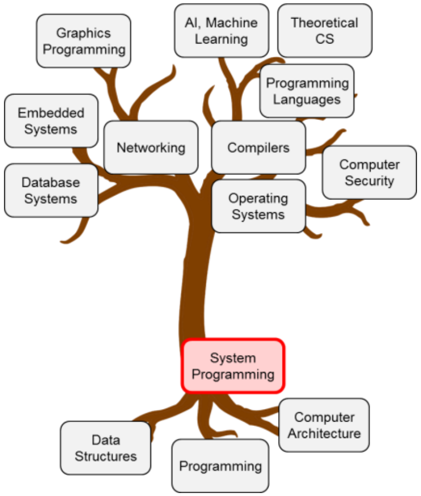
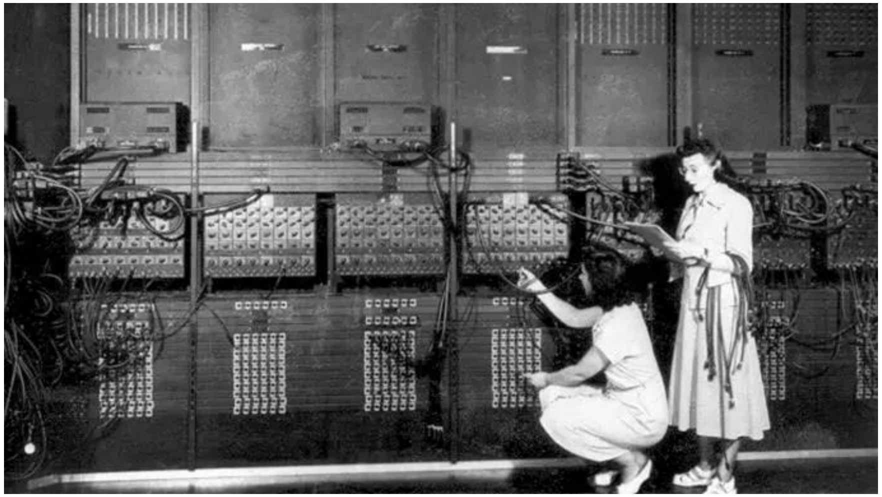
Introduction to System Programming



Module Outline

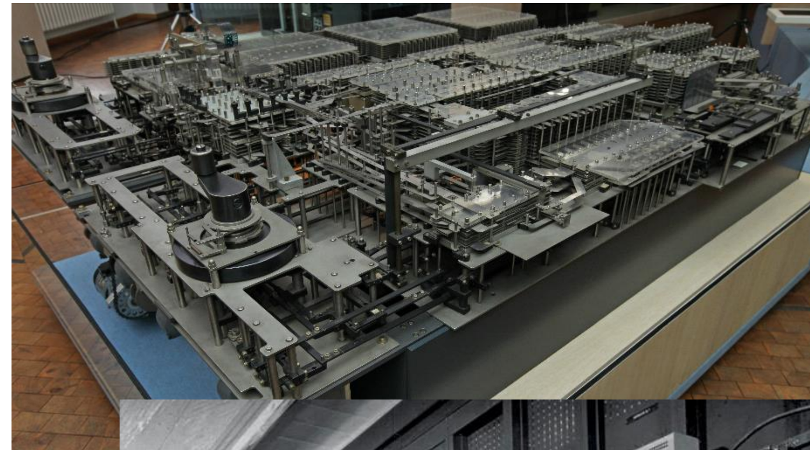
- A Brief History of Computing
- Basic Organization and Operation of a Computer System
- System Software
- System Programming
- Module Summary

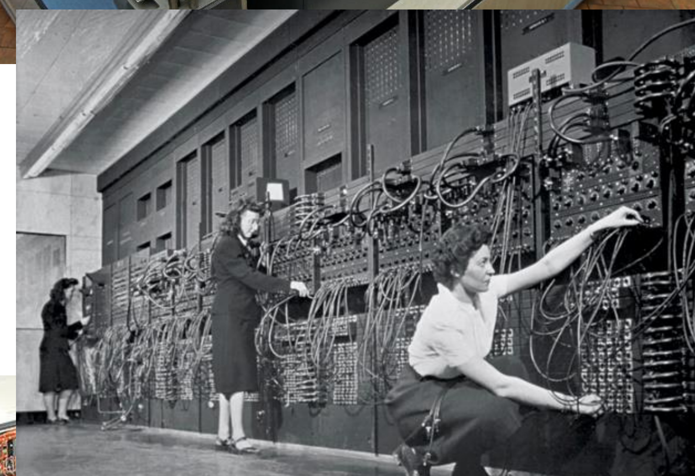


A (Very) Brief History of Computing

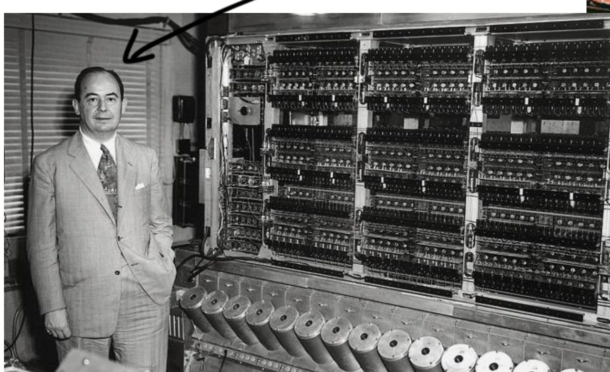
(Very) Brief History of Computing

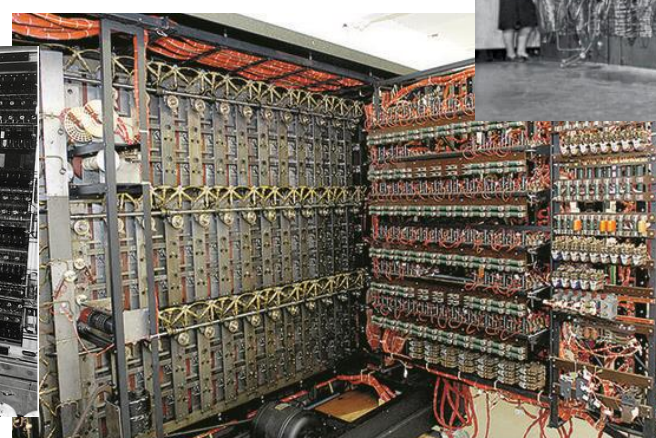
- **1940's: special-purpose computers**
  - Z1, Colossus, ENIAC
    "Programmable" by rewiring the system





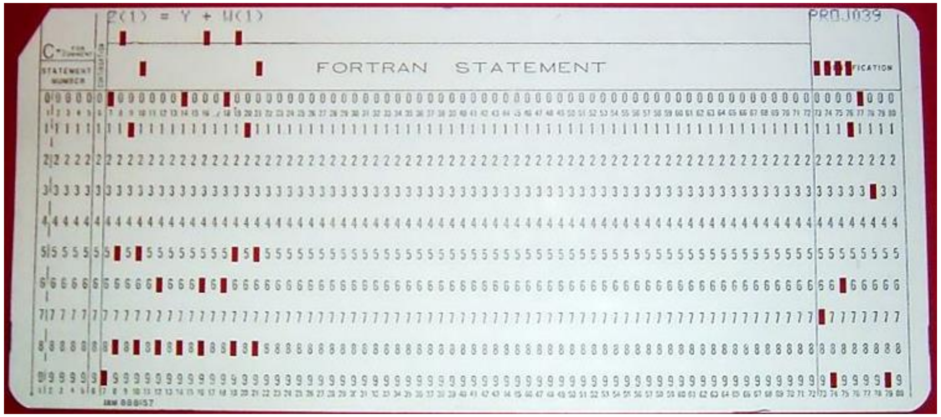
- **early 1950's: general-purpose computers**
  - EDVAC, bombe
  - programs stored in memory
  - CPU fetch-execute cycle Von Neumann
  - single program, single user at a time





(Very) Brief History of Computing

- **mid 1950's: batch programming**
  - operator combines programs into batches of programs
  - executing a batch meant executing the programs one by one
  - results available after all jobs in the batch had completed
  - "resident monitor": a first primitive version of system software
    - control card interpreter
    - loader
    - device drivers



(Very) Brief History of Computing

- **mid 1950's: batch programming (cont'd)**
  - no protection:
    a faulty job reading too many cards, over-
    writing the resident monitor's memory, or entering an
    endless loop would affect the entire system
  - lead to:
    - operating modes (user/monitor)
    - memory protection
    - execution timers

CSE 컴퓨터공학부
Department of Computer Science & Engineering

(Very) Brief History of Computing

- **early 1960's: multiprogramming**
  - more memory → keep several programs in memory at once (memory partitioning to separate the different jobs)
  - OS monitor could switch between jobs when one became idle (i.e., waiting for I/O)
  - e.g., IBM OS/360

- **mid 1960's: timesharing**
  - switch between jobs periodically
  - access via remote terminals
  - e.g., CTSS, MULTICS, UNIX

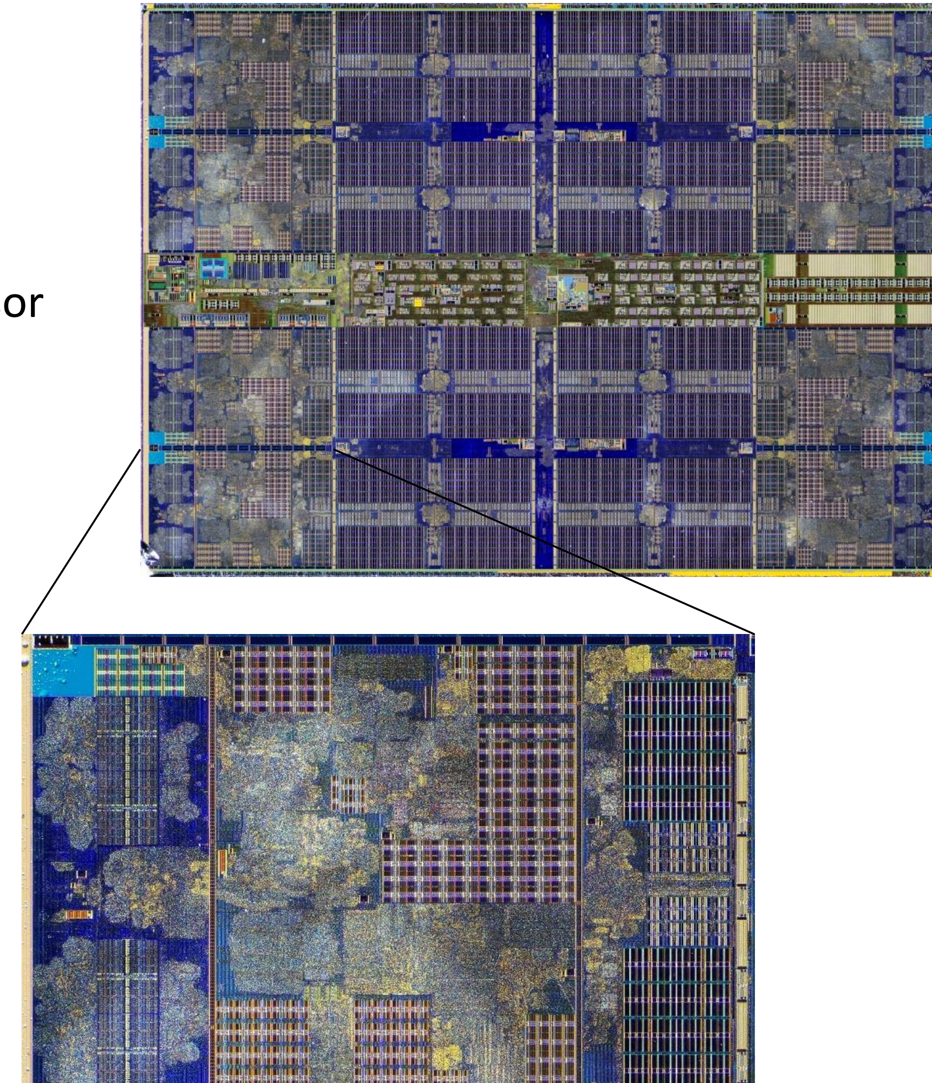
(Very) Brief History of Computing

- **late 1970's: personal computers**
  - single user, dedicated workstation
  - WIMP user interface
  - peripherals connected directly
  - single processor, time-sharing

Today

AMD Zen2 Core Complex Die

Source: Wikichip



processor(core)

- **Parallel processing**
  - highly parallel and complex CPU
    - several physical processors
    - several cores per physical processor
    - hyper-threading in a single core
  - heterogeneous cores
    - GPU, CPU, accelerators
  - large memory, fast network
  - several users, several programs

- **System software a must**

The Fastest Computer Today

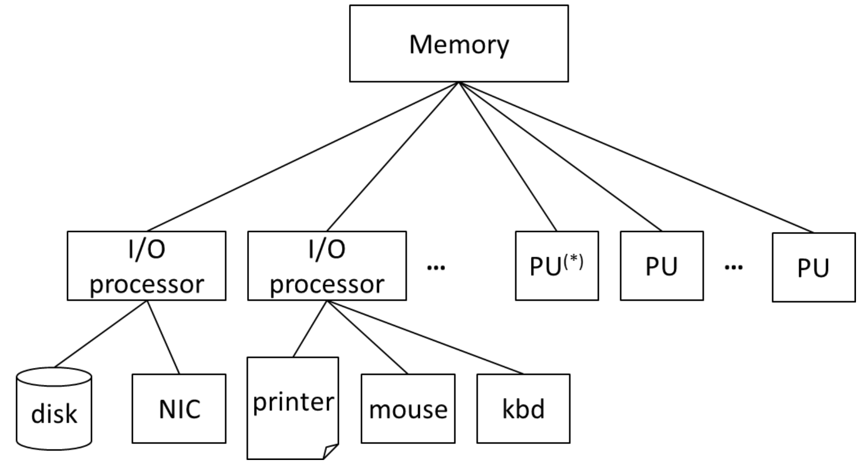
- **Frontier (Oak Ridge National Laboratory)**
  - 8,699,904 cores
  - AMD CPUs (EPYC 64C) + GPUs (MI250X)
  - 74 cabinets
  - Interconnect
    - CPU-GPU: AMD Infinity
    - system: Slingshot network

  - 1.7 EF peak performance
  - 1.2 EF Linpack
  - #1 on the top500.org list since June 2022
  - the only computer to reach one exaflop/s

The Development of Computing Power

System	Year	Speed
Z1	1938	1.00 IP/s
ENIAC	1946	5.00 kIP/s
Atlas	1962	1.00 MFLOP/s
Cray-2	1985	1.41 GFLOP/s
ASCI Red	1997	1.06 TFLOP/s
Roadrunner	2008	1.02 PFLOP/s
Frontier	2022	1.10 EFLOP/s 10^{18}

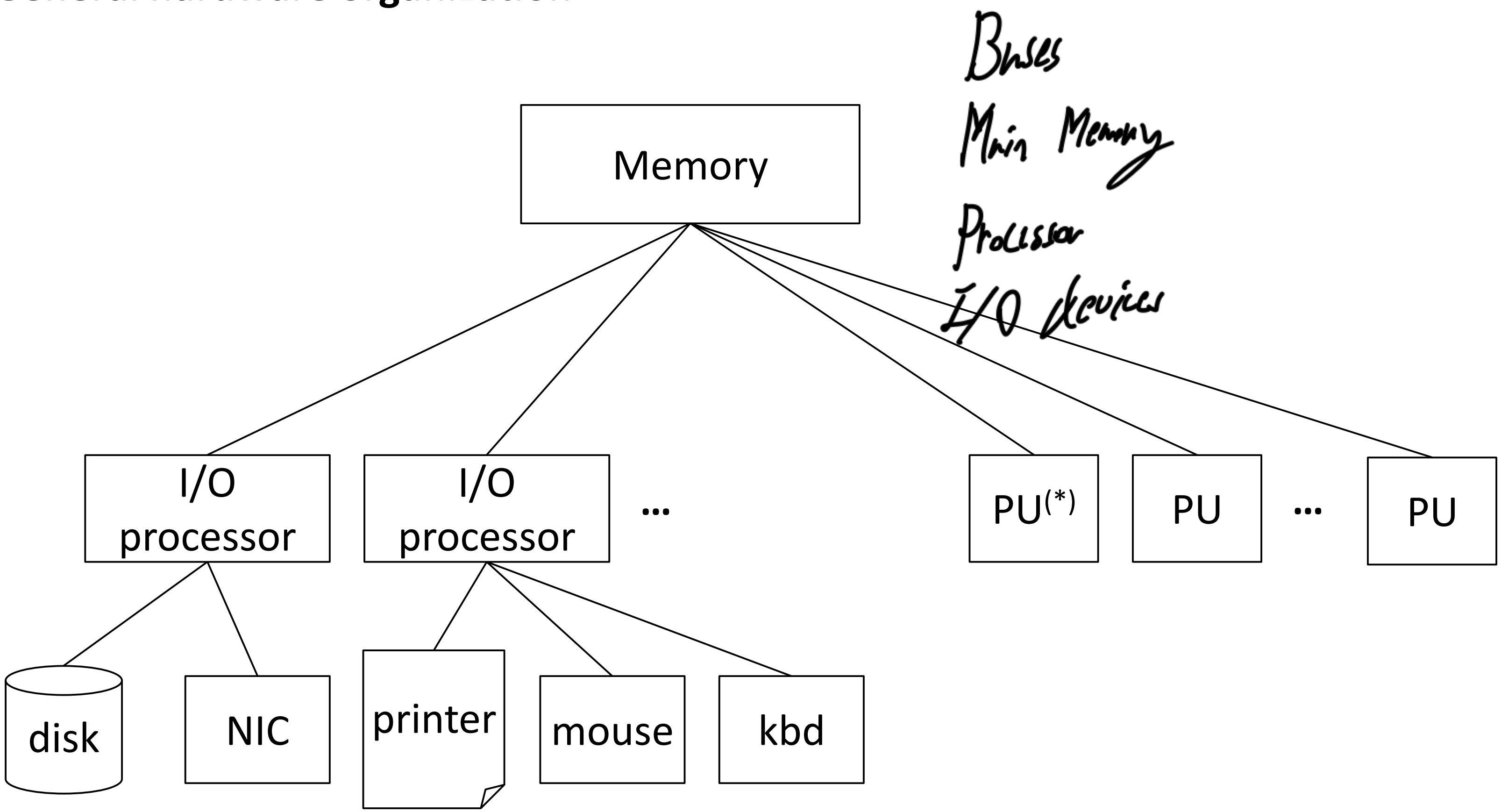
- Your smartphone provides about the same performance as the world's fastest supercomputer from 1997 – 2000, ASCI Red. Yet, it (reference: Samsung Galaxy S22),
  - requires about 25 times fewer processors to do so
  - is about 70'000 times cheaper
  - consumes about 450'000 times less power



Basic Organization and Operation Modern Computer Systems

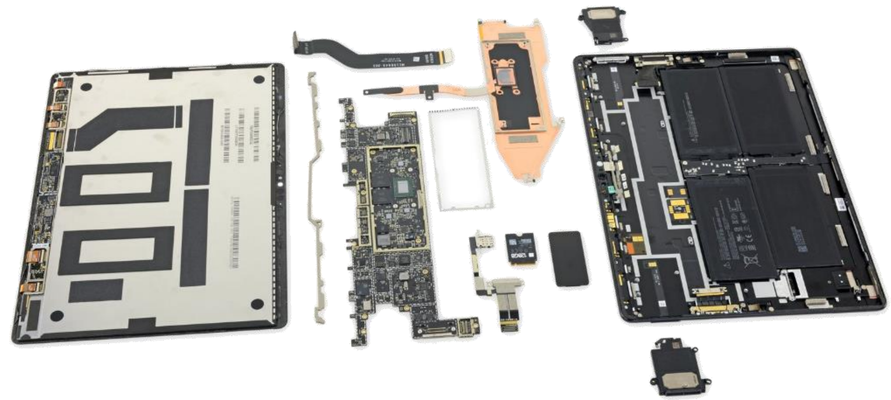
Hardware Organization

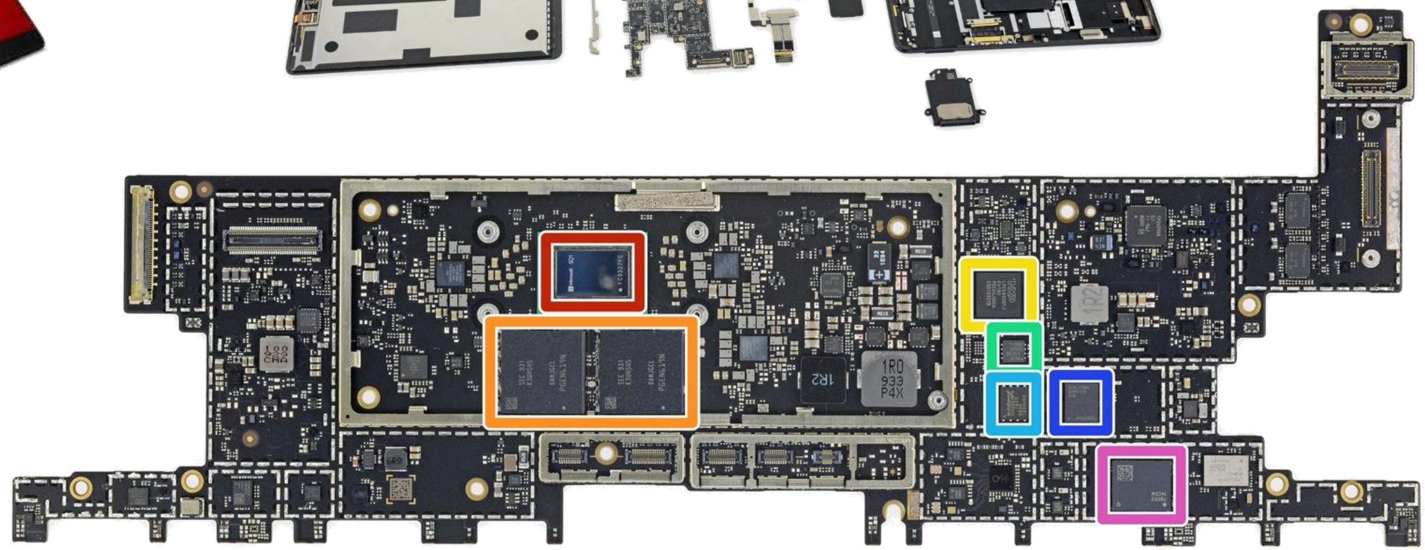
- **General hardware organization**



(*) PU = Processing Unit

Hardware Organization: Microsoft Surface Pro

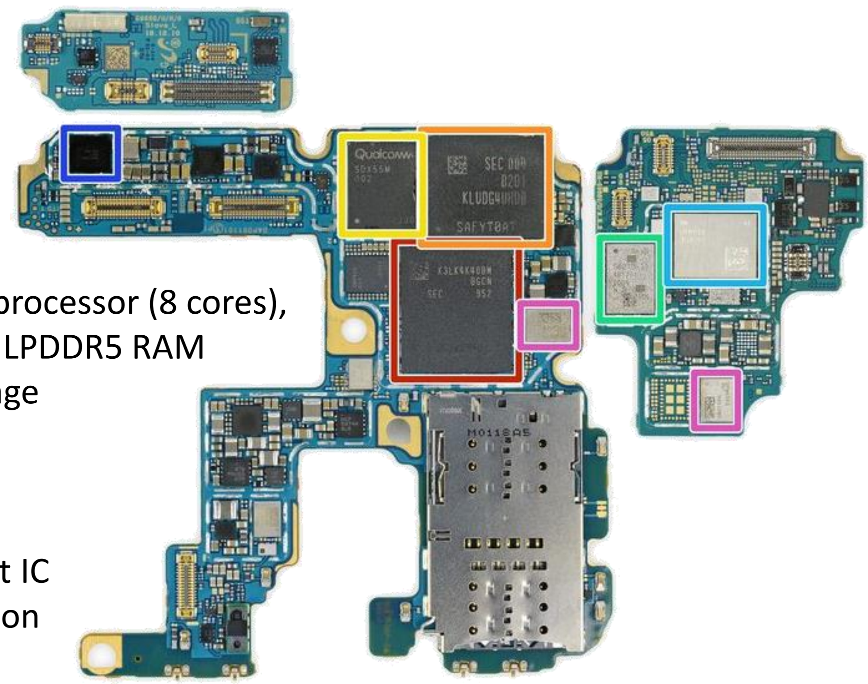




- Microsoft ARM processor
- 2x4GB Samsung LPDDR4X RAM
- NXP EV180 microcontroller
- Macronix 16Mb serial NOR flash memory
- Winbond 256 Mb serial flash memory
- Qualcom RF module
- Qorvo Wifi module

image sources: Microsoft, iFIXIT

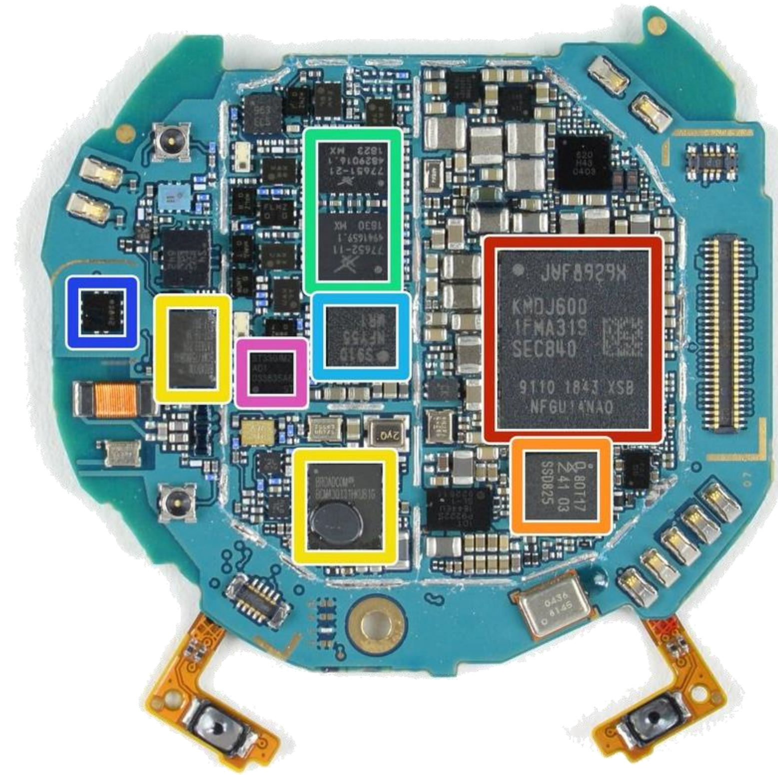
Hardware Organization: Samsung Galaxy S20 Ultra



- Qualcom Snapdragon 865 processor (8 cores), overlaid by 12GB Samsung LPDDR5 RAM
- 128GB Samsung flash storage
- Qualcomm 5G modem
- Skyworks RF module
- Qorvo Wifi module
- Maxim power management IC
- Qualcom power amplification modules

image sources: Samsung Electronics, iFIXIT

Hardware Organization: Samsung Galaxy Watch



- Samsung Exynos 9110 (dual core)
- NXP NFC module
- Broadcom Wifi/Bluetooth modules
- Skyworks power amplifiers
- STMicroelectronics barometric pressure sensor
- ST Micro 32-bit ARM SecurCore

image sources: Samsung, iFIXIT

Program Execution

- **Applications execute under the assumption they run exclusively on the hardware**
  - private memory
  - private compute resources (CPU cores, GPUs, accelerators, …)
  - uninterrupted access to peripherals (disk, network, …)

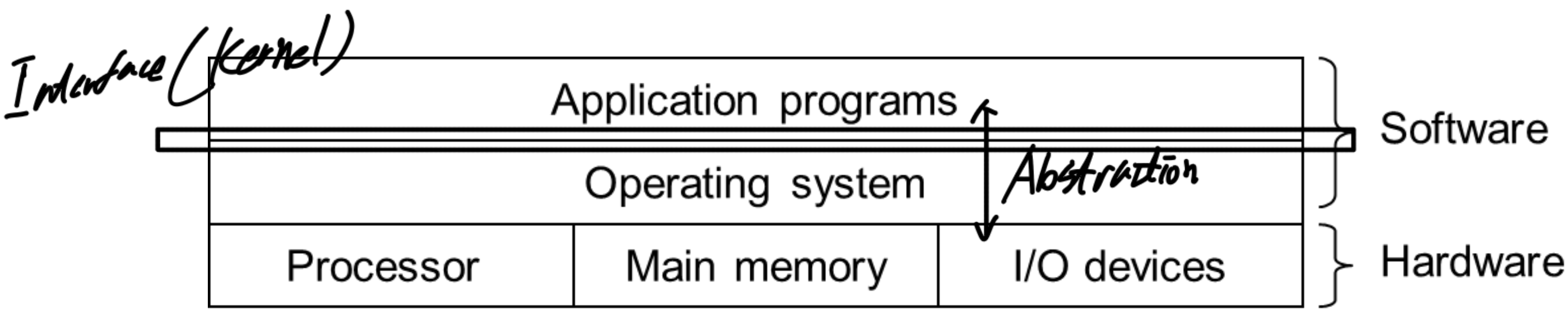
  - yet, programs may
    - run in parallel
    - be multi-threaded
    - communicate with each other
    - access shared physical resources

→ This illusion is maintained by the operating system (OS)

Virtualization

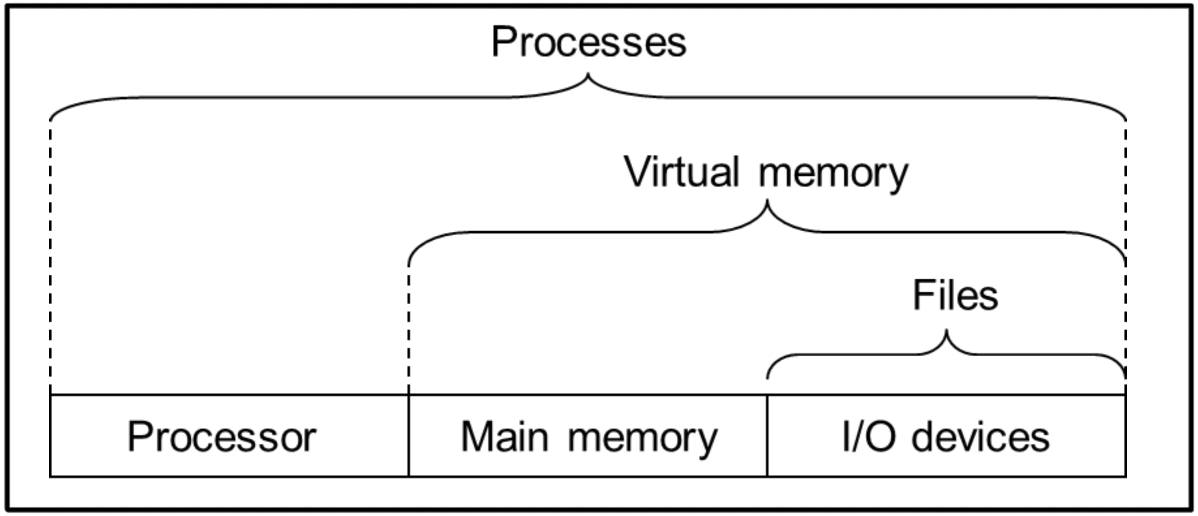
Operating System Basics

- **The operating system manages the hardware**
  - protect H/W from misuse by buggy/malicious programs
  - provide simple and uniform mechanisms for manipulating hardware devices



Operating System Basics

- **Fundamental abstractions**
  - processes
  - virtual memory
  - files



Abstraction 1: Files

- **Abstraction of physical storage**
  - sequence of bytes
  - single interface to interact with files
  - organized in a hierarchical structure (directories)

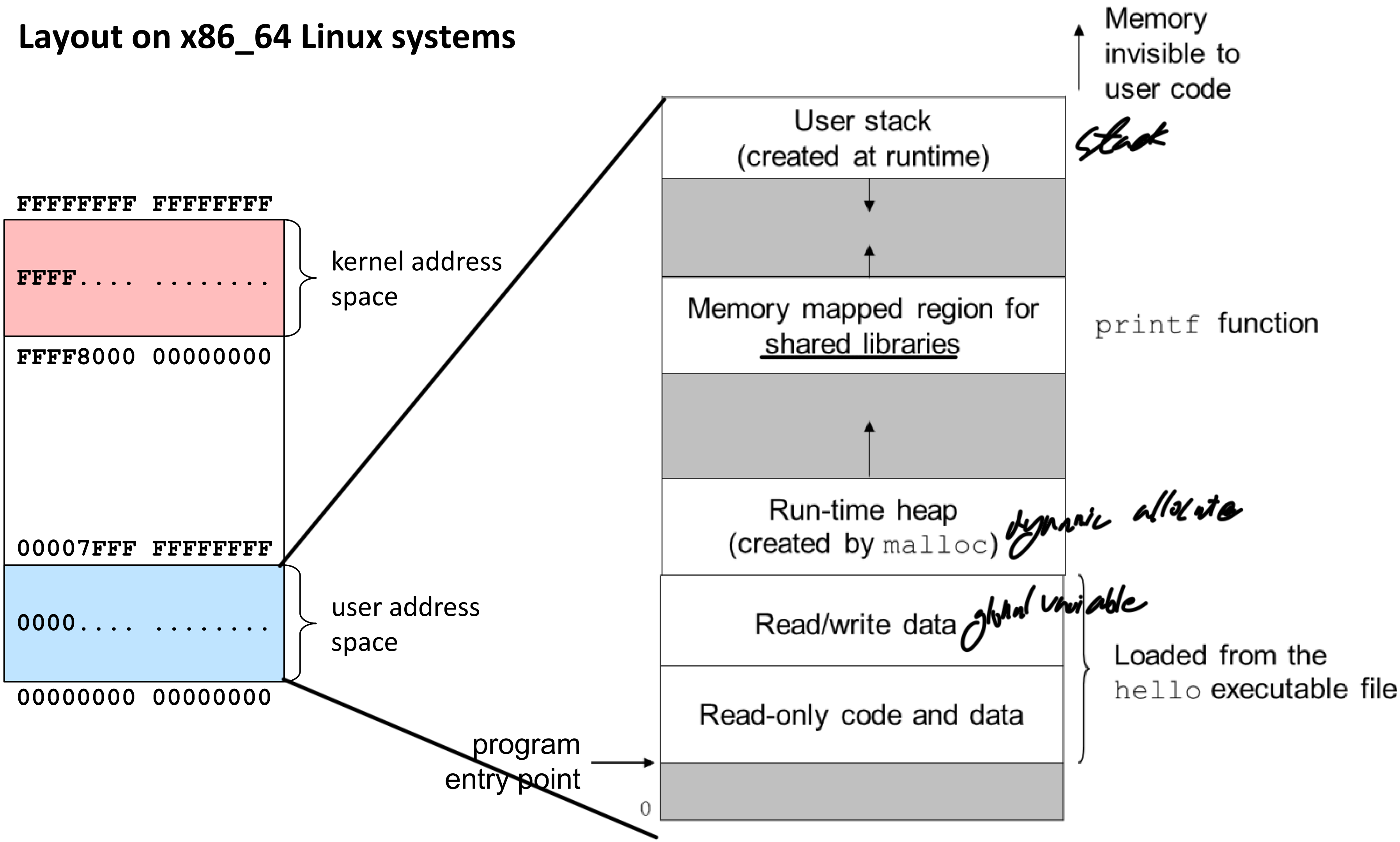
```
$ dirtree lecture/code/02/Unix
lecture/code/02/Unix/
├ dedup/
│ ├ dedup.py
│ └ strings.txt
├ dirsize/
│ ├ dirsize
│ ├ dirsize.bash.txt
│ ├ dirsize.c
│ ├ dirsize.find.sh
│ ├ dirsize.ls.sh
│ └ dirsize.py
└ dirtree/
  ├ .dirtree.c.swp
  ├ Makefile
  ├ dirtree
  └ dirtree.c
```

Abstraction 2: Virtual Memory

- **Abstraction of physical memory**
- Provides each running program with the illusion that it has exclusive use of the main memory
- Managed by the OS with the help of a hardware translation unit: the memory management unit (MMU)
- Virtual memory also provides the basis for
  - paging
  - sharing
  - mmap

Abstraction 2: Virtual Memory

- **Layout on x86_64 Linux systems**



Abstraction 2: Virtual Memory

- Layout on x86_64 Linux systems

global area (Read/Write [illegible])

code area (Rd only)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char global[1024*1024];

int main(int argc, char *argv[])
{
  char *ptr = malloc(1024*1024);

  printf("Hello, world\n");

  printf("  address of main:     %p\n", main);
  printf("  address of printf:   %p\n", printf);
  printf("  address of global:   %p\n", global);
  printf("  address of ptr:      %p\n", &ptr);
  printf("  address of mem[ptr]: %p\n", ptr);

  while (1) sleep(1);

  return EXIT_SUCCESS;
}
```

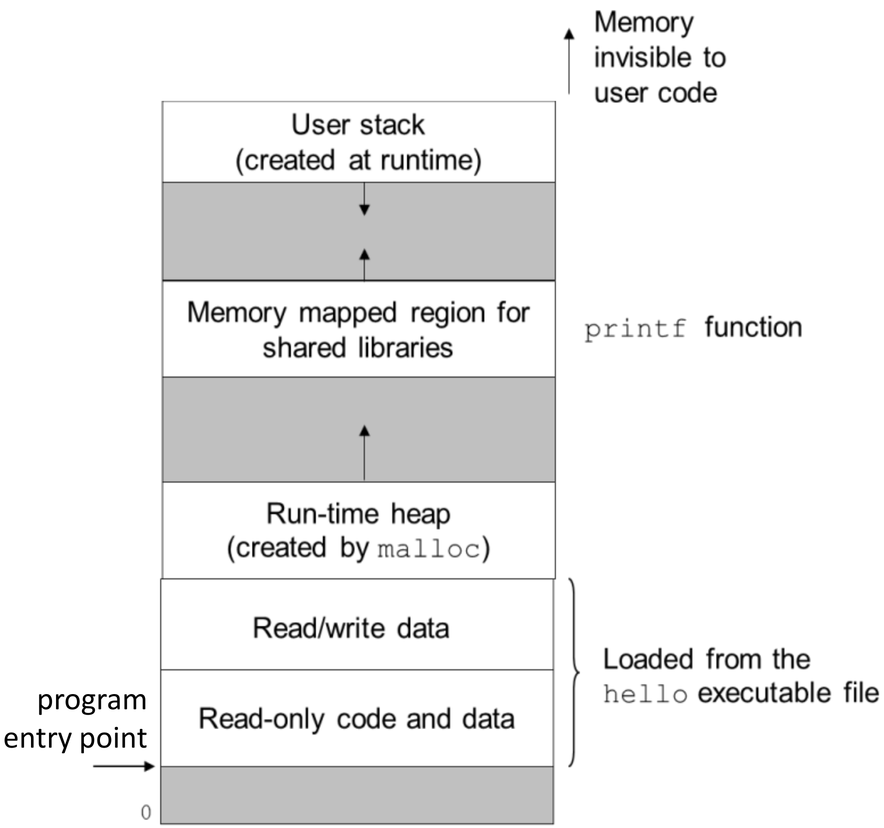
hello.c

```
$ gcc -Wall -o hello hello.c
$ ./hello
Hello, world
  address of main:       0x559f777b4155
  address of printf:     0x7fd7e0d33cf0
  address of global:     0x559f777b7060
  address of ptr:        0x7ffe605f9fd0
  address of mem[ptr]:   0x7fd7e0bd8010
^C
$ ./hello
Hello, world
  address of main:       0x55c051daf155
  address of printf:     0x7f522a055cf0
  address of global:     0x55c051db2060
  address of ptr:        0x7ffe8bee2bb0
  address of mem[ptr]:   0x7f5229efa010
^C
$
```

Change every time run

Virtual Memory

- **Layout on x86_64 Linux systems**




```
$ ./hello
Hello, world
Hello, world
  address of main:        0x55dbeb167155
  address of printf:      0x7f1eb5d5bcf0
  address of global:      0x55dbeb16a060
  address of ptr:         0x7fff566ff4c0
  address of mem[ptr]:    0x7f1eb5c00010
^Z
[1]+  Stopped                 ./hello
$ bg
[1]+ ./hello &
$ ps
  PID TTY          TIME CMD
 7225 pts/4    00:00:00 bash
 7375 pts/4    00:00:00 hello
 7398 pts/4    00:00:00 ps
$ pmap 7375
7375:   ./hello
000055dbeb166000      4K r---- hello
…
000055dbeb16a000      4K rw--- hello
000055dbeb16b000   1024K rw---   [ anon ]
000055dbecc28000    132K rw---   [ anon ]
00007f1eb5c00000   1040K rw---   [ anon ]
00007f1eb5d04000    160K r---- libc.so.6
00007f1eb5d2c000   1448K r-x-- libc.so.6
…
00007f1eb5f13000      8K rw---   [ anon ]
00007f1eb5f15000      8K r---- ld-linux-x86-64.so.2
00007f1eb5f17000    152K r-x-- ld-linux-x86-64.so.2
…
00007f1eb5f4b000      8K rw--- ld-linux-x86-64.so.2
00007fff566df000    136K rw---   [ stack ]
00007fff56715000     16K r----   [ anon ]
00007fff56719000      4K r-x--   [ anon ]
ffffffffff600000      4K r-x--   [ anon ]
 total             4620K
$ gdb ./hello
```

Abstraction 3: Processes

- **Abstraction of a running program**

- Provides each running program with the illusion that it has exclusive access to the CPU

- Multiple processes can run concurrently
  - multi-core processors: true parallelism
  - single-cores: apparent parallelism through context-switching

Abstraction 3: Processes

- **Processes on a Linux system**

```
$ ps -AfeH
UID        PID  PPID  C STIME TTY          TIME CMD
root         2     0  0 Sep06 ?        00:00:00 [kthreadd]
root         3     2  0 Sep06 ?        00:00:00   [ksoftirqd/0]
root         5     2  0 Sep06 ?        00:00:00   [kworker/0:0H]
root         7     2  0 Sep06 ?        00:00:12   [rcu_sched]
root         8     2  0 Sep06 ?        00:00:00   [rcu_bh]
root         9     2  0 Sep06 ?        00:00:00   [migration/0]
…
root         1     0  0 Sep06 ?        00:00:00 init [3]
root      1574     1  0 Sep06 ?        00:00:00   /sbin/udevd --daemon
root      2173     1  0 Sep06 ?        00:00:00   supervising syslog-ng
root      2174  2173  0 Sep06 ?        00:00:00     /usr/sbin/syslog-ng --persist-file /var/lib/…
root      2202     1  0 Sep06 ?        00:00:00   /usr/sbin/crond
message+  2230     1  0 Sep06 ?        00:00:00   /usr/bin/dbus-daemon --system
root      2436     1  0 Sep06 ?        00:00:00   dhcpcd -m 4 enp5s0
root      2502     1  0 Sep06 ?        00:00:00   /usr/sbin/cupsd -C /etc/cups/cupsd.conf -s /etc/cups/…
ntp       2554     1  0 Sep06 ?        00:00:01   /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u ntp:ntp
root      2586     1  0 Sep06 ?        00:00:00   /usr/sbin/sshd
root      5096  2586  0 Sep06 ?        00:00:00     sshd: bernhard [priv]
bernhard  5108  5096  0 Sep06 ?        00:00:00       sshd: bernhard@notty
…
bernhard 27826     1  0 01:23 ?        00:00:00   /usr/bin/urxvt
bernhard 27827 27826  0 01:23 pts/2    00:00:00     bash
bernhard 29815 27827  0 02:09 pts/2    00:00:00       ps -AfeH
$ ps -AfeH | wc -l
689
$
```

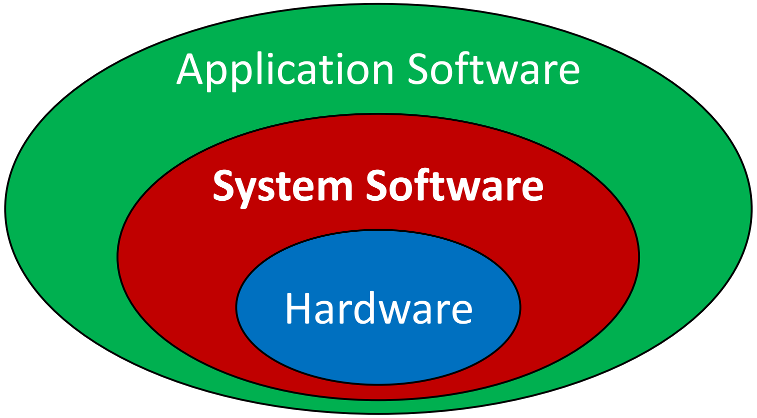
Abstraction 3: Processes

- **Process Isolation: Memory**

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int counter = 0;

int main(int argc, char *argv[])
{
  pid_t pid = getpid();

  printf("Process %5d: counter is located "
         "at address %p\n", pid, &counter);

  while (1) {
    printf("Process %5d: counter = %3d\n",
           pid, counter++);
    sleep(1);
  }

  return EXIT_SUCCESS;
}
```

count.c

```
$ ./count &
[1] 21616
$ [21616] counter located at 0x55555555805c
Process 21616: counter =   0
Process 21616: counter =   1
Process 21616: counter =   2
./count &
[2] 21617
$ [21617] counter located at 0x55555555805c
Process 21617: counter =   0
Process 21616: counter =   3
Process 21617: counter =   1
Process 21616: counter =   4
Process 21617: counter =   2
Process 21616: counter =   5
Process 21617: counter =   3
Process 21616: counter =   6
Process 21617: counter =   4
Process 21616: counter =   7
Process 21617: counter =   5
```



System Software

System Software?

- **What exactly is system software?**

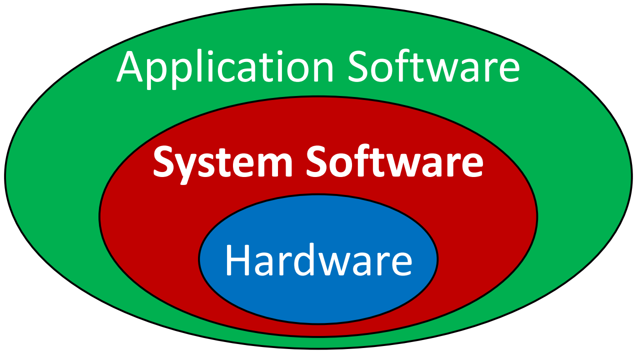
	Application Software	System Software
Purpose?	Software written to perform a specific task independent of the actual hardware	
Accesses hardware how?	Uses the services provided by system software to perform its function and interact with hardware	
Interacts with?	Interacts with hardware through the API provided by the kernel	
Programming language?	Written in many different programming languages	
Machine dependent?	Is (hopefully) machine independent	
Fault “tolerant”?	Errors simply crash the application	

System Software

- **Interface between application software and hardware**

	Application Software	System Software
Purpose?	Software written to perform a specific task independent of the actual hardware	Software enabling users to interact with the computer system
Accesses hardware how?	Uses the services provided by system software to perform its function and interact with hardware	Controls and manages the hardware
Interacts with?	Interacts with hardware through the API provided by the kernel	Interacts with hardware directly
Programming language?	Written in many different programming languages	Typically written in a flavor of C and assembly
Machine dependent?	Is (hopefully) machine independent	Is machine dependent
Fault “tolerant”?	Errors simply crash the application	Errors often lead to catastrophic failures

System Software



- **System software**
  Software designed to operate and control the hardware
  of a computer and to provide a platform for running application software.

- System software includes
  - kernel
  - programs that enable interaction with hardware
    - assembler
    - compiler
    - linker
  - low-level tools
    - inspection tools
    - disk checking and defragmenting

System Programming

System Programming

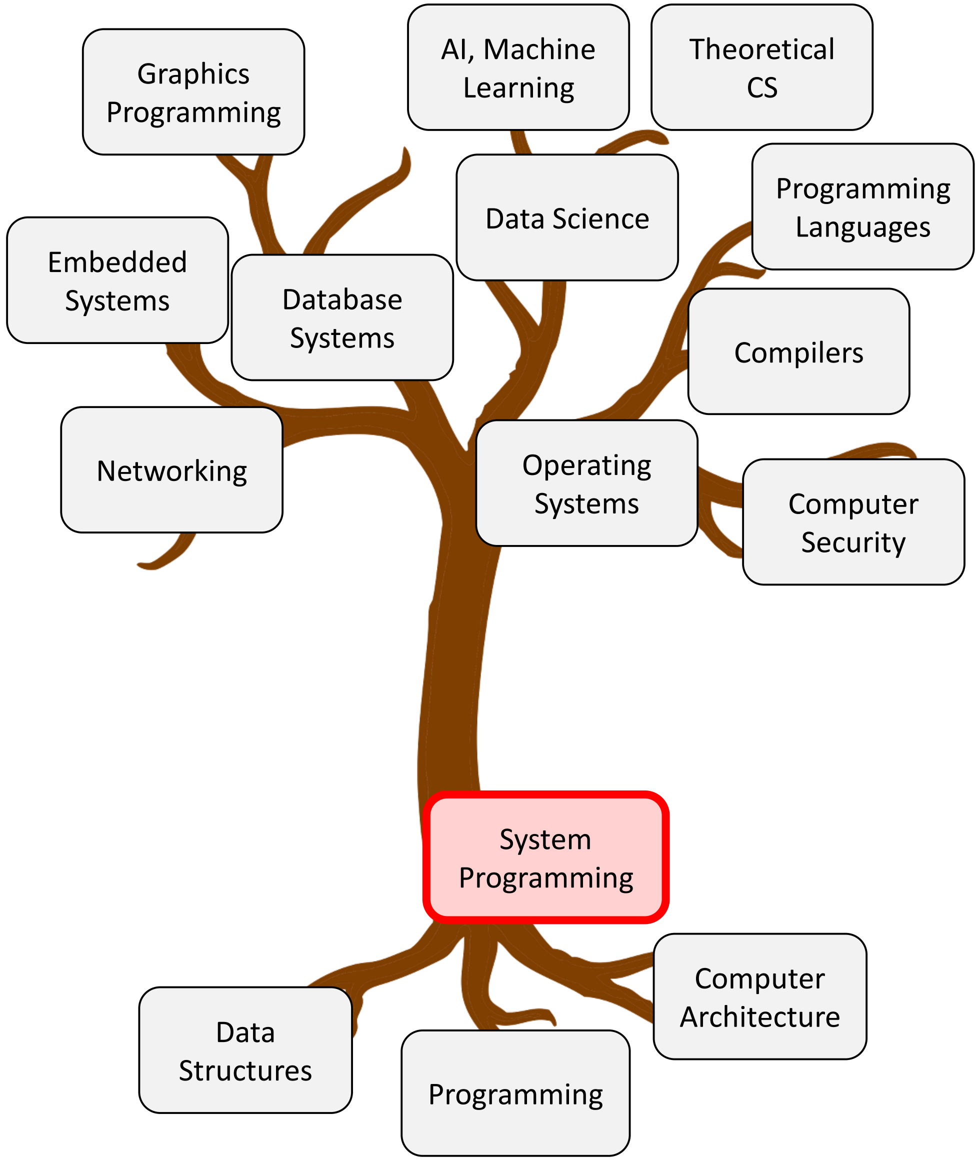
- This course: not about *writing* system software

- **Goal 1: Understanding the general concepts of system software**
  - abstractions
  - interaction hardware / system software

- **Goal 2: Know how to use the services provided by system software**
  - system calls
  - how different system software tools work

- **Goal 3: Become a better programmer**
  - Practice creates masters

Why System Programming?

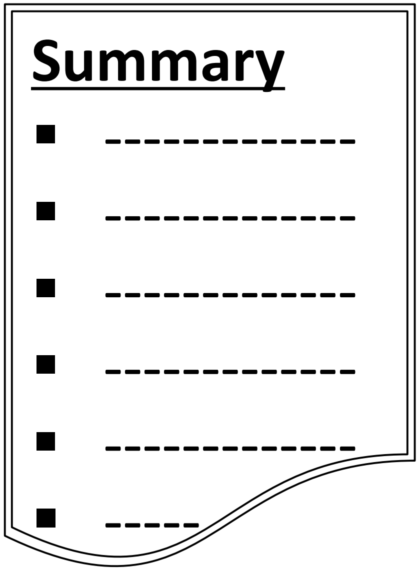


Why System Programming?

**"I program everything in Python.
I don't need to know this stuff."**

Why System Programming?

- **Well, I got bad news for you**
  - The Python interpreter/runtime
    https://github.com/python/cpython
  - Numpy
    https://github.com/numpy/numpy
  - Tensorflow
    https://github.com/tensorflow/tensorflow
  - Java (VM & compiler)
    https://github.com/openjdk/jdk
  - Julia
    https://github.com/JuliaLang/julia
  - R
    https://www.r-project.org/ (https://svn.r-project.org/R/trunk/)



Module Summary

Module Summary

- **Over the course of less than 100 years, computer systems have evolved from huge, slow monsters to the most high-tech ubiquitously available devices**

- **All computer systems share a common underlying architecture**
  - from high-performance servers to tiny embedded devices

- **Computer systems are all about abstractions**
  - to simplify the task of programming and using the computer system
  - these abstractions are provided by system software:
    the operating system, compilers, low-level tools

Module Summary

- **In this course, you will**
  - come to understand the general concepts of system software,
  - learn how to use the services provided by system software,
  - learn how different system software tools work, and
  - familiarize yourself with Linux, system tools, and C

- **Understanding how a computer system works and executes programs is essential for almost all hot areas in computer science**
  - cyber security
  - data science
  - machine learning and artificial intelligence
  - general super computing
  - quantum computing
  - and many more