ESOF 422:

Advanced Software Engineering: Cyber Practices

Operating System Internals, Windows

Reese Pearsall

Spring 2025

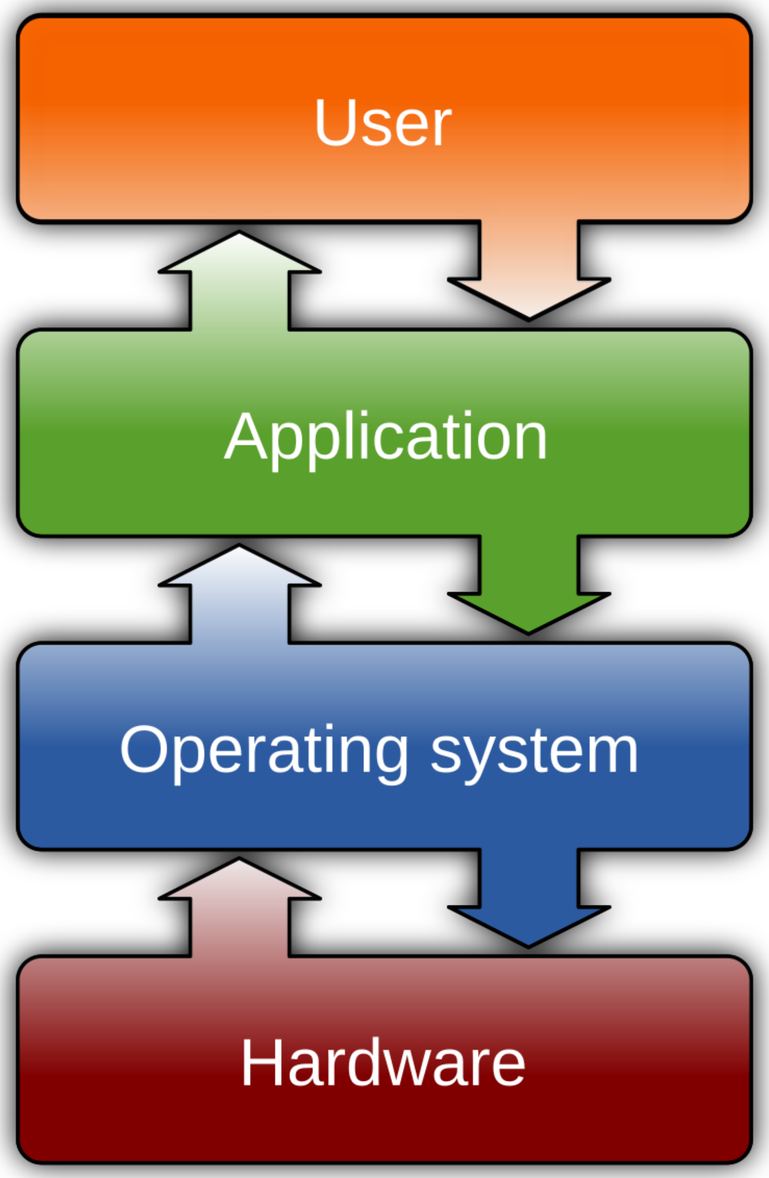
Operating Systems

- In the physical world, the environment of where the crime was committed is a crucial element for how they investigate

- The operating system is a piece of software that manages a computer's hardware and its resources.
  - ❑ acts as a middleman between a user and the computer's resources
  - ❑ any malicious action typically has to go through the OS
  - ❑ vital to understand how the OS works and structures things when responding to an incident

- The OS has artifacts that may be helpful for our investigations!

We will be working with Windows, but many of these same principles apply for Linux and MacOS



OS Permissions

- Code execution happens within different privilege levels (“protection rings”)

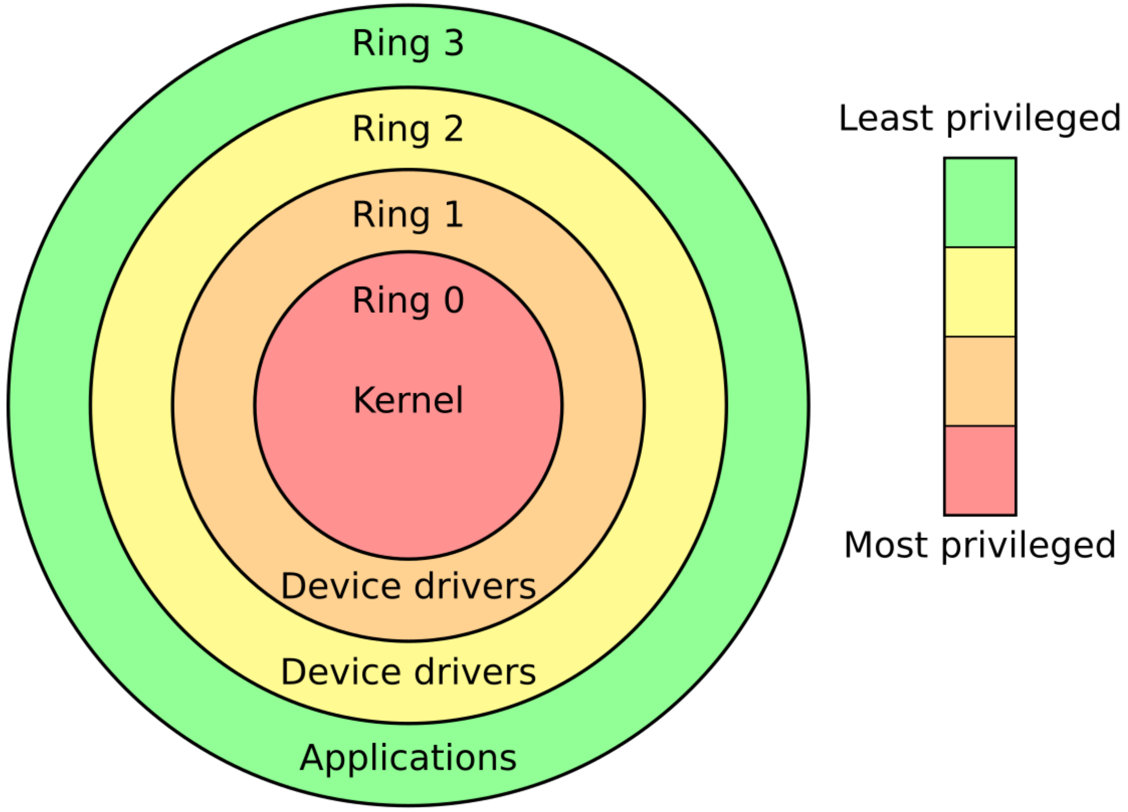
- Depending on the ring, the code may or may not be allowed access to computers resources and hardware

Ring 0: full direct access to hardware and resources
→ Typically, only reserved for kernel code

Ring 1 and 2: some access to hardware, but still within a controlled environment

Ring 3: no access to hardware
→ This is where user code, and most software runs

The **kernel** is the heart of the OS. It provides all the functionality and services to manage the computer’s resources and hardware



OS Permissions

Common Windows accounts and their permissions:

- `SYSTEM` (or LocalSystem)

→ Unlimited privileges, runs in kernel mode (ring 0)

- `TrustedInstaller`

→ Special account used for installer and update services
→ Ring 3 (but triggers many ring 0 services)

- `Administrator`

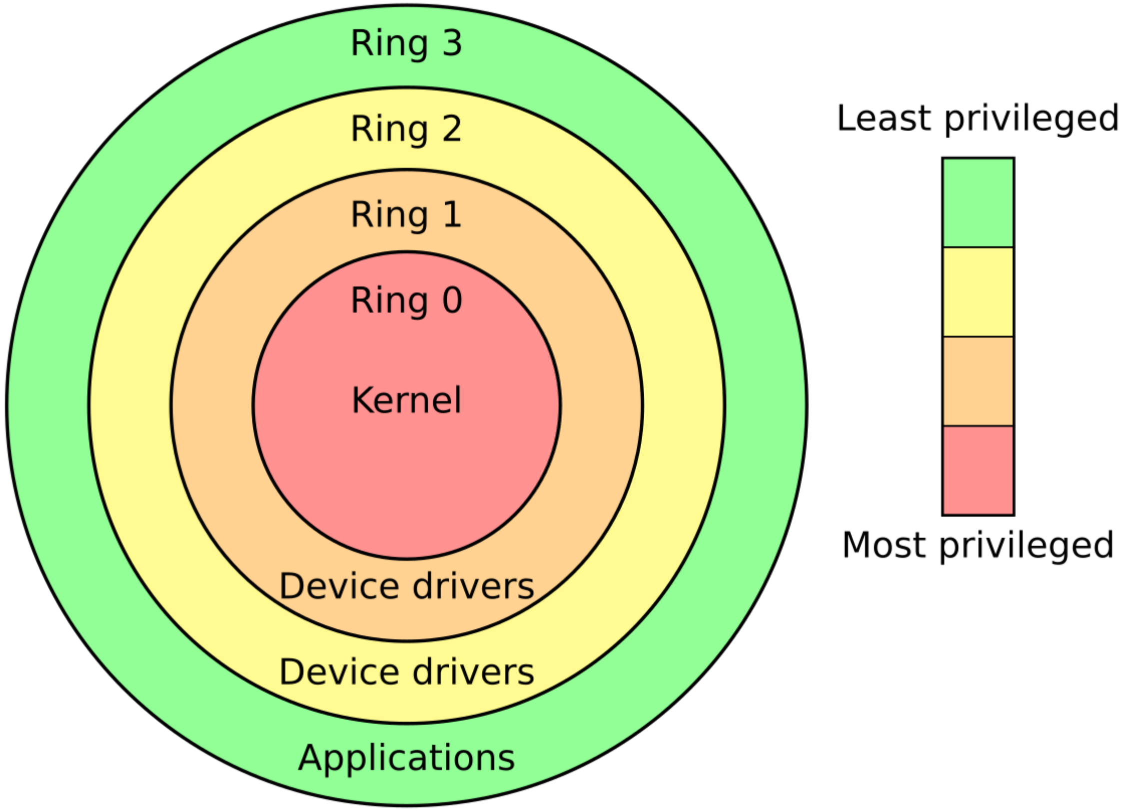
→ High-level control with admin rights
→ Ring 3 (but comes with ability to escalate to ring 0)

- Normal user

→ Standard user with limited access
→ Cannot make system-wide changes
→ Ring 3

- `NetworkService`

→ Low-level account used for basic networking services
→ Ring 3



If a user’s program wants to use and modify the file on the computer, how does it do it?

If a user’s program wants to use and modify the file on the computer, how does it do it? System call!

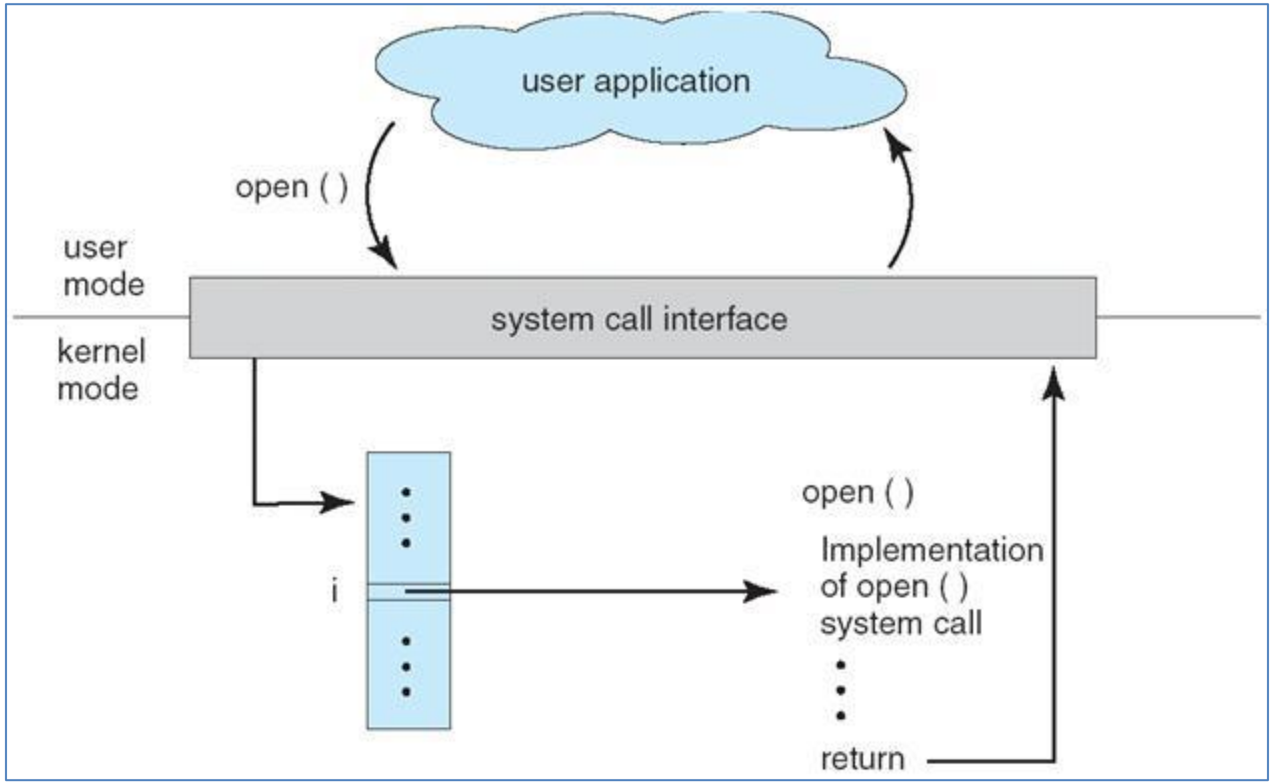
A **system call** is a predefined, programmatic way to request a service from the OS kernel

A system call must be used if you want to
- Interact with a file
- Communicate on a network
- Spawn a new process

An interrupt is used to switch from user mode (ring 3) to kernel mode (ring 0)

System calls in Windows are scattered across different files, but `kernel32.dll` and `ntdll.dll` are more commonly used

The standard libraries we use in code typically handle the system calls for us

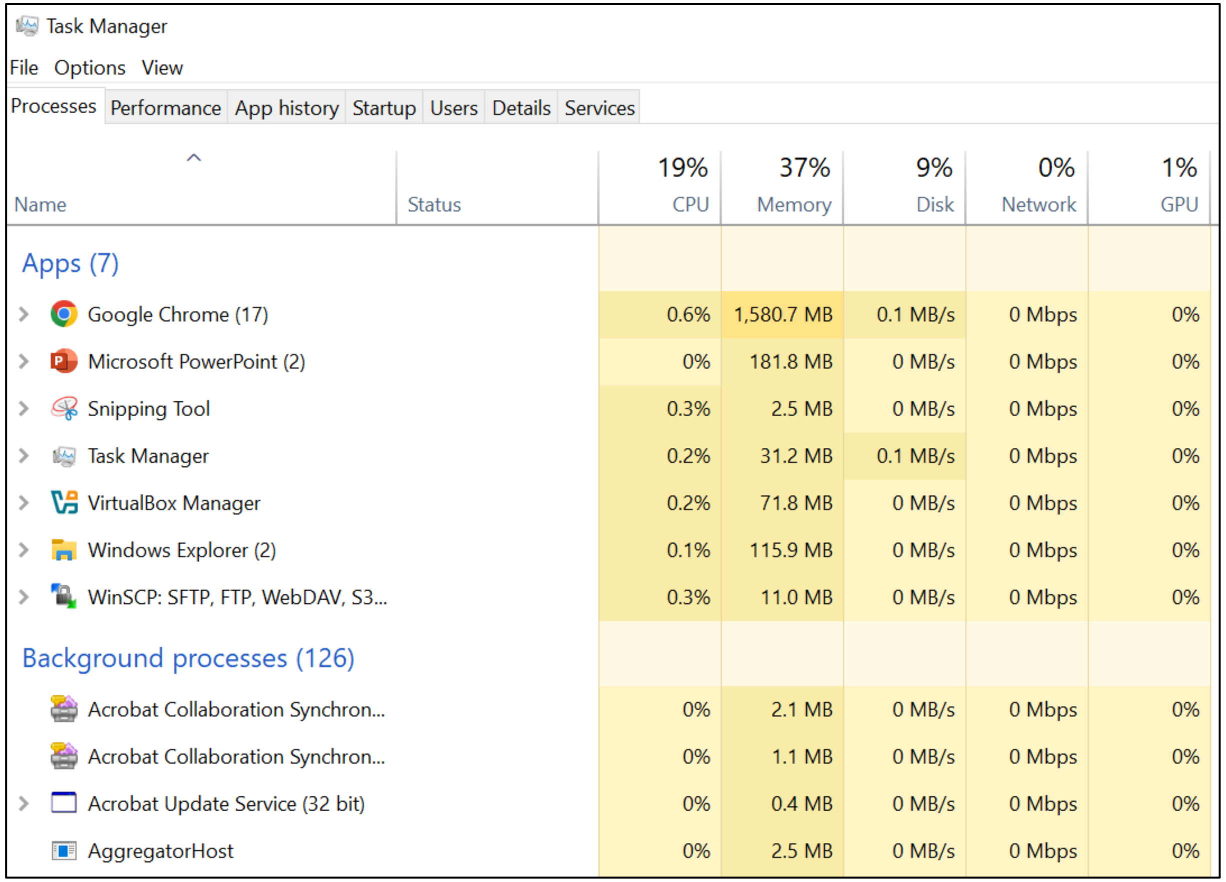


A **process** is a running program in memory. The OS is responsible for creating, managing, and terminating processes

All code and data for a process exist in memory within a *process address space*

If malware is running, it should be an active process that the OS is maintaining

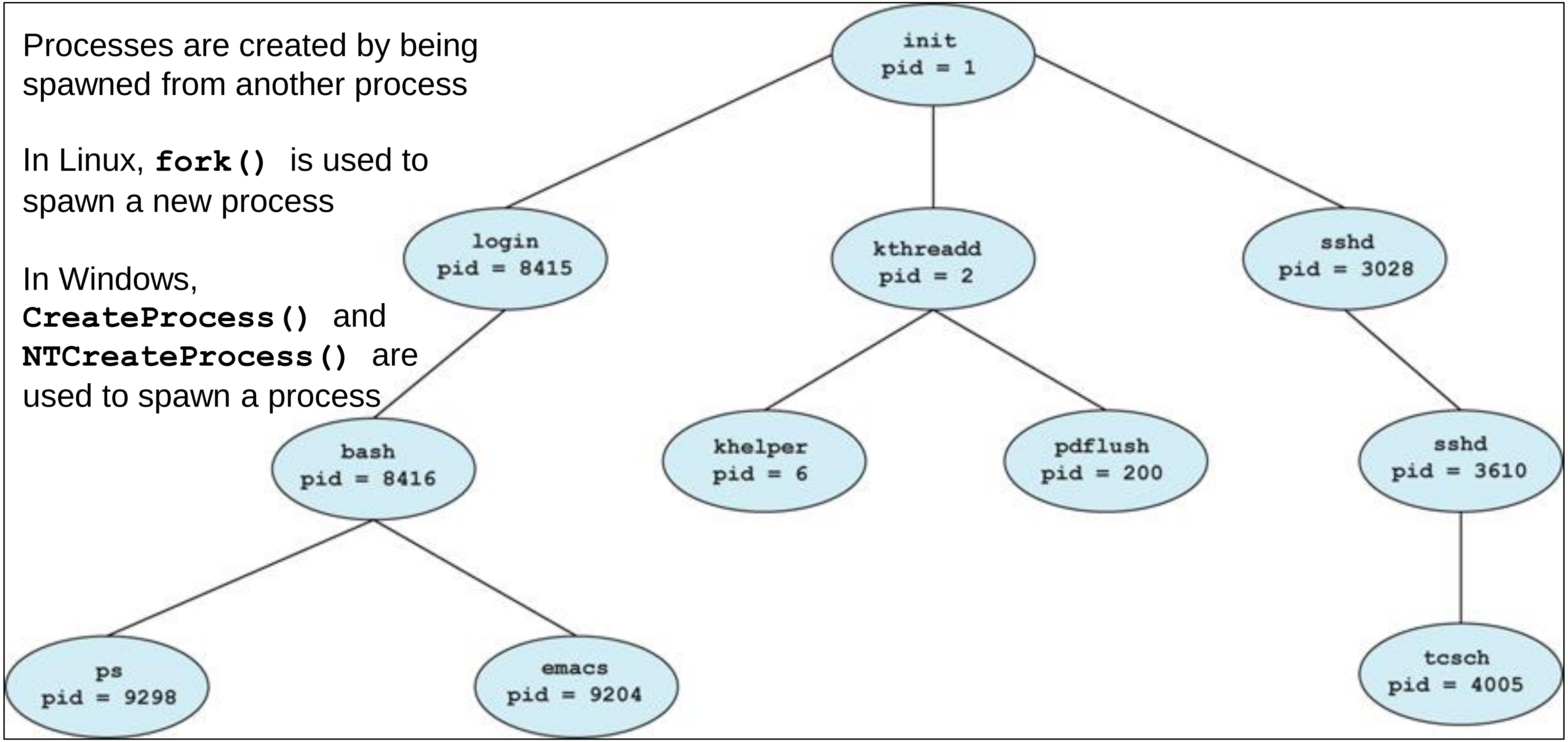
Processes have a set address space, meaning that bad guys could inject malicious code into another process to avoid detection (process injection)

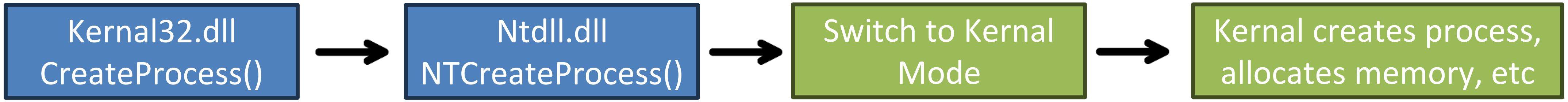


Processes are created by being spawned from another process

In Linux, **`fork()`** is used to spawn a new process

In Windows, **`CreateProcess()`** and **`NTCreateProcess()`** are used to spawn a process





The OS keeps track of helpful information for each process.

Every process has a process ID (PID). This is a unique value assigned to the process by the OS.

For each process, it will keep track of the parent process ID (PPID)
→ Very helpful to see why a process is running and its purpose

Process	Description
smss.exe PID: 4 PPID: 0	Windows Session Manager used to load the kernel and start session during boot
explorer.exe PID: 6 PPID: 4	Controls the Desktop, taskbar, file explorer GUI
program.exe PID: 77 PPID: 6	User application that was created by clicking an icon on the GUI

0 = no parent

The OS keeps track of helpful information for each process.

Every process has a process ID (PID). This is a unique value assigned to the process by the OS.

For each process, it will keep track of the parent process ID (PPID)
→ Very helpful to see why a process is running and its purpose

smss.exe
PID: 4
PPID: 0

Windows Session Manager used to load the kernel and start session during boot

0 = no parent

explorer.exe
PID: 6
PPID: 4

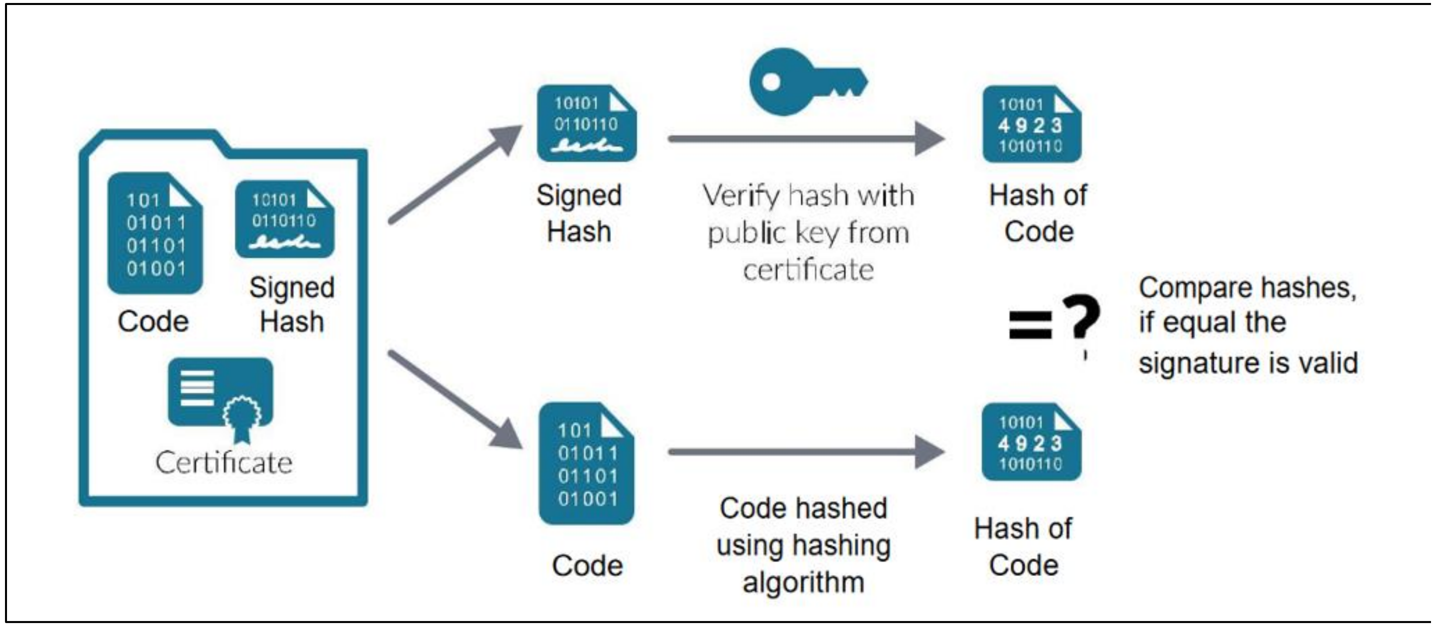
Controls the Desktop, taskbar, file explorer GUI

program.exe
PID: 77
PPID: 6

powershell.exe
PID: 120
PPID:77

If we see a suspicious process, we can check the PPID to see where it came from

“Trusted” windows process creates it= likely benign behavior
Untrusted process creates it = possibly malicious



Microsoft provides a “Trusted Signature” or digital signature for their `.exe` files. Potentially malicious .exe files or processes will not have a trusted signature

Signature Verification

Signed file, valid signature

File Version Information

Original Name	AddInProcess64.exe
Internal Name	AddInProcess64.exe
File Version	1.0.0.0
Date signed	2025-03-08 16:03:00 UTC

Signers

— InLine

Name	InLine
Status	Valid
Issuer	Microsoft ID Verified CS EOC CA 01
Valid From	01:11 PM 03/08/2025
Valid To	01:11 PM 03/11/2025
Valid Usage	1.3.6.1.4.1.311.97.1.0, Code Signing, 1.3.6.1.4.1.311.97.651
Algorithm	sha384RSA
Thumbprint	91C71CD1B81D99B0B51D1EC542A4FB27BBBF42E2
Serial Number	33 00 02 01 B5 D7 23 72 EE 41 6F 0D 8F 00 00 00 02 01 B5

+ Microsoft ID Verified CS EOC CA 01
+ Microsoft ID Verified Code Signing PCA 2021
+ Microsoft Identity Verification Root Certificate Authority 2020

Counter Signers

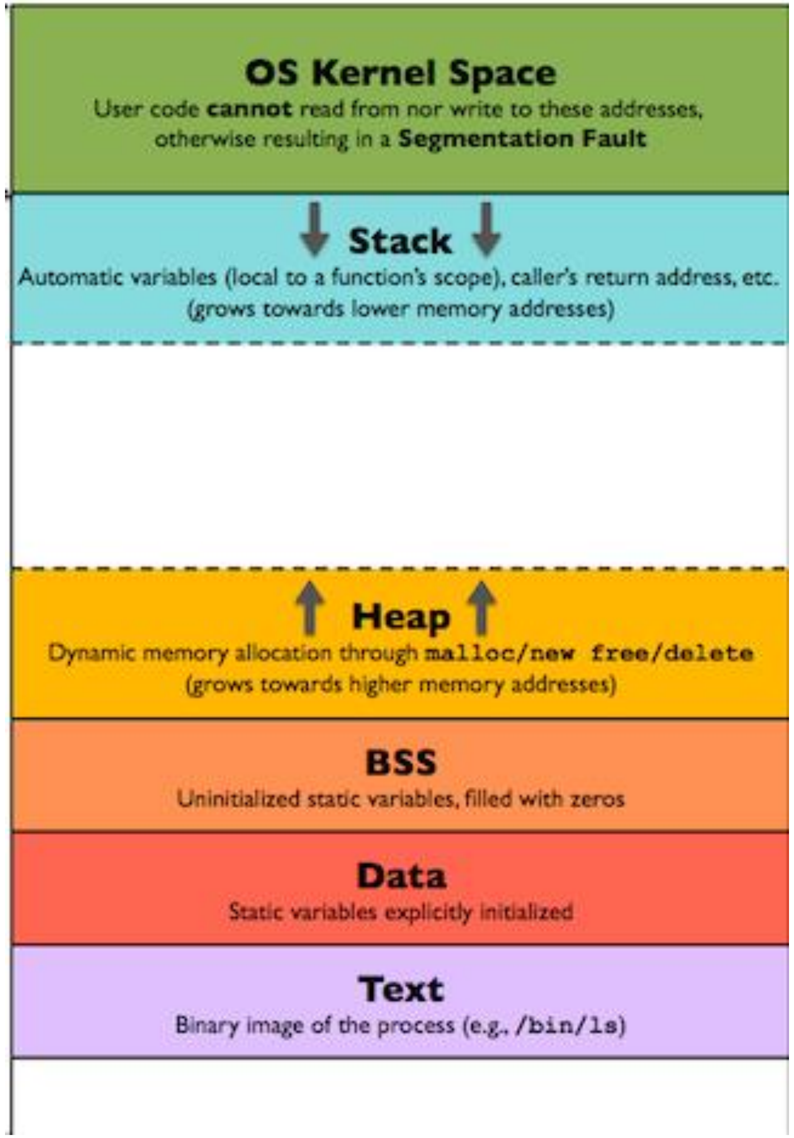
+ Microsoft Public RSA Time Stamping Authority
+ Microsoft Public RSA Timestamping CA 2020
+ Microsoft Identity Verification Root Certificate Authority 2020

Process Address Space

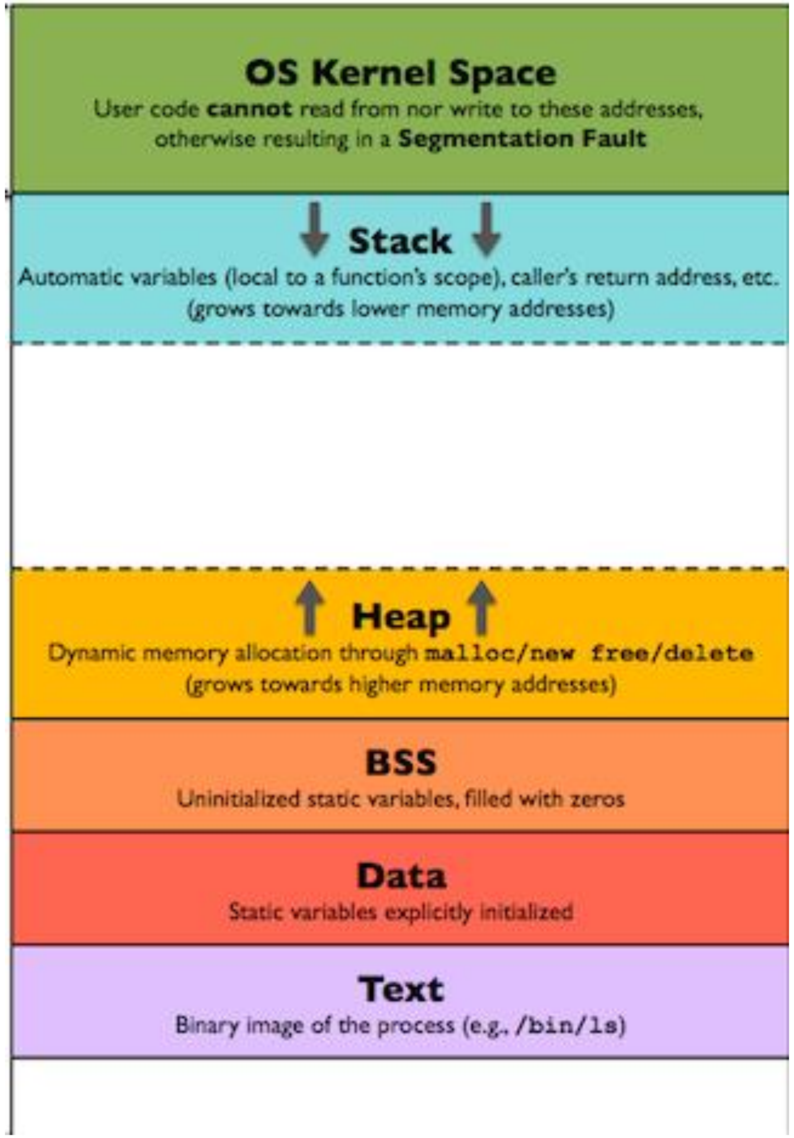
The process address space has all the code and data for a specific process. Each process gets their own

This is the stuff that will exist in RAM!

We can "dump" a process (with a process ID) to see raw content of the process address space



Process Address Space



Variables (data)

Code for a process
(interesting stuff)

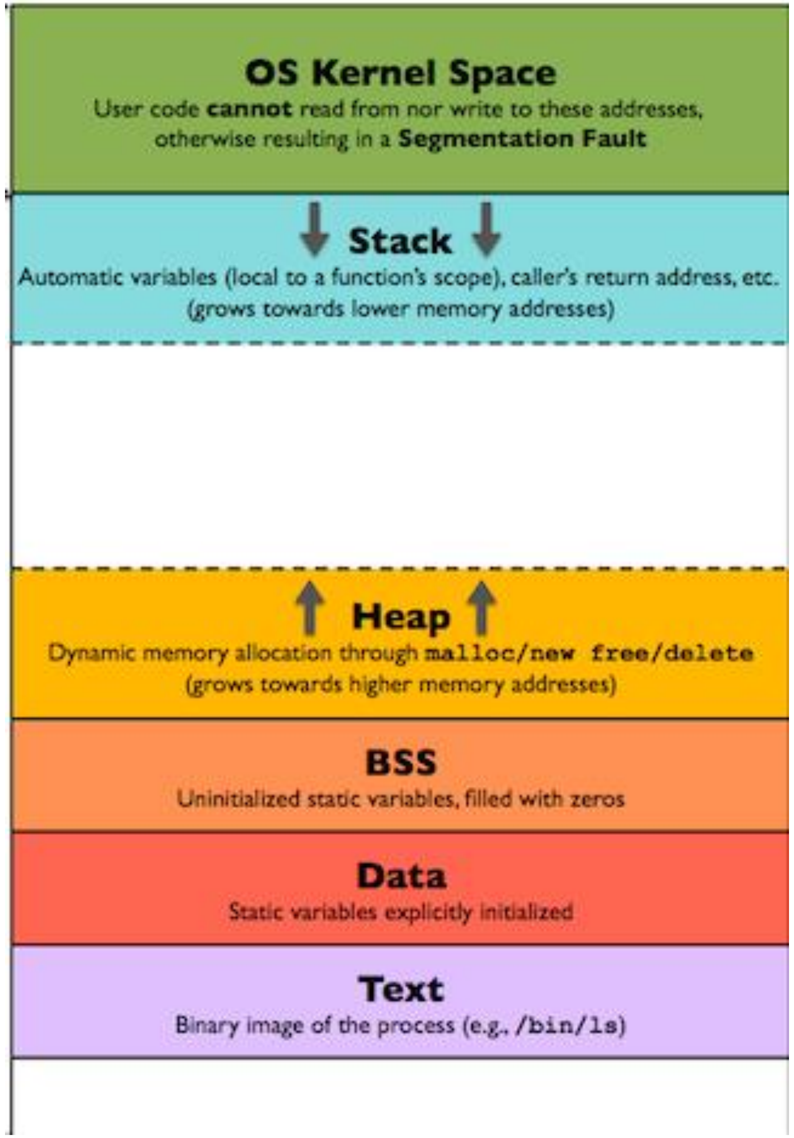
Process Address Space

The stack is a valuable section for forensics, as provides valuable insight into which code was being executed and what data was being processed

Temporary variables and data

Memory for data requested by malloc

Code for a process (interesting stuff)



Process Address Space

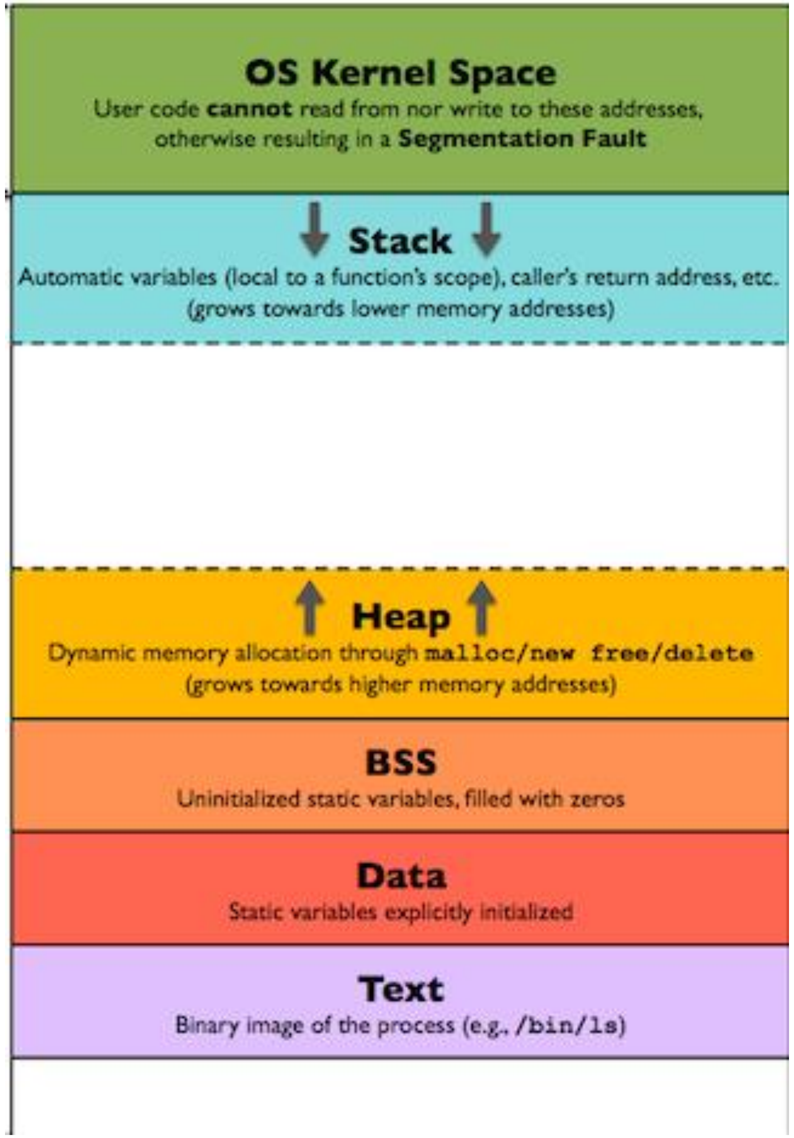
The stack is a valuable section for forensics, as provides valuable insight into which code was being executed and what data was being processed

OS System calls and code

Temporary variables and data

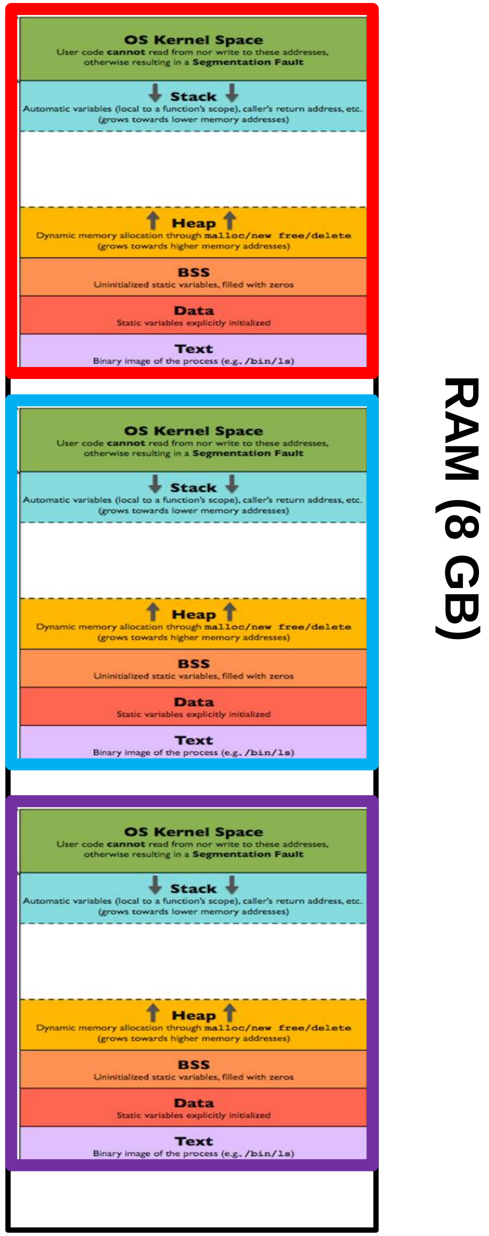
Memory for data requested by malloc

Code for a process (interesting stuff)



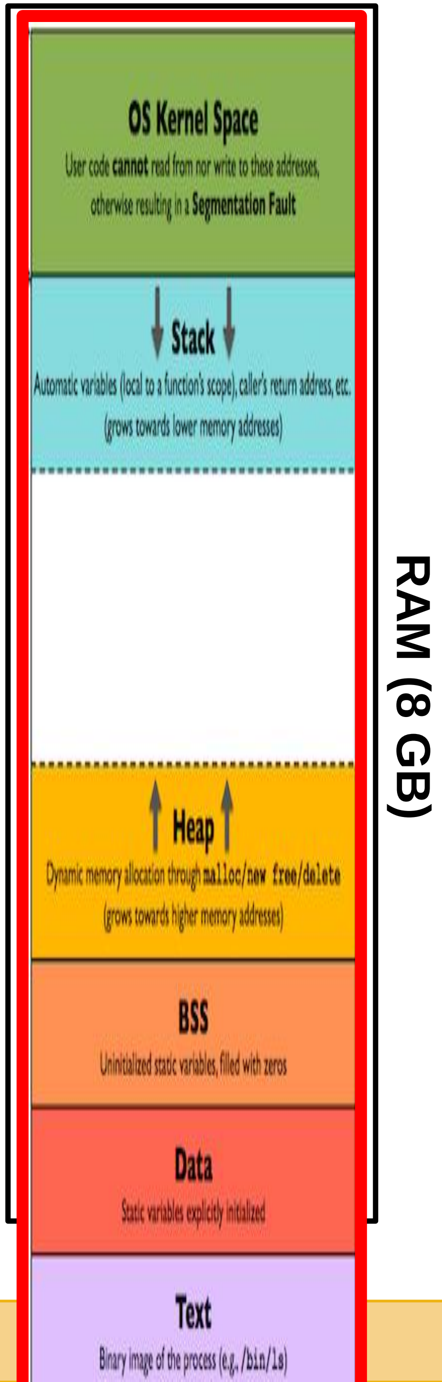
Memory

For a process to be executed, its process address space must exist in RAM

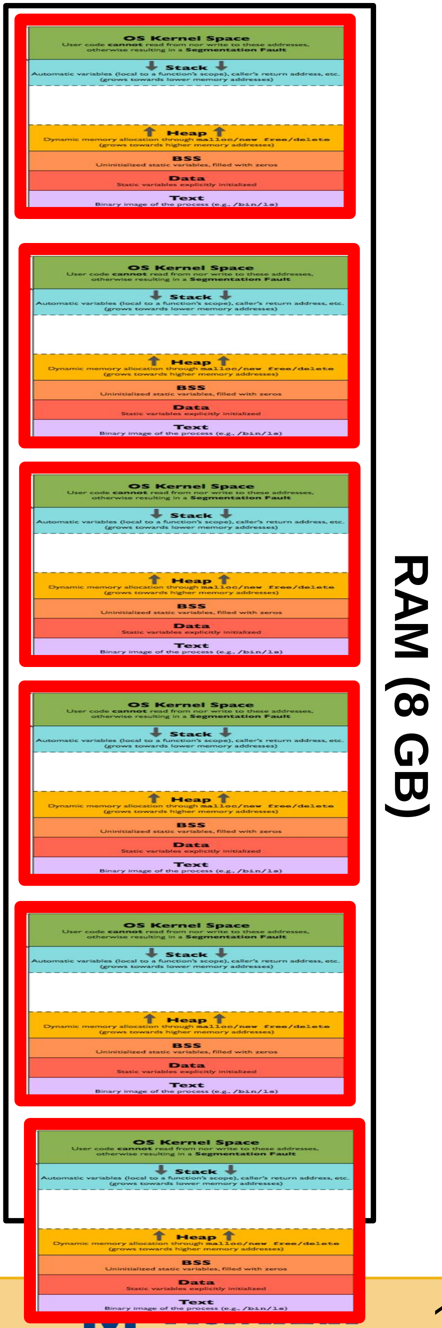


Memory

What if we have a really big process?

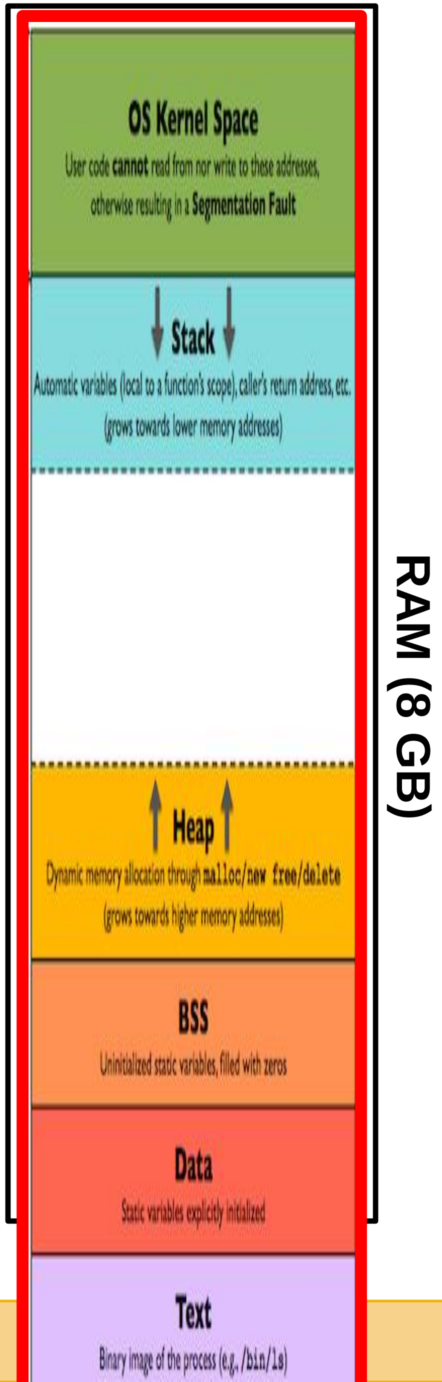


What if we have a lot of processes that all need to be in RAM?



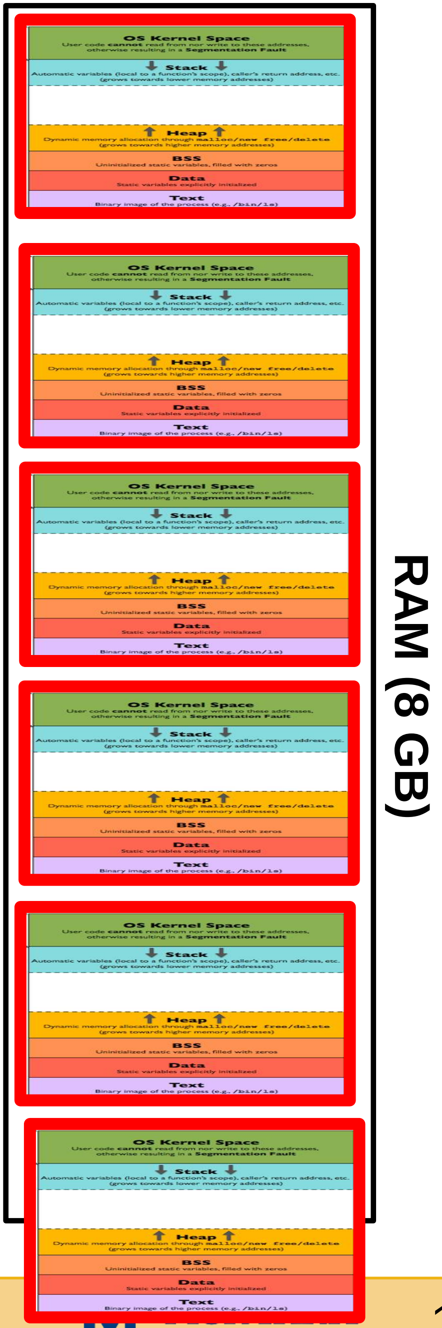
Memory

What if we have a really big process?



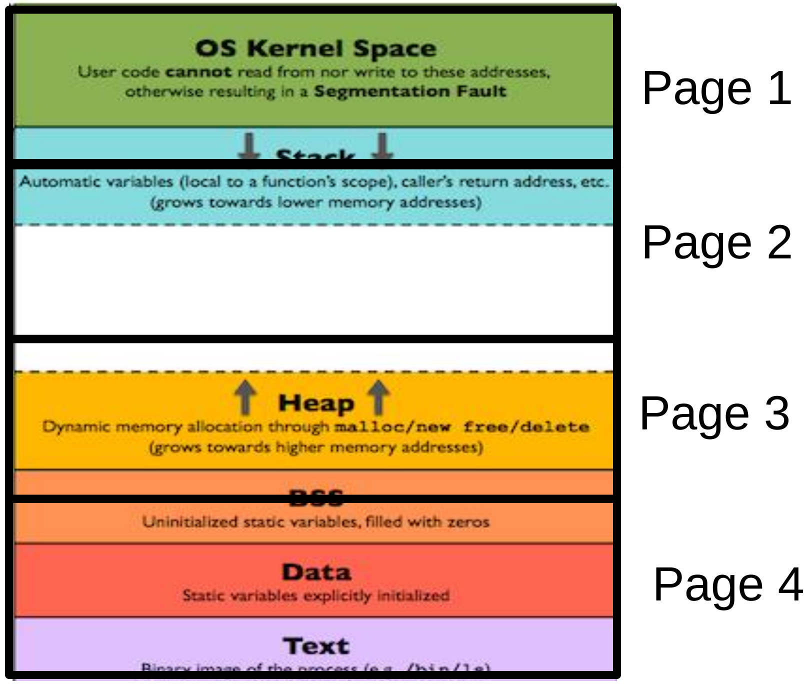
What if we have a lot of processes that all need to be in RAM?

Virtual Memory



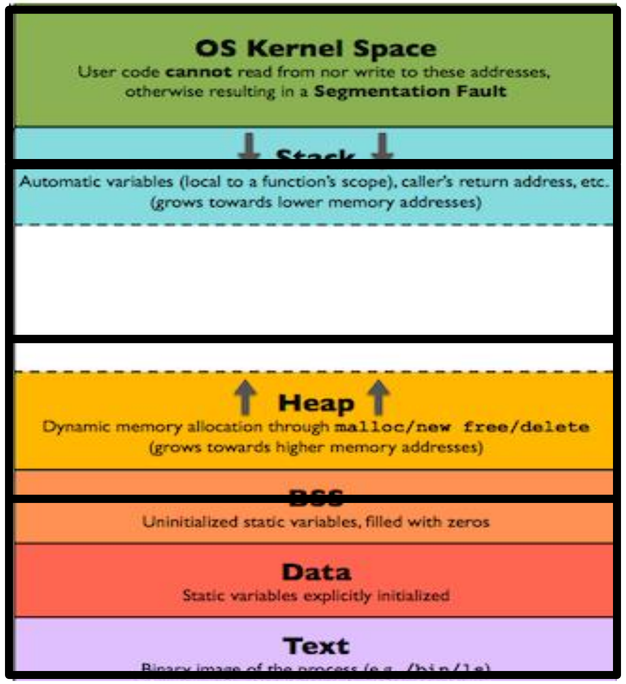
Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

Process address space is split into fixed-size **pages**



Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

Process address space is split into fixed-size **pages**



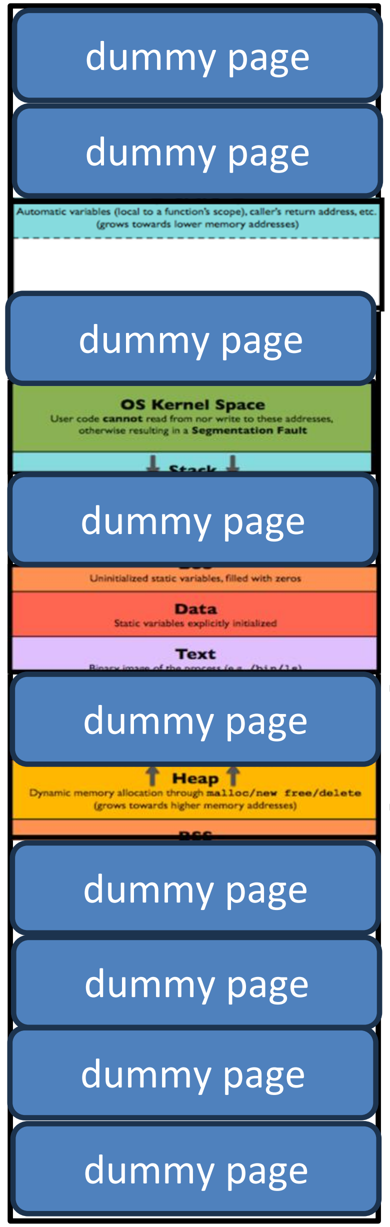
Page 1

Page 2

Page 3

Page 4

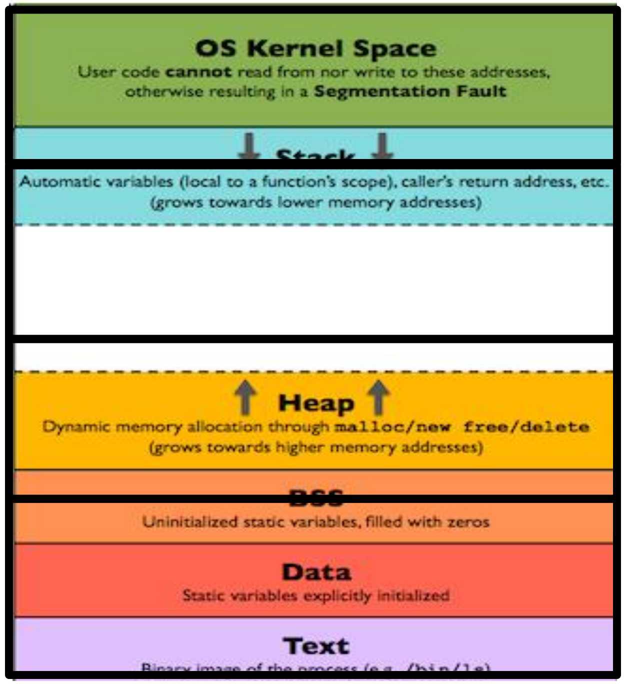
Pages may be scattered throughout memory



RAM (8 GB)

Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

Process address space is split into fixed-size **pages**



Page 1

Page 2

Page 3

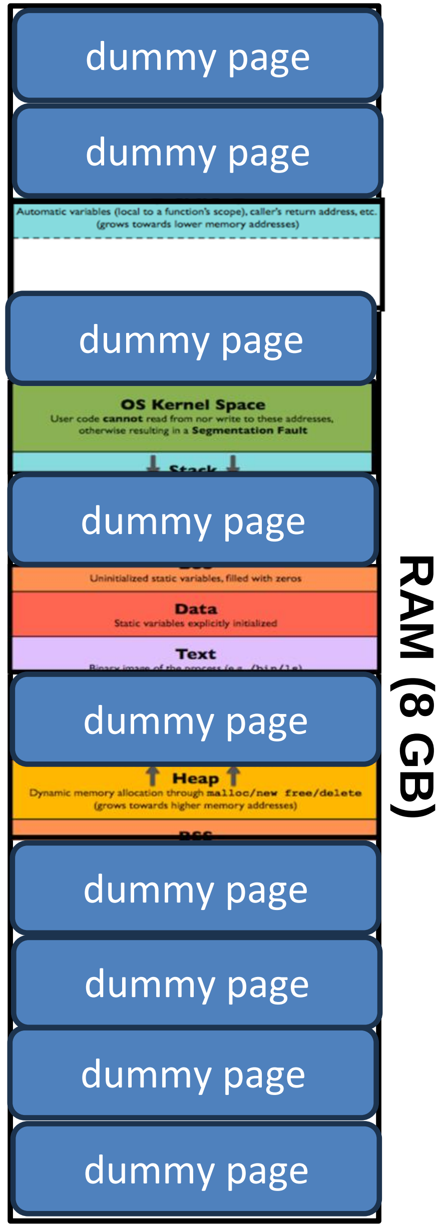
Page 4

Pages may be scattered throughout memory

Suppose a new page needs to be added into RAM

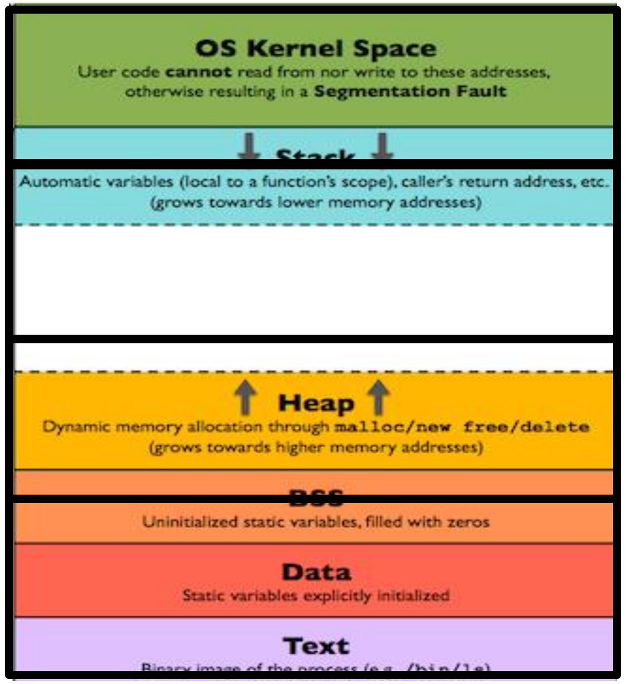
dummy page

The OS will determine which page to remove from RAM and swap into secondary storage



Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

Process address space is split into fixed-size **pages**



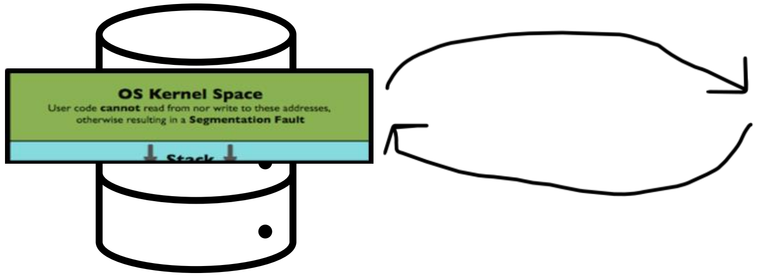
Page 1

Page 2

Page 3

Page 4

Disk

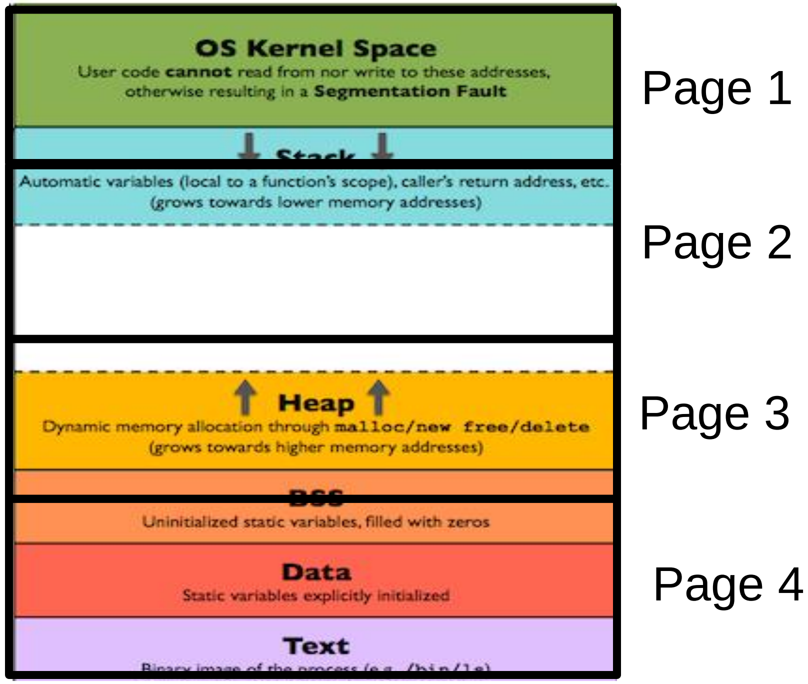


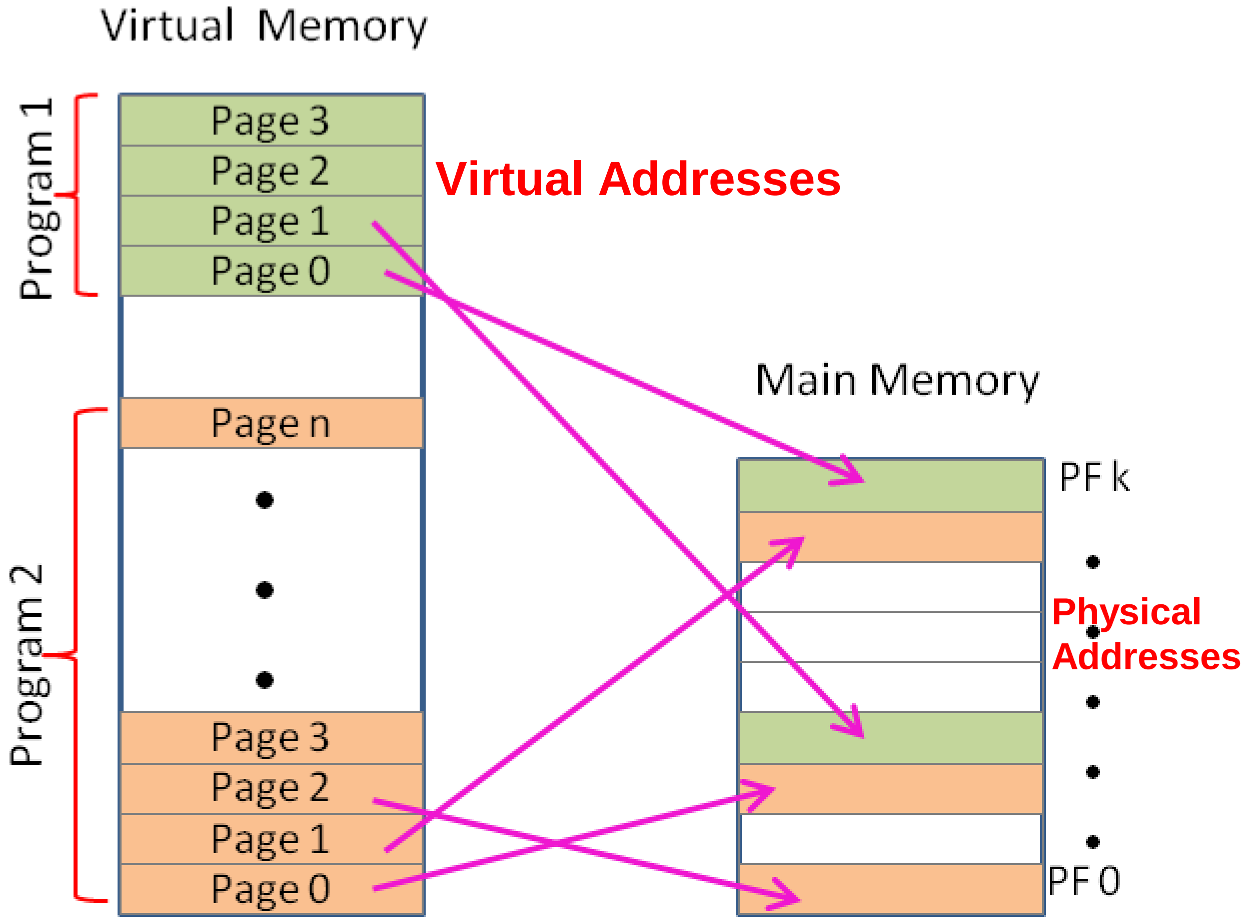
RAM (8 GB)

On demand paging will swap unused pages from RAM to Disk

Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

Process address space is split into fixed-size **pages**

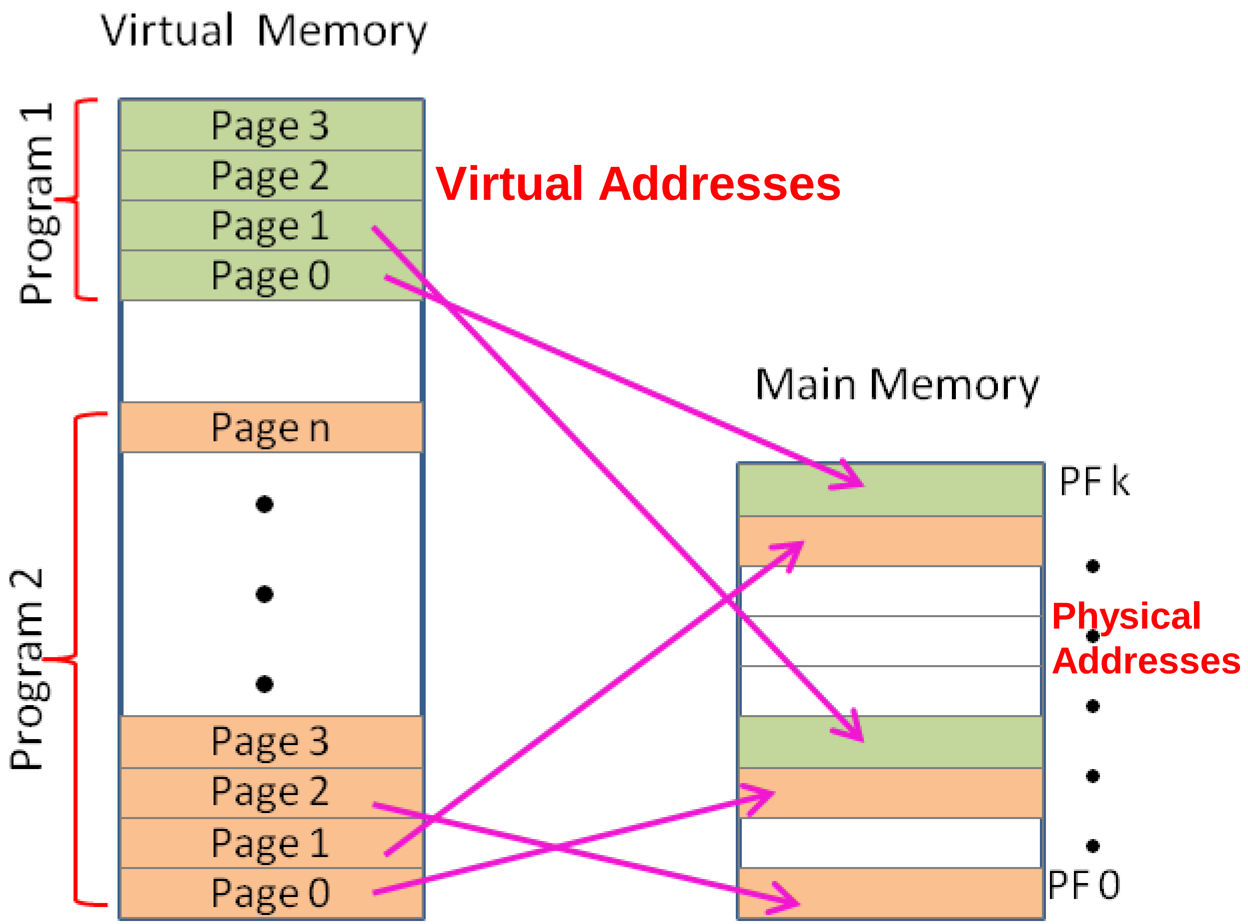




Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

Process address space is split into fixed-size **pages**

We now need a way to convert virtual addresses to physical addresses (and whether they are on disk or RAM)

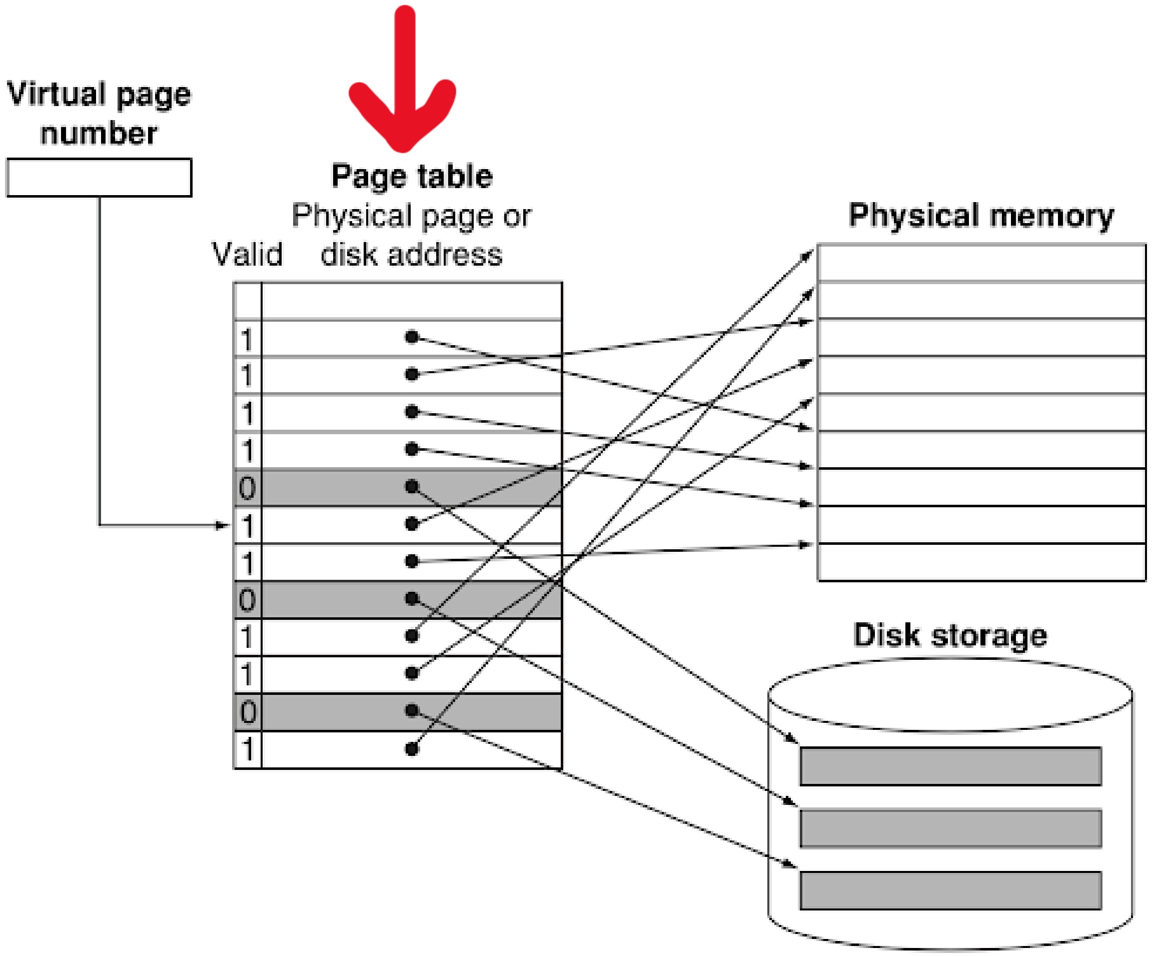


Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

We now need a way to convert virtual addresses to physical addresses (and whether they are on disk or RAM)

We have a page table that will look at a virtual address of a page , and returns the physical address of a page

Do we have one page single table?



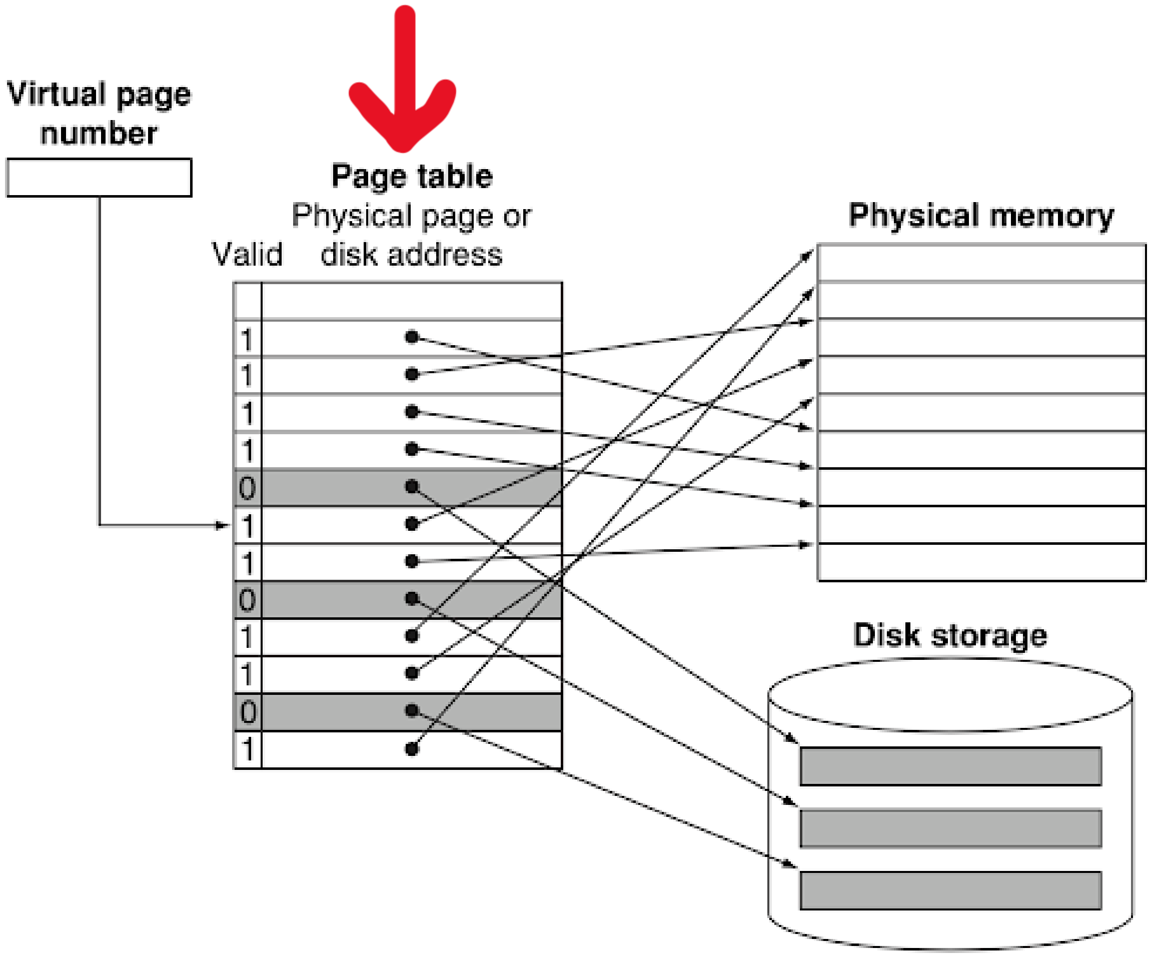
Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)

We now need a way to convert virtual addresses to physical addresses (and whether they are on disk or RAM)

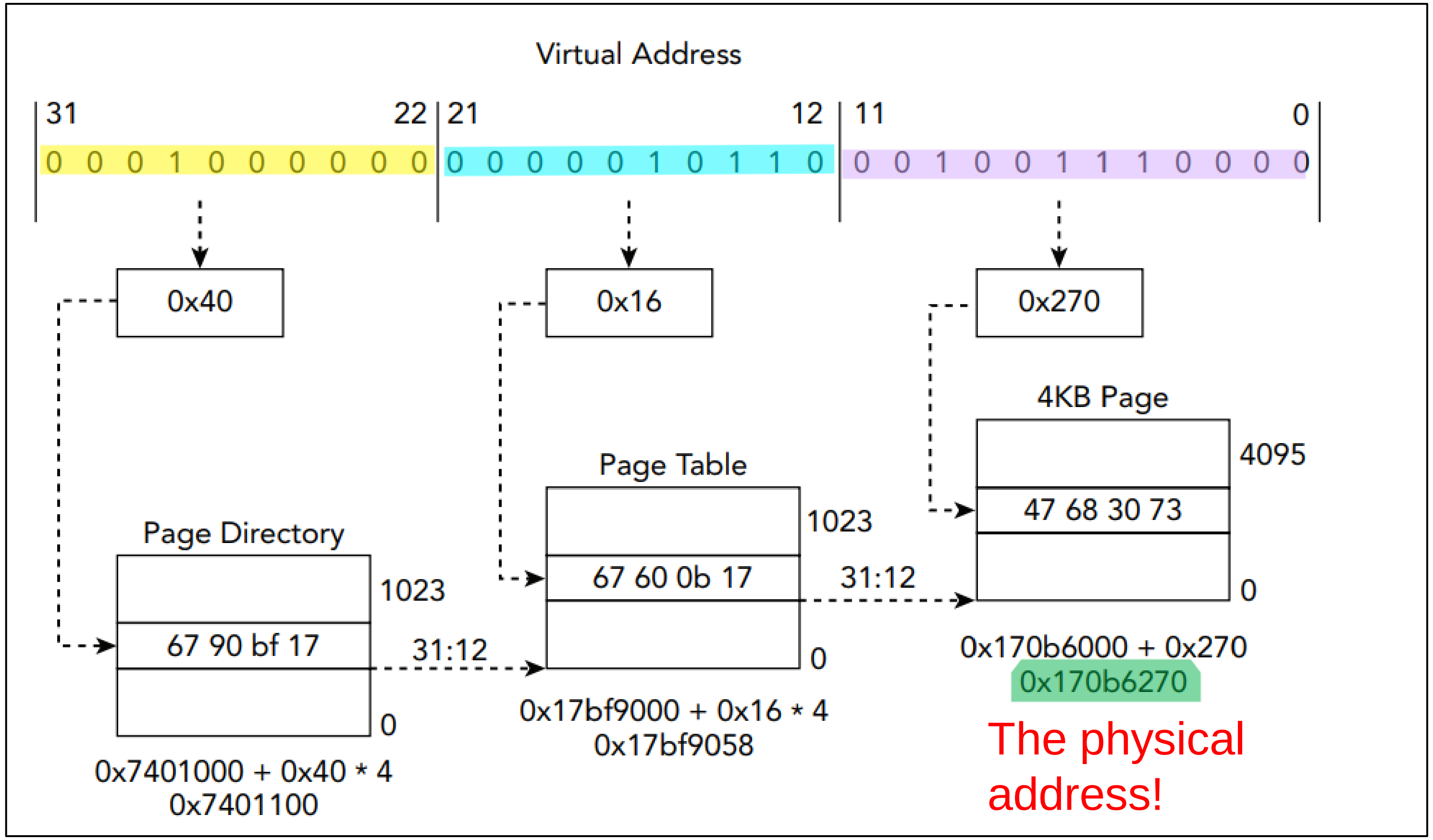
We have a page table that will look at a virtual address of a page , and returns the physical address of a page

Do we have one page single table?

No, most OS have *several* page tables



Virtual Memory is a memory management technique that allows a computer to swap memory pages between RAM and secondary storage (disk)



Virtual addresses consist of different pieces.

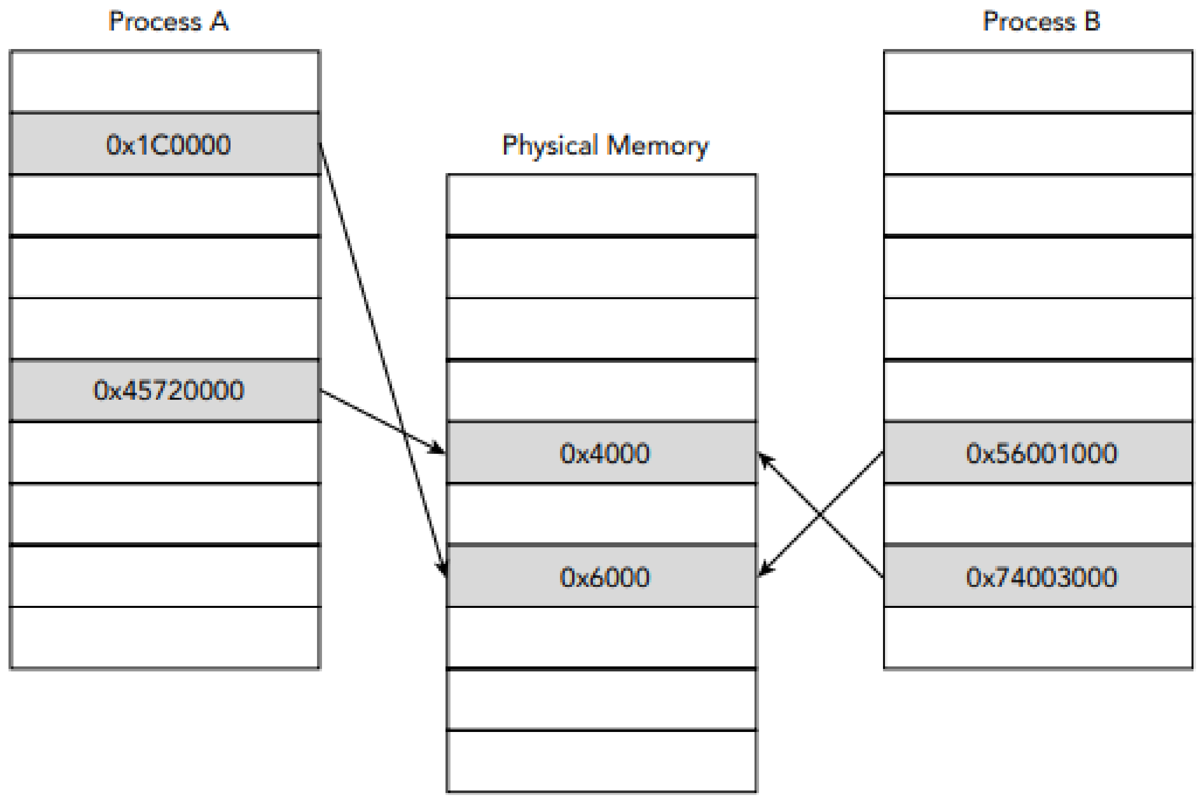
These pieces are used at different points in the process

Where to look (offset) for page directory

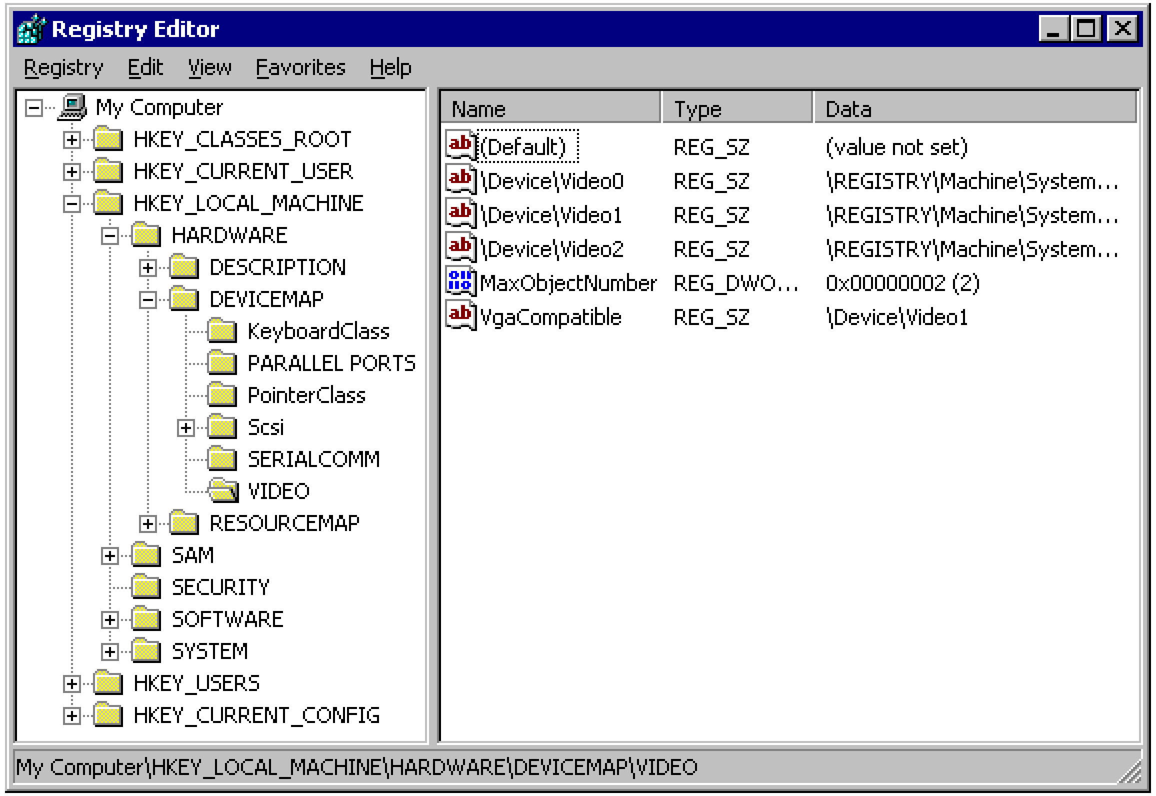
Where to look (offset) for page table

Where to look (offset) for page

Processes can share memory



Windows Registry



The **Windows registry** is a database of key-value pairs for information, settings, options, and other values for software and hardware

Windows Data Structures

Table 2-1: Common Storage Sizes for C Basic Data Types

Type	32-Bit Storage Size (Bytes)	64-Bit Storage Size (Bytes)
`char`	1	1
`unsigned char`	1	1
`signed char`	1	1
`int`	4	4
`unsigned int`	4	4
`short`	2	2
`unsigned short`	2	2
`long`	4	Windows: 4, Linux/Mac: 8
`unsigned long`	4	Windows: 4, Linux/Mac: 8
`long long`	8	8
`unsigned long long`	8	8
`float`	4	4
`double`	8	8
`pointer`	4	8

Table 2-2: Common Storage Sizes for Some Windows Types

Type	32-Bit Size (Bytes)	64-Bit Size (Bytes)	Purpose/Native Type
`DWORD`	4	4	Unsigned long
`HMODULE`	4	8	Pointer/handle to a module
`FARPROC`	4	8	Pointer to a function
`LPSTR`	4	8	Pointer to a character string
`LPCWSTR`	4	8	Pointer to a Unicode string

Windows Data Structures

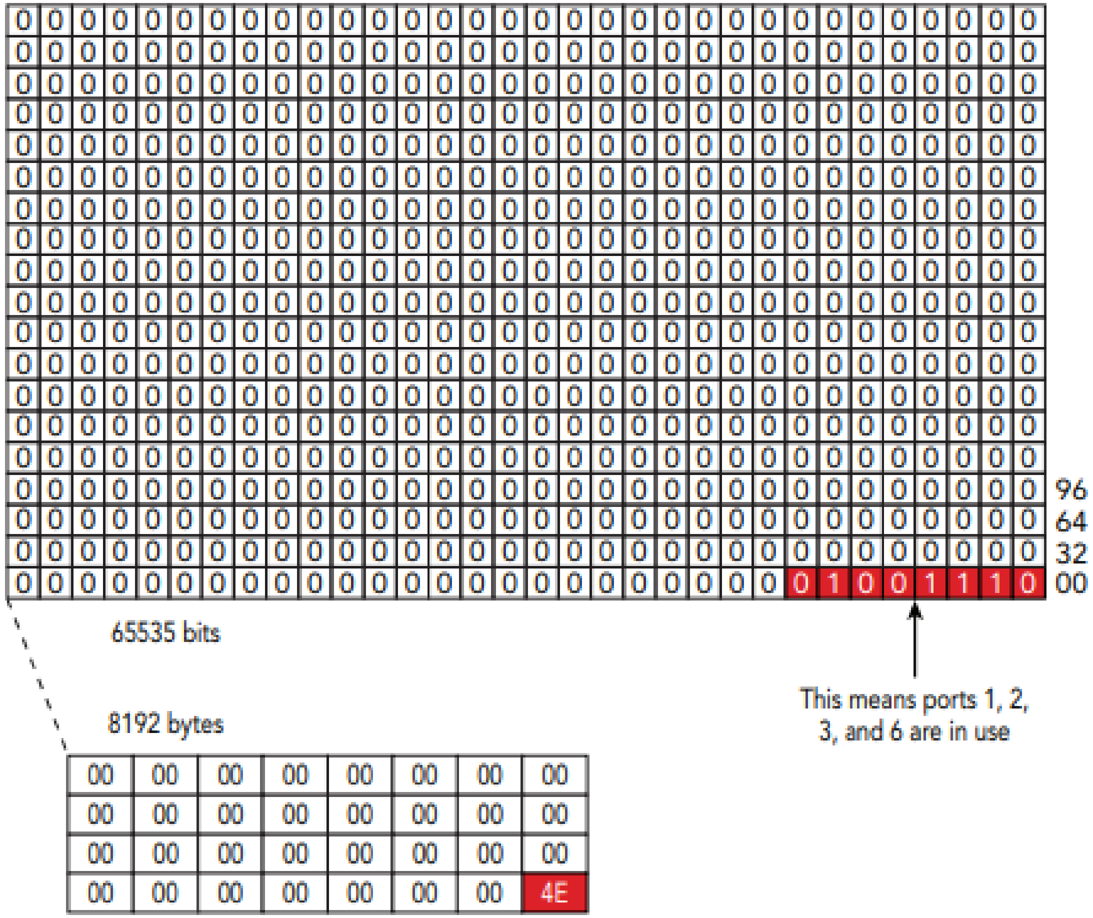


Figure 2-3: An example of a Windows bitmap of in-use network ports

Bitmap is a sequence of zeros and ones

Windows uses an internal bitmap to represent which ports are in use

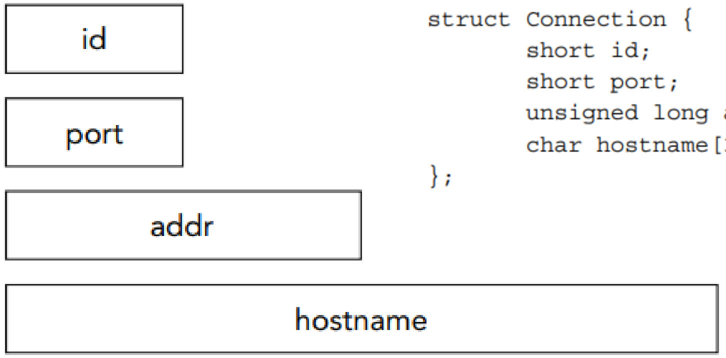
Windows Data Structures

Table 2-3: Structure Type Information for a Network Connection Example

Byte Range	Name	Type	Description
0–1	`id`	`short`	Unique record ID
2–3	`port`	`short`	Remote port
4–7	`addr`	`unsigned long`	Remote address
8–39	`hostname`	`char[32]`	Remote hostname

Another mechanism for commonly aggregating data is a **record** (struct-like data structure).

Knowing common record formats makes reading through raw Hex much easier!




```
struct Connection {
        short id;
        short port;
        unsigned long addr;
        char hostname[32];
};
```

Figure 2-4: Network connection record example

```
0000000: 0100 0050 ad3d de09 7777 772e 766f 6c61   ...P.>X.www.vola
0000010: 7469 6c69 7479 666f 756e 6461 7469 6f6e  tilityfoundation
0000020: 2e6f 7267 0000 0000                       .org....
```