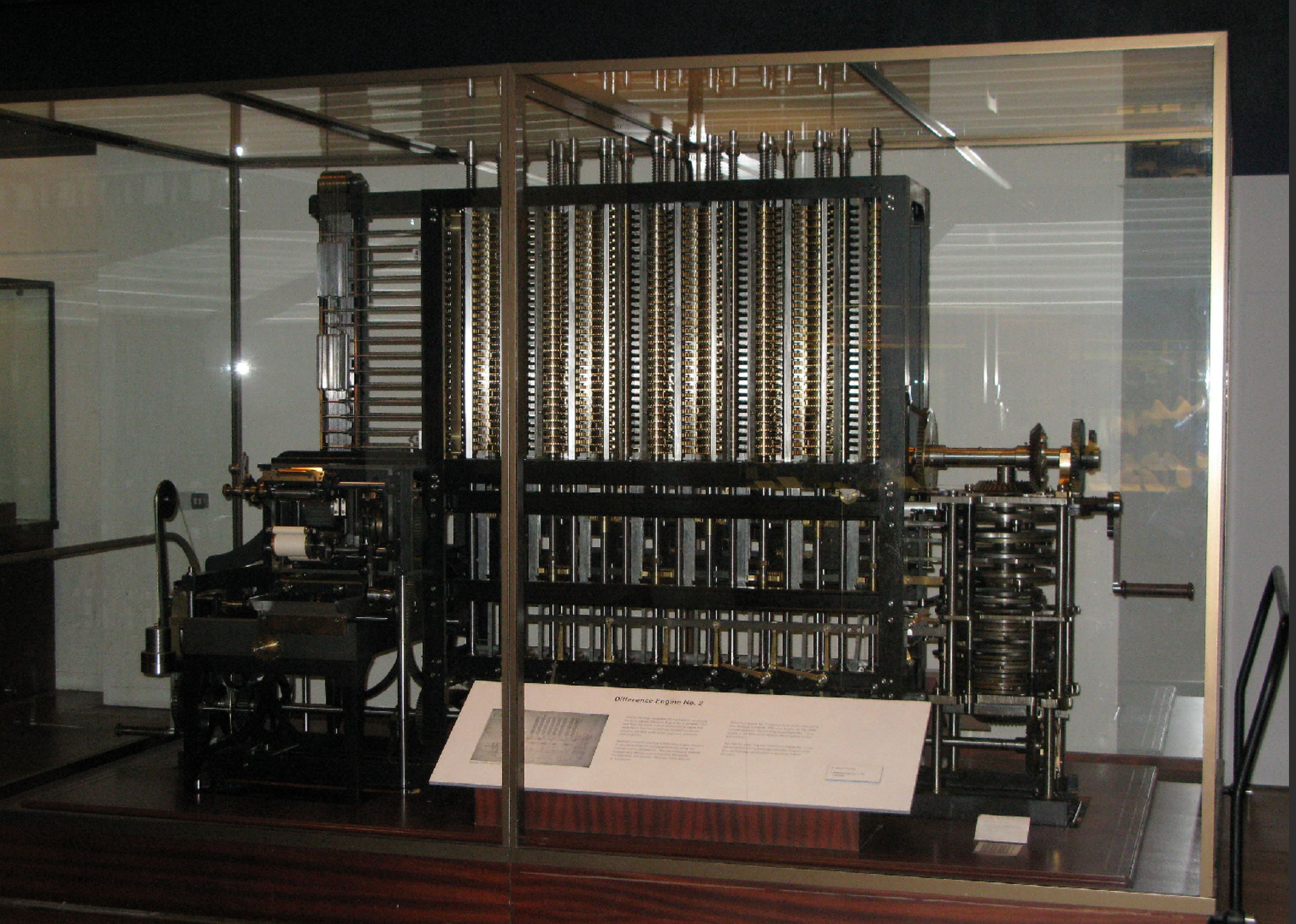
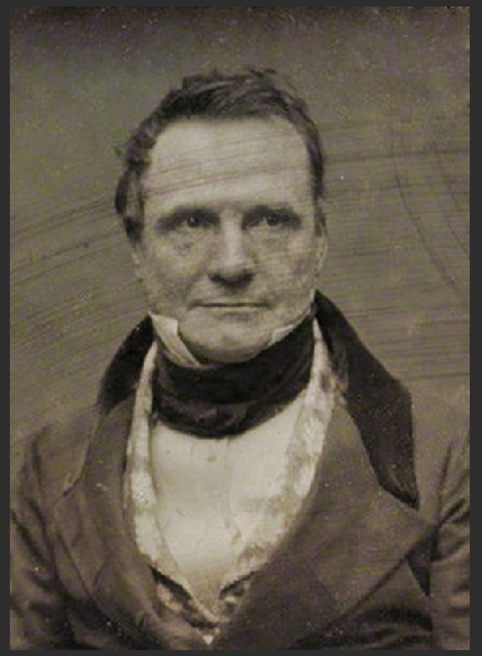
Introducción a Python

Cristián G. Sanchez & Carlos J. Ruestes

una
computadora
es una
he-rra-mien-ta

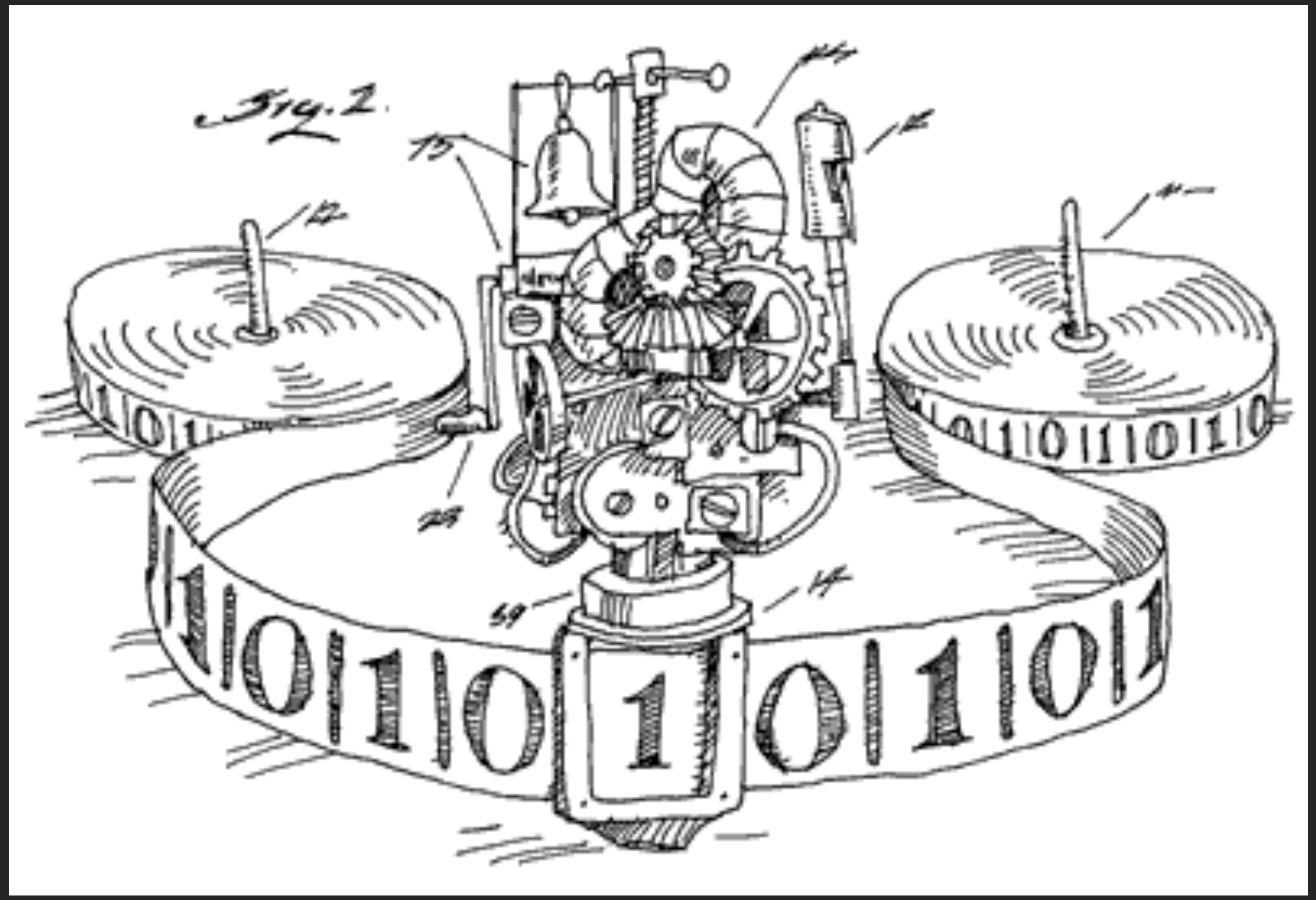




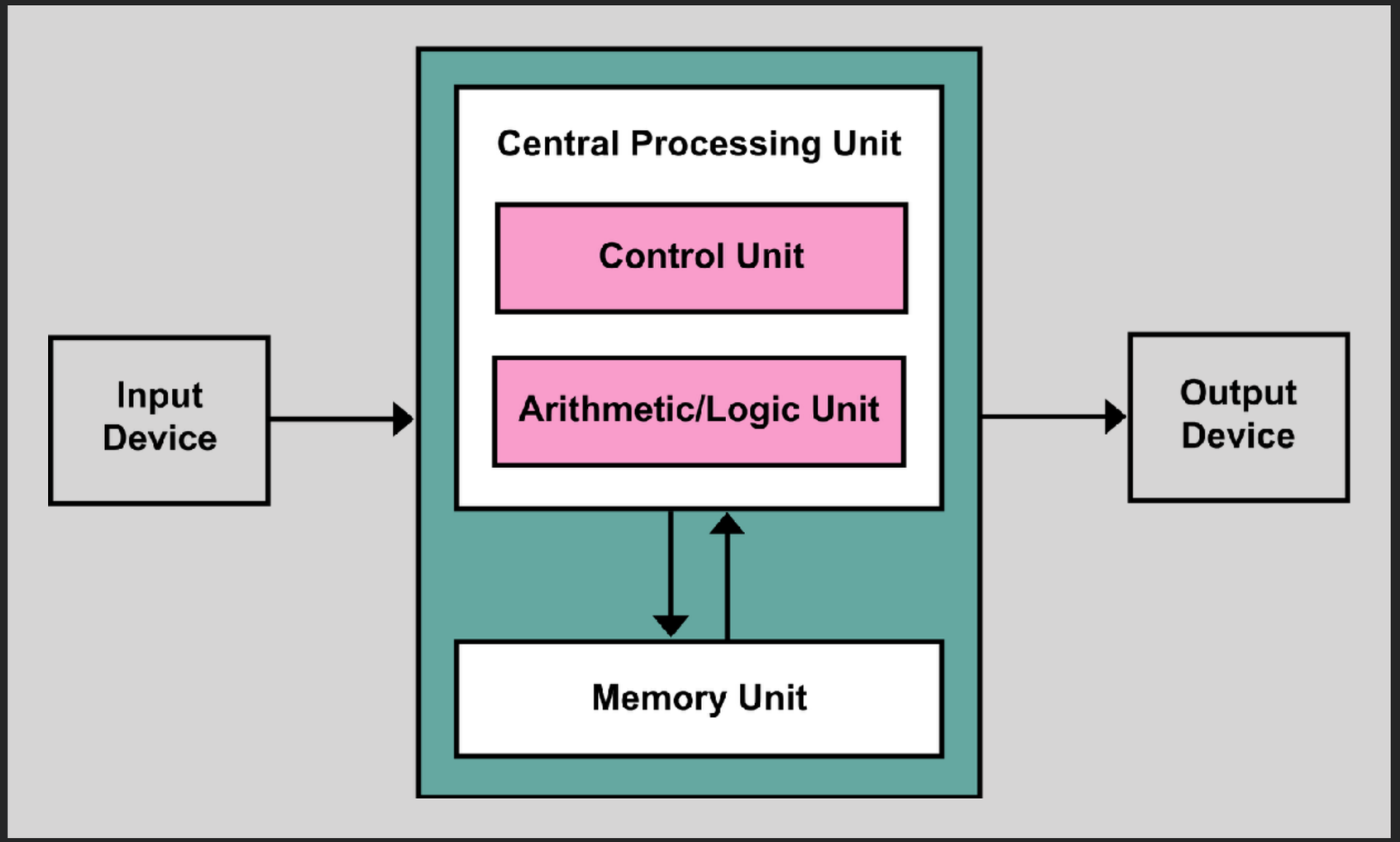
Charles Babbage
(1791-1871)

Alan Turing
(1912-1954)

Máquina de Turing
1937



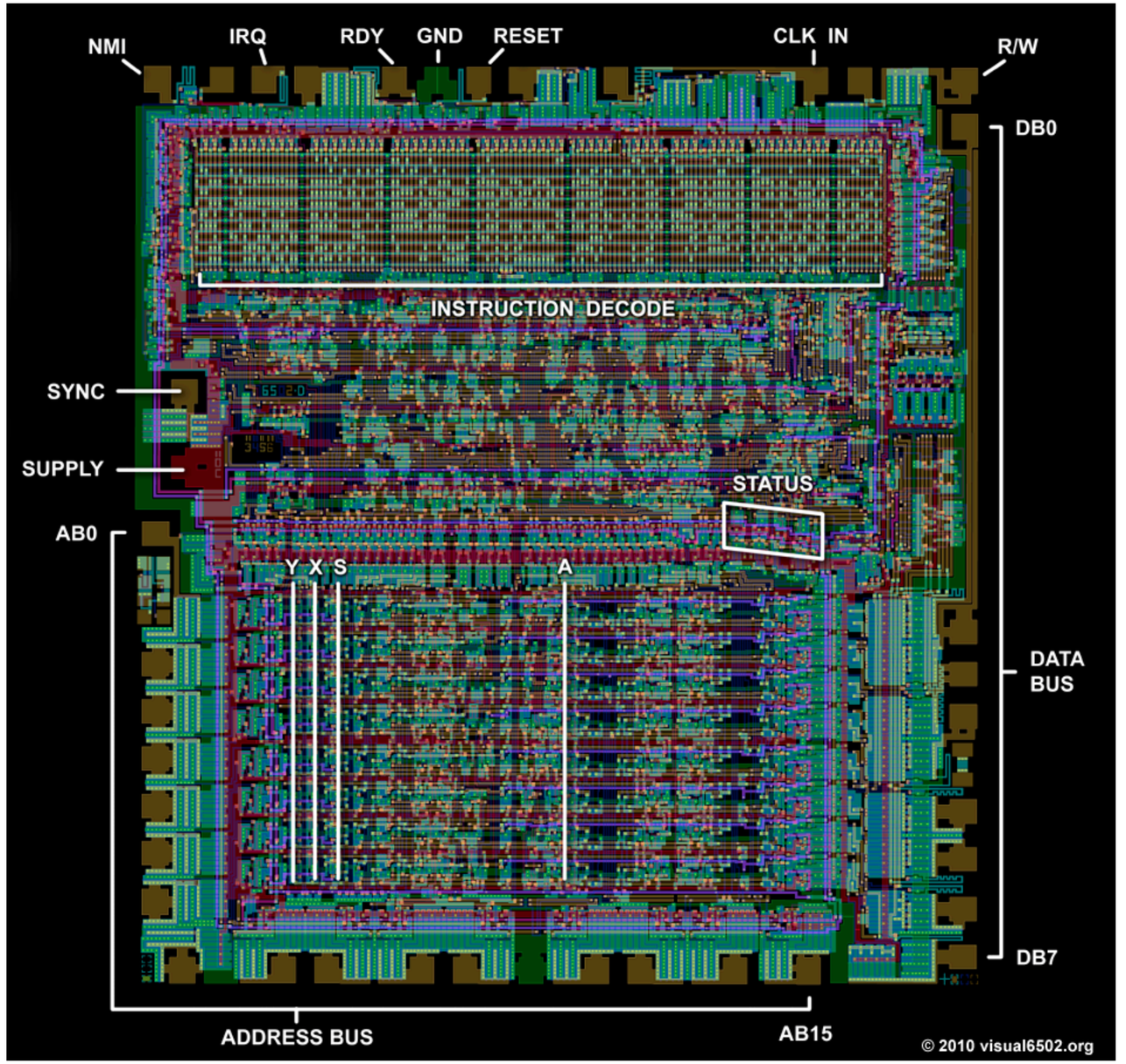

Alan Turing
the enigma
THE EXTRAORDINARY STORY OF THE BRILLIANT SCIENTIST WHOSE 'ENIGMA' DEVICE BROKE GERMANY'S MOST SECRET WWII CODE, WHO PIONEERED THE MODERN COMPUTER AGE AND WHO FINALLY FELL VICTIM TO THE COLD WAR WORLD OF MILITARY SECRETS AND
by Andrew Hodges
THE IMITATION GAME



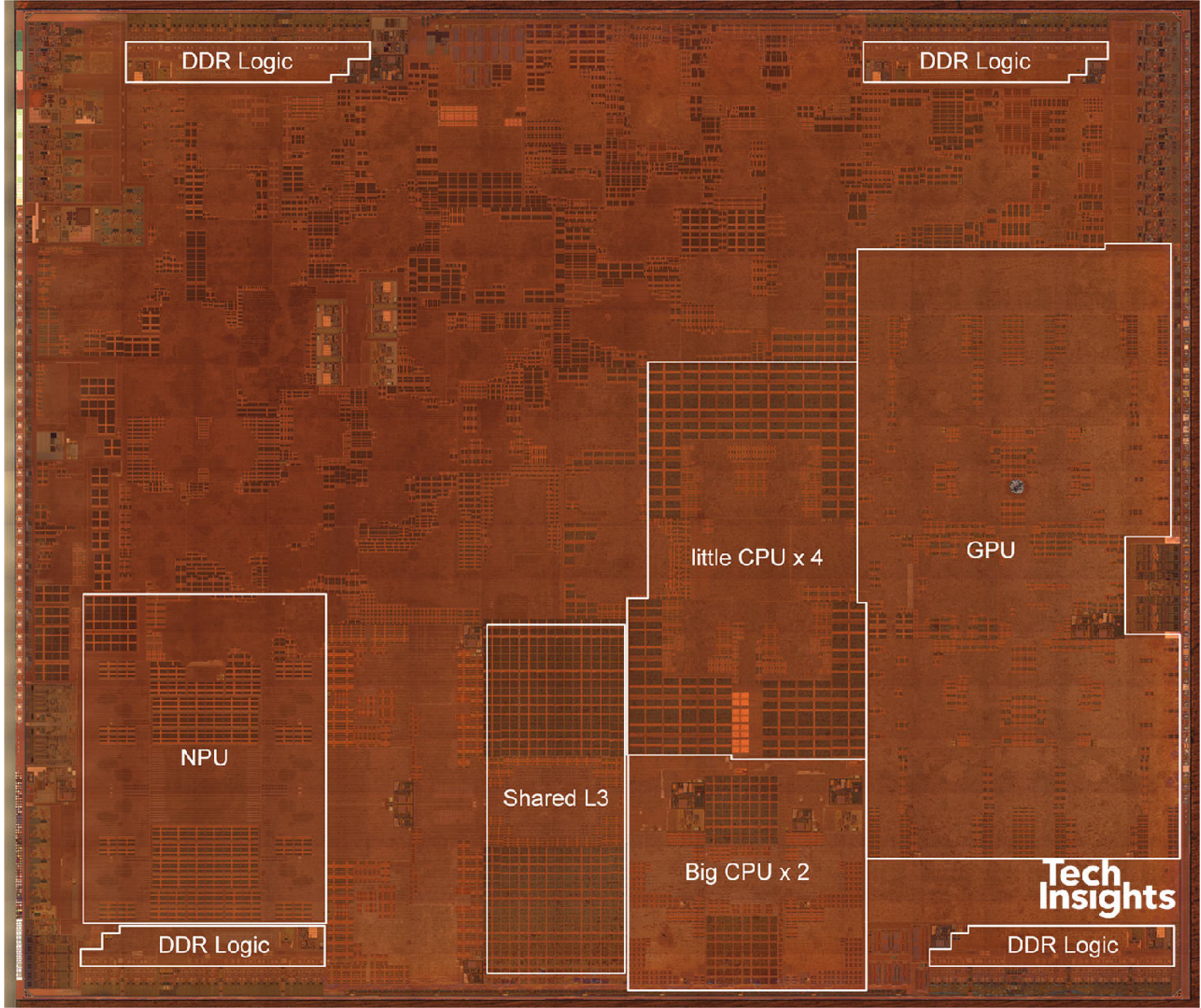
arquitectura de von Neumann

John von Neumann
(1903-1957)

4528 transistores



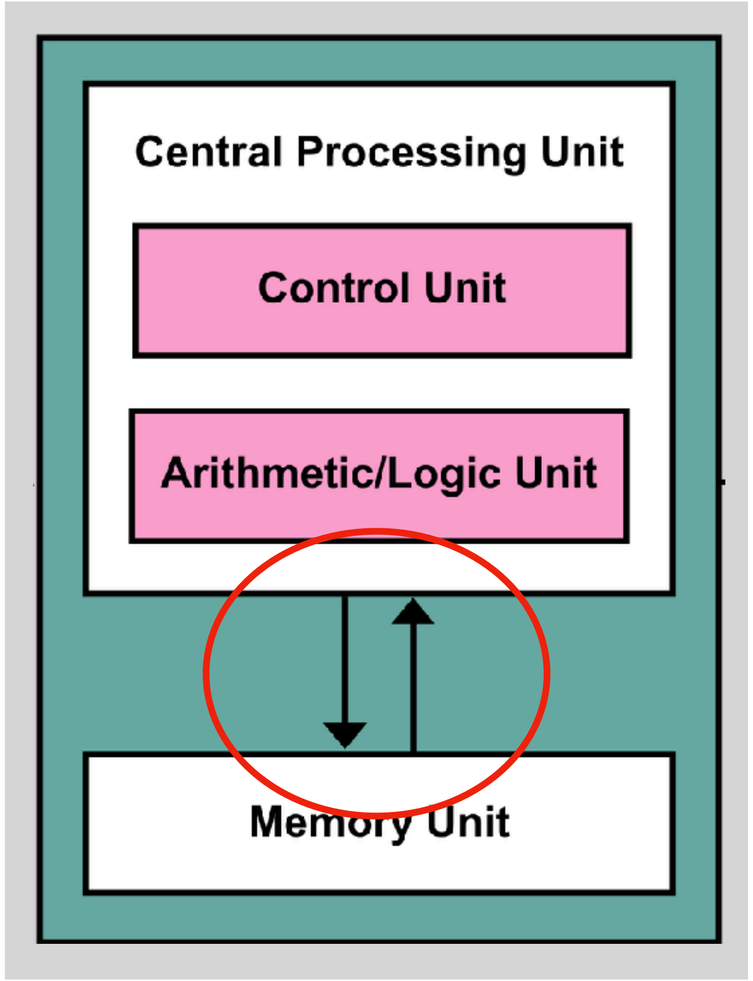
10.000.000.000
transistores



(intermezzo)

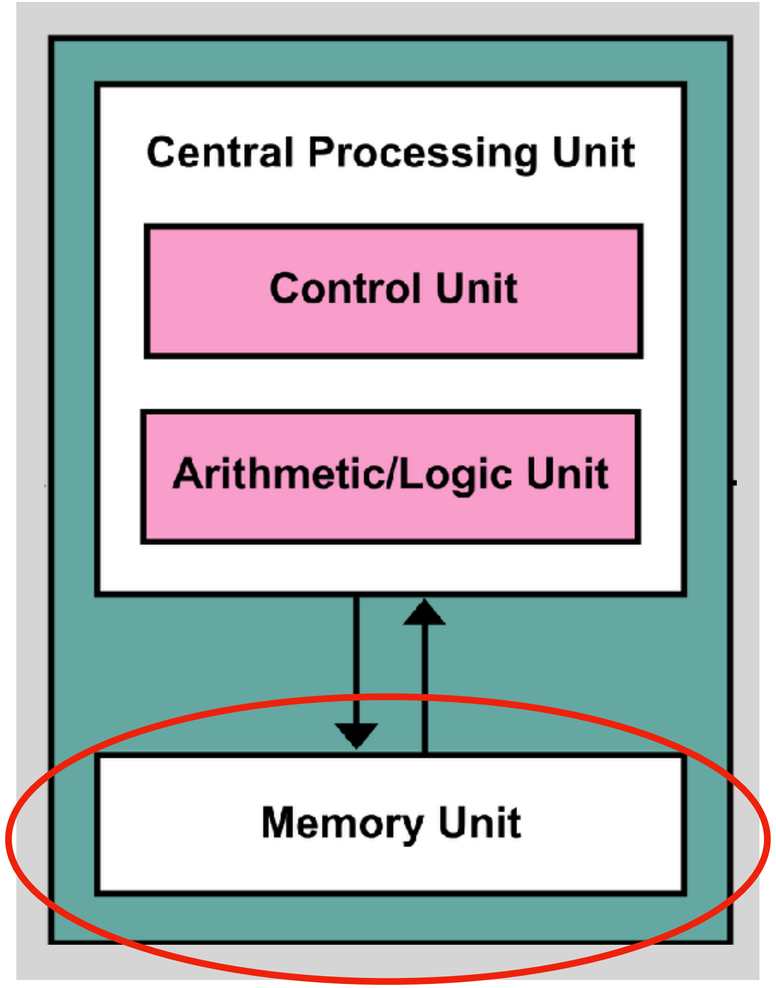
nuevos paradigmas

- concurrencia
- machine learning
- computación cuántica
- ...



BUS

cuello de botella
de
von Neuman



COMPUTER ARCHITECTURE

Byte-Addressable Memory

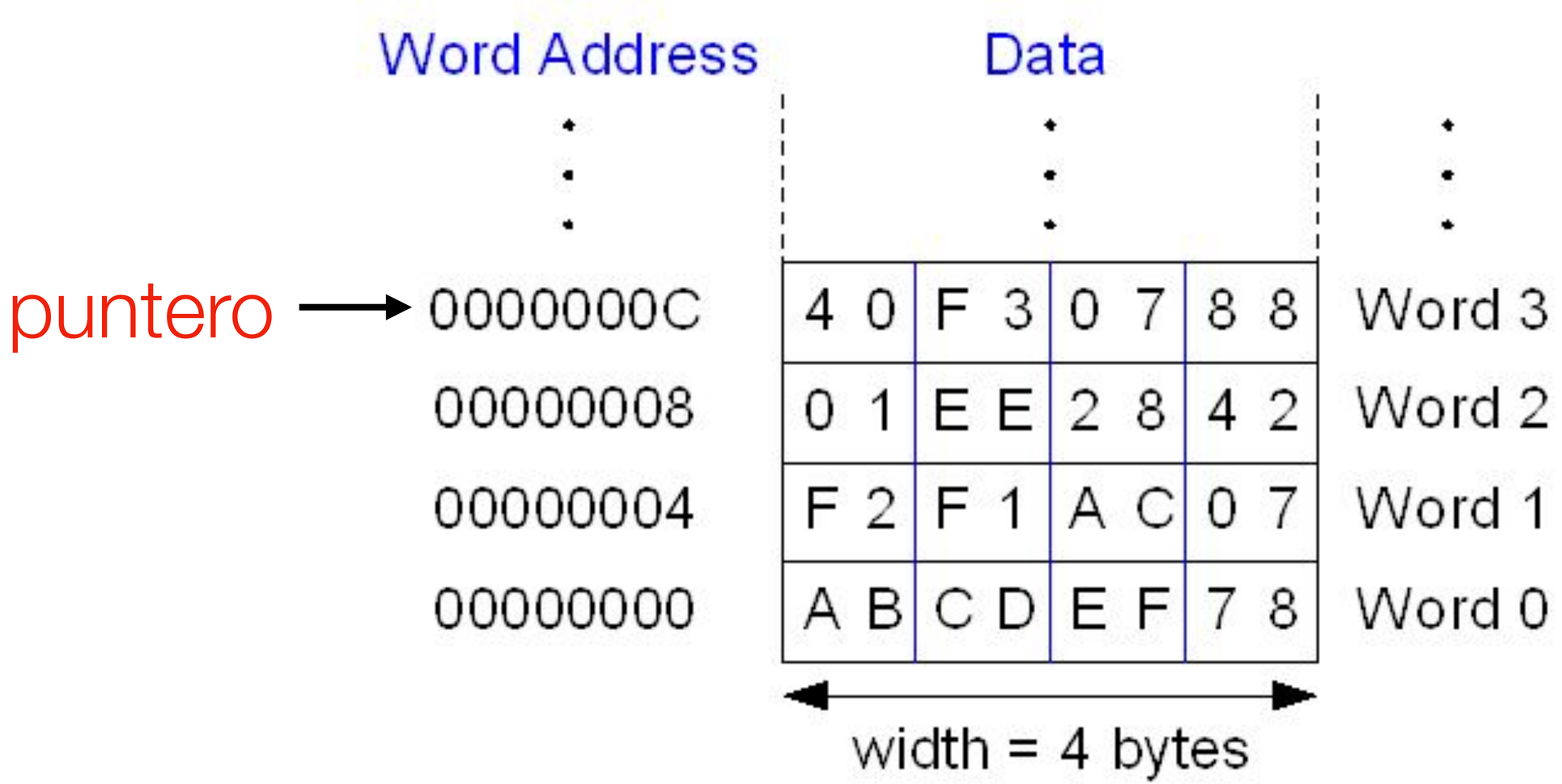
- Each data byte has unique address
- Load/store words or single bytes: load byte (`lb`) and store byte (`sb`)
- 32-bit word = 4 bytes, so word address increments by 4

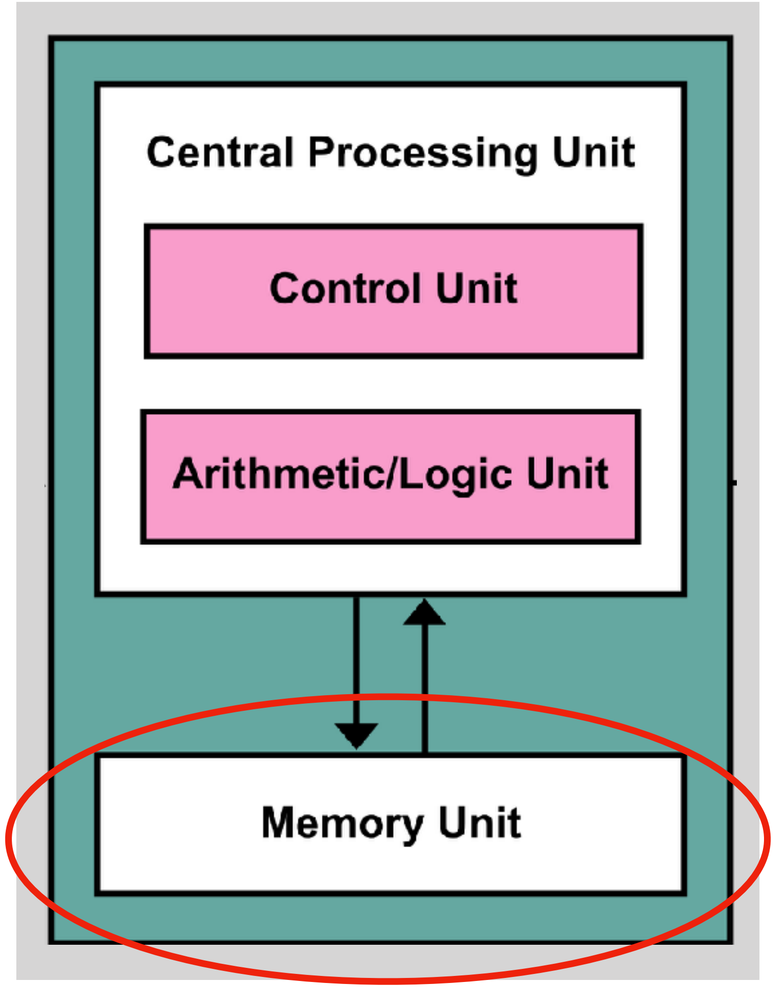
Word Address	Data	
0000000C	4 0 F 3 0 7 8 8	Word 3
00000008	0 1 E E 2 8 4 2	Word 2
00000004	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

width = 4 bytes

ELSEVIER

memoria direccionable


Word Address
Data
puntero
0000000C
00000008
00000004
00000000
4 0 F 3 0 7 8 8
0 1 E E 2 8 4 2
F 2 F 1 A C 0 7
A B C D E F 7 8
Word 3
Word 2
Word 1
Word 0
width = 4 bytes



COMPUTER ARCHITECTURE

Byte-Addressable Memory

- Each data byte has unique address
- Load/store words or single bytes: load byte (`lb`) and store byte (`sb`)
- 32-bit word = 4 bytes, so word address increments by 4

Word Address	Data	
0000000C	4 0 F 3 0 7 8 8	Word 3
00000008	0 1 E E 2 8 4 2	Word 2
00000004	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

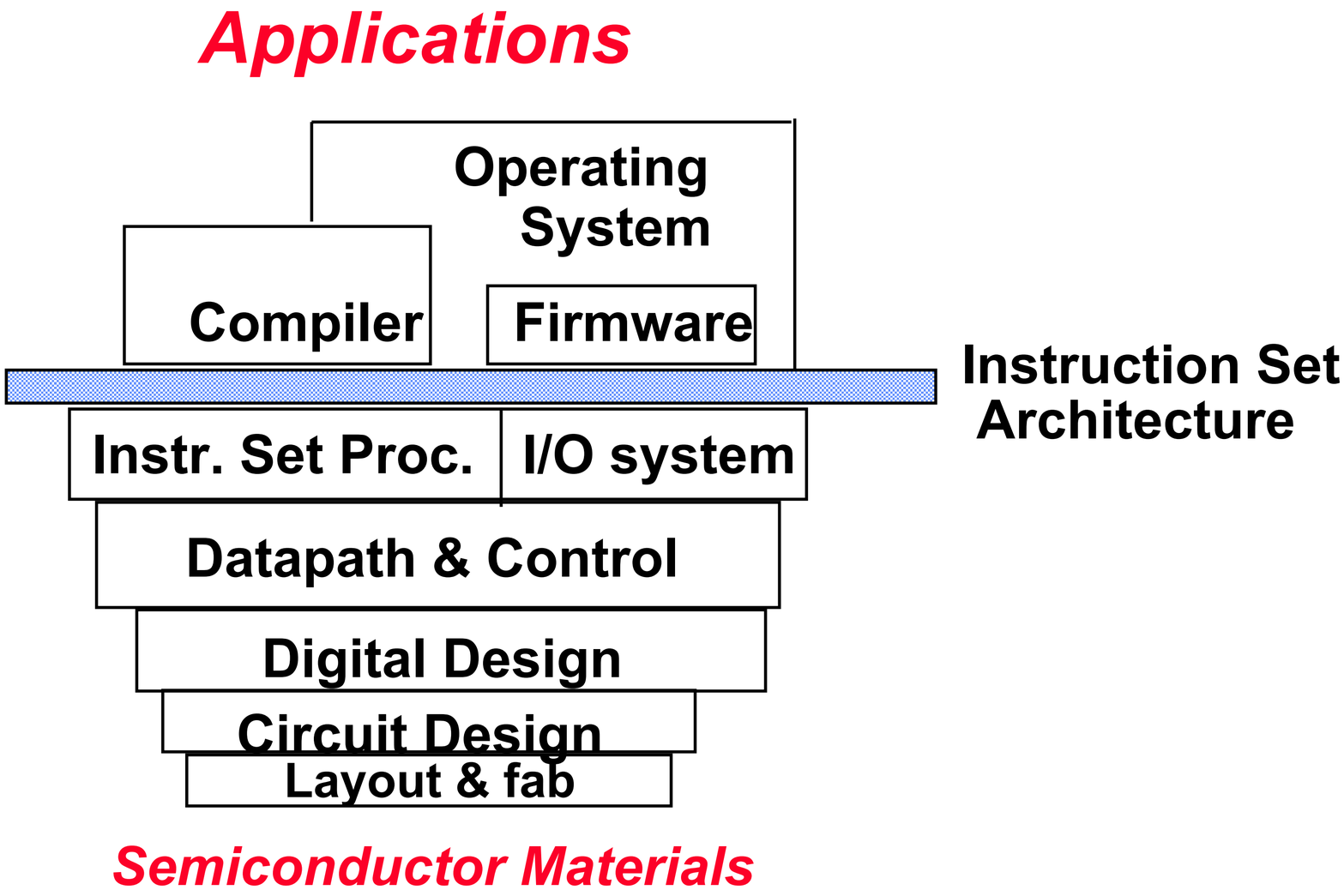
width = 4 bytes

ELSEVIER

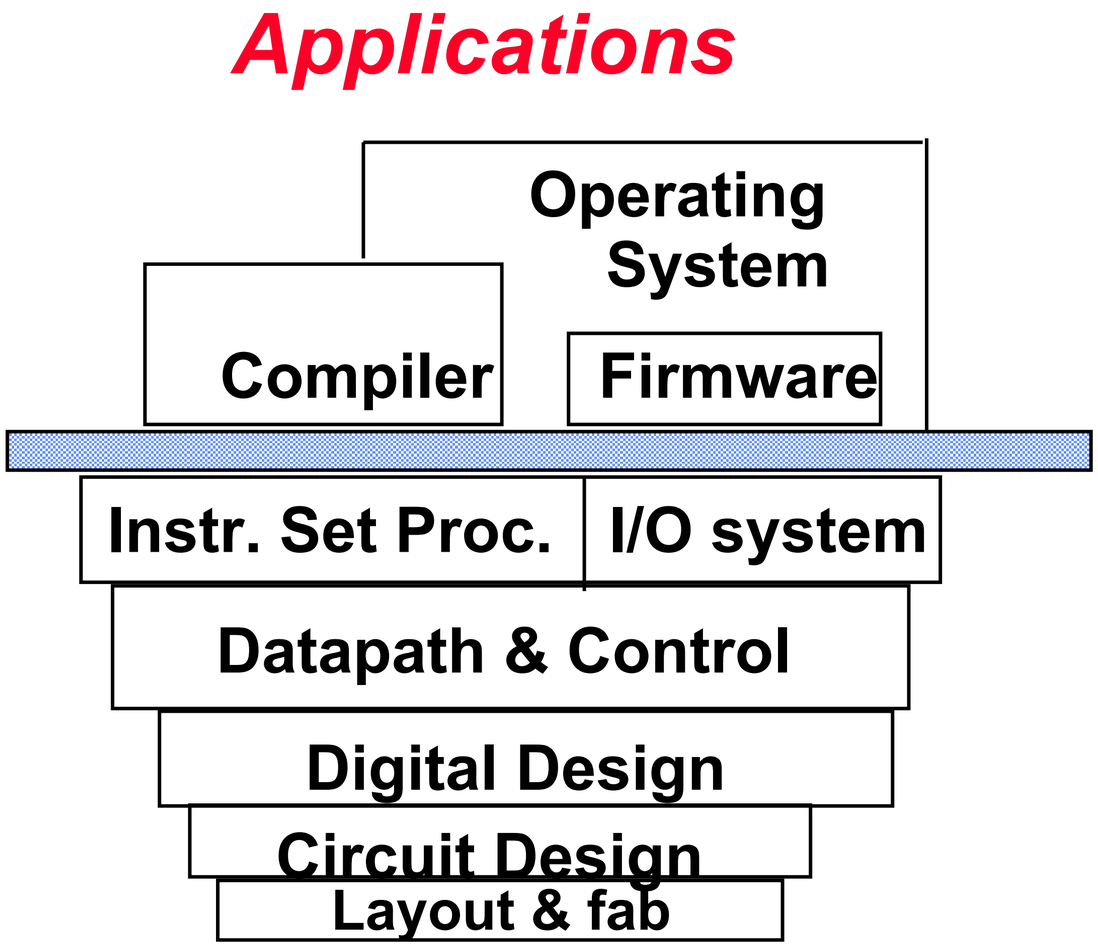
memoria direccionable

arquitectura de computadoras

arquitectura de computadoras



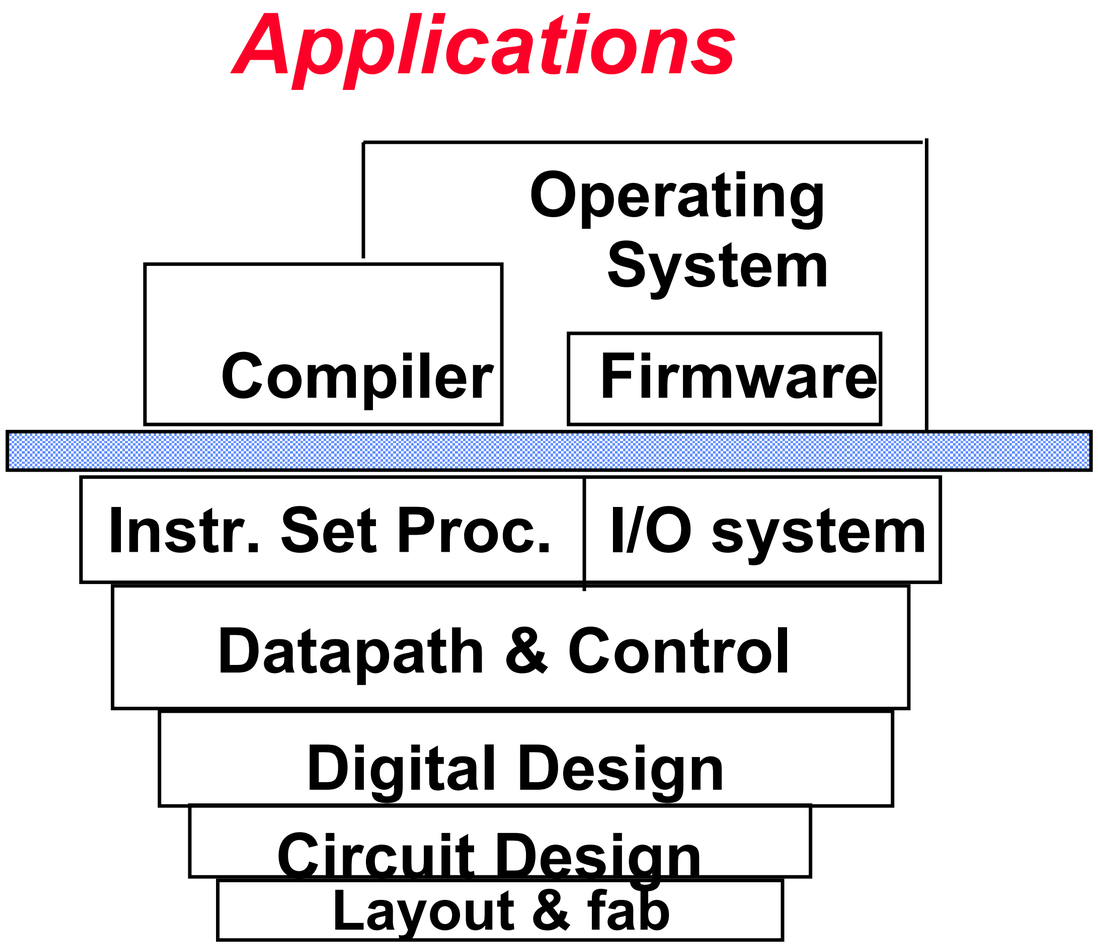
arquitectura de computadoras



Software

Instruction Set Architecture

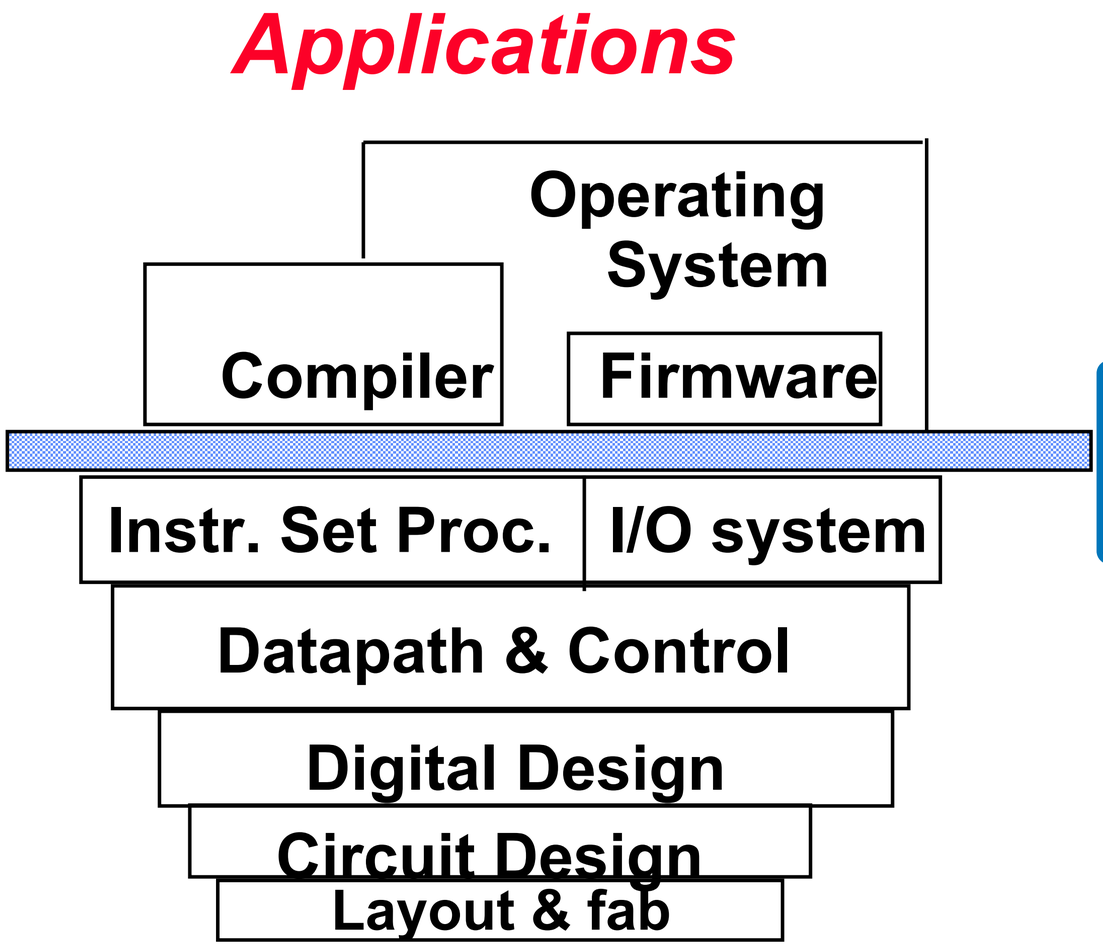
arquitectura de computadoras



Software

Hardware

arquitectura de computadoras



Software

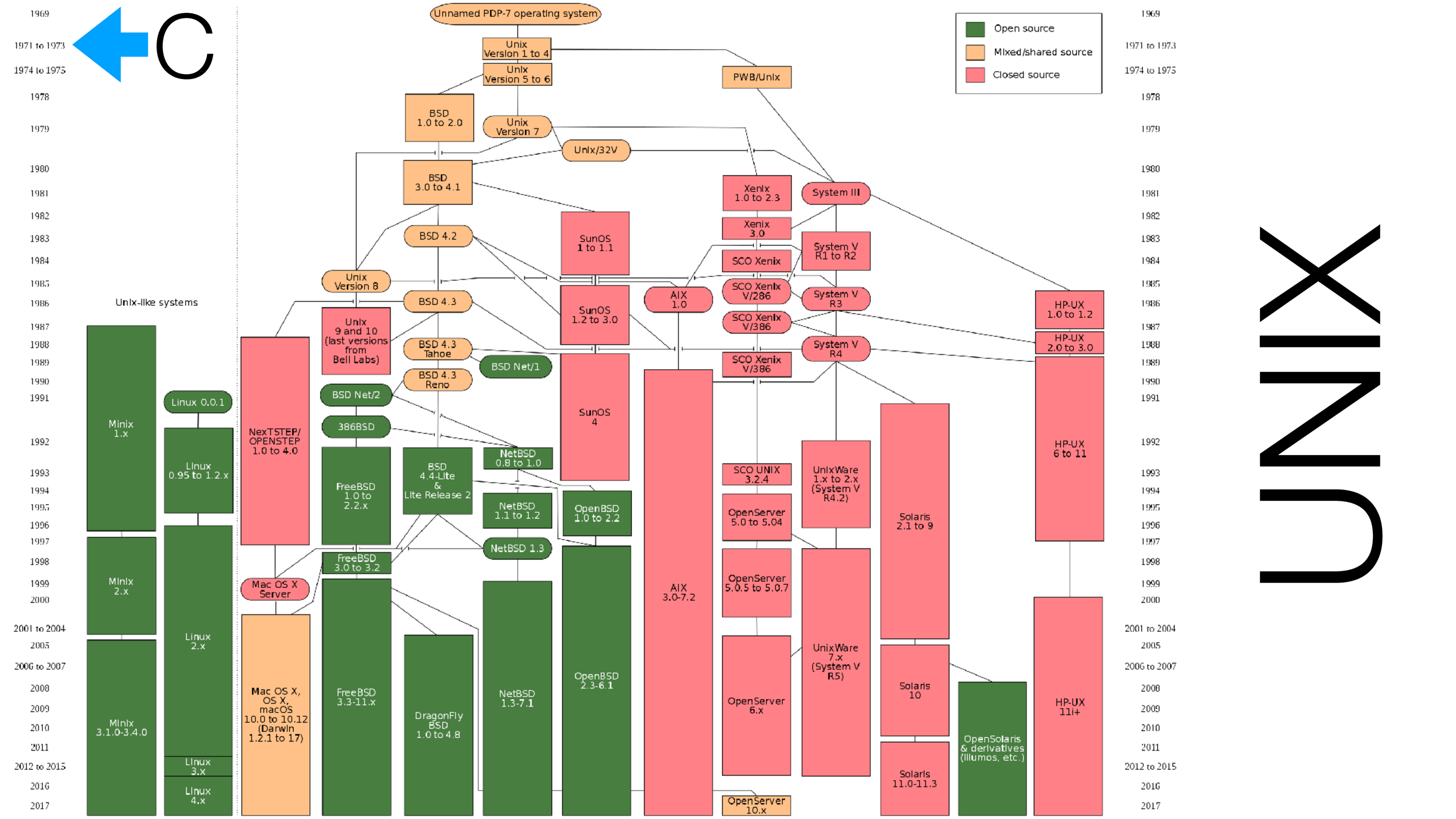
Instruction Set Architecture

Como el software “habla” con el hardware

Hardware

Dennis Ritchie

Ken Thompson


C
UNIX
Unix-like systems
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 2007
2008
2009
2010
2011
2012 to 2015
2016
2017
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
Unix Version 7
PWB/Unix
BSD 1.0 to 2.0
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
SunOS 1 to 1.1
System V R1 to R2
SCO Xenix
Unix Version 8
SCO Xenix V/286
AIX 1.0
System V R3
BSD 4.3
SunOS 1.2 to 3.0
HP-UX 1.0 to 1.2
SCO Xenix V/386
HP-UX 2.0 to 3.0
Unix 9 and 10 (last versions from Bell Labs)
BSD 4.3 Tahoe
BSD Net/1
System V R4
BSD 4.3 Reno
SCO Xenix V/386
Minix 1.x
Linux 0.0.1
BSD Net/2
NexTSTEP/ OPENSTEP 1.0 to 4.0
SunOS 4
386BSD
HP-UX 6 to 11
Linux 0.95 to 1.2.x
NetBSD 0.8 to 1.0
FreeBSD 1.0 to 2.2.x
BSD 4.4-Lite & Lite Release 2
SCO UNIX 3.2.4
UnixWare 1.x to 2.x (System V R4.2)
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
OpenServer 5.0 to 5.04
Solaris 2.1 to 9
NetBSD 1.3
FreeBSD 3.0 to 3.2
OpenServer 5.0.5 to 5.0.7
Minix 2.x
Mac OS X Server
AIX 3.0-7.2
Linux 2.x
Mac OS X, OS X, macOS 10.0 to 10.12 (Darwin 1.2.1 to 17)
FreeBSD 3.3-11.x
DragonFly BSD 1.0 to 4.8
NetBSD 1.3-7.1
OpenBSD 2.3-6.1
OpenServer 6.x
UnixWare 7.x (System V R5)
Solaris 10
OpenSolaris & derivatives (Illumos, etc.)
HP-UX 11i+
Minix 3.1.0-3.4.0
Linux 3.x
Solaris 11.0-11.3
Linux 4.x
OpenServer 10.x

1978

THE

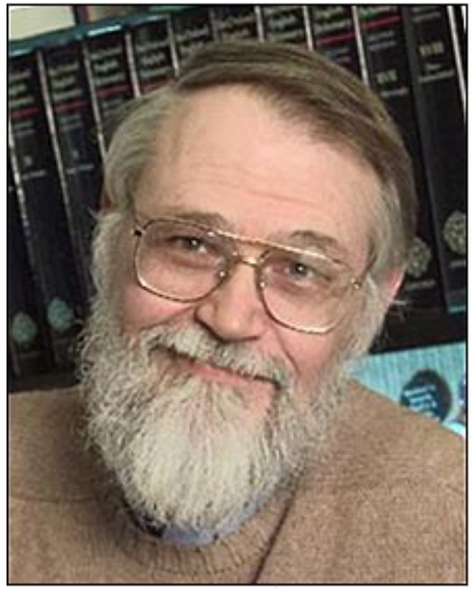
C

PROGRAMMING
LANGUAGE

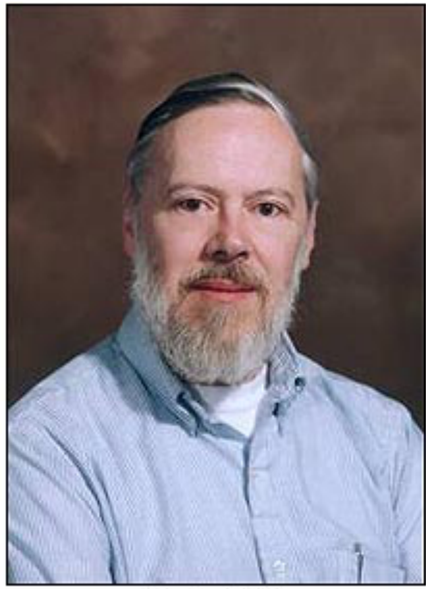
Brian W. Kernighan • Dennis M. Ritchie

PRENTICE-HALL SOFTWARE SERIES

Ken Thompson

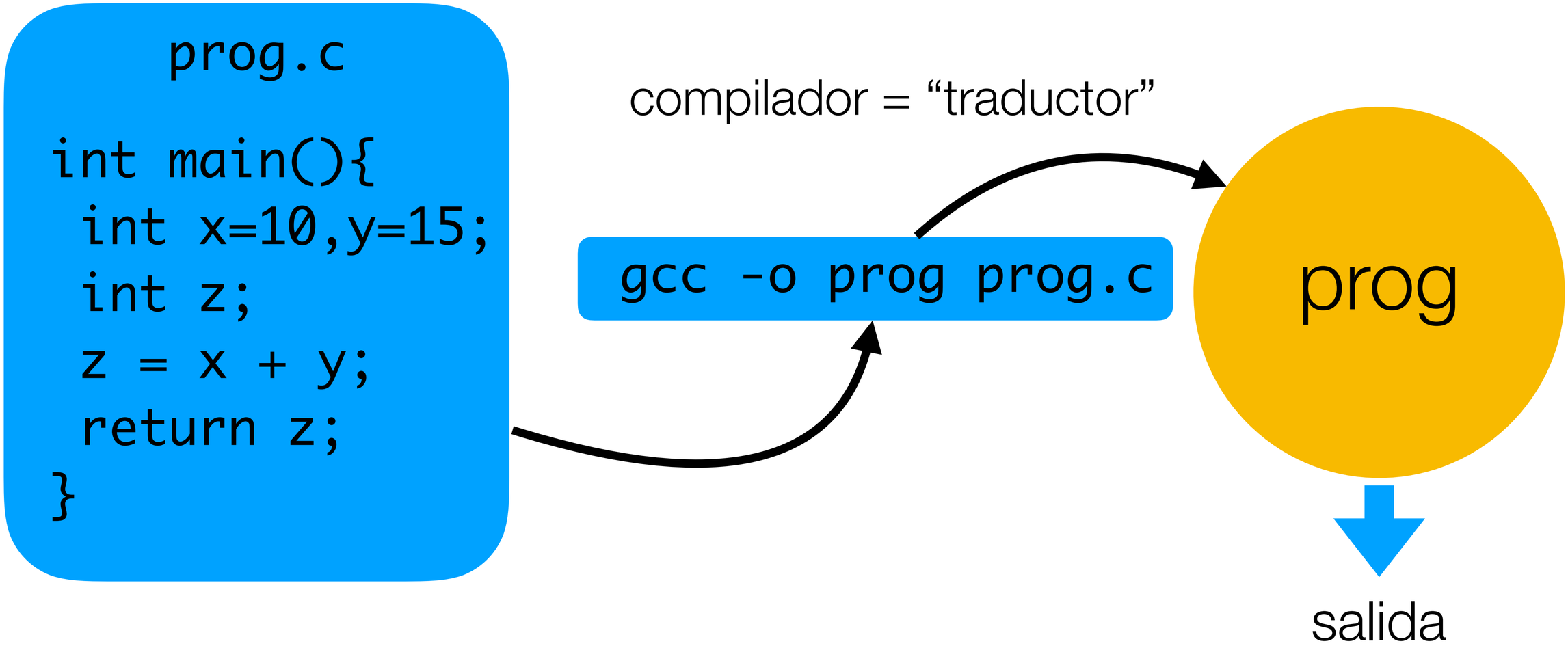


Brian Kernighan



Dennis Ritchie

C es un lenguaje “compilado”



C es un lenguaje “compilado”

```
CGSMBP:ejemplo_asm cgs$ cat prog.c
int main(){
 int x=10,y=15;
 int z;
 z = x + y;
 return z;
}
CGSMBP:ejemplo_asm cgs$ gcc -o prog ./prog.c
CGSMBP:ejemplo_asm cgs$ ./prog
CGSMBP:ejemplo_asm cgs$ echo $?
25
```

```
int main(){
 int x=10,y=15;
 int z;
 z = x + y;
 return z;
}
```

C

```c
int main(){
 int x=10,y=15;
 int z;
 z = x + y;
 return z;
}
```

assembler

```
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 10, 14    sdk_version 10, 14
        .globl  _main                   ## -- Begin function main
        .p2align        4, 0x90
_main:                                  ## @main
        .cfi_startproc
## %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        movl    $0, -4(%rbp)
        movl    $10, -8(%rbp)
        movl    $15, -12(%rbp)
        movl    -8(%rbp), %eax
        addl    -12(%rbp), %eax
        movl    %eax, -16(%rbp)
        movl    -16(%rbp), %eax
        popq    %rbp
        retq
        .cfi_endproc
```

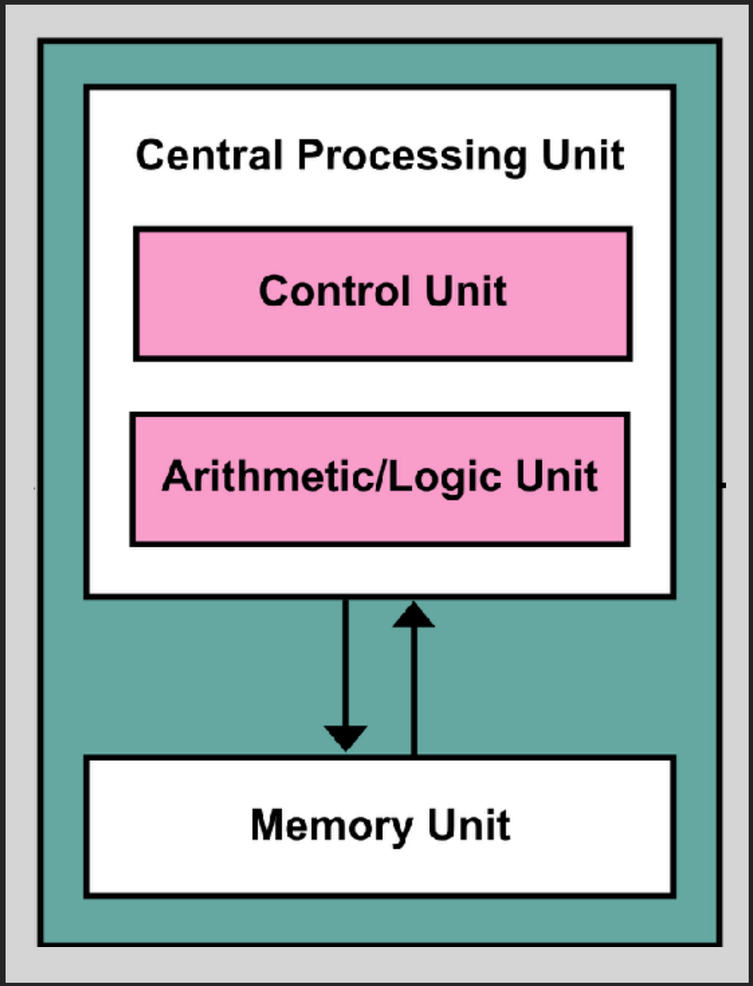
C

```c
int main(){
 int x=10,y=15;
 int z;
 z = x * y;
 return z;
}
```

assembler

```
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 10, 14    sdk_version 10, 14
        .globl  _main                   ## -- Begin function main
        .p2align        4, 0x90
_main:                                  ## @main
        .cfi_startproc
## %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        movl    $0, -4(%rbp)
        movl    $10, -8(%rbp)
        movl    $15, -12(%rbp)
        movl    -8(%rbp), %eax
        imull   -12(%rbp), %eax
        movl    %eax, -16(%rbp)
        movl    -16(%rbp), %eax
        popq    %rbp
        retq
        .cfi_endproc
```

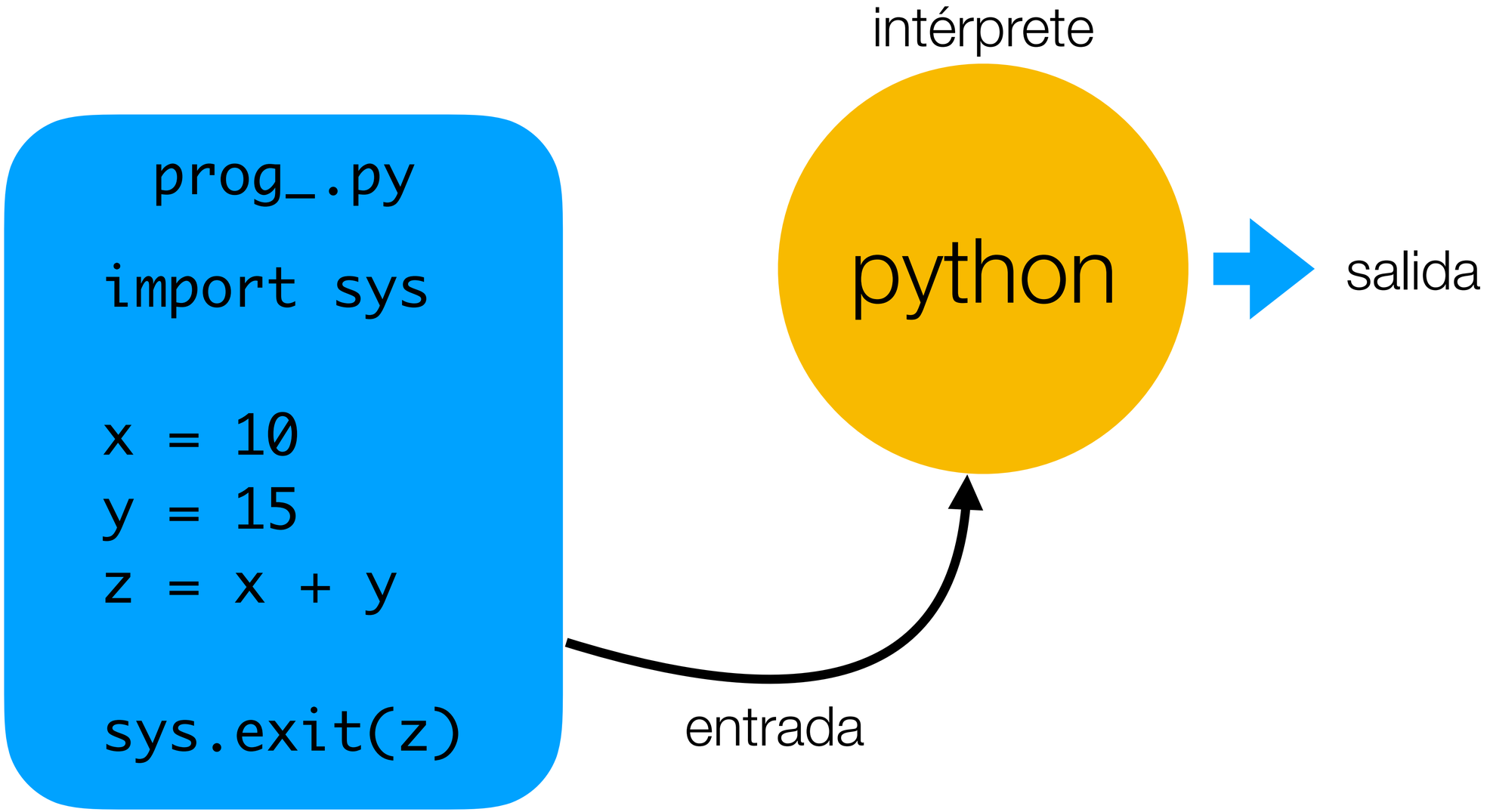
assembler




```
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 10, 14    sdk_version 10, 14
        .globl  _main                   ## -- Begin function main
        .p2align        4, 0x90
_main:                                  ## @main
        .cfi_startproc
## %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        movl    $0, -4(%rbp)
        movl    $10, -8(%rbp)
        movl    $15, -12(%rbp)
        movl    -8(%rbp), %eax
        imull   -12(%rbp), %eax
        movl    %eax, -16(%rbp)
        movl    -16(%rbp), %eax
        popq    %rbp
        retq
        .cfi_endproc
```

Python es un lenguaje “interpretado”

```
CGSMBP:ejemplo_asm cgs$ cat prog_.py
import sys

x = 10
y = 15
z = x + y

sys.exit(z)
CGSMBP:ejemplo_asm cgs$ python prog_.py
CGSMBP:ejemplo_asm cgs$ echo $?
25
```


intérprete
prog_.py
import sys
x = 10
y = 15
z = x + y
sys.exit(z)
python
salida
entrada

Python

1	enteros de precisión ilimitada
1.0, 1.0e-5	float = “C double” = IEEE 754 64 bit float
1.0+2j	complejo = par ordenado de floats
True, False	Booleano, representado con un entero 0 o 1
…	dict, set, list, strings, … ,user defined objects

0.1 + 0.2 = ???

https://0.30000000000000004.com

en Python

todo

es un objeto

Operadores Numéricos

`X + Y`	suma
`X - Y`	resta
`X * Y`	producto
`X / Y`	división
`X // Y`	división entera
`X % Y`	resto de la división entera
`-X`	negativo de X
`+X`	identidad
`X | Y`	operación “o” a nivel de bits
`X & Y`	operación “y” a nivel de bits
`X ^ Y`	operación “o exclusivo” a nivel de bits
`X << N`	mover bits a la izquierda N lugares
`X >> N`	mover bits a la derecha N lugares
`~ X`	inversión a nivel de bits
`X ** Y`	potencia
`abs(X)`	valor absoluto
`int(X)`	parte entera
`float(X)`	conversión a punto flotante
`complex(X) , complex(X,Y)`	conversión a complejo

Operadores Lógicos

el resultado es siempre `True` o `False`

`X < Y`	es menor a
`X <= Y`	es menor o igual a
`X > Y`	es mayor que
`X >= Y`	es mayor o igual a
`X == Y`	es igual a
`X != Y`	es distinto a
`X is Y`	es el mismo objeto
`X is not Y`	no es el mismo objeto
`X < Y < Z`	comparaciones encadenadas
`not X`	negación lógica
`X or Y`	“o” lógico
`X and Y`	“y” lógico

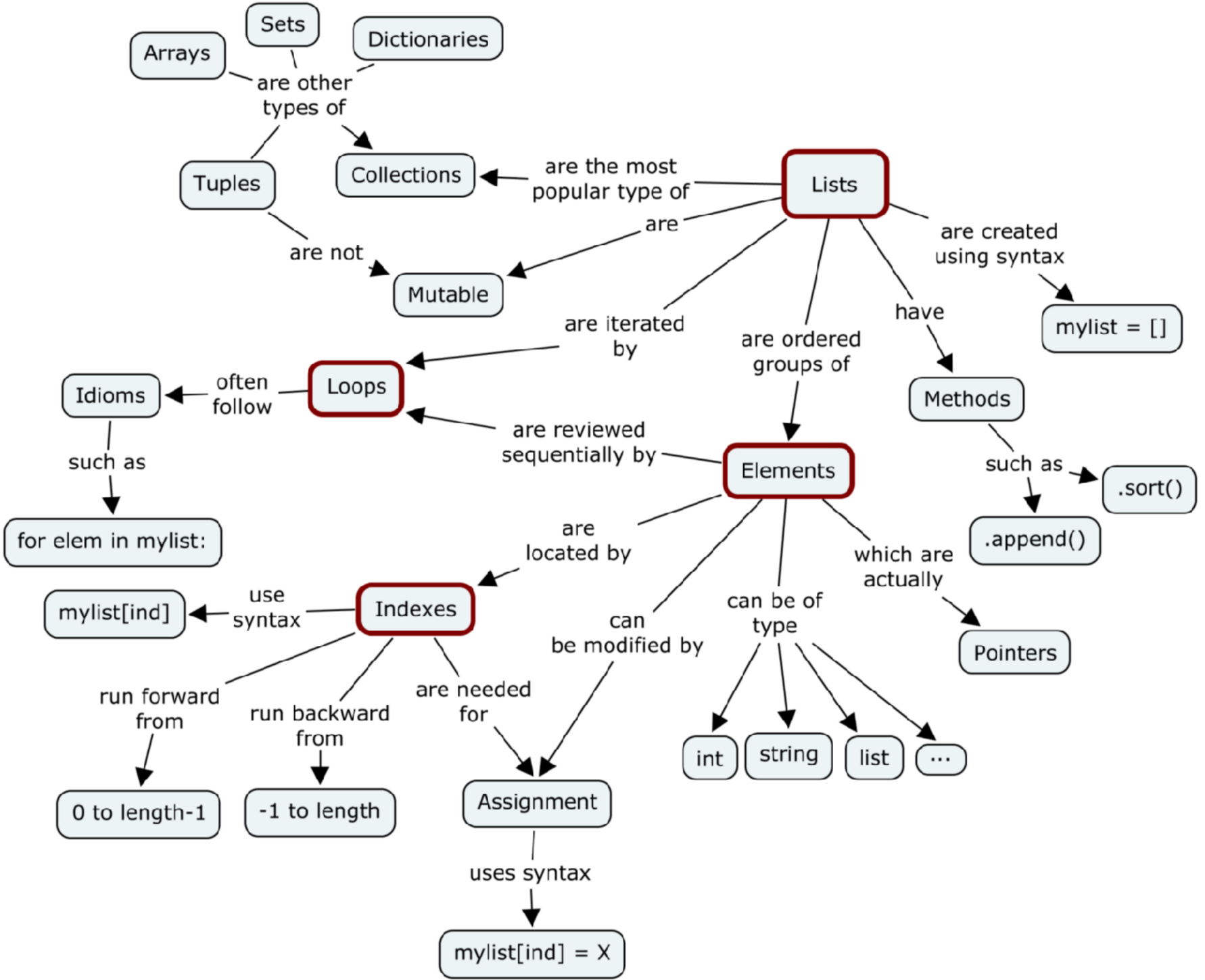
Strings (cadenas de caracteres)

```
"esto es un string"

'esto es un string'

“””esto
   es
   un string”””
```

listas

```
[1.0,1,True,”esto”,[1,2],u”🙋”]
```

son iterables!!!!!


Sets
Arrays
Dictionaries
are other types of
Tuples
Collections
are the most popular type of
Lists
are
are not
Mutable
are created using syntax
mylist = []
are iterated by
are ordered groups of
have
Loops
Idioms
often follow
Methods
are reviewed sequentially by
Elements
such as
for elem in mylist:
such as
.sort()
.append()
are located by
which are actually
Pointers
mylist[ind]
use syntax
Indexes
can be modified by
can be of type
run forward from
run backward from
are needed for
0 to length-1
-1 to length
Assignment
int
string
list
...
uses syntax
mylist[ind] = X

variables

```
a = 1.0
```

```
b = [1.0,2,True]
```

```
c = complex(1.0,2)
```

mutable o inmutable?

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

1

= la asignación
iguala punteros

hacer ejemplo de id()

funciones

```
def func(x):
    return x**2
```

indentación es sintaxis!!!!

```
def func(x):
    return x**2
```

```
while test:
    suite
[else:
    suite]
```

```
if test:
    suite
[elif test:
    suite]
[else:
    suite]
```

```
for target in iterable:
    suite
[else:
    suite]
```

bucles

```
for target in iterable:
    suite
[else:
    suite]
```

```
while test:
    suite
[else:
    suite]
```

bucles

`break` interrumpe inmediatamente el bucle

`continue` pasa inmediatamente a la siguiente iteración

condicionales

```
if test:
    suite
[elif test:
    suite]
[else:
    suite]
```

Crear un nuevo “notebook”.

Implementar en una función
el método de Newton-Raphson
para encontrar la raíz más cercana a una dada conjetura
para una función arbitraria.

La función debe devolver el valor de la raíz encontrada
y el número de iteraciones que se hicieron
y debe incorporar un control de divergencia.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$