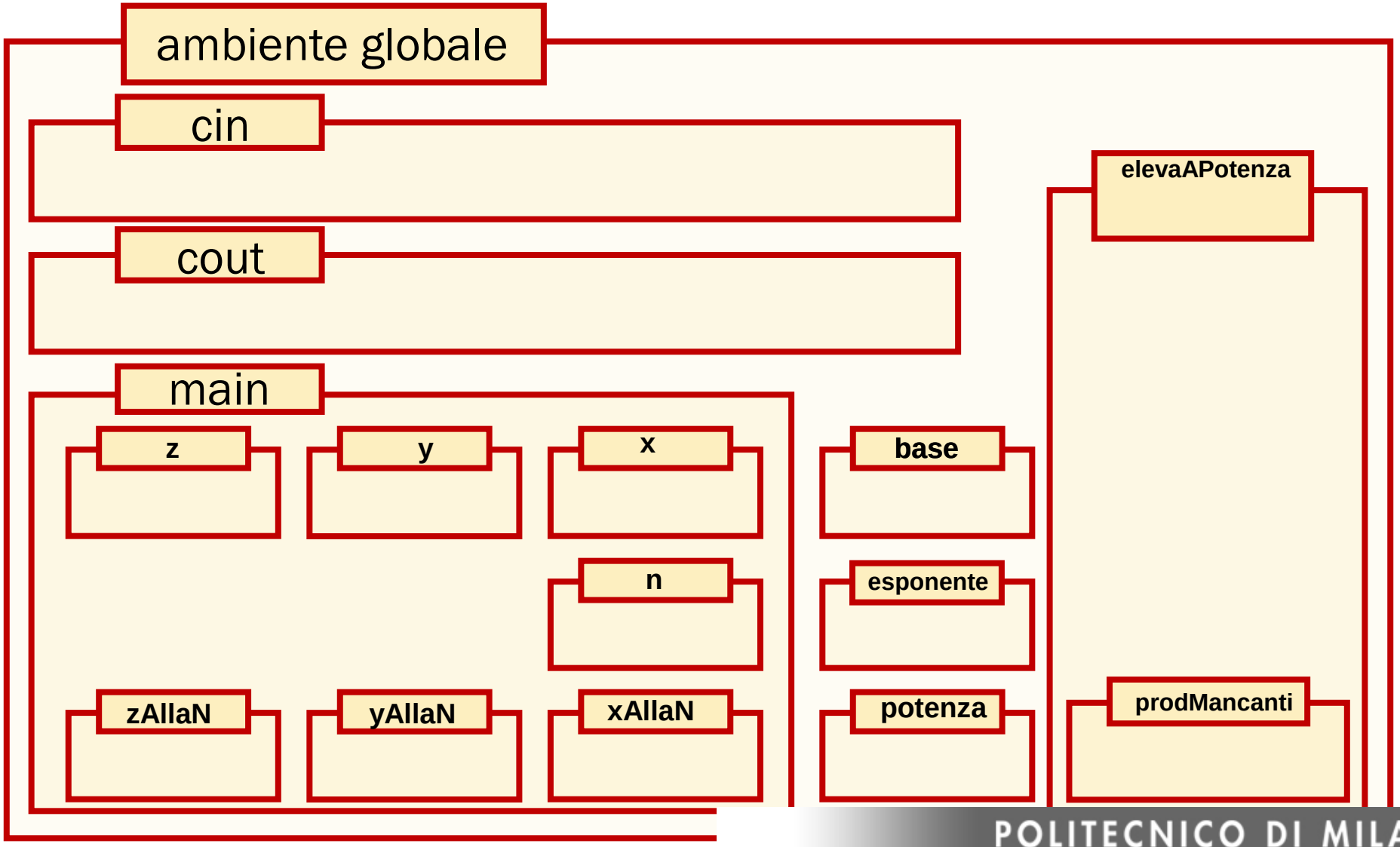
INFORMATICA

Comunicazione tra funzioni mediante parametri

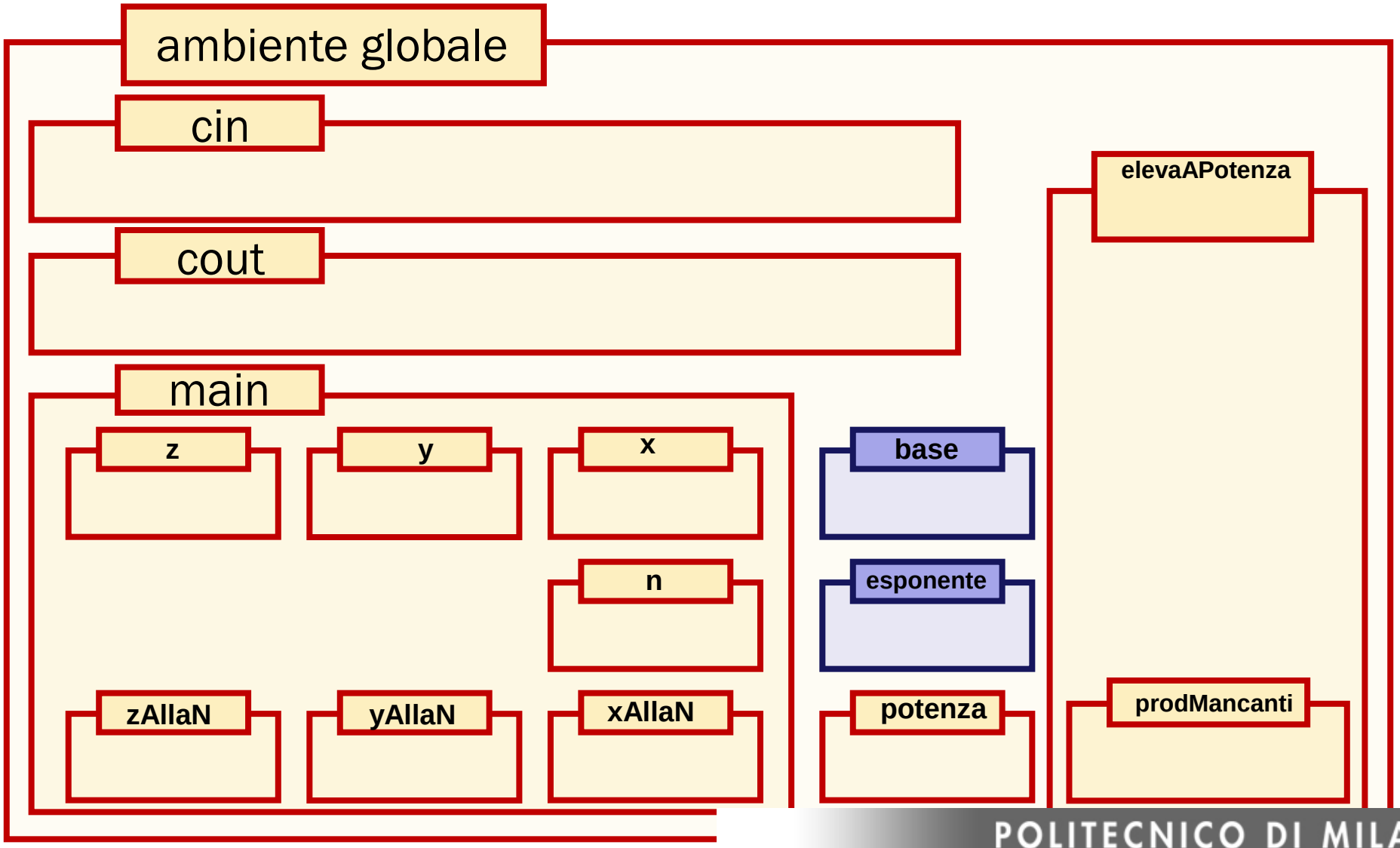
INTERFACCIA

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat.
 * Si fa uso di una funzione senza parametri.
 */
#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed elevaAPotenza
  int base, esponente, potenza;

void elevaAPotenza()
{ //versione con esponente positivo
    int prodMancanti;                //variabile locale ad elevaAPotenza
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
       potenza *= base;
}

void main()
{ int x, y, z, n                     //valori su cui operare
     xAllaN, yAllaN, aALLaN          //contengono x,y,z, elevati ad n

  //presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
         << "Se x, y, z sono interi positivi e n intero > 2" << endl
         << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
         << endl;

  //leggi i dati e verifica che rispondano alle specifiche
    cout << "Inserisci x,y,z n, separati da almeno uno spazio: "
         << endl;
    cin >> x >> y >> z >> n;
    if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;

  //calcola x elevato a n, con risultato in xAllaN
    base = x; esponente = n;
    elevaAPotenza();
    xAllaN = potenza;
```

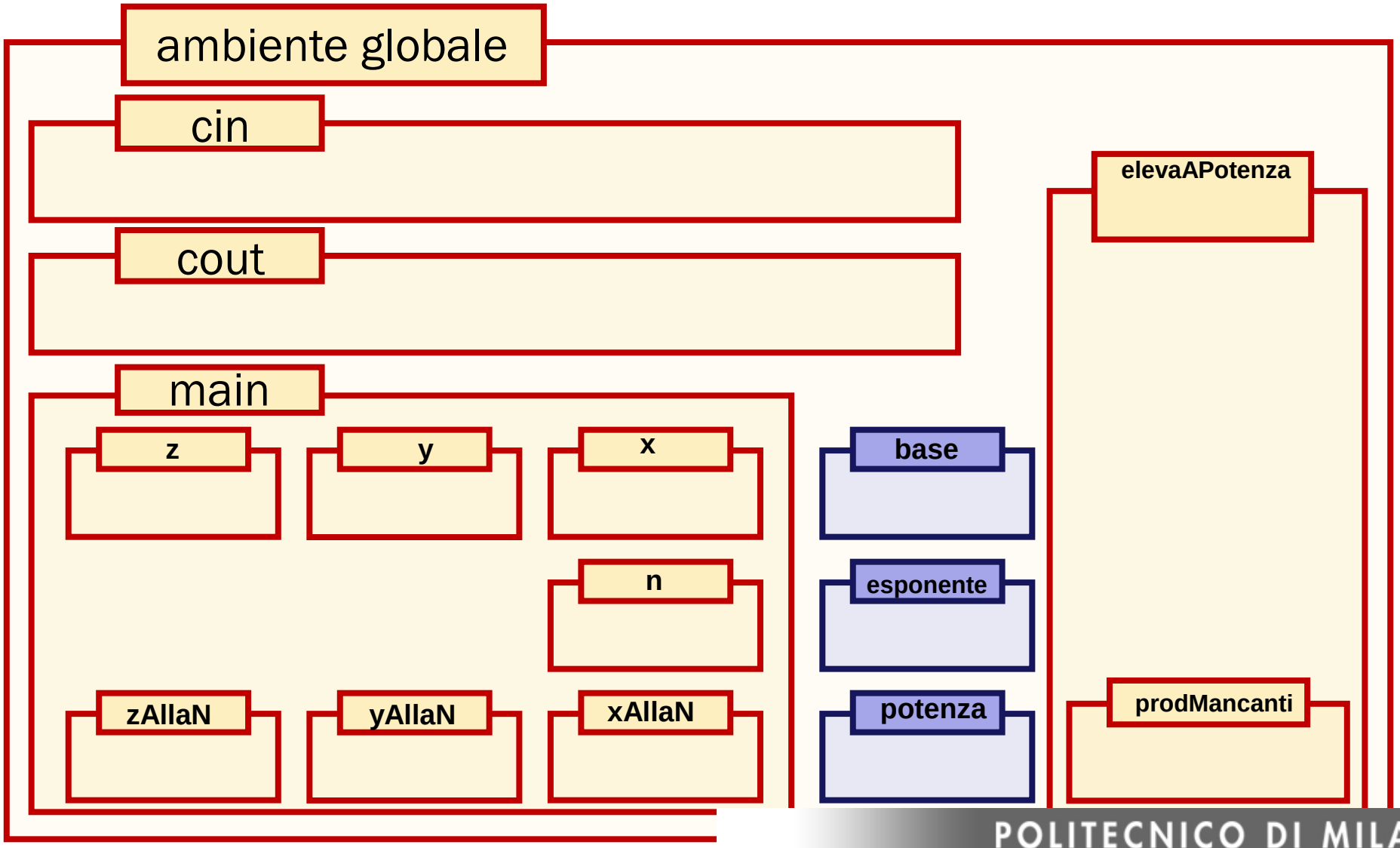
```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat.
 * Si fa uso di una funzione senza parametri.
 */
#include <iostream.h>
```



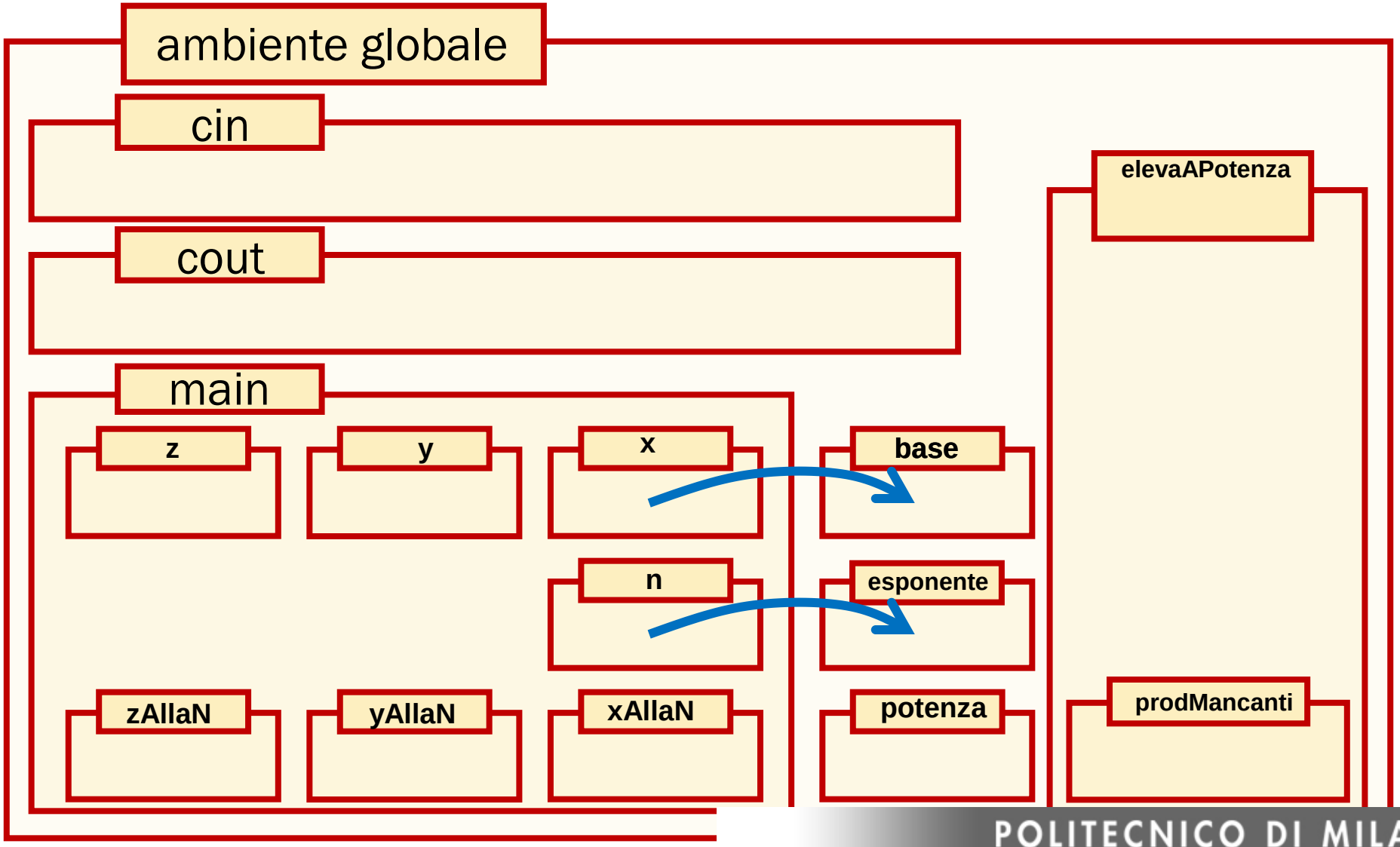
```
//variabili globali usate per la comunicazione fra main ed elevaAPotenza
  int base, esponente, potenza;

void elevaAPotenza()
```



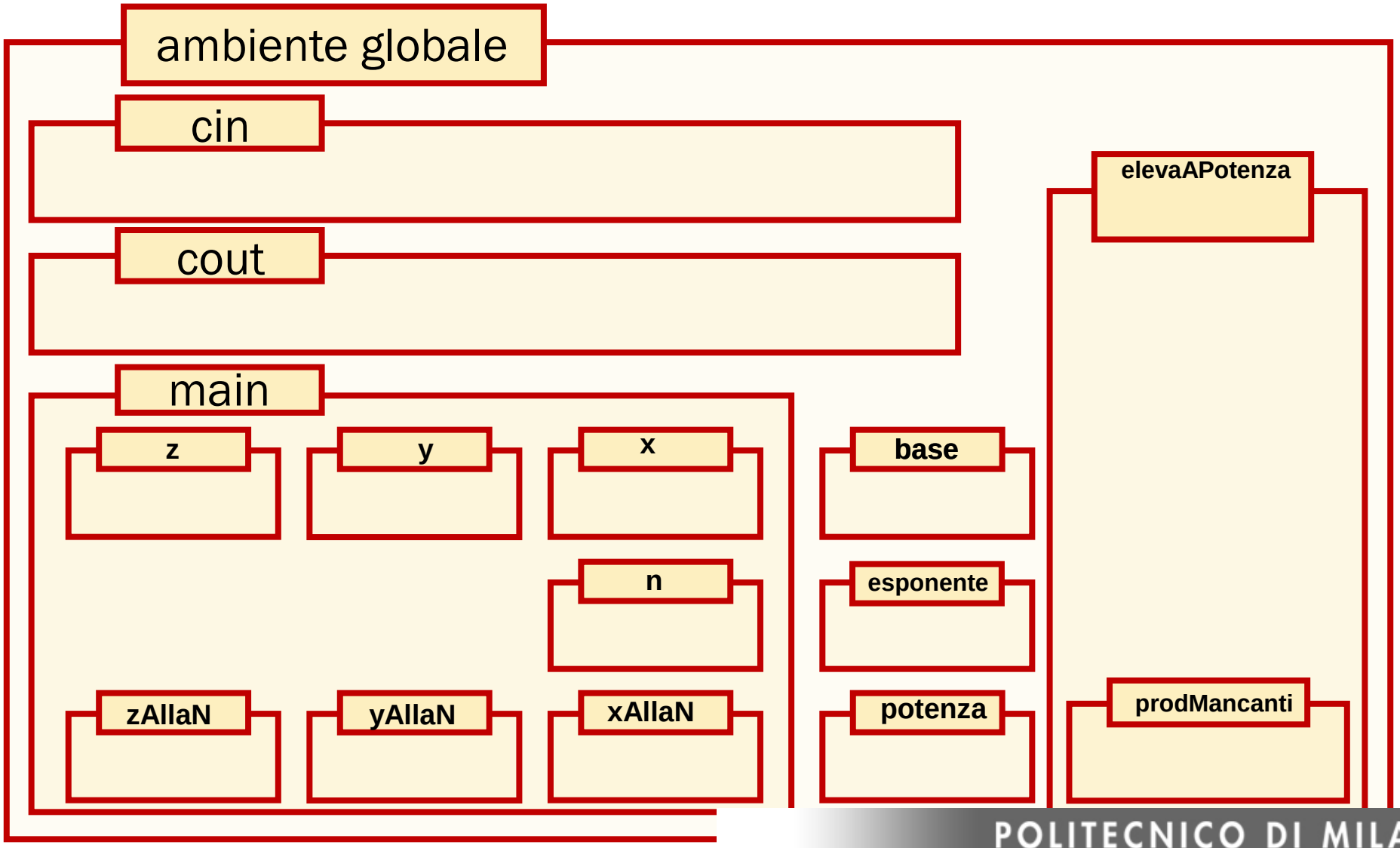
```
//variabili globali usate per la comunicazione fra main ed elevaAPotenza
  int base, esponente, potenza;

void elevaAPotenza()
```



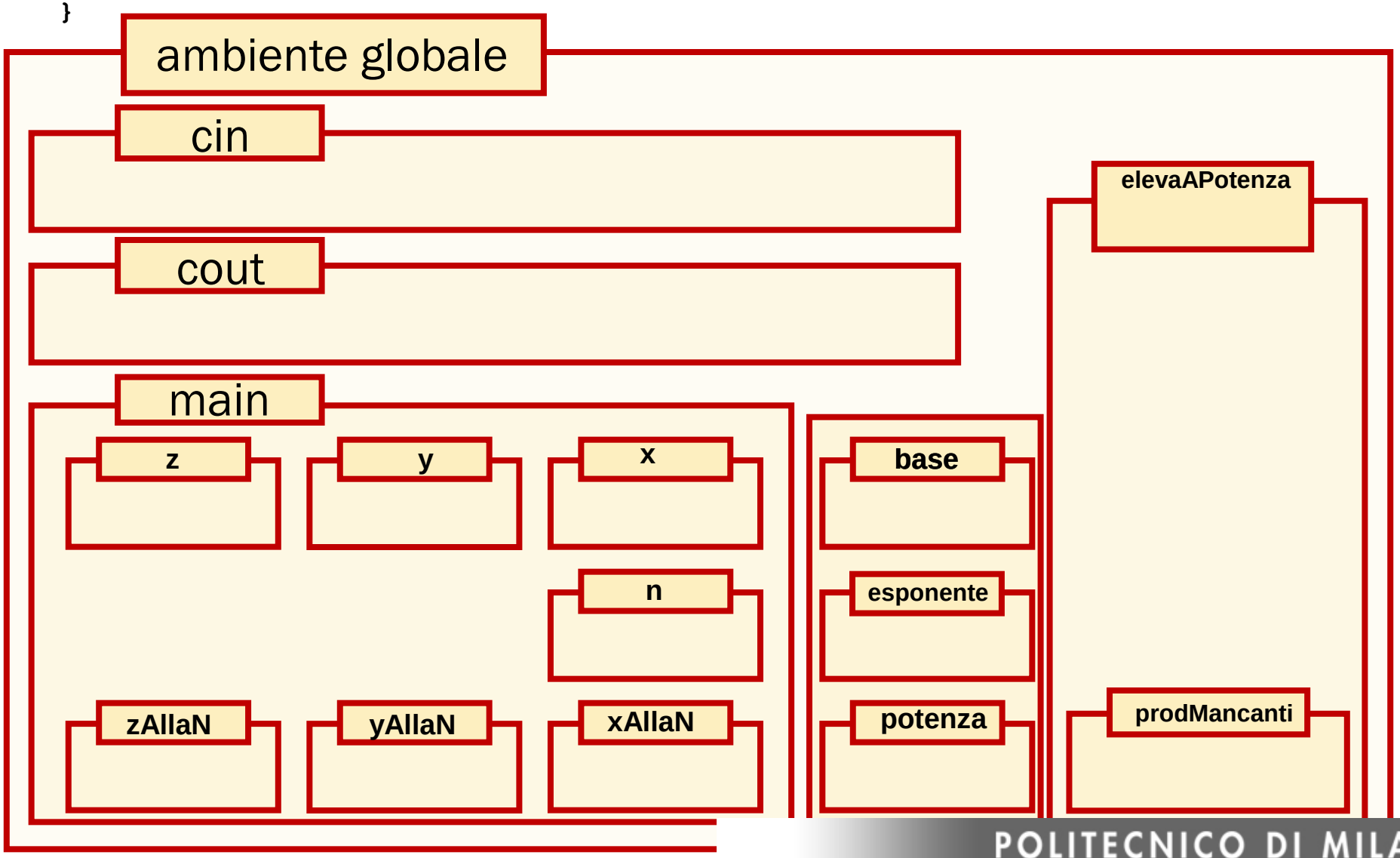
```
//calcola x elevato a n, con risultato in x AllaN
  base = x; esponente = n;
  elevaAPotenza();
  xAllaN = potenza;
```



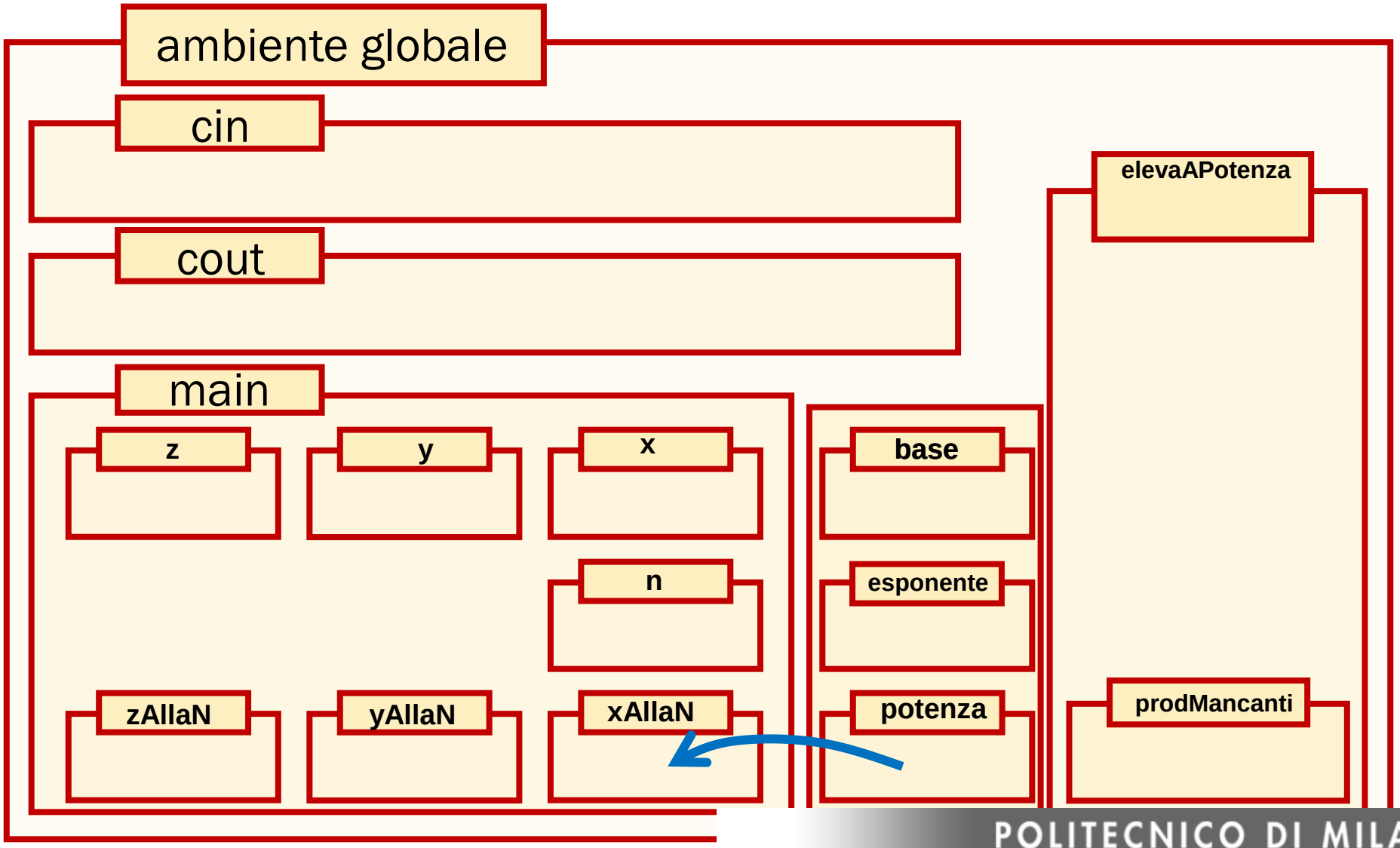
```
void elevaAPotenza()
{ //versione con esponente positivo
    int prodMancanti;              //variabile locale ad elevaAPotenza
    potenza = 1;
```



```
int prodMancanti;          //variabile locale ad elevaAPotenza
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
  }
```



```
base = x; esponente = n;
elevaAPotenza();
 xAllaN = potenza;
```



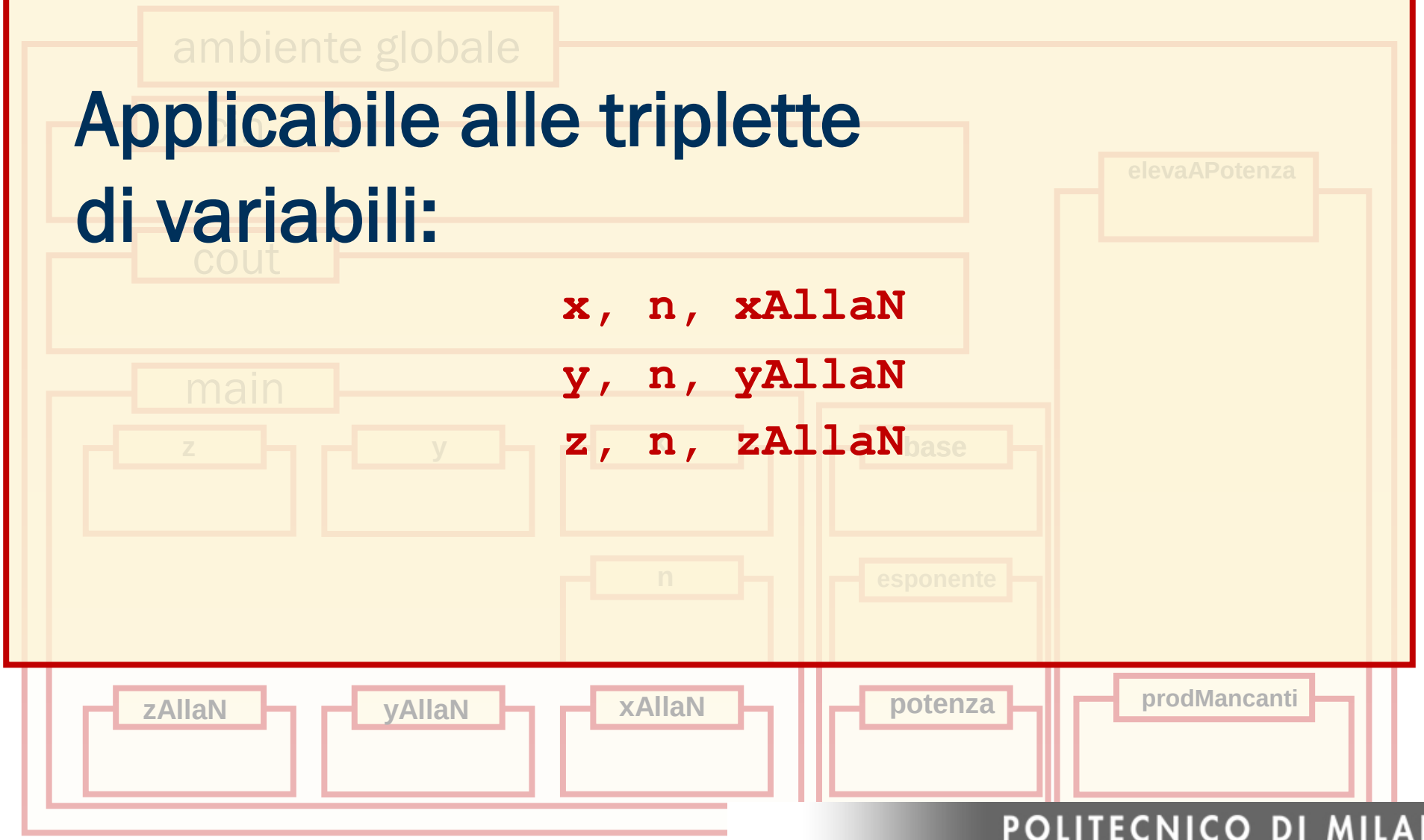
```
base = x; esponente = n;
elevaAPotenza();
 xAllaN = potenza;
```

Applicabile alle triplette di variabili:

x, n, xAllaN

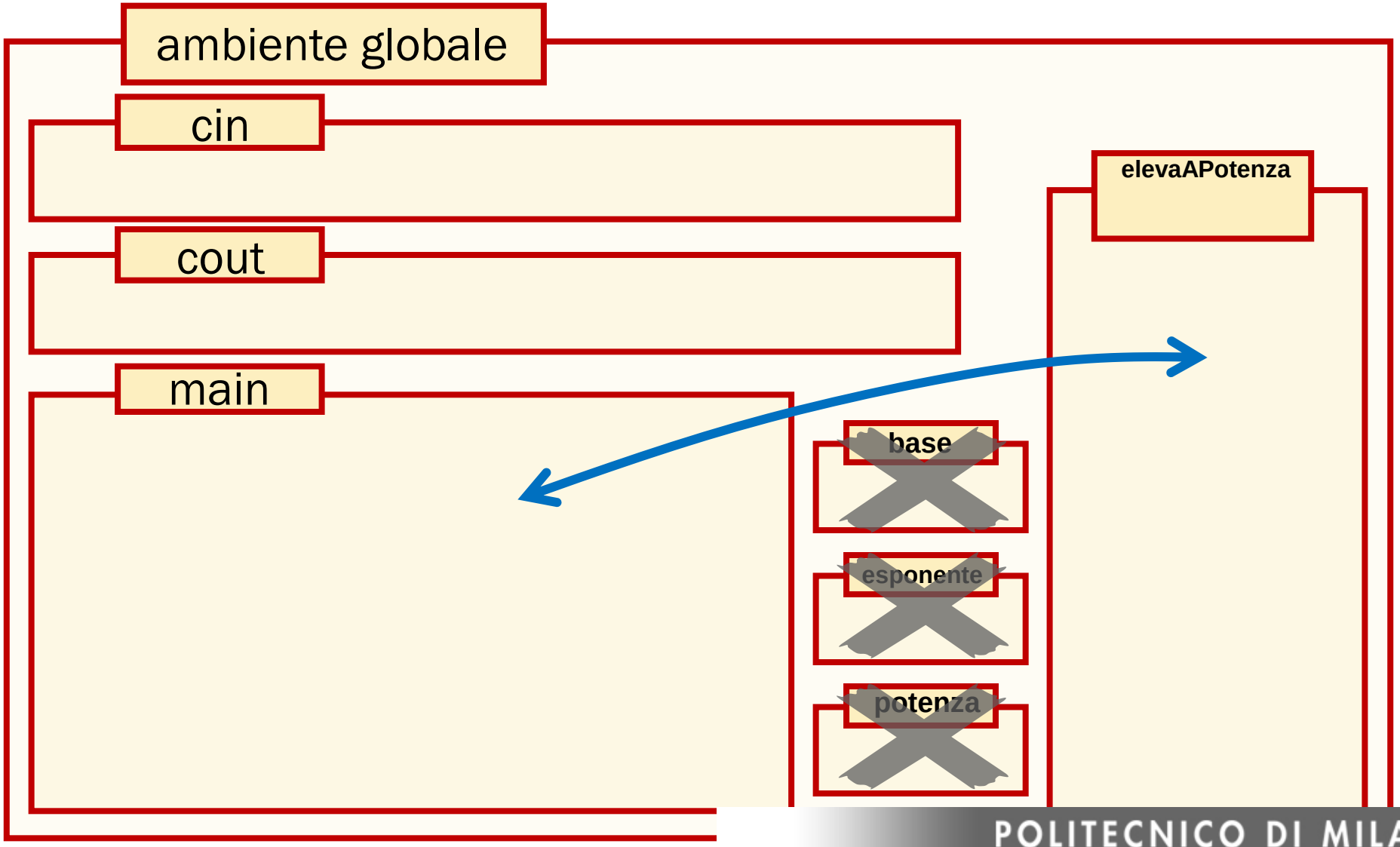
y, n, yAllaN

z, n, zAllaN



```
/* Tentativo senza speranza di dimostrare la falsit
 * teorema di Fermat.
 * Si fa uso di na funzione senza parametri.
 */
```

NUOVA STRATEGIA



```
base = x; esponente = n;
elevaAPotenza();
 xAllaN = potenza;
```

PARAMETRI

```
void miaFunzione(valoreIn1,
                 valoreIn2,
                 valoreOut1)
```

PARAMETRI

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat.
 * Si fa uso di una funzione senza parametri formali per ottenere dati,
 * sia per fornire risultati.
 */
#include <iostream.h>

void elevaAPotenza(int base, int esponente, int & potenza)
{ //versione con esponente positivo
    int prodMancanti;          //variabile locale ad elevaAPotenza
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
}

void main()
{ int x, y, z, n                          //dati su cui operare
      xAllaN, yAllaN, aALLaN  //contengono x,y,z, elevati ad n

  "presenta le funzionalità del programma"

  "leggi i dati e verifica che rispondano alle specifiche"

  //calcola x elevato a n, con risultato in xAllaN
    elevaAPotenza(x, n, xAllaN);

  //calcola y elevato a n, con risultato in yAllaN
    elevaAPotenza(y, n, yAllaN);

  //calcola z elevato a n, con risultato in zAllaN
    elevaAPotenza(z, n, zAllaN)
```

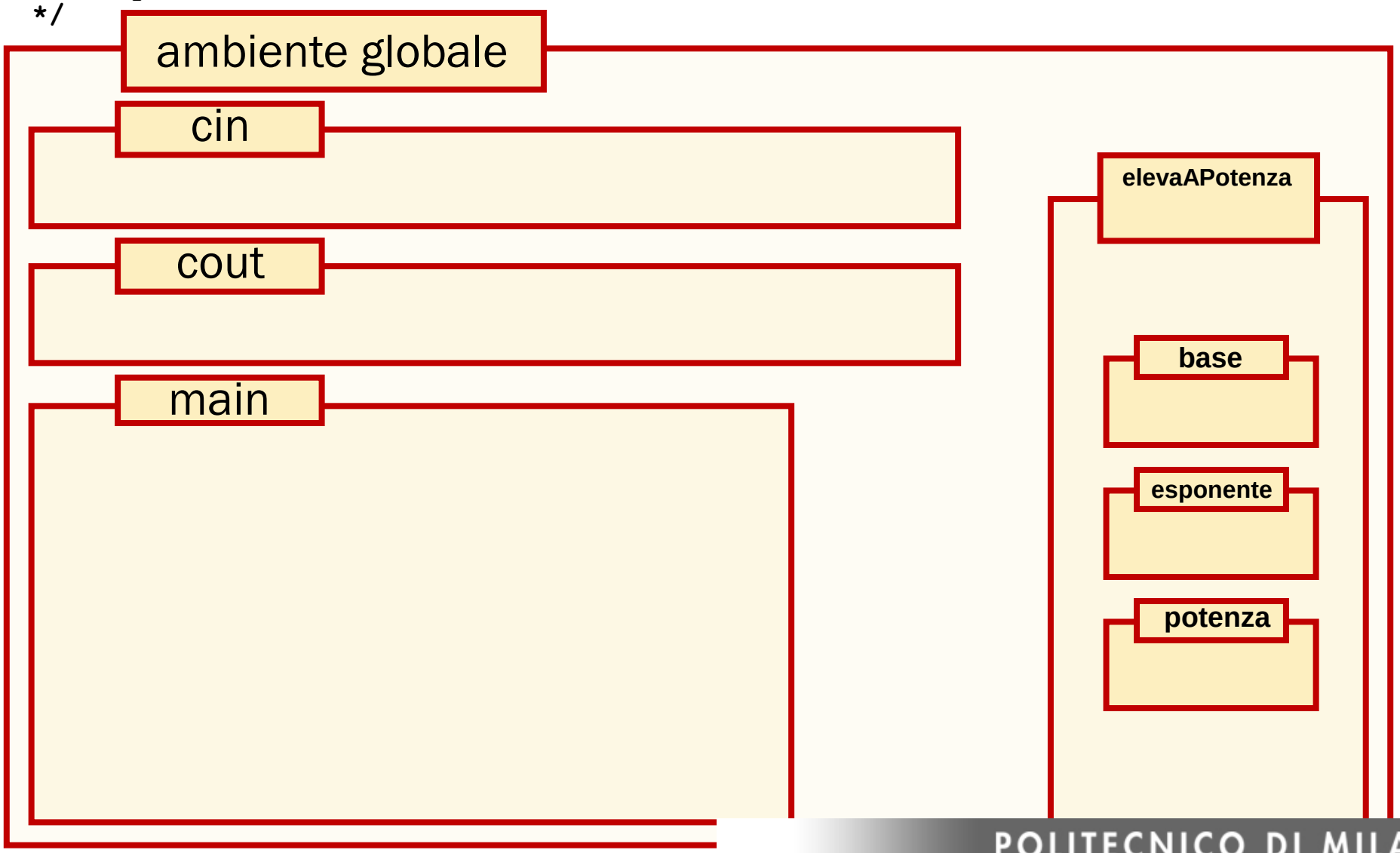
2 parametri in ingresso

PARAMETRI

```
/* Tentativo senza speranza di dimostrare la falsità dell'ulti
 * teorema di Fermat.
 * Si fa uso di una funzione senza parametri                 ati,
 * sia per fornire risultati.
 */
#include <iostream.h>

void elevaAPotenza(int base, int esponente, int & potenza)
{ //versione con esponente positivo
    int prodMancanti;          //variabile locale ad elevaAPotenza
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
}

void main()
{ int x, y, z, n                          //dati su cui operare
      xAllaN, yAllaN, aALLaN  //contengono x,y,z, elevati ad n

  "presenta le funzionalità del programma"

  "leggi i dati e verifica che rispondano alle specifiche"

  //calcola x elevato a n, con risultato in xAllaN
    elevaAPotenza(x, n, xAllaN);

  //calcola y elevato a n, con risultato in yAllaN
    elevaAPotenza(y, n, yAllaN);

  //calcola z elevato a n, con risultato in zAllaN
    elevaAPotenza(z, n, zAllaN)
```

1 parametro in uscita

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat .
 * Si fa uso di una funzione con parametri, sia per ricevere dati,
 * sia per fornire risultati.
 */
```



```
void elevaAPotenza(int base, int esponente, int & potenza)
{ //versione con esponente positivo
    int prodMancanti;              //variabile locale ad elevaAPotenza
    potenza = 1;
```

Parametri in ingresso

base

esponente

Parametri in uscita

& potenza

```
void elevaAPotenza(int base, int esponente, int & potenza)
{ //versione con esponente positivo
    int prodMancanti;          //variabile locale ad elevaAPotenza
```

Parametri in ingresso

```
int base

int esponente
```

Parametri in uscita

```
int & potenza
```

I nomi dei parametri vengono usati compatibilmente con il proprio tipo

```
void elevaAPotenza(int base, int esponente,
{ //versione con esponente positivo
    int prodMancanti;          //variabile locale ad elevaAPotenza
```

Chiamata

ambiente globale

cin

main

elevaAPotenza

base

esponente

potenza

```
void main()
   {…
    elevaAPotenza(x, n, xAllaN)
    …
   }
```

```
void elevaAPotenza(int base, in esponente, int & potenza)
```

Parametri formali

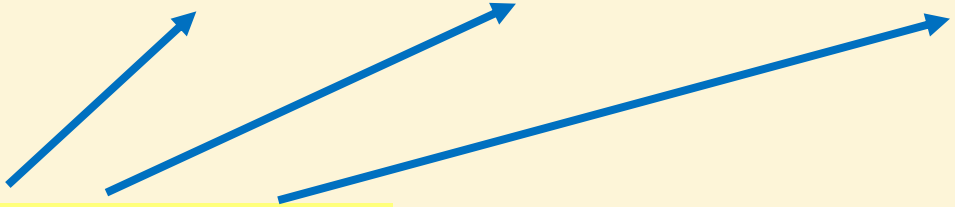
```
void main()
   {…
    elevaAPotenza(x, n, xAllaN)
    …
   }


void elevaAPotenza(int base, in esponente, int & potenza)
```

Parametri effettivi

```
void main()
   {…
    elevaAPotenza(x, n, xAllaN)
    …
   }
```

```
void elevaAPotenza(int base, in esponente, int & potenza)
```

```
void elevaAPotenza(int base, in esponente, int & potenza)
{ … }


void main()
{…
    elevaAPotenza(y, n, yAllaN)
 …
}
```



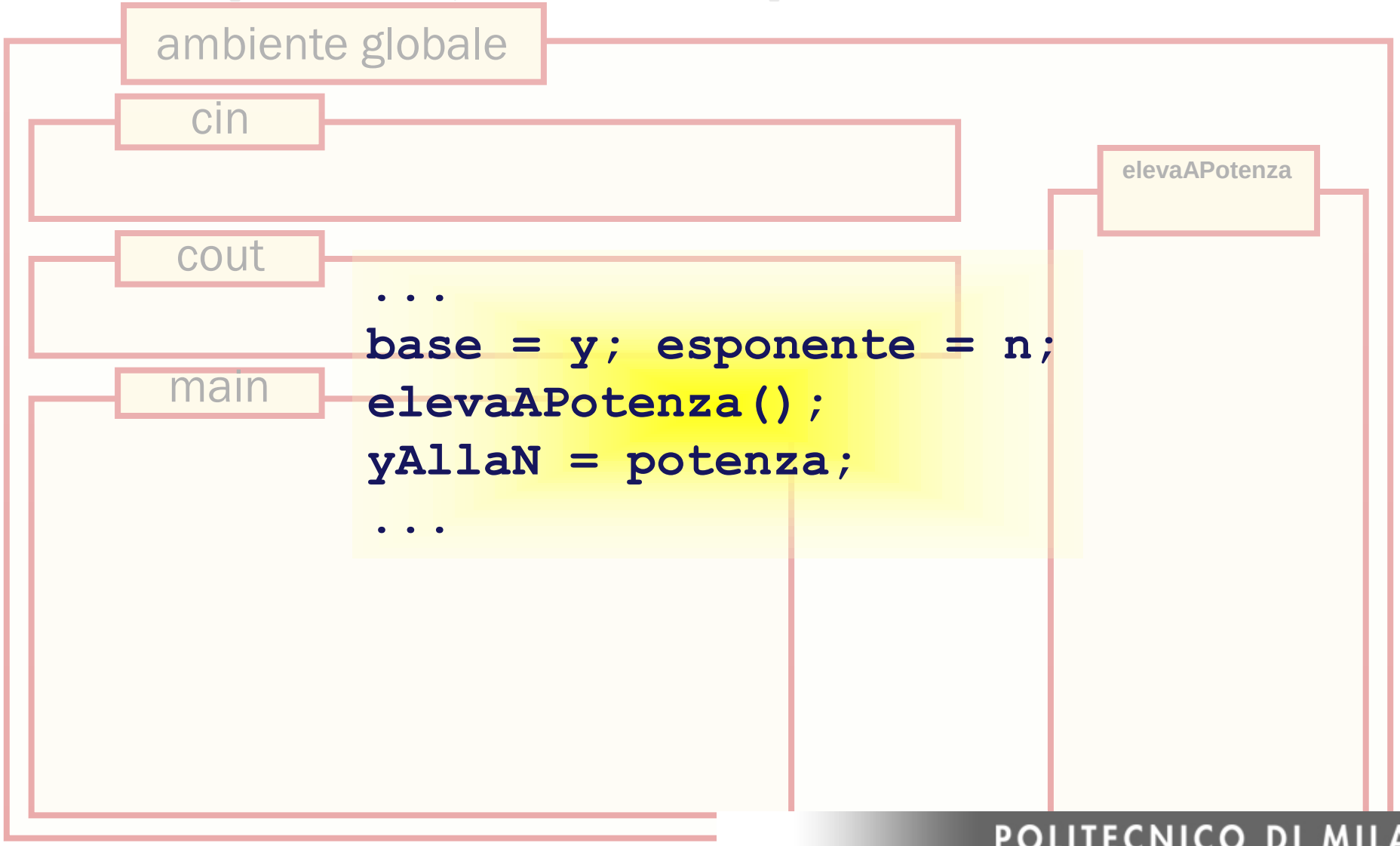
```
void elevaAPotenza()
{ … }

void main()
{…
  { base = y; esponente = n;
    elevaAPotenza()
    yAllaN = potenza;
 …
}
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione c                    vere dati,
 * sia per fornire risultati
*/
#include <iostream.h>

void elevaAPotenza(int base, int esponente, int & potenza)
{ //versione con esponente positivo
    int prodMancanti;              //variabile locale ad elevaAPotenza
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
}

void main()
{ int x, y, z, n                   //dati su cui operare
     xAllaN, yAllaN, aALLaN        //contengono x,y,z, elevati ad n

  "presenta le funzionalità del programma"

  "leggi i dati e verifica che rispondano alle specifiche"

  //calcola x elevato a n, con risultato in xAllaN
    elevaAPotenza(x, n, xAllaN);

  //calcola y                                  to in yAllaN
    elevaAPote

  //calcola z elevato a n, con risultato in zAllaN
    elevaAPotenza(z, n, zAllaN);

"Verifica se xAllaN + yAllaN è uguale a zAllaN e stampa il risultato"
}
```

Parametri formali

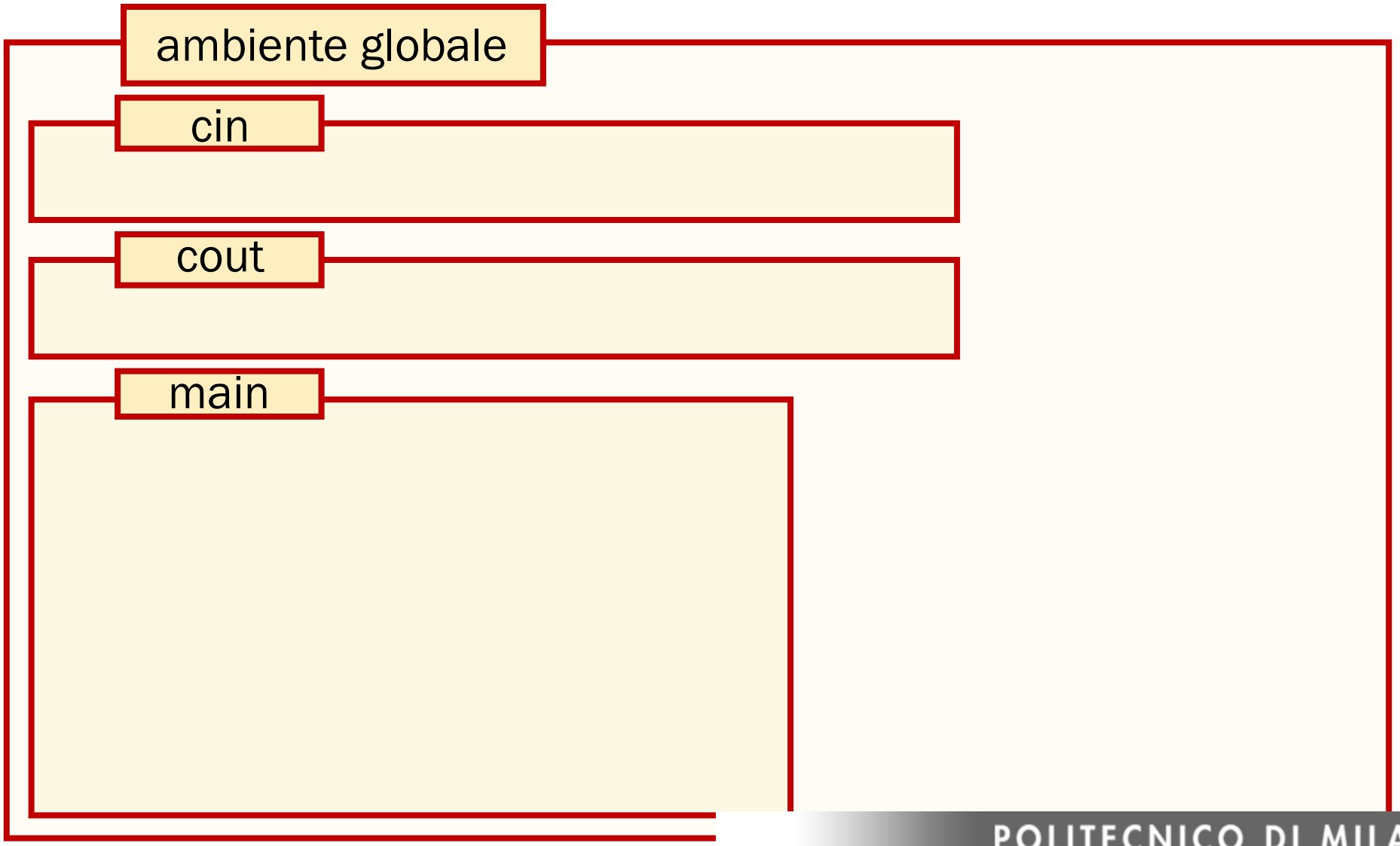
Parametri effettivi

```
//calcola x elevato a n, con risultato in xAllaN
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

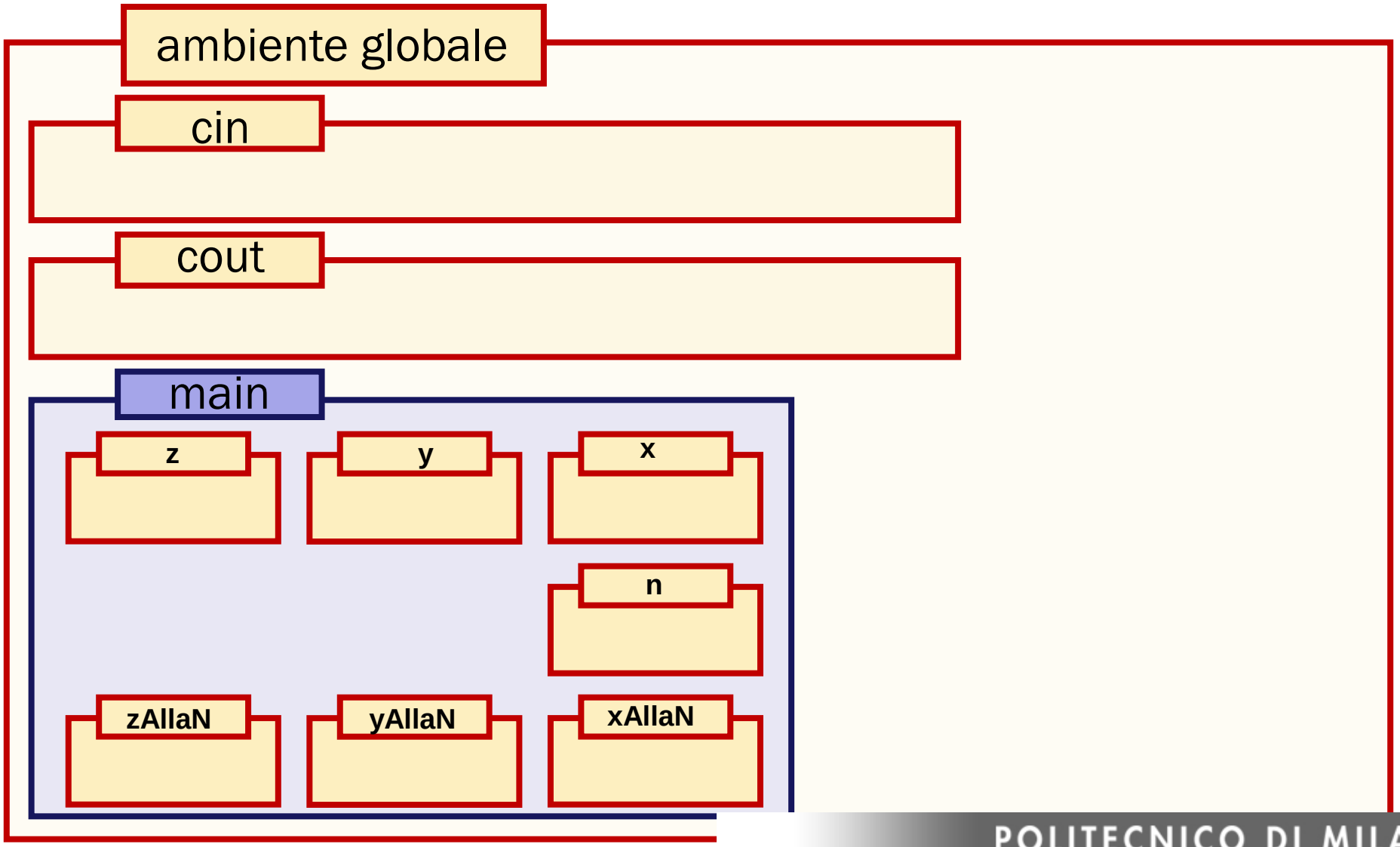


```
   * sia per fornire risultati
*/
#include <iostream.h>

void elevaAPotenza(int base, int esponente, int & potenza)
```

```
void main()
{ int x, y, z, n                    // dati su cui operare
      xAllaN, yAllaN, zALLaN        // contengono x,y,z, elevati ad n
```

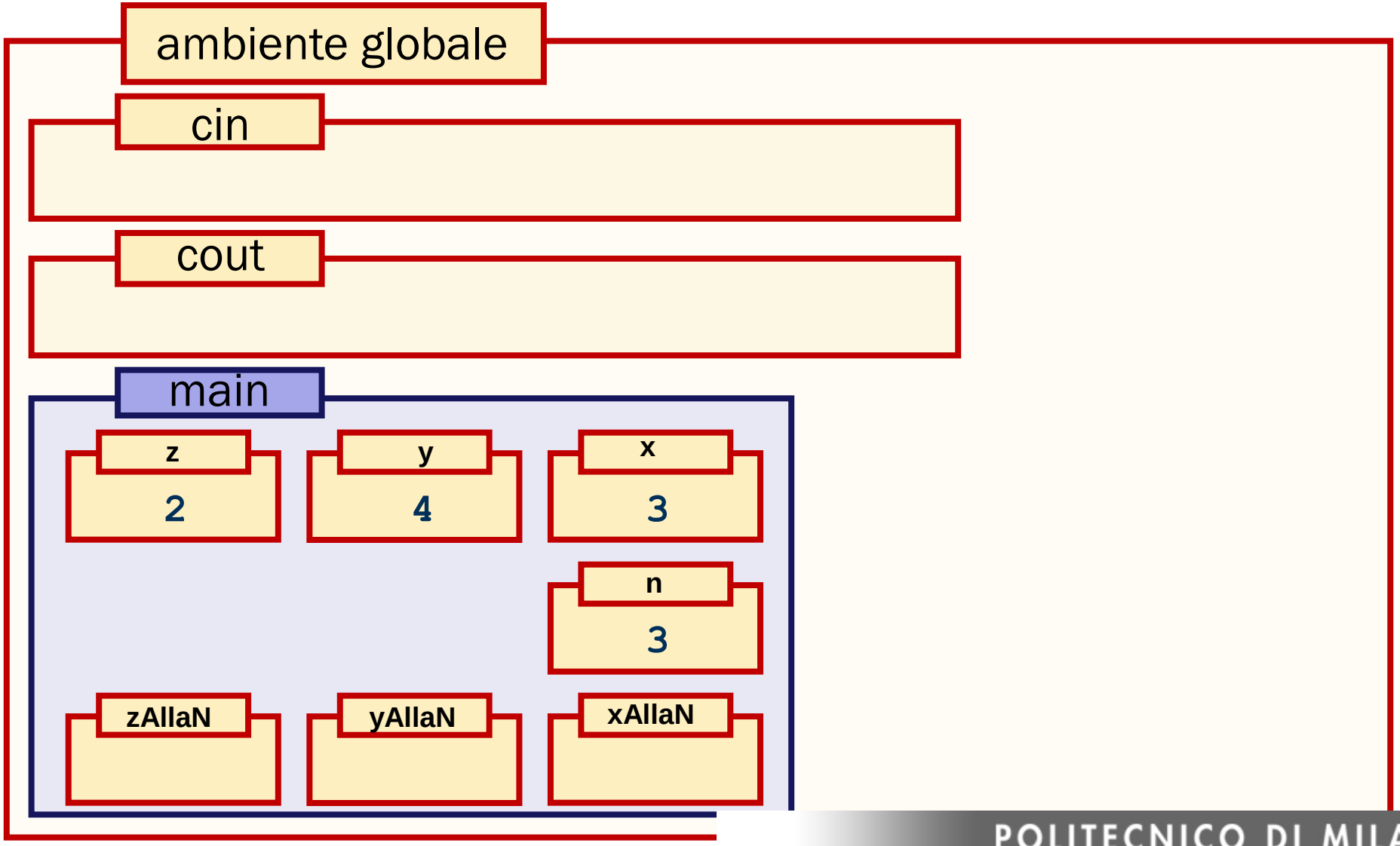


```
void main()
{ int x, y, z, n                    // dati su cui operare
      xAllaN, yAllaN, zALLaN        // contengono x,y,z, elevati ad n
```



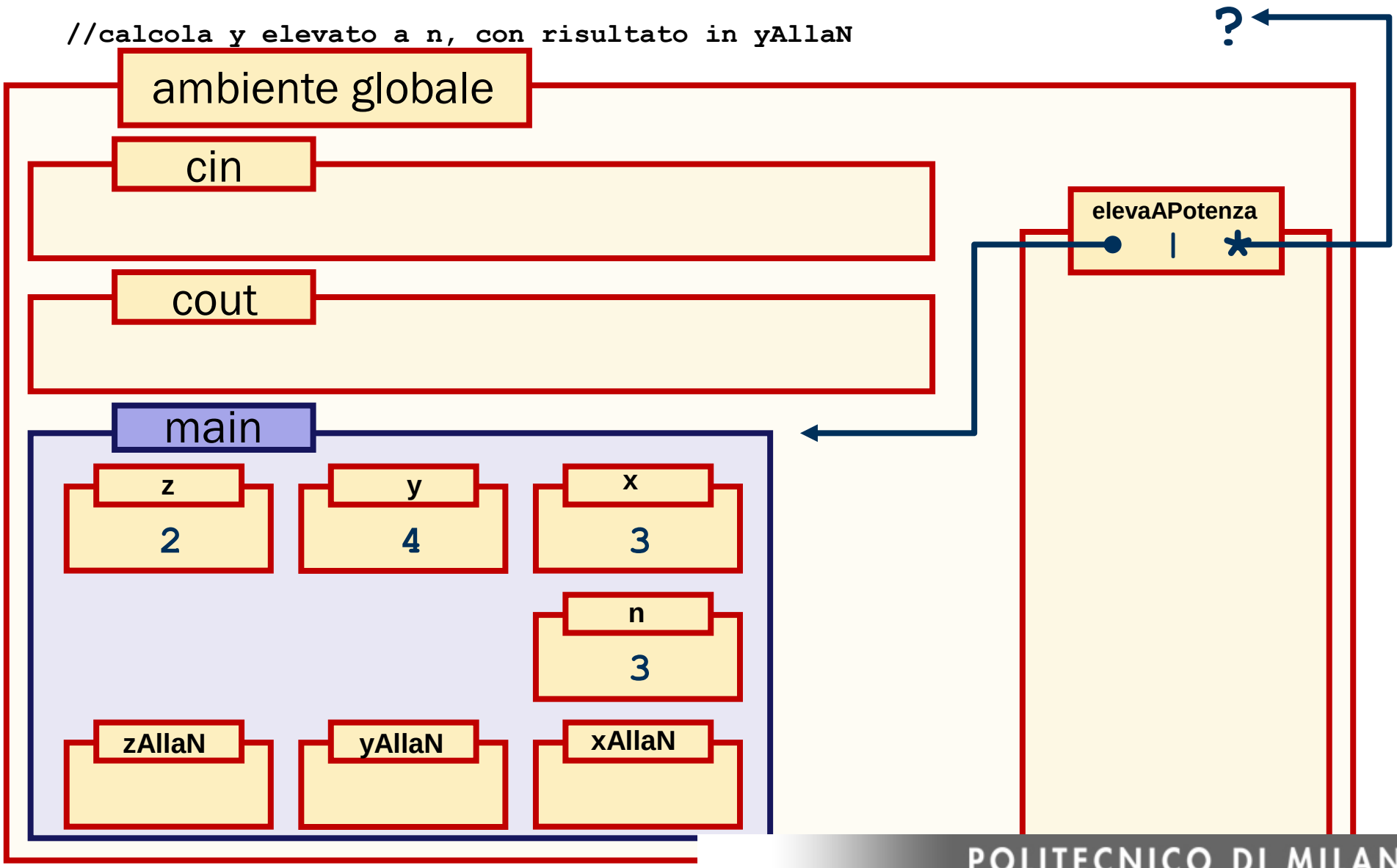
"presenta le funzionalità del programma"

"leggi i dati x,y,z,n e verifica che rispondano alle specifiche"

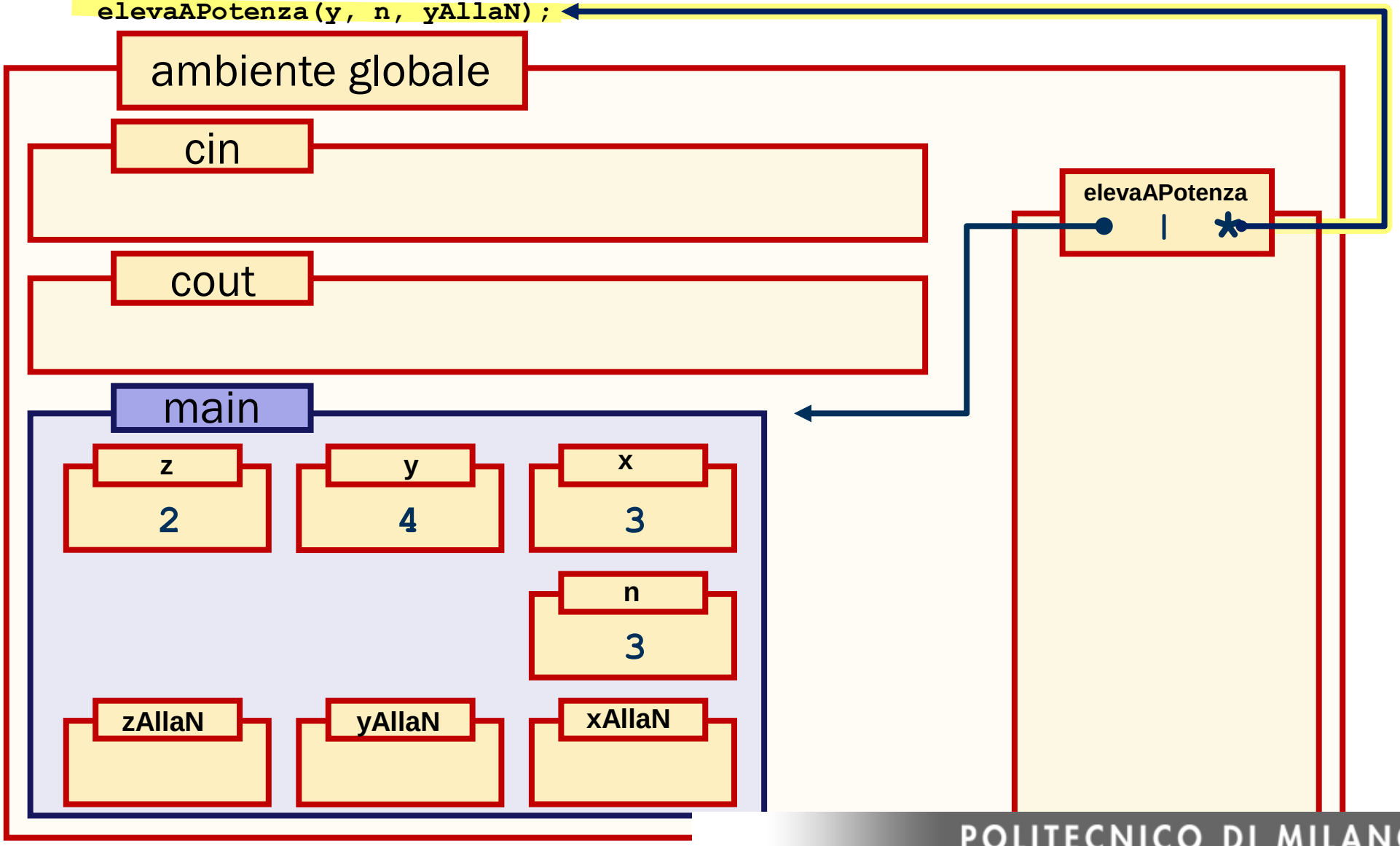


```
//calcola x elevato a n, con risultato in x
 elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri

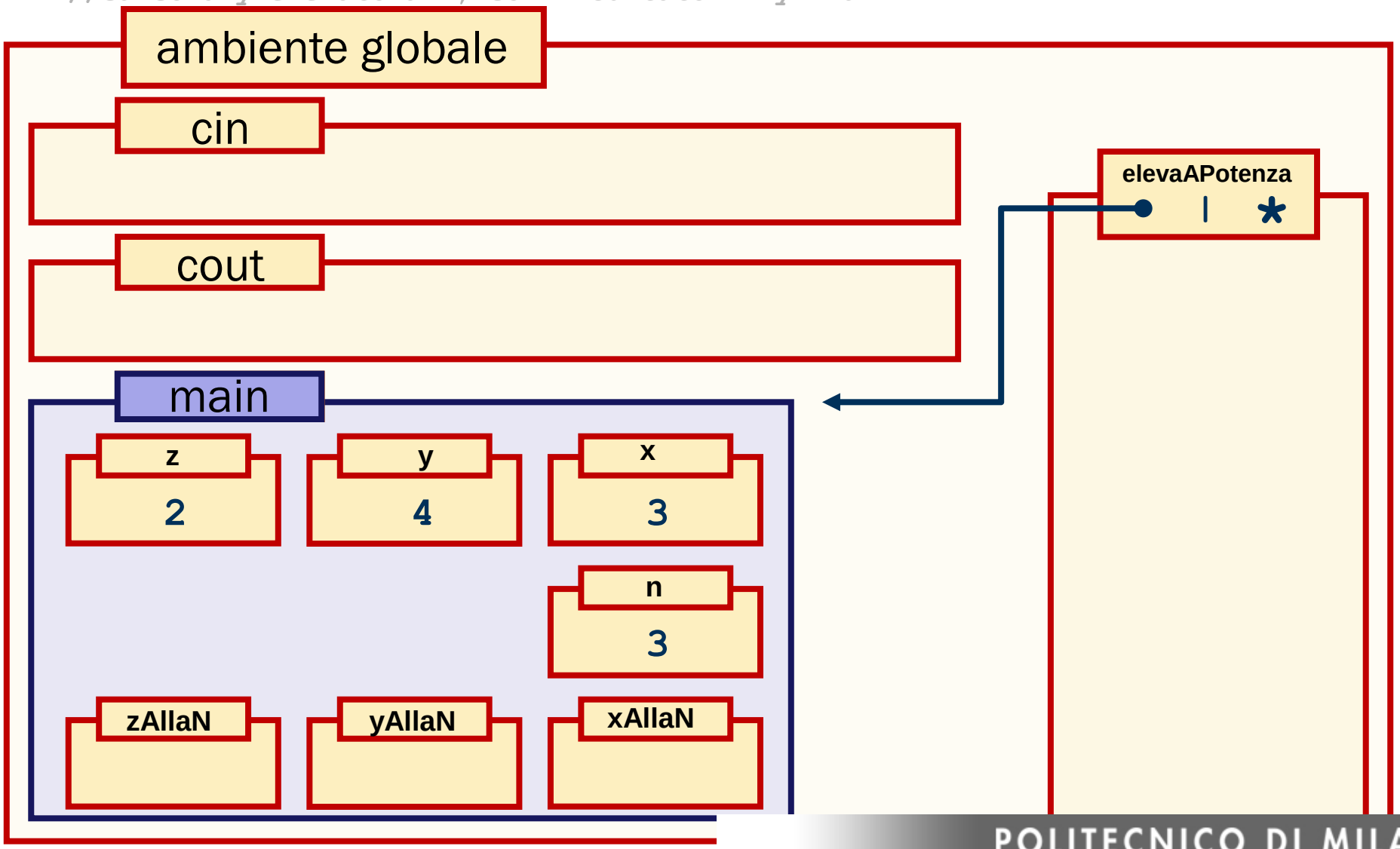


```
elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN);
```



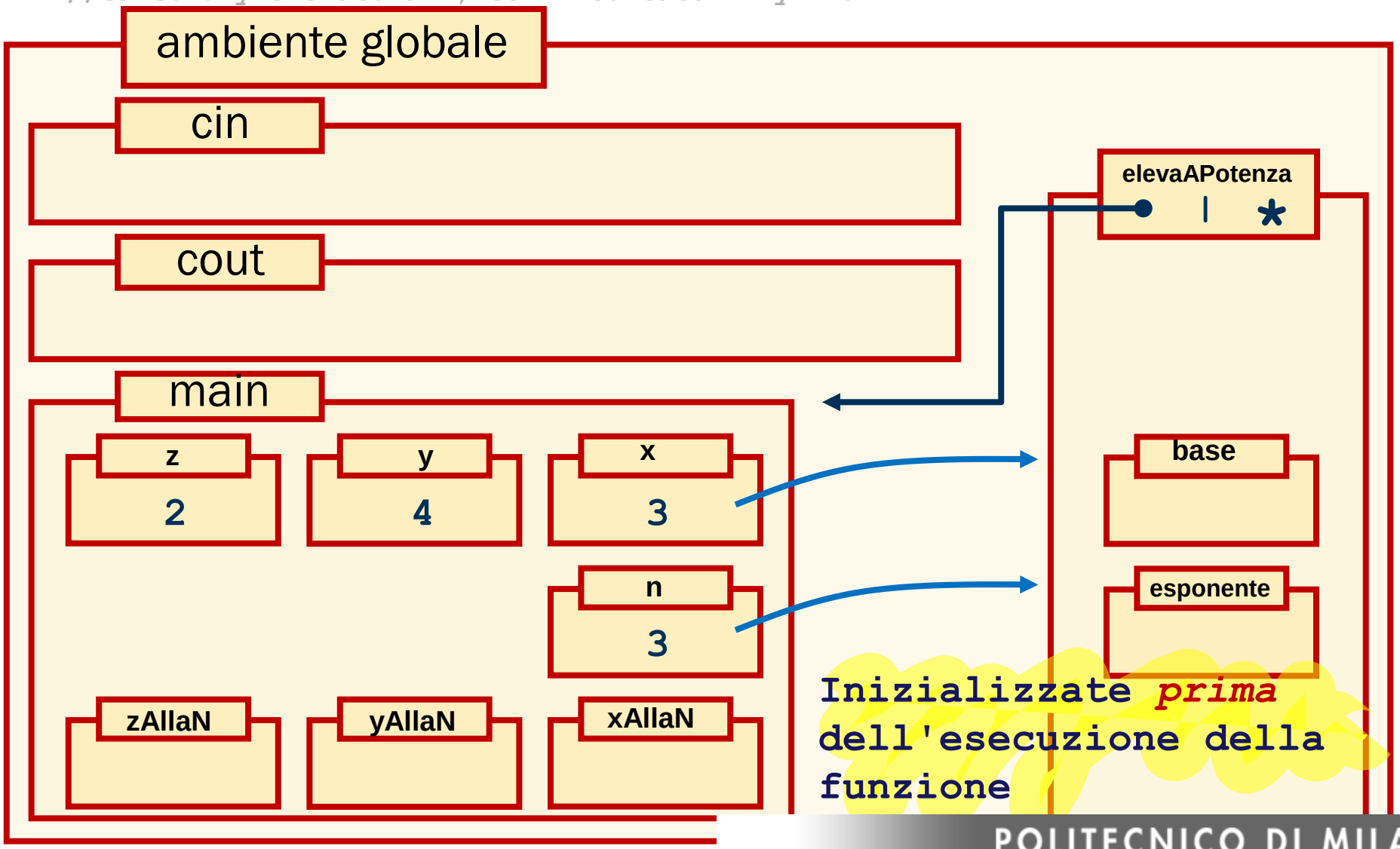
Passaggio di parametri

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```



```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

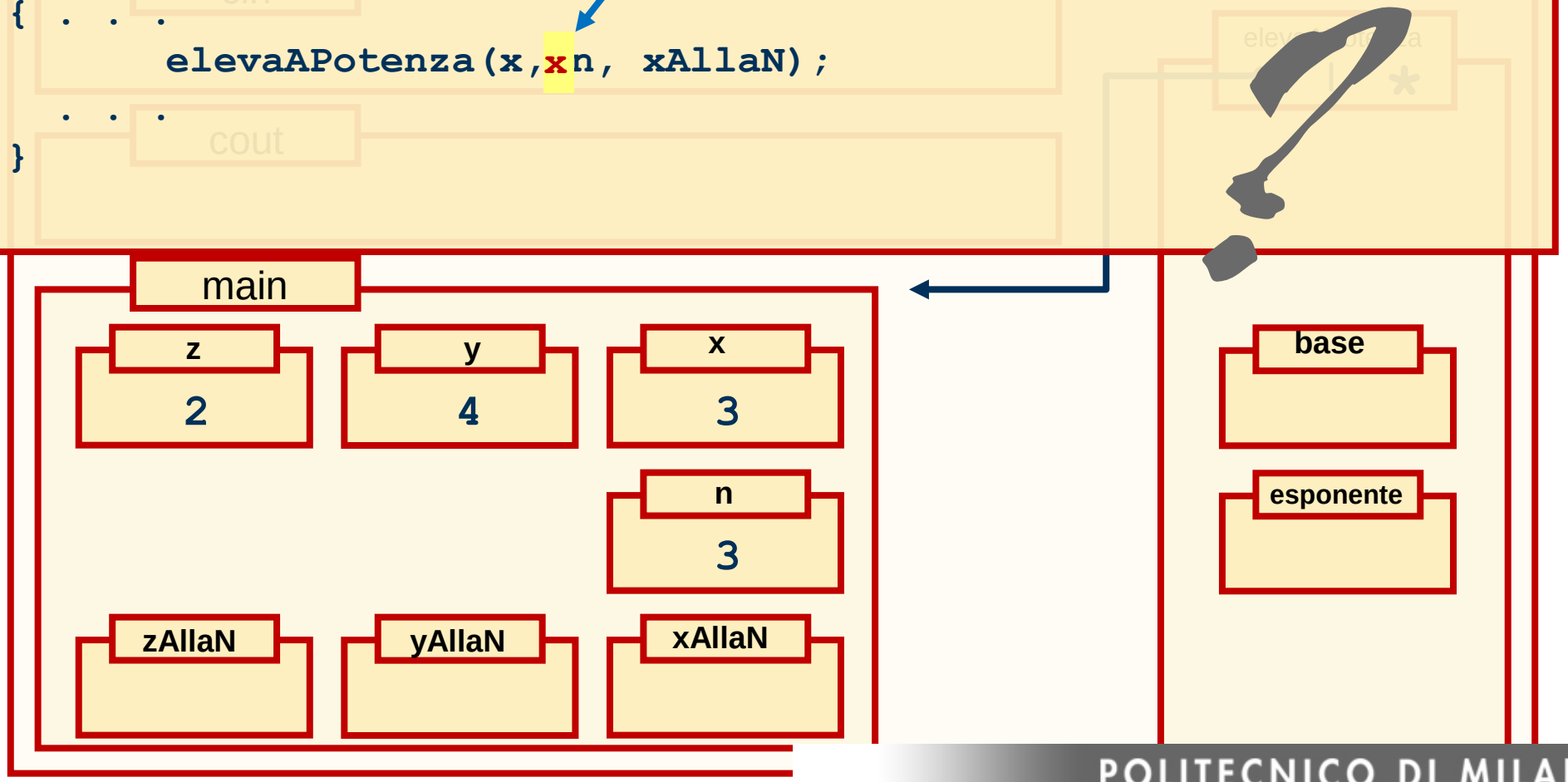
Parametri in ingresso



```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

Parametri in ingresso

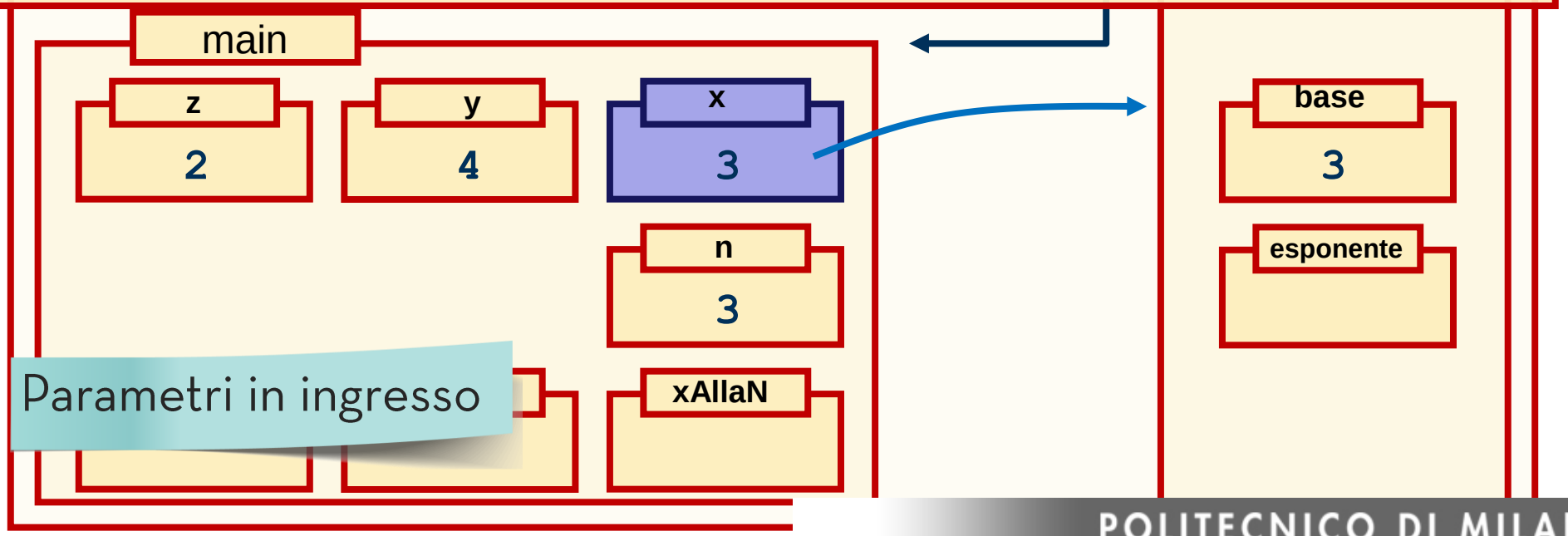
```
void elevaAPotenza (int base, int esponente, int & potenza)
{. . .}
void main()
{ . . .
      elevaAPotenza(x,xn, xAllaN);
  . . .
}
```



```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

Parametri in ingresso

```
void elevaAPotenza (int base, int esponente, int & potenza)
{. . .}
void main()
{ . . .
      elevaAPotenza(x,xn, xAllaN);
  . . .
}
```

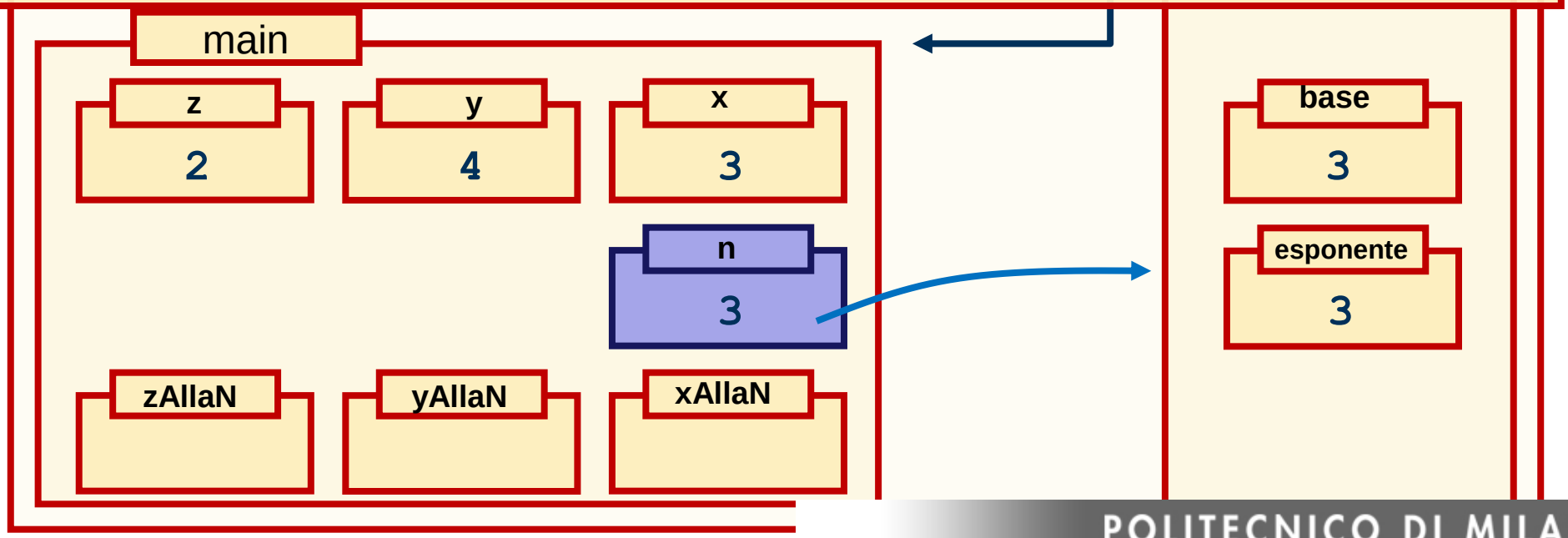


Parametri in ingresso

Parametri in ingresso

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

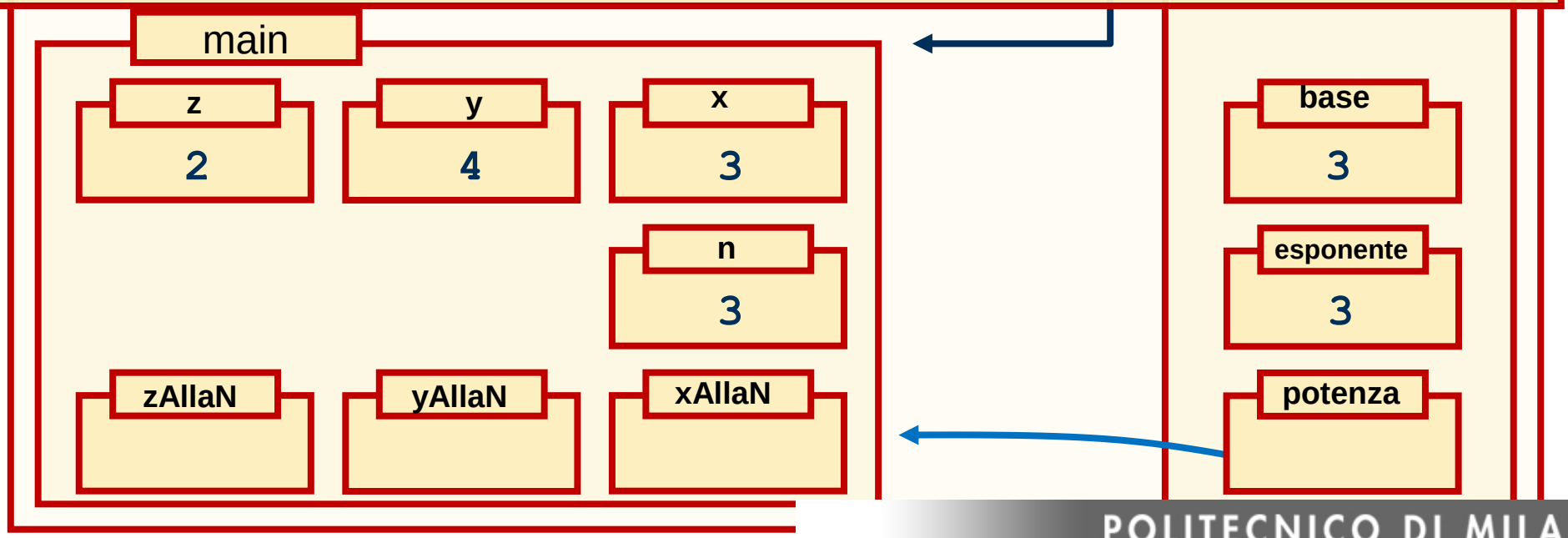
```
void elevaAPotenza (int base, int esponente, int & potenza)
{. . .}
void main()
{ . . .
      elevaAPotenza(x, n, xAllaN);
  . . .
}
```



Parametri in uscita

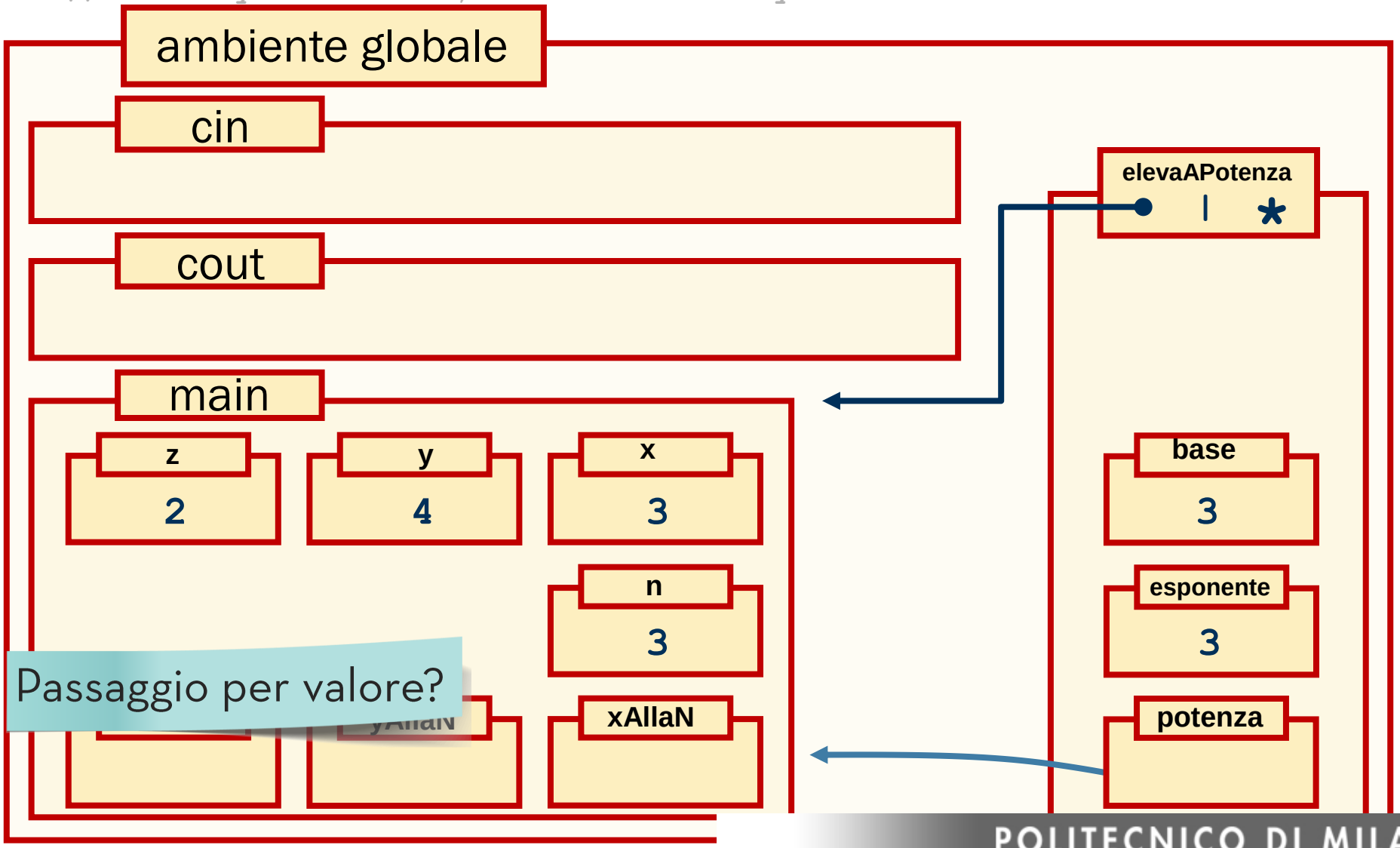
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

```
void elevaAPotenza (int base, int esponente, int & potenza)
{. . .}
void main()
{ . . .
      elevaAPotenza(x, n, xAllaN);
  . . .
}
```



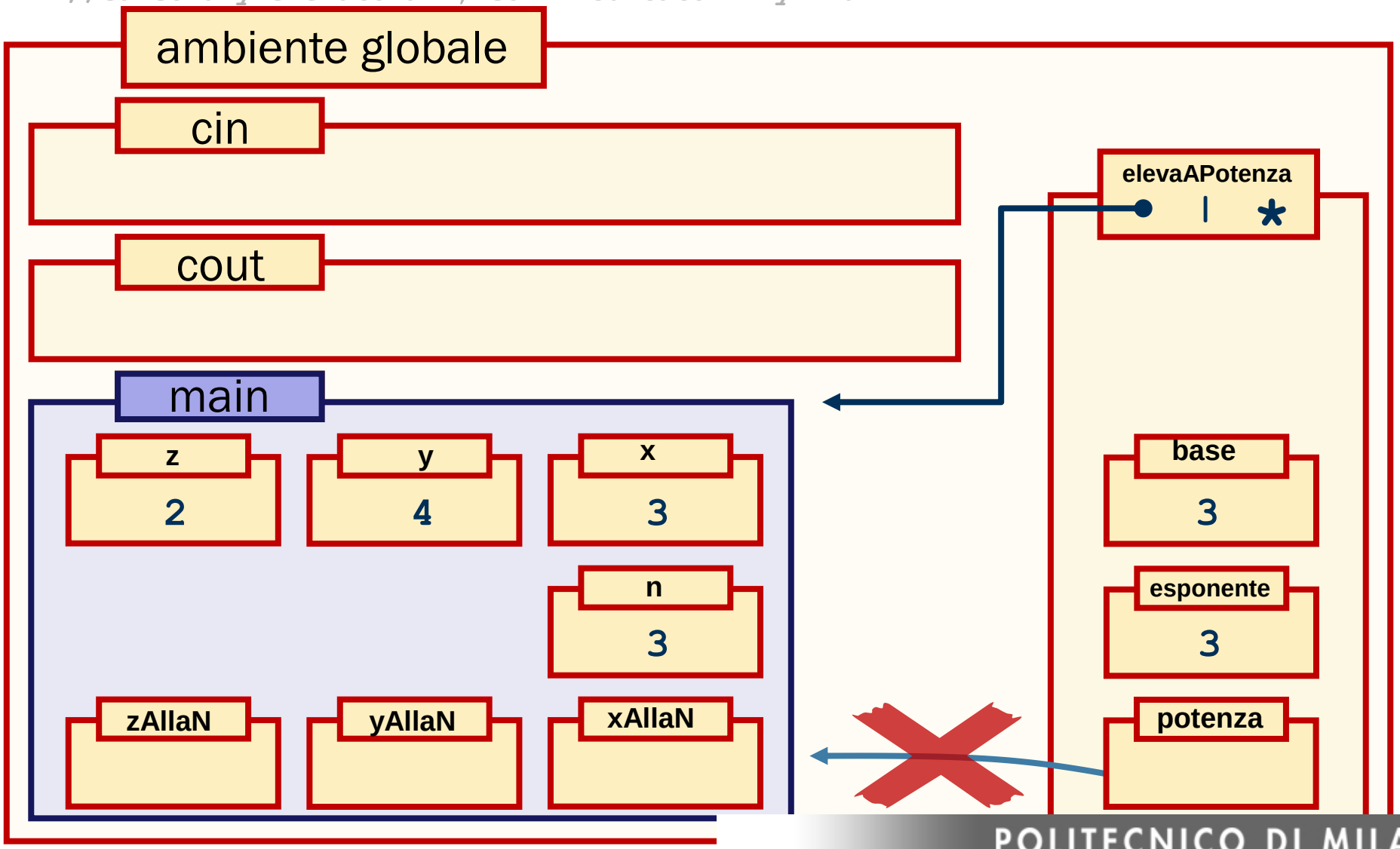
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



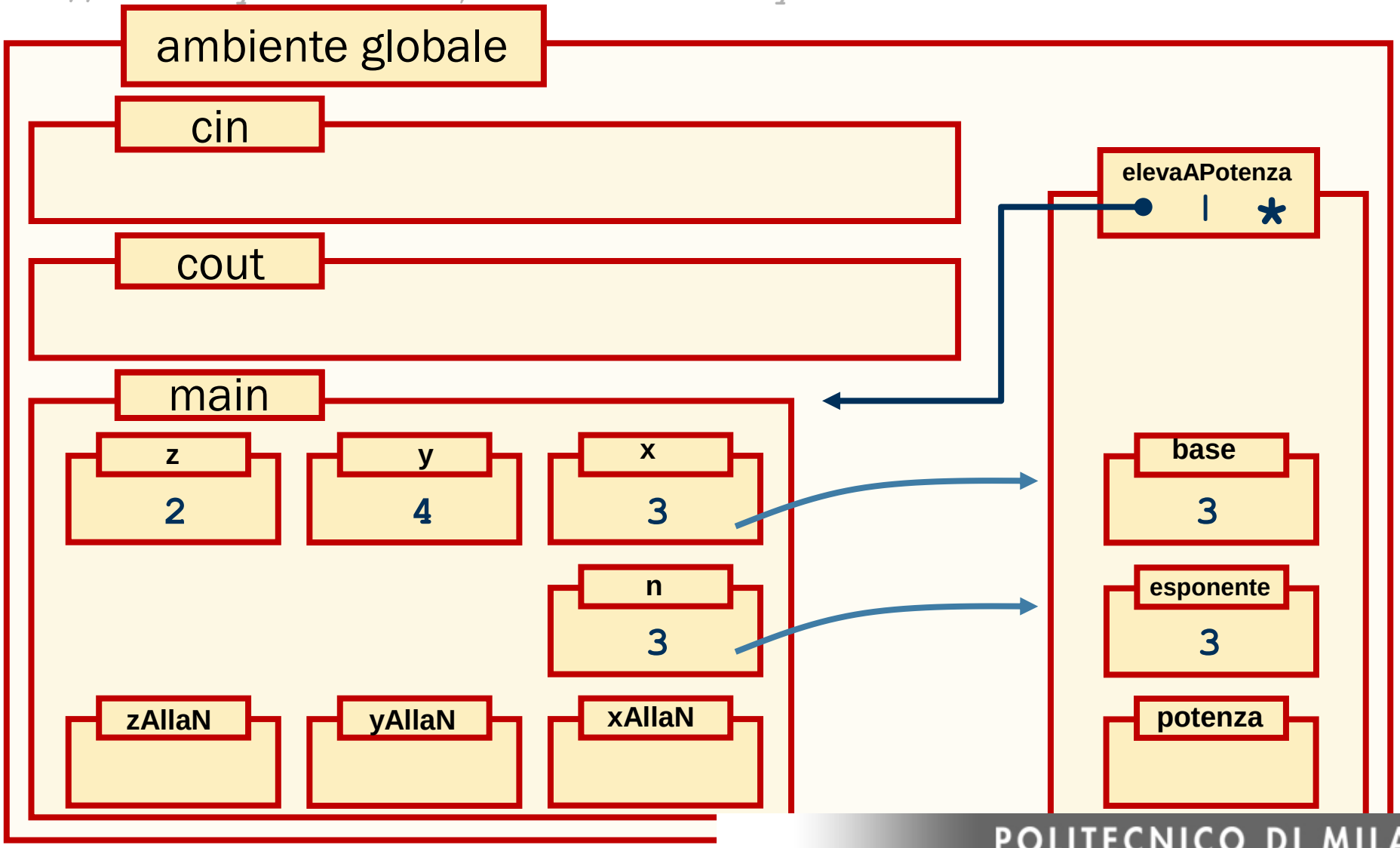
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



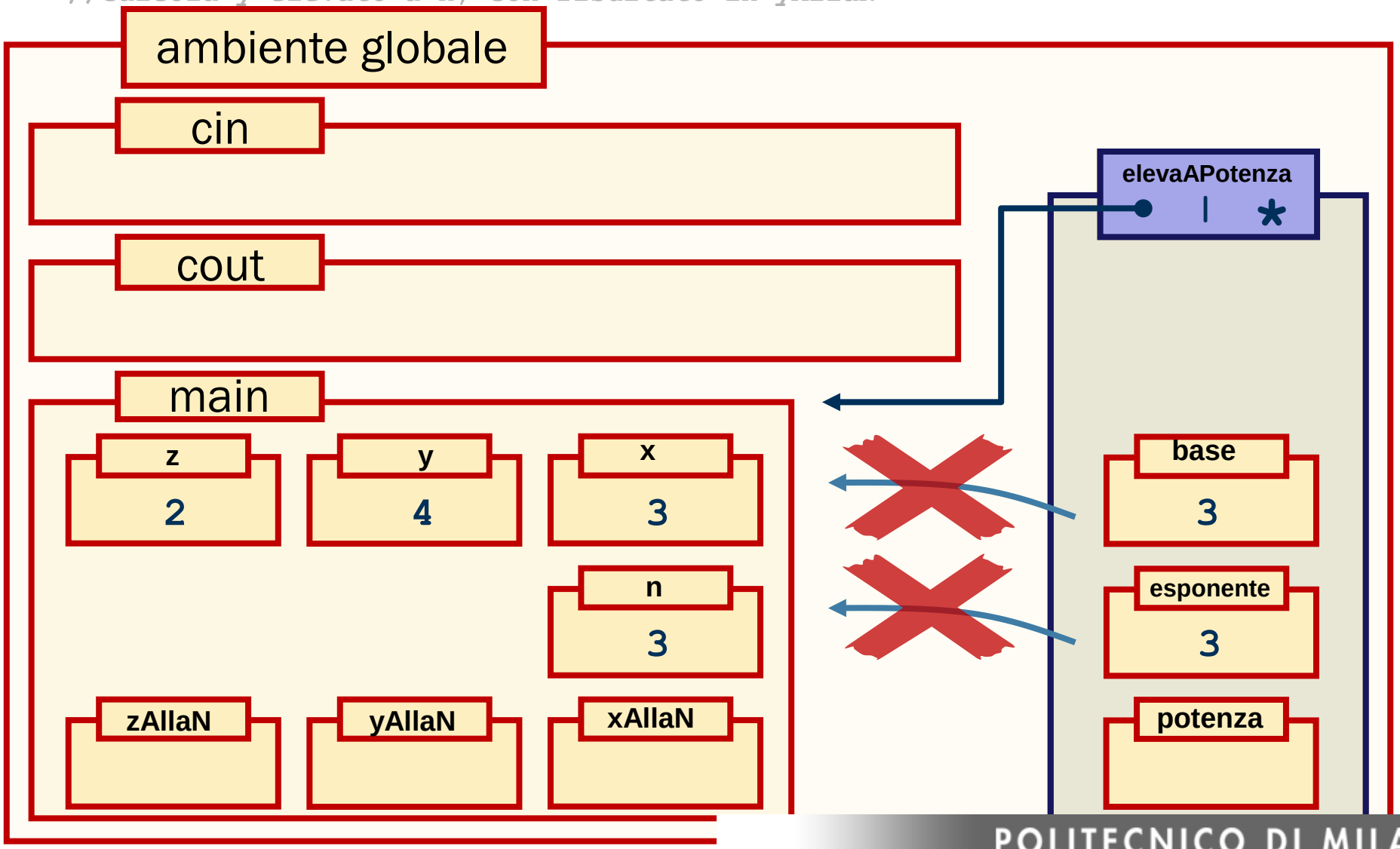
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



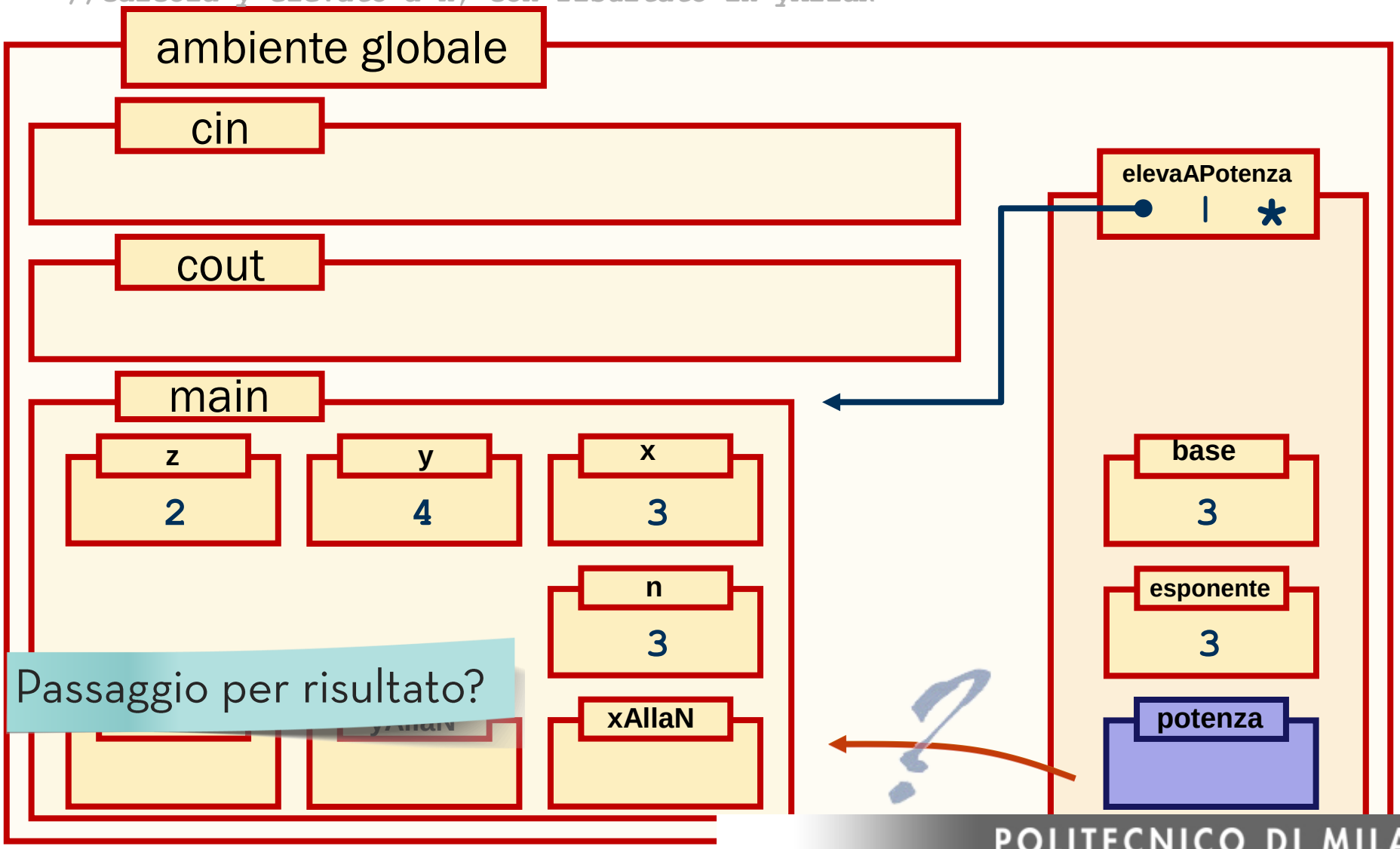
Parametri in uscita

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```



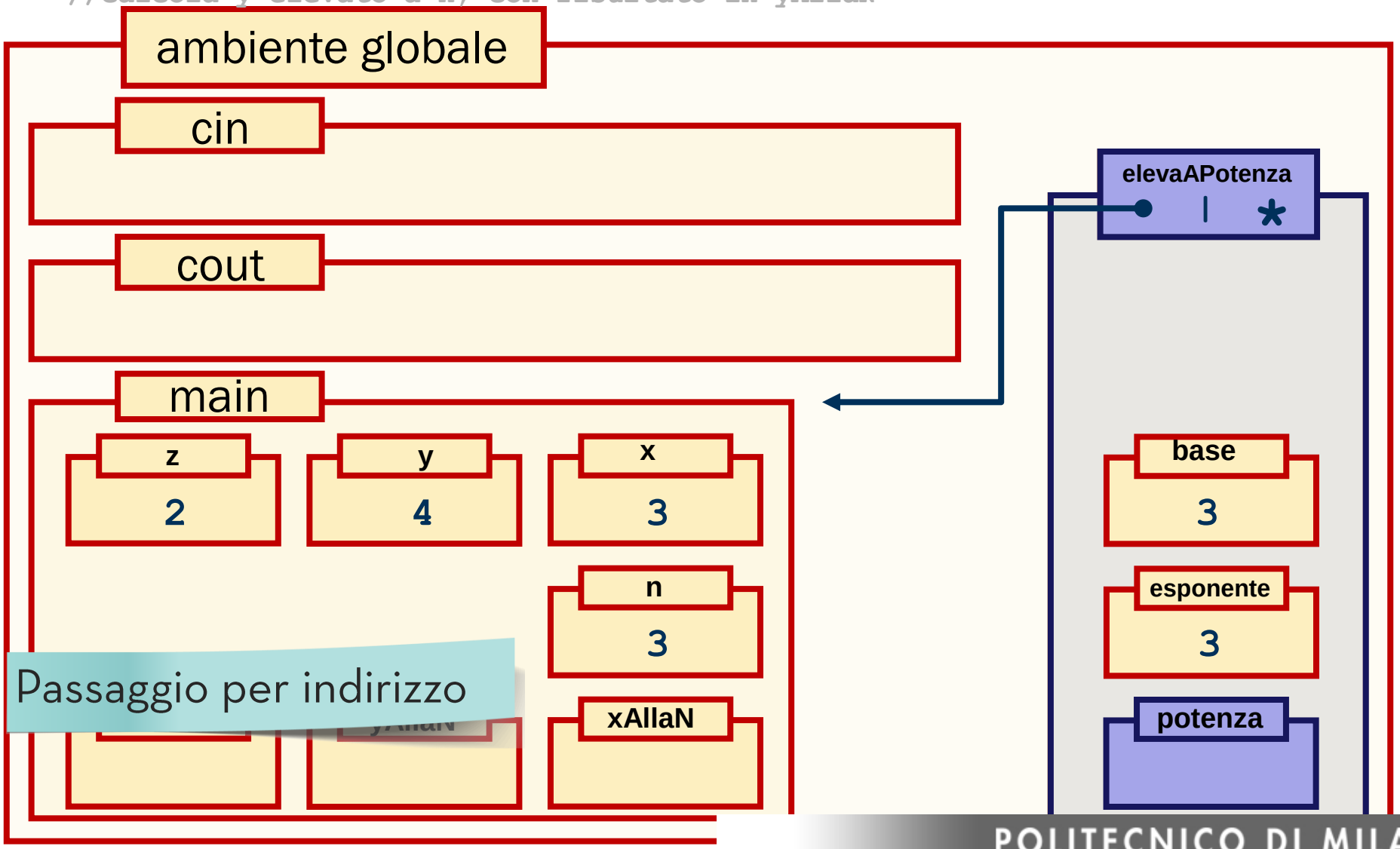
```
//calcola x elevato a n, con risultato in xAllaN
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita

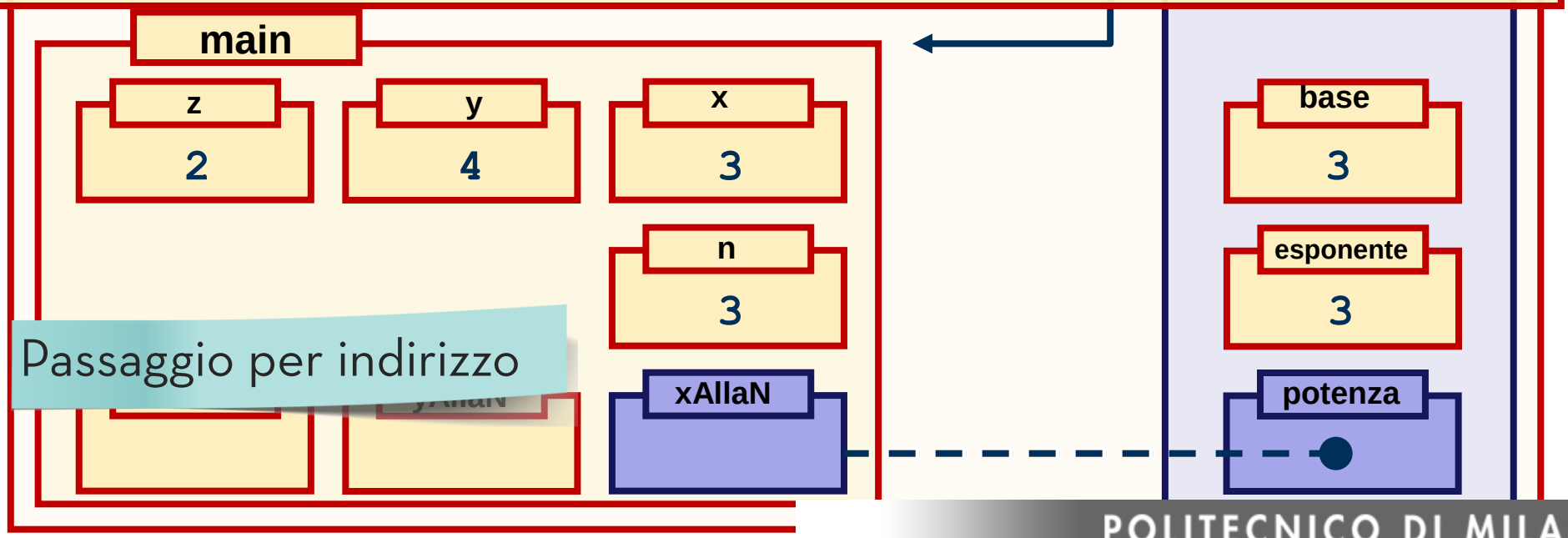


Passaggio per indirizzo

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

Parametri in uscita

```
potenza = 1;
. . .cin
potenza *= base;
. . .
cout
```

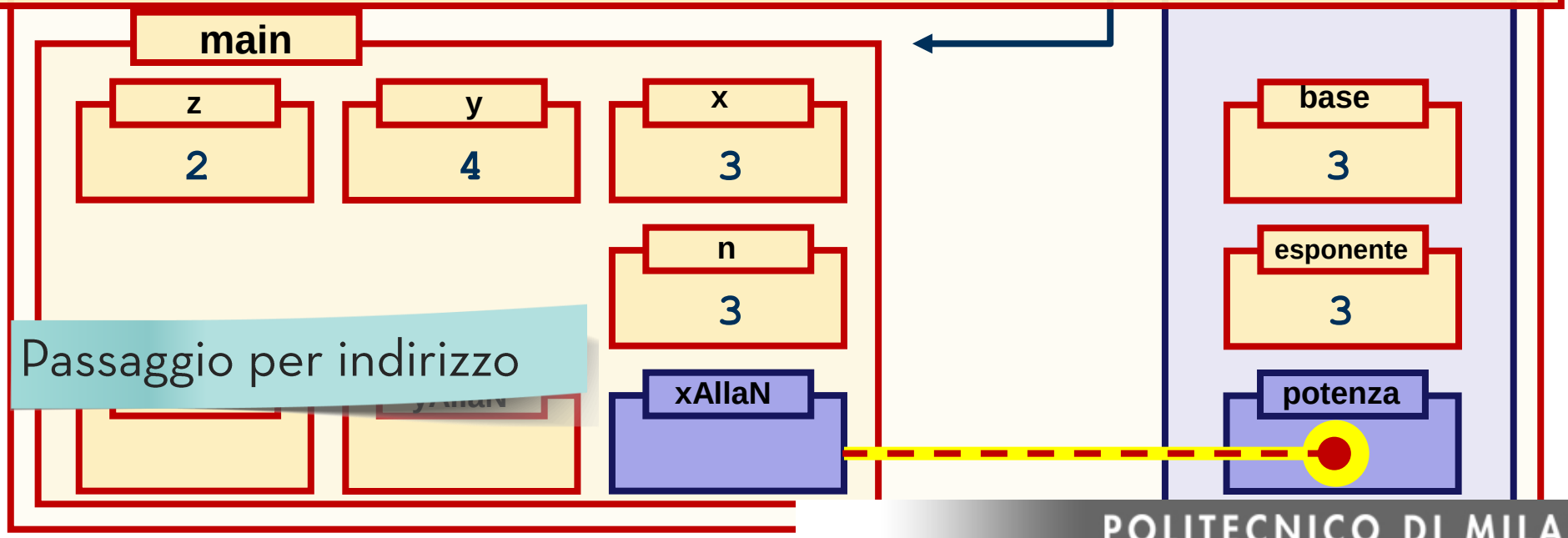


Passaggio per indirizzo

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

Parametri in uscita

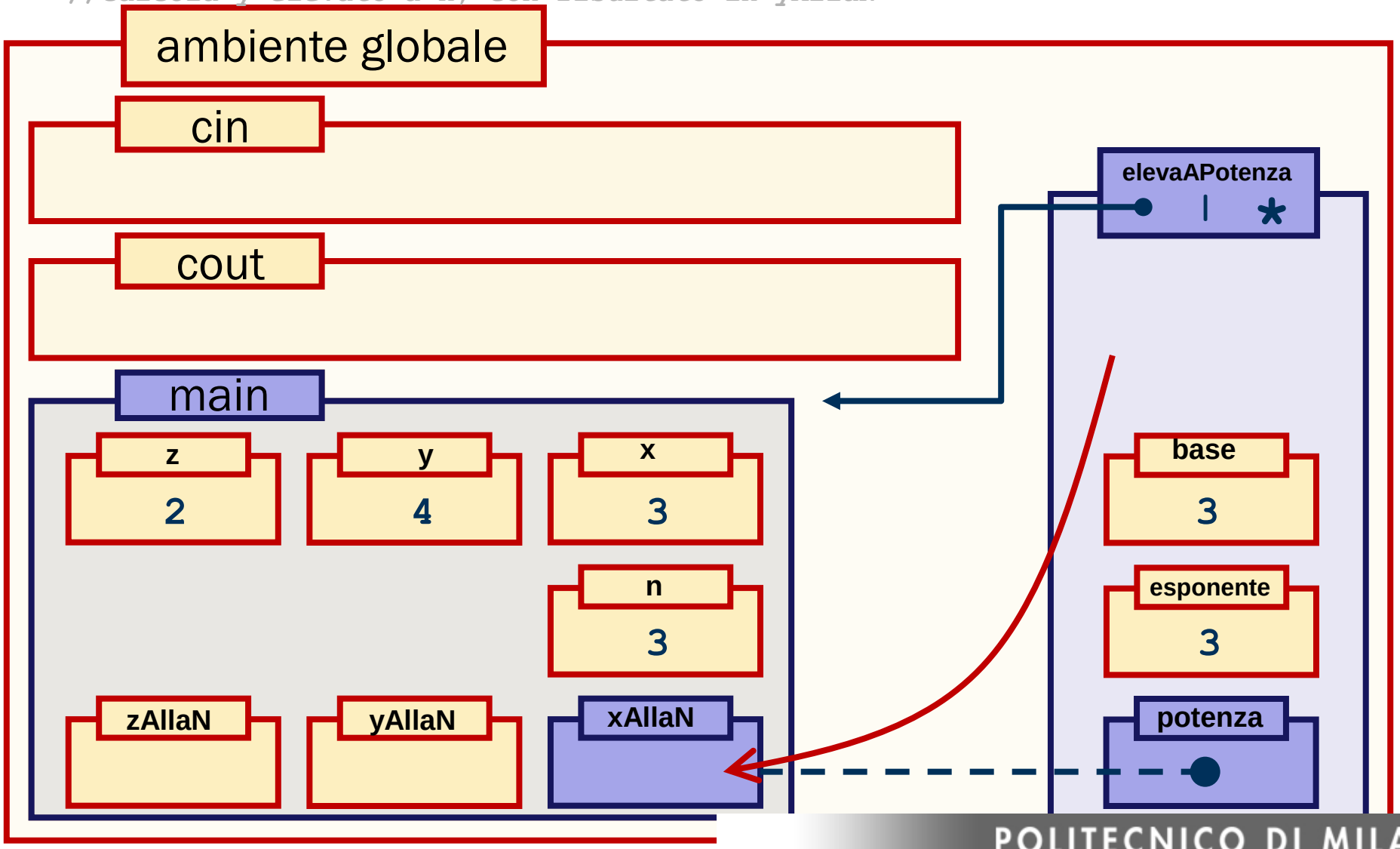
```
potenza = 1;
xAllaN
potenza *= base;
xAllaN
```



Passaggio per indirizzo

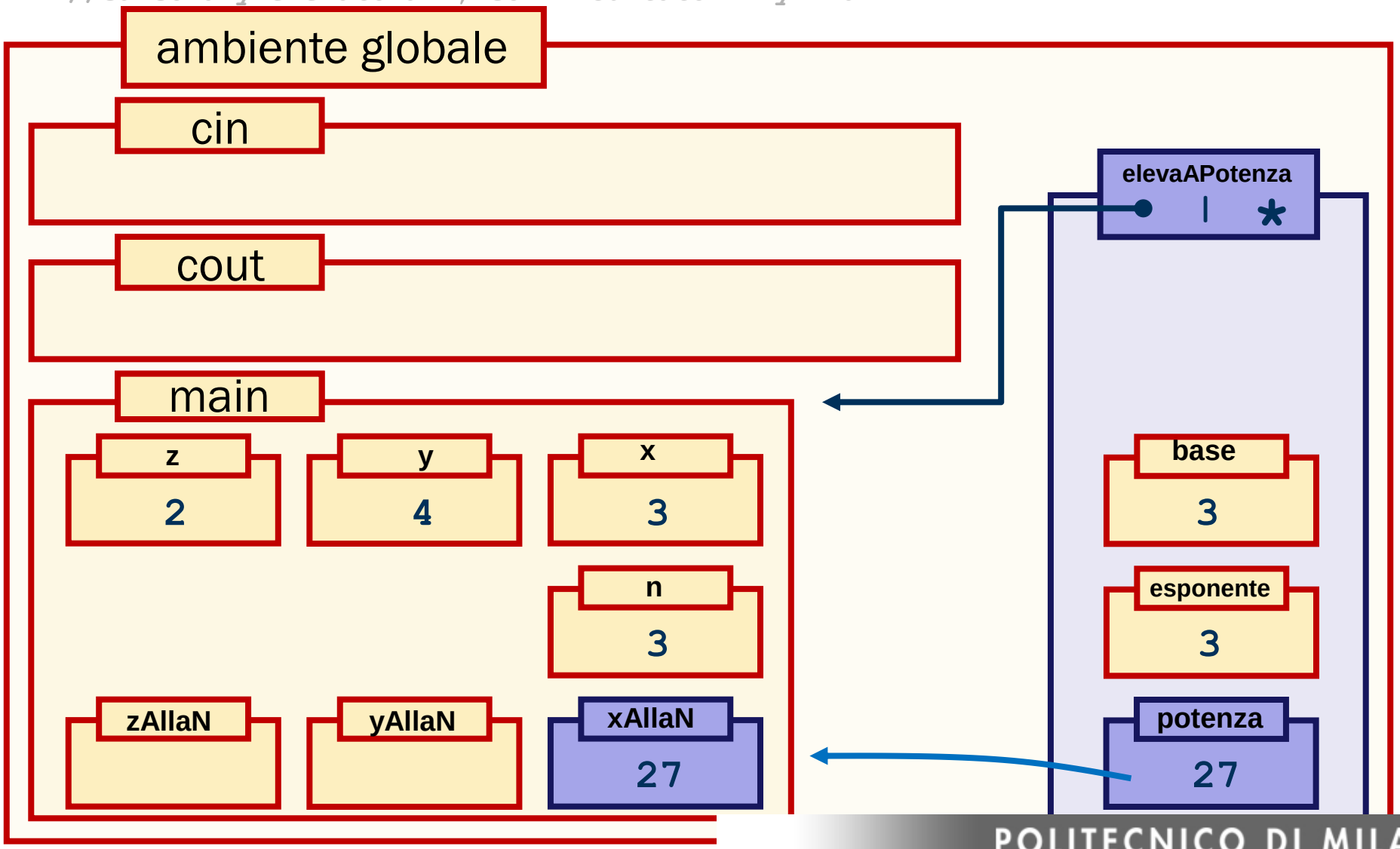
Parametri in uscita

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```



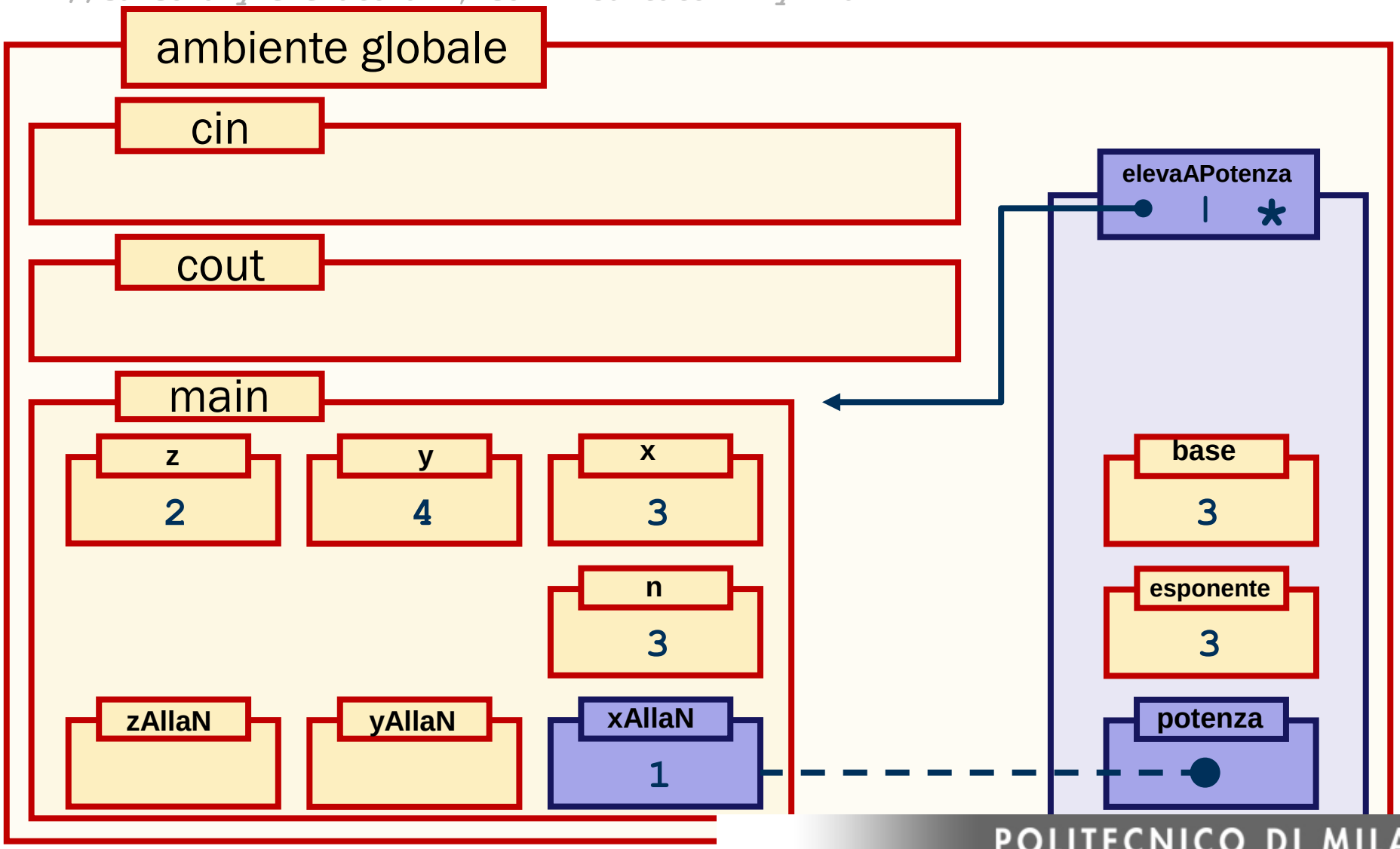
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



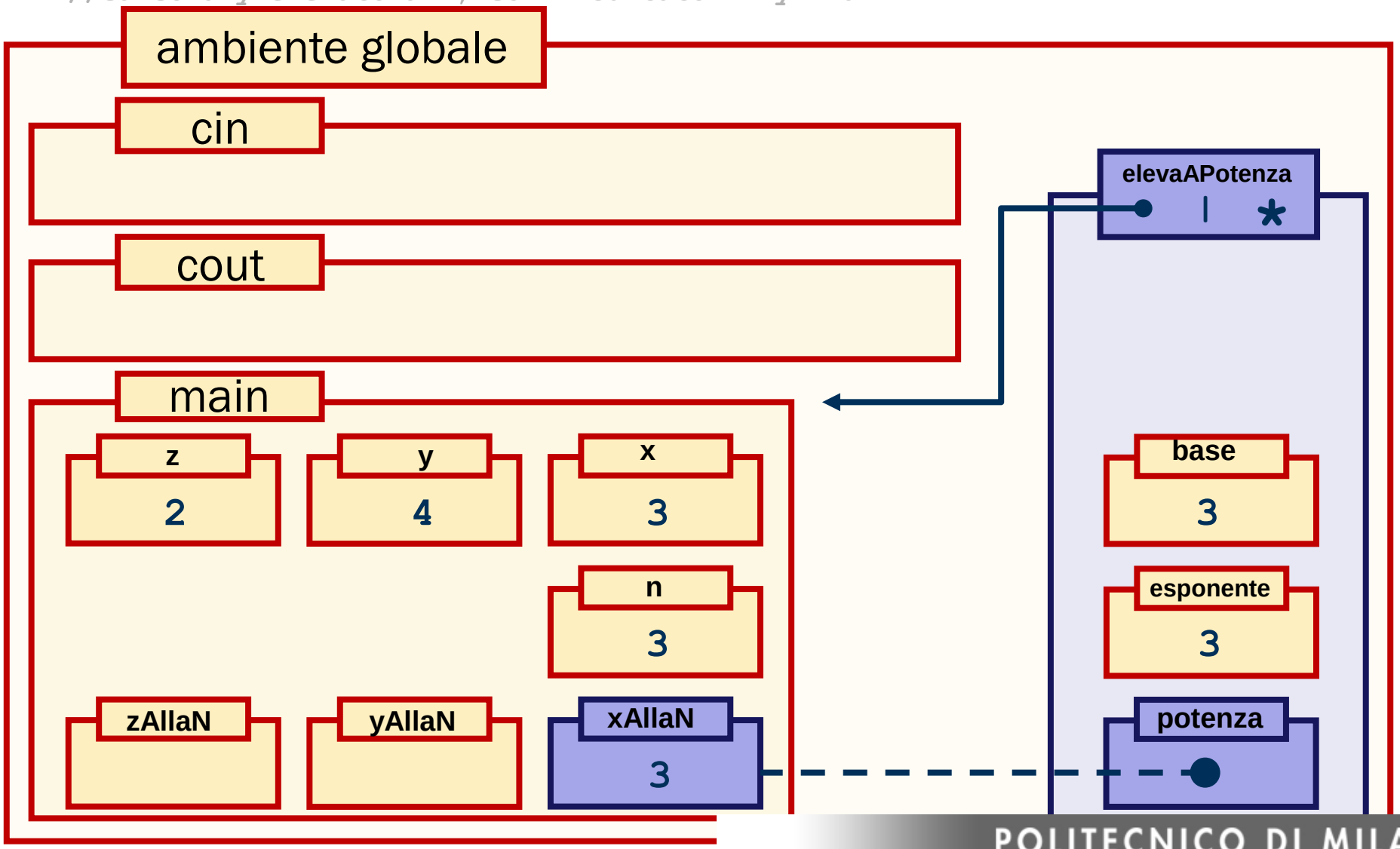
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



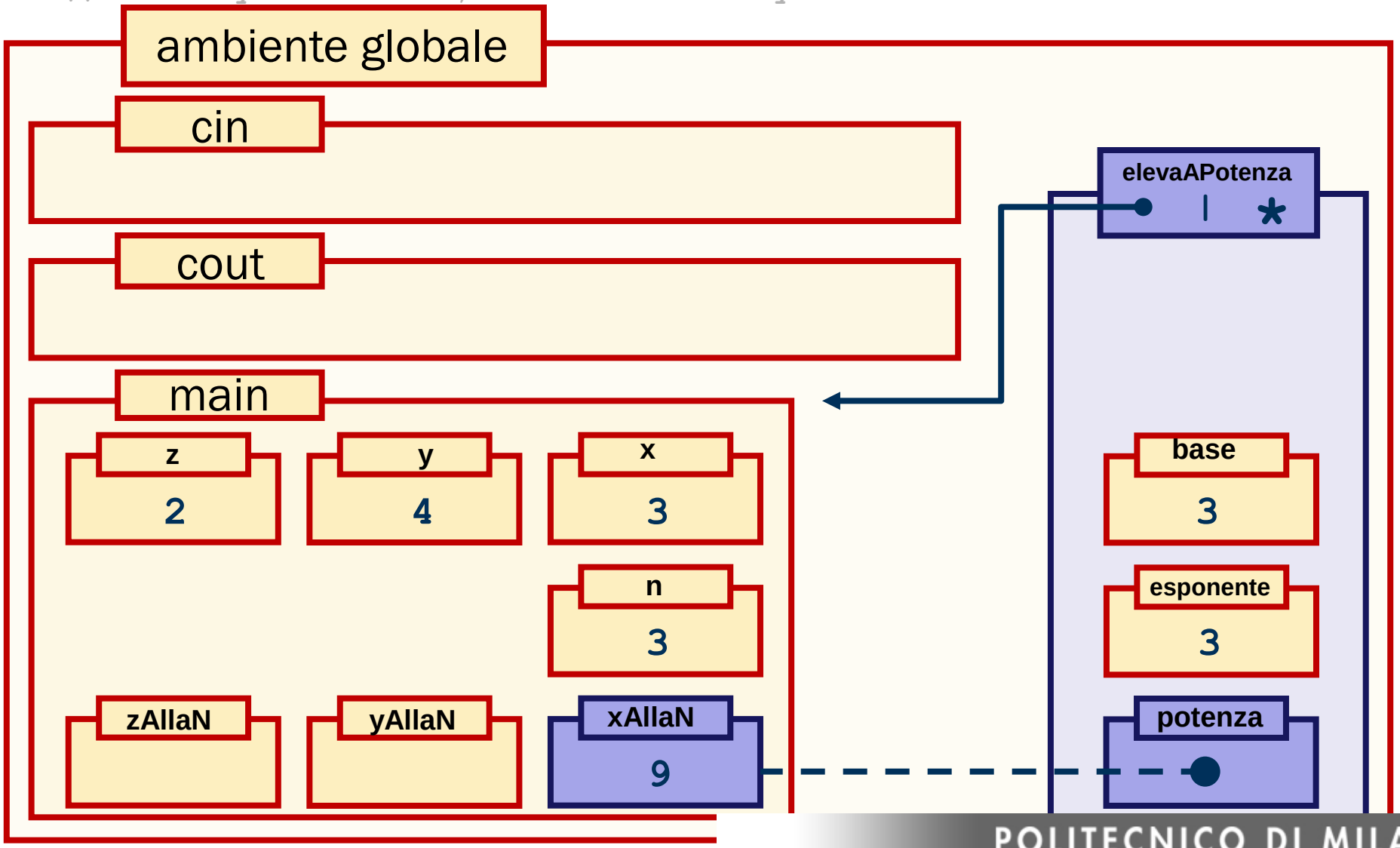
```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

Parametri in uscita



```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

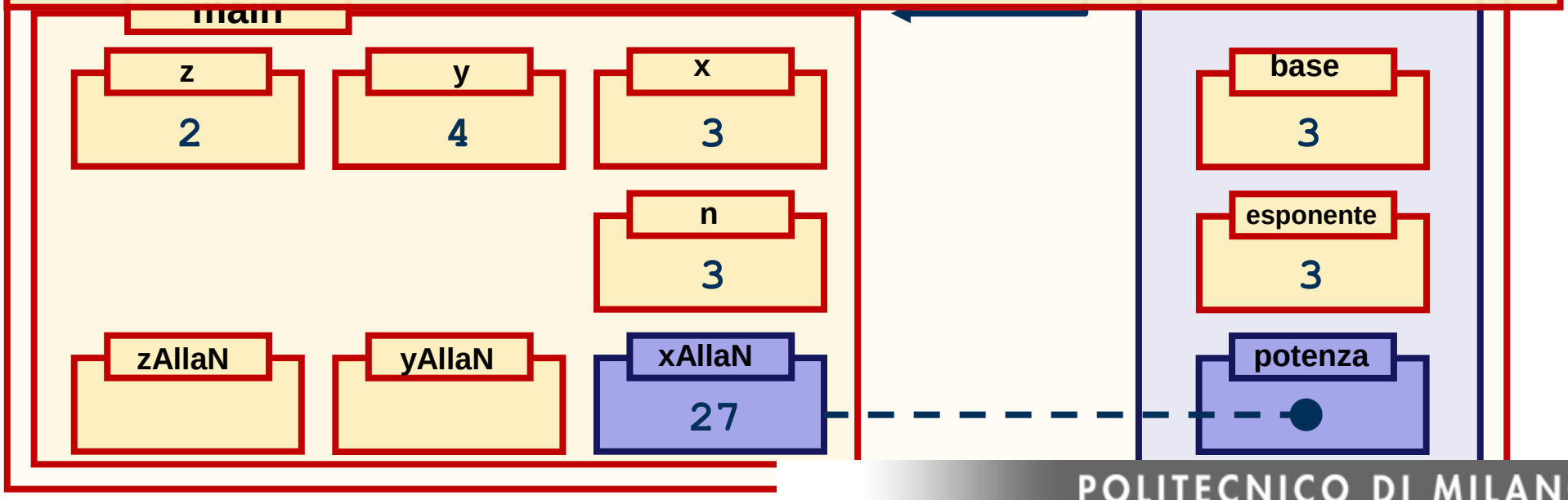
Parametri in uscita



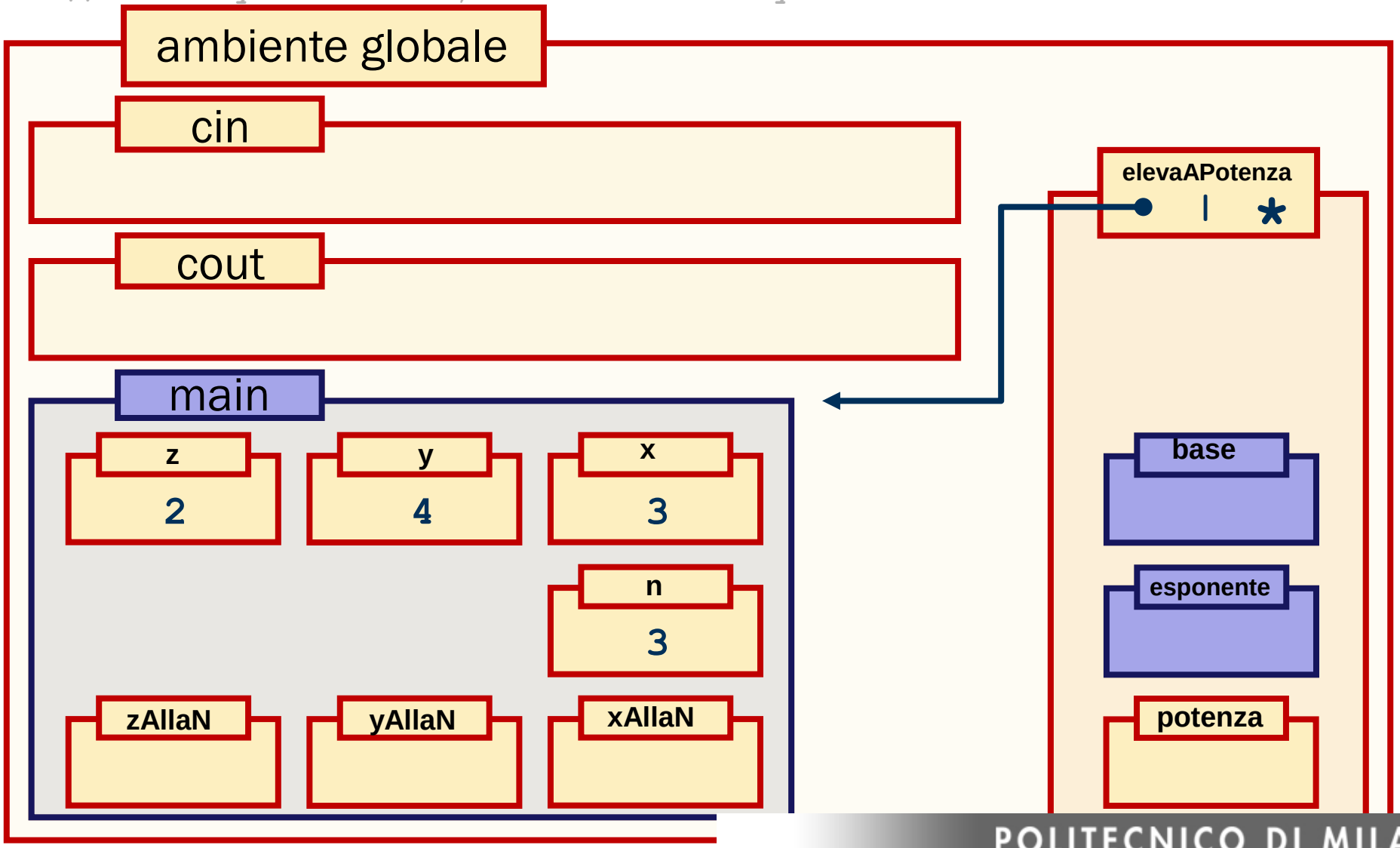
Parametri in uscita

```
//calcola x elevato a n, con risultato in x
  elevaAPotenza(x, n, xAllaN);
```

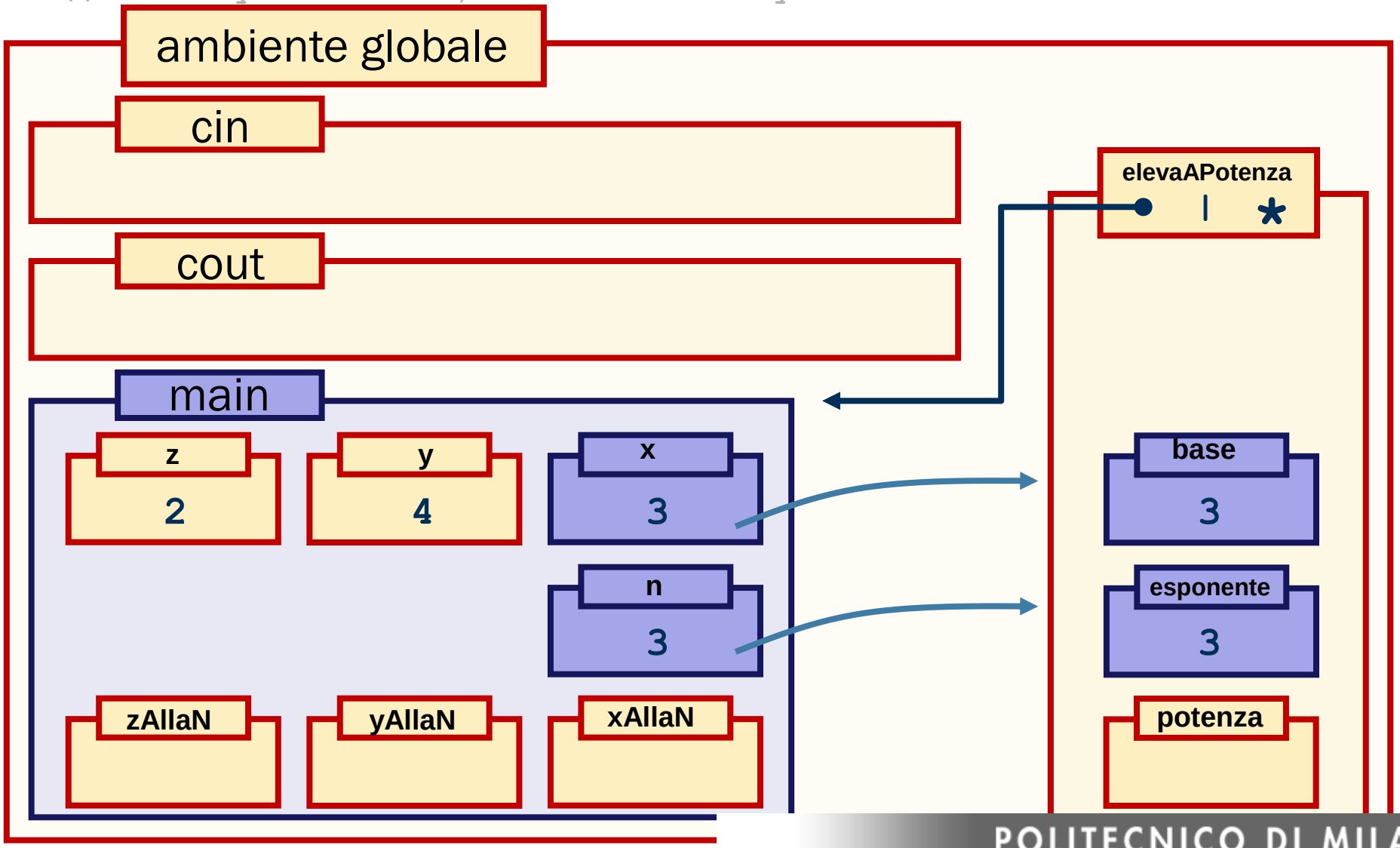
```
void elevaAPotenza (int base, int esponente, int & potenza)
{. . .}
void main()
{ . . .
     elevaAPotenza(x, n, xAllaN);
  . . .cout
}
```



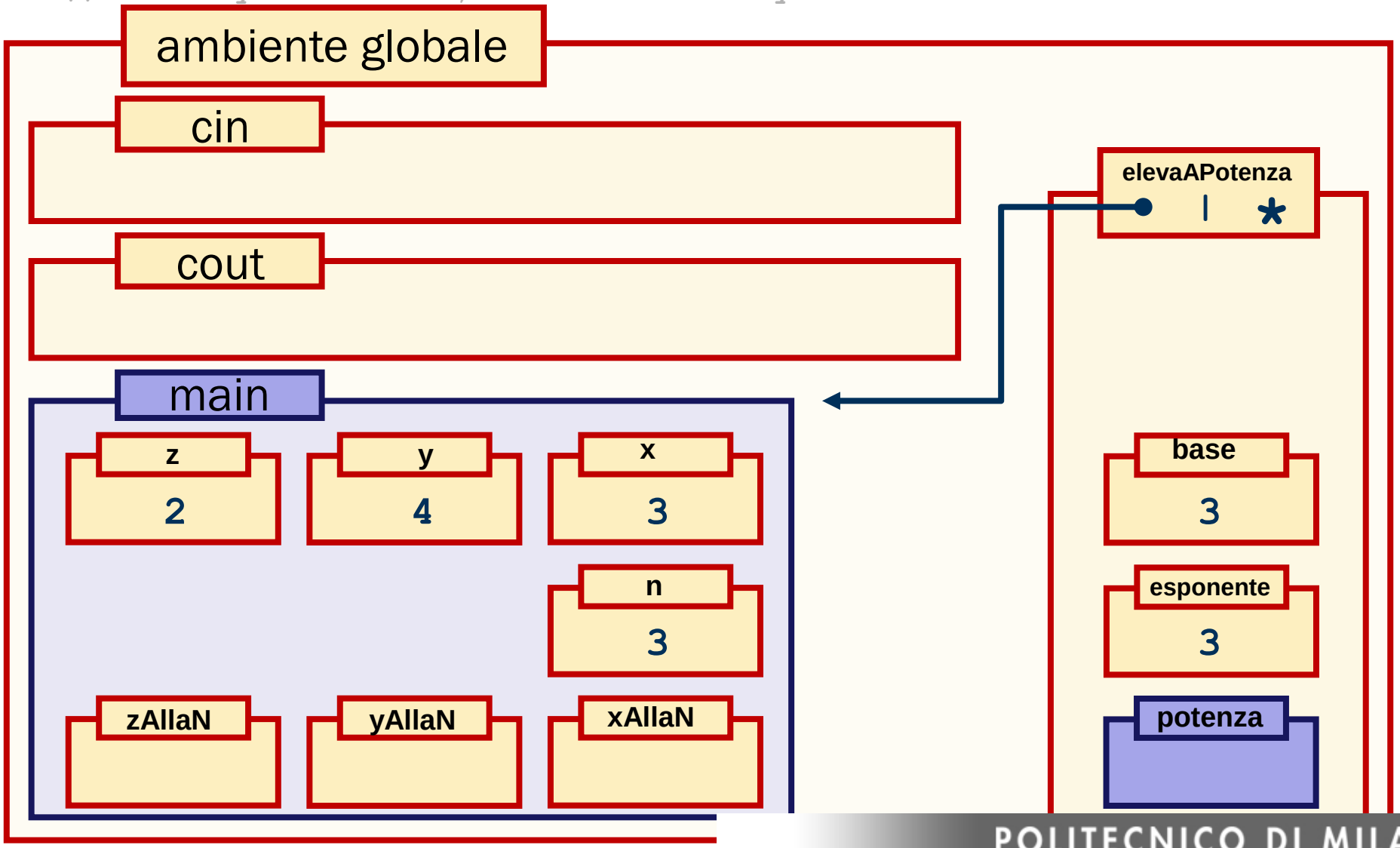
```
//calcola x elevato a n, con risultato in xAllaN
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```



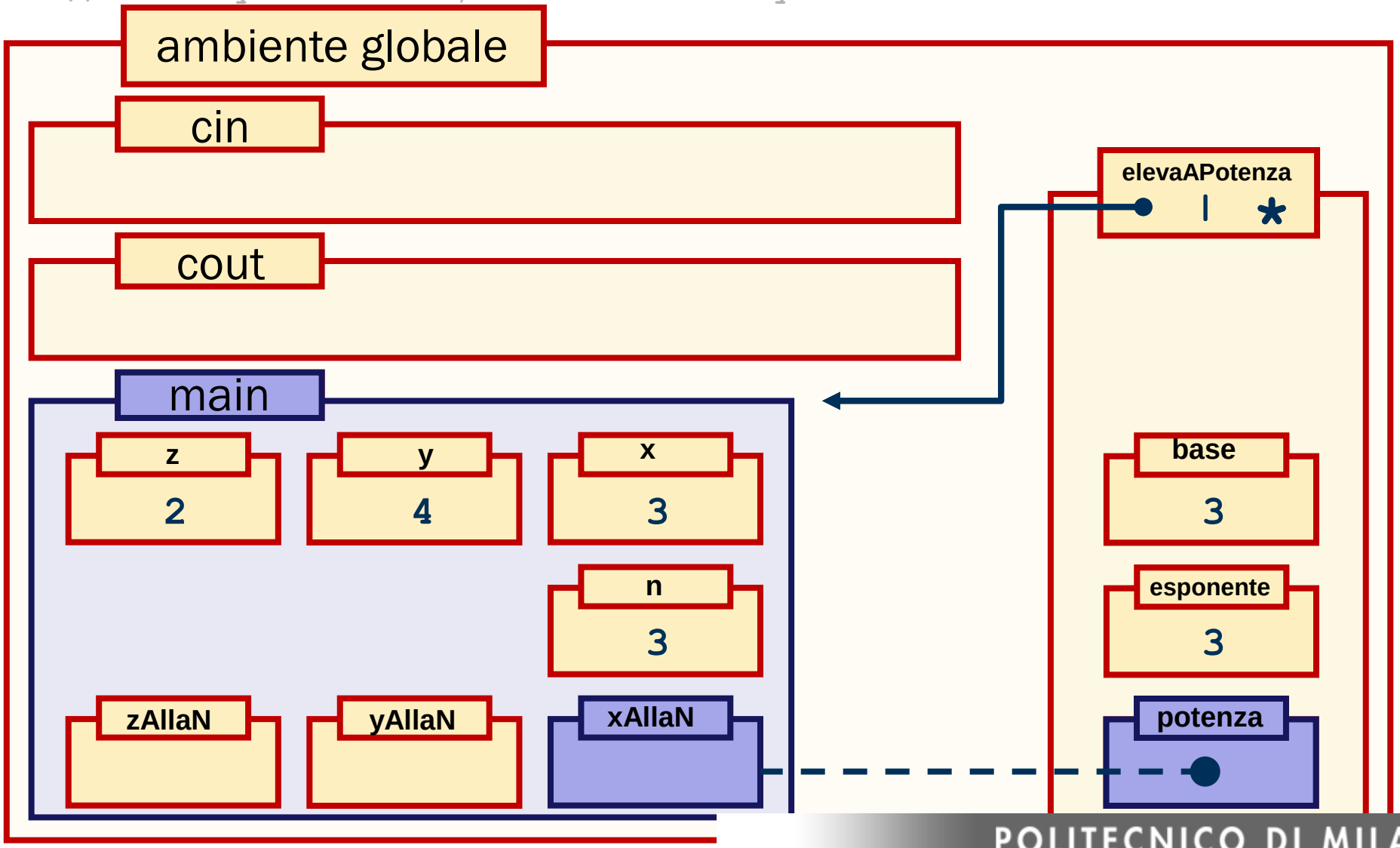
```
//calcola x elevato a n, con risultato in xAllaN
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```



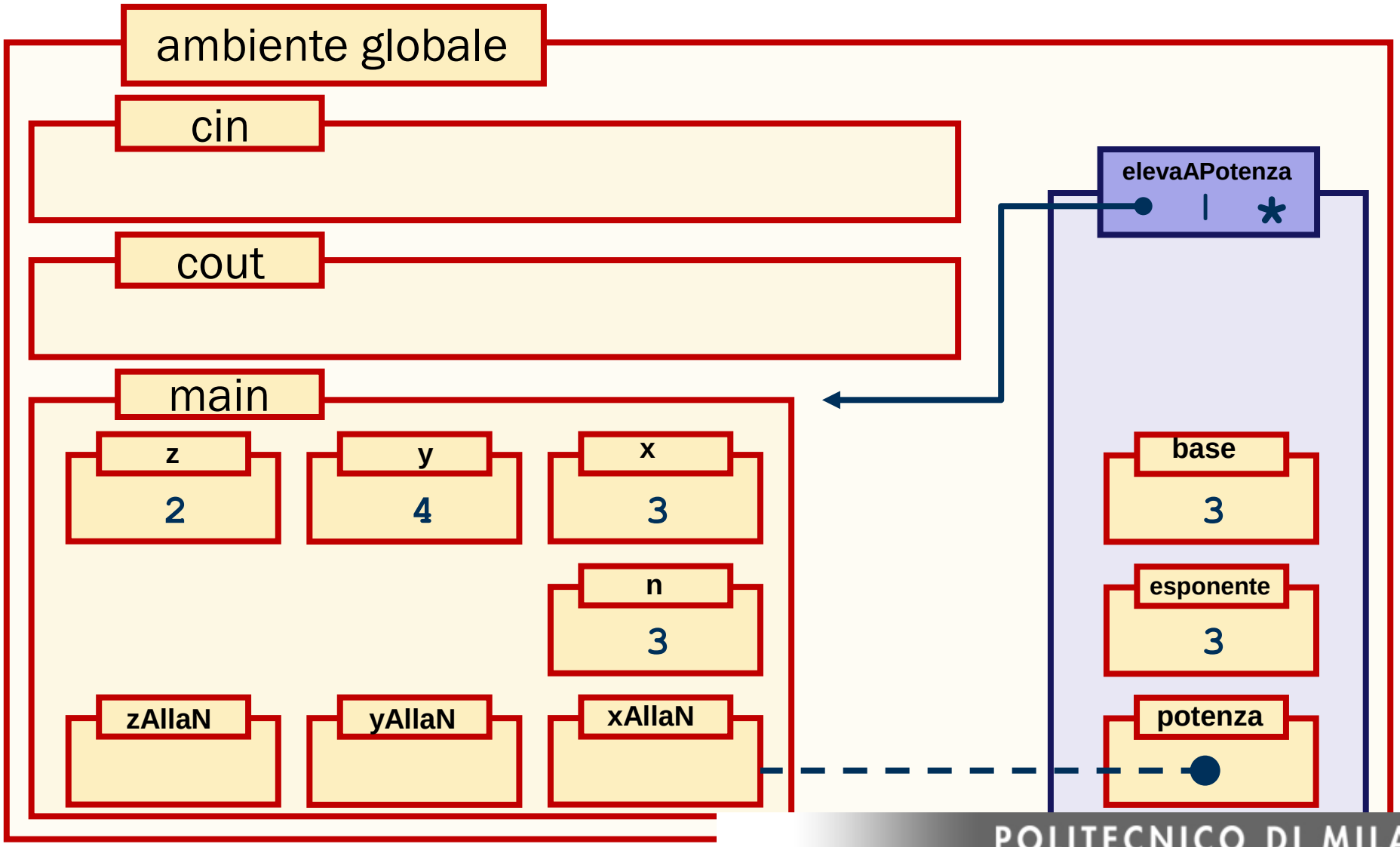
```
//calcola x elevato a n, con risultato in xAllaN
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```



```
//calcola x elevato a n, con risultato in xAllaN
  elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
```

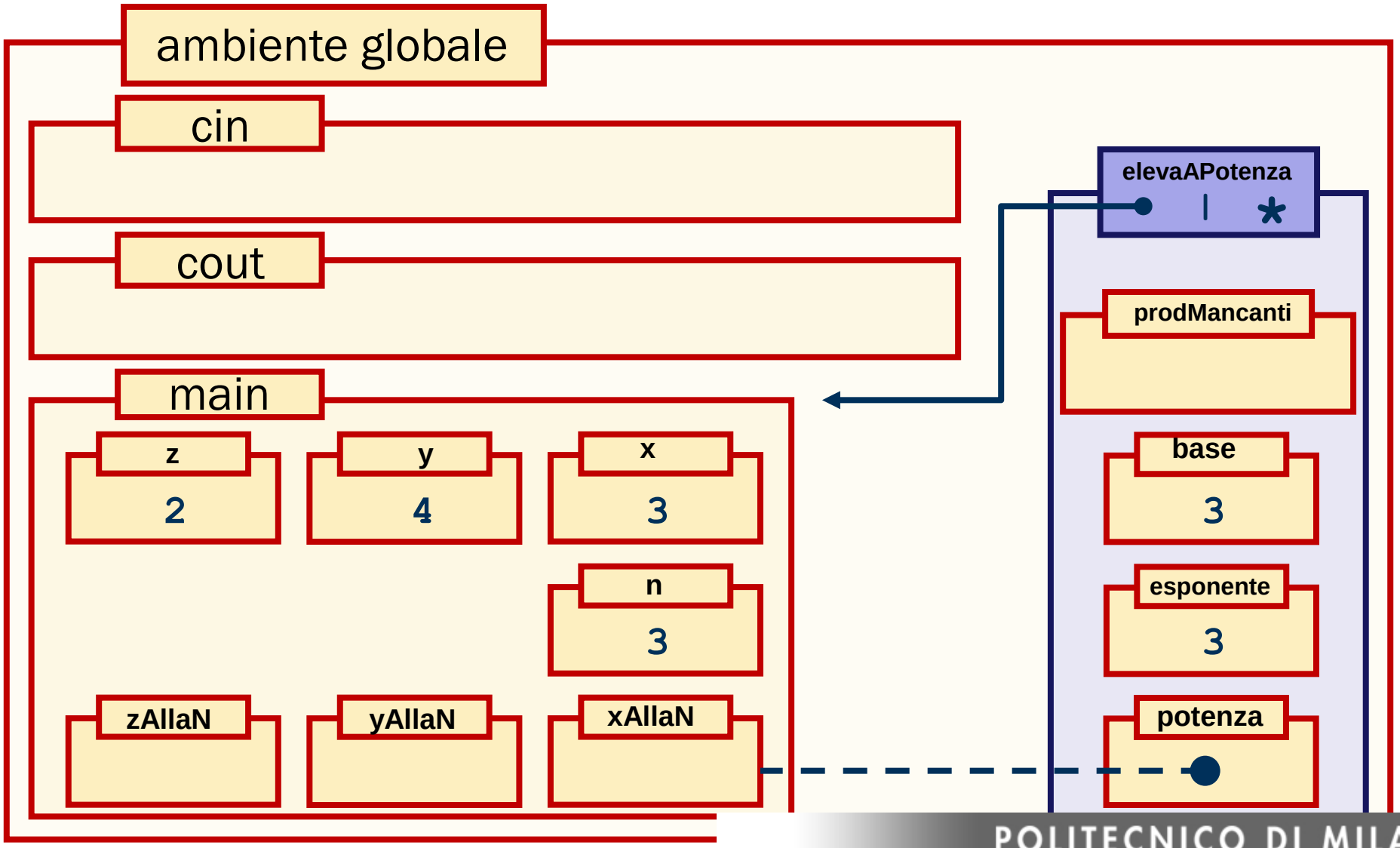


```
void elevaAPotenza(int base, int esponente, int & potenza)
{//versione per esponente positivo
   int prodMancanti;                    //variabile locale ad elevaAPotenza
```

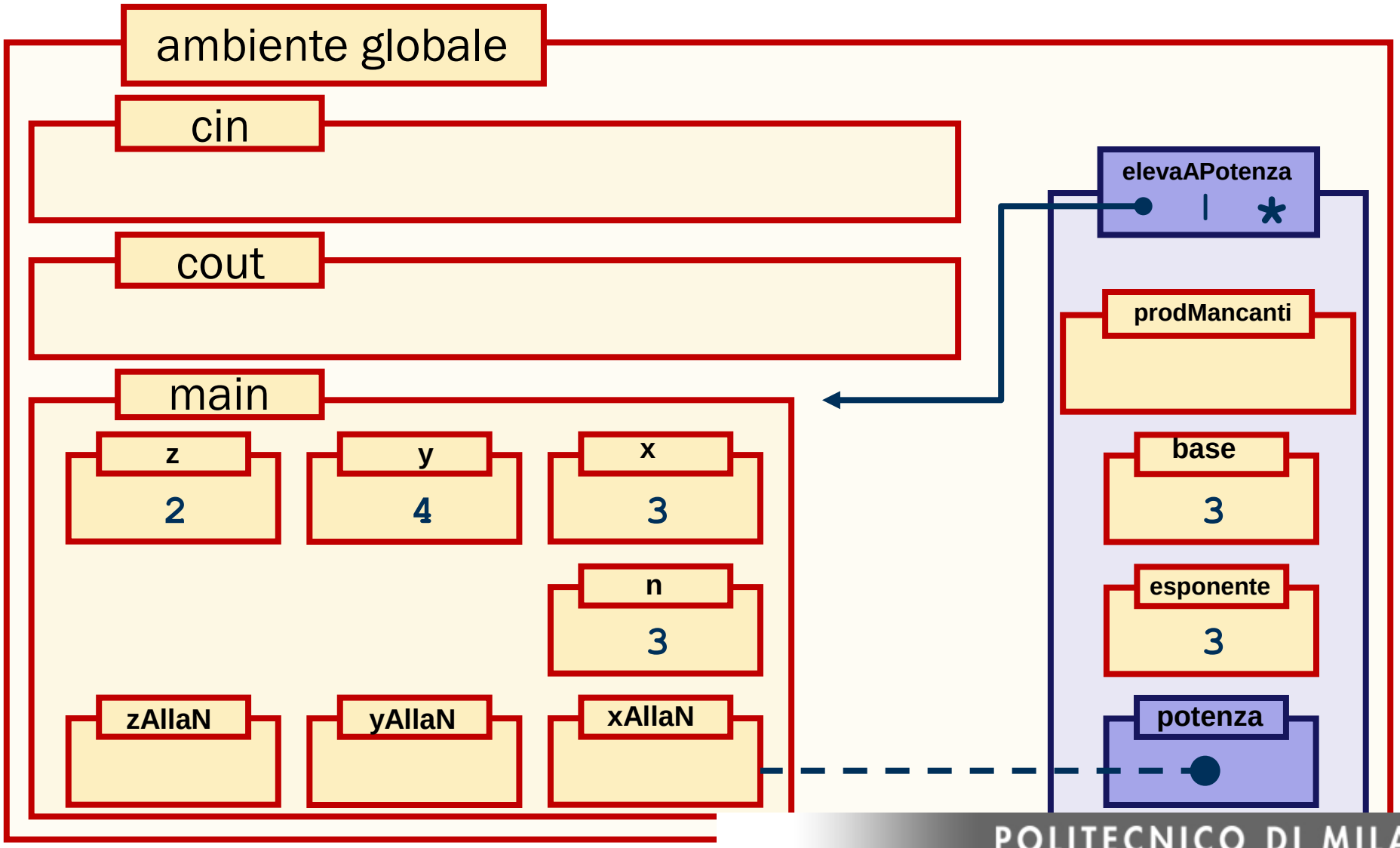


Passaggio per indirizzo

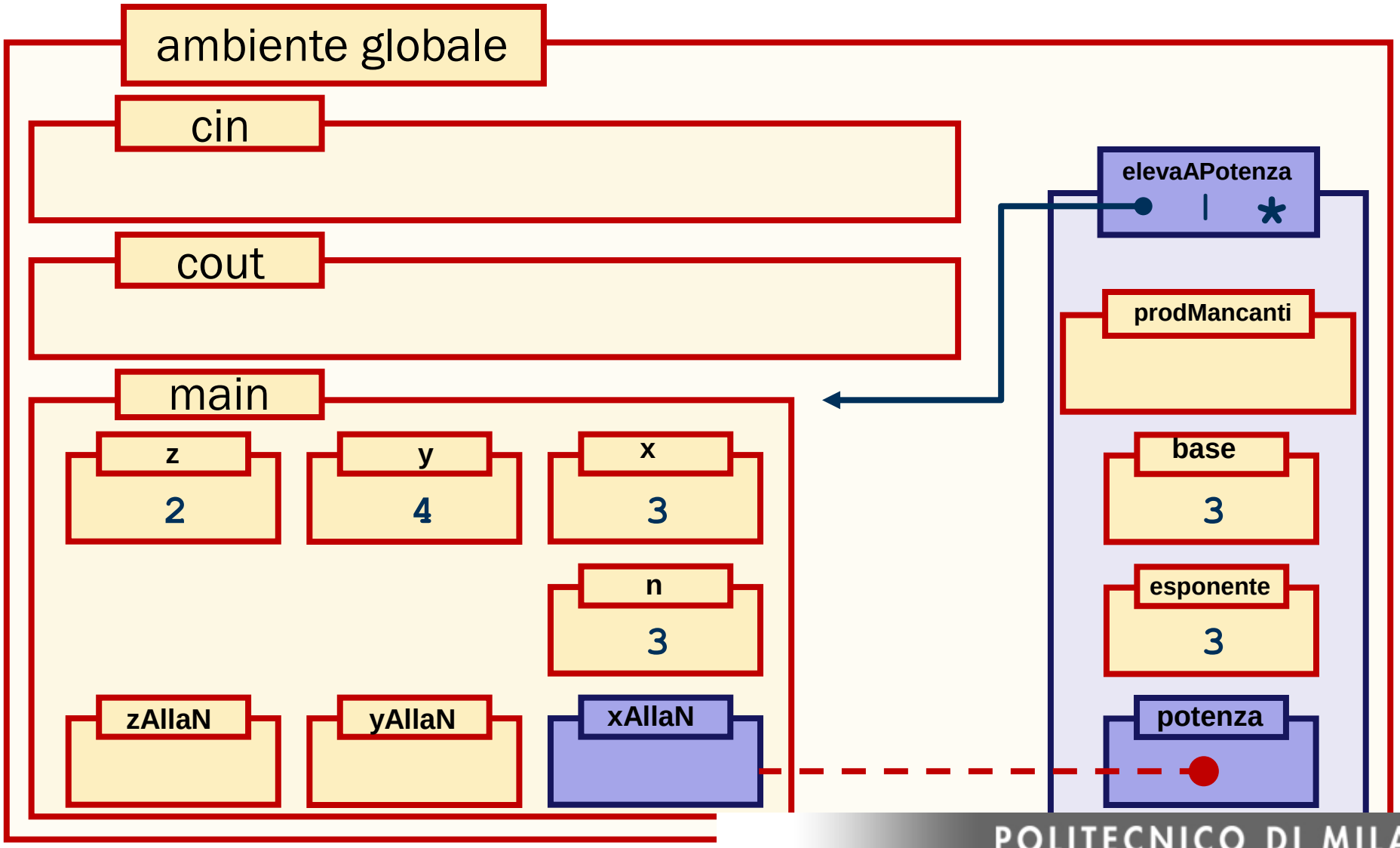
```
{//versione per esponente positivo
   int prodMancanti;                    //variabile locale ad elevaAPotenza
   potenza = 1;
```



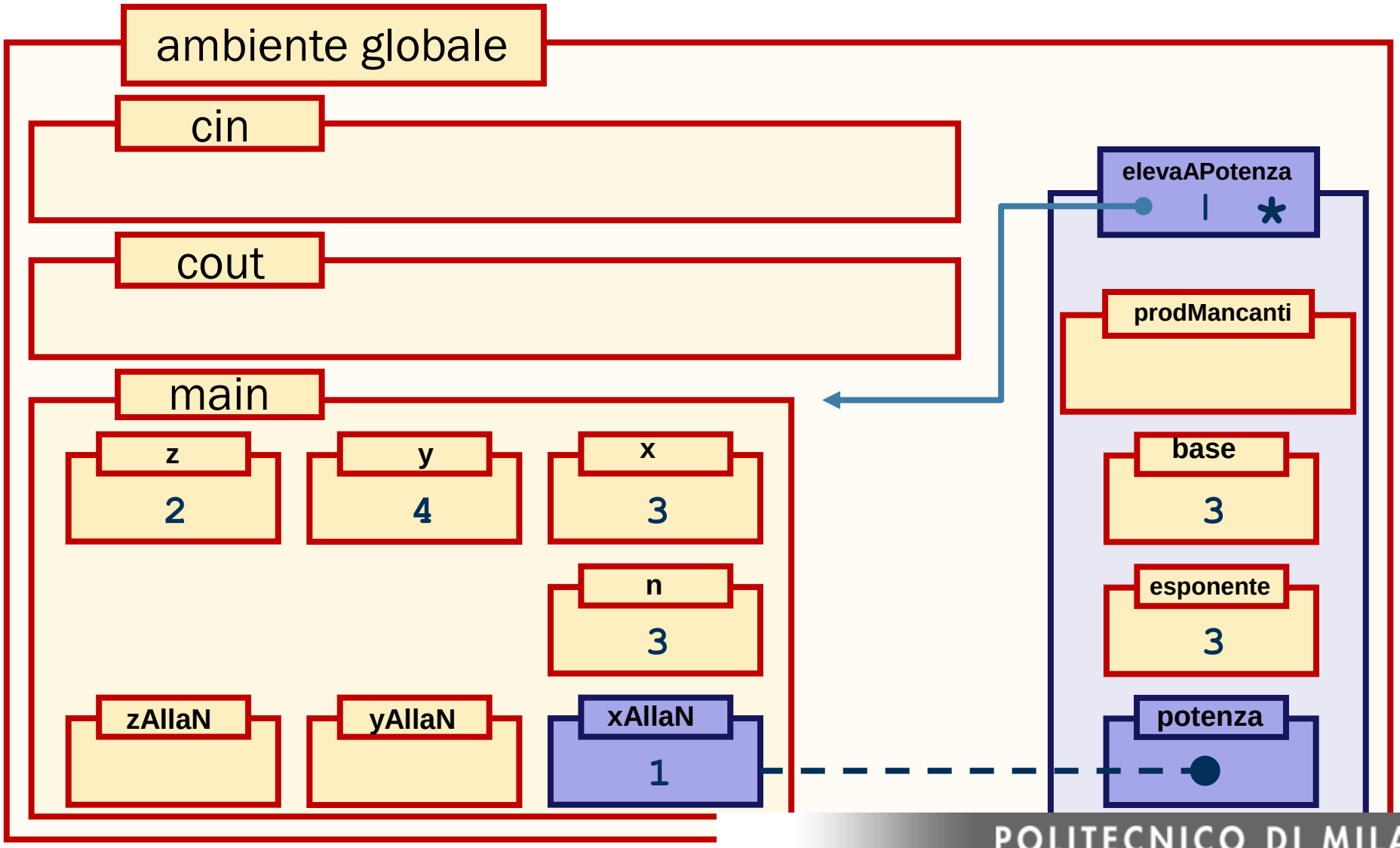
```
int prodMancanti;          //variabile locale ad elevaAPotenza
potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
```



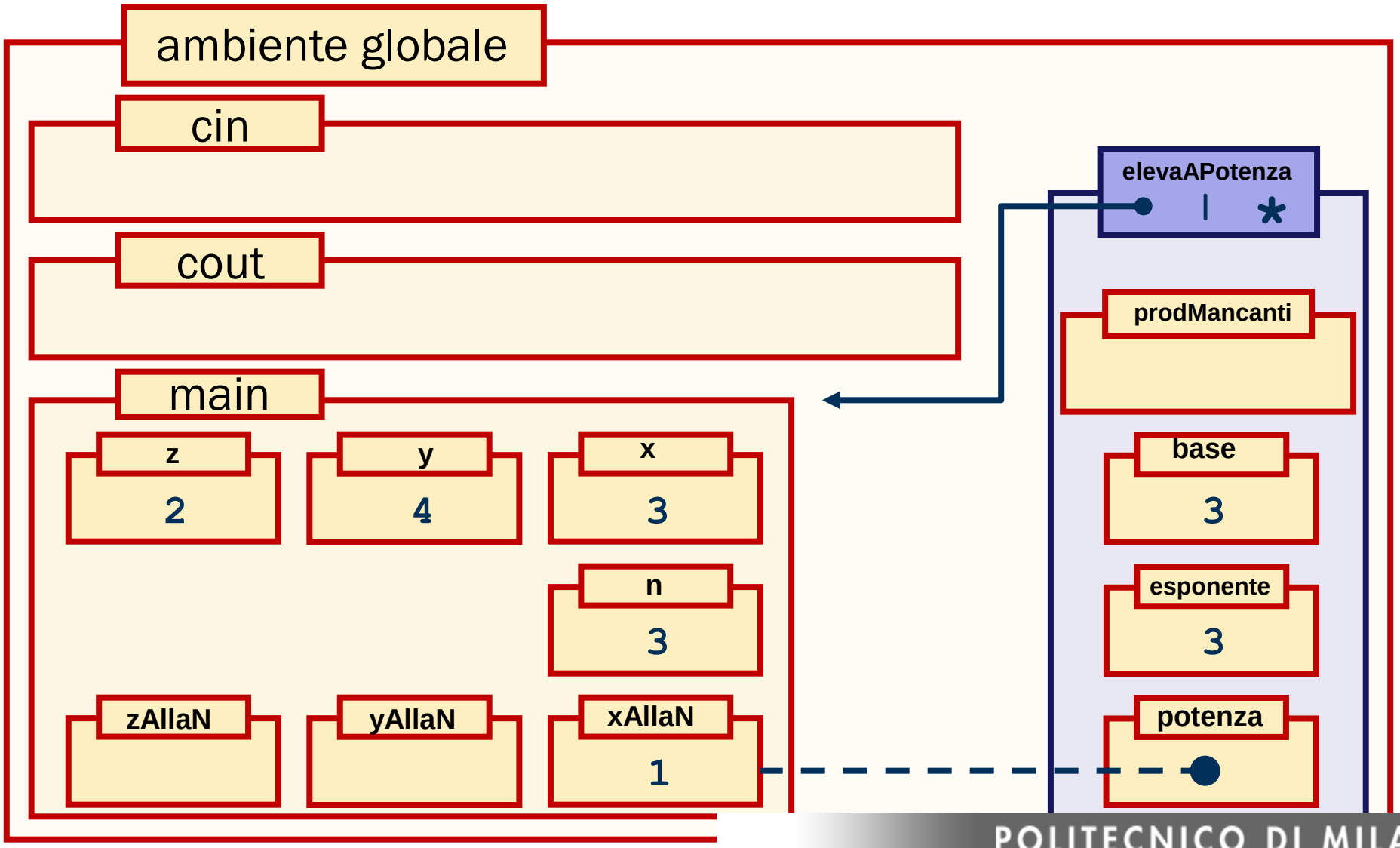
```
int prodMancanti;          //variabile locale ad elevaAPotenza
potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
```



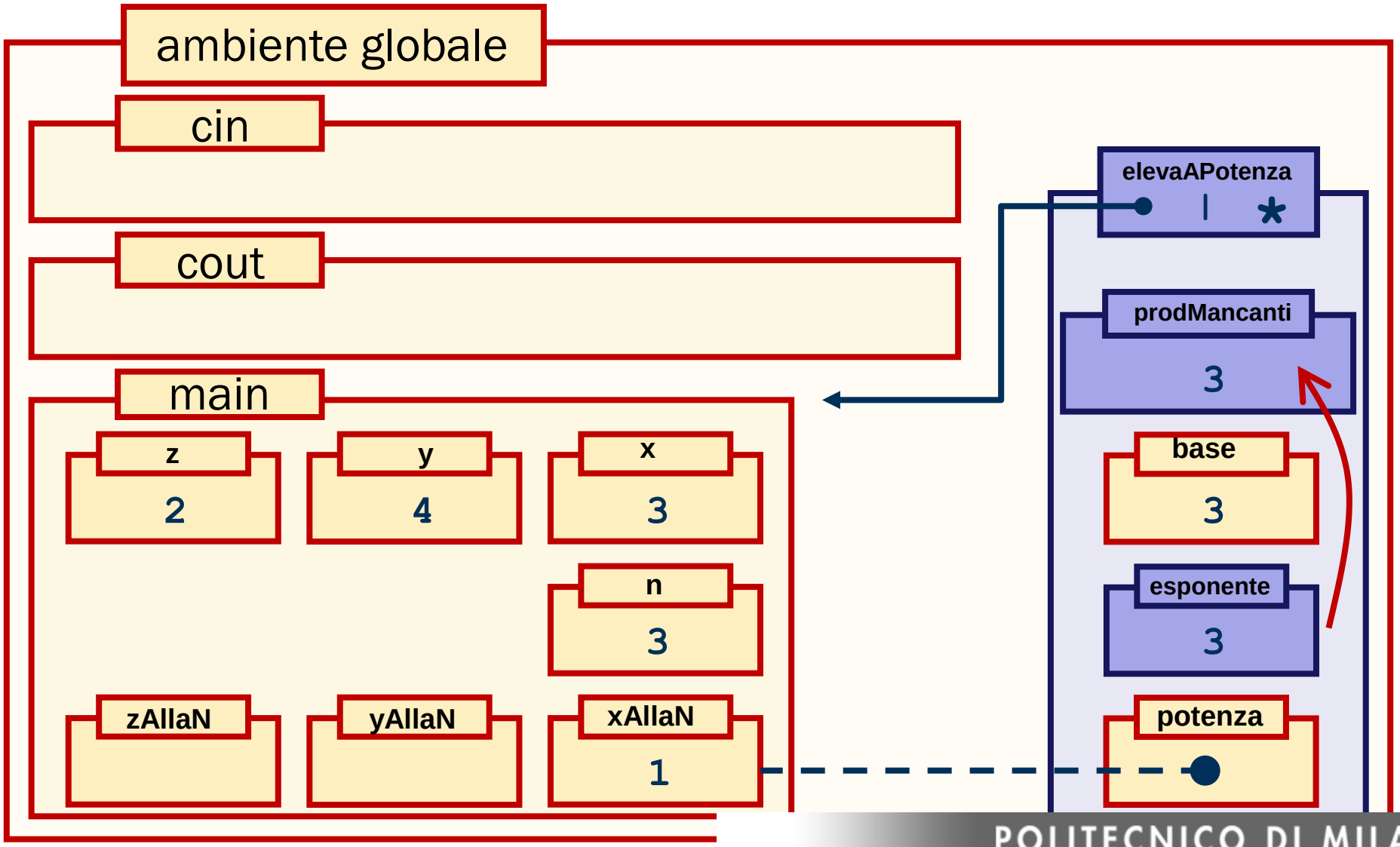
```
int prodMancanti;          //variabile locale ad elevaAPotenza
potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
```



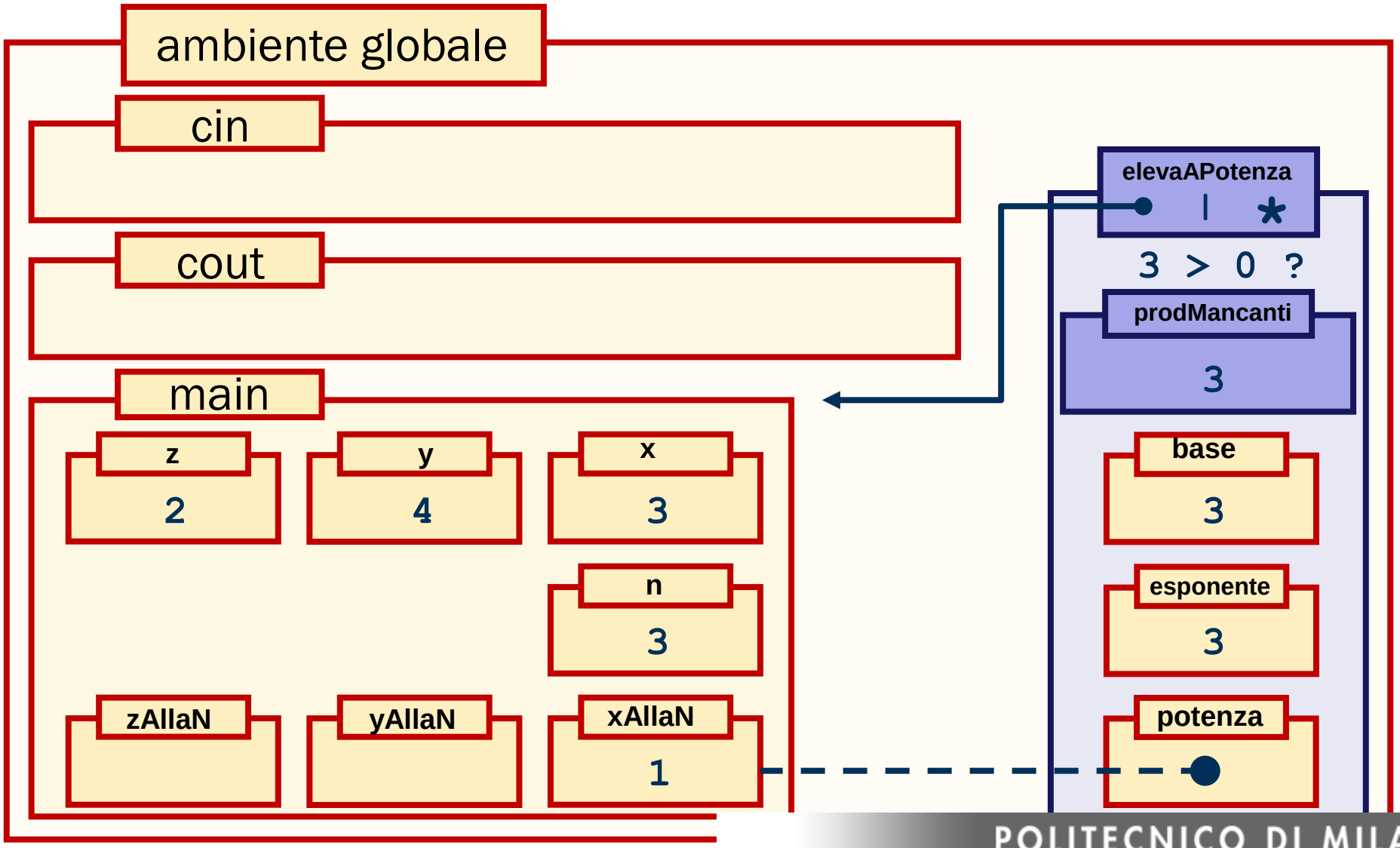
```
potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
```



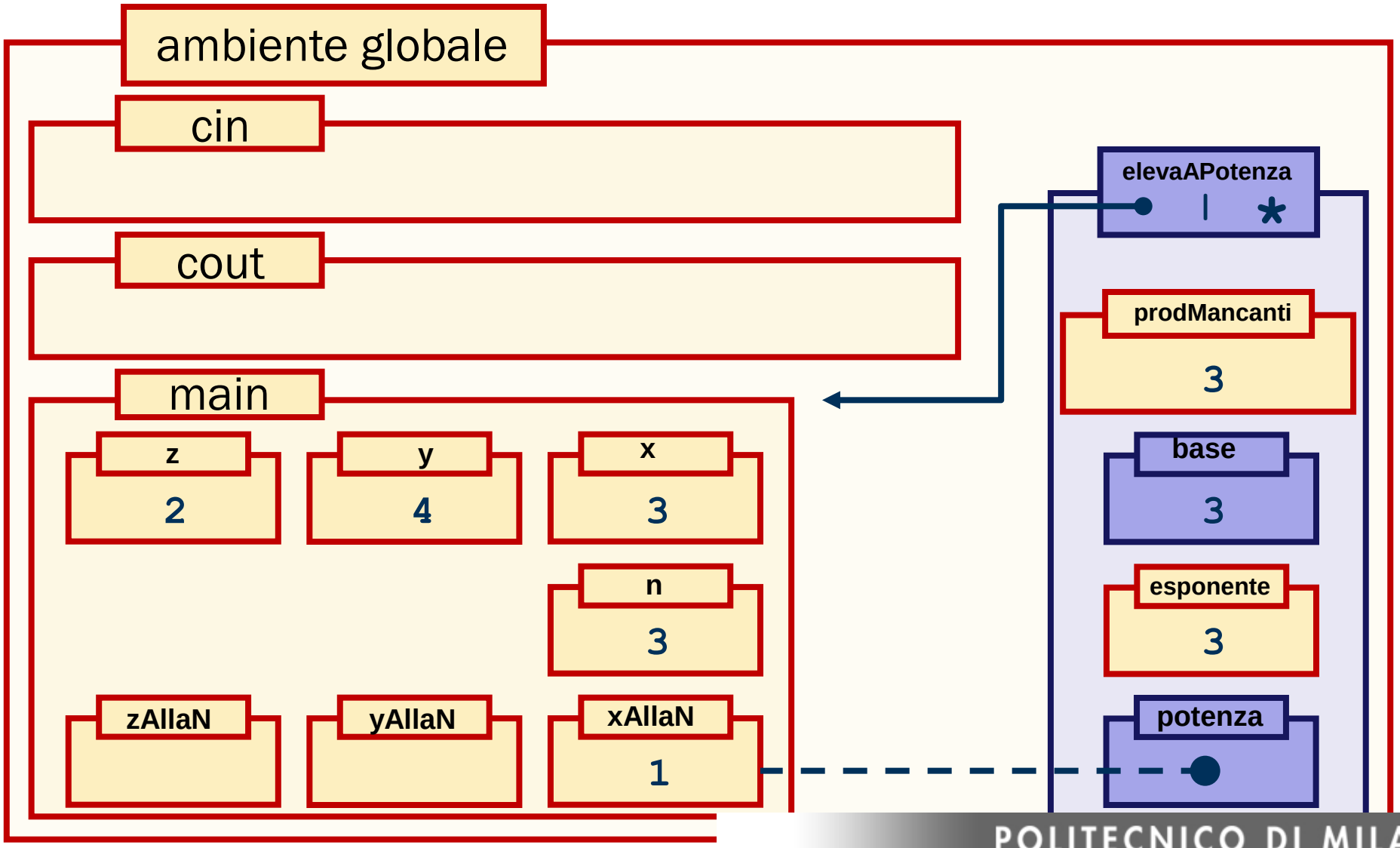
```
potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
```



```
potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
```

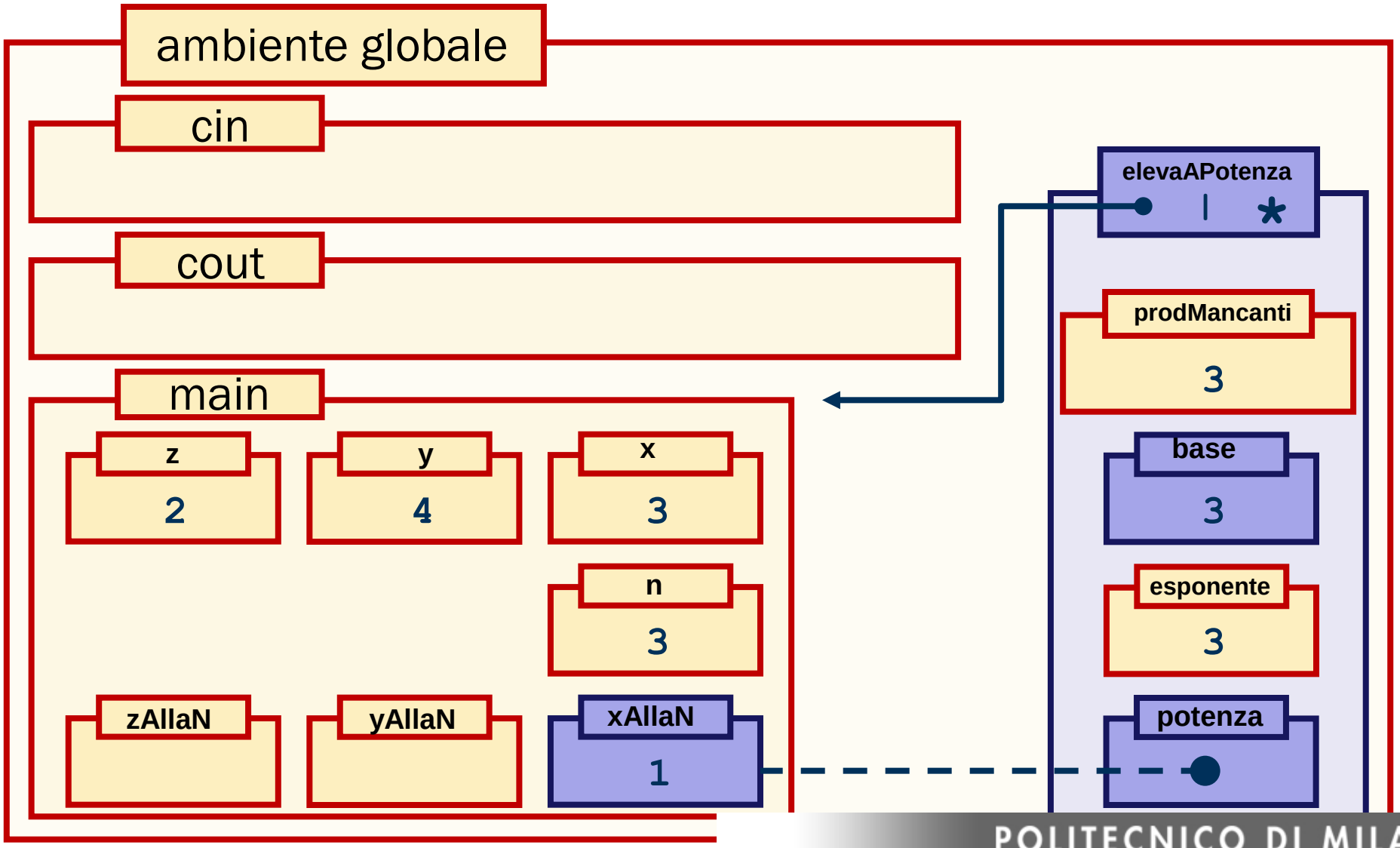


```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
  potenza *= base;
}
```



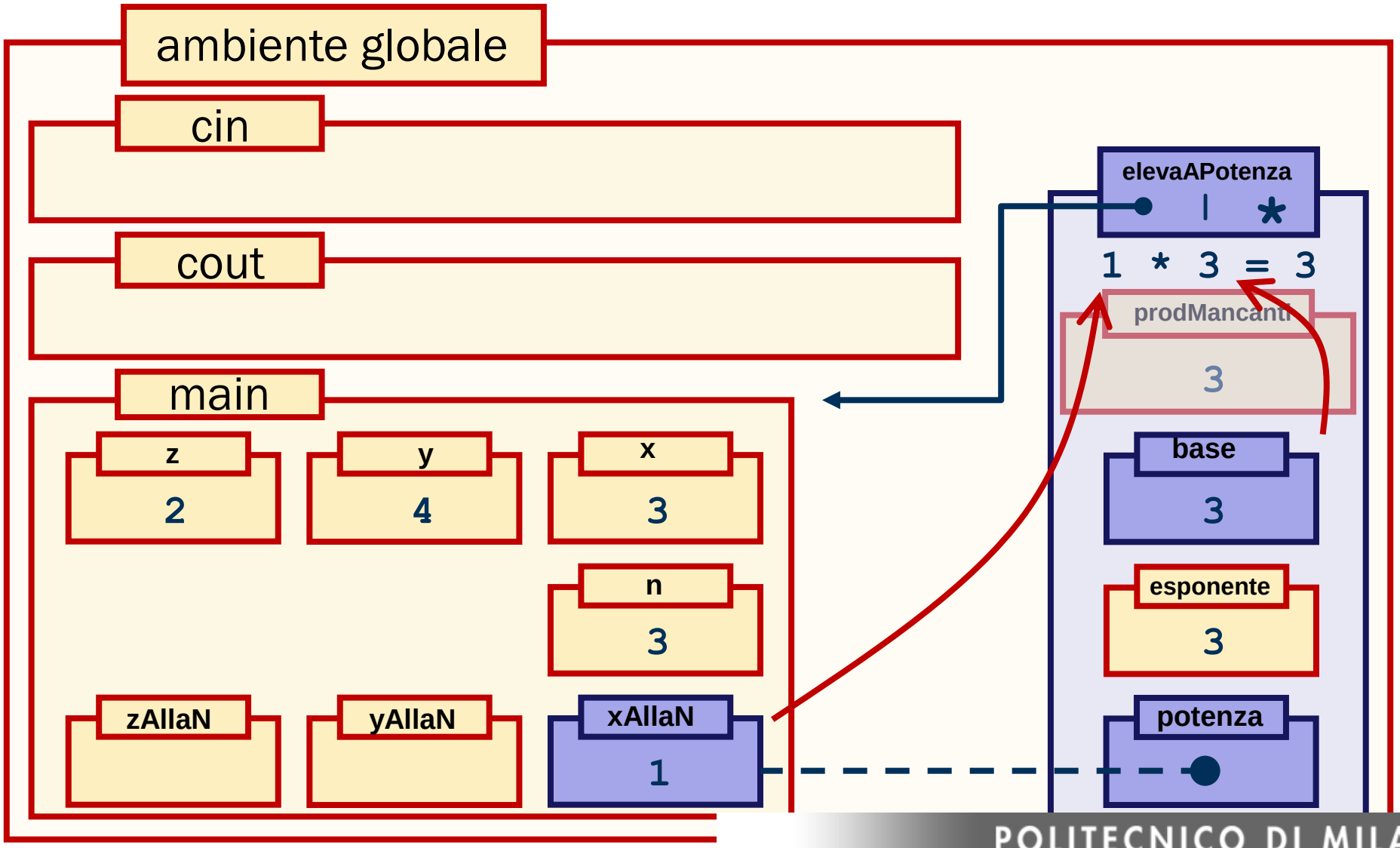
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
  potenza *= base;
}
```

xAllaN *= base



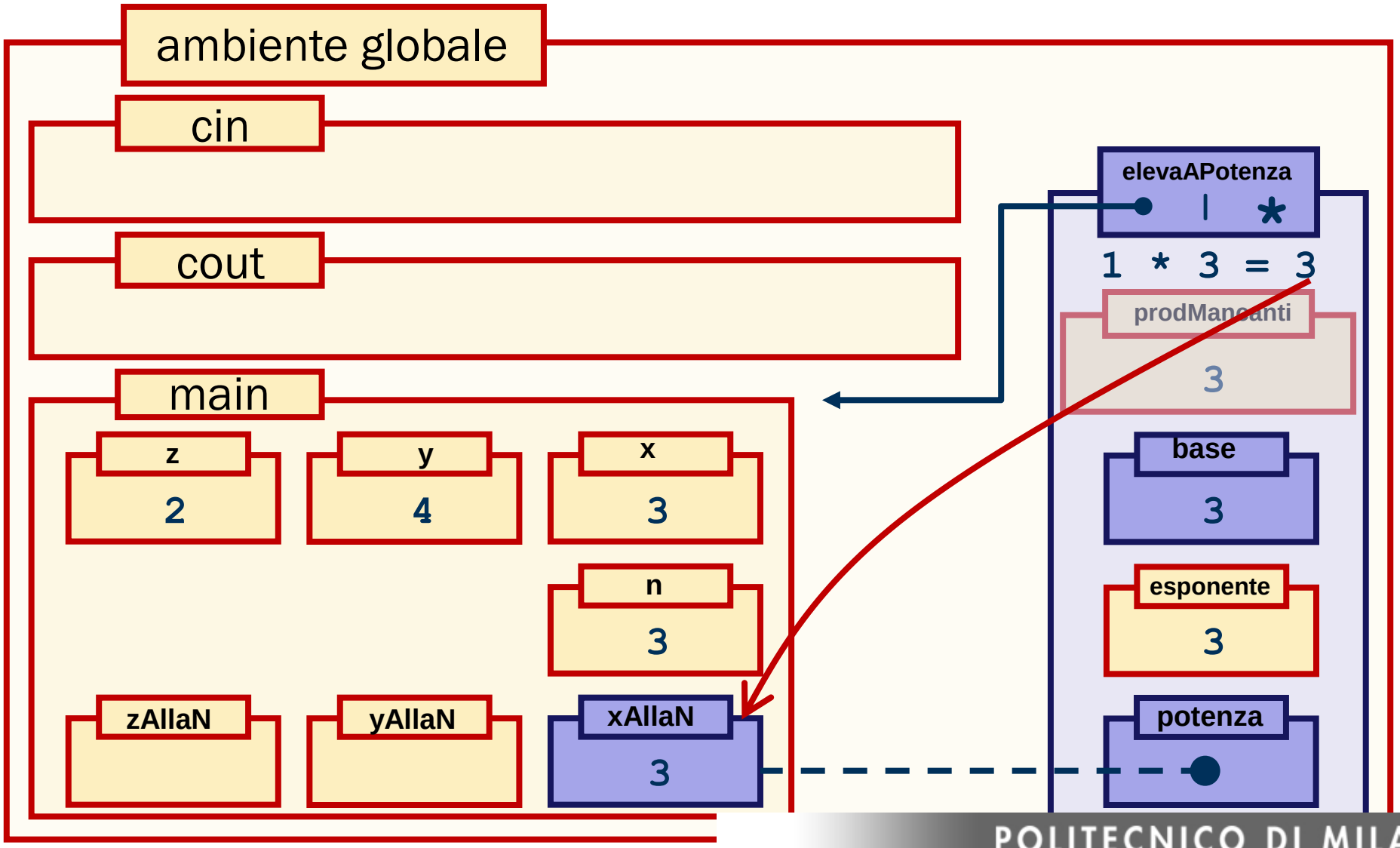
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
  potenza *= base;
}
```

xAllaN *= base

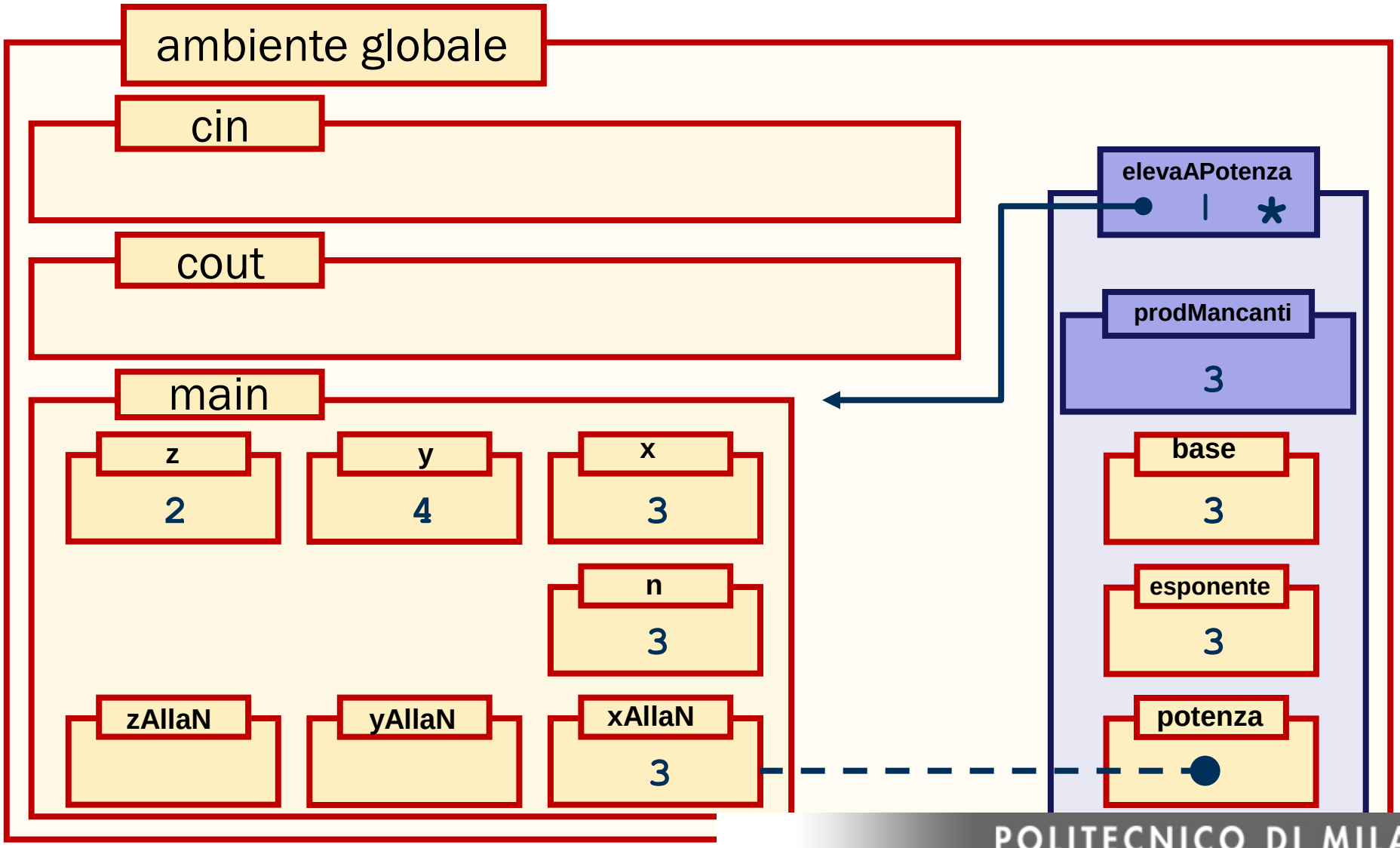


```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
  potenza *= base;
}
```

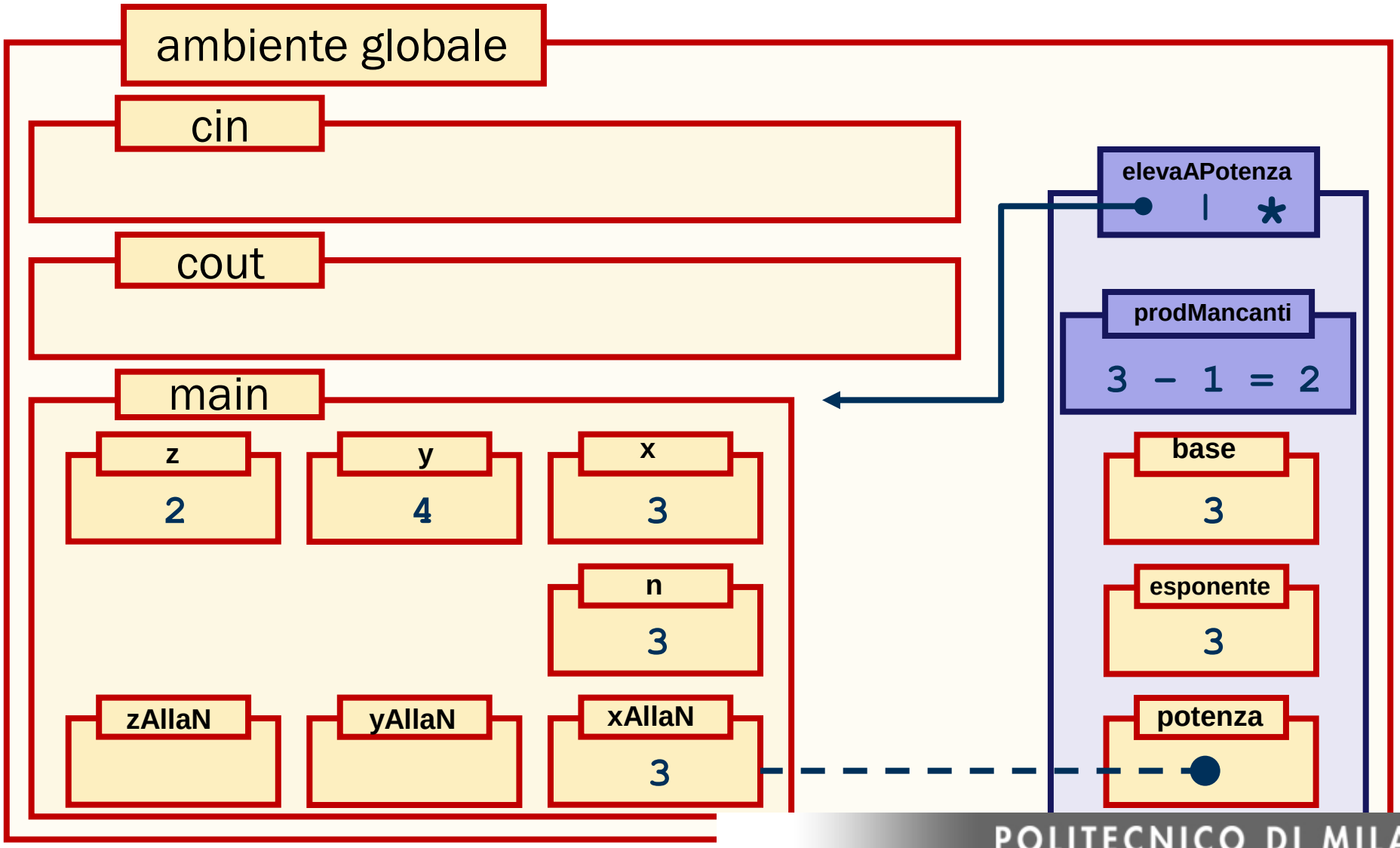
xAllaN *= base



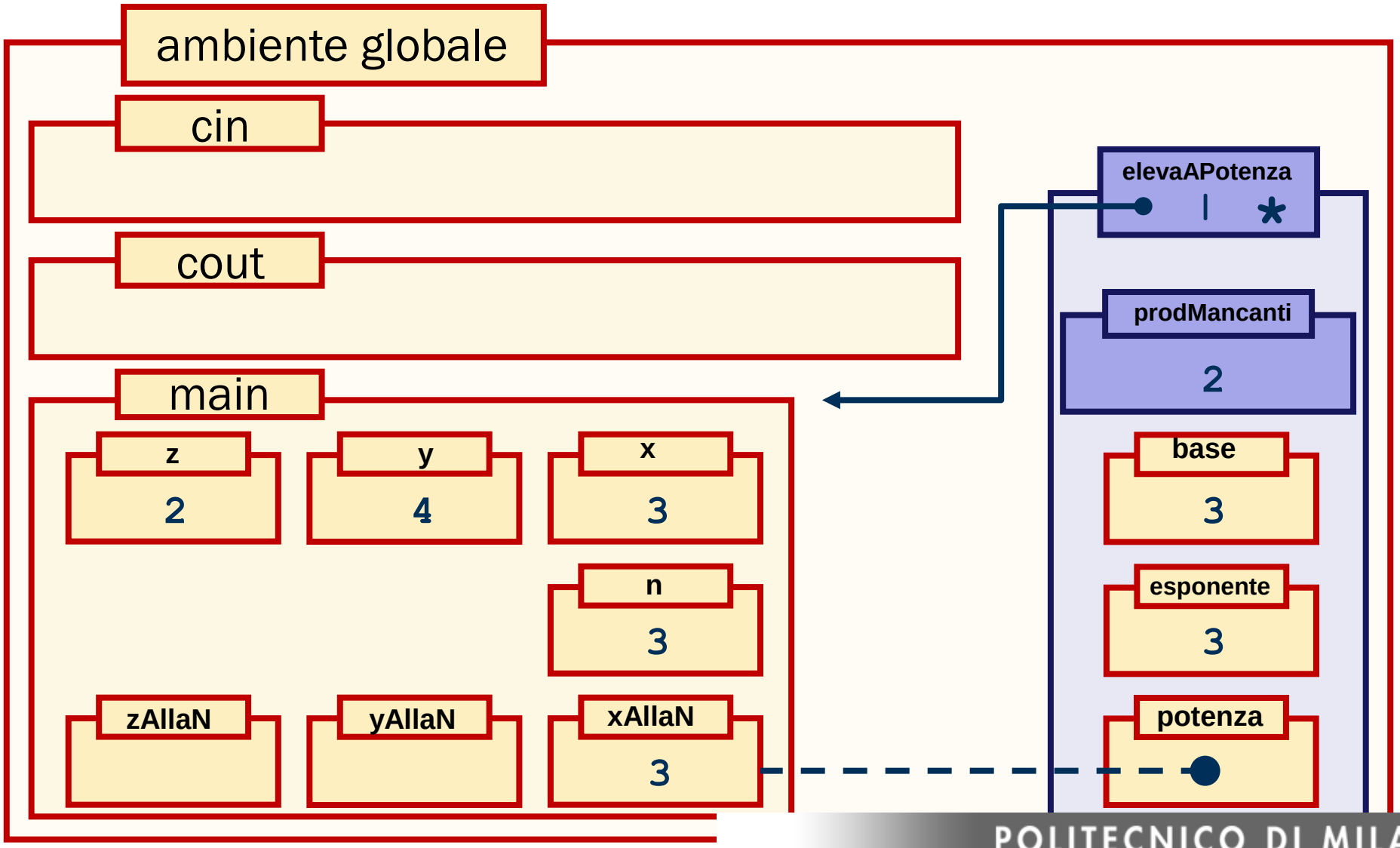
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
}
```



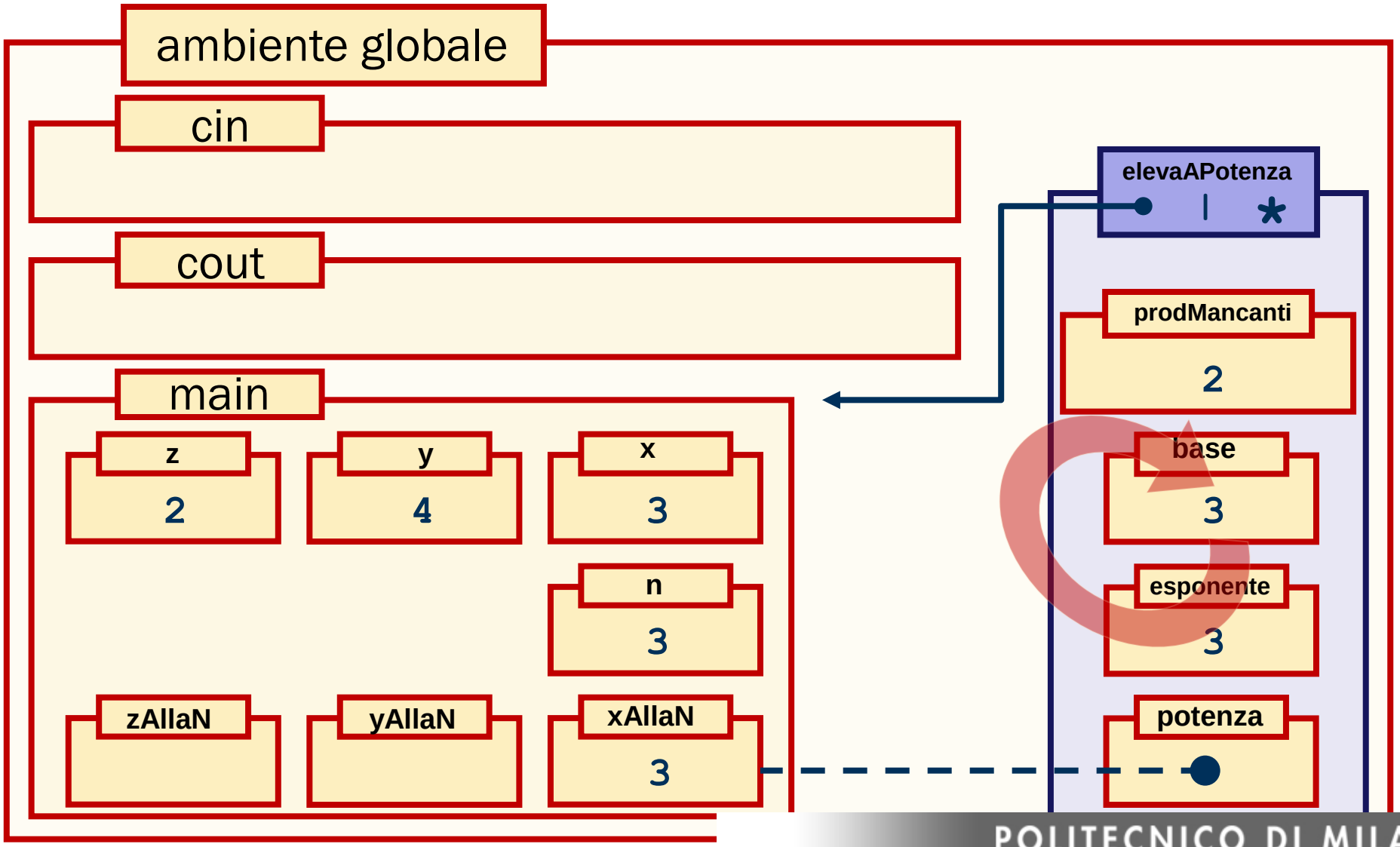
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
}
```



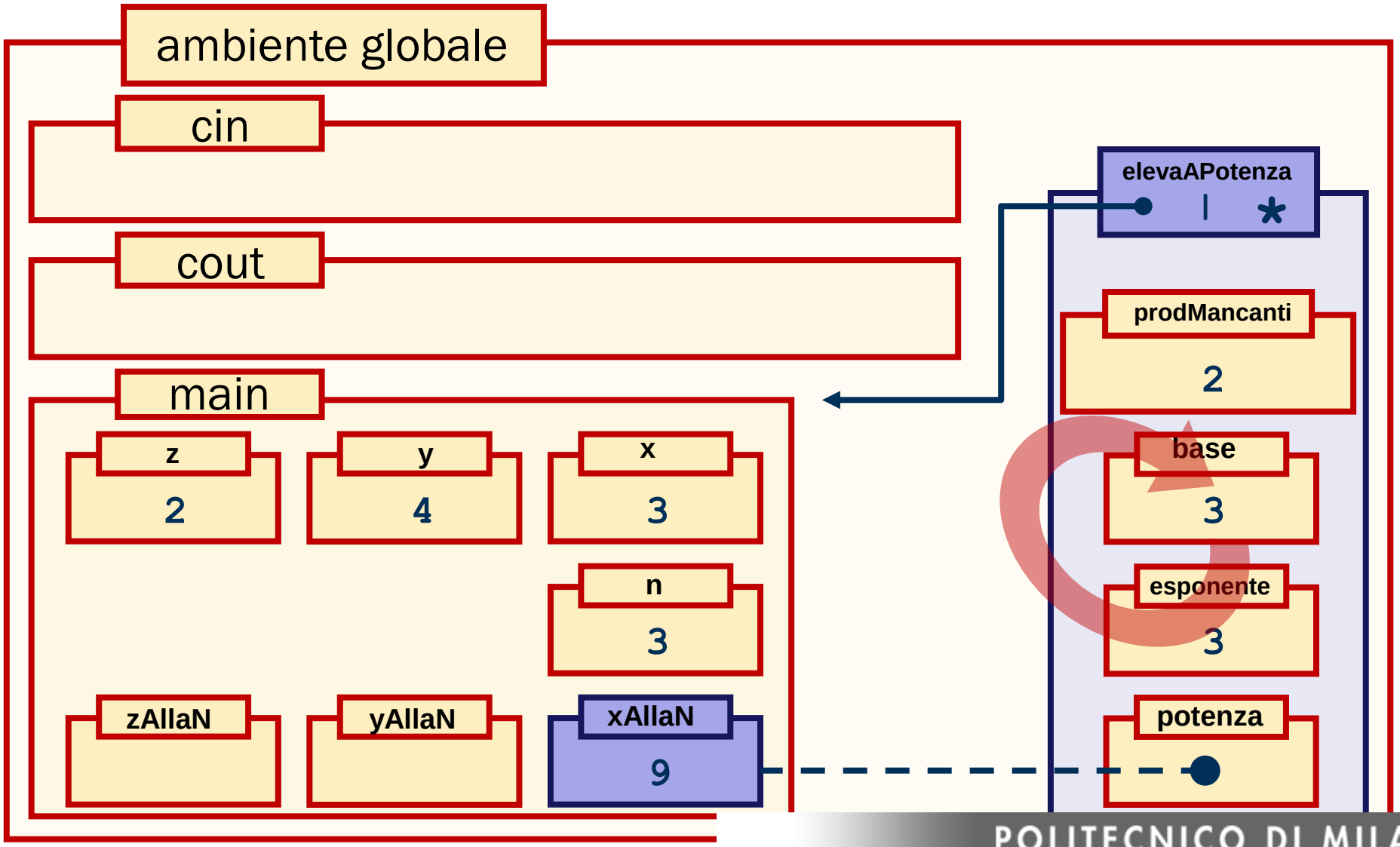
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
}
```



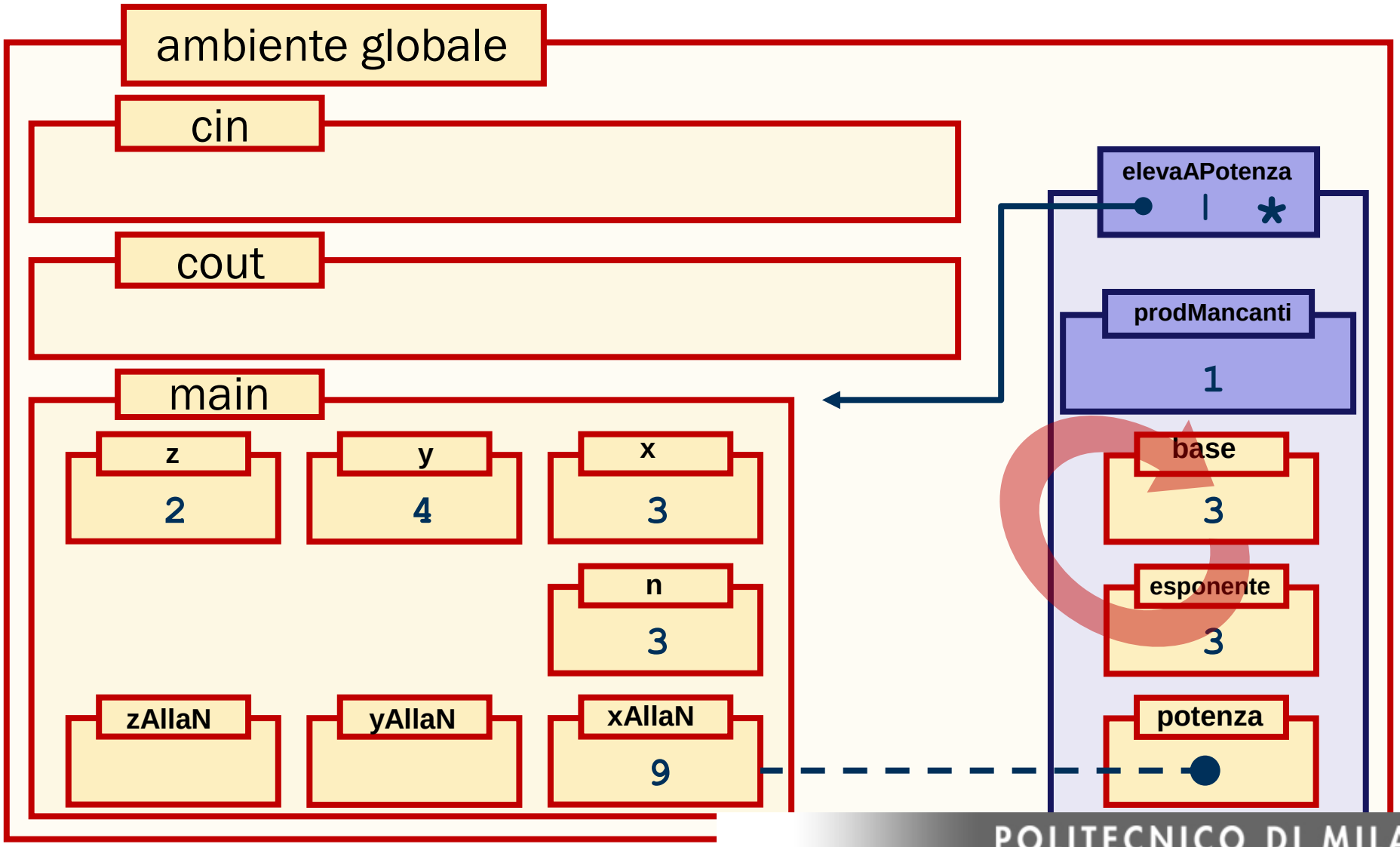
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
    potenza *= base;
}
```



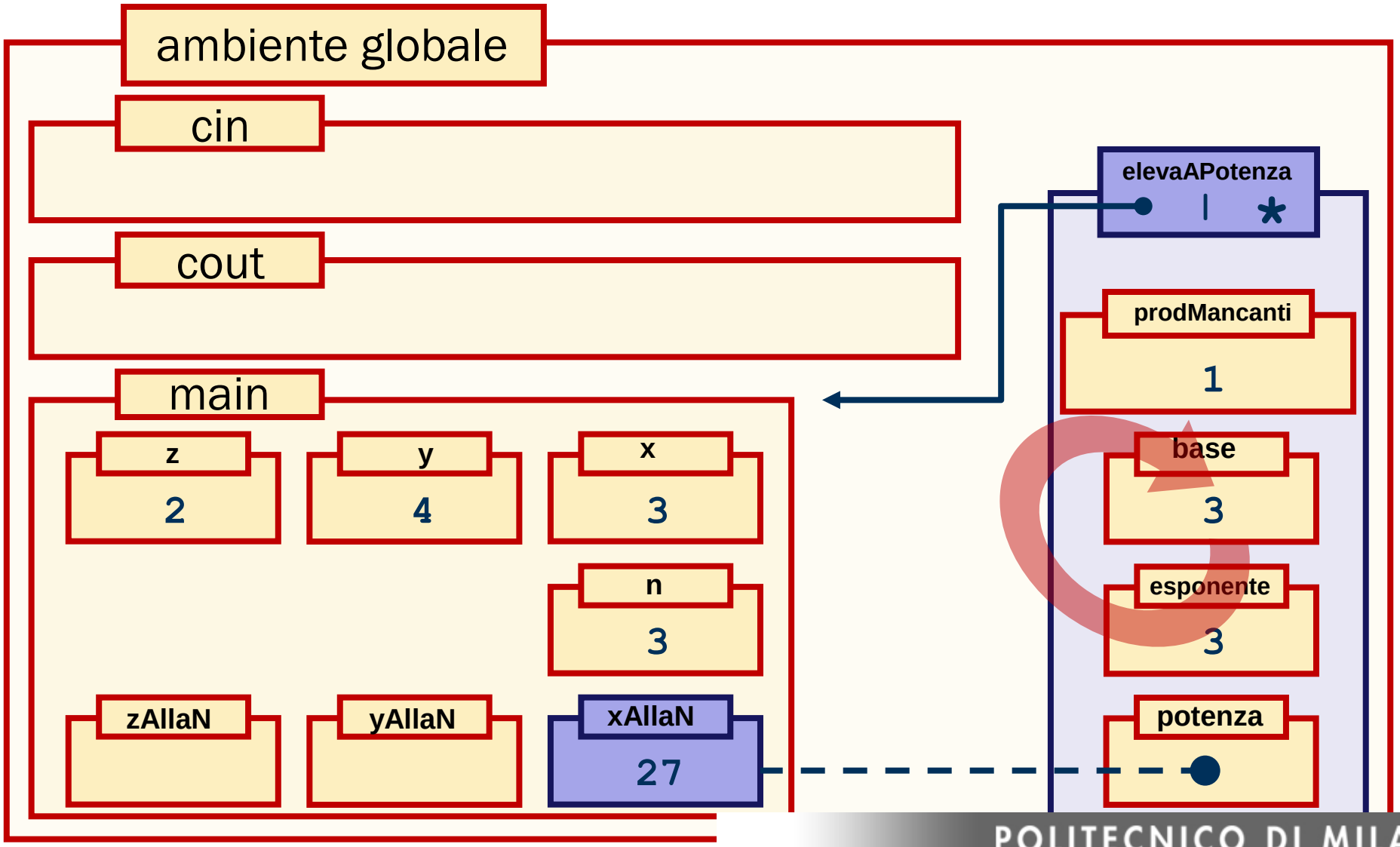
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
    potenza *= base;
}
```



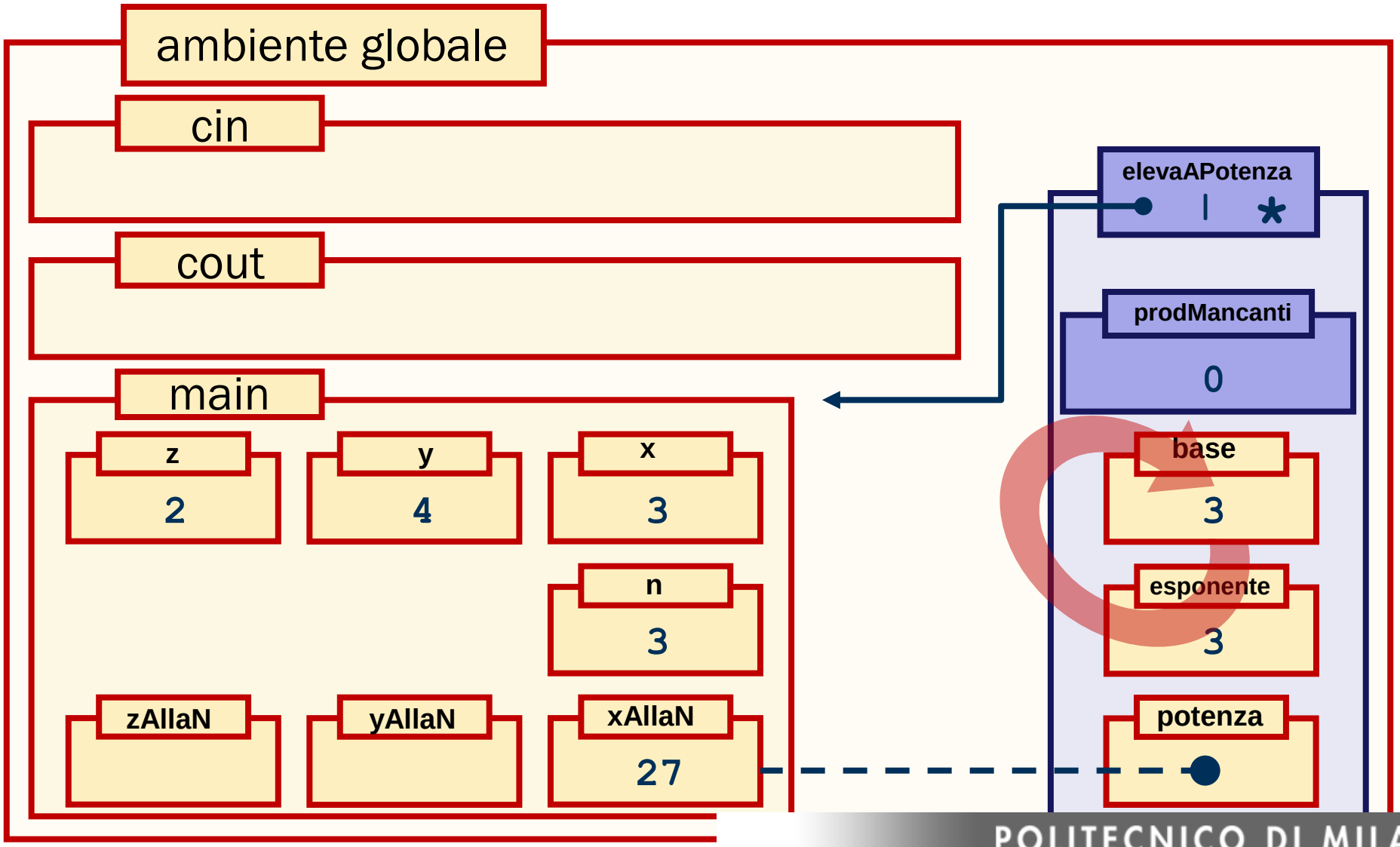
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
    potenza *= base;
}
```



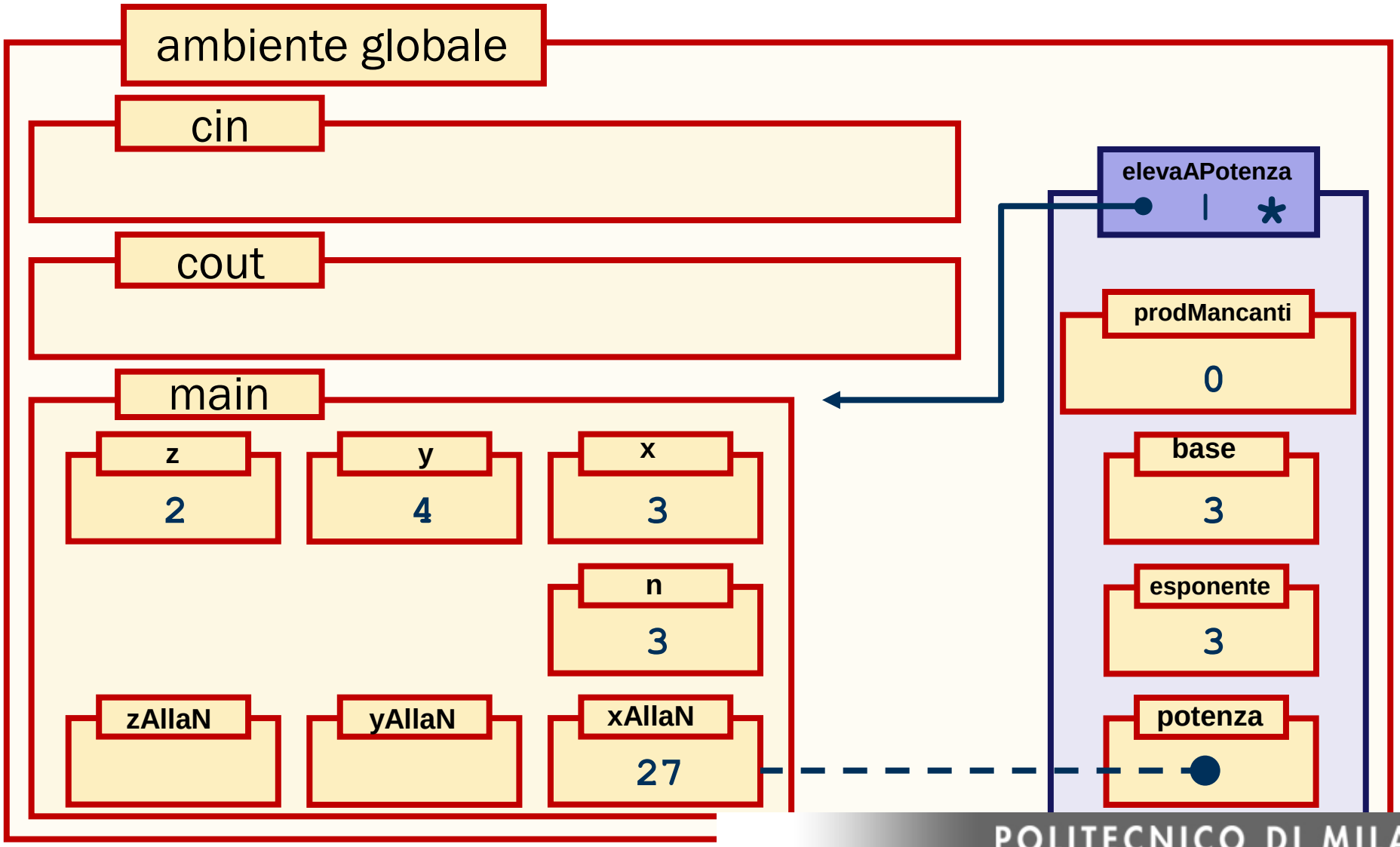
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
    potenza *= base;
}
```



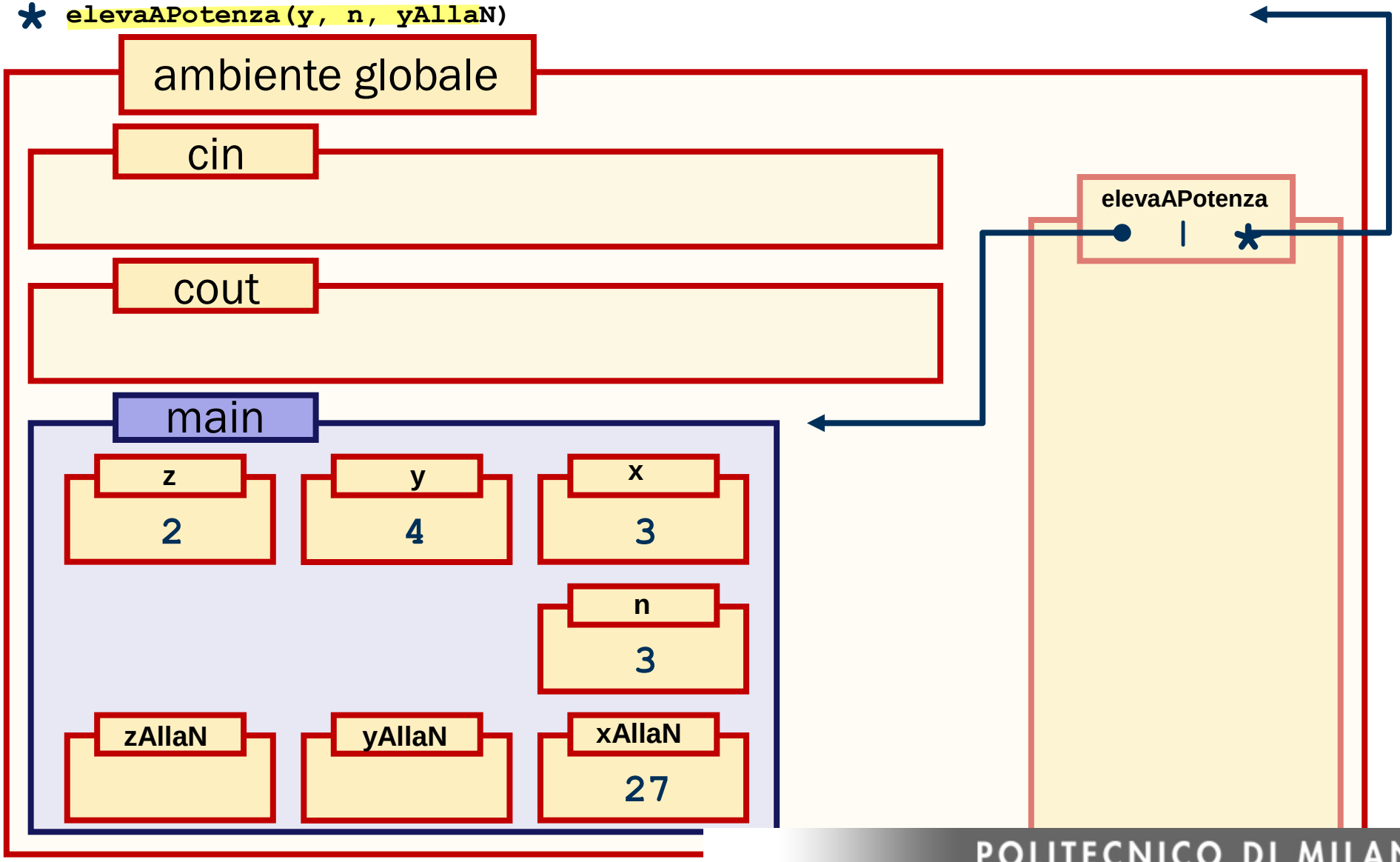
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
    potenza *= base;
}
```



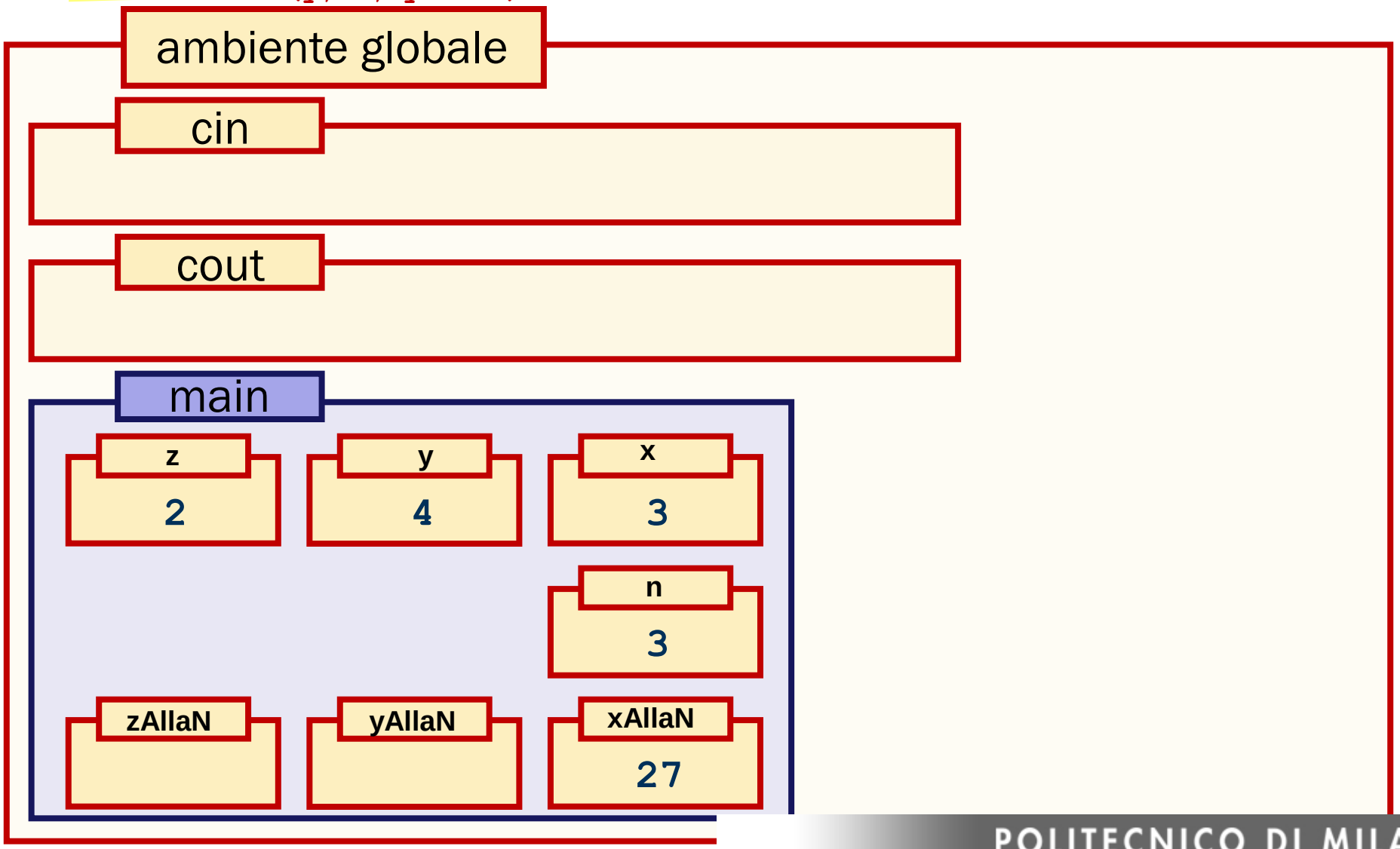
```
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
}
```



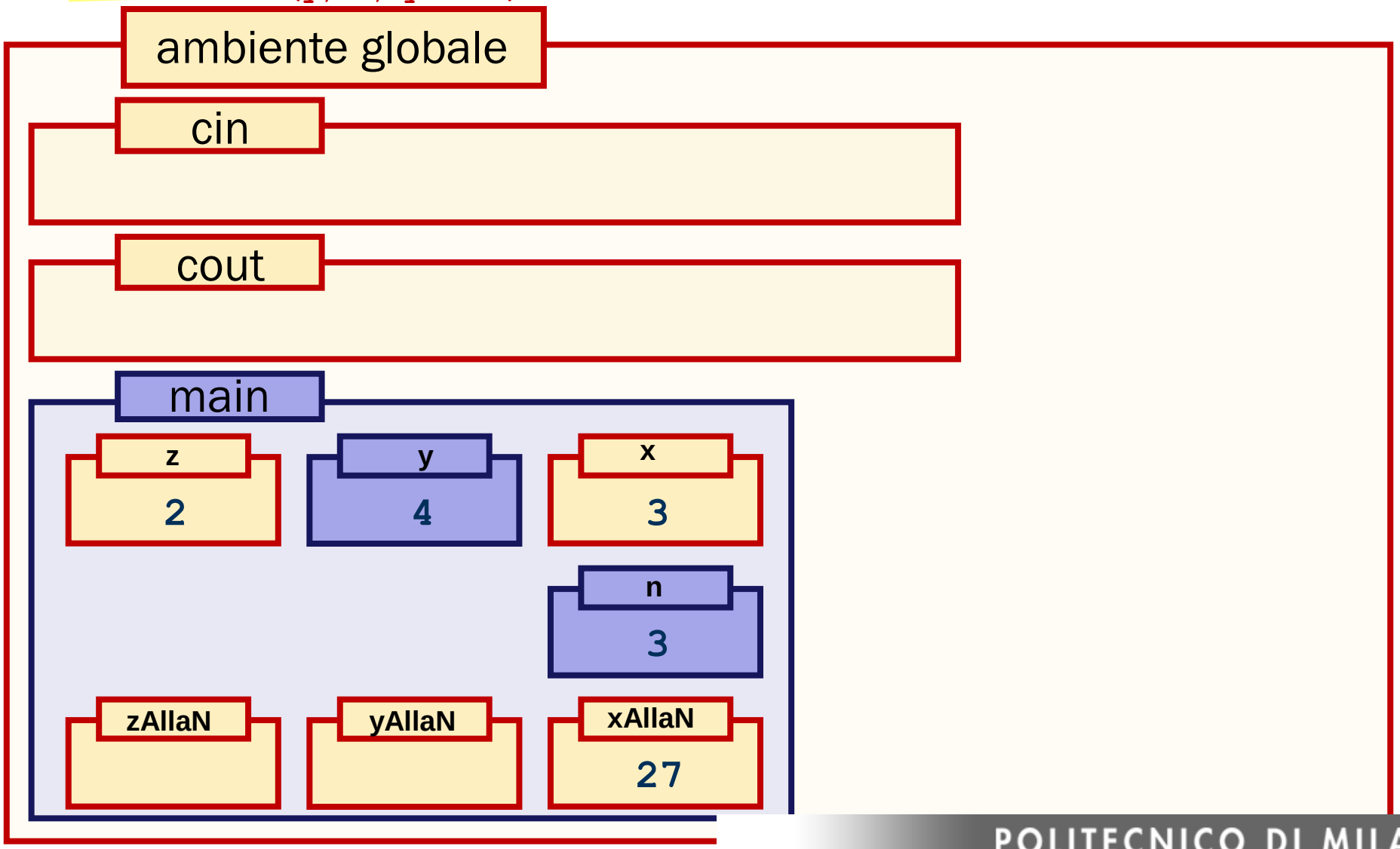
```
elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
elevaAPotenza(y, n, yAllaN)
```



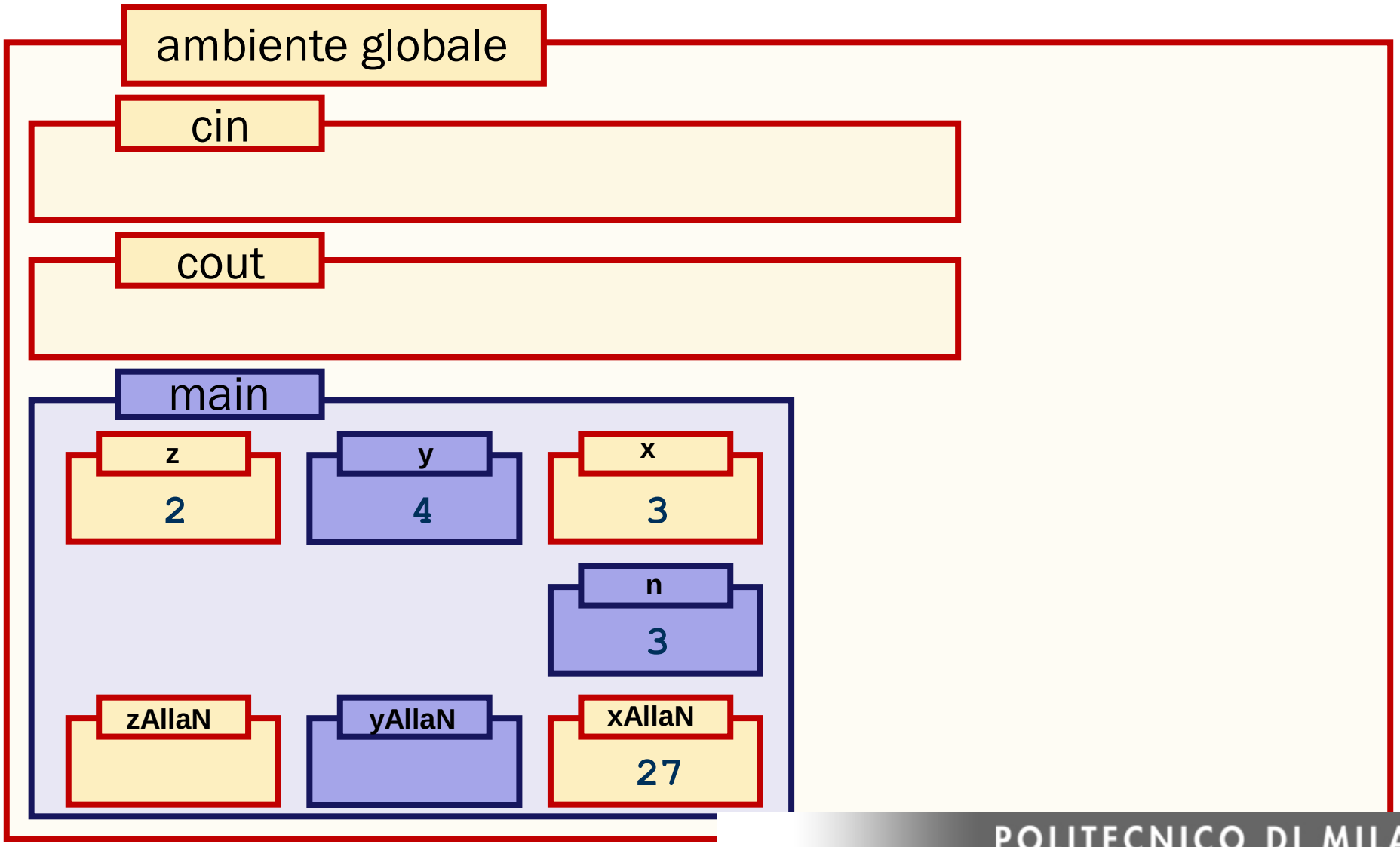
```
elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
```



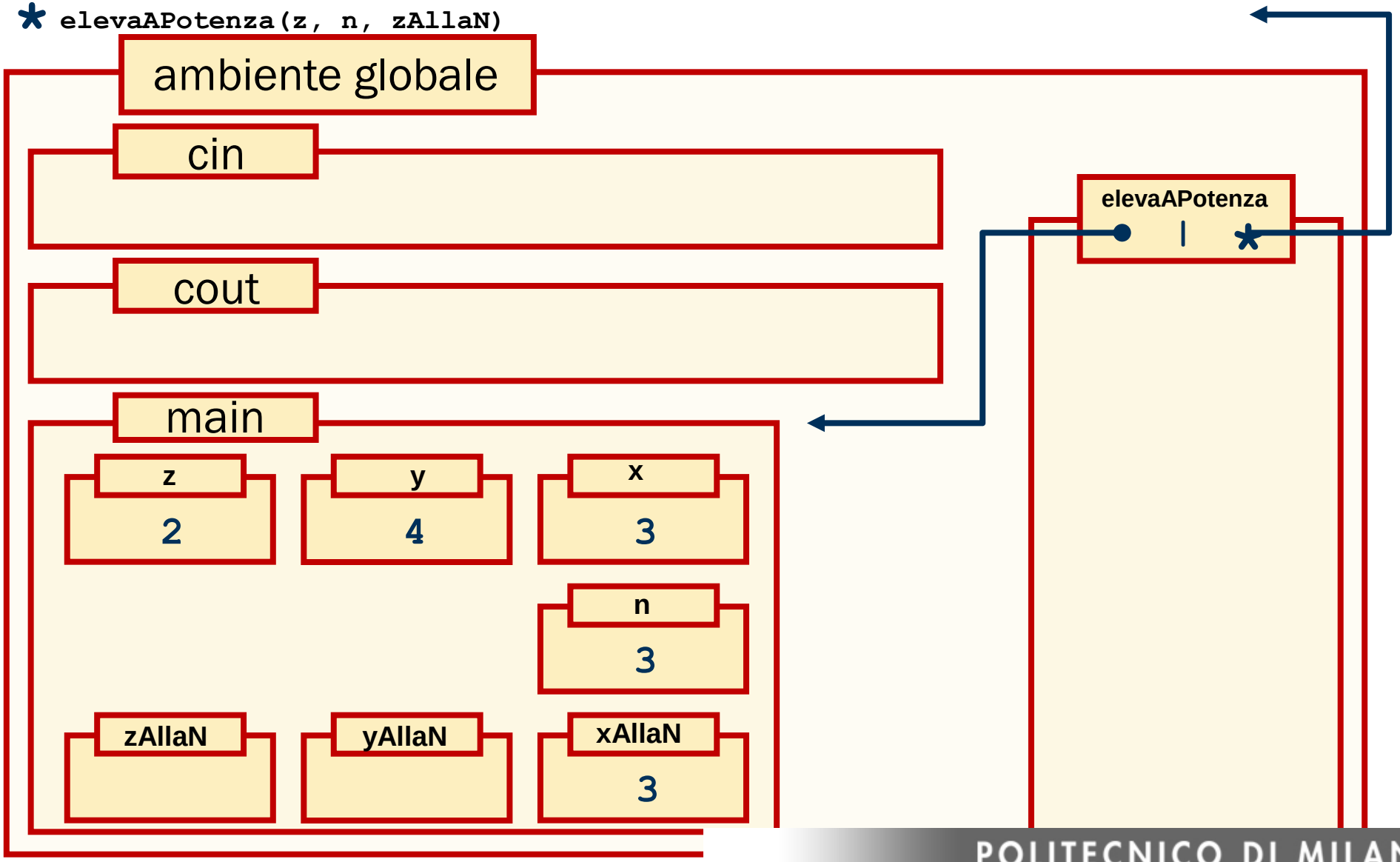
```
elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
```



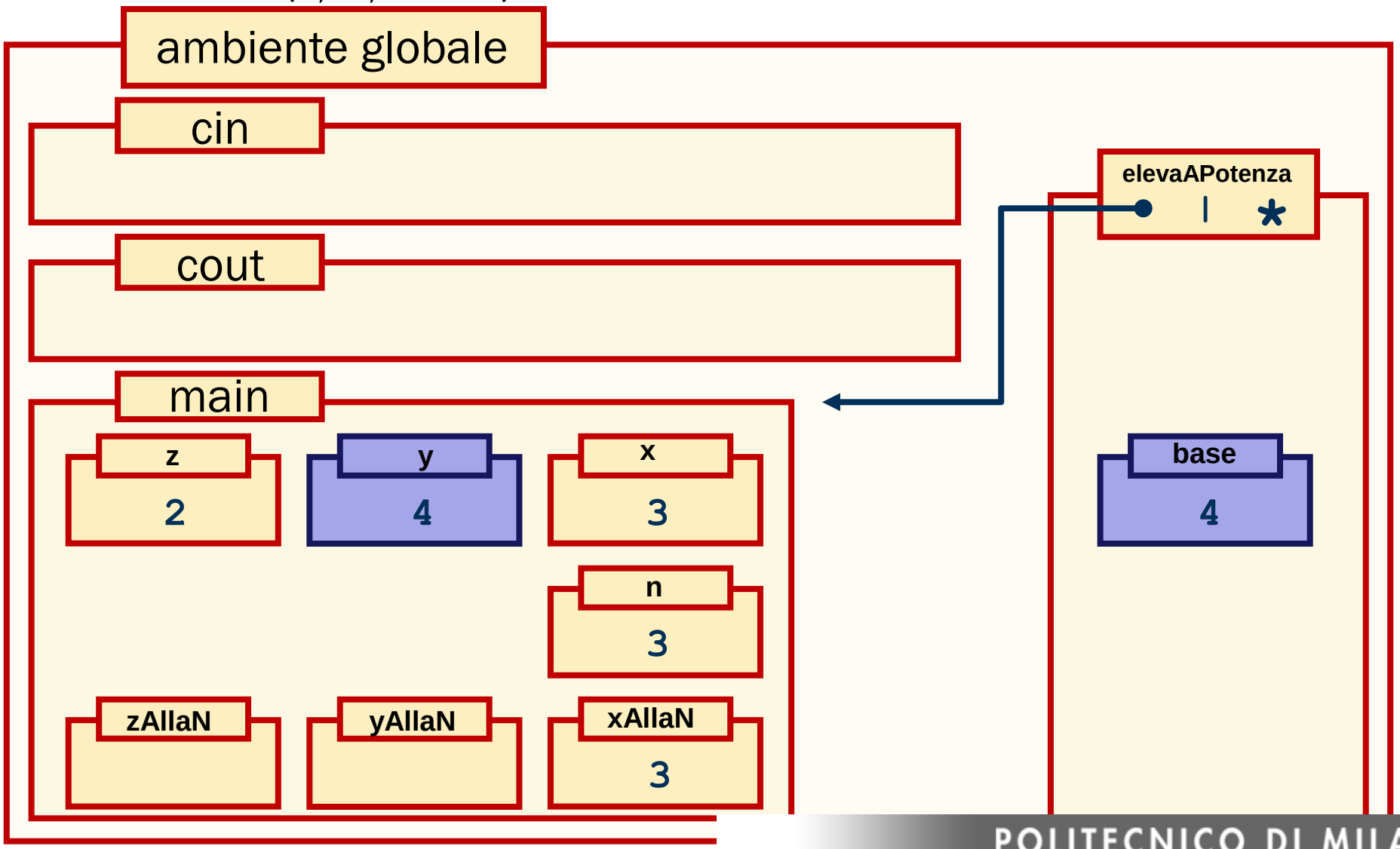
```
elevaAPotenza(x, n, xAllaN);

//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
```



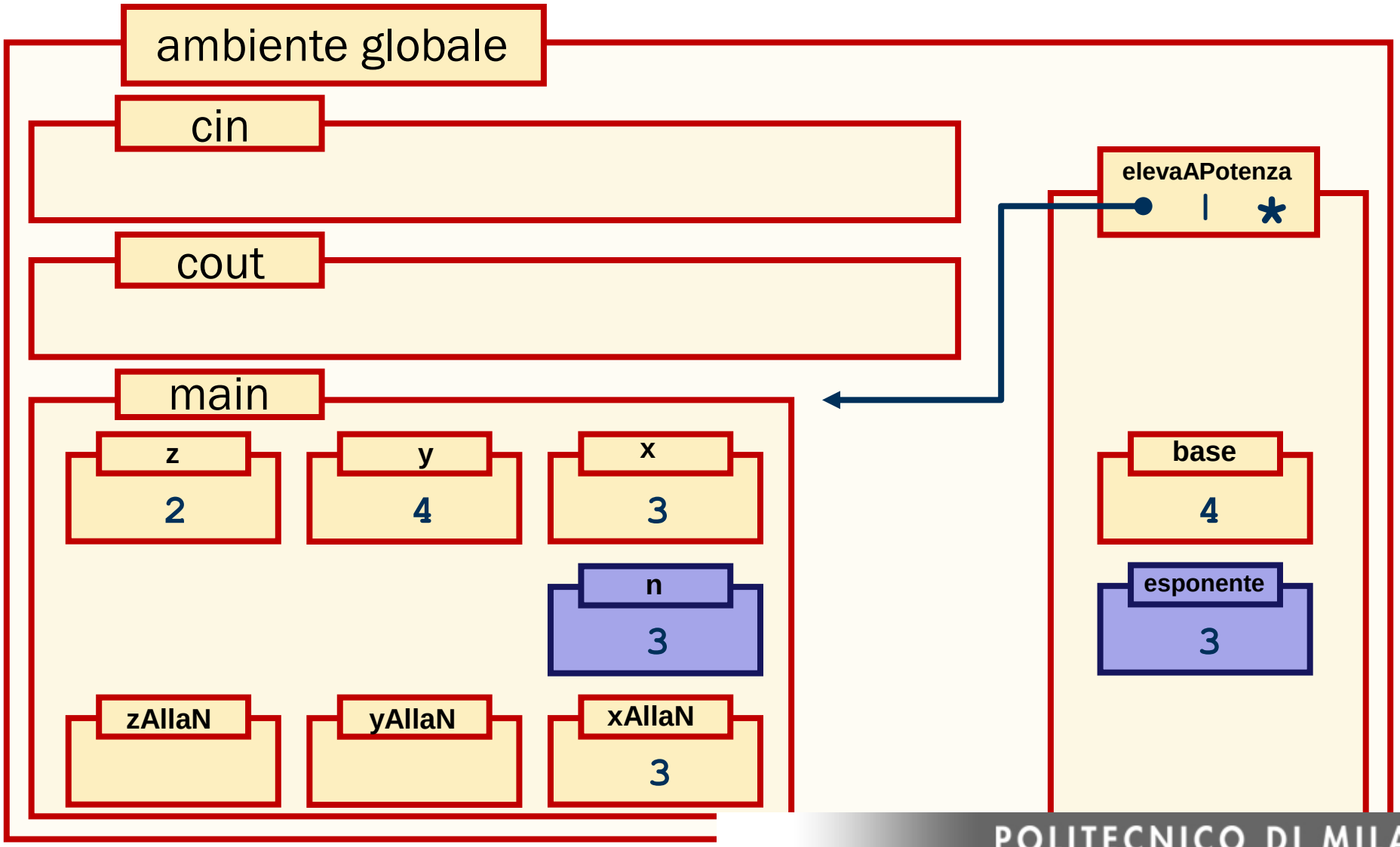
```
//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
//calcola z elevato a n, con risultato in zAllaN
* elevaAPotenza(z, n, zAllaN)
```



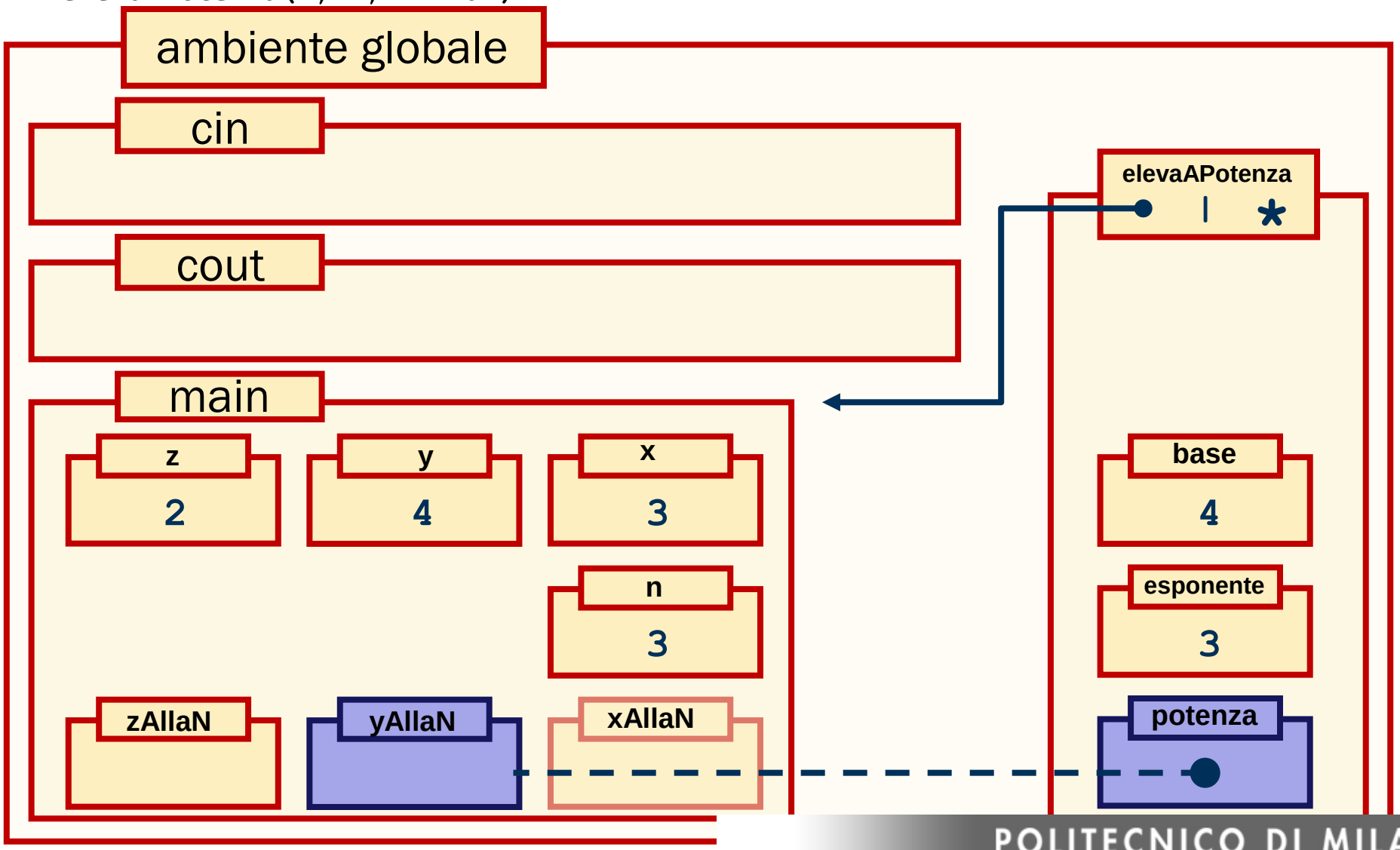
```
//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
//calcola z elevato a n, con risultato in zAllaN
  elevaAPotenza(z, n, zAllaN)
```



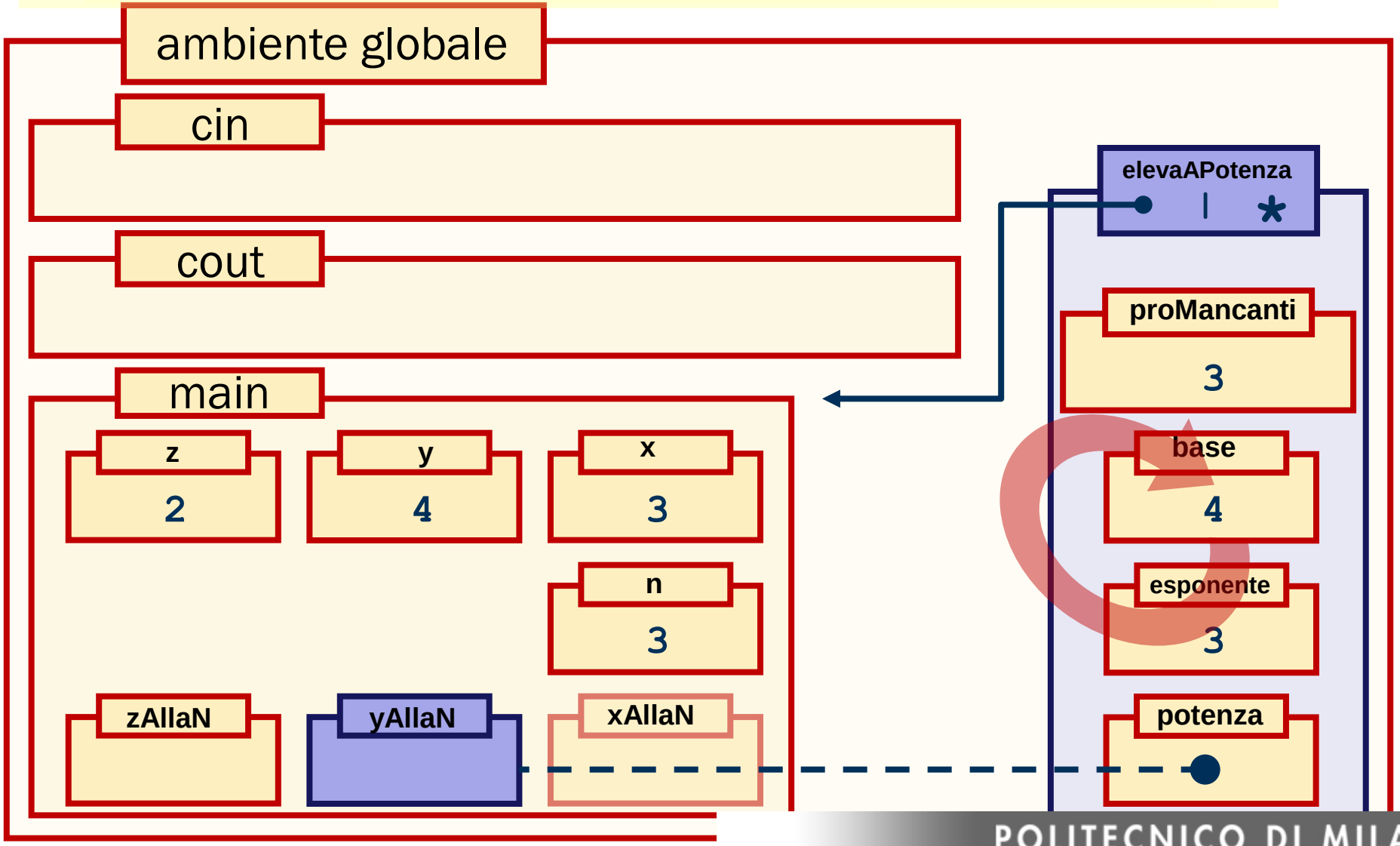
```
//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
//calcola z elevato a n, con risultato in zAllaN
  elevaAPotenza(z, n, zAllaN)
```



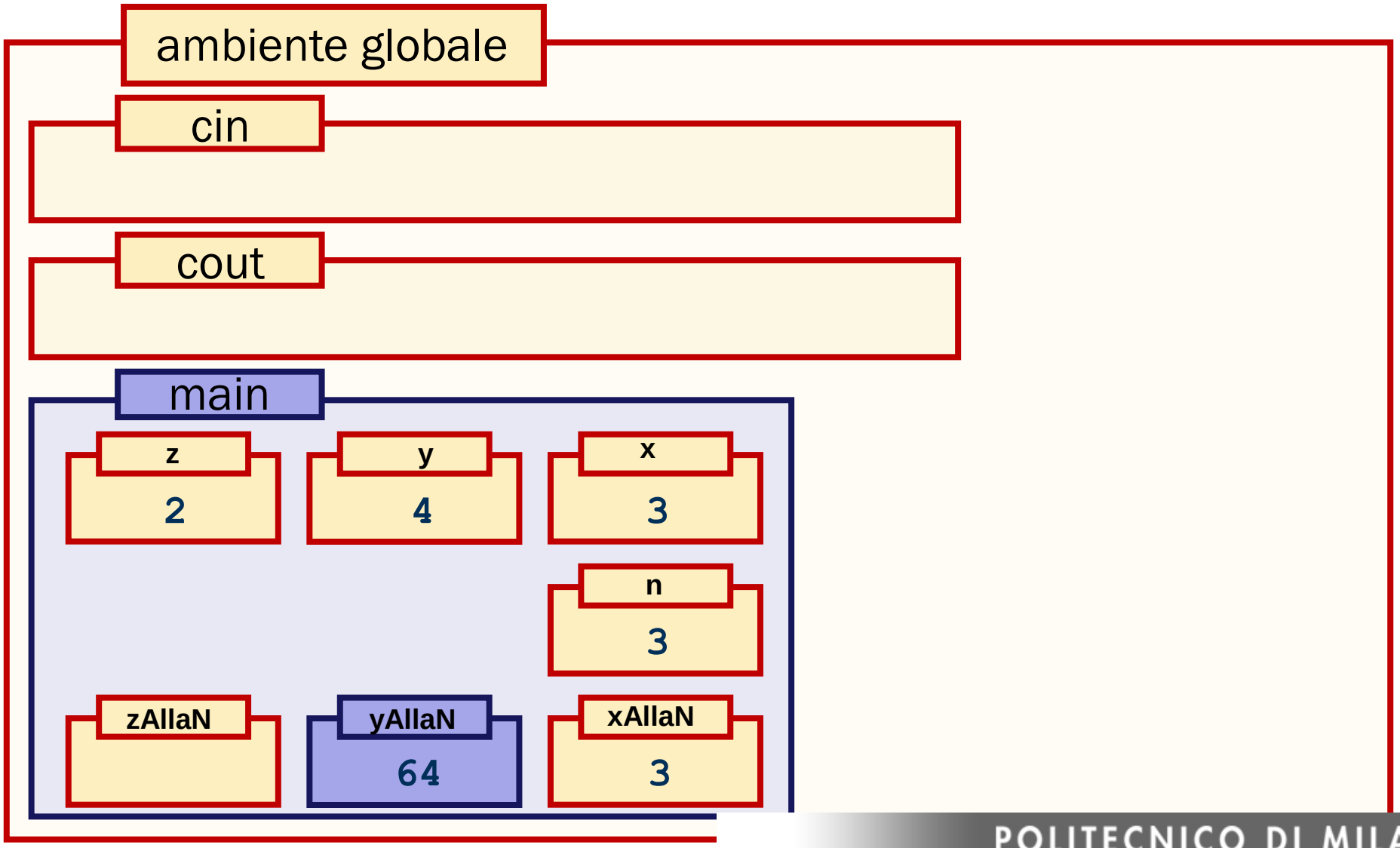
```
//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
//calcola z elevato a n, con risultato in zAllaN
  elevaAPotenza(z, n, zAllaN)
```



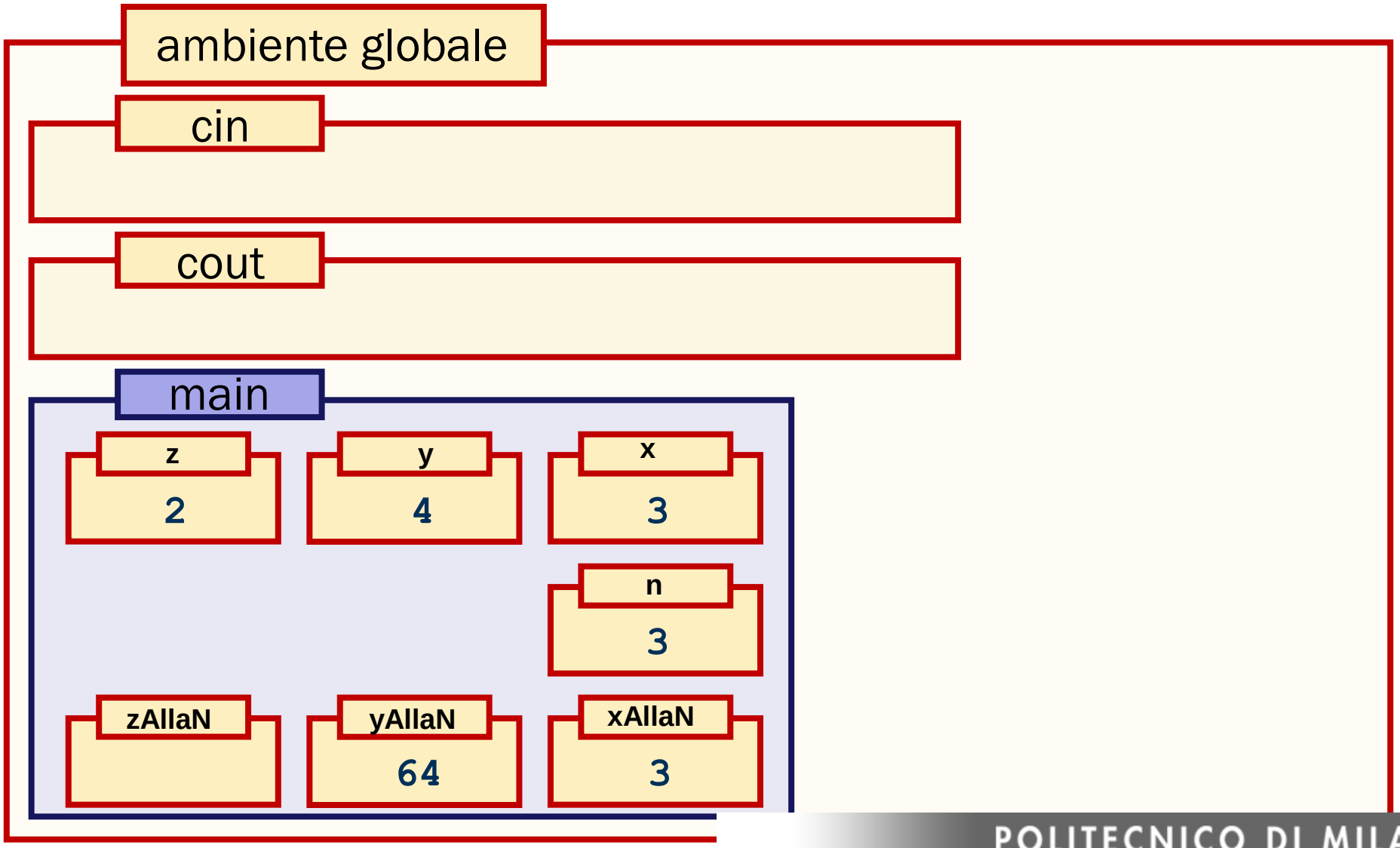
```
int prodMancanti;          //variabile locale ad elevaAPotenza
Potenza = 1;
for (prodmancanti = esponente; prodMancanti > 0; prodMancanti--);
   potenza *= base;
```



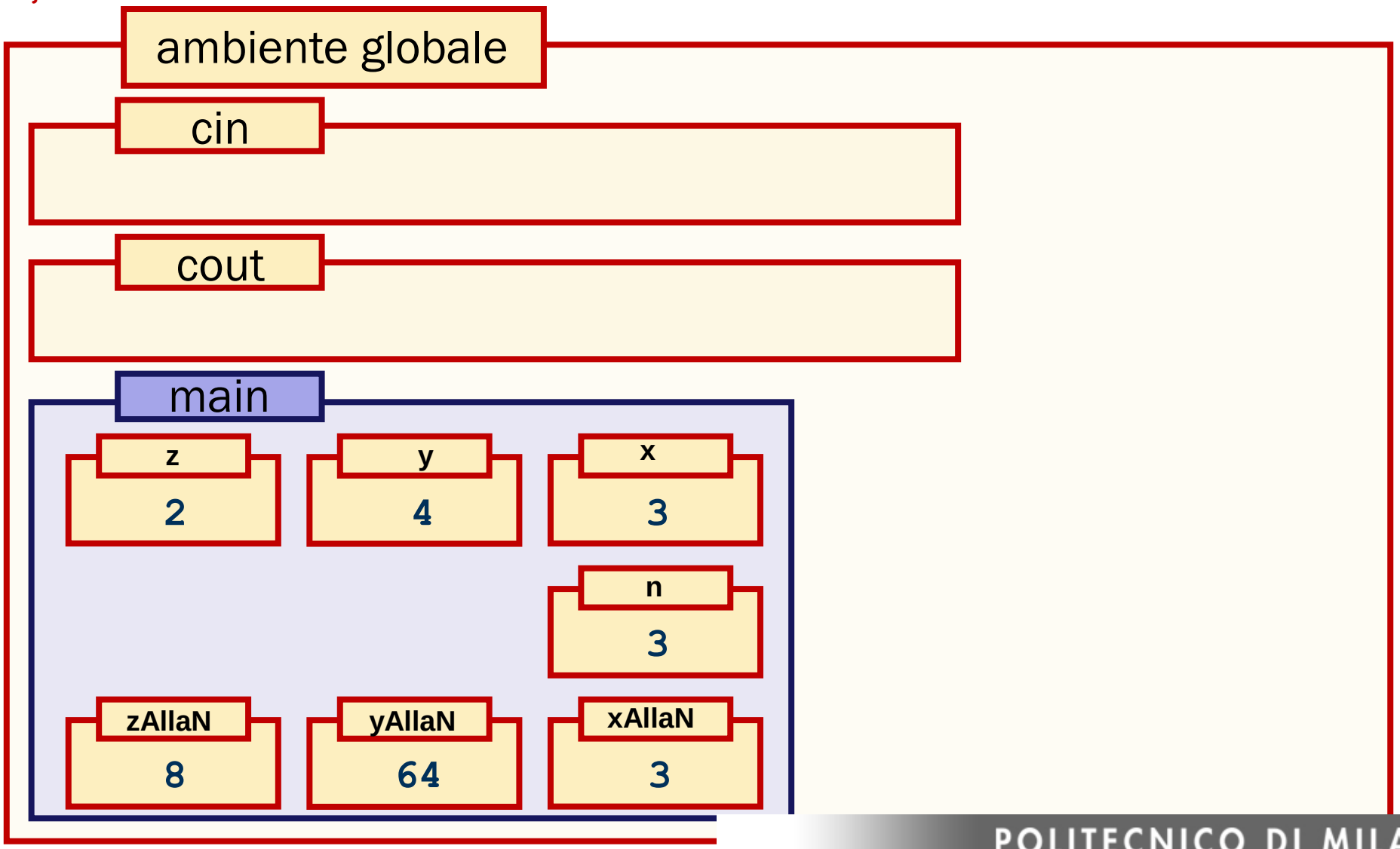
```
//calcola y elevato a n, con risultato in yAllaN
  elevaAPotenza(y, n, yAllaN)
//calcola z elevato a n, con risultato in zAllaN
  elevaAPotenza(z, n, zAllaN)
```



```
//calcola z elevato a n, con risultato in zAllaN
  elevaAPotenza(z, n, zAllaN)

"Verifica se xAllaN + yAllaN è uguale a zAllaN e stampa il risultato"
```



```
elevaAPotenza(z, n, zAllaN)

"Verifica se xAllaN + yAllaN è uguale a zAllaN e stampa il risultato"
}
```



```
elevaAPotenza(z, n, zAllaN)

"Verifica se xAllaN + yAllaN è uguale a zAllaN e stampa il risultato"
}
```