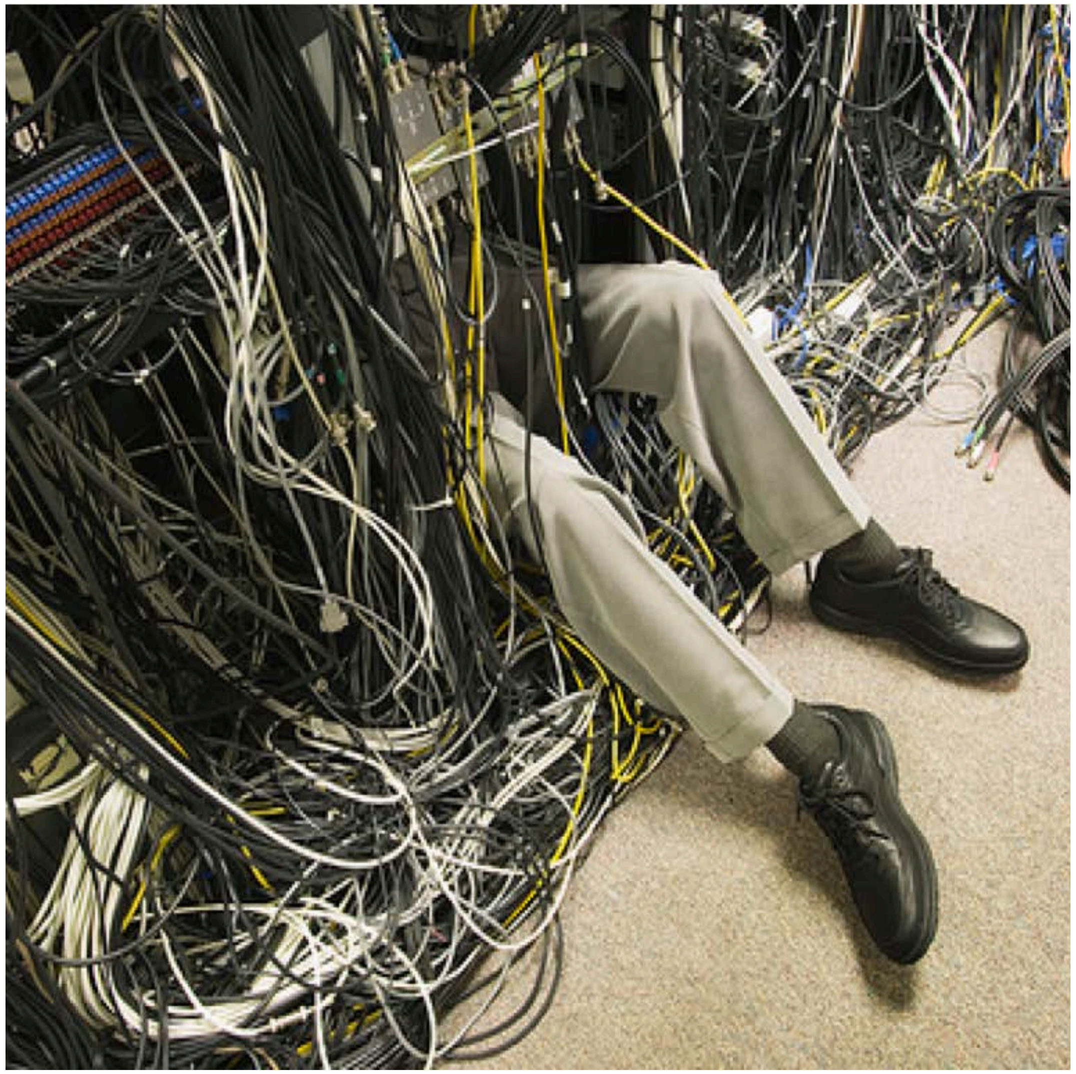
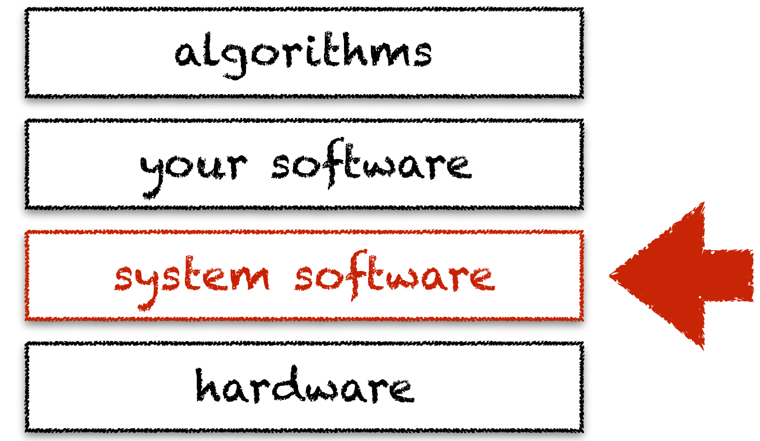
system software



learning objectives



- understand the role of an operating system
- understand the role of interpreters and compilers
- understand the role of runtime systems & libraries

what's system software?

application software consists in programs that help to solve a particular computing problem, e.g., write documents, browse the web, etc.

system software consists in programs that sit between application software and the hardware, providing common services to application software

examples of system software

- operating systems, game engines
- virtual machines and interpreters
- language runtimes, standard libraries

bits of history

- 1940s 1950s — no system software
- 1960s — batch systems

the waiting era

- 1970s — multi-user & time-sharing

the sharing era

- 1980s — personal desktop computers

the personal era

- 1990s — distributed systems

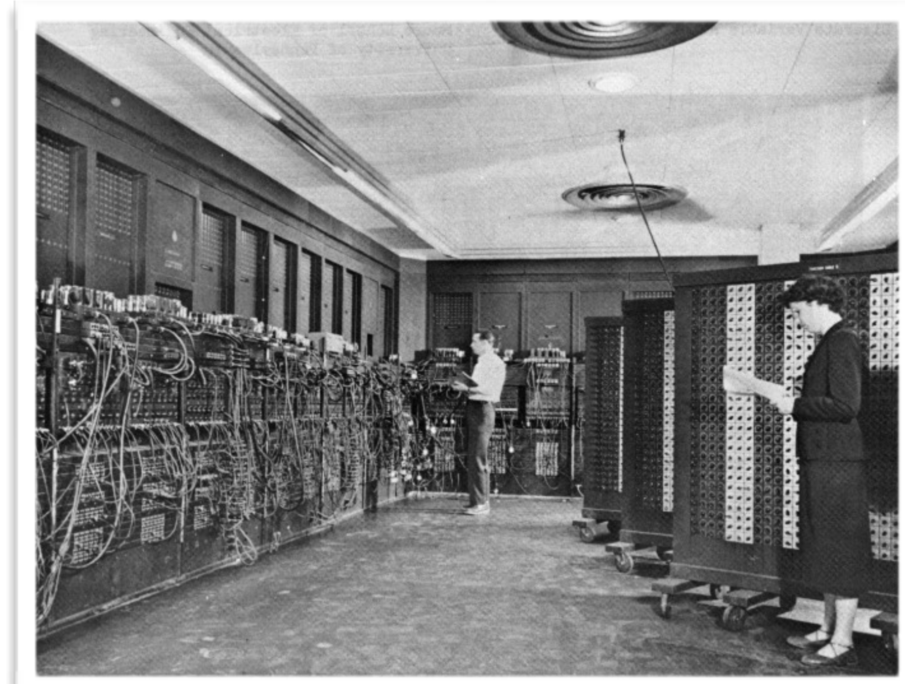
the communication era

- 2000s — mobile systems
- 2010s — ubiquitous systems

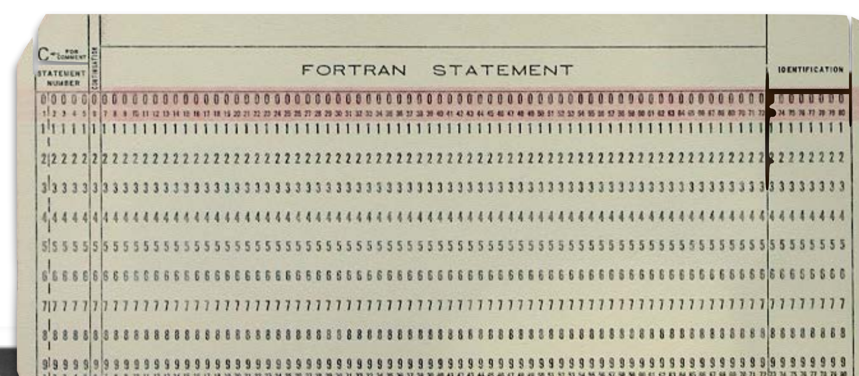
the digital transformation era

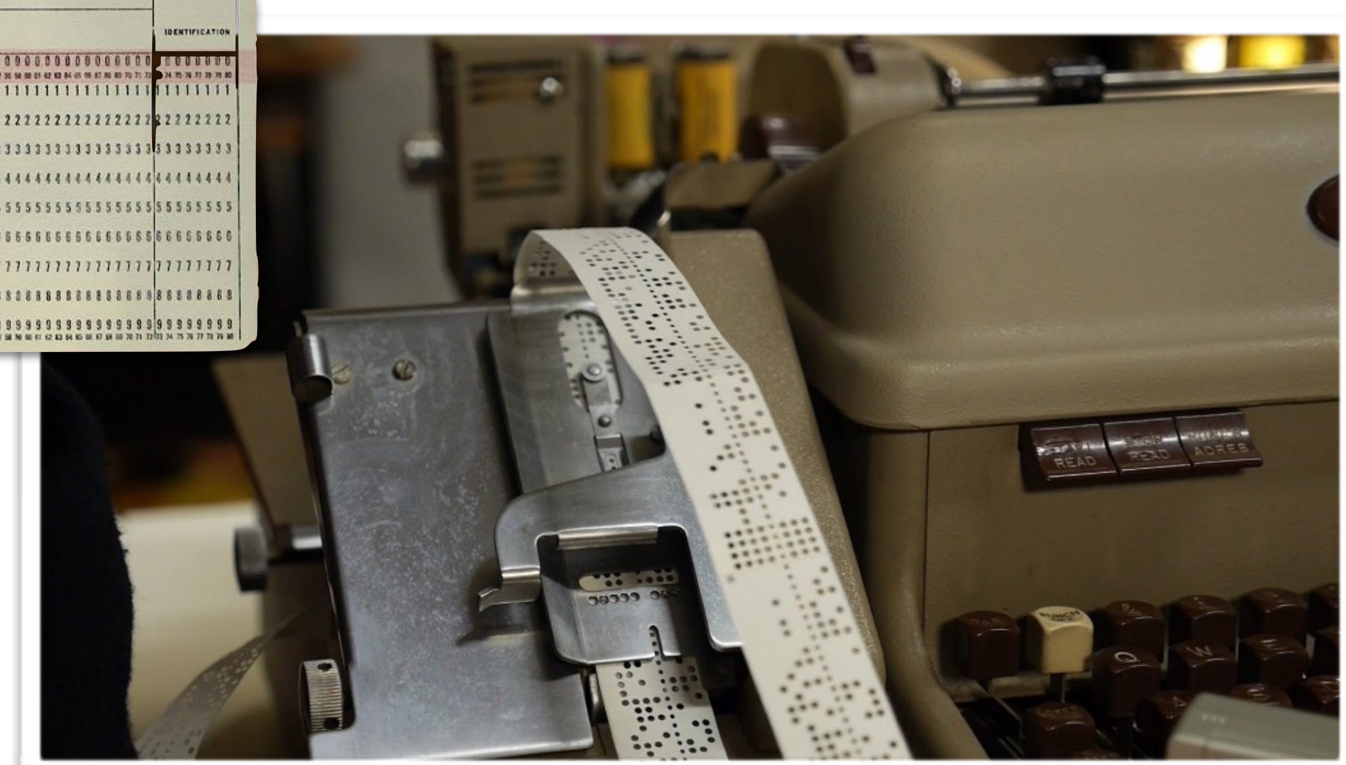
no systems software

- 1940s: programming based on dials & switches
- 1950s: single user, punched cards, paper tape



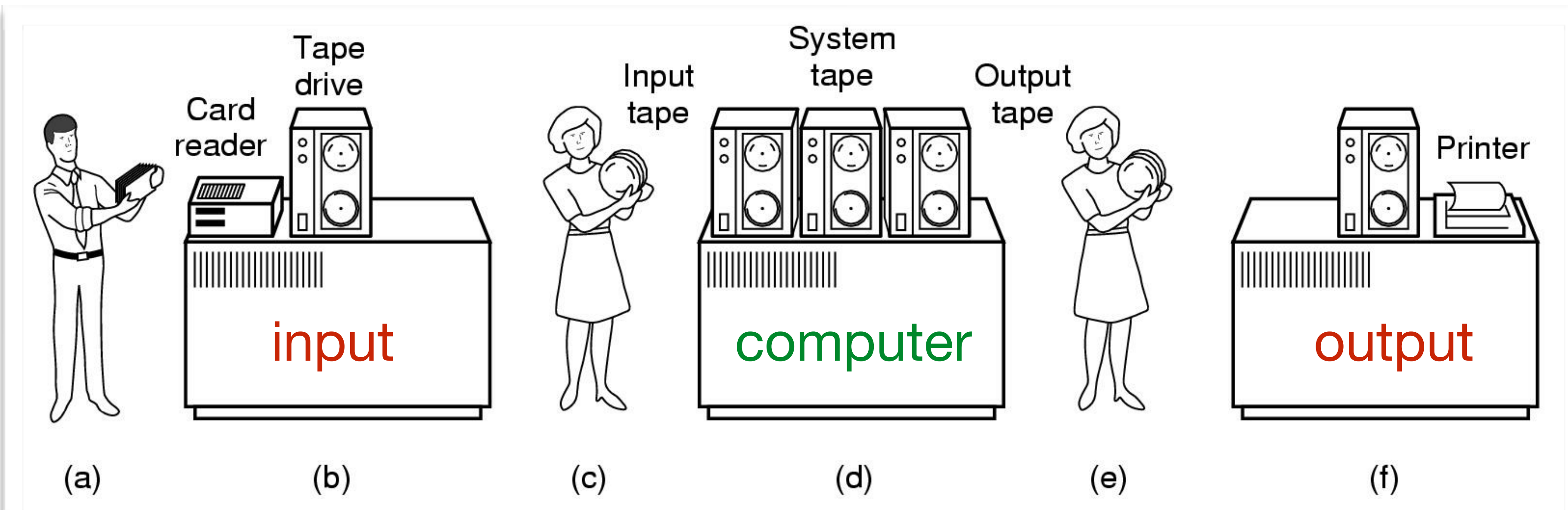
ENIAC: 30 tons, 200 kilowatts





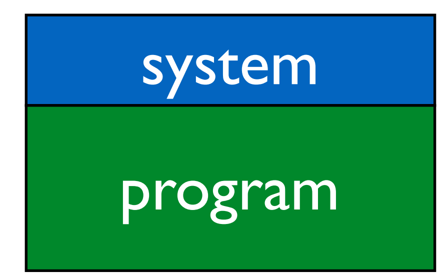
1960s

batch systems

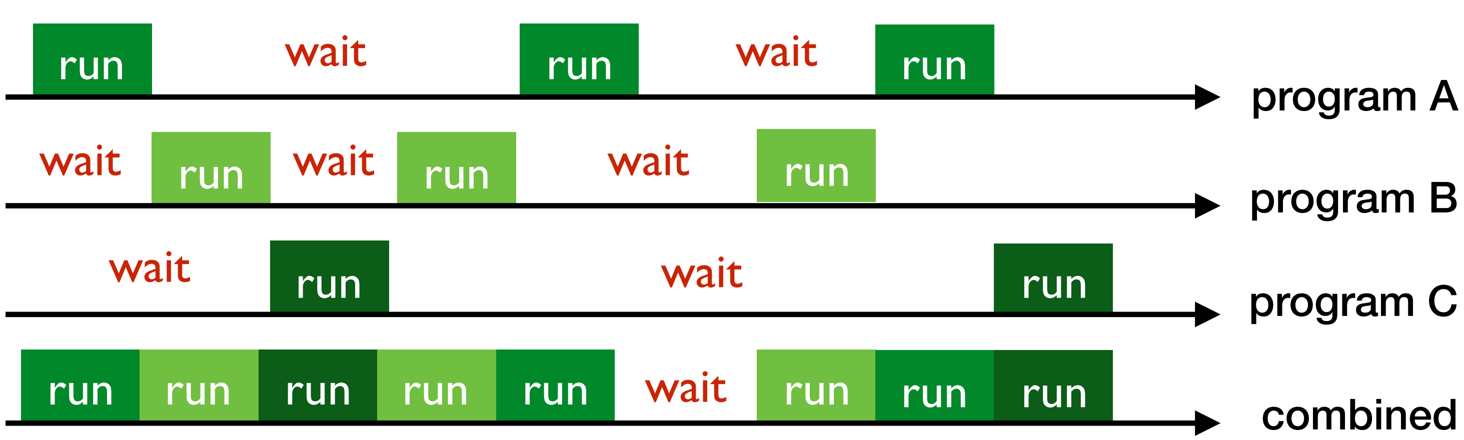


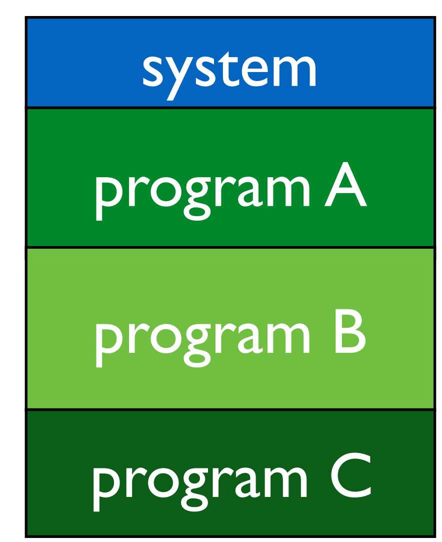
(a) put cards into reader
(b) read cards to tape
(c) put input tape on computer
(d) perform the computation
(e) put output tape on printer
(f) print output tape on paper

- first uni-programmed batch systems



- then multi-programmed batch systems





multi-user & time-sharing

from Multics...

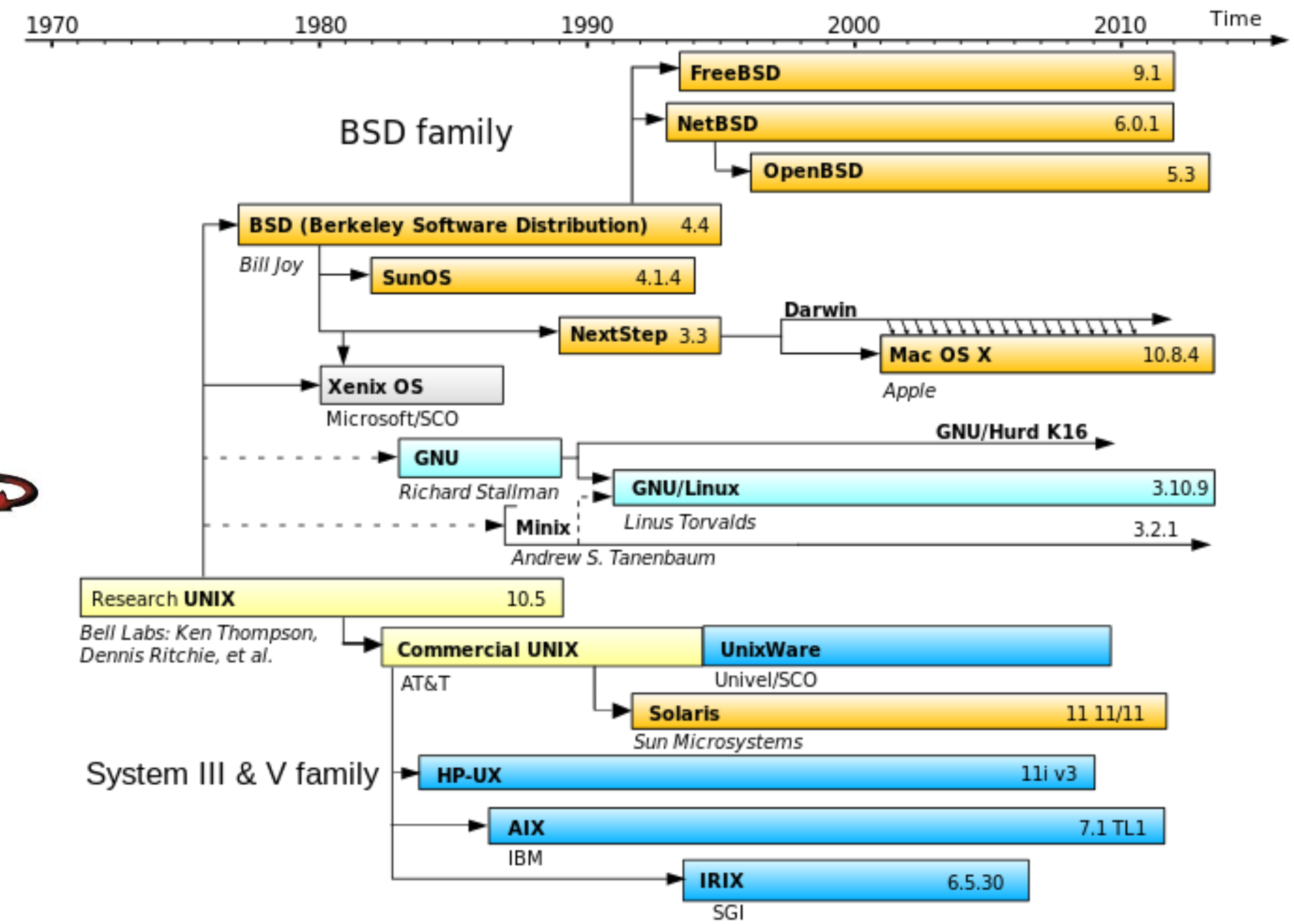
Ken Thompson & Dennis Ritchie @ Bell Labs

...to Unix

- 1960s: disasters... but great learning & innovations
  - OS/360: years behind schedule, shipped with 1000 known bugs
  - Multics: started in 1963, working in 1969, far too complex

- 1970s: finally mastering complexity thanks to:
  - higher level structured languages (Algol, C, Pascal, etc.)
  - portable operating systems code (C was invented for that)
  - stacking layers (kernel, compilers, libraries, etc.)

Unix



- after the Multics "disaster", Ken Thompson, Dennis Ritchie & others decided to redo the work on a much smaller scale at Bell Labs
- in 1972, Unix was rewritten from assembly language to C programming language, resulting in the first portable operating system
- in 1975, Ken Thompson was on sabbatical at Berkeley and worked with Bill Joy, then a graduate student, which eventually lead to BSD Unix
- in 1980, the DARPA project chose BSD Unix as basis for DARPANet
- in 1982, Bill Joy joined Sun Microsystems six months after its creation as full co-founder and extended BSD Unix to make it a networked operating system

microprocessors & Moore's law

a microprocessor is a computer processor integrating all functions of a central processing unit on a single chip

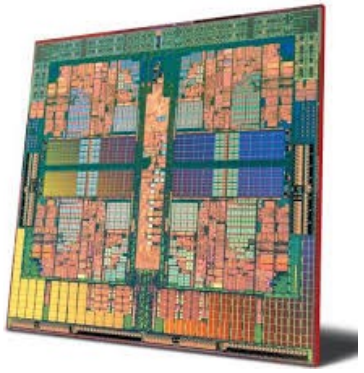
the number of transistors in a dense integrated circuit doubles approximately every two years

- this is unique across all engineering fields
- transportation increased speed from 20 km/h (horse) to 2'000 km/h (concorde) in 200 years but the computer industry has been doing this every decade for the past 60 years
- the advent of the microprocessor triggered the decline of mainframes and led to the personal computer revolution

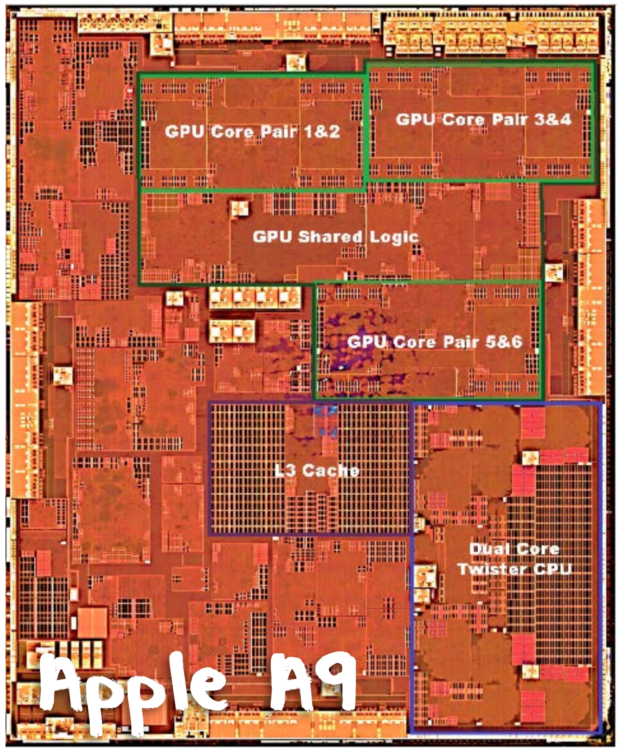
writing system software is about mastering exponential complexity

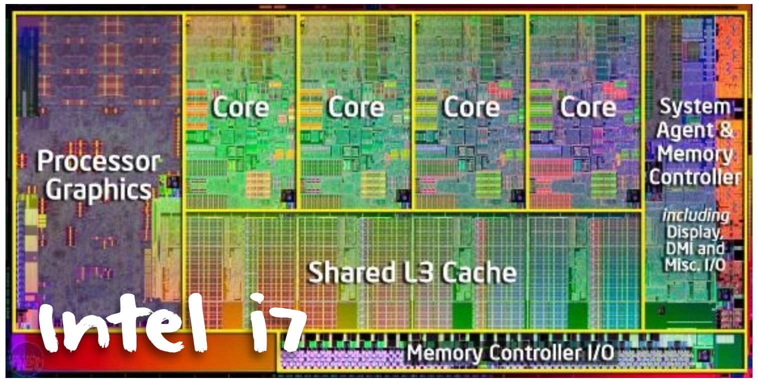
As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem and now that we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them - it has created the problem of using its products.

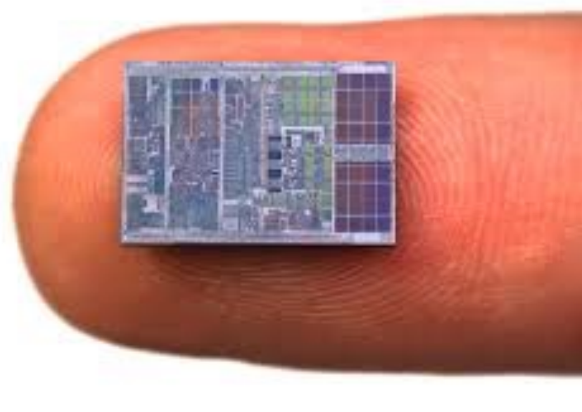
Edgster Dijkstra, The Humble Programmer. Communication of the ACM, vol. 15, no. 10. October 1972. Turing Award Lecture.



the industry is now going **multicore**







acceleration

1980s: one man, one computer

- workstation, personal computers
- graphical user interfaces

1990s: the network is the computer

- the internet accessible to all
- distributed operating systems

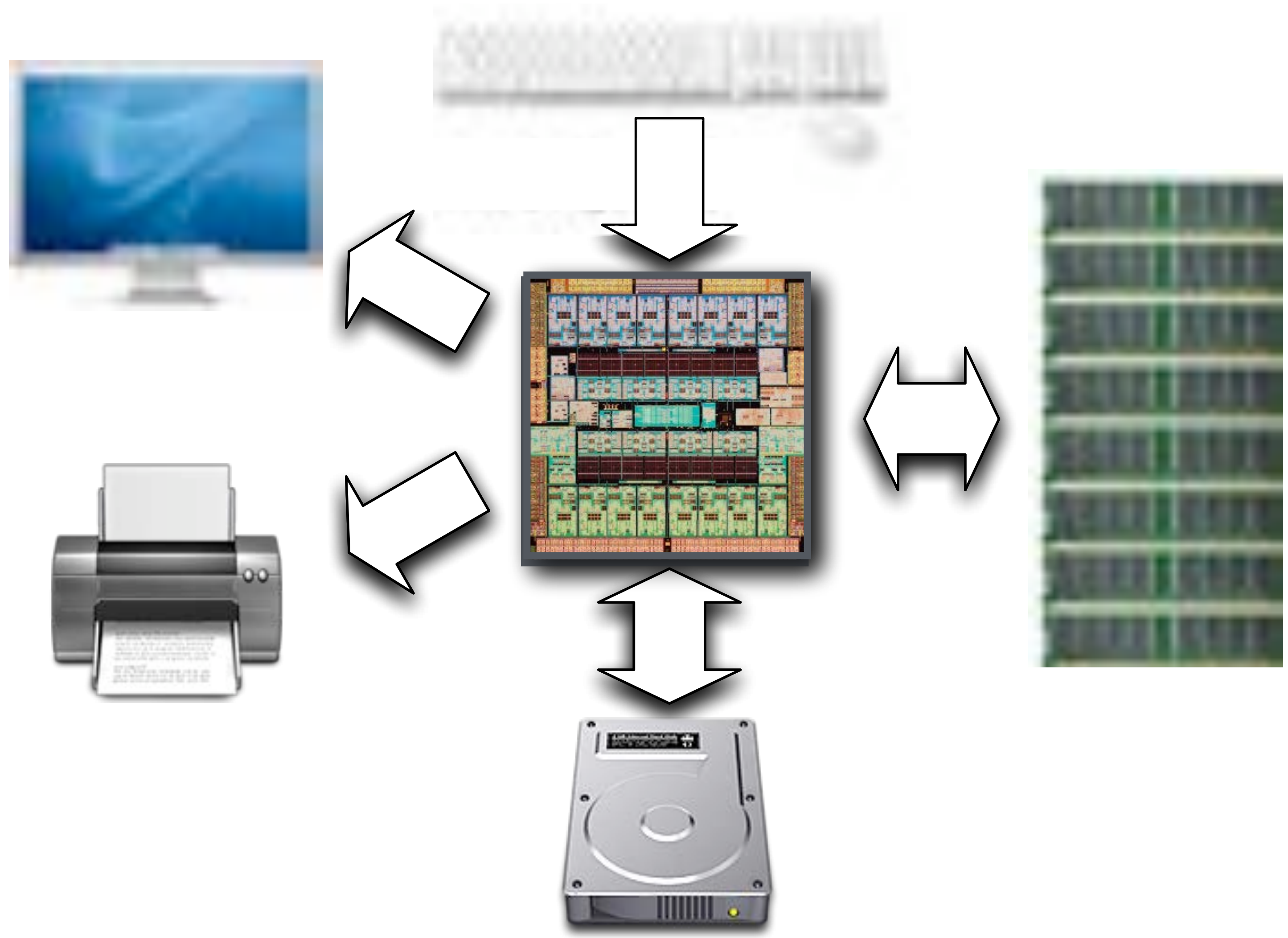
2000s: my phone is my computer

- smartphones & tablets as computers
- generalization of wireless networks

2010s: everything is a computer

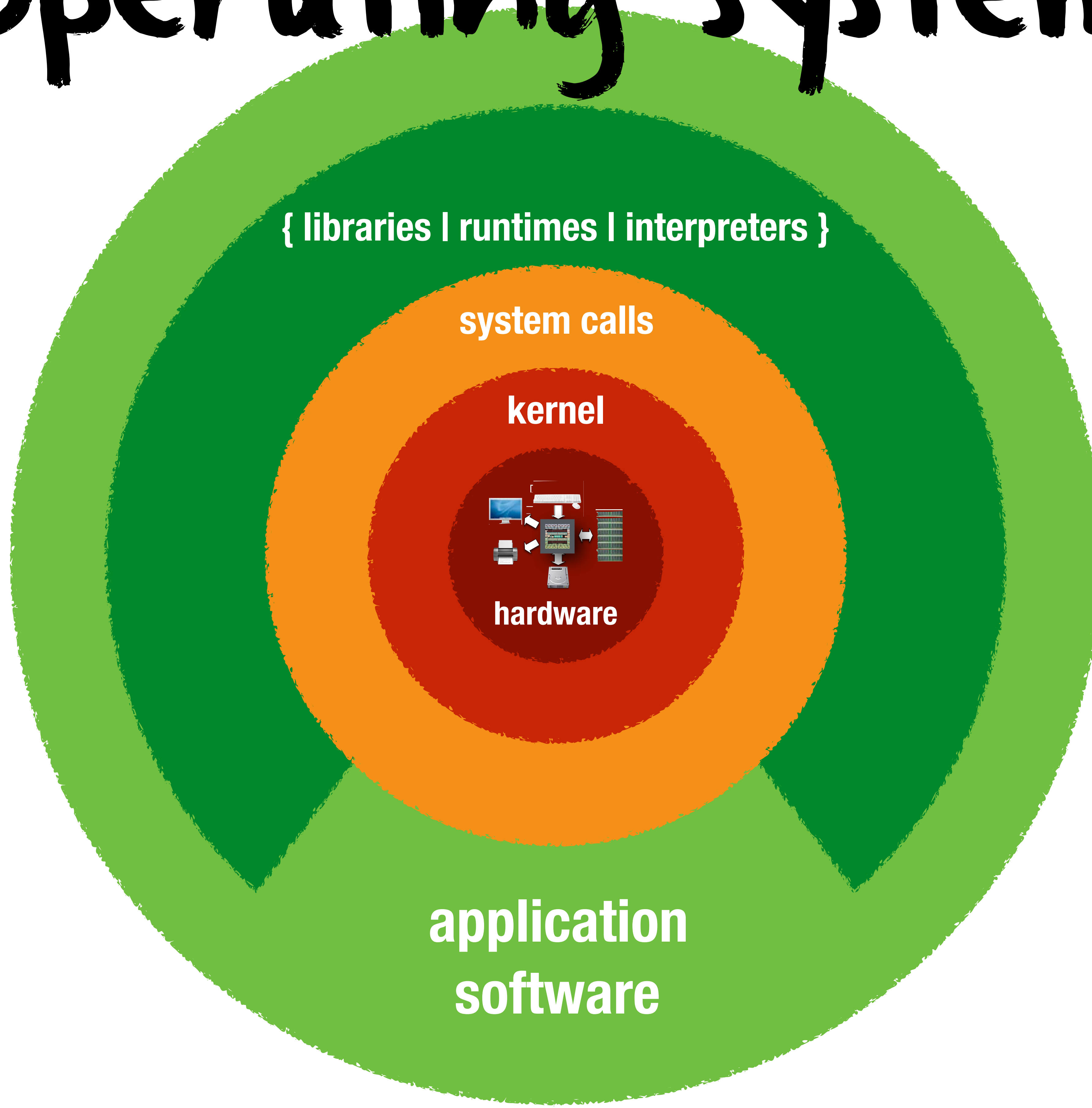
- smart objects & the internet of things
- personal networks connected to the cloud

operating system

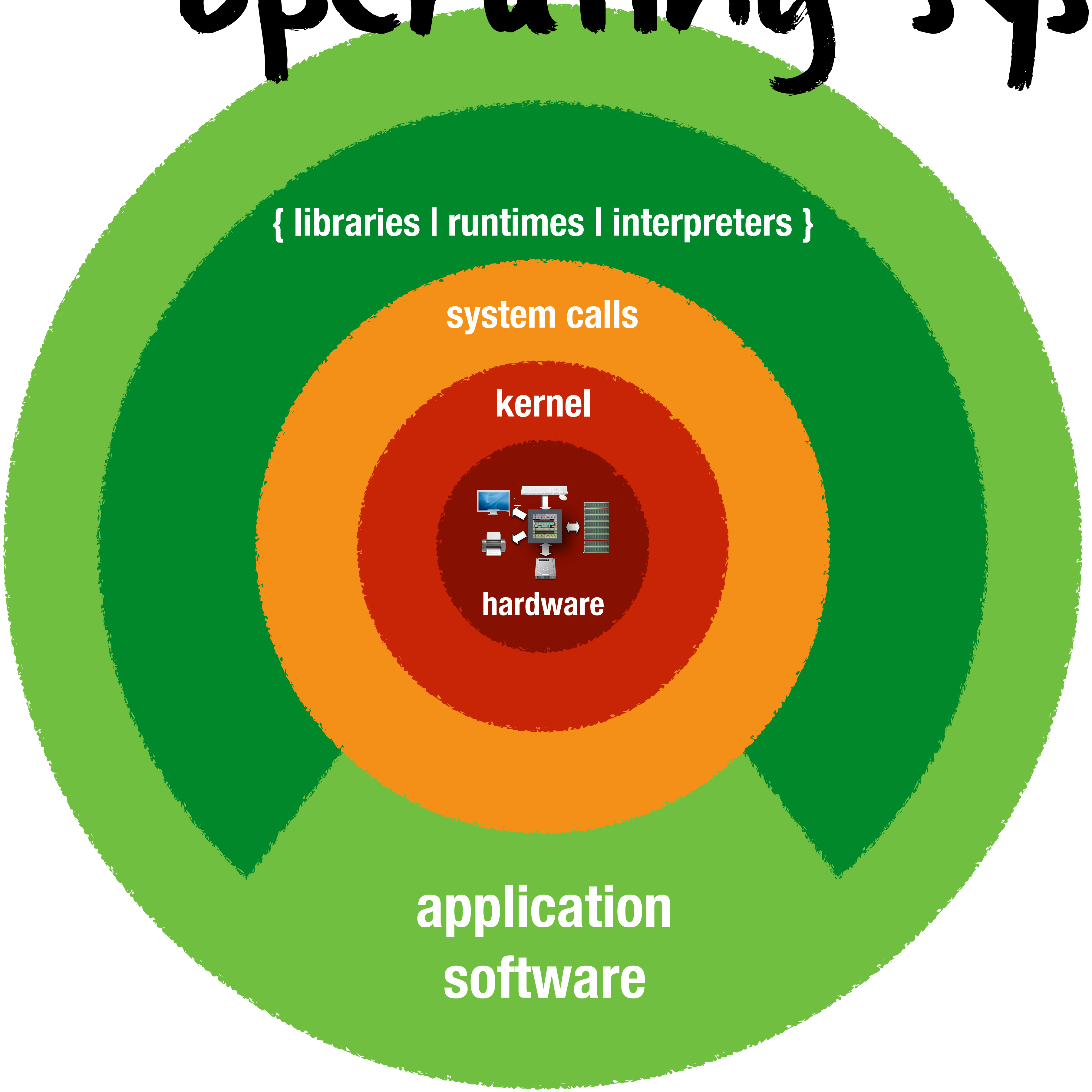


controls the access to hardware resources (cpu, memory, input/output devices, etc.) and acts as an interface with application software

operating system

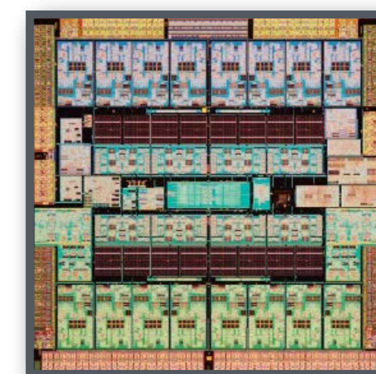


operating system

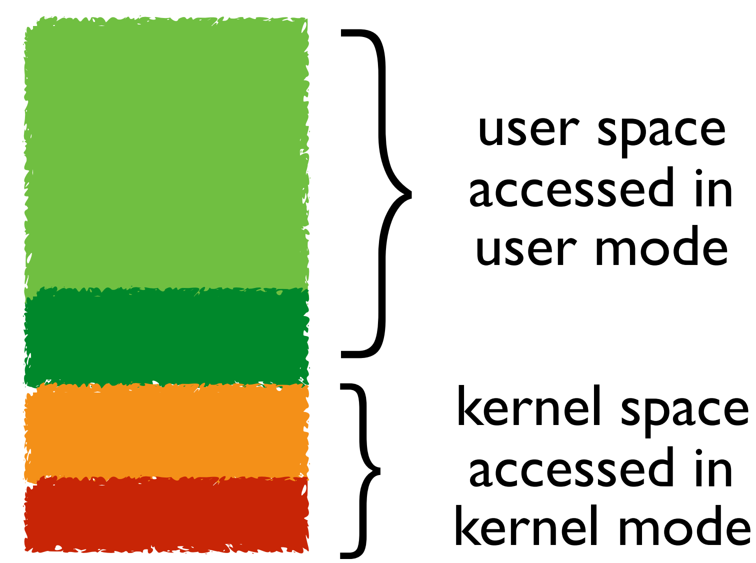


processor modes

- kernel mode (system)
- user mode (application)



memory protection



operating system

resources managed by operating systems

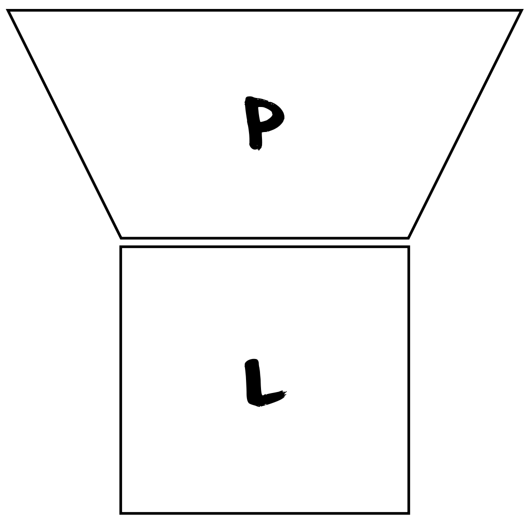
- **cpu:** process management
- **memory:** memory management
- **input/output:** i/o management
    - keyboard, mouse, display
    - touch screen, haptic interface, network
    - printer, audio device, connectors (usb, dvi, etc.)
    - compass, accelerometer, global positioning system
    - etc...
- **storage:** storage and file management

	reality (physical resources)	abstraction (virtual resources)
cpu	***n* parallel cores**	***m* concurrent threads**, *with* $m \gg n$
memory	**subset of 2^k addressable memory** *on a **k** bits machine, e.g., for* $k = 64$, *this is typically* **8 to 32 gigabytes**	**full 2^k addressable memory** *for* $k = 64$, *this is* **16 exabytes** $\cong 16 \times 10^6$ **terabytes** $\cong 16 \times 10^9$ **gigabytes**
	in addition, each thread can access the full **2k addressable memory** *as if it was for its exclusive use*	
storage	*hard disk drive (hdd), solid state drive (ssd), usb keys, etc…*	*file system offering persistency*
network	***i* network interfaces**, *e.g., wifi, ethernet*	***j* network connections**, *with with* $j \gg i$

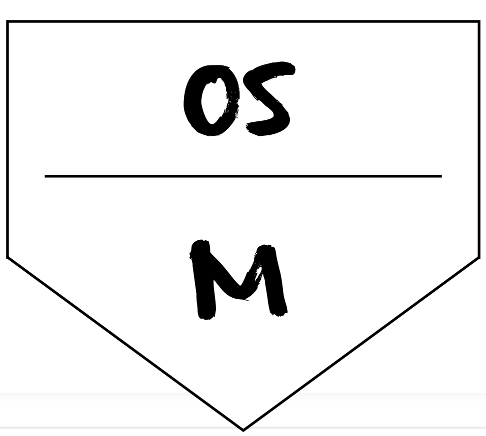
executions and interpreters

concept

program P
written in
language L

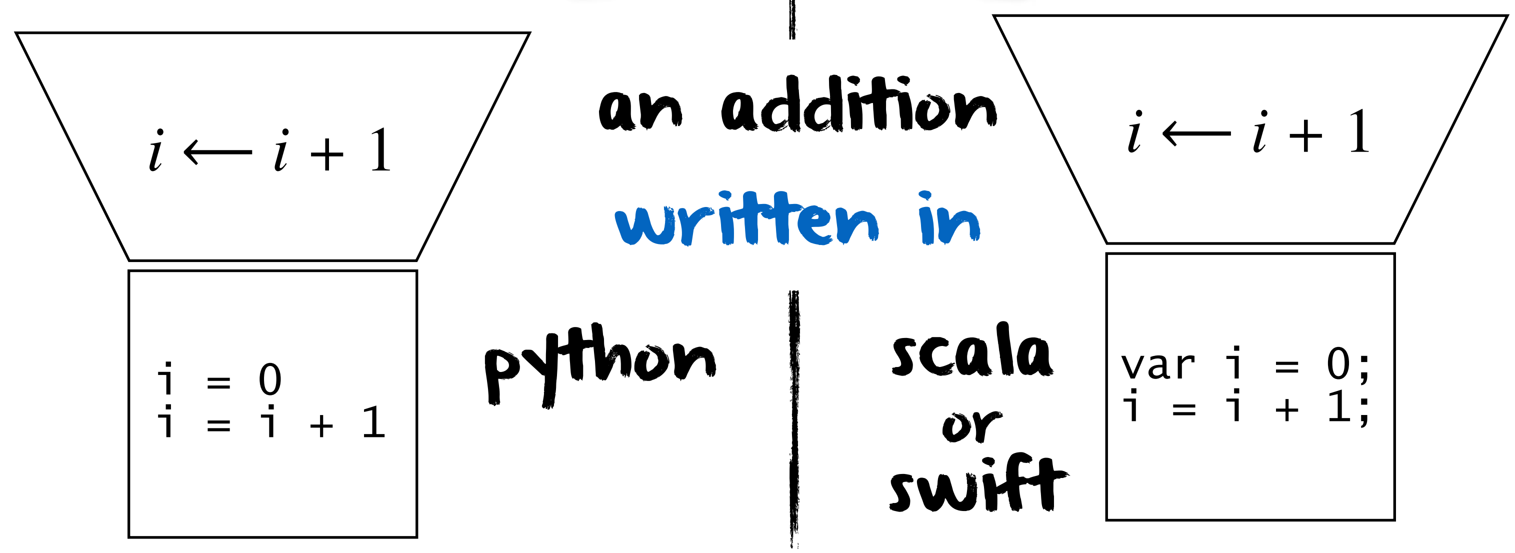


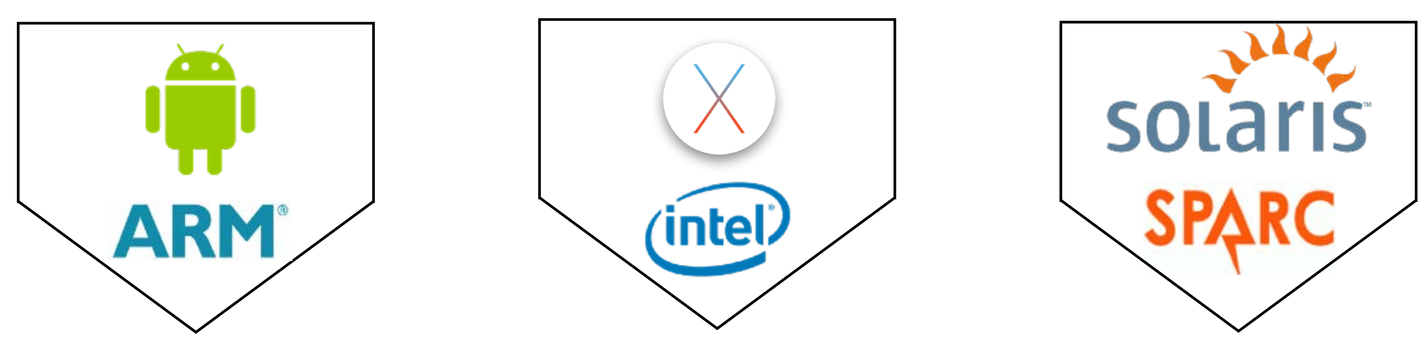
operating system OS
controlling machine
executing language M



examples

an addition
written in
python | scala or swift





Samsung S7 running Android on ARM

MacBook Pro running OS X on Intel

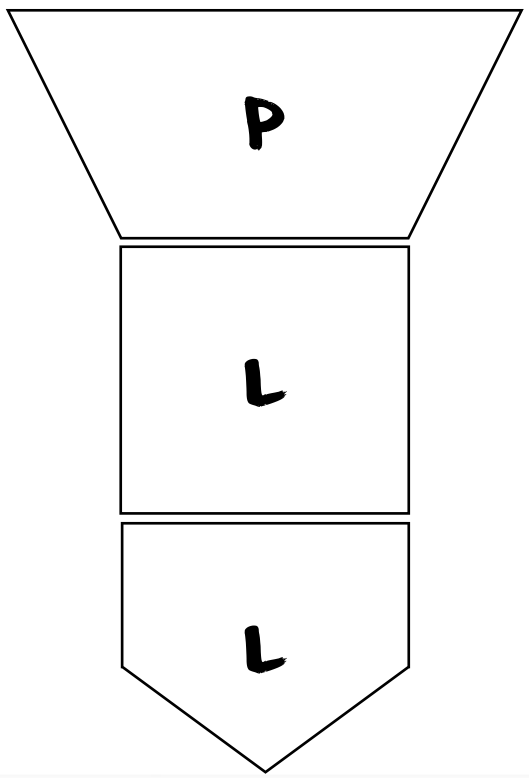
Oracle Server running Solaris on SPARC

machine language M ⇔ instruction set ⇔ byte code

executions and interpreters

concept

program P
written in
language L
running on
machine L



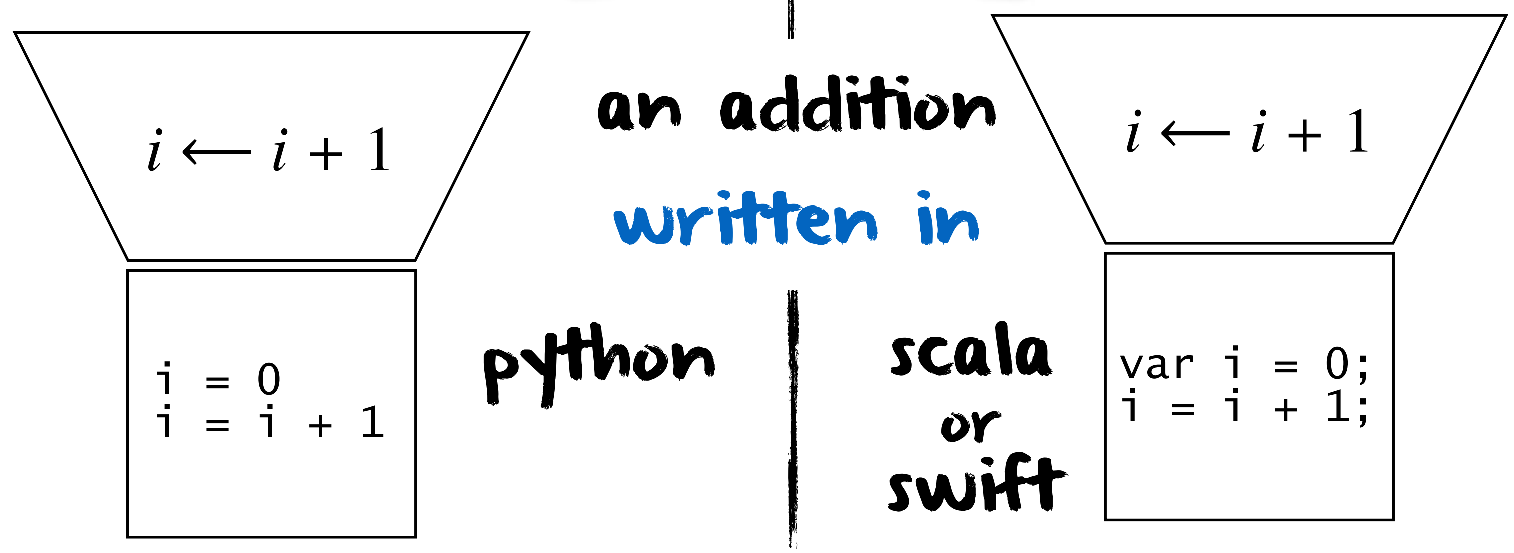
program language must match machine language

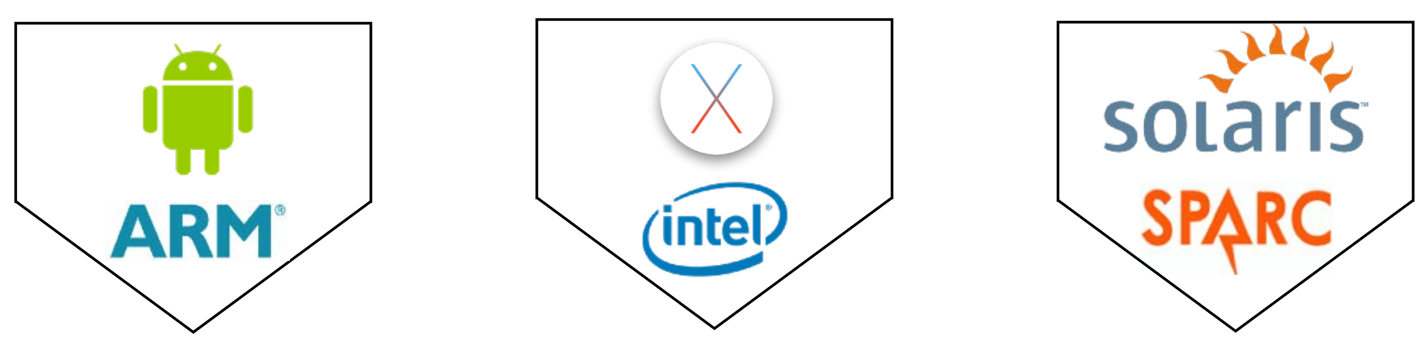
we forget about the operating system for now

examples

an addition
written in

python | scala or swift





Samsung S7 running Android on ARM

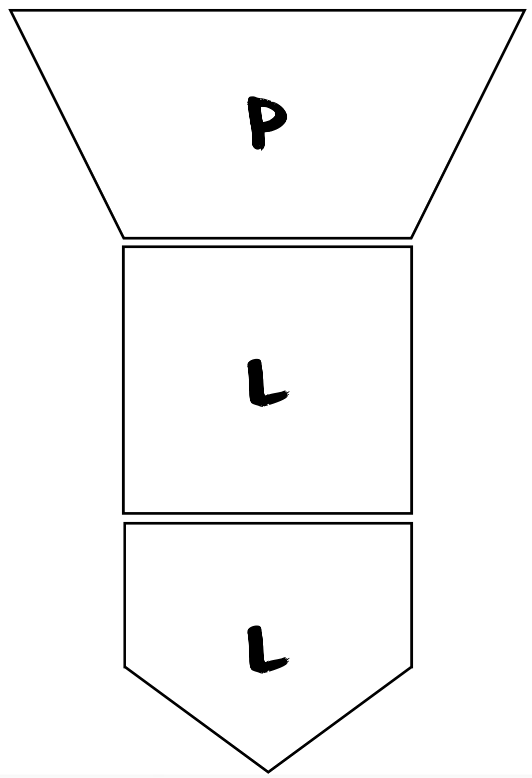
MacBook Pro running OS X on Intel

Oracle Server running Solaris on SPARC

executions and interpreters

concept

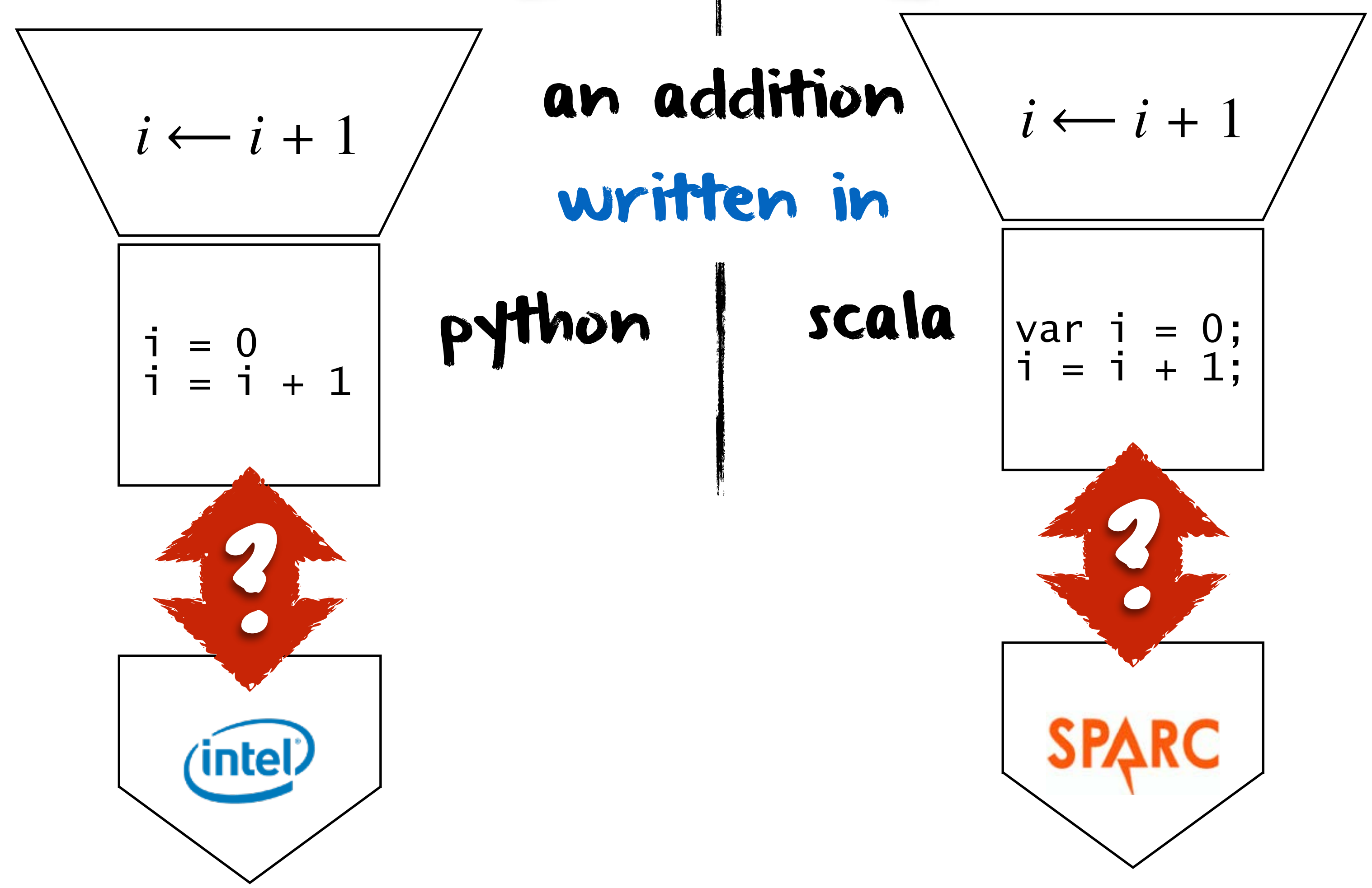
program P
written in
language L
running on
machine L



program language must match machine language

we forget about the operating system for now

examples

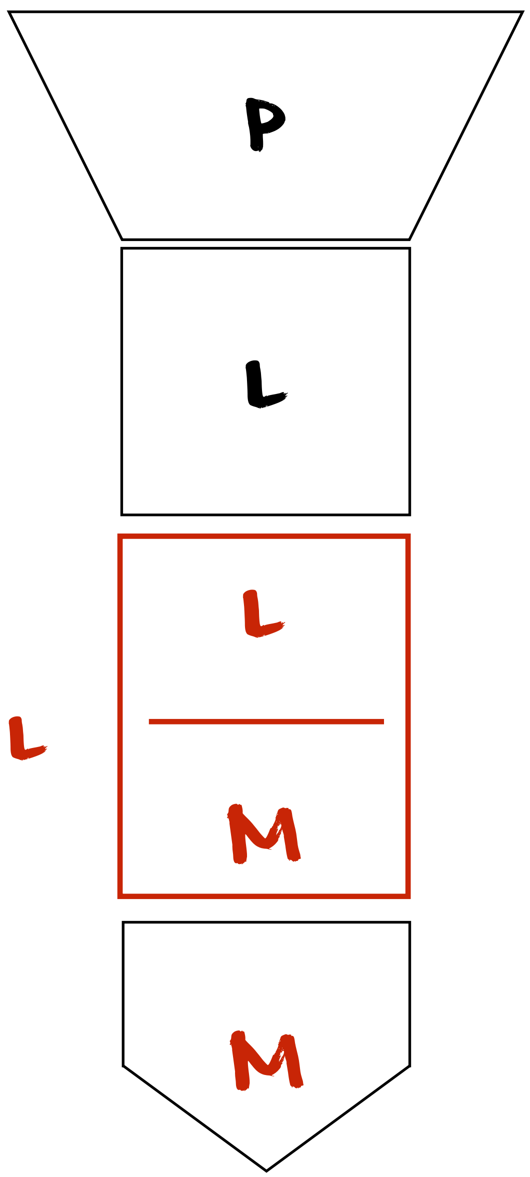


problem!

executions and interpreters

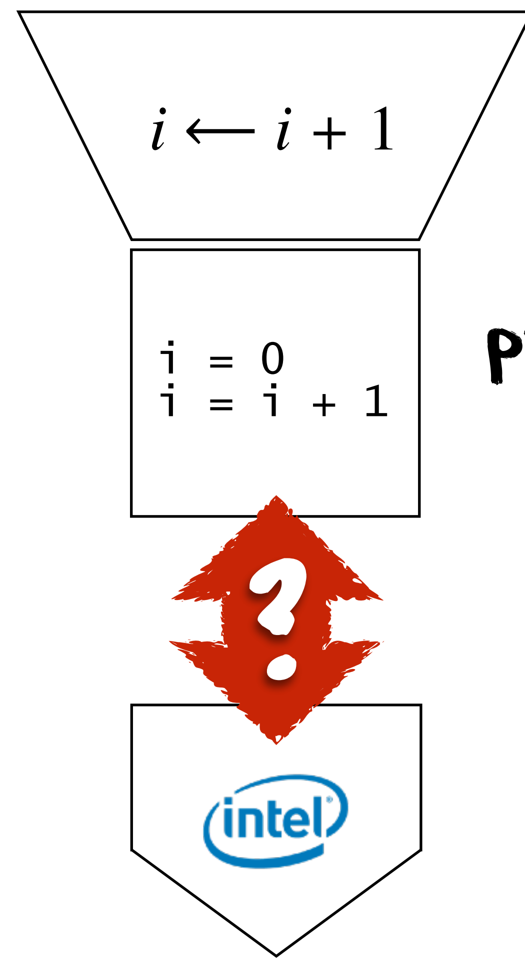
concept

program P
written in
language L
running on
interpreter L
running on
machine M

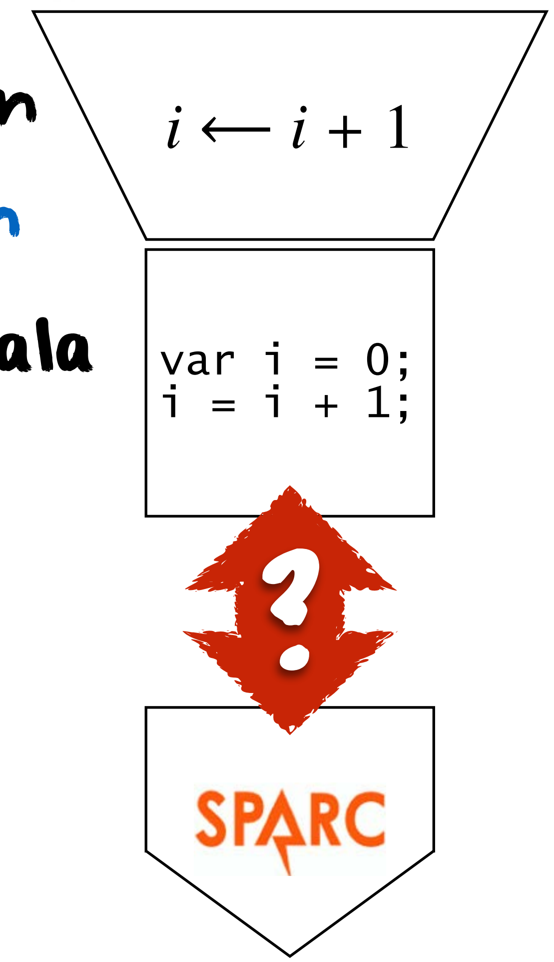


an interpreter dynamically translates language L into language M

examples

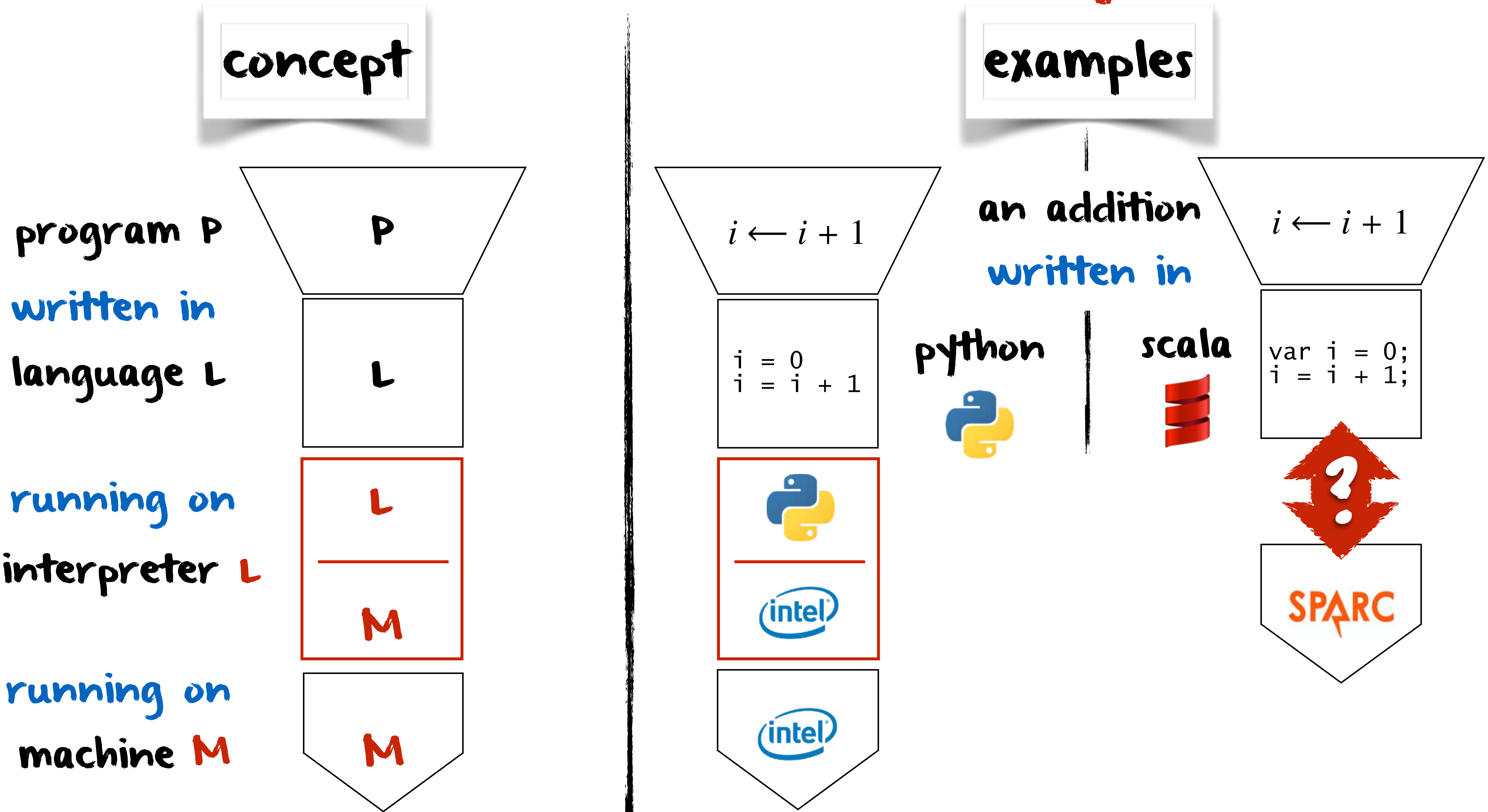


an addition
written in
python | scala



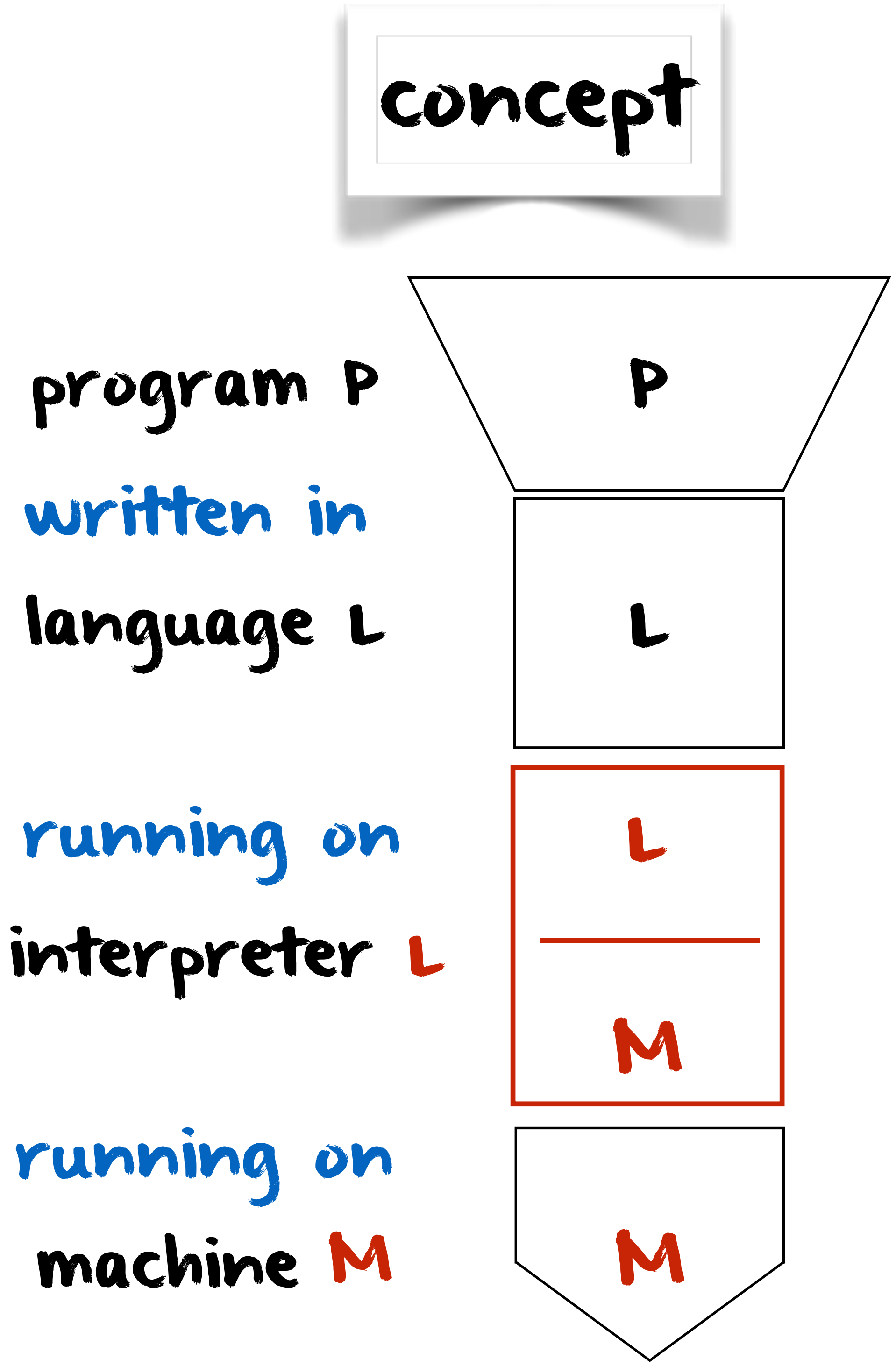
solution!

executions and interpreters

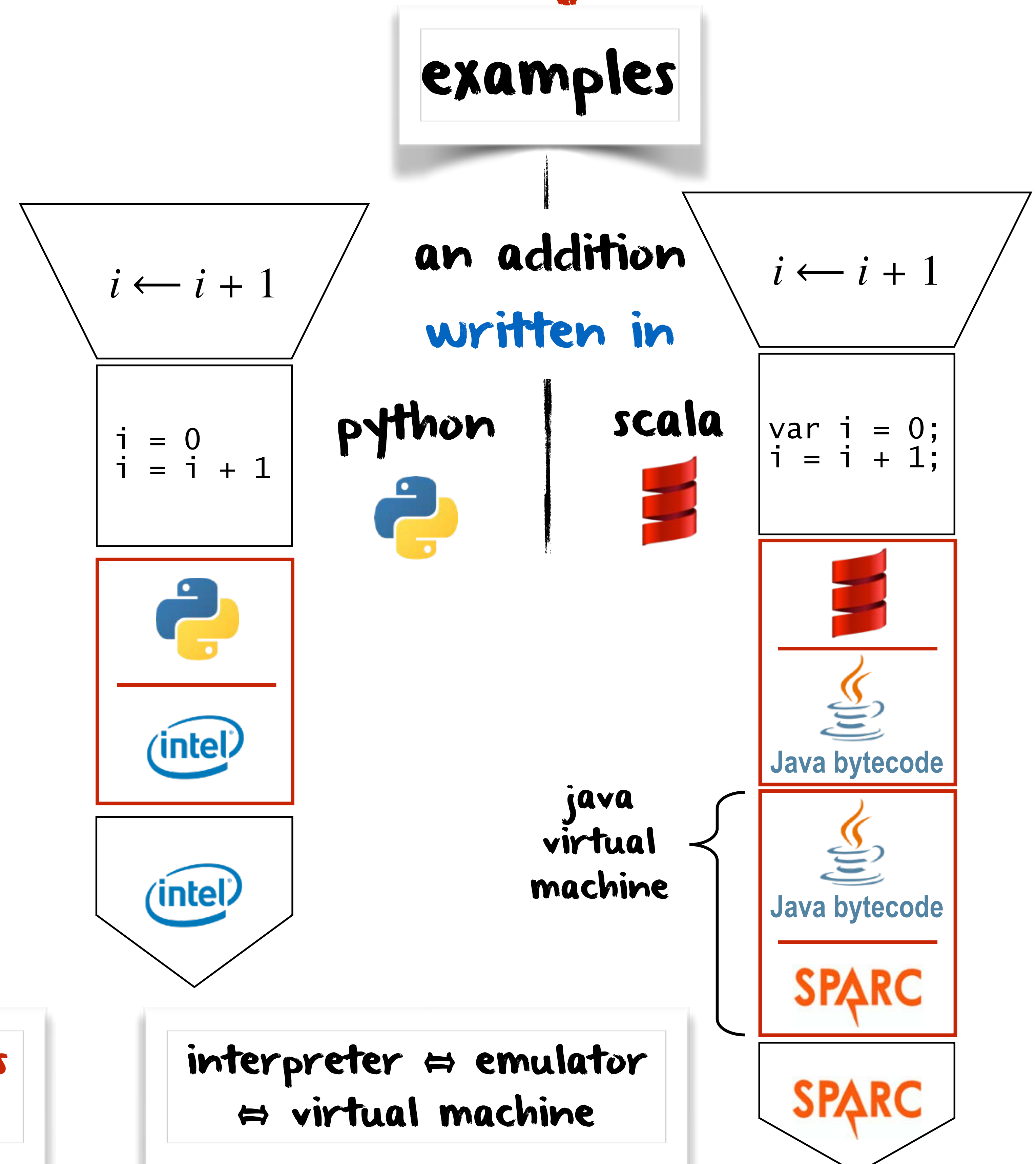


an interpreter dynamically translates language L into language M

executions and interpreters



an interpreter dynamically translates
language L into language M



interpreter ⇔ emulator
⇔ virtual machine

what's a compiler

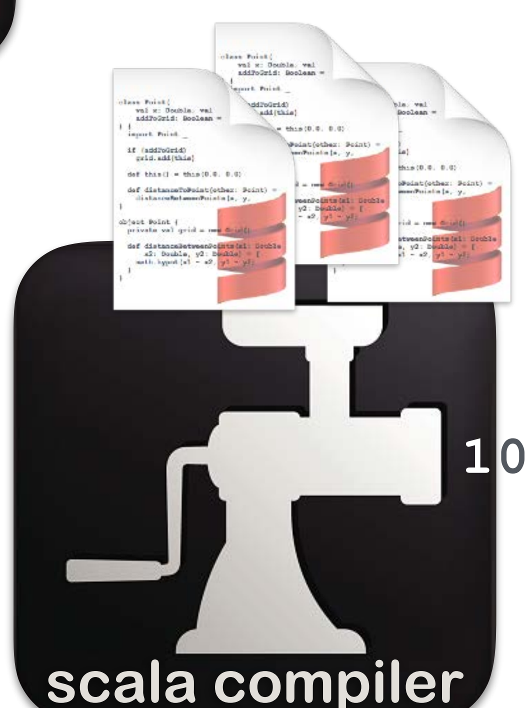
a program that translates human-understandable source code to machine-understandable byte code

0010010100101011000100101011001101001110011111001101010...

LDR R3 R1

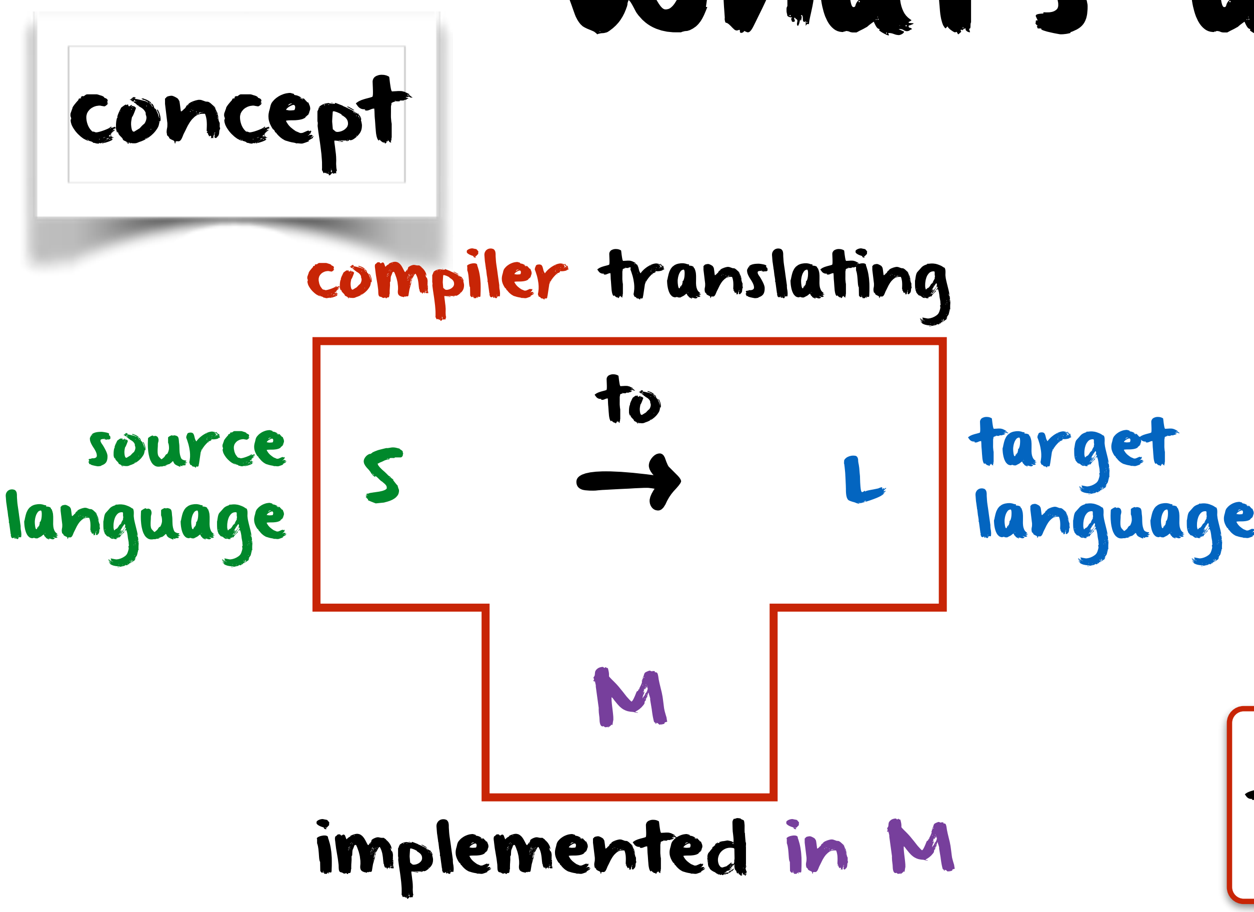
ADD R3 R1 R2

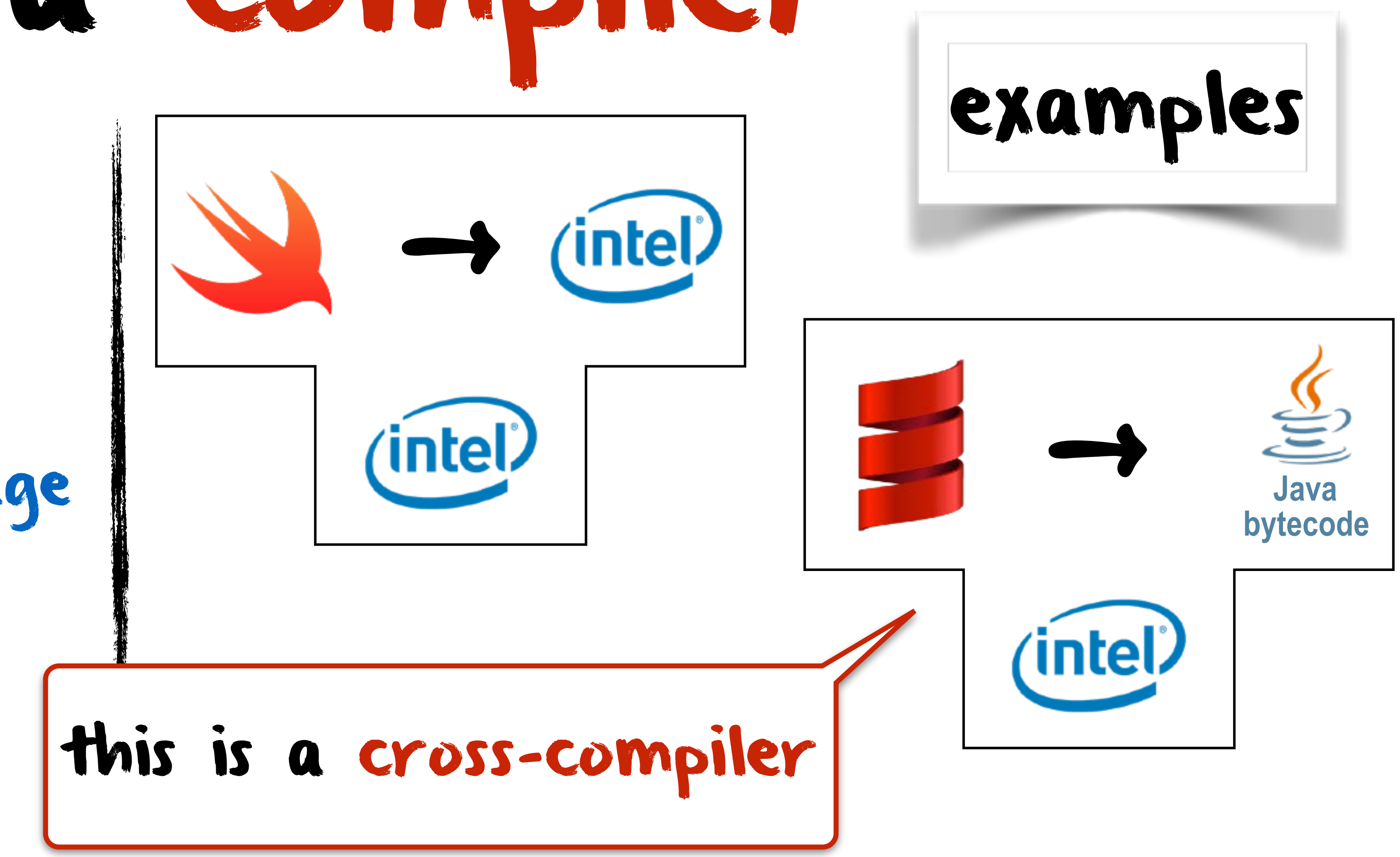
STR R3 R1

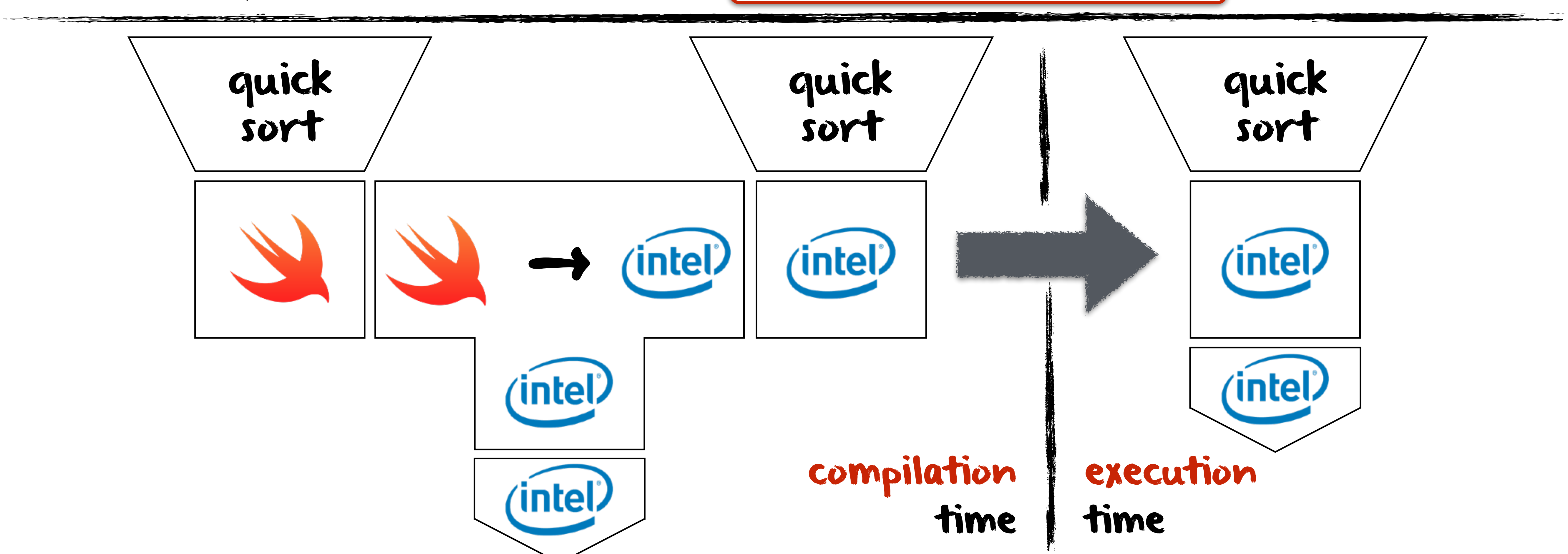


10011101100010010101100110100111001110011010...

what's a compiler

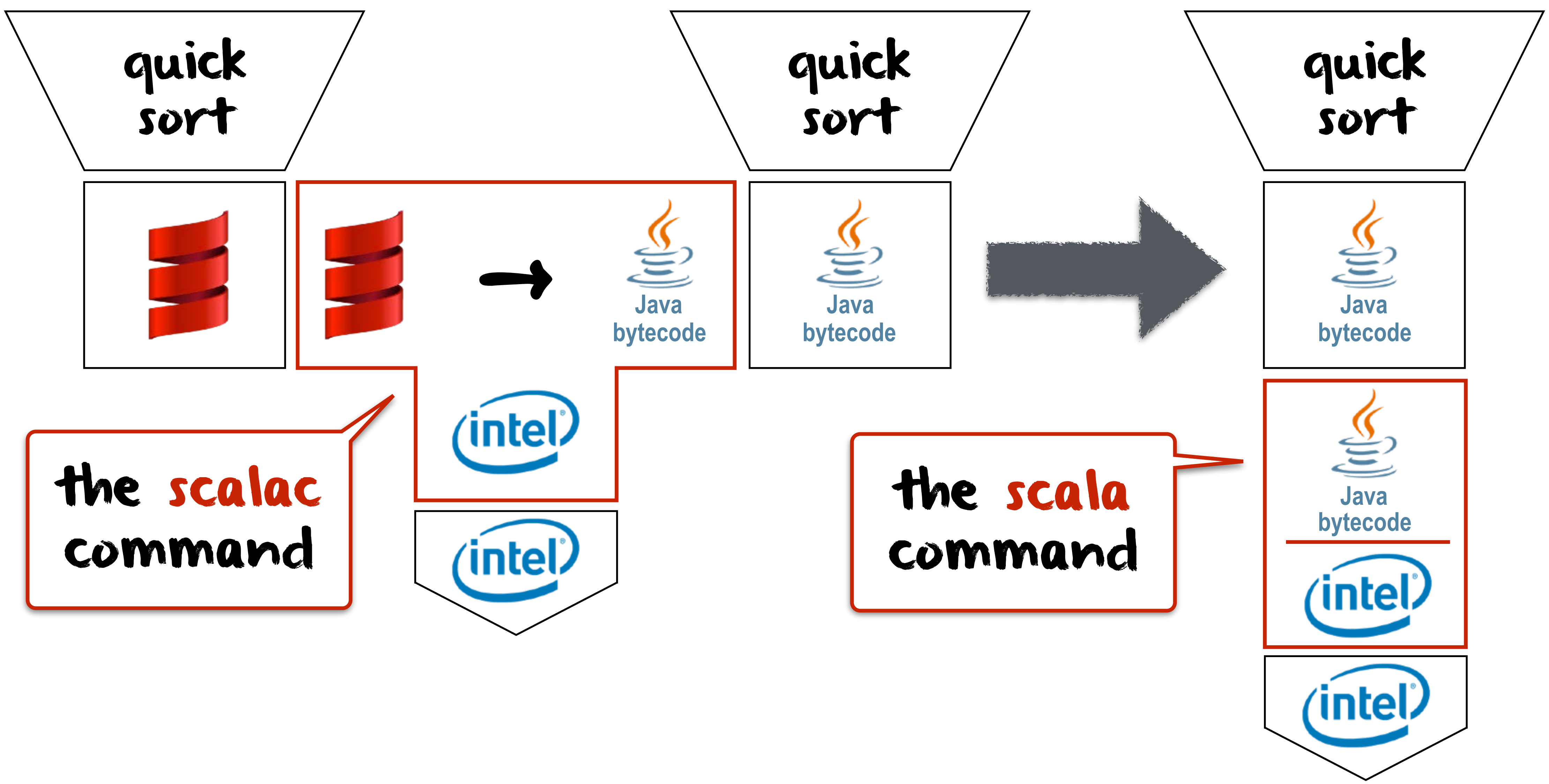






what's a compiler

the example of scala



static vs. dynamic

TRANSLATION

the translation occurs at compile time, before the execution, while the program is static

INTERPRETATION

the interpretation occurs at run time, during the execution, while the program is dynamic

what are runtime systems & libraries?

a library contains **predefined bricks** (functions, objects, etc.) that help create software, e.g., strings, dates, lists, input/output functions, etc.

a runtime system is the **mortar** that glues the various parts of software **during execution**

---

where is args stored?

where does println(...) come from?

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Hello, world!")
  }
}
```

where do Array & String come from ?

how is "Hello, world!" passed to println(...)?