Virtualization

15-410 Spring 2006

Mike Cui

Synchronization

- Kernel due tonight
- If you are using your late days, don't forget to register on the website
- Be alert for an opportunity to study large warm floppy disks at midnight

Outline

- Overview
- Full Virtualization
- Virtualization on x86
- Paravirtualization
- Hardware Assisted Virtualization
- Software Implementation

Virtualization

Process of presenting and partitioning computing resources in a *logical* way rather than what is dictated by their *physical* reality

Old topic

Virtual Machine

An execution environment identical to a physical machine, each with the ability to execute a full operating system

Q: Process : OS :: OS :
A: Virtualization layer

IBM System 370

VM/CMS ~1967

VM - Virtualization Layer

CMS - Single-user DOS-like operating system

1000 users, each user gets a personal mainframe!

Motivation

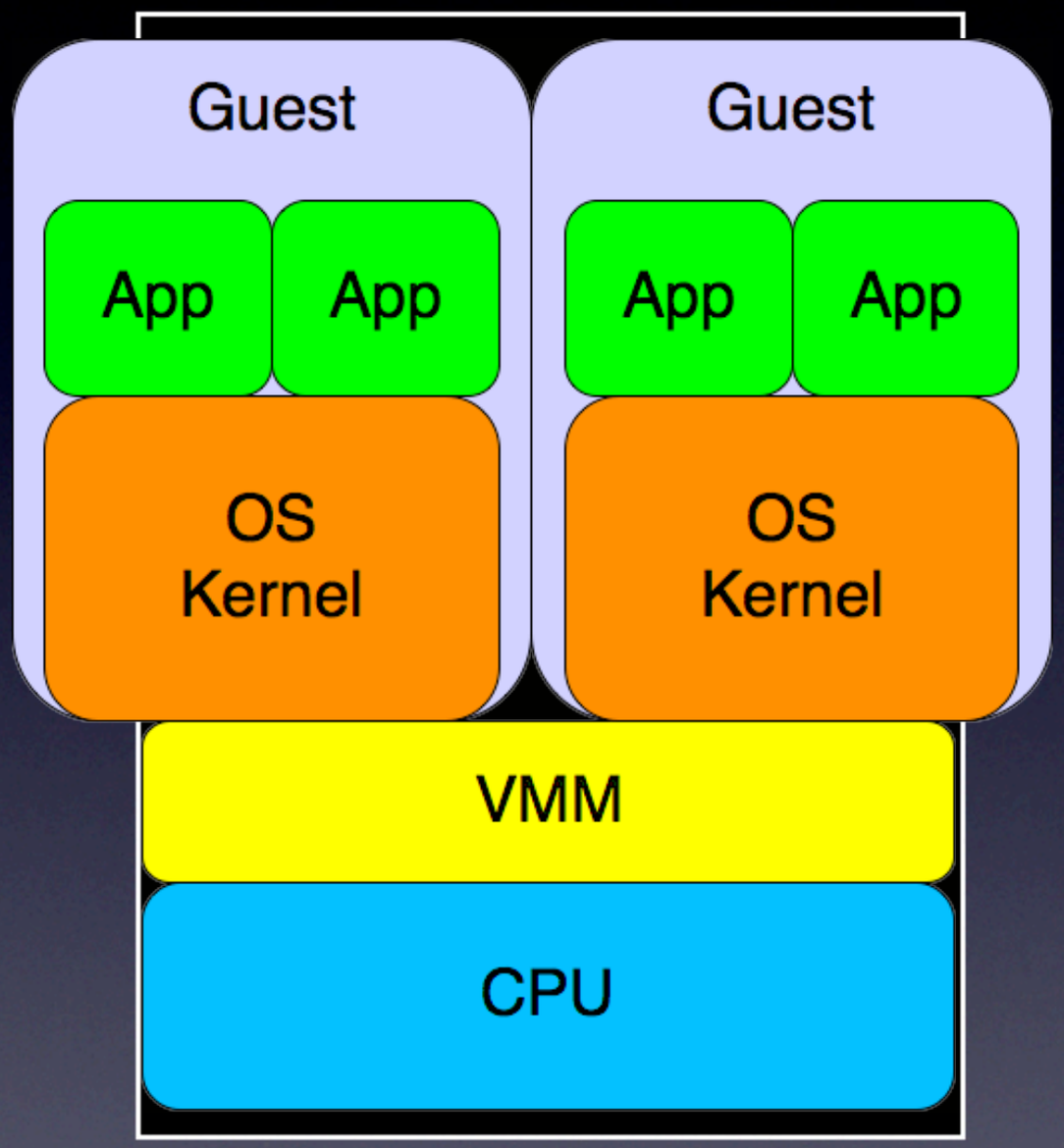
- Virtual machines are easier to manage
  - Easier to move/store/copy
- Virtual machines allow computer resources to be used more efficiently
- It's convenient to run more than one OS simultaneously
- Billion-dollar industry

Motivation for You

- Virtualization is cool
- Application of OS concepts you already know from this class
- Impress VMware interviewer

Virtual Machine Layers

The virtualization layer is commonly referred to as the *Virtual Machine Monitor* (*VMM*) or *Hypervisor*



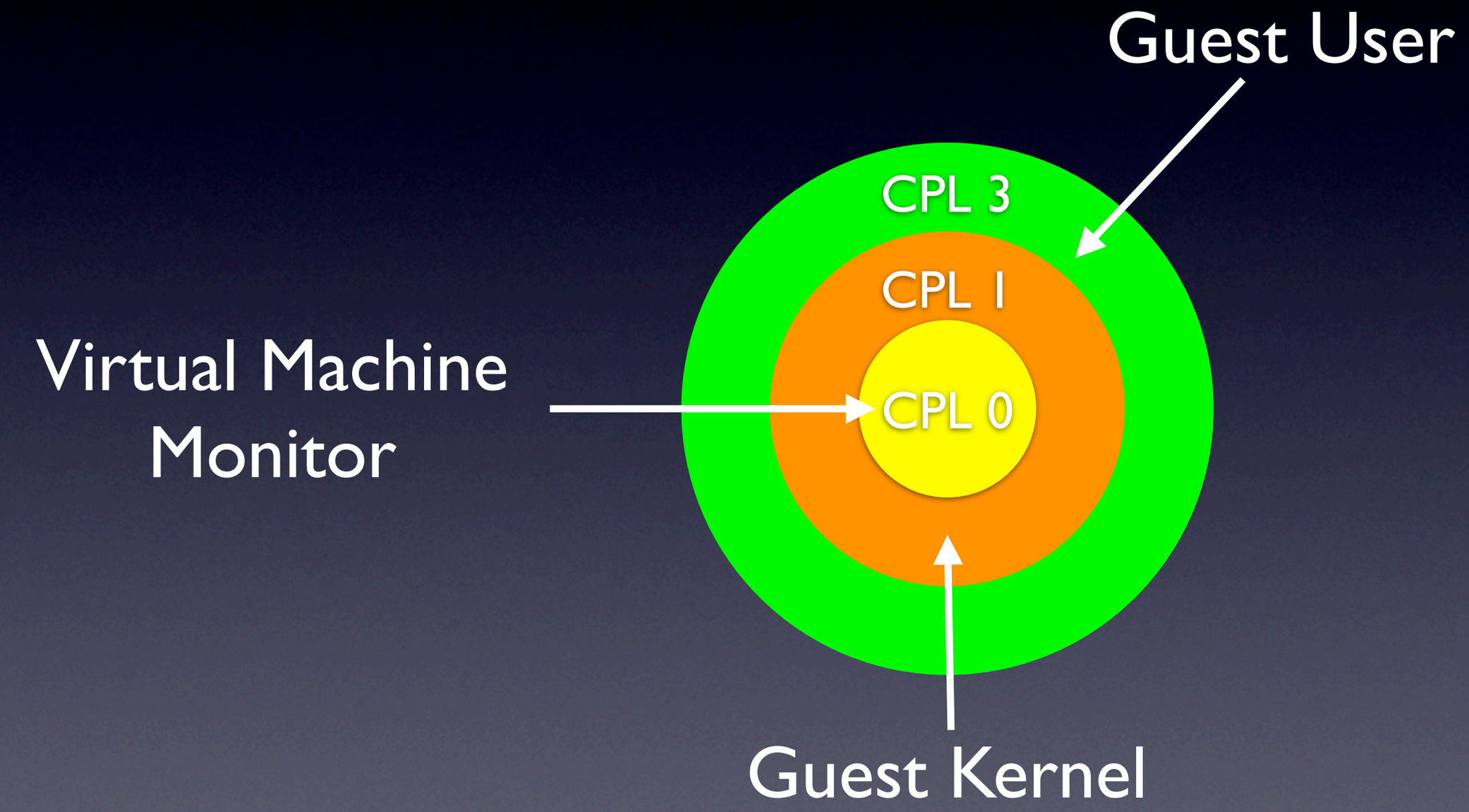
Virtualization Layer

- Runs with the highest privileges
- Controls and allocates hardware resources for the virtual machine
- Occupies a small region of the virtual address space
- It is the "operating system"

Protection

- VMM needs to be protected from the guest
- Guest kernel needs to be protected from its users
- Need 3 privilege levels
  - Luckily, on x86, we have 4

Protection


Guest User
CPL 3
CPL 1
CPL 0
Virtual Machine Monitor
Guest Kernel

Full Virtualization

The VMM creates and illusion that each guest operating system owns the hardware exclusively and executes with the highest privileges.

Process : OS :: OS : VMM

Challenge?

Guest OS

- CPL 0 code running at CPL 1
- Accesses physical memory
- Uses virtual memory
- Installs interrupt / system call handlers
- Does not know about the VMM
- Controls hardware devices

Trap and Emulate

Guest kernel runs at lower privilege level than VMM

When guest attempts to execute privileged instruction, trap into the VMM, and emulate the instruction.

Which instructions should be trapped?

(Which instructions should not be trapped?)

Disable Paging

Guest tries to write to %CR0

This instruction at CPL 1 causes #GP

VMM's #GP handler decodes the faulting instruction

#GP handler emulates the instruction (How?)

#GP handler returns execution to the next instruction

Physical Memory Protection

- Physical memory is a shared resource
- Do not allow guest to access it directly
- Instead, map *virtual* frames for the guest
- Hide *real* frames from the guest
- How?

Physical Memory Virtualization

- Just like virtual memory, so use it!
- Always run guest with paging enabled
  - When guest tries to disable it, don't!
- When guest runs with paging disabled
  - Point %CR3 to virtual frame mappings
  - Trap accesses to %CR0 and %CR3

Virtual Memory Virtualization

- Guest OS will need to use virtual memory
- Need 2 level virtual address translation
  - Virtual page to virtual frame
  - Virtual frame to real frame
- Another level of indirection?
  - Implement in software

Virtual Virtual Memory

- Rewrite the guest's page tables
  - Trap writes to %CR3
  - Walk through page tables and translate virtual frames to real frames
- Problem?

Page Table Shadowing

- Instead, make a *shadowed* copy of the page tables containing real frame addresses
- Processors sees the *shadowed* page tables
- Guest OS sees its own *virtual* page tables
- Trap reads from %CR3, so guest never finds out about the shadowed page tables
- Problem?

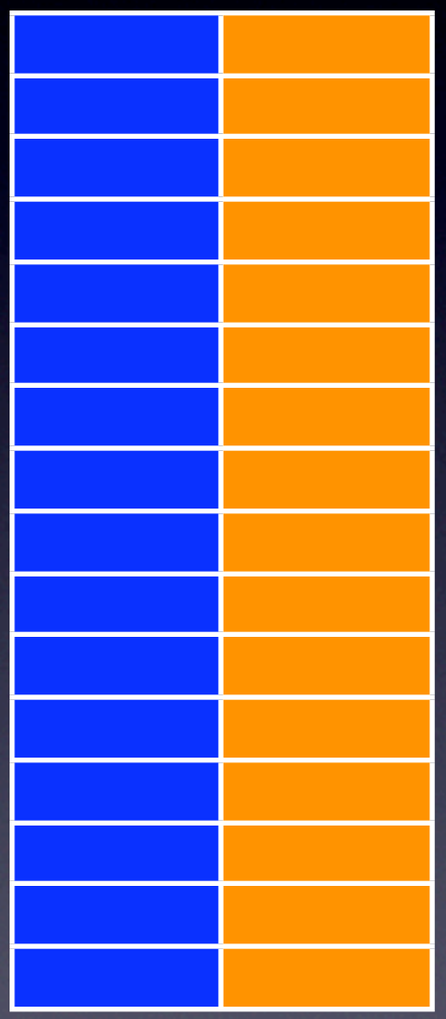
Page Table Tracing

- Trap guest writes to its page tables
  - Map guest page tables read only
  - Update shadowed page tables whenever guest page tables are modified

Guest sets up its page tables

Trap writes to %CR3

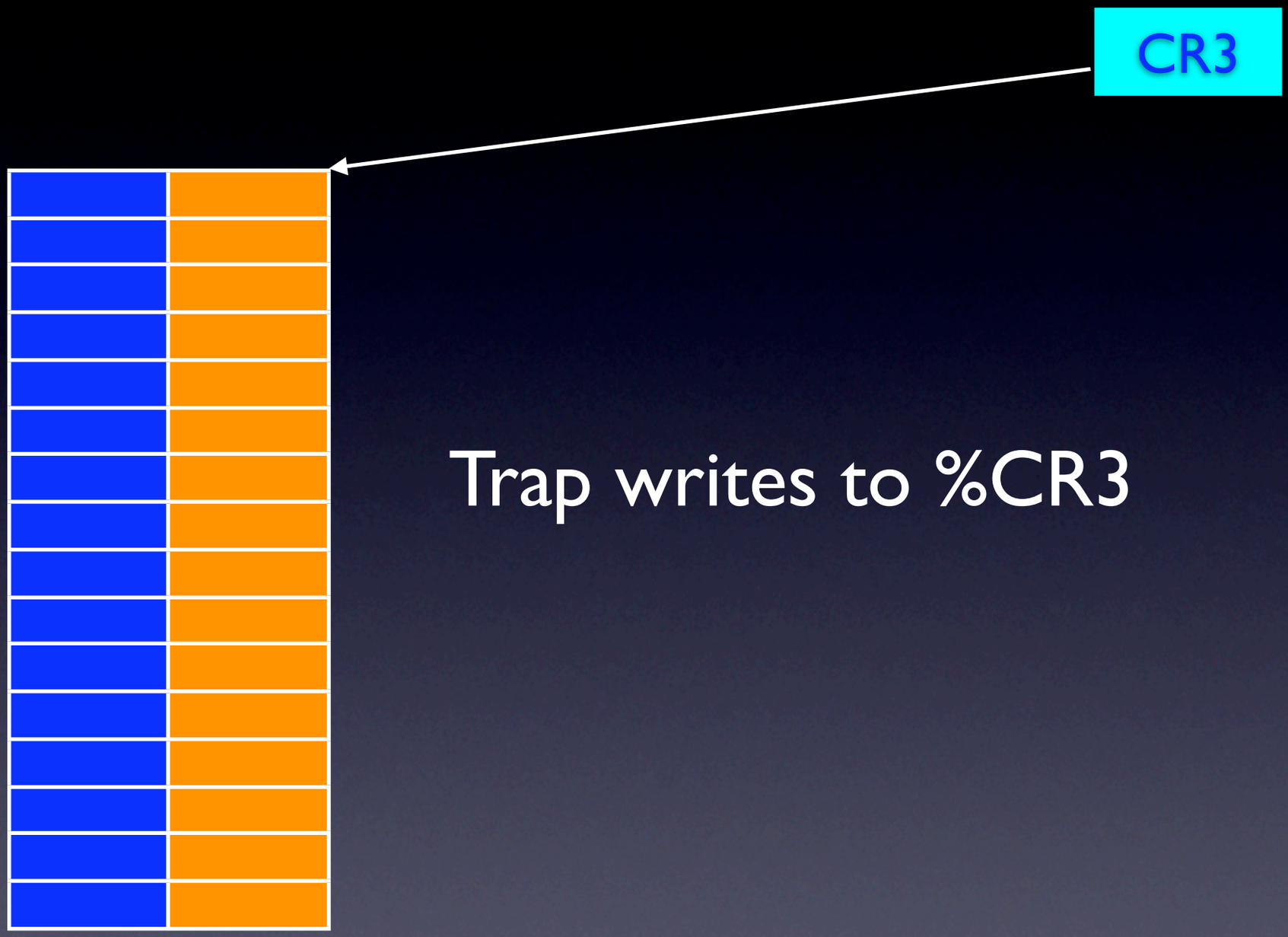
Guest sets up its page tables



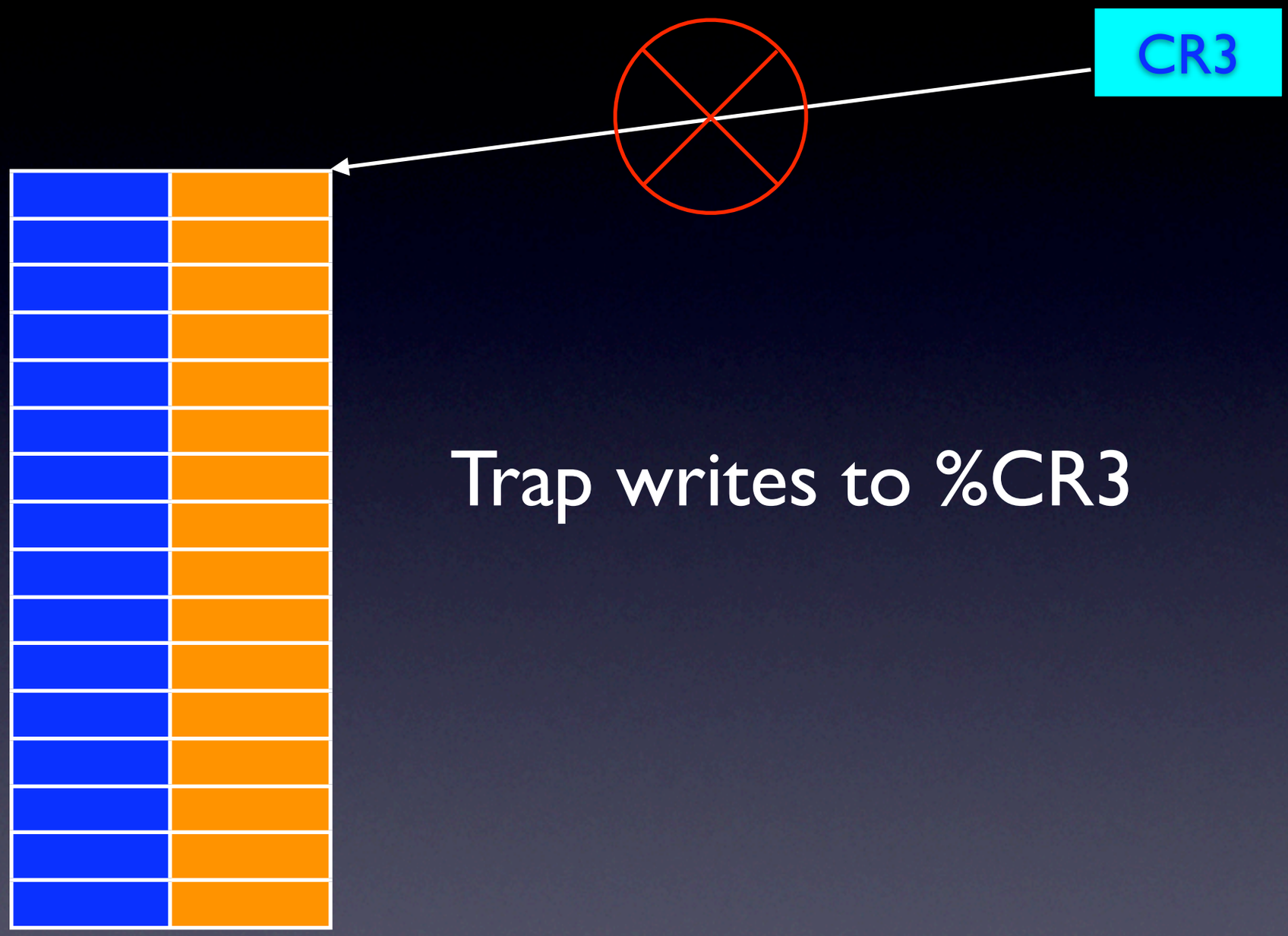
Trap writes to %CR3

Virtual Page Virtual Frame

Guest sets up its page tables



Guest sets up its page tables

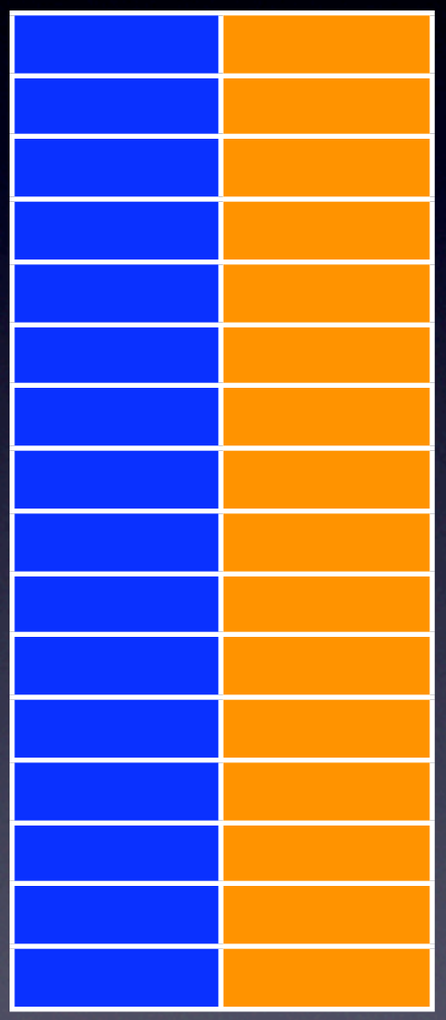


Trap writes to %CR3

Virtual Page Virtual Frame

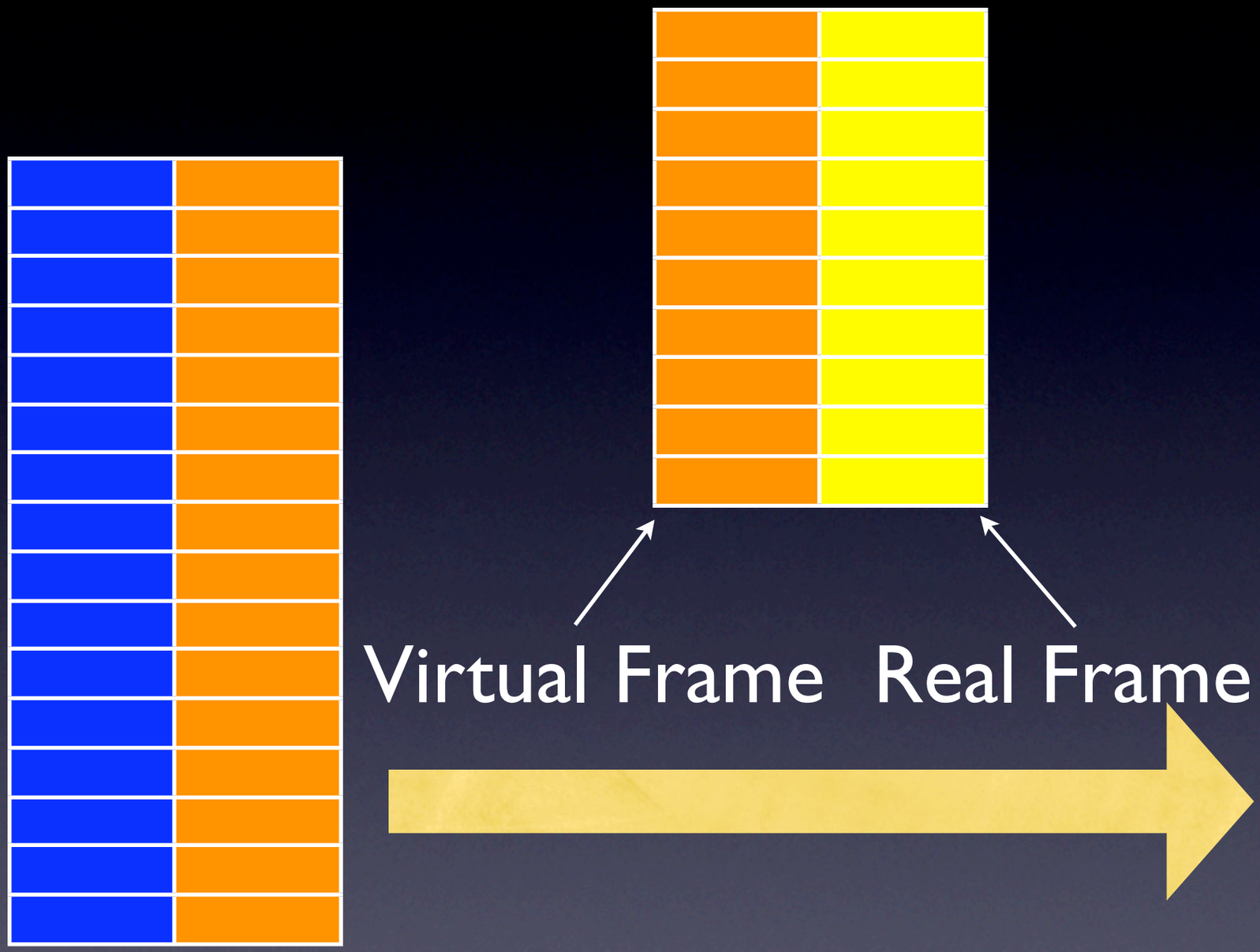
VMM makes shadowed page tables

VMM makes shadowed page tables



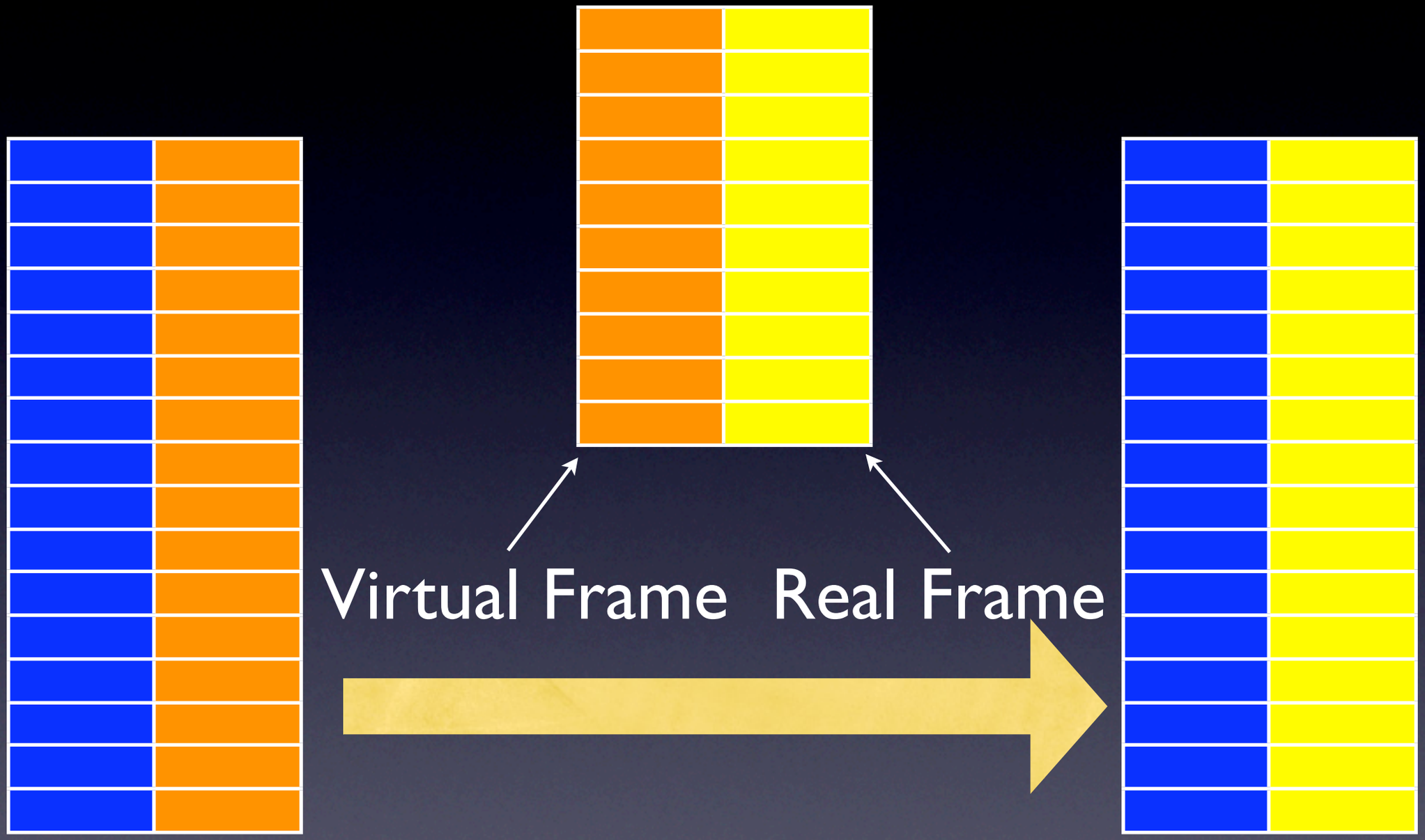
Guest Page Table

VMM makes shadowed page tables



Guest Page Table

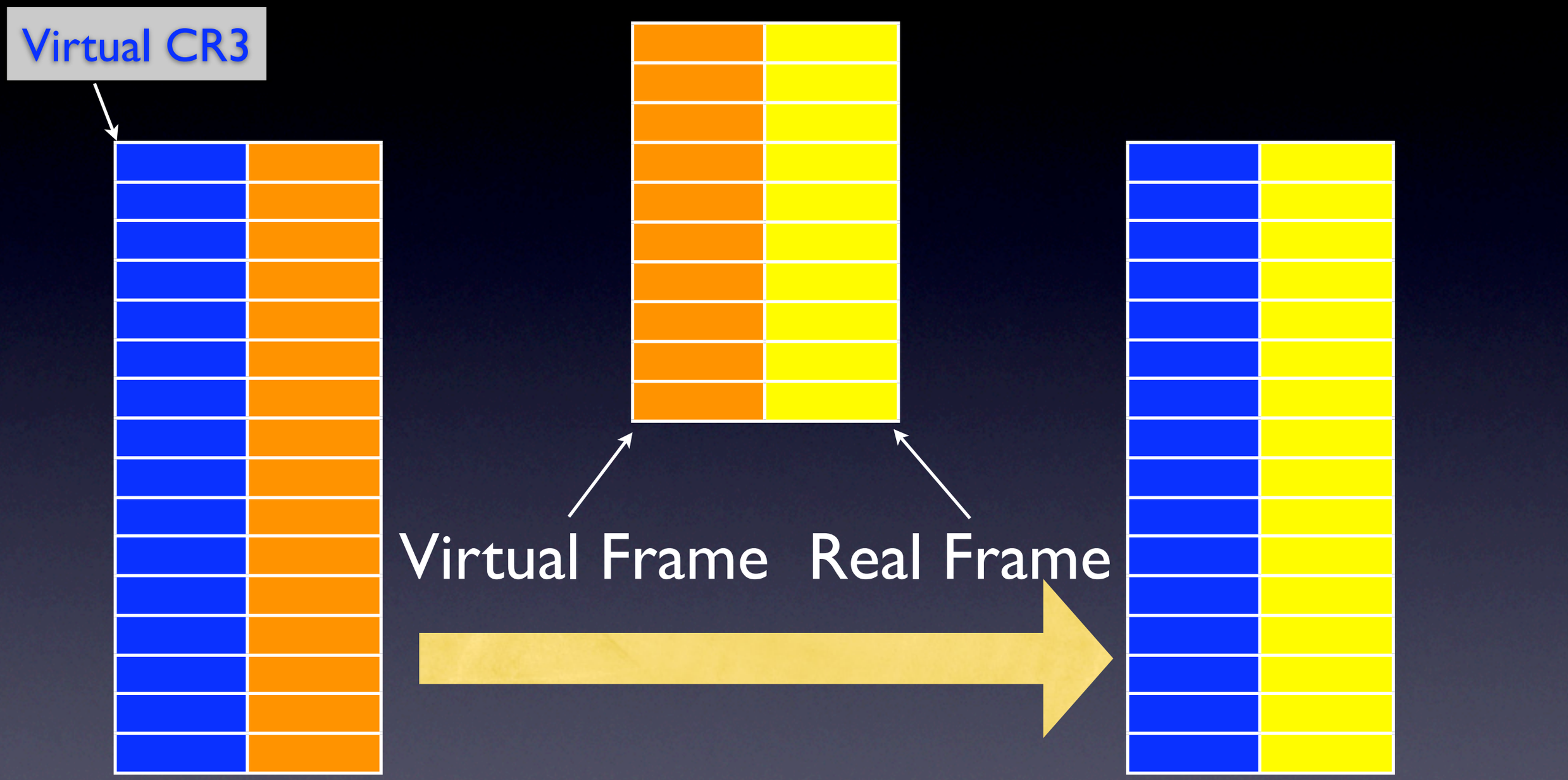
VMM makes shadowed page tables



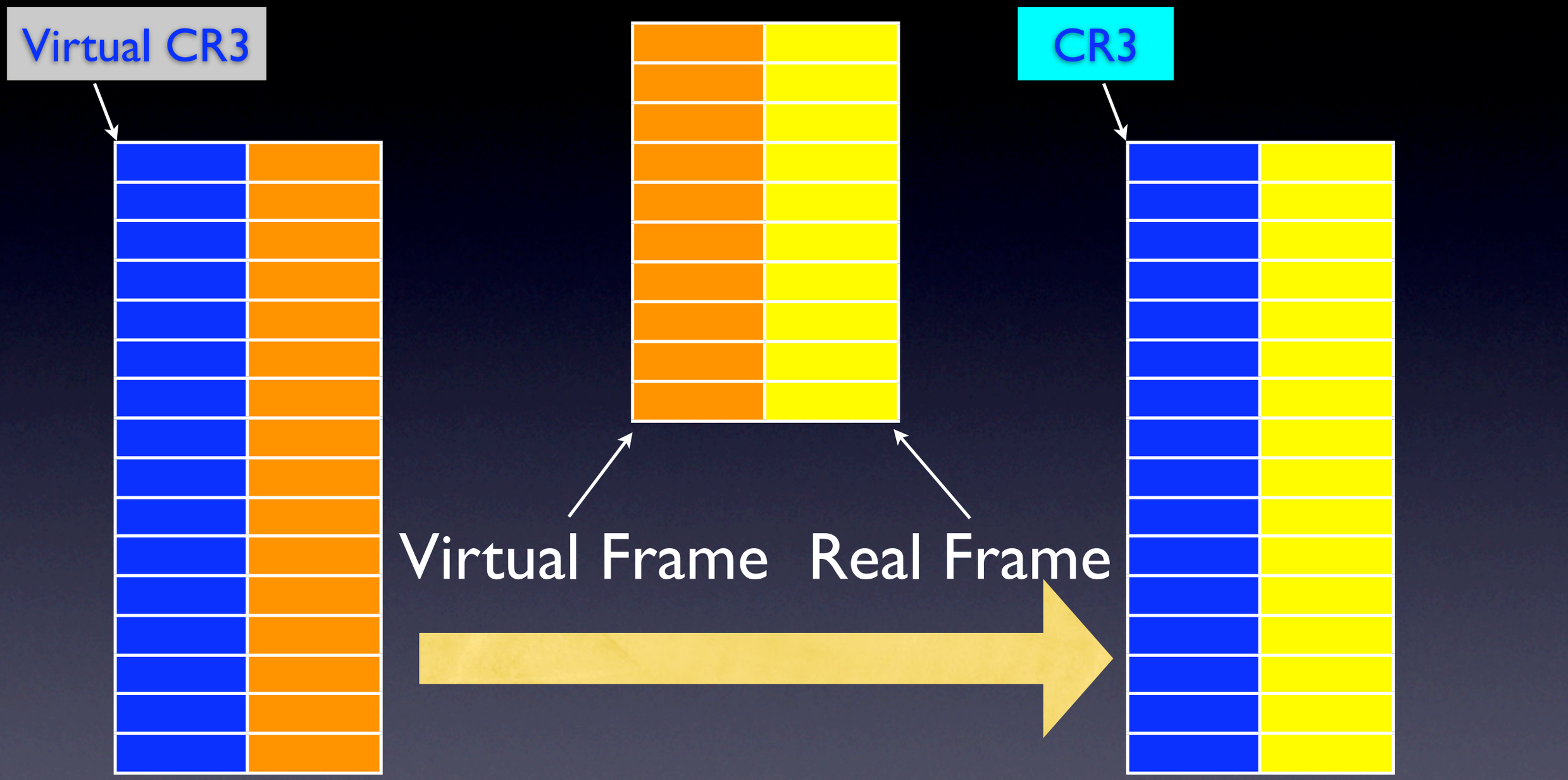
Guest Page Table

Shadowed Page Table

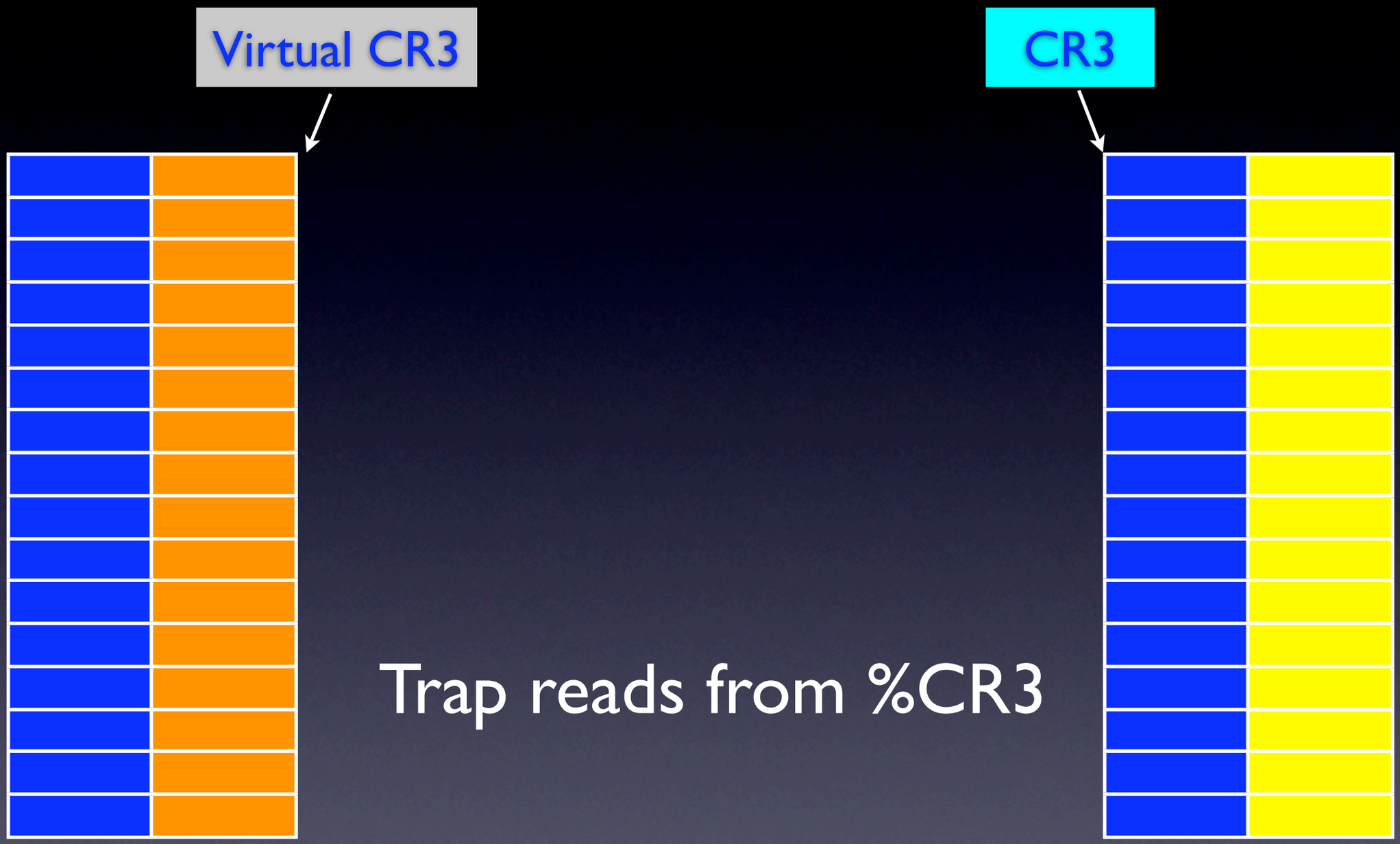
VMM makes shadowed page tables



VMM makes shadowed page tables

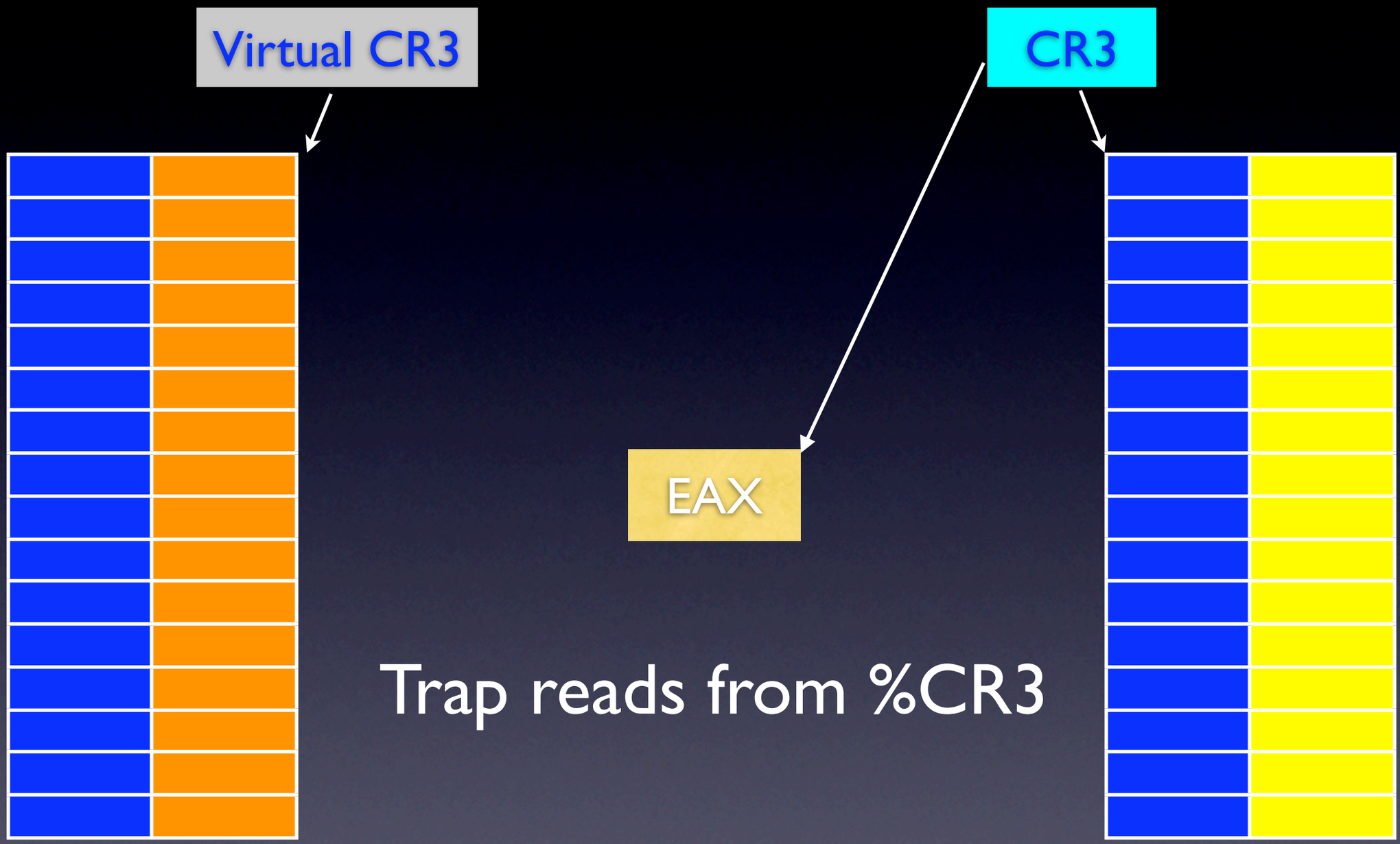


Shadowed page tables are hidden from the guest



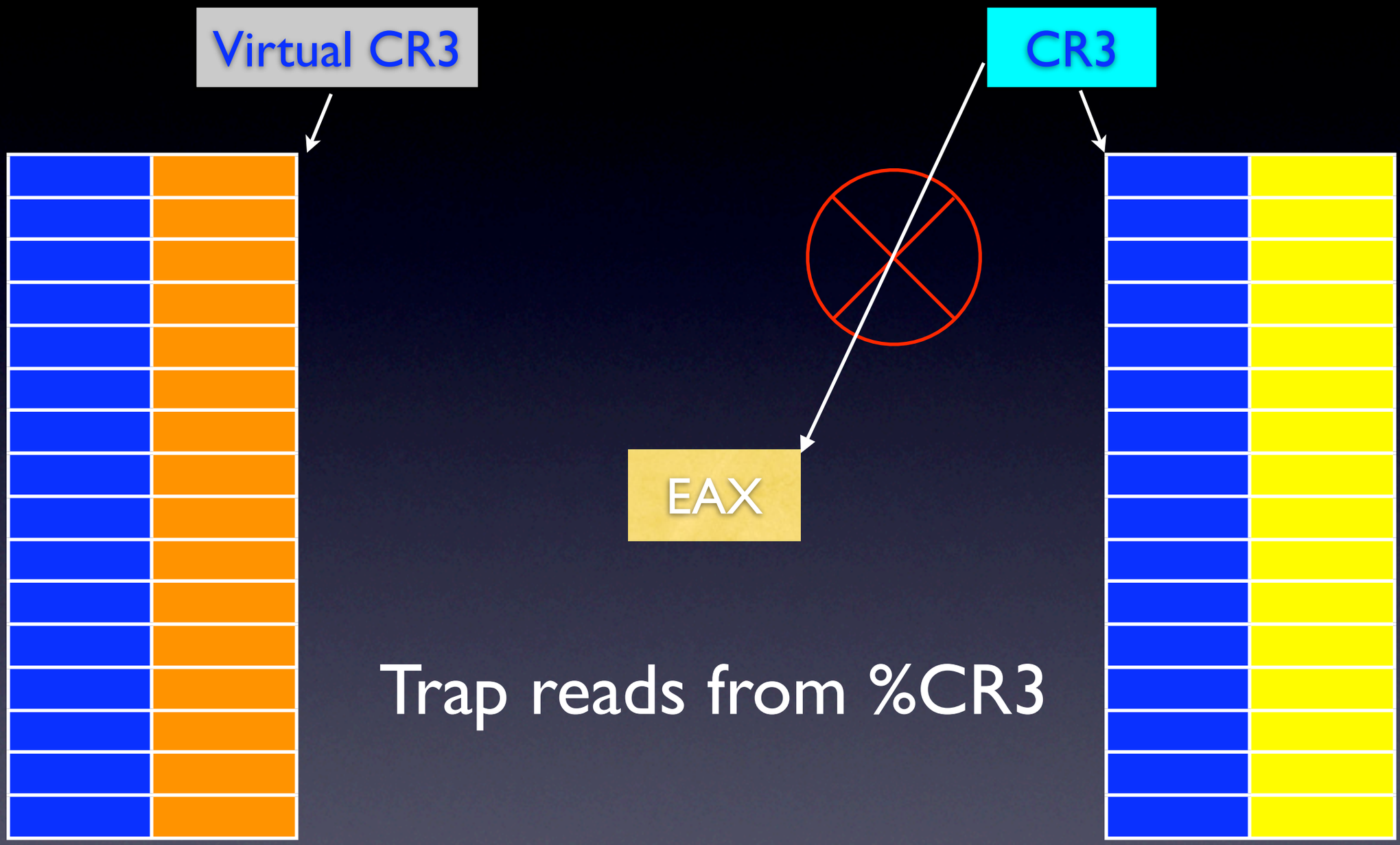
Trap reads from %CR3

Shadowed page tables are hidden from the guest



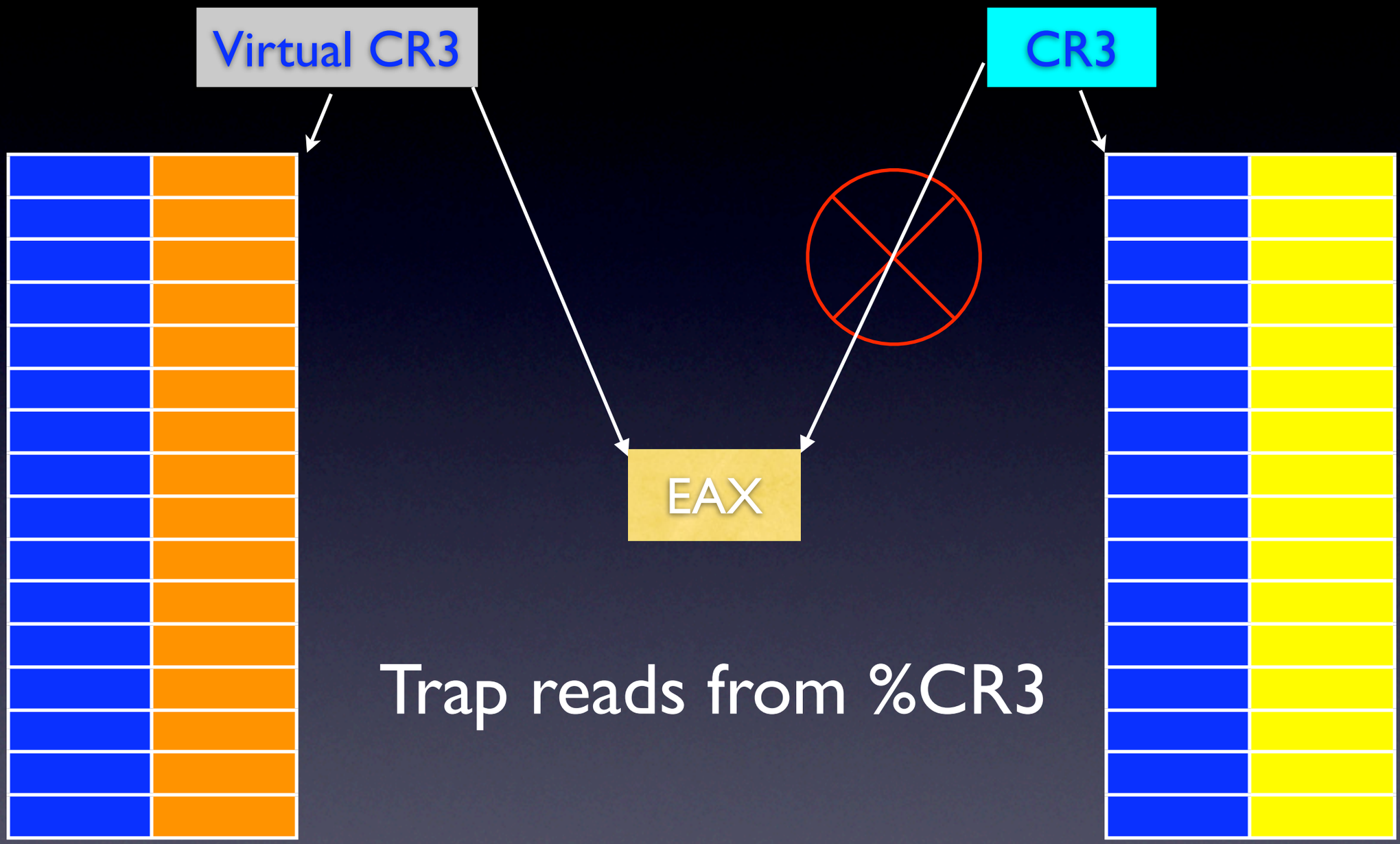
Trap reads from %CR3

Shadowed page tables are hidden from the guest



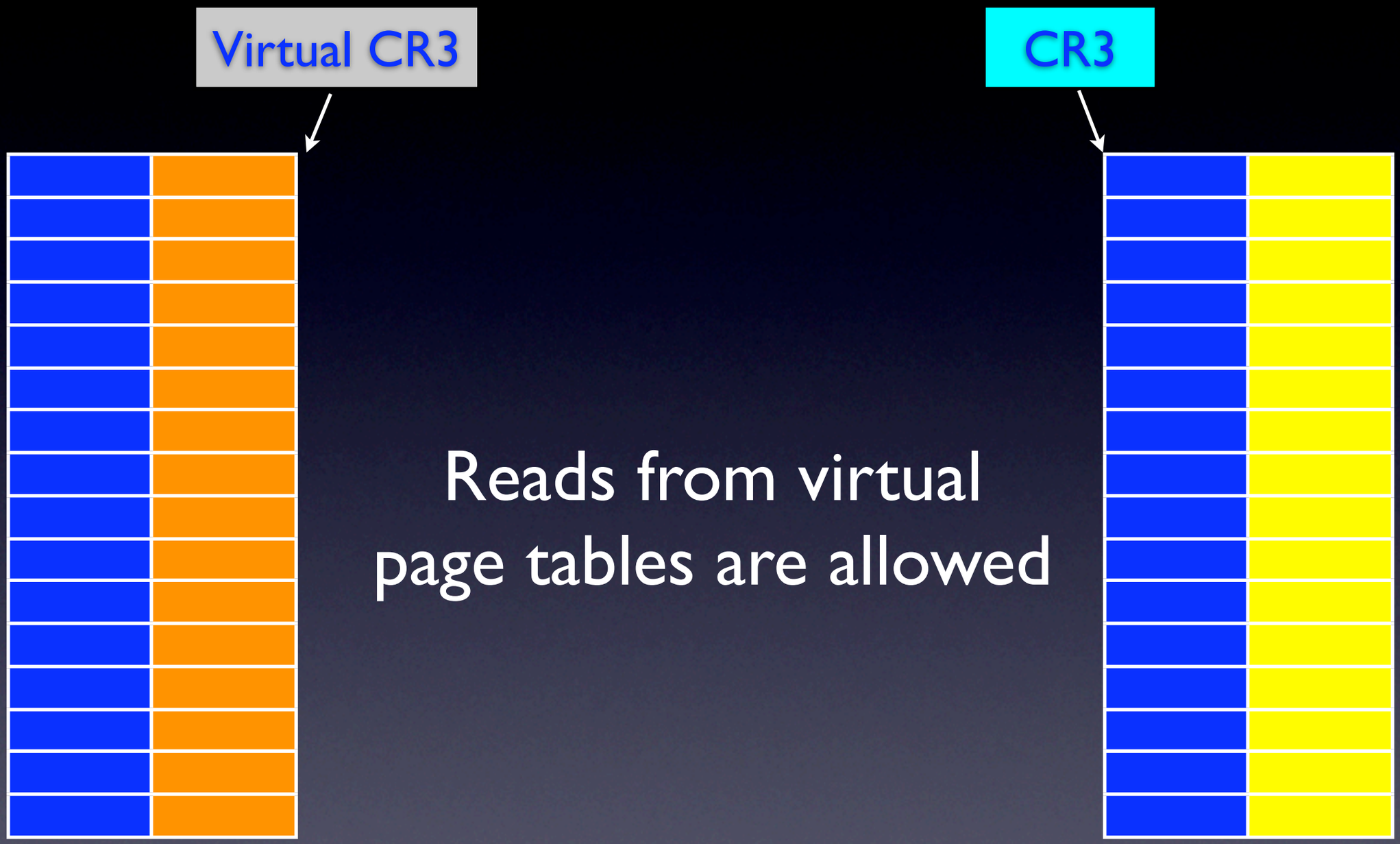
Trap reads from %CR3

Shadowed page tables are hidden from the guest

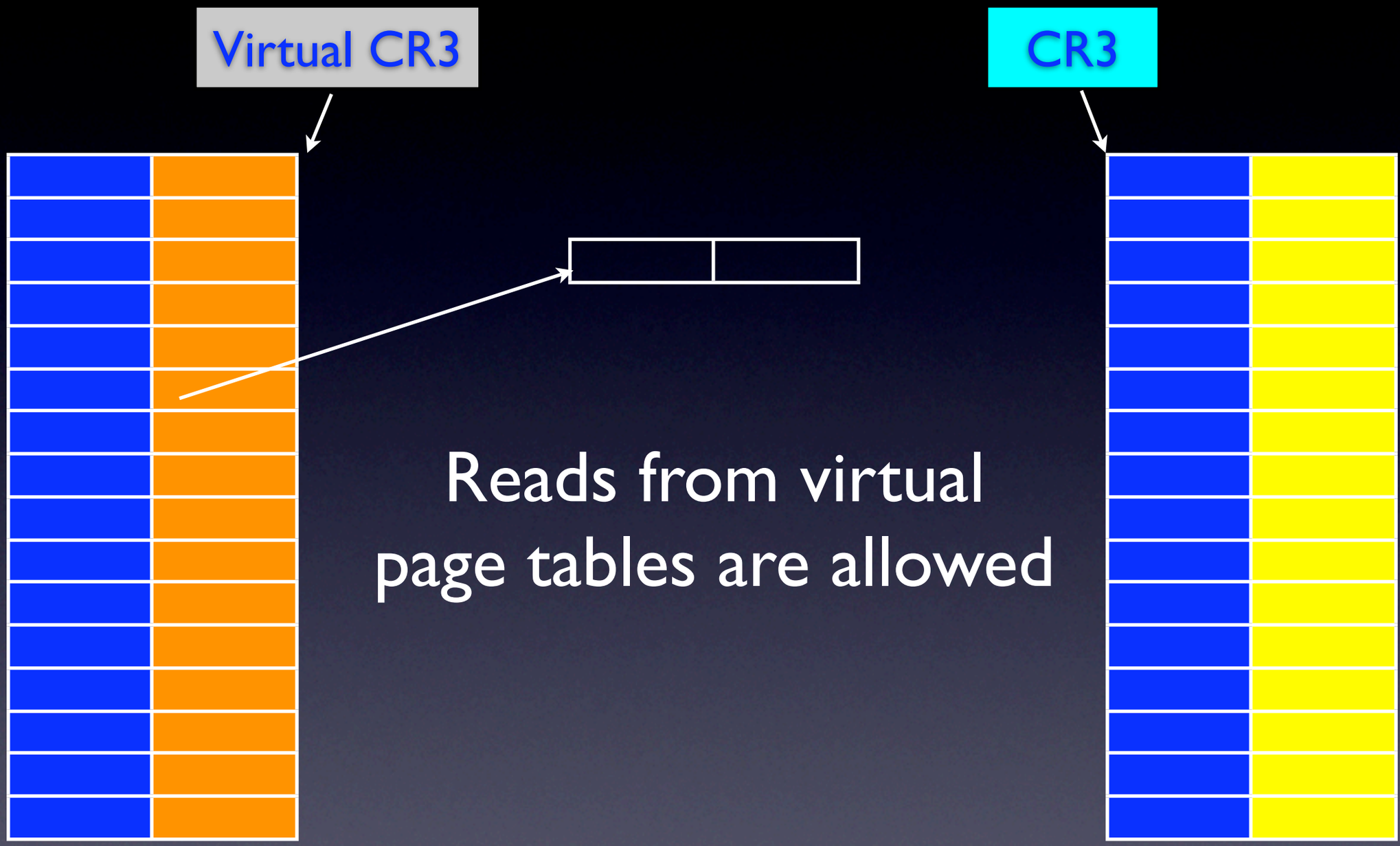


Trap reads from %CR3

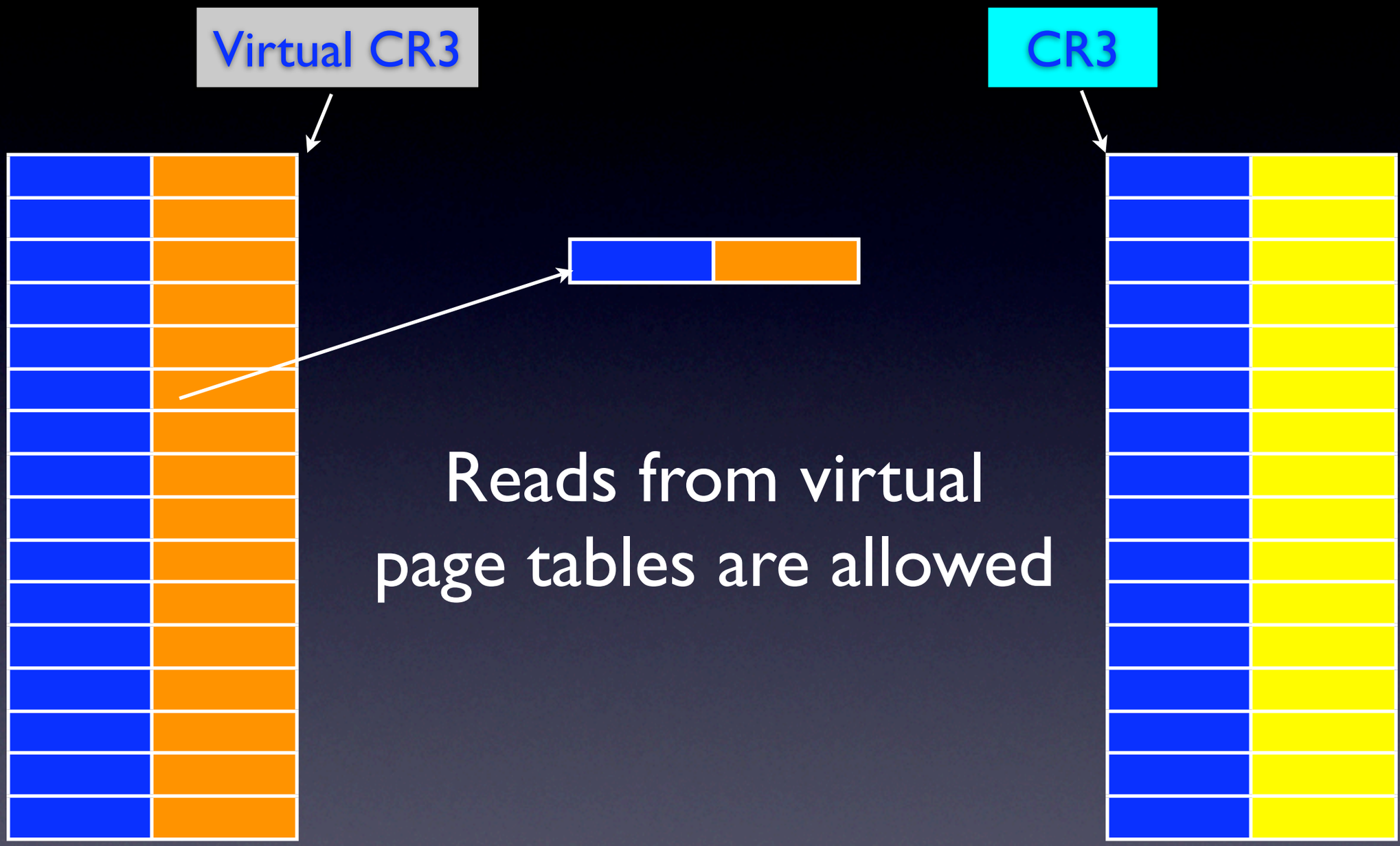
Guest gets a consistent view of the its page tables



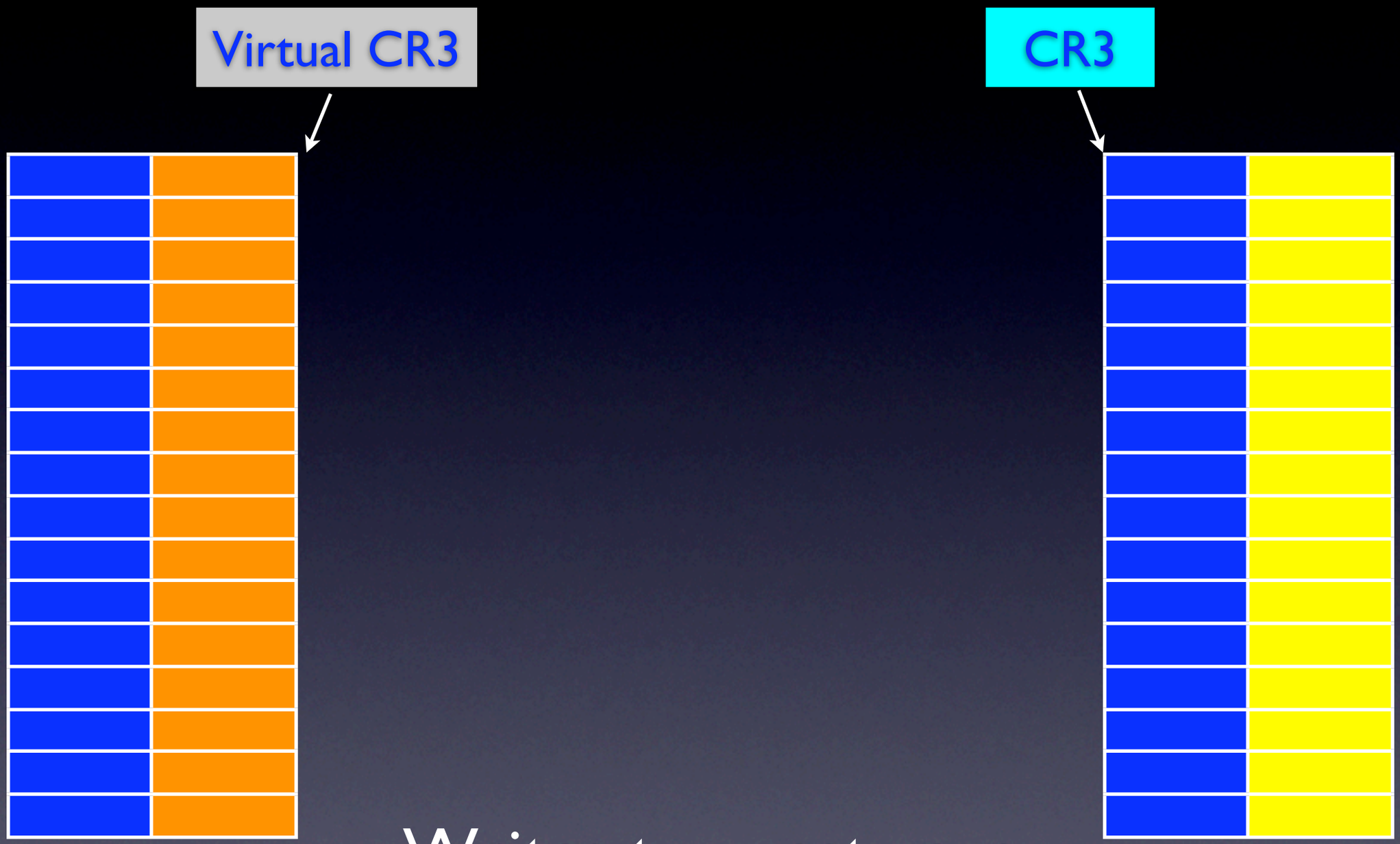
Guest gets a consistent view of the its page tables



Guest gets a consistent view of the its page tables

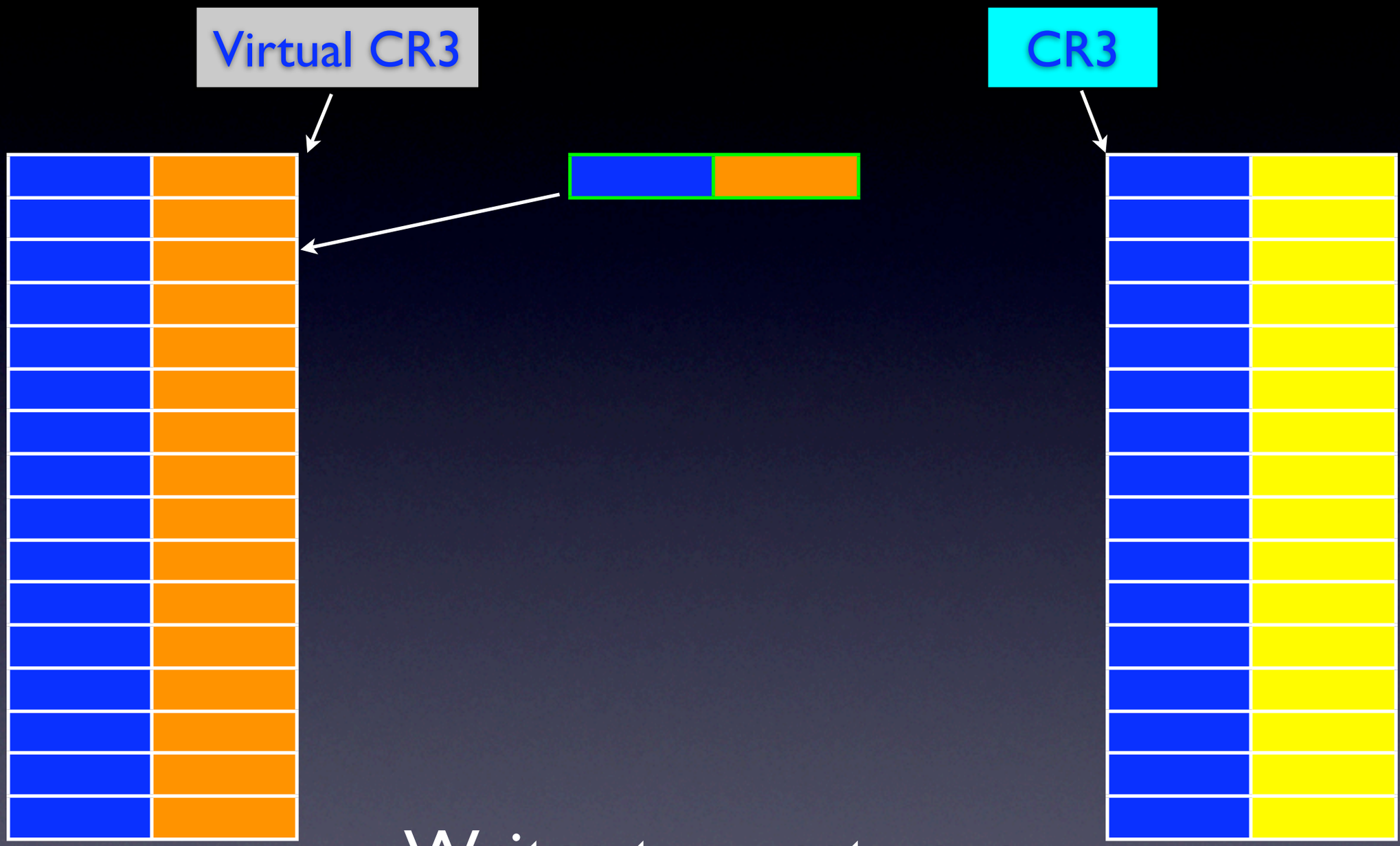


VMM traces guest page table updates



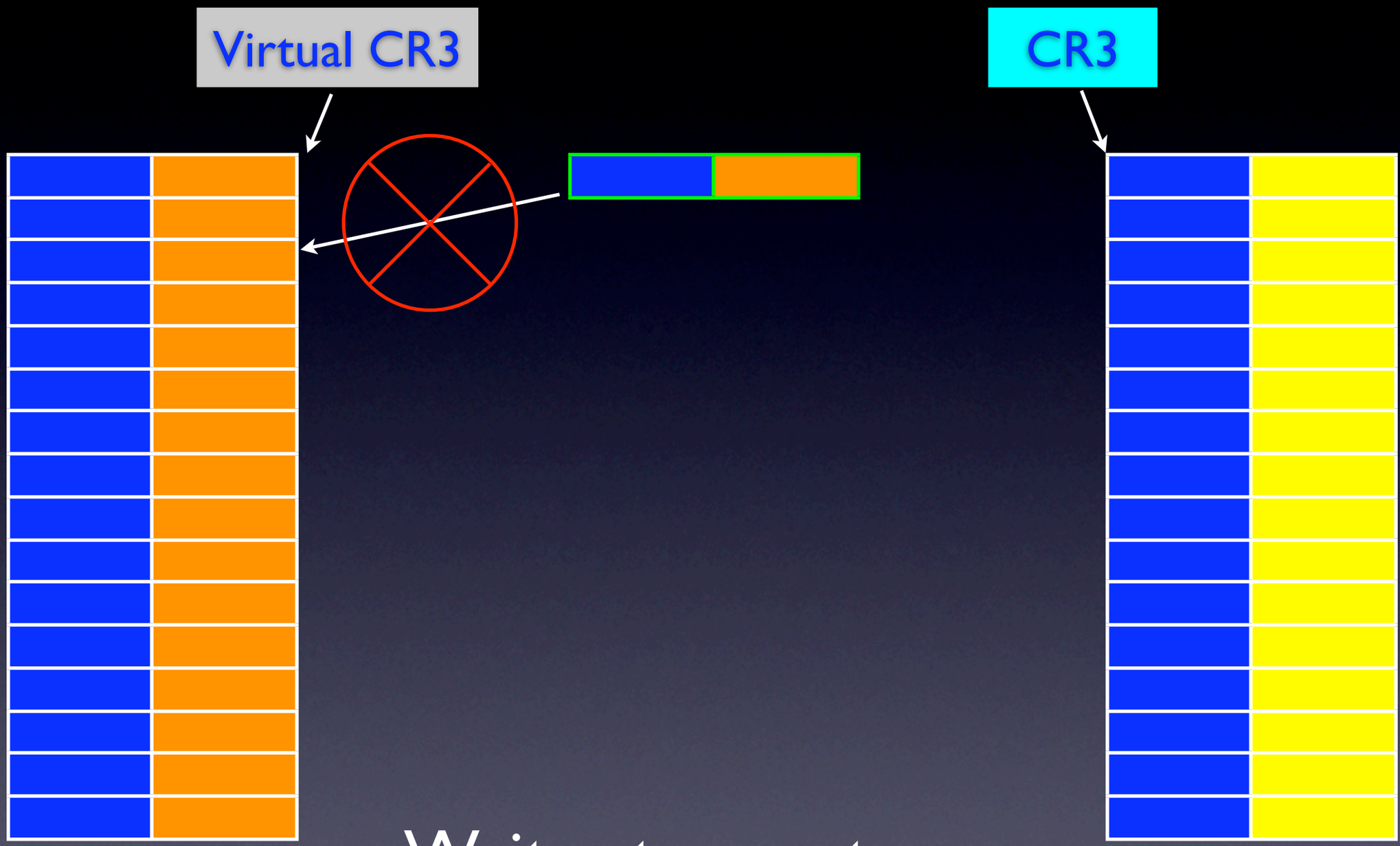
Writes to guest page tables are trapped

VMM traces guest page table updates



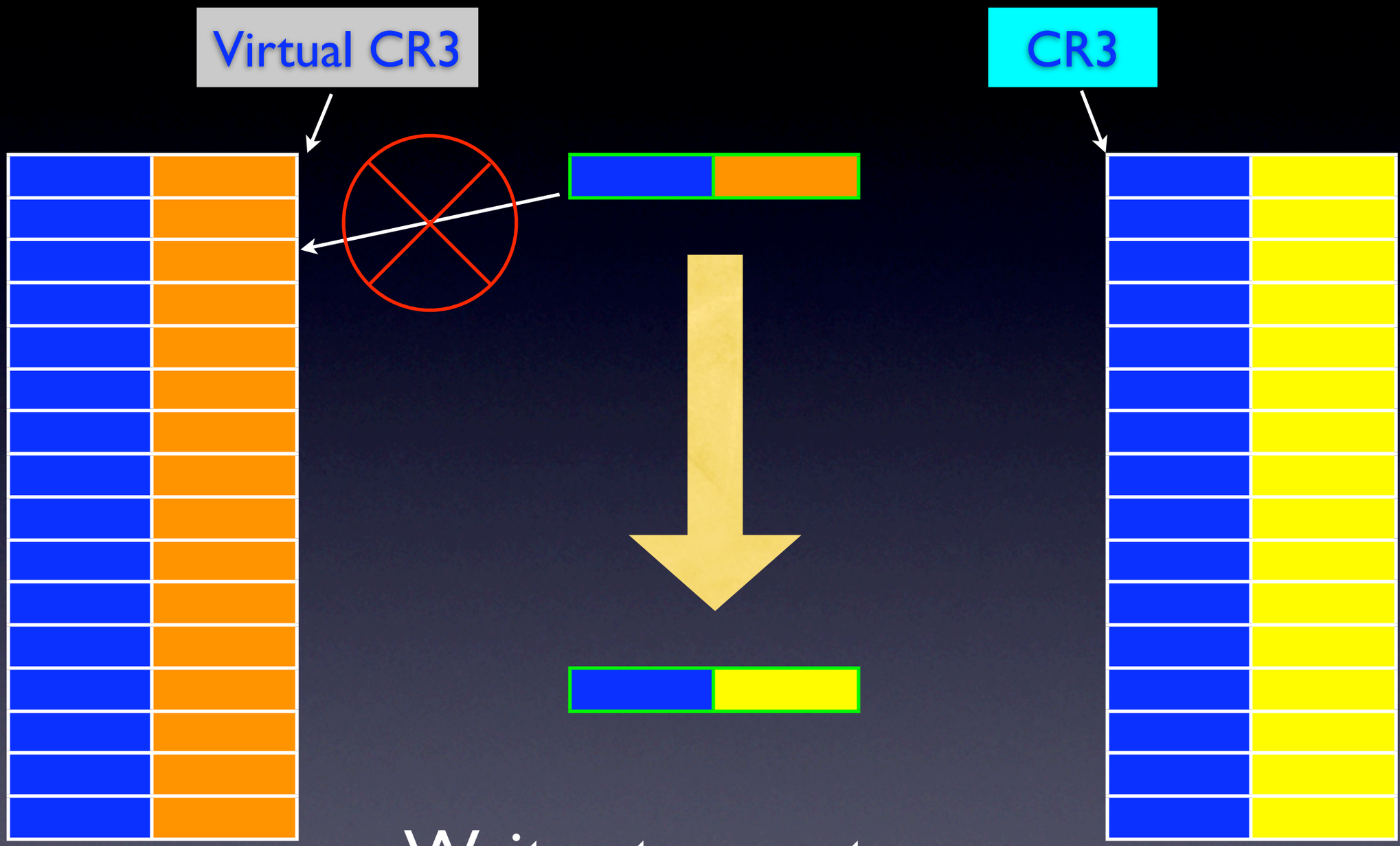
Writes to guest page tables are trapped

VMM traces guest page table updates



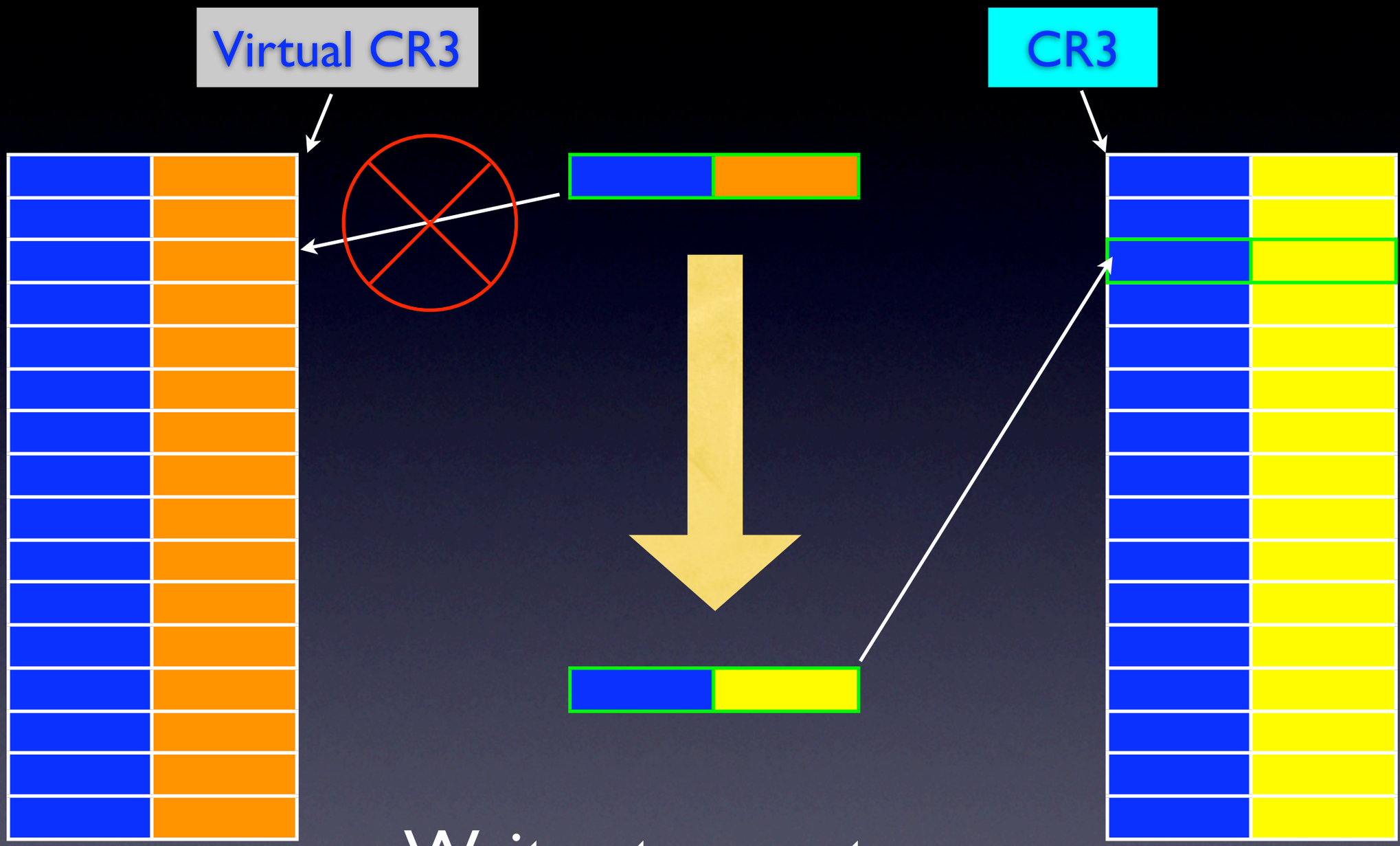
Writes to guest page tables are trapped

VMM traces guest page table updates



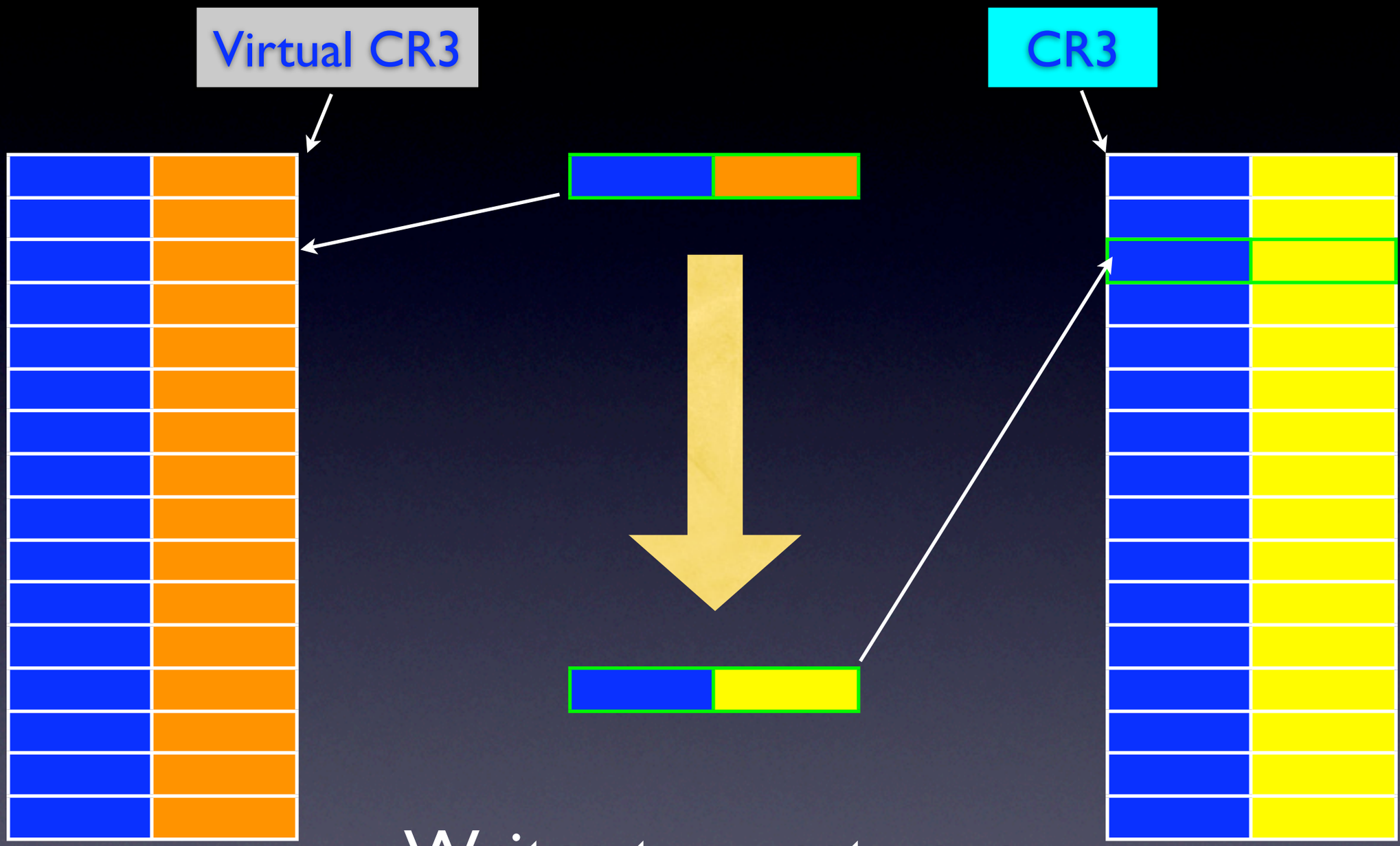
Writes to guest page tables are trapped

VMM traces guest page table updates



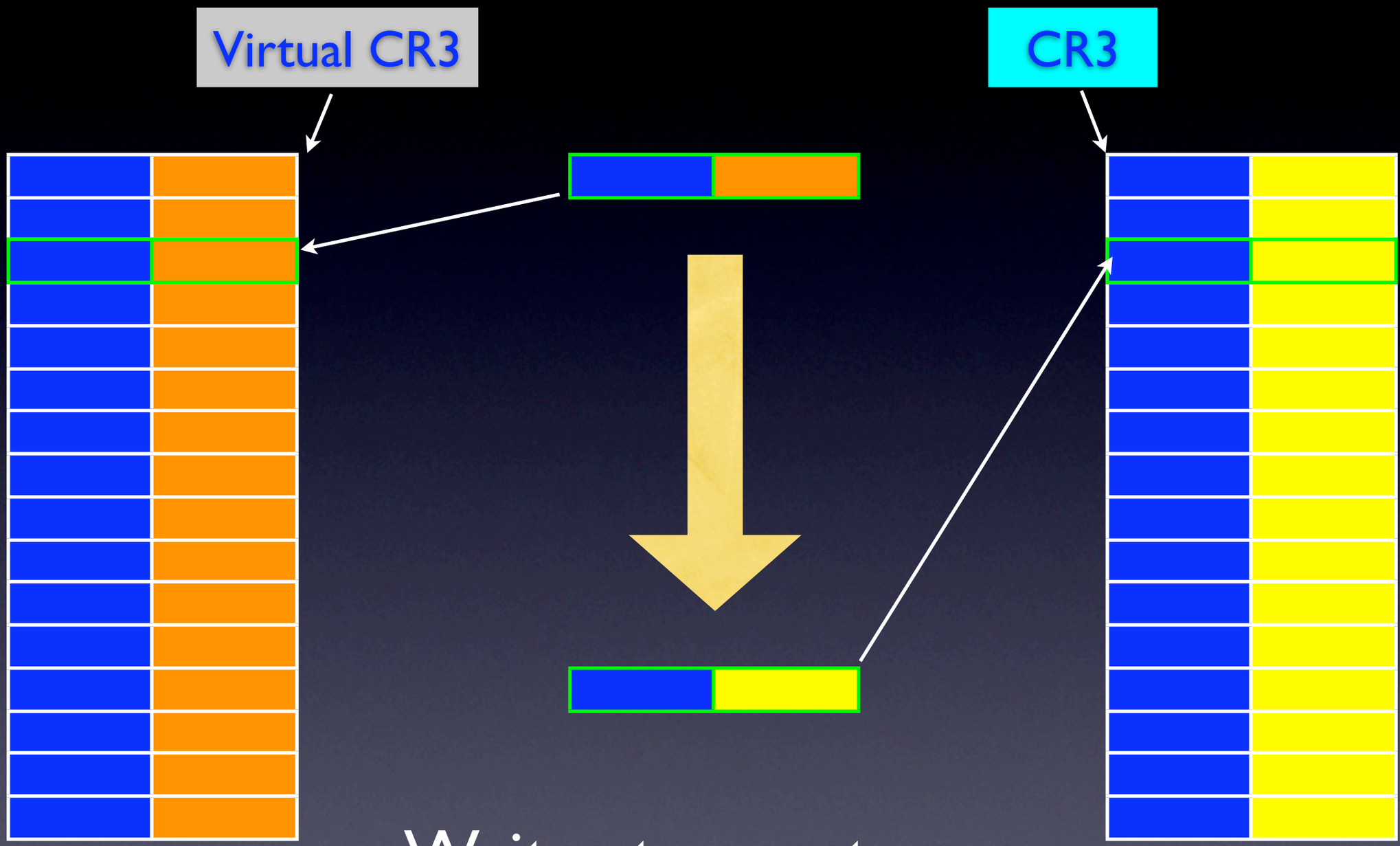
Writes to guest page tables are trapped

VMM traces guest page table updates



Writes to guest page tables are trapped

VMM traces guest page table updates



Writes to guest page tables are trapped

Shadowing

- Keep two copies of every protected state
  - Shadowed copy, what the processor sees
  - Virtual copy, what the guest sees
- Make sure guest sees only the virtual copy
- Do this for IDT, GDT, and control registers
- Cache shadowed objects for performance

Tracing

- Trace writes to virtual state by trapping
  - Writes control registers trap automatically
  - Use memory protection to trap writes to memory
    - Problem?
- Update shadowed state in trap handler

Protecting the VMM

- VMM lives in the same virtual address space as guest
  - Guest does not know about the VMM
  - Guest may need to use the same address space
- VMM lives in the shadow
  - Guest access to VMM space are emulated

Project 4 Option

- Write a trap-and-emulate VMM
- Be able to run two instances of your P3 kernel inside VMs
- We will provide an x86 instruction decoder

Just Kidding!

x86 Hardware

- x86 ISA is not "virtualizable"
- 17 privileged instructions behave differently in non-privileged mode
  - Doesn't trap when you want it to
  - Doesn't do the right thing if you don't want it to trap
  - e.g. POPF just ignores the IF flag

Virtualization on x86

- Possible, through a lot of clever hacks
- VMware (1998)
  - Dynamically rewrites guest kernel instructions to properly trap into VMM
  - Directly execute guest user code
  - Only 20% performance overhead, or less

Paravirtualization Motivation

- Full virtualization is expensive and complicated
- If guest OS can be modified to work with the hypervisor, then it's unnecessary to
  - Trap and emulate
  - Shadow and trace
  - Dynamically recompile guest kernel code

Paravirtualization Implementation

Guest OSes are modified to accept the fact it is running inside a VM.

The VMM does not create an *illusion* that the VM owns all machine resources.

Instead, the VMM provides an *hypercall interface* to provide service to VMs.

Hypercall Interface

- Allows the guest to *voluntarily* trap into the hypervisor
- All guest access to hardware state happens through hypercalls
- The guest does not access hardware state directly at all
- No instruction decoding necessary

Physical Memory

- Guest knows that it does not own all of physical memory
- Guest requests physical memory from hypervisor
- There is no distinction between virtual frame address and real frame address

Virtual Memory

- Guest relinquishes ownership of its own page tables
- All paging operations happen through hypercalls
  - mmu_update() - batch update
  - update_va_mapping() - update one entry
- No shadows or traces

Hardware Abstraction

Paravirtualization = Hardware abstraction

The hypervisor abstracts away all the hardware details from the guest operating system

Therefore, it's only natural that you should have more than one guest operating system

Hypervisor Examples

- Xen - portable hypervisor
  - Hypercall interface defined for *any* ISA
    - mmu_update()
- VMware - x86 specific
  - Hypercall interface similar to hardware
    - VMI_SetCR3()

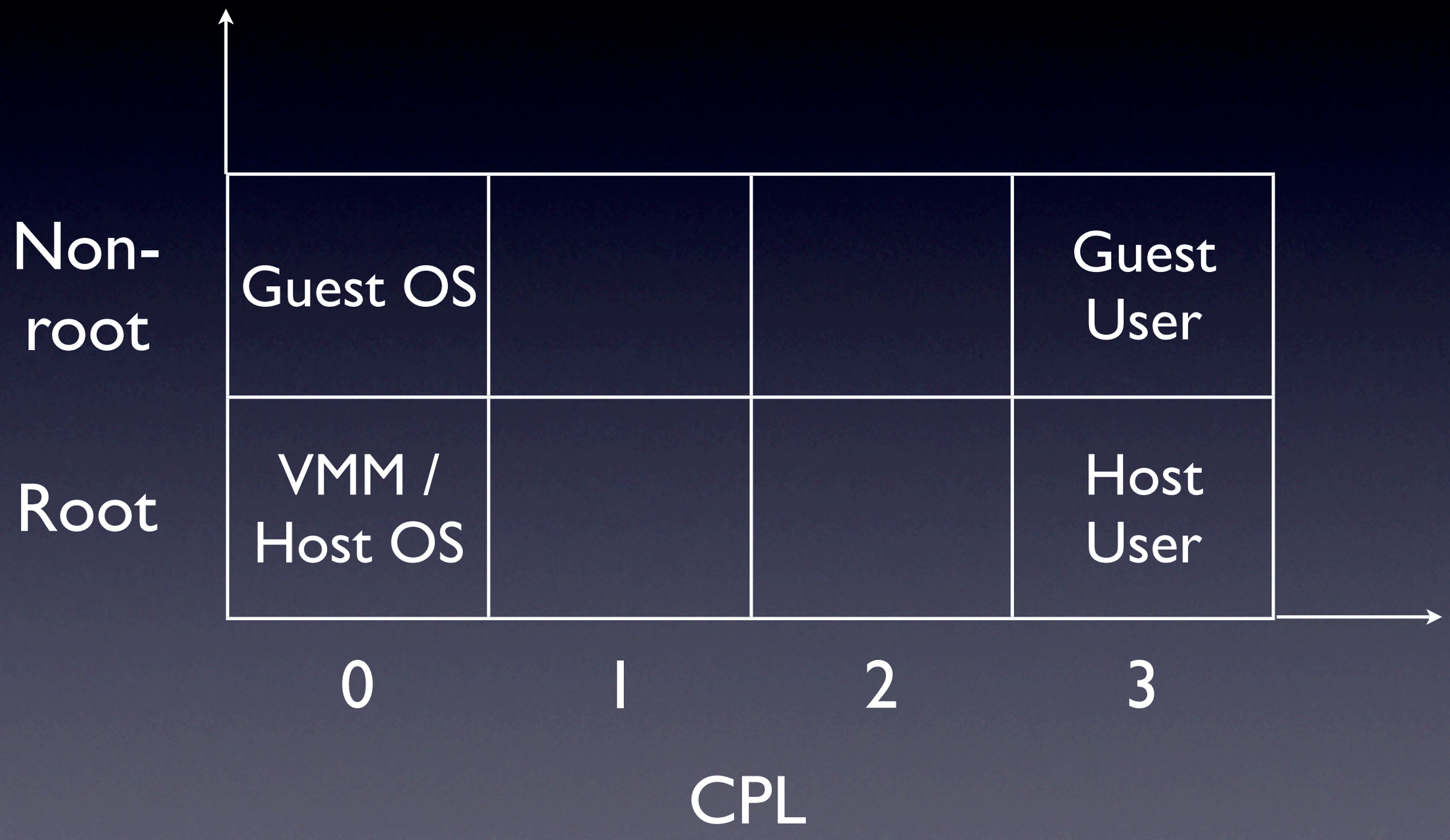
Hardware Assisted Virtualization

- Intel VT (Vanderpool) - Core Duo/Solo
- AMD SVM (Pacifica) - soon to come
- Provides a hardware mode of operation for virtual machines
- Architectural extension to make x86 virtualization *easier*
  - Does not *replace* the VMM

Modes of Operations

- Root and non-root mode, orthogonal to privilege levels
- Root mode - VMM, or regular OS
  - Full access, normal operation
- Non-root mode - Guest
  - Less “privileged” than root mode, allows trapping into root mode

Modes of Operations



Guest OS in its Own World

- Guest lives in its own address space
  - Does not conflict with VMM
- Guest kernel runs at CPL 0
  - No unnecessary traps
  - Can have its own interrupt handlers
  - Those 17 instructions do the right things

Mode Switching

- Unconditional traps to root mode
  - VM extension instructions
  - Read from %CR3
- Conditional traps, defined by VMM
  - Write to control registers
  - Certain interrupts/exceptions

Control Register Shadowing

- In non-root mode, control registers are shadowed by the processor
- VMM defines bit mask for shadowed bits, and a bit mask of virtual values
- Instructions that reads control registers return virtual values

Control Register Tracing

- Not all writes trap, VMM defines which
  - For example, VMM can specify a list of up to four legal %CR3 values
- Legal writes execute directly
- Illegal writes trap
- (Why is this useful?)

Virtual Memory Virtualization

- No hardware support (yet)
- VMM is still responsible for shadowing/ tracing page tables
  - Explains why read from %CR3 unconditionally traps

Virtualization Software

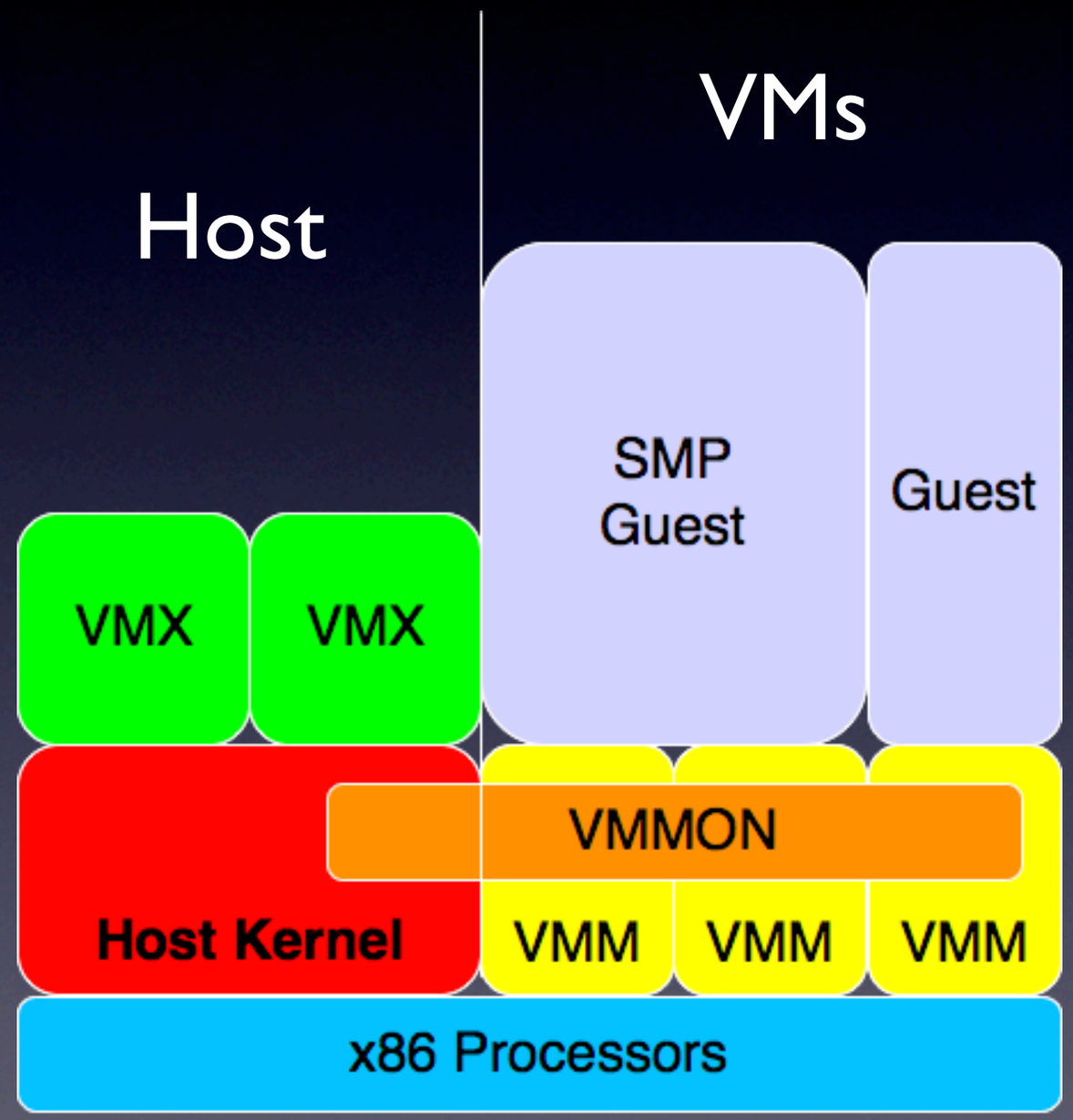
- VMM can't really be an operating system
  - It has to be *small* so it can't have device drivers, filesystem, and network stack
- Users want to have virtual machines on top of existing operating systems
  - But the VMM needs to take control of the hardware when it runs

As a User Application

- User mode part
  - User interface
  - Provides device emulation services
  - Allocate host resource to VMs
- Kernel mode part
  - Driver to load VMM and kick host out

VMware Workstation

- VMX - user app
- VMMON - kernel driver
- One VMM per virtual processor
- Host schedules VMX
- VMX schedules VMM
- VMM host-independent



Device Emulation

- VMM traps guest access to virtual devices
- VMM sets up a device service request, and then switches the host OS
- Host OS returns to VMX
- VMX serves the device request, possibly making host OS system calls
- 4 context switches

Device Emulation Optimization

- Avoid context switching
  - Queue up device emulation requests
  - Switch to back to host only when absolutely necessary, e.g. timer
- Do not context switch at all
  - Make host cooperate

“Bare-Metal” Virtualization

- VMware ESX Server
  - Custom host kernel that allows the VMM to call into it directly
  - Fancy scheduler optimized for VMs
  - Special file system optimized for VM disk images
  - As little as 5% performance overhead!

Summary

- Full Virtualization
  - Trap and emulate, shadow and trace
- Paravirtualization
  - Voluntary trap, hardware abstration
- Hardware Assisted Virtualization
- Virtualization on a Host OS

Further Reading

- J.S. Robin, and C.E. Irvine, "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor"
- M. Rosenblum, and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends"
- S. Devine, E. Bugnion, and M. Rosenblum, "Virtualization system including a virtual machine monitor for a computer with a segmented architecture", US Patent 6,397,242
- P. Barham, et. al., "Xen and the Art of Virtualization"
- "Xen Developer's Reference"
- "VMI Specification" from VMware
- R. Uhlig, et. al., "Intel Virtualization Technology"
- "Intel VT Specification"
- M. Rosenblum, et. al., "Optimizing the Migration of Virtual Computers"