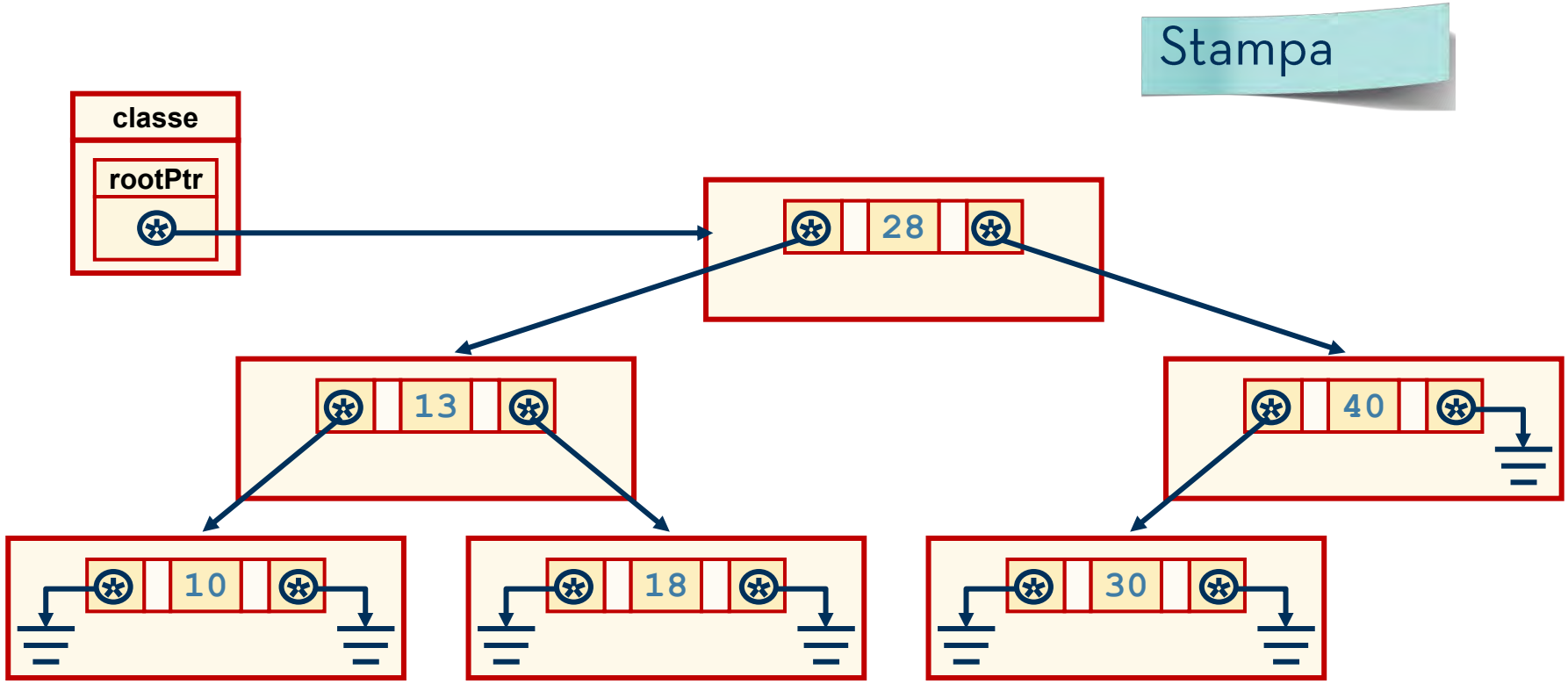
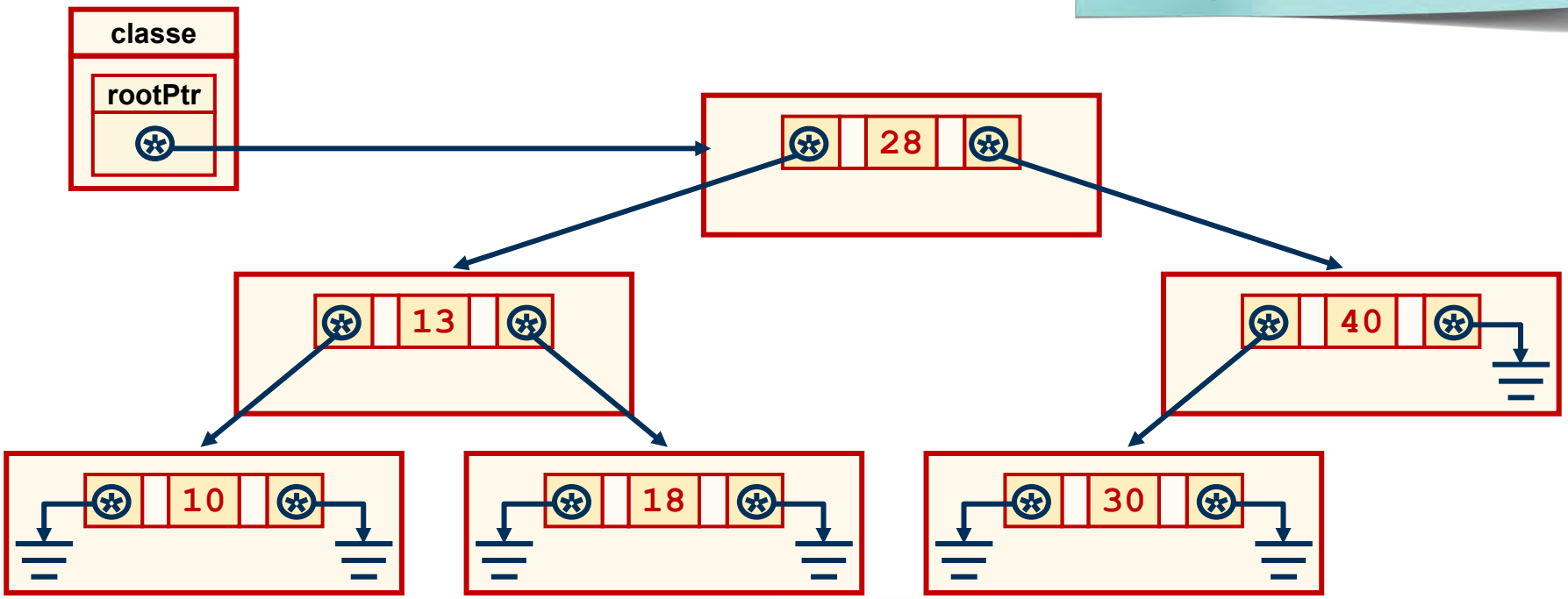
POLITECNICO DI MILANO

INFORMATICA

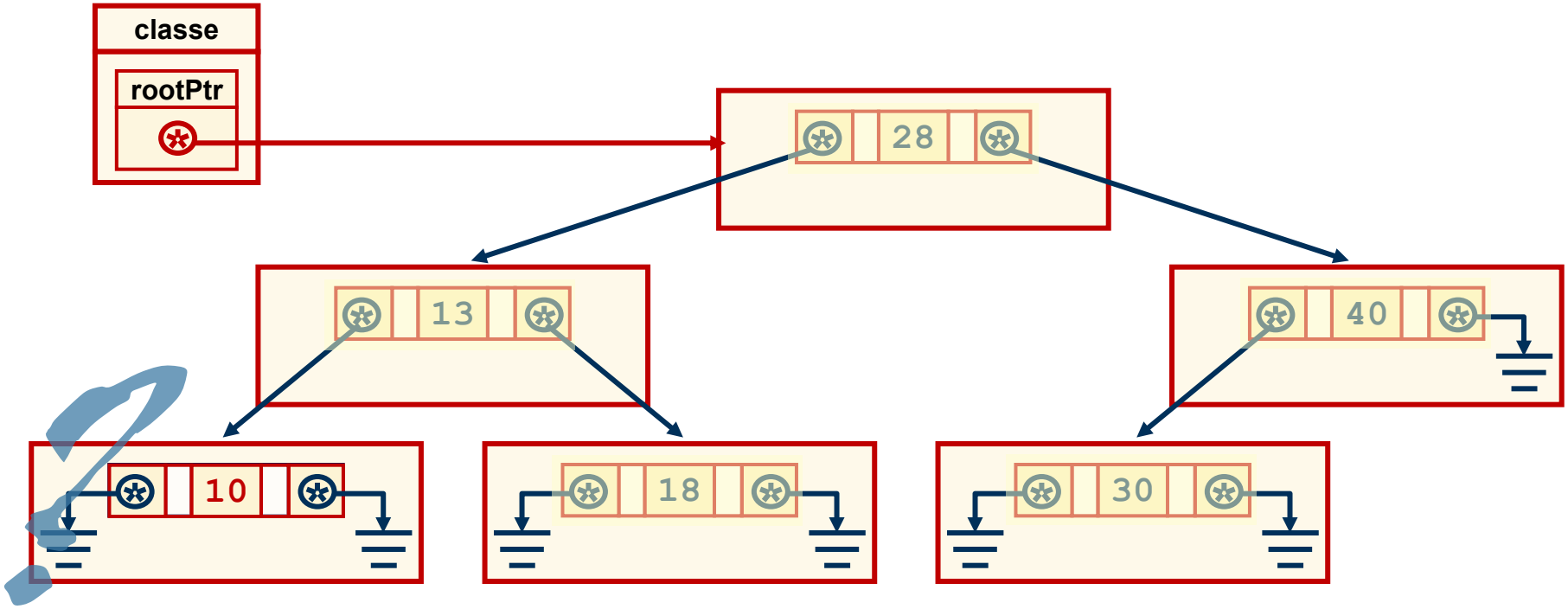
Attraversamento di alberi binari: ricorsione e iterazione, pile e code


Stampa
classe
rootPtr
28
13
40
10
18
30

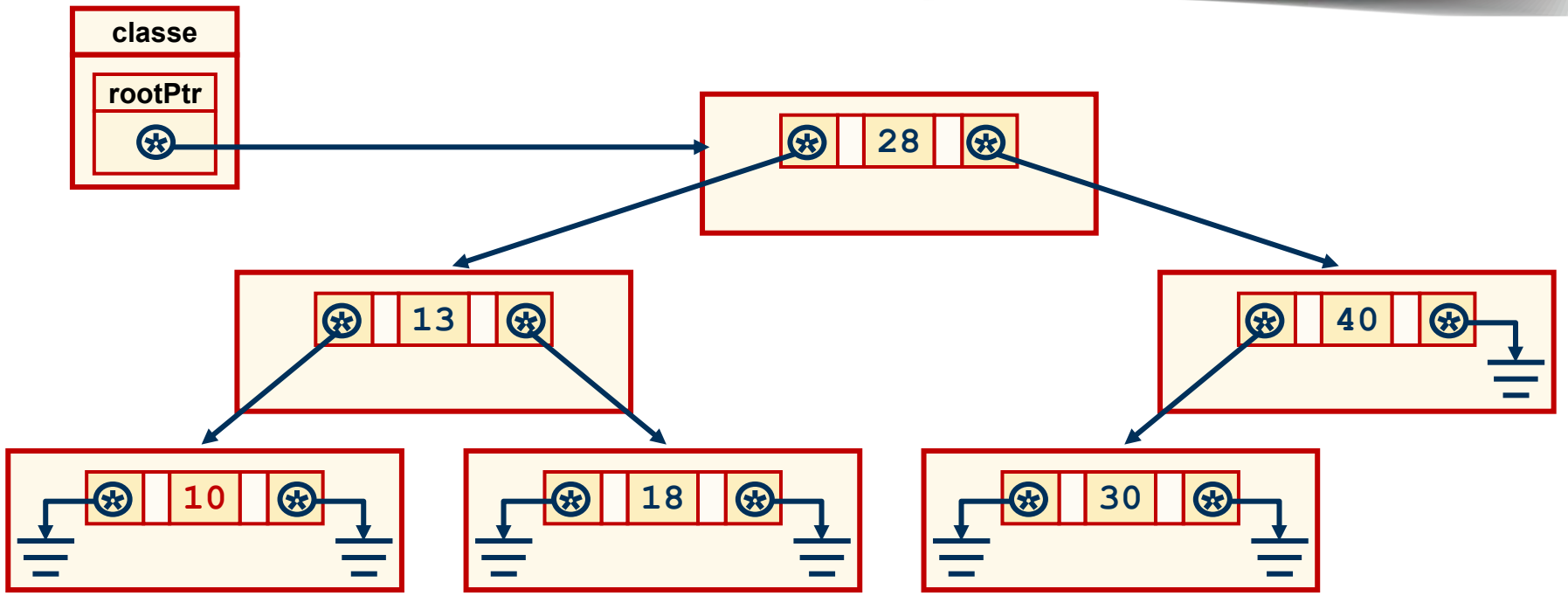
Stampa della matricola



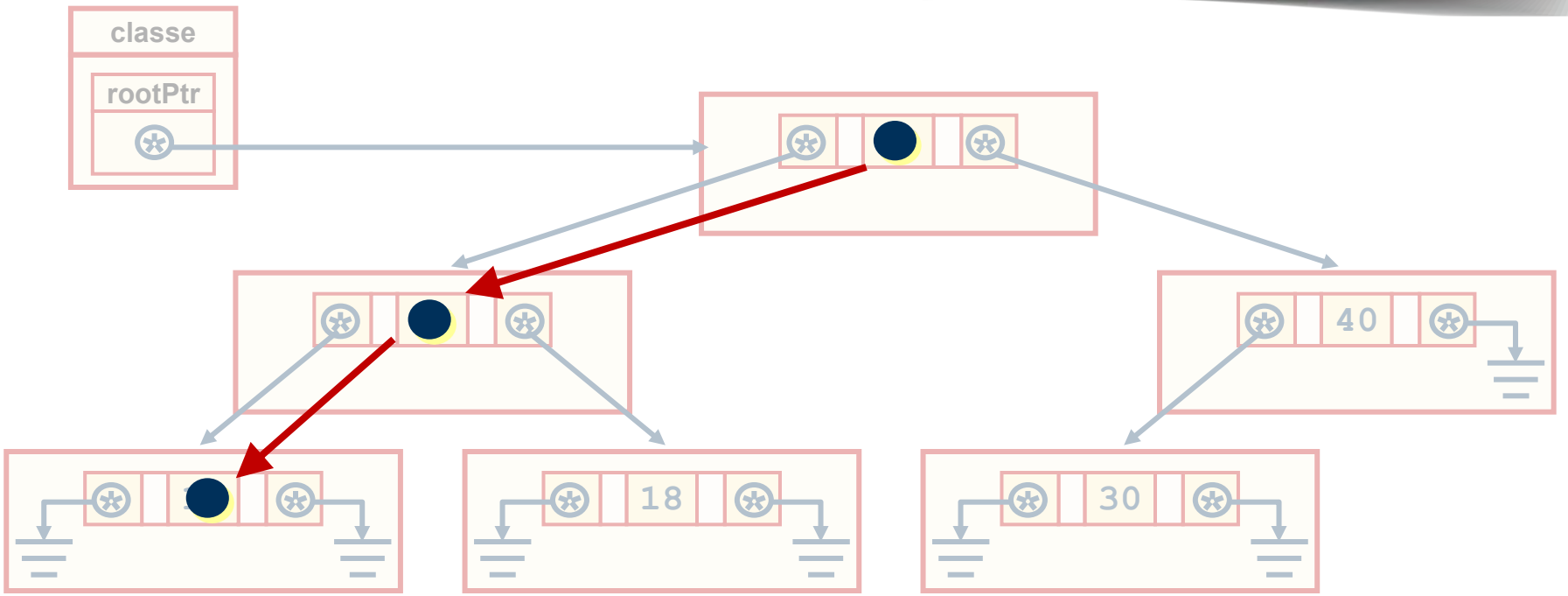
10 13 18 28 30 40

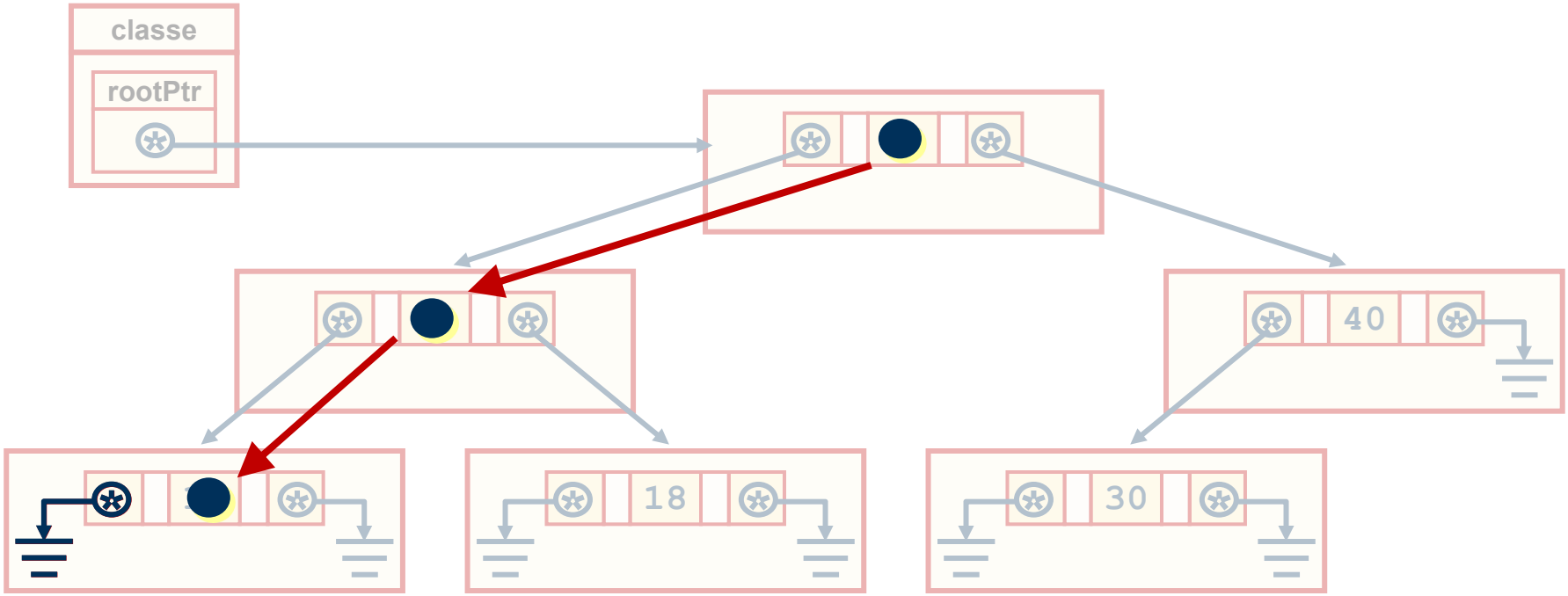

classe
rootPtr
28
13
40
10
18
30

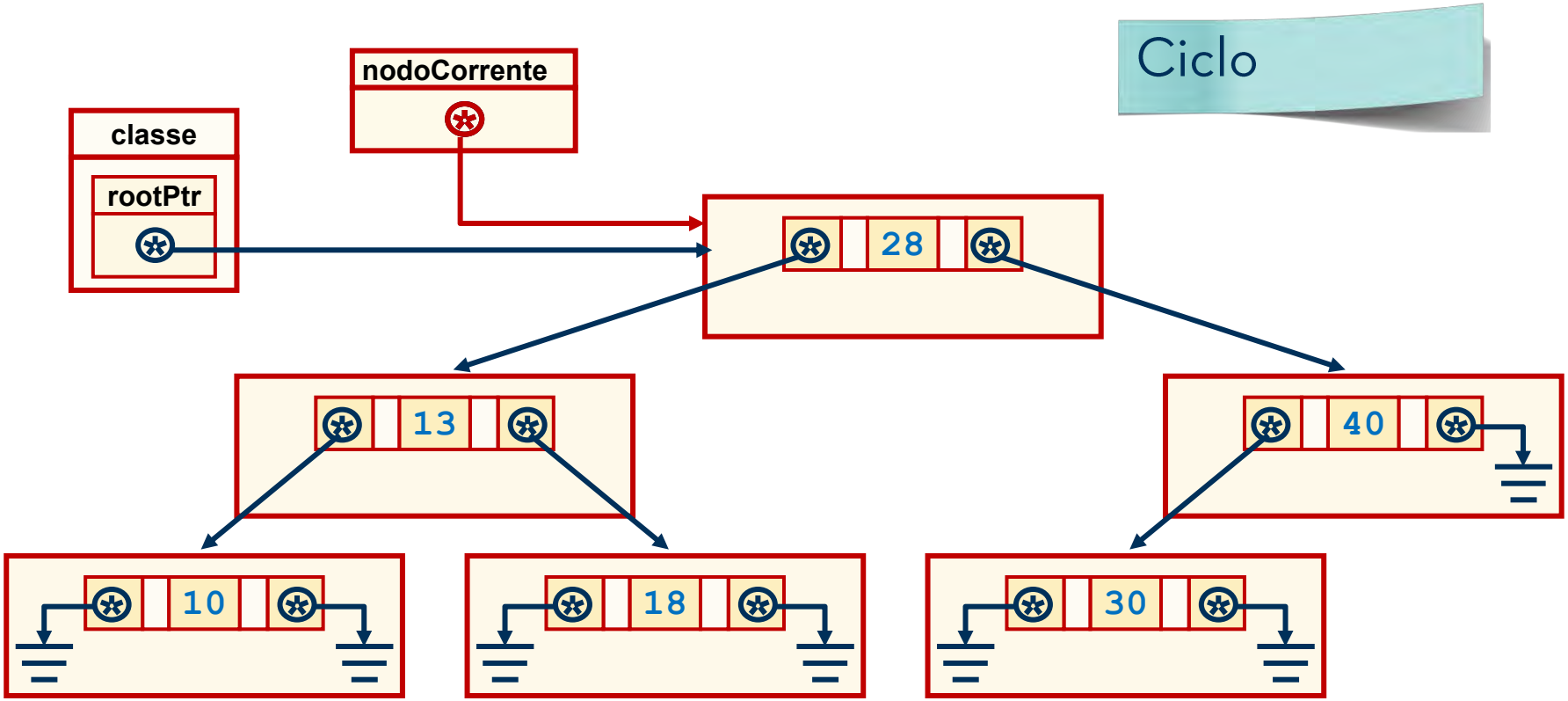
Valore a matricola minima

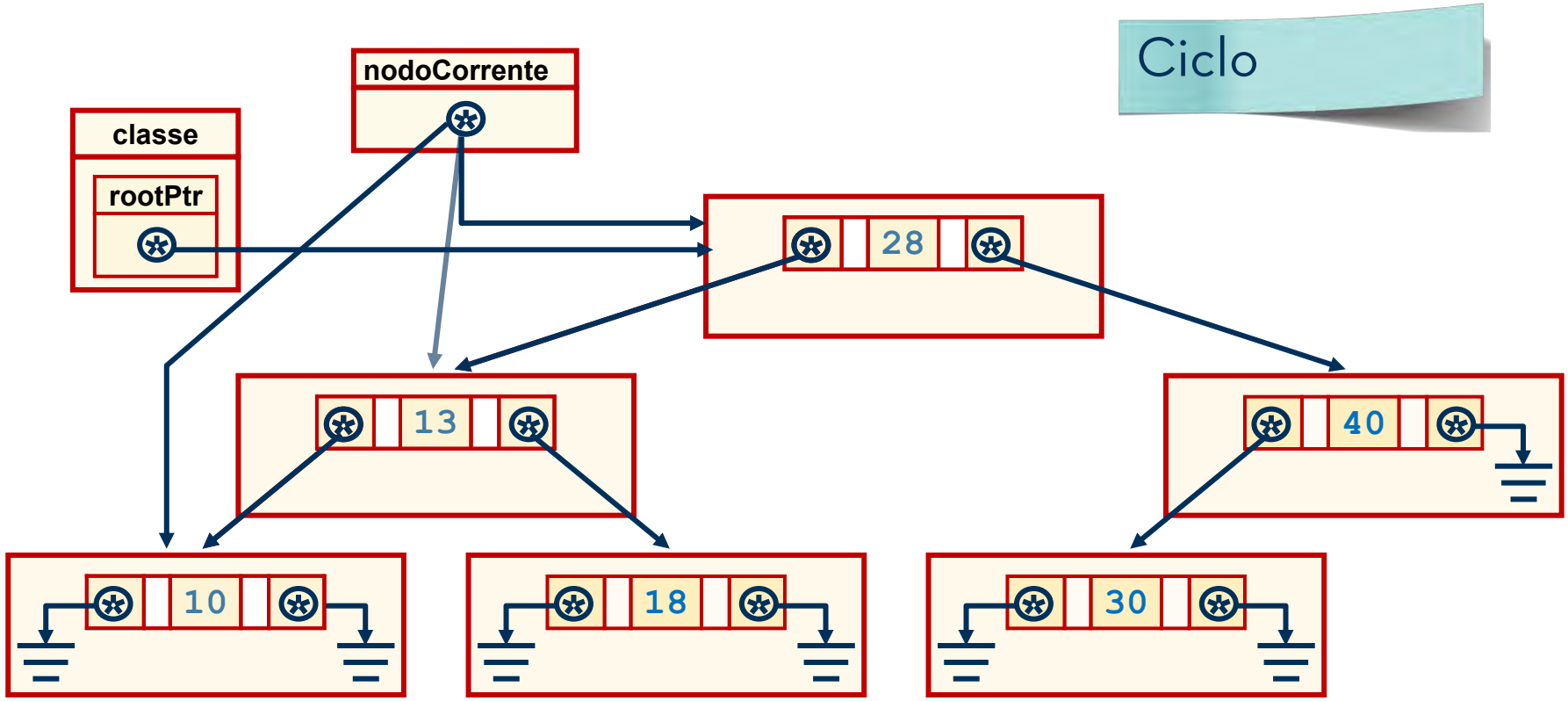


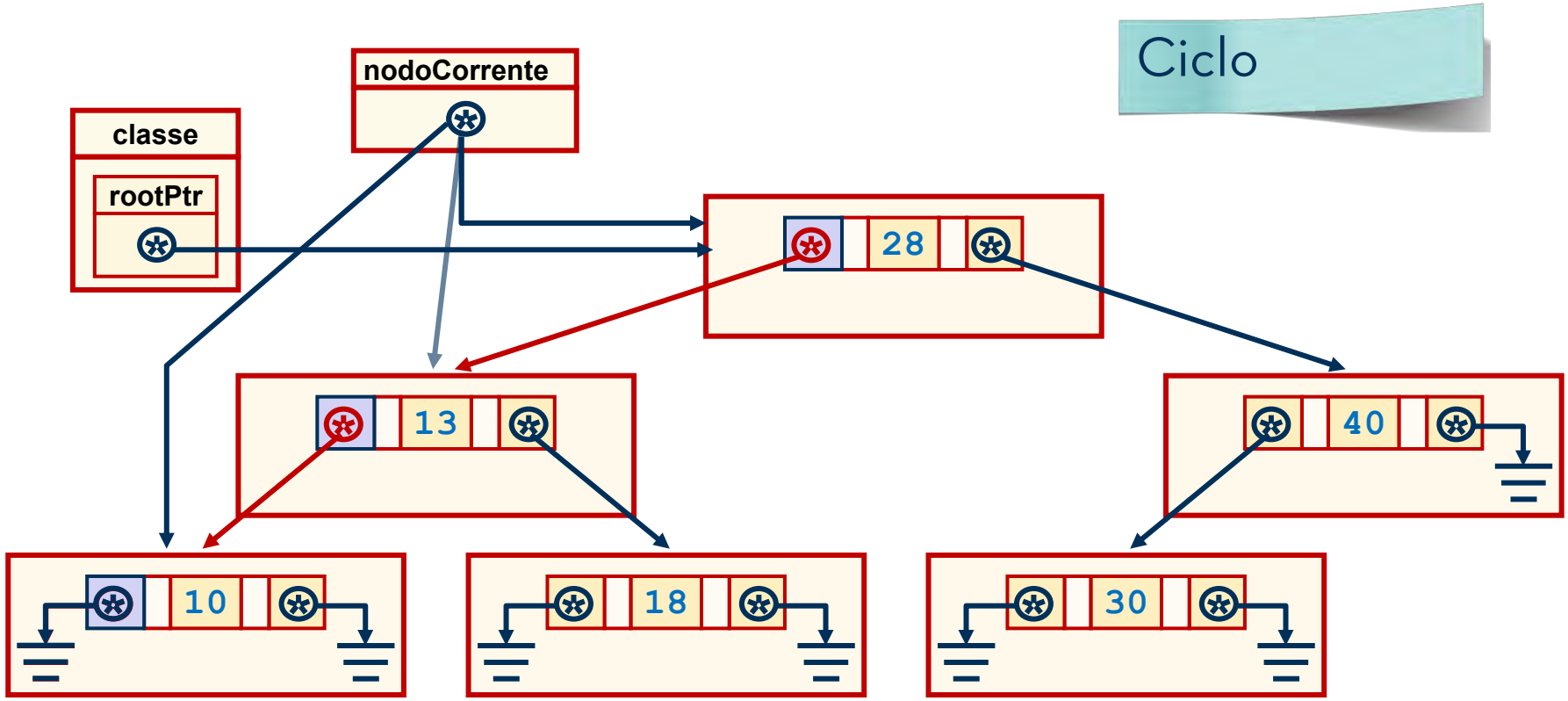
Valore a matricola minima

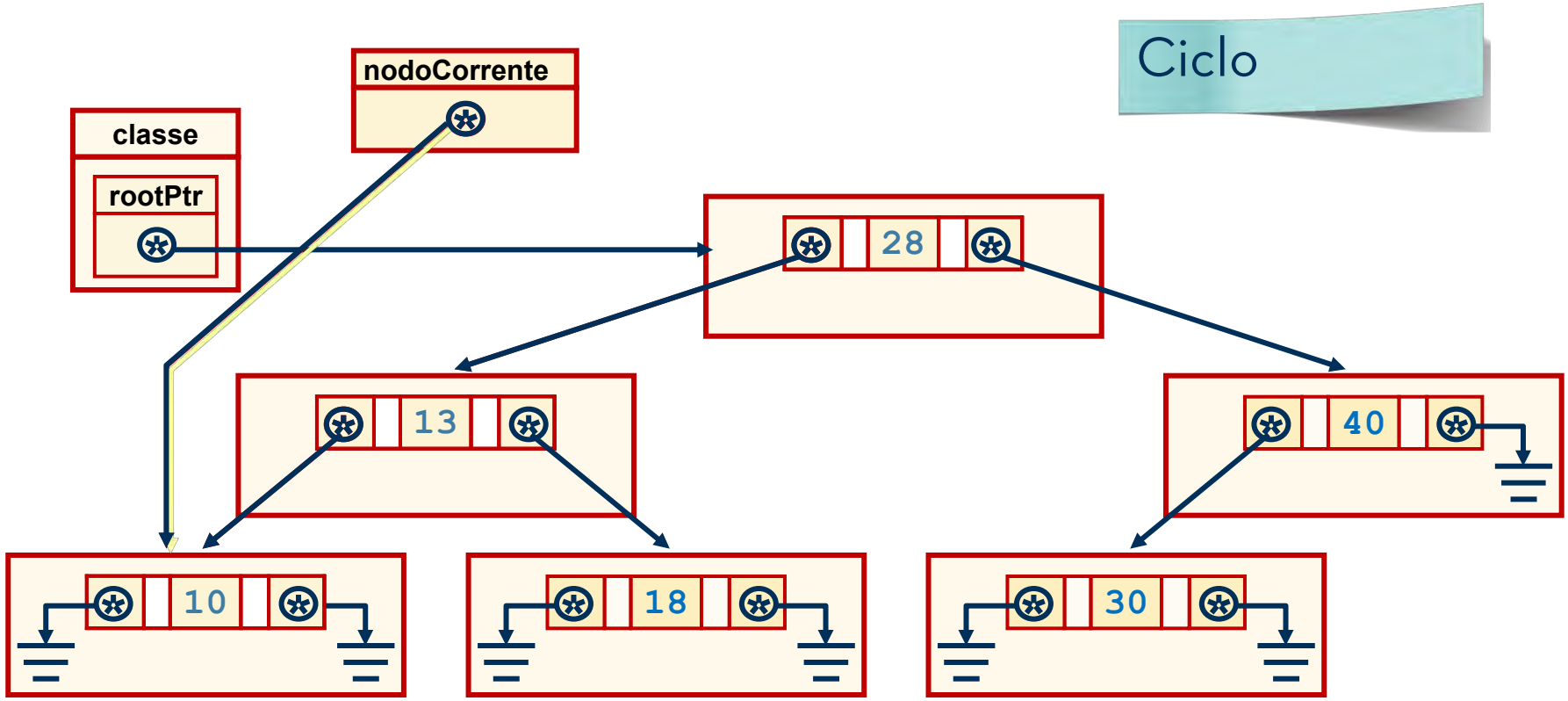


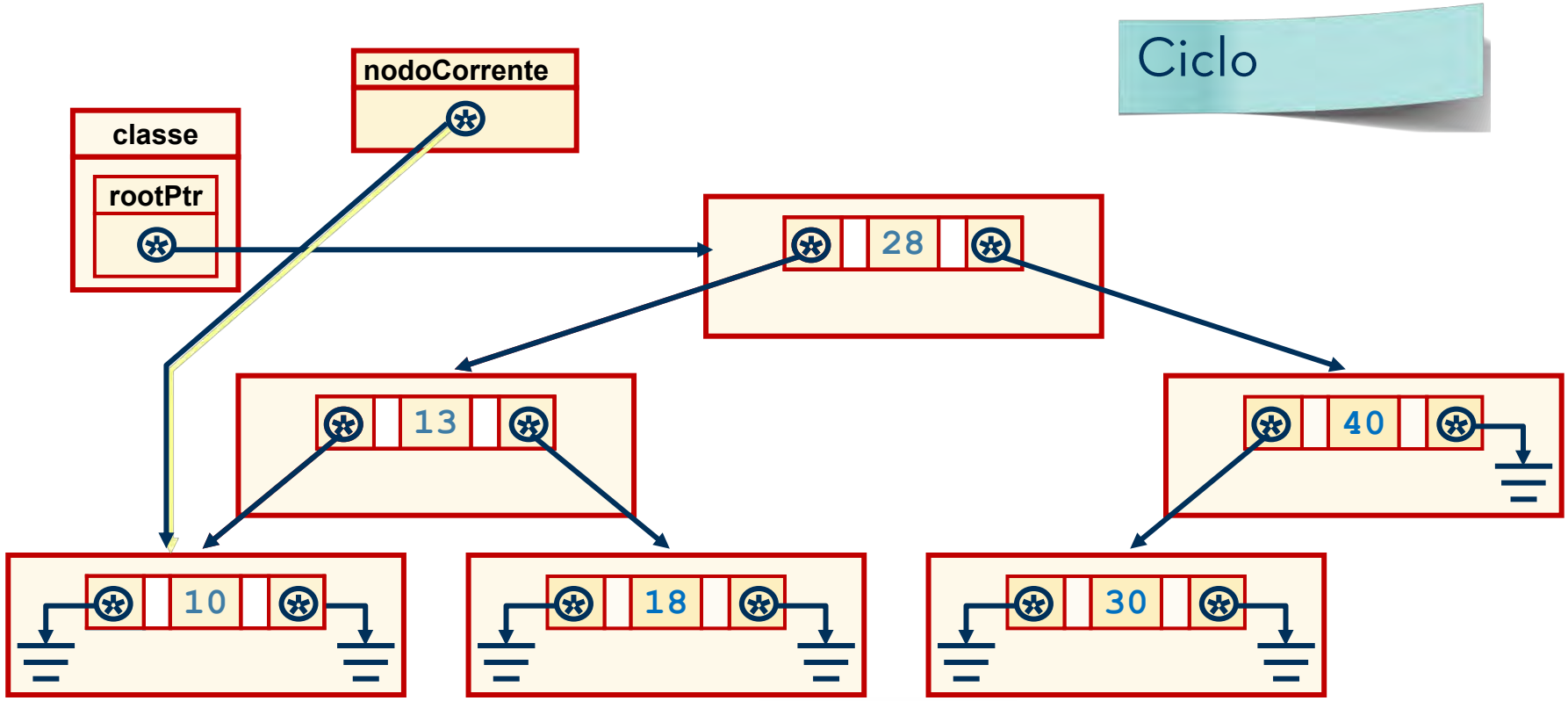

classe
rootPtr
40
18
30

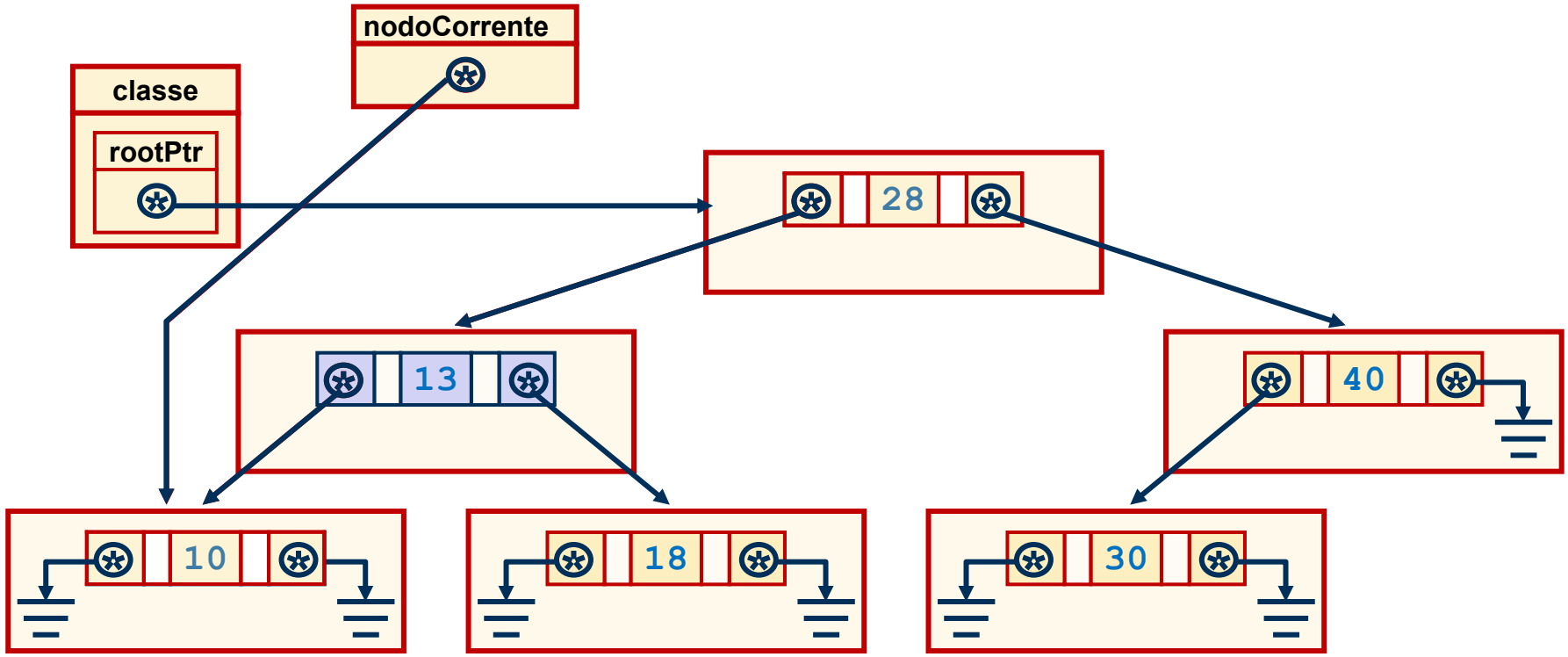

Ciclo
nodoCorrente
classe
rootPtr
28
13
40
10
18
30

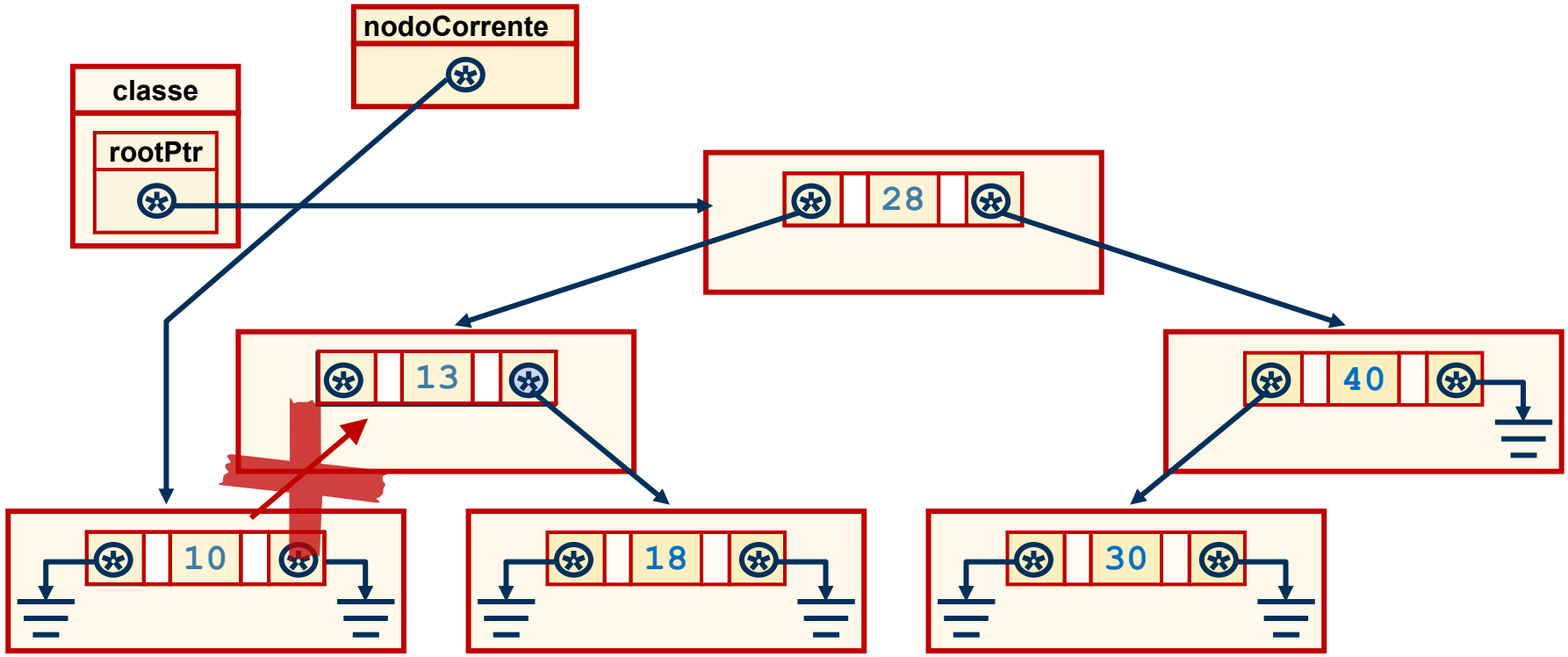

Ciclo
nodoCorrente
classe
rootPtr
28
13
40
10
18
30

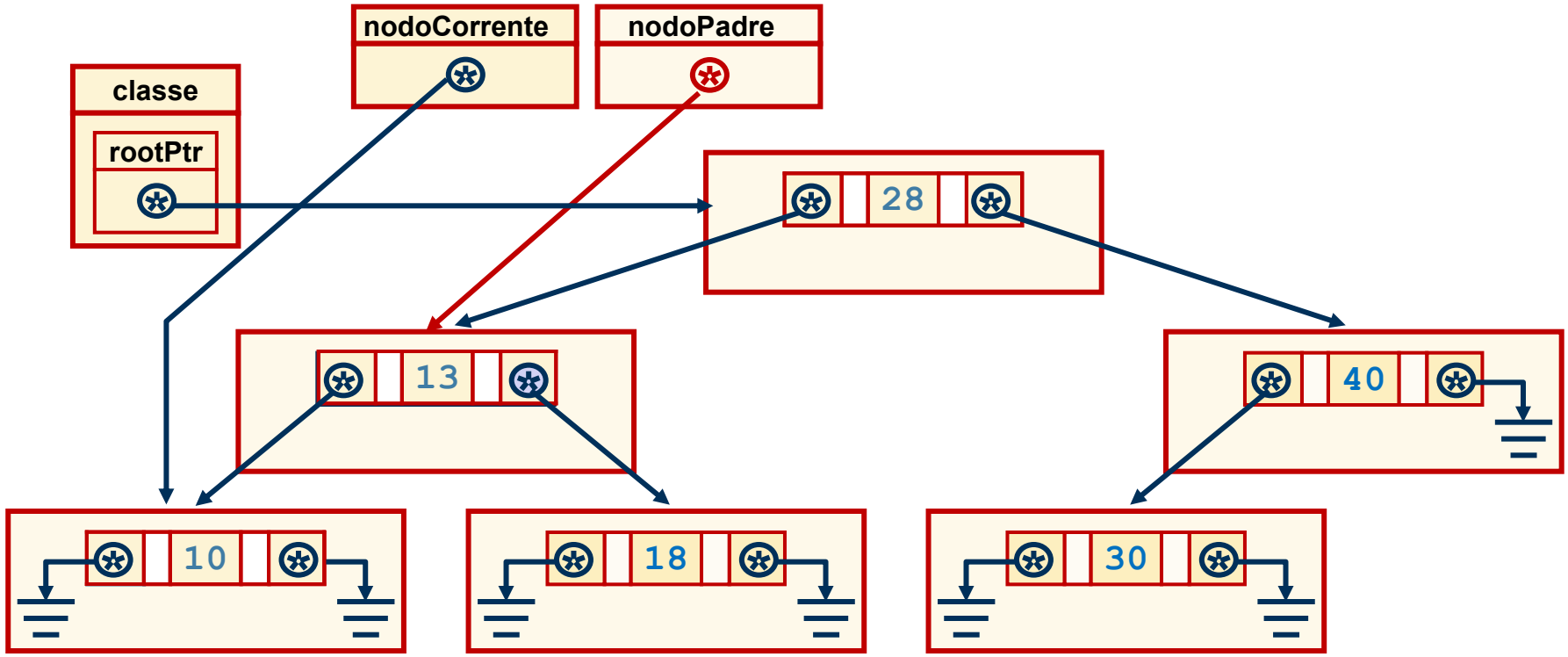

Ciclo
nodoCorrente
classe
rootPtr
28
13
40
10
18
30

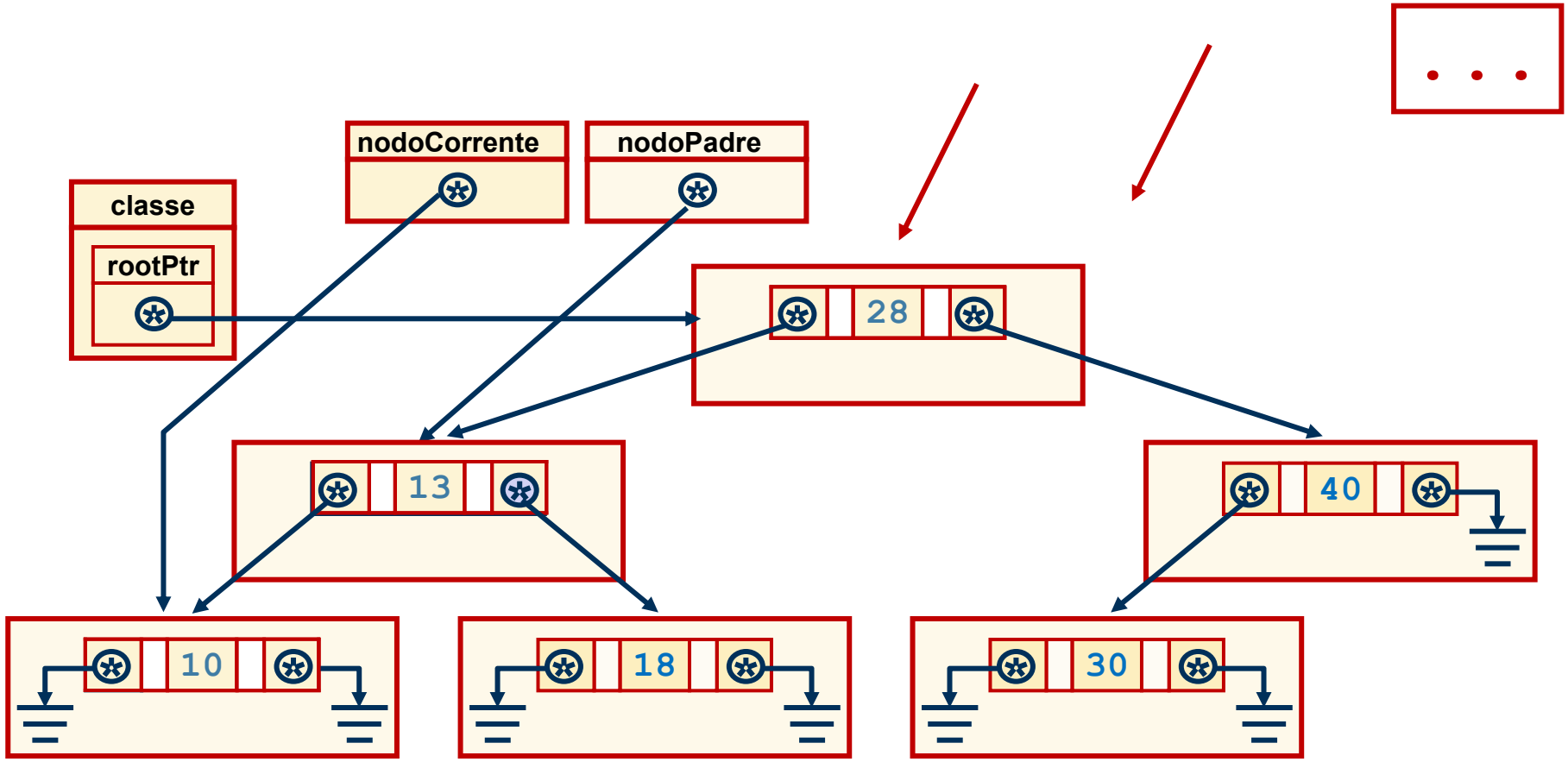

Ciclo
nodoCorrente
classe
rootPtr
28
13
40
10
18
30


Ciclo
nodoCorrente
classe
rootPtr
28
13
40
10
18
30

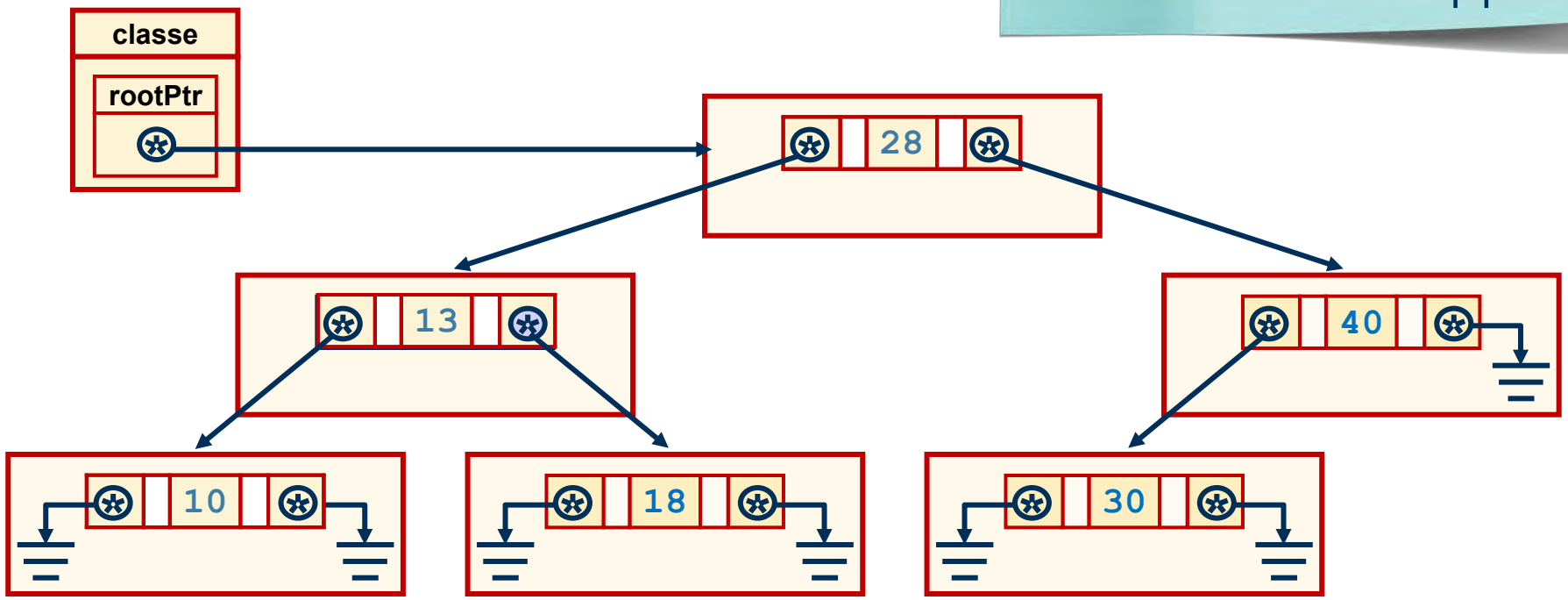

nodoCorrente
classe
rootPtr
28
13
40
10
18
30

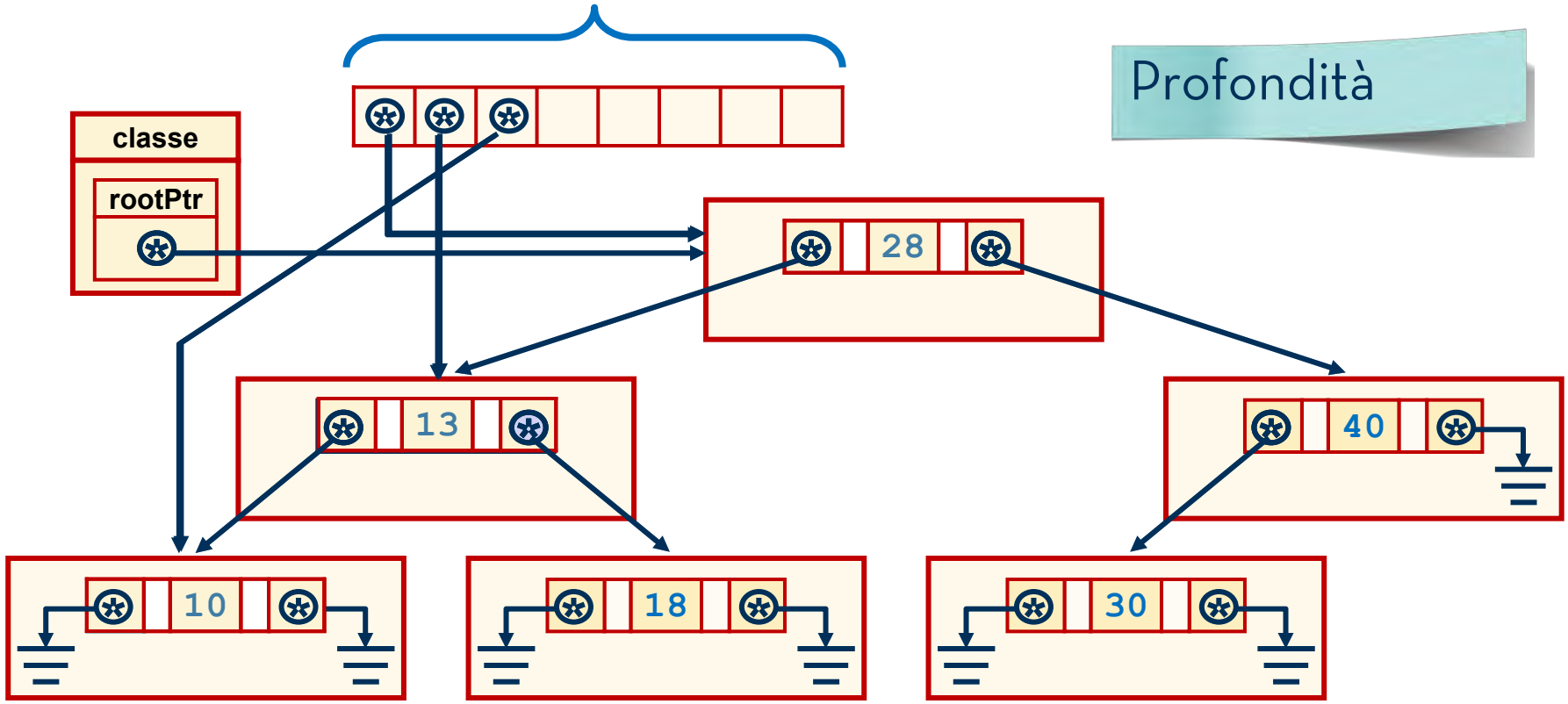

nodoCorrente
classe
rootPtr
28
13
40
10
18
30

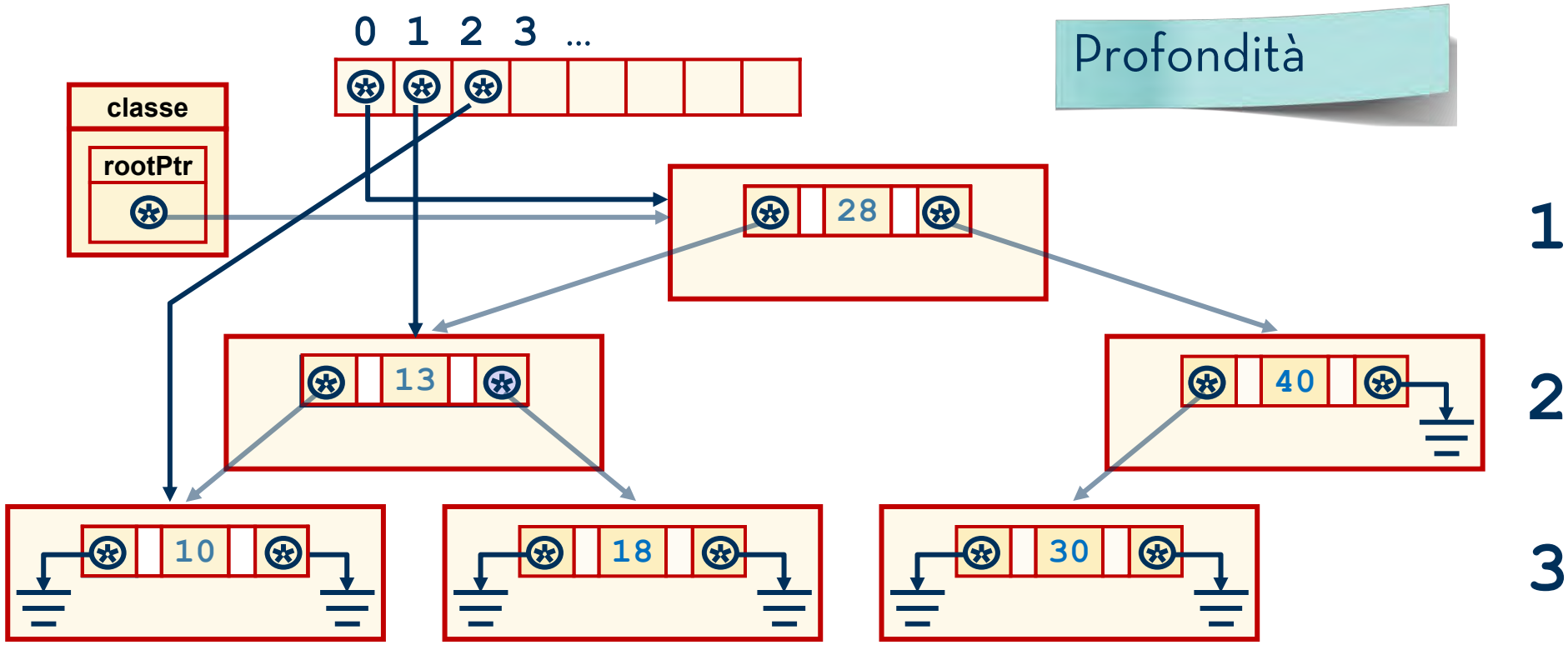

nodoCorrente
nodoPadre
classe
rootPtr
28
13
40
10
18
30

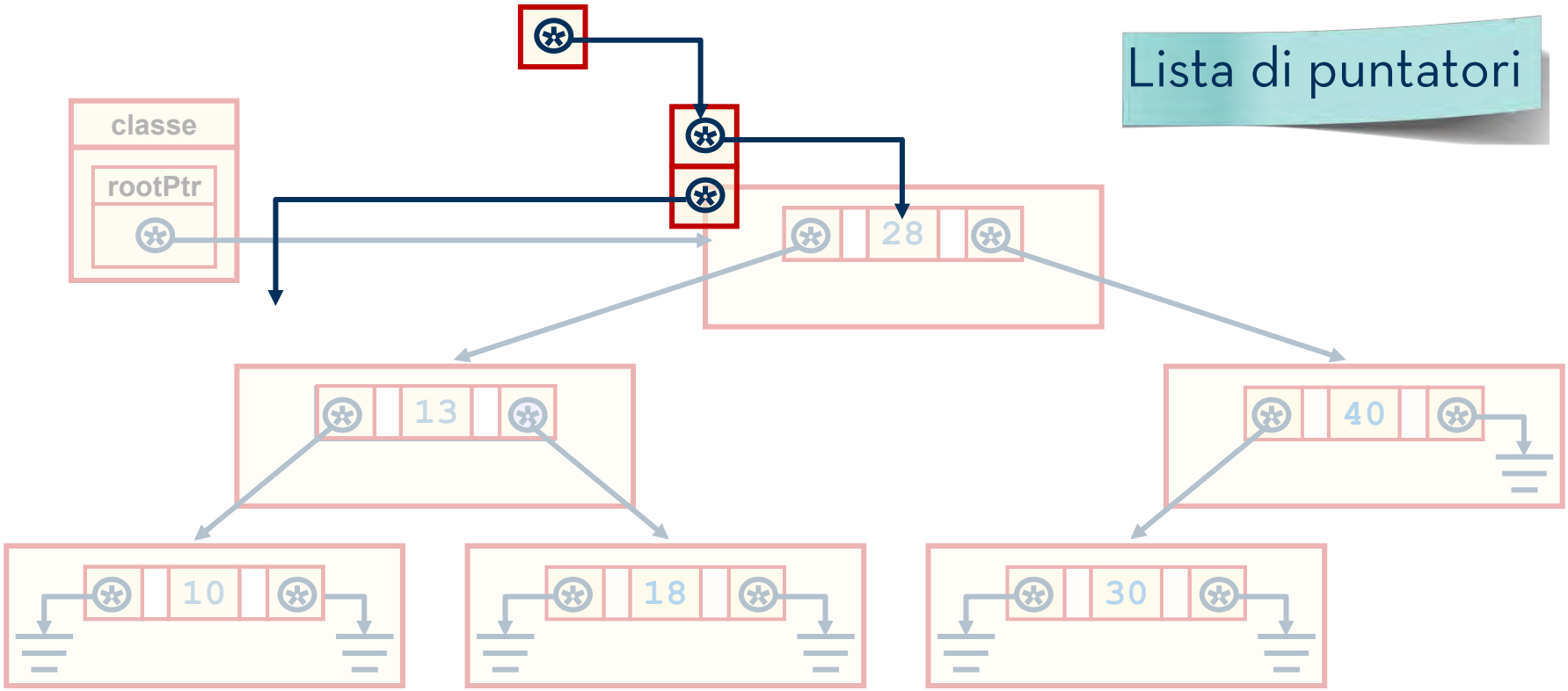

. . .
nodoCorrente
nodoPadre
classe
rootPtr
28
13
40
10
18
30

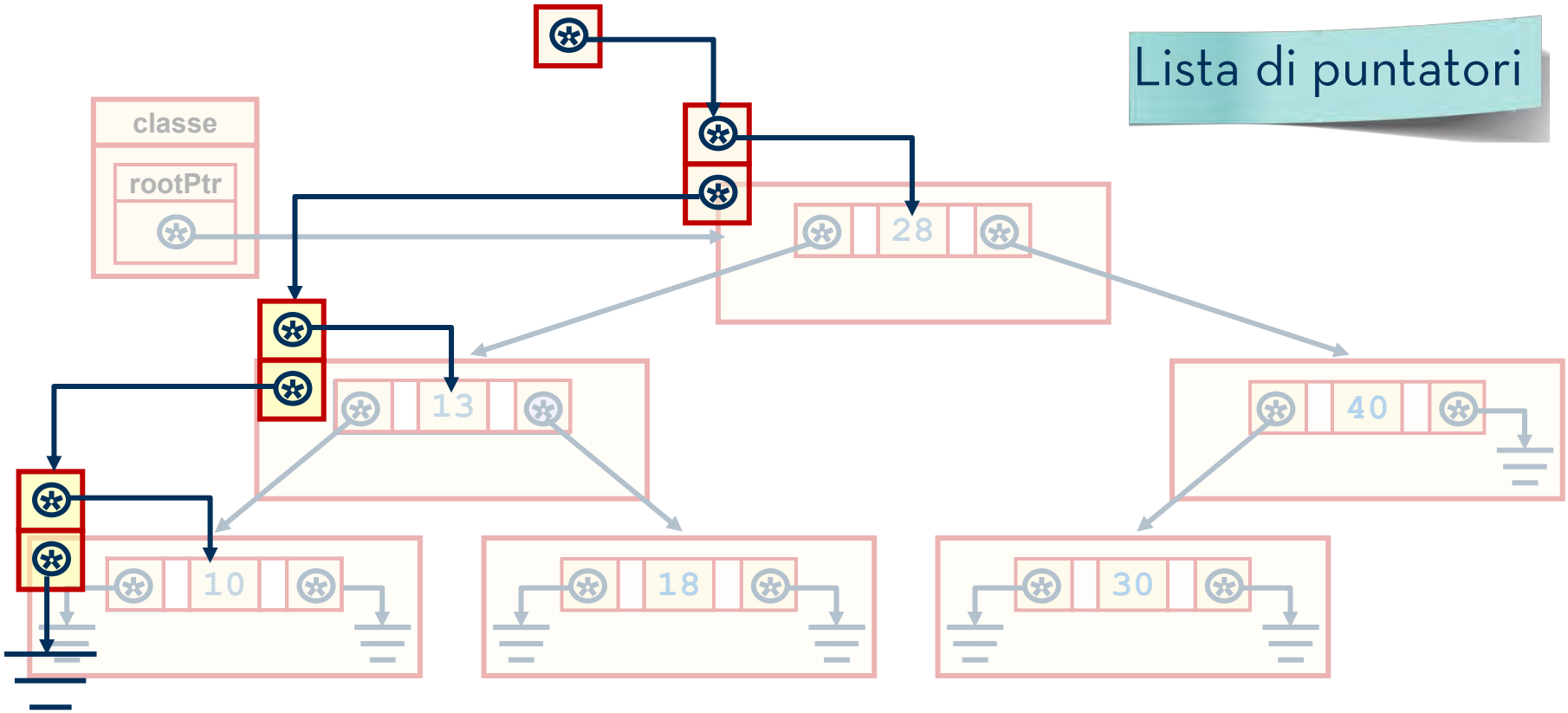
Struttura dati di supporto

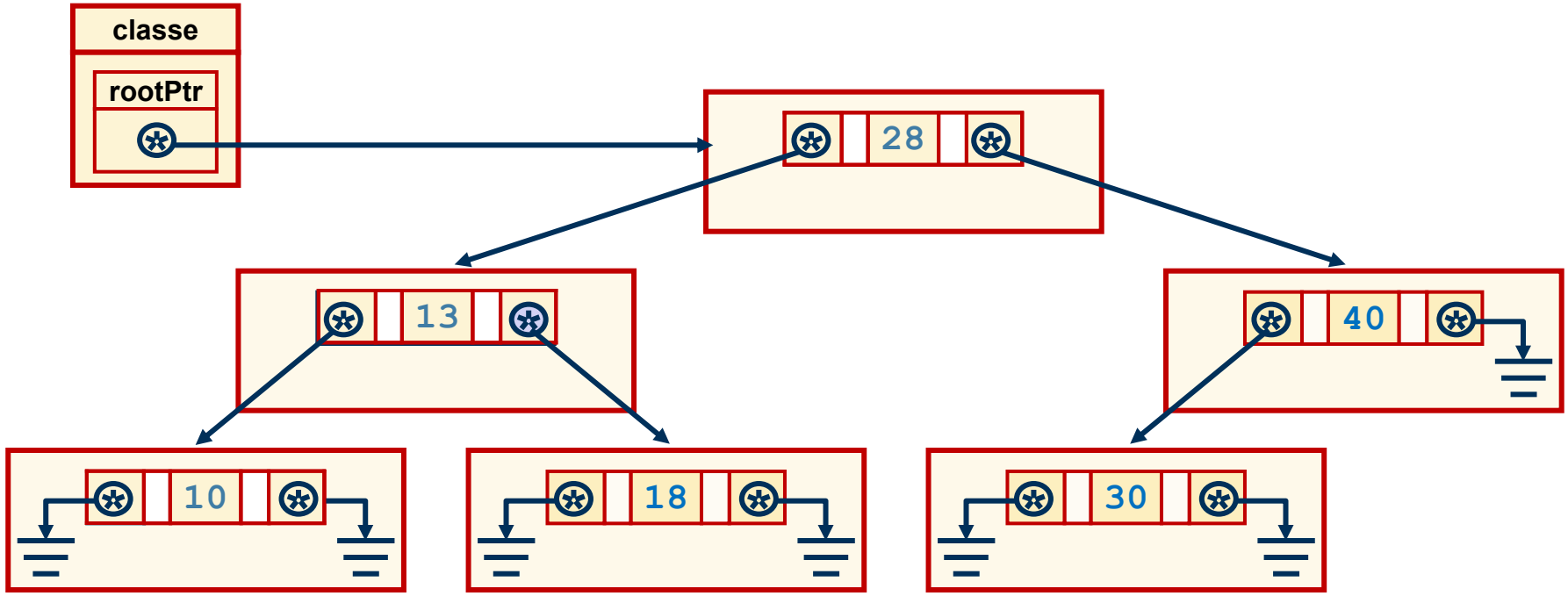


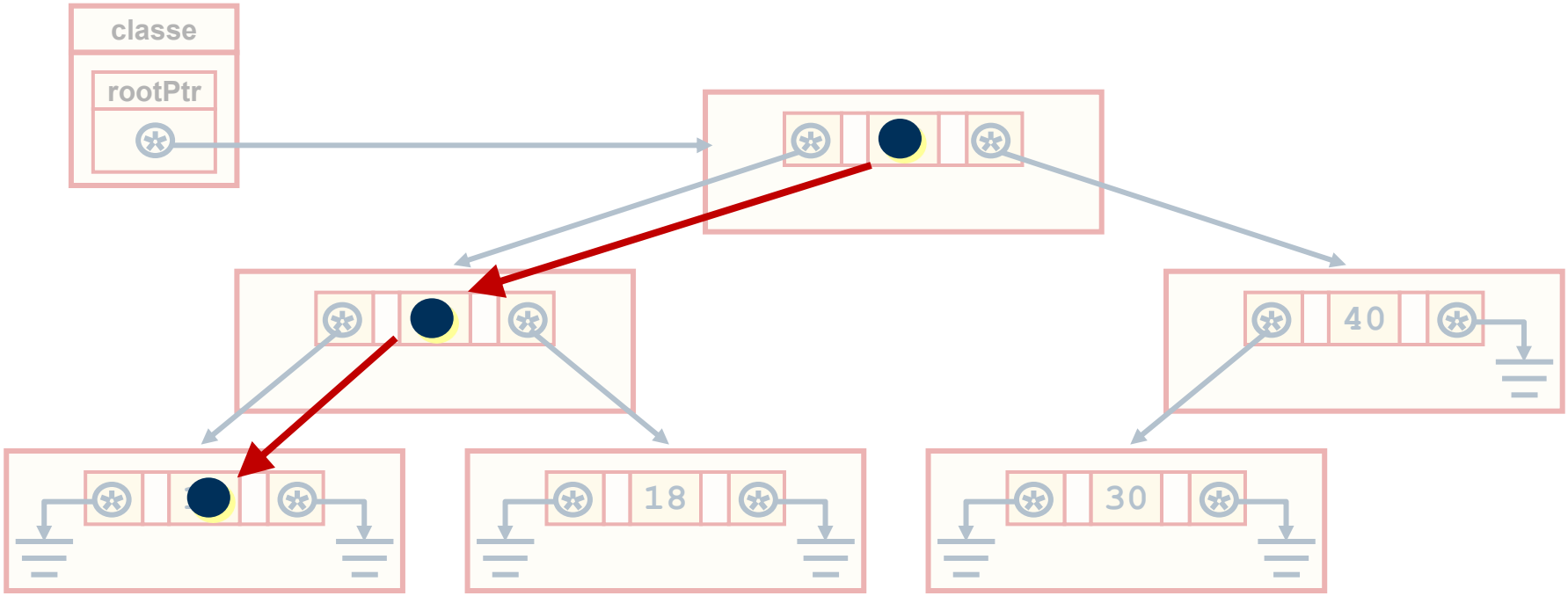

Profondità
classe
rootPtr
28
13
40
10
18
30

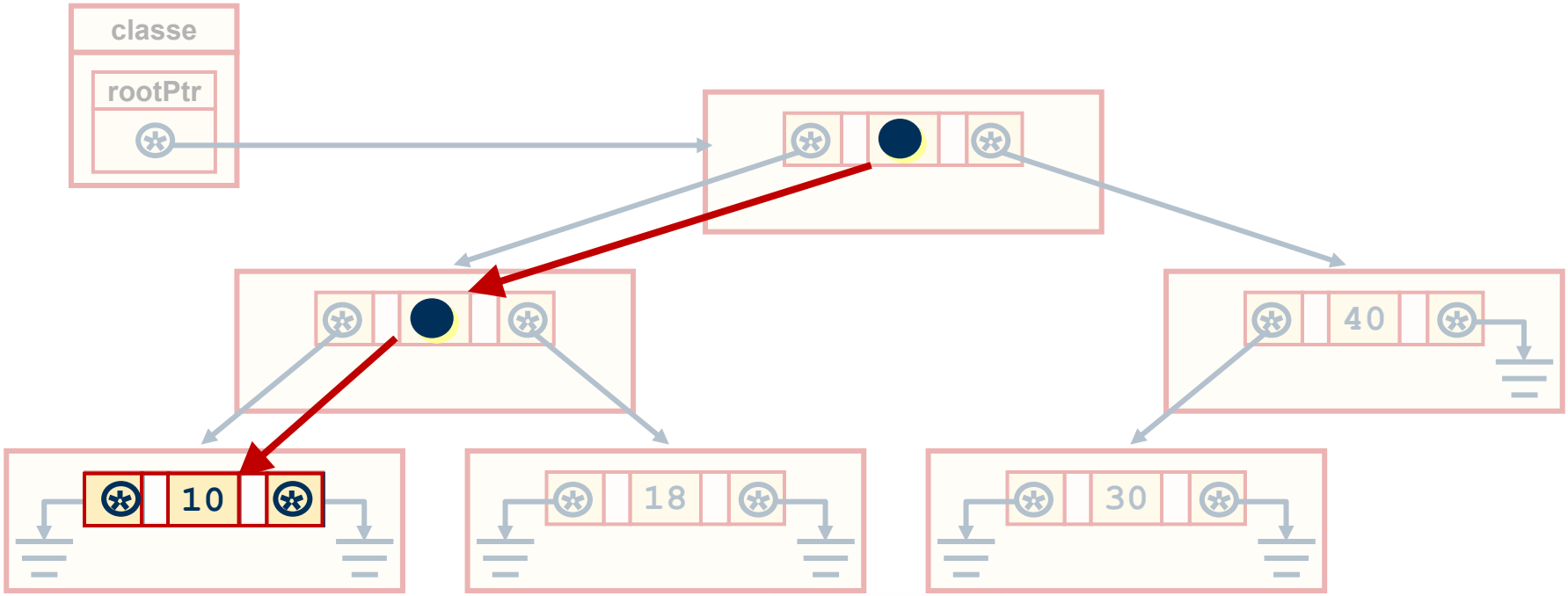

0 1 2 3 ...
classe
rootPtr
Profondità
28
13
40
10
18
30
1
2
3

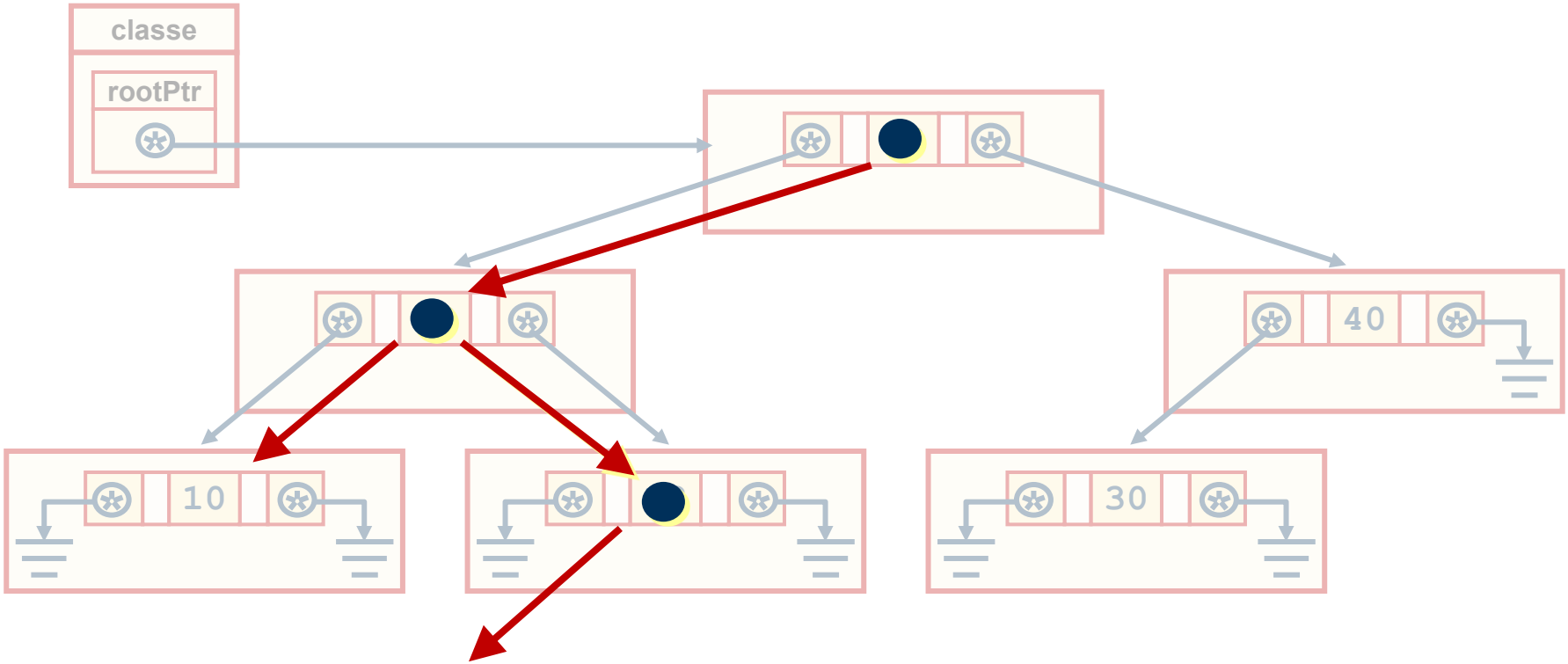

Lista di puntatori
classe
rootPtr
28
13
40
10
18
30

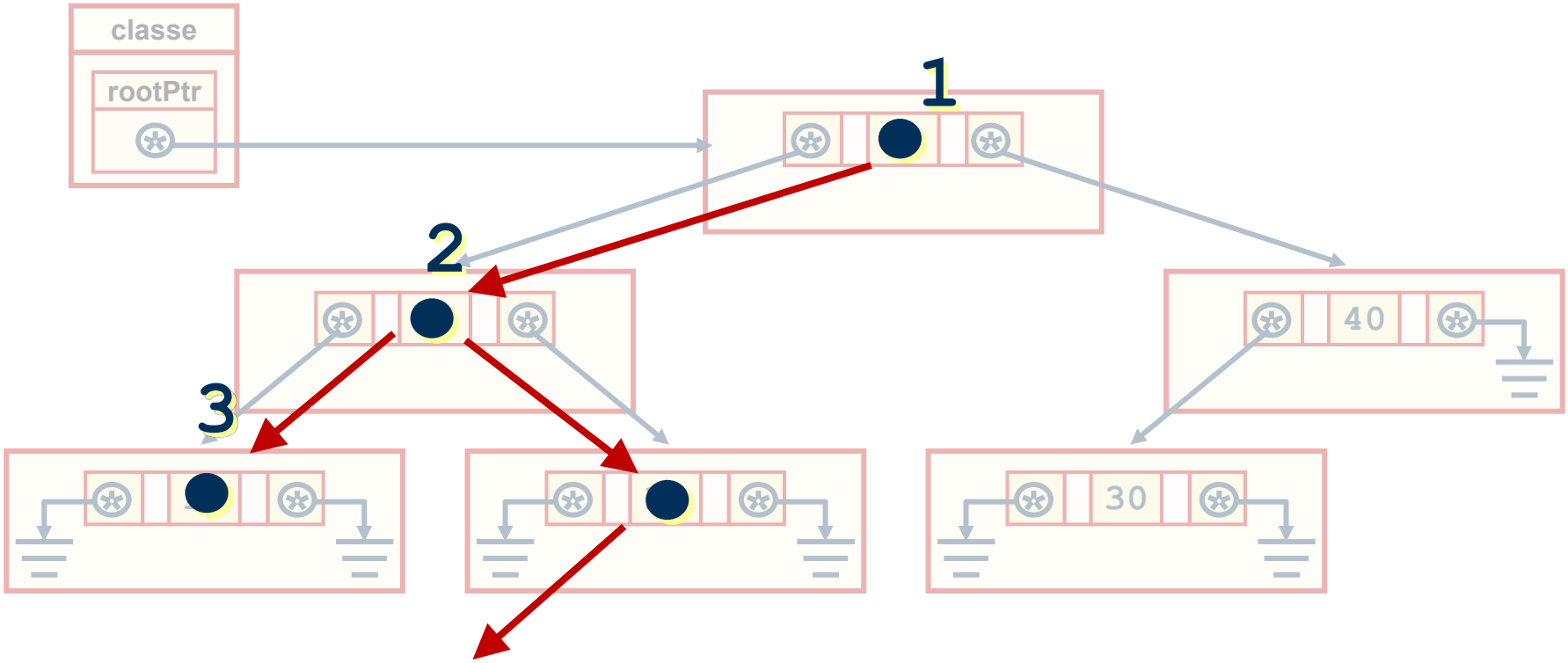

Lista di puntatori
classe
rootPtr
28
13
40
10
18
30

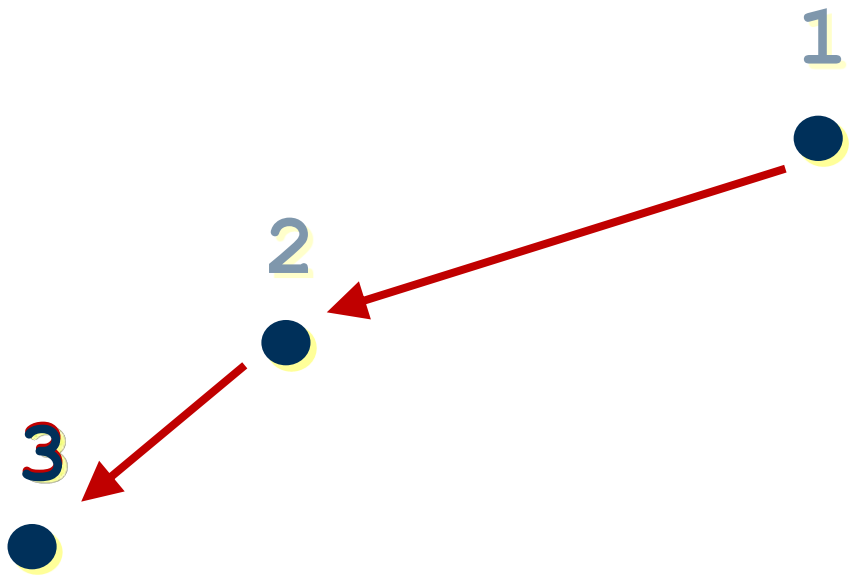

classe
rootPtr
28
13
40
10
18
30


classe
rootPtr
40
18
30


classe
rootPtr
40
10
18
30

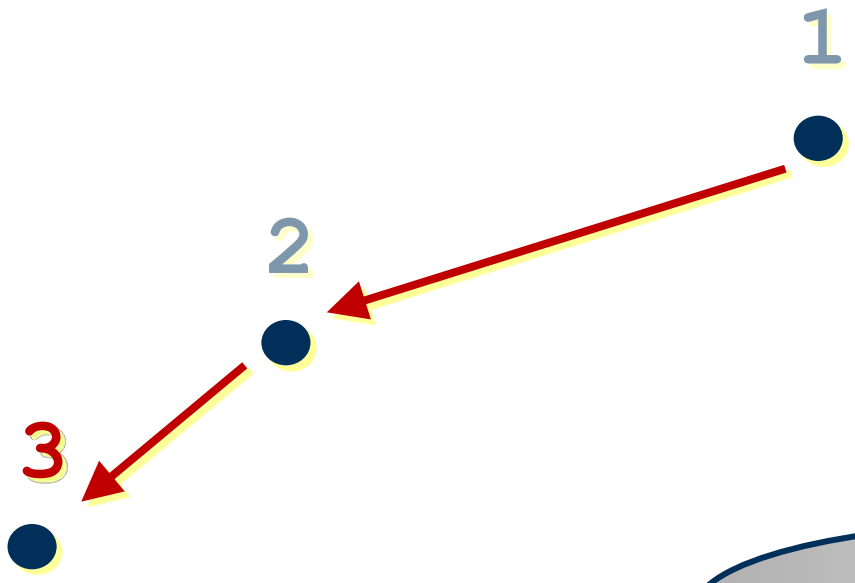

classe
rootPtr
40
10
30


classe
rootPtr
1
2
3
40
30

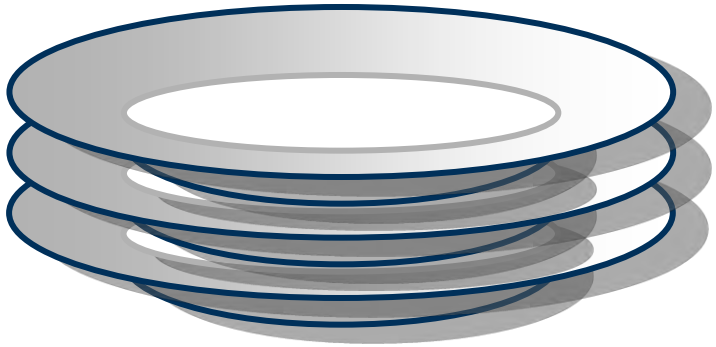


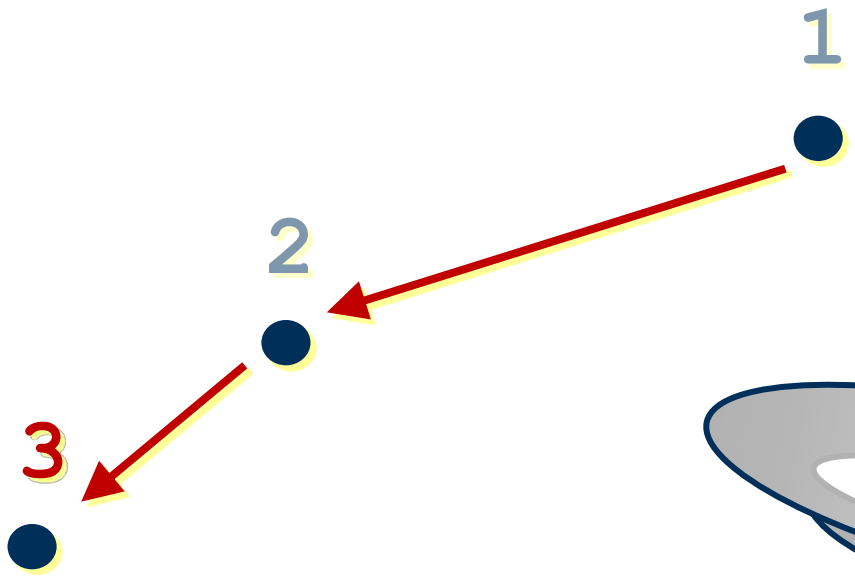
Last In First Out

Strutture a pila

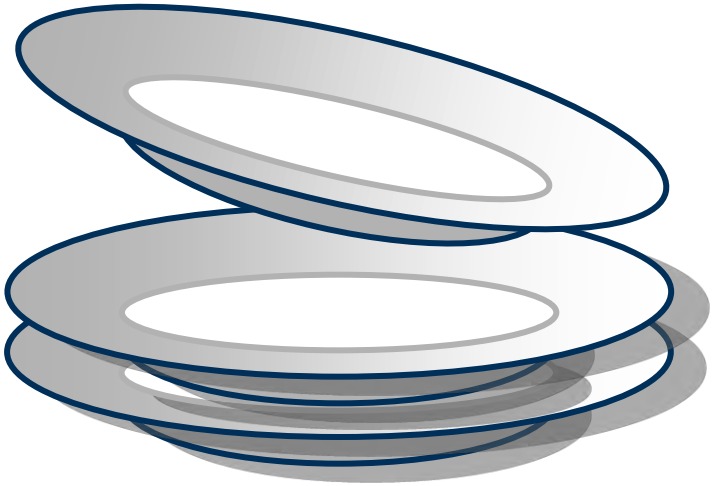


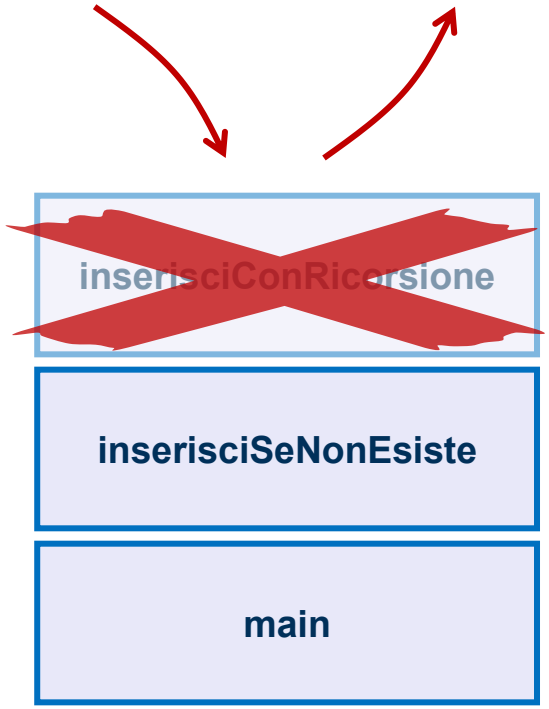
Last
In
First
Out





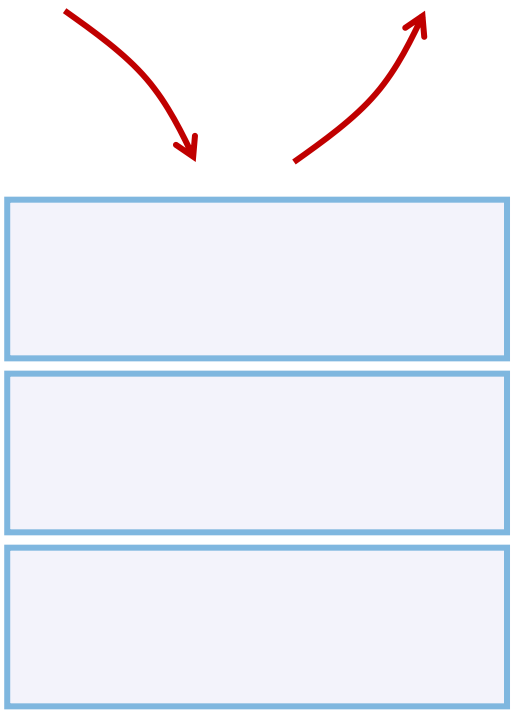
Last
In
First
Out

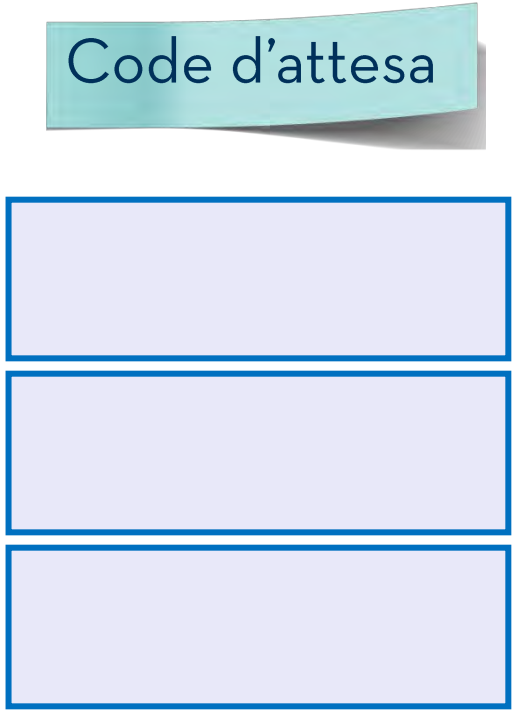




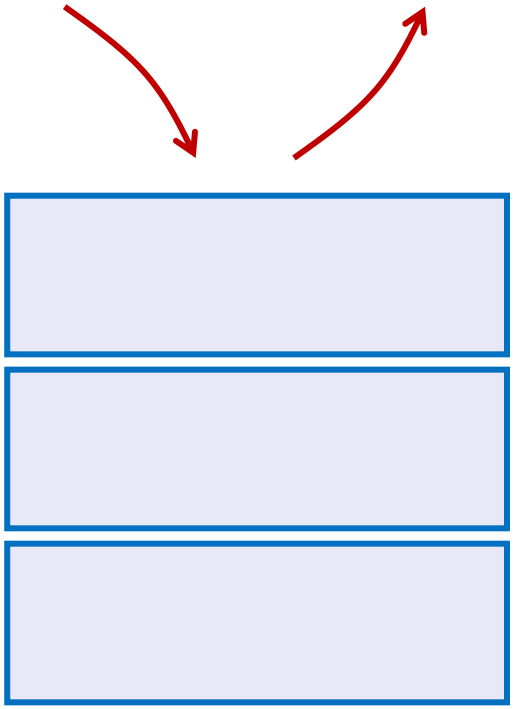
Last
In
First
Out

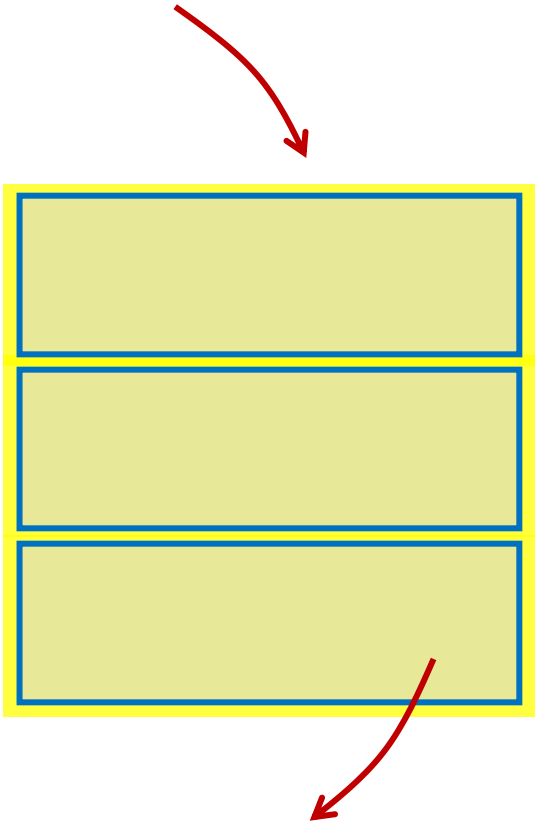
Last
In
First
Out



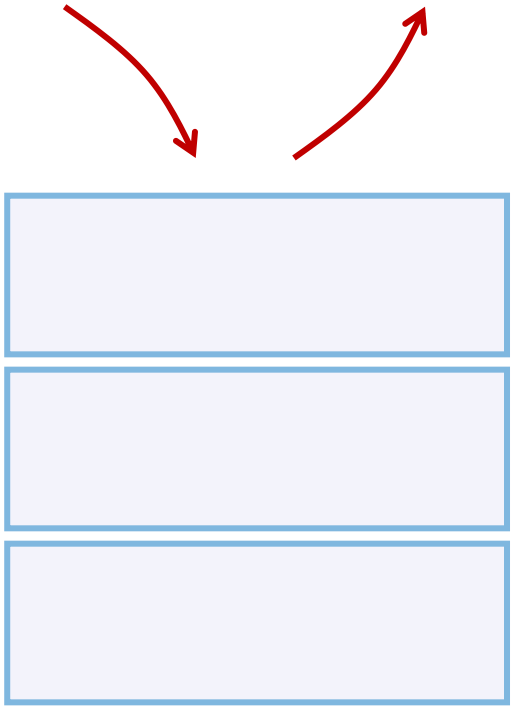


Last
In
First
Out

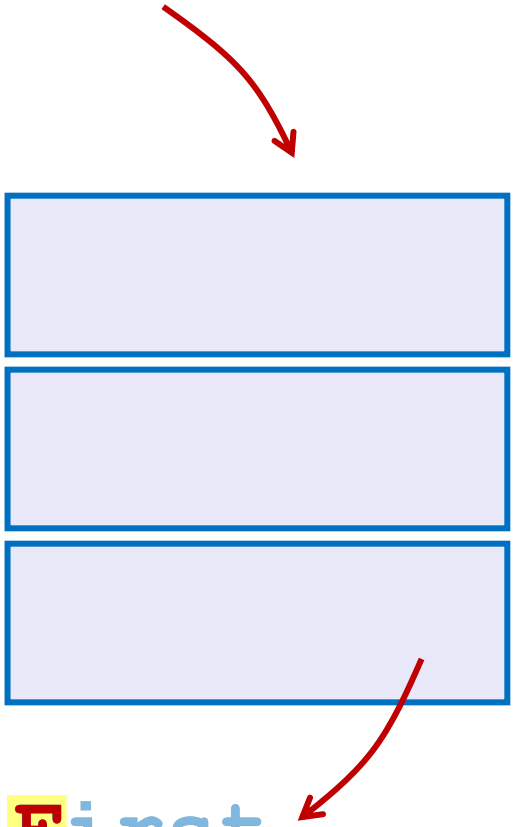




Last
In
First
Out

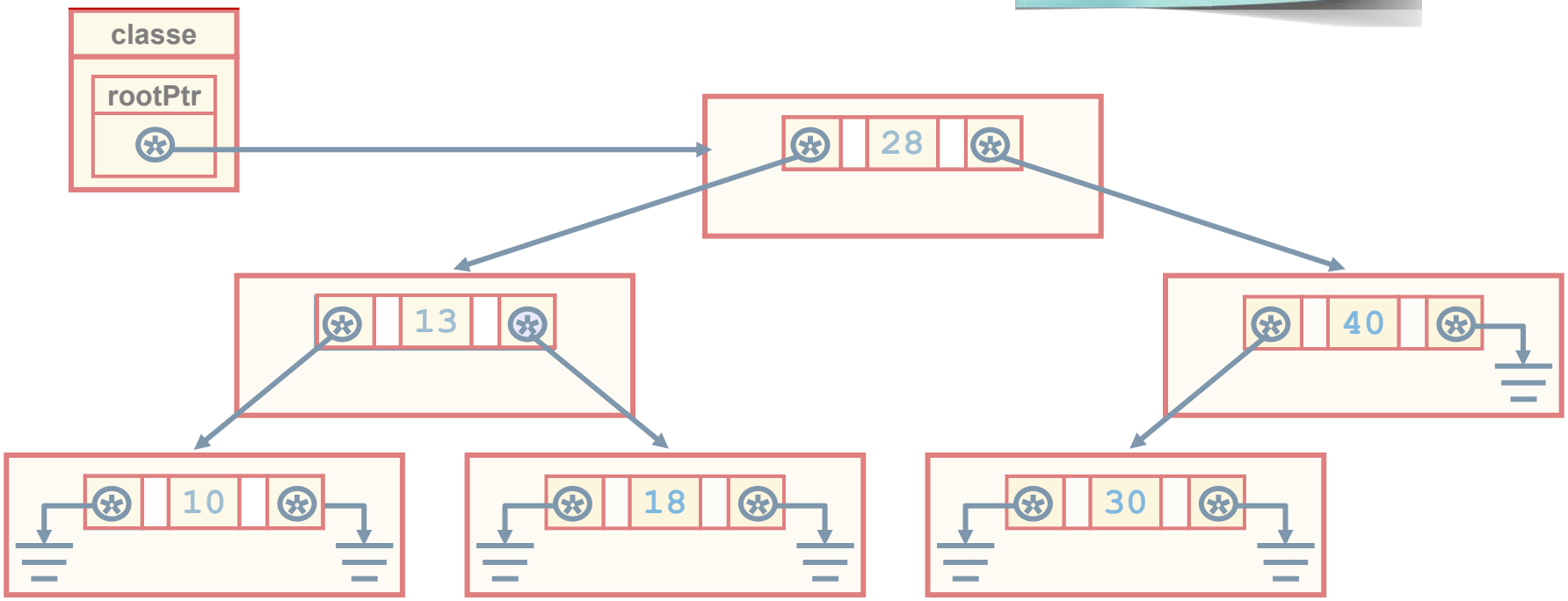


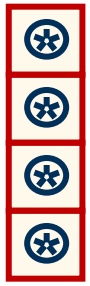
Last
In
First
Out

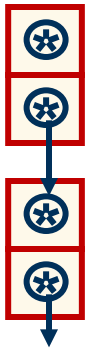


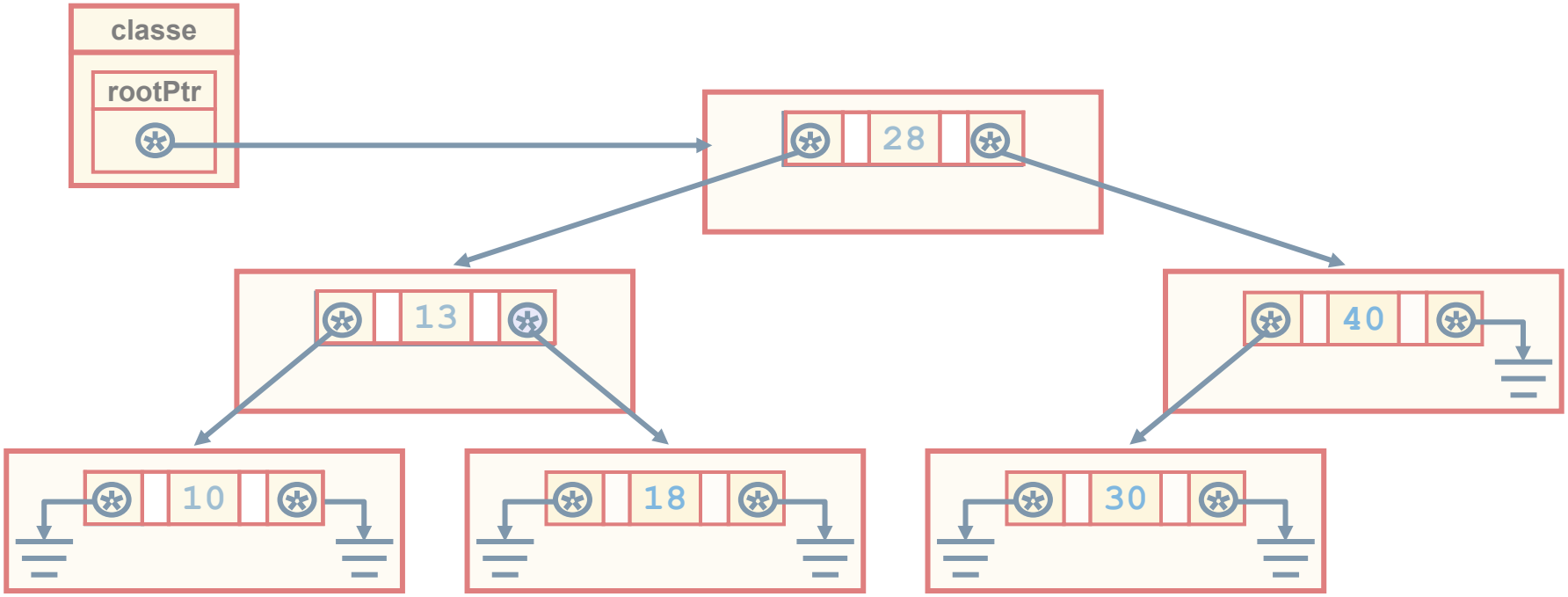
First
In
First
Out

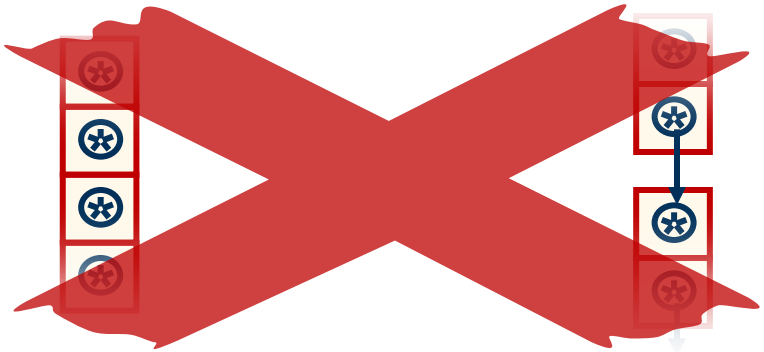
Struttura a pila



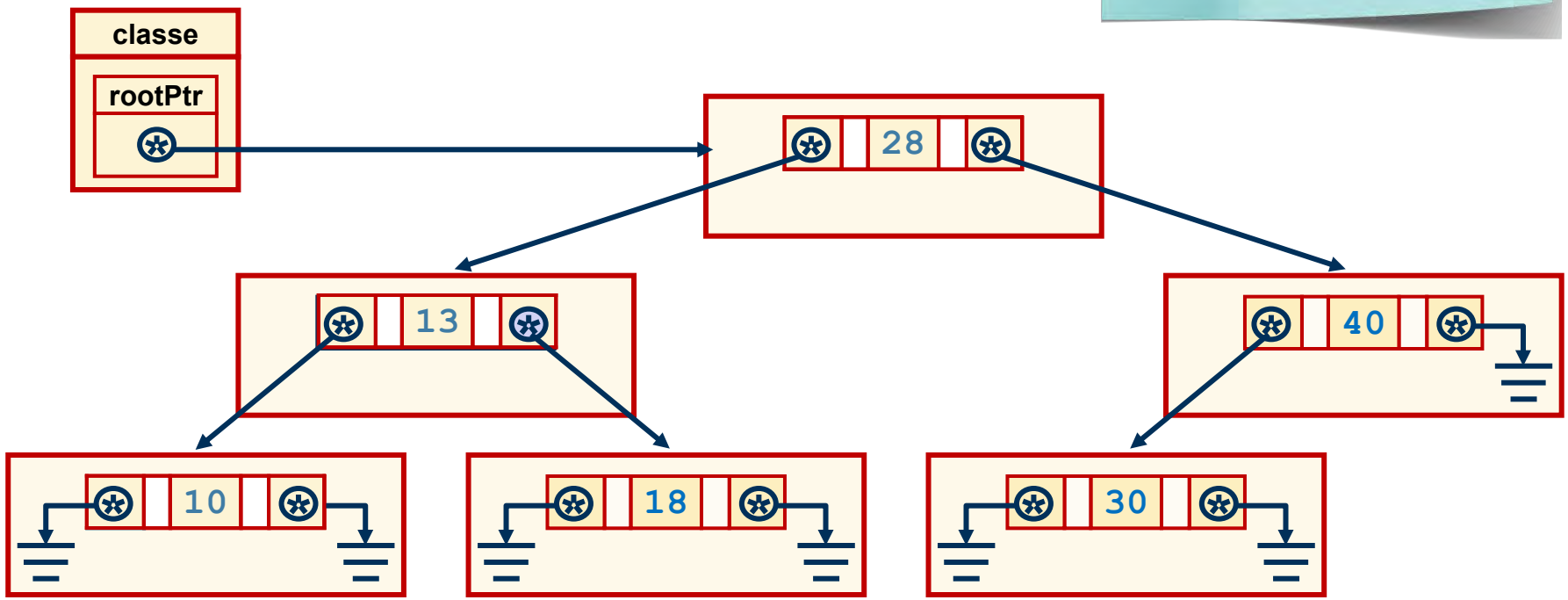


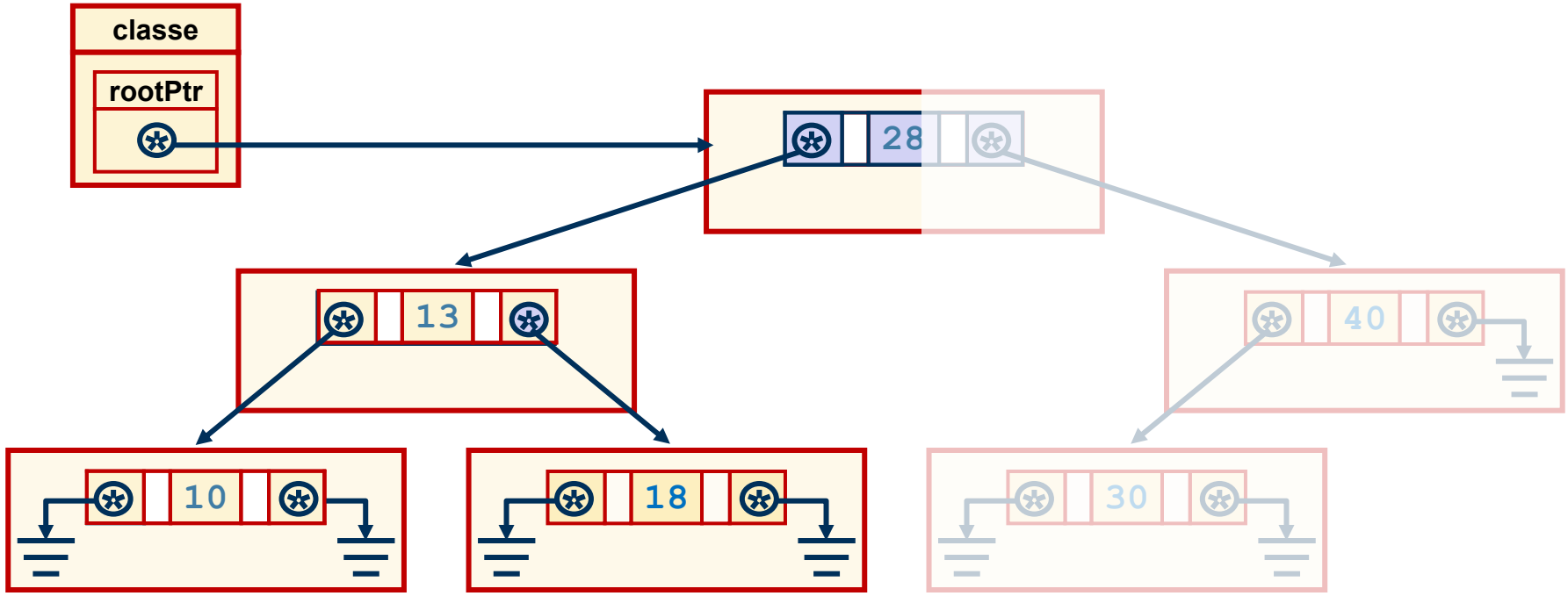



classe
rootPtr
28
13
40
10
18
30

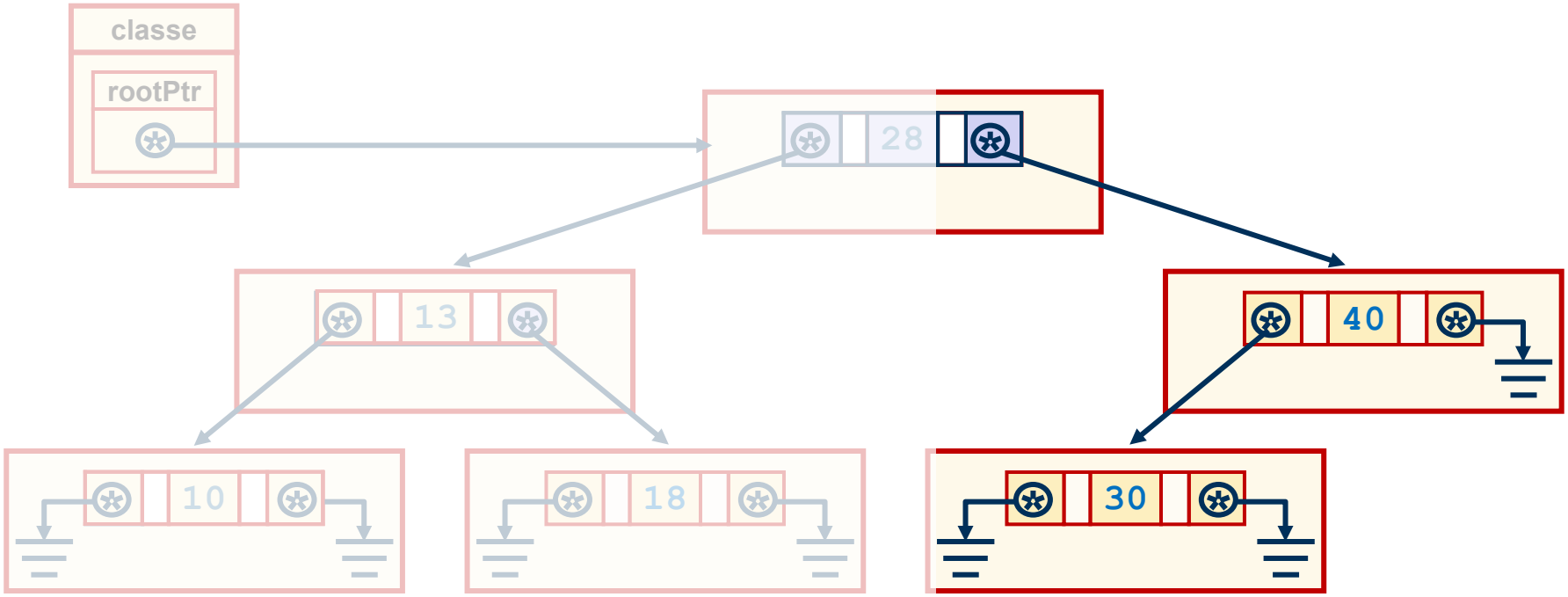


Funzione ricorsiva

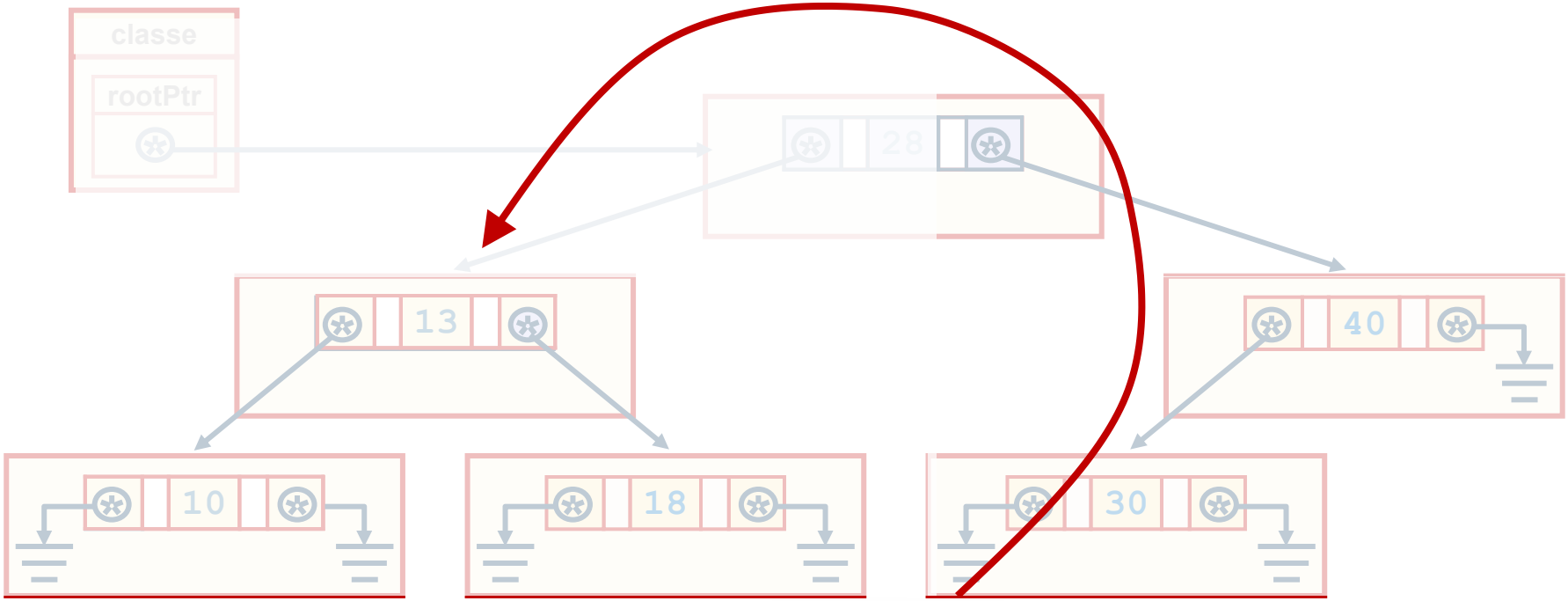




a matricola inferiore



a matricola superiore



sottoalbero sinistro → *radice* → *sottoalbero destro*

```
void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

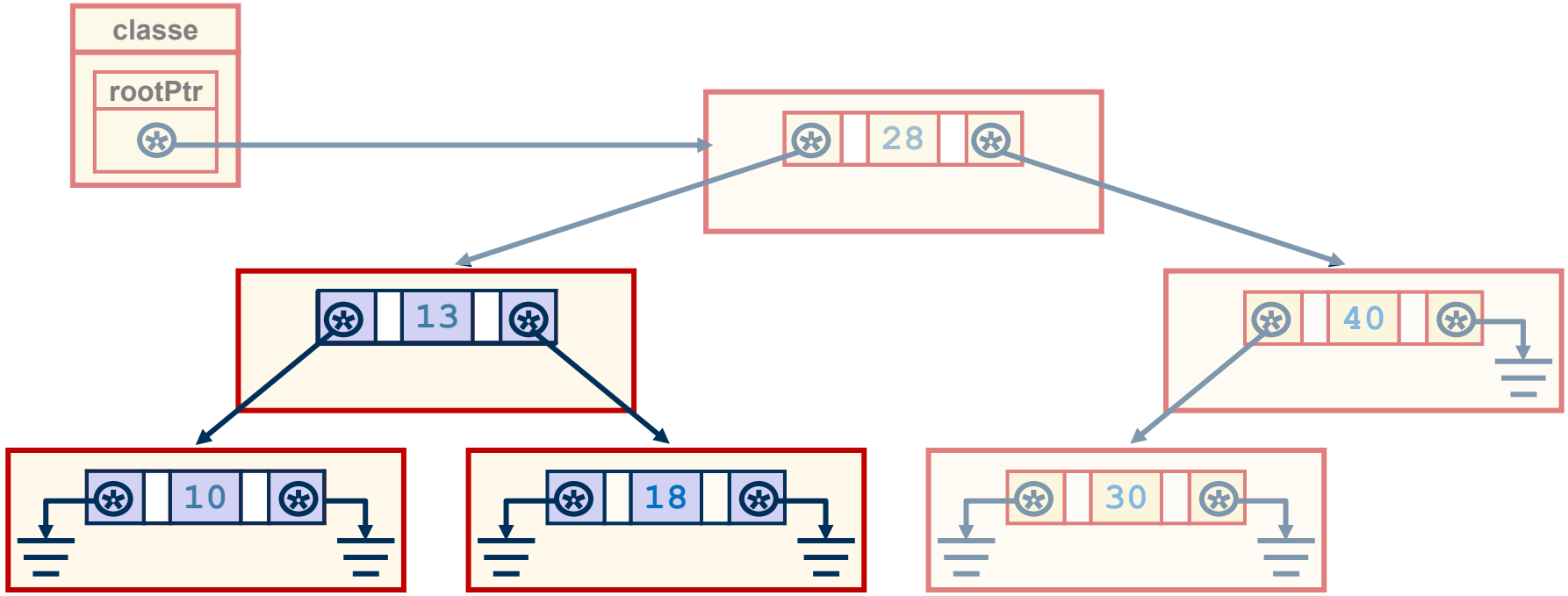
```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```


classe
rootPtr
28
13
40
10
18
30

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
void Tree :: recursiveInOrder(TreeNode *ptr)
{ ...
  recursiveInOrder(ptr->leftPtr);
  ...
}
```

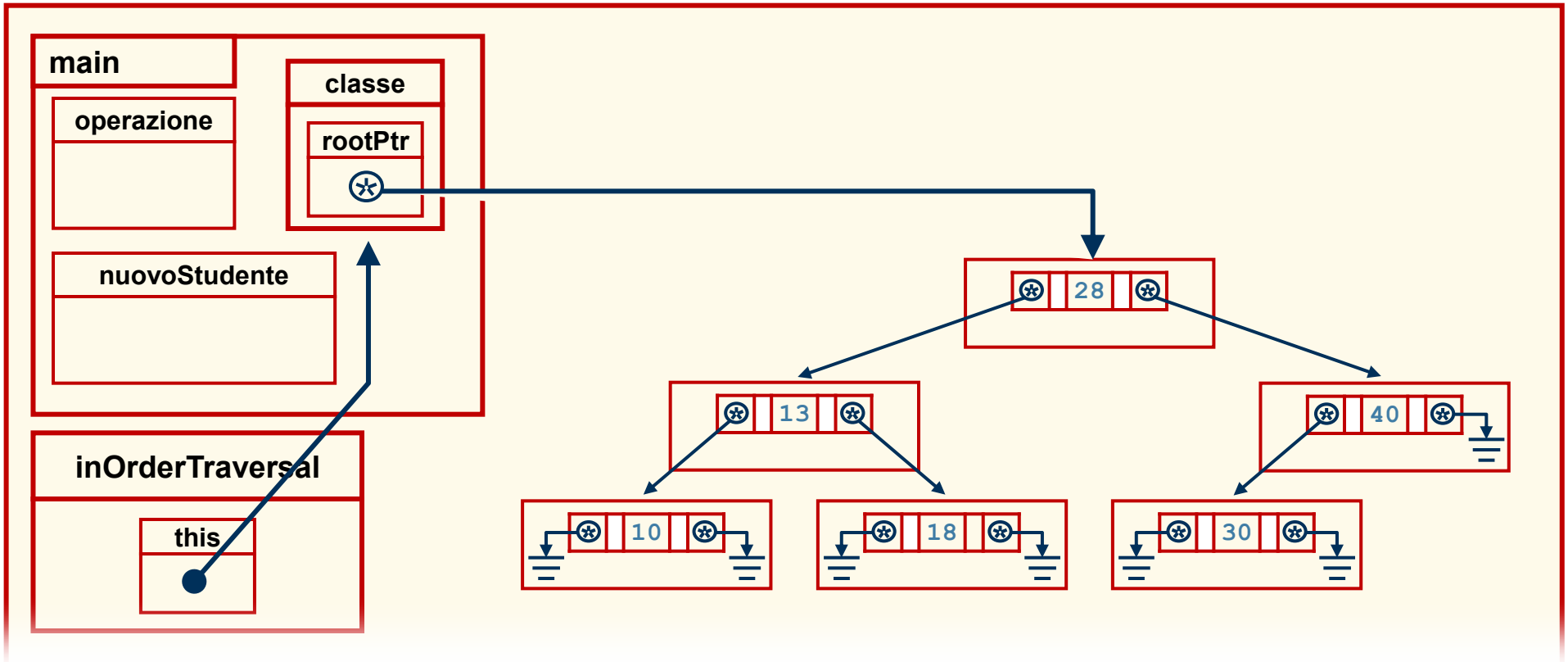
```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{ ...
  recursiveInOrder(ptr->leftPtr);
  ...
}
```

```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

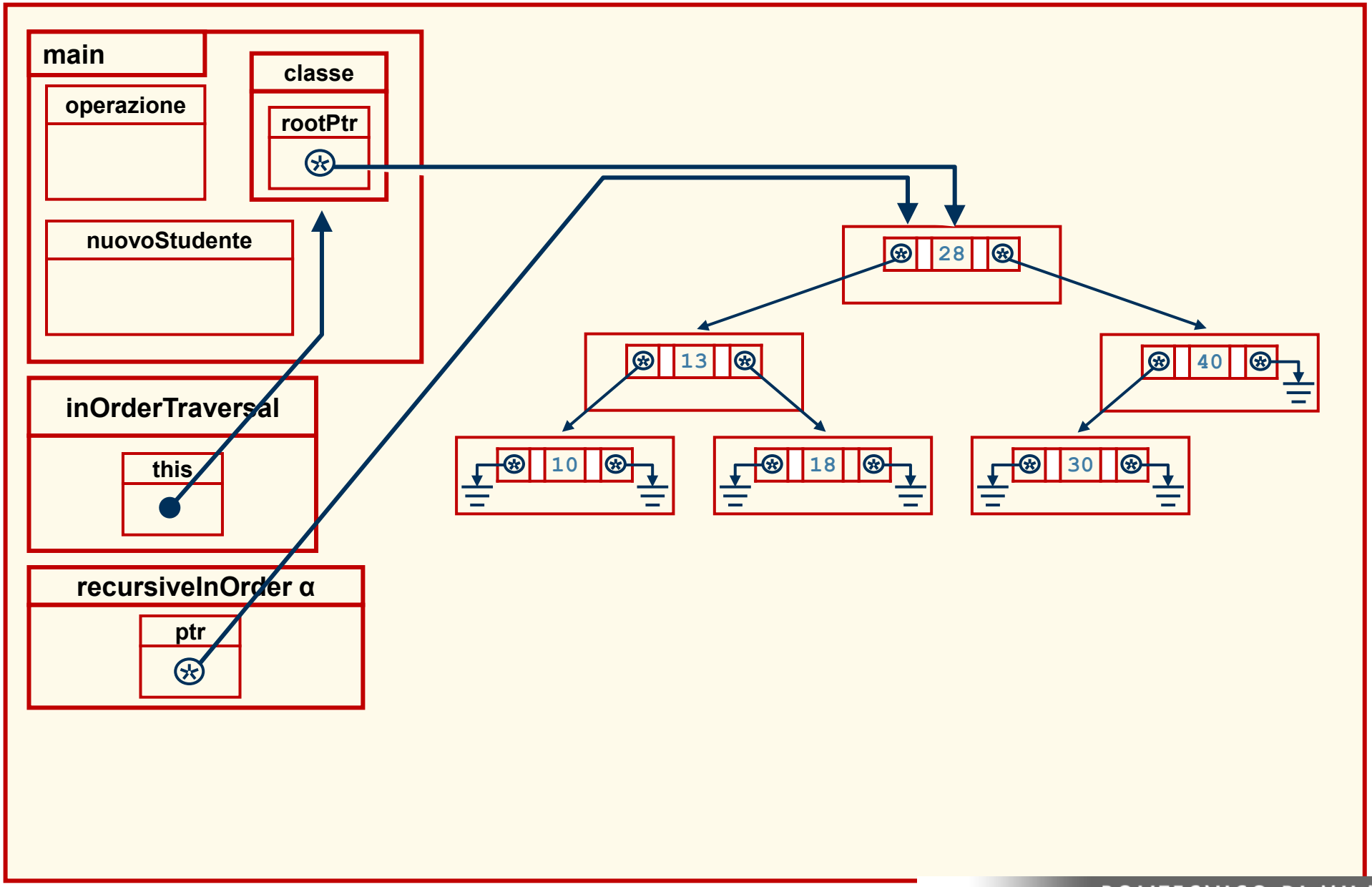
Doppia chiamata ricorsiva

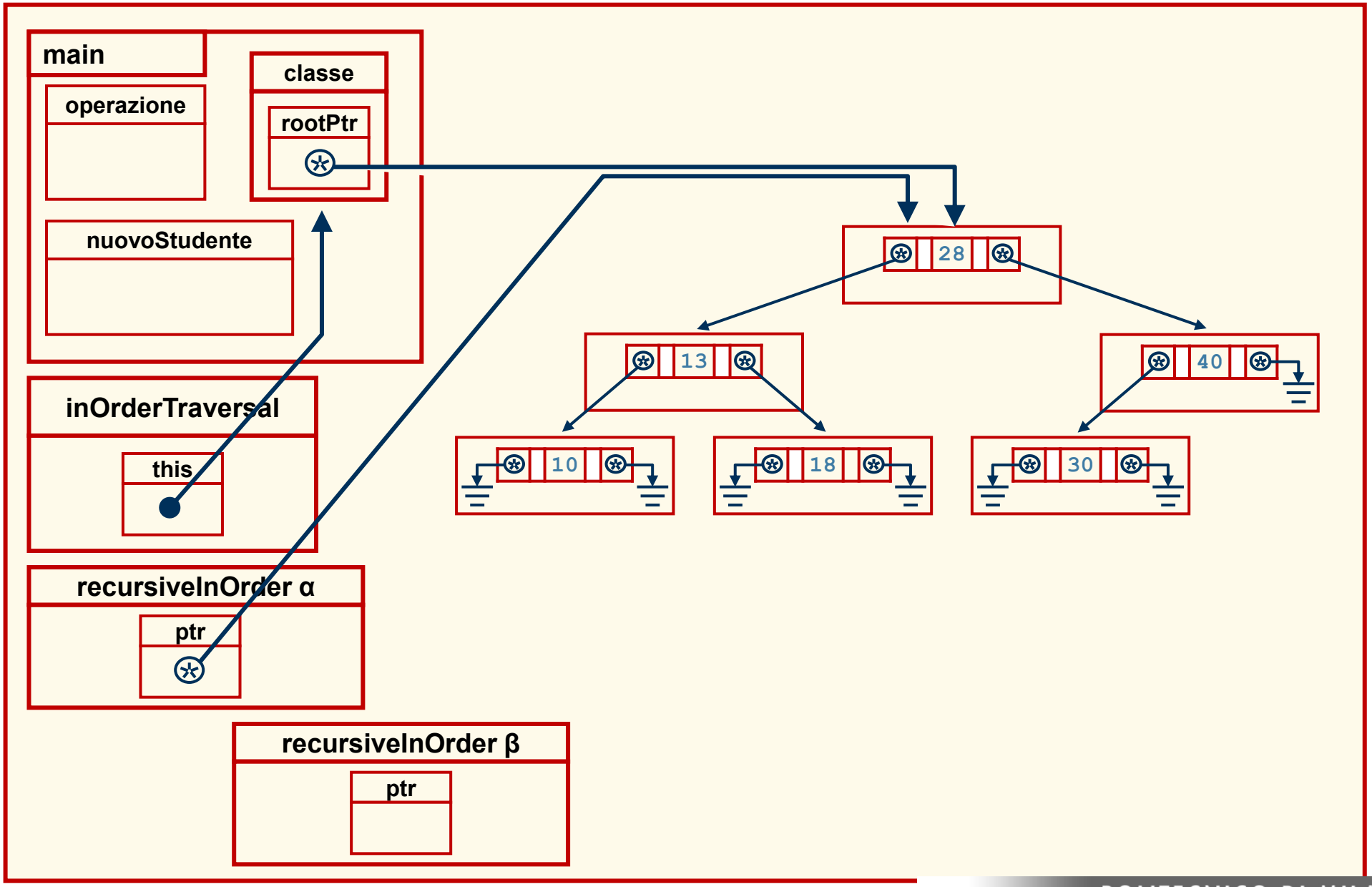
```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

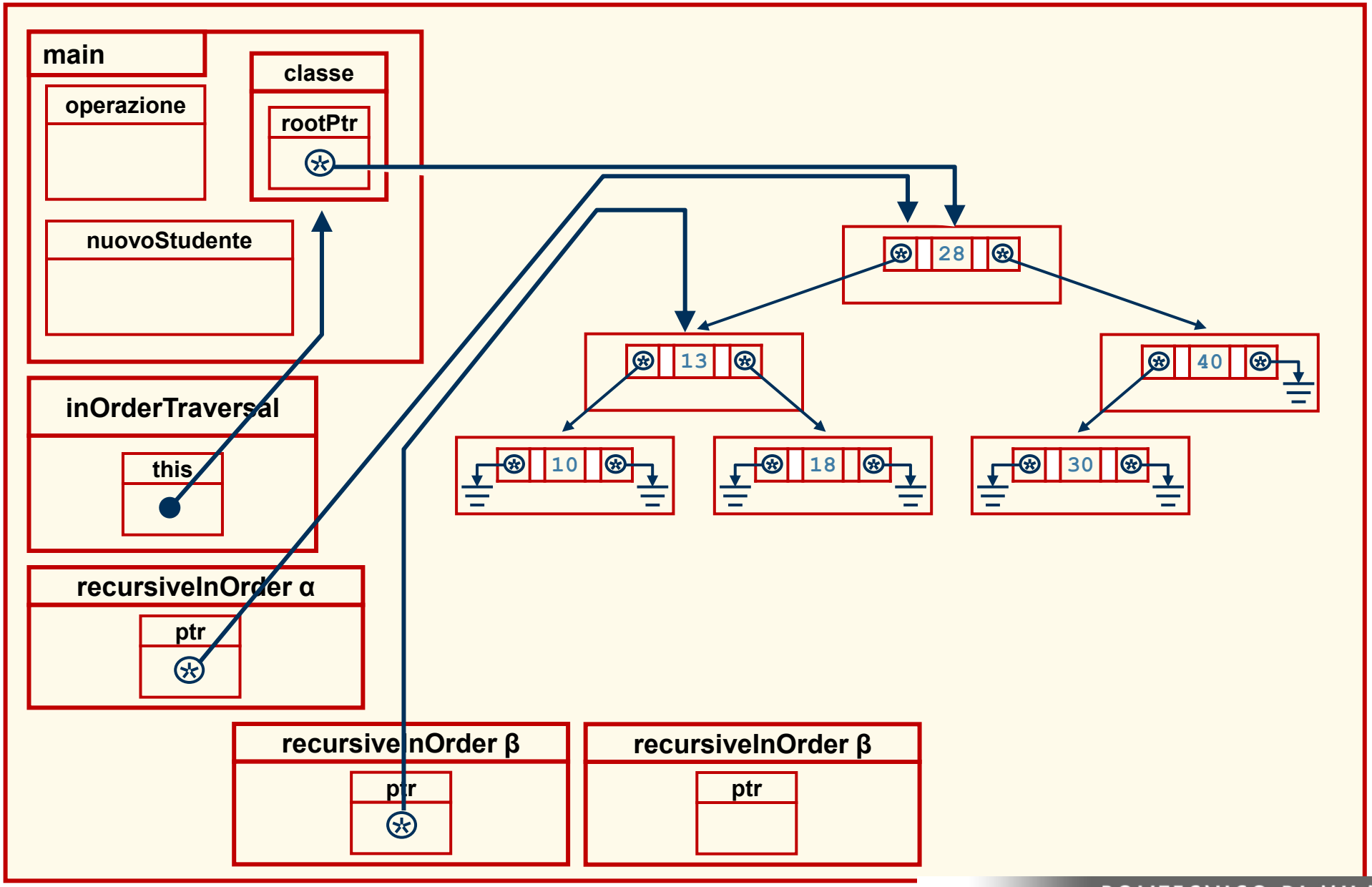
```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{
  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

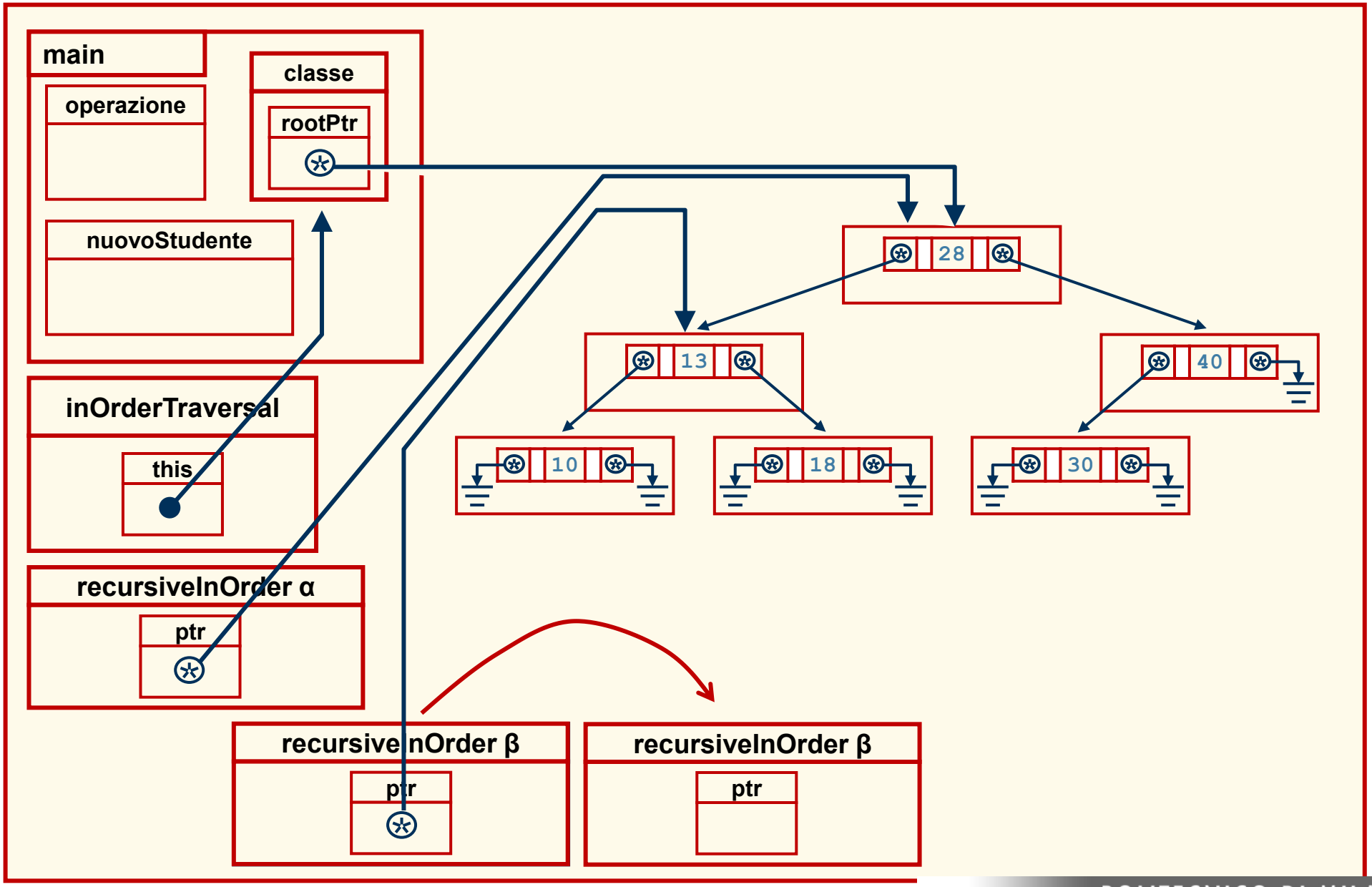


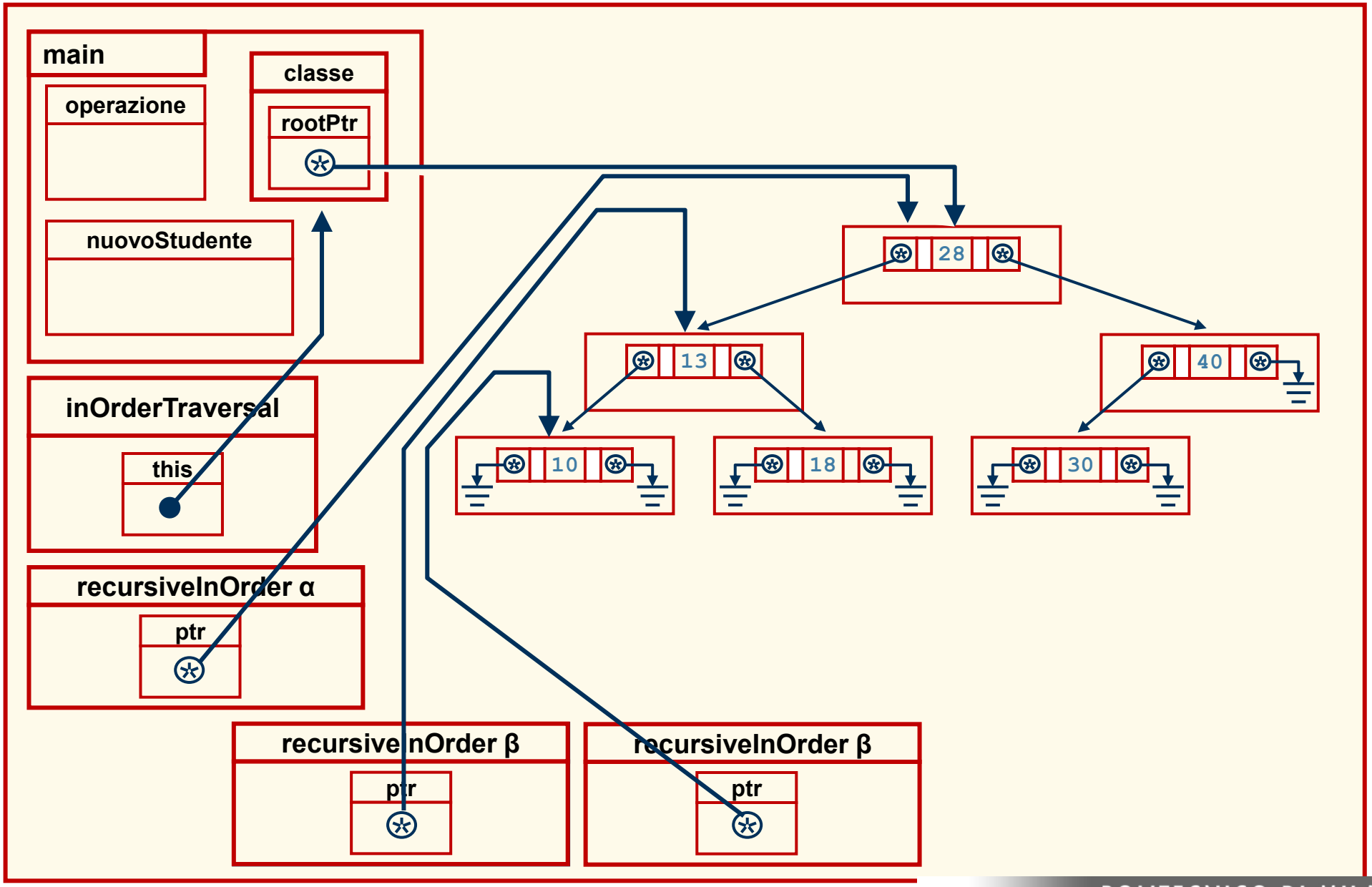

```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);        α
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);        β
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);       γ
      }
}
```

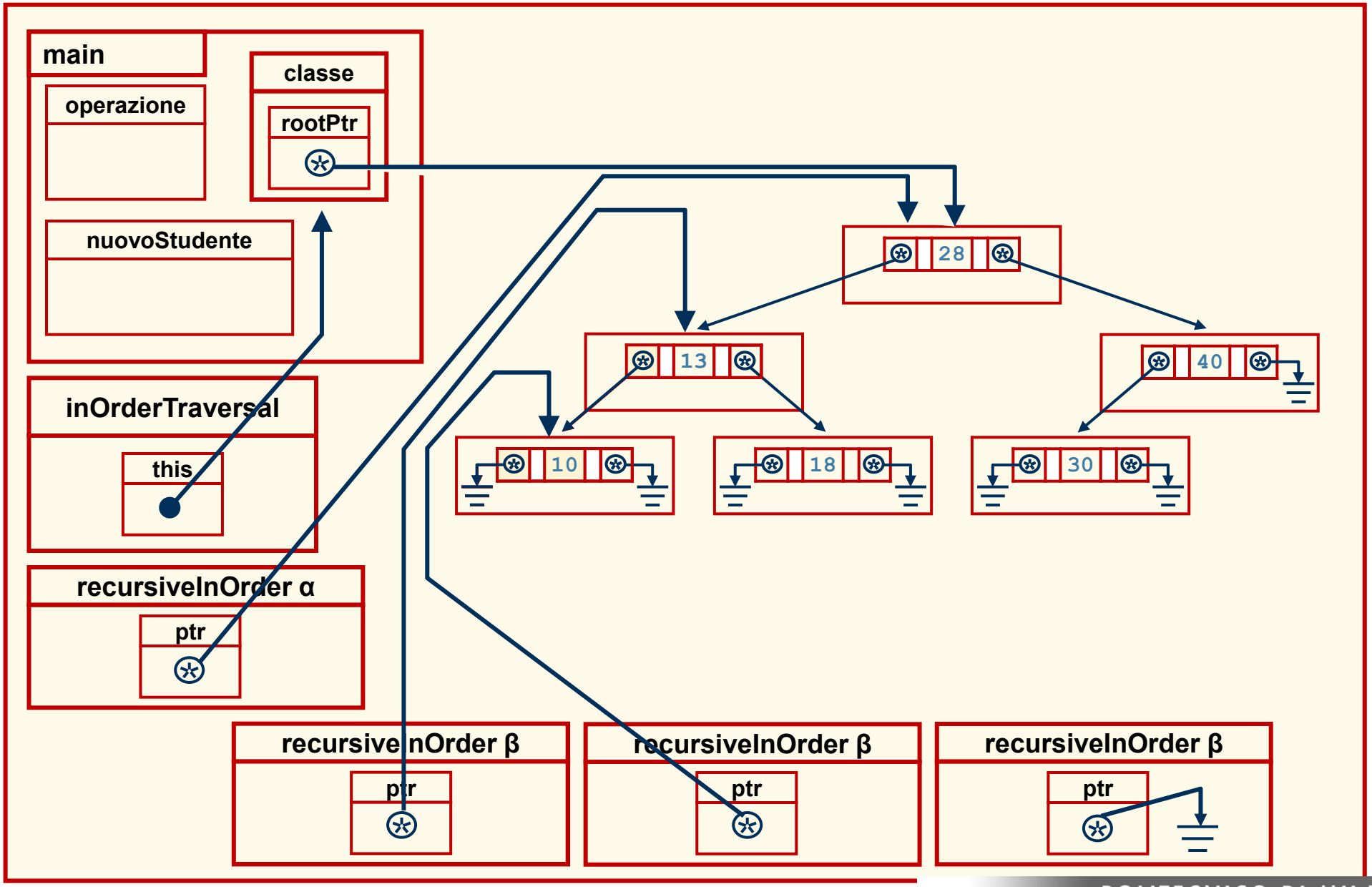

main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
28
13
40
10
18
30

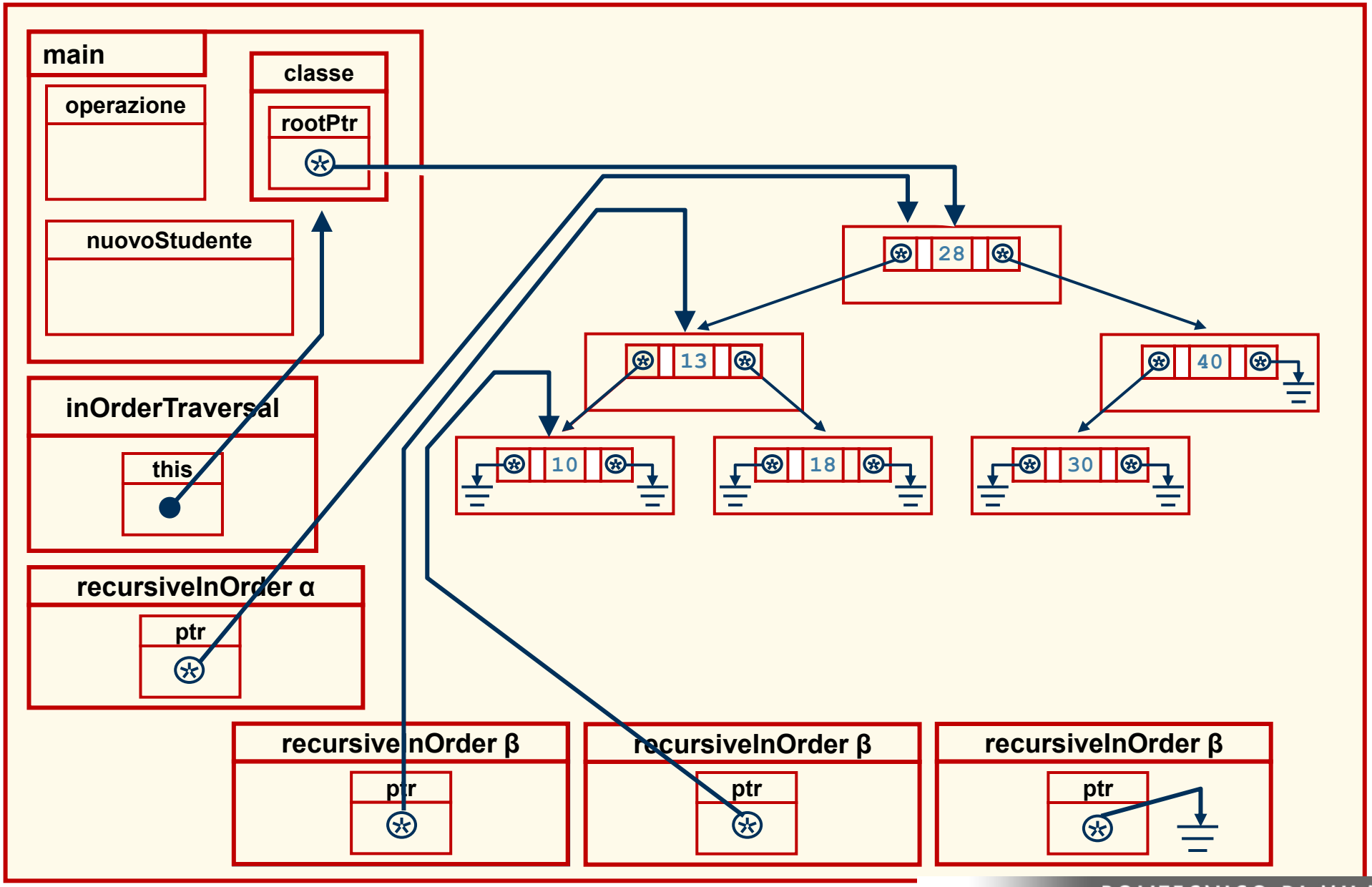

main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
28
13
40
10
18
30

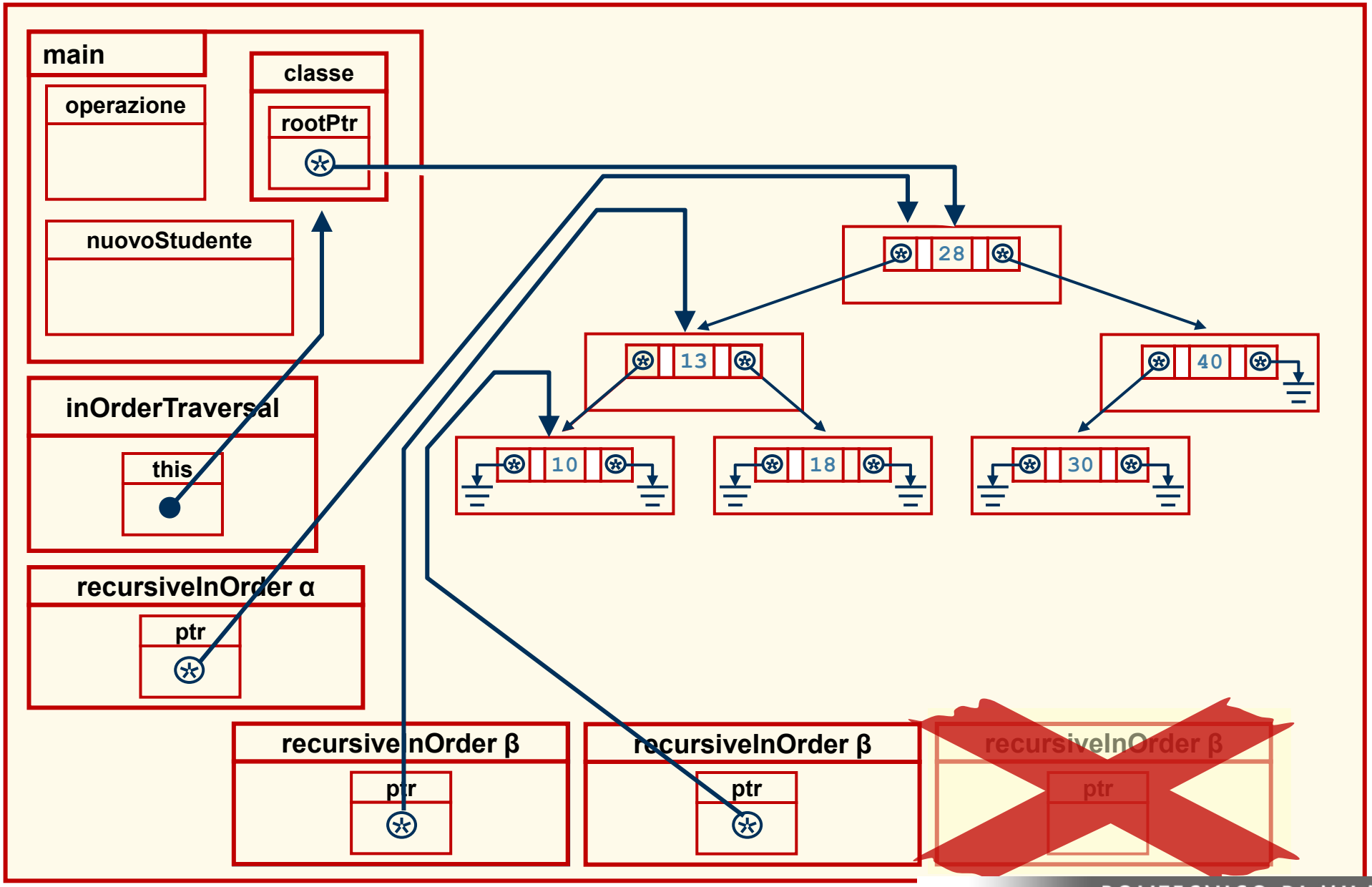

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

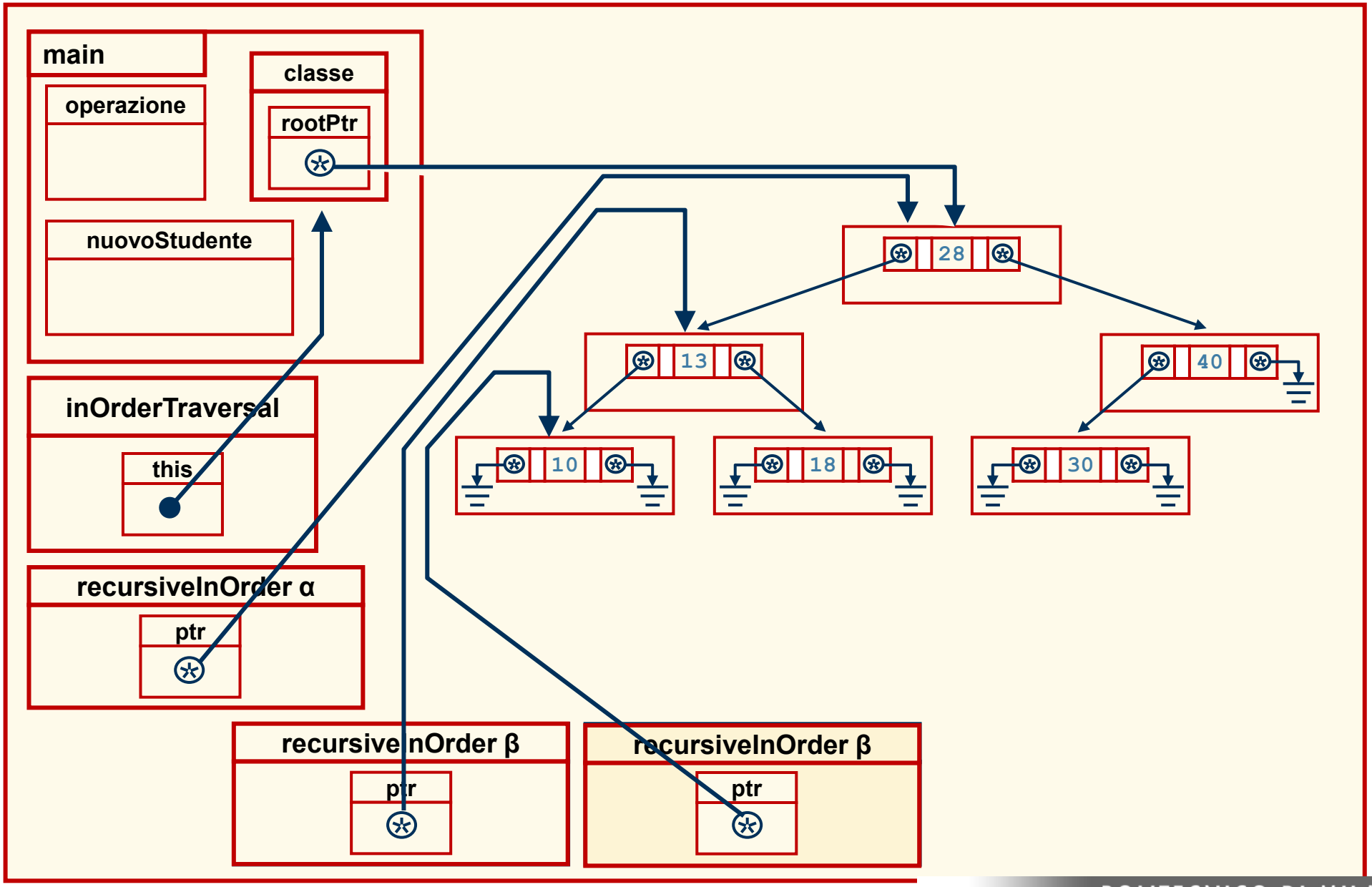

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

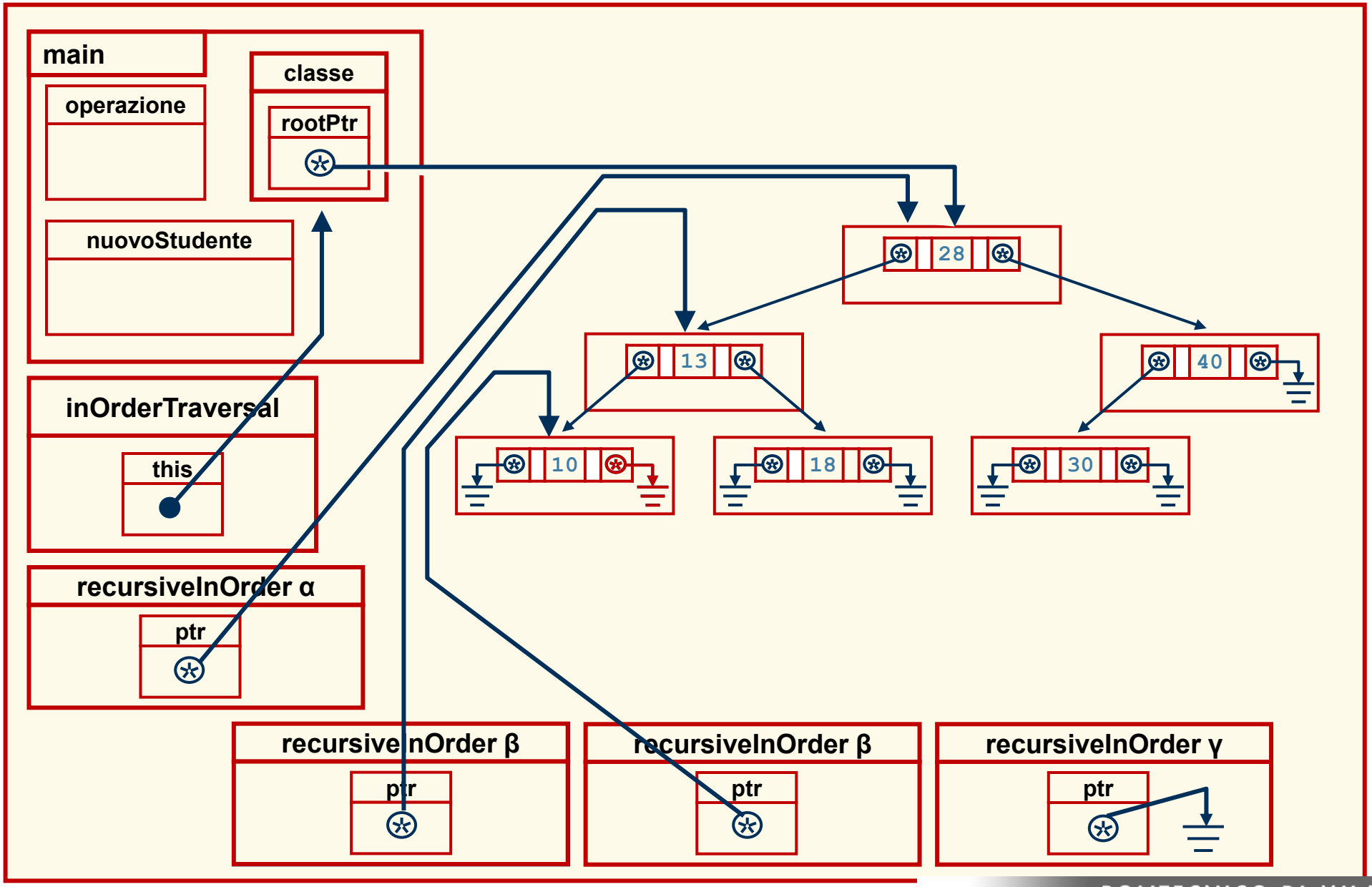

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

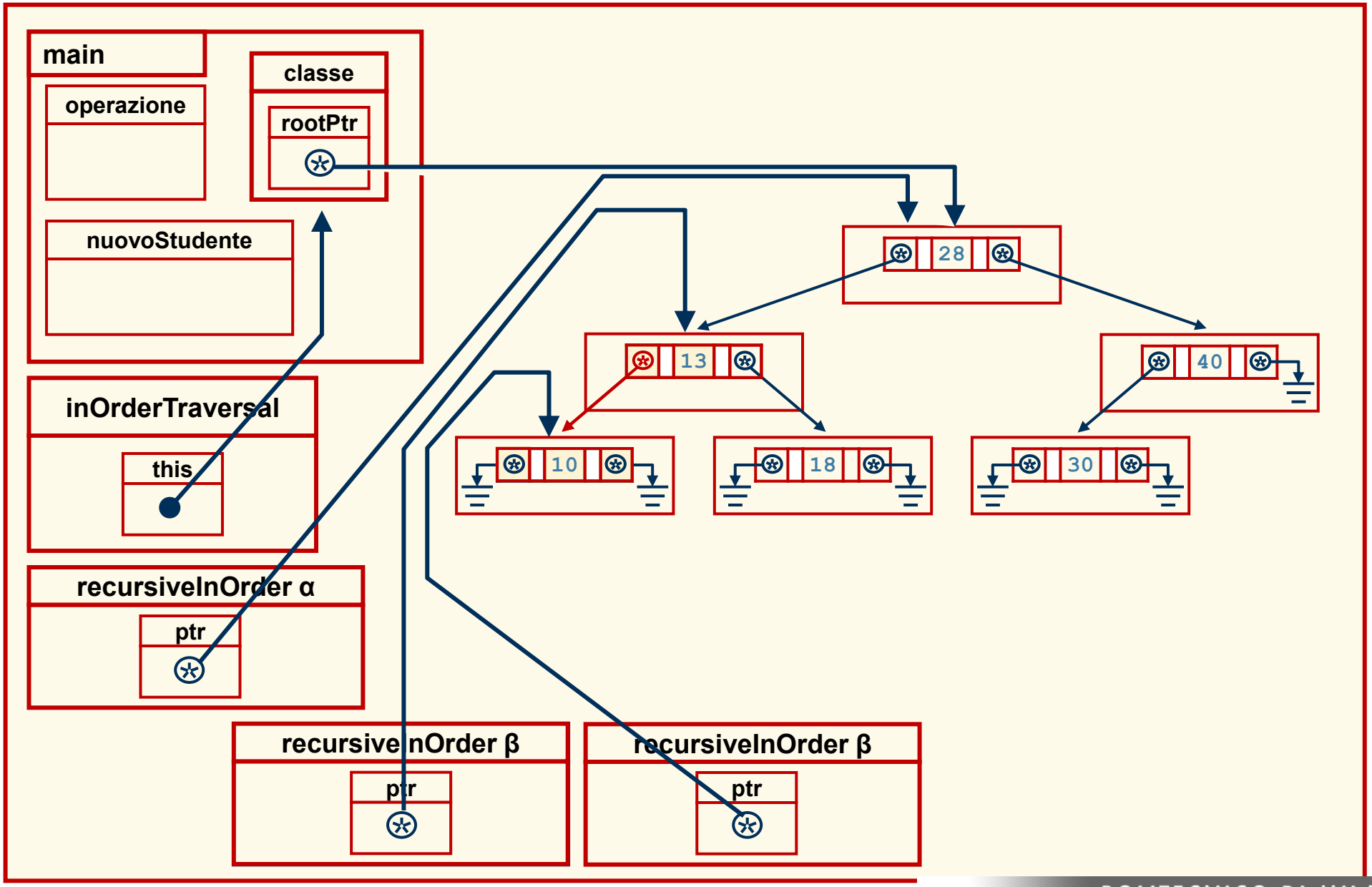

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

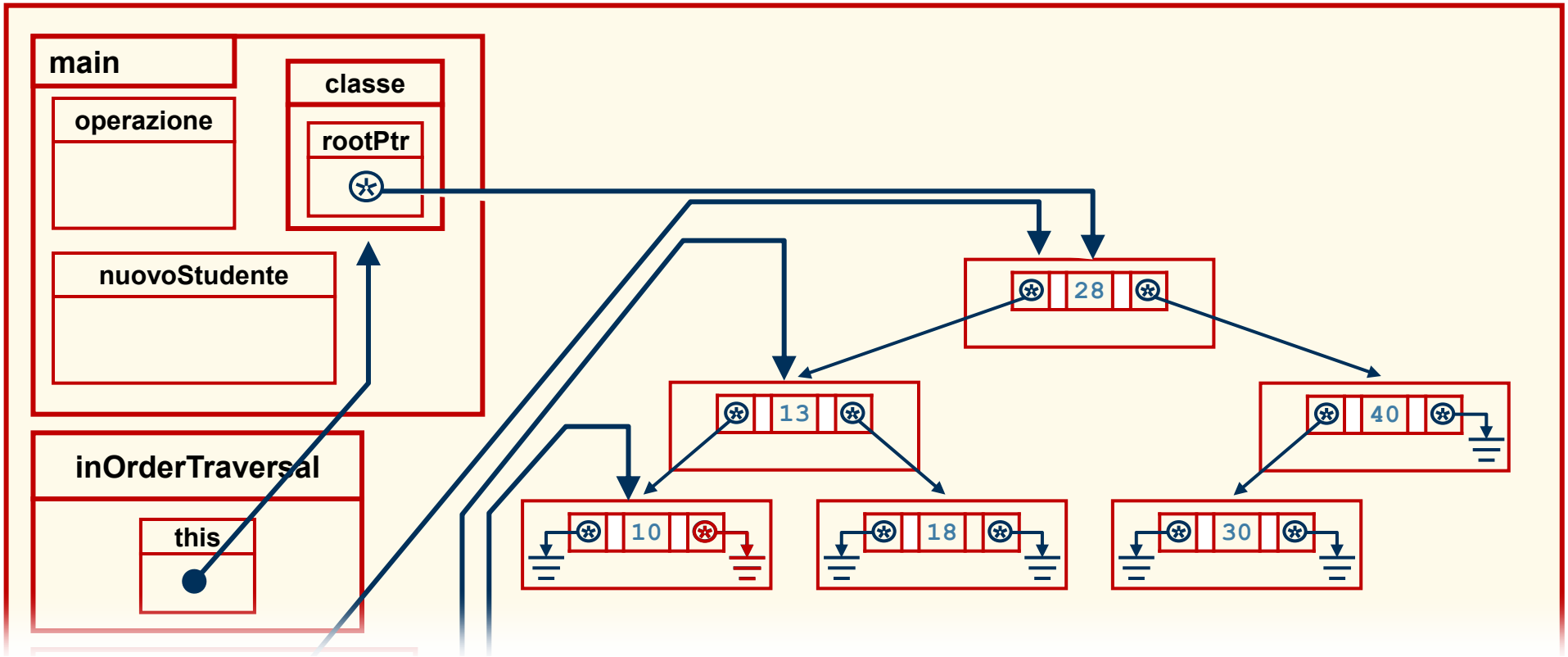

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr


main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

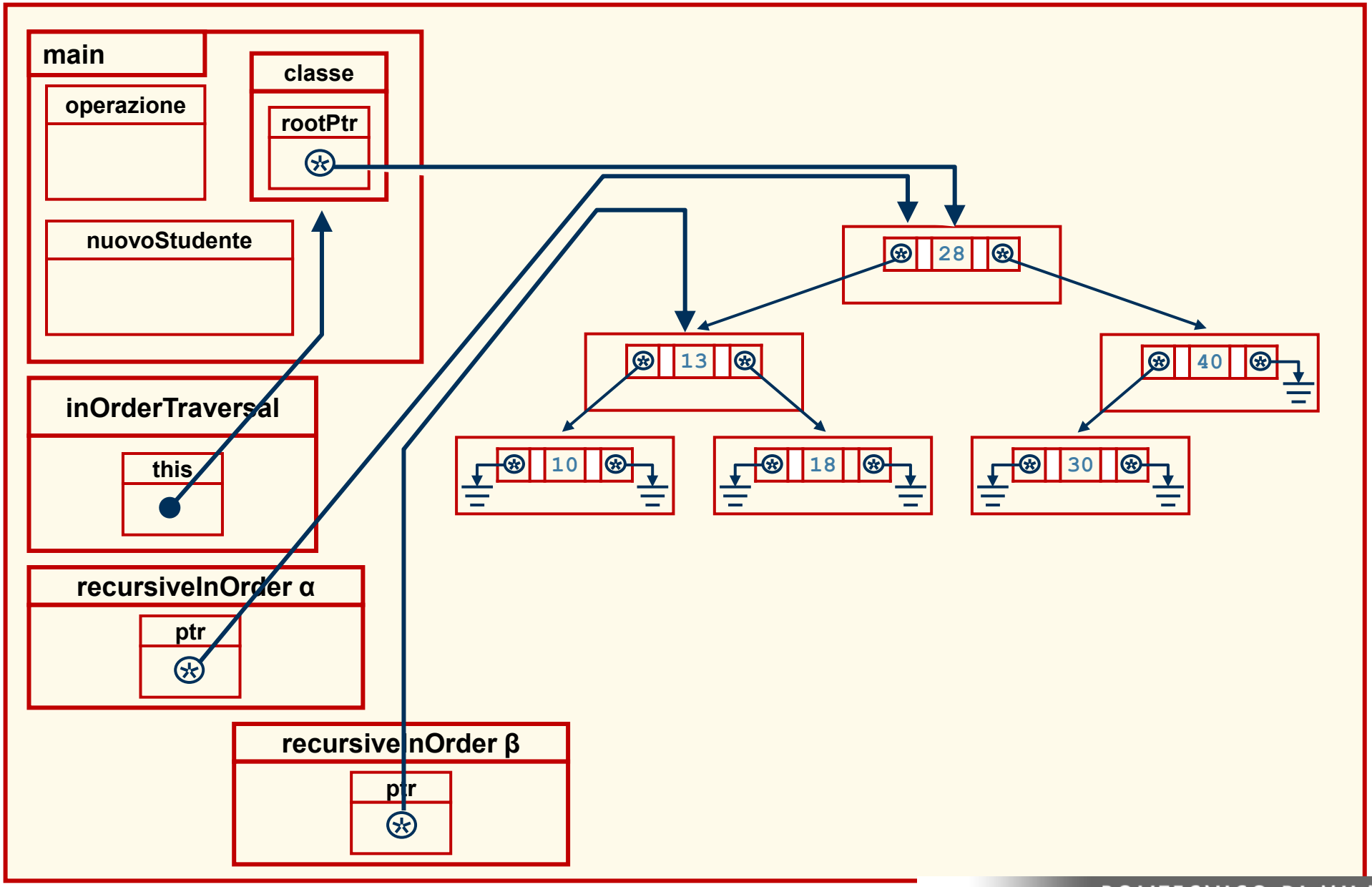

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

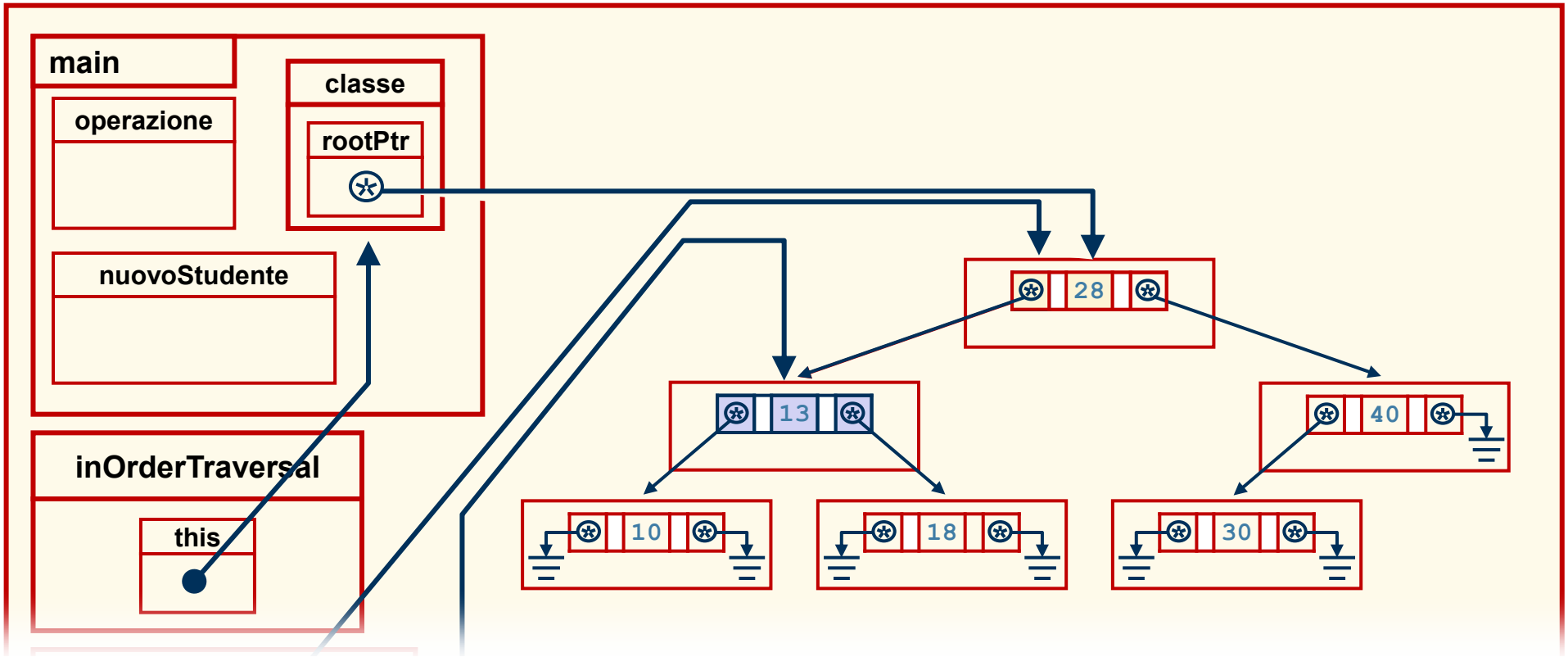

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr
recursiveInOrder γ
ptr


main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder β
ptr

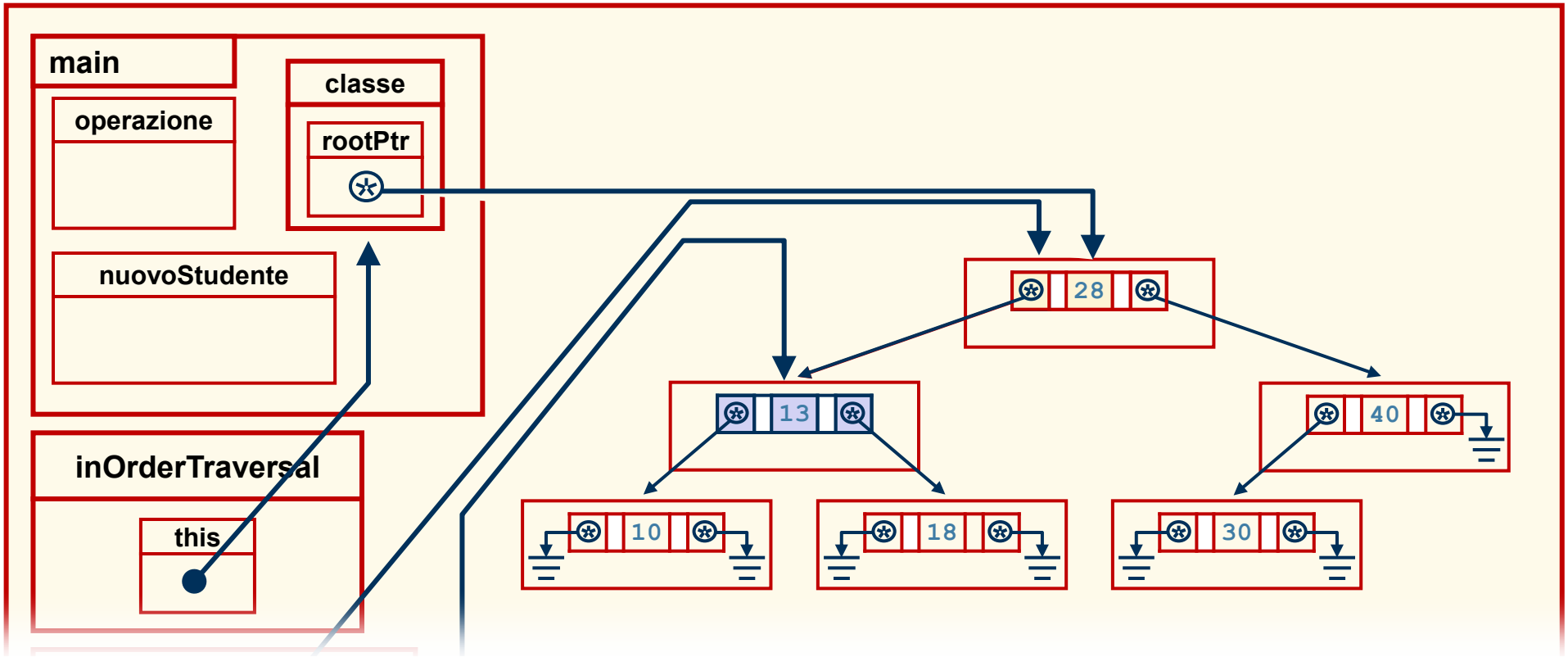



```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);      α
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);      β
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);      γ
      }
}
```

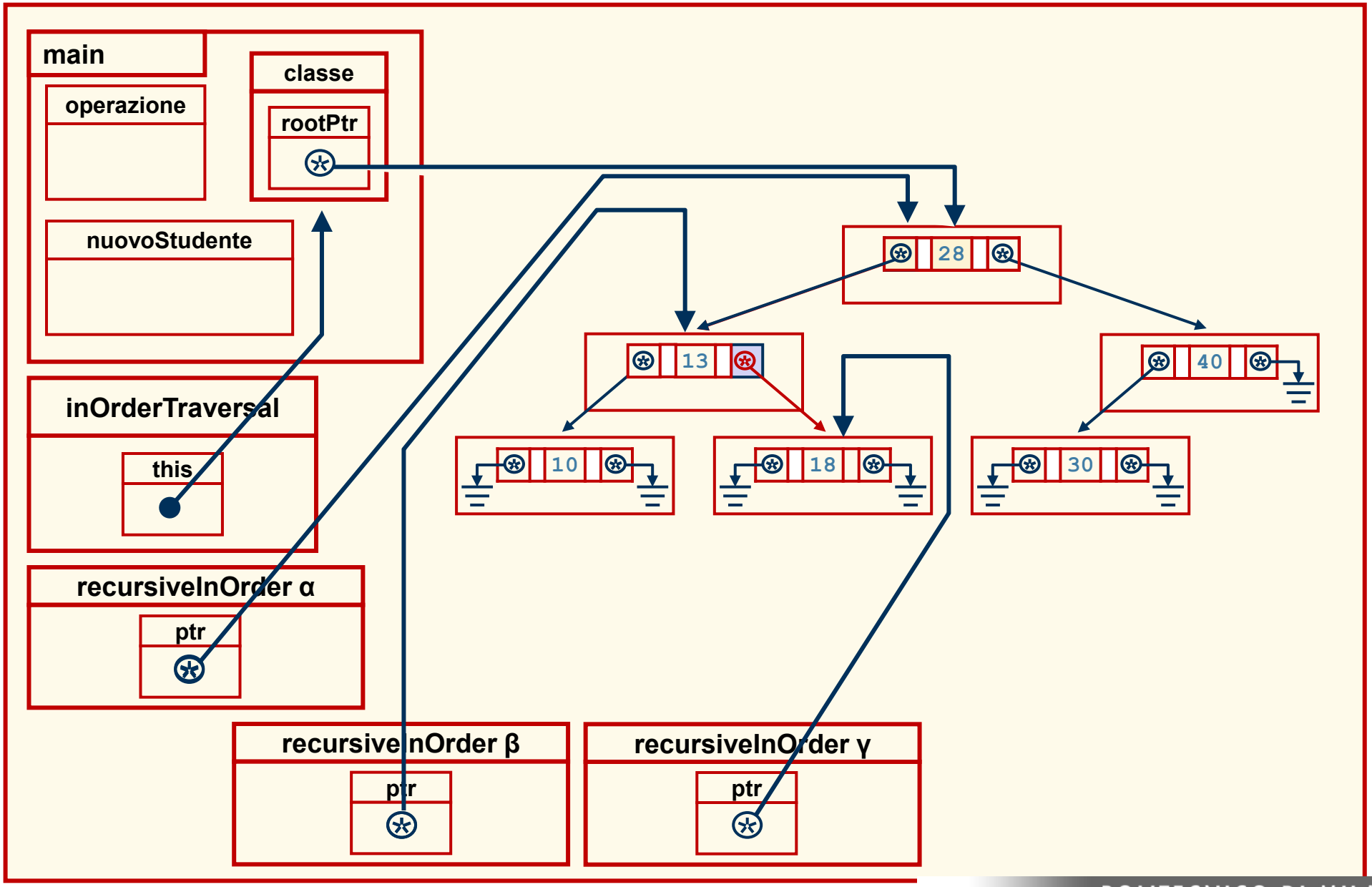

main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
28
13
40
10
18
30

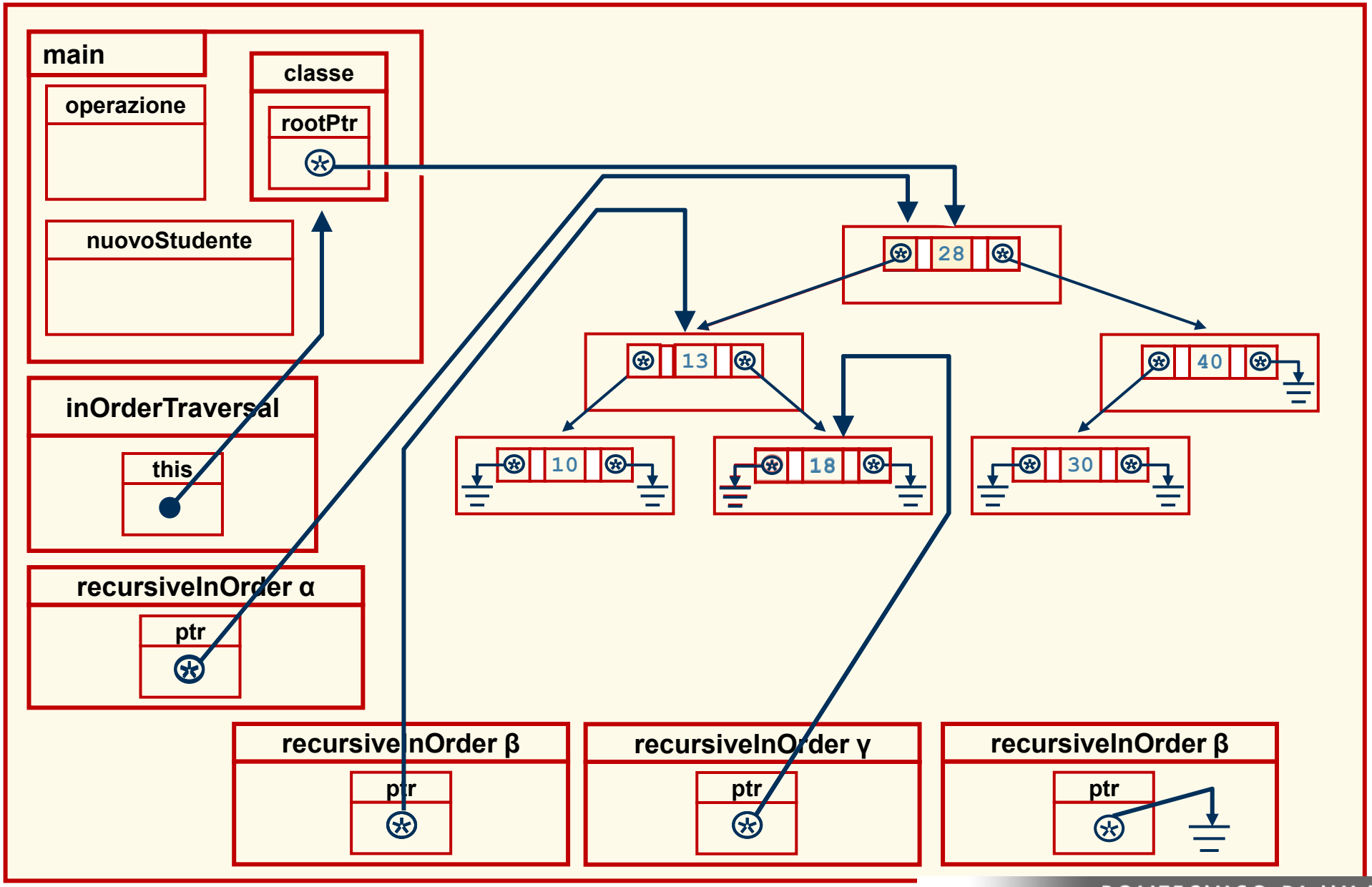


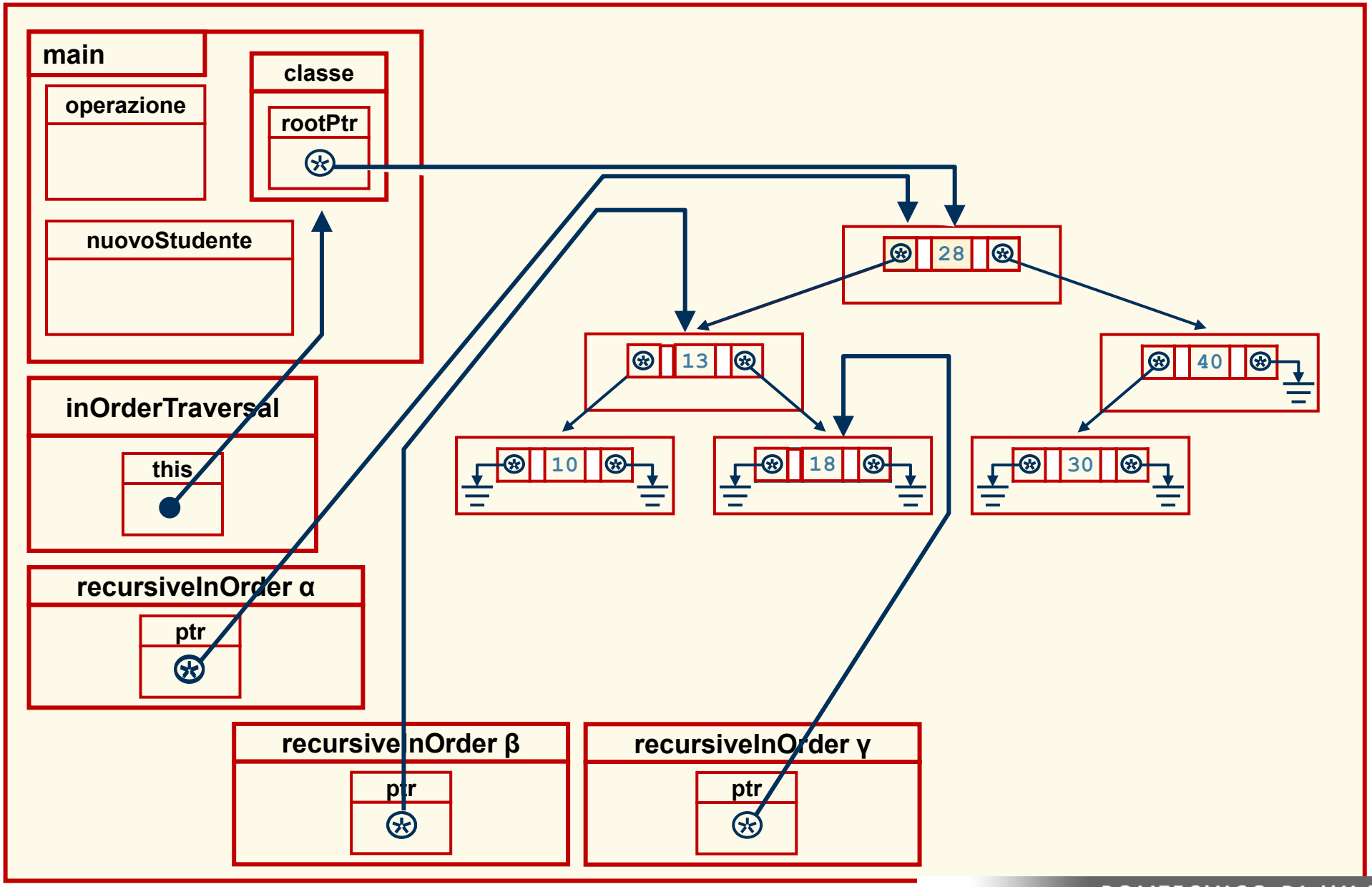

```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);      α
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);      β
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);      γ
      }
}
```

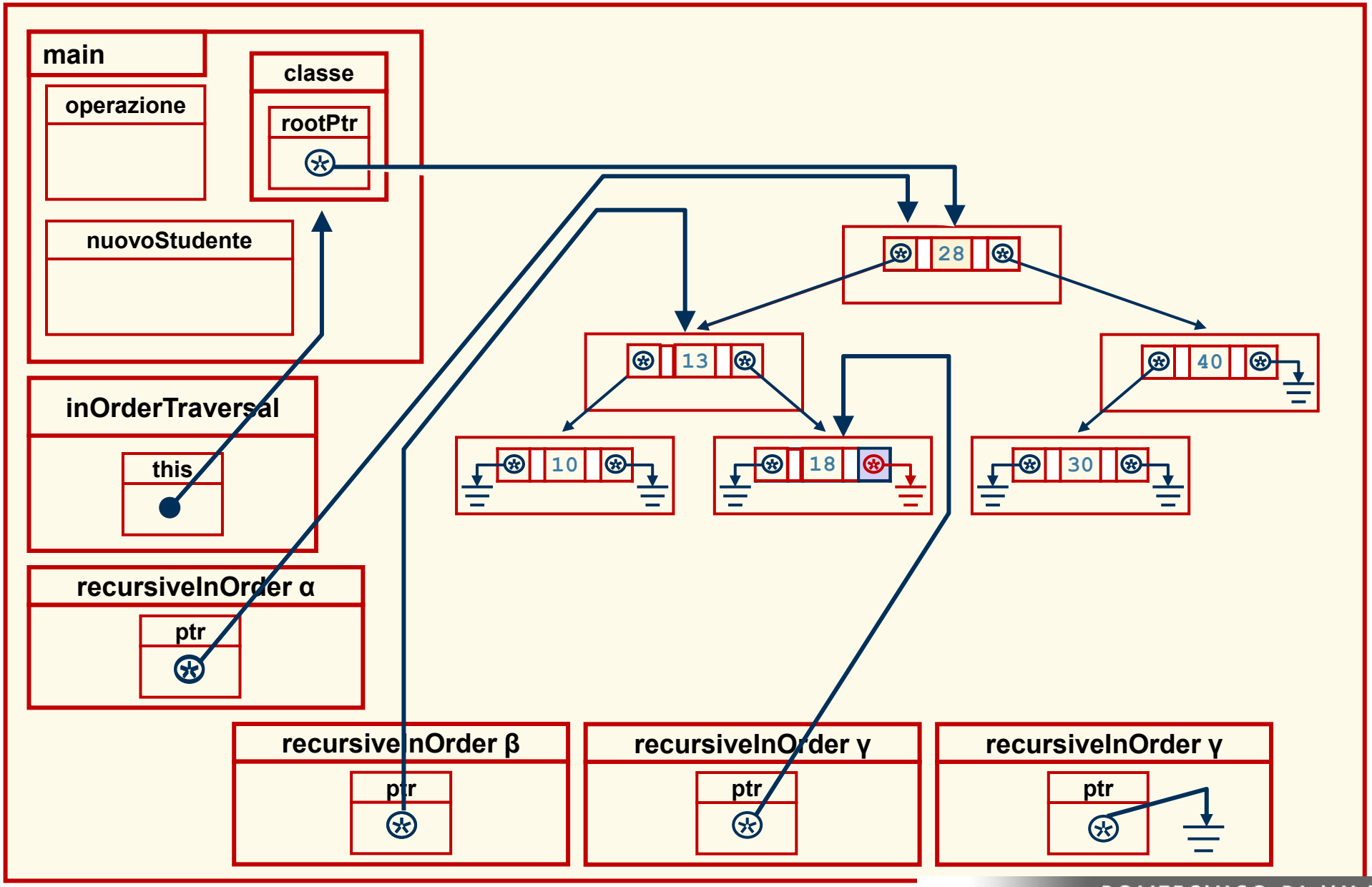


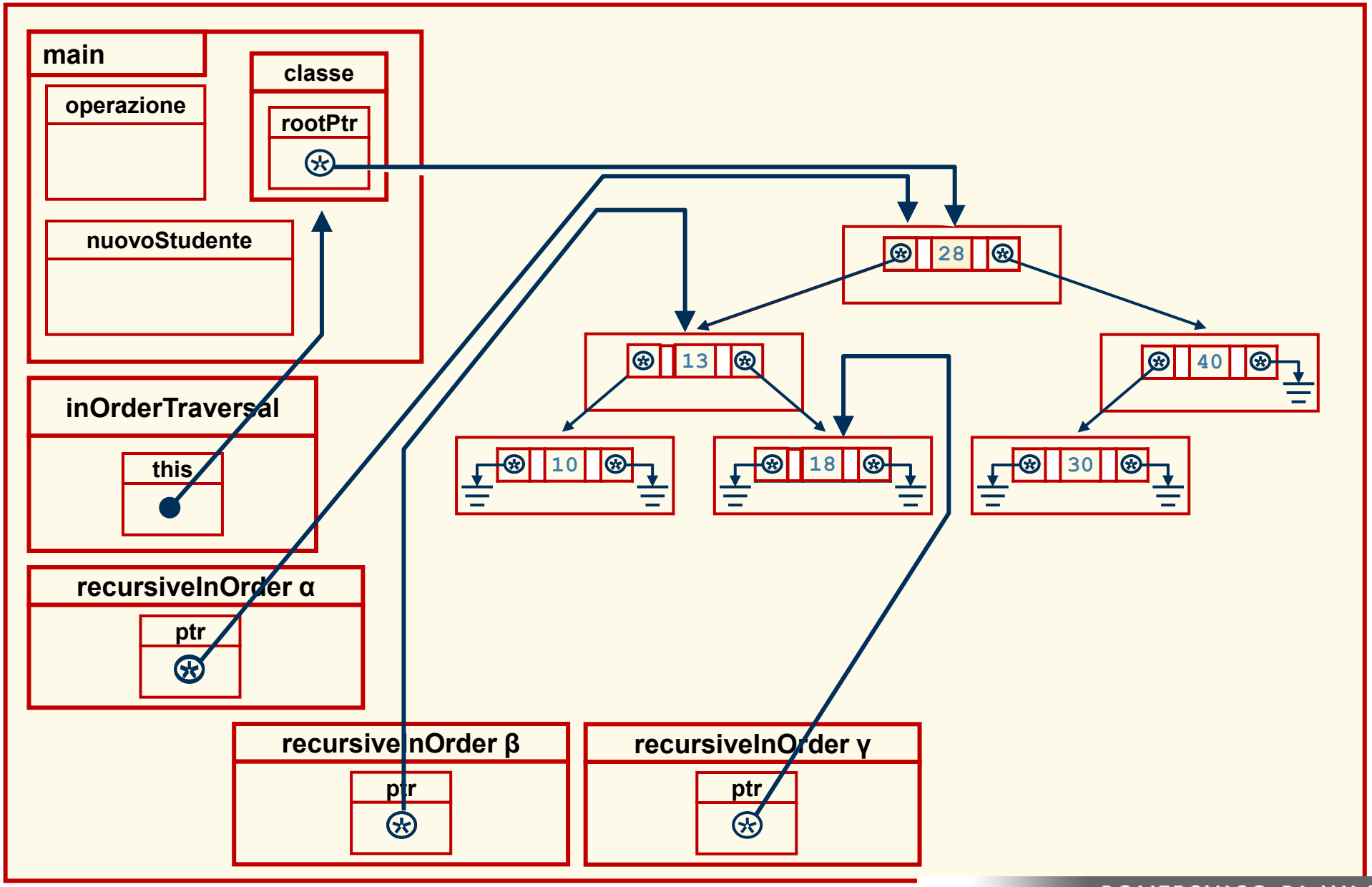

```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);      α
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);      β
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);      γ
      }
}
```

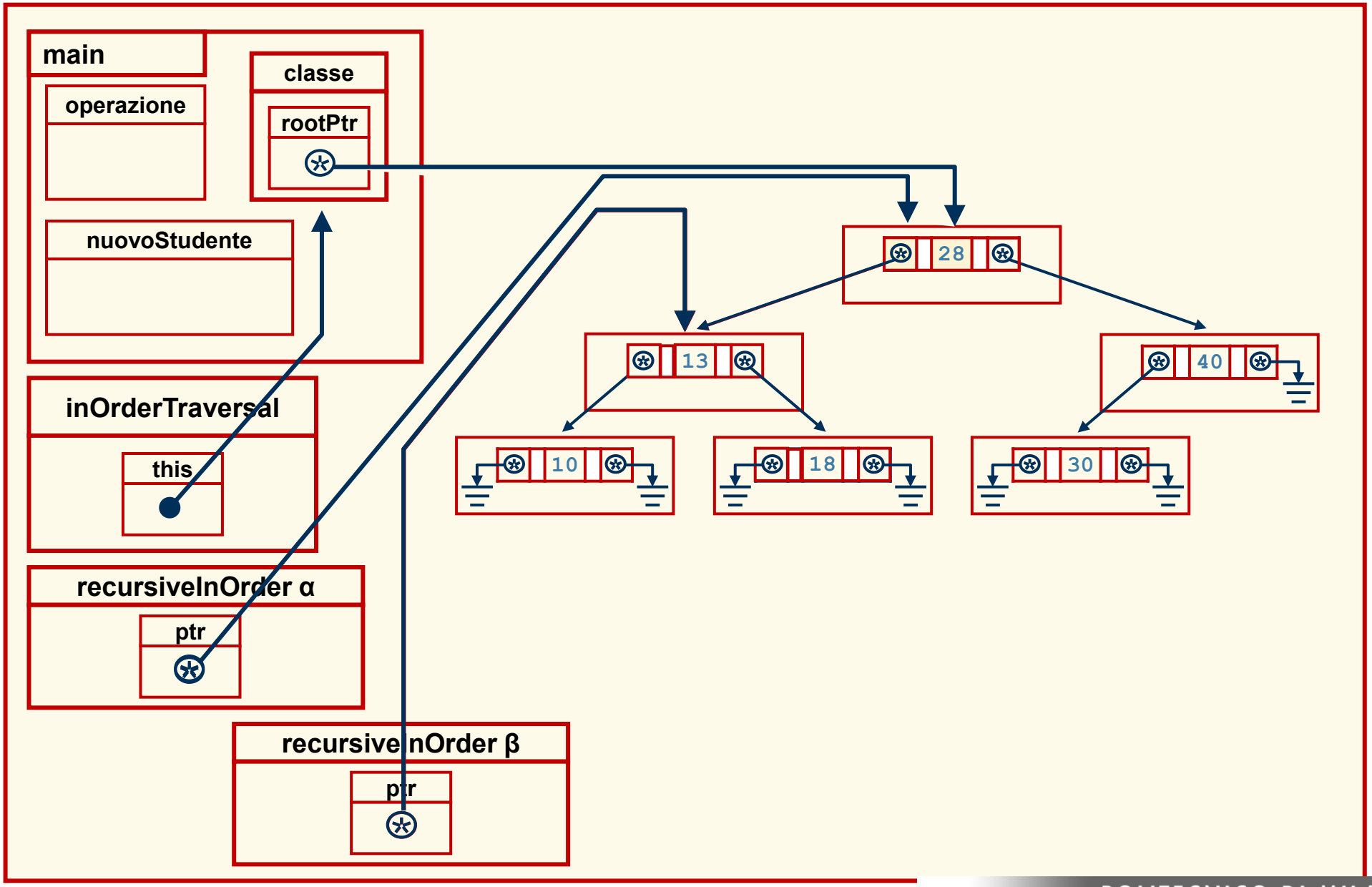

main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder γ
ptr
28
13
40
10
18
30

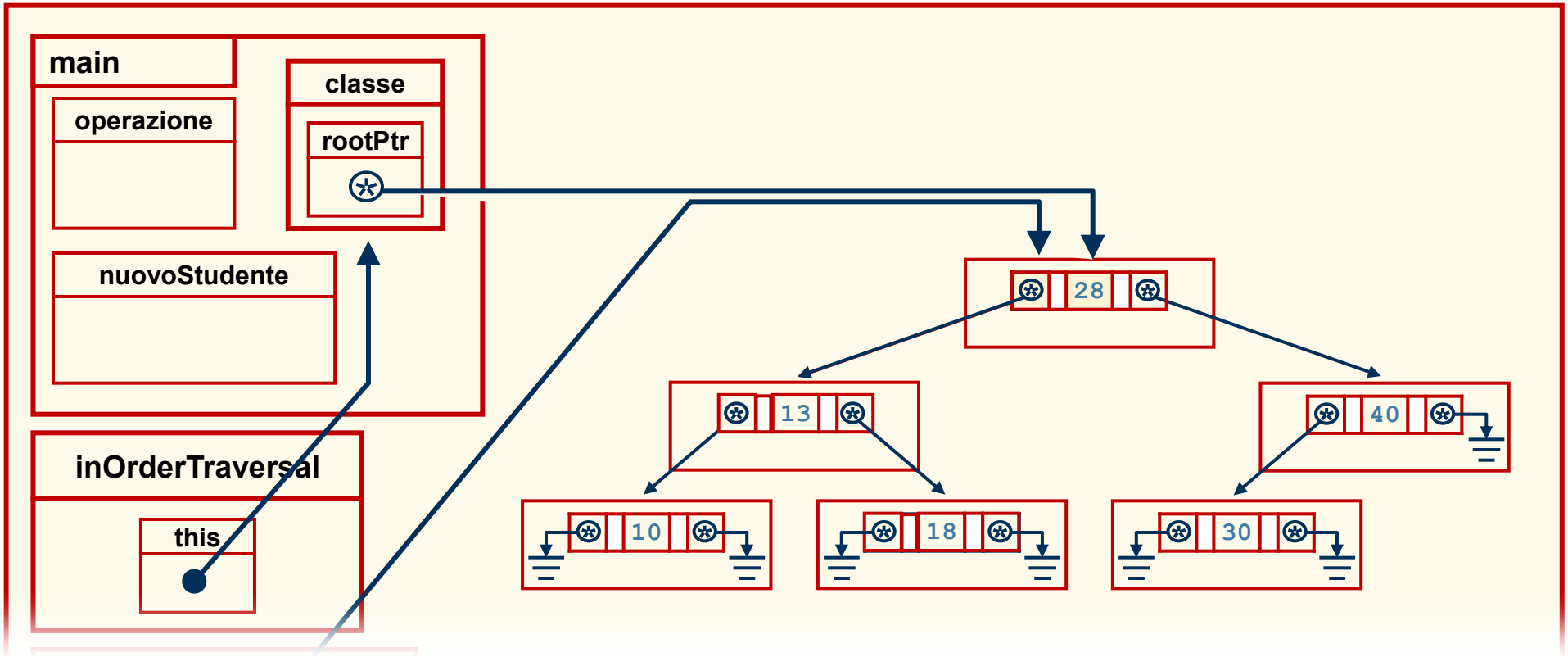

main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
28
13
40
10
18
30
recursiveInOrder β
ptr
recursiveInOrder γ
ptr
recursiveInOrder β
ptr


main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder γ
ptr
28
13
40
10
18
30


main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder γ
ptr
recursiveInOrder γ
ptr

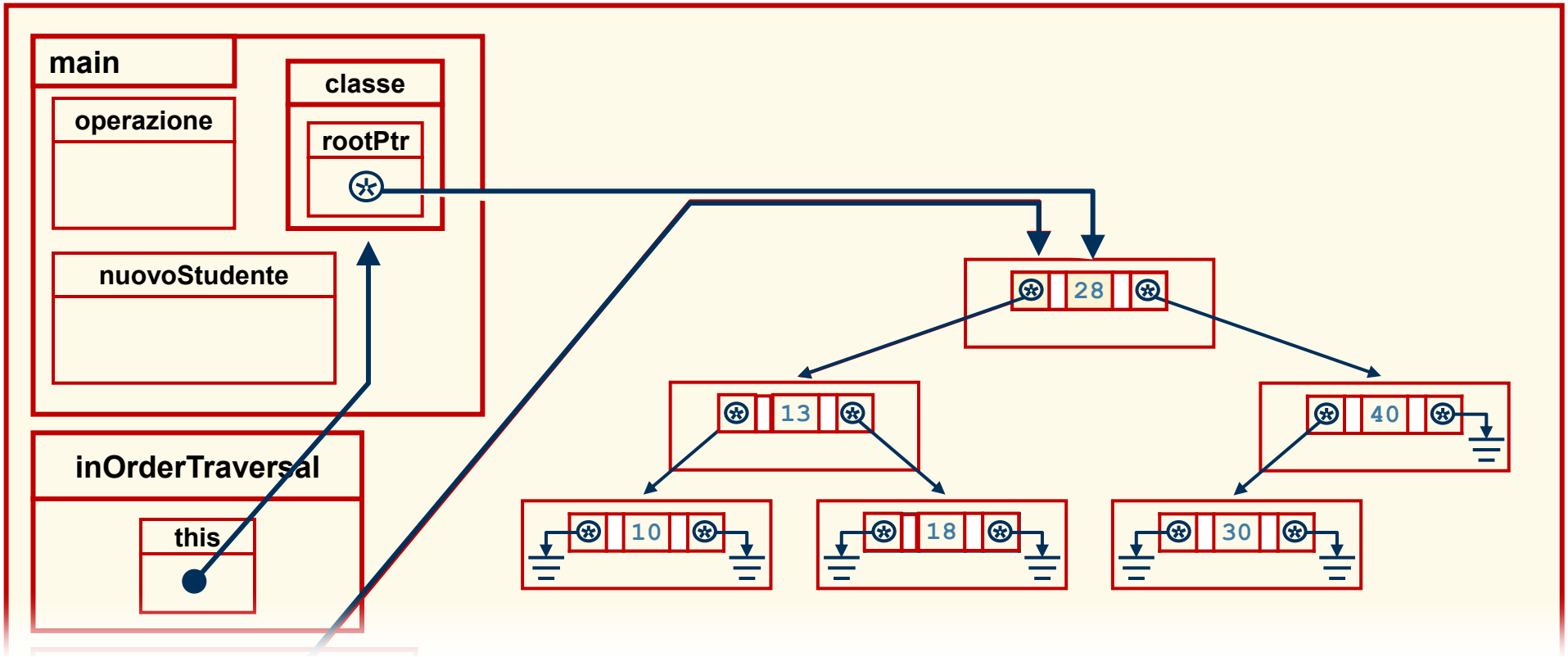

main
operazione
classe
rootPtr
nuovoStudente
28
13
40
10
18
30
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
recursiveInOrder γ
ptr


main
operazione
classe
rootPtr
nuovoStudente
inOrderTraversal
this
recursiveInOrder α
ptr
recursiveInOrder β
ptr
28
13
40
10
18
30

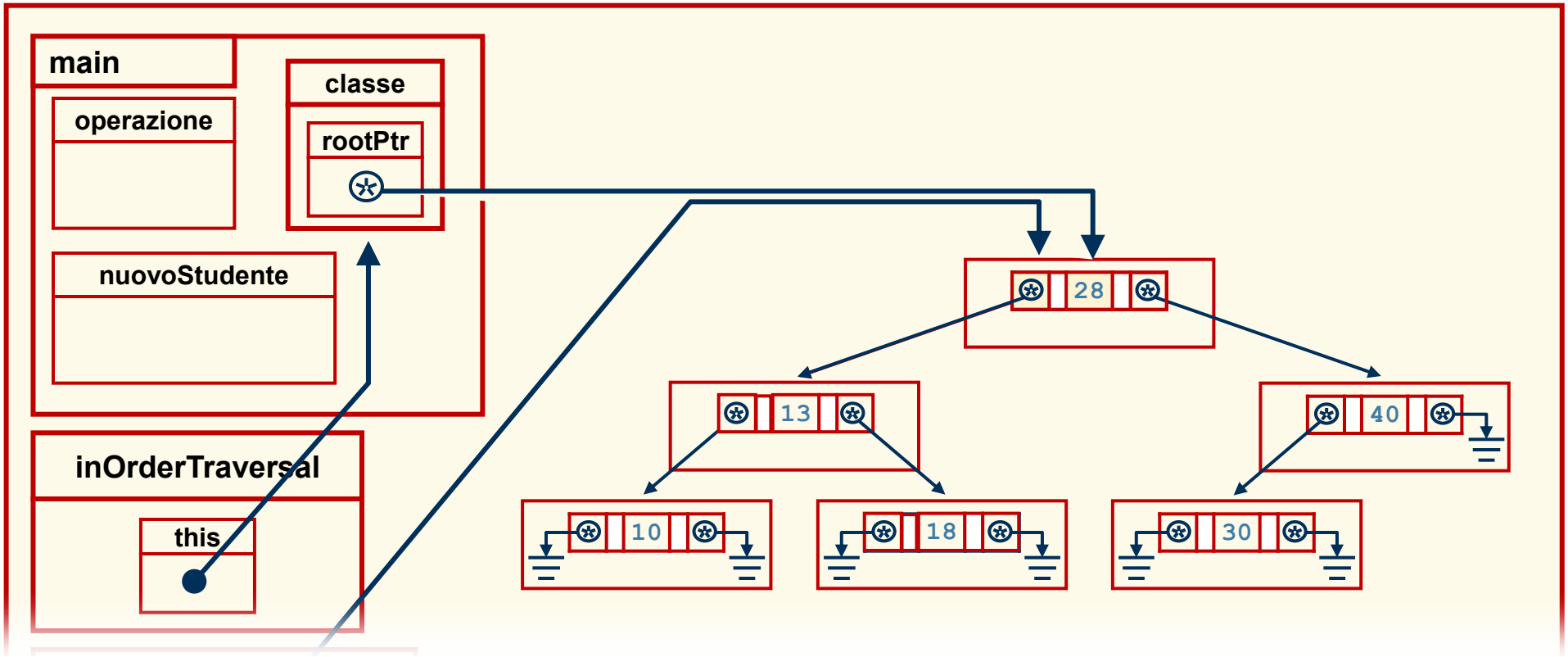



```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);
      }
}
```

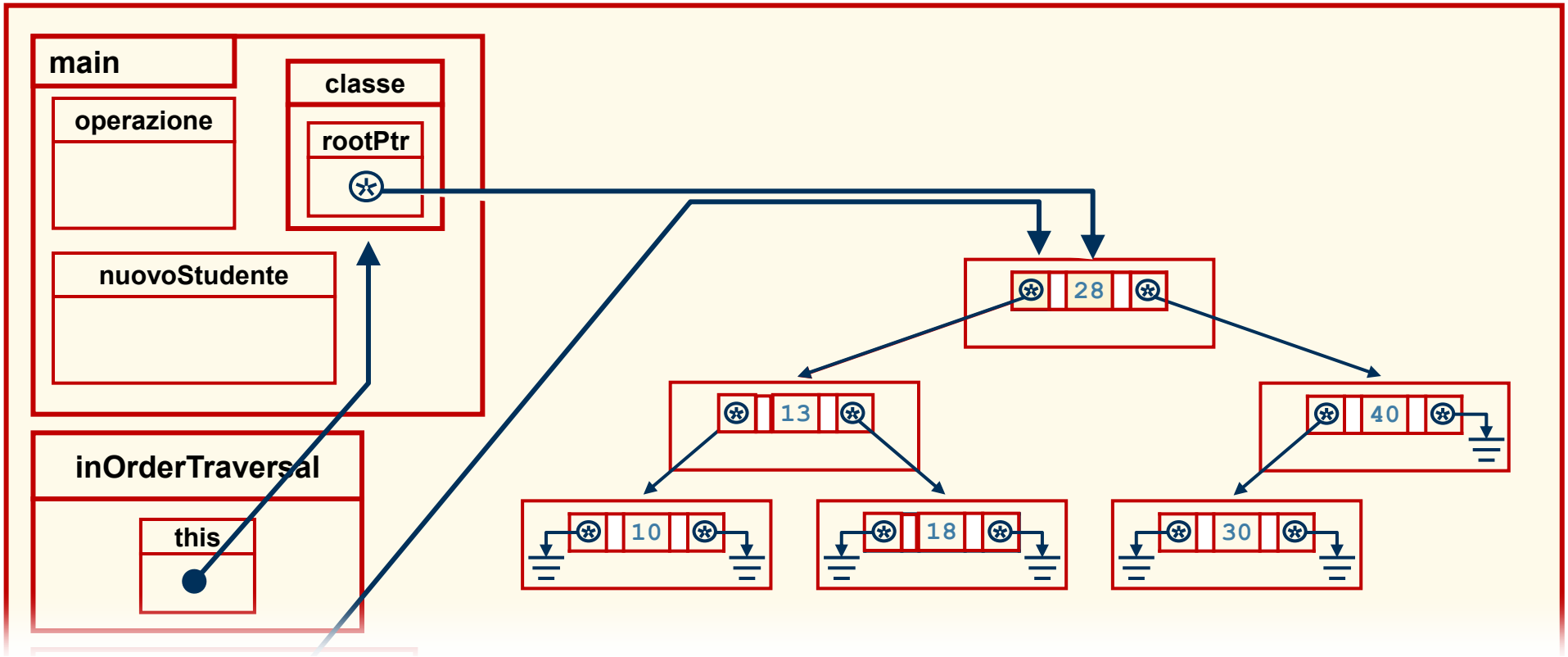
β



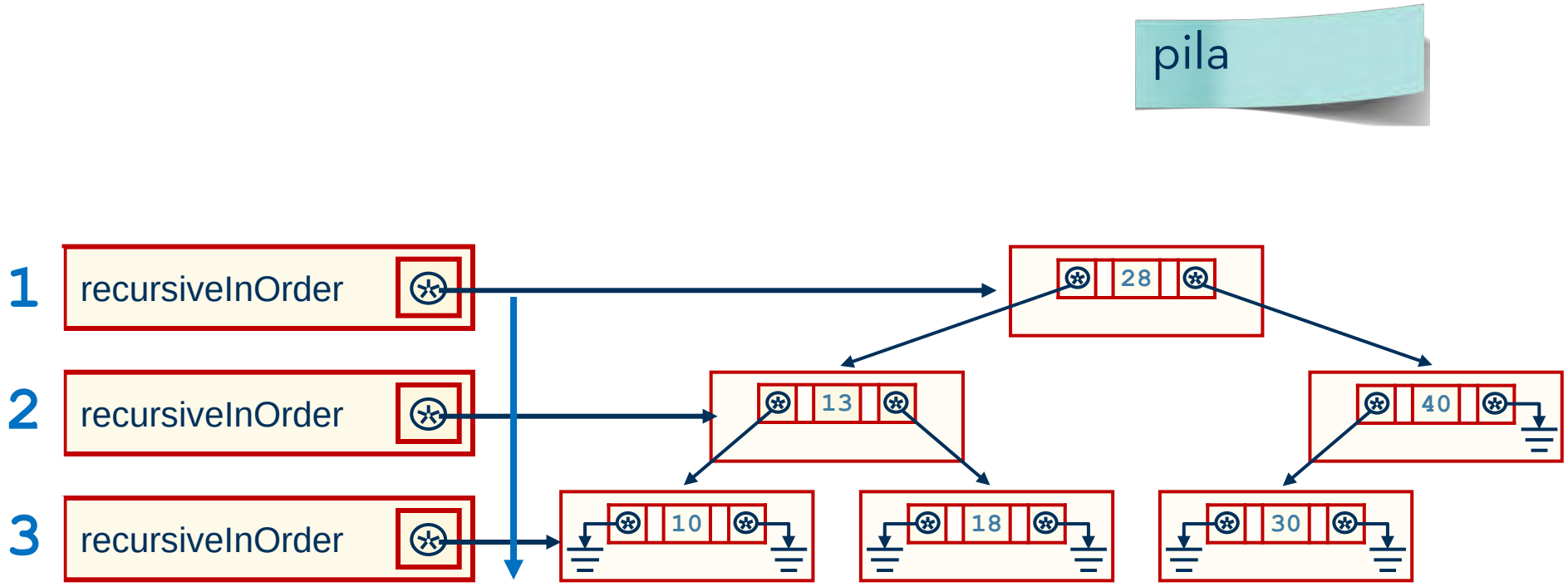

```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);
      }
}
```



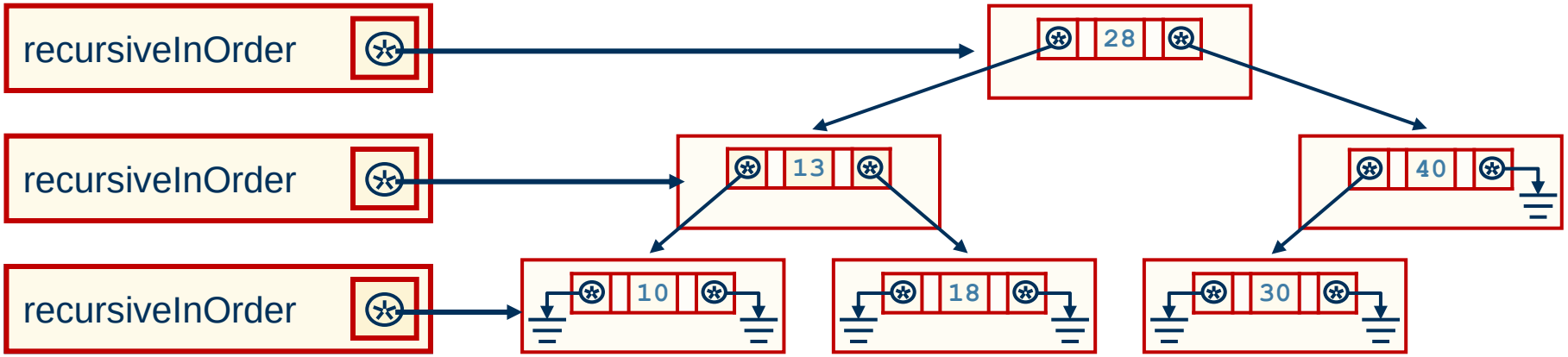

```
void Tree :: inOrderTraversal()
{   recursiveInOrder(rootPtr);
}
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);
      }
}
```



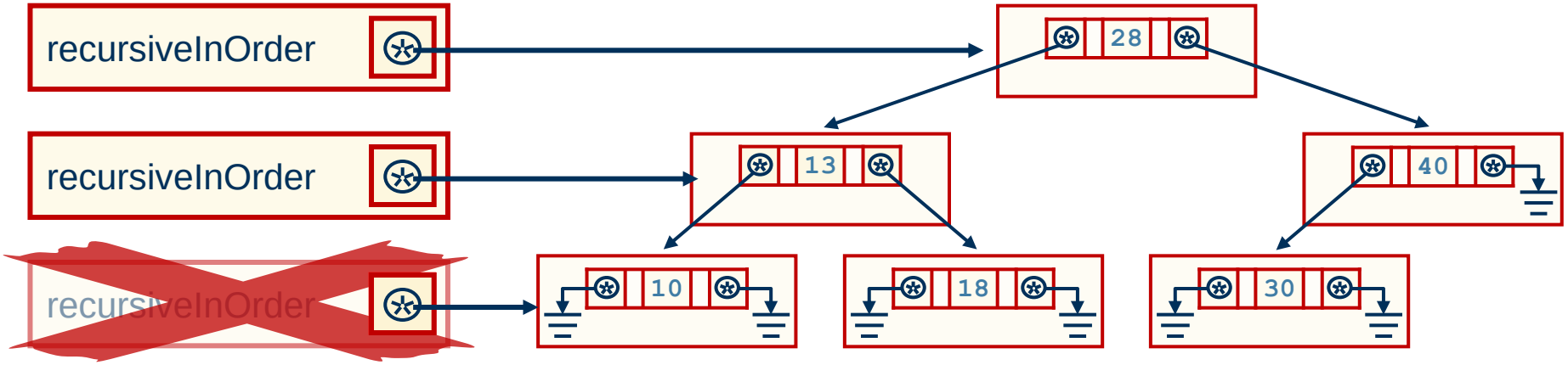
10 13 18 28 30 40



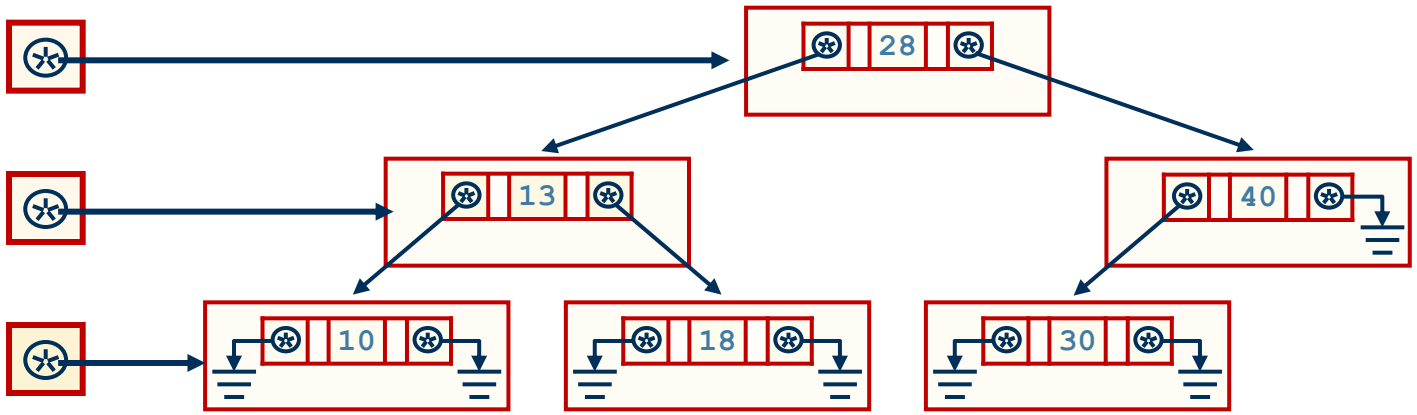

```
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
     { recursiveInOrder(ptr->leftPtr);
       cout << ptr->datiStud.matricola << ' ';
       recursiveInOrder(ptr->rightPtr);
     }
}
```




```
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
     { recursiveInOrder(ptr->leftPtr);
       cout << ptr->datiStud.matricola << ' ';
       recursiveInOrder(ptr->rightPtr);
     }
}
```

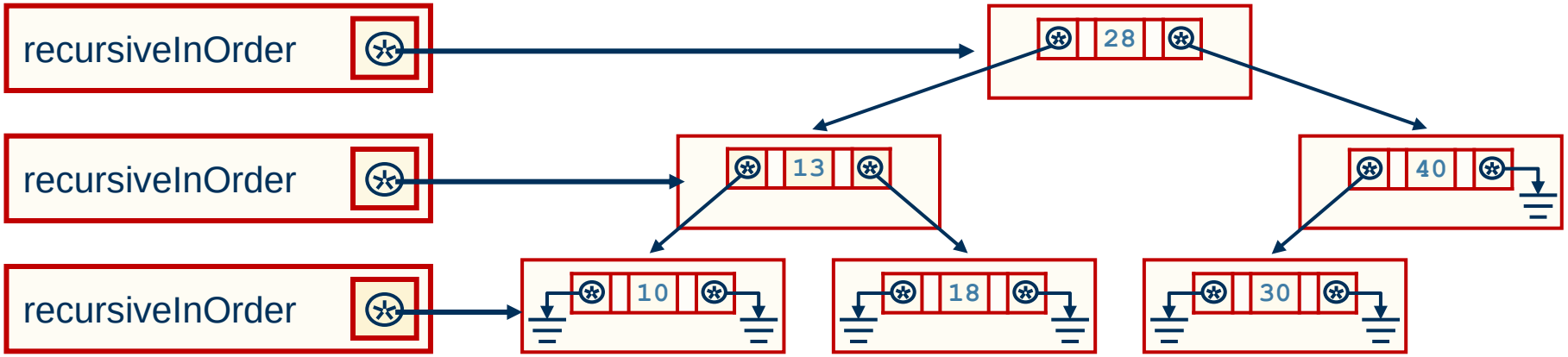



```
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
     { recursiveInOrder(ptr->leftPtr);
       cout << ptr->datiStud.matricola << ' ';
       recursiveInOrder(ptr->rightPtr);
     }
}
```

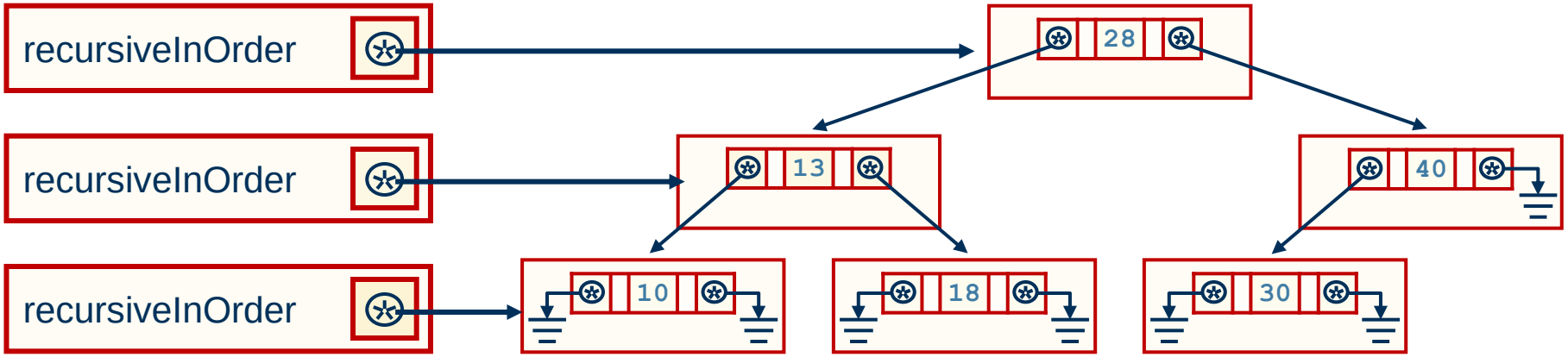


soluzione iterativa

Doppia ricorsione




```
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
     { recursiveInOrder(ptr->leftPtr);
       cout << ptr->datiStud.matricola << ' ';
       recursiveInOrder(ptr->rightPtr);
     }
}
```




```
void Tree :: recursiveInOrder(TreeNode *ptr)
{   if (ptr != 0)
      { recursiveInOrder(ptr->leftPtr);
        cout << ptr->datiStud.matricola << ' ';
        recursiveInOrder(ptr->rightPtr);
      }
}
```

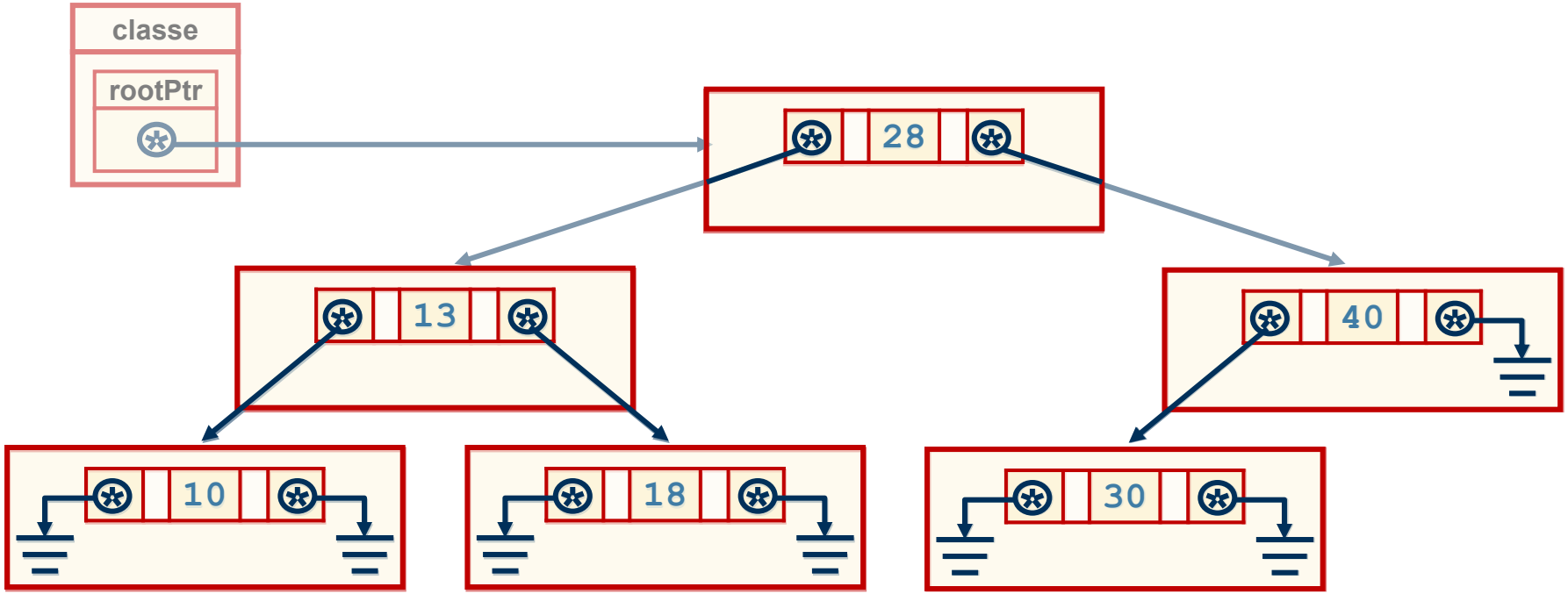
```
void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
class Tree
{
  public:
    ...
    void inOrderTraversal();
  private:
    ...
    void recursiveInOrder(TreeNode *);
};

void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

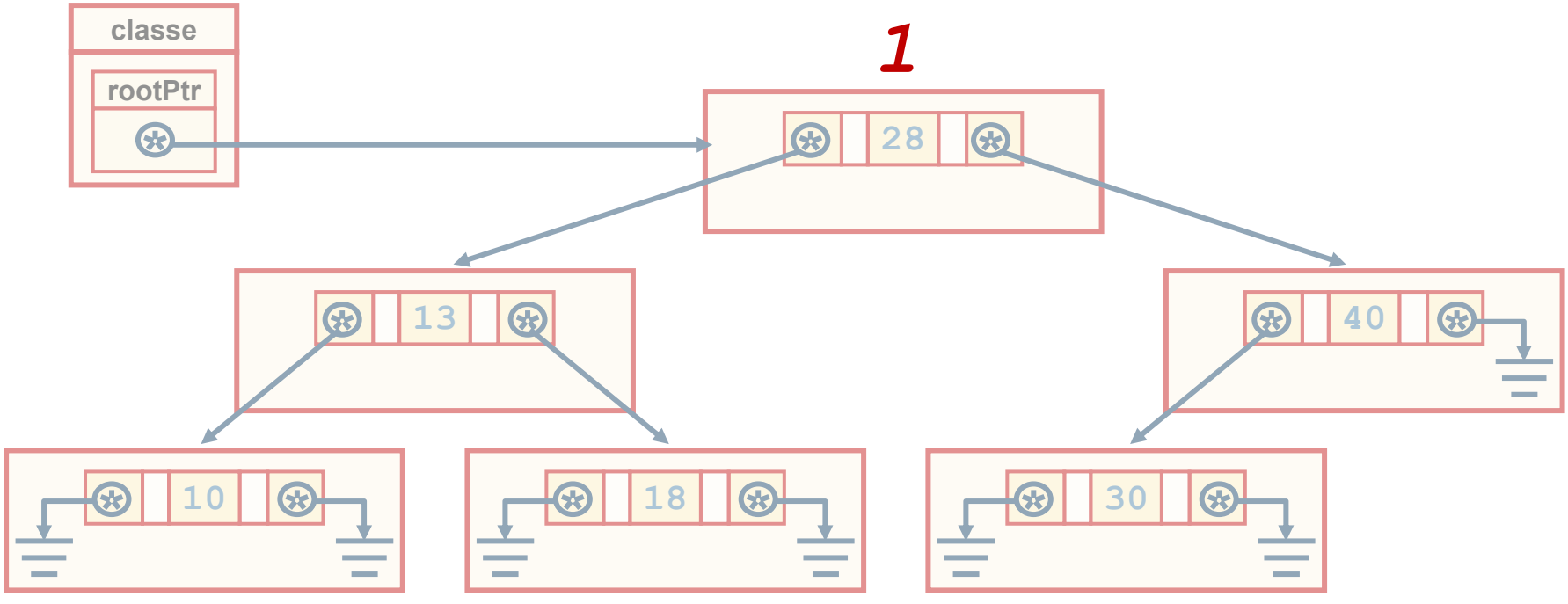
```
class Tree
{
  public:
    ...
    void inOrderTraversal();
  private:
    ...
    void recursiveInOrder(TreeNode *);
};

void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

```
class Tree
{
  public:
    ...
    void inOrderTraversal();
  private:
    ...
    void recursiveInOrder(TreeNode *);
};

void Tree :: inOrderTraversal()
{
  recursiveInOrder(rootPtr);
}

void Tree :: recursiveInOrder(TreeNode *ptr)
{  if (ptr != 0)
    { recursiveInOrder(ptr->leftPtr);
      cout << ptr->datiStud.matricola << ' ';
      recursiveInOrder(ptr->rightPtr);
    }
}
```

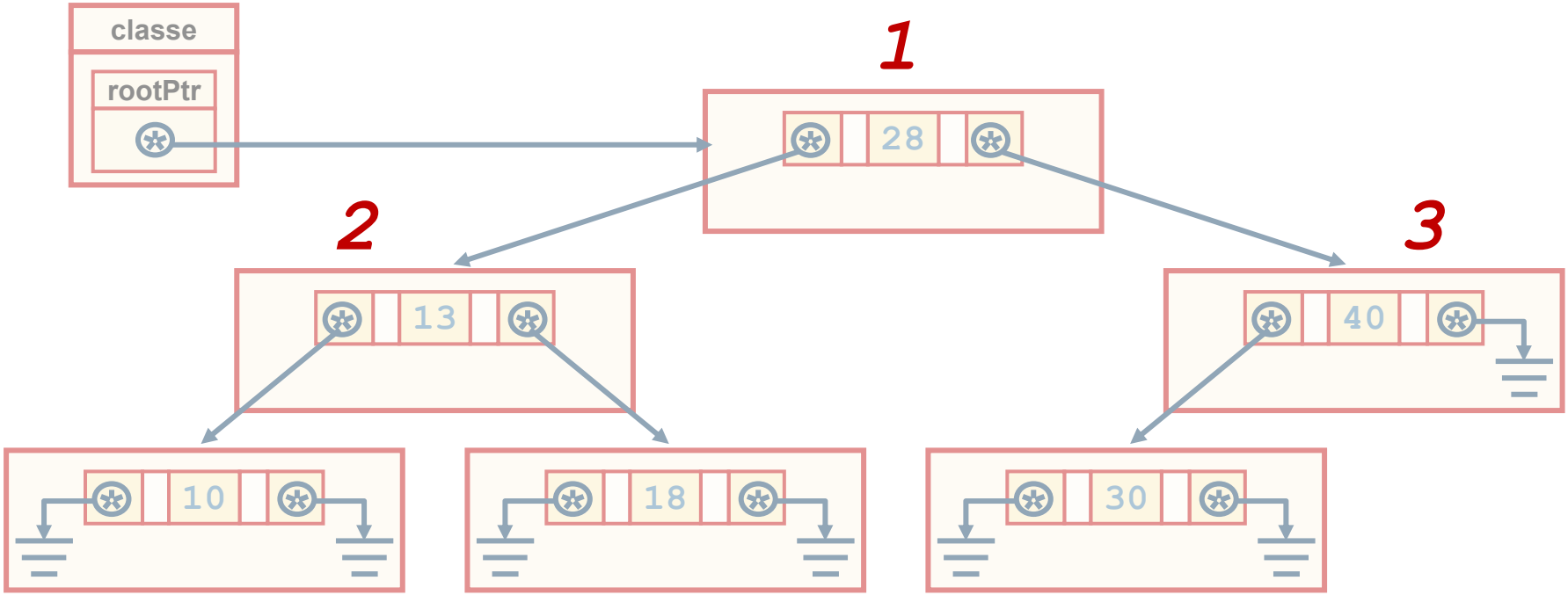
inOrderTraversal

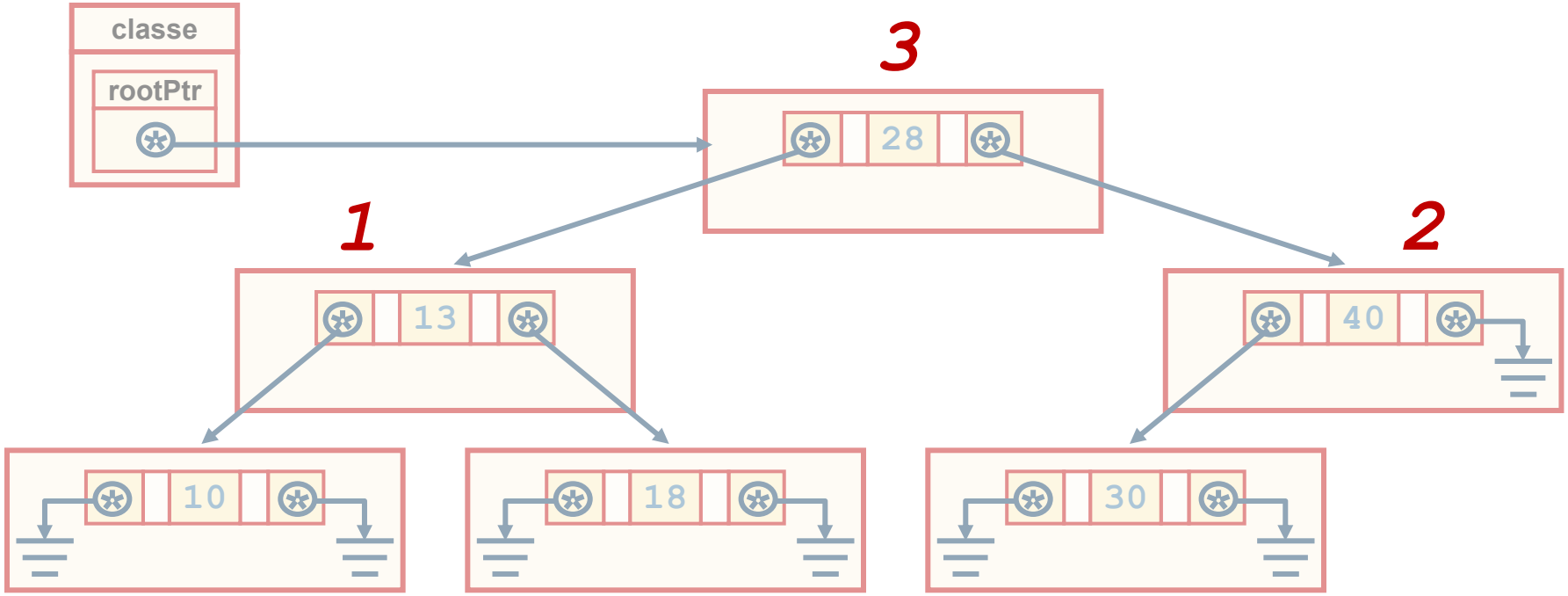


inOrderTraversal

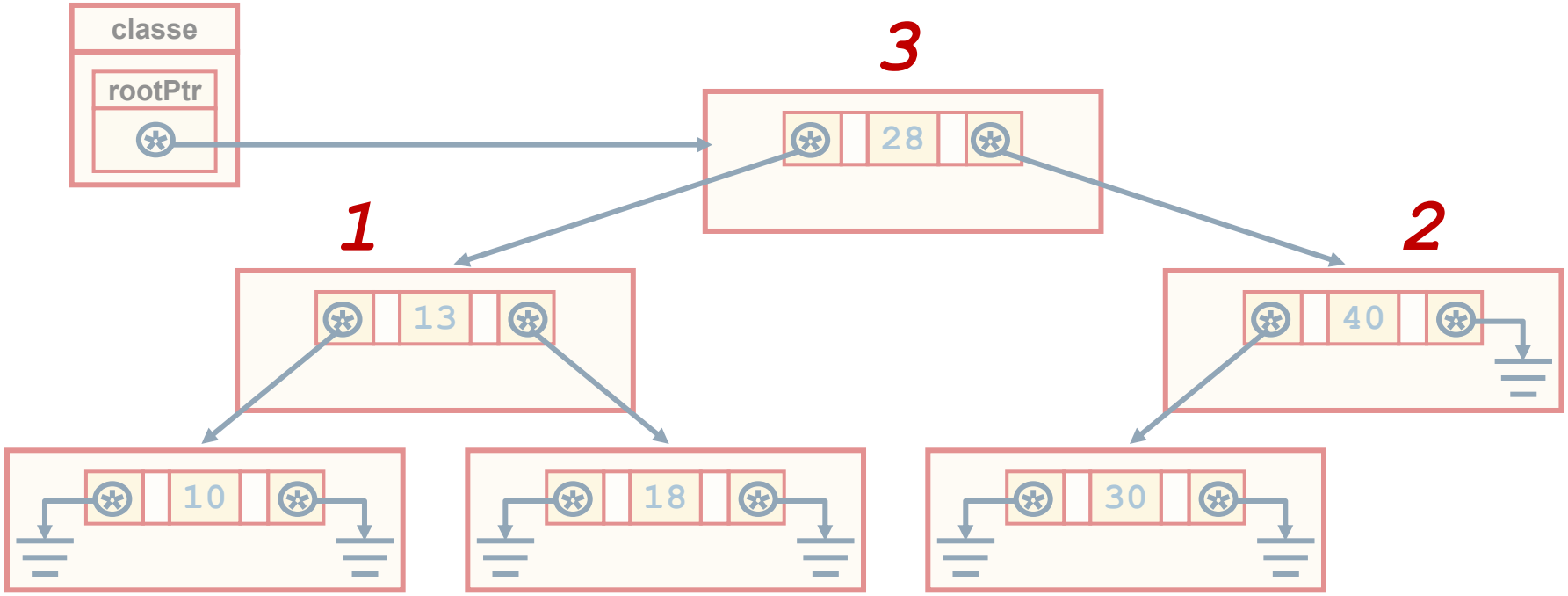


preOrderTraversal

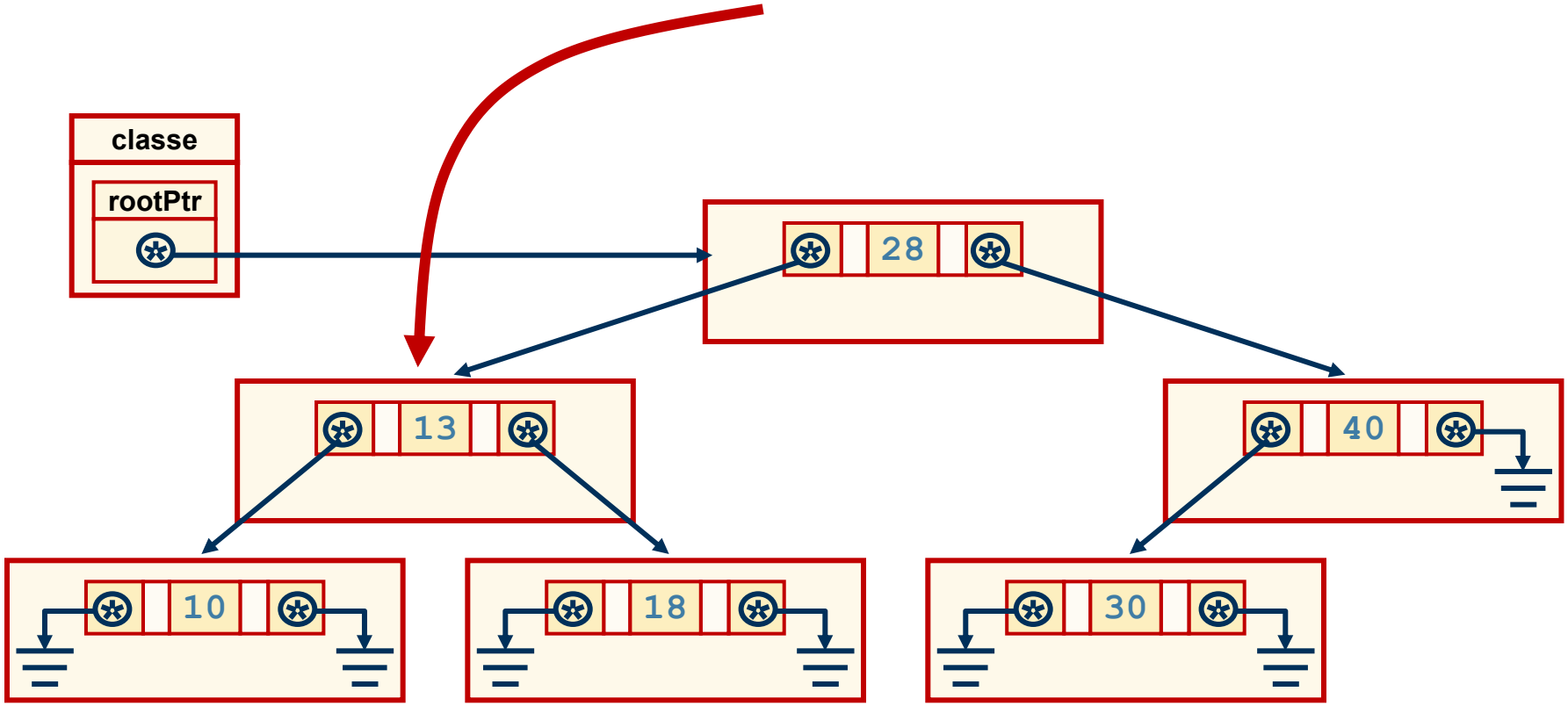


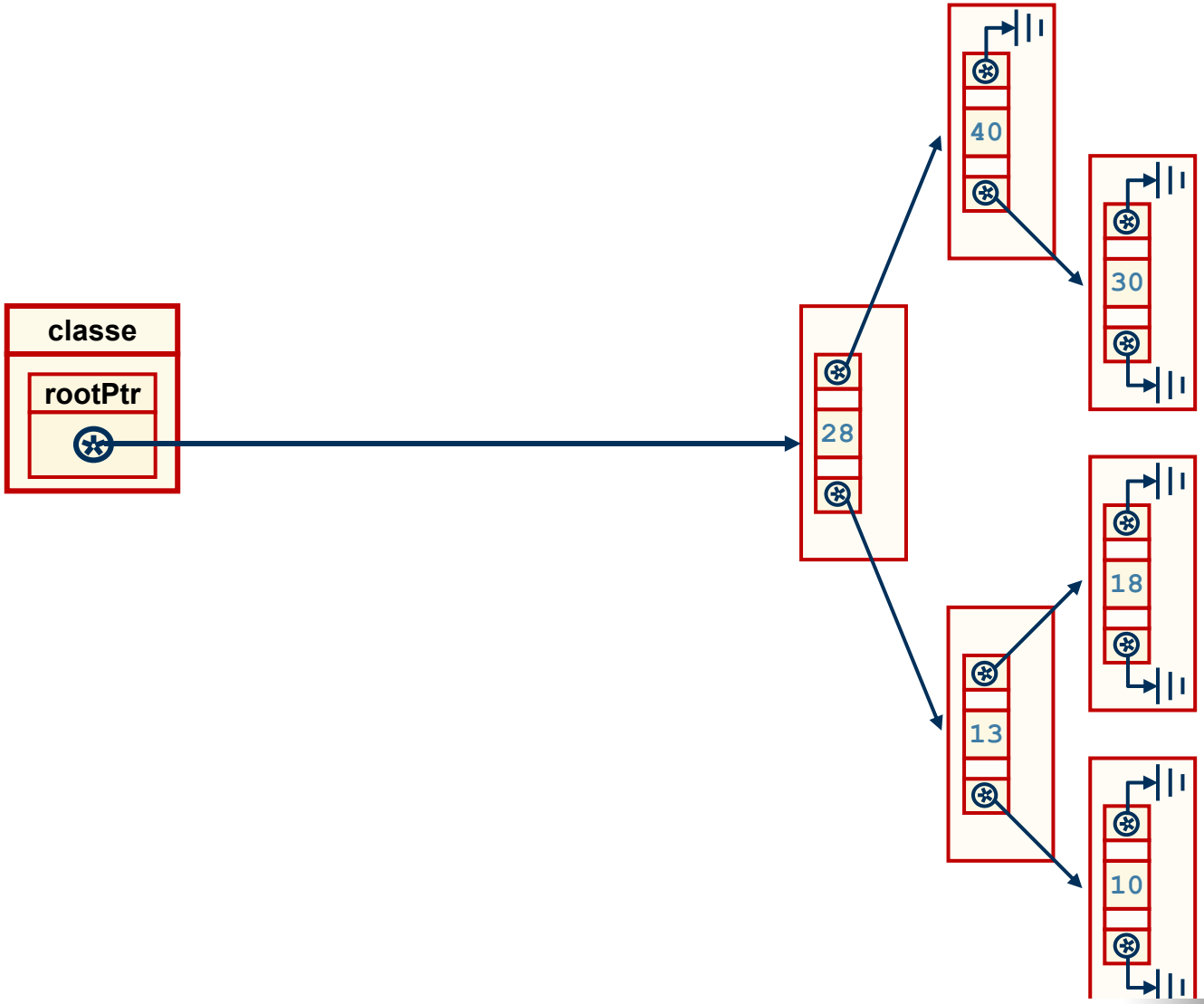

classe
rootPtr
3
28
1
13
2
40
10
18
30

postOrderTraversal



Albero rovesciato



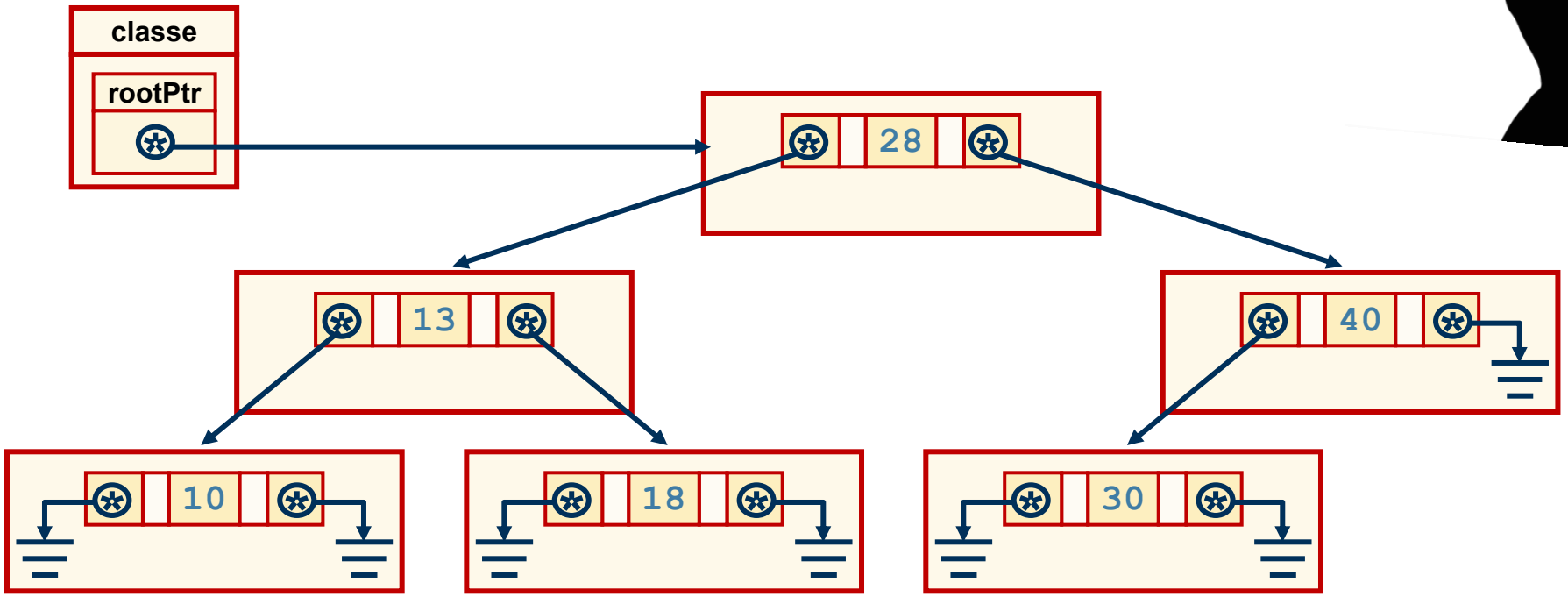

classe
rootPtr
28
40
30
13
18
10

```
class Tree
{
  public:
    Tree();
    void inserisciSeNonEsiste(Studente);
    void inOrderTraversal();
    void preOrderTraversal();
    void postOrderTraversal();
    void stampaAlbertoRovesciato();
  private:
    TreeNode *rootPtr;
  //funzioni di servizio
    void inserisciConRicorsione(TreeNode *&, Studente);
    void recursiveInOrder(TreeNode *);
    void recursivePreOrder(TreeNode *);
    void recursivePostOder(TreeNode *);
    void stampaRovesciatoRicorsiva(TreeNode *, int);
};
```

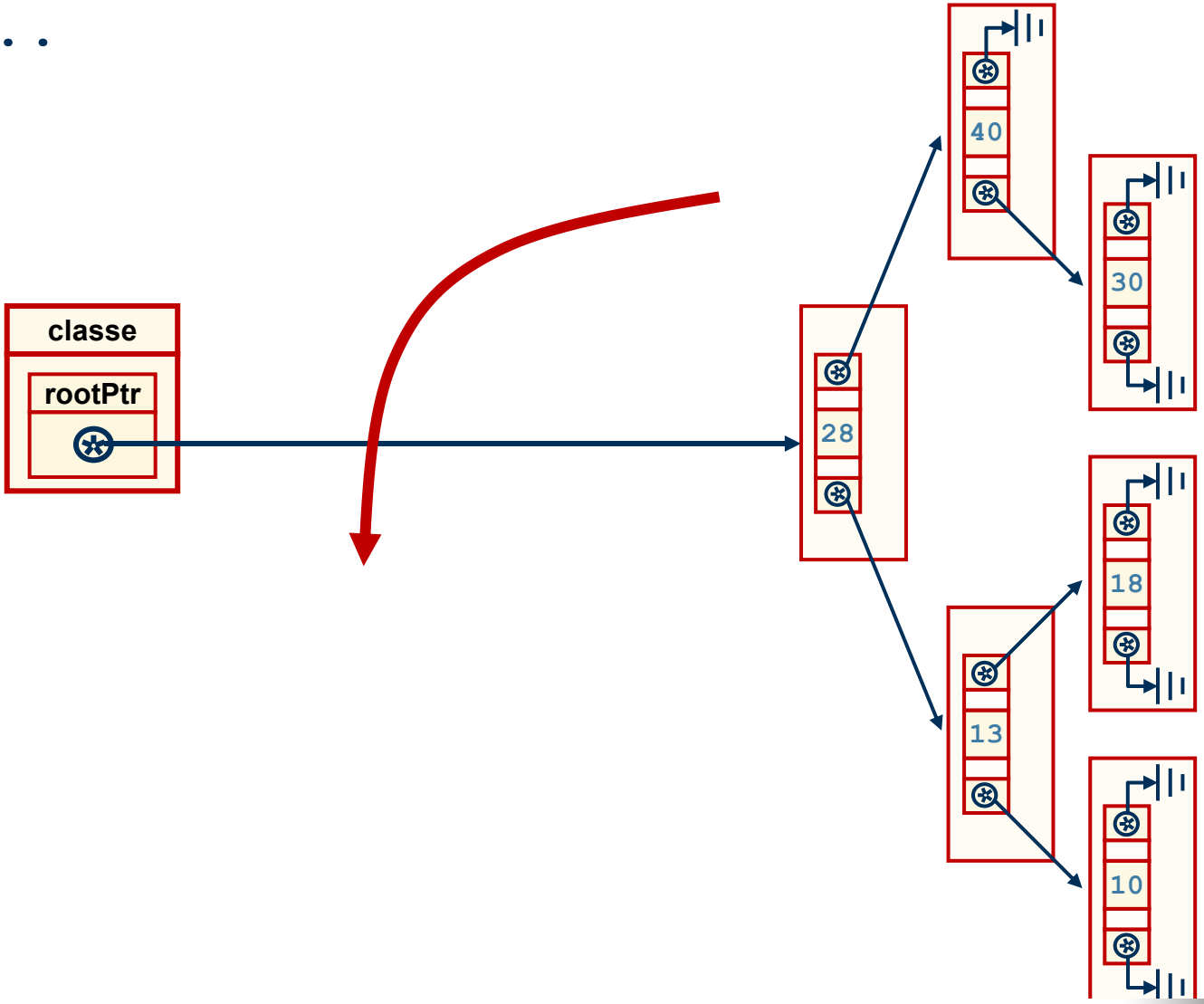
```
class Tree
{
  public:
    Tree();
    void inserisciSeNonEsiste(Studente);
    void inOrderTraversal();
    void preOrderTraversal();
    void postOrderTraversal();
    void stampaAlbertoRovesciato();
  private:
    TreeNode *rootPtr;
  //funzioni di servizio
    void inserisciConRicorsione(TreeNode *&, Studente);
    void recursiveInOrder(TreeNode *);
    void recursivePreOrder(TreeNode *);
    void recursivePostOder(TreeNode *);
    void stampaRovesciatoRicorsiva(TreeNode *, int);
};
```

```
class Tree
{
  public:
    Tree();
    void inserisciSeNonEsiste(Studente);
    void inOrderTraversal();
    void preOrderTraversal();
    void postOrderTraversal();
    void stampaAlbertoRovesciato();
  private:
    TreeNode *rootPtr;
  //funzioni di servizio
    void inserisciConRicorsione(TreeNode *&, Studente);
    void recursiveInOrder(TreeNode *);
    void recursivePreOrder(TreeNode *);
    void recursivePostOder(TreeNode *);
    void stampaRovesciatoRicorsiva(TreeNode *, int);
};
```

Tree :: stampaAlbero(…)



```
Tree :: stampaAlberoRovesciato(…)
{...
}
```



```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
40
30
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

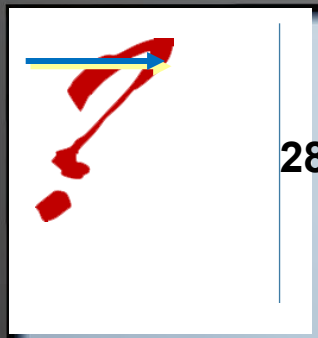
```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```



40

30

28

18

13

10

```
void Tree:: stampaAlberoRovesciato()
{

  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
        40
            30
    28
            18
        13
            10
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{

  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+ ...);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+ ...);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+ ...);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+incrIndent);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+incrIndent);
      cout << "-"
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
        40
            30
    28
            18
        13
            10
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+incrIndent);
      cout << setw(indent) << "-" << setw(dataDim)
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
        40
            30
    28
            18
        13
            10
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{ const int dataDim = 3;
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+incrIndent);
      cout << setw(indent) << "-" << setw(dataDim)
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
void Tree:: stampaAlberoRovesciato()
{
  cons int initIdent = 10;
  cout << "stato dell'albero ruotato di 90° in senso "
       << "antiorario" << endl;
  stampaRovesciatoRicorsiva(rootPtr, initIdent);
}

void Tree:: stampaRovesciatoRicorsiva(TreeNode *ptr, int indent)
{ const int dataDim = 3;
  const int incrIndent = 3;
  if (ptr != 0)
    { stampaRovesciatoRicorsiva(ptr->rightPtr,indent+incrIndent);
      cout << setw(indent) << "-" << setw(dataDim)
           << ptr->datiStud.matricola << endl;
      stampaRovesciatoRicorsiva(ptr->leftPtr);
    }
}
```

```
        - 40
              - 30
- 28
              - 18
        - 13
              - 10
```