Curso: Engenharia de Computação

Linguagens Formais e Compiladores

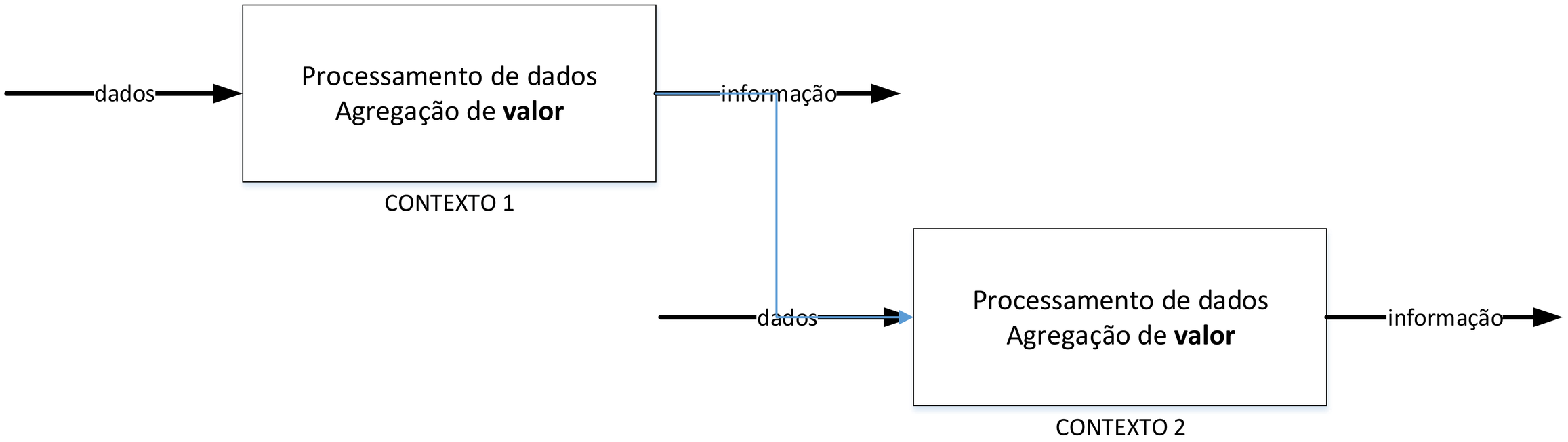
Prof. Clayton J A Silva, MSc
clayton.silva@professores.ibmec.edu.br

dados x informação

Dado: atributo de algum objeto, que, isoladamente, sem ser processado, possui um **valor bruto**, a ser explorado para um fim específico

Informação: **resultado do processamento do dado**, agregando-lhe valor para uma **aplicação cuja finalidade** seja bem definida

Importância do **contexto**



Dados e informação

Domínios de Programação

Inteligência Artificial

Programação de Sistemas

Aplicações Comerciais

Aplicações Científicas

Critérios de avaliação de Linguagens de Programação

Critérios de avaliação de LP

- Legibilidade
- Facilidade de escrita
- Confiabilidade

Legibilidade

- A legibilidade concorre diretamente para a **facilidade de manutenção**, o que está modernamente consagrado como uma medida importante da **qualidade** de programas e linguagens.

- A legibilidade deve considerar o **domínio do problema**.

- A **simplicidade geral** influencia diretamente a legibilidade. Possuir **múltiplos recursos** – ou seja, haver mais de uma maneira de realizar operações - e **sobrecarregar** operadores são aspectos que definem o maior ou menor grau de simplicidade.

Legibilidade

- A legibilidade também é afetada pela **ortogonalidade** da LP, definida como a característica de um **conjunto pequeno de construções primitivas** poder ser **combinado a um número pequeno de formas** para **produzir estruturas de controle e de dados**.

- A presença de **mecanismos** adequados para **definir tipos de dados e de estruturas** também afeta a legibilidade.

- **Dispor de sentenças que indiquem o propósito** contribui também para a legibilidade, possuir **identificadores de formatos** variados e **clareza das palavras especiais** concorre para legibilidade.

Facilidade de escrita

- A facilidade da escrita é a medida do quanto a LP pode ser usada para criar os programas. Também deve considerar o domínio do problema.
- A **simplicidade** geral assim como a **ortogonalidade** obviamente afetam diretamente a facilidade de escrita.
- A capacidade de **definir e usar estruturas ou operações complicadas omitindo os detalhes**, definida como **abstração**, é fundamental para facilitar a escrita dos códigos. A abstração pode se manifestar nos processos estabelecidos no código quanto nos dados.

Facilidade de escrita

- A existência de **operadores poderosos**, definida como expressividade da LP, é muito importante para facilitar a escrita de códigos.

Confiabilidade

- A confiabilidade de um programa é definida pelo atendimento ou **conformidade a todas as suas especificações**, em todas as condições de utilização.
- **Simplicidade**, **ortogonalidade**, **tipagem** e **sintaxe** influenciam a confiabilidade dos programas.
- A **possibilidade de verificar tipos**, ou seja, de realizar testes para detectar erros tanto na compilação quanto na execução concorre para aumentar a confiabilidade.

Confiabilidade

- A **habilidade de tratar exceções** em tempo de execução, ou seja, interceptar erros em tempo de execução, tomar **medidas corretivas** e prosseguir contribui para o aumento da confiabilidade.

- A utilização de **apelidos** para **acessar diretamente células de memória** influencia a confiabilidade da LP.

Tabela 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**LEGIBILIDADE**	**FACILIDADE DE ESCRITA**	**CONFIABILIDADE**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

[1] O quarto critério principal é o custo, que não foi incluído na tabela porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam.

Influências nos projetos das Linguagens de Programação

Arquitetura de computadores

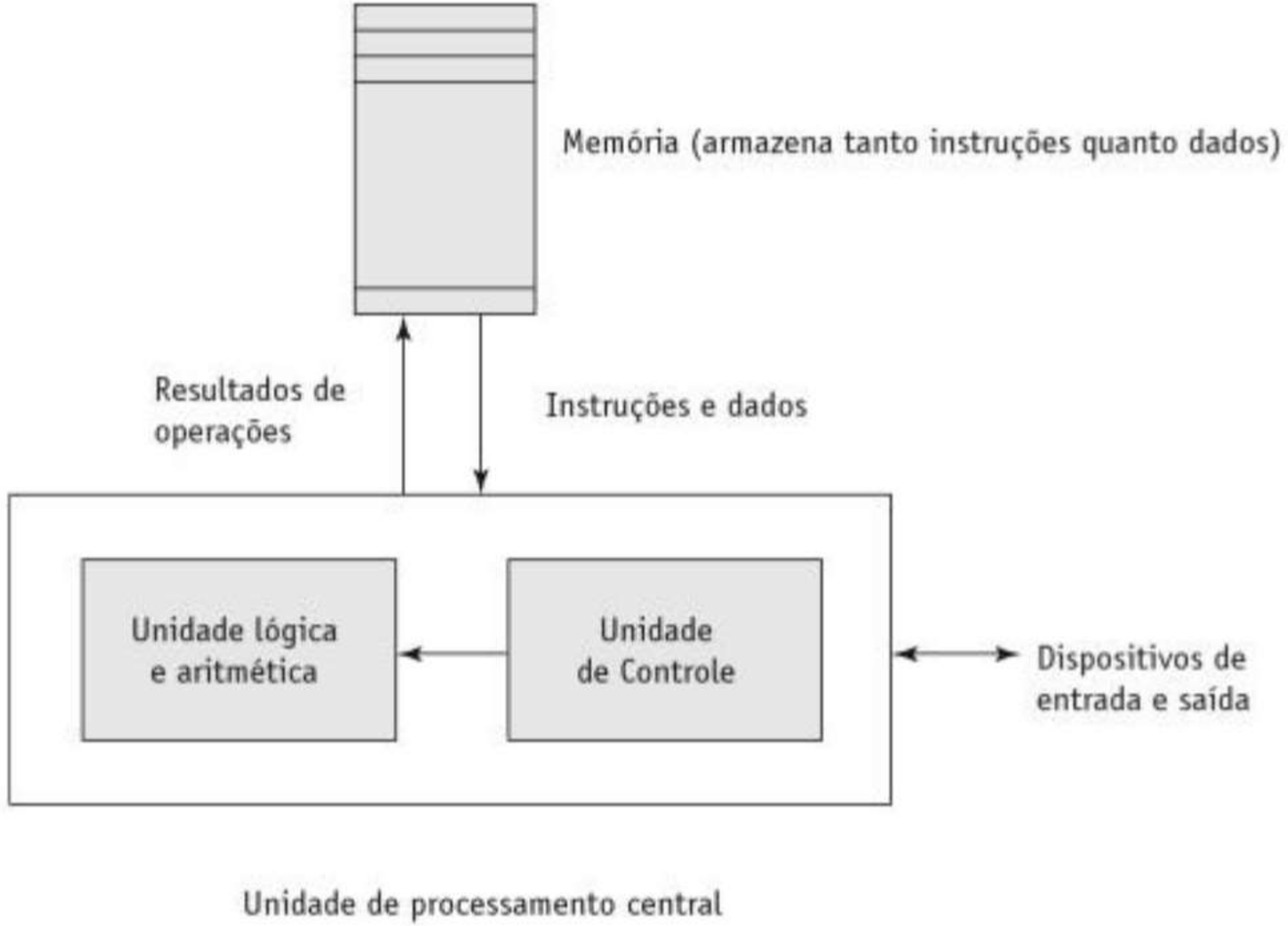
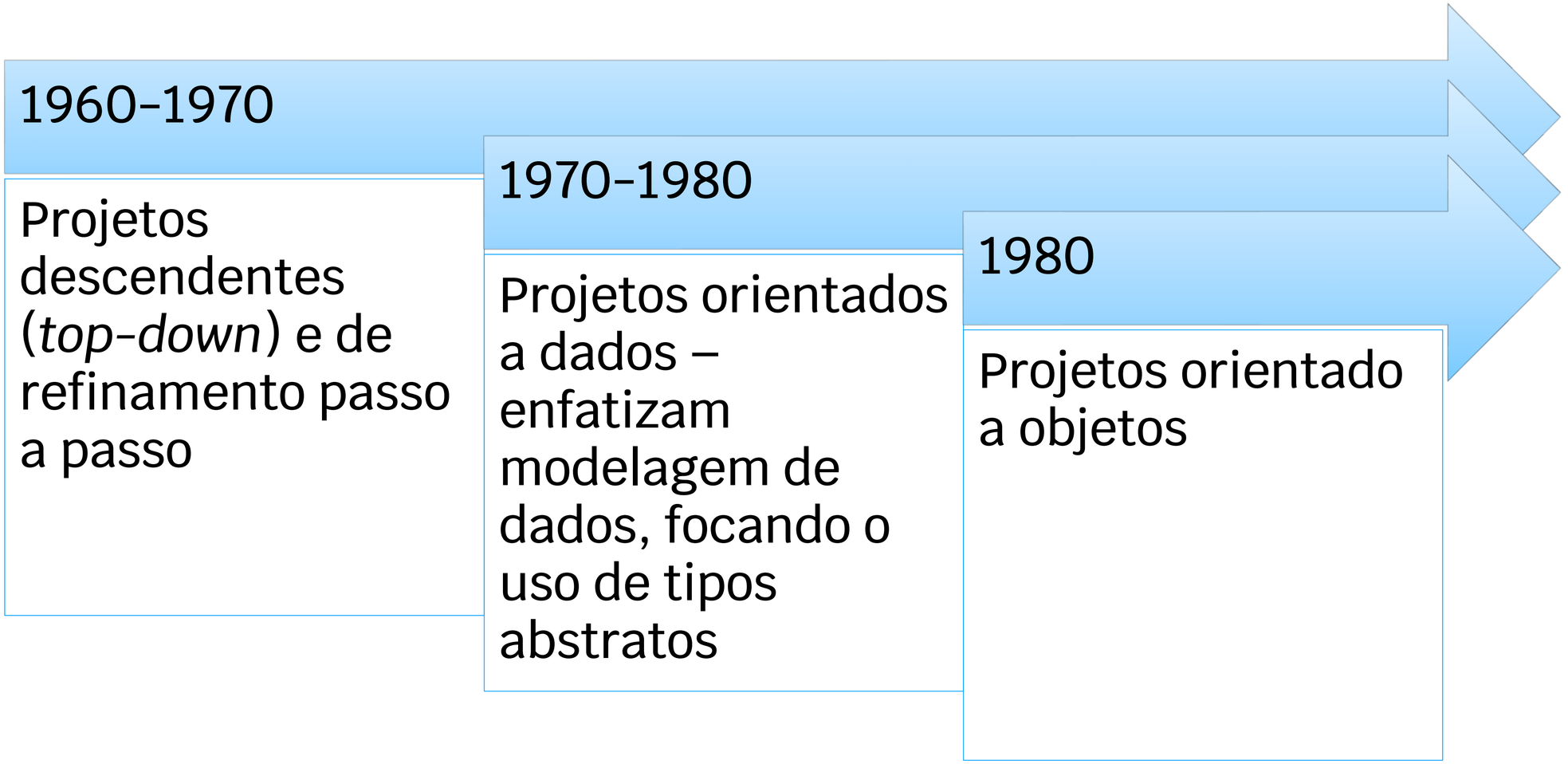


Figura 1.1 A arquitetura de computadores de von Neumann.

Metodologias



Categorias das Linguagens de Programação

Categorias das LP

- LP **imperativas**: algoritmo especificado em detalhes e **ordem de execução** deve ser incluída
- LP **funcionais**: construção do software utilizando **funções puras** – **declarativa**, ao invés de imperativa
- LP **lógicas**: baseada em **regras**, especificadas sem uma ordem particular – o sistema de implementação da linguagem escolha a ordem de execução
- LP **orientada a objetos**: suportam paradigma de orientação a objetos no desenvolvimento do software – evolução das imperativas
- Linguagens de **marcação**/de programação híbrida: as de marcação não são de programação
- (*) Linguagens de **scripting**

(*) Não há consenso em classificar como uma categoria entre os autores

Métodos de implementação

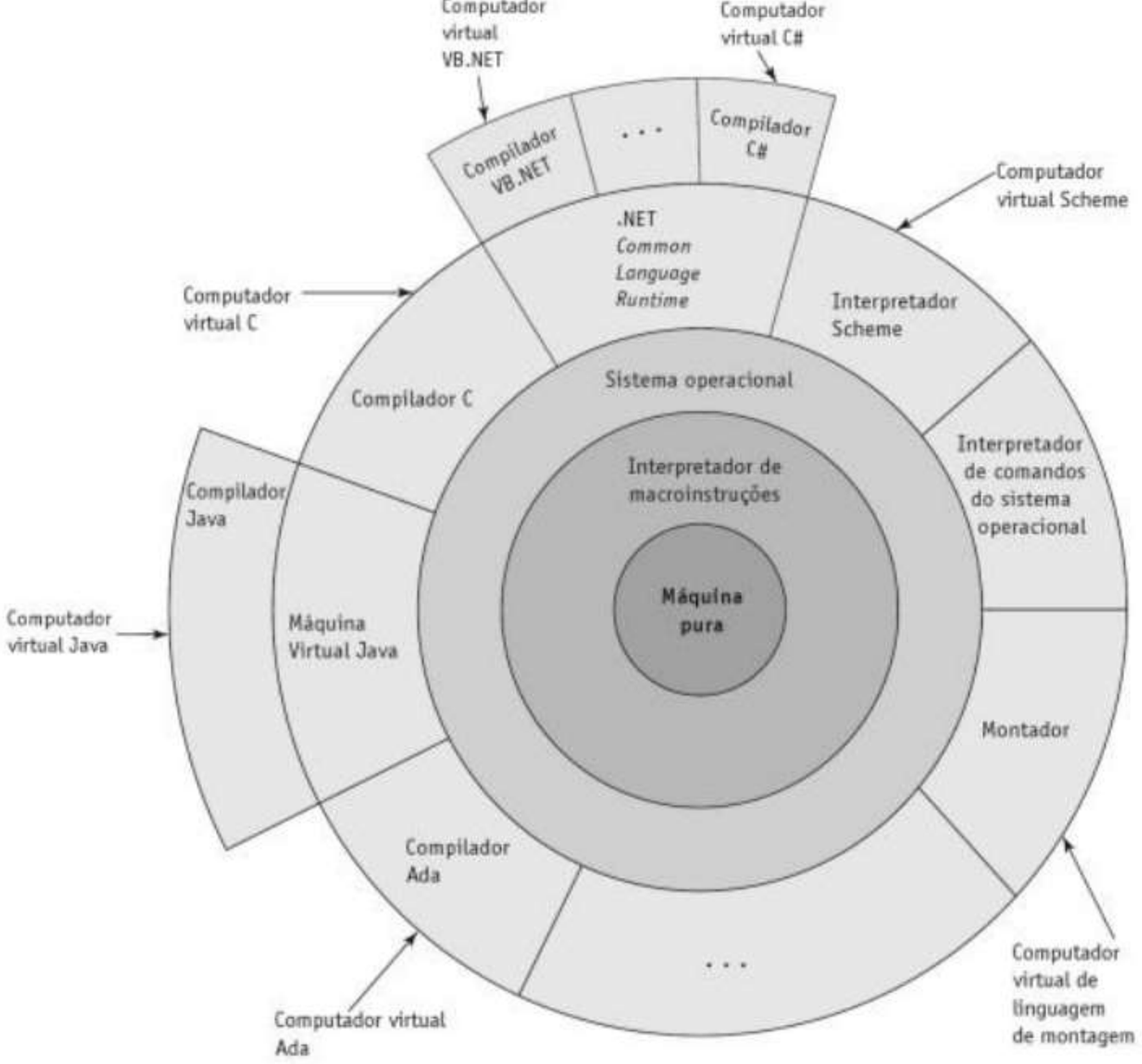


Figura 1.2 Interface em camadas de computadores virtuais, fornecida por um sistema de computação típico.

Compilação

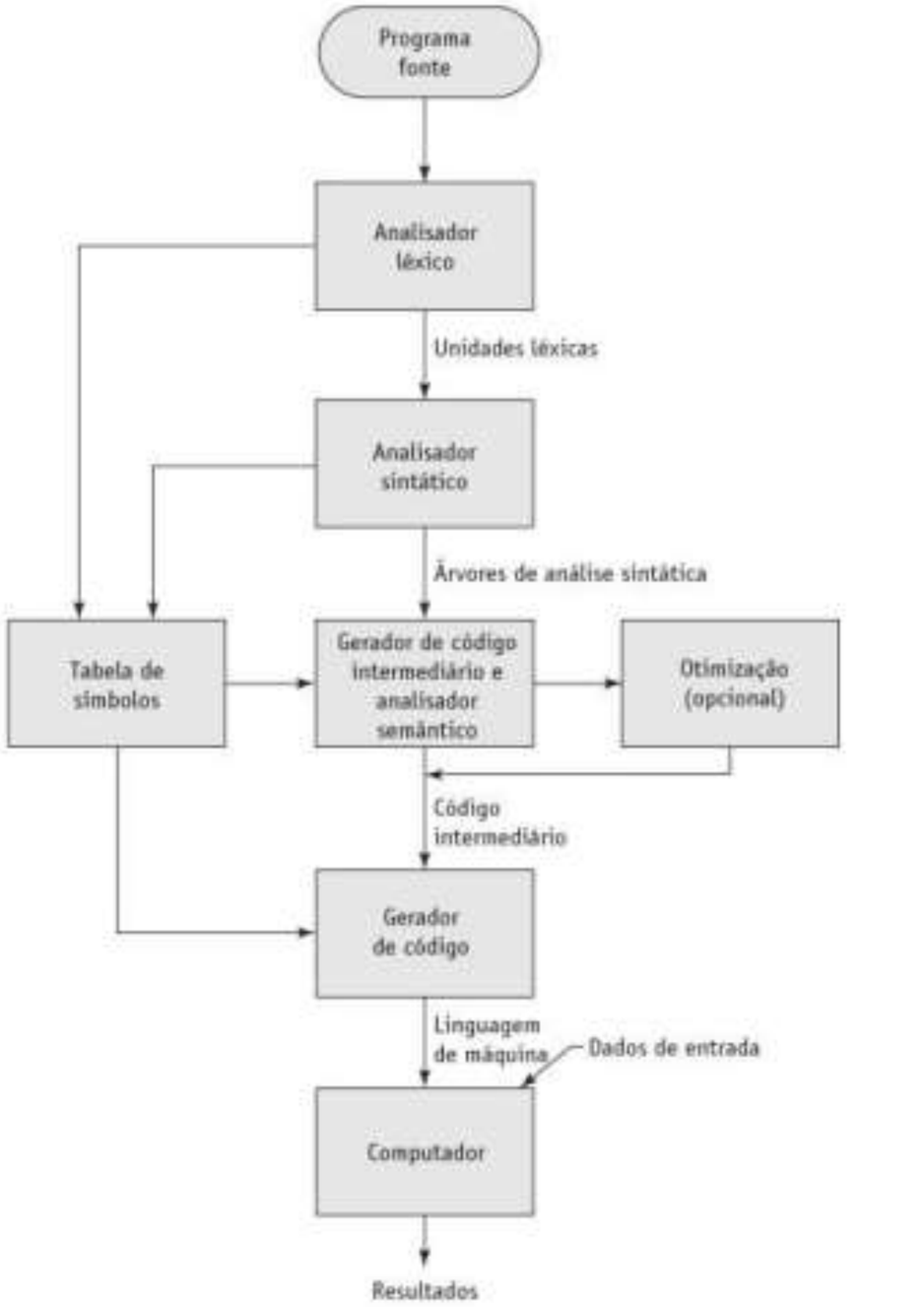


Figura 1.3 O processo de compilação.

Métodos de implementação: compilação

1. **Analisador léxico**: agrupar os caracteres do programa na linguagem fonte em unidades léxicas (identificadores, palavras especiais, operadores e símbolos de pontuação) – ignora os comentários

2. **Analisador sintático**: construir árvores de análise sintática, que representam a estrutura sintática do programa

3. **Gerador do código intermediário** e **analisador semântico**: o código intermediário se aproxima do nível de montagem e o analisador semântico trata da análise do significado das expressões, das instruções e das unidades de programa

Métodos de implementação
Compilação

4. **Otimizador**: aumentar a eficiência da execução, tornando os programas menores e mais rápidos – opcional

5. **Gerador de código**: traduzir o código intermediário para o código em linguagem de máquina

– A **tabela de códigos** serve como uma base de dados para o analisador semântico e gerador de código

Métodos de implementação: ligação e carga

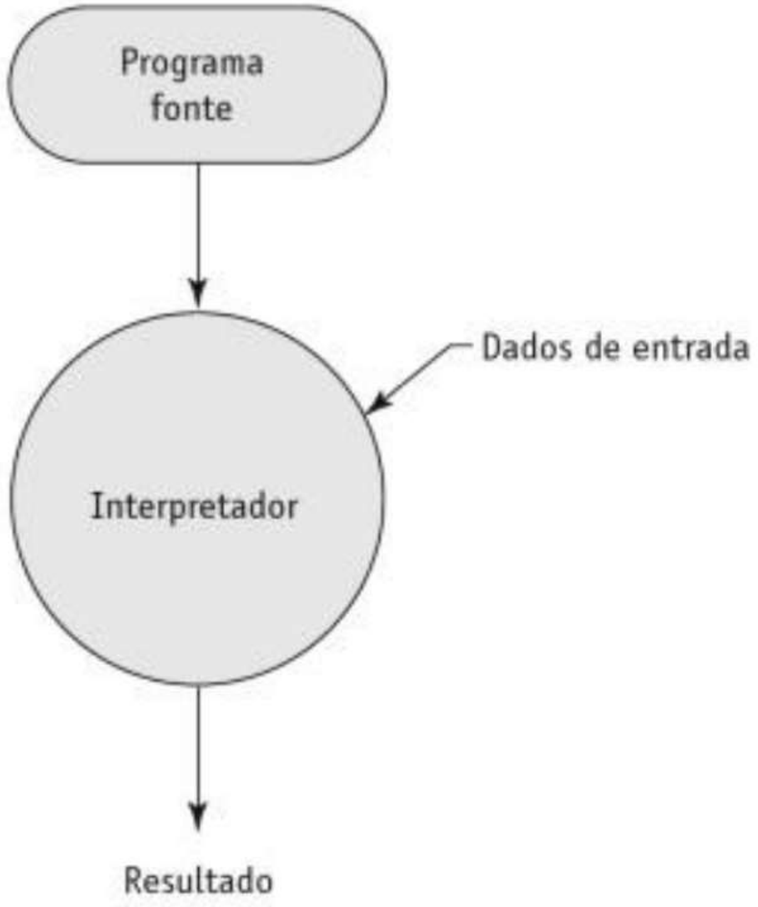
- O programa de usuário geralmente **precisa de programas do sistema operacional**. A operação de ligação **conecta o programa do usuário com o sistema operacional**.
- A ligação também **conecta o programa do usuário com outros programas de usuários** previamente compilados.
- Programa ligador (***linker***) executa a ligação
- Código de usuário + código de sistema = módulo de carga, ou imagem **executável**

Métodos de implementação: pré-processadores

- O pré-processador é um programa que **processa outro programa imediatamente antes dele ser compilado**
- As instruções do pré-processador são **embutidas nos programas**
- Trata-se essencialmente de um programa que expande macros.

Métodos de implementação: interpretação pura

1. Os programas são interpretados por um *software* executado em uma **máquina virtual**
2. As sentenças são **executadas uma por vez**



Comparação compilação x interpretação

- A interpretação tem vantagem de permitir **fácil implementação** de muitas operações em tempo de depuração em código fonte

- A **execução** em sistemas que utilizam interpretação apresenta **tempo mais longo** do que em sistemas compilados

- A compilação normalmente requer **mais espaço em memória** para execução do programa

Métodos de implementação híbridos

- **Desempenho intermediário** entre compilação e interpretação
- O código intermediário, chamado de ***bytecode*** em Java, fornece **portabilidade** a qualquer máquina que tenha interpretador de *bytecodes*

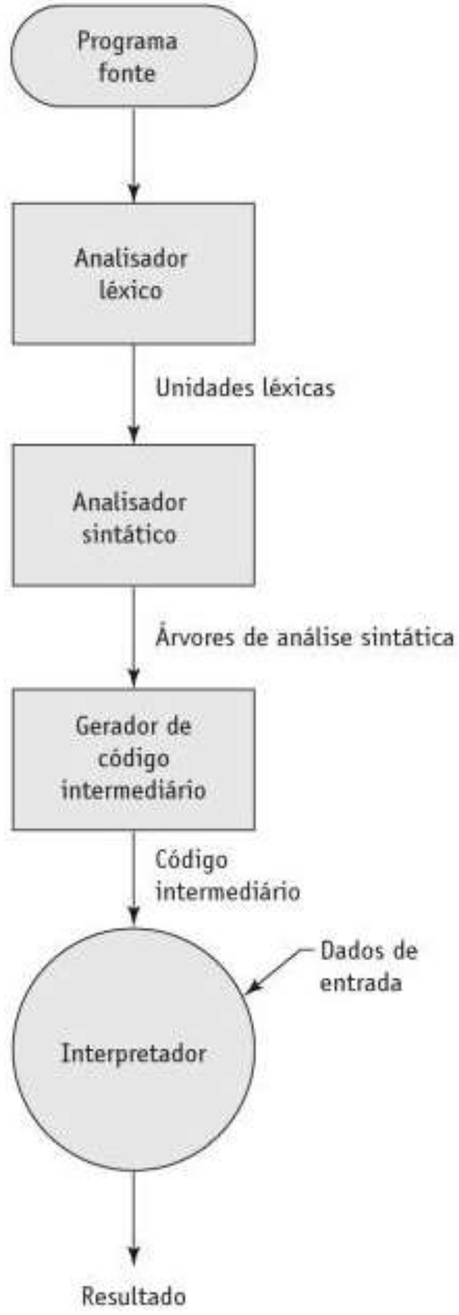


Figura 1.5 Sistema de implementação híbrido.

A descrição geral da sintaxe de LP

Descrição de LP

- Uma LP é um **conjunto de cadeias de caracteres** formadas a partir de um **alfabeto**
- As cadeias de caracteres de uma linguagem são chamadas de **sentenças**
- As unidades sintáticas de mais baixo nível das linguagens são chamadas de ***lexemas,*** que incluem **caracteres** - literais numéricos, operadores, palavras especiais e outros

Descrição de LP

- Seja a sentença, em linguagem C: `cont = 2 * j + 17;`
- São lexemas presentes na sentença:

```
cont
=
2
*
j
+
17
;
```

Descrição de LP

- Os *lexemas* são divididos em grupos, que se configuram **categorias**, chamadas de ***token***. Na sentença, em C, `cont = 2 * j + 17;`

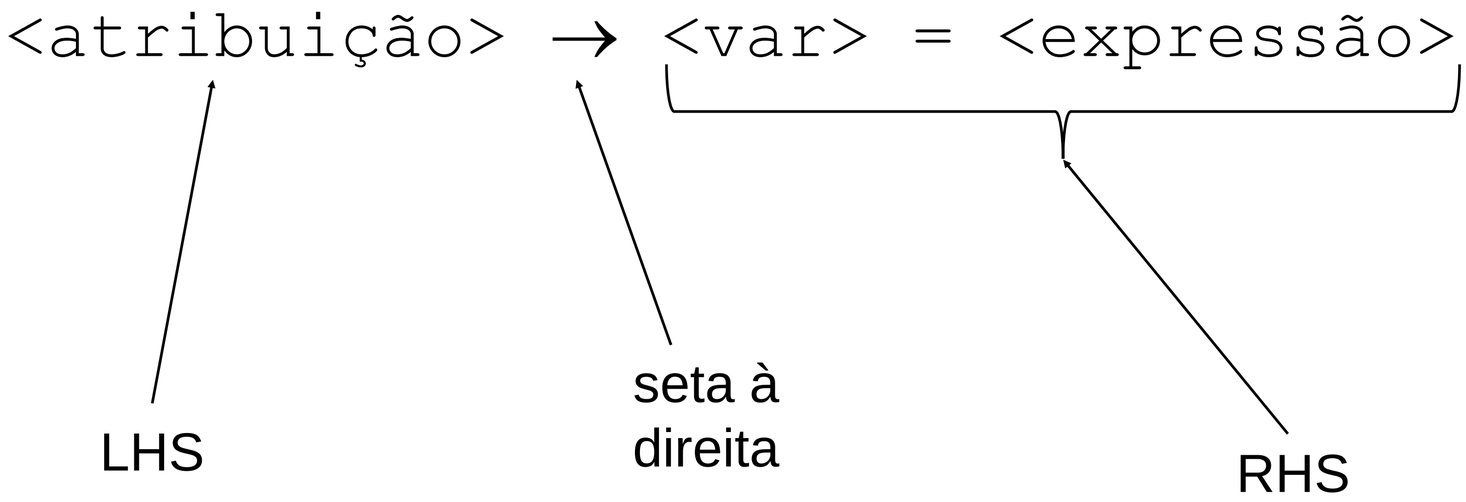
Lexemas	tokens
`cont`	`identificador`
`=`	`sinal_igual`
`2`	`int_literal`
`*`	`mult_op`
`j`	`identificador`
`+`	`soma_op`
`17`	`int_literal`
`;`	`ponto_virgula`

Forma de *Backus-Naur* (BNF)

- **Forma de *Backus-Naur (BNF)*** para descrição de LP: notação natural para **descrever sintaxe**
- **Metalinguagem**: linguagem para descrever outra linguagem
- A BNF usa **abstrações** para descrever estruturas sintáticas
- Cada sentença é definida por uma expressão com uma **abstração à esquerda** (LHS, do inglês *left-hand side*) seguida por **seta à direita** e uma **abstração à direita** (RHS, do inglês *righ-hand side*) – a definição completa é chamada de **regra** ou **produção**

Forma de *Backus-Naur* (BNF)

- Seja uma sentença de atribuição, na linguagem C. Usando a BNF, pode ser descrita como



Forma de *Backus-Naur* (BNF)

- Seja uma sentença de atribuição, na linguagem C. Usando a BNF, pode ser descrita como

<atribuição> → <var> = <expressão>

total = sub1 + sub2

Forma de *Backus-Naur* (BNF)

- As **abstrações** são chamadas de **símbolos não terminais.** Os *lexemas* e *tokens* das regras são chamados de **símbolos terminais.**

<atribuição> → <var> = <expressão>

<expressão> → <var> + <var>

Forma de *Backus-Naur* (BNF)

- Os símbolos não terminais podem ter **mais de uma definição**. Cada definição pode ser **representada separadamente** ou **representada por uma regra única utilizando-se o símbolo |, equivalente à disjunção, Ou**

```
<expressão> → <var> + <var>
            | <var> - <var>
            | <var> * <var>
            | <var> / <var>
```

Forma de *Backus-Naur* (BNF)

- As listas podem ser descritas utilizando-se **recursão**

```
<expressão> → <var>
              | <var> + <expressão>
              | <var> - <expressão>
              | <var> * <expressão>
              | <var> / <expressão>
```

Forma de *Backus-Naur* (BNF)

- Uma descrição BNF, chamada de **gramática**, é uma coleção de regras para descrever uma linguagem
- A gramática para descrever uma linguagem começa com um símbolo não terminal que é o **símbolo inicial** e prossegue com a apresentação de todas as regras, na sequência em que aparecem os símbolos não terminais
- A partir do símbolo inicial, sucessivas cadeias na sequência da gramática são derivadas, substituindo-se os símbolos não terminais, até alcançar os símbolos terminais – **derivação**
- Cada uma das cadeias da derivação é chamada de **forma sentencial**

Forma de *Backus-Naur* (BNF)

- Seja a gramática dada pelas regras

```
<programa> → begin <lista_cmdo> end
<lista_cmdo> → <cmdo>
             | <cmdo>; <lista_cmdo>
<cmdo> → <var> := <expressão>
<var> → A | B | C
<expressão> → <var> + <var>
            | <var> - <var>
            | <var>
```

Forma de *Backus-Naur* (BNF)

- Uma derivação será

```
<programa> => begin <lista_cmdo> end
           => begin <cmdo> end
           => begin <var> := <expressão>
           => begin A := <expressão>
           => begin A := <var> + <var>
```

- ...prosseguindo até completar todas as possibilidades

Referências

- Sebesta, Robert W.; Conceitos de Linguagens de Programação; Capítulo 1, Capítulos 3.1, 3.2, 3.3; Bookman

IBMEC.BR

/IBMEC

IBMEC

@IBMEC_OFICIAL

@IBMEC

Ibmec