ECE 5984 Virtualization Technologies

Virtualization without Hardware Support

Pierre Olivier

Introduction

- **Some ISA fail to meet the constraints of the Popek & Goldberg theorem**
  - No VMM relying purely on direct execution + trap-and-emulate can be built on these ISA
- **Some systems circumvent these ISA limitation to build a VMM**
  - Trade-off some of Popek & Goldberg criteria (performance/equivalence/safety)
- **Disco (MIPS, 1997)**
- **Vmware Workstation (x86-32, 1999)**
- **Xen (x86-32, 2003)**
- **KVM for ARM (ARMv5/v6, 2010)**
  - Don't mix it up with ARM/KVM which is for ARM ISA versions with hardware support for virtualization!

Outline

1) Disco (MIPS, 1997)

2) Vmware Workstation (x86-32, 1999)

3) Xen (x86-32, 2003)

4) KVM/ARM (ARMv6/v7/v8, 2010)

Outline

1) Disco (MIPS, 1997)

2) Vmware Workstation (x86-32, 1999)

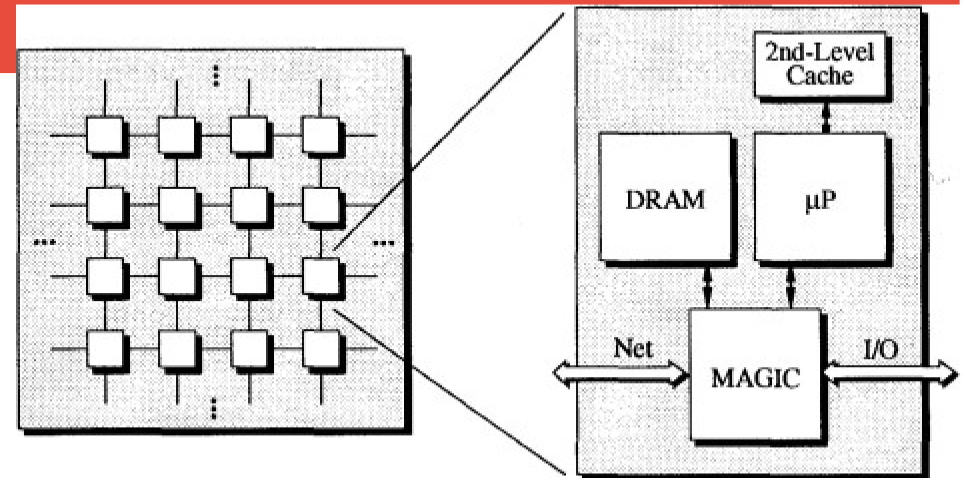
3) Xen (x86-32, 2003)

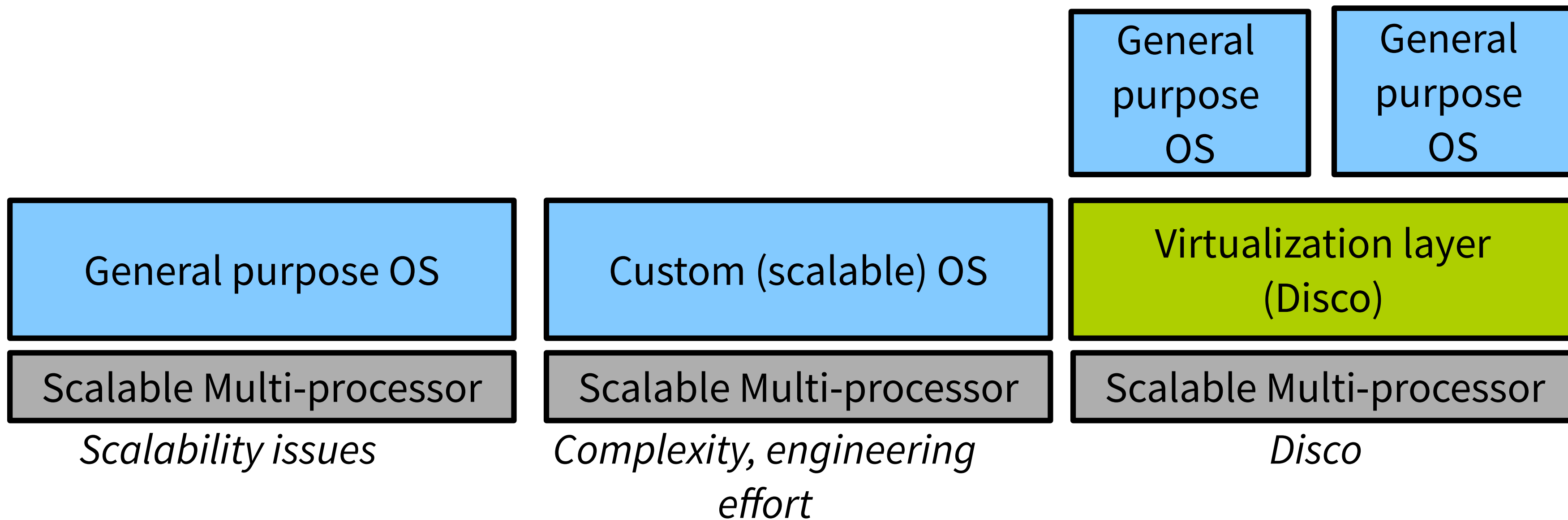
4) KVM/ARM (ARMv5/v6, 2010)

Disco

Kuskin, Jeffrey, et al. "The stanford flash multiprocessor." ACM SIGARCH Computer Architecture News. Vol. 22. No. 2. IEEE Computer Society Press, 1994.

- **Goal: run commodity OS on *scalable* multi-processors**
  - Stanford FLASH processor (MIPS)





Bugnion, Edouard, et al. "Disco: Running commodity operating systems on scalable multiprocessors." ACM Transactions on Computer Systems (TOCS) 15.4 (1997): 412-447.
Govil, Kinshuk, et al. "Cellular Disco: Resource management using virtual clusters on shared-memory multiprocessors." ACM SIGOPS Operating Systems Review. Vol. 33. No. 5. ACM, 1999.

Disco

Adherence to Popek & Goldberg criterias

- **Equivalence:**
  - Need to ***rewrite OS sources*** (IRIX) to relocate the kernel from KSEG0 to KSEG
    - It means *recompilation*

Region	Base	Length	Access K,S,U	MMU	Cache
USEG	0x0000 0000	2 GB	✓,✓ ✓	mapped	cached
KSEG0	0x8000 0000	512 MB	✓,x,x	unmapped	cached
KSEG1	0xA000 0000	512 MB	✓,x,x	unmapped	uncached
KSSEG	0xC000 0000	512 MB	✓,✓,x	mapped	cached
KSEG3	0xE000 0000	512 MB	✓,x,x	mapped	cached

Source: textbook

- **MIPS has 3 levels of execution: kernel (most privileged), supervisor , and user**
  - Disco hypervisor in kernel mode
  - Guest OS in supervisor mode (shifted from kernel mode)
  - Guest applications in user mode
  - That way, hypervisor is protected from the guest, and guest OS is protected from applications

Disco

Adherence to Popek & Goldberg criterias (2)

- **Safety**
  - Relies on virtual memory protection and MIPS execution modes (Kernel, Supervisor, User)
    - Protect VMM from VM
    - Protect guest OS from applications
- **Performance**
  - Most code should run directly and VMM handles trap efficiently → does not hold on MIPS
    - Special memory page to replace read-only privileged registers
    - Hypercalls
    - Larger TLB

Disco

MIPS privileged registers, Hypercalls

- **MIPS has privileged registers only accessible from kernel mode, that the guest OS expects to be able to manipulate**
  - 1 trap / register access is unacceptable
  - Frequently accessed privileged registers are set in dedicated special memory pages
    - Guest OS is rewritten to access this page rather than the original register
- **Disco implements *hypercalls***
  - Synchronous call from the rewritten guest OS to the VMM
    - Similar to an application making a *system call* to an OS
  - Ex: hypercall when the guests OS put a page in its free-list
    - Instructs the VMM that the guest is not using the page (for now) so it can be allocated to another VM

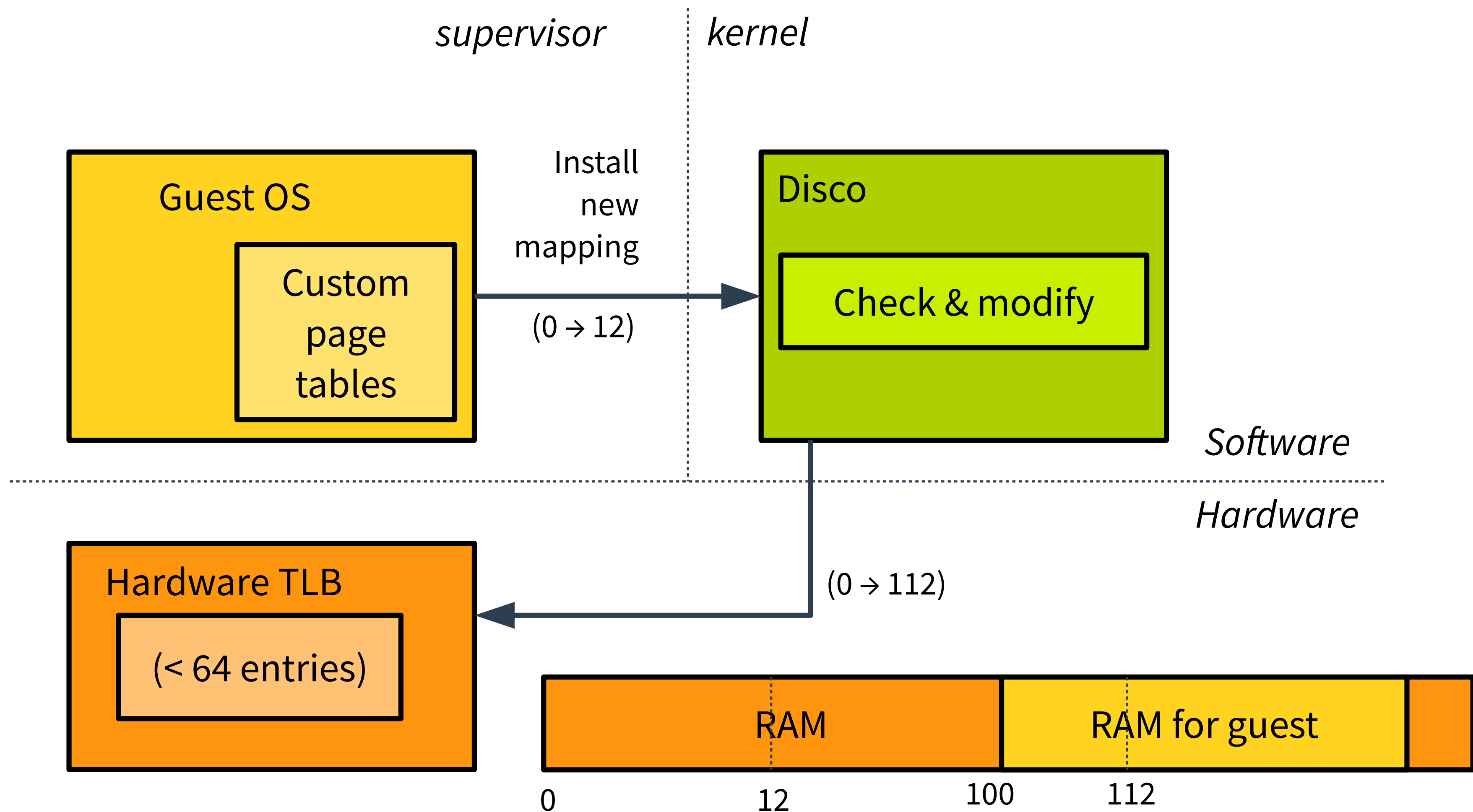
Disco

Memory virtualization: L2TLB

- **MIPS has a *software* TLB**
  - Managed by the OS: on TLB miss a handler is called an the OS is responsible for inserting the mapping into the TLB
    - OS defines its own page table format, walked in the handler to resolve and insert the mapping
  - When running with virtualization, a TLB miss traps to the hypervisor
    - The hypervisor cannot install the new mapping itself because it is unaware of the guest page table format
    - Hypervisor needs to call the guest OS TLB miss handler, which (tries to) install the mapping
      - ➔ Traps to the hypervisor for verification and remapping

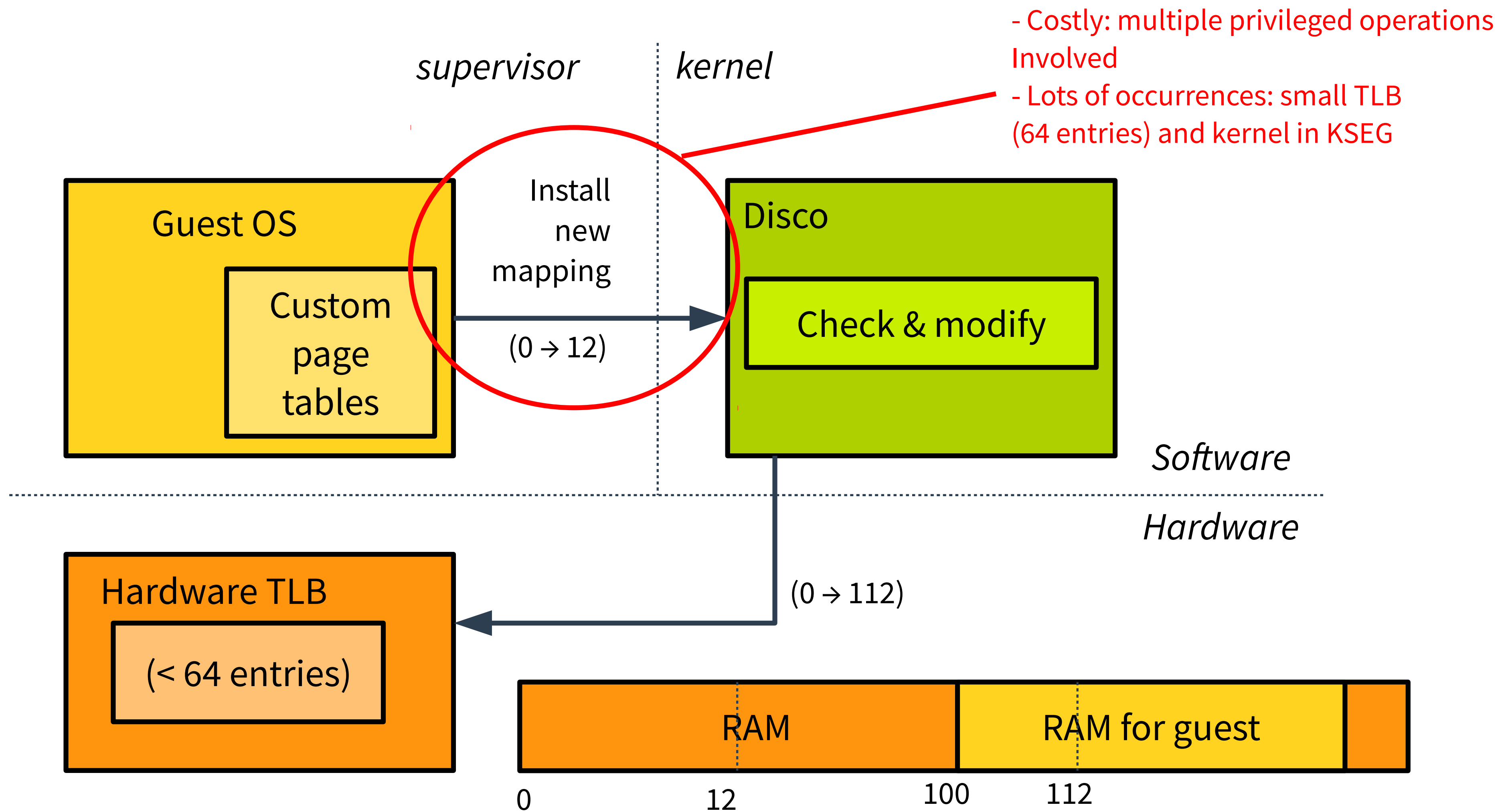
Disco

L2TLB (2)



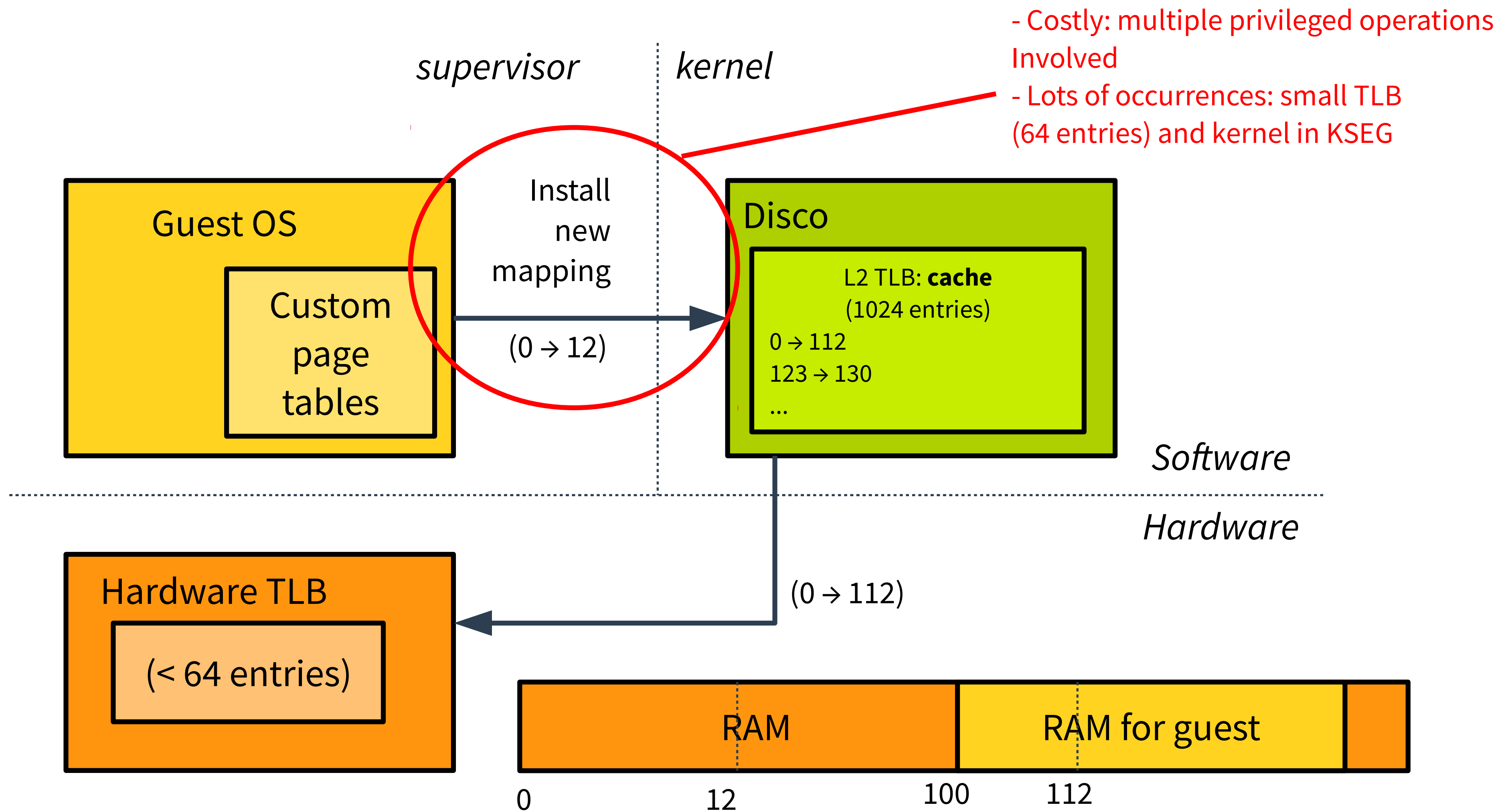
Disco

L2TLB (2)



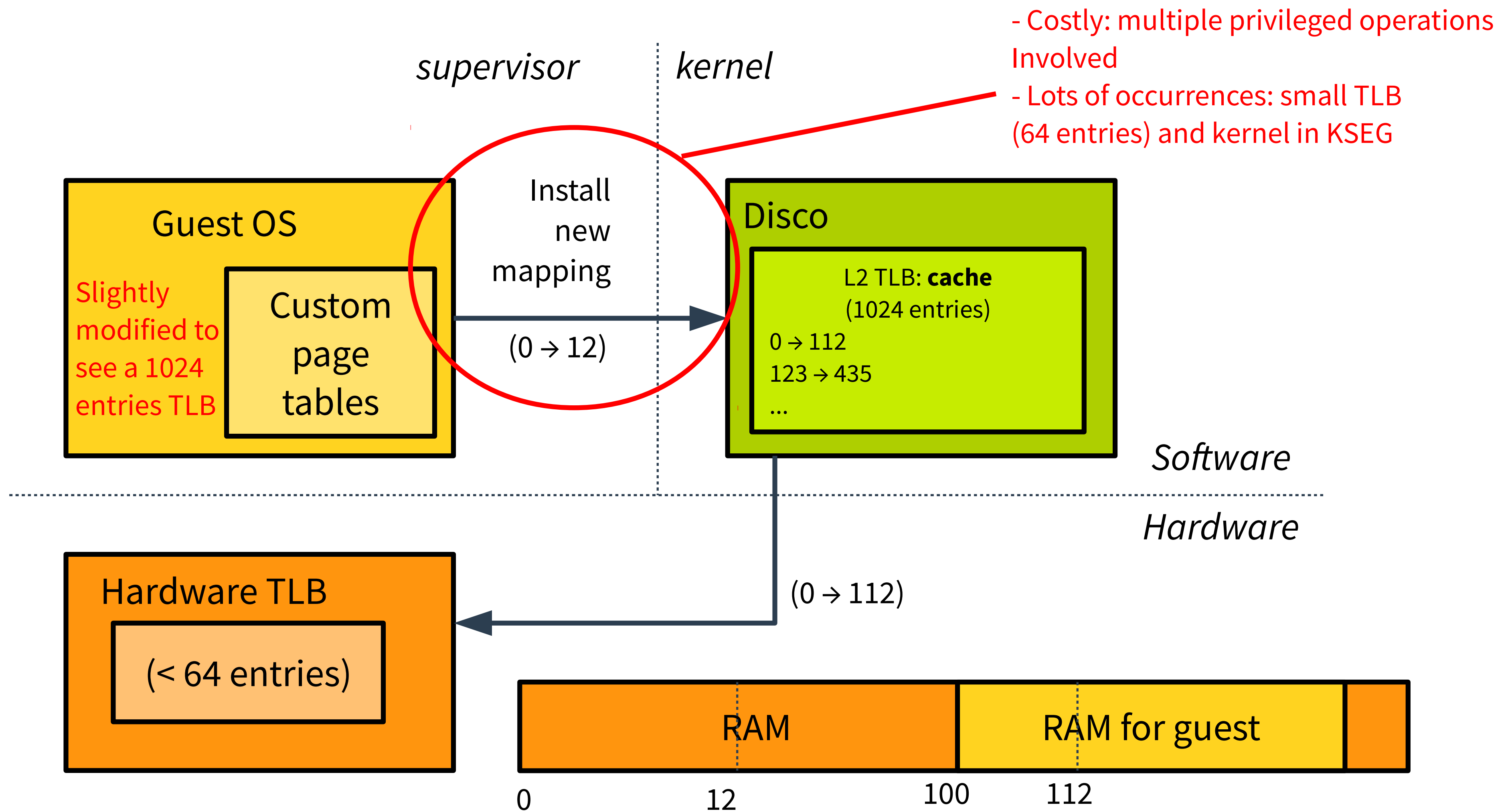
Disco

L2TLB (2)



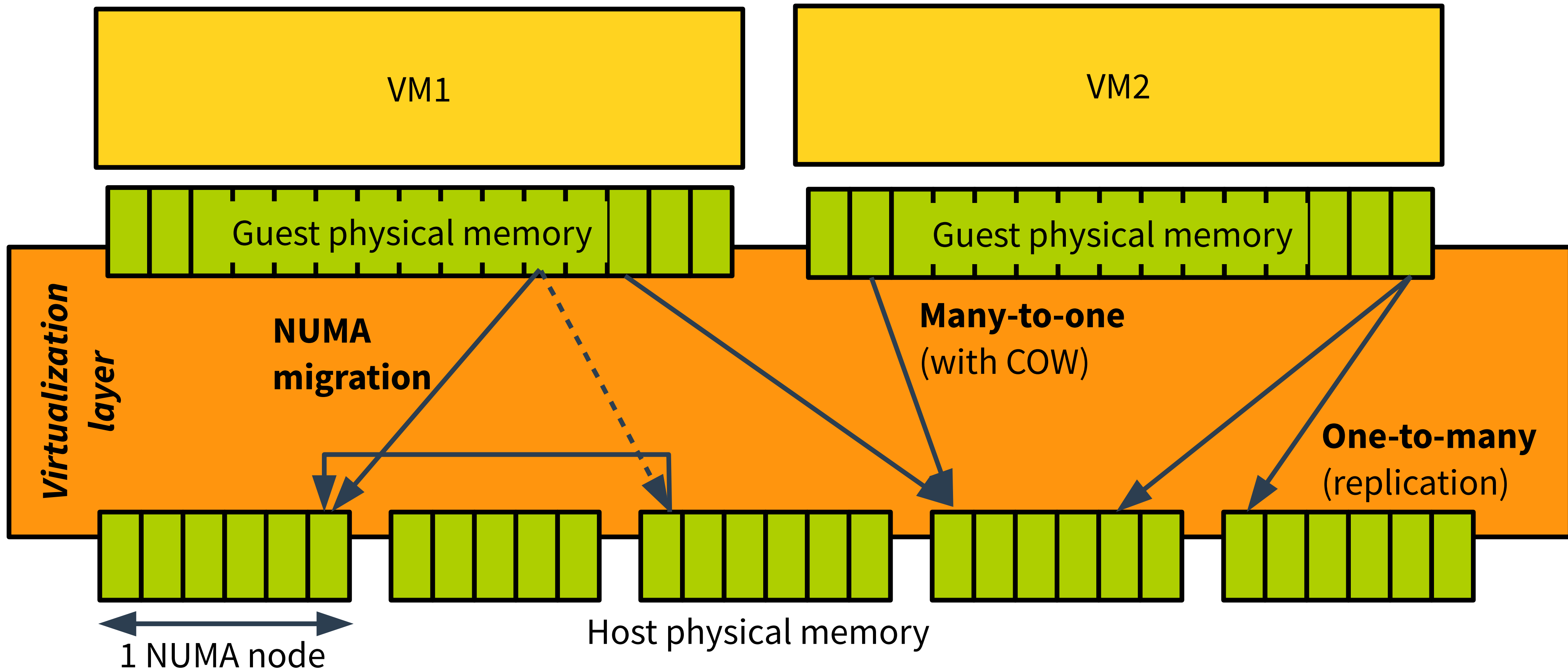
Disco

L2TLB (2)



Disco

Physical memory virtualization



All done in a ***completely transparent way*** *from the guest OS point of view*
- Enhanced scalability for commodity OS

Outline

1) Disco (MIPS, 1997)

2) Vmware Workstation (x86-32, 1999)

3) Xen (x86-32, 2003)

4) KVM for ARM (ARMv5/v6, 2010)

Vmware Workstation

- **Goal: running *totally unmodified* OS on x86-32**
  - Using a type-II (hosted) hypervisor with Linux or Windows host OS
- **Equivalence**
  - X86-32 fails P&G theorem
  - Hybrid virtualization: Vmware workstation combines direct execution with ***dynamic binary translation*** (fast emulation/interpretation)
- **Safety**
  - Use ***segment truncation*** for isolation
  - Focus on a subset of x86-32 instruction to run specific guest OS
- **Performance**
  - Goal: run near native speed, worst case same performance as previous generation CPUs
- **VMWare offers *full equivalence at the cost of performance* (DBT, MMU virtualization)**

Vmware Workstation

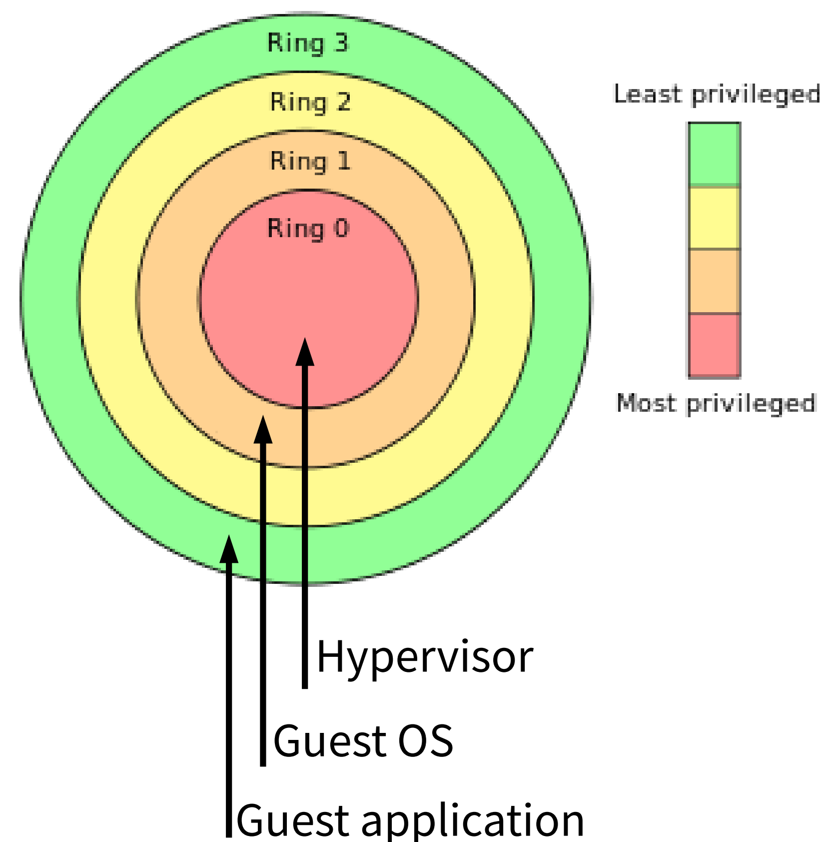
X86-32

- **Native execution mode: *protected mode***
  - Also legacy execution modes real, system management, v8086
- **In protected mode, *current privilege level* (cpl)**
  - Kernel: %cpl = 0
  - User: %cpl = 3
  - Iopl bits in FLAGS register optionally enable user code to disable interrupts
- **Implements both segmentation and paging**
  - Code (%cs), stack (%ss), data (%ds) and extra (%es, %fs, %gs) segment registers
  - 3 level page tables rooted at %cr3 with hardware TLB
  - Logical address → [segmentation] → linear address → [paging] → physical address

Vmware Workstation

X86-32

Source: wikipedia



- **With Vmware:**
  - Hypervisor runs in ring 0
  - Guest OS runs in ring 1, *de-privileged from ring0*
  - Application runs in ring 3
    - *Guest OS is protected from applications through the page table access bits*

Vmware Workstation

When to use direct execution, when to interpret

Input: Current state of the virtual CPU
Output: True if the direct execution subsystem may be used;
False if binary translation must be used instead

```
if !cr0.pe then
    return false;
end if
if eflags.v8086 then
    return true;
end if
if (eflags.iopl >= cpl) || (!eflags.if) then
    return false;
end if
/* a few corner cases here */
return true;
```

Vmware Workstation

When to use direct execution, when to interpret

Input: Current state of the virtual CPU
Output: True if the direct execution subsystem may be used;
False if binary translation must be used instead

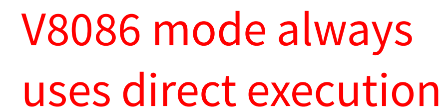
```
if !cr0.pe then
        return false;
end if
if eflags.v8086 then
        return true;
end if
if (eflags.iopl >= cpl) || (!eflags.if) then
        return false;
end if
/* a few corner cases here */
return true;
```

Vmware Workstation

When to use direct execution, when to interpret

Input: Current state of the virtual CPU
Output: True if the direct execution subsystem may be used;
False if binary translation must be used instead

```
if !cr0.pe then
    return false;
end if
if eflags.v8086 then
    return true;
end if
if (eflags.iopl >= cpl) || (!eflags.if) then
    return false;
end if
/* a few corner cases here */
return true;
```

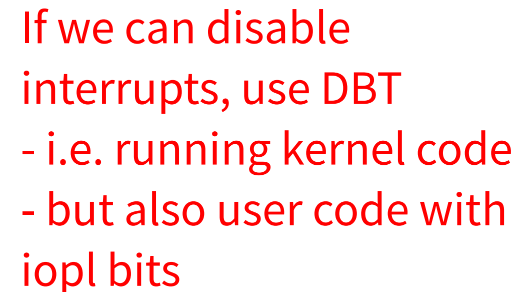


Vmware Workstation

When to use direct execution, when to interpret

Input: Current state of the virtual CPU
Output: True if the direct execution subsystem may be used;
False if binary translation must be used instead

```
if !cr0.pe then
    return false;
end if
if eflags.v8086 then
    return true;
end if
if (eflags.iopl >= cpl) || (!eflags.if) then
    return false;
end if
/* a few corner cases here */
return true;
```



Vmware Workstation
Dynamic Binary Translation

Input: Current state of the virtual CPU
Output: True if the direct execution subsystem may be used;
False if binary translation must be used instead

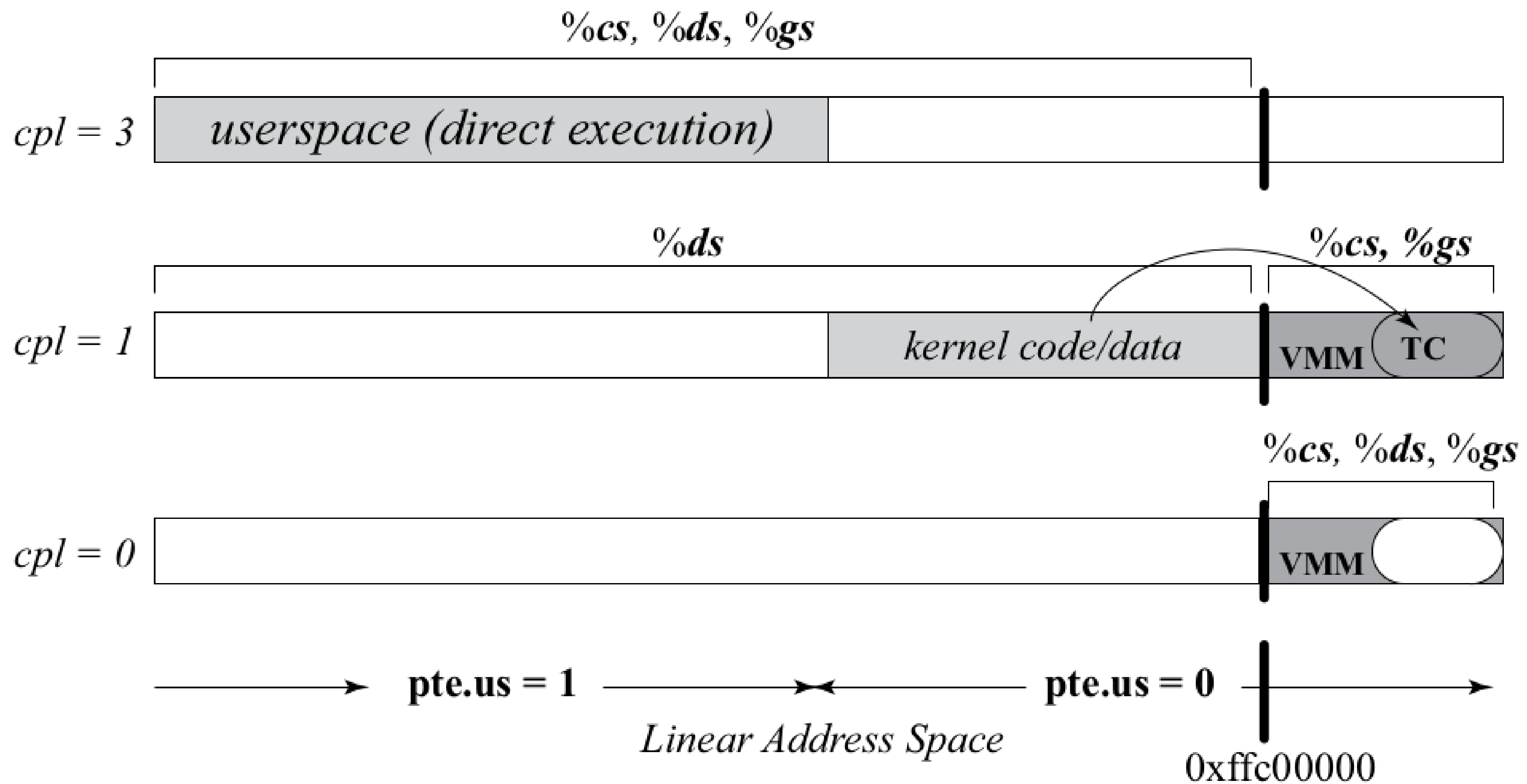
```
if !cr0.pe then
    return false;
end if
if eflags.v8086 then
    return true;
end if
if (eflags.iopl >= cpl) || (!eflags.if) then
    return false;
end if
/* a few corner cases here */
return true;
```

Long story short: use Dynamic Binary Translation when running in guest-supervisor mode and Direct Execution when running in guest-user mode

- **Dynamic Binary Translation**
  - Translates *dynamically* (at runtime), ahead of time, guest instructions, replacing privileged operations with trap instructions
  - Unit of translation: basic bloc rather than a single instruction
  - Translation cache and heavy optimizations

Vmware Workstation

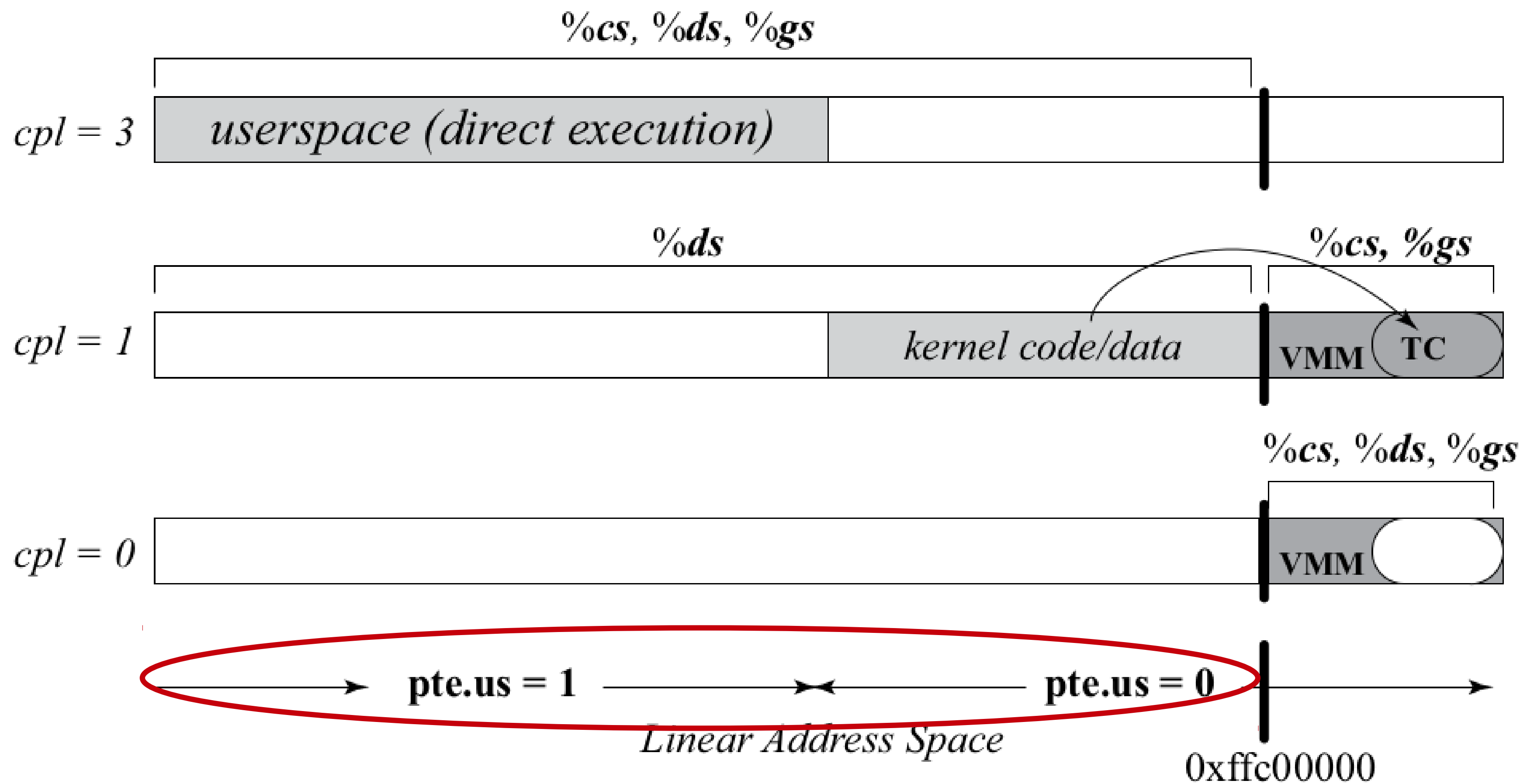
Segment truncation



Source: textbook

Vmware Workstation

Segment truncation



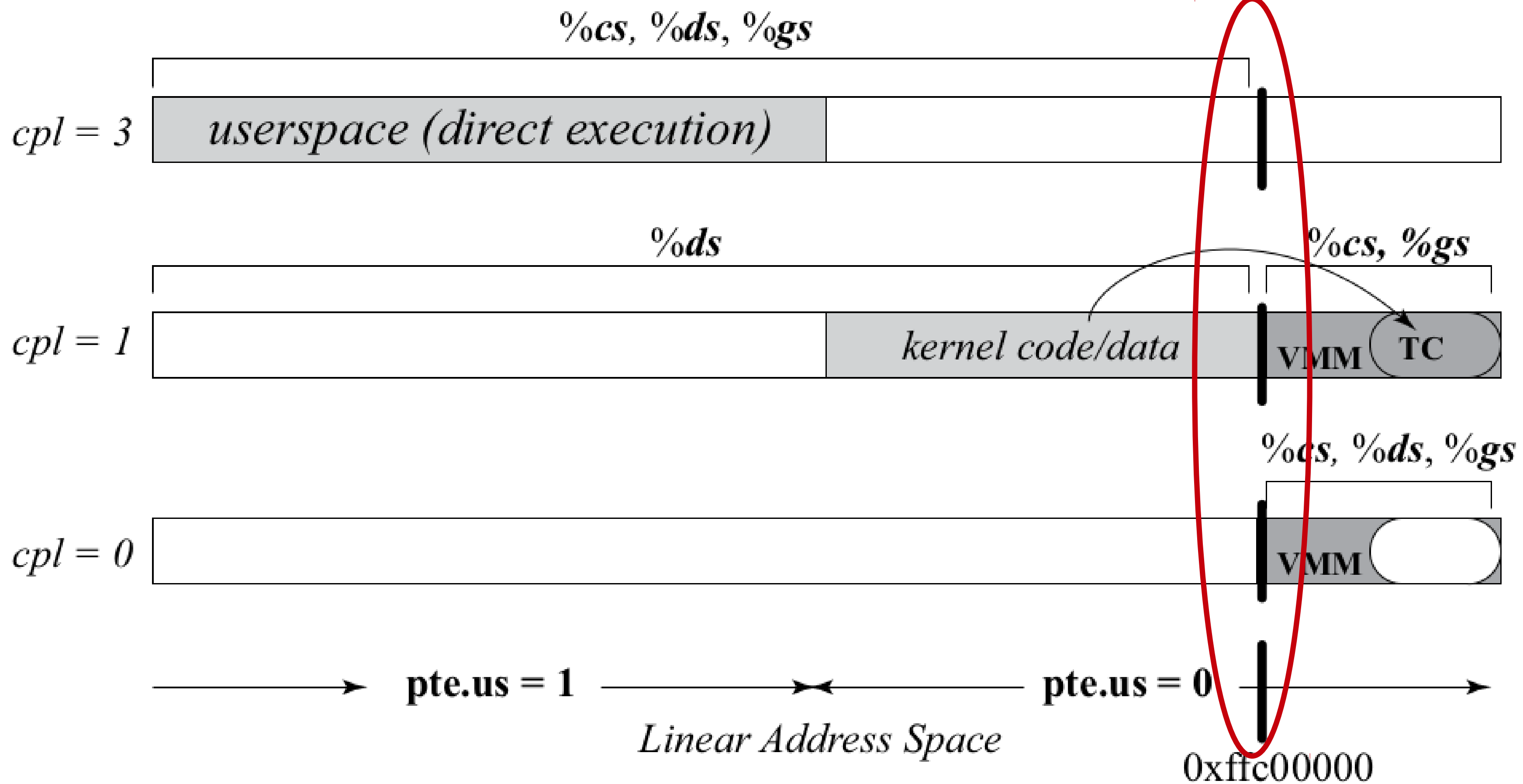
PT permission bit used to protect guest OS from guest applications

Source: textbook

Vmware Workstation

Segment truncation

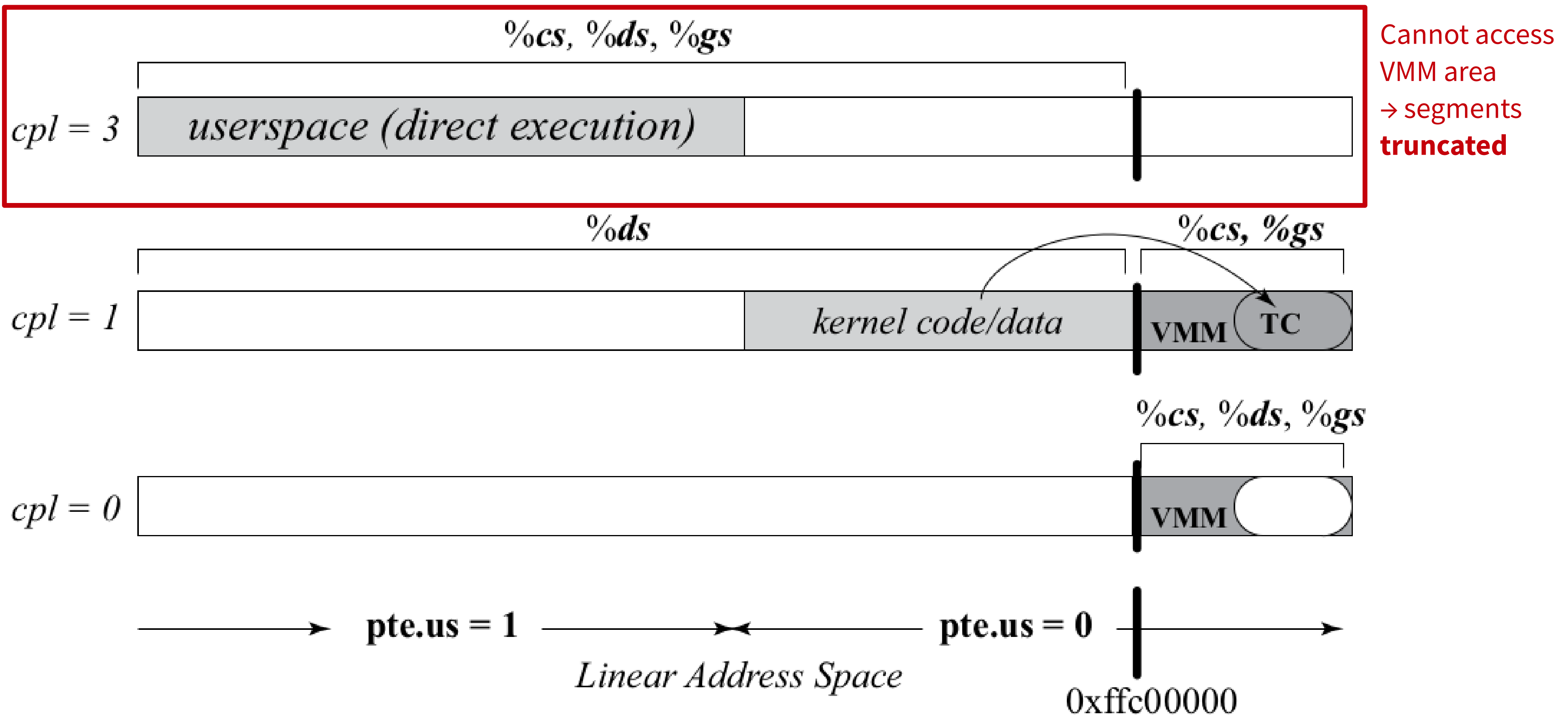
Segmentation used to protect VMM from guest



Source: textbook

Vmware Workstation

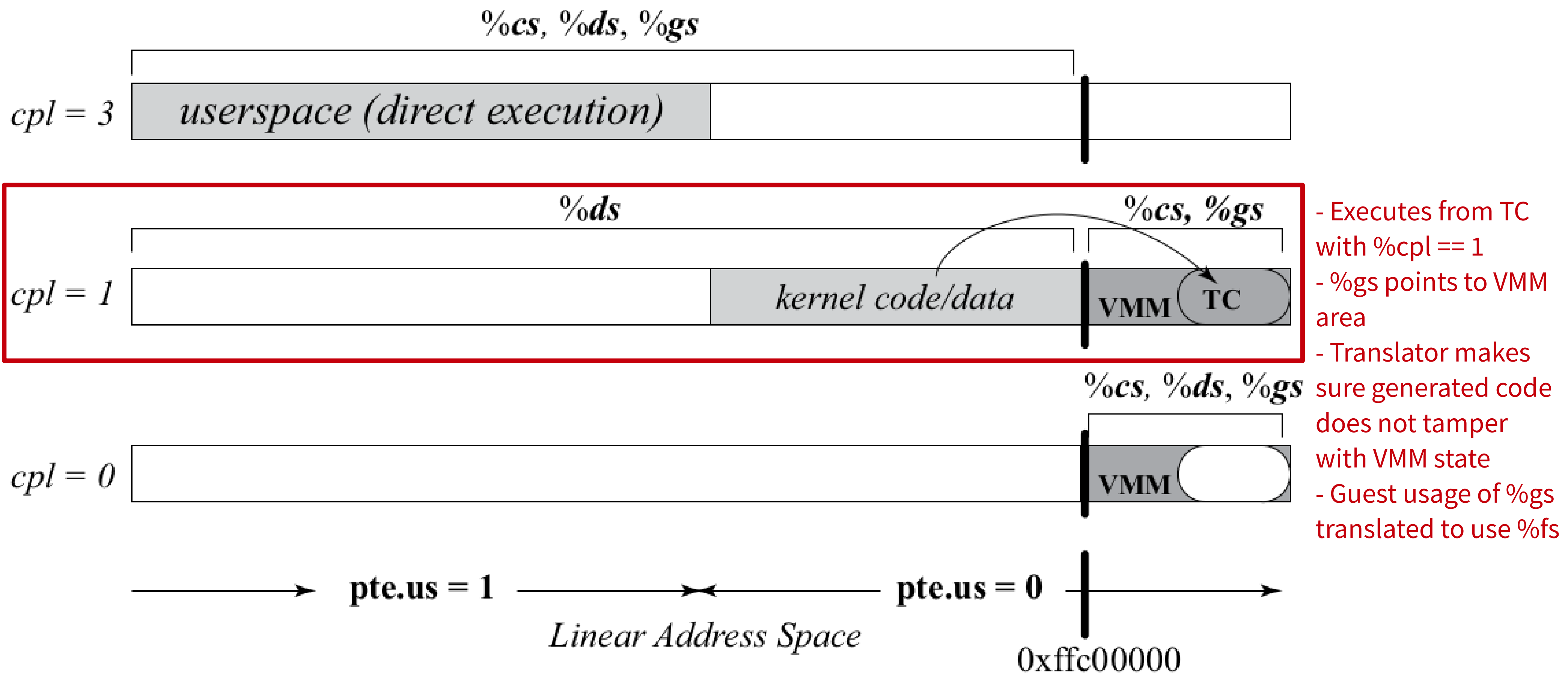
Segment truncation



Cannot access VMM area → segments **truncated**

Source: textbook

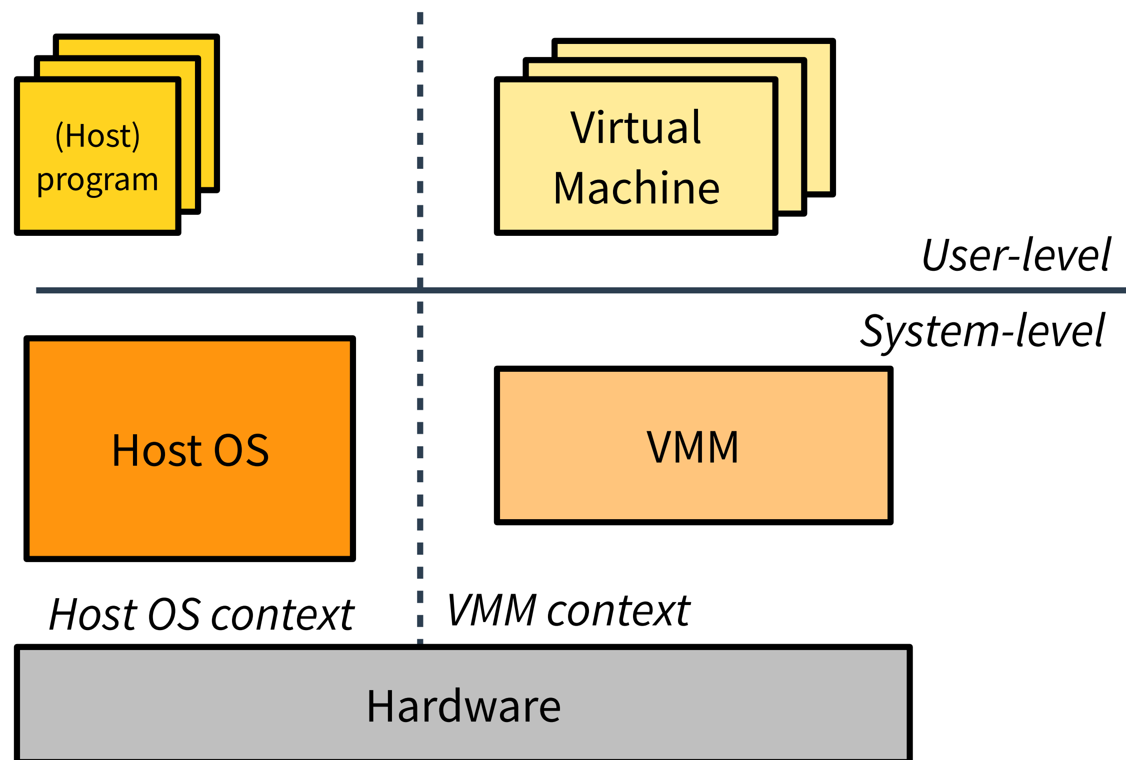
Vmware Workstation
Segment truncation



Source: textbook

Vmware Workstation
Hypervisor and host OS coexistence

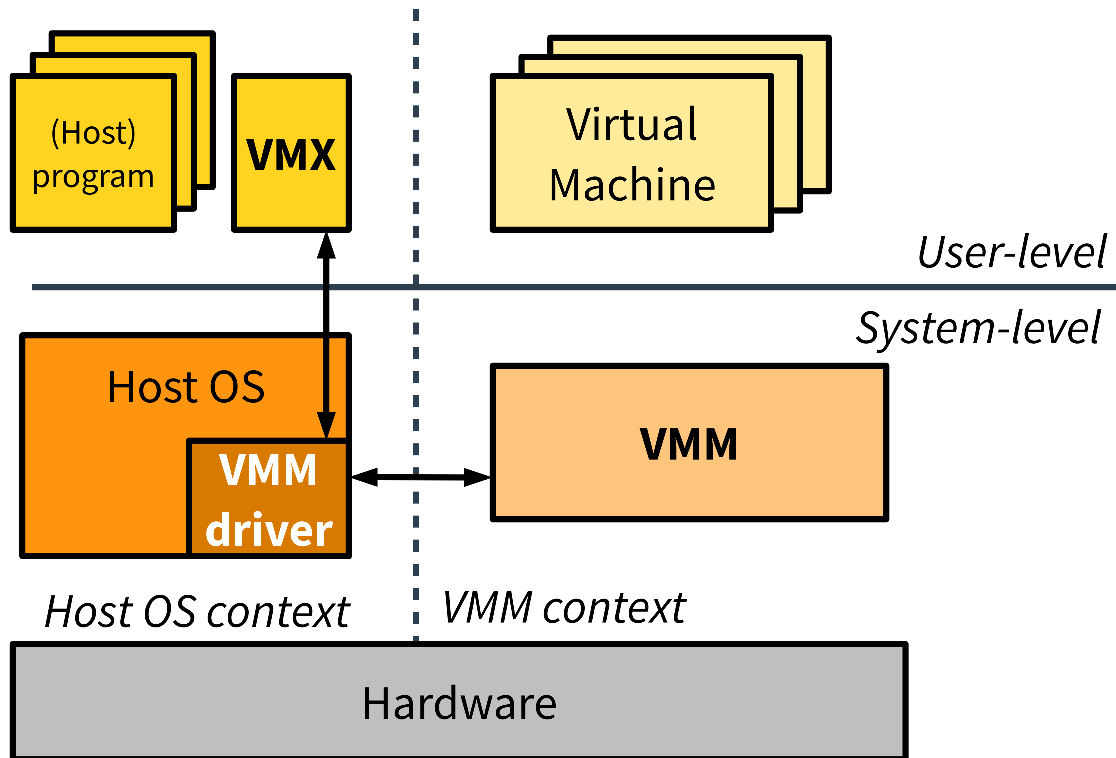
- Vmware VMM has full control of the CPU when executing a VM



Vmware Workstation

Hypervisor and host OS coexistence

- Vmware VMM has full control of the CPU when executing a VM



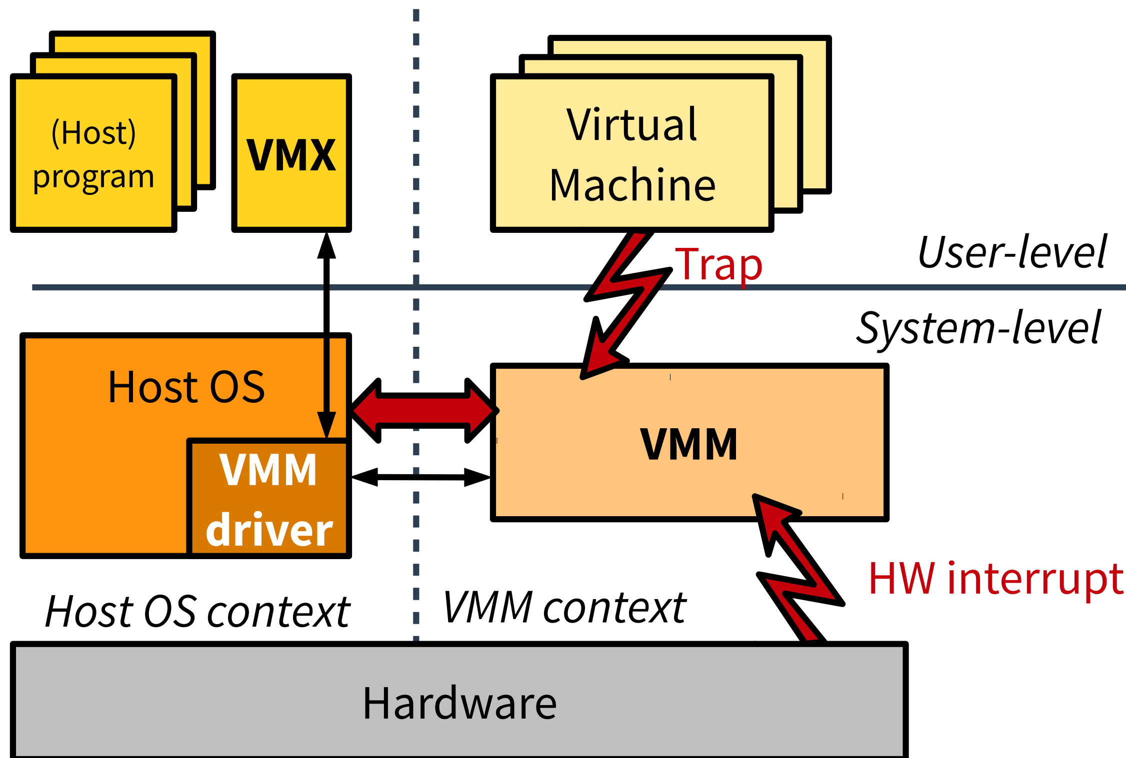
Adapted from textbook

- Host user-space program *VMX* for management and I/O emulation
- Host kernel code (VMM driver)
- VMM
  - Same privilege level as host OS but completely separated when running
    - Host OS paused and removed from virtual memory: world switch

Vmware Workstation

Hypervisor and host OS coexistence

- Vmware VMM has full control of the CPU when executing a VM

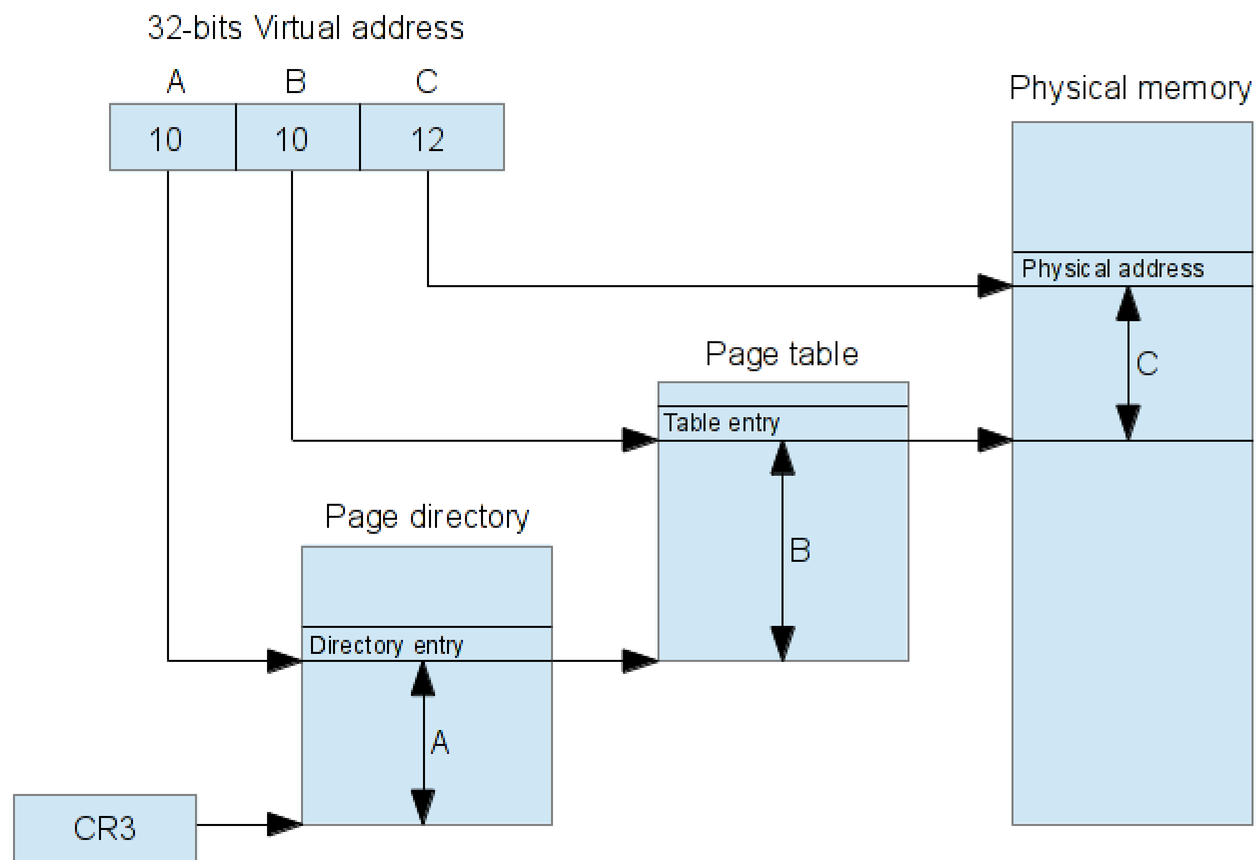


Adapted from textbook

- Host user-space program *VMX* for management and I/O emulation
- Host kernel code (VMM driver)
- VMM
  - Same privilege level as host OS but completely separated when running
    - Host OS paused and removed from virtual memory: **world switch**

Vmware Workstation

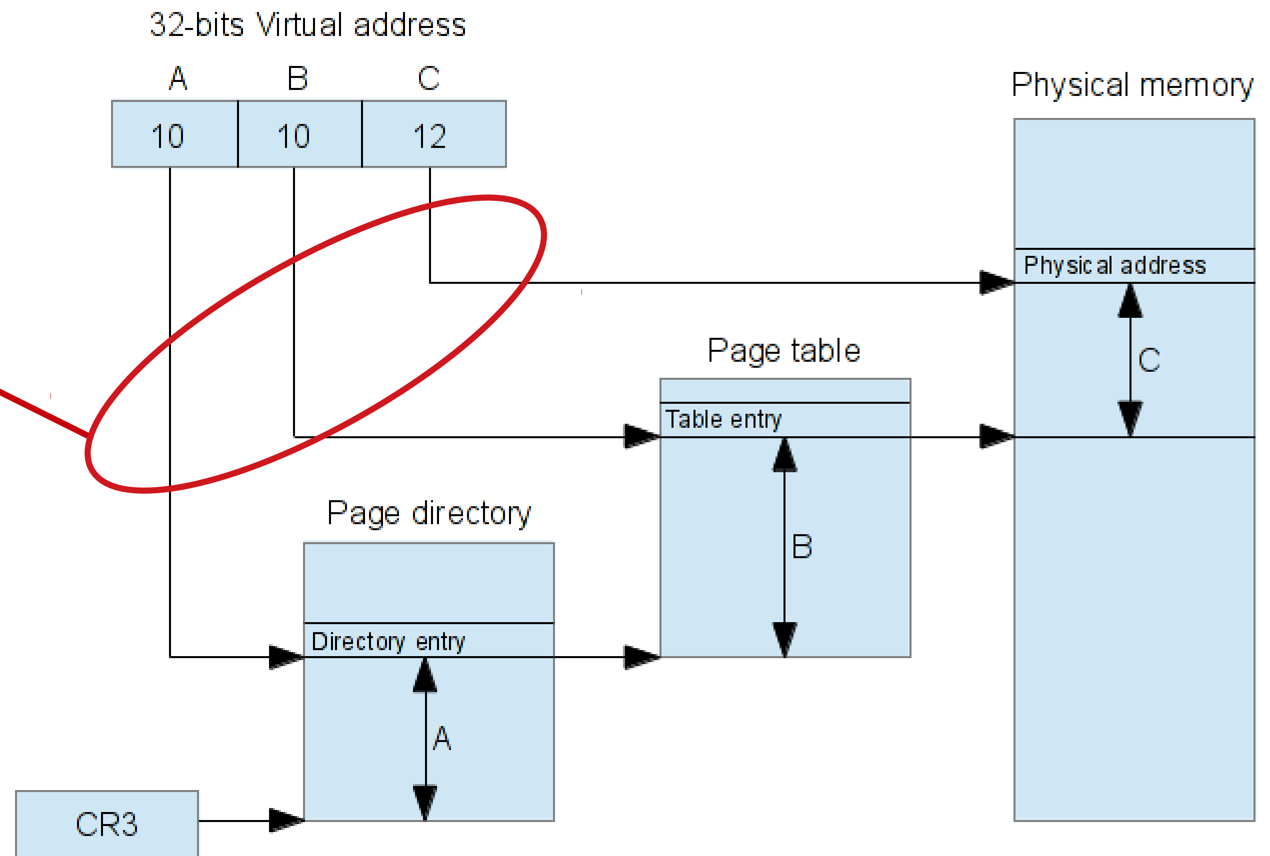
MMU virtualization: x86 paging



Vmware Workstation

MMU virtualization: x86 paging

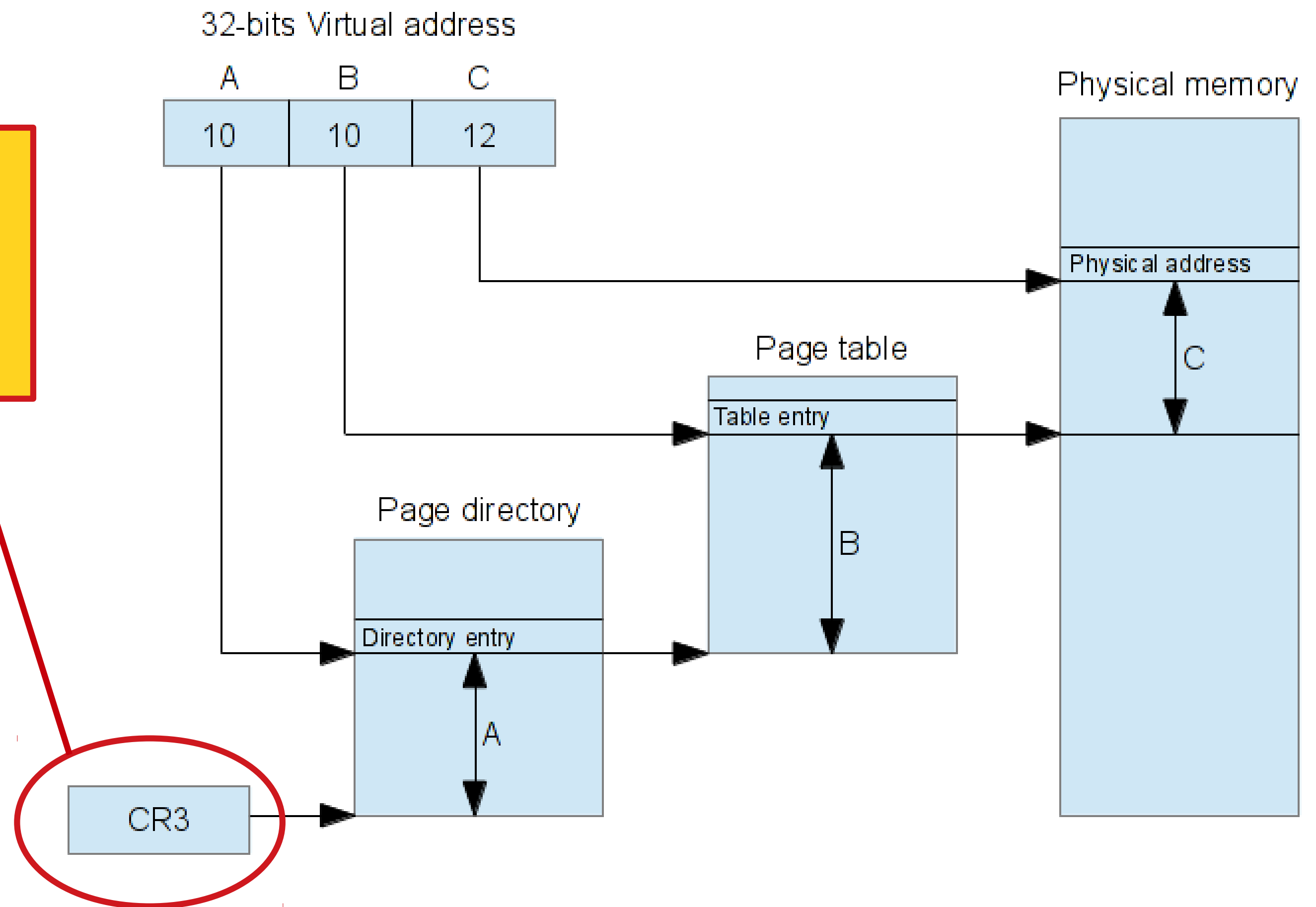
Translation made automatically by the hardware,
MMU walks the page table



Vmware Workstation

MMU virtualization: x86 paging

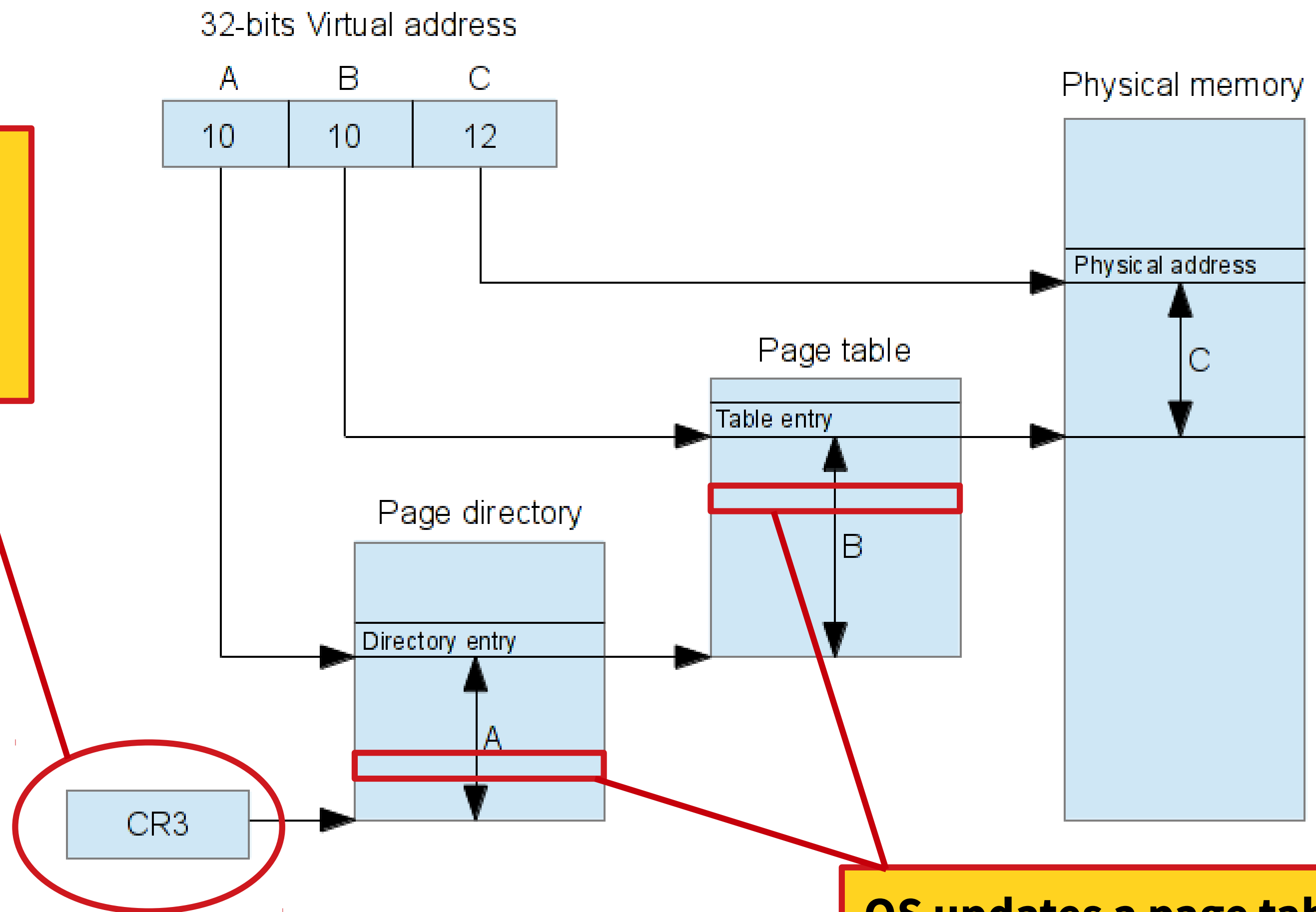
OS Installs a new page table:
Write in cr3 the address of the root page



Vmware Workstation

MMU virtualization: x86 paging

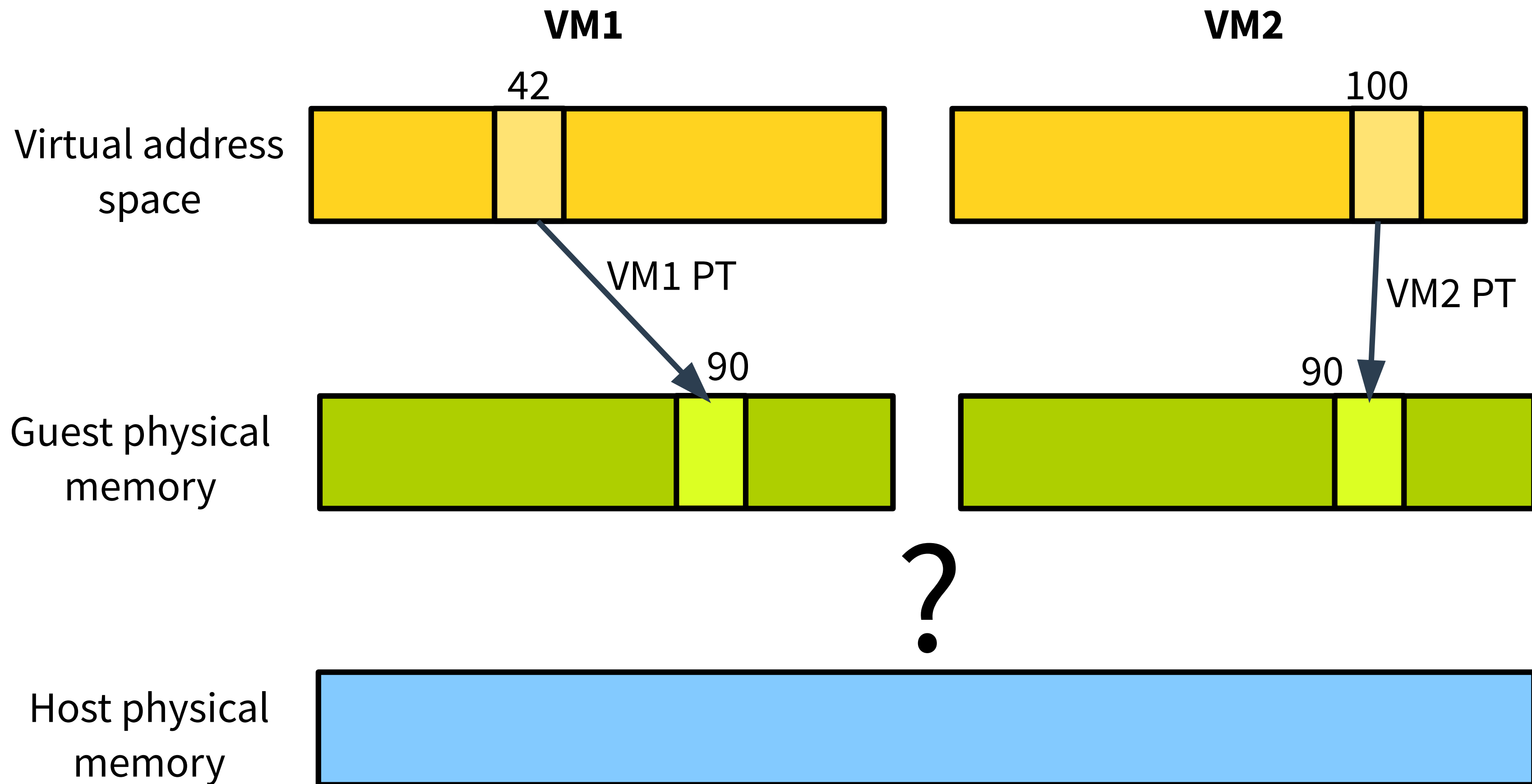
OS Installs a new page table:
Write in cr3 the address of the root page



OS updates a page table:
Write in memory in pages corresponding to page table

Vmware Workstation

MMU virtualization: shadow page tables



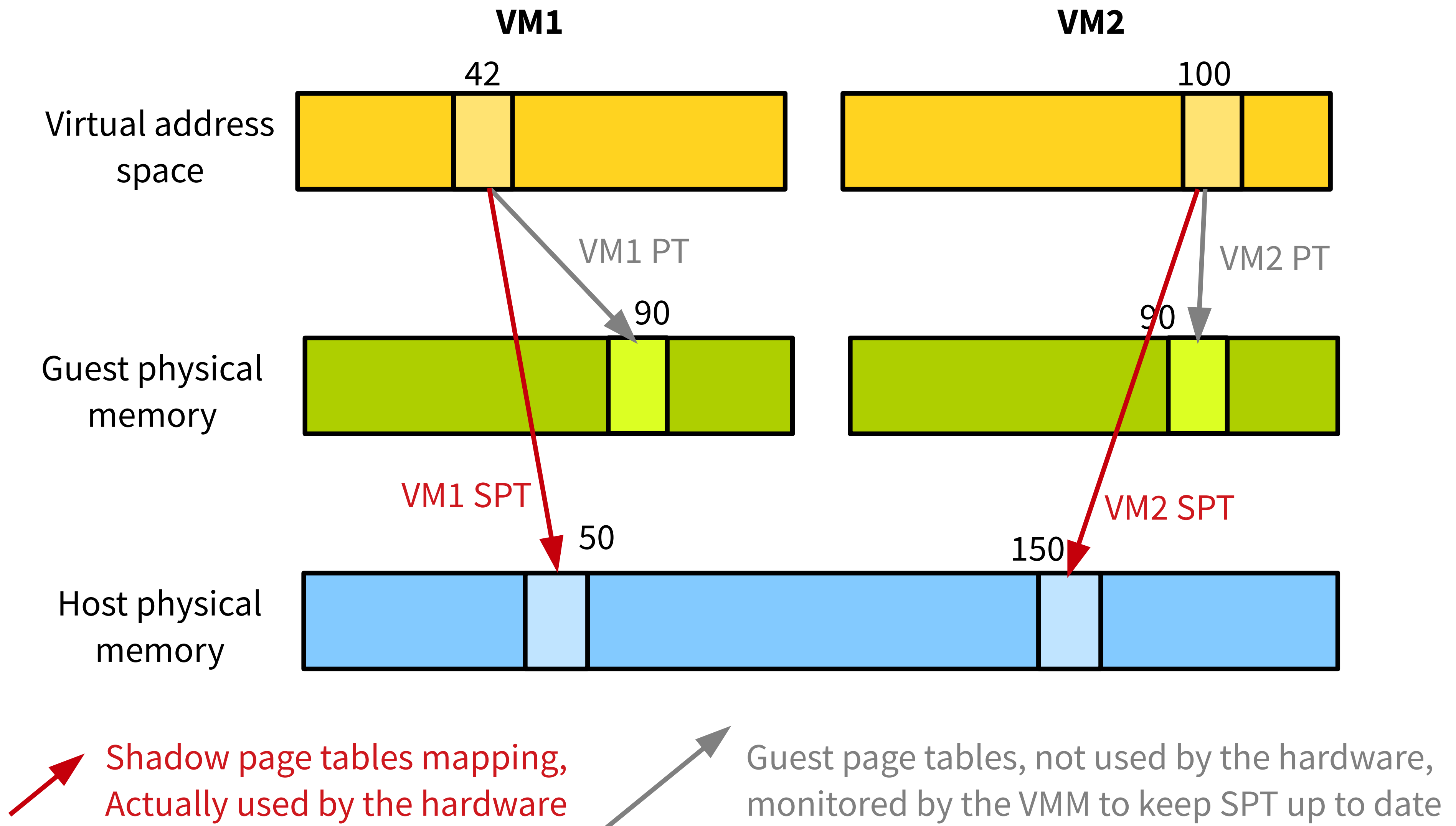
Vmware Workstation

MMU virtualization: shadow page tables

- **Each VM installs a page table by `mov` to `%cr3`**
  - Privileged operation, traps to the hypervisor
    - *The hypervisor cannot install the guest page table as-is* because two VMs may want to map the same physical page
- *The hypervisor actually installs a different page table* on the hardware, corresponding to guest-virtual → host physical mapping
  - Called the *shadow page table*
- Guest page table (not used by the hardware), is mapped read-only
  - Each update (i.e. modification of the page table) will trap the the hypervisor (shadow/hidden page fault) to keep the shadow page table in sync
- Relatively high performance cost:
  - Shadow page faults: trap and overhead for page table updates
  - Regular page faults: also traps to VMM, overhead

Vmware Workstation

MMU virtualization: shadow page tables



Outline

1) Disco (MIPS, 1997)

2) Vmware Workstation (x86-32, 1999)

3) Xen (x86-32, 2003)

4) KVM for ARM (ARMv5/v6, 2010)

Xen

- **Xen target x86-32**
- **Approach: paravirtualization**
  - *The guest OS sources can be (slightly) modified*
    - Replace sensitive, unprivileged instructions with direct calls to the hypervisor: hypercalls
    - Need to recompile: loss of equivalence
    - Mainly targets Linux, NetBSD and Solaris also available, as well as an experimental port of Windows XP
  - However applications can run unmodified
- **Paravirtualization: get performance at the cost of equivalence**

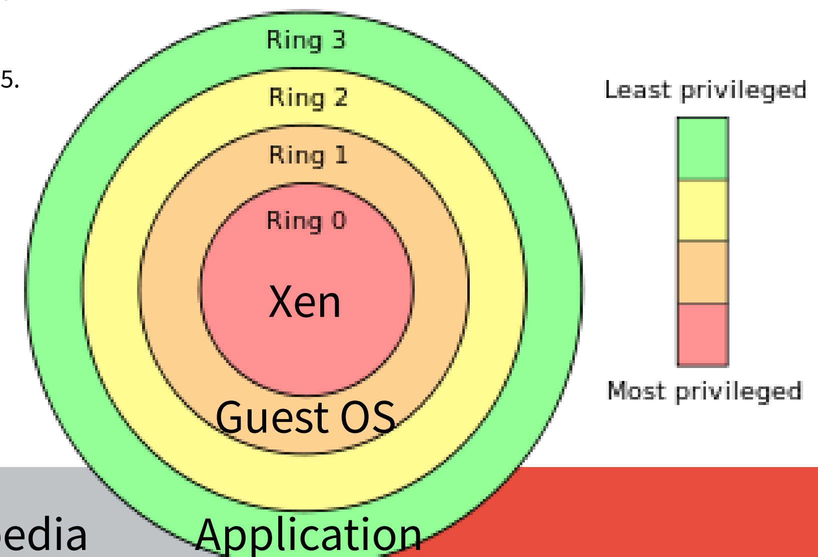
[1] Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

Xen

Xen paravirtualized interface

Memory Management	
Segmentation	Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
Paging	Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.
CPU	
Protection	Guest OS must run at a lower privilege level than Xen.
Exceptions	Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
System Calls	Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
Interrupts	Hardware interrupts are replaced with a lightweight event system.
Time	Each guest OS has a timer interface and is aware of both 'real' and 'virtual' time.
Device I/O	
Network, Disk, etc.	Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.

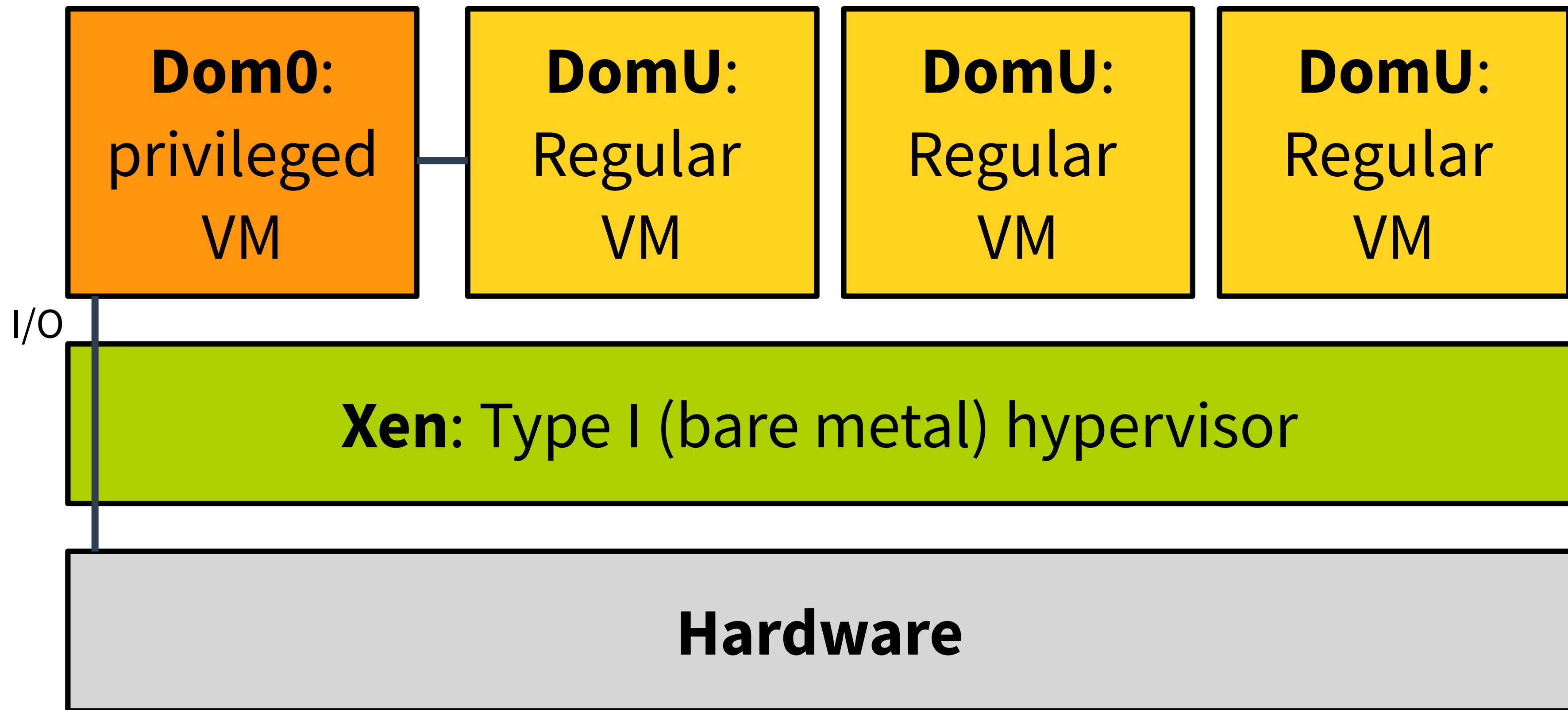
Source: Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.



Source: wikipedia

Xen

Xen architecture



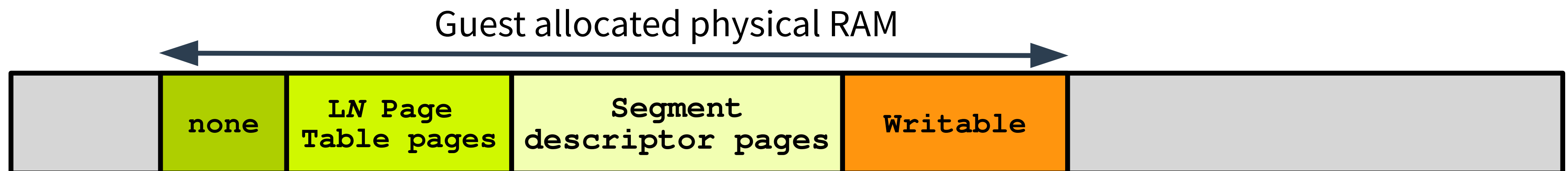
Dom0:

- first VM to execute when Xen starts
- handles administrative tasks
- handles device emulation

Xen

MMU virtualization: direct paging

- **Contrary to shadow paging, guest page tables are used directly**
  - However they are not setup and updated directly by the guest
- **Guest page tables are mapped read-only for the guest**
  - Guest installs/update page tables through *hypercalls*
    - **`mmu_update`**
    - Can be **batched** to avoid multiple traps such as with shadow paging
- **Guest *directly requests guest-virtual to host-physical mapping***
- **In the hypervisor, the hypercall implementation *check* the validity of the requested mapping**
  - Invariant enforced: only pages with writable type have a writable mapping in the PT

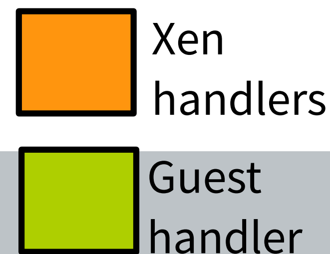


Xen

CPU virtualization: interrupts & system calls

- **At boot time a guest registers its Interrupt Descriptor Table with Xen through the `set_trap_table` hypercall**
  - IDT contains handler addresses for each interrupt number
- **Xen installs its own IDT on the hardware**
  - Most interrupts are simply forwarded to the guest through its registered IDT
  - (validated) guest syscall handler for fast syscall processing (no switch to ring 0)
  - Guest page fault handler is modified to avoid accessing cr2

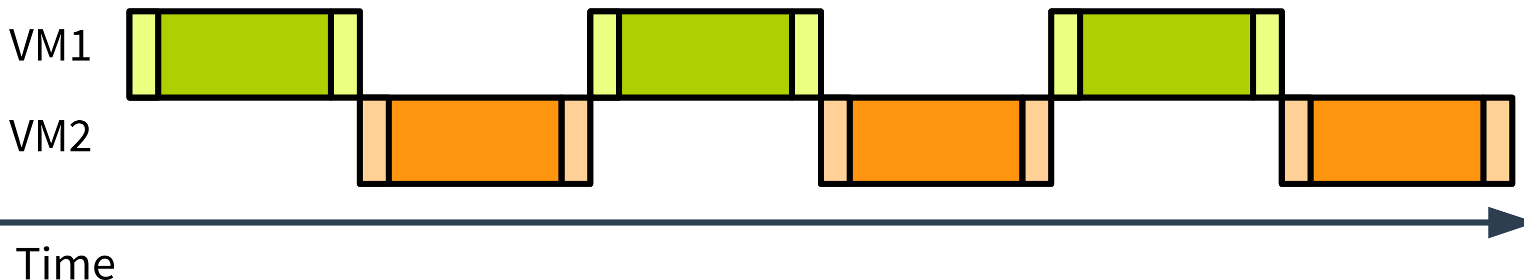
Interrupt/ exception #	Description
0	Divide error
1	Debug exception
2	NMI
3	Breakpoint
...	
14	Page fault
...	
128	**System call (convention)**
...	



Xen

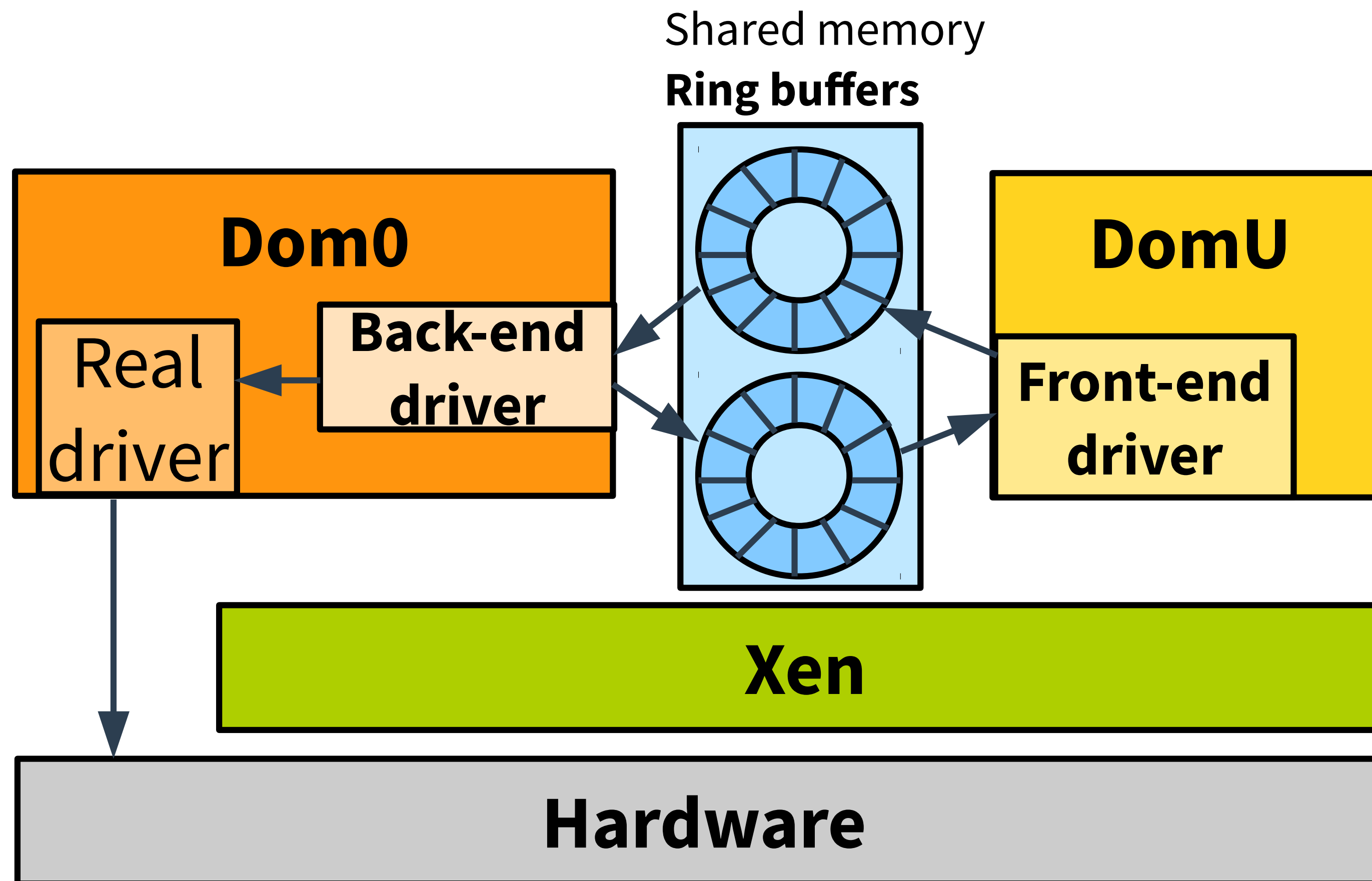
CPU virtualization: time management

- **Two types of time a Xen guest needs to be aware of:**
  - Wall-clock time: how much absolute time has passed since a given referential point in the past
    - Useful to keep track of the time of day, schedule operations in the future (ex: cron)
  - Virtual time: how much time a guest has spent actually running
    - Useful to ensure fair scheduling within the guest
      - ➔ The guest itself is not scheduled 100% of the time
    - 1PCPU, 2 domains with 1VCPU and 2 tasks each, each task supposed to be scheduled for an equal amount of time
      - ➔ Each domain (i.e. VCPU) also scheduled for an equal amount of time
      - ➔ This situation may occur: 1 task in each domain gets close 50% of the PCPU, other task gets close to nothing
- **Wall clock-time computed from:**
  - Initial system type, current system time, timestamp counter



Xen

I/O virtualization: paravirtualized drivers



Outline

1) Disco (MIPS, 1997)

2) Vmware Workstation (x86-32, 1999)

3) Xen (x86-32, 2003)

4) KVM for ARM (ARMv5/v6, 2010)

KVM/ARM

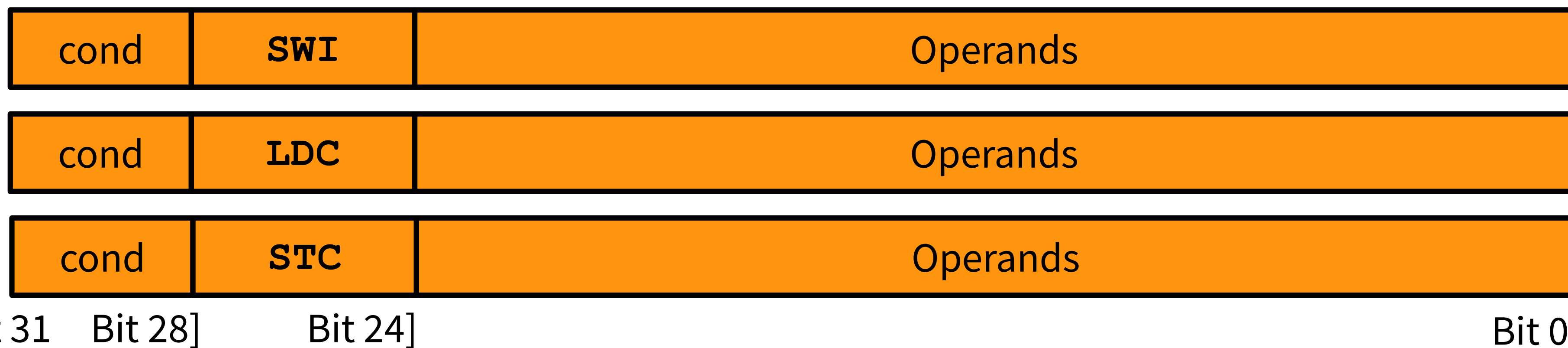
Lightweight paravirtualization

- **ARM is not directly virtualizable**
  - *Lightweight paravirtualization*: script-based technique to automatically paravirtualize a guest OS
    - Replace sensitive instructions with hypervisor calls
    - Completely automated, no guest-OS specific expertise required
- **Different from Xen virtualization requiring *manual source modification* and *guest-OS specific expertise***
- **Script tested successfully on multiple kernel versioms**
  - Only concerned by ASM files (C compilers do not generate privileged instructions)
  - Work with regular expressions, replacing privileged instructions with trap instructions (exceptions)

Lightweight paravirtualization: trap instructions

- **With which trappable instruction should be replace sensitive and unprivileged ones?**
  - `SWI`: Software Interrupt, normally used for syscalls → traps
    - Only 24 bits to encode the instruction to emulated (type + operands) → not enough space
  - `SWI` in supervisor mode, `LDC`/`STC` in user mode
    - Load/Store from coprocessors 1-6 (traps)

https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf
- page A4-210
- page A3-31



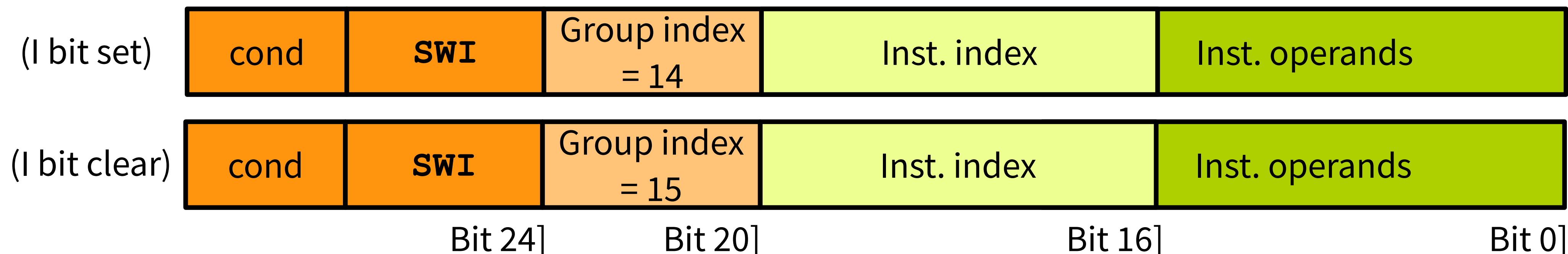
KVM/ARM

Instructions encoding

Sensitive, unprivileged instructions to encode classified into 15 groups:

- **Group 0: Status register access instructions (5 in total)**

- **Groups 2 – 12: Sensitive load-stores**
- **Groups 14 & 15: Sensitive data processing**



Summary

	Disco	VMware Workstation	Xen	KVM for ARM
Architecture	MIPS	x86-32	x86-32	ARMv5
Hyp type	Type-1	Type-2 (§4.2.4)	Type-1 with dom0 (§4.4)	Type-2 (§4.5)
Equivalence	Requires modified kernel	Binary-compatible with selected kernels	Required modified (paravirtualized) kernels (§4.3)	Required modified (lightweight paravirtualized kernels (§4.5)
Safety	Via de-privileged execution using strictly virtualized resources	Via dynamic binary translation; isolation achieved via segment truncation	Via de-privileged execution with safe access to physical names	Via de-privileged execution using strictly virtualized resources
Performance	Via localized kernel changes and L2TLB (§4.1.2)	By combining direct execution (or applicastions) with adaptive dynamic binary translation (§4.2.3)	Via paravirtualization of CPU and IO interactions	Via paravirtualization of CPU and IO interactions

Source: textbook

Readings

- Bugnion, Edouard, et al. "Disco: Running commodity operating systems on scalable multiprocessors." ACM Transactions on Computer Systems (TOCS) 15.4 (1997): 412-447.
- Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.
- Chisnall, David. The definitive guide to the xen hypervisor. Pearson Education, 2008.
- Dall, Christoffer, and Jason Nieh. "KVM for ARM." Proceedings of the 12th Ottawa Linux Symposium (2010).