How Python Works Underneath

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Model of Computer Hardware

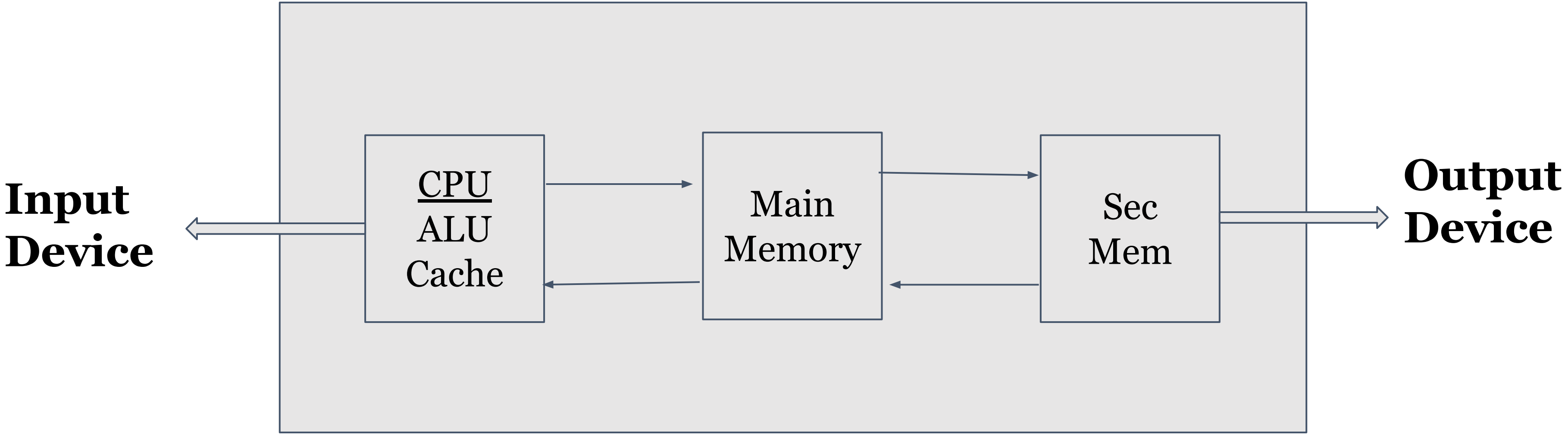
- A computer has these key components (simplified view)
  - CPU: Computational unit. It has a set of defined instructions like add, mult, store, read, check_condition, jump_on_condition,... It can execute these instructions only - data for these instructions is taken from memory
  - Memory: A sequence of memory words; CPU can read/write any individual memory word by giving its address
  - Input/Output Devices: To input or output to/from CPU/Memory from outside - e.g. keyboard and printer/screen
  - Secondary memory (disk): Permanent storage - generally written and read in blocks to/from memory

How Computer HW Executes a Program

- A program is a sequence of CPU instructions - has to be loaded in contiguous memory
- Eg. z = x*y will be written as:

```
Load R1 x
Load R2 y
Mult R1 R2
Write R1 z
```

- The location of first instruction of program is given to CPU
- CPU executes the instruction, loads the next instruction, and keeps repeating this till "end/stop" (recall there may be jump on condition instructions - for loop and conditionals)



Input Device

CPU

ALU

Cache

Main Memory

Sec Mem

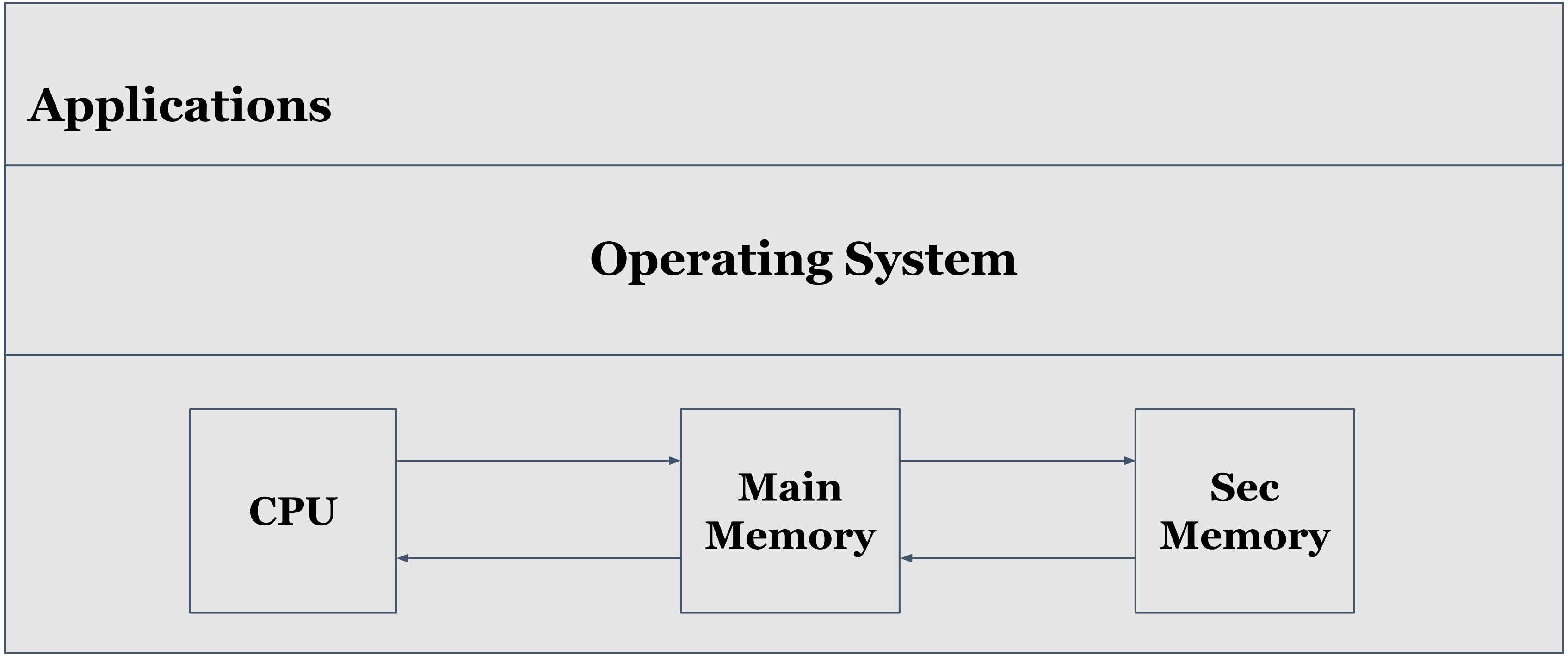
Output Device

Operating Systems

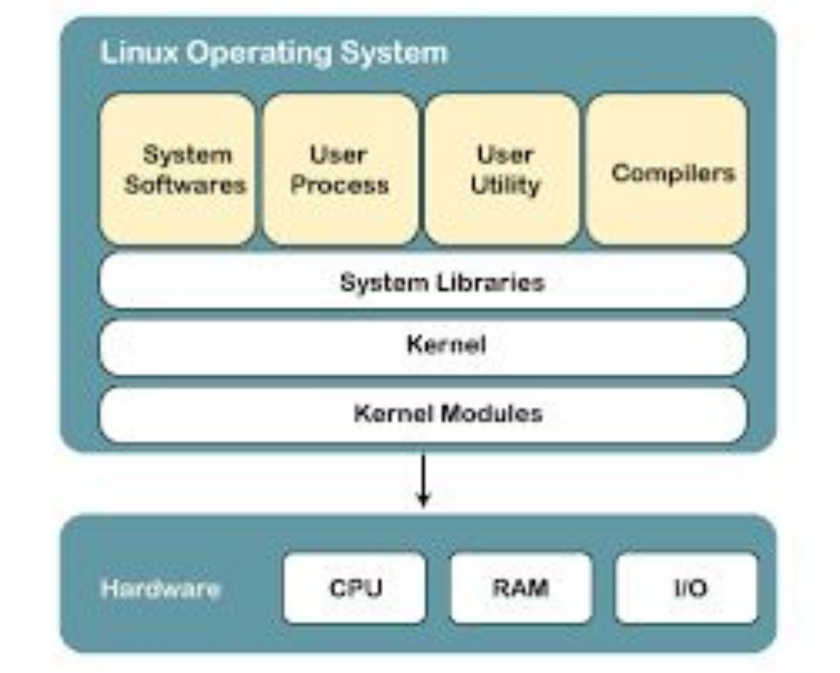
- Using hardware directly - very hard (have to load the program, give its first location, manage I/O…)
    - First computers built - this had to be done
- Operating Systems (OS): are programs that run on the hardware and provide interface to users and programmers
- An OS is always running program - which takes input commands from users, gets them executed on hardware, and provides output
- We almost never see hardware - we only interact with OS - all our interaction with the computer is through the OS
- The OS provides us a "shell" to give it commands directly, or programs can make requests: to execute, get memory, open a file,..

How OS executes our Programs

- All programs users write are "applications" - including the programs we may download (e.g. editor, python compiler,...)
- To execute a program, we have to give the program to OS, which then gets it executed on hardware
  - The program has to be executable - i.e. in the machine language
- When given a user program, OS creates a process to run on CPU, instructs it to execute the user program code, and assigns it some memory which user program can use (for variables, etc)
- The OS mediates the input/output of our program


Applications
Operating System
CPU
Main Memory
Sec Memory

Machine with Operating System



Source: https://www.javatpoint.com/what-is-linux

Programs in Compiled Language

- We run an editor program (through OS), and create a program using C/C++ syntax, save it in a file (through OS)
- Run the compiler program, give it our C file as input, and the compiler produces the machine language code in a file
  - Compiler is also a program that can take input and produce output
- This executable file (which is machine language version of our program) we can give to the OS to run
- The machine language (assembly) has instructions like: LOAD R1 X, STORE R2 Y, ADD R1 R2, MULT R1 R2, ….: each is an operation directly performed by the CPU

Python Programs / Interpreted Languages

- Python works differently - python compilers does not generate machine code for python programs
- Instead they generate a byte-code - which is "higher level" - in between the low level machine code and high level Python
- This byte code cannot be run on the machine - instead it is given to an interpreter
    - The interpreter is a program which runs on the hardware and which takes the bytecode and executes them efficiently
- This is why Python (or Java) are called interpreted languages (as compared to compiled languages) - note that there is a compiler in python also which generates bytecode which is interpreted
- Bytecode for your program - with some effort you can see it

Python Bytecode – example

- Actual bytecode is in bytes
- Readable version provided by dis (disassembler)
- >>>import dis

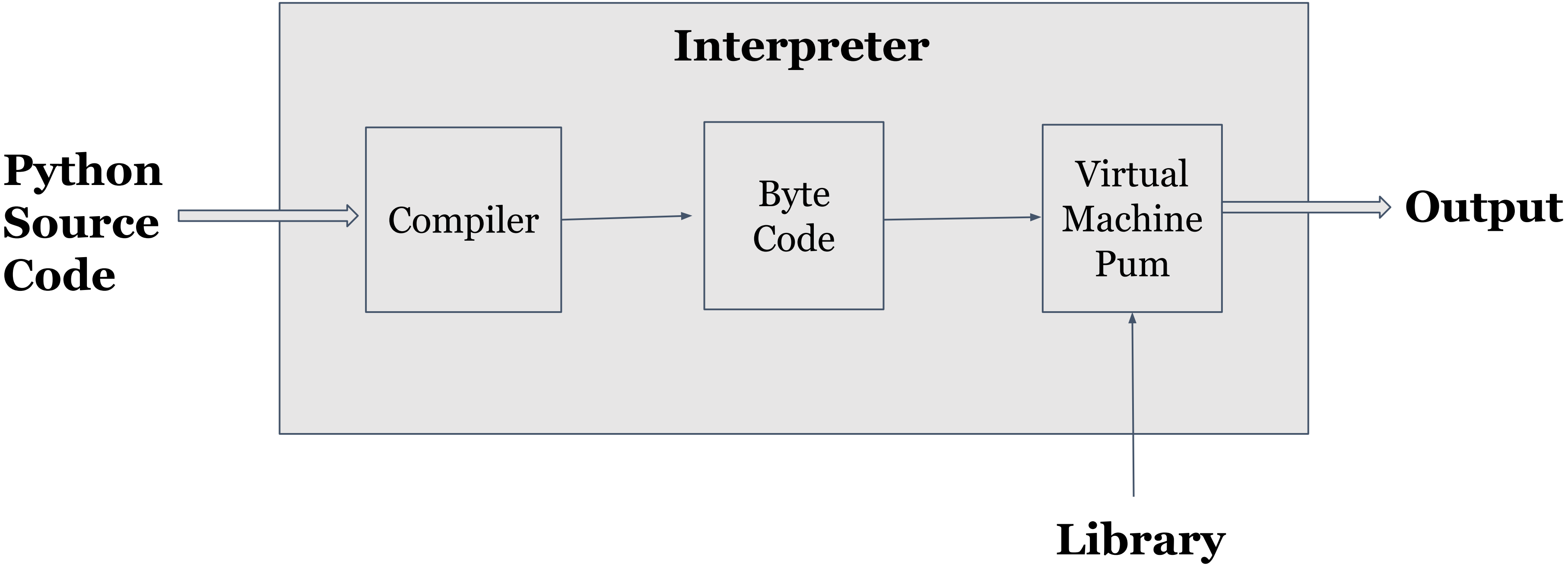
```
def f(x,y):
    z = x*y
    return(z)
```

- >>>dis.dis(f)

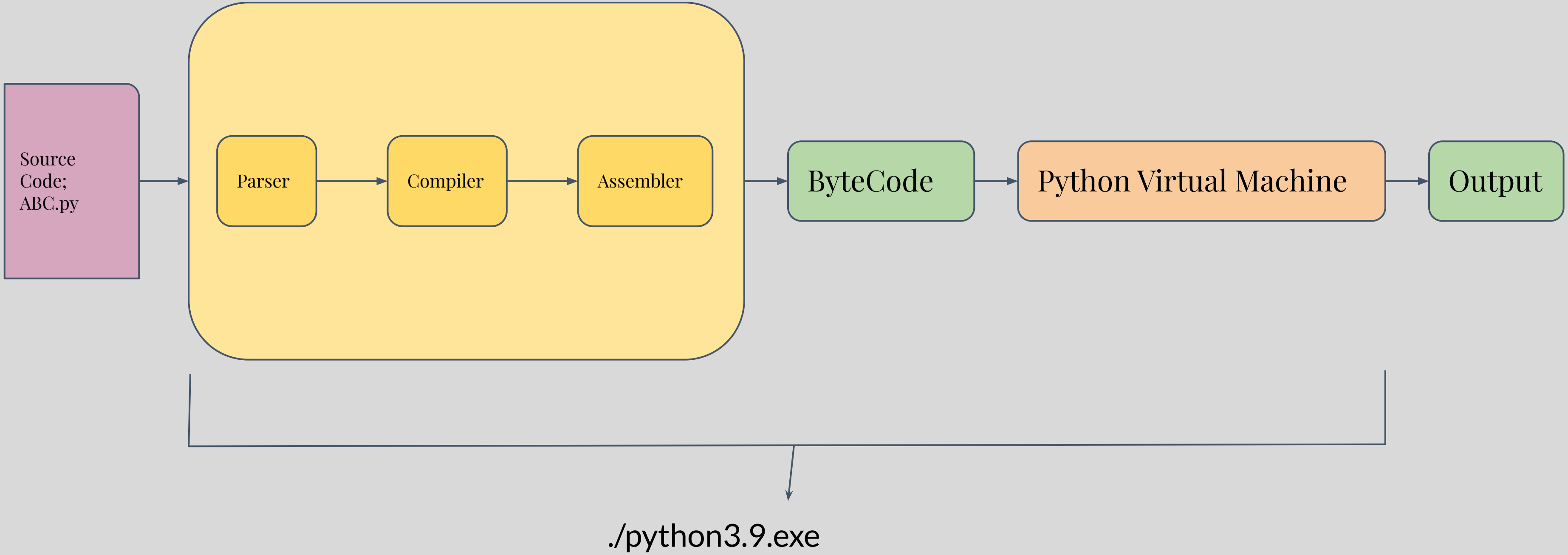
```
1       0 RESUME            0

2        2 LOAD_FAST          0 (x)
        4 LOAD_FAST          1 (y)
        6 BINARY_OP          5 (*)
       10 STORE_FAST          2 (z)

3       12 LOAD_FAST          2 (z)
       14 RETURN_VALUE
```


Python Source Code
Interpreter
Compiler
Byte Code
Virtual Machine Pum
Output
Library

Cpython : A Python Interpreter



Python Program Execution

- So, when we execute our program, we actually execute the python system (generally Cpython) which has a compiler to compile source code to bytecode, and an interpreter (a runtime system) to execute bytecode
  - We dont directly run our programs on hardware as in compiled languages
- That is why the interpreted languages are often a little slower than the compiled languages

Quiz

Which of these are true

A. Python is an interpreted language, but has a compiler

B. The python compiler converts each source code line to bytecode

C. The python interpreter executes the bytecode

D. The bytecode can be executed directly on hardware

Compiled vs Interpreted

Compiled

- Source code translated to machine language
- Generated code is machine dependent (so compilers for each machine)
- Code is generally faster
- Compilers can do a lot of compile time checks

Interpreted

- Source translated to intermediate code
- Gen code is mc independent
- Code execution is slower (as interpreter runs in app mode)
- Interpreters do a lot more checking at runtime

More on Execution

- Python runtime system has a frames stack - each function gets a frame when it is executing (as in pythontutor) - local vars are kept in the stack frame
- For each frame, there is an evaluation stack - this is where the bytecode is executed (e.g. x*y*z)
- All values are kept in objects and variables point to those objects - for integer, float etc, objects are immutable, i.e. they never changed - new objects are created when we assign a new value to a variable (for other types of objects this will change)
- Objects are allocated on heap memory (pointed to by variables in the stack frame)
- Objects to which no variable is pointing is useless and is removed by "garbage collection"
- Many videos/articles available on this, e.g: https://www.youtube.com/watch?v=0Om2gYU6clE&t=400s
-

Summary

- Basic machine architecture: CPU, Main memory, Secondary storage, and I/O devices; CPU executes instructions in machine language, which may load and store data from main memory
- OS is the software running on the machine which provides an interface between user programs and hardware
- Compiled languages - programs converted to machine code, then given to OS to execute
- Interpreted languages - compiled into intermediate language (bytecode), which is then executed by an interpreter
- Bytecode execution: stack frames which keeps vars of fns/global; eval stack where bytecode evaluation is done
- Vars point to objects on heap memory - immutable objects do not change their state; new objects are created

Python Built-in Functions

- Python has many built in functions - these can be used anywhere a function can be used

  https://www.w3schools.com/python/python_ref_functions.asp

  https://docs.python.org/3/library/functions.html

- Each function works on some types (e.g. abs works on numbers)
- Some standard functions which work on many types: abs(), ascii(), bin(), int(), float(), round()
- Some that apply to list type objects: len(), max(), sum()..
- Some others: type(), id()

Some built-in functions

abs(x), bool(x), round(x,ndigits=None) # if ndigits value not provided, default is used - round

bin(x), oct(x) # binary and octal values

chr(i), ord(c) #character for an integer i; integer of the unicode character c

dir(object) # list of attributes/names of that object

dir() # names in the current scope

type(object) # type of the object

help(request) # returns help

float(x), int(x) # returns a float/int of parm x, which is a number or a string

input(), print(),

len(s) # number of items in s - s may be a sequence (string, type, list, range), collection (set, dictionary)

reversed(seq) # returns an integrator that reverses the traversal from end to start

sorted(utterable, key=None, reversed=False) # returns a sorted list from utterable; key is a function which can be specified for sorting; reversed will reverse the order to

max(l), min(l), max(x1, x2, ...) #returns the max/min

sum(iterable)

locals() # local variables in the scope

range(n1, n2, step=1) # actually returns a class, but can use it as a function

Other Operators in Python

- Python has some more operators also - we have seen arithmetic, relational (comparison), boolean (logical),
- Assignment operators: besides "= " we have some more

  x += 5 # same as x = x+5

  x *= y # same as x = x*y

  Similar assignment ops for -, %, /, //, **

- Identity operators: `is` (checks if two variables are same), `is not`
  - True if both x and y point to the same object
  - On every assignment, python creates a new object
- Membership operator: `in` (x in y - checks if x is in y), `not in`
  - Works on iterable objects like lists, strings, …
- (Bitwise operations: on binary representation of operand)

Announcements

- Quiz in next class - for last half hour
-
- Assignment 1 to be released soon on GC: to use knowledge of python till now (vars, expressions, conditionals, loops, functions) to solve problems
- Assignments are to be done on VS code - you will submit your code files on GC.