Language Design & Implementation – 6CC509

Lecture 2: Programming Languages

Ioannis Tsioulis

Language Design & Implementation – 6CC509

Lecture 2: Programming Languages

Ioannis Tsioulis

slides acknowledgements:
CSD, University of Crete
A. Whitbrook, U. of Derby
D. Voorhis, U. of Derby
M. Scott, U. of Rochester
D. Watt, U. of Glasgow

Programming Languages

Understand the Distinction: Computational vs Language Paradigms

- Models of Computation (aka Computational Paradigms)
  - Turing Machines
  - Lambda Calculus
  - Cellular Automata
  - Combinatory logic
  - Etc.

- Language Paradigms
  - Some say there are three known, and one yet to be discovered (or at least implemented):
    - http://wiki.c2.com/?ThereAreExactlyThreeParadigms
  - Some say there are many:
    - https://en.wikipedia.org/wiki/Programming_paradigm
  - Some say there are none:
    - http://wiki.c2.com/?ThereAreNoParadigms

p a r a d i g m

noun, plural –ata , (/ˈpærədaɪm/)

1) an outstandingly clear or typical example or archetype
2) a philosophical and theoretical framework of a scientific school or discipline

[Merriam-Webster]

3) Paradigm comes from Greek παράδειγμα (paradeigma), "pattern, example, sample"

In science and philosophy, a paradigm is a distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to a field.
In rhetoric, the purpose of paradeigma is to provide an audience with an illustration of similar occurrences.

Understand the Distinction: Syntax vs Semantics

pragmatics

- Language = syntax + semantics

What you can say.

a = 3;

What you say *does*. (Literally, what it *means.)*

"The value denoted by evaluating the literal expression '3' is assigned to the variable identified by the name 'a'."

s e m a n t i c s

noun, plural, (/siˈmantiks/)

1) the study of the meanings of words and phrases in language
2) the meanings of words and phrases in a particular context

[Merriam-Webster]

3) Semantics comes from Greek σημασία/σημαντικός (sēmantikós) is the study of reference, meaning, or truth.

In programming language theory, semantics is the field concerned with the rigorous mathematical study of the meaning of programming languages.
Semantics describes the processes a computer follows when executing a program in a specific language.

[Wikipedia]

p r a g m a t i c s

noun, plural, (/prag'matiks/)

1) the relation between signs or linguistic expressions and their users
2) the relationship of sentences to the environment in which they occur

[Merriam-Webster]

3) In linguistics and related fields, pragmatics is the study of how context contributes to meaning

Theories of pragmatics go hand-in-hand with theories of semantics, which studies aspects of meaning, and syntax which examines sentence structures, principles, and relationships.

[Wikipedia]

Understand the Distinction: Imperative vs Declarative

- Imperative
  1. Do this first.
  2. Then do that.
  3. Do the other thing last.
- Declarative
  - There are three things to do.

Understand the Distinction: Procedural vs Functional

- A language isn't "functional" just because it has a construct called 'function'.
  - C is not a functional programming language, though it might be used to build functional programming languages.
  - JavaScript is closer to being a functional programming language than C.
- It's helpful to understand why. (Homework!)

Understand the Distinction: Languages vs IDEs or Editors

- Visual Studio
- RAD Studio
- RStudio
- Emacs
- Eclipse
- Netbeans
- IntelliJ
- Notepad++

Editors/IDEs (generally) aren't languages.

They're usually not interesting in language terms, but interesting Editor/IDE features might be language dependent.

Test: Can the language exist and be used without these?

There are visual programming languages *entirely* dependent on a graphical environment.

Despite the name, Visual Studio isn't one of them.

Understand the Distinction: Languages vs Libraries

Is a language standard library…

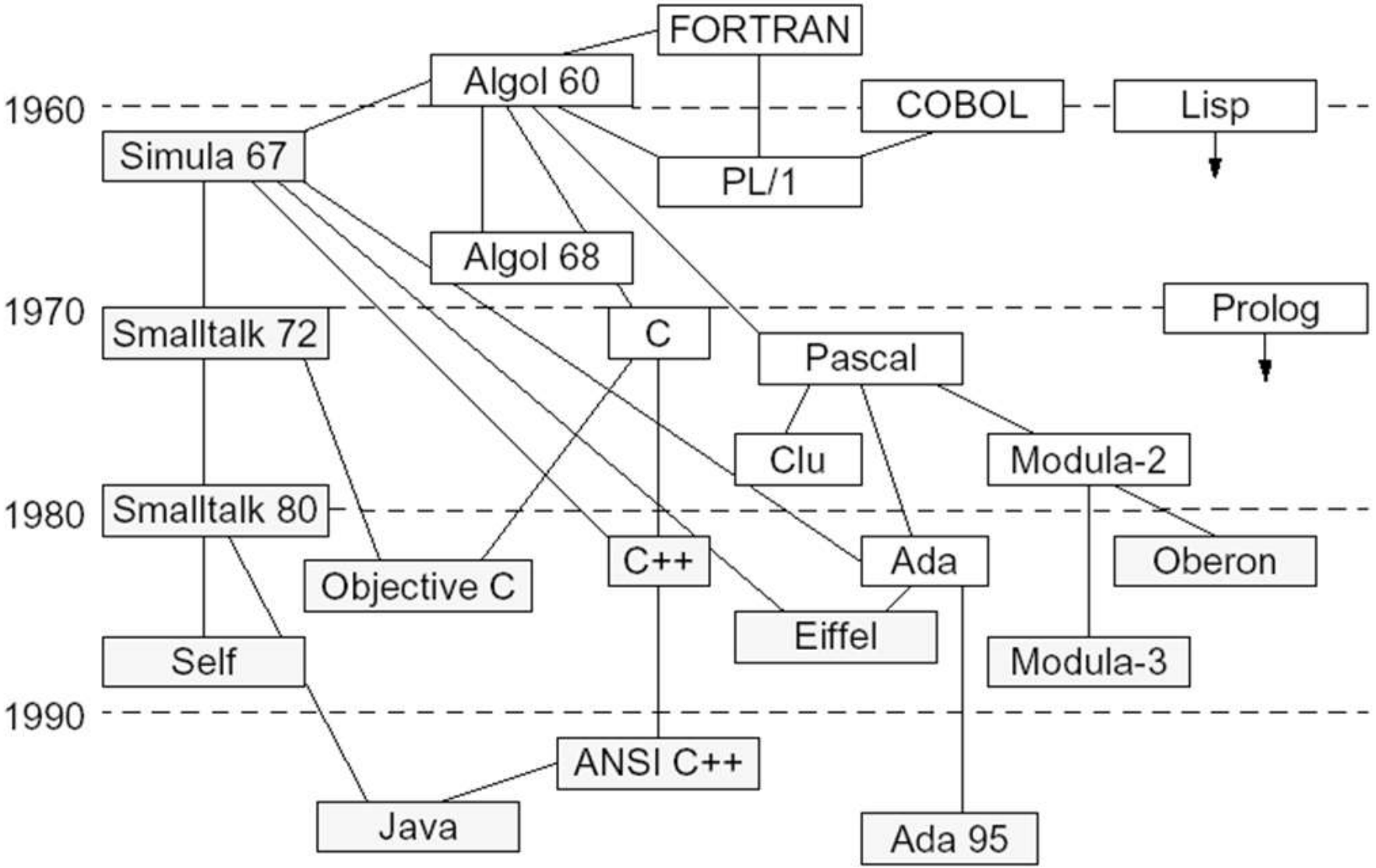
- E.g. the C standard library (libc), C++ STL, C++ Standard Library, .NET Framework, Java Platform, etc.

…part of the language?

Programming Languages Timeline

- **FlowMatic**
  - 1955 Grace Hopper, UNIVAC
- **ForTran**
  - 1956 John Backus, IBM
- **AlgOL**
  - 1958 ACM Language Committee
- **LISP**
  - 1958 John McCarthy, MIT
- **CoBOL**
  - 1960 Committee on Data Systems Languages
- **BASIC**
  - 1964 John Kemeny & Thomas Kurtz, Dartmouth
- **PL/I**
  - 1964 IBM Committee
- **Simula**
  - 1967 Norwegian Computing Center, Kristen Nygaard & Ole-Johan Dahl
- **Logo**
  - 1968 Seymour Papert, MIT
- **Pascal**
  - 1970 Nicklaus Wirth, Switzerland
- **C**
  - 1972 Dennis Ritchie & Kenneth Thompson, Bell Labs
- **Smalltalk**
  - 1972 Alan Kay, Xerox PARC
- **Objective C**
  - 1985 Brad Cox, Stepstone Systems
- **C++**
  - 1986 Bjarne Stroustrup, Bell Labs
- **Eiffel**
  - 1989 Bertrand Meyer, France
- **Visual BASIC**
  - 1990 Microsoft
- **Delphi/Object Pascal**
  - 1995 Borland
- **Object CoBOL**
  - 1995 MicroFocus
- **Java**
  - 1995 Sun Microsystems

Programming Languages History



Five Generations of Programming Languages

- First: **Machine** Languages
  - machine codes
- Second: **Assembly** Languages
  - symbolic assemblers
- Third: High Level **Procedural** Languages
  - (machine independent) imperative languages
- Fourth: **Non-procedural** Languages
  - domain specific application generators
- Fifth: **Natural** Languages

Each generation is at a higher level of abstraction

The First Generation (1940s)

- In the beginning … was the **Stone Age**: **Machine Languages**
  - Binary instruction strings
  - Introduced with the first programmable computer
  - Hardware dependent

I need to calculate the total sales.
The sales tax rate is 10%.
To write this program, I'll multiply the purchase price by the tax-rate and add the purchase price to the result.
I'll store the result in the total sales field.

State the problem

I need to:
Load the purchase price
Multiply it by the sales tax
Add the purchase price to the result
Store the result in total price

Translate into the instruction set

I need to know:
What is the instruction to load from memory?
Where is purchase price stored?
What is the instruction to multiply?
What do I multiply by?
What is the instruction to add from memory?
What is the instruction to store back into memory?

Translate into machine operation codes (op-codes)

```
187E:0100 75 17 80 3E 0D
187E:0110 B9 FF FF 8B D1
187E:0120 42 33 C9 8B D1
187E:0130 5B FF BE E7 04
187E:0140 01 BF 01 00 CD
187E:0150 47 18 A2 19 00
187E:0160 2B F1 58 C3 73
187E:0170 B4 59 CD 21 59
```

Program entered and executed as machine language

The Second Generation (Early 1950s)

- Then we begin to study **improvements**: **Assembly Languages**
  - 1-to-1 substitution of mnemonics for machine language commands
  - Hardware dependent

I need to calculate the total sales.
The sales tax rate is 10%.
To write this program, I'll multiply the purchase price by the tax-rate and add the purchase price to the result.
I'll store the result in the total sales field.

State the problem

I need to:
Load the purchase price
Multiply it by the sales tax
Add the purchase price to the result
Store the result in total price

Translate into the instruction set

The **ASSEMBLER** converts instructions to op-codes:
What is the instruction to load from memory?
Where is purchase price stored?
What is the instruction to multiply?
What do I multiply by?
What is the instruction to add from memory?
What is the instruction to store back into memory?

```
POP SI
MOV AX,[BX+03]
SUB AX,SI
MOV WORD PTR
[TOT_AMT],E0D7
MOV WORD PTR
[CUR_AMT],E1DB
ADD [TOT_AMT],AX
```

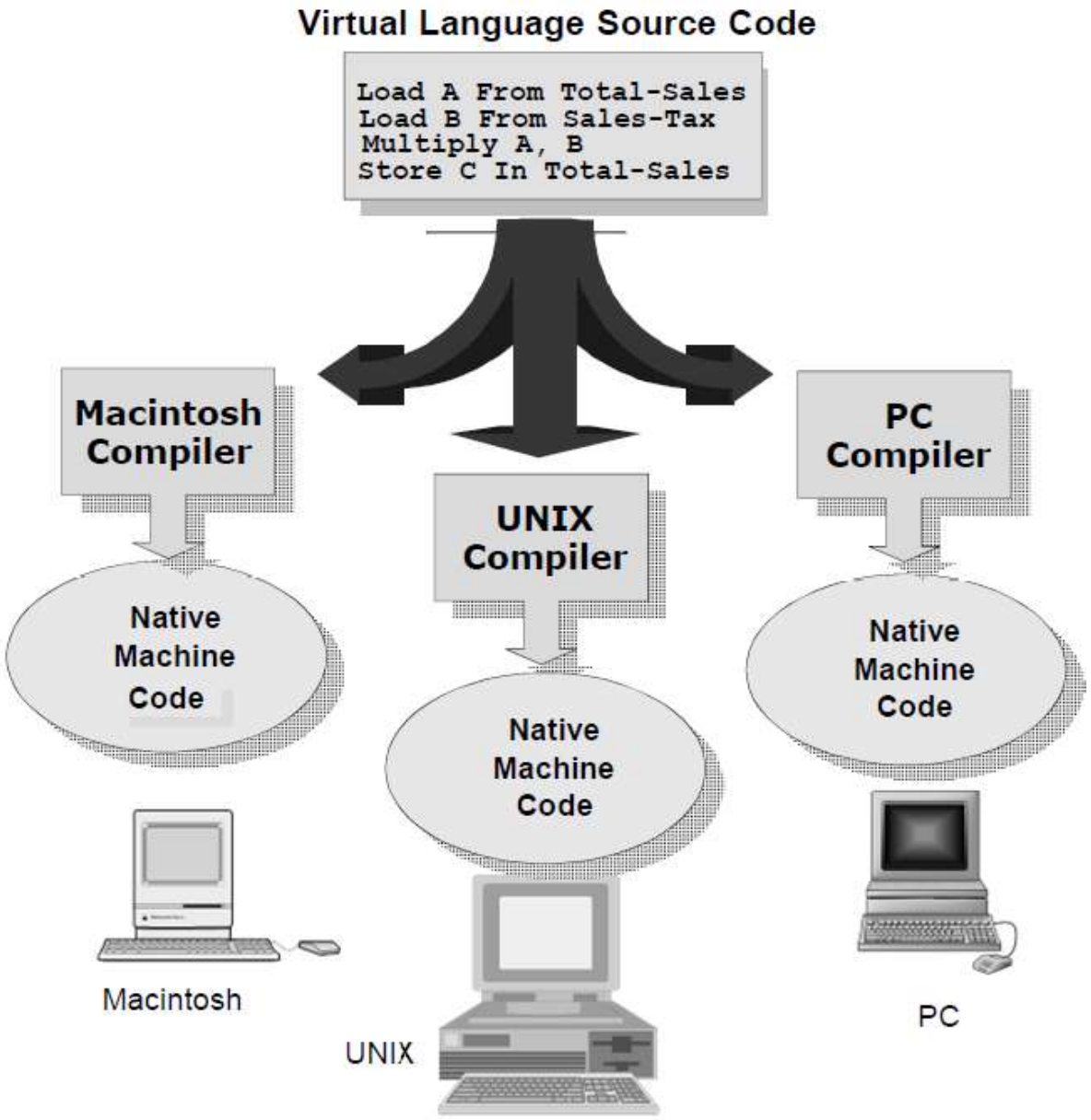
Translate into machine operation codes (op-codes)

```
187E:0100 75 17 80 3E 0D
187E:0110 B9 FF FF 8B D1
187E:0120 42 33 C9 8B D1
187E:0130 5B FF BE E7 04
187E:0140 01 BF 01 00 CD
187E:0150 47 18 A2 19 00
187E:0160 2B F1 58 C3 73
187E:0170 B4 59 CD 21 59
```

Program executed as machine language

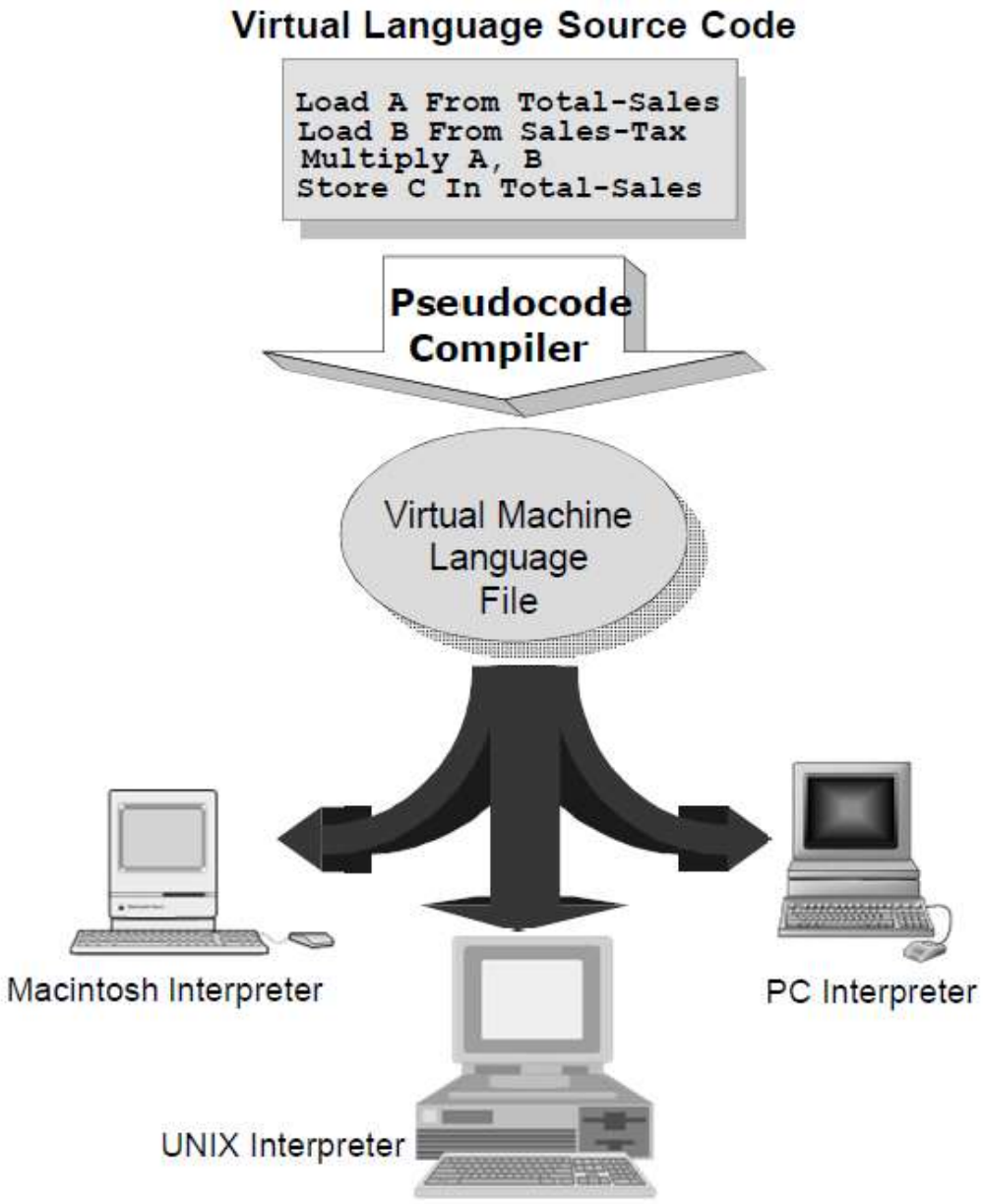
The Second Generation (1950s)

- The invention of the **Compiler**
  - Grace Murray Hopper (Flowmatic)
- Each CPU has its own **specific machine language**
  - A program must be translated into machine language before it can be executed on a particular type of CPU



The Second Generation (1950s)

- **Interpreters** and **Virtual Machine Languages**
  - Speedcoding
  - UNCOL
- Virtual Machine Languages are intermediaries between the statements and operators of high-level programming languages and the register numbers and operation codes of native machine



The Third Generation (1955-65)

- **High-level Procedural Languages** make programming easier
- FORTRAN, ALGOL, LISP, COBOL, BASIC, PL/I

I need to calculate the total sales.
The sales tax rate is 10%.
To write this program, I'll multiply the purchase price by the tax-rate and add the purchase price to the result.
I'll store the result in the total sales field.

State the problem

The **COMPILER** translates:
Load the purchase price
Multiply it by the sales tax
Add the purchase price to the result
Store the result in total price

```
salesTax = purchasePric * TAX_RATE;
totalSales = purchasePrice + salesTax;
```

Translate into the instruction set

The **ASSEMBLER** converts instructions to op-codes:
What is the instruction to load from memory?
Where is purchase price stored?
What is the instruction to multiply?
What do I multiply by?
What is the instruction to add from memory?
What is the instruction to store back into memory?

```
POP SI
MOV AX,[BX+03]
SUB AX,SI
MOV WORD PTR
[TOT_AMT],E0D7
MOV WORD PTR
[CUR_AMT],E1DB
ADD [TOT_AMT],AX
```

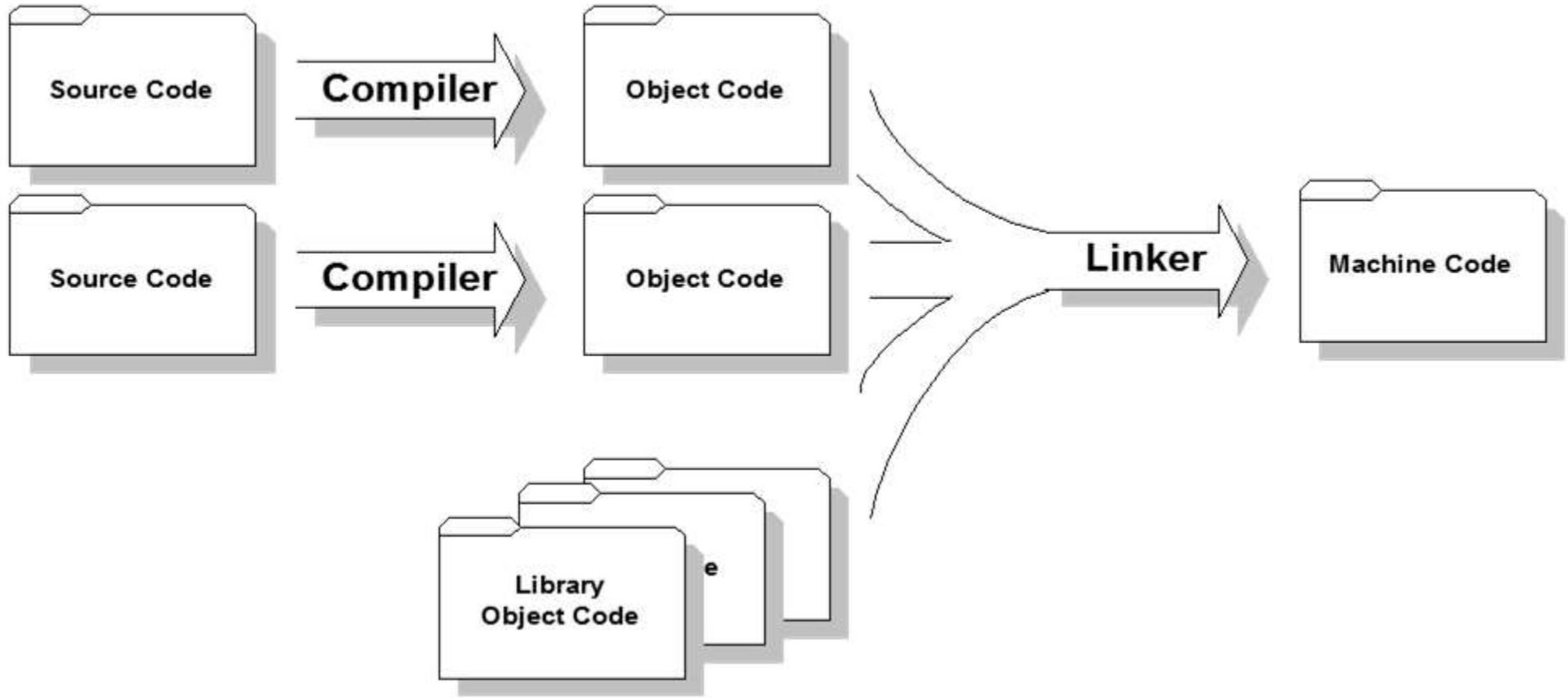
Translate into machine operation codes (op-codes)

```
187E:0100 75 17 80 3E 0D
187E:0110 B9 FF FF 8B D1
187E:0120 42 33 C9 8B D1
187E:0130 5B FF BE E7 04
187E:0140 01 BF 01 00 CD
187E:0150 47 18 A2 19 00
187E:0160 2B F1 58 C3 73
187E:0170 B4 59 CD 21 59
```

Program executed as machine language

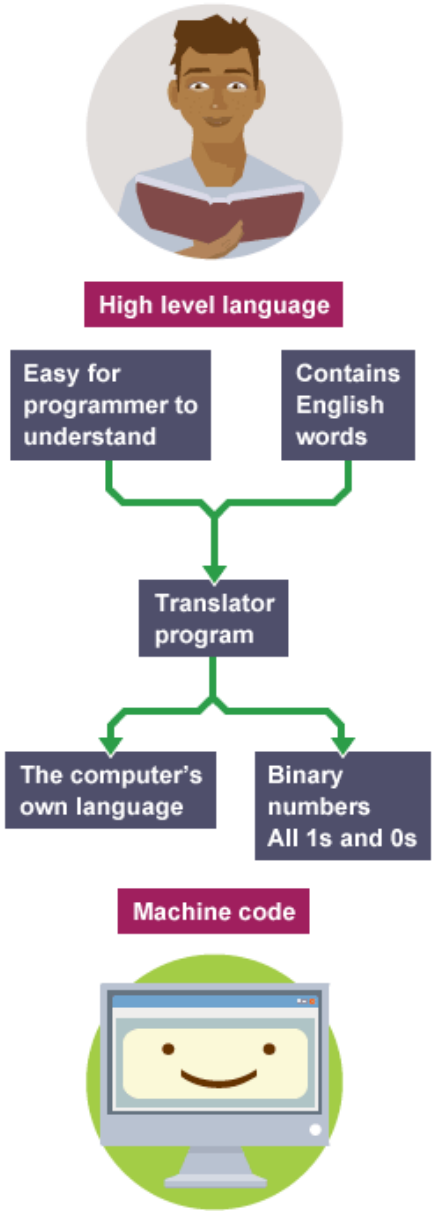
The Conventional Programming Process

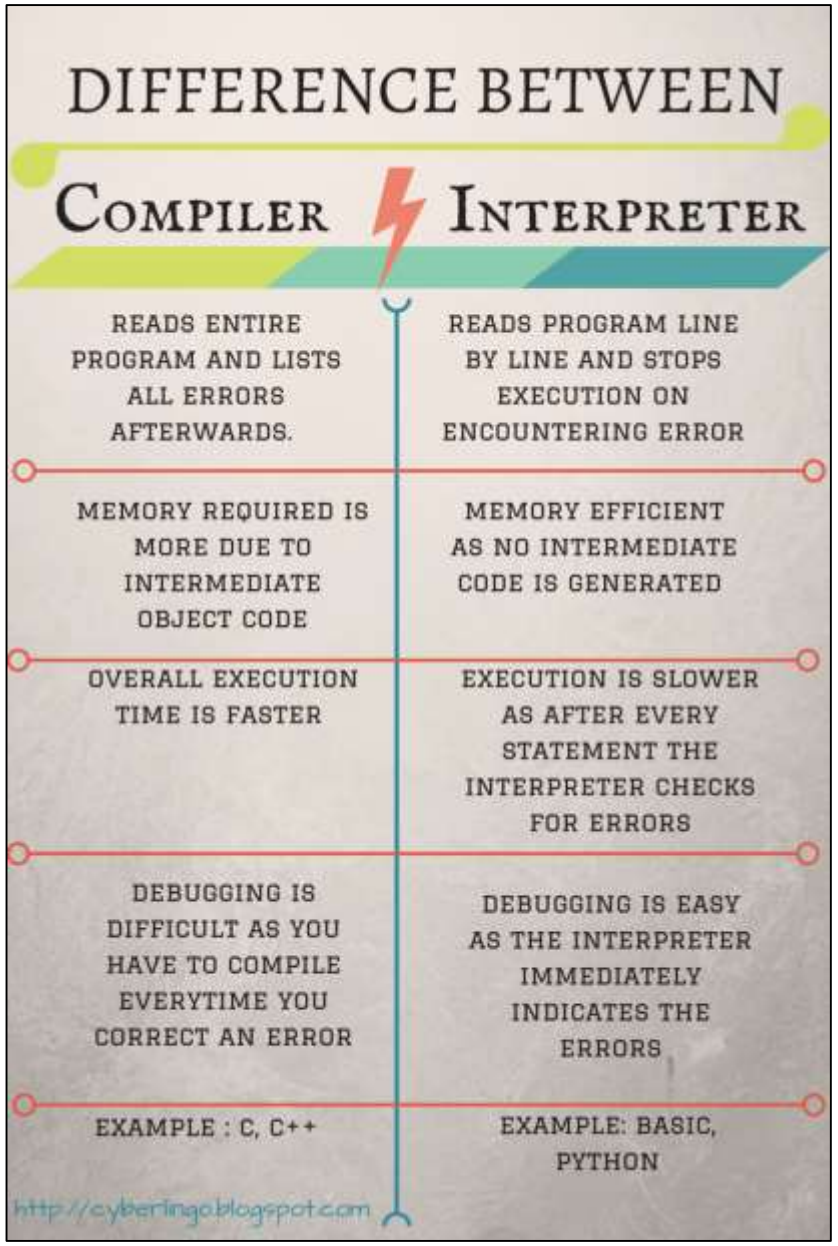
- A **compiler** is a software tool which translates source code into a specific target language for a particular CPU type
- A **linker** combines several object programs eventually developed independently



Fourth Generation Languages (1980)

- **Non-procedural Languages** (problem-oriented)
  - User specifies what is to be done not how it is to be accomplished
  - Less user training is required
  - Designed to solve specific problems
- Diverse Types of 4GLs
  - Spreadsheet Languages
  - Database Query Languages
  - Decision Support Systems
  - Statistics
  - Simulation
  - Optimization
  - Decision Analysis
  - Graphics Systems





Interpreter

- Translates **each line** of the program into the appropriate **machine code** as the program is executed.

Compiler

- Translates the **source code** for a program into **executable code**
- Translates either to:
  - **Machine code**, or
  - An **intermediate language** (see later)

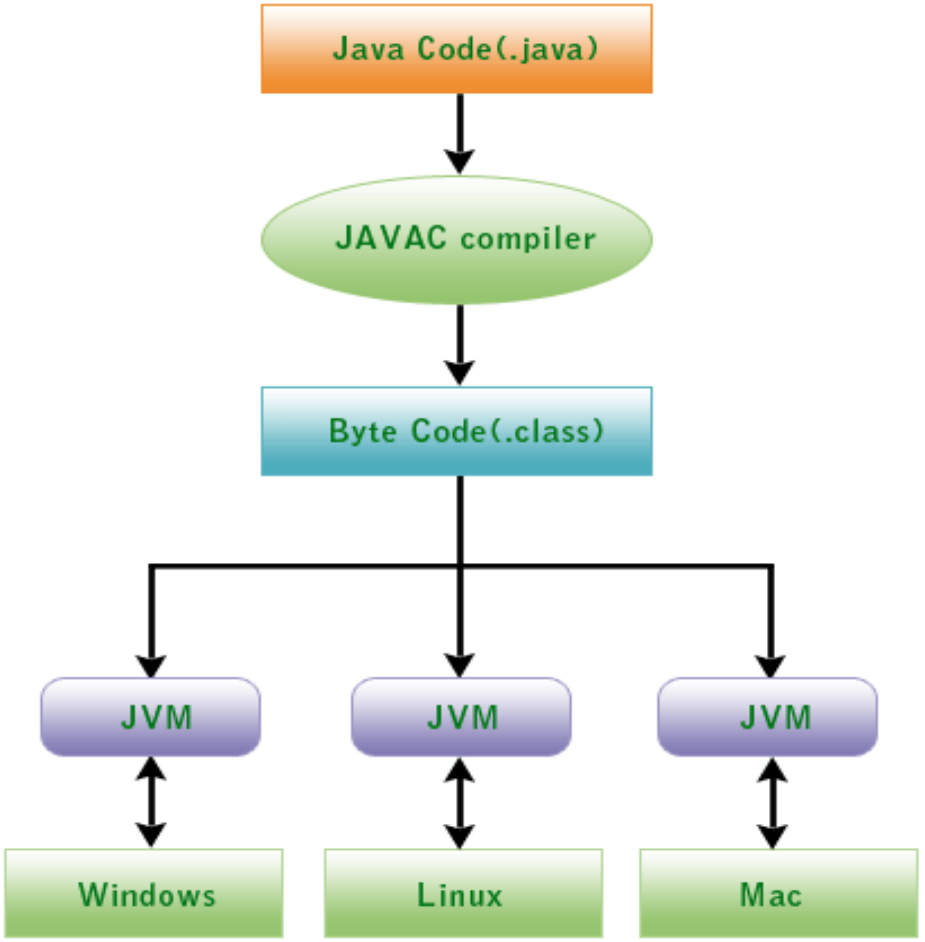
Managed Code

Managed Environments

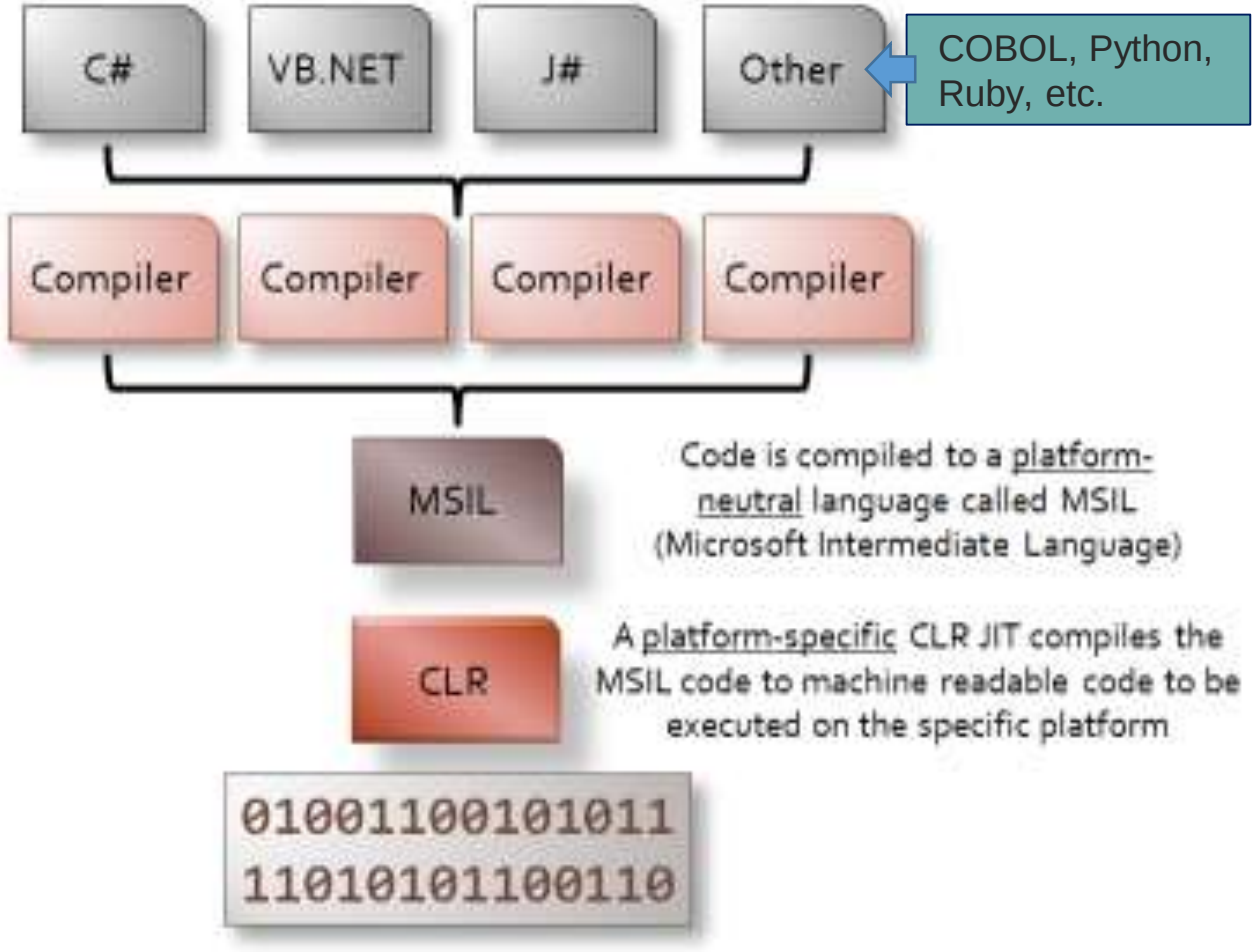
- In recent years, the focus for a lot of application development has shifted to the use of ***Managed Code***.
- In **Managed Environments**, the compiler compiles the source code to an ***intermediate language***
- When the program is run, the intermediate language is run under the control of a ***run time system*** or ***virtual machine*** that prevents the program from doing things it should not be able to.
- The run-time system or virtual machine also includes a ***garbage collector*** to ensure that all memory used by the program is released when it is no longer needed.
- The primary environments that run managed code today are:
  - **Java**
  - **.NET**

Managed Code: Java vs .NET

Java



.NET

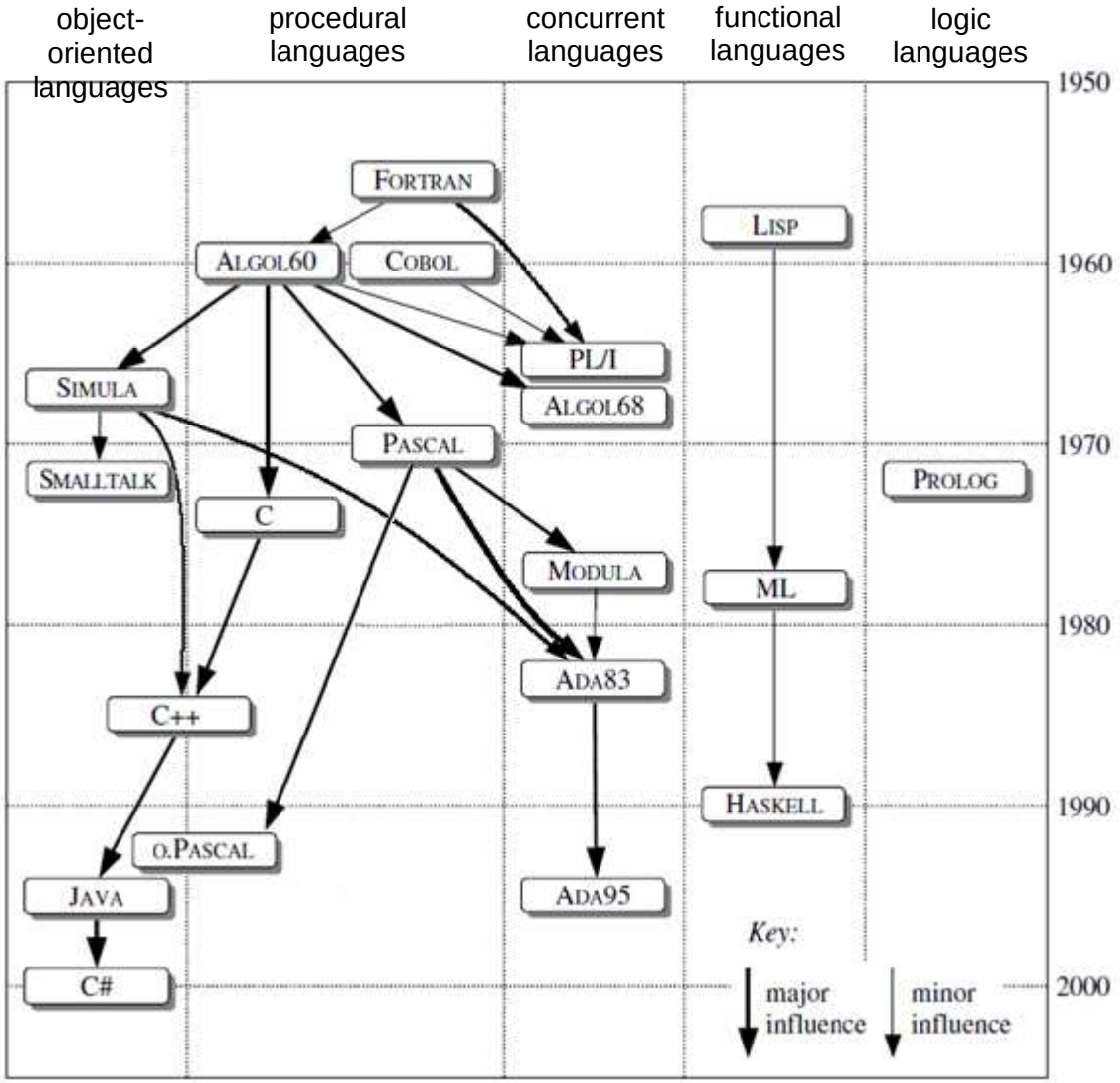


How do Programming Languages Differ?

- **Common Constructs**:
  - basic data types (numbers, etc.);
  - variables;
  - expressions;
  - statements;
  - keywords;
  - control constructs;
  - procedures;
  - comments;
  - errors ...

- **Uncommon Constructs**:
  - type declarations;
  - special types (strings,
  - arrays, matrices,...);
  - sequential execution;
  - concurrency constructs;
  - packages/modules;
  - objects;
  - general functions;
  - generics;
  - modifiable state;...

Historical development of major programming languages



D. Watt (2004), Programming Language Design Concepts, Wiley.

- Different selections of key concepts support radically different styles of programming, which are called **paradigms**.
- There are six major paradigms.
  - **Procedural Languages** is characterized by the use of variables, commands, and procedures;
  - **Object-oriented Languages** by the use of objects, classes, and inheritance;
  - **Concurrent Languages** by the use of concurrent processes, and various control abstractions;
  - **Functional Languages** by the use of functions;
  - **Logic Languages** by the use of relations; and
  - **Scripting Languages** by the presence of very high-level features

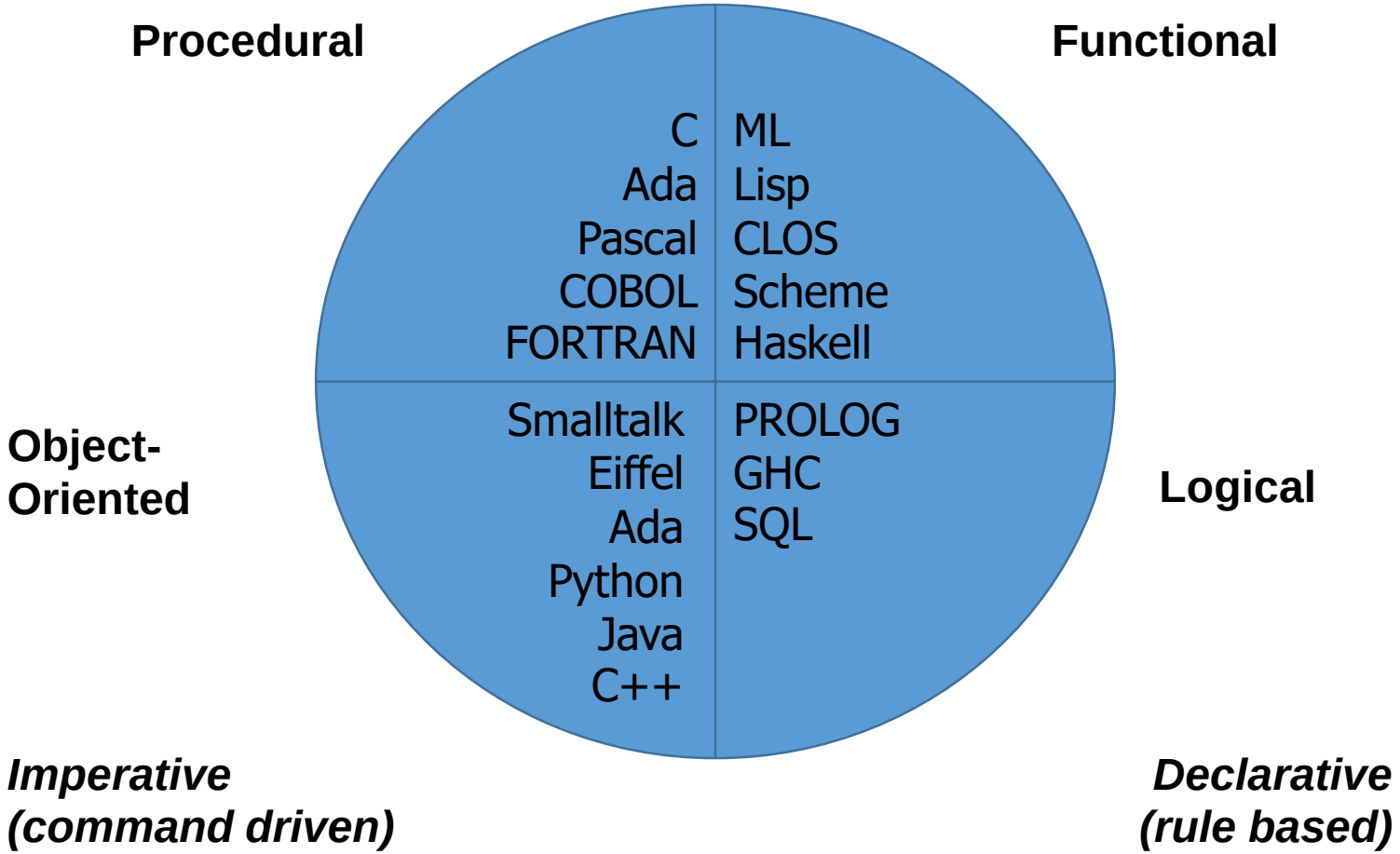
Language Styles …

- The top-level division distinguishes between
  - **Imperative Languages**
    - the focus is on how the computer should do it.
  - **Declarative Languages**
    - the focus is on what the computer is to do

- **Procedural Languages**
  - Individual statements
    - FORTRAN, ALGOL, Cobol, Pascal, C, Ada
- **Object-oriented Languages**
  - Bring together data and operations
    - Smalltalk, C++, Eiffel, Python, Ada95, Java
- **Functional Languages**
  - When you tell the computer to do something it does it
    - LISP, Scheme, CLOS, ML, Haskell
- **Logic Languages**
  - Inference engine that drives things
    - Prolog, GHC

Wikipedia: List of programming languages by type

… and Programming Paradigms



Imperative Programming

- It is the oldest but still the dominant paradigm
  - It is based on commands that update variables held in storage
  - Variables and assignment commands constitute a simple but useful abstraction from *the memory fetch and update of machine instruction sets* (von Neumann model)
  - Imperative programming languages can be implemented very efficiently
- Why imperative paradigm still dominant?
  - It is related to the nature and purpose of programming
- What is a program?
  - Programs are written to model real-world processes affecting real-world objects
  - Imperative programs model such processes
  - Variables model such objects

Declarative Programming

- Whereas **imperative** (von Neumann) languages are based on statements (assignments in particular) that influence subsequent computation via the side effect of changing a value in memory, **declarative** languages are based on expressions that have values.

Most of the functional and logic languages include some imperative features.

Procedural Languages

- The **procedural** languages are the most familiar and successful.
- The basic means of computation is the modification of variables through procedures and functions.
  - They include Fortran, Ada 83, C, and Pascal.

The division between the procedural and object-oriented languages is often very fuzzy.

Some object-oriented languages still keep their procedural flavour, such as Object Pascal (Delphi) and C++.

Object-oriented Languages

- **Object-oriented** languages trace their roots to Simula 67.
- Most are closely related to procedural languages, but have a much more structured and distributed model of both memory and computation.
- Rather than picture computation as the operation of a monolithic processor on a monolithic memory, object-oriented languages picture it as interactions among semi-independent objects, each of which has both its own internal state and subroutines to manage that state.
  - Smalltalk is the purest of the object-oriented languages
  - C++ and Java are the most widely used
  - It is also possible to devise object-oriented functional languages (the best known of these is the CLOS extension to Common Lisp), but they tend to have a strong imperative flavour

Functional Languages

- **Functional** languages employ a computational model based on the recursive definition of functions.
- They take their inspiration from the lambda calculus, a formal computational model developed by Alonzo Church in the 1930s.
- In essence, a program is considered a function from inputs to outputs, defined in terms of simpler functions through a process of refinement.
  - Languages in this category include Lisp, ML, and Haskell.

Logic Languages

- **Logic** or **constraint-based** languages take their inspiration from predicate logic.
- They model computation as an attempt to find values that satisfy certain specified relationships, using goal-directed search through a list of logical rules.
  - Prolog is the best-known logic language.
  - The term is also sometimes applied to the SQL database language, the XSLT scripting language, and programmable aspects of spreadsheets such as Excel and its predecessors.

Scripting Languages

- **Scripting** languages are a subset of the von Neumann languages.
- They are distinguished by their emphasis on "gluing together" components that were originally developed as independent programs.
- Several scripting languages were originally developed for specific purposes:
  - csh and bash, for example, are the input languages of job control (shell) programs
  - Awk was intended for report generation
  - PHP and JavaScript are primarily intended for the generation of web pages with dynamic content (with execution on the server and the client, respectively)
  - Other languages, including Perl, Python, Ruby, and Tcl, are more deliberately general purpose
- Most place an emphasis on rapid prototyping, with a bias toward ease of expression over speed of execution.

Concurrent Languages

- **Concurrent** (parallel) languages also form a separate class, but the distinction between concurrent and sequential execution is mostly independent of the classifications above.
  - Most concurrent programs are currently written using special library packages or compilers in conjunction with a sequential language such as Fortran or C.
  - A few widely used languages, including Java, C#, and Ada, have explicitly concurrent features.

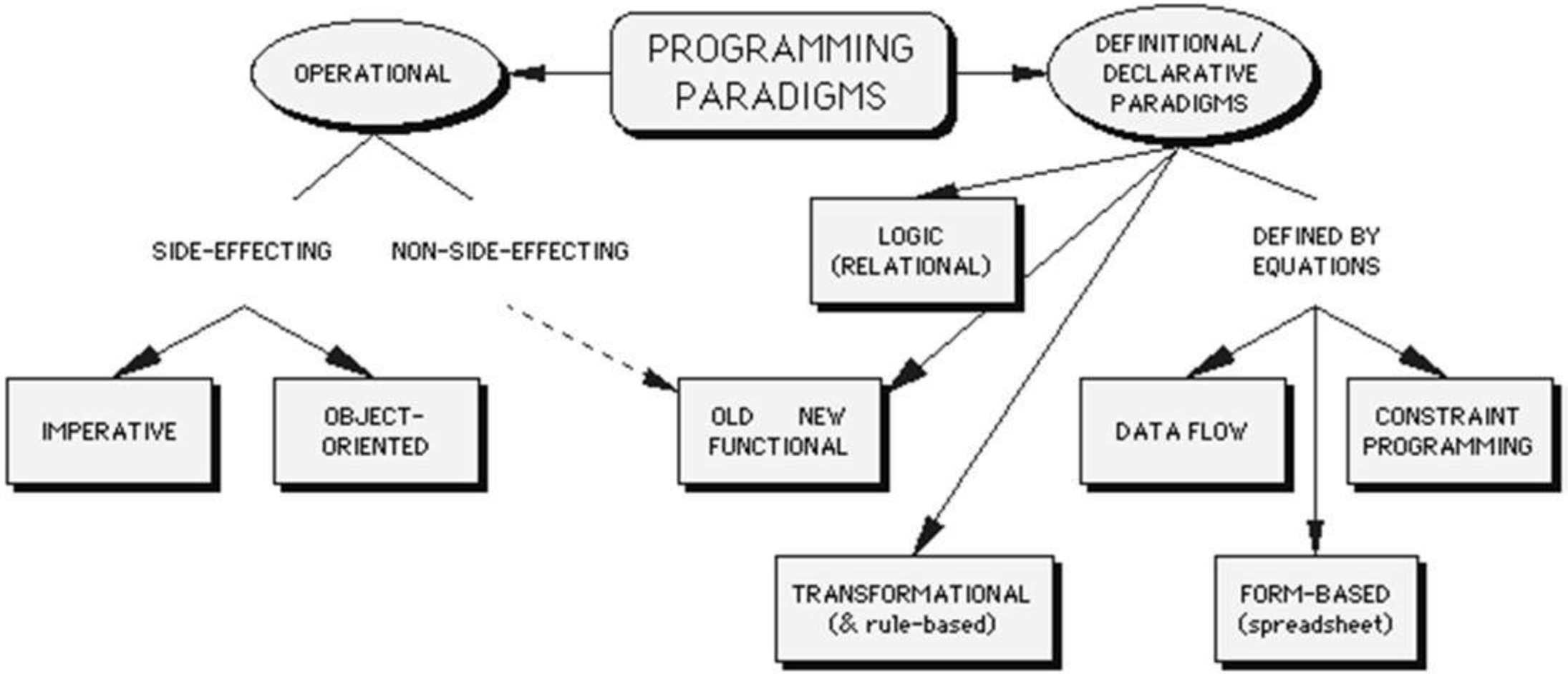
Programming Paradigms

- A programming language is a problem-solving tool

Imperative Languages		
Procedural:	program = algorithms + data	good for decomposition
Object-oriented:	program = objects + messages	good for encapsulation

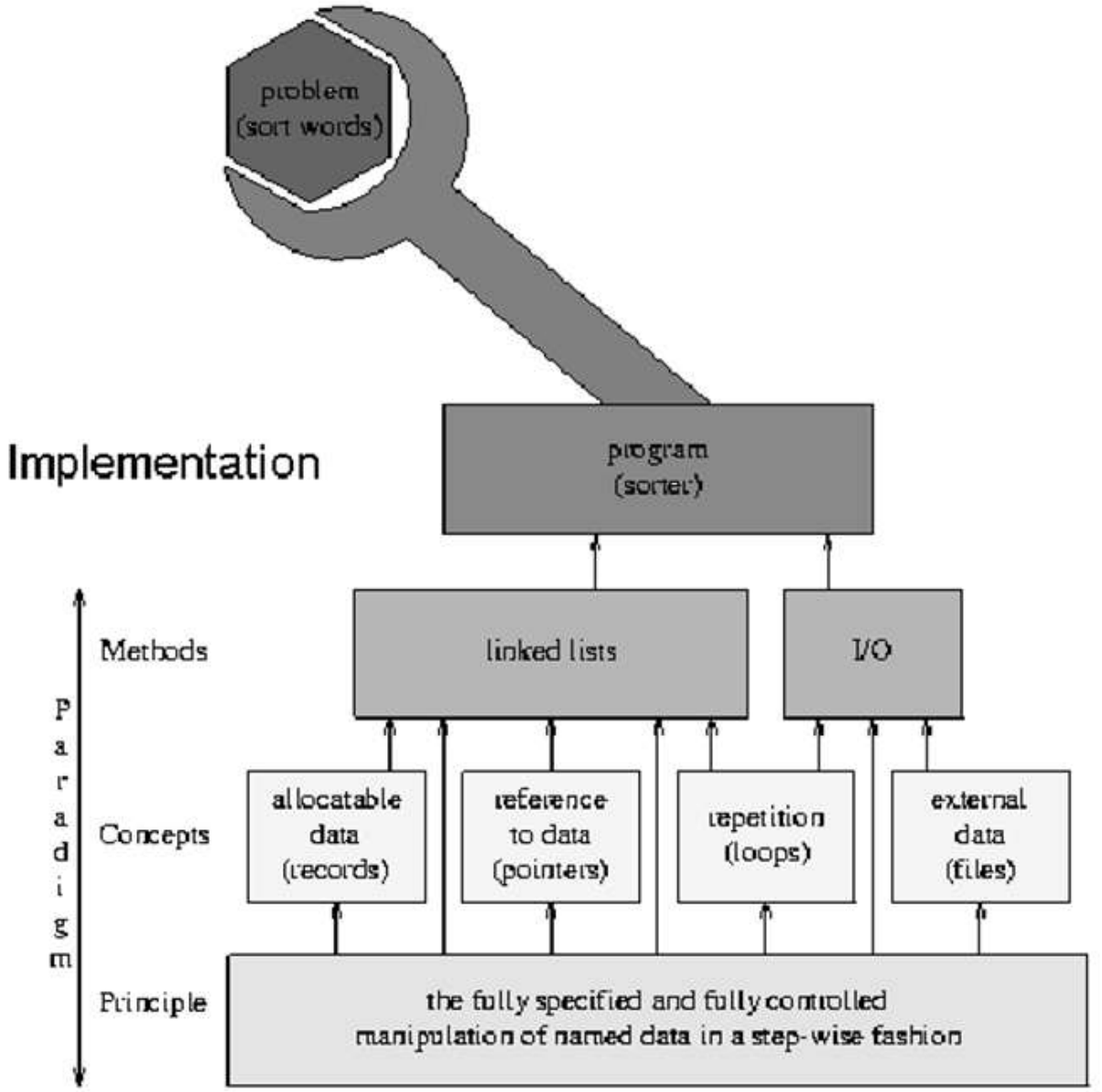
Declarative Languages		
Functional:	program = functions . functions	good for reasoning
Logic programming:	program = facts + rules	good for searching

Programming Paradigms



What is a Programming Paradigm ?

- A set of **coherent abstractions** usec effectively model a problem/domain
- A mode of thinking aka a **programm methodology**

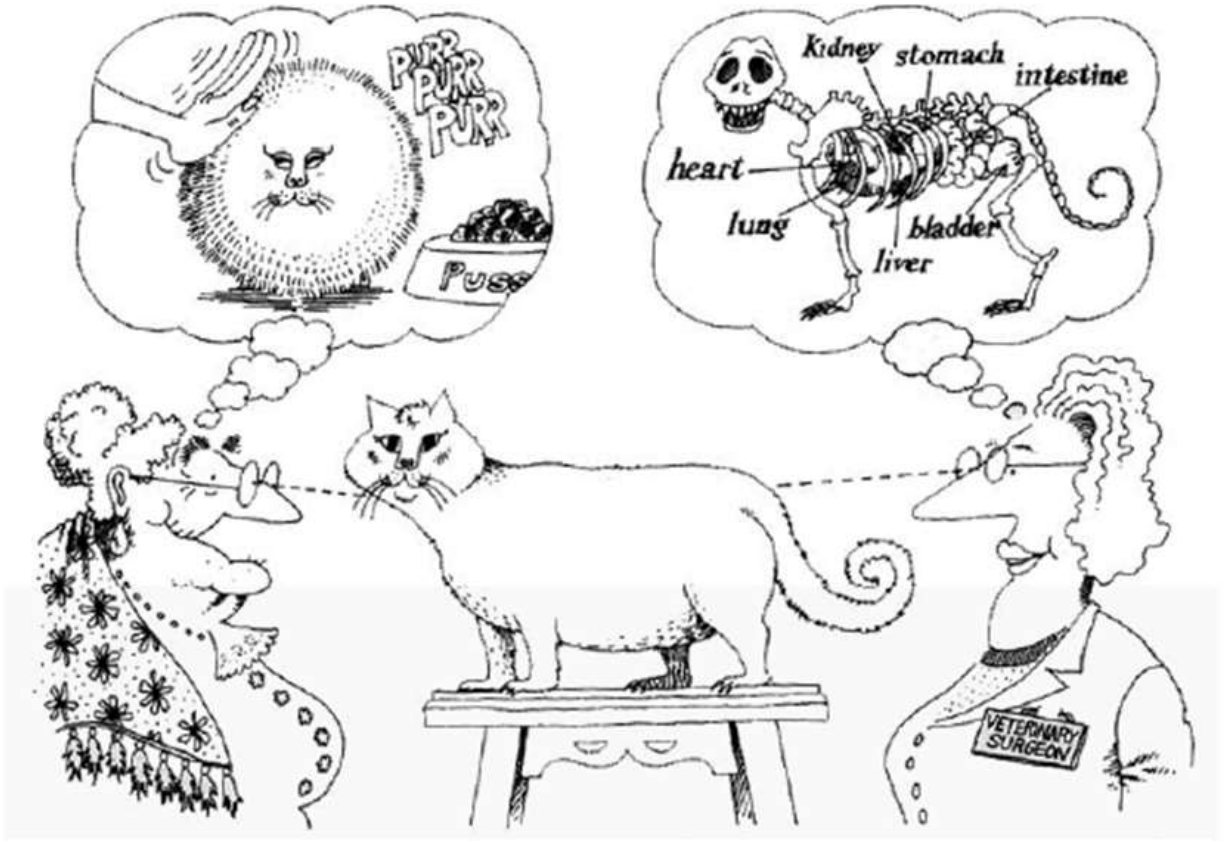


What is an Abstraction ?

- The intellectual tool that allows us to deal with concepts apart from particular instances of those concepts (Fairley, 1985)
- An **abstraction** denotes the essential characteristics of an object that distinguish it from all other objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer (Booch, 1991)
- **Abstraction**, as a process, denotes the extracting of the essential details about an item, or a group of items, while ignoring the inessential details
- **Abstraction**, as an entity, denotes a model, a view or some other focused representation for an actual item (Berard, 1993)
- **Abstraction** is the separation of the logical properties of data or function from their implementation (Dale and Lily, 1995)

What is an Abstraction ?

- In summary, abstraction allows us access to the relevant information regarding a problem/domain, and ignores the remainder
- Abstraction is a technique to manage, and cope with, the complexity of the tasks we perform
  - The ability to model at the right level a problem/domain, while ignore the rest
- The use of abstraction, allows us to
  - control the level and amount of detail,
  - communicate effectively with users



Focus is on the essential characteristics of some object which yields clearly defined boundaries
It is relative to the perspective of the viewer

- ***The history of PLs is a long road towards richer abstraction forms***

Examples of Abstractions in PLs

- **Procedural** (abstraction of a statement) allows us to introduce new operations
  - Using the name of a sequence of instructions in place of the sequence of instructions
  - Parameterization allows high level of flexibility in the performance of operations
- **Data** (abstraction of a data type) allows us to introduce new types of data
  - A named collection that describes a data object
  - Provides a logical reference to the data object without concern for the underlying memory representation
- **Control** (abstraction of access details) allows us to iterate over items without knowing how the items are stored or obtained
  - A way of indicating the desired effect without establishing the actual control mechanism
  - Allows designers to model iteration (e.g., Iterator), concurrency, and synchronization

Programming Methodologies & Abstraction

Programming Methodologies	Abstraction Concepts	Programming Languages Constructs
Structured Programming	Explicit Control Structures	Do-while and other loops Blocks and so forth
Modular Programming	Information Hiding	Modules with well-defined interfaces
Abstract Data Types Programming	Data Representation Hiding	User-defined Data Types
Object-Oriented Programming	Reusing Artifacts	Classes, Inheritance, Polymorphism

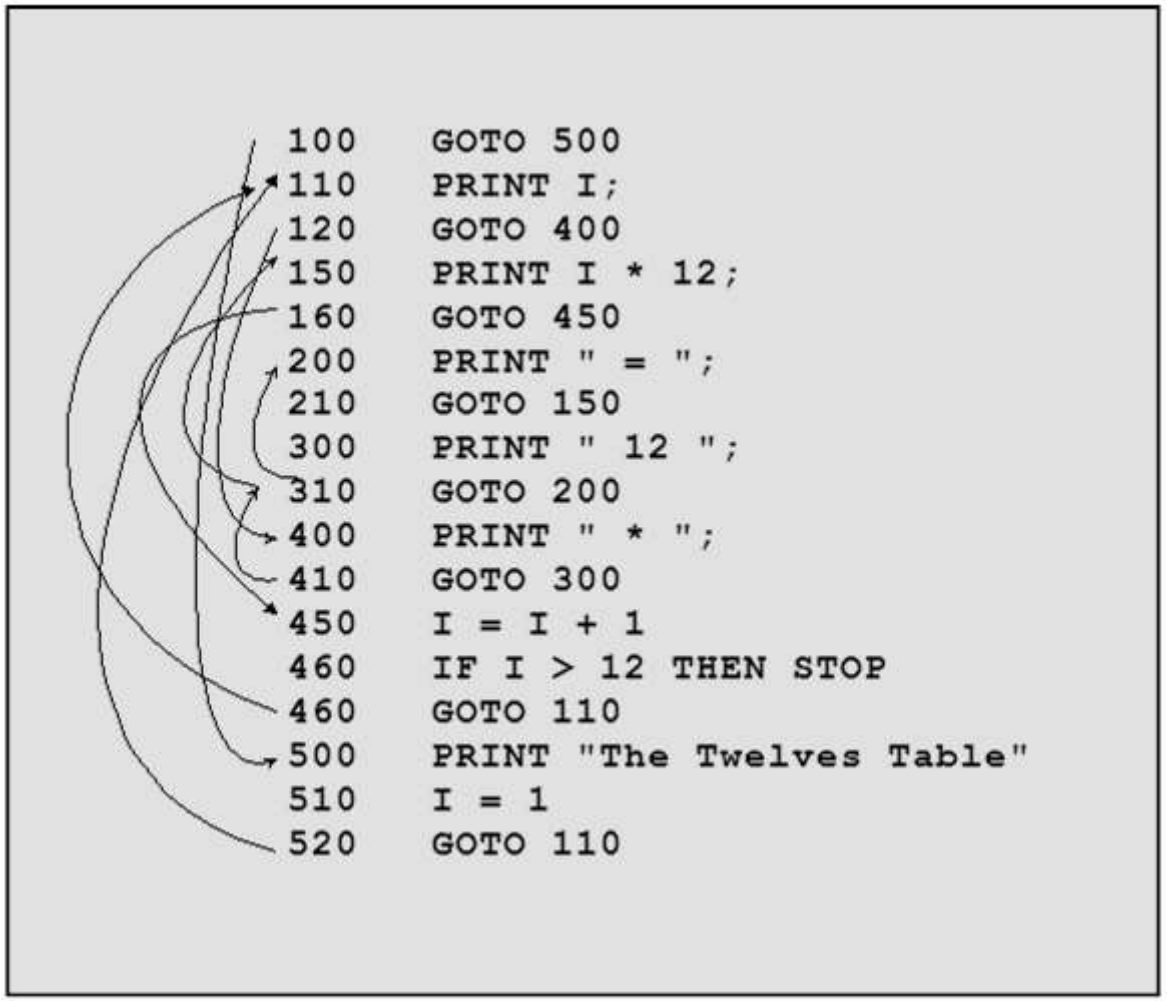
The Programming Style Evolution

Programming ↓	*sequencing of instructions for the computer*
Procedural Programming ↓	*functional decomposition: functions are building blocks, data is global.*
Modular Programming ↓	*data organized into modules for functions which operate on them*
Object-Based Programming ↓	*models of objects which encapsulate data and functions together: abstraction and info hiding*
Object-Oriented Programming	*modeling of objects also support of inheritance and polymorphism.*

Early Programming (1950s)

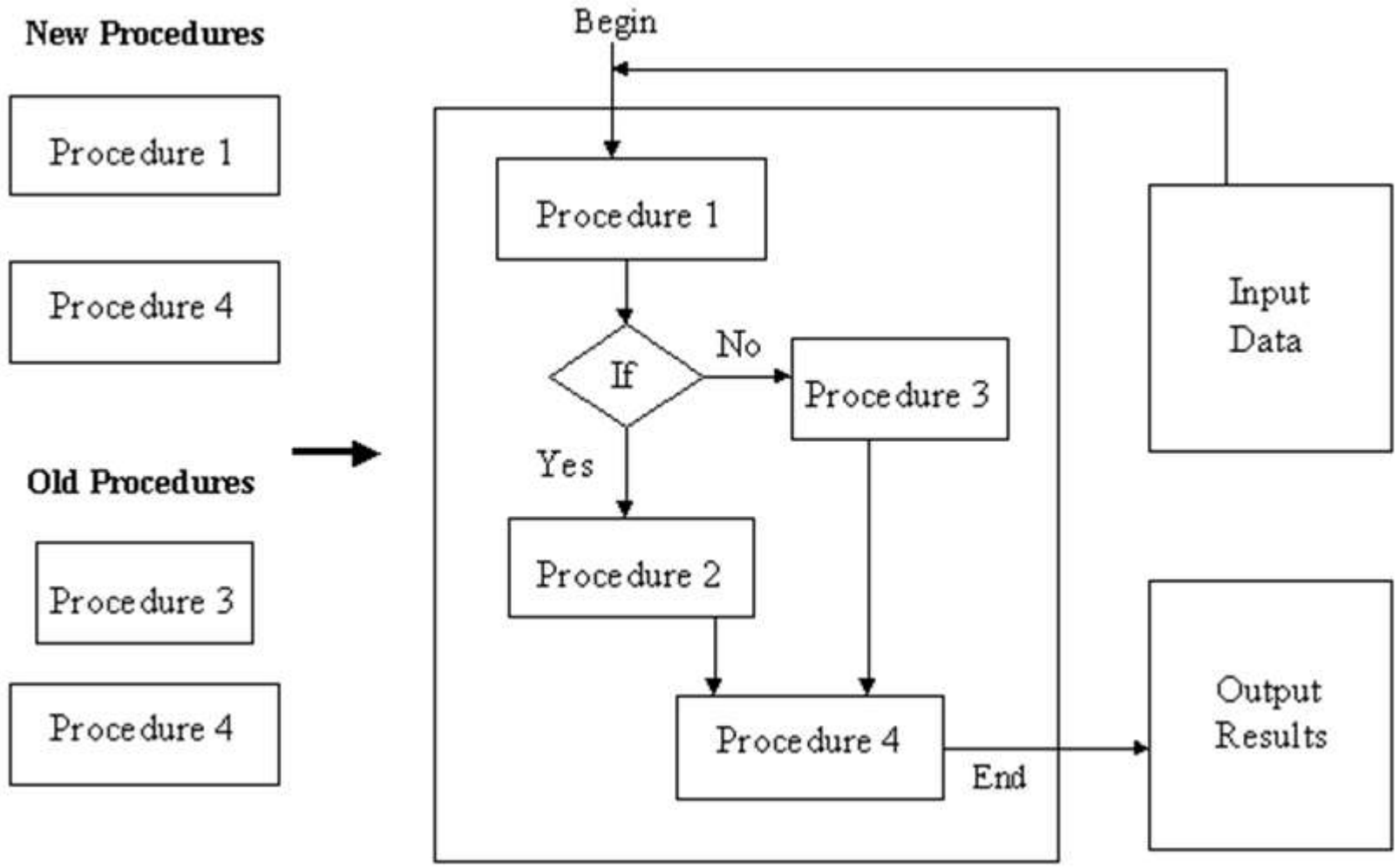
- Execute one statement after the other
- Uses GOTO to jump
- Single Entrance, Single Exit

- Subroutine (GOSUB)
  - Provided a natural division of labour
  - Could be reused in other programs
  - Elimination of Spaghetti-code



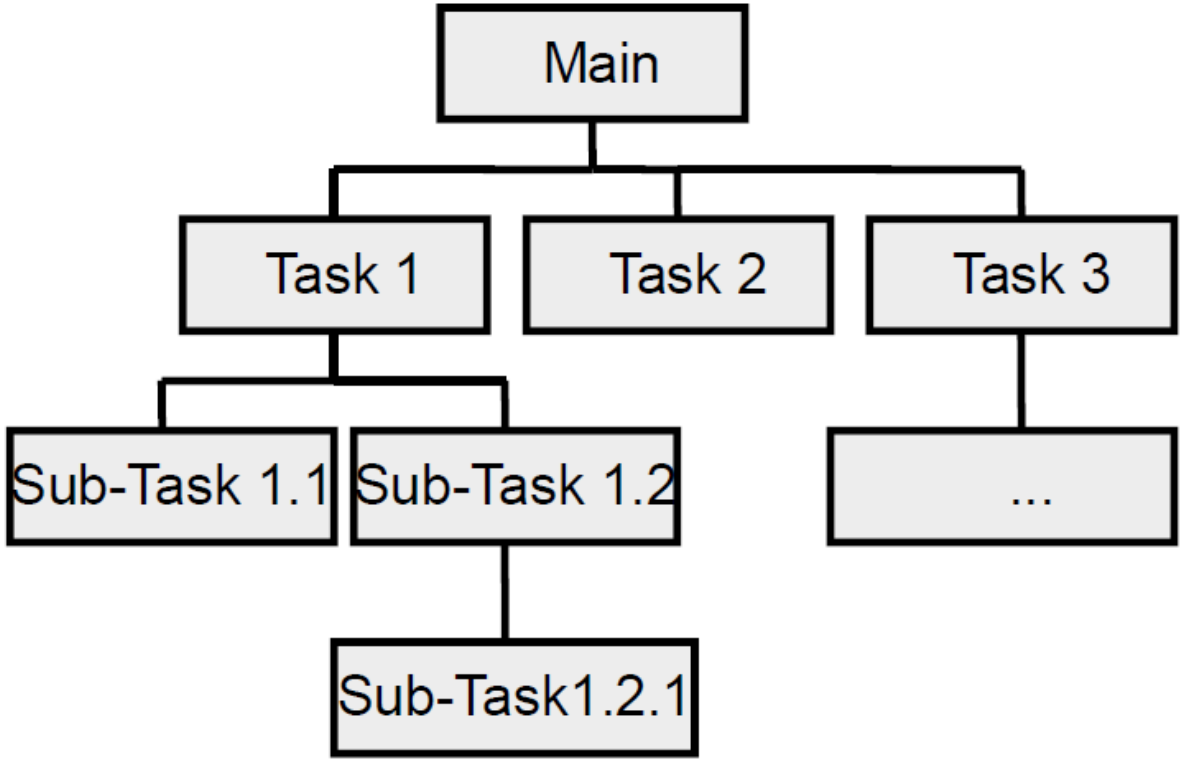
Procedure-Based Programming

- Defines the world as 'procedures' operating on 'data'
  - procedures have clearly defined interfaces
- Only 4 programming constructs
  - Sequence
  - Selection
  - Iteration
  - Recursion
- Modularization

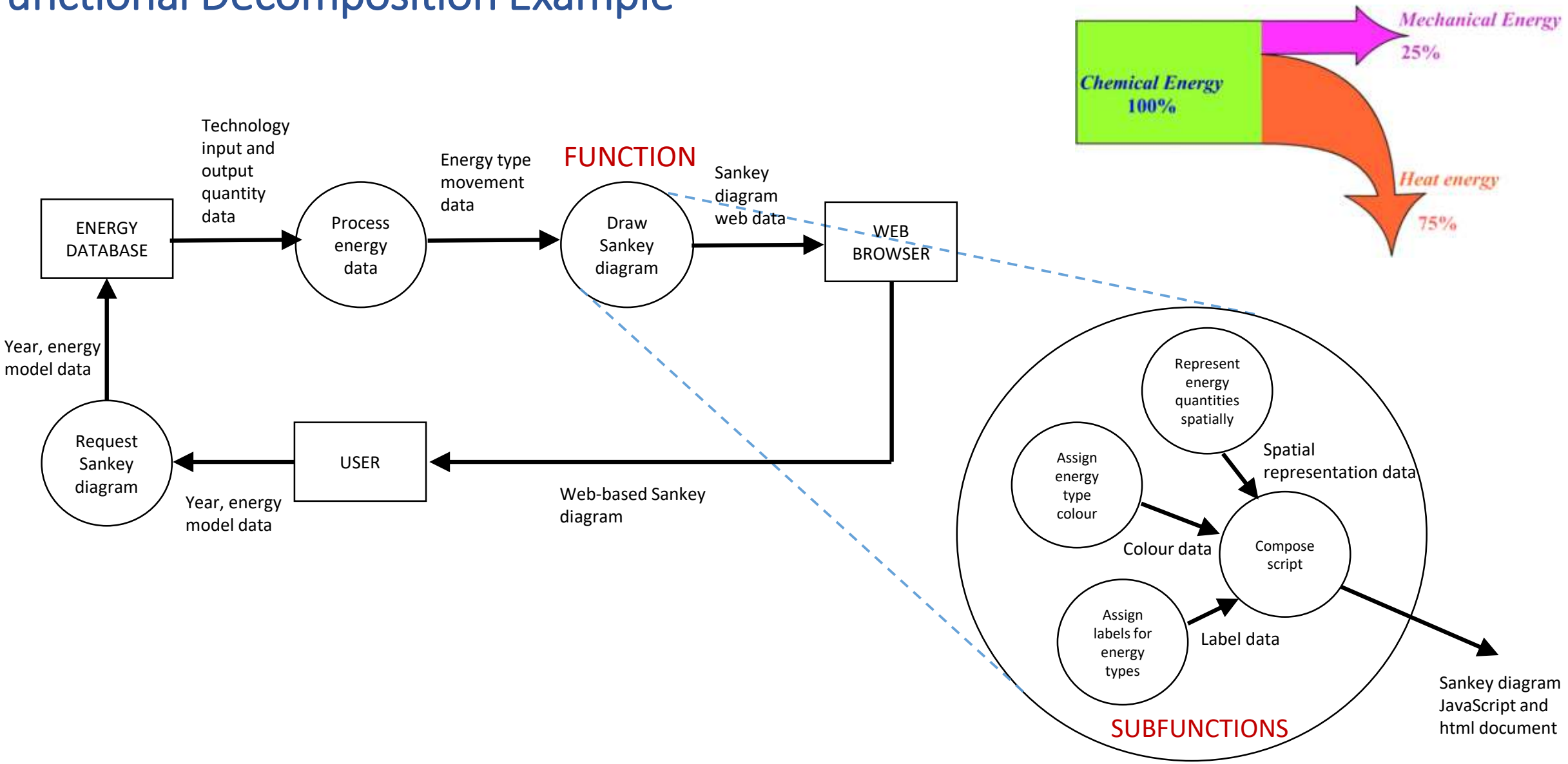


Structured Programming (1965)

- Divide and Conquer
  - Break large-scale problems into smaller components that are constructed independently
  - A program is a collection of procedures, each containing a sequence of instructions
- Functional Decomposition
  - Use of procedural hierarchy



Functional Decomposition Example

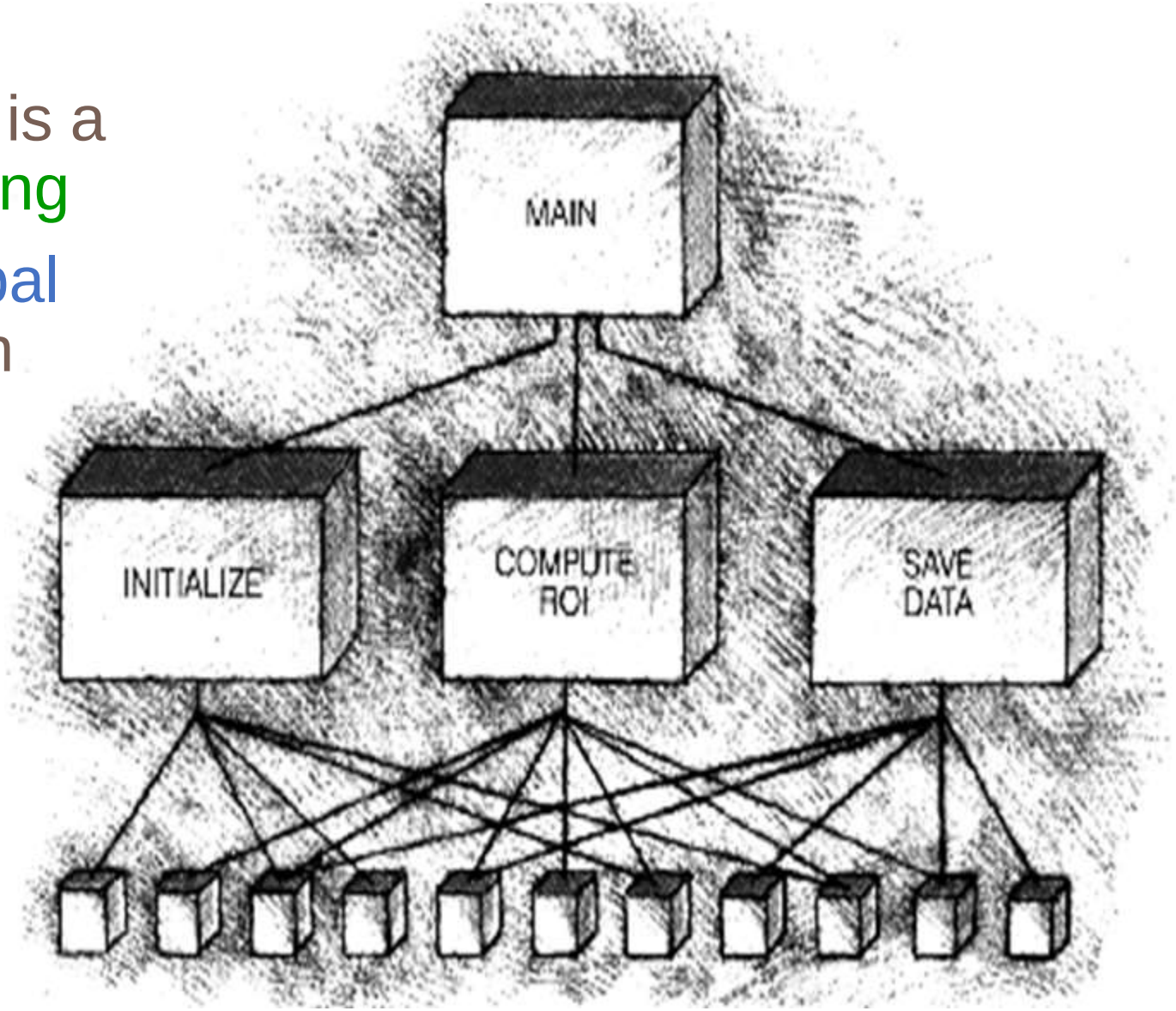


Structured Programming Problems

- Structured programming has a serious limitation:
  - It's rarely possible to anticipate the design of a completed system before it's implemented
  - The larger the system, the more restructuring takes place
- Software development had focused on the modularization of code
  - data moved around
    - argument/parameter associations
  - or data was global
    - works okay for tiny programs
    - Not so good when variables number in the hundreds
- Code reuse limited

Don’t use Global Variables

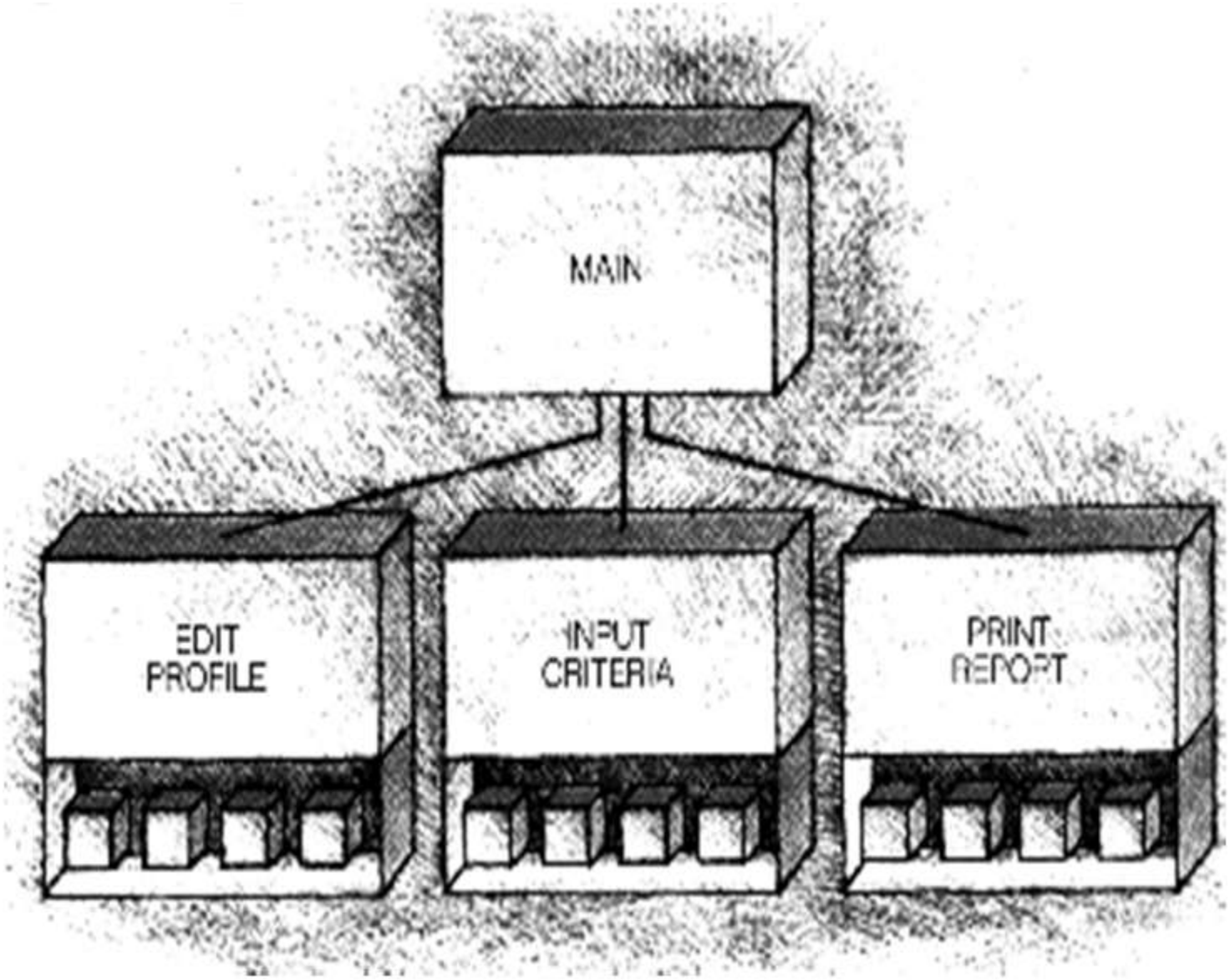
- Sharing data (global variables) is a violation of modular programming
- All modules can access all global variables without any restriction
  - No module can be developed and understood independently
- Global data are dangerous
  - This makes all modules dependent on one another



Credit: OOT A Managers' perspective, Dr Taylor

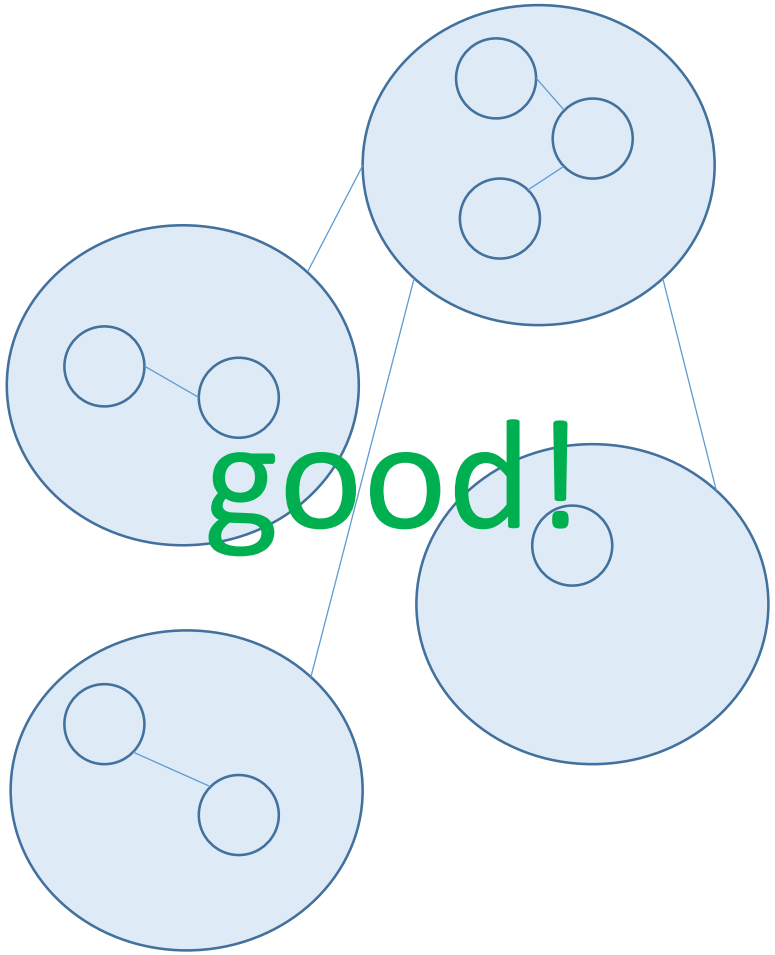
… instead Modularize Data

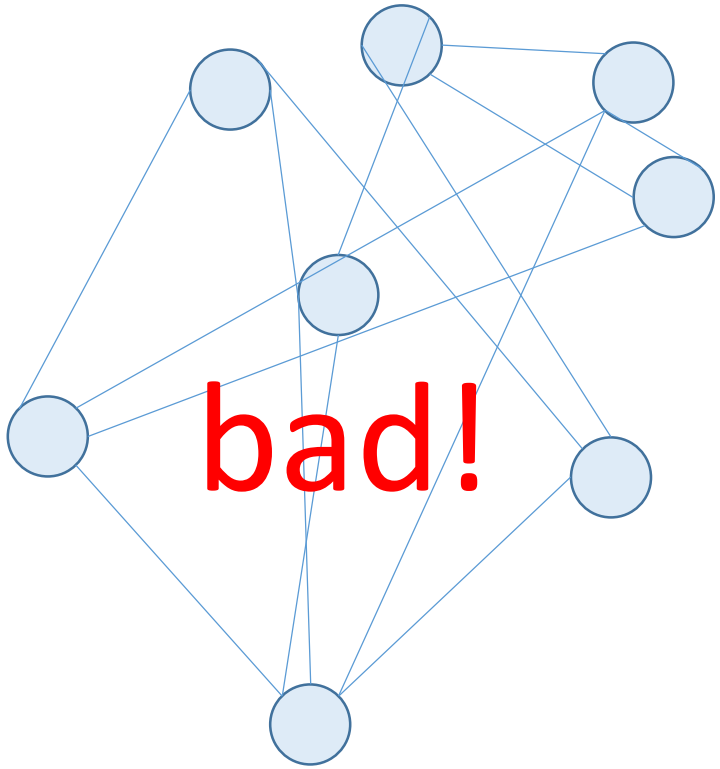
- Localize data inside the modules
  - This makes modules more independent of one another
  - Local Data



Credit: OOT A Managers' perspective, Dr Taylor

Modularisation: Coupling and Cohesion





An established general principle of programming is that **low coupling** and **high cohesion** are good. This is achieved by **modularisation.**

Modularisation: Information Hiding

- An improvement:
  - Give each procedure (module) it's own local data
  - This data can only be "touched" by that single subroutine
  - Subroutines can be designed, implemented, and maintained more easily
  - Other necessary data is passed amongst the procedures via argument/parameter associations

No Variables

```
main()
{
    DoTheFirstThing();
    DoTheSecondThing();
}
```

The main() function cannot see or change the variables a, or b

Local variable

```
void DoTheFirstThing()
{
    int a = 10;
}
```

The variable a is only visible inside the subprogram, DoTheFirstThing.

It can be changed and used here, but is invisible in all other functions

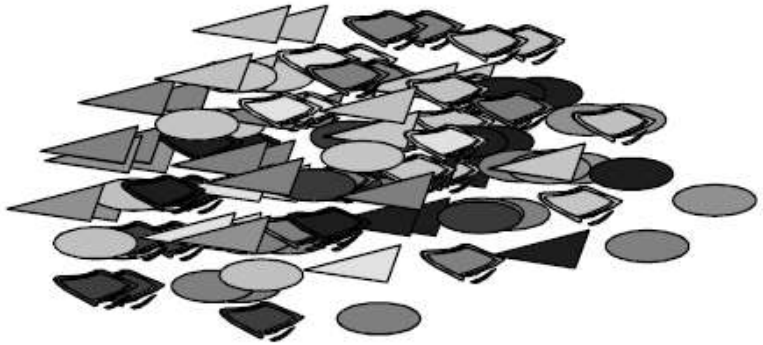
Local variable

```
void DoTheSecondThing()
{
    int b = 3;
}
```

The variable b is only visible inside the subprogram, DoTheNextThing.

It can be changed and used here, but is invisible in all other functions

The Evolution of Software Design Methods



1st Generation
Spaghetti-Code

2nd & 3rd Generation :
functional decomposition

Software =

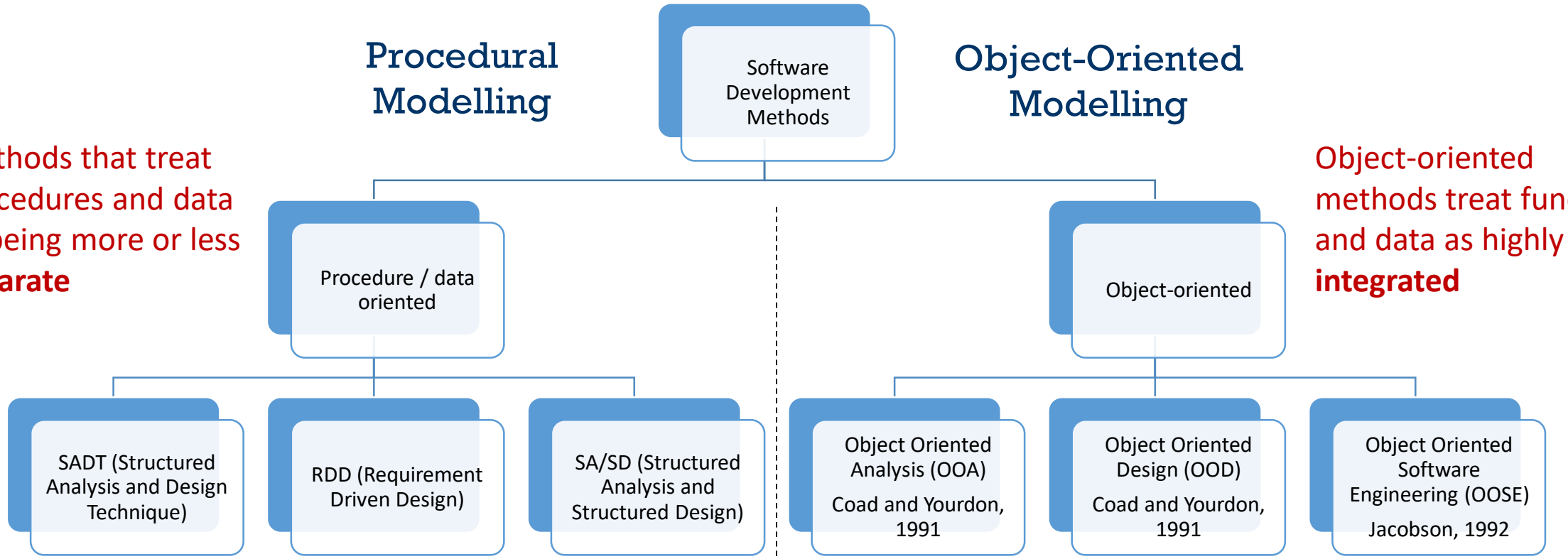
Data (Shapes)

+

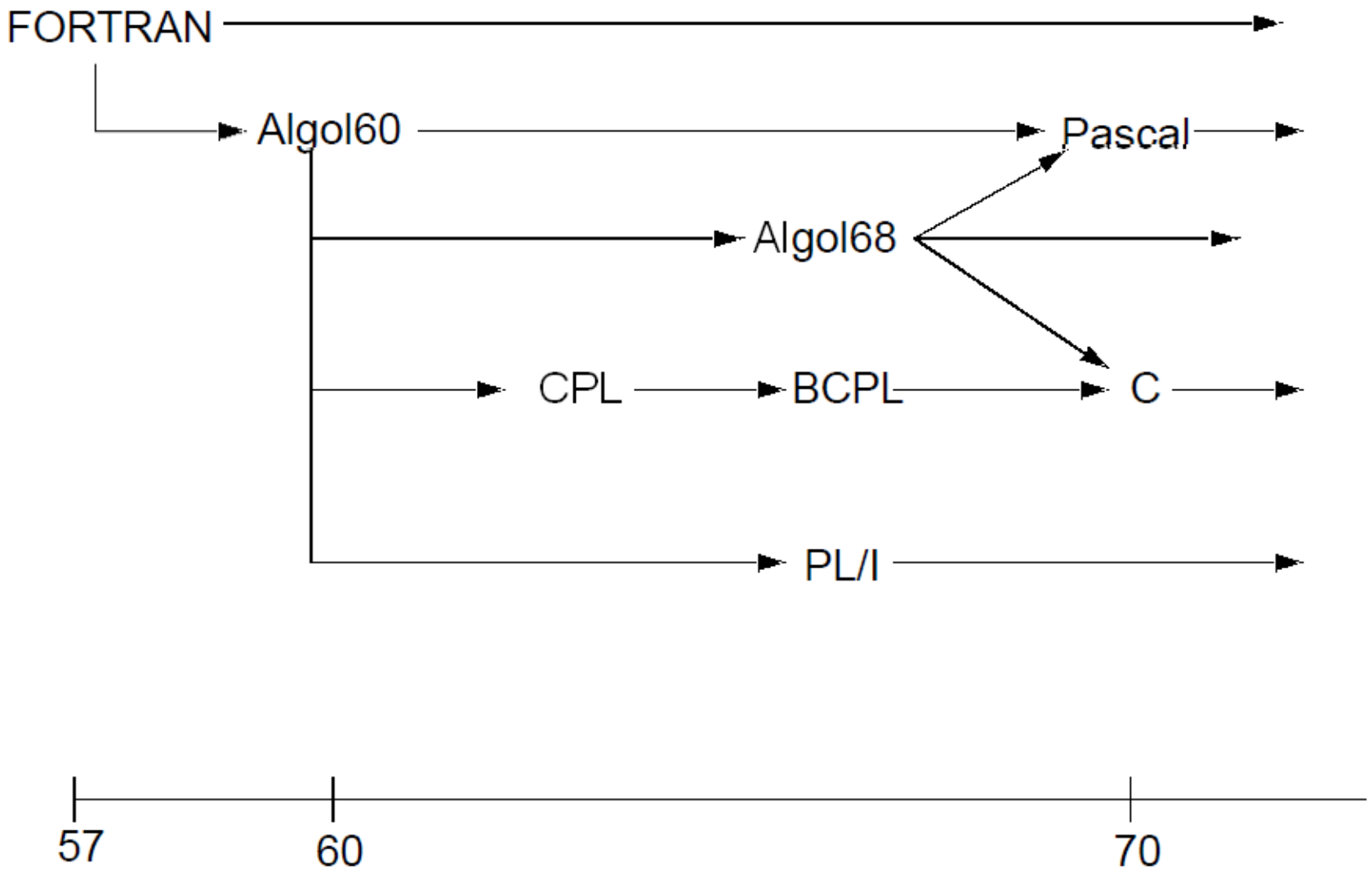
Functions (Colors)

4th Generation
object decomposition

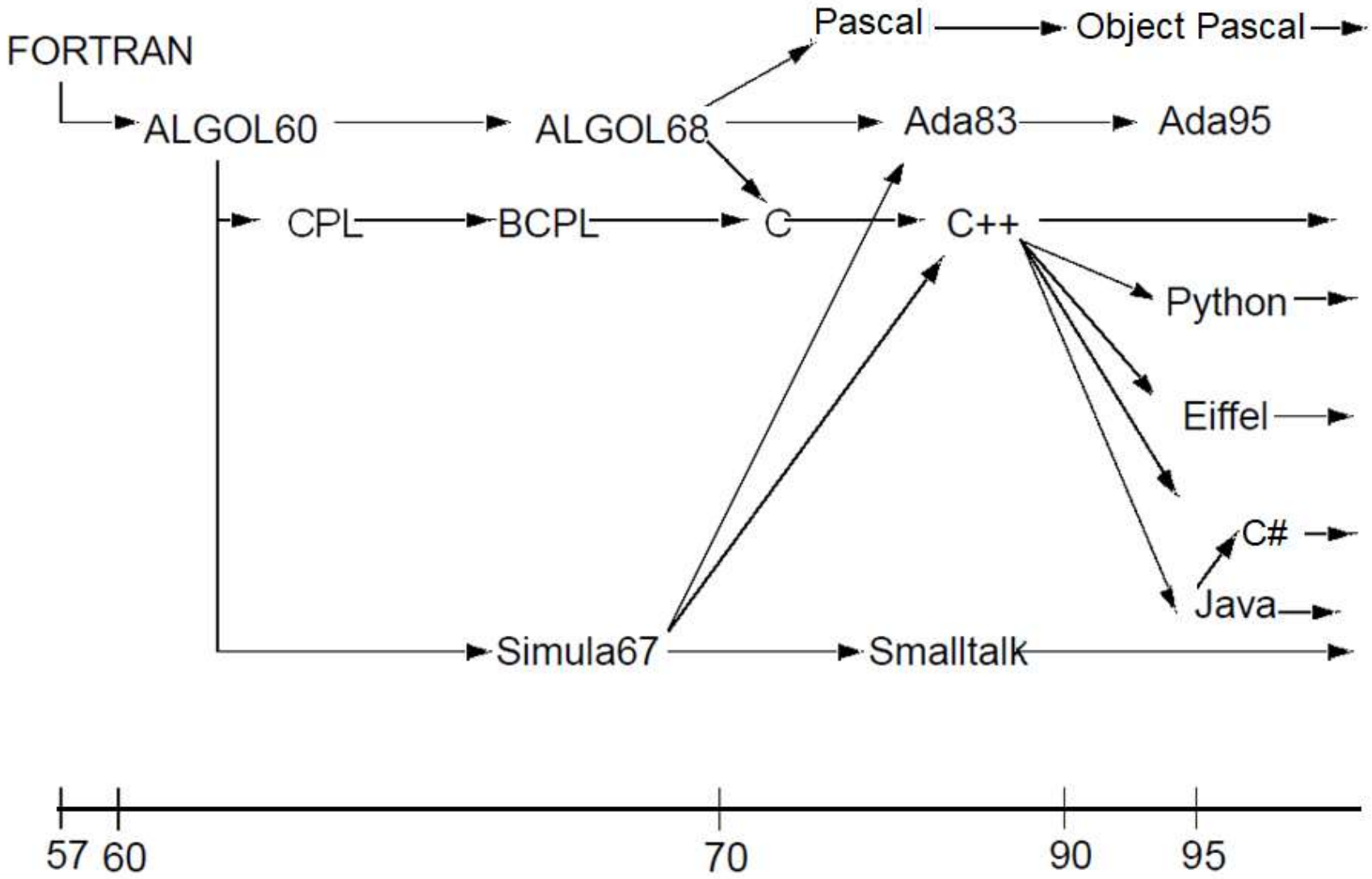
Software Development Methods



Procedural Programming: History



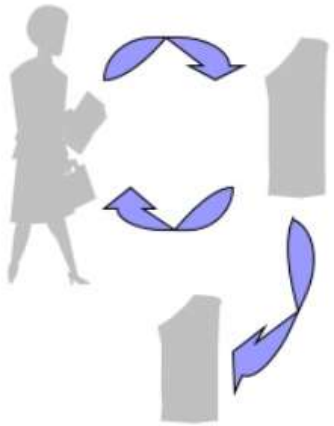
Object-Oriented Programming: History



Procedural vs Object-Oriented Modelling

- The purpose, as with all other **modelling** methods, is to understand the application in terms of the system's **functional requirements**
- **Procedure / data modelling** considers the system's data and behaviour separately
- **Object-oriented modelling** combines them and regards them as integrated objects that interact
- Object-oriented design consists of the following five steps:
  1. **Identifying the objects**
  2. **Organising the objects**
  3. **Describing how they interact**
  4. **Defining the operations on the objects**
  5. **Defining the objects internally**

Procedural



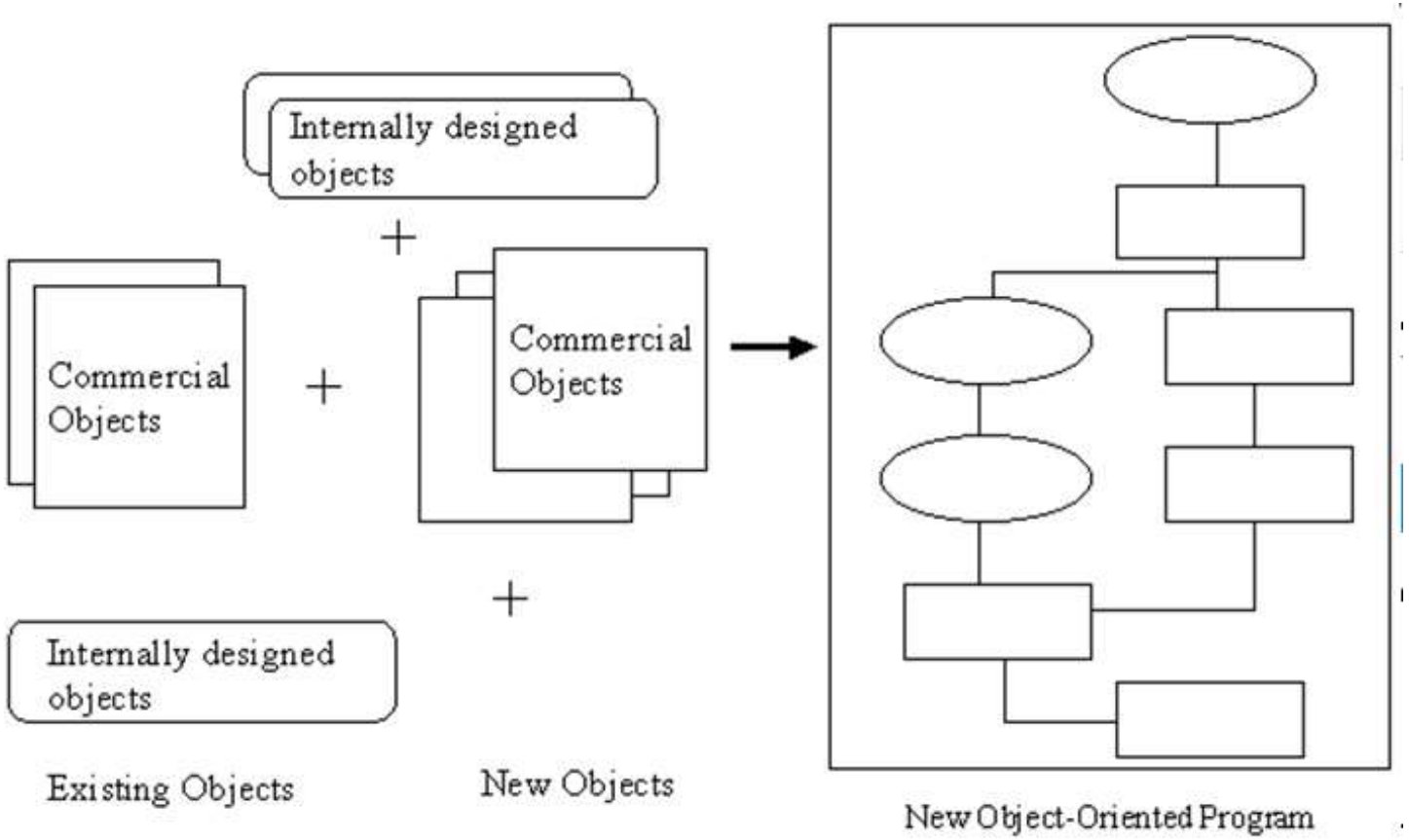
Withdraw, deposit, transfer

Object Oriented

Customer, money, account

Object-Oriented Programming Style

- Software Objects: software packet abstracting the salient behaviour and attributes of a real object into a software package that simulates the real object
- Well-constructed programs are tested components
  - Link data with procedures
  - If object function/interface is clearly defined, then object implementation may change at will
- OOP key concepts:
  - Object Classes
  - Encapsulation
  - Inheritance
  - Polymorphism

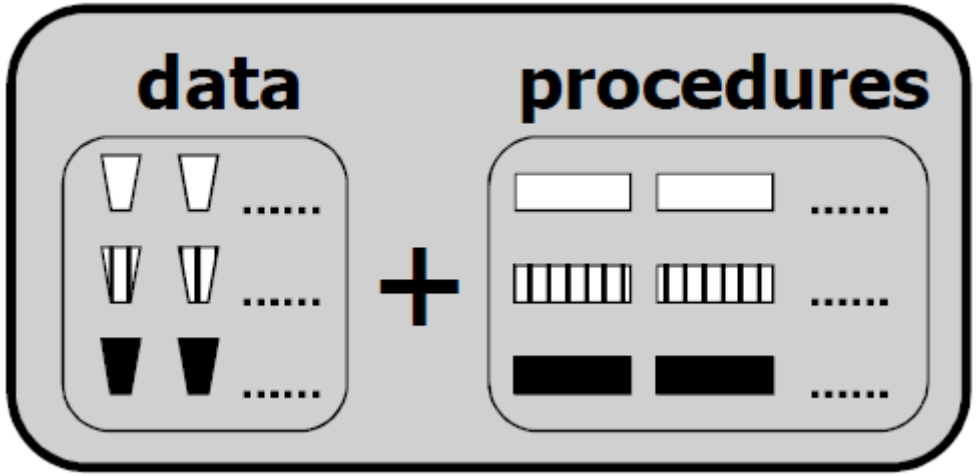


Object-Oriented vs Procedural Programming Style

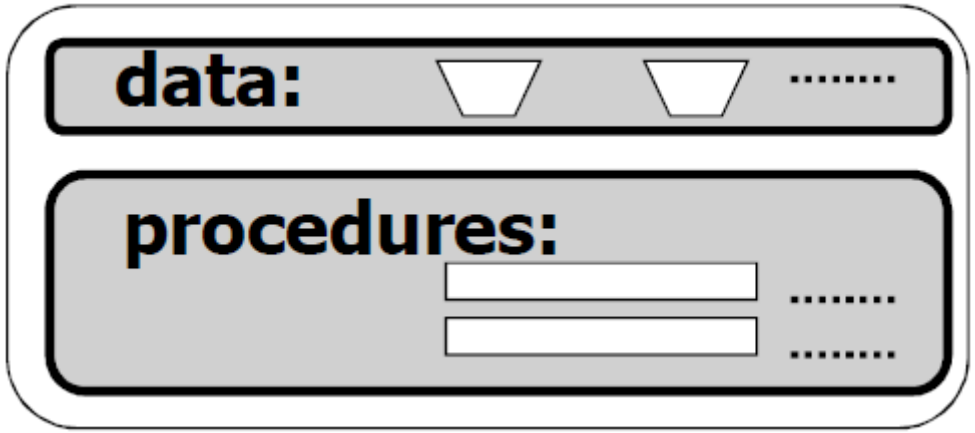
- Three Keys to Object-Oriented Technology
  - Objects
  - Messages (event-driven interaction)
  - Classes

- Translation for structured programmers
  - Variables
  - Function Calls
  - Data Types

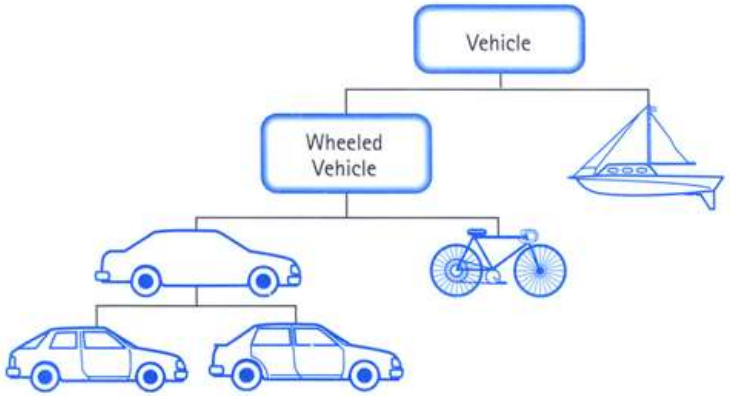
Procedural



Object Oriented



Principles of Object-Oriented Design (OOD)



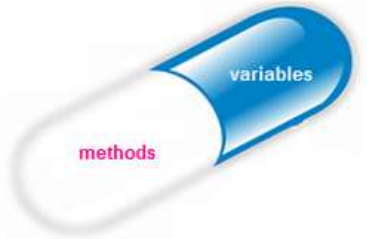
INHERITANCE

OO

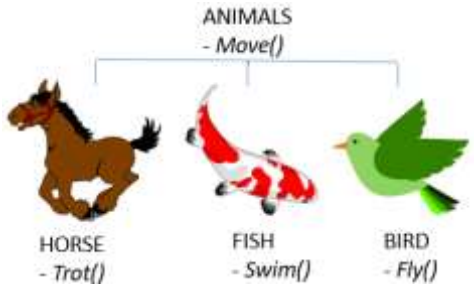
POLYMORPHISM

ENCAPSULATION

- When a sub-class **inherits** from another class it inherits all of its properties and methods



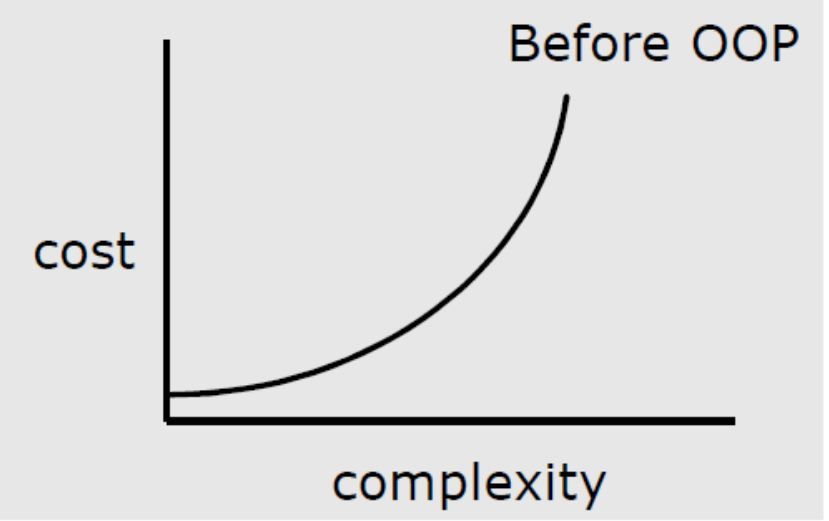
- With **polymorphism**, each subclass may have its own way of implementing the function.

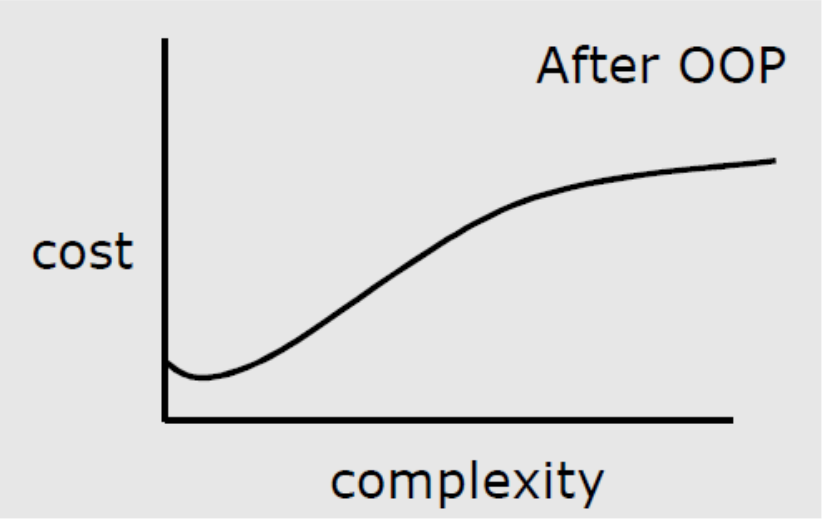


- Object-orientation binds together the data (attributes / properties) and the functions (methods) that manipulate it, within a class. This is known as **encapsulation**.

Benefits of OO-Programming in Software Development

- We should always strive to engineer our software to make it reliable and maintainable
  - Develop programs incrementally
  - Don't need to understand everything up front (including things you will never use)
  - Avoids spaghetti code
  - No need to start from scratch every time
- As the complexity of a program increases, its cost to develop and revise grows exponentially
  - OOP speeds development and revision time





Programming Languages Review

- Inheritance
- Encapsulation
- Polymorphism

- **Object oriented**
  - C#
  - C++
  - Java
  - Python
  - Obj Pascal/Delphi
  - Smalltalk
  - Simula
  - Ruby
  - Ada
- **Procedural**
  - Fortran
  - Cobol
  - Algol
  - Pascal
  - C
  - BASIC

Relies on predefined and well-organized procedures, functions or sub-routines

- **Object based**
  - Visual Basic
- **Predicate Logic**
  - Lisp
  - Prolog
- **Scripting**
  - Perl
  - PHP
  - JavaScript
  - ASP
- **Functional**
  - Erlang
  - Python
  - Haskell

Uses encapsulation to support objects, but need not support inheritance or polymorphism

Designed for integrating and communicating with other programming languages. Associated with creation of dynamic Web pages

Treats computation as the evaluation of mathematical functions and avoids changing-state of an object after creation. Often uses recursion.

TIOBE Ranking…

Be it Machine Learning, Data Analytics, Data Processing, Web Development, Enterprise Software Development or taking the photo of Blackhole: **Python is everywhere**.

Also, popular programming language ranking site TIOBE ranked **Python** as the third most popular general programming language behind **Java** and **C**.

As shown by the TIOBE index, **Java** is still the most dominant enterprise programming language and will remain so.

Java's runtime, **JVM** is one of the best pieces of software engineering and offers a solid foundation for Java.

Top 10 In-Demand programming languages to learn in 2020 | by Md Kamaruzzaman | Towards Data Science

The TIOBE Programming Community index is an indicator of the popularity of programming languages.
index | TIOBE - The Software Quality Company

Dec 2020	Dec 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	16.48%	+0.40%
2	1	↓	Java	12.53%	-4.72%
3	3		Python	12.21%	+1.90%
4	4		C++	6.91%	+0.71%
5	5		C#	4.20%	-0.60%
6	6		Visual Basic	3.92%	-0.83%
7	7		JavaScript	2.35%	+0.26%
8	8		PHP	2.12%	+0.07%
9	16	⇈	R	1.60%	+0.60%
10	9	↓	SQL	1.53%	-0.31%
11	22	⇈	Groovy	1.53%	+0.69%
12	14	↑	Assembly language	1.35%	+0.28%
13	10	↓	Swift	1.22%	-0.27%
14	20	⇈	Perl	1.20%	+0.30%
15	11	⇊	Ruby	1.16%	-0.15%
16	15	↓	Go	1.14%	+0.15%
17	17		MATLAB	1.10%	+0.12%
18	12	⇊	Delphi/Object Pascal	0.87%	-0.41%
19	13	⇊	Objective-C	0.81%	-0.39%
20	24	⇈	PL/SQL	0.78%	+0.04%

Thank You!