CMSC421: Principles of Operating Systems

Nilanjan Banerjee

Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

Announcements

- Project 0 and Homework 1 are due this week
- Readings from Silberchatz [2nd chapter]
    - Section 2.3

Lets write a system call in the kernel (sys_strcpy)

int strcpy(char *src, char *dest, int len)

can return values of
size of at most long? Why?

asmlinkage long sys_strcpy(char *src, char *dest, int len)

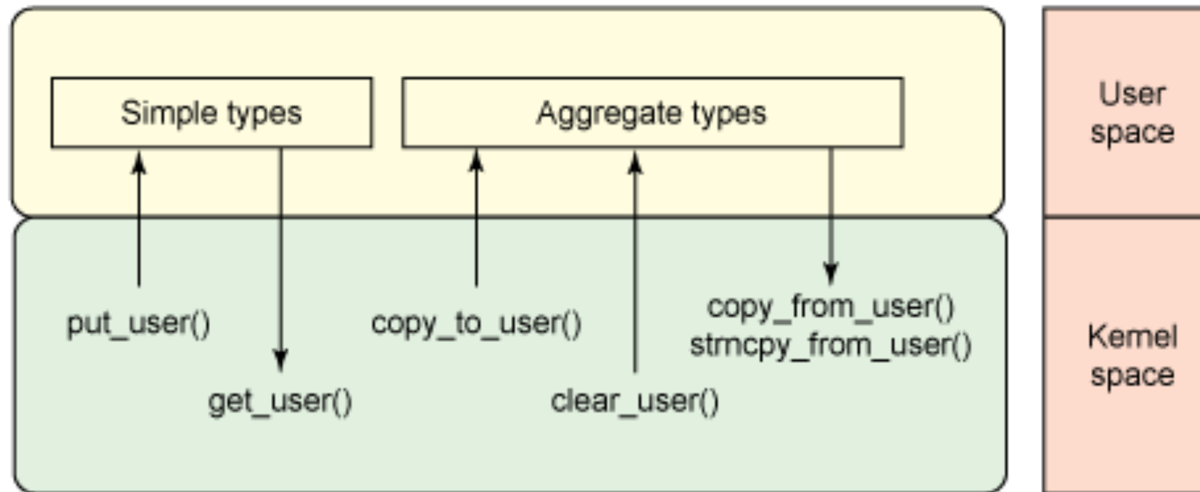
compiler directive
params will be read from stack

Issues to think about when writing system calls

- Moving data between the kernel and user process
  - Concerns: security and protection

- Synchronization and concurrency **(will revisit)**
  - Several (so called) kernel threads might be accessing the same data structure that you want to read/write
  - Simple solution (disable interrupts “cli”)
    - Usually not a good idea
  - Big problem in preemptive CPU (which is almost every CPU) and multi-processor systems
    - CONFIG_SMP or CONFIG_PREEMPT

Useful kernel API functions for bidirectional data movement

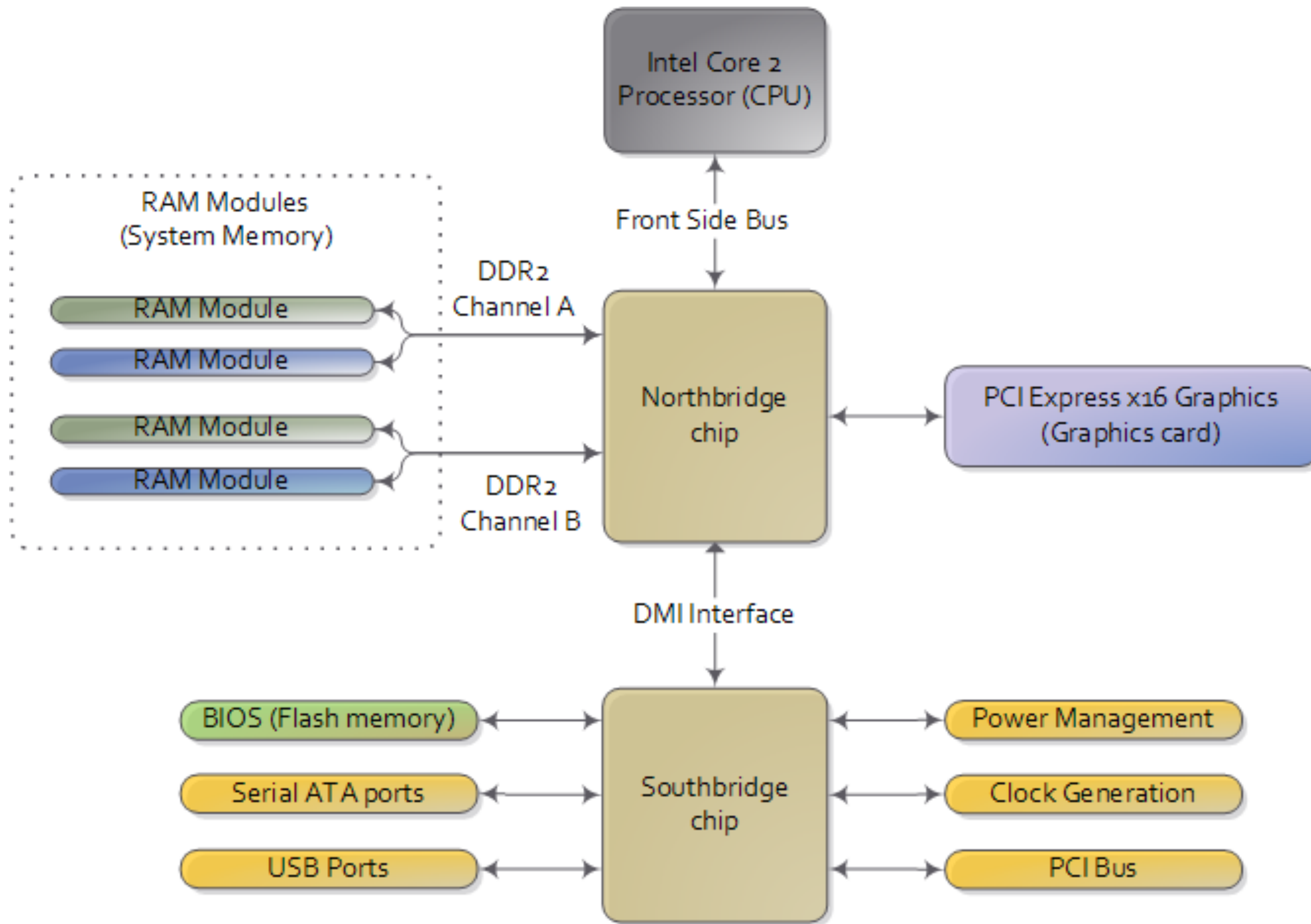


- *access_ok (type, addr, size)*: type (VERIFY_READ, VERIFY_WRITE)
- *get_user(x, ptr)* --- read a char or int from user-space
- *put_user(x, ptr)* --- write variable from kernel to user space
- *copy_to_user(to, from, n)* --- copy data from kernel to userspace
- *copy_from_user(to, from, n)* – copy data to kernel from userspace
- *strnlen_user(src, n)* – checks that the length of a buffer is n
- *strcpy_from_user(dest, src, n)* ---copies from kernel to user space

Acknowledgement: http://www.ibm.com/developerworks/linux/library/l-kernel-memory-access/index.html

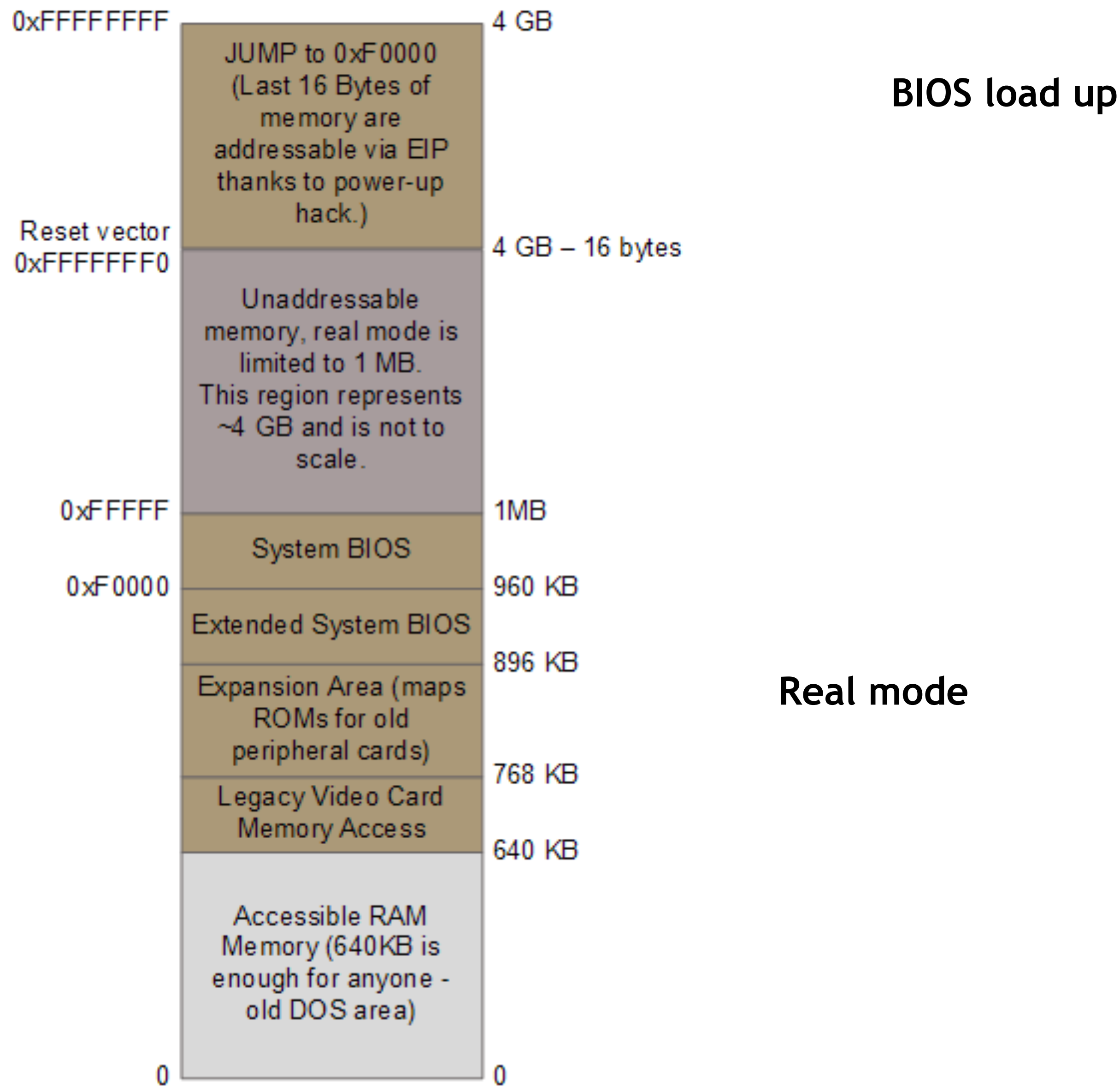
Linux Bootup process

Intel Motherboard



Acknowledgement:http://duartes.org/gustavo/blog/post/motherboard-chipsets-memory-map

Memory Organization during CPU bootup

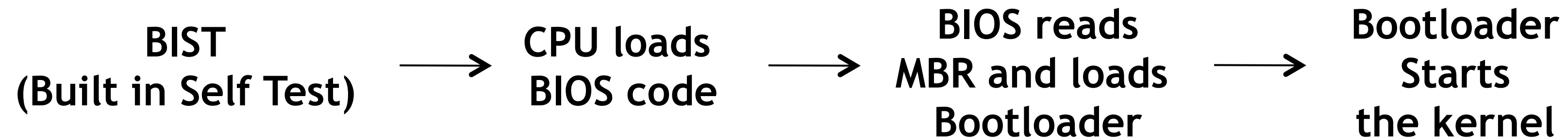


BIOS load up

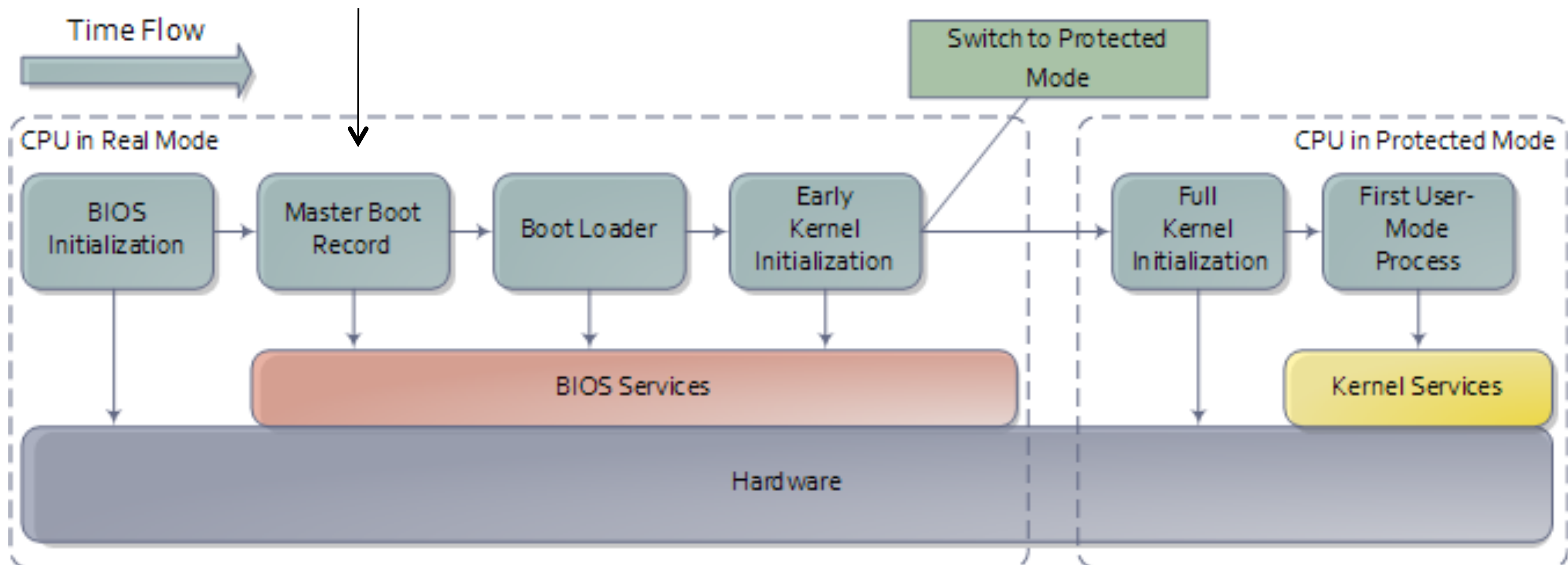
Real mode

Acknowledgement:http://duartes.org/gustavo/blog/post/how-computers-boot-up

Bootup Process

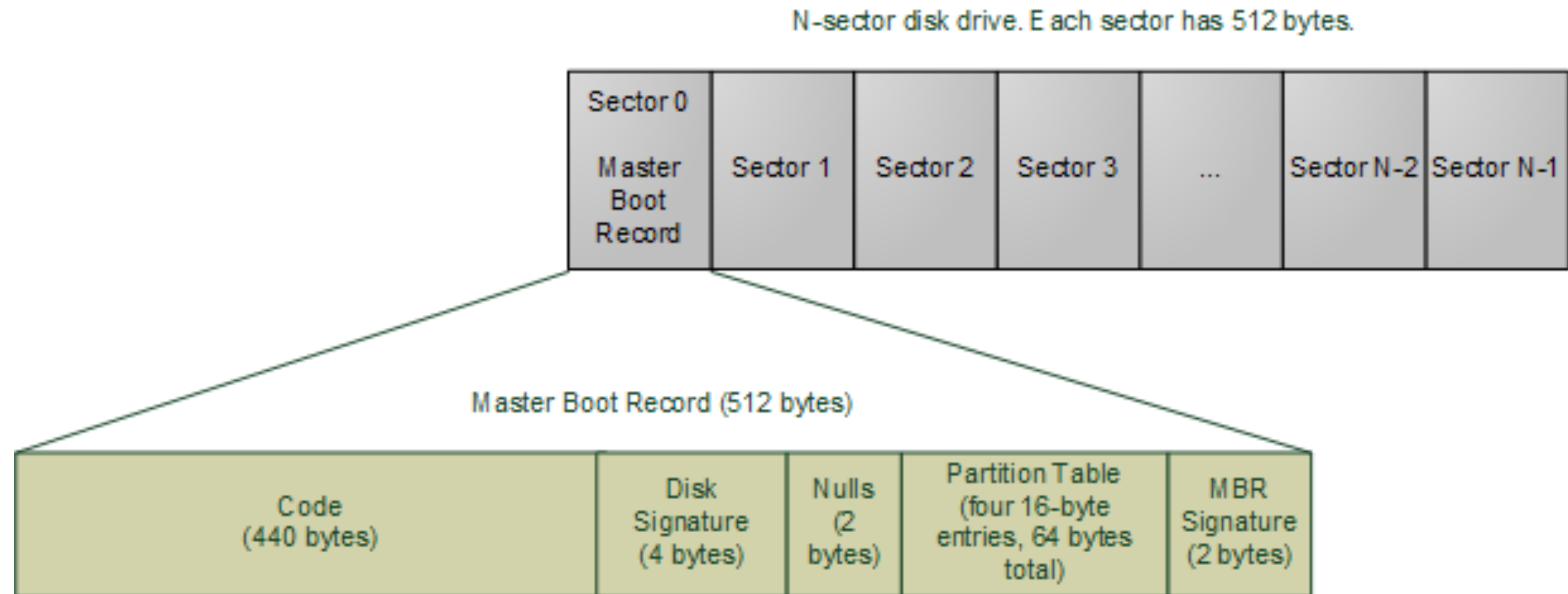


One disk page
(512 bytes)



Acknowledgement:http://duartes.org/gustavo/blog/post/how-computers-boot-up

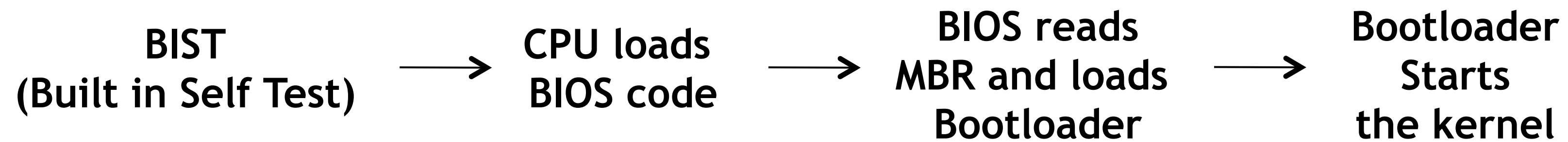
Reading the first disk sector



Boot loader Stage 1 (loads Stage 2)	Boot loader Stage 2 (presents users with OS options)	Boot loader Stage 3 (loads the OS)

Acknowledgement:http://duartes.org/gustavo/blog/post/how-computers-boot-up

Bootup Process



One disk page
(512 bytes)

Bootloader
(stage0, stage1, stage2)

Time Flow

Switch to Protected Mode

CPU in Real Mode

BIOS Initialization

Master Boot Record

Boot Loader

Early Kernel Initialization

BIOS Services

CPU in Protected Mode

Full Kernel Initialization

First User-Mode Process

Kernel Services

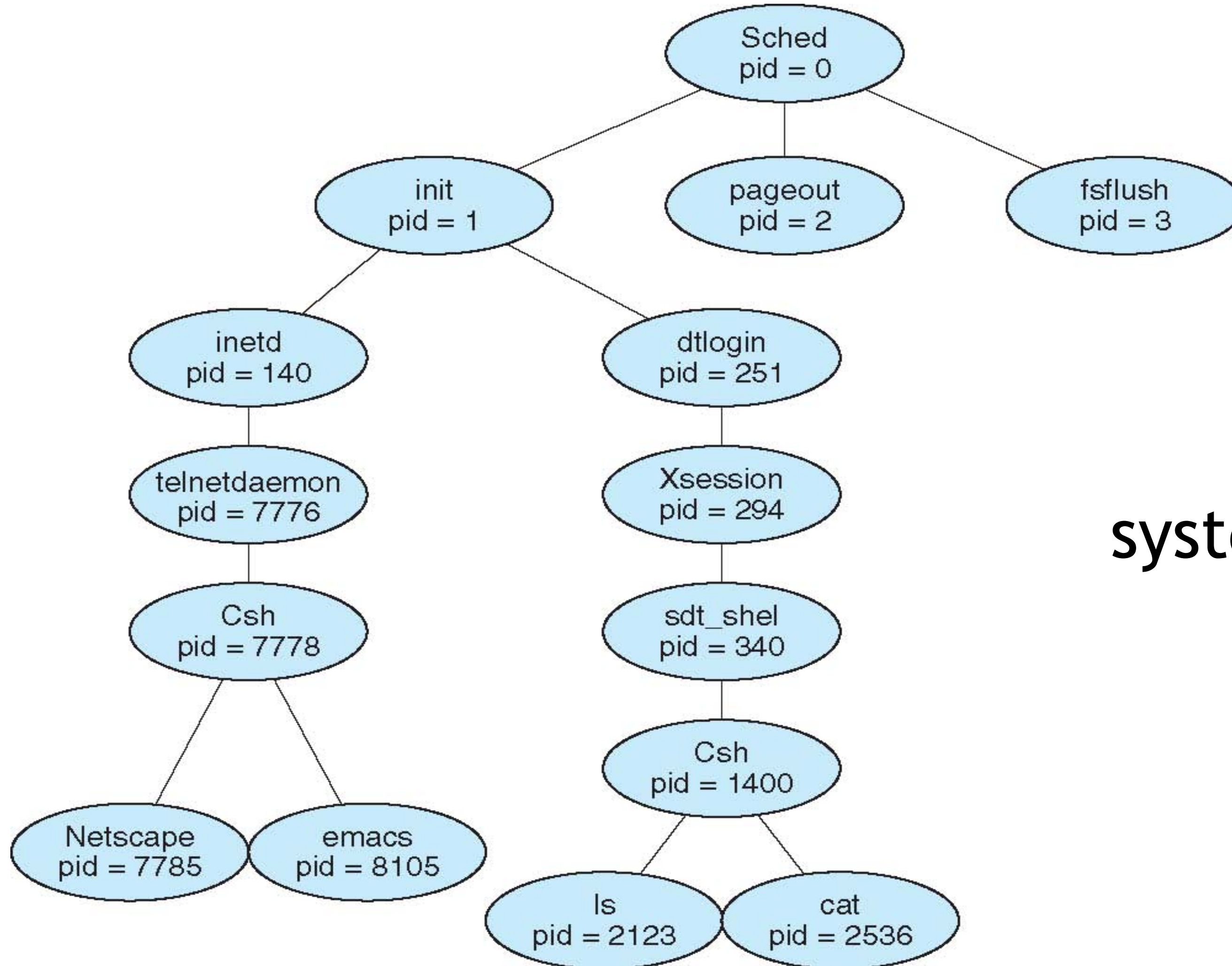
Hardware

Acknowledgement:http://duartes.org/gustavo/blog/post/how-computers-boot-up

Lets take a look at some code
(Coreboot, GRUB, Kernel)

Processes

Process Tree generation



system process tree

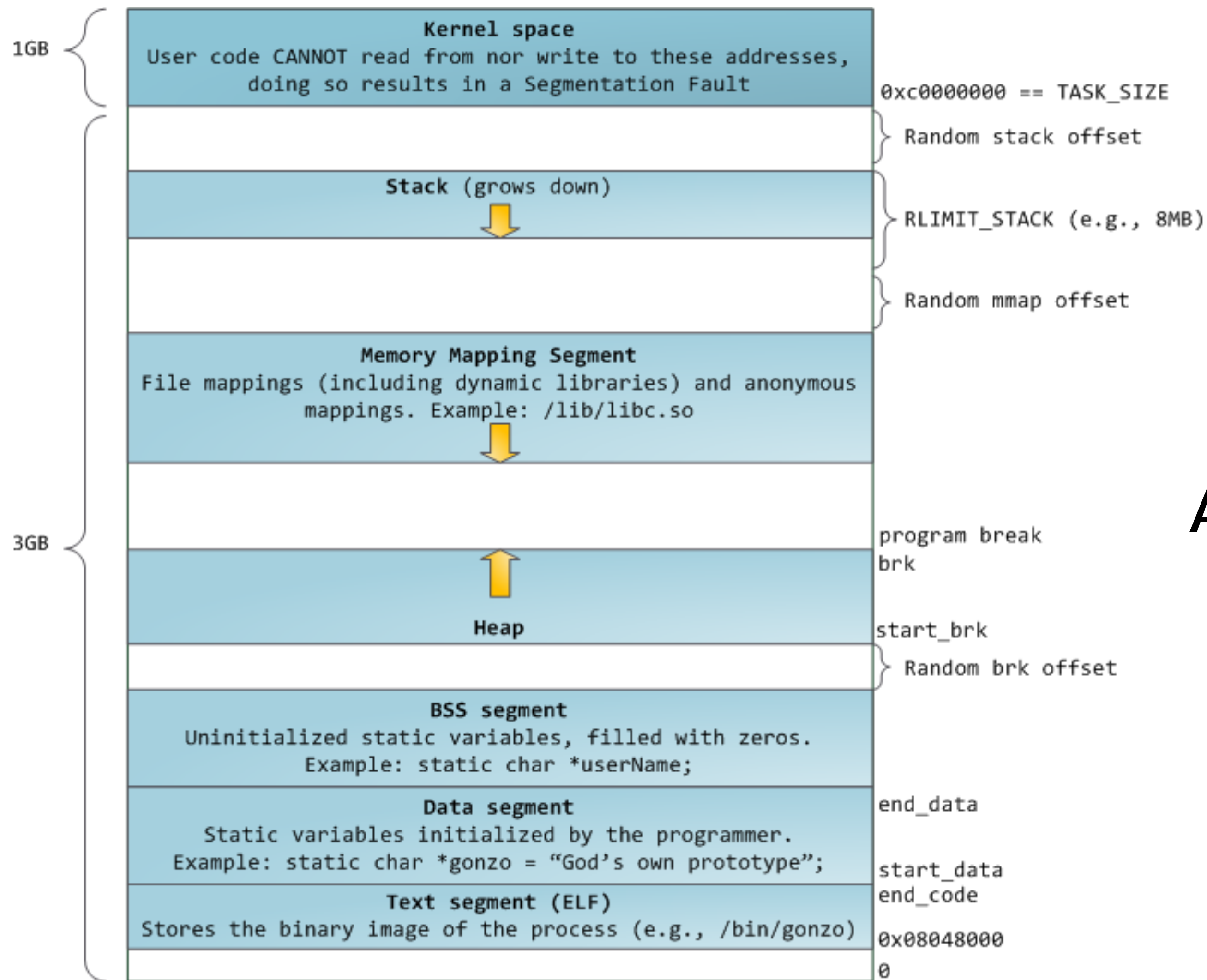
But what is a process?

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks

- Process – a program in execution; process execution must progress in sequential fashion

- A process includes:
  - program counter
  - stack
  - data section

Process Memory looks like.



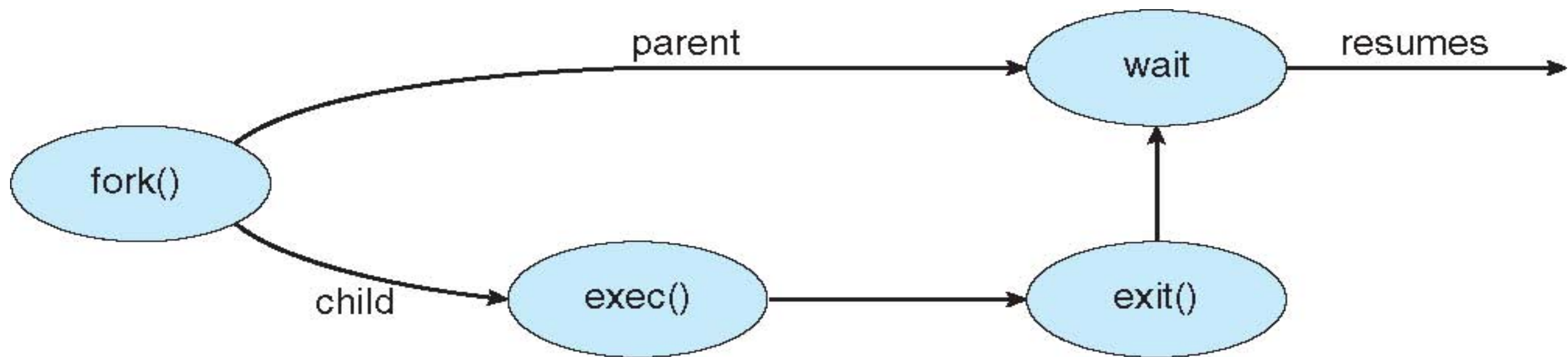
virtual
Address space

why virtual?

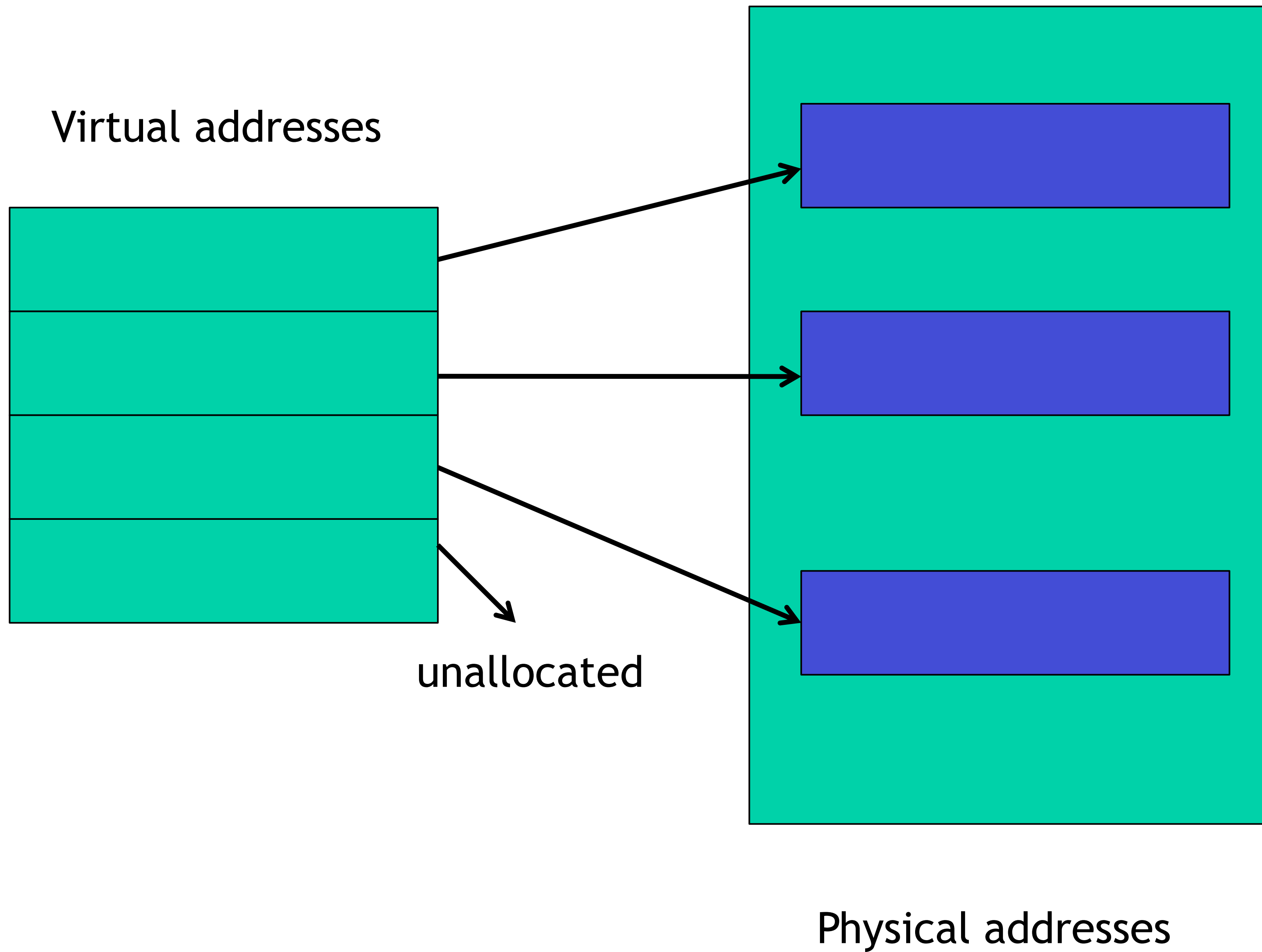
How do we create new processes in userland (fork)

Lets see a demo

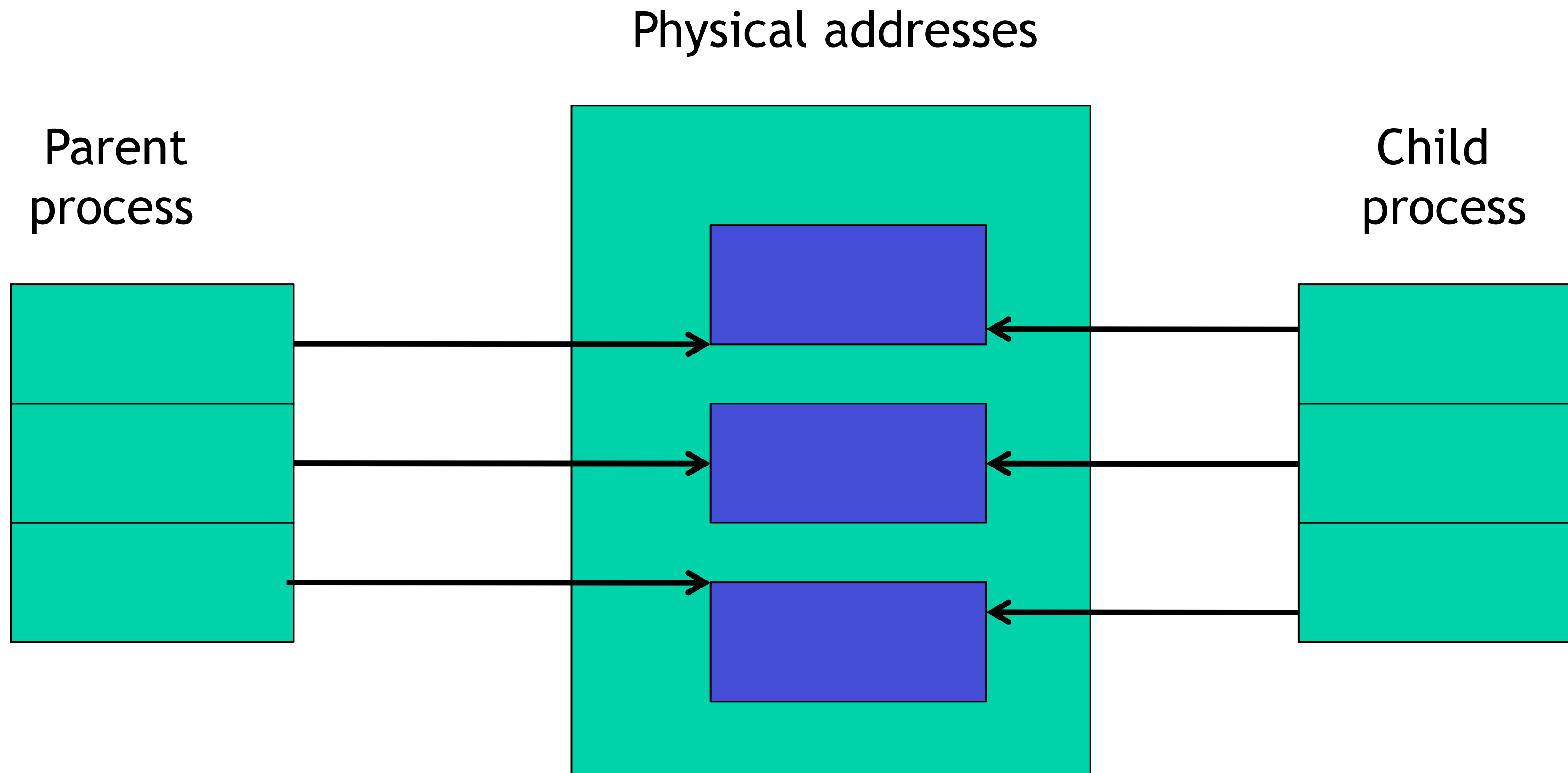
What is really happening here



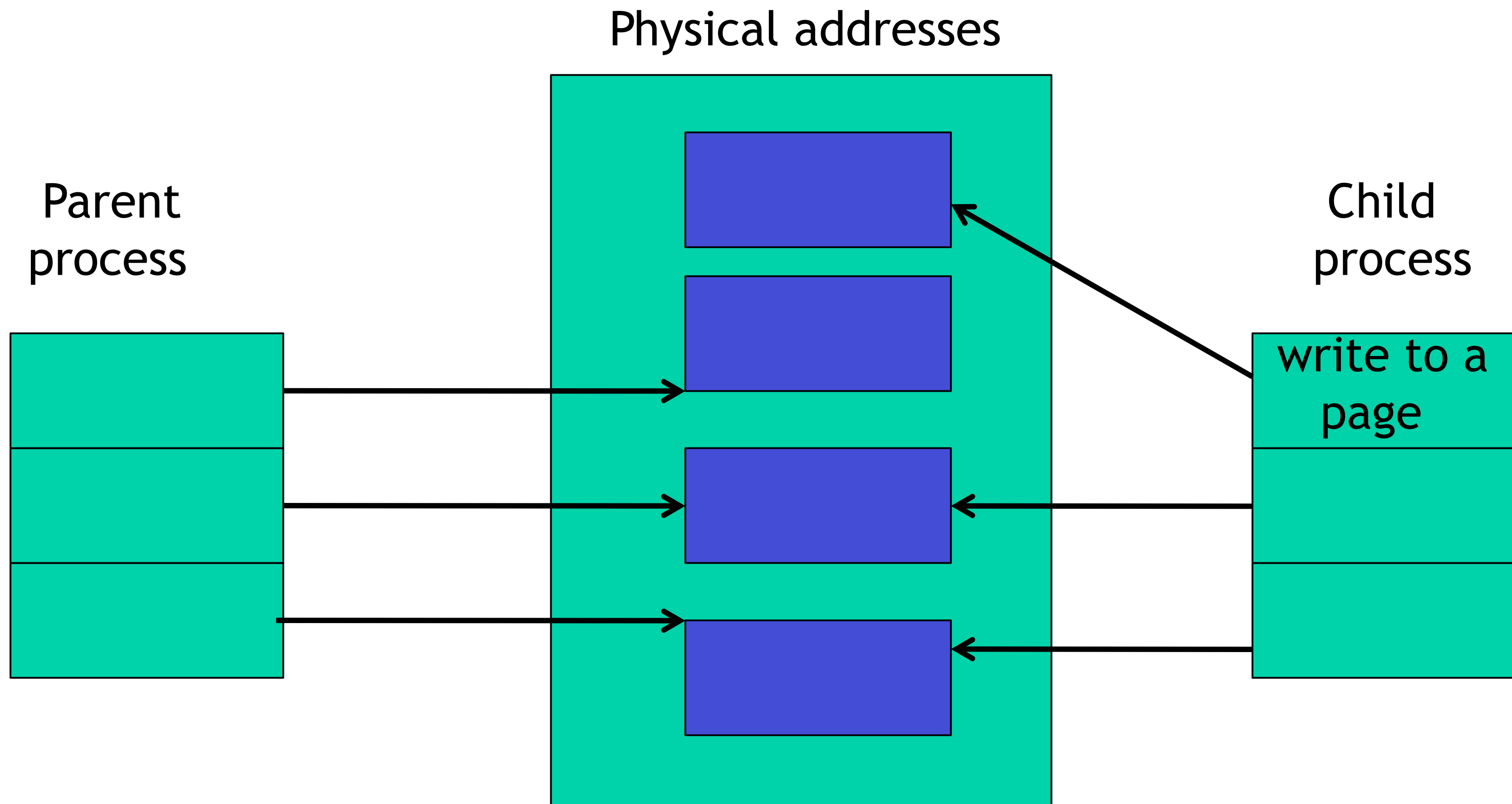
What does the memory structure look like before fork()



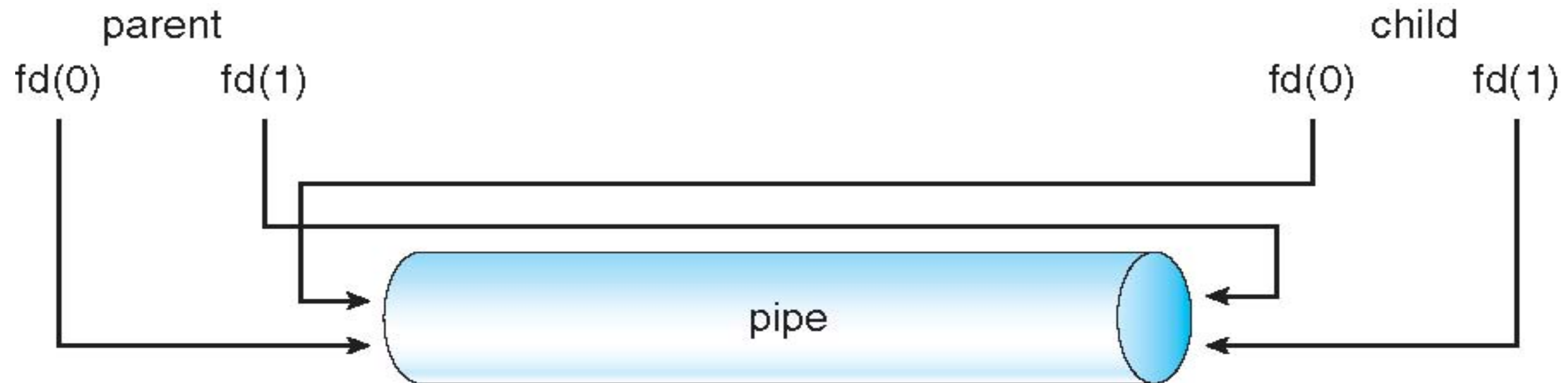
What does it look like after forking?



Fork() Copy-on-write policy



communication child/parent process (Unnamed pipes)




```
Pipe(fid); // where int fid[2] fid[0] is the read
    from the pipe and fid[1] is write to the pipe

dup2(oldfid, newfid) //creates an alias to oldfid
//very handy when you do not want to use file
    descriptors and use standard ones
```

Kernel data structure for processes (PCB)

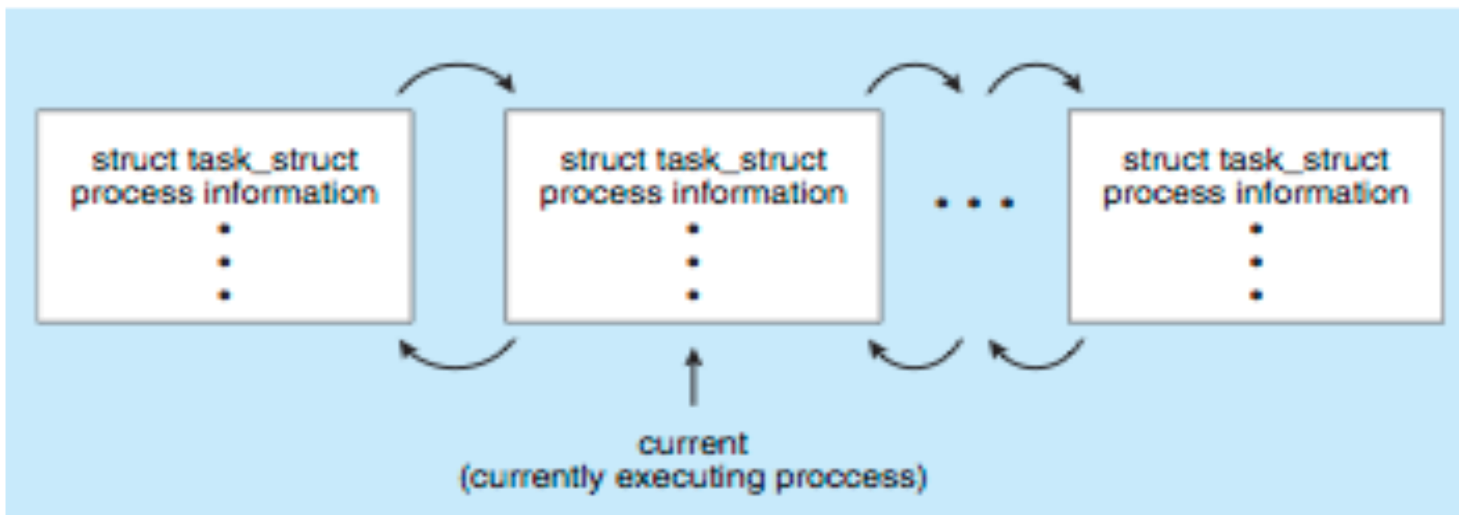
Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Kernel data structure for processes (PCB)

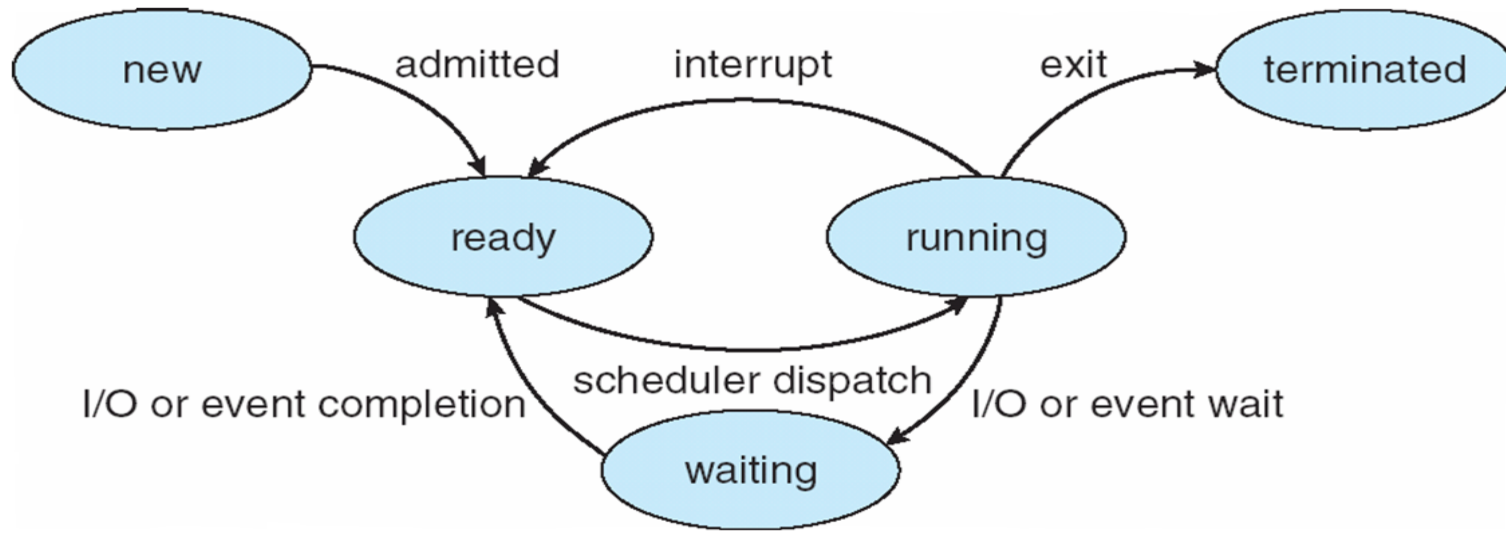
- Represented by the C structure `task_struct`

```
pid t pid; /* process identifier */
long state; /* state of the process */
unsigned int time slice /* scheduling information */
struct task struct *parent; /* this process's parent */
struct list head children; /* this process's children */
struct files struct *files; /* list of open files */
struct mm struct *mm; /* address space of this pro */
```



Process States

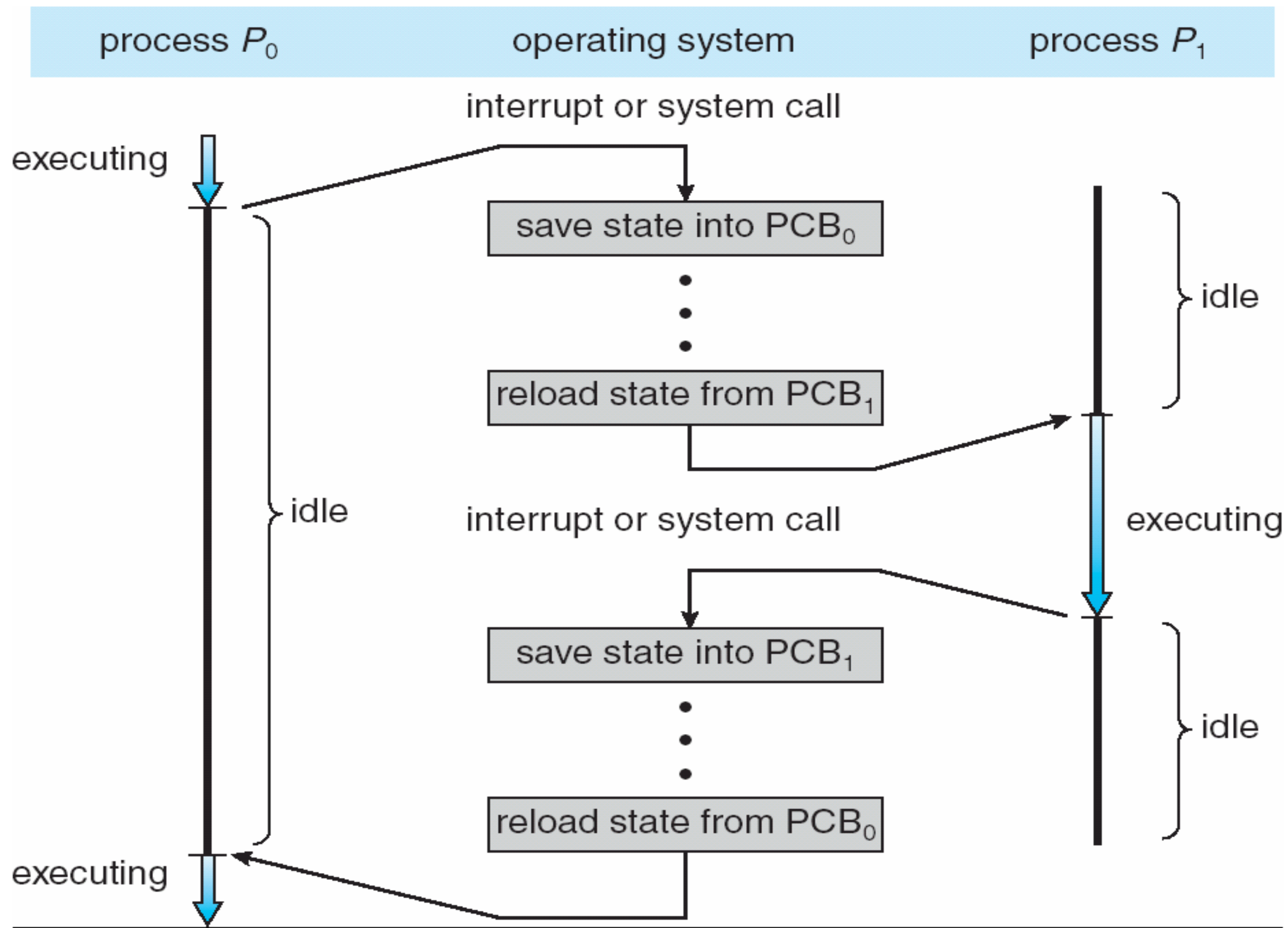
- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution



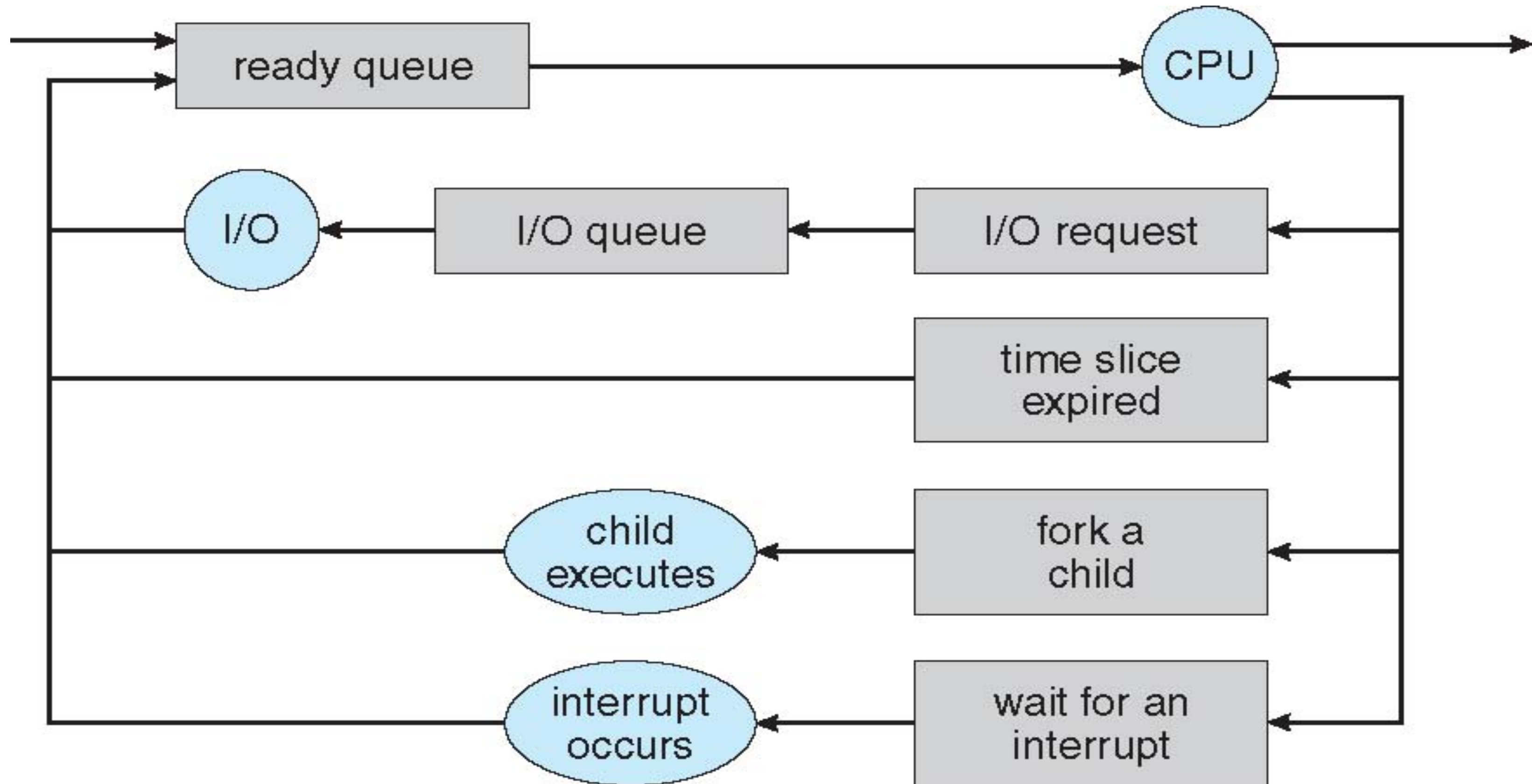
Process Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch.
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

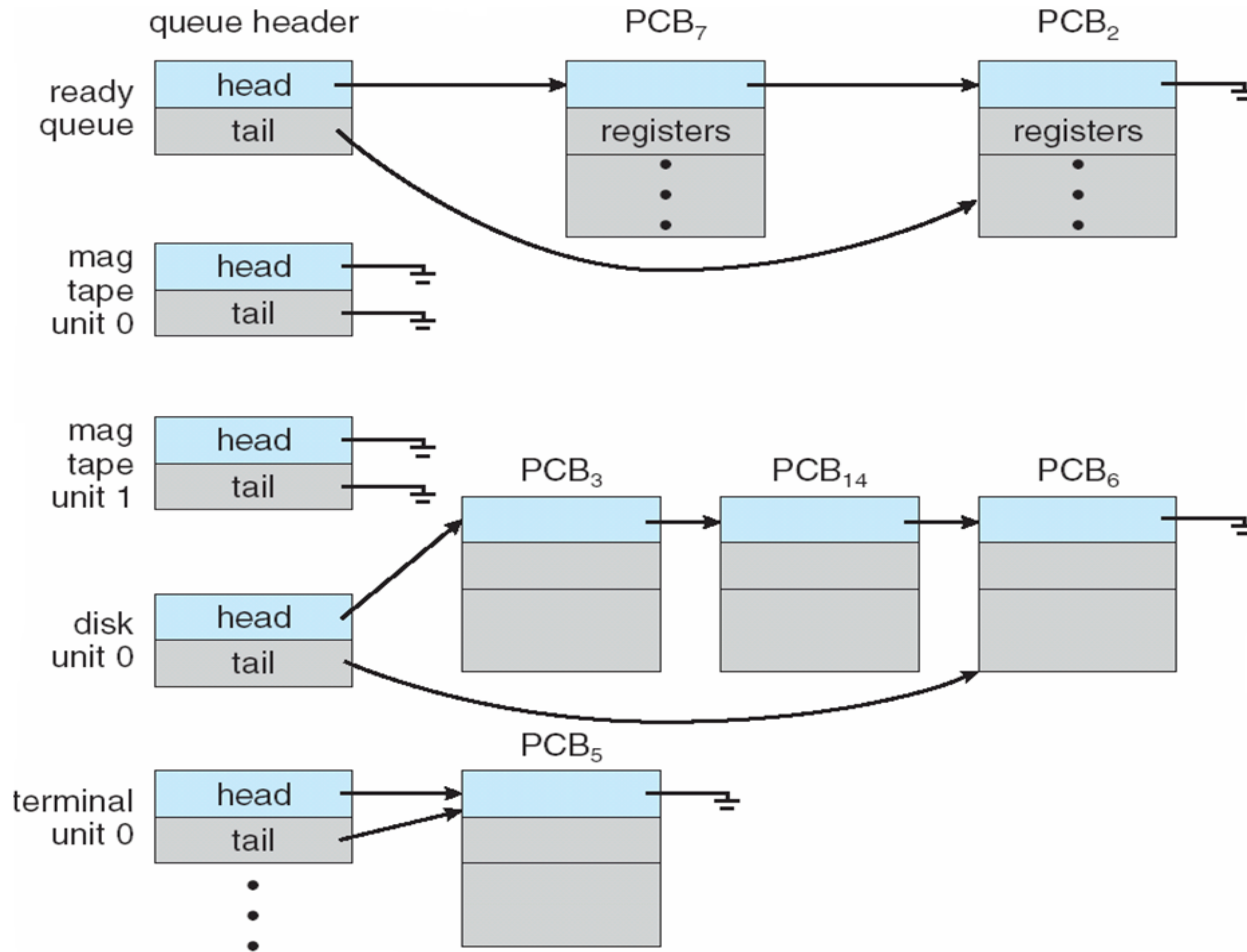
Process Context Switch



Process Scheduling



Process Queues



Lets take a kernel dive to study
the process data structure and fork() system call

Next class

- Process management
    - Inter-process communication (Named pipes, shared memory (shmget, mmap), message passing)
    - Intro to threads

An in-class discussion (a bit-hack)