

Teaching **L**ondon **C**omputing

A Level Computer Science

Topic 7: OS and Running a Program



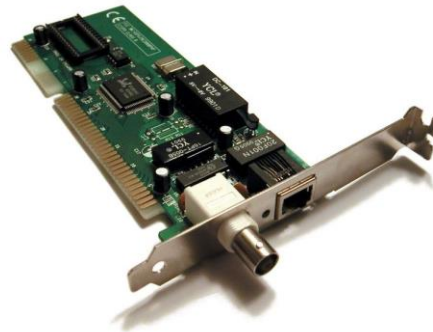
Aims

- Recap Operating Systems
 - Compiling and linking
 - From LMC to an OS
 - Challenges of running programs
 - Processes
 - Memory management
-

Recap: What's an OS?

OS Handlers Interfaces

- I/O devices
 - Keyboard
 - Monitor
 - Network
 - Disk
- File system
- User interface



ISA bus
Ethernet card



System Calls

- The OS is a program
 - Other programs use OS functions
- Some examples using Python

```
import os, sys

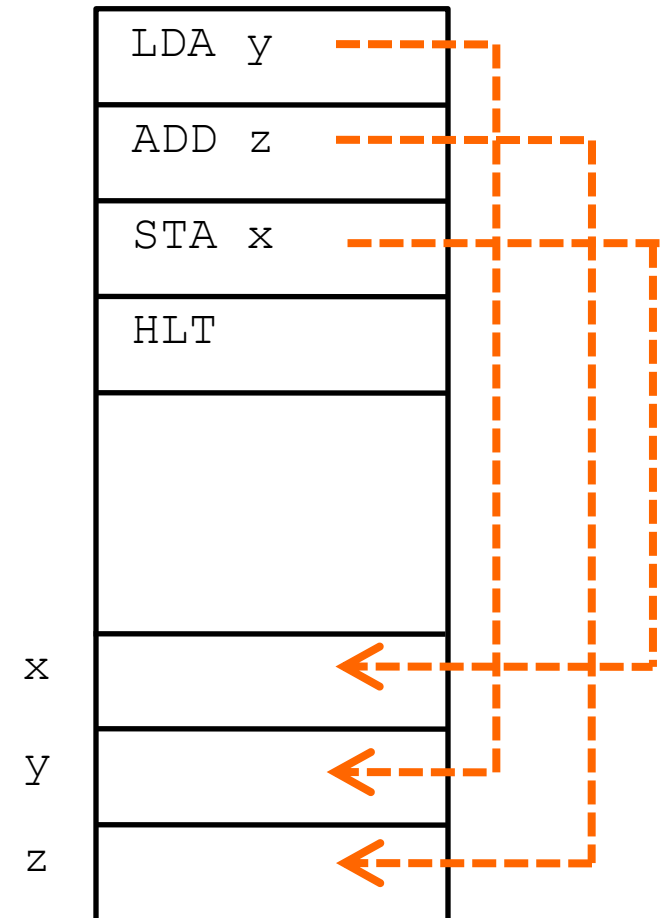
print("OS name:", os.name)
print("Platform:", sys.platform)
print("Process id:", os.getpid())
print("User id:", os.getuid())
print("File meta data:", os.stat("os1.py"))
print("Current directory", os.getcwd())
os.chdir("..")
print("Current directory", os.getcwd())

for root, dirs, files in os.walk('.'):
    if len(dirs) > 0 :
        print(root, "contains directories:", dirs)
```

Recall LMC

- Variables
 - Instruction contain the address of data
- Branches
 - Instructions contain address of instructions

But do we know where everything is?



Challenges Beyond LMC

- **Computer runs many programs**
 - Applications
 - Interfaces to device – e.g. networks, keyboard
 - **Many programs share memory**
 - Which memory locations free?
 - How much memory needed?
 - **Programs are incomplete!**
 - Only the OS knows how to do I/O
 - Applications interface to OS
-


Overview of Solutions

- Compiler does not generate a complete program
 - **Linker** combines parts
 - **Loader** places it in memory
 - OS manages processes and interrupts
 - **Interrupt handlers** respond to I/O events
 - Applications run in **processes**
 - **Scheduler** juggles processes
 - Processor and OS Manage Memory
 - Memory organised in **pages**
 - Addresses are **translated**
 - Pages come and go
-



Options for Practical Work

Challenge and Options

- Pupils think that GUI is the Computer
 - ... or Where?
 - Microsoft + IT manager want to keep it like that
 - Raspberry Pi: e.g. headless
 - See e.g. <https://www.youtube.com/watch?v=Ioih6MHNNqc>
 - Virtual machine
 - Local or Web-based e.g. c9.io
-

Comparison

Raspberry Pi

- Hard to set up without interacting with network
- Clear what is happening
- Easy to manage once setup
- Available to pupils

Virtual Machine

- Local
 - Moderately clear
 - Some management issues
- Web based
 - Easy to deliver
 - Potential confusion
 - ... especially if s/w IDE

Is practical work important for OS?

Exercise: Set Up c9.io

- Cloud Nine is a web-based s/w development environment
 - command line programs
 - web programs
 - Create a free account on c9.io
 - Need access to an email account
 - Create a basic project (see exercise sheet)
-

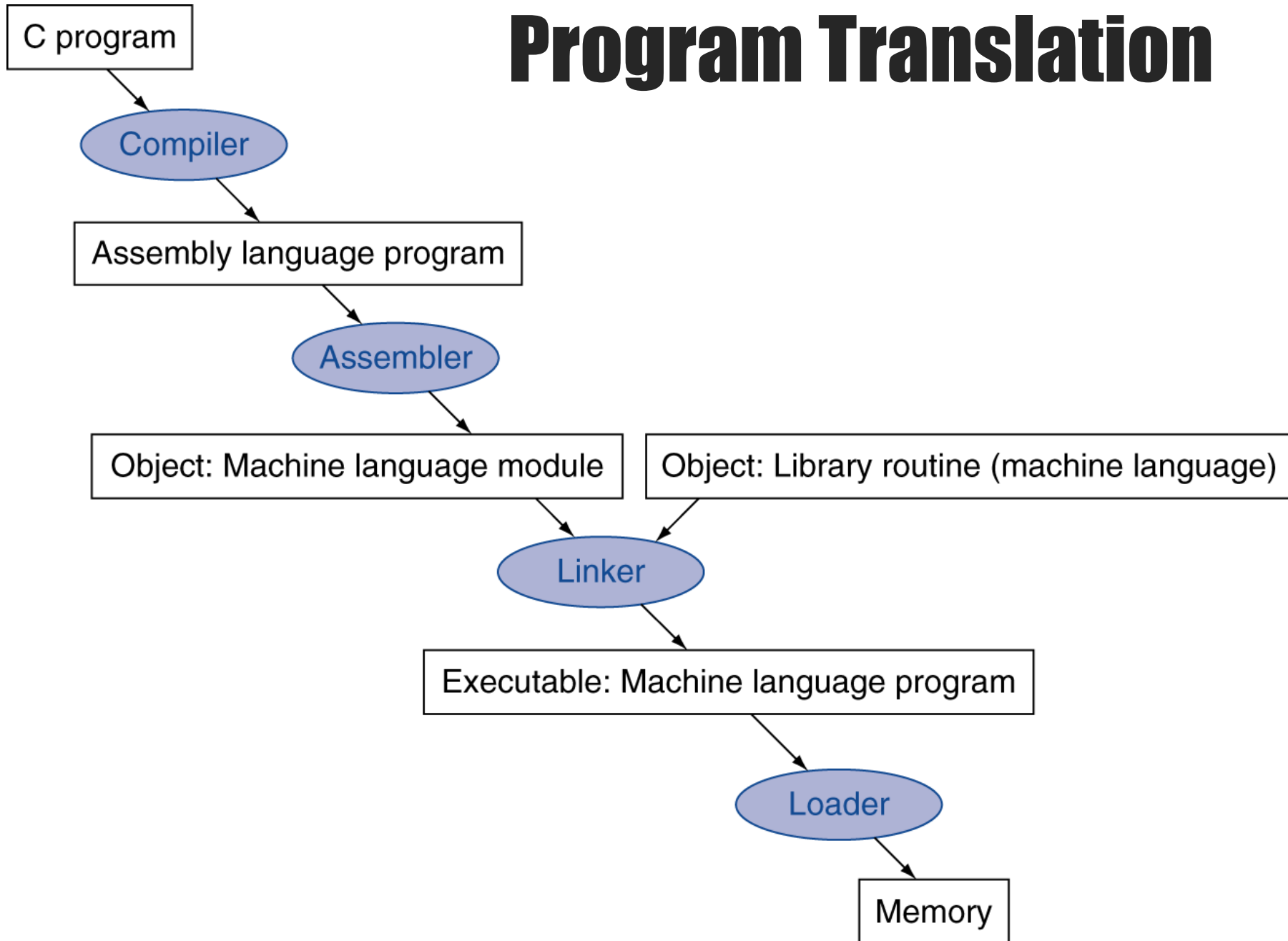
Exercise 1.1- 1.6 Command Line

- Use the command line to
 - Look at files
 - Create files and directories
 - Run programs
-

Compiling and Linking

Problem 1: compiler does not generate a complete program

Program Translation



Assembly & Object Code

- Assembly code
 - Textual representation of machine code
 - Produced by compiler
 - Object code
 - Binary representation
 - Produced by assembler
 - Normally compiler and assembler combined
-

Example C Program

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, const char *argv[]) {

    // check for argument
    if (argc < 2) {
        return 1 ;
    }

    // convert argument
    int num = atoi(argv[1]);
    printf("%d\n", num+1);
    return 0 ;
}
```

11:main.c **** int num = atoi(argv[1]);

36 .loc 1 12 0

37 001c 488B45E0 movq -32(%rbp), %rax

38 0020 4883C008 addq \$8, %rax

39 0024 488B00 movq (%rax), %rax

40 0027 4889C7 ~~movq %rax, %rdi~~

41 002a E8000000 call atoi

41 00 00

42 002f 8945FC ~~movl %eax, -4(%rbp)~~

12:main.c **** printf("%d\n", num+1);

43 .loc 1 13 0

44 0032 8B45FC movl -4(%rbp), %eax

45 0035 8D5001 leal 1(%rax), %edx

46 0038 B8000000 movl \$.LC0, %eax

46 00 00

47 003d 89D6 movl %edx, %esi

48 003f 4889C7 movq %rax, %rdi

49 0042 B8000000 movl \$0, %eax

49 00 00

50 0047 E8000000 call printf

Directive

Assembly
Instruction

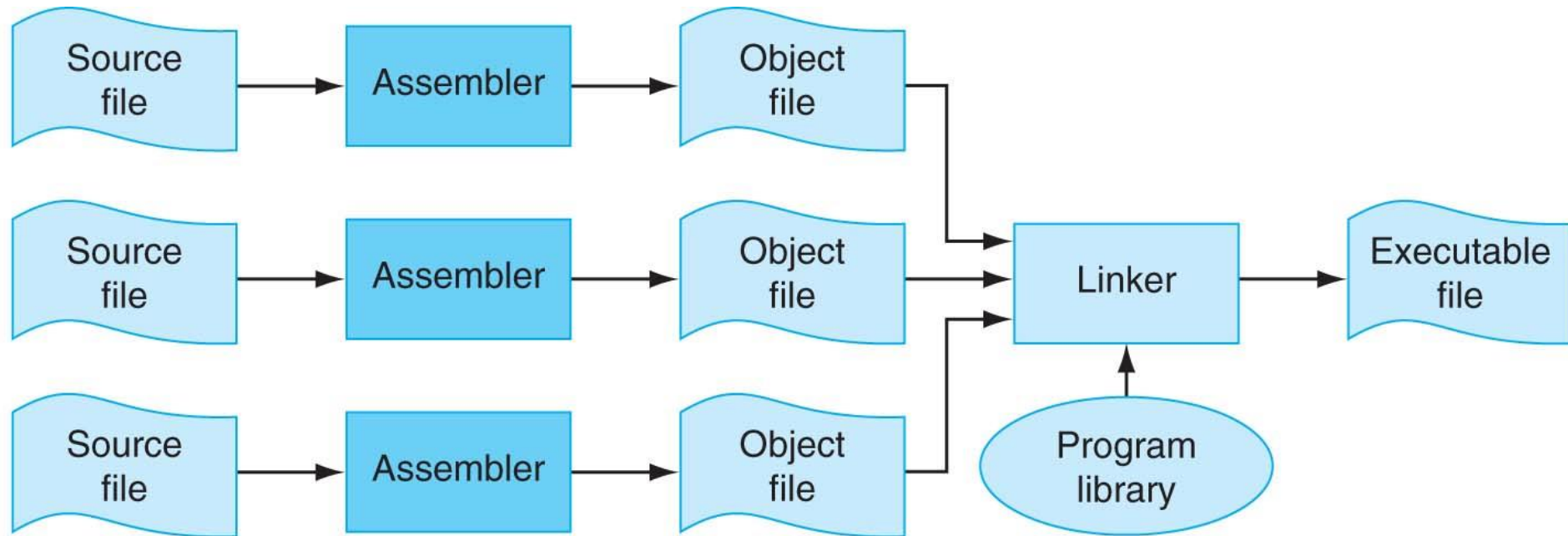
Machine
Instruction

Address

Symbol

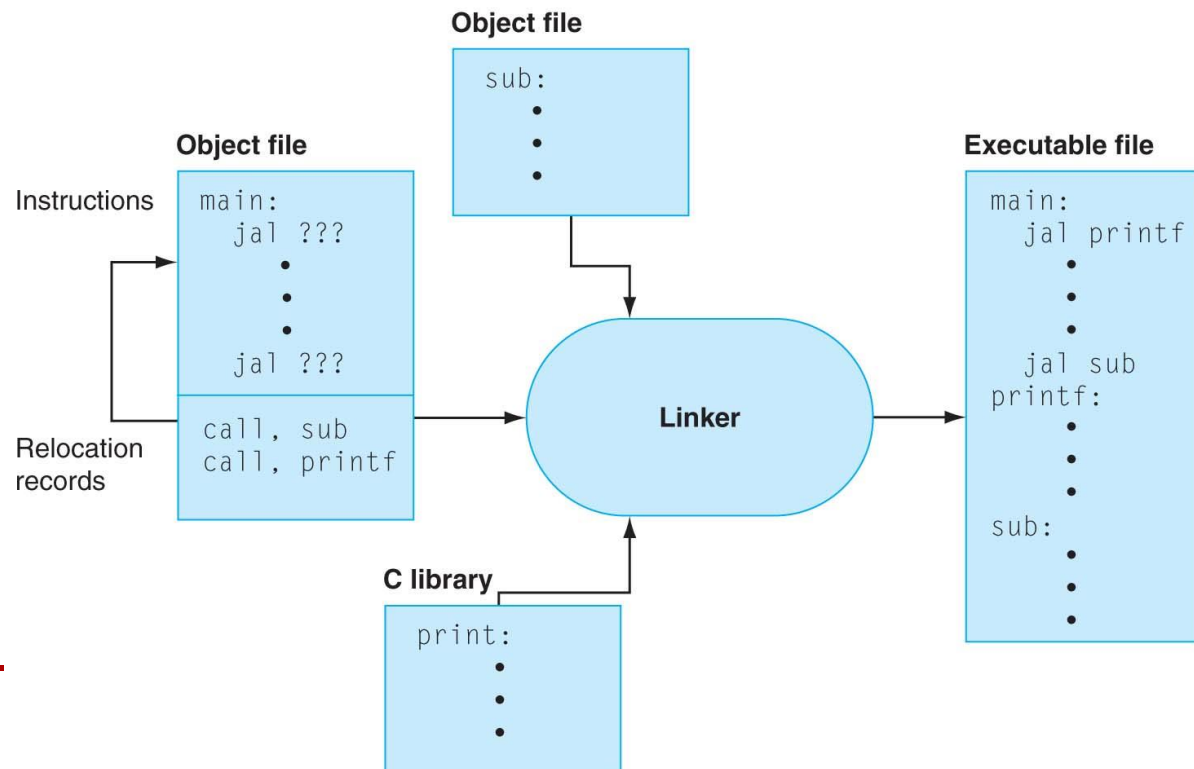
Separate Compilation

- No need to compile all code at once
 - Library code
 - Separate files
- Linker combines the object files



Linker Stitches Files Together

- Searches object files and libraries for non-local symbols
- Resolves references in different files
- Combines them into a single executable file



Disassembly: objdump

```
// check for argument
if (argc < 2) {
    return 1 ;
}
```

Compare

Jump

**Jump (to
end)**

0000000000400504 <main>:

```
400504: 55
400505: 48 89 e5
400508: 48 83 ec 20
40050c: 89 7d ec
40050f: 48 89 75 e0
400513: 83 7d ec 01
400517: 7f 07
400519: b8 01 00 00 00
40051e: eb 35
```

```
push    %rbp
mov     %rsp,%rbp
sub     $0x20,%rsp
mov     %edi,-0x14(%rbp)
mov     %rsi,-0x20(%rbp)
cmpl    $0x1,-0x14(%rbp)
jg      400520 <main+0x1c>
mov     $0x1,%eax
jmp     400555 <main+0x51>
```

Disassembly: objdump

Unresolved jump

00000000000400504 <main>:

...

400520:	48 8b 45 e0	mov	-0x20(%rbp), %rax
400524:	48 83 c0 08	add	\$0x8, %rax
400528:	48 8b 00	mov	(%rax), %rax
40052b:	48 89 c7	mov	%rax, %rdi
40052e:	e8 dd fe ff ff	callq	400410 <atoi@plt>
400533:	89 45 fc	mov	%eax, -0x4(%rbp)
400536:	8b 45 fc	mov	-0x4(%rbp), %eax
400539:	8d 50 01	lea	0x1(%rax), %edx
40053c:	b8 58 06 40 00	mov	\$0x400658, %eax
400541:	89 d6	mov	%edx, %esi
400543:	48 89 c7	mov	%rax, %rdi
400546:	b8 00 00 00 00	mov	\$0x0, %eax
40054b:	e8 a0 fe ff ff	callq	4003f0 <printf@plt>
400550:	b8 00 00 00 00	mov	\$0x0, %eax
400555:	c9	leaveq	
400556:	c3	retq	

Loading a Program

- Load from image file on disk into memory
 1. Read header to determine sizes
 2. Allocates space in memory
 3. Copy text and initialized data into memory
 4. Set up arguments on stack
 5. Initialize registers
 6. Jump to startup routine
 7. ... and calls main
 8. When main returns, do “exit” systems call
 - Loader also does dynamic linking
-

Summary

- Compiled programs are incomplete
- Linker adds libraries
- ... but linked programs are incomplete too
- Loader resolves references to shared libraries (dynamic linking)

Still to solve:

- How can several programs run at once?
 - How can programs move around in memory?
-

Exercise 2.1 – 2.4

- Use the online VM to compile a C program and understand
 - Libraries
 - Linking
-

Processes & Interrupts

Lots of stuff happening at once!

Interrupts

- I/O device speed varies
- How many CPU clock cycles per keystroke?
- How to avoid CPU waiting for I/O?
- Interrupt
 - Signal from outside the CPU
 - ... changes the program

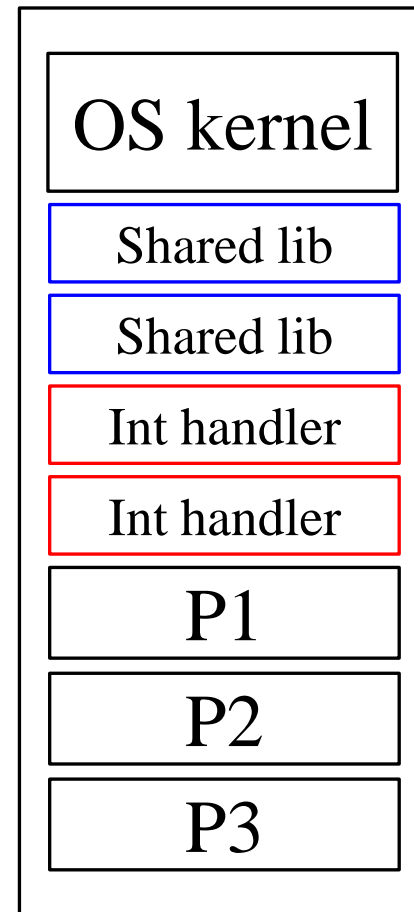


Processes

- An instance of a program running
 - Process has memory
 - Process does I/O
 - Processes are created by other processes
 - Multiple processes
 - Avoid waiting for I/O – better throughput
 - User can listen to music while coding
 - One process **protected** from another
 - Looping program?
 - Blue screen?
-

Inside the Computer

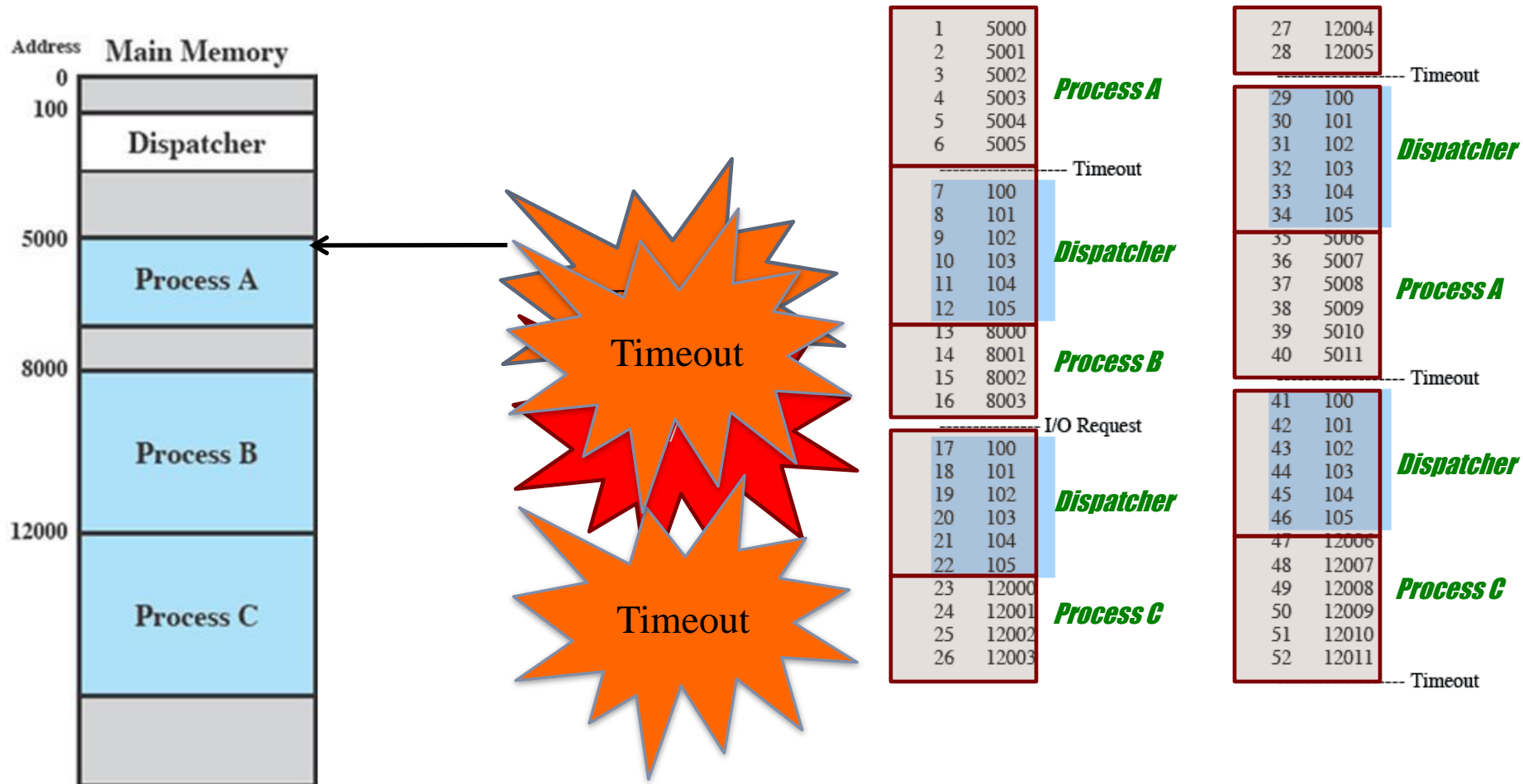
- Interrupt handler
 - Responds to I/O
- Libraries
 - Shared between programs
- Processes
 - Instances of a program running
- OS Kernel
 - Keeps track!



How Many Programs Run at Once?

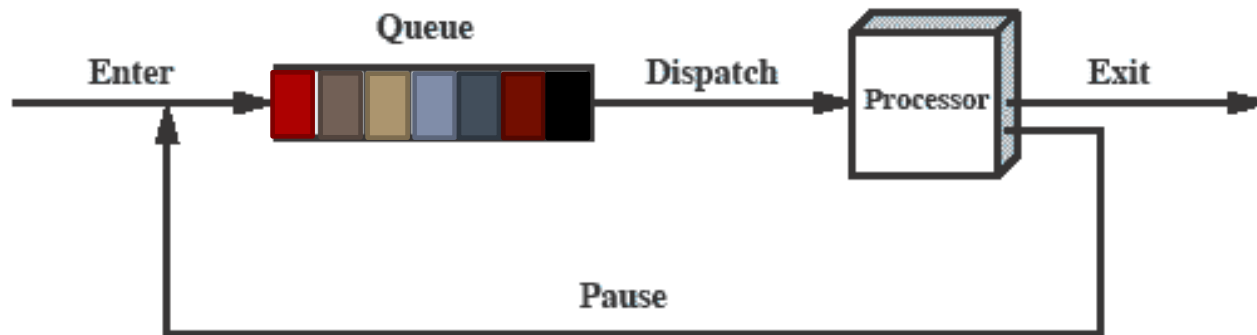
- Uniprocessor – only one really
 - Illusion of multiple processes
 - ... juggling
 - Multiprocessors – several programs at once
 - ... still requires juggling
 - Concurrency v parallelism
 - Parallel programming: use multiple processors to go faster e.g. weather forecast
 - Concurrency: play music while editing
-

Trace: CPU Point of View



Scheduler

- Queue of processes ready to run
 - Other processes waiting for an I/O
- Which one to run next?
 - Don't want the music to stop



(b) Queuing diagram

Summary

- Many user processes sharing the computer
 - Music and coding
 - Multiple users
 - OS has a scheduler to switch between processes
 - Processes are protected from each other
 - Ok if your program loops
-

Memory Management

Lots of processes in memory

Memory Management Problem

- How much memory?
 - Process may need more or less memory
 - Where in memory?
 - Processes come and go – where is space
 - Program must work in any location
 - Solutions
 - Paged memory
 - Address translation
-

Processes, Pages and Frames

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

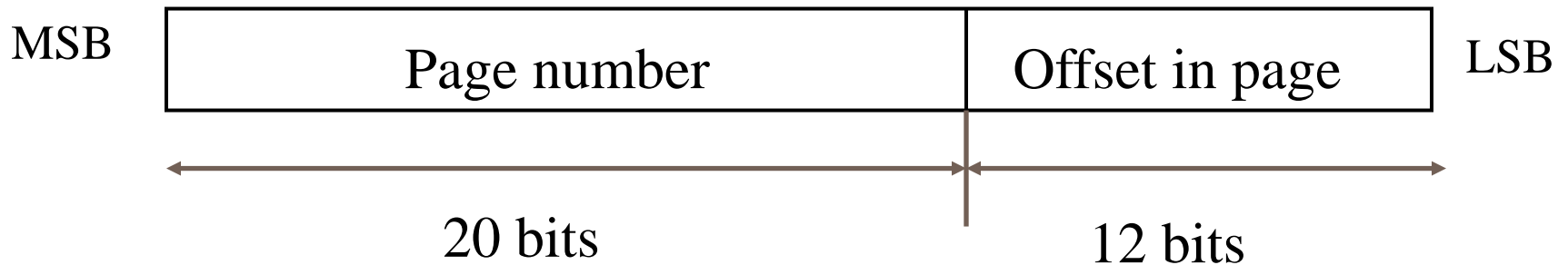
- Program memory divided into pages
- Computer memory divided into frames
- Pages allocated to frames
- **Page table**
 - List of frames used by each process

Paged Memory

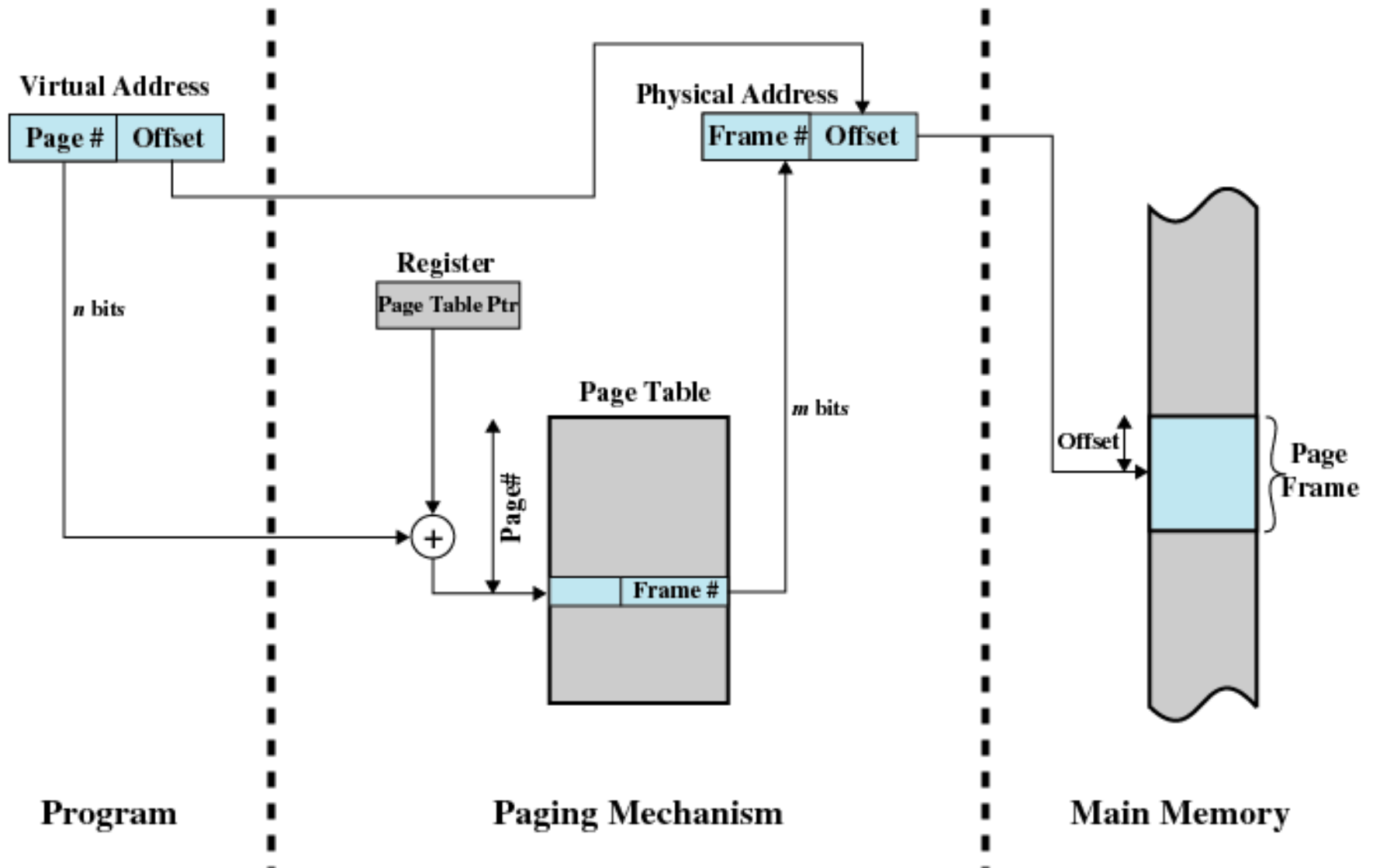
- Good for using memory efficiently
 - New problem
 - **Machine code addresses are all wrong!**
 - Solution
 - Translate address
 - Logical (virtual) address → Physical address
 - Each time memory is accessed
-

Address Translation

- Compiler assumes that program will run in its own computer
 - Logical addresses
- Each memory access translated
 - Memory management unit (MMU) in CPU
 - Replaces top bits of address: page → frame



Address Translation



Virtual Memory

- Not all pages in memory at all
 - Total number of pages may exceed available memory
 - Some pages temporarily stored in a page file on disk
-

Summary

- Program compiled to assuming own computer
 - Programs allocated memory in pages
 - Pages move around: on and off disk
 - Mechanisms
 - Page table – where is each page?
 - Address translation – page → frame
 - Virtual memory
-

Exercise 3.1 – 3.4

- Look at processes and memory using c9





Summary

Operating System Structure

- Principles
 - Operating system calls user programs
 - Kernel: control what runs
 - Library of system calls
 - Services
 - Kernel
 - Able to use all processor instruction & registers
-

Layer

7	System call handler					
6	File system 1		...		File system m	
5	Virtual memory					
4	Driver 1	Driver 2	...			Driver n
3	Threads, thread scheduling, thread synchronization					
2	Interrupt handling, context switching, MMU					
1	Hide the low-level hardware					

OS is about Illusions

- *Several programs are running simultaneously*
 - The computer switches from one to another
 - *My program has all the memory*
 - The memory is shared between programs
 - *Disk is organised in files*
 - The disk has blocks; the OS maps file names to blocks
 - *Storage devices work the same*
 - Files may be arranged differently on magnetic, flash and optical drives
 - **OS creates an ‘ideal’ computer from a real one**
-