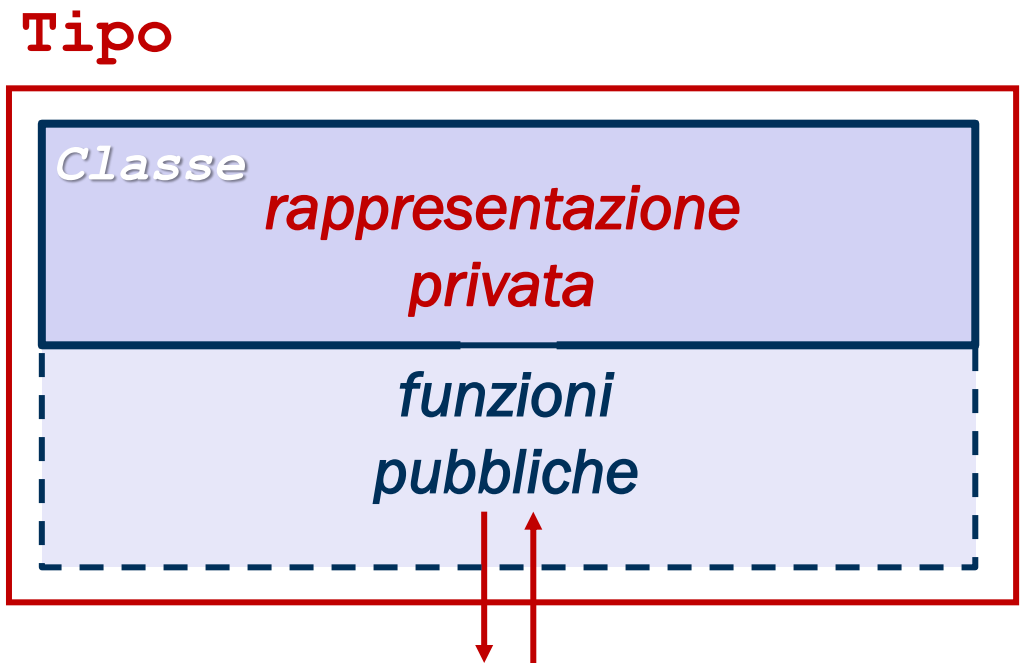
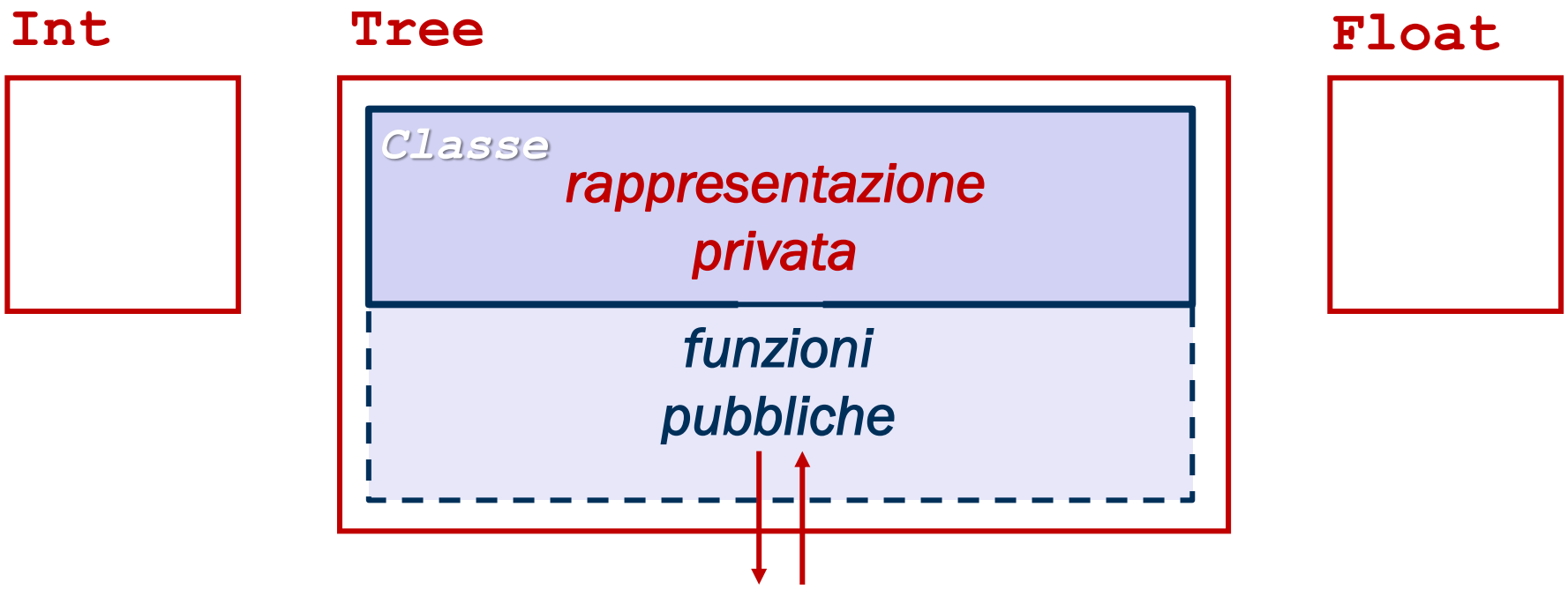
POLITECNICO DI MILANO

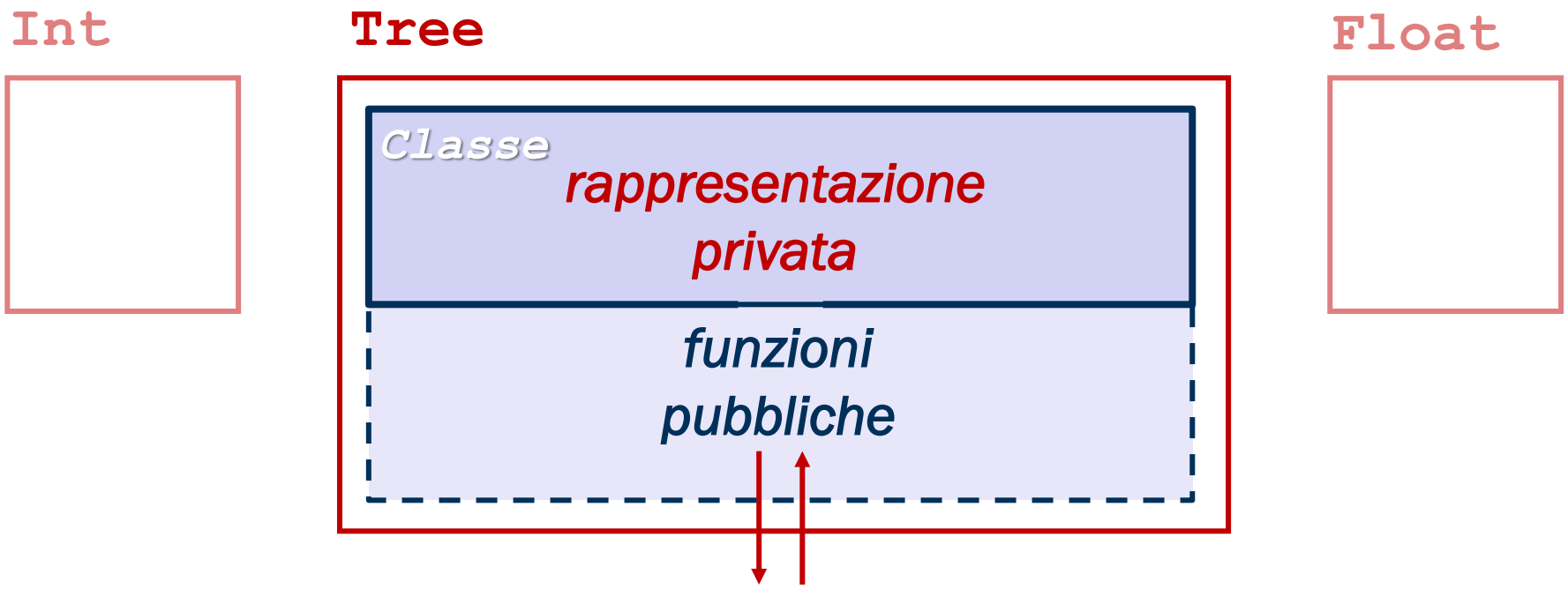
INFORMATICA

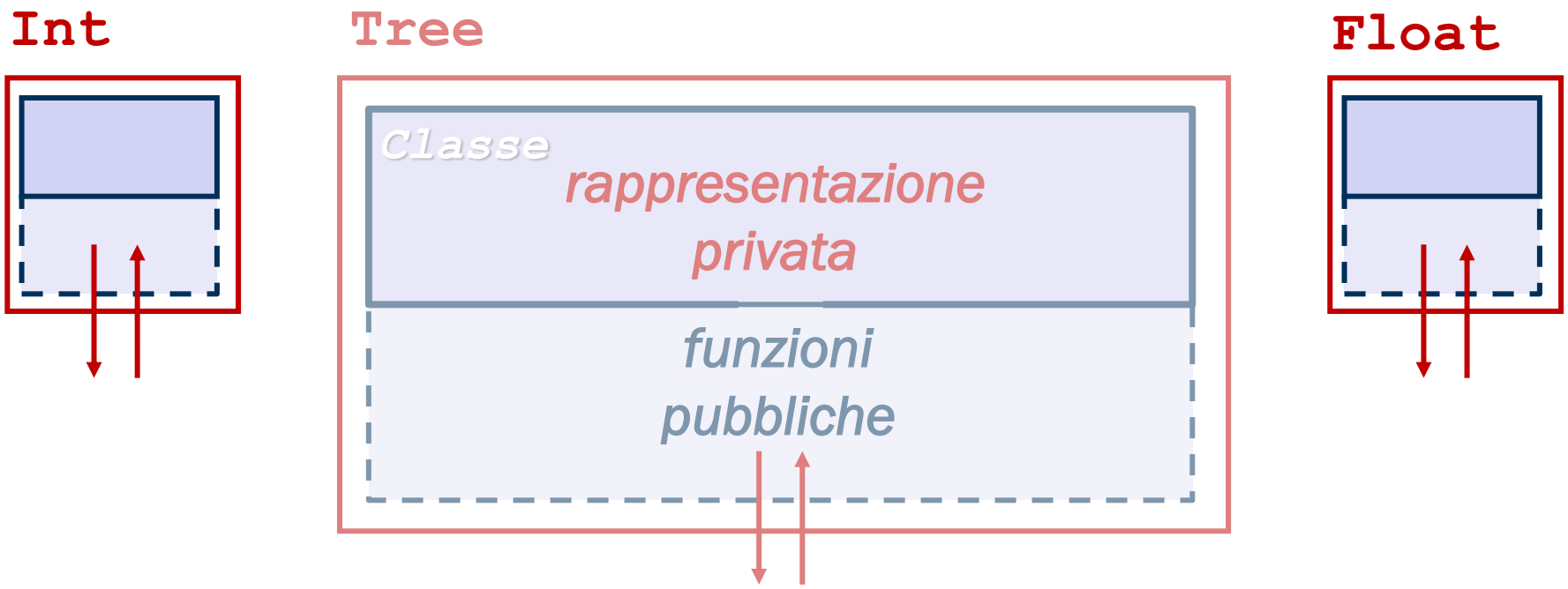
Le classi e i tipi di dato astratti: personalizzazione del linguaggio


Tipo
Classe
rappresentazione
privata
funzioni
pubbliche

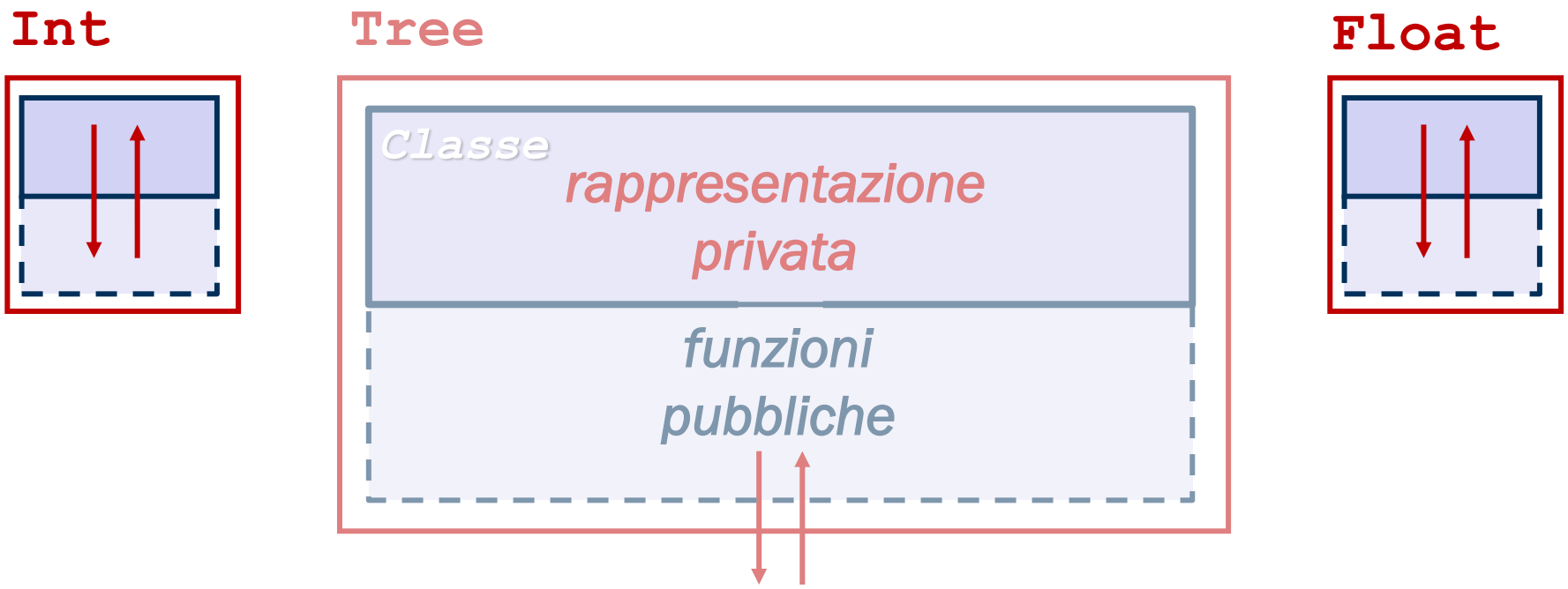

Int
Tree
Float
Classe
rappresentazione
privata
funzioni
pubbliche

Tipo di dato astratto

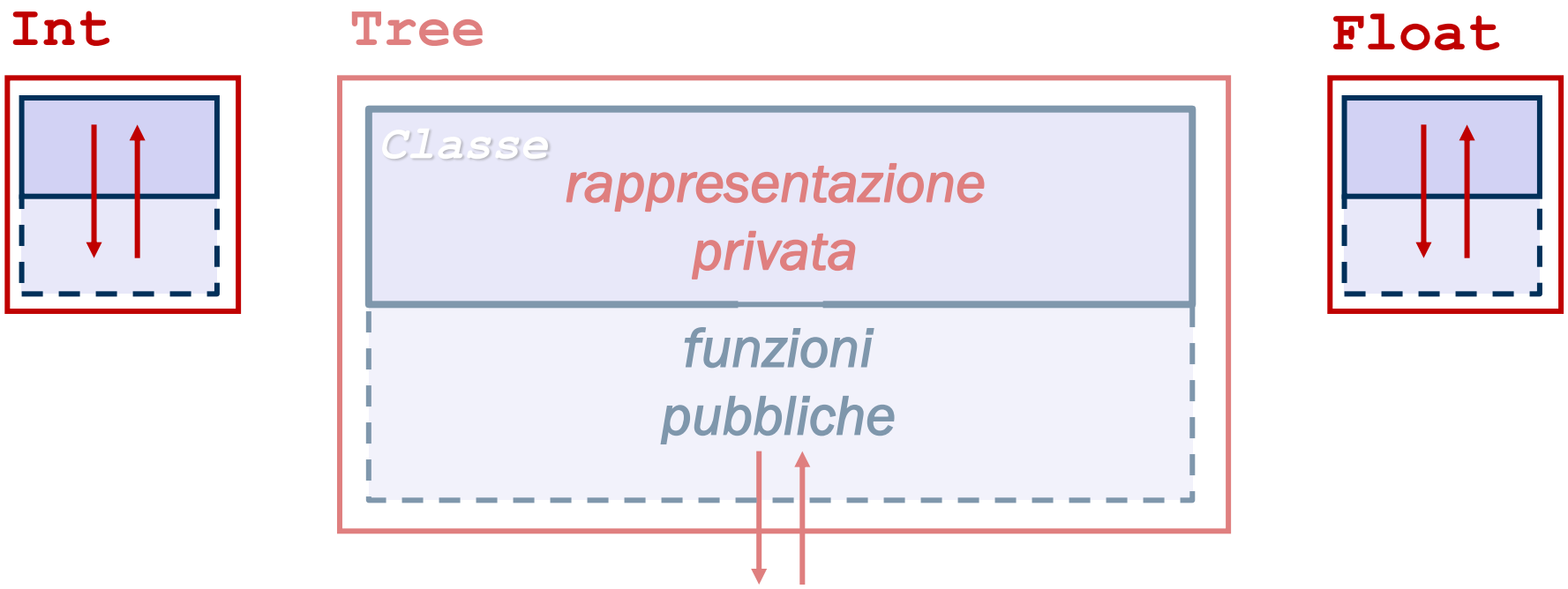




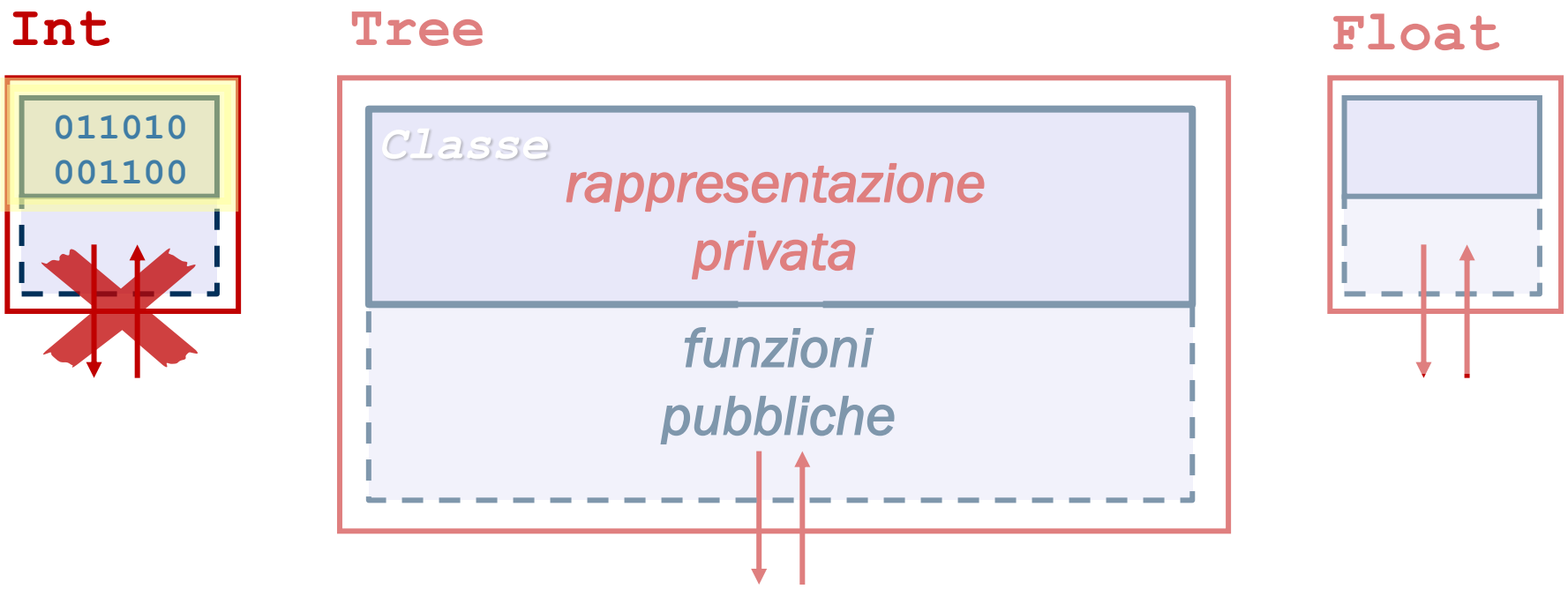
Operazioni predefinite



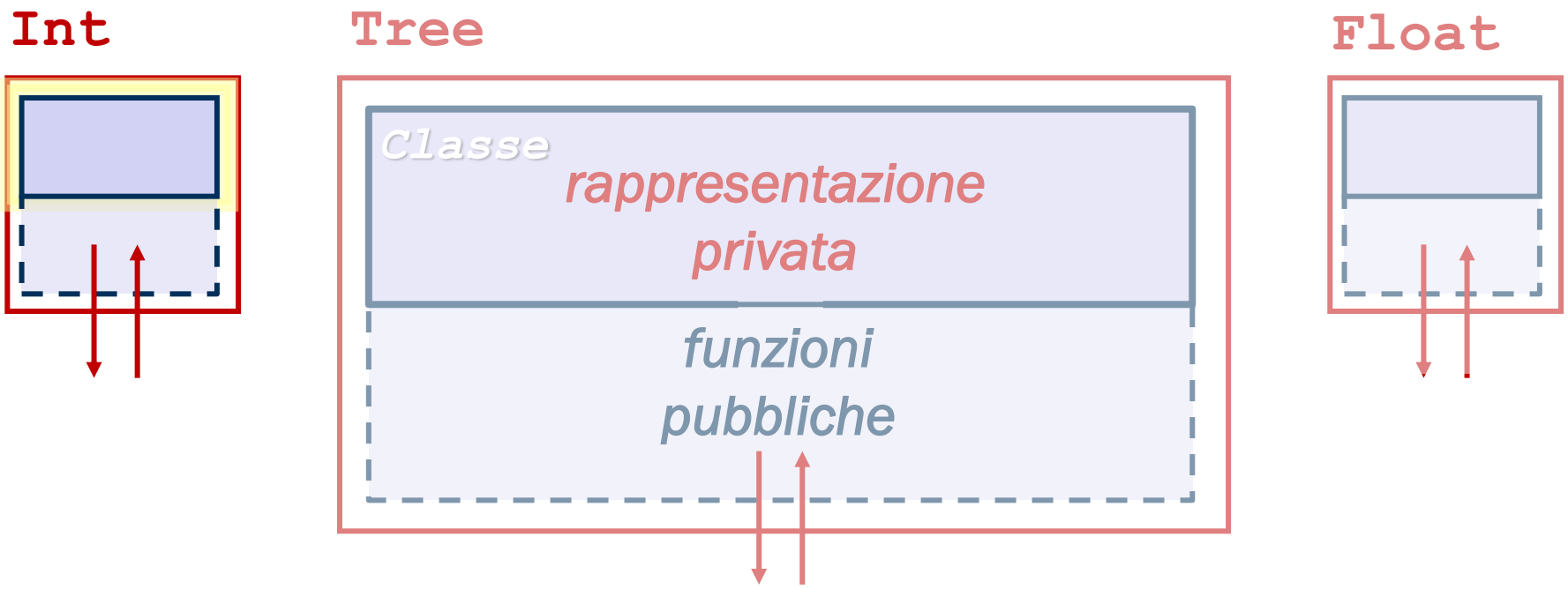
Operazioni predefinite

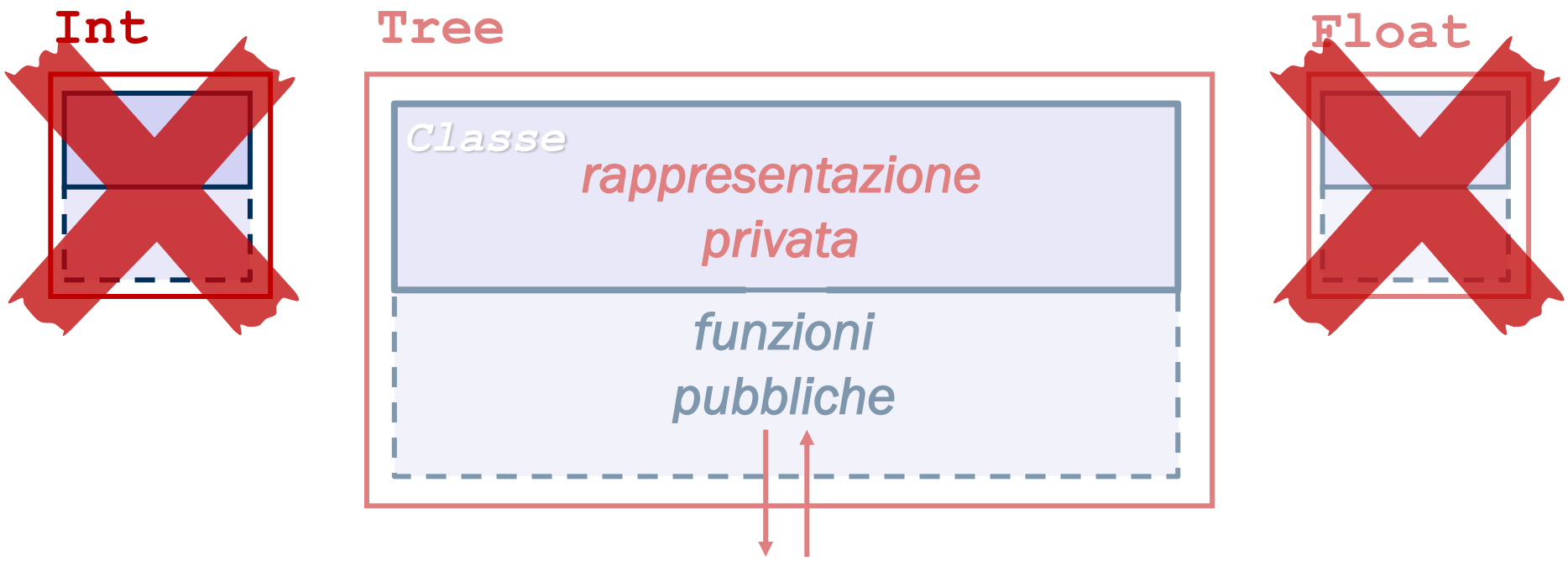


Rappresentazione interna inaccessibile

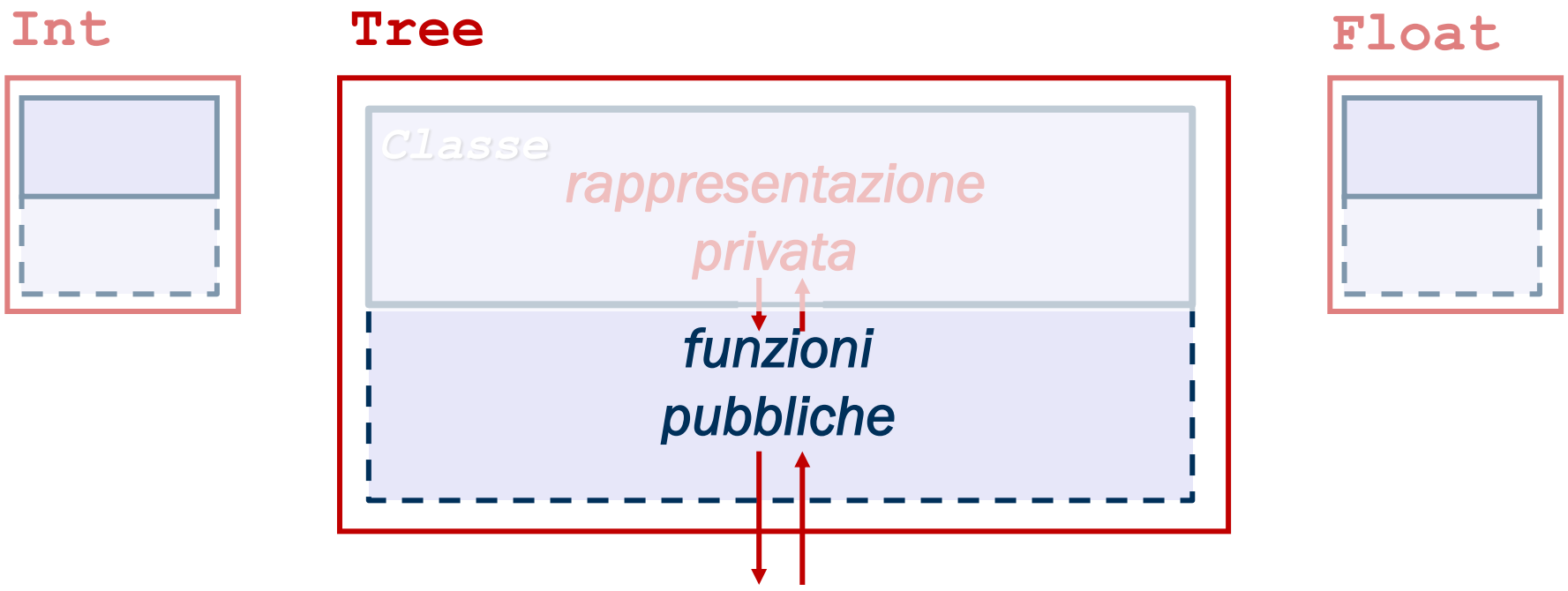

Int
011010
001100
Tree
Classe
rappresentazione
privata
funzioni
pubbliche
Float

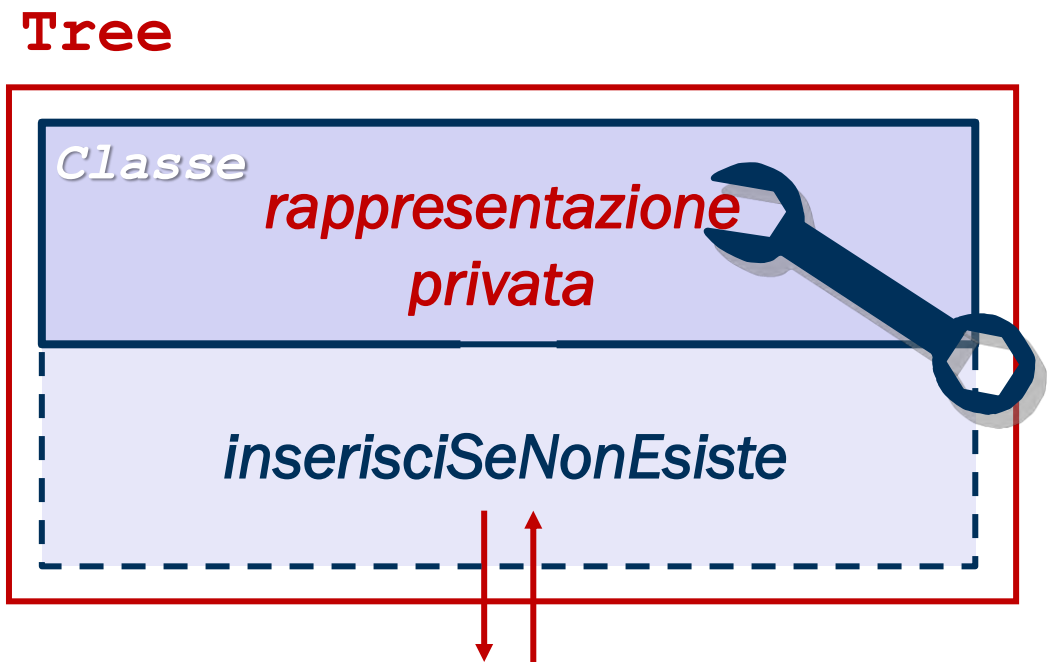
Rappresentazione consistente





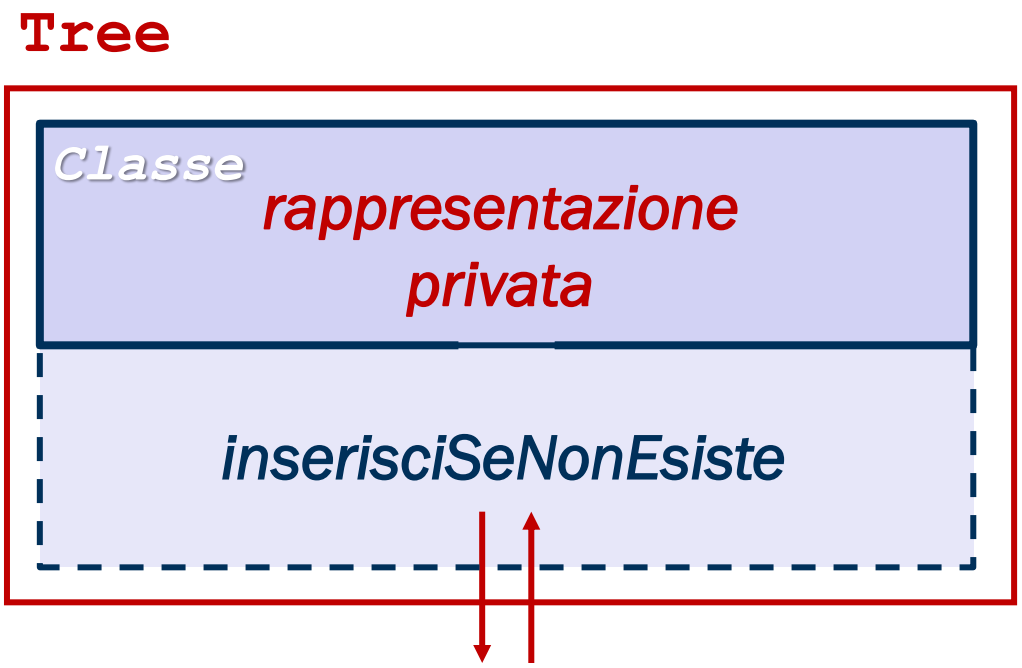
overflow
underflow


Int
Tree
Classe
rappresentazione
privata
funzioni
pubbliche
Float




```
cout << " anno di nascita (00-99): ";
cin >> nuovoStudente.dataNascita.anno;
classe.inserisciSeNonEsiste(nuovoStudente);
classe.stampaAlberoRovesciato();
}
```

Funzione membro




```
cout << " anno di nascita (00-99): ";
cin >> nuovoStudente.dataNascita.anno;
classe.inserisciSeNonEsiste(nuovoStudente);
classe.stampaAlberoRovesciato();
}
```

```
class Tree
{
  public:
    Tree();

    void funzioneABC(...);
    void funzioneABC(...);

    ...
  private:
    ...
};
```

```
class Tree
{
  public:
    Tree();

    void funzioneABC(int, char);
    void funzioneABC(TreeNode);

    ...
  private:
    ...
};
```

Overloading

```
class Tree
{
  public:
    Tree();

    void funzioneABC(int, char);
    void funzioneABC(TreeNode);

    ...
  private:
    ...
};
```

Sovraccaricamento

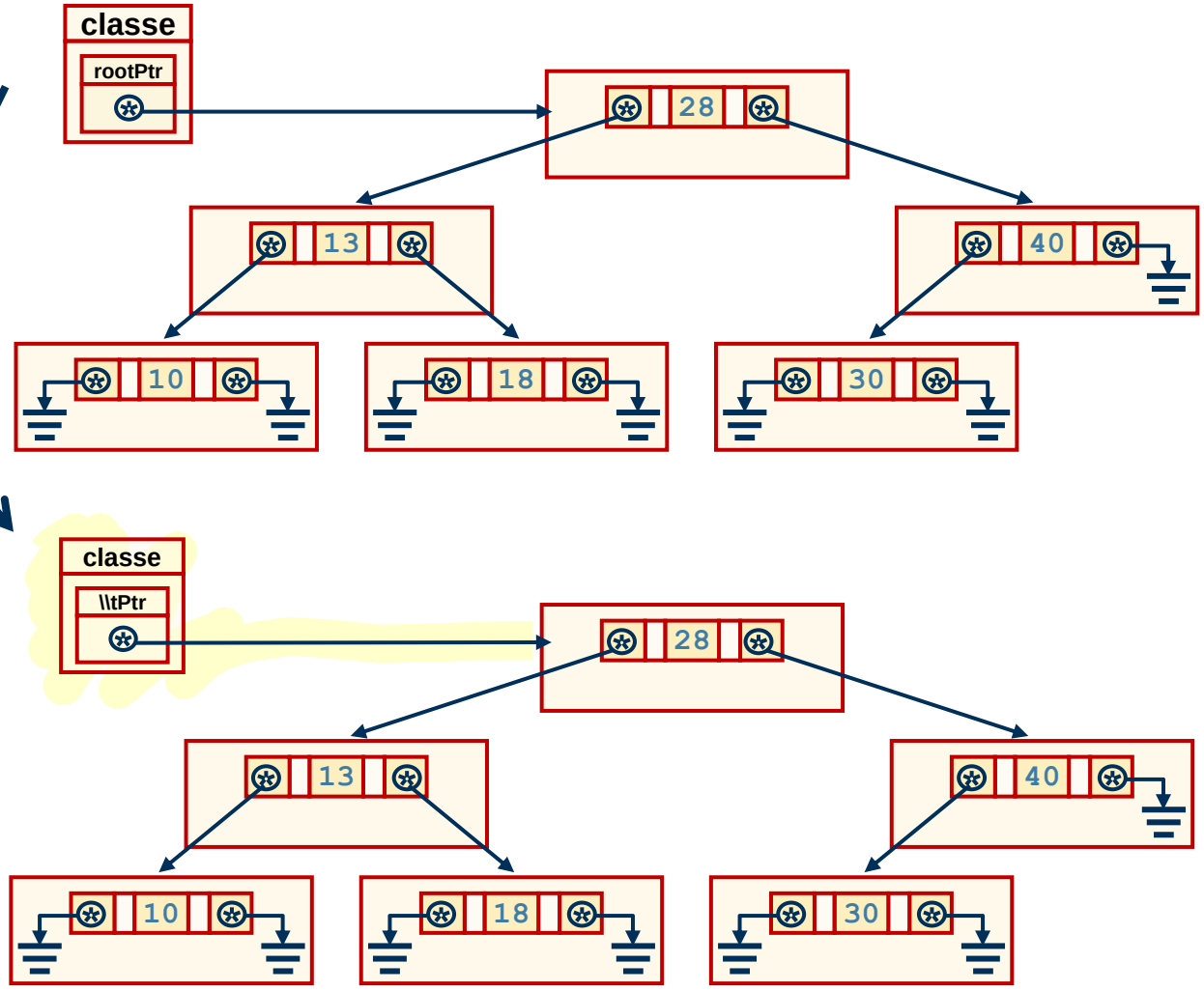
```
class Tree
{
  public:
    Tree();

    void funzioneABC(int, char);
    void funzioneABC(TreeNode);

    ...
  private:
    ...
};
```

```
class Tree
{
  public:
    Tree();
    Tree(...);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
}
```

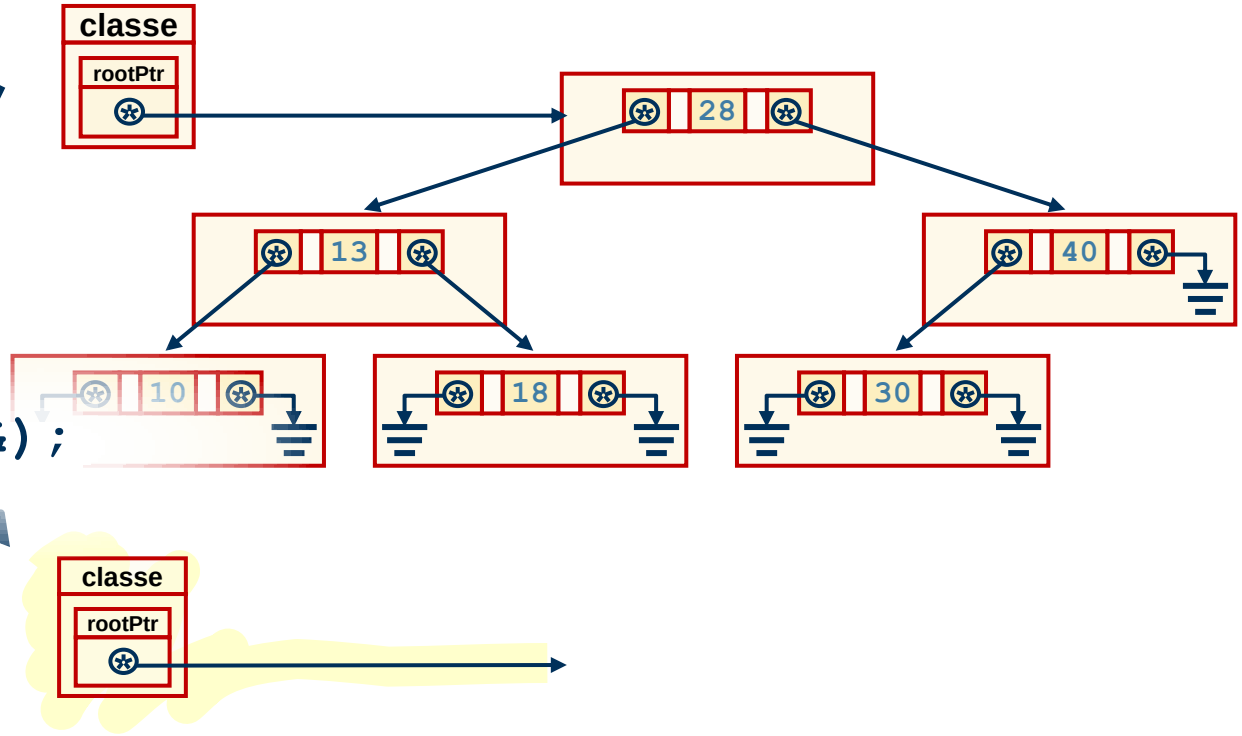
```
class Tree
{
  public:
    Tree();
    Tree(...);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
}
```

Costruttore per copia

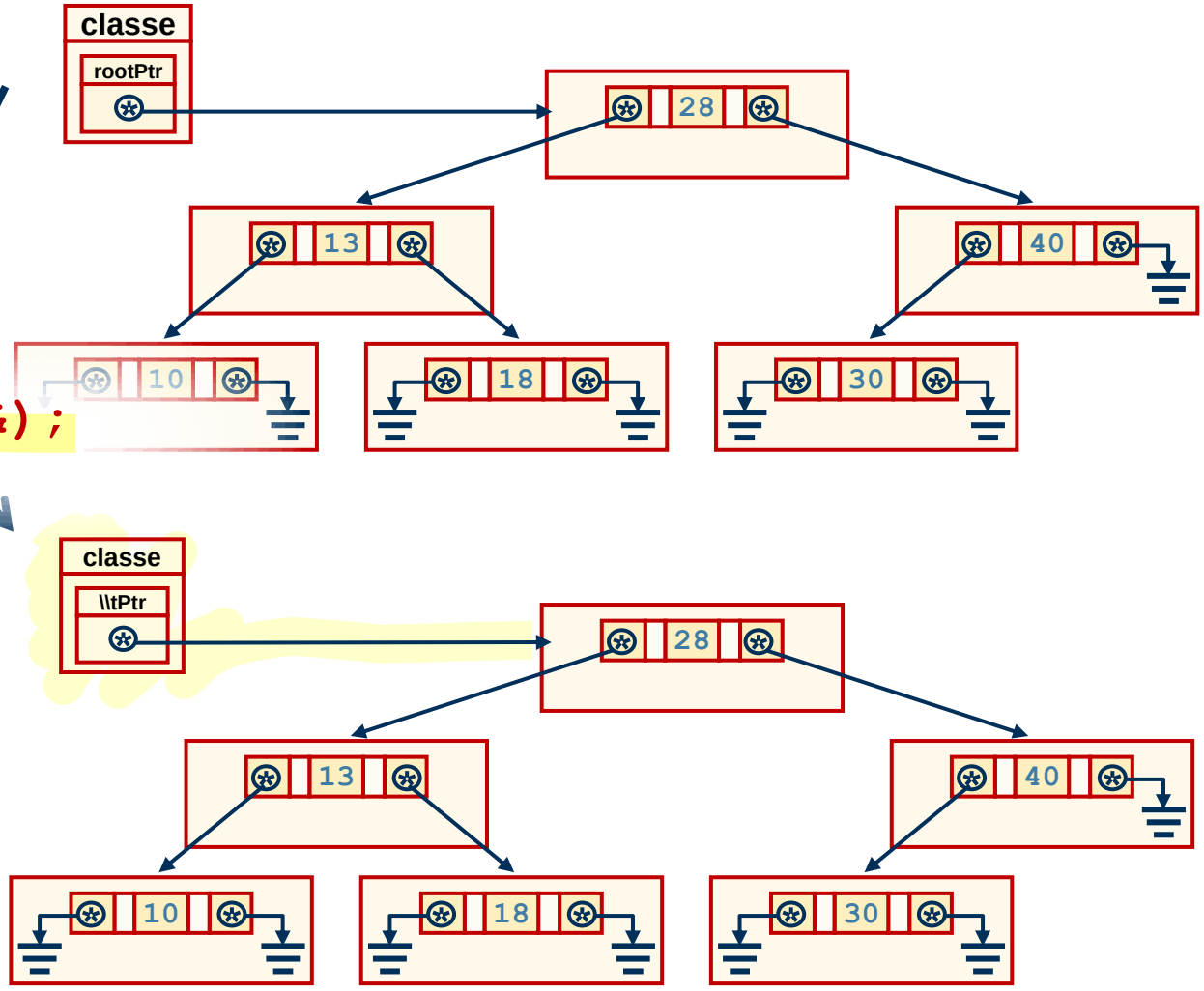
```
class Tree
{
  public:
    Tree();
    Tree(...);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
}
```

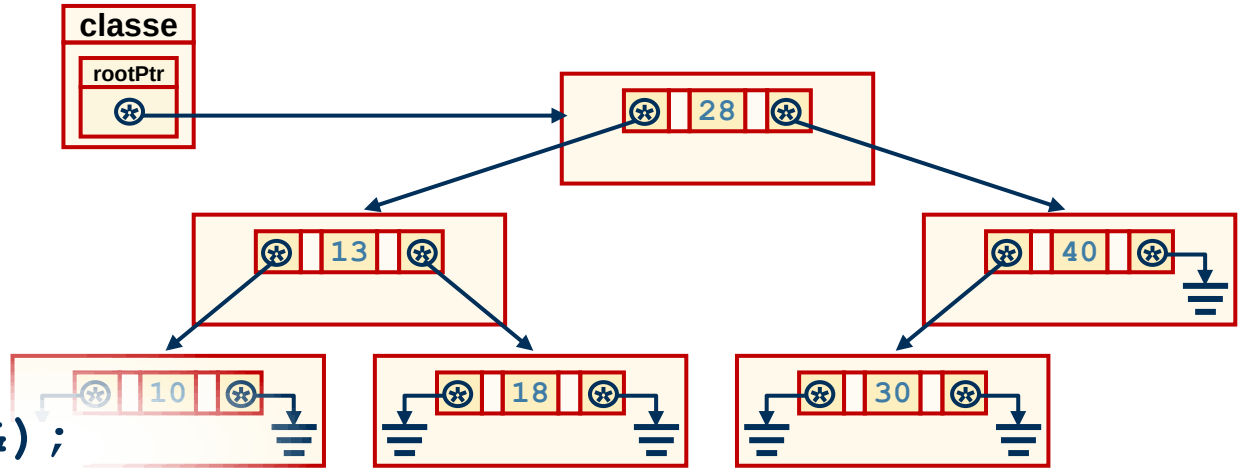


```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
  ...
  Tree classeBis(classe);
}
```



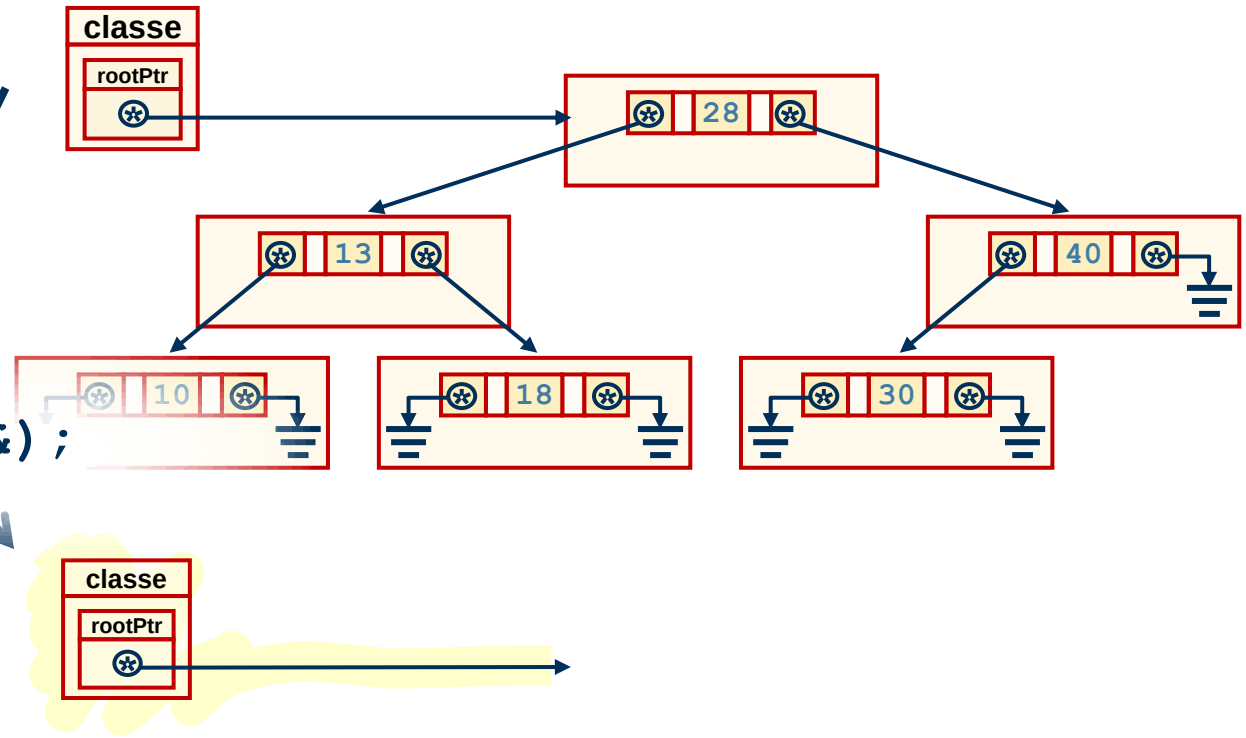
```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    ...
  private:
    ...
};
```



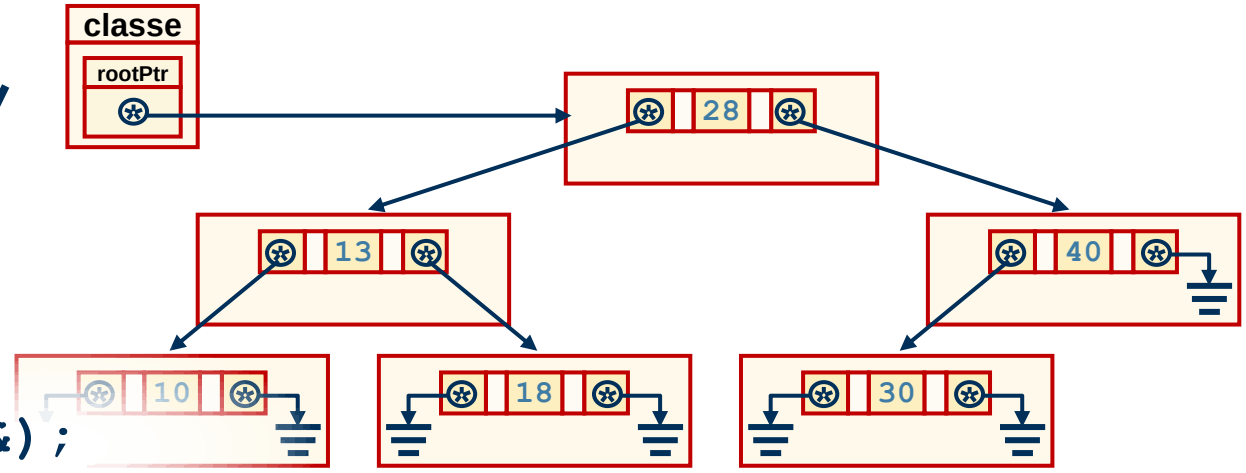


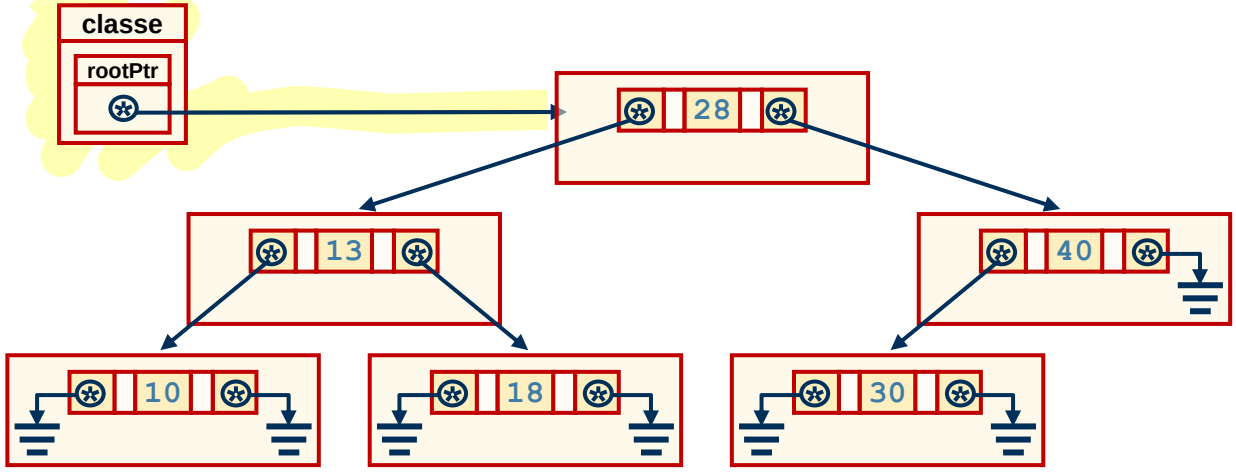

```
class Tree
{
  public:
     Tree();
     Tree(const Tree &);
     ...
  private:
     ...
};z

void main()
{ ...
  Tree classe;
}
```

```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
  ...
  Tree classeBis(classe);
}
```



```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
  ...
  Tree classeBis(classe);
}
```





```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    ...
  private:
    ...
};

void main()
{ ...
  Tree classe;
  ...
  Tree classeBis(classe);
}
```

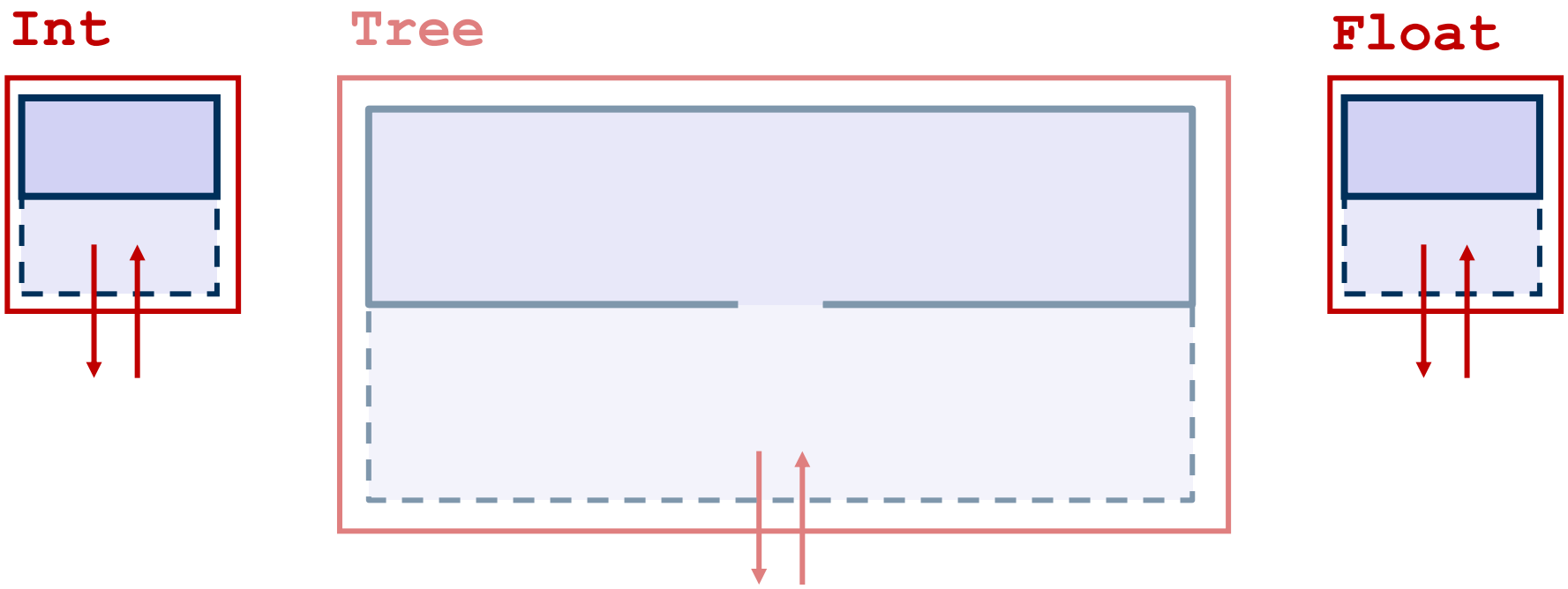
```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    ...
  private:
    ...
};
```

matricola	cognome	nome	sesso	dataNascita		
				giorno	mese	anno
7	Rossi	Marco	M	5	10	75
48	Neri	Anna	F	4	7	75
63	Verdi	Remo	M	5	8	76
84	Gialli	Carla	F	5	11	75

```
class Tree
{
  public:
    Tree();
    Tree(const Tree &);
    Tree(const Studente[]);
    ...
  private:
    ...
};
```

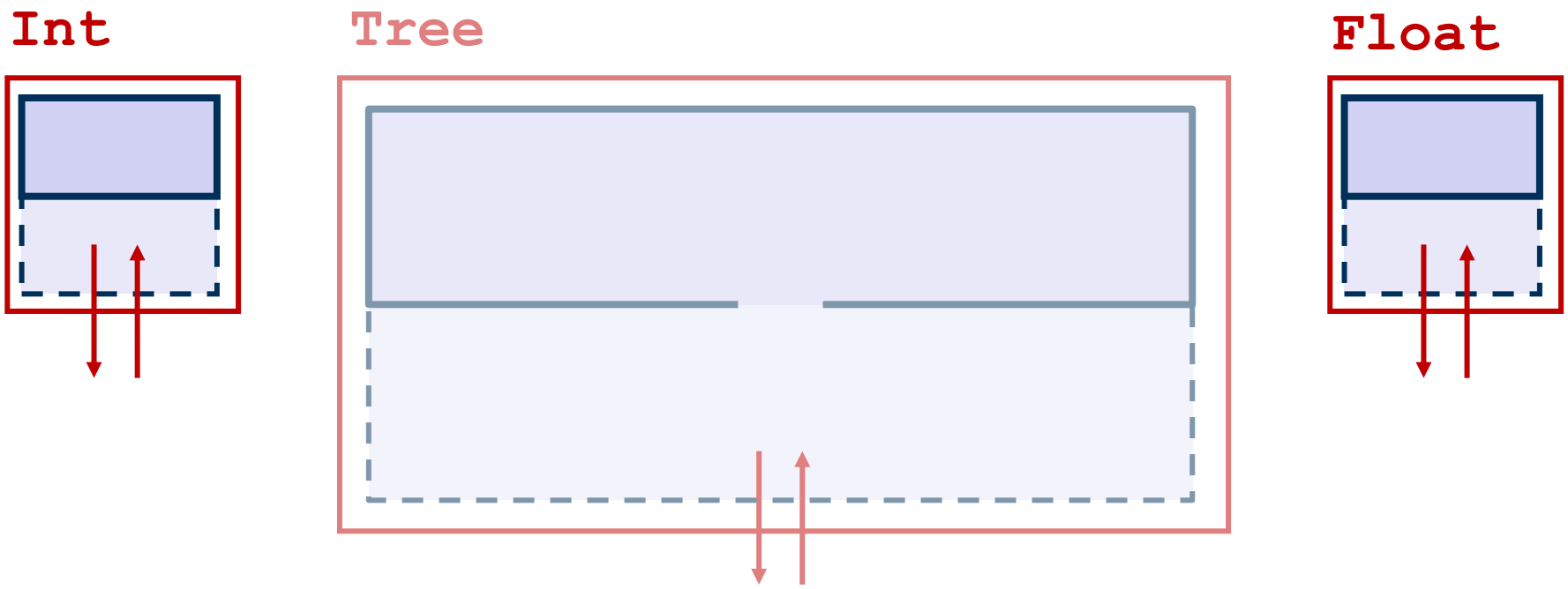
matricola	cognome	nome	sesso	dataNascita		
				giorno	mese	anno
7	Rossi	Marco	M	5	10	75
48	Neri	Anna	F	4	7	75
63	Verdi	Remo	M	5	8	76
84	Gialli	Carla	F	5	11	75

Operatori



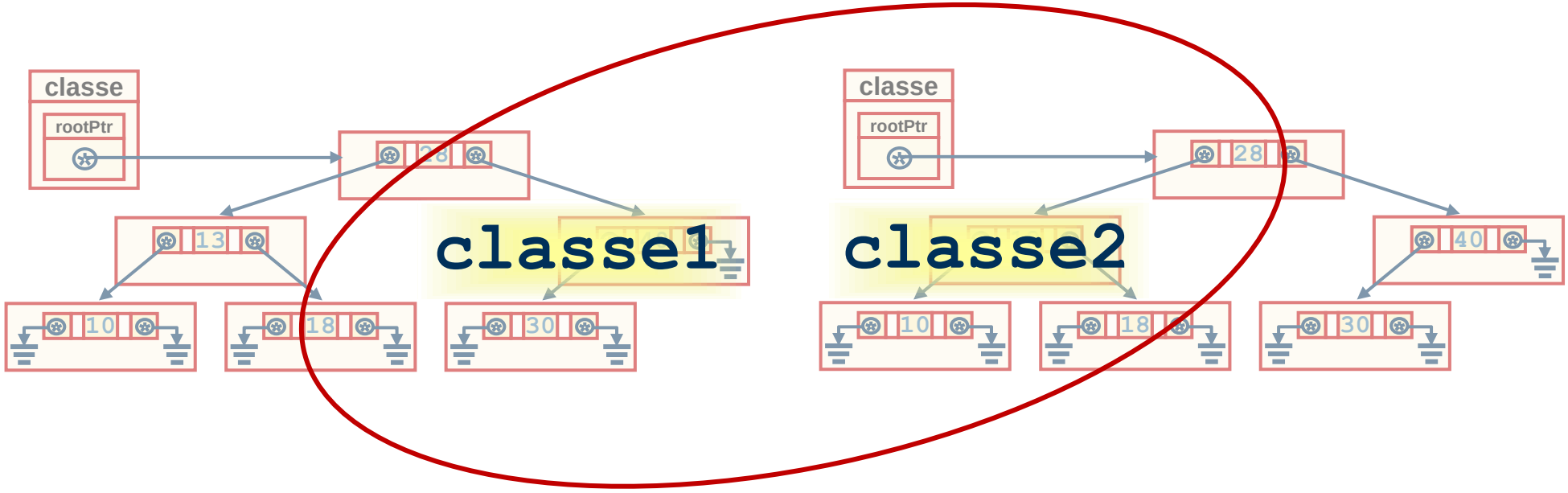

```
a = b + c
```

Overloading di operatori




```
a = b + c
```

Operatore +


classe
rootPtr
28
13
10
18
30
classe1
classe
rootPtr
28
40
10
18
30
classe2

Operatore +

```
classe1 + classe2
```

Sovraccaricare l'operatore +

```
classe1 + classe2
```

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    ...
  private:
    ...
};
```

classe1 + classe2

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ...

  private:
    ...
};
```

```
classe1 = classe2
```

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ...

  private:
    ...
};
```

```
classe1.operator+(classe2);
```

```
classe3 = classe1 + classe2
```

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ...

  private:
    ...
};
```

```
classe1 = classe2
```

Compilatore C++

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ...

  private:
    ...
};
```

classe3 = classe1 + classe2

?

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ...

  private:
    ...
};
```

```
classe1.operator+(classe2);
```

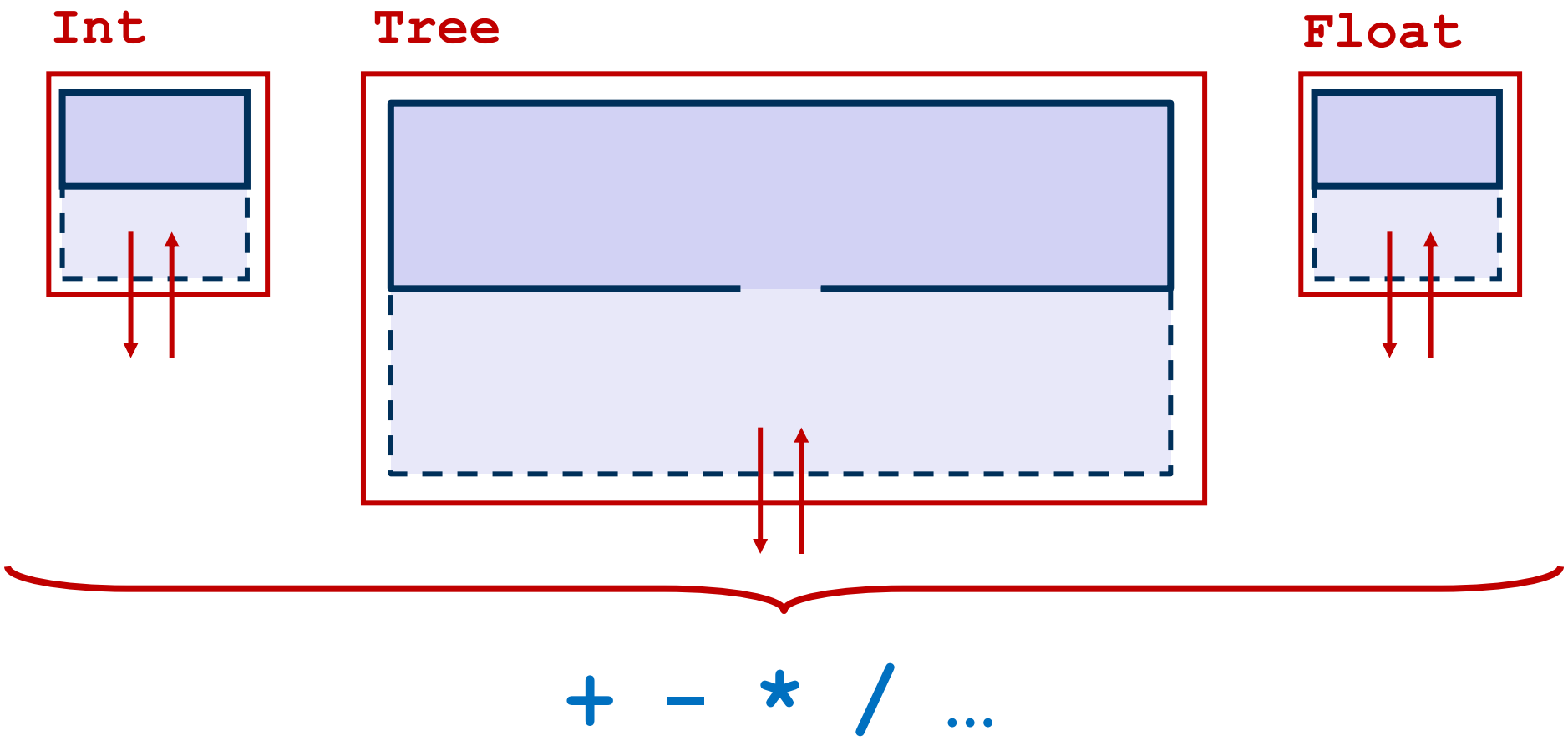
```
classe3 = classe1 + classe2
```

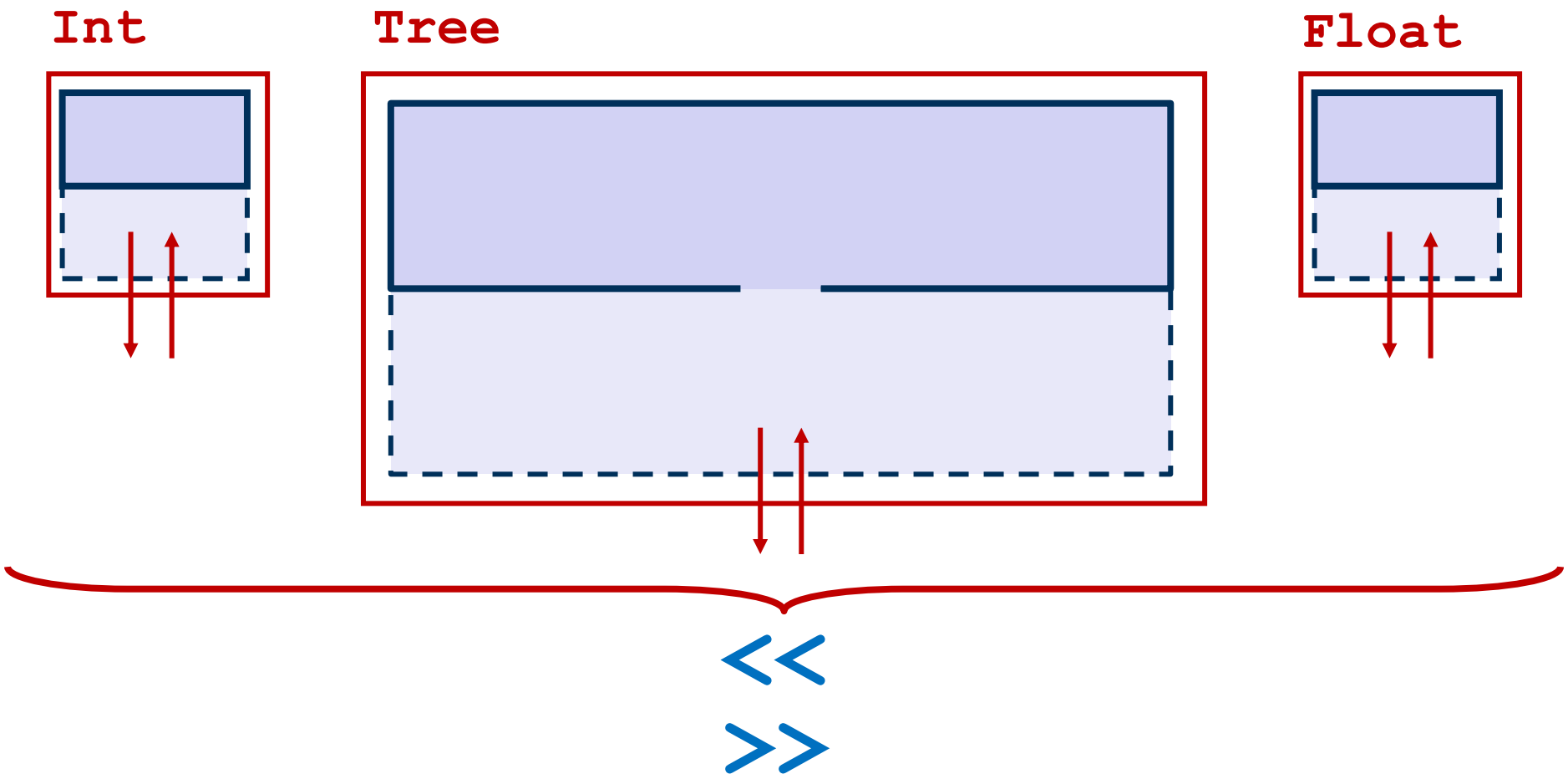
```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ...

  private:
    ...
};
```

```
classe3.operator=(classe1.operator+(classe2));
```

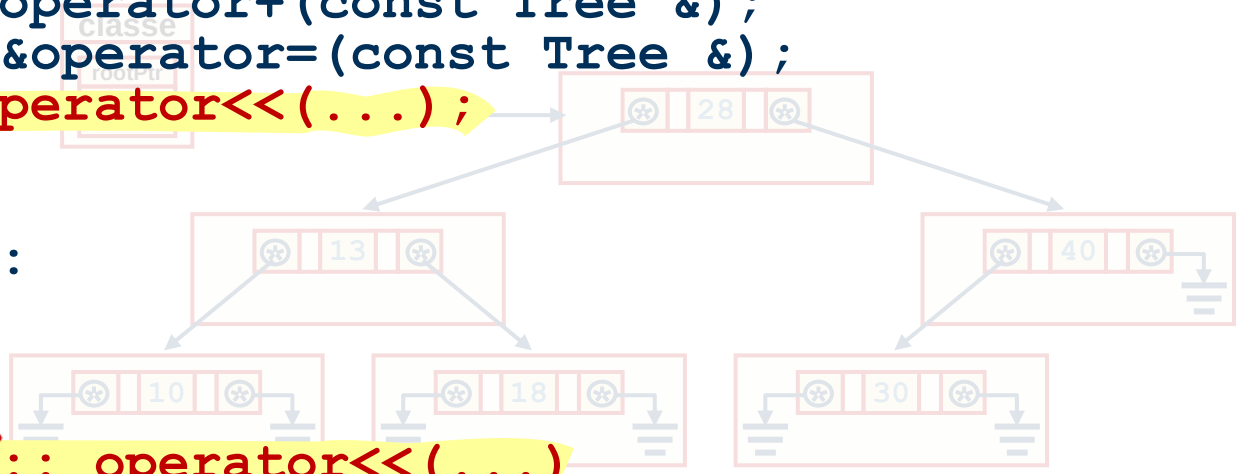
classe3 = classe1 + classe2

Significato "naturale"




Int
Tree
Float
<<
>>

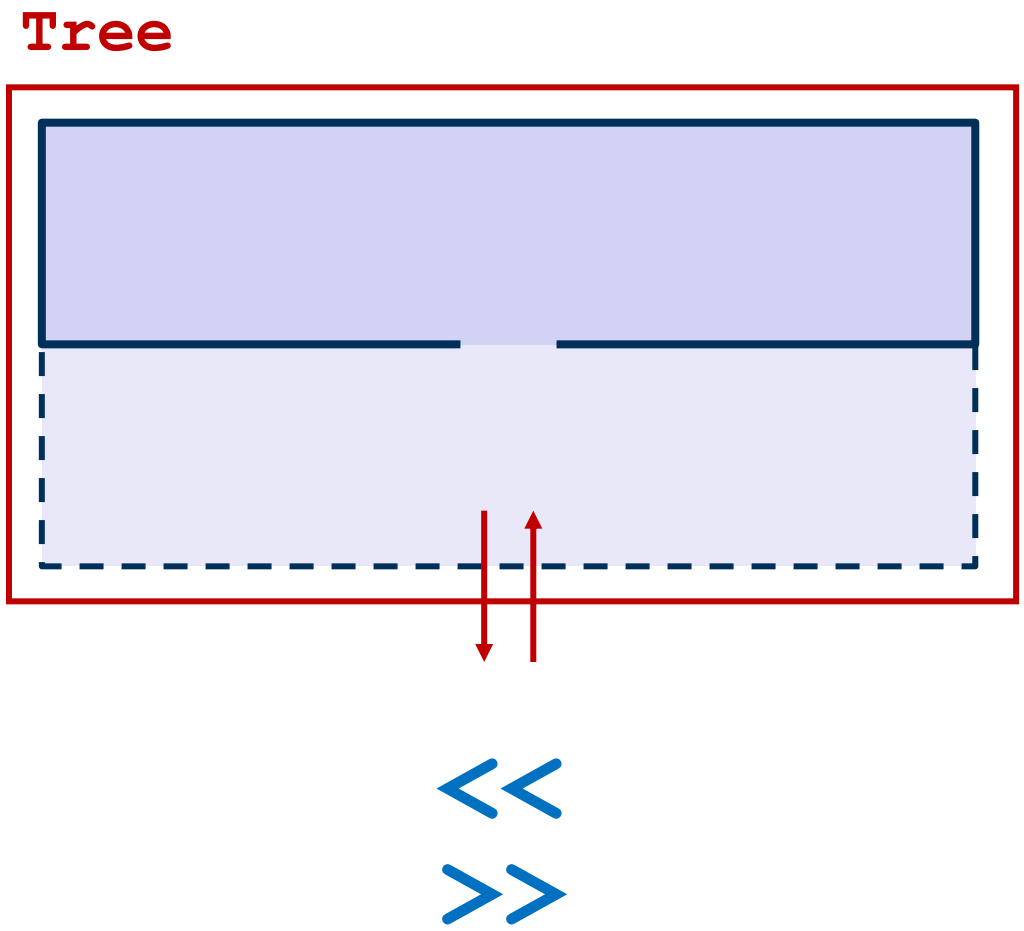
```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ... operator<<(...);
    ...

  private:
    ...
};

... Tree :: operator<<(...)
{
  ...
}
```



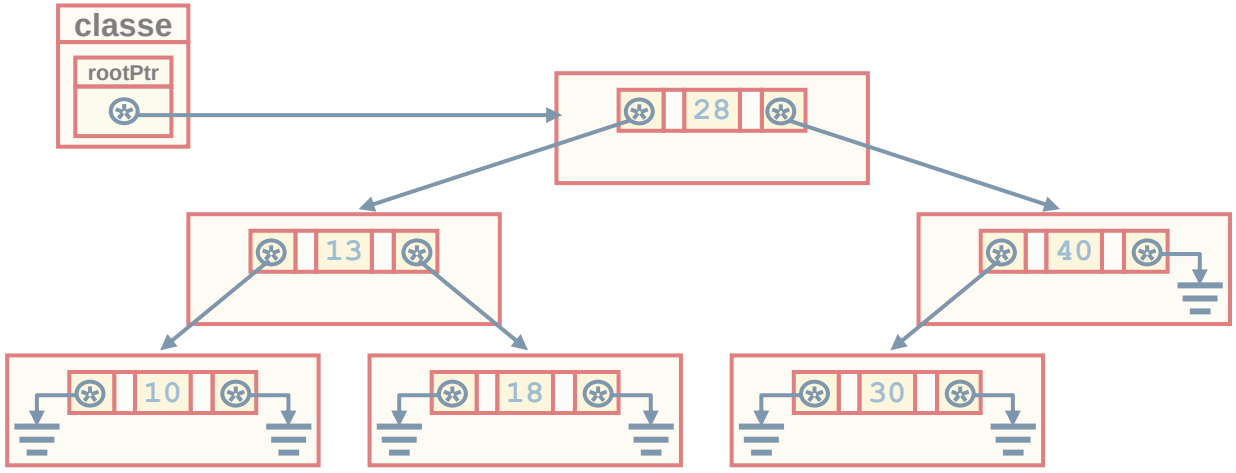
<<

>>

Overloading di << e >>



Operatori binari



<<

>>

Operatori binari

```
cin >> alfa;
cout << beta;
```

operando *operatore* *operando*

```
cin >> alfa;
cout << beta;
```

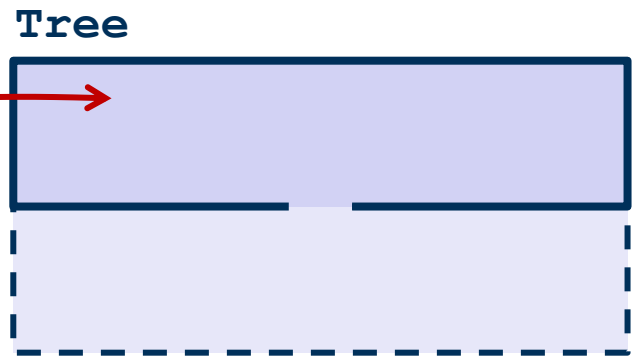
Tree operatore operando

```
cin >> alfa;
cout << beta;
```

Tree operatore operando

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    Tree &operator=(const Tree &);
    ... operator<<(...);
    ...

  private:
    ...
};

... Tree :: operator<<(...)
{
  ...
}
```

```
ostream &operator<<(...)
{
   ...
}

class Tree
{ friend ostream &operator<<(...);
  public:
     Tree();
     ...

  private:
     ...
};
```



```
ostream &operator<<(...)
{
  ...
}

class Tree
{ friend ostream &operator<<(...);
  public:
    Tree();
    ...

  private:
    ...
};
```

cin → istream

cout → ostream

```
ostream &operator<<(...)
{
  ...
}

class Tree
{ friend ostream &operator<<(...);
  public:
    Tree();
    ...

  private:
    ...
};
```

```
ostream &operator<<(  ostream & output, ...)
{
  ...
}

class Tree
{ friend ostream &operator<<(   ostream &, ...);
  public:
    Tree();
    ...

  private:
    ...
};
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
  ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
    Tree();
    ...

  private:
    ...
};
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
cout << classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
  ...
}

class Tree
{ friend ostream &operator<<(  ostream &,  const Tree &);
  public:
    Tree();
    ...

  private:
    ...
};
```

```
cout << a << b << c;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
operator <<(cout, classe1)
cout <<  classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(  ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
operator <<(cout, classe1)
cout <<  classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
operator <<(cout, classe1)
cout <<  classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
operator <<(cout, classe1)
cout <<  classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

```
operator <<(cout, classe1)
cout <<  classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
}

class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
    Tree();
    ...

  private:
    ...
};
```

```
cout <<  classe1;
```

```
ostream &operator<<( ostream & output, const Tree &tab)
{
  ...
  return output;
}
class Tree
{ friend ostream &operator<<( ostream &, const Tree &);
  public:
    Tree();
    ...

  private:
    ...
};
```

```
cout << classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
   return output;
}
class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

cout

```
cout <<  classe1;
```

```
ostream &operator<<( ostream & output, const Tree &tab)
{
  ...
  return output;
}
class Tree
{ friend ostream &operator<<( ostream &, const Tree &);
  public:
    Tree();
    ...

  private:
    ...
};
```

cout

```
cout << classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
   return output;
}
class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

cout

```
cout <<  classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
   return output;
}
class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

cout

```
cout << classe1;
```

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
   return output;
}
class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...

  private:
     ...
};
```

cout

cout << classe1;

```
ostream &operator<<(  ostream & output,  const Tree &tab)
{
   ...
   return output;
}
class Tree
{ friend ostream &operator<<(   ostream &,  const Tree &);
  public:
     Tree();
     ...
  private:
     ...
};
```

cout

cout

```
(cout << classe1) << classe2;
```

```
cout << classe1 << classe2 << classe3;
```

```
cout << classe1 << classe2 << classe3;
```

```
classe1 + classe2 + classe3
```

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    ...

  private:
    ...
};
```

`classe1 + classe2 + classe3`

```
class Tree
{
  public:
    Tree();
    Tree operator+(const Tree &);
    ...

  private:
    ...
};
```

classe1 + classe2 + classe3

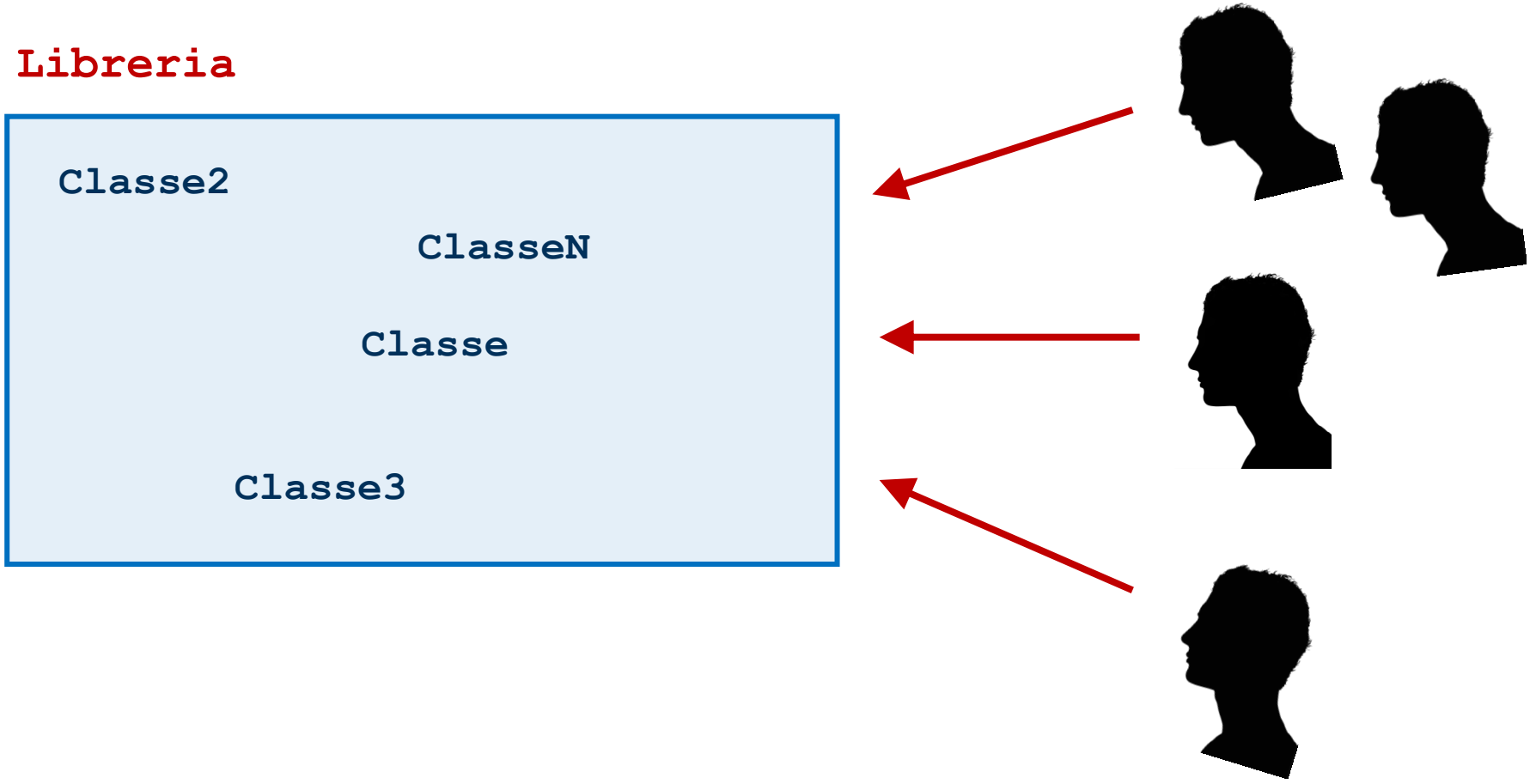
Conclusioni

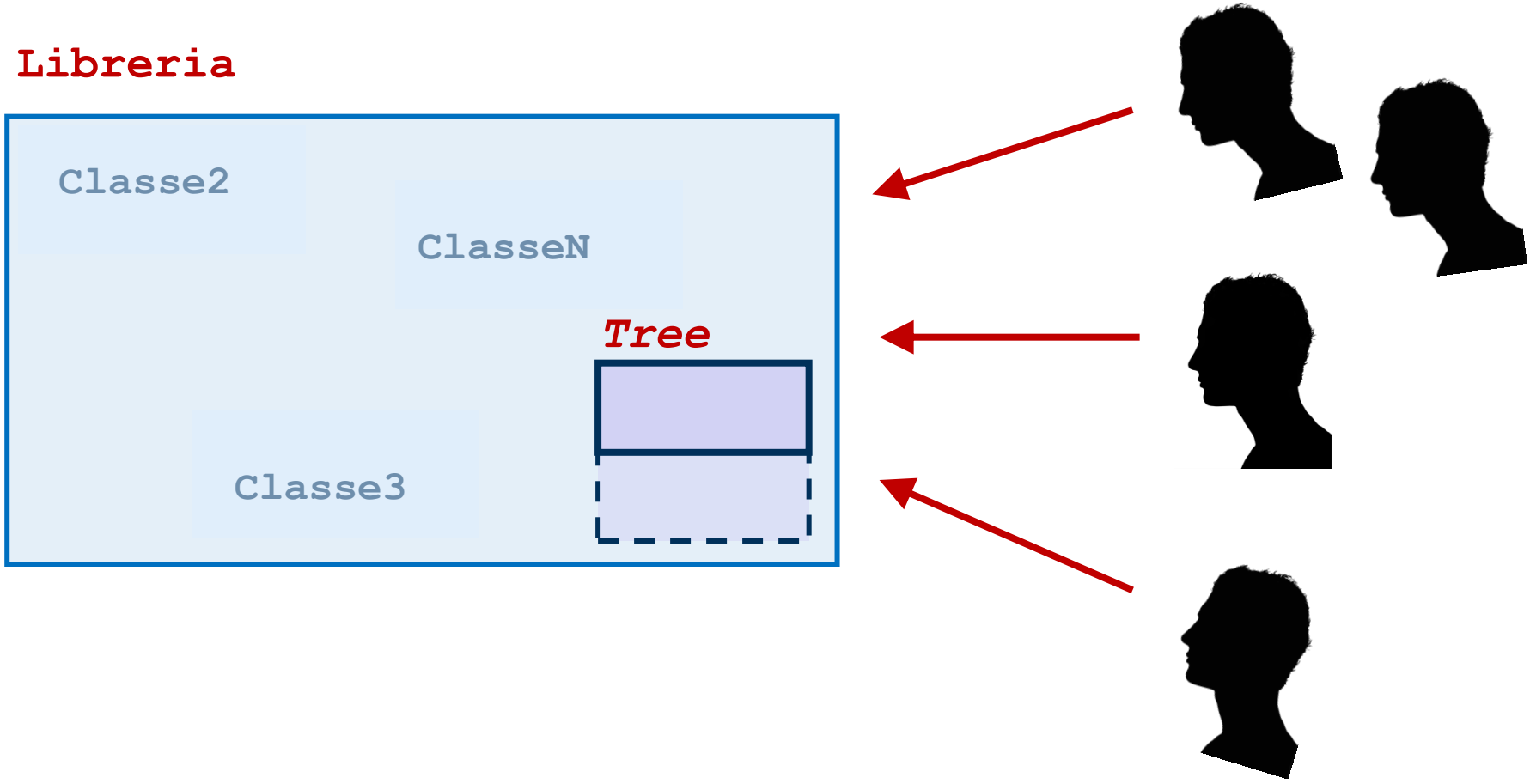
1. inserisci 4 monete da 100 lire
2. tira il cassetto corrispondente alla marca desiderata
3. estrai il pacchetto
4. richiudi il cassetto
5. ritira il resto
6. fine


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

Tree :: Tree();
{
  rootPtr = 0;
}

void  Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
  inserisciConRicorsione(rootPtr, nuovoStudente);
}

voit Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                    Studente nuovoStudente)
{ if (ptr == 0)
    { ptr = new TreeNode(NuovoStudente);
    }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
   cout << "studente di matricola " << nuovoStudente.matricola
        << " già presente" << endl;
}
```

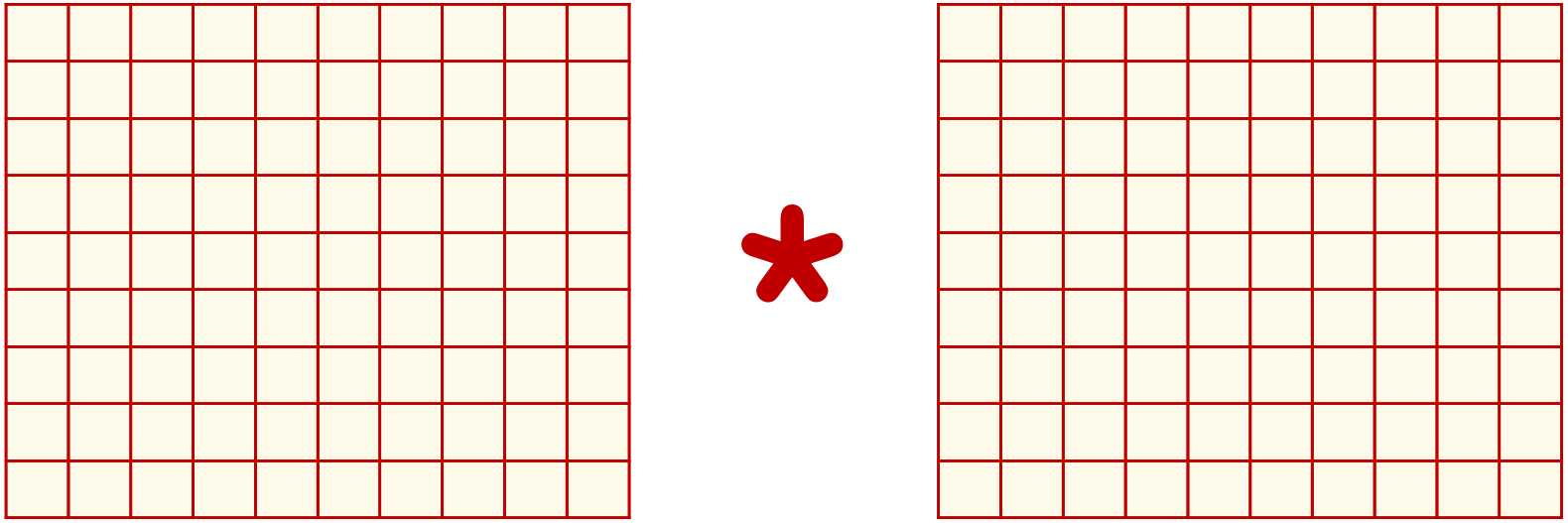
Classe

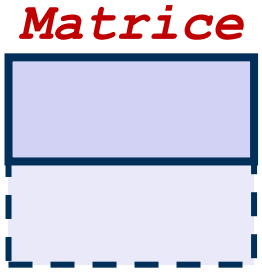
```
class Tree
{
  public:
    Tree();
    void inserisciSeNonEsiste(Studente);
    void inOrderTraversal();
    void stampaAlbertoRovesciato();
  private:
    TreeNode *rootPtr;
  //funzioni di servizio
    void inserisciConRicorsione(TreeNode *&, Studente);
    void recursiveInOrder(TreeNode *);
 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

Tree :: Tree();
{
  rootPtr = 0;
}

void  Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
   inserisciConRicorsione(rootPtr, nuovoStudente);
}

voit Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                    Studente nuovoStudente)
{ if (ptr == 0)
     { ptr = new TreeNode(NuovoStudente);
     }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
   cout << "studente di matricola " << nuovoStudente.matrico
        << " già presente" << endl;
}
```

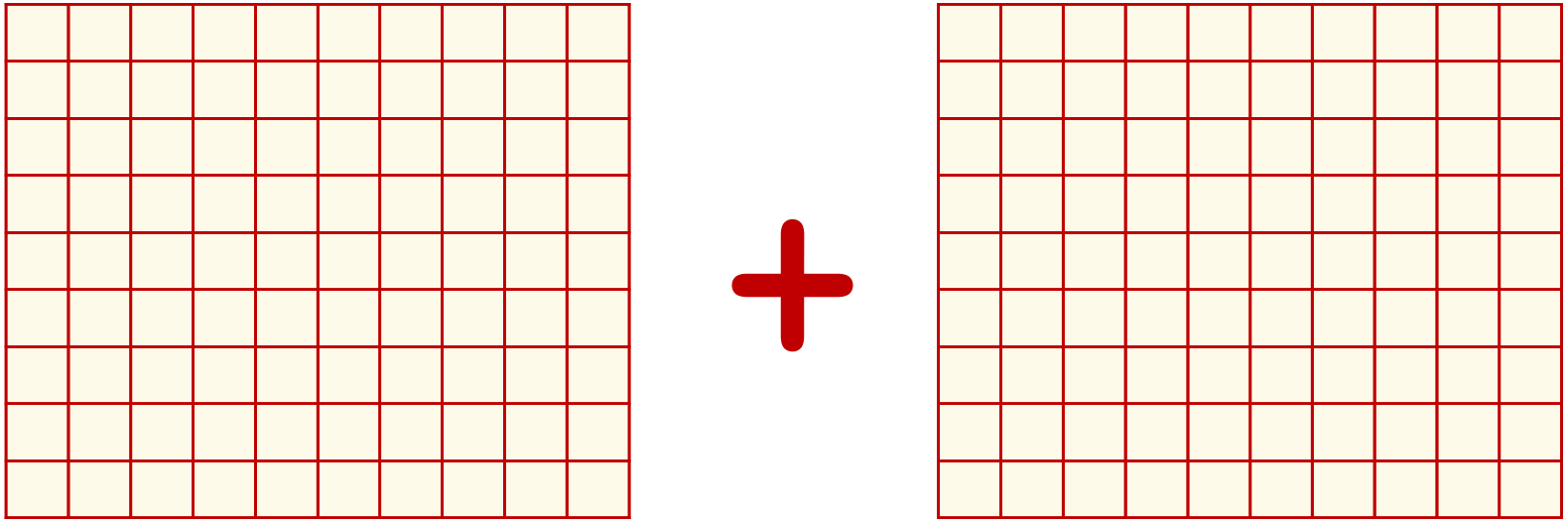

Libreria
Classe2
ClasseN
Classe
Classe3

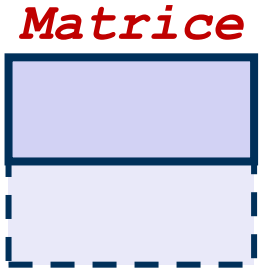

Libreria
Classe2
ClasseN
Tree
Classe3

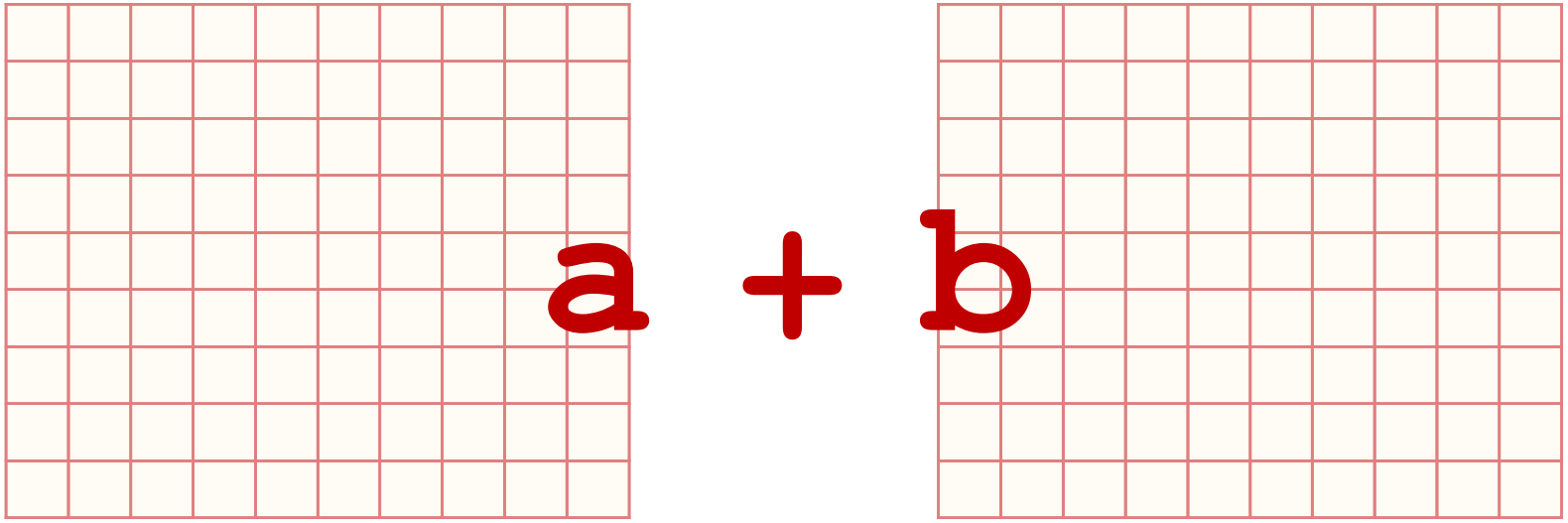
Matrici

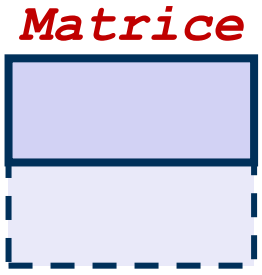


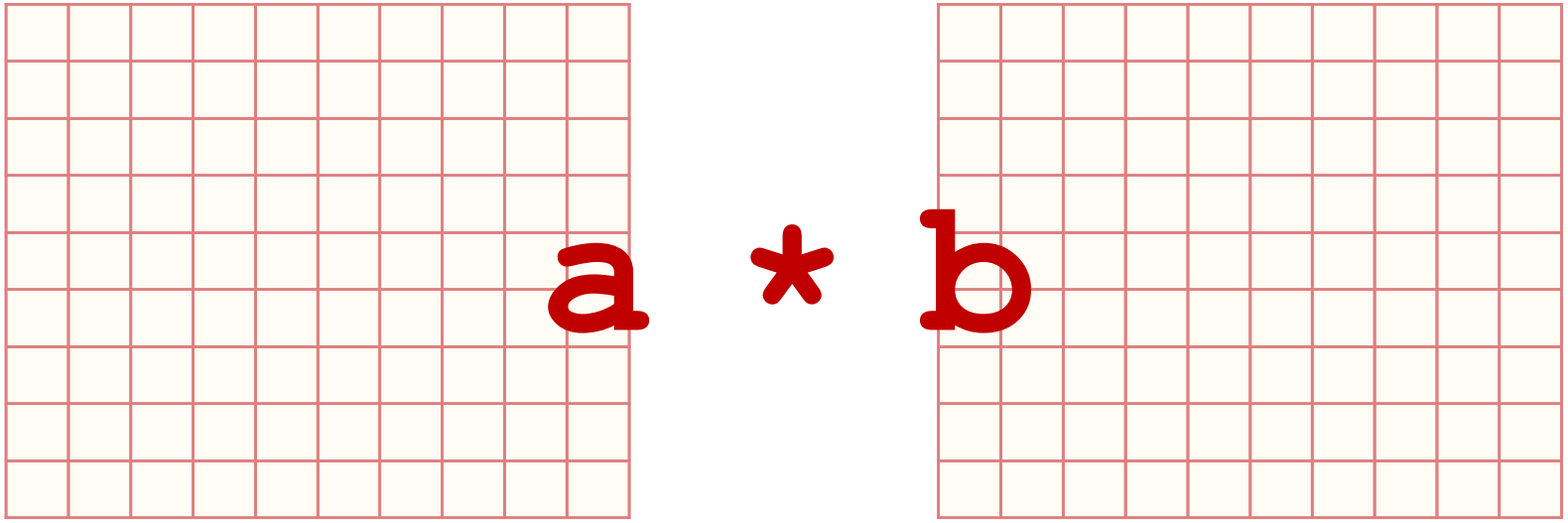


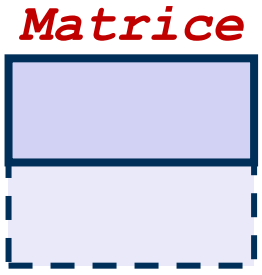



Matrice


a + b


Matrice


a * b


Matrice

Uso "naturale"