Syscalls, exceptions, and interrupts, …oh my!

Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

[Altinbuken, Weatherspoon, Bala, Bracy, McKee, and Sirer]

Announcements

- P4-Buffer Overflow is due tomorrow
  - Due Tuesday, April 16th

- C practice assignment
  - Due Friday, April 19th

Outline for Today

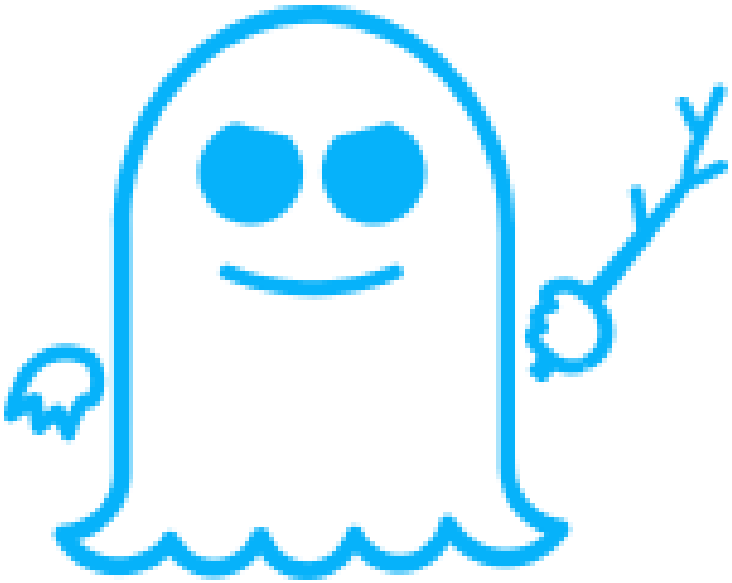
- How do we protect processes from one another?
  - Skype should not crash Chrome.
- How do we protect the operating system (OS) from other processes?
  - Chrome should not crash the computer!
- How does the CPU and OS (software) handle exceptional conditions?
  - Division by 0, Page Fault, Syscall, etc.

Outline for Today

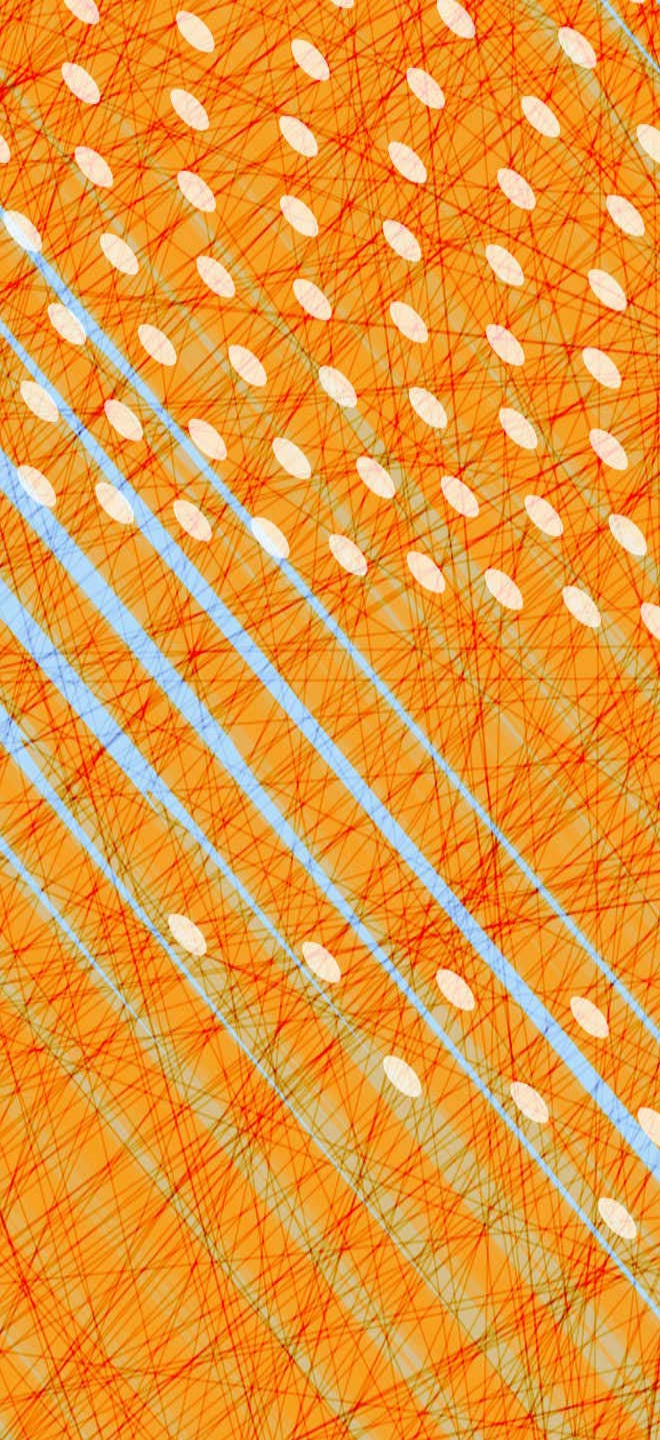
- How do we protect processes from one another?
  - Skype should not crash Chrome.
- Operating System
- How do we protect the operating system (OS) from other processes?
  - Chrome should not crash the computer!
- Privileged Mode
- How does the CPU and OS (software) handle exceptional conditions?
  - Division by 0, Page Fault, Syscall, etc.
- Traps, System calls, Exceptions, Interrupts

Meltdown and Spectre Security Bug

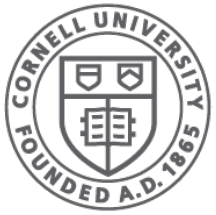
MELTDOWN



SPECTRE



Operating System



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

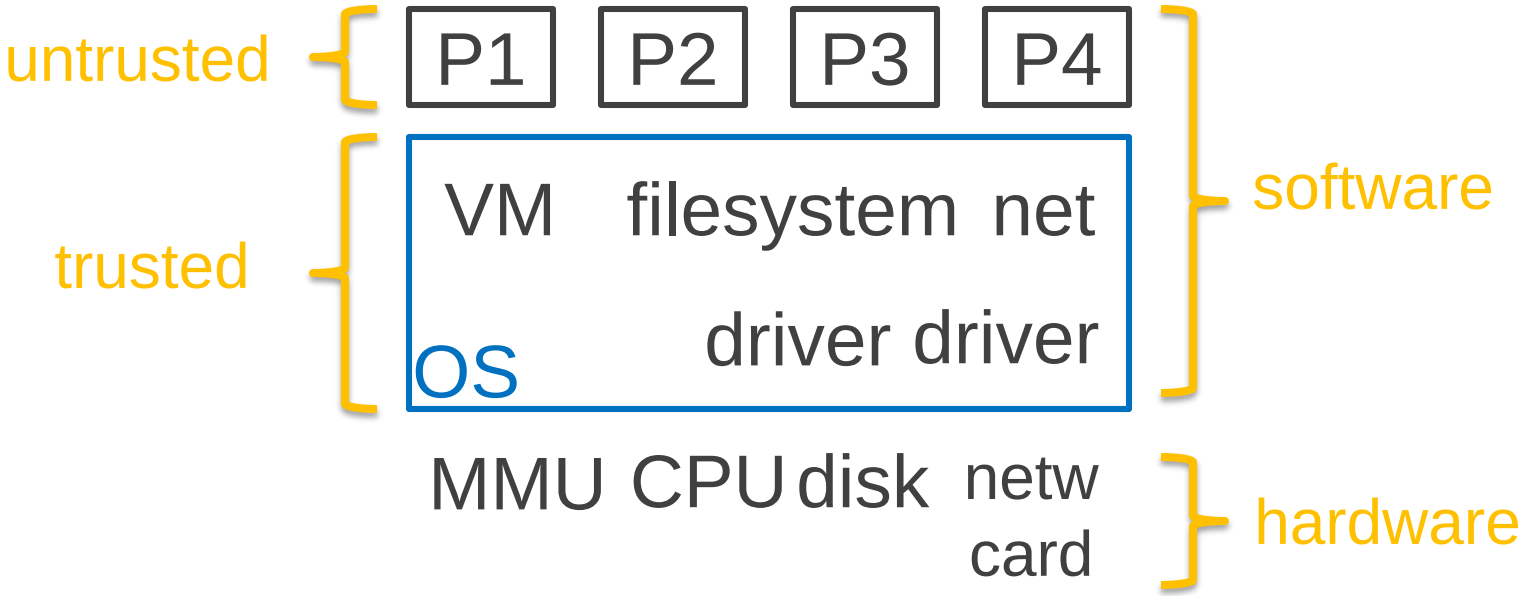
Operating System

- Manages all of the software and hardware on the computer.
- Many processes running at the same time, requiring resources
  - CPU, Memory, Storage, etc.

- The Operating System multiplexes these resources amongst different processes, and isolates and protects processes from one another!

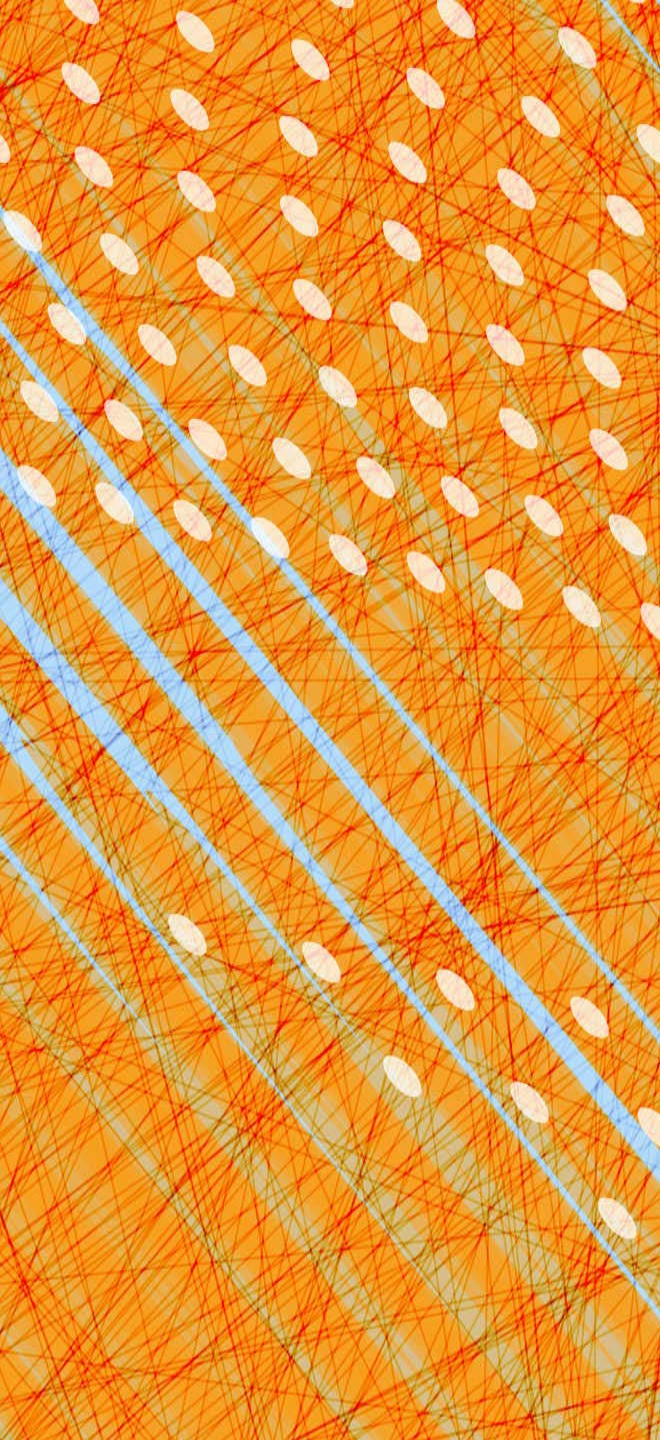
Operating System

- Operating System (OS) is a trusted mediator:
  - *Safe control transfer between processes*
  - *Isolation (memory, registers) of processes*

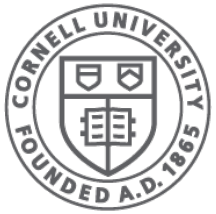


Outline for Today

- How do we protect processes from one another?
  - Skype should not crash Chrome.
  - Operating System
- How do we protect the operating system (OS) from other processes?
  - Chrome should not crash the computer!
  - Privileged Mode
- How does the CPU and OS (software) handle exceptional conditions?
  - Division by 0, Page Fault, Syscall, etc.
  - Traps, System calls, Exceptions, Interrupts

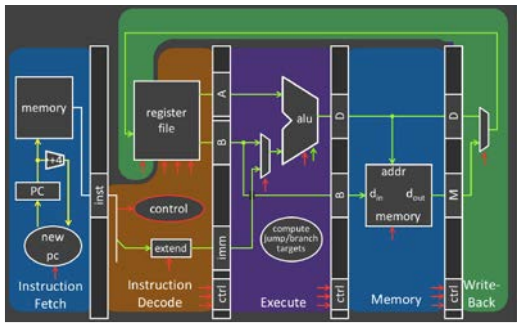


Privileged (Kernel) Mode



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

One Brain, Many Personalities



Brain

You are what you execute.

Personalities:
hailstone_recursive
Microsoft Word
Minecraft
Linux ← *yes, this is just software like every other program that runs on the CPU*

Are they all equal?

Trusted vs. Untrusted

- Only trusted processes should access & change important things
  - Editing TLB, Page Tables, OS code, OS sp, OS fp…

- If an untrusted process could change the OS' sp/fp/gp/*etc*., OS would crash!

Privileged Mode

CPU Mode Bit in Process Status Register

- Many bits about the current process
- Mode bit is just one of them

- Mode bit:
  - **0 = user mode = untrusted**:
    "Privileged" instructions and registers are disabled by CPU
  - **1 = kernel mode = trusted**
    All instructions and registers are enabled

Privileged Mode at Startup

1. Boot sequence
   - load first sector of disk (containing OS code) to predetermined address in memory
   - Mode ← 1; PC ← predetermined address

2. OS takes over
   - initializes devices, MMU, timers, etc.
   - loads programs from disk, sets up page tables, *etc.*
   - Mode ← 0; PC ← program entry point
     - User programs regularly yield control back to OS

Users need access to resources

- If an untrusted process does not have privileges to use system resources, how can it
  - Use the screen to print?
  - Send message on the network?
  - Allocate pages?
  - Schedule processes?

Solution: System Calls

System Call Examples

`putc()`: Print character to screen

- Need to multiplex screen between competing processes

`send()`: Send a packet on the network

- Need to manipulate the internals of a device

`sbrk()`: Allocate a page

- Needs to update page tables & MMU

`sleep()`: put current prog to sleep, wake other

- Need to update page table base register

System Calls

System calls called *executive calls* (*ecall*) in RISC-

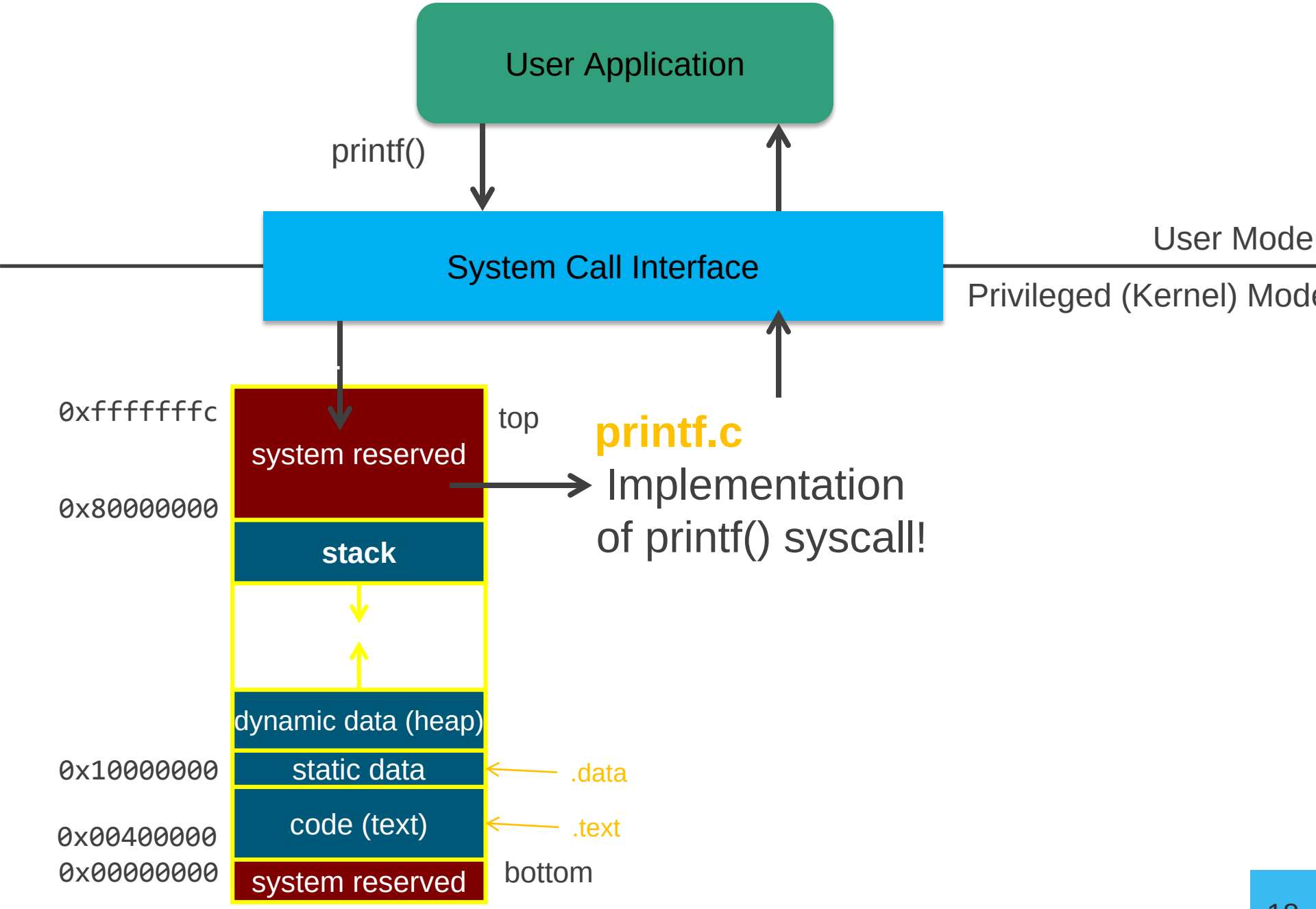
System call: Not just a function call

- Don’t let process jump just anywhere in OS code
- OS can’t trust process’ registers (sp, fp, gp, etc.)

ECALL instruction: safe transfer of control to OS

RISC-V system call convention:

- Exception handler saves temp regs, saves ra, …
- but: a7 = system call number, which specifies the operation the application is requesting


User Application
printf()
System Call Interface
User Mode
Privileged (Kernel) Mode
0xffffffff
top
system reserved
0x80000000
stack
dynamic data (heap)
0x10000000
static data
.data
code (text)
.text
0x00400000
0x00000000
system reserved
bottom
printf.c
Implementation
of printf() syscall!

Libraries and Wrappers

Compilers do not emit SYSCALL instructions

- Compiler doesn’t know OS interface

Libraries implement standard API from system API

libc (standard C library):

- getc() → ecall
- sbrk() → ecall
- write() → ecall
- gets() → getc()
- printf() → write()
- malloc() → sbrk()
- …

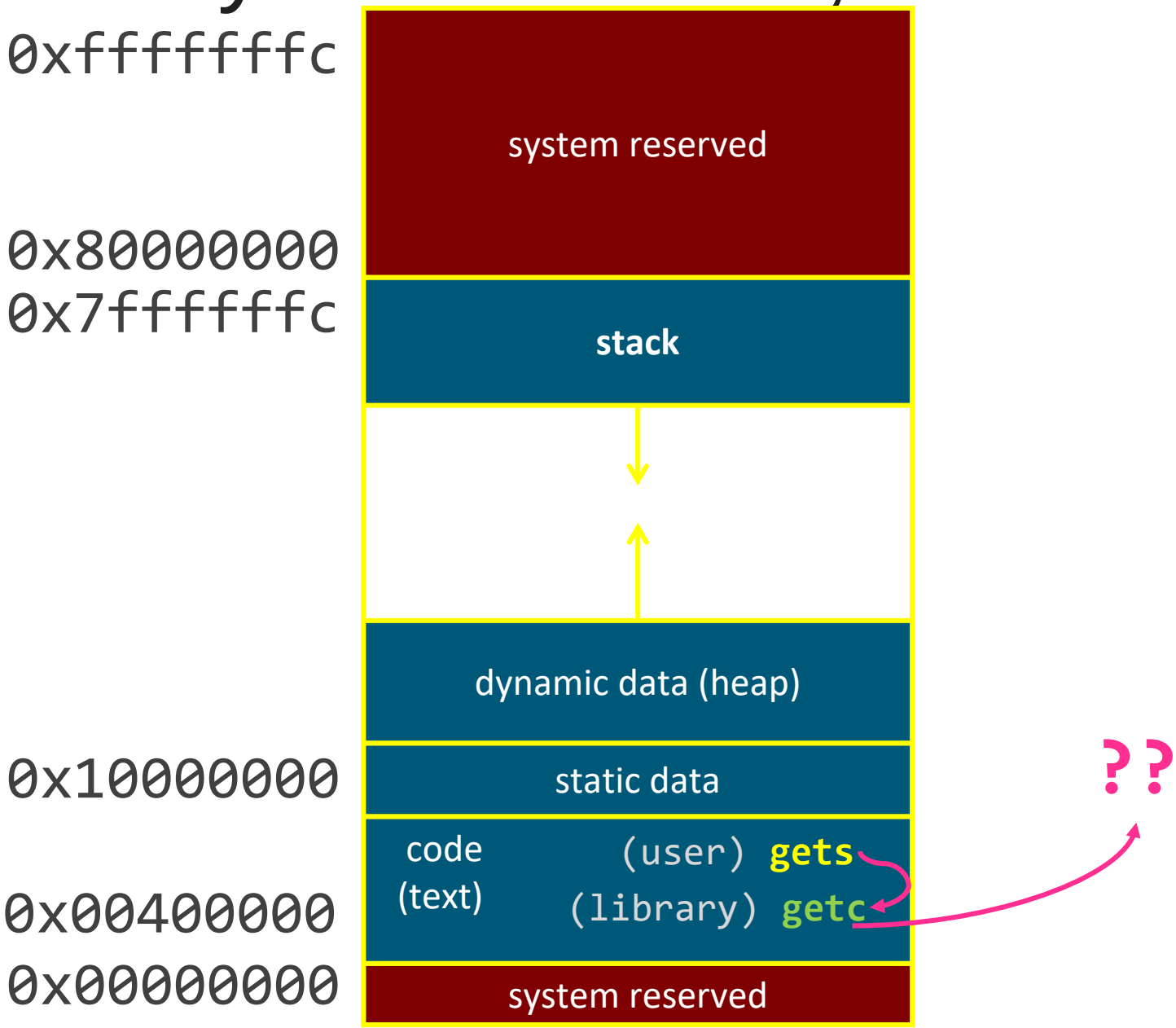
Invoking System Calls

```
char *gets(char *buf) {
  while (...) {
    buf[i] = getc();
  }
}

int getc() {
  asm("addi a7, 0, 4");
  asm(“ecall");
}
```

4 is number for getc syscall

Anatomy of a Process, v1



Where does the OS live?

In its own address space?

- Syscall has to switch to a different address space
- Hard to support syscall arguments passed as pointers

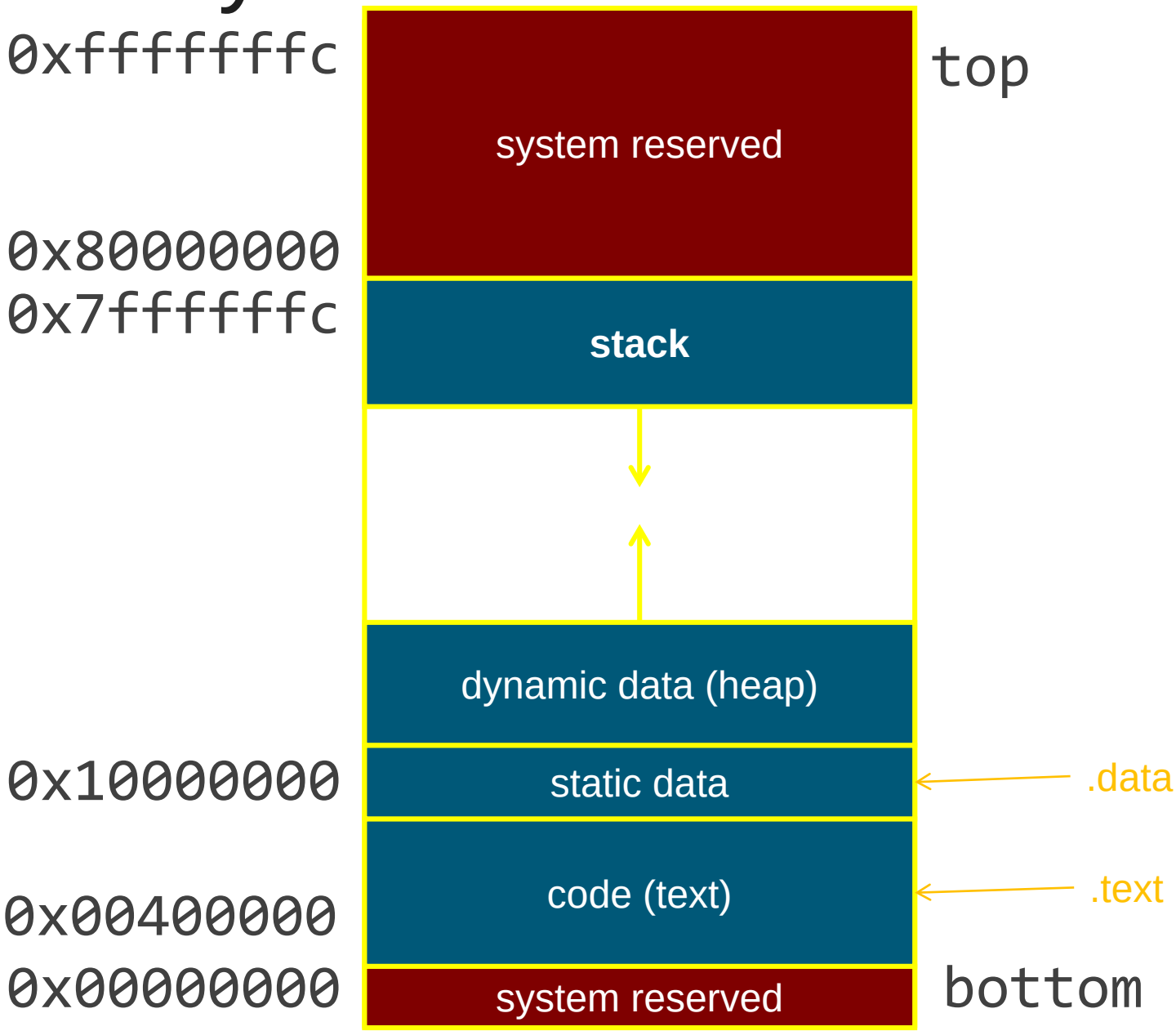
. . . So, NOPE

In the same address space as the user process?

- Protection bits prevent user code from writing kernel
- Higher part of virtual memory
- Lower part of physical memory

. . . Yes, *this is how we do it.*

Anatomy of a Process



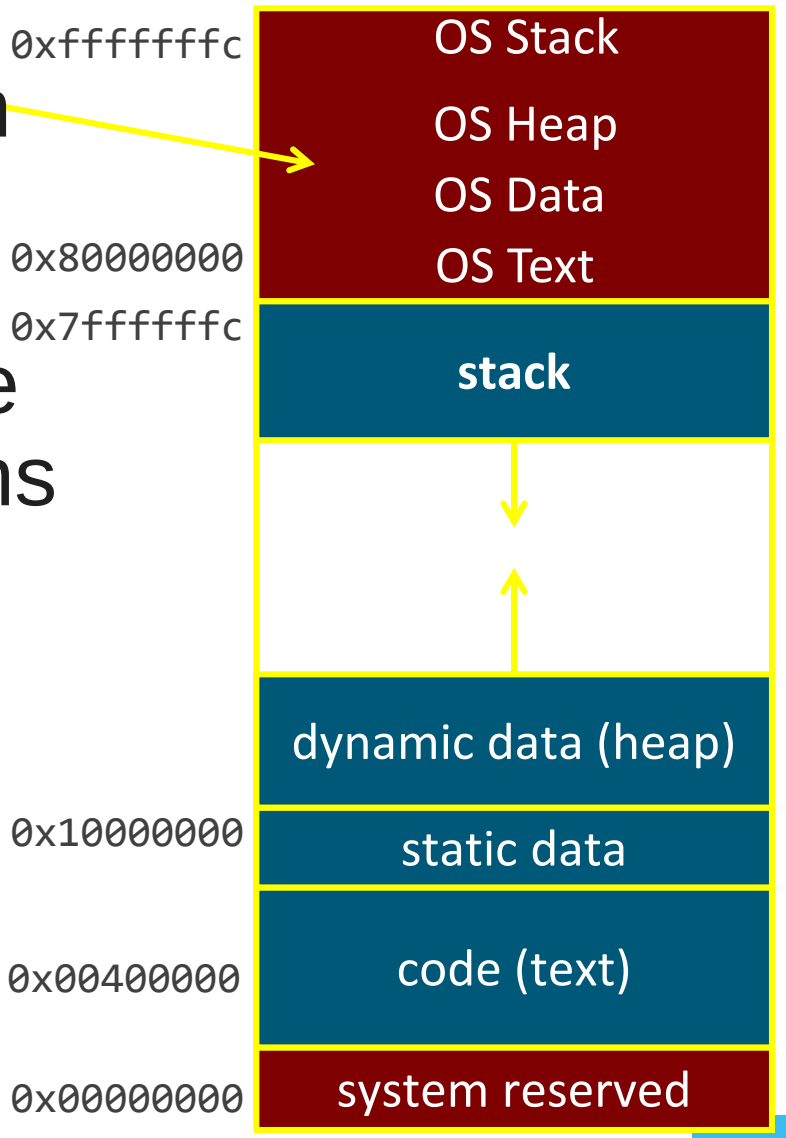
Full System Layout

All kernel text & most data:

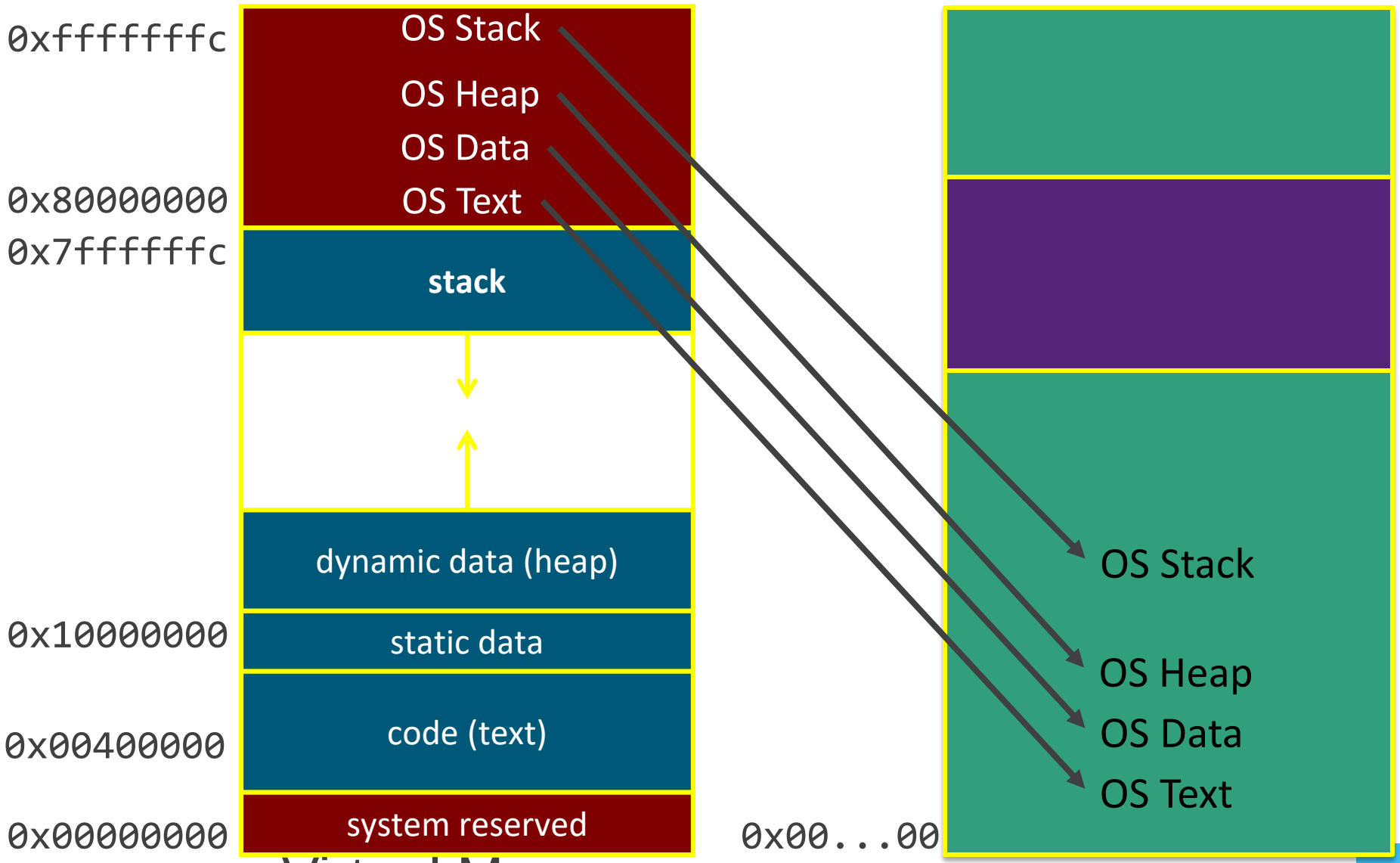
- At same virtual address in every address space

OS is omnipresent, available to help user-level applications

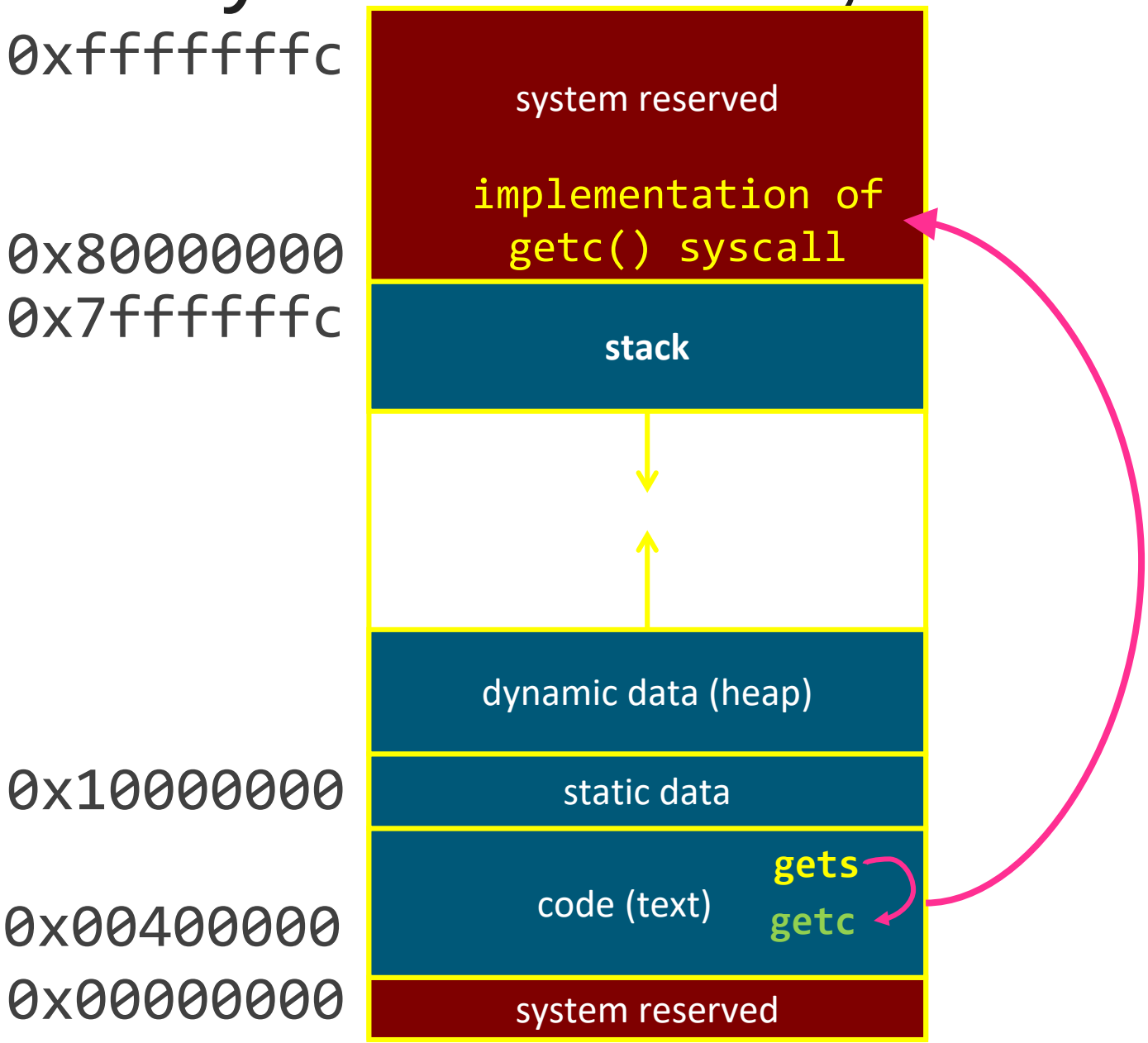
- Typically in high memory



Full System Layout



Anatomy of a Process, v2



Clicker Question

Which statement is FALSE?

A) OS manages the CPU, Memory, Devices, and Storage.
B) OS provides a consistent API to be used by other processes.
C) The OS kernel is always present on Disk.
D) The OS kernel is always present in Memory.
E) Any process can fetch and execute OS code in user mode.

Clicker Question

Which statement is FALSE?

A) OS manages the CPU, Memory, Devices, and Storage.
B) OS provides a consistent API to be used by other processes.
C) The OS kernel is always present on Disk.
D) The OS kernel is always present in Memory.
E) Any process can fetch and execute OS code in user mode.

November 1988: Internet Worm

Internet Worm attacks thousands of Internet hosts

Best Wikipedia quotes:

"According to its creator, the Morris worm was not written to cause damage, but to gauge the size of the Internet. The worm was released from MIT to disguise the fact that the worm originally came from Cornell."

"The worm …determined whether to invade a new computer by asking whether there was already a copy running. But just doing this would have made it trivially easy to kill: everyone could run a process that would always answer "yes". To compensate for this possibility, Morris directed the worm to copy itself even if the response is "yes" 1 out of 7 times. This level of replication proved excessive, and the worm spread rapidly, infecting some computers multiple times. Morris remarked, when he heard of the mistake, that he "should have tried it on a simulator first"."

Computer Virus TV News Report 1988

Clicker Question

Which of the following is not a viable solution to protect against a buffer overflow attack? (There are multiple answers, just pick one of them.)

(A) Prohibit the execution of anything stored on the Stack.
(B) Randomize the starting location of the Stack.
(C) Use only library code that requires a buffer length to make sure it doesn't overflow.
(D) Write only to buffers on the OS Stack where they will be protected.
(E) Compile the executable with the highest level of optimization flags.

Inside the SYSCALL instruction

`ECALL is s SYSCALL in RISC-V`

`ECALL` instruction does an atomic jump to a controlled location (i.e. RISC-V 0x8000 0180)

- Switches the sp to the kernel stack
- Saves the old (user) SP value
- Saves the old (user) PC value (= return address)
- Saves the old privilege mode
- Sets the new privilege mode to 1
- Sets the new PC to the kernel syscall handler

Inside the SYSCALL implementation

Kernel system call handler carries out the desired system call

- Saves callee-save registers
- Examines the syscall ecall number
- Checks arguments for sanity
- Performs operation
- Stores result in a0
- Restores callee-save registers
- Performs a “supervisor exception return” (***SRET***) instruction, which restores the privilege mode, SP and PC

Takeaway

- It is necessary to have a privileged (kernel) mode to enable the Operating System (OS):
  - provides isolation between processes
  - protects shared resources
  - provides safe control transfer

Outline for Today

- How do we protect processes from one another?
  - Skype should not crash Chrome.
  - Operating System
- How do we protect the operating system (OS) from other processes?
  - Chrome should not crash the computer!
  - Privileged Mode
- How does the CPU and OS (software) handle exceptional conditions?
  - Division by 0, Page Fault, Syscall, etc.
  - Traps, System calls, Exceptions, Interrupts

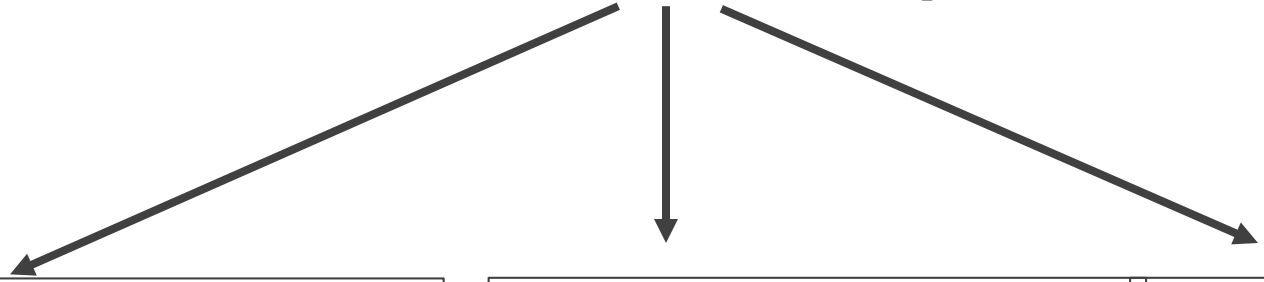
Exceptional Control Flow

Anything that *isn't* a user program executing its own user-level instructions.

System Calls:

- just one type of exceptional control flow
- Process requesting a service from the OS
- Intentional – *it's in the executable!*

Software Exceptions



Trap

Intentional

Examples:

System call

(OS performs service*)*

Breakpoint traps

Privileged instructions

Fault

Unintentional but

Possibly recoverable

Examples:

Division by zero

Page fault

Abort

Unintentional

Not recoverable

Examples:

Parity error

*One of **many** ontology / terminology trees.*

Hardware support for exceptions

SEPC register

- *Supervisor Exception Program Counter* or SEPC
- 32-bit register, holds addr of affected instruction
- Syscall case: Address of ECALL

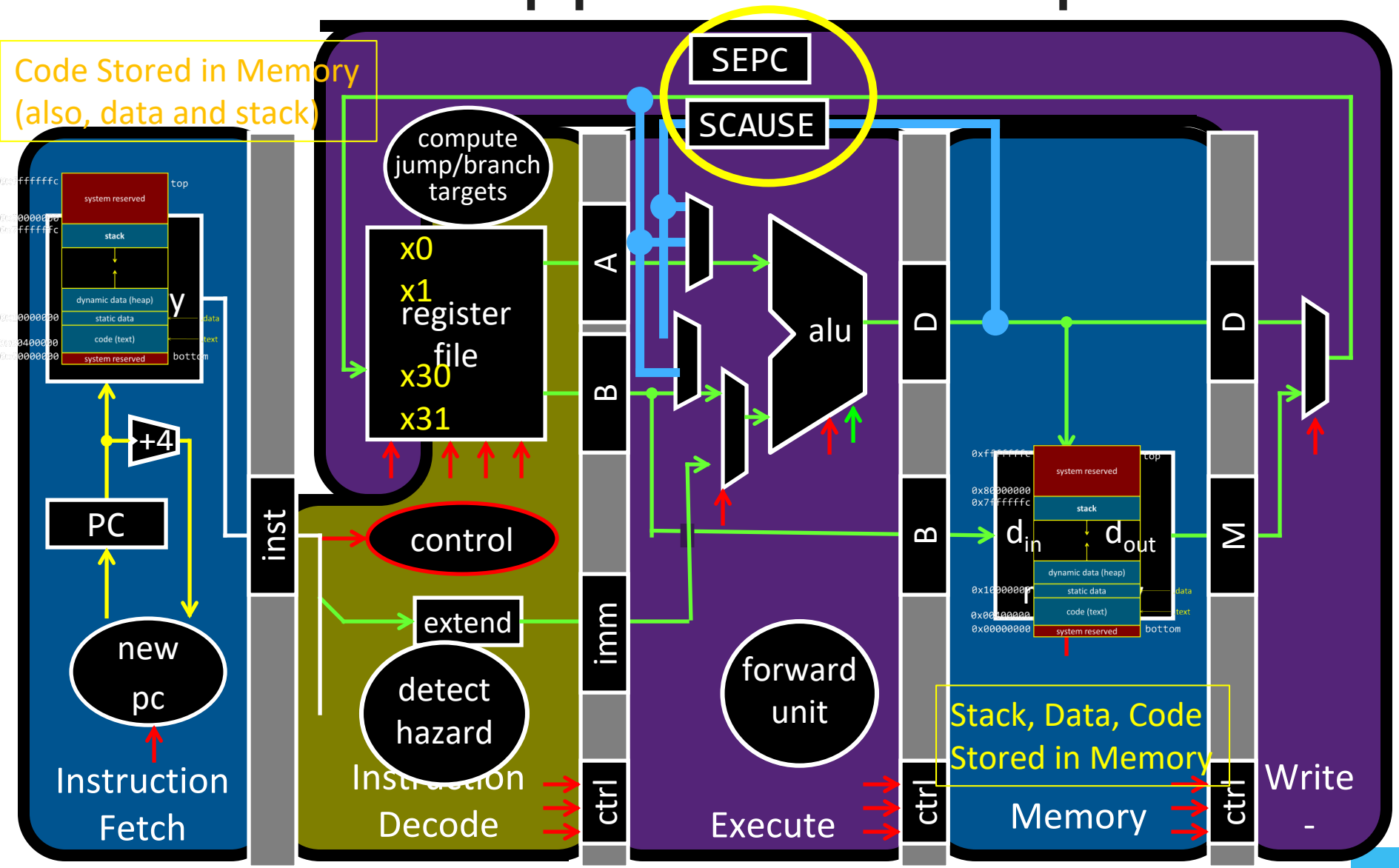
SCAUSE register

- *Supervisor Exception Cause Register* or SCAUSE
- Register to hold the cause of the exception
- Syscall case: 8, ECALL

Special instructions to load TLB

- Only do-able by kernel

Hardware support for exceptions



Hardware support for exceptions

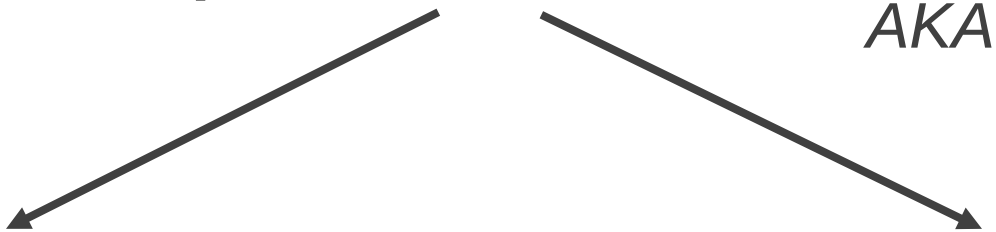
Precise exceptions: Hardware guarantees (similar to a branch)

- Previous instructions complete
- Later instructions are flushed
- SEPC and SCAUSE register are set
- Jump to prearranged address in OS
- When you come back, restart instruction

- Disable exceptions while responding to one
  - Otherwise can overwrite SEPC and SCAUSE

Exceptional Control Flow

AKA Exceptions



Hardware interrupts

Asynchronous

= caused by events external to CPU

Software exceptions

Synchronous

= caused by CPU executing an instruction

Maskable

Can be turned off by CPU

Example: alert from network device that a packet just arrived, clock notifying CPU of clock tick

Unmaskable

Cannot be ignored

Example: alert from the power supply that electricity is about to go out

Interrupts & Unanticipated Exceptions

No ECALL instruction. Hardware steps in:

- Saves PC of supervisor exception instruction (SEPC)
- Saves cause of the interrupt/privilege (Cause register)
- Switches the sp to the kernel stack
- Saves the old (user) SP value
- Saves the old (user) PC value
- Saves the old privilege mode
- Sets the new privilege mode to 1
- Sets the new PC to the kernel ~~syscall hander~~ interrupt/exception handler

SYSCAL

Inside Interrupts & Unanticipated Exceptions

interrupt/exception handler handles event

Kernel ~~system call handler~~ carries out ~~system call~~

all

- Saves ~~callee-save~~ registers
- Examines the ~~syscall number~~ cause
- ~~Checks arguments for sanity~~
- Performs operation
- ~~Stores result in a0~~ all
- Restores ~~callee-save~~ registers
- Performs a SRET instruction (restores the privilege mode, SP and PC)

Clicker Question

What else requires both Hardware and Software?

A) Virtual to Physical Address Translation
B) Branching and Jumping
C) Clearing the contents of a register
D) Pipelining instructions in the CPU
E) What are we even talking about?

Clicker Question

What else requires both Hardware and Software?

A) Virtual to Physical Address Translation
B) Branching and Jumping
C) Clearing the contents of a register
D) Pipelining instructions in the CPU
E) What are we even talking about?

Address Translation: HW/SW Division of Labor

Virtual → physical address translation!

Hardware

- has a concept of operating in physical or virtual mode
- helps manage the TLB
- raises page faults
- keeps Page Table Base Register (PTBR) and ProcessID

Software/OS

- manages Page Table storage
- handles Page Faults
- updates Dirty and Reference bits in the Page Tables
- keeps TLB valid on context switch:
  - Flush TLB when new process runs (x86)
  - Store process id (RISC-V)

Demand Paging on RISC-V

1. TLB miss
2. Trap to kernel
3. Walk Page Table
4. Find page is invalid
5. Convert virtual address to page + offset
6. Allocate page frame
   - Evict page if needed
7. Initiate disk block read into page frame
8. Disk interrupt when DMA complete
9. Mark page as valid
10. Load TLB entry
11. Resume process at faulting instruction
12. Execute instruction

Summary

Trap

- Any kind of a control transfer to the OS

Syscall

- Synchronous, process-initiated control transfer from user to the OS to obtain service from the OS
- e.g. SYSCALL

Exception

- Synchronous, process-initiated control transfer from user to the OS in

- e.g. Divide by zero, TLB miss, Page fault

Interrupt

- Asynchronous, device-initiated control transfer from user to the OS
- e.g. Network packet, I/O complete