CMSC421: Principles of Operating Systems

Nilanjan Banerjee

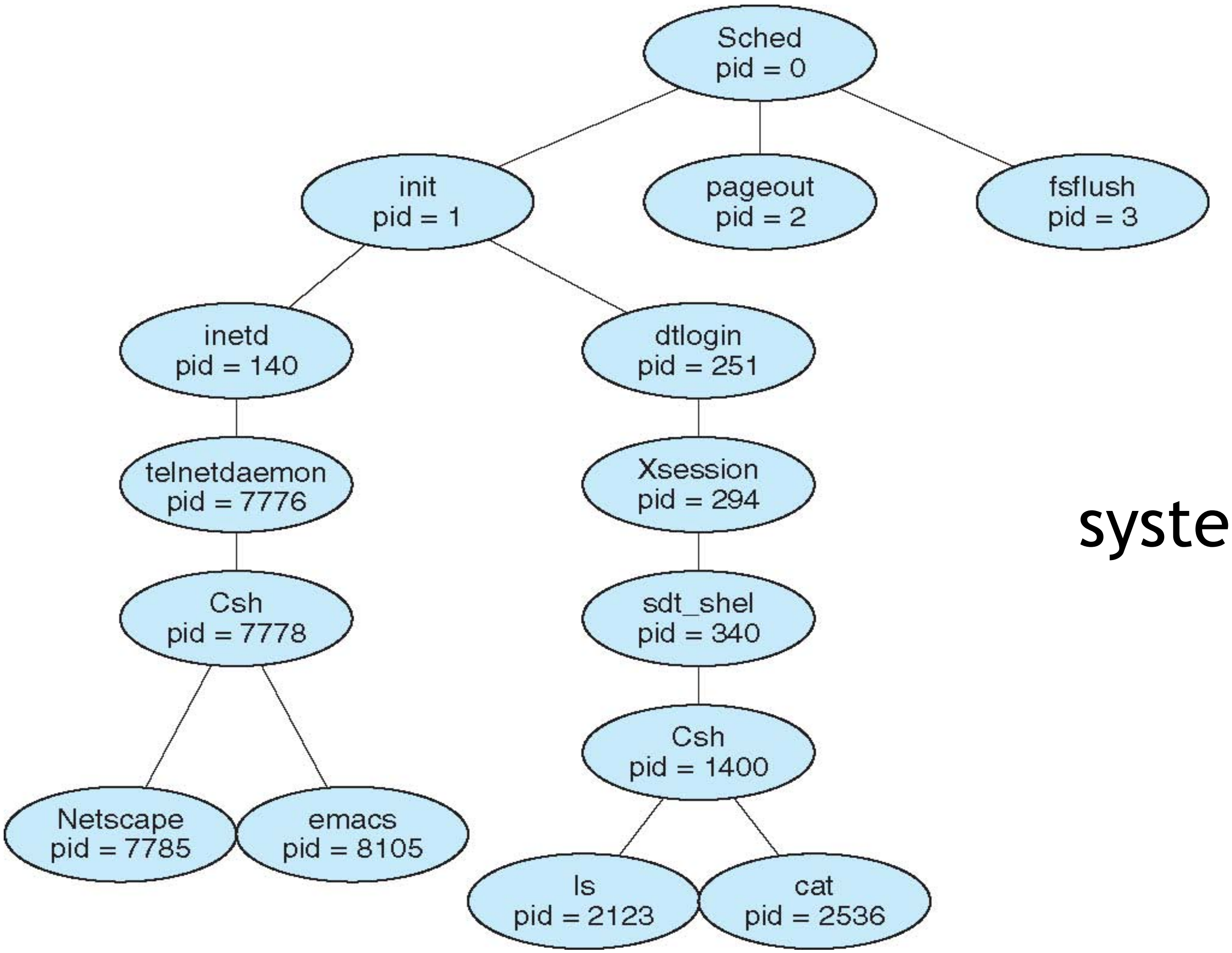
Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

Announcements

- Readings from Silberchatz [3rd chapter]
- Late Policy

Processes

Process Tree generation



system process tree

But what is a process?

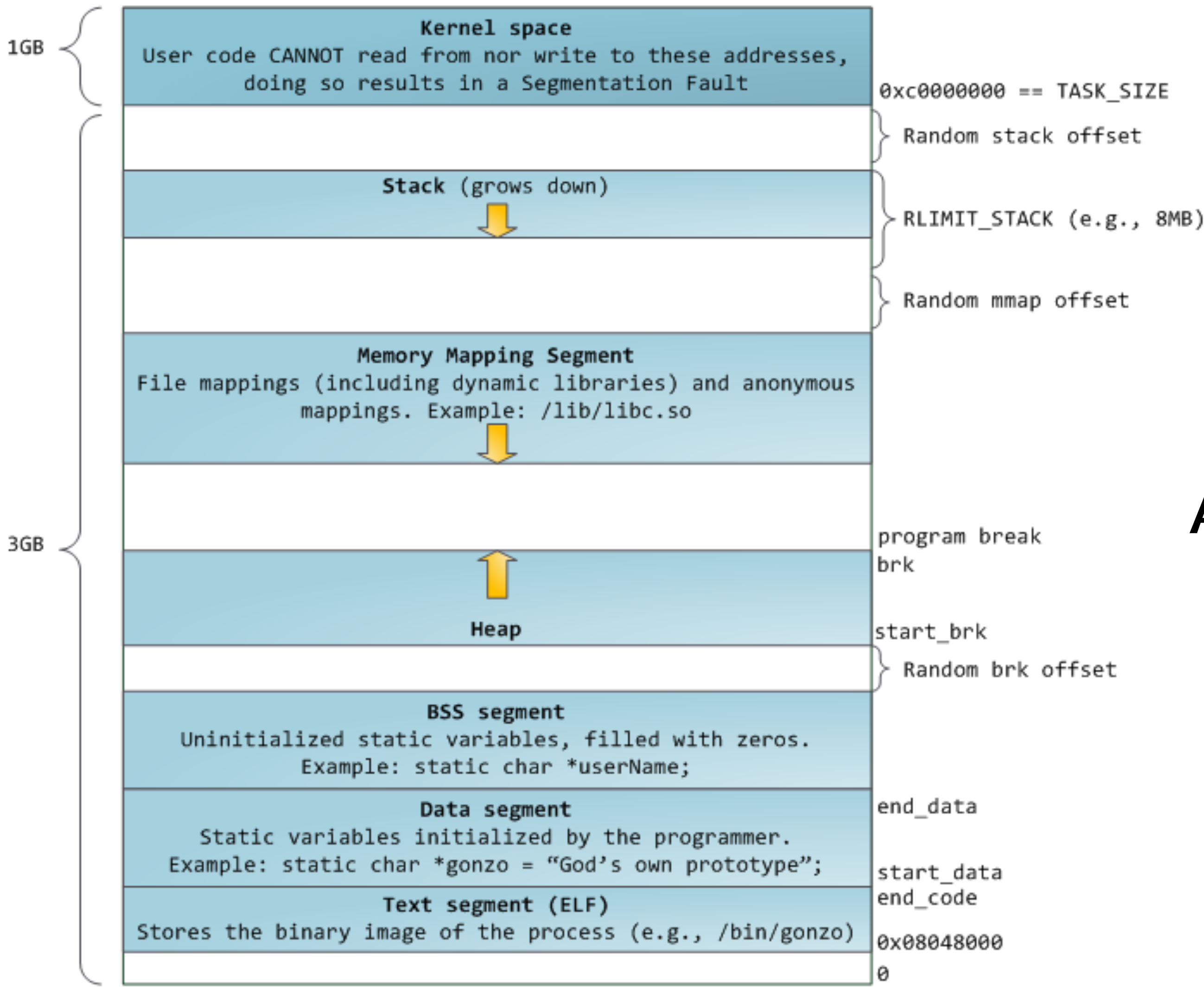
- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks

- Textbook uses the terms *job* and *process* almost interchangeably

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

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

Process Memory looks like.

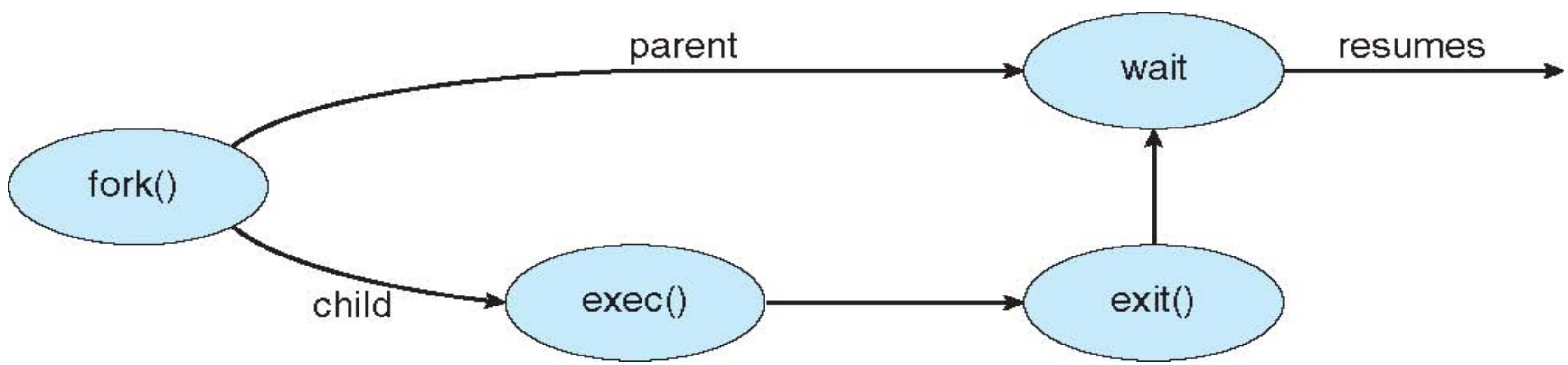


virtual
Address space

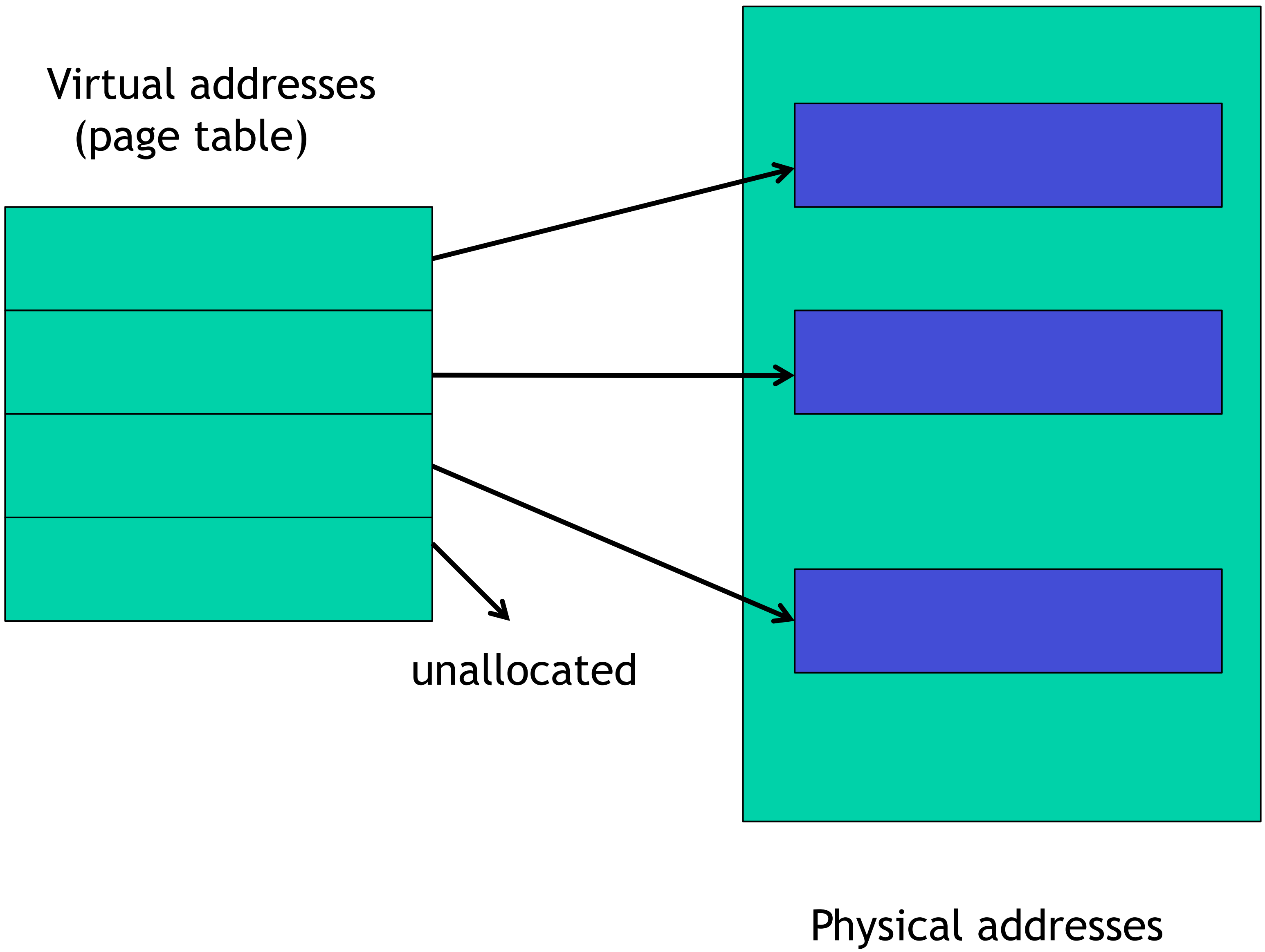
why virtual?

How do we create new processes in userland (fork)
Lets see a demo

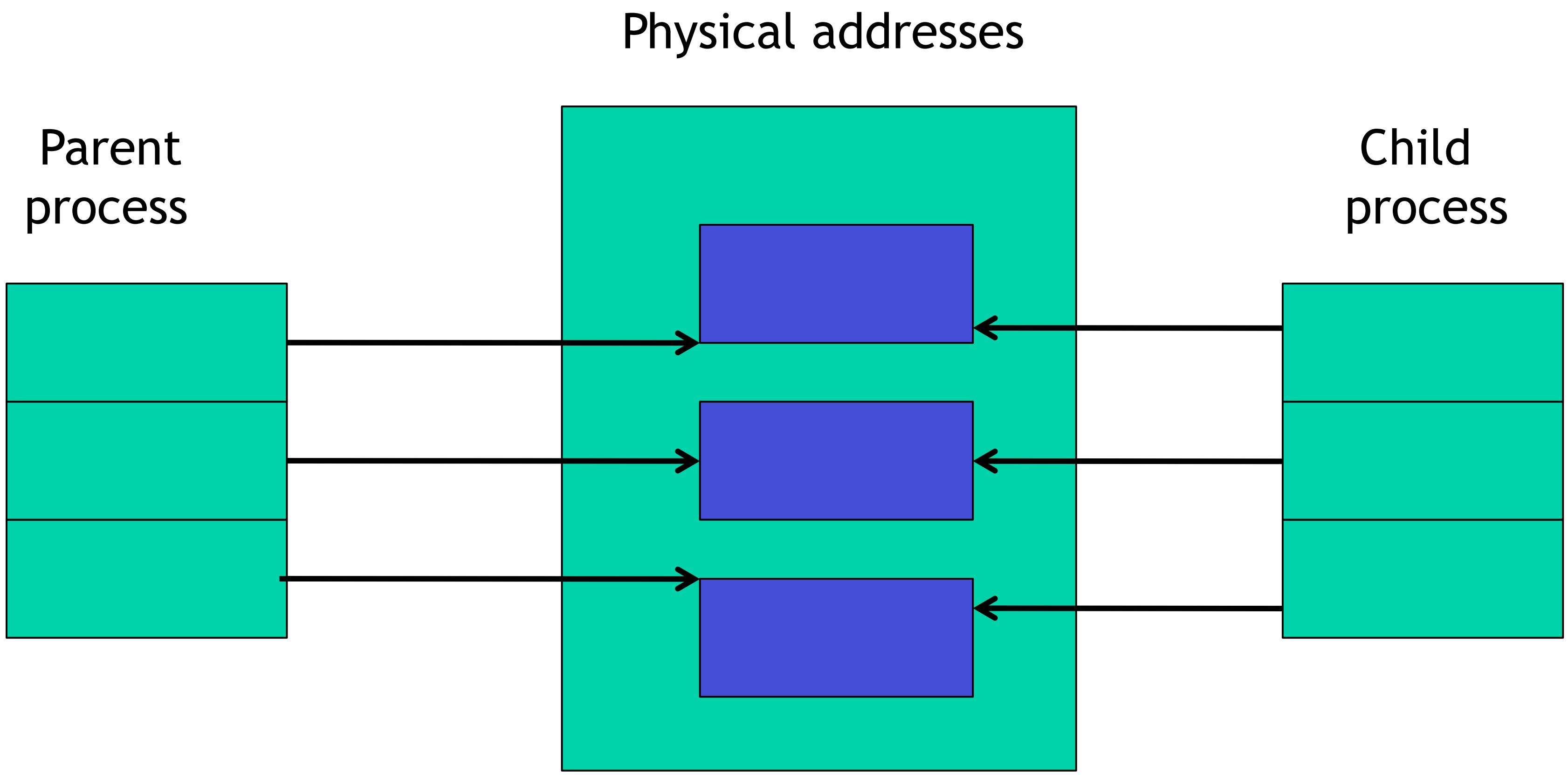
What is really happening here



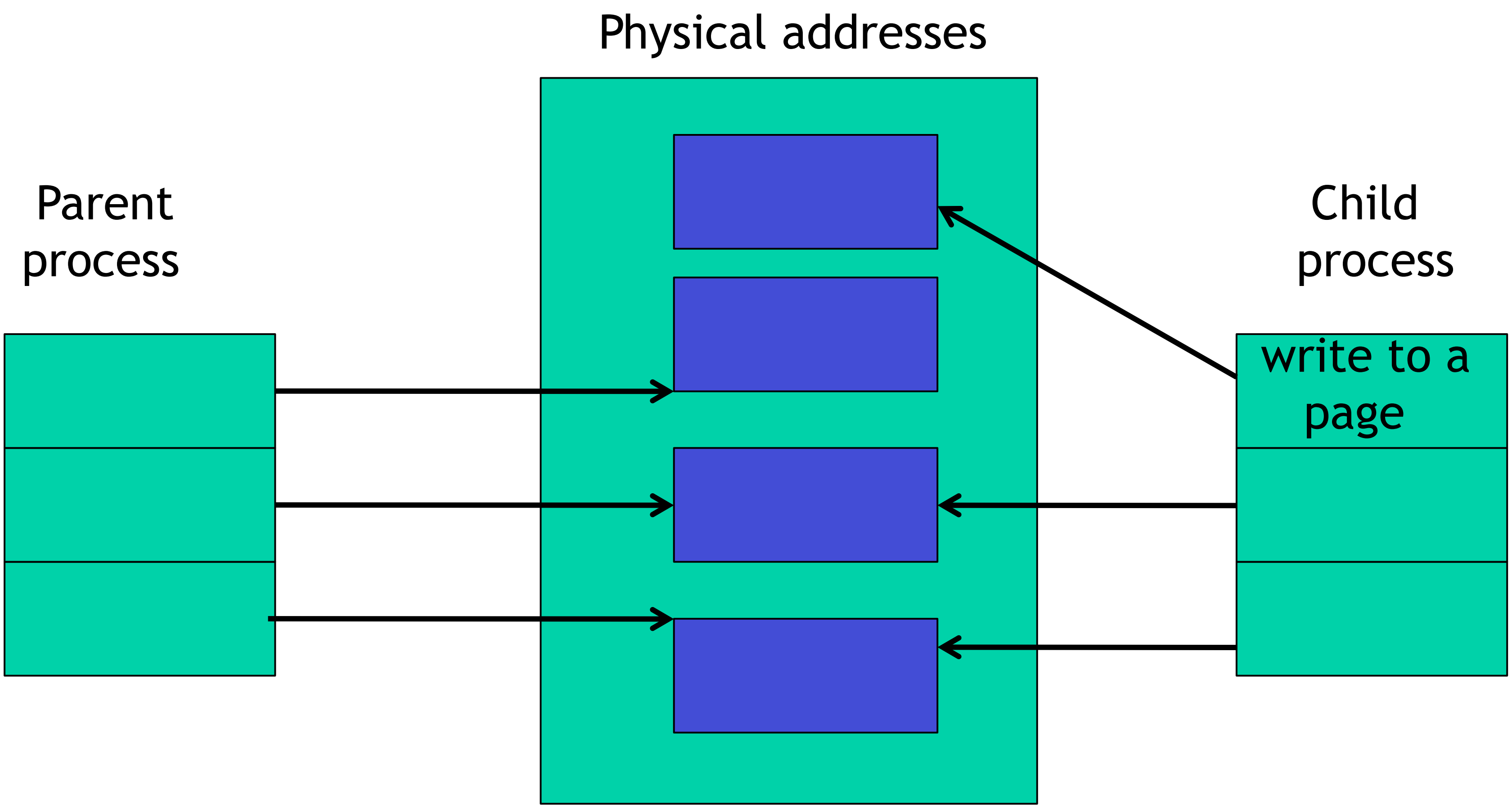
Fork() primer into virtual memory management



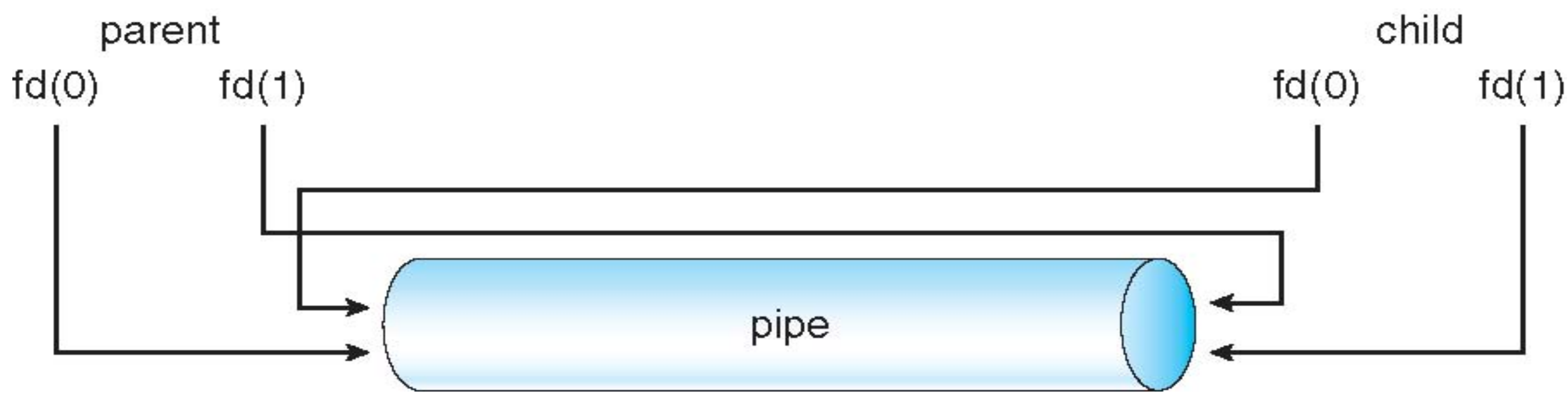
Fork() Copy-on-write policy



Fork() Copy-on-write policy



communication child/parent process (Unnamed pipes)

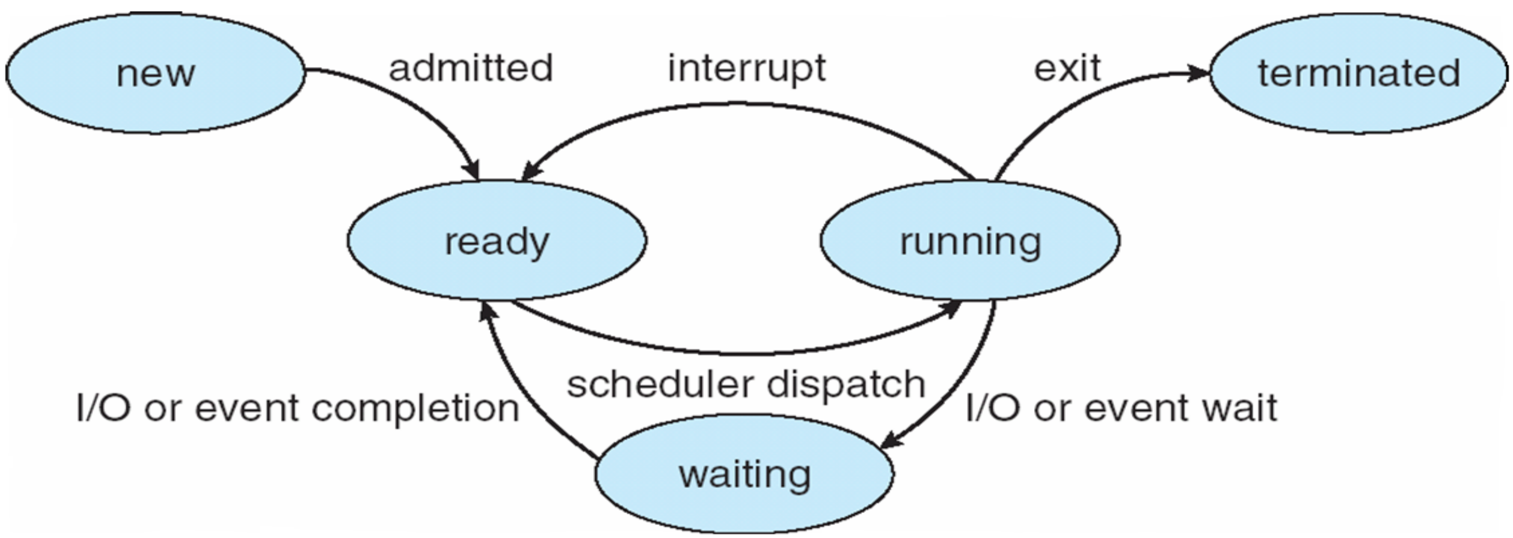



```
Pipe(fid); // where int fid[2] fid[0] is the read
    from the pipe and fid[1] is write to the pipe

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

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



Kernel data structure for processes (PCB)

Information associated with each process

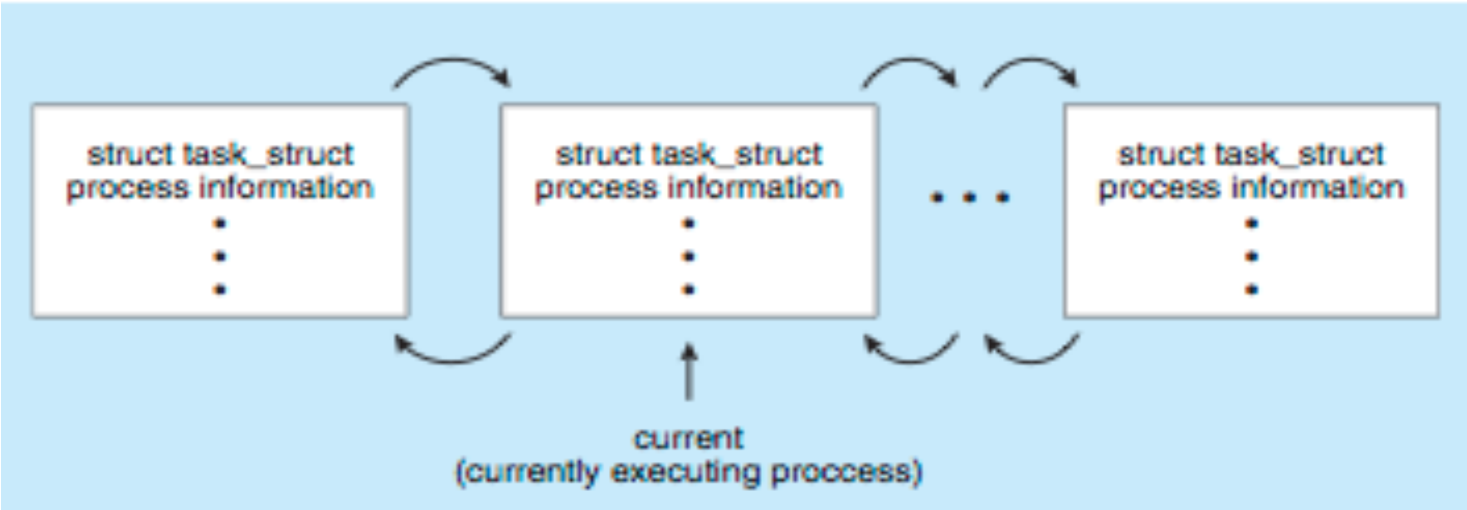
- Process state (running, waiting, ready, terminated)
- Program counter (instruction that is executing right now)
- CPU registers (eax, ebx, etc)
- CPU scheduling information (when would it scheduled)
- Memory-management information (pages that are allocated, non-allocated, what do they correspond to)
- Accounting information (several things like time created, time run, time waiting etc.)
- I/O status information (file descriptors in addition to stdin, stdout, stderr, other files, pipe information, FIFO, shared memory, mmaped file)

get a plethora of information in /proc/<pid>/

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 Context Switch

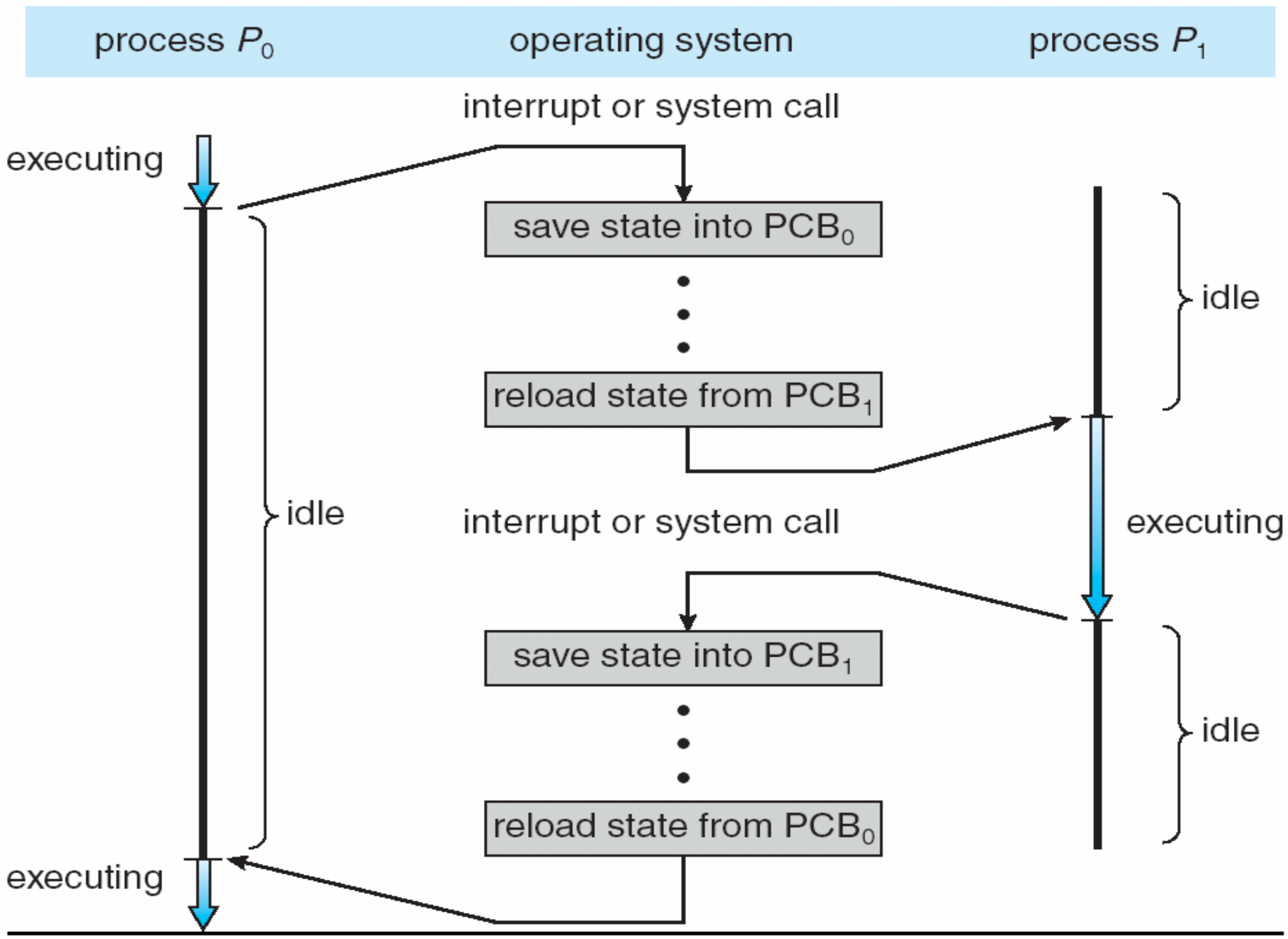
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.

- **Context** of a process represented in the PCB

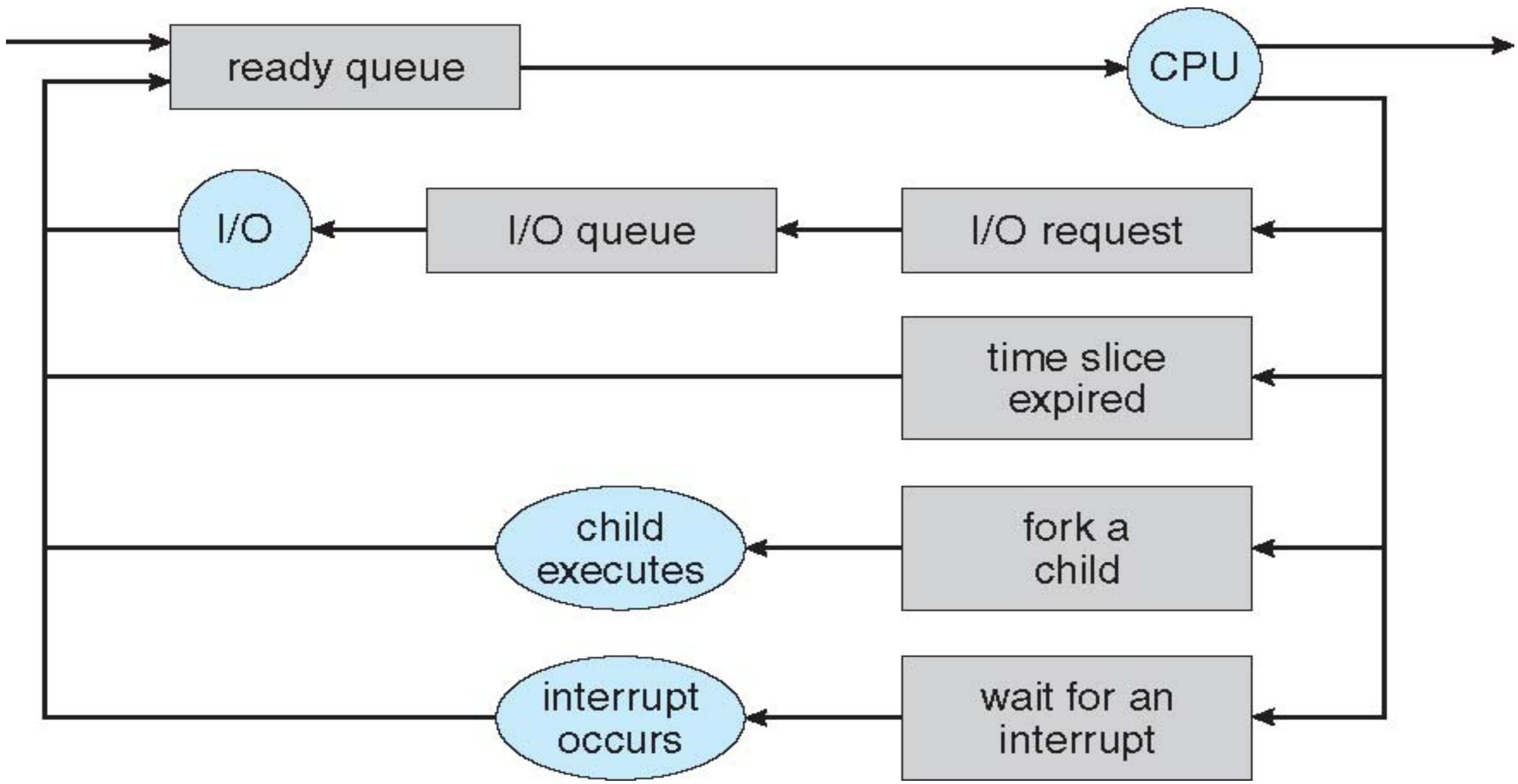
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch

- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

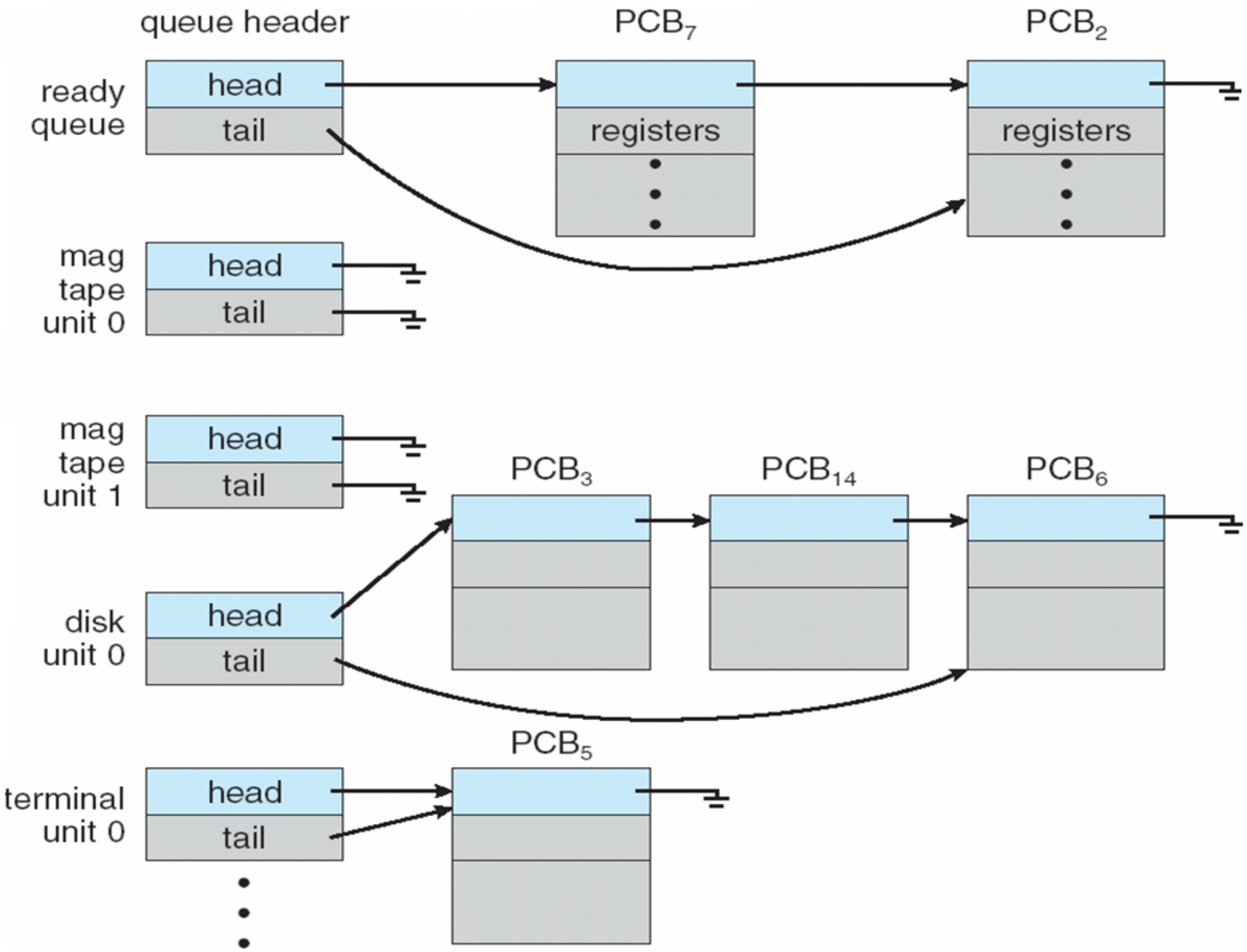
Process Context Switch



Process Scheduling



Process Queues



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

duplicate task_struct

Schedule child process

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)