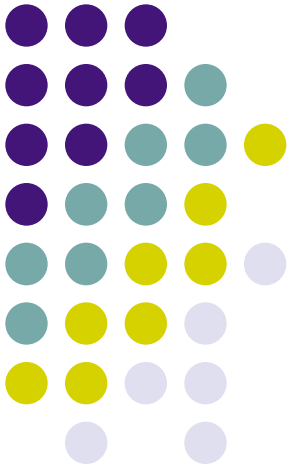
Processes

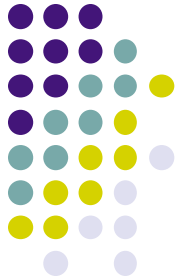
ECE469, Jan 17

Yiying Zhang

1. program process difference
17. process define
18. process state
19. pcb process control block
21. Process Creation/Termination
40. Context Switch



Review

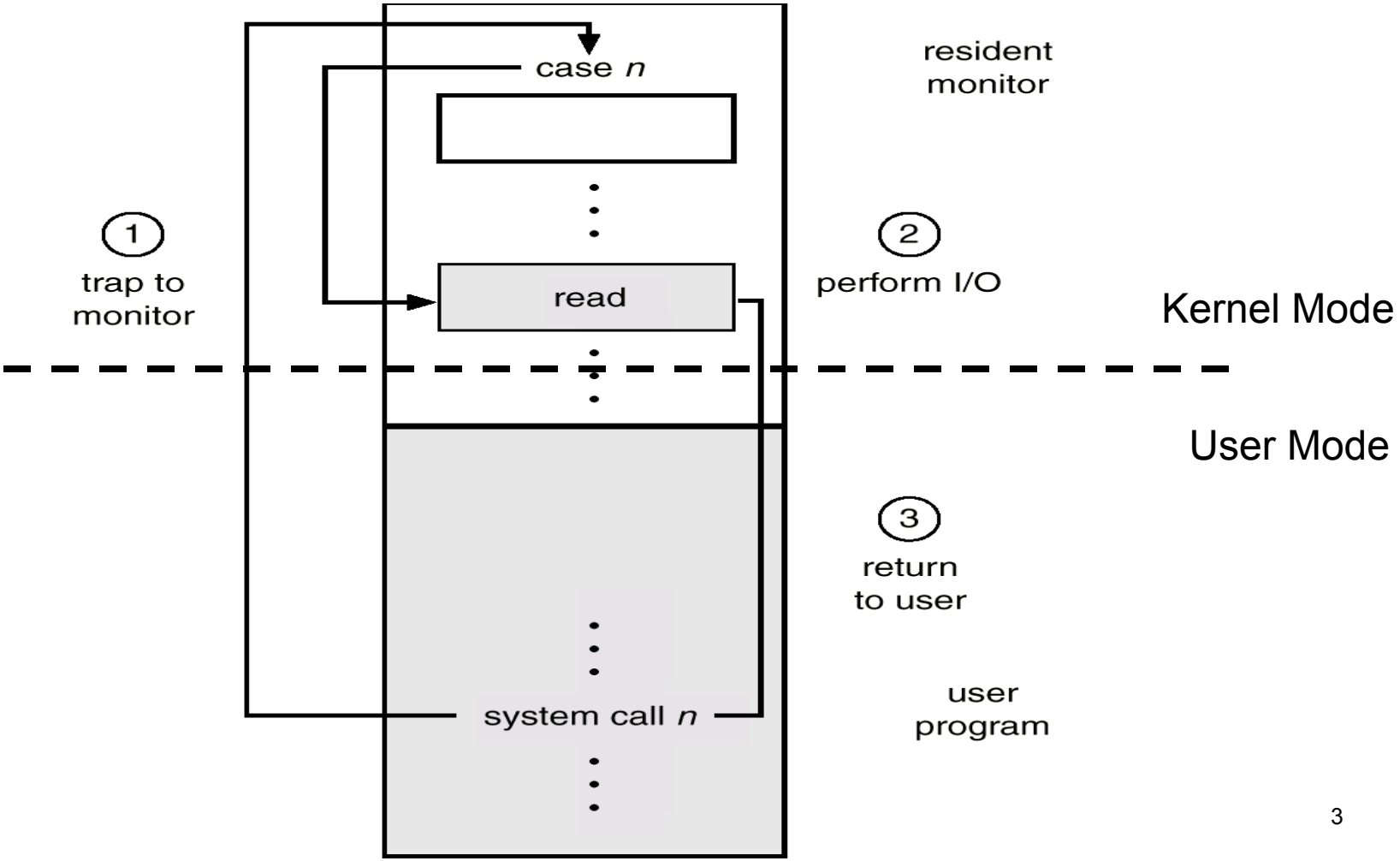


- OS is a resource manager
- OS presents an extended machine
- OS is a “giant interrupt handler”

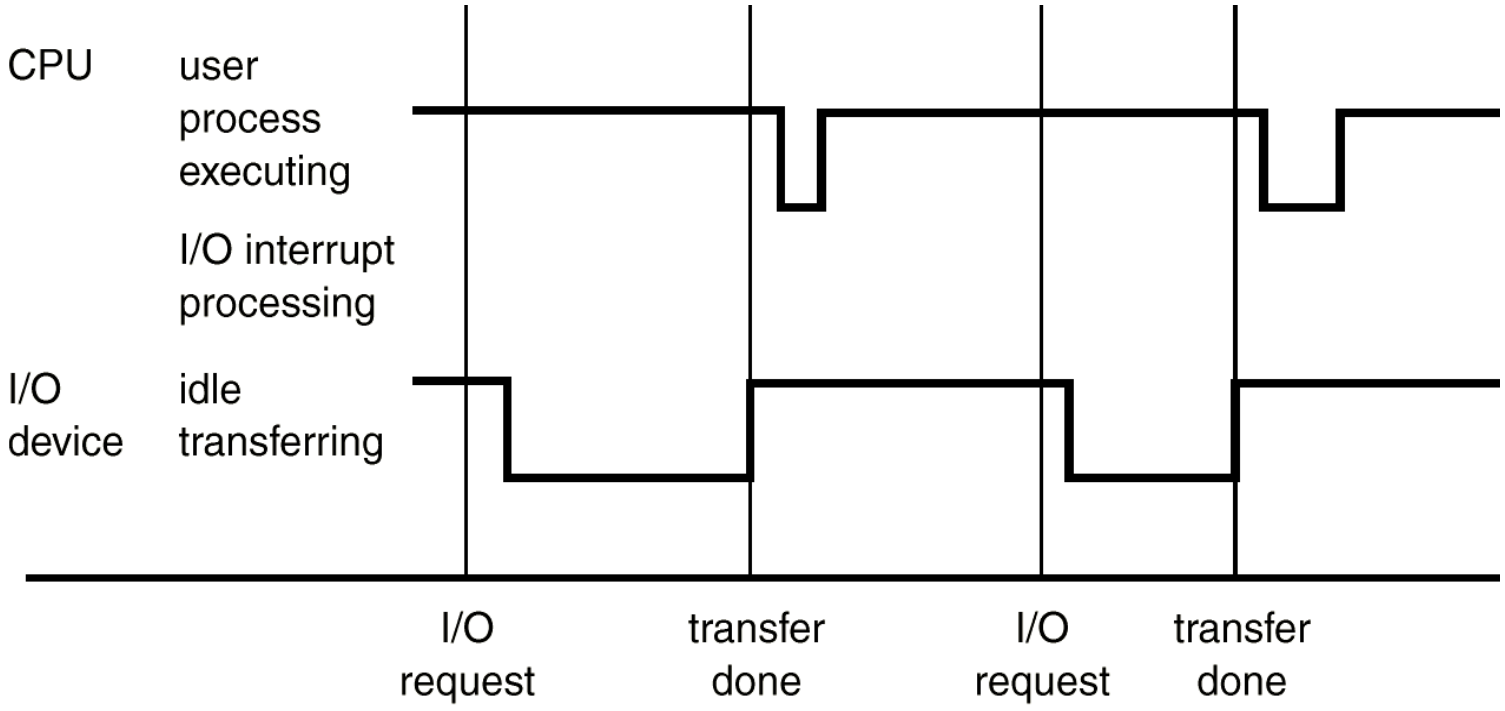
- System calls

- Interrupt

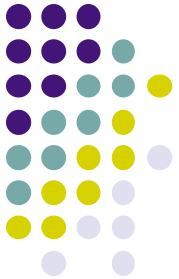
Use of a system call to perform I/O



Interrupt time line for a single process doing I/O



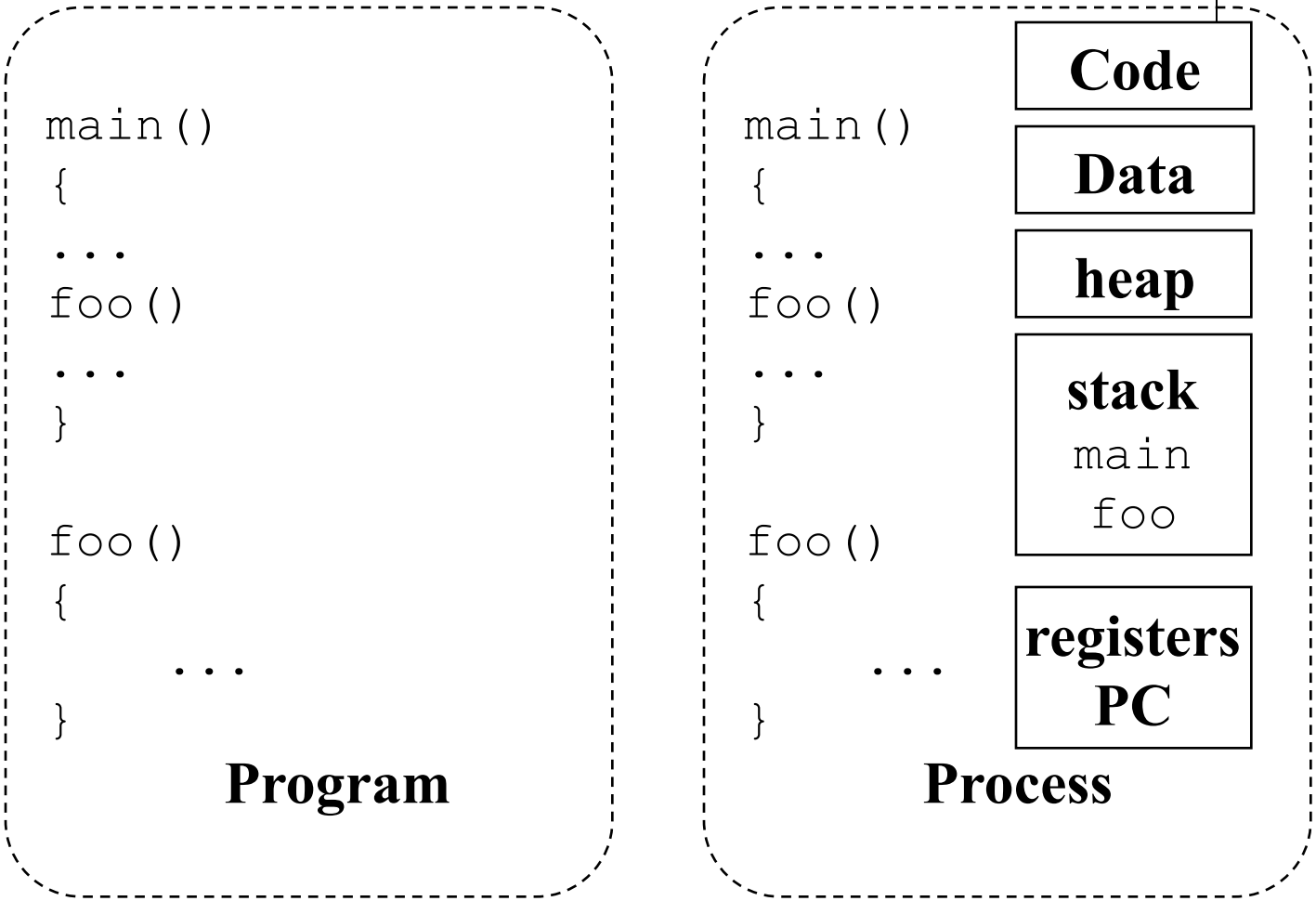
Users, Programs, Processes



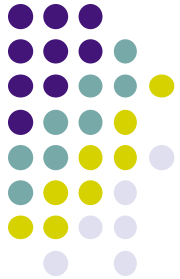
- Users have accounts on the system
- Users launch programs
  - Can many users launch the same program?
  - Can one user launch many instances of the same program?

→ A process is an “instance” of a program

Program vs. Process

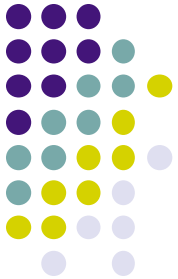


Program as A Process Collection, or a Program Collection



- Firefox (output of “ps ux | grep firefox”)
  - [~]$ ps ux | grep firefox
  - yiying 529 0.0 0.2 326756 9836 ? Sl 2012 1:43 /usr/lib64/xulrunner-2/plugin-container /usr/lib64/mozilla/plugins-wrapped/nswrapper_32_64.libflashplayer.so -greomni /usr/lib64/xulrunner-2/omni.ja -appomni /usr/lib64/firefox/omni.ja 25752 plugin
  - yiying 25752 3.9 6.6 1240212 250328 ? Sl 2012 8039:58 /usr/lib64/firefox/firefox
- gcc (via “gcc –pipe –v”)
  - /usr/libexec/cpp |
  - /usr/libexec/cc1 |
  - /usr/libexec/as, followed by
  - /usr/libexec/elf/ld

Users, Programs, Processes



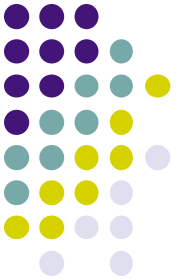
- Users have accounts on the system
- Users launch programs
  - Can many users launch same program?
  - Can one user launch many instances of the same program?

→ A process is an “instance” of a program

→ A “program” can be a set of programs

→ A “program” can be a set of processes

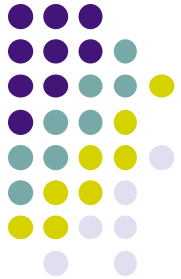
So What Is A Process? (1)



- It's one instance of a "program"

- Any relationship between two instances?

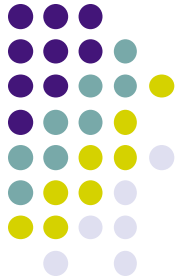
What Does This Program Do?



```
int myval;

int main(int argc, char *argv[])
{
    myval = atoi(argv[1]);
    printf("myval is %d, loc 0x%lx\n",
                    myval, (long) &myval);
}
```

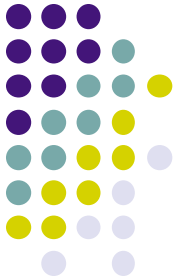
Instances of Programs



- The address of the static variable was always the same!
- The values were different!
- Implications:
  - Do instances think they're using the same address?
  - Are they seeing each other?

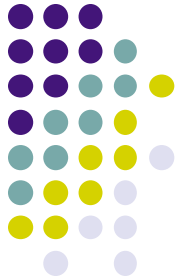
- Conclusion: addresses are not absolute!
- What are the benefits?
  - Compiler/linker/loader do not have to be concerned
  - Allows address space to be bigger than memory ?

So What Is A Process? (2)



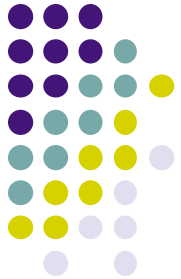
- It is one instance of a “program”
- It is separate from other instances

Sequential execution of each process



- Assuming single-threaded program

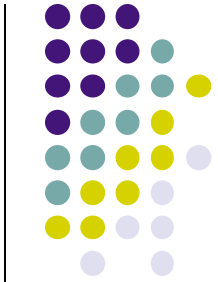
- No concurrency inside a process
- Everything happens sequentially
- Often with interleaved CPU/IO operations



Concurrent Processes

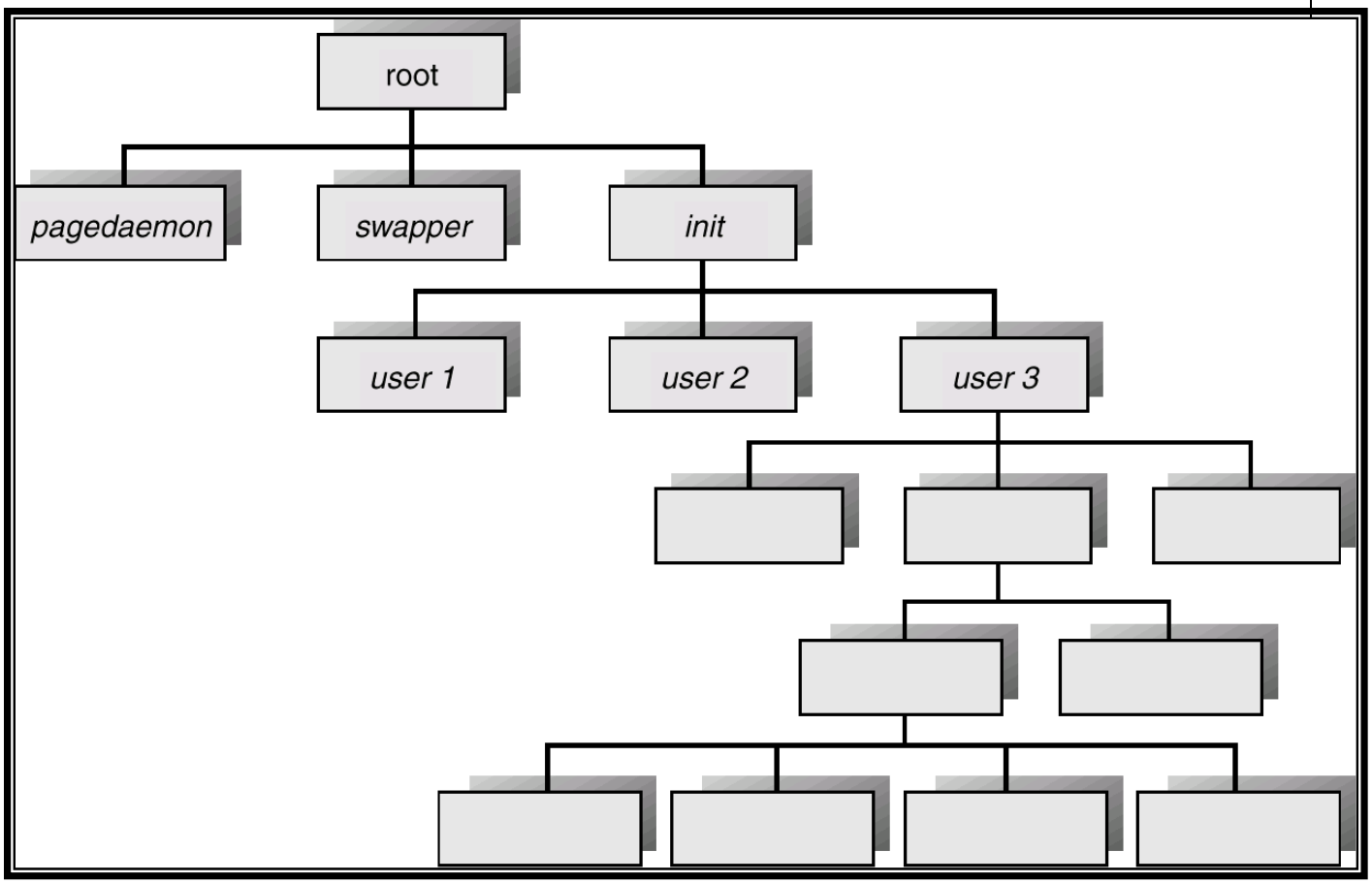
- Processes in a system can execute concurrently (multitasking)
- Motivations for allowing concurrent execution
  - Physical resource sharing (system utilization)
  - Computational speedup – with several CPUs
  - Modularity (firefox)
  - Convenience (desktop: firefox, xclock, emacs, xterm)

Process Creation

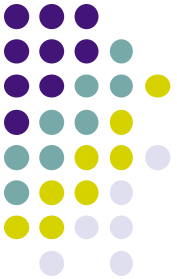


- Who created process cpp?
- Who created cc1?
- Who created gcc?
- Who created shell?

Processes Tree on a UNIX System



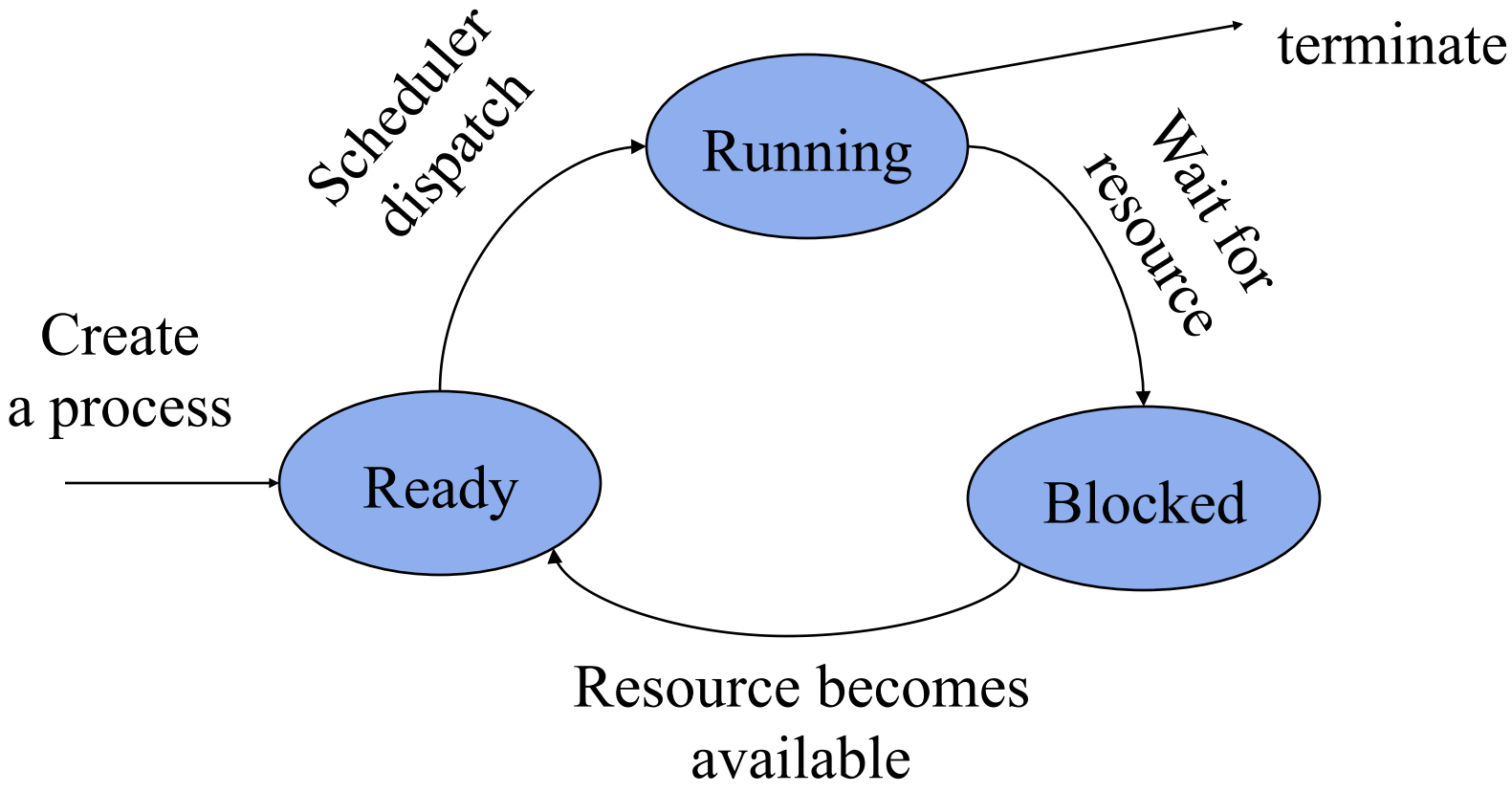
So What Is A Process? (3)



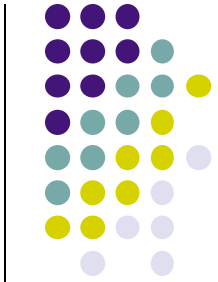
- It's one instance of a "program"
- It's separate from other instances

- It can start ("launch") other processes
- It can be launched by them

Life cycle of a process: Process State Transition

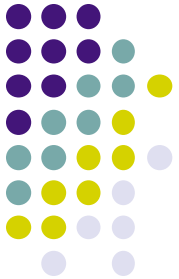


Kernel data structure: Process Control Block (Process Table)



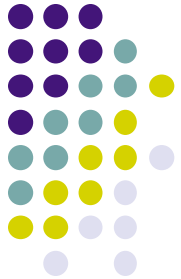
- Process management info
  - State (ready, running, blocked)
  - PC & Registers, parents, etc
  - CPU scheduling info (priorities, etc.)
- Memory management info
  - Segments, page table, stats, etc
- I/O and file management
  - Communication ports, directories, file descriptors, etc.

Process ID



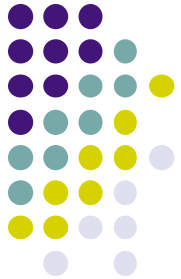
- Every process has an ID – process ID
- Does a program know its process ID?
- When a program is running, how does the process know its ID?

Process Creation/Termination



- Who should *actually* create/terminate processes? (bring program to life)
  - Who manages processes?
- But user process decides when and what
- So what should OS provide user process?
- What should the semantics of the interface be?

Process Creation – interface design options



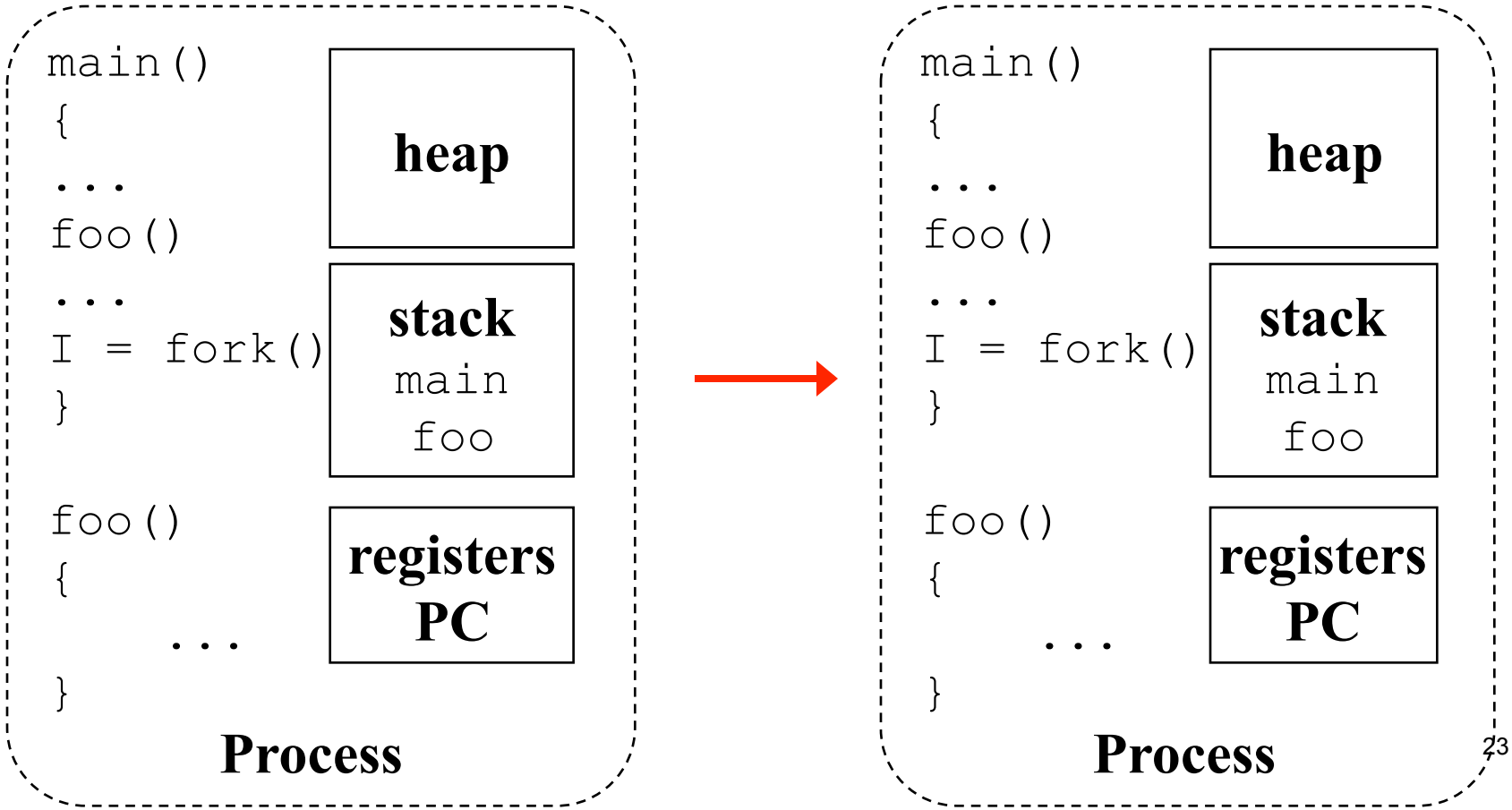
- The system call creates a child process

- Execution possibilities?
  - Parent and child execute concurrently
  - Parent waits till child finishes

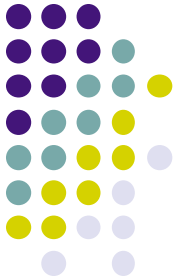
- Address space possibilities?
  - Child duplicate of parent (code and data)
  - Child has a new program loaded into it

Process Creation –- UNIX interfaces (1)

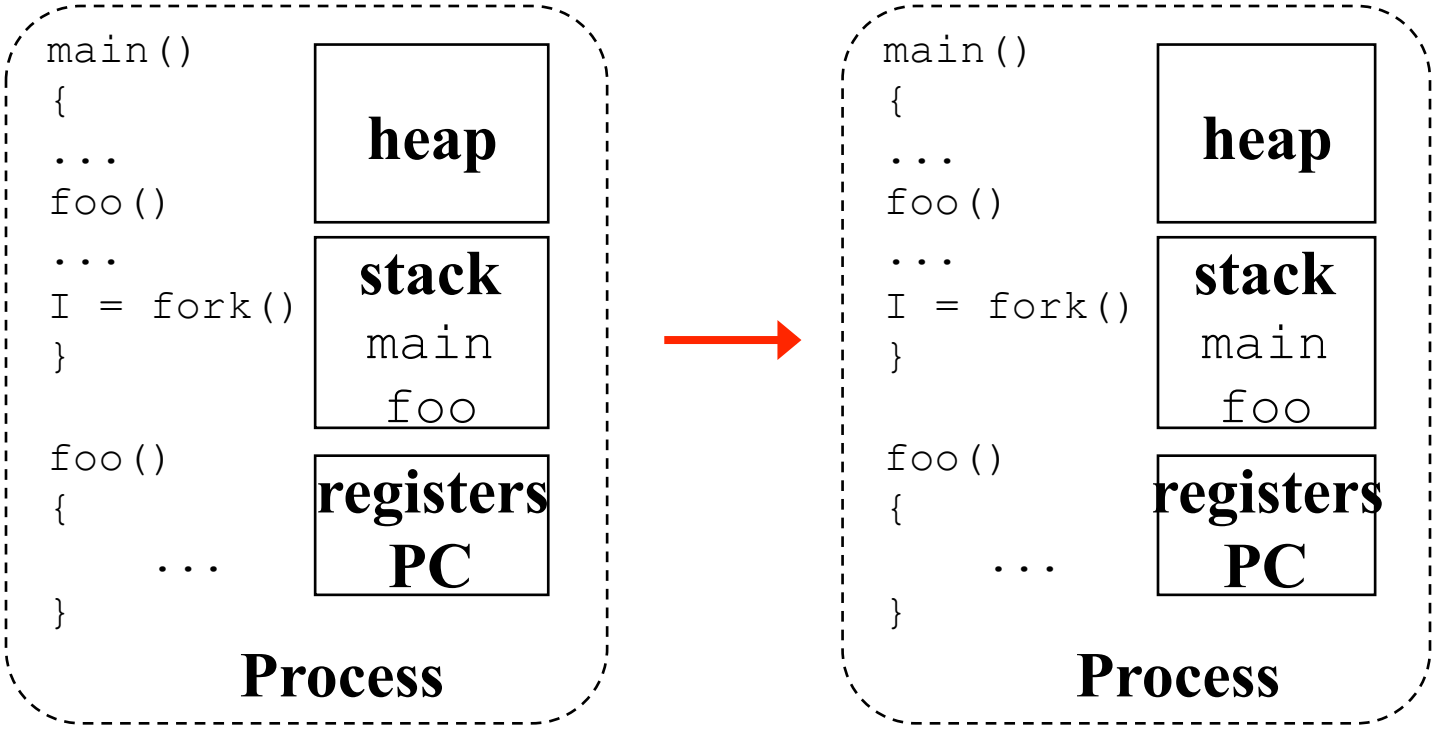
- *fork()* system call creates a duplicate of the original process



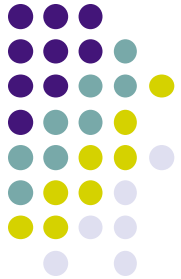
Process Creation –- UNIX interfaces (2)



- *fork()* system call creates a duplicate of the original process
  - What is the major benefit?
  - How to disambiguate who is who?



C program forking a new process

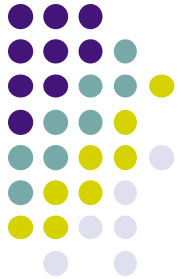


```
#include <stdio.h>

void main()
{
  int pid; int was = 3;
  pid = fork(); /* fork another process */

  if (pid < 0) { /* error occurred */          exit(-1); }
  else if (pid == 0) { /* child process */
     printf("child: was = %d\n", was);
  else { /* pid> 0; parent process ; pid is child process's */
     printf("parent: child process id = %d\n", pid);
     wait(NULL);    exit(0);
  }
}
```

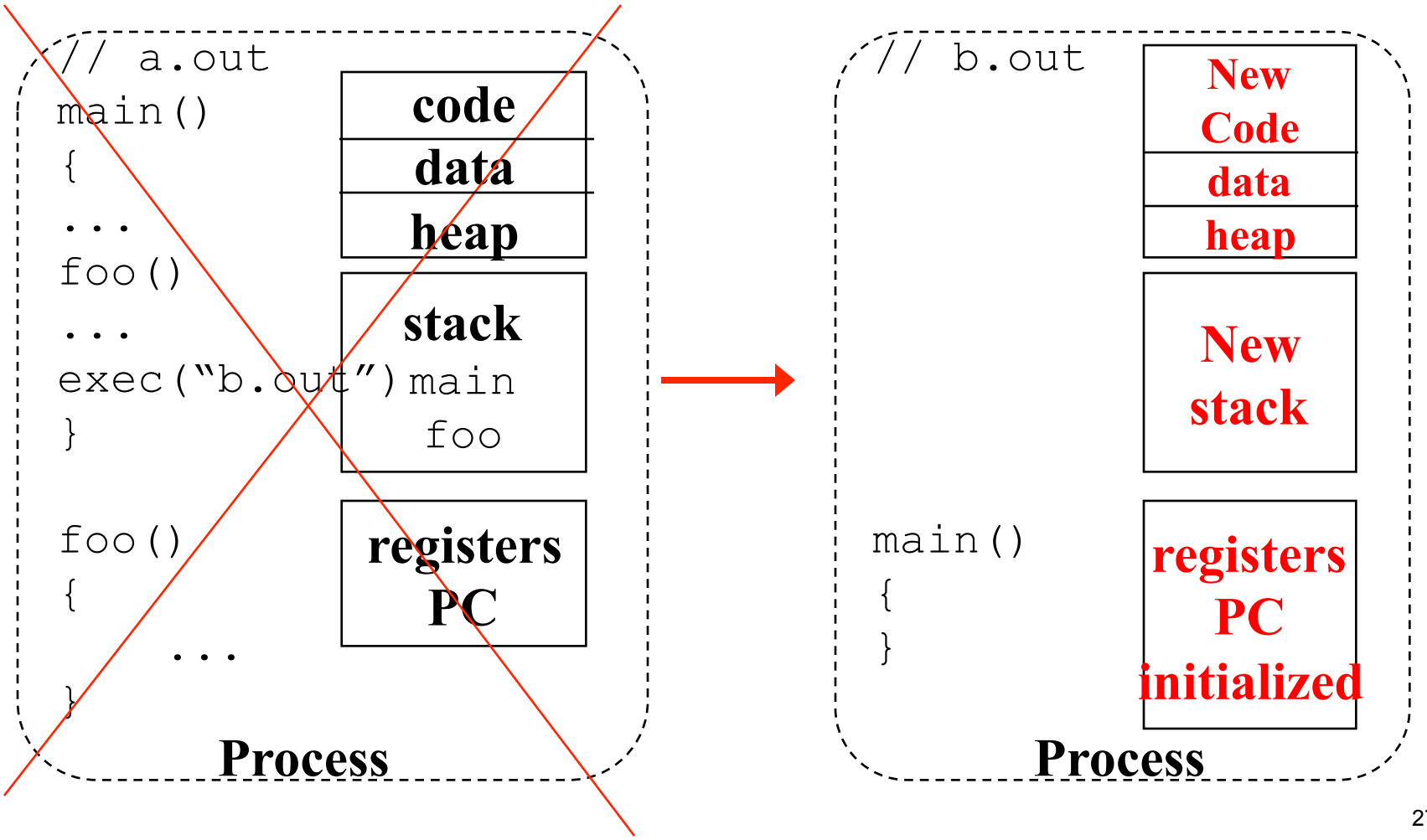
Process Creation –- UNIX interfaces (3)



- *fork()* system call creates a duplicate of the original process
  - What is the major benefit?
  - How to disambiguate who is who?

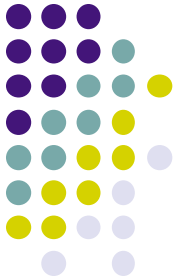
- *exec()* system call used after a *fork* to replace the process' code/address space with a new program
  - Important: BOTH code and data, i.e., the whole address space is replaced !

exec("b.out")



Afterwards, only one thing about the process was kept, which is?

Process Creation – UNIX interfaces (4)

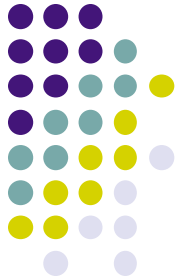


UNIX has 4 system calls:

- fork – create a copy of this process
  - Clone would have been a better name!
- exec – replace this process with this program

- wait – wait for child process to finish
- kill – (potentially) end a running process

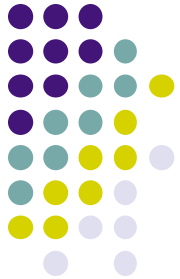
C program forking a new process



```
#include <stdio.h>
void main()
{
  int pid;        int was = 3;
  pid = fork(); /* fork another process */

  if (pid == 0) { /* child process */
     sleep(2);  printf("child: was = %d\n", was);
     execlp("/bin/ls", "ls", NULL);}
  else { /* pid > 0; parent process */
     was = 4;
     printf("parent: child process id = %d; was=%d\n", pid, was);
     wait(NULL);     exit(0);
  }
}
```

C program forking a new process

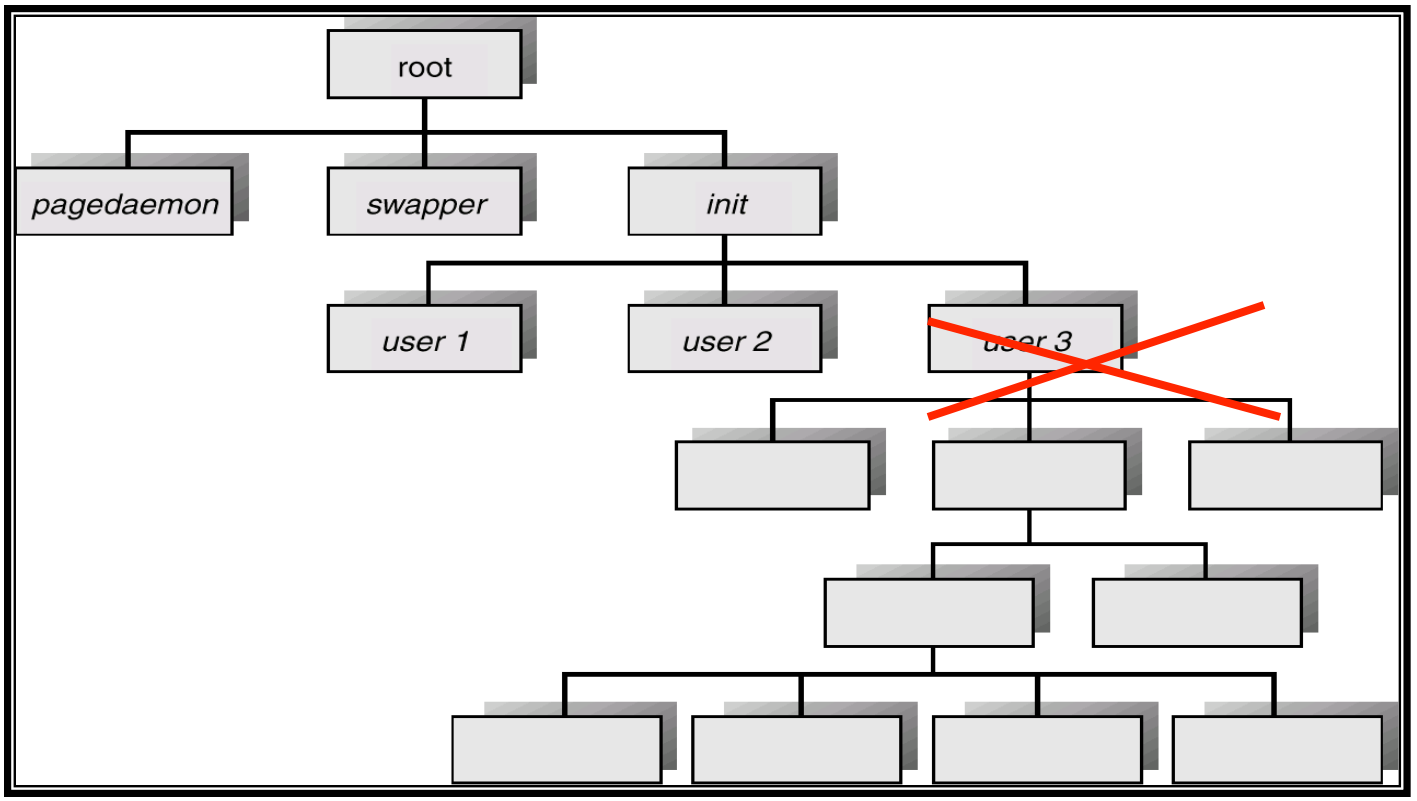


```
#include <stdio.h>

void main()
{
  int pid;       int was = 3;
  pid = fork(); /* fork another process */

  if (pid == 0) { /* child process */
     sleep(2); was=9;  printf("child: was = %d\n", was);
     execlp("/bin/ls", "ls", NULL);      was = 10;
     printf("It's me, your child was = %d\n", was);}
  else { /* pid > 0; parent process */
     was = 4;
     printf("parent: child process id %d was=%d\n", pid, was);
     wait(NULL);     exit(0);
  }
}
```

Processes Tree on a UNIX System



What happens when a parent process disappears?

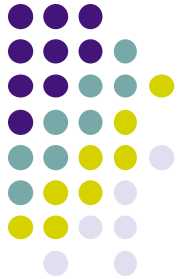
all child processes are killed by the OS, or

all child processes reset init as their parent

Break

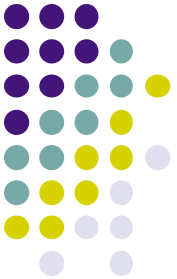
- 100 passengers are boarding an airplane with 100 seats. Everyone has a ticket with his seat number. These 100 passengers boards the airplane in order. However, the first passenger lost his ticket so he just take a random seat. For any subsequent passenger, he either sits on his own seat or, if the seat is taken, he takes a random empty seat. What's the probability that the last passenger would sit on his own seat? There is a very simple explanation for the result.

Multiprogramming needs CPU scheduling



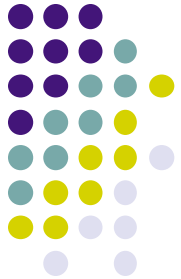
- Without any hardware support, what can the OS do to a running process?

System calls may trigger Scheduler



- Block – wait on some event/resource
  - Network packet arrival (e.g., recv())
  - Keyboard, mouse input (e.g., getchar())
  - Disk activity completion (e.g., read())
- Yield – give up running for now

Non-Preemptive Scheduler



- A non-preemptive scheduler: a scheduler that is only invoked by explicit block/yield calls, or terminations
  - Only method when there is no timer!

- The simplest form

  Scheduler:

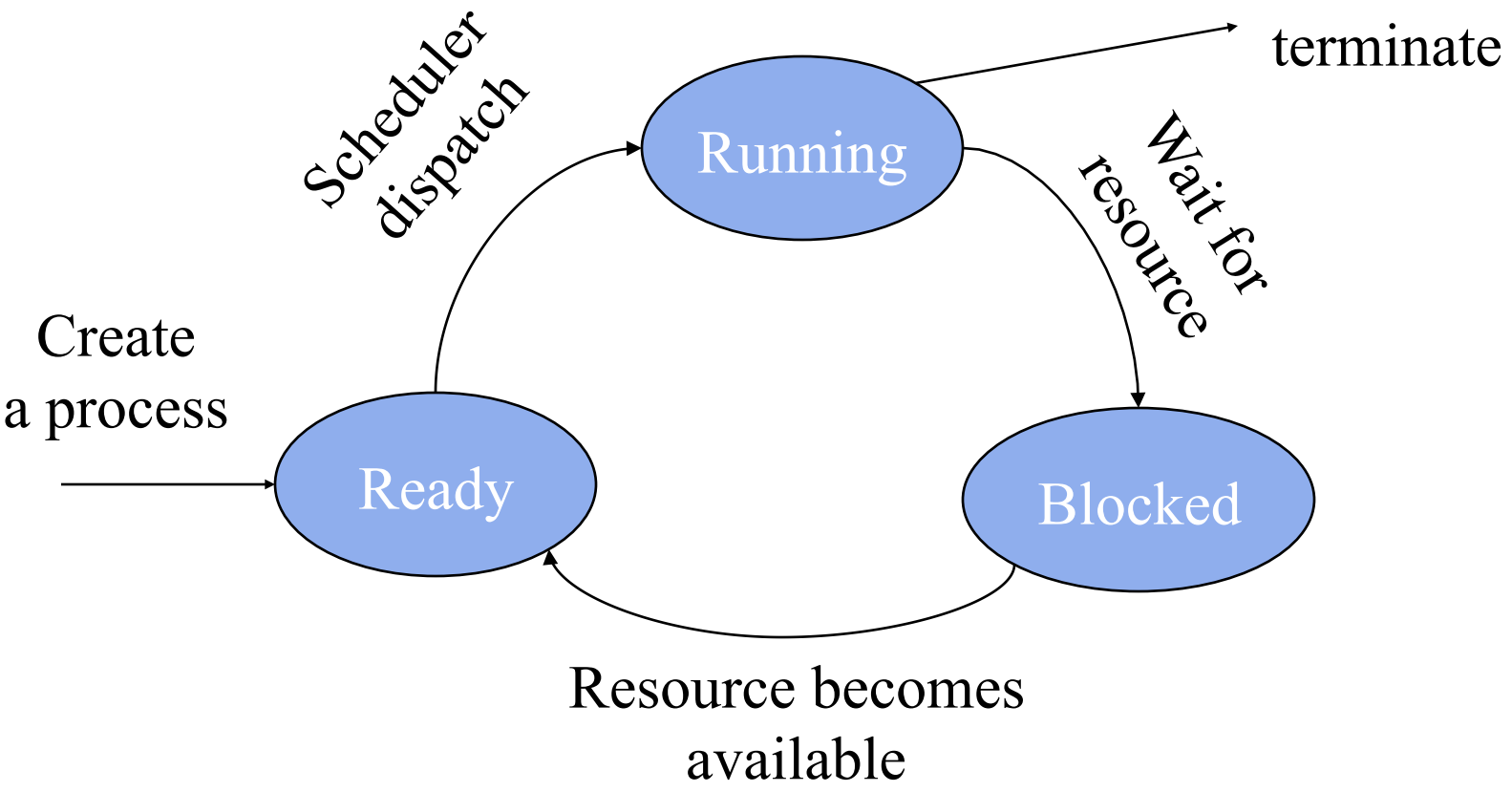
  save current process state (into PCB)

  choose next process to run

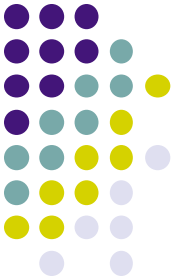
  dispatch (load PCB and run)

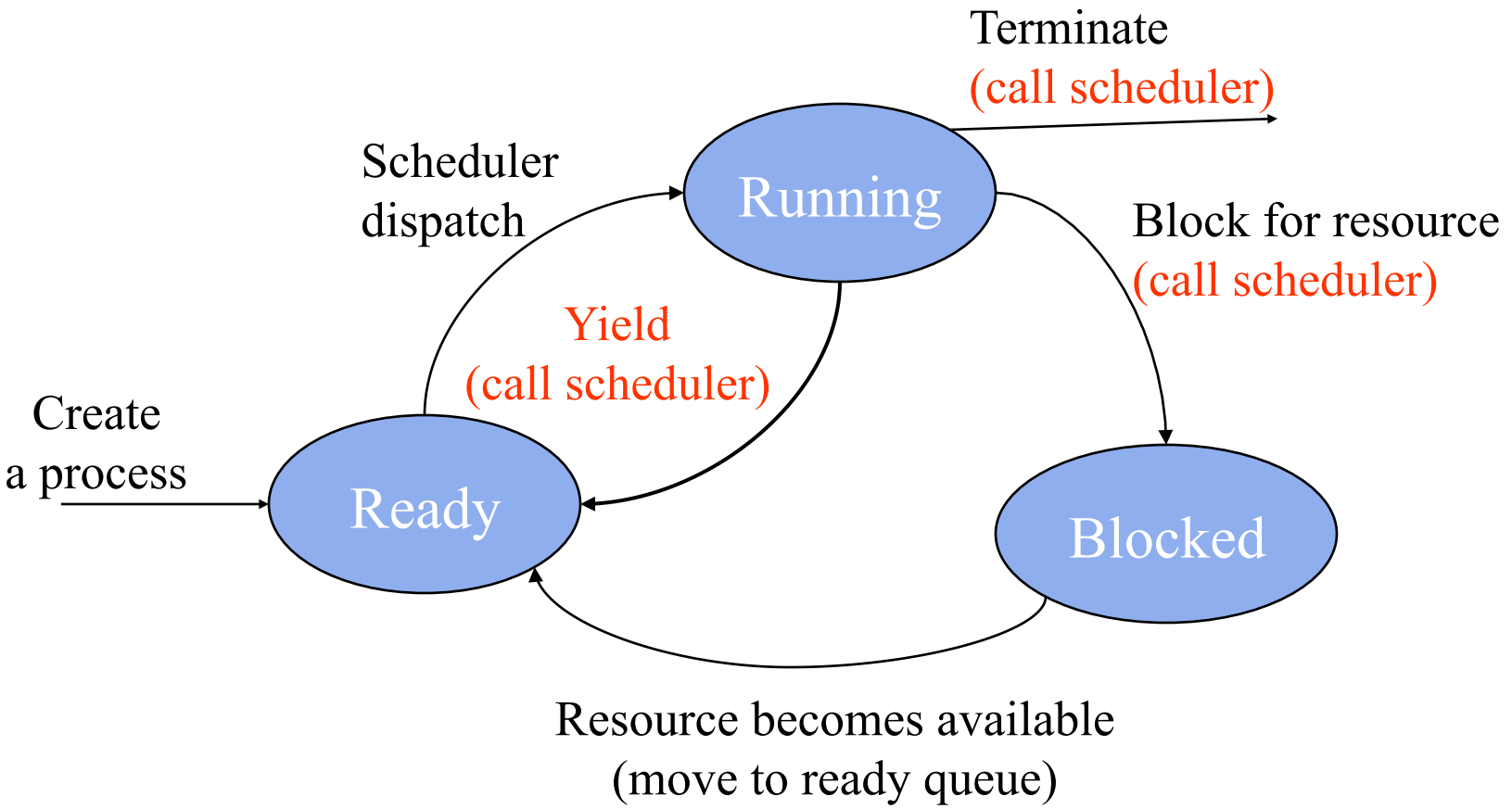
- Used in Windows 3.1, Mac OS

Our Friend -- the Transition Diagram



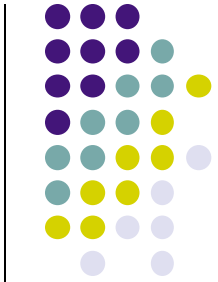
Process State Transition of Non-Preemptive Scheduling





How does a process terminate itself?

Context Switch

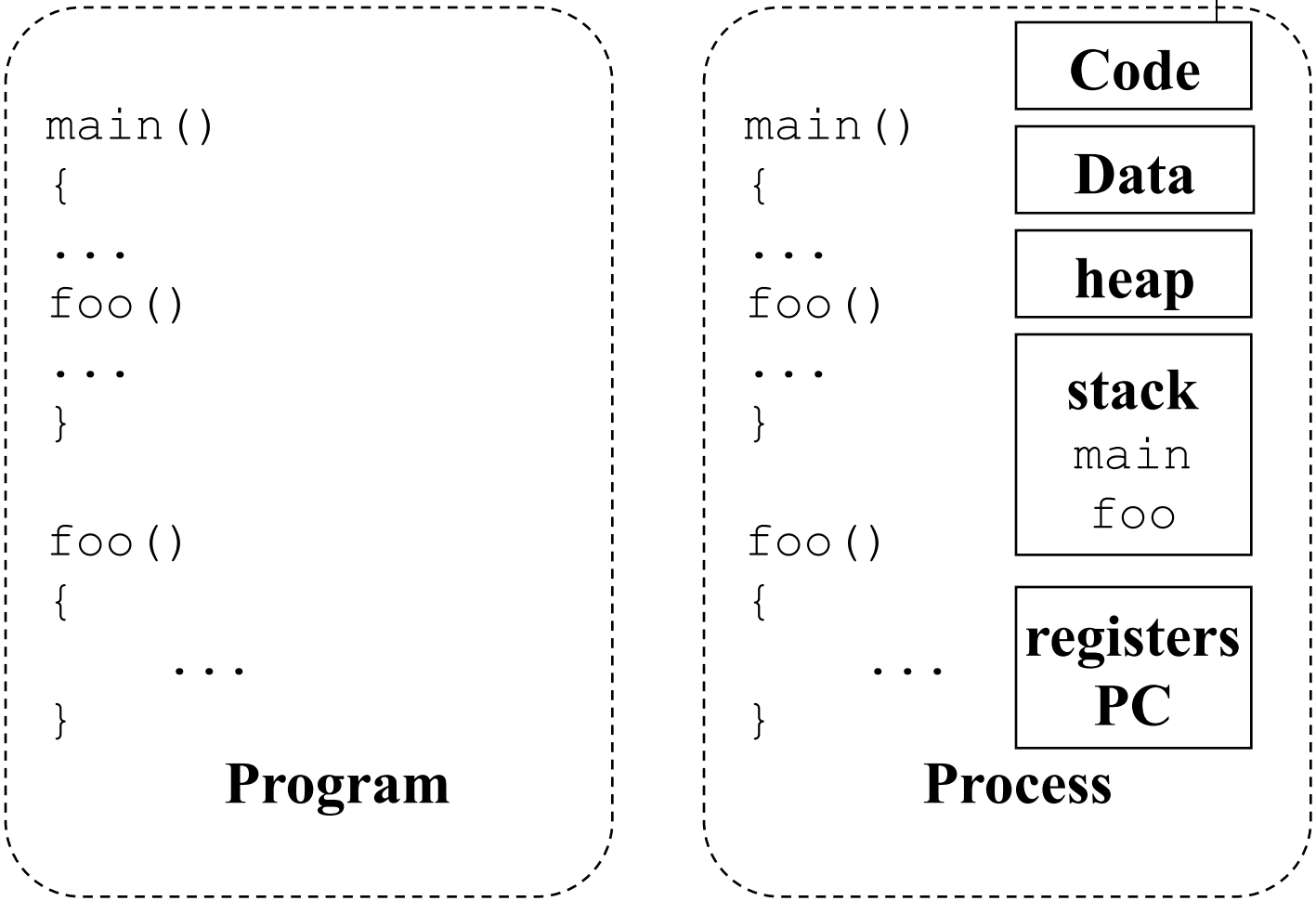


- Definition:

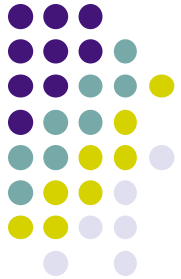
  switching the CPU to run another process, which involves (1) saving the state of the old process and (2) loading the state of the new process

- What state?

[lec3] Program vs. Process



Context Switch

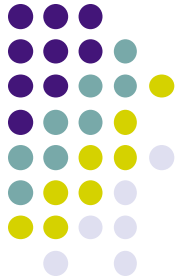


- Definition:

  switching the CPU to run another process, which involves (1) saving the state of the old process and (2) loading the state of the new process

- What state?
  - What about L1/L2 cache content?

Context Switch



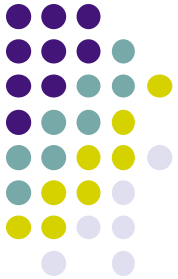
- Definition:

  switching the CPU to another process, which involves saving the state of the old process and load the state of the new process

- What state?

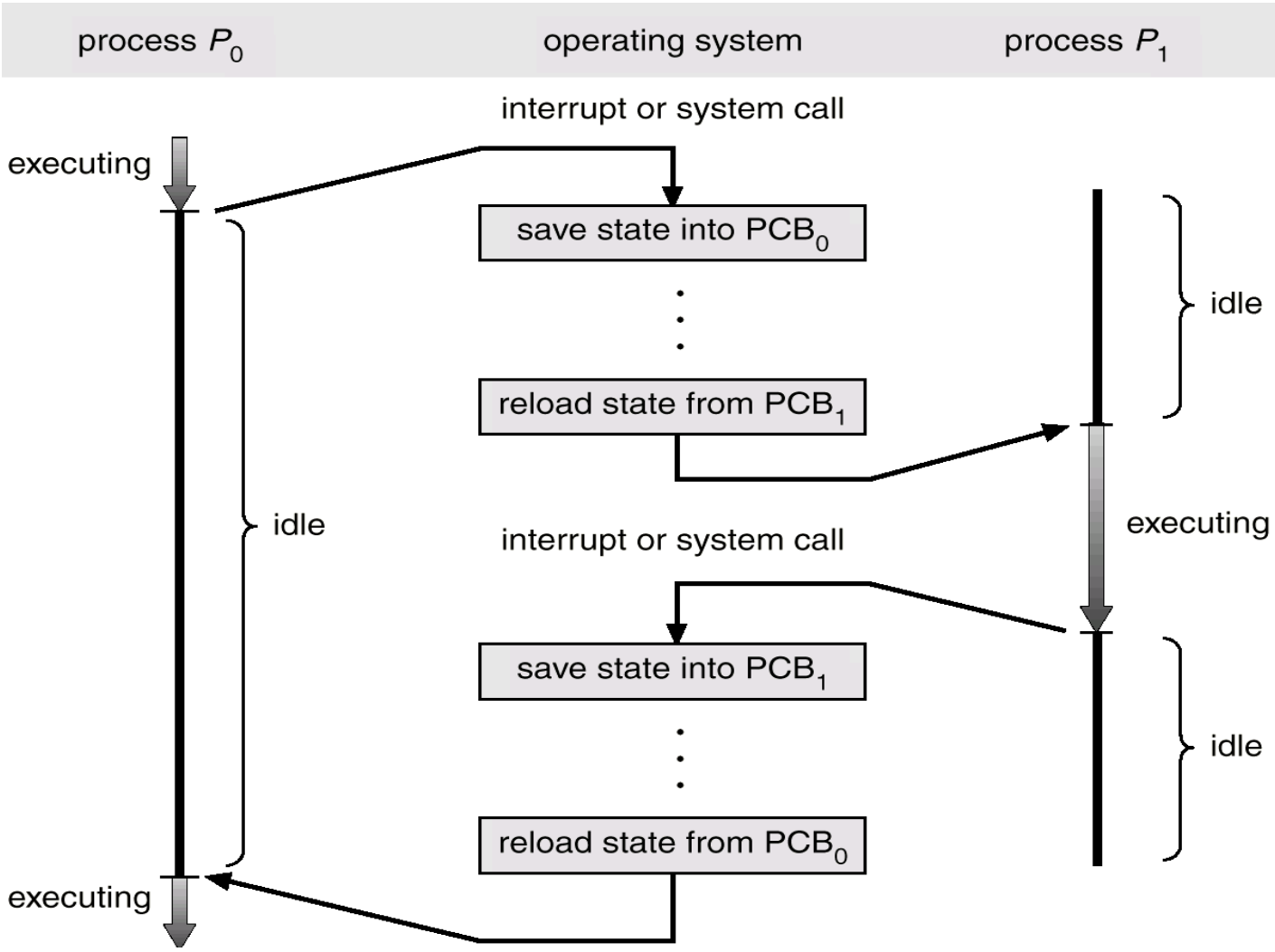
- Where to store them?

[lec3] Process Control Block (Process Table)

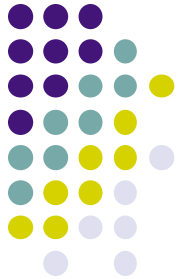


- Process management info
  - State (ready, running, blocked)
  - PC & Registers, parents, etc
  - CPU scheduling info (priorities, etc.)
- Memory management info
  - Segments, page table, stats, etc
- I/O and file management
  - Communication ports, directories, file descriptors, etc.

Context Switch



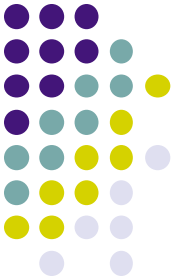
Physical Memory & Multiprogramming

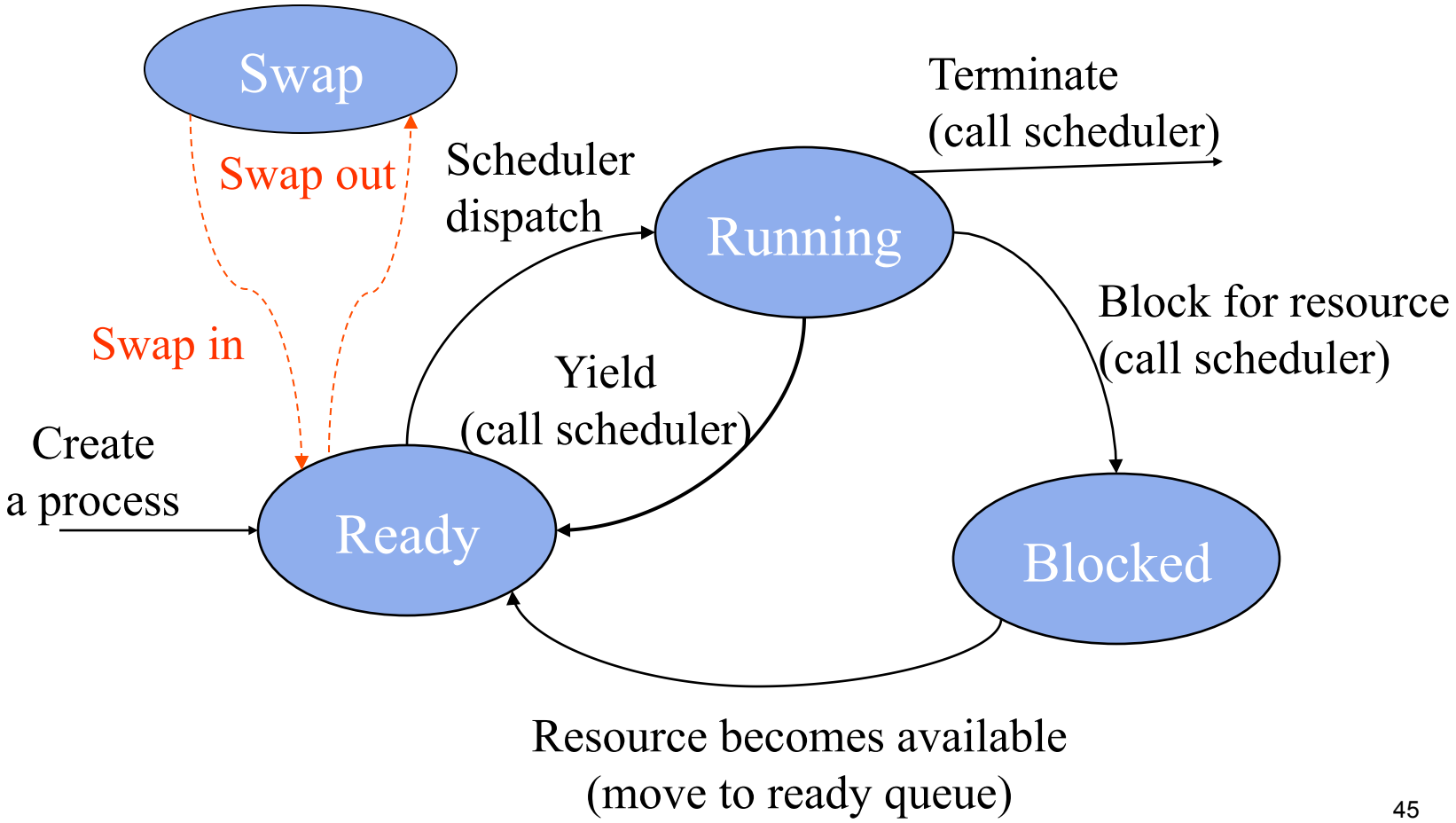


- Want to run many programs
- Programs need memory to run
- Memory is a scarce resource

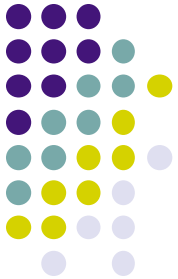
- What happens when
  M(a) + M(b) + M(c) > physical mem?

Add Job Swapping to State Transition Diagram





Summary



- What is a process?
- PCB
  - Process management info
  - Memory management info
  - I/O, file management info
- Process state transition
- Concurrent processes
- Process creation and termination
- Multiprogramming and context switch

Reading Assignment

- Dinosaur Chapter 3 by Thursday

- Comet Chapters 4,5,6