# Cloud Computing

## Virtualization: Part 1

Hong Xu
Department of Computer Science

Spring 2020

香港城市大學
City University of Hong Kong

# Outline

- Concepts
- Virtualization architecture
- CPU and OS basics
- Types of CPU virtualization

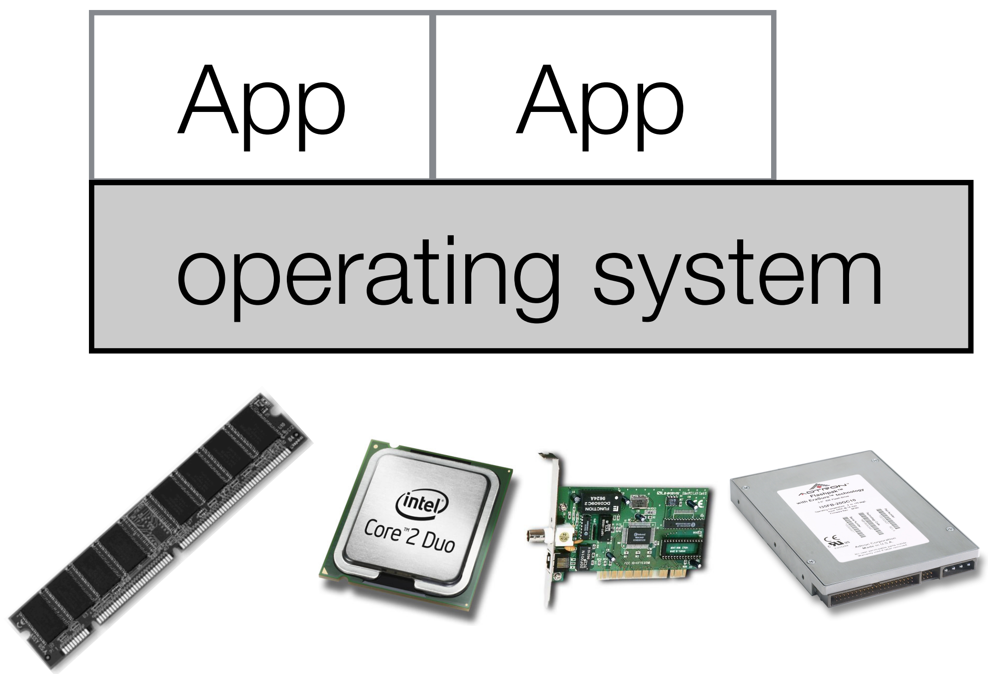
# What is virtualization?

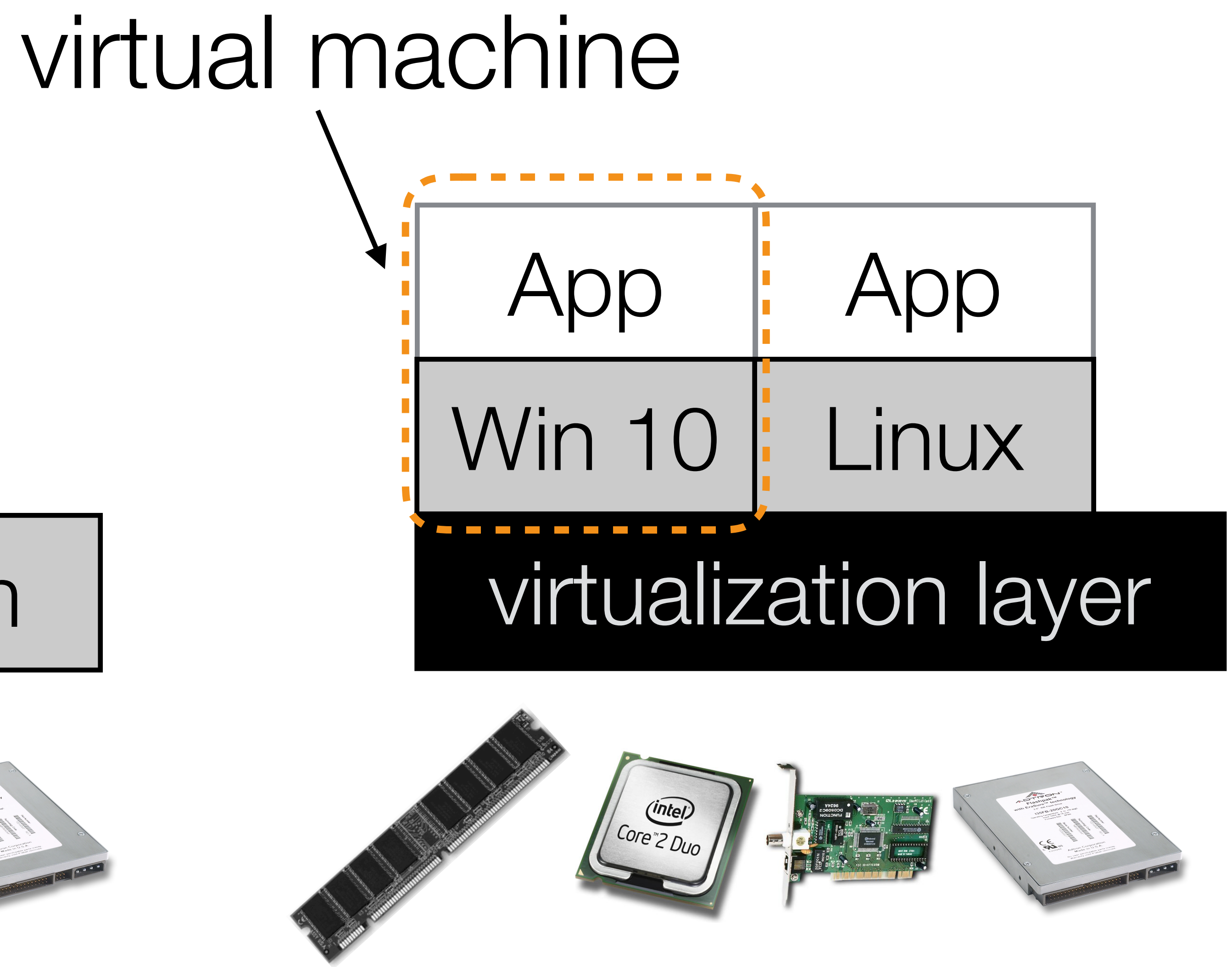
- Virtualization is a broad term. It can be applied to all types of resources (CPU, memory, network, etc.)

- Allows one computer to “look like” multiple computers, doing multiple jobs, by sharing the resources of a single machine across multiple environments.

  - virtualization started in 1960’s in IBM’s mainframe

# Virtualization



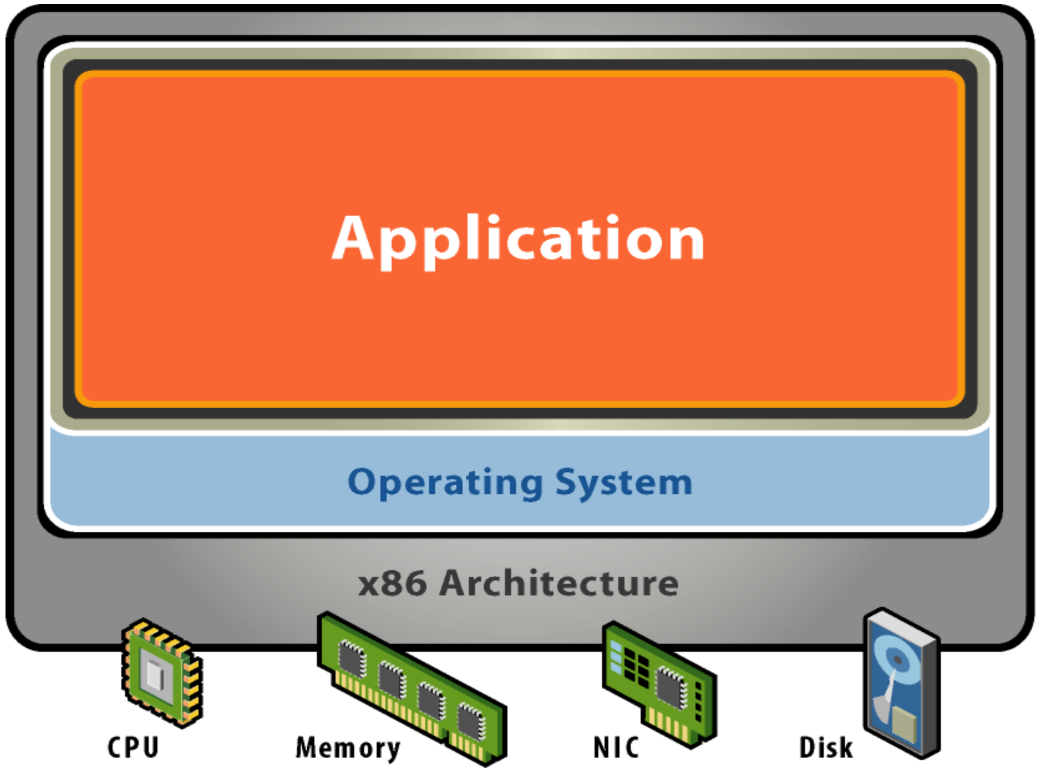




***'Nonvirtualized' system***
A single OS controls all hardware platform resources

***Virtualized system***
It makes it possible to run multiple Virtual Machines on a single physical platform

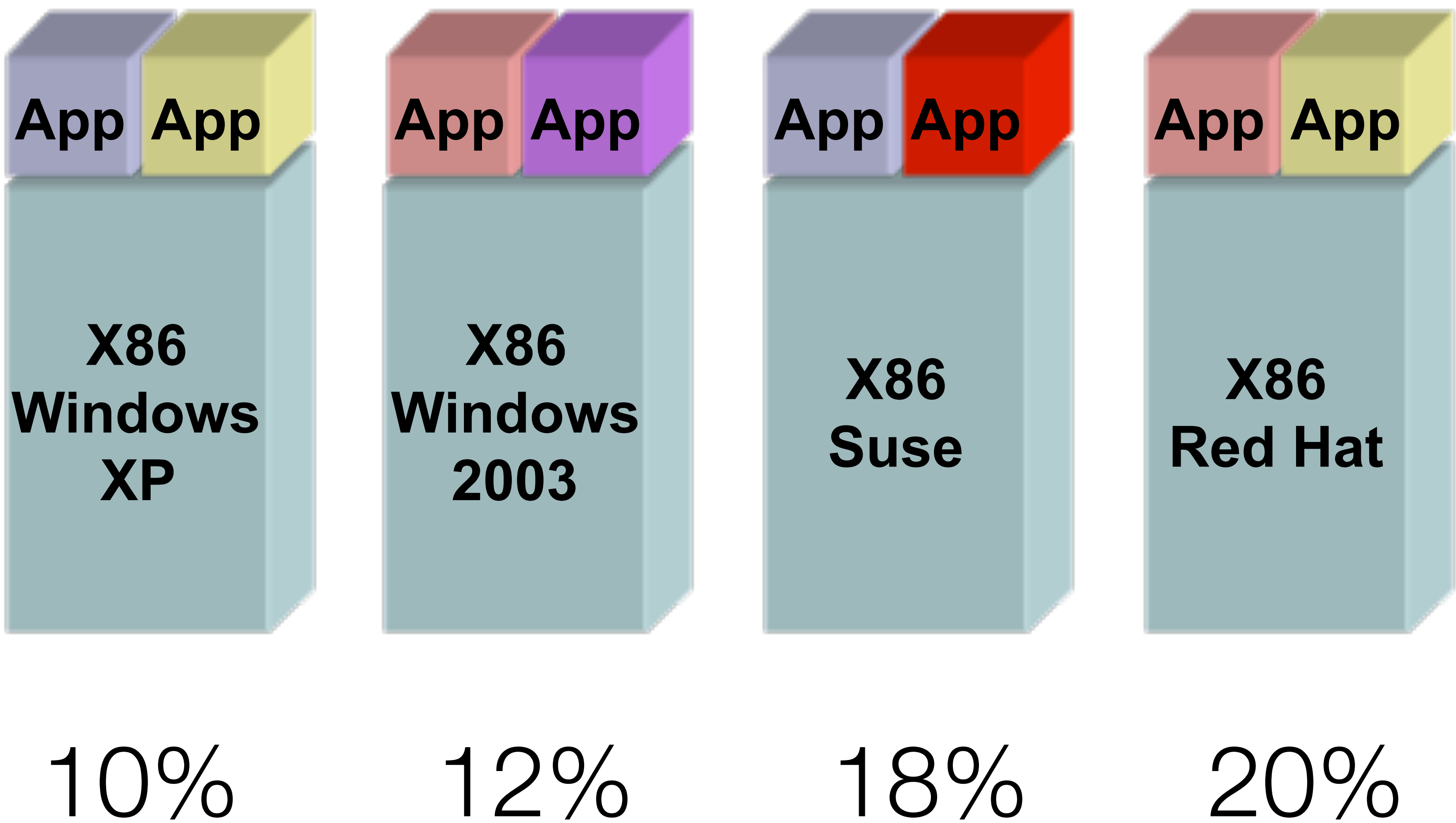
# The old model

- A server for every application
- Software and hardware are tightly coupled



# The old model

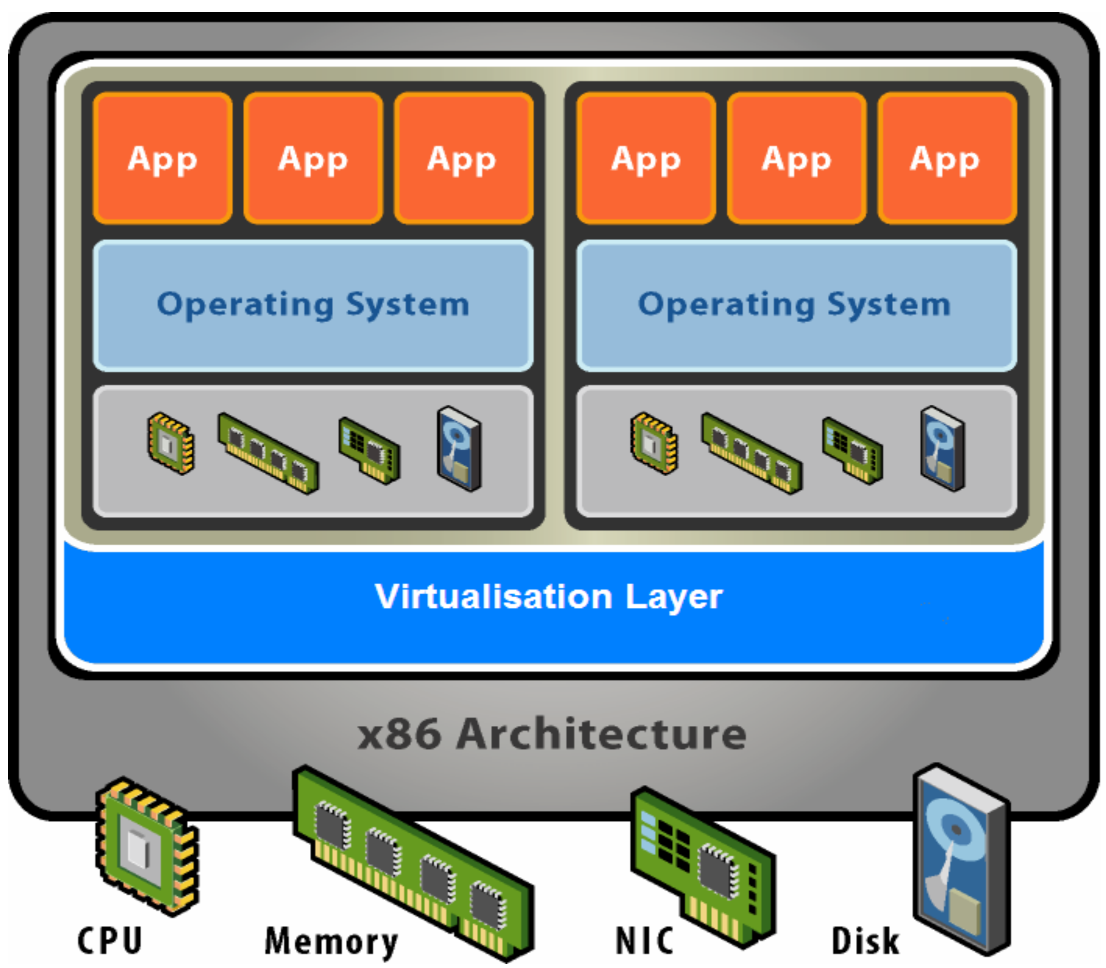
- Big disadvantage: low utilization



# The new model

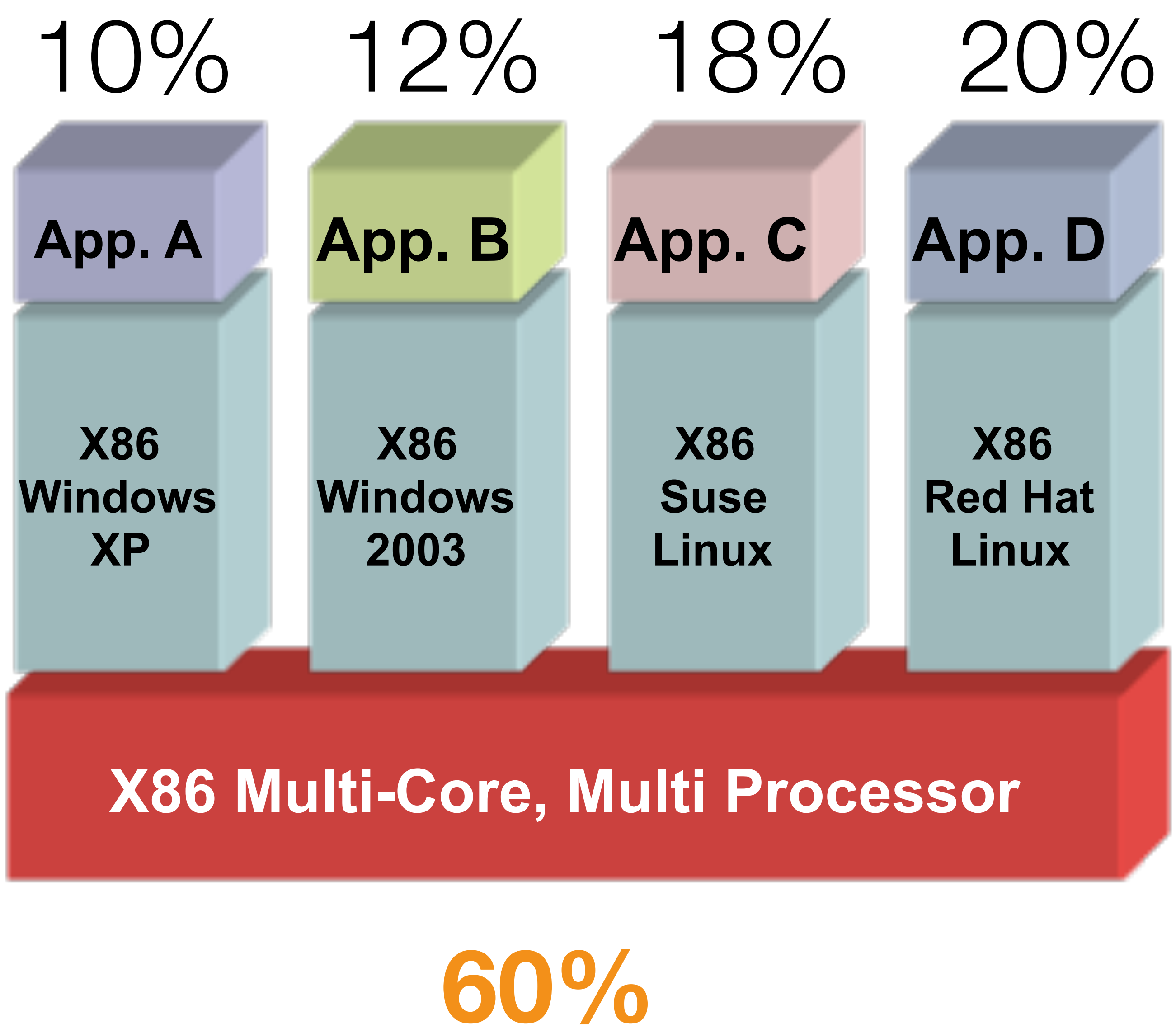
- Physical resources are virtualized. OS and applications as a single unit by encapsulating them into *virtual machines*

- Separate applications and hardware

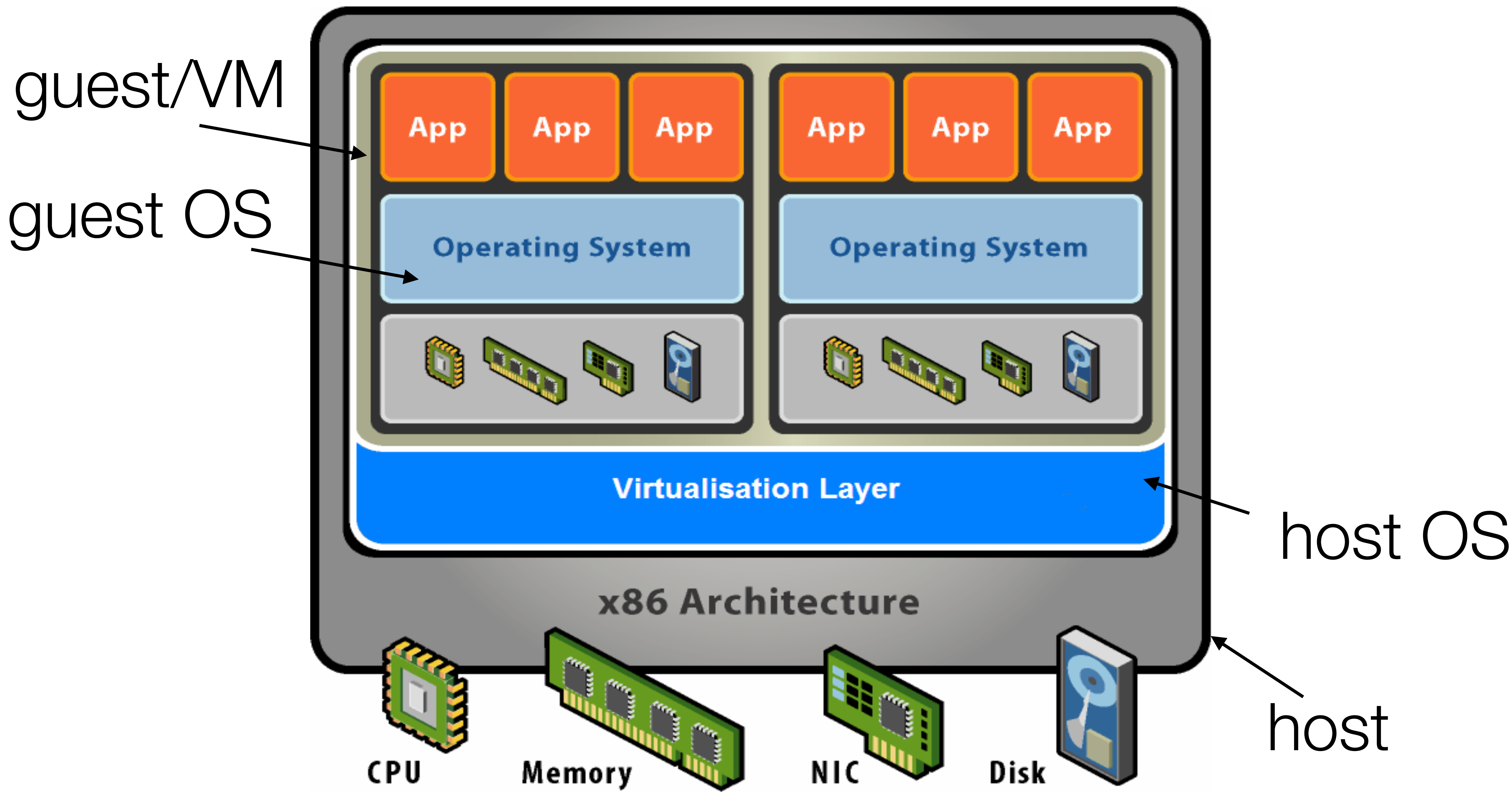


# The new model

- Big advantage: improved utilization



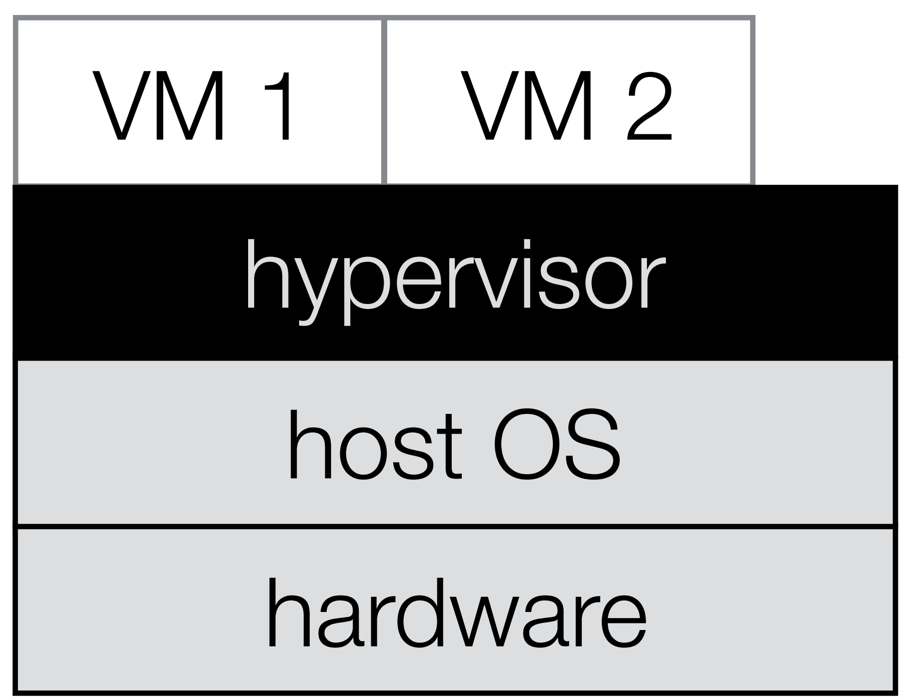
# Some terms


guest/VM
guest OS
App
App
App
App
App
App
Operating System
Operating System
Virtualisation Layer
host OS
x86 Architecture
host
CPU
Memory
NIC
Disk

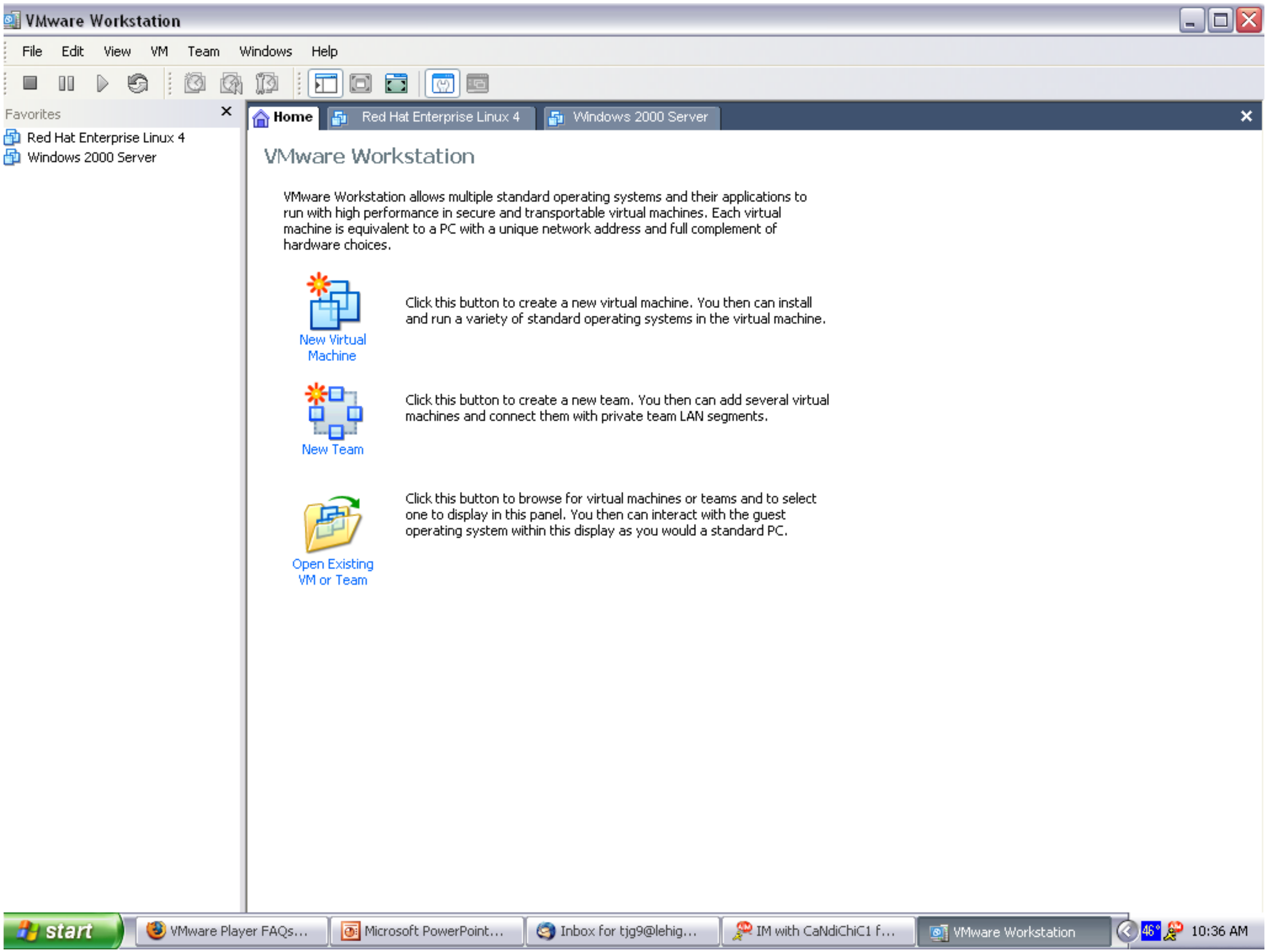
# Virtualization architecture

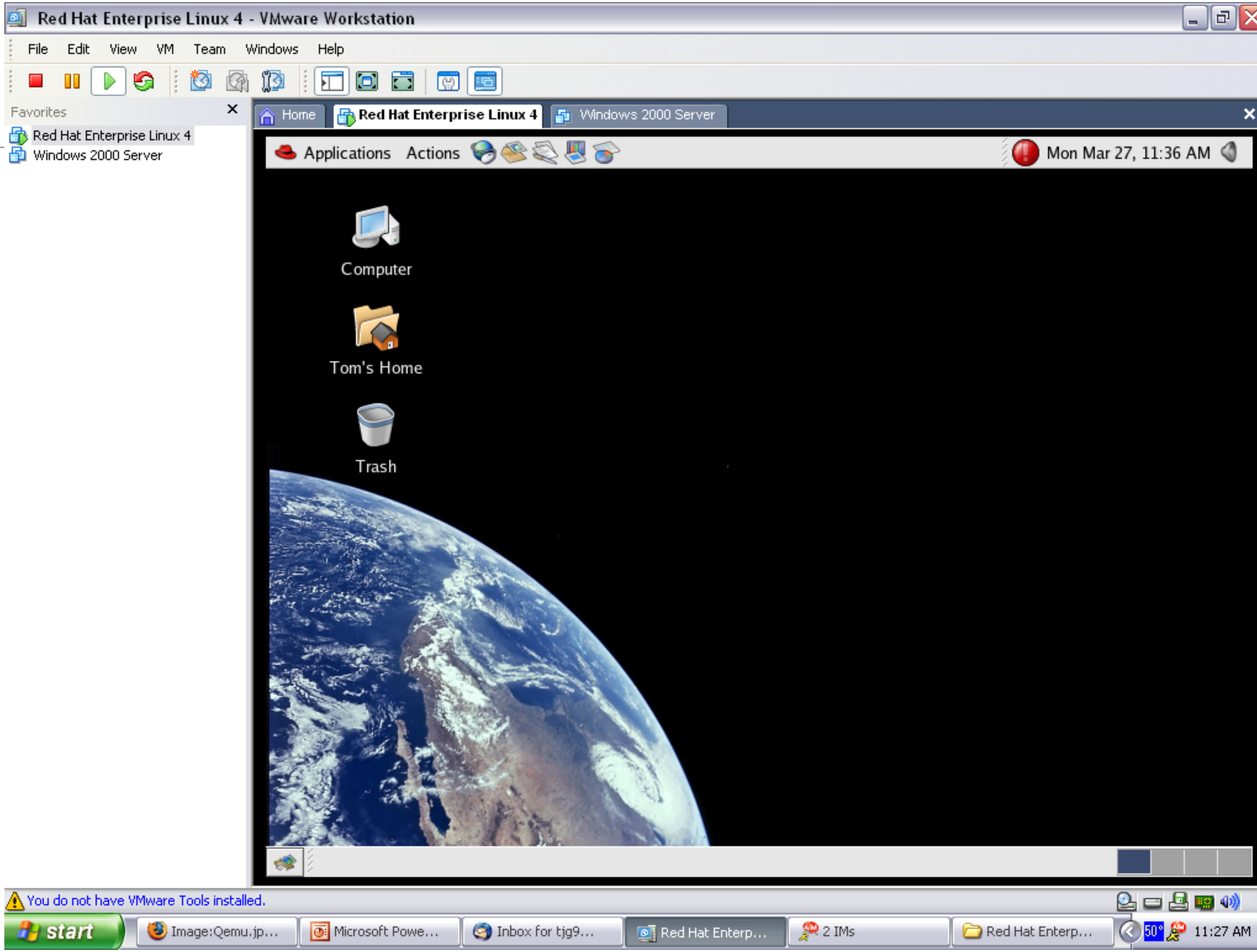
# Hosted architecture

- A hosted architecture installs and runs the virtualization layer as an application on top of an operating system



# Hosted architecture example



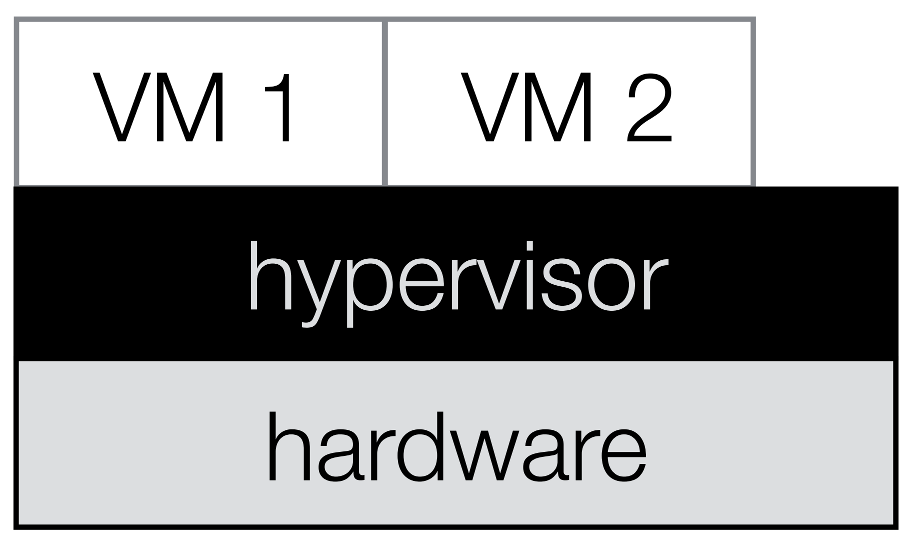

Red Hat Enterprise Linux 4 - VMware Workstation
File Edit View VM Team Windows Help
Favorites
Red Hat Enterprise Linux 4
Windows 2000 Server
Home
Red Hat Enterprise Linux 4
Windows 2000 Server
Applications Actions
Mon Mar 27, 11:36 AM
Computer
Tom's Home
Trash
You do not have VMware Tools installed.
start
Image:Qemu.jp...
Microsoft Powe...
Inbox for tjg9...
Red Hat Enterp...
2 IMs
Red Hat Enterp...
11:27 AM

# Hosted architecture

- Indirect access to hardware through the host OS
  - performance penalty, usually for desktops and personal use

# Hypervisor architecture

- The hypervisor architecture installs the virtualization layer, called **hypervisor**, directly on a clean x86-based system
  - Installer is usually an ISO installing a tailor-made OS



# Hypervisor architecture

- It has direct access to hardware resources
  - A hypervisor is more efficient than a hosted architecture and delivers greater scalability, robustness, and performance
  - For production use

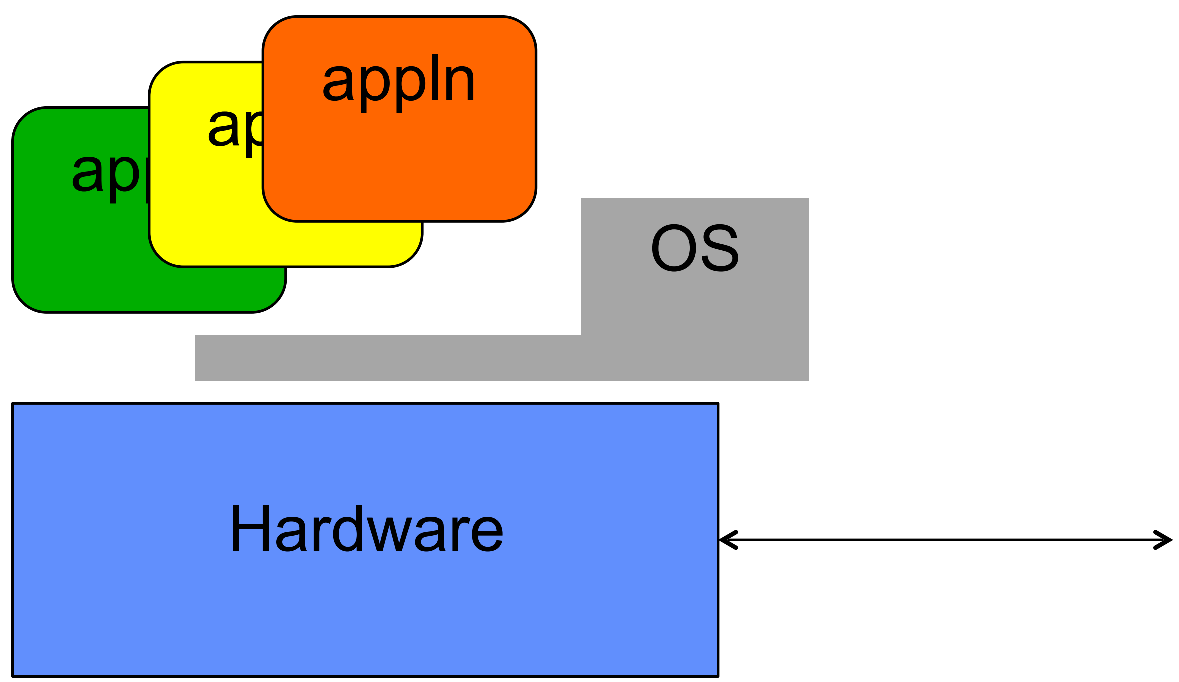
# CPU virtualization

# OS review

Credit: Prof. John Kubiatowicz’s slides for CS162, Spring 2015, UC Berkeley

# What is an operating system?

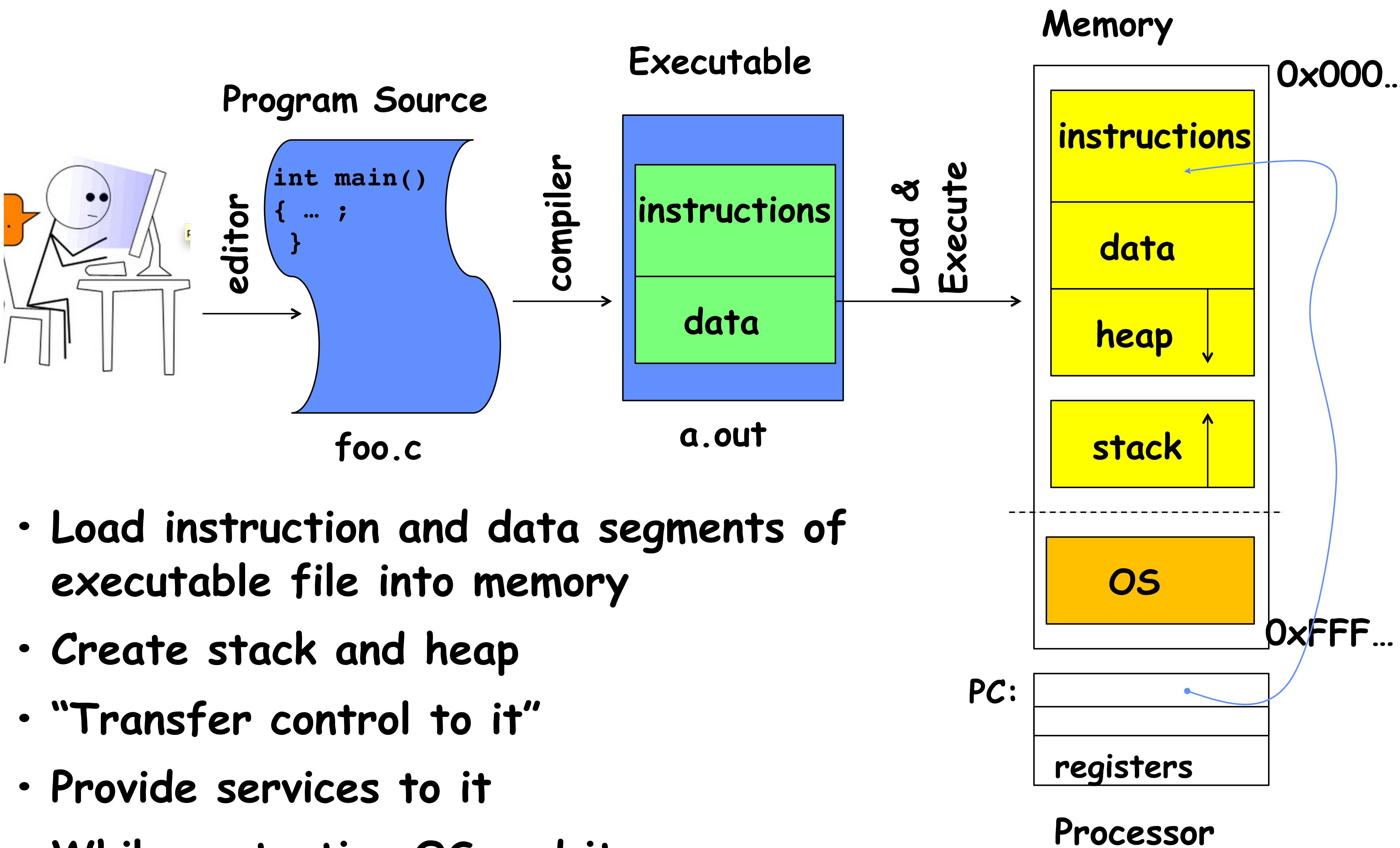
- Special layer of software that provides application access to hardware resources
  - Convenient abstraction of complex hardware devices
  - Protected access to shared resources
  - Security and authentication
  - Communication amongst logical entities



# Four fundamental OS concepts

- Thread
  - Single unique execution context
  - Program Counter, Registers, Execution Flags, Stack
- Address Space w/ Translation
  - Programs execute in an address space that is distinct from the memory space of the physical machine
- Process
  - An instance of an executing program is a process consisting of an address space and one or more threads of control
- Dual Mode operation/Protection
  - Only the “system” has the ability to access certain resources
  - The OS and the hardware are protected from user programs and user programs are isolated from one another by controlling the translation from program virtual addresses to machine physical addresses

# OS Bottom Line: Run Programs



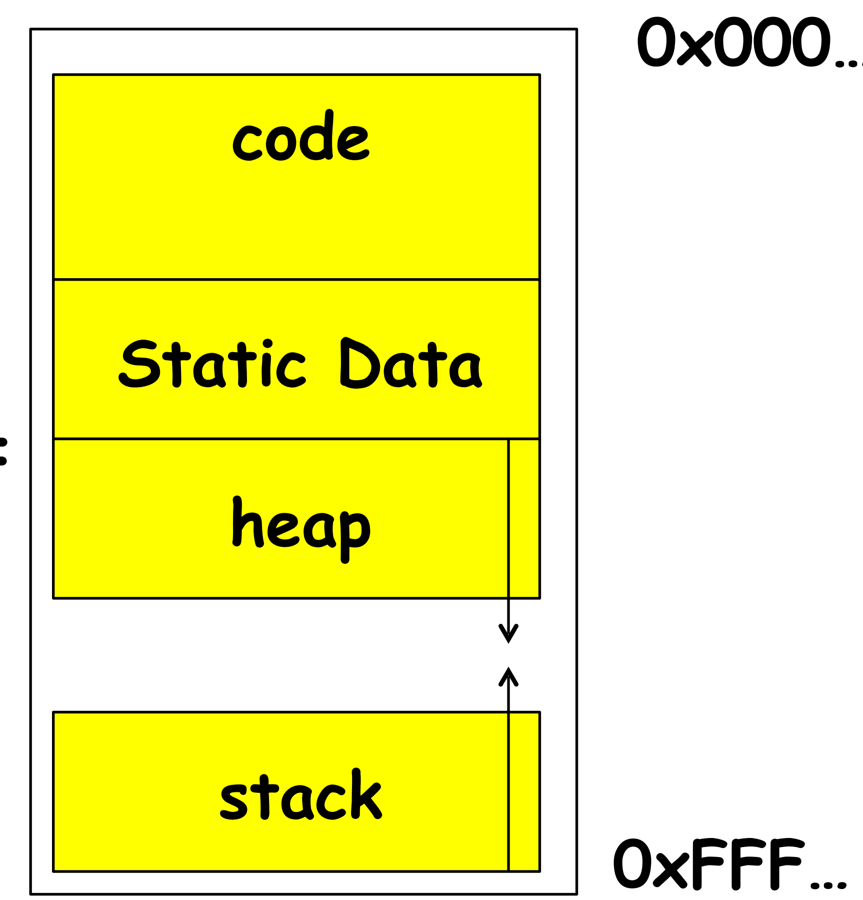

- Load instruction and data segments of executable file into memory
- Create stack and heap
- "Transfer control to it"
- Provide services to it
- While protecting OS and it

# First OS Concept: Thread of Control

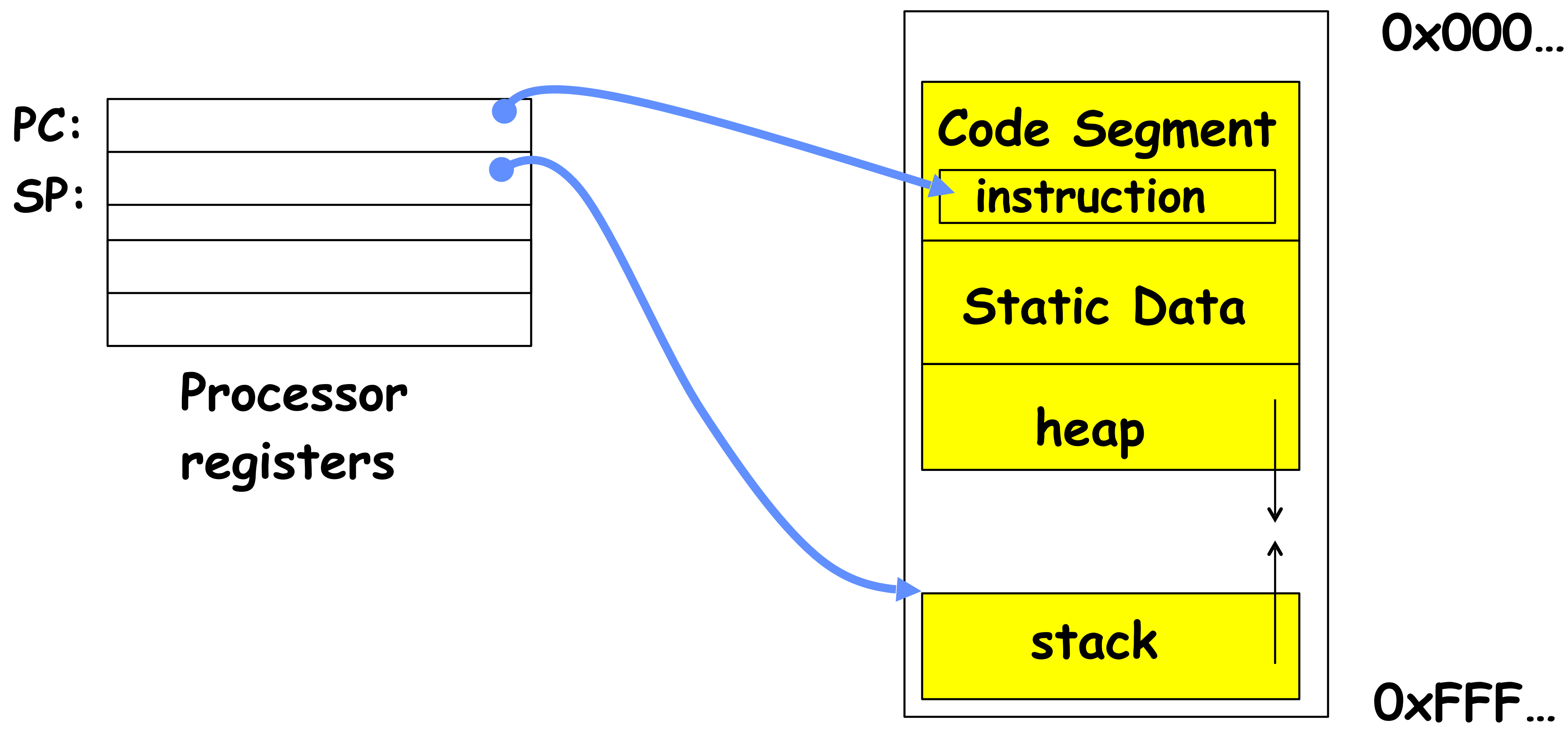
- Thread: Single unique execution context
  - Program Counter, Registers, Execution Flags, Stack
- A thread is executing on a processor when it is resident in the processor registers.
- PC register holds the address of executing instruction in the thread.
- Certain registers hold the context of thread
  - Stack pointer holds the address of the top of stack
    - » Other conventions: Frame Pointer, Heap Pointer, Data
  - May be defined by the instruction set architecture or by compiler conventions
- Registers hold the root state of the thread.
  - The rest is "in memory"

# Second OS Concept: Program's Address Space

- **Address space ⇒ the set of accessible addresses + state associated with them:**
  - For a 32-bit processor there are $2^{32}$ = 4 billion addresses
- **What happens when you read or write to an address?**
  - Perhaps Nothing
  - Perhaps acts like regular memory
  - Perhaps ignores writes
  - Perhaps causes I/O operation
    - » (Memory-mapped I/O)
  - Perhaps causes exception (fault)

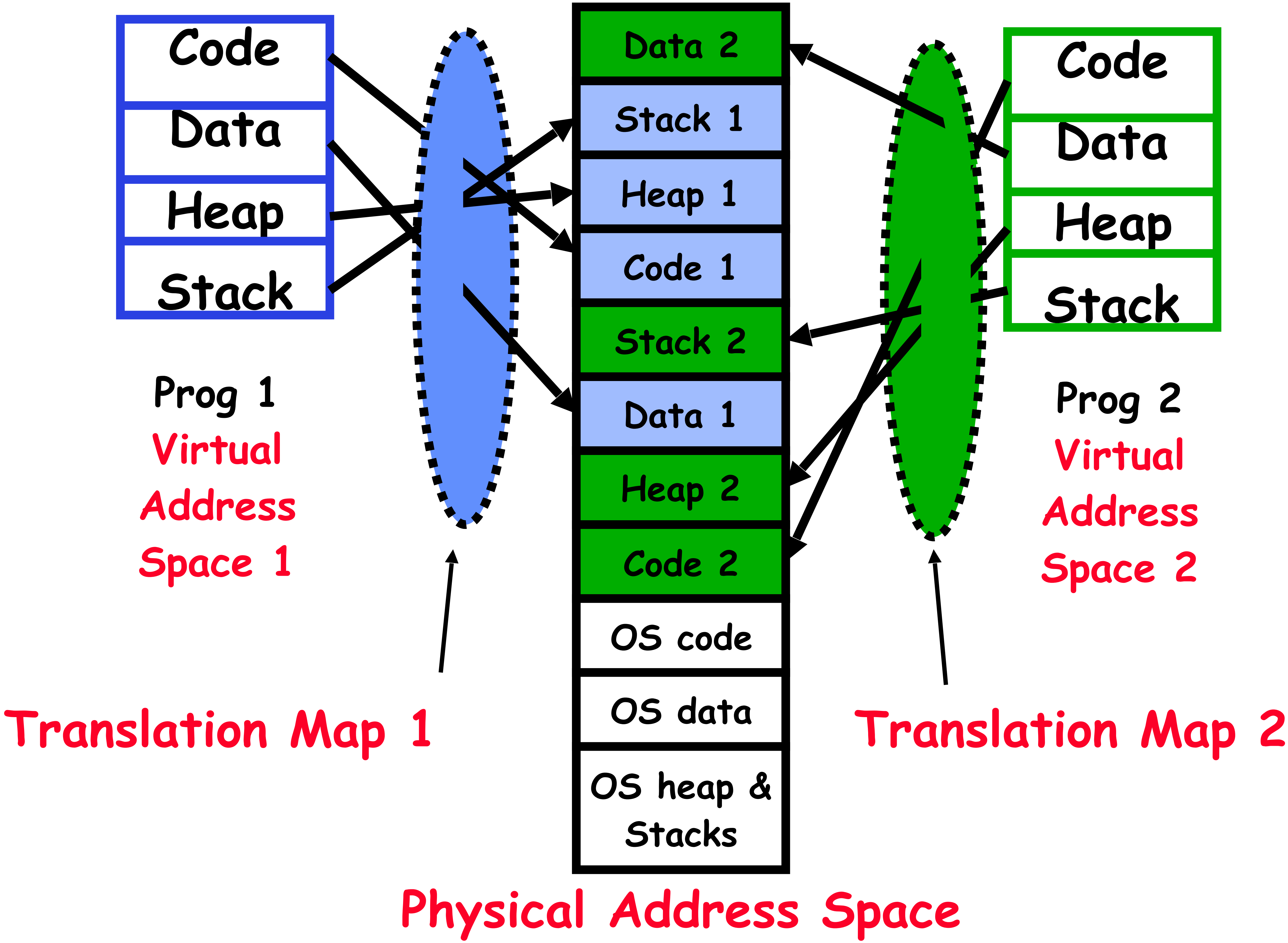


# Address Space: In a Picture




- What's in the code segment? Data? (global var)
- What's in the stack segment?
  - automatic variables, register values, etc.
- What's in the heap segment?
  - variables from dynamic memory allocation (malloc, etc.)

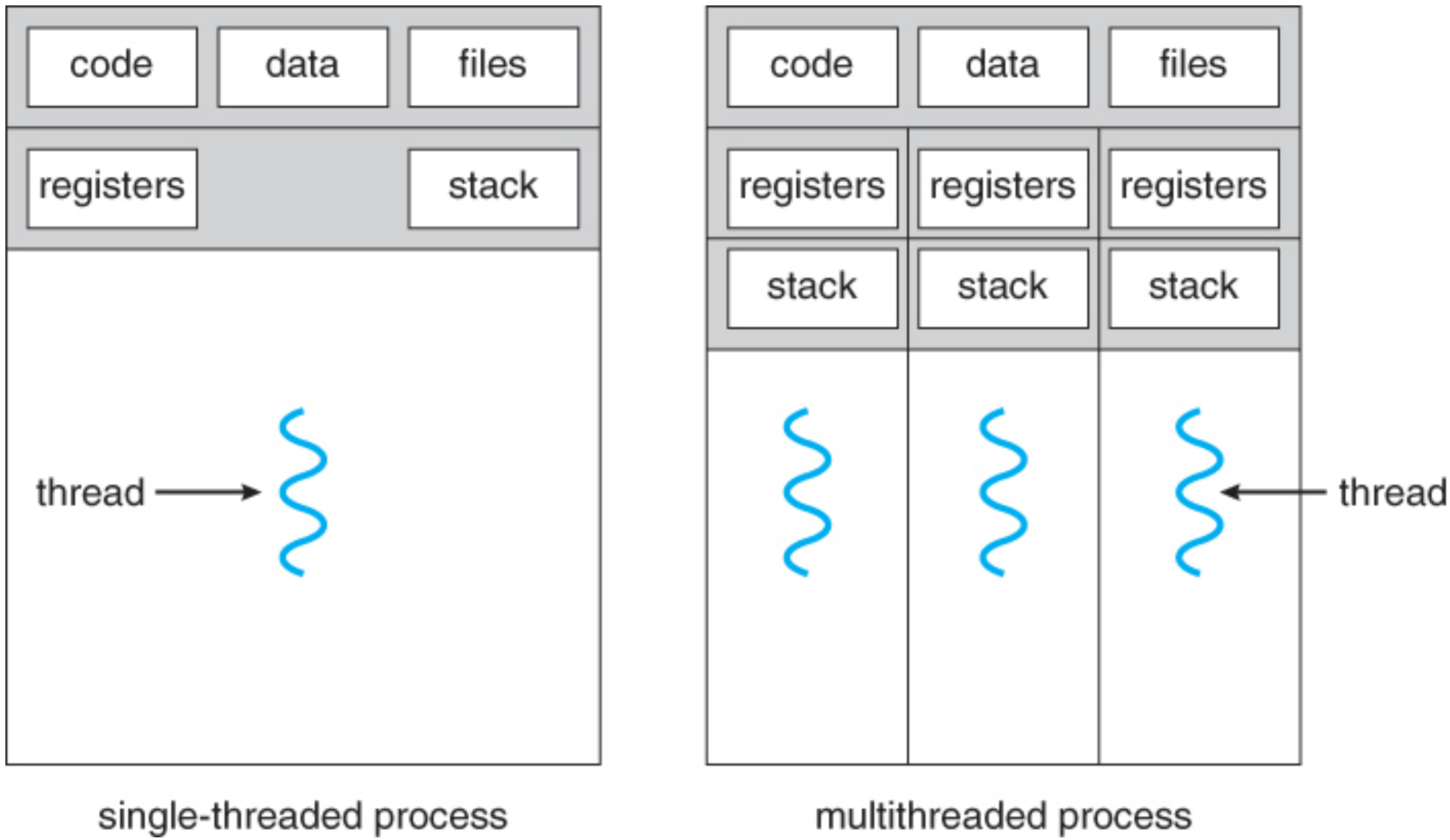
## Providing Illusion of Separate Address Space: Load new Translation Map on Switch



# Third OS Concept: Process

- Process: execution environment with Restricted Rights
  - Address Space with One or More Threads
  - Owns memory (address space)
  - Owns file descriptors, file system context, …
  - Encapsulate one or more threads sharing process resources
- Why processes?
  - Protected from each other!
  - OS Protected from them
  - Navigate fundamental tradeoff between protection and efficiency
  - Processes provides memory protection
- Application instance consists of one or more processes

# Process vs thread



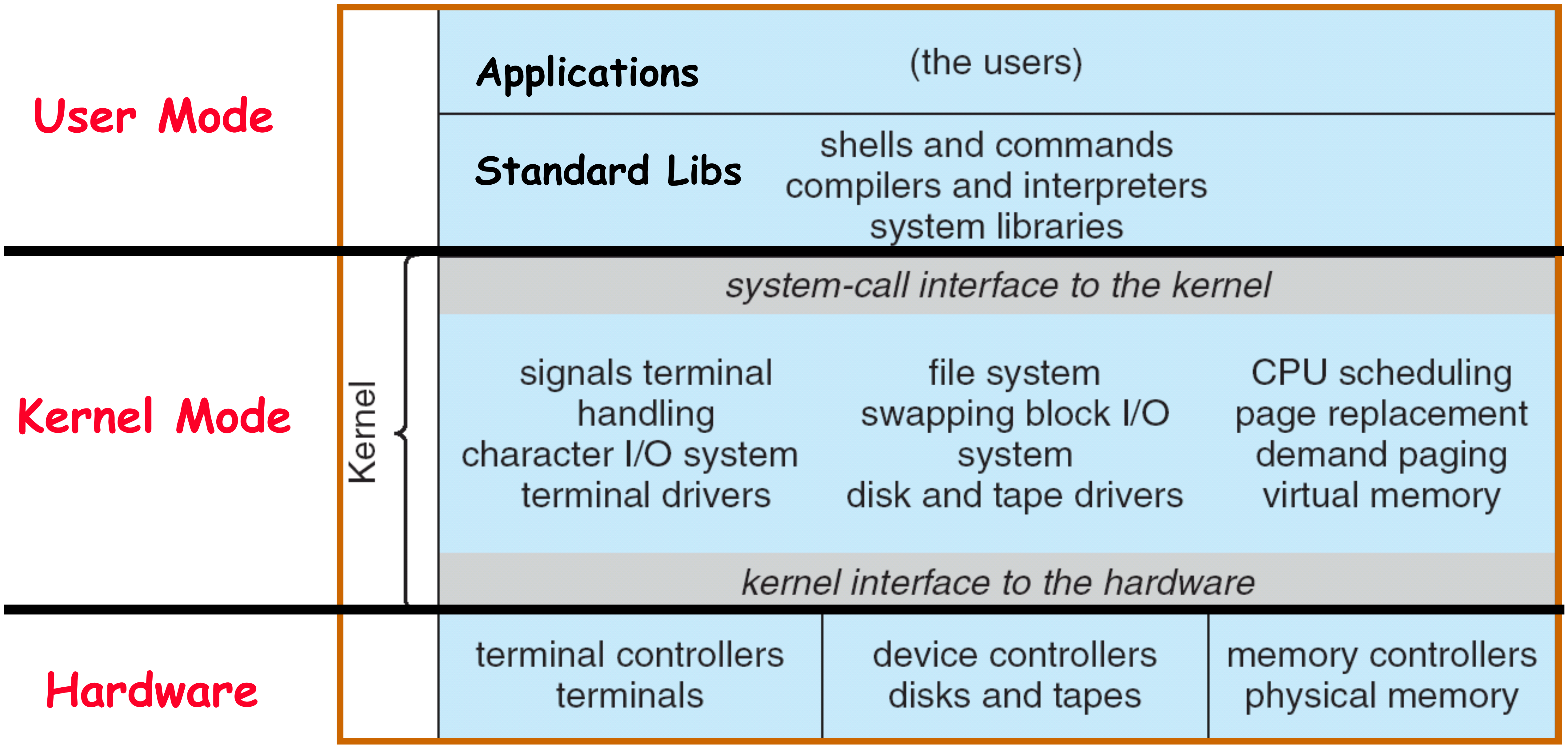
# Protection

- Operating System must protect itself from user programs
  - Reliability: compromising the operating system generally causes it to crash
  - Security: limit the scope of what processes can do
  - Privacy: limit each process to the data it is permitted to access
  - Fairness: each should be limited to its appropriate share
- It must protect User programs from one another
- Primary Mechanism: limit the translation from program address space to physical memory space
  - Can only touch what is mapped in
- Additional Mechanisms:
  - Privileged instructions, in/out instructions, special registers
  - syscall processing, subsystem implementation
    - » (e.g., file access rights, etc)

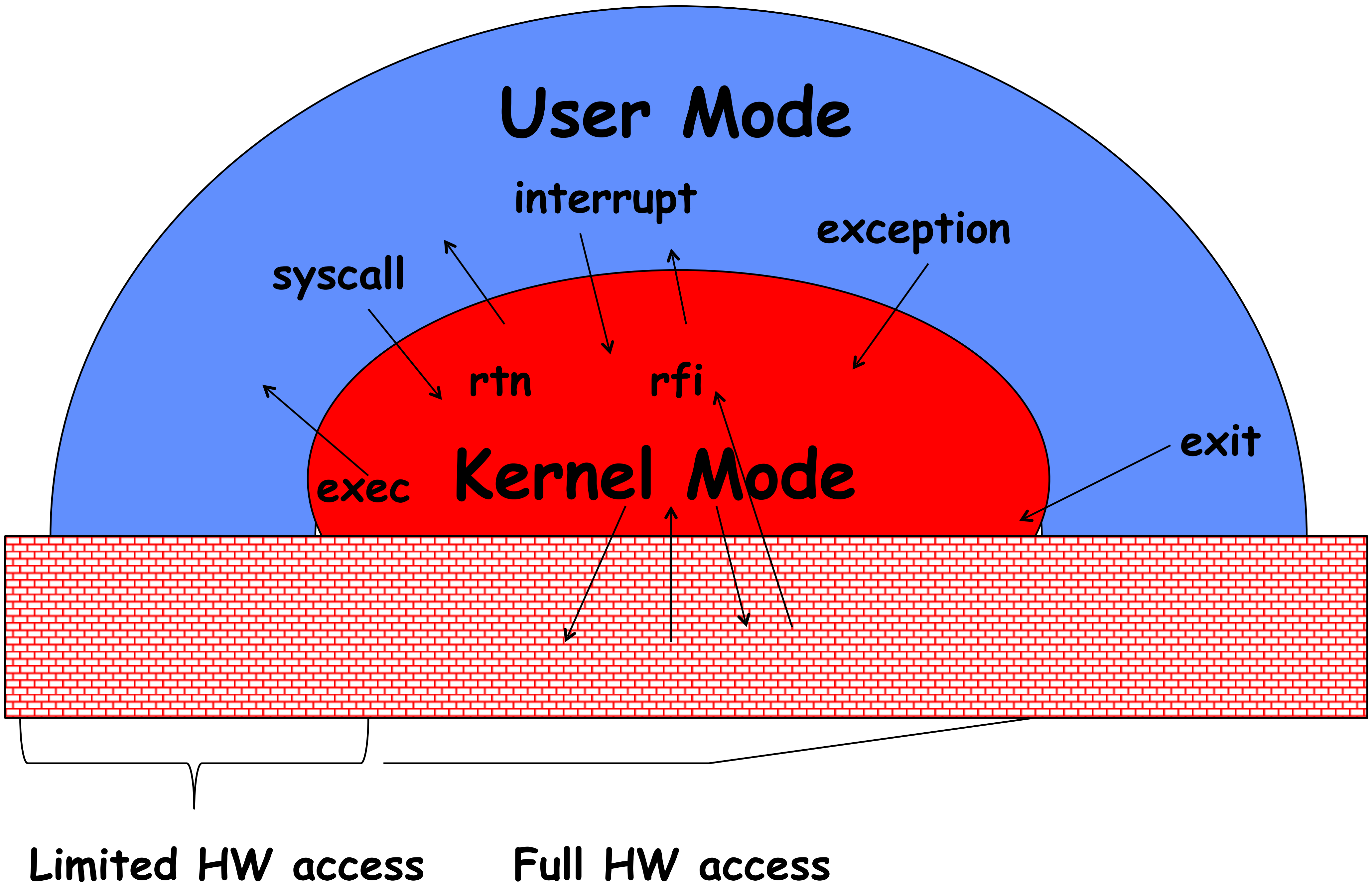
# Fourth OS Concept: Dual Mode Operation

- Hardware provides at least two modes:
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode: Normal programs executed
- What is needed in the hardware to support "dual mode" operation?
  - a bit of state (user/system mode bit)
  - Certain operations / actions only permitted in system/kernel mode
    - » In user mode they fail or trap
  - User->Kernel transition sets system mode AND saves the user PC
    - » Operating system code carefully puts aside user state then performs the necessary operations
  - Kernel->User transition clears system mode AND restores appropriate user PC
    - » return-from-interrupt

# For example: UNIX System Structure

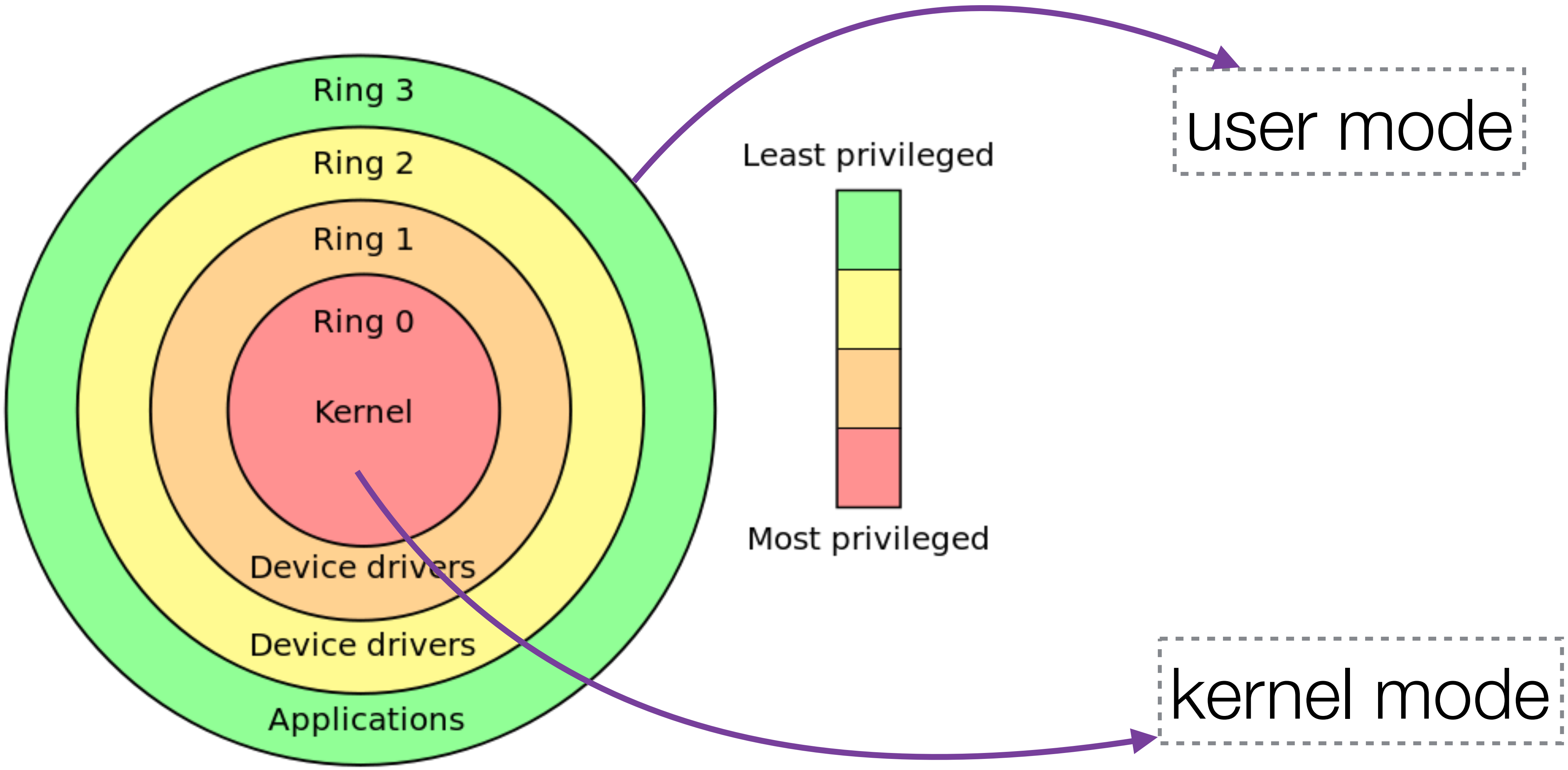


# User/Kernel (Privileged) Mode



# Protection rings

- Enforced in hardware in x86 architectures




Source: Wikipedia

# How to virtualize a CPU?

- Basically, the CPU does not care whether you are the guest OS or not
  - Have the PC point to somewhere in the RAM
- If it’s unprivileged code, code that execute in userspace
  - It’s safe to run no matter it is from the guest OS or host OS
  - Why?

# What about privileged code?

- Currently, there’re 3 implementations
  - Full virtualization
  - Para virtualization
  - Hardware-assisted virtualization

# Types of CPU virtualization

# Full virtualization

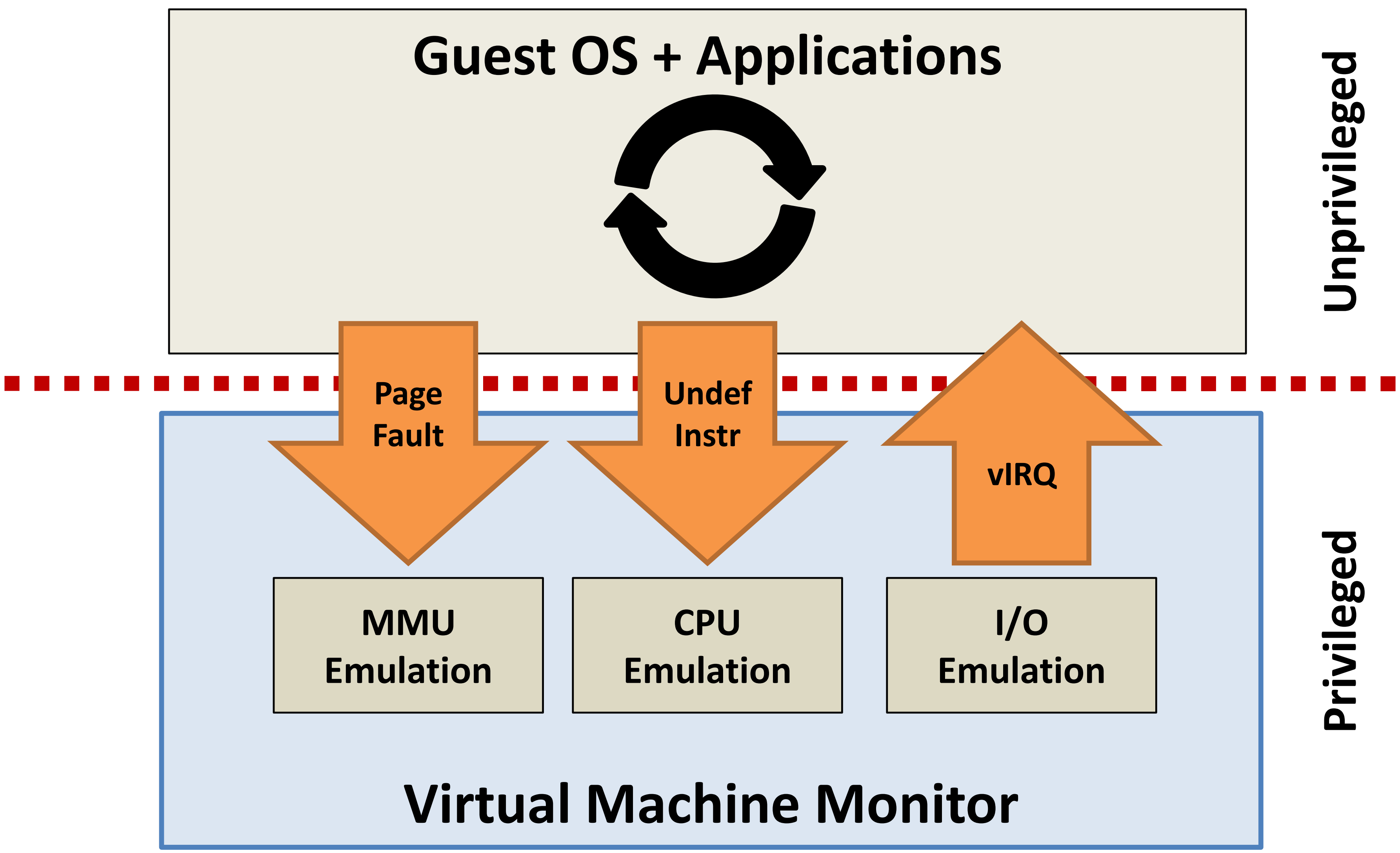
| | |
|---|---|
| Ring 3 | Guest applications |
| Ring 2 | |
| Ring 1 | Guest OS kernel |
| Ring 0 | Hypervisor, Host OS |
| Host hardware | |

hardware emulation

binary translation

# Full virtualization

- Hardware is emulated by the hypervisor



# Full virtualization

- The hypervisor presents a complete set of emulated hardware to the VM's guest operating system, including the CPU, motherboard, memory, disk, disk controller, and network cards.
- For example, Microsoft Virtual Server 2005 emulates an Intel 21140 NIC card and Intel 440BX chipset.
- Regardless of the actual physical hardware on the host system, the emulated hardware remains the same.

# Full virtualization

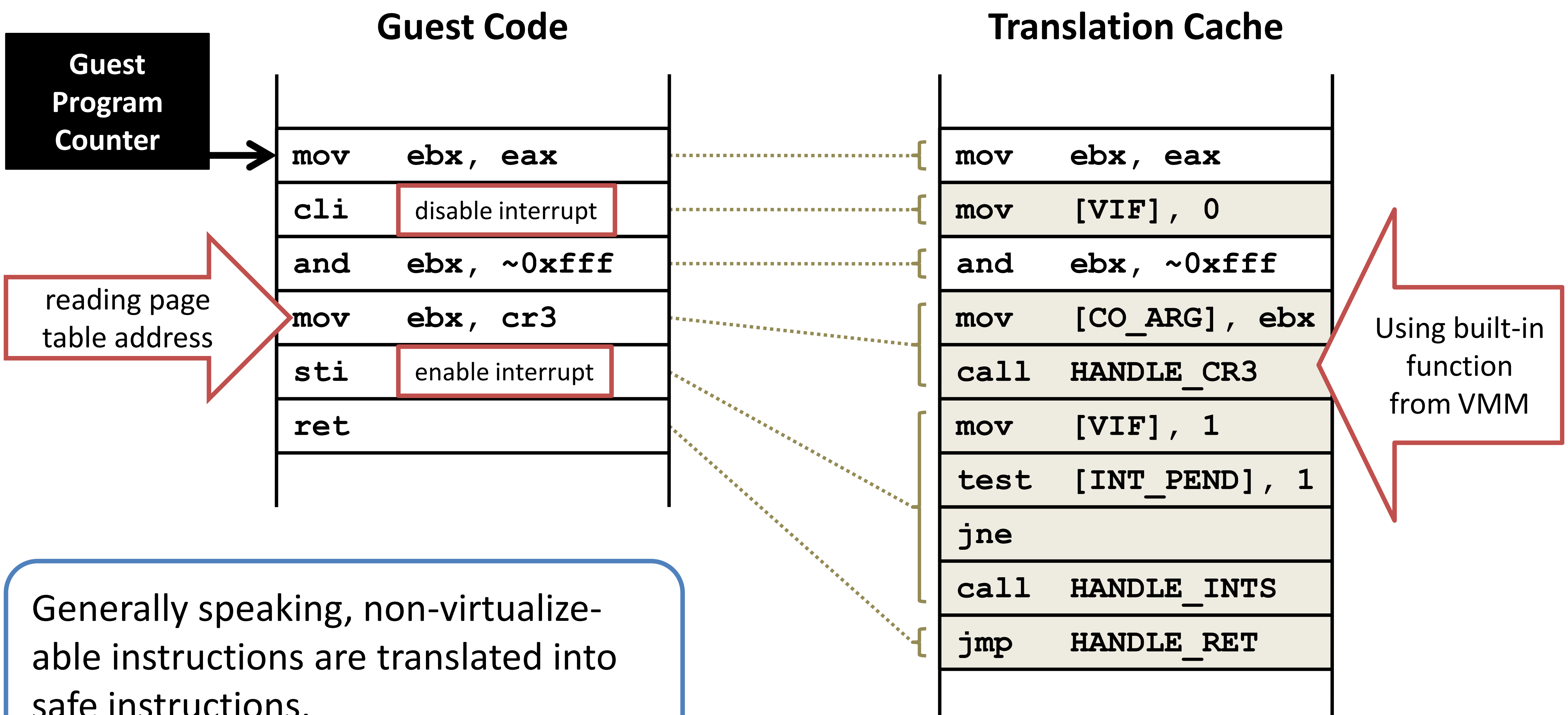
- Binary translation – step 1: trapping I/O calls

| | |
|---|---|
| Ring 3 | Guest applications |
| Ring 2 | |
| Ring 1 | Guest OS kernel |
| Ring 0 | Hypervisor, Host OS |
| Host hardware | |

whenever the guest OS asks for hardware, e.g. asking BIOS for a list of hardware, it’s trapped by the hypervisor

# Full virtualization

- Binary translation – step 2: emulate/translate




Generally speaking, non-virtualize-able instructions are translated into safe instructions.

# Full virtualization

- The guest OS is tricked to think that it’s running privileged code in Ring 0, while it’s actually running in Ring 1 of the host with the hypervisor emulating the hardware and trapping privileged code
- Unprivileged instructions are directly executed on CPU

# Full virtualization

- Advantages:
  - Keeps the guest OS unmodified
  - Prevents an unstable VMs from impacting system performance; VM portability
- Disadvantages:
  - Performance is not good

# Para-virtualization

- Developed to overcome the performance penalty of full virtualization with hardware emulation
- “Para” means “besides,” “with,”, or “alongside.”

# Para-virtualization

| | |
|---|---|
| Ring 3 | Guest applications |
| Ring 2 | |
| Ring 1 | Modified Guest OS kernel |
| Ring 0 | Hypervisor, Host OS |
| Host hardware | |

include virtualization APIs and drivers

no binary translation

# Para-virtualization

- Can be done in two ways:
  - A recompiled OS kernel. Easy for Linux, Windows doesn’t support
  - Paravirtualization drivers for some hardware, e.g. GPU, NIC

# Para-virtualization

- Guest OS is aware that it runs in a virtualized environment. It talks to the hypervisor through specialized APIs to run privileged instructions.

- These system calls, in the guest OS, are also called “hypercalls.”

- Performance is improved. The hypervisor can focus on isolating VMs and coordinating.

# Hardware-assisted

<table>
<tr><td rowspan="4">Non-root mode</td><td>Ring 3</td><td>Guest applications</td></tr>
<tr><td>Ring 2</td><td></td></tr>
<tr><td>Ring 1</td><td></td></tr>
<tr><td>Ring 0</td><td>Guest OS kernel</td></tr>
<tr><td>Root mode</td><td>Ring -1</td><td>Hypervisor</td></tr>
<tr><td colspan="3">Host hardware</td></tr>
</table>

likely to emerge as the standard for server virtualization into the future

e.g., Intel® VT-x, AMD® V

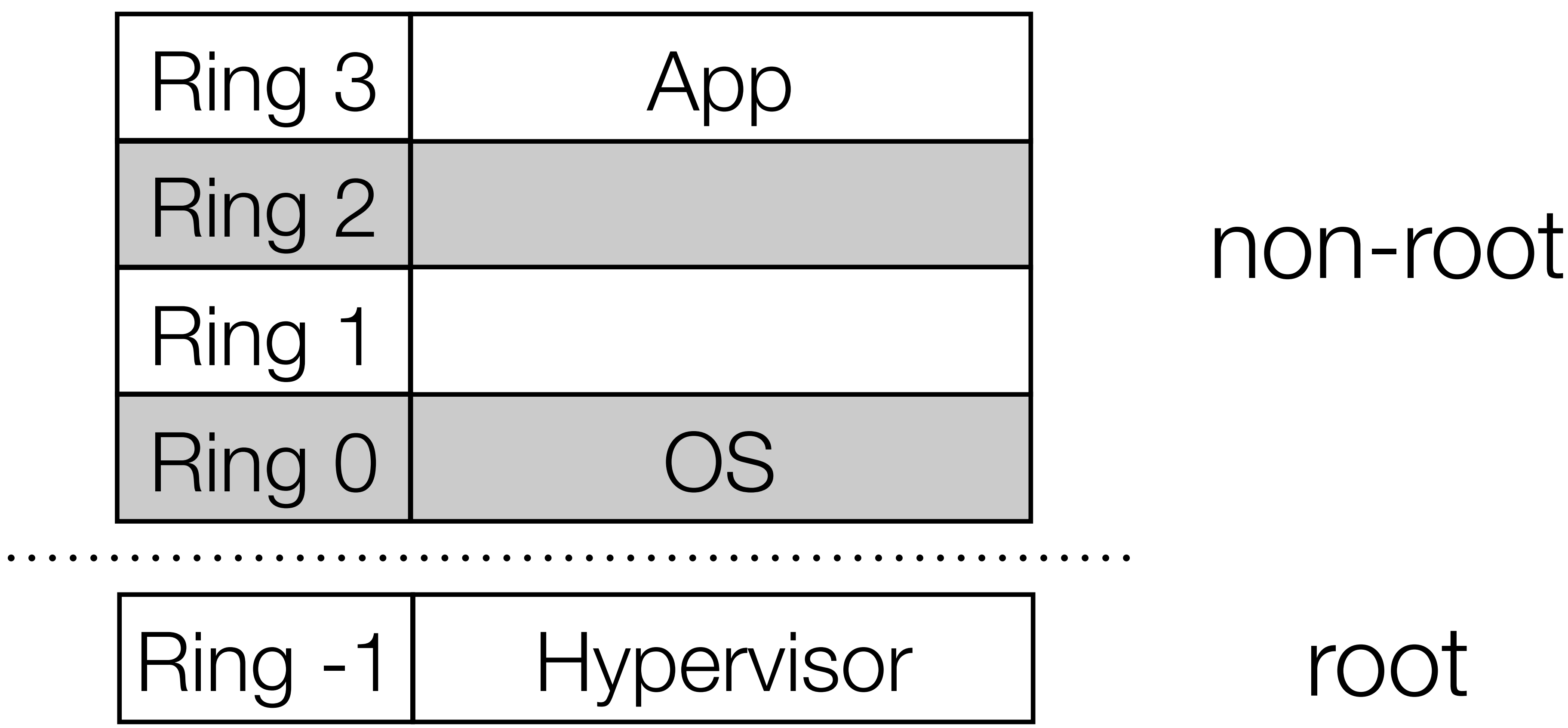
# Hardware-assisted

Originally the machine is executing normally, without any guest OS.

| | |
|---|---|
| Ring 3 | App |
| Ring 2 | |
| Ring 1 | |
| Ring 0 | OS |

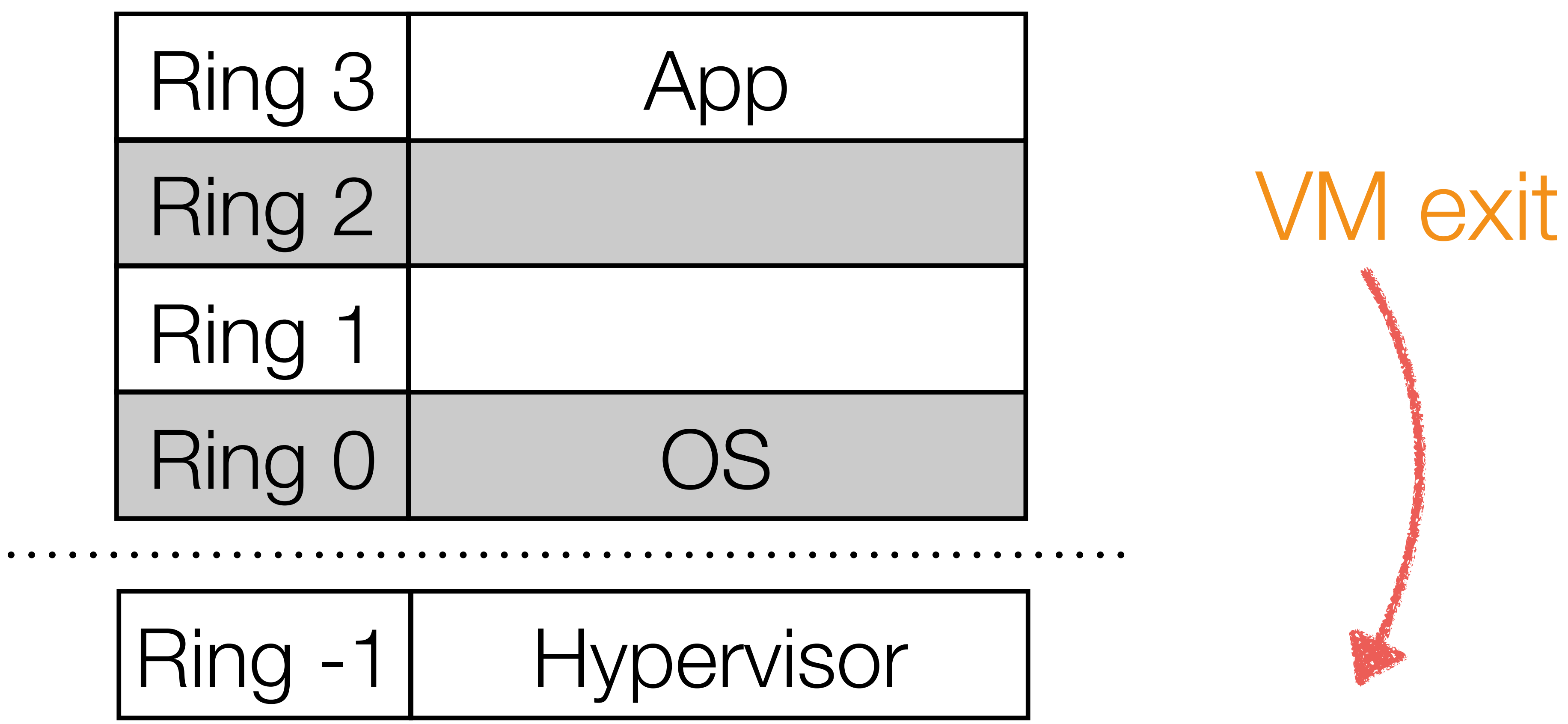
# Hardware-assisted

When the hypervisor launches a VM,



# Hardware-assisted

When the guest OS meets certain triggers, which requires the hypervisor to exercise system control, a transition of control happens:



# Hardware-assisted

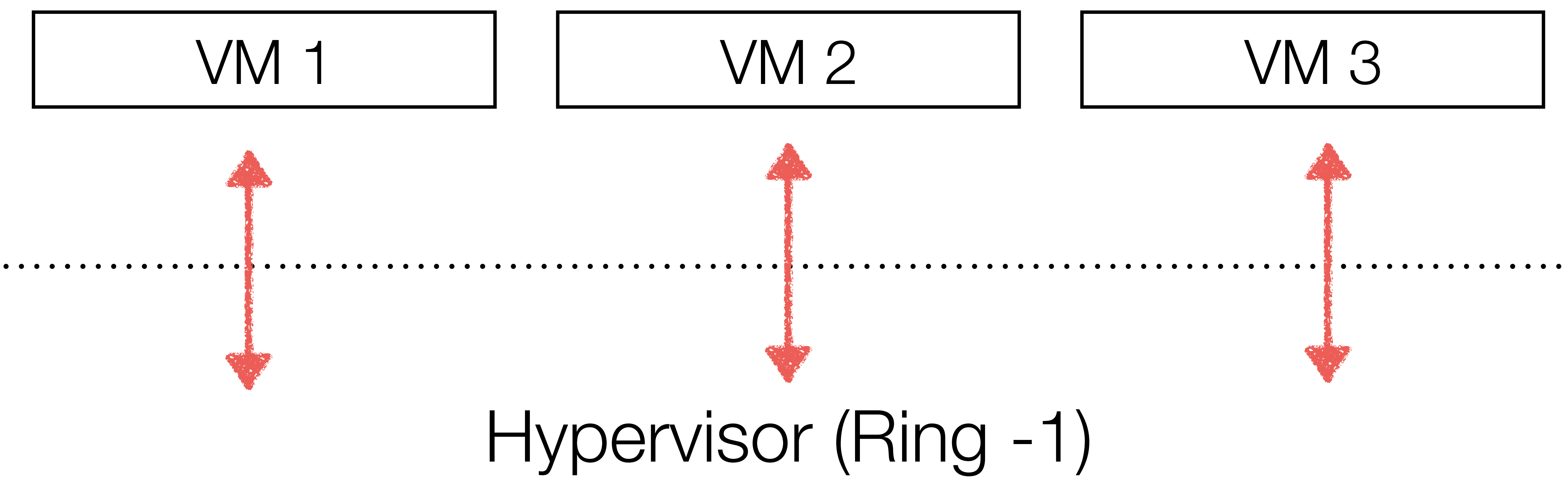
When the hypervisor finishes, the control switches back to non-root mode, the VM continues

| | |
|---|---|
| Ring 3 | App |
| Ring 2 | |
| Ring 1 | |
| Ring 0 | OS |
| Ring -1 | Hypervisor |

VM entrance

# Hardware-assisted

The bigger picture here:




Each VM is allocated a specific hardware address space for performance isolation.

# A Summary

| | Full | Para- | Hardware-assisted |
|---|---|---|---|
| Handling privileged insturctions | binary translation | hypercalls | non-root/ root mode |
| Guest OS modifications | no | yes | no |
| Performance | good | best | good |
| Examples | VMware, VirtualBox | Xen | Xen, VMware, VirtualBox, KVM |

# Credit

- Dr. WONG Tsz Yeung, CSCI 4180, CUHK
- Virtualization technology introduction, Intel Corporation. 28 July 2008.
- Wely Lau, wely@ncs.com.sg