UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

AULA 01 - INTRODUÇÃO À COMPILAÇÃO DE PROGRAMAS

MATA61 – COMPILADORES

- Qual é o **tema** deste curso?

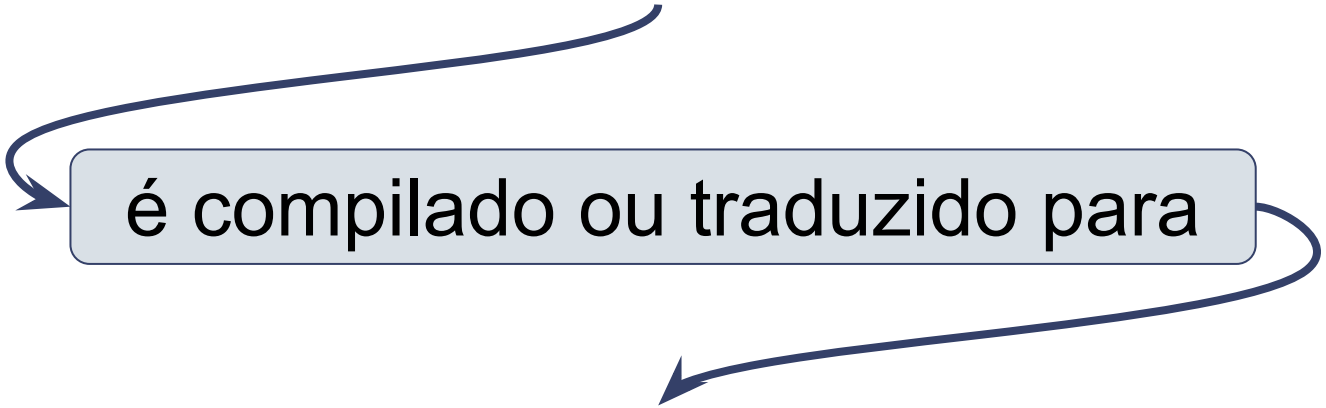
Este curso é sobre implementação de compiladores para linguagens de programação.

- O que é um compilador?

Um **compilador** é um software de tradução.

programa em uma linguagem "fonte"

(*source language*)



programa em uma linguagem "alvo"

(*target language*)

- Por que compiladores são necessários?

- Por que linguagens de programação são necessárias?

- Por que linguagens de programação são necessárias?

HUMANO

- Linguagem natural
- Abstrações
- Operações abstratas

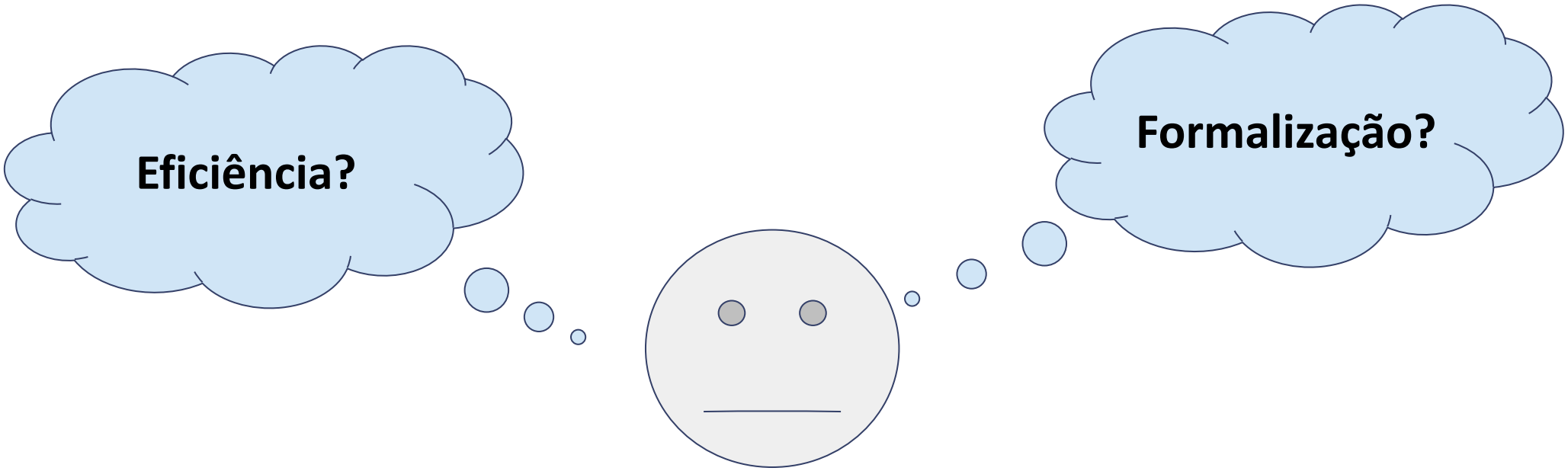
COMPUTADOR

- Linguagem de máquina
- Registradores, memória, ...
- Operações primitivas

+ Nível de abstração -

Tradutores automáticos são necessários

- **Compiladores**, interpretadores, montadores (*assemblers*)



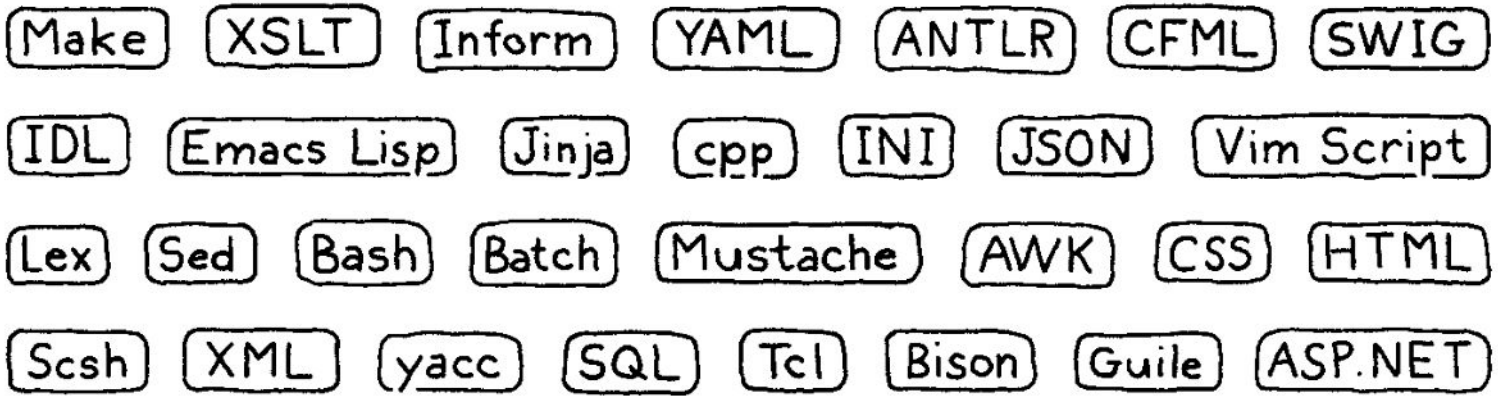
Formalização de Linguagens de Programação

Pré-requisito: **Linguagens Formais e Autômatos**

	LFA	Tópicos
1. Linguagens	✓	• regulares • livres de contexto
2. Gramáticas	✓	• regulares • livres de contexto
3. Reconhecedores	✓	• autômatos finitos • autômatos de pilha

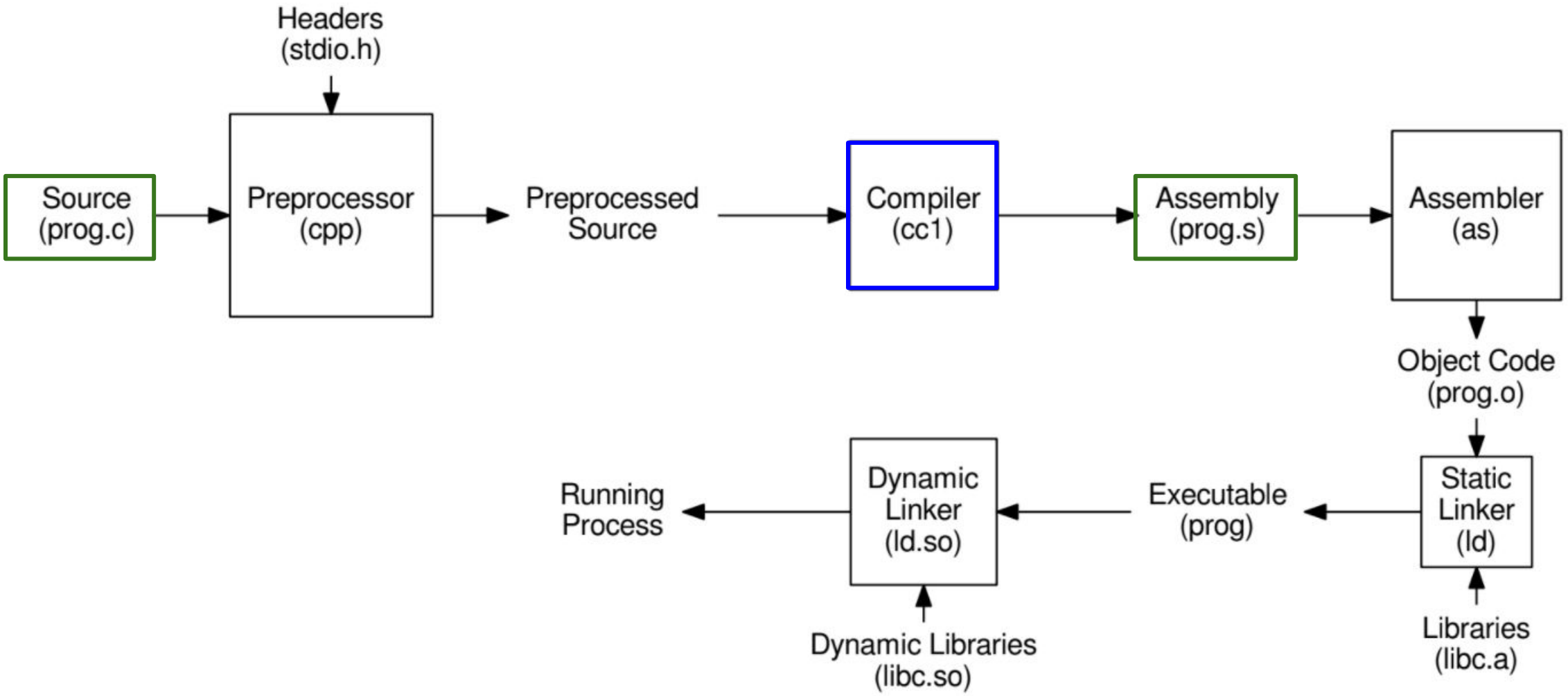
- Por que estudar (implementação de) compiladores?

- Compreender o processo de compilação?
- Melhorar habilidades de programação?
- Desenvolver software do "mundo real"?
- Projetar **novas linguagens**?



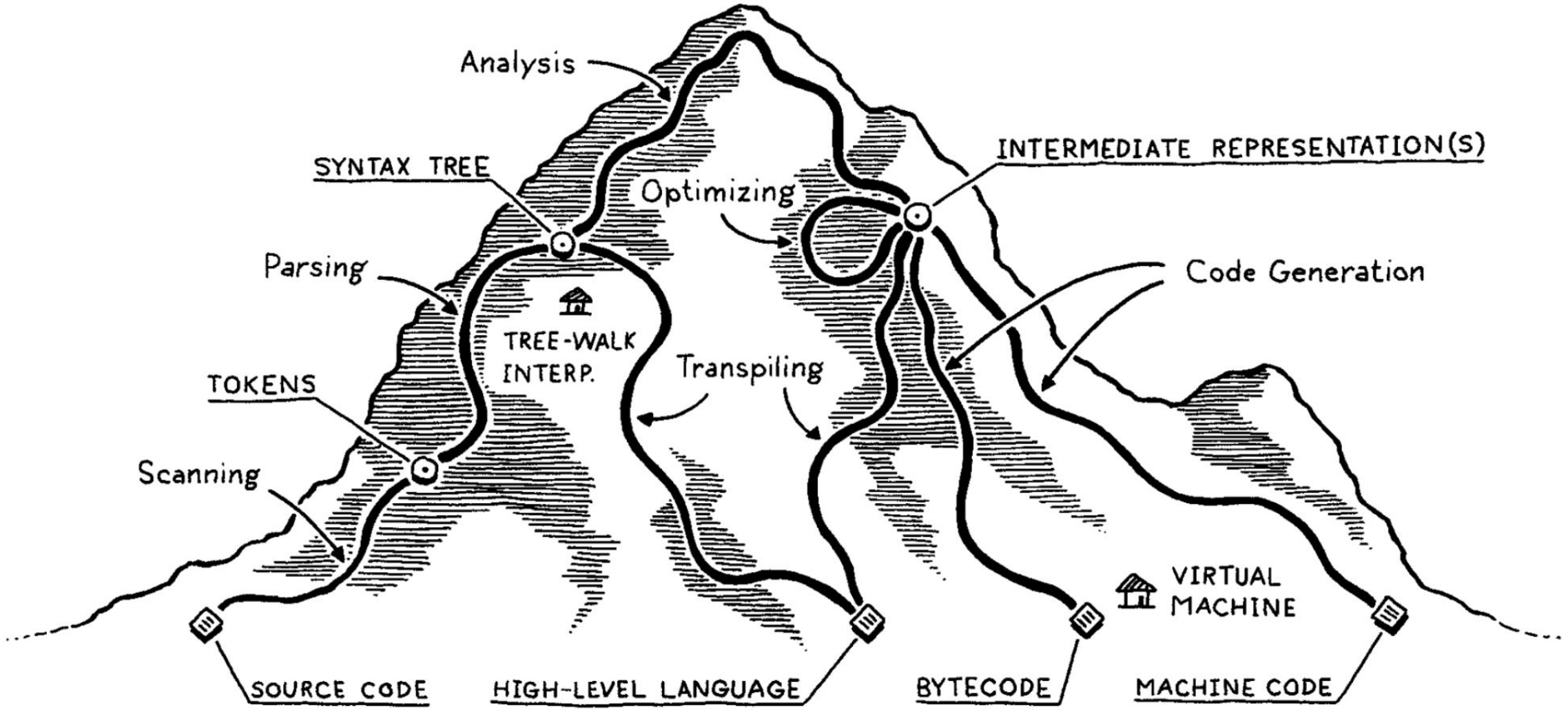
Fonte: Crafting Interpreters. Copyright © 2021 by Robert Nystrom.

Contexto de compilação de programas em C



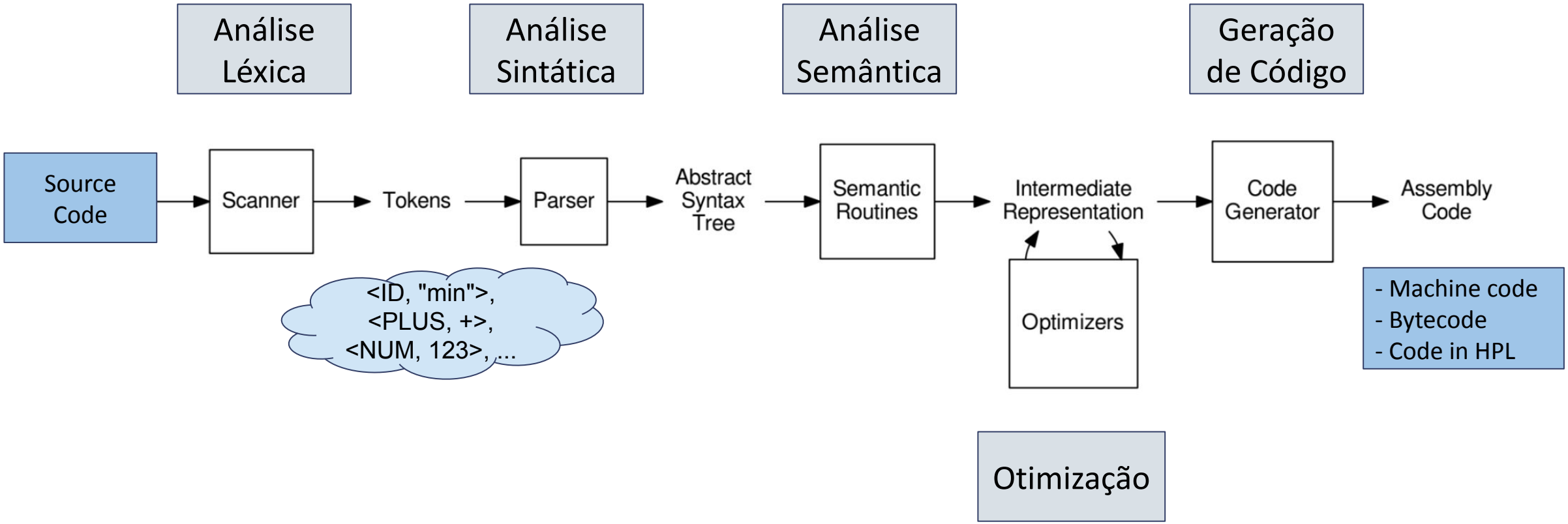
Compilador C

Processo de Compilação



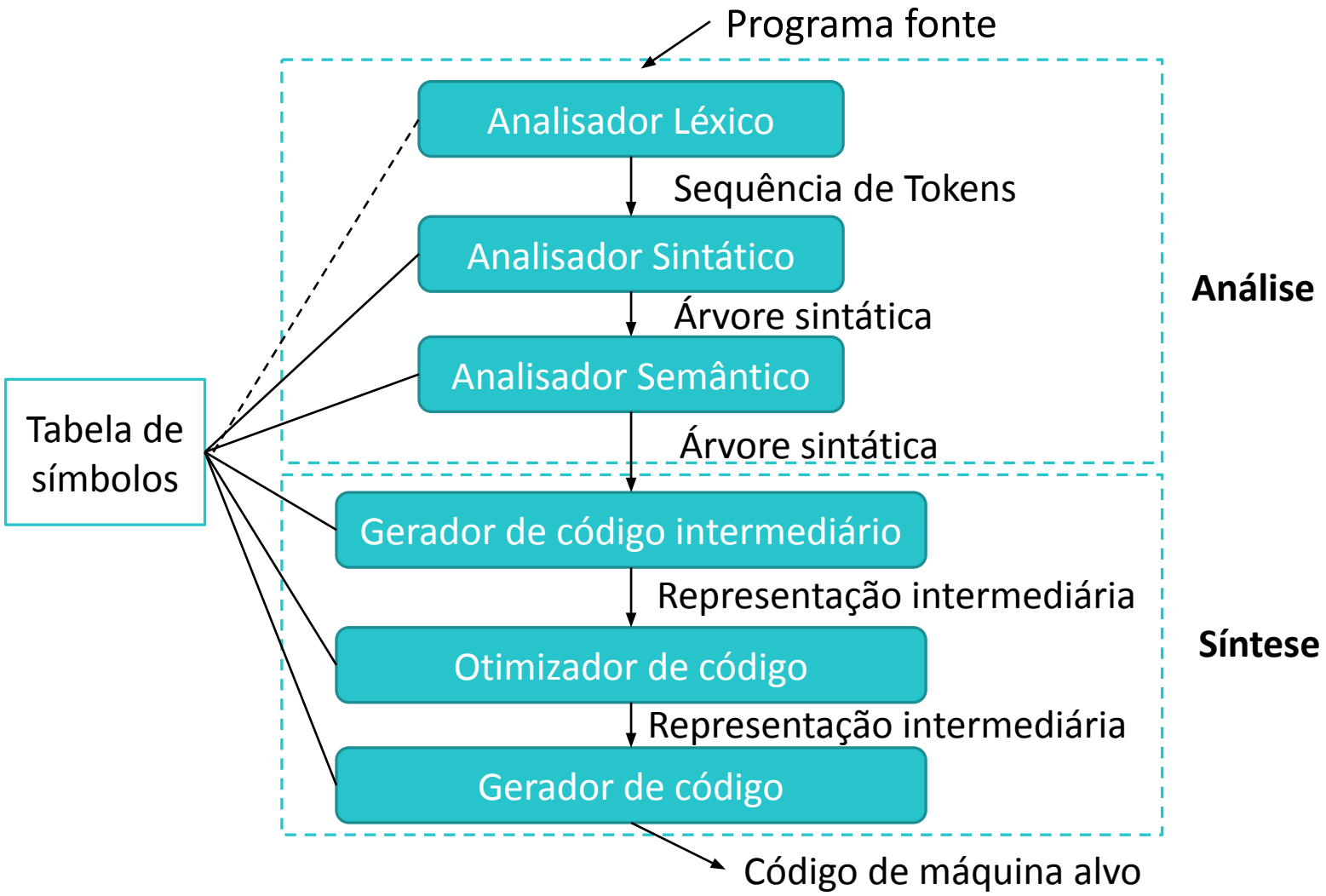
Fonte: Crafting Interpreters. Copyright © 2021 by Robert Nystrom.

Processo de Compilação



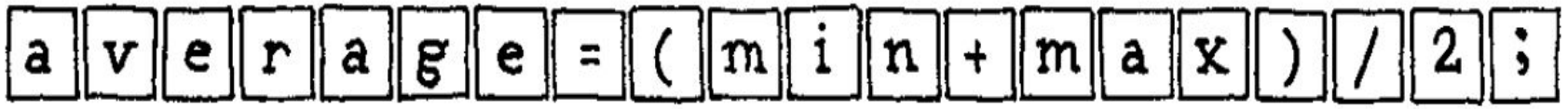
Processo de Compilação

- **Análise** e **Síntese**
- ***Frontend*** e ***backend***
- **Transformações sucessivas** entre representações do programa fonte.
- Atividades transversais:
  - **Gerência de tabela de símbolos**
  - **Tratamento de erros**



Análise Léxica

- Sequência de caracteres (programa-fonte)



- Sequência de **tokens (categoria e lexema)**

categoria	id	assign	lpar	id	plus	id	rpar	div	num	semic
lexema	average	=	(	min		max	)	/	2	;

Exercício 1

- Sequência de caracteres (programa-fonte)

  x := y + z * 100;

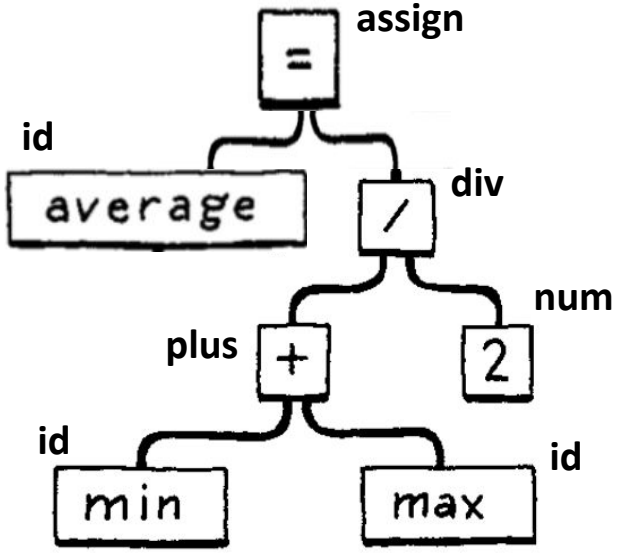
- Sequência de **tokens** (categoria, lexema)

id	atribui	id	soma	id	mult	num	pt_virg

Análise Sintática

- Sequência de tokens

- Árvore sintática abstrata



Exercício 2

- Sequência de caracteres (programa-fonte)

  x := y + z * 100;

- Sequência de tokens

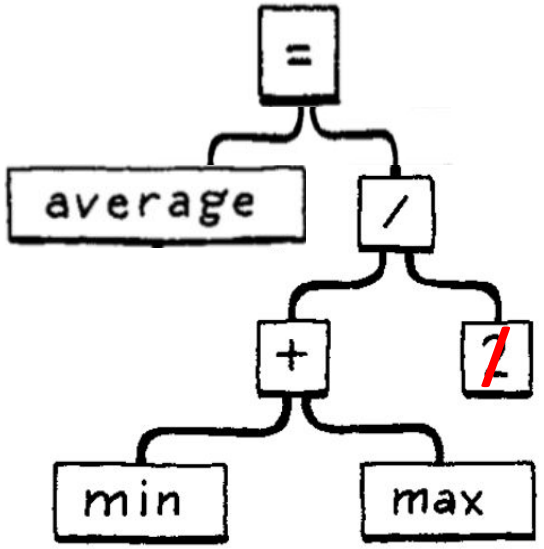
id	atribui	id	soma	id	mult	num	pt_virg
x	:=	y		z		100	

- Árvore sintática abstrata

  ?

Análise Semântica

- Semântica ~ **significado** de sentenças sintaticamente corretas
- Análise semântica
  - Verificação de tipos



Supondo que as variáveis foram declaradas como float:

int2float
|
2

Exercício 3

- Considere que apenas as variáveis x e y usadas no Exercício 1 foram declaradas como "float". Modificar, se necessário, a árvore sintática abstrata do Exercício 2 para conversão de valores do tipo "int" para "float".
- Considere que x e y foram declaradas como "int" e apenas z foi declarada como "float". Como seria a árvore?

Representação Intermediária

Uma representação intermediária (**IR**) serve como uma "interface" entre linguagens e plataformas.

Uma **IR** deve:

- ser independente de linguagem fonte e de linguagem objeto;
- facilitar otimização, análise e geração de código eficientes.

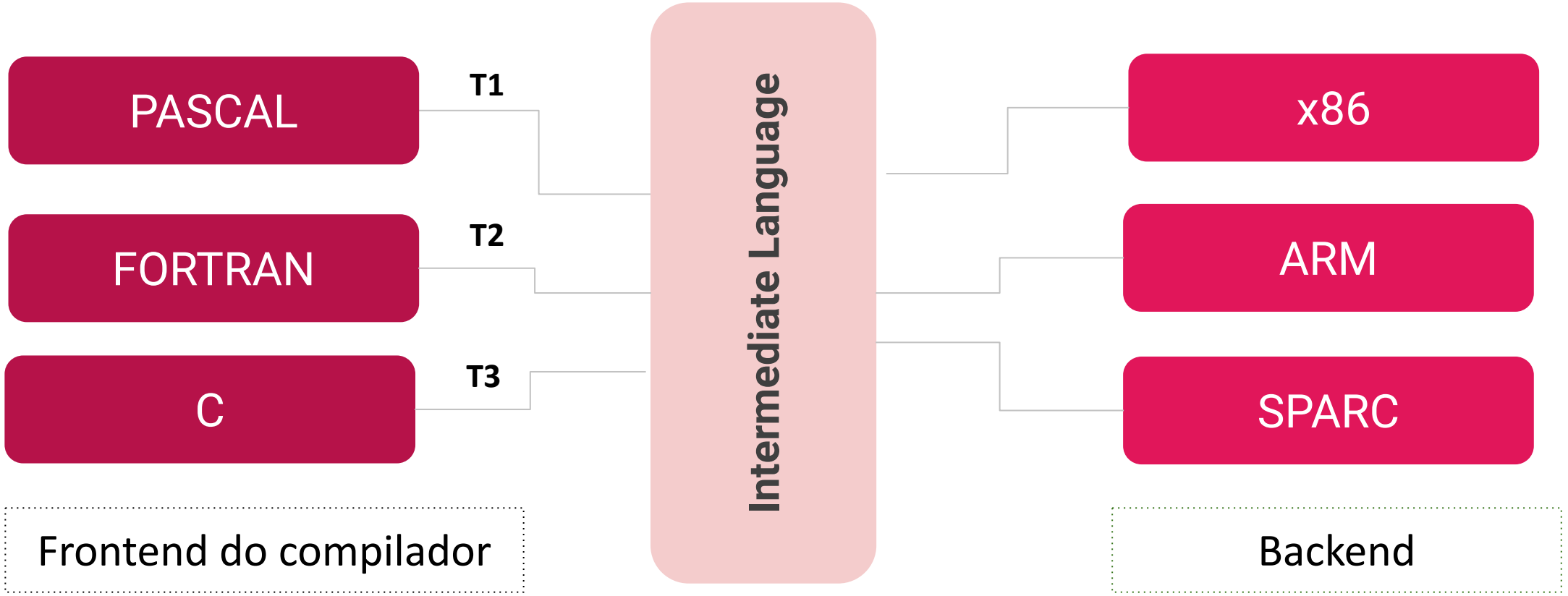
Three-address code (TAC ou 3AC)

average = (min + max) / 2;

temp1 := <id,*min*> + <id,*max*>
temp2 := int2float(2)
temp3 := temp1 / temp2
<id,a*verage*> := temp3

Uma instrução TAC possui no máximo três operandos e, em geral, é uma combinação de **atribuição** e um **operador binário**.

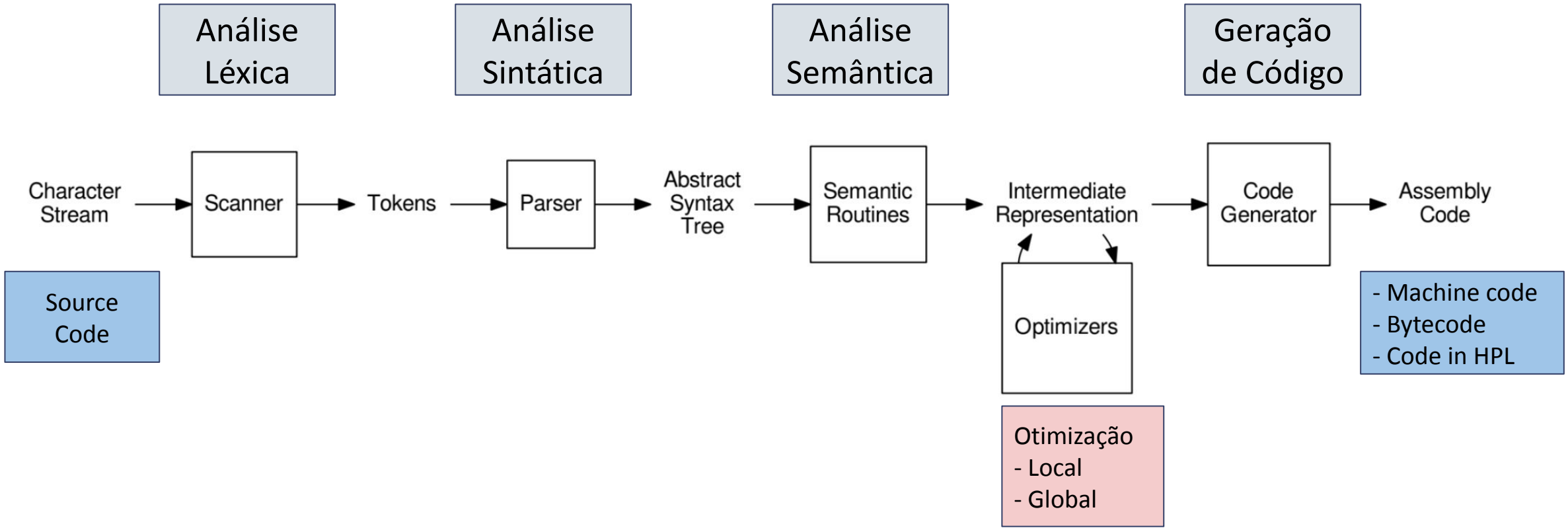
Geração de Código Intermediário



Exercício 4

Representar o trecho de programa do Exercício 1 usando a representação TAC. Assumir que todas as variáveis foram declaradas como "float".

Processo de Compilação



Otimização de Código (intermediário)

temp1 := <id,*min*> + <id,*max*>
temp2 := int_para_real(2)
temp3 := temp1 / temp2
<id,a*verage*> := temp3

average = (min + max) / 2;

temp1 := <id,*min*> + <id,*max*>
~~temp2 := int_para_real(2)~~
temp3 := temp1 / 2.0 ~~temp3 := temp1 / 2.0~~
<id,a*verage*> := temp1 / 2.0

Otimização de Código

Constant folding

- A avaliação da expressão sempre tem o mesmo resultado

```
pennyArea = 3.14159 * (0.75 / 2) * (0.75 / 2);
```

- Substituição da expressão pelo valor calculado em tempo de compilação

```
pennyArea = 0.4417860938;
```

Geração de Código

- Questões
  - alocação de registros
  - geração de código em assembly

```
CRVL z
CRCT 100.0
MULT
CRVL y
SOMA
ARMZ x
```

```
x := y + z * 100.0;
```

Atividades Transversais

Gerência de Tabela de Símbolos

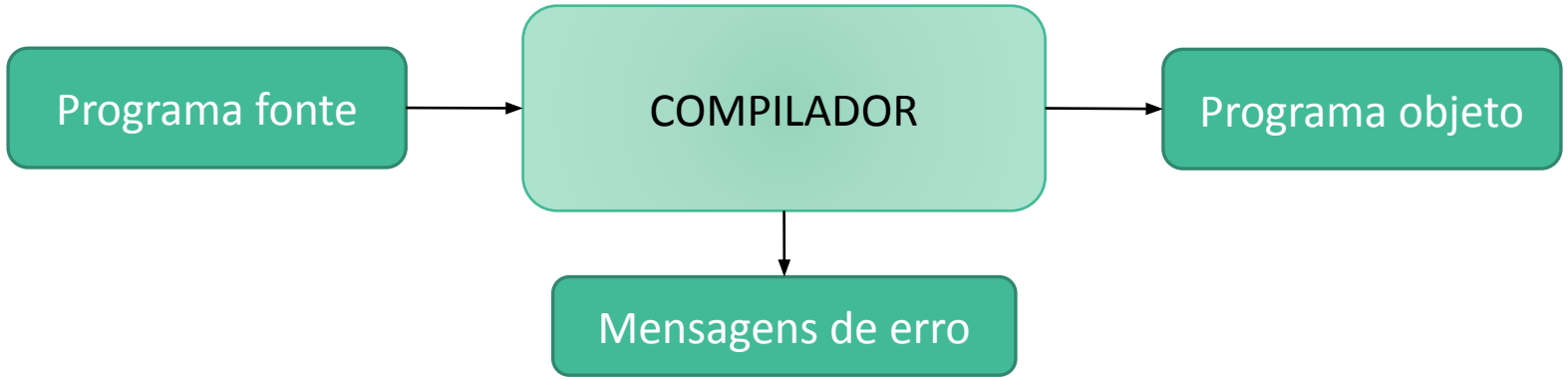
	categoria	**nome**	**tipo**	**...**
1	var	x	real	...
2	var	y	real	...
3	var	z	real	...
...	...	...	...	...

Tratamento de Erros

- Detecção
- Recuperação

COMPILADOR

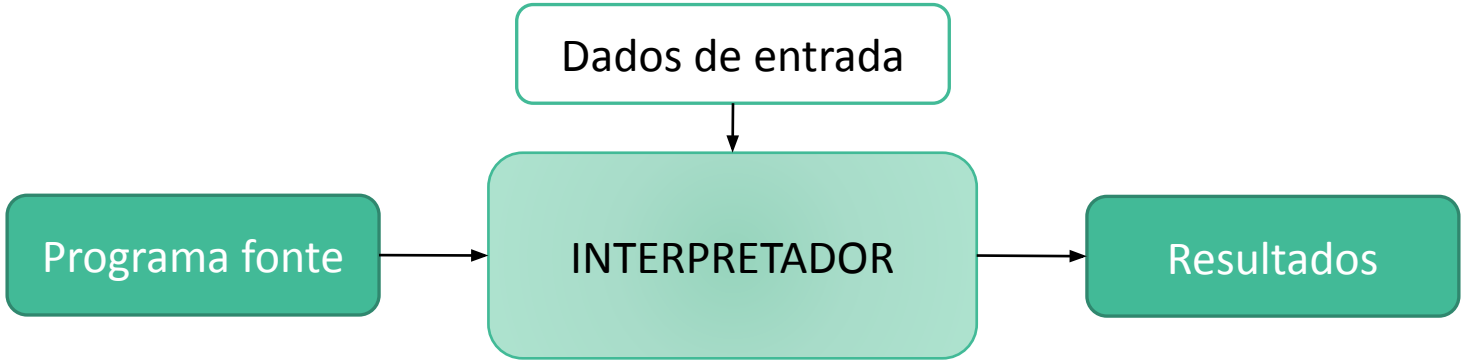
Traduz (e otimiza) um programa fonte para um programa objeto equivalente



O programa objeto executável recebe a entrada e gera a saída do programa.

INTERPRETADOR

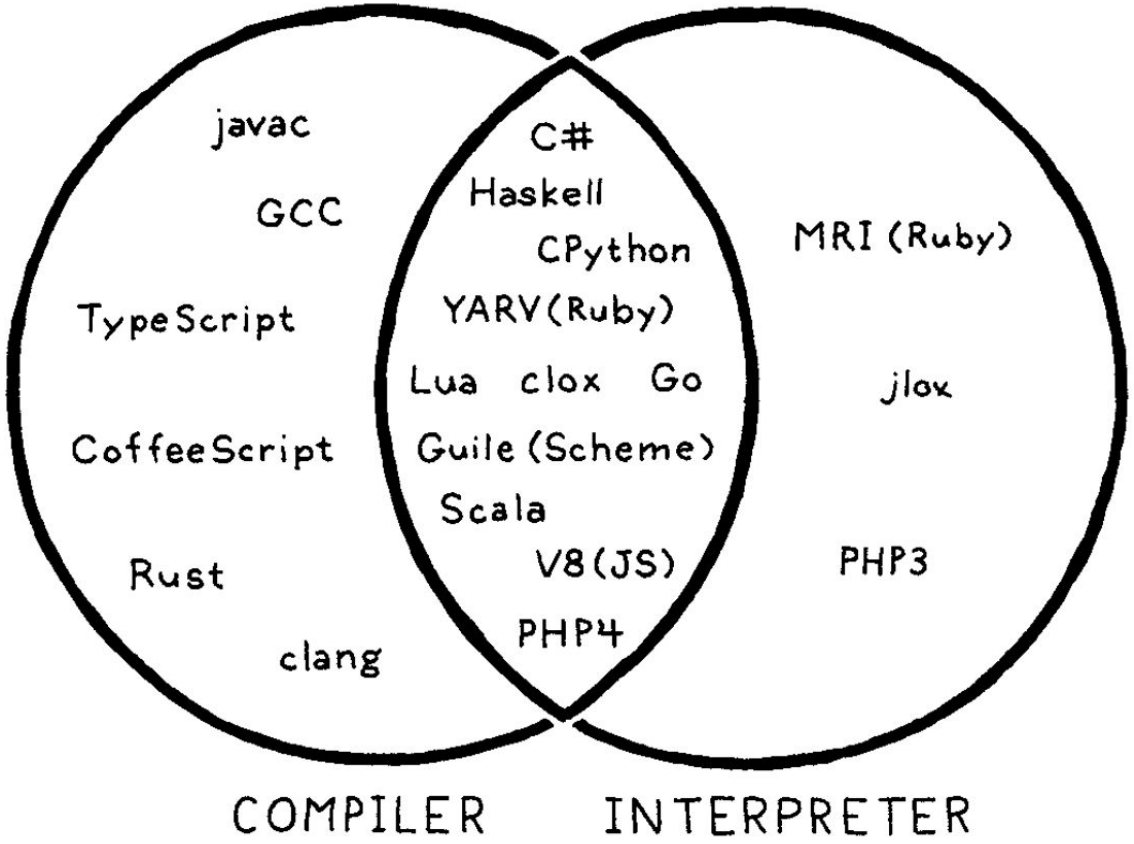
Analisa e executa cada linha do programa fonte diretamente



Vantagens

- execução imediata do programa fonte

Compilador e Interpretador



Processo de Tradução

Source language:	**Target language:**
C++ (**Programming language)**	Sparc code (**Machine language**)
Java (**Programming language**)	Java bytecode (**Abstract machine**)
C++ (**Programming language**)	C (**Programming language**)
LaTeX (**Application language**)	HTML (**Application language**)

Projeto de um compilador

Recomendações para a construção de compiladores

- Observar rigorosamente a definição da linguagem
- Utilizar ferramentas de geração (flex e bison)
- Utilizar práticas modernas de engenharia de software
- Utilizar algoritmos e métodos (re)conhecidos
- Validar o compilador através de um conjunto representativo de casos de teste.

Definição de Linguagem de Programação

Especificação mínima de uma linguagem

- conjunto de símbolos permitidos em programas;
- conjunto de programas válidos;
- significado dos programas válidos.

Sintaxe

- forma, estrutura.

Semântica

- significado, tipos.

Ferramentas para a Construção de Compiladores

- Lex/Flex
  - gera analisador léxico a partir de expressões regulares que especificam L
- Yacc/Bison
  - gera analisador sintático a partir de uma gramática livre de contexto G que especifica L

Compilador é Software

- Atributos de qualidade
- Arquitetura de software
- Desenvolvimento: técnicas, práticas, ferramentas, pessoas

ATRIBUTOS DE QUALIDADE DE UM COMPILADOR

- Correção do compilador
  - Traduzir apenas programas corretos
  - Rejeitar programas incorretos e emitir msg de erro
  - Gerar código objeto equivalente
- Eficiência
  - Memória e tempo usados pelo compilador
- Eficiência do código gerado
  - Memória e tempo usados pelo código gerado
  - Tarefa do compilador: Otimização
- Suporte ao usuário
  - Tarefa do compilador:
    - Tratamento de erros
- Robustez
  - Fornecer uma reação razoável para toda entrada

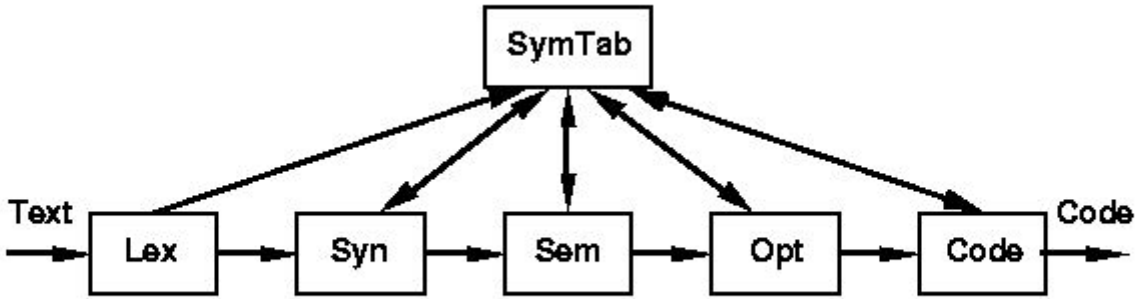
Arquitetura de Software

Compiladores

- Década de 70
  - compilação = processo sequencial

Pipeline, Batch

Traditional Compiler Model



Traditional Compiler Model with Shared Symbol Table

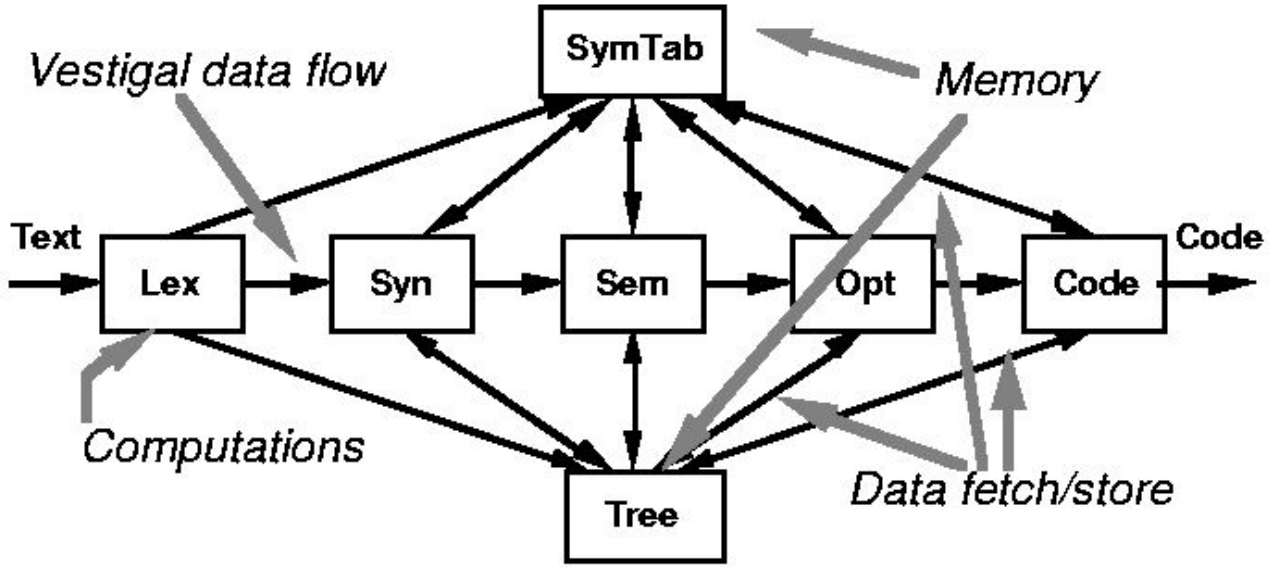
Arquitetura de Software

Compiladores

Híbrida

Anos 80

- gramática de atributos
- árvore sintática abstrata decorada
- vestígios de fluxo de dados

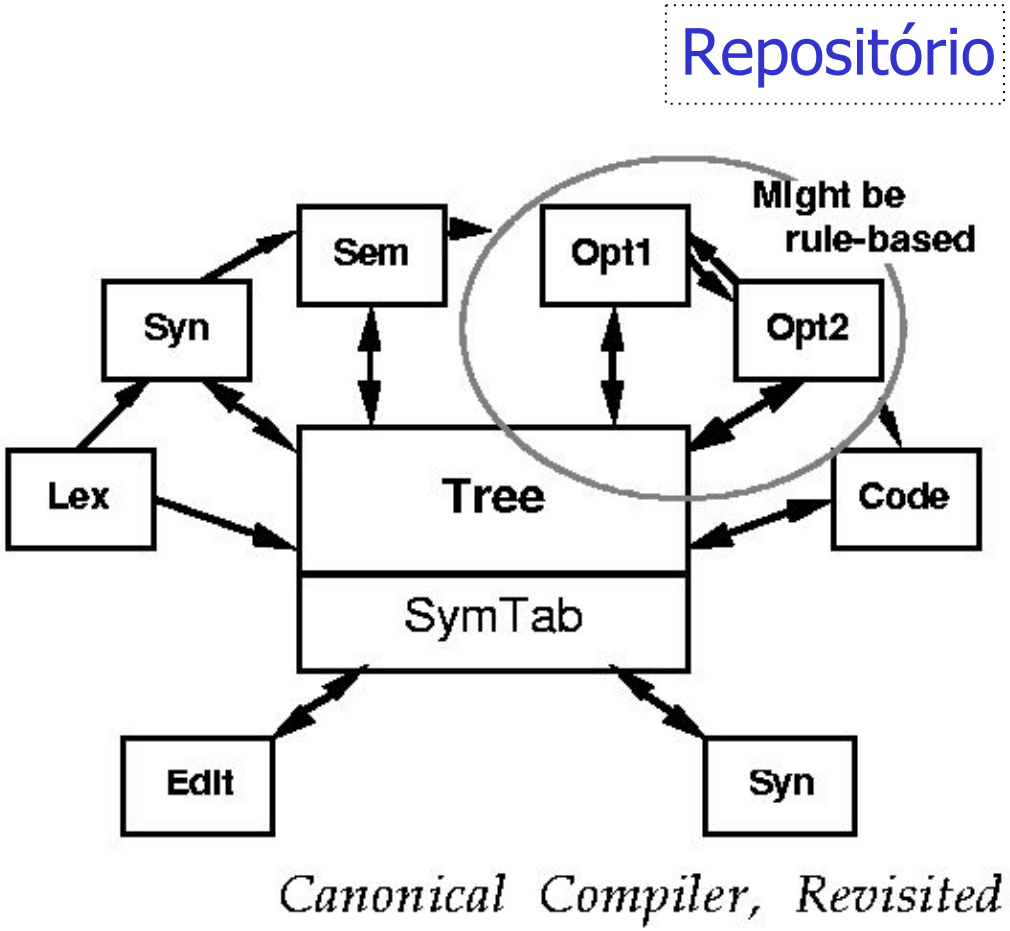


Modern Canonical Compiler

Arquitetura de Software

Compiladores

- Atividades utilizam a árvore sintática e a tabela de símbolos
  - acesso e manipulação de dados
- Esta arquitetura facilita a representação de ferramentas que manipulam a RI
  - editores de sintaxe
  - analisadores



Canonical Compiler, Revisited