IC-5701
Compiladores e Intérpretes

Profesor:
Ing. Allan Rodríguez Dávila, MGP

Agenda

TEC

- Aspectos Generales
- Especificación de Lenguajes
- Procesadores de Lenguajes
- Conceptos de Compilación
- Máquinas Reales y Abstractas

Procesadores de Lenguajes

Procesadores de Lenguajes

- Un procesador de lenguaje es un programa que procesa programas expresados en un lenguaje en particular (Lenguaje fuente). Además es implementado en algún lenguaje.
- Tipos:
  - Traductores
    - Compiladores
  - Intérpretes

Procesadores de Lenguajes

- Lectura (50 minutos)
  - Identificar características de los Traductores y Compiladores
  - Identificar características de los Intérpretes
  - Diferencias Compiladores vs Intérpretes
  - Explicar los diagramas Tombstone

Traductores

- Acepta texto en un lenguaje y genera texto equivalentemente semántico en otro lenguaje
  - Fuente – Destino
- Tipos de traductores de lenguajes de programación:
  - Ensambladores
  - Compiladores

Compiladores

- Traduce desde un lenguaje de alto nivel a uno de bajo nivel
  - Ensamblador traduce de código ensamblador al código máquina correspondiente

  Programa fuente – Programa objeto
- Tipos:
  - Ensambladores (disassembler)
  - Compiladores (decompilador)

Compiladores

- Chequeo de forma
  - Sintáctico
  - Restricciones Contextuales
- Varios lenguajes están involucrados en la traducción:
  - Lenguaje fuente, lenguaje destino, lenguaje del traductor (implementación)

Compiladores

- Cross compiler
  - Compilador que corre en una máquina pero genera código para otra máquina
- Compilación en dos etapas
  - Compila a un código intermedio y luego éste lo compila a un código destino

Intérpretes

- Acepta cualquier programa expresado en un lenguaje en particular y lo corre inmediatamente.
- Funciona procesando, analizando y ejecutando las instrucciones del programa fuente.
  - Uno a la vez

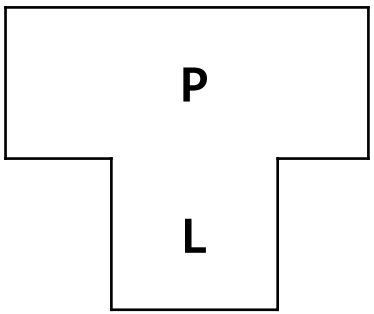
Intérpretes

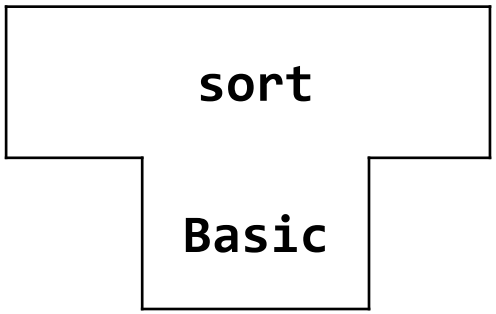
- No traduce el programa antes de la ejecución.
- Tiende a ser lento.

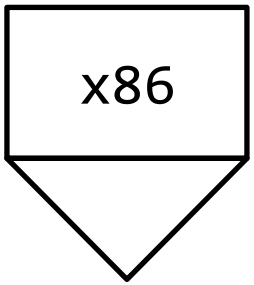
Compiladores vs Intérpretes

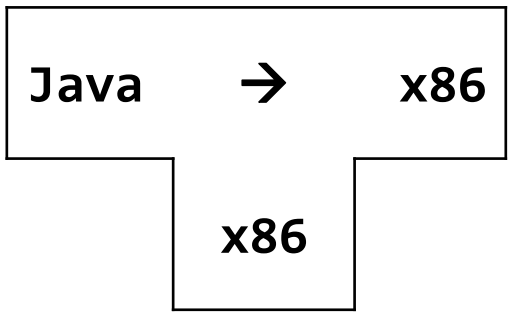
Diagramas Tombstone

- Utilizados por los procesadores de lenguajes
- Facilitan la representación de los traductores e intérpretes









Compiladores

- Tokens vs Lexemas
- Tabla de símbolos
- Boostrapping
- Código intermedio

Conceptos de Compilación

TEC

Compilador

- Programa que lee un programa en un lenguaje (el lenguaje fuente) y lo traduce en un programa equivalente en otro lenguaje (lenguaje destino).

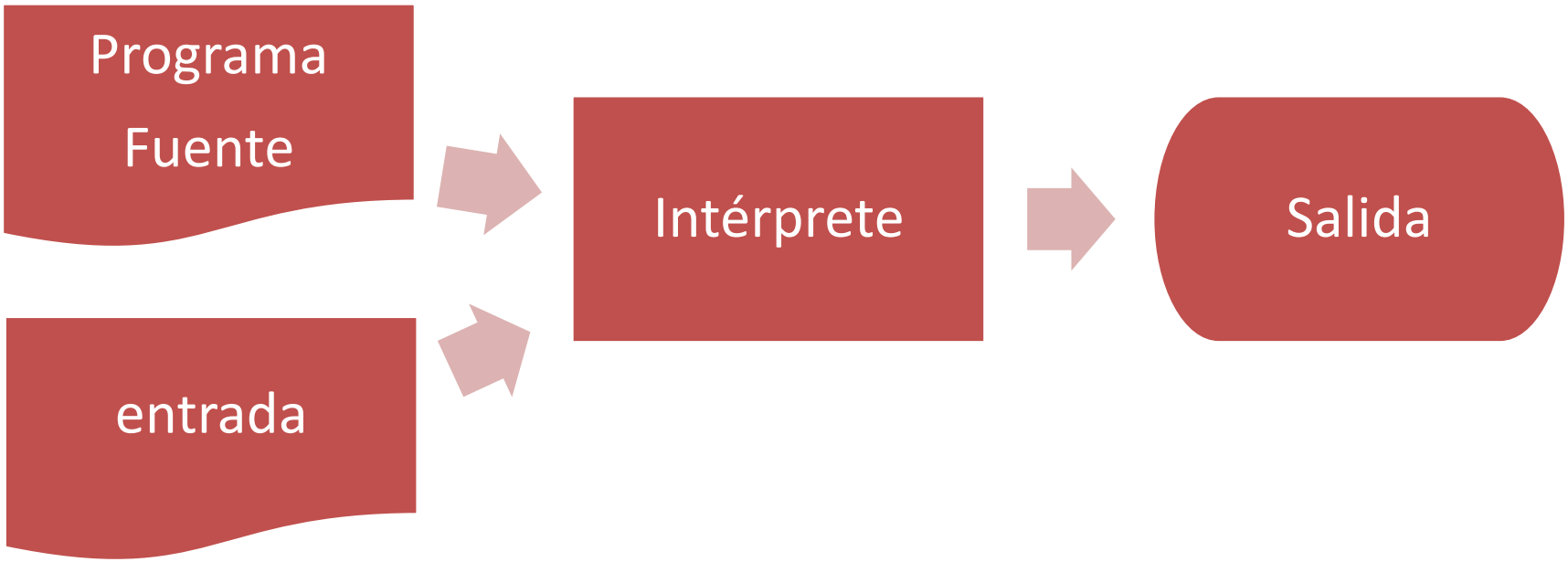
Compilador

TEC

Intérprete

- Ejecuta directamente las operaciones especificadas en el programa de origen (fuente) con las entradas proporcionadas por el usuario

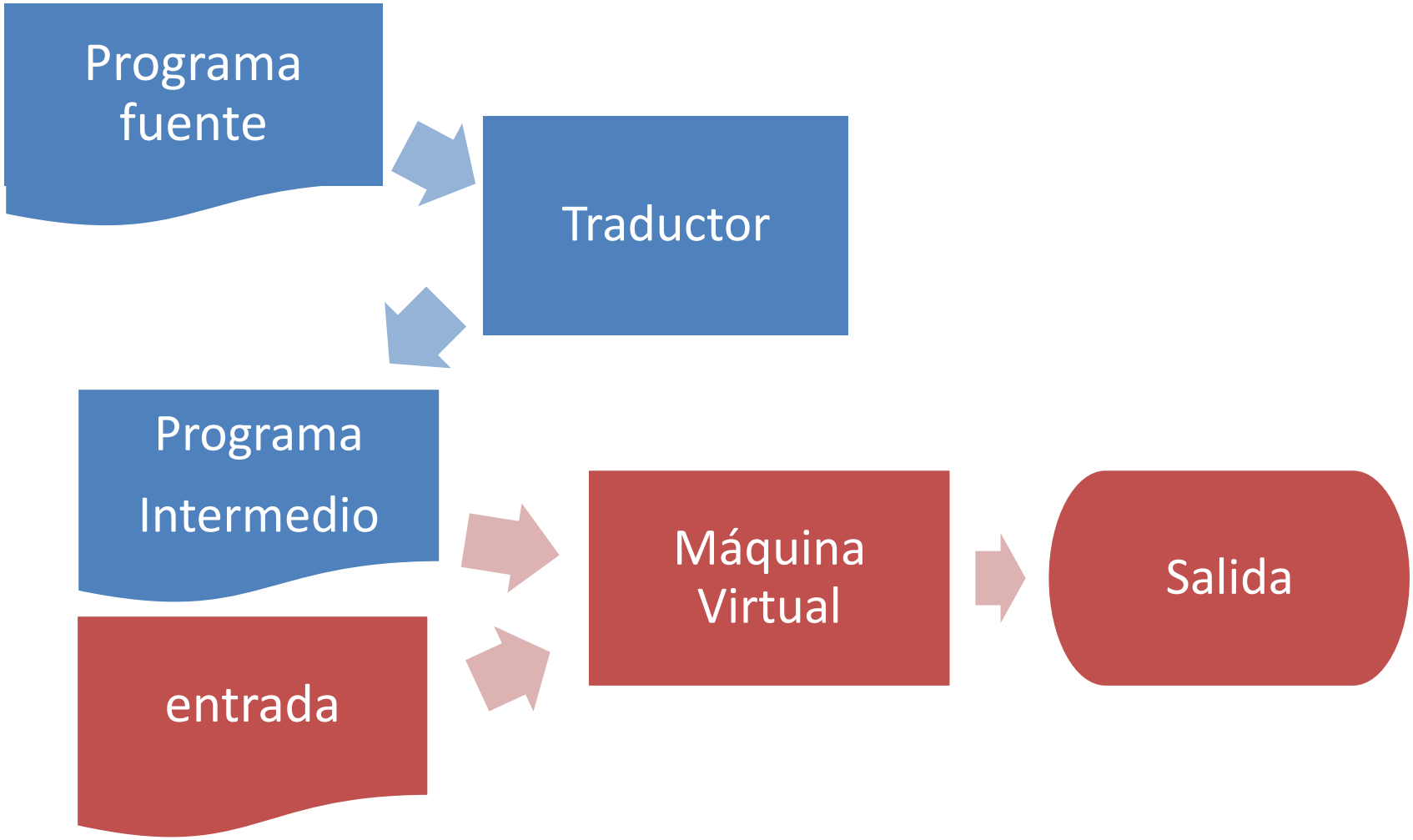
Intérprete



Compiladores Just-in-time

- Primero traducen a código intermedio.
- Traducen a código máquina justo antes de ejecutar el programa intermedio.
- Java

Just-in-time


Programa fuente
Traductor
Programa Intermedio
entrada
Máquina Virtual
Salida

TEC

Tabla de símbolos

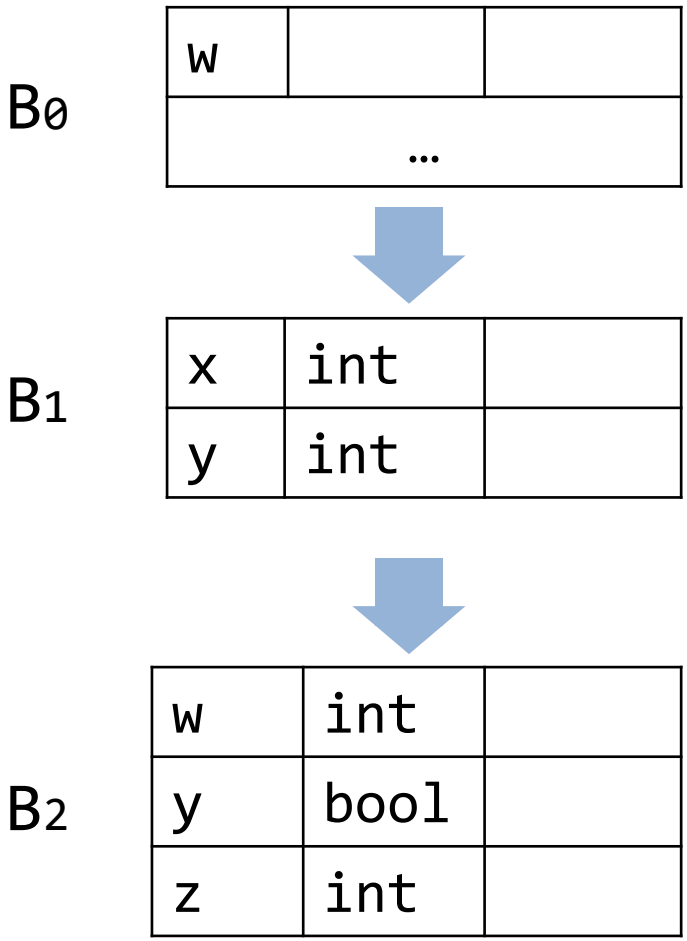
- Estructuras de datos utilizada para guardar información relativa al programa fuente.
- Incremental
- Identificador, cadenas de caracteres, tipo, posición en el repositorio de almacenamiento

Tabla de símbolos

```
{//Bloque1
      int x1; int y1;
      {//Bloque2
            int w2; bool y2; int z2;
            … w2 …; … x1…; … y2; … z2 …;
      }
       … w0 …; … x1…; … y1 …;
}
```

TEC

Tabla de símbolos



TEC

Lexemas

- Secuencia de caracteres significativos.

```
posicion=inicial+velocidad*60
```

- Lexemas:
  - `posicion`
  - `=`
  - `inicial`
  - `+`
  - `velocidad`
  - `*`
  - `60`

TEC

Tokens

- Para cada lexema se produce un token
  - [nombre,valor-atributo]

```
posicion = inicial + velocidad * 60
```

- Tokens:
  - [id,1]
  - [=]
  - [id,2]
  - [+]
  - [id,3]
  - [*]
  - [60]

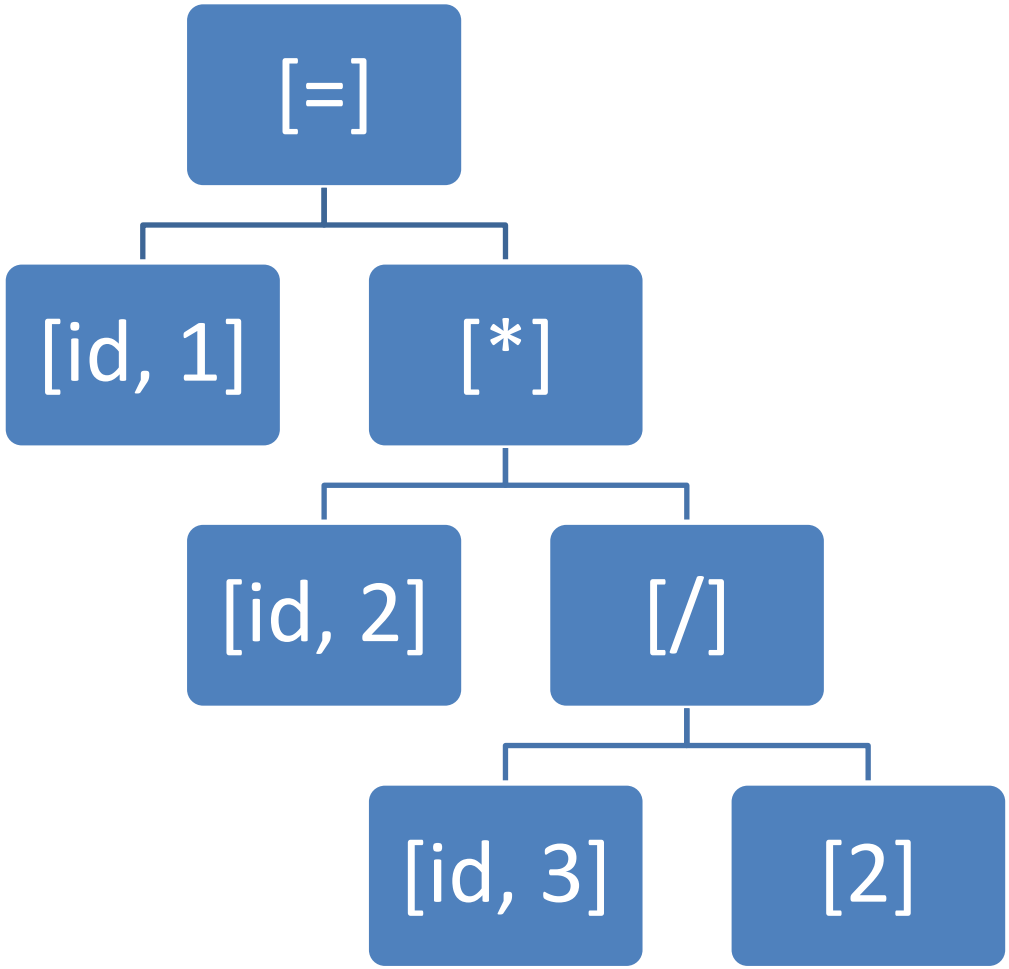
TEC

Árbol Sintáctico

- Representación intermedia que describe la estructura gramatical del flujo de tokens.
- Cada nodo interior representa una operación y los hijos representan los argumentos

TEC

Árbol Sintáctico



TEC

Código intermedio

- Secuencia de instrucciones con tres operandos por instrucción.
- Máximo un operador del lado derecho
- No se permiten expresiones acumuladas

TEC

Código intermdedio

```
x + y * z
```

```
t1 = y * z
t2 = x + t1
```

Máquinas Reales y Abstractas

TEC

Máquinas Reales y Abstractas

- Se utilizan máquinas reales para interpretar máquinas abstractas.
- Se tiene el concepto de una máquina nueva poderosa, construirla es oneroso.

Máquinas Reales y Abstractas

- Se construye un intérprete para la máquina abstracta.
- Ejemplo: diseño de nuevo microprocesador
  - Se define en términos de entradas, salidas y conjunto de operaciones permitidas

Máquinas Reales y Abstractas

- Con Máquinas abstractas se calcula la cantidad de recursos requeridos (tiempo, memoria) para las operaciones sin construir un sistema real

Máquinas Reales y Abstractas

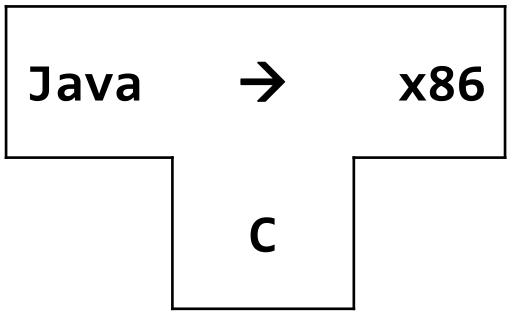
- Más barato
- Se utiliza emulador para desarrollarla
- Estimar recursos
- Mismo comportamiento que máquina real

Bootstrapping

TEC

Procesador de Lenguaje

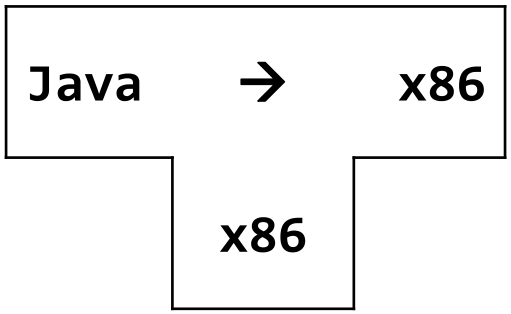
- Un compilador de **Java** a lenguaje **x86** expresado (escrito) en **C**



TEC

Procesador de Lenguaje

- Un compilador de **Java** a lenguaje **x86** expresado (escrito) en **x86**



Procesador de Lenguaje

- Un procesador de lenguaje es un programa que procesa programas expresados en un lenguaje en particular (lenguaje fuente). Además es **implementado en algún (otro) lenguaje**.

Bootstrapping

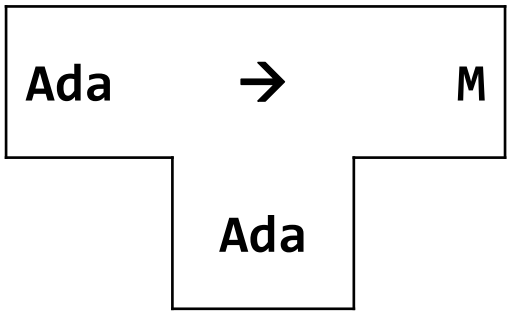
- Un procesador de lenguaje es un programa que procesa programas expresados en un lenguaje en particular (lenguaje fuente). Además es **implementado en sí mismo**.

- **El procesador de lenguaje es usado para procesarse a sí mismo.**

TEC

Full Bootstrapping

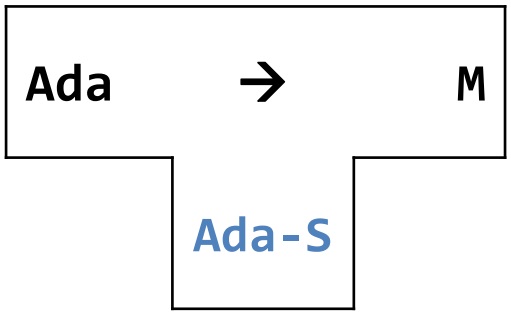
- Se requiere un compilador de **Ada** a lenguaje máquina **M** expresado (escrito) en **Ada**



TEC

Full Bootstrapping

- Se requiere un compilador de **Ada** a lenguaje máquina **M** expresado (escrito) en **Ada**

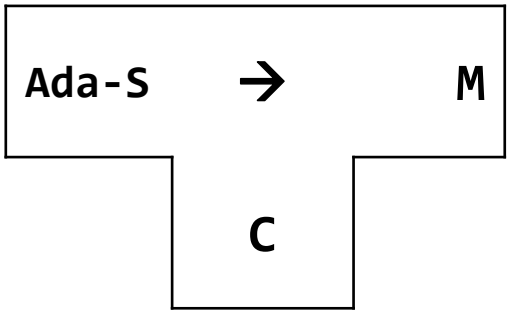


TEC

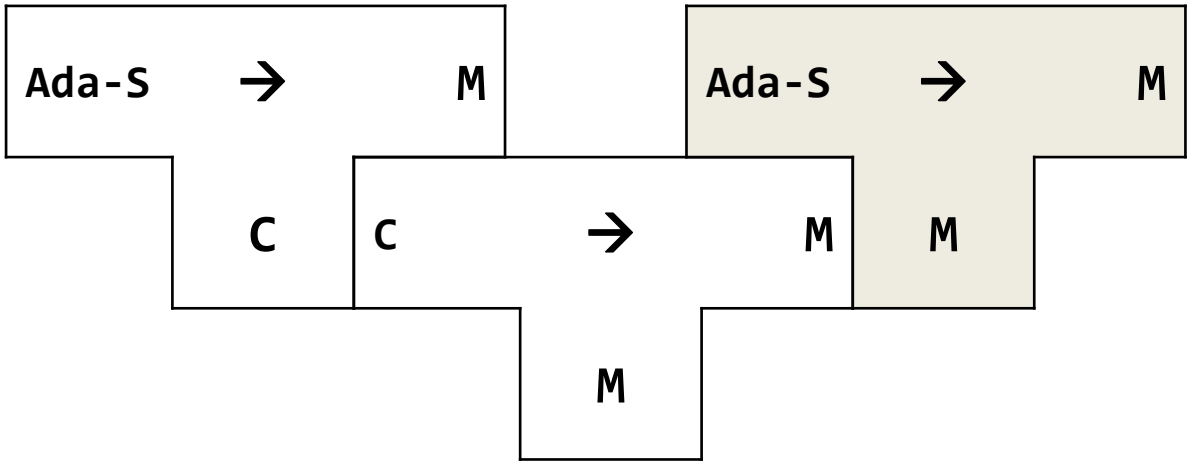
Full Bootstrapping

- Se requiere un compilador de **Ada** a lenguaje máquina **M** expresado (escrito) en **Ada**

Versión #1



Compilación

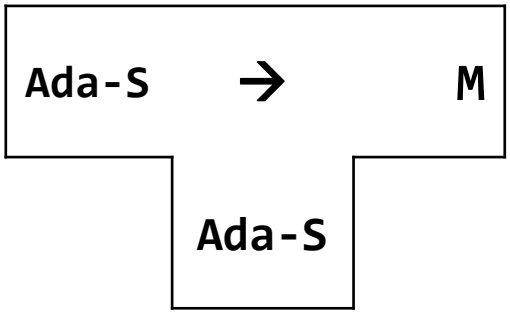


TEC

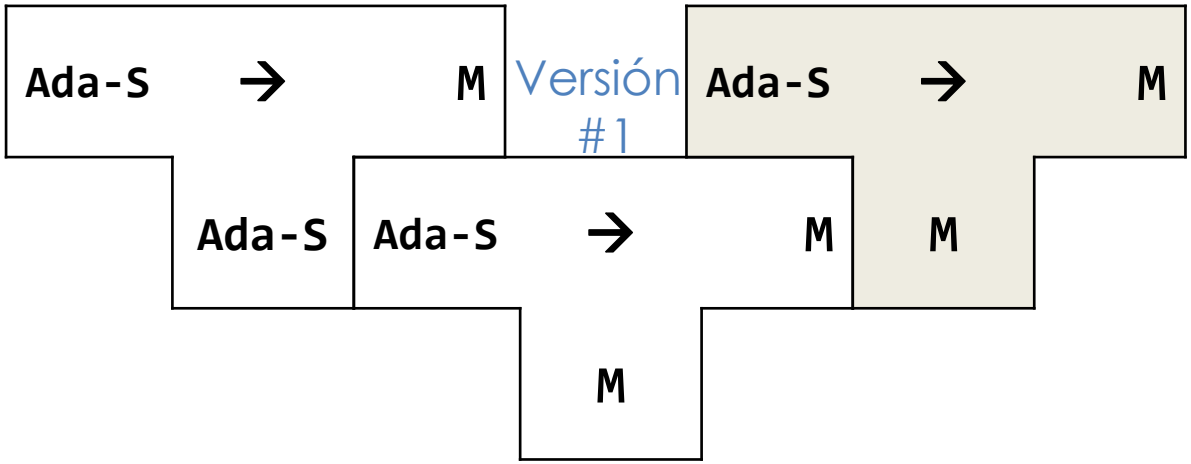
Full Bootstrapping

- Se requiere un compilador de **Ada** a lenguaje máquina **M** expresado (escrito) en **Ada**

Versión #2



Compilación

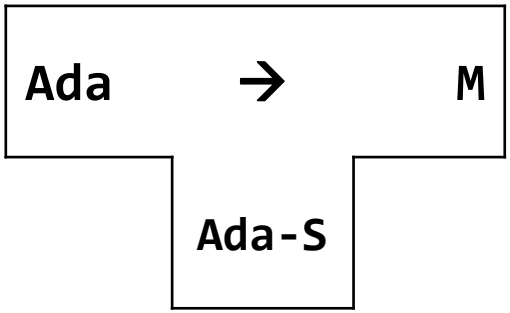


TEC

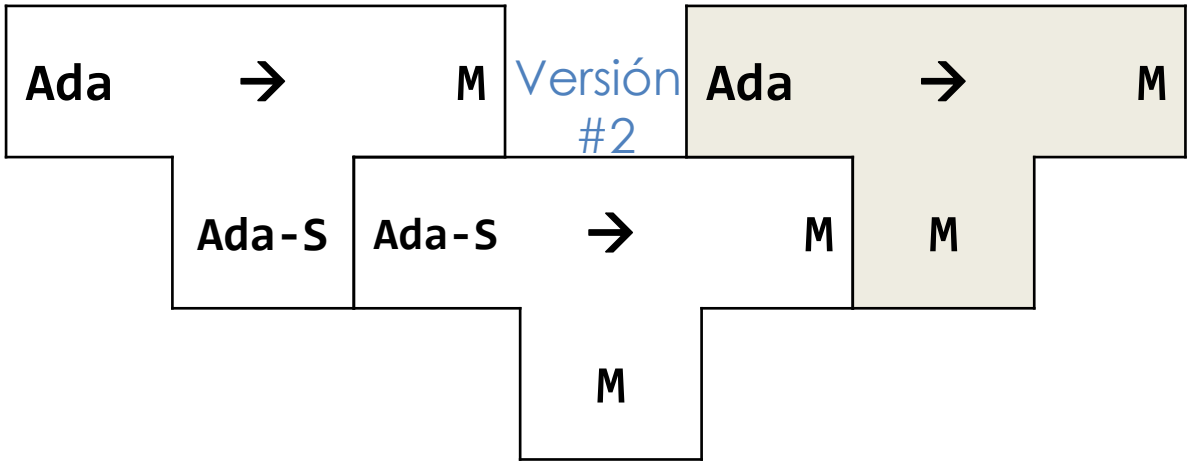
Full Bootstrapping

- Se requiere un compilador de **Ada** a lenguaje máquina **M** expresado (escrito) en **Ada**

Versión #3



Compilación

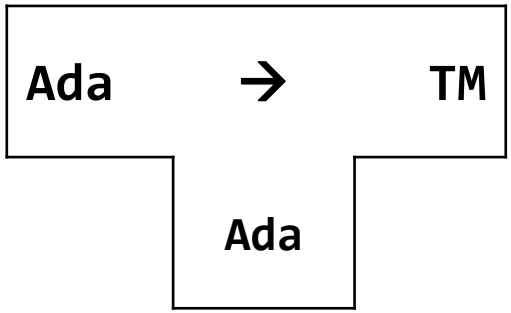


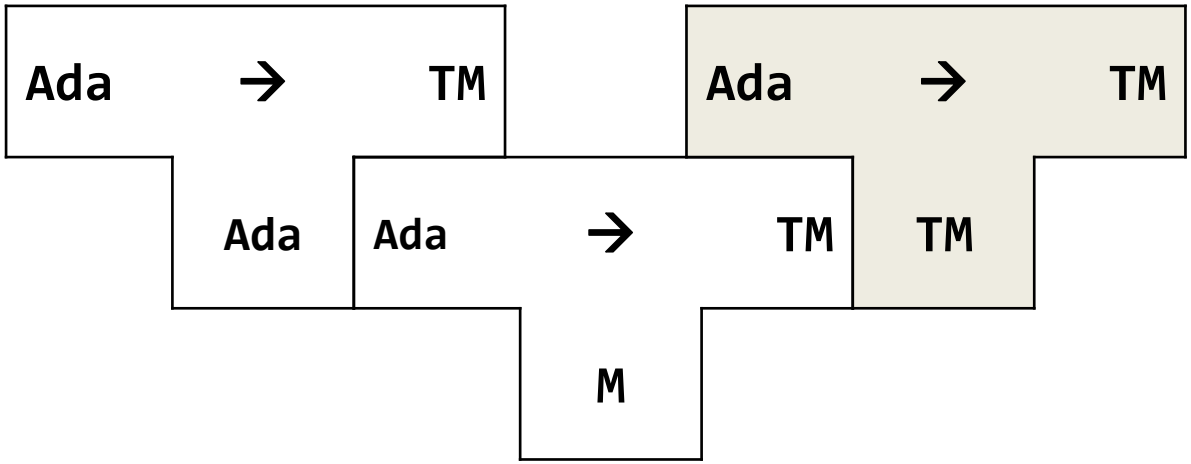
TEC

Half Bootstrapping

- Se requiere un compilador de **Ada** a lenguaje máquina **TM** expresado (escrito) en **Ada**

Compilación





Portafolio #1

- Dibuje los diagramas Tombstone necesarios para las siguientes especificaciones:
- Programa Parser expresado en C—
- Máquina x64
- Intérprete C-- expresado en x64

Portafolio #1

- Dibuje los diagramas Tombstone necesarios para lograr que un nuevo lenguaje C— logre Full Bootstrapping, asumiendo que inicialmente estaría escrito en Python
  - Diseñado para arquitectura x64

Portafolio #1

- Identifique los lexemas de los siguientes códigos:
  - `int x=10**3//4/-557.45-668.56>80?1:-1;`

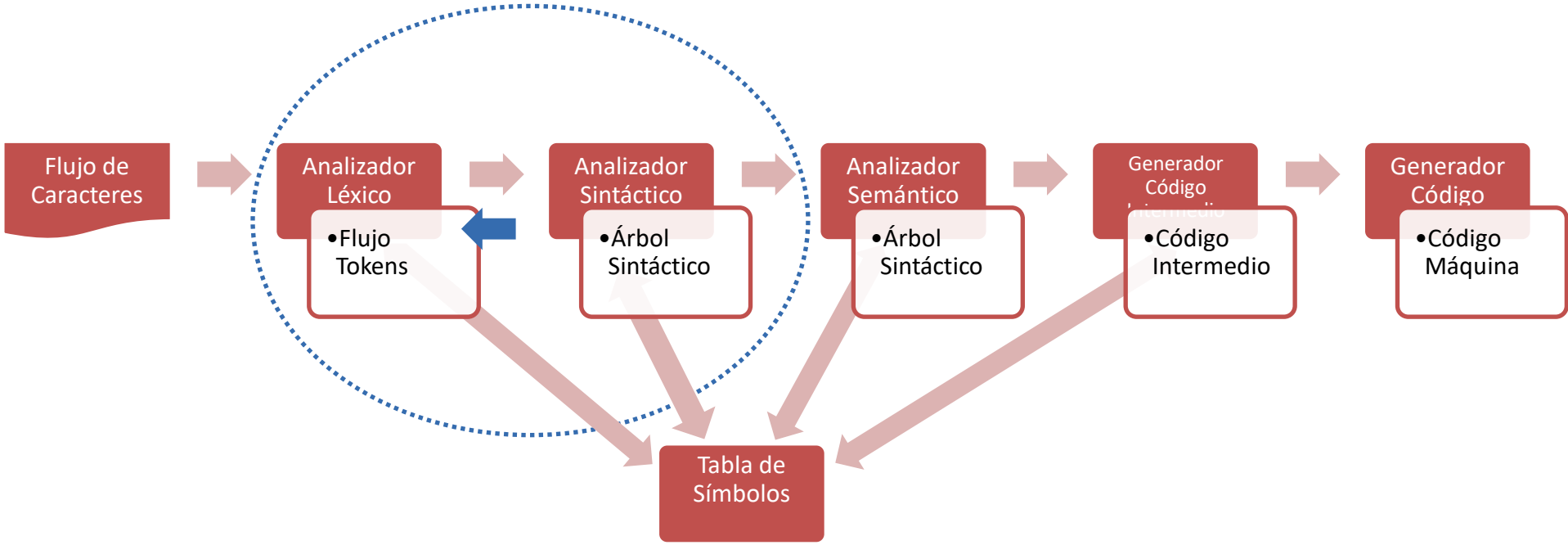
Portafolio #1

- Identifique los lexemas de los siguientes códigos:
  - ```
    def area_rectangulo(base,altura):
      area=base*altura;
      return area;
    ```
  - ```
    fun f_1(x) = let val x=7
    in
      (let val x=3 in x-1 end) + (let val y=x+4 in y+2 end)
    end
    ```

Proceso de compilación

Compilador

Compilación



Proceso de Compilación

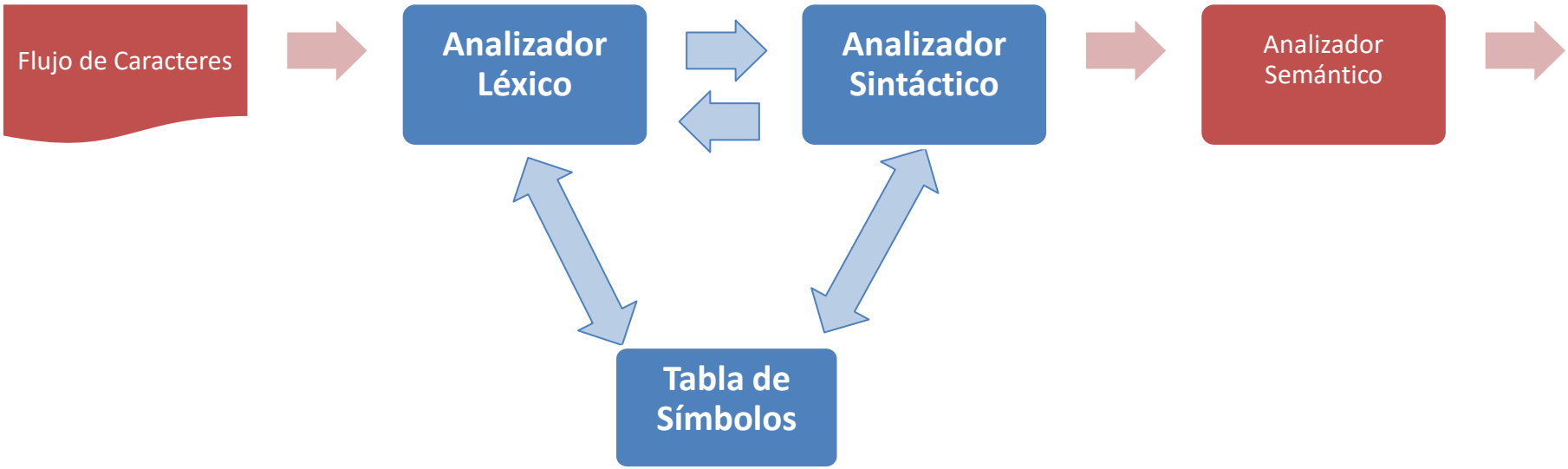
- Lectura (Portafolio)
  - Identificar las principales características de las diferentes fases del proceso de compilación:
    - Análisis léxico
    - Análisis sintáctico
    - Análisis semántico
    - Generación de código intermedio
    - Optimización de código
    - Generación de código
    - Administración de tabla de símbolos

Análisis Léxico

Análisis Léxico

- Primera fase del compilador
- Lee los caracteres del programa fuente
- Envía los tokens al analizador sintáctico
- Interactúa con la tabla de símbolos

Interacción



Análisis Léxico

- Lee el texto origen
- Elimina comentarios y espacios en blanco
- Correlaciona los mensajes de error con el programa fuente.
- Expansión de macros

Procesos

- Escaneo
  - Limpieza de código
- Análisis léxico
  - Secuencia de tokens

Tokens

- Par [nombre,valor-atributo]
- Nombre es un símbolo abstracto que representa un tipo de unidad léxica

```
printf(“Total = %d”,puntuacion);
```

- Tokens:
  - [printf] //tipo id
  - [“Total = %d”] //tipo literal
  - [id,1] //tipo id

Patrón

- Descripción de la forma de los lexemas
  - `else` → `e, l, s, e`
  - `posicion` → `relación de muchas cadenas`

Lexemas

- Secuencia de caracteres que coinciden con el patrón de un token.

```
printf(“Total = %d”,puntuacion);
```

- Lexemas:
  - `printf`
  - `(`
  - `“Total = %d”`
  - `,`
  - …

Tokens, Patrones y Lexemas

Token	Descripción informal	Lexemas ejemplo
`if`	Caracteres i, f	`if`
`else`	Caracteres e, l, s, e	`else`
`comparacion`	< o > o <= o >= o == o !=	`<=, !=`
`id`	Letra seguida por letras y dígitos	`pi, puntuacion, D2`
`numero`	Cualquier constante numérica	`3.1415, 0, 6.02e23`
`literal`	Cualquier cosa excepto “ rodeada por dos “’s	`“Hola Mundo”`

Atributos de los tokens

- Apuntador a la entrada en la tabla de símbolos del identificador
  - Cadena de caracteres
  - Tipo
  - Línea de primera aparición
  - Espacio de memoria requerido

Ejercicio

- Divida el siguiente código en lexemas

```
float cuadradoLimitado(float x)
{
    /*devuelve x al cuadrado, pero nunca más de 100*/
    float limiteInferior = -10.0;
    float limiteSuperior = 10.0;
    return (x<=limiteInferior||x>=limiteSuperior)?100:x*x;
}
```

- Qué lexemas deberían tener valores léxicos asociados?

Ejercicio

- Divida el siguiente código en lexemas

```
float cuadradoLimitado(float x)
{
    /*devuelve x al cuadrado, pero nunca más de 100*/
    float limiteInferior = -10.0;
    float limiteSuperior = 10.0;
    return (x<=limiteInferior||x>=limiteSuperior)?100:x*x;
}
```

- Conviértalos a tokens!

Ejercicio

- Transcriba el código

```
float cuadradoLimitado(float x)
{
    /*devuelve x al cuadrado, pero nunca más de 100*/
    float limiteInferior = -10.0;
    float limiteSuperior = 10.0;
    return (x<=limiteInferior||x>=limiteSuperior)?100:x*x;
}
```

- Con tokens (Incluir tabla simbolos)

Ejercicio

- Divida el siguiente código en lexemas y convíertalos a tokens

```
/*Uso del while con ejemplo*/
 int x; int y;
 x=read();
 y=read();
 while(x!=y){
    if(y<=x){
        y=y-x;
    } else {
        x = x-y;
    }           }
 /*Salida de resultado*/
 write("Resultado: ");
 write(x);
```

Bibliografía

- Programming Language Processors in Java: compilers and interpreters. Watt, David, Brown, Deryck. Pearson Education. 2000

- Compilers: principles, techniques and tools (2da. ed.). Aho, Alfred. Pearson Education. 2007


TEC
Tecnológico
de Costa Rica