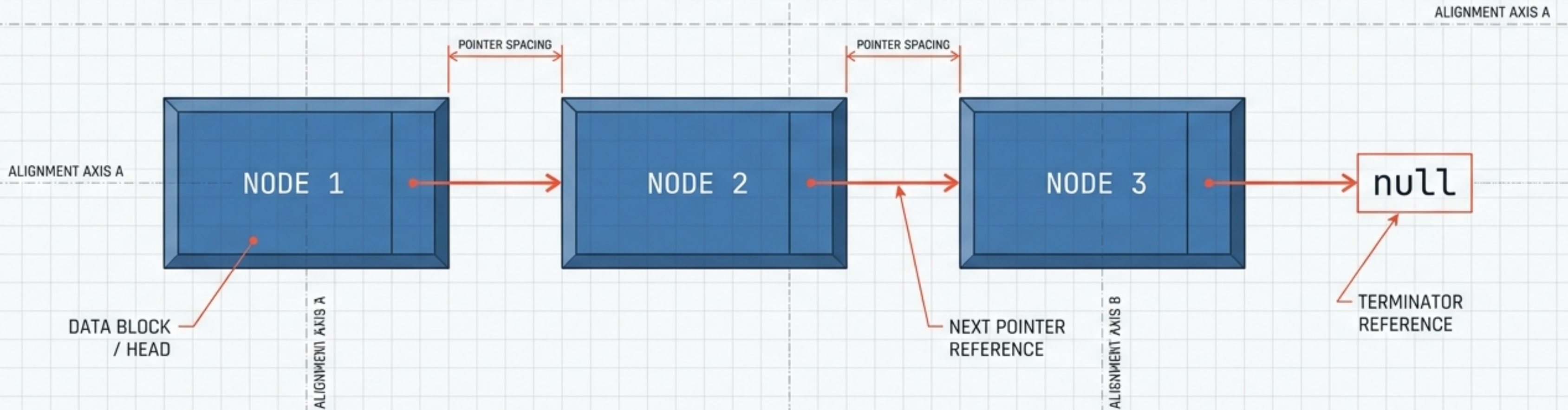


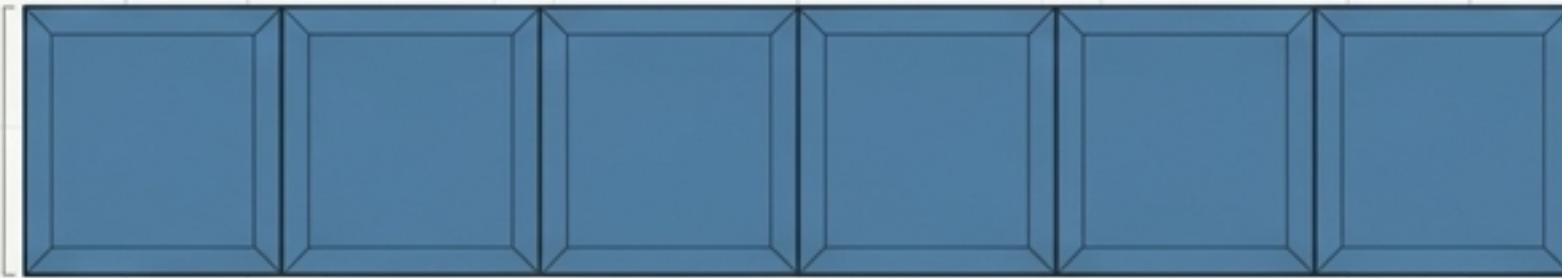
Construindo Listas Ligadas do Zero

Estruturas Dinâmicas de Memória: Da Lógica ao Código Raiz (Sem Java Collections)

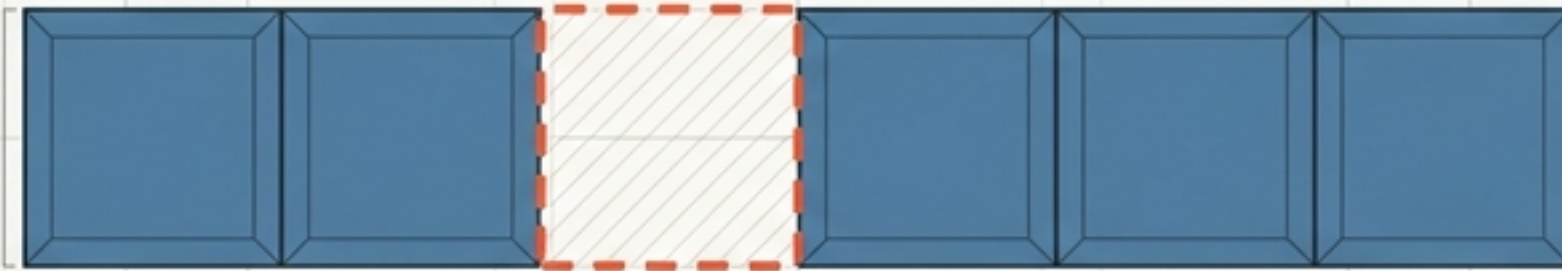


O Problema dos Vetores Estáticos

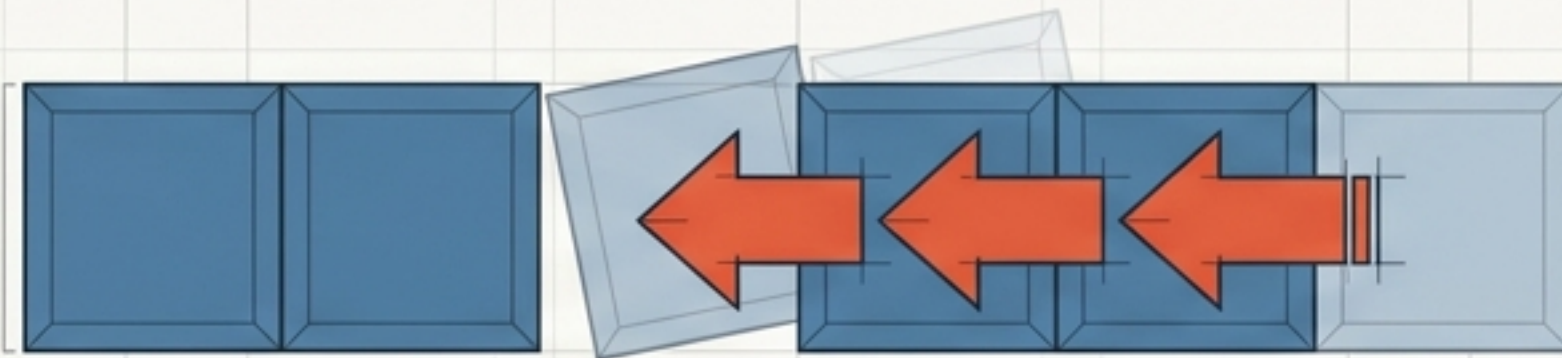
1. Vetor Contínuo →



2. Remoção →



3. O Custo do Deslocamento →



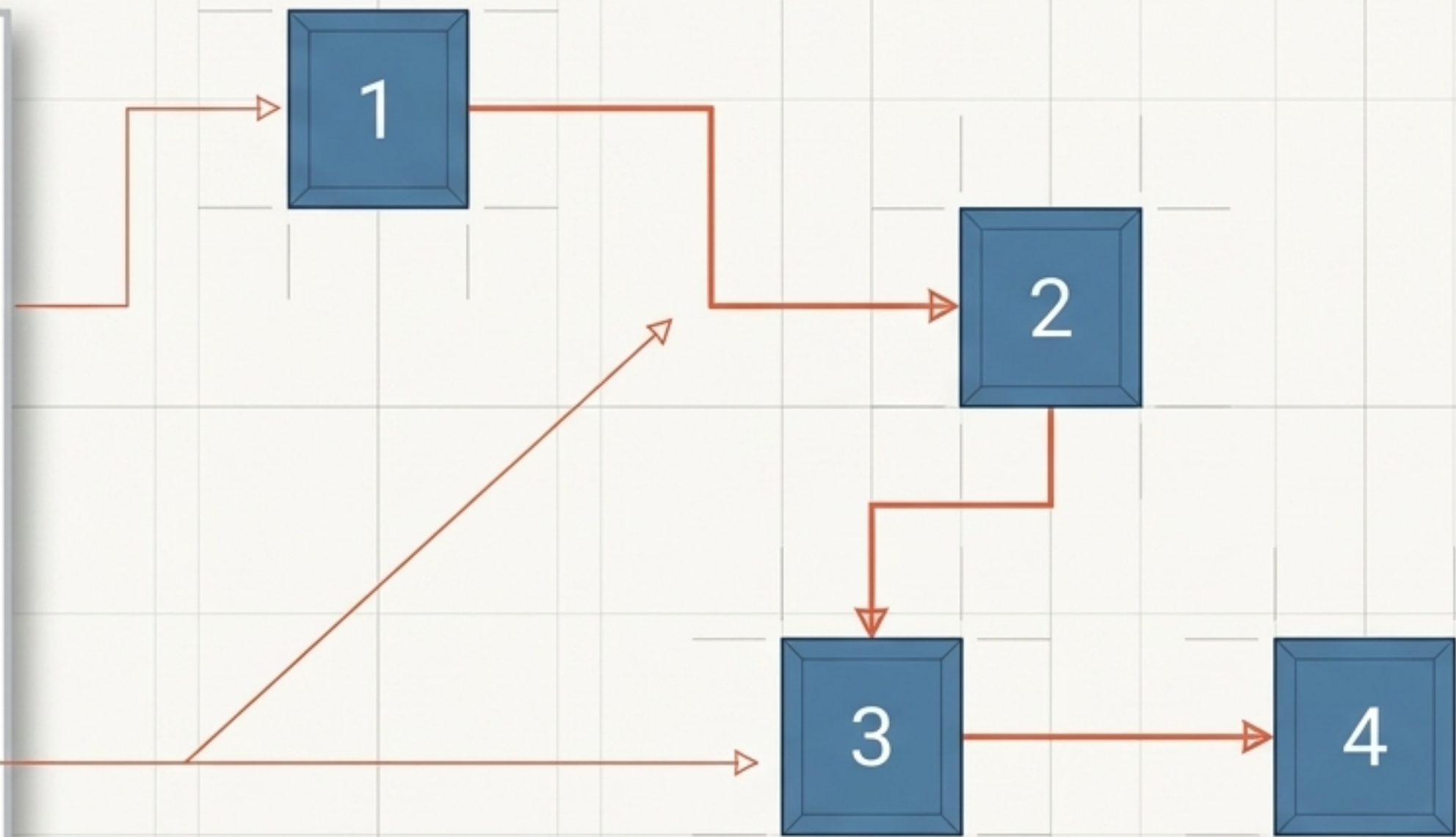
Alterar a posição de todos os itens custa caro. Precisamos de uma estrutura flexível onde a memória não dite a ordem.

A Solução: Alocação Não Sequencial

Não existe índice, apenas conexões.

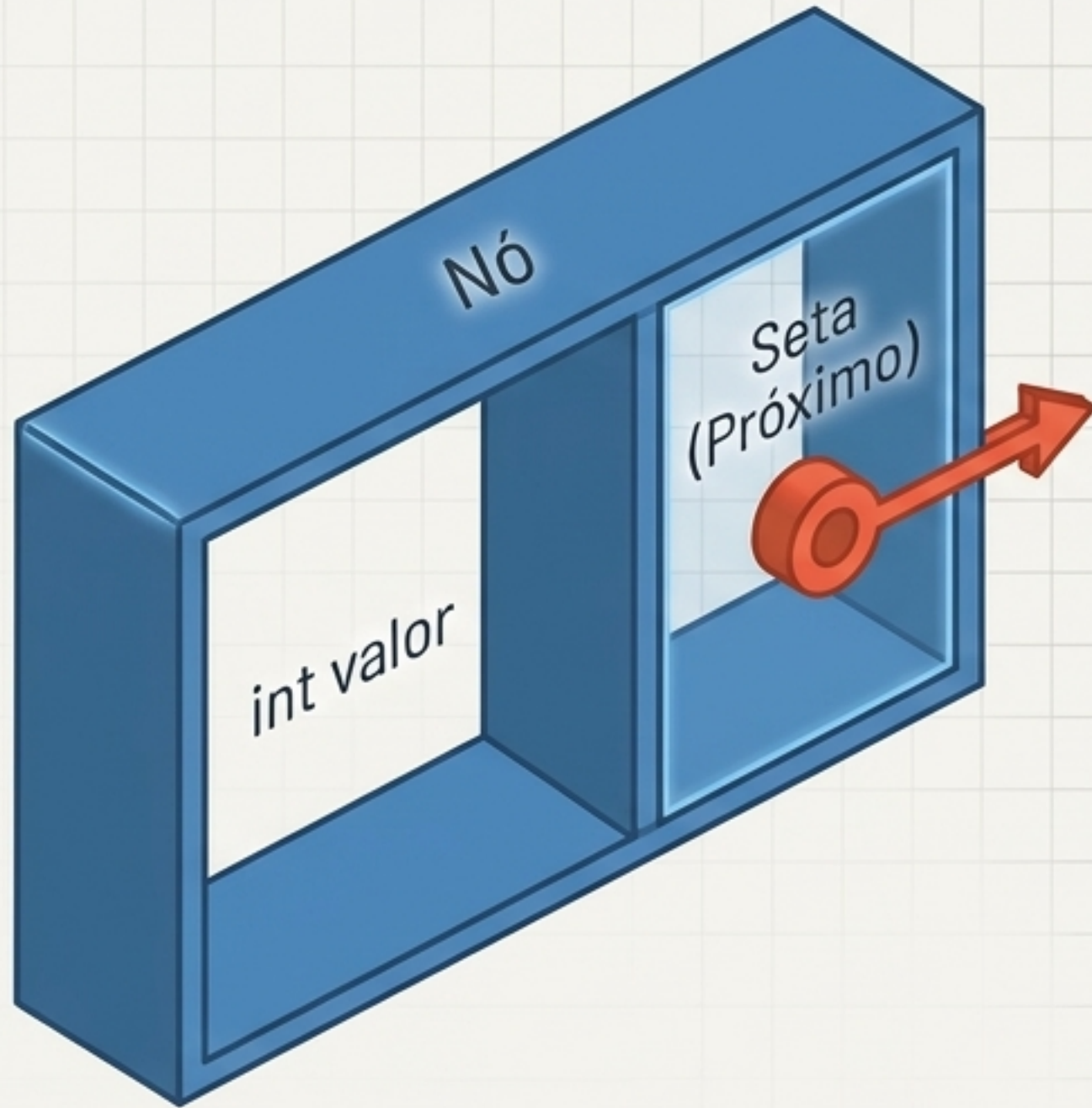
Analogia

Como em uma sala de aula: você aponta para um colega, que aponta para outro. Você não precisa conhecer a sala toda, apenas quem é o próximo.



A Anatomia de um Nó (Conceito vs. Código)

'CONCEITO'



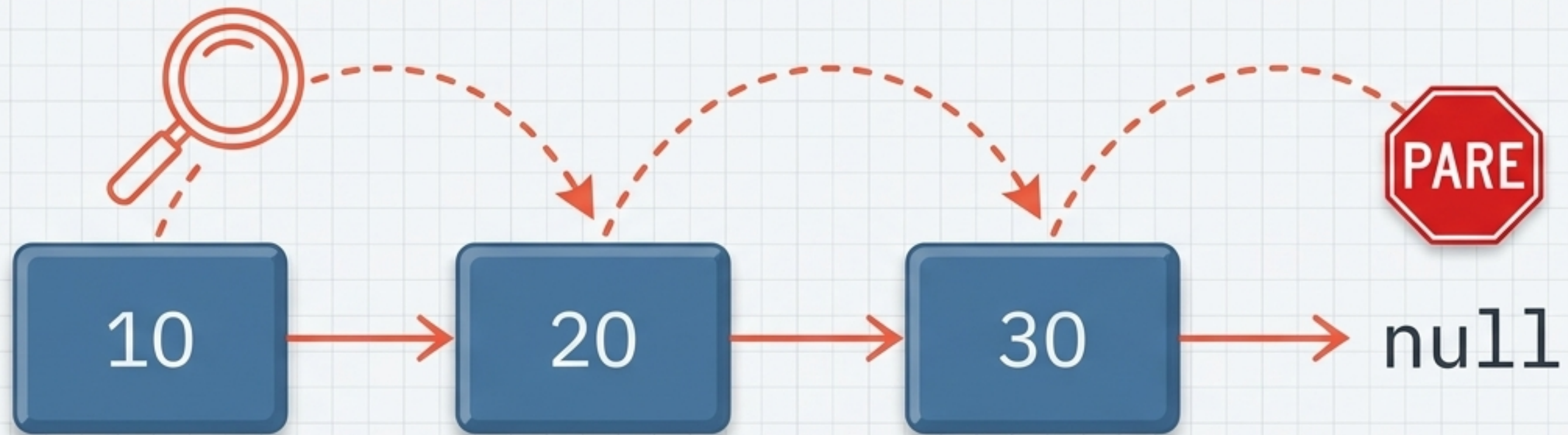
'CÓDIGO'

```
public class No {  
    private int valor;  
    private No prox;  
  
    No(int valor){ this.valor = valor; }  
    // getters e setters...  
}
```

O segredo está aqui: o atributo 'prox' é do próprio tipo 'No'. Ele armazena a referência para um cara igual a ele.

Navegando sem Índices (O Loop Mágico)

Como andar na lista sem o clássico `for(i=0; i<n; i++)`?



Lógica de Travessia: Enquanto o nó atual não apontar para o vazio (null), pule para o próximo.

Imprimindo a Lista (O Código)

A condição de parada:
chegamos no fim do fio.

```
public void imprimirLista(No no) {  
    while(no != null) {  
        System.out.println(no.getValor());  
        no = no.prox;  
    }  
}
```

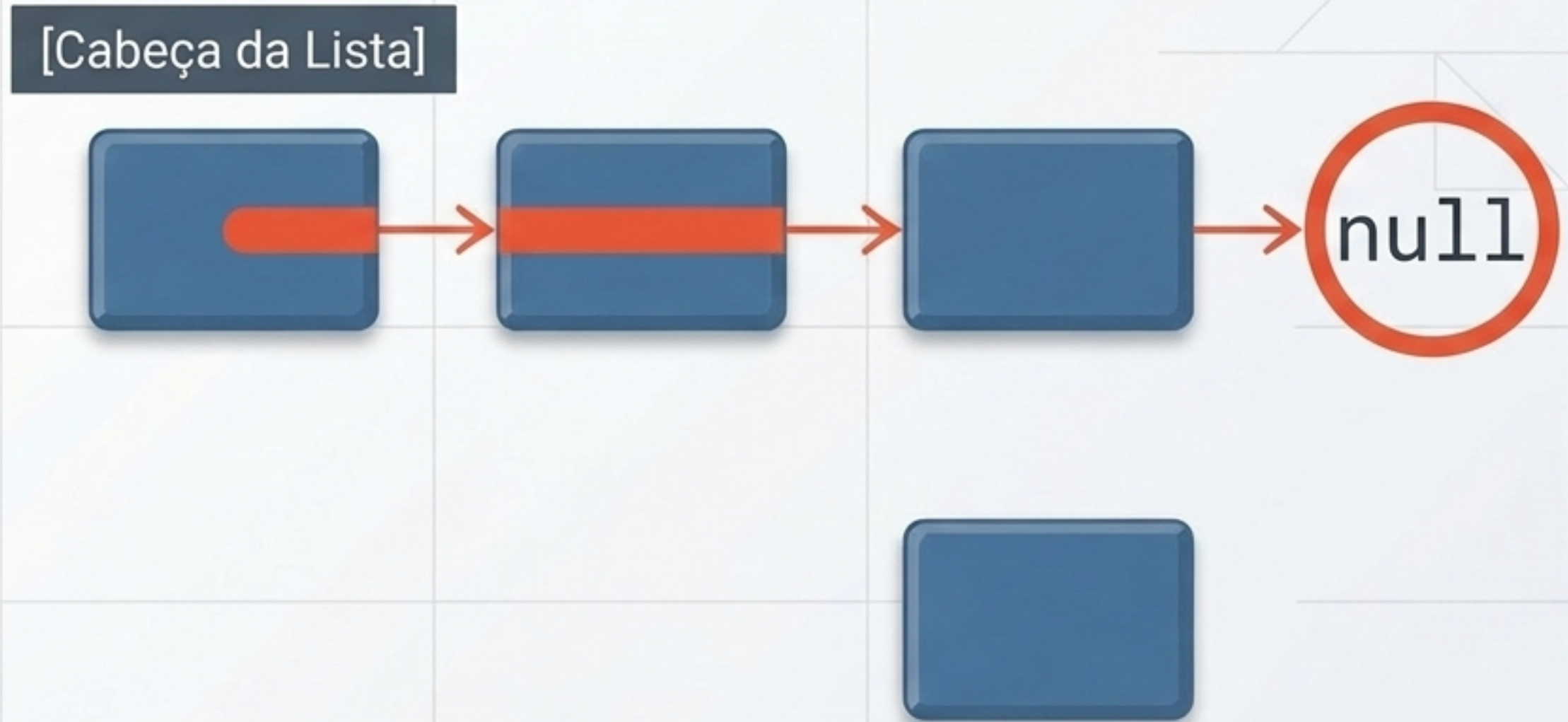


ATENÇÃO: A atualização da referência!
Sem esta linha, você cria um loop infinito
preso eternamente no primeiro elemento.

Inserindo no Final: A Busca pelo Último

O Desafio: Nós só conhecemos a 'Cabeça' da lista.

A Ação: Para inserir no fim, precisamos varrer a lista inteira até encontrar o único cara que aponta para o nada.

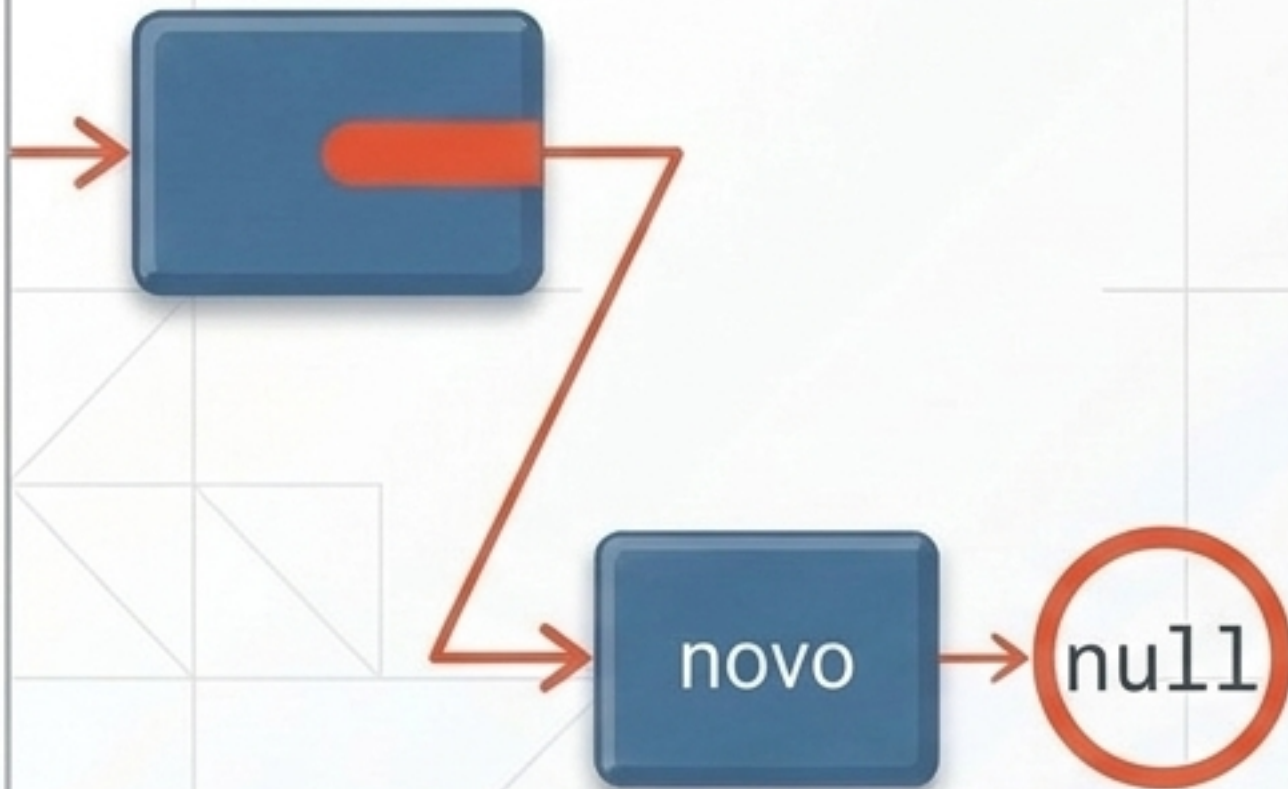


Inserindo no Final (O Código)

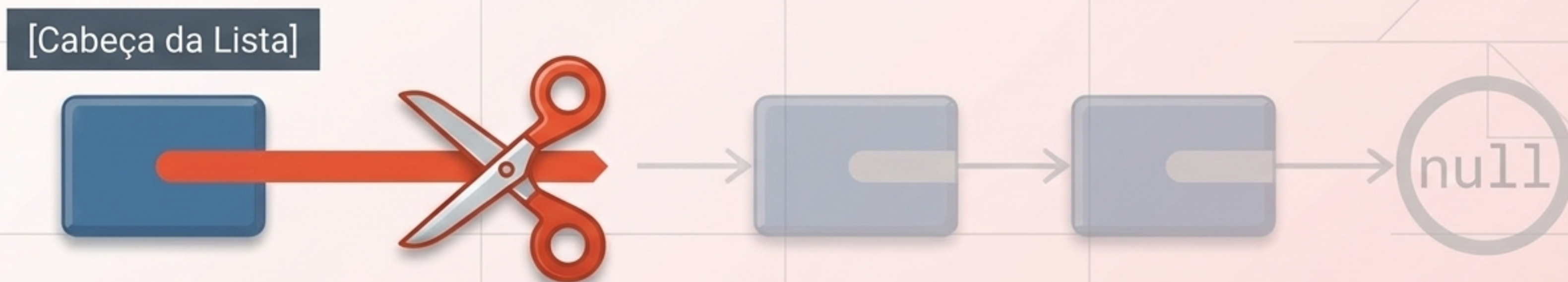
Varre a lista: "Quem eu estou olhando agora tem um próximo?"

```
while(inicio.getProx() != null) {  
    inicio = inicio.getProx();  
}  
inicio.setProx(novo);
```

Achei o último! Agora, ele deixa de apontar para null e aponta para o nosso novo nó.



CUIDADO CRÍTICO: O Limbo da Memória



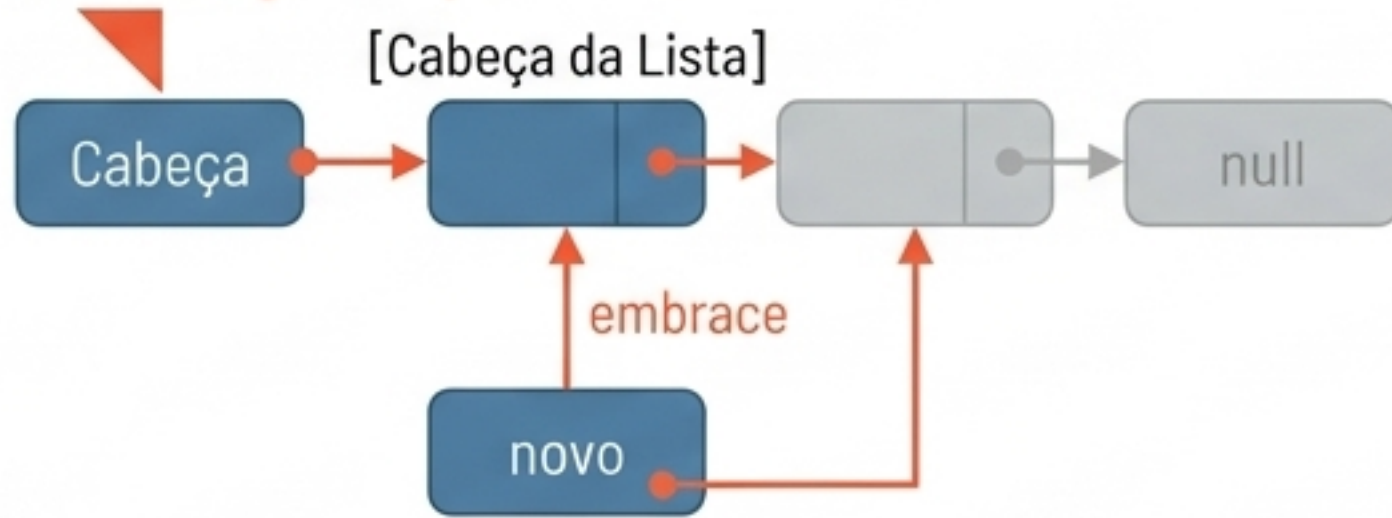
PERIGO: Se você perder a referência de um nó, você perde a lista toda. **1 milhão** de itens vão pro espaço sem chance de recuperação.

A ordem das ligações importa absolutamente tudo na inserção no início.

Inserindo no Início (A Lógica Correta)

A Dança das Cadeiras

Passo 1 (Segure o próximo)



Passo 2 (Solte o anterior)



A Regra de Ouro da Inserção:

Primeiro o novo nó abraça a lista.
Só depois a cabeça da lista passa
a apontar para ele.

Inserindo no Início (O Código)

```
novo.setProx(inicio.getProx()); // Passo 1
inicio.tProx(inicio.getProx()); // Passo
inicio.setProx(novo); // Passo 2
```



[1] Passo 1: O 'prox' do novo nó segura firmemente o resto da lista. (A lista está a salvo)

[2] Passo 2: Agora é seguro atualizar a cabeça para apontar para o novo nó.

A Regra de Ouro: Validar a Lista Vazia

Usado em
inserirNoFinal



```
if (inicio.getProx() == null) {  
    inicio.setProx(novo);  
    return;  
}
```



Usado em
inserirNoInicio

Dica Raiz: Nunca tente acessar o `'.getProx()'` de algo que não existe. Sempre **valide** se a cabeça aponta para **nulo** antes de tentar percorrer a lista para evitar um fatal **NullPointerException**.

A Lista em Ação: Classe Principal

```
public static void main(String args[]) {  
    ○ No inicio = new No();  
    inicio.inserirNoInicio(inicio, 50);  
    inicio.inserirNoFinal(inicio, 60);  
    inicio.inserirNoFinal(inicio, 70);  
    inicio.inserirNoInicio(inicio, 30);  
    inicio.inserirNoInicio(inicio, 20);  
    inicio.inserirNoFinal(inicio, 80);  
    inicio.imprimirLista(inicio.getProx());  
}
```

Aqui damos o 'play'.
Criamos o Nó
Sentinela (uma
cabeça vazia)
apenas para segurar
a referência
principal.

As operações
disparam as
alocações
dinâmicas na
memória.

0 Teste de Mesa Visual

Estado Inicial	
Após Inserções Base (50, 60, 70)	
Após Início (30, 20)	
Após Final (80)	

```
> 20  
> 30  
> 50  
> 60  
> 70  
> 80
```

← A saída no console reflete
perfeitamente a arquitetura
que montamos fio a fio.

Missão Cumprida (Prática Raiz)

- [] Refaça as classes 'No' e 'Principal' do zero sem copiar.
- [] Desenhe caixas e setas no papel antes de digitar uma única linha de código.
- [] Tente criar um método 'removeDoFinal()' usando a mesma lógica visual.

Não tem magia, tem ponteiros! Domine a ligação e você domina a memória.