

CMP1054 – EDI

Java

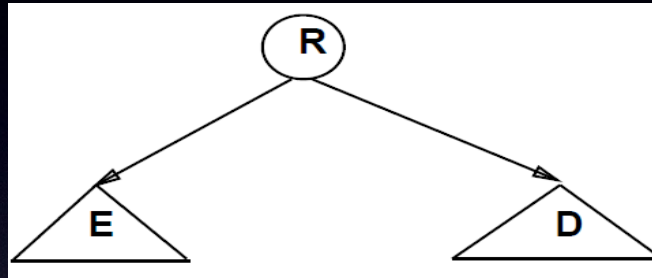
Árvores Binárias de Pesquisa.

Construtor, vazia, pesquisar e inserir.

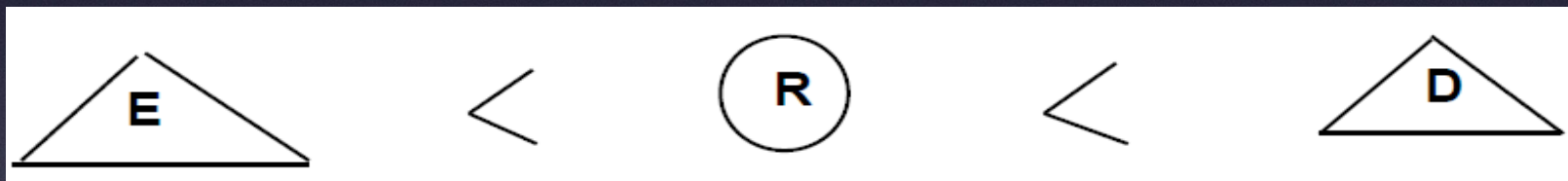
Prof. Dr. José Olimpio Ferreira

Árvores Binárias de Pesquisa

- Para qualquer nó que contenha um registro



- Temos a relação invariante



- Todos os registros com chaves menores estão na sub-árvore à esquerda.
- Todos os registros com chaves maiores estão na sub-árvore à direita.

Árvores Binárias de Pesquisa

- Definição
- Operações
 - Pesquisar (Buscar)
 - Verificar se está vazia
 - Inserir
 - Remover
 - Nó folha
 - Nó com um filho
 - Nó com dois filhos
 - Questões de implementação

Item

- Classe Item

- Atributos

- String nome, fone

- Métodos

- Item(String, String)
- String getItem()
- String getNome()

Nome
Fone

classe Item

```
public class Item {  
    private String nome, fone;  
    public Item(String pNome, String pFone) {  
        nome = pNome;  
        fone = pFone;  
    }  
    public String getNome() { return nome; }  
    public String getFone() { return fone; }  
    @Override  
    public String toString(){  
        return (nome + "\n" + fone + "\n");  
    }  
}
```

Abp

- Classe Abp

- Classe No

- Atributos

- pai, fe, fd

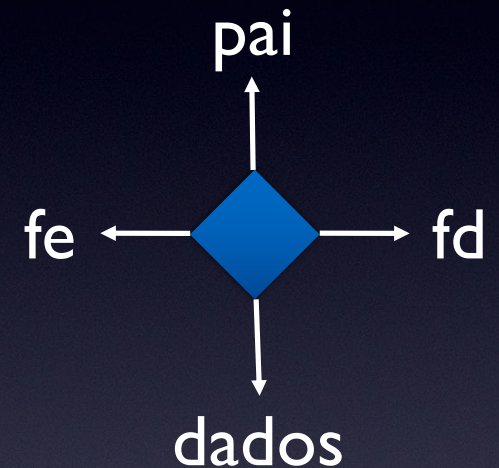
- do tipo referencia a No

- dados

- do tipo referencia a Item

- Métodos

- No(Item)



Abp

● Atributos

- tamanho
 - int
- raiz
 - referencia a No

● Métodos

- Abp()
- boolean vazia()
- No consultar(Item)
- Item pesquisar(Item)
- boolean inserir(Item)

- No mínimo(No)
- No máximo(No)
- No antecessor(No)
- No sucessor(No)
- Item remover(Item)
- String visitaOrdem()
- String visitaPreOrdem()
- Strint visitaPosOrdem()

classe Abp

```
package ed1;
public class Abp {
    private class No {
        private Item dados;
        private No fd, fe, pai;
        public No(Item aux) {
            dados = aux;
            fd = fe = pai = null;
        }
    }
    private No raiz;
    private int tamanho;
    public Abp() {
        raiz = null; tamanho = 0;
    }
    public int getTamanho() {
        return tamanho;
    }
    public boolean vazia() {
        return (raiz == null);
    }
    private No consultar(Item obj) {
        // implementar depois
        return null;
    }
    public Item pesquisar(Item obj) {
        // implementar depois
        return null;
    }
    public boolean inserir(Item obj) {
        // implementar depois
        return true;
    }
}
```

```
private No maximo(No obj) {
    // implementar depois
    return obj;
}
private No minimo(No obj) {
    // implementar depois
    return obj;
}
private No antecessor(No obj) {
    // implementar depois
    return obj;
}
private No sucessor(No obj) {
    // implementar depois
    return obj;
}
public Item retirar(Item obj) {
    // implementar depois
    return obj;
}
public void visitaEmOrdem(StringBuffer aux) {
    // chamar método recursivo
}
public void visitaEmPreOrdem(StringBuffer aux) {
    // chamar método recursivo
}
public void visitaEmPosOrdem(StringBuffer aux) {
    // chamar método recursivo
}
public void testaIntegridade(StringBuffer aux) {
    // chamar método recursivo
}
}
```

}

Abp()

```
public Abp() {
```

```
    raiz = null; // inicializa a raiz com null  
    tamanho = 0;
```

```
}
```

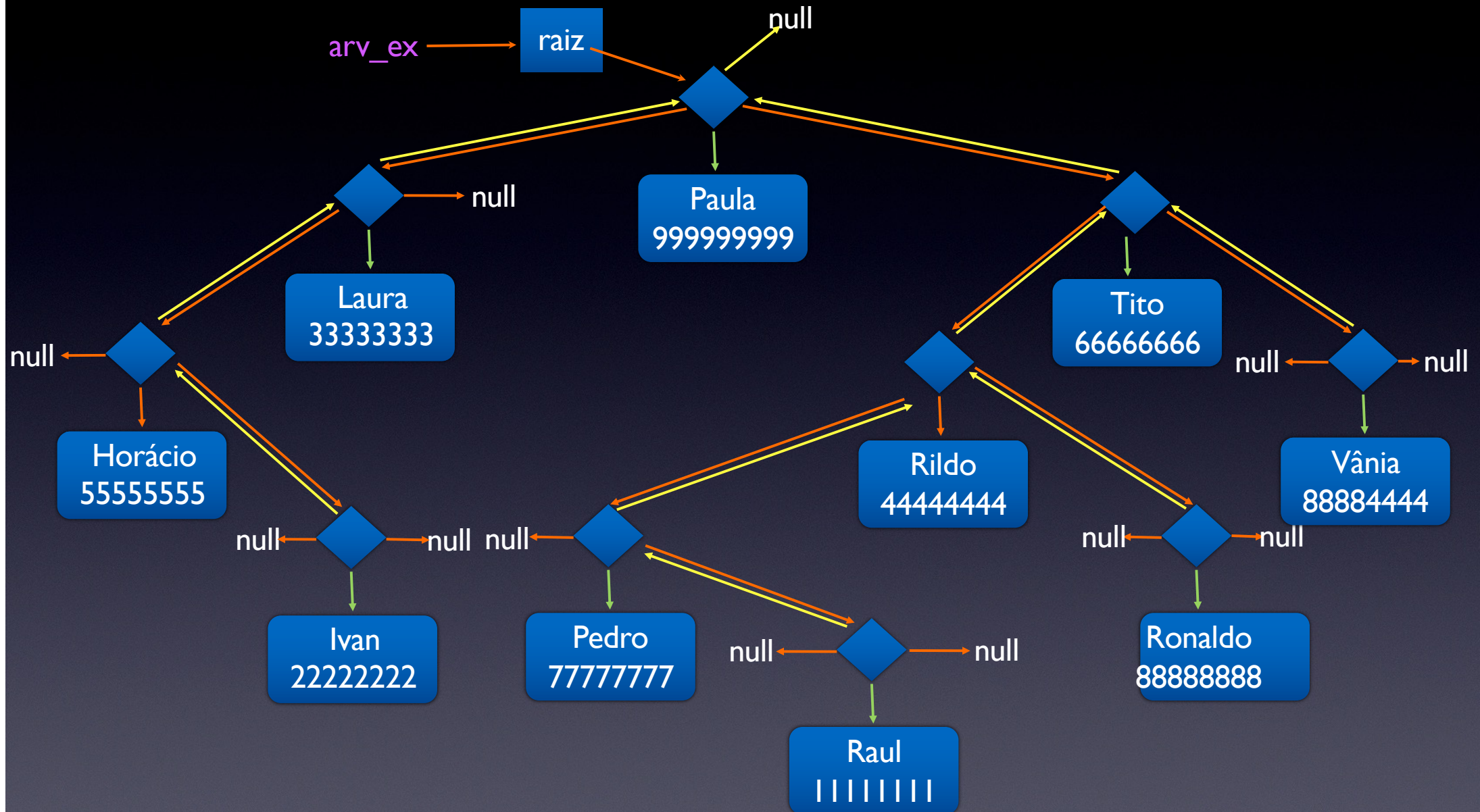


vazia()

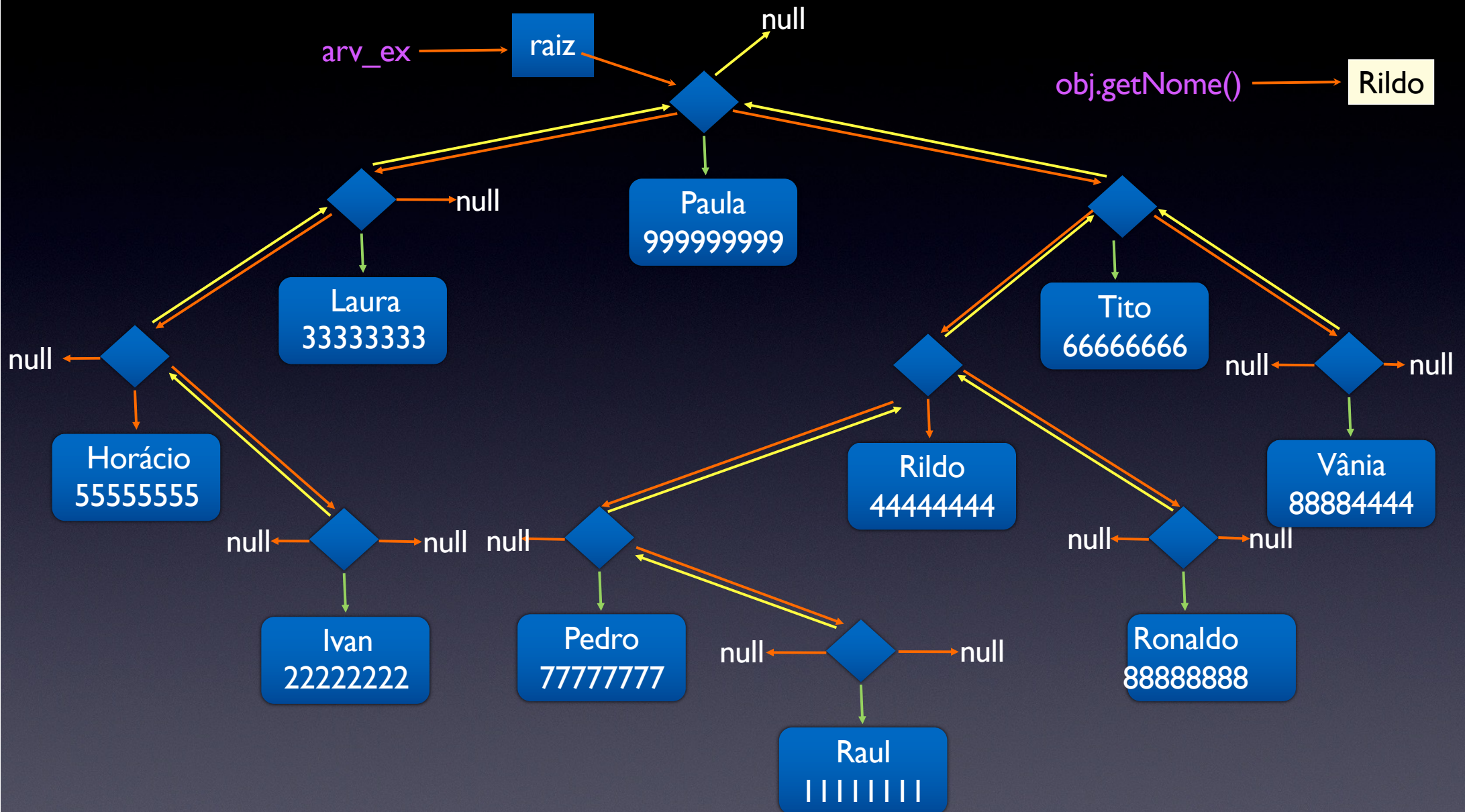
```
public boolean vazia() {  
    return ( raiz == null ); // retorna true  
    caso raiz == null  
}
```



Árvore Binária de Pesquisa



No consultar(*Item* obj) e *Item* pesquisar(*Item* obj)



private No consultar(Item obj)

// é privada e retorna um No

```
private No consultar(Item obj) {  
  
    No aux = raiz;  
  
    while ( aux != null ){  
        if (obj.getNome().compareTo(aux.dados.getNome()) < 0) aux =  
aux.fe;  
        else  
            if (obj.getNome().compareTo(aux.dados.getNome()) > 0)  
aux = aux.fd;  
        else return aux; //Sucesso(encontrou)  
    }  
  
    return null; // Insucesso --> null  
}
```

public Item pesquisar(Item obj)

// é pública e retorna um Item

```
public Item pesquisar(Item obj) {
```

```
    No aux = consultar(obj);
```

```
    if ( aux == null ) {
```

```
        return null //Insucesso → não encontrou
```

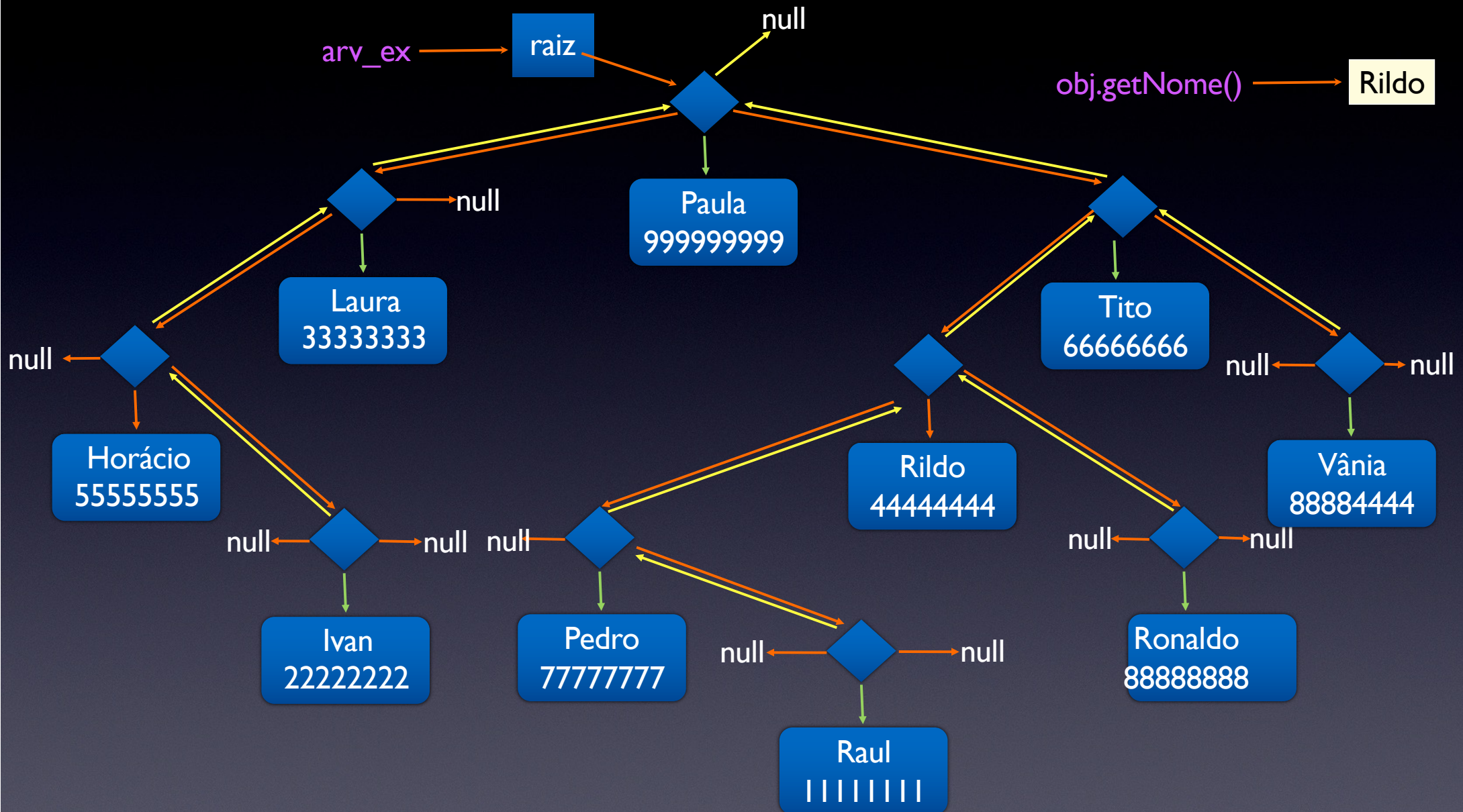
```
    }
```

```
        return (new Item(aux.dados.getNome(), aux.dados.getFone()));
```

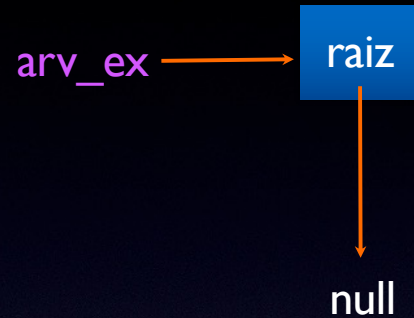
```
    //Ssucesso --> encontrou
```

```
}
```

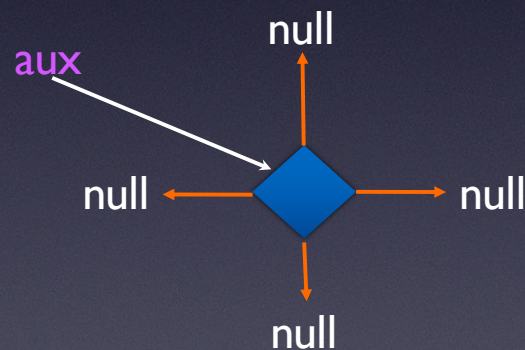

No consultar(*Item* obj) e *Item* pesquisar(*Item* obj)



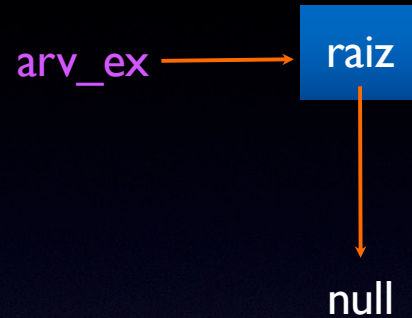
boolean inserir(Item obj)



- Cria-se o No e armazena-se seu endereço em aux (do tipo No).



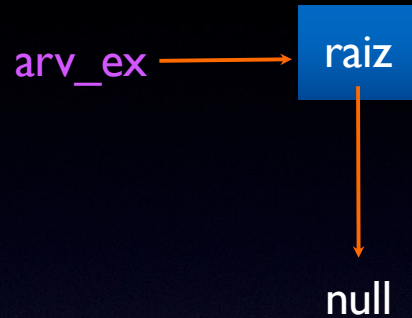
boolean inserir(Item obj)



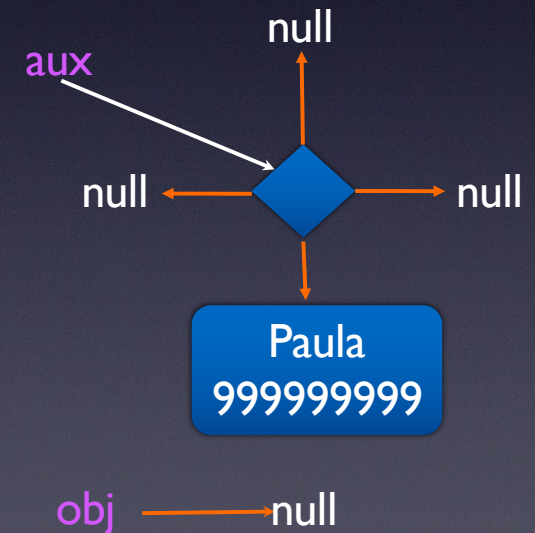
- Liga-se o Item referenciado por novo ao No referenciado por aux.



boolean inserir(Item obj)



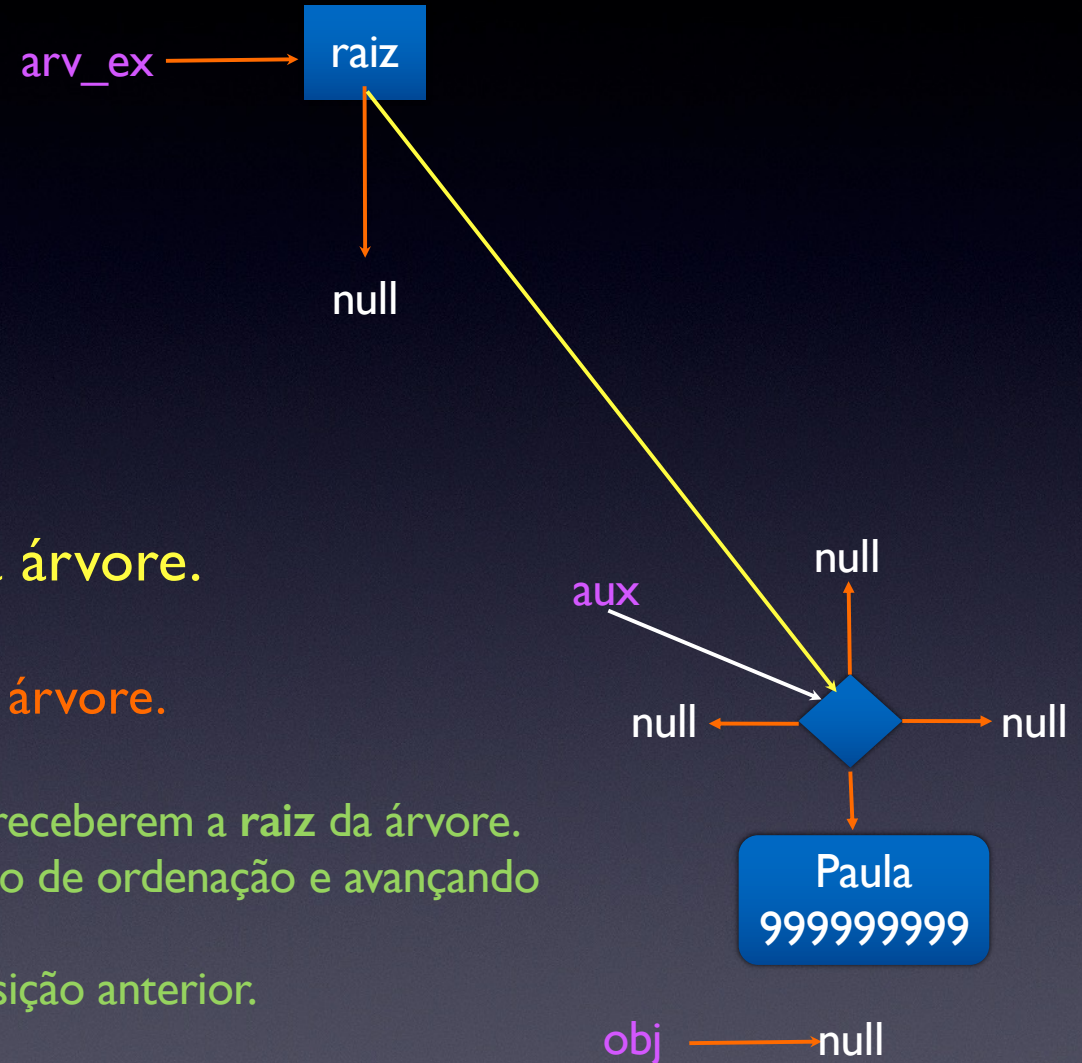
- Liga-se o Item referenciado por `obj` ao No referenciado por `aux`.
- Faz-se `obj` receber `null`.



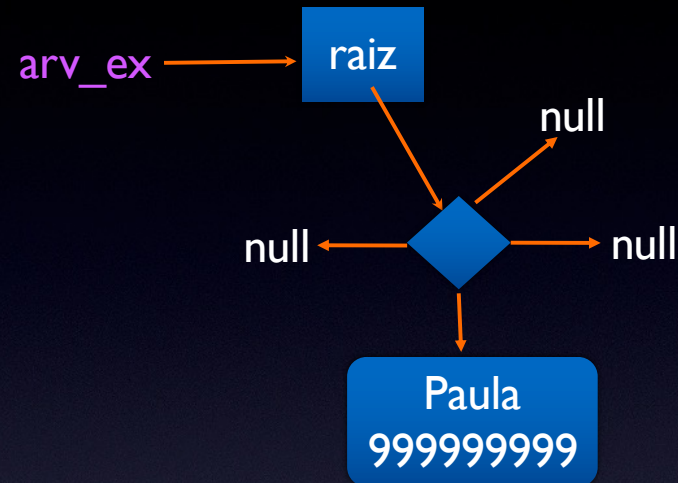
boolean inserir(Item obj)

- Liga-se o No referenciado por **aux** à árvore.

- Verifica se a árvore é vazia.
- Caso seja vazia o No **aux** será a **raiz** da árvore.
- Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

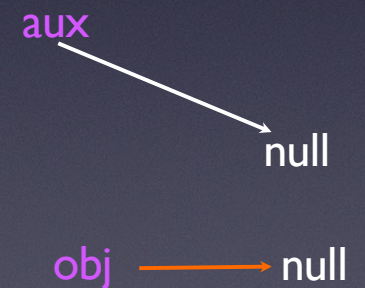


boolean inserir(Item obj)

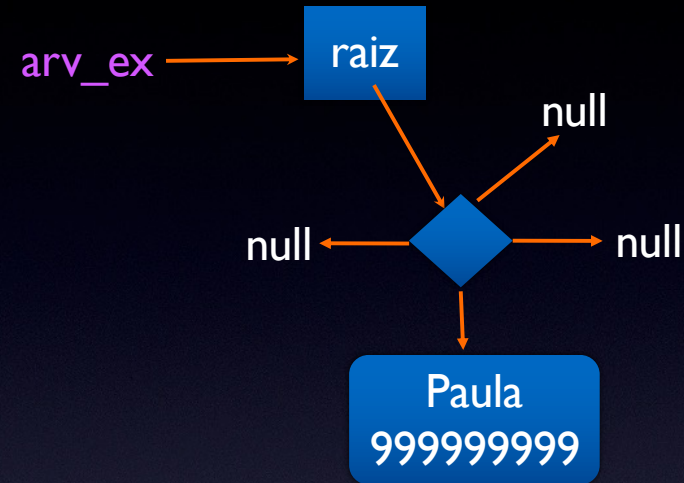


- Liga-se o No referenciado por **aux** à árvore.

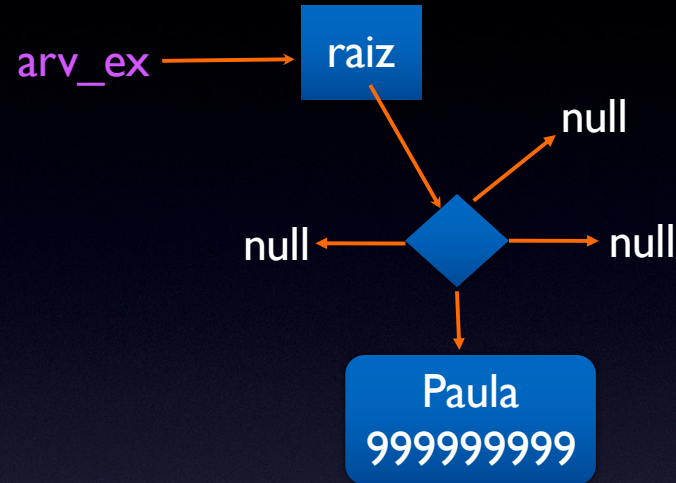
- Verifica se a árvore é vazia.
- Caso seja vazia o No **aux** será a **raiz** da árvore.
- Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.



Árvore após Primeira inserção



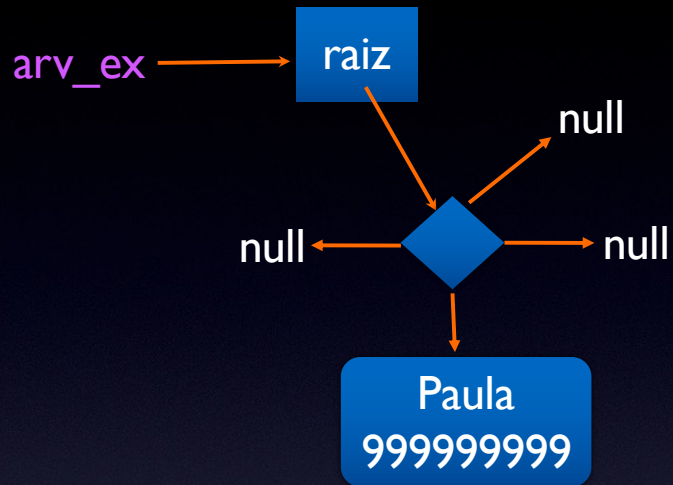
boolean inserir(Item obj)



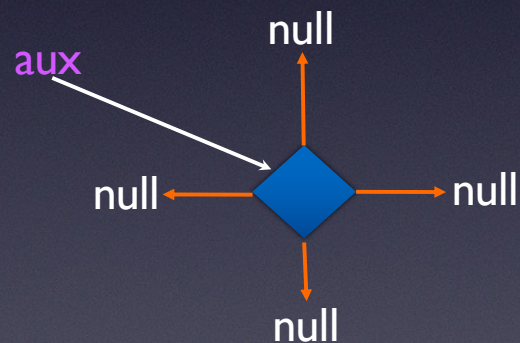
- Na chamada do método inserir recebe o objeto novo do tipo Item



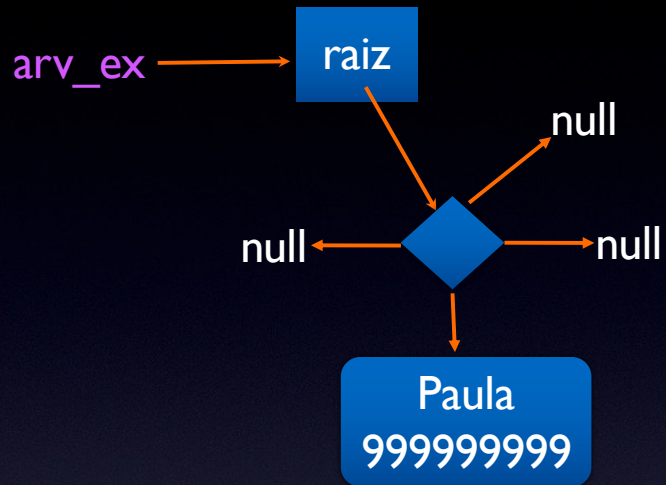
boolean inserir(Item obj)



- Cria-se o No e armazena-se seu endereço em aux (do tipo No).



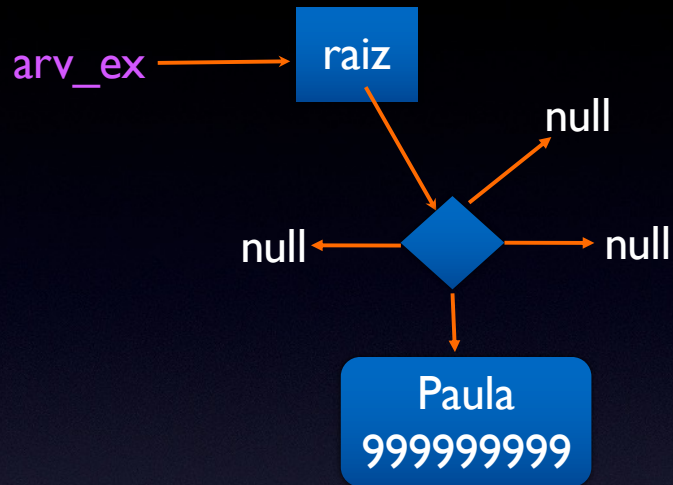
boolean inserir(Item obj)



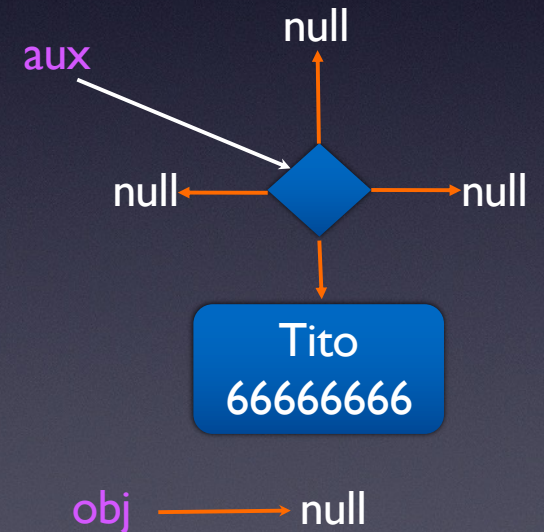
- Liga-se o Item referenciado por novo ao No referenciado por aux.



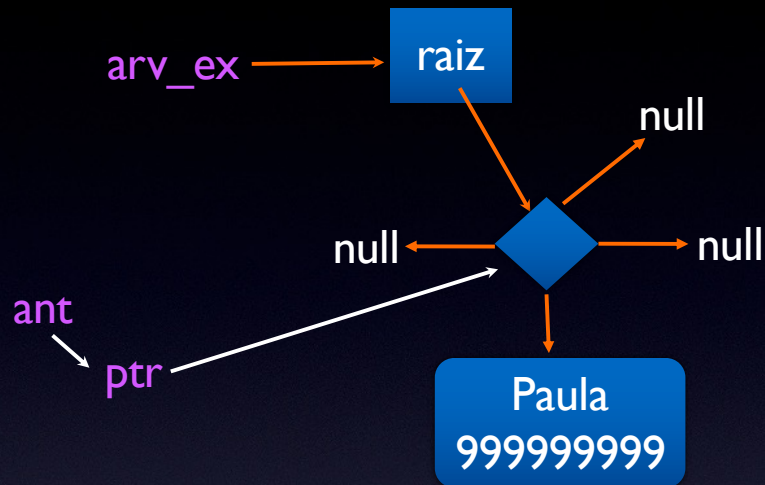
boolean inserir(Item obj)



- Liga-se o Item referenciado por novo ao No referenciado por aux.
- Faz-se obj receber 0.

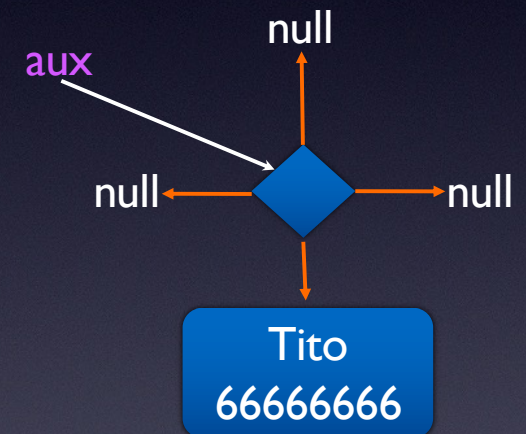


boolean inserir(Item obj)

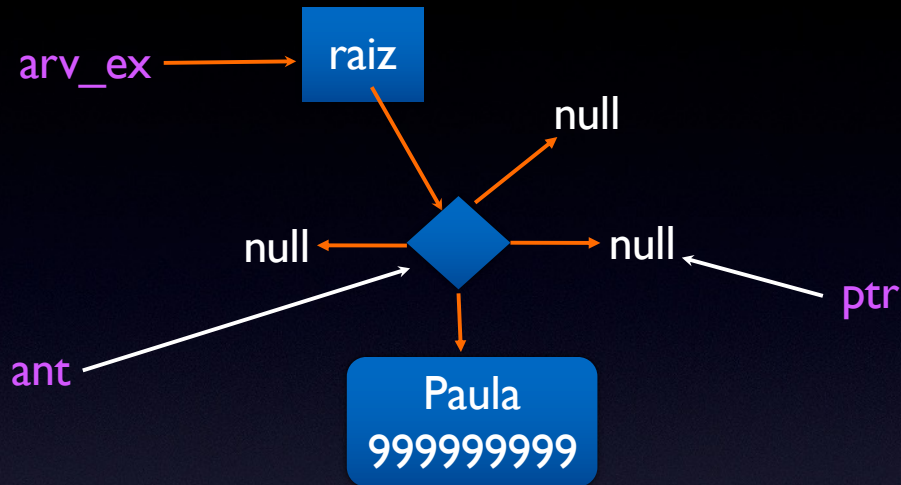


- Liga-se o No referenciado por **aux** à árvore.

- Verifica se a árvore é vazia.
- Caso seja vazia o No novo será a **raiz** da árvore.
- Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

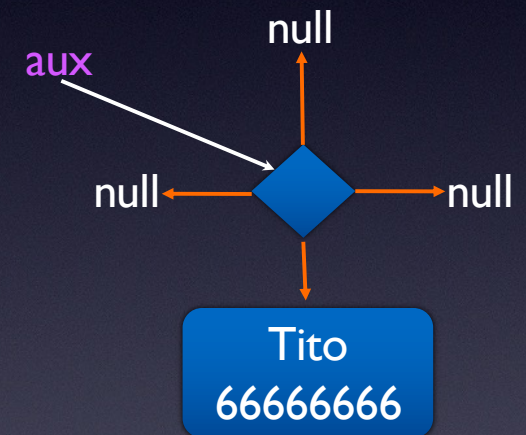


boolean inserir(Item obj)

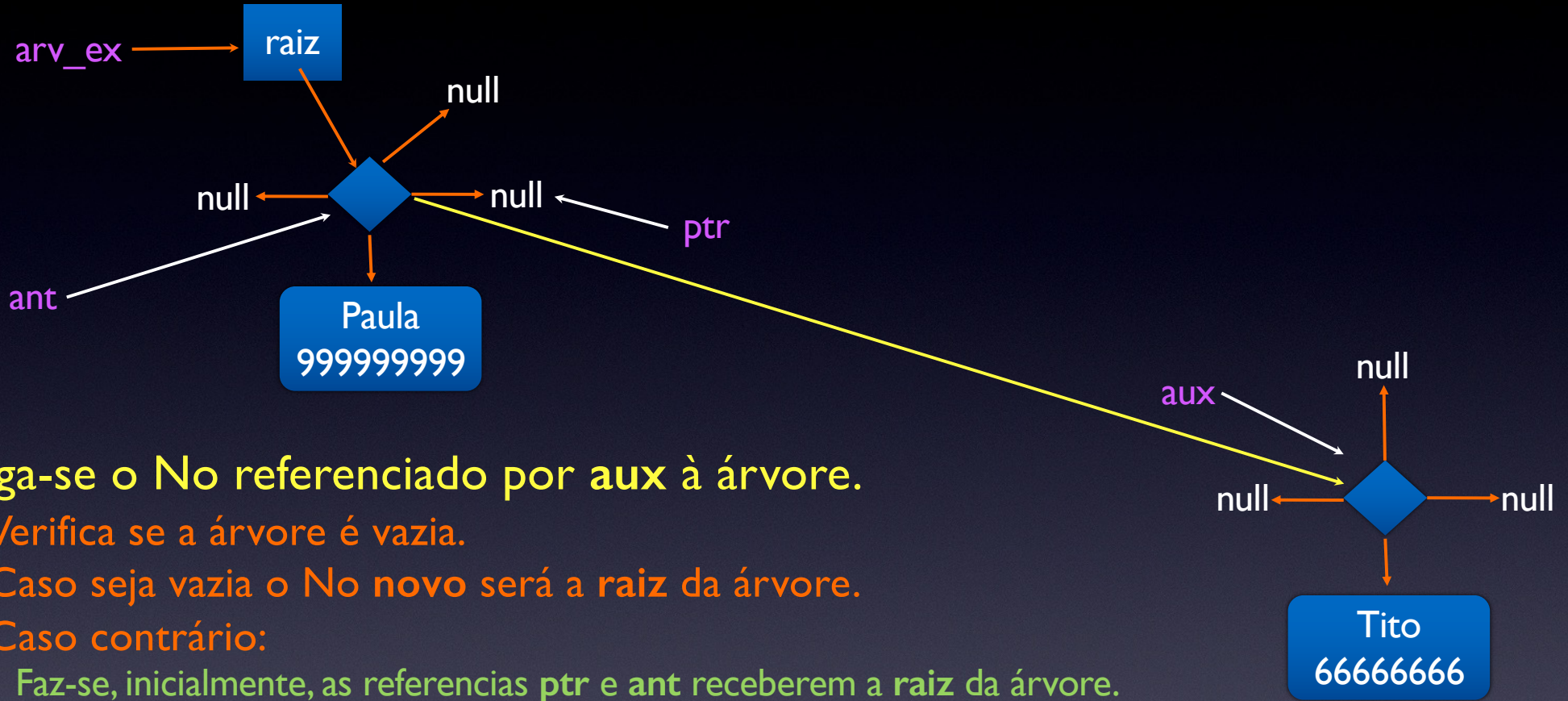


- Liga-se o No referenciado por **aux** à árvore.

- Verifica se a árvore é vazia.
- Caso seja vazia o No novo será a **raiz** da árvore.
- Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

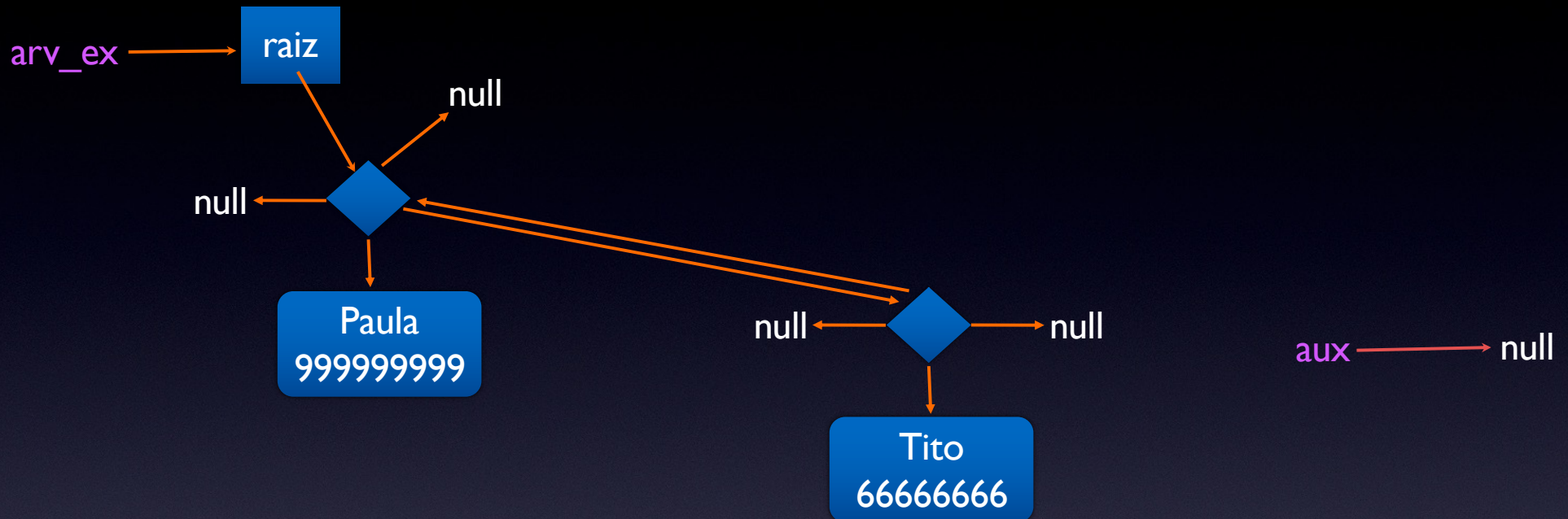


```
boolean inserir(Item obj)
```



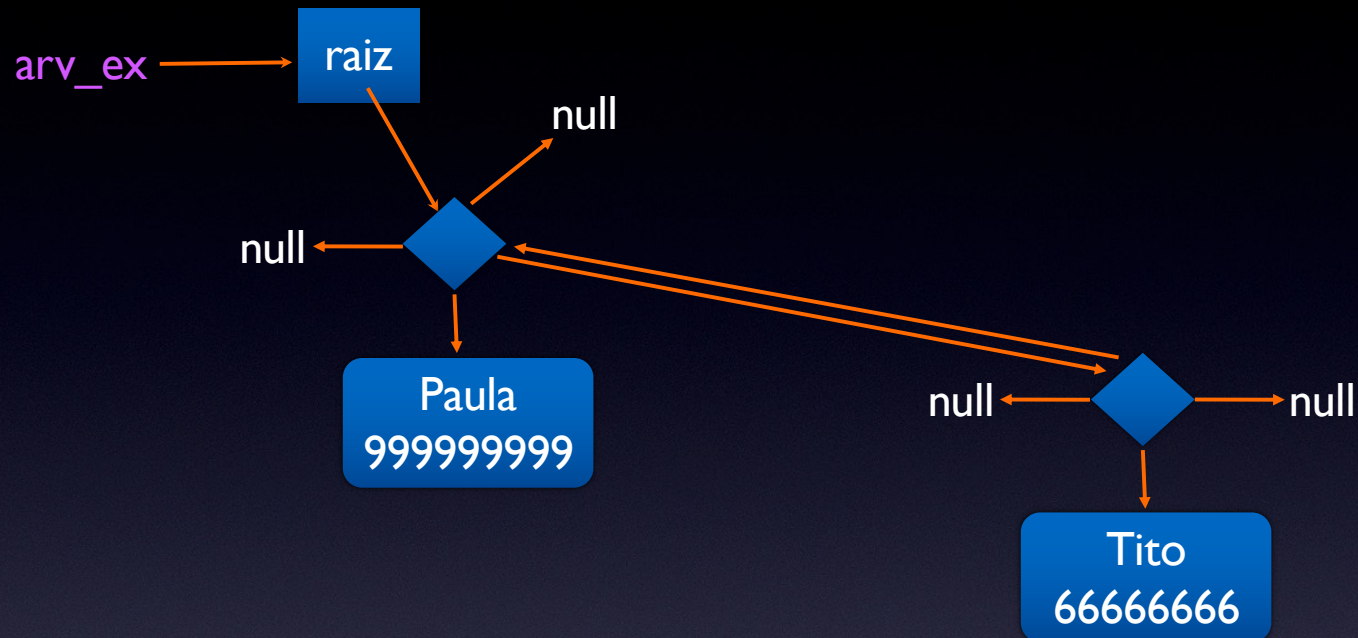
- Liga-se o No referenciado por **aux** à árvore.
 - Verifica se a árvore é vazia.
 - Caso seja vazia o No **novo** será a **raiz** da árvore.
 - Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

boolean inserir(Item obj)

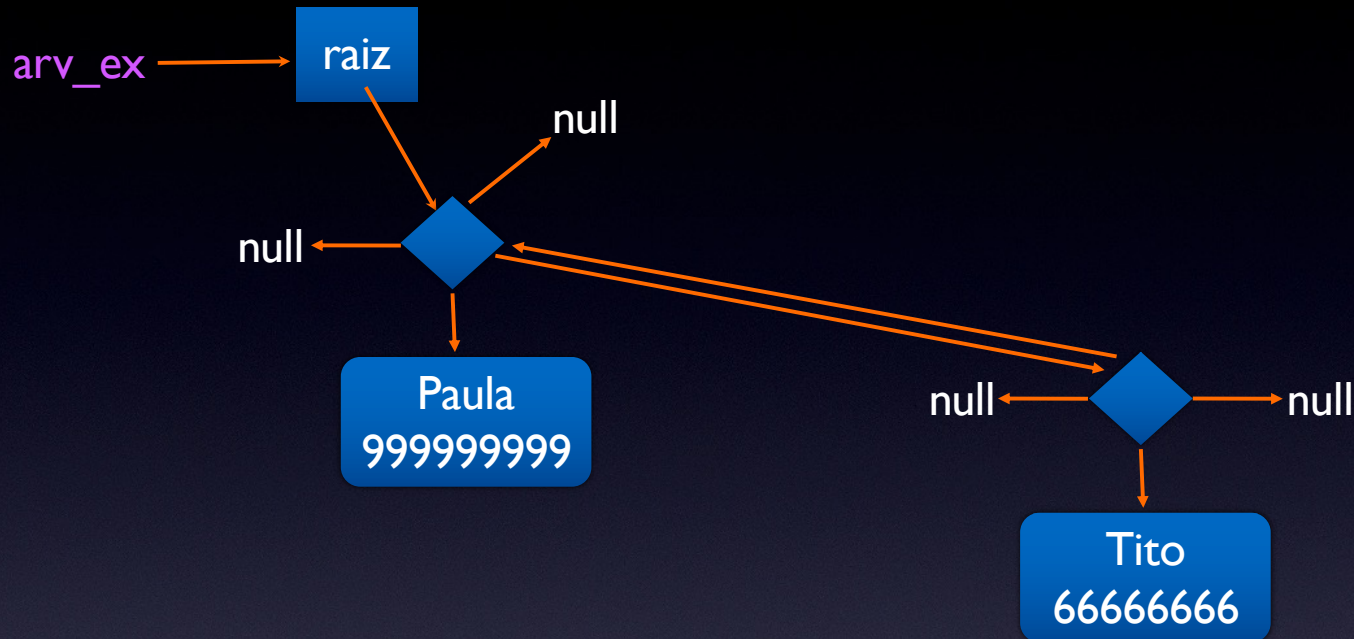


- Liga-se o No referenciado por **aux** à árvore.
 - Verifica se a árvore é vazia.
 - Caso seja vazia o No novo será a **raiz** da árvore.
 - Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

Árvore após 2 inserções



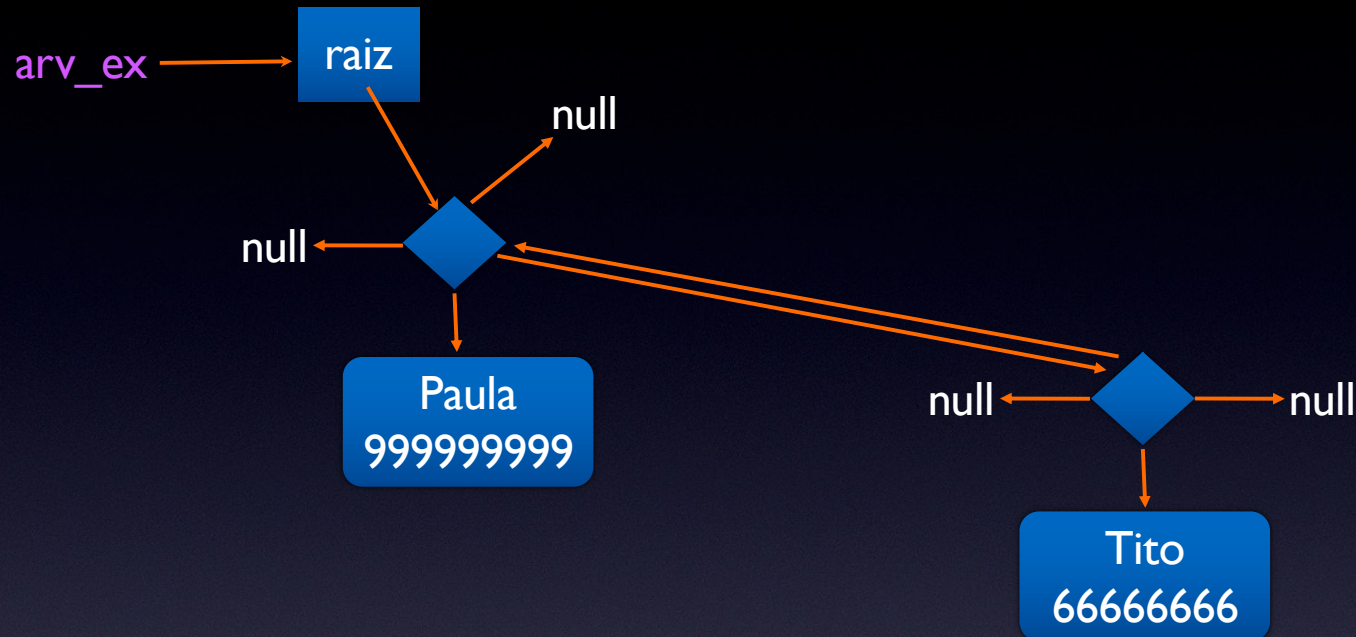
boolean inserir(Item obj)



- Na chamada do método `inserir` recebe o objeto novo do tipo `Item`



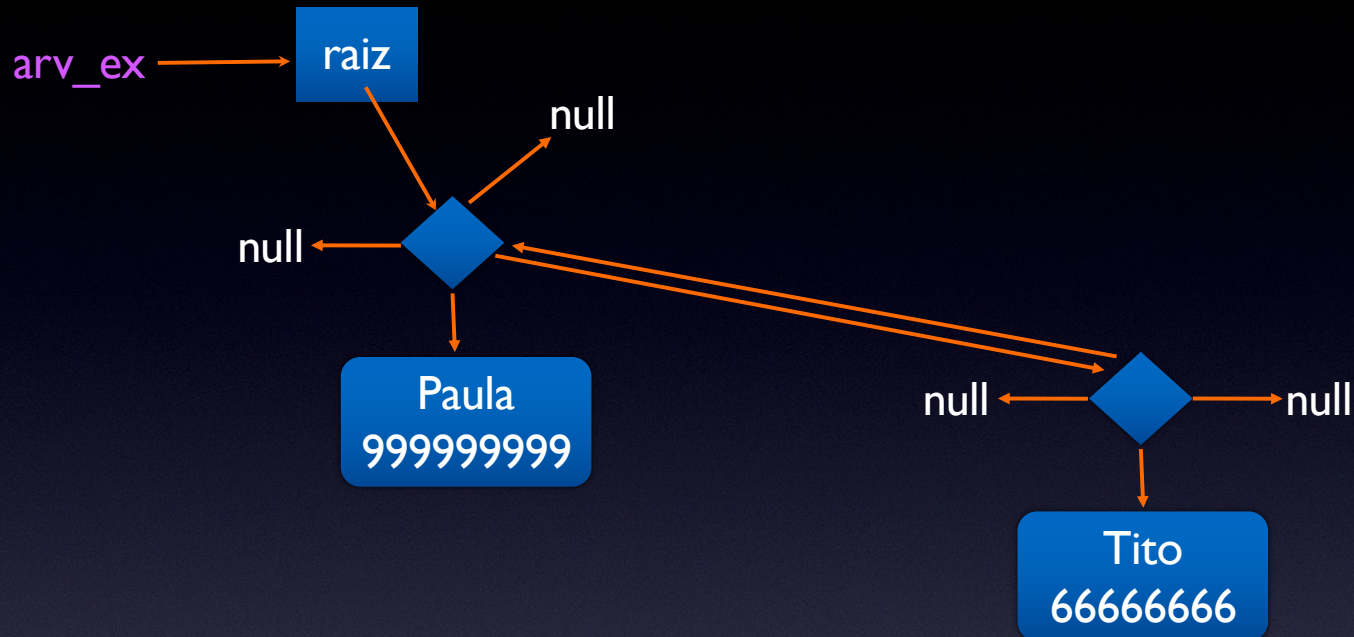
boolean inserir(Item obj)



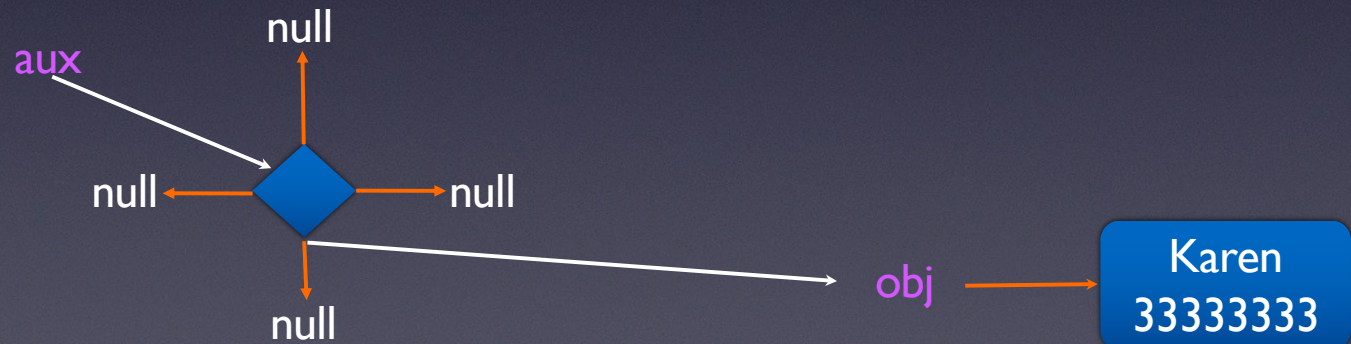
- Cria-se o No e armazena-se seu endereço em `aux` (do tipo No).



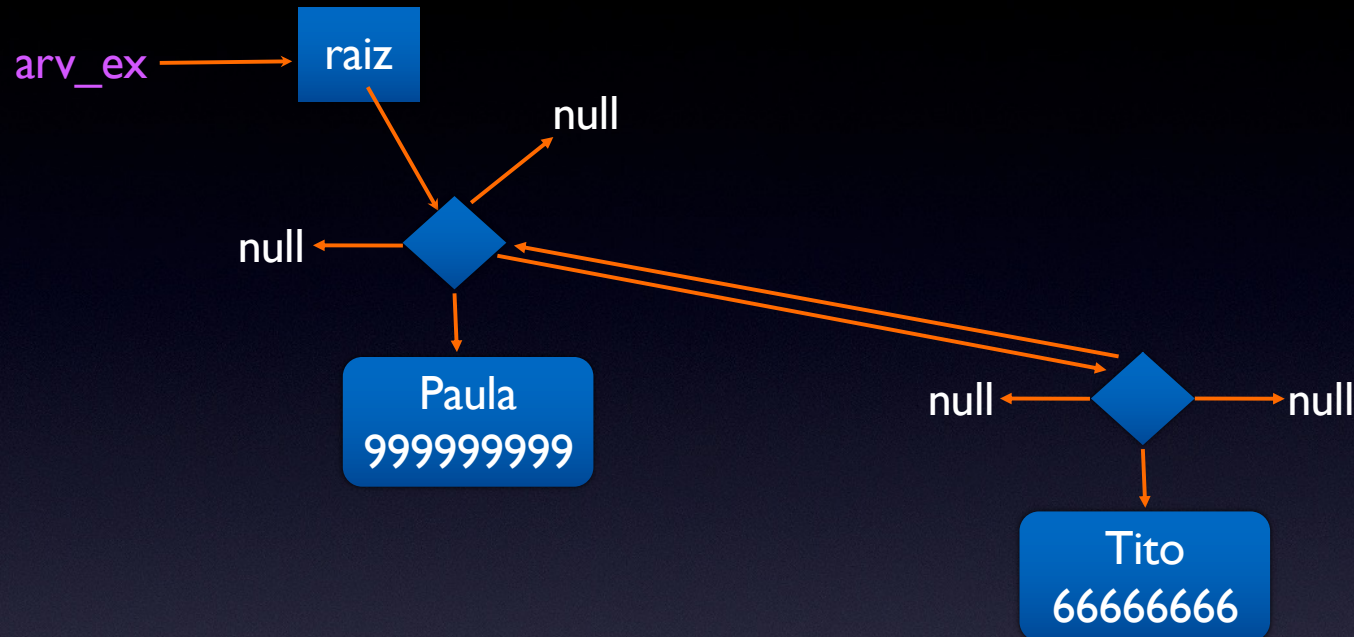
boolean inserir(Item obj)



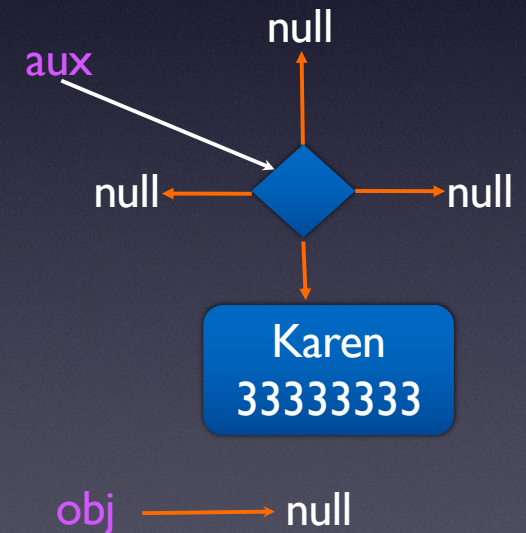
- Liga-se o Item referenciado por novo ao No referenciado por aux.



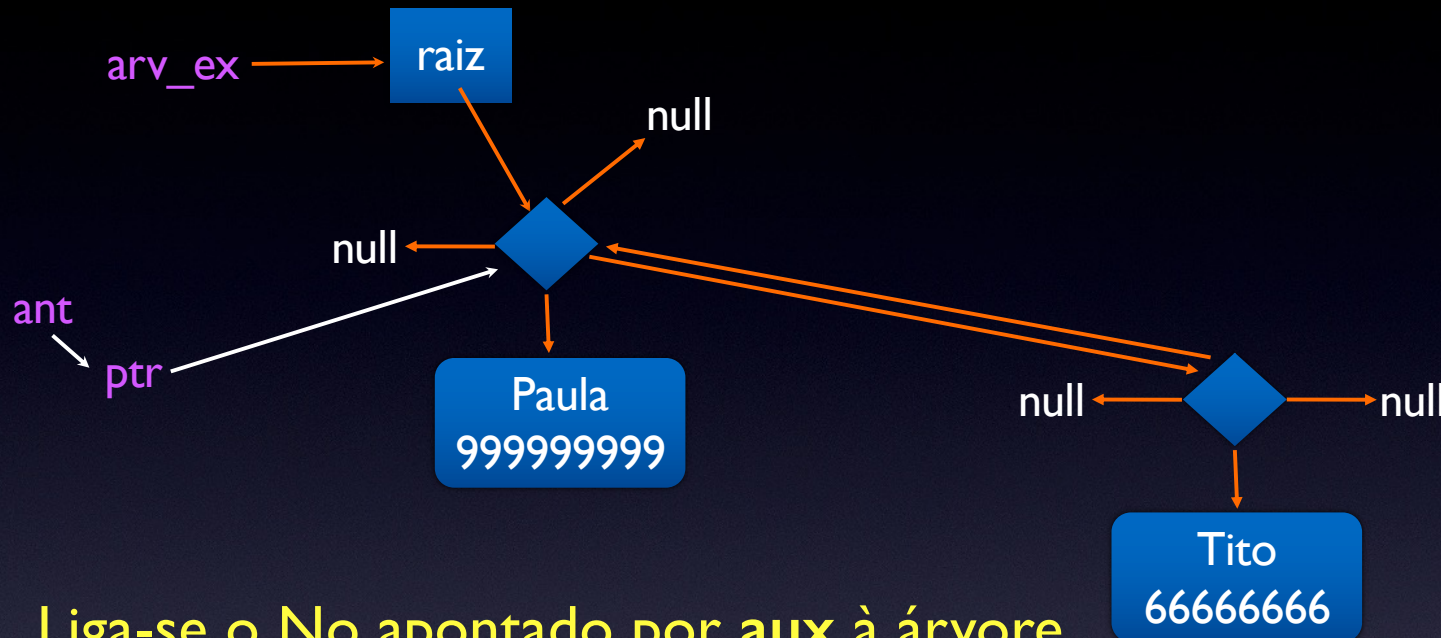
boolean inserir(Item obj)



- Liga-se o Item referenciado por novo ao No referenciado por aux.
- Faz-se novo receber 0.

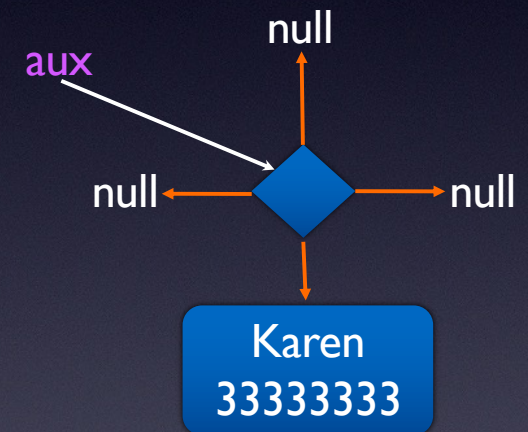


boolean inserir(Item obj)

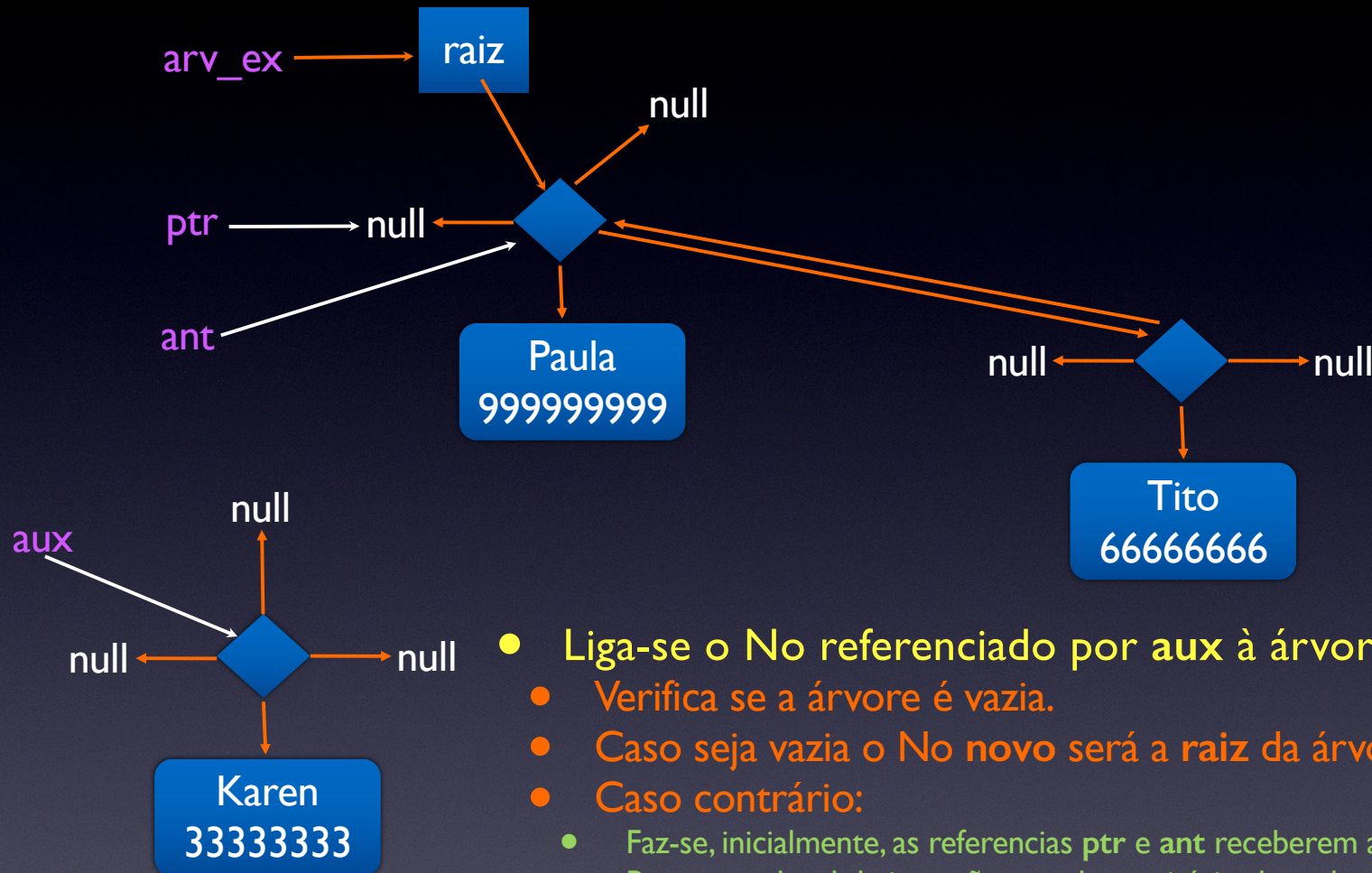


- Liga-se o No apontado por **aux** à árvore.

- Verifica se a árvore é vazia.
- Caso seja vazia o No novo será a **raiz** da árvore.
- Caso contrário:
 - Faz-se, inicialmente, os ponteiros **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com o ponteiro **ptr**.
 - Usa-se o ponteiro **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

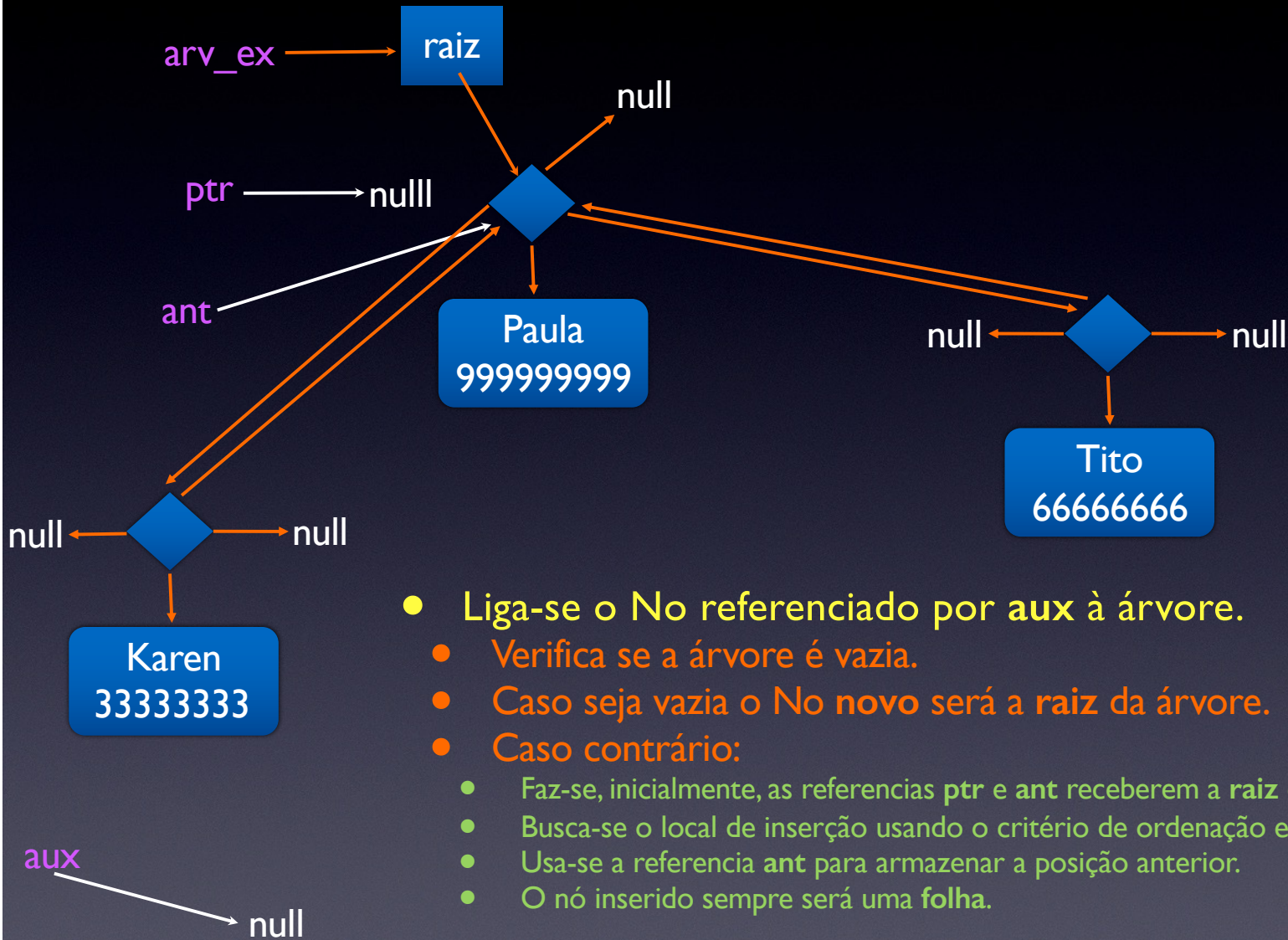


boolean inserir(Item obj)



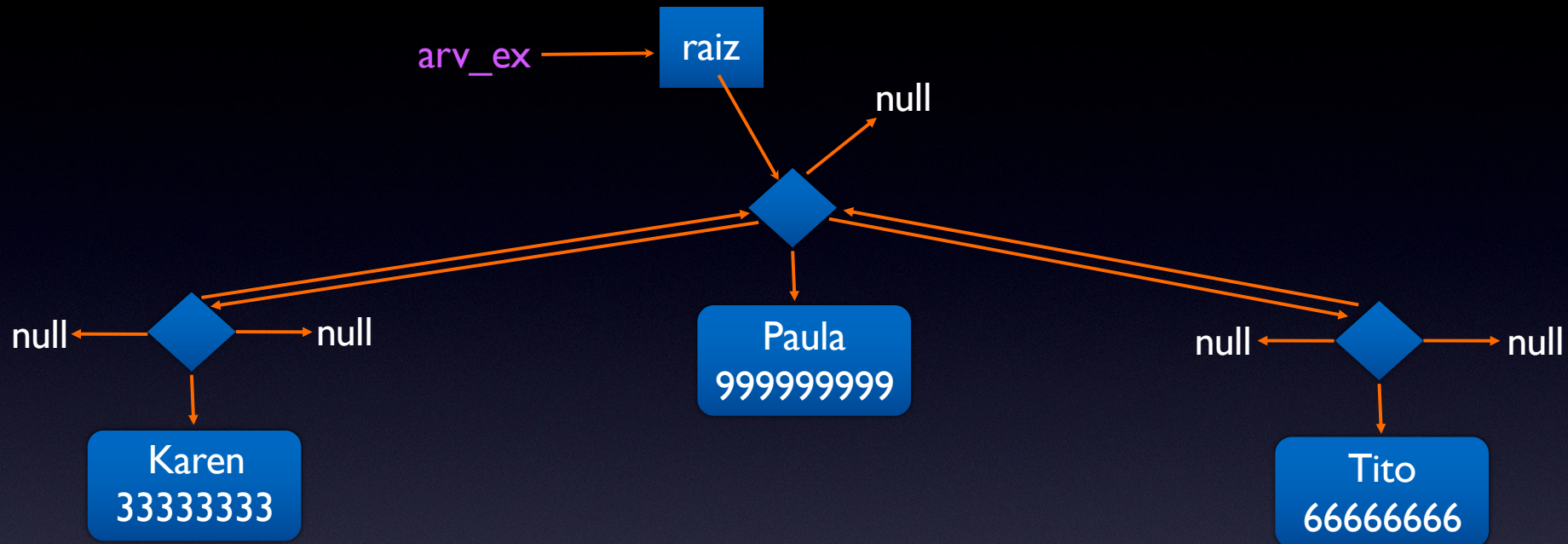
- Liga-se o No referenciado por `aux` à árvore.
- Verifica se a árvore é vazia.
- Caso seja vazia o No novo será a `raiz` da árvore.
- Caso contrário:
 - Faz-se, inicialmente, as referencias `ptr` e `ant` receberem a `raiz` da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia `ptr`.
 - Usa-se a referencia `ant` para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.


```
boolean inserir(Item obj)
```



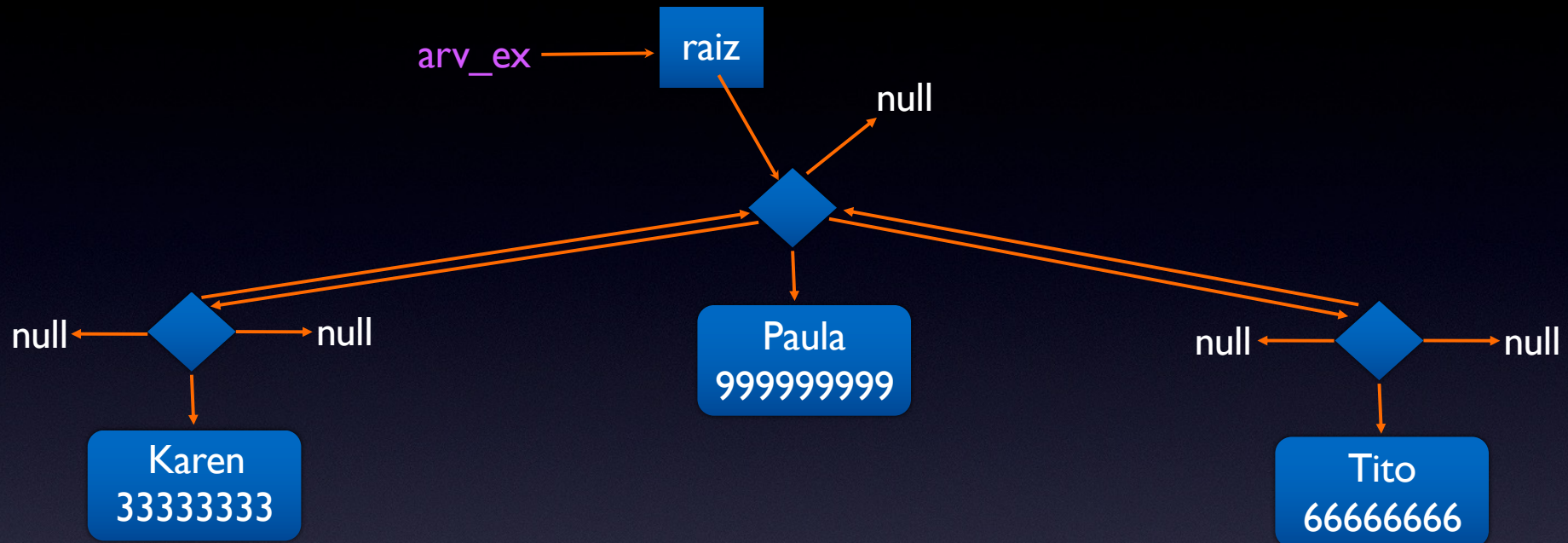
- Liga-se o No referenciado por **aux** à árvore.
 - Verifica se a árvore é vazia.
 - Caso seja vazia o No novo será a **raiz** da árvore.
 - Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

boolean inserir(Item obj)



- Liga-se o No referenciado por **aux** à árvore.
 - Verifica se a árvore é vazia.
 - Caso seja vazia o No novo será a **raiz** da árvore.
 - Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.

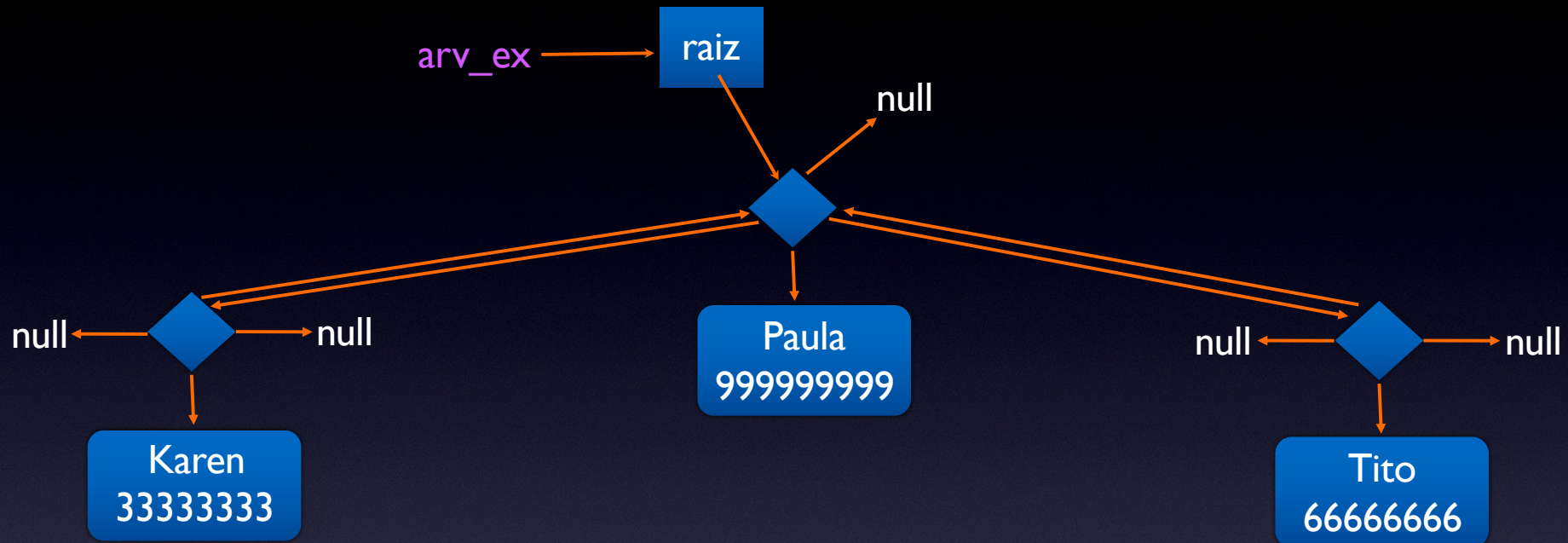
Árvore após 3 inserções



boolean inserir(Item obj)

```
public boolean inserir(Item obj) {  
    // cria-se um novo nó  
    No aux = new No(obj);  
    // verifica-se a árvore está vazia e caso afirmativo faz do nó aux a raiz da árvore  
    if (vazia()) { raiz = aux; return true; }  
    // encontra o local de inserção que é sempre numa folha  
    No ptr = raiz;  
    No ant = raiz;  
    while ( ptr != null ){  
        if (obj.getNome().compareTo(ptr.dados.getNome()) < 0) { ant = ptr; ptr = ptr.fe; }  
        else  
            if (obj.getNome().compareTo(ptr.dados.getNome()) > 0) { ant = ptr; ptr = ptr.fd;  
    }  
        else return false; // Insucesso --> item já está na árvore  
    }  
    // faz o nó referenciado por ant o pai do nó aux  
    aux.pai = ant;  
    // verifica-se é filho a esquerda ou a direita  
    if (obj.getNome().compareTo(ant.dados.getNome()) < 0) { ant.fe = aux; }  
    else { ant.fd = aux; }  
    return true; // Sucesso  
}
```

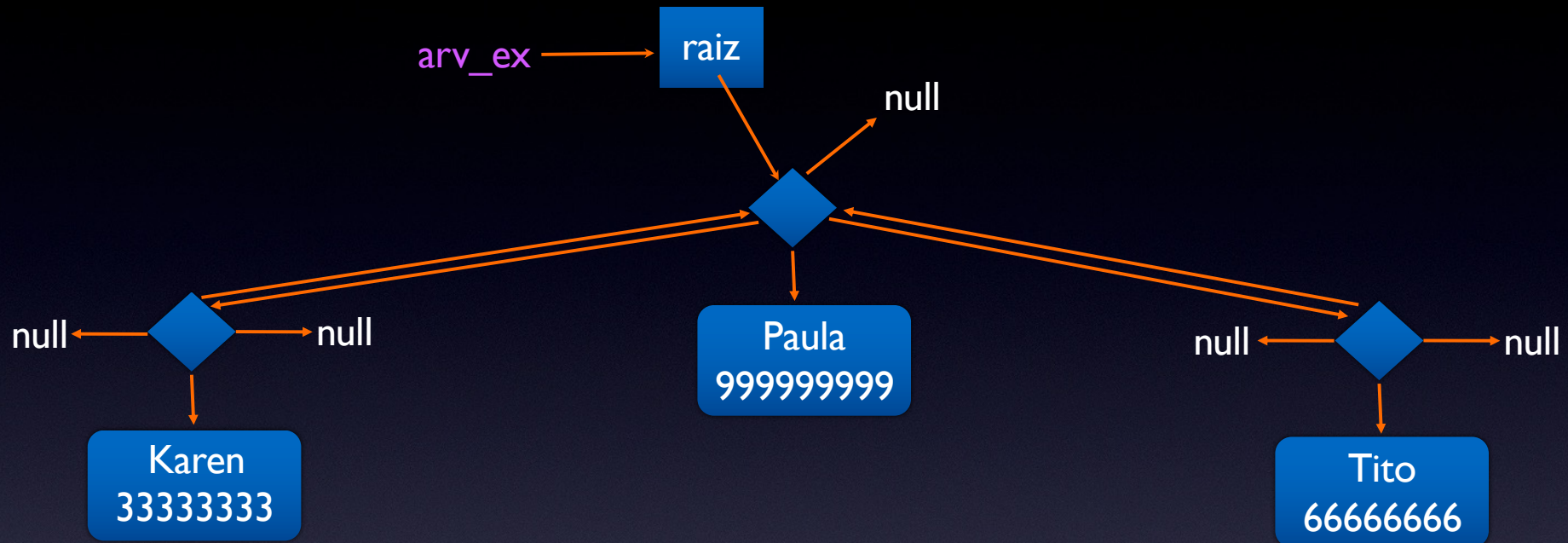

boolean inserir(Item obj)



- Na chamada do método `inserir` recebe o objeto novo do tipo `Item`



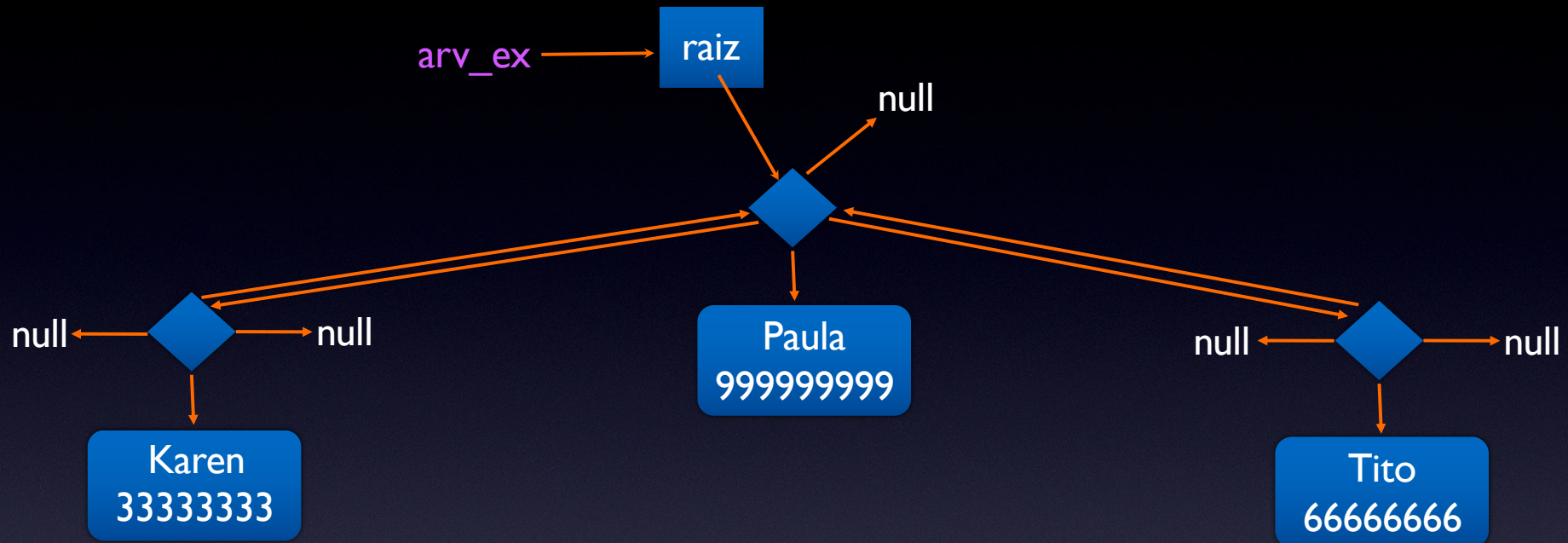
boolean inserir(Item obj)



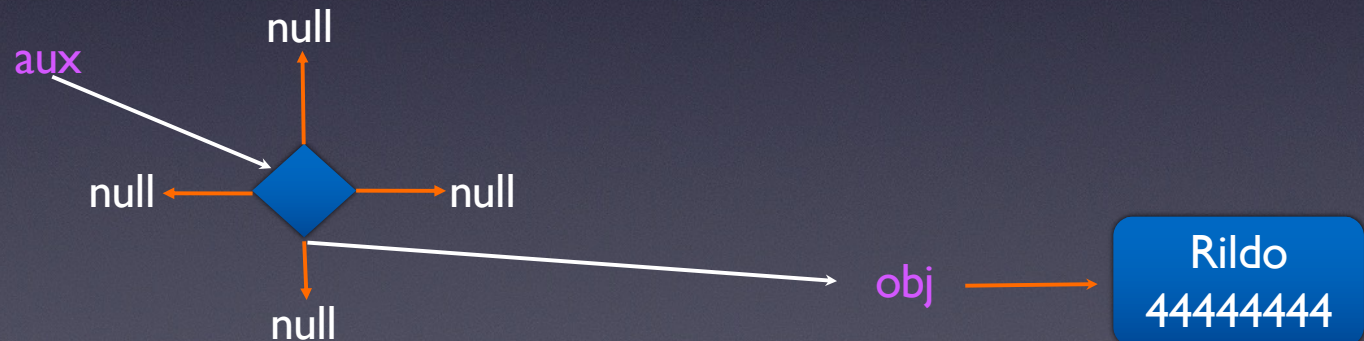
- Cria-se o No e armazena-se seu endereço em `aux` (do tipo No).



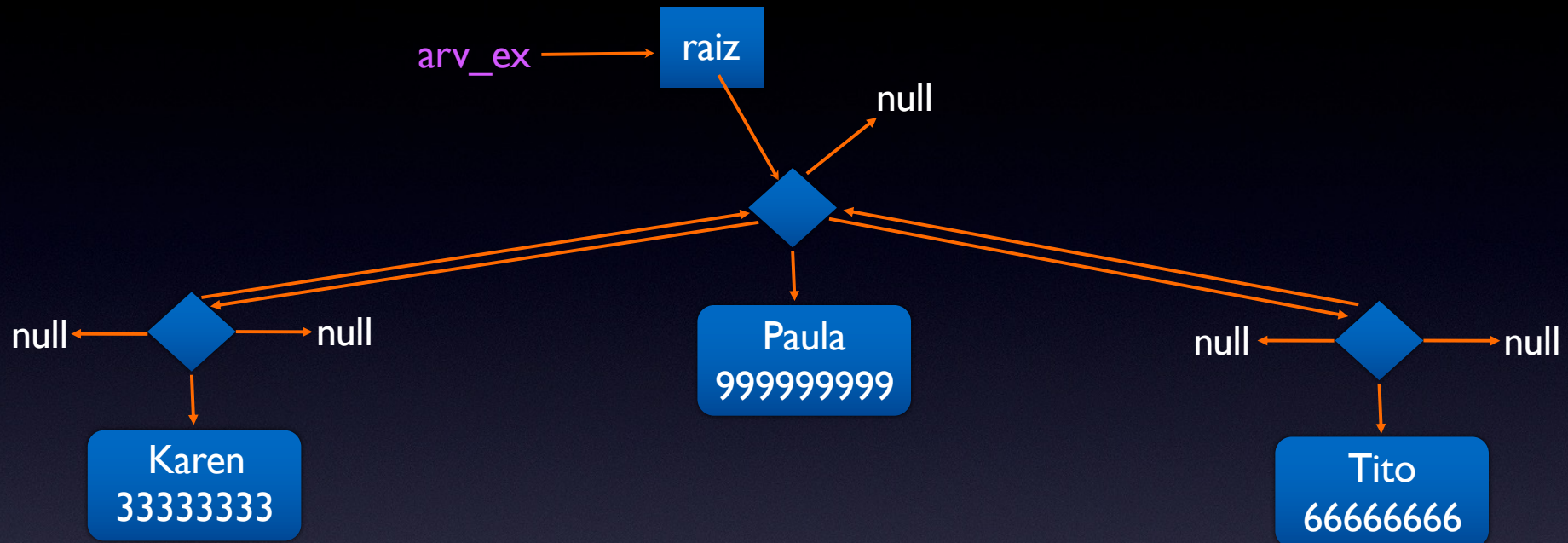
boolean inserir(Item obj)



- Liga-se o Item referenciado por novo ao No referenciado por aux.

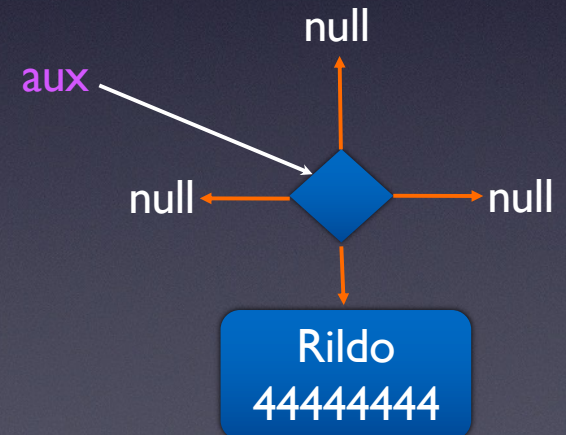


boolean inserir(Item obj)

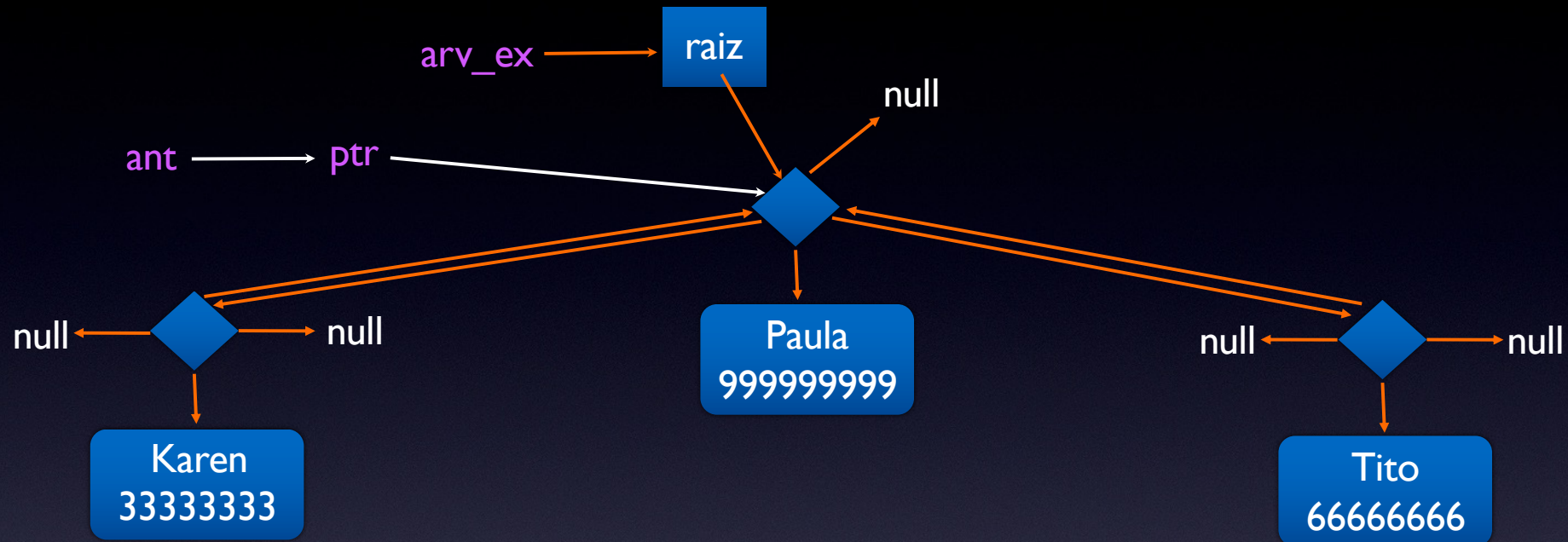


- Liga-se o Item referenciado por novo ao No referenciado por aux.
- Faz-se novo receber 0.

obj → null

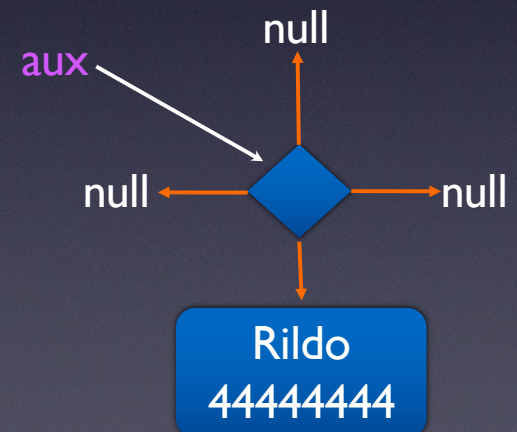


boolean inserir(Item obj)

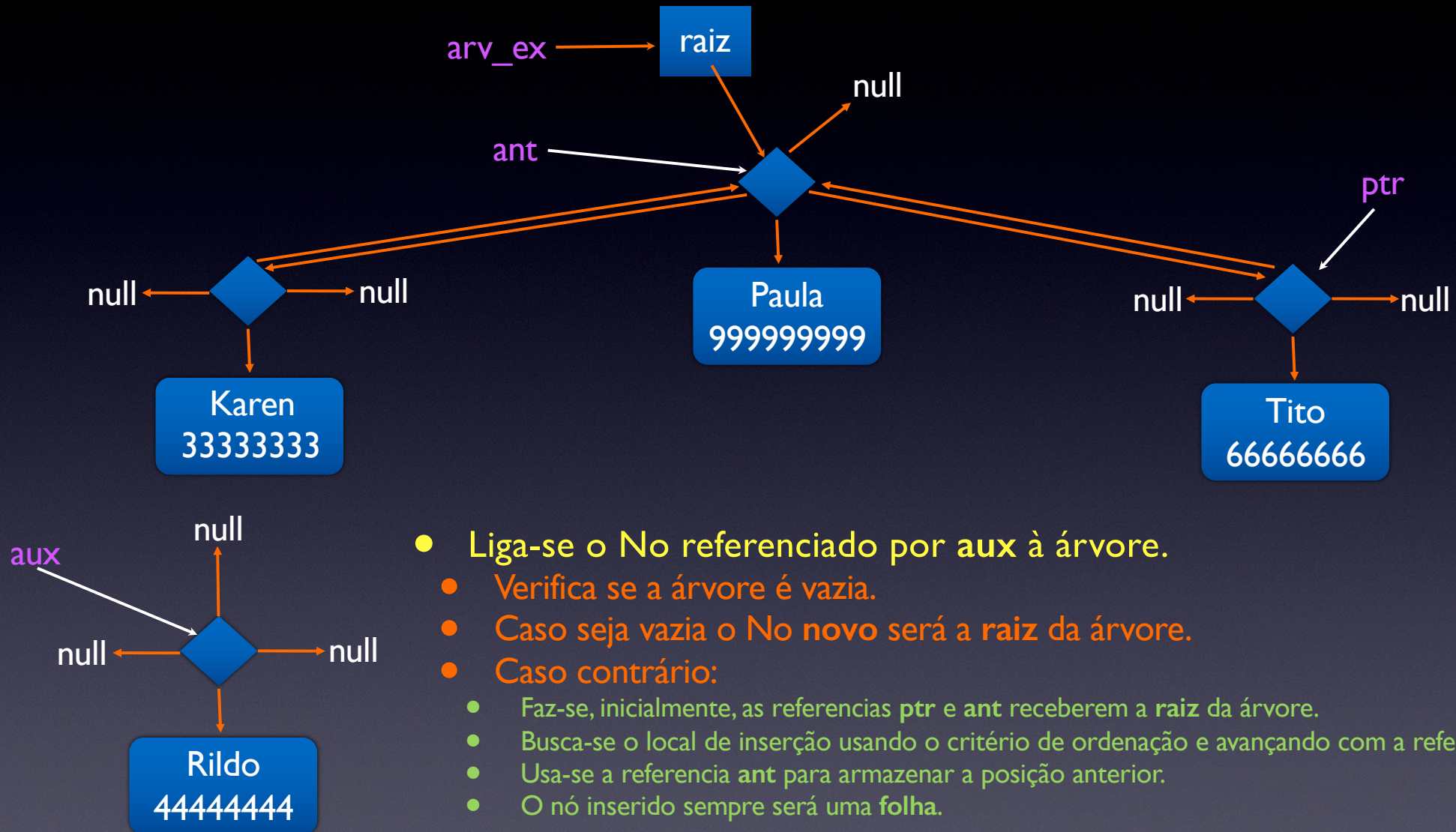


- Liga-se o No apontado por aux à árvore.

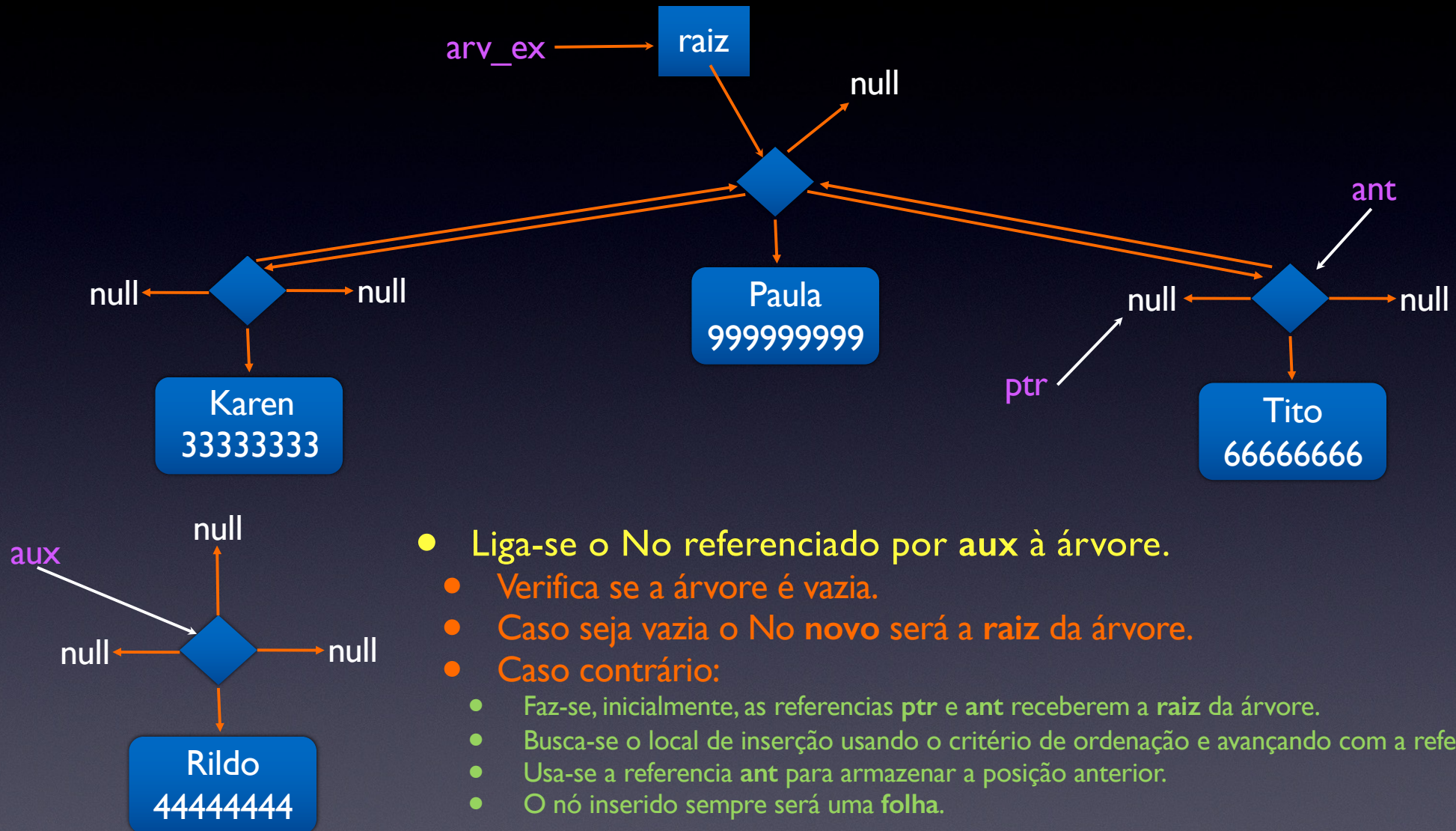
- Verifica se a árvore é vazia.
- Caso seja vazia o No novo será a raiz da árvore.
- Caso contrário:
 - Faz-se, inicialmente, os ponteiros ptr e ant receberem a raiz da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com o ponteiro ptr.
 - Usa-se o ponteiro ant para armazenar a posição anterior.
 - O nó inserido sempre será uma folha.



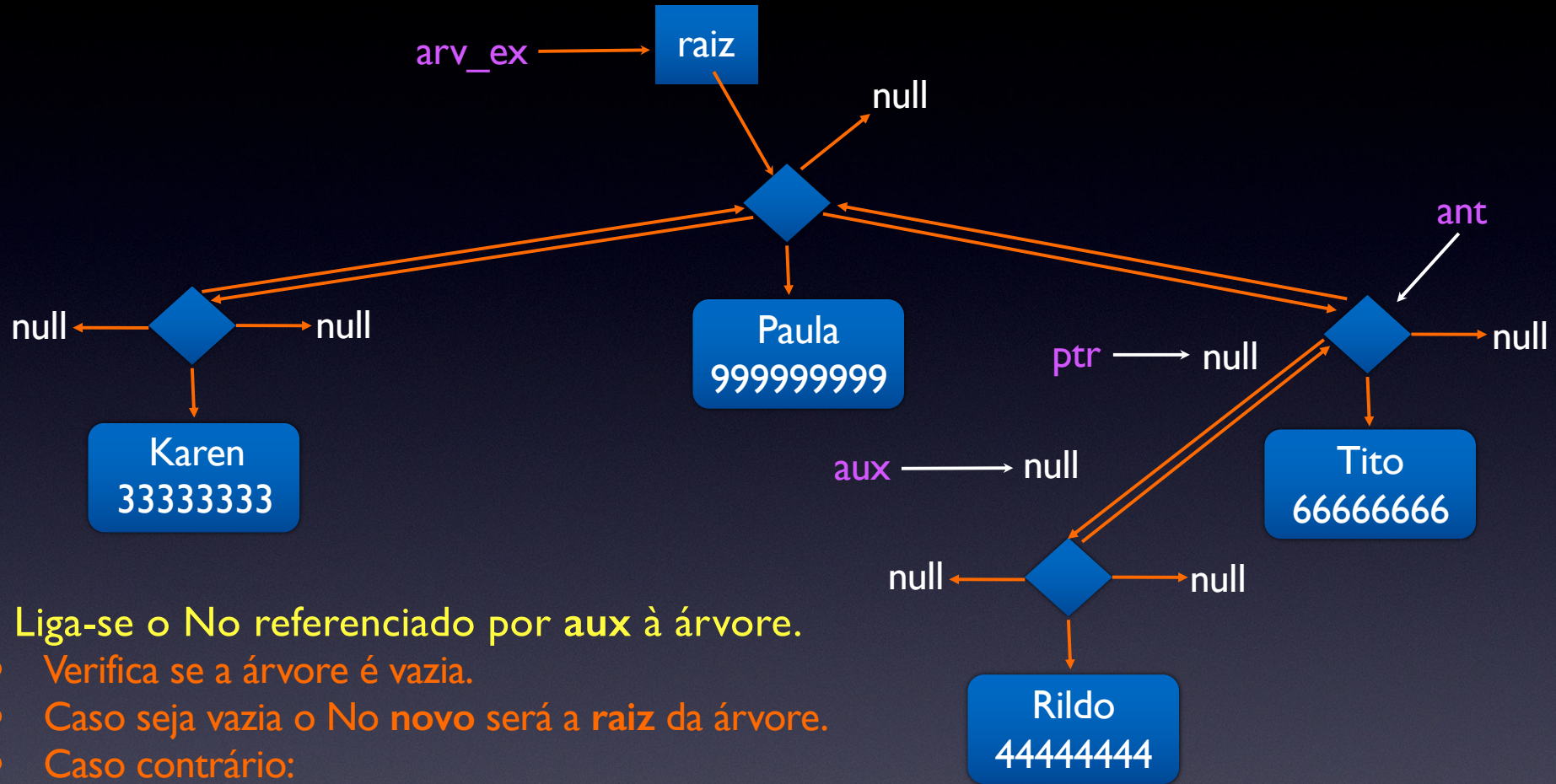
boolean inserir(Item obj)




boolean inserir(Item obj)



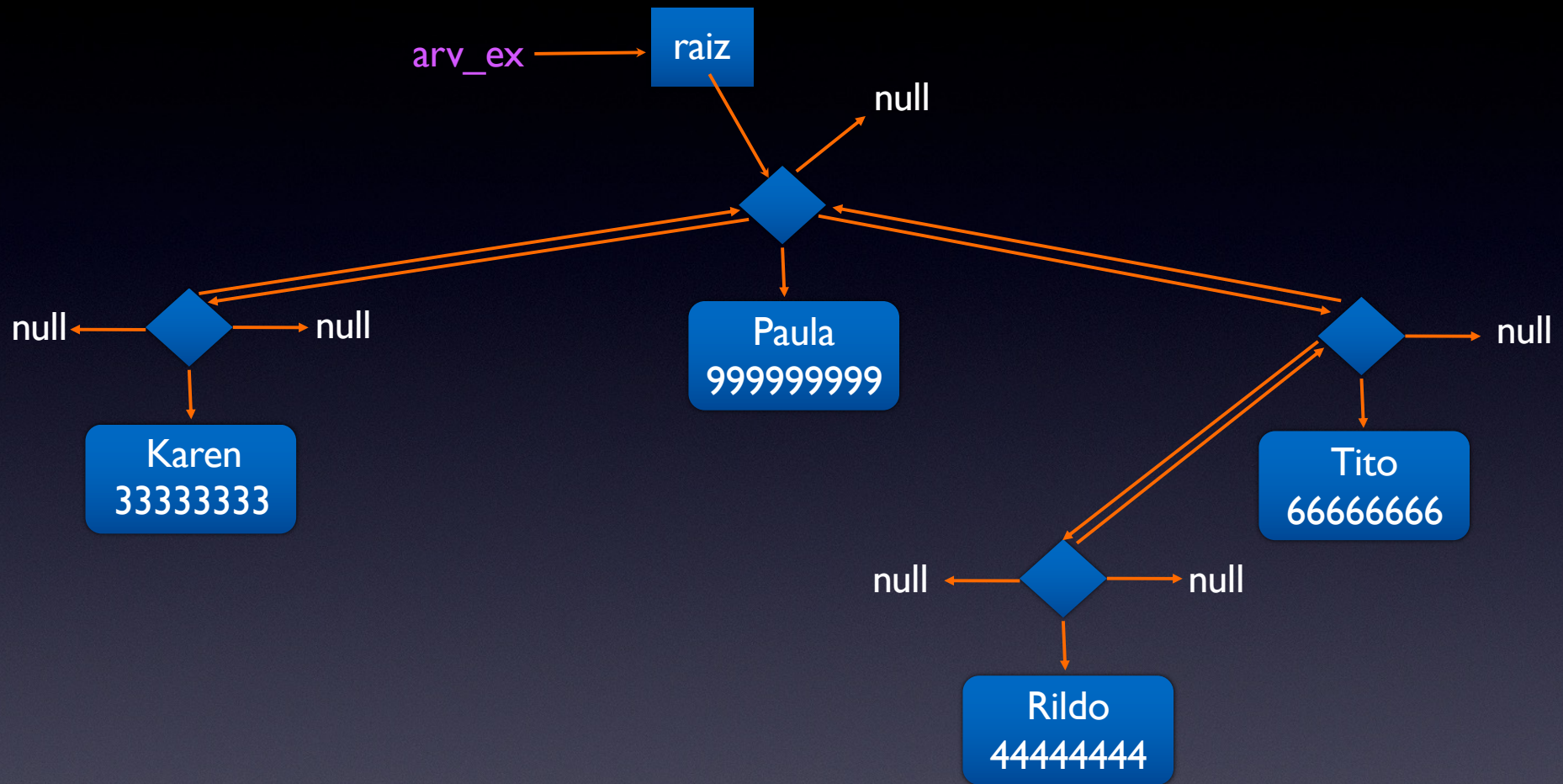
```
boolean inserir(Item obj)
```



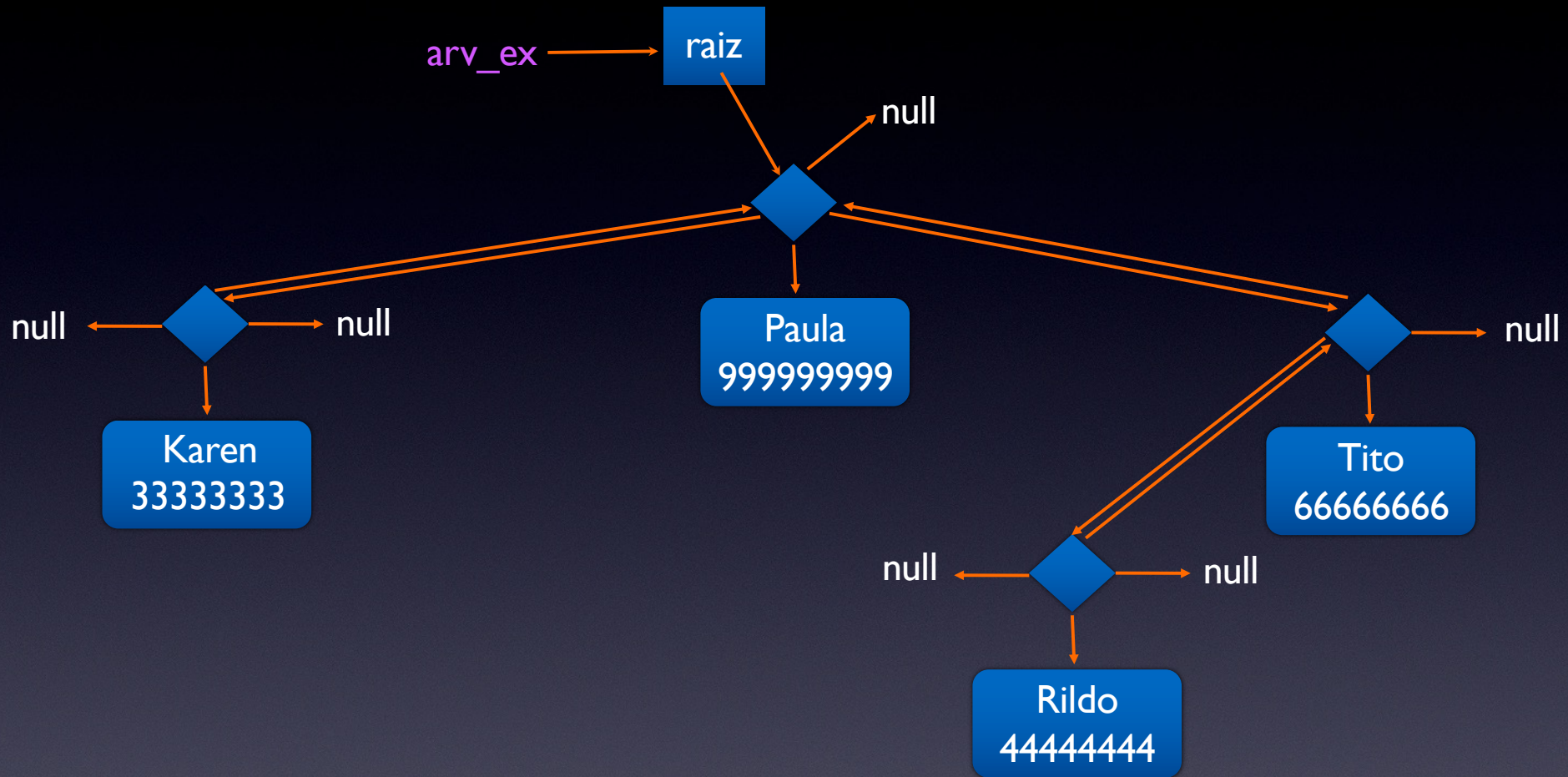
- Liga-se o No referenciado por **aux** à árvore.
 - Verifica se a árvore é vazia.
 - Caso seja vazia o No **novo** será a **raiz** da árvore.
 - Caso contrário:
 - Faz-se, inicialmente, as referencias **ptr** e **ant** receberem a **raiz** da árvore.
 - Busca-se o local de inserção usando o critério de ordenação e avançando com a referencia **ptr**.
 - Usa-se a referencia **ant** para armazenar a posição anterior.
 - O nó inserido sempre será uma **folha**.
- 
- ```
graph TD; Node["Rildo
44444444"] --> Diamond[""]; Diamond --> Title["Ligação do nó à árvore"]
```



# Árvore após 4 inserções



# boolean inserir(Item obj)

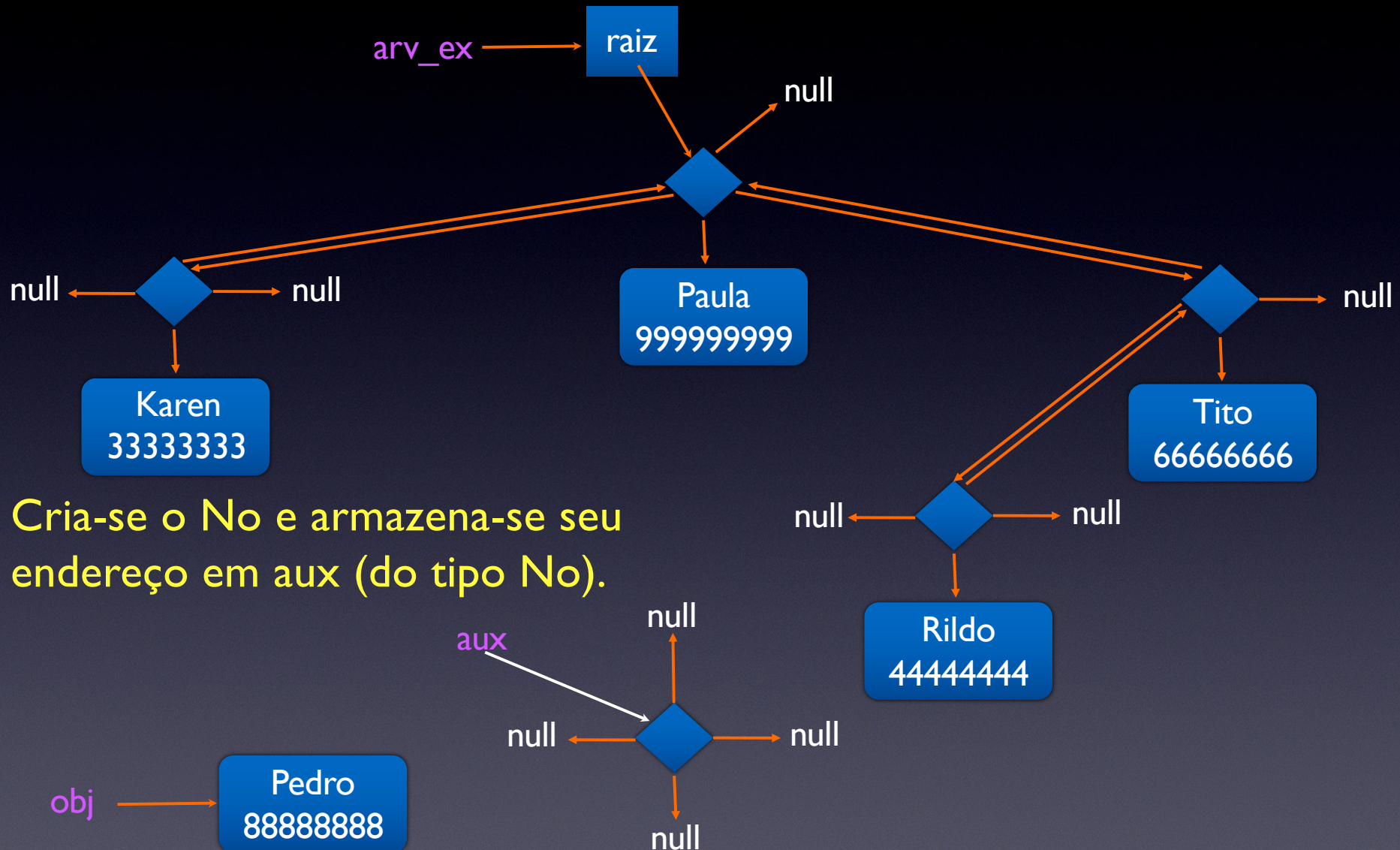


- Na chamada do método inserir recebe o objeto novo do tipo Item

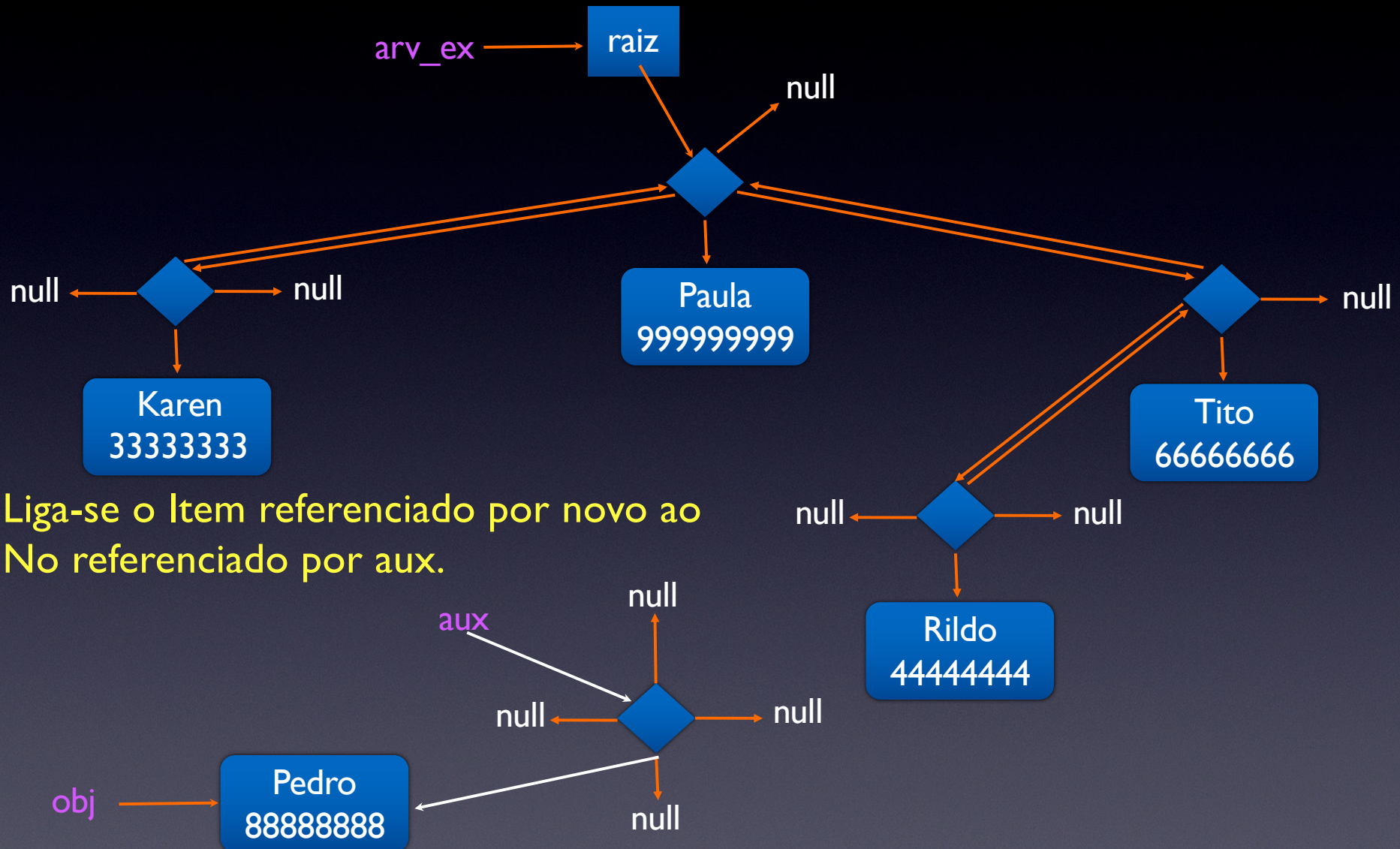




# boolean inserir(Item obj)

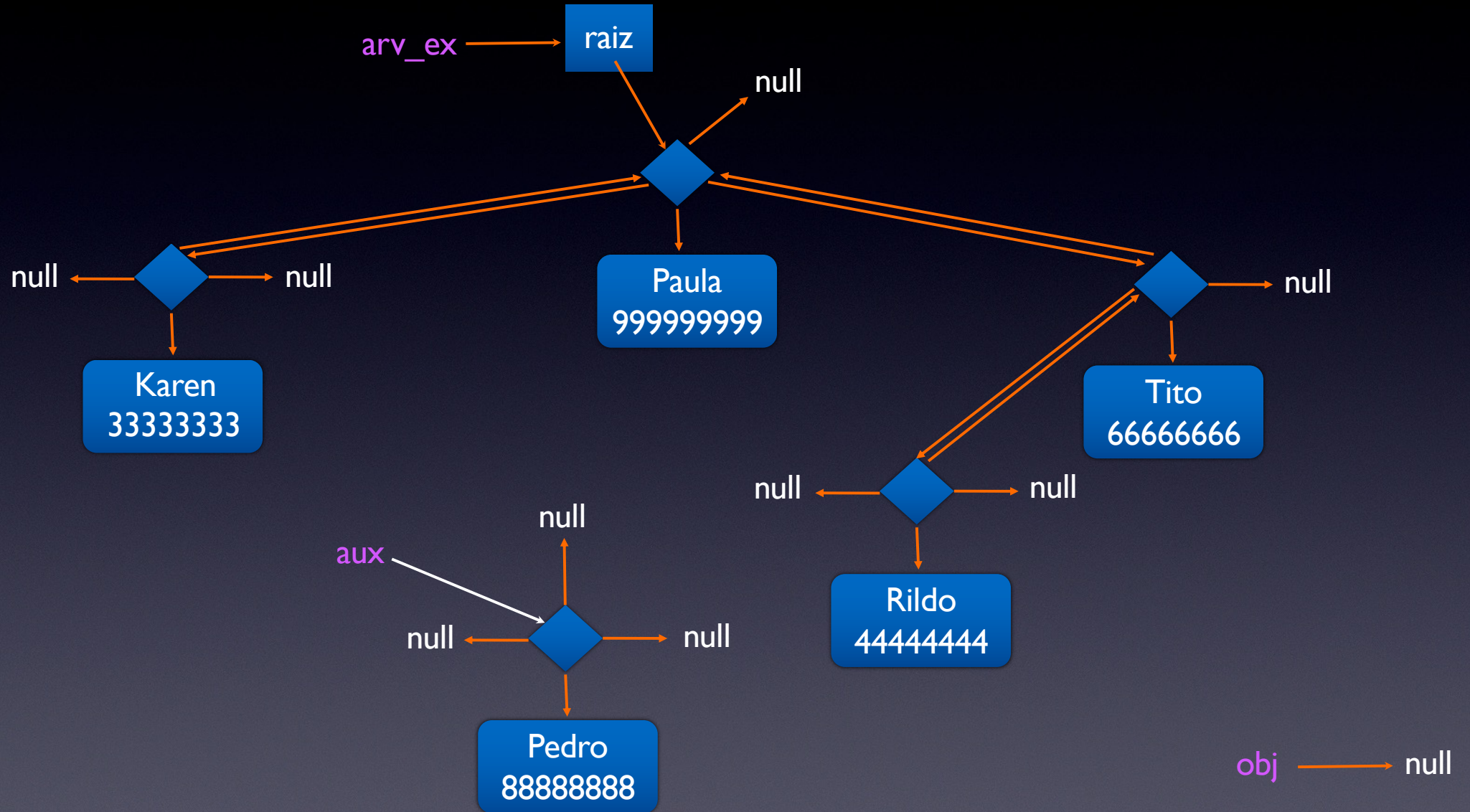


# boolean inserir(Item obj)

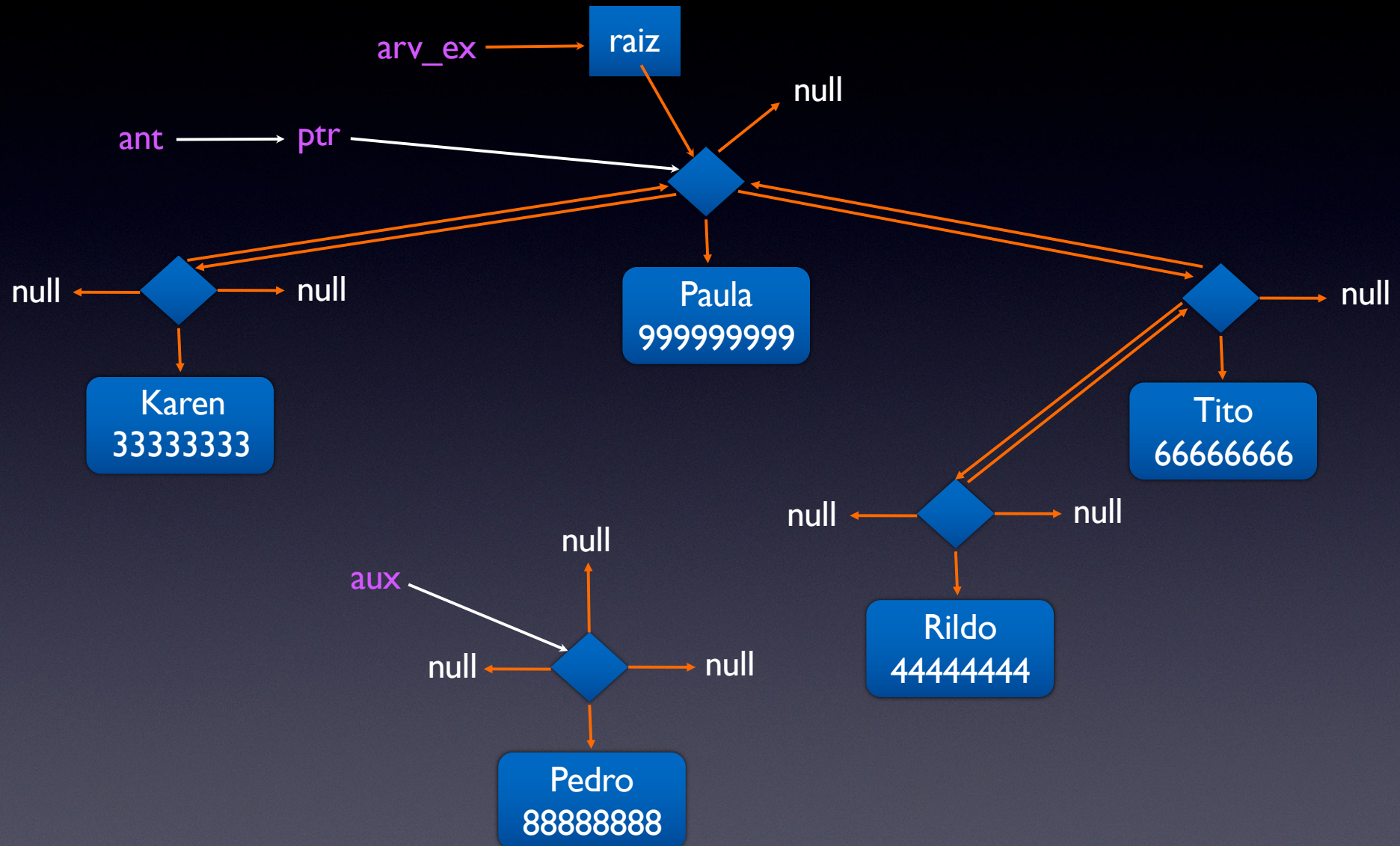




```
boolean inserir(Item obj)
```

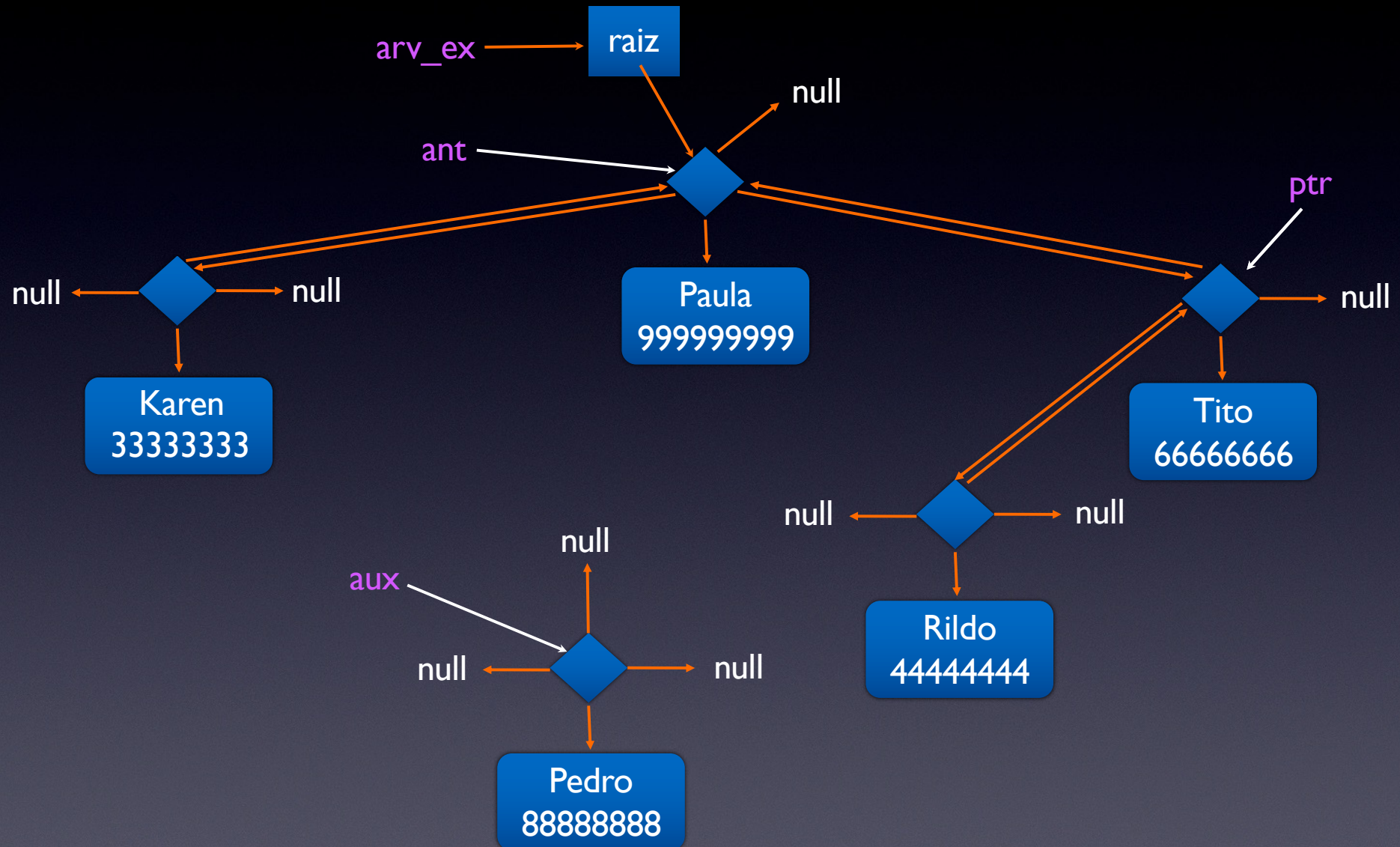


# boolean inserir(Item obj)

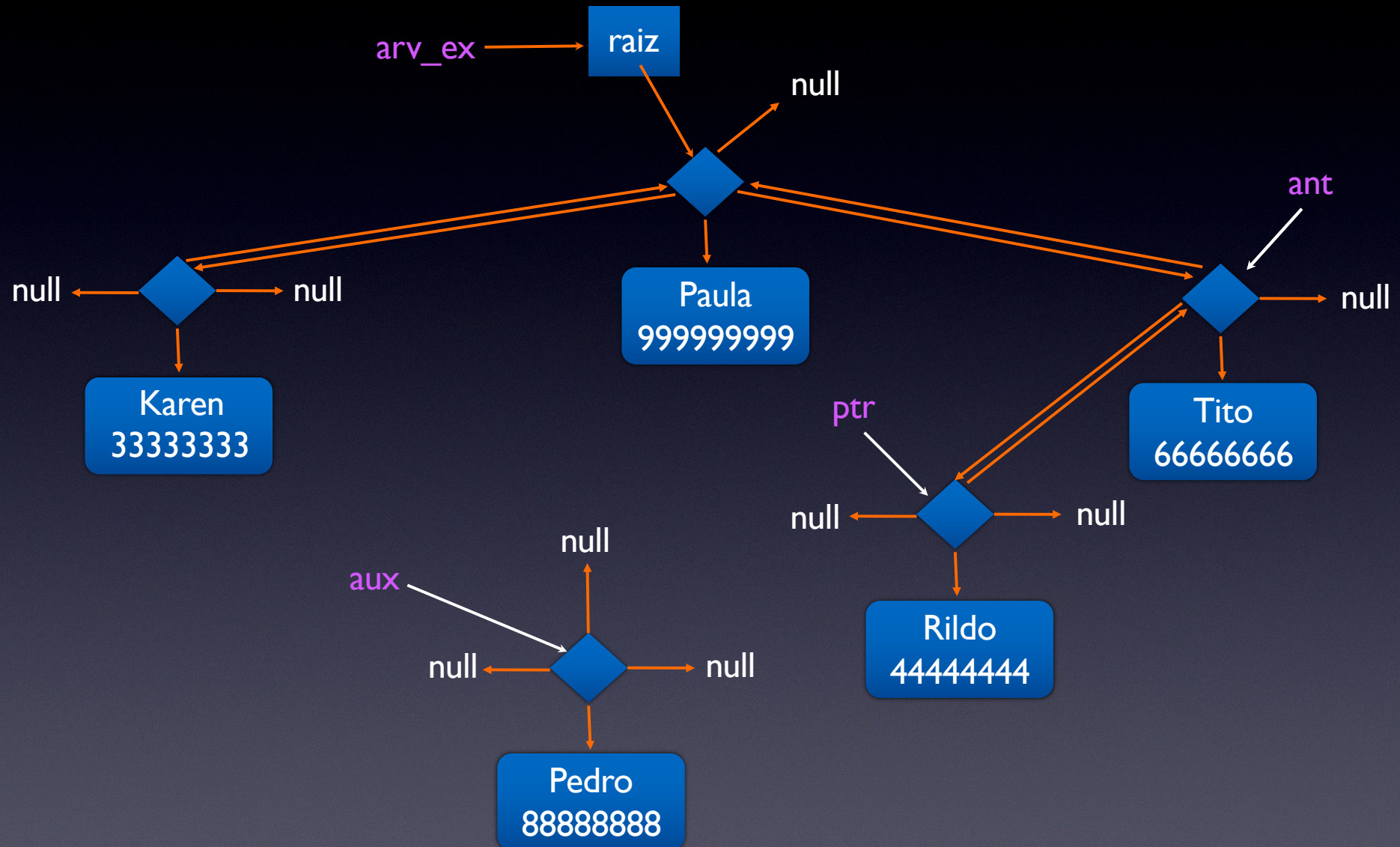




# boolean inserir(Item obj)

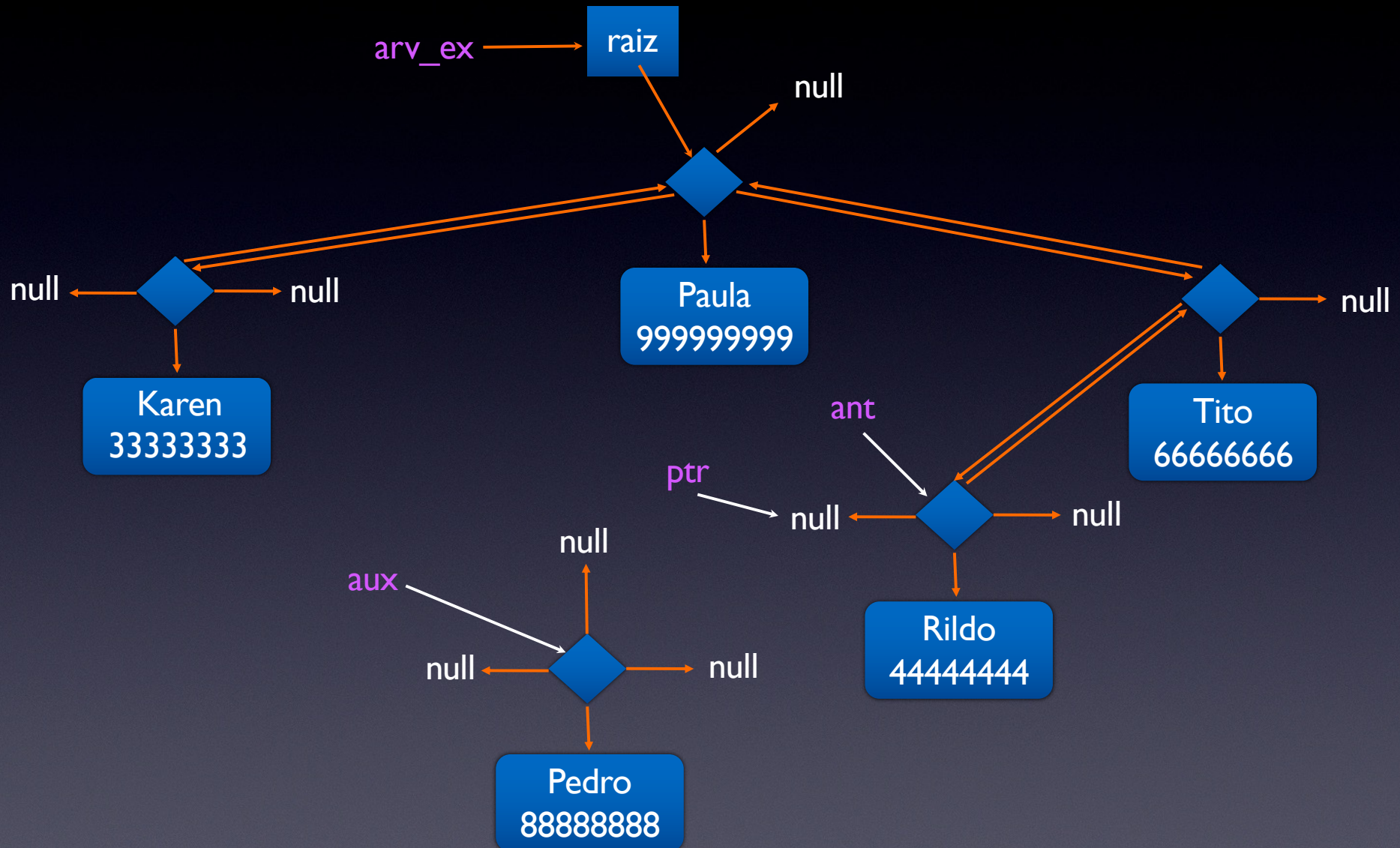


# boolean inserir(Item obj)

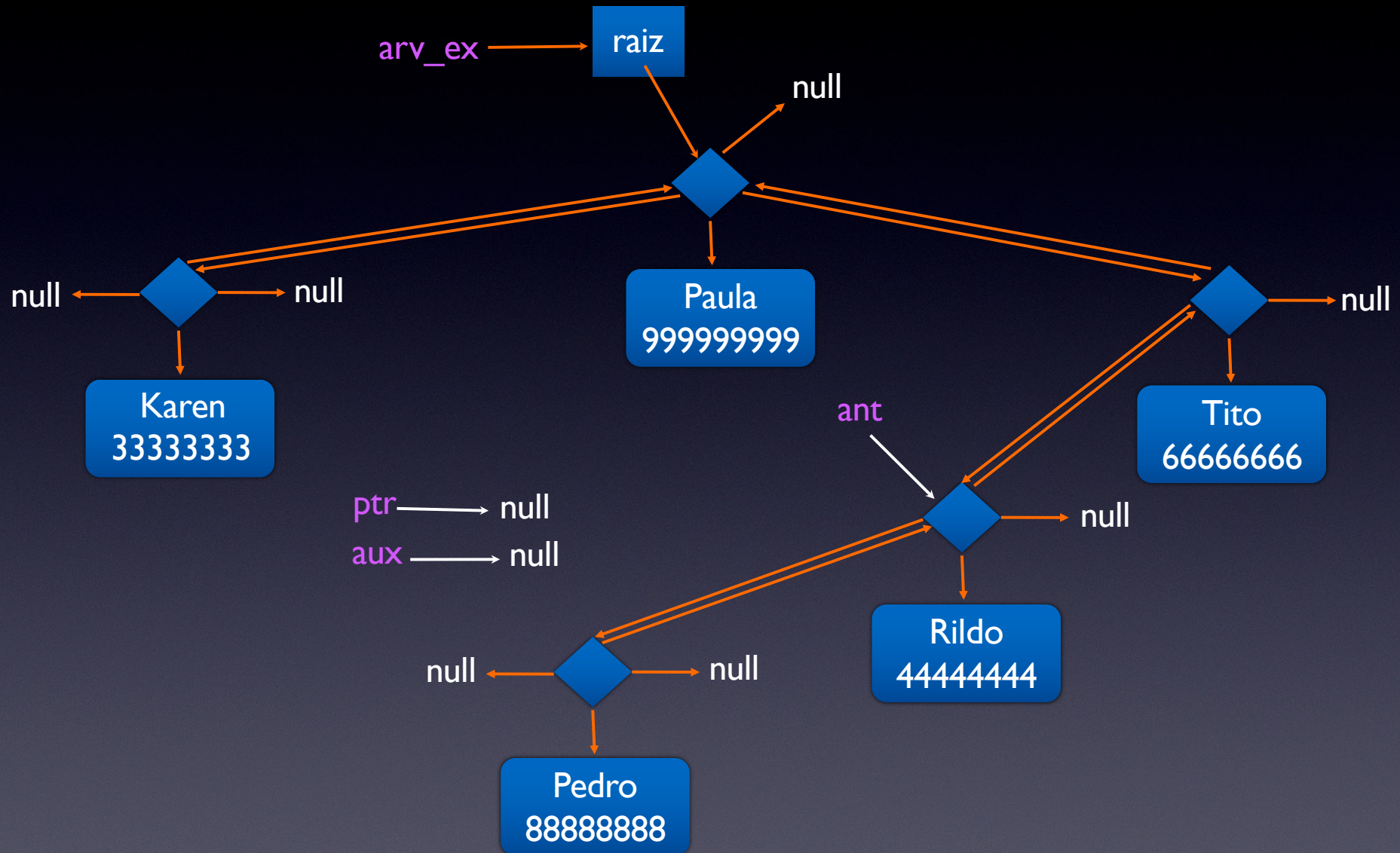




# boolean inserir(Item obj)

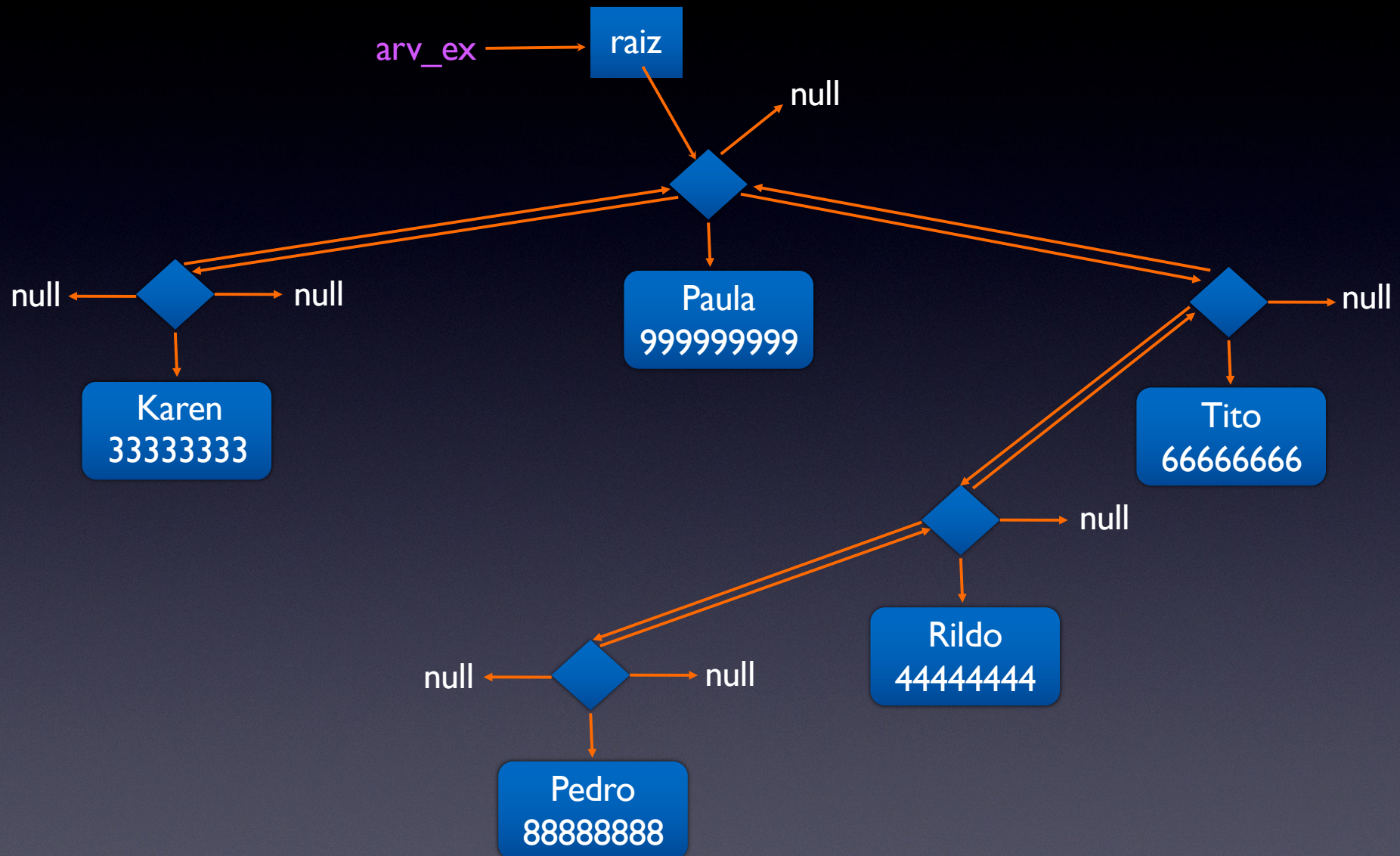


# boolean inserir(Item obj)





# Árvore após 5 inserções



# Exercício a ser entregue.

- Construa uma árvore binária de pesquisa inserindo na ordem apresentada a seguir com as seguintes informações, contendo (Nome, Telefone), sendo nome o atributo chave.
- (Mario, 12345678), (Talita, 87654321), (José, 56781234), (Laura, 98764567), (Katia, 56789876), (Luan, 23458765), (Nivaldo, 32148790), (Vando, 35467897), (Pedrina, 13467908), (Wendell, 41964704), (Higor, 41239678), (Ivan, 91827364), (Paula, 31040192).
- Com relação à árvore resultante:
  - Qual a altura da árvore?
  - Qual a altura do nó (José, 56781234) e do nó (Luan, 23458765)?
  - Qual a profundidade dos nós (Laura, 98764567) e (Wendell, 41964704)?
  - Quantos níveis tem esta árvore?
  - Se ela estivesse cheia (completa) em todos os níveis, quantos nós ela teria em cada nível? Qual é a fórmula que pode ser usada para encontrar estes valores?
  - Quantos nós ela teria no total? Qual é a fórmula que encontra este valor?



### Classe Produto

- descricao: string
- qde: int ( $\geq 0$ )
- preco: double ( $> 0$ )
- + getDescricao(): string
- + setDescricao(string): void
- + setPreco(double): bool
- + setQde(int): bool
- + getProduto(): string

### Classe TestaArvore

- obj: Abp
- + criarProduto(): Produto\*
- + menu(): void

ou

### Classe Base

- + criarProduto(): Produto
- + main(): void

### Classe Abp

- raiz: No
- qde: int
- + Abp()
- + ~Abp()
- + inserir(Produto\*): bool
- + remover(Produto\*): Produto\*
- + pesquisar(Produto\*): Produto\*
- consultar(Produto): No\*
- máximo(No\*): No\*
- mínimo(No\*): No\*
- antecessor(No\*): No\*
- sucessor(No\*): No\*
- vazia(): bool

### Classe No

- pai: No\*
- fe: No\*
- fd: No\*
- + No(Produto\*)