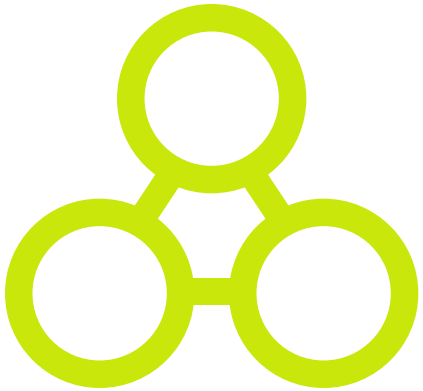


nemo

ontology & conceptual modeling research group

Linguagens de Programação

1 – Introdução

Vítor E. Silva Souza

(vitor.souza@ufes.br)

http://www.inf.ufes.br/~vitorsouza

Departamento de Informática

Centro Tecnológico

Universidade Federal do Espírito Santo

Conteúdo do curso

- Introdução;
- Amarrações;
- Valores e tipos de dados;
- Variáveis e constantes;
- Expressões e comandos;
- Modularização;
- Polimorfismo;
- Exceções;
- Concorrência;
- Avaliação de linguagens.

- Estes slides foram baseados em:
  - Slides do prof. Flávio M. Varejão;
  - Livro “Linguagens de Programação – Conceitos e Técnicas” (Varejão);
  - Livro “Linguagens de Programação – Princípios e Paradigmas, 2a edição” (Tucker & Noonan).

Por que estudar LPs

- Maior capacidade de desenvolver soluções computacionais para problemas;
- Maior habilidade ao usar uma LP;
- Maior capacidade para escolher LPs apropriadas;
- Maior habilidade para aprender novas LPs;
- Maior habilidade para projetar novas LPs.

Contexto

Processo de Desenvolvimento de Software

Papel das LPs no PDS

- O objetivo de LPs é tornar mais efetivo o processo de desenvolvimento de software (PDS);
- PDS visa geração e manutenção de software de modo produtivo e garantia de padrões de qualidade;
- Principais Propriedades Desejadas em um Software:
  - Confiabilidade;
  - Manutenibilidade;
  - Eficiência.

Papel das LPs no PDS

- **Requisitos**: análise da viabilidade tecnológica;
- **Projeto**: utilização de métodos e modelos adequados à linguagem;
- **Implementação**: papel essencial da LP;
- **Validação**: podem facilitar o processo (ex.: depuradores);
- **Manutenção**: modularização, legibilidade, etc. influenciam.

Objetivos de projeto

- Segundo Tucker & Noonan:
  - **Simplicidade e legibilidade**: fácil de escrever, ler, aprender, ensinar;
  - **Clareza nas ligações** (amarrações, parte 2 do curso);
  - **Confiabilidade**: tratamento de exceções, restringir vazamento de memória, tipagem forte, sintaxe/semântica bem definidas, verificação & validação;
  - **Suporte**: compiladores acessíveis (baratos, muitas plataformas), cursos, livros, comunidade;
  - **Abstração**: não ter que reinventar a roda;

Objetivos de projeto

- Segundo Tucker & Noonan (continua):
  - **Ortogonalidade**: menor número de regras excepcionais possível, cidadãos de primeira classe;
  - **Implementação eficiente**: implementações iniciais de Java, Ada e Algol sofreram críticas.

Propriedades desejáveis em LPs

- Segundo Varejão:
  - Legibilidade;
  - Redigibilidade;
  - Confiabilidade;
  - Eficiência;
  - Facilidade de aprendizado;
  - Modificabilidade;
  - Reusabilidade;
  - Portabilidade.

Legibilidade

- Marcadores de bloco:

```
if (x > 1)
  if (x == 2)
    x = 3;
else
  x = 4;
```

- Desvios incondicionais: goto;
- Duplicação de significado de vocábulos:
  - Java: `this`;
  - C/C++: `*p = (*p)*q;`

Legibilidade

- Efeitos colaterais:

```
int x = 1;
int retornaCinco() {
  x = x + 3;
  return 5;
}
void main() {
  int y;
  y = retornaCinco();
  y = y + x;
}
```

Redigibilidade

- Tipos de Dados Limitados (FORTRAN);
- Ausência de Tratamento de Exceções;
- Conflito Ocasional com Legibilidade:

```
void f(char *q, char *p) {
  for (;*q=*p; q++,p++);
}
```

O que faz o código acima?

Confiabilidade

- Declaração de tipos:

```
boolean u = true;
int v = 0;
while (u && v < 9) {
  v = u + 2;
  if  (v == 6) u = false;
}
```

Algo de errado no código acima?
Em Java, compila? E em C?

Tratamento de exceções

```
// Abre uma conexão com um banco de dados.
Connection conn = null;
try {
  Class.forName(driver);
  conn = DriverManager.getConnection(url, usu, senha);
}
catch (ClassNotFoundException | SQLException ex) {
 System.out.println("Problemas ao abrir conexão...");
}
return conn;
```

Separa o código de tratamento de erro.

Eficiência

- Verificação dinâmica de tipos;
- Controle de índice de vetor:
  - Java o faz;
  - C/C++ não.

Facilidade de aprendizado

- Excesso de características pode ser prejudicial:

```
c = c + 1;
c+=1;
c++;
++c;
```

- C++ vs. Java:
  - Herança múltipla;
  - Herança pública e privativa;
  - Ligação tardia ativada ou desativada;
  - Qual linguagem é mais fácil de aprender?

Modificabilidade

- Uso de constantes:

```
const float pi = 3.14;
```

- Em C, `pi` seria de fato constante?
- E em C++?

Reusabilidade

- Criação de bibliotecas de função:

```
void troca (int *x, int *y) {
  int z = *x;
  *x = *y;
  *y = z;
}
```

- Frameworks e plataformas de desenvolvimento.

Portabilidade

- Rigor no projeto;
- Pode constrastar com eficiência.

Especificação de LPs

- Três componentes: léxico, sintaxe e semântica:
- Por exemplo:

```
a = b;
```

  - Léxico: **a**, **=**, **b**, **;** fazem parte da linguagem;
  - Sintaxe: **a** seguido de **=**, seguido de **b**, seguido de **;** é um comando válido de atribuição;
  - Semântica: o valor de **b** deve ser copiado para a posição de memória definida por **a**.

Especificação de LPs

- Sintaxe é geralmente definida em BNF:

```
<expressão> ::= <valor> | <valor><operador><expressão>
<valor> ::= <número> | <sinal><número>
<número> ::= <semsinal> | <semsinal>.<semsinal>
<semsinal> ::= <dígito> | <dígito><semsinal>
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<sinal> ::= + | -
<operador> ::= + | - | / | *
```

Esse assunto será retomado na disciplina de Compiladores.

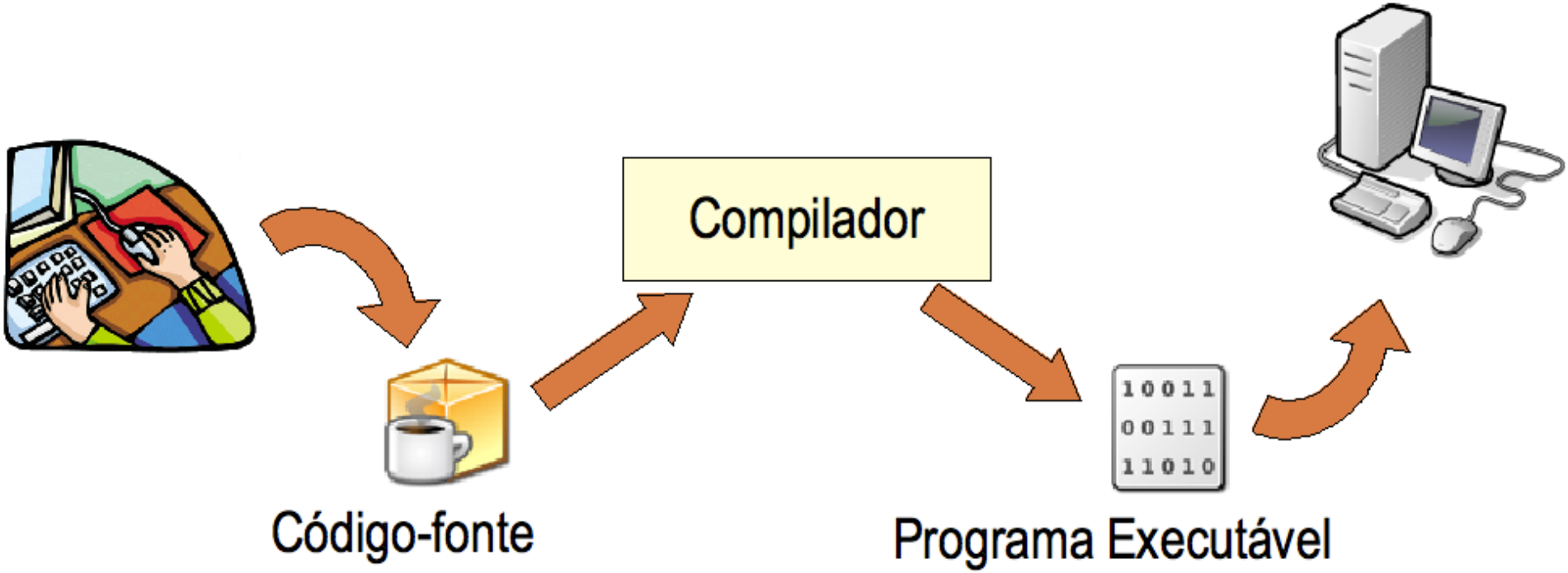
Especificação de LPs

- Semântica:
  - Descrição informal em linguagem natural;
  - Enfoque operacional: usar LP mais elementar.
- Necessidade de padronização:
  - ISO, IEEE, ANSI, NIST, etc.;
  - Auxilia na portabilidade e na aceitação da LP;
  - *Timing* é importante.

Exemplos: ANSI/ISO Cobol 2002, ISO Fortran 2004, ISO Haskell 1998, ISO Prolog 2000, ANSI/ISO C 1999, ANSI/ISO C++ 2003, ANSI/ISO Ada (2005), ANSI Smalltalk 2002, ISO Pascal 1990.

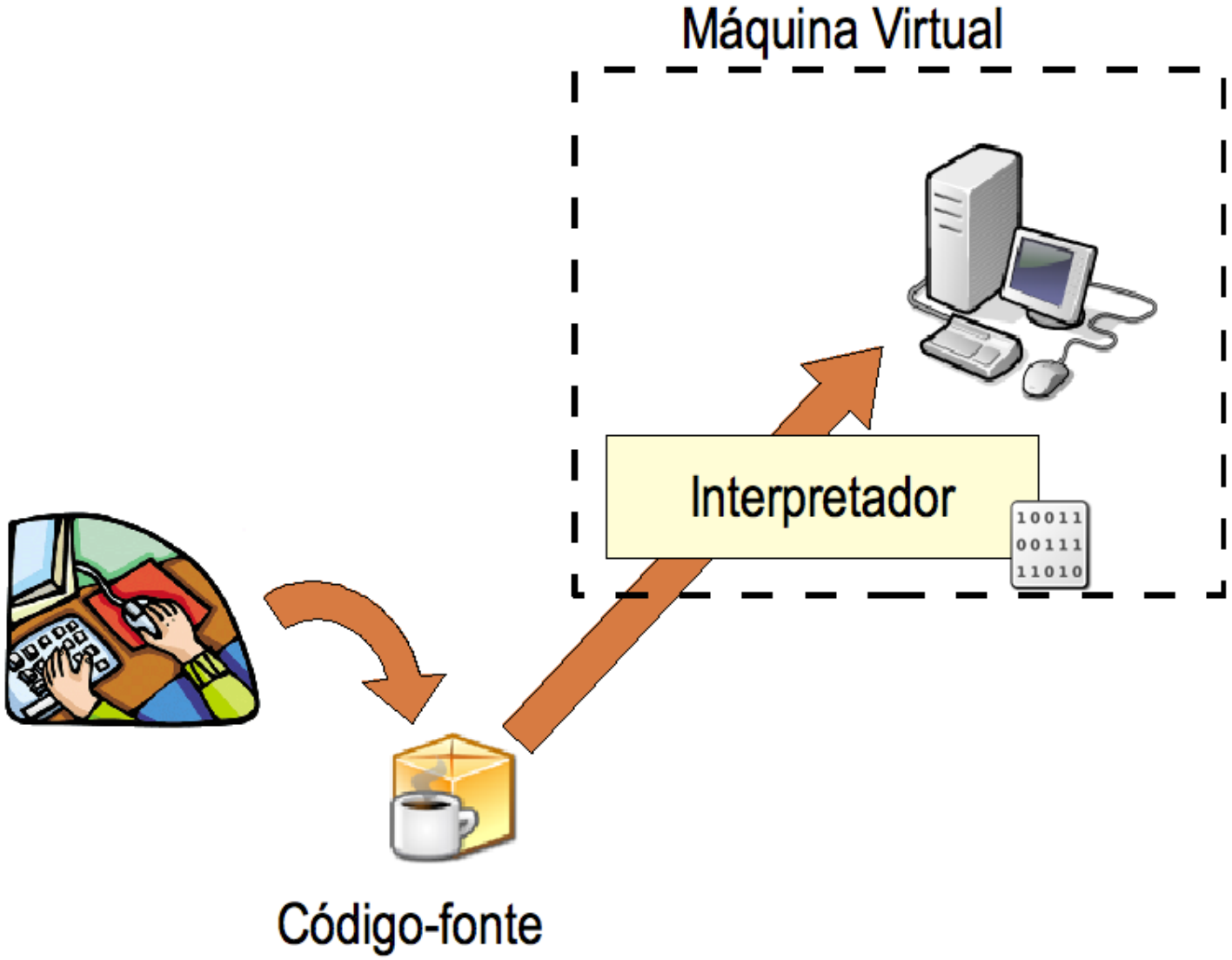
Implementação de LPs

- Compilação, interpretação ou híbrido?
- Compilação:



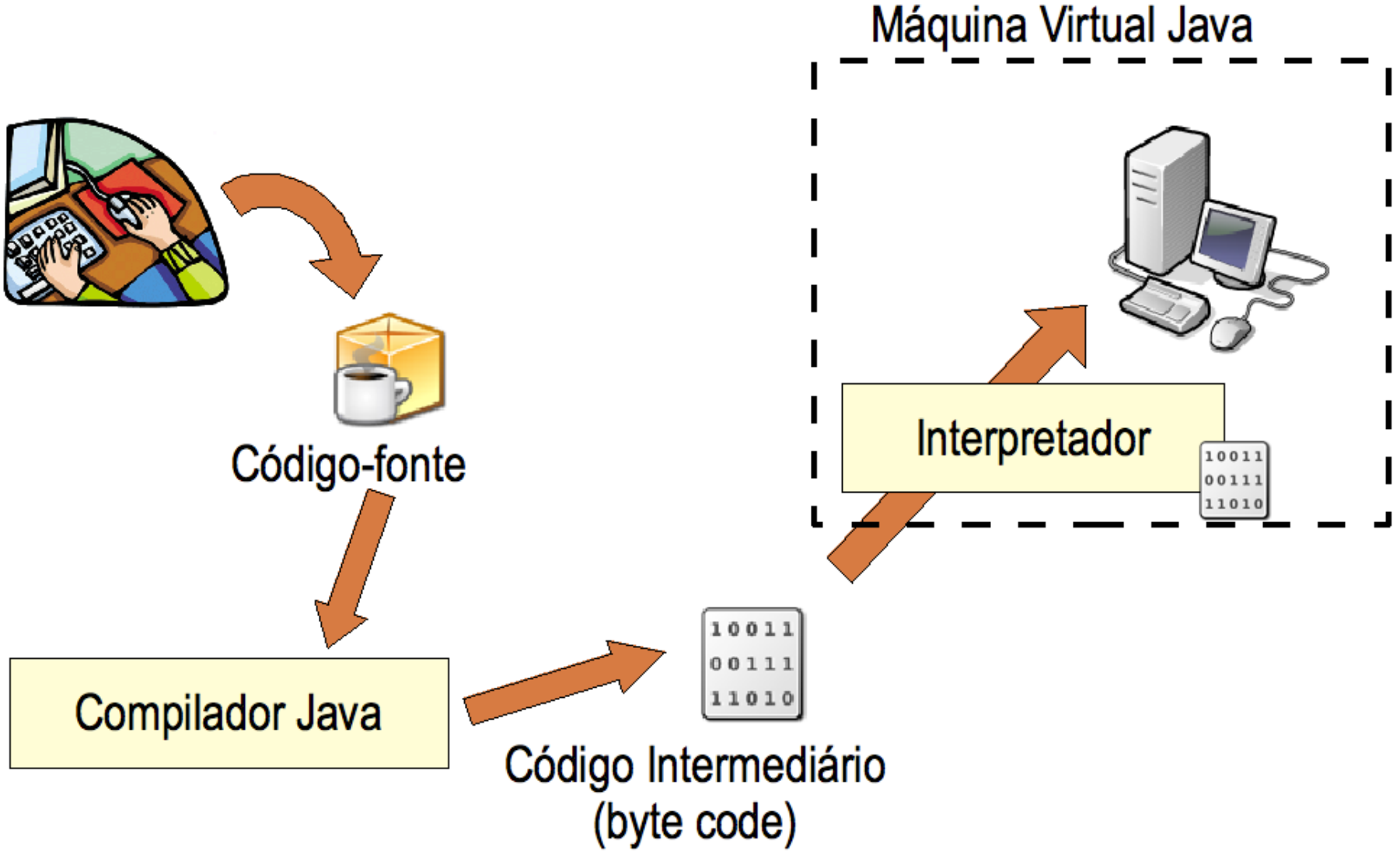
Implementação de LPs

- Interpretação:



Implementação de LPs

- Híbrido:

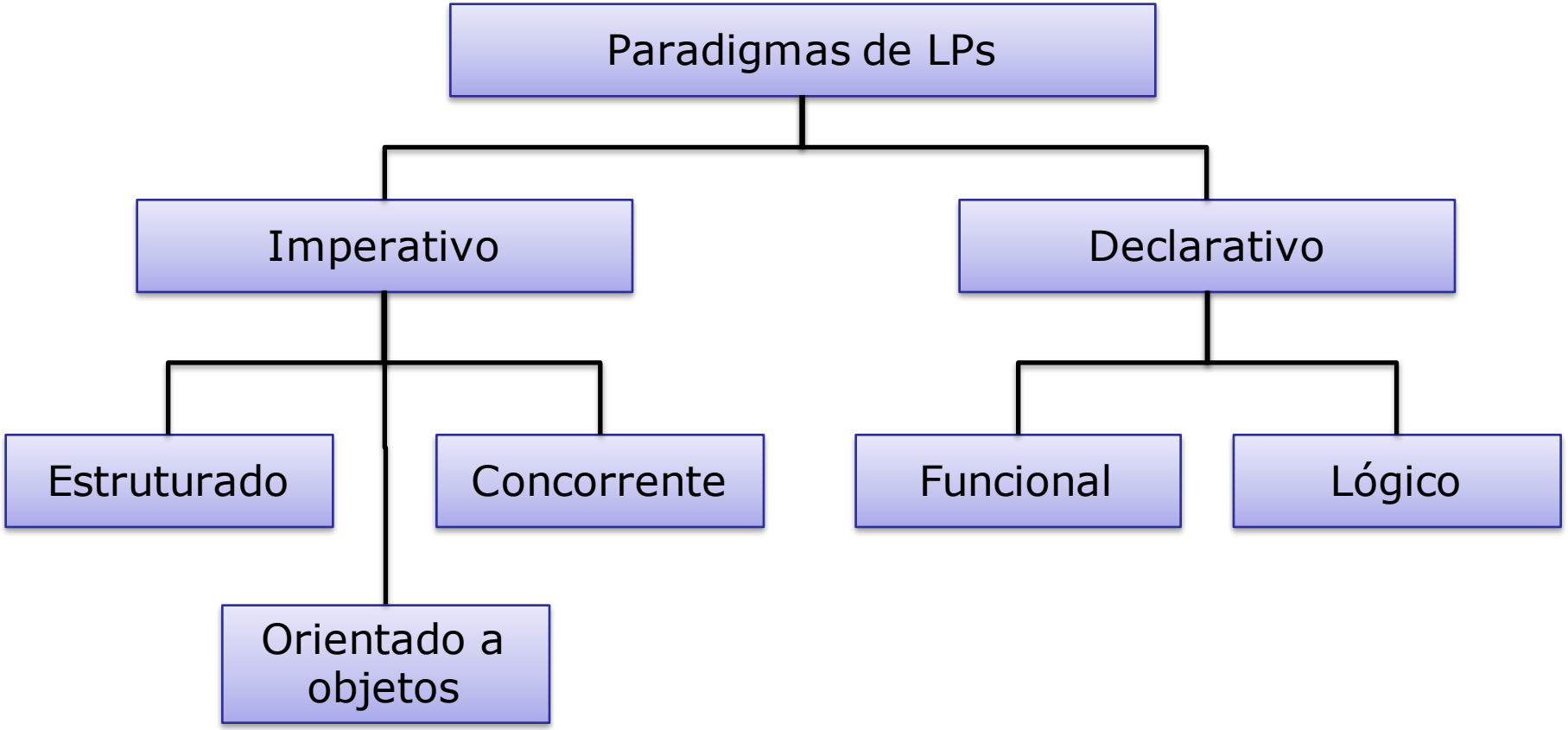


Implementação de LPs

- Compilação:
  - Eficiência;
  - Problemas com portabilidade e depuração;
- Interpretação pura:
  - Flexibilidade, portabilidade, facilidade para prototipação e depuração;
  - Problemas com eficiência e maior consumo de memória;
  - Raramente usada;
- Híbrido:
  - Une vantagens (e desvantagens) dos outros métodos;
  - JVM, JIT-compiler.

Paradigmas de LPs

- Segundo Varejão:

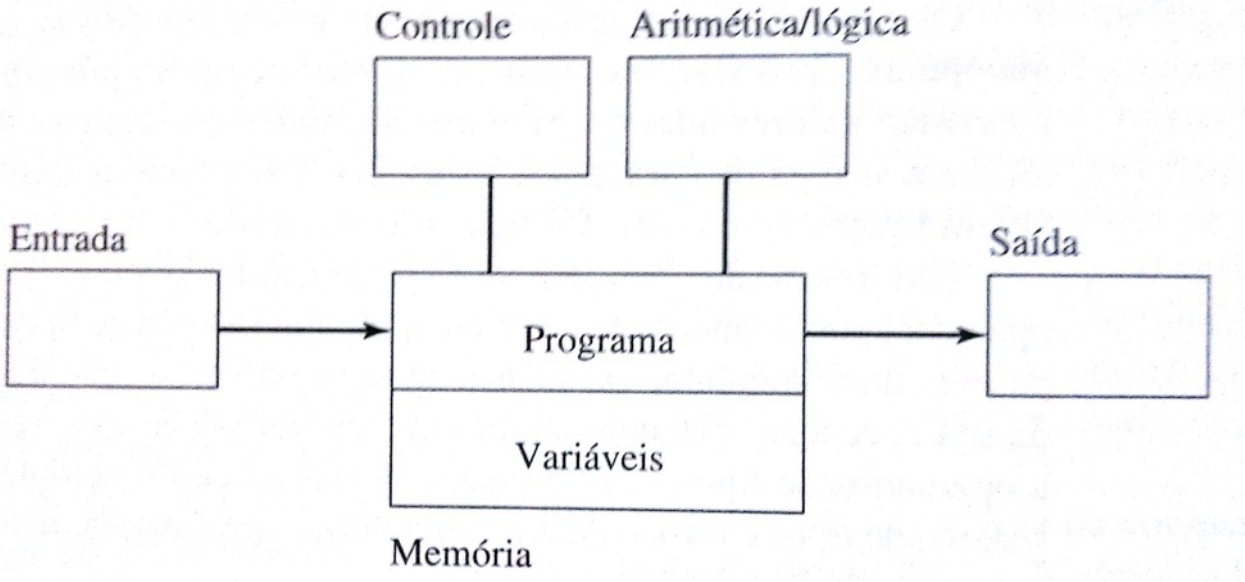


Paradigmas de LPs

- Segundo Tucker & Noonan, há quatro paradigmas:
  - Imperativo;
  - Orientado a objetos;
  - Funcional;
  - Lógico.
- Outras características tratadas como tópicos especiais:
  - Manipulação de eventos;
  - Concorrência;
  - Correção (especificação formal).

Paradigma imperativo

- Processo de mudança de estados;
- Variável, valor e atribuição;
- Células de memória;
- Arquitetura de von Neumann-Eckert: entrada-processamento-saída.

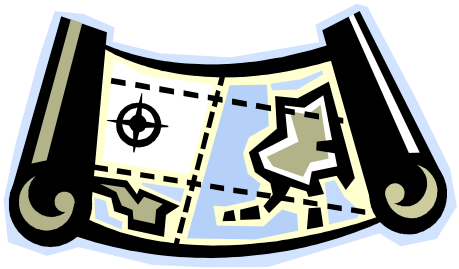


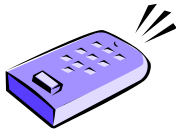
Paradigma estruturado

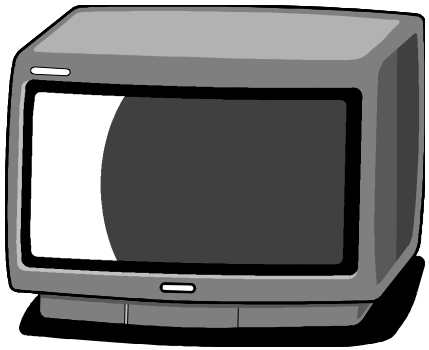
- Refinamentos sucessivos;
- Blocos aninhados de comandos;
- Desestímulo ao uso de desvio incondicional;
- Abordagem top-down, organizando o fluxo de controle.
- Exemplos: Pascal, C, Fortran, Cobol, Perl, etc.

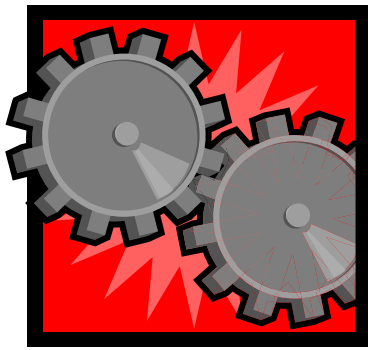
Paradigma Orientado a Objetos

- Abstração, encapsulamento, modularização;
- Herança e polimorfismo;
- Exemplos: Smalltalk, Java, C++, C#, Python, etc.









Paradigma declarativo

- Especificação sobre a tarefa a ser realizada;
- Abstrai-se como o computador é implementado;
- “Faça isso” (imperativo) vs. “É preciso chegar neste estado” (sem dizer como, declarativo);
- Em alguns aspectos mais próximo do que aprendemos na matemática, por exemplo:

```
x = x + 1
```

  - Atribuição ou equação sem solução?

Paradigma funcional

- Programa composto por funções;
- Funções compostas por outras funções;
- Exemplos: Lisp, Haskell.

```
(defun fatorial (n)
  (if (= n 0)
      1
      (* n (fatorial (- n 1)))))
```

Paradigma lógico

- Dedução automática, baseado em regras;
- Cláusulas definem predicados e relações factuais;
- Exemplo: Prolog.

```
% Fatos:
progenitor(maria,joao).
progenitor(jose,joao).
progenitor(joao,mario).

% Regras:
descendente(Y,X) :- progenitor(X,Y).
descendente(Y,X) :- progenitor(A,Y), descendente(A,X).

% Questões:
q1 :- descendente(mario,jose).
```

Evolução das LPs

- Dificuldade de programação em linguagem de máquina;
- Foco de primeiras LPs era eficiência de processamento e consumo de memória;
- A medida que o hardware evoluiu, o foco mudou para a baixa produtividade de programação, surgindo:
  - Programação estruturada;
  - Tipos abstratos de dados;
  - Orientação a objetos.

Origem das LPs

- Fortran (Formula Translator) – 1957, IBM:
  - Aplicações numéricas;
  - Eficiência computacional (ex.: não há alocação dinâmica);
- Lisp (List Processor) – 1959, MIT:
  - Programação funcional;
  - Processamento simbólico (IA);
  - Common Lisp, Scheme;

Origem das LPs

- Algol (Algorithmic Language) – 1960, ETH Zurich:
  - Programação estruturada;
  - 1ª LP com sintaxe formal definida;
  - Importância teórica (ALGOL-like);
- Cobol (Common Business Oriented Lang.) – 1960, DoD:
  - Aplicações comerciais;
  - Muitos dados, pouca computação;
- Basic – 1964, Darthmouth University:
  - Ensino para leigos;
  - Estudantes de artes e ciências humanas;
  - Precursora do Visual Basic, da Microsoft;

Origem das LPs

- Pascal – 1964, Niklaus Wirth (Stanford University):
  - Ensino de programação estruturada;
  - Simplicidade;
  - Precursora do Object Pascal e Delphi;
- C – 1972, Dennis Ritchie (AT&T Bell Labs):
  - Desenvolvimento de SOs / Unix;
  - Precursora do C++;
- Prolog – 1972, Unv. Aix-Marselha / Univ. Edinburgo:
  - Programação lógica;
  - Inteligência artificial;

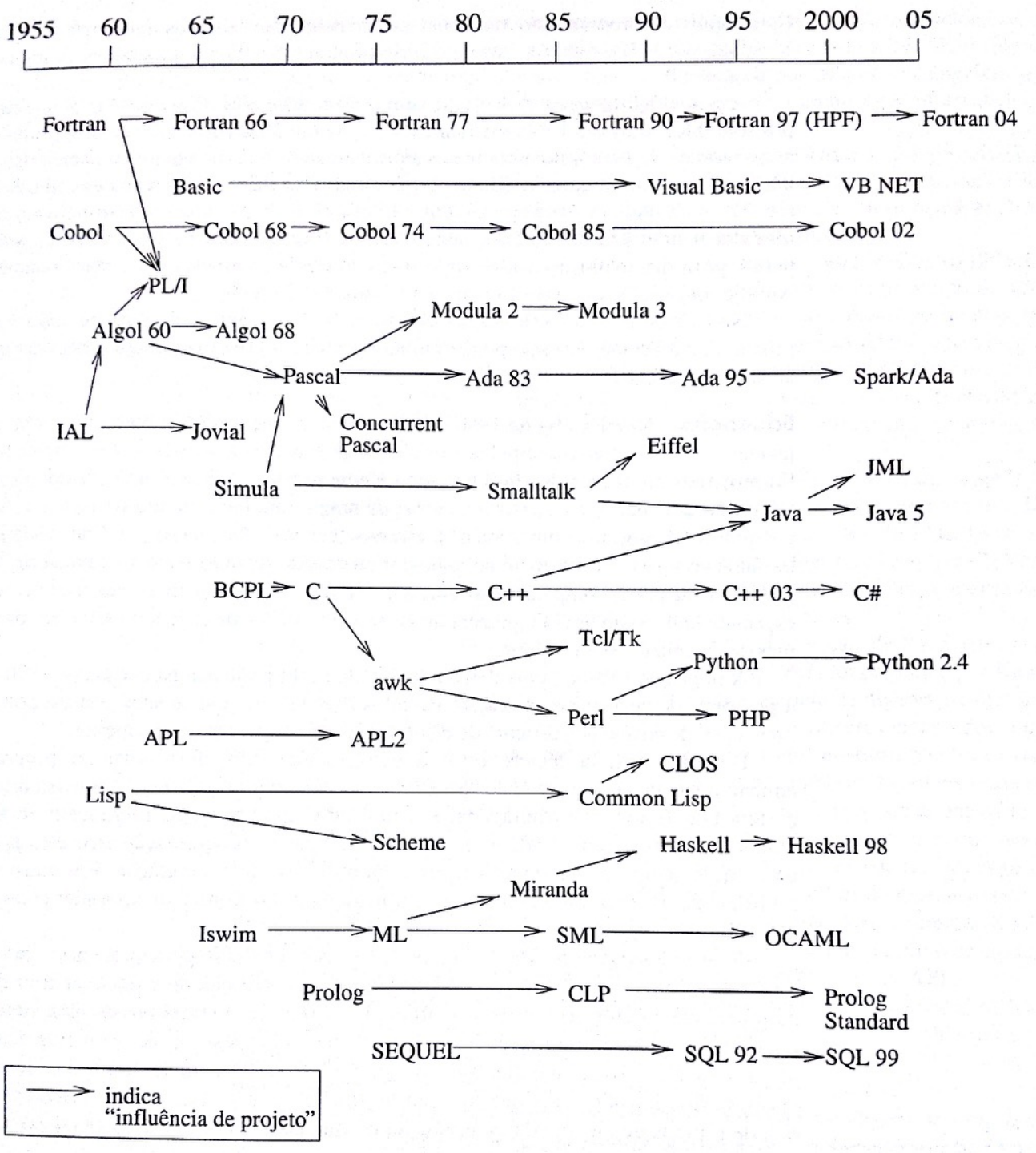
Origem das LPs

- Smalltalk – 1972, Xerox PARC:
  - Programação OO (1ª LP OO pura);
  - Inovações em GUIs;
- Ada (Lovelace) – 1983, DoD:
  - Programação concorrente;
  - Centenas de pessoas envolvidas durante 8 anos;
  - Grande e complexa;
  - Sistemas de tempo real;
- C++ – 1985, Bjarne Stroustrup (AT&T Bell Labs):
  - Disseminação da programação OO;
  - Linguagem complexa.

Origem das LPs

- Java – 1995, James Gosling (Sun Microsystems):
  - Hoje pertence à Oracle;
  - Criada com foco na Internet;
  - Mais simples e confiável que C++;
- Linguagens modernas:
  - C#;
  - Groovy;
  - Go;
  - Haskell;
  - Lua;
  - Perl;
  - PHP;
  - Python;
  - Ruby;
  - Scala;
  - Shell Script;
  - Swift;
  - E muitas outras...

Origem das LPs



http://nemo.inf.ufes.br/