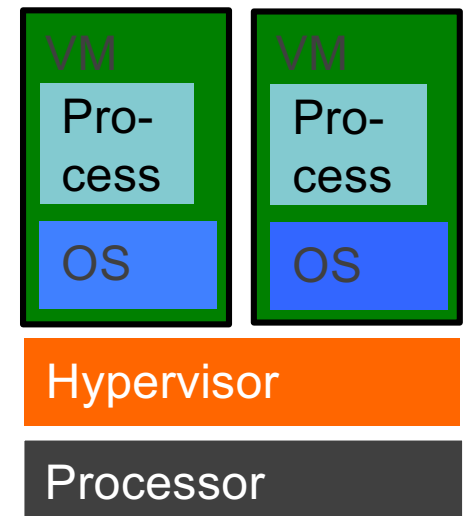




School of Computer Science & Engineering
COMP9242 Advanced Operating Systems

2025 T3 Week 03 Part 2
Virtualisation Principles
@GernotHeiser



Copyright Notice

These slides are distributed under the Creative Commons Attribution 3.0 License

- You are free:
 - to share—to copy, distribute and transmit the work
 - to remix—to adapt the work
- under the following conditions:
 - **Attribution:** You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:

“Courtesy of Gernot Heiser, UNSW Sydney”

The complete license text can be found at
<http://creativecommons.org/licenses/by/3.0/legalcode>

Today's Lecture

- What are virtual machines, and why do we have them
- Mechanics: how do they work
- Modern hardware support
- Fun and games with hypervisors
- Generally provide background for later comparison with microkernels

Virtual Machine Basics

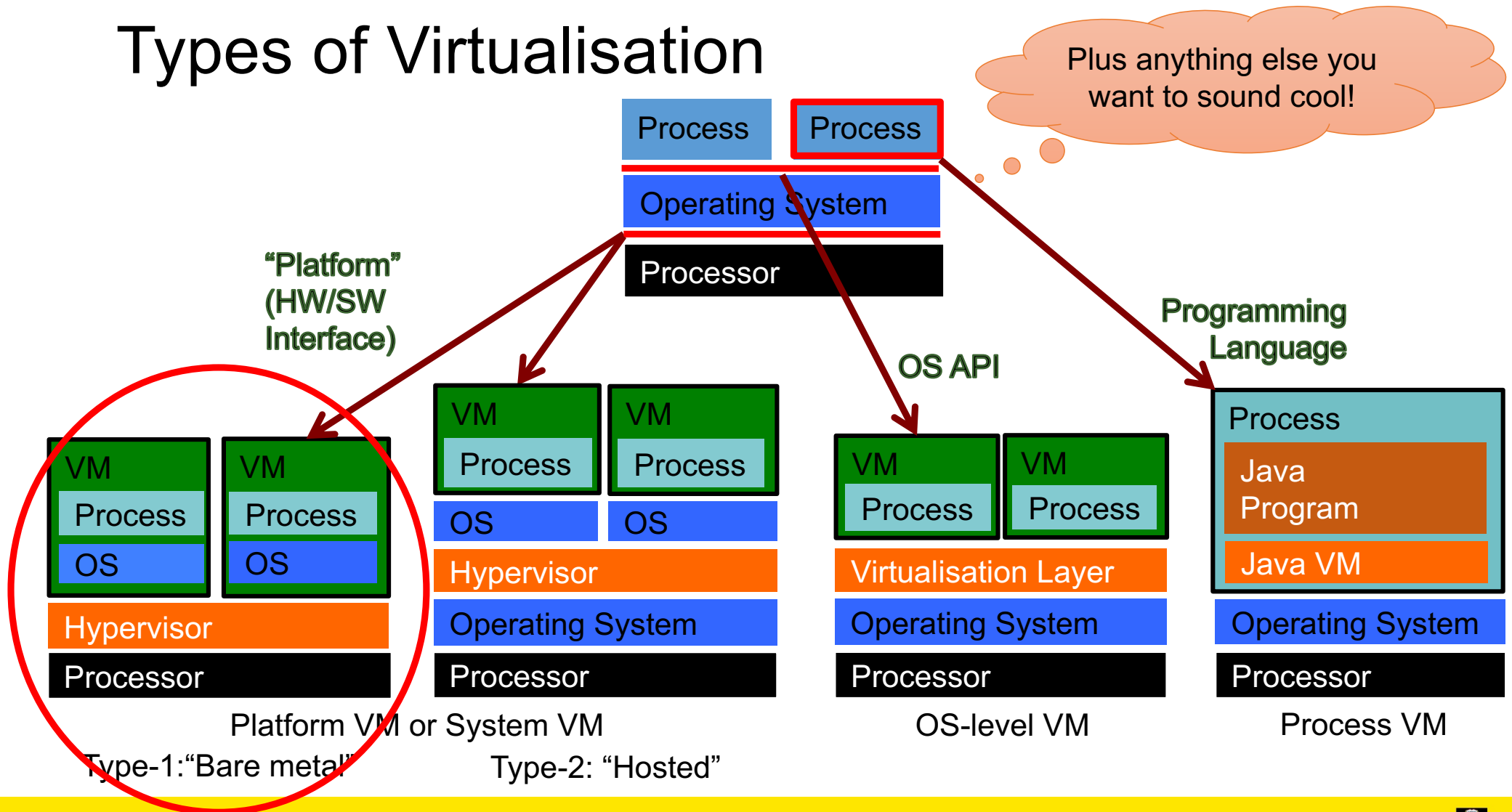
Virtual Machine (VM)

“A VM is an efficient, isolated duplicate of a real machine” [Popek&Goldberg 74]

- **Duplicate:** VM should behave identically to the real machine
 - Programs cannot distinguish between real or virtual hardware
 - Except for:
 - Fewer resources (potentially different between executions)
 - Some timing differences (when dealing with devices)
- **Isolated:** Several VMs execute without interfering with each other
- **Efficient:** VM should execute at speed close to that of real hardware
 - Requires that most instructions are executed directly by real hardware

“Hypervisor” aka “virtual machine monitor” (VMM):
Software layer implementing the VM

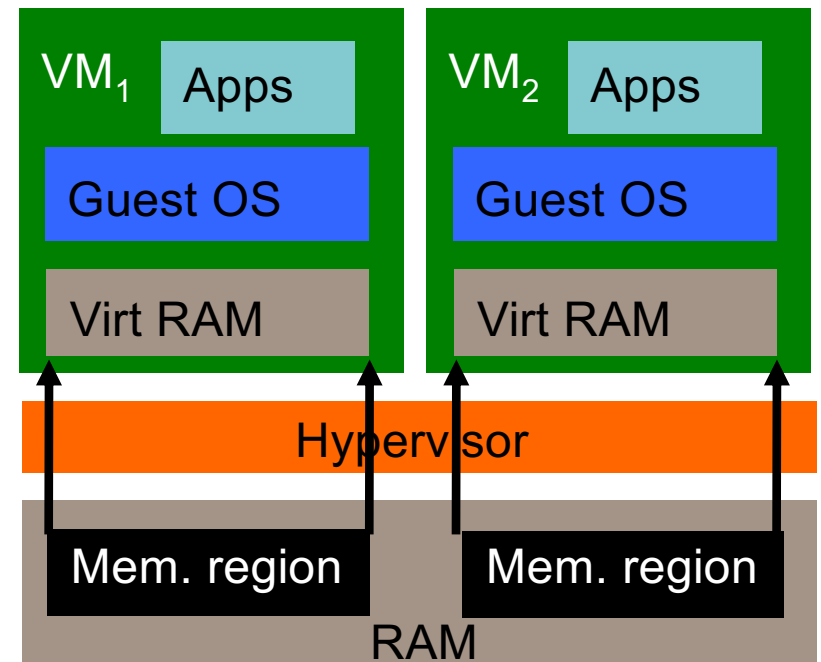
Types of Virtualisation



Why Virtual Machines?

- Historically used for easier sharing of expensive mainframes
 - Run several (even different) OSes on same machine
 - called *guest operating system*
 - Each on a subset of physical resources
 - Can run single-user single-tasked OS in time-sharing mode
 - legacy support

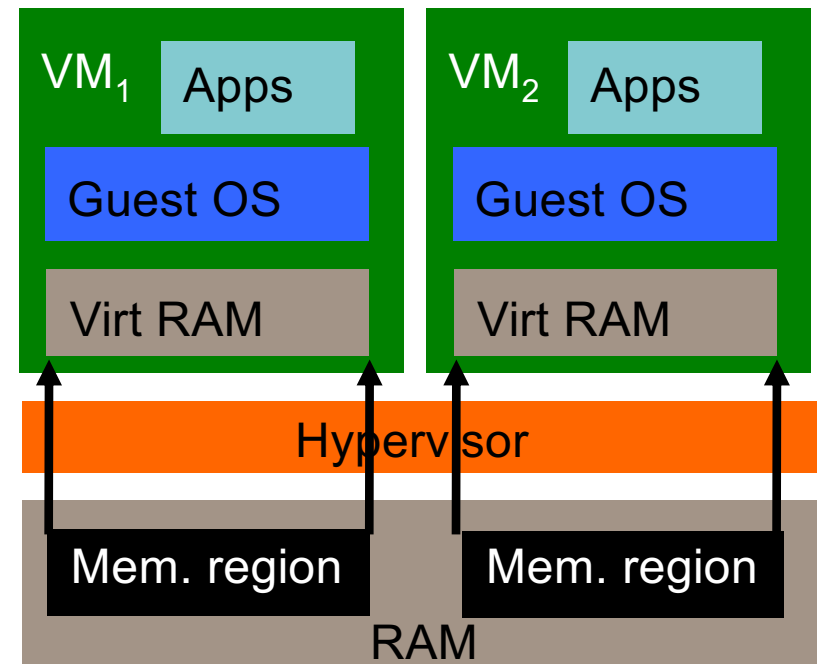
Obsolete
by 1980st



Why Virtual Machines?

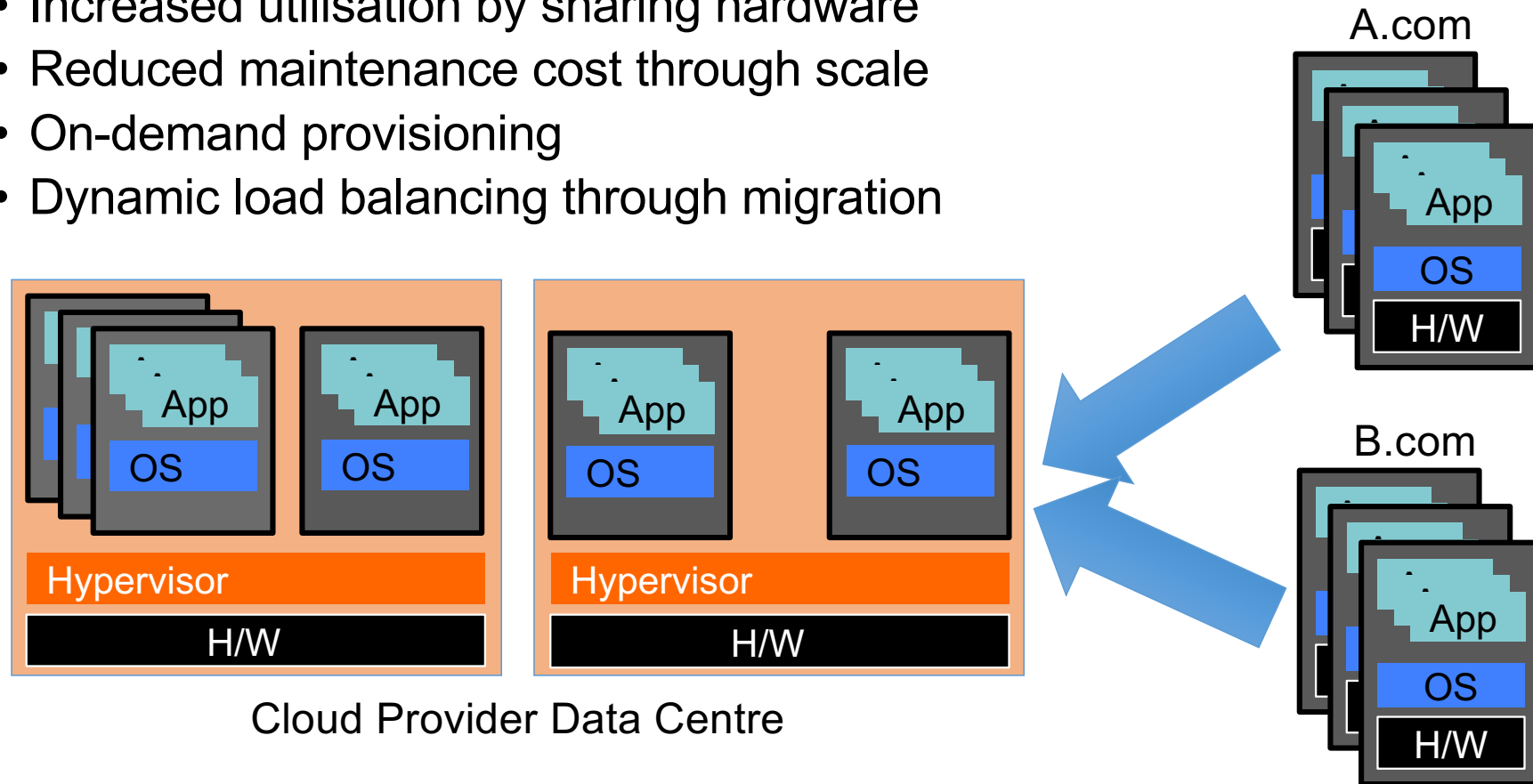
- Heterogenous concurrent guest OSes
 - eg Linux + Windows
- Improved isolation for consolidated servers: QoS & Security
 - total mediation/encapsulation:
 - replication
 - migration/consolidation
 - checkpointing
 - debugging
- Uniform view of hardware

Would not be needed if
OSes provided proper
security & resource
management!



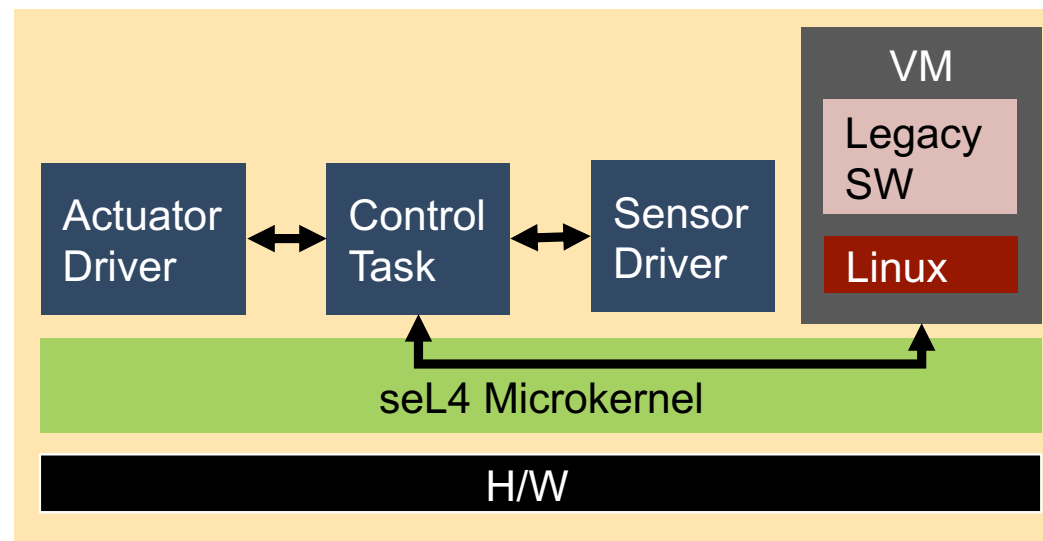
Why Virtual Machines: Cloud Computing

- Increased utilisation by sharing hardware
- Reduced maintenance cost through scale
- On-demand provisioning
- Dynamic load balancing through migration



Why Virtual Machines: Embedded Systems

- Integrate (untrusted) legacy stacks depending on Linux functionality
- Communicate with native components
- “Mixed-criticality system” (MCS)



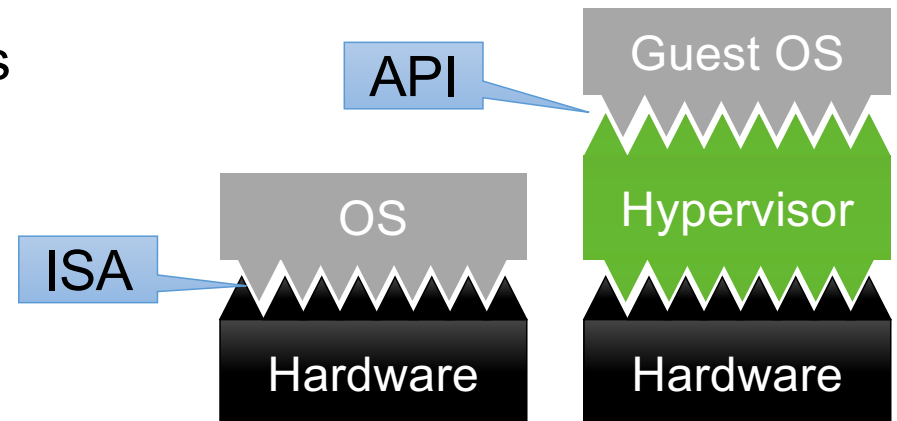
Hypervisor aka Virtual Machine Monitor

- Software layer that implements virtual machine
- Controls resources
 - Partitions hardware
 - Schedules guests
 - “*world switch*”
 - Mediates access to shared resources
 - e.g. console, network

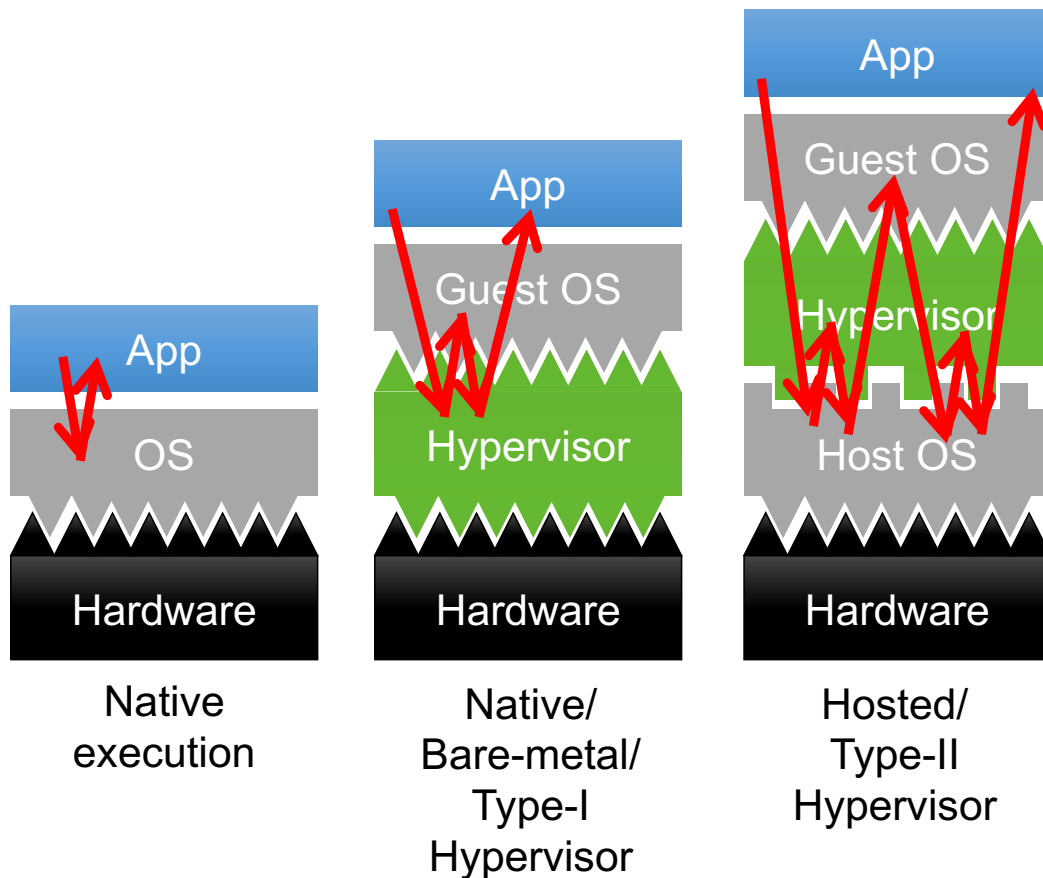
Implications:

- Hypervisor executes in *privileged* mode
- Guest OS executes in *unprivileged* mode

Privileged guest instructions
trap to hypervisor



Native vs Hosted Hypervisor



- Hosted VMM besides native apps
 - Sandbox untrusted apps
 - Convenient for running alternative OS on desktop
 - leverage host drivers

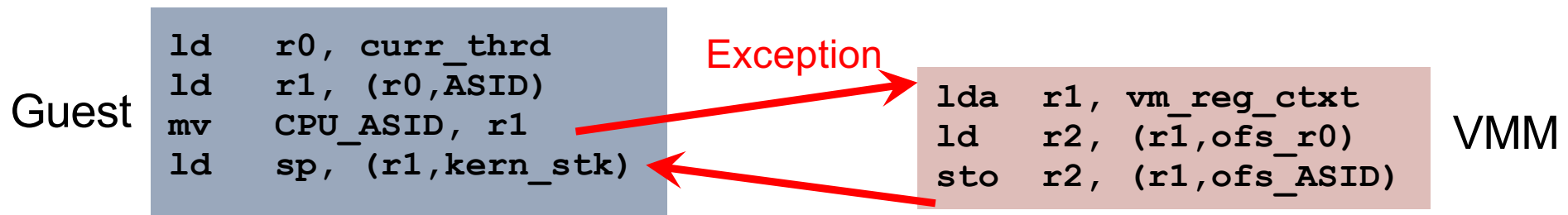
Overheads:

- Double mode switches
- Double context switches
- Host not optimised for exception forwarding

Virtualisation Mechanics

Instruction Emulation

- Traditional *trap-and-emulate* (T&E) approach:
 - guest attempts to access physical resource
 - hardware raises exception (trap), invoking HV's exception handler
 - hypervisor emulates result, based on access to virtual resource



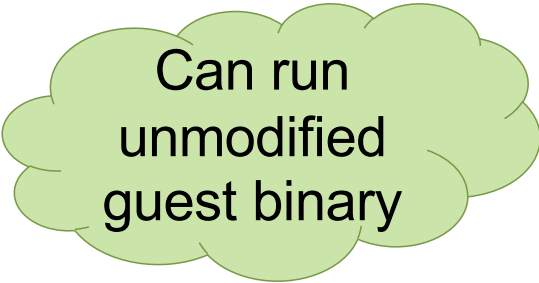
Most instructions do not trap

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

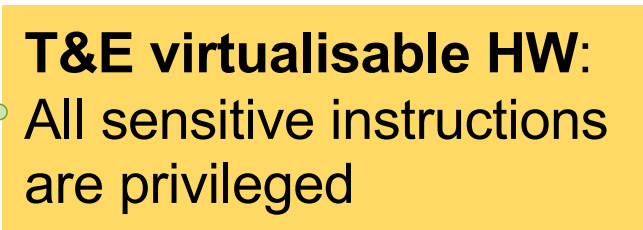
Trap & Emulate (T&E) Requirements

- **Privileged state:** determines resource allocation
 - Incl. privilege level, PT ptr, exception vectors...
- **Sensitive instruction:**
 - **control sensitive:** change privileged state
 - **behaviour sensitive:** expose privileged state
 - eg privileged instructions which NO-OP in user state
- **Innocuous instruction:** not sensitive
- **Privileged instruction:** when executed in user mode will *trap*

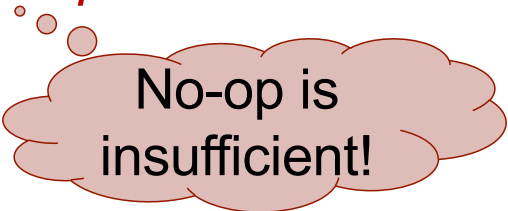
- Some inherently sensitive, e.g. set interrupt level
- Some context-dependent, e.g. store to page table



Can run
unmodified
guest binary



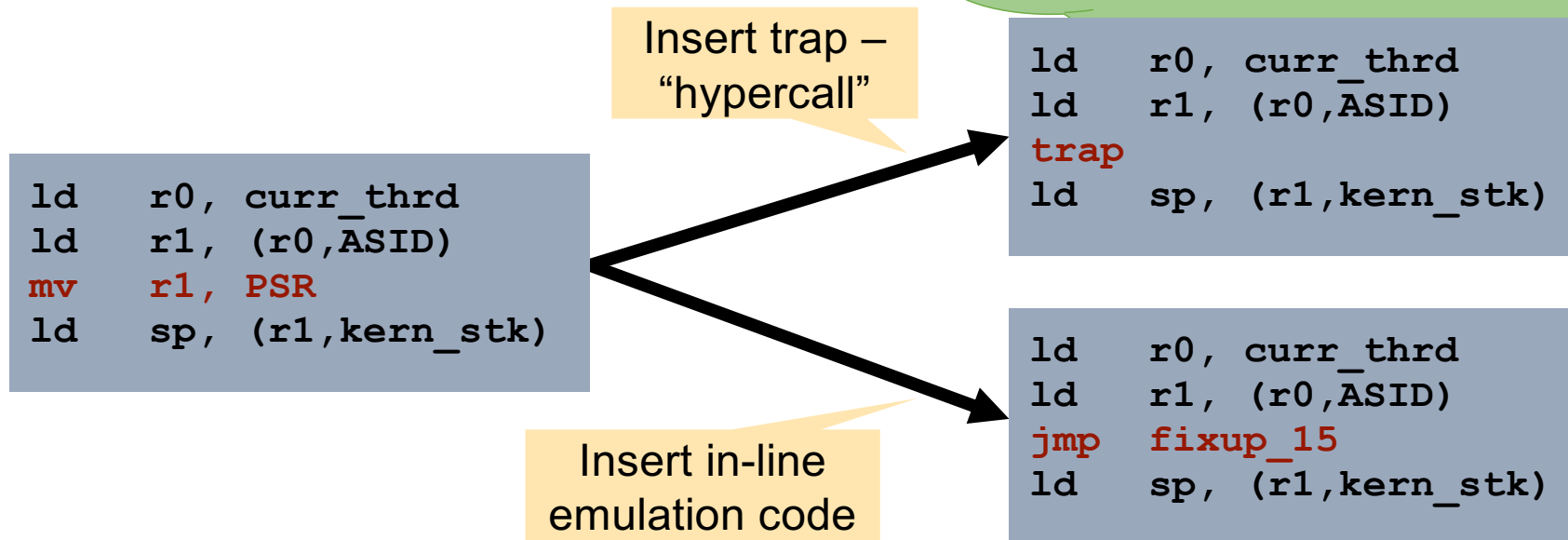
T&E virtualisable HW:
All sensitive instructions
are privileged



No-op is
insufficient!

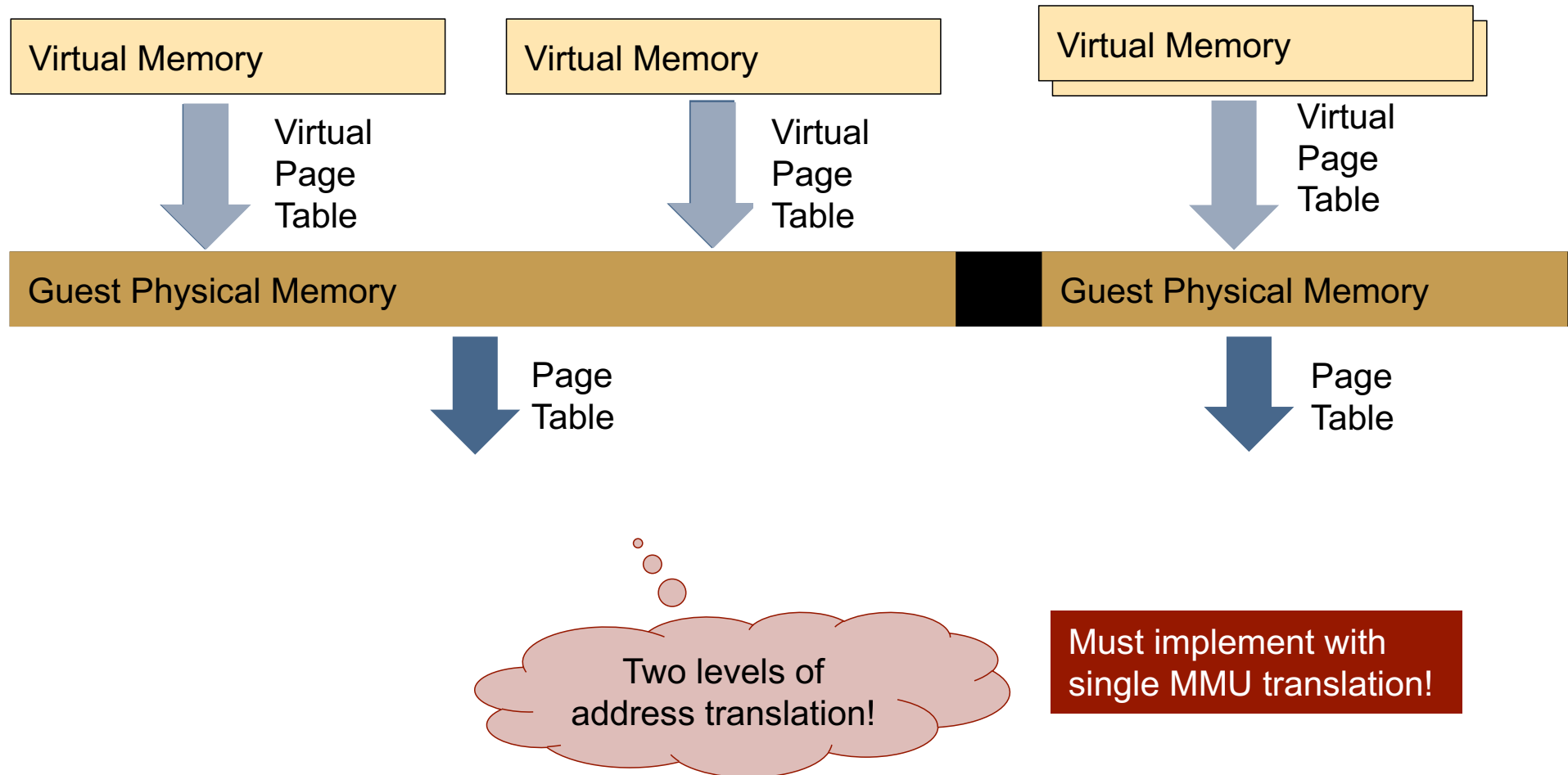
"Impure" Virtualisation

- Support non-T&E hardware
- Improve performance

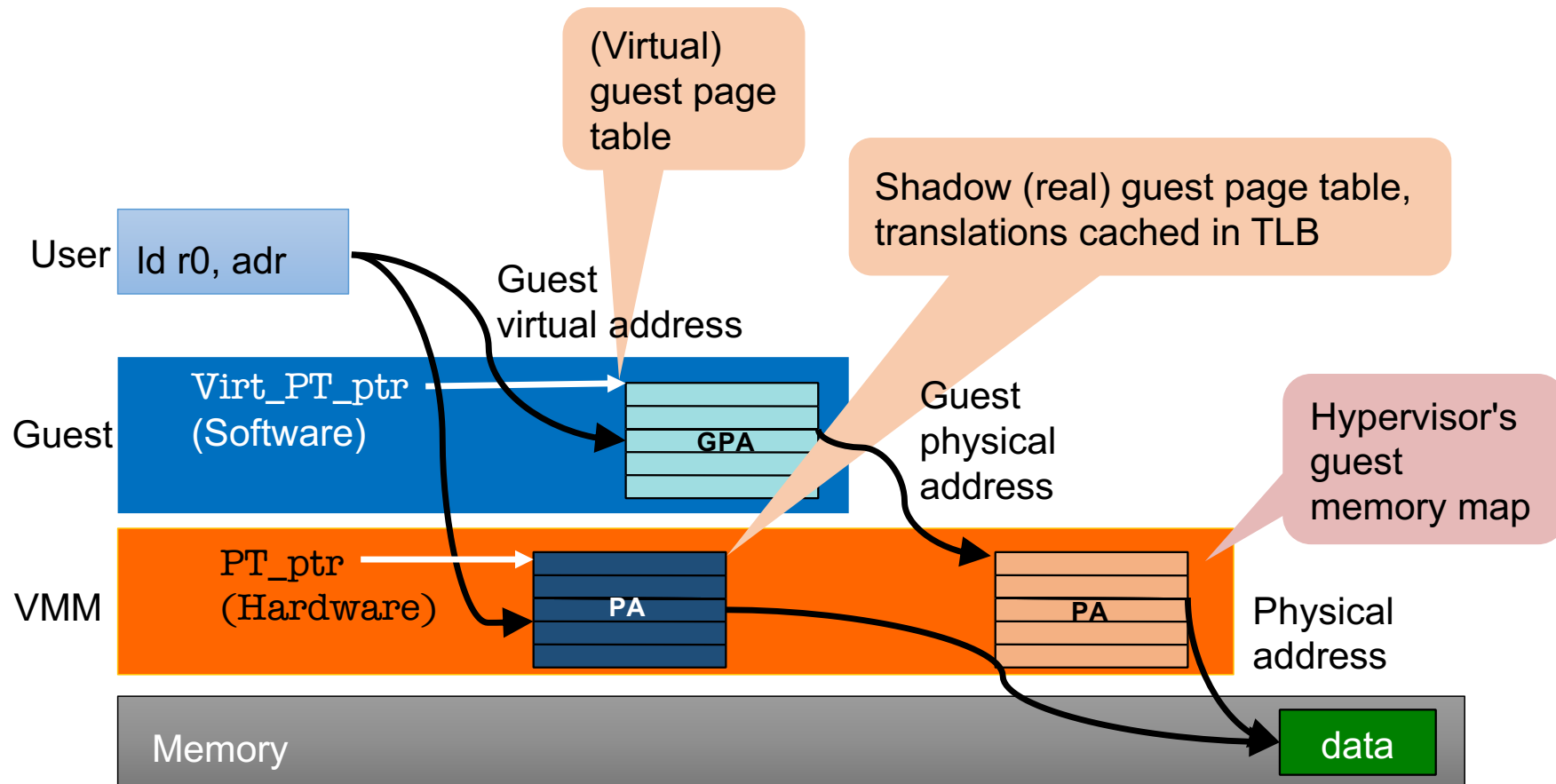


- Modify binary: *binary translation* (VMware)
- Modify hypervisor "ISA": *para-virtualisation*

Virtualisation vs Address Translation



Shadow Page Table



Shadow Page Table

Hypervisor must shadow (virtualize) PT updates by guest:

- trap guest writes to guest PT
- translate guest PA in guest (virtual) PTE using memory map
- insert translated PTE in shadow PT

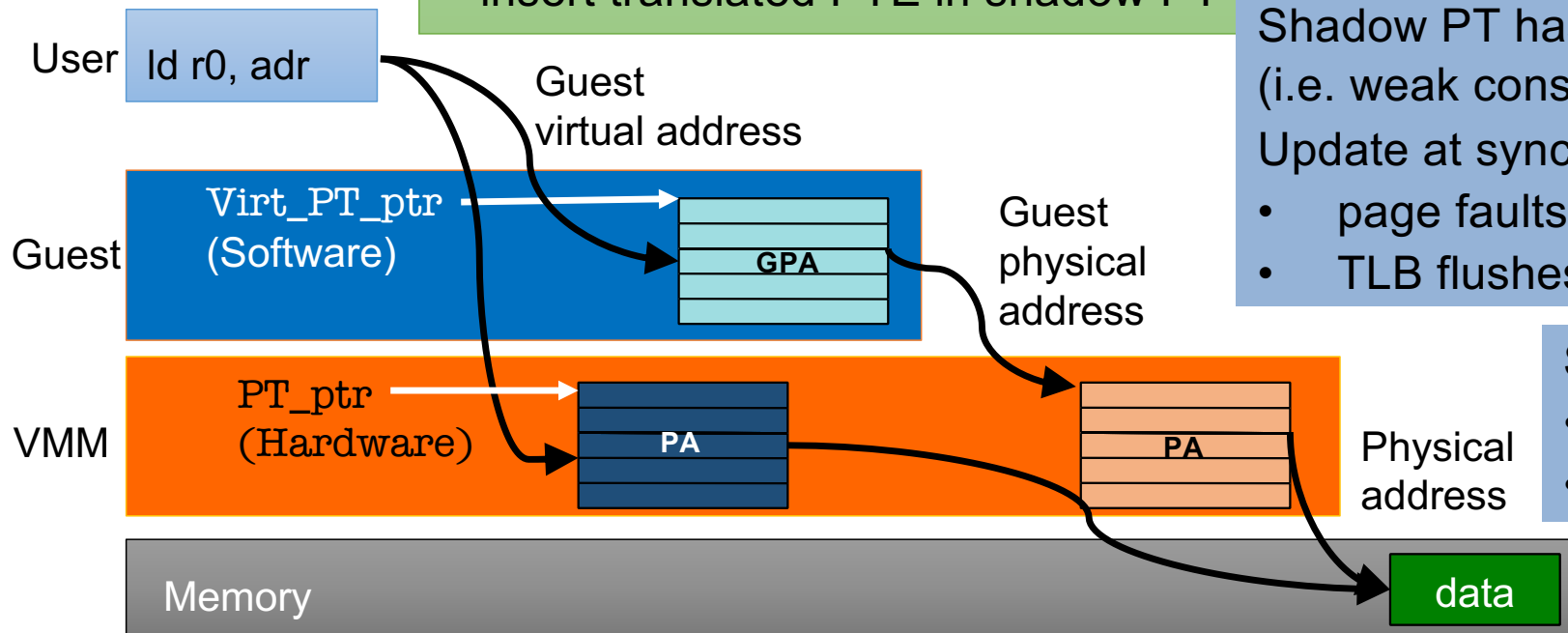
Shadow PT has TLB semantics (i.e. weak consistency) ⇒

Update at synchronisation points:

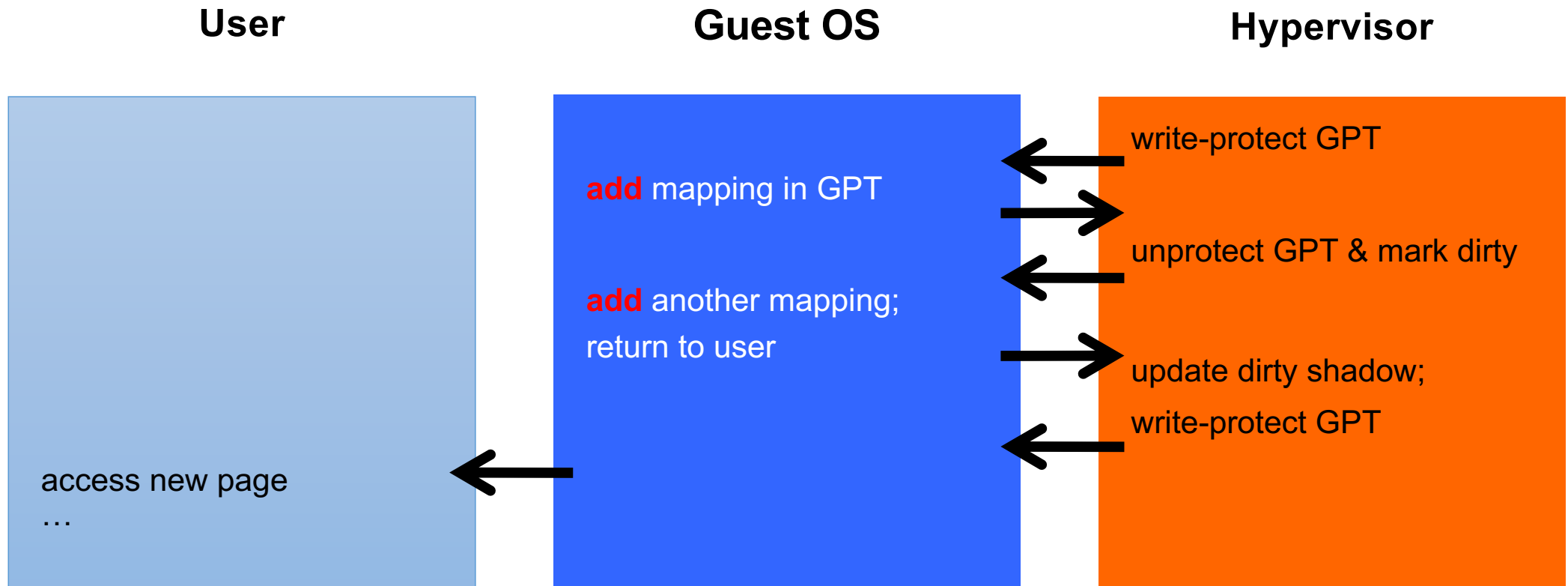
- page faults
- TLB flushes

SPT is a *virtual TLB*

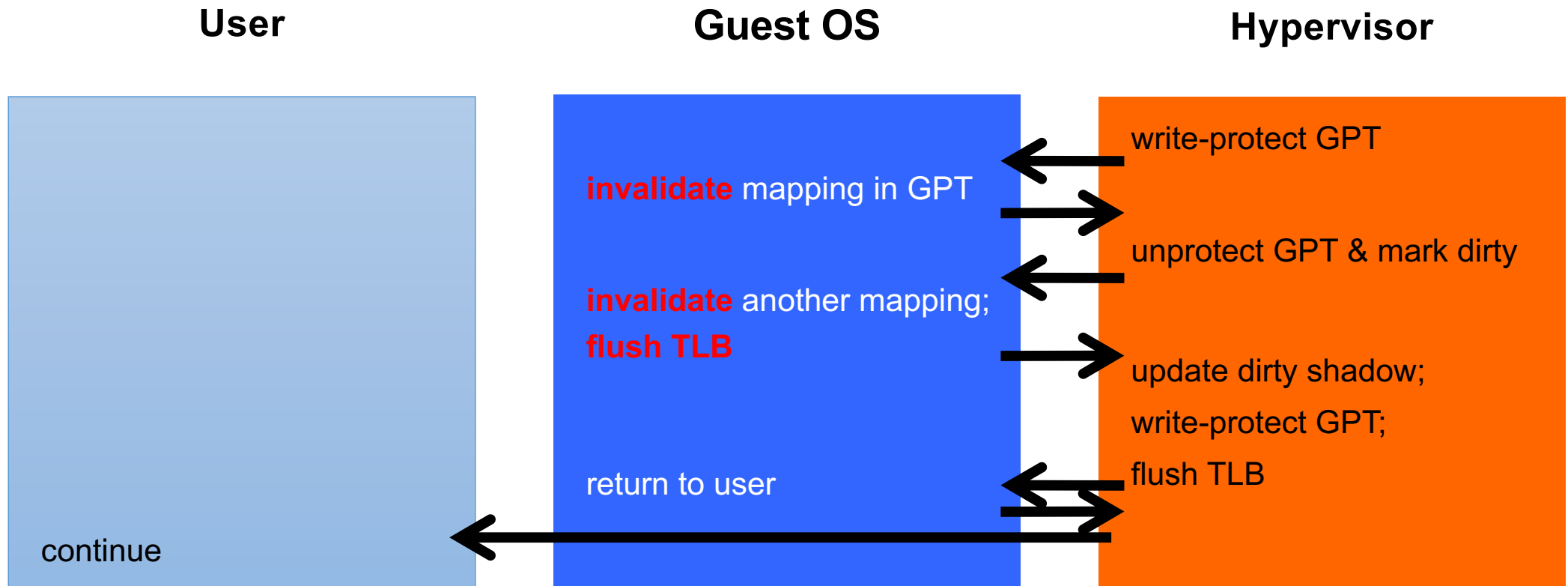
- similar semantics
- can be incomplete



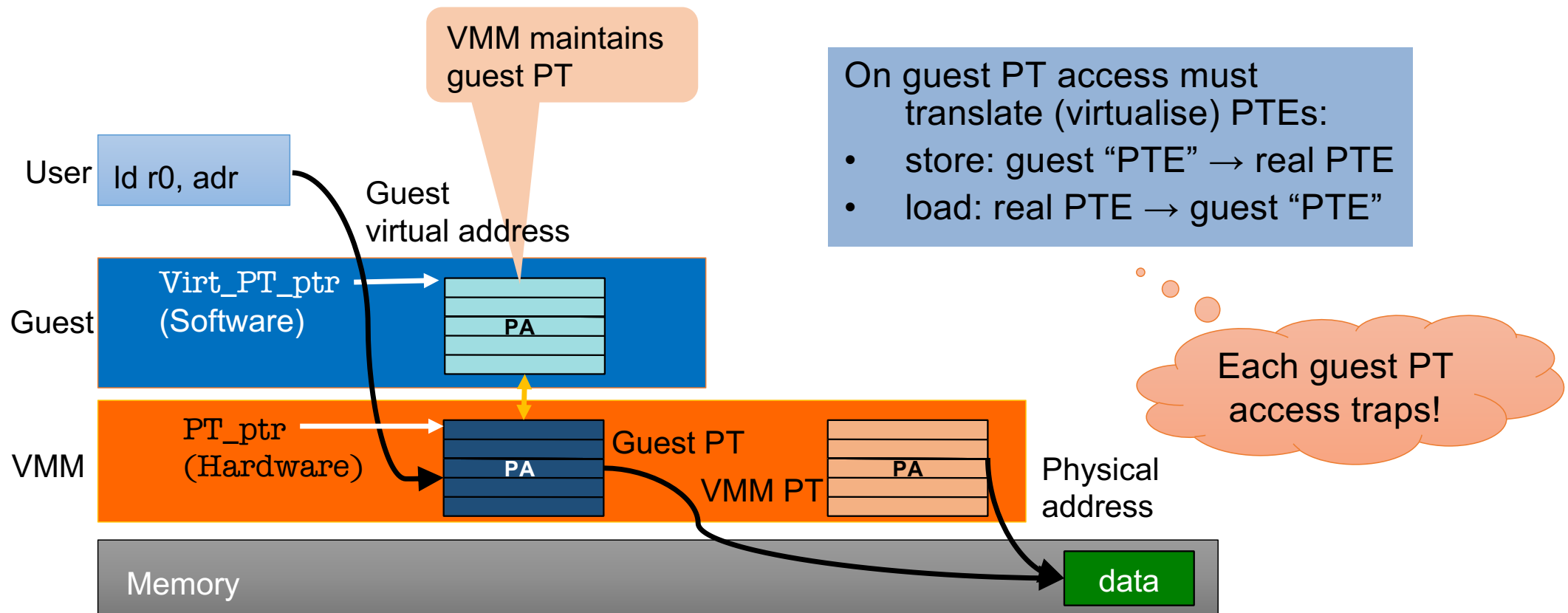
Lazy Shadow Update: Map



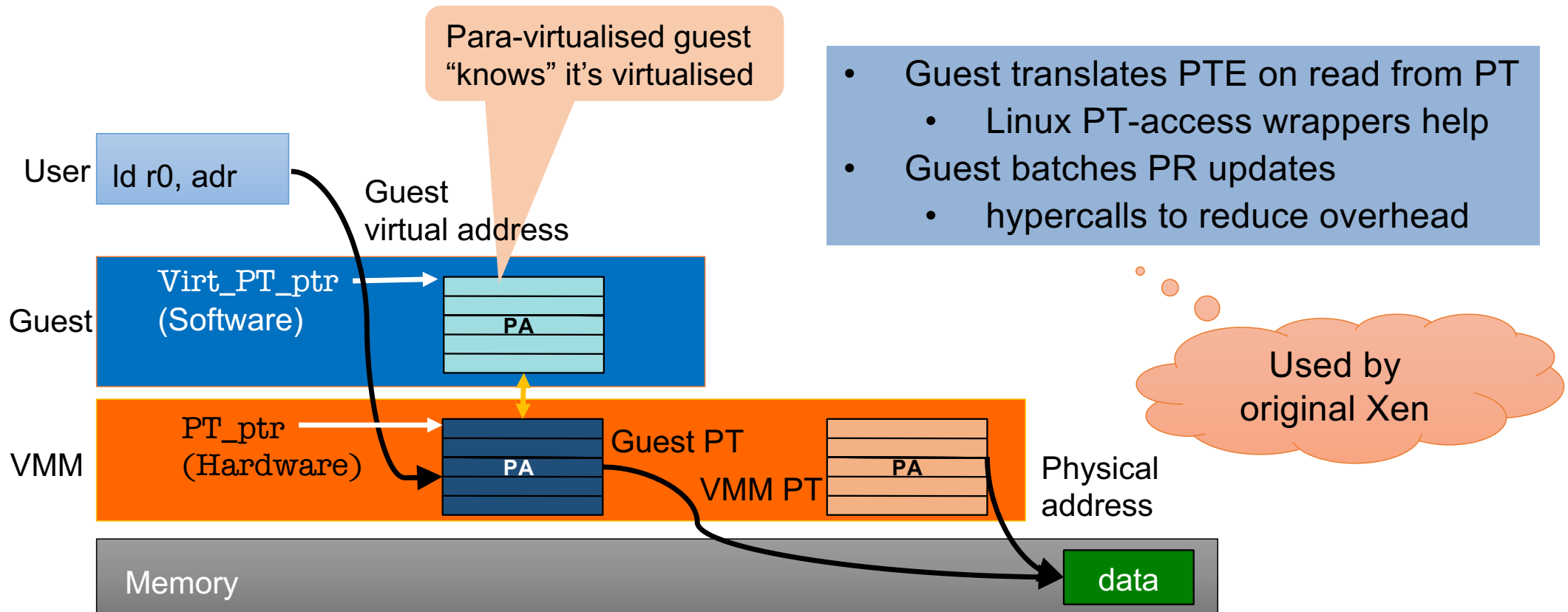
Lazy Shadow Update: Unmap



Real Guest Page Table



Optimised Guest Page Table

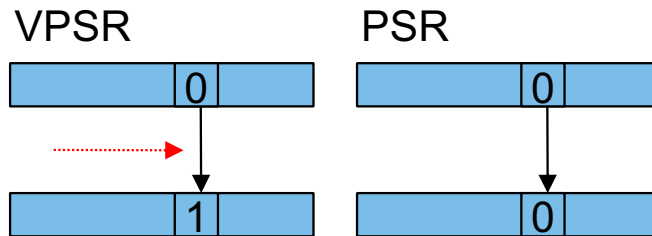


Guest Self-Virtualisation

Minimise traps by holding some virtual state inside guest

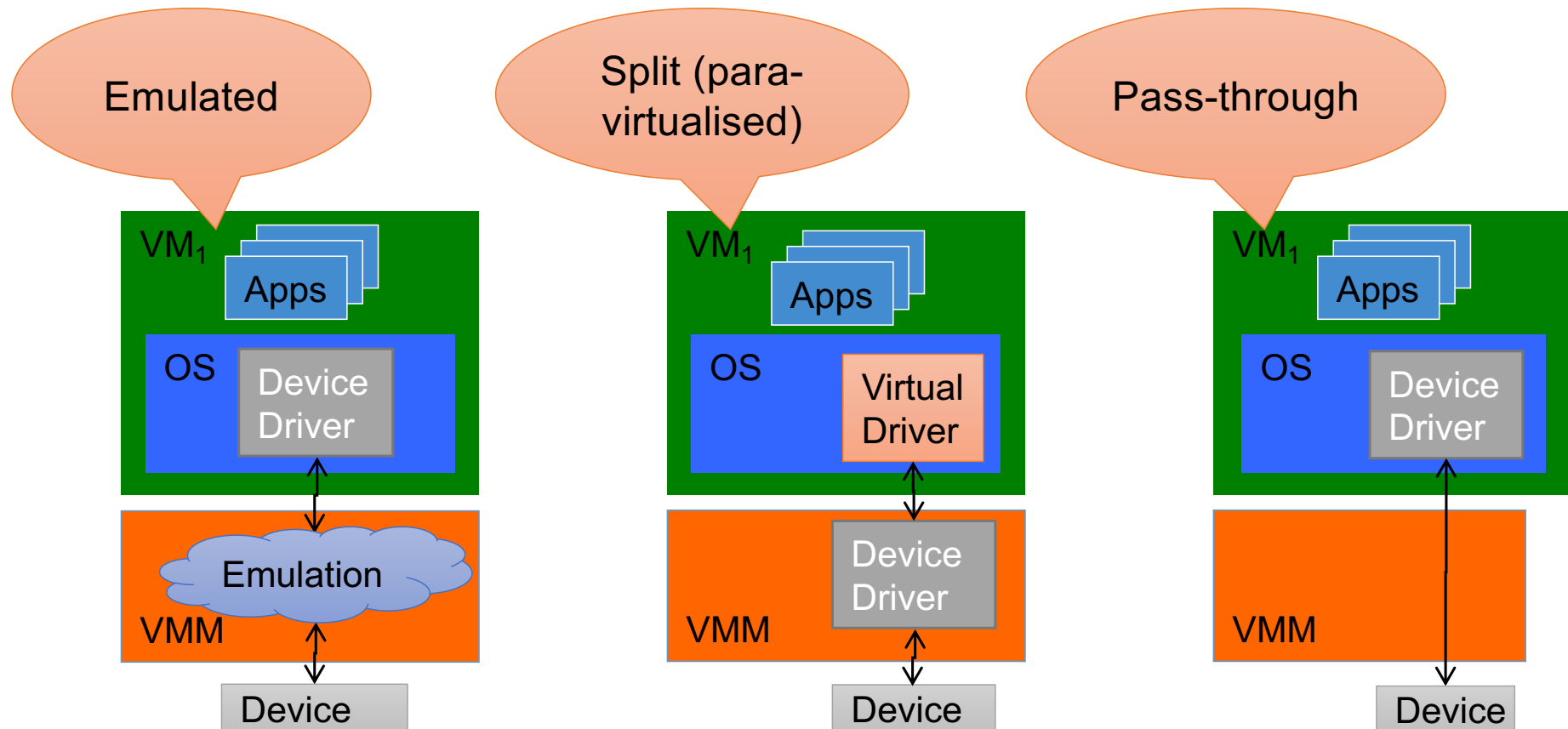
Example: Interrupt-enable in virtual PSR

- guest and VMM agree on VPSR location
- VMM queues guest IRQs when disabled in VPSR

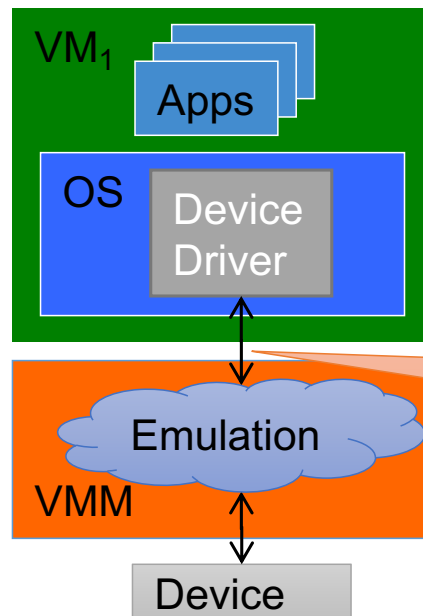


```
mov    r1, #VPSR
ldr    r0, [r1]
orr    r0, r0, #VPSR_ID
sto    r0, [r1]
```

Device Models



Emulated Device



Each device access must trap and be emulated

- unmodified native driver
- high overhead!
- may not actually work, as may violate device timing constraints

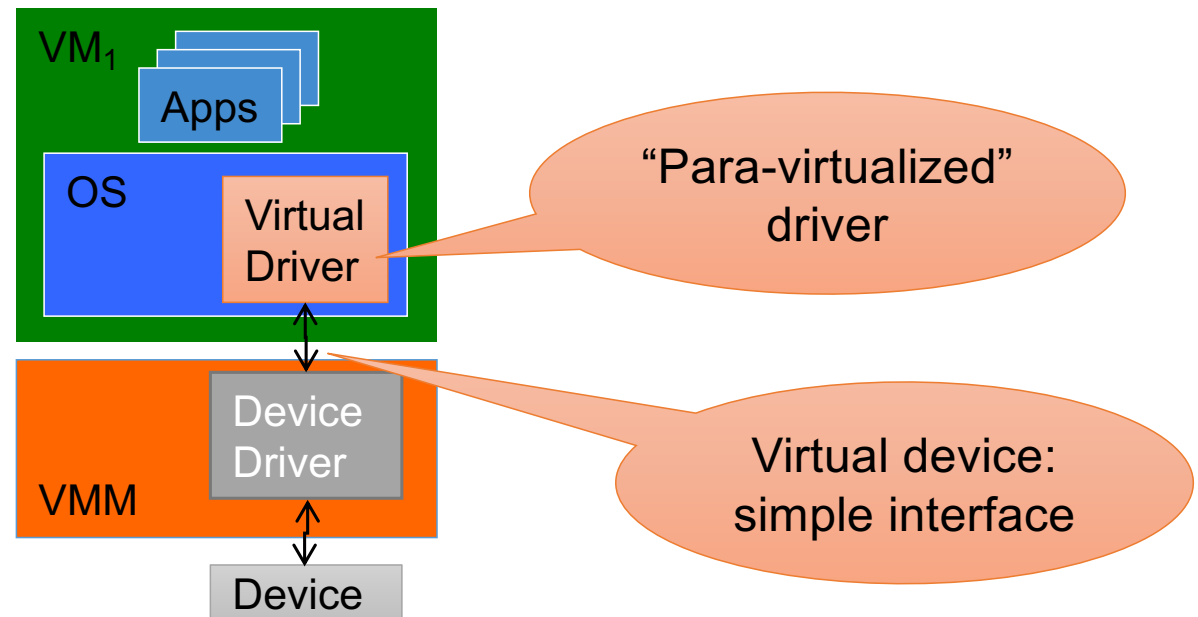
Device register accesses

Split Driver

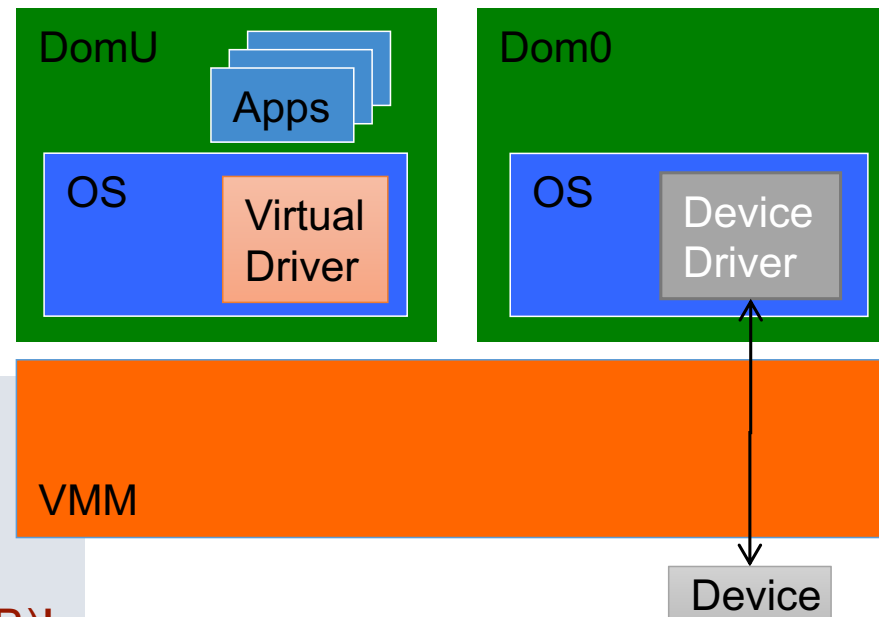
VirtIO: Linux I/O virtualisation interface

Simplified, high-level device interface

- small number of hypercalls
- new (but very simple) driver
- low overhead
- must port drivers to hypervisor



Driver OS (Xen Dom0)



Leverage native drivers

- no driver porting
- must trust complete driver guest!
- huge *trusted computing base (TCB)*!

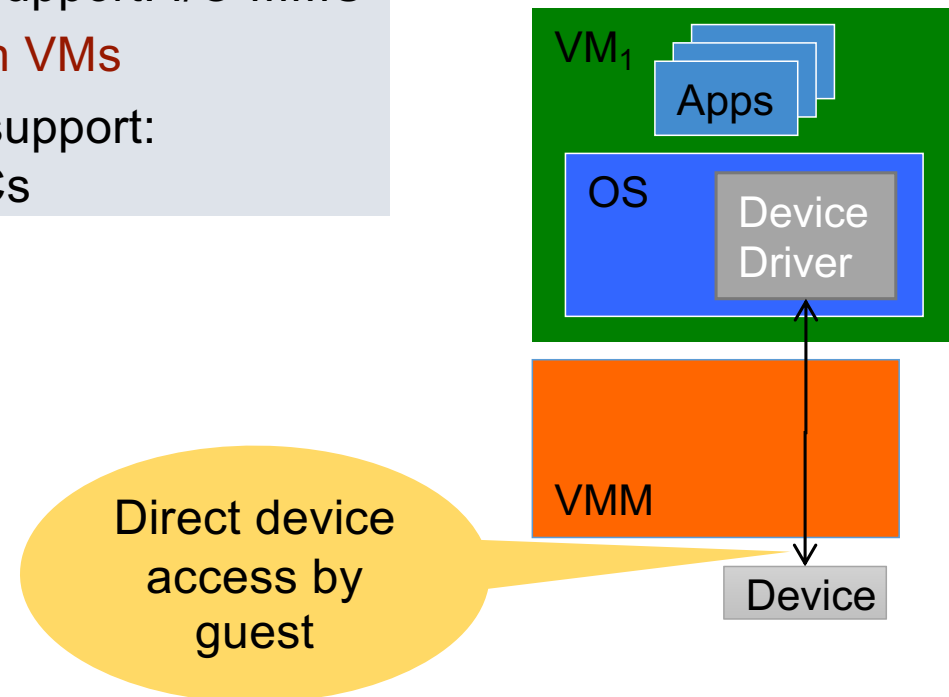
Pass-Through Driver

Unmodified native driver

- **Must trust driver (and guest) for DMA**
 - except with hardware support: I/O MMU
- **Can't share device between VMs**
 - except with hardware support: multi-queue recent NICs

“Self-virtualising” devices:

- Single-root I/O virtualisation (SRIOV)
- NIC presenting multiple, isolated virtual NIC interfaces



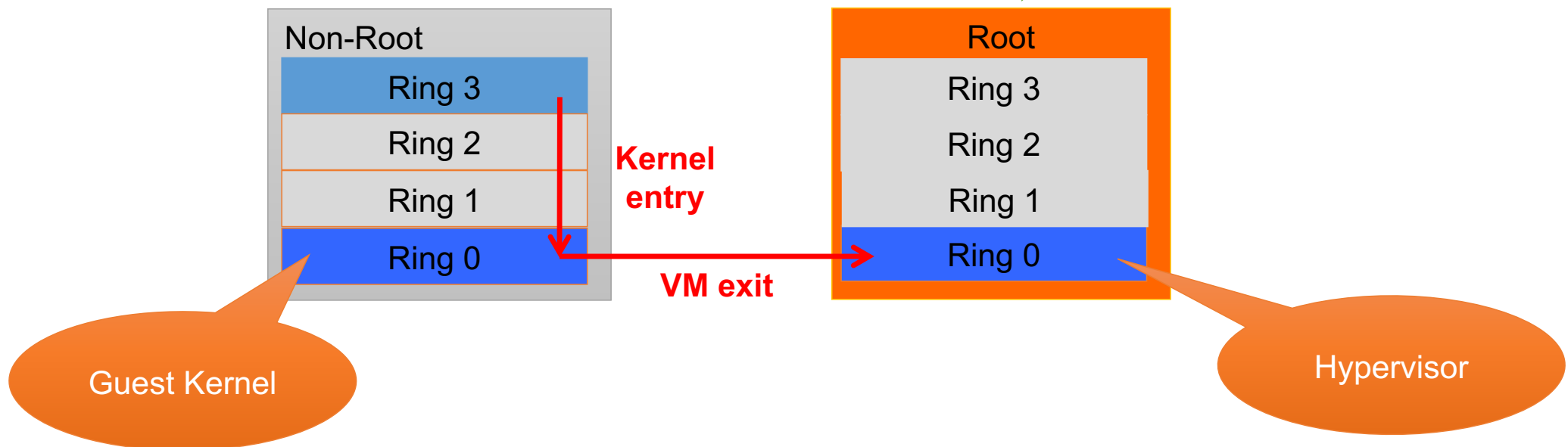
Modern Hardware Support

x86 Virtualisation Extensions: VT-x

New processor mode: VT-x root mode

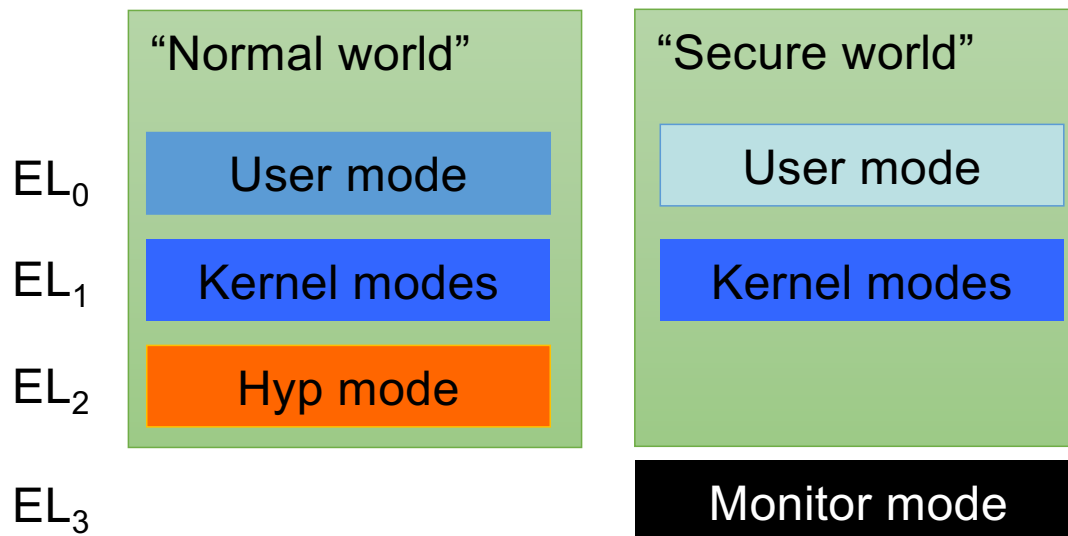
- orthogonal to protection rings
- entered on virtualisation trap

Traditional
x86 behaviour



Arm Virtualisation Extensions [1/6]

EL₂ aka “hyp mode”

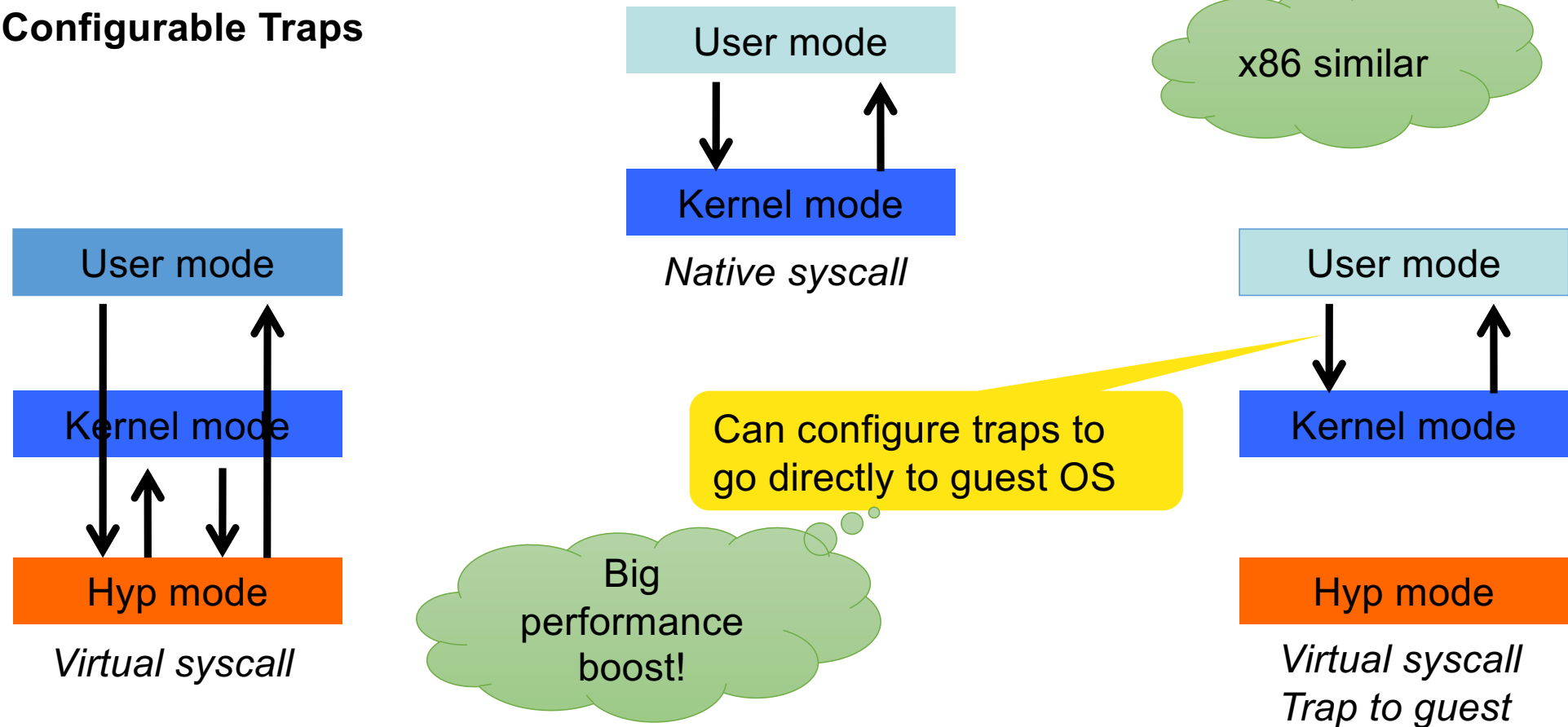


New privilege level

- Strictly higher than kernel (EL₁)
- Virtualizes or traps *all* sensitive instructions
- Originally only available in Arm TrustZone “normal world”
- Latest ISA revision supports it also in “secure world”

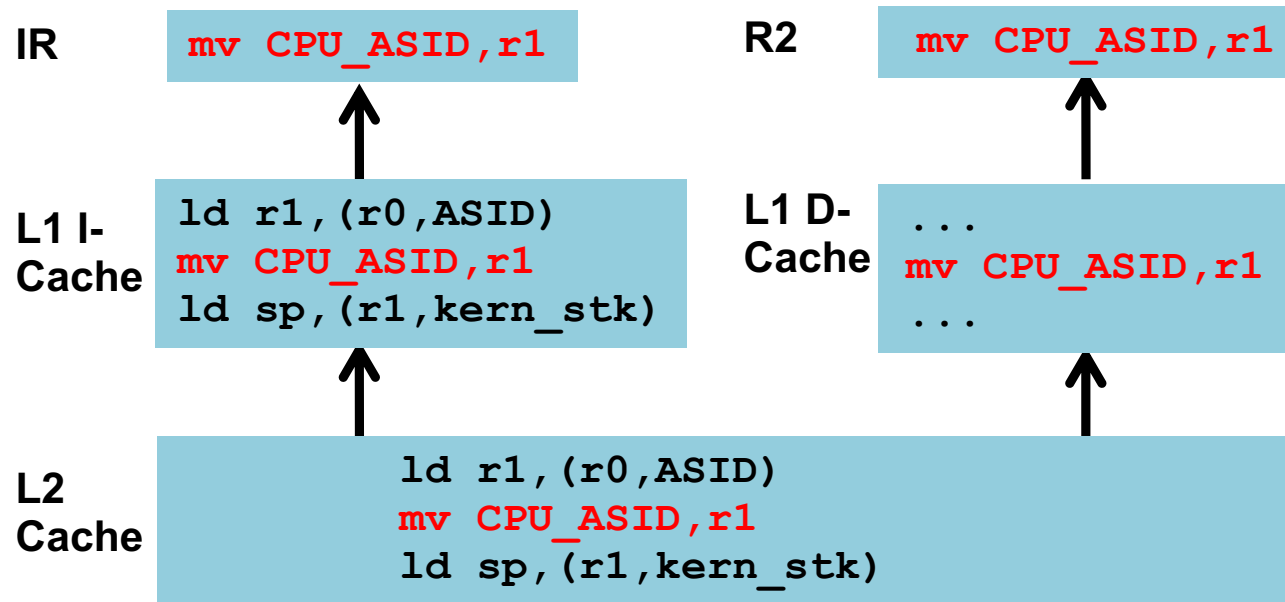
Arm Virtualisation Extensions [2/6]

Configurable Traps



Arm Virtualisation Extensions [3/6]

Emulation

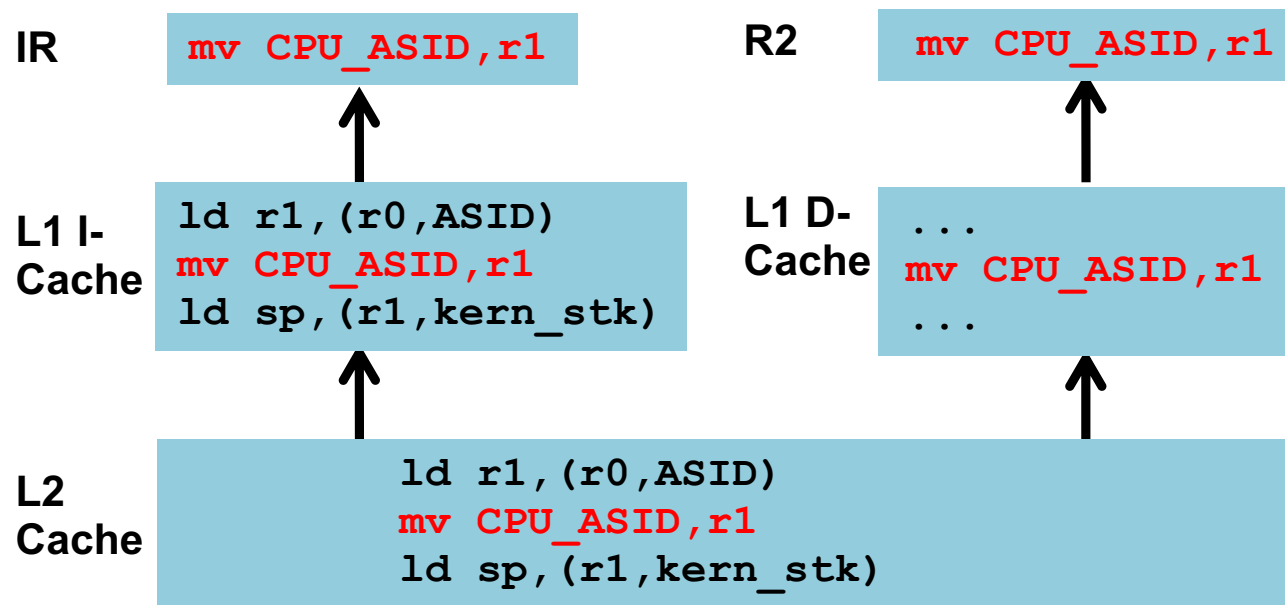


- 1) Load faulting instruction:
 - **Compulsory L1-D miss!**
- 2) Decode instruction
 - **Complex logic**
- 3) Emulate instruction
 - Usually straightforward

Arm Virtualisation Extensions [3/6]

Emulation

No x86
equivalent

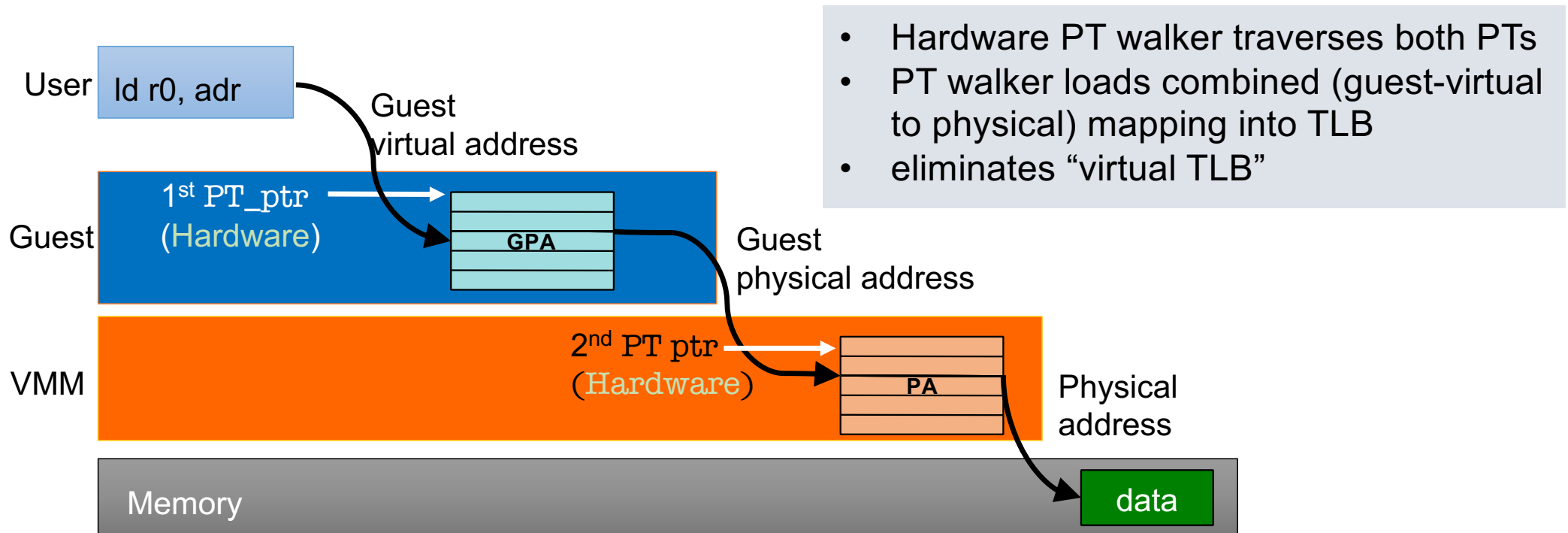


- 1) HW decodes instruction
 - No L1 miss
 - No software decode
- 2) SW emulates instruction
 - Usually straightforward

Arm Virtualisation Extensions (4)

2-stage translation

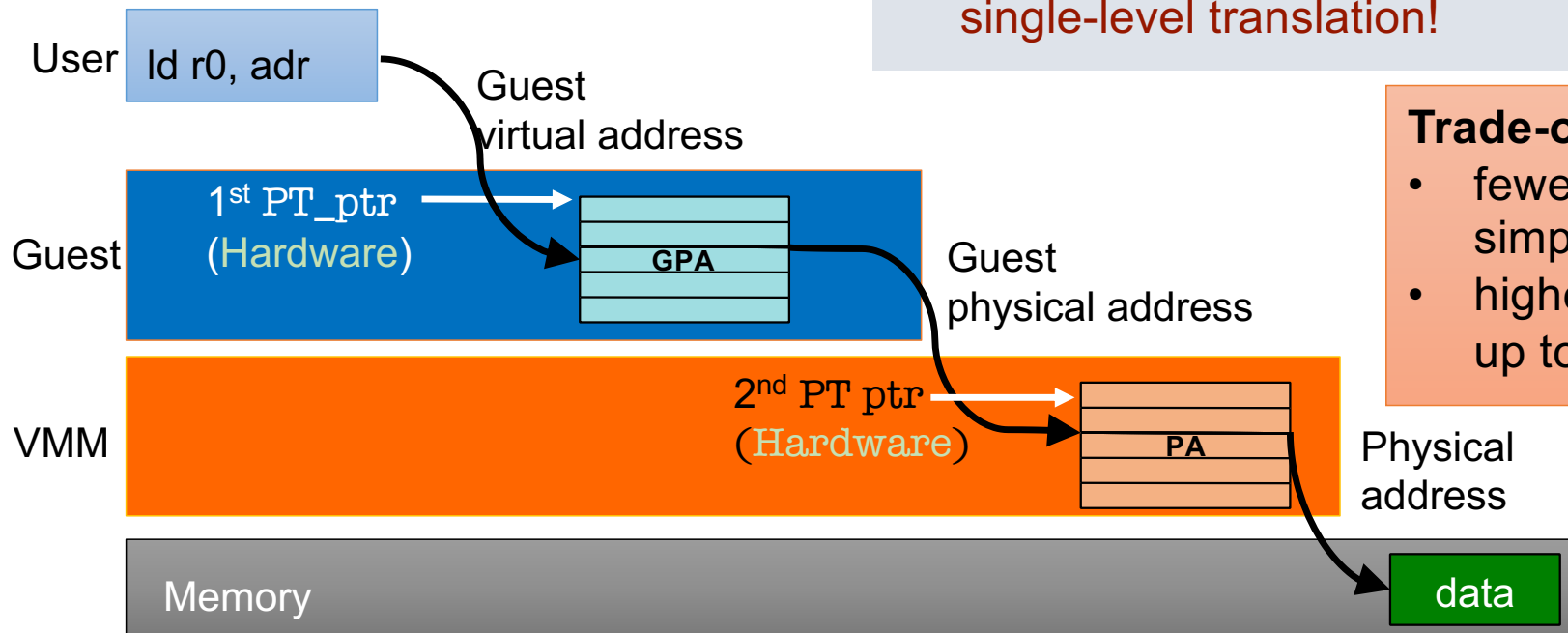
x86 similar
(EPTs)



Arm Virtualisation Extensions [4/6]

2-stage translation cost

- On page fault walk twice number of page tables!
- Can have a page miss on each, requiring PT walk
- $O(n^2)$ misses in worst case for n-level PT
- Worst-case cost is massively worse than for single-level translation!

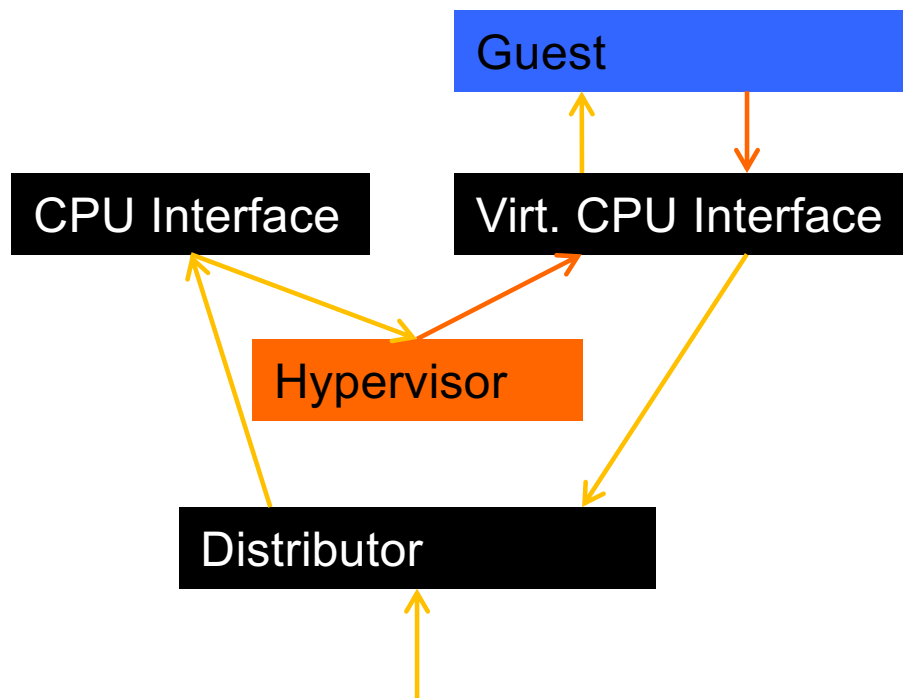


Trade-off:

- fewer traps
- simpler implementation
- higher TLB-miss cost up to 50% of run-time!

Arm Virtualisation Extensions [5/6]

Virtual Interrupts

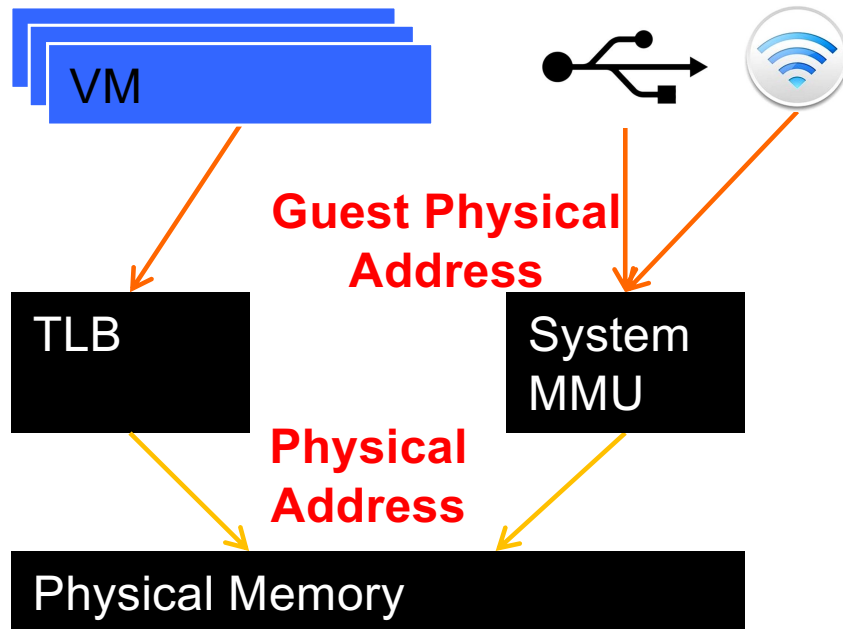


- 2-part IRQ controller
 - global “distributor”
 - per-CPU “interface”
- New H/W “virt. CPU interface”
 - Mapped to guest
 - Used by HV to forward IRQ
 - Used by guest to acknowledge
- Halves hypervisor invocations for interrupt virtualization

x86: similar for edge-triggered IRQs

Arm Virtualisation Extensions [6/6]

System MMU (I/O MMU)



- Devices use virtual addresses
- Translated by *system MMU*
 - elsewhere called IOMMU
 - translation cache, like TLB
 - reloaded from I/O page table

x86 different
(VT-d)

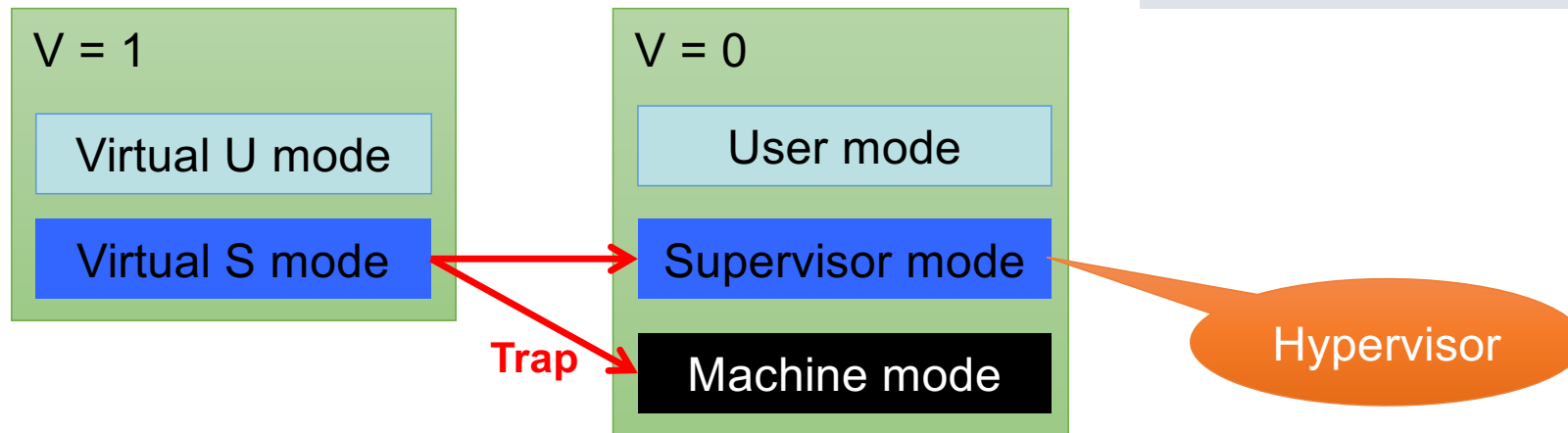
Many ARM
SoCs
different

- Can do pass-through I/O safely
 - guest accesses device registers
 - no hypervisor invocation

RISC-V H Extension

Add virtual U+S modes

- Extra registers for VM state
- Re-direct VS traps to S
- 2-stage address translation
- VIRQ injection



World Switch Comparison

x86

- VM state is ≈ 4 KiB
- Save/restore done by hardware on VMexit/VMentry
- Fast and simple

Arm

- VM state is 488 B
- Save/restore done by hypervisor
- Selective save/restore
 - Eg traps w/o world switch

RISC-V

- VM state ≈ 80 B
- Save/restore done by hypervisor
- Selective save/restore
 - Eg traps w/o world switch

