INT101
Programming Fundamental

2021/1

Bachelor Science in Information Technology (B.Sc.IT)

School of Information Technology (SIT)

King Mongkut's University of Technology Thonburi (KMUTT)

Basic Abstractions of Software and Hardware Architectures for Programming

- **Abstraction**
  - a general idea or quality rather than an actual person, object, or event; an abstract quality or character (from Merriam-Webster)

- **Architecture**
  - the manner in which the components of a computer or computer system are organized and integrated (from Merriam-Webster)

Definition of *abstraction*

1 **a** : the act or process of abstracting : the state of being abstracted

b : an abstract idea or term

2 : absence of mind or preoccupation

3 : abstract quality or character

Definition of *architecture*

1 : the art or science of building

specifically : the art or practice of designing and building structures and especially habitable ones

2 **a** : formation or construction resulting from or as if from a conscious act

// the *architecture* of the garden

b : a unifying or coherent form or structure

// a novel that lacks *architecture*

5 : the manner in which the components of a computer or computer system are organized and integrated

// different program *architectures*

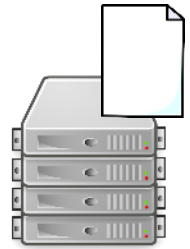
Understand How A Computer Program Works

input

CPU
(processing)

Memory
(storing)

output

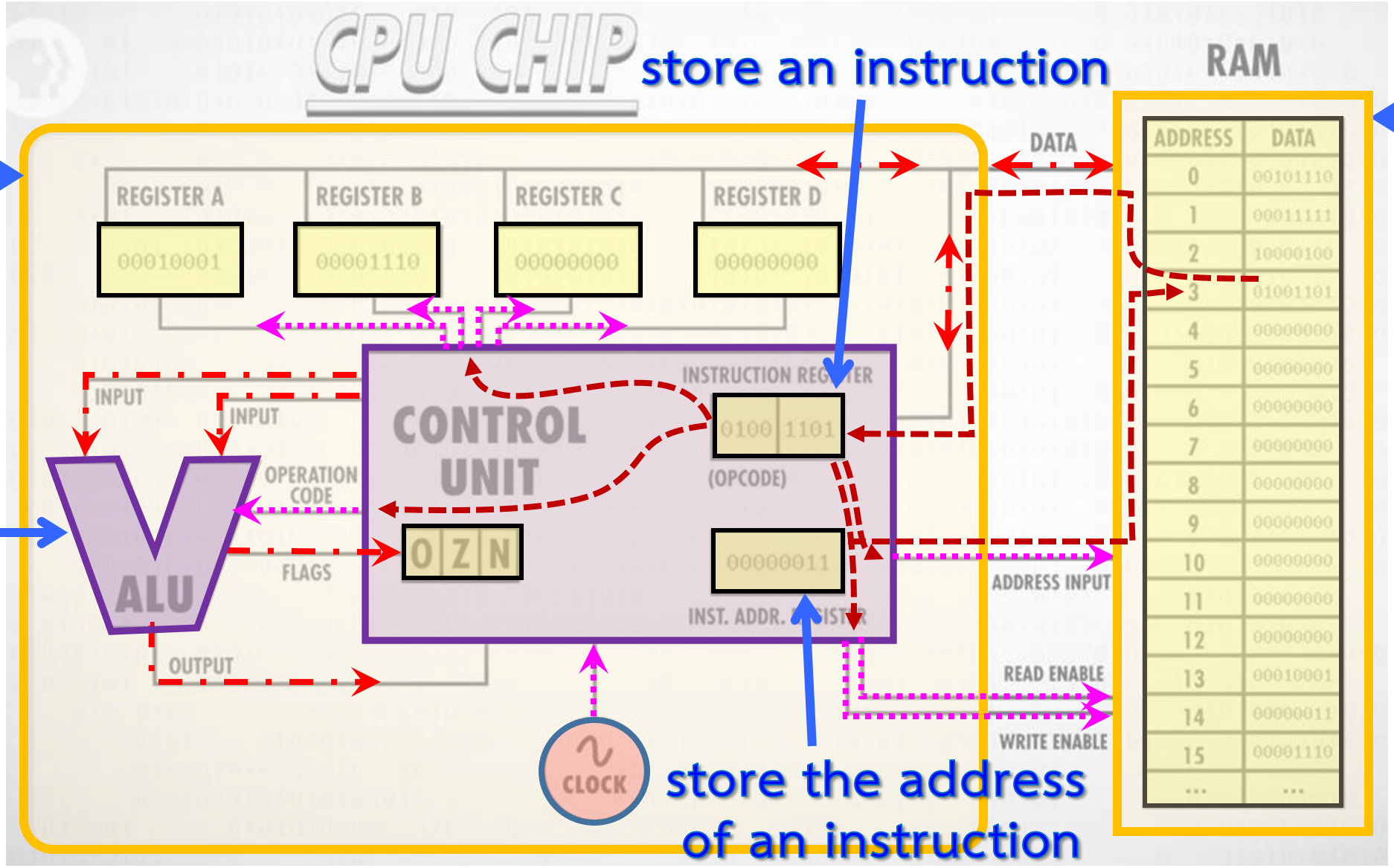


I/O = Input/Output

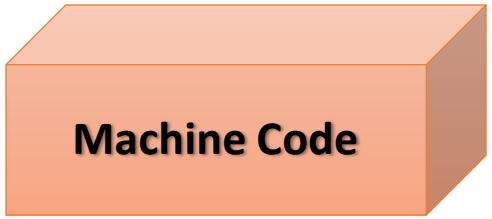
https://www.khanacademy.org/computing/code-org/computers-and-the-internet/how-computers-work/v/khan-academy-and-codeorg-introducing-how-computers-work

The Central Processing Unit (CPU): Crash Course Computer Science #7

https://www.youtube.com/watch?v=FZGugFqdr60



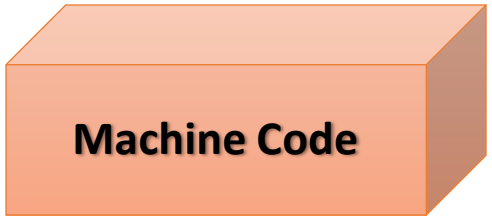
Programming Languages



First Generation
Programming Language (1GL)

```
1000101111001000
0010101111000011
```

Programming Languages

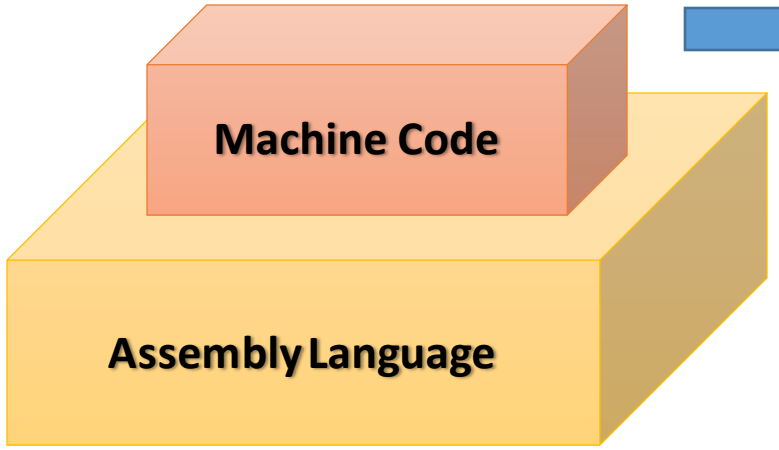


First Generation
Programming Language (1GL)

```
OPCODE D W MOD REG R/M
100010 1 1 11 001 000   MOV CX,AX ; CX ← AX
001010 1 1 11 000 011   SUB AX,BX ; AX ← AX - BX
```

CPU Intel 8086

Programming Languages



First Generation
Programming Language (1GL)

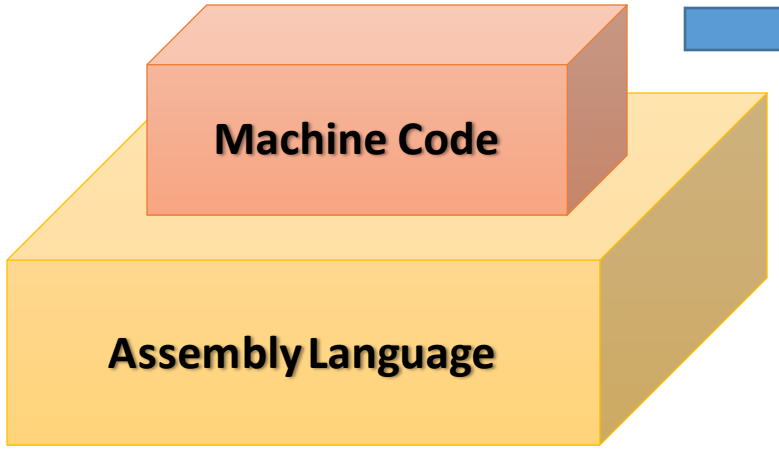
Second Generation
Programming Language (2GL)

```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC            ; procedure begins here
        CMP   AX,97     ; compare AX to 97
        JL    DONE      ; if less, jump to DONE
        CMP   AX,122    ; compare AX to 122
        JG    DONE      ; if greater, jump to DONE
        SUB   AX,32     ; subtract 32 from AX
DONE:   RET             ; return to main program
SUB32   ENDP            ; procedure ends here
```

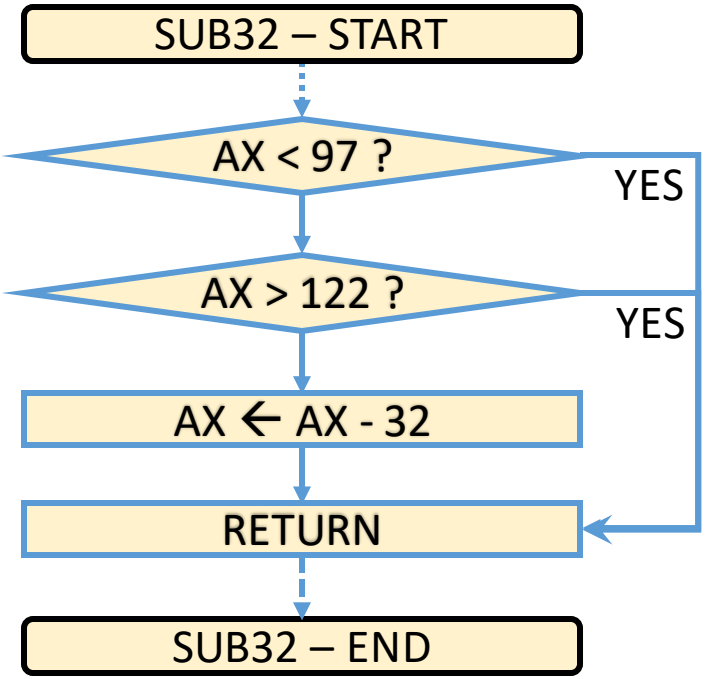
http://www.cybercomputing.co.uk/Languages/Languages/Low_level/assembly.html

Programming Languages



First Generation
Programming Language (1GL)

Second Generation
Programming Language (2GL)

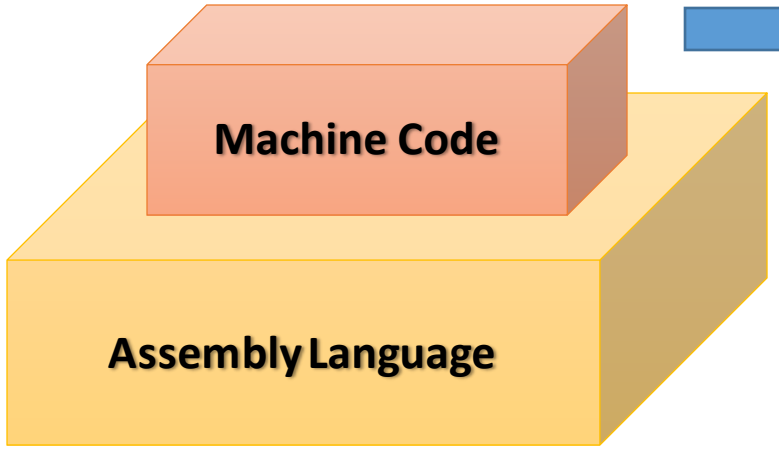



```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC            ; procedure begins here
        CMP   AX,97     ; compare AX to 97
        JL    DONE      ; if less, jump to DONE
        CMP   AX,122    ; compare AX to 122
        JG    DONE      ; if greater, jump to DONE
        SUB   AX,32     ; subtract 32 from AX
DONE:   RET             ; return to main program
SUB32   ENDP            ; procedure ends here
```

http://www.cybercomputing.co.uk/Languages/Languages/Low_level/assembly.html

Programming Languages

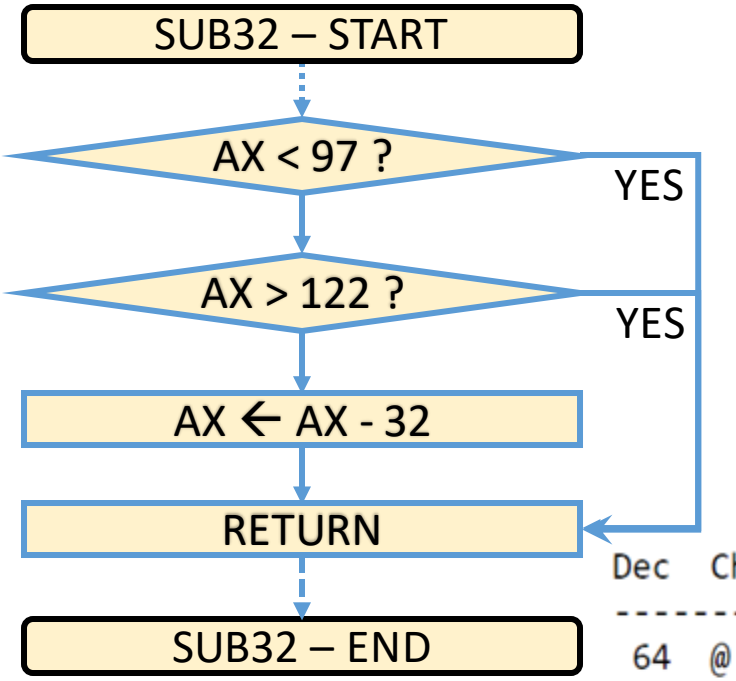


First Generation Programming Language (1GL)

Second Generation Programming Language (2GL)

```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC            ; procedure begins here
        CMP   AX,97     ; compare AX to 97
        JL    DONE      ; if less, jump to DONE
        CMP   AX,122    ; compare AX to 122
        JG    DONE      ; if greater, jump to DONE
        SUB   AX,32     ; subtract 32 from AX
DONE:   RET             ; return to main program
SUB32   ENDP            ; procedure ends here
```

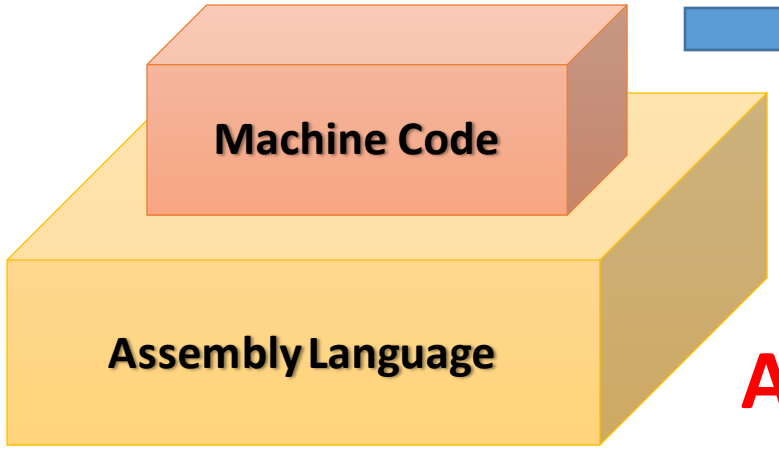


97 – 32 = 65

Dec	Char	Dec	Char
64	@	96	`
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f

http://www.cybercomputing.co.uk/Languages/Languages/Low_level/assembly.html

Programming Languages



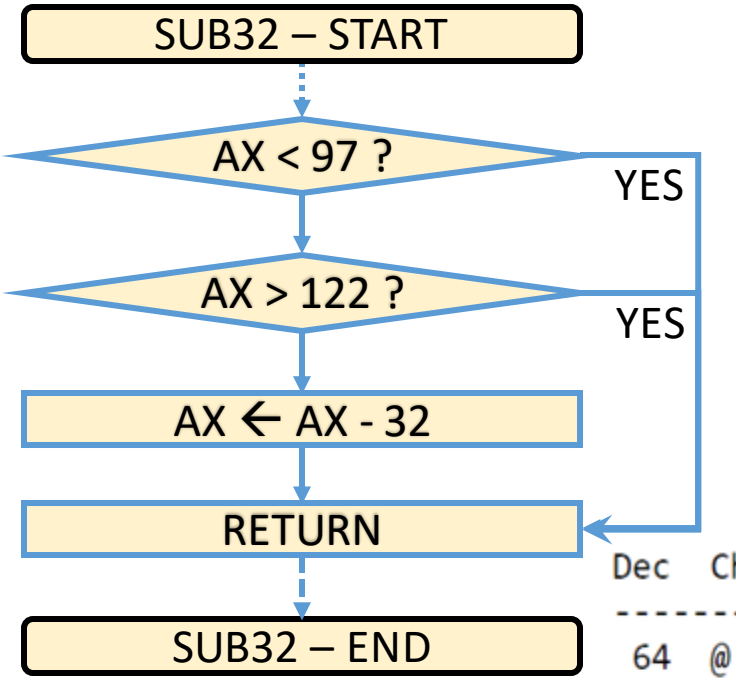
First Generation Programming Language (1GL)

Second Generation Programming Language (2GL)

Abstraction

```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC            ; procedure begins here
        CMP   AX,97     ; compare AX to 97
        JL    DONE      ; if less, jump to DONE
        CMP   AX,122    ; compare AX to 122
        JG    DONE      ; if greater, jump to DONE
        SUB   AX,32     ; subtract 32 from AX
DONE:   RET             ; return to main program
SUB32   ENDP            ; procedure ends here
```



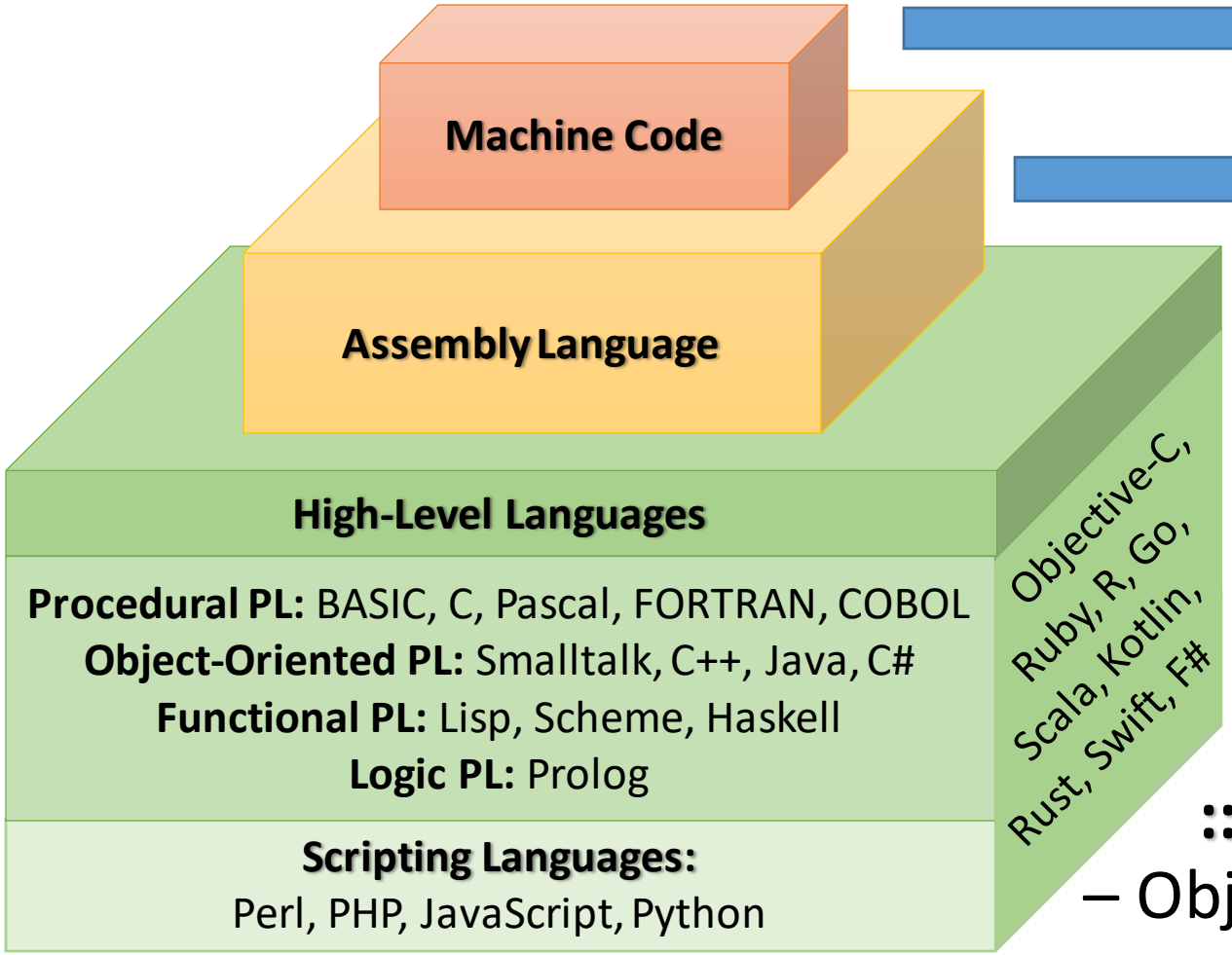
UPPER(AX)

97 – 32 = 65

Dec	Char	Dec	Char
64	@	96	`
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f

http://www.cybercomputing.co.uk/Languages/Languages/Low_level/assembly.html

Programming Languages



First Generation Programming Language (1GL)

Second Generation Programming Language (2GL)

Third Generation Programming Language (3GL)

Fourth Generation Programming Language (4GL) : SQL

:: Programming Language Paradigms ::

– Object-Oriented Programming Languages

– Functional Programming Languages

– Logic Programming Languages

– Reactive Programming Languages

https://en.wikipedia.org/wiki/Programming_paradigm

Imperative -> Procedural -> Structured -> Object-Oriented

Imperative Programming

- Statements
  - read, compute, write
- Branching: IF THEN
- Jumping: GOTO

Procedural Programming

- Subprogram
- Subroutine
- Procedure
- Function

Structured Programming

- Block of statements
  - Local variables in block-scope

```
   I = 1
   N = 10
   FAC = 1
START:
   IF I <= N THEN
      FAC = FAC * I
      I = I + 1
      GOTO START
   END
   PRINT FAC
```

if, goto/label

```
FUNCTION FACTORIAL(N)
   I = 1
   FAC = 1
   WHILE I <= N DO
      FAC = FAC * I
      I = I + 1
   RETURN FAC

RESULT = FACTORIAL(10)
PRINT RESULT
```

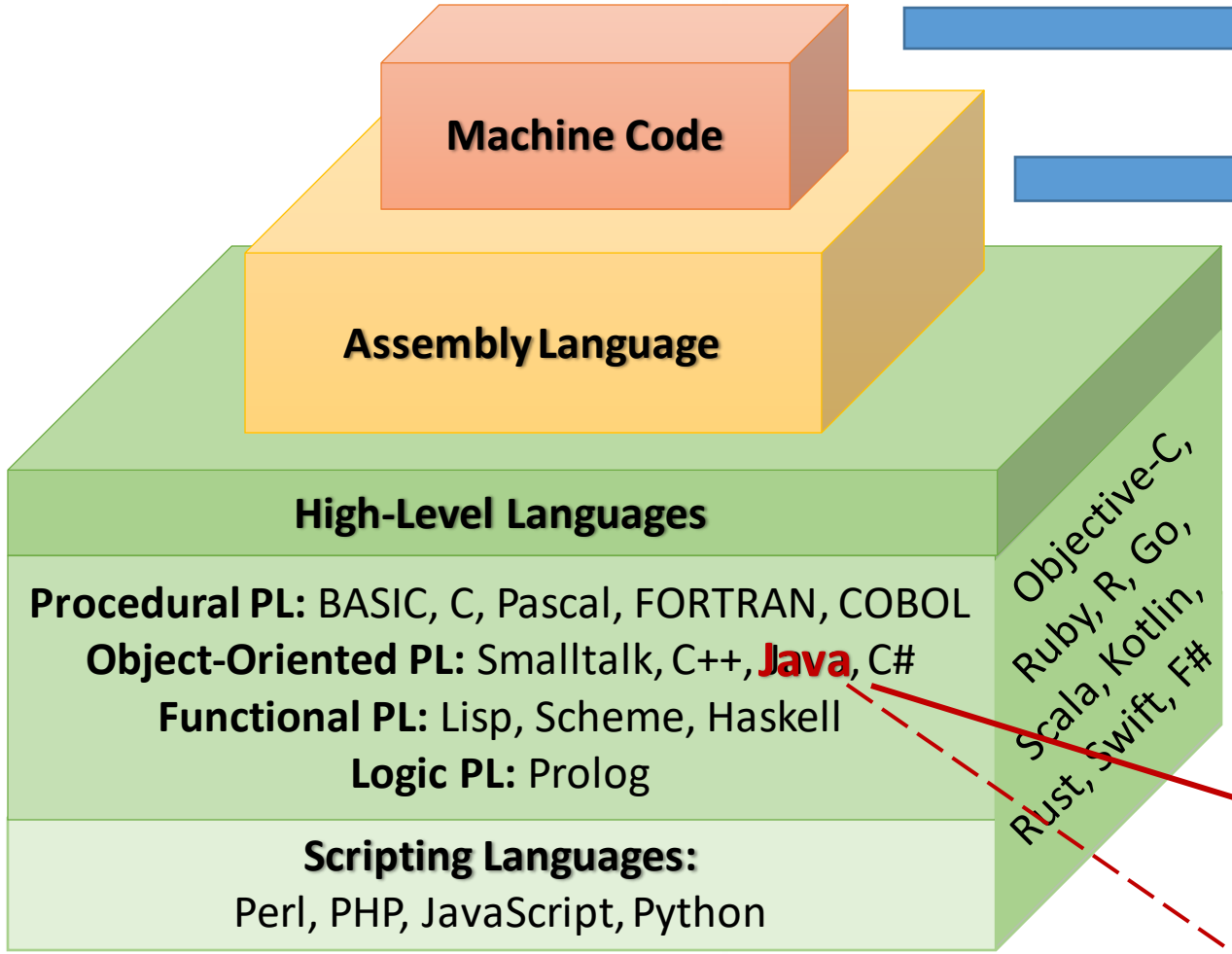
procedure / reusable

```
FUNCTION FACTORIAL(N)
   FAC = 1
   FOR I = 1 TO N DO
      FAC = FAC * I
   RETURN FAC

RESULT = FACTORIAL(10)
PRINT RESULT
```

block scope

Programming Languages



First Generation Programming Language (1GL)

Second Generation Programming Language (2GL)

Third Generation Programming Language (3GL)

Fourth Generation Programming Language (4GL) : SQL

Programming Language Paradigms:

- Imperative Programming Languages
  - Procedural Programming Languages
  - Structured Programming Languages
  - **Object-Oriented Programming Languages**
  - Aspect-Oriented Program. Languages
- Declarative Programming Languages
  - **Functional Programming Languages**
  - Logic Programming Languages
  - Reactive Programming Languages

Multi-Paradigm Programming Languages

https://en.wikipedia.org/wiki/Programming_paradigm

Programming Language Paradigms

Imperative Prog. Lang.

specify **what** should be done and **how**

Procedural Prog. Lang.

FORTRAN, COBOL

subroutine, procedure, function

Structured Prog. Lang.

C, Pascal

block structure, subroutine, …

Object-Oriented Prog. Lang. (OOP)

Smalltalk, C++, Java, C#, …

object = data + function

Aspect-Oriented Prog. Lang. (AOP)

cross-cutting concerns

AspectJ, …

Declarative Prog. Lang.

specify **what** should be done

Functional Prog. Lang.

Lisp, Scheme, Haskell

Functions are first-class citizen.

Logic Prog. Lang.

Prolog

A program is a set of rules and facts.

Reactive Prog. Lang. (Rx)

ReactiveX (RxJava, RxJS, …)

response to situations

Other Programming Language Issues

Sequential Programming vs.
Concurrent Programming

Compilation (Compiler) vs.
Interpretation (Interpreter)

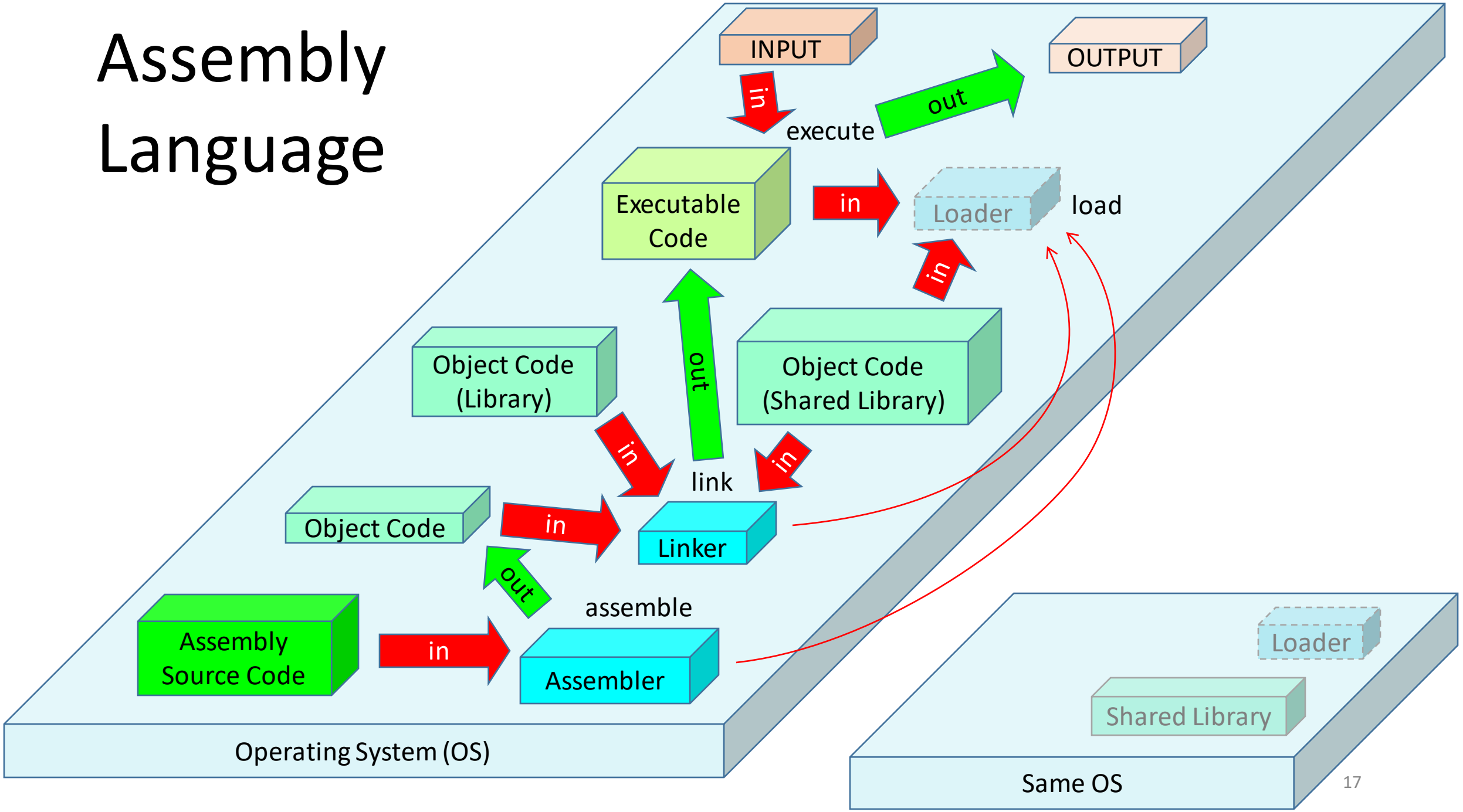
Strongly-typed
vs. **Weakly-typed**

Manual vs. **Automatic**
Memory Management

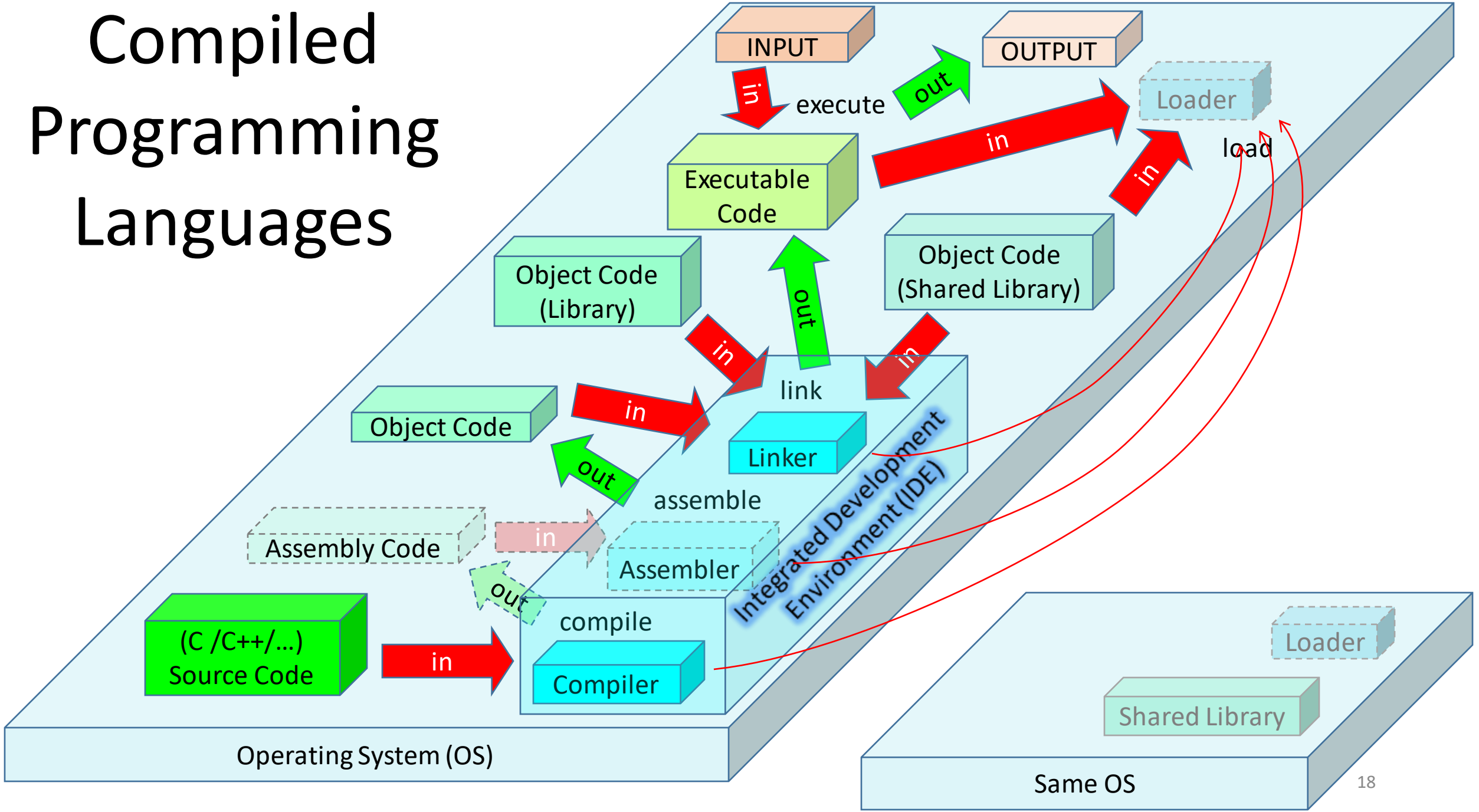
Building and Running a Computer Program

Assembling Compiling Interpreting

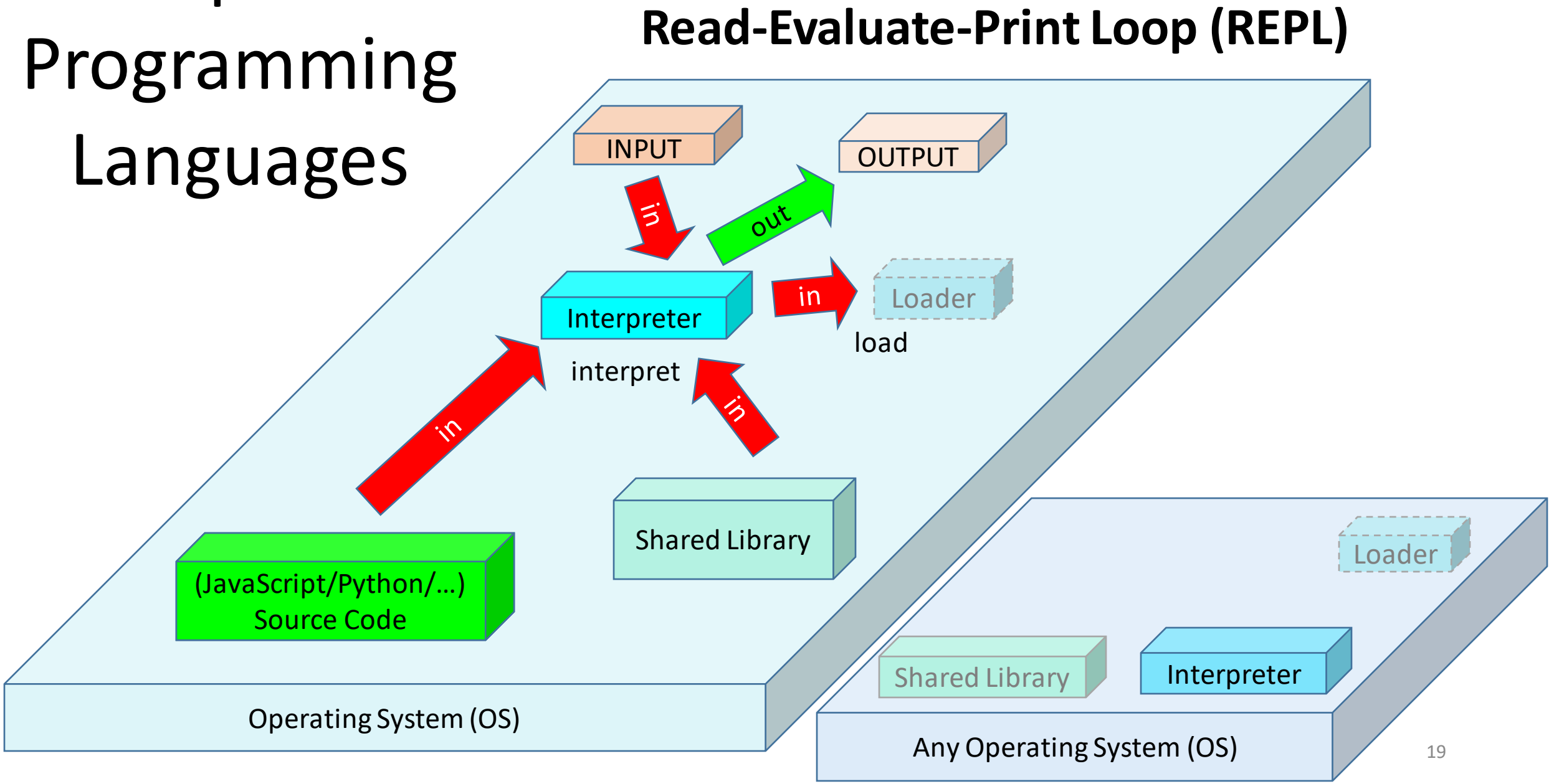
Assembly Language



Compiled Programming Languages



Interpreted Programming Languages



Compiled Languages

– A platform-dependent compiler is required for each platform to generate an executable program from a source code.

– The executable program can run only on one specific platform.

+ The executable program is a machine code, so it runs fast, and without a compiler.

+ The compilation process helps screening out some types of errors before they show up at runtime.

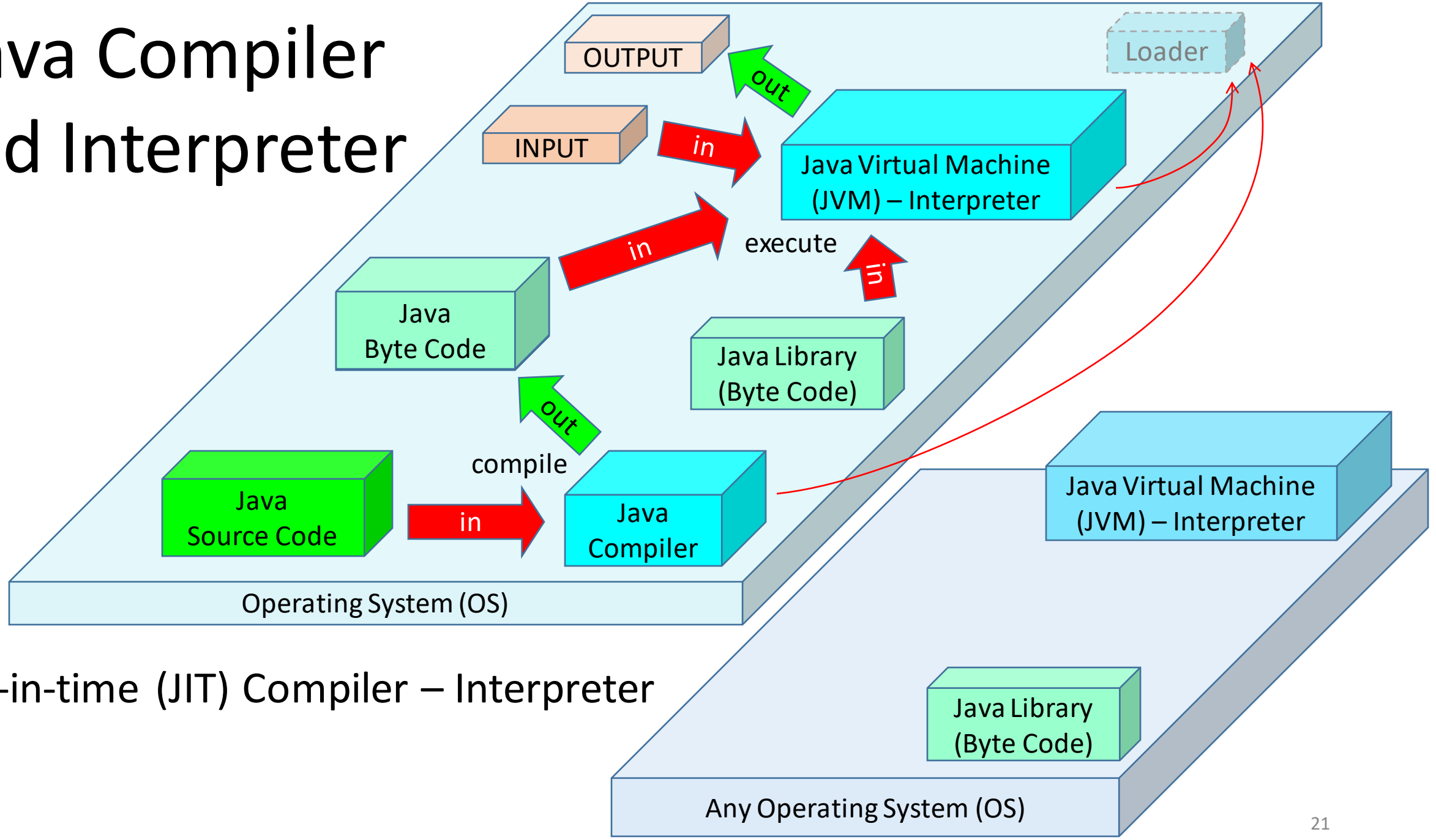
Interpreted Languages

– A platform-dependent interpreter is always required for each platform in order to run the program.

+ The program can run on any platform that has an interpreter.

– The interpreter interprets the source code and execute them line-by-line (one statement at a time), so it is slow.

Java Compiler and Interpreter



Just-in-time (JIT) Compiler – Interpreter

An **integrated development environment** (**IDE**) is an application software that provides comprehensive facilities to programmers for software development.
An IDE normally consists of at least

- a source code editor,
- build automation tools and
- a debugger.

Some IDEs, such as NetBeans and Eclipse, contain the necessary compiler, interpreter.

Common Features in many IDEs:

- Syntax Highlighting
- Code Completion
- Code Search
- Debugging
- Build Automation Integration
- Refactoring
- Version Control
- Visual Programming

https://en.wikipedia.org/wiki/Integrated_development_environment

Object-Oriented Programming Concept

An Object-Oriented Program

- A program is
  - **a collection of objects**
    - **sending messages** to one another
      - to perform some tasks

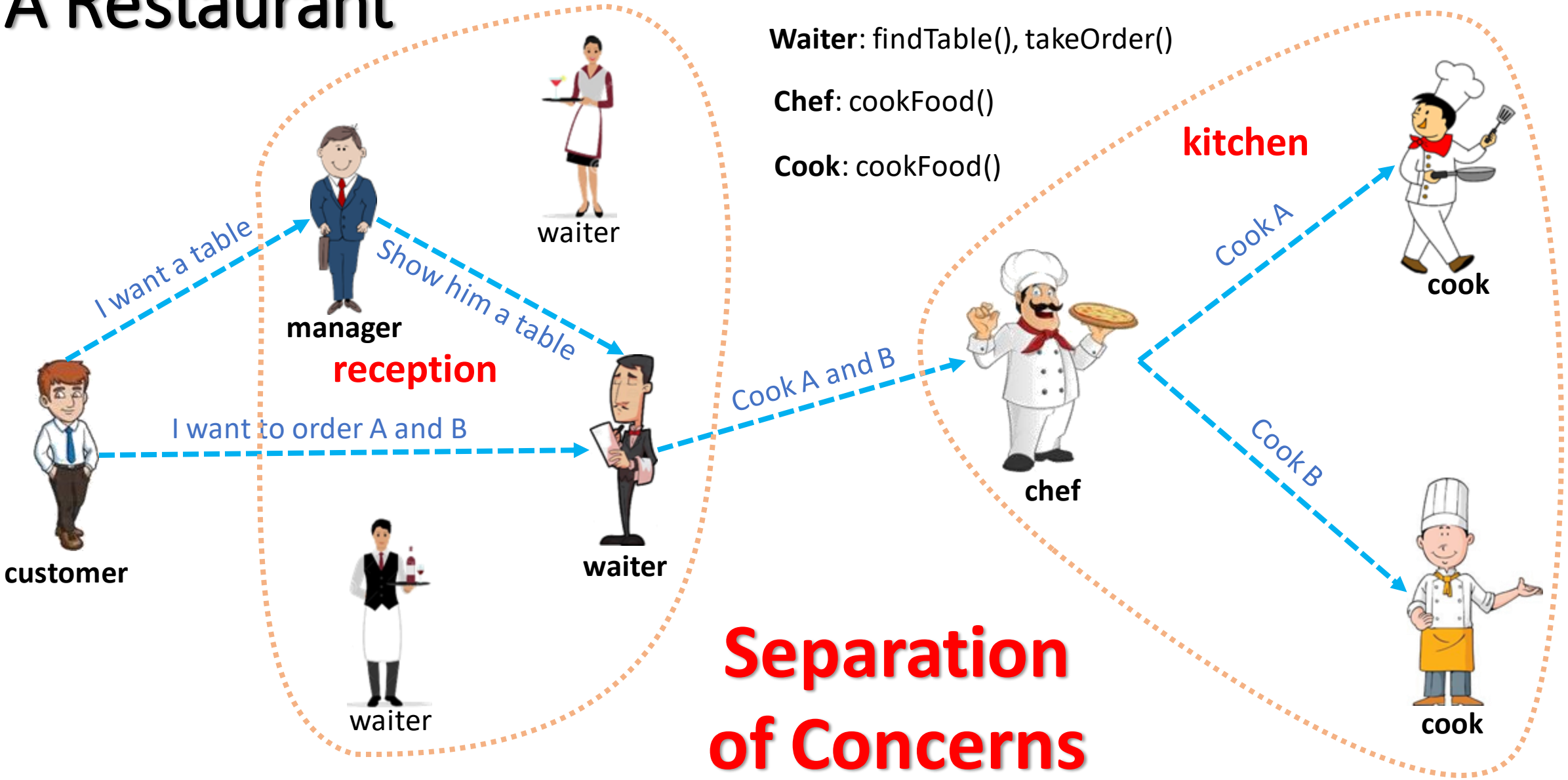
A Restaurant

Manager: findTable()

Waiter: findTable(), takeOrder()

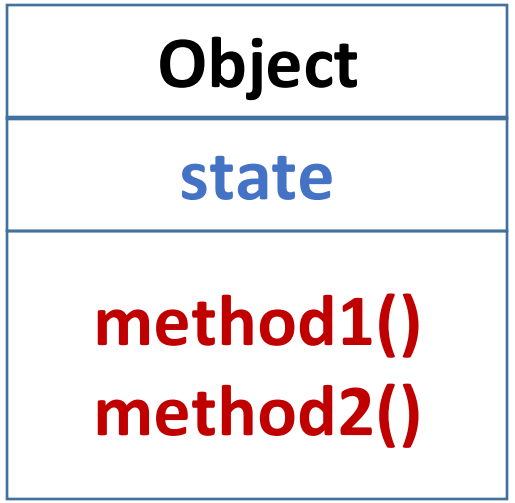
Chef: cookFood()

Cook: cookFood()



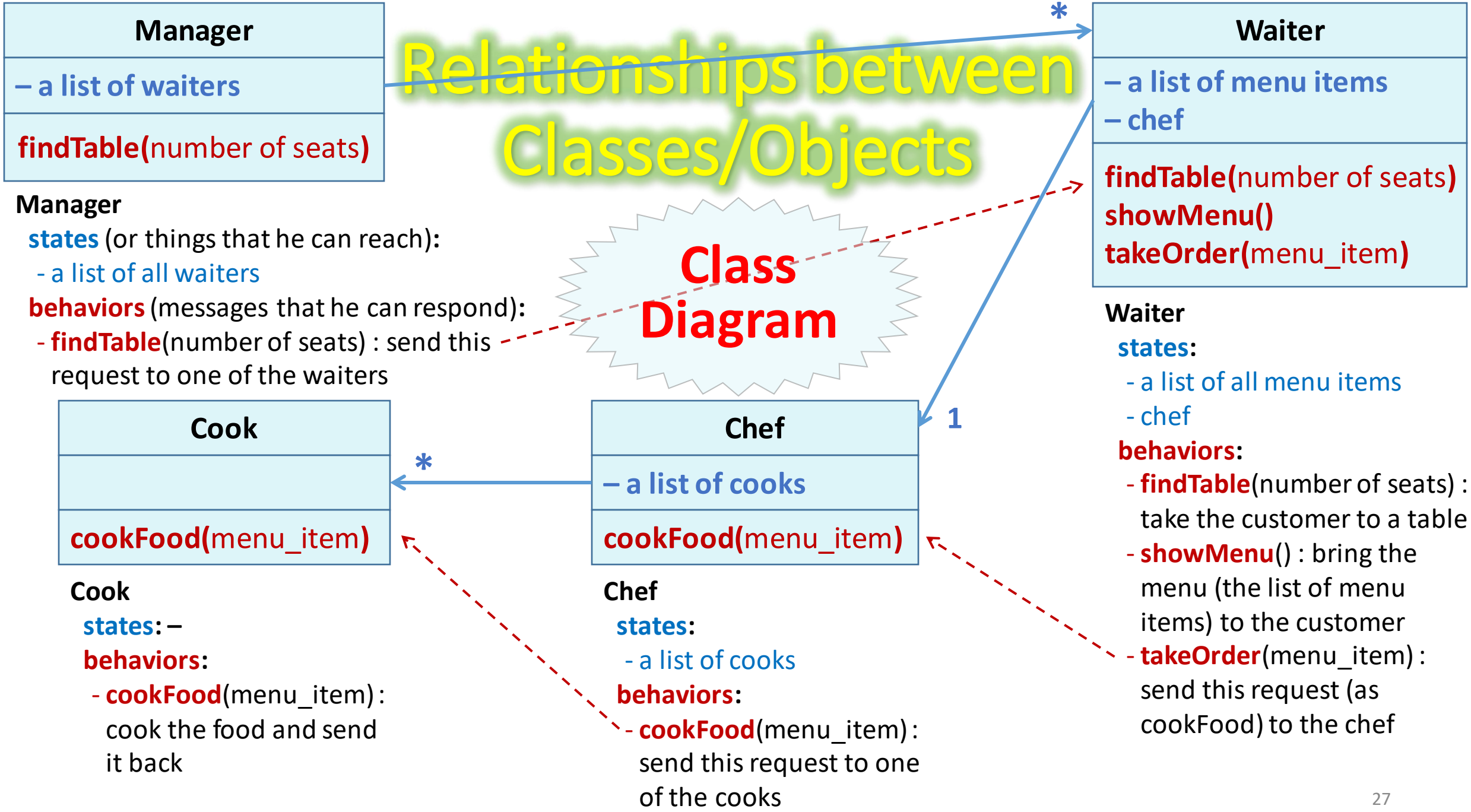
Separation of Concerns

An Object



- Object is an entity that
  - Have a **state** (data/information),
  - Can behave according to the **messages** received.
    - All possible messages that an object can receive are pre-defined: **methods.**
    - Upon receiving a message, object may change its state.

Relationships between Classes/Objects



Manager

states (or things that he can reach):
- a list of all waiters

behaviors (messages that he can respond):
- **findTable**(number of seats) : send this request to one of the waiters

Waiter

states:
- a list of all menu items
- chef

behaviors:
- **findTable**(number of seats) : take the customer to a table
- **showMenu**() : bring the menu (the list of menu items) to the customer
- **takeOrder**(menu_item) : send this request (as cookFood) to the chef

Cook

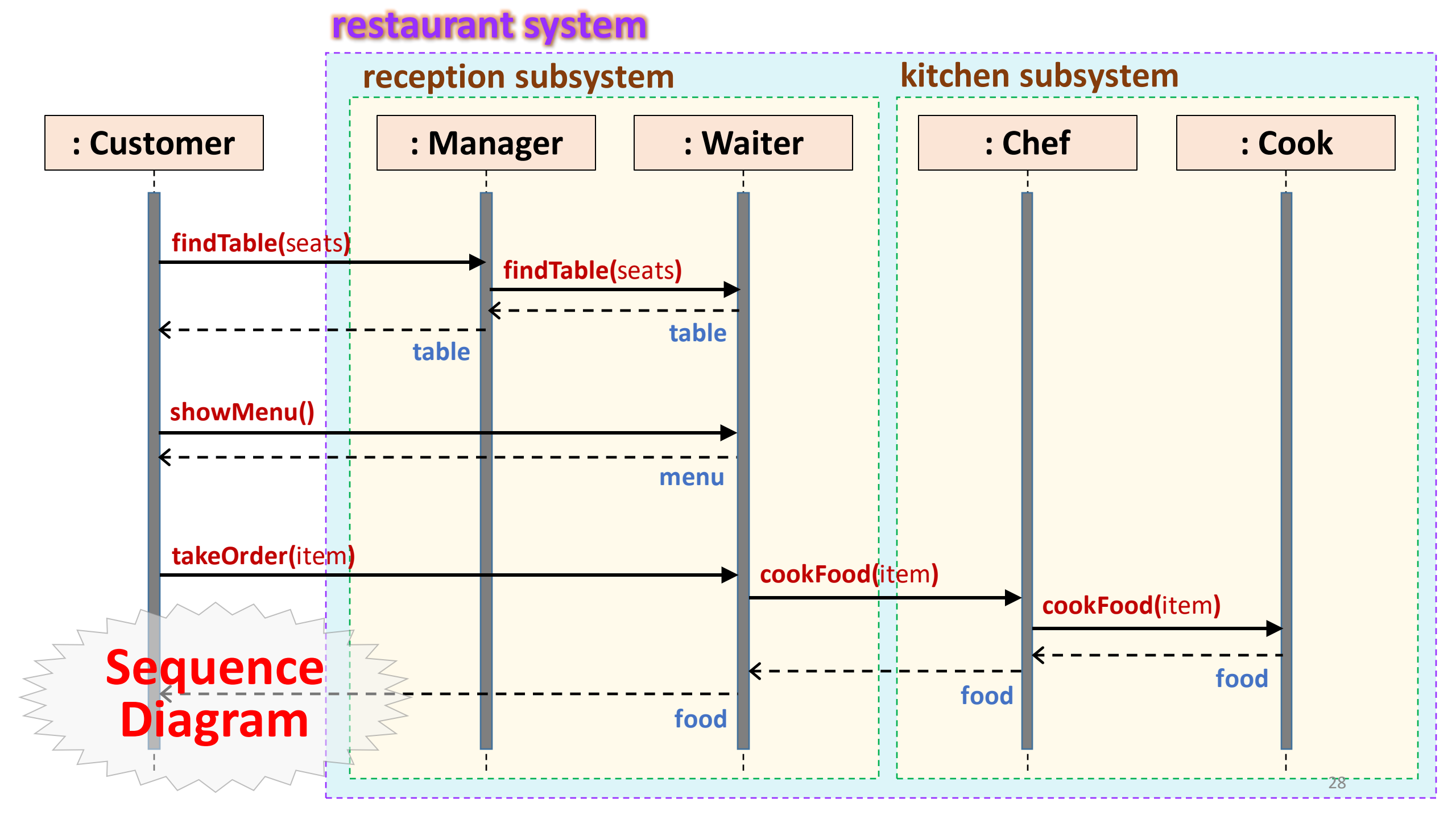
states: –

behaviors:
- **cookFood**(menu_item) : cook the food and send it back

Chef

states:
- a list of cooks

behaviors:
- **cookFood**(menu_item) : send this request to one of the cooks


restaurant system
reception subsystem
kitchen subsystem
: Customer
: Manager
: Waiter
: Chef
: Cook
findTable(seats)
findTable(seats)
table
table
showMenu()
menu
takeOrder(item)
cookFood(item)
cookFood(item)
food
food
food
Sequence Diagram

Car (Object – Instance)

odometer

speed

- unlockDoor()
- lockDoor()
- openDoor()
- closeDoor()

doorLockStatus

doorOpenStatus

- turnWheel()
- break()

wheelPosition

- turnOnAirConditioner()
- turnOffAirConditioner()
- setAirConditionerTemperature()
- setAirConditionerFanSpeed()

airConOnOffStatus

airConTemperature

airConFanSpeed

- startEngine()
- stopEngine()
- changeGear()
- accelerate()

engineStatus

gearStatus

- turnOnRadio()
- turnOffRadio()
- setRadioVolume()
- setRadioChannel()

radioOnOffStatus

radioVolumeLevel

radioChannel

Car (Object –> Collection of Objects)

- **Door System**
  - unlock()
  - lock()
  - open()
  - close()

  lockStatus
  openStatus

- **Engine**
  - start()
  - stop ()
  - changeGear()
  - accelerate()

  engineStatus
  gearStatus

- **Wheel System**
  - turnWheel()
  - break()

  wheelPosition

  speed
  odometer

- **Air Conditioner**
  - turnOn()
  - turnOff()
  - setTemperature()
  - setFanSpeed()

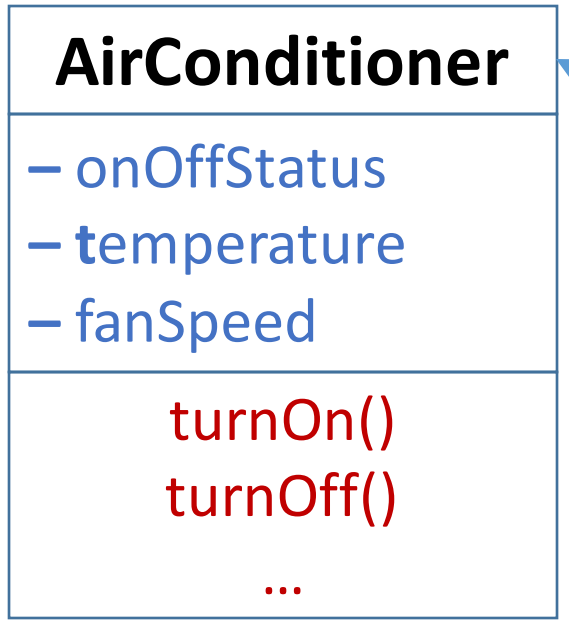
  onOffStatus
  temperature
  fanSpeed

- **Radio**
  - turnOn()
  - turnOff()
  - setVolume()
  - setChannel()

  onOffStatus
  volumeLevel
  channel

Car (Object – Instance)

- Door System
- Engine
- Wheel System
- Air Conditioner
- Radio



- turnOnAirConditioner()
- turnOffAirConditioner()
- setAirConditionerTemperature()
- setAirConditionerFanSpeed()

Air Conditioner

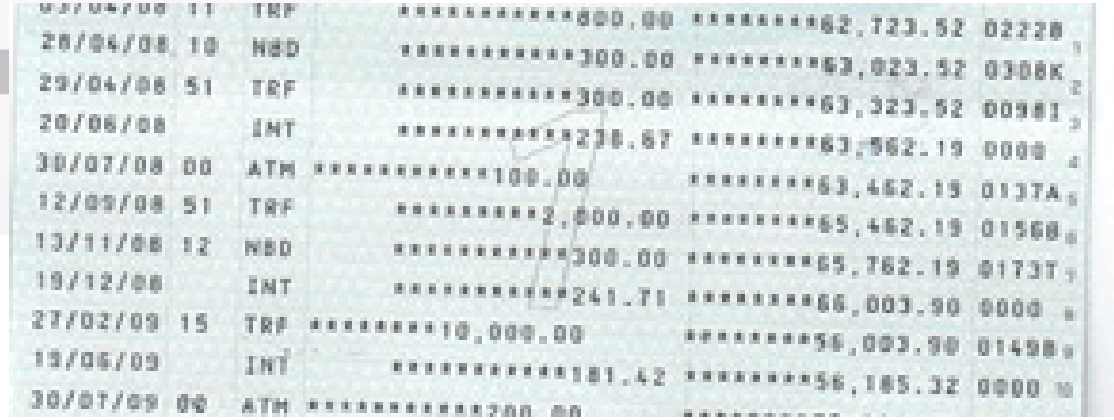
turnOn()
turnOff()
setTemperature()
setFanSpeed()

onOffstatus
temperature
fanSpeed

Vending Machine

- Methods
- State

Bank Account



- an account number
- an account owner
- a balance
- a transaction history

- deposit – an amount to deposit, the date of deposit
- withdraw – an amount to withdraw, the date of withdraw
- transfer – an amount to transfer, an account to transfer to, the date of transfer
- inquiry – (the date of inquiry)

- adding an interest – the date of adding the interest, (interest rate)

- open a new account – an account owner, the date of account opening
- close the account – the date of account closing

Elevator (Lift)

- Methods
- State

Air Conditioner

- turn on
- turn off
- increase/decrease temperature
- increase/decrease fan speed
- set air direction – the air direction
- set on/off timer – the time interval to turn on/off

Television

- turn on
- turn off
- set on/off timer – the time interval to change to
- change the channel – the channel to change to
- go back to the previous channel
- increase/decrease volume
- change the TV mode (TV, Cable TV, HDMI, VGA, …)
- change contrast/brightness
- …

Dice (rolling dice)

- Methods
- State

Vending Machine

for customer

receiveMoney()
returnMoney()
giveProduct()

moneyInserted
totalMoney
totalProducts

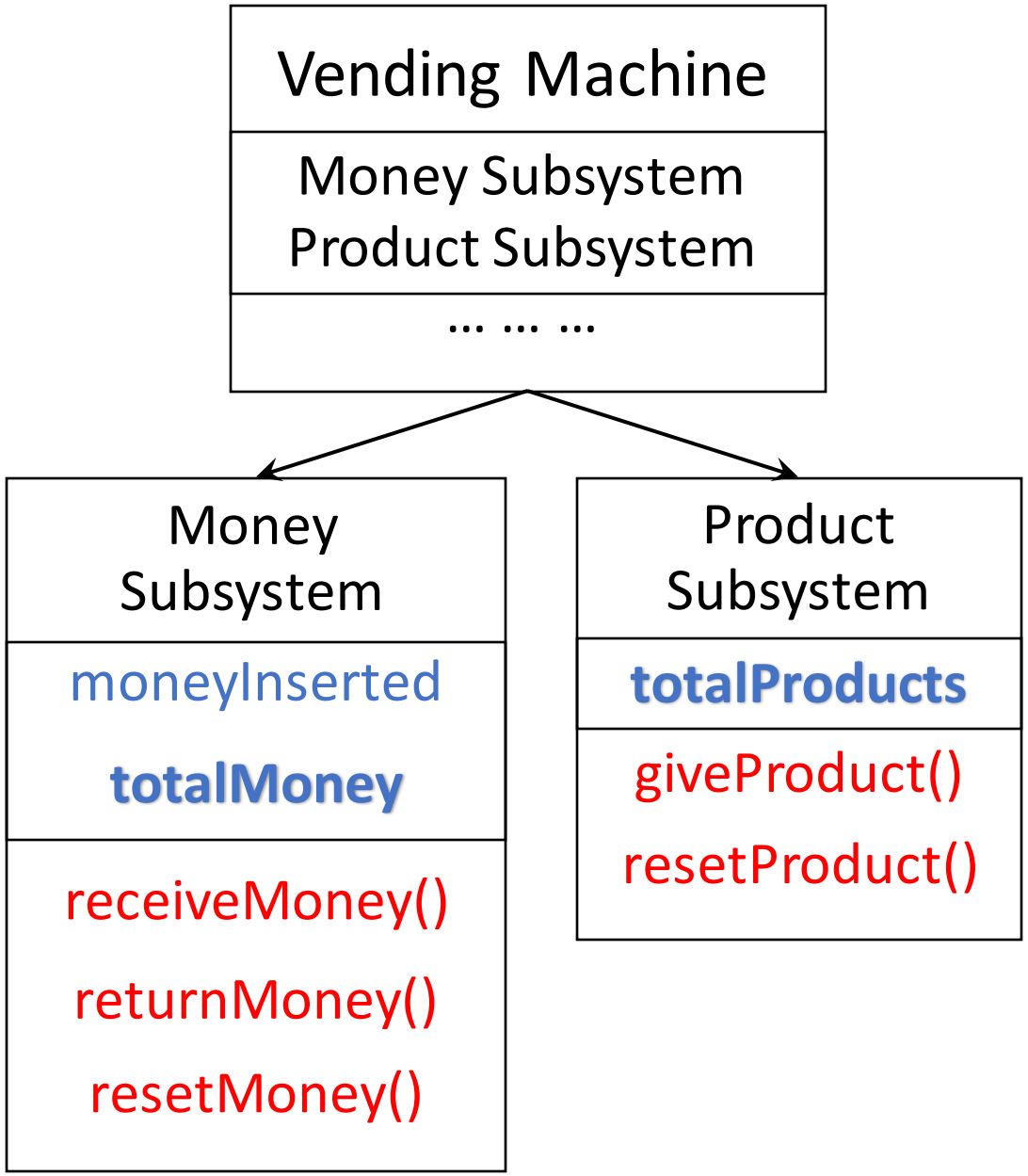
---

resetMoney()
resetProduct()

for service maintenance

Other Issues:

- On/Off Switch
  - OnOffStatus
- Temperature Control
  - currentTemperature
  - targetTemperature



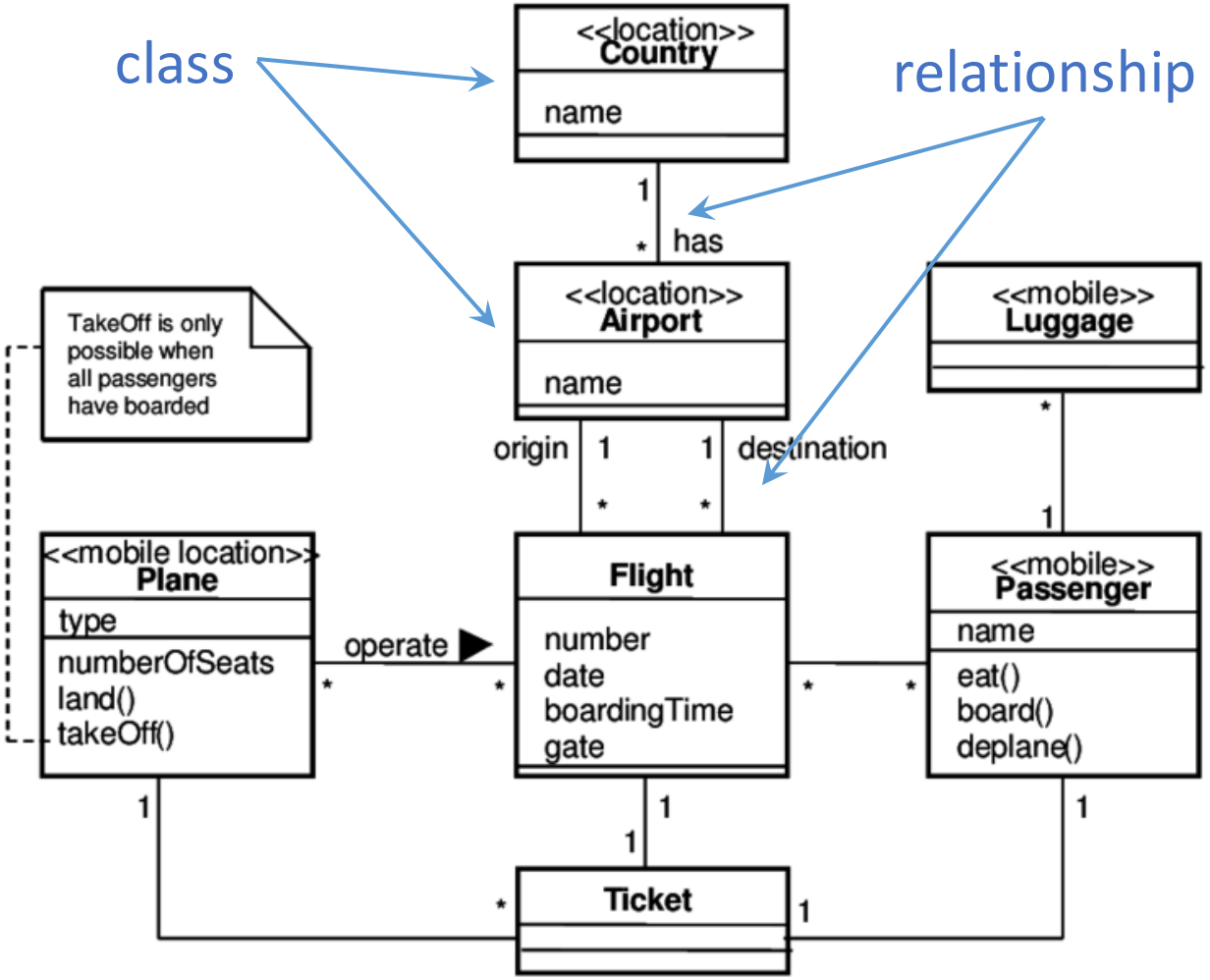
Elevator (Lift)

outside	up() down()	**requestedDirectionsFromFloors**
inside	openDoor() closeDoor()	doorStatus
	gotoFloor()	**floorsToGo**
	alarmOn() alarmOff()	alarmStatus
administration	turnOn() turnOff()	onOffStatus
internal status	currentFloor movingDirection	currentWeight maxWeight

Elevator
Engine Controller Alarm Subsystem Door Subsystem Weight Subsystem
… …

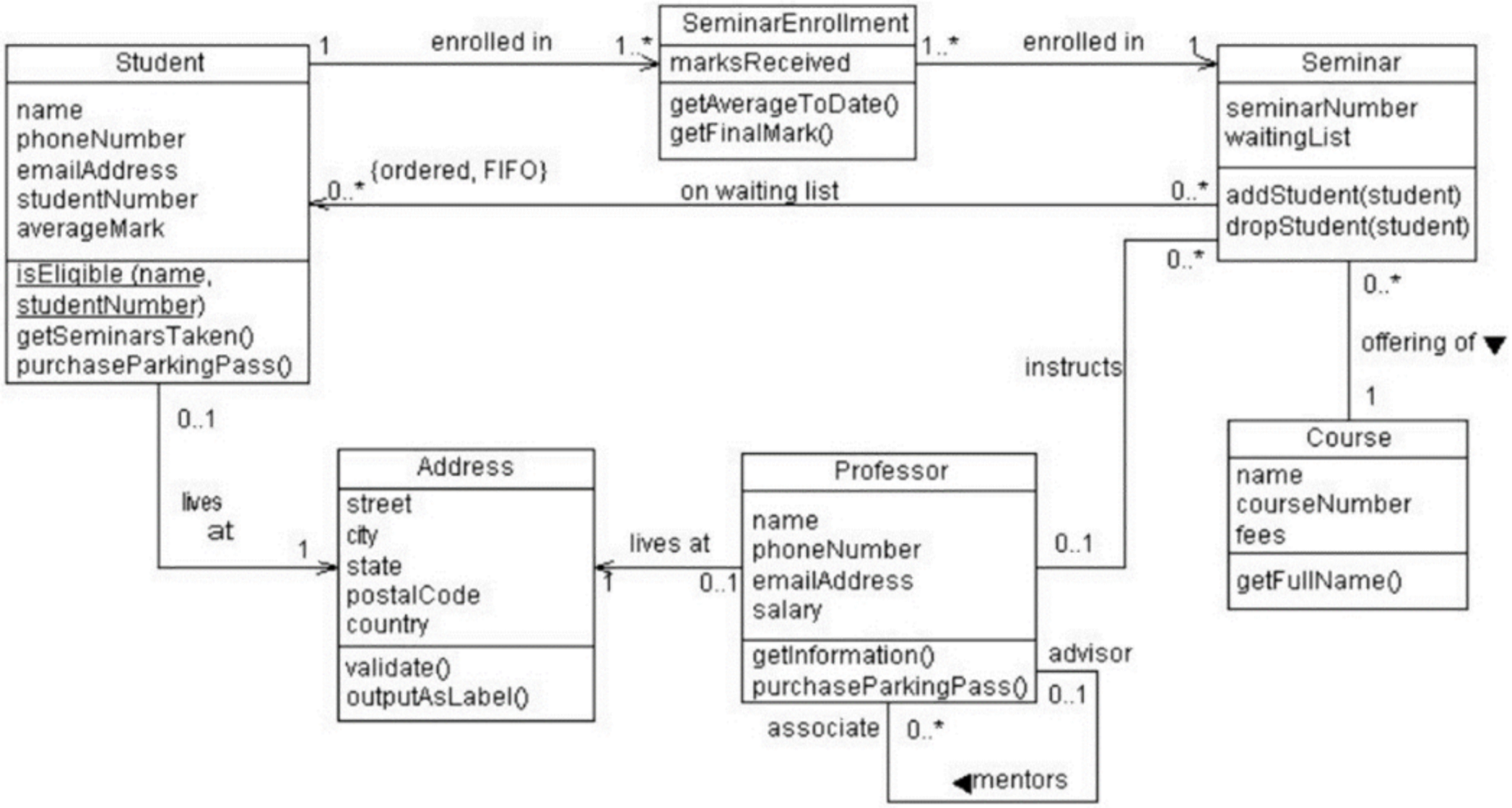
Unified Modeling Language (UML): Class Diagram

- The static structure of a system showing **class structures** in the system and **the relationships among these classes**
- A class structure consists of
  - state (attributes / instance variables)
  - behaviors (operations / methods)



https://www.researchgate.net/figure/Class-diagram-of-the-airport-example_fig1_226891682

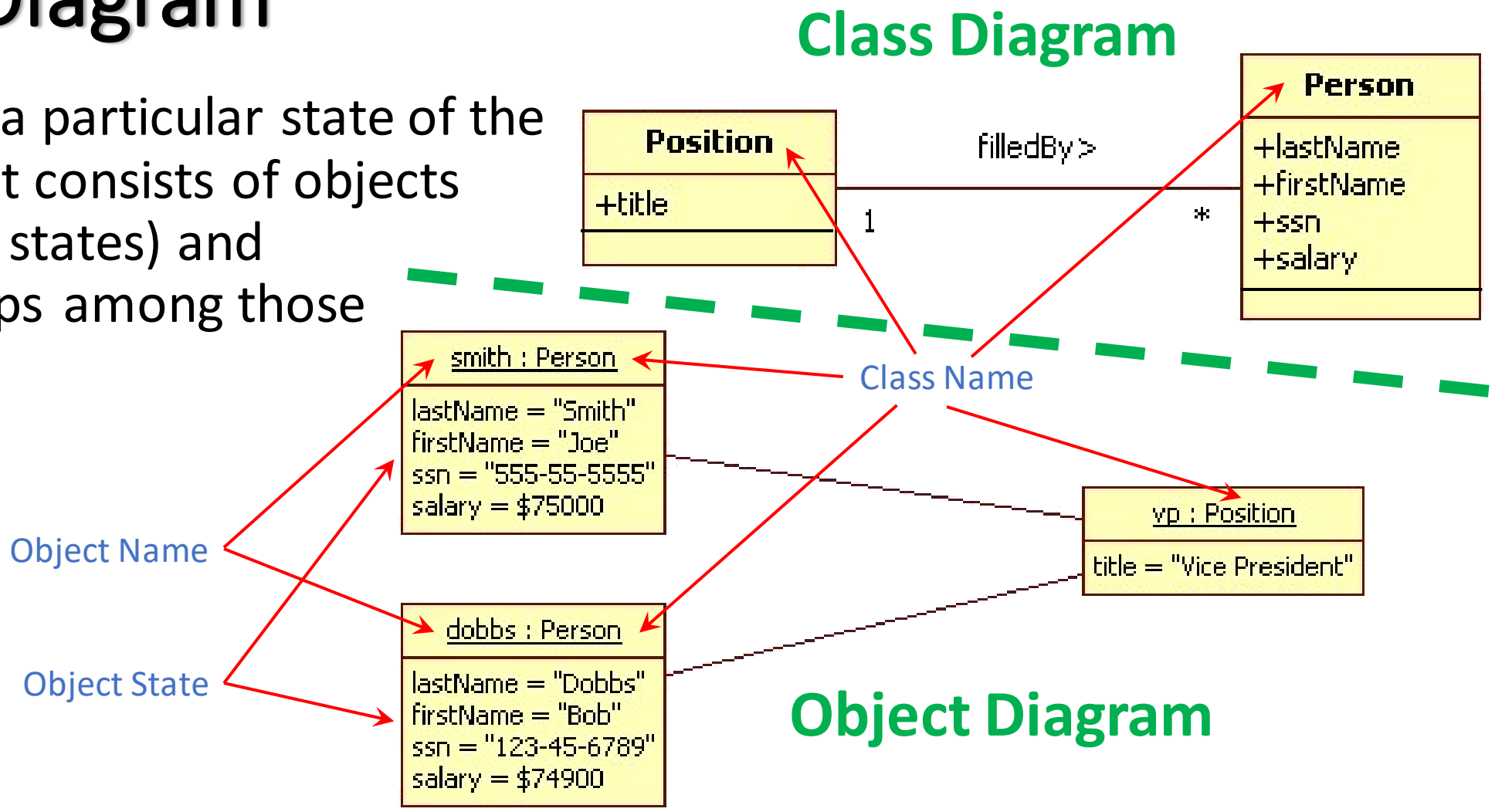
Class Diagram Example



https://www.slideshare.net/Nadia_Nazeer/class-diagrams-64840010

Unified Modeling Language (UML): Object Diagram

- Represent a particular state of the system that consists of objects (with their states) and relationships among those objects



http://www.cs.sjsu.edu/faculty/pearce/oom/ooa/domain/domains.htm