

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Spring 2022 Lecture 19



Virtual Memory Virtual Machines

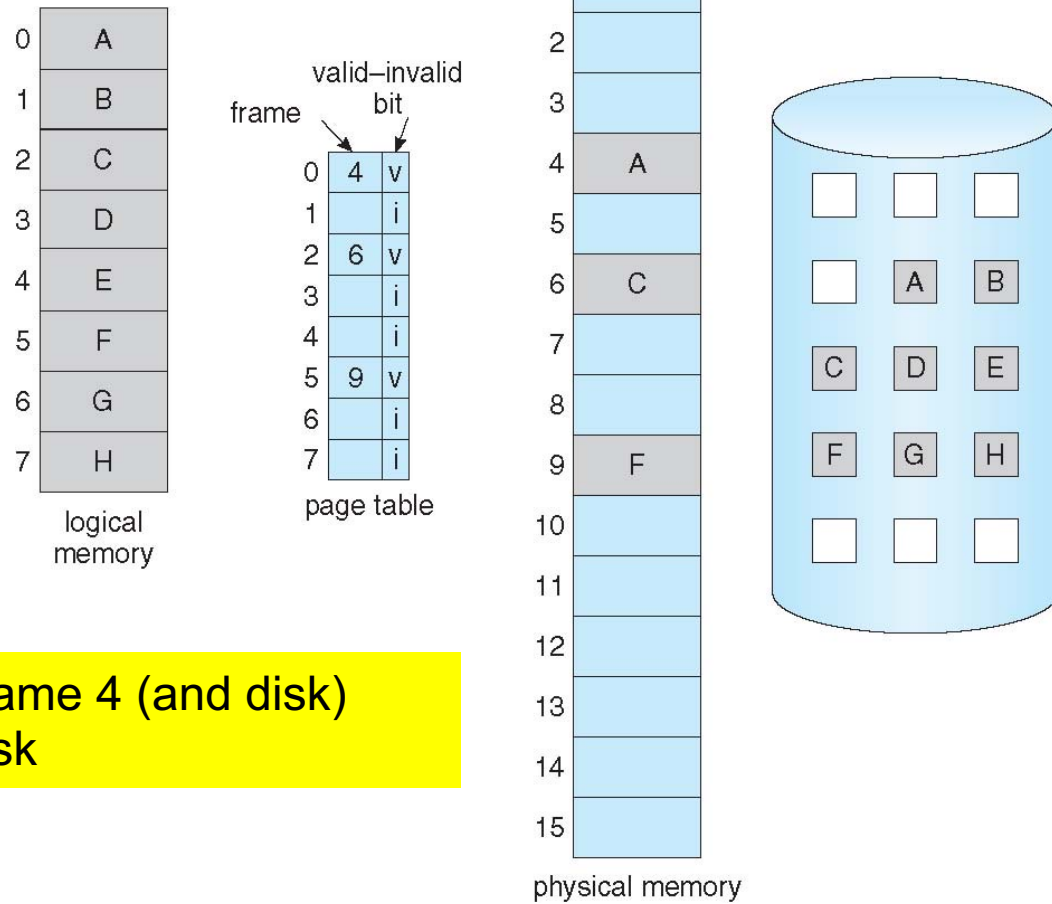
Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Demand paging: Basic Concepts

- Demand paging: pager brings in only those pages into memory what are needed
- How to determine that set of pages?
 - Need new MMU functionality to implement demand paging
- If pages needed are already **memory resident**
 - No difference from non-demand-paging
- If page needed and not memory resident
 - Need to detect and load the page into memory from storage
 - Without changing program behavior
 - Without programmer needing to change code

Page Table When Some Pages Are Not in Main Memory



Page 0 in Frame 4 (and disk)
Page 1 in Disk

Page Fault

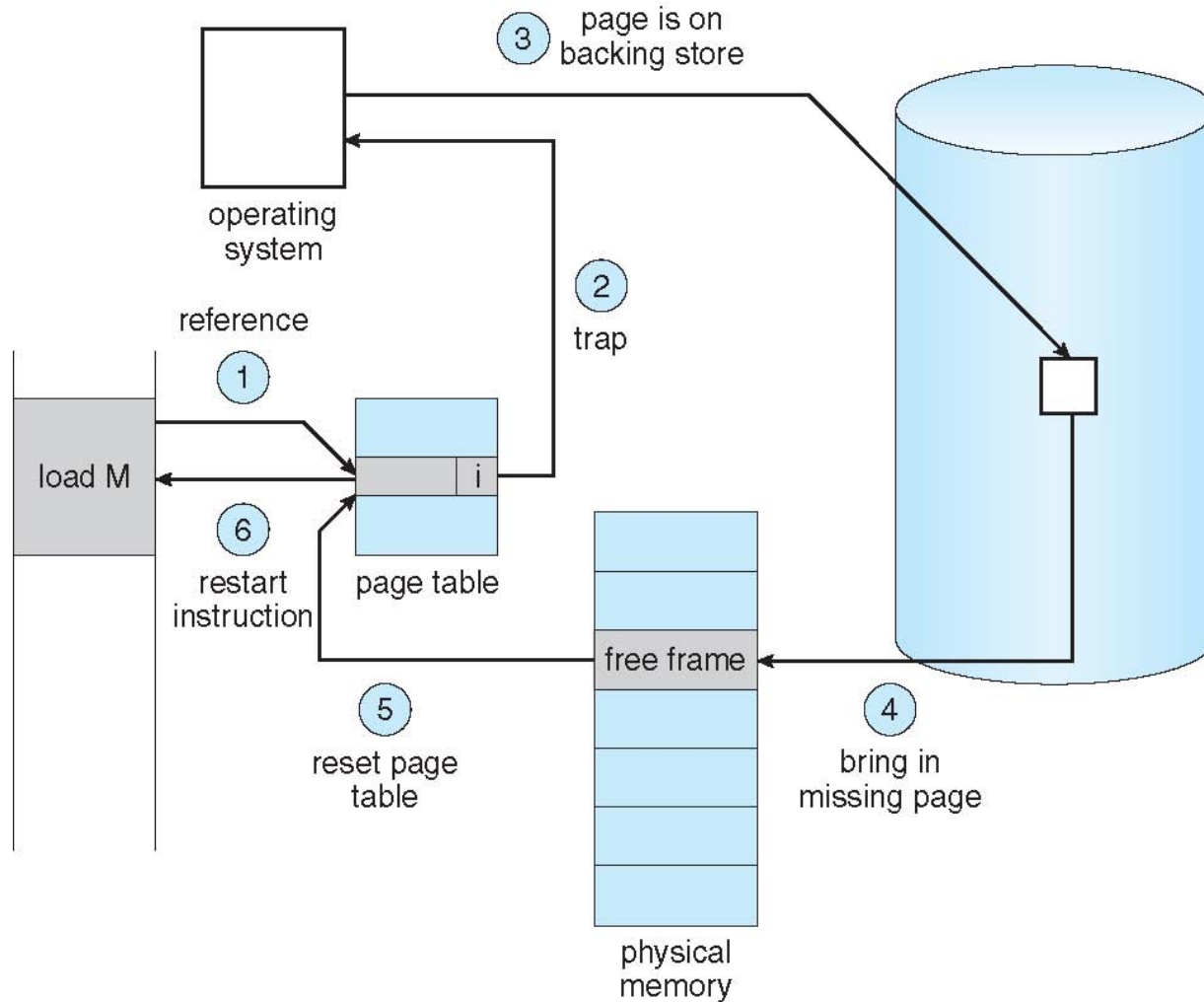
- If there is a reference to a page, first reference to that page will trap to operating system: Page fault

Page fault

- Operating system looks at a table to decide:
 - Invalid reference \Rightarrow abort
 - Just not in memory, but in *backing storage*, $\rightarrow 2$
- Find free frame
- Get page into frame via scheduled disk operation
- Reset tables to indicate page now in memory
Set validation bit = **v**
- Restart the instruction that caused the page fault

Page fault: context switch because disk access is needed

Steps in Handling a Page Fault



Performance of Demand Paging (Cont.)

- Three major activities
 - Service the interrupt – careful coding means just several hundred instructions needed
 - Read the page – relatively long time
 - Restart the process – again just a small amount of time
- Page Fault Rate $0 \leq p \leq 1$
 - if $p = 0$ no page faults
 - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

Hopefully $p \ll 1$

$$\begin{aligned} \text{EAT} = & (1 - p) \times \text{memory access time} \\ & + p (\text{page fault overhead} \\ & \quad + \text{swap page out} + \text{swap page in}) \end{aligned}$$

Page swap time = seek time + latency time

Demand Paging Simple Numerical Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- $EAT = (1 - p) \times 200 \text{ ns} + p (8 \text{ milliseconds})$
 $= (1 - p) \times 200 + p \times 8,000,000 \text{ nanosec.}$
 $= 200 + p \times 7,999,800 \text{ ns}$

Linear with page
fault rate

- If one access out of 1,000 causes a page fault, then
EAT = 8.2 microseconds.
This is a slowdown by a factor of 40!!
- If want performance degradation < 10 percent, $p = ?$
 - $220 > 200 + 7,999,800 \times p$
 $20 > 7,999,800 \times p$
 - $p < .0000025$
 - < one page fault in every 400,000 memory accesses

We make some simplifying assumptions here.

Demand paging and the limits of logical memory

- Without demand paging
 - All pages of process **must be** in physical memory
 - Logical memory **limited** to size of physical memory
- With demand paging
 - All pages of process **need not be** in physical memory
 - Size of logical address space is **no longer constrained** by physical memory
- Example
 - 40 pages of physical memory
 - 6 processes each of which is 10 pages in size
 - But each process only needs 5 pages *as of now*
 - Run 6 processes with 10 pages to spare



Higher degree of multiprogramming

Coping with over-allocation of memory

Example

- Physical memory = 40 pages
- 6 processes each of which is of size 10 pages
 - But are using 5 pages each as of now
- What happens if each process needs all 10 pages?
 - 60 physical frames needed
- **Option: Terminate** a user process
 - But paging should be transparent to the user
- **Option: Swap out** a process
 - Reduces the degree of multiprogramming
- **Option: Page replacement:** selected pages.
Policy?



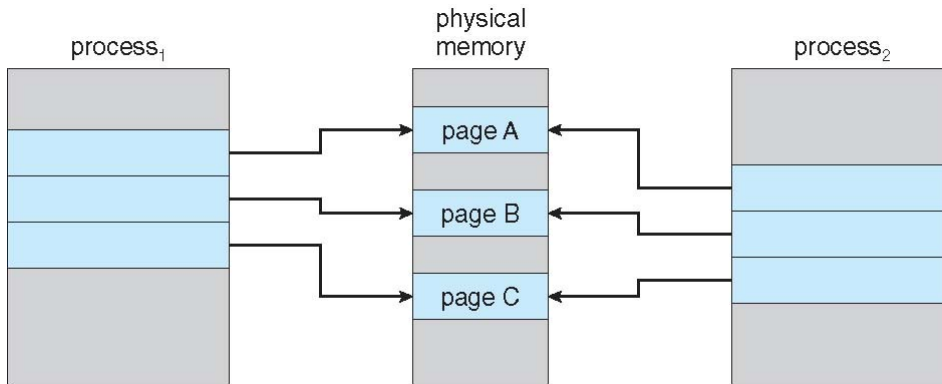
Solving the Fork mystery_(Copy-on-Write)

- **Copy-on-Write** (COW) allows both parent and child processes to initially *share* the same pages in memory
 - If either process modifies a shared page, only then is page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a **pool** of **zero-fill on-demand** pages
 - Pool should always have free frames for fast demand page execution
 - Don't want to have to free a frame as well as other processing on page fault
 - Why zero-out a page before allocating it? (**security**)

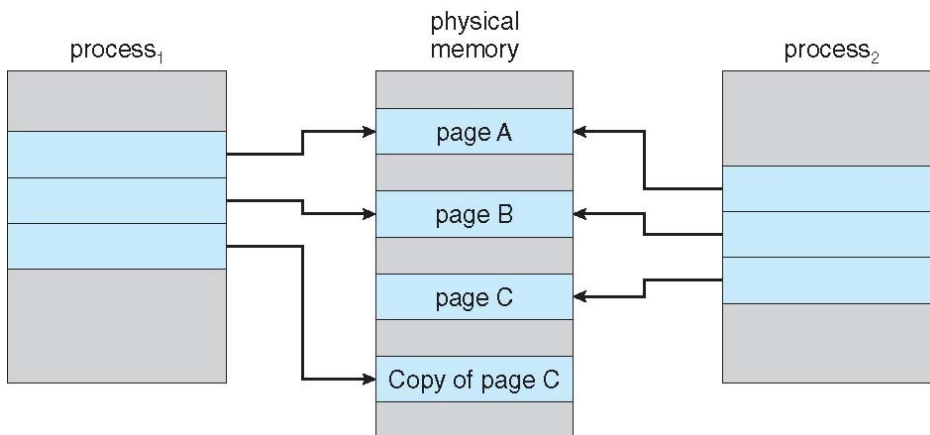
For
security

Copy-on-write

Before Process 1 Modifies Page C



After Process 1 Modifies Page C



What Happens if there is no Free Frame?

- Could be all used up by process pages or kernel, I/O buffers, etc
 - How much to allocate to each?
- Page replacement – find some page in memory, but not really in use, page it out
 - Algorithm – terminate? swap out? replace the page?
 - Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Continued to Page replacement etc...

Page Replacement

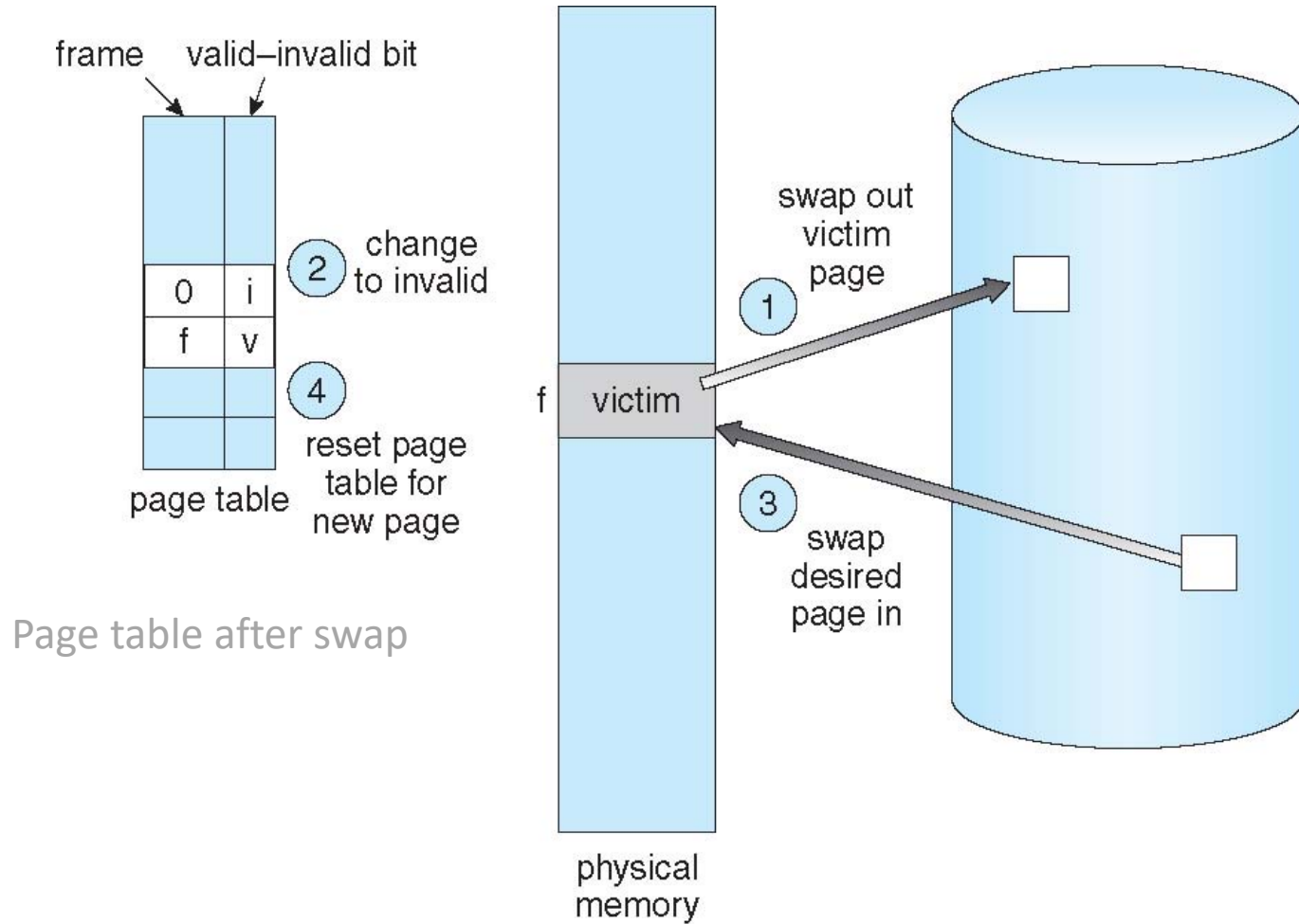
- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk

Basic Page Replacement

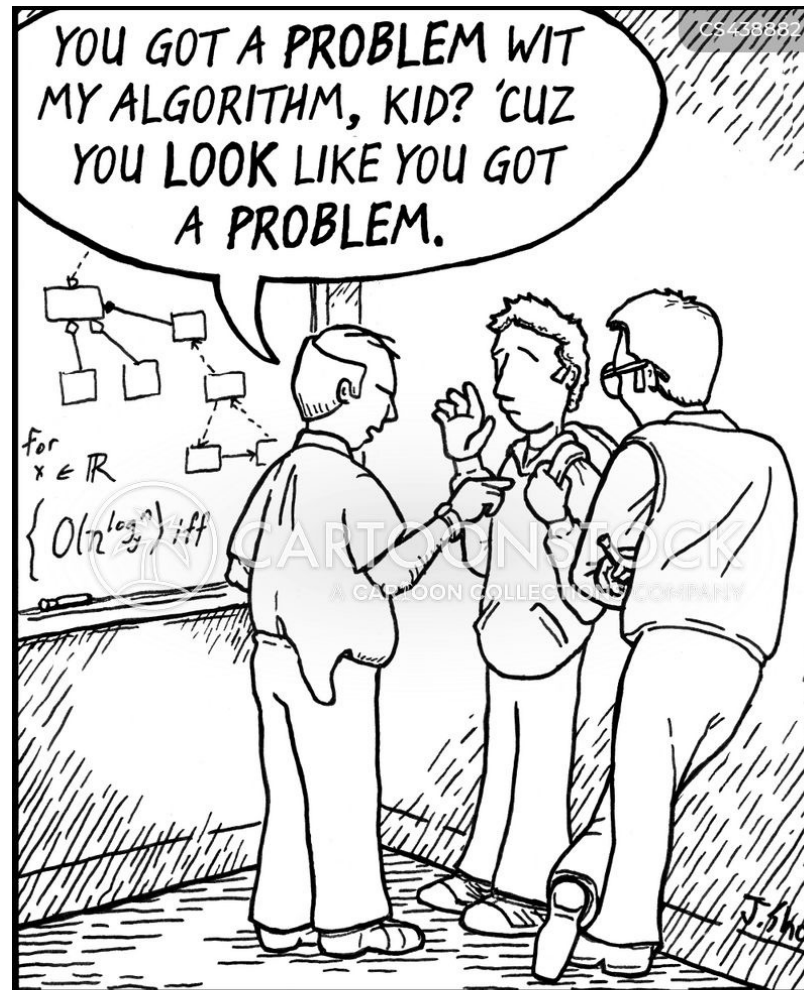
1. Find the location of the desired page on disk
2. Find a free frame:
 - I. If there is a free frame, use it
 - II. If there is no free frame, use a page replacement algorithm to select a **victim frame**
 - III. Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement



More algorithms ...



Jim unwittingly wanders into a rough section of the Computer Science department.

Virtual Machines

- We need to do a context switch here.
- The next assignment involves containers. WE need to start this now to allow us to focus on the project in the later part of the semester.
- Let us look at Virtual machines and Containers next, and we'll restore the context and come back to Page replacement algorithms.

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2021



Virtualization & Containerization

Slides based on

- Various sources

Virtualization

Ch 18 + external

- Why we need virtualization?
- The concepts and terms
- Brief history of virtualization
- Types of virtualization
- Implementation Issues
- Containers



We will skip implementation specific details. Please consult the documentation and watch related videos.

Isolation and resource allocation

Isolation levels:

- **Process:** Isolated address space
- **Container:** Isolated set of processes, files and network
- **Virtual Machines (VM):** Isolated OSs
- **Physically isolated** machines

Resource allocation:

- Resources need to be allocated to processes/containers/VMs and
- managed to serve needs best.

Virtualization

- A Virtual scheme provides a simpler perspective of a Physical scheme. Needs mapping.
 - Example: each process a separate virtual address space.
 - OS allocates physical memory and disk space and handles mapping.
- System (“machine”) virtualization
 - A machine needs its own CPU, memory, storage, I/O to run its OS and apps. “Machine” = {CPU, memory, storage, I/O, OS, apps}
 - Needs to be isolated from other machines.
 - “Virtual machines” allocated resources from physical hardware, with allocation done by a Virtual Machine Monitor (VMM or hypervisor).
 - A virtual machine can be “migrated” from one physical system to another.

Virtualization



"Tell that intern that you can't migrate physical machines."

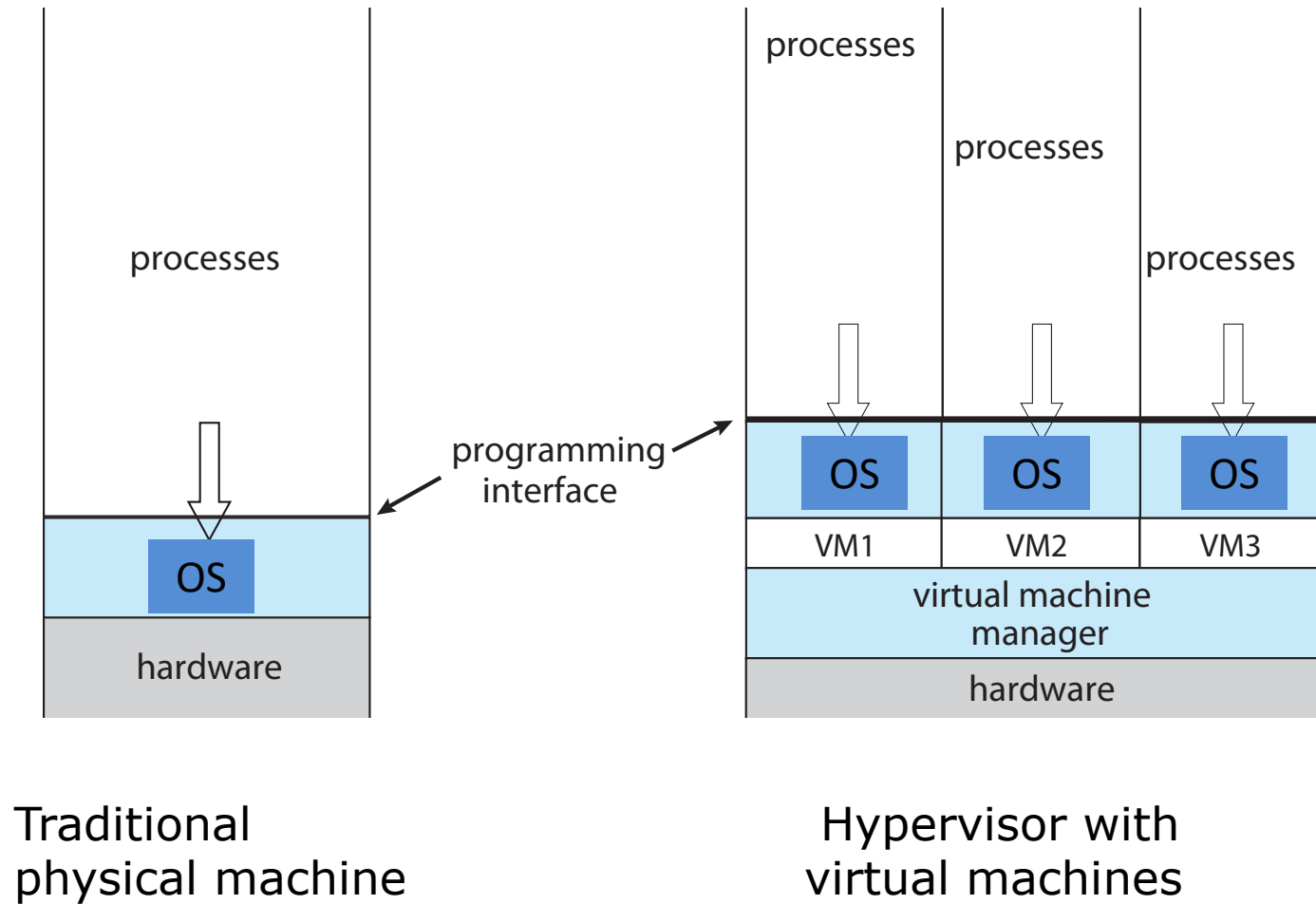
Virtualization

- Processors have gradually become very powerful
- Dedicated servers can be very underutilized (5-15%)
- Virtualization allow a single server to support several virtualized servers: typical [consolidation ratio](#) 6:1
- Power cost a major expense for data centers
 - Companies frequently locate their data centers in the middle of nowhere where power cost is low
- If a hardware server crashes, would be nice to migrate the load to another one.
- A key component of cloud computing

Virtual Machines (VM)

- **Virtualization** technology enables a single PC/server to simultaneously run multiple Virtual Machines,
 - with different operating systems or multiple sessions of a single OS.
- A machine with virtualization can host many applications, including those that run on different operating systems, on a single platform.
- The host operating system can support a number of virtual machines, each of which has the characteristics of a particular OS.
- The software that enables virtualization is a **virtual machine monitor (VMM)**, or **hypervisor**.

Virtual Machines (VM)



Kinds of Virtual Systems

Virtualization

- Hypervisor based
 - Full virtualization: bare metal hypervisor
 - Para virtualization: modified guest OS
 - Host OS virtualization
- Container system: multiple user space instances
- Environment virtualization
 - Java virtual machine, Dalvic virtual machine
- Software simulation of hardware/ISA
 - Android JDK
 - SoftPC etc.
- Emulation using microcode

Brief history

- Early 1960s IBM experimented with two independently developed hypervisors - SIMMON and CP-40
- Common CPU **modes**: **user** and **supervisor** (*Privileged*)
- In 1974, Popek and Goldberg published a paper which listed what conditions a computer architecture should satisfy to support virtualization efficiently
 - *Privileged instructions: Those that trap if the processor is in user mode and do not trap if it is in system mode (supervisor mode).*
 - *Sensitive instructions: that attempt to change the configuration of resources in the system or whose behavior or result depends on the configuration of resources*
 - Theorem. For any conventional third-generation computer, an effective VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.
 - The x86 architecture that originated in the 1970s did not meet these for requirements for decades.

•

“Strictly Virtualizable”

A processor or mode of a processor is *strictly virtualizable* if, when executed in a lesser privileged mode:

- all instructions that access privileged state trap
- all instructions either trap or execute identically

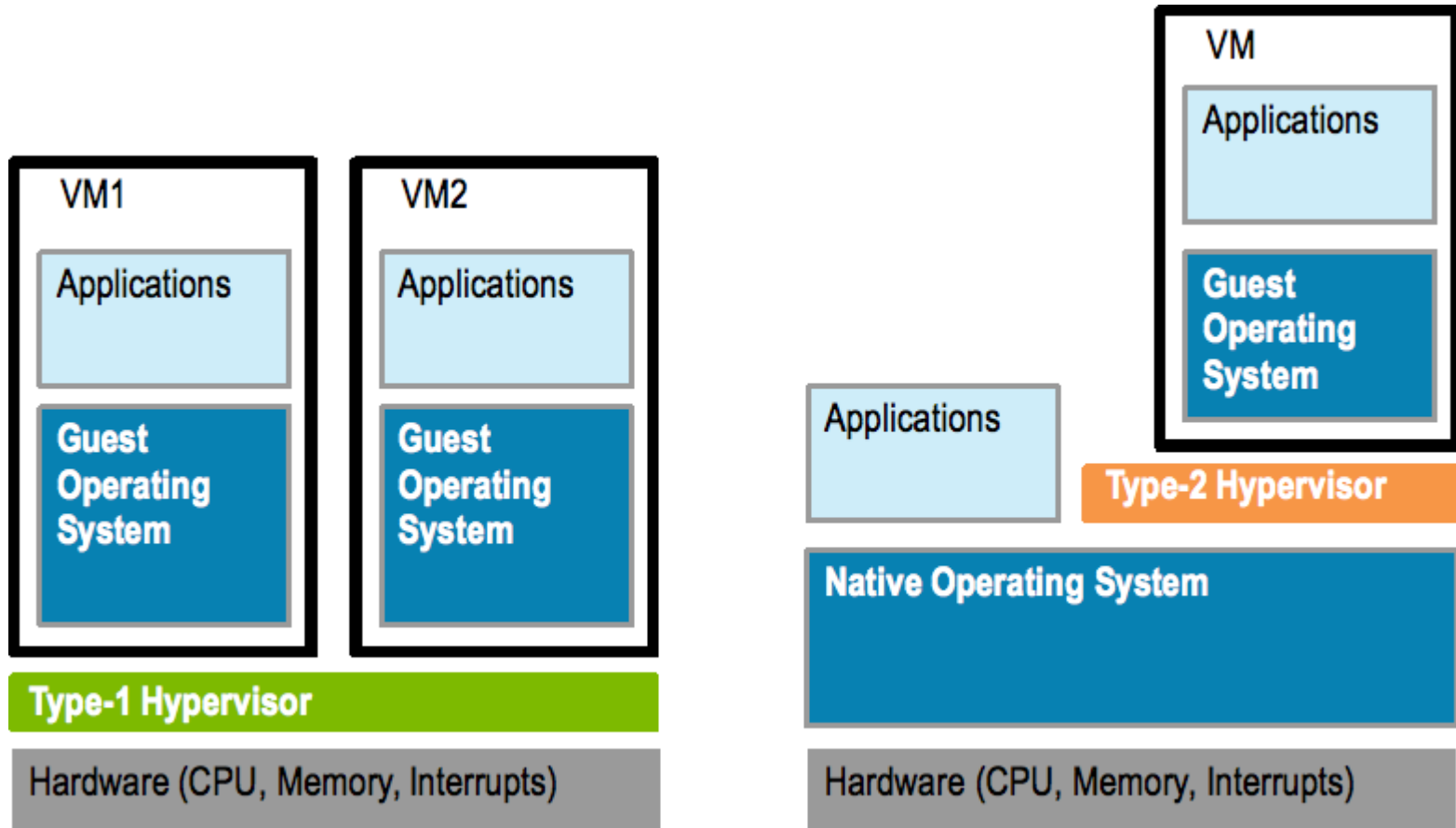
Brief history (recent)

- Stanford researchers developed a new hypervisor and then founded VMware
 - first virtualization solution for x86 in 1999
 - Linux, windows
- Others followed
 - Xen, 2003 University of Cambridge, Xen Project community
 - KVM, 2007 startup/Red Hat
 - VirtualBox (Innotek GmbH/Sun/Oracle) , 2007
 - Hyper-V (Microsoft), 2008
- Cgroups (2007 Google), Docker Engine 2013

Implementation of VMMs

- **Type 1 hypervisors** - Operating-system-like software built to provide virtualization. Runs on ‘bare metal’.
 - Including VMware ESX, Joyent SmartOS, and Citrix XenServer
- Also includes general-purpose operating systems that provide standard functions as well as VMM functions
 - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
 - Including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

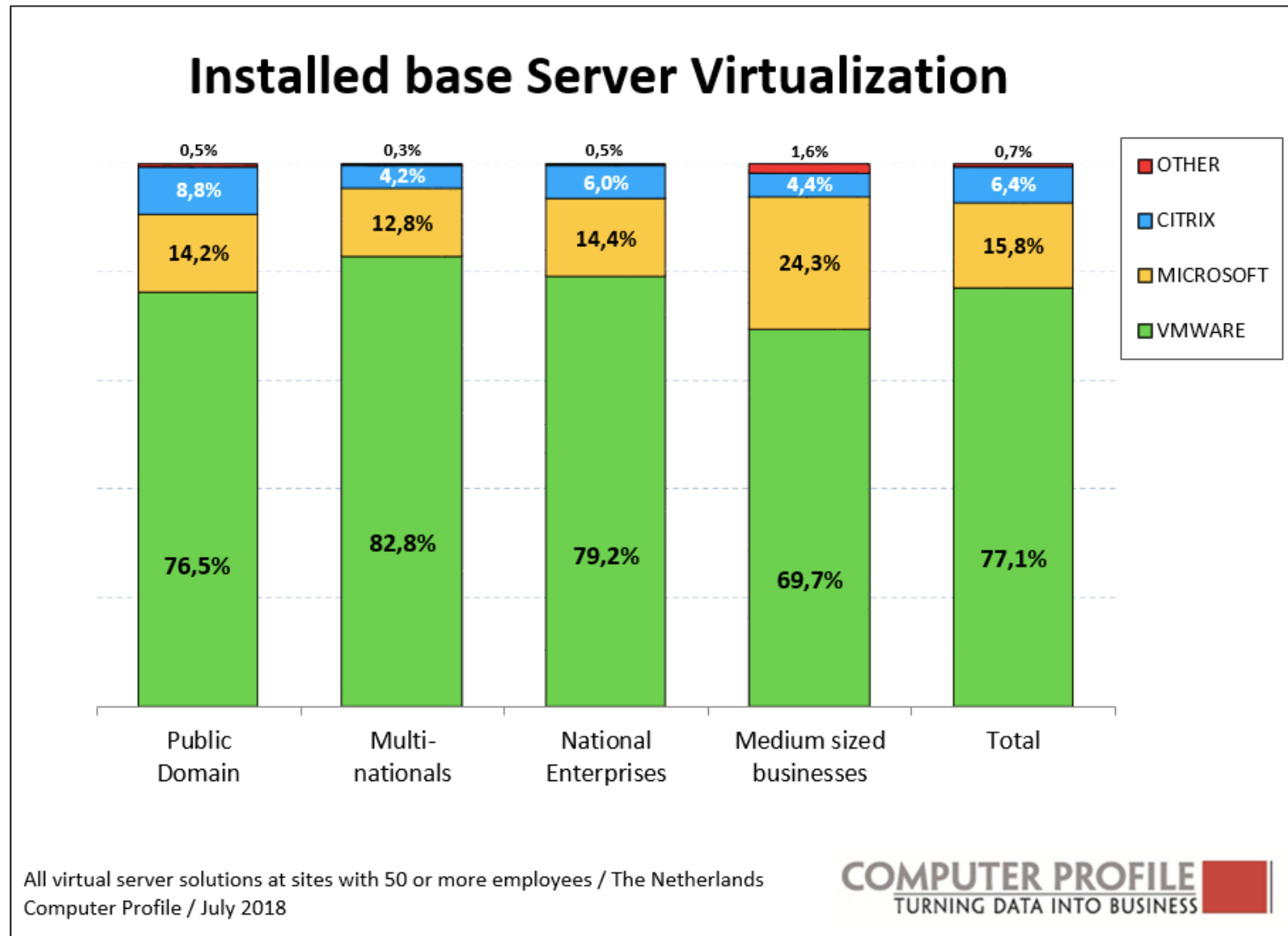
Implementation of VMMs



A higher layer uses services of the lower layers.

<https://microkernelndude.files.wordpress.com/2012/01/type1-vs-2.png>

Market share



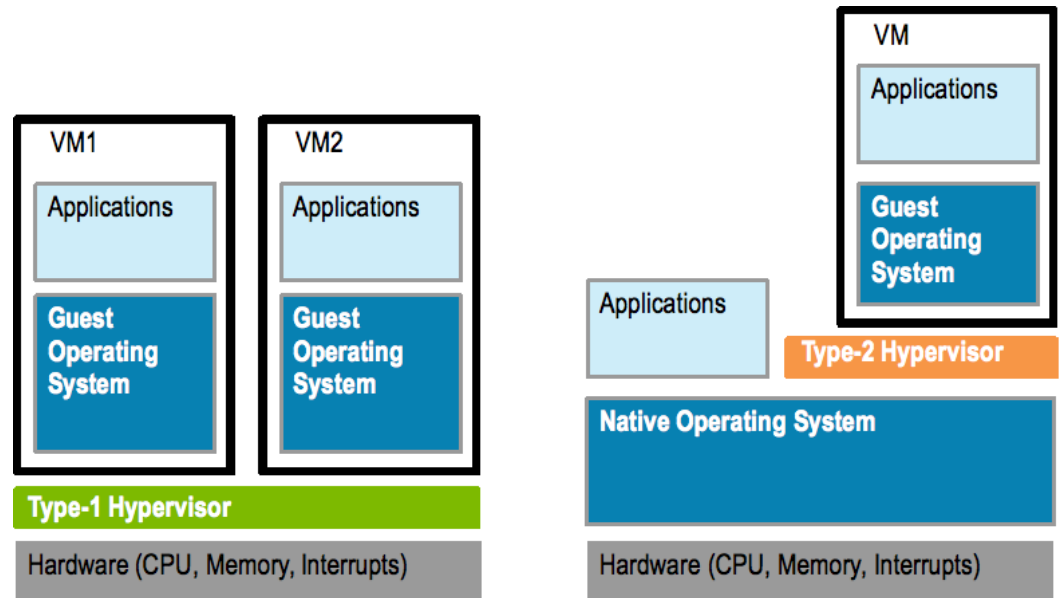
User mode and Kernel (supervisor) mode

- Special instructions:
- Depending on whether it is executed in kernel/user mode
 - “Sensitive instructions”
- Some instructions cause a trap when executed in user-mode
 - “Privileged instructions”
- A machine is virtualizable only if sensitive instructions are a subset of privileged instructions
 - Intel’s 386 did not always do that. Several sensitive 386 instructions were ignored if executed in user mode.
- Fixed in 2005 virtualization may need to be enabled using BIOS
 - Intel CPUs: VT (Virtualization Technology)
 - AMD CPUs: SVM (Secure Virtual Machine)

Virtualization support

- Terminology:
 - Guest Operating System
 - The OS running on top of the hypervisor
 - Host Operating System
 - For a type 2 hypervisor: the OS that runs on the hardware "executions"
- Create environments in which VMs can be run
- When a guest OS is started in an environment, continues to run until it causes an exception and traps to the hypervisor
 - For e.g., by executing an I/O instruction
- Set of operations that trap is controlled by a hardware bit map set by hypervisor
 - trap-and-emulate approach becomes possible

Implementation of VMMs



What problems do you see?

- What mode does hypervisor run in? Guest OSs?
- Are Guest OSs aware of hypervisor?
- How is memory managed?
- How do we know what is the best choice?

Virtual Machine (VM) as a software construct

- Each VM is configured with some number of processors, some amount of RAM, storage resources, and connectivity through the network ports.
- Once the VM is created it can be activated on like a physical server, loaded with an operating system and software solutions, and used just like a physical server.
- Unlike a physical server, VM only sees the resources it has been configured with, not all of the resources of the physical host itself.
- The hypervisor facilitates the translation and I/O between the virtual machine and the physical server.

Virtual Machine (VM) as a set of files

- Configuration file describes the attributes of the virtual machine containing
 - server definition,
 - how many virtual processors (vCPUs)
 - how much RAM is allocated,
 - which I/O devices the VM has access to,
 - how many network interface cards (NICs) are in the virtual server
 - the storage that the VM can access
- When a virtual machine is instantiated, additional files are created for logging, for memory paging etc.
- Copying a VM produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself

Virtualization benefits

- Run multiple, OSes on a single machine
 - **Consolidation**, app dev, ...
- Security: Host system protected from VMs, VMs protected from each other
 - Sharing though shared file system volume, network communication
- Freeze, suspend, running VM
 - Then can move or copy somewhere else and **resume**
 - **Live migration**
 - Snapshot of a given state, able to restore back to that state
 - **Clone** by creating copy and running both original and copy
- Hence – cloud computing

Building Block – Trap and Emulate

- VM needs two modes: both in real user mode
 - virtual user mode and virtual kernel mode
- When Guest OS attempts to execute a privileged instruction, what happens?
 - Causes a trap
 - VMM gains control, analyzes error, executes operation as attempted by guest
 - Returns control to guest in user mode
 - Known as **trap-and-emulate**
- Trap-and-emulate was the technique used for implementing floating point instructions in CPUs without floating point coprocessor

Handling sensitive instructions

- Some CPUs didn't have clean separation between privileged and non-privileged instructions
 - Sensitive instructions
 - Consider Intel x86 `popf` instruction
 - If CPU in privileged mode -> all flags replaced
 - If CPU in user mode -> on some flags replaced
 - No trap is generated
- Binary translation (complex) solves the problem
 1. If guest VCPU is in user mode, guest can run instructions natively
 2. If guest VCPU in kernel mode (guest believes it is in kernel mode)
 1. VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
 2. Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)
 3. Cached translations can reduce overhead
- Not needed in newer processors with virtualization support.

Type 1 Hypervisors

- Run on top of *bare metal*
- Guest OSs believe they are running on bare metal, are unaware of hypervisor
 - are not modified
 - Better performance
- Choice for data centers
 - Consolidation of multiple OSes and apps onto less HW
 - Move guests between systems to balance performance
 - Snapshots and cloning
- Hypervisor creates runs and manages guest OSes
 - Run in kernel mode
 - Implement device drivers
 - provide traditional OS services like CPU and memory management
- Examples: VMWare esx (dedicated) , Windows with Hyper-V (includes OS)

Type 2 Hypervisors

- Run on top of host OS
- VMM is simply a process, managed by host OS
 - host doesn't know they are a VMM running guests
- poorer overall performance because can't take advantage of some HW features
- Host OS is just a regular one
 - Individuals could have Type 2 hypervisor (e.g. Virtualbox) on native host (perhaps windows), run one or more guests (perhaps Linux, MacOS)

Full vs Para-virtualization

- Full virtualization: Guest OS is unaware of the hypervisor. It thinks it is running on bare metal.
- Para-virtualization: Guest OS is modified and optimized. It sees underlying hypervisor.
 - Introduced and developed by Xen
 - Modifications needed: Linux 1.36%, XP: 0.04% of code base
 - Does not need as much hardware support
 - allowed virtualization of older x86 CPUs without binary translation
 - Not used by Xen on newer processors

CPU Scheduling

- One or more virtual CPUs (vCPUs) per guest
 - Can be adjusted throughout life of VM
- When enough CPUs for all guests
 - VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs (CPU overcommitment)
 - VMM can use scheduling algorithms to allocate vCPUs
 - Some add fairness aspect

CPU Scheduling (cont)

- Oversubscription of CPUs means guests may get CPU cycles they expect
 - Time-of-day clocks may be incorrect
 - Some VMMs provide application to run in each guest to fix time-of-day