A.A. 08/09

Fondamenti di Programmazione

(canale E-O)

Docente: Prof.ssa Tiziana Calamoneri
calamo@di.uniroma1.it

Esercitatore: Dott. Roberto Petroccia
petroccia@di.uniroma1.it

Pagina del corso:
http://twiki.di.uniroma1.it/twiki/view/Programmazione1/EO/WebHome

---

Esercitazione del 17/12/08

Indice

1. Precisazioni allocazione della memoria.
2. Uso del tool valgrind.
3. Definizione ed uso di liste in C.
4. Esercizi su strutture e liste.

getchar

```
 int c; //char c;
(c = getchar());
 while (c != '\n') {
    vettChar1[i++] = c;
    printf("%c\n", c);
    (c = getchar());
}
```

E.A.M.

Se accedo a V[i-1] allora i > 0 necessariamente

```
if (V[i-1] == k)                    ERRATO
if (V[i-1] == k && i > 0)           ERRATO
if (i > 0  && V[i-1] == k)          CORRETTO
```

Se accedo a V[i+1] allora i > n-1 necessariamente

```
if (V[i+1] == k)                    ERRATO
if (V[i+1] == k && i < N-1)         ERRATO
if (i < N-1  && V[i+1] == k)        CORRETTO
```

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Allocazione della memoria.

STATICA

DINAMICA

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Allocazione della memoria.

STATICA

Vettori unidimensionali o multidimensionali

(Devo conoscere le dimensioni quando vado ad allocare la memoria)

```
int V[10];

int M[5][6];
```

DINAMICA

Allocazione della memoria.

STATICA

Vettori unidimensionali o multidimensionali

(Devo conoscere le dimensioni quando vado ad allocare la memoria)

int V[10];

int M[5][6];

Alloca la memoria in maniera sequenziale

DINAMICA

Allocazione della memoria.

STATICA

Vettori unidimensionali o multidimensionali

(Devo conoscere le dimensioni quando vado ad allocare la memoria)

int V[10];

int M[5][6];

Alloca la memoria in maniera sequenziale

DINAMICA

Tramite l'utilizzo di istruzioni malloc e calloc

(Posso allocarla senza l'uso di valori costanti)

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Allocazione della memoria.

STATICA

Vettori unidimensionali o multidimensionali

(Devo conoscere le dimensioni quando vado ad allocare la memoria)

```
int V[10];
int M[5][6];
```

Alloca la memoria in maniera sequenziale

DINAMICA

Tramite l'utilizzo di istruzioni malloc e calloc

(Posso allocarla senza l'uso di valori costanti)

```
int *V = (int*) malloc (sizeof(int) * 10);
int** M = (int **) malloc (sizeof(int) * num_riga);
for (i = 0; i < num_riga; i++) {
    M[i] = (int*) malloc(sizeof(int) * num_colonna);
}
```

Allocazione della memoria.

STATICA

Vettori unidimensionali o multidimensionali

(Devo conoscere le dimensioni quando vado ad allocare la memoria)

```
int V[10];

int M[5][6];
```

Alloca la memoria in maniera sequenziale

DINAMICA

Tramite l'utilizzo di istruzioni malloc e calloc

(Posso allocarla senza l'uso di valori costanti)

```
int *V = (int*) malloc (sizeof(int) * 10);
int** M = (int **) malloc (sizeof(int) * num_riga);
for (i = 0; i < num_riga; i++) {
    M[i] = (int*) malloc(sizeof(int) * num_colonna);
}
```

Alloca la memoria in maniera sequenziale

Allocazione della memoria.

STATICA

int M[5][5];

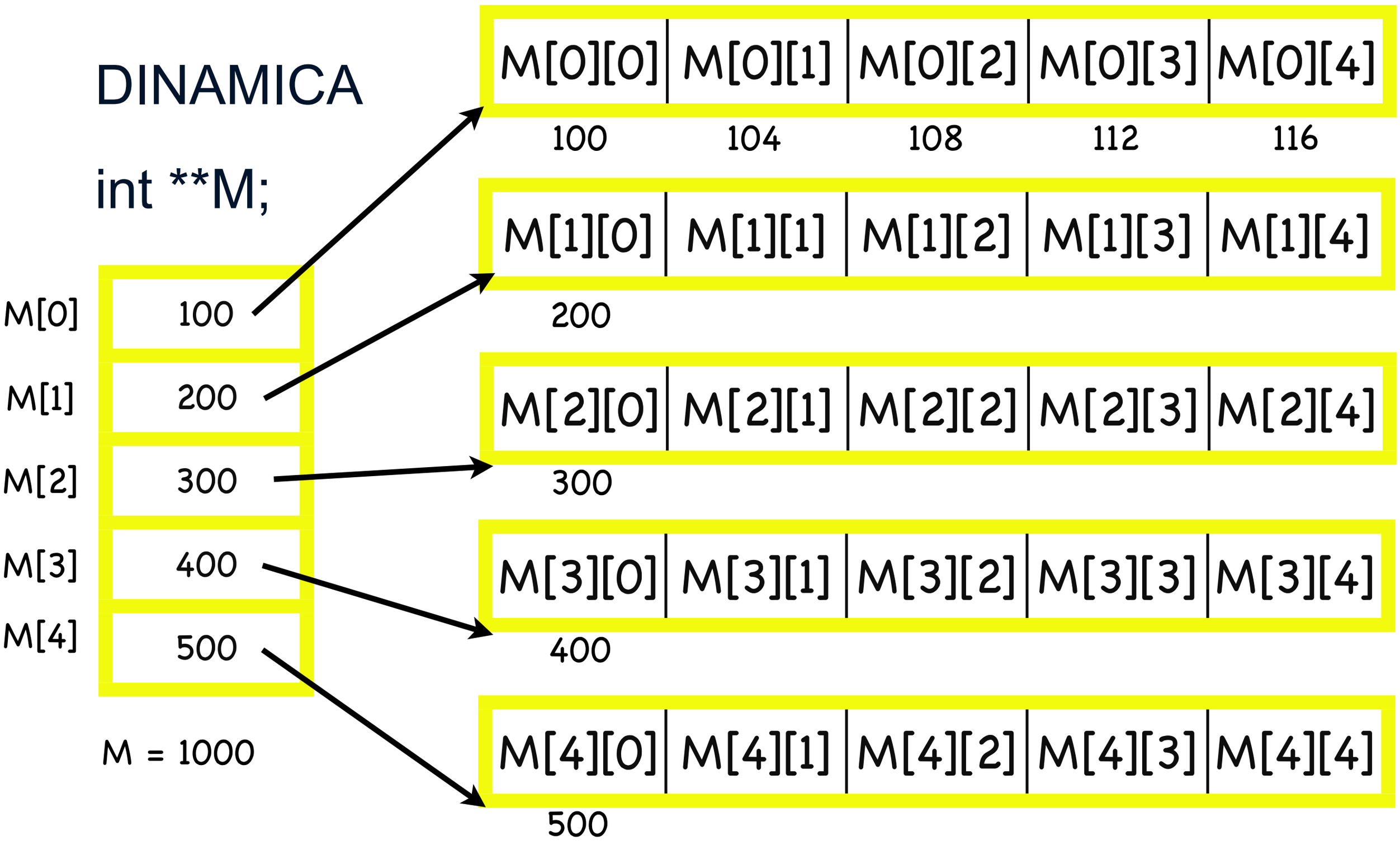
M[0][0]	M[0][1]	M[0][2]	M[0][3]	M[0][4]
M[1][0]	M[1][1]	M[1][2]	M[1][3]	M[1][4]
M[2][0]	M[2][1]	M[2][2]	M[2][3]	M[2][4]
M[3][0]	M[3][1]	M[3][2]	M[3][3]	M[3][4]
M[4][0]	M[4][1]	M[4][2]	M[4][3]	M[4][4]

M[0][0]	M[0][1]	M[0][2]	M[0][3]	M[0][4]	M[1][0]	M[1][1]	M[1][2]	M[1][3]	M[1][4]

Allocazione della memoria.

DINAMICA

int **M;



Allocazione della memoria.

A cosa serve l'allocazione dinamica se posso scrivere:

```
int n;
scanf("%d\n", &n);
int V[n];
```

```
void matrix (int r, int c, int M[][c]) {
    ..............
}
```

oppure

```
int n = contaQualcosa(...);
int V[n];
```

Allocazione della memoria.

A cosa serve l'allocazione dinamica se posso scrivere:

```
int n;
scanf("%d\n", &n);
int V[n];
```

```
void matrix (int r, int c, int M[][c]) {
    ..............
}
```

oppure

```
int n = contaQualcosa(...);
int V[n];
```

QUESTE ISTRUZIONI NON RISPETTANO LO STANDARD ANCI C 89

Allocazione della memoria.

A cosa serve l'allocazione dinamica se posso scrivere:

```
int n;
scanf("%d\n", &n);
int V[n];
```

```
void matrix (int r, int c, int M[][c]) {
    ..............
}
```

oppure

```
int n = contaQualcosa(...);
int V[n];
```

QUESTE ISTRUZIONI NON RISPETTANO LO STANDARD ANCI C 89

Compilando con gcc -pedantic ottenete un warning

La memoria viene allocata sullo stack non sulla RAM oltre un certo valore avrete un errore

Allocazione della memoria.

STATICA

Va usato con valori costanti, quando scrivete il programma conoscete già la massima memoria da utilizzare

```
int V[10];

int M[5][6];
```

DINAMICA

Quando scrivete il codice non sapete quanta memoria occorre

```
int *V = (int*) malloc (sizeof(int) * 10);
int** M = (int **) malloc (sizeof(int) * num_riga);
for (i = 0; i < num_riga; i++) {
    M[i] = (int*) malloc(sizeof(int) * num_colonna);
}
```

Allocazione della memoria.

STATICA

Va usato con valori costanti, quando scrivete il programma conoscete già la massima memoria da utilizzare

```
int V[10];

int M[5][6];
```

Con l'allocazione dinamica dovete preoccuparvi voi di liberare la memoria con l'istruzione free

```
free(V);
```

DINAMICA

Quando scrivete il codice non sapete quanta memoria occorre

```
int *V = (int*) malloc (sizeof(int) * 10);
int** M = (int **) malloc (sizeof(int) * num_riga);
for (i = 0; i < num_riga; i++) {
    M[i] = (int*) malloc(sizeof(int) * num_colonna);
}
```

Valgrind

Errori comuni:

1) Uso (confronto/operazioni) di variabili senza averle inizializzate

2) Accesso a zone di memoria errate (vettore V di n elementi: V[-1], V[n])

Valgrind

Errori comuni:

1) Uso (confronto/operazioni) di variabili senza averle inizializzate

2) Accesso a zone di memoria errate (vettore V di n elementi: V[-1], V[n])

Per controllare questo tipo di errori potete usare il programma valgrind (va scaricato - potete usare il gestore di pacchetti della vostra distribuzione Linux)

Valgrind

Errori comuni:

1) Uso (confronto/operazioni) di variabili senza averle inizializzate

2) Accesso a zone di memoria errate (vettore V di n elementi: V[-1], V[n])

Per controllare questo tipo di errori potete usare il programma valgrind (va scaricato - potete usare il gestore di pacchetti della vostra distribuzione Linux)

```
gcc programma.c -o programma.out

valgrind -v ./programma.out
```

Valgrind

Errori comuni:

1) Uso (confronto/operazioni) di variabili senza averle inizializzate

2) Accesso a zone di memoria errate (vettore V di n elementi: V[-1], V[n])

Per controllare questo tipo di errori potete usare il programma valgrind (va scaricato - potete usare il gestore di pacchetti della vostra distribuzione Linux)

gcc programma.c -o programma.out

valgrind -v ./programma.out

Viene eseguito il programma e vengono controllati gli accessi in memoria e l'uso di variabili non inizializzate (attendibile al 98%)

DOMANDE???

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

LISTE

Metodi di organizzazione di dati che si ottengo tramite l'utilizzo di strutture ricorsive.

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

LISTE

Metodi di organizzazione di dati che si ottengo tramite l'utilizzo di strutture ricorsive.

Una struttura ricorsiva contiene un membro di tipo puntatore, che fa riferimento ad una struttura dello stesso tipo di quella in cui è contenuto.

```
struct coppia {
        int val1;
        int val2;
        struct coppia * next;
};
```

LISTE

Metodi di organizzazione di dati che si ottengo tramite l'utilizzo di strutture ricorsive.

Una struttura ricorsiva contiene un membro di tipo puntatore, che fa riferimento ad una struttura dello stesso tipo di quella in cui è contenuto.

```
struct coppia {
        int val1;
        int val2;
        struct coppia * next;
};
```

Il campo next viene usato per collegare diversi elementi di tipo coppia tra di loro.

LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

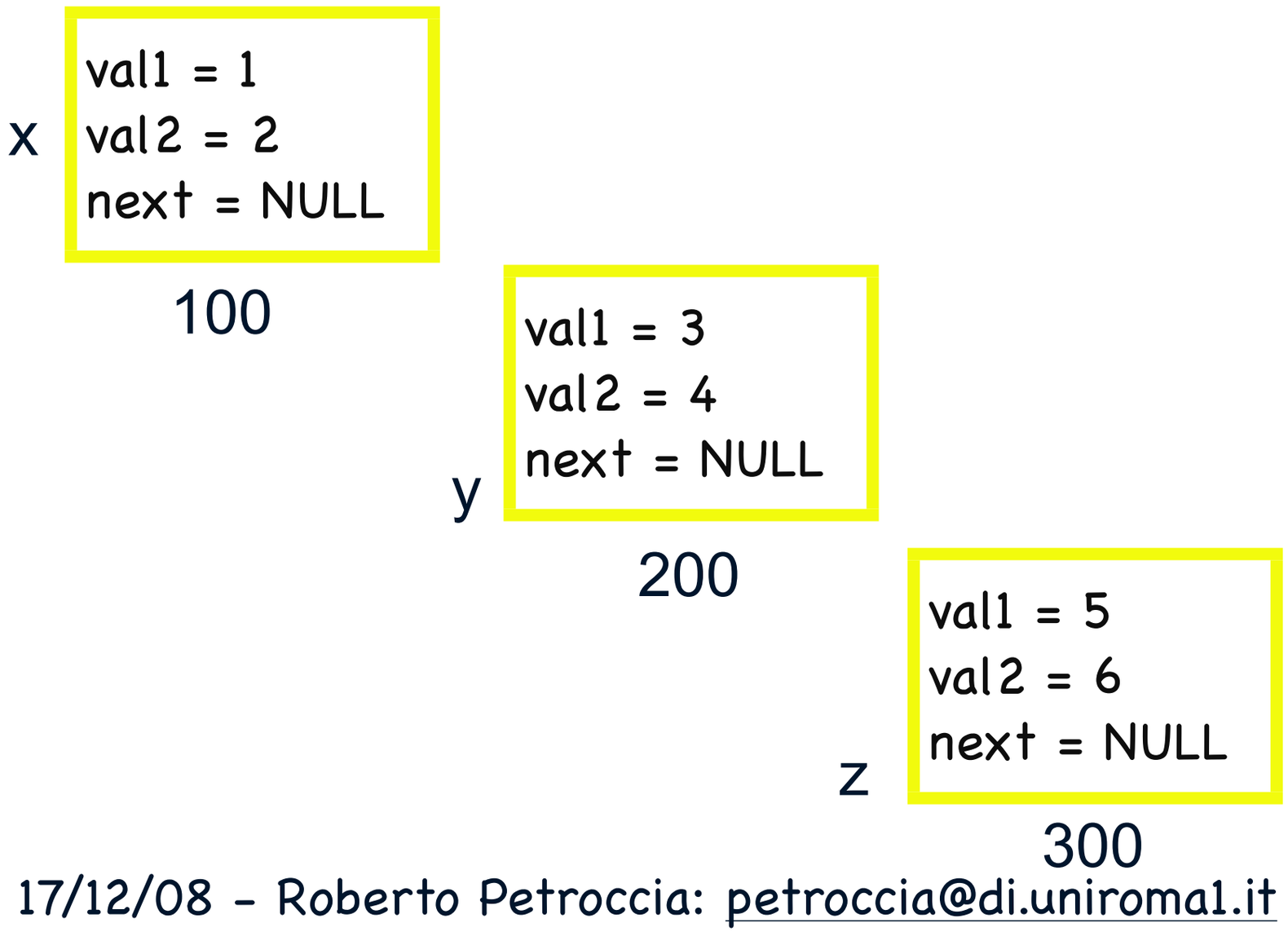
```
coppia x, y, z;

x.val1 = 1;
x.val2 = 2;
x.next = NULL;

y.val1 = 3;
y.val2 = 4;
y.next = NULL;

z.val1 = 5;
z.val2 = 6;
z.next = NULL;
```

LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

```
coppia x, y, z;

x.val1 = 1;
x.val2 = 2;
x.next = NULL;

y.val1 = 3;
y.val2 = 4;
y.next = NULL;

z.val1 = 5;
z.val2 = 6;
z.next = NULL;
```

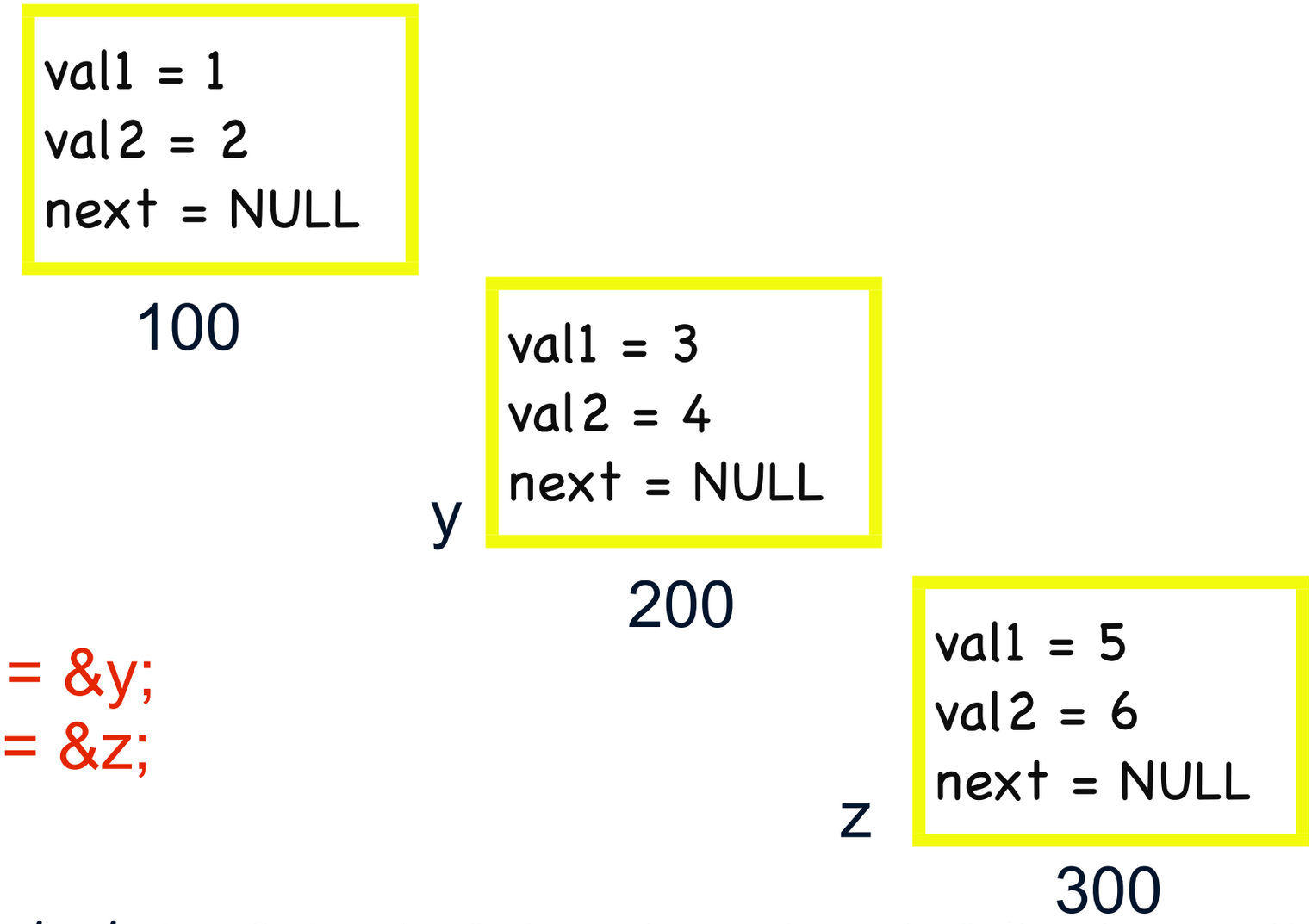


LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

```
coppia x, y, z;

x.val1 = 1;
x.val2 = 2;
x.next = NULL;

y.val1 = 3;
y.val2 = 4;
y.next = NULL;

z.val1 = 5;
z.val2 = 6;
z.next = NULL;

x.next = &y;
y.next = &z;
```

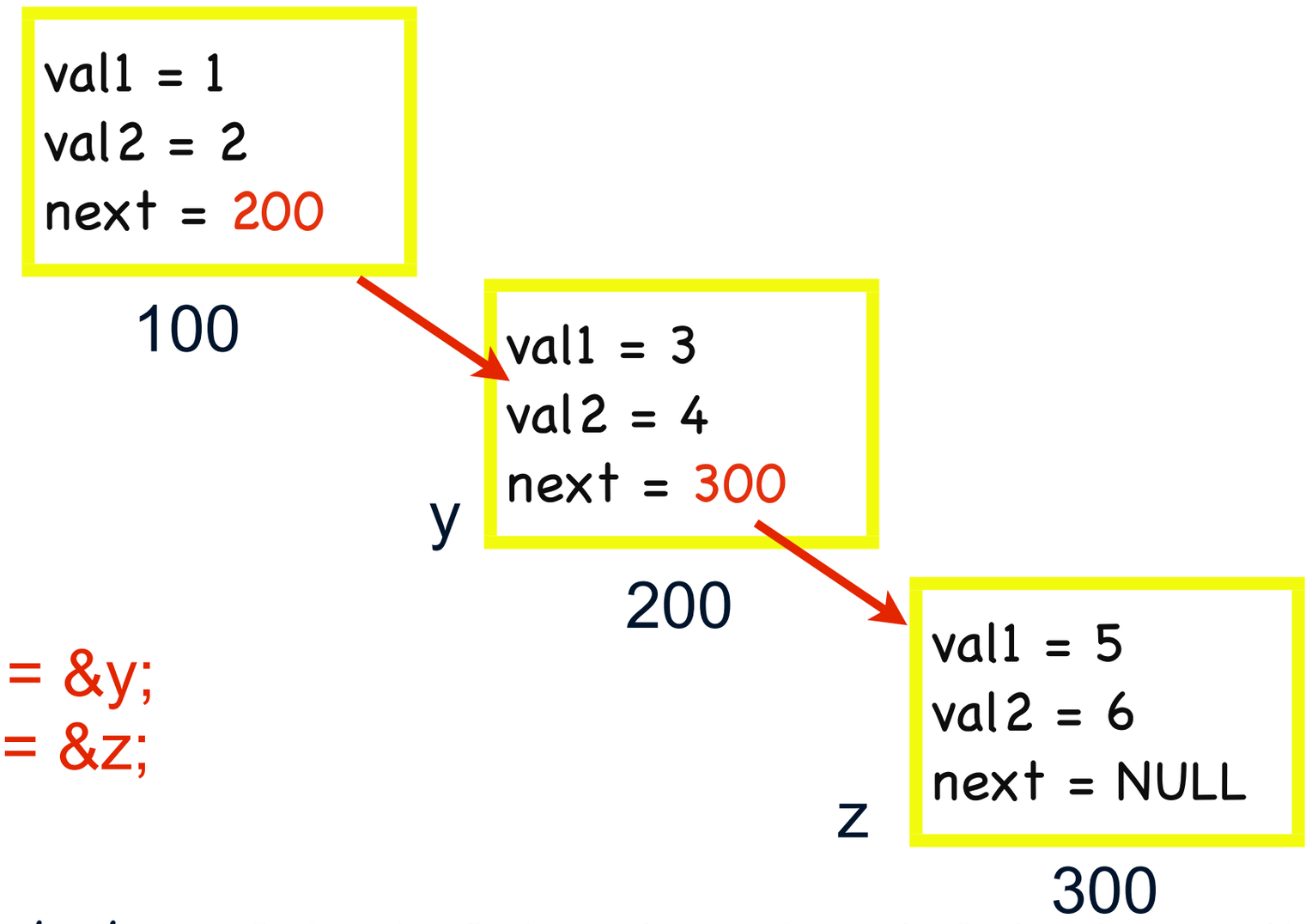


LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

```
coppia x, y, z;

x.val1 = 1;
x.val2 = 2;
x.next = NULL;

y.val1 = 3;
y.val2 = 4;
y.next = NULL;

z.val1 = 5;
z.val2 = 6;
z.next = NULL;

x.next = &y;
y.next = &z;
```

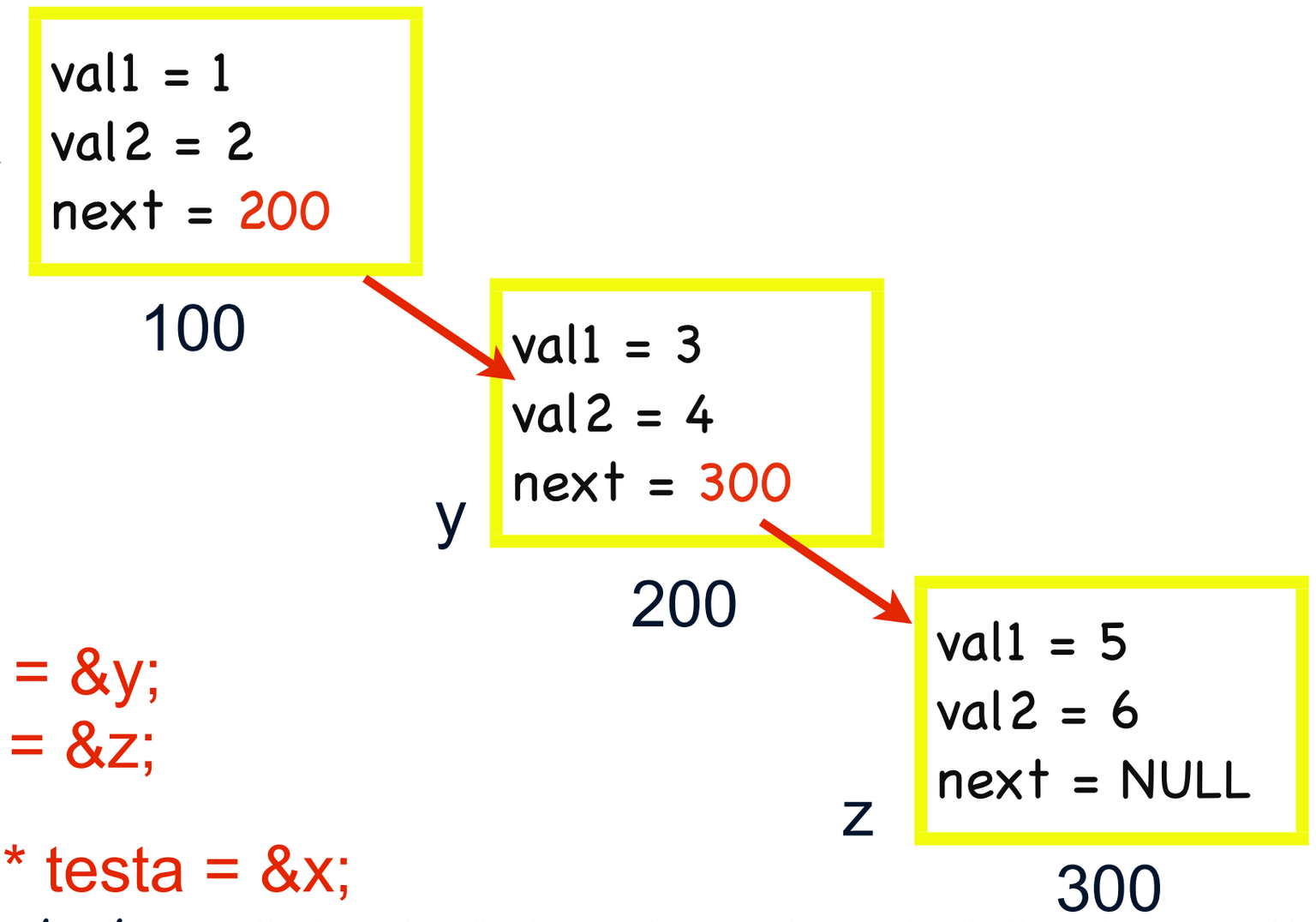


LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

```
coppia x, y, z;

x.val1 = 1;
x.val2 = 2;
x.next = NULL;

y.val1 = 3;
y.val2 = 4;
y.next = NULL;

z.val1 = 5;
z.val2 = 6;
z.next = NULL;

x.next = &y;
y.next = &z;

coppia* testa = &x;
```

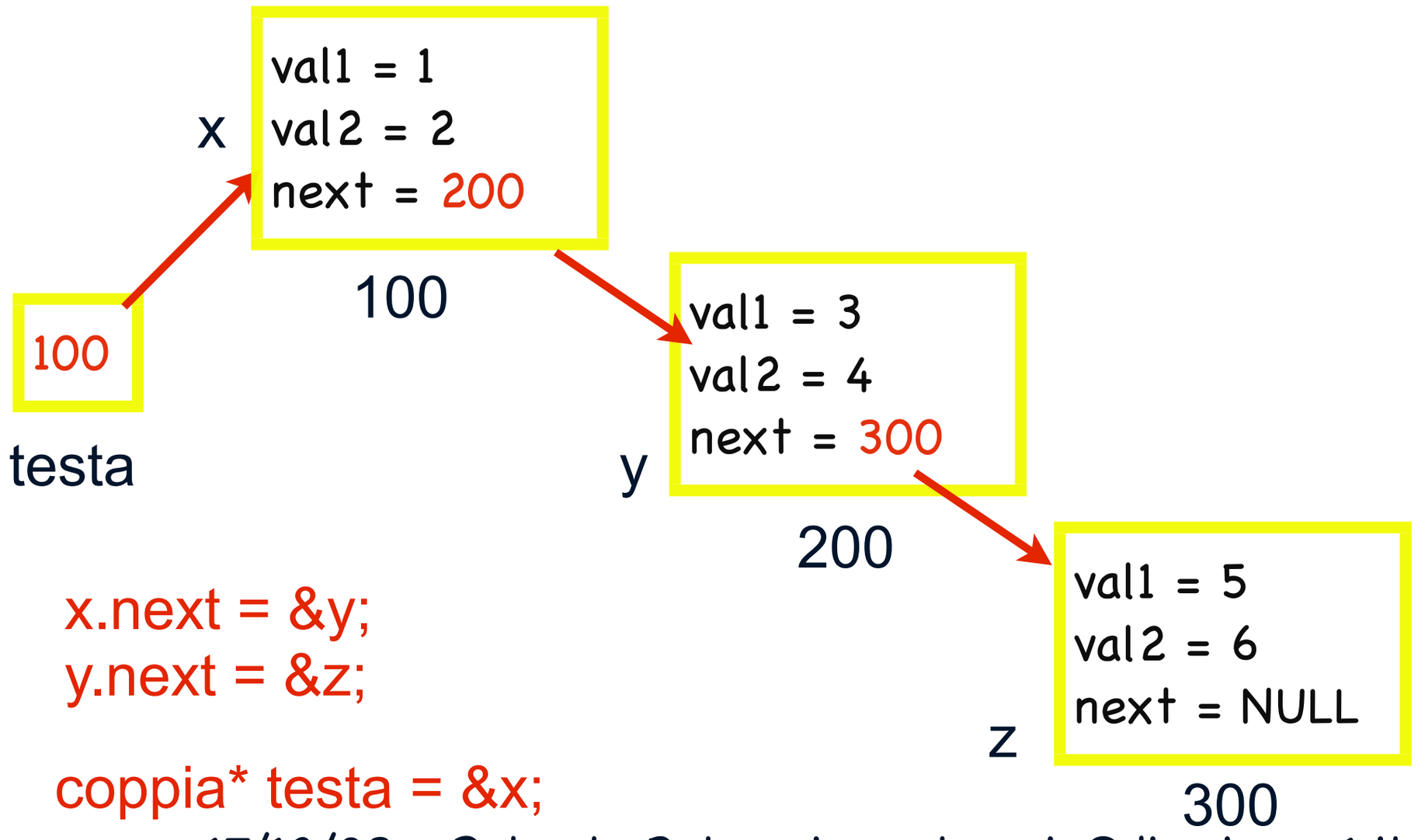


LISTE

Esempio:

```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

```
coppia x, y, z;

x.val1 = 1;
x.val2 = 2;
x.next = NULL;

y.val1 = 3;
y.val2 = 4;
y.next = NULL;

z.val1 = 5;
z.val2 = 6;
z.next = NULL;
```




```
x.next = &y;
y.next = &z;

coppia* testa = &x;
```

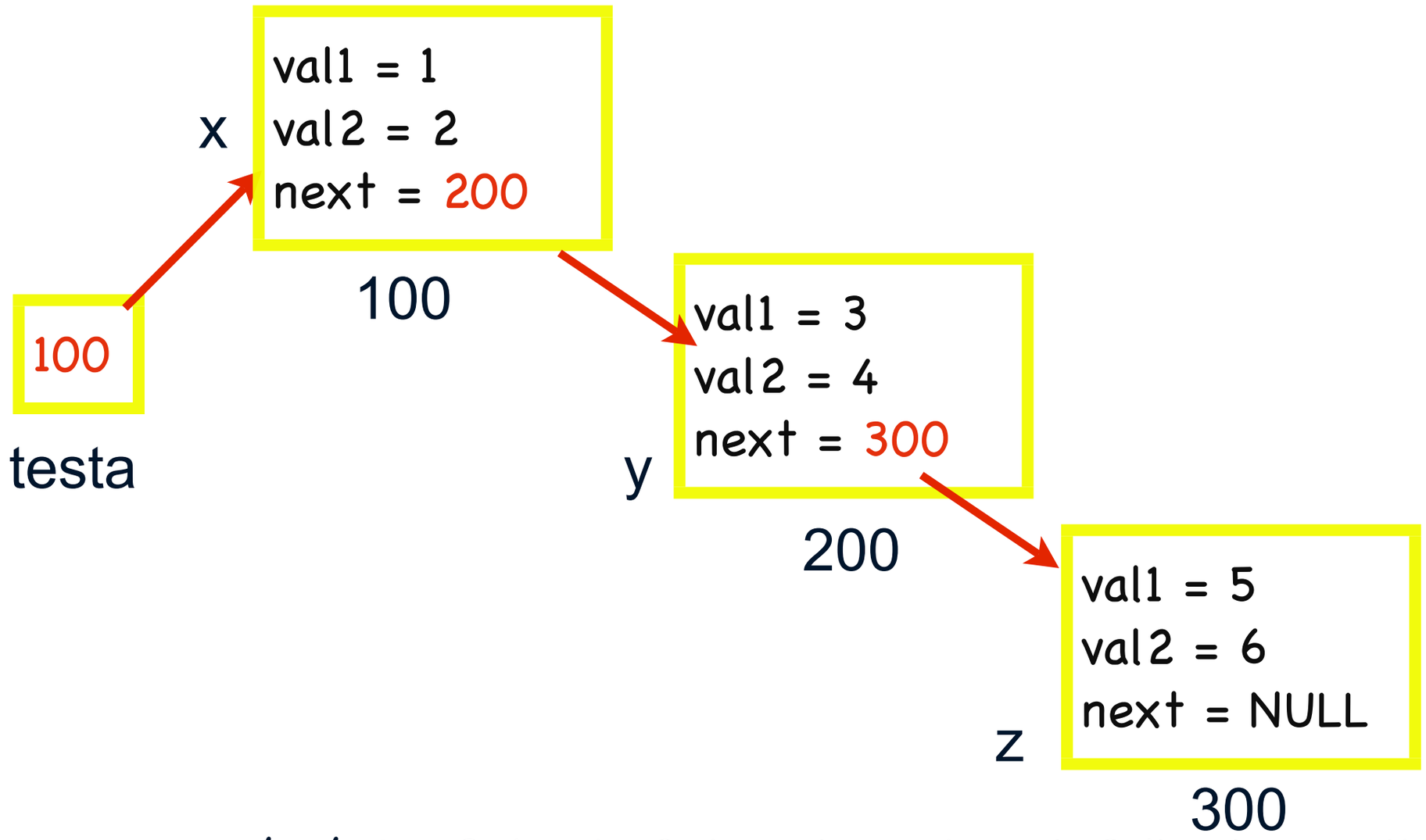
LISTE

Esempio:

quanto vale testa->val1?

quanto vale testa->next->val1?

quanto vale testa->next->next?



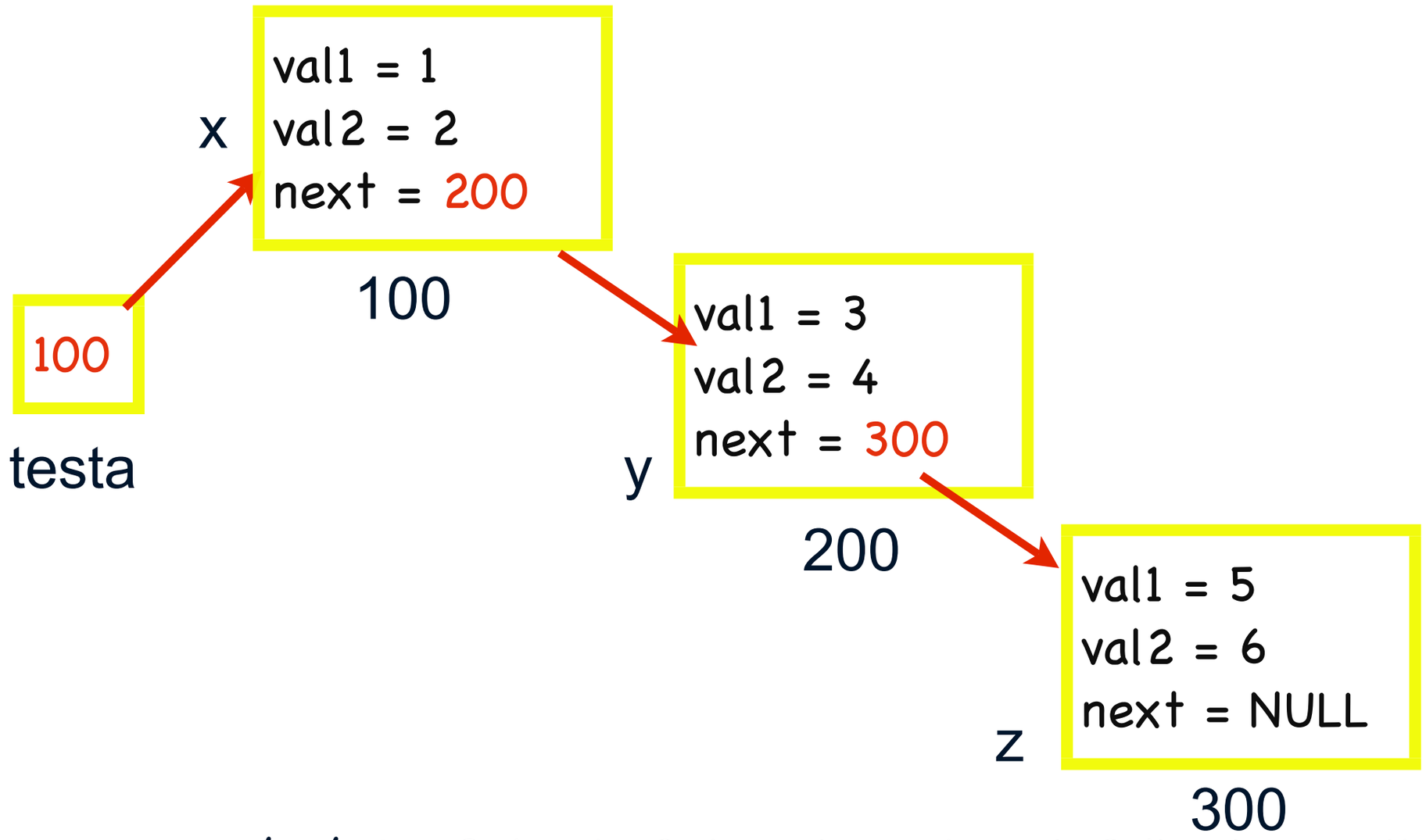
LISTE

Esempio:

quanto vale testa->val1? 1

quanto vale testa->next->val1? 3

quanto vale testa->next->next->next? NULL



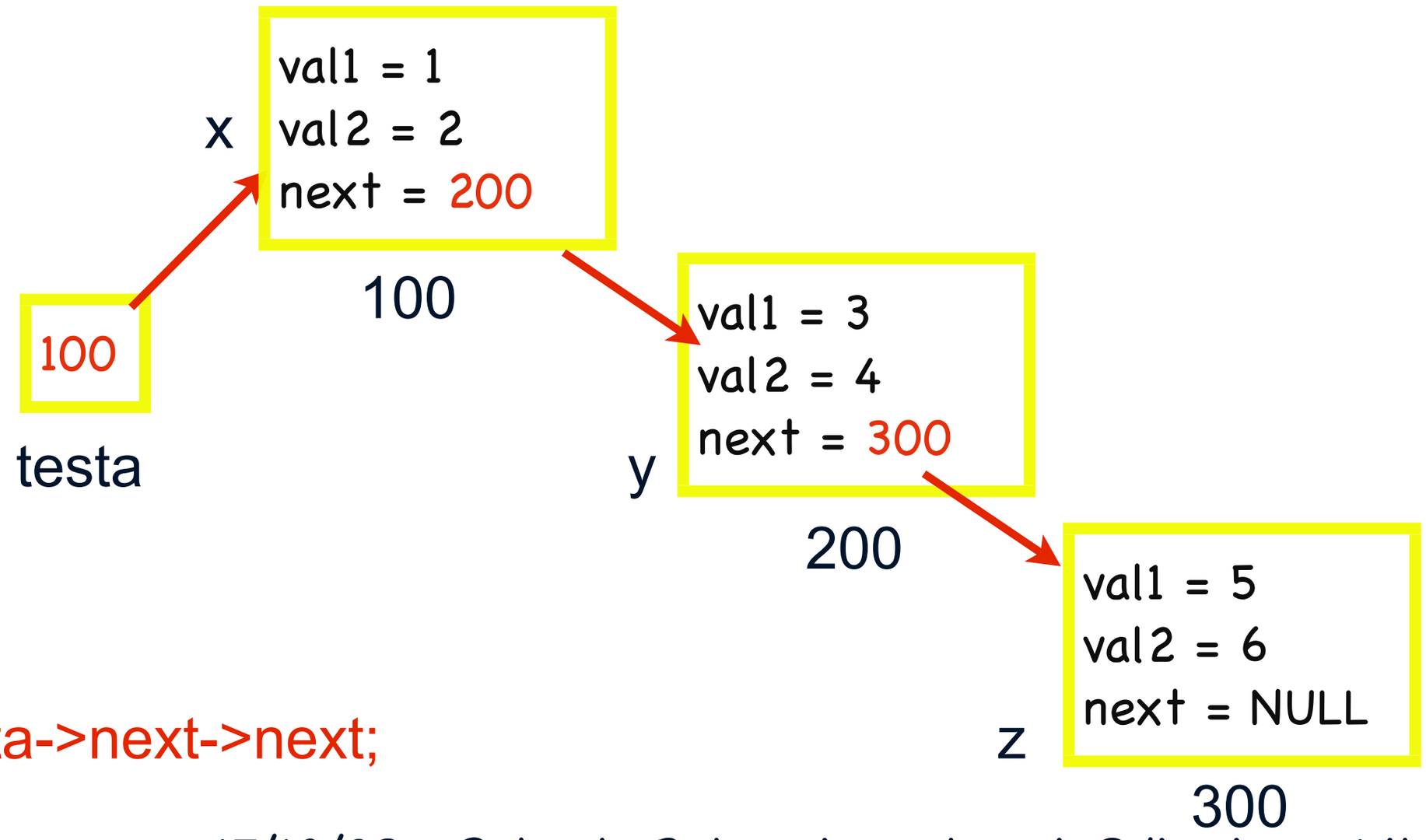
LISTE

Esempio:

quanto vale testa->val1? 1

quanto vale testa->next->val1? 3

quanto vale testa->next->next->next? NULL



testa->next = testa->next->next;

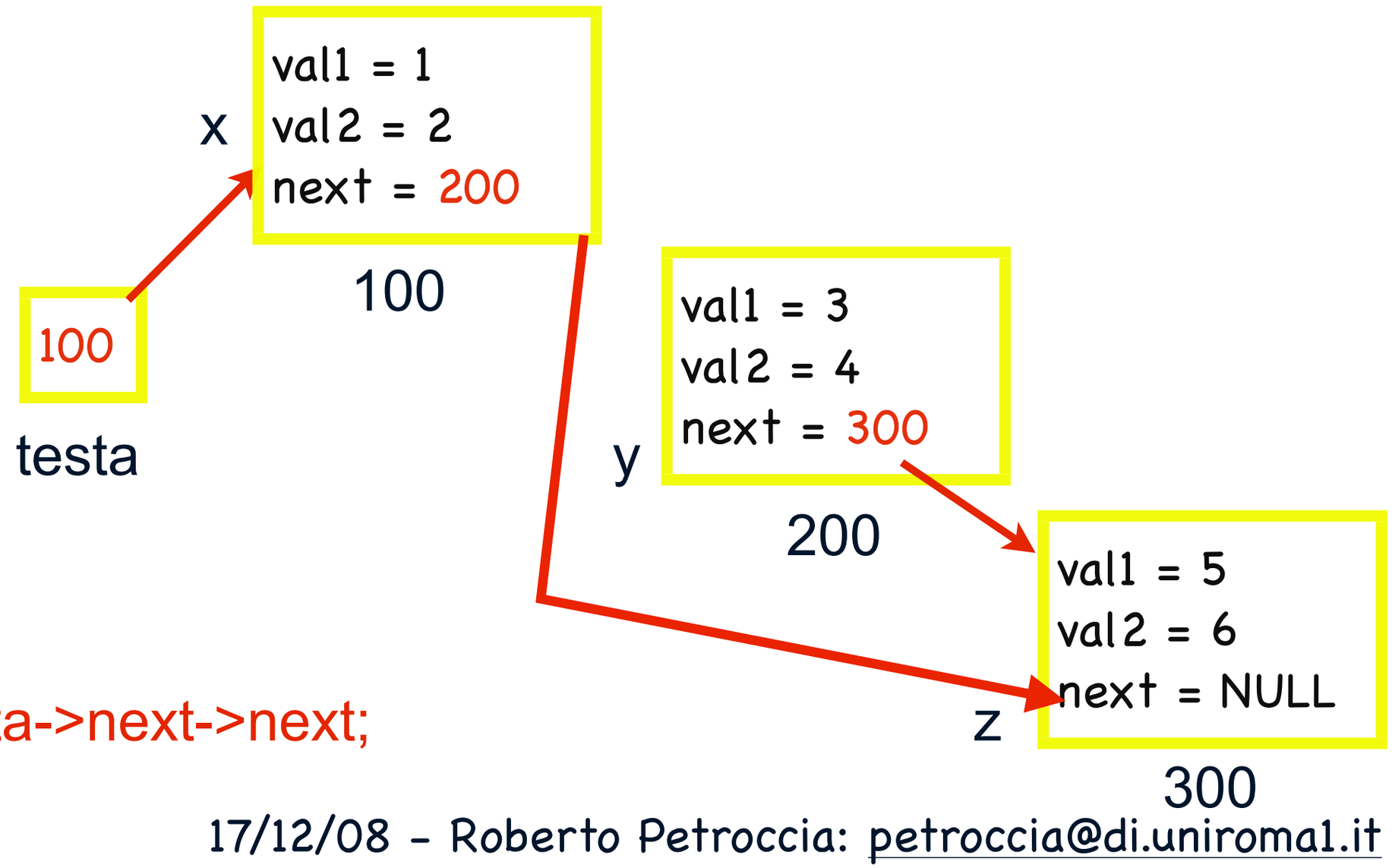
LISTE

Esempio:

quanto vale testa->val1? 1

quanto vale testa->next->val1? 3

quanto vale testa->next->next->next? NULL



testa->next = testa->next->next;

LISTE

- Una lista concatenata è una collezione di strutture ricorsive (nodi) connesse da puntatori

- Si accede alla lista tramite un puntatore al suo primo elemento (testa della lista)

- Si accede agli elementi intermedi per mezzo dei puntatori di concatenazione

- Il puntatore dell'ultimo elemento della lista deve essere posto a NULL.

LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```

LISTE

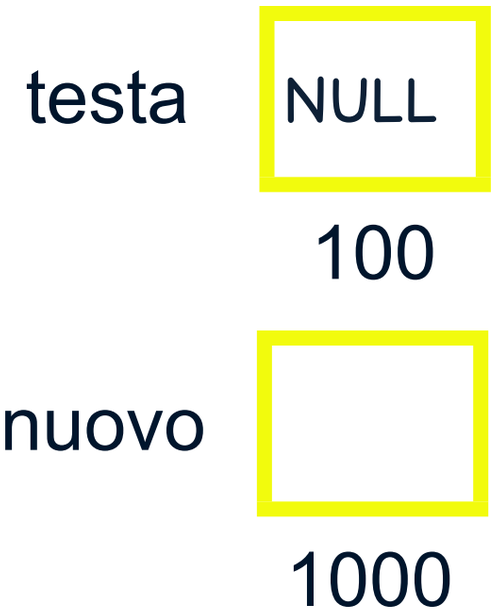
```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```

LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```

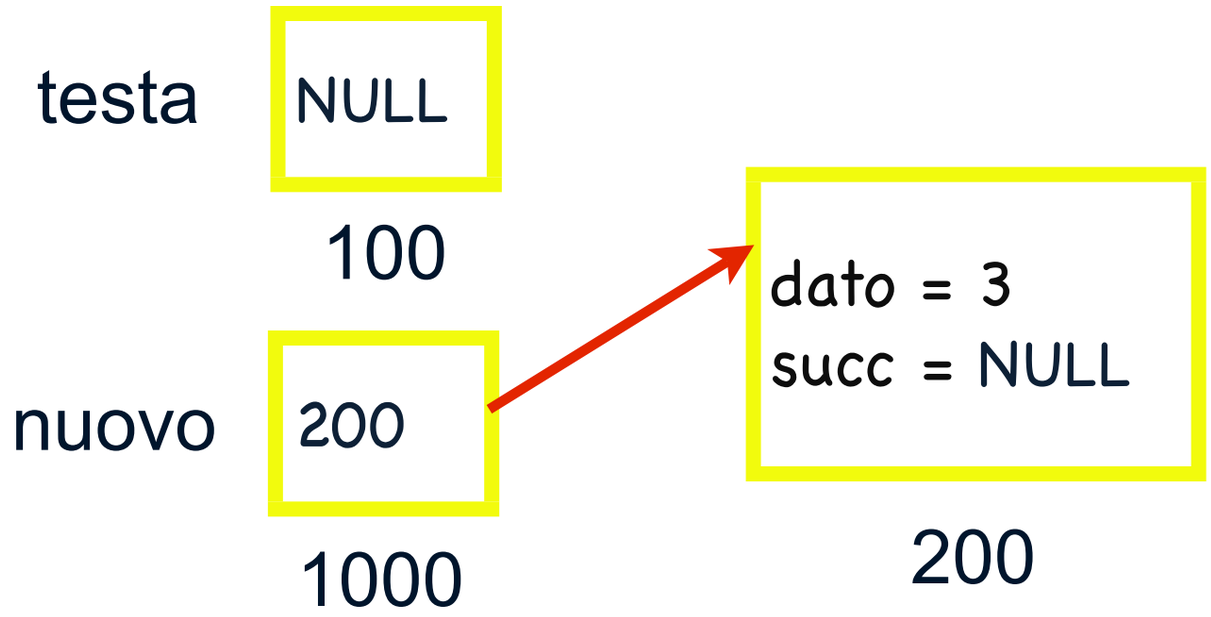


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;
```

```
nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

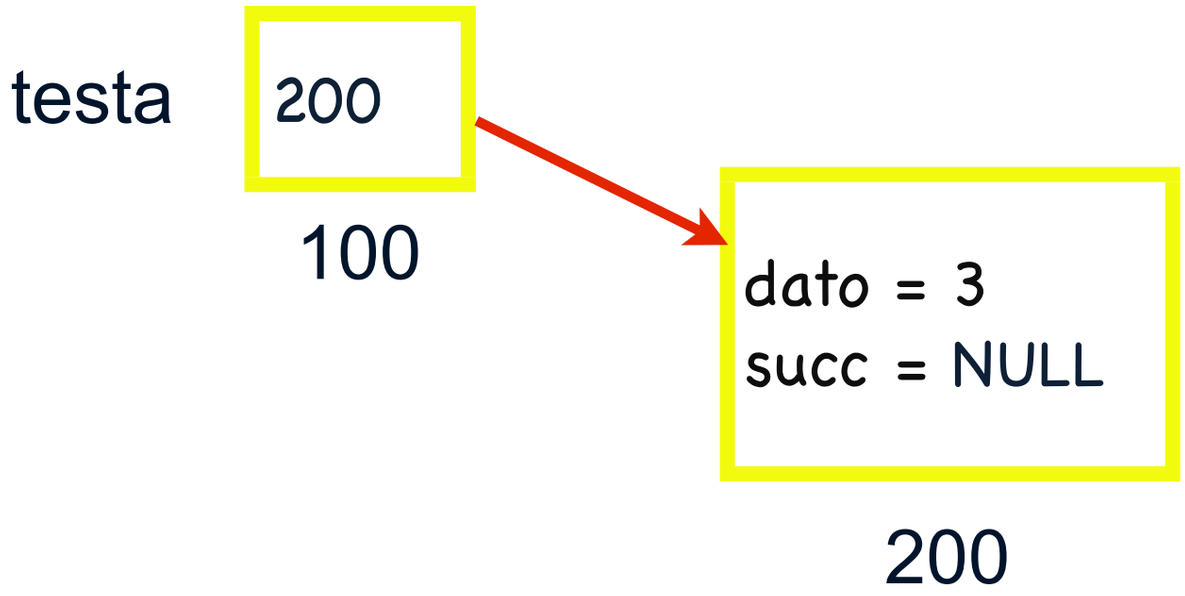
```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

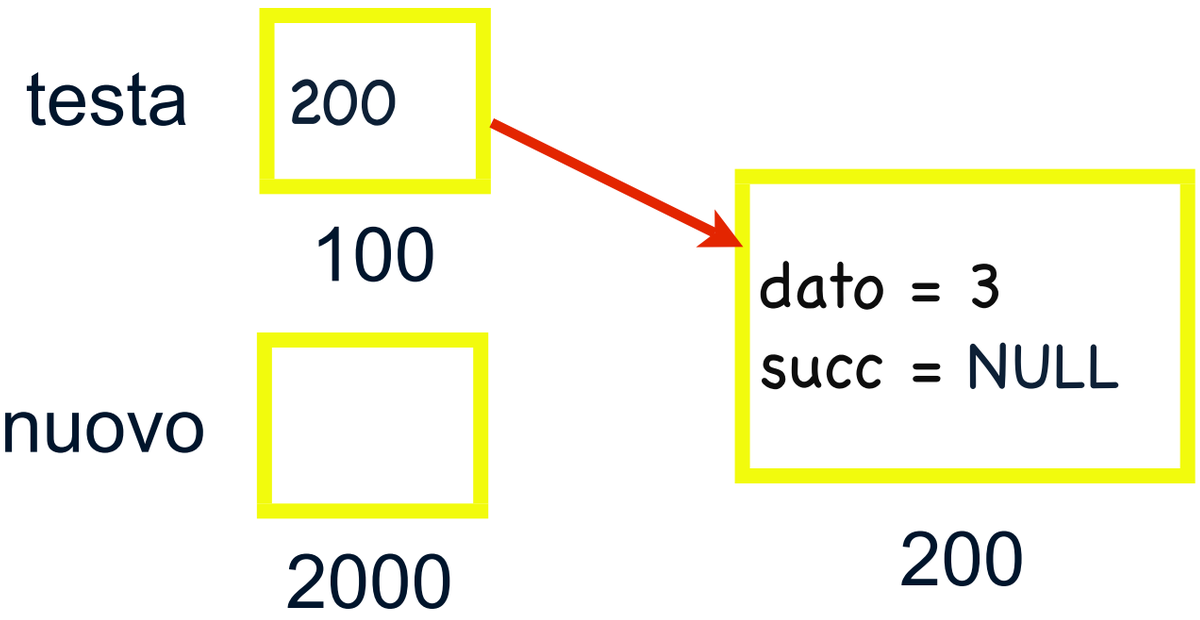
```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

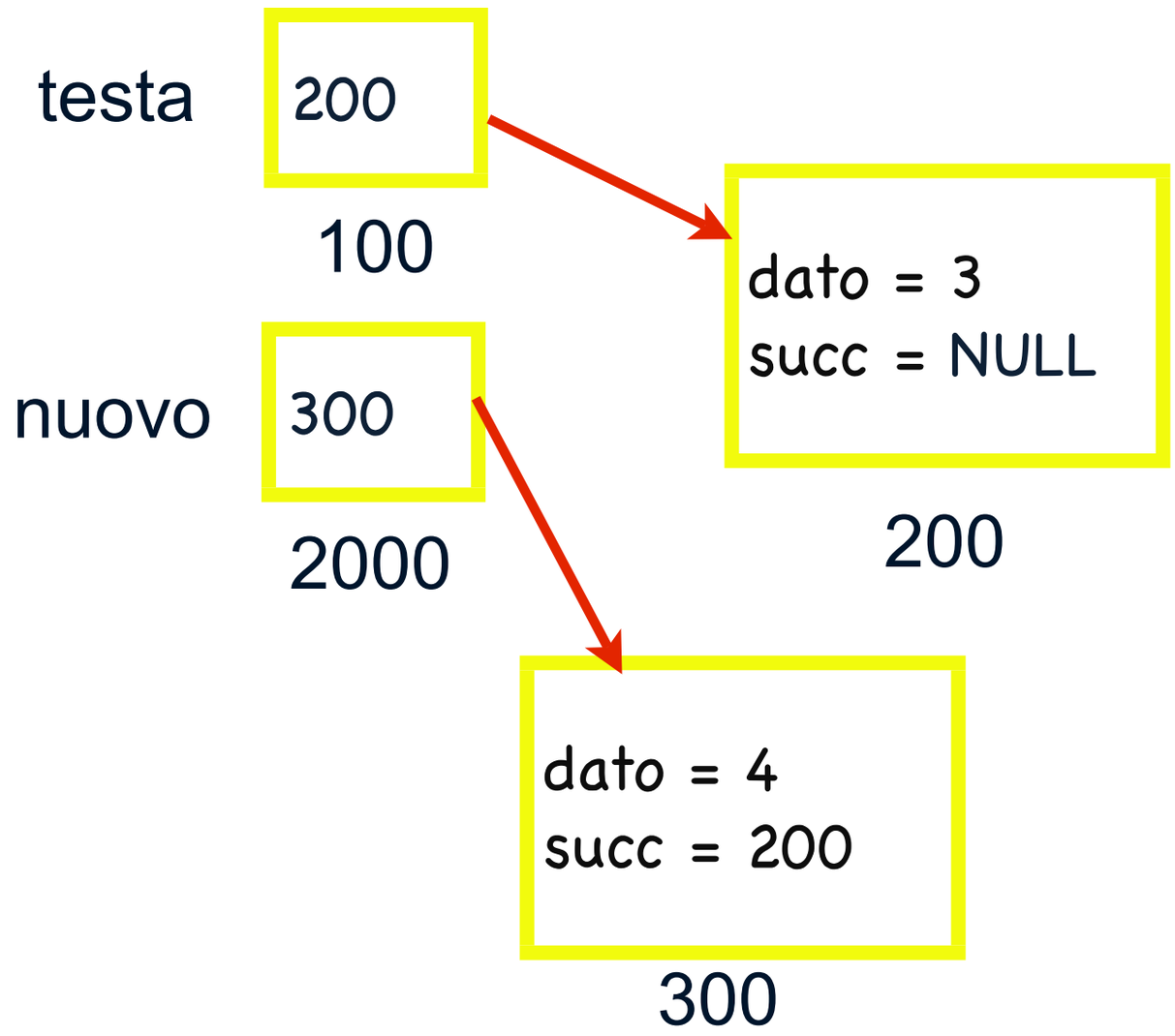
```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

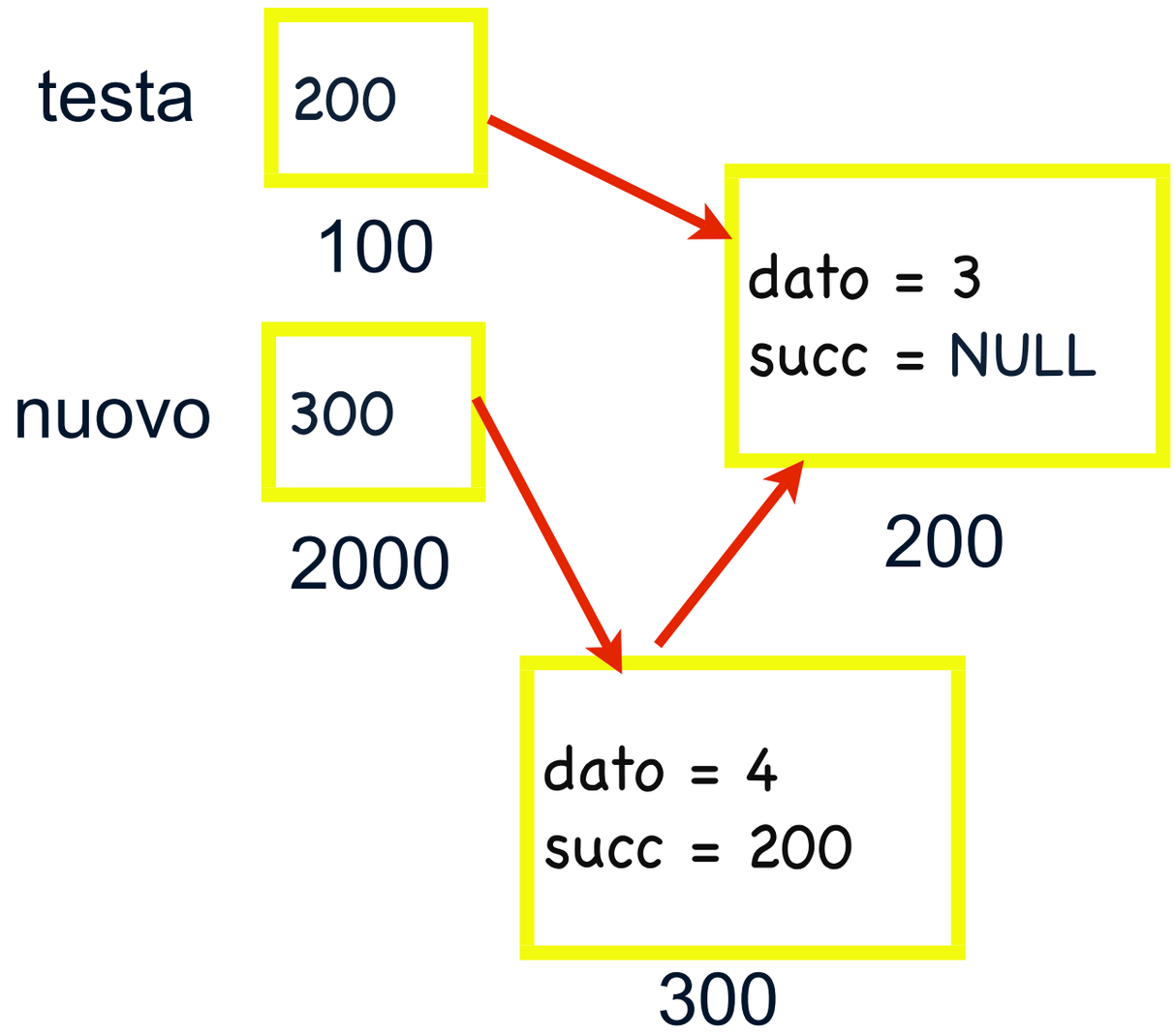
```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

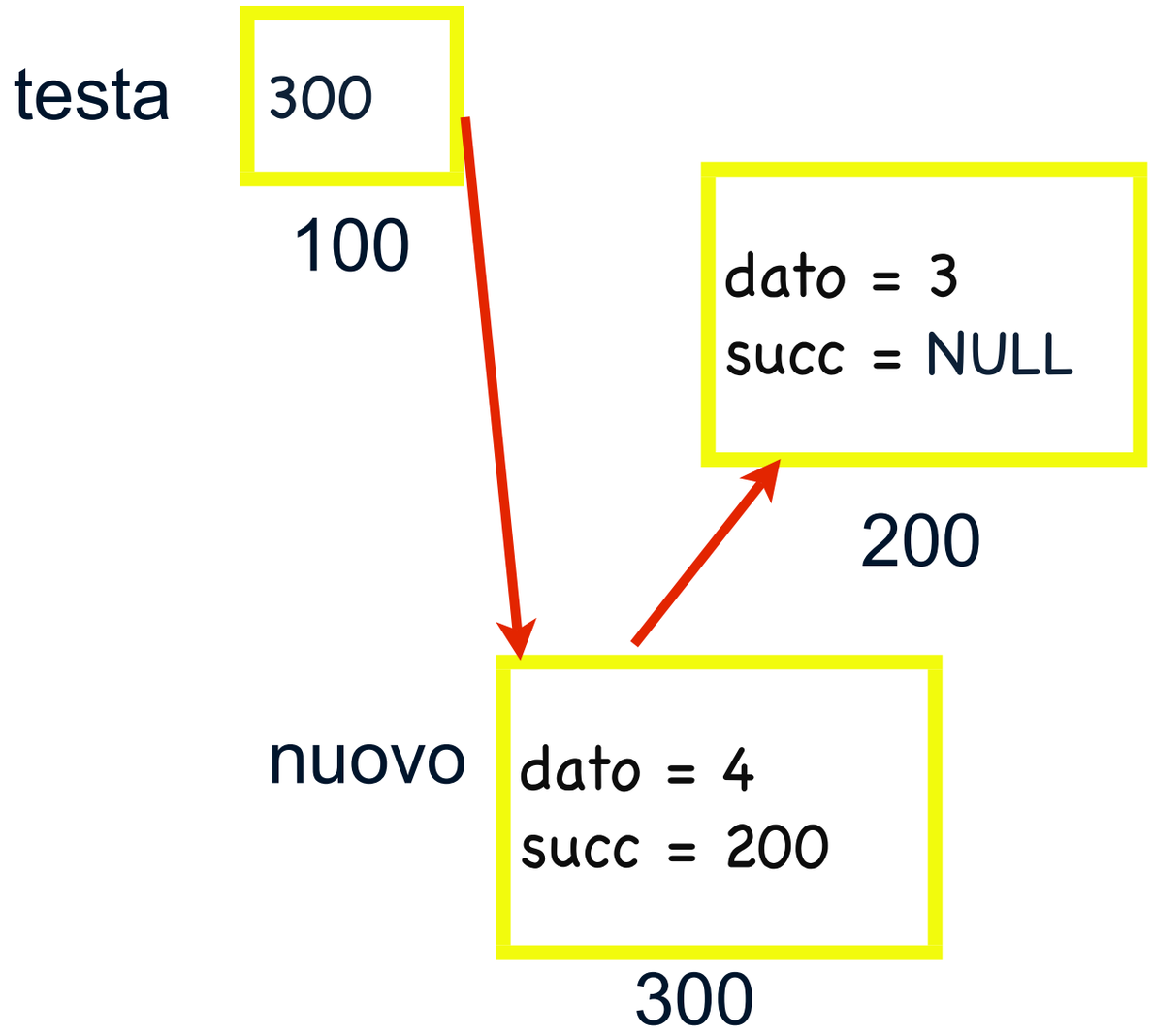
```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

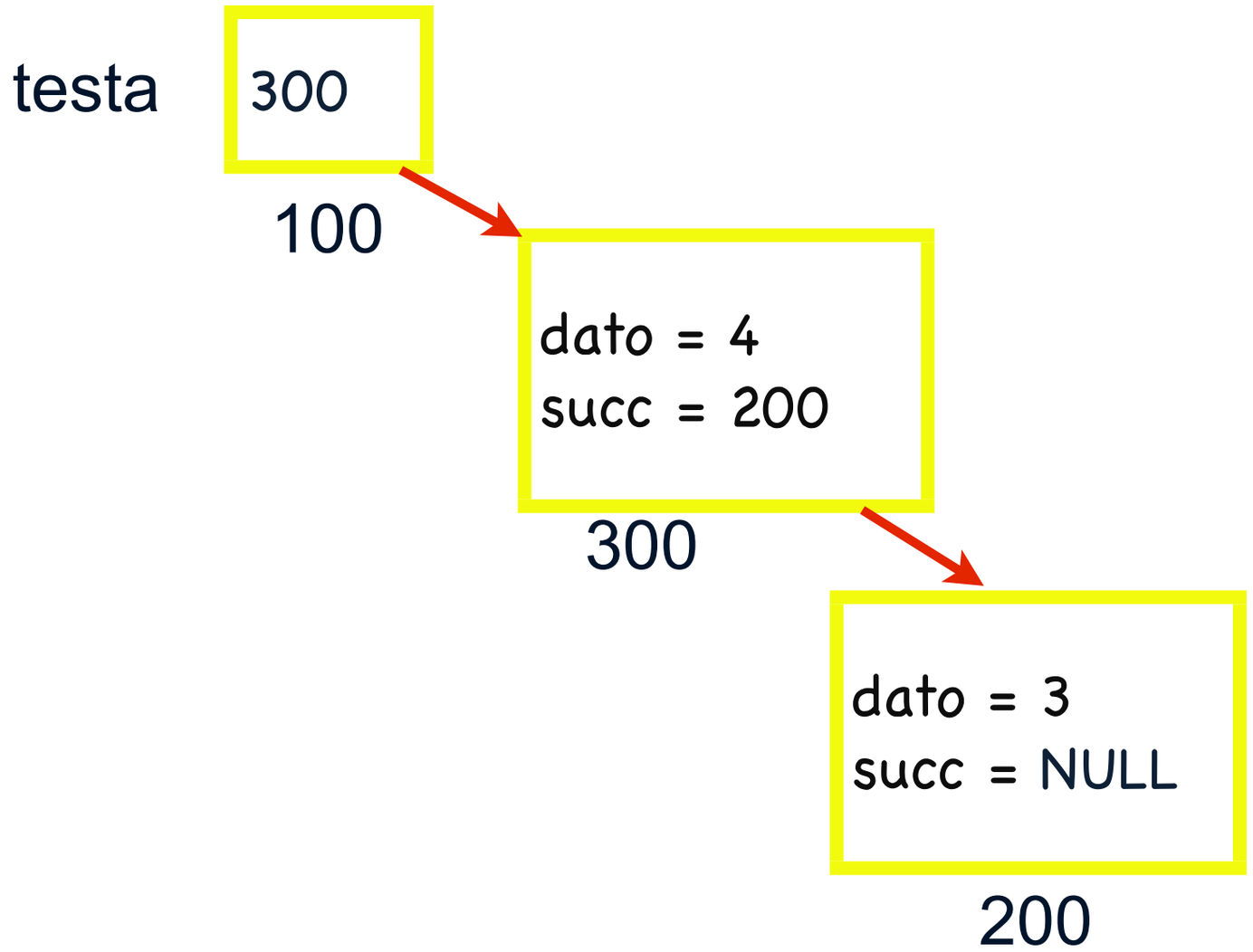
```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;


nl* InserisciInTesta(nl* testa, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=testa;
       return nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```

testa NULL

100

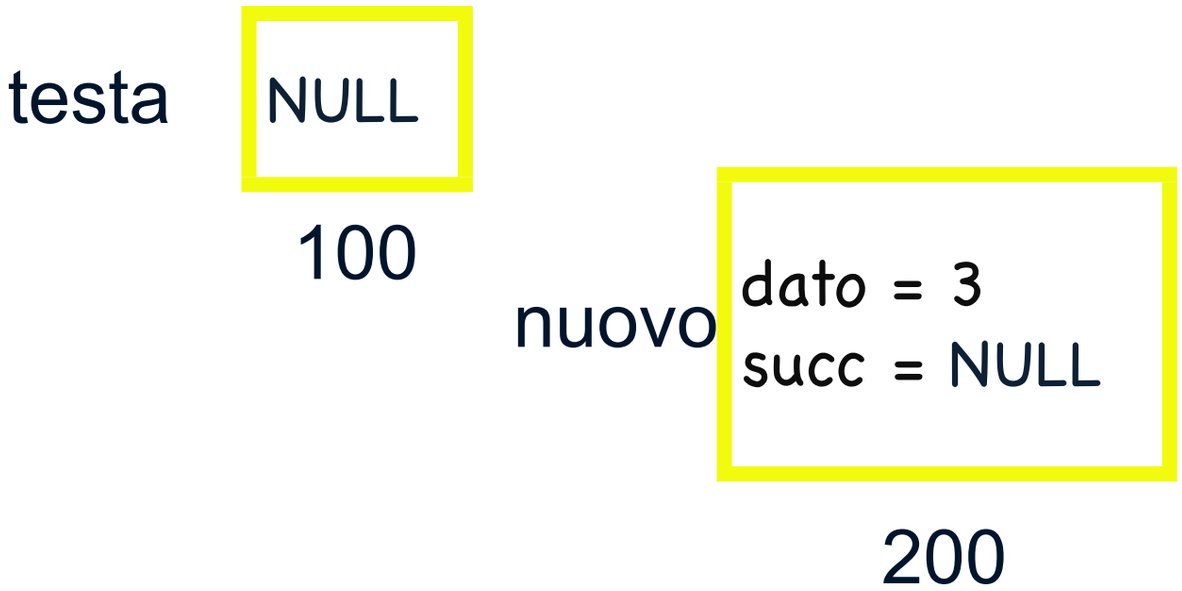
```
nl* InserisciInTesta(nl* testa, int val)
{
   nl nuovo;
   nuovo.dato=val;
   nuovo.succ=testa;
   return &nuovo;
}
```

LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;
```

```
nl* InserisciInTesta(nl* testa, int val)
{
   nl nuovo;
   nuovo.dato=val;
   nuovo.succ=testa;
   return &nuovo;
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```

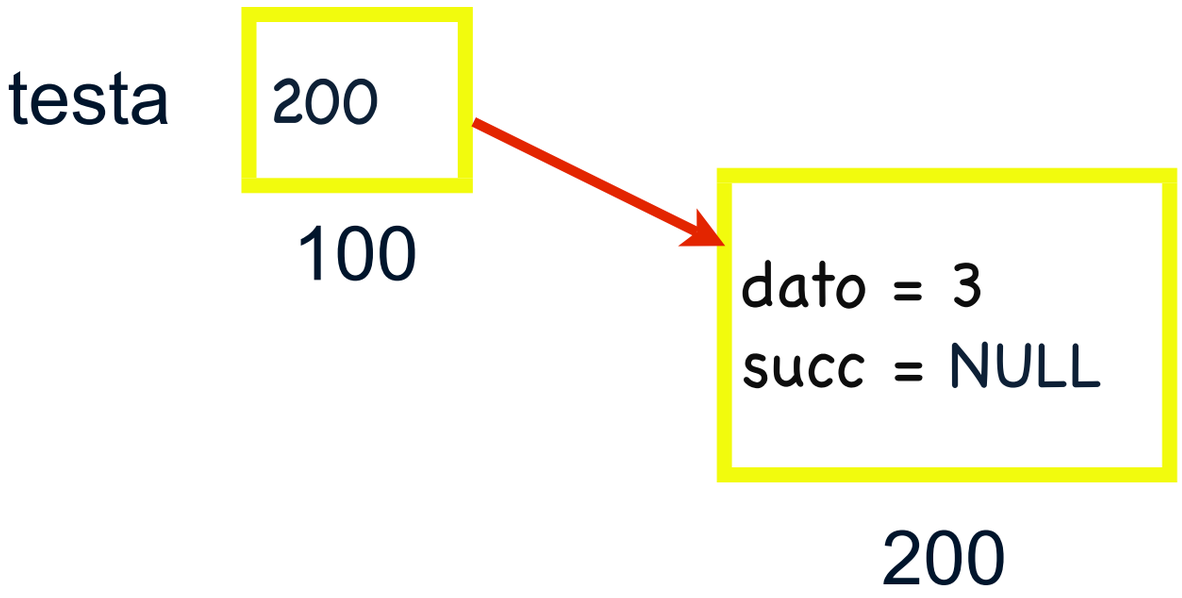


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;
```

```
nl* InserisciInTesta(nl* testa, int val)
{
   nl nuovo;
   nuovo.dato=val;
   nuovo.succ=testa;
   return &nuovo;
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```

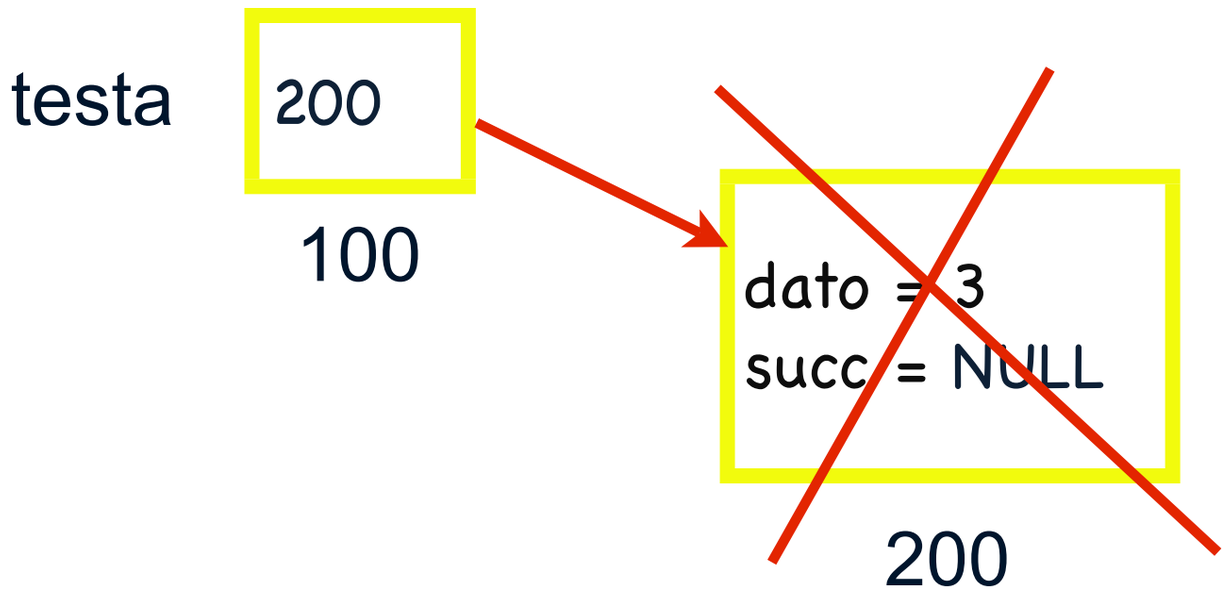


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;
```

```
nl* InserisciInTesta(nl* testa, int val)
{
   nl nuovo;
   nuovo.dato=val;
   nuovo.succ=testa;
   return &nuovo;
}
```

```
int main
{
   nl* testa = NULL;
   testa = inserisciInTesta(testa, 3);
   testa = inserisciInTesta(testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

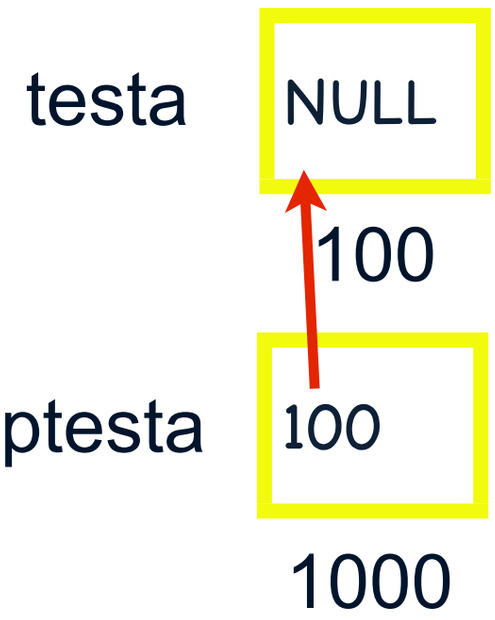

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

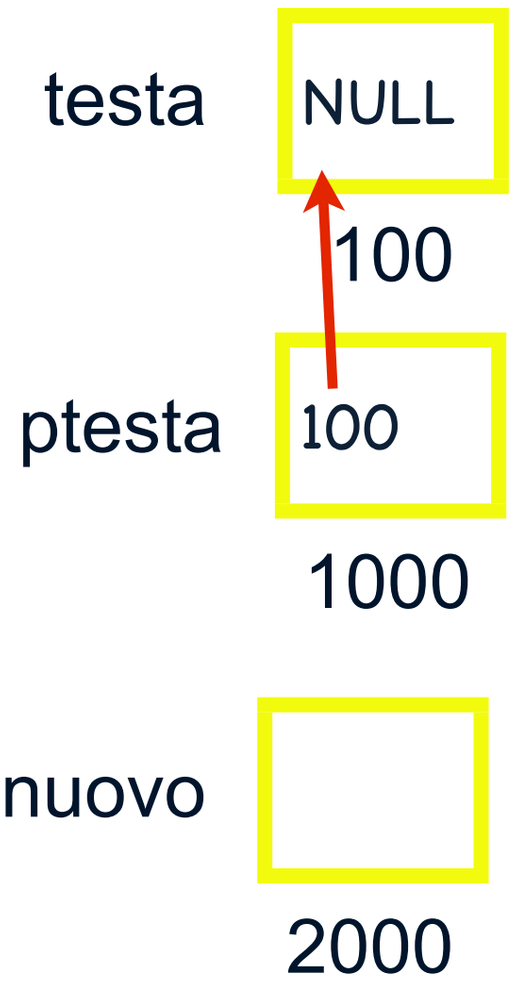


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

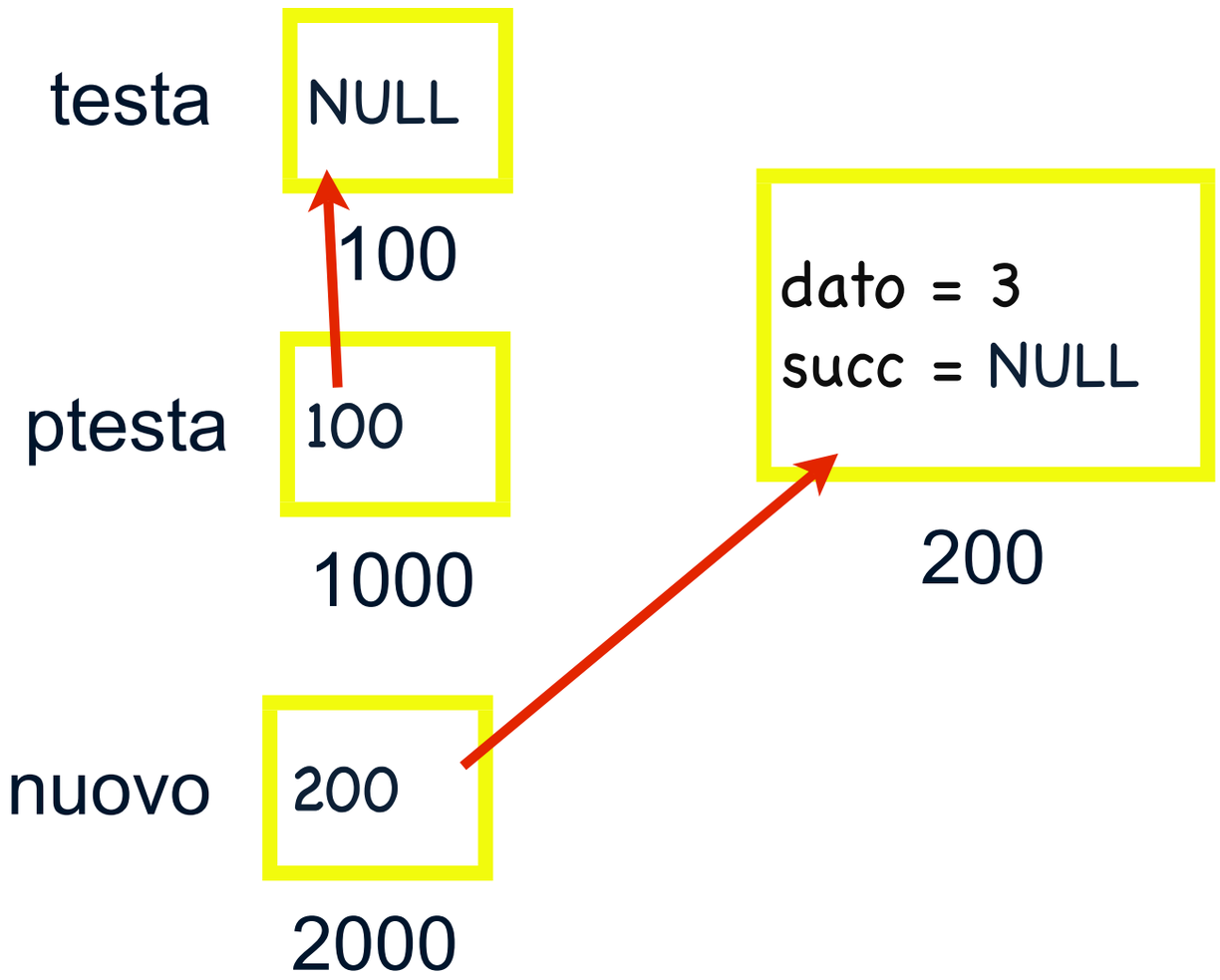


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

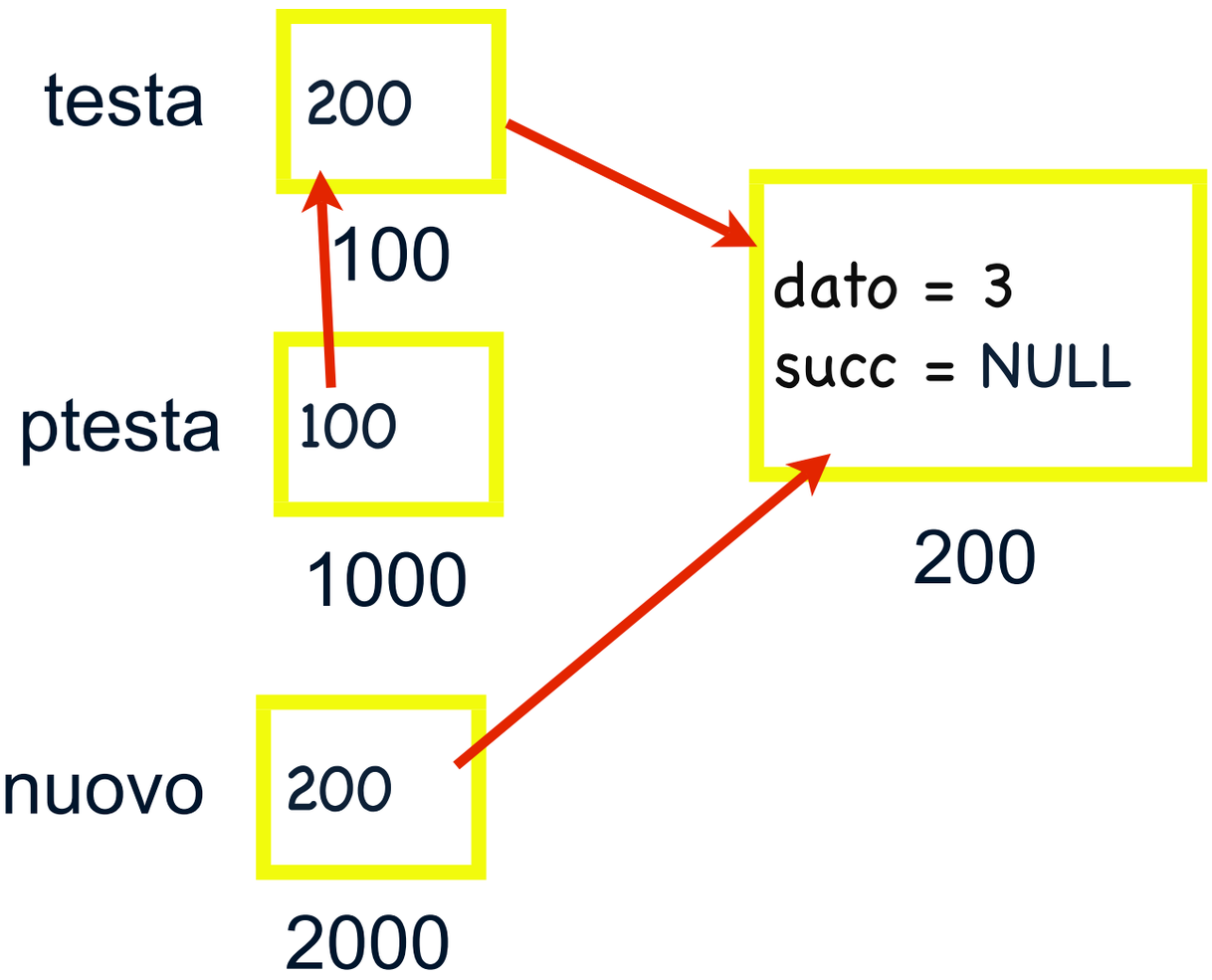


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

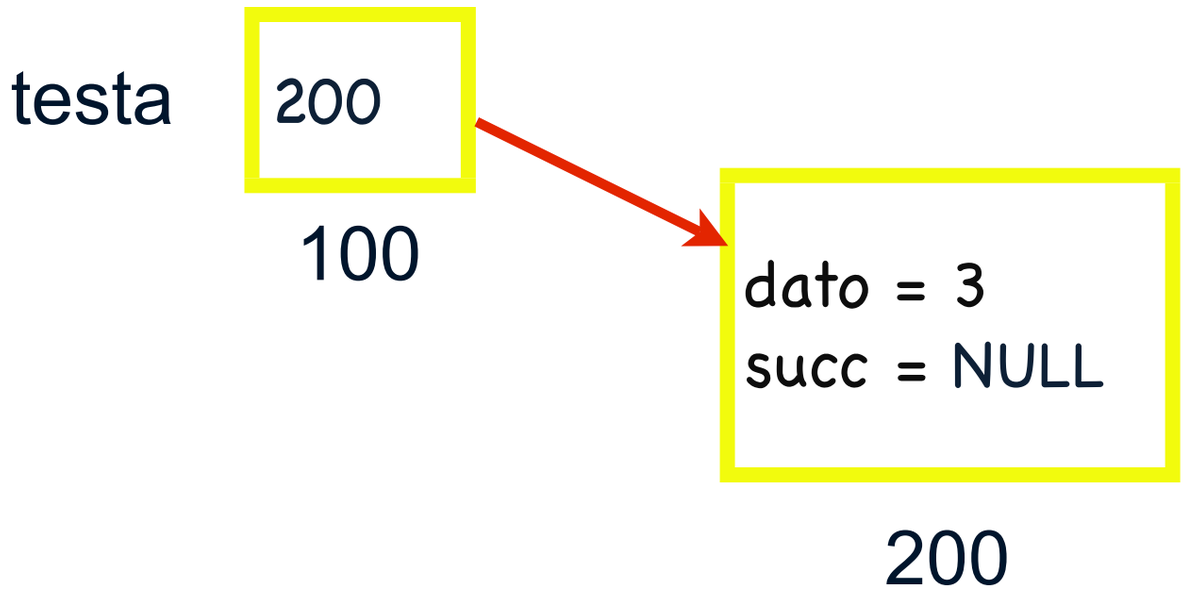
```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```



LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```



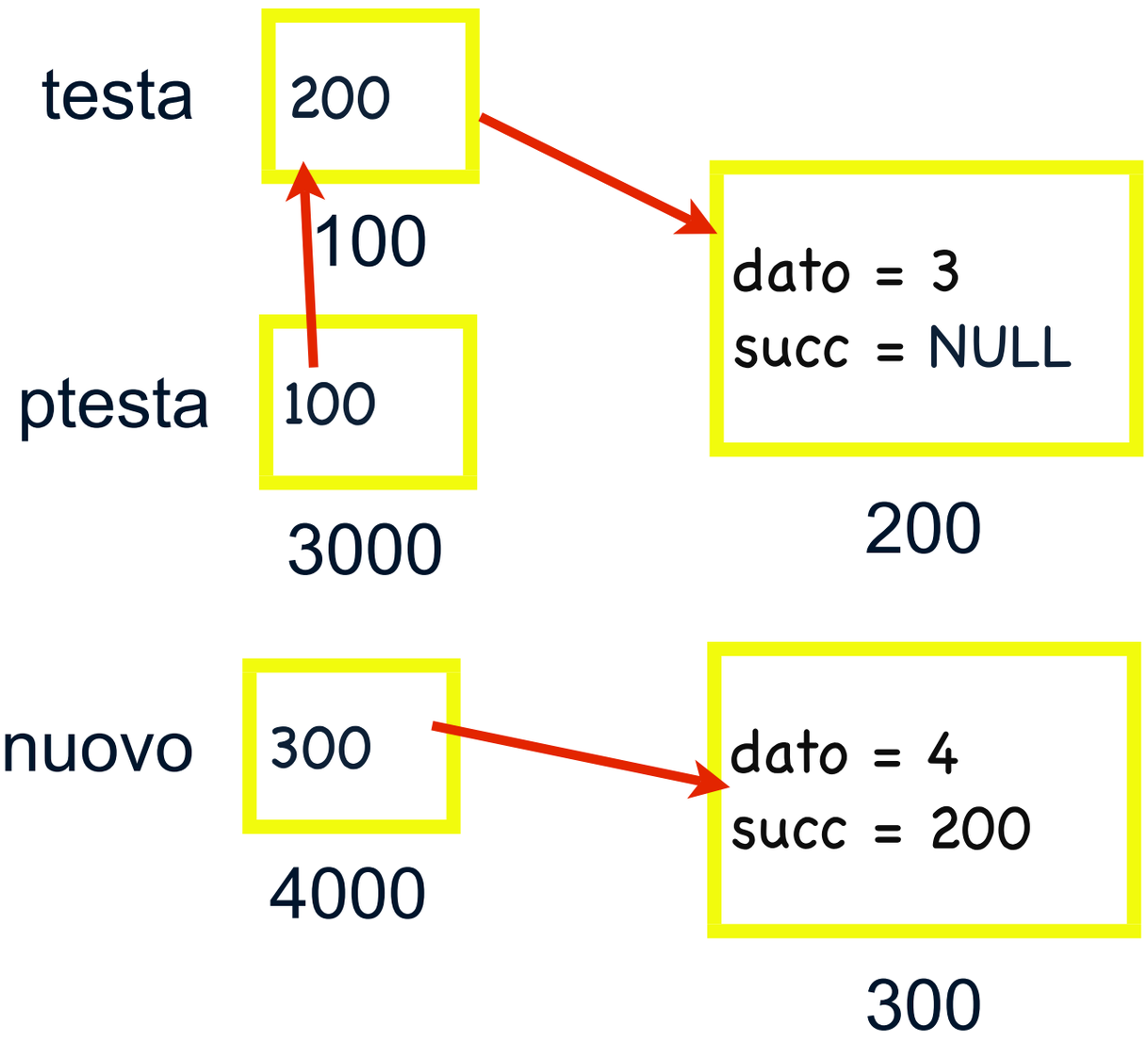

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

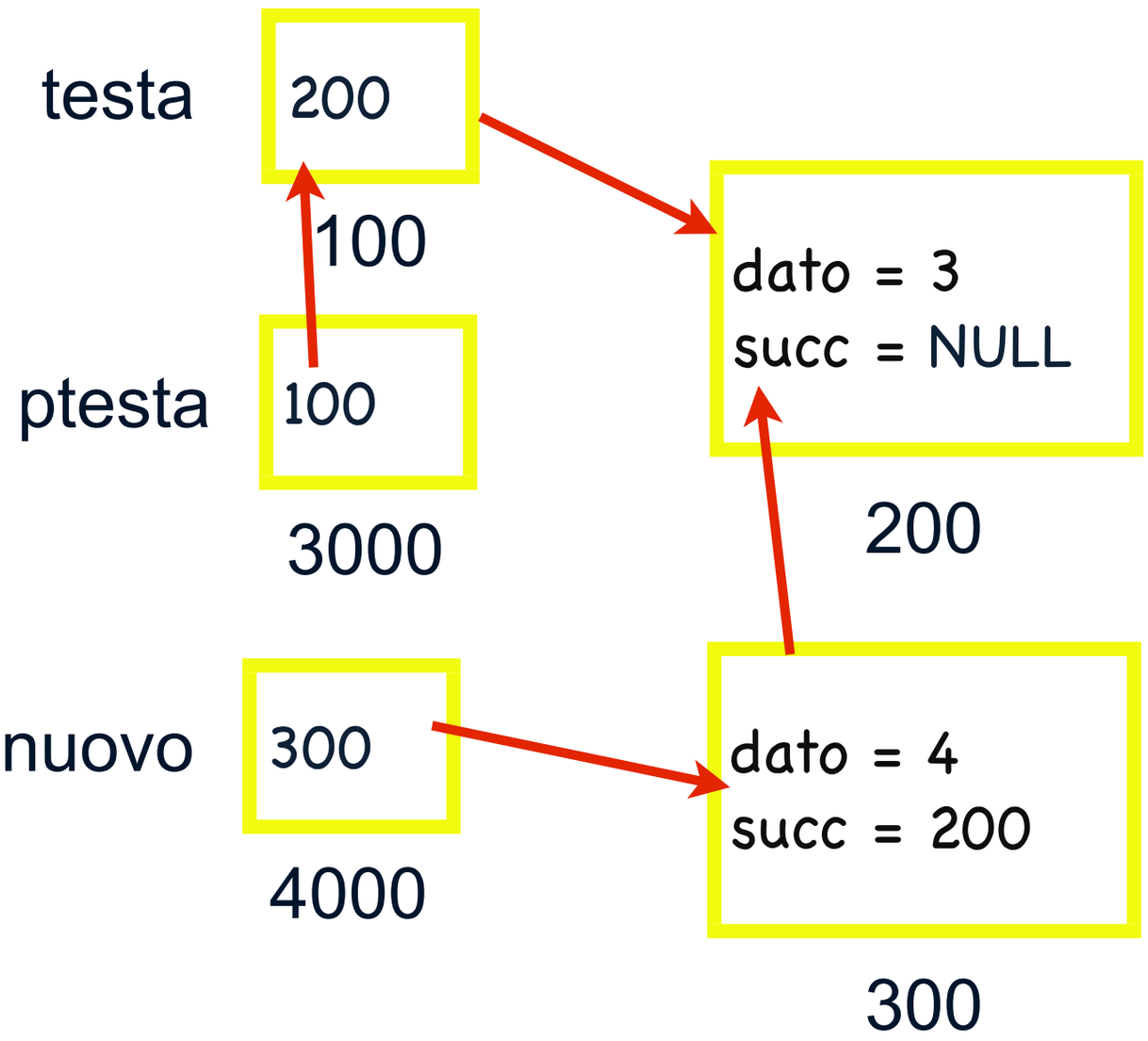


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

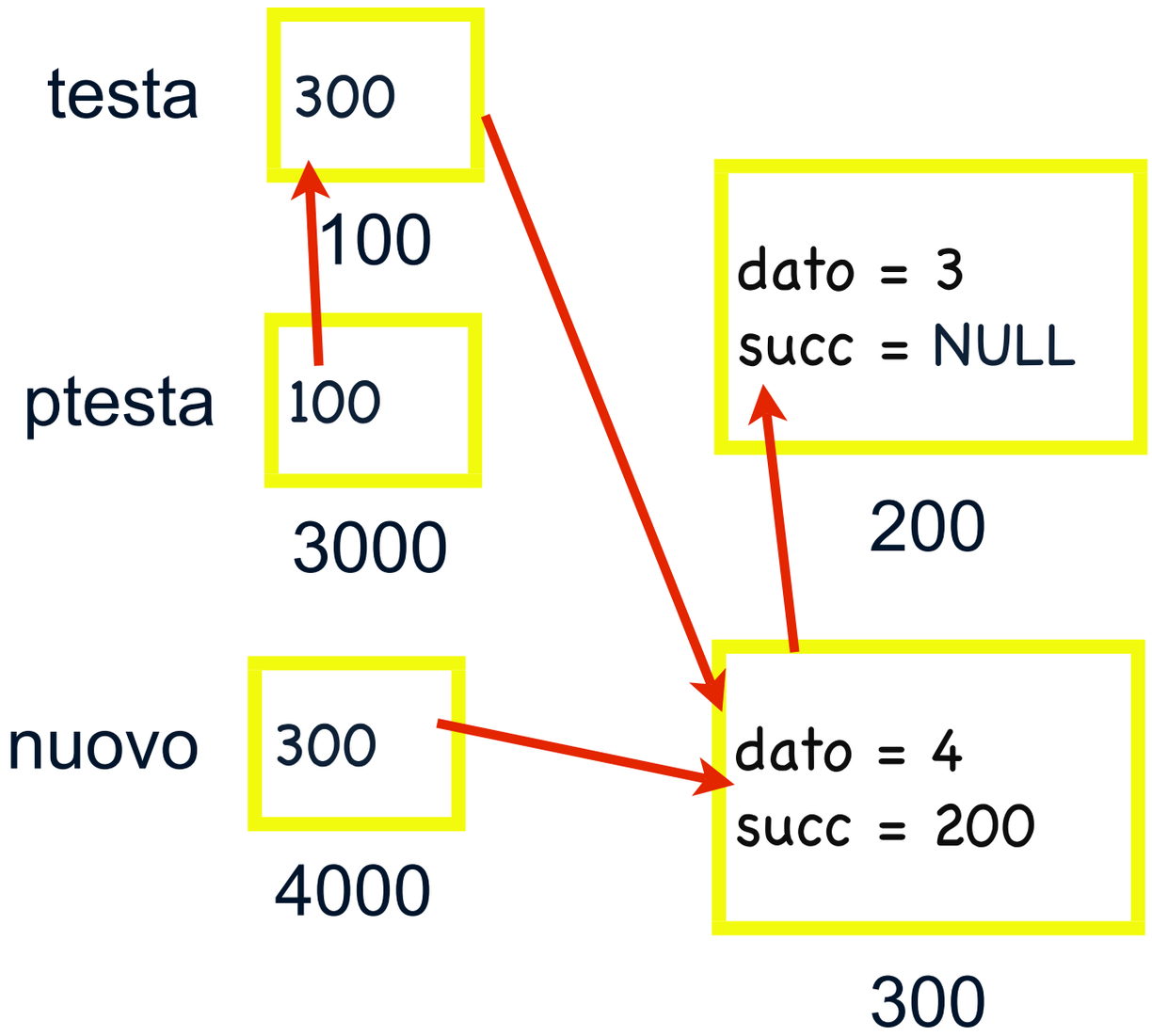


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

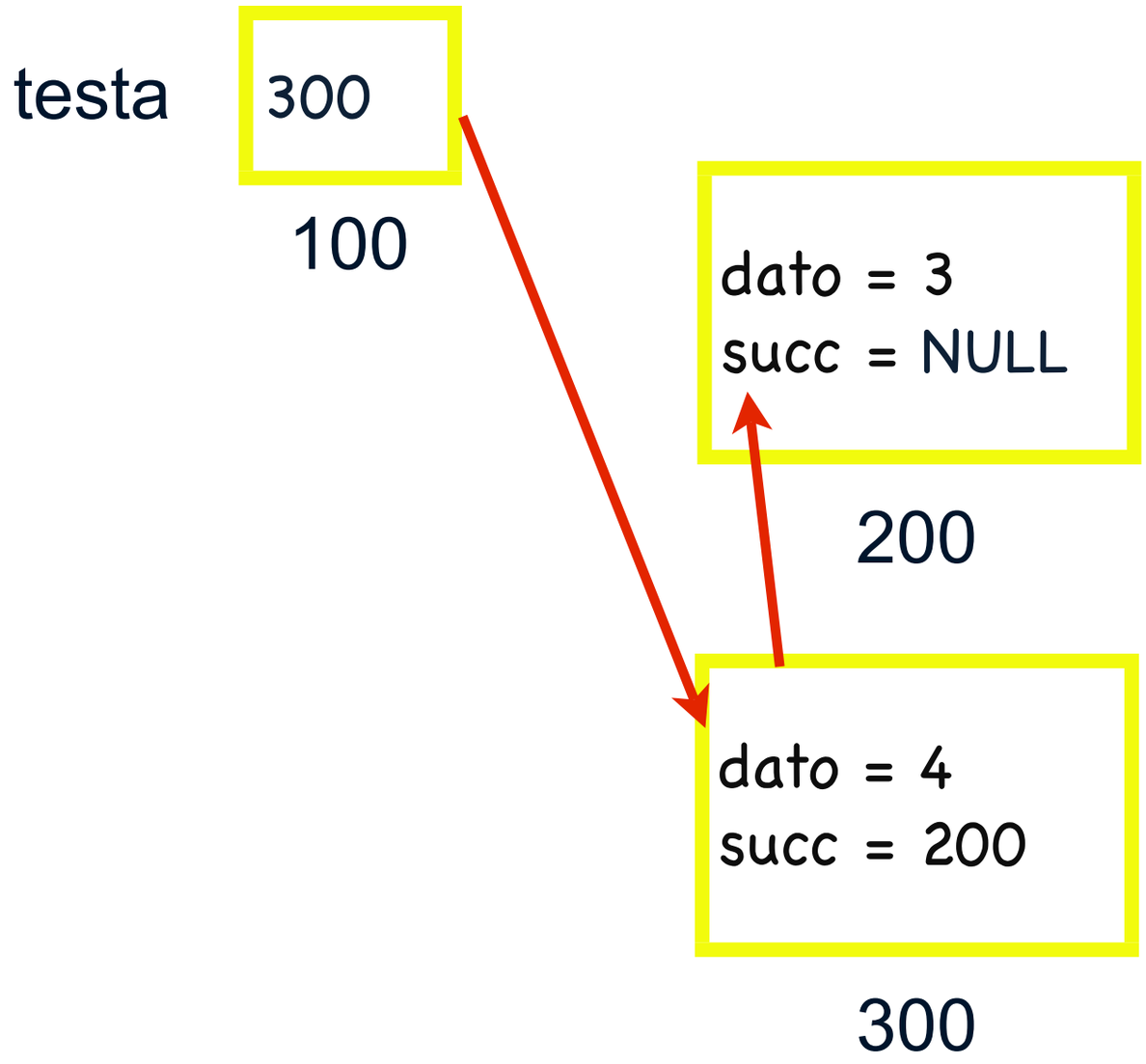


LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

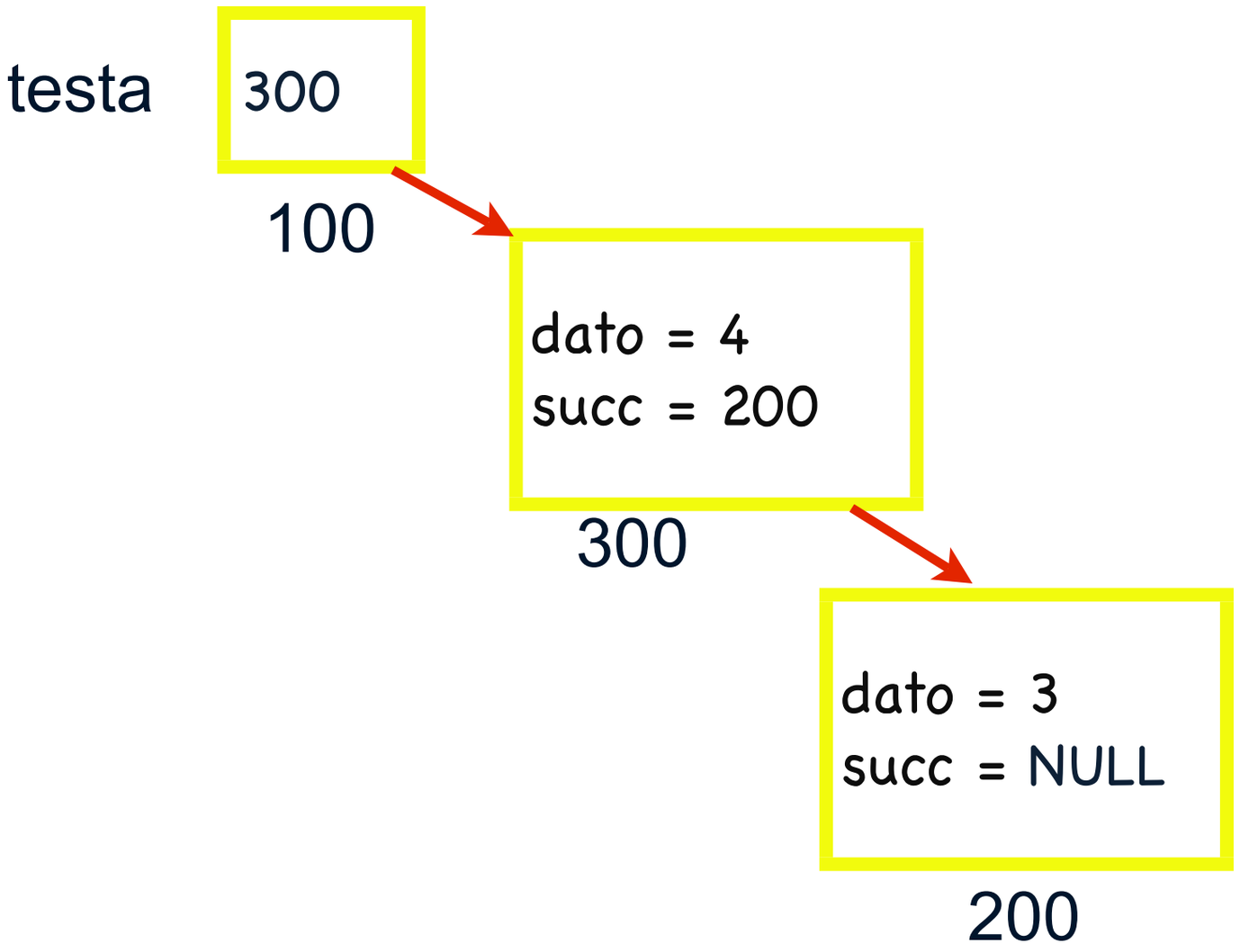


LISTE

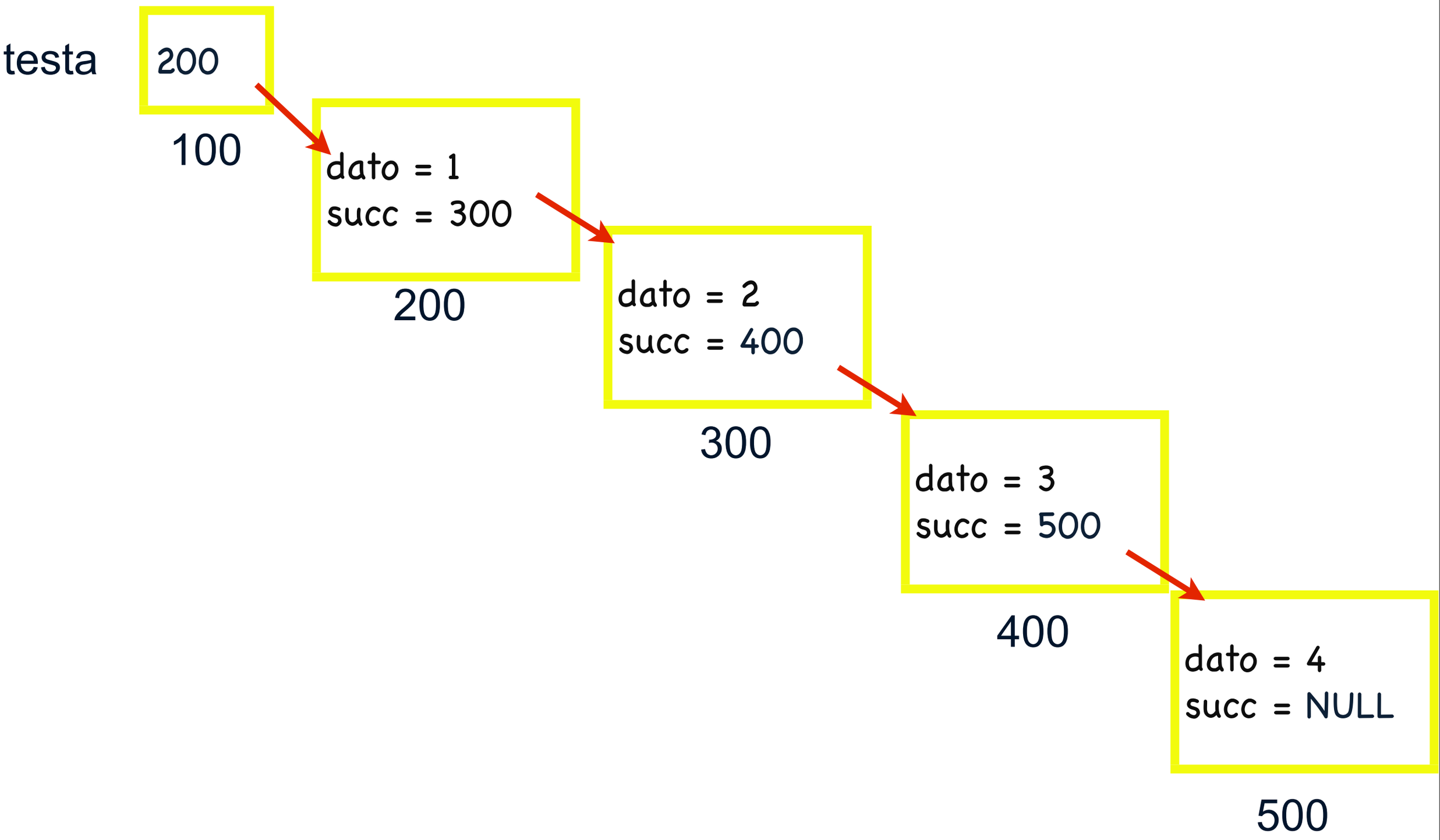
```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
struct nodoLista* testa;
typedef struct nodoLista nl;
```

```
int main
{
   nl* testa = NULL;
   inserisciInTesta(&testa, 3);
   inserisciInTesta(&testa, 4);
}
```

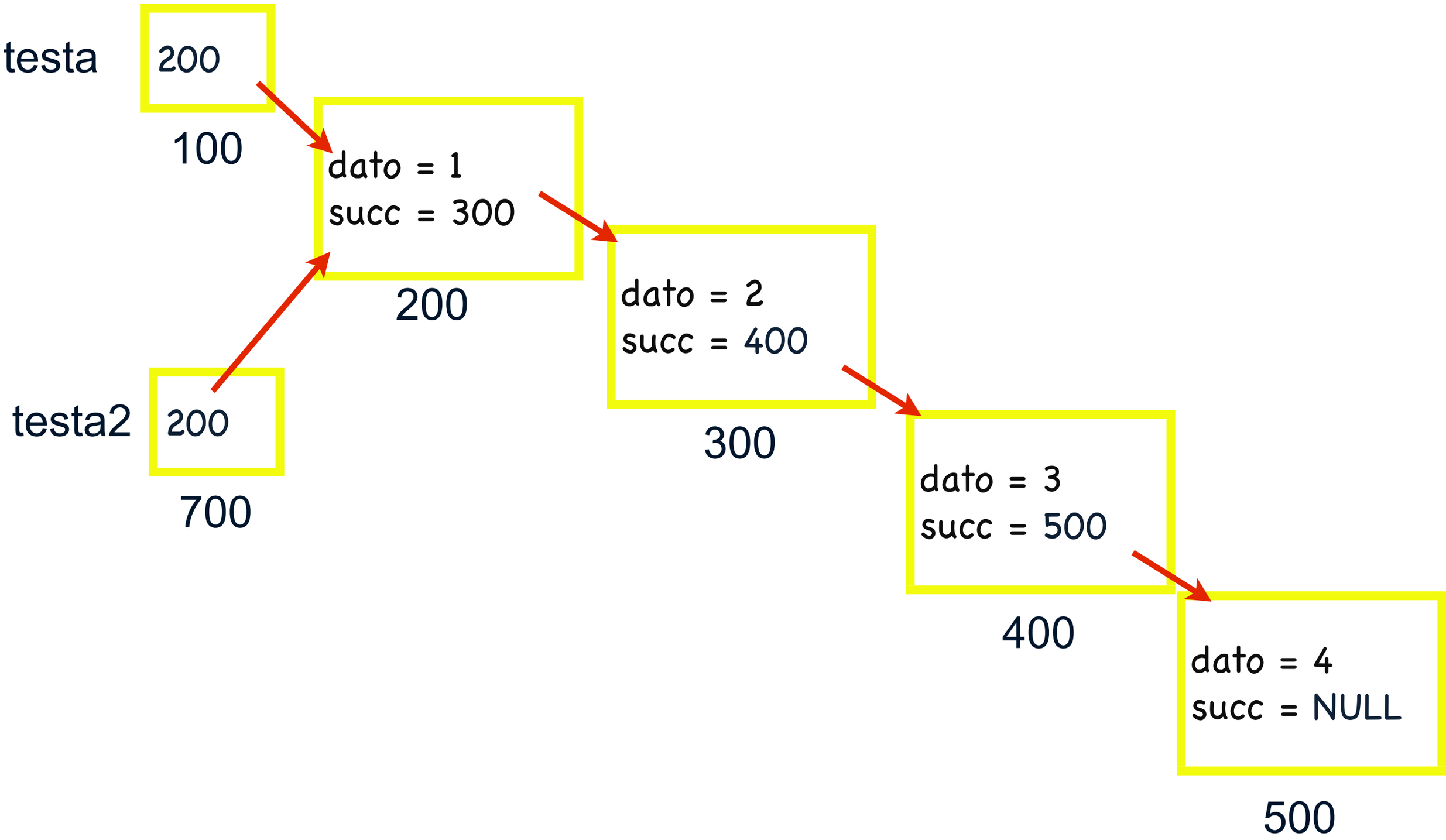
```
void InserisciInTesta(nl** ptesta, int val)
{
   nl* nuovo;
   nuovo=(nl*)malloc(sizeof(nl);
   if (nuovo!=NULL)
      {nuovo->dato=val;
       nuovo->succ=*ptesta;
       *ptesta=nuovo;
      }
}
```



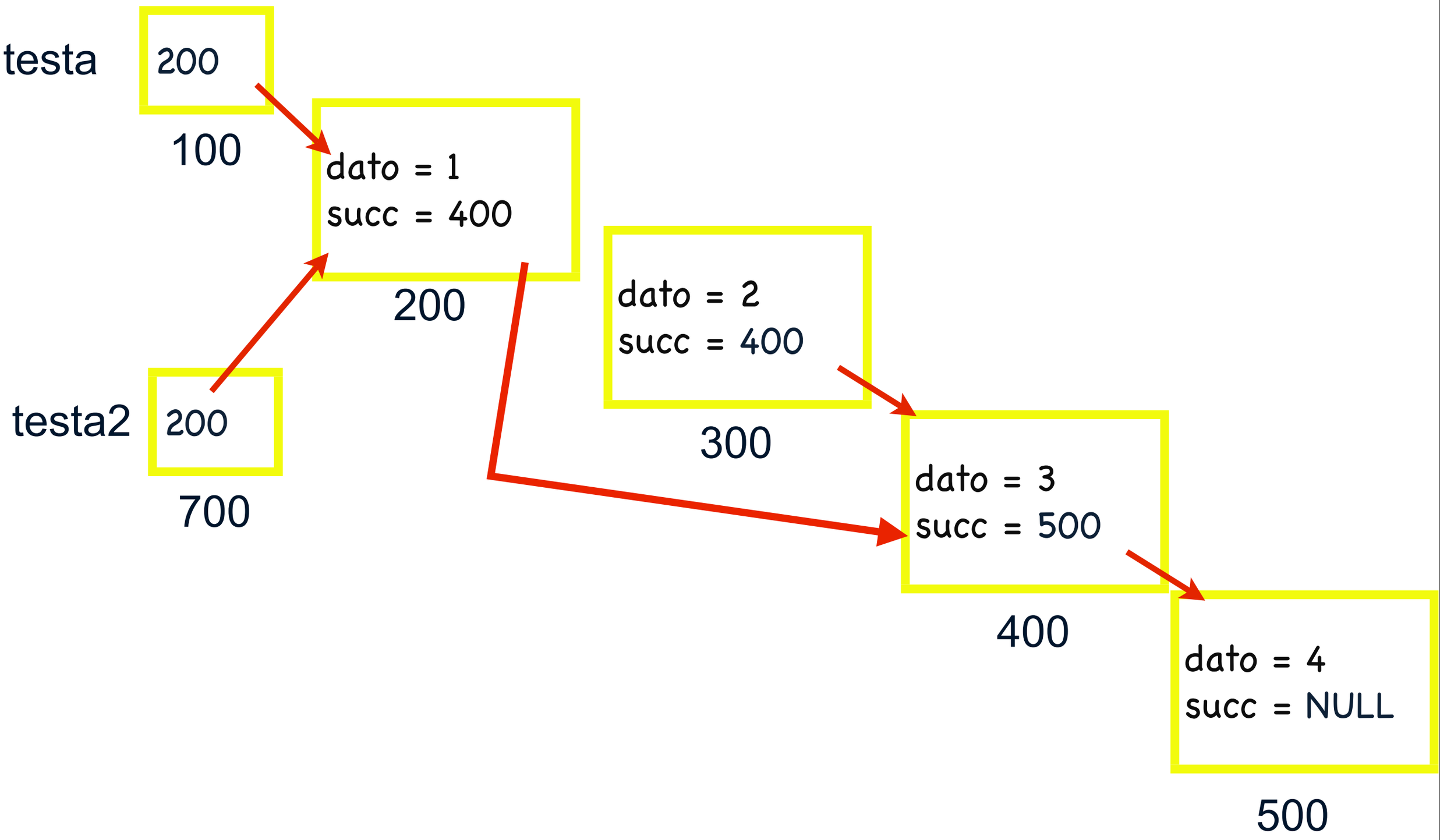
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

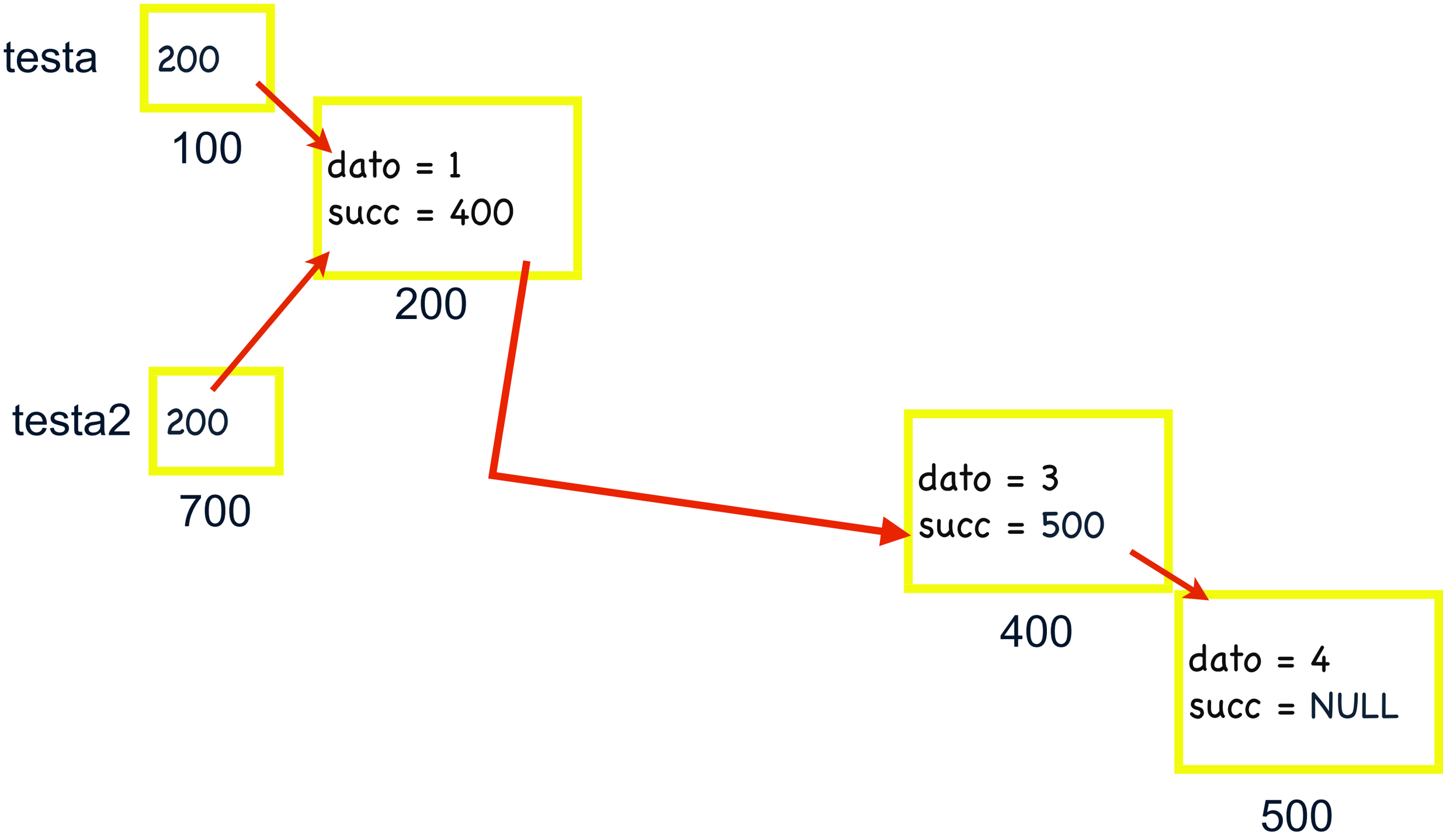
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
dato = 3
succ = 500
400
dato = 4
succ = NULL
500
testa2
200
700

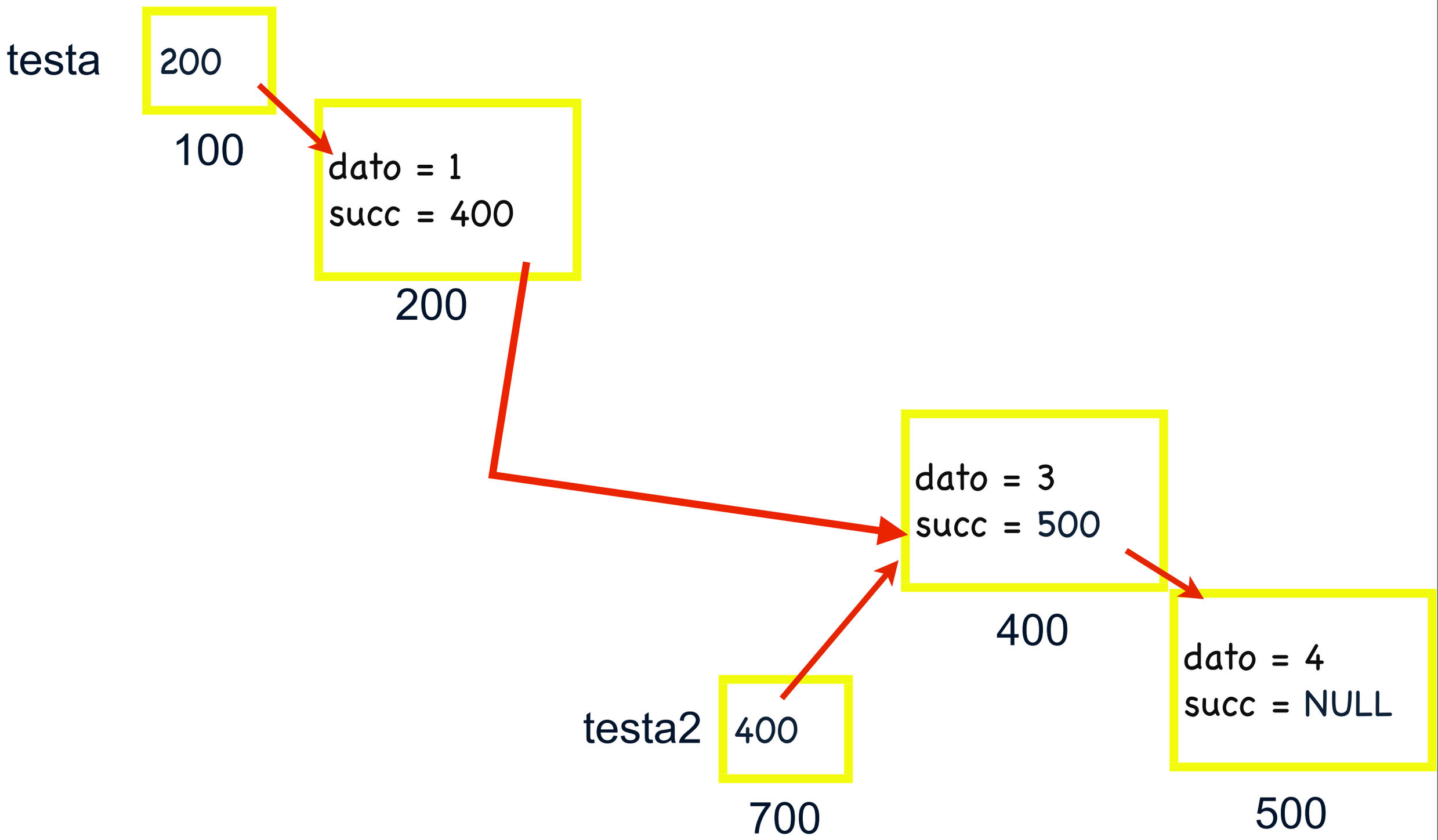
LISTE


testa
200
100
dato = 1
succ = 400
200
dato = 2
succ = 400
300
testa2
200
700
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

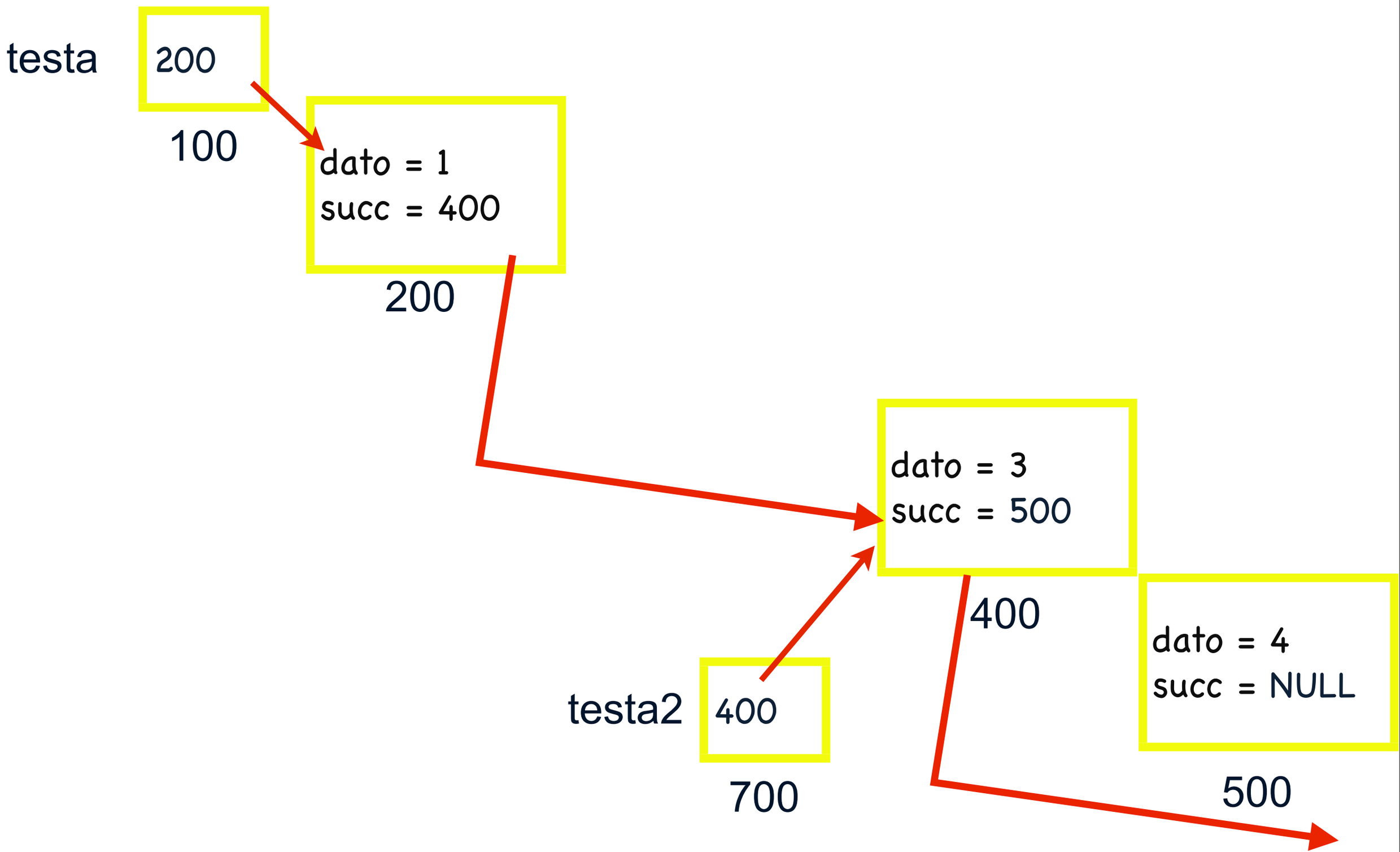
LISTE


testa
200
100
dato = 1
succ = 400
200
testa2
200
700
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

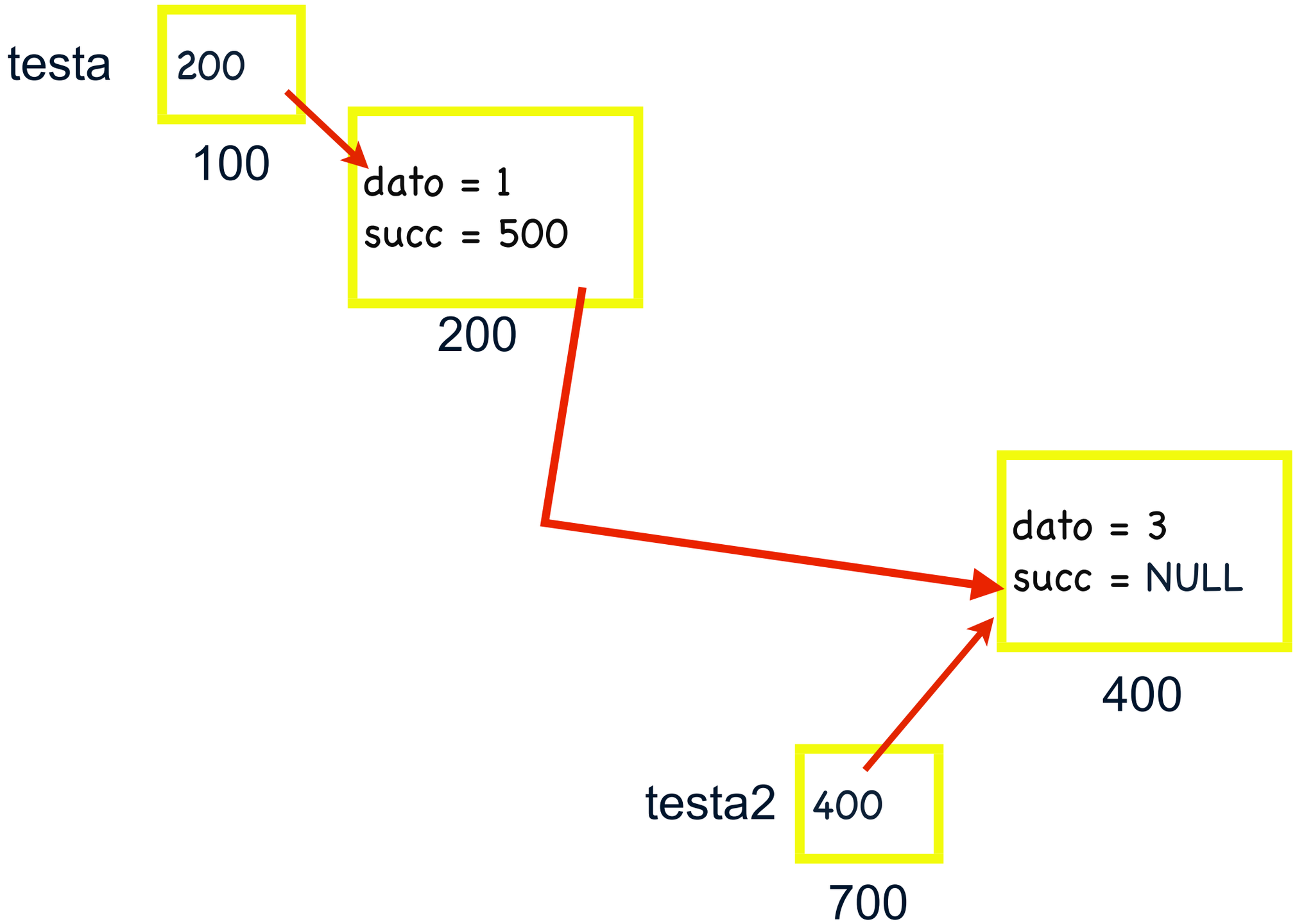
LISTE


testa
200
100
dato = 1
succ = 400
200
dato = 3
succ = 500
400
dato = 4
succ = NULL
500
testa2
400
700

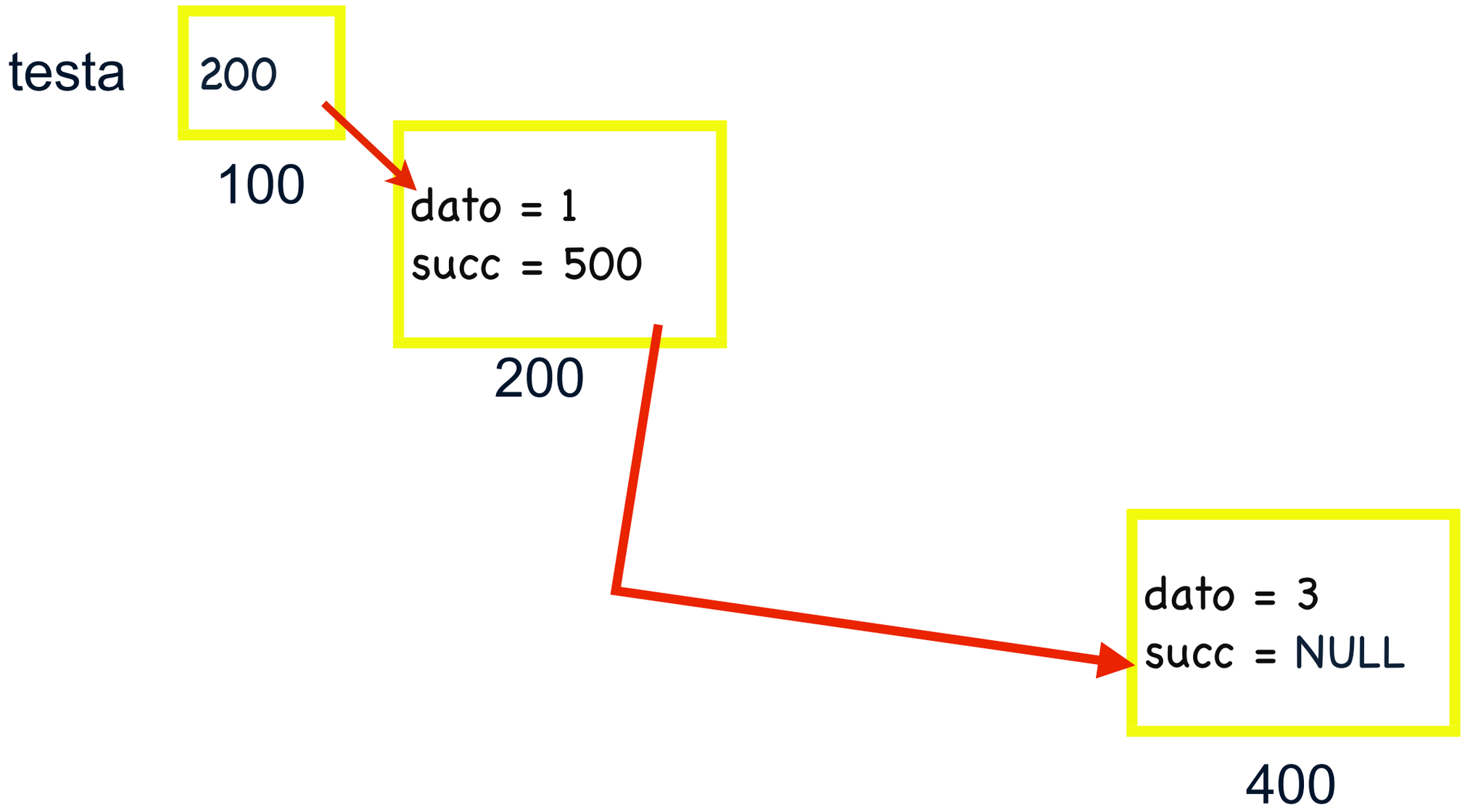
LISTE


testa
200
100
dato = 1
succ = 400
200
dato = 3
succ = 500
400
dato = 4
succ = NULL
500
testa2
400
700

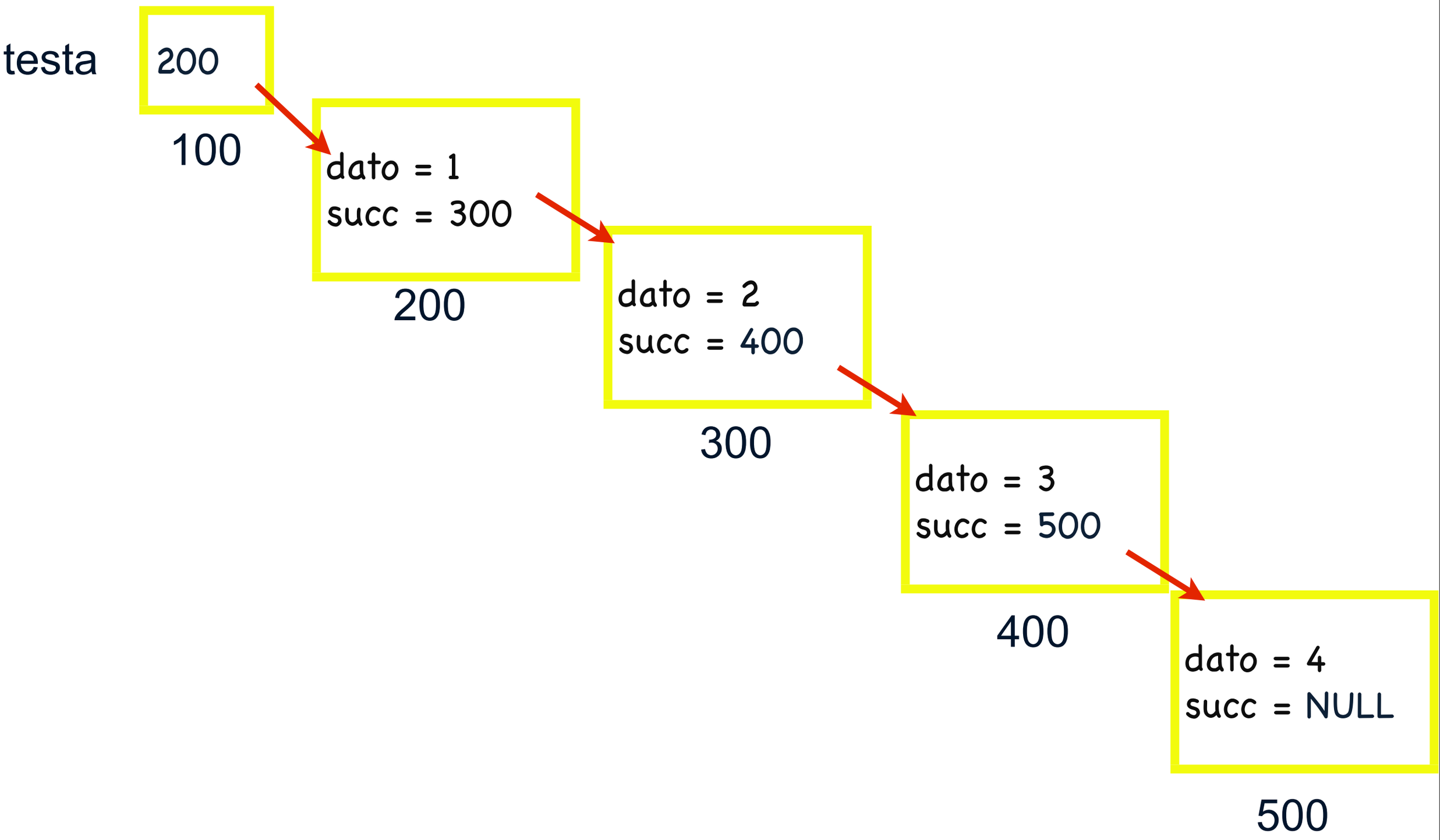
LISTE


testa
200
100
dato = 1
succ = 500
200
dato = 3
succ = NULL
400
testa2
400
700

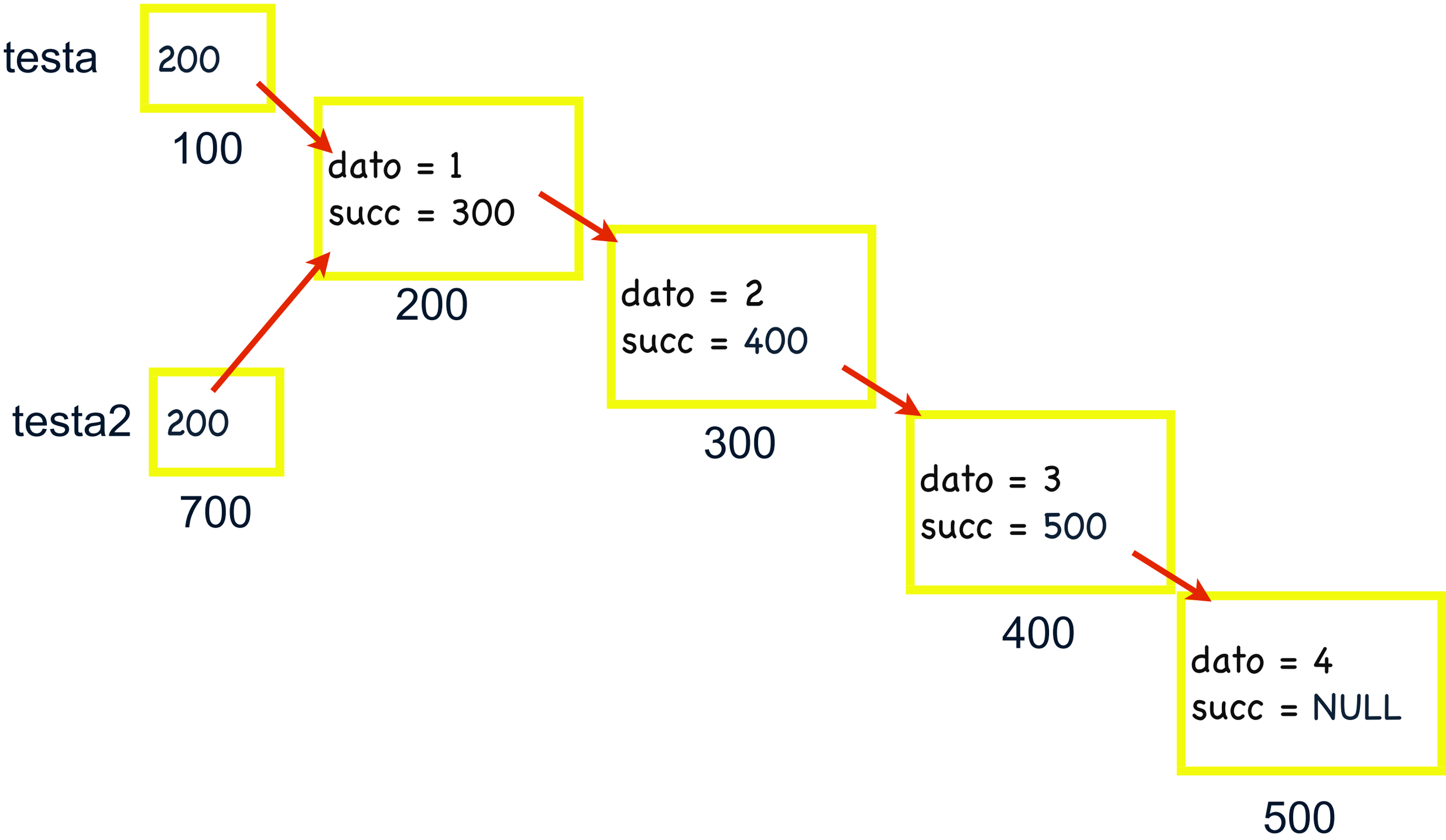
LISTE



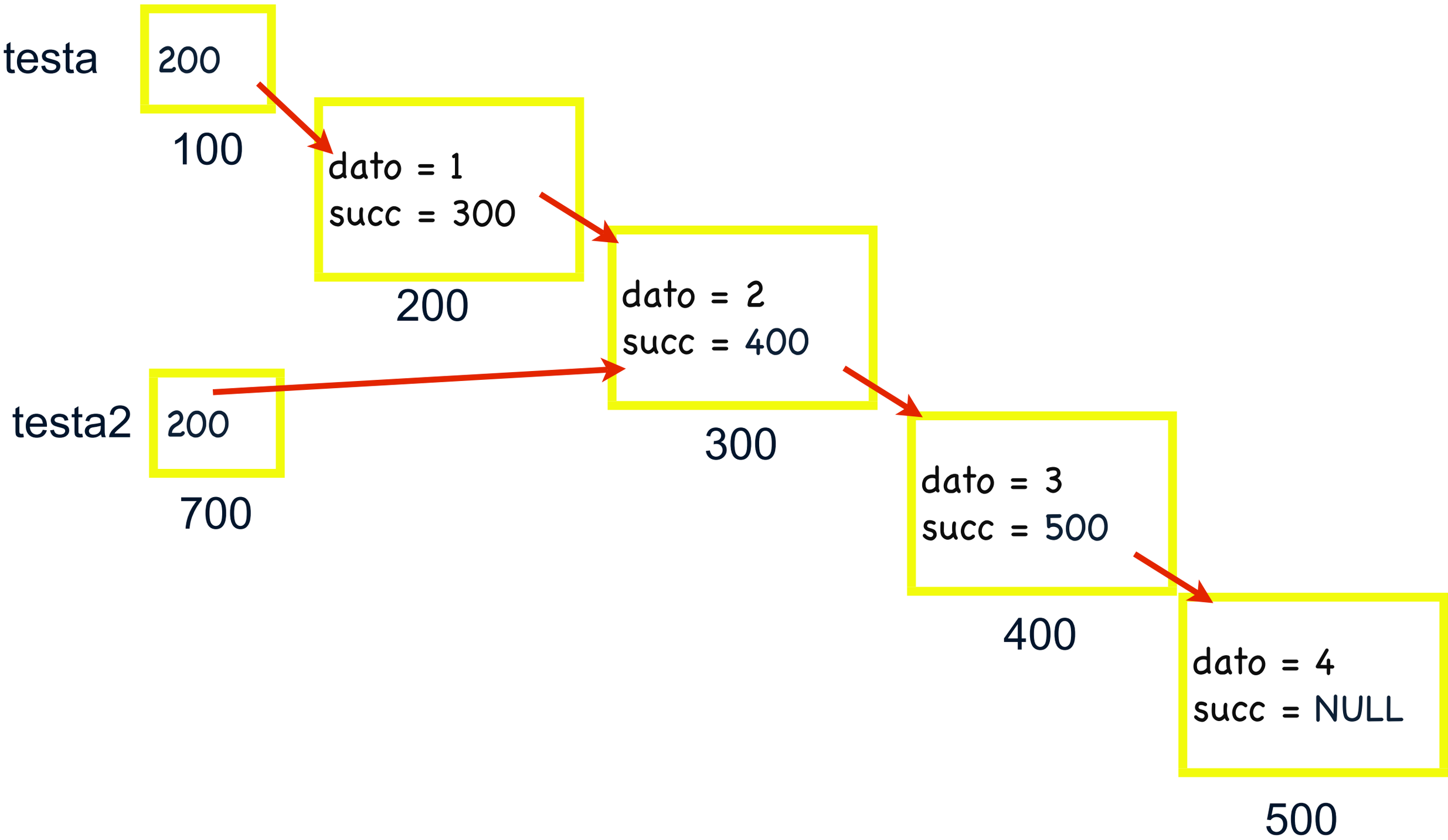
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

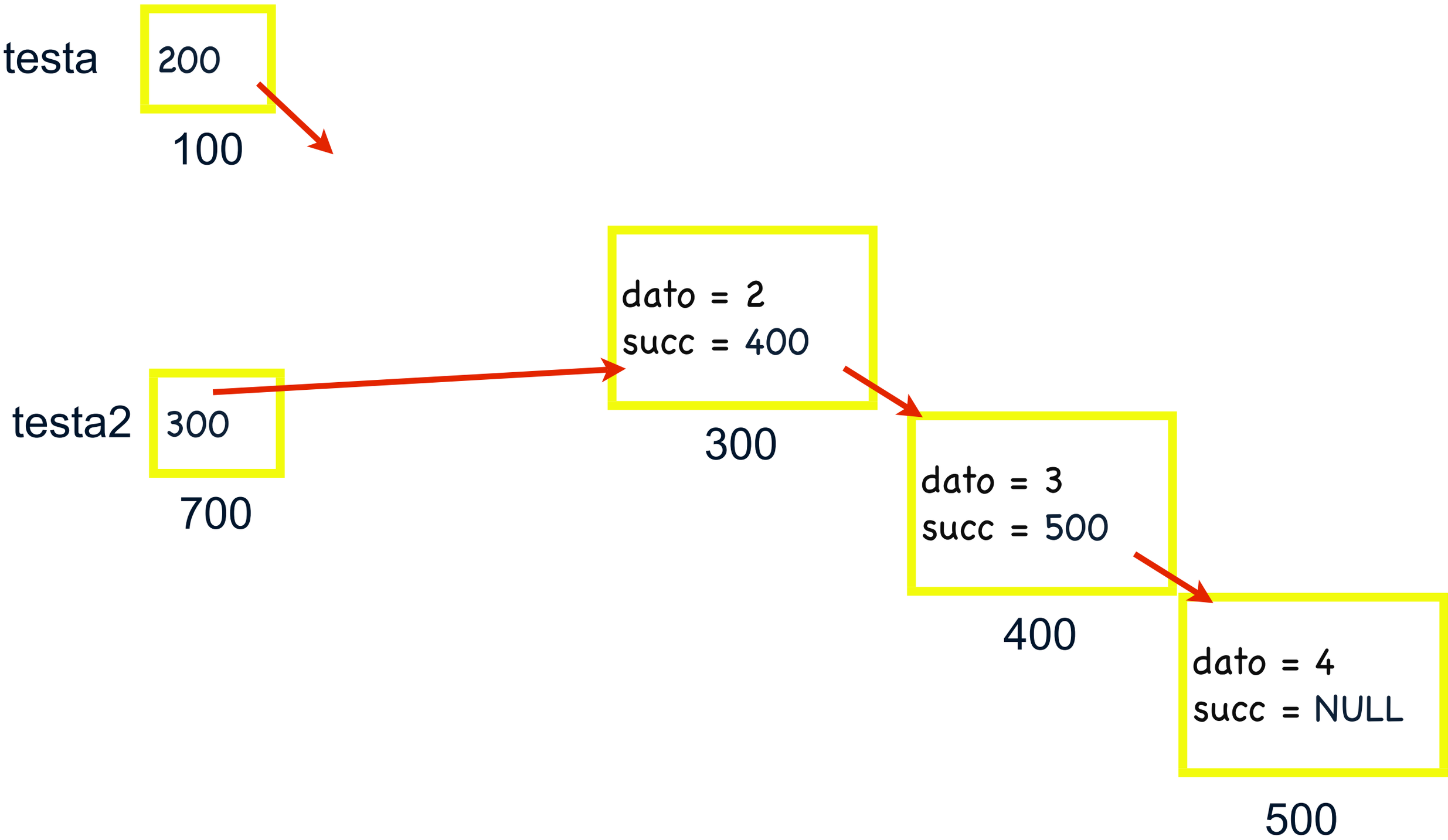
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
dato = 3
succ = 500
400
dato = 4
succ = NULL
500
testa2
200
700

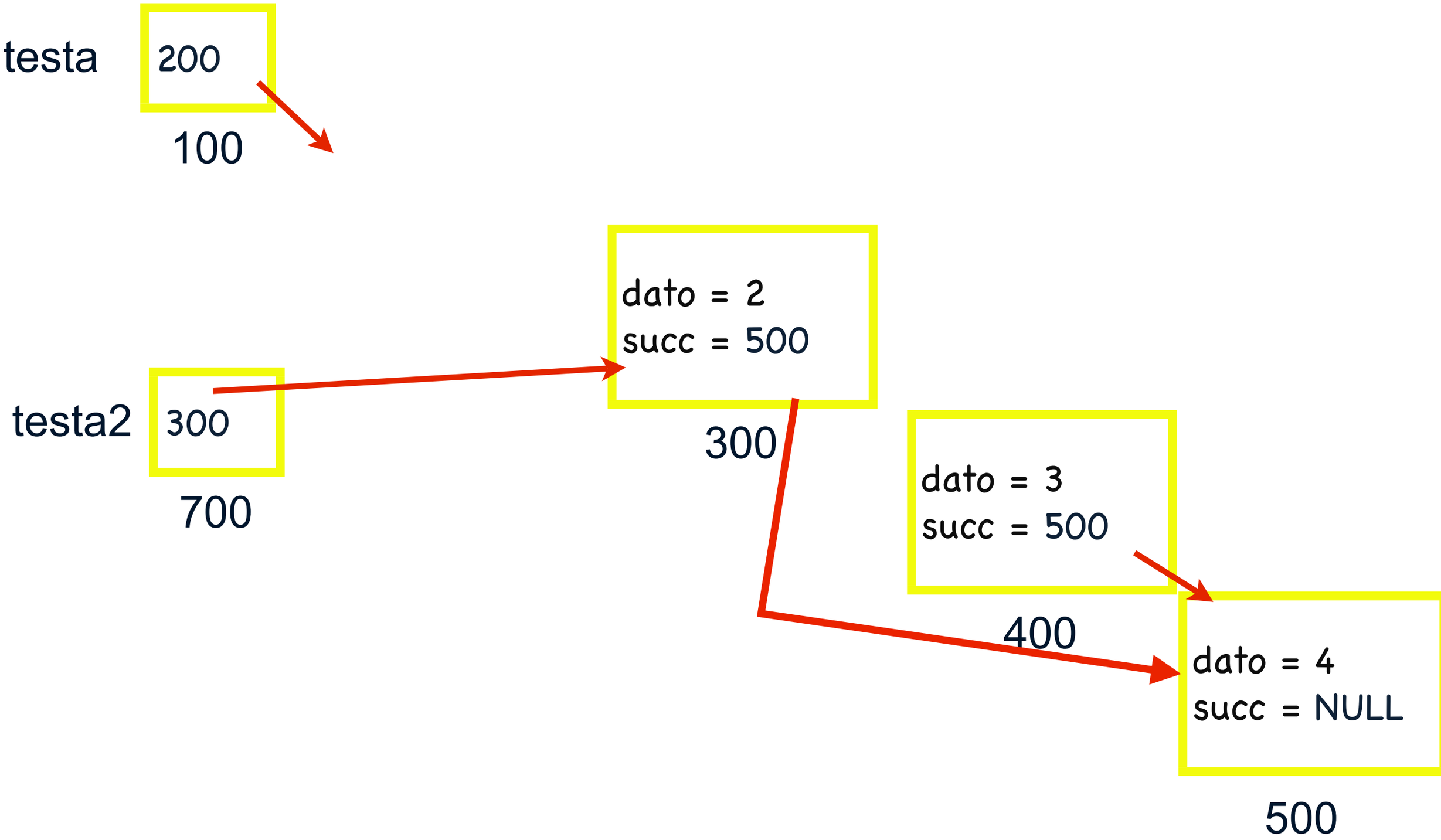
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
testa2
200
700
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

LISTE


testa
200
100
testa2
300
700
data = 2
succ = 400
300
data = 3
succ = 500
400
data = 4
succ = NULL
500

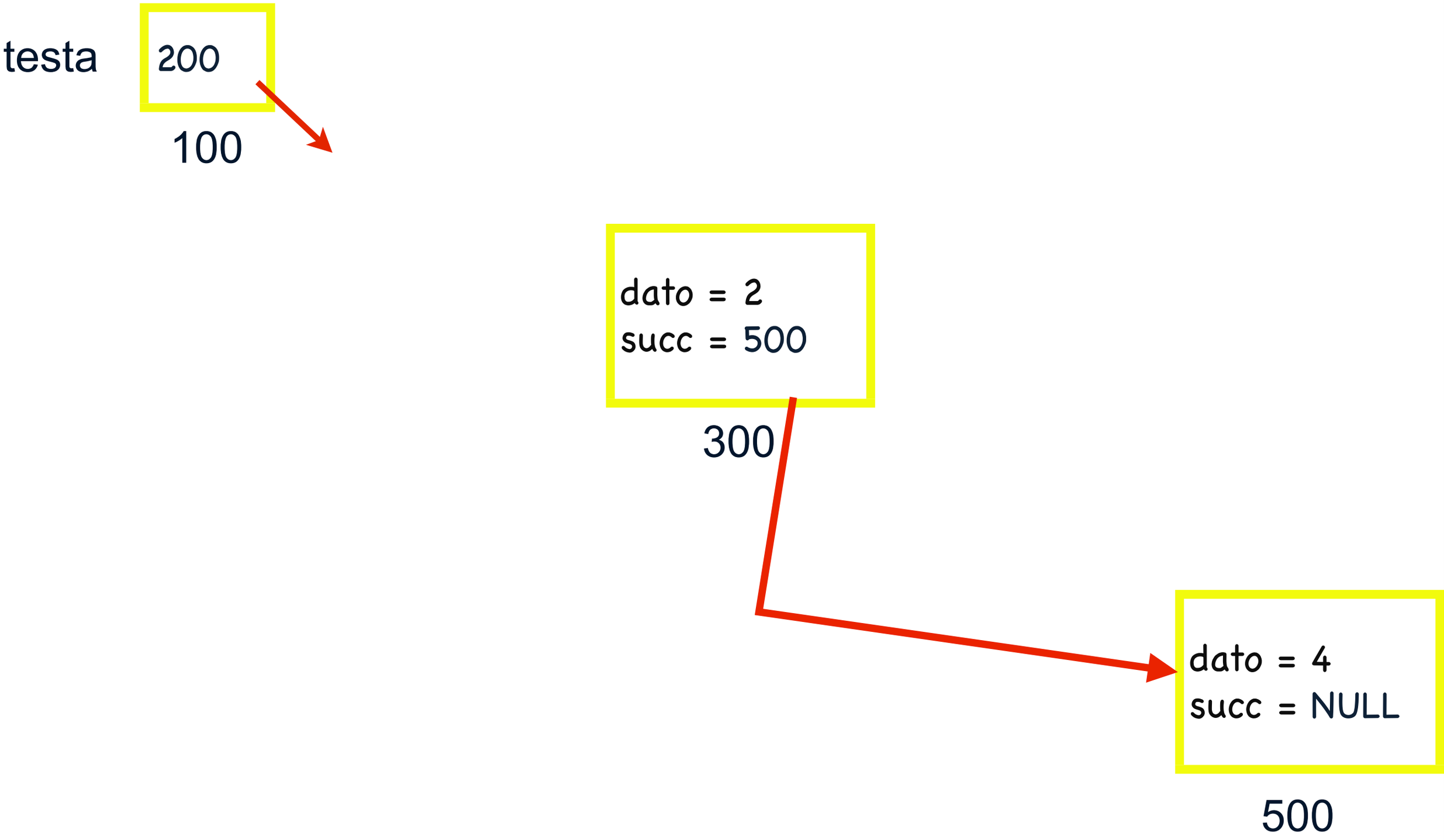
LISTE


testa
200
100
testa2
300
700
data = 2
succ = 500
300
data = 3
succ = 500
400
data = 4
succ = NULL
500

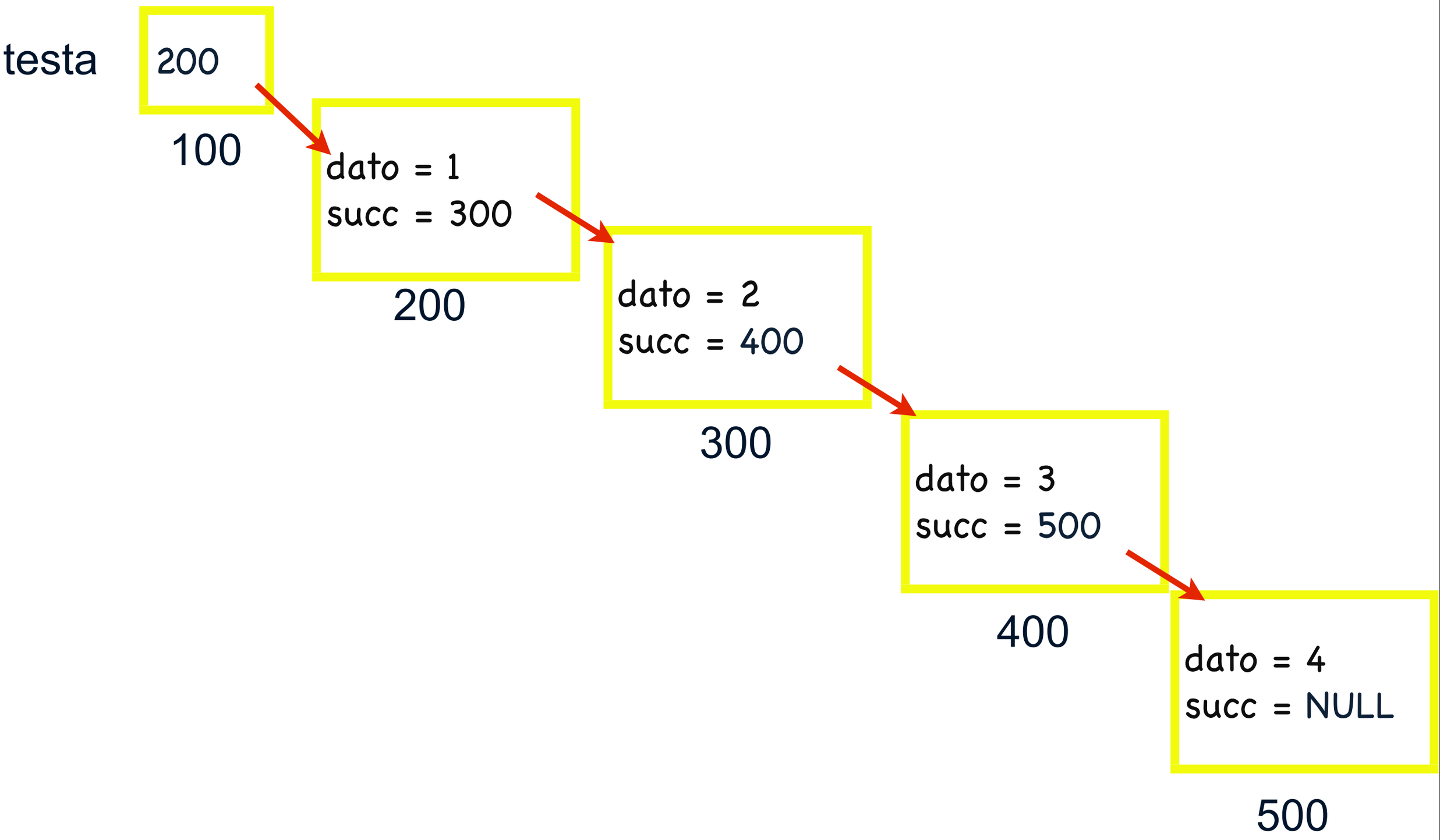
LISTE


testa
200
100
dato = 2
succ = 500
300
dato = 4
succ = NULL
testa2
700
700
500

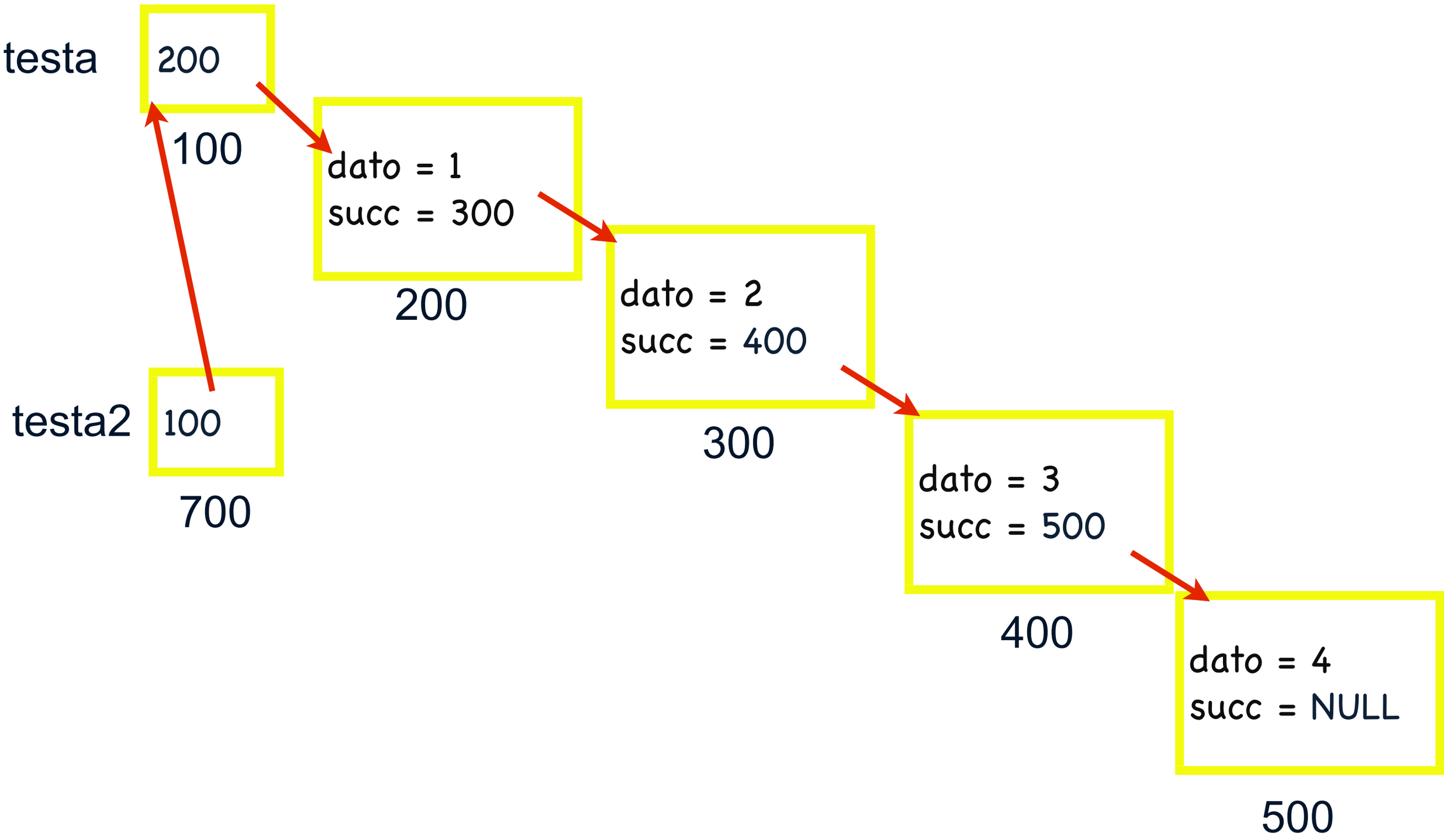
LISTE



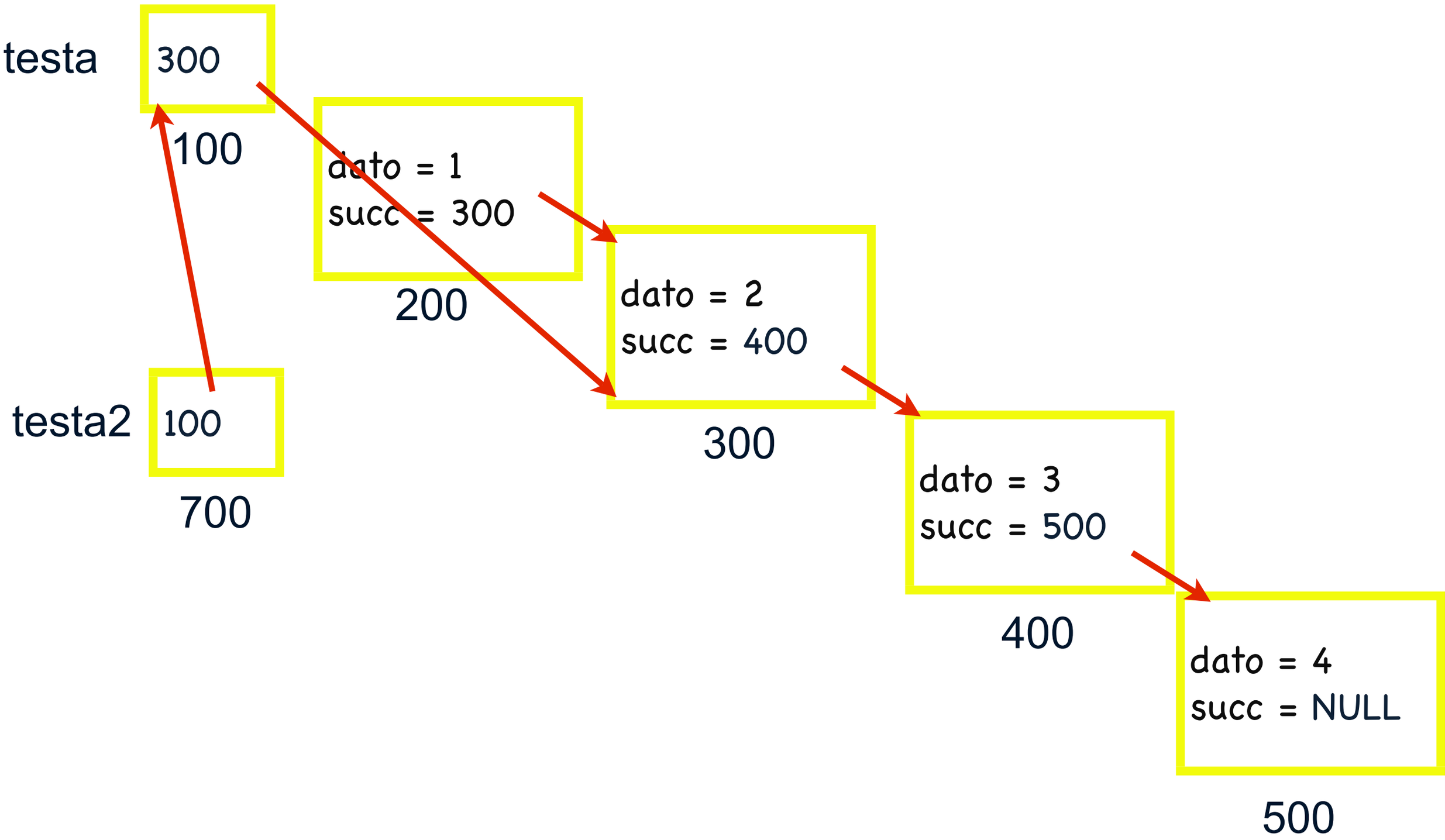
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

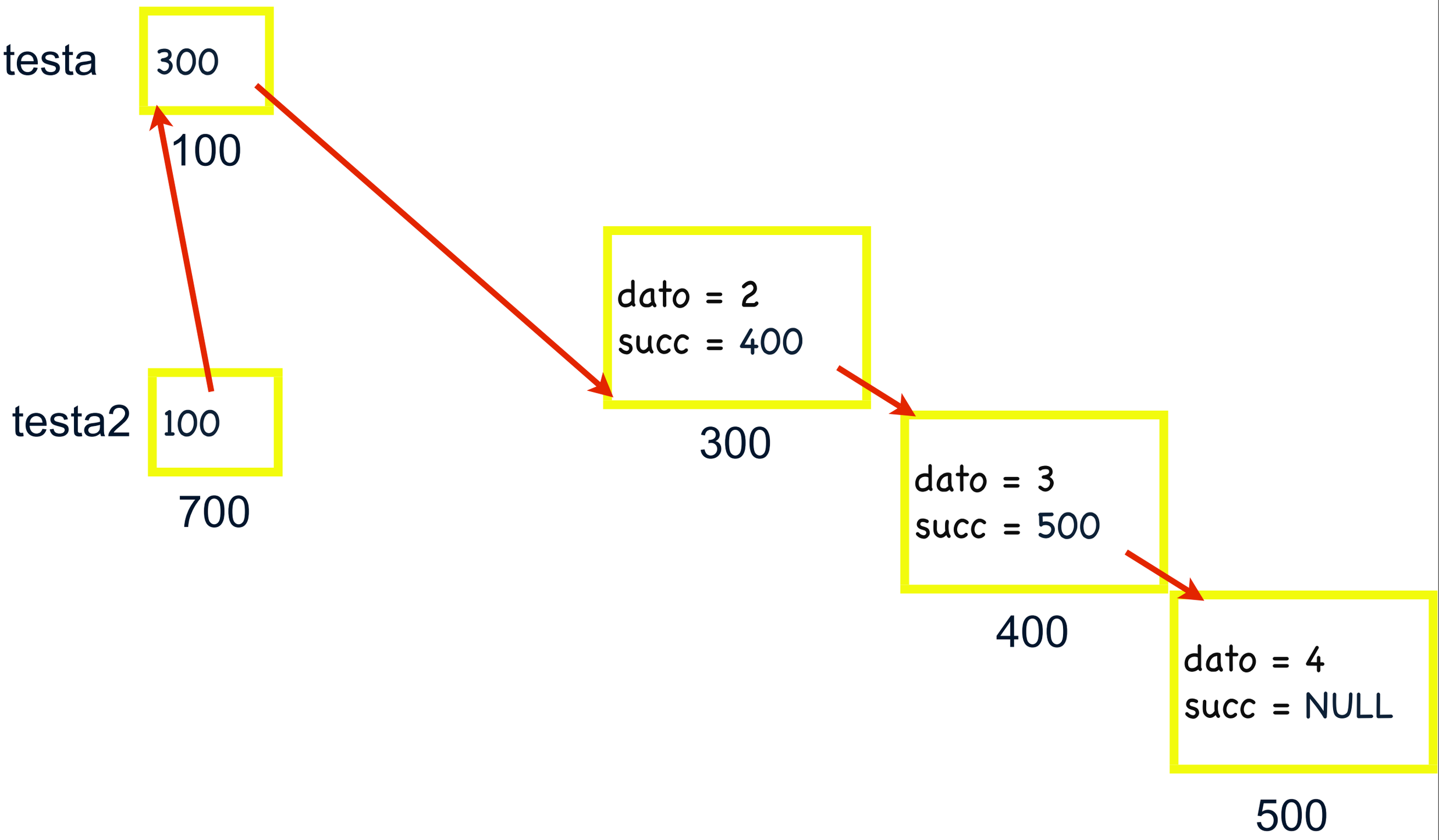
LISTE


testa
200
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
testa2
100
700
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

LISTE


testa
300
100
dato = 1
succ = 300
200
dato = 2
succ = 400
300
testa2
100
700
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

LISTE


testa
300
100
testa2
100
700
dato = 2
succ = 400
300
dato = 3
succ = 500
400
dato = 4
succ = NULL
500

LISTE

```
struct nodoLista
{
   int dato;
   struct nodoLista* succ;
};
typedef struct nodoLista nl;
```

```
int ListaVuota(nl* testa)
{
   return testa==NULL;
}
```

```
void StampaLista (nl* corr)
{
   if (ListaVuota(corr)) {
      printf("lista vuota");
   }
   else  {
         while(corr!=NULL) {
            printf("%d->", corr->dato);
            corr=corr->succ;
         }
   }
}
```

DOMANDE???

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

Esercizi su LISTE

1. Scrivere una funzione che quante volte un intero k compare in una lista di interi

2. Scrivere una funzione che restituisce la media degli elementi di una lista di interi

3. Scrivere una funzione che inverte una lista di interi

TUTTE LE FUNZIONI POSSONO ESSERE IMPLEMENTATE IN MANIERA ITERATIVA E RICORSIVA

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it

BUONE FESTE A TUTTI

17/12/08 - Roberto Petroccia: petroccia@di.uniroma1.it