

MBAUSP
ESALQ

PARADIGMAS DE DESENVOLVIMENTO DE SOFTWARE I E II

Professor Helder Prado Santos

MBAUSP ESALQ

A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

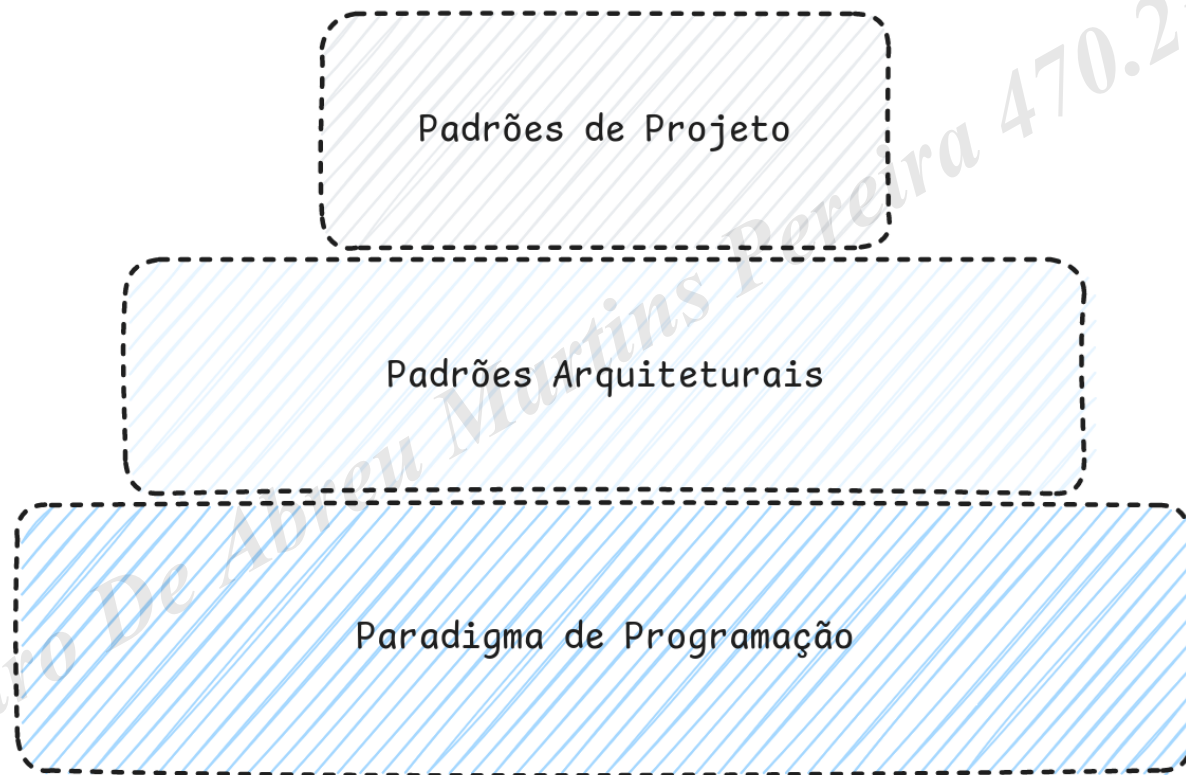
Proibida a reprodução, total ou parcial, sem autorização.

Lei nº 9610/98

Paradigma na Programação



O Paradigma é a Base da Solução



Paradigma Imperativo (Década de 50)

- ❑ **Contexto:** O paradigma imperativo foi um dos primeiros estilos de programação adotados, em uma época em que o foco era controlar diretamente os recursos do computador.
- ❑ **Linguagens iniciais:** Assembly, Fortran.
- ❑ **Características:** Comandos sequenciais, uso de variáveis para armazenar estado e controle explícito do fluxo de execução.
- ❑ **Impacto:** Esse paradigma serviu como a base para a programação estruturada e oferece controle direto sobre o hardware.

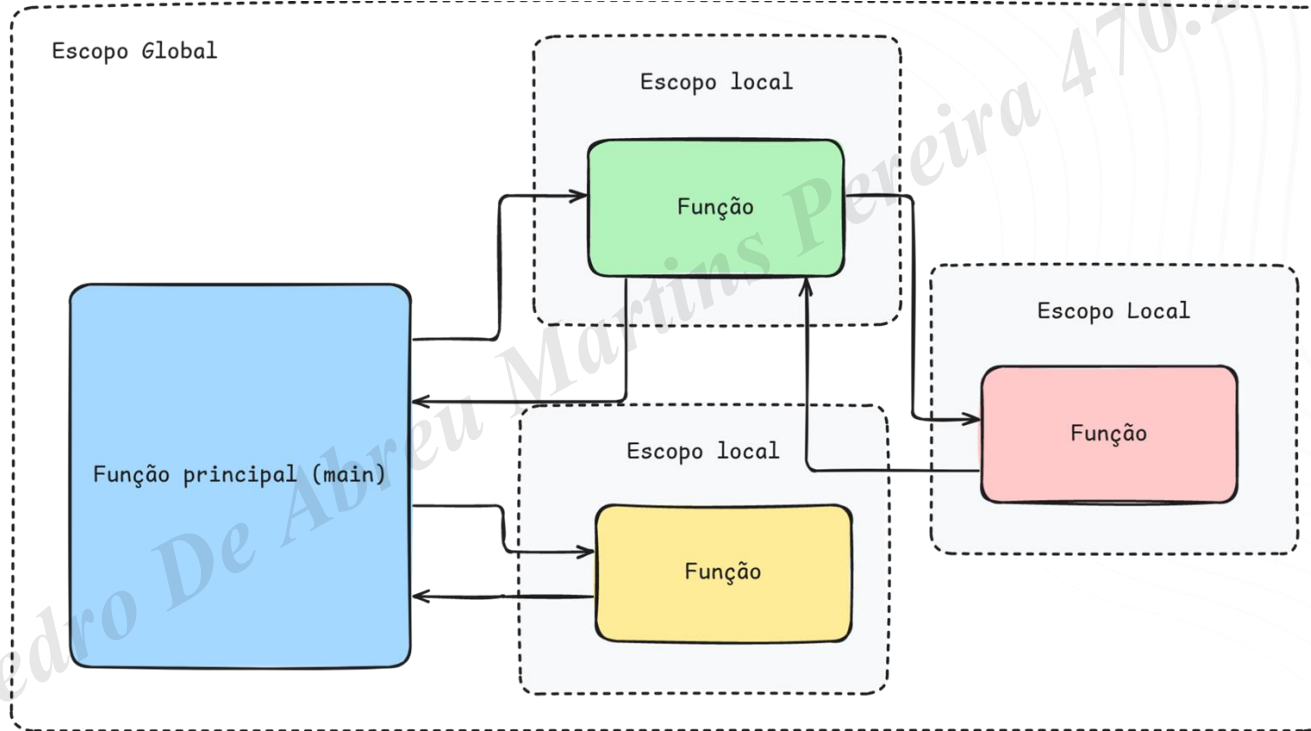
Paradigma Funcional (Década de 1950)

- ❑ **Contexto:** Baseado na teoria matemática das funções, esse paradigma cresceu com a popularização de linguagens específicas.
- ❑ **Linguagens:** Lisp, Haskell.
- ❑ **Características:** O paradigma funcional evita estados mutáveis e efeitos colaterais, priorizando funções puras e imutabilidade.
- ❑ **Impacto:** Oferece vantagens em programação concorrente e processamento de dados, sendo muito utilizado em problemas de matemática e computação científica, assim como em frameworks modernos para manipulação de dados e concorrência.

Paradigma Procedural (Década de 1960)

- ❑ **Contexto:** O paradigma procedural surgiu como uma evolução natural do imperativo, ao introduzir estruturas que organizam o código em procedimentos ou sub-rotinas. Isso ajudou a tornar o código mais modular e reutilizável.
- ❑ **Linguagens:** C, Fortran, COBOL, Pascal.
- ❑ **Características:** Organização do código em procedimentos ou funções que podem ser chamados para realizar tarefas específicas. O programa é construído como uma sequência de chamadas a esses procedimentos, o que facilita o fluxo controlado de execução.
- ❑ **Vantagens:** Melhor modularidade em comparação ao imperativo puro, permitindo a reutilização de código e a redução de redundância.
- ❑ **Exemplos de uso:** Aplicações científicas, engenharia e sistemas empresariais que requerem processos repetitivos e organizados.

Paradigma Procedural (Década de 1960)



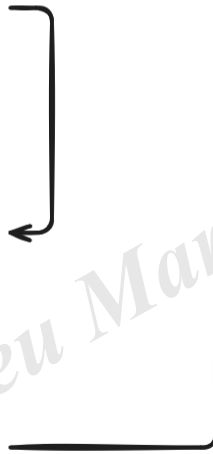
Paradigma Estruturado (Década de 60)

- ❑ **Contexto:** Conforme os programas se tornaram mais complexos, surgiu a necessidade de criar uma estrutura mais organizada para o código.
- ❑ **Linguagens:** C, Pascal.
- ❑ **Características:** Introdução de estruturas de controle (como loops e condicionais), subrotinas e funções, facilitando a manutenção e legibilidade do código.
- ❑ **Impacto:** O paradigma estruturado abriu caminho para o desenvolvimento de programas mais organizados, reduzindo a "codificação spaghetti" (código desorganizado e de difícil leitura).

Paradigma Estruturado (Década de 60)


Code

```
goto label_1;  
statement_1;  
statement_2;  
  
label_1:  
statement_3;  
statement_4;  
  
goto label_3;  
statement_5  
  
label_2;
```



Code

```
goto label_4;  
statement_6;  
statement_7;  
  
label_3:  
statement_8;  
statement_9;  
  
label_4;  
statement_10;  
statement_11;
```



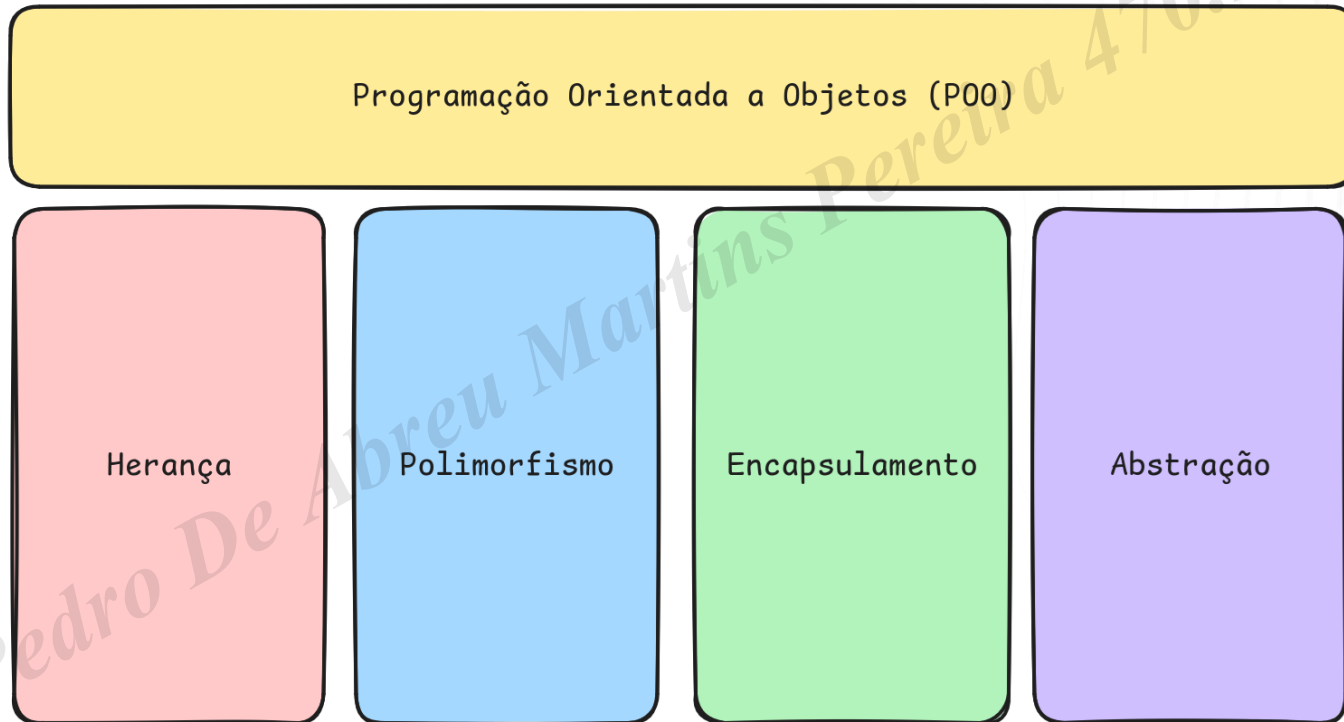
Paradigma Lógico (Década de 1970 e 1980)

- ❑ **Contexto:** Criado para aplicações de inteligência artificial e resolução de problemas baseados em lógica.
- ❑ **Linguagens:** Prolog, Datalog.
- ❑ **Características:** Foca em declarar relações e regras que descrevem o problema, deixando para o computador a busca de soluções lógicas.
- ❑ **Impacto:** Embora não seja amplamente utilizado para aplicações comerciais, é essencial em áreas de IA, como linguagens naturais e inferência lógica.

Paradigma Orientado a Objetos (OO) (Década de 80)

- ❑ **Contexto:** Surgiu como resposta à necessidade de modularidade e reutilização de código para sistemas cada vez mais complexos.
- ❑ **Linguagens:** Java, C++.
- ❑ **Características:** Organiza o código em "objetos" que representam entidades do mundo real ou conceitual, encapsulando dados e comportamentos.
- ❑ **Impacto:** Tornou-se o paradigma dominante em desenvolvimento corporativo e jogos, promovendo a criação de bibliotecas reutilizáveis e ajudando a gerenciar a complexidade de sistemas grandes.

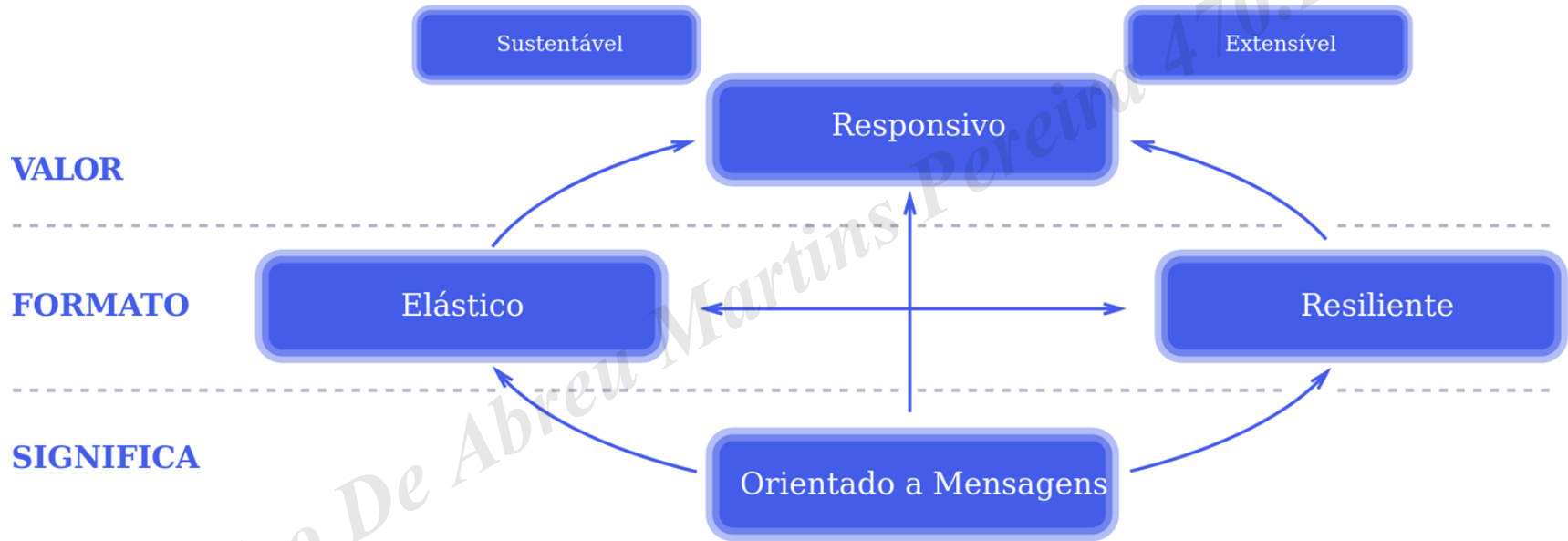
Paradigma Orientado a Objetos (OO) (Década de 80)



Paradigma Reativo (2000 em diante)

- ❑ **Contexto:** Com o aumento da demanda por sistemas em tempo real e responsivos, o paradigma reativo se tornou mais relevante.
- ❑ **Ferramentas e Tecnologias:** RxJS, Reactor, frameworks de programação reativa.
- ❑ **Características:** Utiliza fluxos de dados assíncronos e programação orientada a eventos, onde mudanças em uma parte do sistema reagem automaticamente a mudanças em outra.
- ❑ **Impacto:** Essencial para aplicações modernas como dashboards em tempo real, plataformas de streaming e sistemas distribuídos.

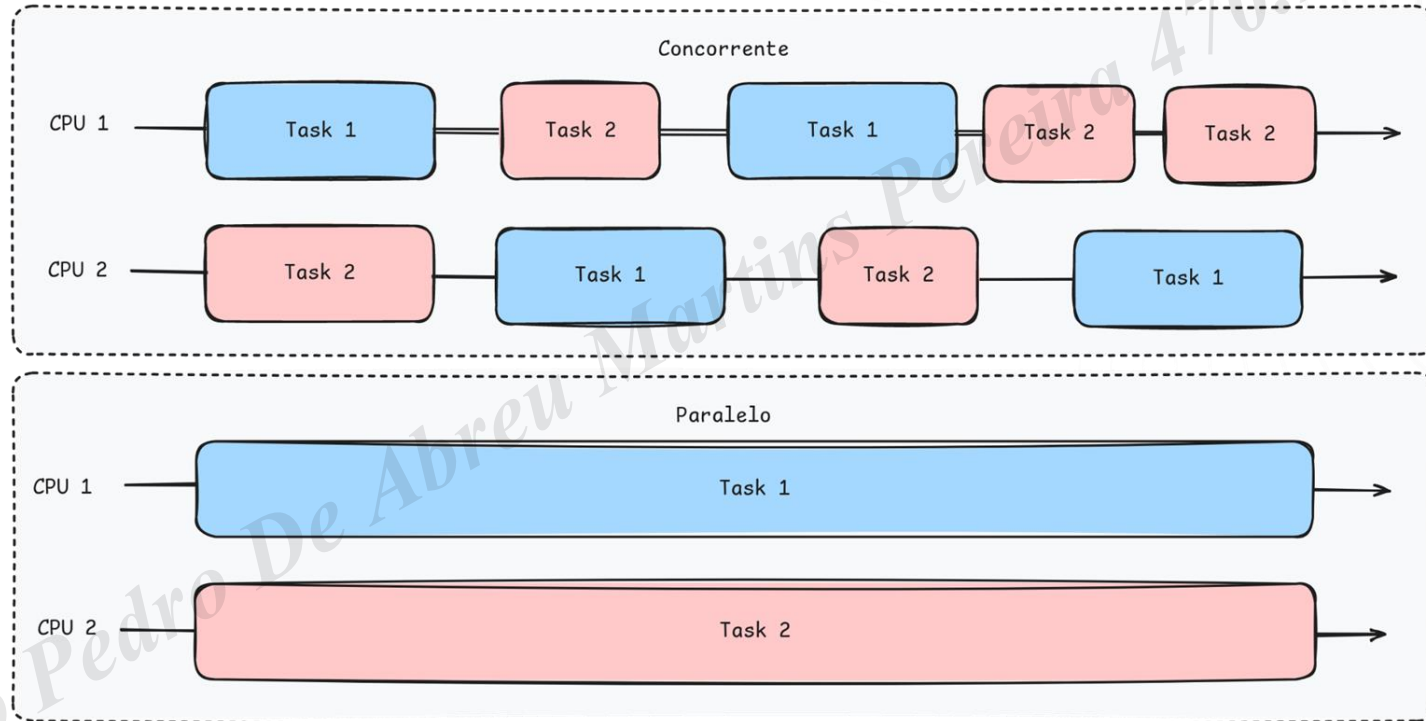
Paradigma Reativo (2000 em diante)



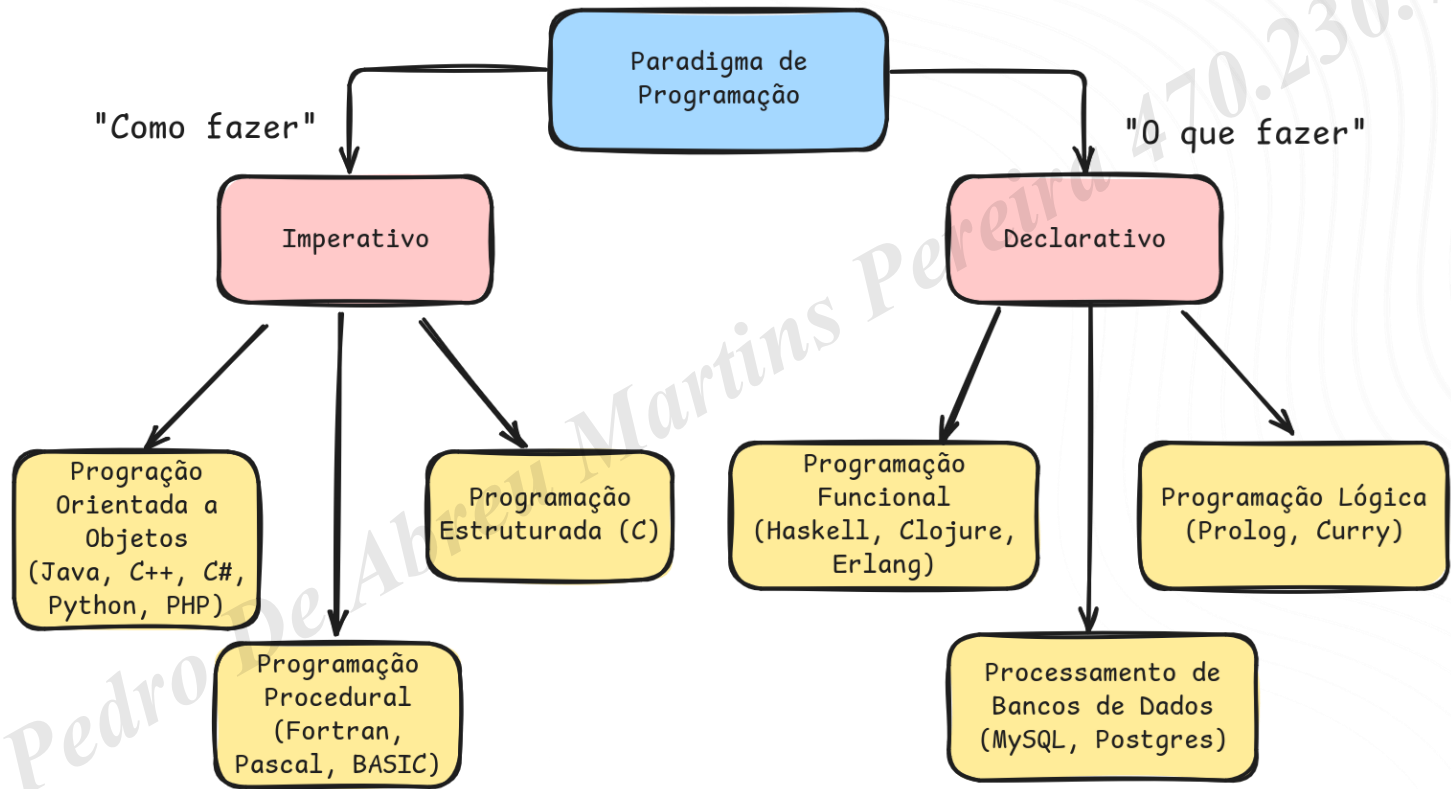
Paradigmas Concorrentes e Paralelos (ao longo das décadas)

- ❑ **Contexto:** Com a chegada dos processadores multicore, a programação concorrente e paralela tornou-se necessária para aproveitar melhor o hardware.
- ❑ **Ferramentas e Tecnologias:** Threads, OpenMP, MPI, frameworks de atores como Akka.
- ❑ **Características:** Foco na execução simultânea de múltiplas tarefas para aumentar o desempenho e a eficiência de aplicativos que processam grandes volumes de dados.
- ❑ **Impacto:** Essencial para áreas como computação científica, simulações, e big data, ajudando a criar programas mais rápidos e capazes de processar grandes volumes de dados em paralelo.

Paradigmas Concorrentes e Paralelos (ao longo das décadas)



Paradigma Imperativo e Declarativo



Paradigma Imperativo e Declarativo

Paradigma Imperativo	Paradigma Declarativo
Como fazer	O que fazer
Mais verboso	Menos verboso
Variáveis mutáveis	Imutabilidade
Comandos sequenciais	Conjunto de statements
Mais explícito	Menos explícito
Maior foco no fluxo de dados	Maior foco na lógica
Mais legível e alterável	Mais difícil de entender
Mais difícil de escalar	Mais fácil de escalar

Preparando uma xícara de café usando os paradigmas

Imperativo:

- "João, pegue a cafeteira. Coloque água até o nível indicado. Abra o compartimento do pó de café. Coloque três colheres de pó. Feche o compartimento. Coloque a cafeteira na base. Aperte o botão de ligar. Aguarde até o café ficar pronto. Pegue uma xícara e coloque o café nela."

Declarativo:

- "João, faça uma xícara de café."

Exemplo prático dos paradigmas


```
# Lista de números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Variável para armazenar a soma dos números pares
soma_pares = 0

# Loop para iterar pela lista e somar os pares
for numero in numeros:
    if numero % 2 == 0: # Verifica se o número é par
        soma_pares += numero # Adiciona o número par à soma

print("Soma dos números pares:", soma_pares)
```

Exemplo prático dos paradigmas



```
# Lista de números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Soma dos números pares usando funções de alta ordem
soma_pares = sum(filter(lambda x: x % 2 == 0, numeros))

print("Soma dos números pares:", soma_pares)
```

Exemplo prático dos paradigmas



```
SELECT nome, idade  
FROM clientes  
WHERE cidade = 'São Paulo' AND idade > 30;
```

Exemplo prático dos paradigmas

```
class Livro:
    def __init__(self, titulo, autor, ano_publicacao):
        self.titulo = titulo
        self.autor = autor
        self.ano_publicacao = ano_publicacao
        self.disponivel = True

    def emprestar(self):
        if self.disponivel:
            self.disponivel = False
            print(f'O livro "{self.titulo}" foi emprestado.')

            self.disponivel = True
            print(f'O livro "{self.titulo}" foi devolvido.')
```


Exemplo prático dos paradigmas

```
class Biblioteca:
    def __init__(self):
        self.livros = []

    def adicionar_livro(self, livro):
        self.livros.append(livro)
        print(f'0 livro "{livro.titulo}" foi adicionado à biblioteca.')

    def listar_livros(self):
        print("\nLista de livros na biblioteca:")
        for livro in self.livros:
            print(livro)

    def emprestar_livro(self, titulo):
        for livro in self.livros:
            if livro.titulo == titulo:
                livro.emprestar()
                return
        print(f'0 livro "{titulo}" não foi encontrado na biblioteca.')

    def devolver_livro(self, titulo):
        for livro in self.livros:
            if livro.titulo == titulo:
                livro.devolver()
                return
```

Exemplo prático dos paradigmas

```
# Criando alguns livros
livro1 = Livro("1984", "George Orwell", 1949)
livro2 = Livro("O Senhor dos Anéis", "J.R.R. Tolkien", 1954)
livro3 = Livro("O Código Da Vinci", "Dan Brown", 2003)

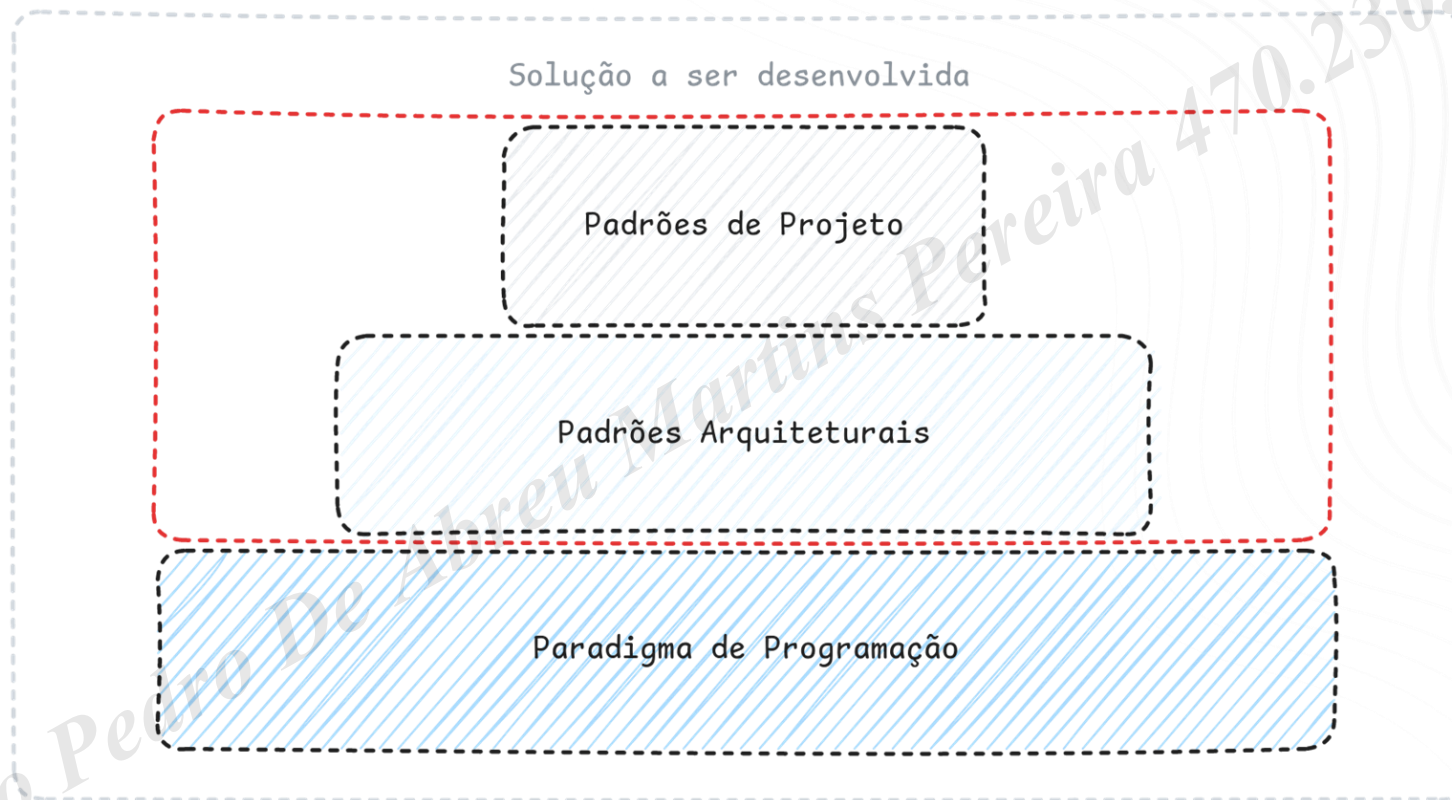
# Adicionando livros à biblioteca
biblioteca.adicionar_livro(livro1)
biblioteca.adicionar_livro(livro2)
biblioteca.adicionar_livro(livro3)

# Listando livros na biblioteca
biblioteca.listar_livros() # [livro1, livro2, livro3]

# Emprestando um livro
biblioteca.emprestar_livro("1984")
biblioteca.listar_livros() # [livro2, livro3]

# Devolvendo um livro
biblioteca.devolver_livro("1984")
biblioteca.listar_livros() # [livro1, livro2, livro3]
```

Escolhendo o Paradigma Adequado



MBAUSP ESALQ

Obrigado!

[Prof. Helder Prado Santos | LinkedIn: linkedin.com/in/helderprado](#)