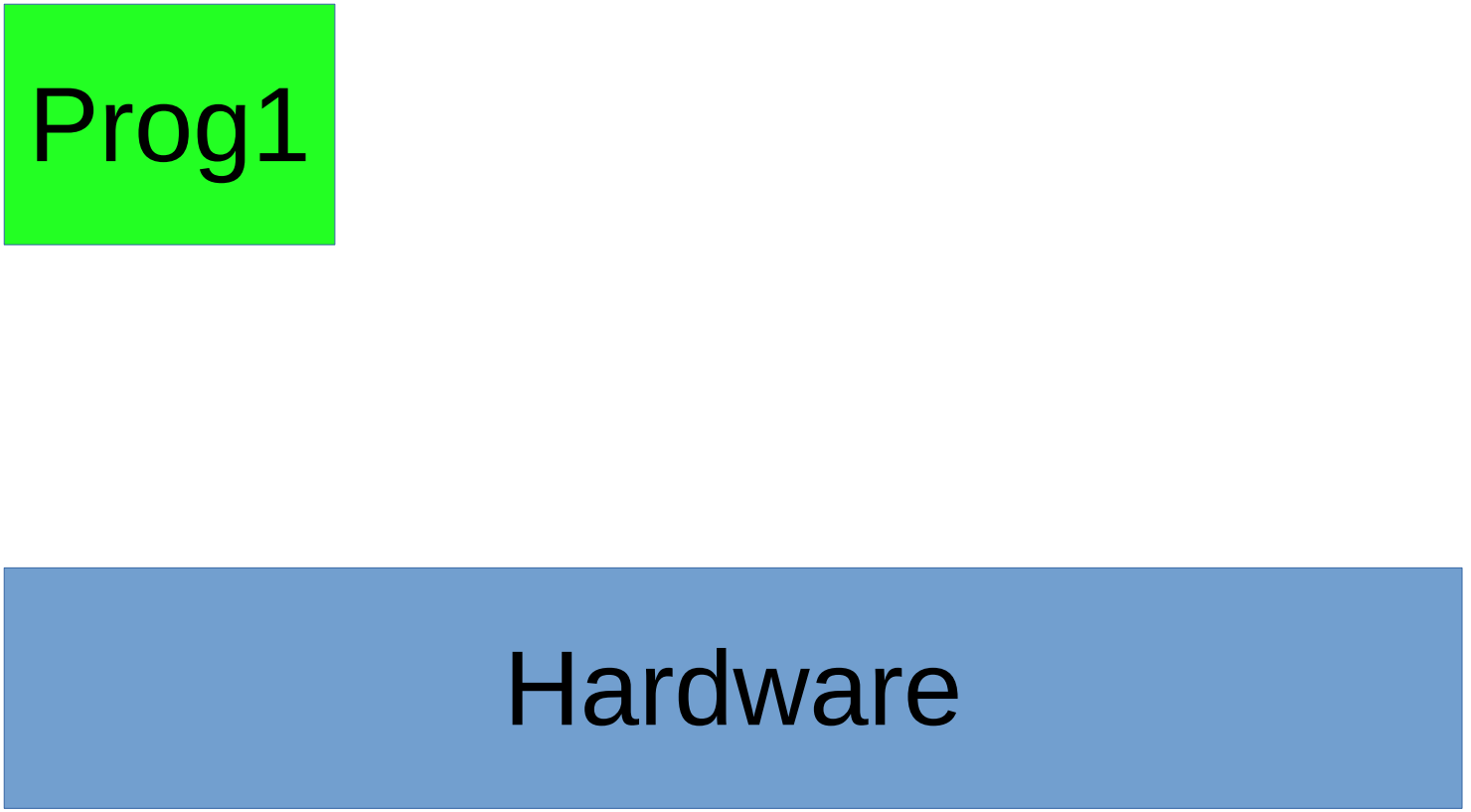
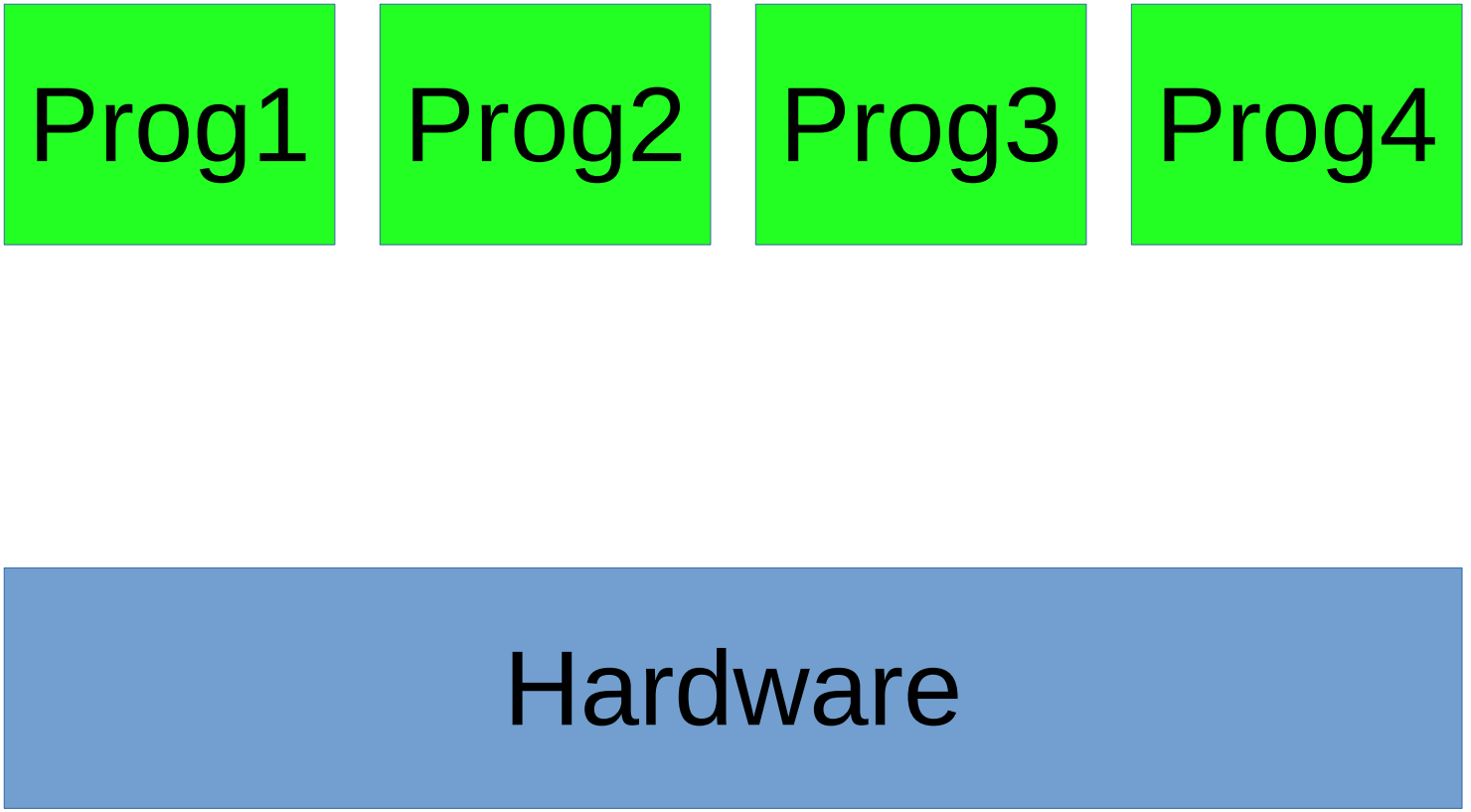
TDDB68/TDDE47
Concurrent Programming and Operating Systems

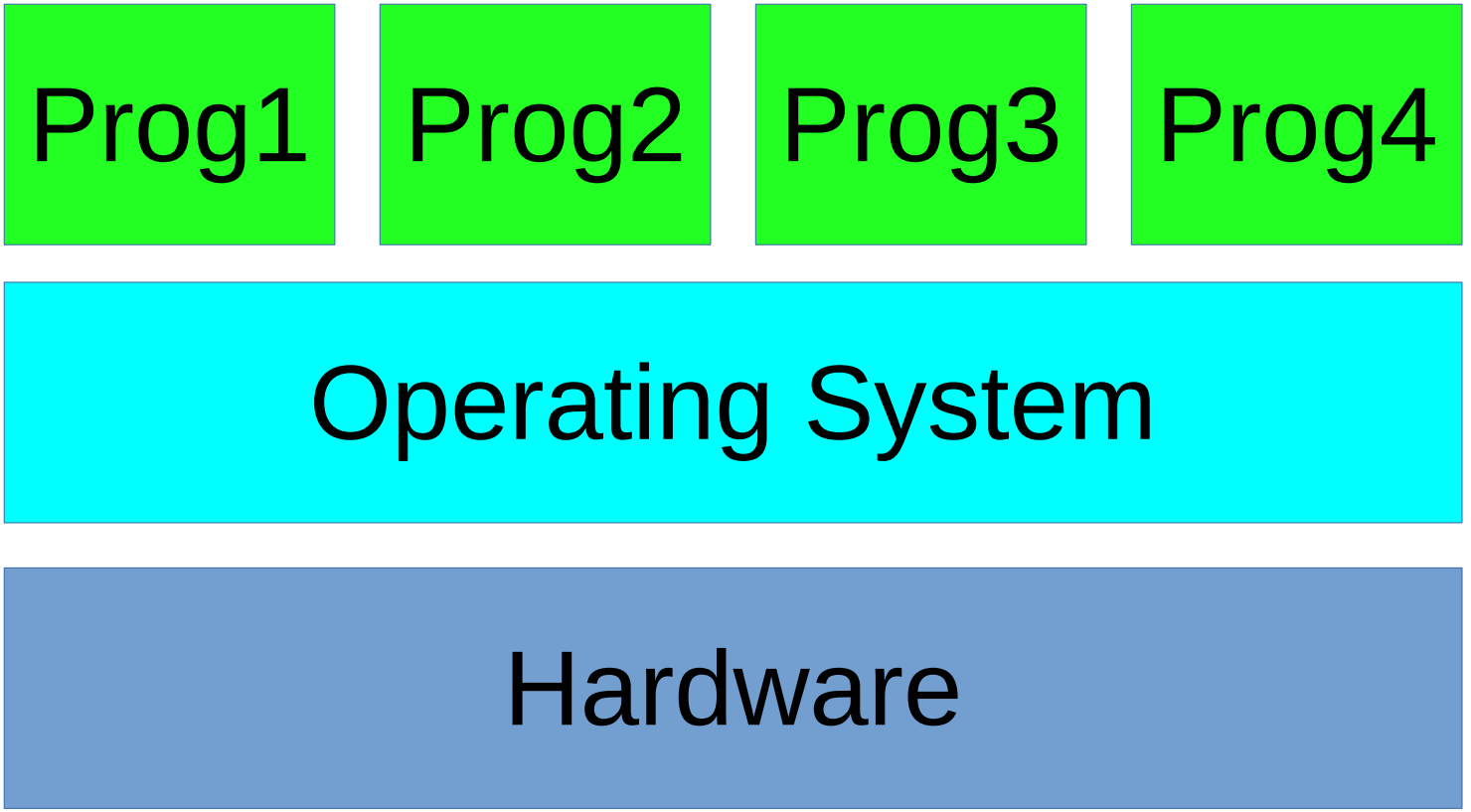
Lecture 1:
Introduction, interrupts and system calls

Mikael Asplund
Real-time Systems Laboratory
Department of Computer and Information Science

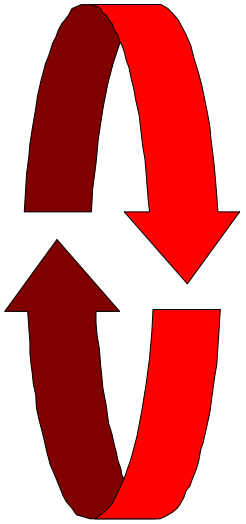
Hardware


Prog1
Hardware

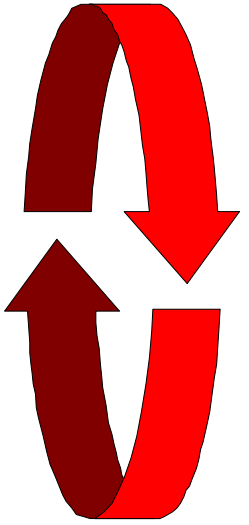

Prog1
Prog2
Prog3
Prog4
Hardware


Prog1
Prog2
Prog3
Prog4
Operating System
Hardware

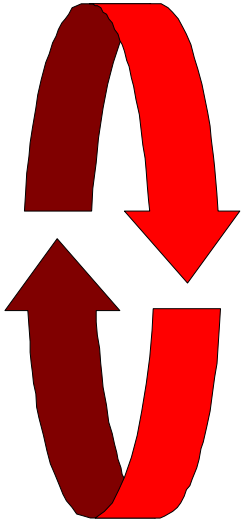
Concurrency



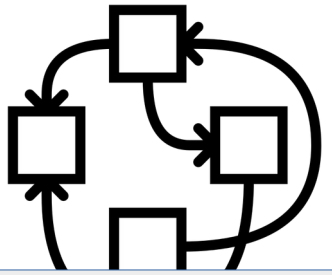
1. Do this
2. Do that
3. Wait for X
4. Send Y
5. Do something
6. Wait a while
7. Do something else



1. Send Z
2. Wait for X
3. Do nothing
4. Send something else
5. Do this

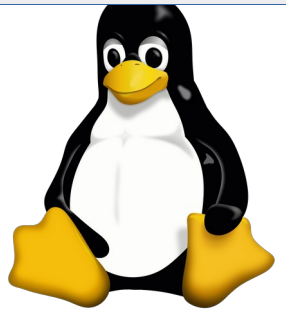


1. Wait for query
2. Connect to database
3. Send db-query
4. Wait for results
5. Create web page
6. Reply to request



In this course:

Focus on basic principles



Courses

- TDDB68: 6hp course for D + U + I + M.Sc. + Erasmus
  - C/C++ with pointers a prerequisite
- TDDE47: 8hp course for IT
  - More time and some PBL sessions for learning about memory allocation and pointers
- TDDE47 and TDDB68 have the same exam and labs.
- TDDD82: Three lectures on processes and synchronization shared

Course web

- TDDB68:
  - https://www.ida.liu.se/~TDDB68
- TDDE47
  - https://www.ida.liu.se/~TDDD47
- Same content – report if anything is strange

People

- Examiner: Mikael Asplund
- Course assistant: Felipe Boeira
- Lab assistants:
  - TDDB68 Group A: Felipe Boeira
  - TDDB68 Group B: Rodrigo Saar Moraes
  - TDDB68 Group C: Jesper Jonsson
  - TDDB68 Group D: Viktor Hellqvist
  - TDDB68 Group E: Benjamin Hansson
  - TDDB68 Group F: Oscar Bergman
  - TDDE47 group: Erik Norrestam Held

Examination

- Labs 3hp for TDDB68, 5hp for TDDE47
- Written exam 3hp
- Bonus assignment 0hp
  - 4p bonus points on the exam
  - Pass all labs no later than **2020-03-13**
  - Need to be first-time registered in VT1 2020

Labs

- Pintos
  - Stanford educational OS
- Challenging!
- Register in webreg today
  - Only new students!

Deadlines

- **2020-01-23** Sign up in webreg (new students)
- **2020-03-13** Lab deadline to get the bonus points on the exam
- **2020-03-30** Last chance to pass the labs in VT1

Lab feedback

- **Promise:**
  - No later than 2w from the soft deadline
  - Lab6 at least one chance to correct if you hand in on the 10th

- Soft deadlines:
  - Lab0: January 29
  - Lab1: February 5
  - Lab2: February 15
  - Lab3: February 21
  - Lab4: February 27
  - Lab5: March 4
  - Lab6: March 10

Time management

- Lectures 8*2h + 1*4h
- Lessons 3*2h (to prepare for labs)
- Labs 36h (42 for TDDE47)
- Self-study TDDB68: ~100h, TDDE47: ~145h

Time management

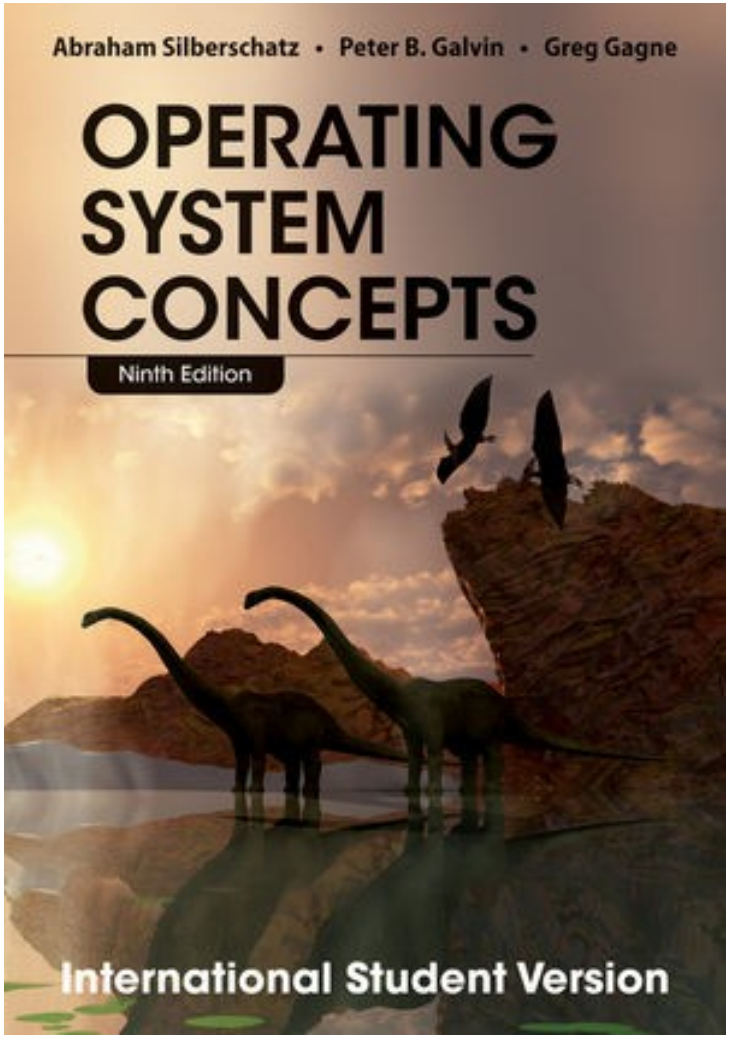
- Le
- Le
- La
- Self-study TDDB68: ~100h, TDDE47: ~145h

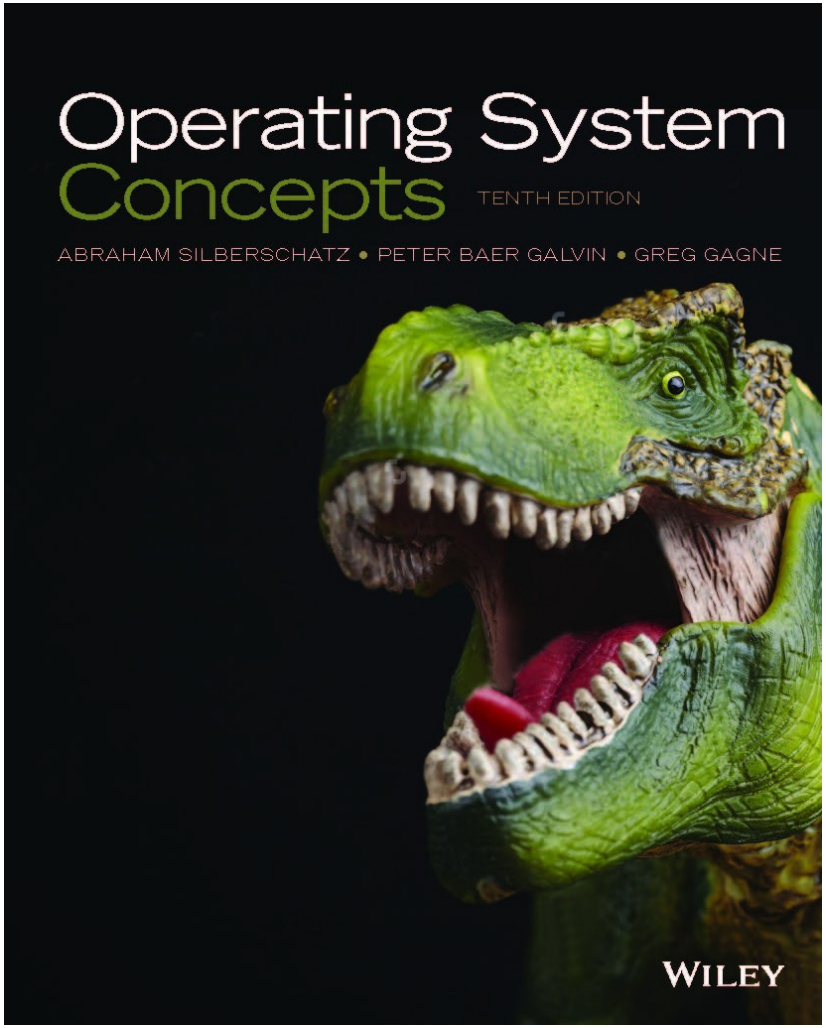
If you’re not spending **10+ hours/week** of free timedoing labs and studying for this course, you will be in trouble.

Do not cheat!

- Labs are hard and you might find solutions online
- It is **not allowed** to copy anyone else's solution
- We are obliged to report suspected plagiarism to the disciplinary board

Course book


Abraham Silberschatz • Peter B. Galvin • Greg Gagne
OPERATING SYSTEM CONCEPTS
Ninth Edition
International Student Version


Operating System Concepts
TENTH EDITION
ABRAHAM SILBERSCHATZ • PETER BAER GALVIN • GREG GAGNE
WILEY

Exam

- Previous exams available
- Solutions **not available**
- New this year: part A and B
  - Part A: must be passed, will contain multiple choice questions, one question on concurrency, and one multi-part question on basic knowledge
  - Part B: will only be corrected if part A has been passed, only used to determine higher grades (4 or 5)

Lectures vs. Labs

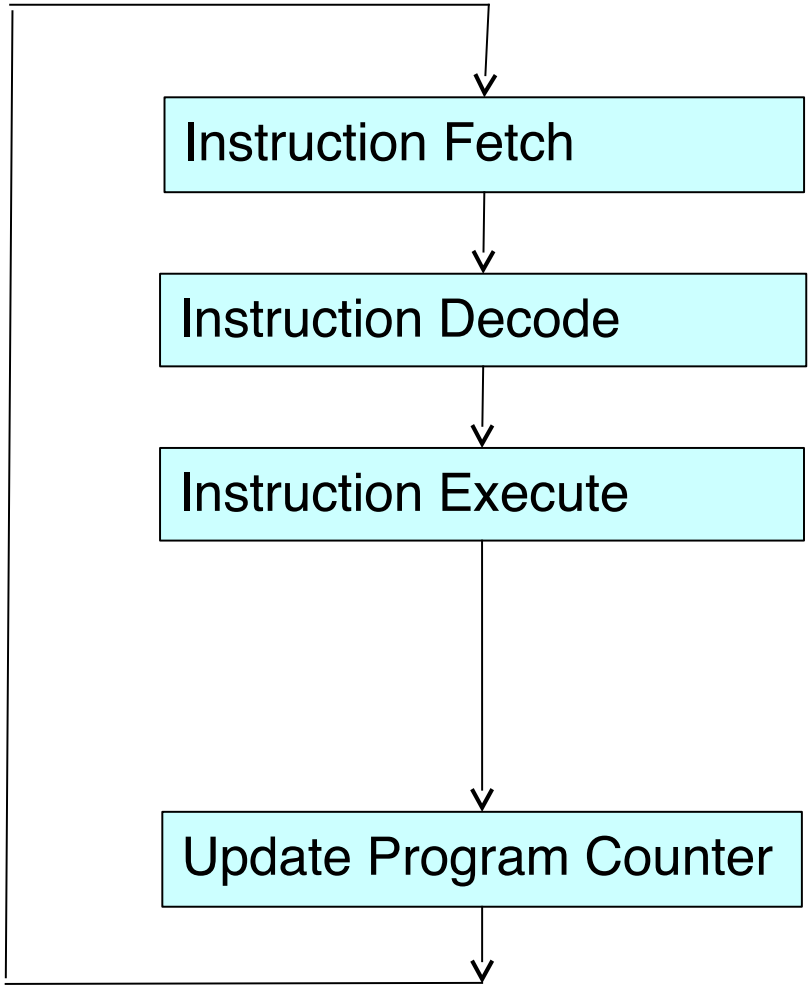
- Intentionally different scope
- Lectures provide an overview of a vast area
- Labs provide hands-on experience with a particular educational OS.

Changes from last year

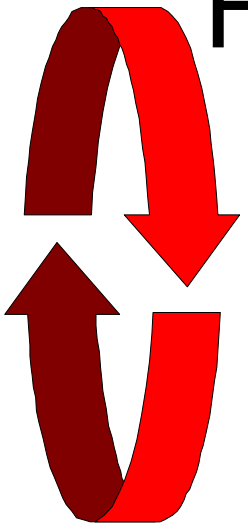
- Lab structure
  - Lessons in time
- More lab assistants

What does an operating system do?

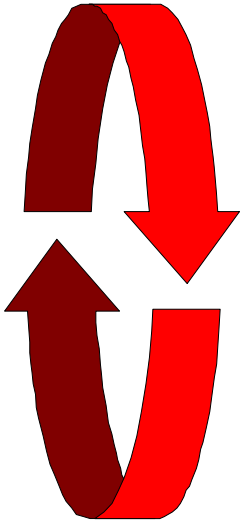
Program execution (von-Neumann cycle) by a processor



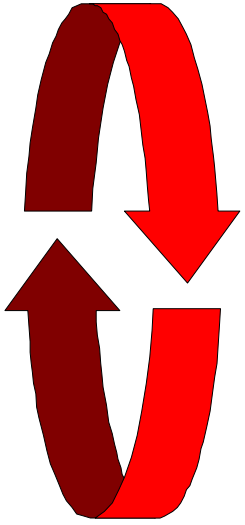
How to do concurrency?



1. Do this
2. Do that
3. Wait for X
4. Send Y
5. Do something
6. Wait a while
7. Do something else

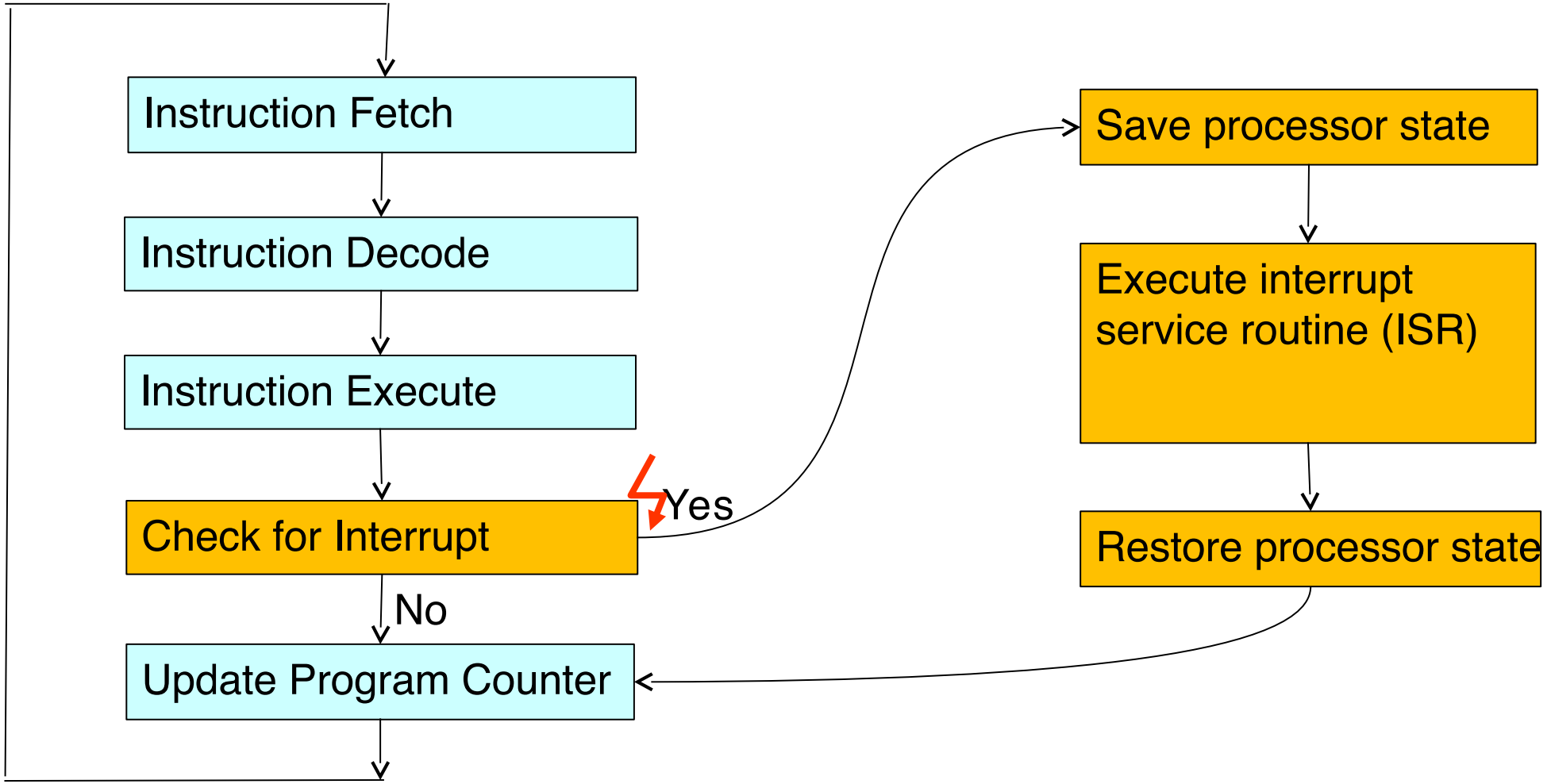


1. Send Z
2. Wait for X
3. Do nothing
4. Send something else
5. Do this

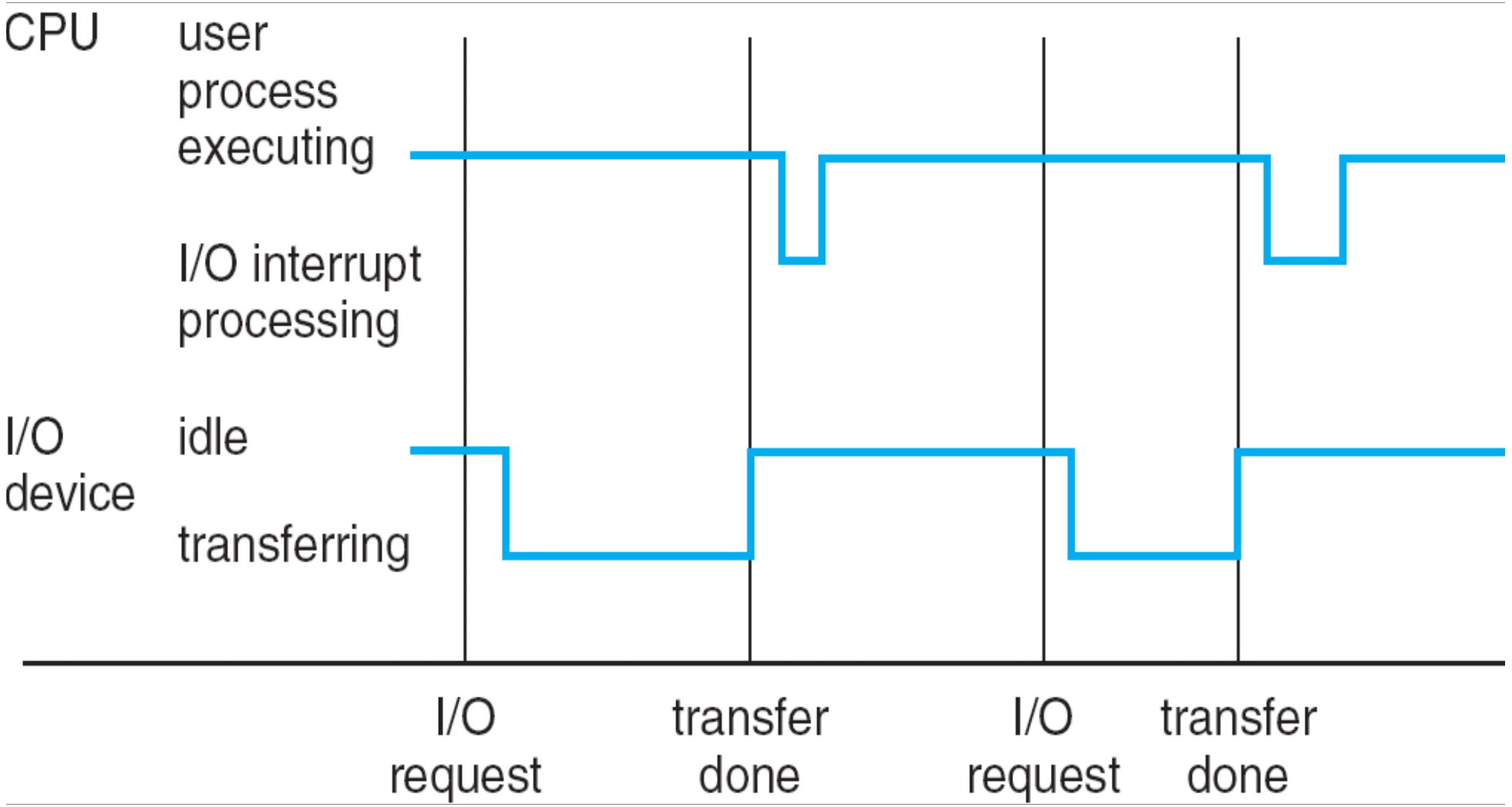


1. Wait for query
2. Connect to database
3. Send db-query
4. Wait for results
5. Create web page
6. Reply to request

Interrupts


Instruction Fetch
Instruction Decode
Instruction Execute
Check for Interrupt
Yes
No
Update Program Counter
Save processor state
Execute interrupt service routine (ISR)
Restore processor state

Interrupt Timeline



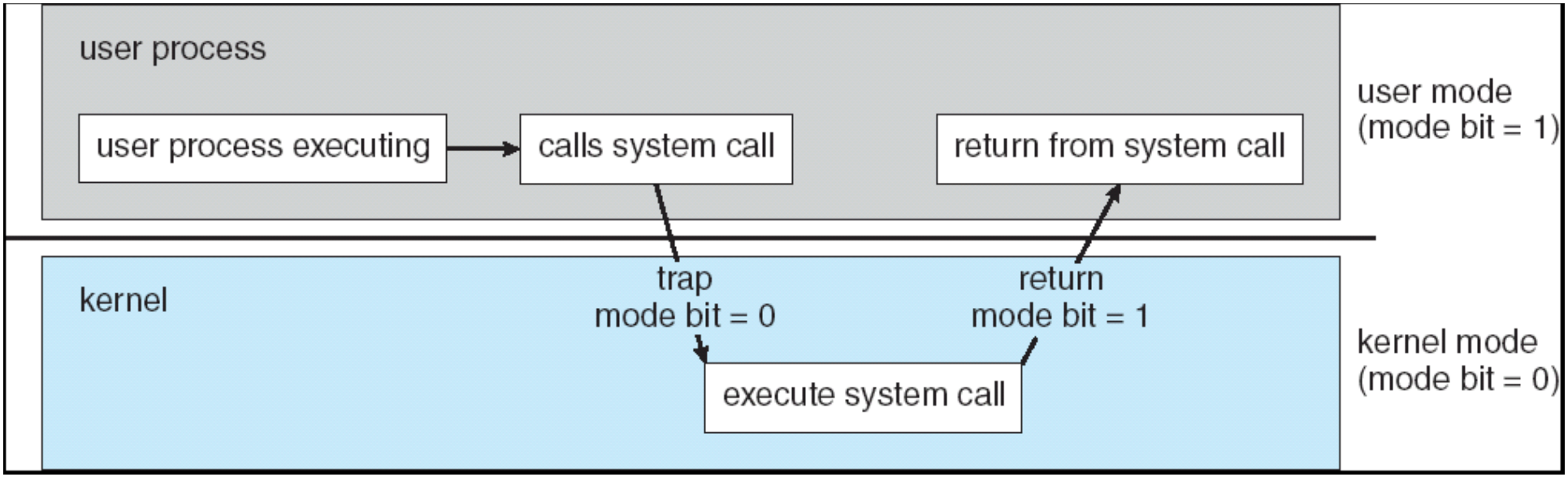
Software Interrupts

- A ***trap*** is a *software*-generated interrupt caused either by an error or a user request.
  - Examples: Division by zero;
    Request for OS service

An operating system is ***interrupt*** **driven**.

Why not just have a function API?

Dual Mode


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

Types of System Calls

- **Process control**
  load, execute, end, abort, create, terminate, wait ...
  memory allocation and deallocation
- **File management**
  open, close, create, delete, read, write, get/set attributes...
- **Device management**
  request / release device, read, write, ...
- **Information maintenance**
  get / set time, date, system data, process / file attributes
- **Communications**
  create / delete connection, send, receive, ...

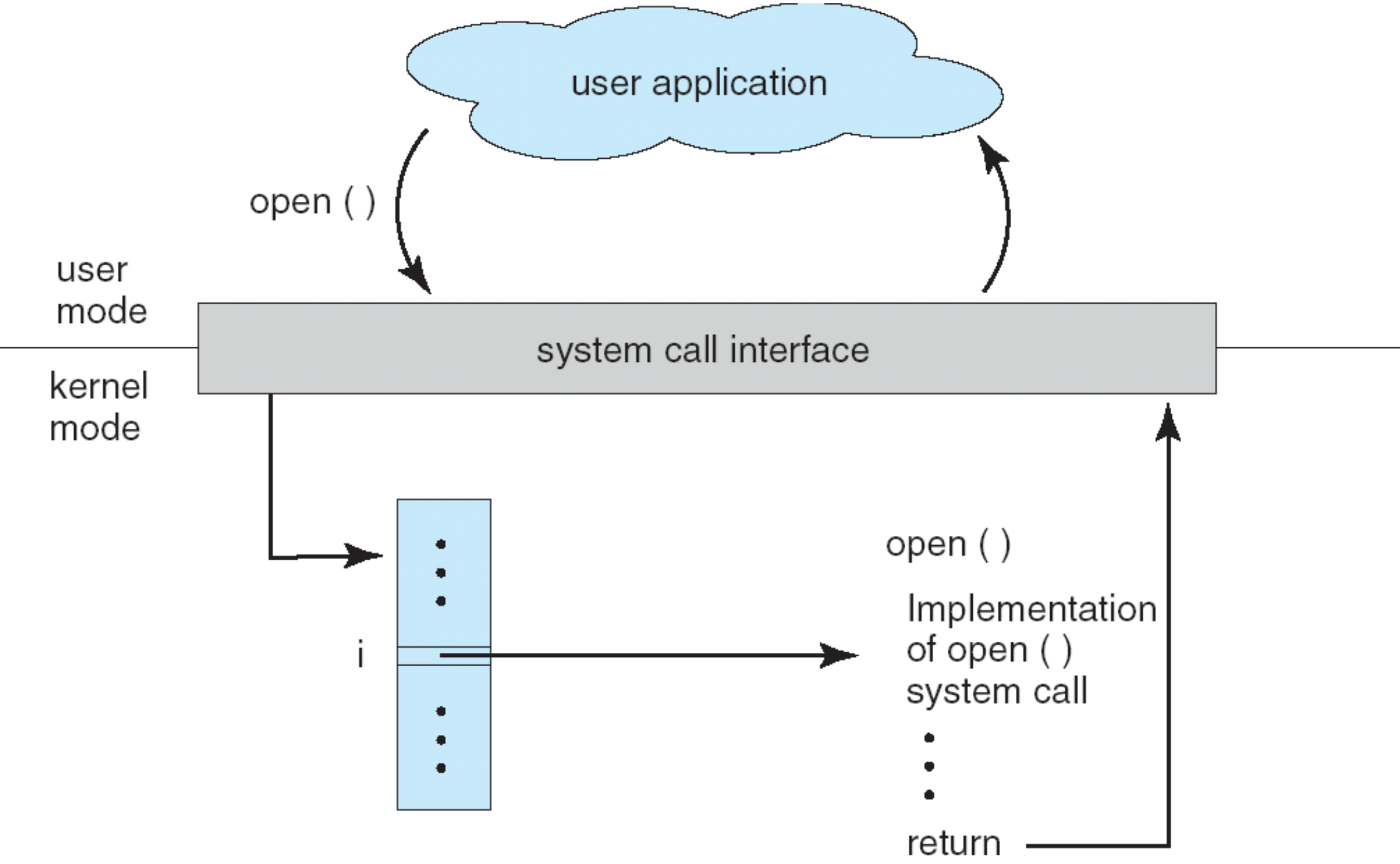
Syscalls In Linux

(man syscalls)

Tracing system calls

Linux – ltrace
Mac OSX - dtrace

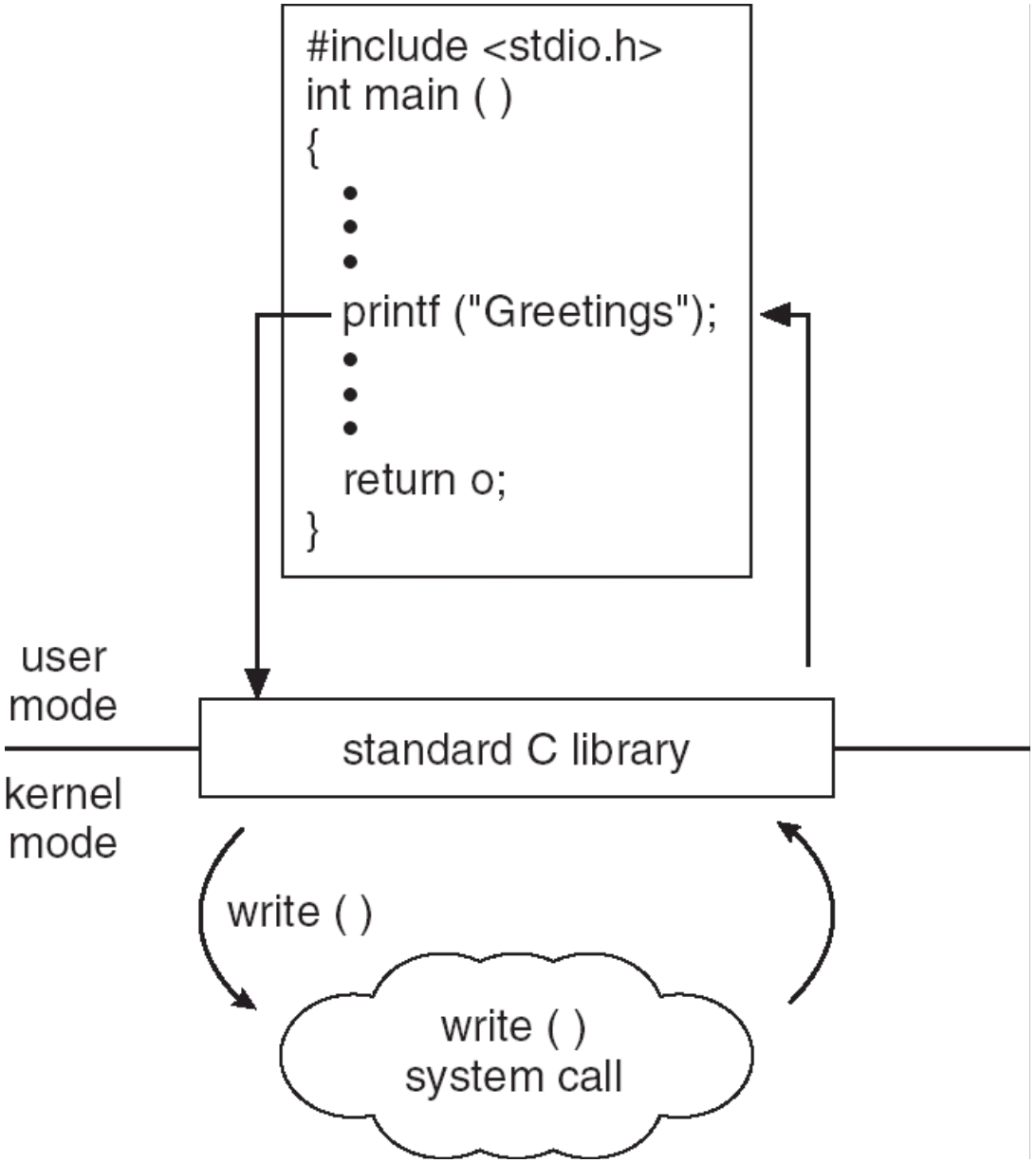
System Call API – OS Relationship



System Call API Implementation

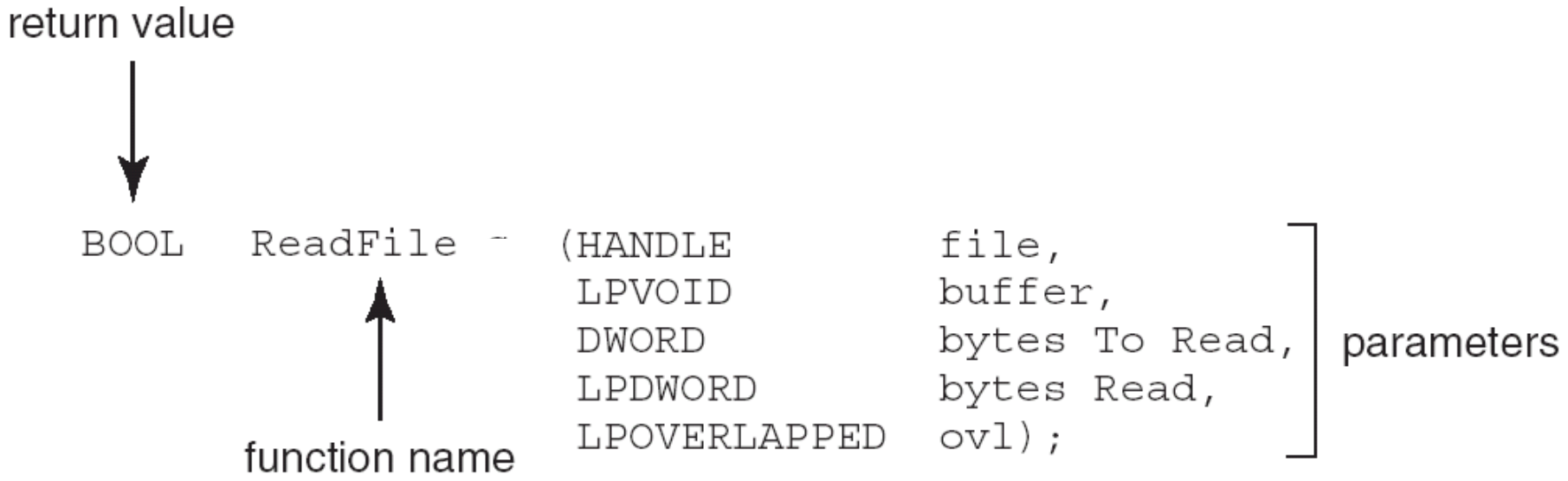
- **System call implementation**
  - hardware-specific
- **System call interface**
  - usually, in C
  - standard across multiple platforms
- **Advantage:**
  - Caller does not need to know anything about how the system call is implemented
  - Most details of OS interface hidden from programmer by API

Printf example



Example of a System Call API

- ReadFile() function in Win32 API (function for reading from a file)



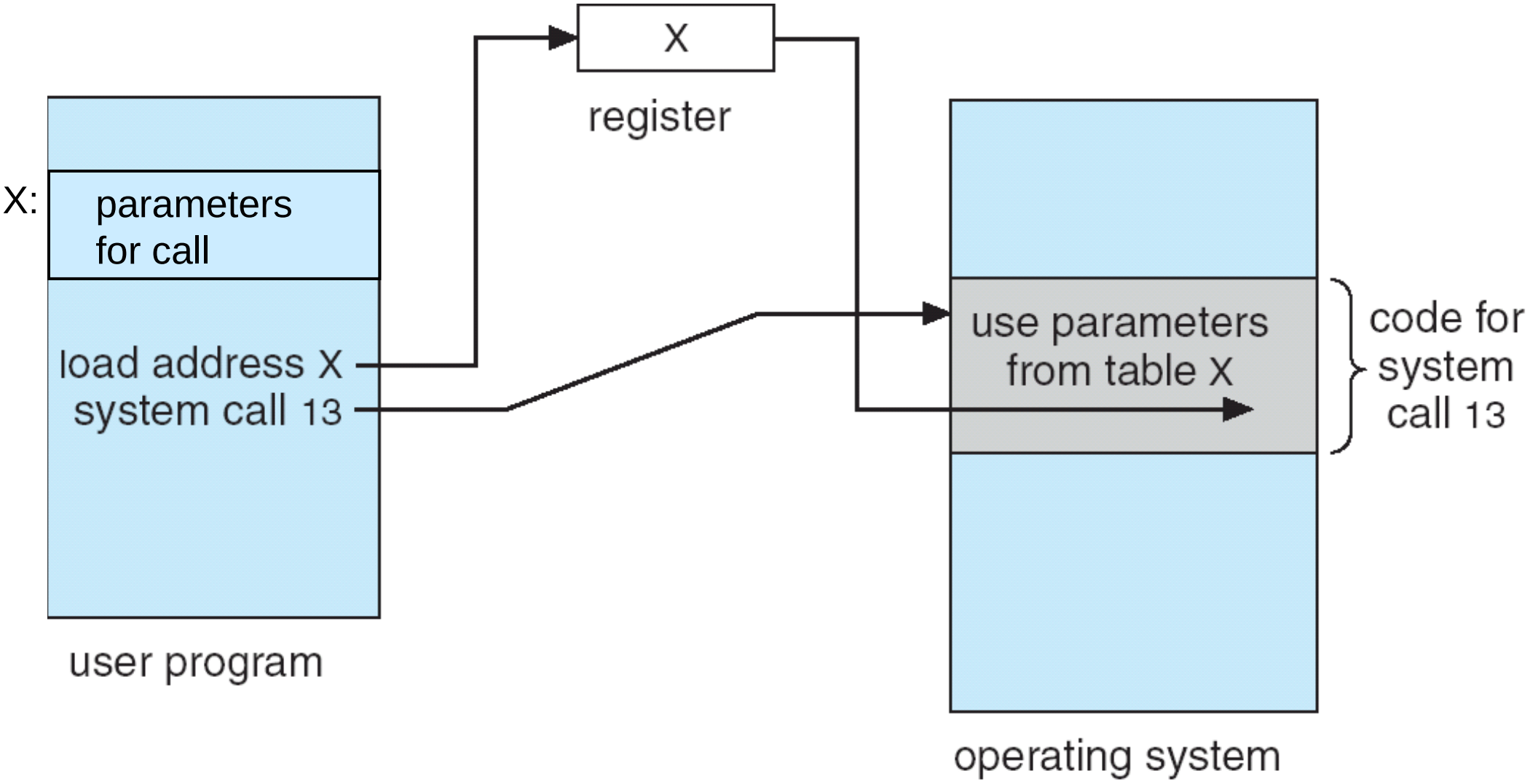
- Parameters passed to ReadFile():
  - file — the file to be read
  - buffer — a buffer where the data will be read into and written from
  - bytesToRead — the number of bytes to be read into the buffer
  - bytesRead — the number of bytes read during the last read
  - ovl — indicates if overlapped I/O is being used

System Call Parameter Passing

- Three general methods used to pass parameters to syscalls:
  - Parameters **in *registers***
  - Parameters **in a *block in memory***, and address to block in a register
    - This approach taken by Linux
  - Parameters ***pushed onto the stack***

- What are the advantages/disadvantages?

System Call Parameter Passing via Block



System Programs

- User's view of an Operating System
- Provide a convenient environment
  - File management
  - Program development
  - Graphical user interface
  - ...

Summary

- **Operating System** = OS Kernel + System Programs
  - Mediates all accesses to system resources
  - **Interrupt**-driven
    - Error handling
    - Controlled access to system resources, e.g.
      - I/O devices, DMA
      - CPU time sharing
    - …
- **Dual-Mode** (user mode, kernel mode)
  - **System Call** API for portability

Reading guidelines

- 9th edition
  - Chapter 1: Sections 1.1-1.7
  - Chapter 2: 2.3-2.5
- 10th edition
  - Chapter1: 1.1-1.5
  - Chapter2: 2.3-2.4

What's next?

- Today
  - Sign up in Webreg!
  - Get the book
- Tomorrow
  - 13-17 Introductory C lecture
  - Try some C programming
- Thursday
  - 8-10 lesson to introduce the labs
- Friday
  - First lab (lab0)