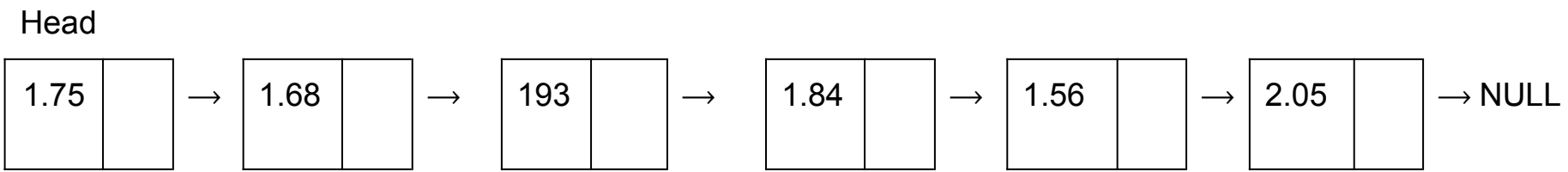
Estrutura de Dados

Estrutura de Dados Estática vs Estrutura de Dados Dinâmica:

- A estrutura de dados estática apresenta um tamanho fixo, onde os dados armazenados podem ser modificados sem mudança no espaço de memória alocado.

Índices:	0	1	2	3	4	5
Valores:	1.75	1.68	1.93	1.84	1.56	2.05

- A estrutura de dados dinâmica **não** apresenta um tamanho fixo, logo, apresentam um tamanho que pode variar através da alocação de memória em tempo real. A memória pode ser dinamicamente alocada ou desalocada durante a execução do programa.

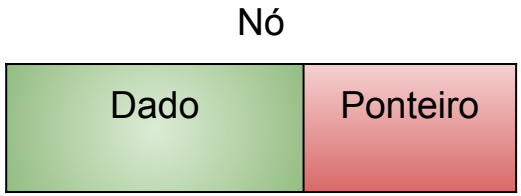


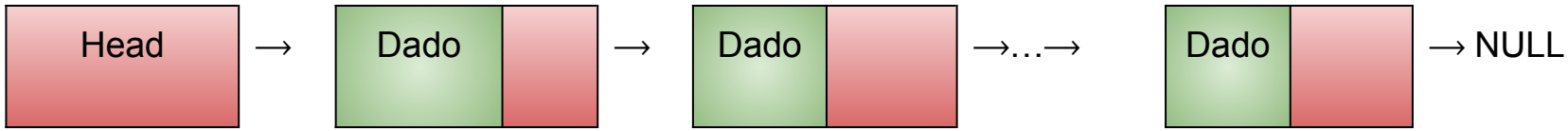
Estrutura de Dados Estática vs Estrutura de Dados Dinâmica:

Características	Estática	Dinâmica
Alocação de memória	Tempo de compilação	Tempo de execução
Tamanho	Não pode ser alterado	Pode ser alterado
Utilização da memória	Ineficiente	Eficiente (reuso de memória)
Acesso	Rápido	Mais lento
Tipos de estruturas	Arrays, Stacks, Queues, Trees com tamanho fixo	Lists, Trees, Hash tables

Lista Ligada (Encadeada)

- Estrutura dinâmica, ou seja, sem tamanho fixo.
- Melhor complexidade de tempo em operações de inserção e remoção de dados.
- Estrutura:





Lista Ligada (Encadeada)

- Operações importantes na manipulação de listas:
  - Verificar se a lista está vazia
  - Percorrer a lista
  - Adicionar um nó
  - Retornar um nó
  - Verificar se o nó existe
  - Remover o nó

Lista Ligada (Encadeada)

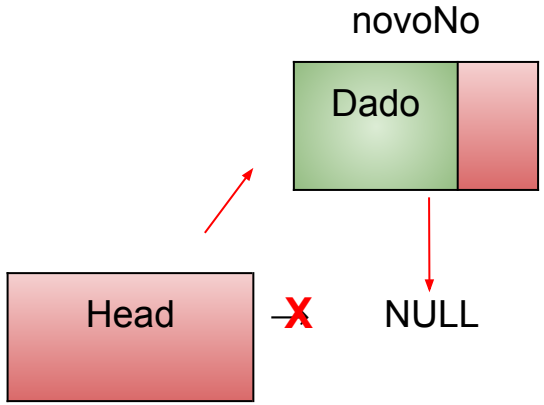
```python
'''Classe no em uma lista'''
class no:
    def __init__(self,valor):
        self.valor = valor
        self.prox = None

'''Classe lista ligada'''
class listaLgd:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return not self.head
```

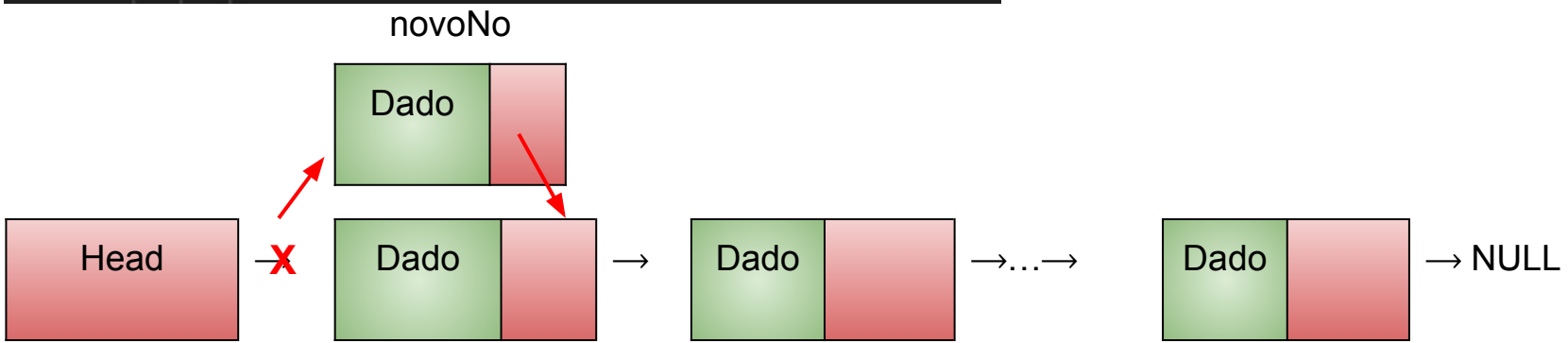
Lista Ligada (Encadeada)

```
19  #Exibindo lista
20      def __str__(self):
21          lst = ""
22          posAtual = self.head
23          while(posAtual!=None):
24              lst += f'{posAtual.valor}->'
25              posAtual = posAtual.prox
26          lst +="None"
27          return lst
```

Lista Ligada (Encadeada)

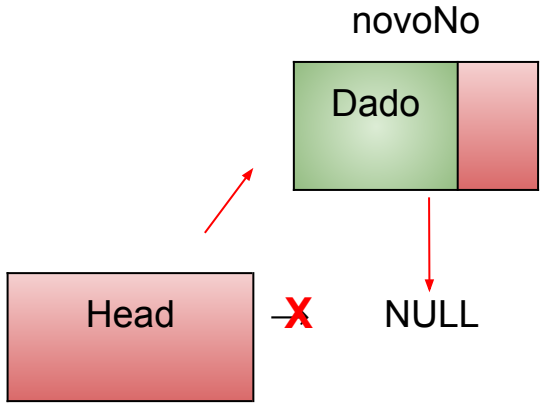
```
28  #Adicionando nó no início da lista
29      def add_begin(self,novoNo):
30          if self.is_empty():
31              self.head = novoNo
32              novoNo.prox = None
33          else:
34              novoNo.prox = self.head
35              self.head = novoNo
```

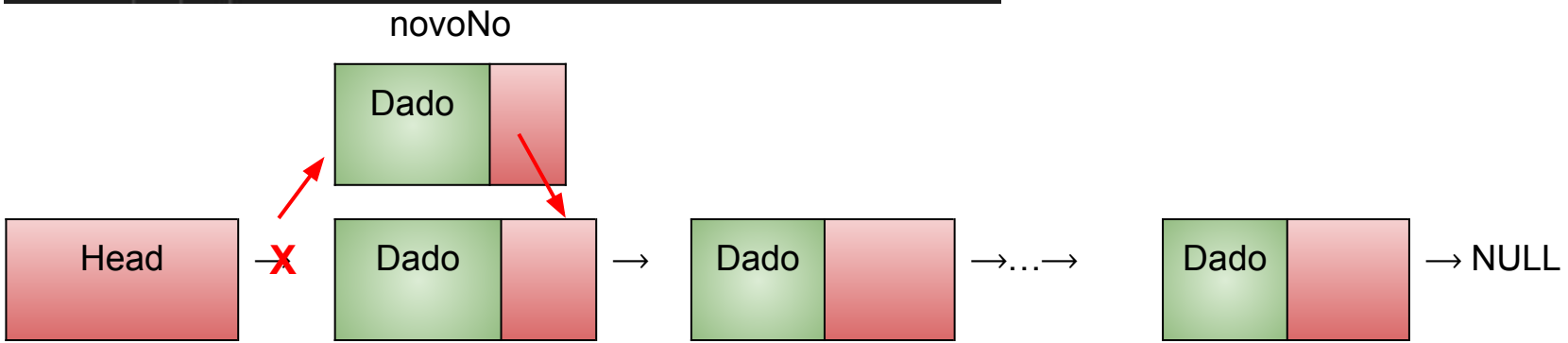




Lista Ligada (Encadeada)

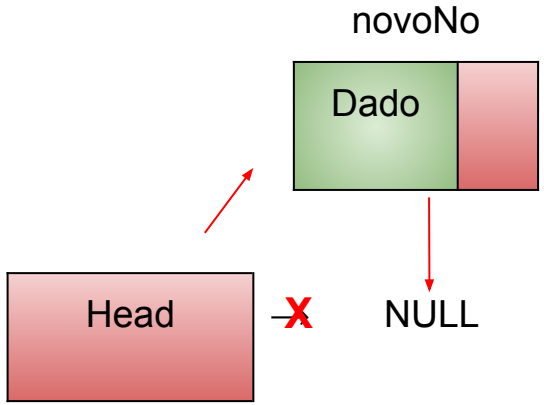
```
28  #Adicionando nó no início da lista
29      def add_begin(self,novoNo):
30          if self.is_empty():
31              self.head = novoNo
32              novoNo.prox = None
33          else:
34              novoNo.prox = self.head
35              self.head = novoNo
```

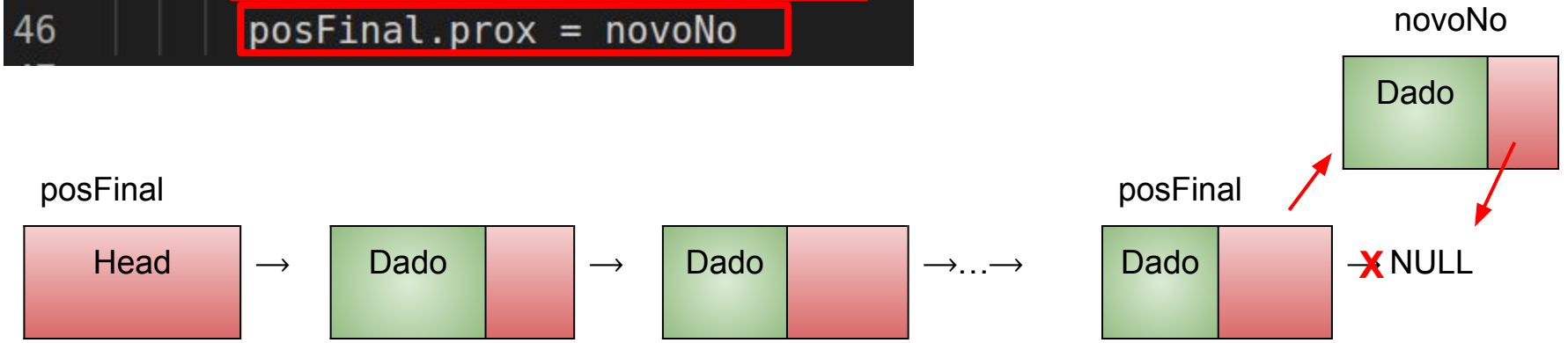




Lista Ligada (Encadeada)

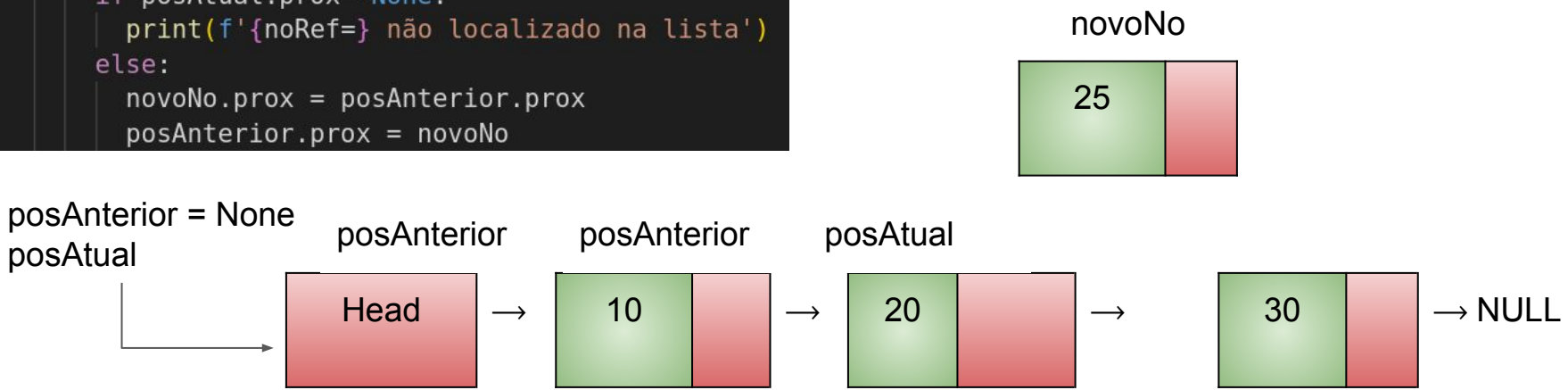
```
38  #Adicionando nó ao final da lista
39      def add_end(self,novoNo):
40          if self.is_empty():
41              self.head = novoNo
42          else:
43              posFinal=self.head
44              while posFinal.prox != None:
45                  posFinal = posFinal.prox
46              posFinal.prox = novoNo
```





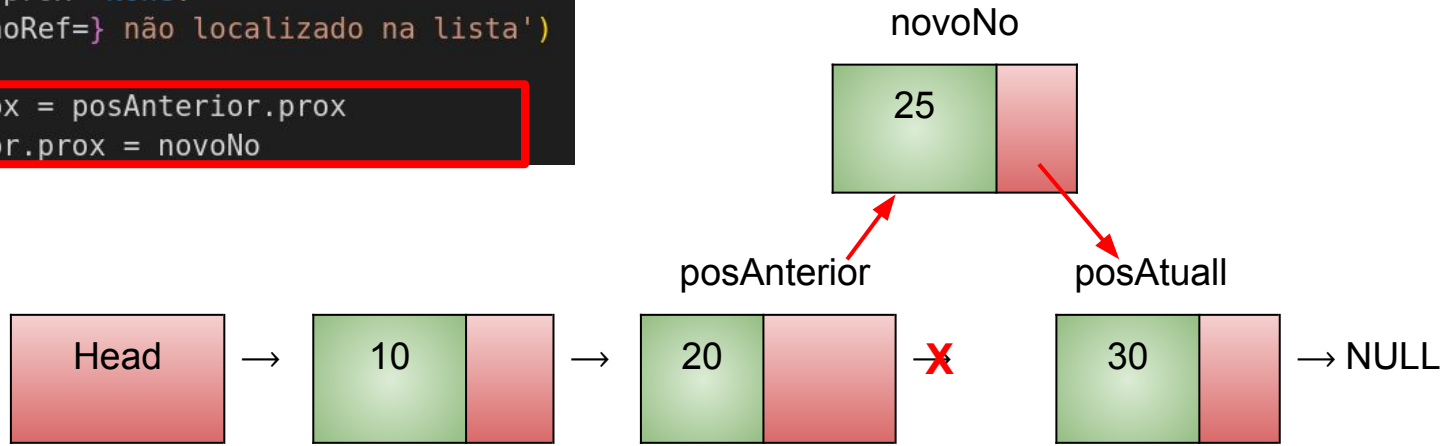
Lista Ligada (Encadeada)

```
48  #Adiciona antes de determinado no
49    def add_before(self,novoNo,noRef):
50      posAtual = self.head
51      posAnterior = None
52      while (posAtual.valor != noRef) :
53        posAnterior = posAtual
54        if(posAtual.prox==None):
55          break
56        posAtual = posAtual.prox
57
58      if posAtual.prox==None:
59        print(f'{noRef=} não localizado na lista')
60      else:
61        novoNo.prox = posAnterior.prox
62        posAnterior.prox = novoNo
```



Lista Ligada (Encadeada)

```
48  #Adiciona antes de determinado no
49    def add_before(self,novoNo,noRef):
50      posAtual = self.head
51      posAnterior = None
52      while (posAtual.valor != noRef) :
53        posAnterior = posAtual
54        if(posAtual.prox==None):
55          break
56        posAtual = posAtual.prox
57
58      if posAtual.prox==None:
59        print(f'{noRef=} não localizado na lista')
60      else:
61        novoNo.prox = posAnterior.prox
62        posAnterior.prox = novoNo
```



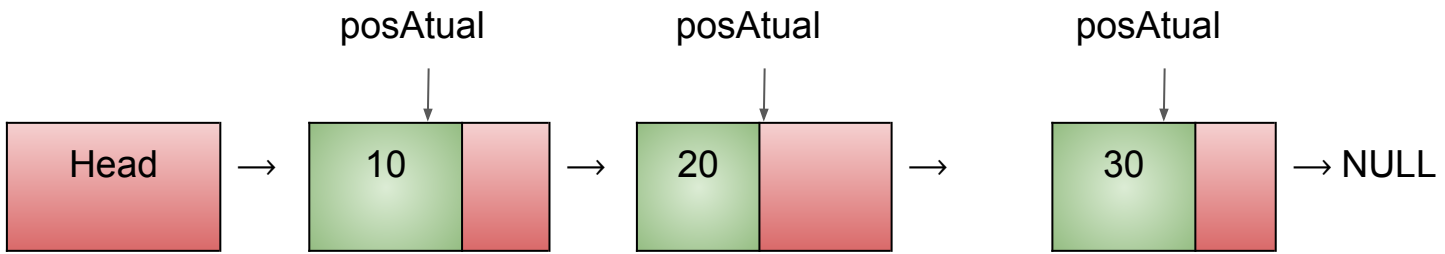
Lista Ligada (Encadeada)

```
64  #Retorna primeiro nó da lista
65    def return_first(self):
66      if self.is_empty():
67        print(f'Erro: Lista vazia')
68        return
69      return self.head.valor
70
71  #Retorna último nó da lista
72    def return_last(self):
73      if self.is_empty():
74        print(f'Erro: Lista vazia')
75        return
76      posFinal = self.head
77      while posFinal.prox!=None:
78        posFinal = posFinal.prox
79      return posFinal.valor
```

Lista Ligada (Encadeada)

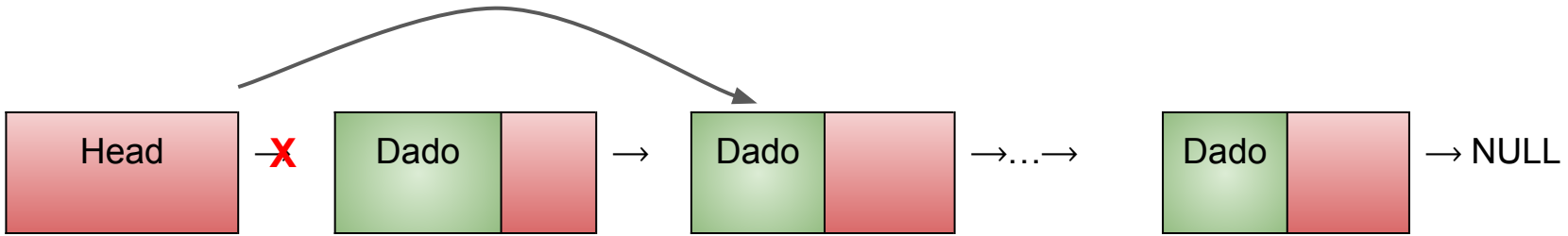
```
81  #Encontrar um elemento ou chave numa lista
82    def find_key(self,chave):
83      if self.is_empty():
84        print(f'Erro: Lista vazia')
85        return
86      posAtual = self.head
87      while posAtual != None:
88        if(posAtual.valor == chave):
89          print(f'Chave {chave} encontrada na lista')
90          break
91        posAtual = posAtual.prox
92      else:
93        print(f'A chave {chave} não foi encontrada na lista')
```

Chave = 30



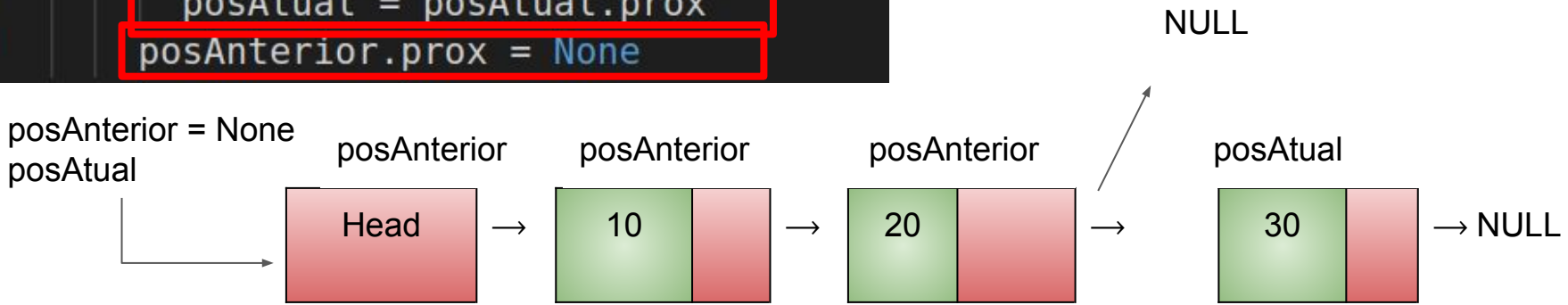
Lista Ligada (Encadeada)

```
94   #Remover o primeiro elemento da lista
95       def remove_first(self):
96           if self.is_empty():
97               print(f'Erro: Lista vazia')
98               return
99           self.head = self.head.prox
```



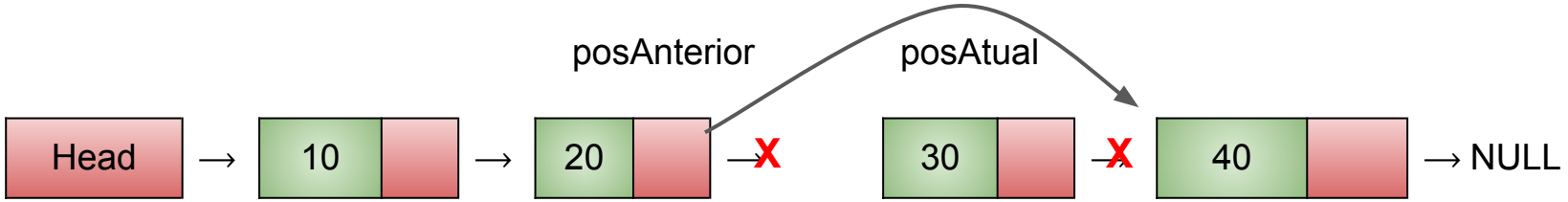
Lista Ligada (Encadeada)

```
101   #Remover o último elemento da lista
102     def remove_last(self):
103       if self.is_empty():
104         print(f'Erro: Lista vazia')
105         return
106       posAtual = self.head
107       posAnterior = None
108       while posAtual.prox != None:
109         posAnterior = posAtual
110         posAtual = posAtual.prox
111       posAnterior.prox = None
```



Lista Ligada (Encadeada)

```
113   #Remove um nó da lista com determinada chave
114     def remove_key(self,chave):
115       if self.is_empty():
116         print(f'Erro: Lista vazia')
117         return
118       posAtual = self.head
119       posAnterior = None
120       while posAtual.valor != chave:
121         posAnterior = posAtual
122         posAtual = posAtual.prox
123       posAnterior.prox = posAtual.prox
```



Pilha

- Estrutura linear de dados que segue a política Last In First Out (LIFO)

Pilha

- Principais operações - Is_empty(): verificar se a pilha está vazia.

```
4   class Pilha():
5       def __init__(self):
6           self.pilha = []
7       #Is_Empty: Verifica se a pilha está vazia:
8       #False, se tem pelo menos 1 elemento
9       #True, se tamanho da pilha é zero
10      def is_empty(self):
11          return not len(self.pilha)
```

Pilha

- Principais operações - push(): incluir um elemento no topo da pilha.

```
13      #Push: Incluir um elemento no topo da pilha.
14      def push(self,e):
15          self.pilha.append(e)
16          return self.pilha
```

Pilha

- Principais operações - pop(): retirar um elemento do topo da pilha

```
18      #Pop: Retirar um elemento do topo da pilha
19      def pop(self):
20          if not self.is_empty():
21              return self.pilha.pop()
22          return "Pilha vazia."
```

Pilha

- Principais operações - top(): retorna valor do elemento no top sem remover.

```
24      #Top: Retornar o valor do elemento no top sem remover.
25      def top(self):
26          if not self.is_empty():
27              return self.pilha[-1]
28          return "Pilha vazia"
```

Pilha

- Implementação com lista ligada

```
1   #Pilha usando lista ligada
2   class No:
3       def __init__(self, valor):
4           self.valor = valor
5           self.prox = None
6
7   class Pilha_Llgd:
8     def __init__(self):
9       self.head = None
```

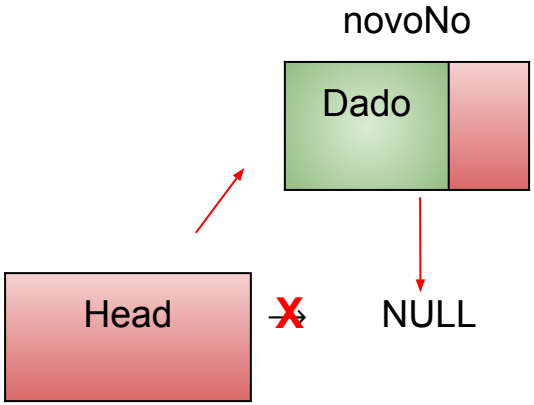
```
11    def is_empty(self):
12      return not self.head
```

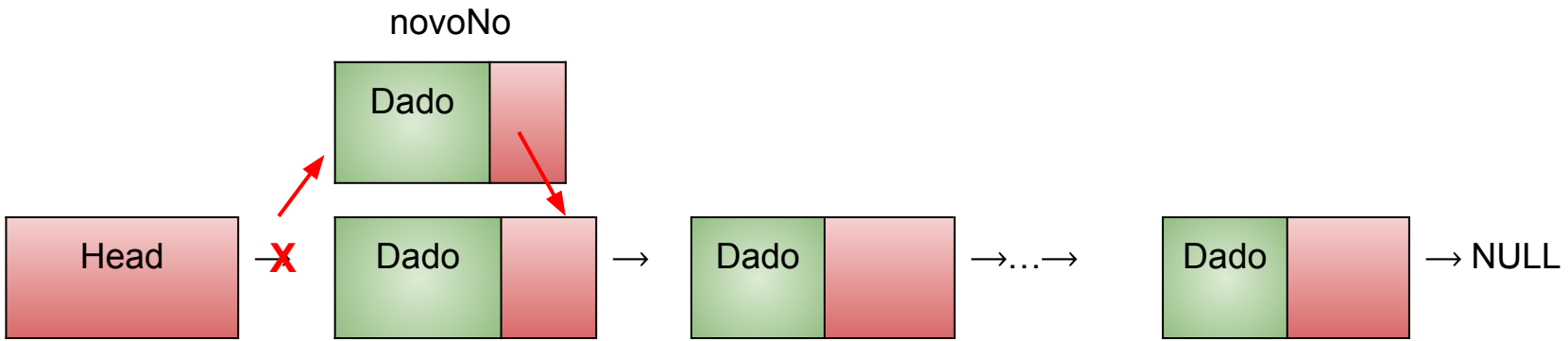
```
30    #Top: Retornar o valor do elemento no top sem remover.
31      def top(self):
32        if self.is_empty():
33          return "Pilha vazia"
34        return self.head.valor
```

Pilha

- Implementação com lista ligada

```
14  #Push: Incluir um elemento no topo da pilha.
15      def push(self,novoNo):
16          if self.is_empty():
17              self.head = novoNo
18              novoNo.prox = None
19          else:
20              novoNo.prox = self.head
21              self.head = novoNo
```

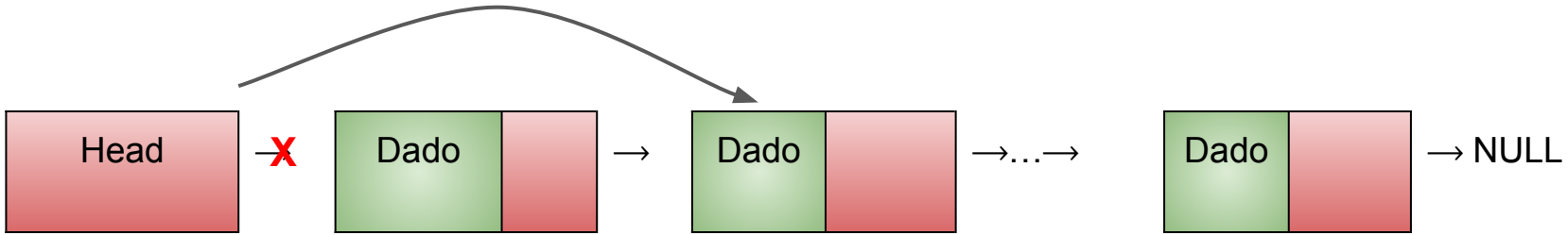




Pilha

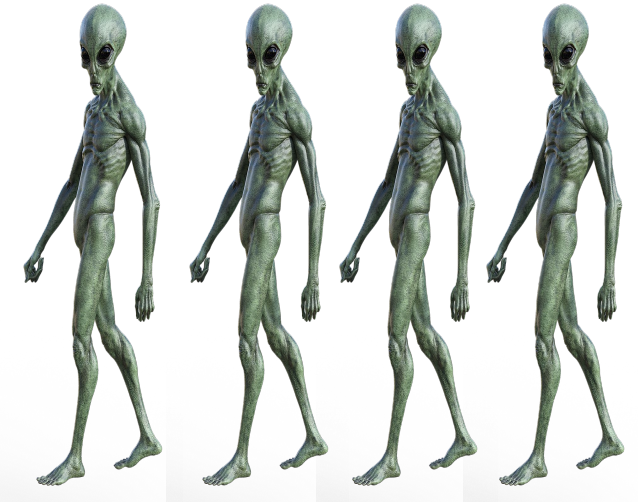
- Implementação com lista ligada

```
23  #Pop: Retirar um elemento do topo da pilha
24      def pop(self):
25          if self.is_empty():
26              return "Pilha vazia"
27          self.head = self.head.prox
28          return "Removido com sucesso"
```



Fila

- Estrutura linear de dados que segue a política First In First Out (FIFO)



Fila

- Operações importantes na manipulação de fila:
  - enqueue(): adiciona um novo elemento no final da fila.
  - dequeue(): remove o primeiro elemento da fila
  - is_empty(): verifica se a fila está vazia (True) ou não (False).
  - peek(): retorna o primeiro elemento da fila sem remover ele.

Fila

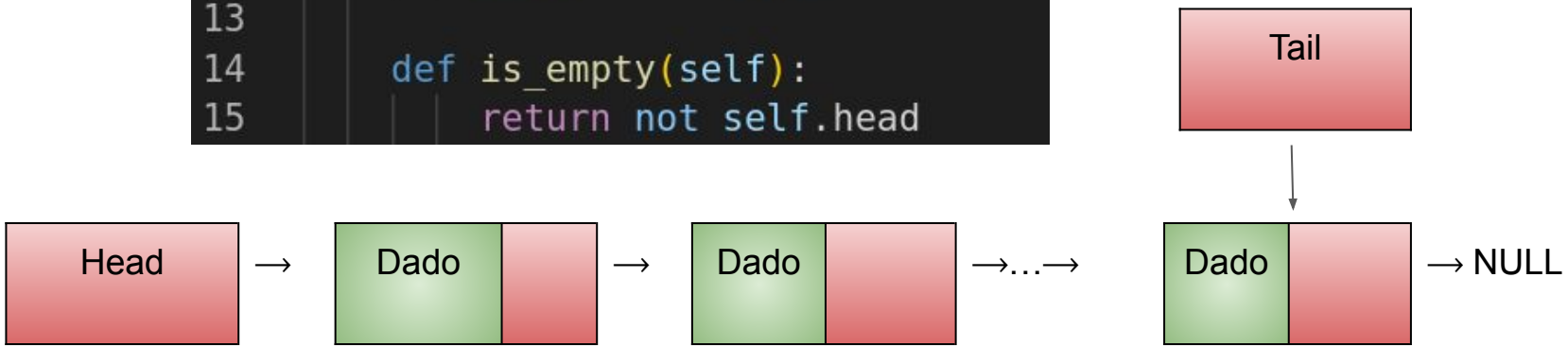
- Operações importantes na manipulação de fila:
  - enqueue(): adiciona um novo elemento no final da fila.
  - dequeue(): remove o primeiro elemento da fila
  - is_empty(): verifica se a fila está vazia (True) ou não (False).
  - peek(): retorna o primeiro elemento da fila sem remover ele.

Fila

```python
#Fila usando listas
class Fila():
    def __init__(self):
        self.fila = []

    def is_empty(self):
        return not len(self.fila)

    def enqueue(self, e):
        self.fila.append(e)
        return self.fila

    def dequeue(self):
        if self.is_empty():
            return "Fila vazia"
        return self.fila.pop(0)

    def peek(self):
        if self.is_empty():
            return "Fila vazia"
        return self.fila[0]
```

Fila

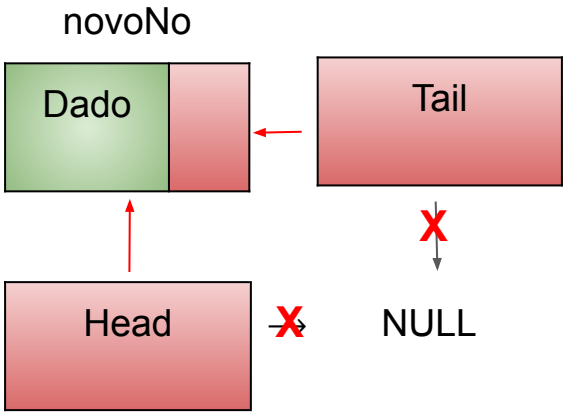
```python
#Fila com lista ligada

class Node:
    def __init__(self, valor):
        self.valor = valor
        self.prox = None


class Fila:
    def __init__(self):
        self.head = None
        self.tail = None

    def is_empty(self):
        return not self.head
```

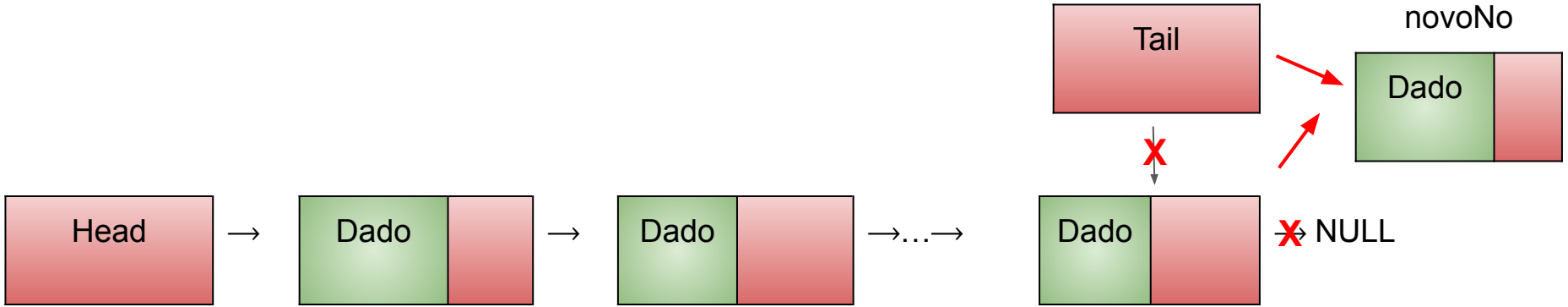


Pilha

- Implementação com lista ligada

```
17      def enqueue(self, novoNo):
18          if self.is_empty():
19              self.head = novoNo
20              self.tail = novoNo
21          else:
22              self.tail.next = novoNo
23              self.tail = novoNo
```

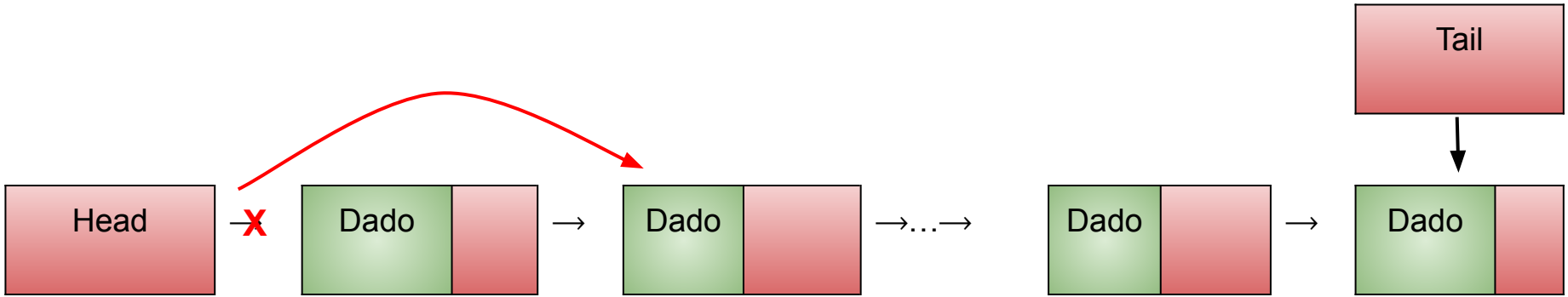




Pilha

- Implementação com lista ligada

```
25      def dequeue(self):
26          if self.is_empty():
27              return "Fila vazia"
28          e = self.head.valor
29          self.head = self.head.prox
30          return e
```



Fila

```
32      def peek(self):
33          if self.is_empty():
34              return  "Fila vazia"
35          return self.head.valor
36
37      def __str__(self):
38        lst = ""
39        posAtual = self.head
40        while(posAtual!=self.tail.prox):
41          lst += f'{posAtual.valor}->'
42          posAtual = posAtual.prox
43        lst +="None"
44        return lst
```