UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BCC6002 - Aspectos de Linguagens de Programação

Prof. Dr. Rodrigo Hübner

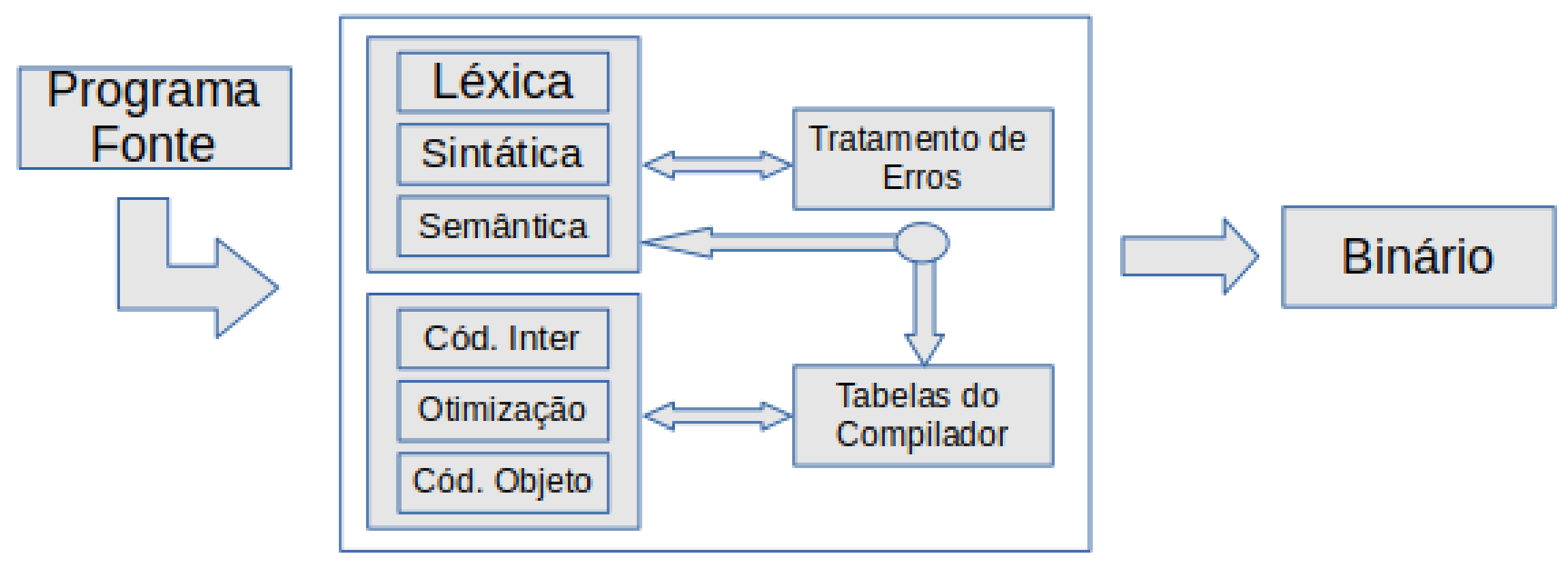
Aula 02: Conceitos de sistemas de computação

Conceitos de sistemas de computação

Nesta aula iremos falar sobre o conceito de:

- Compiladores
- *Assemblers*
- Tradutores
- *Linkers* (estático e dinâmico)
- *Loaders*
- Máquinas Virtuais
- *Just-in-Time* e *Ahead of Time*

Compiladores



Linguagens compiladas

- `C`, `C++`, `C#`, `Objective-C`, `Fortran`, `Go`, `Rust`, `Delphi (Object Pascal)`, `Pascal`, ...
  - Necessário "remontar" o programa sempre que necessitar realizar uma alteração
  - Liga bibliotecas já compiladas (*linker*)
  - Não é necessário um processo de análise e tradução toda vez que é executado

Assembler (Montador)

```
mov edx,2
mov esi,4
add eax,ebx
sub eax,ecx
imul  edx,eax
mov eax,edx
mov edx,0
cmp   esi,0
```

Assembly

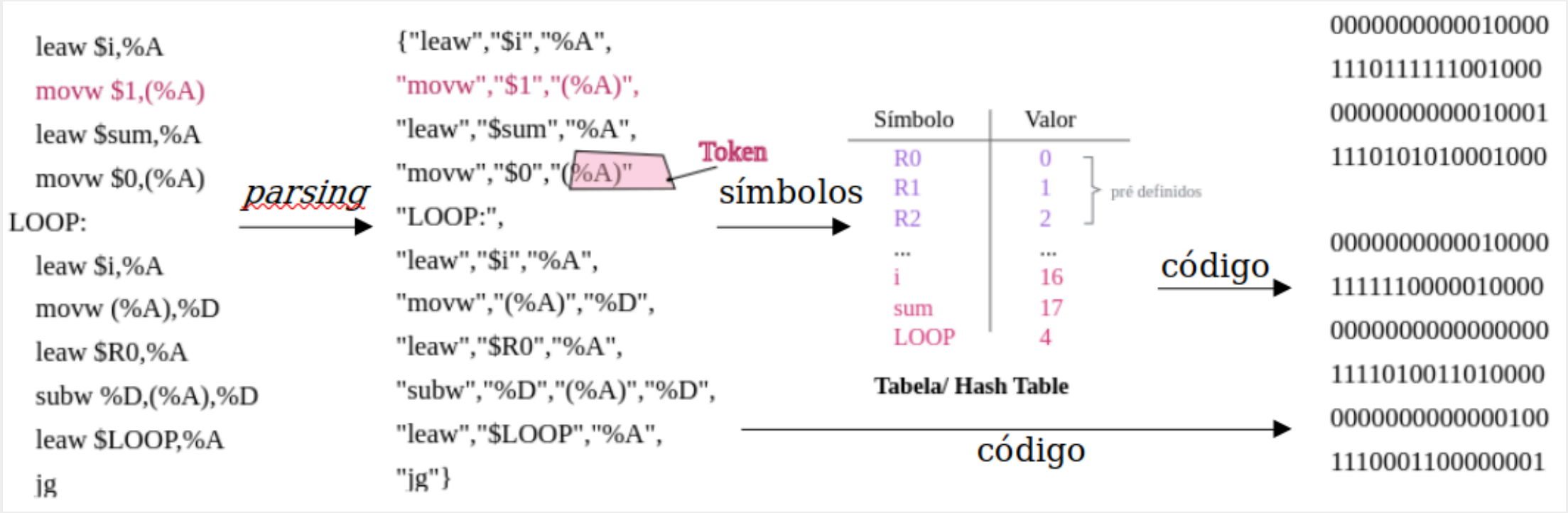
Assembler

```
ce fa ed fe 07 00 00 00 03 00 00 00 01 00 00 00
04 00 00 00 38 01 00 00 00 00 00 00 01 00 00 00
c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 28 00 00 00 54 01 00 00
28 00 00 00 07 00 00 00 07 00 00 00 02 00 00 00
00 00 00 00 5f 5f 74 65 78 74 00 00 00 00 00 00
00 00 00 00 5f 5f 54 45 58 54 00 00 00 00 00 00
00 00 00 00 00 00 00 00 1b 00 00 00 54 01 00 00
00 00 00 00 7c 01 00 00 02 00 00 00 00 04 00 80
00 00 00 00 00 00 00 00 5f 5f 64 61 74 61 00 00
00 00 00 00 00 00 00 00 5f 5f 44 41 54 41 00 00
00 00 00 00 00 00 00 00 1b 00 00 00 0d 00 00 00
6f 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 24 00 00 00
10 00 00 00 00 0a 0a 00 00 00 00 00 02 00 00 00
18 00 00 00 8c 01 00 00 04 00 00 00 bc 01 00 00
18 00 00 00 0b 00 00 00 50 00 00 00 00 00 00 00
02 00 00 00 02 00 00 00 01 00 00 00 03 00 00 00
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Executável

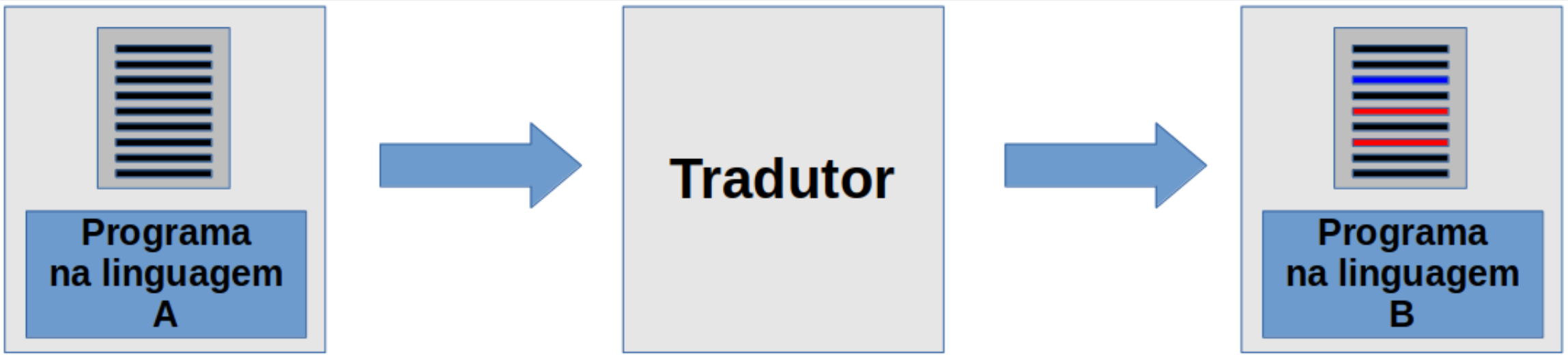
O *Assembly*

- Possibilita criar programas em linguagens mais "humanas"
- Necessário etapas de compilação: *parsing* e geração de código



Tradutores

- Realiza a tradução entre linguagens de alto-nível (geralmente)



Linguagens traduzidas

- `Python`, `PHP`, `Ruby`, `Javascript`, `TypeScript` `R`, `Lua`, `VBScript`, `ActionScript`, ...
  - Em toda execução o tradutor é invocado
  - Algumas bibliotecas podem estar pré-compiladas (CPython)
  - Passa pelo processo geral de compilação, mas não é um compilador (não gera código de máquina)
- **Tradução / Compilação** $\rightarrow$ Modo **híbrido**
  - **`Java`**, `Jython`, `Clang/Clang++`, `Julia`, ...

Linkers

- **Tradução pura** obriga o processo de recompilação de todo o código (*overhead*)
- Uma alternativa é ter diferentes partes compiladas separadamente (incluindo linguagens interpretadas)
- O ***linker*** "junta" blocos compilados em única solução
  - Coloca o código e os dados simbolicamente em memória
  - Determina os endereços dos rótulos de dados e instruções
  - Junta referências internas e externas

Linkers

- **Estático**
  - Cria um **executável único**
  - Mais fácil de gerenciar e instalar
  - Permite melhores otimizações
  - Não permite um sistema de *plugin*

Linkers

- **Dinâmico**
  - Partes são geradas separadamente e adicionadas à execução
  - As partes (`dll` ou `so`) já estão otimizadas
  - Substituir de forma independente (*patches*)
  - Fortemente dependente das bibliotecas e permite indireções

Falácia: somente *linker* dinâmico carrega o necessário na execução, porém, **SOs modernos fazem com que somente páginas importantes carreguem com o *linker* estático**.

Loaders

- Utiliza um executável "pronto" para:
  - Leitura do cabeçalho (tamanho de código e segmentos)
  - Alocação de memória necessária
  - Copia instruções e dados p/ memória
  - Inicia registros e *stack pointer* no primeiro local livre
  - Salto $\rightarrow$ rotina inicial carregando os argumentos para o programa inicial
    - Quando finaliza, o programa invoca um *system call* `exit`

Máquinas Virtuais (VMs)

- Duplica os recursos de hardware para um ambiente seguro, chamado de “virtual”. Exemplos:
  - **`JVM`**
    - `Quercus`, `jRuby`, `Nashorn`, `Clojure`, `Scala`, `Groovy`, `Jython`
  - **`LLVM`**
    - `Julia`, `Clang`, `Lua`, `CUDA`, `OpenCL`, `Objective-C`, `Swift`, `C#`, ...
  - **`GraalVM`**
  - Provê melhor migração e portabilidade de LPs
  - Execução não é necessariamente híbrida

Depuradores

- Ou **debugger**, consiste em encontrar problemas em código
- Pode interromper uma execução de máquina e ler endereços de registradores (VMs é mais flexível)
- Apesar da importância, **pode ser um problema**
  Incerteza de Heisenberg (conhecida como `Heisenbug`)
- Exemplos de *debuggers*:
  - `GDB`, `PDB`, `JSwat`, `Eclipse`, `Valgrind`, ...

Just-in-Time (`JIT`)

- É uma das possíveis melhores **combinações entre compilação e interpretação**
- Fornecido em VMs, onde a compilação p/ `bytecode` não é necessário para situações repetidas de dados
- Execução geralmente [bem] superior às LPs puramente interpretadas
- Pode gerar atrasos com a compilação bytecode $\rightarrow$ LP de máquina
- LPs mais comuns que fornecem `JIT`:
  - `JVM` ($>$ 4.0), `RPython`, `.NET`, `LLVM`, ...

Ahead of Time (`AOT`)

- Realiza a compilação em um passo anterior a execução
- Possui vantagens como:
  - Melhor desempenho de **arranque**
  - Menor sobrecarga de tempo de execução
  - Mais segurança (código não revertido)
- LPs que utilizam do `AOT`: `Rust`, `Go`, `Swift`, `Kotlin`, `Dart`, ...

Tarefas

- Instalar e testar `Clang` / `Clang++` (p/ comparar com `GCC`)
- Instalar e testar `PyPy` (p/ comparar com `CPython`)
- Instalar e testar `Dart` (p/ comparar com com `RPython`)
- Pesquisar pelo menos **dois** programas em `Python` e o mesmo em `C` / `C++` e `Dart` a seguir (além de entender o que fazem e o recurso mais explorado computacionalmente):
  - `n-body`
  - `Mandelbrot`
  - `binary-trees`

Próxima aula

- **Laboratório**: domínio de problemas, comparação de execução entre linguagens (conceito de *benchmark*).