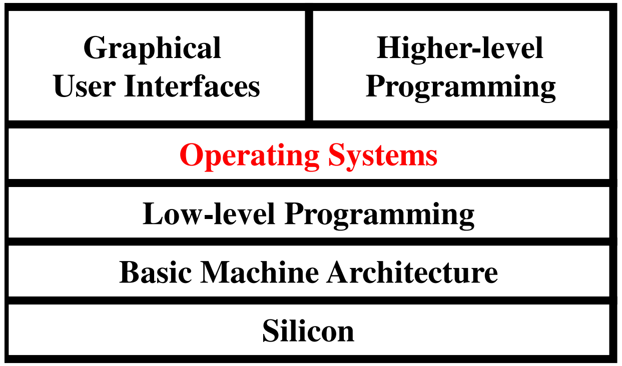


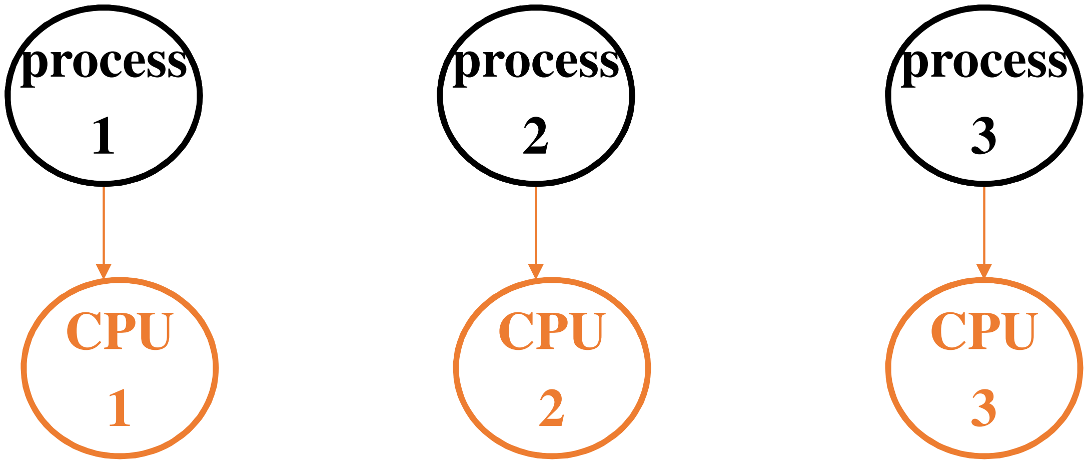
CSCU9V4
Systems

Systems lecture 16
Operating System 3

Processors, Threads and Scheduling

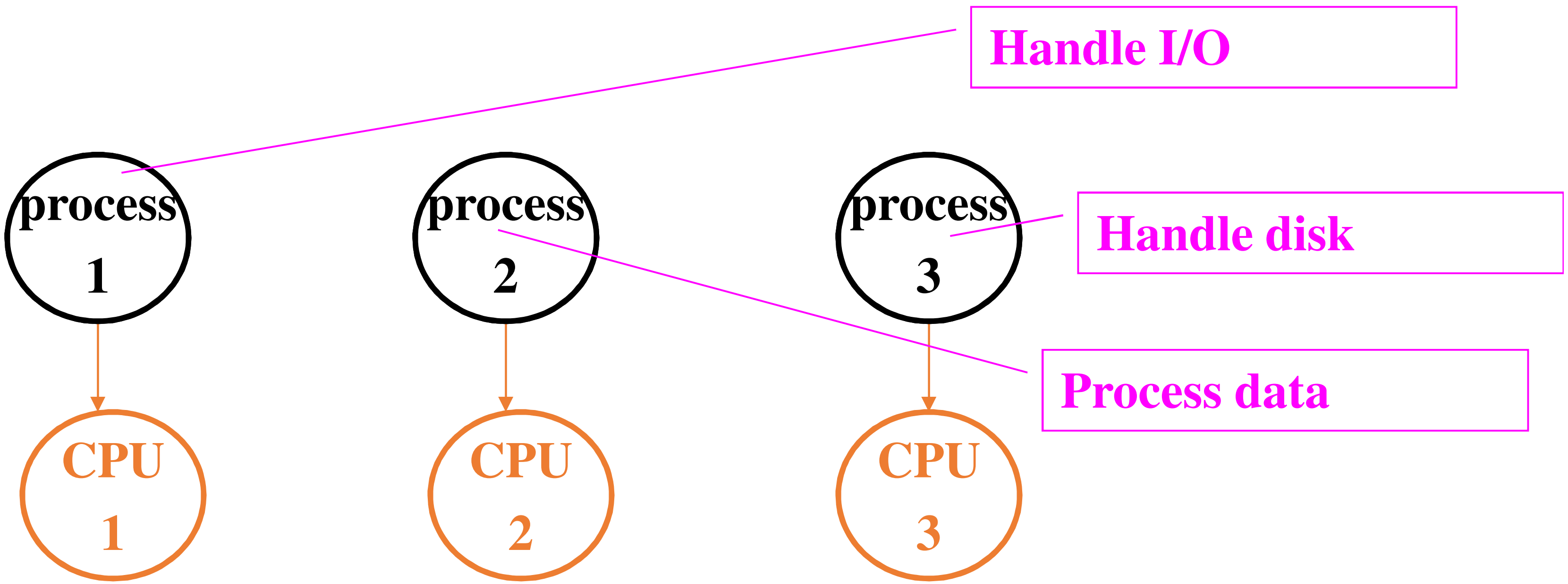
Executing programs: *user's view*

- it is common to have a number of programs executing in parallel.
  - convenience of user
  - performance
- each program is running under a separate process



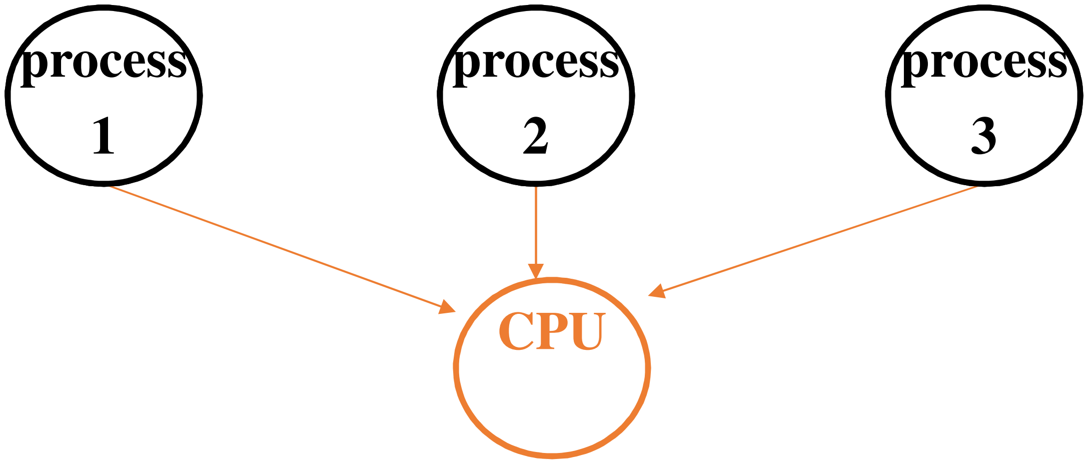
Executing programs: *programmer's view (1)*

- it is common for a programmer to have a number of programs in parallel.
  - convenience of coding and design
  - performance
- each program is in effect running under a separate process

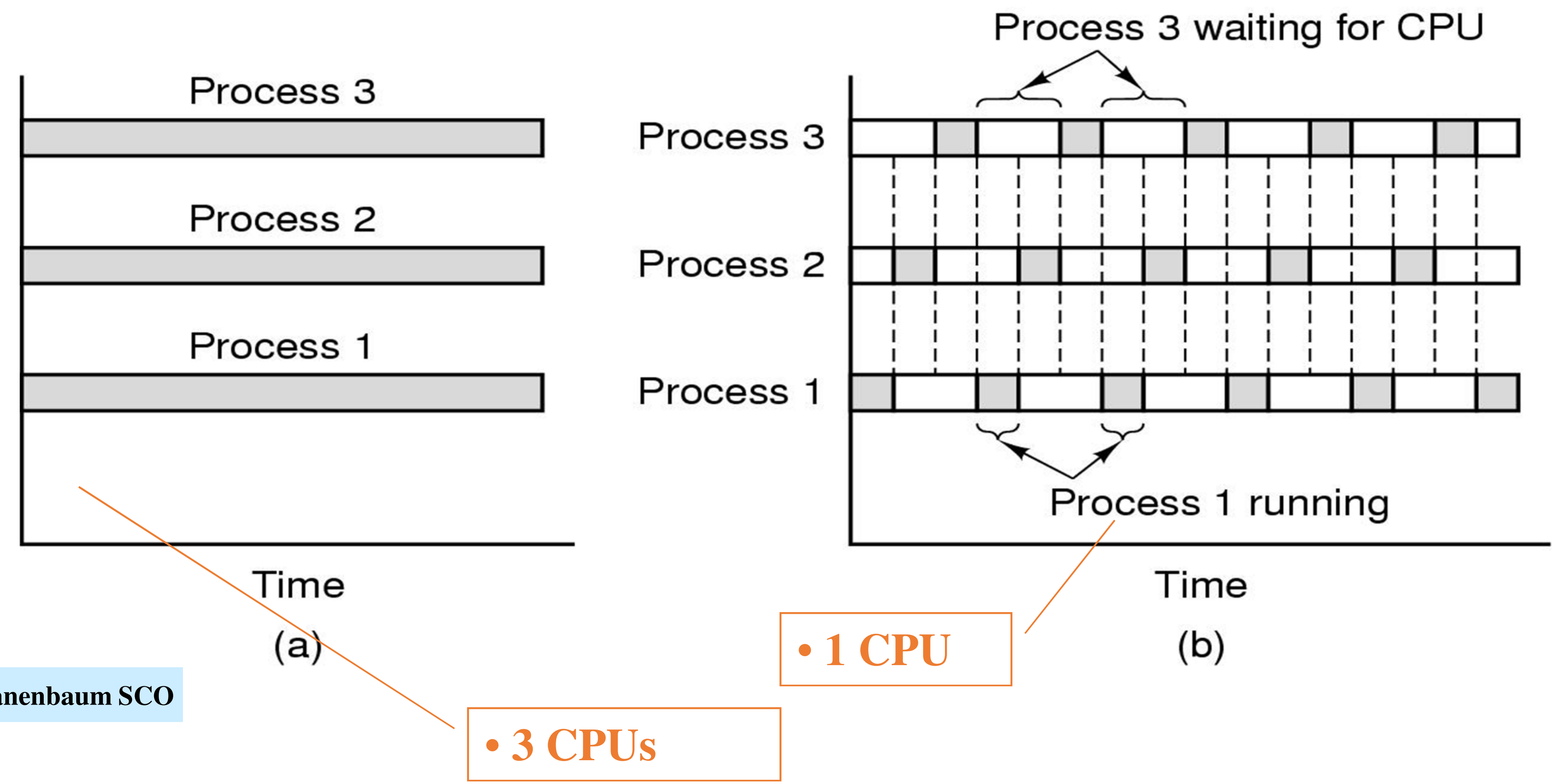


Executing programs: *programmer's view (2)*

- A single task may be implemented as a number of independent processes
- at minimal performance loss can just use 1 CPU
  - And no financial cost
- … indeed may be more inefficient than 1 program



Executing programs: *programmer's view (3)*

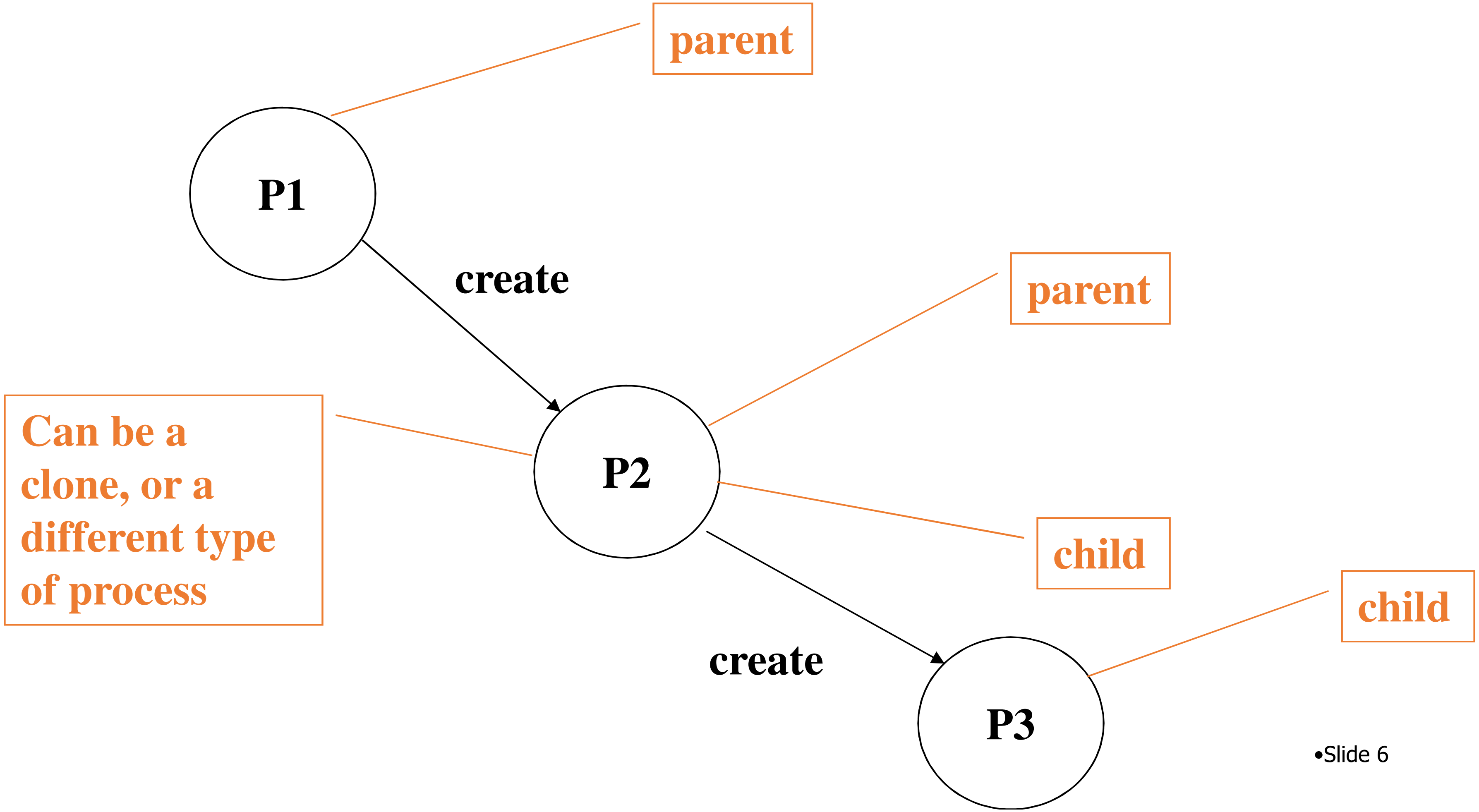


Tanenbaum SCO

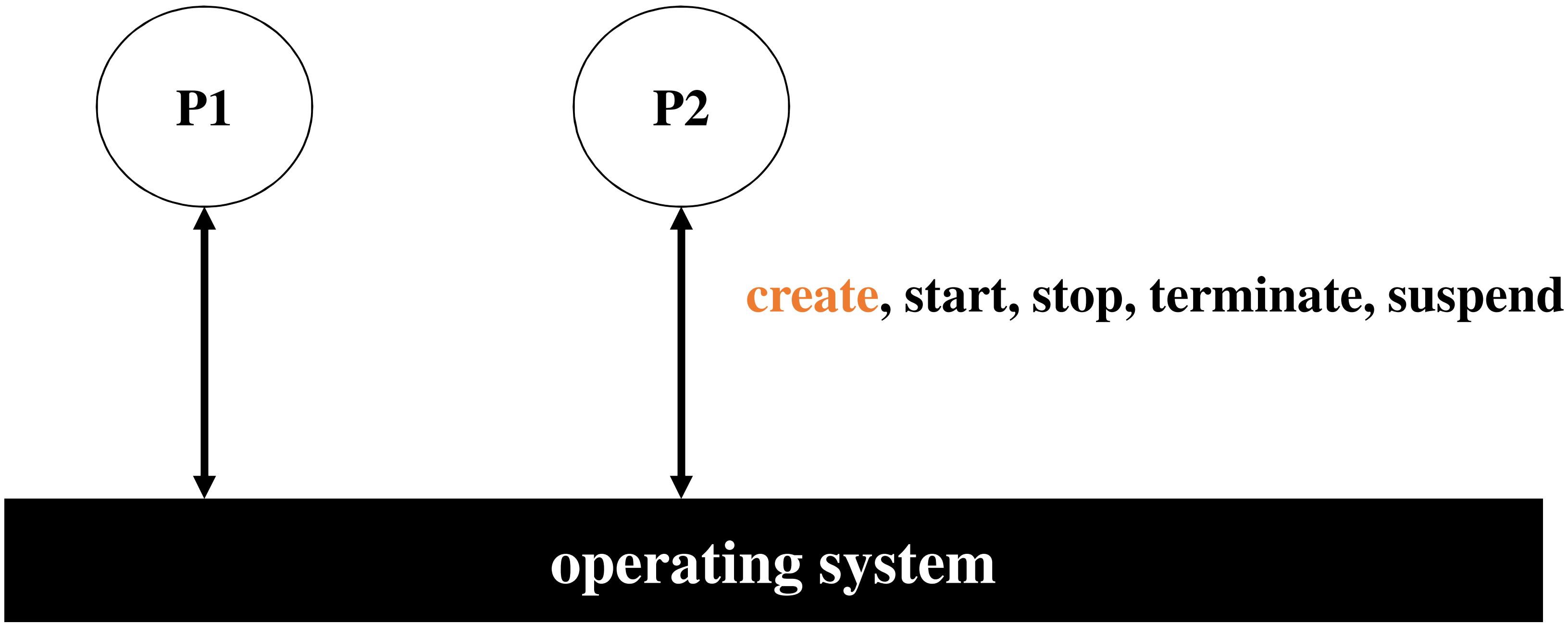
The actual performance difference may be minimal (tasks are mostly I/O bound)

Creating processes: *programmer's view*

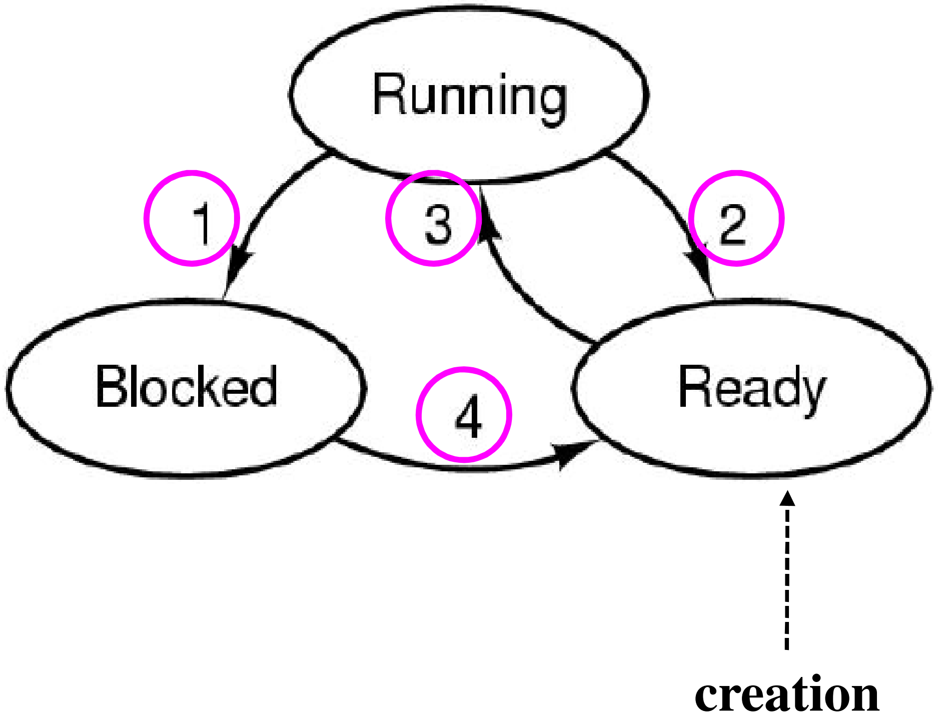
- processes can be created *dynamically*
- … the creator is called a parent
- … the created is called a child



Process management: *programmer's view*



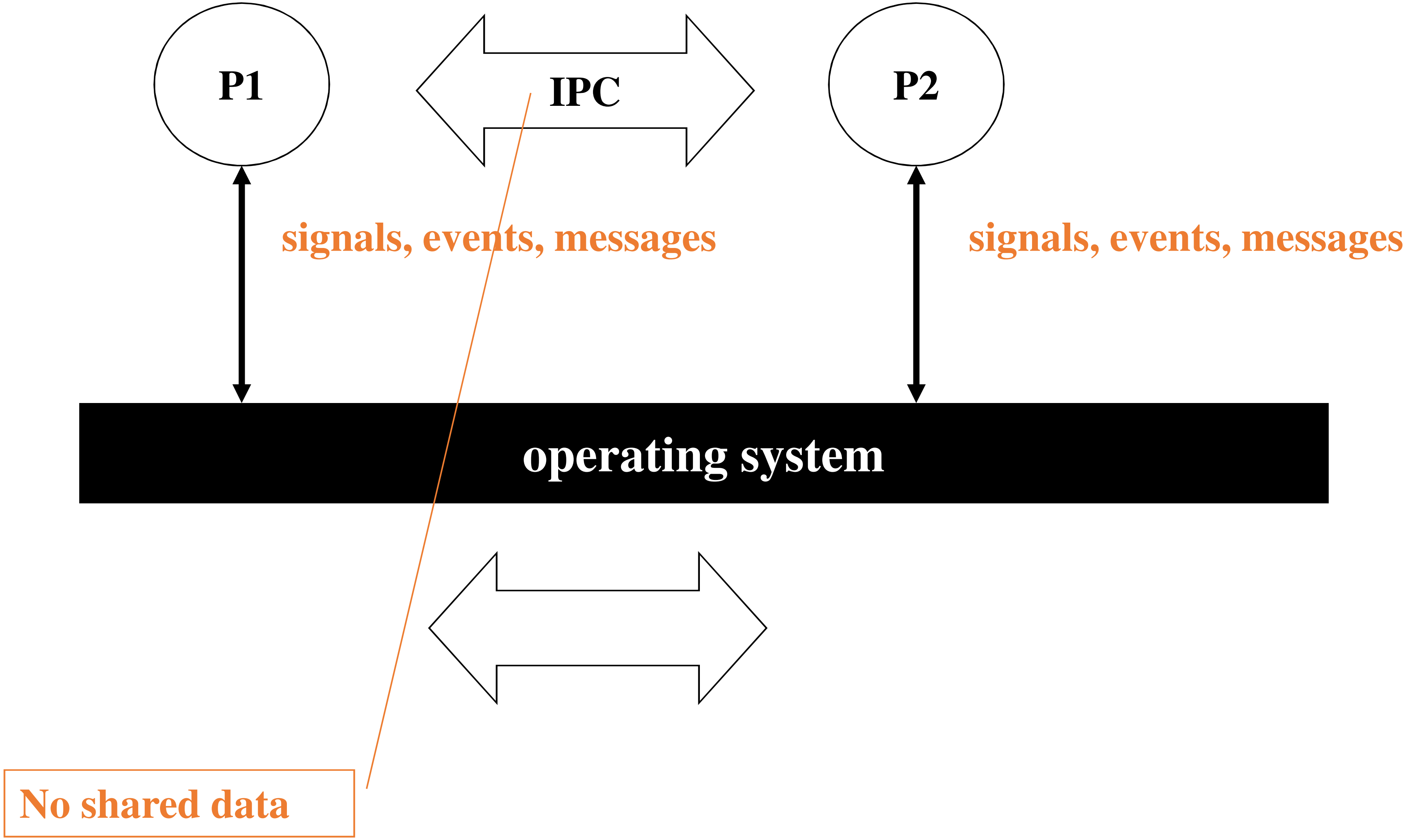
Process states: *programmer's view*



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Tanenbaum MOS

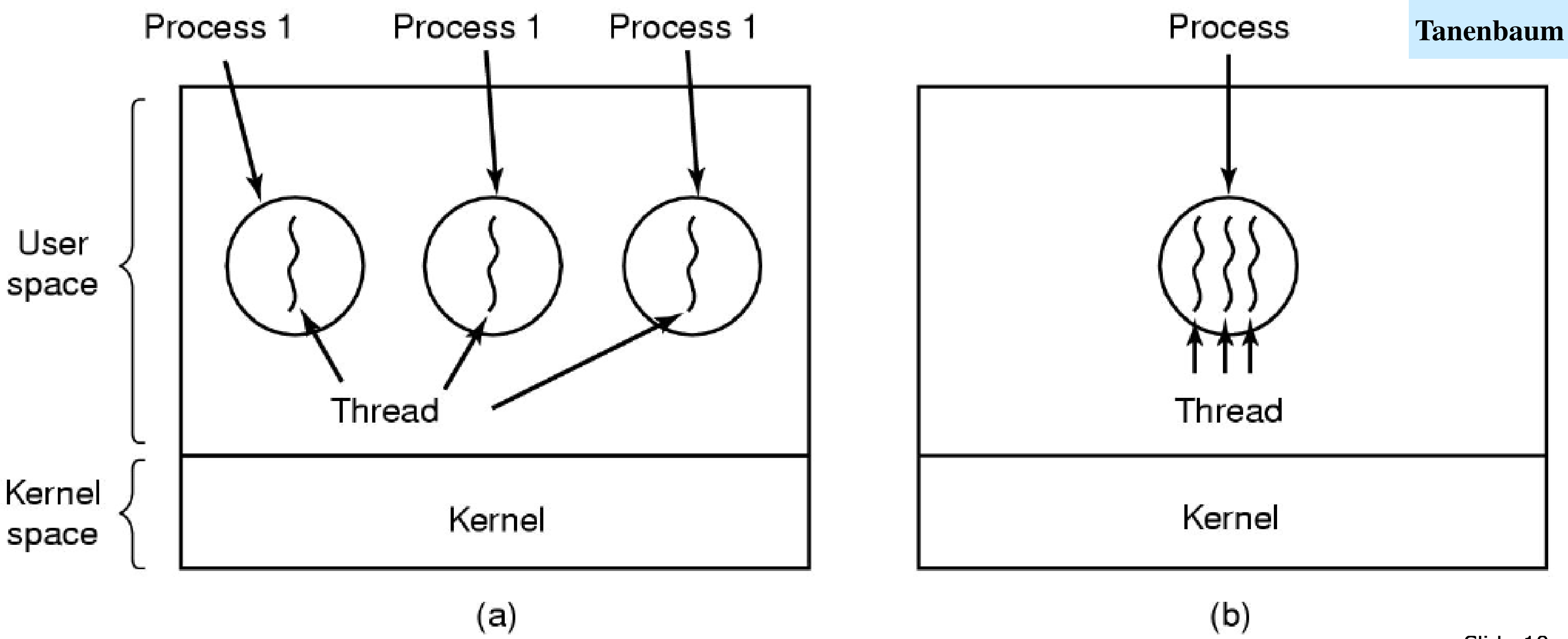
Inter-process communication: *programmer's view*



Lightweight processes: *threads (1)*

- a process has its own address space with single “thread” of control
  - no shared data
- but a process can have a number of *threads* in the same address space
  - shared data between threads
- more efficient to switch between threads
- .. again threads can be created and managed dynamically like processes

Tanenbaum MOS

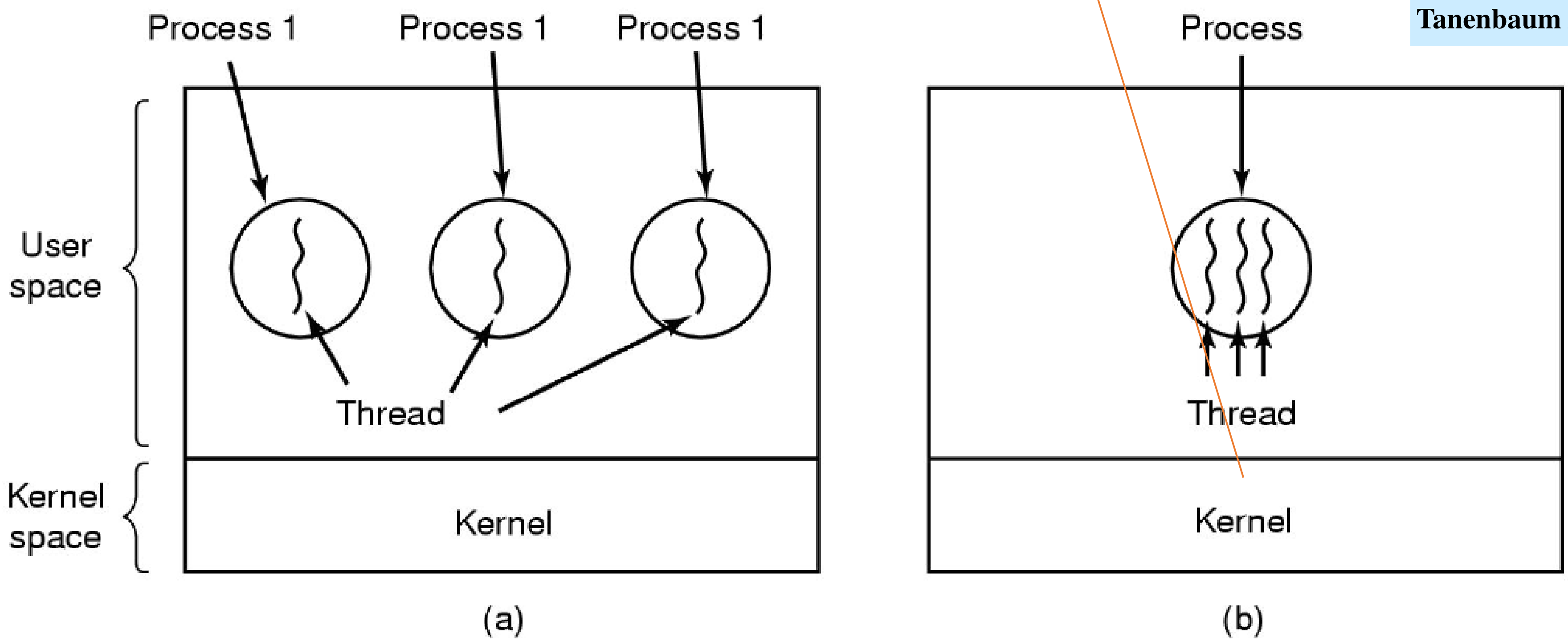


Lightweight processes: *threads (2)*

- OS handles calls such as:
  - new
  - start
  - run
  - isAlive
  - etc

a kernel thread may handle a number of user threads.

Tanenbaum MOS



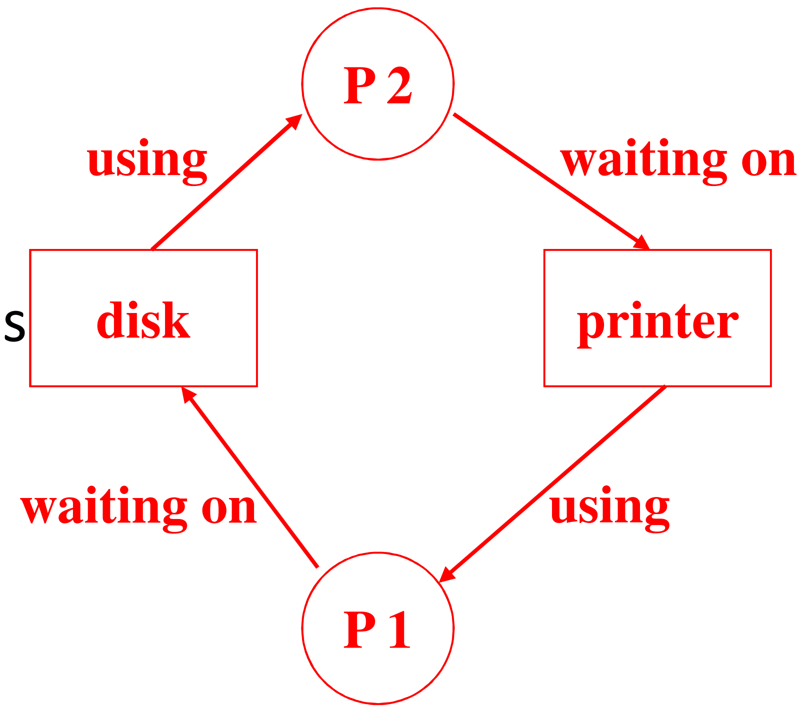
Slide 11

Synchronisation

- an issue for both threads and processes
- vying for a shared resource: the processes *interact*

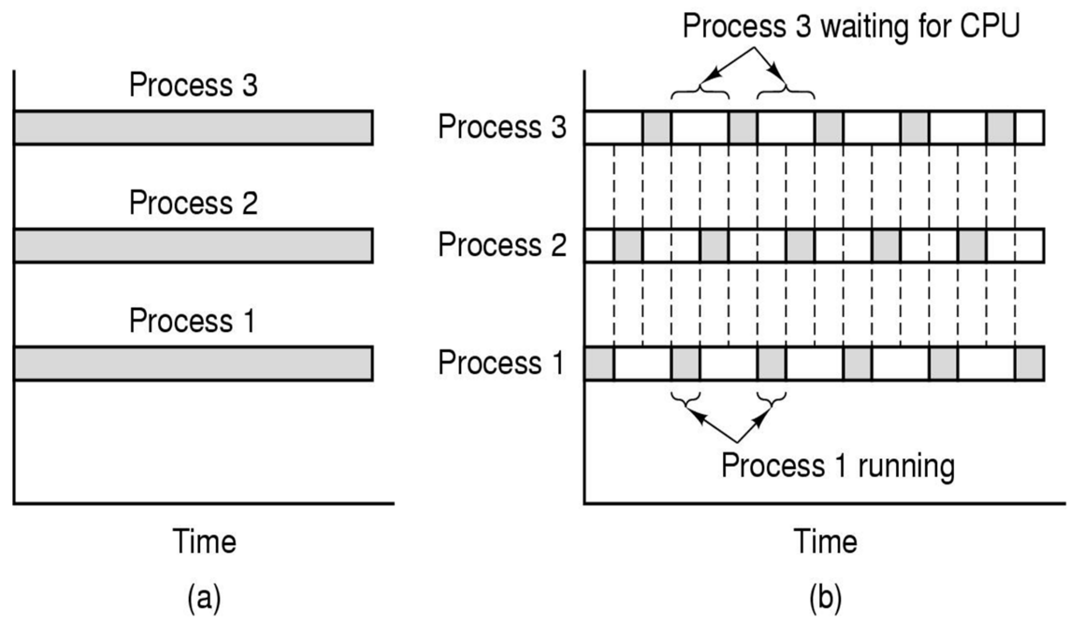
Data, files, devices

- many examples demonstrate difficulties of interacting threads, e.g.
  - dining philosophers
  - deadlock
- OS supports
  - critical regions
  - mutual exclusion
- … through synchronization primitives
  - synchronization
  - wait
  - notify/signal

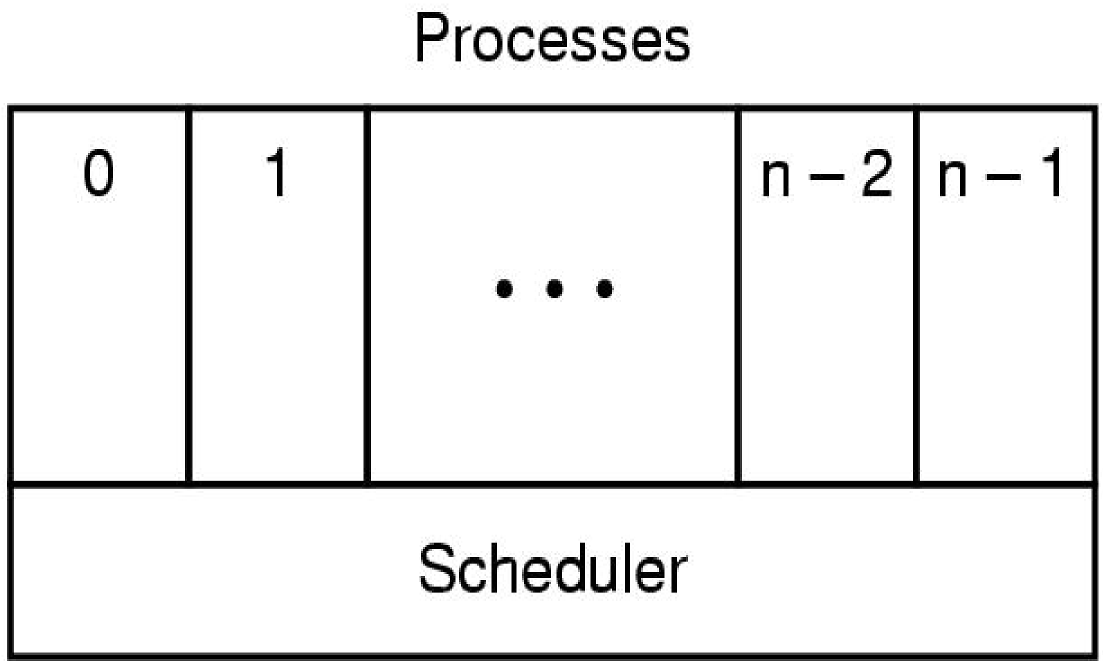


Process scheduling

- How does the OS decide what to do next?
    - Scheduling processes
    - Scheduling threads
- allocation of time
- processes may have different priorities
- must handle interrupts
- be efficient and fair



Tanenbaum MOS



What to do next?

- Can be easy to decide:
  - E.g. user presses a button on a mouse on a quiet system. (I.e. one with only one process in the ready queue)
- But can also be difficult
  - User is running a CPU-intensive job and (say) doing WP at the same time
- Poor scheduling can make even a powerful machine seem un-necessarily unresponsive

"Using TSO is like kicking a dead whale along the beach." (attributed so Steve Johnson)

TSO was an old IBM operating system. IBM used to make most of the mainframe computers in the world. Mainframe computers were what was around in the 1960's and 1970's (and still are) before PC's and servers took over…

Slide 14

Scheduling Issues

- How to schedule tasks depends on the tasks
  - Normal users are happy to wait 0.5 or even 1 or 2 seconds for service after pressing a button
    - Depends on users' expectations
    - Which depend on what the user is doing

  - Some equipment may require much faster service
    - Real-time equipment
      - Which includes peripherals like discs, CDs
      - Also sound and video interfaces
      - As well as less common equipment
        - Robot arms
        - PET scanners,….

- The operating system scheduler needs to know what sort of process each one is.

Slide 15

Scheduling issues cont'd

- If users wait longer than 1-2 seconds they may well start pressing more buttons
  - Or even give up
- They will complain that the system ***lacks response***
  - Yet all it may be is that the system scheduler is delaying servicing their requests in order to service other requests
- Scheduling needs to take the tasks into account
  - Which is why a desktop OS is fine for a desktop PC, but not necessarily for machines running external equipment.
  - These often have their own real-time operating system schedulers

Slide 16

Scheduling methods

- Choices:
  - Run started task to completion or
  - Interrupt (pre-empt) tasks to give service to others
    - Time-slicing
- Pre-emption has an overhead
  - Context-switching
- But non-pre-emptive systems may starve some processes
  - Possibly indefinitely

Slide 17

Example cont'd

Process	Time
P1	1.0
P2	0.3
P3	0.2
P4	2.0

- **In turn**

Process	P1	P2	P3	P4
Wait time	0	1	1.3	1.5
Fin. time	1	1.3	1.5	3.5

- **Shortest first**

Process	P3	P2	P1	P4
Wait time	0	0.2	0.5	1.5
Fin time	0.2	0.5	1.5	3.5

- **Time-slicing, 0.2 seconds quantum.**

P1	P2	P3 (T) 0.6	P4	P1	P2 (T) 1.1	P1	P4	P1	P4	P1 (T) 2.1	P4	P4	P4	P4	P4	P4	P4 (T) 3.5	
0	0.2	0.4	0.6	0.8	1.0	1.1	1.3	1.5	1.7	1.9	2.1	2.3	2.5	2.7	2/9	3.1	3.3	3.5

Slide 18

End of Systems Lectures