SBVCONC: Construção de Compiladores

Aula 01: Apresentação da Disciplina e Visão Geral Sobre Construção de Compiladores

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Quem Sou Eu?

- Professor Dr. David Buzatto
  - Bacharel em Sistemas de Informação (UniFEOB);
  - Mestre em Ciência da Computação (UFSCar);
    - Reengenharia de Software, IHC e RIAs;
  - Doutor em Biotecnologia (UNAERP);
    - Bioinformática (Biologia Computacional);
      - Comparação Estrutural de Proteínas Tóxicas;
      - Relação entre Conformação e Atividade;
  - Contato exclusivamente via davidbuzatto@ifsp.edu.br

Apresentação da Disciplina

- Construção de Compiladores;
- 6 aulas semanais, durante 19 semanas, totalizando 114 aulas semestrais.

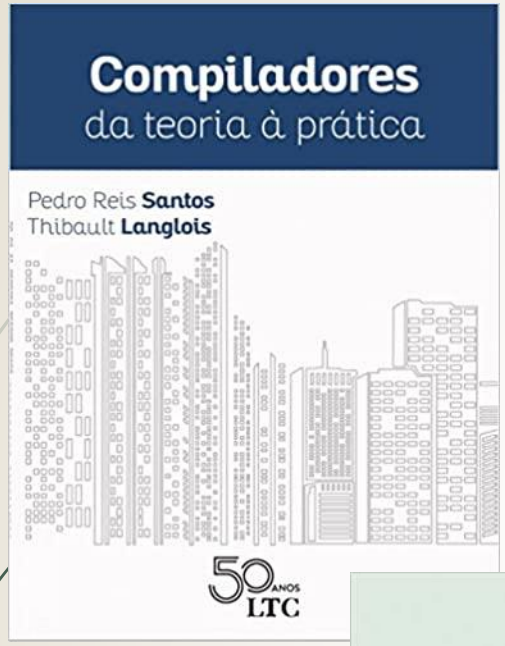
Logística
Critérios de Avaliação

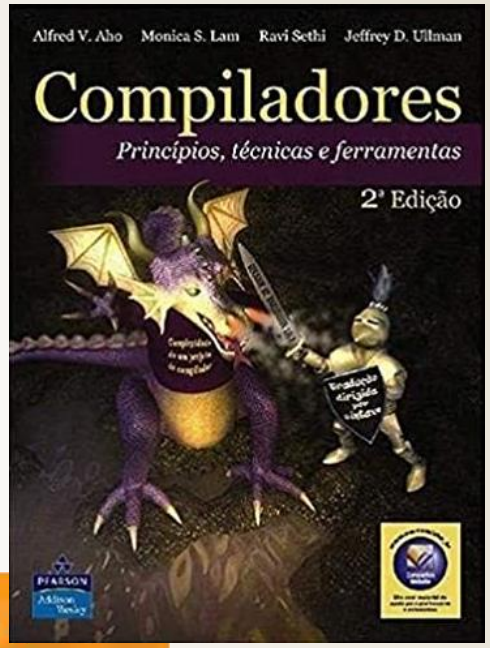
Onde:

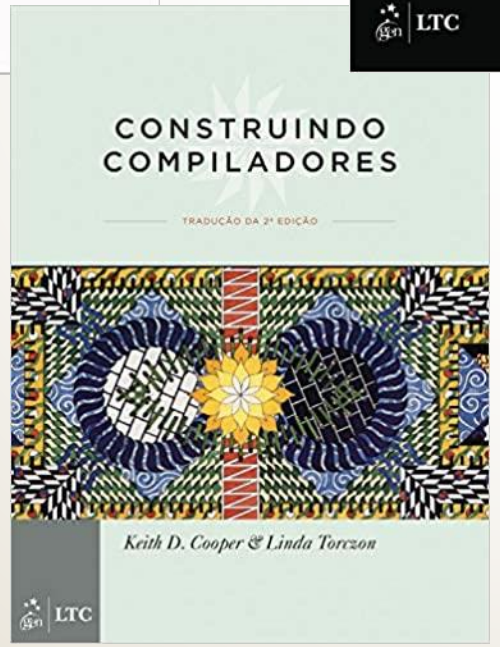
- M: média final;
- q: quantidade de agrupamentos temáticos;
  - A_i: nota da avaliação diagnóstica de um agrupamento temático i, sendo que $A_i = \{x \mid 0 \leq x \leq 10 \wedge x \in \mathbb{Q}\}$;
  - e_i: quantidade de listas de exercícios de um agrupamento temático i;
    - E_{j_i}: nota da lista de exercícios j de um agrupamento temático i, sendo que $E_{j_i} = \{x \mid 0 \leq x \leq 10 \wedge x \in \mathbb{Q}\}$;
- p: quantidade de projetos;
  - P_i: nota do projeto i, sendo que $P_i = \{x \mid 0 \leq x \leq 10 \wedge x \in \mathbb{Q}\}$;
- D: desafio opcional, onde somente o primeiro a entregar e a acertar ganha meio ponto. Se não acertar, o segundo a entregar é avaliado e assim por diante. Um aluno só pode ganhar uma vez por semestre.

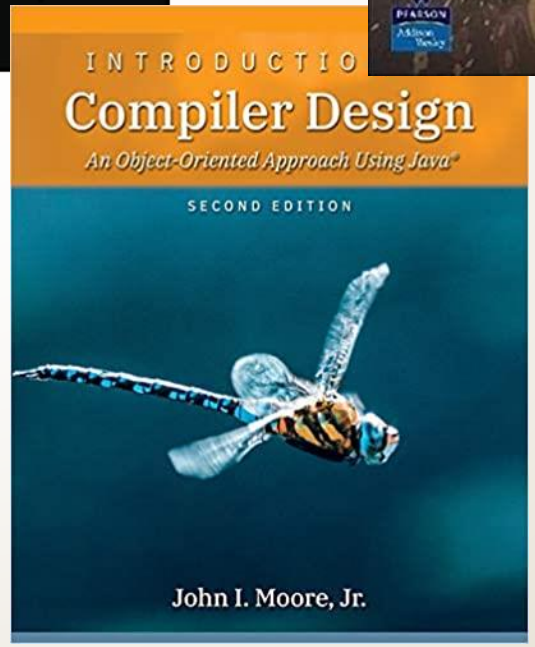
$$M = \begin{cases} \dfrac{\displaystyle\sum_{i=1}^{q}\left(\dfrac{A_i}{2} + \dfrac{A_i}{2} * \dfrac{\displaystyle\sum_{j_i=1}^{e_i} E_{j_i}}{e_i}\right)}{q} + D, & \text{se } p = 0 \\ \dfrac{\displaystyle\sum_{i=1}^{q}\left(\dfrac{A_i}{2} + \dfrac{A_i}{2} * \dfrac{\displaystyle\sum_{j_i=1}^{e_i} E_{j_i}}{e_i}\right)}{q} + \dfrac{\displaystyle\sum_{i=1}^{p} P_i}{p}}{2} + D, & \text{se } p > 0 \end{cases}, q \in \mathbb{N}^* \wedge p \in \mathbb{N}$$

Bibliografia









Bibliografia

Nesta iteração da disciplina, grande parte do material será baseada nesta obra, inclusive o conteúdo de vários slides e projetos! Não se preocupe, todo o material será oferecido!

Apresentação da Disciplina

- Se nosso objetivo fosse esgotar o assunto sobre compiladores, teríamos que ter diversas disciplinas, tanto em nível de graduação, quanto de pós-graduação, somente com esse enfoque, sendo assim, algumas simplificações e o estabelecimento de alguns compromissos serão necessários para que possamos encaixar o conteúdo principal em um curso de graduação.

Apresentação da Disciplina

- Curso orientado a projetos, ou seja, a parte "divertida" de estudar compiladores;
- Definição de uma linguagem de programação com código fonte simples, mas poderosa o bastante para ser interessante e desafiadora;
- A linguagem alvo é uma linguagem de montagem para uma máquina virtual com arquitetura de pilha, pois a geração de código é mais simples, além de eliminarmos a necessidade de lidar com registradores de propósito geral;
- Não utilizaremos nenhuma ferramenta além da linguagem Java e do NetBeans como IDE, ou seja, tudo será implementado manualmente!

Apresentação da Disciplina

- Análise sintática descendente recursiva, usando um *token* de *lookahead* ($LL(1)$)e representação intermediária aplicando árvores de análise sintática;
- Implementaremos a linguagem **CPRL** (**C***ompiler* ***PR****oject* ***L****anguage*), projetada para ensinar construção de compiladores, utilizando a linguagem de programação Java;
- Máquina virtual simples, mas similar à JVM (*Java Virtual Machine*), chamada CVM (CPRL *Virtual Machine*).

Uma Observação Sobre Analisadores Descendentes Recursivos $LL(1)$

This pattern shows how to implement parsing decisions that use a single token of lookahead. It's the weakest form of recursive-descent parser, but the easiest to understand and implement. If you can conveniently implement your language with this LL(1) pattern you should do so. – *Terence Parr*, criador do ANTLR (ANother Tool for Language Recognition) (https://www.antlr.org/)

Projeto

- Implementação de um compilador para uma linguagem de programação pequena;
  - Linguagem fonte simples (CPRL);
  - Linguagem alvo simples (linguagem *assembly* para a CVM, uma máquina virtual simples baseada em pilha);
  - Construção do compilador, um passo por vez:
    - Série de 7 pequenos subprojetos;
  - Várias implementações de modelo para guiá-los no processo;
  - Ao fim da disciplina, caso haja tempo, implementaremos um compilador para uma outra linguagem que definiremos oportunamente, usando para isso o ANTLR.

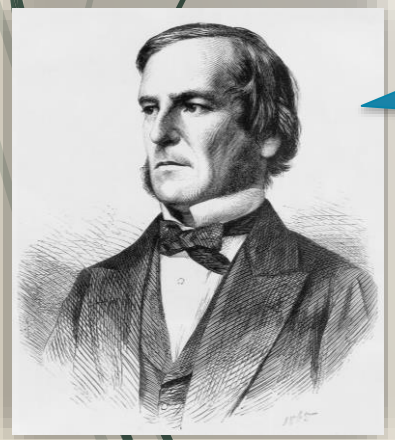
Recursos para a Implementação do Compilador

- Código fonte em Java para:
  - CVM (máquina virtual);
  - *Assembler* para a CVM;
  - *Disassembler* para o código de máquina da CVM;
- Esqueletos de código fonte em Java para várias partes do compilador;
- Documentação da linguagem CPRL para um compilador completo;
- Conjunto de programas corretos e incorretos escritos em CPRL para testar suas implementações;
- Todos os testes e execuções serão feitos dentro do NetBeans.

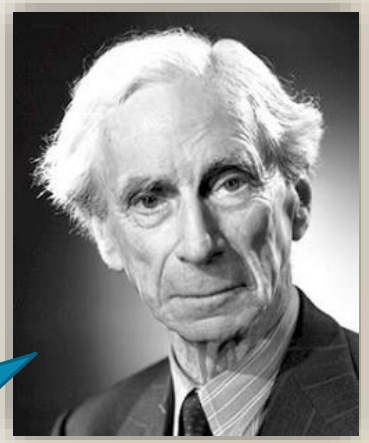
CONC5: Construção de Compiladores

Linguagens de Programação

- Meios de comunicação entre pessoa/pessoa e pessoa/máquina;
- Provem um *framework* para a formulação da solução de um problema na forma de um *software*;
- Podem aumentar ou inibir a criatividade;
- Influenciam as formas que pensamos sobre o projeto de *software* fazendo com que algumas estruturas de um programa sejam mais fáceis de descrever do que outras.



"Language is an instrument of human reason, and not merely a medium for the expression of thought." – George Boole



"By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems." – Bertrand Russell

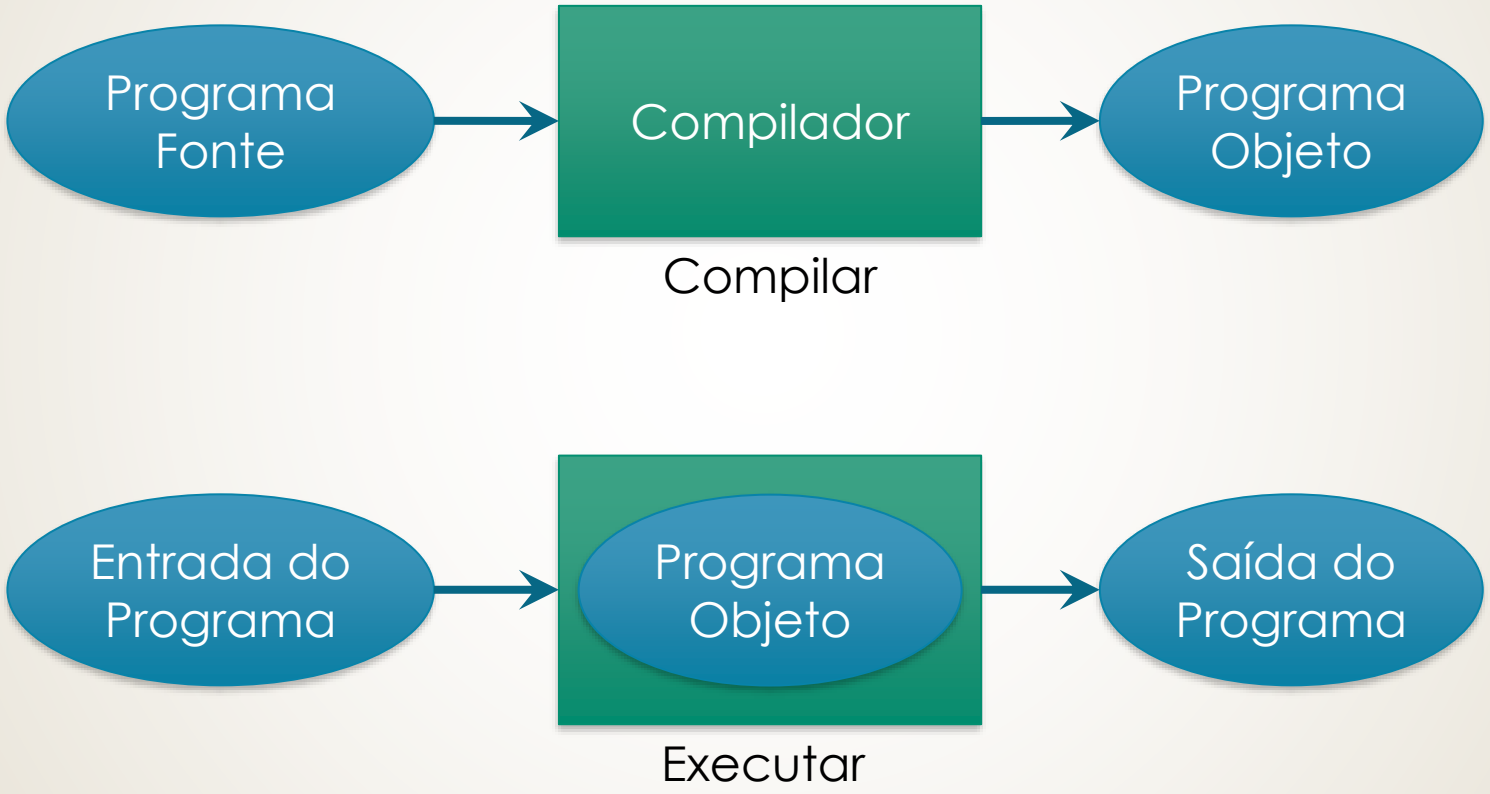
O Papel das Linguagens de Programação

- Independência de Máquina;
- Portabilidade;
- Reuso;
- Abstração;
- Comunicação de ideias;
- Produtividade;
- Confiabilidade (detecção de erros).

Tradutores e Compiladores

- No contexto de linguagens de programação, um **tradutor** é um programa que aceita como entrada um texto escrito em uma linguagem, chamada de linguagem fonte, e o converte em uma representação semanticamente equivalente em outra linguagem, esta denominada linguagem alvo ou linguagem objeto;
- Se a linguagem fonte é uma linguagem de alto nível e a linguagem alvo é uma linguagem de baixo nível, então o tradutor é chamado de **compilador**.

Visão Simplificada do Ciclo Compilar/Executar



Linguagem versus Implementação

Linguagem

- Identificadores podem ter um número arbitrário de caracteres;
- Tipos integrais com uma quantidade arbitrária de dígitos;
- A precisão dos tipos de ponto flutuante não é especificada.

Implementação

- Pode restringir a quantidade de caracteres significantes;
- Pode restringir o intervalo válido para os tipos integrais;
- A precisão de tipos de ponto flutuante é, normalmente, determinada pela máquina.

O Papel dos Compiladores

- Um compilador precisa, primeiramente, verificar se o programa fonte é válido em relação à definição da linguagem fonte;
- Se o programa fonte for válido, o compilador precisa produzir um programa em linguagem de máquina -do computador alvo- que seja **semanticamente equivalente** ao programa fonte e **razoavelmente eficiente**;
- Se o programa fonte não for válido, o compilador precisa fornecer ao programador algum tipo *feedback* que o informe, de forma inteligível, a natureza e a localização de quaisquer erros que possam ter ocorrido.

Outros Processadores de Linguagens

- Montadores (*Assembler*):
  - Traduz uma linguagem de montagem simbólica (*assembly*) em código de máquina;
- Tradutores entre linguagens de alto nível (*transpiler*):
  - C++ para C, TypeScript para JavaScript etc;
- Interpretadores;
- Ferramentas de teste ou reengenharia;
- Pré-processadores de macros;
- Desmontadores (*Disassemblers*);
- Descompiladores (*Decompilers*).

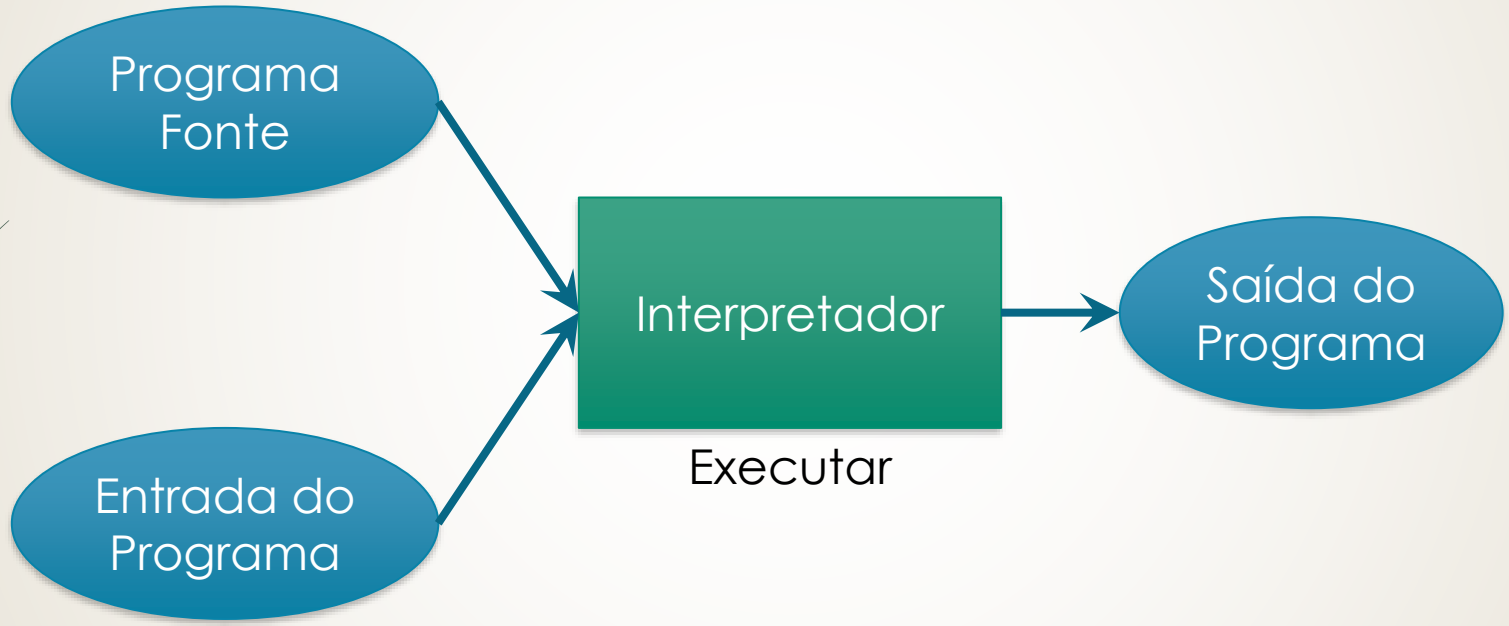
Ambientes Integrados de Desenvolvimento (IDE)

- Editores dirigidos à sintaxe;
- Formatadores de código fonte;
- Relatório de erros;
- Refatoração;
- Depurador em nível de código;
- Analisador (*profiler*) de tempo de execução.

Interpretadores

- Traduz e executa as instruções do programa fonte imediatamente, uma linha por vez;
- Não analisa nem traduz o programa inteiro antes de iniciar a execução, pois a tradução é feita toda vez que o programa for executado;
- O programa fonte é tratando basicamente como uma forma de entrada de dados ao interpretador;
  - O controle reside no interpretador, não no programa no usuário;
  - O programa do usuário é passivo ao invés de ativo;
- Alguns interpretadores executam traduções sintáticas elementares, como comprimir palavras chave em códigos de operação usando um *byte*.

Visão Simplificada de um Interpretador



Exemplos de Interpretadores

- Interpretadores das linguagens BASIC e Lisp;
- Laço de leitura-avaliação-impressão (*Read-Eval-Print Loop (REPL)*) para linguagens de programação:
  - JShell da linguagem Java ou kotlinc-jvm da linguagem Kotlin;
- *Java Virtual Machine* (JVM):
  - Java é compilada em código intermediário, chamado de *Java bytecode*, que é interpretado pela JVM;
- Interpretadores de comando de sistemas operacionais:
  - Vários *shells* Unix (sh, csh, bash etc), capazes de executar arquivos de *script* (*shell scripts*);
  - Prompt de comando do Windows, capaz de executar arquivos em lote (*batch files*);
- Interpretador SQL.

Compiladores versus Interpretadores

- Compilação:
  - Processo de duas etapas (compilar e executar);
  - Melhor detecção de erros;
  - Programas compilados executam mais rápido;
- Interpretação:
  - Processo de um passo (executar);
  - Provê *feedback* rápido ao usuário;
  - Bom para prototipação;
  - Sistemas altamente interativos;
  - Penalidade na performance.

Emuladores

- Um **emulador** ou **máquina virtual** é um interpretador do conjunto de instruções de uma máquina. A máquina que é emulada pode ser real ou hipotética;
- Da mesma forma que uma máquina real, os emuladores tipicamente usam um ponteiro de instrução (*program counter*) e um ciclo busca-decodifica-executa (*fetch-decode-execute*);
- Executar um programa em um emulador é funcionalmente equivalente a executar o programa diretamente na máquina, mas o programa terá um certo grau de degradação de performance em um emulador;
- Uma máquina real pode ser vista como um interpretador implementado em *hardware*. Por outro lado, um emulador pode ser visto como uma máquina implementada em *software*.

Compiladores Interpretativos

- Um **compilador interpretativo** ou **compilador-interpretador** é uma combinação de um compilador e um interpretador de baixo nível (emulador). O compilador traduz o programa usando o conjunto de instruções do emulador que, por sua vez, é utilizado para executar o programa compilado;
- **Exemplo:** Oracle/Sun Java Development Kit
  - javac é o compilador;
  - java é o emulador da JVM.

Compiladores *Just-In-Time*

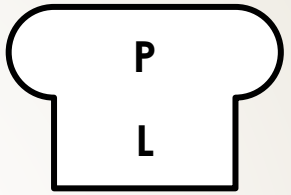
- Um **compilador *Just-In-Time* (JIT)** é um compilador que converte o código do programa fonte em código de máquina nativo enquanto o programa é executado;
- A plataforma Java fornece um compilador JIT junto à JVM que traduz *bytecode* Java em código de máquina nativo. O uso do compilador JIT é opcional;
- A tradução de uma método é feita quando o método é executado pela primeira vez;
- Métodos que são executados repetidamente terão sua performance aumentada significativamente.

Escrevendo um Compilador

- Escrever um compilador envolve três linguagens:
  - **Linguagem Fonte:**
    - Entrada do compilador;
  - **Linguagem de Implementação:**
    - A linguagem usada para escrever o compilador;
  - **Linguagem Alvo:**
    - Saída do compilador.

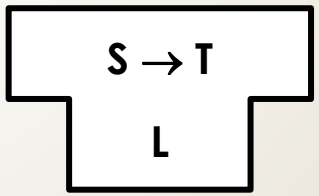
Diagramas Lápide (T-Diagramas)

- Programa **P** expresso na linguagem **L**, que pode ser uma linguagem de máquina:

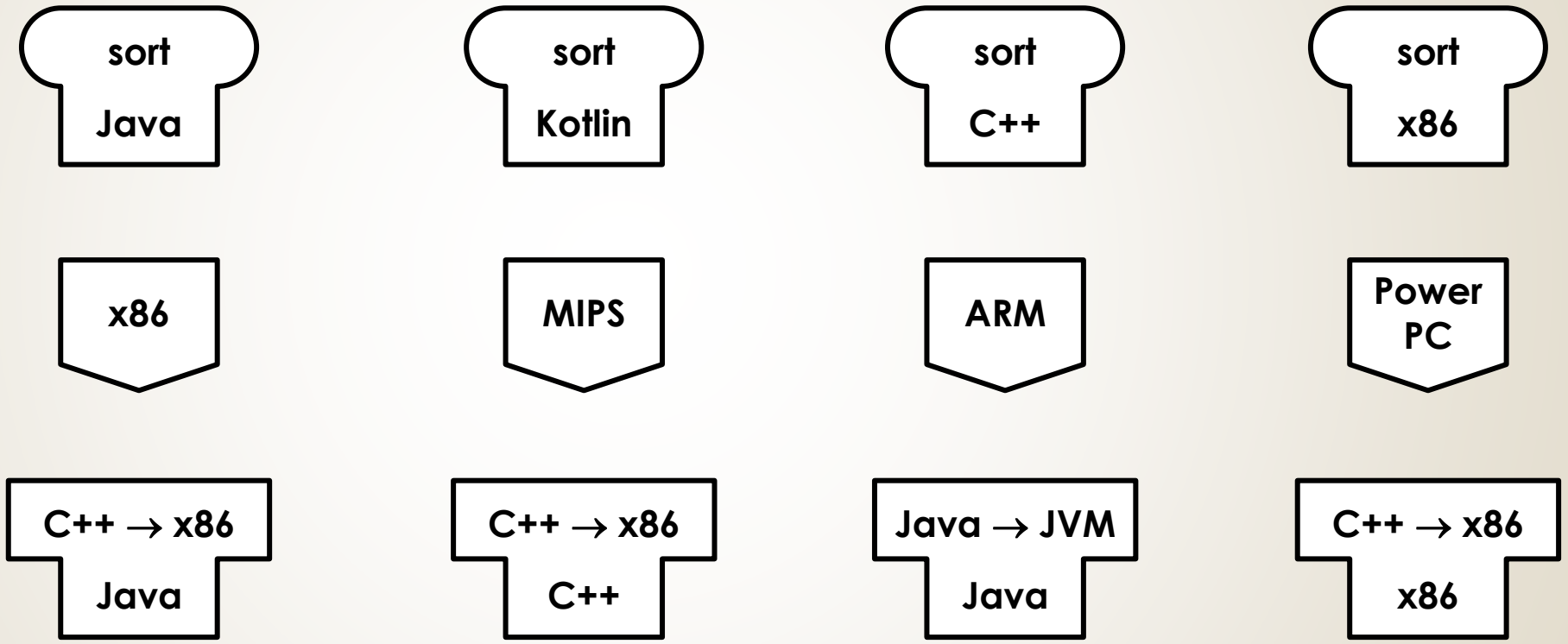


- Máquina **M**:

- Tradutor **S**-a-**T** expresso na linguagem **L**, que pode ser uma linguagem de máquina:

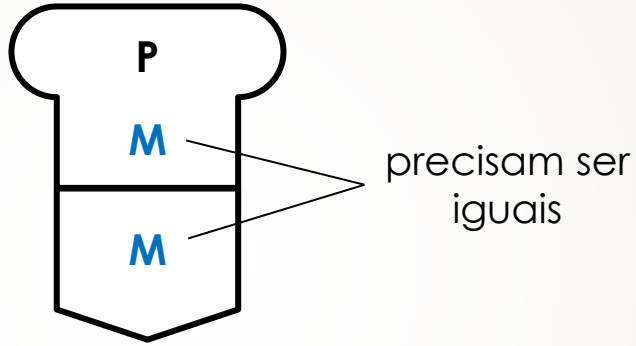


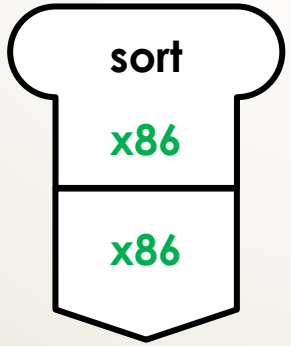
Exemplos de T-Diagramas

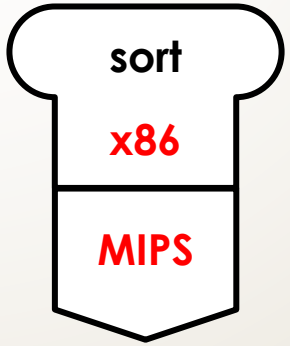


Exemplos de T-Diagramas

- Executando o programa **P** na máquina **M**:

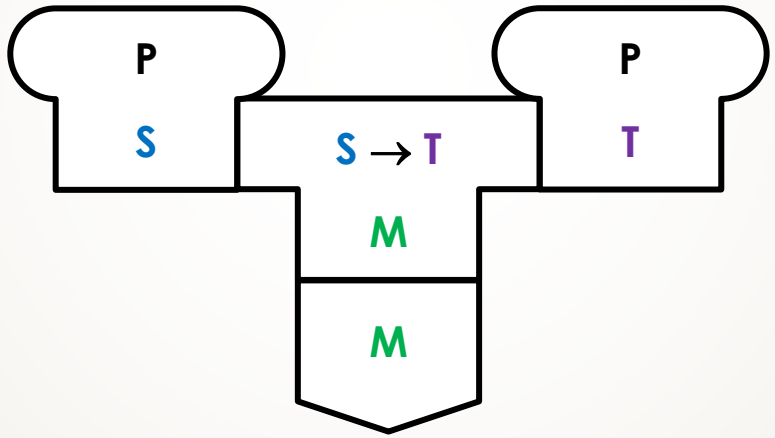






Exemplos de T-Diagramas

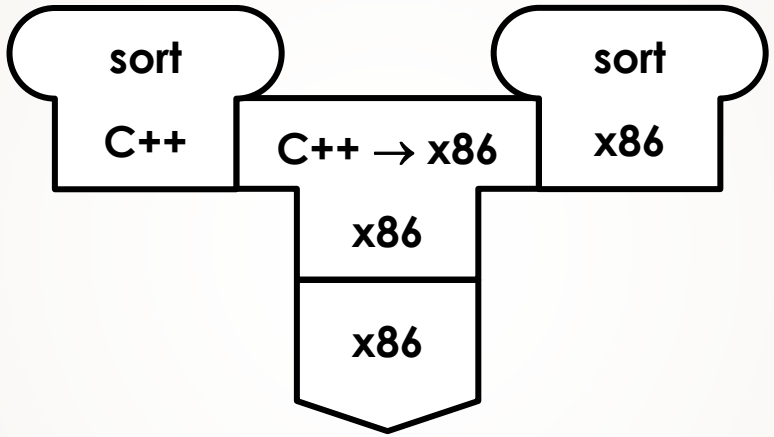
- Compilando um programa:



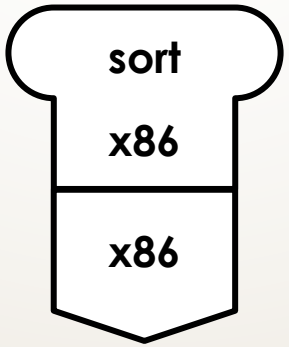
Exemplos de T-Diagramas

- Exemplo compilando e executando um programa:

- **Compilando:**

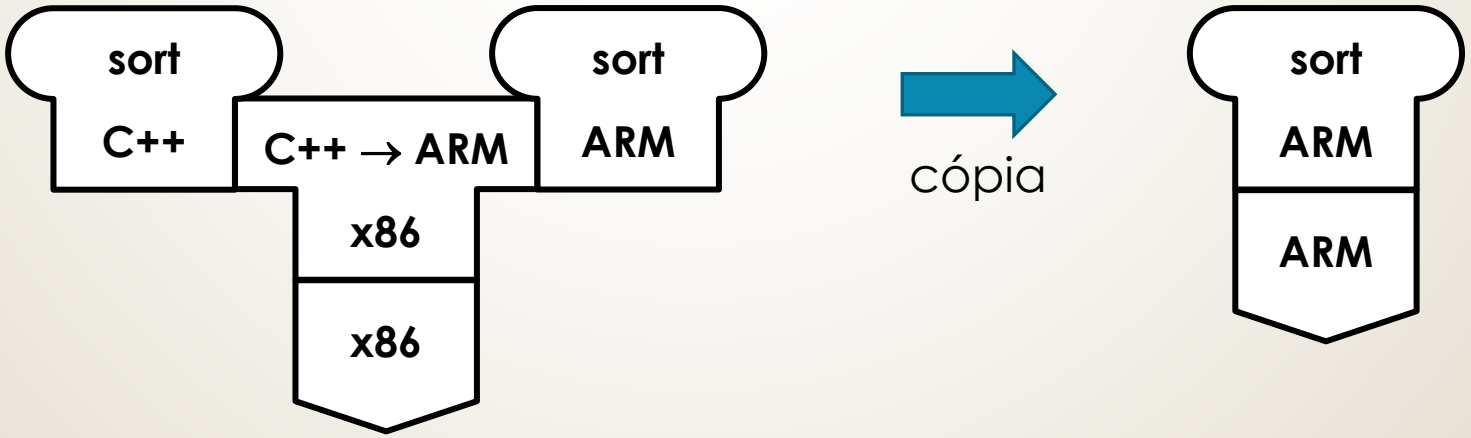


- **Executando:**

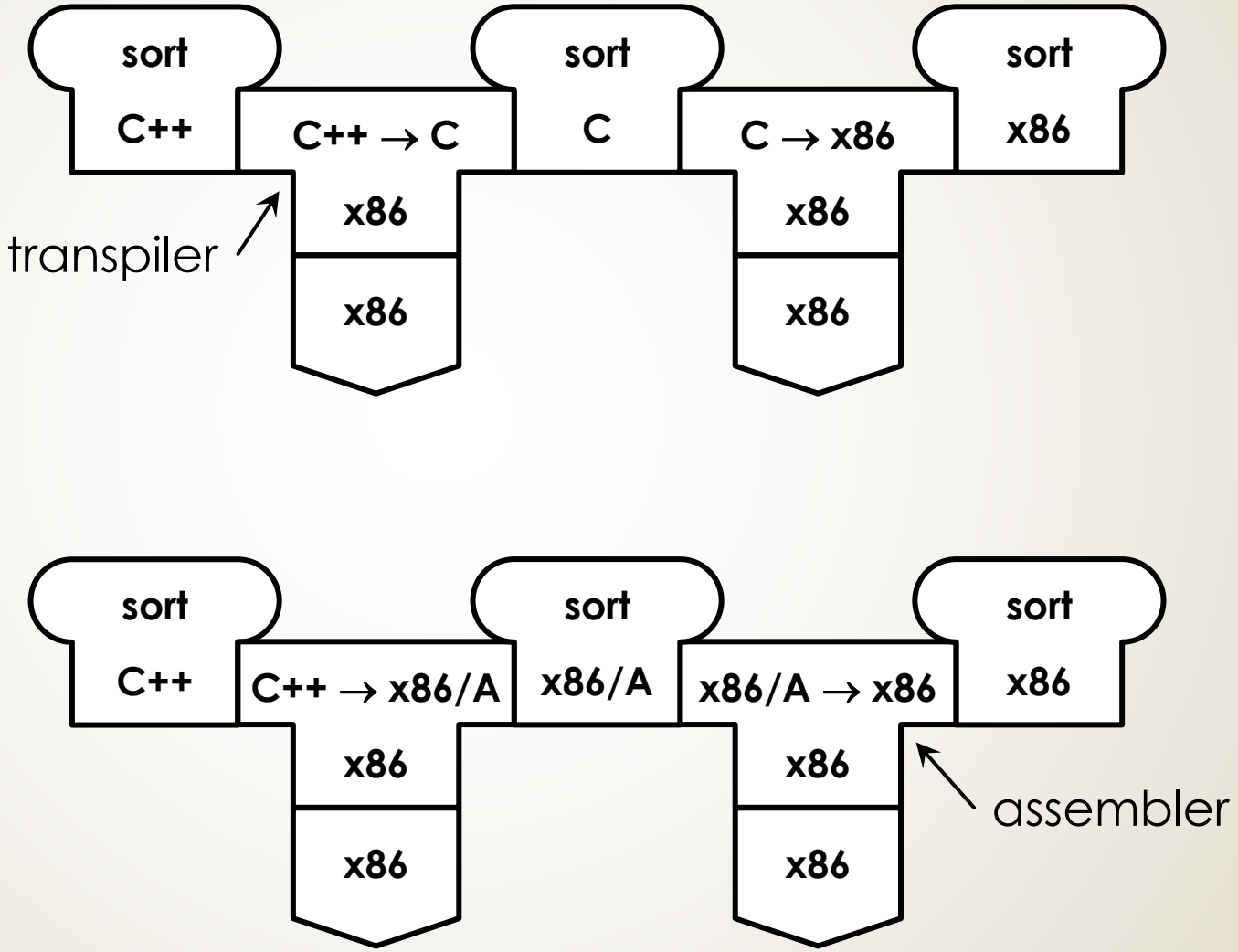


Compilador Cruzado (*Cross-Compiler*)

- Um compilador cruzado roda em uma máquina e gera código alvo para uma máquina diferente;
- A saída do compilador cruzado deve ser copiada/baixada para a máquina alvo para então ser executada;
- Normalmente usado no desenvolvimento mobile ou em sistemas embarcados.



Compilador de Dois Estágios

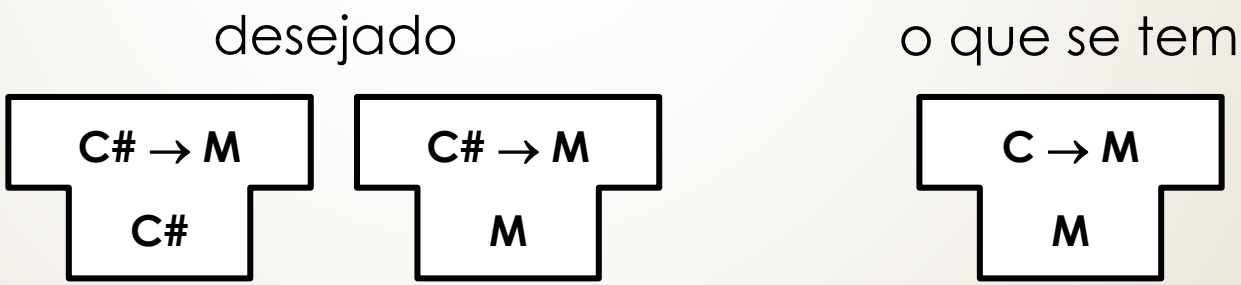


Usando a Linguagem Fonte como Linguagem de Implementação

- É comum se escrever um compilador na linguagem em que ele compilará, ou seja, escrever um compilador da linguagem C++ usando a linguagem C++;
- **Vantagens:**
  - Teste não trivial da linguagem compilada;
  - Utilização de apenas uma linguagem pelos desenvolvedores;
  - Apenas um compilador precisa ser mantido;
  - Se forem feitas alterações no compilador que aumentam a performance, então recompilando o compilador aumentará sua performance;
- Para uma linguagem de programação nova, como escreveríamos um compilador dessa linguagem na própria linguagem? Quem nasceu primeiro, o ovo ou a galinha?

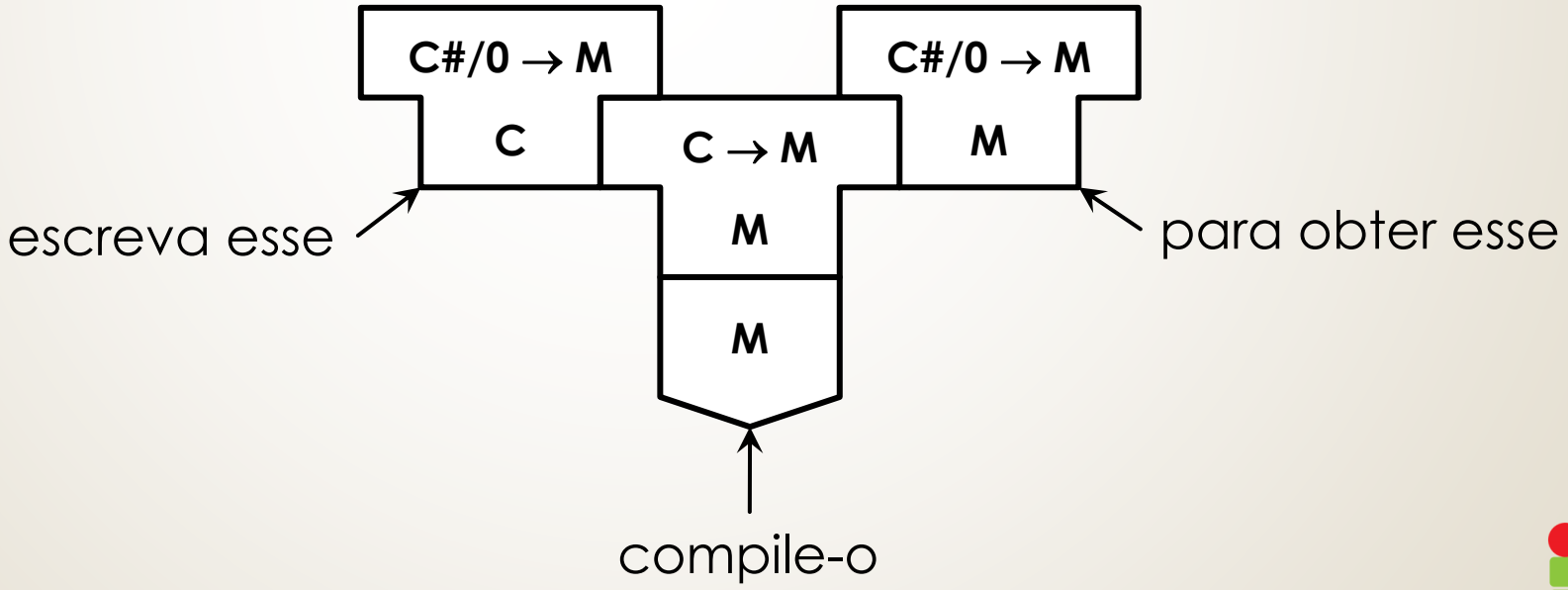
O *Bootstrapping* de um Compilador

- **Problema:** suponha que queremos construir um compilador para a linguagem C# que será executado na máquina M. Além disso, assumimos que já existe um compilador que é executado na máquina M, talvez para a linguagem C. Ainda, desejamos que o código fonte do compilador da linguagem C# seja escrito em C#.



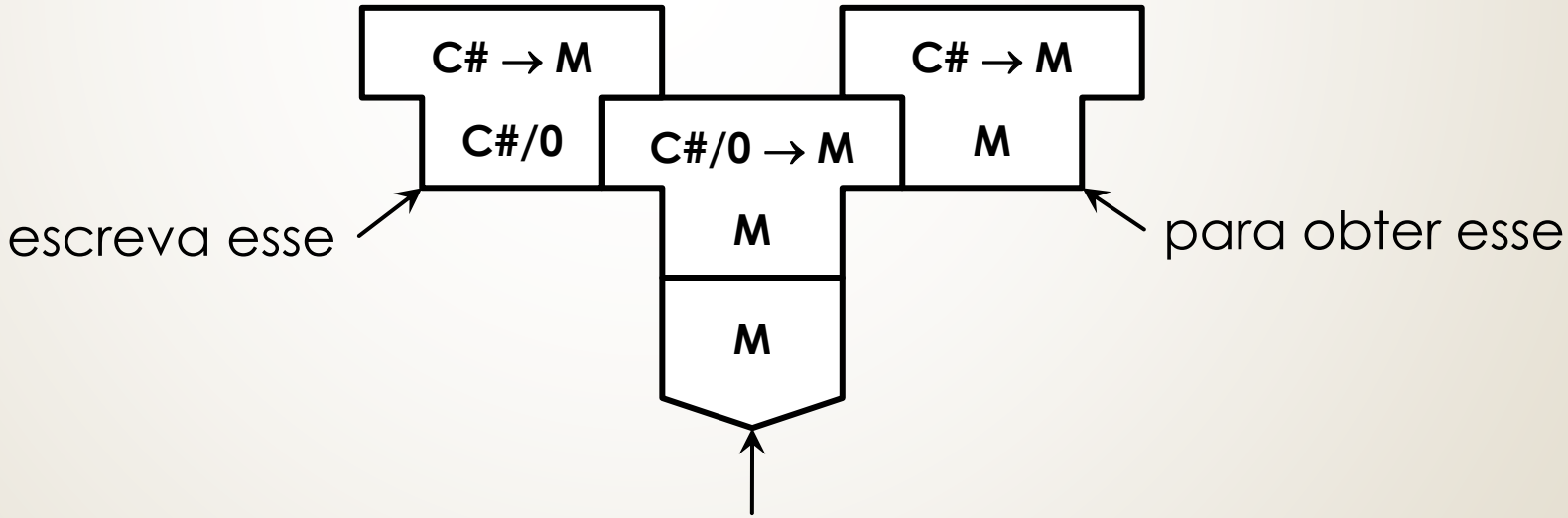
O *Bootstrapping* de um Compilador
Passo 1

- Primeiramente, selecionamos um subconjunto de C# (chamaremos de C#/0) que é suficientemente completo para escrever um compilador;
- Escrevemos um compilador para a linguagem C#/0 em C e o compilamos:



O *Bootstrapping* de um Compilador
Passo 2

- Agora, escrevemos o compilador completo da linguagem C# em C#/0;
- Compilamos esse compilador usando o compilador obtido no passo 1:

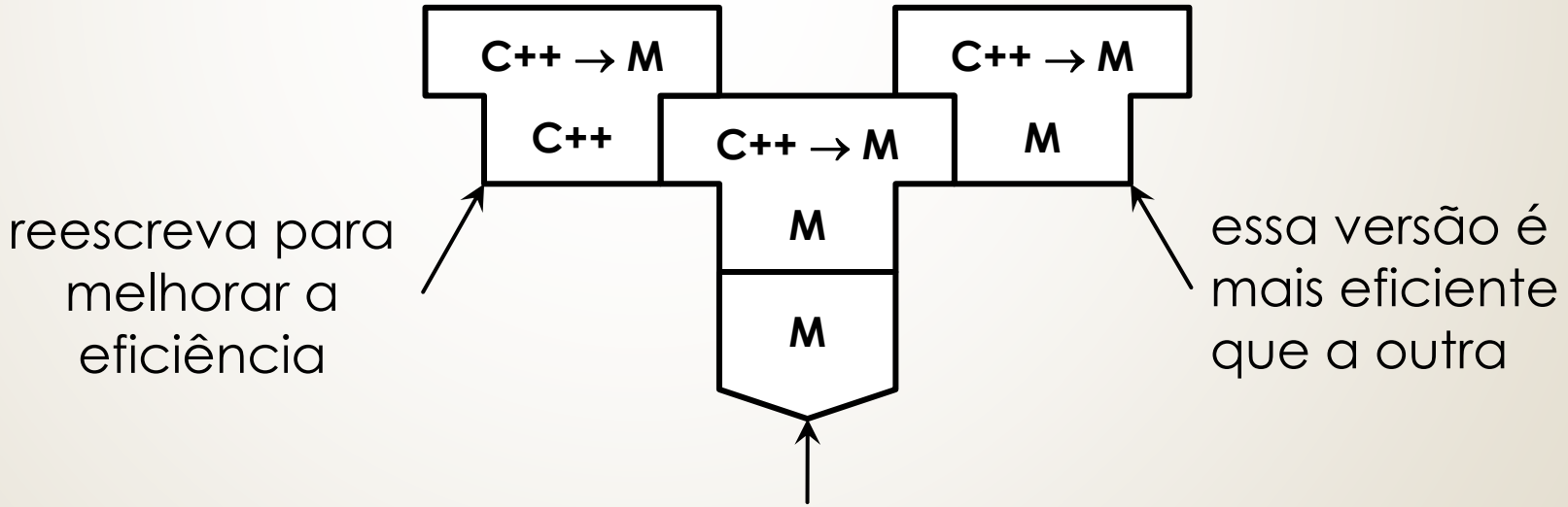


Eficiência

- **Eficiência de um programa:**
  - Velocidade;
  - Uso de memória;
- **Eficiência de um compilador:**
  - Eficiência do próprio compilador;
  - Eficiência do código objeto que ele gera.

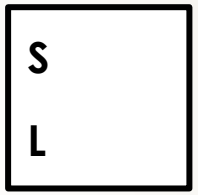
Melhorando a Eficiência de um Compilador

- Suponha que temos o compilador de uma linguagem, digamos C++, escrito em C++;
- Se modificarmos o compilador para aumentar a eficiência do código objeto gerado, então podemos recompilar o compilador para obter uma versão mais eficiente:

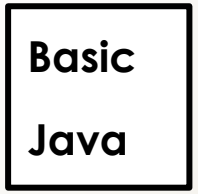


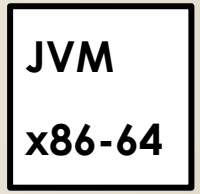
T-Diagrama de um Interpretador

- Um interpretador para a linguagem **S** expresso na linguagem **L**, que pode ser uma linguagem de máquina:

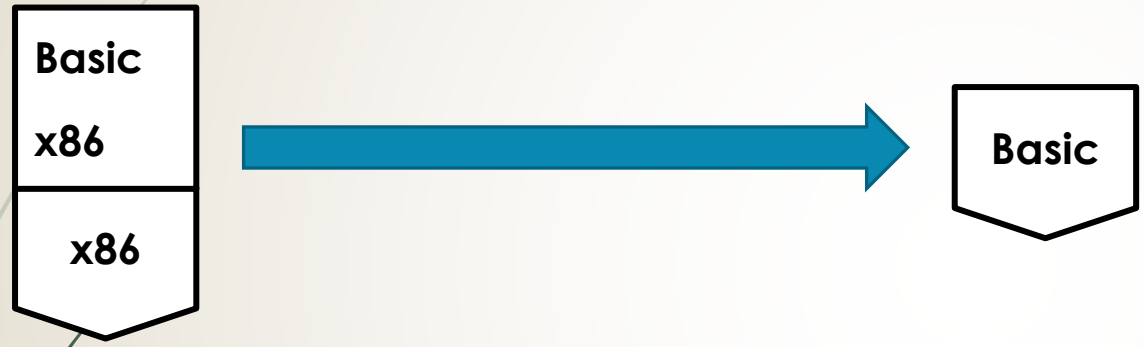


- **Exemplos:**



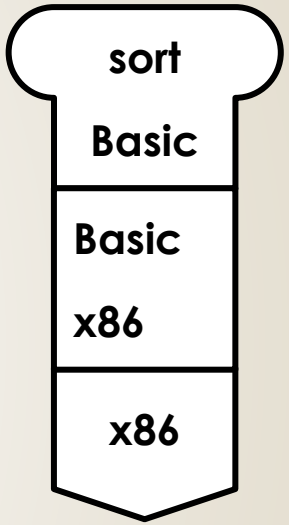


Executando um Interpretador



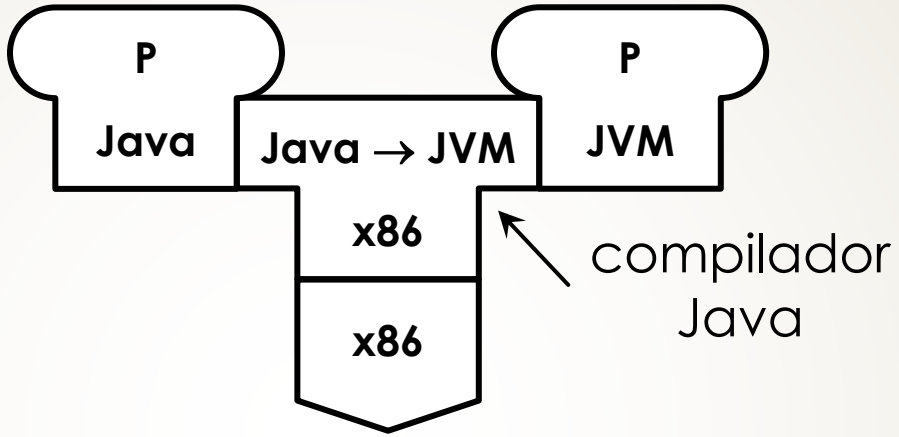
Funcionalmente equivalente a uma máquina Basic, ou seja, uma máquina que executa comandos em Basic diretamente em hardware

Exemplo:

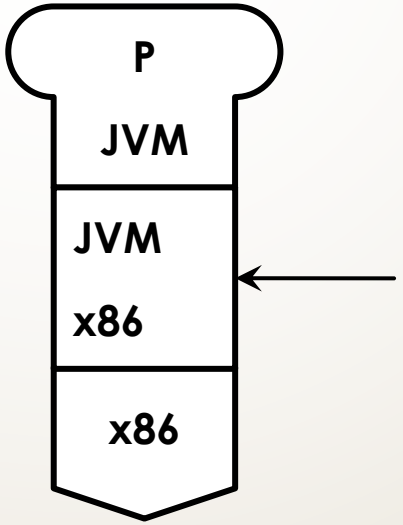


Compilando e Executando um Programa em Java

- **Compilando:**



- **Executando:**

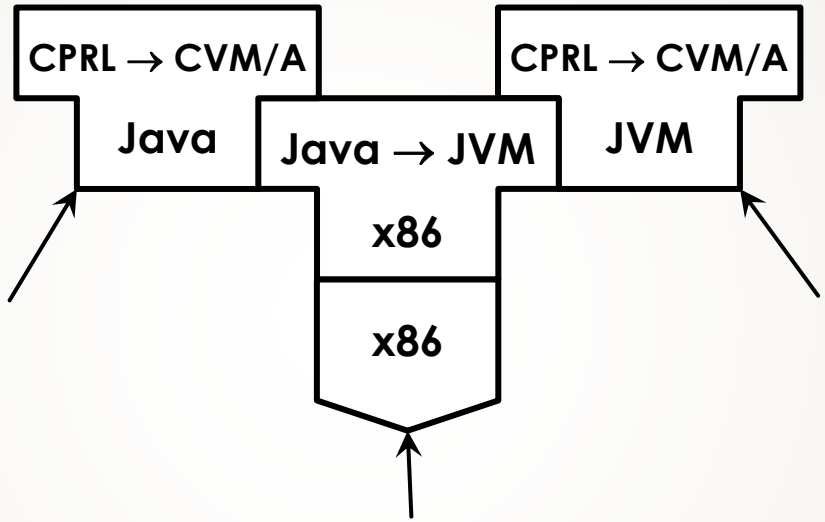


O Projeto do Compilador

- **Linguagem Fonte:** CPRL
- **Linguagem Alvo:** CVM/A, linguagem *assembly* para a CVM;
- **Linguagem de Implementação:** Java;
  - Durante o curso, vocês escreverão um compilador CPRL-para-CVM/A em Java;
  - O *assembler* da CVM será fornecido;
  - Ao compilar seus programas, vocês terão um compilador que executa na JVM.

O Projeto do Compilador

vocês escreverão o compilador em Java

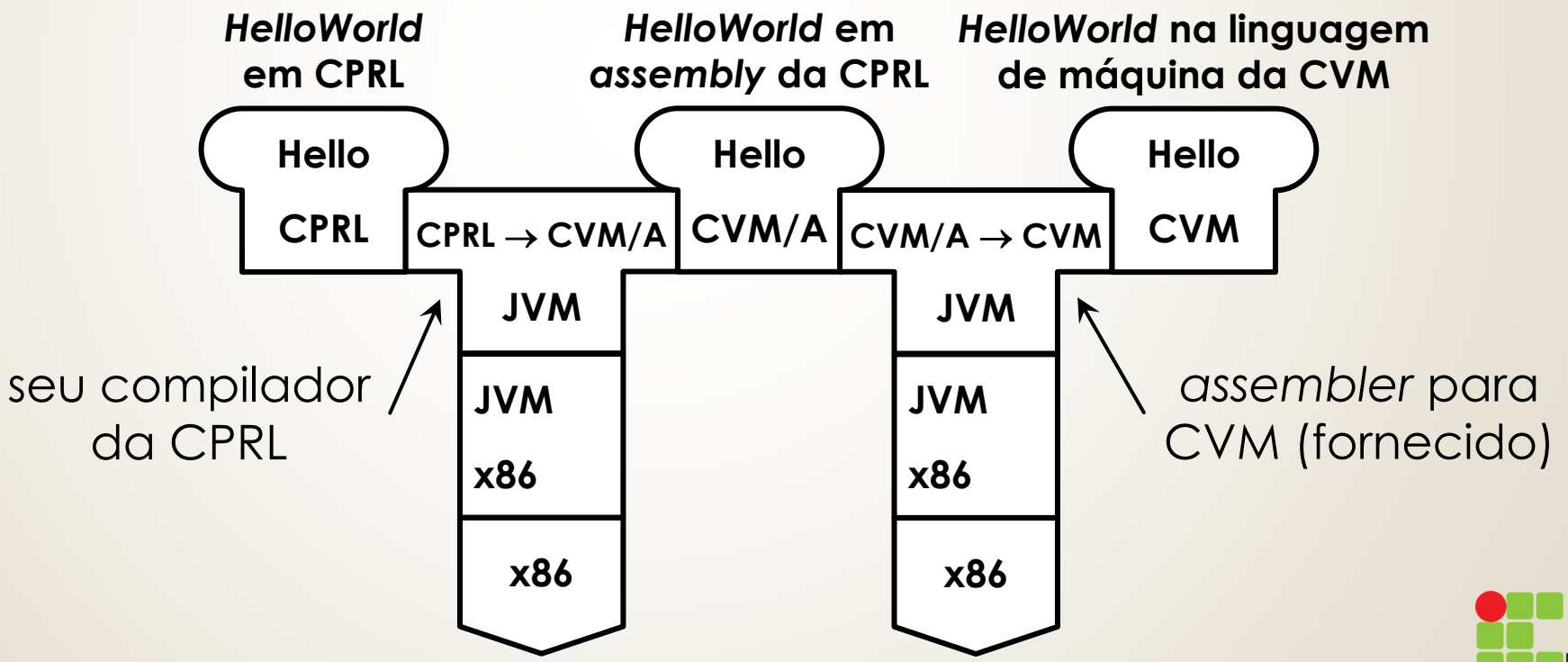


essa é versão compilada do seu compilador para a linguagem CPRL

usarão o compilador da linguagem Java para compilar o compilador da CPRL

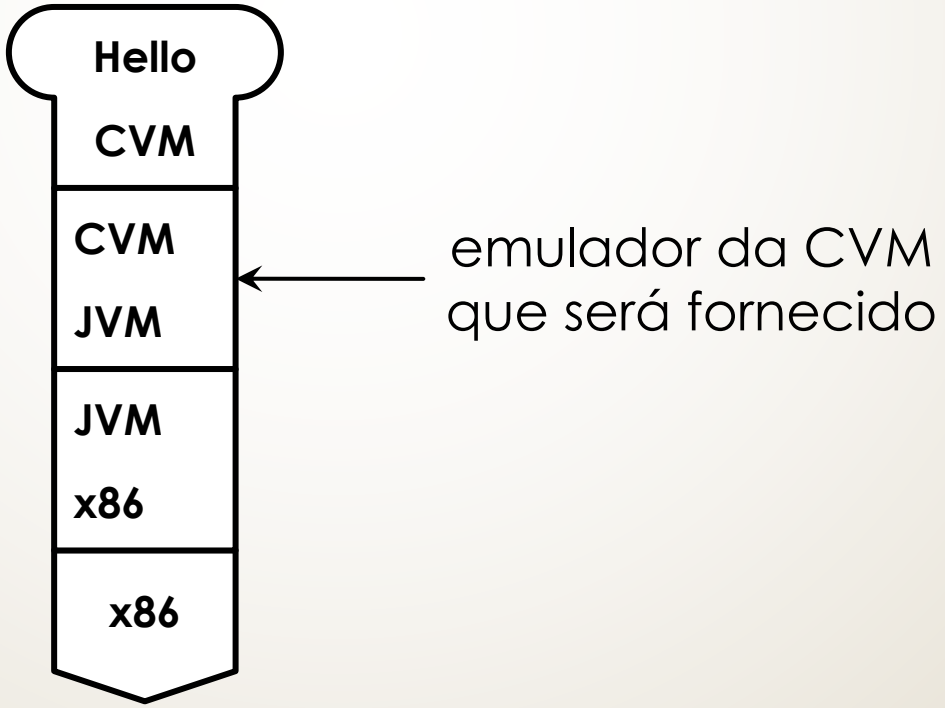
O Projeto do Compilador

- Assim que seus compiladores estiverem funcionando, vocês poderão escrever programas de teste em CPRL, compila-los e montá-los.



O Projeto do Compilador

- O interpretador (emulador) da CVM que roda na JVM será fornecido. Vocês usarão o interpretador da CVM para executar os programas compilados nos seus compiladores e montados usando o assembler da CVM.



Bibliografia

MOORE JR., J. I. **Introduction to Compiler Design: an Object Oriented Approach Using Java**. 2. ed. [s.l.]:SoftMoore Consulting, 2020. 284 p.

AHO, A. V.; LAM, M. S.; SETHI, R. ULLMAN, J. D. **Compiladores: Princípios, Técnicas e Ferramentas**. 2. ed. São Paulo: Pearson, 2008. 634 p.

COOPER, K. D.; TORCZON, L. **Construindo Compiladores**. 2. ed. Rio de Janeiro: Campus Elsevier, 2014. 656 p.

JOSÉ NETO, J. **Introdução à Compilação**. São Paulo: Elsevier, 2016. 307 p.

SANTOS, P. R.; LANGOLOIS, T. **Compiladores: da teoria à prática**. Rio de Janeiro: LTC, 2018. 341 p.