CS 7172
Parallel and Distributed Computing

Process

Kun Suo

Computer Science, Kennesaw State University

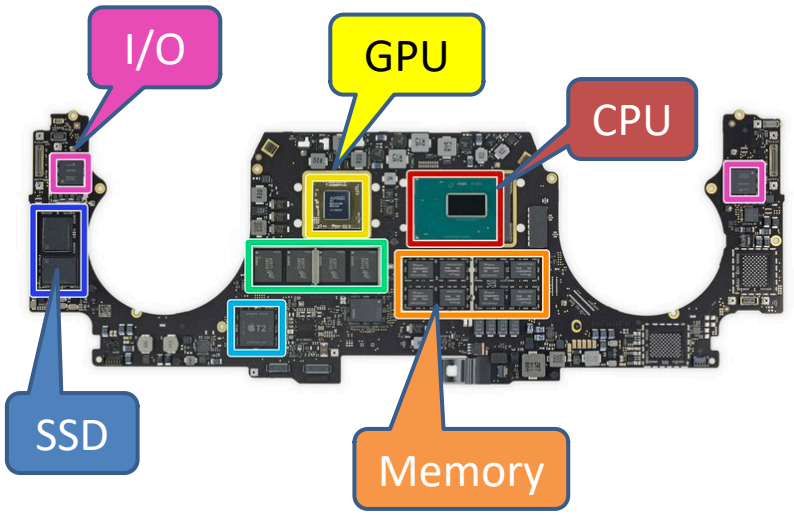
https://kevinsuo.github.io/

Outline

- Hardware review
- What is process?
  - Process vs Program
  - Linux Process Control Block
- Process related System calls
  - Fork
- Process on Distributed OSes

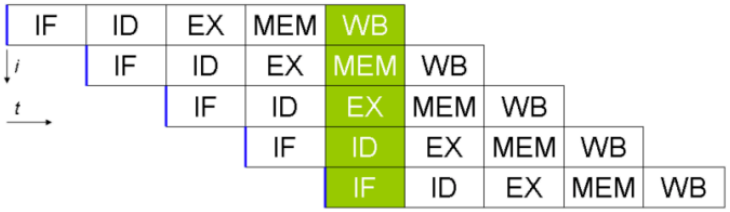
Computer Hardware Review

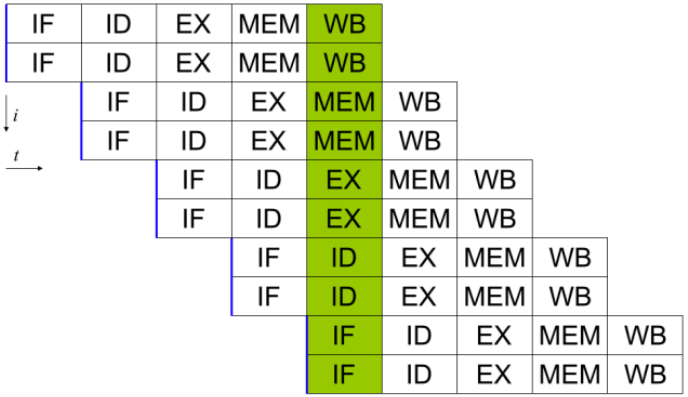
Computer Hardware Review

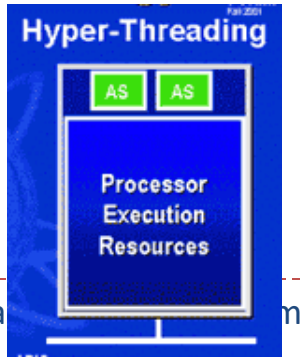


Central Processing Unit (CPU)

- Components
  - Arithmetic Logic Unit (ALU) -> Compute and data
  - Control Unit (CU) -> control device and system
- Clock rate
  - The speed at which a CPU is running
- Data storage
  - General-purpose registers: EAX, EBX …
  - Special-purpose registers: PC (program counter), SP (stack), IR (instruction register) …
- Parallelism
  - Instruction-level parallelism
  - Thread-level parallelism
    - Hyper-threading: duplicate units that store architectural states
    - Replicated: registers. Partitioned: ROB, load buffer… Shared: reservation station, caches

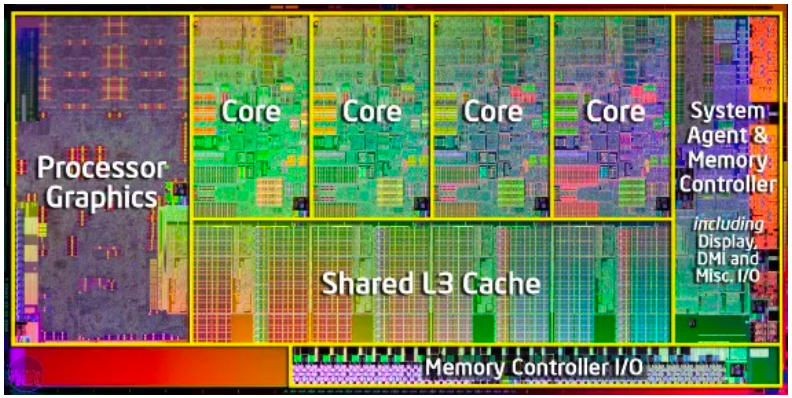


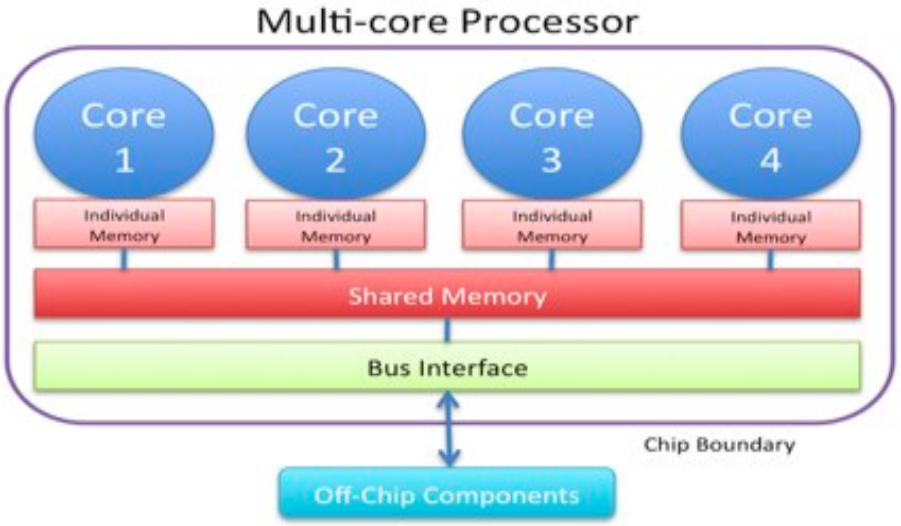




Multi-Core Processors (SMP)

- Multiple CPUs on a single chip

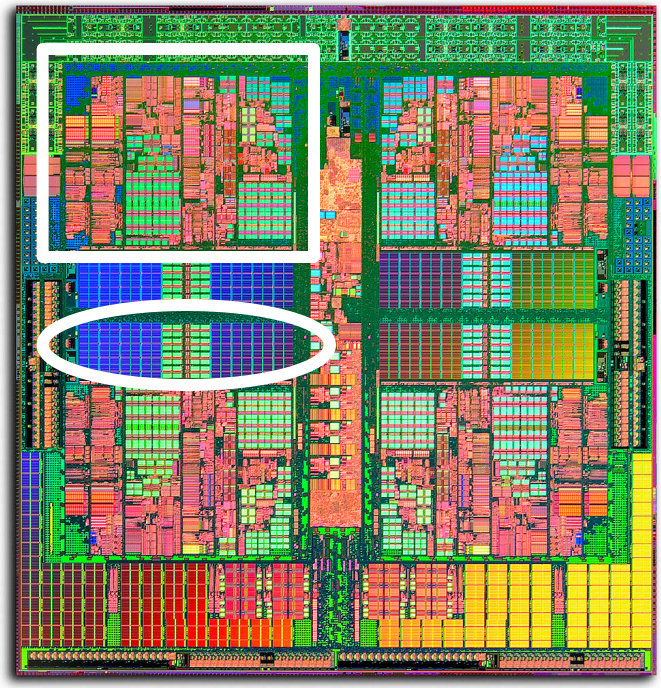


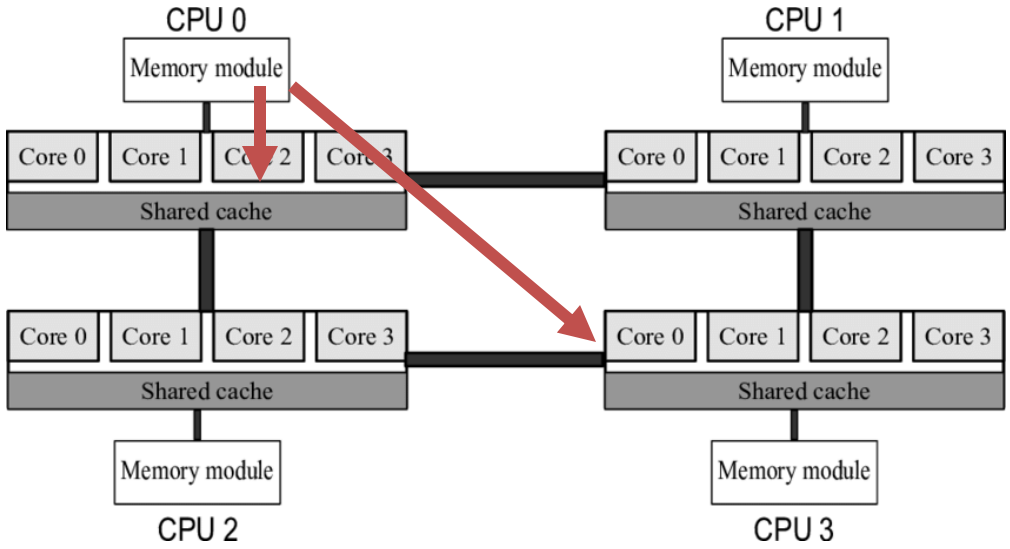


Symmetric multiprocessing (**SMP**)

Multi-Core Processors (NUMA)

- Multiple CPUs on a single chip

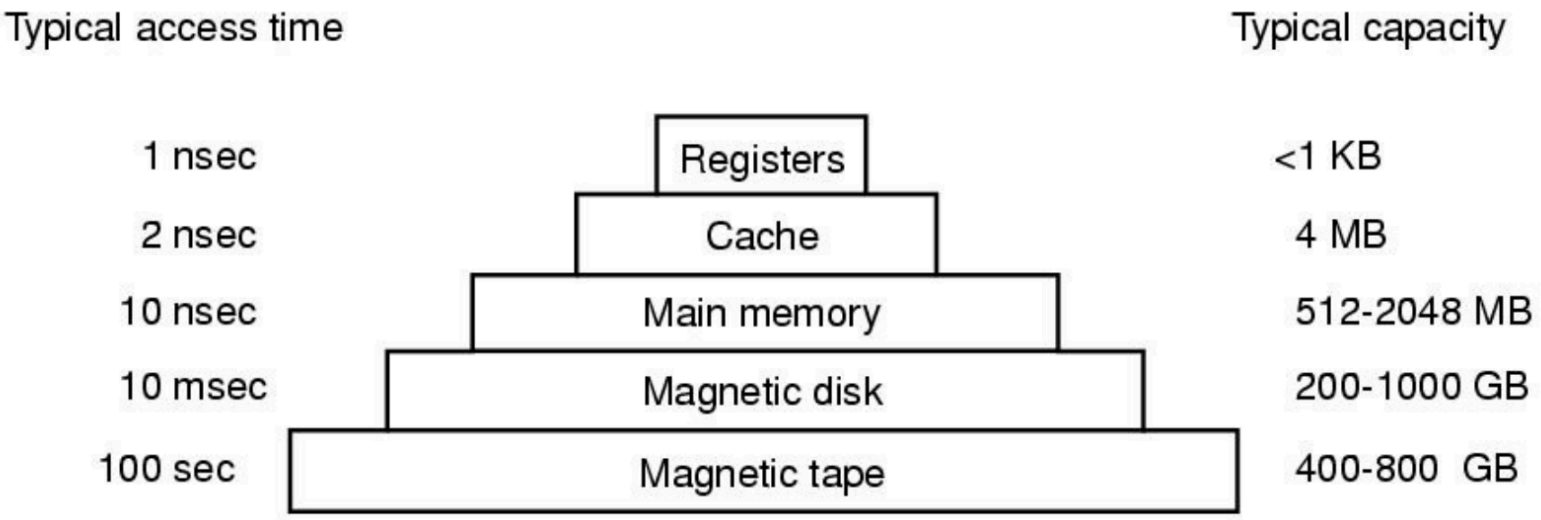




Non-uniform memory access (**NUMA**)

Memory

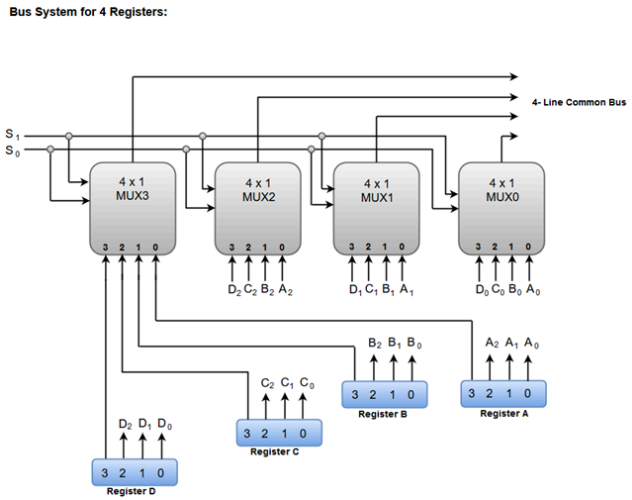
- A typical memory hierarchy

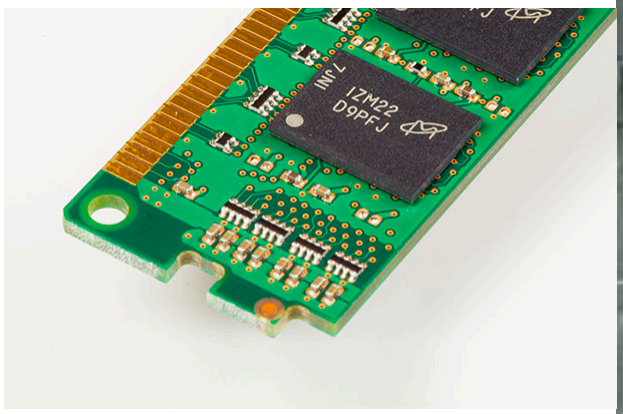


Minimize the access time vs. Cost

Memory

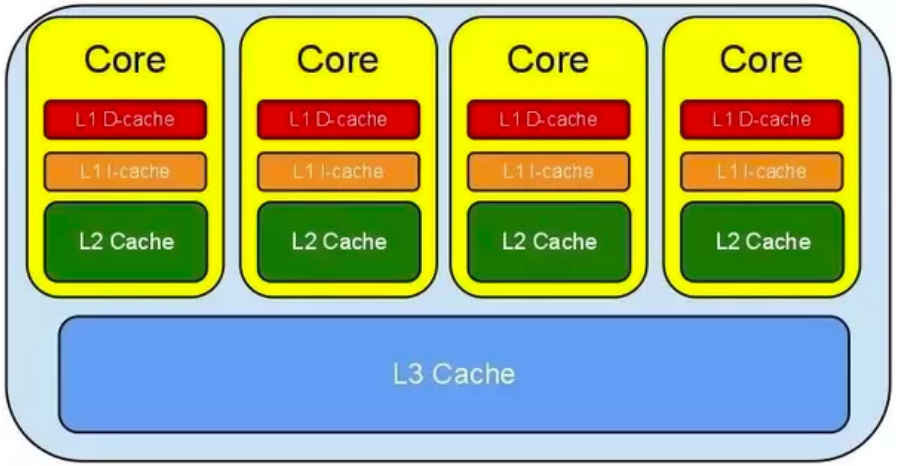
- A typical memory hierarchy

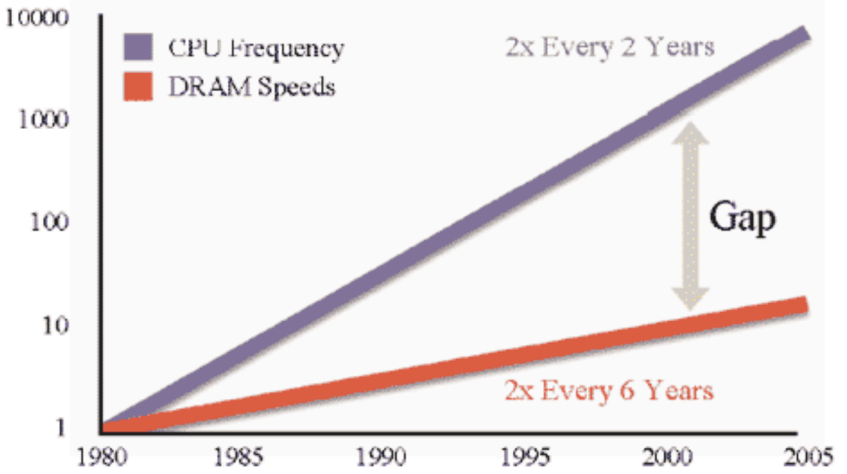




Cache

- Why Cache is important?





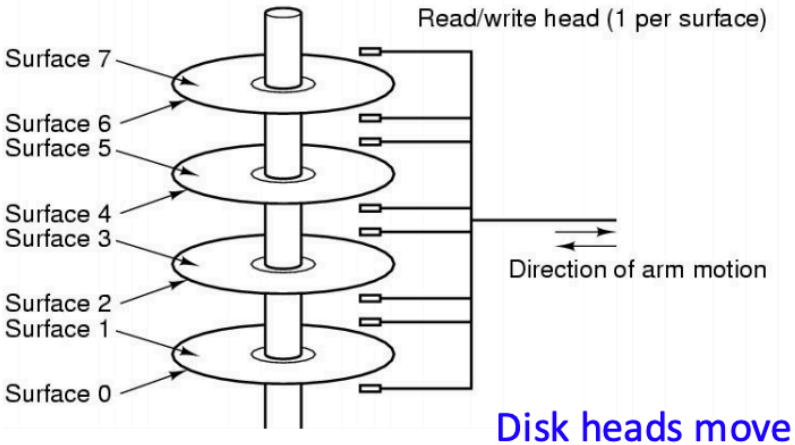
A larger size than registers
A much faster speed than memory
Tradeoff between performance and cost

MacBook Pro

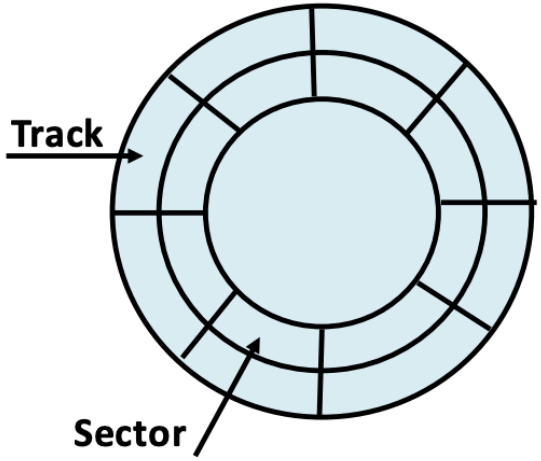
Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro15,1
Processor Name:	Intel Core i9
Processor Speed:	2.3 GHz
Number of Processors:	1
Total Number of Cores:	8
L2 Cache (per Core):	256 KB
L3 Cache:	16 MB
Hyper-Threading Technology:	Enabled
Memory:	16 GB
Boot ROM Version:	220.270.99.0.0 (iBridge: 16.16.6568.0.0,0)
Serial Number (system):	C02YR4JHLVCJ
Hardware UUID:	DCC2D30A-9630-57B5-89A2-5F2B85254DC1

Disk

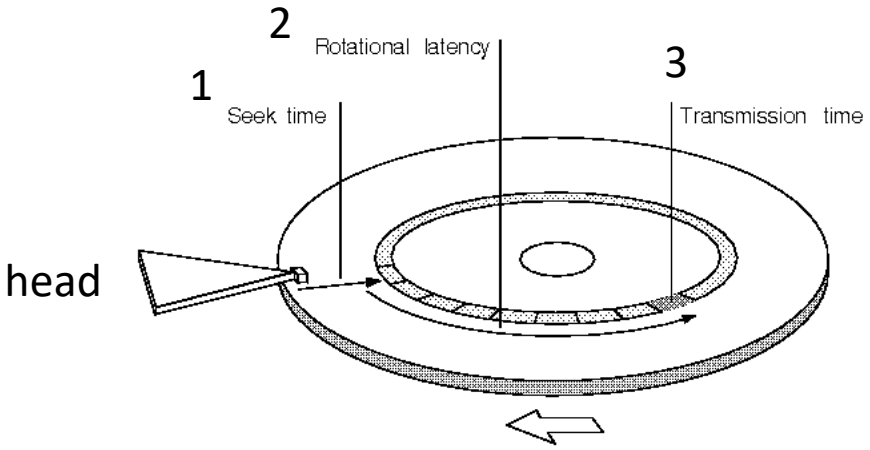


Disk heads move together



- A stack of platters, a surface with a magnetic coating
- Typical numbers (depending on the disk size):
  - 500 to 2,000 tracks per surface
  - 32 to 128 sectors per track
    - A sector is the smallest unit that can be read or written
- Originally, all tracks have the same number of sectors

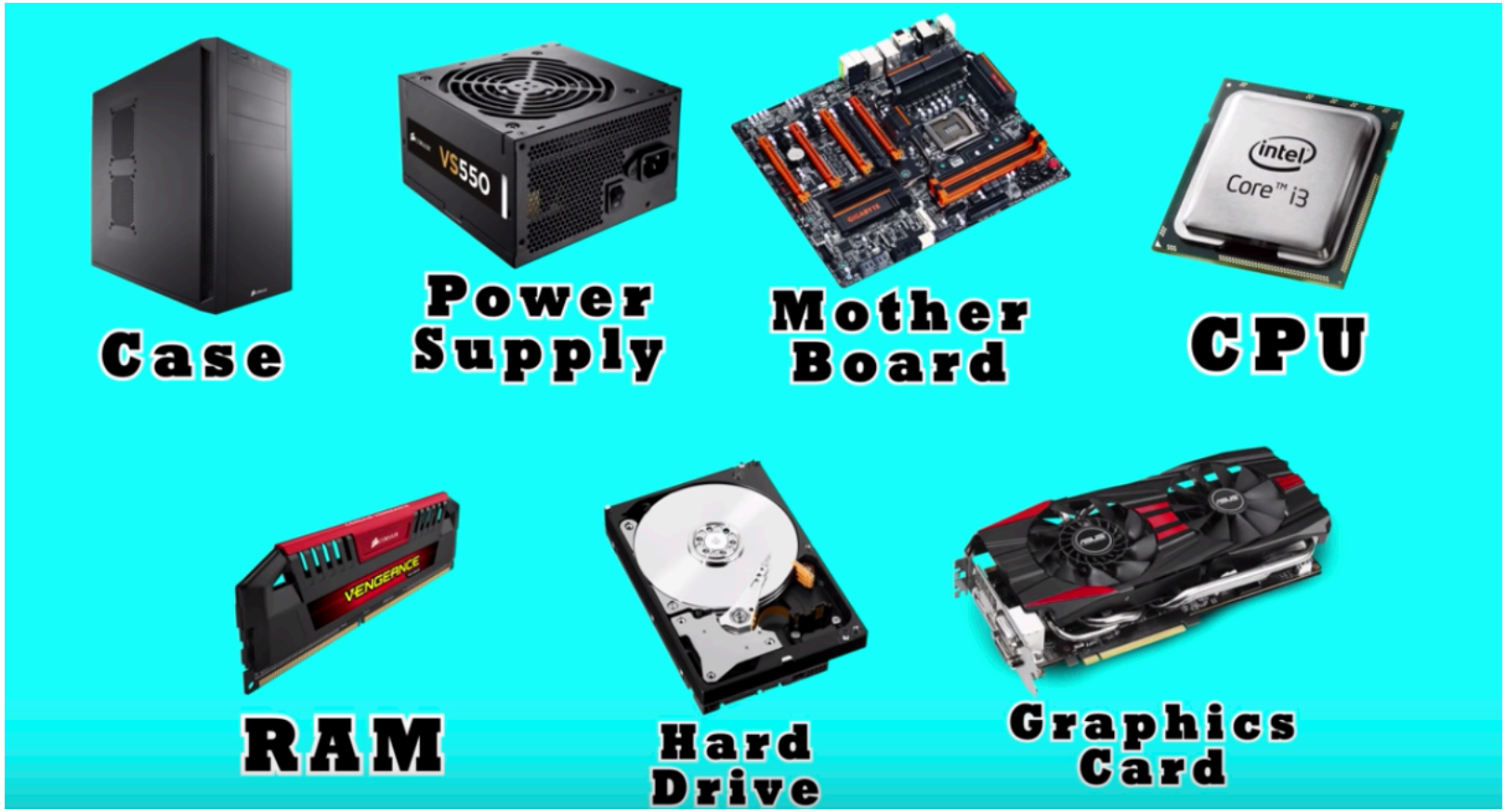
Disk



- Disk head: each side of a platter has separate disk head
- Read/write data is a three-stage process:
  - Seek time: position the arm over the proper track
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 15 ms

Disk R/W Process

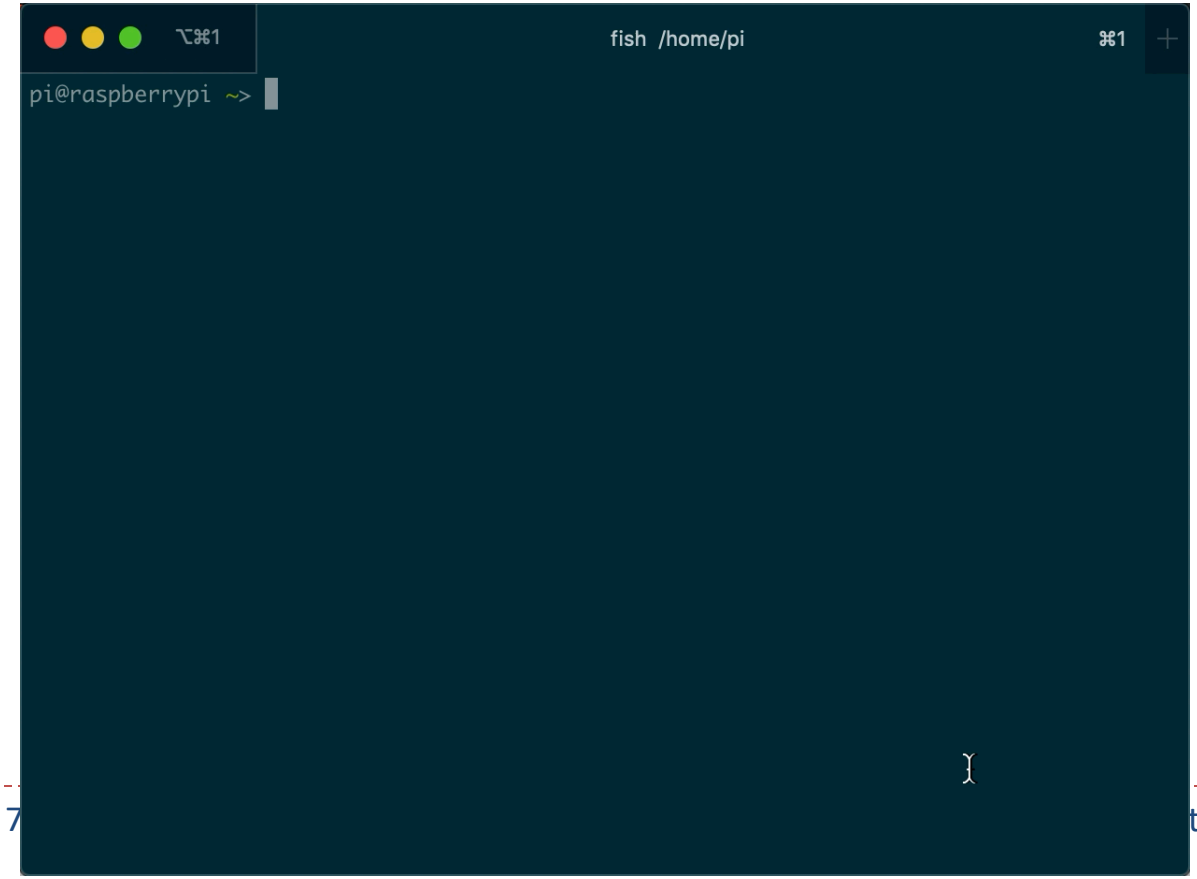
An interesting video introducing hardware



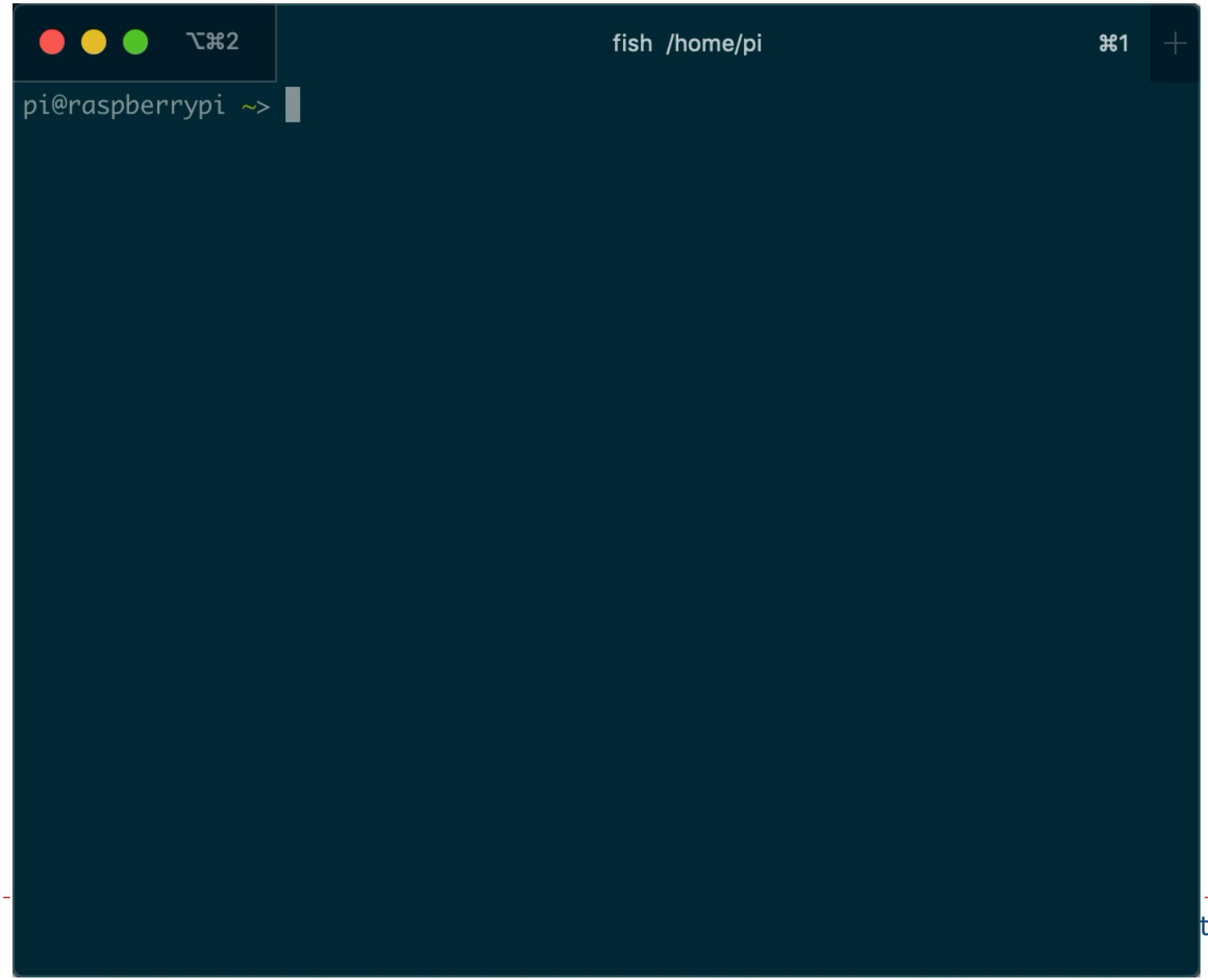
https://www.youtube.com/watch?v=ExxFxD4OSZ0

How to get my machine spec?

- Inxi: https://www.tecmint.com/inxi-command-to-find-linux-system-information/



How to get my machine spec? One command for All!



Outline

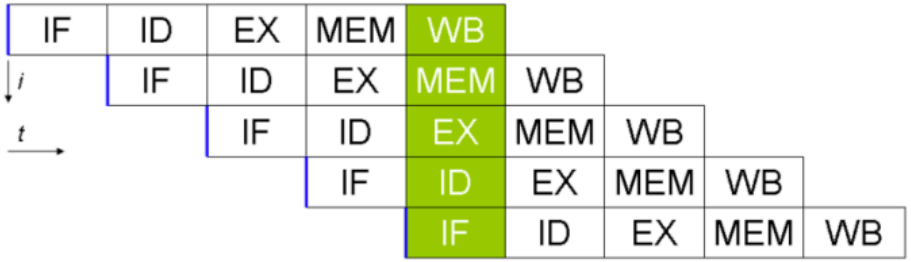
- Hardware review
- What is process?
  - Process vs Program
  - Linux Process Control Block
- Process related System calls
  - Fork
- Process on Distributed OSes

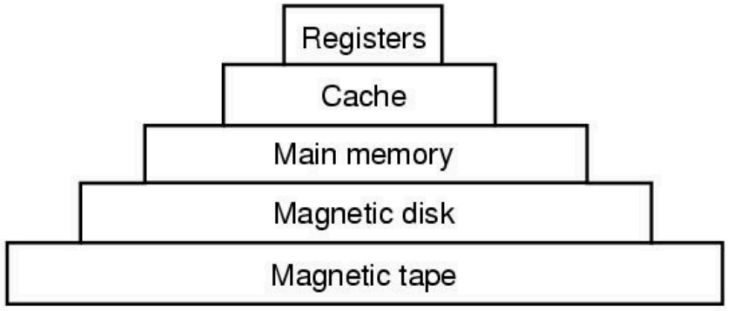
Process

- Definition
  - An instance of a *program* running on a computer
  - An *abstraction* that supports running programs - -> cpu virtualization
  - An *execution stream* in the context of a particular *process state* - -> dynamic unit
  - A *sequential* stream of execution in its *own address space* - -> execution code line by line

Process

- Two parts of a process
  - Sequential execution of instructions
  - Process state
    - registers: PC (program counter), SP (stack pointer),…
    - Memory: address space, code, data, stack, heap …
    - I/O status: open files …



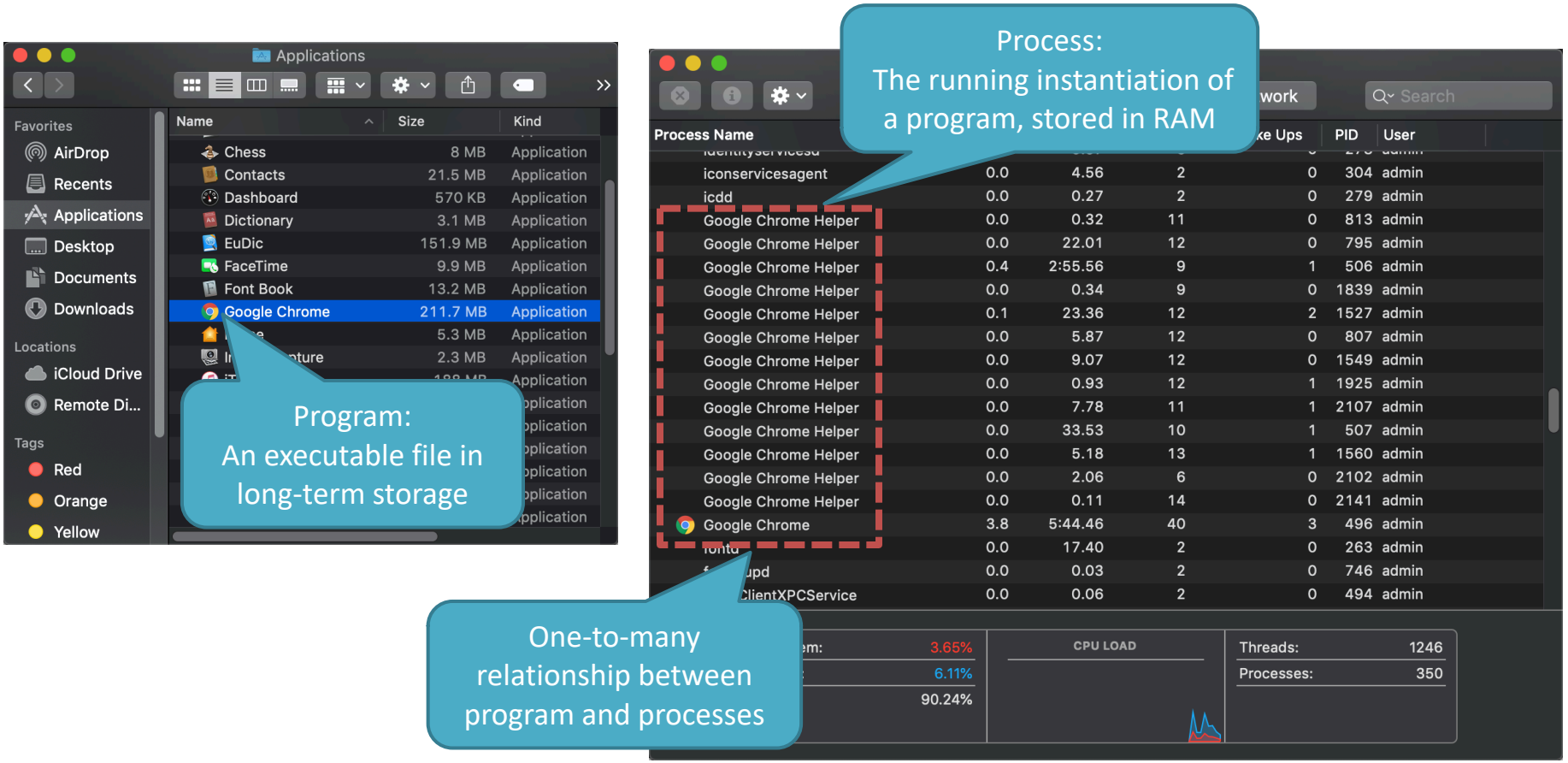


Program vs. Process

- Program != Process
  - Program = static code + data
  - Process = dynamic instantiation of code + data + files …

- No 1:1 mapping
  - Program : process = 1:N
    - A program can invoke many processes

Program vs. Process



Program vs. Process

BASIS FOR COMPARISON	PROGRAM	PROCESS
Basic	Program is a set of instruction.	When a program is executed, it is known as process.
Nature	Passive	Active
Lifespan	Longer	Limited
Required resources	Program is stored on disk in some file and does not require any other resources.	Process holds resources such as CPU, memory address, disk, I/O etc.

https://techdifferences.com/difference-between-program-and-process.html

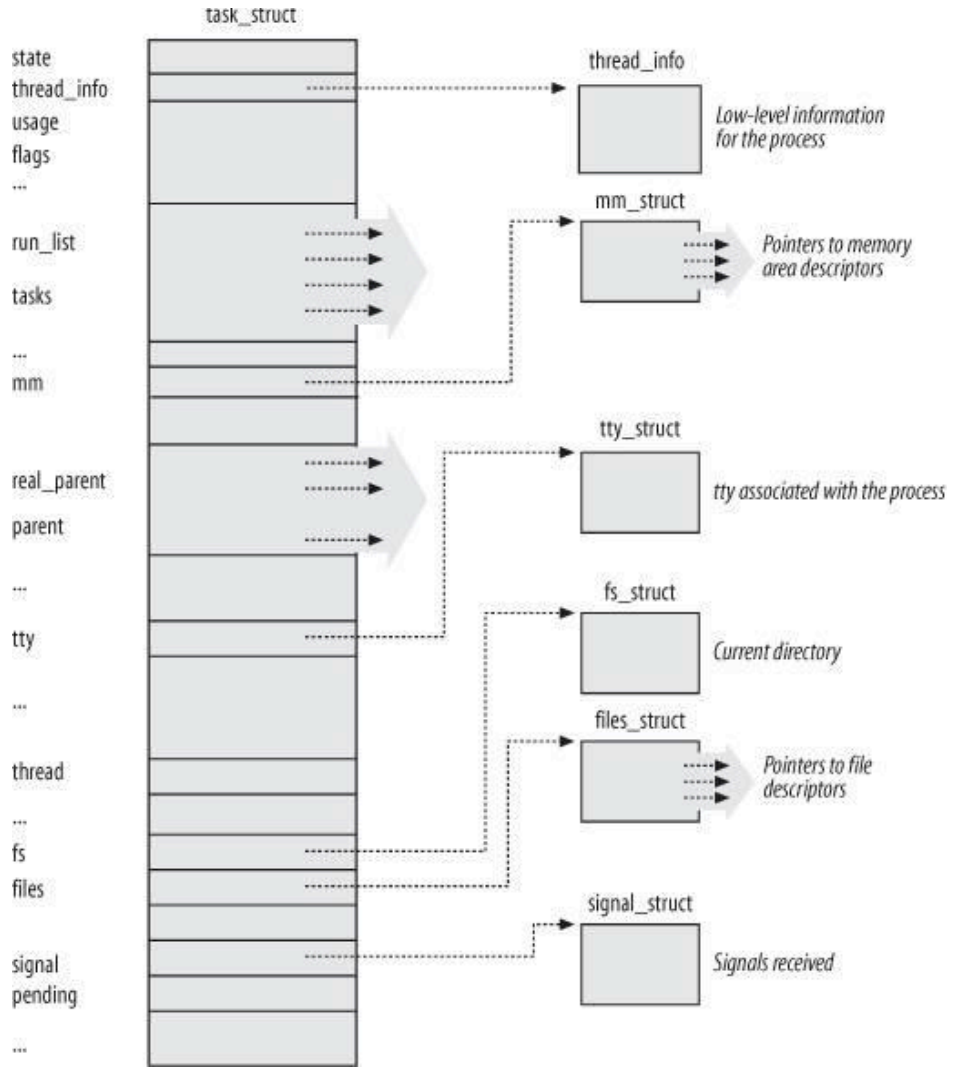
Process Descriptor

- Driving license
  - ID
  - Name
  - Address
  - Birth
  - Time
  - …
- Process control block (PCB)
  - State
  - Identifiers
  - Scheduling info
  - File system
  - Virtual memory
  - Process specific context
  - …

Process in Linux

https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h#L584

- Process control block (PCB)
  - State
  - Identifiers
  - Scheduling info
  - File system
  - Virtual memory
  - Process specific context
  - …

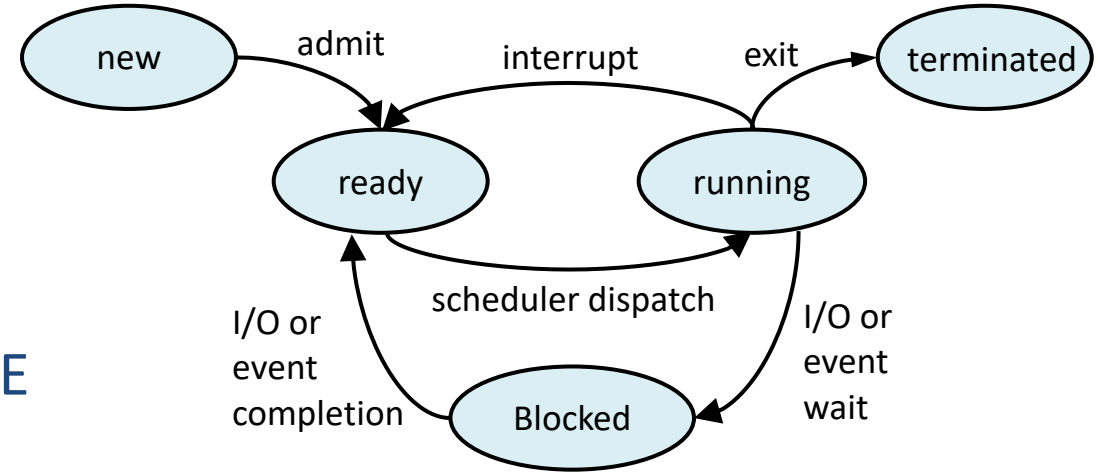


Process in Linux

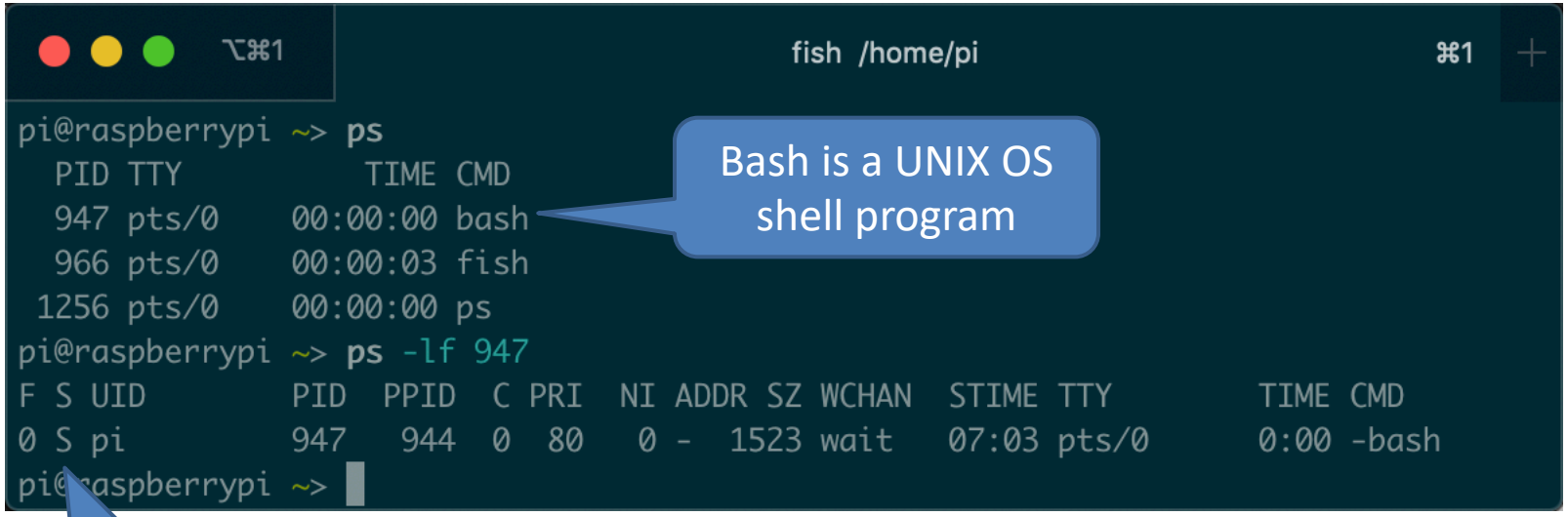
- Process control block (PCB)
  - State 
  - Identifiers
  - Scheduling info
  - File system
  - Virtual memory
  - Process specific context
  - …

Linux PCB (5-state model to describe lifecycle of one process)

- State
  - TASK_RUNNING
    - Running, ready
  - TASK_INTERRUPTABLE
    - Blocked
  - EXIT_ZOMBIE
    - Terminated by not deallocated
  - EXIT_DEAD
    - Completely terminated



$ ps -lf : process information



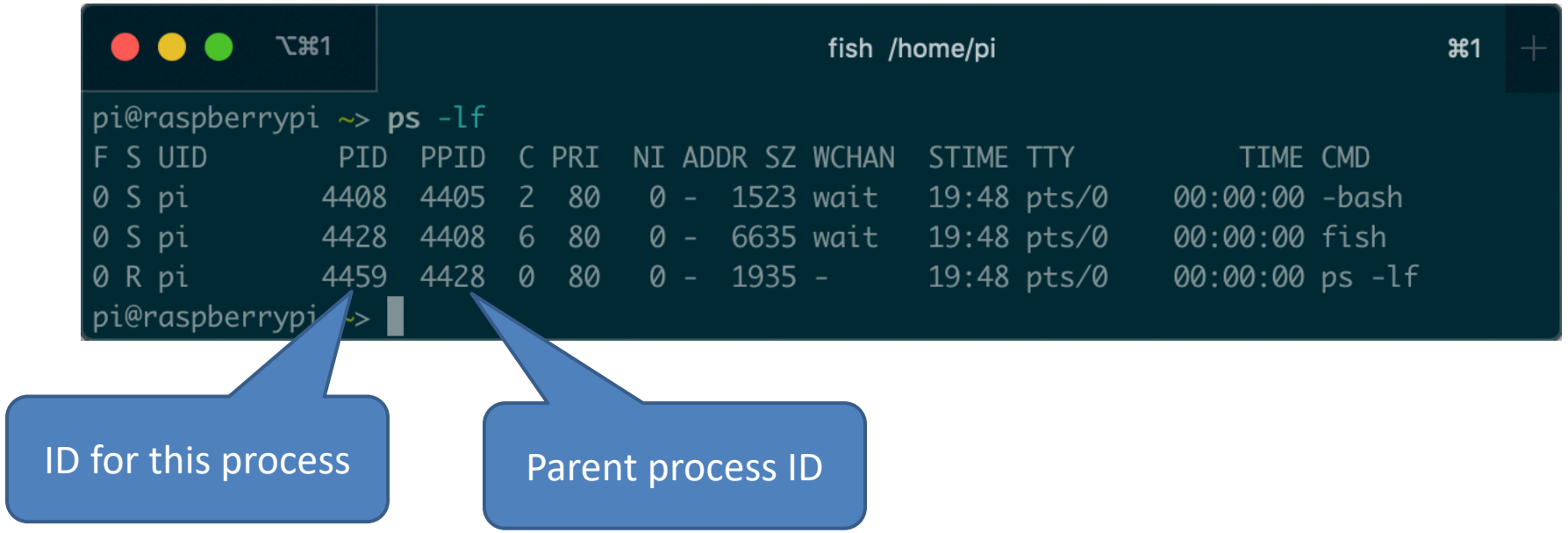
R： The process is running
S： The process is sleeping/idle
T： The process is terminated
Z： The process is in zombie state

Process in Linux

- Process control block (PCB)
  - State
  - Identifiers 
  - Scheduling info
  - File system
  - Virtual memory
  - Process specific context
  - …

Linux Process Control Block (cont')

- Identifiers
  - pid: ID of the process



Process in Linux

- Process control block (PCB)
  - State
  - Identifiers
  - Scheduling info 
  - File system
  - Virtual memory
  - Process specific context
  - …

Linux Process Control Block (cont')

- Scheduling information
  - prio, static_prio, normal_prio
  - rt_priority
  - sched_class

Linux Process Control Block (cont')

- Scheduling information
  - prio, static_prio, normal_prio

(1) Static priority: P1 > P2 = P3 = P4, P1 can execute whenever it needs;
(2) Normal priority: P1 = P2 = P3 = P4, P1 execute depending on the scheduling algorithm;
(3) Prio: dynamic priority, will change over the time

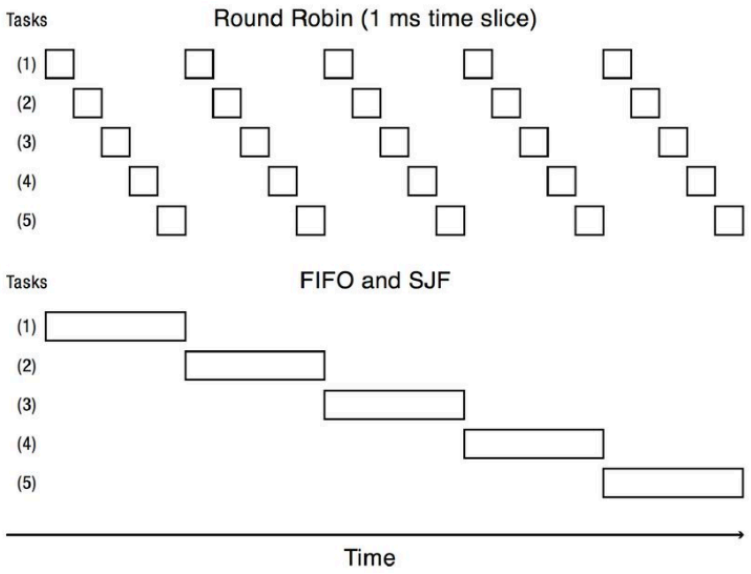
Linux Process Control Block (cont')

- Scheduling information
  - rt_priority

Rt_priority process is always higher than other priority of processes and will be scheduled immediately when it needs

Linux Process Control Block (cont')

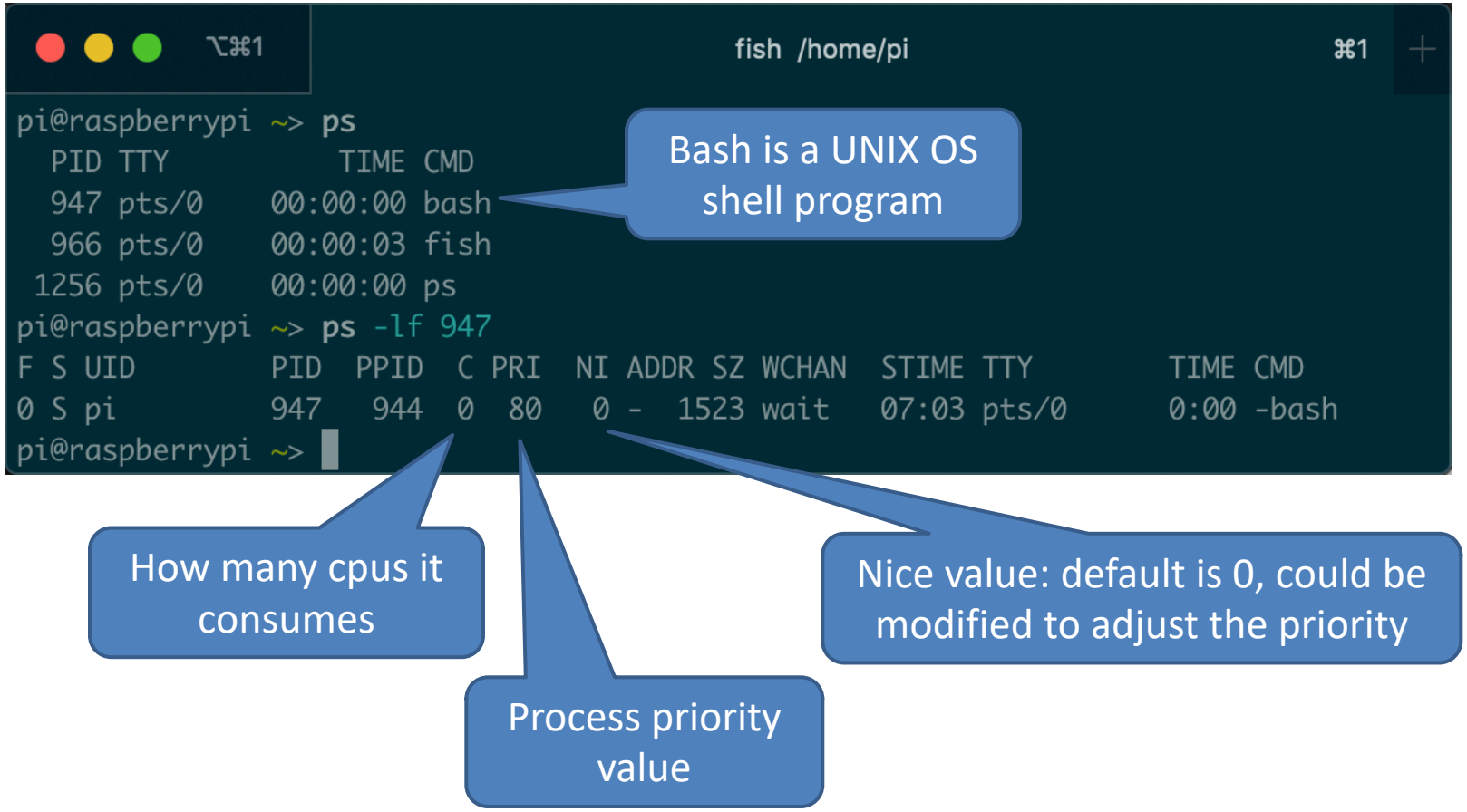
- Scheduling information
  - sched_class: different scheduling policy implementations, e.g., FIFO, SJF, RR…
    - Task->sched_class->pick_next_task(runqueue)



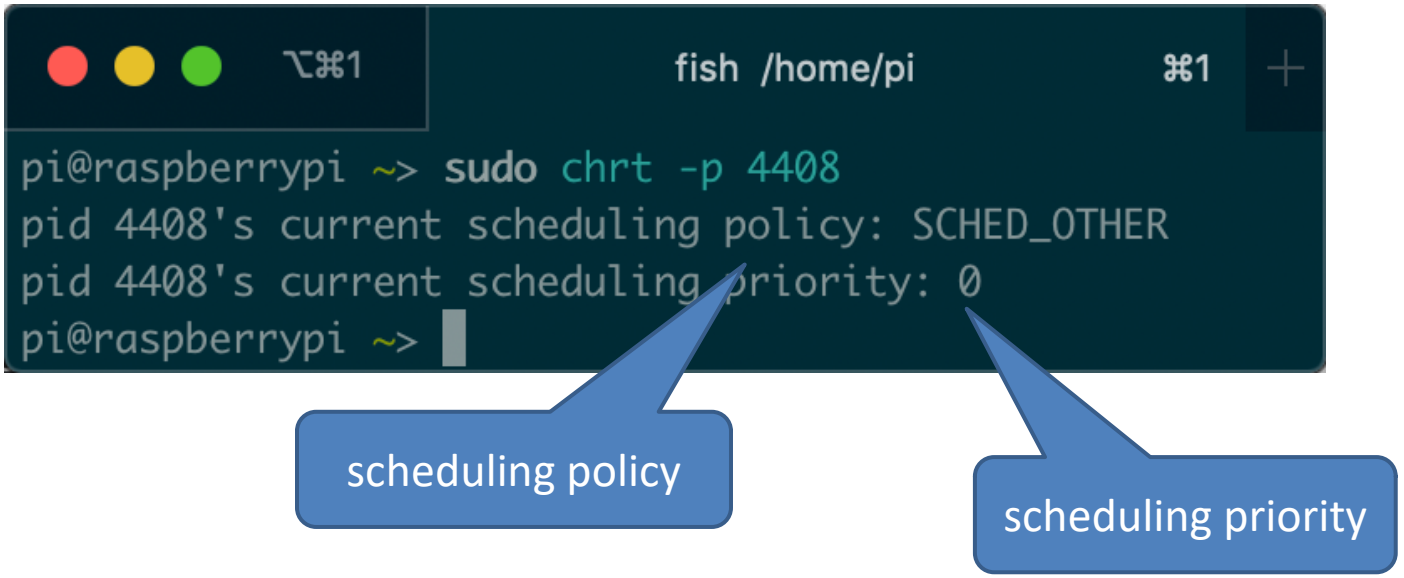
pick_next_task of RR:
pick based on time cycle

pick_next_task of FIFO:
pick based on order

$ ps -lf : process information



$ chrt: process scheduling info



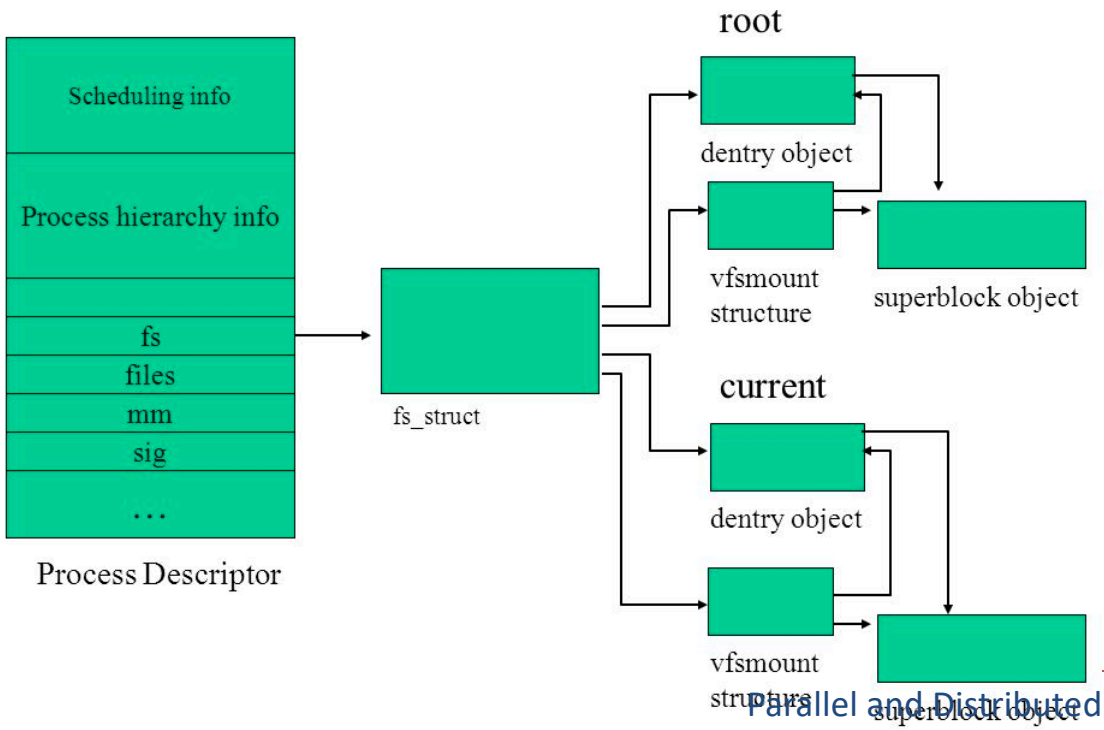
SCHED_OTHER	min/max priority : 0/0
SCHED_FIFO	min/max priority : 1/99
SCHED_RR	min/max priority : 1/99
SCHED_BATCH	min/max priority : 0/0

Process in Linux

- Process control block (PCB)
  - State
  - Identifiers
  - Scheduling info
  - File system 
  - Virtual memory
  - Process specific context
  - …

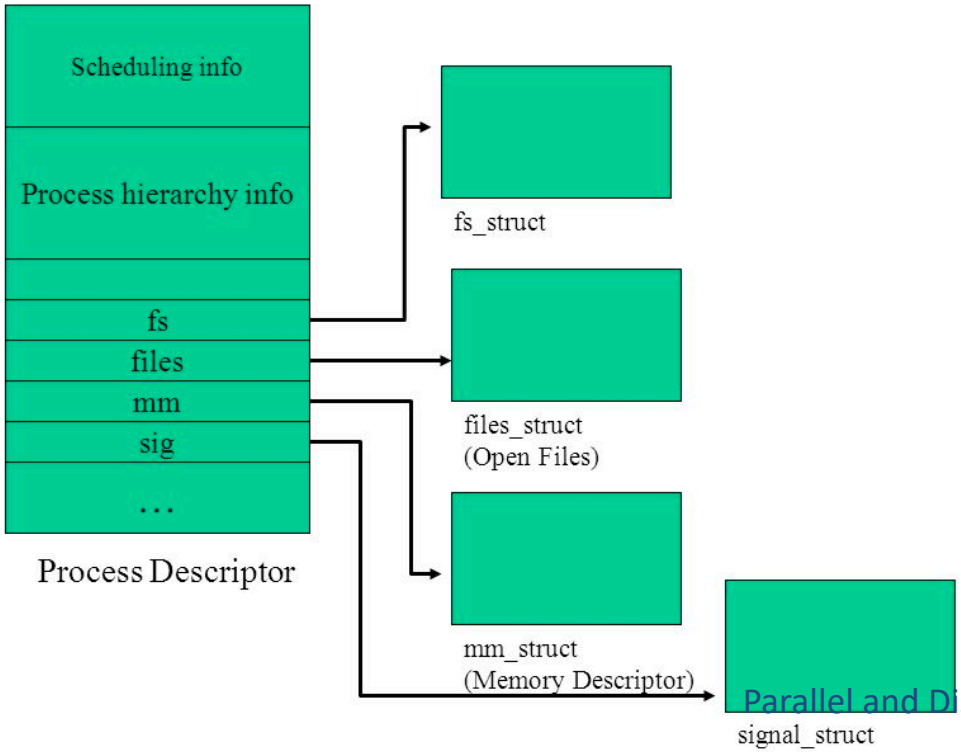
Linux Process Control Block (cont')

- Files
  - fs_struct https://elixir.bootlin.com/linux/v4.2/source/include/linux/sched.h#L1525
    - file system information: root directory, current directory

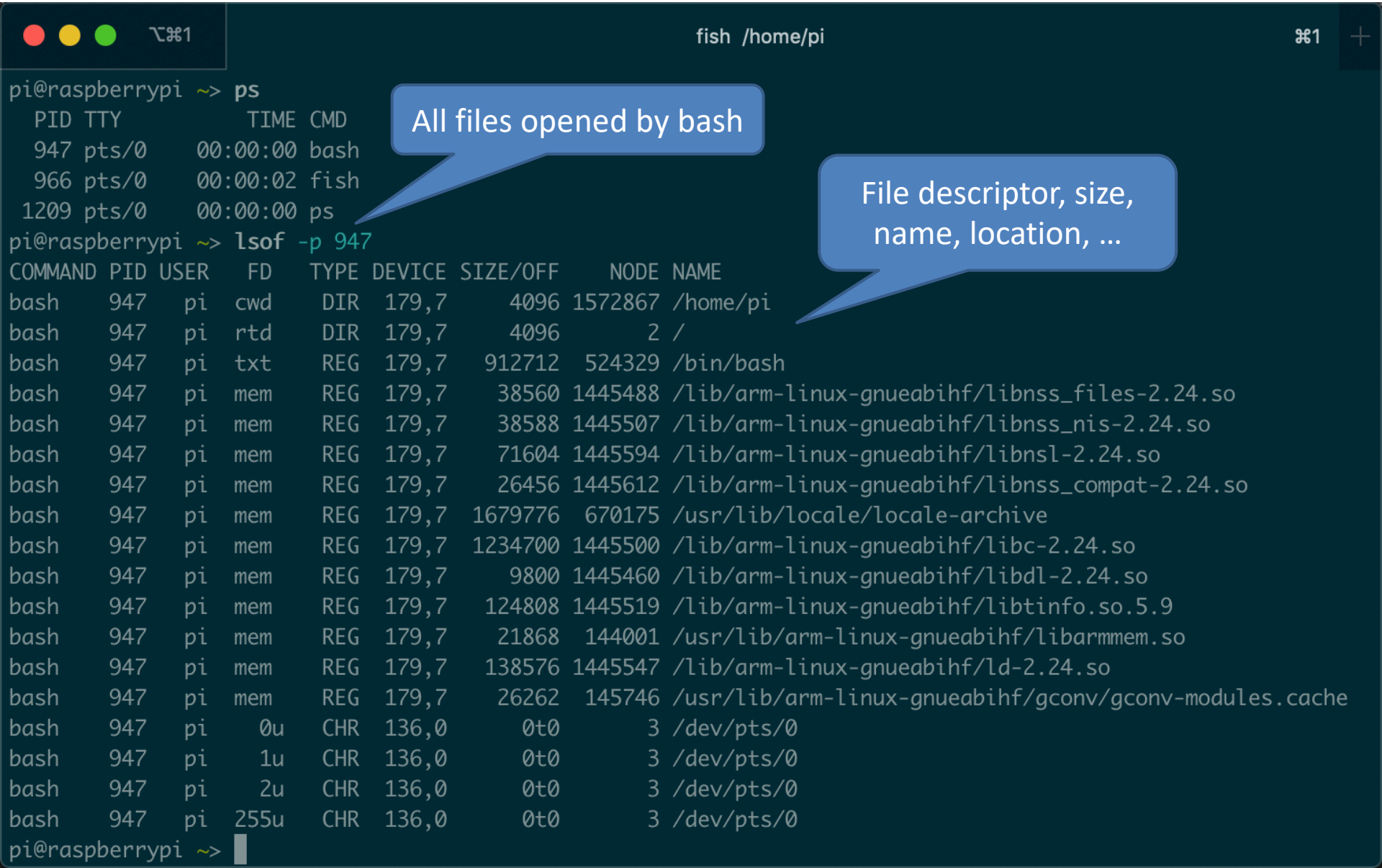


Linux Process Control Block (cont')

- Files
  - files_struct https://elixir.bootlin.com/linux/v4.2/source/include/linux/sched.h#L1528
    - Information on opened files



$lsof: list all open files

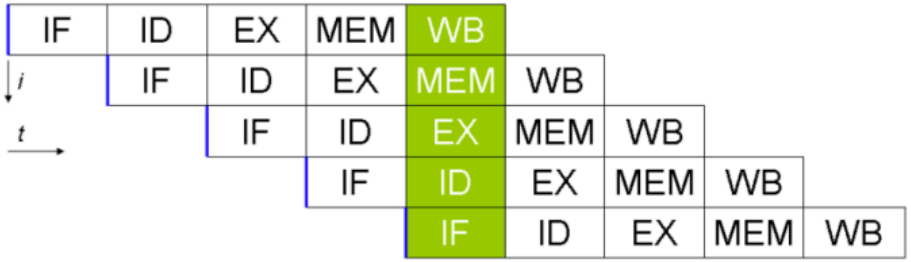


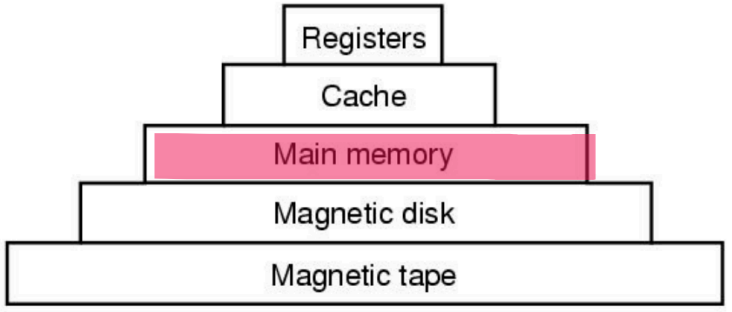
Process in Linux

- Process control block (PCB)
  - State
  - Identifiers
  - Scheduling info
  - File system
  - Virtual memory 
  - Process specific context
  - …

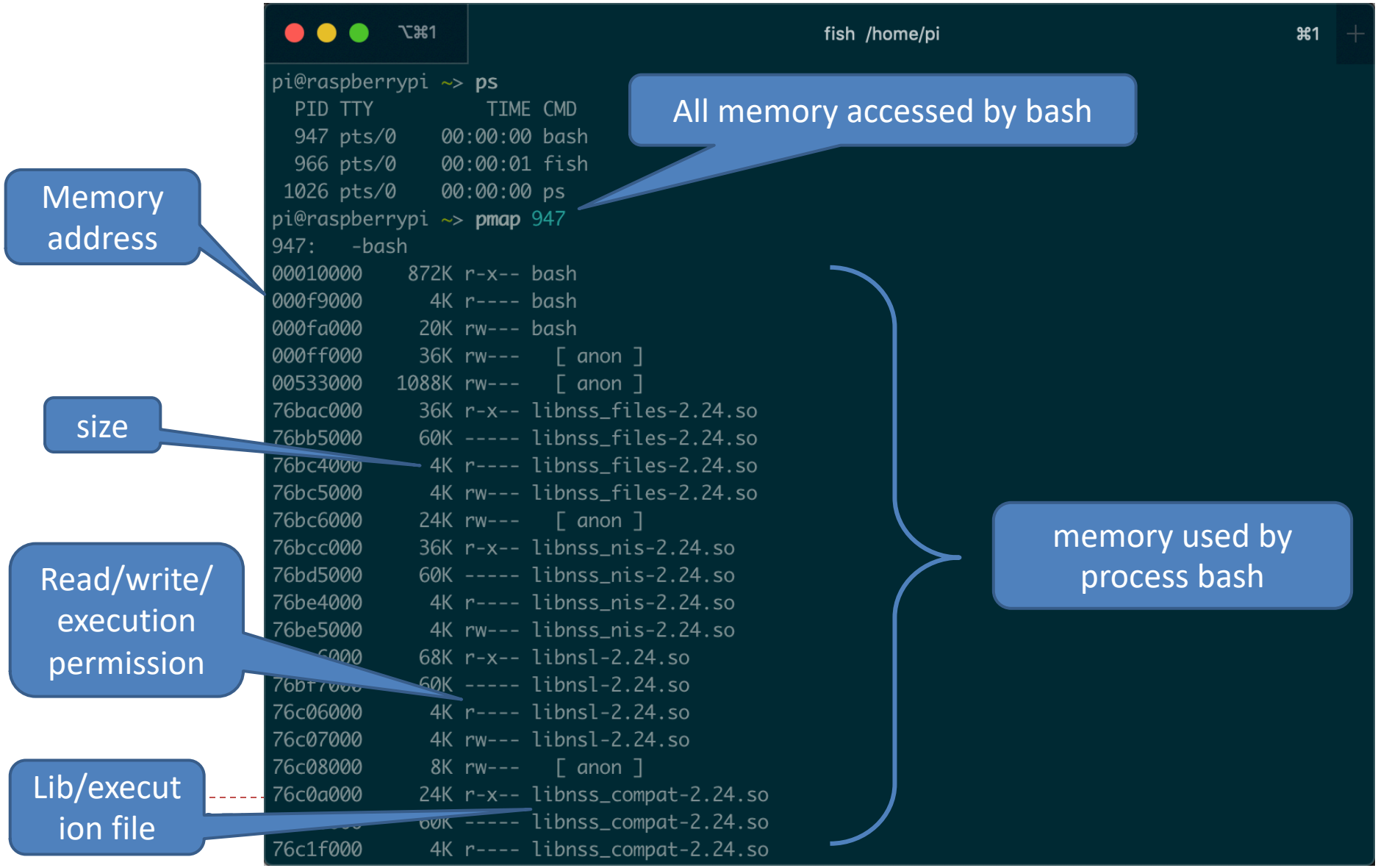
Linux Process Control Block (cont')

- Virtual memory
  - mm_struct: describes the content of a process's virtual memory
    - The pointer to the page table and the virtual memory areas

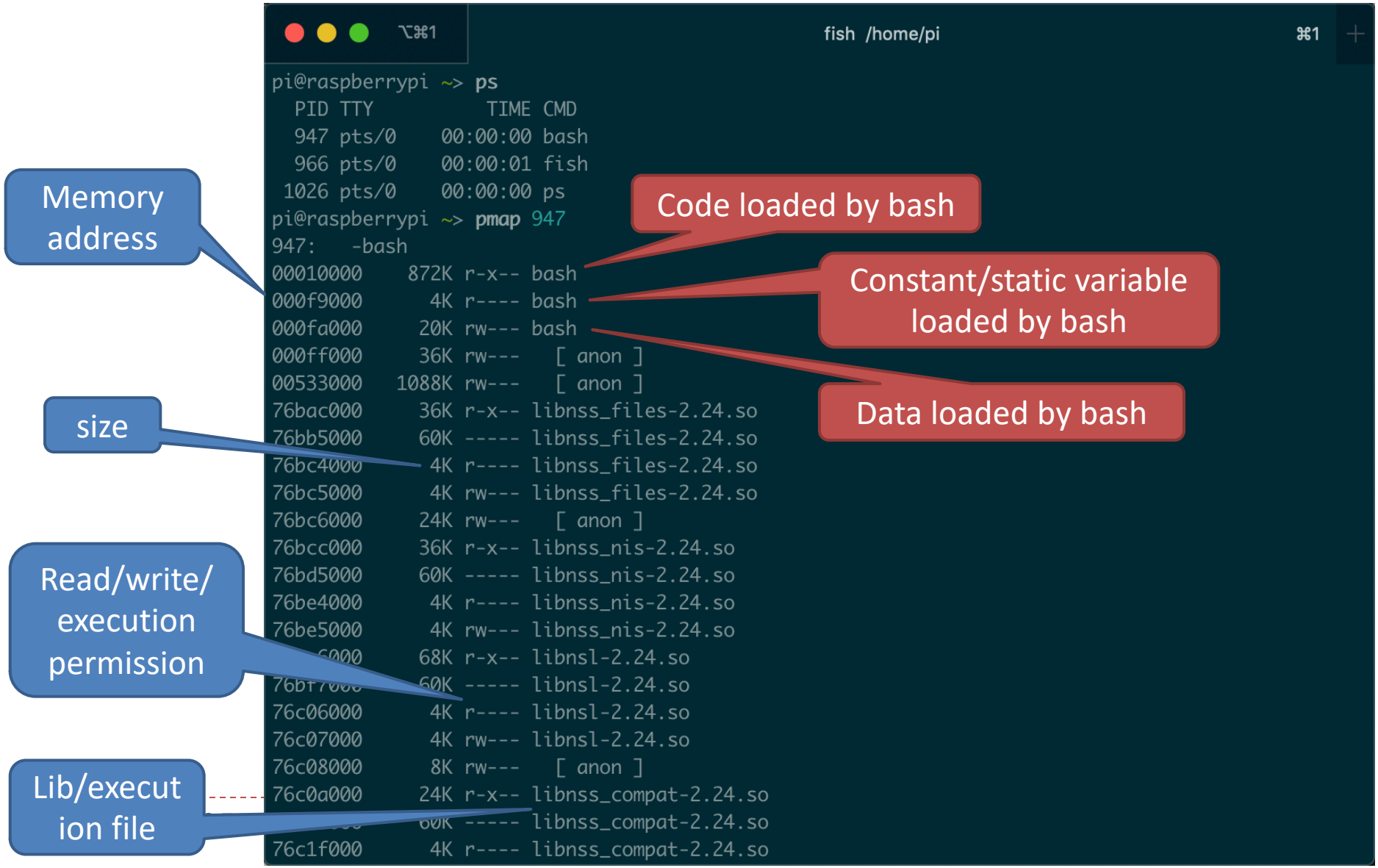




$pmap: memory mapping



$pmap: memory mapping



$pmap: memory mapping

Bash in memory

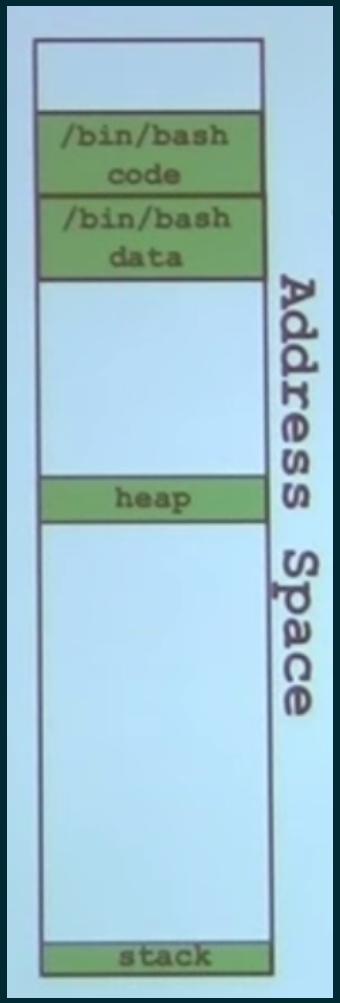
Memory address

size

Read/write/ execution permission

Lib/execut ion file

```
pi@raspberrypi ~> ps
  PID TTY          TIME CMD
  947 pts/0    00:00:00 bash
  966 pts/0    00:00:01 fish
 1026 pts/0    00:00:00 ps
pi@raspberrypi ~> pmap 947
947:   -bash
00010000    872K r-x-- bash
000f9000      4K r---- bash
000fa000     20K rw--- bash
000ff000     36K rw---   [ anon ]
00533000   1088K rw---   [ anon ]
76bac000     36K r-x-- libnss_files-2.24.so
76bb5000     60K ----- libnss_files-2.24.so
76bc4000      4K r---- libnss_files-2.24.so
76bc5000      4K rw--- libnss_files-2.24.so
76bc6000     24K rw---   [ anon ]
76bcc000     36K r-x-- libnss_nis-2.24.so
76bd5000     60K ----- libnss_nis-2.24.so
76be4000      4K r---- libnss_nis-2.24.so
76be5000      4K rw--- libnss_nis-2.24.so
[illegible]  68K r-x-- libnsl-2.24.so
[illegible]  60K ----- libnsl-2.24.so
76c06000      4K r---- libnsl-2.24.so
76c07000      4K rw--- libnsl-2.24.so
76c08000      8K rw---   [ anon ]
76c0a000     24K r-x-- libnss_compat-2.24.so
[illegible]  60K ----- libnss_compat-2.24.so
76c1f000      4K r---- libnss_compat-2.24.so
```

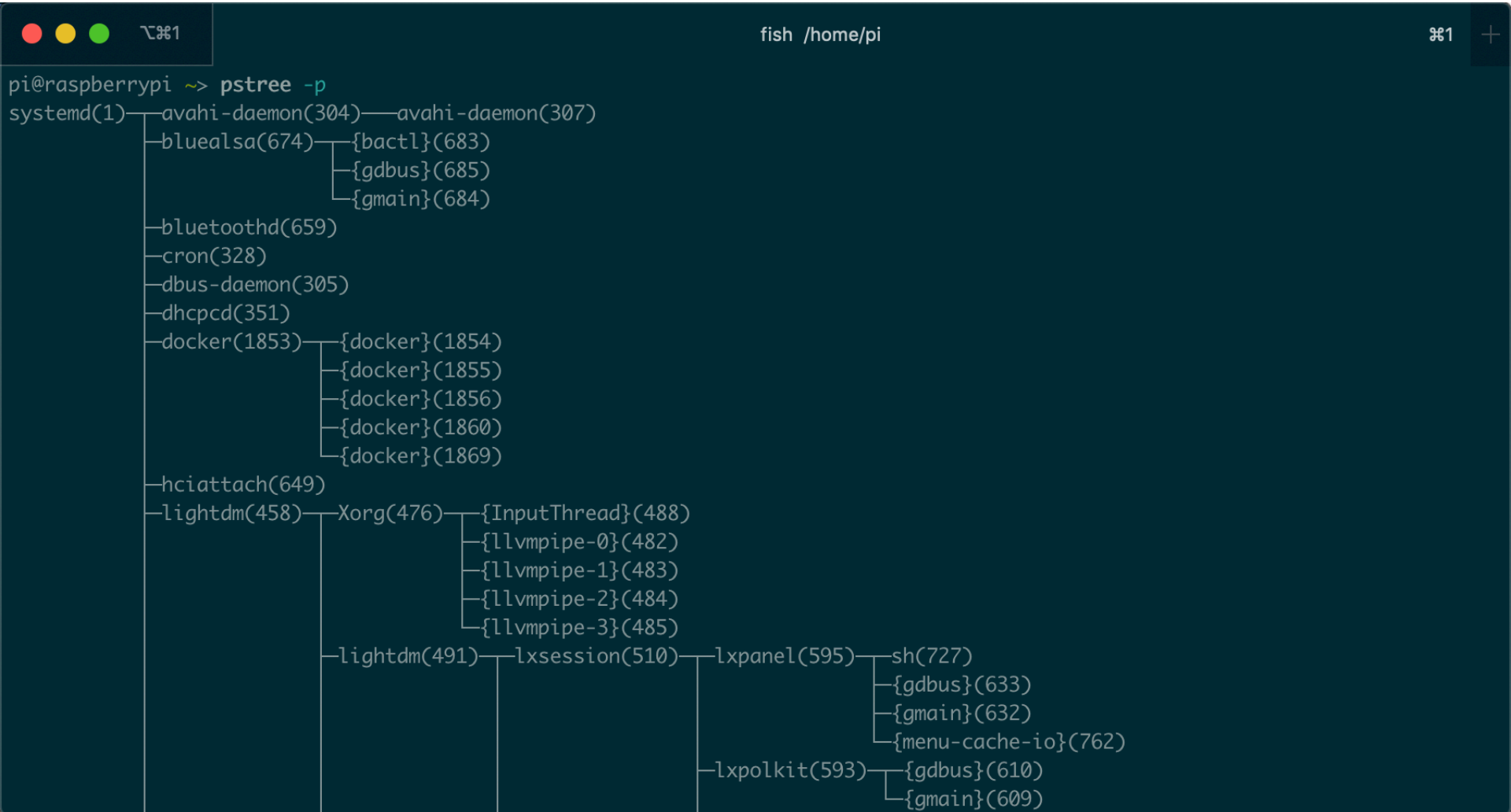


Outline

- Hardware review
- What is process?
  - Process vs Program
  - Linux Process Control Block
- Process related System calls
  - Fork
- Process on Distributed OSes

Where do processes come from?

- Process creation always uses fork() system call



Where do processes come from? First process in the kernel

```
asmlinkage __visible void __init start_kernel(void)
{
        char *command_line;
        char *after_dashes;

        set_task_stack_end_magic(&init_task);
        smp_setup_processor_id();
        debug_objects_early_init();

        cgroup_init_early();

        local_irq_disable();
        early_boot_irqs_disabled = true;

        /*
         * Interrupts are still disabled. Do necessary setups, then
         * enable them.
         */
        boot_cpu_init();
        page_address_init();
        pr_notice("%s", linux_banner);
        setup_arch(&command_line);
```

https://elixir.bootlin.com/linux/latest/source/init/main.c#L551

- The start of linux kernel begins from start_kernel() function, it is equal to the main function of kernel
- set_task_stack_end_magic() creates the first process in the OS
- The first process is the only one which is not created by fork function

Fork() system call

- Process creation always uses fork() system call
- When?
  - User runs a program at command line
    - $./test.o
  - OS creates a process to provide a service
    - Timer, networking, load-balance, daemon, etc.
  - One process starts another process
    - Parents and child process

Fork() system call

https://github.com/kevinsuo/CS7172/blob/master/fork.c

1. vim /home/pi/Downloads (ssh)

vim /home/pi/Dow... ⌘1

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(void) {
        int pid;
        pid = fork();

        if (pid < 0) {
                printf("Fork error!");
        } else if (pid == 0) {
                printf("The child pid is %d, pid:%d\n", getpid(), pid);
        } else {
                printf("The parent pid is %d, pid:%d\n", getpid(), pid);
        }

        return 0;
}
```

Fork() system call

- fork() is called once. But it returns twice!!
  - Once in the parent (return child id)
  - Once in the child (return 0)

Fork() system call

https://github.com/kevinsuo/CS7172/blob/master/fork.c

1. vim /home/pi/Downloads (ssh)

vim /home/pi/Dow... ⌘1

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(void) {
        int pid;
        pid = fork();

        if (pid < 0) {
                printf("Fork error!");
        } else if (pid == 0) {
                printf("The child pid is %d, pid:%d\n", getpid(), pid);
        } else {
                printf("The parent pid is %d, pid:%d\n", getpid(), pid);
        }

        return 0;
}
```

fork() is called once.
But it returns twice!!

```
pi@raspberrypi ~/Downloads> ./test.o
The parent pid is 2510, pid:2511
The child pid is 2511, pid:0
```

Fork() system call

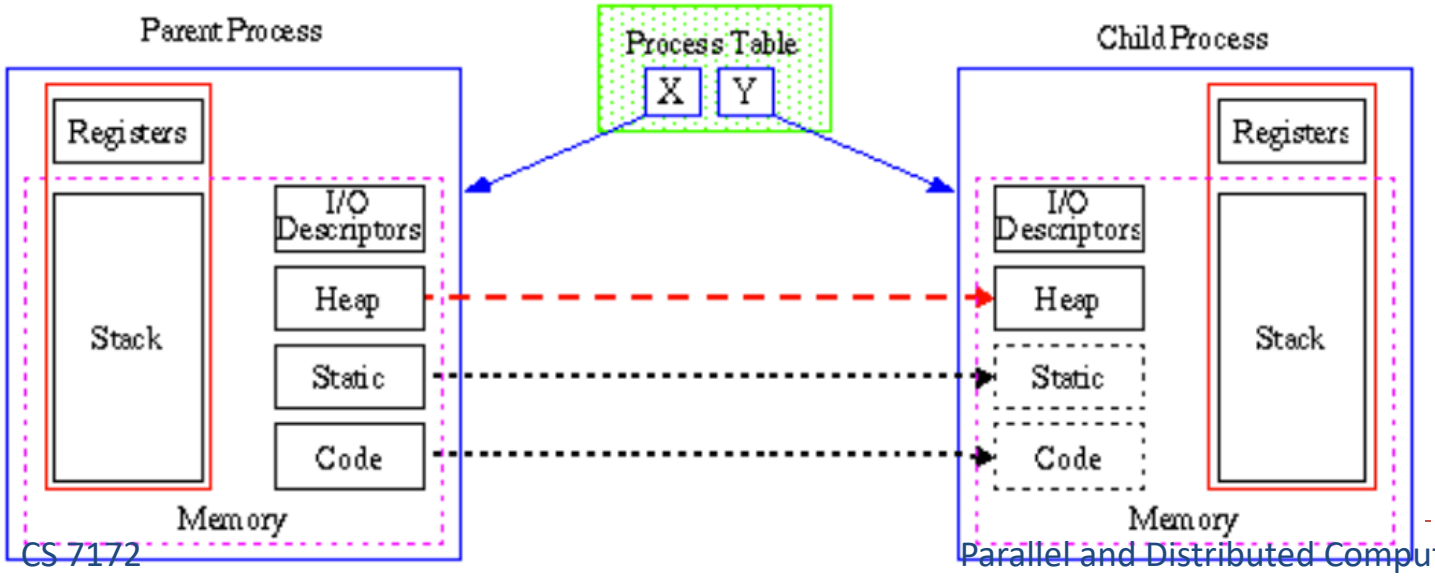
- fork() is the UNIX system call that creates a new process.

- fork () creates a new process that is a **copy** of the calling process.

- After fork () we refer to the caller as the **parent** and the newly-created process as the **child**. They have a special relationship and special responsibilities.

Parent process and child process

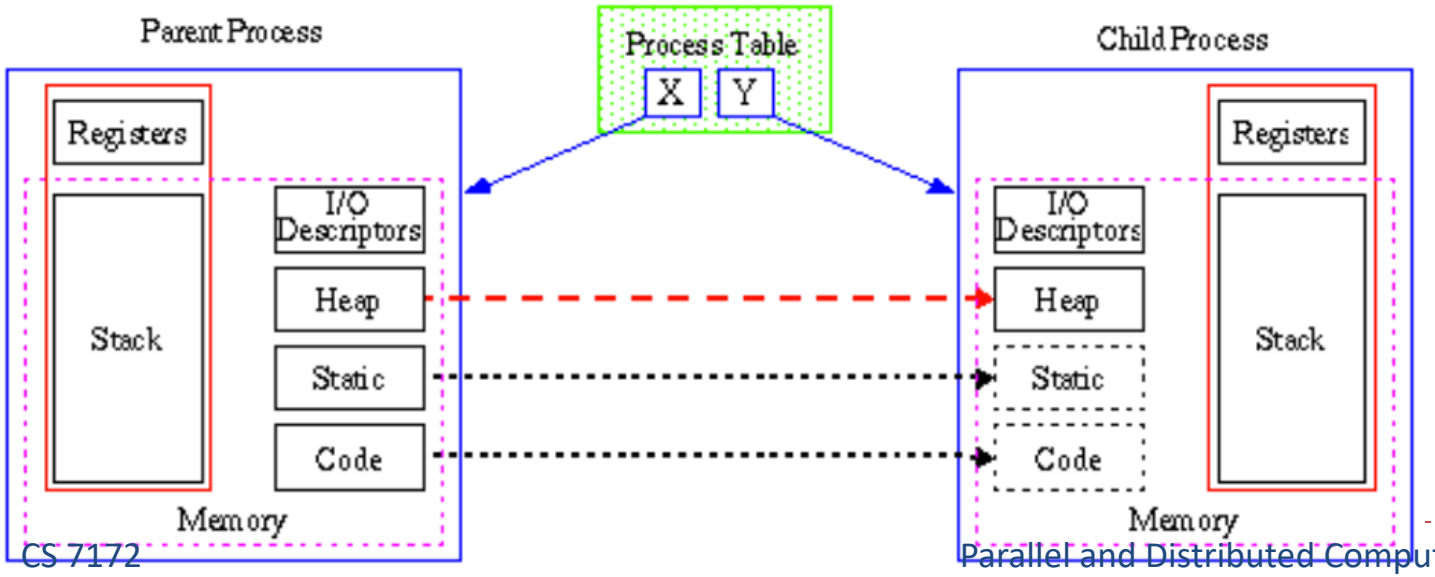
- When a parent process uses fork() to create a child process, the two processes have
  - the **same** program text.
  - but **separate** copies of the data, stack, and heap segments.



Parent process and child process

- The child's stack, data, and heap segments are **initially** exact **duplicates** of the corresponding parts the parent's memory.

- After the fork(), each process can modify the variables in its **own** data, stack, and heap segments without affecting the other process.



Fork() example

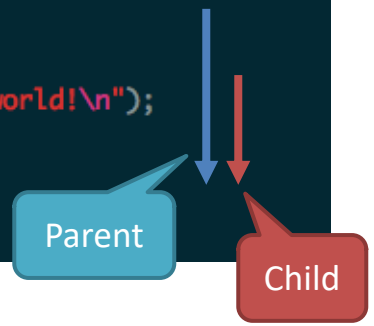
1. vim /home/pi/Downlo

vim /home/pi/Dow... ⌘1

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();

        printf("Hello world!\n");

        return 0;
}
```




```
pi@raspberrypi ~/Downloads> ./a.o
Hello world!
Hello world!
```

1. vim /home/pi/Dowr

vim /home/pi/Dow... ⌘1

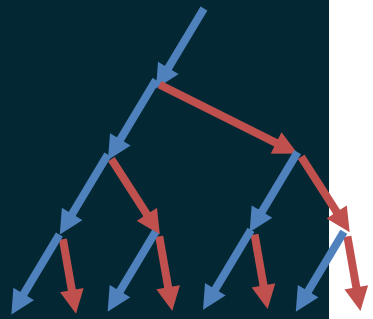
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();

        fork();

        fork();

        printf("Hello world!\n");

        return 0;
}
```



```
pi@raspberrypi ~/Downloads> ./a.o
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

A fork() bomb

- What does this code do?

```
while(1) {
        fork();
}
```

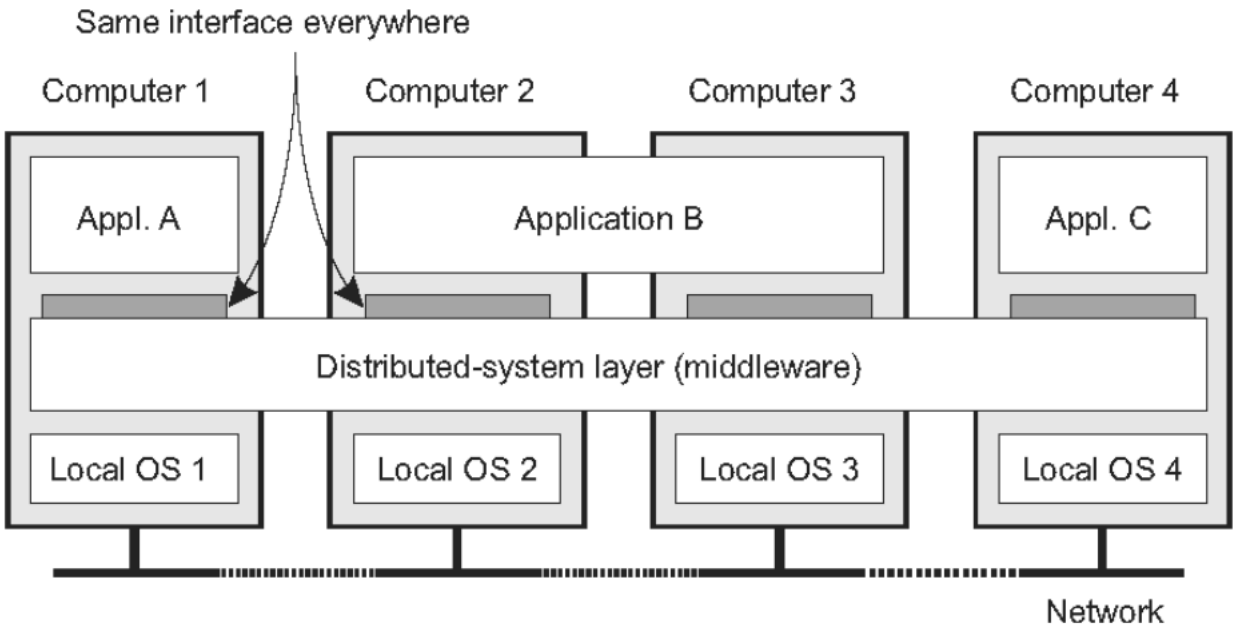
Agent smith

Outline

- Hardware review
- What is process?
  - Process vs Program
  - Linux Process Control Block
- Process related System calls
  - Fork
- Process on Distributed OSes

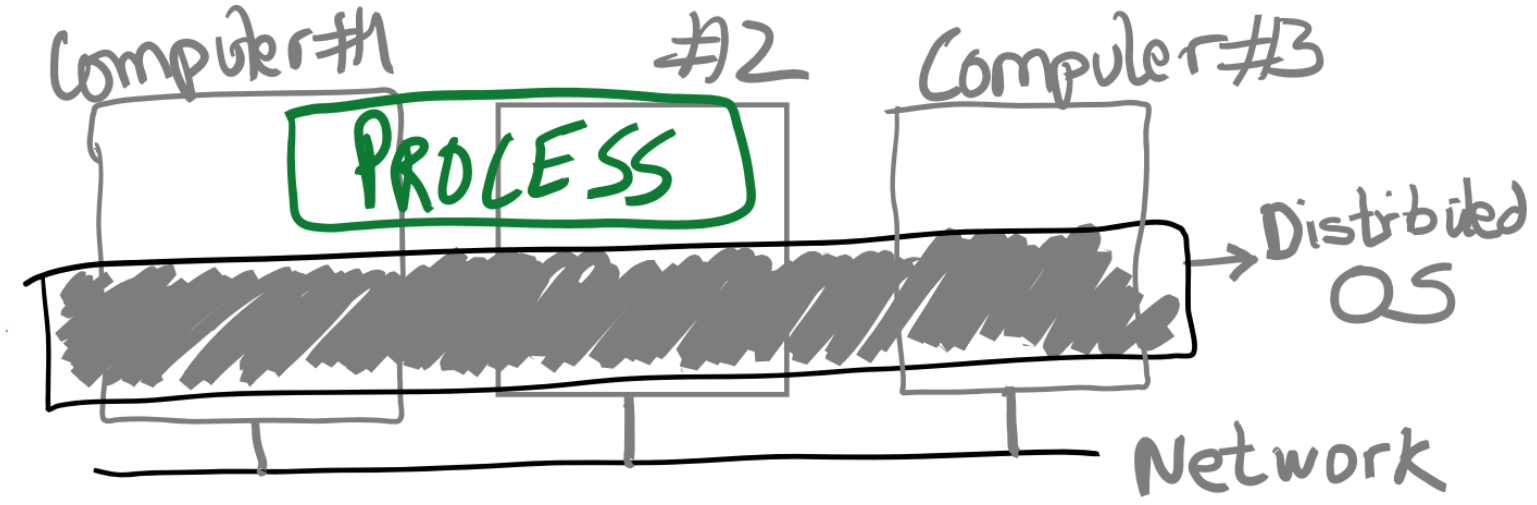
The OS of Distributed Systems

- Commonly used components and functions for distributed applications



The OS of Distributed Systems

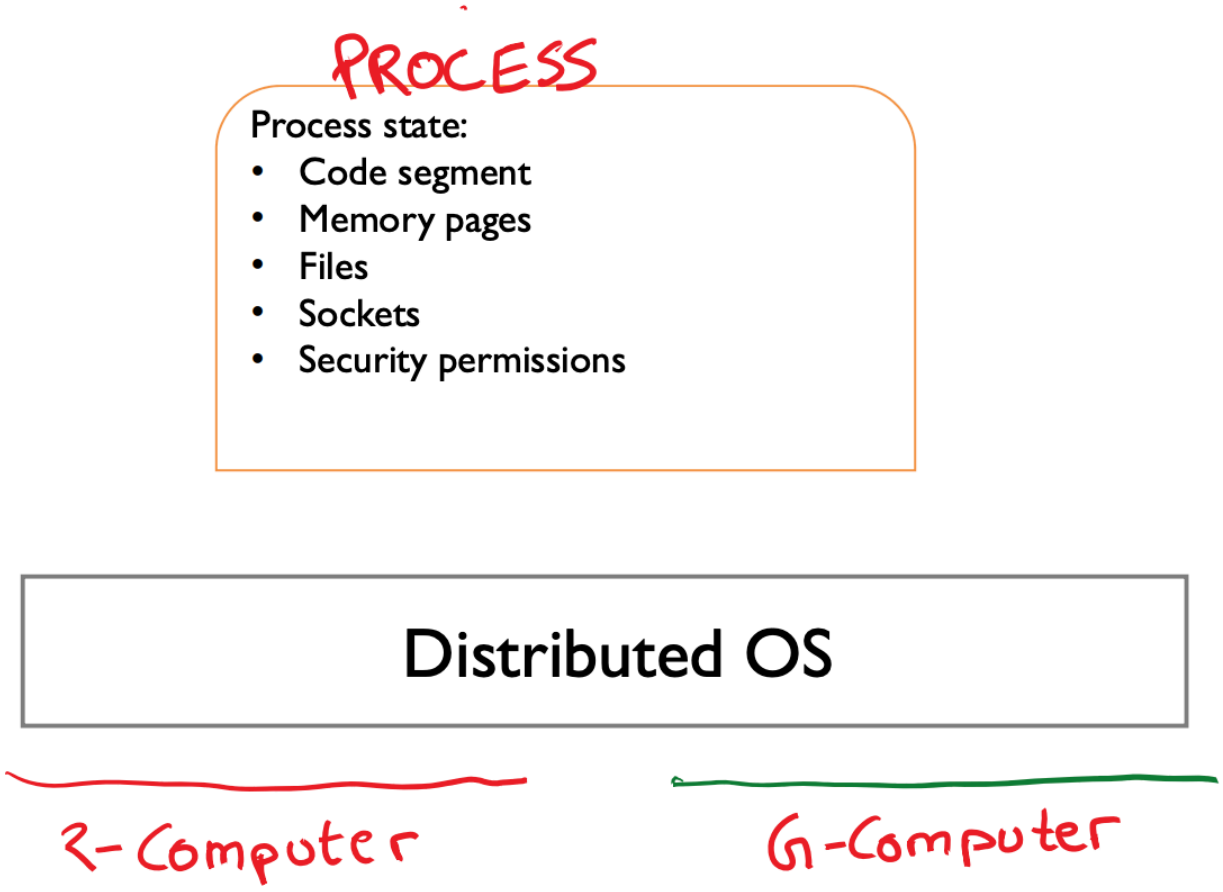
- An OS that spans multiple computers
- Same OS services, functionality, and abstractions as single-machine OS



Distributed OS Challenges

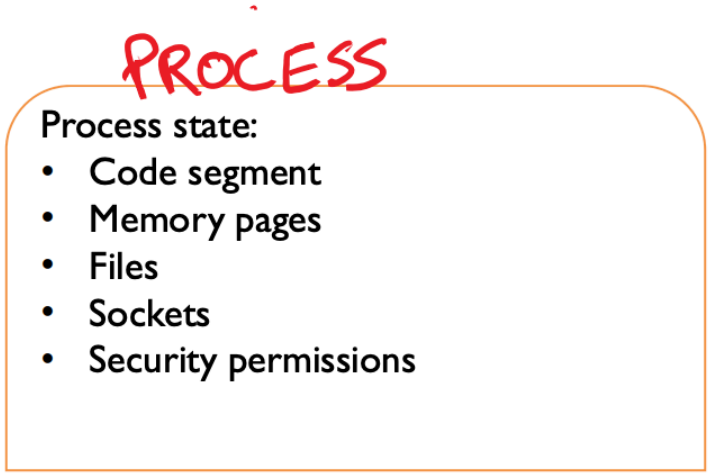
- Providing the process abstraction and resource virtualization is hard
- Resource virtualization must be transparent
  - But in distributed settings, there's always a distinction between local and remote resources
- In a single-machine OS, processes don't care where their resources are coming from:
  - Which CPU cores, when they are scheduled, which physical memory pages they use, etc.
- In fact, providing abstract, virtual resources is one of the main OS services

Processes In Distributed OS

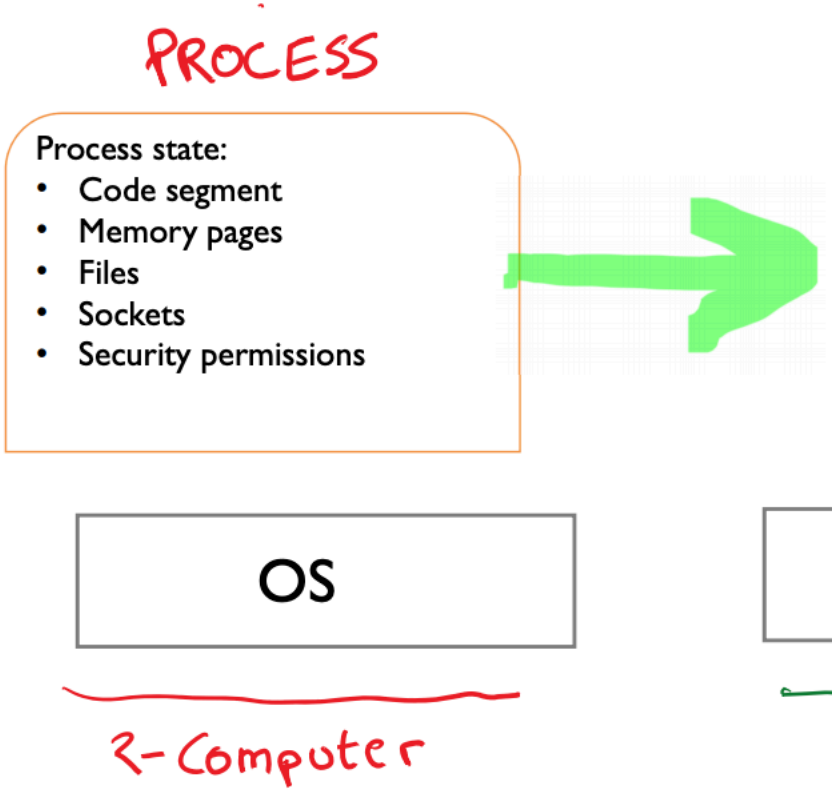


Transparency Issues In Distributed OS

- Where does code run?
- Which memory is used?
  Local vs. remote
- How are files accessed?

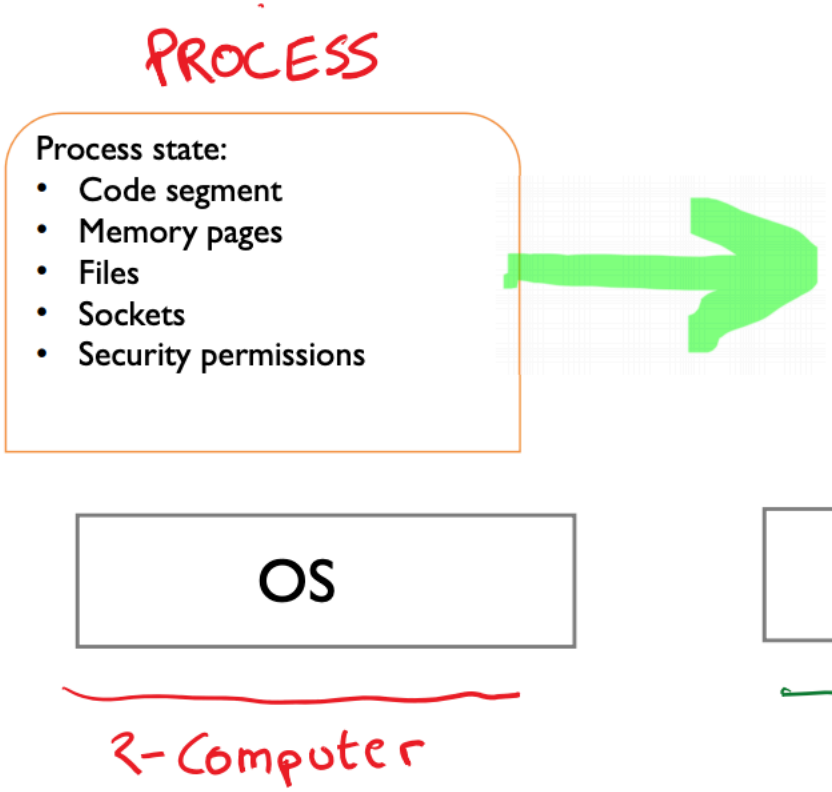


Process Migration



- Move all process state from one computer to another
- Process state can be vast
- Also entangled with other process states
  - ✓ Shared files?
  - ✓ IPC (pipes etc)

Process Migration



- Migrate some state
- Other state, if required, is accessed over the network
- Example: migrate only fraction of pages. Other pages are copied over the network on access.
- Can also be used to access remote hardware devices (GPUs)

Conclusion

- Hardware review
- What is process?
  - Process vs Program
  - Linux Process Control Block
- Process related System calls
  - Fork
- Process on Distributed OSes