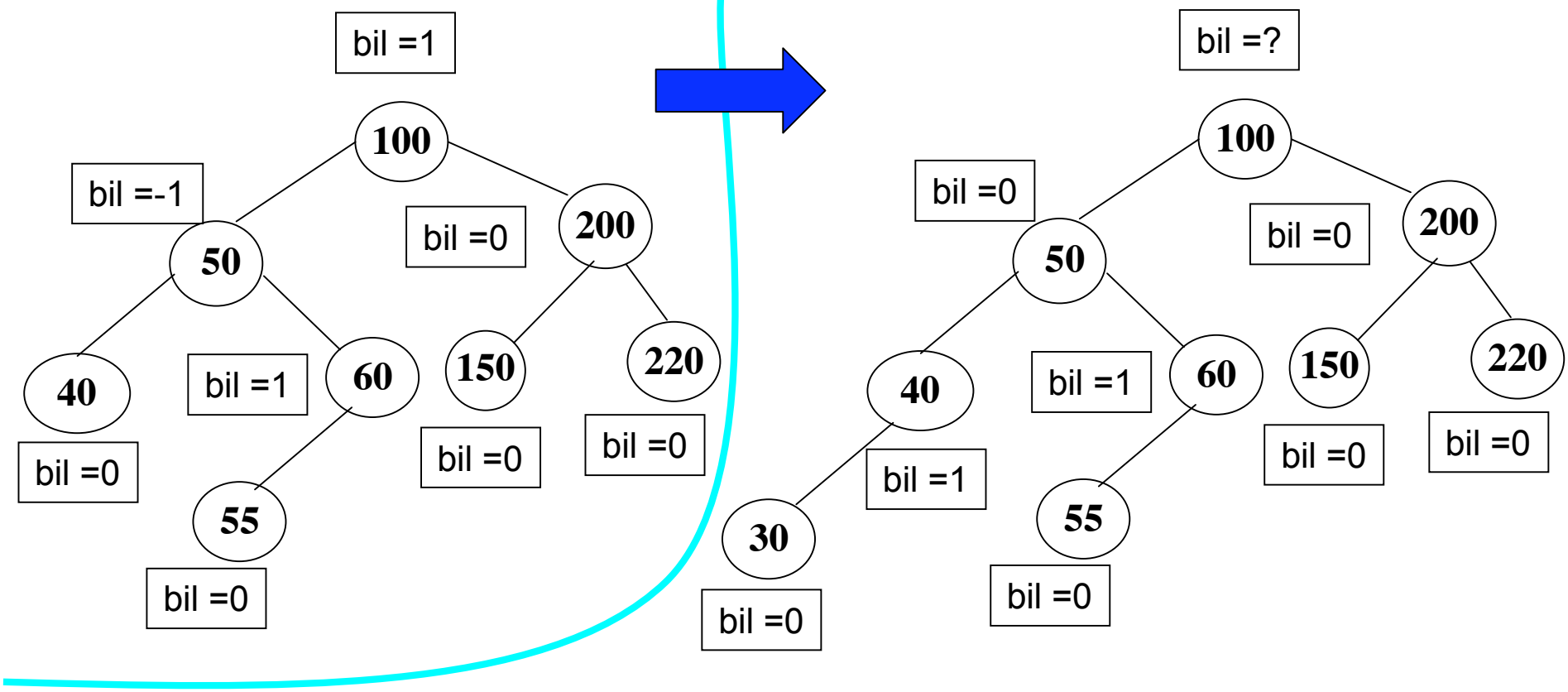
## Inserimento in un ABR n-bilanciato

- Un esempio di mutua ricorsione

- con gli alberi bilanciati in altezza, proprietà più debole, si ottiene maggiore semplicità delle operazioni di ribilanciamento (niente mutua ricorsione)

# Inserimento in un ABR n-bilanciato

**L'inserimento di 30 sbilancia l'albero a sinistra**



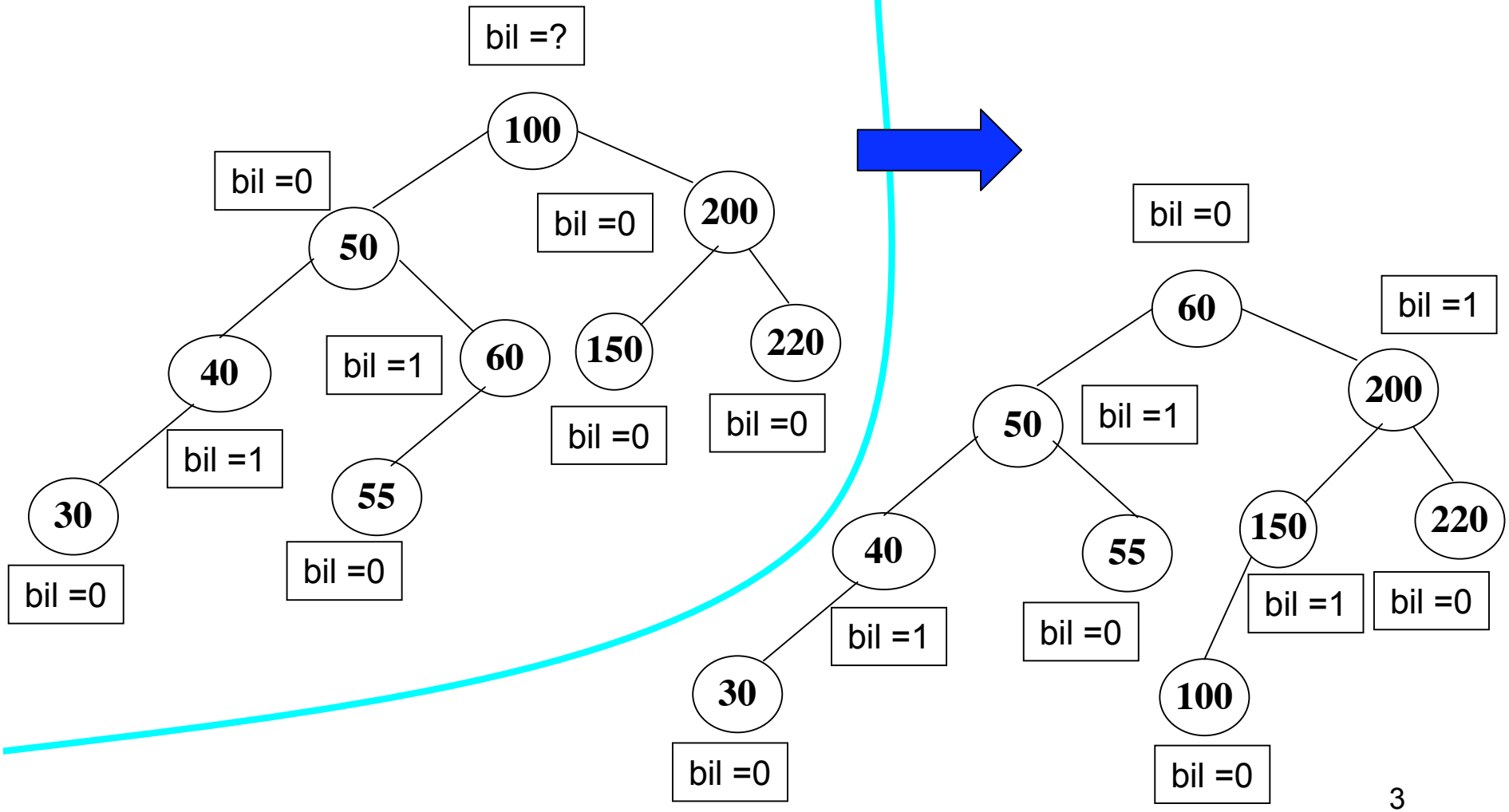

**/*bil = 1 vuol dire che c'è un nodo in più nel sottoalbero sinistro,**

**bil == 0 i due sottoalberi hanno lo stesso numero di nodi**

**bil = -1 vuol dire che c'è un nodo in più nel sottoalbero destro*/**

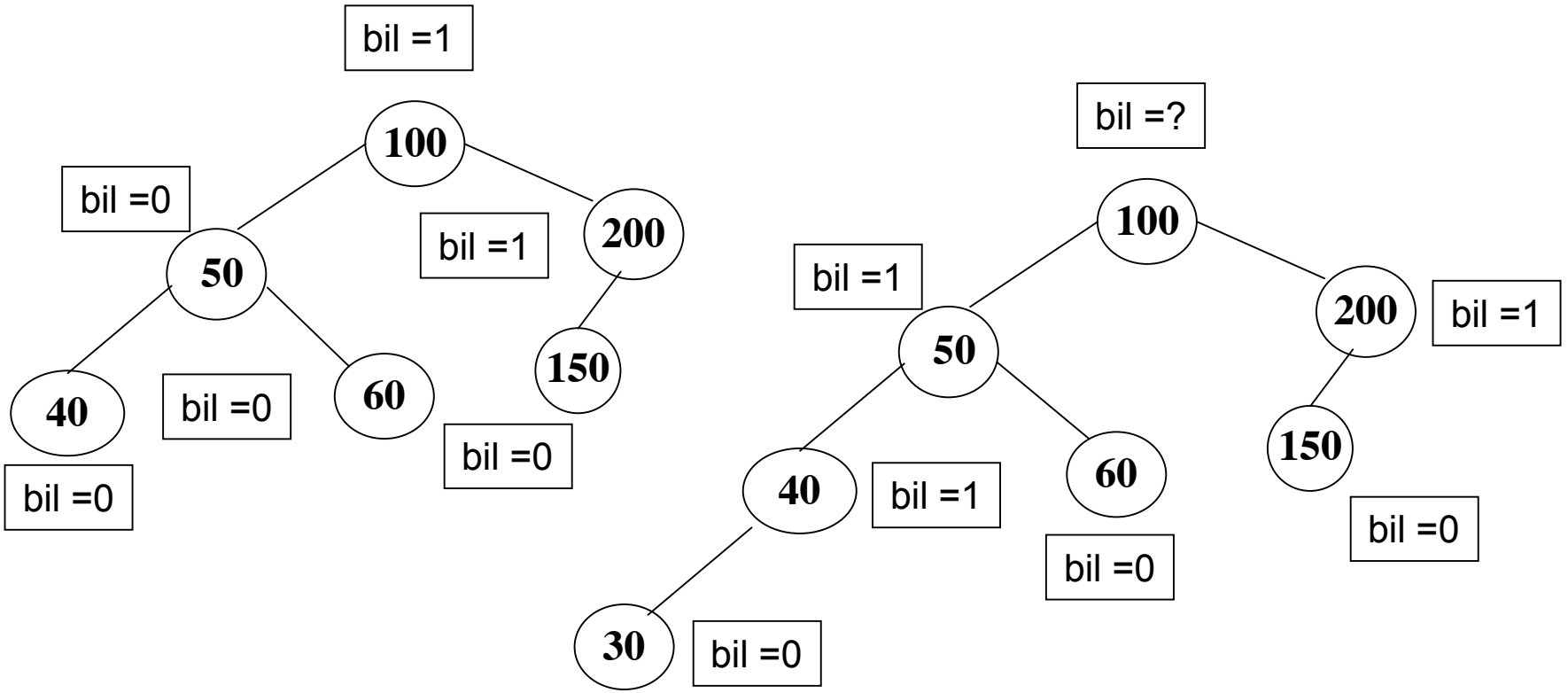
# Inserimento in un albero n-bilanciato: ribilanciamento

**Si cancella il massimo, 60, nel sottoalbero sinistro e se ne copia il valore nella radice, mentre la radice va nel sottoalbero destro.**



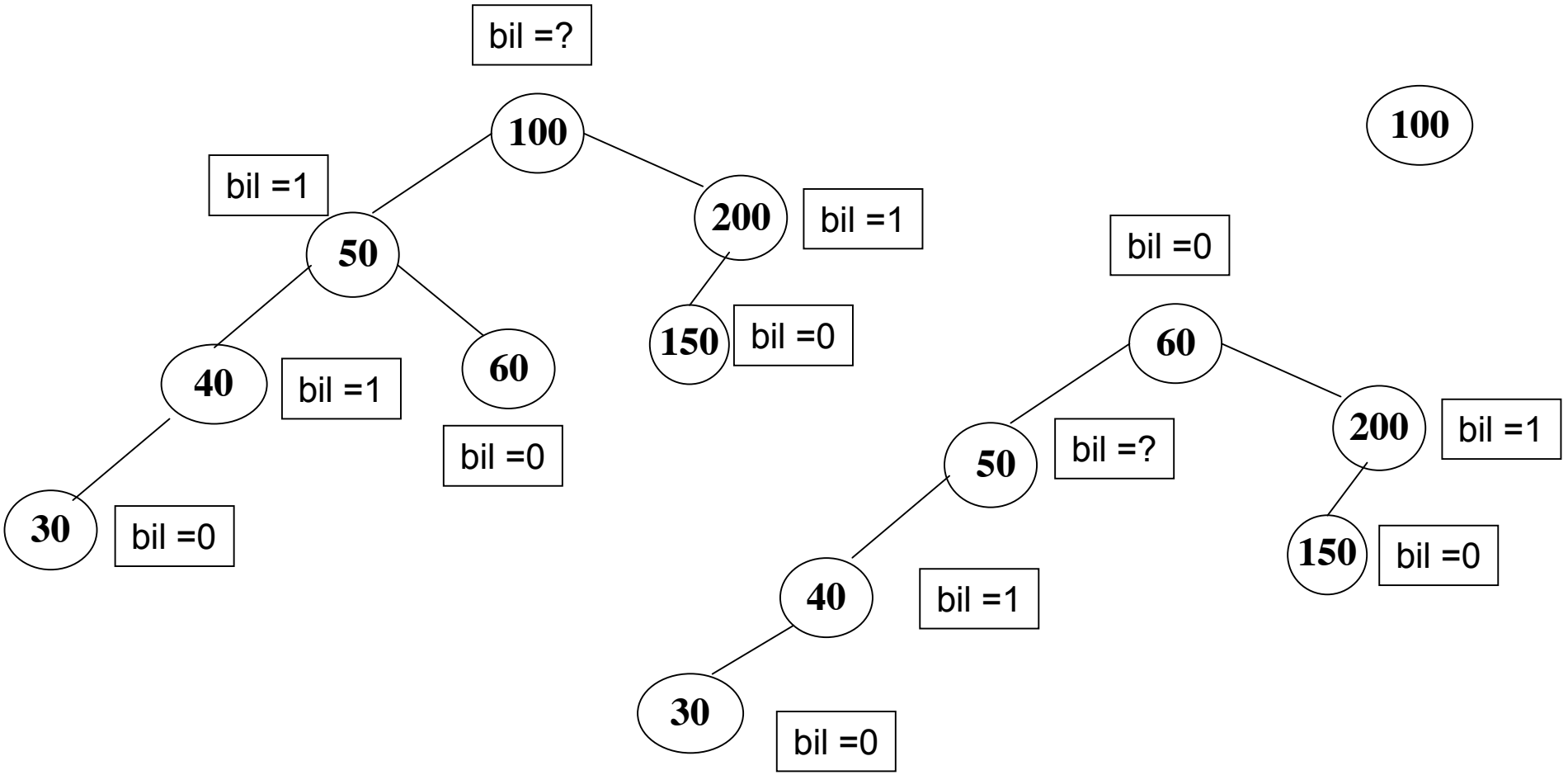
# Inserimento in un albero n-bilanciato: caso 2

**L'inserimento di 30 sbilancia l'albero a sinistra**



# Inserimento in un albero n-bilanciato: caso 2

**Per ribilanciare l'albero, si cancella il massimo, 60, nel sottoalbero sinistro e se ne copia il valore nella radice, mentre la radice va nel sottoalbero destro.**

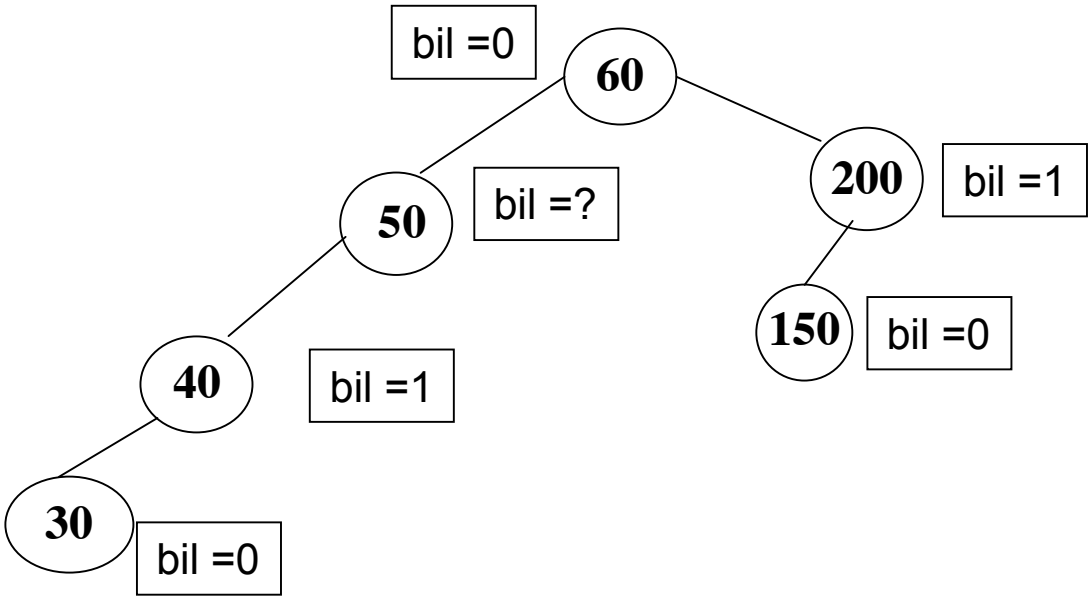


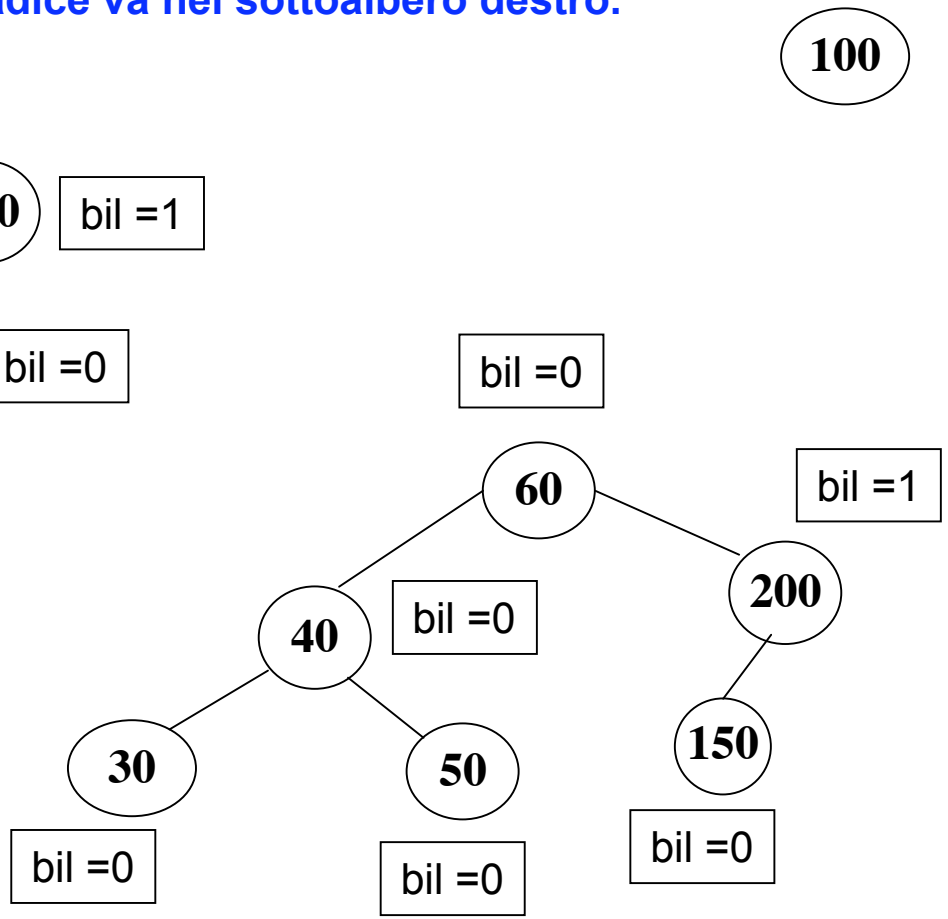



**Occorre ribilanciare anche il sottoalbero radicato in 50!**

# Inserimento in un albero n-bilanciato: caso 2

**Il ribilanciamento si effettua nello stesso modo previsto nel caso dell'inserimento: si cancella il massimo, 40, nel sottoalbero sinistro e se ne copia il valore nella radice, mentre la radice va nel sottoalbero destro.**

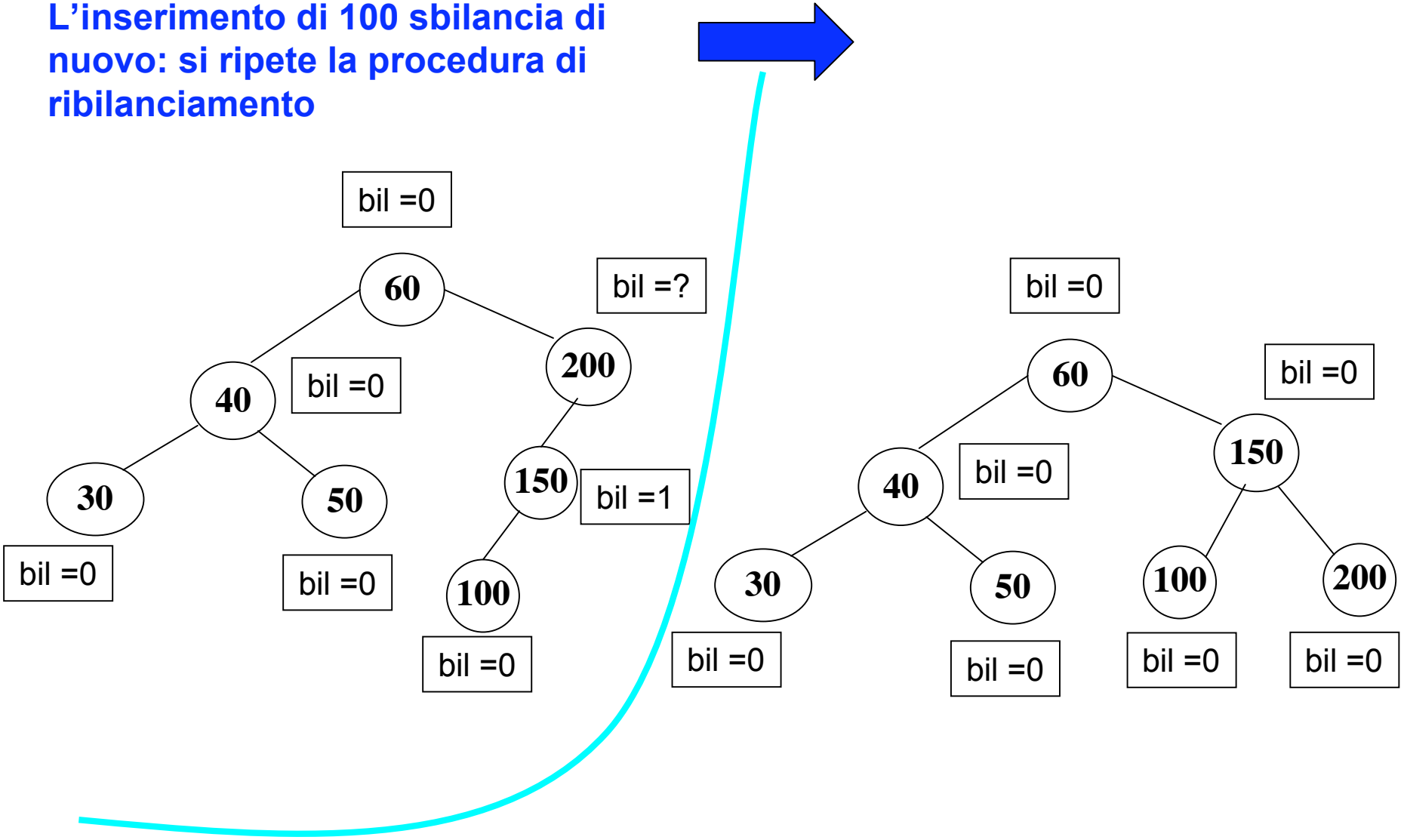



**Ora si deve inserire 100 nel sottoalbero destro**

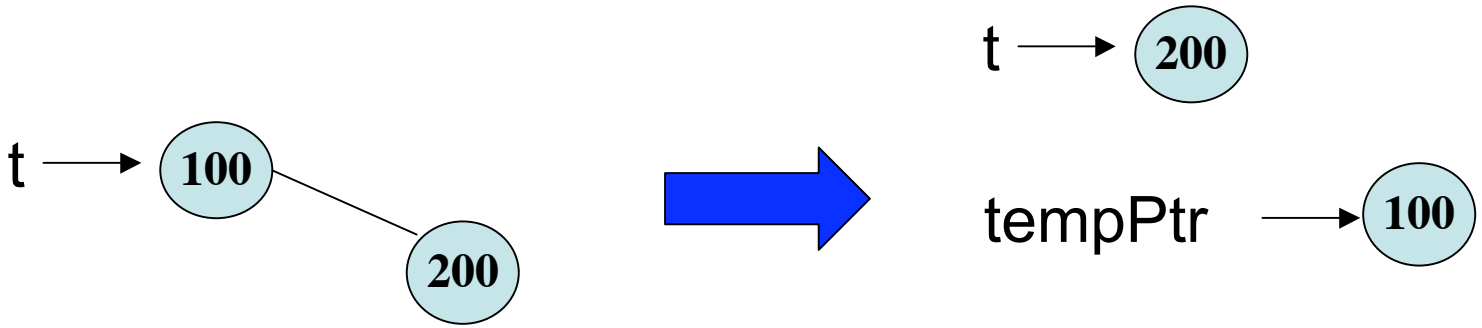
# Inserimento in un albero n-bilanciato: caso 2

**L'inserimento di 100 sbilancia di nuovo: si ripete la procedura di ribilanciamento**



# Cancellazione minimo in un ABR n-bilanciato

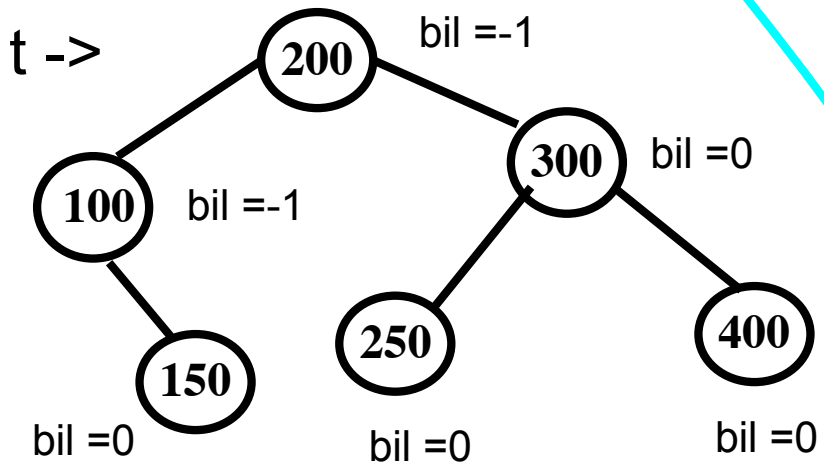
caso base

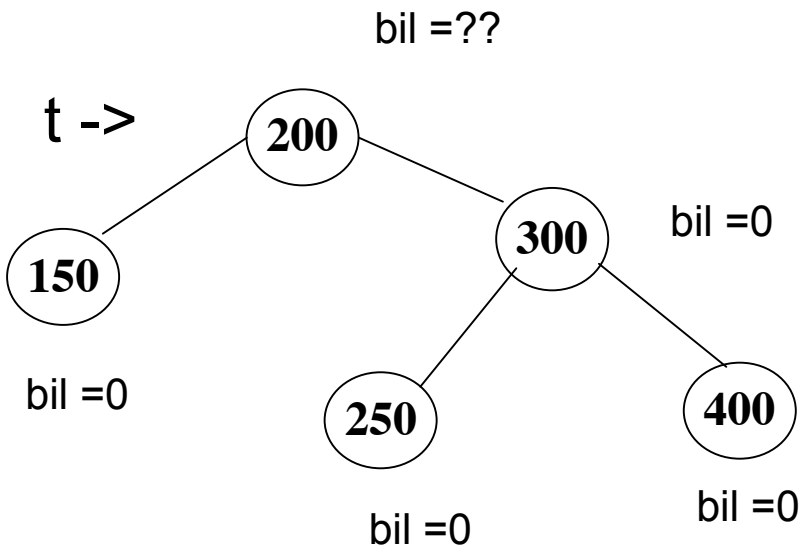



tempPtr è il puntatore d'appoggio per il minimo, che dovrà essere restituito dalla funzione

# Cancellazione minimo in un ABR n-bilanciato

1 passo ricorsivo



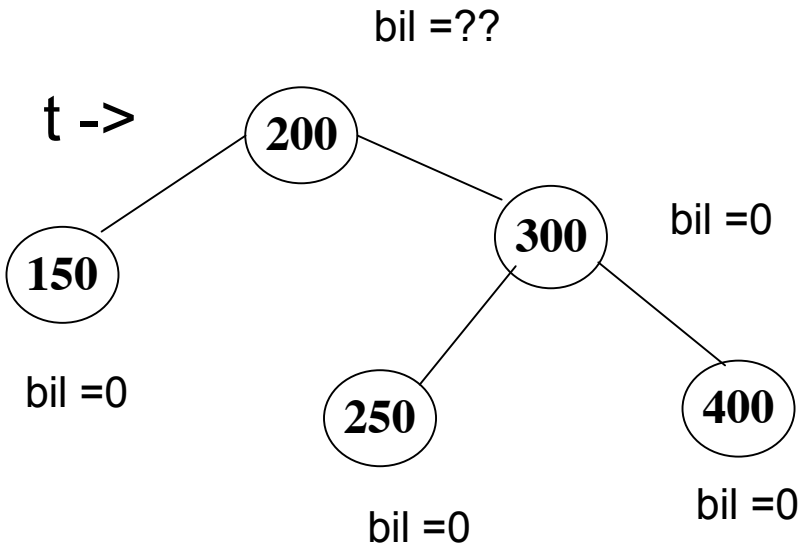



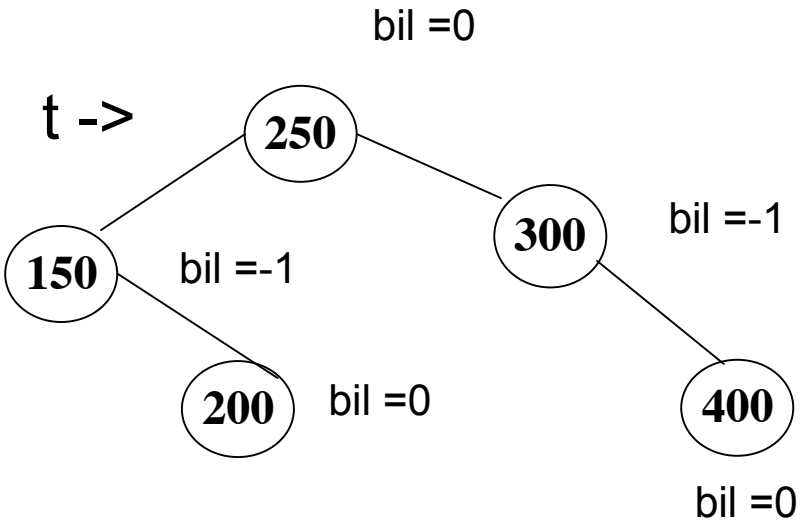

tempPtr -> 100

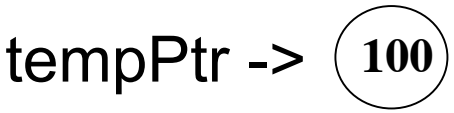
c’e’ uno sbilanciamento a destra!

# Cancellazione minimo in un ABR n-bilanciato

**Per ribilanciare l'albero, si cancella il minimo, 250, nel sottoalbero destro e se ne copia il valore nella radice, mentre la radice va nel sottoalbero sinistro.**



# Alberi binari di ricerca n-bilanciati: i tipi

```
enum FB {SBIL = -1,BIL =0, SBIR = 1};
/*SBIL = 1 vuol dire che c'è un nodo in più nel
sottoalbero  sinistro,
BIL == 0 i due sottoalberi hanno lo stesso numero di nodi
SBIR = -1 vuol dire che c'è un nodo in più nel
sottoalbero destro*/
struct nodo {
  int elem;
  enum FB bil;
  struct nodo *lPtr;
  struct nodo *rPtr;
                  };

typedef struct nodo Nodo;
typedef Nodo* TreePtr;
```

Il codice

```
int InsAlbBinRicBil( TreePtr *t, int nuovo )
/* inserisce  nuovo in un ABR n-bilanciato t, se non presente
 prec: t !=NULL
 postc: restituisce 1 se l'inserimento è avvenuto, 0 altrimenti */
{int ris;
TreePtr tempPtr = *t;
if (tempPtr == NULL) { *t = creaNodoBil(nuovo); return 1;} /*inserimento*/
if (nuovo == tempPtr->elem ) return 0;  /*già presente*/
if (nuovo < tempPtr->elem  )
  { ris = InsAlbBinRicBil( &(tempPtr->lPtr), nuovo );/* l'elemento va nel sottoalb. sinistro */
   if (ris == 1)
      switch (tempPtr -> bil)
              {case (BIL) : tempPtr -> bil = SBIL;break;
               case (SBIL) : {tempPtr -> bil = BIL;
                              RiBilL(tempPtr);break;}  /*si deve ribilanciare l'albero*/
             case (SBIR) : tempPtr -> bil = BIL;break;}
    }
else  /* l'elemento va inserito nel sottoalbero destro */
{ ris = InsAlbBinRicBil( &(tempPtr->rPtr), nuovo );
  if (ris == 1)
     switch (tempPtr -> bil)
              {case (BIL) : tempPtr -> bil = SBIR;break;
                case (SBIL) : tempPtr -> bil =  BIL;break;
                case (SBIR) : {tempPtr -> bil = BIL;
                               ris = RiBilR(tempPtr);break;}} /*si deve ribilanciare l'albero*/
 }  return ris; }
```

## Le operazioni di ribilanciamento

```
int RiBilR(TreePtr t)
/*ribilancia l'albero t, che ha un nodo in piu'  a destra
postc: cancella il minimo nel sottoalbero destro di t , ne mette il valore
alla radice e restituisce 1 se l'inserimento del valore di t nel sottoalbero
sinistro è riuscito*/
{TreePtr minPtr;
int temp;
minPtr = delMinBil(&(t -> rPtr));
temp = t ->elem;
t -> elem = minPtr -> elem;  /*il minimo va alla radice*/
t -> bil = BIL;
return InsAlbBinRicBil(&(t -> lPtr), temp);} /*l'elemento alla radice
deve andare nel sottoalbero sinistro, ma in modo bilanciato*/
```

# Le operazioni di ribilanciamento

```
int RiBilL(TreePtr t)
/*ribilancia l'albero t, che ha un nodo in piu'  a sinistra
postc: cancella il massimo nel sottoalbero sinistro di t , ne mette
il valore alla radice e restituisce 1 se l'inserimento del valore di t
nel sottoalbero sinistro è riuscito*/
{TreePtr maxPtr;
int temp;
maxPtr = delMaxBil(&(t -> lPtr)); /*il massimo va alla radice*/
temp = t ->elem;
t -> elem = maxPtr -> elem;
t -> bil = BIL;
return InsAlbBinRicBil(&(t -> rPtr), temp);} /*l'elemento alla radice
deve andare nel sottoalbero destro */
```

# La cancellazione del minimo

```
TreePtr delMinBil ( TreePtr* t )
/*prec: t != NULL
postc: restituisce un puntatore al minimo di t , che viene calncellato
dall'ABR n-bilanciato t */
 { TreePtr tempPtr = *t;
if (tempPtr-> lPtr == NULL)
/*tempPtr e'il minimo, perche' non ha il figlio sinistro*/
            { *t = (*t) -> rPtr; return tempPtr;}
else tempPtr =  delMinBil(&(*t) -> lPtr); /* continua la ricerca del minimo*/
/*risalendo verso la radice puo' essere necessario ribilanciare*/
      switch ((*t) -> bil)
           {case (BIL) : (*t) -> bil = SBIR;break;
            case (SBIL) : (*t)  -> bil =  BIL;break;
            case (SBIR) :
                     {(*t)  -> bil = BIL;
                    if ( RiBilR((*t) ) != 1)
                       printf("qualcosa non va nel ribilanciamento") ;break;
                     }
           }
 return tempPtr;  }
```

# La cancellazione del massimo

```
TreePtr delMaxBil ( TreePtr* t )
/*prec: t != NULL
postc: restituisce un puntatore al massimo di t , che viene cancellato
dall'ABR n-bilanciato t */
 { TreePtr tempPtr = *t;
if (tempPtr-> rPtr == NULL)
/*tempPtr e'il massimo, perche' non ha il figlio destr*/
            { *t = (*t) -> lPtr; return tempPtr;}
else tempPtr =  delMaxBil(&(*t) -> lPtr); /* continua la ricerca del minimo*/
/*risalendo verso la radice puo' essere necessario ribilanciare*/
      switch ((*t) -> bil)
           {case (BIL) : (*t) -> bil = SBIL;break;
            case (SBIR) : (*t)  -> bil =  BIL;break;
            case (SBIL) :
                     {(*t)  -> bil = BIL;
                    if ( RiBilL((*t) ) != 1)
                       printf("qualcosa non va nel ribilanciamento") ;break;
                     }
           }
 return tempPtr;  }
```