A Tour of
Computer Systems

Instructor: Linyi Li

Slides adapted from Dr. B. Fraser

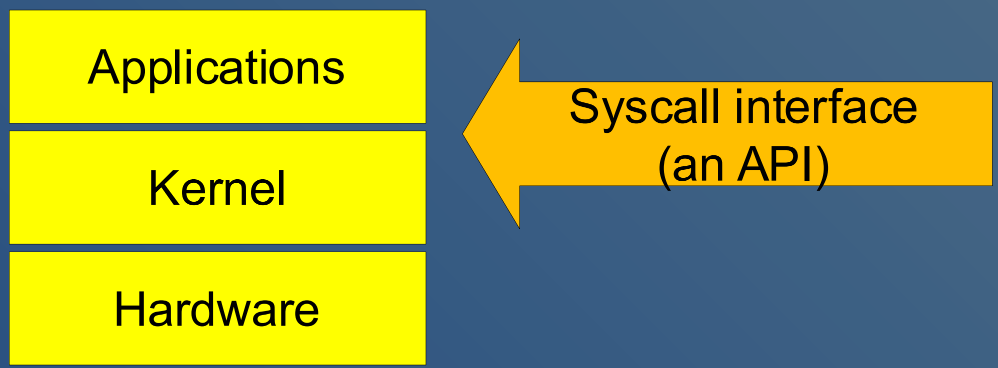
Topics

1) For a program to run, what is needed?
2) How does a computer's hardware work?
3) What does the OS Kernel do?
4) How does a program interact with the OS?

Systems Programming

OS Stack

- Let's discuss the *terminology* necessary for the course and generally for computer systems.
- OS Stack
  - .. Layers of services, each building on lower layer



OS Stack

CMPT 201 deals extensively with the syscall interface

Systems Programming

- Systems programming: ...

  – Low-level languages (e.g., C, C++, Rust) give you the ability to do systems programming, e.g., .. raw memory access.

  (Python and Java don't allow you to do that)

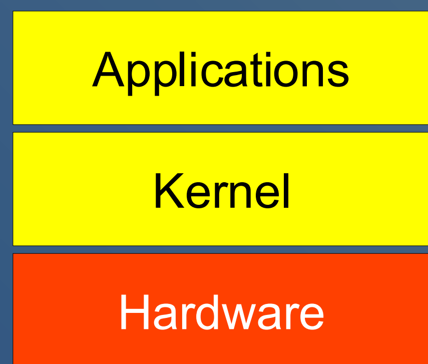
- Higher-level programs

  – Don't typically need a systems programming language, unless it needs high performance.

  – Choose a language that fits the target program's goals.

- Let's look at stack bottom up.

Hardware Layer



OS Stack

Components in Computing

- 2 Fundamental Components in Computing:
  - ..

Handled by the CPU

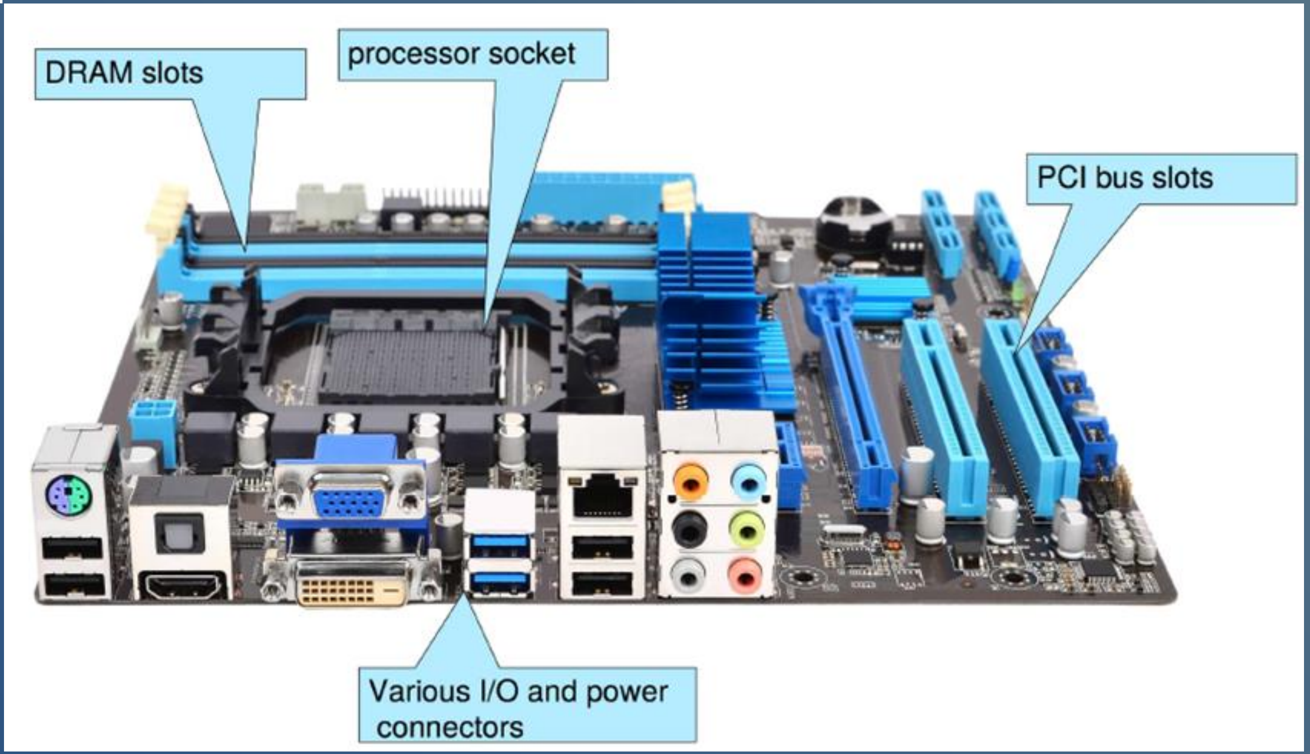
  - ..

Handled by memory (main memory (RAM) and storage)

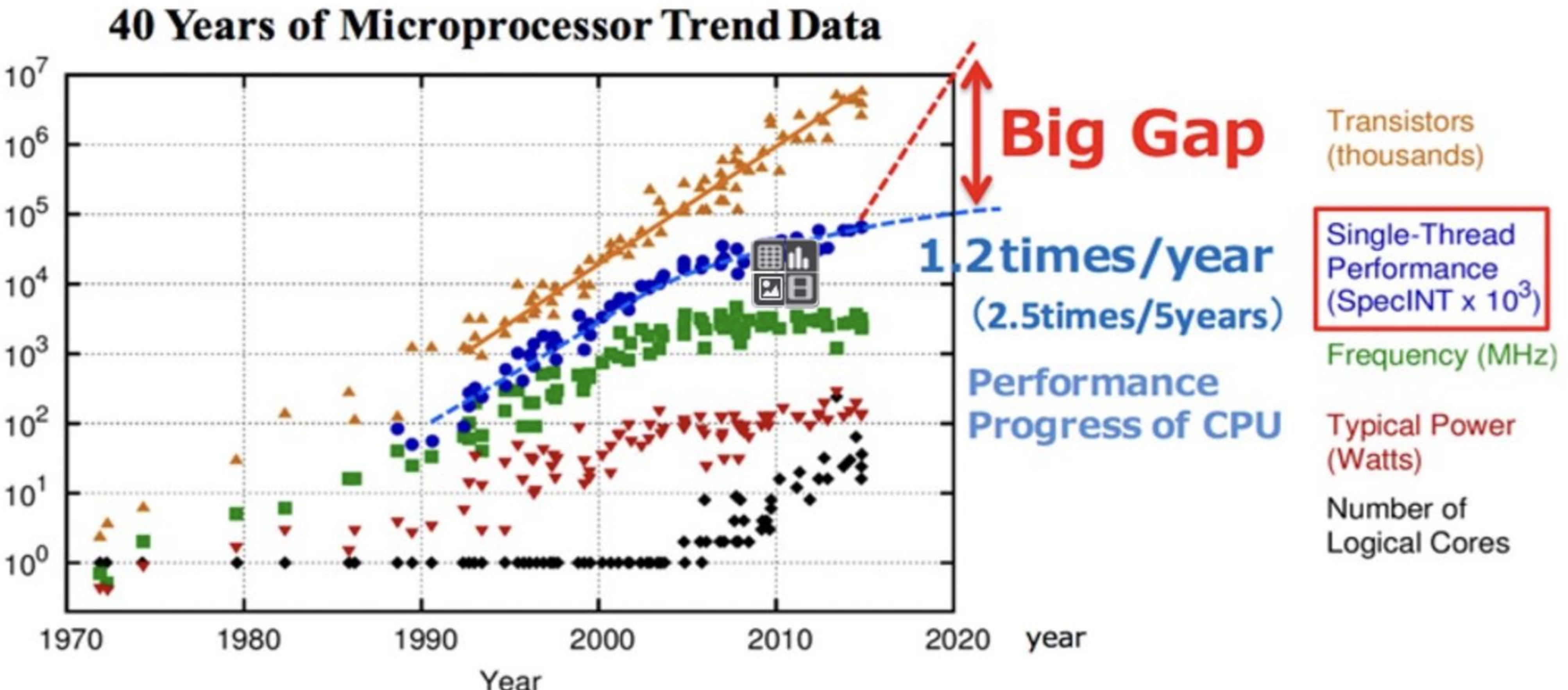
- E.g., a + b => c
  - What is the computation?
  - What is the data?

PC Motherboard

- Von Neumann architecture
  - Current fundamental model of computer design.
  - Fetch data from memory to provide to the CPU.
- Hardware components: CPU, memory, and I/O devices.



Evolution of CPU: Moore's Law



Pre early 2000: *frequency* x 2 every 18 months

Post 2005: *core count* x 2 every 18 months

Reference: Ahmet Ceyhan, Interconnects for Future Technology Generations: CMOS with Copper/Low-κ and Beyond, PhD Thesis, 2018

Evolution of Memory

- CPU needs data from memory
  - CPU was getting faster,
    so memory access had to get faster too.
  - Speed of memory access limited by
    .. memory chip speed, and speed of light!
  - Memory is far away from CPU, and much too slow

CPU

RAM

CPU vs Memory Speed

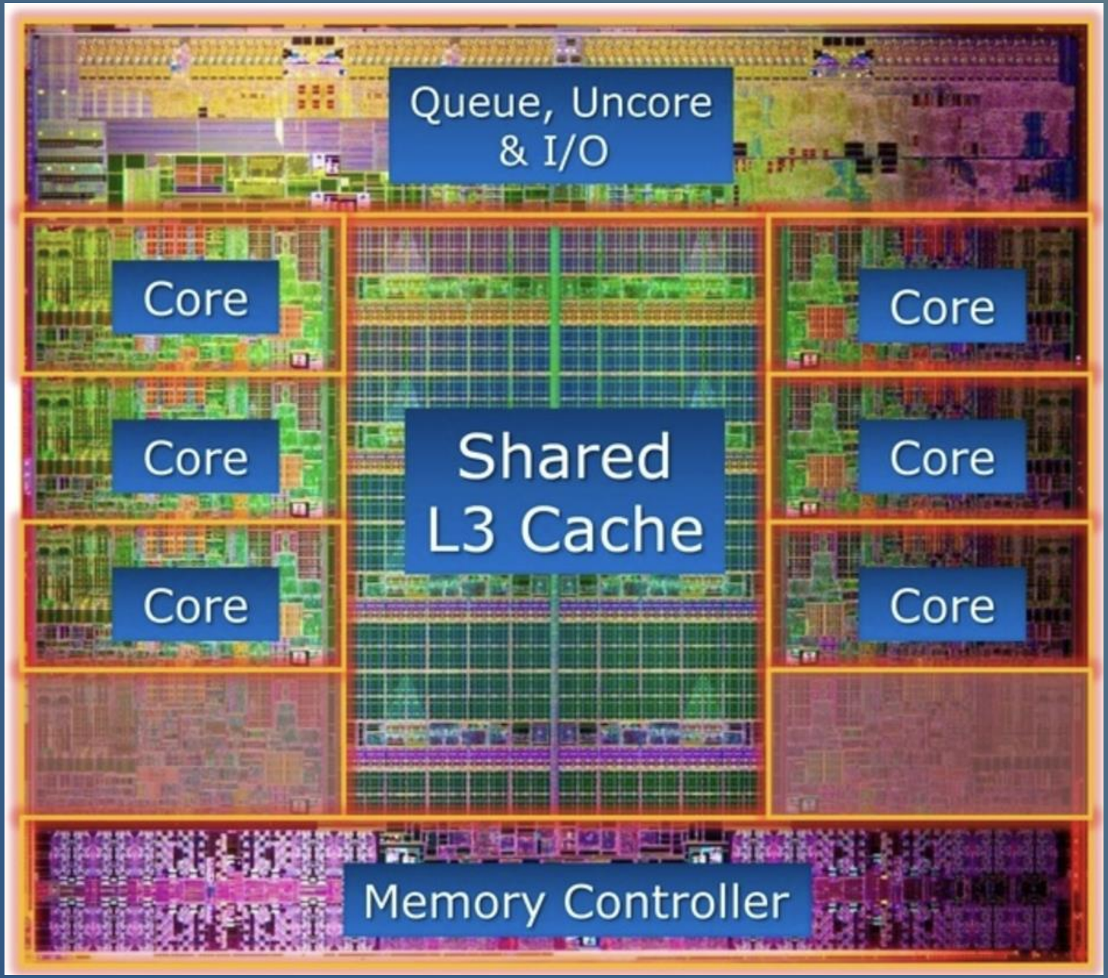
- "Solve" speed gap between CPU and memory access
  - .. very small memory inside a CPU;
  hold data items from memory.
  Very close to CPU, so very fast access to data

- Add cache
  - Much larger in size than registers,
  but much smaller than memory.
  - Quite close (physical distance) to CPU,
  so.. faster access times.
  - Nowadays processors have many caches:

L1 cache	~512 KB	(smallest, closest, fastest)
L2 cache	~8MB	
L3 cache	~32MB	(large, slowest)

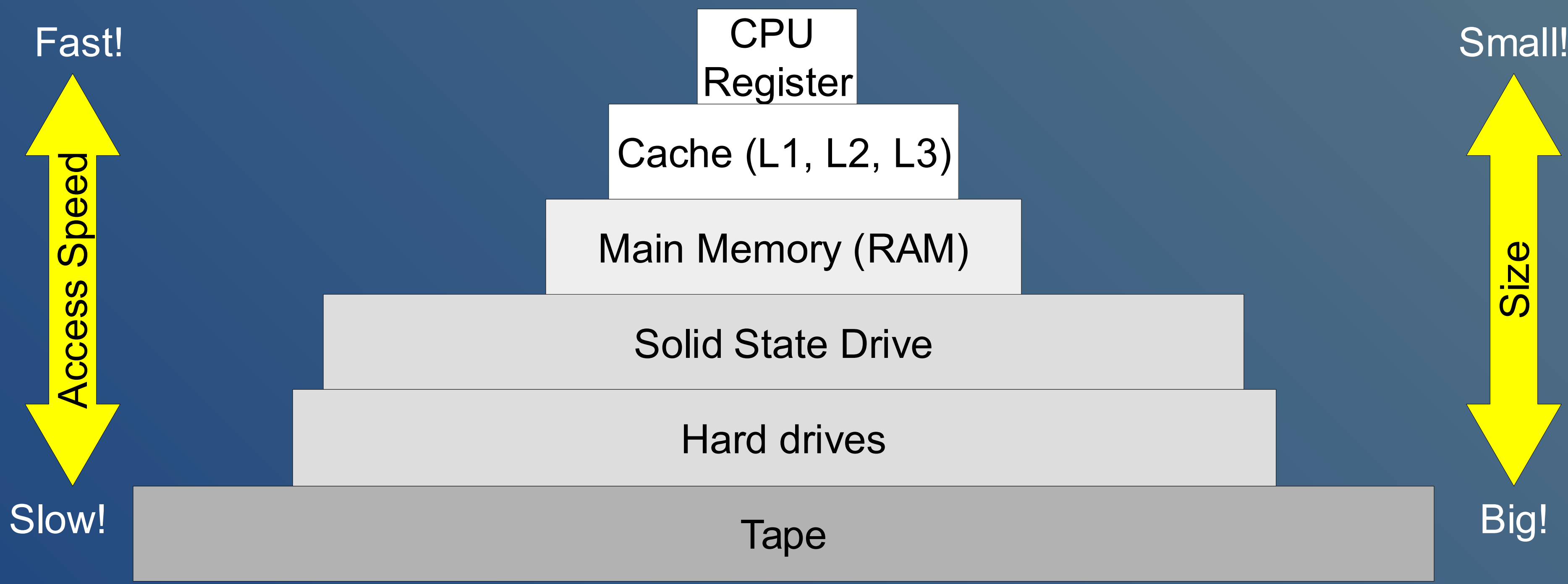
Multi-core Processor

- Desktop CPU today
  - One processor chip
  - Multiple Cores
  - Shared & private caches



Memory Hierarchy

- We want the CPU to feel like it has access to..
  a huge amount of (cheap) fast memory.
  - Intelligently bring data in from large-slow devices (hard drives) into small-fast devices (memory, cache).



Memory Hierarchy

- Trade-offs
  - ..
    Bigger size typically means more expensive (size correlates with price).
  - ..
    faster means closer to CPU.
  - ..
    "Commit" means moving data from memory to disk; i.e., changing state of data from temporary to permanent.
    - e.g., `git commit`.
  - ..
    SSD vs. HDD vs. tape: SSD's fastest but least reliable. A tape is slowest but most reliable and lasts longer.

CPU Architectures

- Instruction Set Architectures (ISA)
  - ..
  - Compiler translates C programs into machine instructions.
  - E.g. ISAs: x86, ARM, RISK-V ("risk-five")
- 32-bit vs. 64-bit architectures
  - For CMPT 201, we care most about 32-bit vs 64-bit because it..

ABCD - Pointers

- What is a pointer in your C program?

a) A memory address.
b) A variable storing a memory address.
c) The data stored in an array.
d) The address of the current instruction.

- Which of the following is true about the following code?

```
char* pLetter;
long long* pCounter;
```

a) sizeof(pLetter) < sizeof(pCounter)
b) sizeof(pLetter) > sizeof(pCounter)
c) sizeof(pLetter) == sizeof(pCounter)
d) Depends on if the system is 32-bit or 64-bit

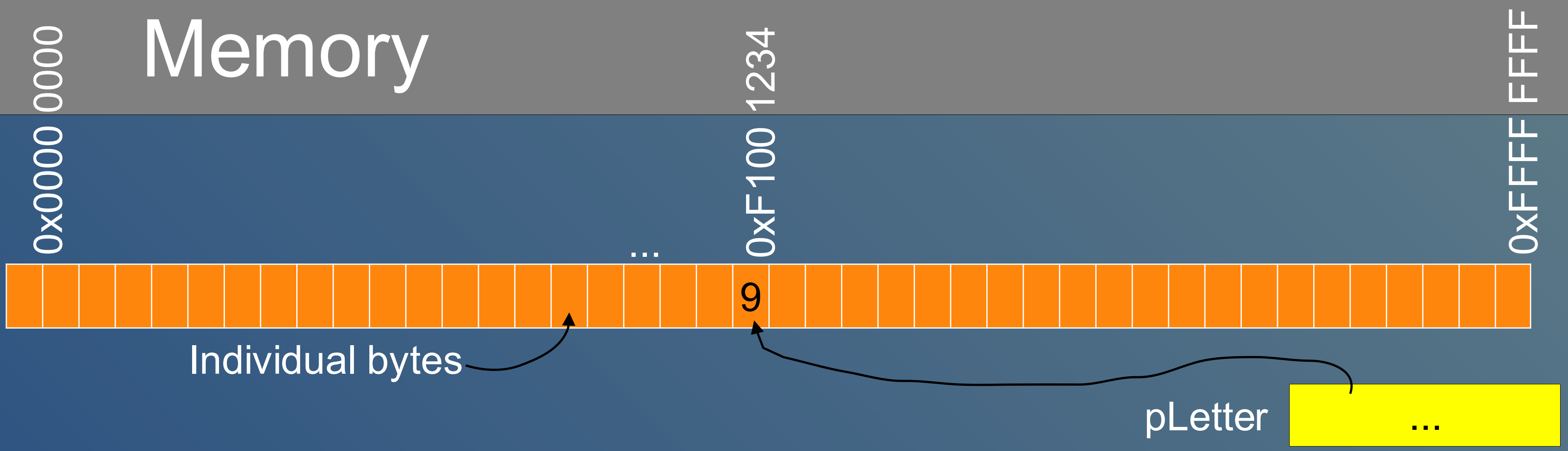
32 vs 64 bit Register Size Implications

- Big Computations:
  In 32-bit, can do 64-bit computation in multiple operations.
- ..
  (32-bit uses 32-bit pointers & 64-bit uses 64-bit pointers).

- .. Address space size
  Pointer size controls the memory address space size
- Bus Width / Memory Channel Width
  Pointer size affects # physical wires connecting to memory.
  - With 64-bits:

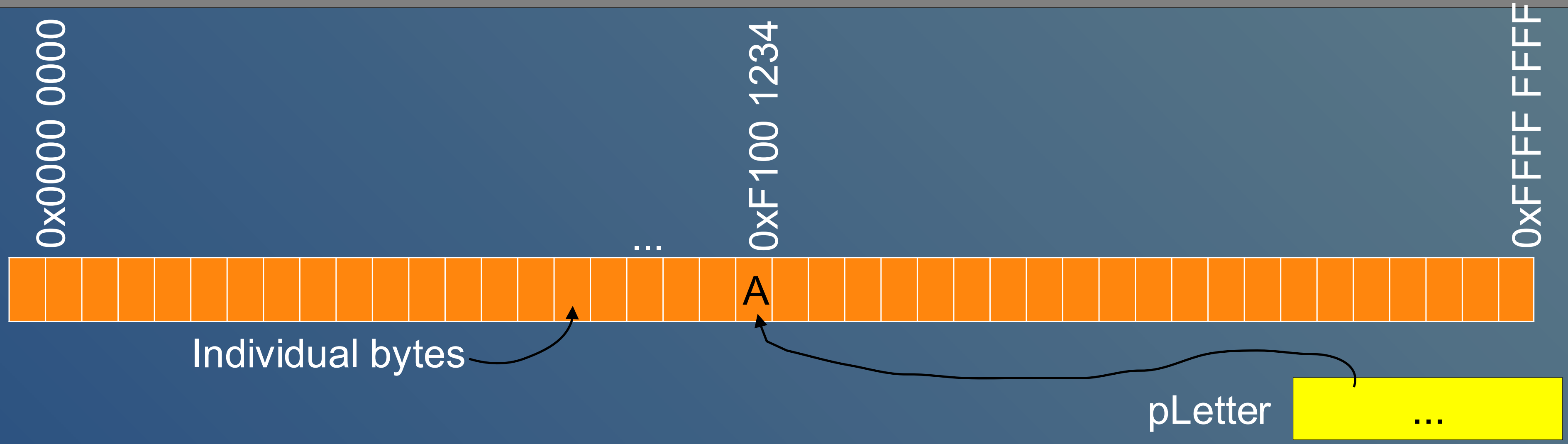
need 64 wires to transfer address from CPU to memory.
need 64 wires to transfer data from memory back to CPU

Memory



- Memory made up of bytes (1 byte = 8 bits).
  - ..
- 32-bit vs 64-bit Word Size
  - The number of bits stored in a CPU's register.
- In a 32-bit system (32-bit word):
  - Addresses are 32-bits:
    0x0000 0000 to 0xFFFF FFFF
  - (Data is retrieved from memory 32-bits at a time (4 bytes) but memory addresses are still byte addresses)

ABCD: Pointer Values



- Which of the following is true?

```
char ch = 'A';
char* pLetter = &ch
```

a) pLetter == ‘A’
b) pLetter == 0x0000 000A
c) pLetter == 0xF100 1230
d) pLetter == 0xF100 1234

ABCD - Memory

- Which of the following is true?

a) 1,000 = MB, 1,000,000 = KB, 1,000,000,000 = GB
b) 1,000 = GB, 1,000,000 = MB, 1,000,000,000 = KB
c) 1,000 = KB, 1,000,000 = MB, 1,000,000,000 = GB
d) 1,000 = GB, 1,000,000 = KB, 1,000,000,000 = MB

B < KB < MB < GB < TB

- If memory (RAM) stored just 16 bytes (16 locations), how many bits do we need in our address?

a) 2-bits
b) 4-bits
c) 8-bits
d) 16-bits

Why 64-bits?

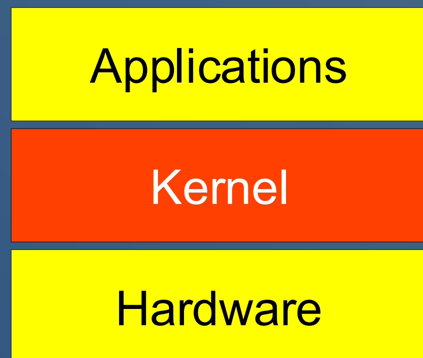
- Why are most computers 64-bit architectures now?
  - Has a 64-bit register
  - Has a 64-bit pointer
  - Allows us to.. address 2^{64} different bytes in memory.

  2^{64}= 16,000,000,000 GB = 16 Exabytes (VERY large)

- In a 32-bit architecture, how much memory can the CPU access?

  a) 65,526 bytes
  b) 2,147,483,648 bytes
  c) 4,294,967,296 bytes
  d) 18,446,744,073,709,551,616 bytes

Kernel Layer



OS Stack

What is the OS?

- Operating System (OS)

  ..

- OS Includes:

  - ..

  Main part that actively manages resources.

  - Supporting tools:

  such as GUI, command line;
  These are what differentiates Linux distributions (“distros”)

What does a Kernel do?

- Kernel's Role
  - .. Resource management
    - many programs want to access the hardware at the same time
    - kernel manages (mediates) access
  - ..

the kernel controls programs (running, stopping, etc.).

  - .. Protection

the kernel provides protection (isolation) for users and programs.

- E.g., users can't access each other's data
- E.g., programs can't interfere with each other's execution.

Event-Driven

- When does a kernel do some work?
  - Generally, the OS lets other programs run and waits for something it needs to do.
  - The kernels is.. event driven:
    It responds to events.
- Events can be:
  - .. Hardware interrupts
    a hardware event like a mouse click, or network packet received
  - ..
    a user-space-application generated call to the kernel
    e.g., application asking kernel to printf() to the screen.
  - .. Signals
    a software interrupt that announces an event to a process
    e.g., SIGINT = ctrl+c, SIGSEGV = segmentation (page) fault

User Mode vs. Kernel Mode

- Privilege mode of CPU execution
  - Kernel Mode runs the OS kernel;
    allows full privilege and full access to the hardware.

    Often called "Ring 0"
  - User Mode runs applications;
    ..

    E.g., instructions that allow direct access to hardware

    E.g., access to certain regions of memory (kernel memory)
- Modern CPUs run in one of those two modes at a given moment.
- ABCD: Which best explains why we need a user mode?

(a) Isolation

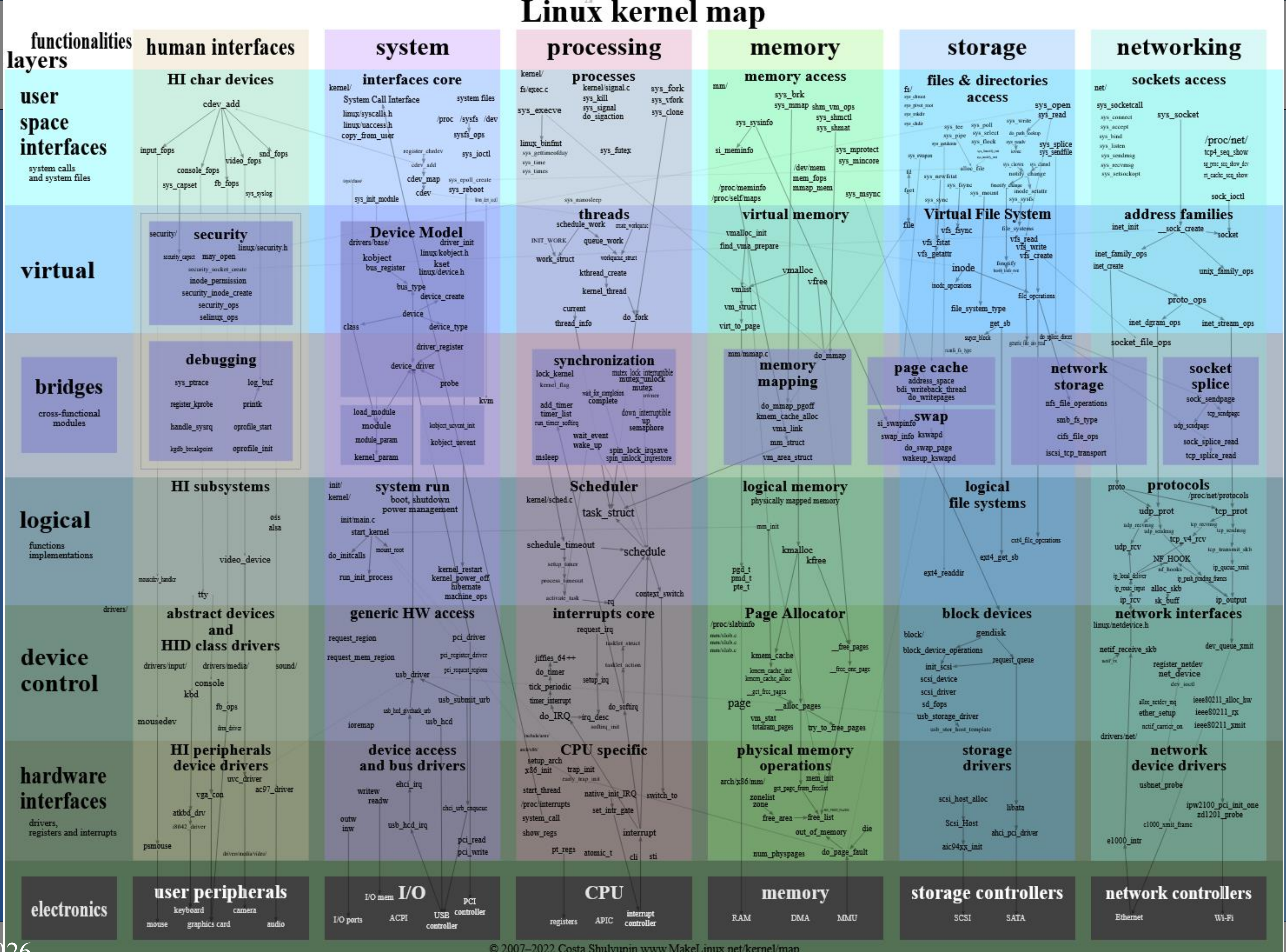
(b) Efficiency

(c) Null pointers

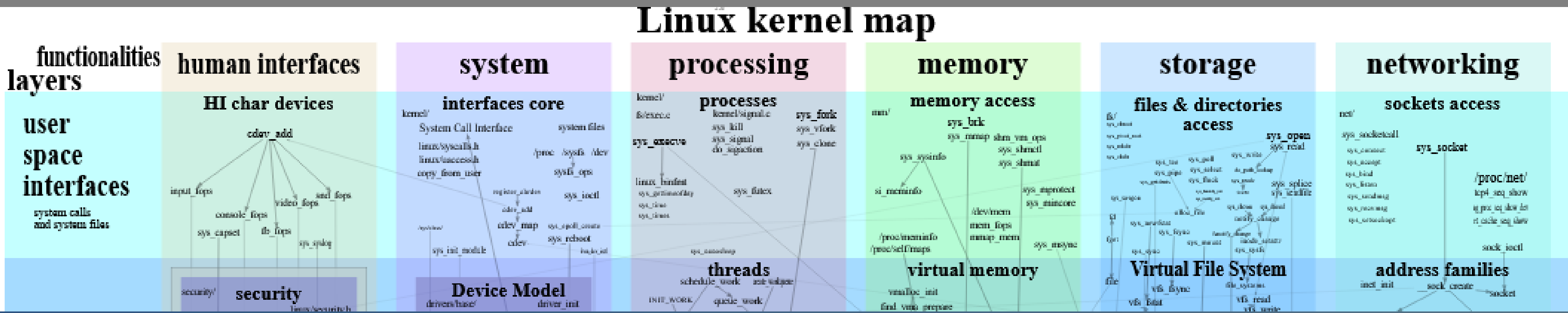
(d) Abstraction

Root user (aside)

- User / Kernel Mode vs Root User
  - The “mode” (privilege level of code) is different than the user-level
  - The root user is still a user, but with full admin privileges
    - Root can run programs and access files that normal users cannot.
    - Root user often called a super user.
  - Root user cannot access kernel memory or protected instructions.


Linux kernel map
functionalities
layers
human interfaces
system
processing
memory
storage
networking
user space interfaces
system calls and system files
HI char devices
interfaces core
processes
memory access
files & directories access
sockets access
virtual
security
Device Model
threads
virtual memory
Virtual File System
address families
bridges
cross-functional modules
debugging
synchronization
memory mapping
page cache
network storage
socket splice
swap
logical
functions implementations
HI subsystems
system run
boot, shutdown power management
Scheduler
logical memory
logical file systems
protocols
device control
abstract devices and HID class drivers
generic HW access
interrupts core
Page Allocator
block devices
network interfaces
hardware interfaces
drivers, registers and interrupts
HI peripherals device drivers
device access and bus drivers
CPU specific
physical memory operations
storage drivers
network device drivers
electronics
user peripherals
I/O
CPU
memory
storage controllers
network controllers
© 2007–2022 Costa Shulyupin www.MakeLinux.net/kernel/map

Important Terms in the Kernel



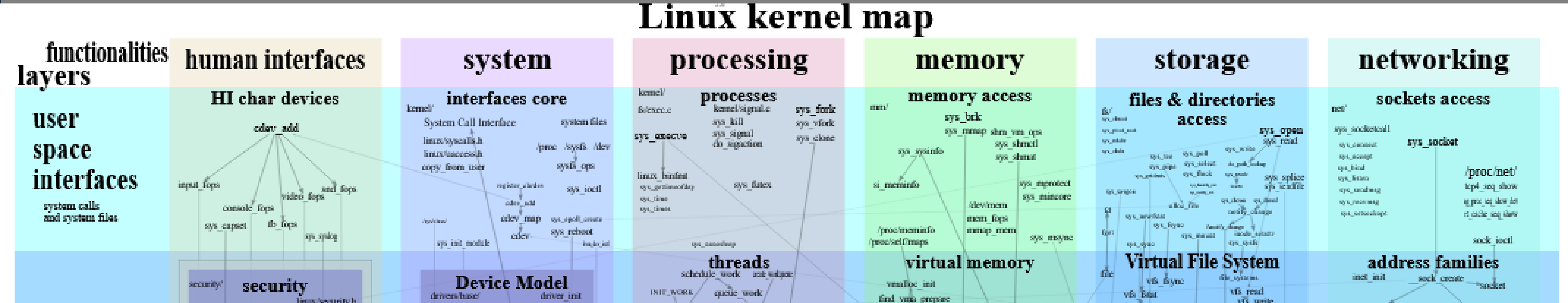
- System
  - Device drivers: every device needs a device driver to control it. E.g., network card device driver talks to hardware to send/receive data to/from the physical network.
- Processing
  - Processes, threads, synchronization, and scheduling

Covered later

- Memory
  - Virtual memory, physical memory, and paging

Covered later

Important Terms in the Kernel (cont)



- Storage
  - File systems, and VFS (Virtual File System).

    VFS is an interface:

    ·· data structures and operations that a file system should support e.g., read and write.
  - By looking like a normal file, many tools can seamlessly work with it e.g., `cat /proc/cpuinfo`

Covered later

- Networking
  - Sockets, TCP, UDP, and IP

ABCD - Kernel

- Which of the following is true?

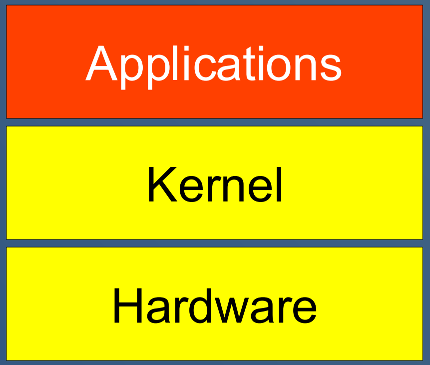
(a) The root user runs programs in kernel mode.

(b) Syscalls allow the kernel to execute user-level applications.

(c) A hardware interrupt is generated when dereferencing a null pointer.

(d) User mode prevents applications from executing privileged instructions.

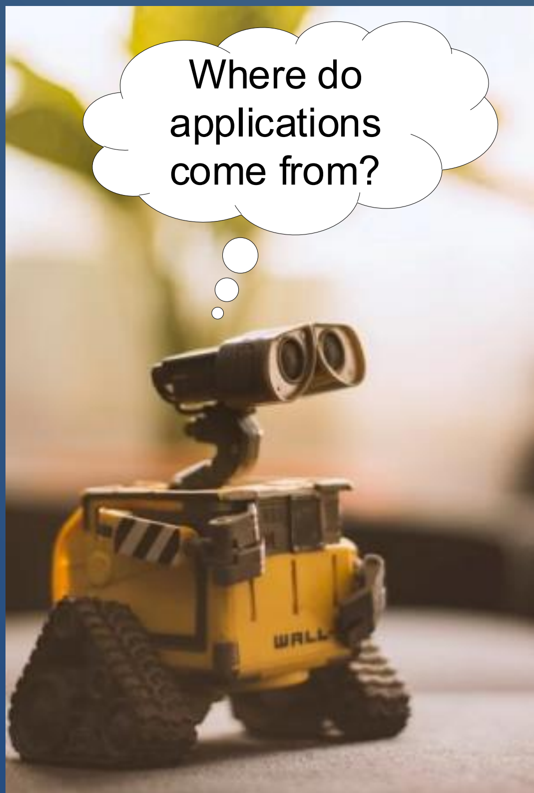
Applications Layer

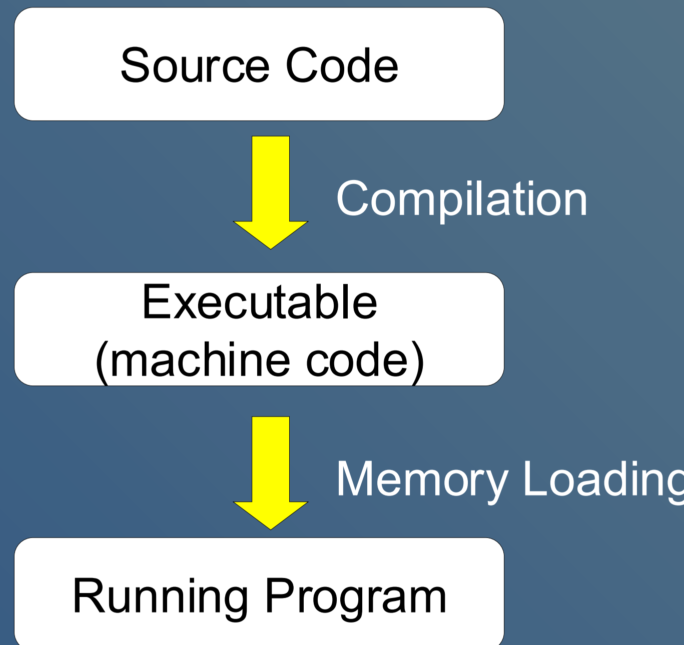


OS Stack

Lifetime of a Program

(briefly)





Compilation vs. Interpretation

(briefly)

- Two major ways to run a program:
  - Compilation (e.g., C, C++)
  - Interpretation (e.g., Python, Bash)

- Performance vs Portability Trade-off
  - Compilation has better performance:

    it directly generates machine code to execute.
  - ..

    machine code for one specific ISA

    E.g., can't run x86 executable on ARM machine
  - Interpretation is slower, but same script can run anywhere there is an interpreter.

Intermediate Representation

(briefly)

- Intermediate Representation (IR)
  - Java bytecode, LLVM bitcode: architecture-neutral ISAs.

  low-level instructions similar to x86 or ARM instructions but they do not target specific CPUs.

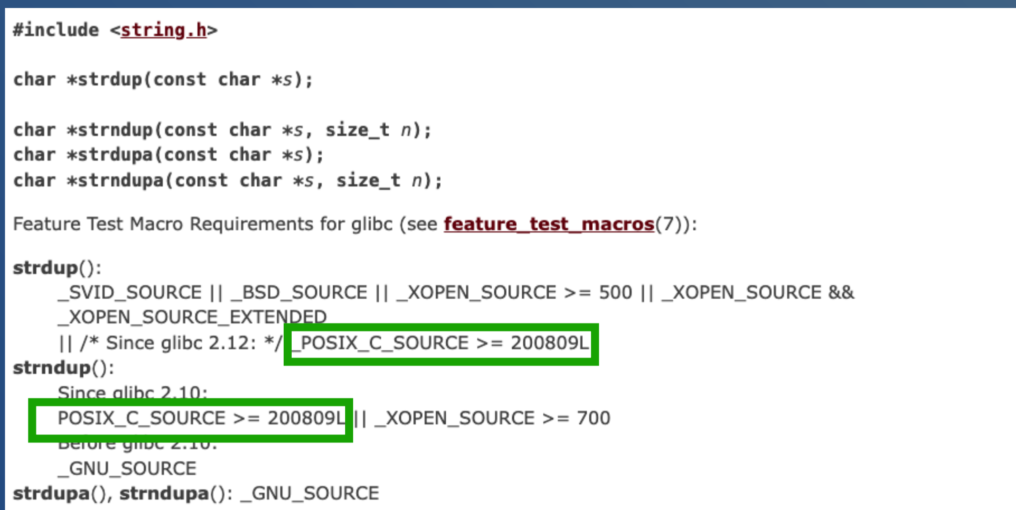
- Steps to using IR
- 1. Compile source code to low-level IR instructions
- 2. Use a backend compiler to compile IR down to an architecture-specific executable
- Rust and Go compilers generate portable LLVM bitcode (in IR), and then use LLVM backend compiler to generate machine code for specific ISA

POSIX

(briefly)

- POSIX = ..
  - A standard for (user-level) software portability across different OSs.
  - Includes programming interface (file I/O, C standard library, etc.) and shell utilities
  - We see it in C to: specifies what features we need:

#define _POSIX_C_SOURCE 200809L



```
#include <string.h>

char *strdup(const char *s);

char *strndup(const char *s, size_t n);
char *strdupa(const char *s);
char *strndupa(const char *s, size_t n);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

strdup():
    _SVID_SOURCE || _BSD_SOURCE || _XOPEN_SOURCE >= 500 || _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
    || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
strndup():
    Since glibc 2.10:
    POSIX_C_SOURCE >= 200809L || _XOPEN_SOURCE >= 700
    Before glibc 2.10:
    _GNU_SOURCE
strdupa(), strndupa(): _GNU_SOURCE

Image: https://www.linkedin.com/pulse/understanding-posix-standard-bridges-operating-systems-logzeta-1bl4f/

ABI

- ABI = ..
- Similar to API = .. Application Programming Interface
  - An API is at the code level:
  Your code calls or accesses the functions of the API, such as provided by a library.
  - An ABI is an interface for a binary (an executable) that an OS defines.
- Compilers generate executables that follow the ABI for the OS
  - E.g., Windows ABI is different from Linux ABI.
  Cannot copy a Windows binary (`.exe`) to a Linux machine and run it (and vice versa).

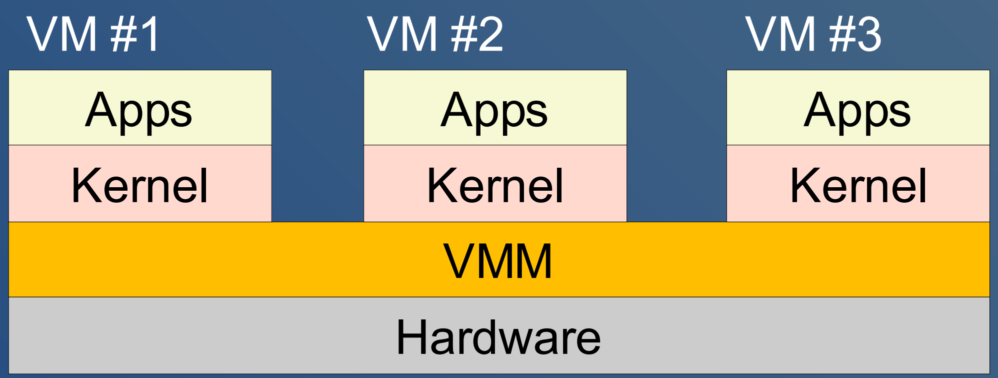
Virtualization

Virtualization of Traditional OS Stack

- Virtualization allows..
  part(s) of our OS stack to be swapped out
  - Lets us be much more flexible!
  - Software can control the environment:
    "*Spin up 3 virtual machines to host new databases*"
- .. Hypervisor:
  software that *provides* virtualization.
  - Also called the Virtual Machine Monitor (VMM)
  - Hypervisor can run at different levels of our OS stack, giving different levels of flexibility

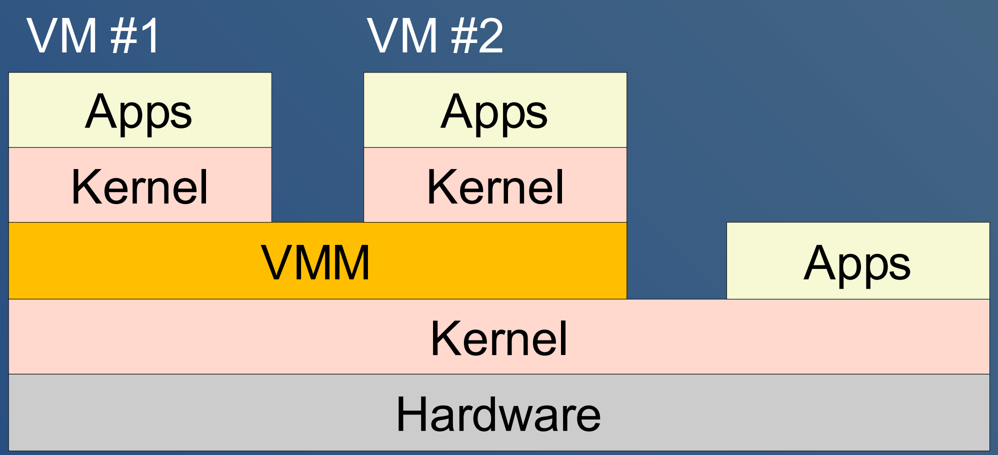
On Hardware

- VMM Directly atop Hardware
  - VMM..
  - This is often used in a data center environment.



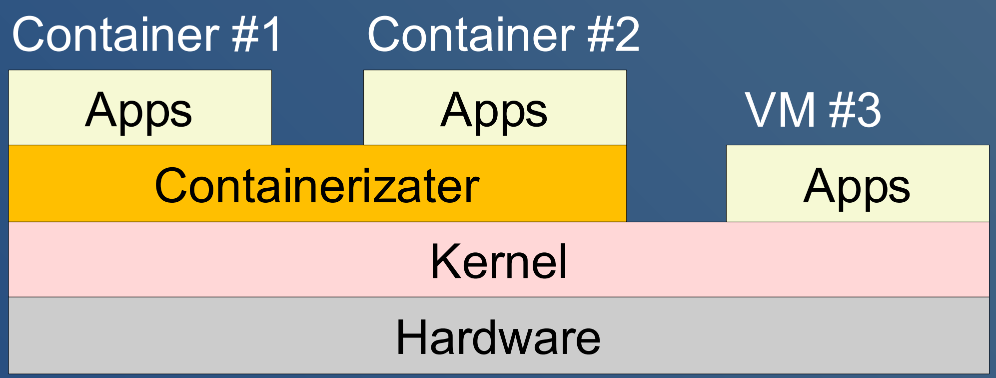
On Kernel

- VMM atop the Kernel
  - A VMM is an application running atop a kernel, along with other applications.
  - The VMM creates/runs/manages VMs.
  - This is often used in a desktop environment, e.g., VMWare Workstation, VirtualBox, QEMU.



Containerization

- Containerization
  - Containerization creates a container not a virtual machine.
  - Container includes..   an isolated set of applications and data.
  - Uses the same OS kernel as rest of the system
  - Uses Linux features for isolation: process isolation (namespaces), resource control/isolation (cgroups), etc.
  - This is the most popular form of virtualization these days, e.g., **Docker**, Podman.



ABCD - Virtualization

- Which of the following is a major benefit of virtualization?

(a) Allows user level applications to call the kernel.

(b) Allows parts of the OS stack to be swapped out under software control.

(c) Allows the kernel to control different pieces of hardware when they are connected at runtime.

(d) Allows application to run without using an OS kernel.

Summary

- OS Stack is the layers of service
  - Hardware, Kernel, Application.
- Memory hierarchy
  - allows programs to access large memories quickly
- Pointers hold addresses,
  - 32 vs 64 bits limit how much memory we can access
- Kernel mode gives OS kernel access to all resources
  - User mode limits what an application can do.
- Applications use the OS's ABI to use services
- Virtualization allows parts of the OS stack to be swapped out under software control.