

# Compiladores

## Introdução

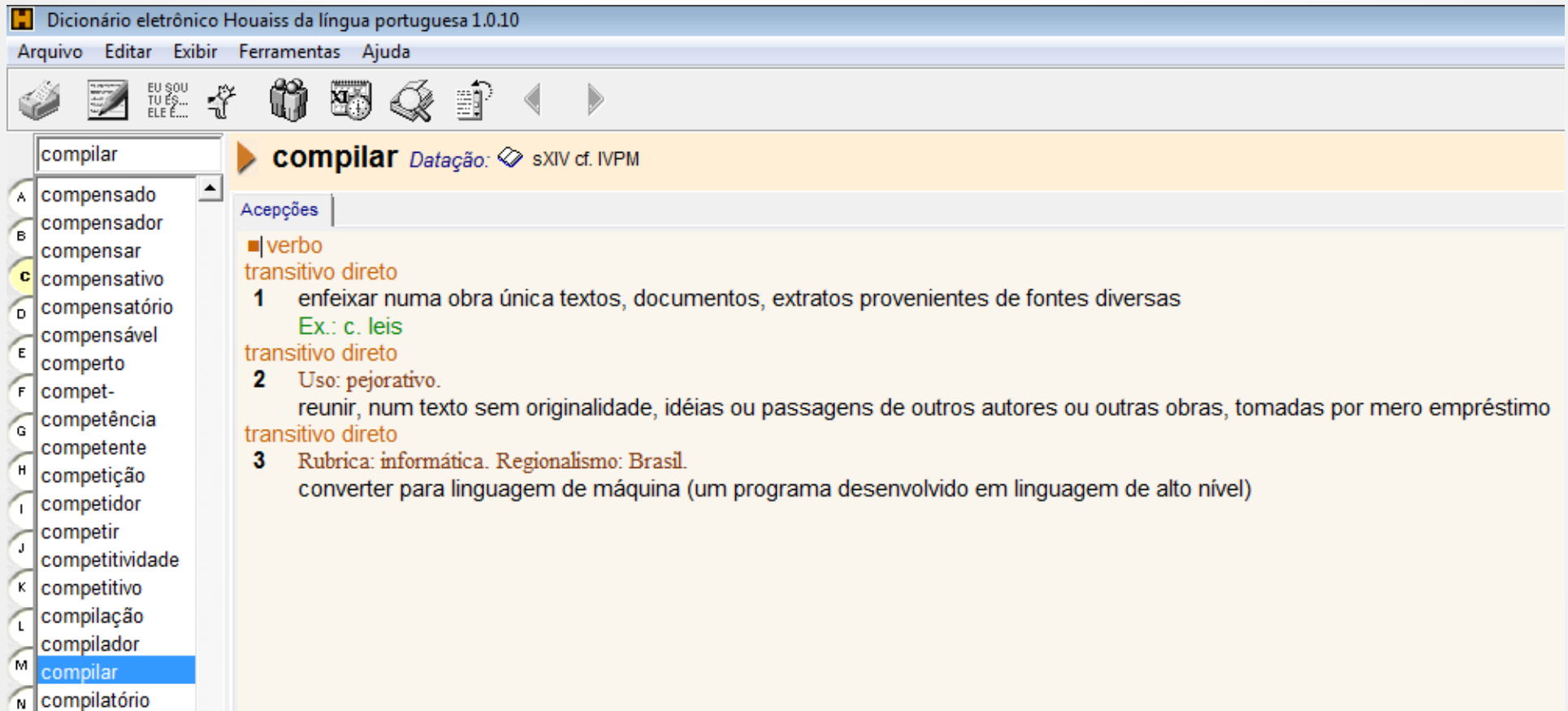
Profa Ana Carolina Lorena

1º semestre 2016



# Compilador

- O que significa a palavra **compilar** ?



The screenshot shows the interface of the 'Dicionário eletrônico Houaiss da língua portuguesa 1.0.10'. The search term 'compilar' is entered in the search bar. The left sidebar lists related words from 'compensado' to 'compilatório', with 'compilar' highlighted. The main area displays the definition for 'compilar', including its grammatical category (verb), transitivity (transitive direct), and three numbered meanings: 1. to compile texts/documents, 2. pejorative use for copying others' work, and 3. computer compilation (regionalism in Brazil).

Dicionário eletrônico Houaiss da língua portuguesa 1.0.10

Arquivo Editar Exibir Ferramentas Ajuda

compilar

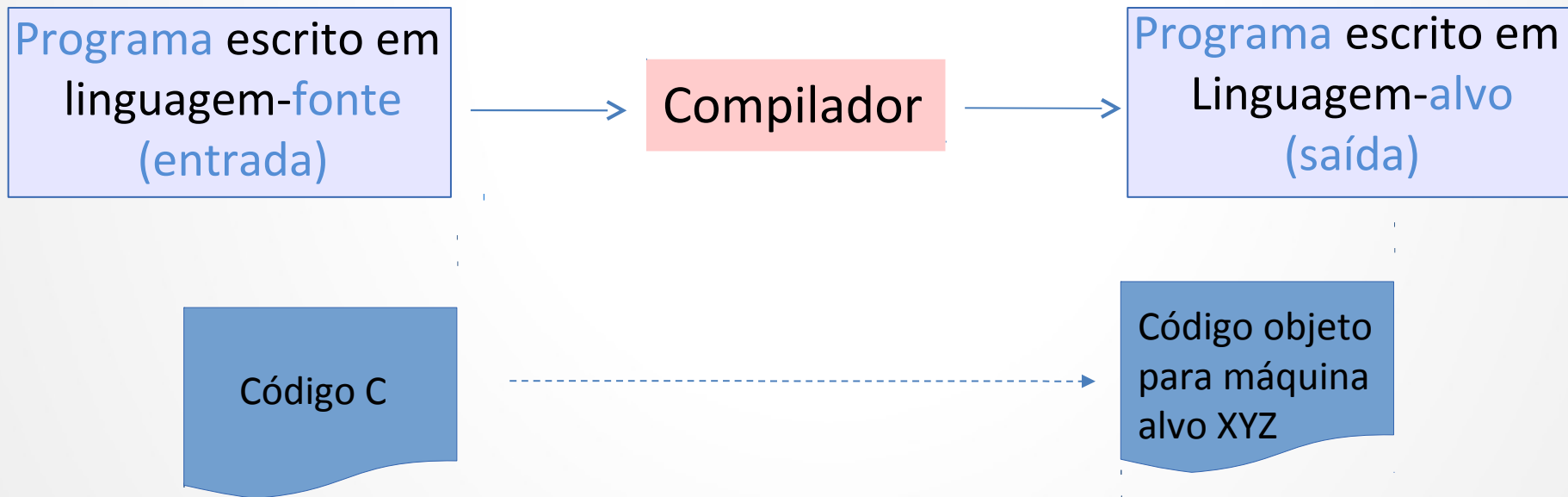
**compilar** *Datação:* sXIV cf. IVPM

Acepções

- verbo
- transitivo direto
- 1 enfeixar numa obra única textos, documentos, extratos provenientes de fontes diversas  
Ex.: c. leis
- transitivo direto
- 2 *Uso: pejorativo.*  
reunir, num texto sem originalidade, idéias ou passagens de outros autores ou outras obras, tomadas por mero empréstimo
- transitivo direto
- 3 *Rubrica: informática. Regionalismo: Brasil.*  
converter para linguagem de máquina (um programa desenvolvido em linguagem de alto nível)

# Compilador

- O que é um **compilador**?
  - É um programa de computador que **traduz** uma linguagem para outra



# Compilador

- **Importância** dos Compiladores
  - Sem eles, ainda estaríamos escrevendo programas em código numérico
    - Usando linguagem de máquina

1101 0110 1101  
1001 1101 0101

Ex: processadores 8x86 – IBM PC

C7 06 0000 0002 (código hexadecimal)



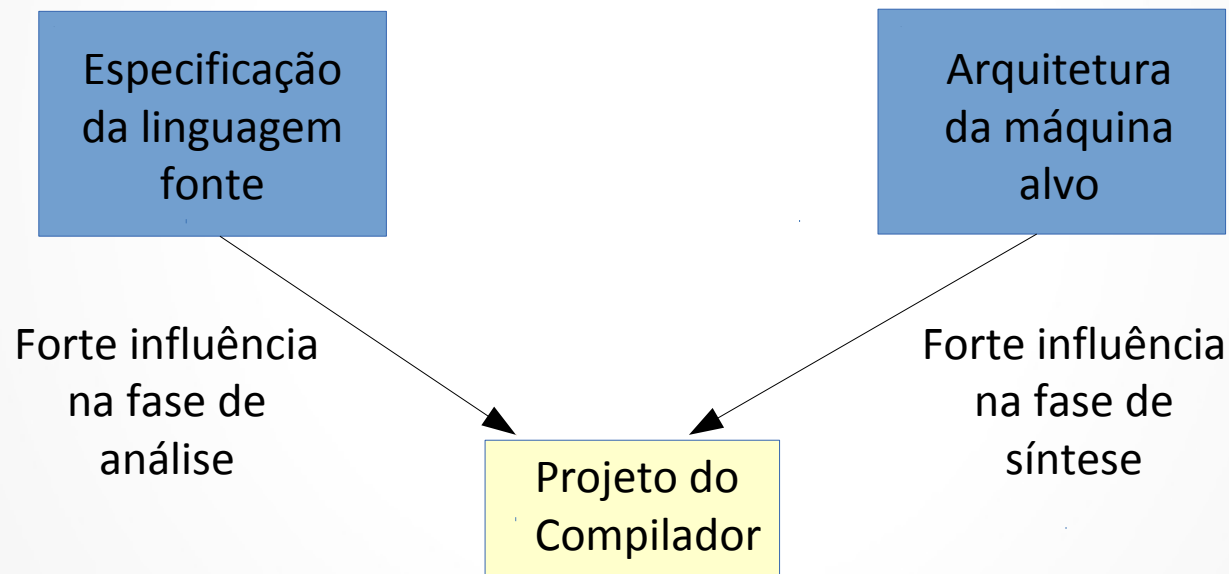
MOV x , 2 (código de montagem)



x = 2 (código em linguagem de alto nível)

# Compilador

- Principais fatores que influenciam o projeto de um compilador



# Histórico

- Primeiros compiladores
  - 1952: primeiro compilador (escrito por Grace Hopper), para a linguagem *A-0 System* (rodava no *UNIVAC I*)
  - 1957: primeiro compilador completo (projeto liderado por John W. Backus, na IBM), para a linguagem *FORTRAN*
  - 1960: surgimento de compiladores para múltiplas plataformas, inicialmente para a linguagem *COBOL*
  - 1962: primeiro *self-hosting compiler* (escrito com a própria linguagem que o compilador traduz), para a linguagem *LISP*
  - Da década de 70 em diante tornou-se comum implementar compiladores com a própria linguagem que ele compila -> Problema de *bootstrapping*

# Histórico

- Estudos de Chomsky tornaram construção de compiladores mais simples e parcialmente automatizável
  - Classifica linguagens segundo complexidade de suas gramáticas e poder dos algoritmos necessários para reconhecê-las
  - Estrutura de linguagens de programação de alto nível normalmente é normalmente representada por gramática do tipo 2 (livre de contexto)

# Programas relacionados

## Interpretador

- Interpreta o código-fonte e executa-o imediatamente, não gera código-objeto
- Compilador é preferível se velocidade de execução for importante

## Montador (*assembler*)

- Traduz para a linguagem de montagem (*assembly*) de um processador particular

## Organizador (*linker*)

- Coleta o código compilado separadamente e coloca tudo em um arquivo executável



# Programas relacionados

## Carregador

- Resolve os endereços relocáveis relativos a um dado endereço base ou inicial, torna o código executável mais flexível

## Pré-Processador

- Ativado pelo compilador antes do início da tradução, pode apagar comentários, incluir outros arquivos e executar substituições de macros

## Editor

- Oferece infraestrutura para escrever o programa fonte, gerando o arquivo a ser compilado, pode ser orientado pela estrutura ou formato da linguagem

# Programas relacionados

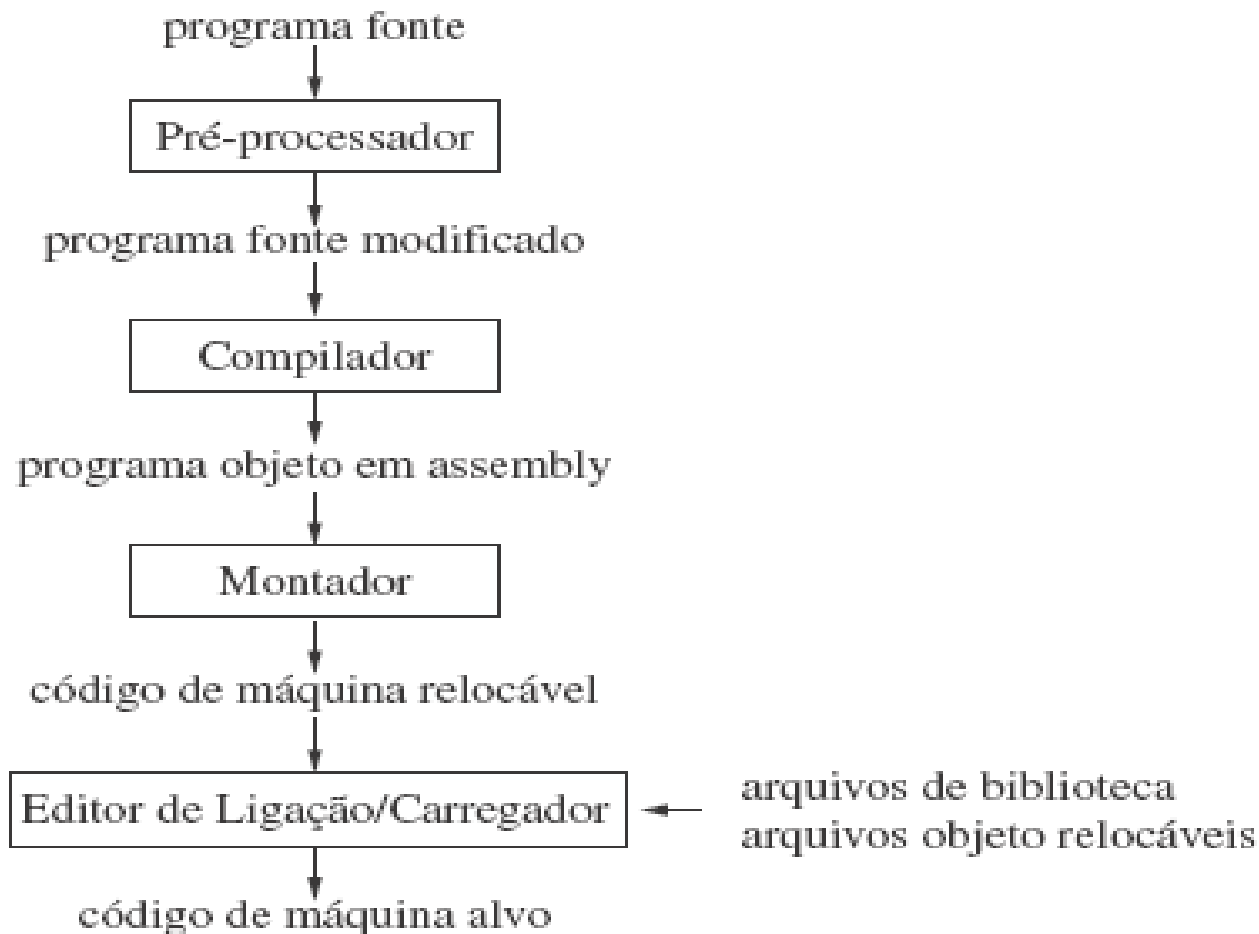
## Depurador

- Utilizado para determinar erros de execução em um programa compilado, costuma ser utilizado de forma integrada em um IDE (*Integrated Development Environment*)

## Gerador de Perfil

- Coleta estatísticas sobre o comportamento de um programa objeto durante sua execução

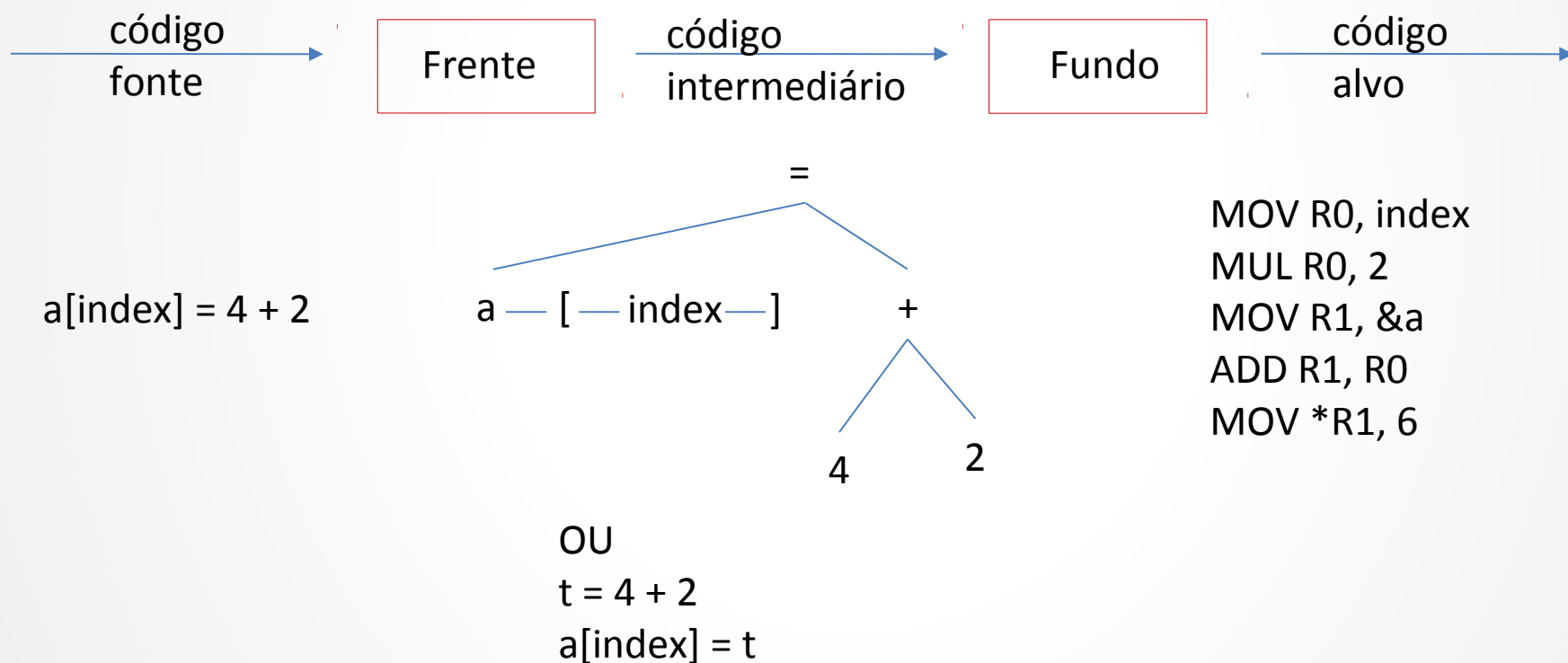
# Compilador



**FIGURA 1.5** Um sistema de processamento de linguagem.

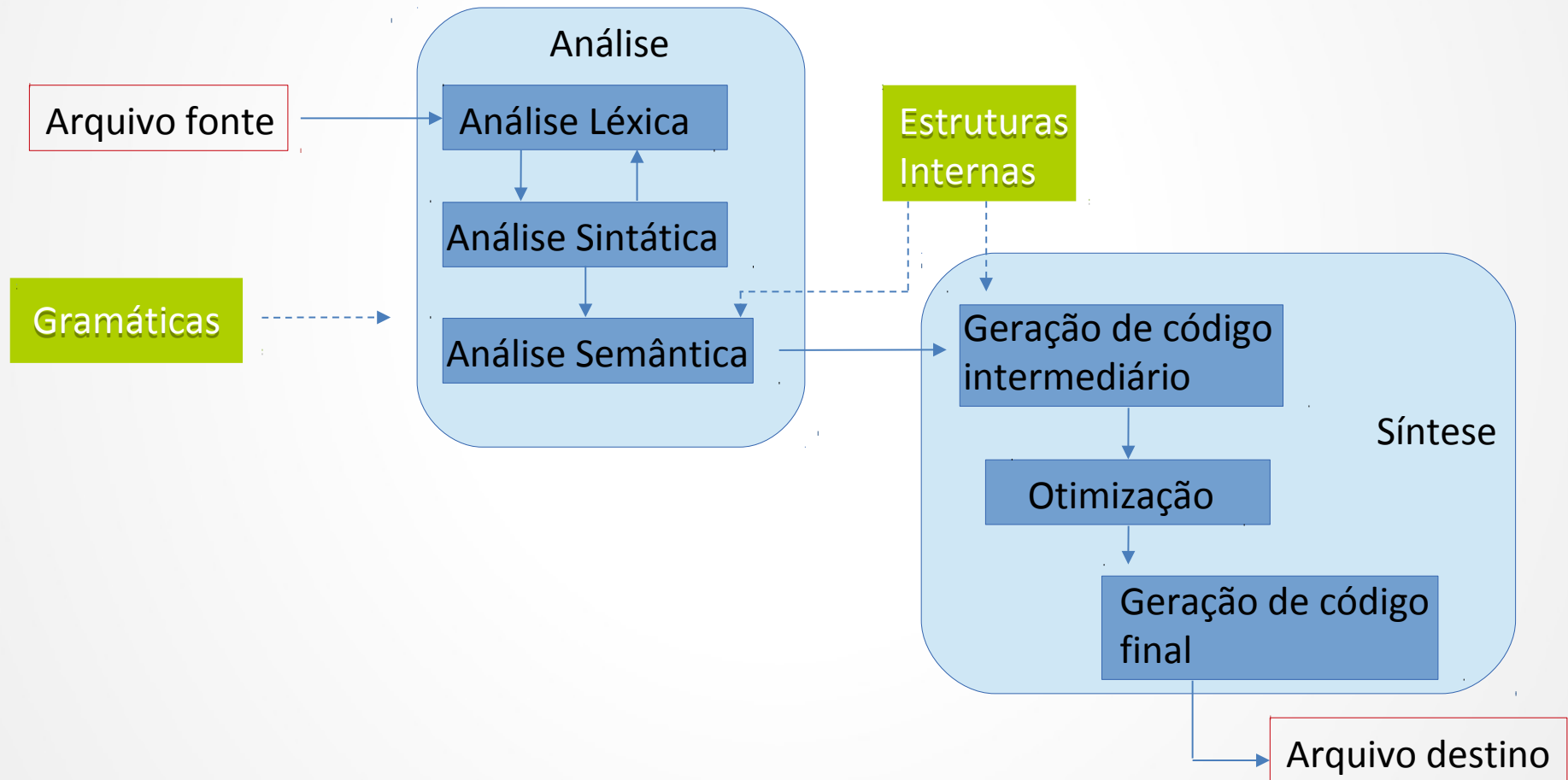
# Compilador

- Visão geral do processo de tradução



# Compilador

- Visão geral do processo de tradução



# Compilador

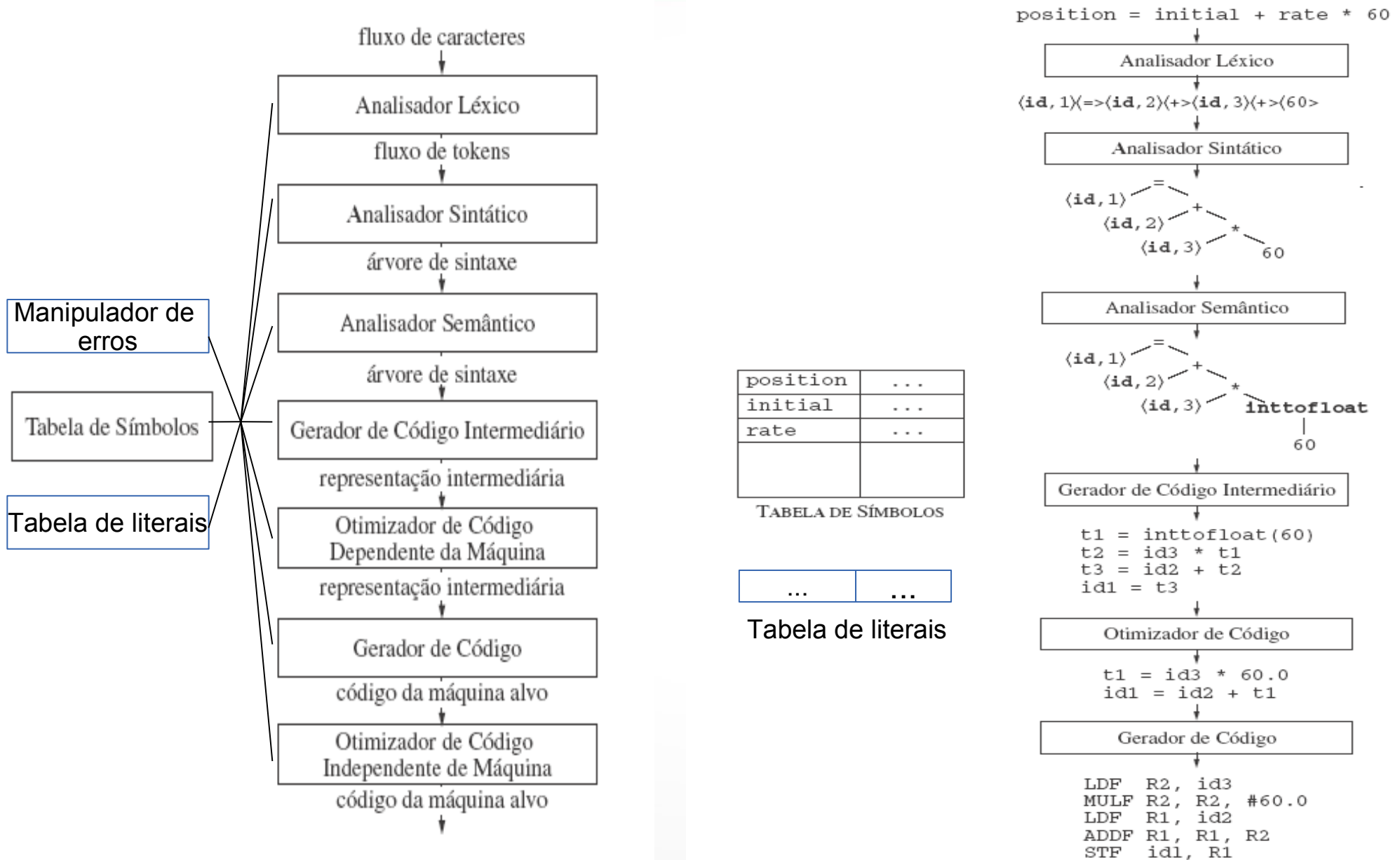


FIGURA 1.6 Fases de um compilador.

# Análise léxica

- Analisador léxico – sistema de varredura
  - Lê código fonte
  - Sequências de caracteres são organizadas como unidades significativas, chamadas lexemas ou *tokens* (como palavras)

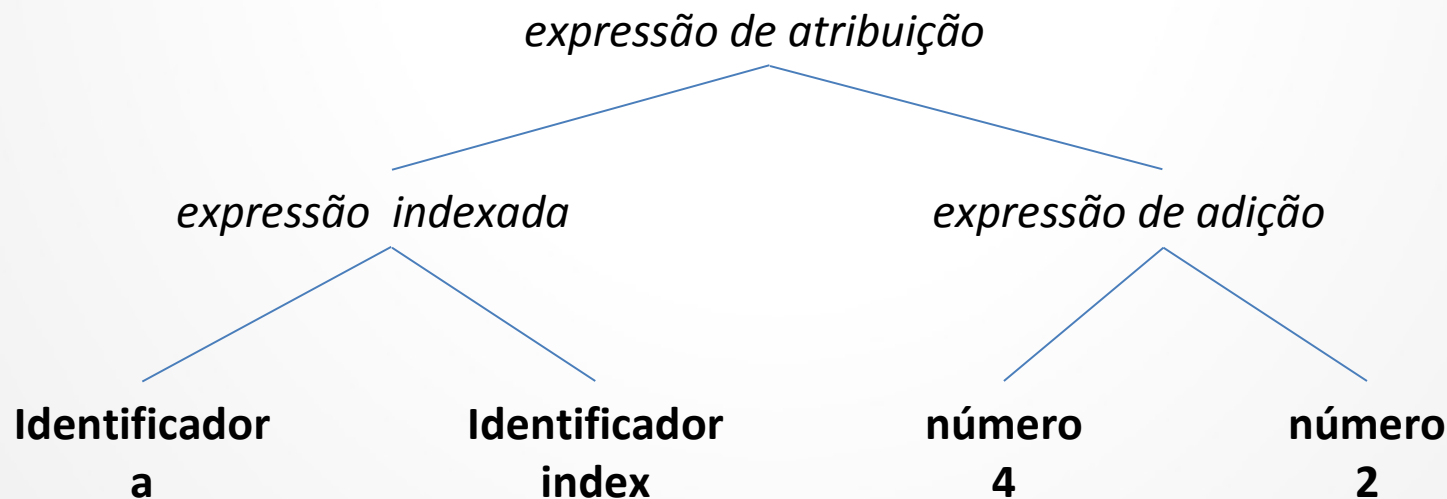
Exemplo:  $a[index] = 4 + 2$

Lexemas	<i>Tokens</i>
a	Identificador
[	Colchete à esquerda
index	Identificador
]	Colchete à direita
=	Atribuição
4	Número
+	Adição
2	Número

Além de reconhecer os *tokens*, o analisador léxico pode inserir identificadores na tabela de símbolos e literais na tabela de literais

# Análise sintática

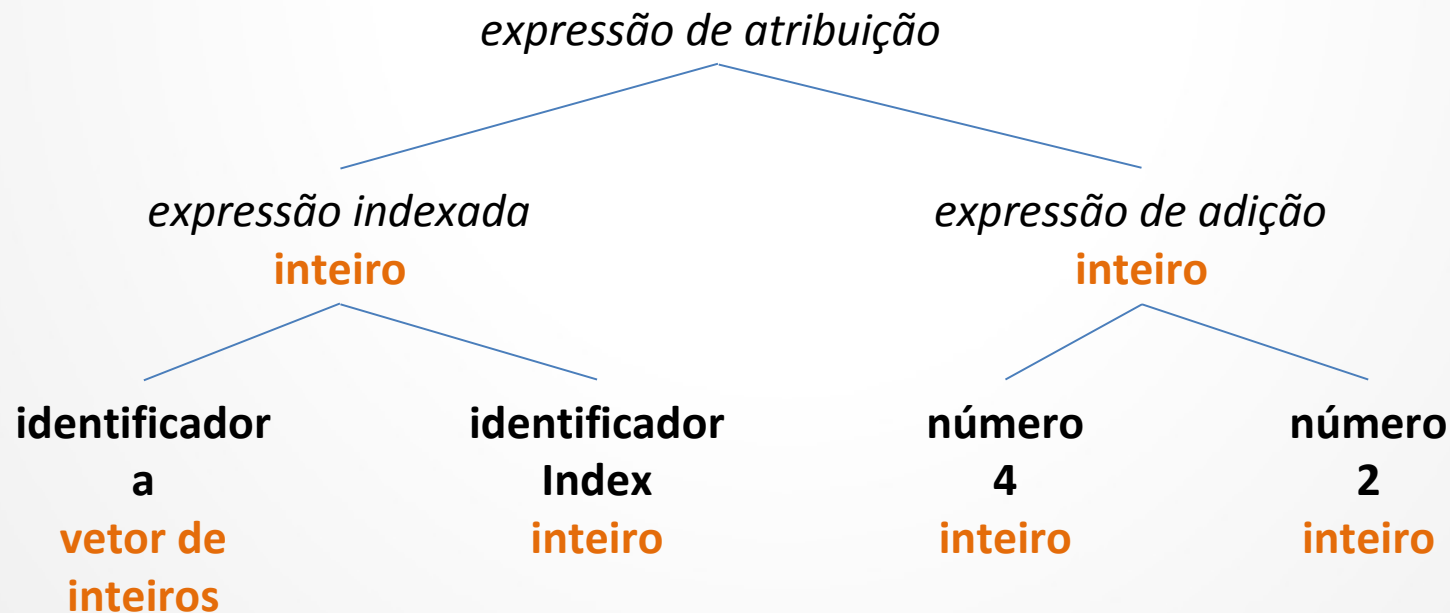
- **Analizador sintático – Parser**
  - Faz análise gramatical
  - Determina os elementos estruturais do programa e seus relacionamentos
  - Os resultados da análise sintática geralmente são representados como uma **árvore de análise sintática** ou uma **árvore sintática**





# Análise semântica

- **Analizador semântico**
  - A semântica de um programa é seu significado, contrastando com sintaxe ou estrutura
  - O analisador semântico faz a verificação de tipos e declarações (semântica estática), e frequentemente adiciona novas informações (atributos) na árvore sintática



# Análise

- A fase de análise

Análise léxica	Verifica se a <b>palavra</b> está bem formada
Análise sintática	Verifica se a <b>sentença</b> está bem formada
Análise semântica	Verifica se o <b>texto (análise de tipos)</b> está coerente

# Geração de código intermediário

- Gerador de Código Intermediário
  - Gera uma representação intermediária linearizada, próxima do código de montagem
  - Essa representação intermediária deve ser:
    - Facilmente produzida
    - Facilmente traduzida para a máquina alvo
  - Uma representação intermediária muito utilizada é o **código de três endereços**

Exemplo:  $a[\text{index}] = 4 + 2$

$t = 4 + 2$

$a[\text{index}] = t$

# Otimização de código intermediário

- Otimizador de Código Intermediário
  - Transforma o código intermediário com o objetivo de produzir um código objeto melhor
  - Código objeto melhor pode significar: um código mais rápido, menor ou que consuma menos energia

Exemplo: o código de três endereços abaixo

$t = 4 + 2$

$a[\text{index}] = t$

pode ser otimizado para

$t = 6$  (empacotamento constante)

$a[\text{index}] = t$

e depois para

$a[\text{index}] = 6$

# Geração de código alvo

- Gerador de Código Alvo

- A partir do código intermediário otimizado, gera o código para a máquina alvo
- Nessa fase as propriedades da máquina alvo se tornam o fator principal
  - Conjunto de instruções
  - Modos de representação de dados
  - Modos de endereçamento
  - Conjunto de registradores

Exemplo:       $a[\text{index}] = 6$

Linguagem de  
montagem  
hipotética



```
MOV R0, index    ;; move valor de index para R0
MUL R0, 2        ;; dobra o valor em R0 (inteiro ocupando 2 bytes)
MOV R1, &a       ;; move endereço de a para R1
ADD R1, R0       ;; adiciona R0 a R1
MOV *R1, 6       ;; move o valor 6 para o endereço apontado em R1
```

# Otimização de código alvo

- Otimizador de Código Alvo

- O compilador tenta melhorar o código-alvo gerado
- Melhorias mais comuns:
  - Escolher outros modos de endereçamento (melhora desempenho)
  - Substituições de instruções lentas por outras mais rápidas (eliminação de operações redundantes ou desnecessárias)

Exemplo:  $a[\text{index}] = 6$

MOV R0, index	{	MOV R0, index	;; move valor de index para R0
MUL R0, 2		SHL R0	;; instrução de deslocamento (dobra valor em R0)
MOV R1, &a		MOV &a[R0], 6	;; endereçamento indexado
ADD R1, R0			
MOV *R1, 6			

# Outros componentes

## Tabela de símbolos

- Guarda informações de identificadores (funções, variáveis, constantes e tipos de dados)
  - Normalmente implementada como tabela **Hash**, por ser muito acessada

## Tabela de literais

Armazena constantes e cadeias de caracteres usados em um programa

Também precisa ser implementada para acesso rápido

# Estruturas de dados

- *Tokens*: cadeia de caracteres
- *Árvore sintática*: árvore dinâmica
  - Nós com campos para armazenar informações (atributos)
- *Tabelas de símbolos e literais*: hashing
- *Código intermediário*: matriz de cadeias de caracteres ou lista ligada



# Compilador

- Passadas
  - É comum um compilador processar um programa fonte diversas vezes
    - Essas repetições são chamadas de **passadas**
  - A passada inicial
    - constrói a árvore sintática ou o código intermediário
  - Demais passadas
    - consistem em acrescentar informações, alterando a estrutura ou produzindo uma representação diferente

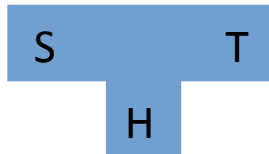
# Compilador

- Tratamento de erros
  - Como compilador responde a erros no programa fonte
  - Erros devem ser reportados por mensagens inteligíveis
  - E compilador deve concluir a compilação mesmo frente ao erro
    - “Se recuperar” do erro

# Compilador

- Compilador do Compilador
  - Como criamos o compilador do compilador ?
    - Abordagem 1: escrever o compilador em linguagem diferente da qual ele deve compilar (para qual já exista um compilador executável)

T-diagrama



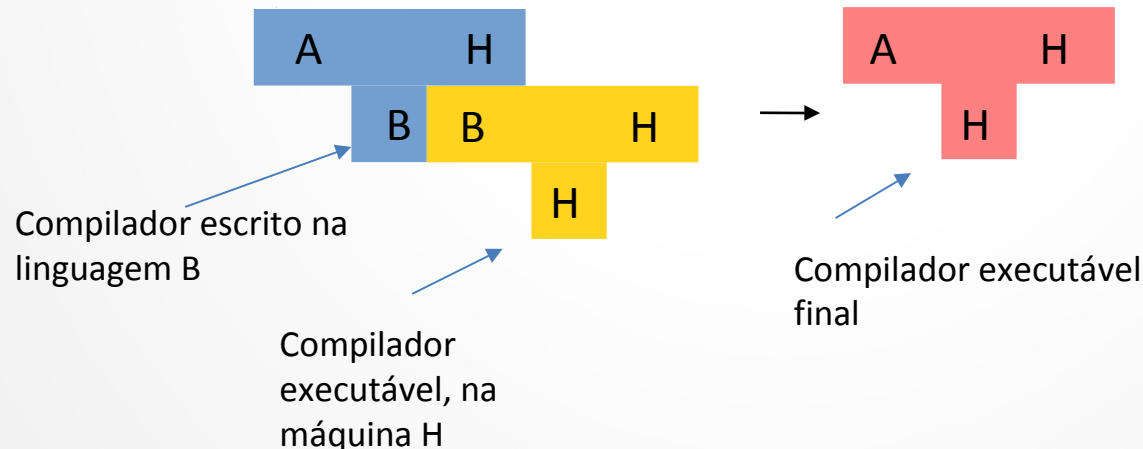
S = linguagem fonte

T = linguagem alvo

H = linguagem hospedeira

(compilador escrito usando ling H, que traduz programa em ling S para ling T)

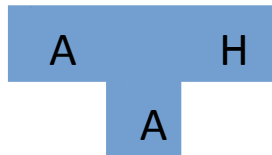
Exemplo



# Compilador

- Compilador do Compilador
  - Como criamos o compilador do compilador ?
    - Abordagem 2: escrever o compilador na mesma linguagem que ele deve compilar (para qual não exista um compilador executável)

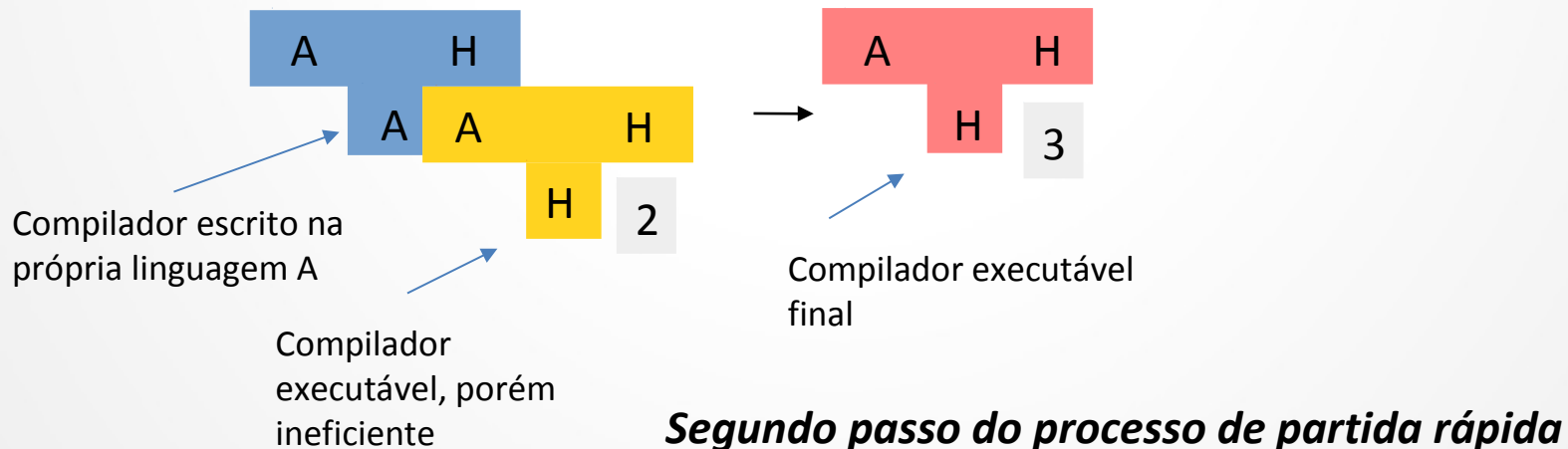
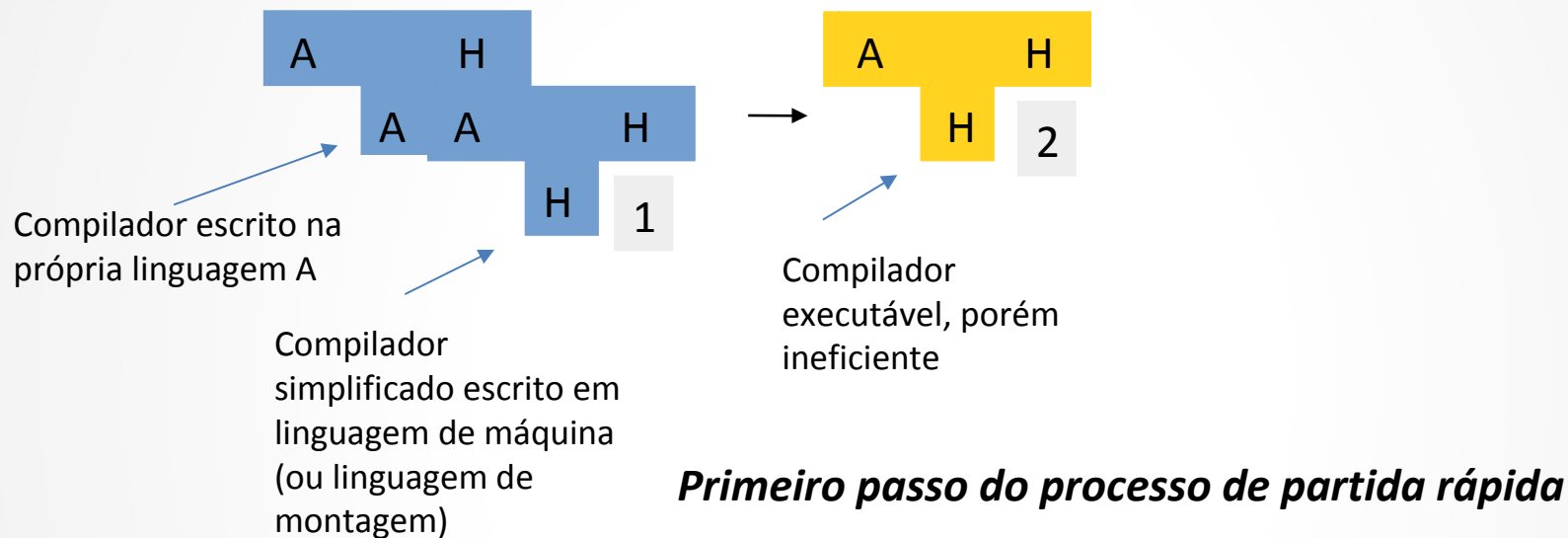
Exemplo



- Processo de **partida rápida**

# Compilador

- Partida Rápida (*Bootstrapping*)



# Linguagens exemplos

- TINY: exemplos em aula
  - Variáveis são inteiros
  - Usa if e repeat como estruturas de controle
  - Comentários entre chaves
  - Aritmética de inteiros (+, -, \* e / inteira)
  - Comparações booleanas (< e =)

# Linguagens exemplos

- C-: linguagem a ser usada no projeto
  - Admite inteiros, matrizes de inteiros e funções
  - Usa if e while como estruturas de controle
  - Especificação feita em LFA

# Linguagens exemplos

- Exemplo fatorial:

```
{TINY}
read x; {inteiro de entrada}
if x > 0 then
    fact := 1;
    repeat
        fact := fact * x;
        x := x - 1;
    until x = 0;
    write fact;
end
```

```
/* C- */
int fact (int x)
{
    if (x>1)
        return x * fact(x-1);
    else
        return 1;
}

void main(void)
{
    int x;
    x = read();
    if (x>0) write (fact(x));
}
```



# Bibliografia

**Capítulo 1** LOUDEN, Kenneth C; SILVA, Flávio S.C.  
Compiladores: princípios e práticas. São Paulo:  
Thomson, 2004. 569 p.

Slides:

- Prof Dr Luiz Eduardo G. Martins