EDAF35: OPERATING SYSTEMS

MODULE 1

INTRODUCTION, OVERVIEW

EDAF35 MODULE 1

CONTENTS

- **Introduction:**
  Motivation, OS Roles, Course Aim, Prerequisites (Quick Recap)

- Organization:
  Lectures, Laboratories, Project, Examination, Support

- Overview of an OS:
  Views, Components, Functionality, Examples

(Material loosely based on the course book, Chapters 1 and 2)

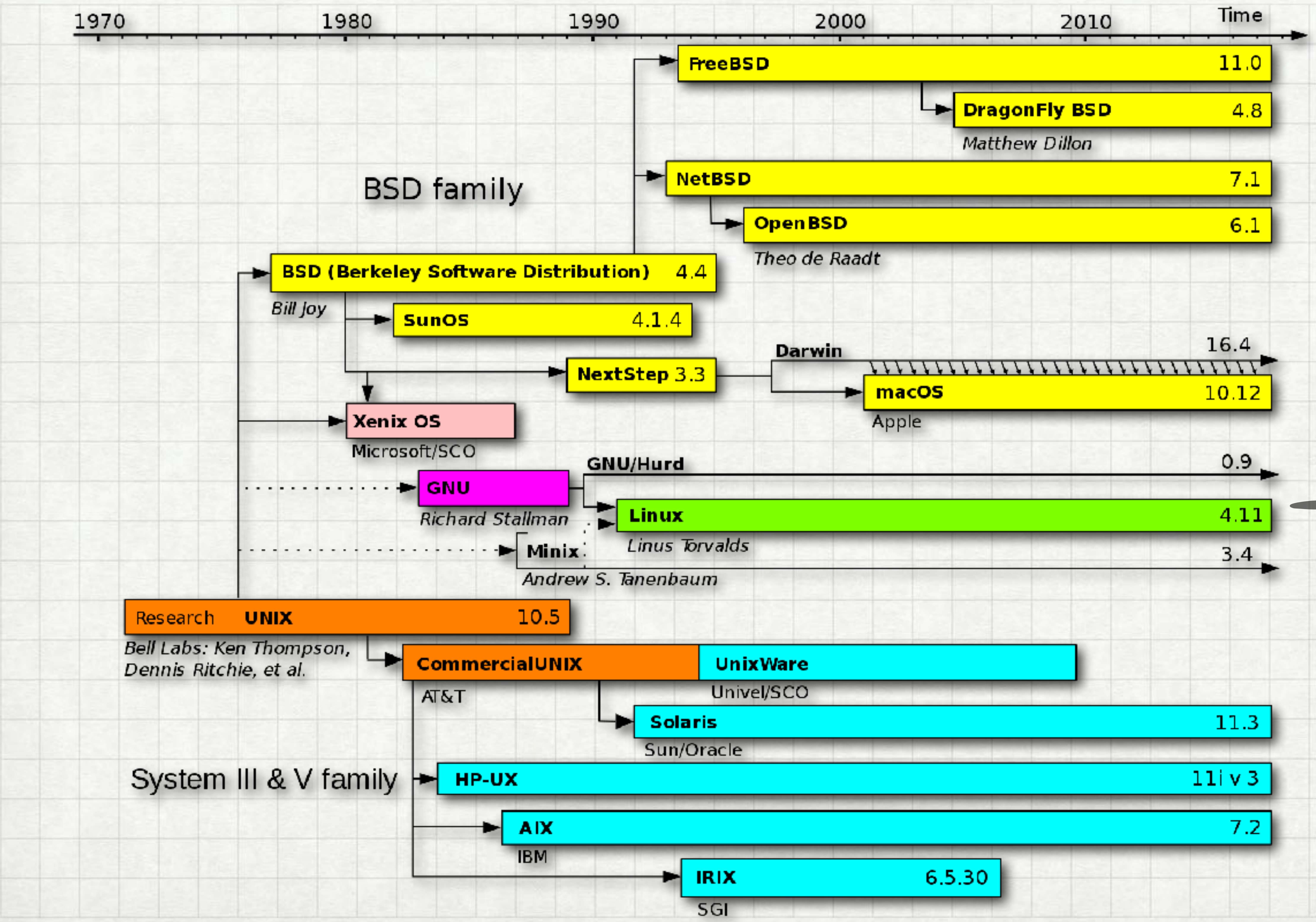
WHAT IS AN OPERATING SYSTEM?

INTRODUCTION

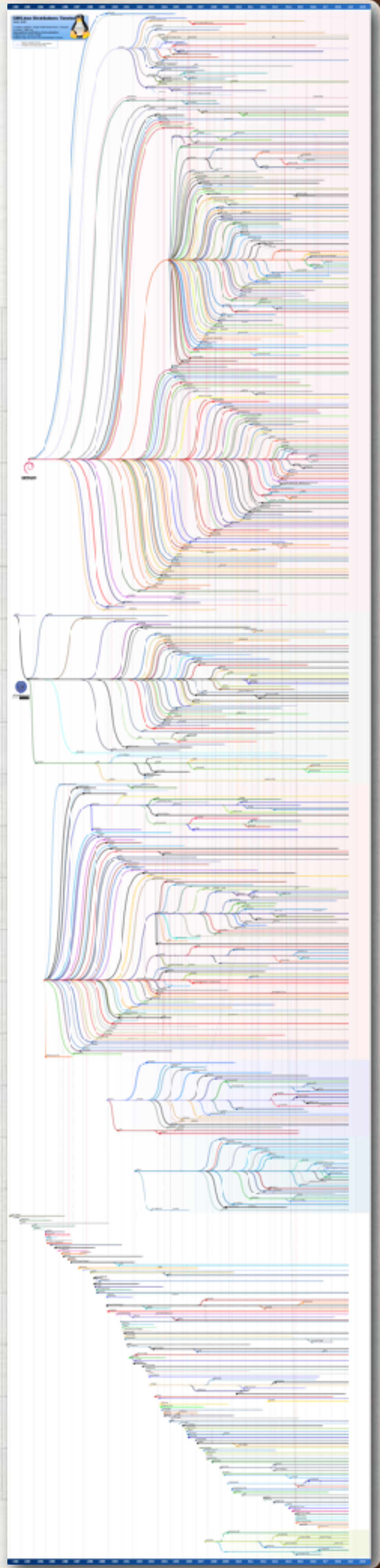
Convenient and/or Efficient

(THE UNIX/LINUX TIMELINE)

INTRODUCTION

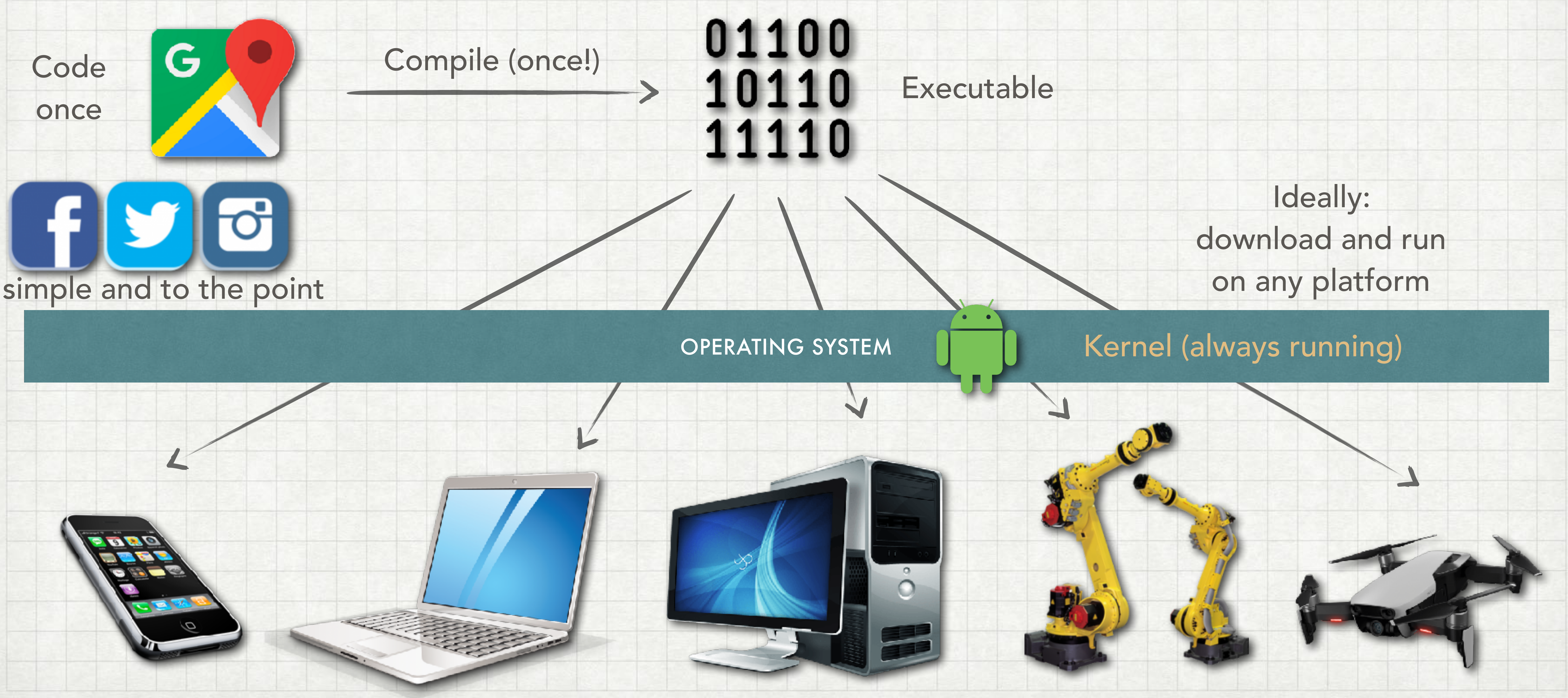


Linux Distributions



BUT WHAT IS AN OPERATING SYSTEM?

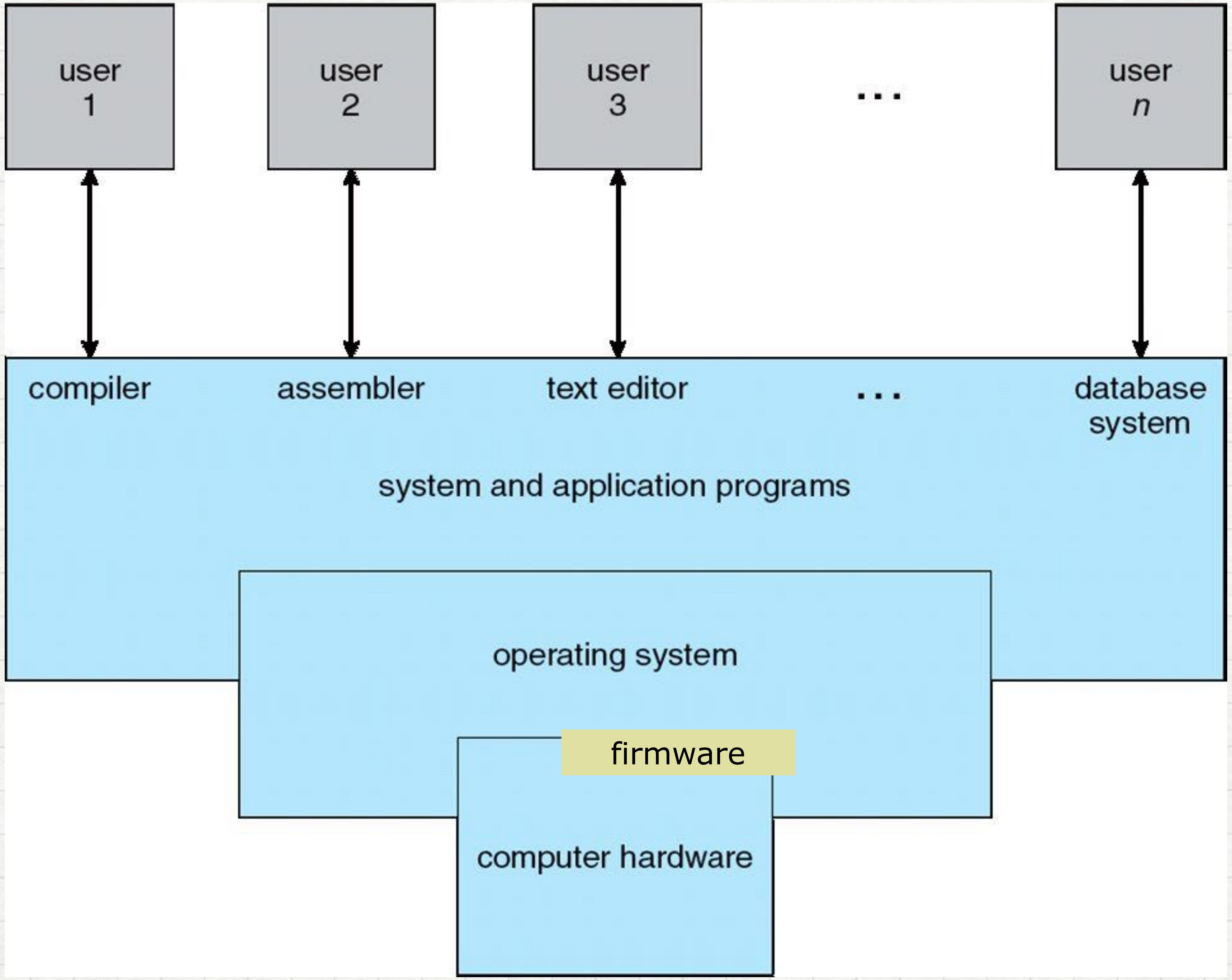
INTRODUCTION



FOUR COMPONENTS OF A COMPUTER SYSTEM

INTRODUCTION

Users



Applications

OS

Hardware

OS ROLE 1: ABSTRACT MACHINE

INTRODUCTION

- Presents a standard set of high-level abstractions
- Extends the hardware with more functionality
- Abstracts away from hardware details

▶ Programmer friendly, Portable code, Reusable/common core

OS ROLE 2: RESOURCE MANAGER

INTRODUCTION

- Allocates resources for users/applications
  (processor time, memory, disk space, device access, network bandwidth, etc.)

- Ensures:
  progress, policies, safety/error handling, efficient use of resources

▶ Safe interaction between applications, safe and fair use of hardware

COURSE AIM

INTRODUCTION

- General elements and principles in OS
- User, application programmer, and OS developer viewpoints
- OS design and implementation choices, internal operations
  - hand-on experience in laboratory and project assignments
  - more experience with C and related development tools
- Examples of the above in specific OS
  - focus on Linux and related

ASSUMED BACKGROUND KNOWLEDGE

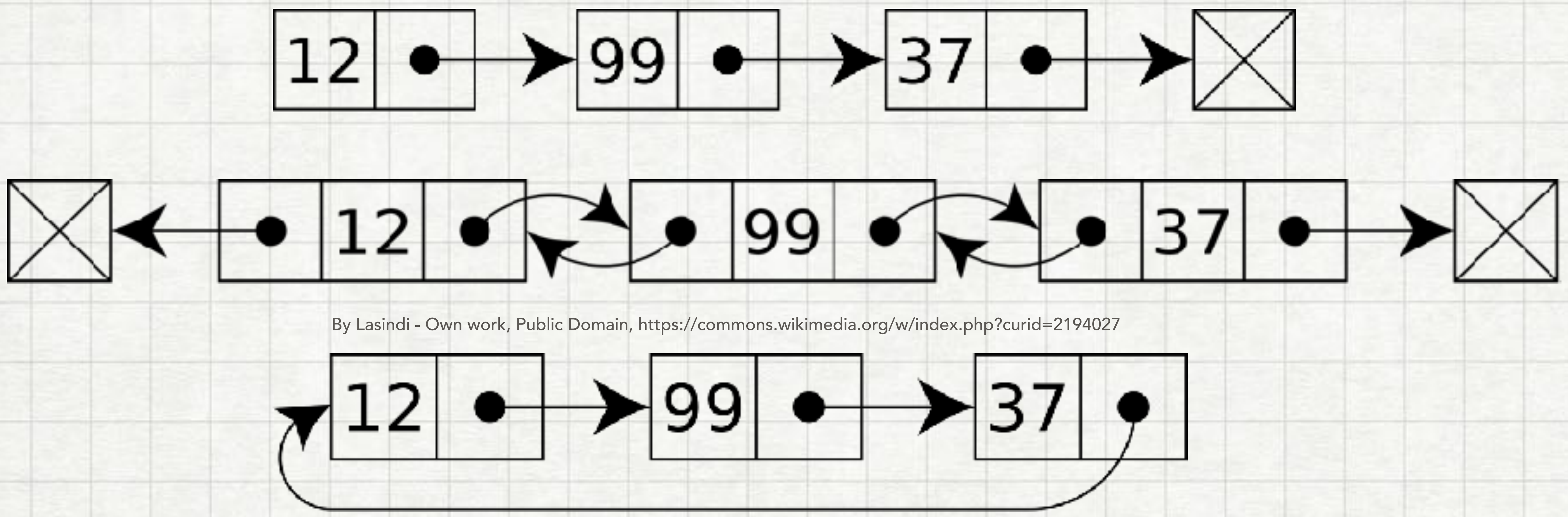
INTRODUCTION

- Data structures and algorithms (EDAA01)
- Good to have at least:
  - Some computer organization and architecture (e.g. EITF20)
  - Some concurrent or real-time programming (e.g. FRTN01, EDAP10)
  - Minimal networking/web programming (e.g. EDAF90)
  - Some programming experience (Java, even better: C)
  - Minimal contact with development tools (git, shell, make, grep, gdb, …)

DATA STRUCTURES IN OS

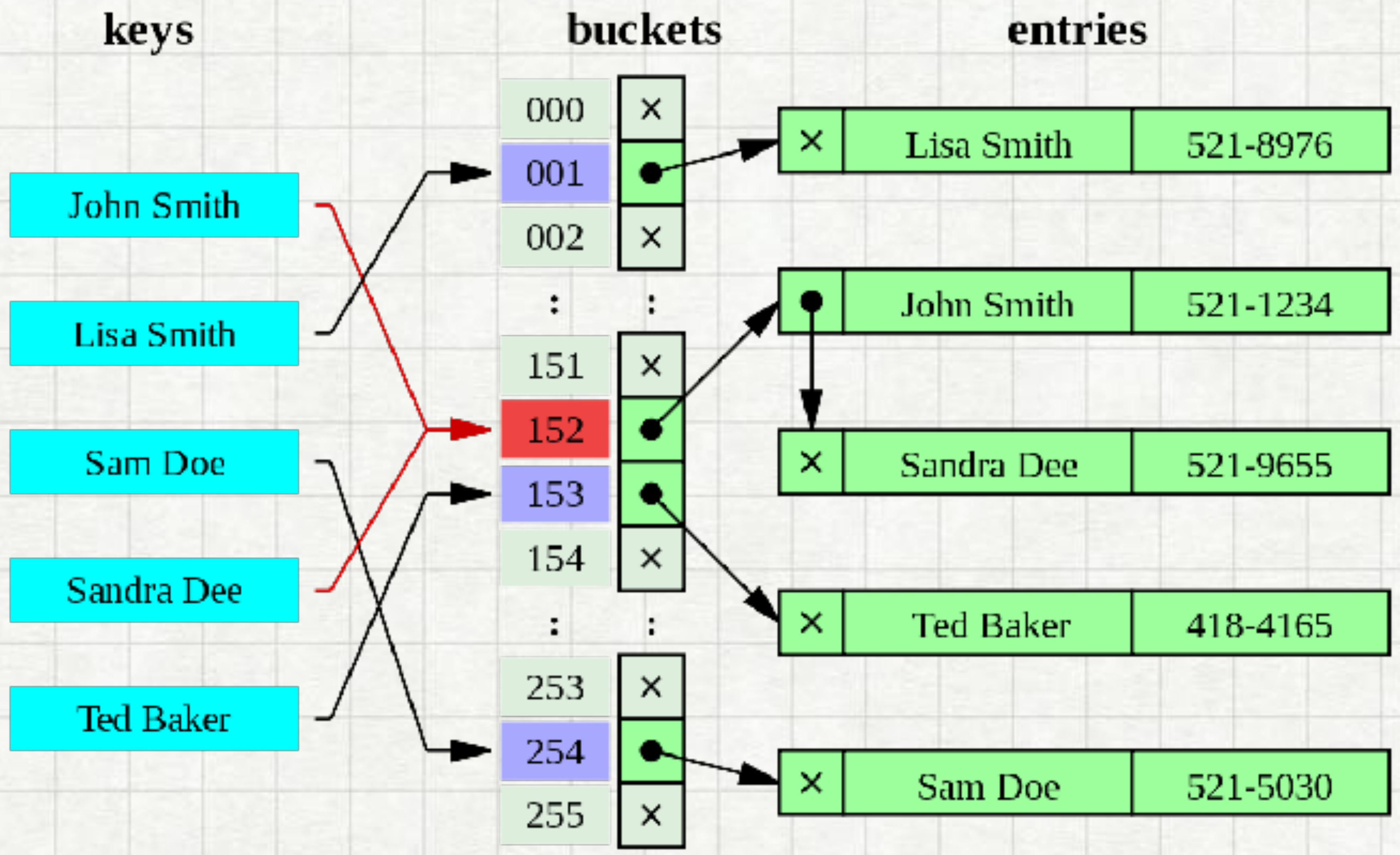
QUICK REMINDER

Lists



By Lasindi - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=2194027

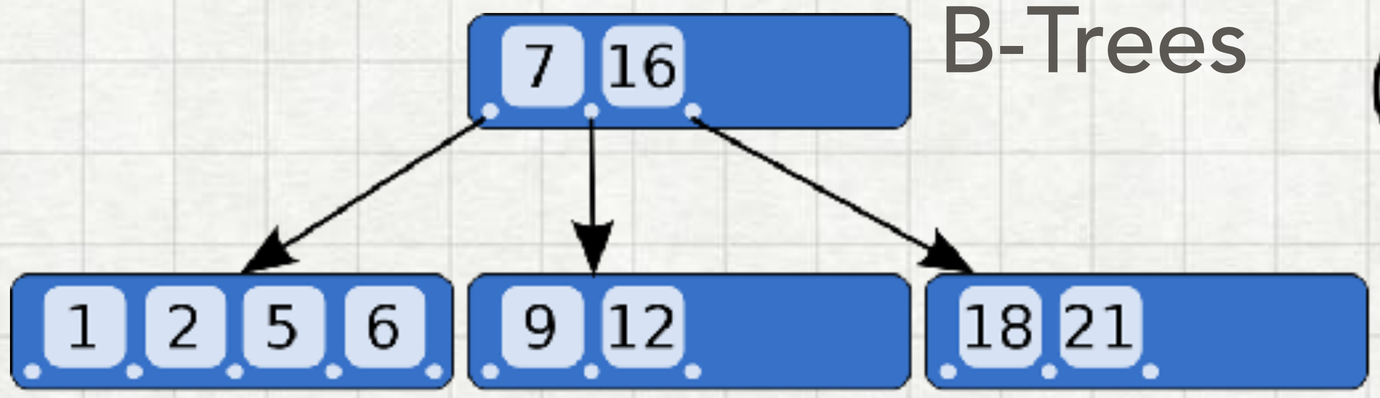
Hash Maps



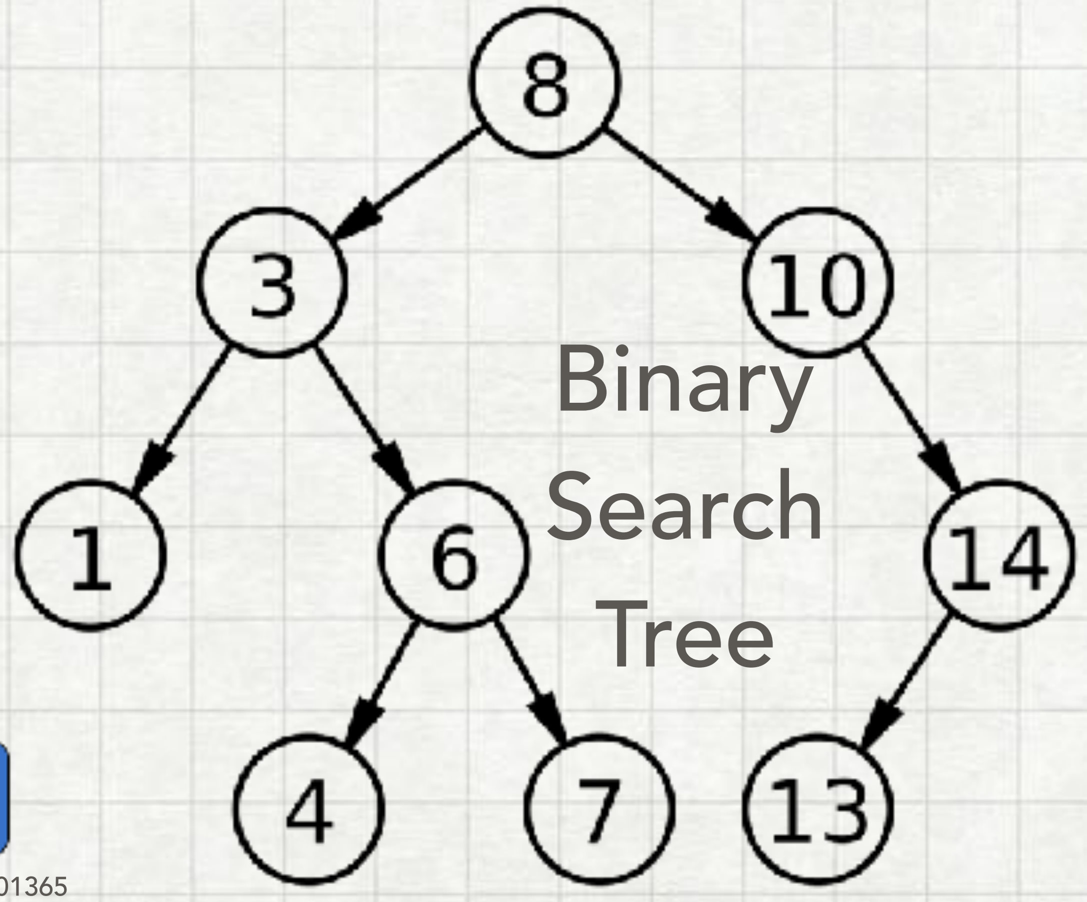
By Jorge Stolfi - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6471915

Trees

B-Trees



By CyHawk - Own work based on [1]., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=11701365

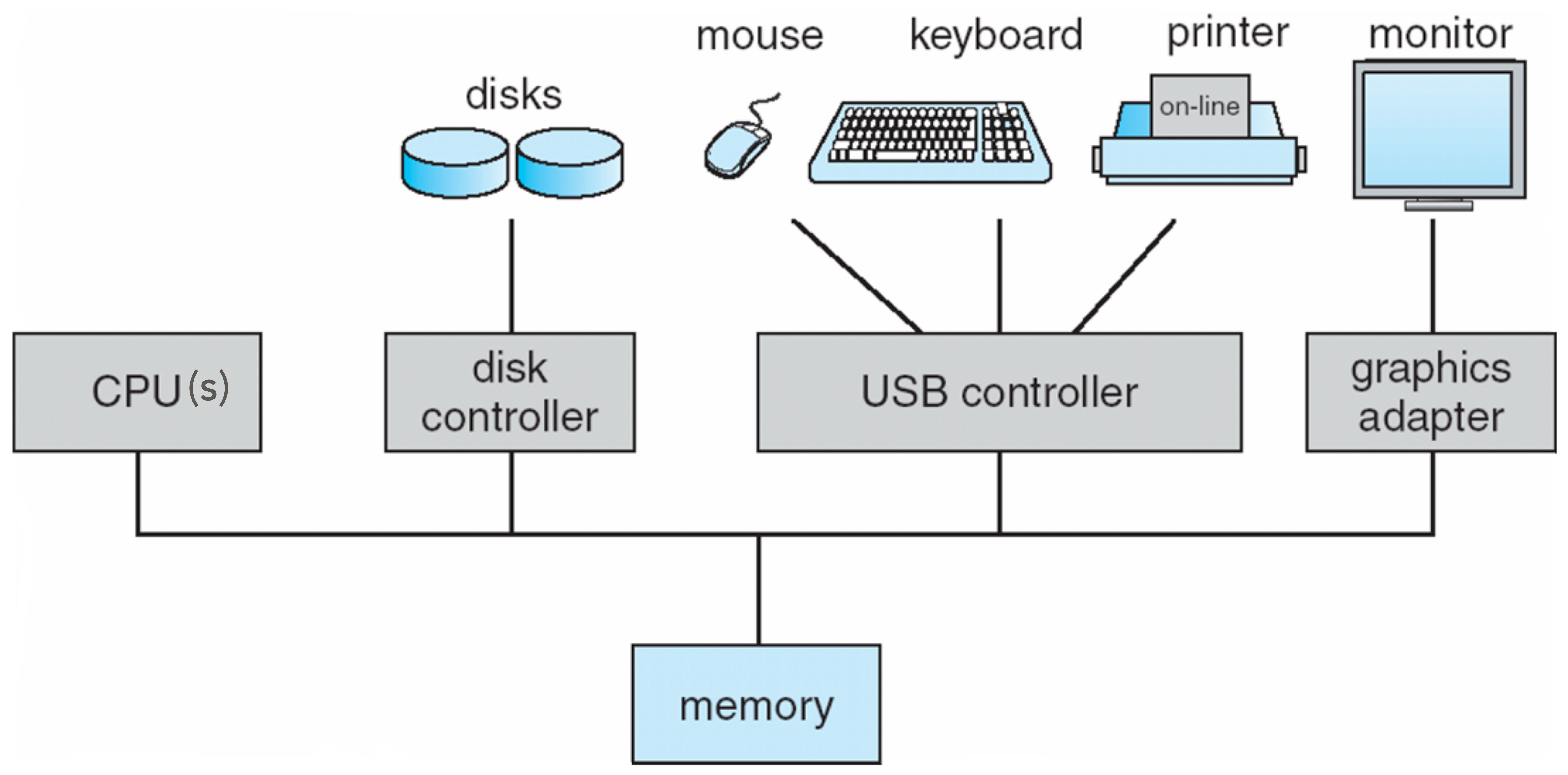


Binary Search Tree

Bitmaps

COMPUTER SYSTEM ARCHITECTURE

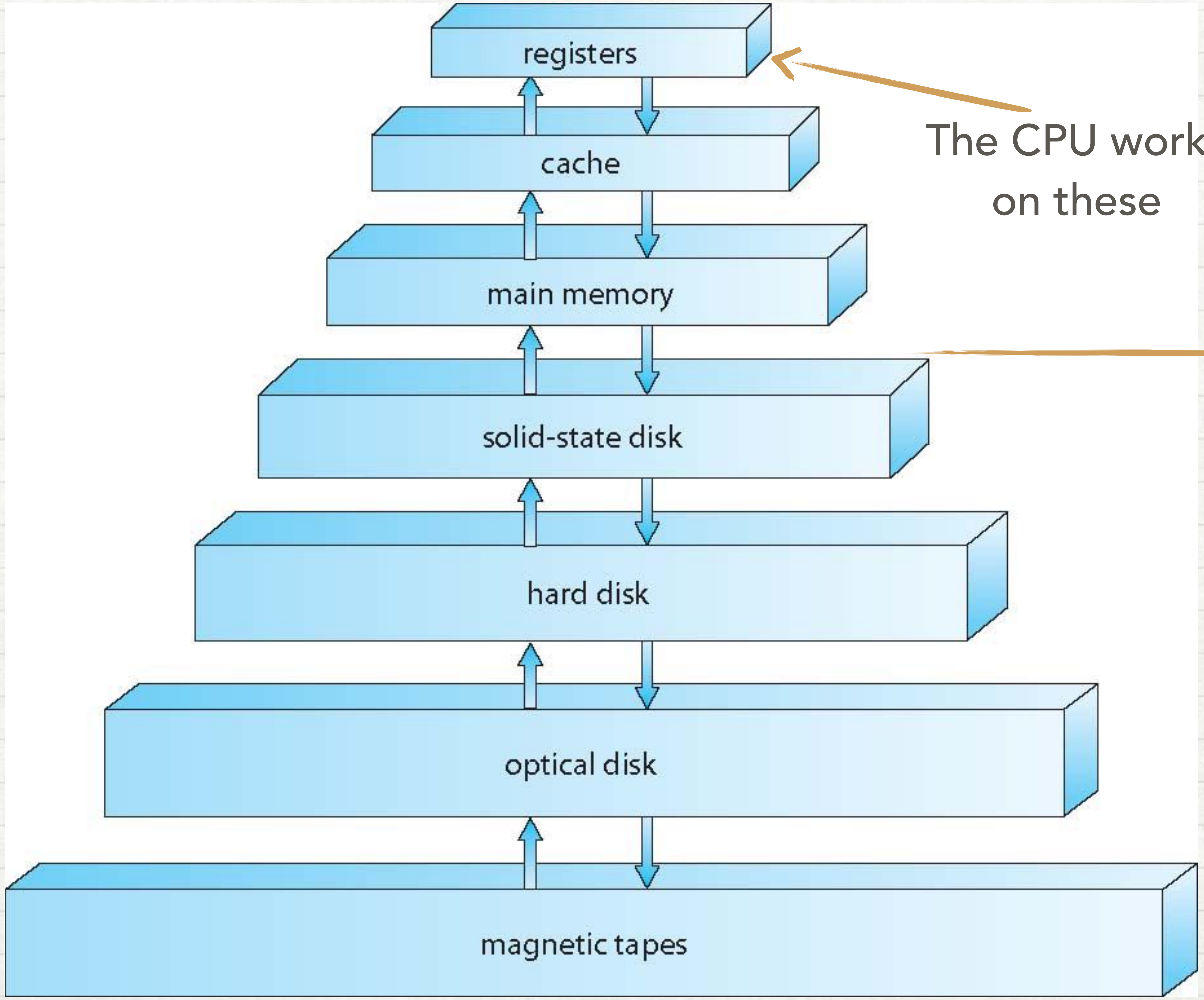
QUICK REMINDER



STORAGE DEVICE HIERARCHY

QUICK REMINDER

Fast, but Small (capacity)!



Large, but Slow!

In or close to the CPU

The CPU works on these

Volatile

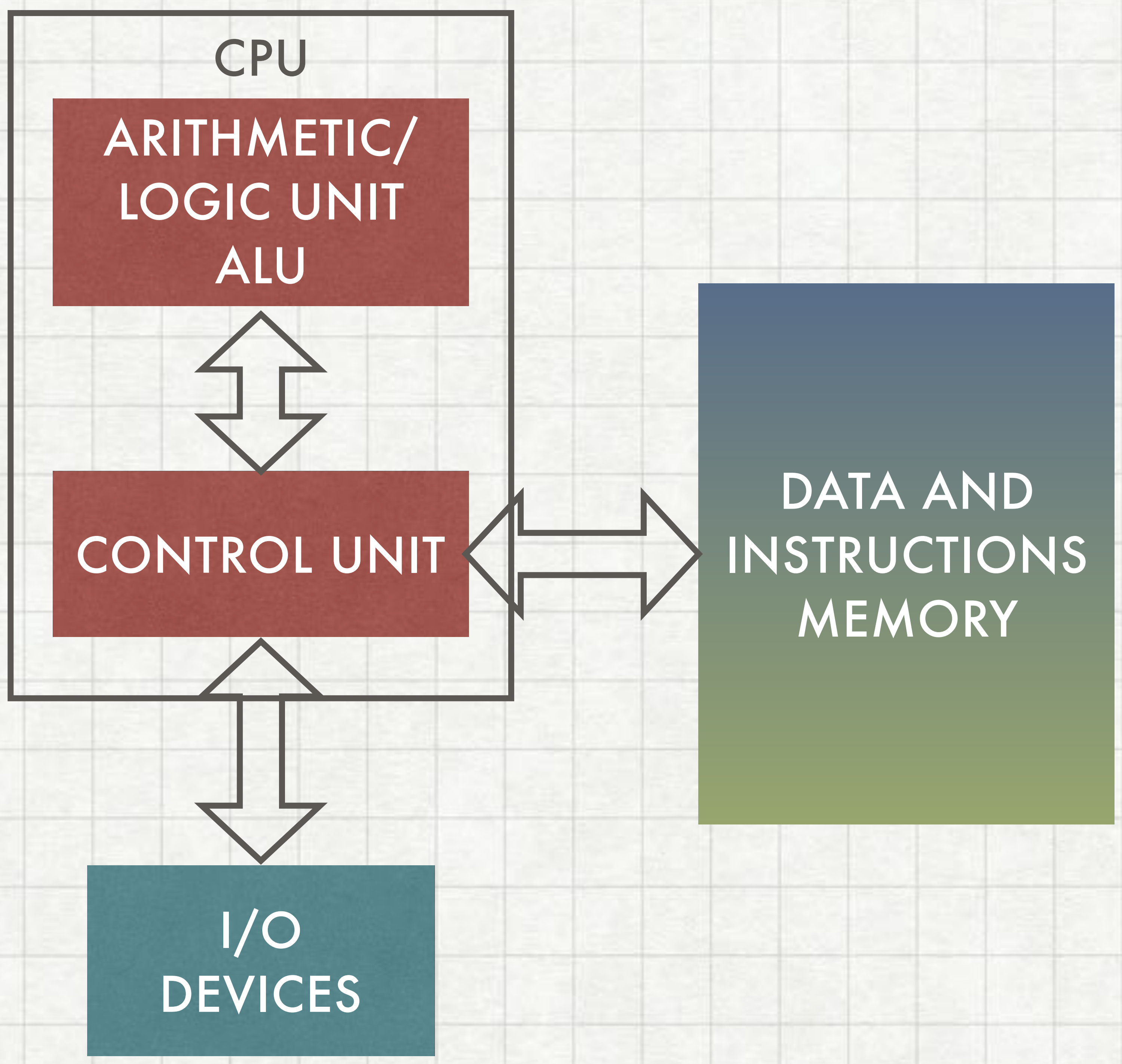
Persistent

Far away from the CPU

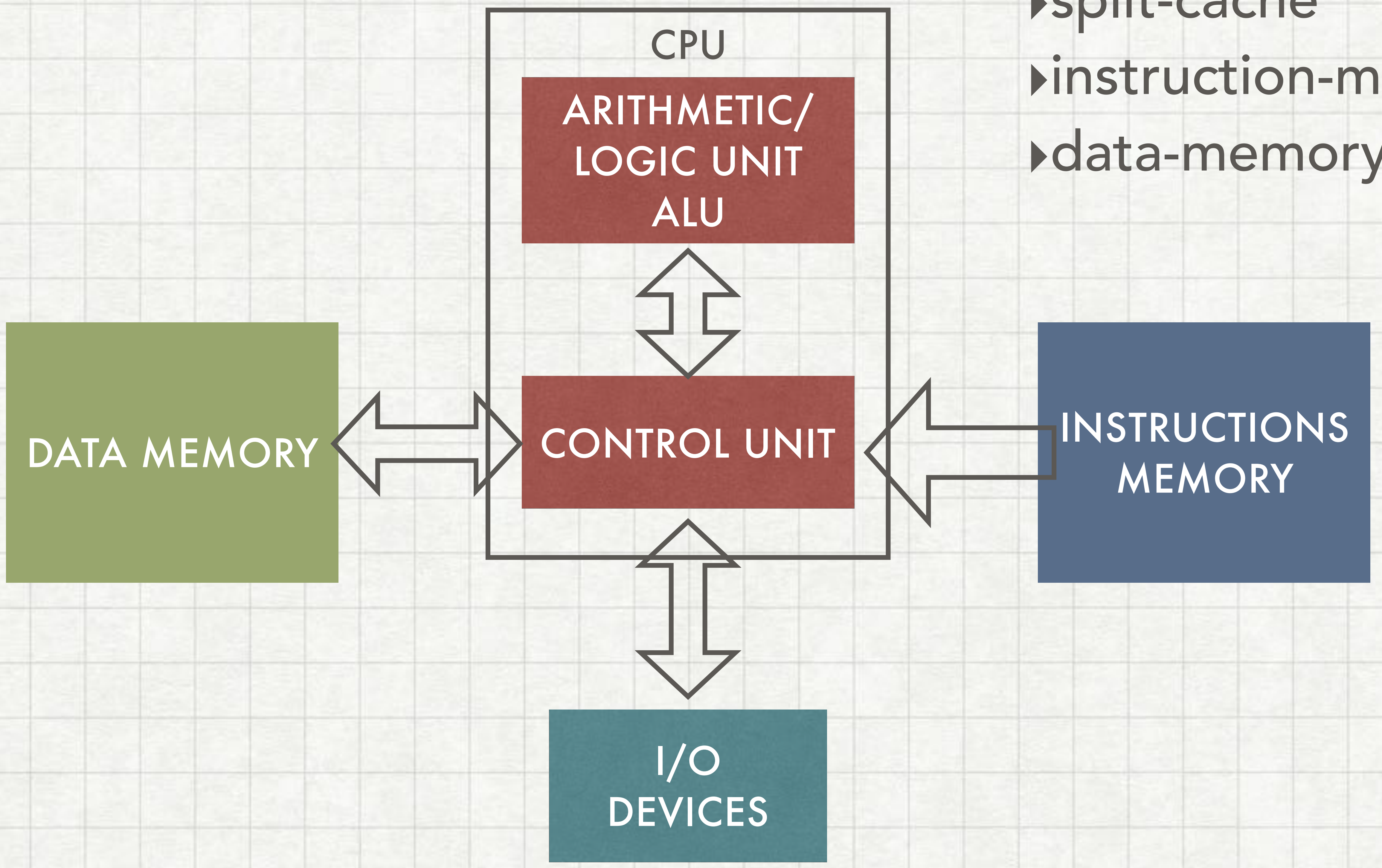
COMPUTER ARCHITECTURES

QUICK REMINDER

von Neumann



Harvard

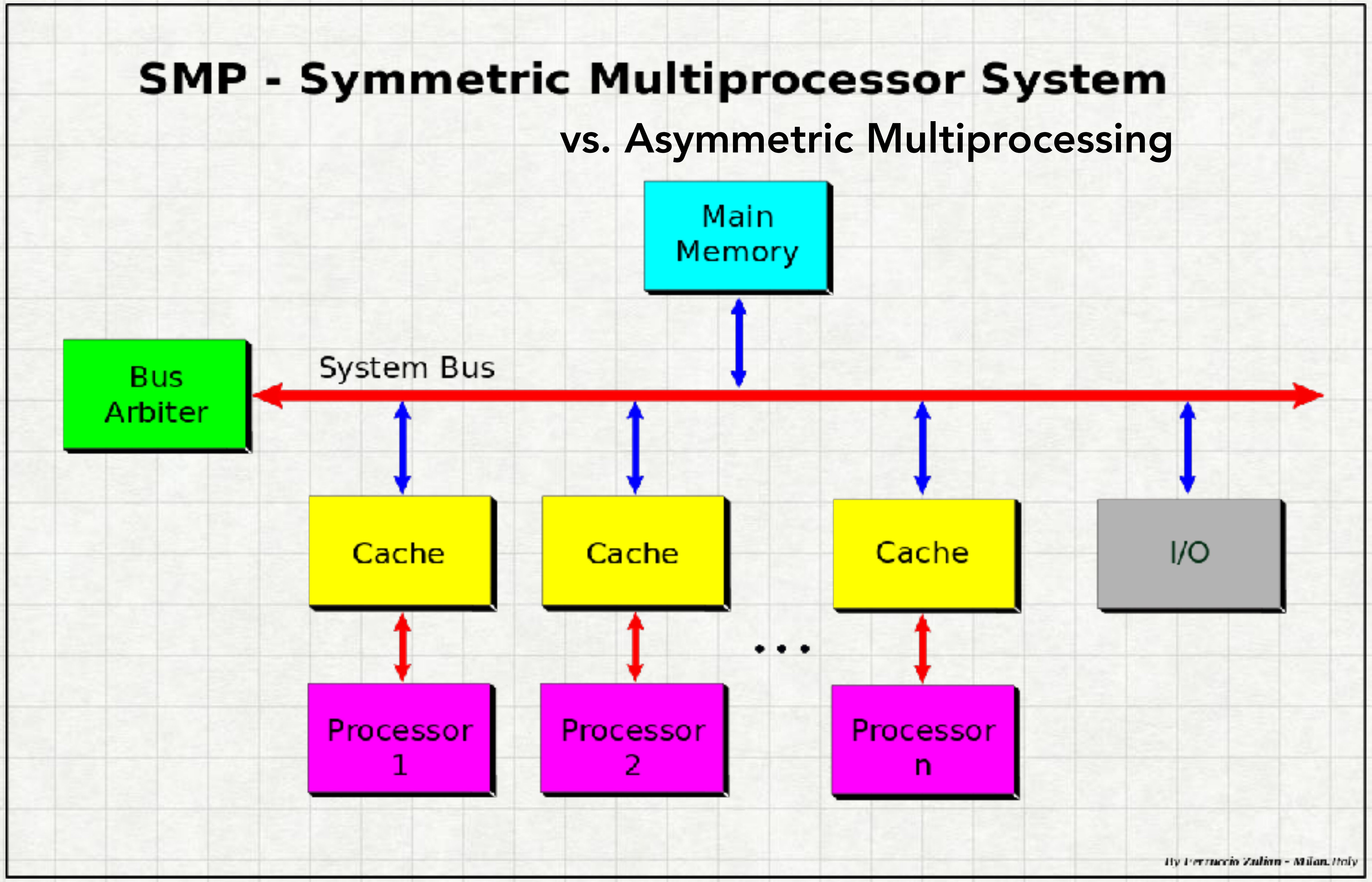


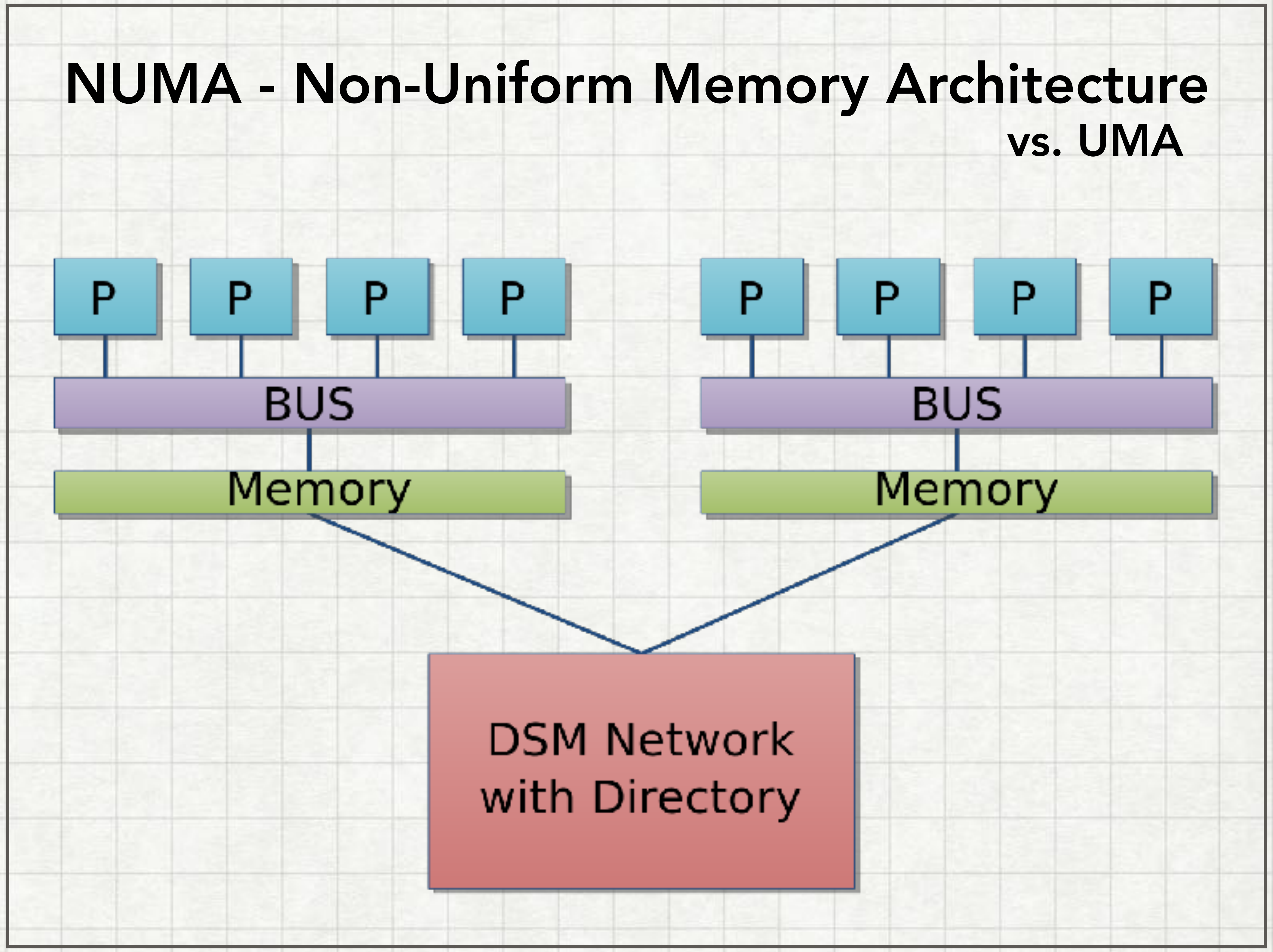
modified Harvard:

- split-cache
- instruction-memory-as-data
- data-memory-as-instruction

BEYOND SINGLE PROCESSOR SYSTEMS

QUICK REMINDER





COURSE ORGANIZATION

COURSE OVERVIEW

ORGANIZATION

ADMIN

- Staff (Dept. of Computer Science):
  - Lecturer: **Flavius Gruian** (Associate Professor of Embedded Systems); Office E:2125b
  - Lab assistant: **Alexandru Dura** (PhD student)
- Books
  - Silberschatz, Galvin & Gagne, "Operating Systems Concepts" 9th ed. - the dinosaur book
  - (adv.) R. Love, "Linux Kernel Development" 3rd ed.
- Website: http://cs.lth.se/edaf35

LECTURES OVERVIEW

ORGANIZATION

ADMIN

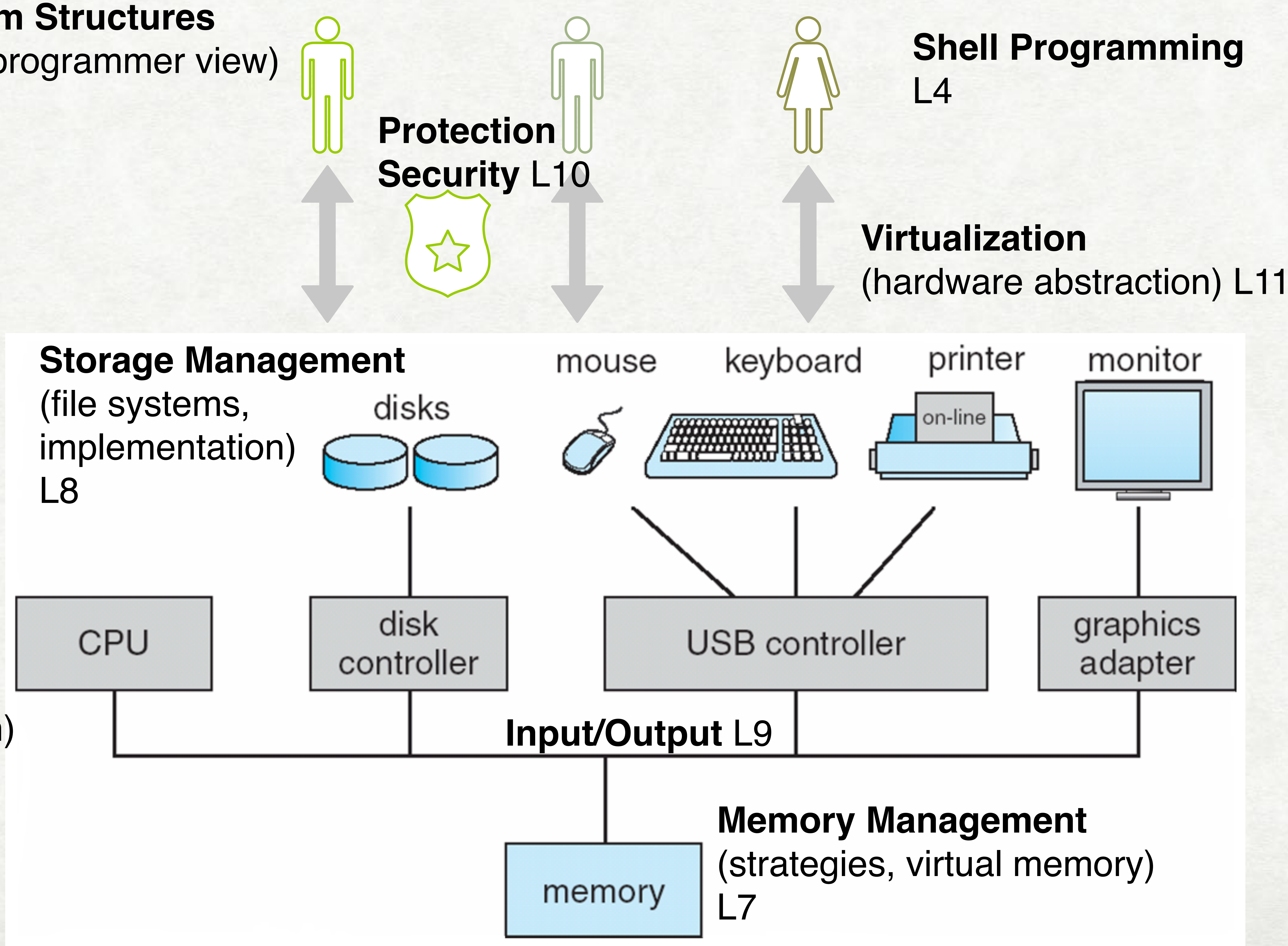
- 11 content (1 guest) + 1 preparatory for exam
- 2/week up to L8, 1/week after that
- ~~Location: E:2116 (consult the "time edit" schedule for changes)~~
- Much as self-study: watch pre-recorded video lectures + 1h Q/A
- Covering mainly the dinosaur book chapters (overviews)
- Consult the course web page for details

ZOOM!
CHECK THE COURSE PAGE

LECTURES OVERVIEW

ADMIN

System Structures
(user/programmer view)
L1



Working in C
(pointers, memory model,
tools, execution)
L2

Process Management
(execution, communication,
scheduling, synchronization)
L3, L5, L6

Exam preparation
L12

LABORATORY ASSIGNMENTS

ORGANIZATION

ADMIN

- developing (parts of) PintOS, educational OS, Stanford, 2009
- 1.5 credit points
- 4 (+1 preparatory) assignments, in C, building upon each other
- work in pairs… or not
- ~~in the Linux labs (University) E:Hacke,~~ or
  on your own machine (Linux or Docker container)
- descriptions and more info on the course web page

PROJECT ASSIGNMENT

ORGANIZATION

ADMIN

- further development of your PintOS version
- 3 credit points
- work on your own time
- submit/present when you are done (not necessarily before the exam)

EXAMINATION
ORGANIZATION

ADMIN

- 3 credit points
- time-constrained home-assignment (very likely using Canvas)
- dedicated lecture for preparation (L12)
- some previous exams available
- ...more on the course page

EDAF35 MODULE 1

CONTENTS

- Introduction:
  Motivation, OS Roles, Course Aim, Prerequisites (Quick Recap)

- Organization:
  Lectures, Laboratories, Project, Examination, Support

- **Overview of an OS:**
  Views, Components, Functionality, Examples

(Material loosely based on the course book, Chapter 2)

OVERVIEW OF AN OS (CH2)

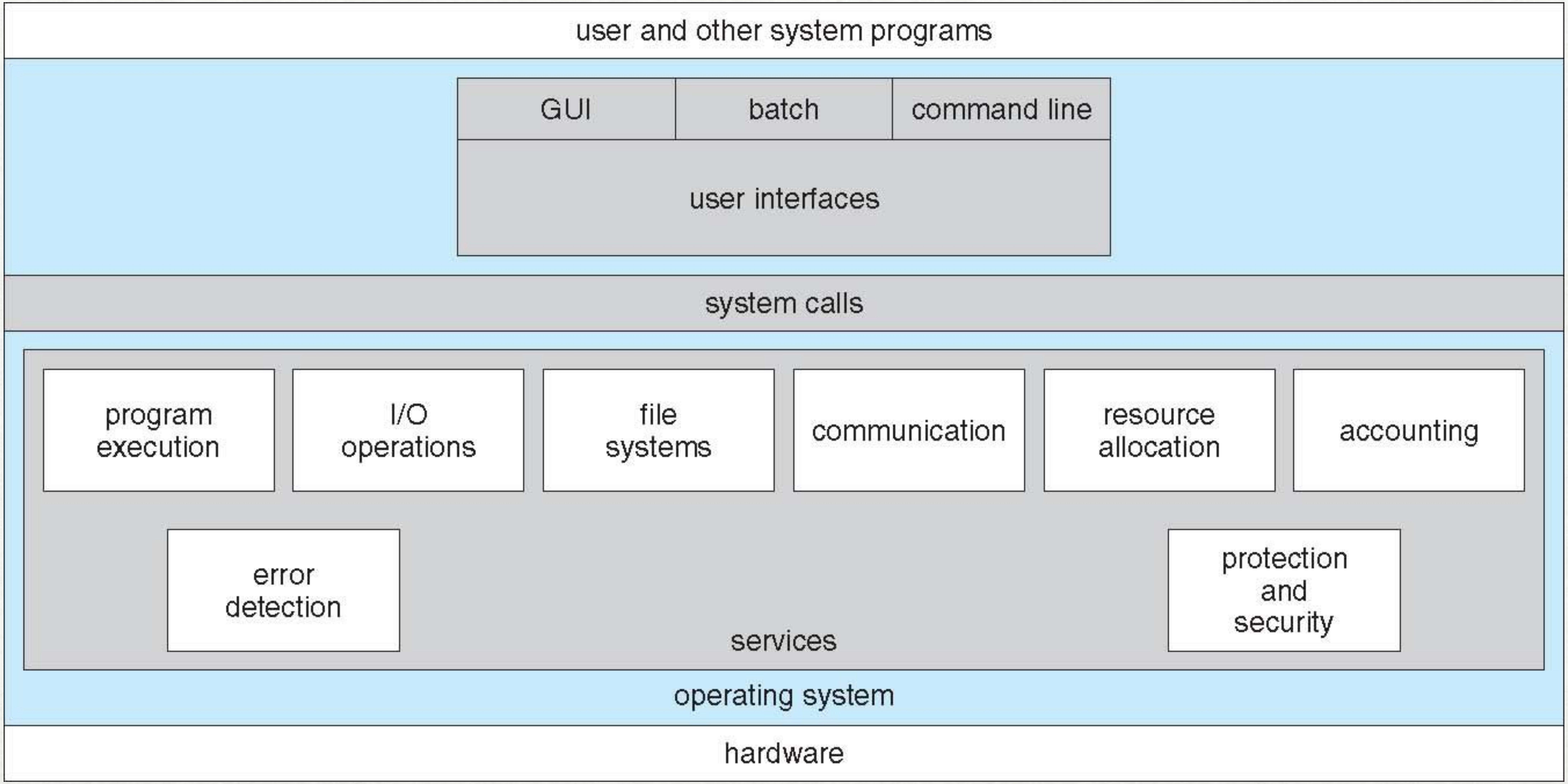
VIEWPOINTS

OVERVIEW OF AN OS

- what services are provided?
  - (**functionality**)
- how are these made available to the user?
  - (**interface**)
- what are the components and how are they interconnected?
  - (**structure**)

OS SERVICES

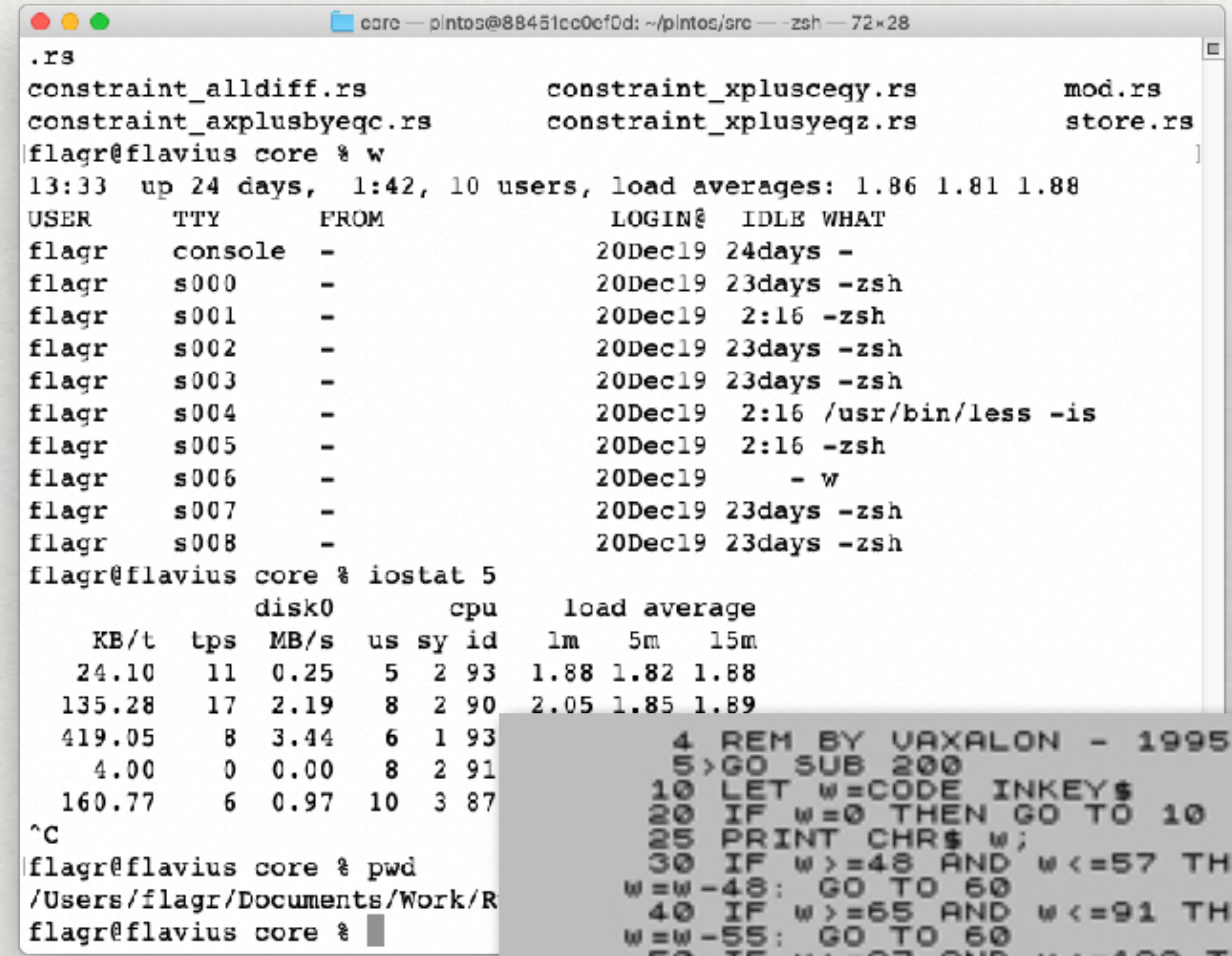
OVERVIEW OF AN OS



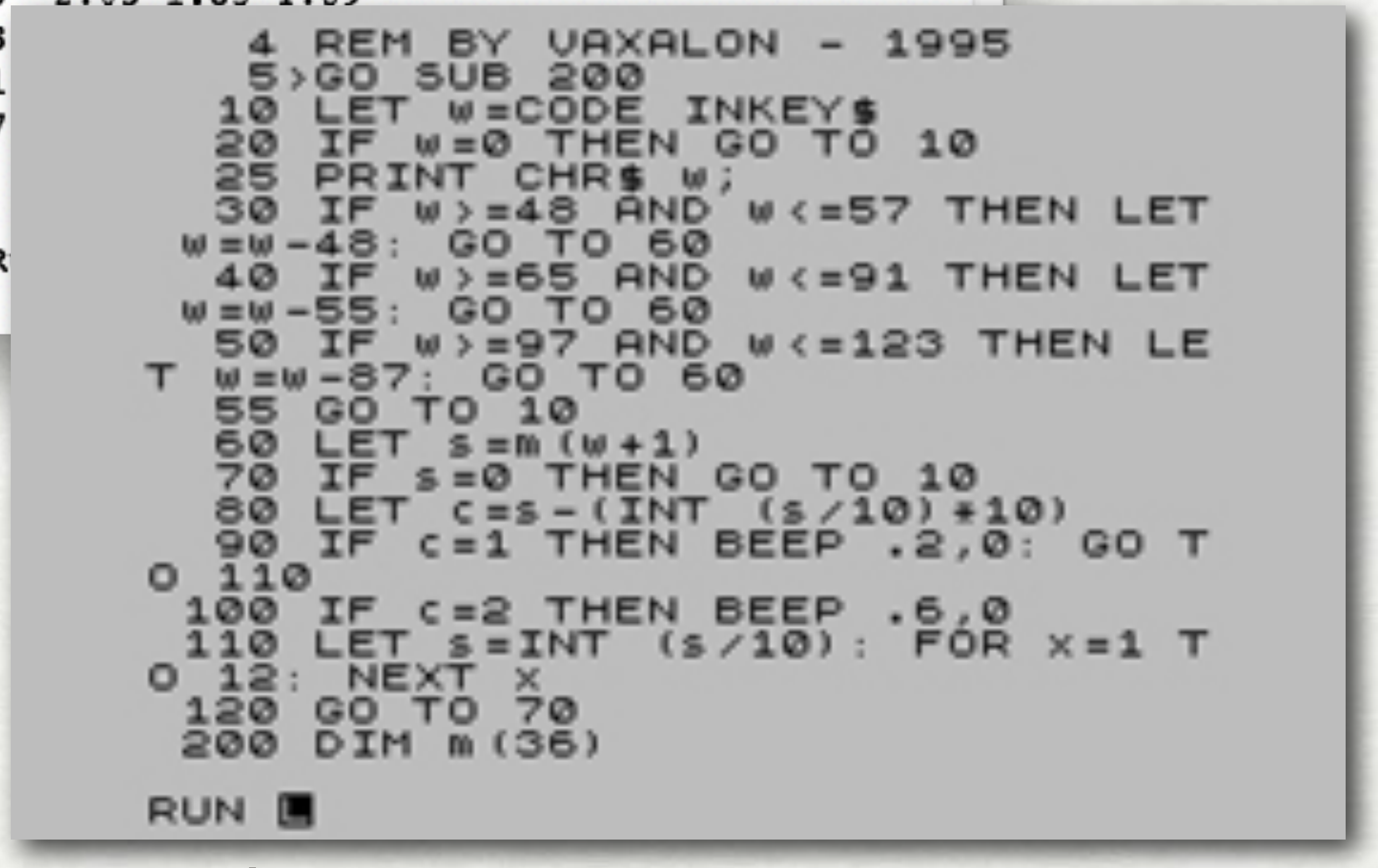
OS USER INTERFACE

OVERVIEW OF AN OS

Command Line Interface (CLI)



ZSH (Linux)



Sinclair ZX Spectrum BASIC (1995)

Graphical User Interface (GUI)

Mac OS X GUI

Touchscreen (iOS)

[When to choose which?]()

SYSTEM CALLS

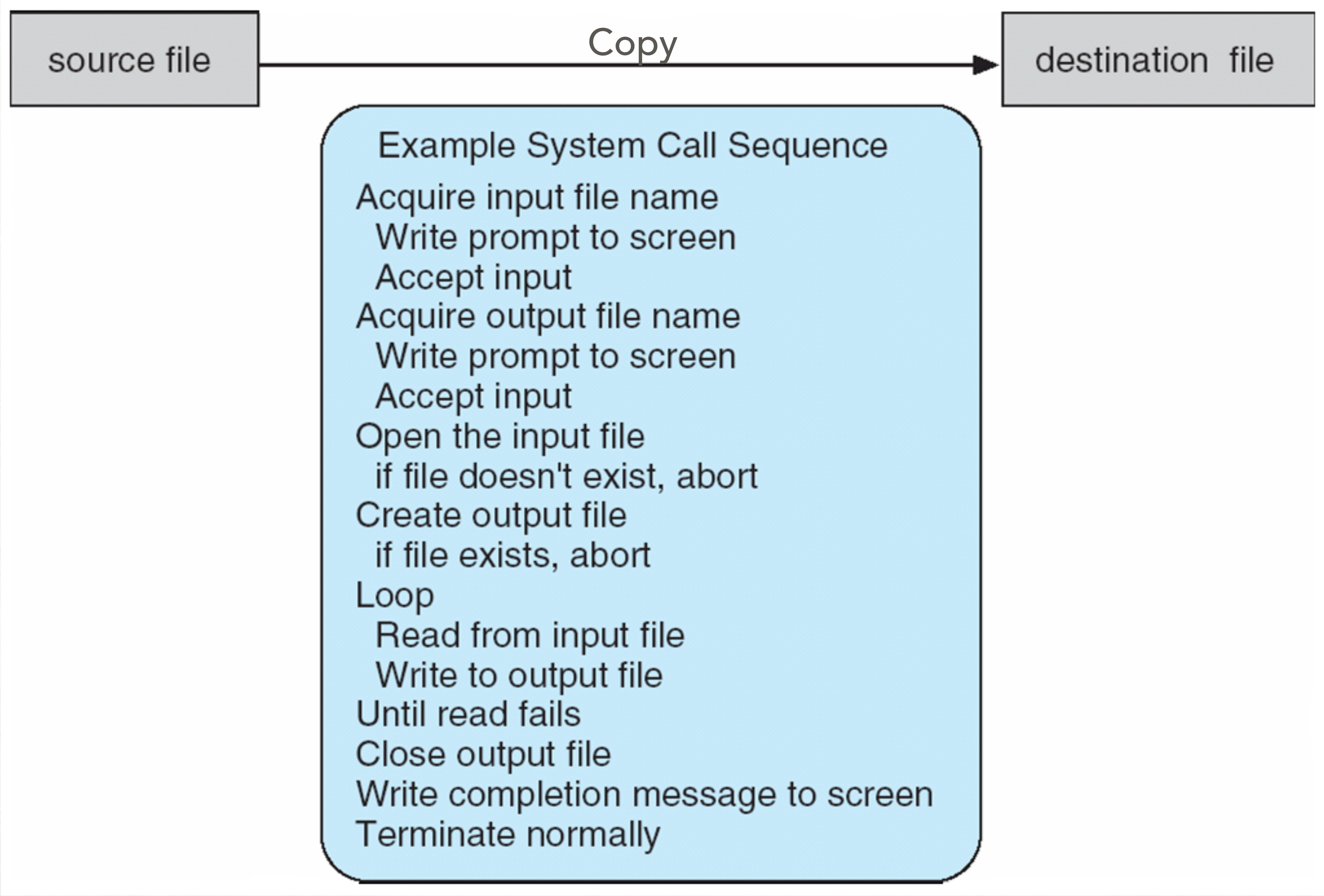
OVERVIEW OF AN OS

- programming interface to the OS services
- used via a high-level interface (API) rather than directly
  - (another layer of abstraction)
- examples: Win32 API (Windows), POSIX API (Unix/Linux), Java API (JVM)

(we'll use generic system-call names in the following examples)

SYSTEM CALLS, AN EXAMPLE

OVERVIEW OF AN OS



SYSTEM CALL STANDARD API EXAMPLE

OVERVIEW OF AN OS

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t      read(int fd, void *buf, size_t count)
└──────┘     └──┘ └──────────────────────────────┘
 return     function          parameters
 value        name
```

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

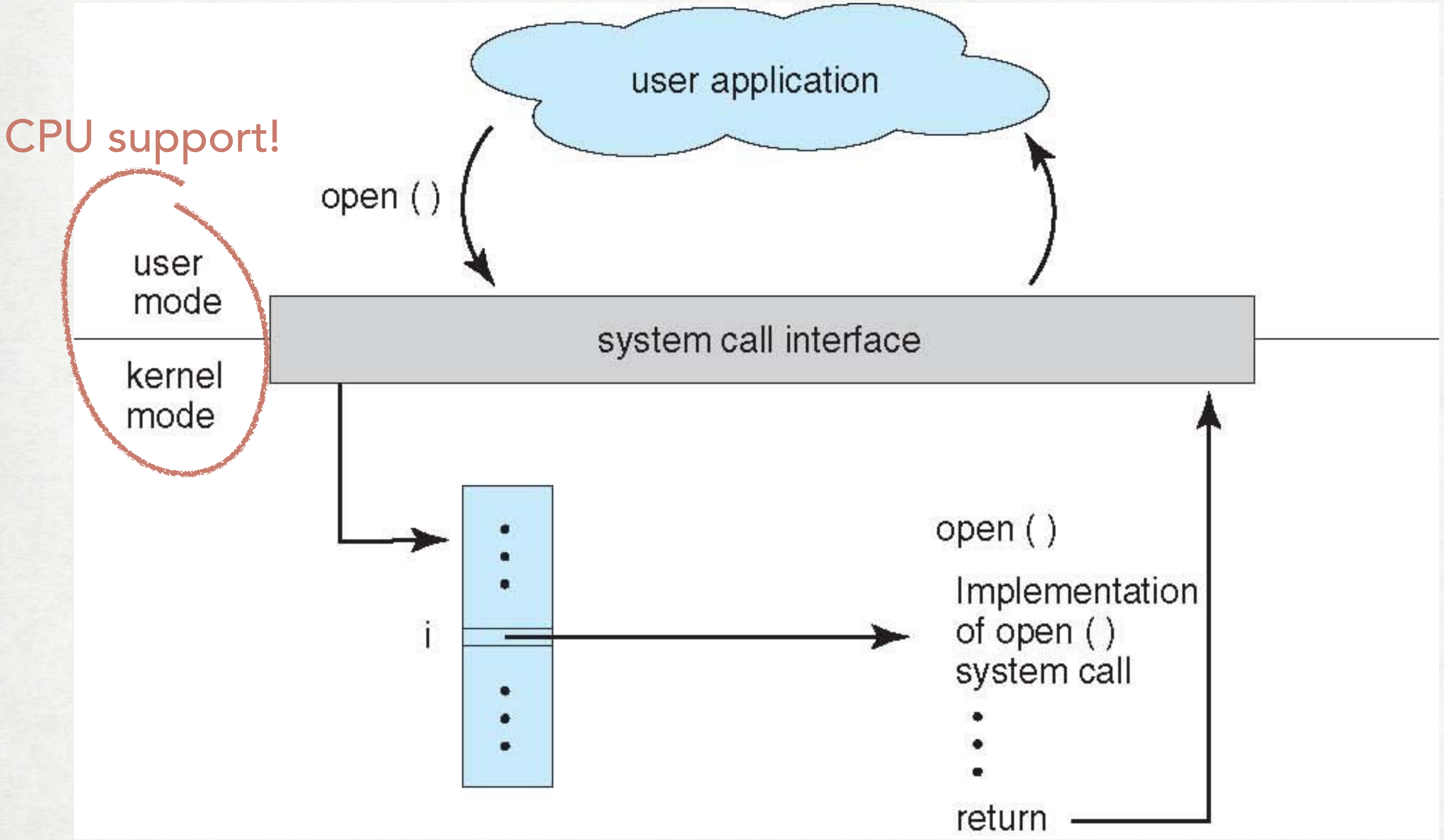
core — pintos@88451ec0ef0d: ~/pintos/src — less ◂ man -s 2 read — 84×34

```
READ(2)                  BSD System Calls Manual                  READ(2)

NAME
     pread, read, readv -- read input

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <sys/types.h>
     #include <sys/uio.h>
     #include <unistd.h>

     ssize_t
     pread(int d, void *buf, size_t nbyte, off_t offset);

     ssize_t
     read(int fildes, void *buf, size_t nbyte);

     ssize_t
     readv(int d, const struct iovec *iov, int iovcnt);

DESCRIPTION
     read() attempts to read nbyte bytes of data from the object referenced
     by the descriptor fildes into the buffer pointed to by buf.  readv()
     performs the same action, but scatters the input data into the iovcnt
     buffers specified by the members of the iov array: iov[0], iov[1],
     ..., iov[iovcnt-1].  pread() performs the same function, but reads
     from the specified position in the file without modifying the file
     pointer.

     For readv(), the iovec structure is defined as:
:
```

man -s 2 read

HANDLING A SYSTEM CALL

OVERVIEW OF AN OS



- one entry point, call identified by number (index in a table of addresses)
- parameters passed:
  - in registers
  - via memory (specific addr)
  - pushed onto the stack

(advantages/drawbacks?)

TYPES OF SYSTEM CALLS

OVERVIEW OF AN OS

- six classes, managing:

1. processes
2. files
3. devices
4. system information
5. communication
6. protection/security

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

SYSTEM PROGRAMS

OVERVIEW OF AN OS

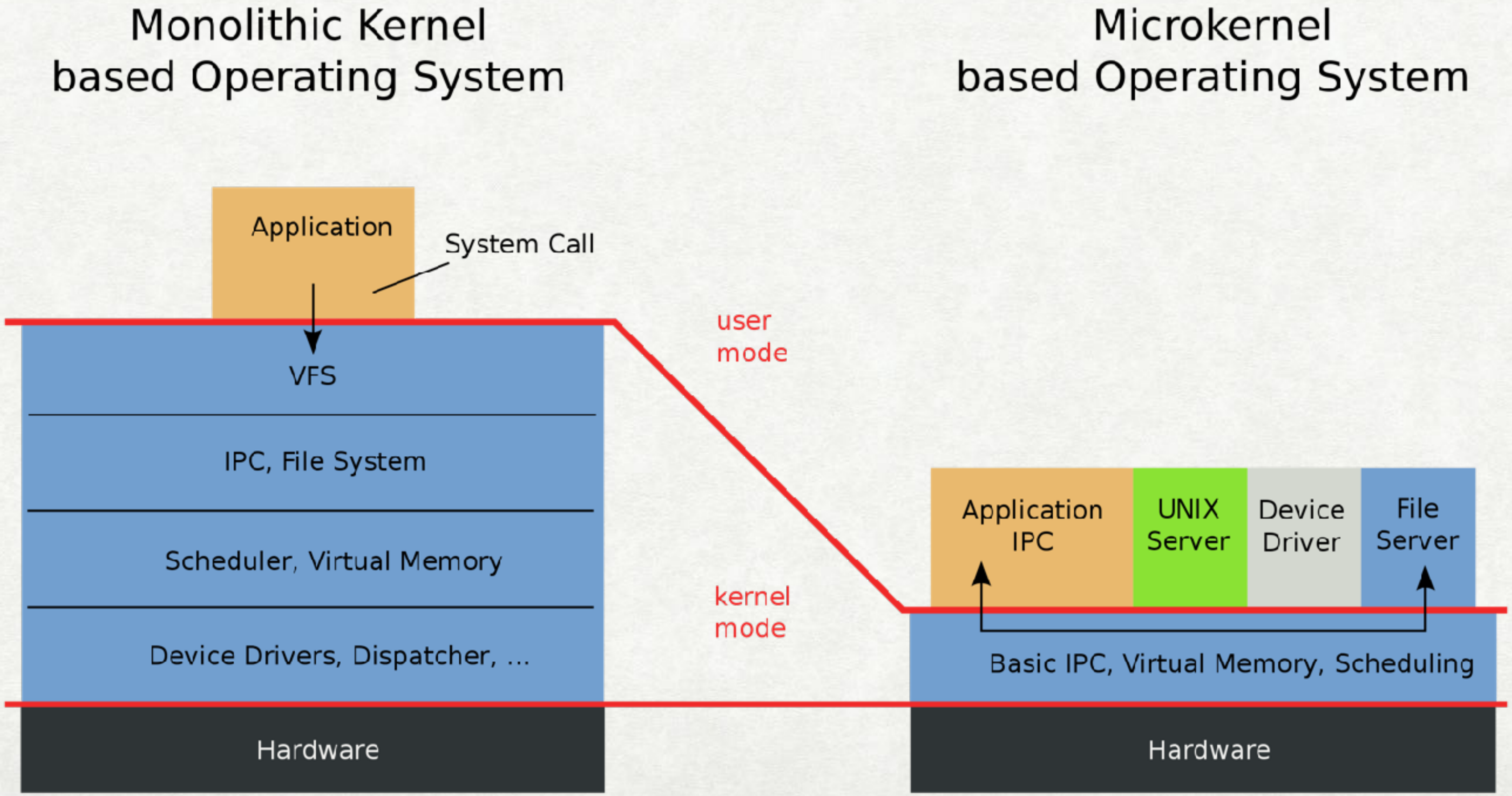
Convenient environment for program development and execution

- **file manipulation**: *create, delete, list, copy, locate, print,…*
- **status information**: *date, time, available resources, logging, debugging, registry,…*
- **file modification**: *edit, search, transform,…*
- **programming support**: *compiler, assembler, linker, emulator, interpreter,…*
- **communication**: *email, instant messages, web browsers, remote desktops,…*
- **background services (daemons)**: *launch at boot, periodic or on demand facilities*

OS STRUCTURE

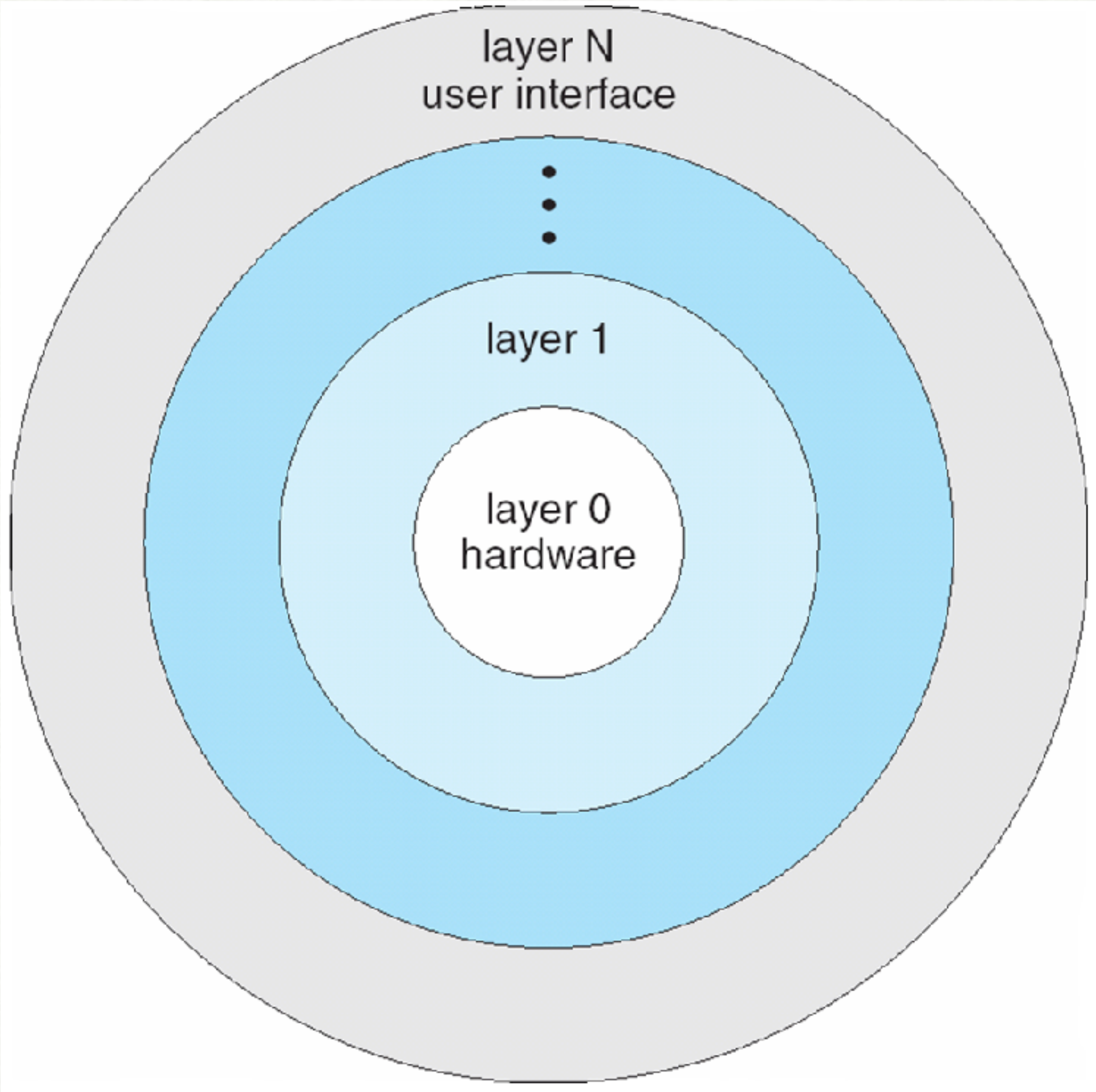
OVERVIEW OF AN OS

- Various ways of structuring, some very abstract — in reality: combinations



LAYERED APPROACH

OVERVIEW OF AN OS

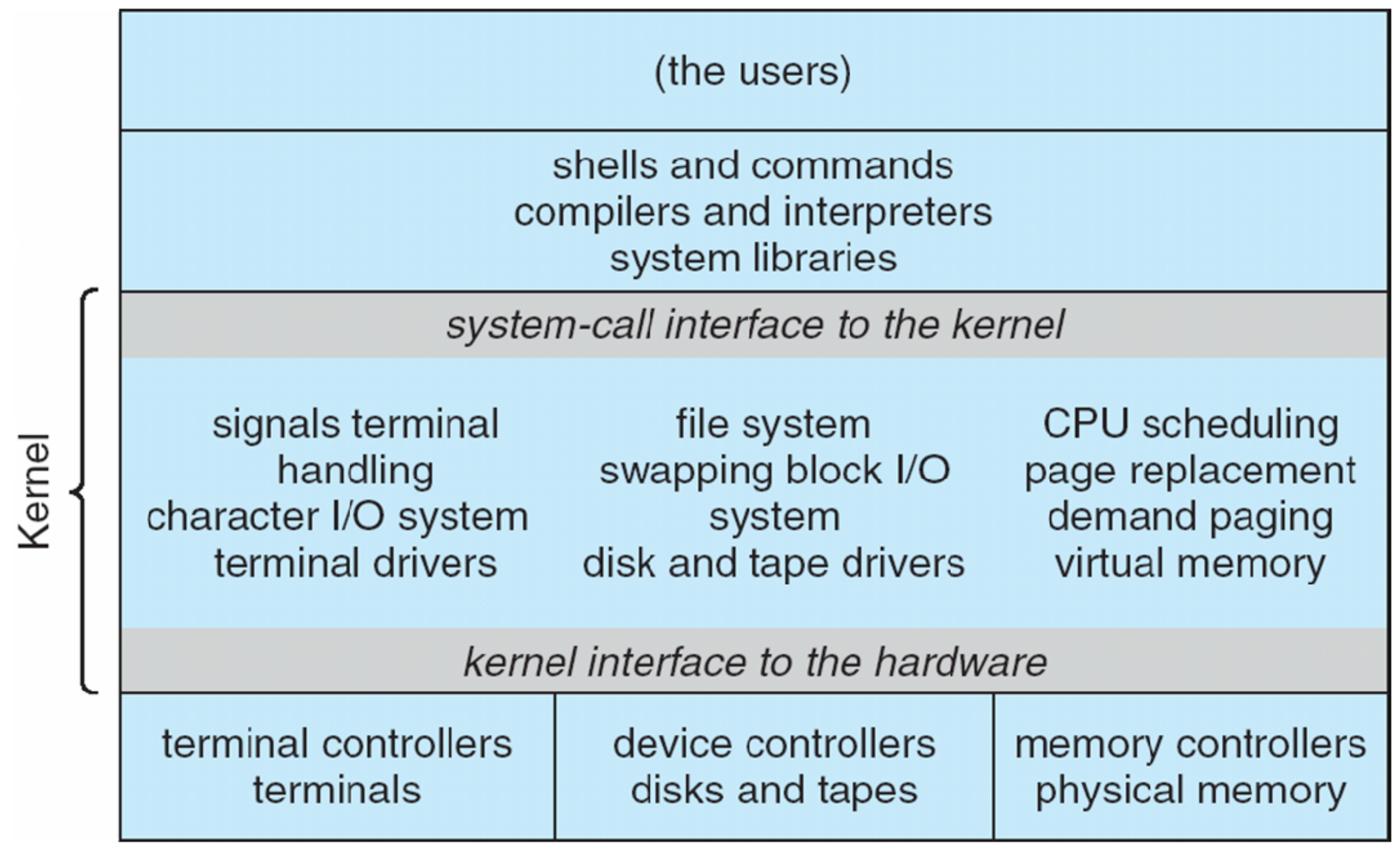


- pros: simple to construct and debug
- cons: overhead of calling through layers
- in practice: only few layers adopted in modern OS

OS STRUCTURE: TRADITIONAL UNIX

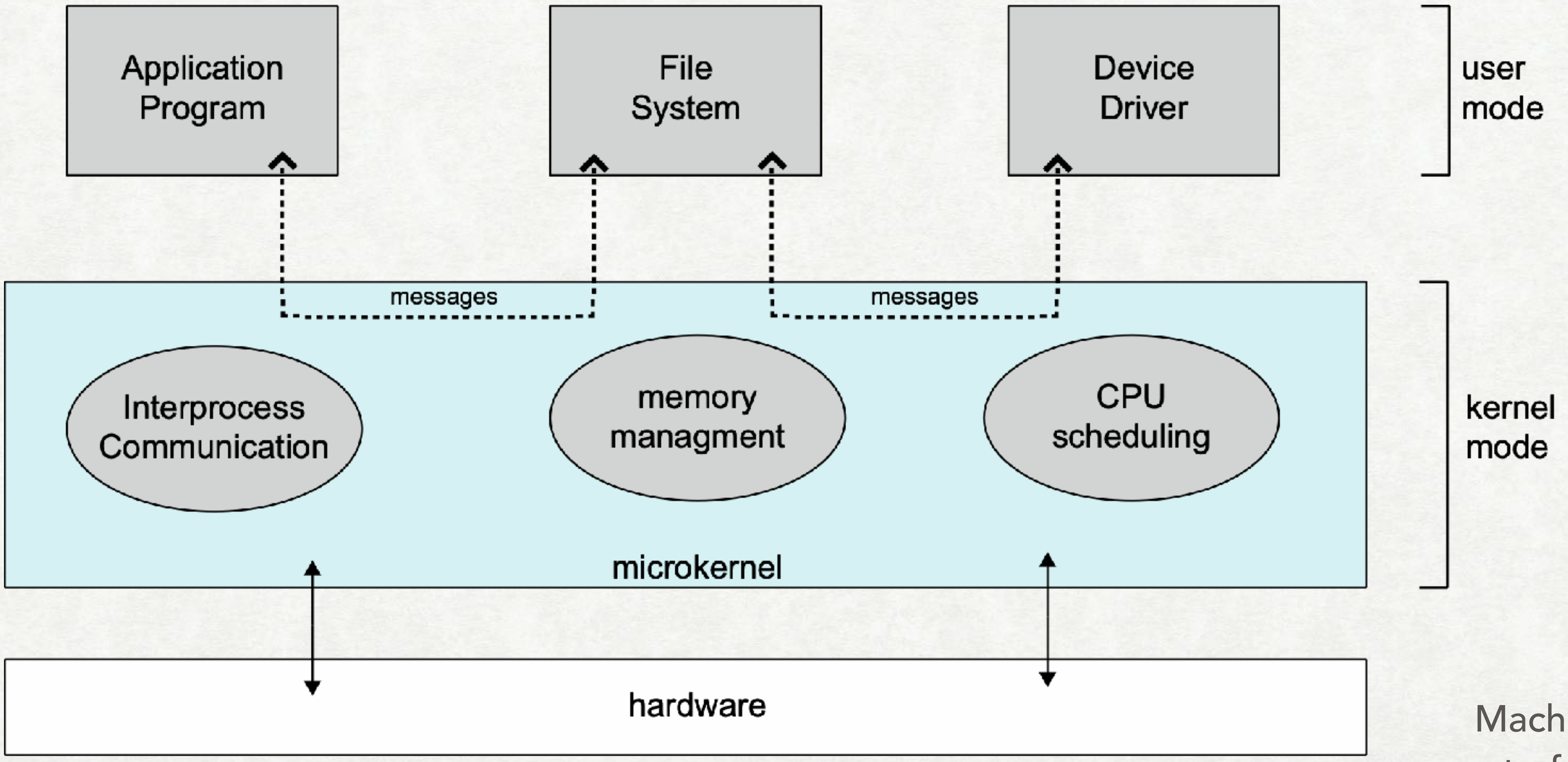
OVERVIEW OF AN OS

- Beyond simple, but not fully layered; monolithic



MICROKERNEL SYSTEM STRUCTURE

OVERVIEW OF AN OS

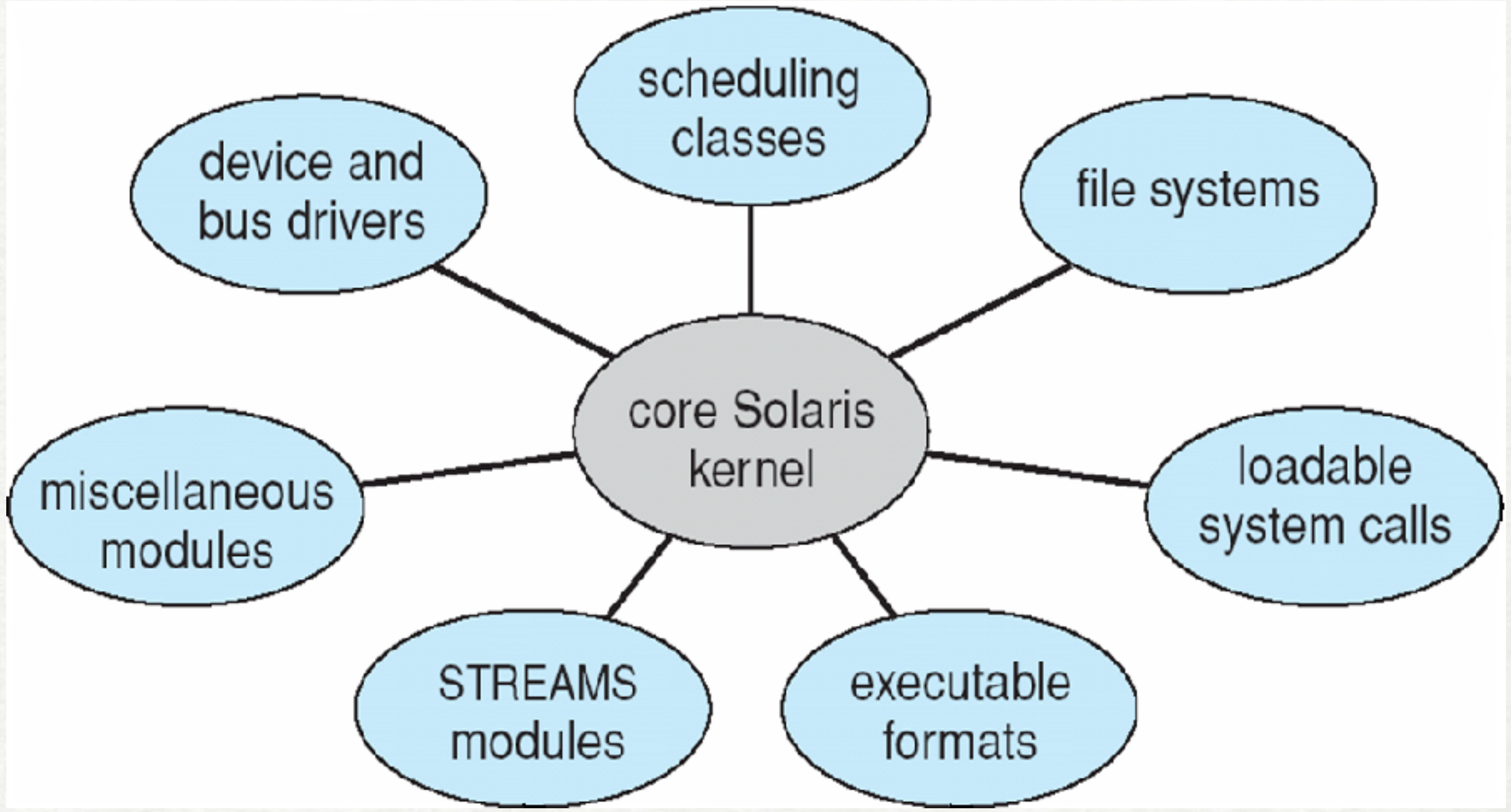


E.g.:
Mach, 1980s CMU…
part of Mac OS X today

MODULES

OVERVIEW OF AN OS

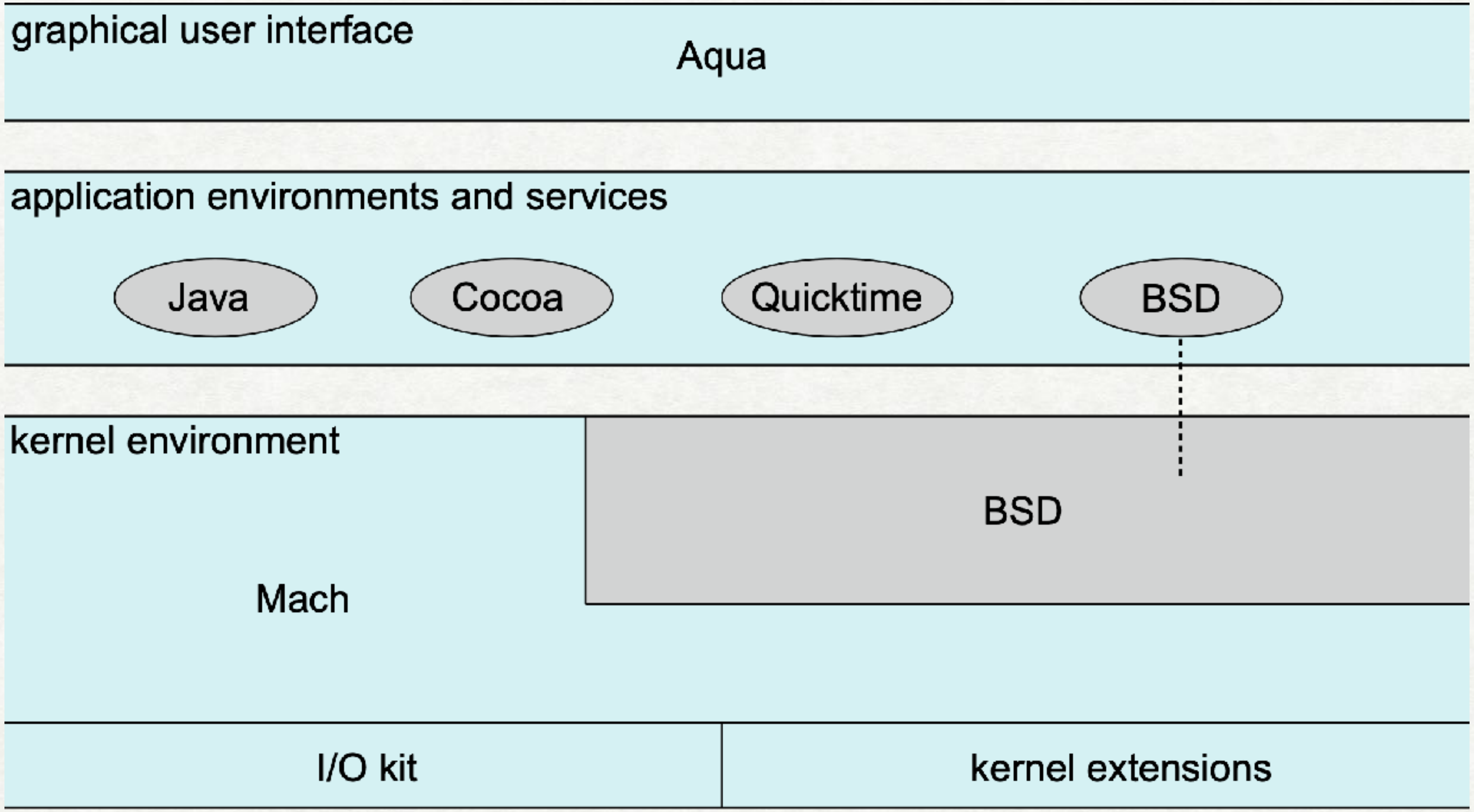
- modern OS: **loadable kernel modules**
  - OO approach, separate core components, known interfaces
  - each loads (in memory) as needed by the kernel



advantages vs. drawbacks?

HYBRID STRUCTURE: MAC OS X

OVERVIEW OF AN OS



Mach microkernel, layers, loadable modules

OS DESIGN AND IMPLEMENTATION

- stakeholders with (often) different goals:
  users, application programmers, OS developers, sys admins
- affected by hardware and overall system purpose
- widely different internal solutions to similar problems
- no universal "best solution" — only successful (and copied) approaches
  — some in this course
- separate **policy** (what) from **mechanism** (how)

OS DESIGN AND IMPLEMENTATION (CONT'D)

OVERVIEW OF AN OS

- language choice:
  - Early OSes — fully written in assembly, C/Algol/shell scripts (system programs)
  - Modern OSes — mix of languages — ASM (low level functionality), C/Rust (main body), C/C++/PERL/Python/shell script (system programs)
  - portability vs. performance tradeoff
- correctness: tested vs. formally verified (seL4)
- emulation: run (trace, debug) on non-native hardware (QEMU)

DEBUGGING

OVERVIEW OF AN OS

- more complex, due to user/kernel modes: *logs, dumps, profiles, emulators,…*
- OS usually generate **log-files** with error information
- on failure
  - applications: **core dump** — file with application memory contents
  - OS: **crash dump** — file with kernel memory contents
- performance tuning: trace listings, profiling (e.g. DTrace)

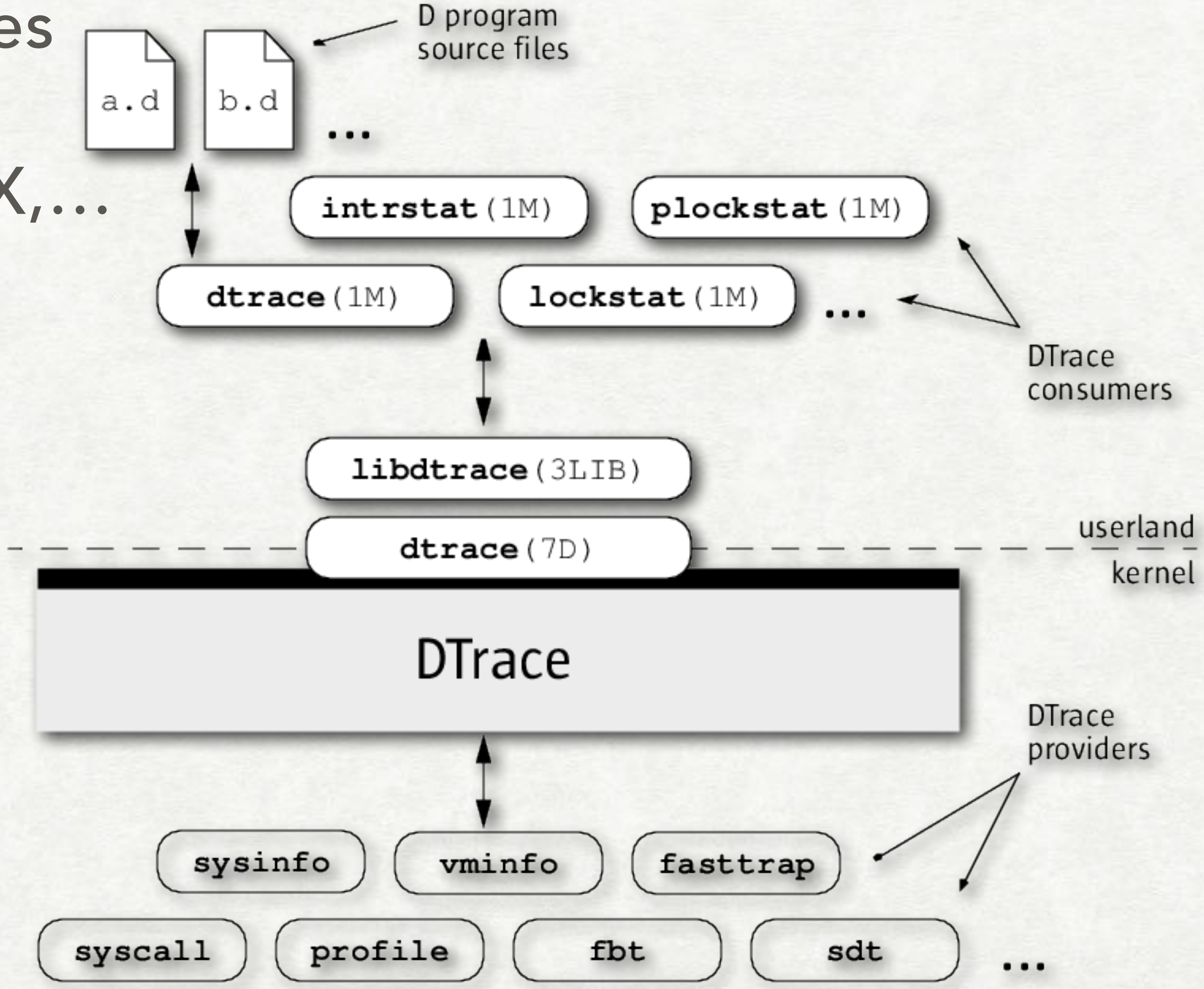
HAVE A LOOK AT THIS!

Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

PERFORMANCE TUNING: DTRACE

OVERVIEW OF AN OS

- DTrace — live instrumentation of user and kernel processes
- Solaris, FreeBSD, Mac OS X,…
- D programming language, scripts
- **probes** fire when code executes in a **provider** and send data to **consumers**




```
# ./all.d `pgrep xclock` XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                        U
  0   -> _XEventsQueued                     U
  0     -> _X11TransBytesReadable           U
  0     <- _X11TransBytesReadable           U
  0     -> _X11TransSocketBytesReadable     U
  0     <- _X11TransSocketBytesreadable     U
  0     -> ioctl                            U
  0       -> ioctl                          K
  0         -> getf                         K
  0           -> set_active_fd              K
  0           <- set_active_fd              K
  0         <- getf                         K
  0         -> get_udatamodel               K
  0         <- get_udatamodel               K
...
  0         -> releasef                     K
  0           -> clear_active_fd            K
  0           <- clear_active_fd            K
  0           -> cv_broadcast               K
  0           <- cv_broadcast               K
  0         <- releasef                     K
  0       <- ioctl                          K
  0     <- ioctl                            U
  0   <- _XEventsQueued                     U
  0 <- XEventsQueued                        U
```

END OF MODULE 1