COSC 3360/6310

Wednesday, February 3

Announcements

- A ***sample input*** for the first assignment is now on MS Teams
  - **`input10.txt`**
    - Checked for correctness
- ***First quiz*** on ***February 8*** at ***4pm*** on ***Blackboard***
  - A ***practice quiz*** will be ***online today at 7pm***
  - Will be remain available until Monday afternoon
  - Please mail me (**jfparis@uh.edu**) any issues you might have

Materials on the quiz

- Anything discussed in class, except assignment hints:
  - **A. Wednesday January 20.pptx**
  - **B. Monday January 25.pptx**
  - **C. Wednesday January 27.pptx**
  - **D. Monday February 1.pptx**
  - **E. Wednesday February 3.pptx**

Chapter I Introduction

Jehan-François Pâris
jfparis@uh.edu

Unix and Linux

UNIX (I)

- Started at Bell Labs in the early 70's as an attempt to build a sophisticated time-sharing system on a very small minicomputer.
- First OS to be almost entirely written in C
- Ported to the VAX architecture in the late 70’s at U. C. Berkeley:
  - Added virtual memory and networking

The fathers of UNIX

Ken Thompson and Denis Ritchie

UNIX (II)

- Became the standard operating systems for ***workstations***
  - Selected by Sun Microsystems
- Became less popular because
  - Too many variants
    - Berkeley BSD, ATT System V, …
  - PCs displaced workstations
  - Windows has a better user interface

UNIX Today

- Several ***free versions*** exist (FreeBSD, Linux):
  - Free access to source code
    - Ideal platform for OS research
- ***Apple OS X*** runs on the top of an updated version of BSD
- ***Android*** runs on top of a heavily customized Linux kernel
- ***Chrome*** runs on top of a vanilla Linux OS

A Rapid Tour

- UNIX kernel is the core of the system and handles the system calls
- UNIX has several shells: **sh, csh**, **ksh**, **bash**
- On-line command manual:
  - **man xyz**
    displays manual page for command **xyz**
  - **man 2 xyz**
    displays manual page for *system call* **xyz(…)**

Most Lasting Impact

- First OS that
  - Run efficiently on very different platforms
  - Had its source code made available to its users
- File system inspired most more recent OSes
- Remains the best platform for OS research

Kernel organizations

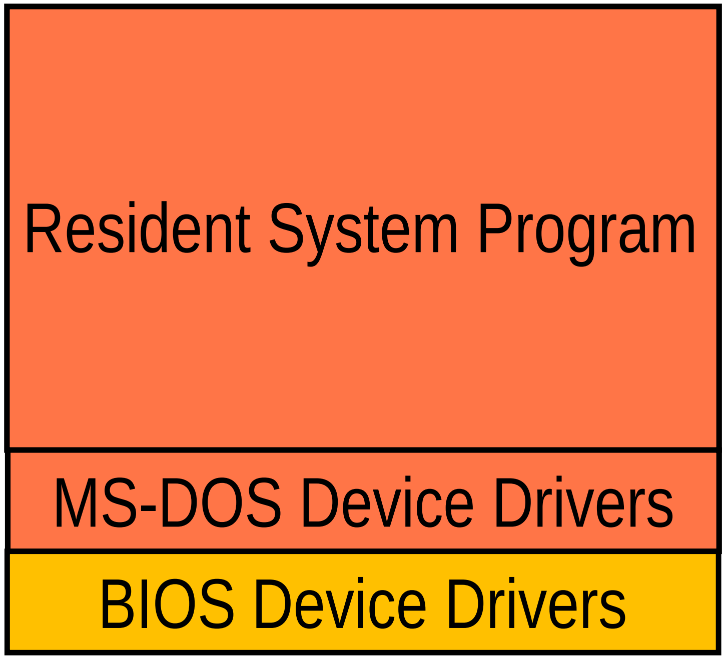
Kernel Organizations

- Three basic organizations:
  - ***Monolithic kernels***:
    - The default
  - ***Layered kernels:***
    - A great idea that did not work
  - ***Microkernels:***
    - Hurt by the high cost of context switches

Monolithic kernels

- No particular organization
  - All kernel functions share the same address space
  - This includes ***devices drivers*** and other ***kernel extensions***
- Lack of internal organization makes the kernel ***hard to manage, extend and debug***

MS-DOS (I)


Resident System Program
MS-DOS Device Drivers
BIOS Device Drivers

The BIOS

- Basic Input-Output System
- Stored on a chip
  - First ROM, now EEPROM
- Takes control of CPU when system is turned on
  - Identifies system components
  - Initiates booting of operating system
- Also provides low-level I/O access routines

The “curse”

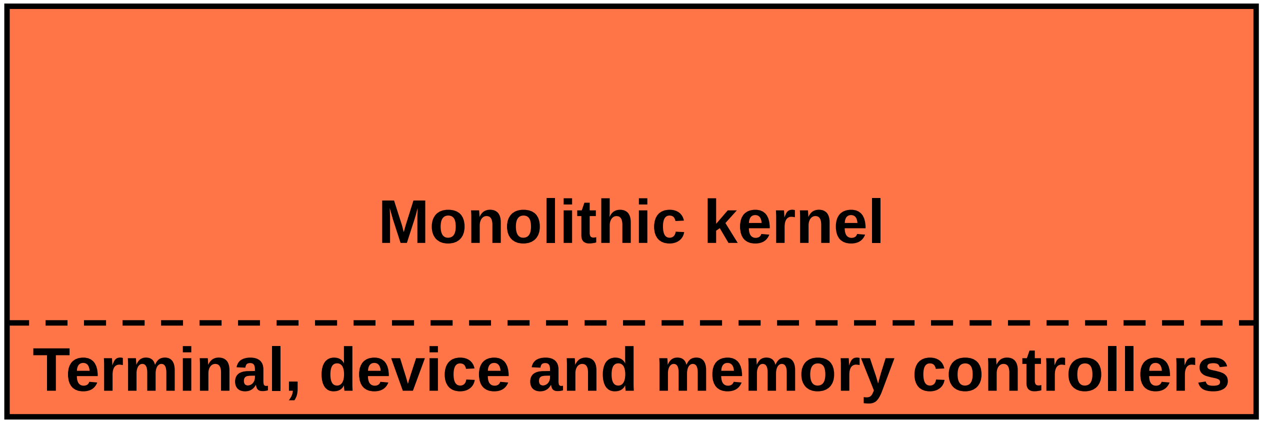
- Hardware lacked dual mode and hardware memory protection
  - Nothing prevented application programs from accessing directly the BIOS
  - Program accessing disk files through BIOS I/O routines assumed a given disk organization
    - Changing it became impossible

The solution

- For a long time, Microsoft could not make radical changes to its FAT-16 disk organization

- Windows XP and all modern operating systems prevent user programs from bypassing the kernel.

UNIX



- Monolithic kernel contains everything that is ***not device-specific*** including file system, networking code, and so forth.

Layered kernel

- Proposed by Edsger Dijkstra
- Implemented as a hierarchy of **layers**:
- Each layer defines a new data object
  - Hiding from the higher layers some functions of the lower layers
  - Providing some new functionality

THE operating system kernel

- (named after Dutch initials of T. U. Eindhoven)

User programs

Buffering for I/O devices

Operator console device driver

Memory management

CPU scheduling

Hardware

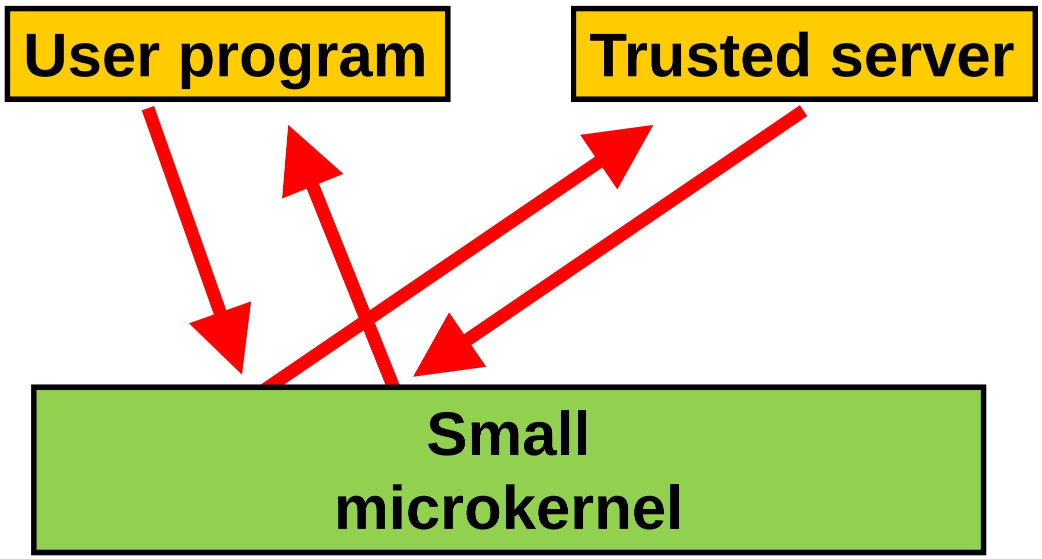
Limitations

- Layered design works extremely well for ***networking code***
  - Each layer offers its own functionality
- Much less successful for kernel design
  - No clear ordering of layers
    - Memory management uses file system features and vice versa

Microkernels

- A reaction against “bloated” monolithic kernels
  - Hard to manage, extend, debug and ***secure***
- Key idea is making kernel ***smaller*** by delegating ***non-essential tasks*** to ***trusted user-level servers***
  - *Same idea as **subcontracting***
- Microkernel keeps doing what cannot be delegated:
  - Security, short-term scheduling, …

How it works (I)



How it works (II)

- Microkernel
  - Receives request from user program
  - Decides to forward it to a user-level server
  - Waits for reply for server
  - Forwards it to user program
- ***Trusted servers*** run outside the kernel
  - Cannot execute privileged instructions

Advantages

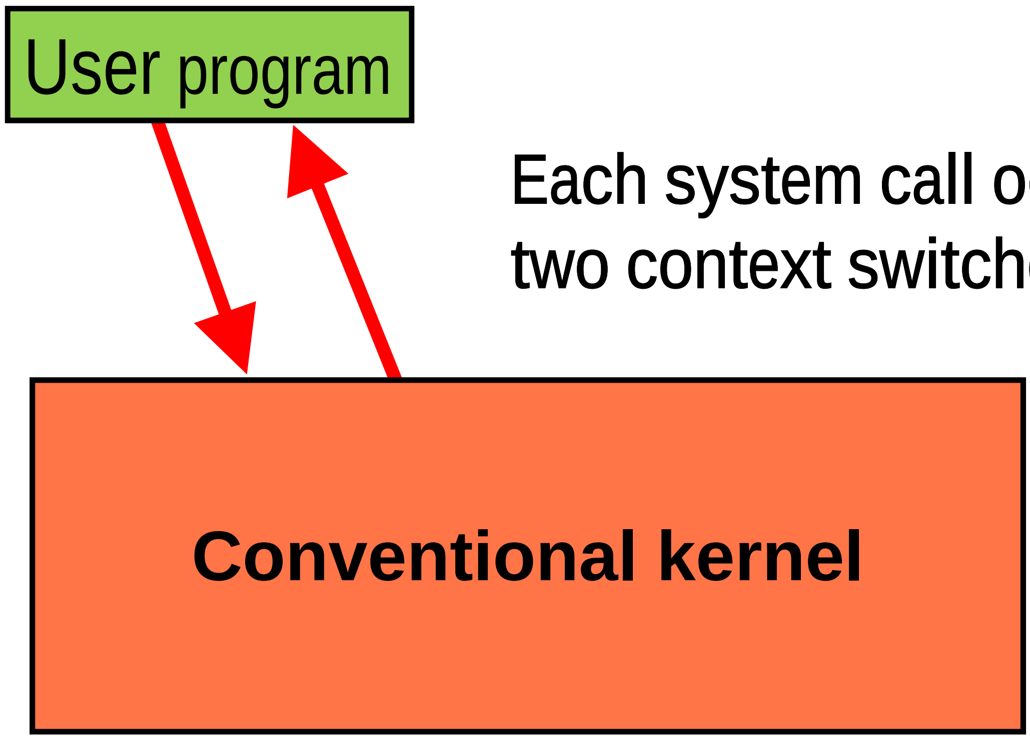
- Kernel is smaller, easier to secure and manage
- Servers run outside of the kernel
  - Cannot crash the kernel
  - Much easier to extend kernel functionality
    - Adding new servers
    - Adding an NTFS server to UNIX microkernel

Major disadvantage

- ***Too slow***
  - Four context switches instead of two

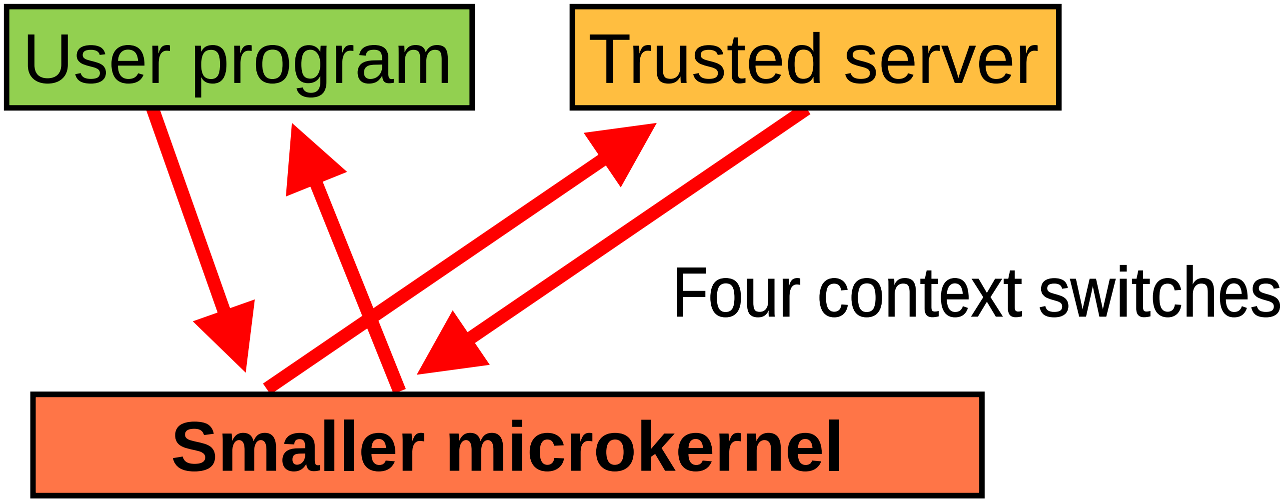
- *Speed remains an essential concern*
- *We don’t like to trade speed for safety (or anything else)*

A conventional kernel



Each system call occasions two context switches

A microkernel



Mach

- Designed in mid 80’s to replace UNIX kernel
- New kernel with different system calls
  - UNIX system calls are routed to an ***emulation server***
- Emulation server was d to run in user space
  - Slowed down the system
  - Server ended inside the kernel

MINIX 3

- MINIX 1 was designed for teaching OS internals
  - Predates Linux
- Now aimed at high reliability (embedded) applications
  - *More willing to trade space for reliability*
- Runs on x86 and ARM processors
- Compatible with NetBSD

MINIX 3 microkernel

- "Tiny" (12,700 lines) microkernel
  - Handles ***interrupts*** and ***message*** passing
  - Only code running in kernel mode
- Other OS functions are handled by ***isolated***, ***protected***, ***user-mode*** processes
  - Each device driver is a separate user-mode process
  - System automatically restarts ***crashed drivers***

Modular kernels

- Linux, Windows
- Modules are object files whose contents can be linked to—and unlinked from—the kernel at any time
  - Run inside the kernel address space
  - *Used to add to the kernel **device drivers** for new devices*

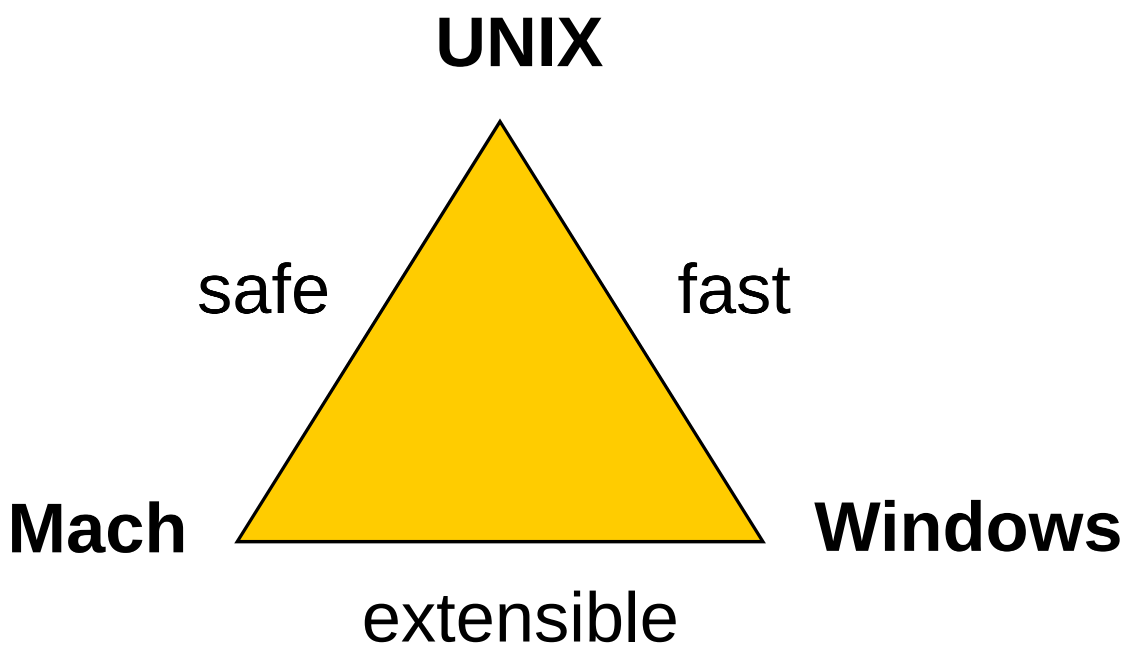
Advantages

- ***Extensibility:***
  - Can add new features the kernel
  - In many cases, the process is completely transparent to the user
- ***Lack of performance penalty:***
  - Modules run in the kernel address space

Disadvantages

- ***Lower reliability***
  - A bad module can corrupt the whole kernel and crash the system.
- ***Serious problem***
  - Many device drivers are poorly written
  - Device drivers account for 85% of reported failures of Windows XP

Current state of the art



Why?

- Unix has a monolithic kernel (which makes it fast) and does not allow extensions (which makes it both safe and non-extensible)
- Windows has a monolithic kernel (which makes it fast) and allows extensions (which makes it both extensible and unsafe)
- Mach allows extensions in user space (which makes it extensible, safe and slow)

Virtual machines

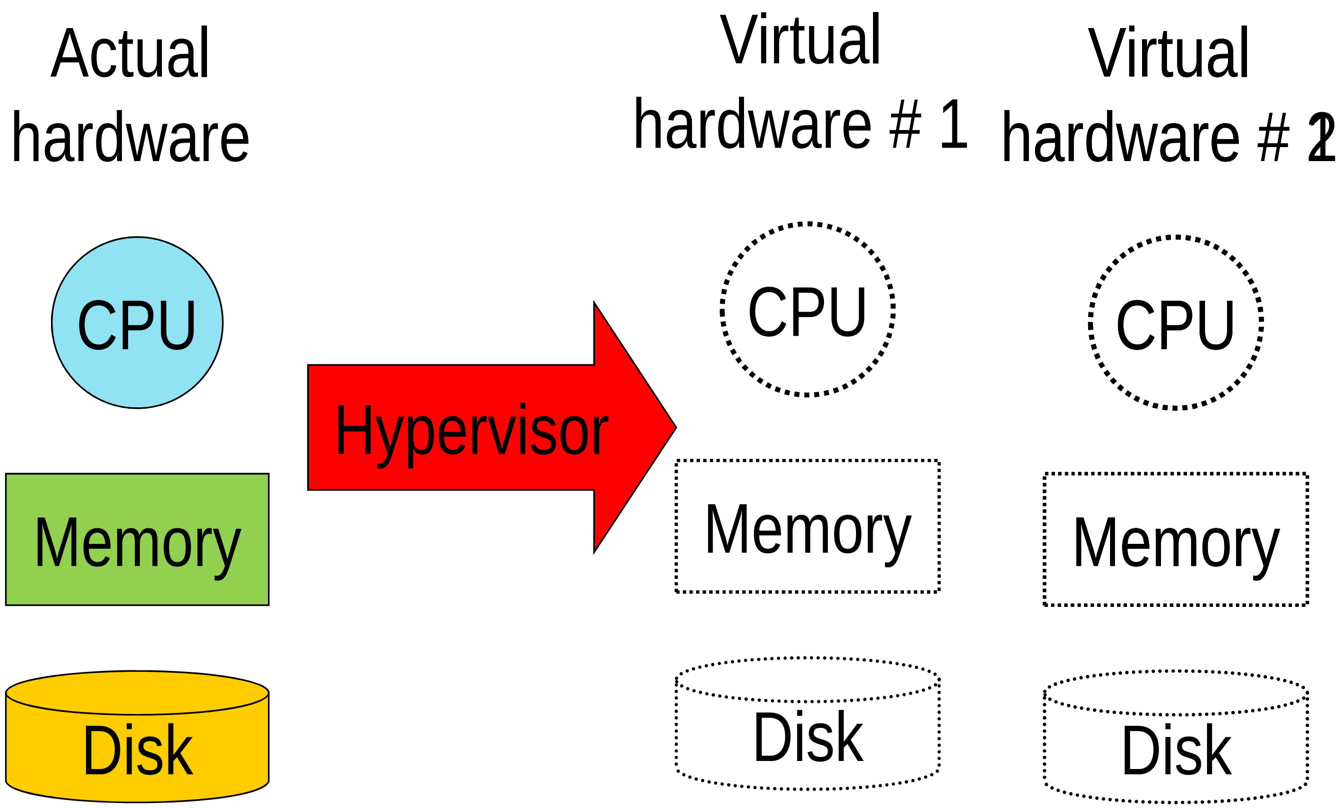
Virtual machines

- Let different operating systems run at the same time on a single computer
  - Windows, Linux and Mac OS
  - A real-time OS and a conventional OS
  - A production OS and a new OS being tested

How it is done

- A ***hypervisor*** /***VM monitor*** defines two or more virtual machines
  - Each virtual machine has
    - Its own virtual CPU
    - Its own virtual physical memory
    - Its own virtual disk(s)
- Can also install VM on top of a ***host OS***
  - ***VM Box***

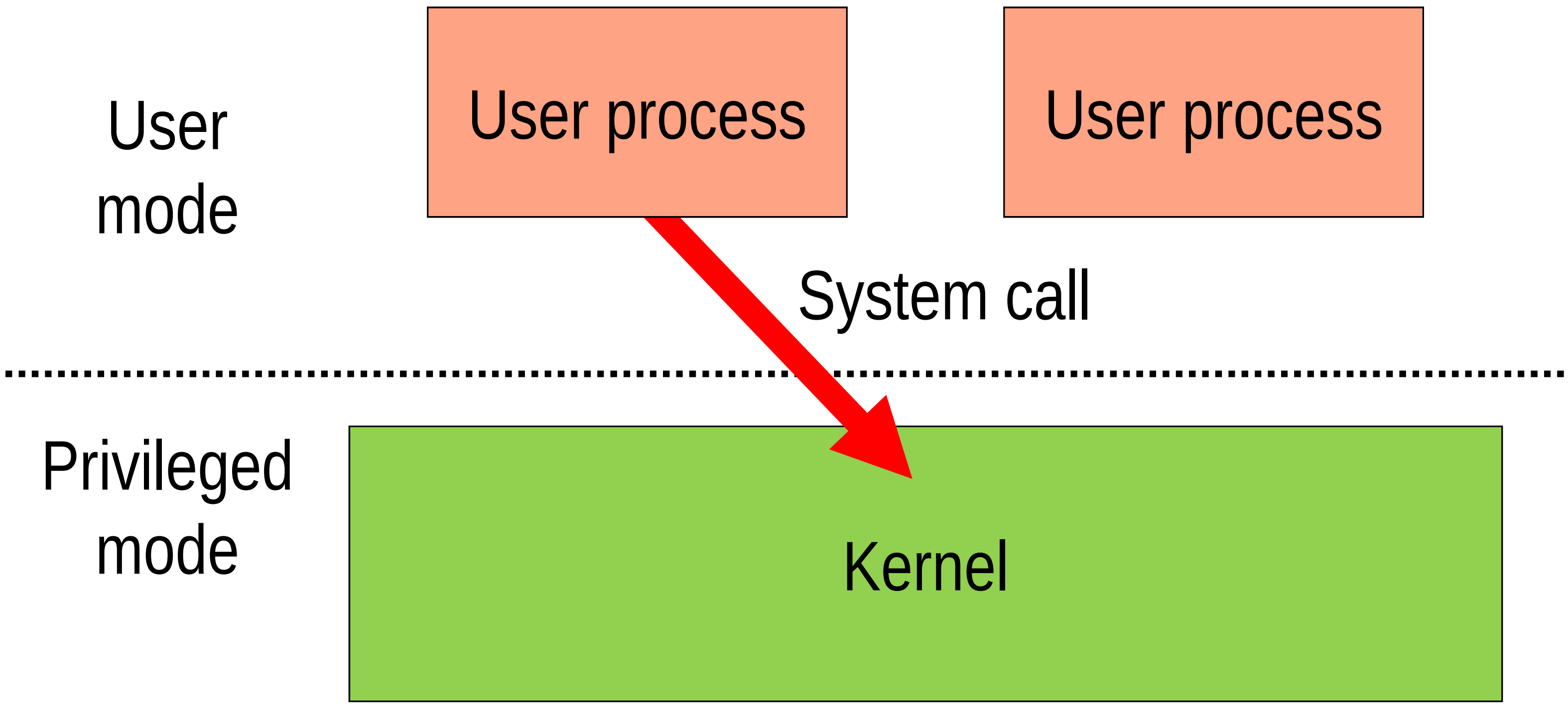
The virtualization process



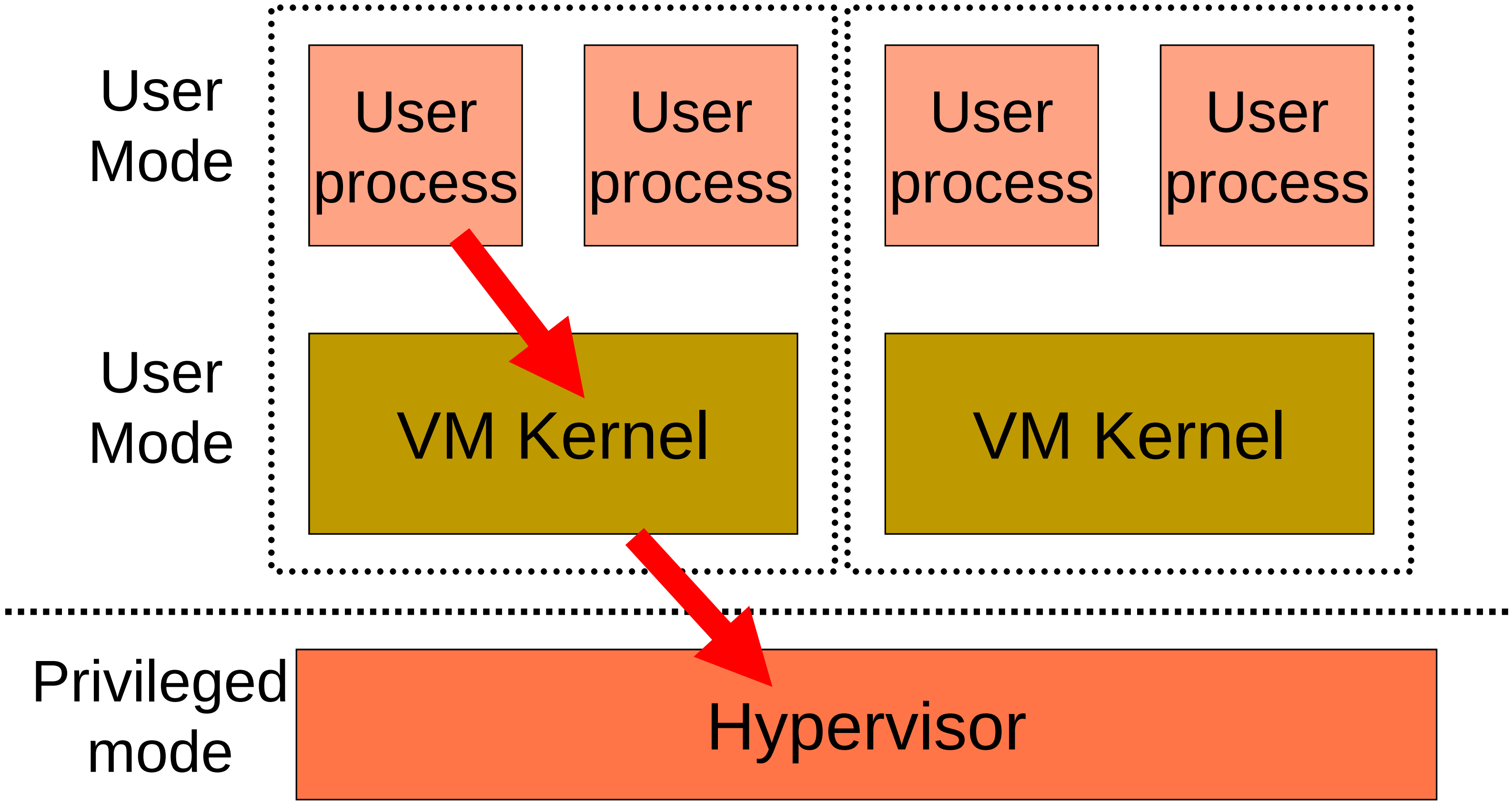
Reminder

- In a conventional OS,
  - Kernel executes in ***privileged/supervisor mode***
    - Can do virtually everything
  - User processes execute in ***user mode***
    - Cannot modify their page tables
    - Cannot execute privileged instructions

A conventional architecture



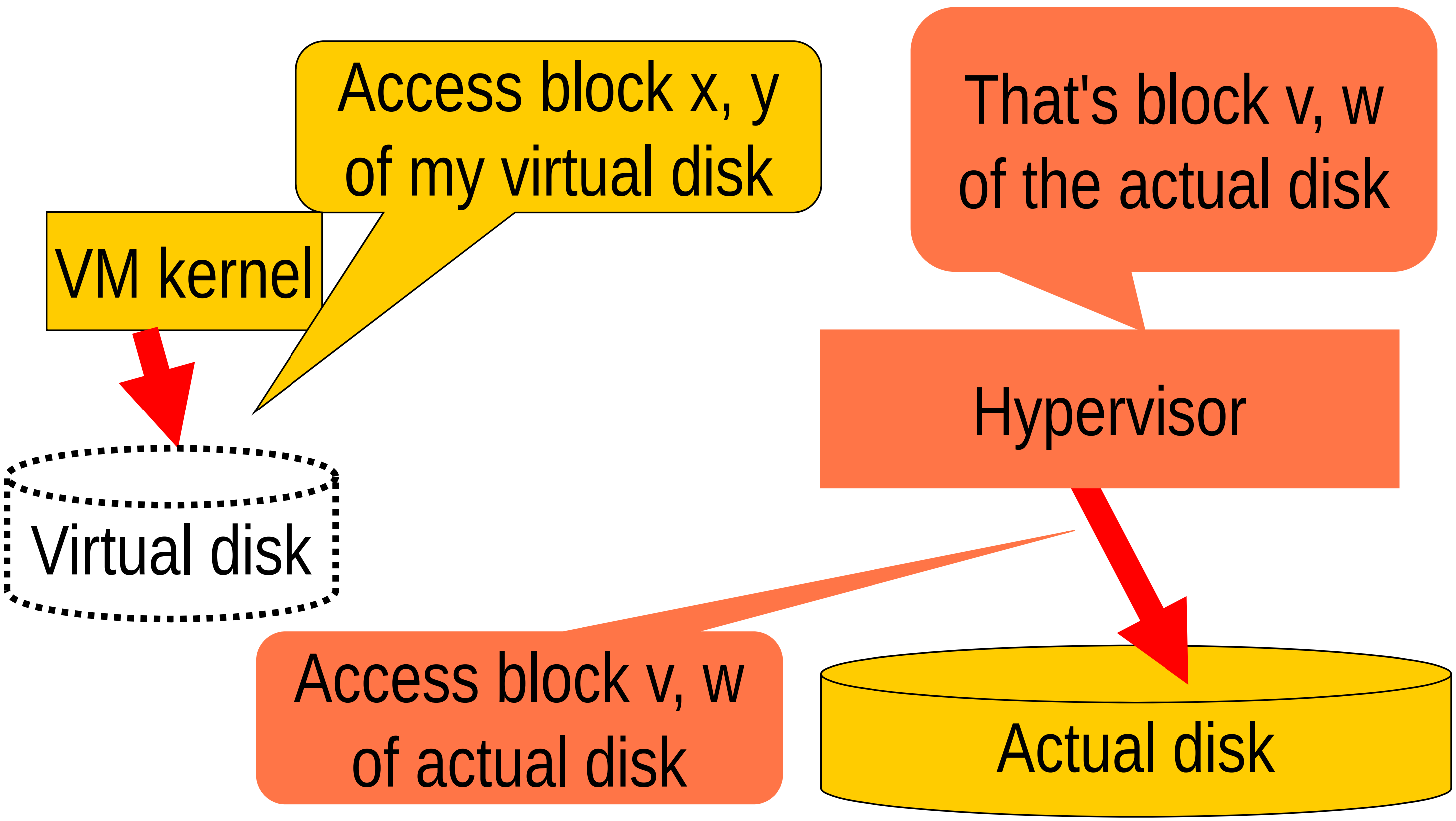
Two virtual machines



Explanations (II)

- Whenever the kernel of a VM issues a privileged instruction, an interrupt occurs
  - The hypervisor takes control and do the physical equivalent of what the VM attempted to do:
    - Must convert virtual RAM addresses into physical RAM addresses
    - Must convert virtual disk block addresses into physical block addresses

Translating a block address



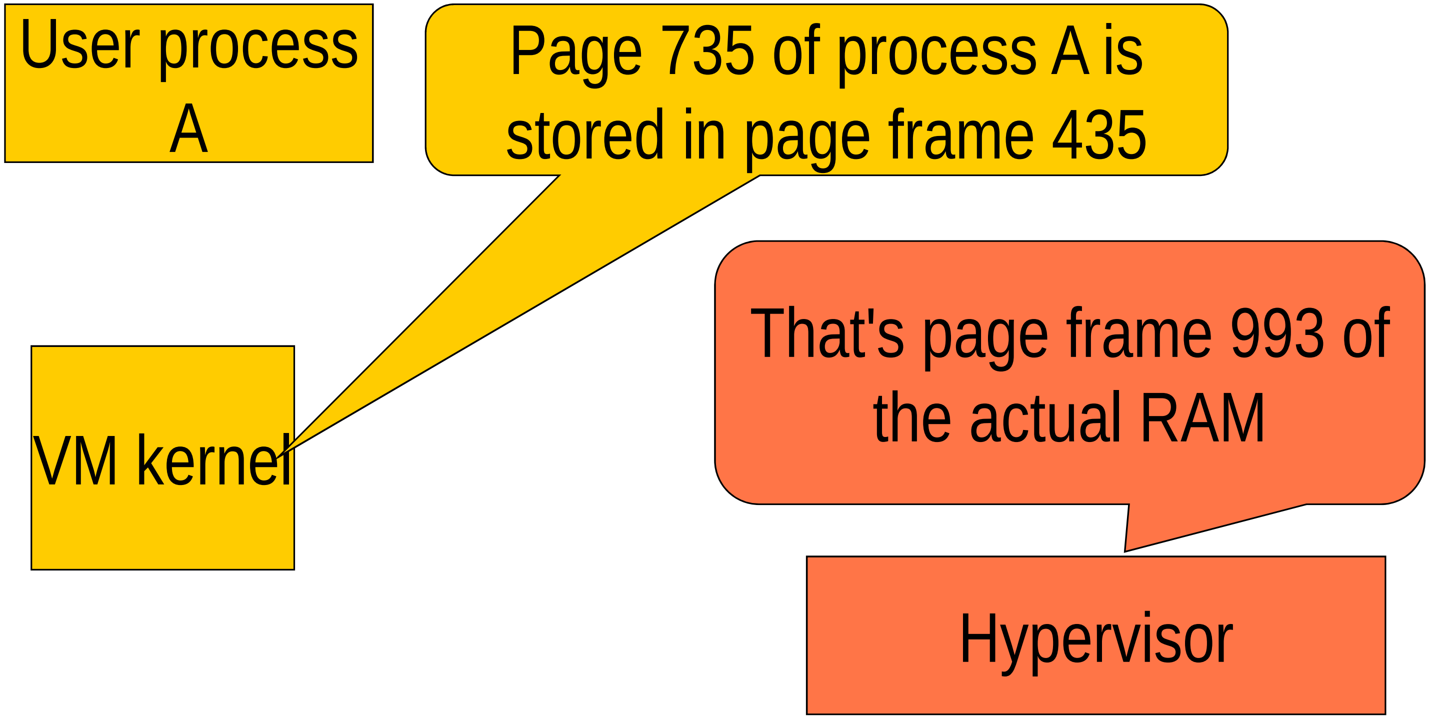
Handling I/Os

- Difficult task because
  - Wide variety of devices
  - Some devices may be shared among several VMs
    - Printers
    - Shared disk partition
      - *Want to let Linux and Windows access the same files*

Virtual Memory Issues

- Each VM kernel manages its own memory
  - Its page tables map program virtual addresses into ***what it believes to be physical addresses***

The dilemma


User process A
Page 735 of process A is stored in page frame 435
VM kernel
That's page frame 993 of the actual RAM
Hypervisor

Nastiest Issue

- The whole VM approach assumes that a kernel executing in user mode will behave exactly like a kernel executing in privileged mode except that privileged instructions will be trapped
- ***Not true for all architectures!***
  - ***Intel x86 Pop flags (POPF) instruction***
  - …

The Virtual Box Solution

- Code Scanning and Analysis Manager (CSAM)
  - Scans privileged code recursively before its first execution to identify problematic instructions
  - Calls the Patch Manager (PATM) to perform *in-situ* patching.

The Xen solution

- Modify the guest kernel to eliminate badly behaving instructions such as POPF
  - ***Paravirtualization***
  - Faster but less flexible
    - Requires open-source kernel

User programs are not affected
❖ Only the kernel

Containers

- Each VM runs its own copy of the kernel
  - Takes memory space
- Containers provide isolated user-space instances that share the same kernel
  - Less overhead
  - Less flexibility
- Docker, LYXC