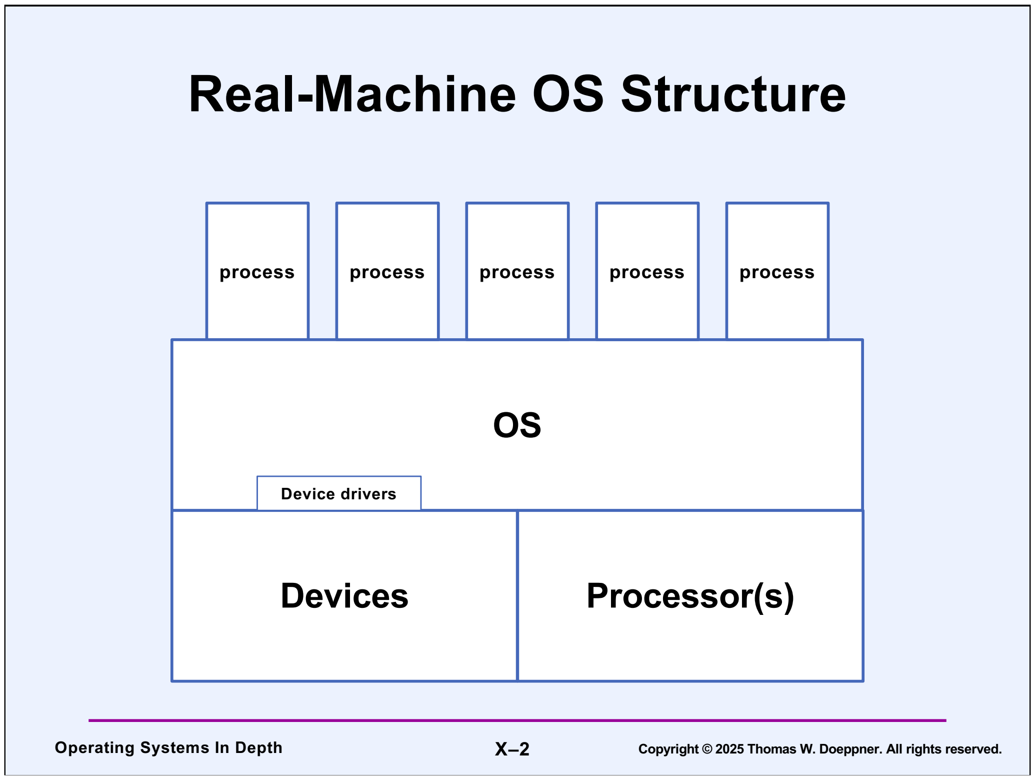
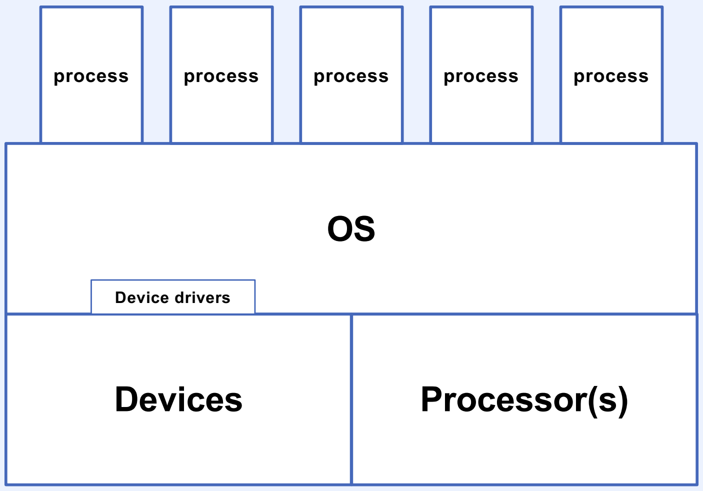
# Virtual Machines

## Part 2: starting ~20 years ago

(continued)

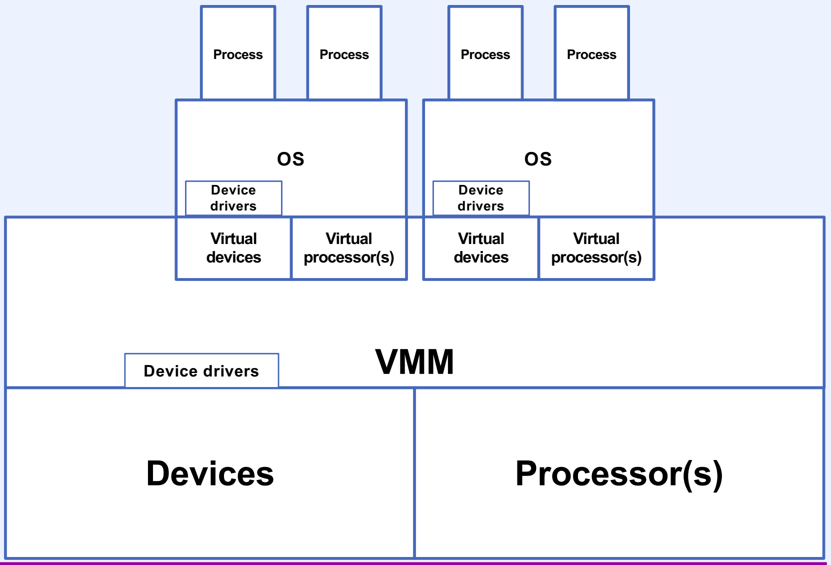


## Real-Machine OS Structure

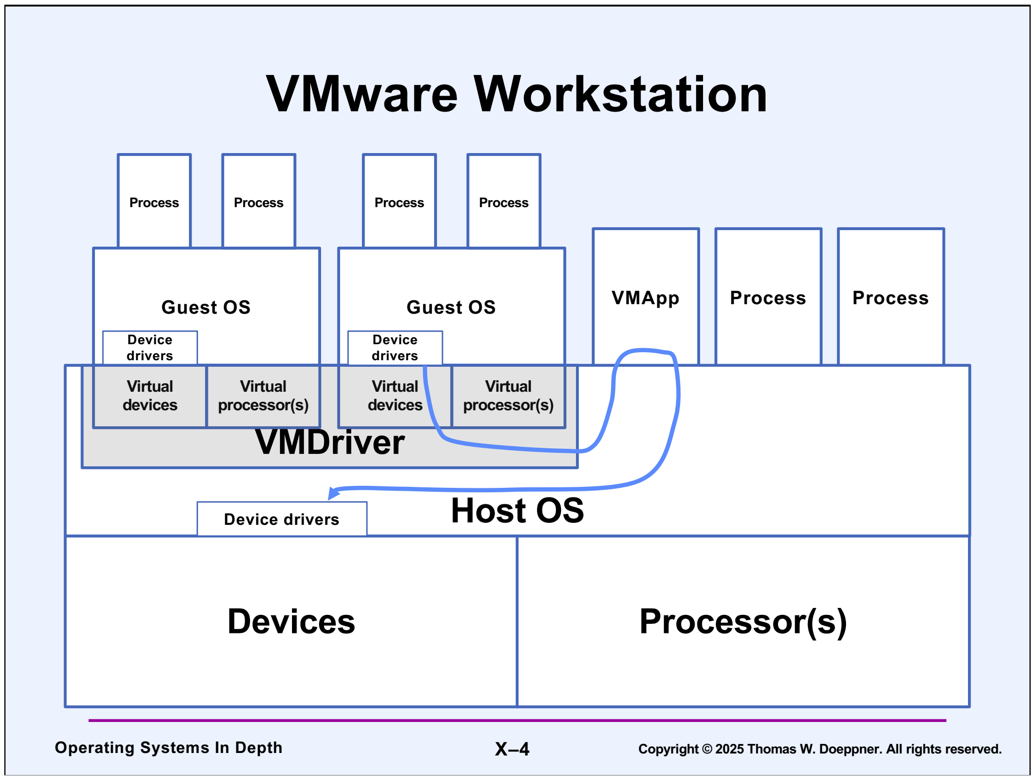


The OS supplies the device drivers needed to access devices.

## On a Virtual Machine …



The VMM requires device drivers as well so it can access the real devices. However, such drivers are written for "standard" operating systems, such as Windows, OS X, and Linux. The VMM is different from all three and probably requires a different structure for its drivers.



## VMware Workstation

In VMWare workstation, the VMM (known as **VMDriver**) is part of a standard OS, such as Linux. When a guest OS is running, it runs on top of a virtual machine provided by VMDriver. But the host OS (Linux in this case) runs directly on the hardware. When VMDriver accesses real devices, it places up calls to **VMApp**, a user application running on the host OS. VMApp places system calls into the host OS to access the device. Each virtual machine (supported by VMDriver) appears as a separate process to the host OS.

VMDriver appears to the host OS as a device driver, making it easy to install into the OS without requiring changes to it.

## Quiz 1

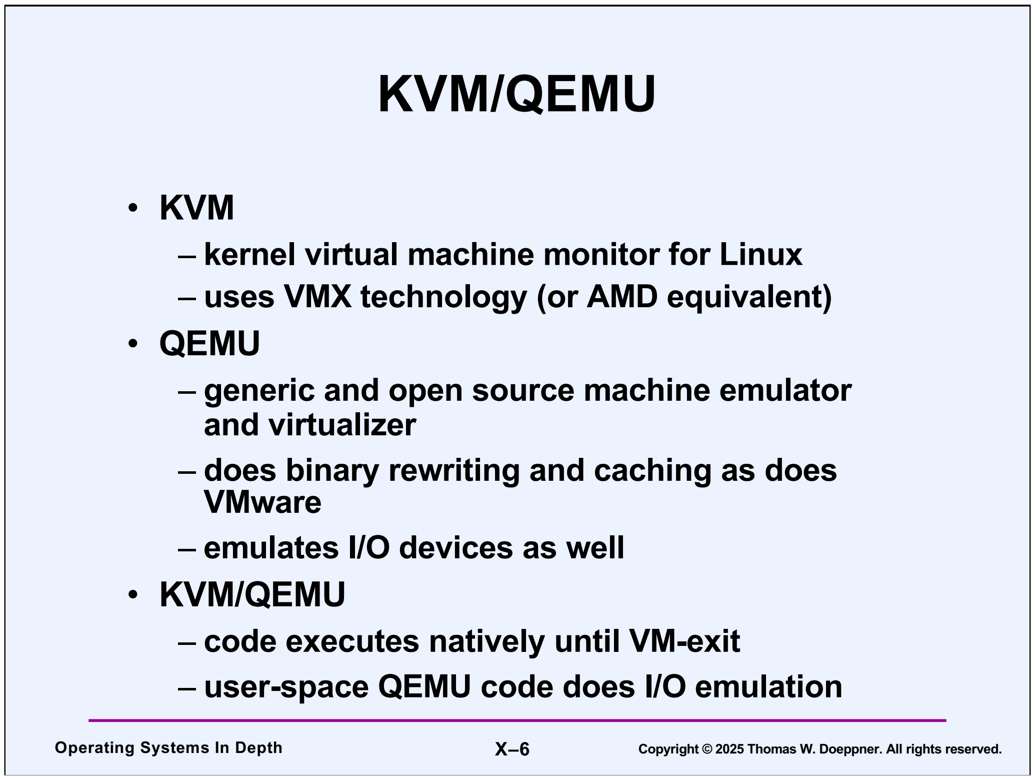
The host OS contains a scheduler that multiplexes the execution of threads on its processors. Each guest OS contains a scheduler that does the same thing.

Assume all threads have the same priority. Assume just one real processor.

a) Threads on the guest OS compete equally for the real processor as do threads on the host OS

b) Threads on the guest OS effectively have higher priority than host OS threads

c) Threads on the guest OS effectively have lower priority than host OS threads



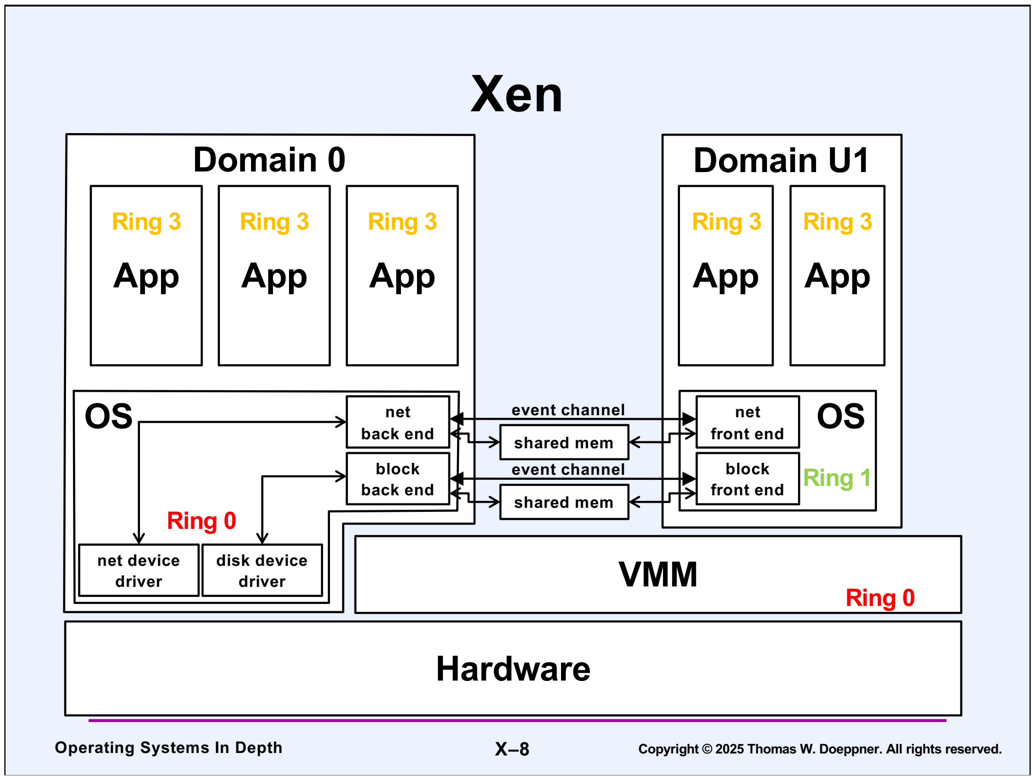
## KVM/QEMU

- KVM
  - kernel virtual machine monitor for Linux
  - uses VMX technology (or AMD equivalent)
- QEMU
  - generic and open source machine emulator and virtualizer
  - does binary rewriting and caching as does VMware
  - emulates I/O devices as well
- KVM/QEMU
  - code executes natively until VM-exit
  - user-space QEMU code does I/O emulation

QEMU stands for “quick emulator”.

## Paravirtualization

- Sensitive instructions replaced with hypervisor calls
  - traps to VMM
- Virtual machine provides higher-level device interface
  - guest machine has no device drivers



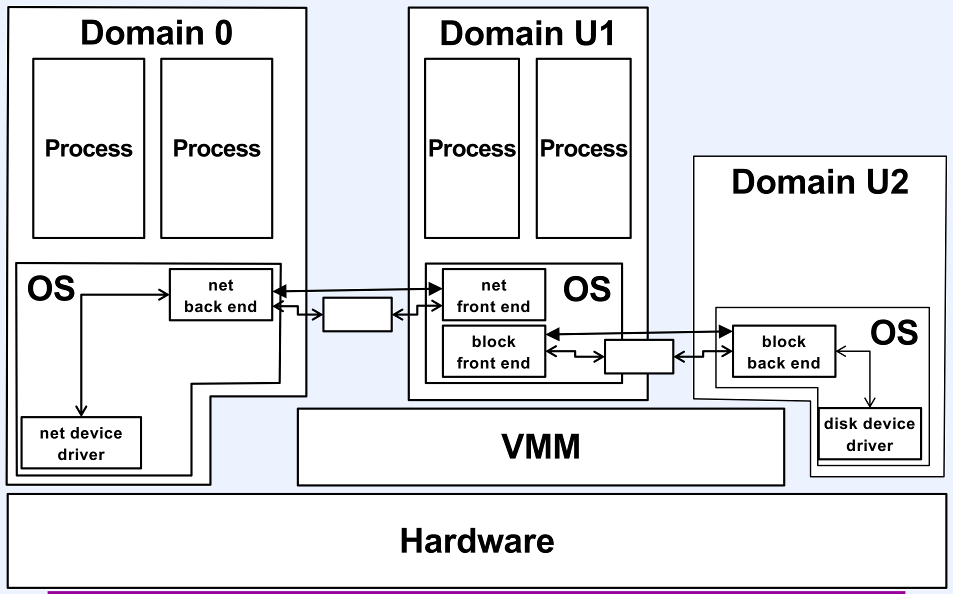
## Xen

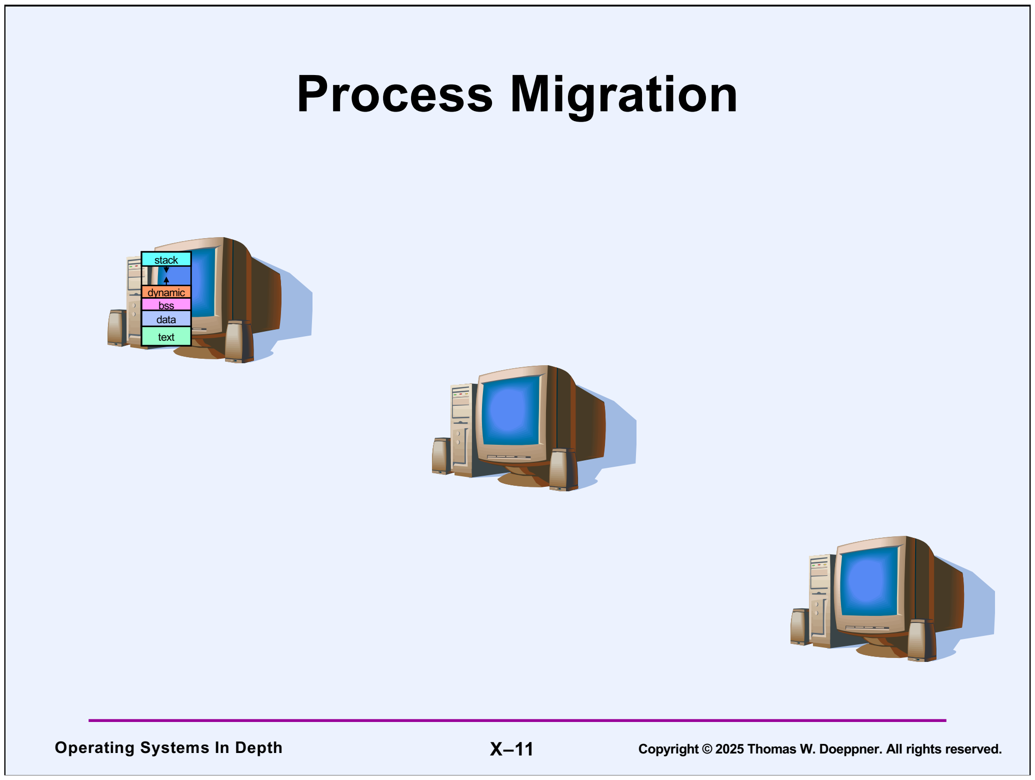
The VMM and the host OS share the kernel – both run in ring 0. ”Domain 0” contains the host OS and its processes, running on the real hardware. The VMM supports multiple VMs, such as that of Domain U1. Domain U! contains an operating system that’s been modified so as not to do I/O directly, but to send requests to the OS in Domain 0.

## Additional Applications

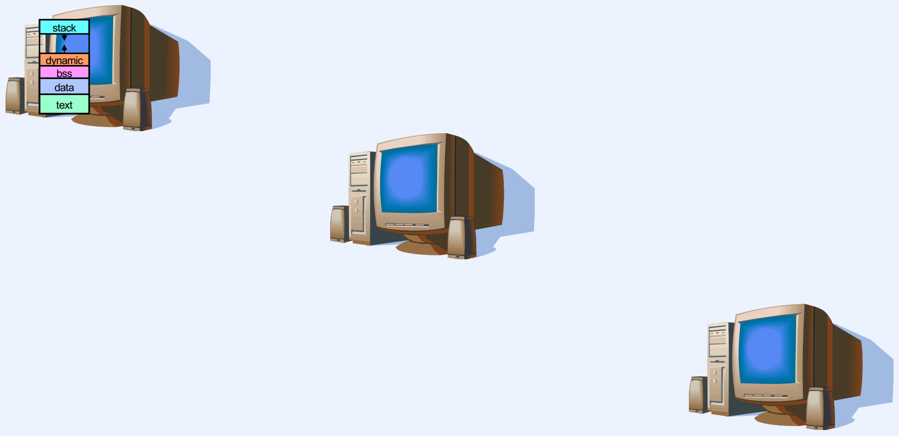
- Sandboxing
  - isolate web servers
  - isolate device drivers
- Migration
  - VM not tied to particular hardware
  - easy to move from one (real) platform to another

# Xen with Isolated Driver



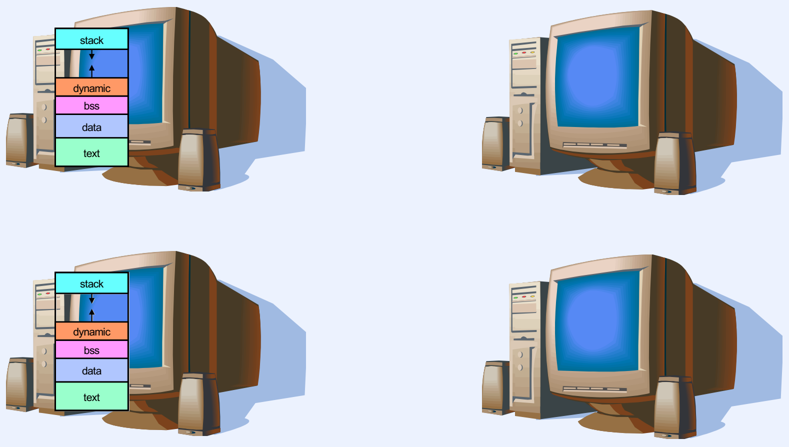


## Process Migration

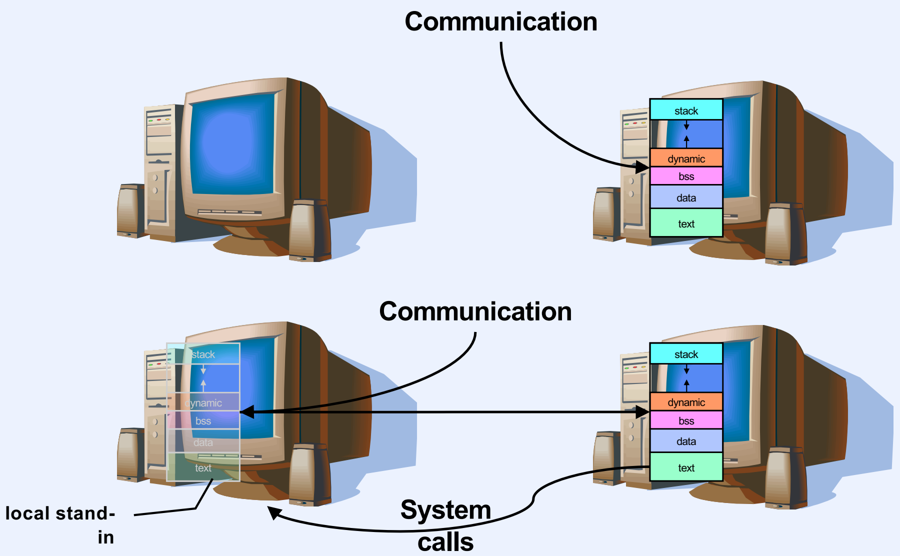


A good survey article on process migration is “Process Migration,” by D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, ACM Computing Surveys, Vol. 39., No. 3 (September 2000).

# Approaches: Before



## Approaches: After



Some portions of a process's kernel context may be difficult or impossible to move, particularly if they are shared with other processes. An example of such difficult-to-move context is a process's communication state, which, at the least, is tied to the home machine because of its IP address.

## Virtual-Machine Migration

- Virtual machines are isolated
  - by definition!
- State is well defined
  - thus easy to identify and move
  - possible exception of virtual memory

## Transferring Virtual Memory

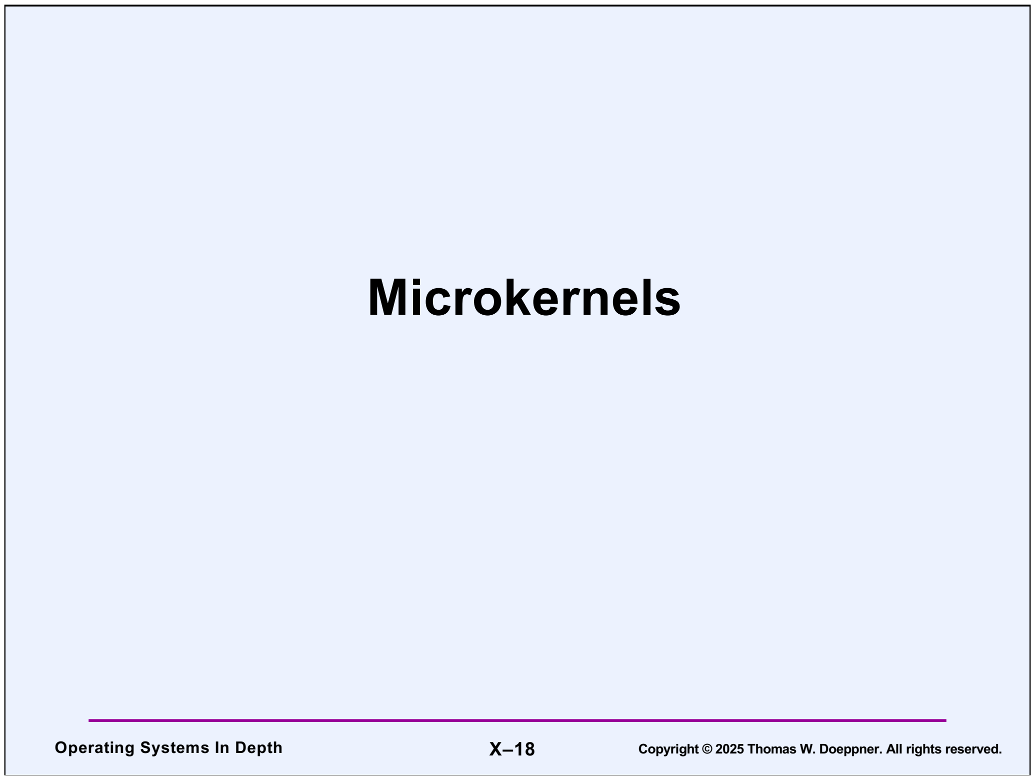
- Eager
  - all
  - dirty
    - (clean pages come from common source)
- Lazy
  - copy on reference
- Straightforward
  - flush everything to file system on source, then access file system on target
- Weird
  - precopy

## Eager–Dirty

- Freeze process on source
- Transfer all dirty pages to target
- Resume process on target

## Precopy

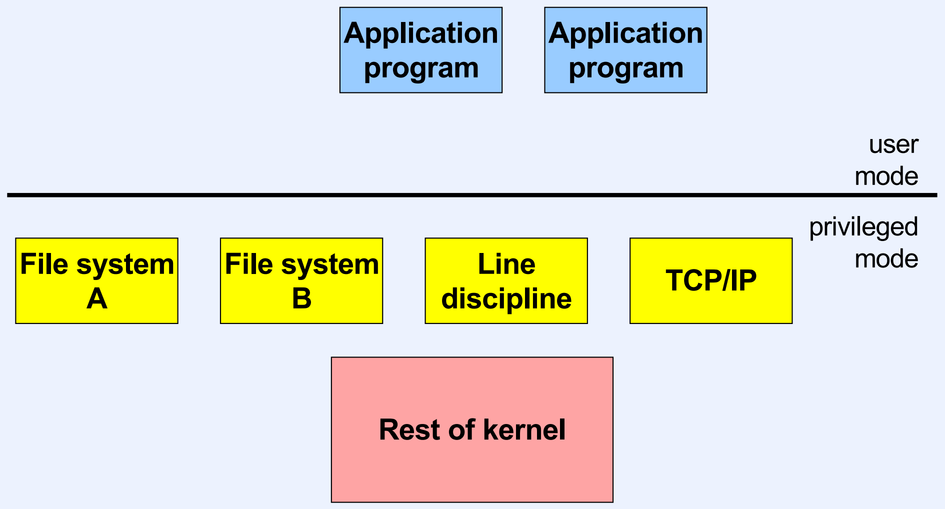
- While process still running on source
  - transfer dirty pages to target (eager–dirty)
- While more than x pages dirty on source
  - transfer newly dirtied pages to target
- Freeze process on source
- Transfer remaining dirty pages to target
- Resume process on target



## Microkernels

Details about Mach can be obtained from “Mach 3 Kernel Principles” by Keith Loepere, available at http://www.shakthimaan.com/downloads/hurd/kernel_principles.pdf.

## Traditional OS Organization



## Quiz 2

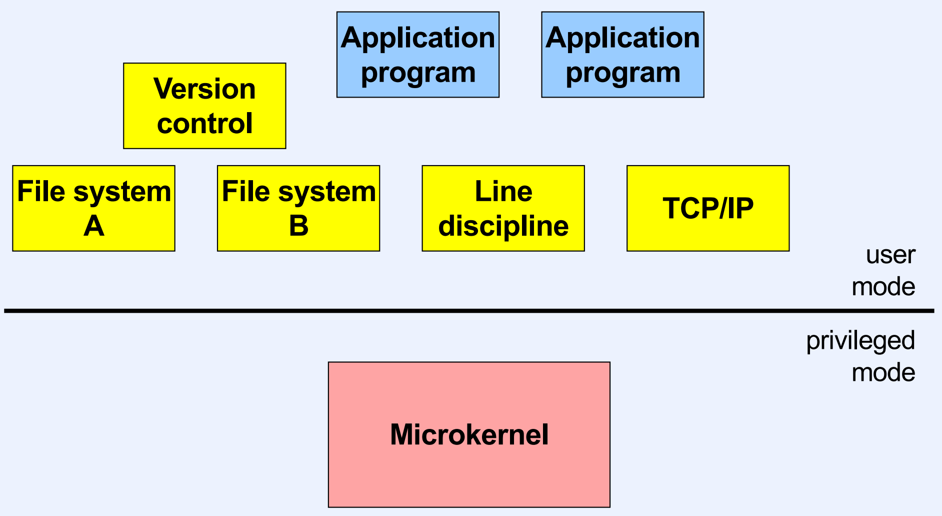
**In the previous slide, assume each of the two application programs runs as a separate process. What’s in the slide employs:**

**a) two address spaces: one for each process, with the kernel existing in a shared portion of the two process address spaces**

**b) three address spaces: one for each process, and one for the kernel**

**c) six address spaces: one for each process, one for each of the four kernel components, and one for the rest of the kernel**

# OS Services as User Apps



## Why?

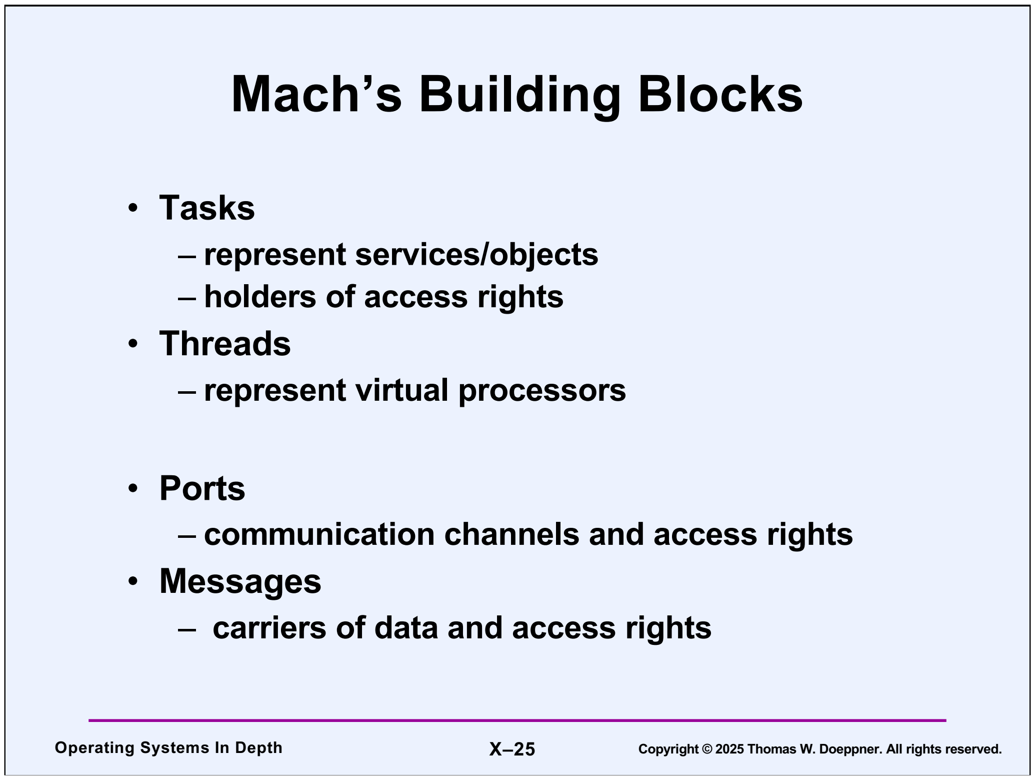
- It’s cool …
- Assume that OS coders are incompetent, malicious, or both …
  – OS components run as protected user-level applications
- Extensibility
  – easier to add, modify, and extend user-level components than kernel components

## Implementation Issues

- **What are the building blocks?**
- **What is run in privileged mode?**

# Mach

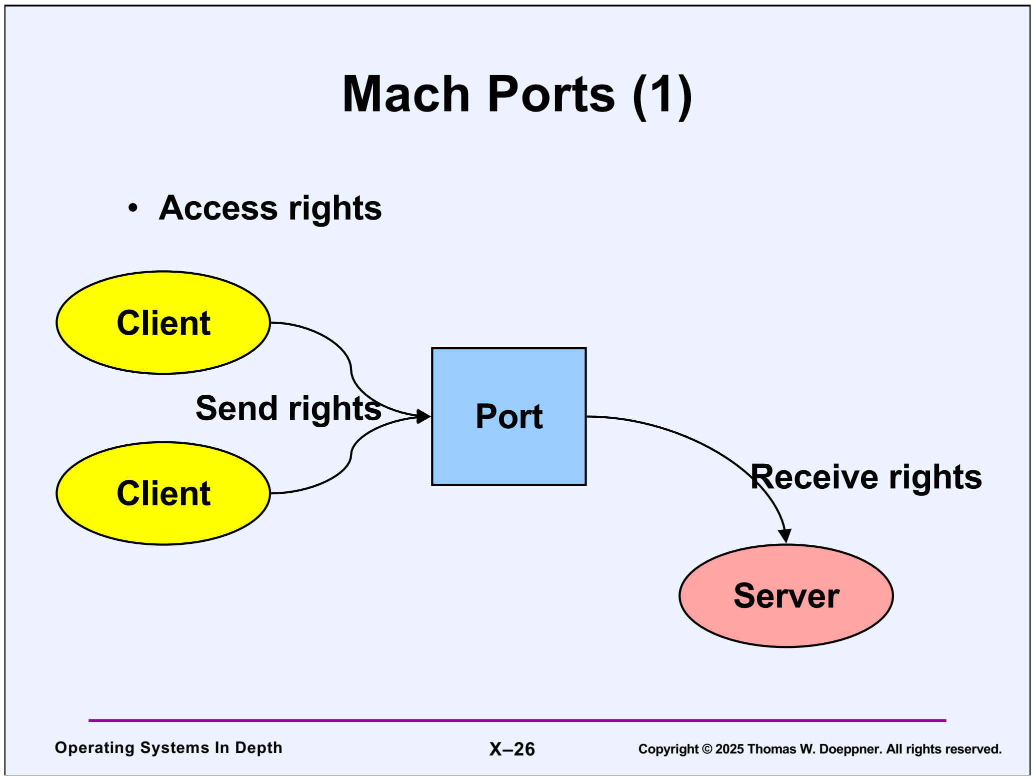
- Developed at CMU, then Utah
- Early versions shared kernel with Unix
  - basis of NeXT OS
    - basis of Apple OS X
- Later versions still shared kernel with Unix
  - basis of OSF/1
- Even later versions actually functioned as working microkernel
  - basis of GNU/HURD project
    - HURD: HIRD of Unix-replacing daemons
    - HIRD: HURD of interfaces representing depth



## Mach's Building Blocks

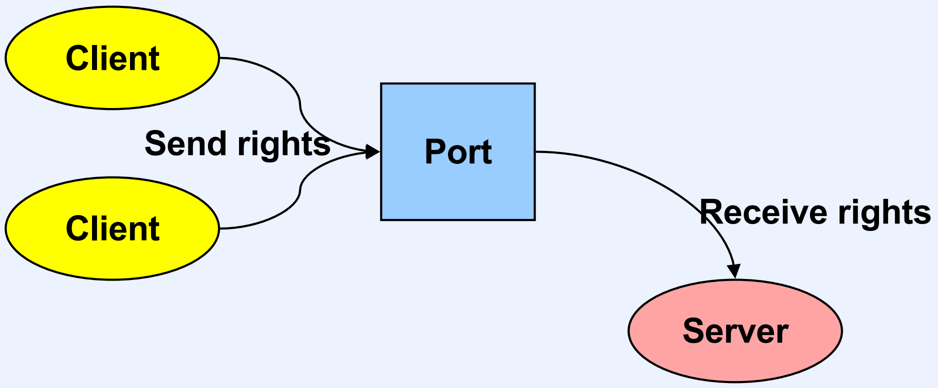
- Tasks
  - – represent services/objects
  - – holders of access rights
- Threads
  - – represent virtual processors
- Ports
  - – communication channels and access rights
- Messages
  - – carriers of data and access rights

Tasks may be used to represent objects. A task itself is an object, as is a thread. There are additional objects implemented by the kernel, including device objects

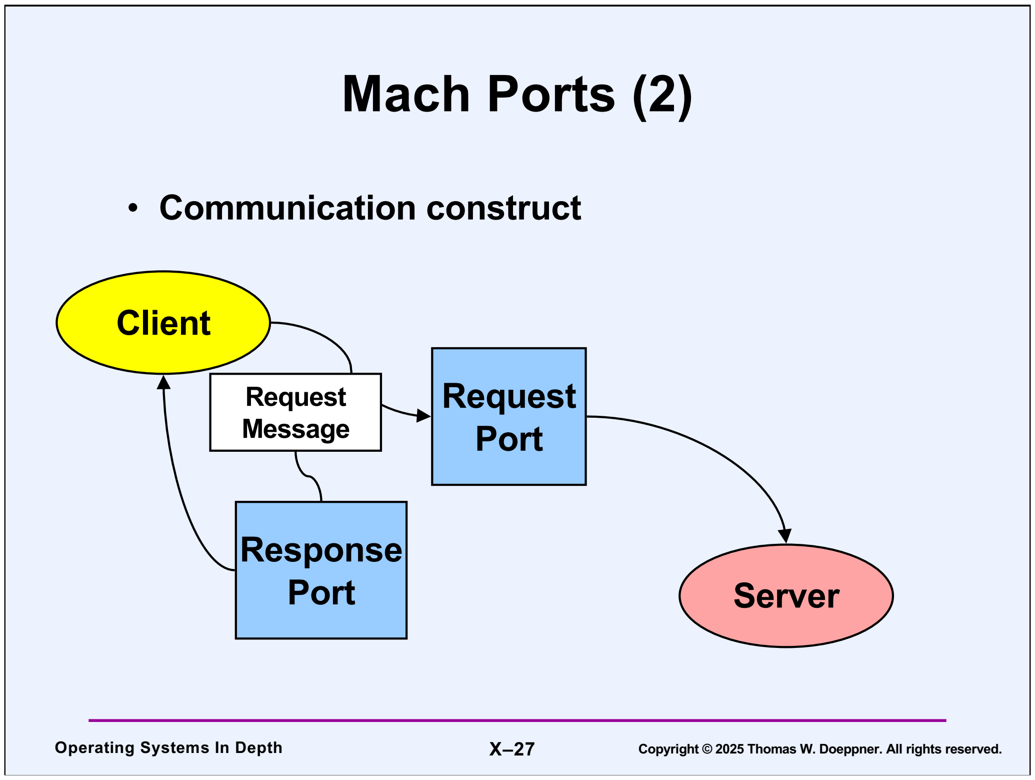


## Mach Ports (1)

- Access rights

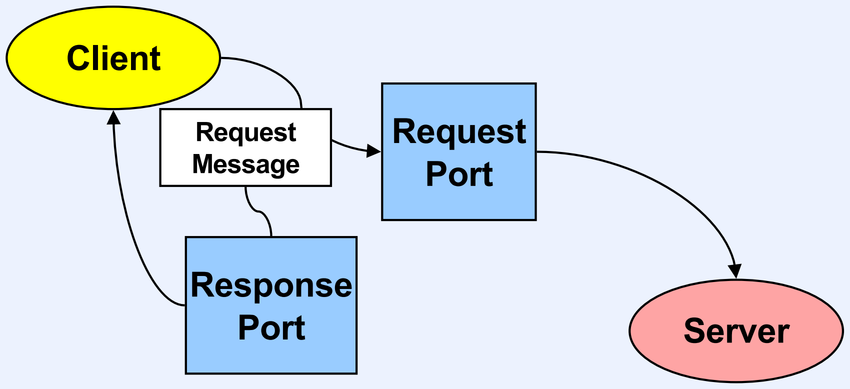


For any particular port, exactly one task has receive rights, but any number may hold send rights. Thus a port represents a one-way communication channel from the holders of send rights to the holder of receive rights.

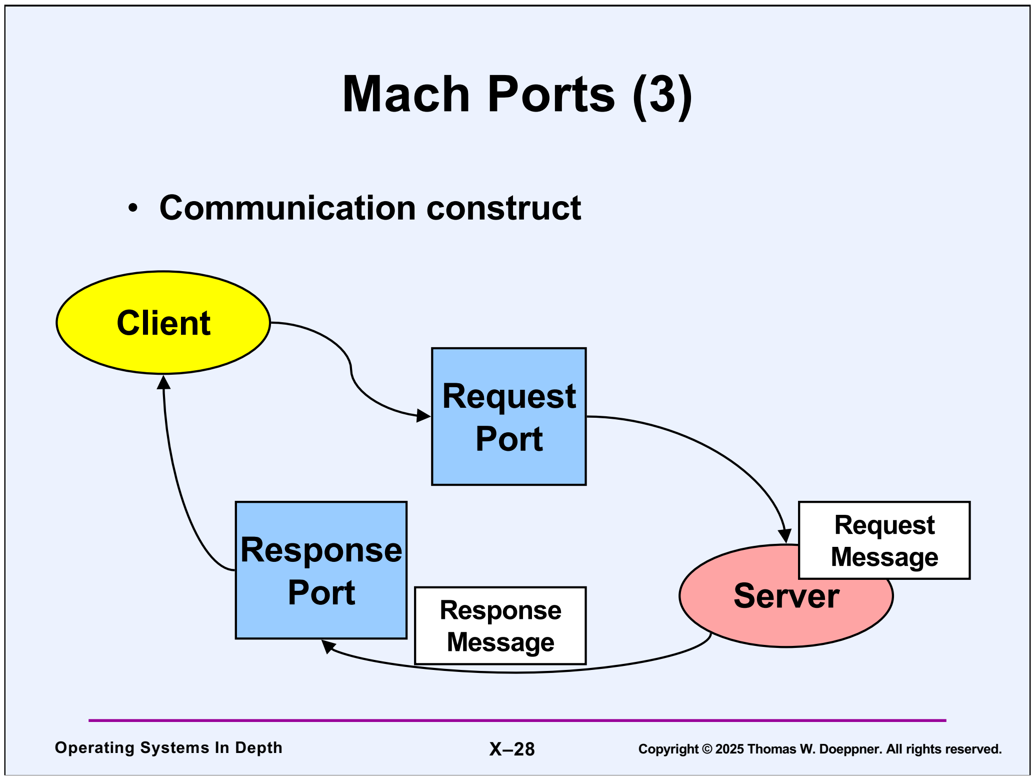


## Mach Ports (2)

- Communication construct

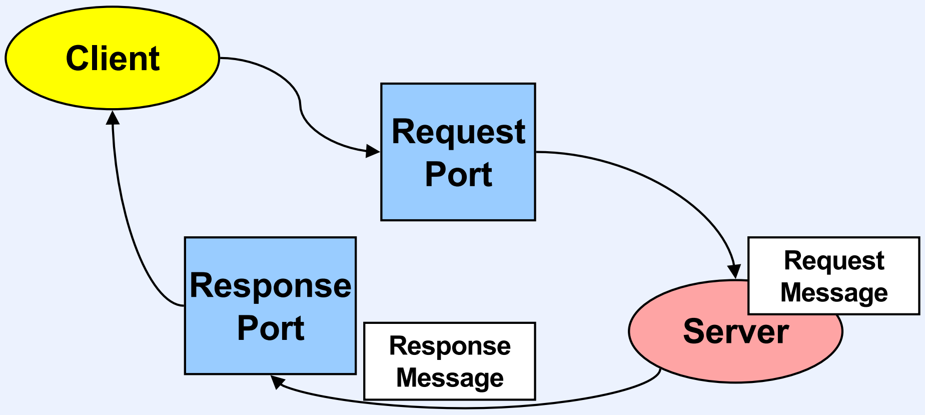


Messages, possibly containing send rights to some other port, may be sent by a holder of send rights to the port to the holder of receive rights.



## Mach Ports (3)

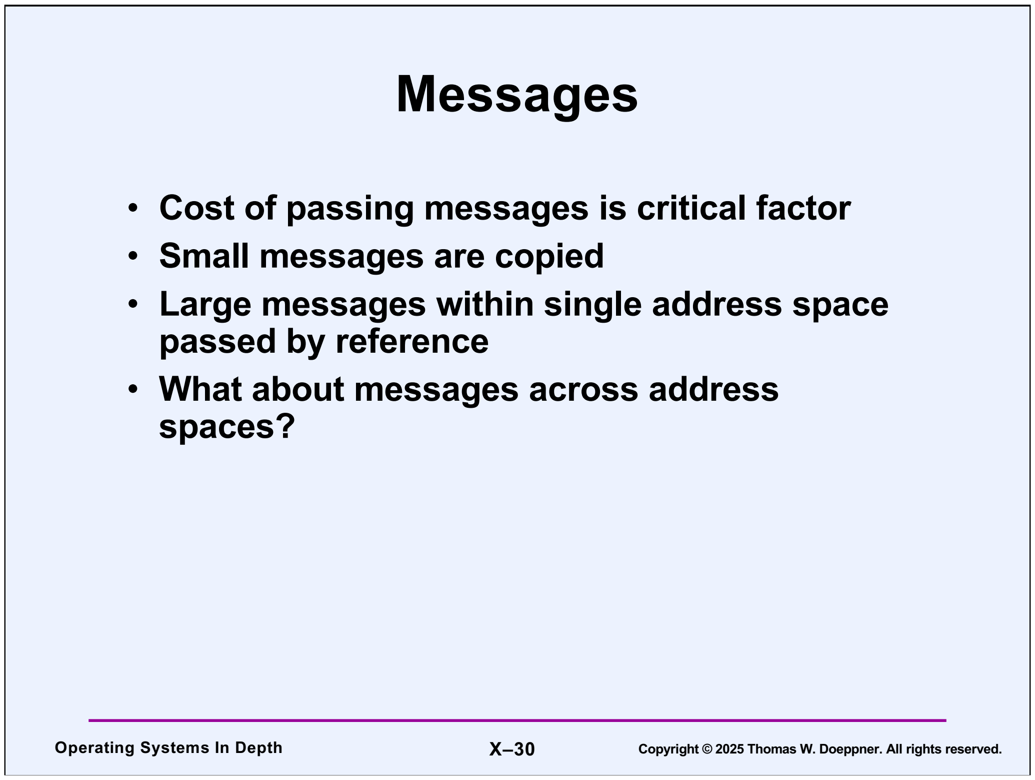
- Communication construct



One may provide a “send-once right” to a port, allowing the recipient to send just one message. This is useful for response ports.

## Method Invocation

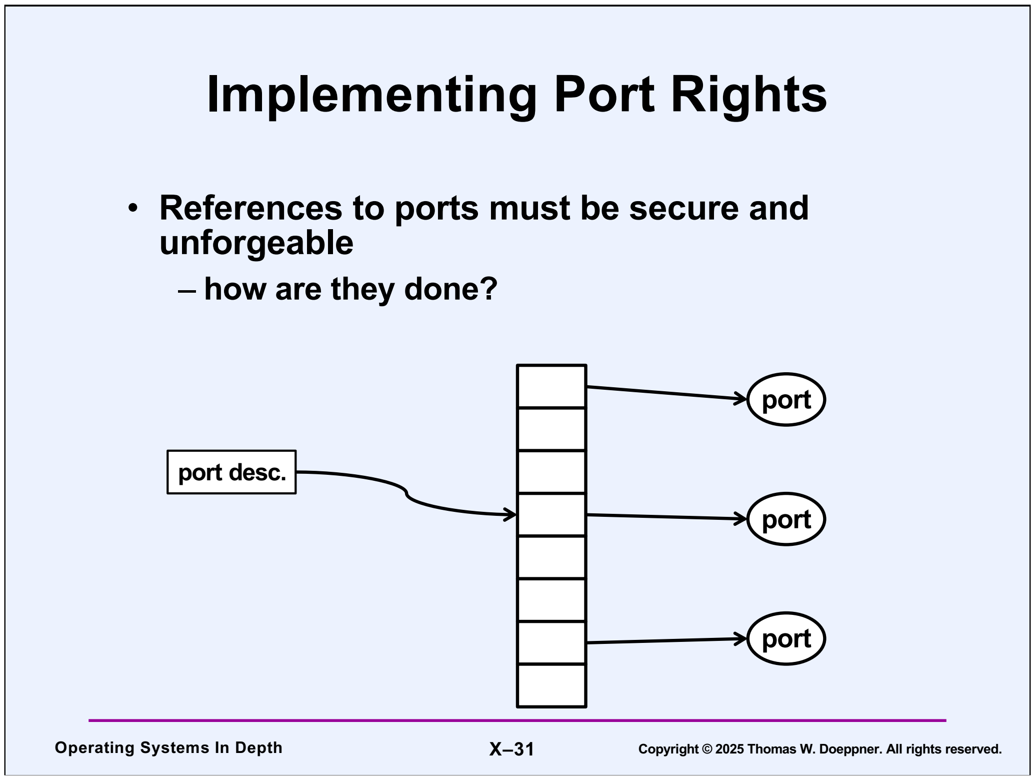
- *Tasks* implement objects
- Access rights to *ports* are secure object references
- *Messages* are method invocations and responses



## Messages

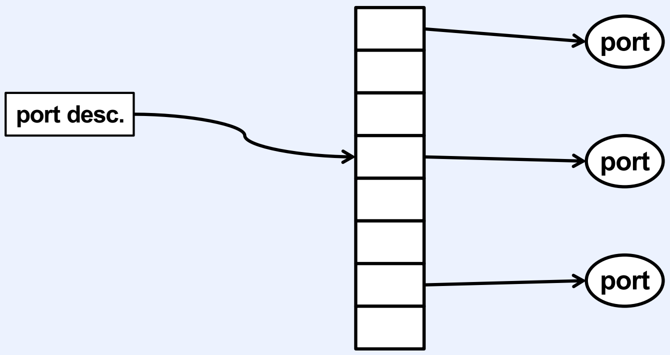
- Cost of passing messages is critical factor
- Small messages are copied
- Large messages within single address space passed by reference
- What about messages across address spaces?

We'll take up the topic of optimizing message transfer across address-space boundaries when we discuss virtual memory.

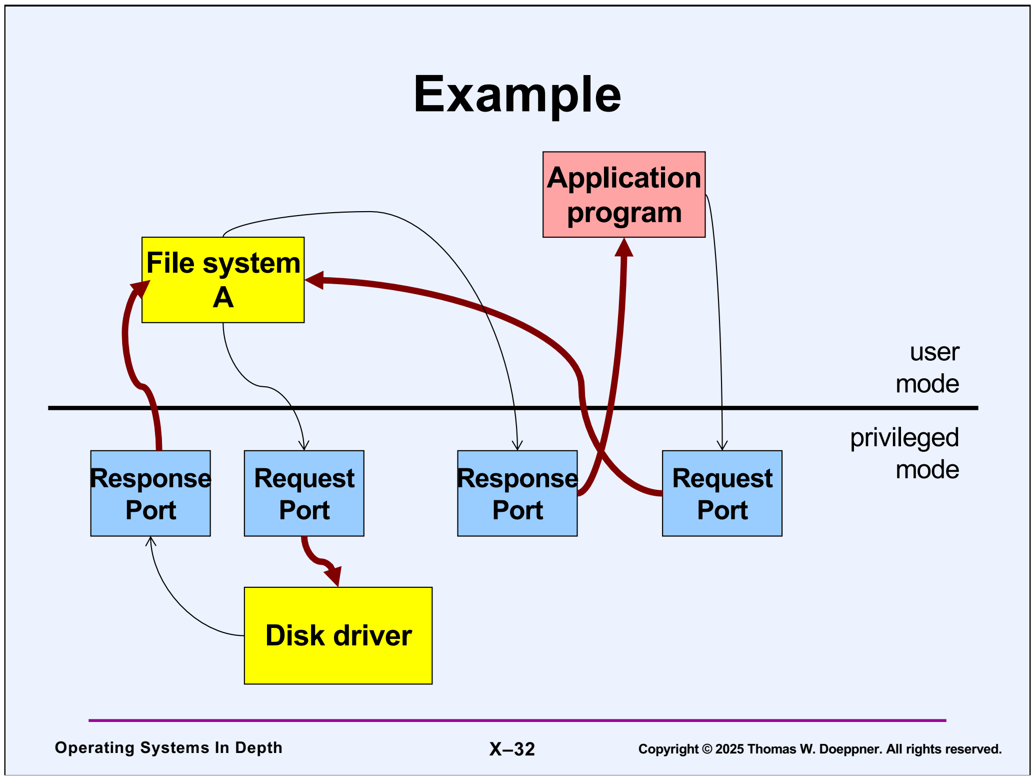


## Implementing Port Rights

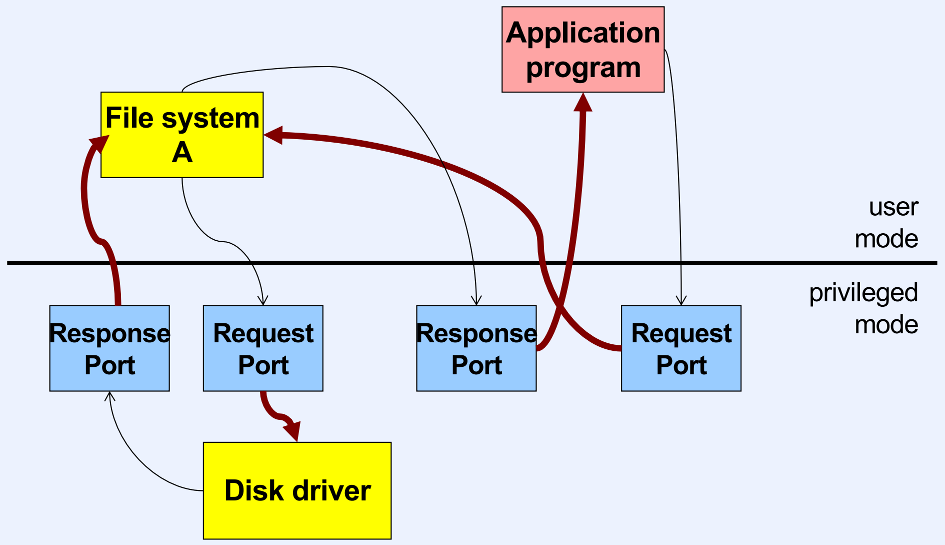
- References to ports must be secure and unforgeable
  - how are they done?



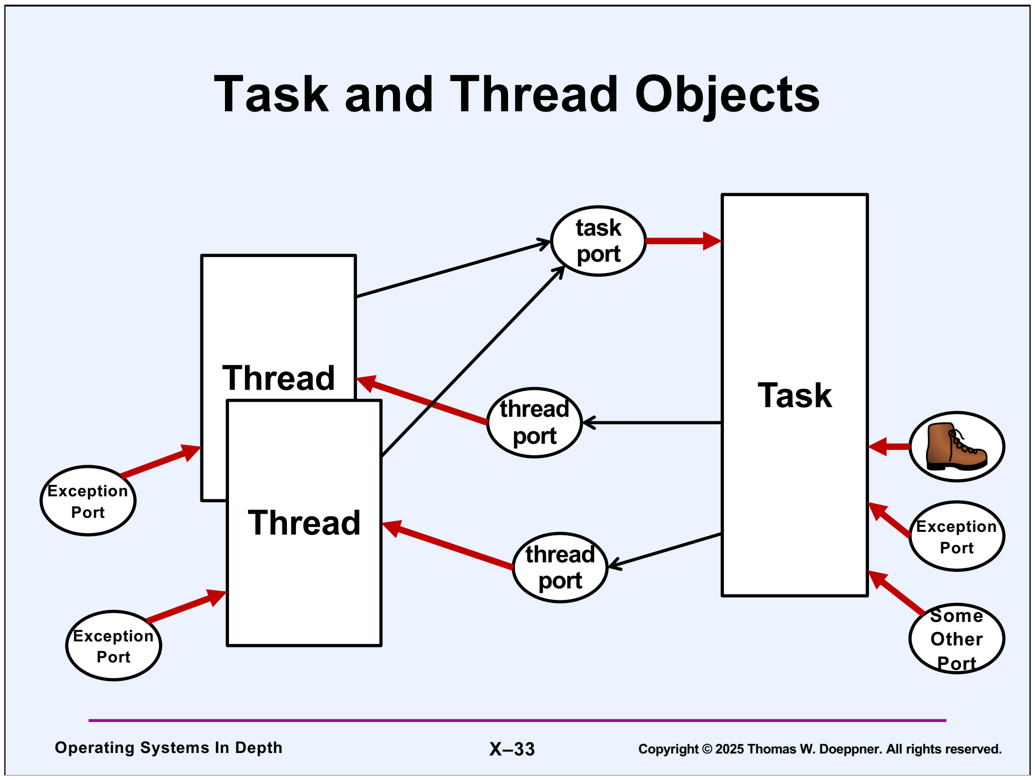
A port reference (or port descriptor) is implemented very much like a file descriptor.



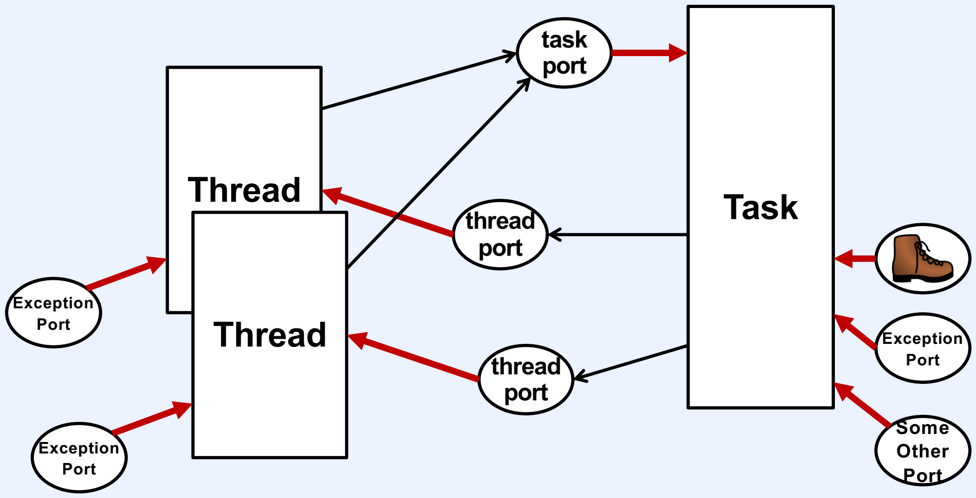
## Example



The fat lines represent receive rights, the skinny lines represent send rights.



## Task and Thread Objects

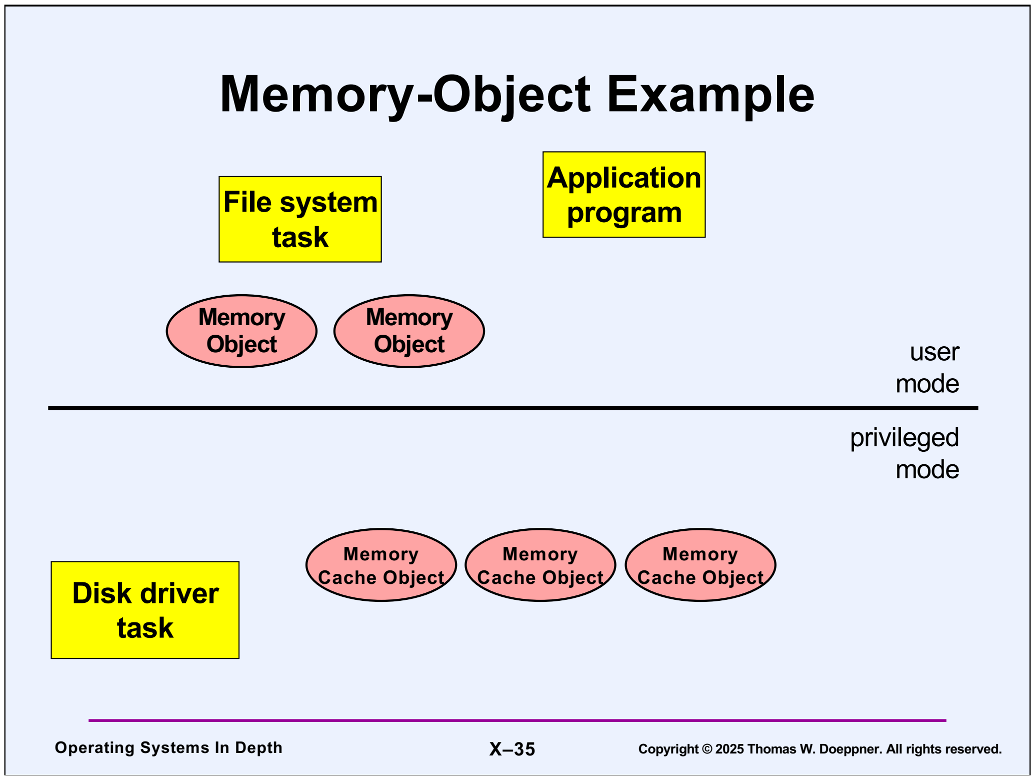


Note that all threads of a task have access to all the port rights owned by the task. Operations on threads via the thread port include suspending and resuming the thread, setting its register context, getting its register context, and terminating it. Creating a thread is an operation on the task port. Operations on task ports include creating a new task and terminating a task. A thread can set up a method to handle messages on its exception port, which are sent in response to exceptions by the kernel. If a thread hasn't set up such a port, or what it has set up doesn't handle a particular exception, the message is sent to the task's exception port.

A task's bootstrap port is used to obtain other port rights from some other object.

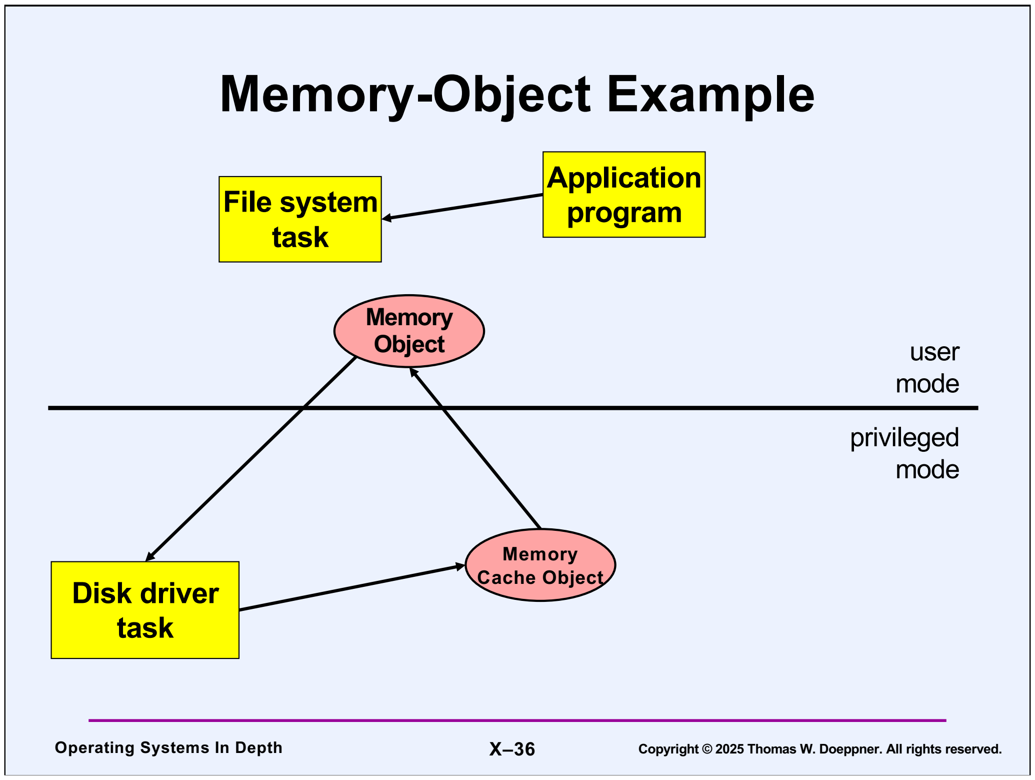
## Virtual Memory

- Memory cache objects
  - implemented in kernel
  - represent what’s in real memory
- Memory objects
  - implemented in kernel or as user tasks
  - represent what’s mapped into real memory



## Memory-Object Example

In this example, we have implemented our own file system, which is managed by a task containing a number of threads. When a file is being used, it's represented as a memory object. Such a file may be mapped into memory (using mmap). A memory cache object represents the pages of the file that are currently in real memory.



## Memory-Object Example

The application program (consisting of at least one thread running in a task) sends a message to the file system to open a file.

The program then maps the file into its address space. This is done by sending a message to a kernel task and passing it a copy of its rights to the memory object. In response, a memory cache object is created.

The program now references a page in the mapped file. The kernel sees that the reference is to a mapped object, but the page is not currently in memory. So, a thread in the kernel sends a request to the memory object for that page, and requests that a response be sent to the memory cache object.

The file system must fetch the page from disk, so it sends a request to the disk driver, which fetches the page and sends it (via a message) to the memory cache object (via the return port that was provided by the memory cache object in its request to the memory object, then passed to the disk driver).

## Quiz 3

**The application thread attempts to read from memory. The page containing the desired bytes is not present and must be fetched from disk. How many threads are involved?**

a) one (just the application thread)

b) two

c) three or more

## Devices

- Device master port exported by kernel
- Tasks holding send rights may request access to any device
  - send rights given for device port

## Successful Microkernel Systems

- 
- 
- …

## Attempts

- Windows NT 3.1
  - graphics subsystem ran as user-level process
  - moved to kernel in 4.0 for performance reasons
- Apple OS X
  - based on Mach
  - all services in kernel for performance reasons
- HURD
  - based on Mach
  - services implemented as user processes
  - no one used it, for performance reasons