Operating Systems

A Biswas,
Dept. of Information Technology

Syllabus

IT 402: OPERATING SYSTEMS

FM – 100 Contact Periods : 3L+1T per week

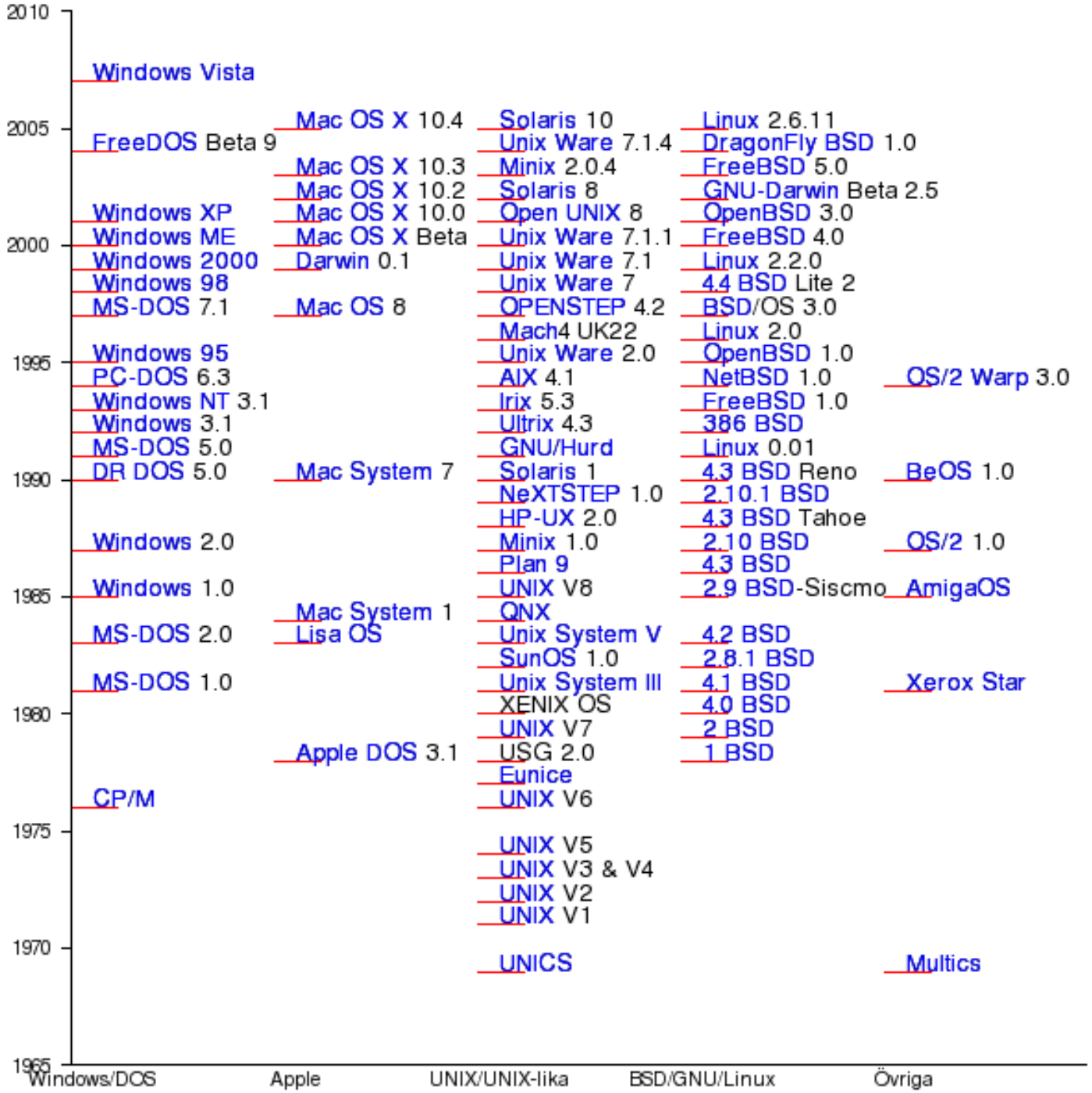
Operating System Overview, Evolution of Operating Systems, PC Hardware and x86 Programming, Address Spaces, Address Spaces on the x86, Structural overview, Interrupts -- hardware and software, privileged instructions, role of interrupts in operating system functions; Multiprocessing and Multiprogramming; Concept of process and Process synchronization, Process Management and Scheduling, Hardware requirements: protection, context switching, privileged mode; Threads and their Management,

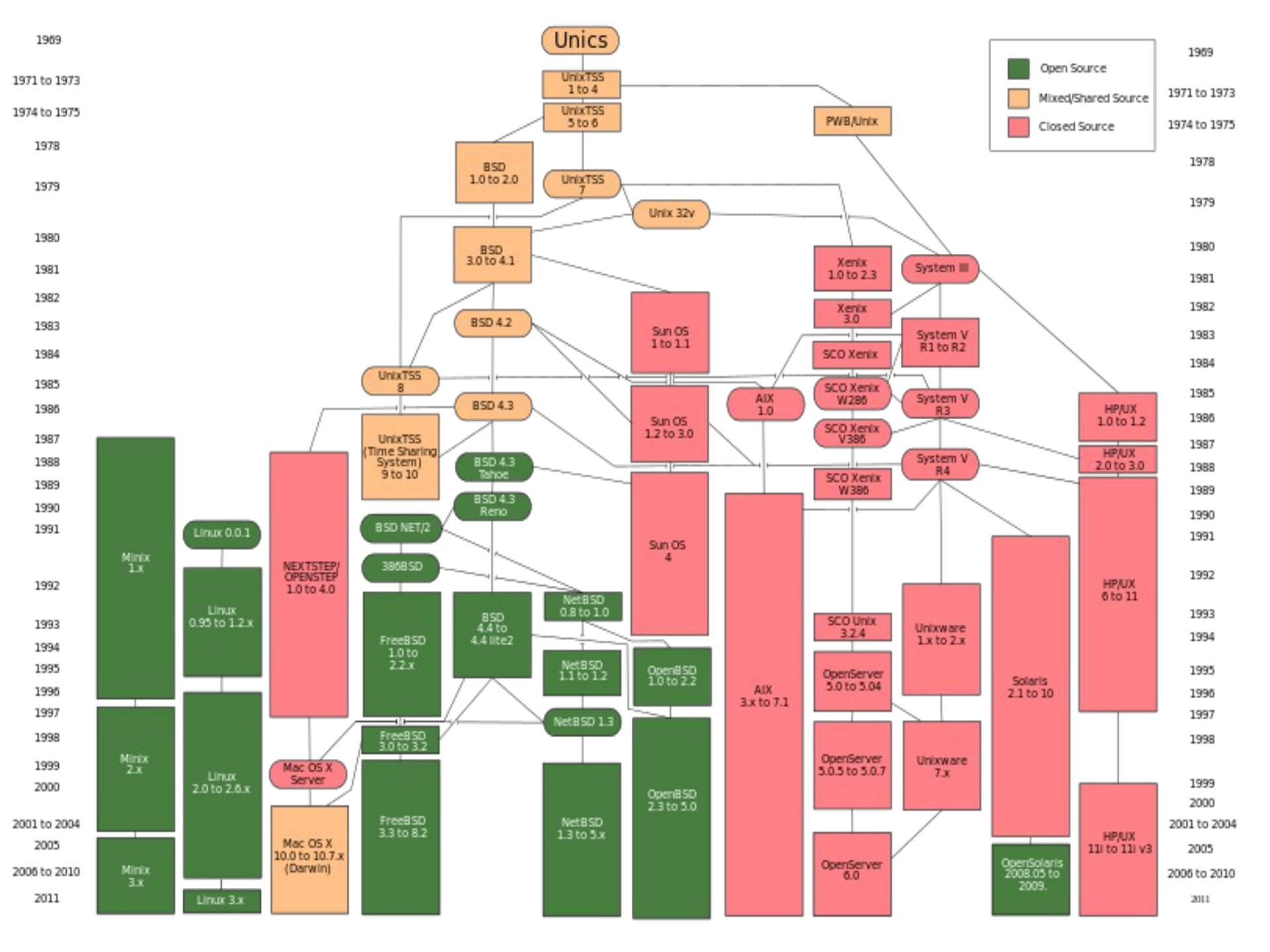
Interprocess Communication, Kernel API, Detection and Prevention of deadlocks, Dynamic Resource Allocation Memory Management: paging, virtual memory management, Design of IO systems, File Management, Device drivers, concept of driver routines.

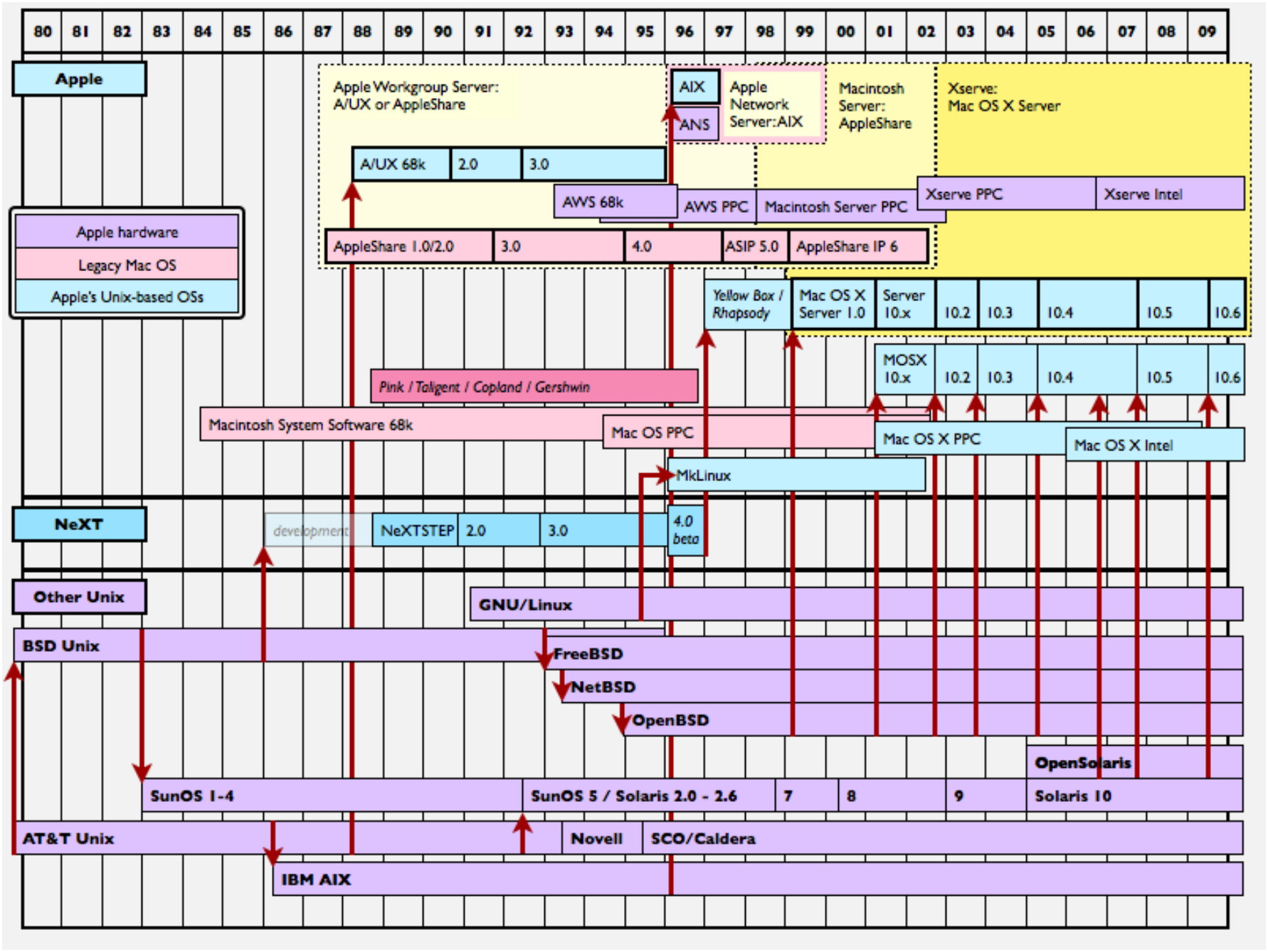
Case Studies: DOS/WINDOWS, UNIX/LINUX

Books

- Operating System Concepts –Galvin
- Computer Systems – Hallaron
- Crowley
- Dhamdhere
- Stallings
- M J Bach


2010
2005
2000
1995
1990
1985
1980
1975
1970
1965
Windows Vista
FreeDOS Beta 9
Windows XP
Windows ME
Windows 2000
Windows 98
MS-DOS 7.1
Windows 95
PC-DOS 6.3
Windows NT 3.1
Windows 3.1
MS-DOS 5.0
DR DOS 5.0
Windows 2.0
Windows 1.0
MS-DOS 2.0
MS-DOS 1.0
CP/M
Mac OS X 10.4
Mac OS X 10.3
Mac OS X 10.2
Mac OS X 10.0
Mac OS X Beta
Darwin 0.1
Mac OS 8
Mac System 7
Mac System 1
Lisa OS
Apple DOS 3.1
Solaris 10
Unix Ware 7.1.4
Minix 2.0.4
Solaris 8
Open UNIX 8
Unix Ware 7.1.1
Unix Ware 7.1
Unix Ware 7
OPENSTEP 4.2
Mach4 UK22
Unix Ware 2.0
AIX 4.1
Irix 5.3
Ultrix 4.3
GNU/Hurd
Solaris 1
NeXTSTEP 1.0
HP-UX 2.0
Minix 1.0
Plan 9
UNIX V8
QNX
Unix System V
SunOS 1.0
Unix System III
XENIX OS
UNIX V7
USG 2.0
Eunice
UNIX V6
UNIX V5
UNIX V3 & V4
UNIX V2
UNIX V1
UNICS
Linux 2.6.11
DragonFly BSD 1.0
FreeBSD 5.0
GNU-Darwin Beta 2.5
OpenBSD 3.0
FreeBSD 4.0
Linux 2.2.0
4.4 BSD Lite 2
BSD/OS 3.0
Linux 2.0
OpenBSD 1.0
NetBSD 1.0
FreeBSD 1.0
386 BSD
Linux 0.01
4.3 BSD Reno
2.10.1 BSD
4.3 BSD Tahoe
2.10 BSD
4.3 BSD
2.9 BSD-Siscmo
4.2 BSD
2.8.1 BSD
4.1 BSD
4.0 BSD
2 BSD
1 BSD
OS/2 Warp 3.0
BeOS 1.0
OS/2 1.0
AmigaOS
Xerox Star
Multics
Windows/DOS
Apple
UNIX/UNIX-lika
BSD/GNU/Linux
Övriga


Unics
Open Source
Mixed/Shared Source
Closed Source
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2010
2011
UnixTSS 1 to 4
UnixTSS 5 to 6
UnixTSS 7
UnixTSS 8
UnixTSS (Time Sharing System) 9 to 10
PWB/Unix
Unix 32v
BSD 1.0 to 2.0
BSD 3.0 to 4.1
BSD 4.2
BSD 4.3
BSD 4.3 Tahoe
BSD 4.3 Reno
BSD NET/2
386BSD
BSD 4.4 to 4.4 lite2
Sun OS 1 to 1.1
Sun OS 1.2 to 3.0
Sun OS 4
AIX 1.0
AIX 3.x to 7.1
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix W286
SCO Xenix V386
SCO Xenix V386
SCO Unix 3.2.4
OpenServer 5.0 to 5.0.4
OpenServer 5.0.5 to 5.0.7
OpenServer 6.0
System III
System V R1 to R2
System V R3
System V R4
Unixware 1.x to 2.x
Unixware 7.x
Solaris 2.1 to 10
OpenSolaris 2008.05 to 2009.
HP/UX 1.0 to 1.2
HP/UX 2.0 to 3.0
HP/UX 6 to 11
HP/UX 11i to 11i v3
Minix 1.x
Minix 2.x
Minix 3.x
Linux 0.0.1
Linux 0.95 to 1.2.x
Linux 2.0 to 2.6.x
Linux 3.x
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
Mac OS X 10.0 to 10.7.x (Darwin)
FreeBSD 1.0 to 2.2.x
FreeBSD 3.0 to 3.2
FreeBSD 3.3 to 8.2
NetBSD 0.8 to 1.0
NetBSD 1.1 to 1.2
NetBSD 1.3
NetBSD 1.3 to 5.x
OpenBSD 1.0 to 2.2
OpenBSD 2.3 to 5.0


80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 00 01 02 03 04 05 06 07 08 09
Apple
Apple Workgroup Server:
A/UX or AppleShare
AIX
ANS
Apple Network Server:AIX
Macintosh Server: AppleShare
Xserve: Mac OS X Server
A/UX 68k
2.0
3.0
AWS 68k
AWS PPC
Macintosh Server PPC
Xserve PPC
Xserve Intel
AppleShare 1.0/2.0
3.0
4.0
ASIP 5.0
AppleShare IP 6
Apple hardware
Legacy Mac OS
Apple's Unix-based OSs
Yellow Box / Rhapsody
Mac OS X Server 1.0
Server 10.x
10.2
10.3
10.4
10.5
10.6
MOSX 10.x
10.2
10.3
10.4
10.5
10.6
Pink / Taligent / Copland / Gershwin
Macintosh System Software 68k
Mac OS PPC
Mac OS X PPC
Mac OS X Intel
MkLinux
NeXT
development
NeXTSTEP
2.0
3.0
4.0 beta
Other Unix
GNU/Linux
BSD Unix
FreeBSD
NetBSD
OpenBSD
OpenSolaris
SunOS 1-4
SunOS 5 / Solaris 2.0 - 2.6
7
8
9
Solaris 10
AT&T Unix
Novell
SCO/Caldera
IBM AIX

How a computer system works

A ***computer system*** is a collection of hardware and software components that work together to run computer programs.

Hello world !!!

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n");
6 }
```

ASCII representation of hello.c

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46

h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123

\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108

l	o	,	<sp>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

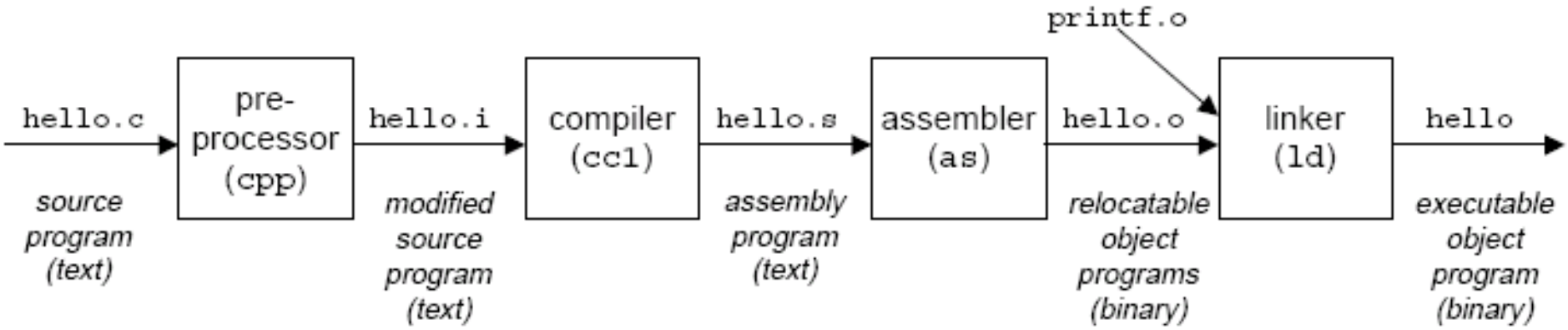
Hello.c is translated to other form

- hello.c is a high-level C program - read and understood by human

- hello.c is translated by other programs into a sequence of low-level *machine-language* instructions.

- These instructions are then packaged in a form called an *executable object program*, and stored as a binary disk file.

Compilation system



Compilation System

- Four phases:
  - Preprocessor
  - Compiler
  - Assembler
  - Linker

1. Preprocessing phase

- Modifies the original C program according to directives (begins with the # character).

- Example: #include<stdio.h>

    1. reads the contents of the system header file stdio.h and
    2. insert it directly into the program text.

hello.c → hello.i

$ cpp hello.c

2. Compilation phase

hello.i → hello.s

- hello.s which contains an *assembly-language program*.

- Each statement in an assembly-language program exactly describes one low-level machine-language instruction in a standard text form.
- $gcc –S hello.c

3. Assembly phase

hello.s → hello.o

- The assembler (as) translates hello.s into machine-language instructions, packages them in a form known as a *relocatable object program*, and stores the result in the object file hello.o.

- The hello.o file is a binary file whose bytes encode machine language instructions rather than characters.

$ gcc –c hello.s

4. Linking phase

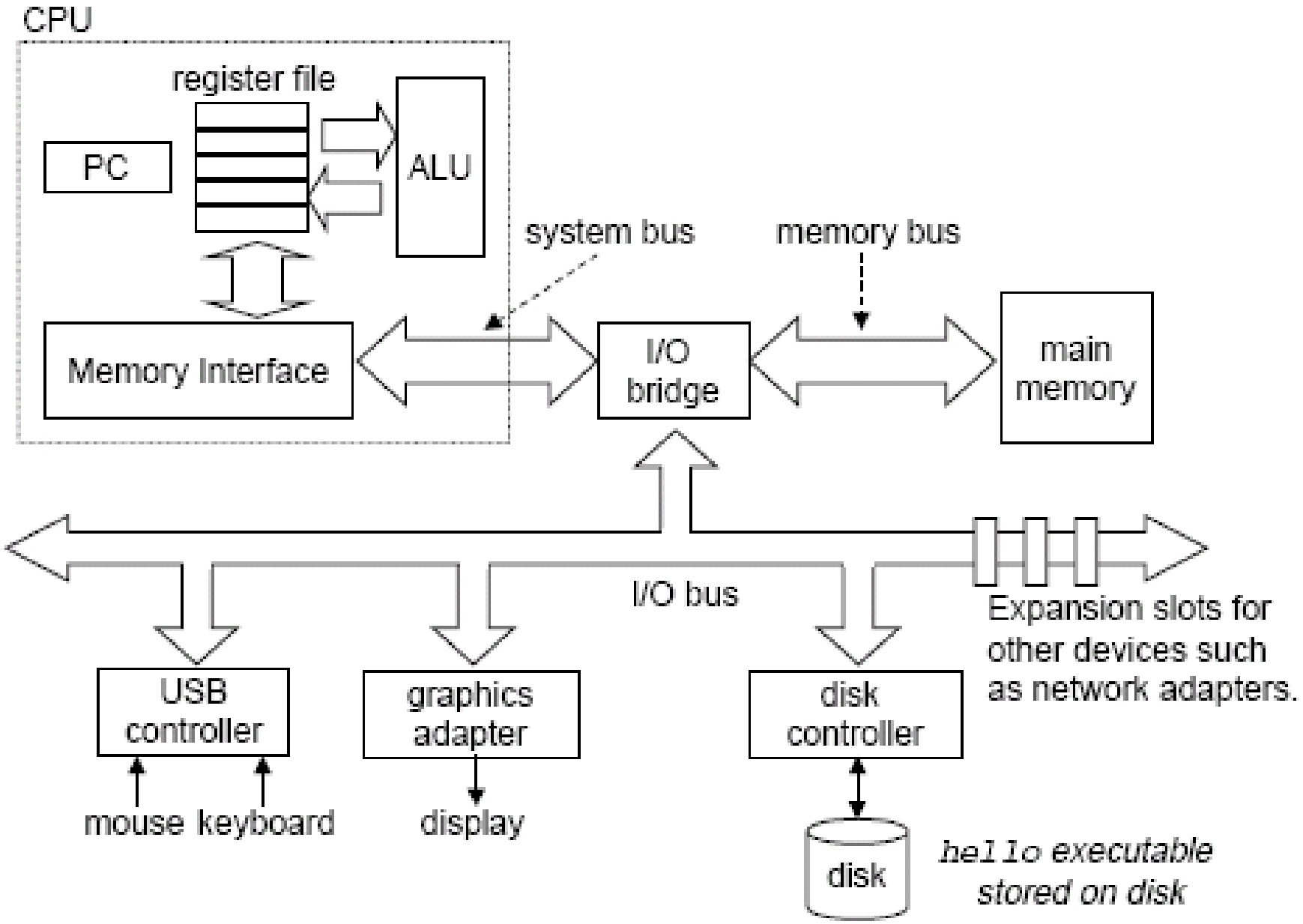
hello.o → hello

Notice that our hello program calls the printf function, the printf function resides in a separate precompiled object file called printf.o, which must somehow be merged with our hello.o program.

The linker (ld) handles this merging. The result is the hello file, which is an *executable object file* (or simply *executable*) that is ready to be loaded into memory and executed by the system.

Shell

```
$ ./hello
Hello world
$
```



Hardware organization

Hardware Organization

Buses:

- a collection of electrical conduits called *buses* that carry bytes of information back and forth between the components.
- designed to transfer fixed-sized chunks of bytes known as *words*.
- The number of bytes in a word (the *word size*) is a fundamental system parameter that varies across systems.
- Intel -4, Sparc – 8

Hardware Organization

I/O Devices:

Input/output (I/O) devices are the system's connection to the external world.

- a keyboard and
- mouse for user input,
- a display for user output, and
- a disk drive (or simply disk) for long-term storage of data and programs.

Hardware Organization

I/O Devices:

- the hello executable program resides on the disk.
- each I/O device is connected to the I/O bus by either a *controller* or an *adapter*.
- the purpose of the controller is to transfer information back and forth between the I/O bus and an I/O device.

Hardware Organization

Main Memory:

The *main memory* is a temporary storage device that holds both a program and the data it manipulates while the processor is executing the program.

Physically, main memory consists of a collection of *Dynamic Random Access Memory (DRAM)* chips.

Logically, memory is organized as a linear array of bytes, each with its own unique address (array index) starting at zero.

Hardware Organization

Main Memory:

In general, each of the machine instructions that constitute a program can consist of a variable number of bytes.

The sizes of data items that correspond to C program variables vary according to type.

For example, on an Intel machine running Linux, data of type short requires two bytes, types int, float, and long four bytes, and type double eight bytes.

Processor

- CPU is the engine that interprets (or *executes*) instructions stored in main memory.
- At its core is a word-sized storage device (or *register*) called the *program counter* (PC).
- PC points at (contains the address of) some machine-language instruction in main memory.
- It reads the instruction from memory pointed at by the program counter (PC),
  - interprets the bits in the instruction,
  - performs some simple *operation* dictated by the instruction, and then
  - updates the PC to point to the *next* instruction

Processor

A few simple operations:

Load: Copy a byte or a word from main memory into a register, overwriting the previous contents of the register.

Store: Copy the a byte or a word from a register to a location in main memory, overwriting the previous contents of that location.

Update: Copy the contents of two registers to the ALU, which adds the two words together and stores the result in a register, overwriting the previous contents of that register.

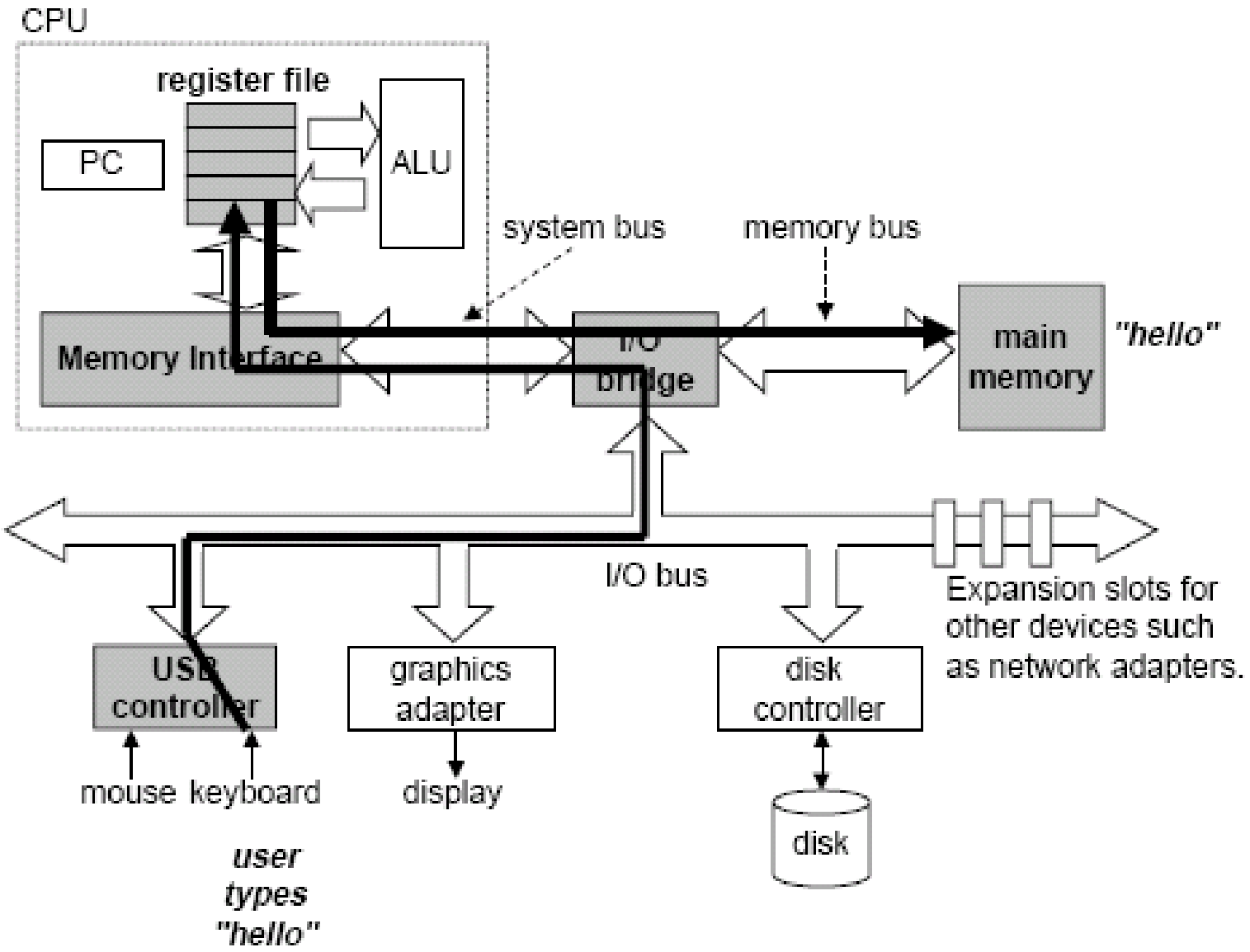
I/O Read: Copy a byte or a word from an I/O device into a register.

I/O Write: Copy a byte or a word from a register to an I/O device.

Jump: Extract a word from the instruction itself and copy that word into the program counter (PC), overwriting the previous value of the PC.

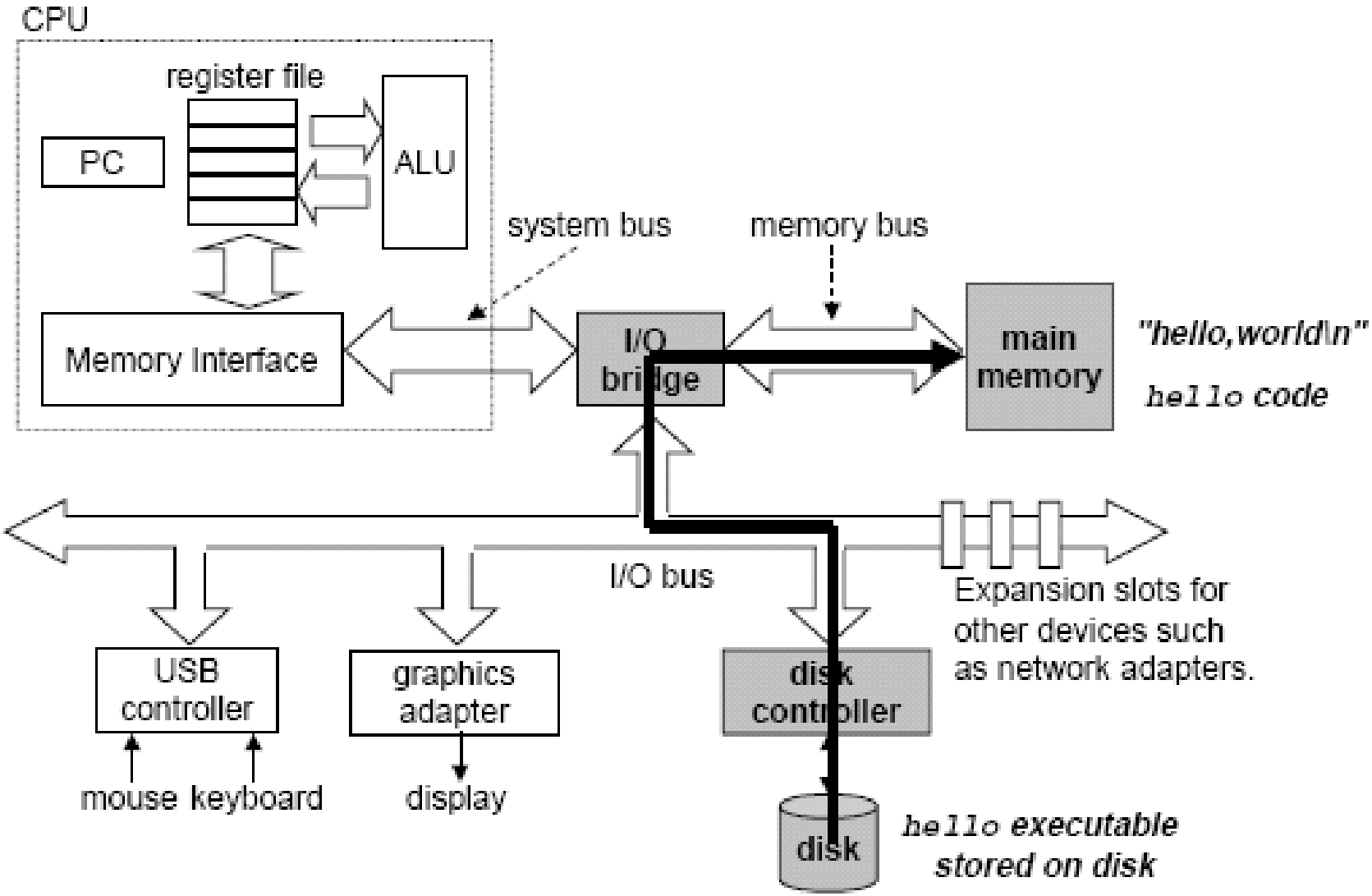
Running the Hello program

1. Shell reads hello 2. Stores in registers 3. Stores in main memory



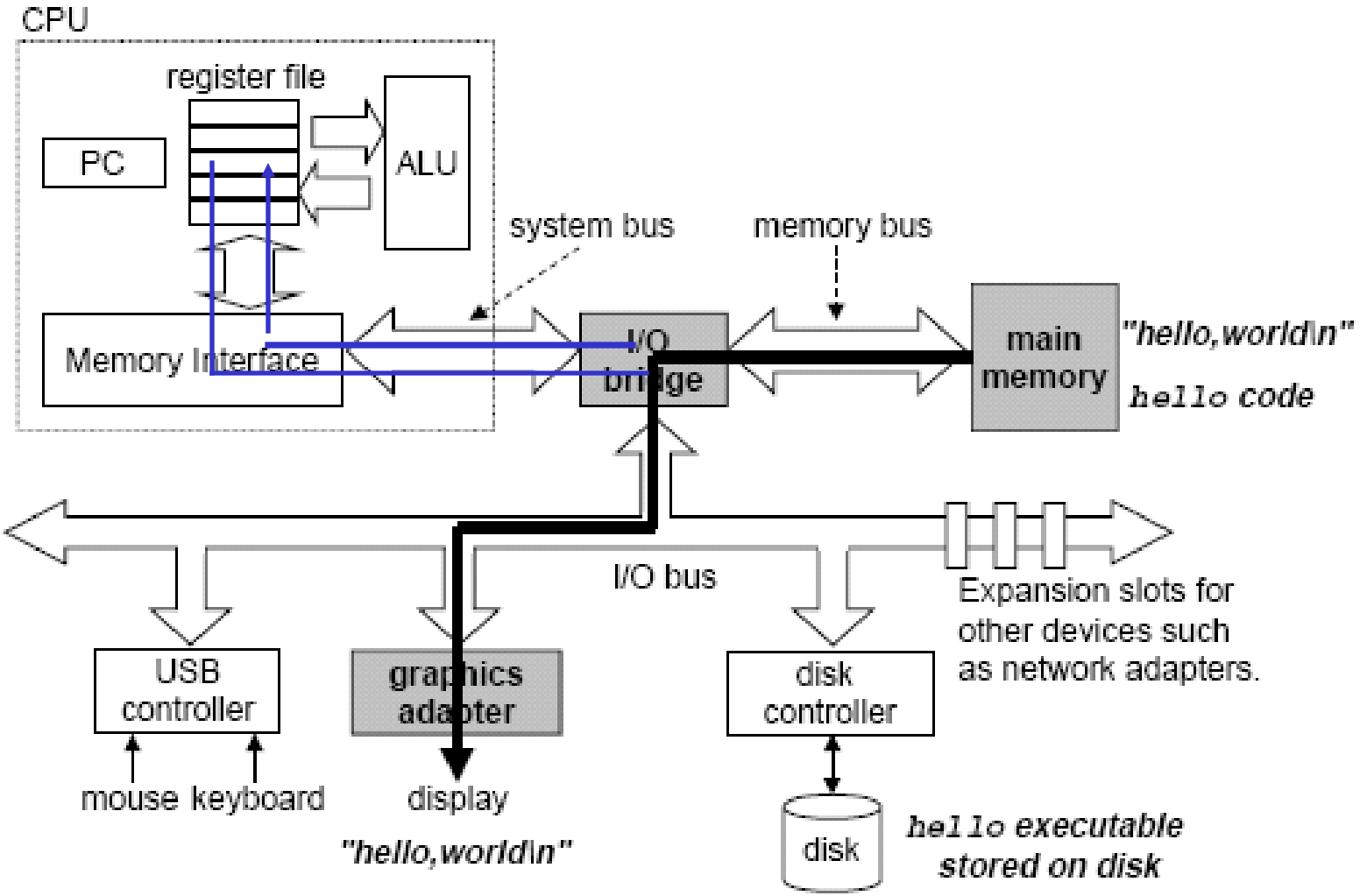
Running the Hello program

1. Hit the enter key 2. Shell loads executable hello from disk to main memory (DMA)

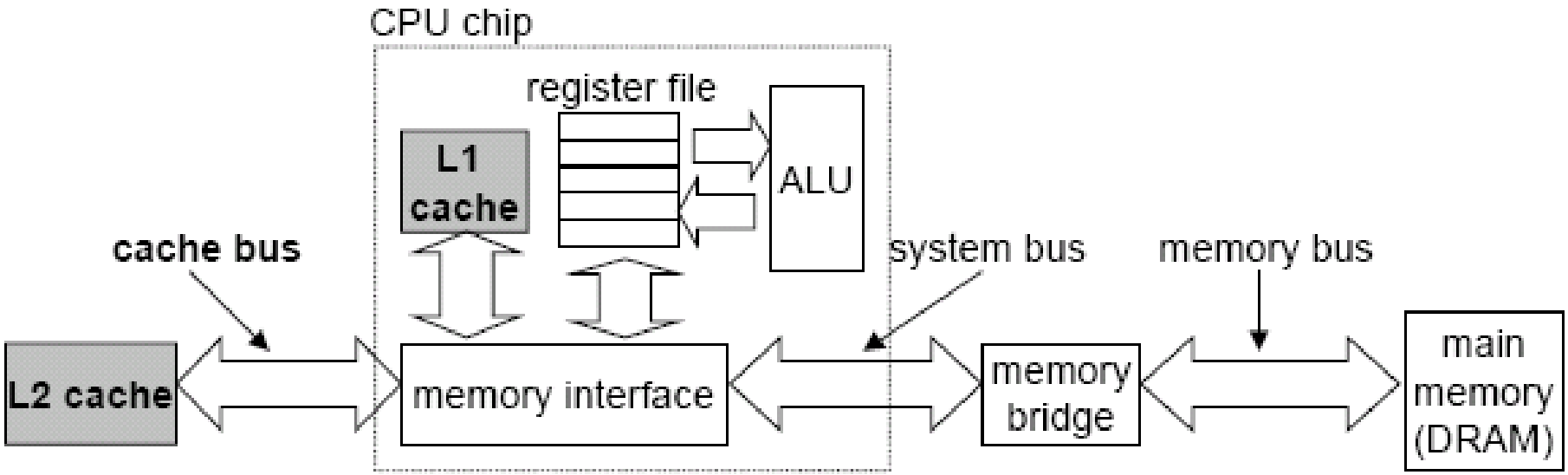


Running the Hello program

1. Processor executes 2. “hello world” copied to registers 3. then to display device



Running the Hello program

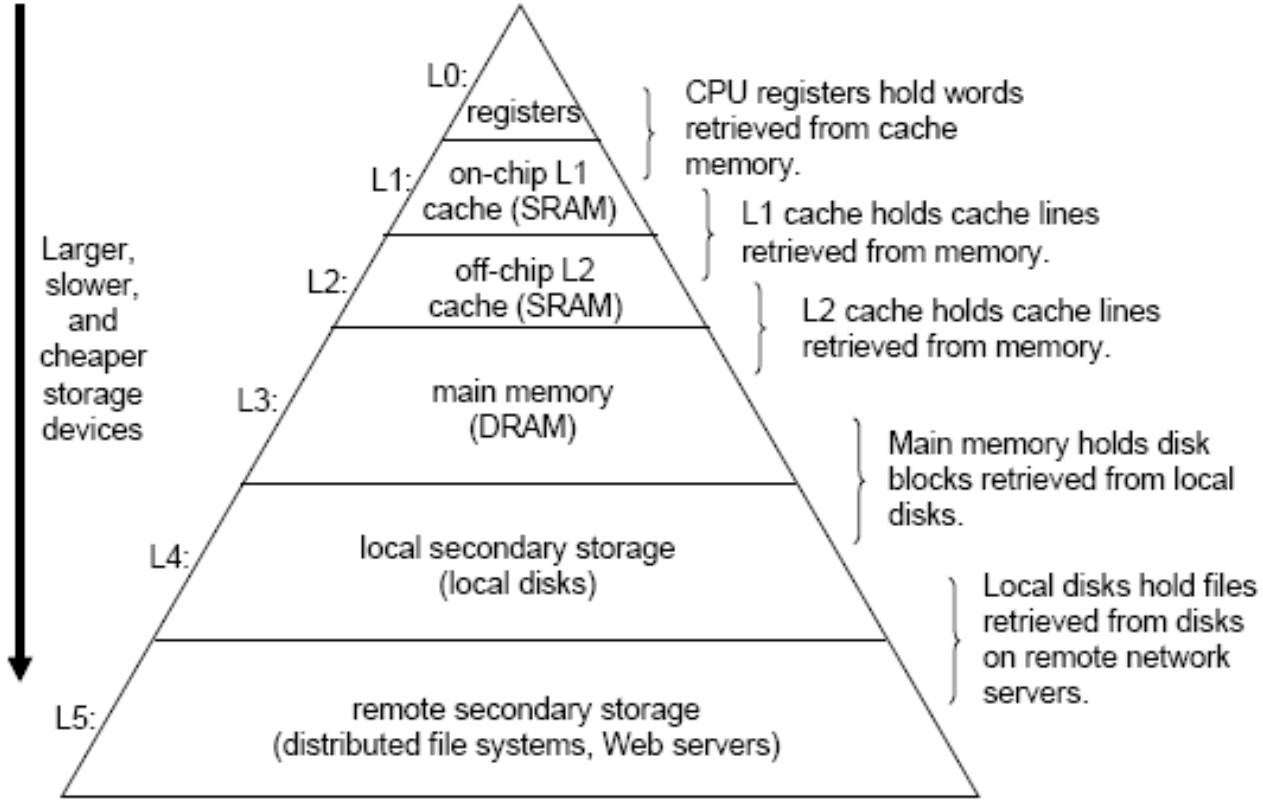


Disk drives are 10 million times slower than the main memory.

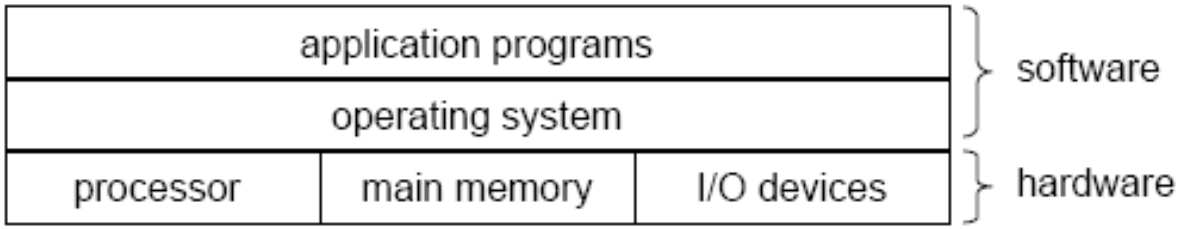
Registers are 100 times faster than main memory.

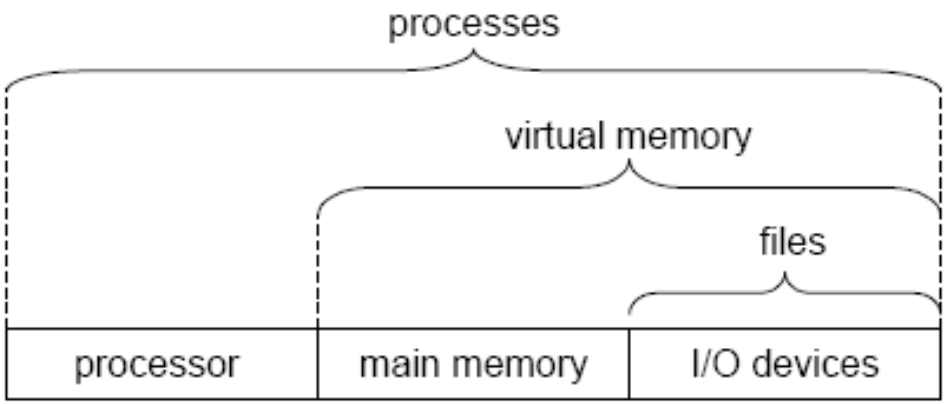
L1 and L2 caches are implemented using Static RAM.

Memory Hierarchy

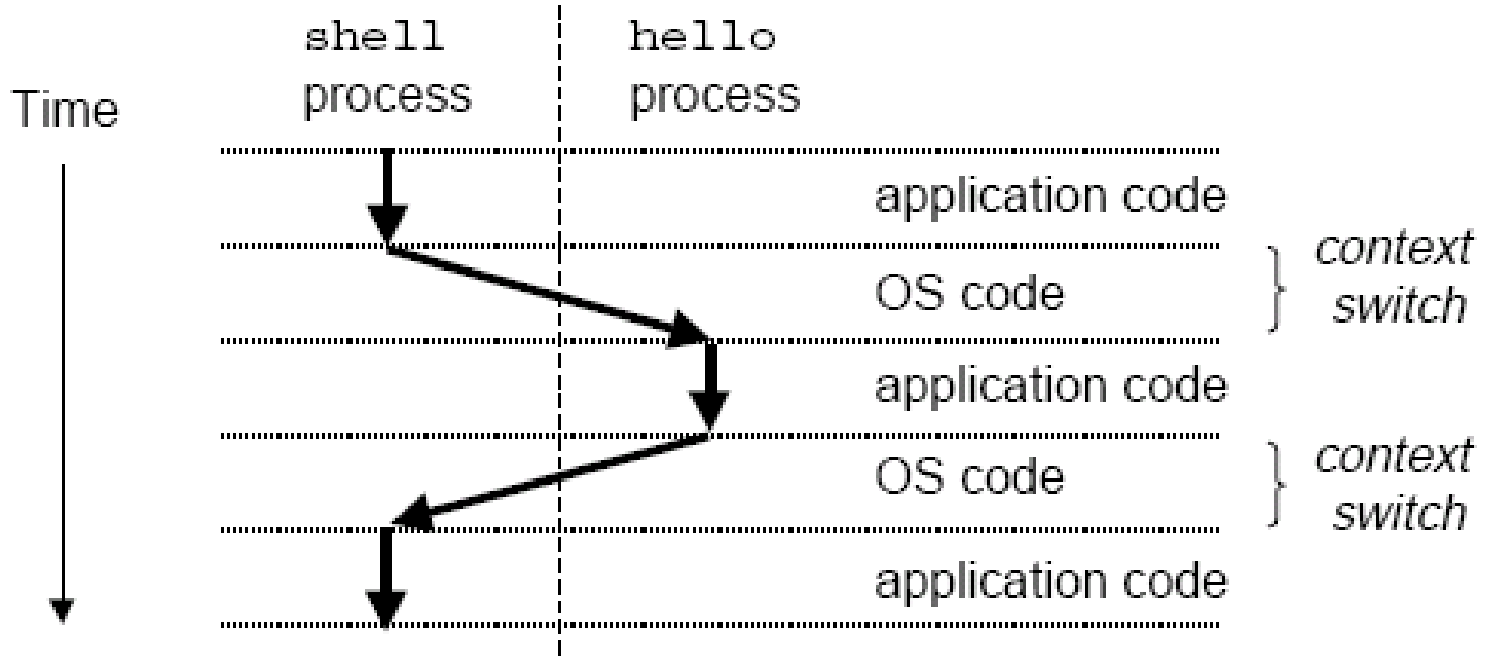


Running the Hello program





Process and context switch



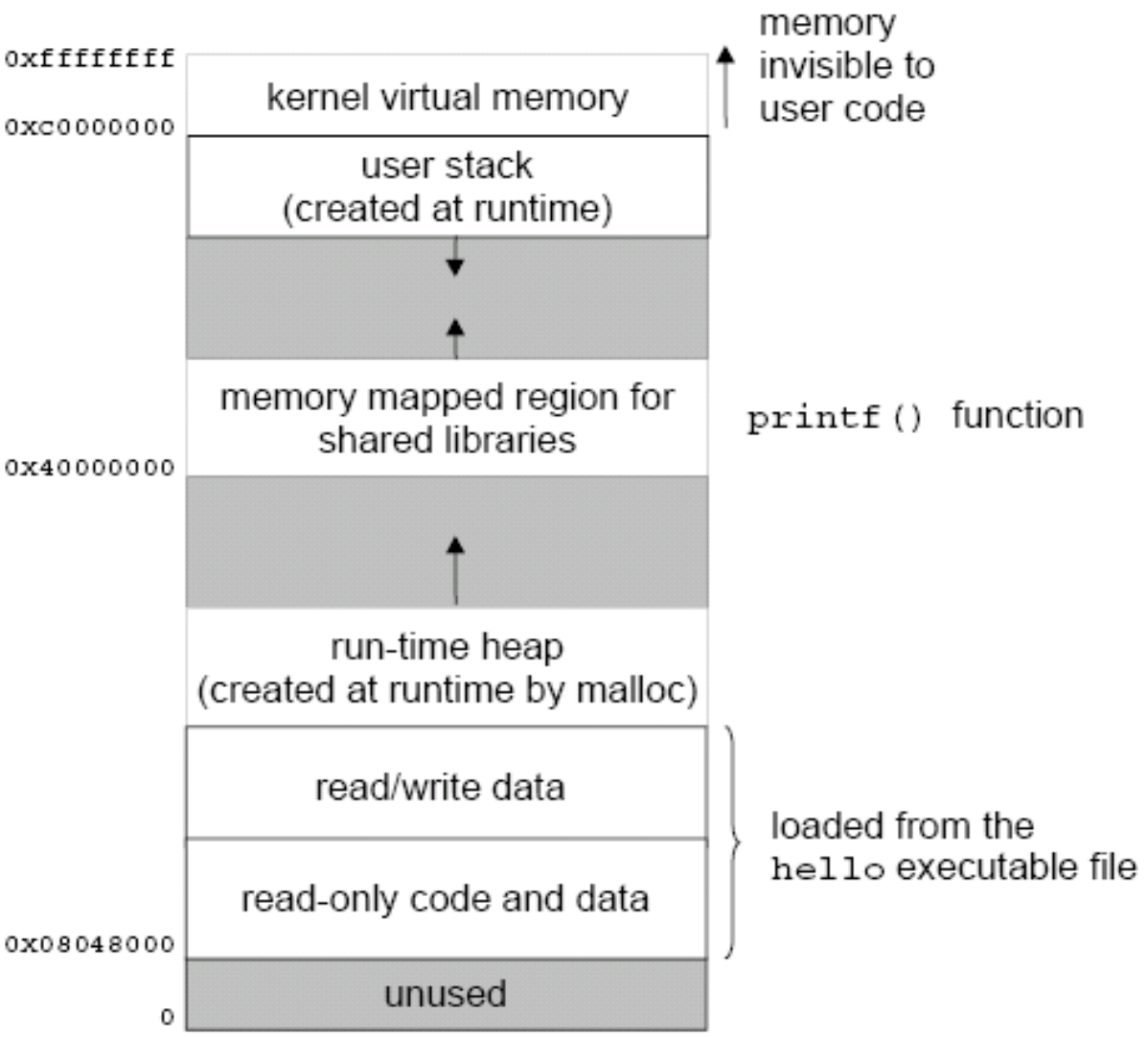
Virtual Memory

Stack:
Used for function calls. Expands and contracts with a function call and retrun

Shared Libraries:
Libraries shared by several processes.

Heap:
Expands and contracts dynamically as malloc and free are called.

Program code and data:
Machine-level instructions, C variables from hello executable.



A network is just another I/O device