Virtual Machines
CS 111
Summer 2025
Operating System Principles
Peter Reiher

Outline

- What is a virtual machine?
- Why do we want them?
- How do virtual machines work?
- Issues in virtual machines

What Is a Virtual Machine?

- Remember, in CS, “virtual” means ”not real”
  - But it looks like it’s real
- So a virtual machine isn’t really a machine
  - But it looks like a machine
- What do we mean by that?
- A *virtual machine* is a software illusion meant to appear to be a real machine
- Virtual machines abbreviated as VMs

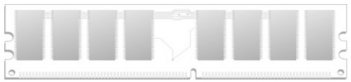
What's That Really Mean?

- We have an actual computer
- We do something in software to make it look like we have multiple computers
  - Or that it's a different kind of computer
  - Making use of the actual computer to do so
- The virtual machine must appear to apps and users to be a real machine

Graphically, . . .

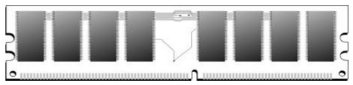
We implement a virtual server on the real hardware

With a set of virtual components



We have a real server computer

With a real CPU



And real RAM

And real peripherals

How?

- Use the real hardware to implement the virtual hardware
  - Instructions for the virtual CPU run on the real CPU
  - Real RAM stores the data for virtual RAM
  - A real disk stores data for the virtual disk
  - Etc.
- But to what purpose?

Why Do We Want Virtual Machines?

- For several reasons
  - Fault isolation
  - Better security
  - To use a different operating system
  - To provide better controlled sharing of the hardware
- Let's consider each reason separately

Fault Isolation

- Operating systems must never crash
  - Since they take everything down with them
- But crashing a virtual machine's operating system need not take down the entire machine
  - Just the virtual machine
- So our correctness requirements can be relaxed
- Similar advantages for faults that could damage devices
  - They damage the virtual device, not the physical

Better Security

- The OS is supposed to provide security for processes
- But the OS also provides shared resources
  - Such as the file system and IPC channels
- A virtual machine need not see the real shared resource
- So processes in other virtual machines are harder to reach and possibly damage

Using a Different Operating System

- Let's say you're running Windows
- But you want to run a Linux executable
- Windows has one system call interface, Linux has a different one
  - So system calls from your Linux executable won't work on Windows
- But if you have a virtual machine running Linux on top of the real machine running Windows . . .
  - Now your application can run fine
  - Assuming you get the virtualization right . . .

Sharing a Machine's Resources

- In principle, an OS can control how to share resources among processes
- But actually guaranteeing a particular allocation of resources is hard
- It's easier to guarantee an entire virtual machine gets a set allocation of resources
  - So the processes running in it will not steal resources from the other virtual machines
- A very big deal for cloud computing

How Do We Run Virtual Machines?

- Easiest if the virtual and real machine use the same ISA (Instruction Set Architecture)
  - Tricky and slow, otherwise
  - So the same ISA is the common case
- Basically, rely on limited direct execution
  - Run as much VM activity directly on the CPU as possible
  - When necessary, trap from the VM
- But trap to what?

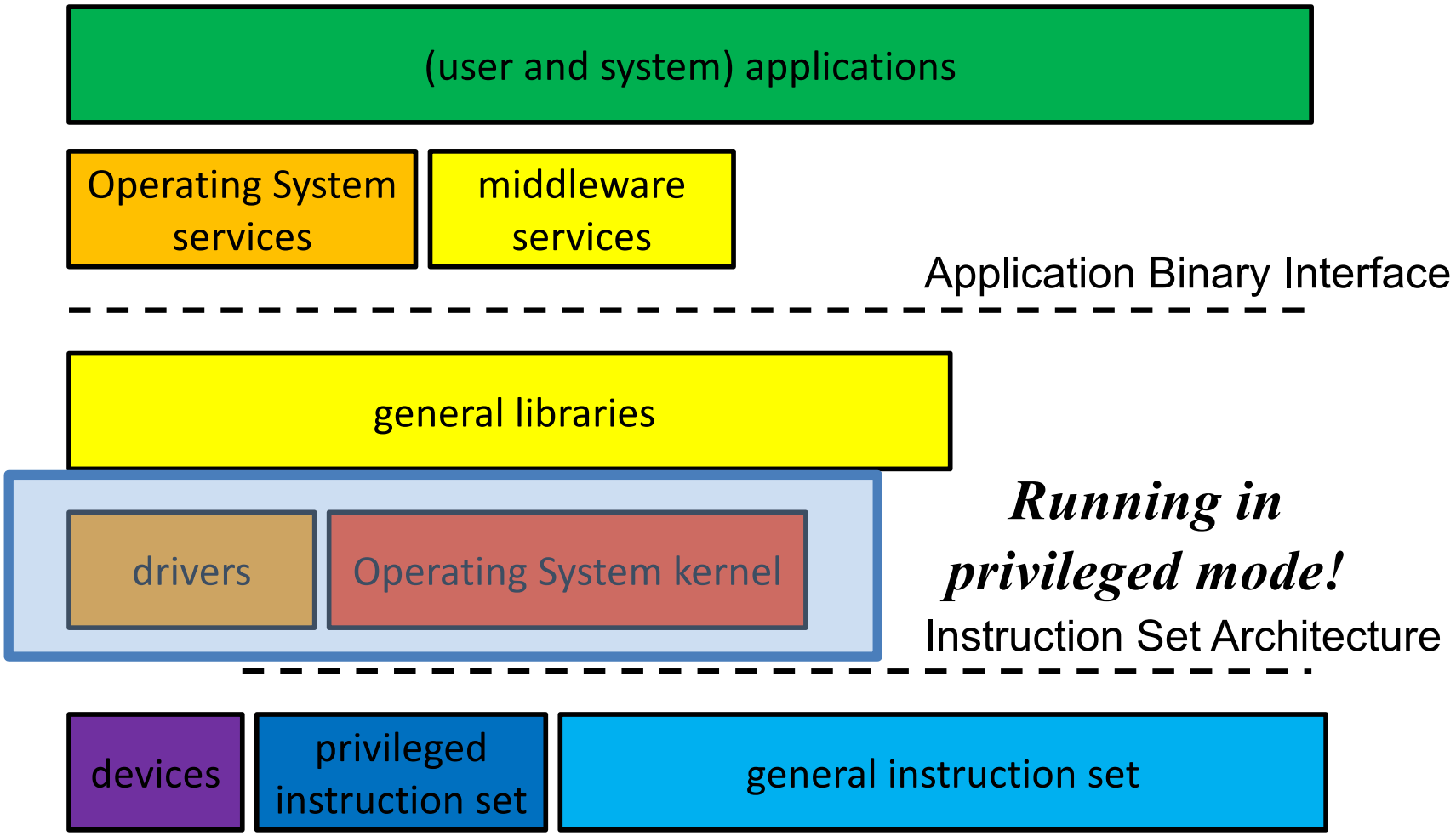
The Hypervisor

- Also known as the Virtual Machine Monitor (VMM)
- A controller that handles all virtual machines running on a real machine
- When necessary, trap from the virtual machine to the VMM
- It performs whatever magic is necessary
- And then returns to limited direct execution
- Much like a process' system call to an OS

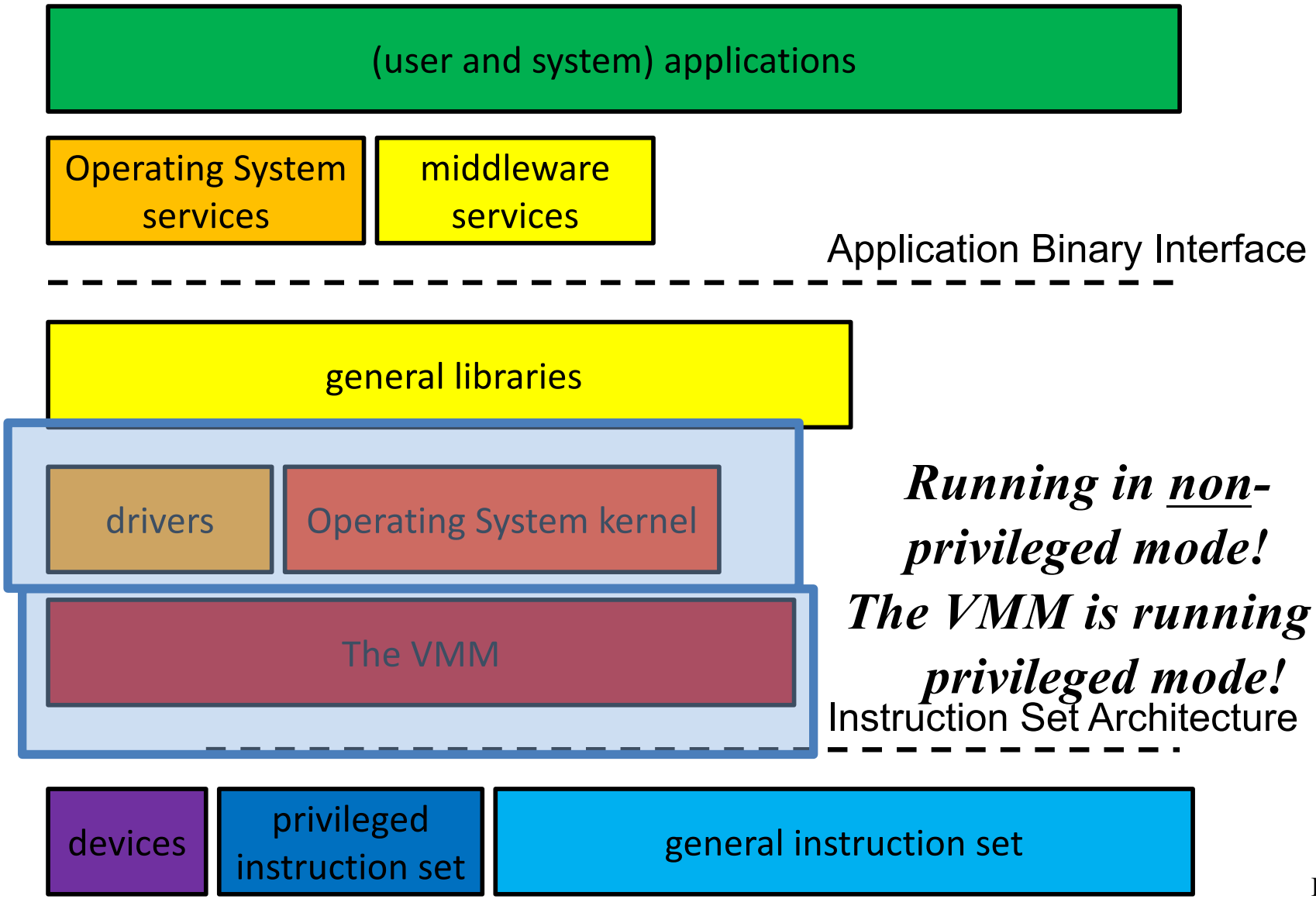
When Is Trapping to the VMM Necessary?

- Whenever the VM does something privileged
  - Kind of like trapping to the OS when a process wants to do something privileged
- The initial system call instruction will trap to the VMM
- Which will typically forward it to the VM's OS
- But subsequent privileged operations trap back to the VMM

The Old Architecture



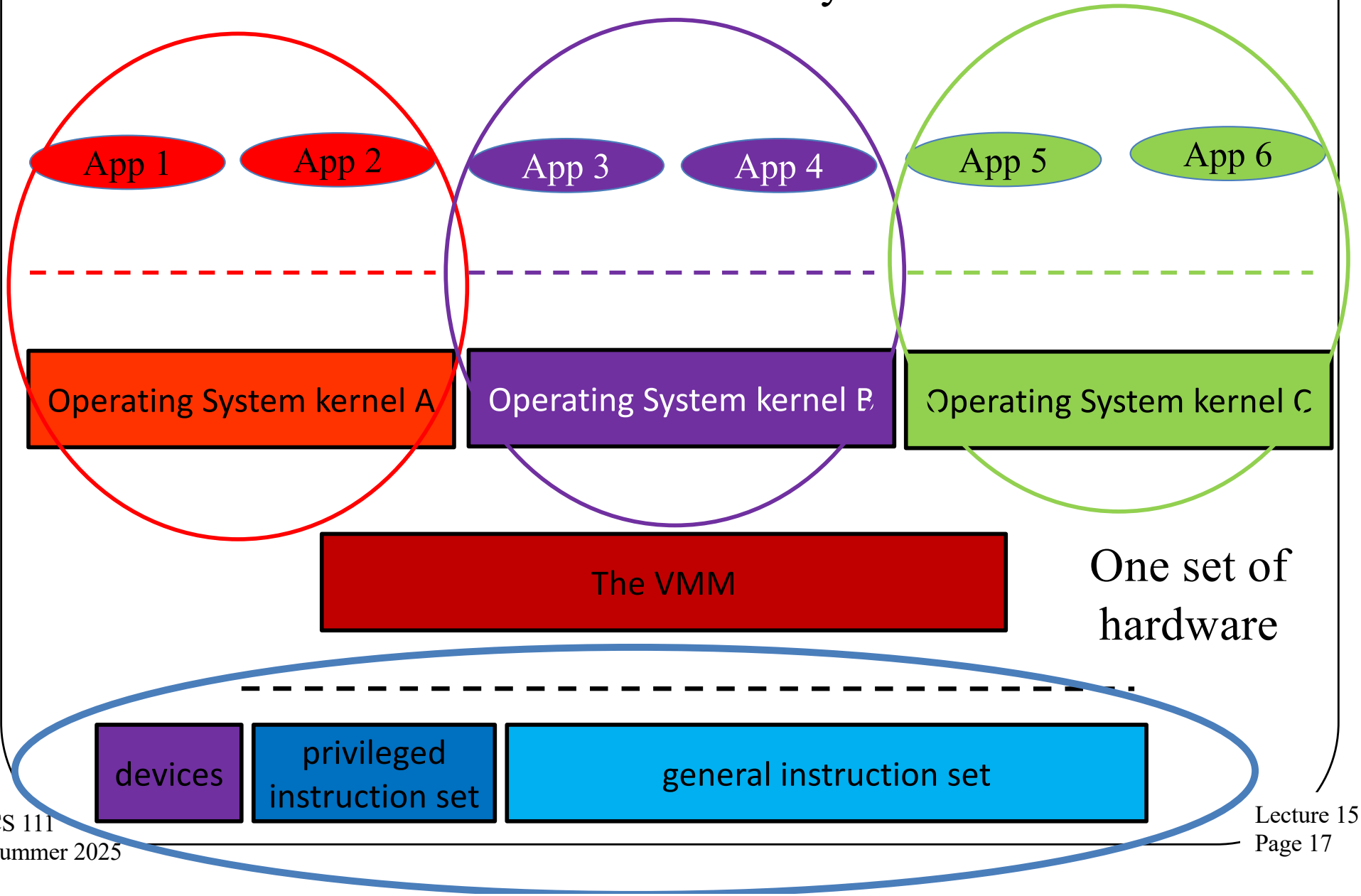
Architecture With a VMM



A More Complex Case

Three virtual machines

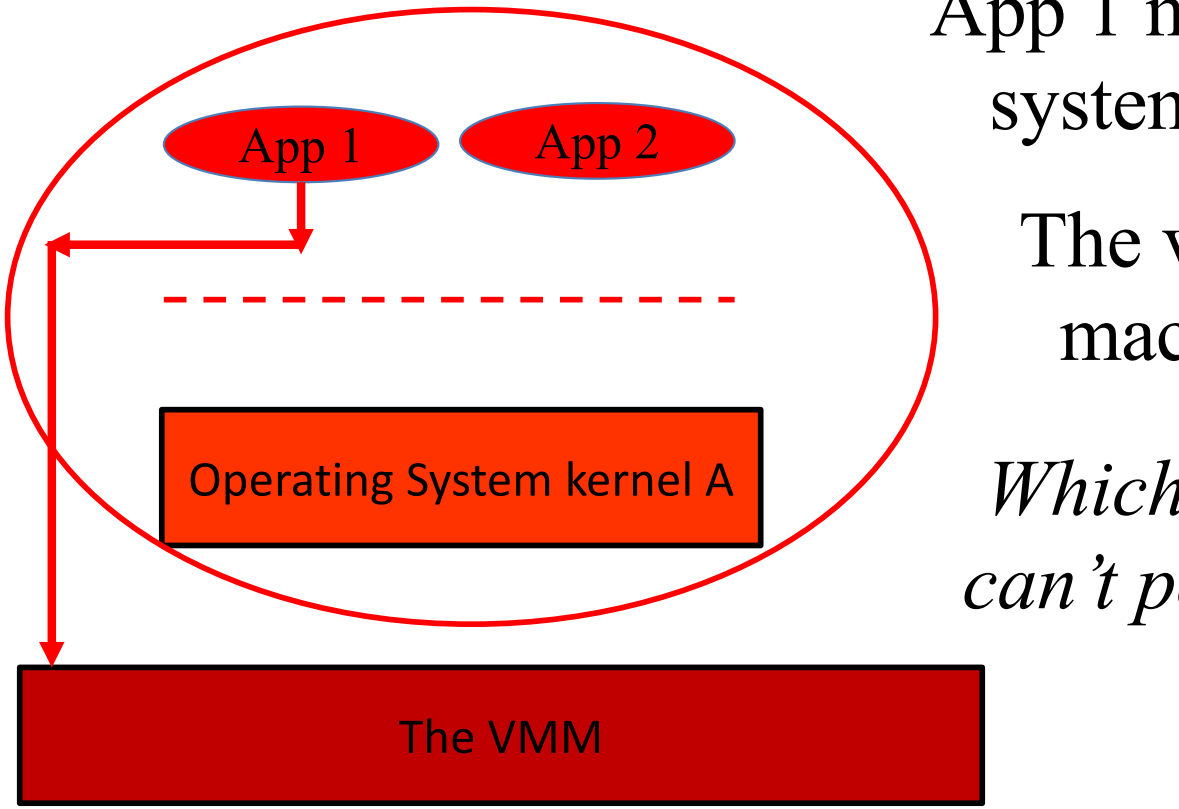
With three system call interfaces



How Do System Calls Work Now?

Using a privileged instruction

It's sent to the VMM instead



App 1 makes a system call

The virtual machine

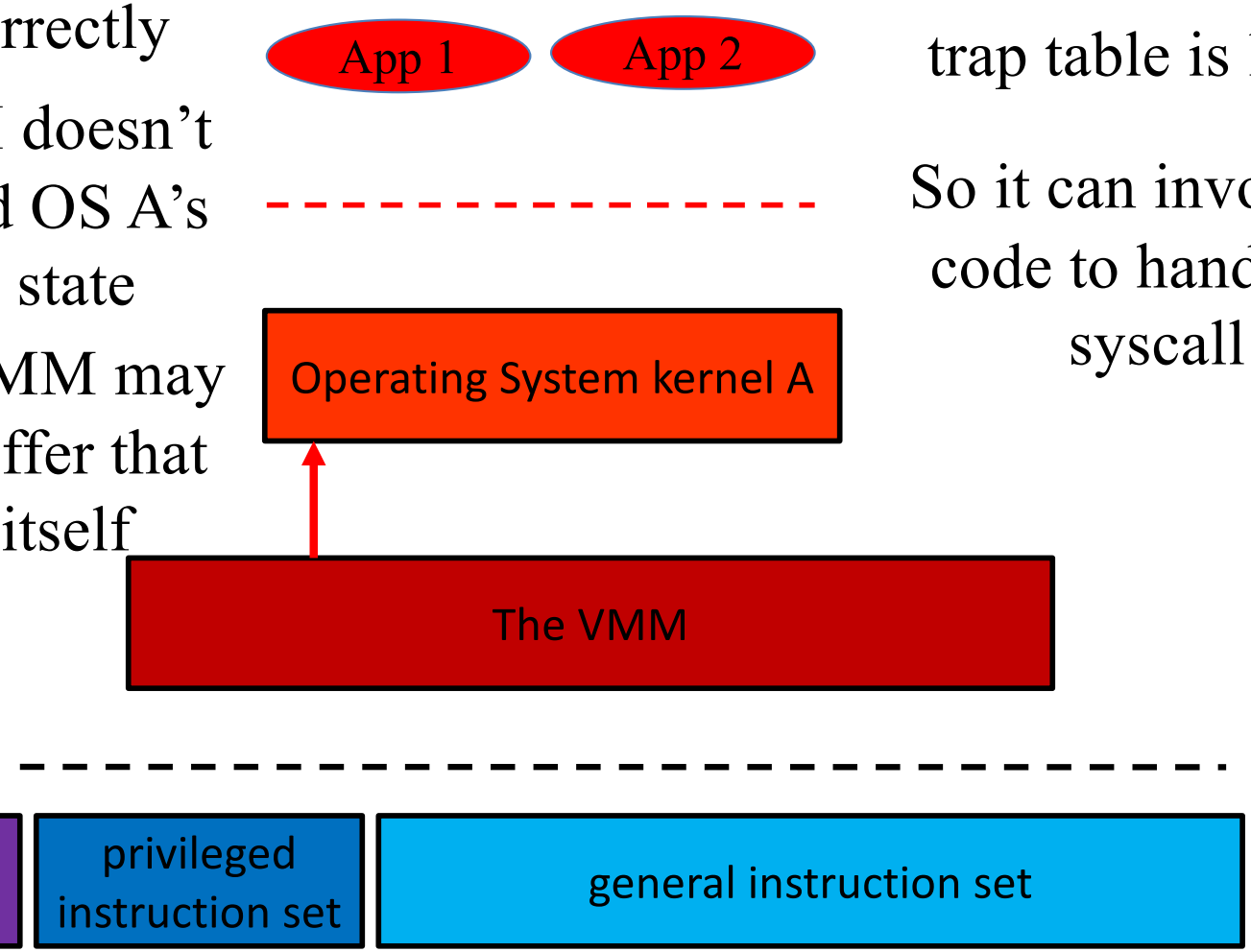
Which OS A can't perform

Yeah, But . . .

The VMM can't perform the system call correctly

The VMM doesn't understand OS A's internal state

And the VMM may not even offer that syscall itself



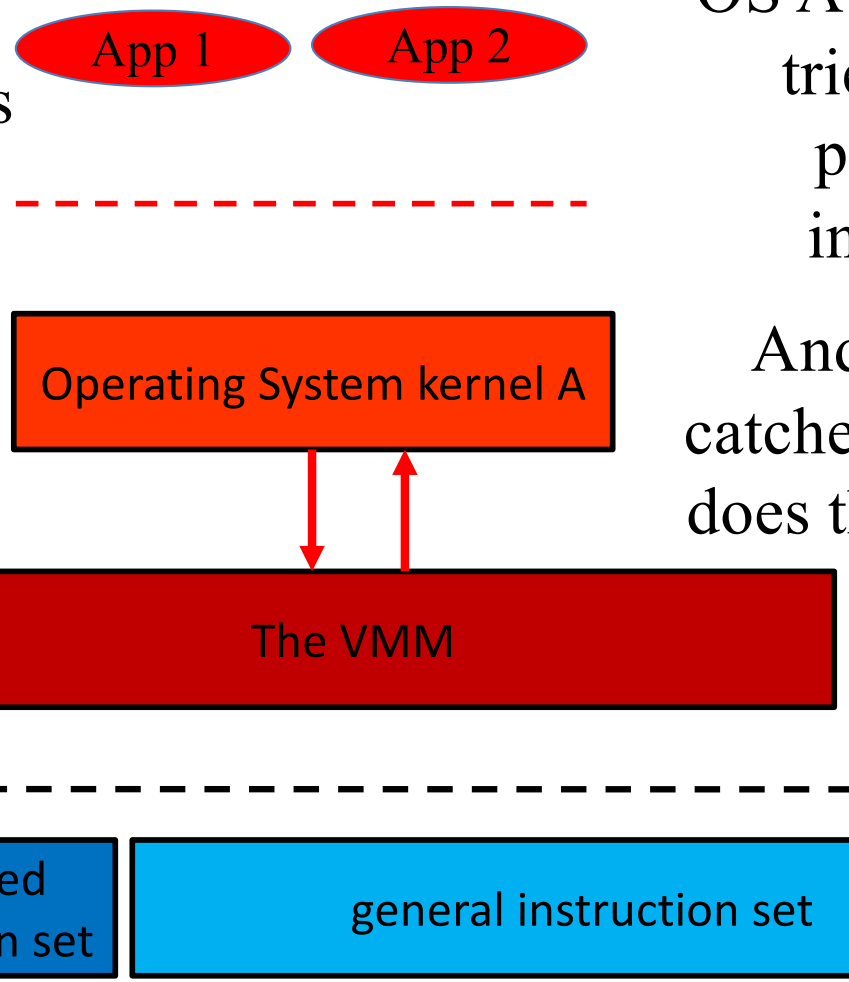
But the VMM knows where A's trap table is located

So it can invoke A's code to handle the syscall!

Yeah, But, Again . . .

If it's a syscall, it may need to use privileged instructions to do its work

But OS A can't use privileged instructions



No problem!

OS A traps when it tries to use a privileged instruction

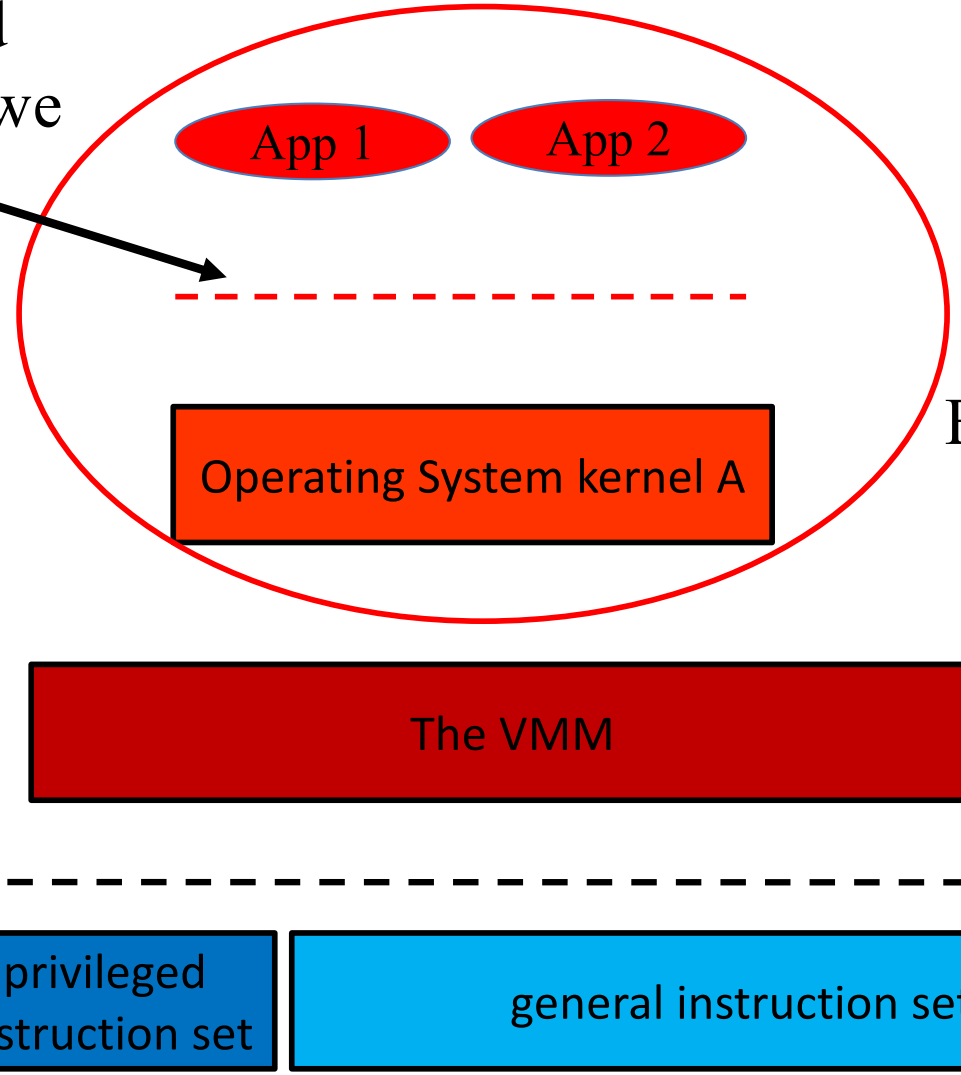
And the VMM catches the trap and does the instruction for A!

What's the Point of That?

- If the VMM is going to do the instruction, why not just run A with privilege?
  - So it can do its own instructions
- Well, the VMM might decide not to do the instruction
  - If, for example, it tries to access another VM's memory
- Or the VMM might block VM A and run VM B for a while instead
- The key point: the VMM controls what happens
  - Even though the OS in the VM thinks it is in control

A Potential Issue

If A is running in non-privileged mode, how can we enforce this interface?



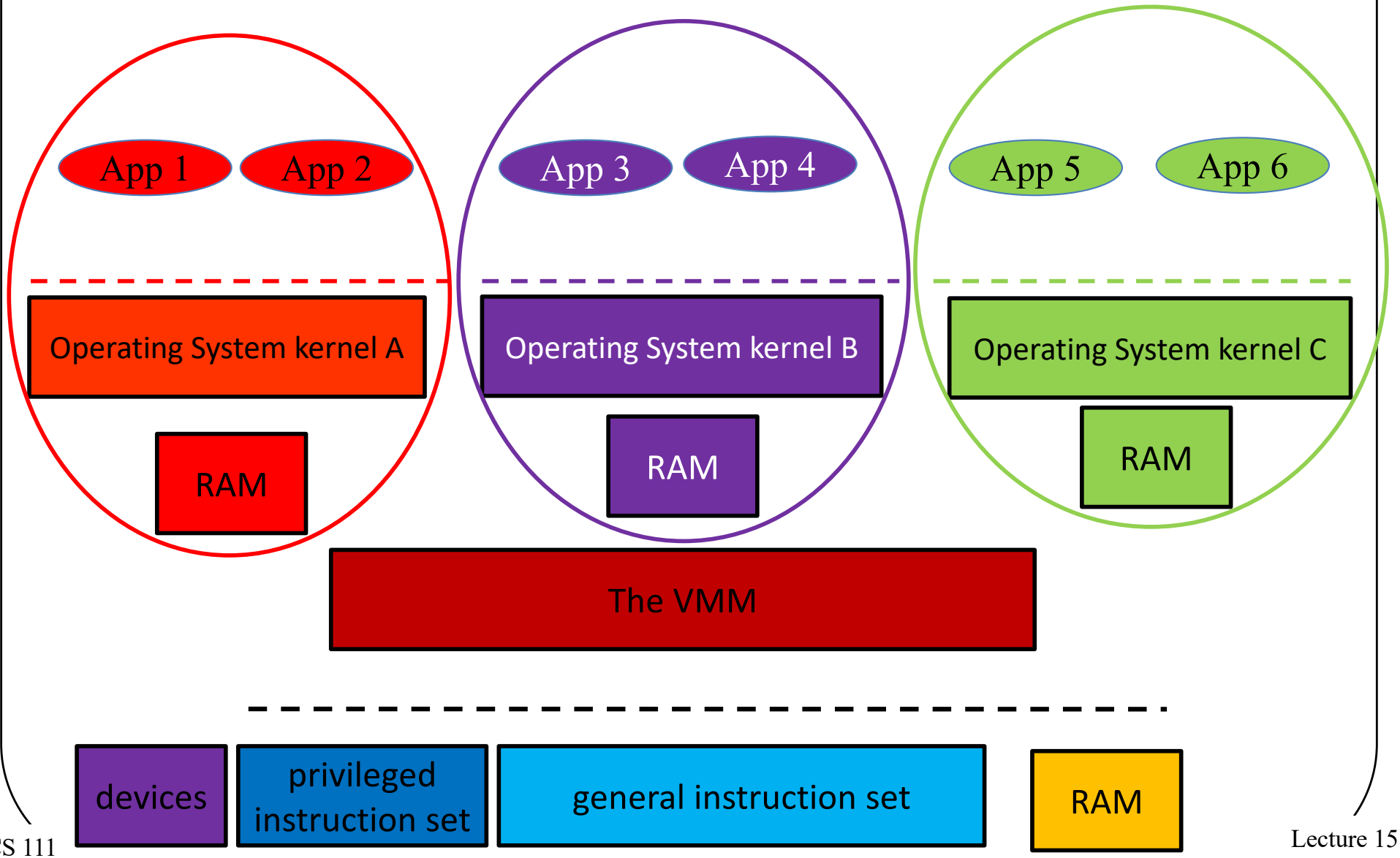
How can we prevent App 1 from messing with A's internal data?

E.g., stop App 1 from killing App 2?

The Core of the Problem

- OS A thinks it's in control
- OS A believes it's providing segregated virtual memories to App 1 and App 2
- A vital technology for doing so is managing page tables and CPU registers pointing to them
- But OS A has no control over those registers
  - The VMM does
- But the VMM knows nothing of the page tables OS A "controls"

Virtualizing the Memory



How To Virtualize Memory

- The virtual OS thinks it has physical memory addresses
  - It provides virtual memory addresses to its processes
  - Handling the virtual-to-physical translations
- The VMM has *machine addresses*
  - Which it translates to physical addresses within a single VM
  - Still using the same paging hardware

This May Be a Bit Confusing . . .

- Now there are three address types:
  - Virtual addresses — Not RAM addresses!
  - Physical addresses — Also not RAM addresses!
  - Machine addresses — These are RAM addresses!
- In this architecture, "physical addresses" aren't really physical
  - They aren't locations on real RAM chips
- Machine addresses are actual locations on real RAM chips

Three Types of Addresses

A virtual address

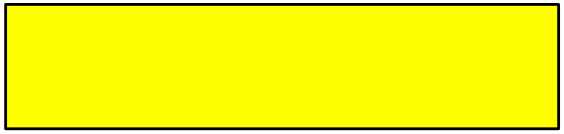
Which translates to . . .

Which isn't really physical

A physical address

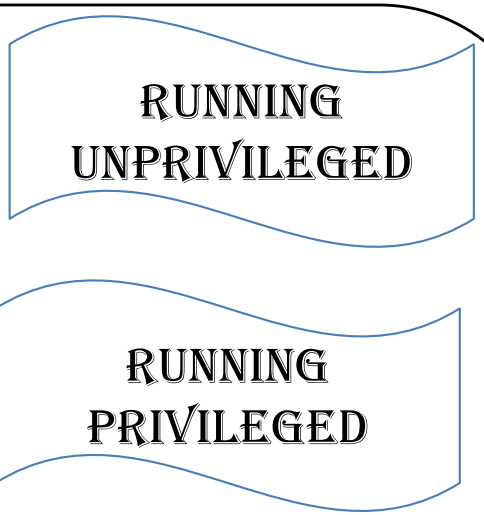
Which translates to . . .

Which is an actual RAM address



A machine address

For Example



App 1 issues virtual address X

Causing a TLB miss and a trap

The VMM invokes OS A

App 1

Operating System kernel A

Since only OS A understands App 1's page table

The VMM

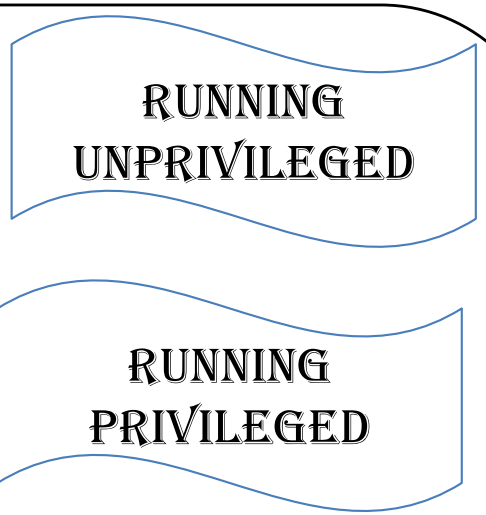
The VMM catches the trap

TLB

RAM

Continuing

And we eventually unwind to run App 1



OS A looks up virtual address X in App 1's page table

And tries to install the physical page number for X in the TLB

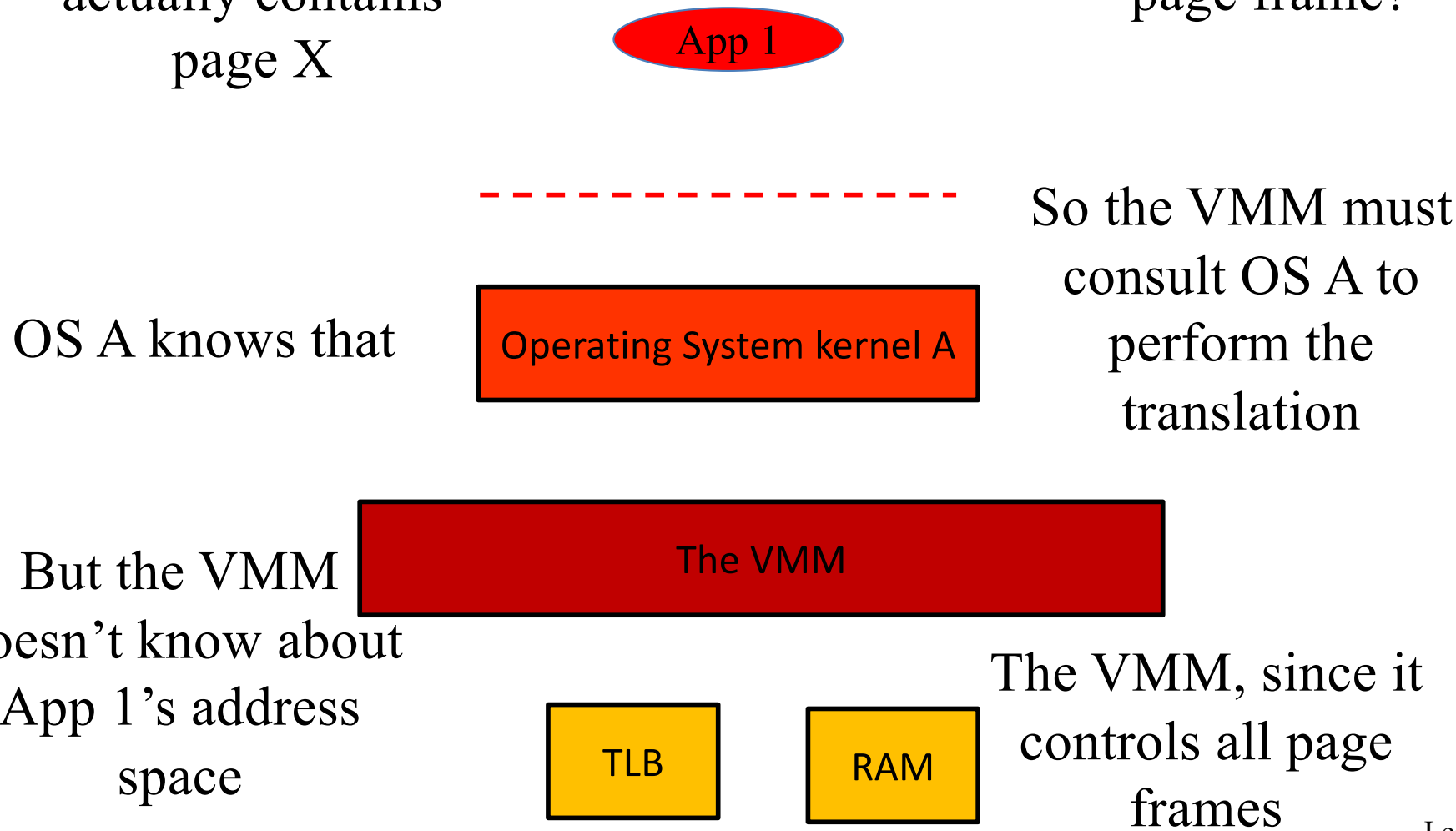
The VMM translates to the right machine address for X in the TLB and installs it

Which causes another trap to the VMM

Looked at Another Way

Some page frame actually contains page X

Who knows which page frame?



OS A knows that

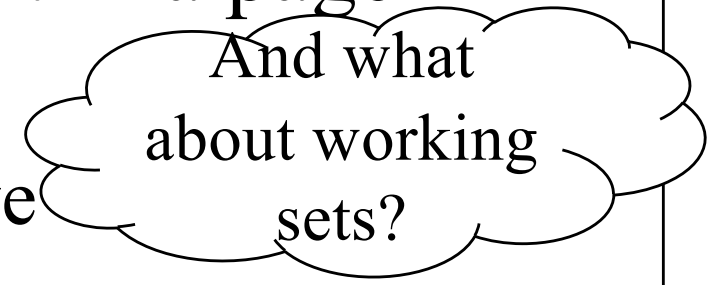
So the VMM must consult OS A to perform the translation

But the VMM doesn't know about App 1's address space

The VMM, since it controls all page frames

This Can Be (Much) More Complex

- What if the page requested isn't in a page frame?
  - It's somewhere on the flash drive
- Now we not only need to wind and unwind through the VMM
- When the guest OS tries to handle the page fault, that goes through the VMM, too
- And back to the guest OS when the page is delivered to RAM



Some Implications

- TLB misses are much more expensive
  - Since we'll be moving back and forth from privileged mode to unprivileged
  - Paying overhead costs each time
  - And we'll run more systems code
- We'll need extra paging data structures in the VMM
  - More overhead
- Virtual machines are thus likely to suffer performance penalties

Making VMs Perform Better

- Adding special hardware
  - Some CPUs have features to make issues of virtualizing the CPU and memory cheaper
- Paravirtualization
  - The basic VM approach assumes the guest OSes in VMs don't know about virtualization
  - If you make some changes to those OSes, they can help make virtualization cheaper

Virtual Machines and Cloud Computing

- Cloud computing is about sharing hardware among multiple customers
- The cloud provider sells/rents computing services to customers
  - Handling all the difficult issues for them
  - So they can just run their applications
- Cloud providers need lots of customers, to make money
  - Which implies they need lots of hardware

The Cloud Environment

A warehouse full of vast numbers of machines

Typically tens of thousands

Packed tightly into racks

Connected by high speed internal networks

And connected to the Internet, to allow customers remote access

The expectation is that the environment will run applications for many separate customers at the same time

Many of which might require multiple computers to run properly

With strong guarantees of isolation between customers

But Why VMs in the Cloud?

- The cloud provider makes the most money by making the most efficient use of the hardware
  - More customers on the same amount of hardware = more profits
- Often, a customer doesn't need the full power of a machine
  - Cloud provider makes more money by using part of that machine for another customer
- But they need strong isolation
- Like that provided by virtual machines . . .

So . . .

- You run everyone in a virtual machine
- Some customers have many virtual machines to handle their big jobs
  - Perhaps each the only VM on a physical machine
- Some customers' virtual machines share physical machines with other customers' VMs
- Customers' work loads fluctuate
- You want the most efficient packing of VMs onto physical machines possible
  - To maximize profits

How To Efficiently Place VMs

- There are many physical nodes and many more VMs
- Which do we put where?
- Often reduces to a bin packing algorithm
- Which tends to be NP-hard
  - Where *n* may depend on the number of servers and/or VMs considered
  - The more factors considered, the harder to solve
- So estimation techniques are used

But Things Change . . .

- Customers' jobs vary in their needs
- So a good bin packing for now might be a bad bin packing 10 minutes later
- The cloud controller can readjust as jobs start and end
- But what about long running jobs?
  - Like hosting large web servers?
- Either take away idle VMs
- Or migrate low use VMs to low use hardware

VMs Aren't Just For Cloud Computing

- As you should know, since your projects use them
- They allow experimentation not easily performed on real hardware
- They allow basic servers to safely divide their resources
- They allow greater flexibility in the software your computer can run

Conclusion

- Virtual machines are a critical technology for modern computing
- Virtual machines are implemented on real machines
- The key issue is providing each VM the illusion of complete control
- While also providing good performance
- VMs are of special importance in cloud computing