# Paradigmas de Linguagens de Programação em Python

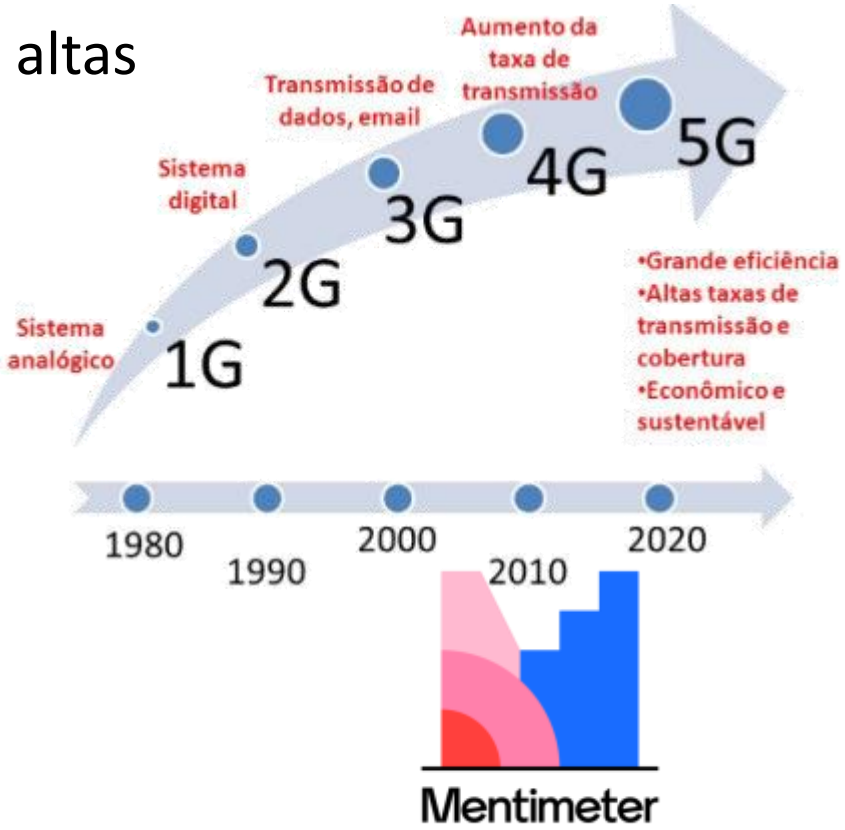
## Trade Offs e Linguagens Compiladas x Interpretadas

Prof. Dr. Daniel Caetano

2021 - 2

# Compreendendo do problema

- **Missão:** desenvolver decodificador para modem 5G
  - Fundamental para internet das coisas
  - Decodificação a taxas muito altas




  - O que é importante nesse caso?

https://www.menti.com/

# Compreendendo do problema

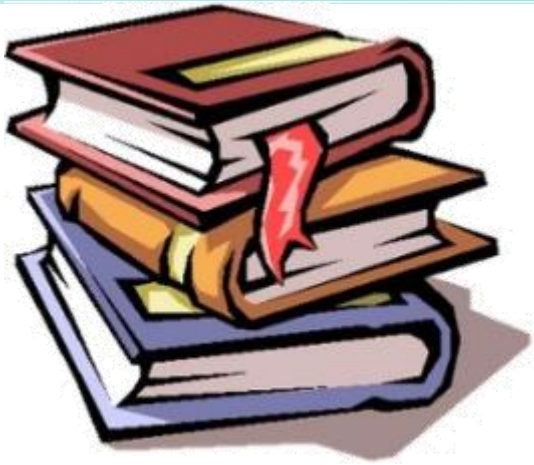
- **Missão:** desenvolver decodificador para modem 5G
  - Será que a linguagem usada influencia em alguma coisa?

# Objetivos

- Compreender os trade offs existentes com relação à linguagem e programação
- Conhecer as principais diferenças entre linguagens interpretadas e compiladas
- Primeiro contato com linguagens variadas

- **Atividade Avaliativa A!**

# Bibliografia da Aula



| Material | Acesso ao Material |
|---|---|
| Apresentação | https://www.caetano.eng.br/aulas/2021b/ara0066.php (Paradigmas de Programação – Aula 3) |
| Livro Texto | Capítulo 1, páginas 21 a 29 |
| Aprenda Mais! | • Vídeo: “Programação através de paradigmas” https://www.youtube.com/watch?v=Pg3UeB-5FdA |

# Trade Offs?

# O que é um trade off?

- Sempre que temos opções conflitantes...
  - E vamos analisar os “prós e contras”
- Exemplo: Carro...
    - Mais bonito x mais barato

# Trade off em hardware?

- Clássico custo x desempenho



**Existe algum outro critério?**

Mentimeter

https://www.menti.com/

# Trade Off na Computação

- Qual é o trade off tradicional na **programação**?

Mentimeter

https://www.menti.com/

# Trade Off Espaço-Tempo

# Trade off espaço tempo

- De que espaço estamos falando?

# Trade off espaço tempo

- E de qual tempo?

# Por que existe essa dicotomia?

- Razão 1: Laços de Repetição

```
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
print (“Olá”)
```

```
x = 1
while x <= 10:
        print (“Olá”)
        x = x + 1
```

**Em geral: decisões e saltos são mais lentos que cálculos***

# Por que existe essa dicotomia?

- Razão 2: Tabelas de Consulta

```
print (2*3)
```

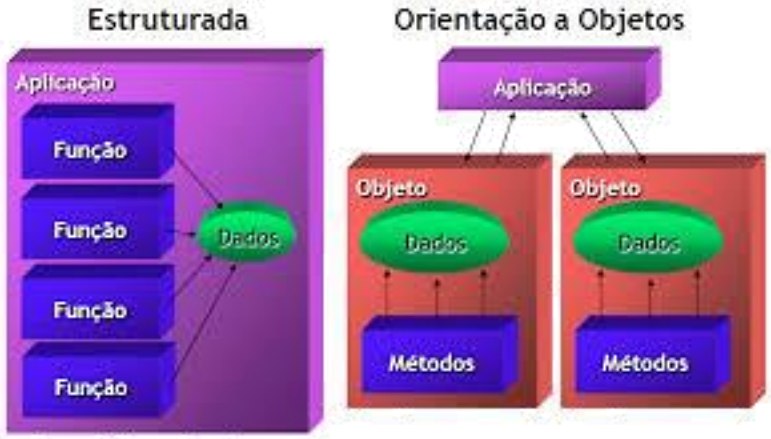
```
duasvezes = [0, 2, 4, 6, 8]
print(duasvezes[3])
```

**Em geral: consultar a memória pode ser bem mais rápido que cálcular***

- Situações comuns
  - Senos, cossenos e tangentes
  - Números pseudo-aleatórios

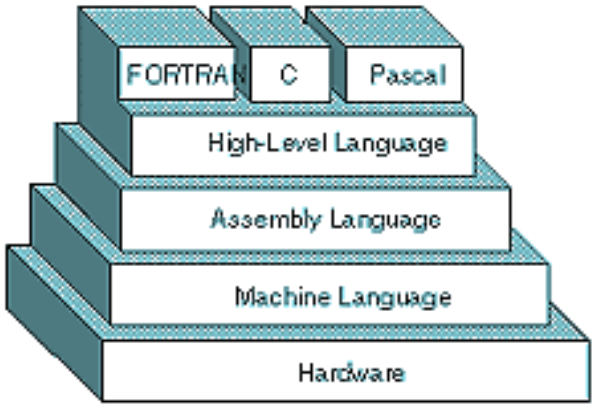
# Essas regras sempre valem?

- Infelizmente, não
  - Otimização do compilador
  - Processamento paralelo
  - Paradigma/arquitetura da linguagem
  - Implementação da linguagem (tradução)



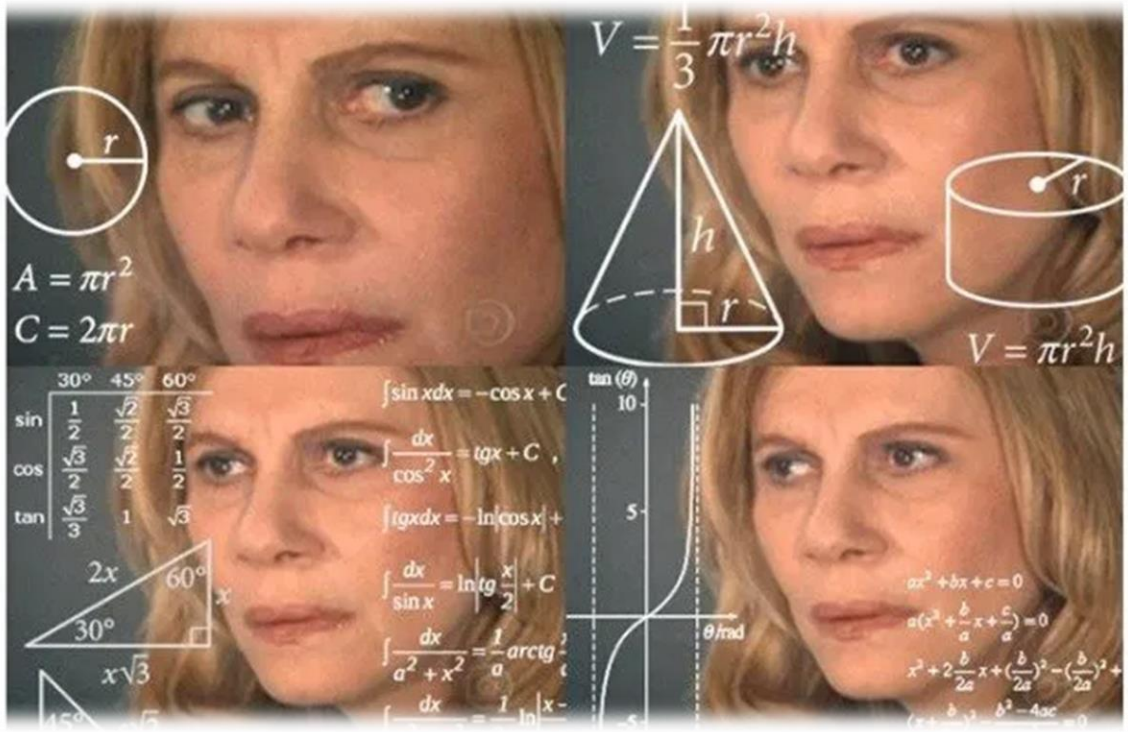
# Paradigma x Desempenho

- No geral...
  - Quanto mais próxima da lógica do computador...
    - Mais dependência da qualidade do programador
    - Execução mais rápida (se bom programador!)
    - Maior chance de erros e dificuldade de depuração.
  - Quanto mais próxima da lógica humana
    - Menor dependência relativa do programador
    - Execução mais lenta(!)
    - Menor chance de erros
    - Facilidade de depuração.



# E a implementação?

- Tradução?
  - O computador entende a nossa língua?




Mentimeter

https://www.menti.com/

# Por que não a nossa língua?

- Linguagem Natural: forma narrativa
- Simples, mas inadequada
- Por quê?
  - "O sapo ouviu um ruído da porta"
- O que isso significa?

Mentimeter

https://www.menti.com/

# Por que não a nossa língua?

- Linguagem Natural: forma narrativa
- Simples, mas inadequada
- Por quê?
  - "O sapo ouviu um ruído da porta"
- O que isso significa?

- Há **ambiguidade**!
  - É impossível dizer o que essa frase significa!

# Há “a” língua dos computadores?

- Tradução para qual língua?
  - Todos os computadores entendem a mesma língua?


Mentimeter

https://www.menti.com/

# A Linguagem dos Computadores

# A Linguagem dos Computadores

- Para começar... O alfabeto...


# A Linguagem dos Computadores

- E a linguagem construída com o alfabeto...

**Assembly i8088**

```
1001.0000b
NOP

1010.0000b
MOV AL,[addr]
```

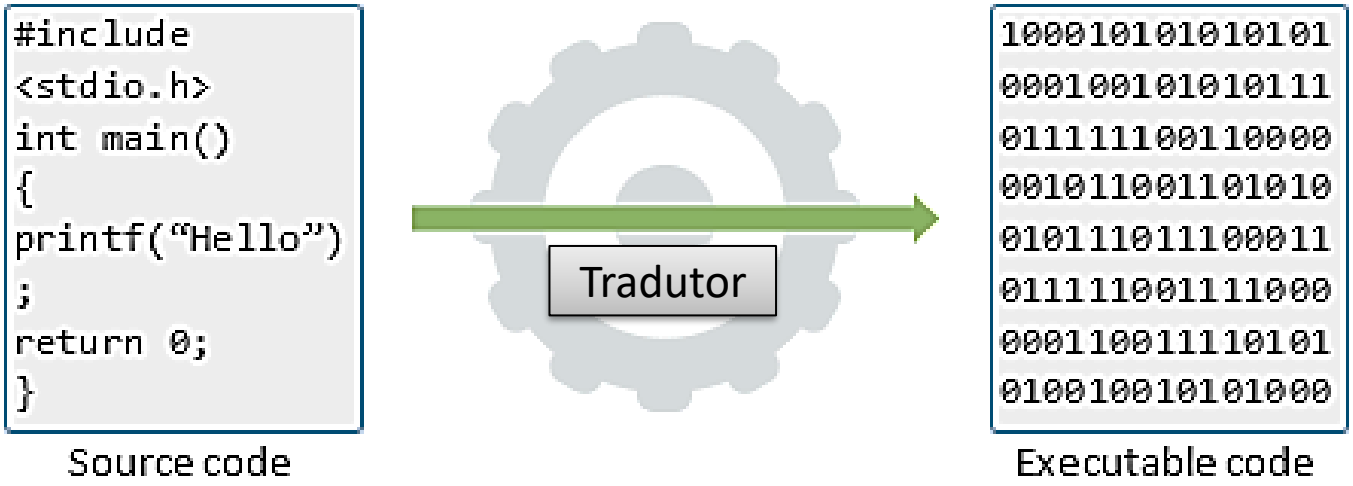
**Assembly z80**

```
1001.0000b
SUB B

1010.0000b
AND B
```

- Se programarmos em Assembly...
  - ...criar um programa para cada equipamento!

# O Papel do "Tradutor"

- Para evitar isso, existem os "tradutores"
  - Programa é escrito em **linguagem de alto nível**
    - Chamado **código fonte**
  - "Tradutor" converte em **linguagem de máquina**
    - Chamado **código binário** ou **executável**



# O Papel do “Tradutor”

Programador

```
#include <io...
int main(void)
  {
  cout << “Oi”;
```

Código Fonte

Tradutor


Celular

```
001010101010
101010101010
110111011011
111110010101
```

Código Binário de PC


Computador PC

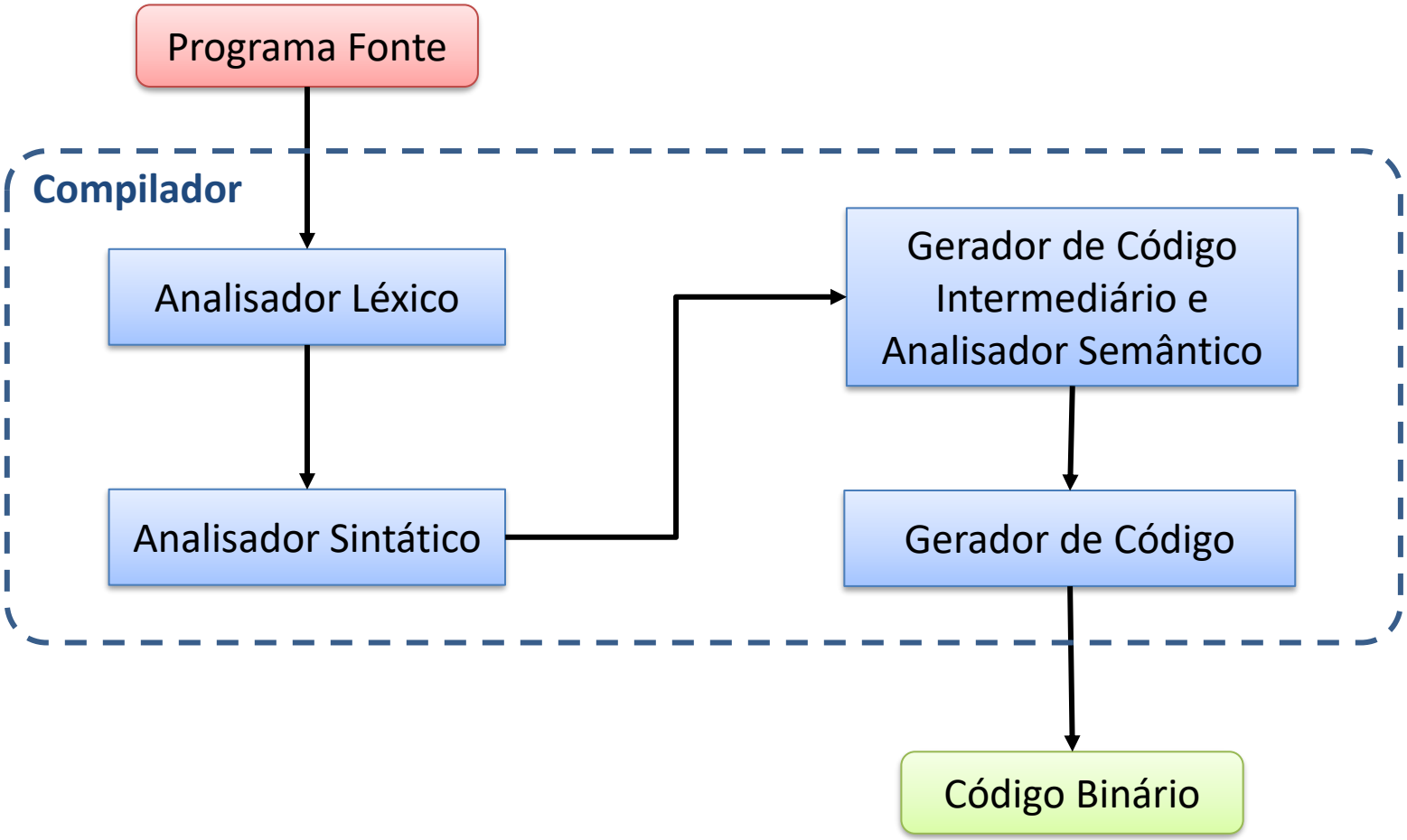
# Tipos de “Tradutores”

- Compiladores
  - Processa arquivo “fonte”...
  - ...gerando o “binário” para um equipamento
  - O “binário” é distribuído para os usuários.
- Interpretadores
  - Executa “diretamente” o arquivo “fonte”
  - O “fonte” é distribuído para os usuários.

# Compiladores e Compilação

# Compiladores

- Traduzem o “código fonte” para “binário”



# Compiladores

- Nas várias etapas são verificados:
  - As instruções
  - Os tipos de dados
  - A coerência do código

**Positivos**

- Verifica inúmeros aspectos enquanto está com o desenvolvedor
- Todo o processo de tradução e verificação é feito apenas uma vez na geração do programa
- Usuário tem acesso apenas a código binário
- Controle é do programador.

**Negativos**

- O binário é específico para um equipamento
- Pode tornar o desenvolvimento mais lento
- Em geral possuem recursos menos poderosos de verificação e recuperação de erros de tempo de execução.

# Compiladores

- Código gerado é específico de um processador
  - Mas e a placa de vídeo?
  - E o áudio?
  - Em geral: fornecidos por bibliotecas e/ou S.O.

- Processo de ligação
  - Associa o código **objeto** binário...
  - Às **bibliotecas** adicionais do sistema ou ambiente
  - Gerando o binário **executável** (CPU+S.O.).

# INTERPRETADORES

# Interpretadores

- Programas que “interpretam” a linguagem fonte
  - Na hora de executar!
  - Verifica tudo que é possível durante a execução

**Positivos**

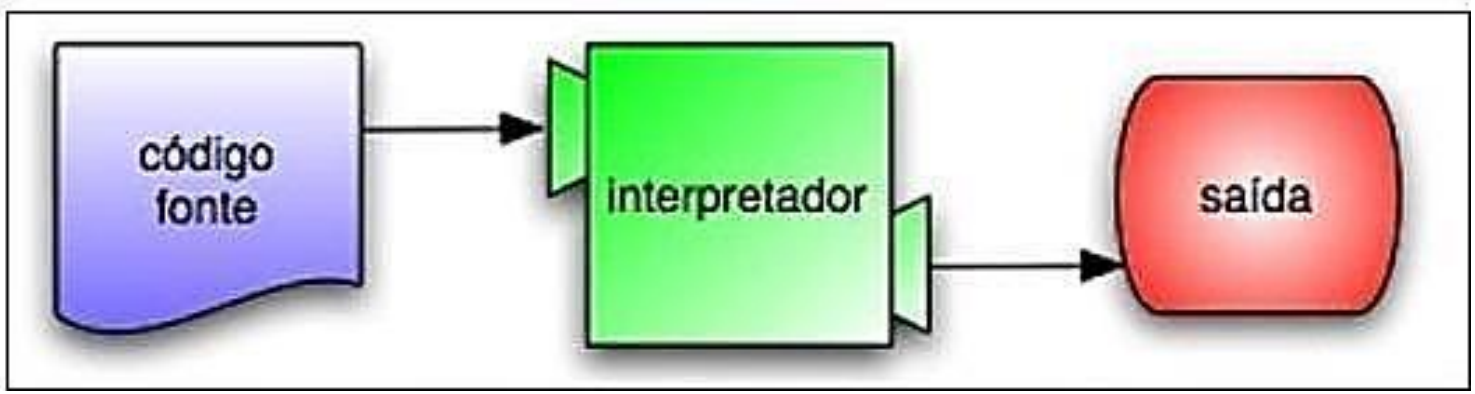
- O desenvolvimento pode ser extremamente ágil
- Em geral possuem recursos poderosos, como coleta de lixo
- O mesmo código executa em “todo lugar”.

**Negativos**

- Verificação só ocorre na hora de executar... Problema pode ocorrer na mão do usuário
- O processo de tradução ocorre toda vez que o programa for executado
- Usuário tem acesso ao código fonte.
- Controle é do interpretador.

# Interpretadores

- O interpretador é específico para CPU+S.O.
- Mesmo código fonte executará em qualquer ambiente (basta existir o interpretador)

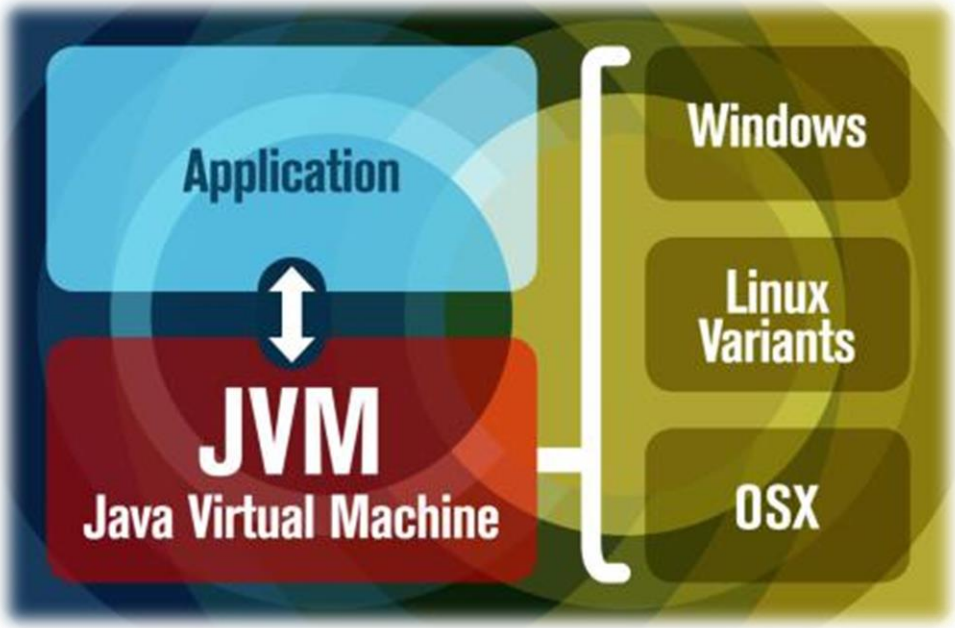



**É um programa!**

# Implementação Híbrida

# Implementação Híbrida

- Um processo de *compilação* que gera um código binário de uma *máquina imaginária*
- Um *interpretador* que implementa essa máquina imaginária: **máquina virtual**



# Implementação Híbrida

Programador

```
import java...
int main(Stri...
  {
  System.out...
```

Código Fonte

Compilador


Computador PC **com Máquina Virtual**


Celular **com Máquina Virtual**

```
001010101010
101010101010
110111011011
111110010101
```

"Binário Genérico"

# Implementação Híbrida

- Combina compilado com interpretado

**Positivos**

- Verifica inúmeros aspectos enquanto está com o desenvolvedor
- A parte mais onerosa do processo de tradução e verificação é feito apenas uma vez na geração do programa
- Usuário tem acesso apenas a um código binário
- Há verificações adicionais em tempo de execução
- Em geral possuem recursos poderosos como coleta de lixo
- Binário executa em “todo lugar”.

**Negativos**

- Pode tornar o desenvolvimento mais lento que os sistemas interpretados
- Podem ser mais pesados para o usuário que programas compilados
- As máquinas virtuais podem ser mais complexas de portar que os interpretadores ou compiladores
- Controle permanece na mão do interpretador (máquina virtual).

# Implementação Híbrida Moderna

- Um processo de *compilação* que gera um código binário de uma *máquina imaginária*
- Um *interpretador* que implementa essa máquina imaginária: **máquina virtual**
- O interpretador traduz binário genérico em binário do computador *host*, melhorando o desempenho após a 1ª execução

**JIT: Just In Time Compiler**

# Compilado x Interpretado x Híbrido

- Dispondo dos três, qual a melhor alternativa?

Mentimeter

https://www.menti.com/

# PRÉ-PROCESSADORES

# Pré-Processadores

- Alguns ambientes e ferramentas implementam facilitadores poderosos
  - Exemplo: macros

```
#define info “Desenvolvido por Daniel Caetano”
```

- Muitas vezes, esses recursos não fazem parte exatamente da linguagem
  - Precisam ser processados antes...
  - Do compilador atuar

# Atuação do Pré-Processador

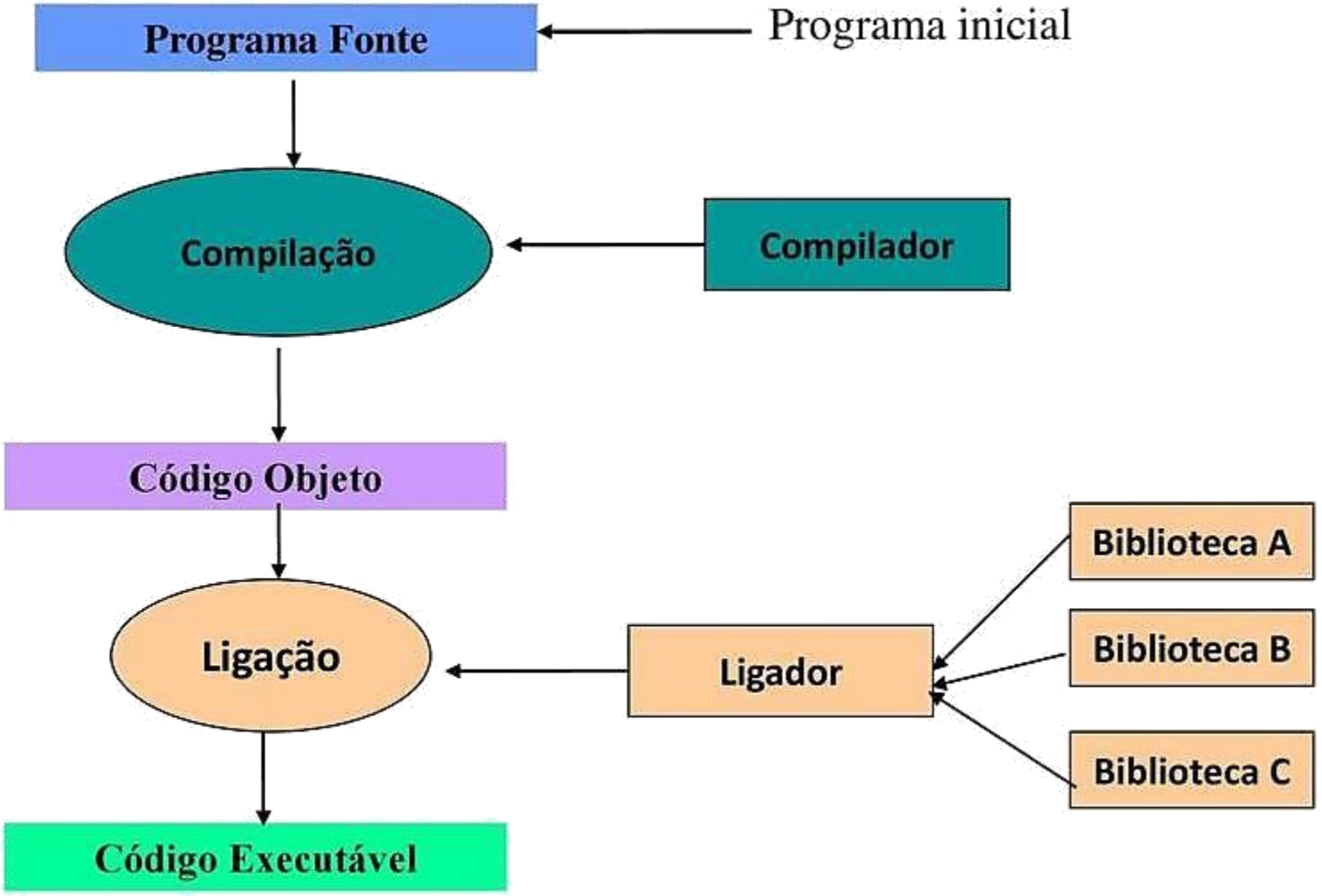
- Código escrito

```
#include <iostream>
#define info "Desenvolvido por Daniel Caetano"
using namespace std;

int main() {
    cout << info << endl;
    return 0;
}
```

- Código a ser compilado

```
#include <iostream>
using namespace std;

int main() {
    cout << "Desenvolvido por Daniel Caetano" << endl;
    return 0;
}
```

# Processo Completo



# Ambientes de Desenvolvimento

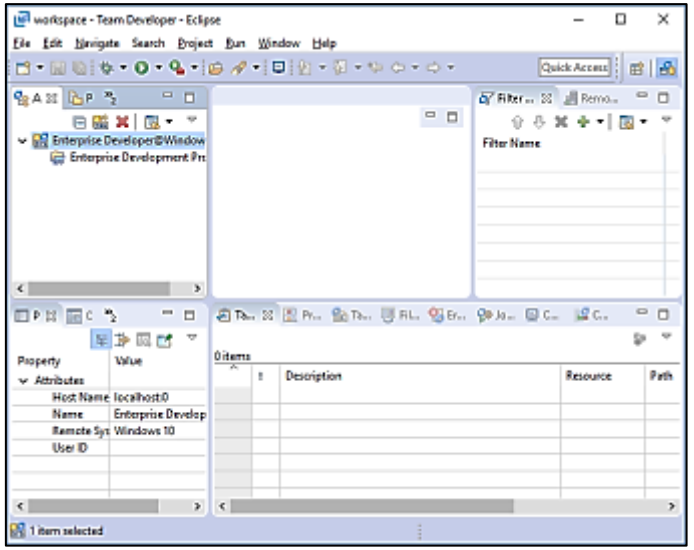
# Ambientes de Desenvolvimento

- Podem ser “montados”
  - Famoso “na raça”
  - Editor
  - Pré-processador
  - Compilador
  - Linkeditor
  - Depurador/Debugger

- Existem os integrados, ou IDEs
  - *Integrated Development Environment*

# Ambientes de Desenvolvimento

- IDEs integram o quê, exatamente?
  - Navegador de arquivos
  - Editor de arquivos
  - Painel de Mensagens/Erros
  - Funções de depuração/debug
  - Execução de código



# Ambientes de Desenvolvimento

- Existem muitas e muito variadas IDEs
  - Simples
    - IDLE, NotePad++
  - Complexas
    - MS Visual Studio, Visual Studio Code, Eclipse, NetBeans, Code::Blocks, Jupyter, Spyder...

# Conhecendo Algumas Linguagens e Ambientes

# Instalação de Ambientes

- Vamos conhecer os seguintes ambientes
  - Python + IDLE
  - Java Development Kit + NetBeans
  - GCC + Code::Blocks

# Ambientes Online

- Existem ambientes online para programar

  https://www.tutorialspoint.com/codingground.htm

- Acompanhe o professor na criação do “Hello World” em cada um desses ambientes
  - C++
  - C#
  - Java
  - Python

# Contagem até 100.000.000

- Em Python

```
import time
print("Começou")
inicio = time.time()
x = 0
while x < 100000000:
    x = x + 1
fim = time.time()
total = fim - inicio
print("Tempo total: " + str(total))
```

# Contagem até 100.000.000

- Em Java

```
package time;
public class Time {
    public static void main(String[] args) {
        System.out.println("Começou");
        long inicio = System.nanoTime();
        int x = 0;
        while (x < 100000000) {
            x = x + 1;
        }
        long fim = System.nanoTime();
        long total = (fim – inicio)/1000000000;
        System.out.println("Tempo total: " +
             ((double)(fim-inicio)/1000000000.0));
    }
}
```

# Contagem até 100.000.000

- Em C++

```
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;

int main() {
    cout << "Começou";
    auto inicio = high_resolution_clock::now();
    int x = 0;
    while (x < 100000000) {
        x = x + 1;
    }
    auto fim = high_resolution_clock::now();
    double total = duration_cast<microseconds>(fim-inicio).count();
    cout << "Tempo total: " << std::fixed << (total/1000000);
    return 0;
}
```

# ATIVIDADE

# Atividade

- Grupos
  - Entrar na sala do grupo para discussão: 20 minutos
- Cada grupo deve instalar a linguagem e contar o tempo necessário para calcular 100.000.000 de vezes a potência de $2^{10}$.
  - Pyhton: `y = 2**10`
  - Java: `double y = Math.pow(2,10);`
  - C++:
    ```
    #include <math.h>
    double y = pow(2,10);
    ```
  - **Grupo 1 e 4**: Python
  - **Grupo 2 e 5**: Java
  - **Grupo 3 e 6**: C++

# Atividade - Discussão

- Tempos de cada grupo
  - **Grupo 1 e 4**: Python
  - **Grupo 2 e 5**: Java
  - **Grupo 3 e 6**: C++

# ENCERRAMENTO

# Resumo e Próximos Passos

- Principais *trade offs*
- As diferentes implementações de linguagem
  - Vantagens e desvantagens
- Implementações e variações na prática
- **Pós Aula**: Aprenda Mais, Pós Aula e Desafio!
  - No padlet: https://padlet.com/djcaetano/paradigmas

---

- Variáveis em Python
  - Nomes e vinculações

# PERGUNTAS?