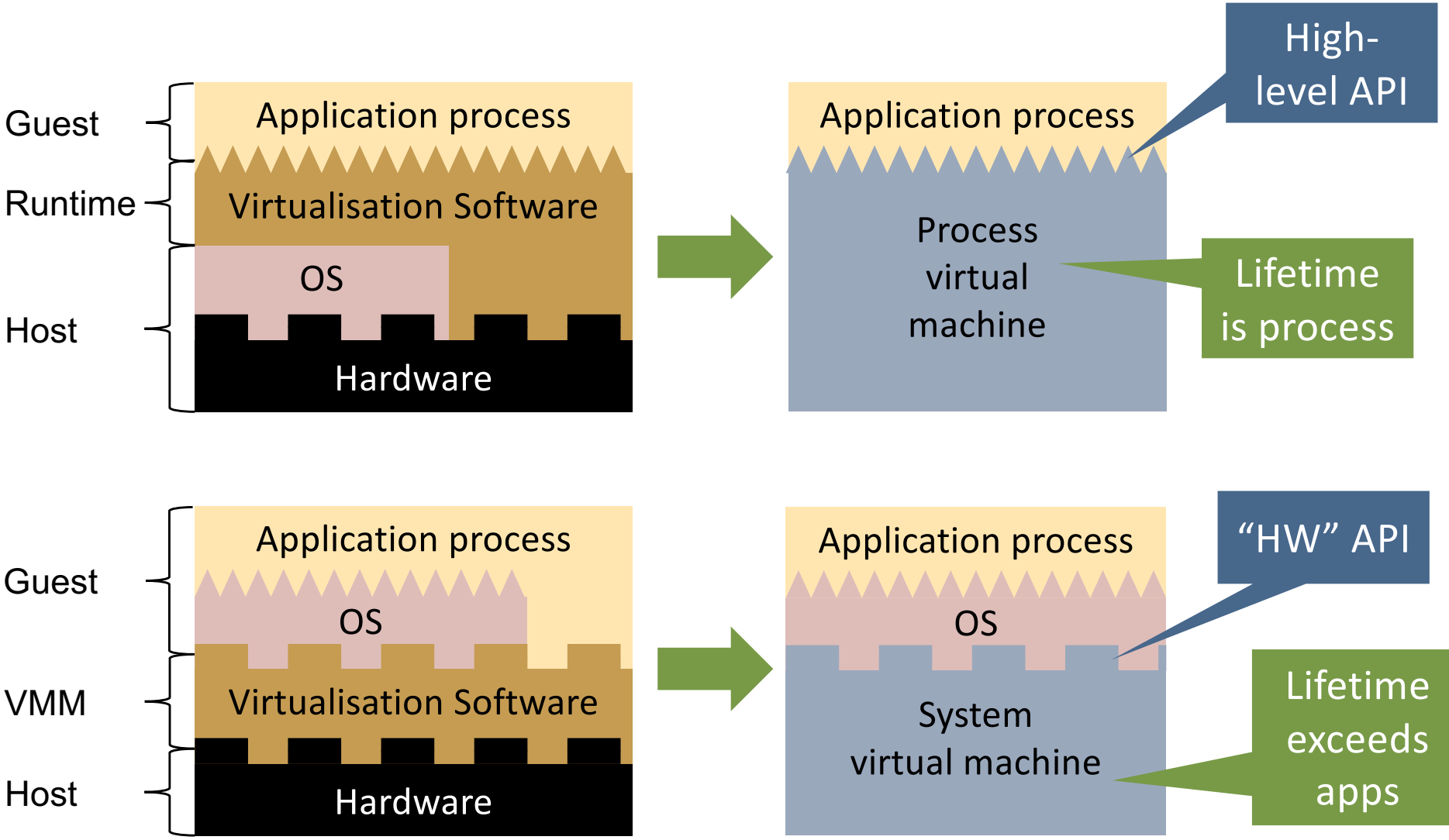
OS as a Virtual Machine

- 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

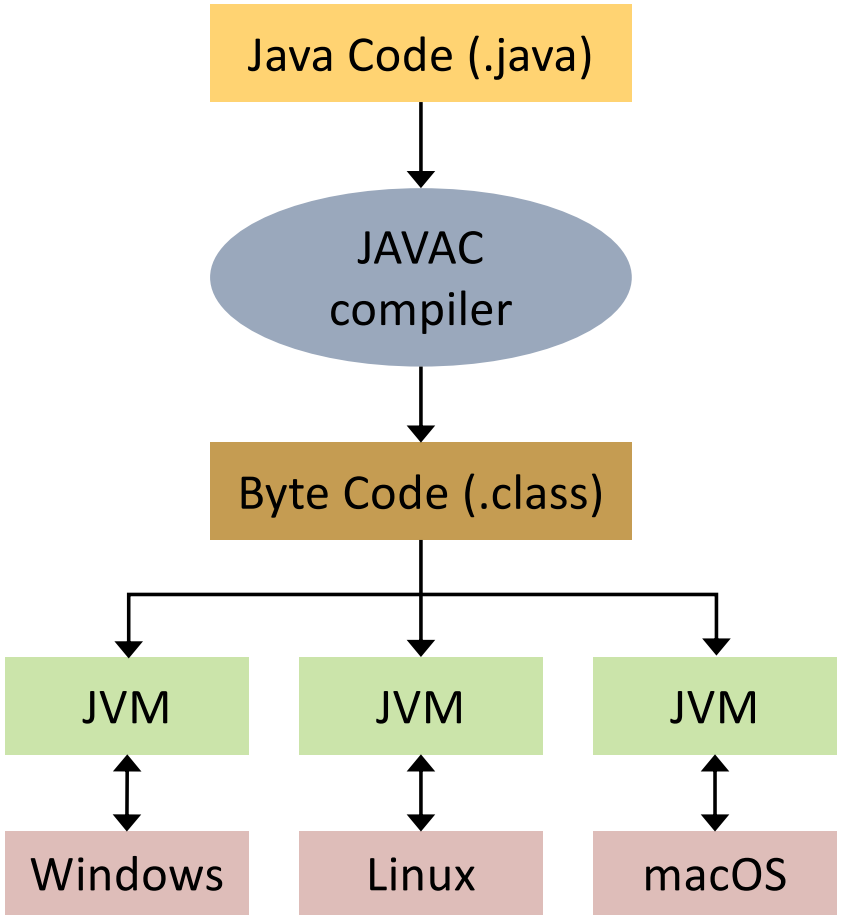


Process vs System Virtual Machine

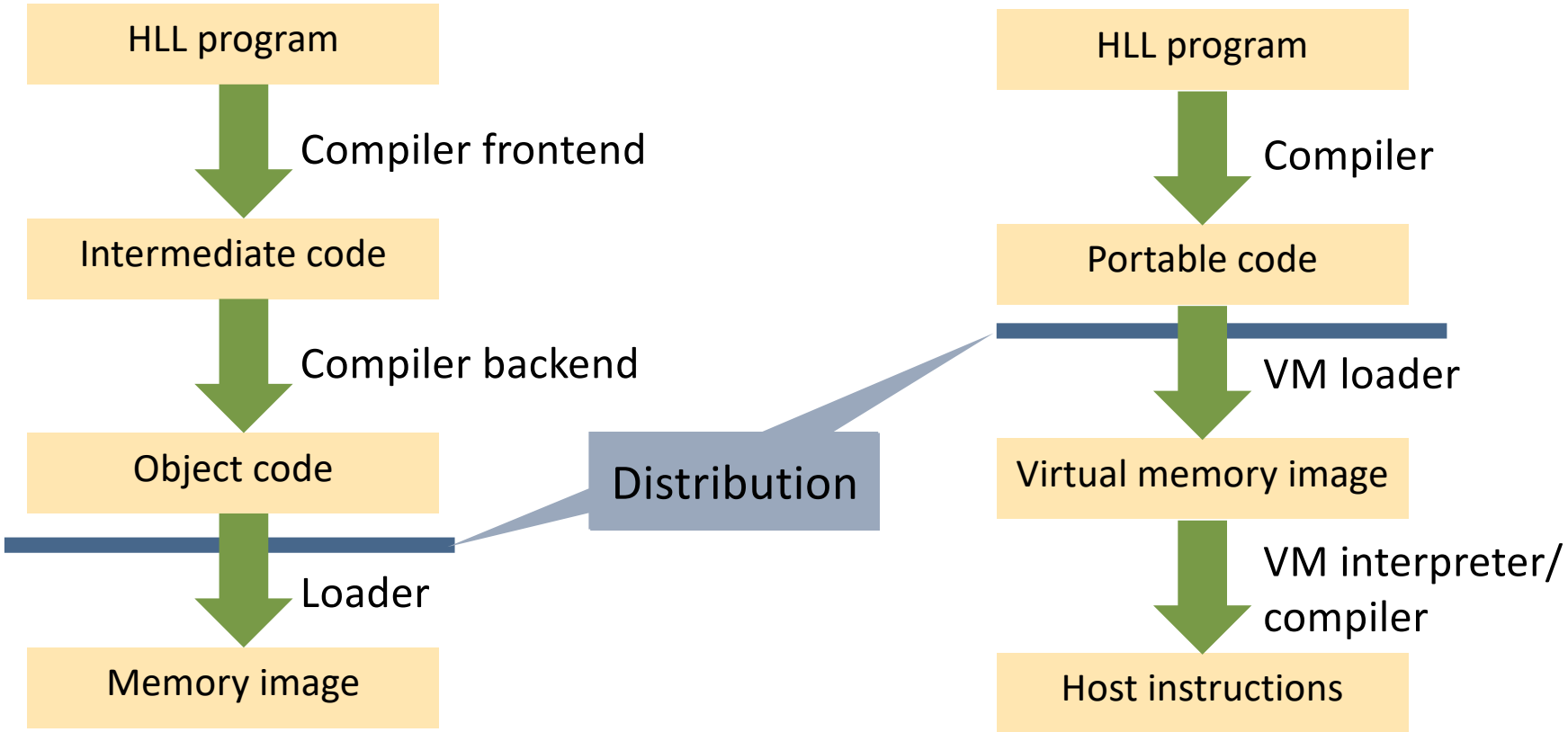


JAVA – Process Virtual Machine

- 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. 

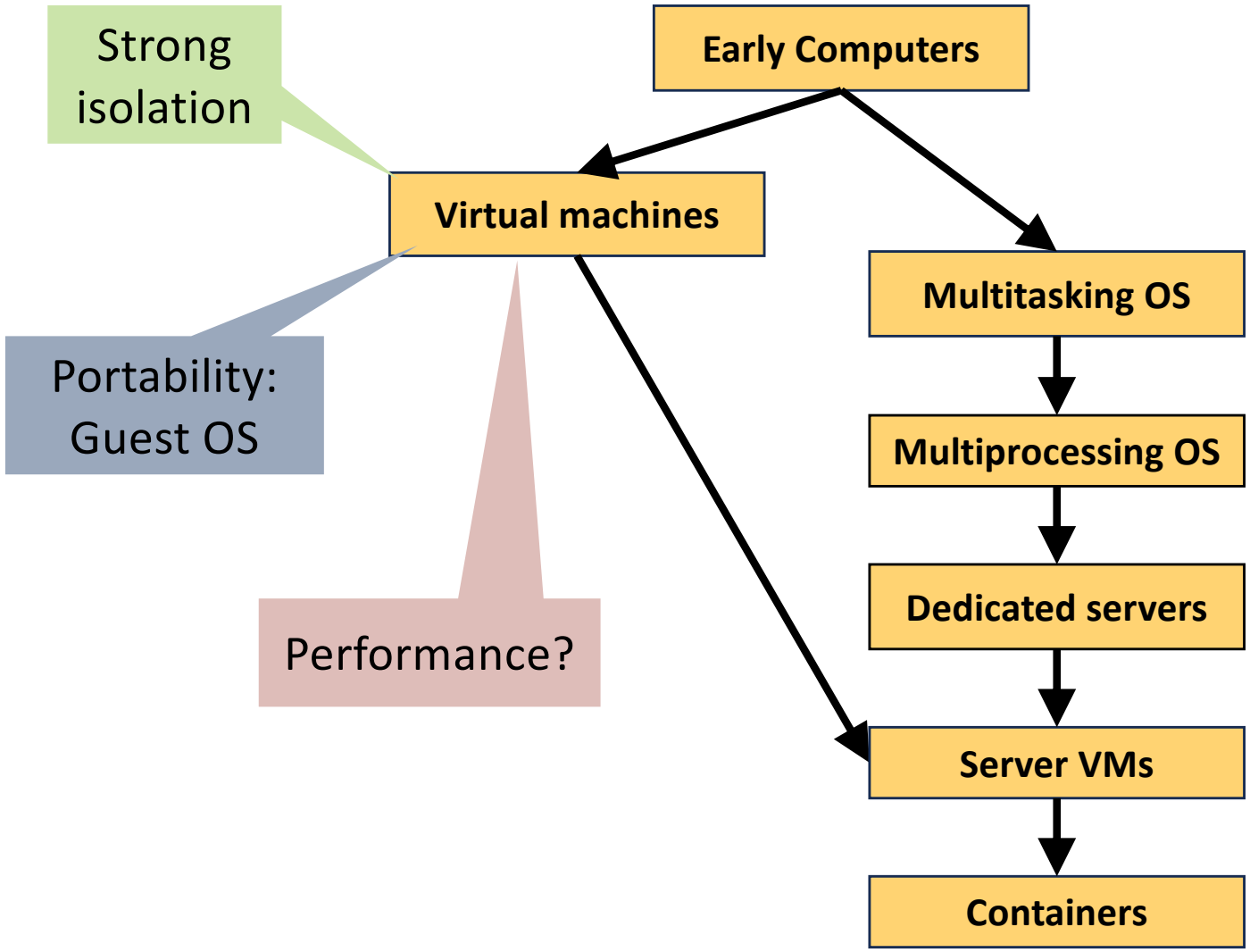
Issues

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

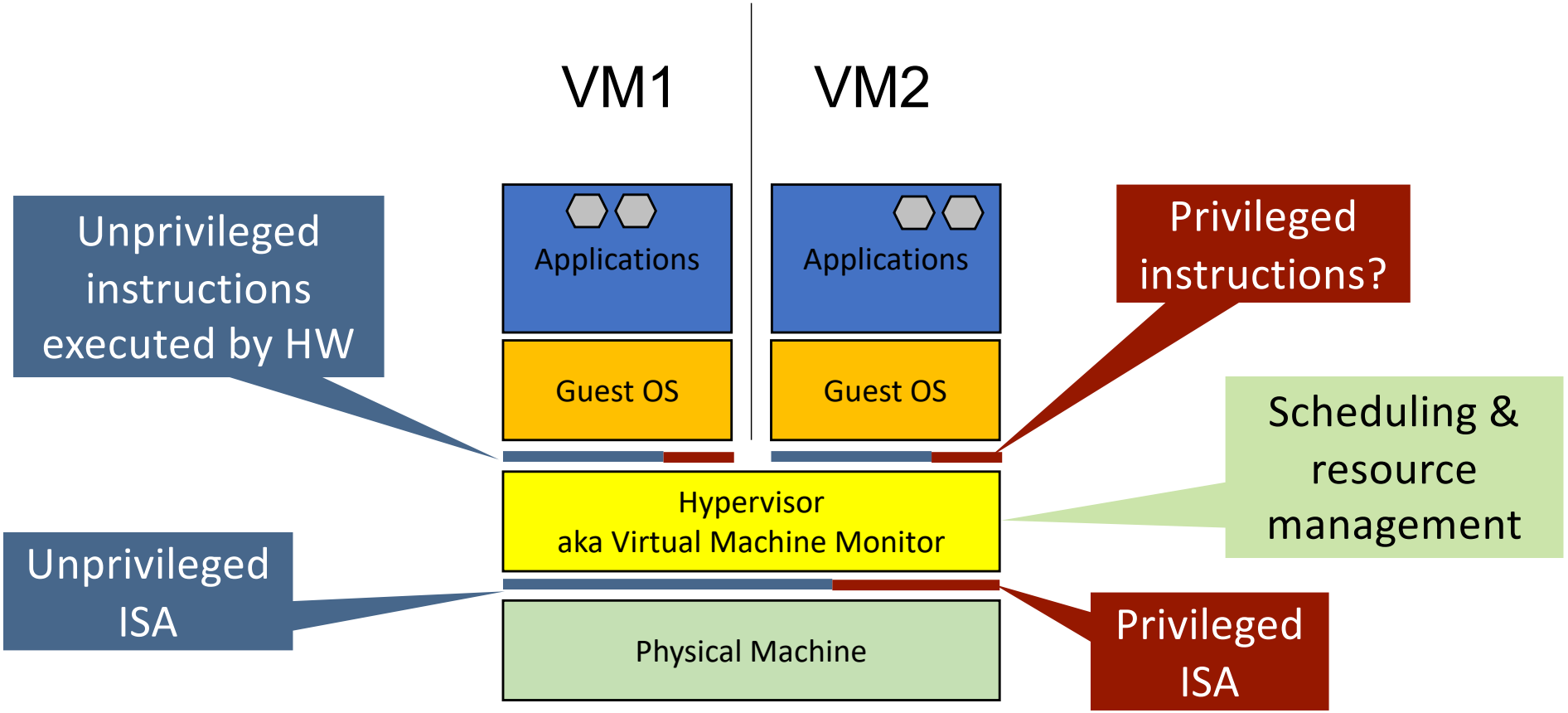
Isn’t This The Job Of The OS?

- 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

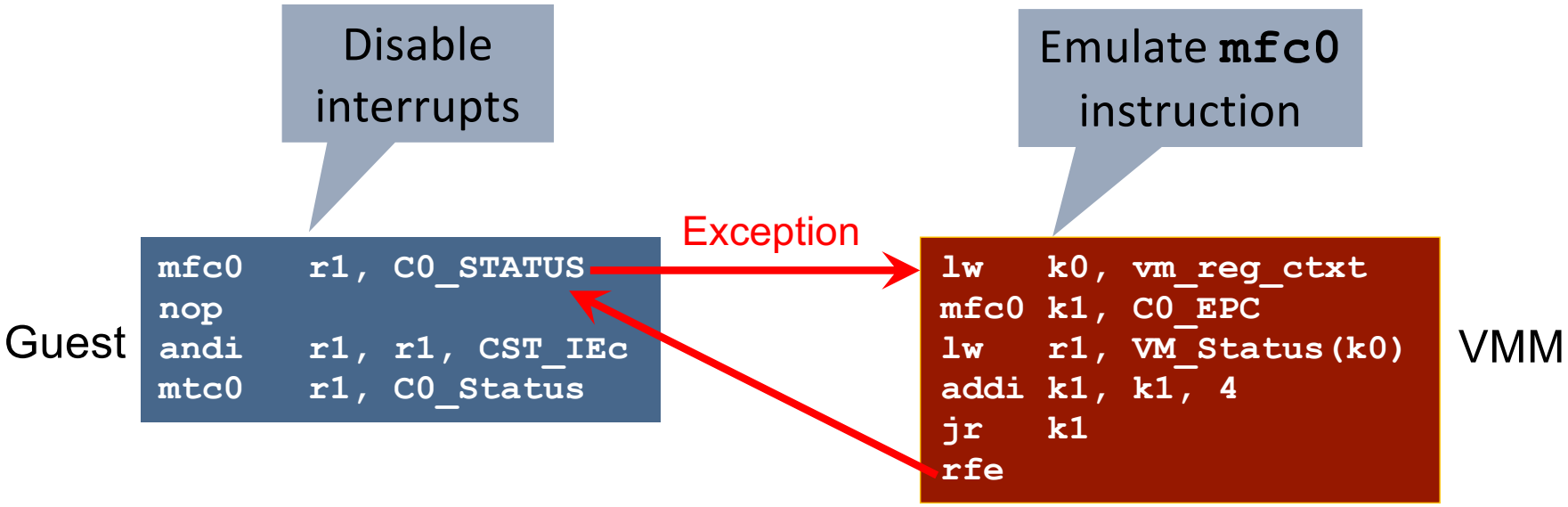
Remember: History of Processes



Virtual Machine: Hypervisor



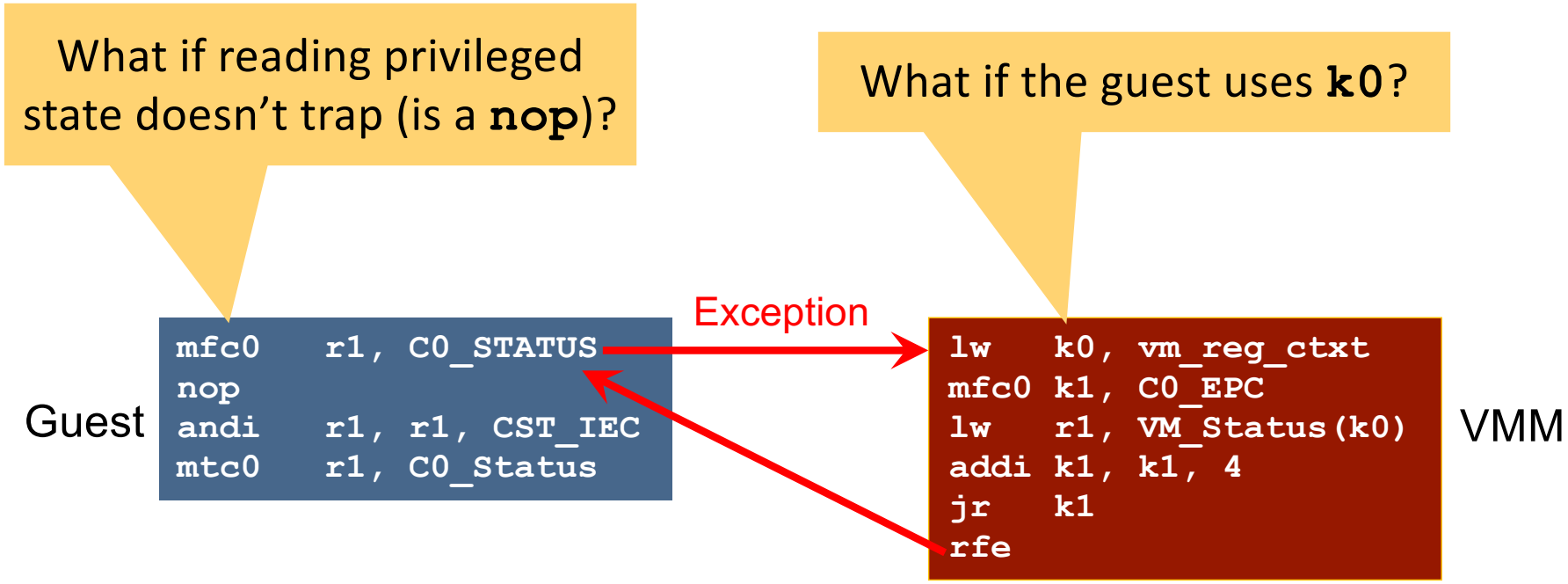
Privileged Instruction: Trap-and-Emulate



Most instructions do not trap

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

Trap-and-Emulate Limitations

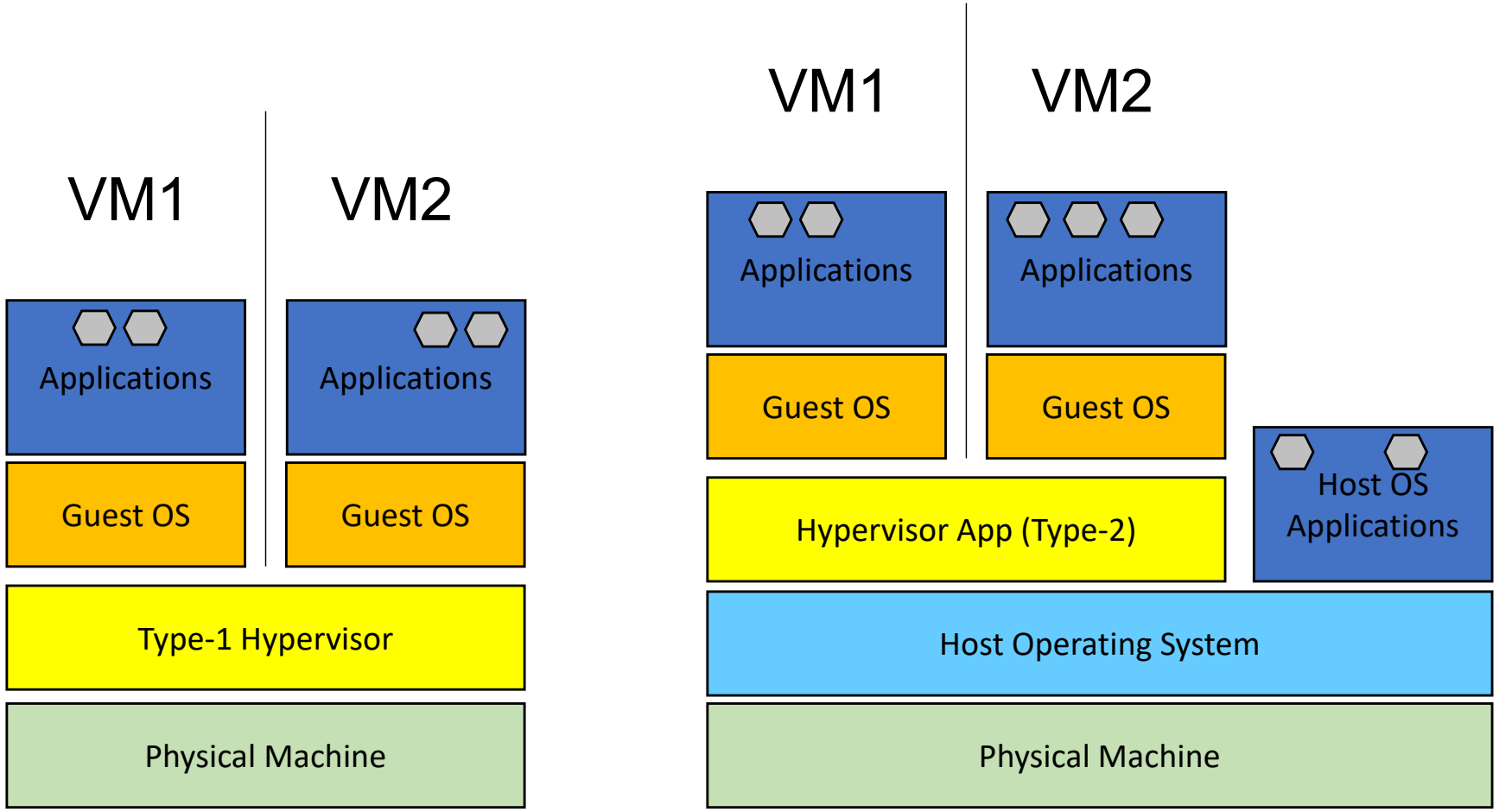


Many ISAs not trap&emulate virtualisable!

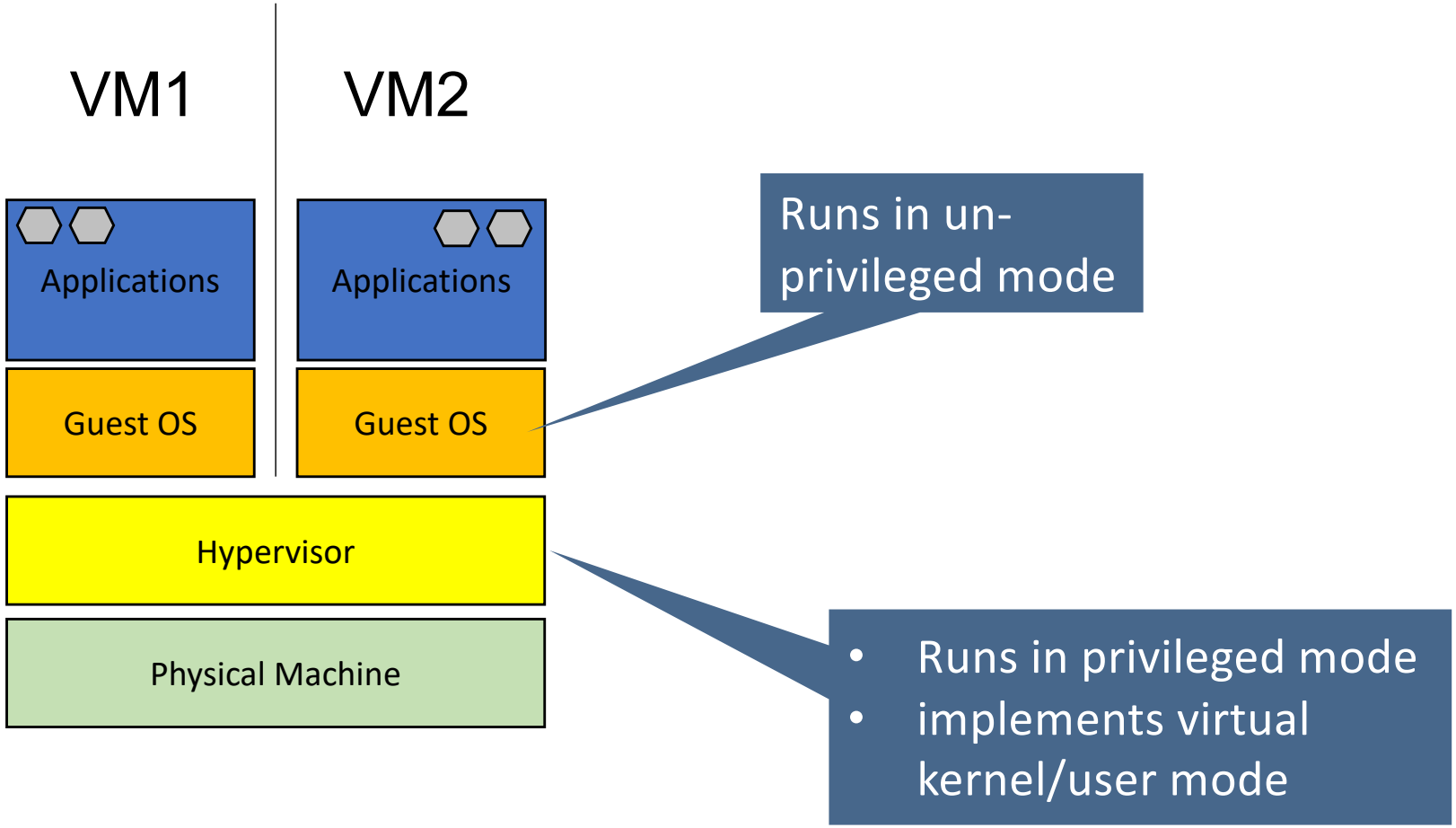
- MIPS k0, k1
- x86 **popf** – no-op from user mode
- Original Arm, …

Virtualisation ISA extensions

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

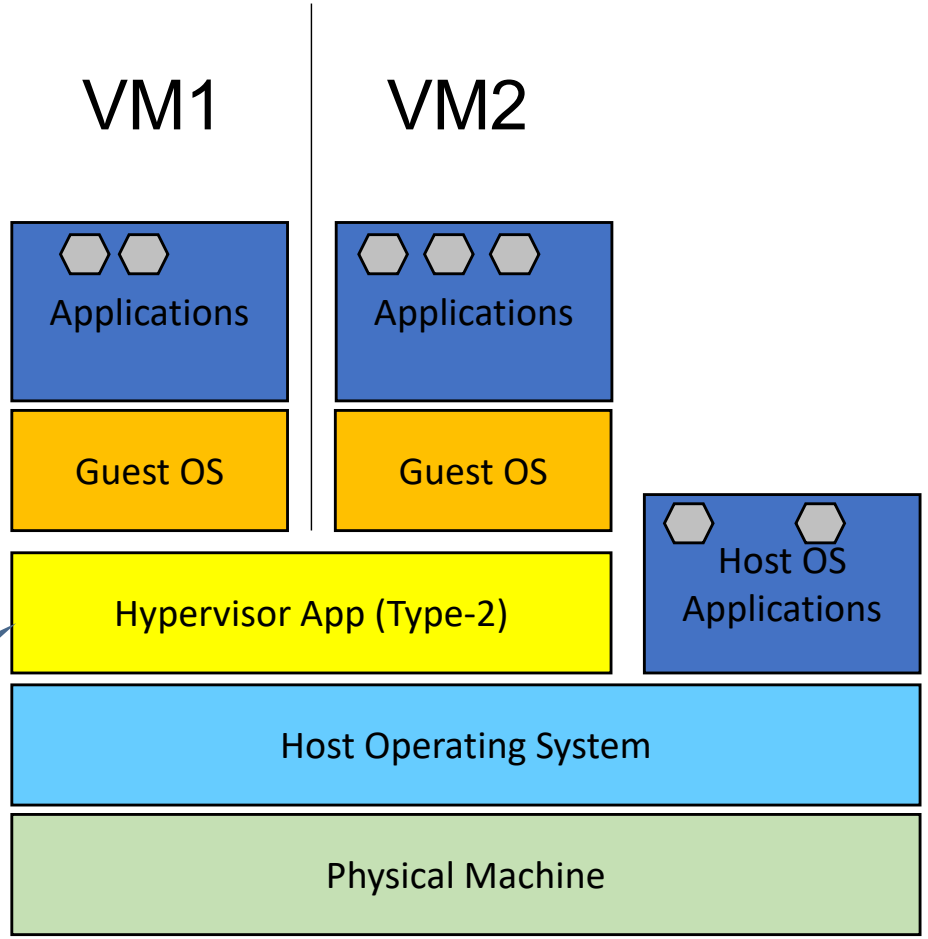


Type-1 (Native) Hypervisor



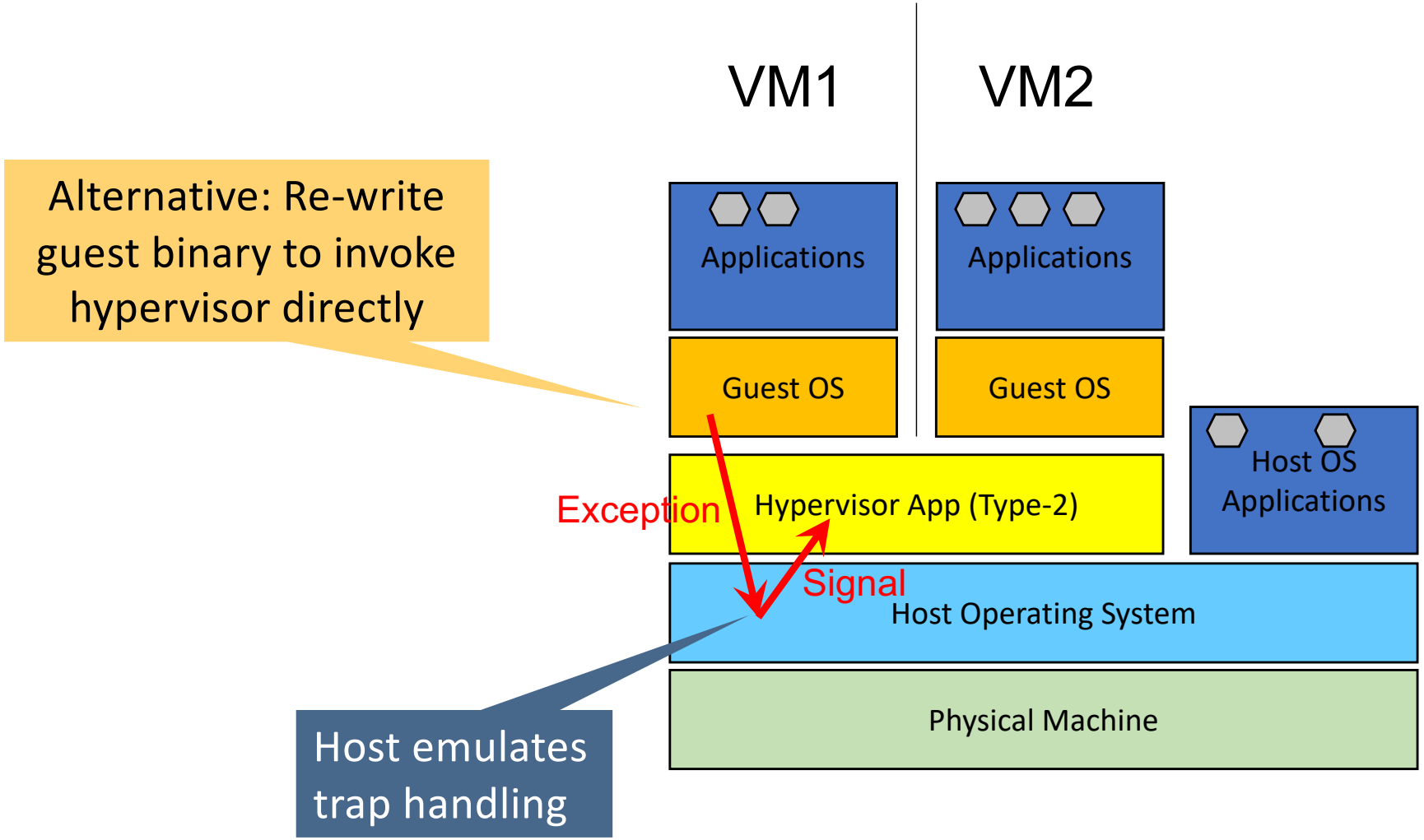
Type-2 (Hosted) Hypervisor

How trap guest instructions?

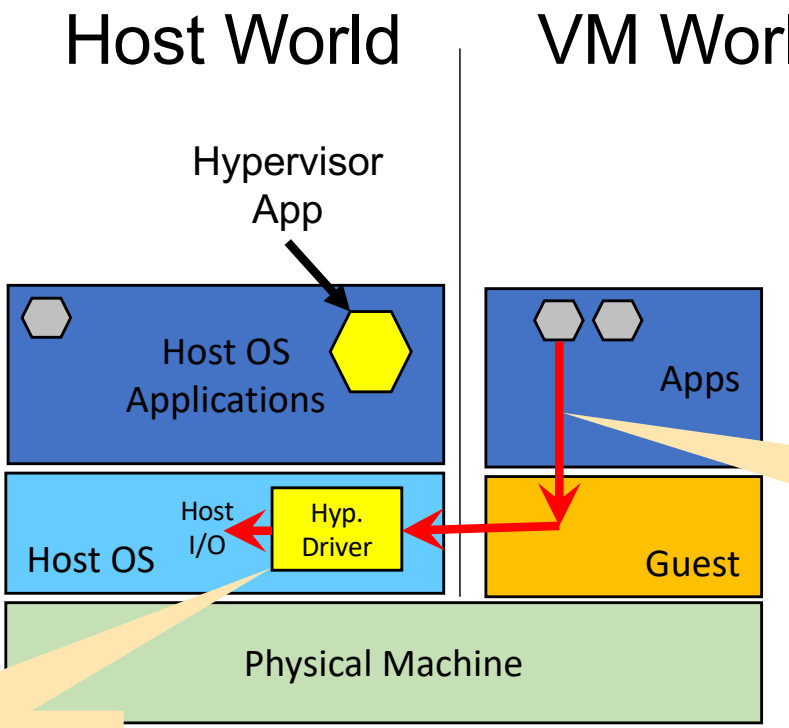


- Runs in un-privileged mode
- implements virtual kernel/user mode
- uses host for I/O etc

Type-2 Hypervisor: Trap&Emulate



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