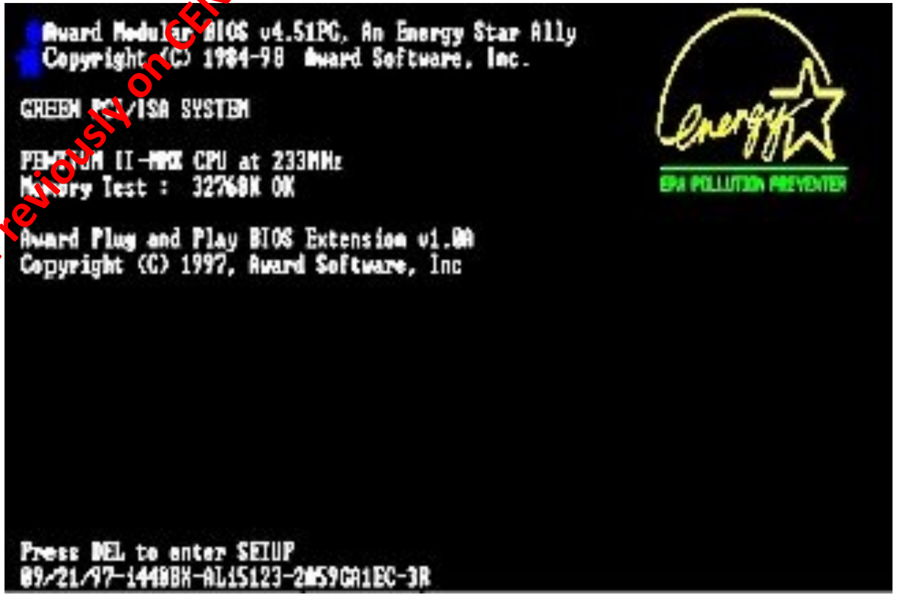
Ceng 111 – Fall 2021
Week 5b

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Previously on CENG111!



```
Award Modular BIOS v4.51PG, An Energy Star Ally
Copyright (C) 1984-98 Award Software, Inc.

GREEN PCI/ISA SYSTEM

PENTIUM II-MMX CPU at 233MHz
Memory Test :  32768K OK

Award Plug and Play BIOS Extension v1.0A
Copyright (C) 1997, Award Software, Inc



Press DEL to enter SETUP
09/21/97-i440BX-AL15123-2A59GA1EC-3R
```

I have been doing it, but it's still not working!
Re-boot
search ID: dren114
© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com

What happens after that power button

BOOTING YOUR COMPUTER

For more details, see e.g.: https://neosmart.net/wiki/mbr-boot-process/

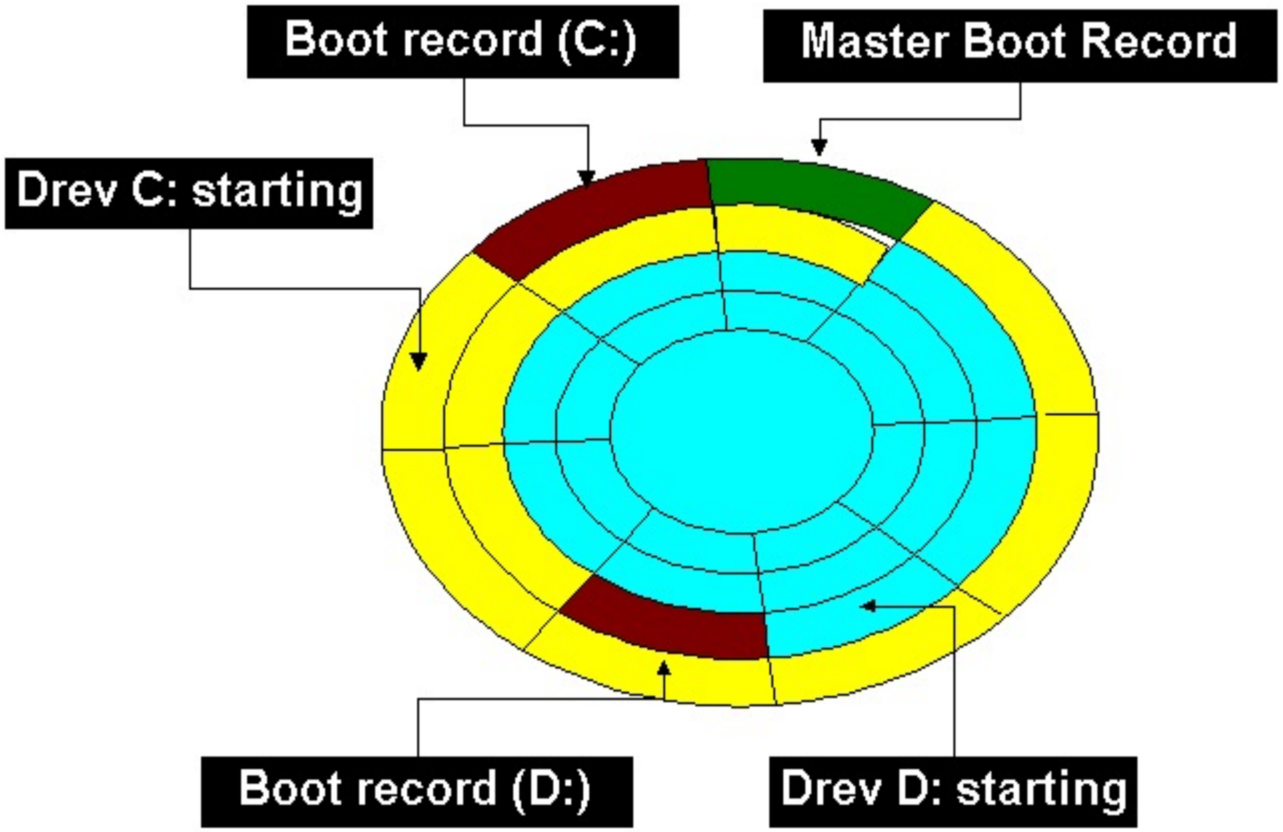
Previously on CENG111!

Booting the Computer

- BIOS (Basic Input-Output System)
  - BIOS is on a Read-Only Memory (ROM)
  - When starting, CPU addresses the address space of the ROM and starts executing the BIOS.
  - BIOS first initiates self-check of hardware (Power-on self test – POST)
  - BIOS initiates the memory and loads the rest of the BIOS into the RAM (because ROM is slow)
  - Then, it goes over non-volatile storage devices to find something bootable.
- Boot Device & MBR
  - BIOS loads MBR from the boot device
  - MBR is not OS specific
  - MBR checks for a partition to boot
  - Loads OS (starts OS kernel)

Previously on CENG111!

Master Boot Record



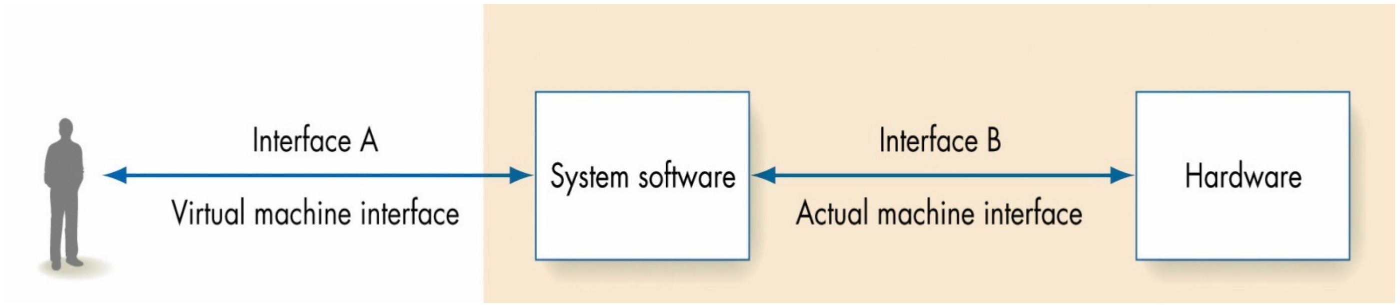
METU Computer Engineering

Previously on CENG111!

UEFI & GPT

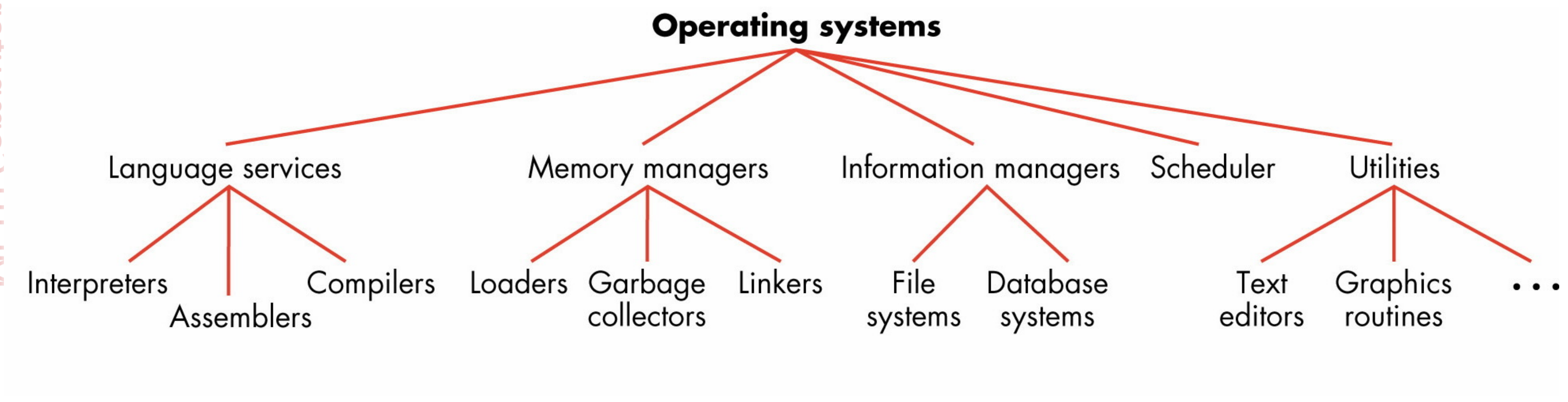
- Unified Extensible Firmware Interface
  - Extends BIOS
  - managed by a group of chipset, hardware, system, firmware, and operating system vendors called the UEFI Forum
  - Faster hardware check compared to BIOS
  - Allows developers to add applications, enabling UEFI to be a lightweight OS
  - Provides secure boot, disables unauthorized applications from booting
  - Provides GUI
- GPT (GUID Partition Table) – required by UEFI
  - Extends MBR
  - GUID: Globally (Universally) unique identifier. Easily creatable almost unique identifier for partitions
  - Supports bigger disks (> 2TB), more partitions (up to 128)
  - It has backup at the end, which can help recover partitions

Previously on CENG111!



OPERATING SYSTEM

Previously on CENG111!



Responsibilities of an OS

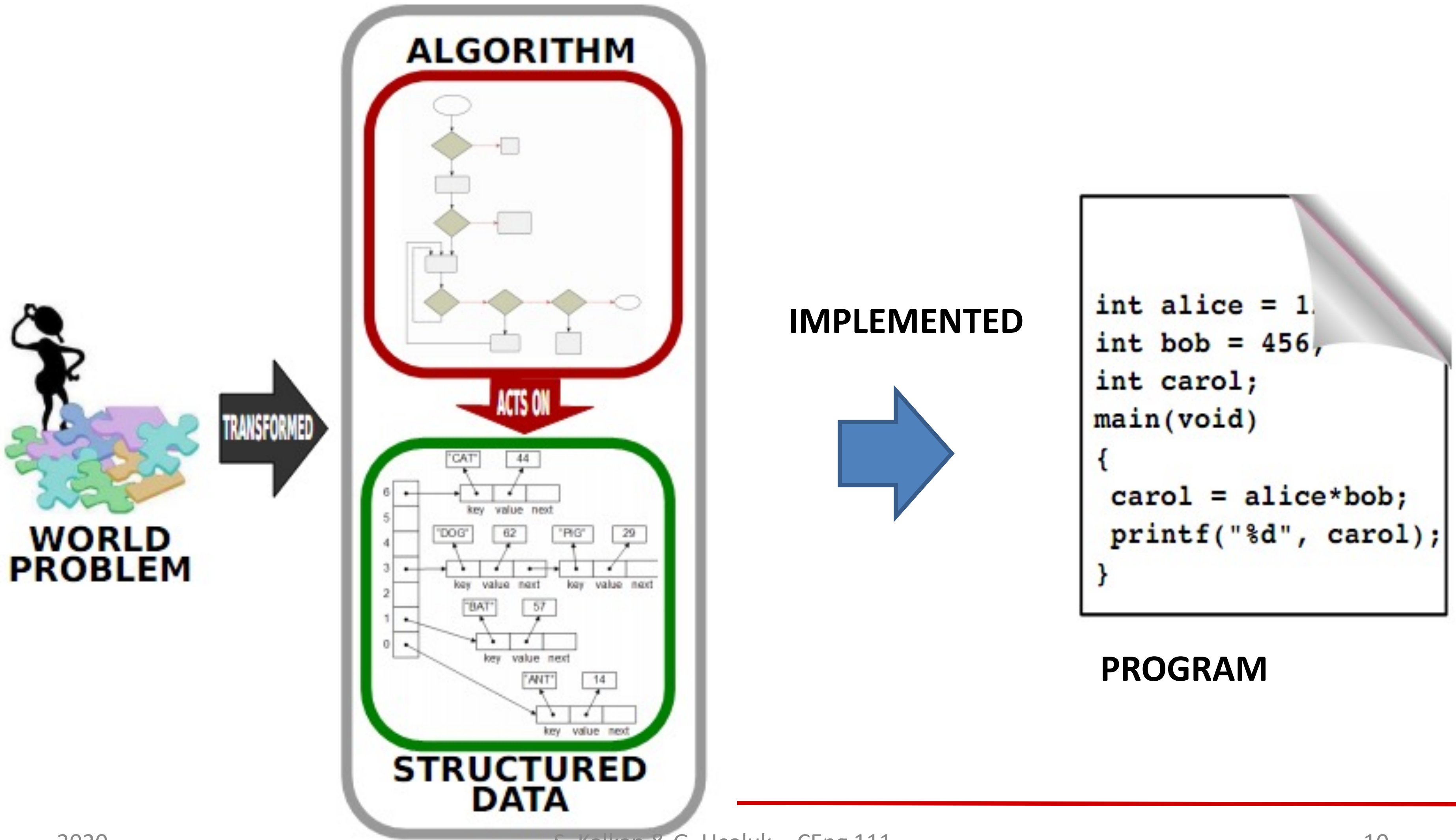
Today

- The world of programming
- Binary representation of data

Administrative Notes

- Midterm date:
  - 22 December, Wednesday, 18:00

Program, Programming

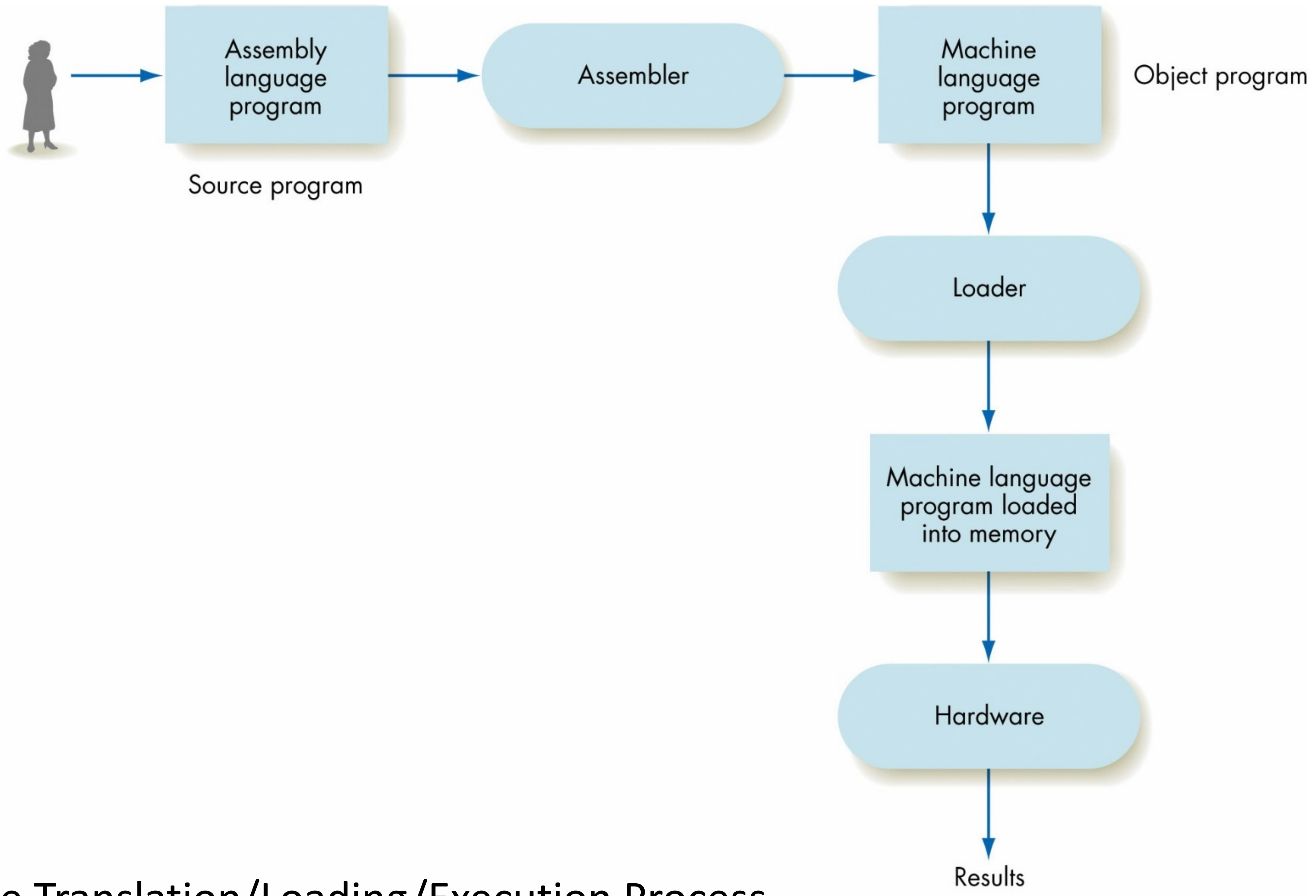


```
int alice = 1
int bob = 456,
int carol;
main(void)
{
 carol = alice*bob;
 printf("%d", carol);
}
```

Program, Programming

- Program:
  - “a series of steps to be carried out or goals to be accomplished”

- A recipe for cooking a certain dish is also a program (but not a computer program).



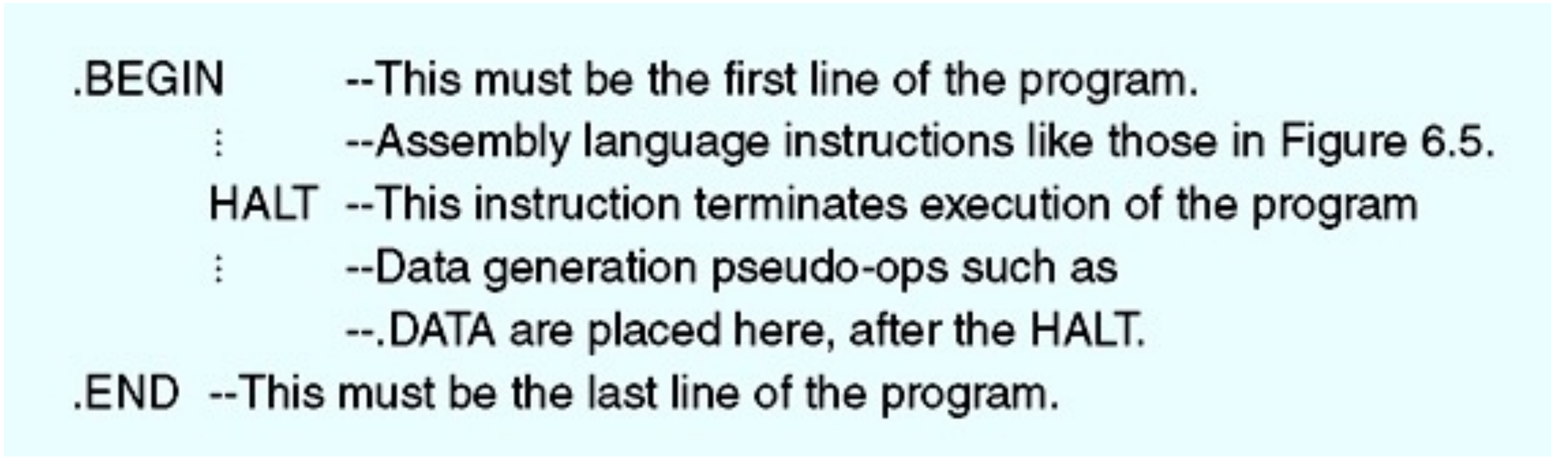
The Translation/Loading/Execution Process

Assembly Language (continued)

- Source program
  - An assembly language program
- Object program
  - A machine language program
- Assembler
  - Translates a source program into a corresponding object program

Assembly Language (continued)

- Advantages of writing in assembly language rather than machine language
  - Use of symbolic operation codes rather than numeric (binary) ones
  - Use of symbolic memory addresses rather than numeric (binary) ones
  - Pseudo-operations that provide useful user-oriented services such as data generation



```
.BEGIN      --This must be the first line of the program.
       :    --Assembly language instructions like those in Figure 6.5.
       HALT --This instruction terminates execution of the program
       :    --Data generation pseudo-ops such as
            --.DATA are placed here, after the HALT.
.END   --This must be the last line of the program.
```

Structure of a Typical Assembly Language Program

Examples of Assembly Language Code

- Arithmetic expression

  A = B + C – 7

  (Assume that B and C have already been assigned values)

Examples of Assembly Language Code (continued)

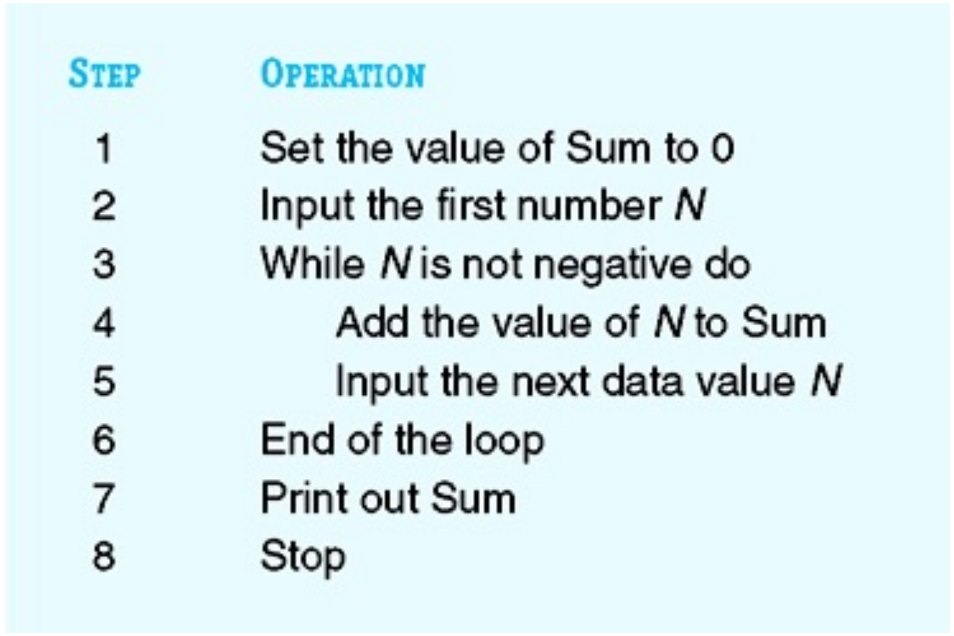
- Assembly language translation

```
            LOAD         B      --Put the value B into register R.
            ADD          C      --R now holds the sum (B + C).
            SUBTRACT SEVEN      --R now holds the expression (B + C - 7).
            STORE       A       --Store the result into A.
              :
              :                --These data should be placed after the HALT.
A:          .DATA       0
B:          .DATA       0
C:          .DATA       0
SEVEN:      .DATA       7      --The constant 7.
```

Examples of Assembly Language Code (continued)

- Problem
  - Read in a sequence of non-negative numbers, one number at a time, and compute their sum
  - When you encounter a negative number, print out the sum of the non-negative values and stop

METU Computer Engineering



Step	Operation
1	Set the value of Sum to 0
2	Input the first number *N*
3	While *N* is not negative do
4	Add the value of *N* to Sum
5	Input the next data value *N*
6	End of the loop
7	Print out Sum
8	Stop

Algorithm to Compute the Sum of Numbers

```
          .BEGIN                      --This marks the start of the program.
          CLEAR     SUM               --Set the running sum to 0 (line 1).
          IN        N                 --Input the first number N (line 2).
--The next three instructions test whether N is a negative number (line 3).
AGAIN:    LOAD      ZERO              --Put 0 into register R.
          COMPARE   N                 --Compare N and 0.
          JUMPLT    NEG               --Go to NEG if N < 0.
--We get here if N ≥ 0. We add N to the running sum (line 4).
          LOAD      SUM               --Put SUM into R.
          ADD       N                 --Add N. R now holds (N + SUM).
          STORE     SUM               --Put the result back into SUM.
--Get the next input value (line 5).
          IN        N
--Now go back and repeat the loop (line 6).
          JUMP      AGAIN
--We get to this section of the program only when we encounter a negative value.
NEG:      OUT       SUM               --Print the sum (line 7)
          HALT                        --and stop (line 8).
--Here are the data generation pseudo-ops
SUM:      .DATA     0                 --The running sum goes here.
N:        .DATA     0                 --The input data are placed here.
ZERO:     .DATA     0                 --The constant 0.
--Now we mark the end of the entire program.
          .END
```

Assembly Language Program to Compute the Sum of Nonnegative Numbers

Translation and Loading

- Before a source program can be run, an assembler and a loader must be invoked
- Assembler
  - Translates a symbolic assembly language program into machine language
- Loader
  - Reads instructions from the object file and stores them into memory for execution

Translation and Loading (continued)

- Assembler tasks
  - Convert symbolic op codes to binary
  - Convert symbolic addresses to binary
  - Perform assembler services requested by the pseudo-ops
  - Put translated instructions into a file for future use

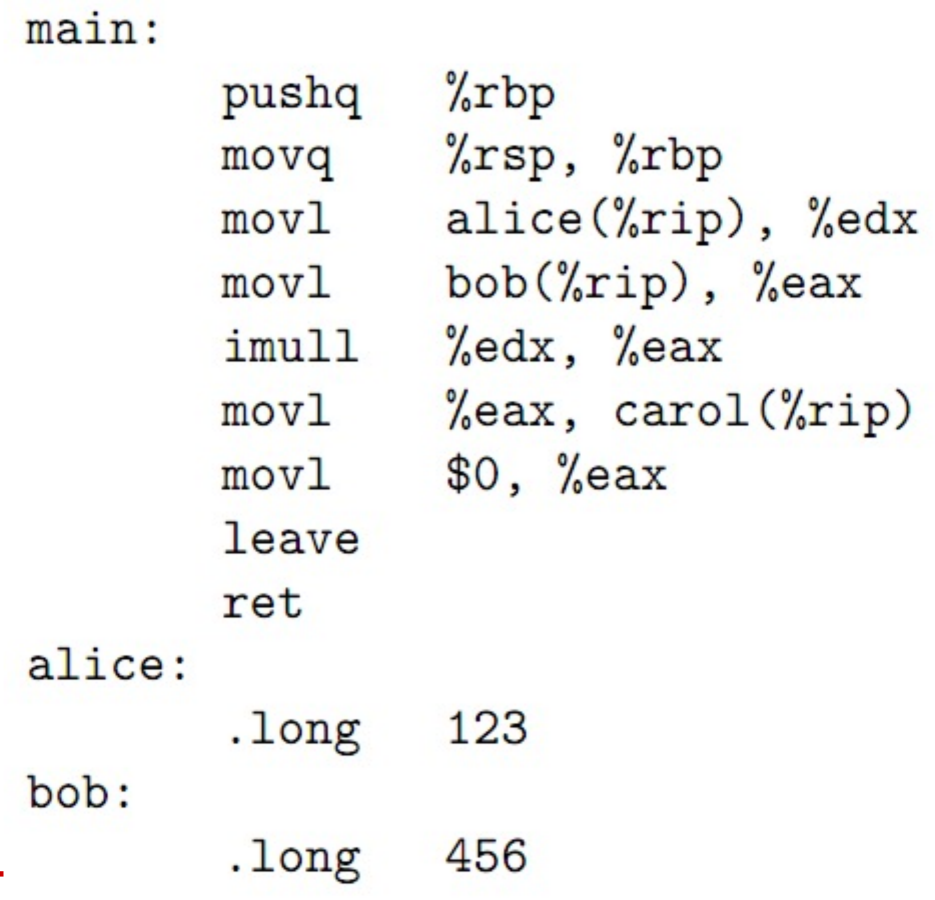
Programming Languages

C code for the example problem:

```
int alice = 123;
int bob = 456;
int carol;
main(void)
{
   carol = alice*bob;
}
```

Programming Languages

Assembly code for the example problem:



```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    alice(%rip), %edx
        movl    bob(%rip), %eax
        imull   %edx, %eax
        movl    %eax, carol(%rip)
        movl    $0, %eax
        leave
        ret
alice:
        .long   123
bob:
        .long   456
```

Programming Languages

Machine code for a simple problem/program:

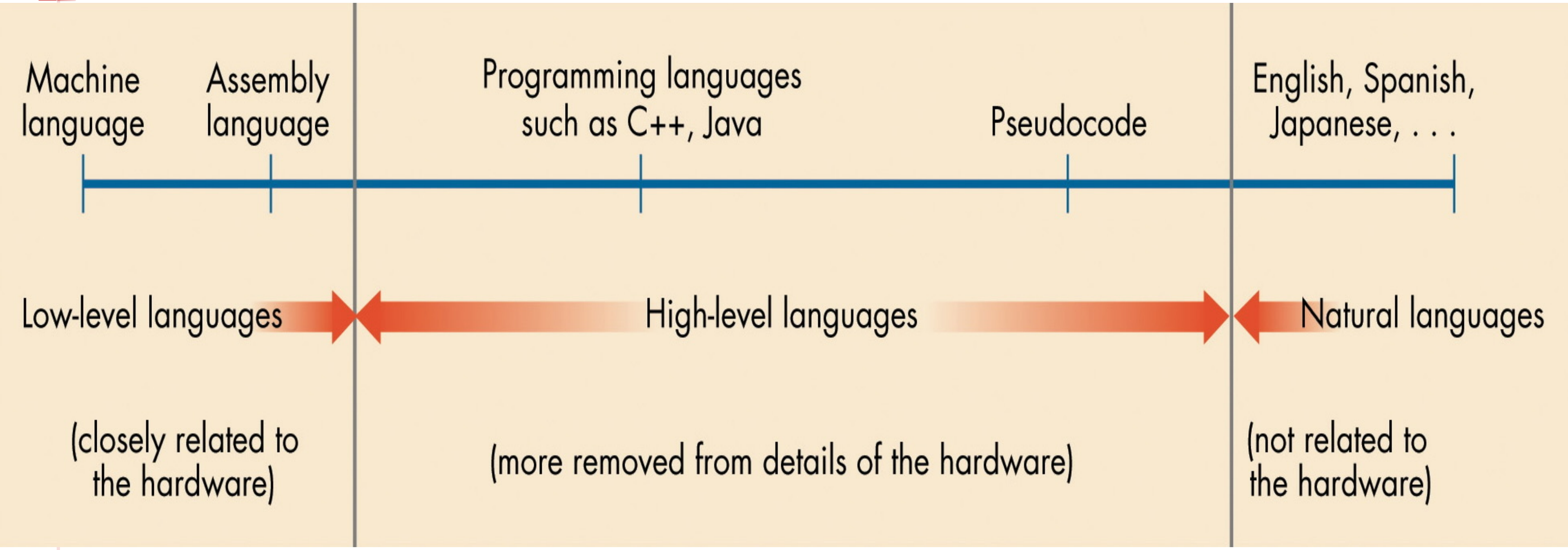
```
01010101 01001000 10001001 11100101 10001011 00010101 10110010 00000011
00100000 00000000 10001011 00000101 10110000 00000011 00100000 00000000
00001111 10101111 11000010 10001001 00000101 10111011 00000011 00100000
00000000 10111000 00000000 00000000 00000000 00000000 11001001 11000011
...
11001000 00000001 00000000 00000000  00000000 00000000
```

```
01010101 01001000 10001001 11100101 10001011 00010101 10110010 00000011
00100000 00000000 10001011 00000101 10110000 00000011 00100000 00000000
00001111 10101111 11000010 10001001 00000101 10111011 00000011 00100000
00000000 10111000 00000000 00000000 00000000 00000000 11001001 11000011
...
11001000 00000001 00000000 00000000  00000000 00000000
```

```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    alice(%rip), %edx
        movl    bob(%rip), %eax
        imull   %edx, %eax
        movl    %eax, carol(%rip)
        movl    $0, %eax
        leave
        ret
alice:
        .long   123
bob:
        .long   456
```

```
int alice = 123;
int bob = 456;
int carol;
main(void)
{
  carol = alice*bob;
}
```

The Spectrum of Programming Languages


Machine language
Assembly language
Programming languages such as C++, Java
Pseudocode
English, Spanish, Japanese, . . .
Low-level languages
High-level languages
Natural languages
(closely related to the hardware)
(more removed from details of the hardware)
(not related to the hardware)

- There is a limit to how high a language can get.
- Why can't we write programs in our spoken language?

Programming Language Paradigms

- Classification / Categorization of programming languages.
  - Imperative Paradigm
  - Functional Paradigm
  - Logical-declarative Paradigm
  - Object-oriented Paradigm
  - Concurrent Paradigm
  - Event-driven Paradigm

Imperative Paradigm

Statement_1

Statement_2

Statement_3

Statement_4

Statement_5

From C:

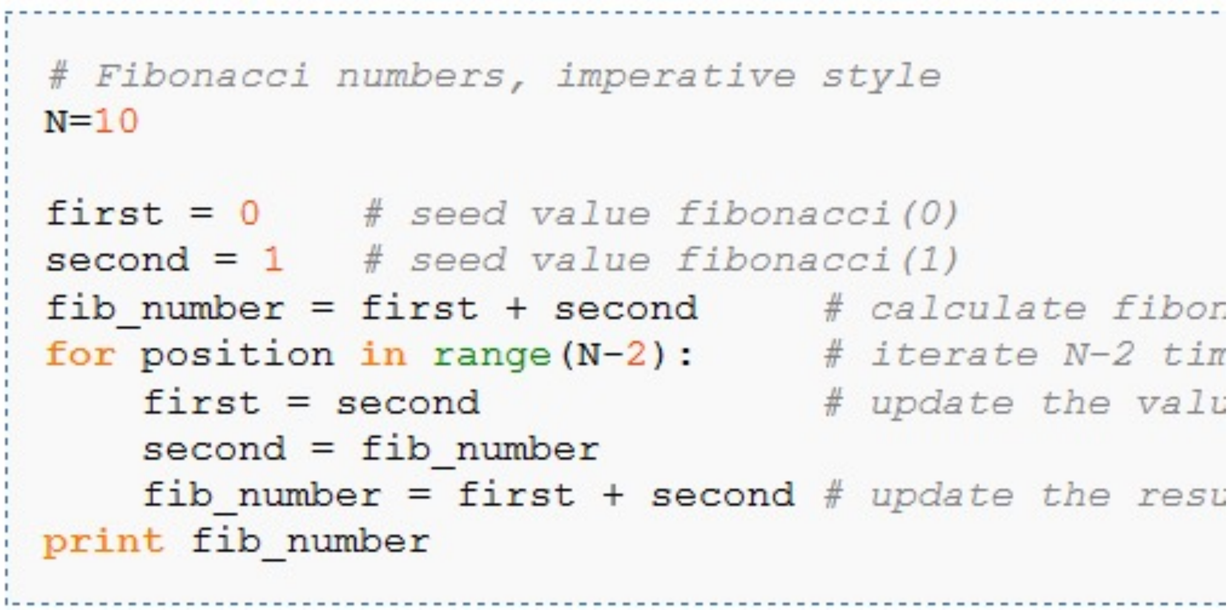
```
int a = 2;
int b = a * 2;
int c;

c = -b – sqrt(b*b - 4*a*c) / (2*a);
```

Functional Paradigm

- Data environment is restricted.
- Functions receive their inputs and return their results to the data environment.
- Programmer’s task:
  - decompose the problem into a set of functions such that the composition of these functions produce the desired result.

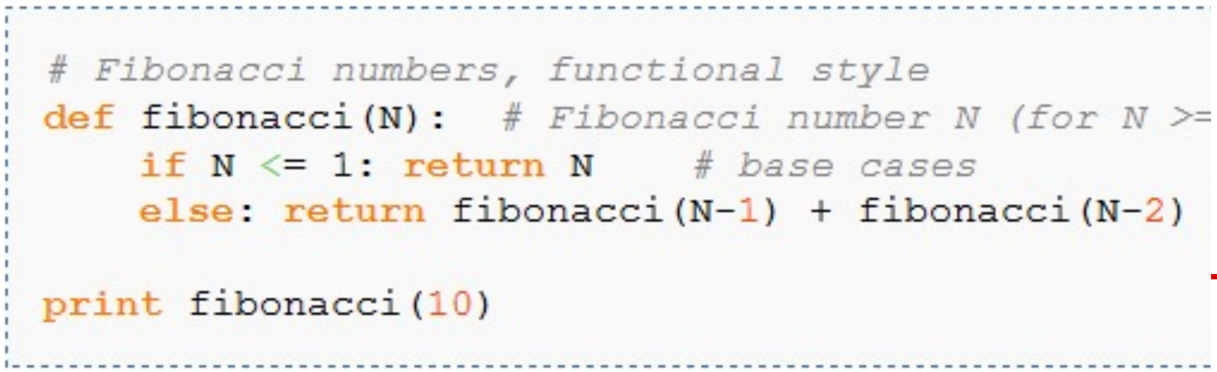
Functional Paradigm

Imperative Version of Fibonacci Numbers



```
# Fibonacci numbers, imperative style
N=10

first = 0      # seed value fibonacci(0)
second = 1     # seed value fibonacci(1)
fib_number = first + second        # calculate fibon
for position in range(N-2):        # iterate N-2 tim
    first = second                 # update the valu
    second = fib_number
    fib_number = first + second # update the resu
print fib_number
```

Functional Version of Fibonacci Numbers



```
# Fibonacci numbers, functional style
def fibonacci(N):   # Fibonacci number N (for N >=
    if N <= 1: return N      # base cases
    else: return fibonacci(N-1) + fibonacci(N-2)

print fibonacci(10)
```

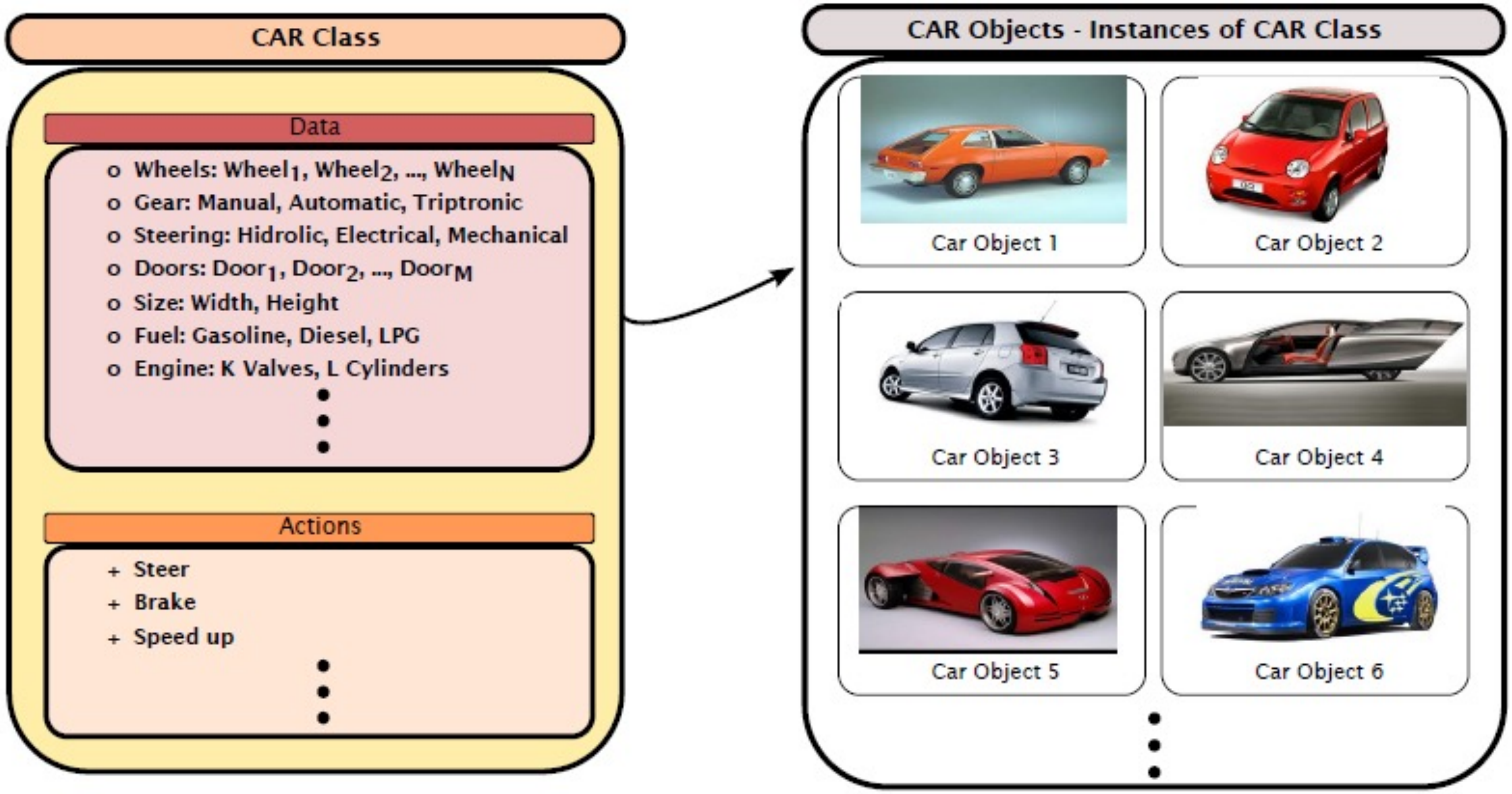
Logical-declarative Paradigm

- The data and the relations are states as rules, or facts.
- The problem is solved by writing new rules/facts.

From Prolog:

```
mother(matilda,ruth).
mother(trudi,paggy).
mother(eve,alice).
mother(zoe,sue).
mother(eve,trudi).
mother(matilda,eve).
mother(eve,carol).
grandma(X,Y) :- mother(X,Z), mother(Z,Y).
```

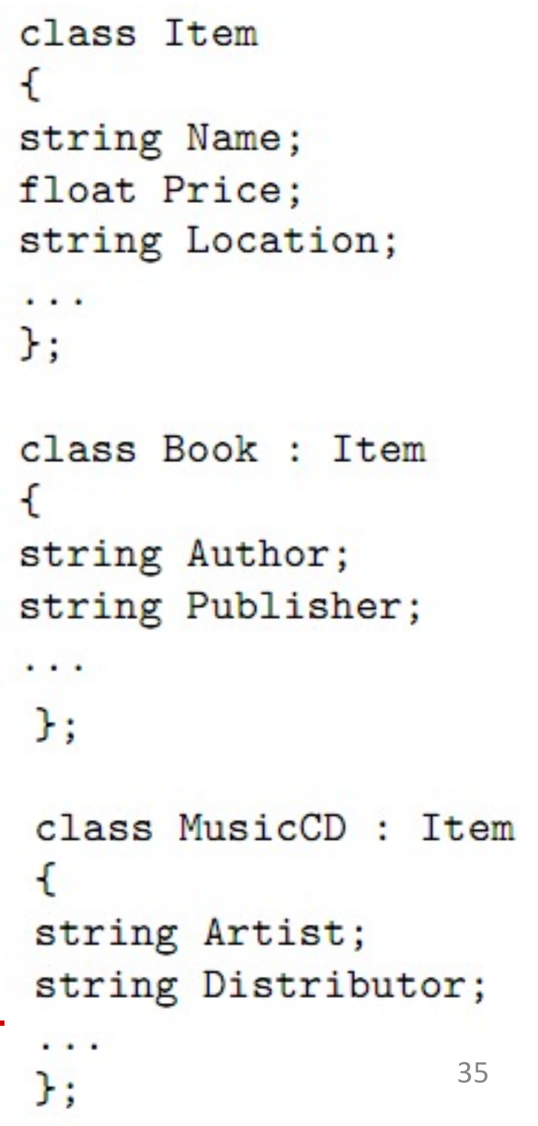
Object Oriented Paradigm


CAR Class
Data
o Wheels: $Wheel_1$, $Wheel_2$, ..., $Wheel_N$
o Gear: Manual, Automatic, Triptronic
o Steering: Hidrolic, Electrical, Mechanical
o Doors: $Door_1$, $Door_2$, ..., $Door_M$
o Size: Width, Height
o Fuel: Gasoline, Diesel, LPG
o Engine: K Valves, L Cylinders
Actions
+ Steer
+ Brake
+ Speed up
CAR Objects - Instances of CAR Class
Car Object 1
Car Object 2
Car Object 3
Car Object 4
Car Object 5
Car Object 6

Object Oriented Paradigm

- Problem is decomposed into objects which hold data and the corresponding functions on the data.
- Objects can be defined using other objects as a basis; the new object inherits from the basis objects.

From C++:



```
class Item
{
string Name;
float Price;
string Location;
...
};

class Book : Item
{
string Author;
string Publisher;
...
 };

 class MusicCD : Item
 {
 string Artist;
 string Distributor;
 ...
 };
```

METU Computer Engineering

Concurrent Paradigm

- Programming using multiple CPUs concurrently.
- The task is to assign the overall flow & data to individual CPUs.
- With the bottleneck in CPU power, this paradigm is going to be the trend in the future.

Event-Driven Paradigm

- A program is composed of events and what to do in case of events.
- The task is to decompose a problem into a set of events and the corresponding functionalities that will be executed in case of events.
- Suitable for Graphical User Interface design.

The hyperspace of languages

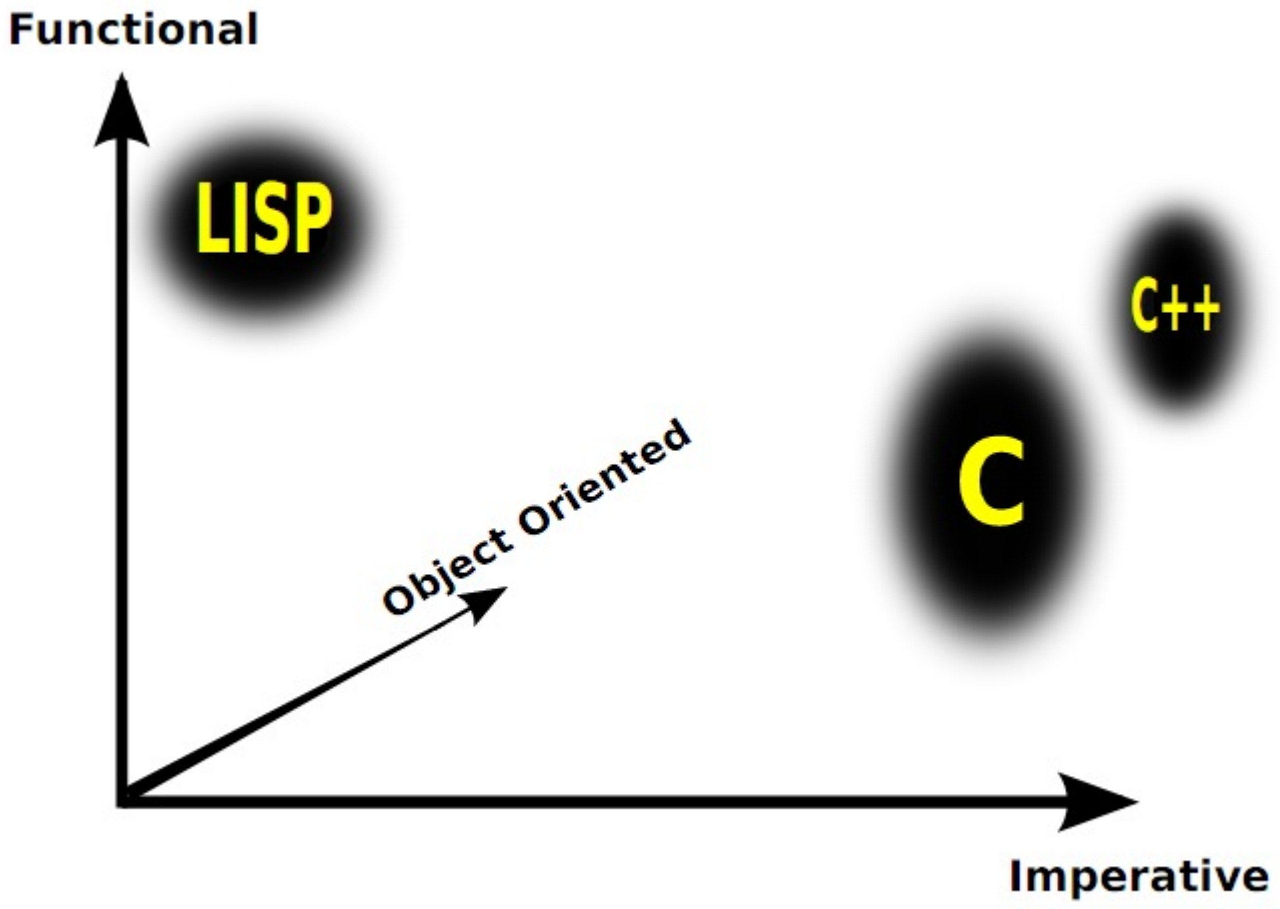


Figure 1.4: The hyperspace of programming *(only 3 axes displayed)*

Zoo of Programming Languages

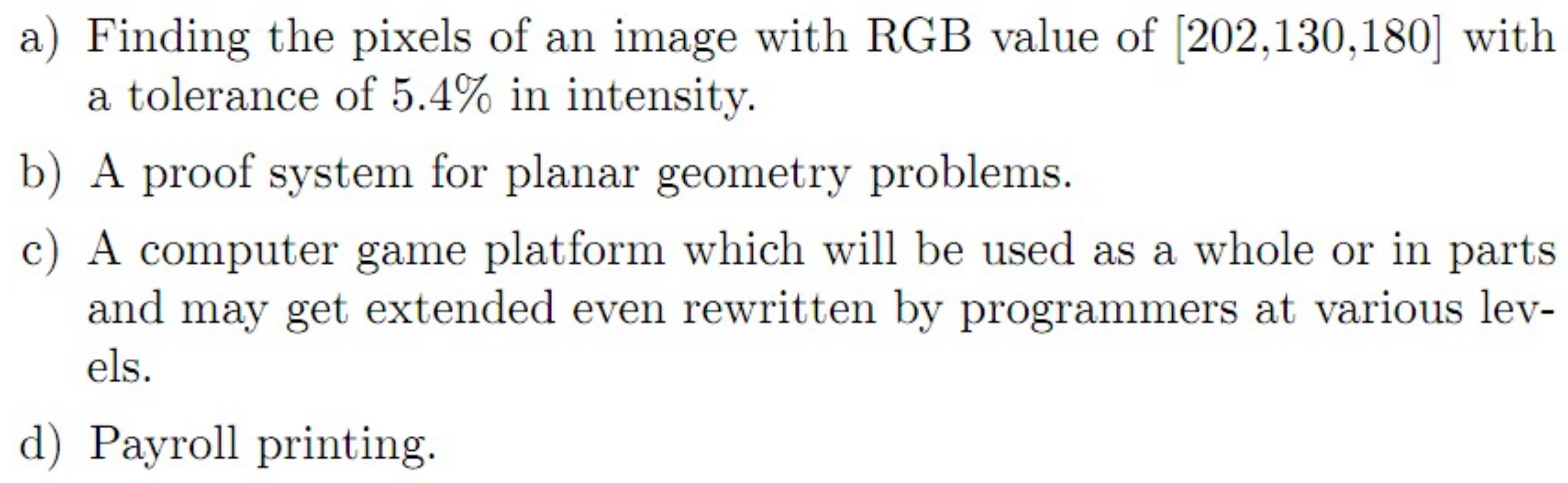
- Around 700 programming languages!
- But, why do we not have a programming language that serves all paradigms/purposes/requirements?

Choosing a PL

- Ex: Moving soil with a shovel and a grader

Factors that affect choosing a PL

- Domain & Technical Nature of the Problem



a) Finding the pixels of an image with RGB value of [202,130,180] with a tolerance of 5.4% in intensity.

b) A proof system for planar geometry problems.

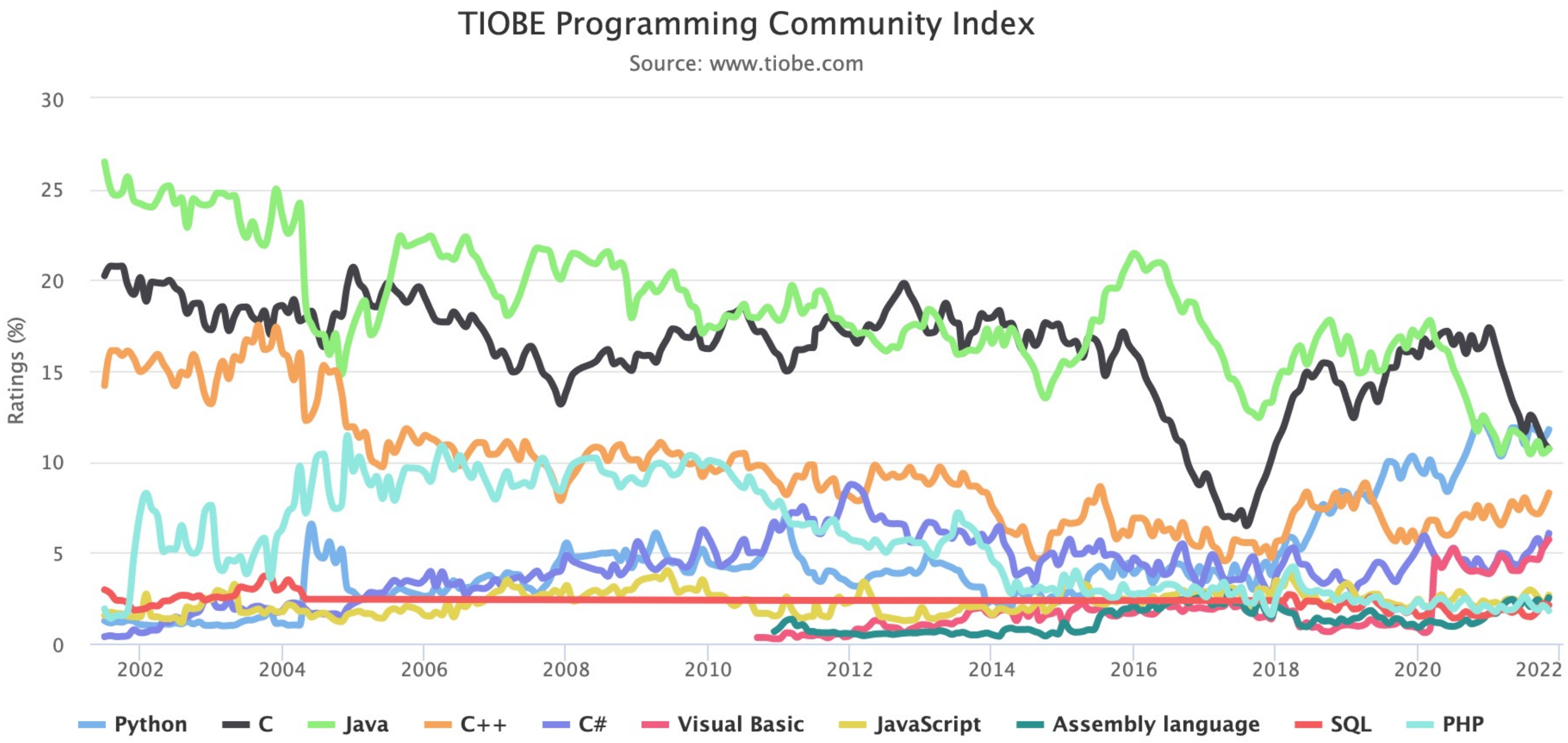
c) A computer game platform which will be used as a whole or in parts and may get extended even rewritten by programmers at various levels.

d) Payroll printing.

Factors that affect choosing a PL

- Personal taste and preference
- Circumstance-imposed constraints
  - e.g., time limit.
- Current trend

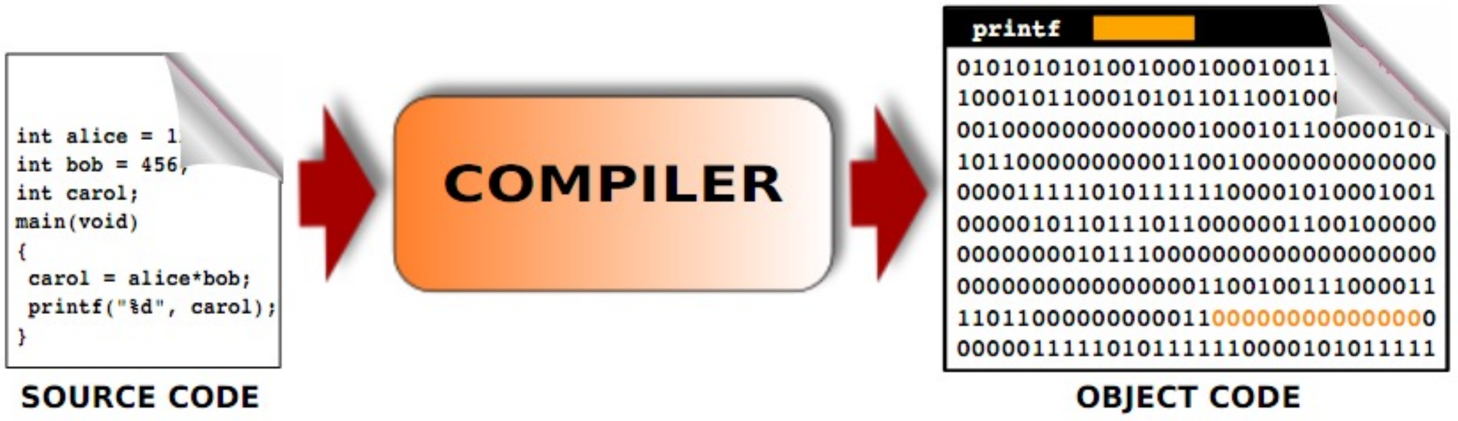
Current trend



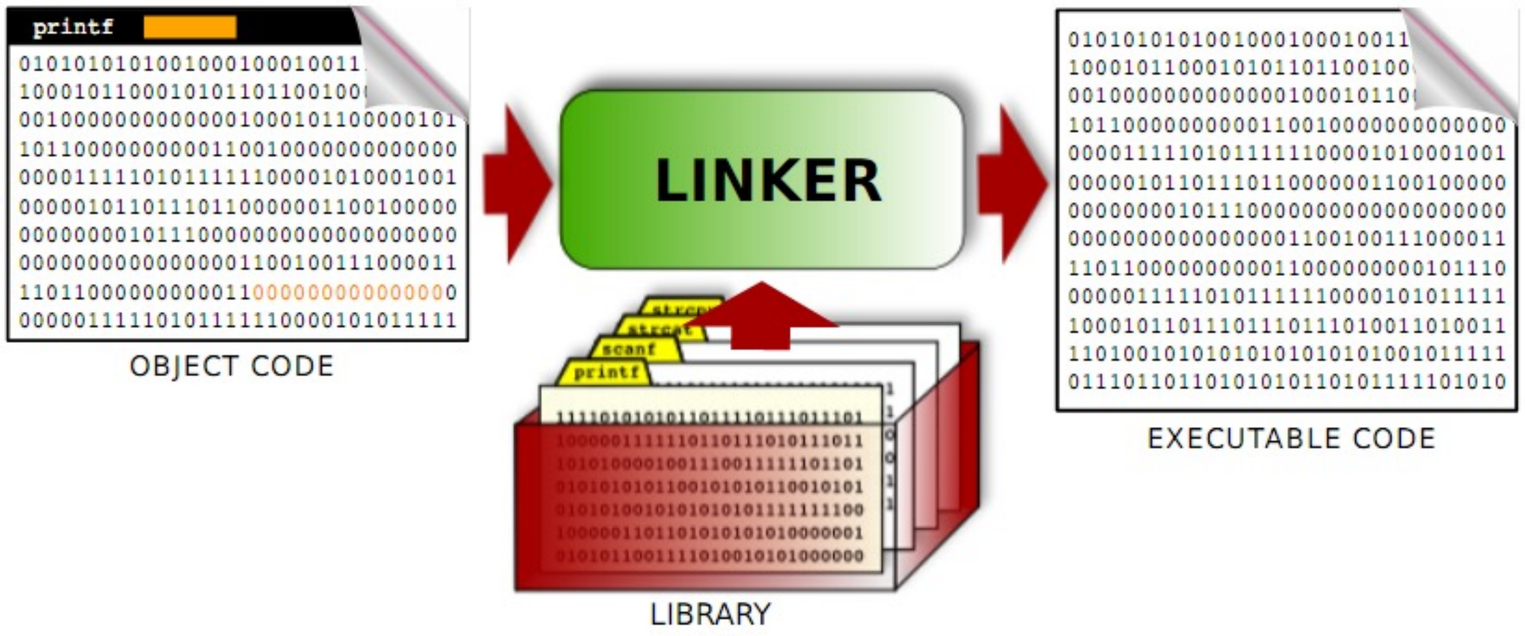
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

How are languages implemented

COMPILATIVE APPROACH



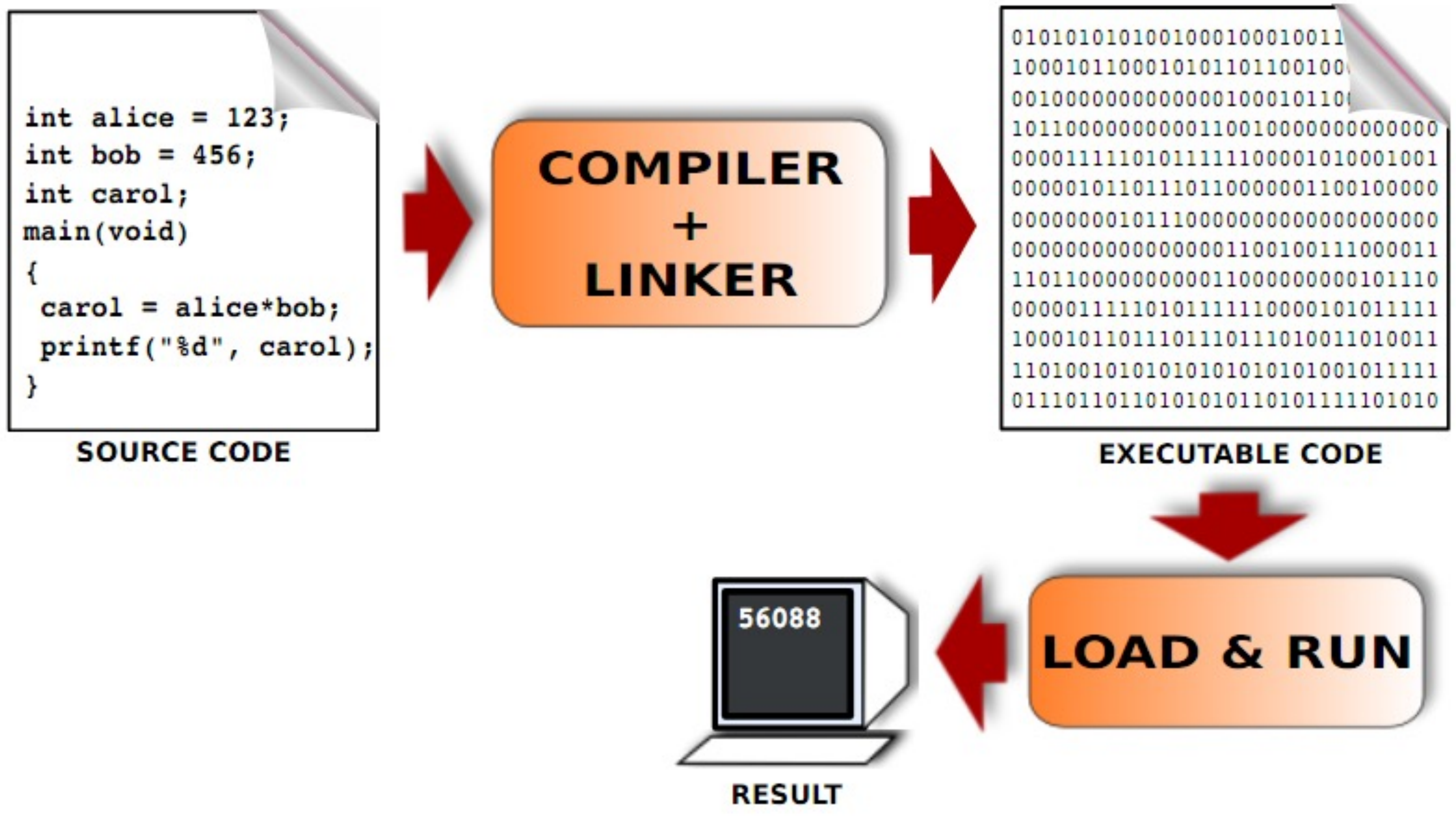
```
int alice = 1
int bob = 456,
int carol;
main(void)
{
 carol = alice*bob;
 printf("%d", carol);
}
```



METU Computer Engineering

COMPILATIVE APPROACH

How are languages implemented

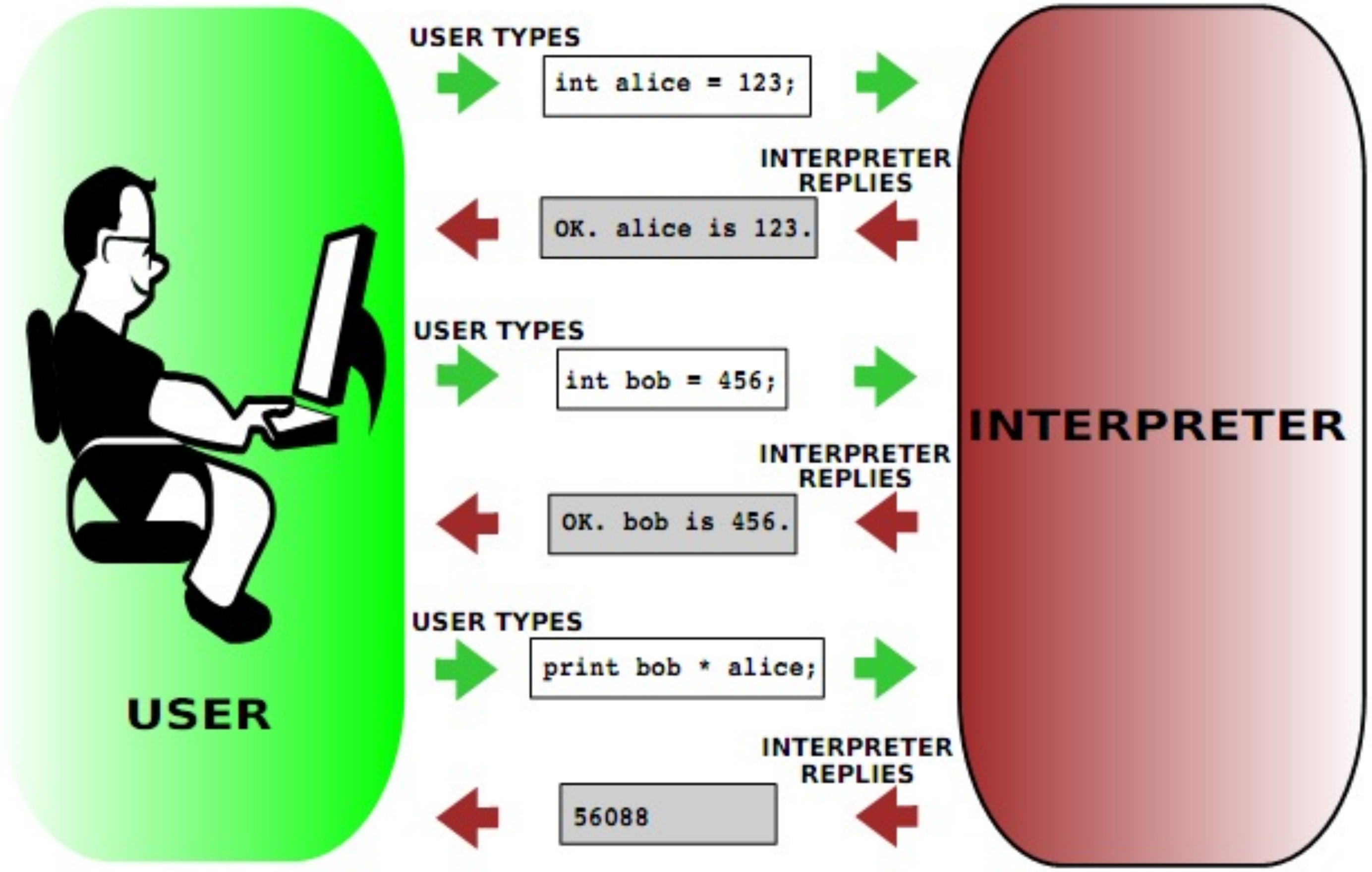


```
int alice = 123;
int bob = 456;
int carol;
main(void)
{
 carol = alice*bob;
 printf("%d", carol);
}
```

```
01010101010010001000100111
1000101100010101101100100
0010000000000000100010110
10110000000000011001000000000000
00001111101011111100001010001001
00000101101110110000001100100000
00000000101110000000000000000000
00000000000000001100100111000011
11011000000000011000000000101110
00000111110101111110000101011111
10001011011101110111010011010011
11010010101010101010101001011111
01110110110101010110101111101010
```

How are languages implemented

INTERPRETIVE APPROACH


USER
USER TYPES
int alice = 123;
INTERPRETER REPLIES
OK. alice is 123.
USER TYPES
int bob = 456;
INTERPRETER REPLIES
OK. bob is 456.
USER TYPES
print bob * alice;
INTERPRETER REPLIES
56088
INTERPRETER