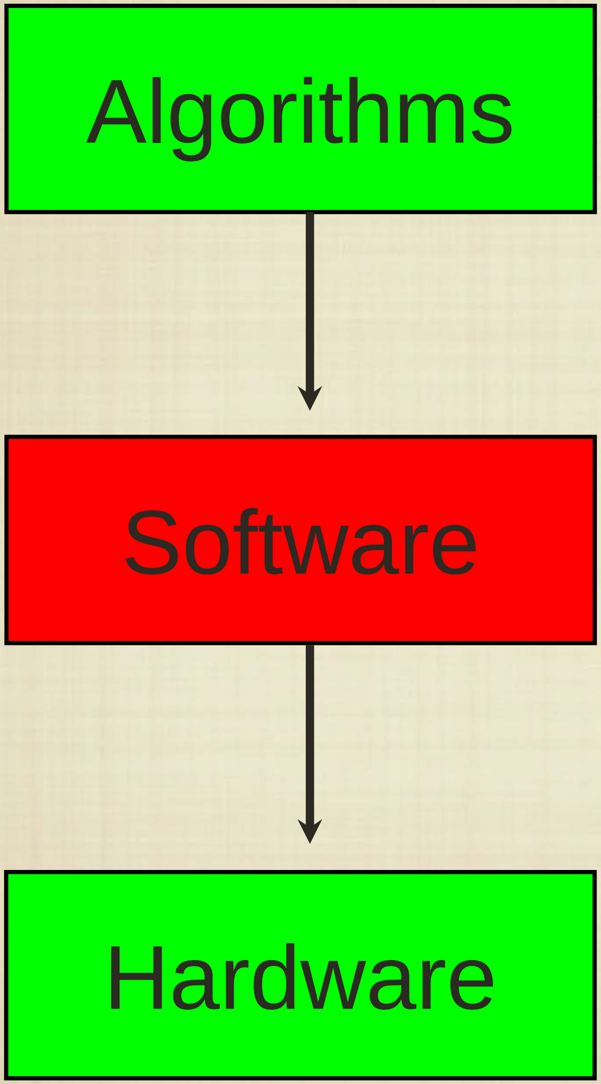
Languages and Data Types

Spring 2014
Carola Wenk

Big Picture



Algorithms

Worst-case analysis of running time, simple linear-time algorithms, and efficient searching and sorting.

Software

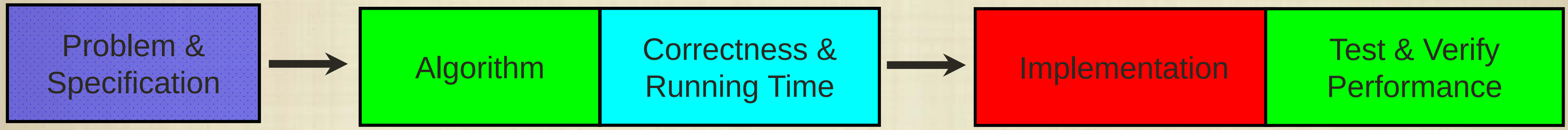
Python: variables, loops, if-then, functions, lists, recursion

Hardware

Von Neumann architecture, logic, gates, circuits, binary numbers, machine instructions

Tool Development

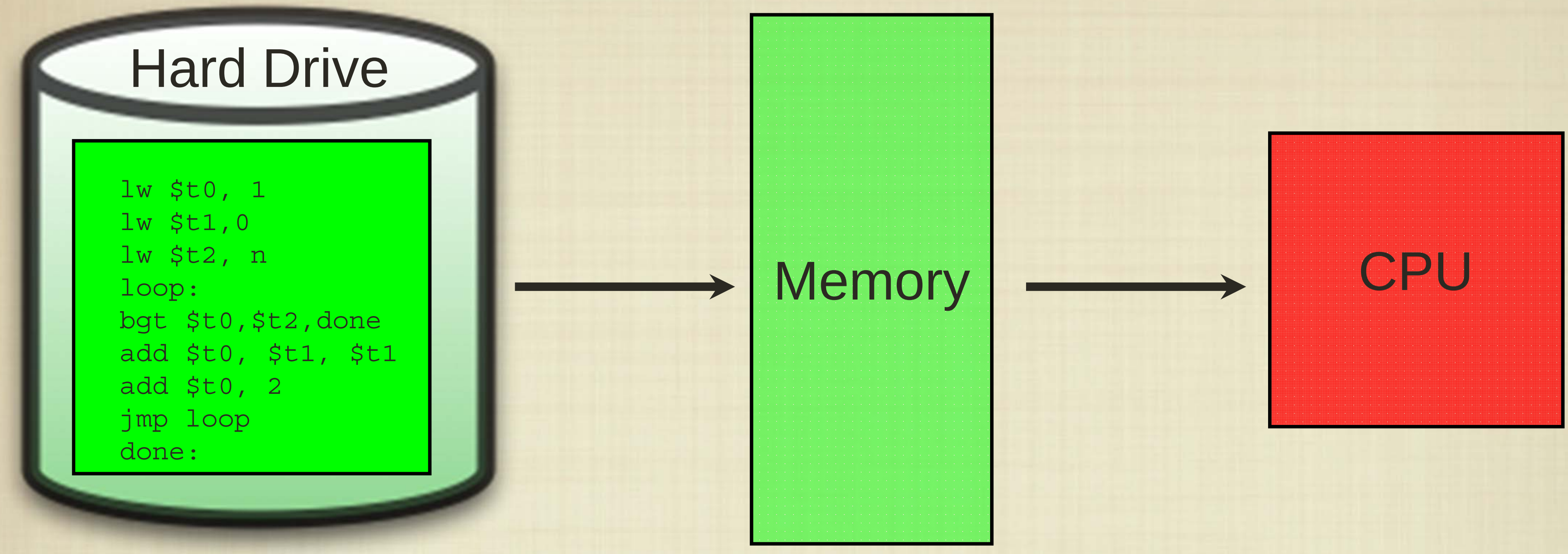
The basic steps of designing and implementing an algorithm:



Computational tools are everywhere because algorithms are defined abstractly.

Once we formulate a problem in a particular application area, we try to use our “toolbox” of algorithms to efficiently manage and process information.

Where do programs “live”?

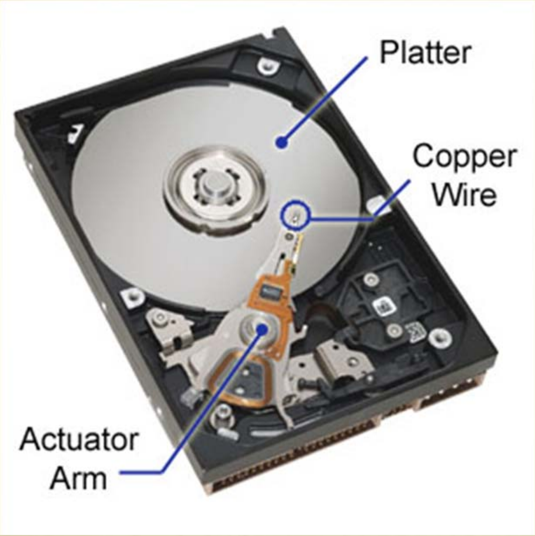


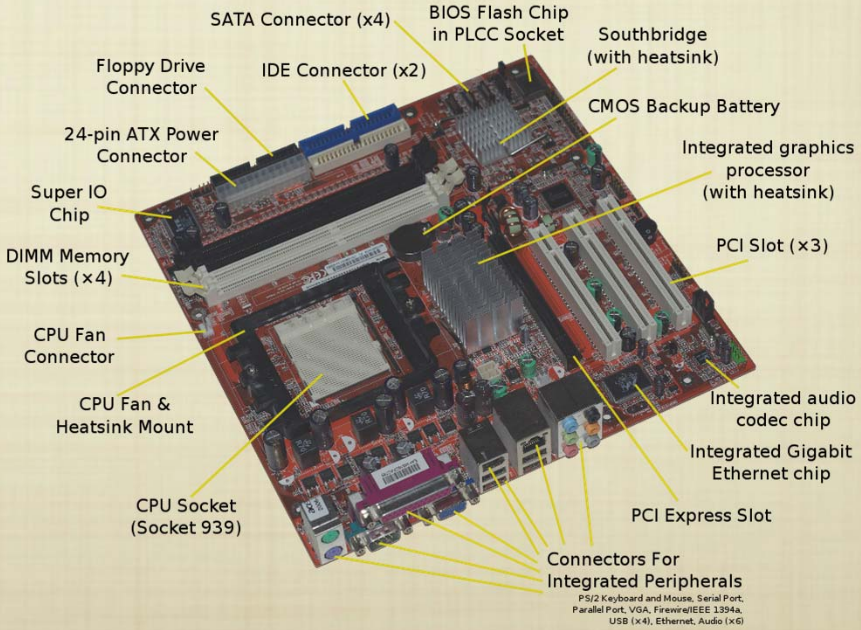
Hard Drive

```
lw $t0, 1
lw $t1,0
lw $t2, n
loop:
bgt $t0,$t2,done
add $t0, $t1, $t1
add $t0, 2
jmp loop
done:
```

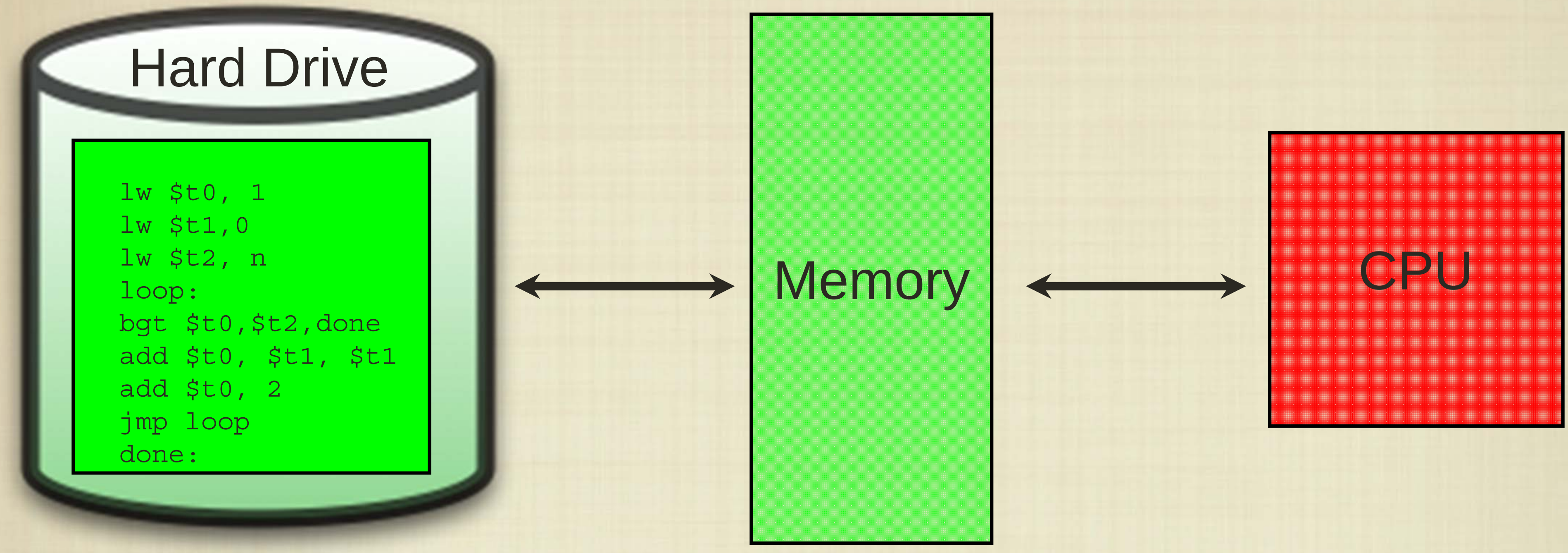
Memory

CPU




SATA Connector (x4)
BIOS Flash Chip in PLCC Socket
Southbridge (with heatsink)
Floppy Drive Connector
IDE Connector (x2)
CMOS Backup Battery
24-pin ATX Power Connector
Integrated graphics processor (with heatsink)
Super IO Chip
PCI Slot (×3)
DIMM Memory Slots (×4)
CPU Fan Connector
CPU Fan & Heatsink Mount
Integrated audio codec chip
Integrated Gigabit Ethernet chip
CPU Socket (Socket 939)
PCI Express Slot
Connectors For Integrated Peripherals
PS/2 Keyboard and Mouse, Serial Port, Parallel Port, VGA, Firewire/IEEE 1394a, USB (×4), Ethernet, Audio (×6)

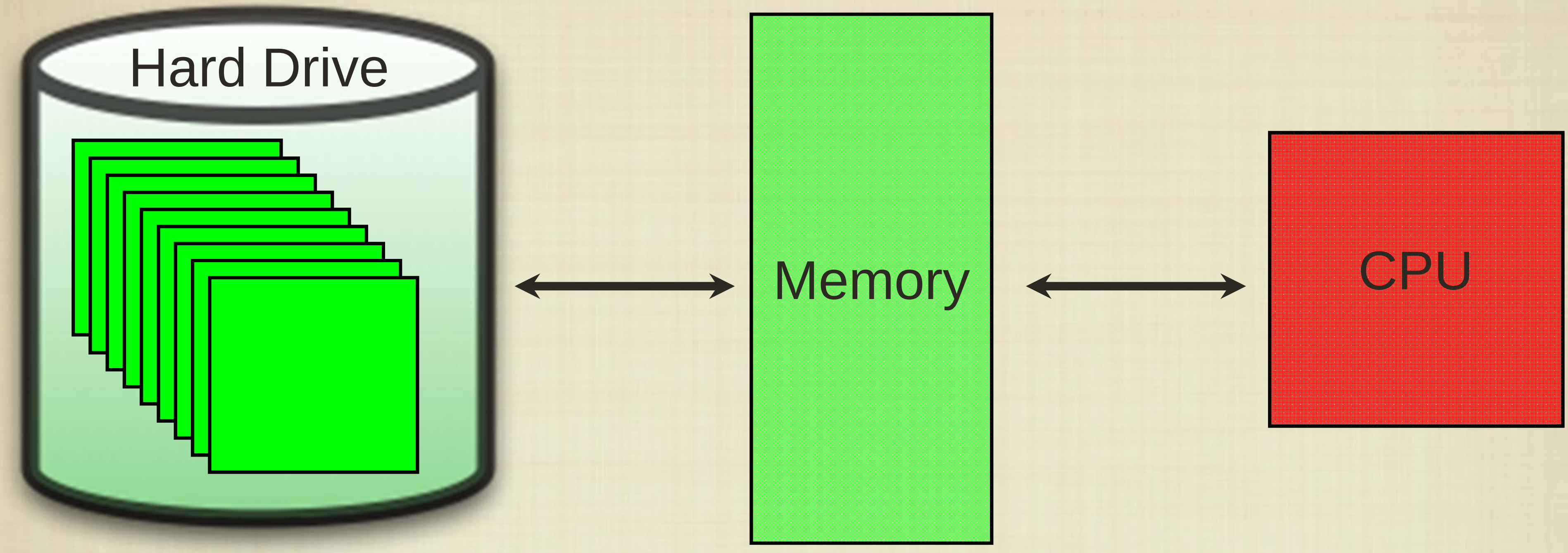
Where do programs “live”?



```
lw $t0, 1
lw $t1,0
lw $t2, n
loop:
bgt $t0,$t2,done
add $t0, $t1, $t1
add $t0, 2
jmp loop
done:
```

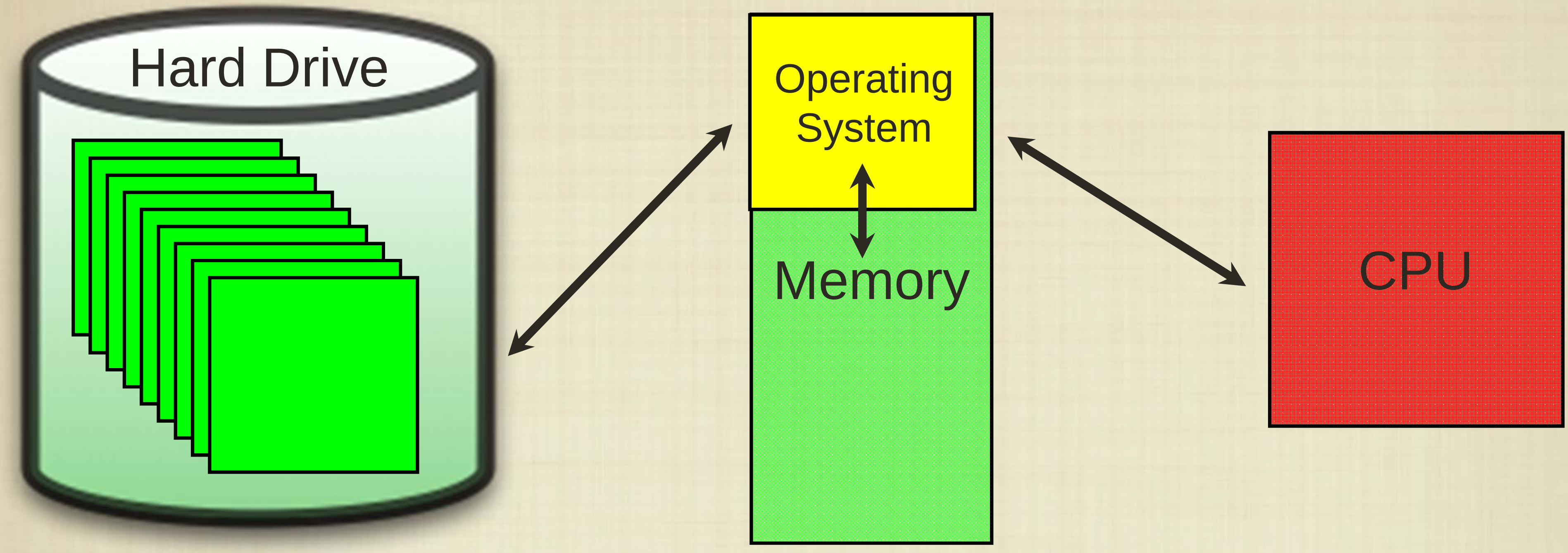
How is the program executed once it is stored on the disk drive?

Where do programs “live”?



How is the program executed once it is stored on the disk drive?

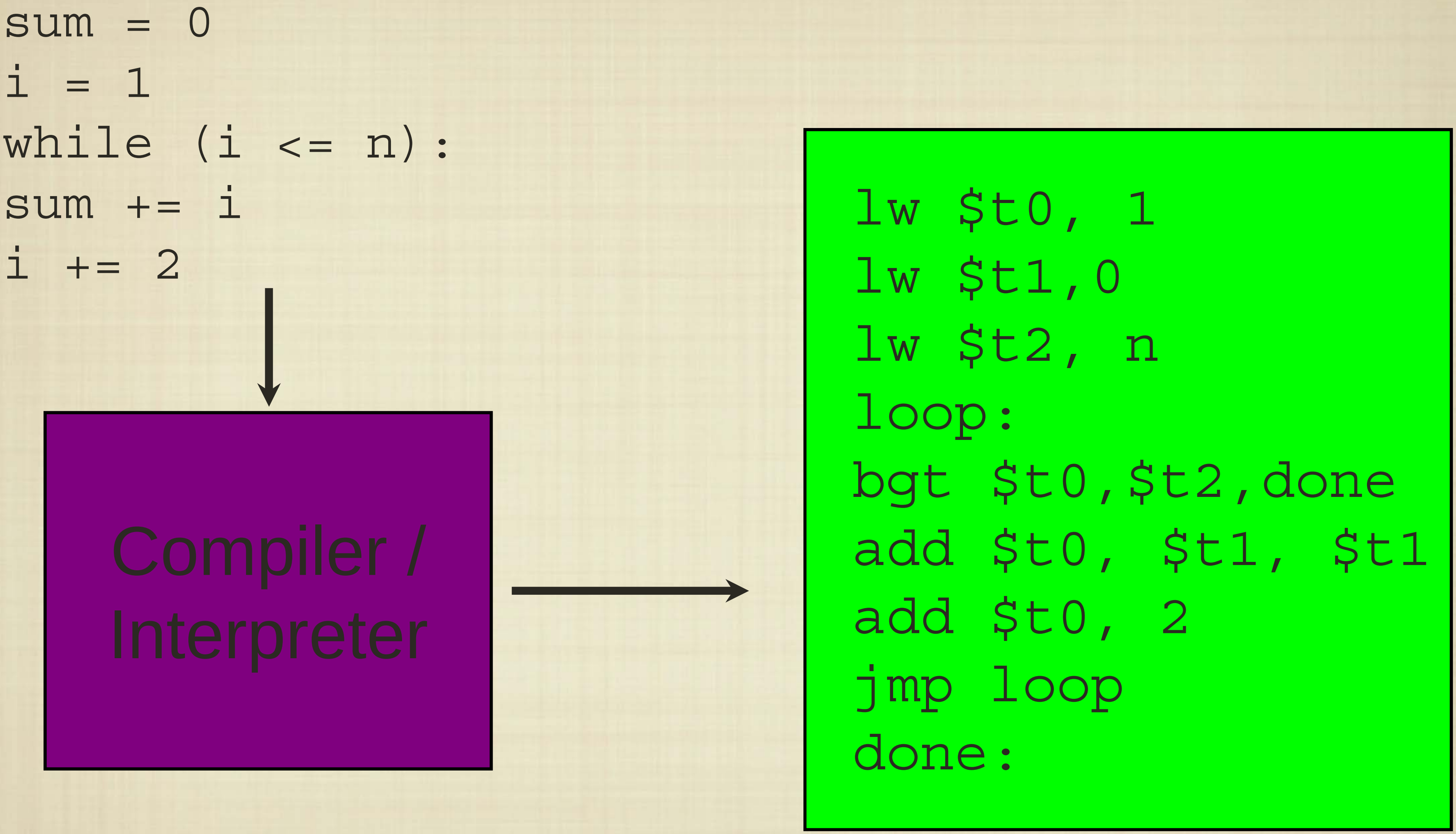
Where do programs “live”?



On modern computers, a program called the operating system is in charge of running one or more programs on the CPU.

Each software program being executed is given appropriate access to system resources (e.g., memory, disk, I/O).

Creating Machine Instructions



```
sum = 0
i = 1
while (i <= n):
sum += i
i += 2
```

Compiler / Interpreter

```
lw $t0, 1
lw $t1,0
lw $t2, n
loop:
bgt $t0,$t2,done
add $t0, $t1, $t1
add $t0, 2
jmp loop
done:
```

Compilers are CPU-specific programs that translate from high-level language to machine code.

Does the Language Matter?

Python

```
sum = 0
i = 1
while (i <= n):
sum += i
i += 2
```

Python
Interpreter

```
lw $t0, 1
lw $t1,0
lw $t2, n
loop:
beq $t0,$t2,done
add $t0, $t1, $t1
add $t0, 2
jmp loop
done:
```

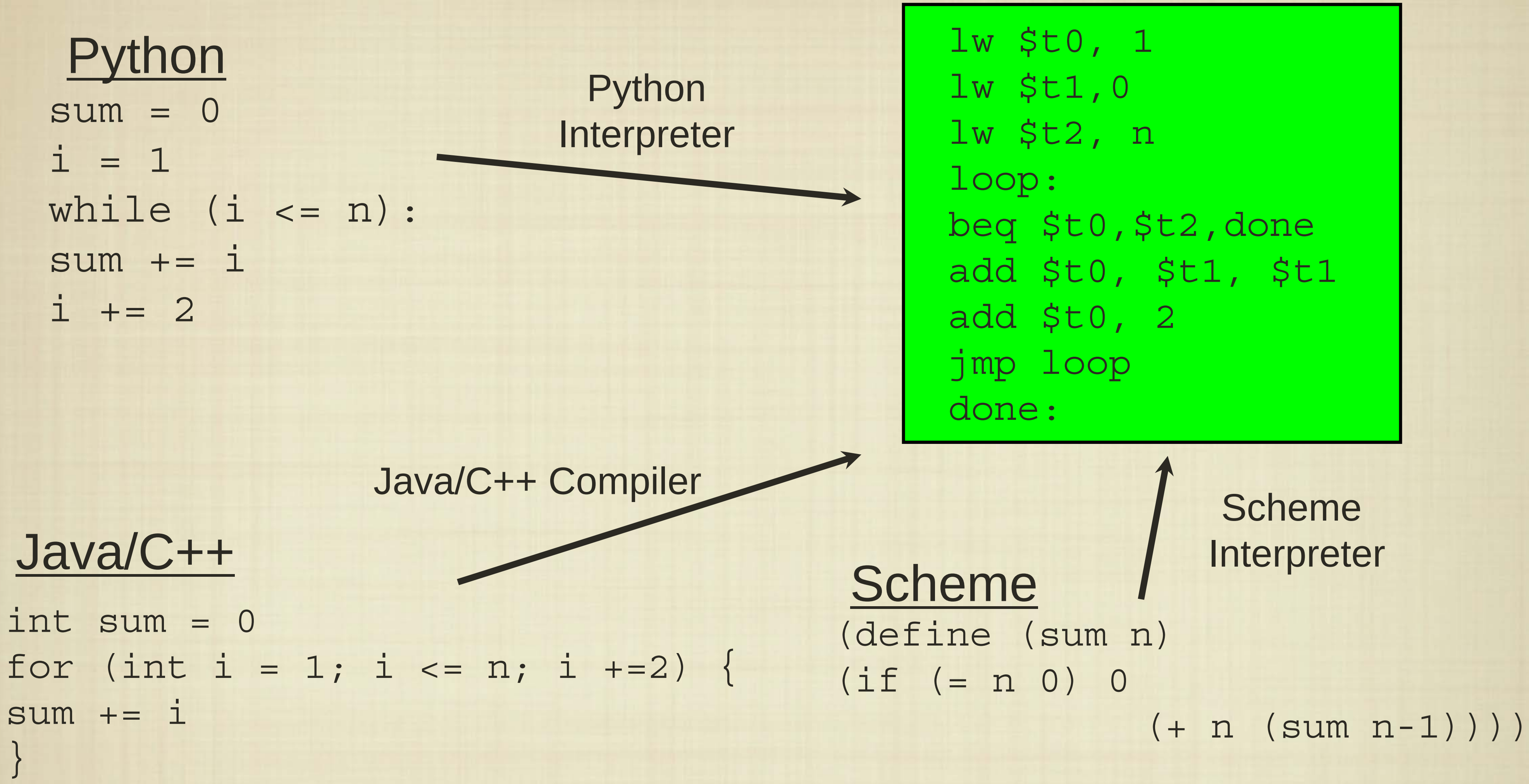
Java/C++

```
int sum = 0
for (int i = 1; i <= n; i +=2) {
sum += i
}
```

Scheme

```
(define (sum n)
(if (= n 0) 0
             (+ n (sum n-1))))
```

Does the Language Matter?



Python

```
sum = 0
i = 1
while (i <= n):
sum += i
i += 2
```

Python Interpreter

```
lw $t0, 1
lw $t1,0
lw $t2, n
loop:
beq $t0,$t2,done
add $t0, $t1, $t1
add $t0, 2
jmp loop
done:
```

Java/C++ Compiler

Java/C++

```
int sum = 0
for (int i = 1; i <= n; i +=2) {
sum += i
}
```

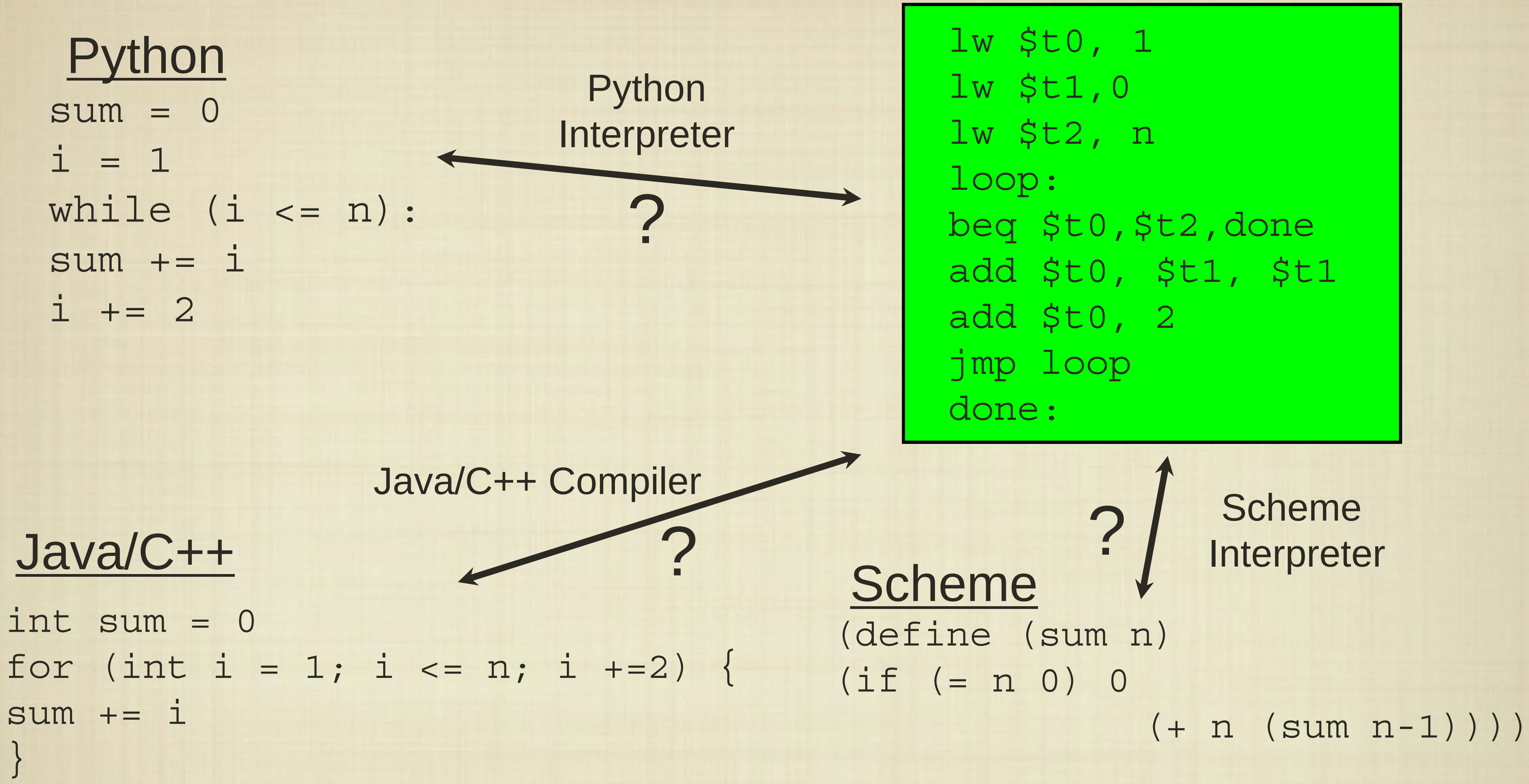
Scheme

```
(define (sum n)
(if (= n 0) 0
                    (+ n (sum n-1))))
```

Scheme Interpreter

Does every machine program have a corresponding high-level version (in every language)?

Does the Language Matter?



Python

```
sum = 0
i = 1
while (i <= n):
sum += i
i += 2
```

Python Interpreter

?

```
lw $t0, 1
lw $t1,0
lw $t2, n
loop:
beq $t0,$t2,done
add $t0, $t1, $t1
add $t0, 2
jmp loop
done:
```

Java/C++ Compiler

?

Java/C++

```
int sum = 0
for (int i = 1; i <= n; i +=2) {
sum += i
}
```

?

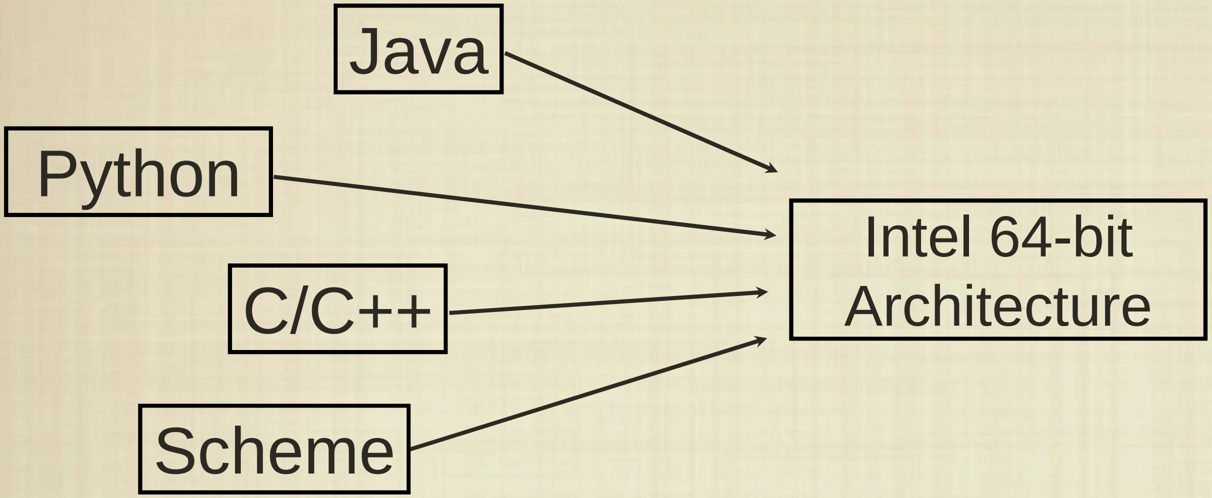
Scheme Interpreter

Scheme

```
(define (sum n)
(if (= n 0) 0
                    (+ n (sum n-1))))
```

Does every machine program have a corresponding high-level version (in every language)?

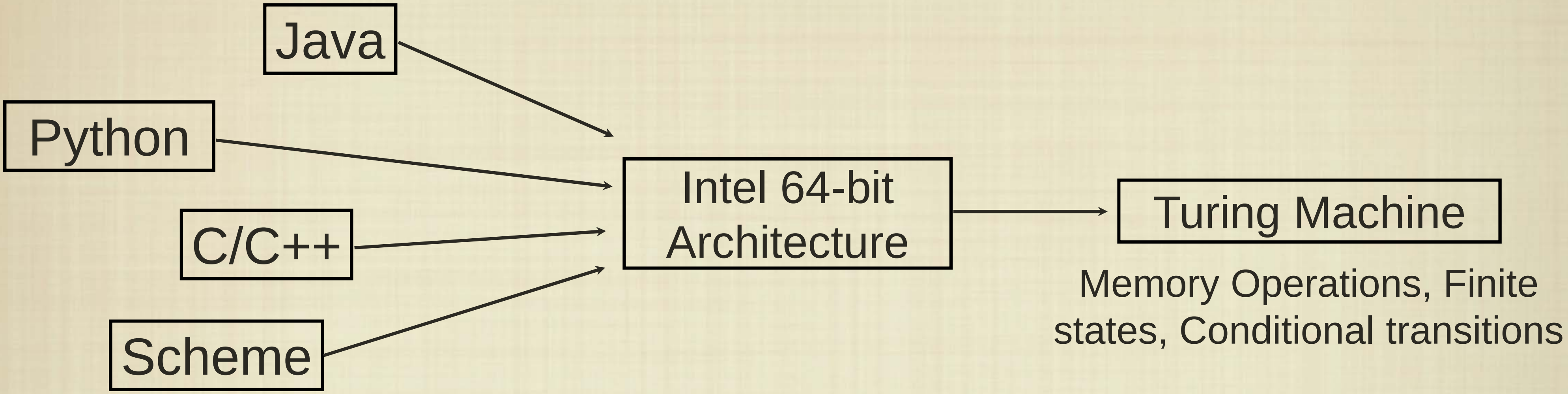
Different Languages



Any program written in a high-level language can be converted into machine instructions that are executed on a von Neumann architecture.

⇒ But is a “machine language” more or less powerful than a high-level programming language?

Different Languages



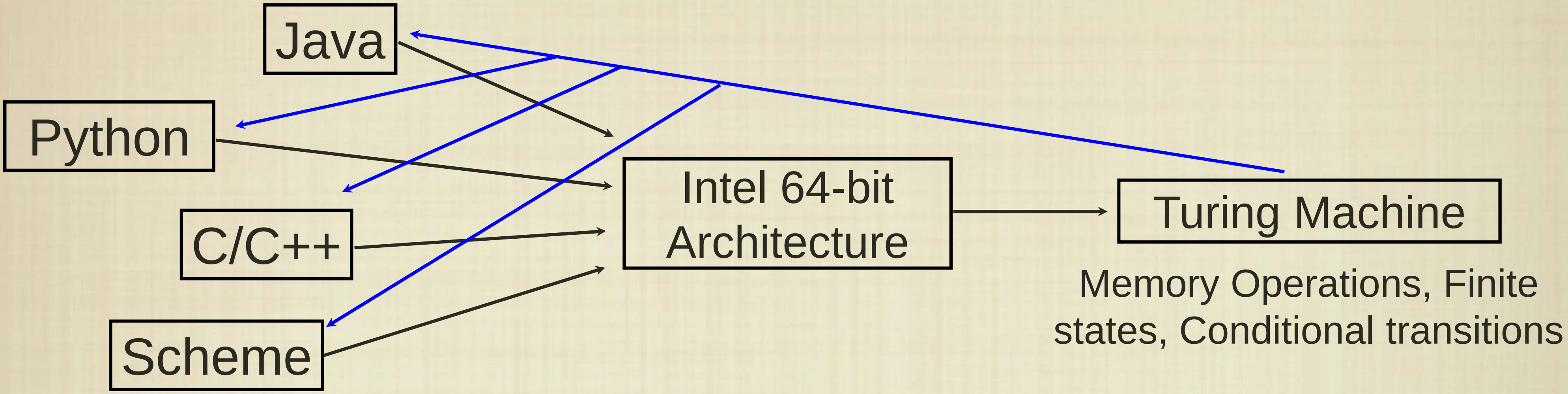
Memory Operations, Finite states, Conditional transitions

Any program written in a high-level language can be converted into machine instructions that are executed on a von Neumann architecture.

Every von Neumann machine implements a Turing machine.

⇒ Can every language implement a Turing machine? If so, all languages would be equally powerful.

Different Languages

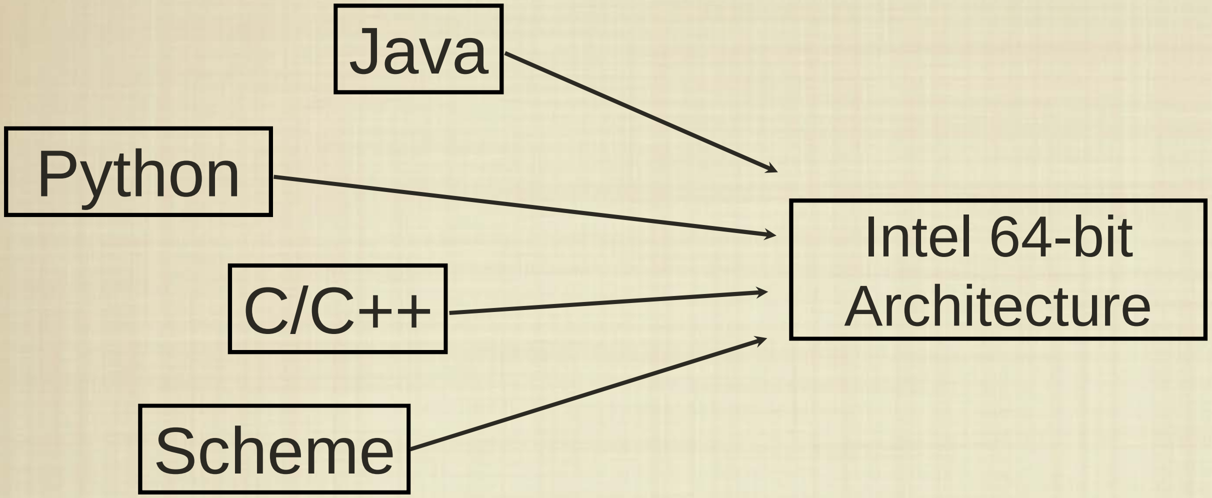

Java
Python
C/C++
Scheme
Intel 64-bit Architecture
Turing Machine
Memory Operations, Finite states, Conditional transitions

Any program written in a high-level language can be converted into machine instructions that are executed in a von Neumann architecture.

Every von Neumann machine implements a Turing machine.

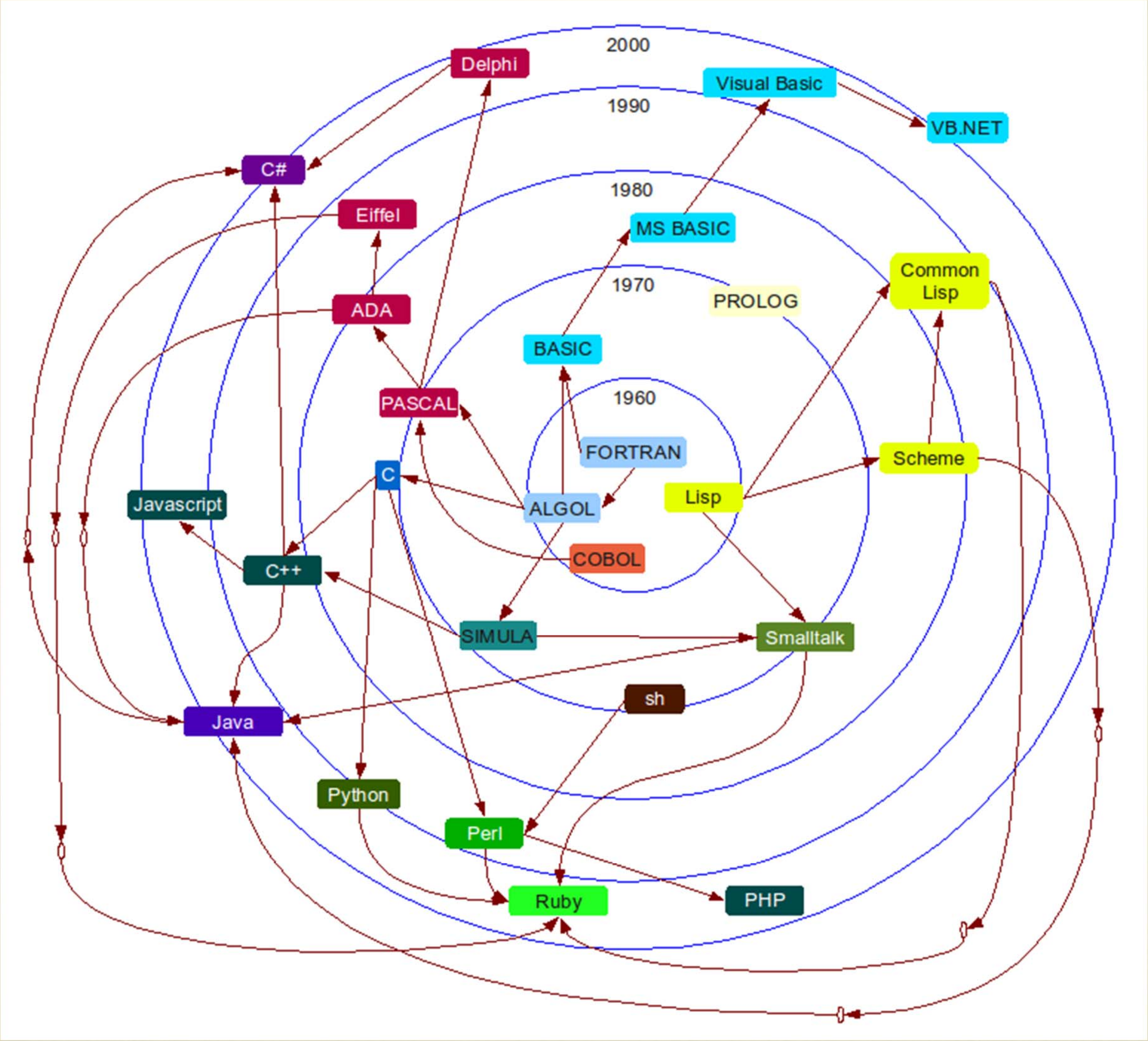
Every high-level language is "Turing-complete", and so languages have essentially the same descriptive power as one another.

Different Languages



In fact, every modern language can utilize every machine instruction, and the primary differences between languages are in syntax and expressiveness.

However, there is often a tradeoff: the more “low-level” a language, the better the performance.


2000
Delphi
Visual Basic
VB.NET
1990
C#
Eiffel
1980
MS BASIC
Common Lisp
1970
PROLOG
ADA
BASIC
PASCAL
1960
FORTRAN
Scheme
C
Lisp
Javascript
ALGOL
COBOL
C++
SIMULA
Smalltalk
sh
Java
Python
Perl
Ruby
PHP

Categories and Uses

- Imperative
  - Python: Interpreted, Easy to prototype ideas
  - Java : Interpreted/Compiled, Platform-independent
  - C/C++: Compiled, General purpose
  - PHP: Interpreted/Compiled, Web scripting

- Functional
  - LISP/Scheme: Interpreted, no differentiation between data/instructions

Languages are translated to machine code by either a compiler or interpreter.