introducción a los lenguajes de programación y sus paradigmas

LENGUAJES Y PARADIGMAS

Rafael Aguilar

ESCOM

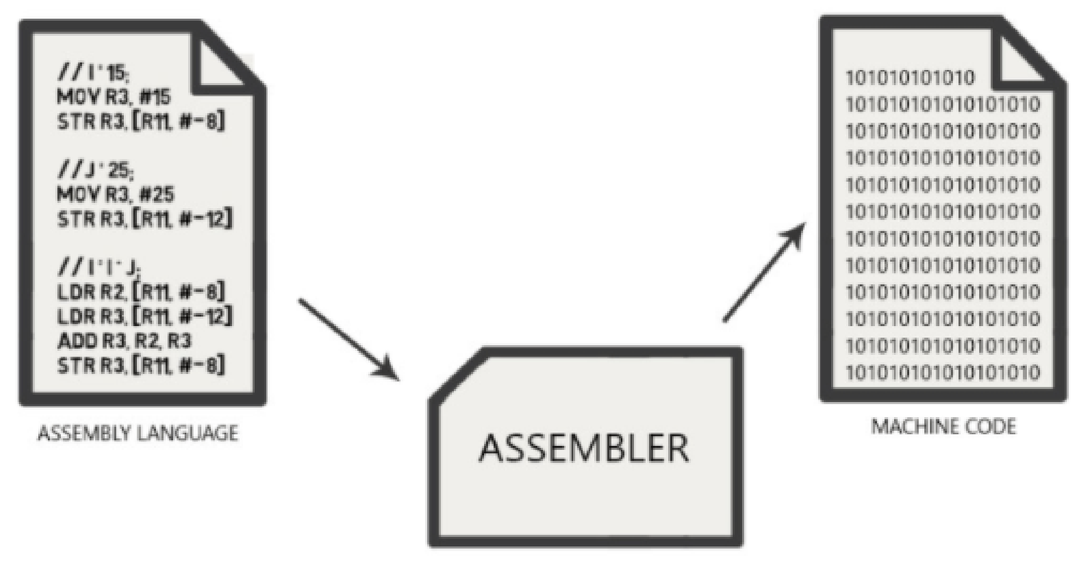
LENGUAJES

DEFINICIÓN

- Es “una notación para comunicar a una computadora lo que queremos que haga” [Kenneth].
- Los lenguajes de programación facilitan la expresión y comunicación de ideas entre personas. Sin embargo, los lenguajes de programación se diferencian de los lenguajes naturales en dos aspectos importantes.
  - Los lenguajes de programación también permiten la comunicación de ideas entre personas y máquinas informáticas.
  - Los lenguajes de programación tienen un dominio expresivo más pequeño que nuestros lenguajes naturales. Es decir, facilitan sólo la comunicación de ideas computacionales. [Tucker]

BREVE HISTORIA

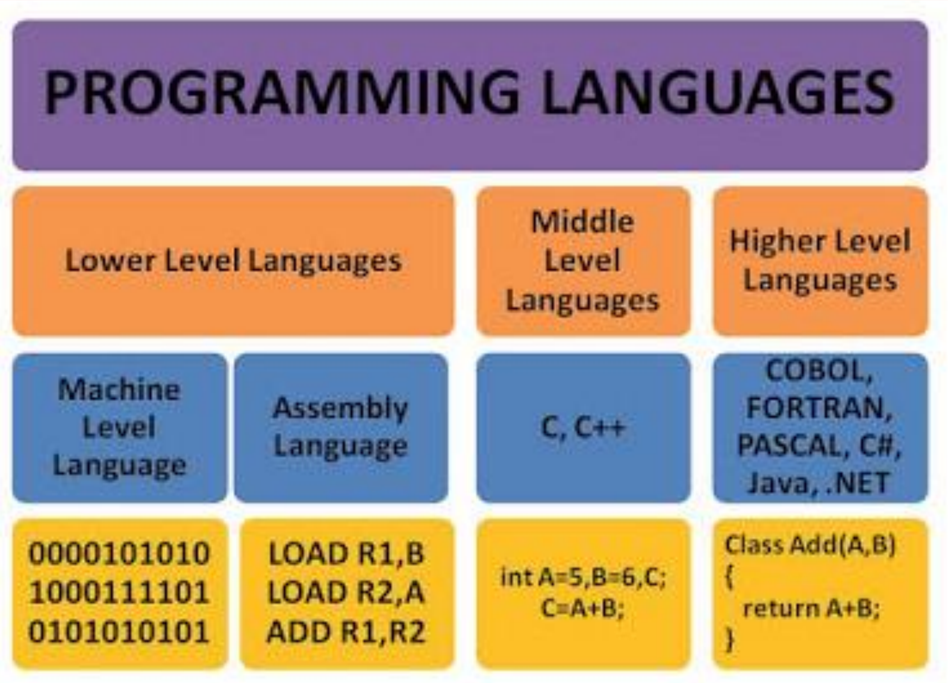
- **Los primeros lenguajes de programación fueron los lenguajes de máquina y los lenguajes ensamblador, a comienzos de la década de 1940.**

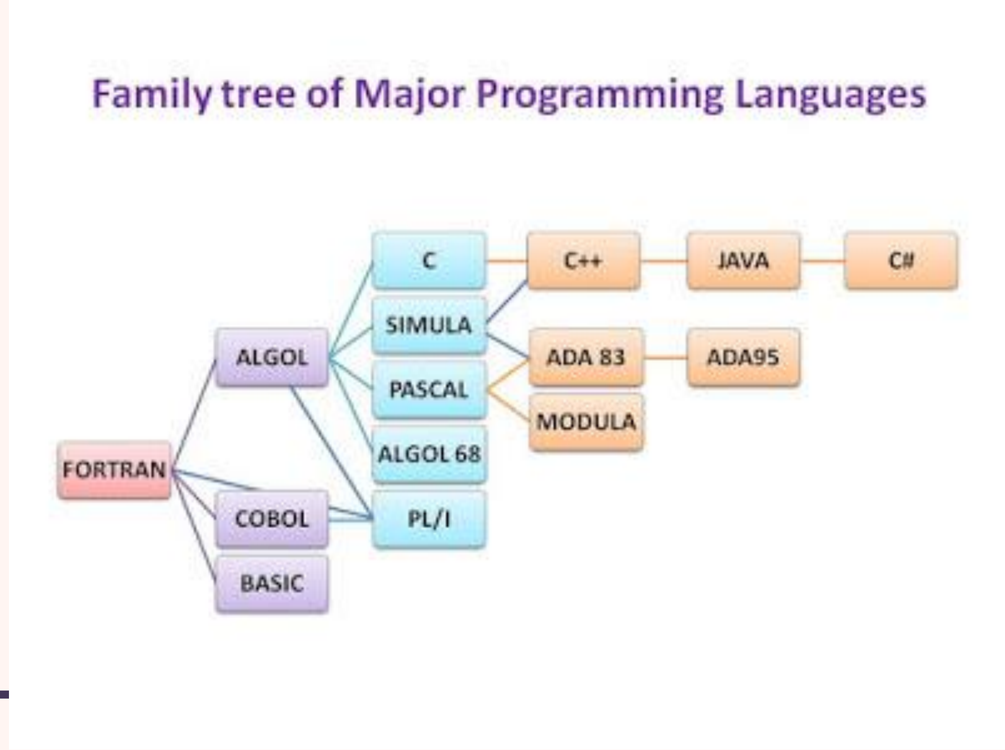


BREVE HISTORIA

- La década de 1950 marcó el inicio de la era de los “Lenguajes de Orden Superior” (HOL, “Higher-order languages”). Los lenguajes HOL se distinguen de sus predecesores, los lenguajes de máquina y ensamblador, en que su estilo de programación es independiente de cualquier arquitectura de máquina en particular.

- Los primeros lenguajes HOL fueron Fortran, Cobol, Algol y Lis.





DOMINIOS EN LENGUAJES DE PROGRAMACIÓN

DOMINIOS

- Quizás el mayor motivador para el desarrollo de lenguajes de programación en las últimas décadas es la demanda en rápida evolución de potencia informática y nuevas aplicaciones por parte de comunidades de usuarios grandes y diversas.

- Las siguientes comunidades de usuarios pueden reclamar una participación importante en el panorama de los lenguajes de programación:

  - Inteligencia artificial
  - Educación
  - Ciencia e ingeniería
  - Sistemas de información
  - Sistemas y Redes
  - World Wide Web

INTELIGENCIA ARTIFICIAL

- La comunidad de programadores de Inteligencia Artificial (IA) ha estado activa desde la década de 1960.

- La IA modela: el comportamiento inteligente humano, la deducción lógica, y la cognición; La manipulación de símbolos, las expresiones funcionales y el diseño de sistemas de prueba lógica han sido objetivos centrales.

- Los paradigmas de la programación funcional y la programación lógica han evolucionado en gran medida gracias a los esfuerzos de los programadores de inteligencia artificial.

- Los lenguajes de programación funcional prominentes a lo largo de los años incluyen Lisp, Scheme, ML y Haskell. Los lenguajes de programación lógica prominentes incluyen Prolog y CLP.

INTELIGENCIA ARTIFICIAL

- **En el ámbito de la programación lógica, sólo un lenguaje, Prolog, ha sido el actor principal, y Prolog ha tenido poca influencia en el diseño de lenguajes en otras áreas de aplicación.**

EDUCACIÓN

- En las décadas de 1960 y 1970, se diseñaron varios lenguajes clave con el objetivo principal de enseñar a los estudiantes sobre programación:

- Basic fue diseñado en la década de 1960 por John Kemeny para facilitar el aprendizaje de la programación a través del tiempo compartido.

- Pascal, un derivado de Algol, fue diseñado en la década de 1970 con el propósito de enseñar programación. Pascal sirvió durante varios años como el idioma principal de enseñanza en los planes de estudio de ciencias de la computación de nivel universitario.

- C, C ++ y Java, son lenguajes de “fuerza industrial”. Estos lenguajes proporcionan a los graduados una herramienta de programación que pueden usar inmediatamente cuando ingresan a la profesión de la informática. Sin embargo, son lenguajes inherentemente más complejos y difíciles de aprender (que Pascal o Basic).

EDUCACIÓN

- Con Python los cursos de introducción a la informática pueden volver a la simplicidad y concentrarse nuevamente en la enseñanza de los primeros básicos de programación. Por ejemplo, Python tiene una sintaxis y una semántica más transparentes, lo que lo hace más fácil de dominar por un novato que cualquiera de sus alternativas de “fuerza industrial”.

CIENCIA E INGENIERÍA

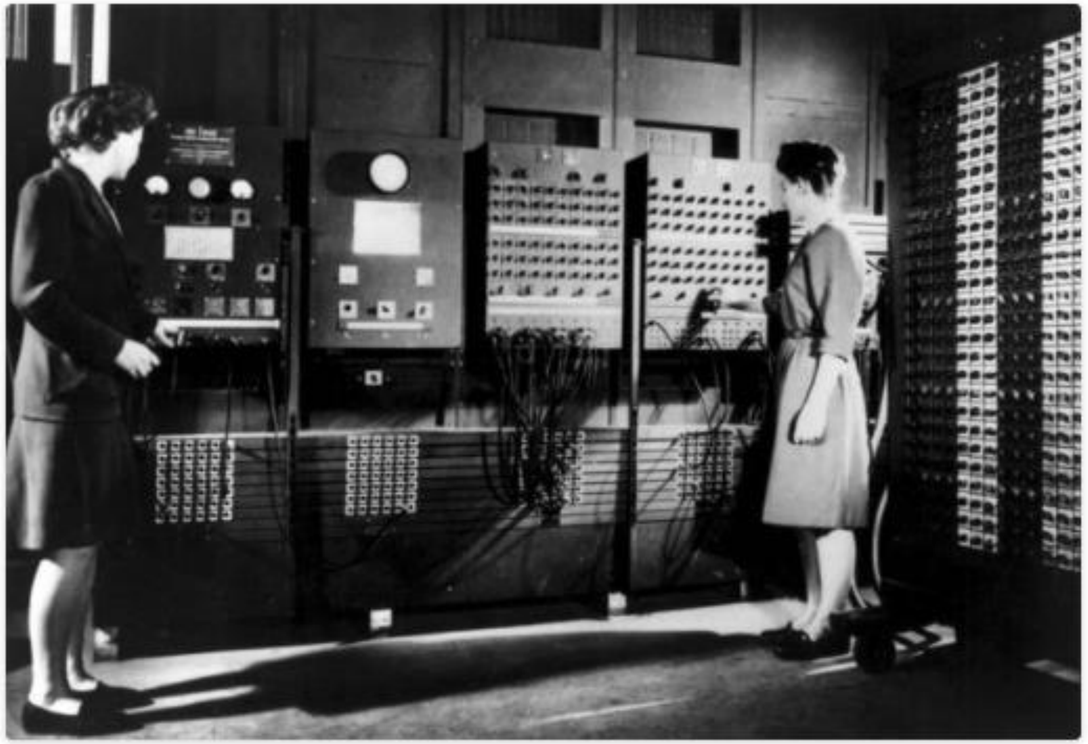
- La comunidad de programación científica y de ingeniería jugó un papel importante en la historia temprana de la computación y sigue desempeñando un papel importante en la actualidad.

- Los primeros programas se escribieron en la década de 1940 para predecir las trayectorias de balística durante la Segunda Guerra Mundial, utilizando las fórmulas físicas conocidas que caracterizan a los cuerpos en movimiento.

- Estos programas fueron escritos por primera vez en lenguaje de máquina y ensamblador por matemáticos especialmente entrenados.

CIENCIA E INGENIERÍA

(Fig. 7) Vannevar Bush (1890–1974) with his differential analyzer Bush joined MIT at age 29 as an electrical engineering professor and led the design of the differential analyzer. During World War II, he chaired the National Defense Research Committee and advised President Franklin D. Roosevelt on scientific matters. (Source: Computer History Museum)



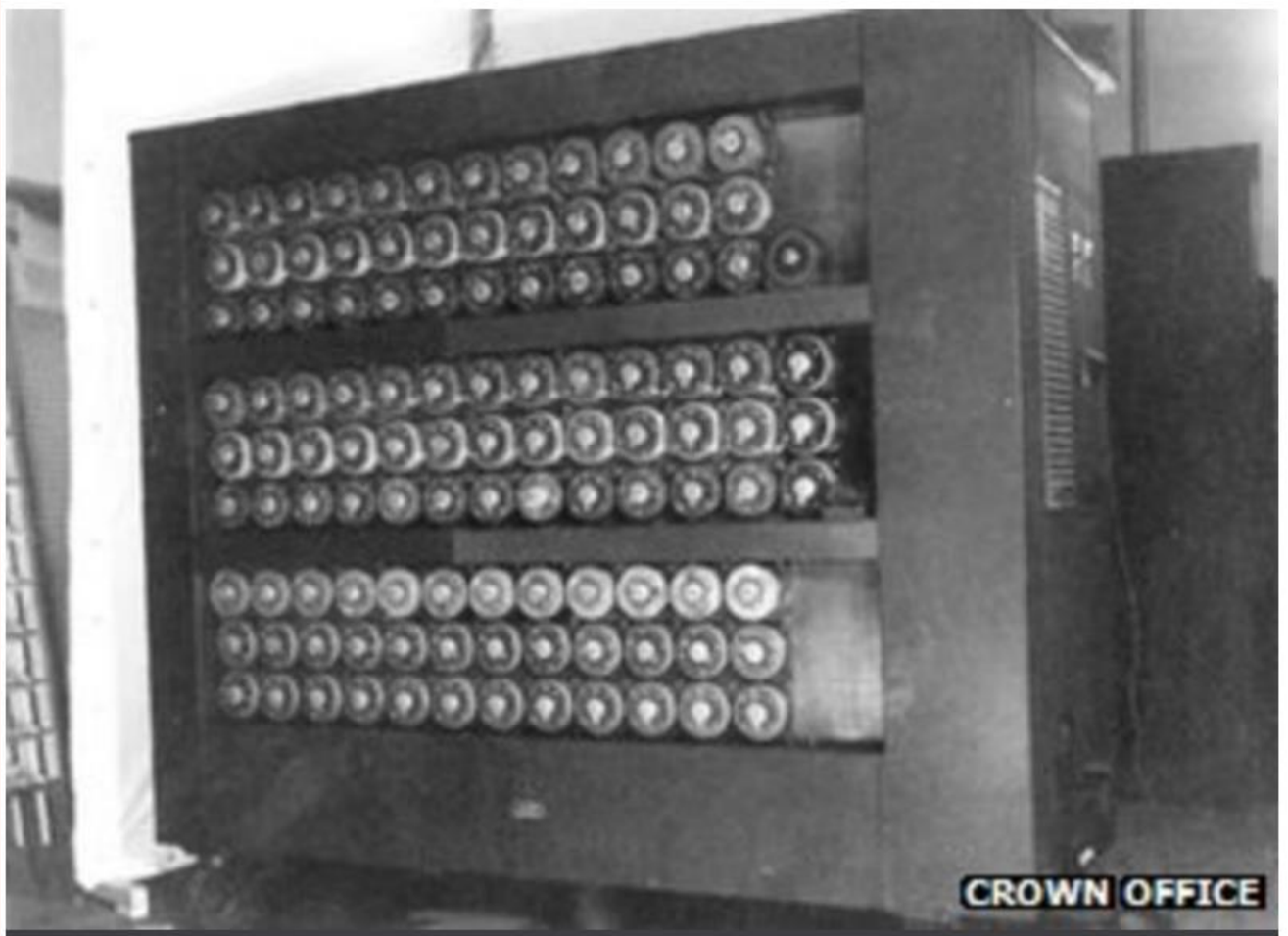
(Fig. 8) 1946, ENIAC programmers Frances Bilas (later Frances Spence) and Betty Jean Jennings (later Jean Bartik) stand at its main control panels. Both held degrees in mathematics. Bilas operated the Moore School's Differential Analyzer before joining the ENIAC project. (Source: Computer History Museum).

CIENCIA E INGENIERÍA

PHOTO DETAILS / DOWNLOAD HI-RES 4 of 11

German forces depended on Enigma machines to encode and decode secret messages transmitted over the radio during World War II. The Enigma machine is on the left. (Photo courtesy of Helge Fykse, Norway)

CIENCIA E INGENIERÍA



Turing helped adapt a device originally developed by Poland to create the bombe

CIENCIA E INGENIERÍA

- Una fuerza impulsora importante detrás de las aplicaciones científicas y de ingeniería a lo largo de su historia es la necesidad de obtener la mayor potencia de procesamiento posible.

- La potencia de procesamiento de las supercomputadoras de hoy se mide en teraflops (billones de operaciones de punto flotante por segundo), y el líder actual corre a una velocidad de 280 teraflops bajo el punto de referencia de rendimiento estándar llamado LINPAK (ver www.topSOO.org).

- Muchas de las aplicaciones científicas y de ingeniería actuales son modelos de sistemas naturales complejos en campos como la bioinformática y las ciencias de la tierra y la atmósfera.

CIENCIA Y TECNOLOGÍA

- El primer lenguaje de programación científica, Fortran I, fue diseñado por John Backus en IBM en 1954. El acrónimo "Fortran" es una abreviatura de "Traductor de fórmulas". Fortran es probablemente el lenguaje de programación científica más utilizado en la actualidad.

- Las primeras versiones de Fortran tenían muchos problemas: el mismo programa Fortran se ejecutaba de forma diferente en diferentes máquinas, o de lo contrario no se ejecutaba en absoluto. Una de las soluciones vino con el lenguaje Algol, abreviatura de "Algorithmic Language", que fue diseñado por un comité internacional en 1959. El principal objetivo de diseño de Algol era proporcionar un lenguaje mejor definido que Fortran tanto para la computación como para la publicación de textos científicos, y algoritmos matemáticos.

CIENCIA Y TECNOLOGÍA

- Al día de hoy, la informática científica sigue siendo una actividad central en la historia de la programación y los lenguajes de programación. Su dominio de problemas se ocupa principalmente de realizar cálculos complejos de forma muy rápida y precisa.

- Los cálculos se definen mediante modelos matemáticos que representan fenómenos científicos. Se implementan principalmente utilizando el paradigma de programación imperativa.

- Los lenguajes de programación modernos que se utilizan ampliamente en el campo de la programación científica incluyen Fortran 90, C, C ++ y High Performance Fortran.

CIENCIA Y TECNOLOGÍA

- Un pronóstico realizado en 1973 por el profesor Jay Forester del MIT, usando sistemas dinámicos:
- Artículo de RT: ”La vida tal y como la conocemos" acabará en 2040: Desentierran siniestra predicción de científicos”
- URL: https://actualidad.rt.com/actualidad/285424-vida-acabar-2040-canal-australia-mit-fin-mundo
- https://link.springer.com/article/10.1007/s41247-016-0014-8

SISTEMAS DE INFORMACIÓN

- **Los programas diseñados para que las instituciones los utilicen para gestionar sus sistemas de información son probablemente los más prolíficos del mundo.**

- **Las corporaciones se dieron cuenta en la década de 1950 de que el uso de computadoras podía reducir en gran medida el tedio de mantener registros y mejorar la precisión y confiabilidad de lo que podían procesar.**

- **Los sistemas de información que se encuentran en las corporaciones incluyen el sistema de nómina, el sistema de contabilidad, los sistemas de ventas y marketing en línea, los sistemas de inventario y fabricación, etc.; estos sistemas se caracterizan por procesar grandes cantidades de datos.**

SISTEMAS DE INFORMACIÓN

- **Tradicionalmente, los sistemas de información se han desarrollado en lenguajes de programación como Cobol y SQL.**

- **Cobol (*Common Business Oriented Language*) fue diseñado por primera vez a fines de la década de 1950 por un grupo de representantes de la industria que deseaban desarrollar un lenguaje que fuera portátil a través de una variedad de arquitecturas de máquinas diferentes; utiliza el inglés como base para su sintaxis y apoya un estilo de programación imperativo.**

SISTEMAS DE INFORMACIÓN

- Por el contrario, SQL surgió en la década de 1980 como una herramienta de programación declarativa para la especificación de bases de datos, la generación de informes y la recuperación de información.

- SQL significa “Lenguaje de consulta estructurado" y es el lenguaje predominante que se utiliza para especificar y recuperar información de bases de datos relacionales. El modelo de base de datos relacional se usa ampliamente, en parte debido a sus sólidos fundamentos matemáticos en el álgebra relacional.

SISTEMAS DE INFORMACIÓN

- Más recientemente, las empresas han desarrollado una amplia gama de aplicaciones de comercio electrónico. Estas aplicaciones a menudo utilizan un modelo de "cliente-servidor" para el diseño de programas, donde el programa interactúa con los usuarios en sitios remotos y proporciona acceso simultáneo a una base de datos compartida.

- La programación impulsada por eventos es esencial en estas aplicaciones, y los programadores combinan lenguajes como Java, Perl, Python y SQL para implementarlos.

PROGRAMACIÓN DE SISTEMAS Y APLICACIONES EN REDES

- **Los programadores de sistemas diseñan y mantienen el software básico que ejecutan los sistemas: componentes del sistema operativo, software de red, compiladores y depuradores de lenguajes de programación, máquinas virtuales e intérpretes, y sistemas integrados y en tiempo real (en teléfonos móviles, cajeros automáticos, aviones, etc.). Estos tipos de software están estrechamente relacionados con las arquitecturas de máquinas específicas, como Intel/AMD x86 y Apple/Motorola/IBM PowerPC.**

PROGRAMACIÓN DE SISTEMAS Y APLICACIONES EN REDES

- La mayoría de estos programas están escritos en C, lo que permite a los programadores acercarse mucho al nivel del lenguaje máquina. La programación de sistemas se realiza típicamente utilizando el paradigma imperativo. Sin embargo, los programadores de sistemas también deben lidiar con la ejecución de programas concurrentes y controlados por eventos, y también tienen que abordar la corrección de errores en el diseño de programas.

PROGRAMACIÓN DE SISTEMAS Y APLICACIONES EN REDES

- El ejemplo, por antonomasia, de un lenguaje de programación de sistemas es C.
- C fue diseñado en 1972 por Dennis Ritchie en Bell Labs. Fue usado en la codificación del sistema operativo Unix, de hecho, alrededor del 95 por ciento del código del sistema Unix está escrito en C.
- C ++ fue diseñado por Bjarne Stroustrup en la década de 1980 como una extensión de C para proporcionar nuevas características que admitirían la programación orientada a objetos.

PROGRAMACIÓN DE SISTEMAS Y APLICACIONES EN REDES

- Los lenguajes de script (secuencias de comandos) se utilizan ampliamente en la actualidad para una variedad de tareas de sistemas. Por ejemplo, un programa awk puede diseñarse rápidamente para verificar la consistencia de un archivo de contraseña en una máquina Unix.

- Algunos de los lenguajes script de programación principales son awk, Perl, Tcl/Tk y Python.

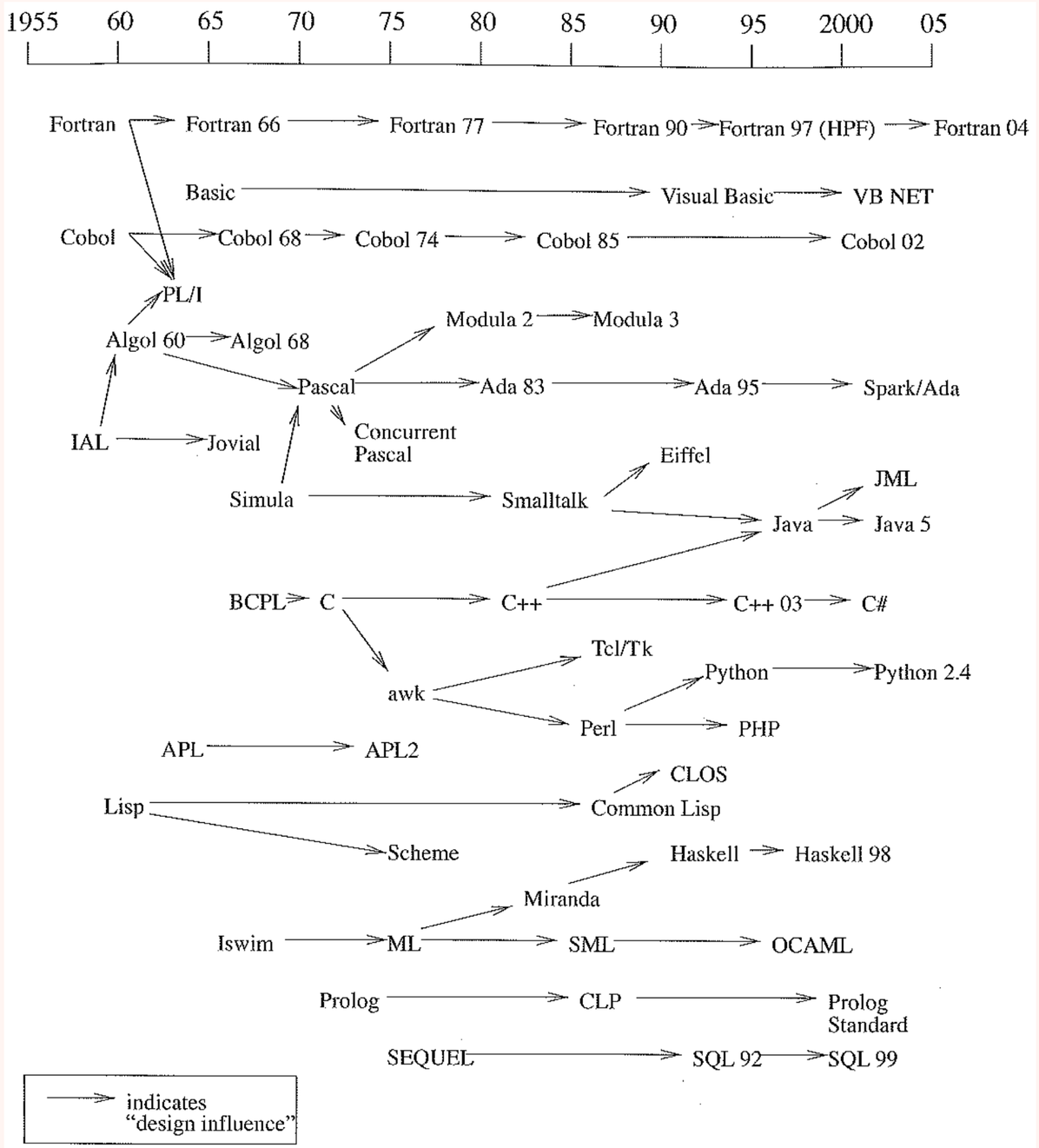
WORLD WIDE WEB

- El área más dinámica para nuevas aplicaciones de programación es Internet, que es el vehículo habilitador para el comercio electrónico y una amplia gama de aplicaciones en la academia, el gobierno y la industria.

- La noción de computación centrada en la Web, y por lo tanto de programación centrada en la Web, está motivada por un modelo interactivo, en el cual un programa permanece continuamente activo esperando que ocurra el próximo evento, respondiendo a ese evento y volviendo a su estado continuamente activo.

WORLD WIDE WEB

- Los lenguajes de programación que brindan soporte a la computación centrada en la Web utilizan la programación basada en eventos, que fomenta la interacción sistema-usuario. La computación centrada en la Web también utiliza el paradigma orientado a objetos, ya que varias entidades que aparecen en la pantalla del usuario se modelan de forma más natural como objetos que envían y reciben mensajes. Los lenguajes de programación que admiten la informática centrada en la Web incluyen Perl, PHP, Visual Basic, Java y Python.



| Figure 1.2 A Snapshot of Programming Language History

Comparison of Top Programming Languages









Metrics	Python	JavaScript	Java	C#	C	C++	Go	R	Swift
Typing Discipline	Strong, dynamically typed	Weakly typed	Statically typed	Statically typed	Weakly typed	Weakly typed	Statically typed	Strong but dynamically typed	Statically, strong and inferred type
PLatform	Linux, graphical user interface, macOS	Visual studio code, Linux, Windows, Mac	Java SE, Java EE, Java ME, Java FX	MonoDe velop, Rider, Visual studio code	WPerl, Cygwin, Linuxeakly typed	Cross-platform software, Cygwin, Perl	PowerPC, FreeBSD, OpenB SD	Windows, MacOS, Windows	iOS, iPadOS, macOS, tvOS and watchOS
Best For	Data analytics, machine learning, even design	Creation of web pages	Creation of complete dynamic applications	Development of desktop applications, web services and web applications	Scripting of system applications and coding embedded systems	Supports object-oriented programming features	Development of cloud applications, DevOps, command line tools	Supports statistical computing and graphics by R core team	System programming, development of mobile and desktop applications and cloud services
Availability	Writing of python scripts, automating tasks, and conduction of data analysis	Available for making interactive web pages	Designing of web applications that may run on a single computer	Used for flexible memory manage ment	Writing the code for operating systems, muchmore complex programs	Widely used for embedded devices and OS kernels	Used for cloud and server side applications, command line tools	Statistical computing, data miners for developing statistical software and data analysis	Used in a wide range of Apple devices like iOS, iPadOS, macOS, tvOS
Designed By	Guido van Rossum	Brendan Eich	James Gosling	Anders Hejlsberg	Dennis Ritchie	Bjarne Stroustrup	Rob Pike, Ken Thomps on	Ross Ihaka	Chris Lattner, John McCall, Doug Gregor
Advantages	Enhanced productivity, easy to learn and write, dynamically typed, vast library support, hassle-free portability	Increased interactivity, richer and enhanced interfaces, less server interaction, great career opportunities	Object-oriented, easy programming language, Supports portability feature, platform independent	Effective memory management, fast and powerful, standard library, object-oriented	Funda mental block for many other programming languages, Portable language, middle-level and structural language, built-infunctions	Mid-level programming language, high portability, fast and powerful, standard library, multi-paradigm	Easy to learn, open sourced, concurrency, static code analysis, fast and hassle-free code implementation	Used for enhanced statistical computing and analysis, supports various data-types, open-source platform, powerful graphics, highly supportive community	Enables a high level of interactivity, fast and modern programming language, much easier to use, easy to locate and correct errors
Disadvantages	Limitations of database, Slower runtime speed, high memory consumption, runtime errors	Browser support Security in the client-side, single inheritance, object-oriented capabilities, lack of debugging facility	Slow and poor performance, no backup facility, provides verbose and complex codes	Run-time checking, test and correct errors, no garbage collection, unsafe, complex	Insufficient memory management, absence of exception handling, Run-time checking, lack of constructor and destructor	No support for garbage pickup, uninsured system security, can cause overload functions	New programming language with notmuch libraries and information, limitedscope, defective dependency management	Used for enhanced statistical computing and analysis, supports various data-types, open-source platform, powerful graphics, highly supportive community	Enables a high level of interactivity, fast and modern programming language, much easier to use, easy to locate and correct errors

COMPILADORES Y MÁQUINAS VIRTUALES

COMPILADORES Y MÁQUINAS VIRTUALES

- Una implementación de un lenguaje de programación requiere que los programas en el lenguaje sean analizados y luego traducidos a una forma que puede ser:

1. Ejecutado por una computadora (es decir, una "máquina real"), o

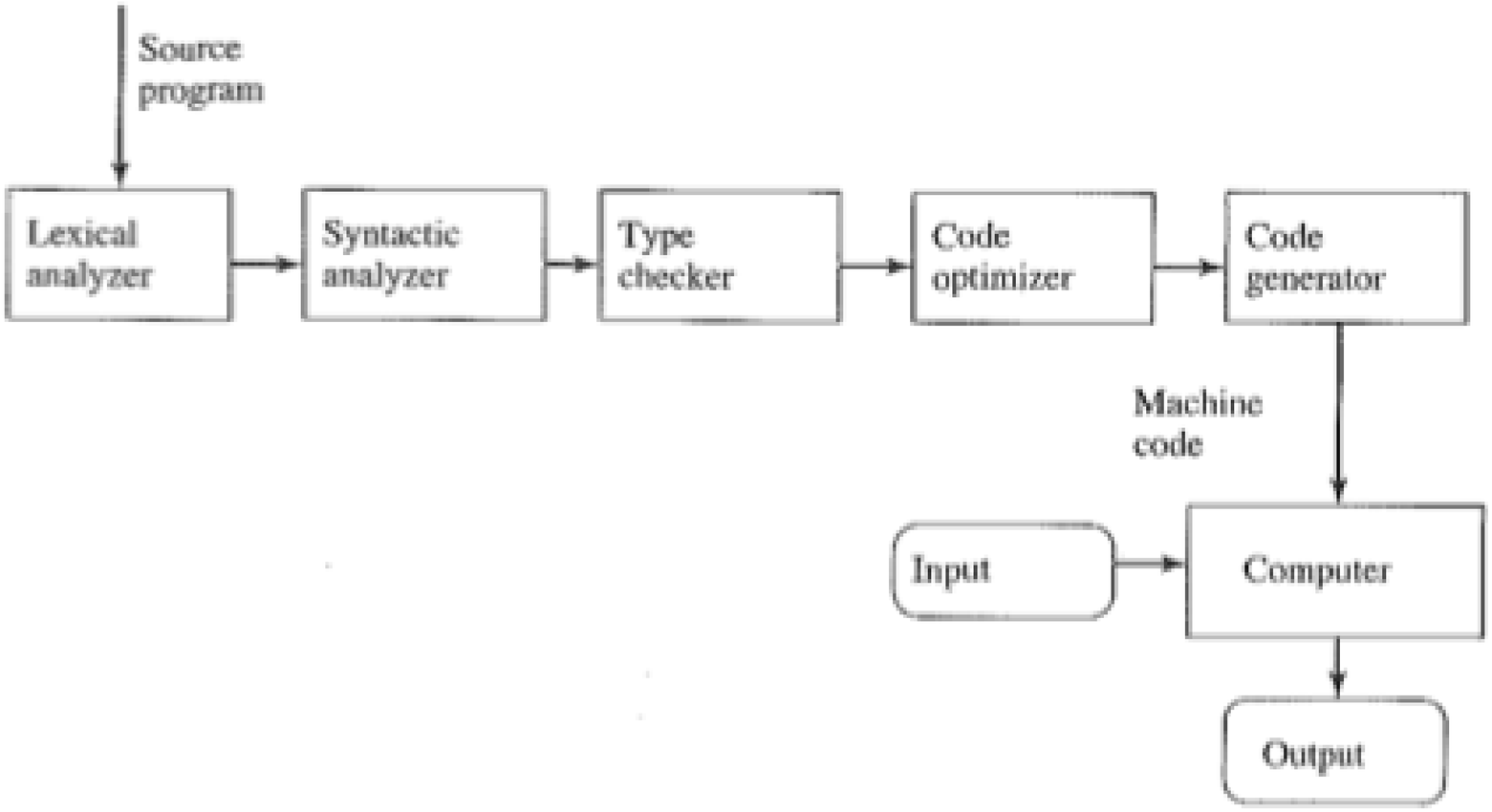
2. Ejecutado por un intérprete (es decir, un software que simula una "máquina virtual" y se ejecuta en una máquina real).

La traducción del primer tipo a menudo se denomina **compilación**, mientras que la traducción del segundo es llamado **interpretación**.

COMPILADORES

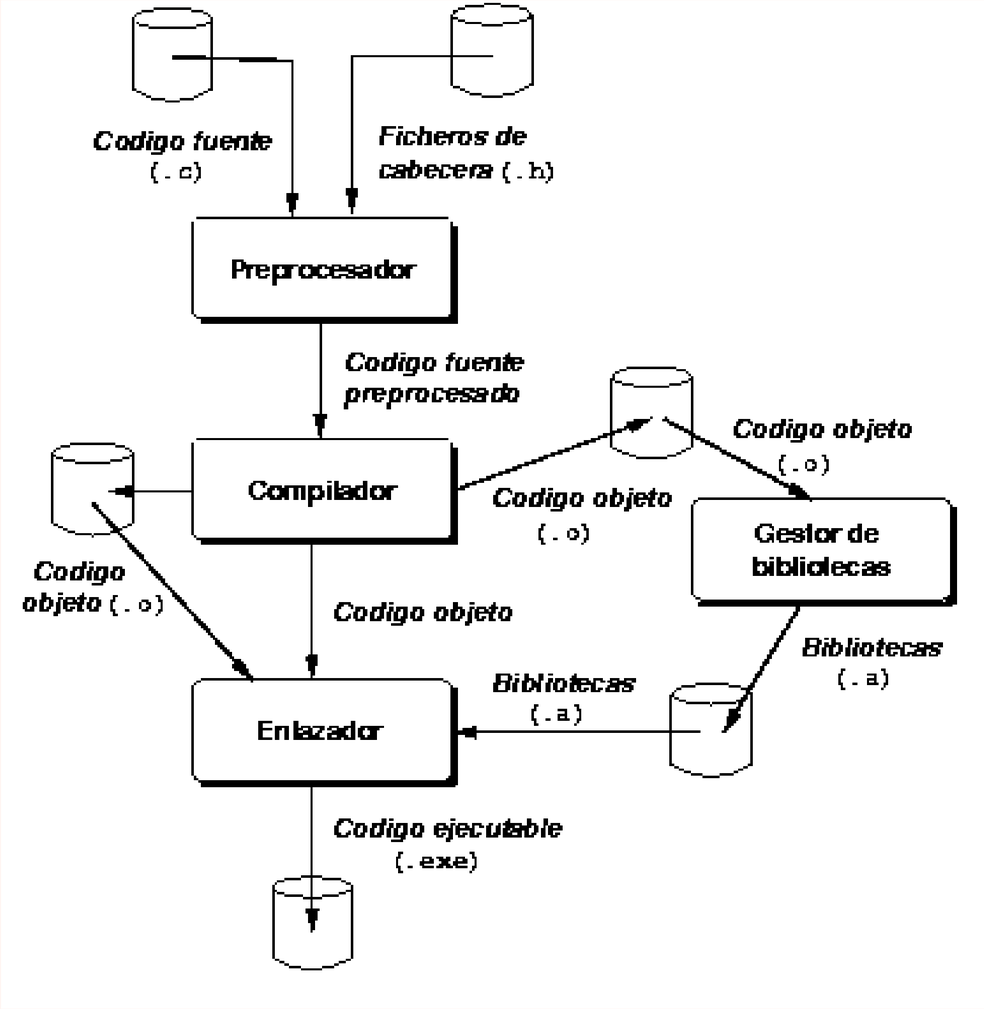
- **El proceso de compilación traduce un programa fuente al lenguaje de una computadora. Posteriormente, el código de máquina resultante se puede ejecutar en esa computadora. Por ejemplo, Fortran, Cobol, C y C ++ son lenguajes compilados típicos. Este proceso se ilustra en la siguiente figura:**

COMPILADORES



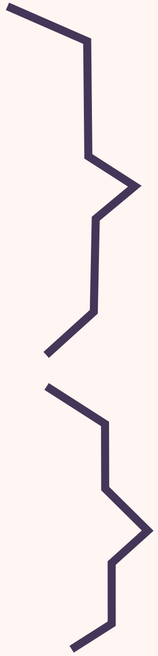
| Figure 1.4 The Compile-and-Run Process

EL COMPILADOR DE C



COMPILADORES

- **Las cinco etapas del proceso de compilación son:**
  - **Análisis léxico**
  - **Análisis sintáctico**
  - **Verificación de tipos**
  - **Optimización de código**
  - **Generación de código**



Búsqueda y notificación de errores al programador.

Generación de código de máquina eficiente para ejecutar en la computadora de destino.

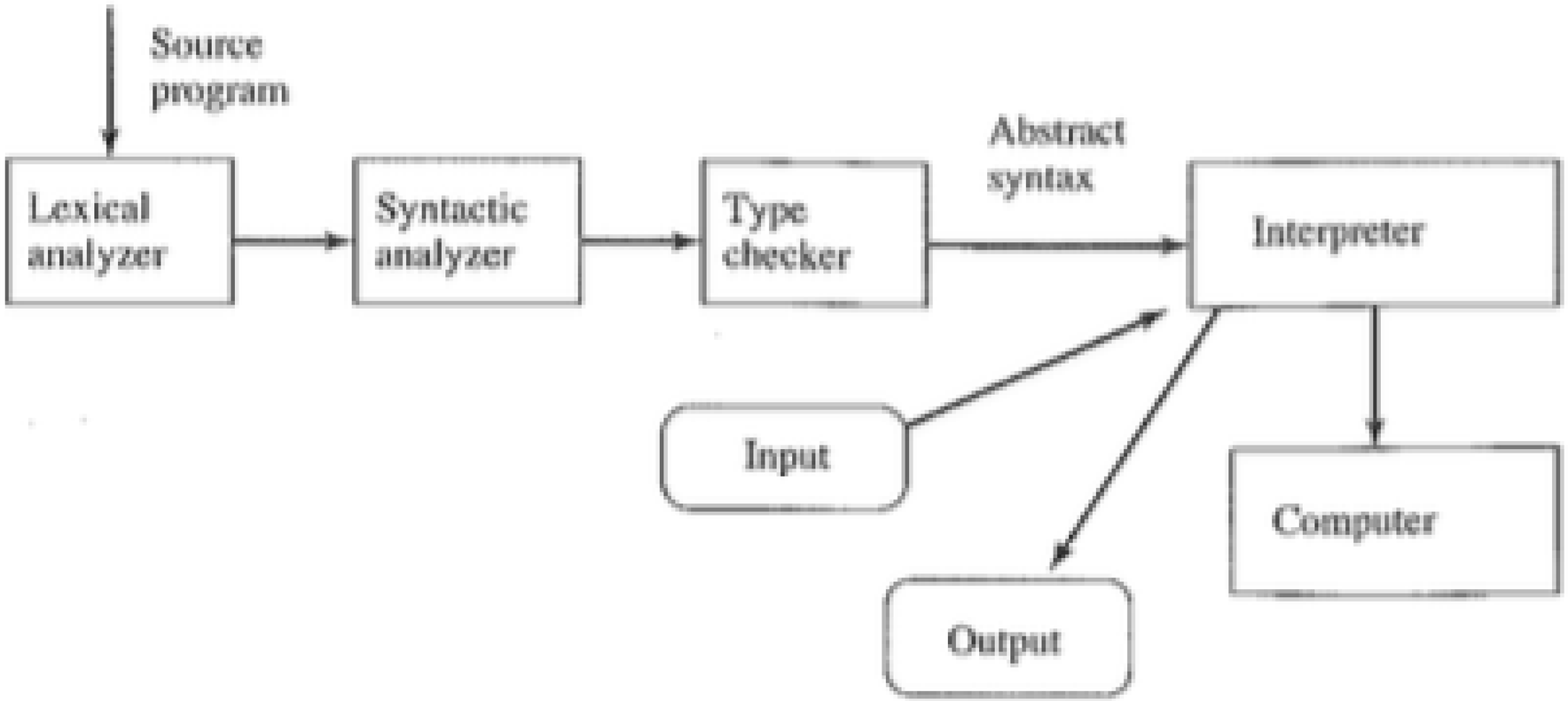
COMPILADORES

- **El código de máquina de un programa compilado se combina con su entrada para ejecutarse en un paso separado que sigue a la compilación. Los errores en tiempo de ejecución generalmente se pueden rastrear hasta el programa fuente mediante el uso de un depurador.**

MÁQUINAS VIRTUALES E INTÉRPRETES

- Los demás lenguajes se implementan mediante un proceso interpretativo, como se muestra en la siguiente Figura. Aquí, el programa fuente se traduce a una forma abstracta intermedia, que luego se ejecuta interpretativamente. Lisp y Prolog, por ejemplo, a menudo se implementan utilizando intérpretes (aunque también existen compiladores para estos lenguajes).

MÁQUINAS VIRTUALES E INTÉRPRETES



| Figure 1.5 Virtual Machines and Interpreters

MÁQUINAS VIRTUALES E INTÉRPRETES

- **Como sugiere la Figura anterior, las primeras tres etapas de un compilador también ocurren en un intérprete. Sin embargo, la representación abstracta del programa que surge de estas tres etapas se convierte en objeto de ejecución por parte de un intérprete.**

- **El intérprete en sí es un programa que ejecuta los pasos del programa abstracto mientras se ejecuta en una máquina real. El intérprete suele estar escrito en un idioma distinto del que se interpreta.**

Definición de «máquina virtual»

Las máquinas virtuales son ordenadores de software que proporcionan la misma funcionalidad que los ordenadores físicos. Como ocurre con los ordenadores físicos, ejecutan aplicaciones y un sistema operativo. Sin embargo, las máquinas virtuales son archivos informáticos que se ejecutan en un ordenador físico y se comportan como un ordenador físico. En otras palabras, las máquinas virtuales se comportan como sistemas informáticos independientes.

Ventajas de las máquinas virtuales:

- Proporcionan opciones de recuperación ante desastres y distribución de aplicaciones.
- Las máquinas virtuales se gestionan y mantienen fácilmente, y ofrecen una amplia disponibilidad.
- Es posible ejecutar varios entornos de sistemas operativos en un solo ordenador físico.

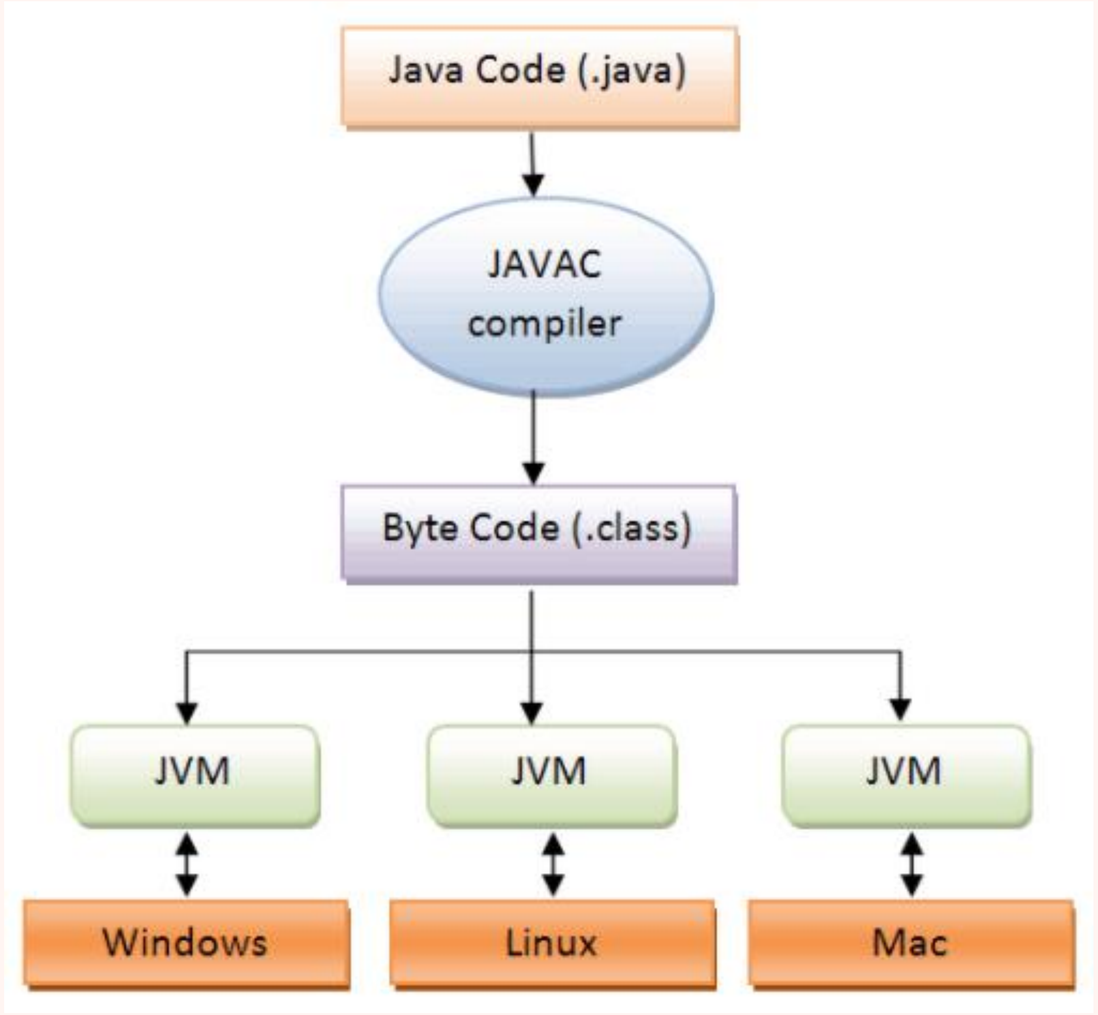
Desventajas de las máquinas virtuales:

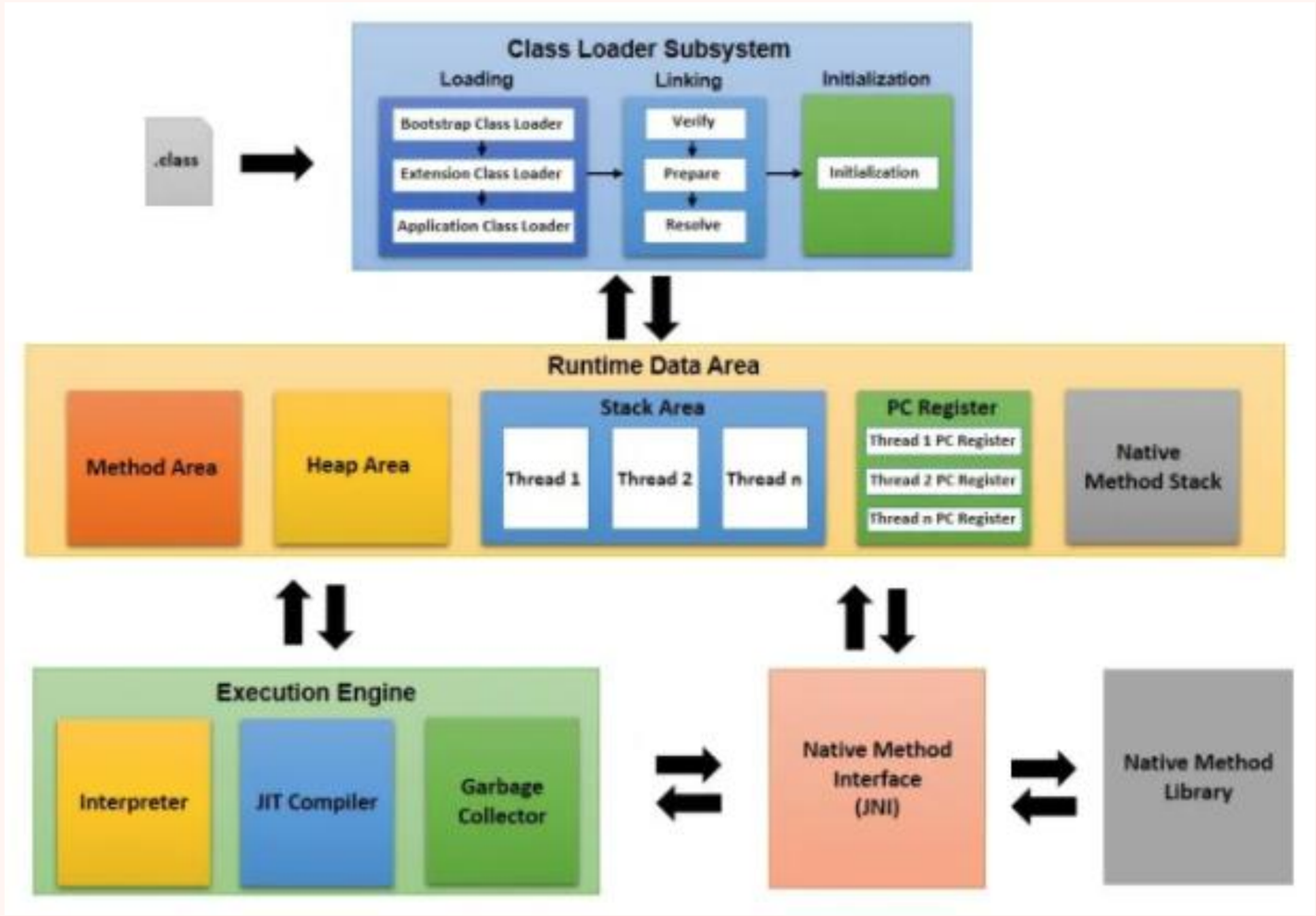
- La ejecución de varias máquinas virtuales en una máquina física puede provocar que el rendimiento sea inestable.
- Las máquinas virtuales son menos eficientes y más lentas que un ordenador físico.

Los dos tipos de máquinas virtuales:

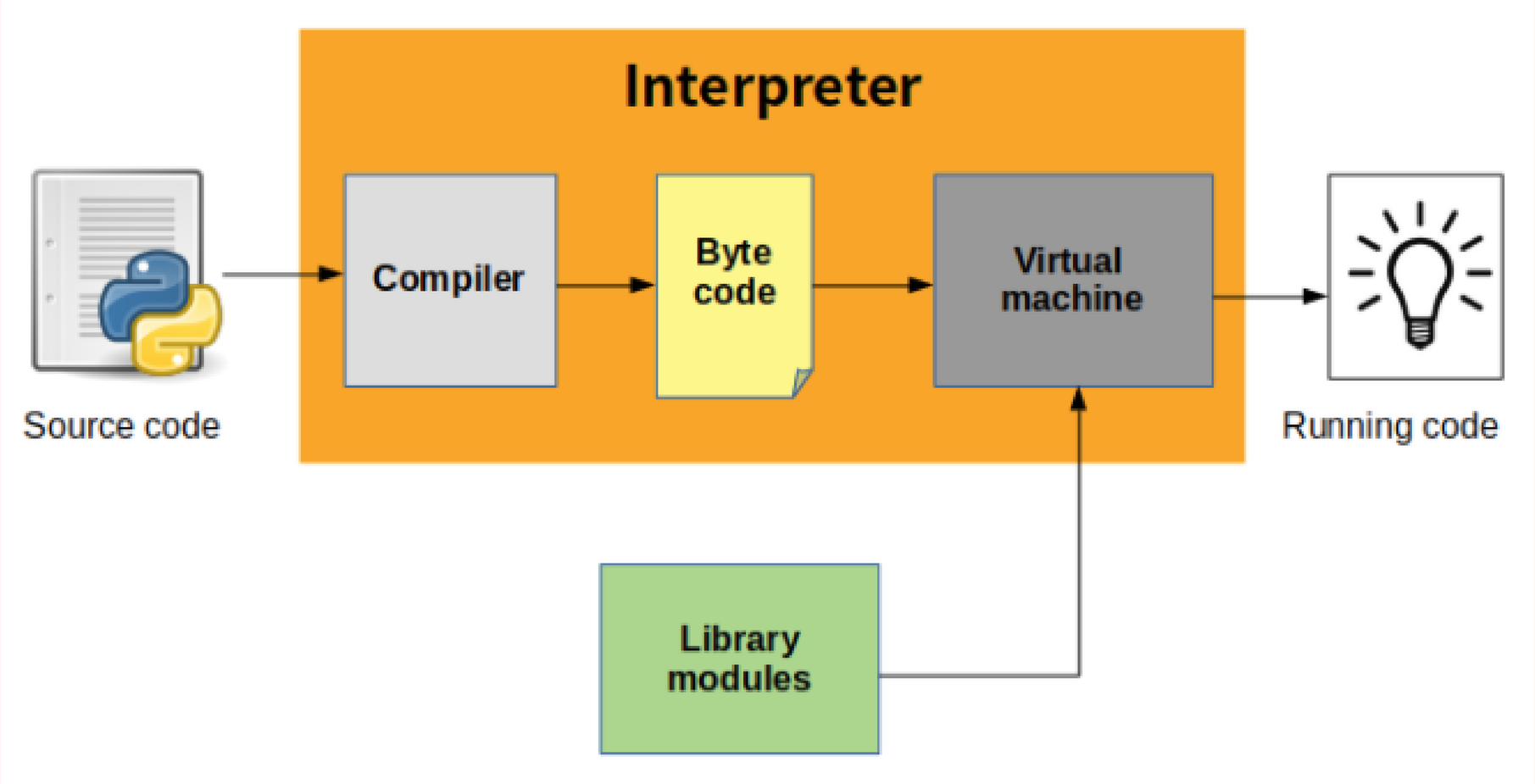
1. Las **máquinas virtuales de procesos** ejecutan programas informáticos en un entorno independiente de la plataforma. Enmascaran la información del hardware o el sistema operativo subyacentes. Esto permite que el programa se ejecute de la misma manera en cualquier plataforma.
2. Las **máquinas virtuales de sistema** permiten compartir los recursos físicos de un ordenador host entre varias máquinas virtuales.

MÁQUINA VIRTUAL DE JAVA

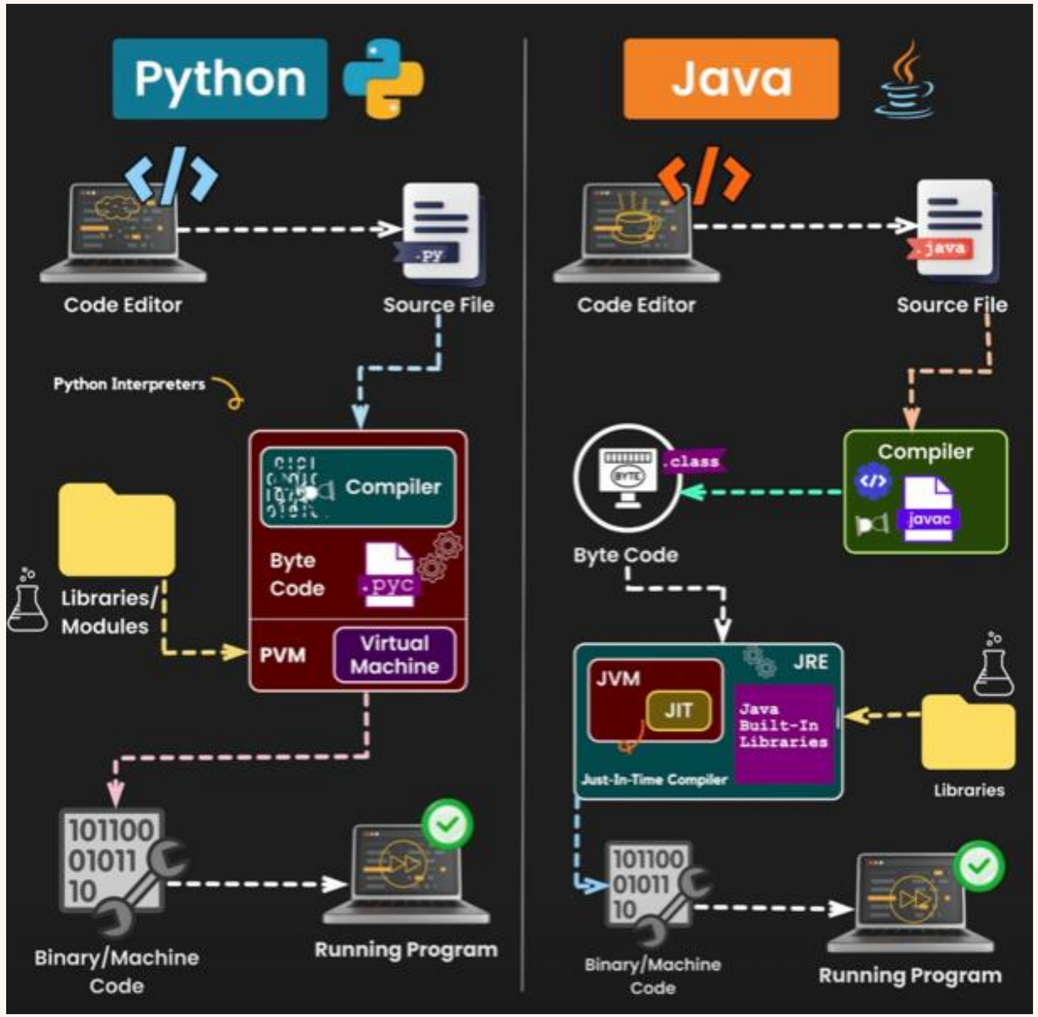


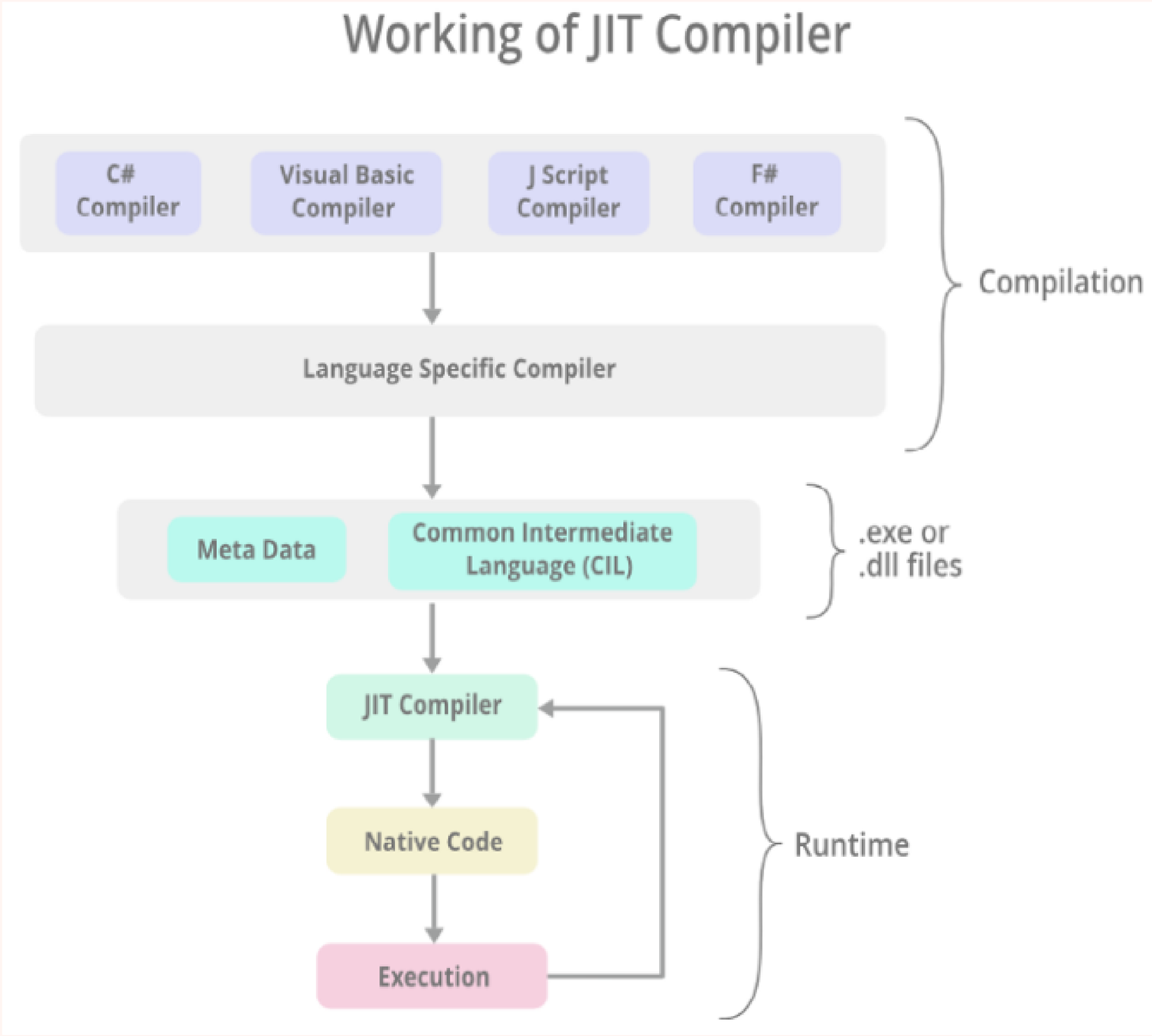


MÁQUINA VIRTUAL DE PYTHON



MÁQUINA VIRTUAL: JAVA Y PYTHON




Working of JIT Compiler
C# Compiler
Visual Basic Compiler
J Script Compiler
F# Compiler
Language Specific Compiler
Compilation
Meta Data
Common Intermediate Language (CIL)
.exe or .dll files
JIT Compiler
Native Code
Execution
Runtime

DEFINICIÓN DE UN LENGUAJE

DEFINICIÓN DE UN LENGUAJE

- La definición de lenguaje se puede dividir libremente en dos partes: sintaxis o estructura y semántica o significado. Discutimos cada una de estas categorías.

SINTAXIS

- La sintaxis de un lenguaje de programación es en muchos aspectos como la gramática de un lenguaje natural.
- Es la descripción de las formas en que se pueden combinar diferentes partes del lenguaje para formar frases y, en última instancia, oraciones.
- Como ejemplo, la sintaxis de la instrucción if en C se puede describir en palabras de la siguiente manera:

SINTAXIS

- **P R O P E R T Y: Una declaración if consta de la palabra "if" seguida de una expresión entre paréntesis, seguida de una sentencia, seguida de una parte “else” opcional que consta de la palabra "else" y otra sentencia.**

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

SINTAXIS

- La descripción de la sintaxis del lenguaje es una de las áreas donde las definiciones formales han ganado aceptación, y la sintaxis de todos los lenguajes ahora se da usando una gramática. Por ejemplo, una regla gramatical para la instrucción if de C se puede escribir de la siguiente manera:

<if-statement> ::= if (<expression>) <statement>
[else <statement>]

or (using special characters and formatting):

if-statement → `if` (*expression*) *statement*
[`else` *statement*]

SINTAXIS

- La estructura léxica de un lenguaje de programación es la estructura de las palabras del lenguaje, que generalmente se denominan tokens. Por tanto, la estructura léxica es similar a la ortografía en un lenguaje natural. En el ejemplo de una instrucción if de C, las palabras if y else son tokens. Otros tokens en lenguajes de programación incluyen identificadores (o nombres), símbolos para operaciones, como + y * y símbolos de puntuación especiales como el punto y coma (;) y el punto (.).

SEMÁNTICA

- La sintaxis representa solo la estructura superficial de un idioma y, por lo tanto, es solo una pequeña parte de la definición de un idioma. La semántica, o significado, de una lengua es mucho más compleja y difícil de describir con precisión. La primera dificultad es que el "significado" se puede definir de muchas formas diferentes. Normalmente, describir el significado de un fragmento de código implica describir los efectos de ejecutar el código, pero no existe una forma estándar de hacerlo. Además, el significado de un mecanismo en particular puede involucrar interacciones con otros mecanismos en el lenguaje, de modo que una descripción completa de su significado en todos los contextos puede volverse extremadamente compleja.

SEMÁNTICA

- Para continuar con nuestro ejemplo de la instrucción if de C, su semántica se puede describir en palabras como sigue (adaptado de Kernighan y Richie, El lenguaje de Programación C):

Una instrucción if se ejecuta evaluando primero su expresión, que debe tener un tipo aritmético o puntero, incluidos todos los efectos secundarios, y si se compara con un valor desigual a 0, se ejecuta la instrucción que sigue a la expresión. Si hay una parte else y la expresión es 0, se ejecuta la instrucción que sigue a “else".

PARADIGMAS

paradigma

Del lat. tardío *paradigma*, y este del gr. παράδειγμα *parádeigma*.

1. m. Ejemplo o ejemplar.

2. m. Teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento. *El paradigma newtoniano.*

3. m. *Ling.* Relación de elementos que comparten un mismo contexto fonológico, morfológico o sintáctico en función de sus propiedades lingüísticas.

4. m. *Ling.* Esquema formal en el que se organizan las palabras que admiten modificaciones flexivas o derivativas.

PARADIGMAS

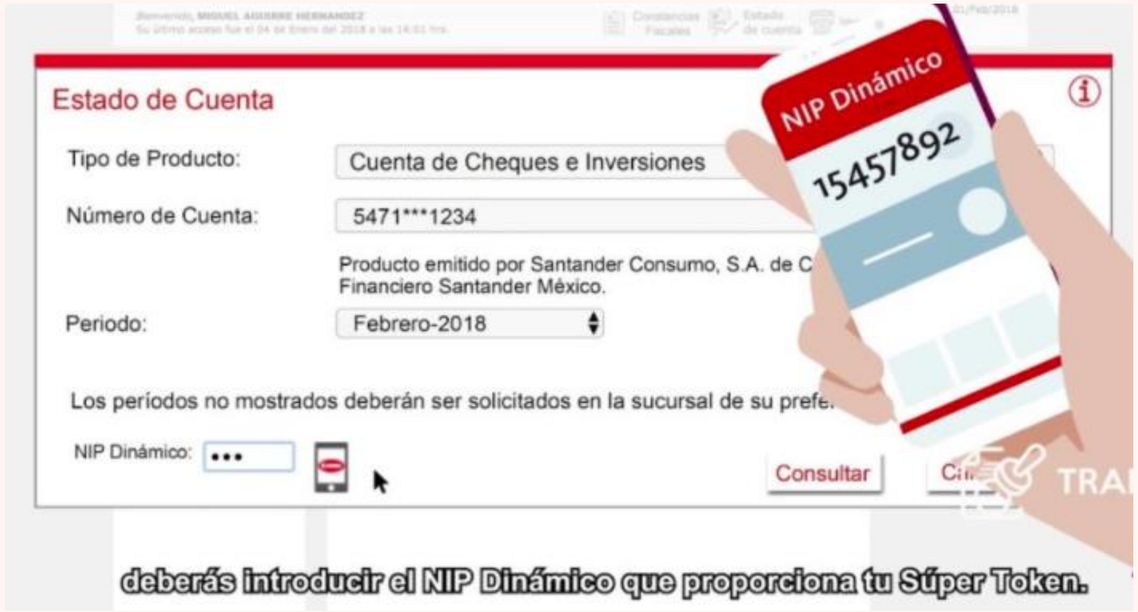
- En general, pensamos en un "paradigma" como un patrón de pensamiento que guía una colección de actividades relacionadas. Un paradigma de programación es un patrón de pensamiento de resolución de problemas que subyace a un género particular de programas y lenguajes.

- Un paradigma de programación indica un método de realizar cómputos y la manera en que se deben estructurar y organizar las tareas que debe llevar a cabo un programa.

- Los paradigmas fundamentales están basados en diferentes modelos de cómputo y por lo tanto afectan a las construcciones más básicas de un programa.

TIPOS DE PARADIGMAS

- **Declarativo**: **Qué** se debe calcular.



- **Imperativo**: **Cómo** se deben hacer las operaciones y cálculos.

TIPOS DE PARADIGMAS

Declarativo: Qué se debe calcular.

- El programador describe o declara el problema mediante proposiciones o expresiones matemáticas.
- No existe un orden de evaluación prefijado.
- El control de flujo suele estar asociado a la composición funcional o la recursividad.
- Las variables son **nombres** asociados a **definiciones**, y una vez instanciadas son inmutables.
- No existe sentencia de asignación

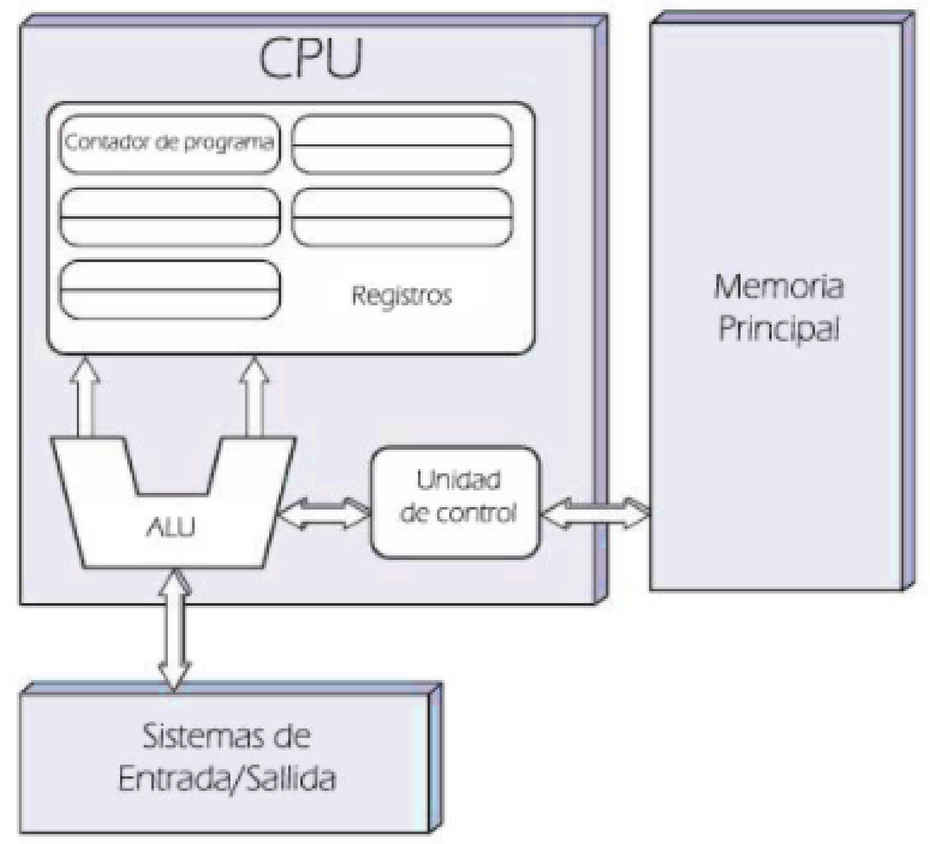
Imperativo: Cómo se deben hacer las operaciones y cálculos.

- Un cómputo consiste en una serie de sentencias, ejecutadas según un control de flujo **explícito**, que **modifican el estado** del programa
- Las variables son **celdas de memoria** que contienen datos (o referencias), pueden ser modificadas, y representan el **estado** del programa
- La sentencia principal es la **asignación**
- Está basado en algoritmos.

PARADIGMA IMPERATIVO

- Los lenguajes de programación comenzaron imitando y abstrayendo las operaciones de una computadora. No es de extrañar que el tipo de ordenador para el que fueron escritos haya tenido un efecto significativo en su diseño.

- En la mayoría de los casos, la computadora en cuestión era el modelo de John von Neumann: una única Unidad Central de Procesamiento que ejecuta secuencialmente instrucciones que operan sobre valores almacenados en la memoria.



Arquitectura von Newmann

PARADIGMA IMPERATIVO

- Estas son características típicas de un lenguaje basado en el modelo de von Neumann: las variables representan ubicaciones de memoria y la asignación permite que el programa opere en estas ubicaciones de memoria.

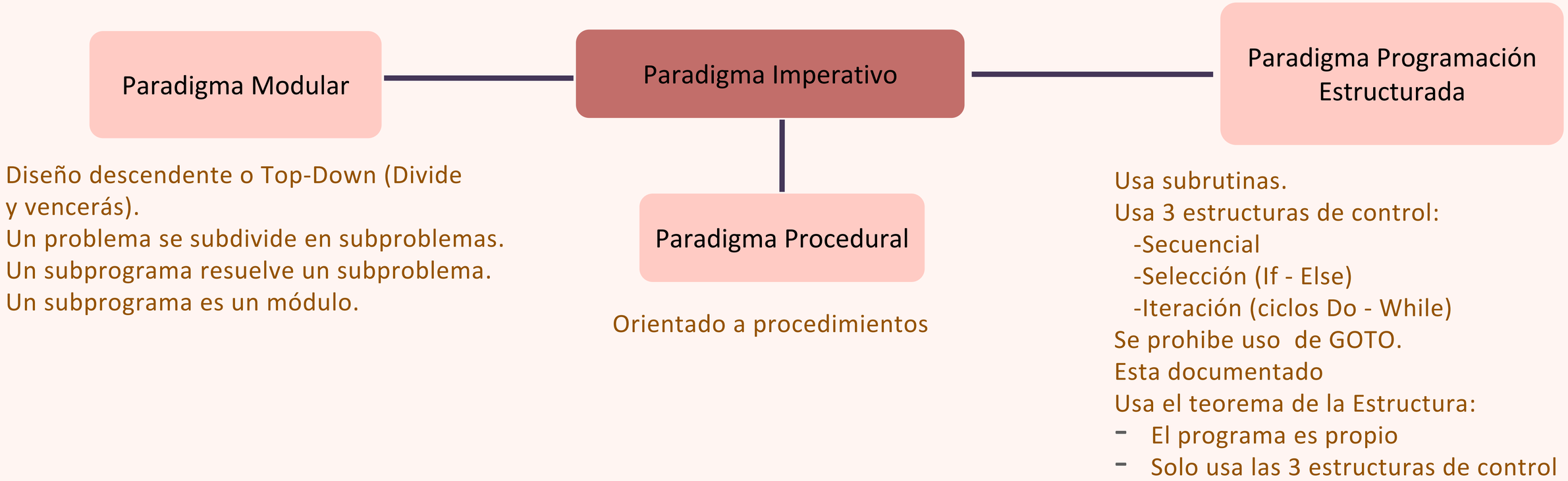
- Un lenguaje de programación que se caracteriza por estas tres propiedades:

  - la ejecución secuencial de instrucciones,
  - el uso de variables que representan ubicaciones de memoria y
  - el uso de asignación para cambiar los valores de las variables

se denomina lenguaje imperativo, porque su característica principal es una secuencia de enunciados que representan mandatos o imperativos.

PARADIGMA IMPERATIVO

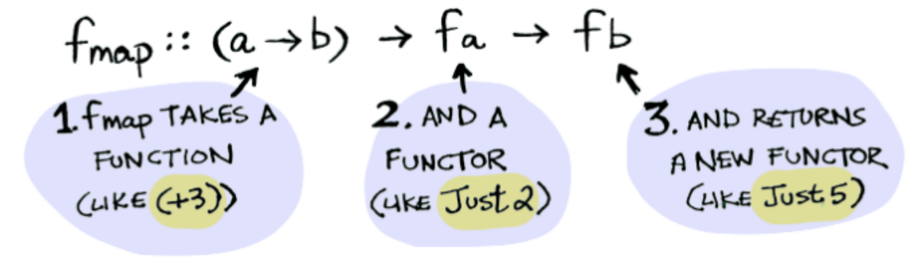
- **Al paradigma imperativo se encuentran asociados los paradigmas procedural, estructurado, y modular.**



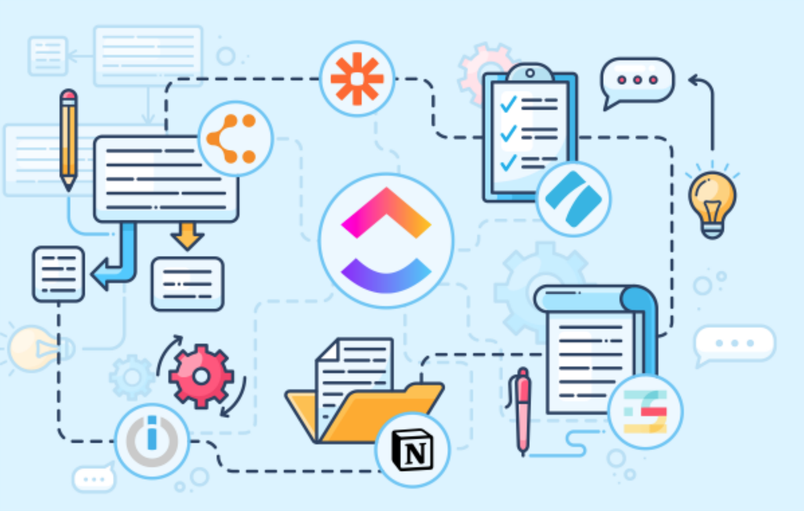
PARADIGMA DECLARATIVO

- Dos paradigmas alternativos para describir la computación provienen de las matemáticas:
  - El paradigma funcional se basa en la noción abstracta de una función tal como se estudia en el cálculo lambda.
  - El paradigma lógico se basa en la lógica simbólica.
- La importancia de estos paradigmas es su correspondencia con los fundamentos matemáticos, lo que les permite describir el comportamiento del programa de manera abstracta y precisa. Esto, a su vez, hace que sea mucho más fácil determinar si un programa se ejecutará correctamente (incluso sin un análisis teórico completo) y hace posible escribir código conciso para tareas muy complejas.

PARADIGMA DECLARATIVO



Paradigma Funcional



Paradigma Programación reactiva

```
Todos los hombres son mortales
Socrates es un hombre
------------------------------
Socrates es mortal
```

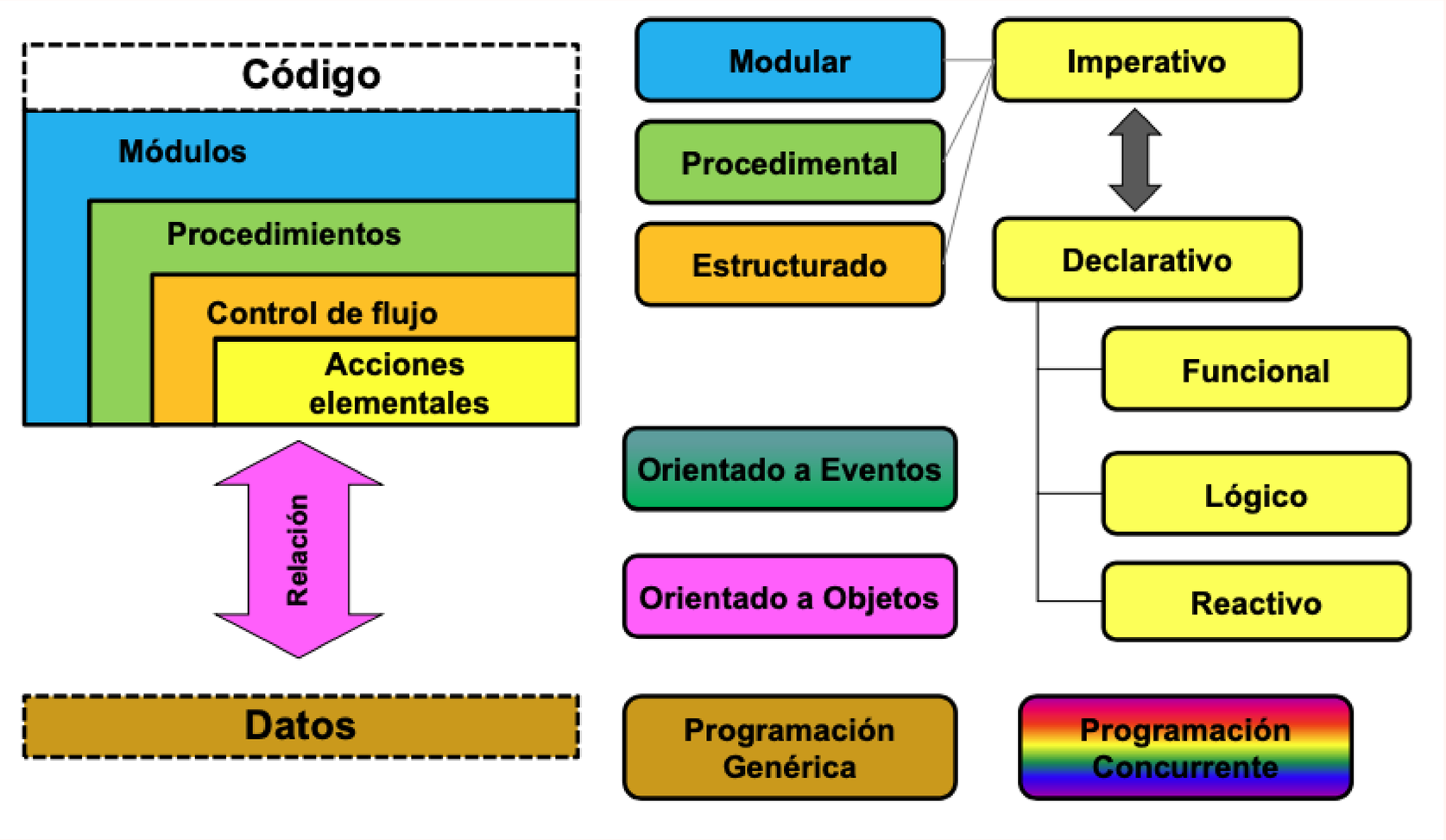
← Lógica de primer orden

Paradigma Lógico

```
% Hechos:
es_español("Manolo").
es_italiano("Marco").
es_colombiano("Marcelo").

% Reglas:
es_europeo(A) :- es_español(A).
es_europeo(A) :- es_italiano(A).
es_americano(A) :- es_colombiano(A).
es_terricola(A) :- es_europeo(A).
es_terricola(A) :- es_americano(A).
son_del_mismo_continente(A,B) :- es_europeo(A), es_europeo(B).
son_del_mismo_continente(A,B) :- es_americano(A), es_americano
```

PARADIGMAS


Código
Módulos
Procedimientos
Control de flujo
Acciones elementales
Relación
Datos
Modular
Procedimental
Estructurado
Imperativo
Declarativo
Funcional
Lógico
Reactivo
Orientado a Eventos
Orientado a Objetos
Programación Genérica
Programación Concurrente

PARADIGMA ORIENTADO A OBJETOS

- El paradigma orientado a objetos (OO), ha adquirido una enorme importancia en los últimos 20 años. Los lenguajes OO permiten a los programadores escribir código reutilizable que funciona de una manera que imita el comportamiento de los objetos en el mundo real;

- Los programadores pueden usar su intuición natural sobre el mundo para comprender el comportamiento de un programa y construir un código apropiado.

- En cierto sentido, el paradigma OO es una extensión del paradigma imperativo, en el sentido de que se basa principalmente en la misma ejecución secuencial con un conjunto cambiante de ubicaciones de memoria, particularmente en la implementación de objetos. La diferencia es que los programas resultantes consisten en una gran cantidad de piezas muy pequeñas cuyas interacciones se controlan cuidadosamente y, sin embargo, se cambian fácilmente.

PARADIGMA ORIENTADO A OBJETOS

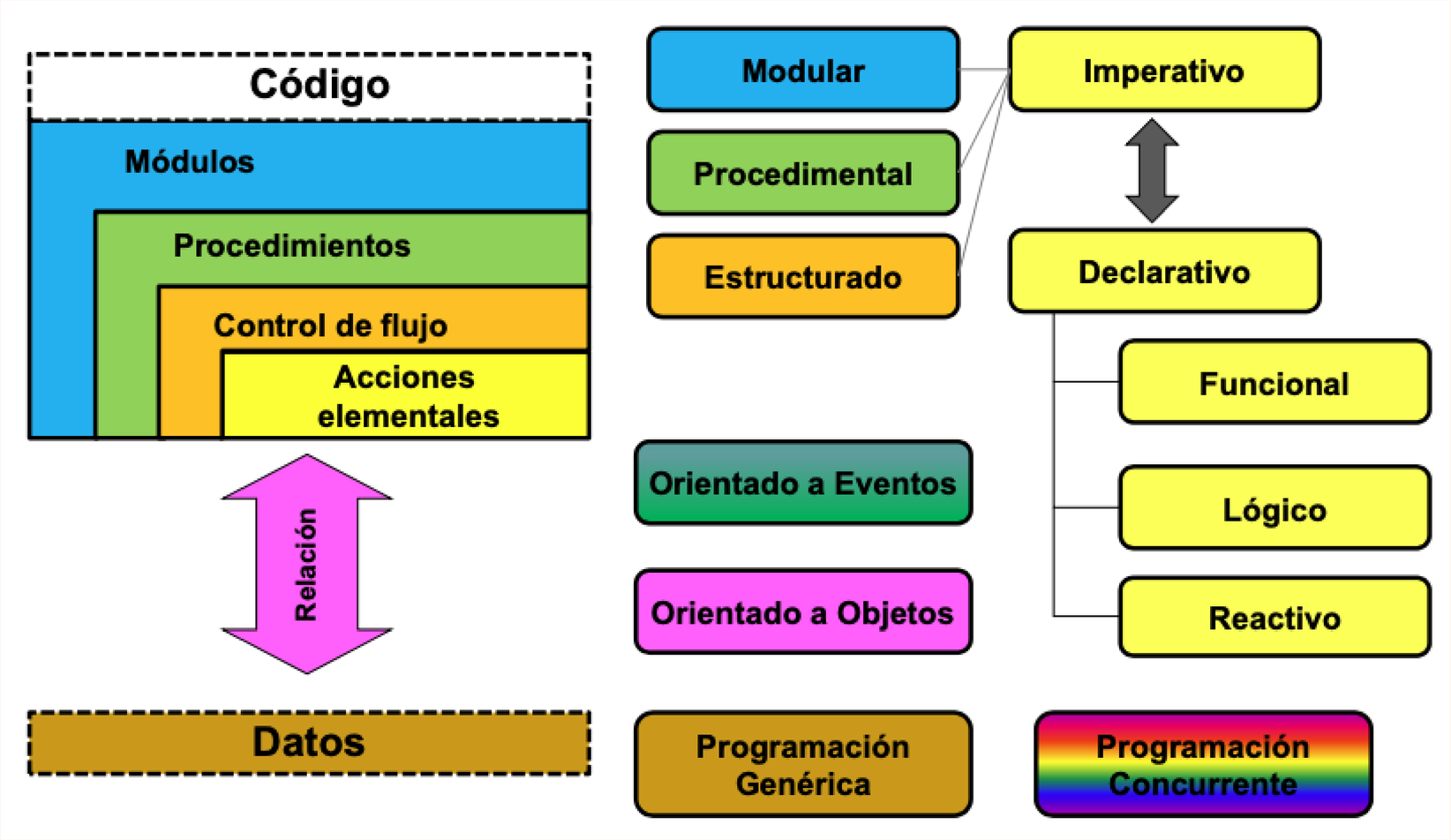
- Además, en un nivel más alto de abstracción, la interacción entre objetos a través del paso de mensajes puede correlacionarse muy bien con la colaboración de procesadores paralelos, cada uno con su propia área de memoria.

- El paradigma orientado a objetos se ha convertido esencialmente en un nuevo estándar, tanto como lo fue el paradigma imperativo en el pasado.

PARADIGMAS

- Otros paradigmas se centran en la estructura y organización de los programas, y son compatibles con los fundamentales.
  - Programación estructurada
  - Programación modular
  - Programación orientada a objetos ,
  - programación orientada a eventos
- También tenemos a los paradigmas asociados a la concurrencia y a los sistemas de tipado.

PARADIGMAS


Código
Módulos
Procedimientos
Control de flujo
Acciones elementales
Relación
Datos
Modular
Procedimental
Estructurado
Imperativo
Declarativo
Funcional
Lógico
Reactivo
Orientado a Eventos
Orientado a Objetos
Programación Genérica
Programación Concurrente

REFERENCIAS

- Tucker, A. (2007). *Programming languages: Principles and paradigms.* (Second edition). McGraw-Hill Education.

- Louden, K. (2011). Programming languages: Principles and practice (Third edition). Course technology.

- Vaca, C. (2011, 11 febrero). *Paradigmas de programación*. Departamento de Informática Universidad de Valladolid. https://www.infor.uva.es/~cvaca/asigs/docpar/intro.pdf