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 14/01/09

Indice

1. Errori comuni nel quarto homework.
2. Esempi sull'uso di liste in C.
3. Esercizi su strutture e liste.

Esercizio 2

Scrivere una funzione RICORSIVA in linguaggio C in grado di ricevere un vettore di STRUCT di dimensione indefinita. Ogni elemento del vettore contiene una variabile struct con un campo di tipo char, uno di tipo unsigned e uno di tipo long. Se il campo di tipo char contiene il carattere 'p' il valore nel campo di tipo unsigned va considerato positivo; se contiene il carattere 'n', il valore va considerato negativo. Il carattere 't' rappresenta il valore sentinella, che identifica la fine del vettore.

La funzione deve salvare nel campo di tipo long dell'elemento i il valore della somma parziale dei campi di tipo unsigned dei primi i + 1 elementi (considerando il segno definito dal carattere 'p' o 'n'), fino al raggiungimento dell'elemento contenente il carattere terminatore 't'.

14/01/09 - Roberto Petroccia: petroccia@di.uniroma1.it

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
    c=getchar();
    v[i].sign = c;
    if(c=='t') {
        break;
    }
    scanf("%u", &(v[i].val));
    i++;
}
 prefix_sum(v,0);
```

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
    c=getchar();
    v[i].sign = c;
    if(c=='t') {
        break;
    }
    scanf("%u", &(v[i].val));
    i++;
}
 prefix_sum(v,0);
```

CORRETTA???

Soluzione Ex. 2

ERRATA

```
int i=0;
char c;
Element v[LIMIT];

while(1){
    c=getchar();
    v[i].sign = c;
    if(c=='t') {
        break;
    }
    scanf("%u", &(v[i].val));
    i++;
}
 prefix_sum(v,0);
```

Soluzione Ex. 2

ERRATA

```
int i=0;
char c;
Element v[LIMIT];

while(1){
    c=getchar();
    v[i].sign = c;
    if(c=='t') {
        break;
    }
    scanf("%u", &(v[i].val));
    i++;
}
 prefix_sum(v,0);
```

```
printf("char c = %c val = %u\n", V[i].sign, V[i].val);
```

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
   c=getchar();
   v[i].sign = c;
   if(c=='t') {
       break;
   }
   scanf("%u", &(v[i].val));
   i++;
}
 prefix_sum(v,0);
```

ERRATA

p 10 n 10 p 4 p 3 n 5 t

```
printf("char c = %c val = %u\n", V[i].sign, V[i].val);
```

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
    c=getchar();
    v[i].sign = c;
    if(c=='t') {
        break;
    }
    scanf("%u", &(v[i].val));
    printf("char c = %c val = %u\n", v[i].sign, v[i].val);
    i++;
}
 prefix_sum(v,0);
```

ERRATO

p 10 n 10 p 4 p 3 n 5 t

```
char c = p   val = 10
char c =     val = 2417771444
char c = n   val = 10
char c =     val = 2417771796
char c = p   val = 4
char c =     val = 0
char c = p   val = 3
char c =     val = 2413891013
char c = n   val = 5
char c =     val = 2417755143
```

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
   c=getchar();
   while(c=='\n'||c==' '||c=='\t')
      c=getchar();

   v[i].sign = c;

   if(c=='t') break;
   scanf("%u", &(v[i].val));

   i++;
}
 prefix_sum(v,0);
```

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
   c=getchar();
   while(c=='\n'||c==' '||c=='\t')
      c=getchar();

   v[i].sign = c;

   if(c=='t') break;
   scanf("%u", &(v[i].val));

   i++;
}
 prefix_sum(v,0);
```

p 10 n 10 p 4 p 3 n 5 t

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
   c=getchar();
   while(c=='\n'||c==' '||c=='\t')
      c=getchar();

   v[i].sign = c;

   if(c=='t') break;
   scanf("%u", &(v[i].val));

   i++;
}
 prefix_sum(v,0);
```

p 10 n 10 p 4 p 3 n 5 t

char c = p val = 10
char c = n val = 10
char c = p val = 4
char c = p val = 3
char c = n val = 5

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
    c=getchar();
    if(c=='p' || c == 'n') {
        v[i].sign = c;
        scanf("%u", &(v[i].val));
        i++;
    }
    if(c=='t') {
         break;
    }
}
 prefix_sum(v,0);
```

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
   c=getchar();
   if(c=='p' || c == 'n') {
      v[i].sign = c;
      scanf("%u", &(v[i].val));
      i++;
   }
   if(c=='t') {
       break;
   }
}
 prefix_sum(v,0);
```

p 10 n 10 p 4 p 3 n 5 t

Soluzione Ex. 2

```
int i=0;
char c;
Element v[LIMIT];

while(1){
   c=getchar();
   if(c=='p' || c == 'n') {
      v[i].sign = c;
      scanf("%u", &(v[i].val));
      i++;
   }
   if(c=='t') {
       break;
   }
}
 prefix_sum(v,0);
```

p 10 n 10 p 4 p 3 n 5 t

char c = p val = 10
char c = n val = 10
char c = p val = 4
char c = p val = 3
char c = n val = 5

Soluzione Ex. 2

```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum -= v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

Soluzione Ex. 2

```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum -= v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```

Soluzione Ex. 2

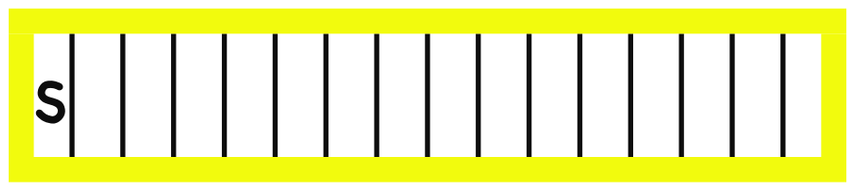
```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum += -1 * v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```

Soluzione Ex. 2

```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum += -1 * v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

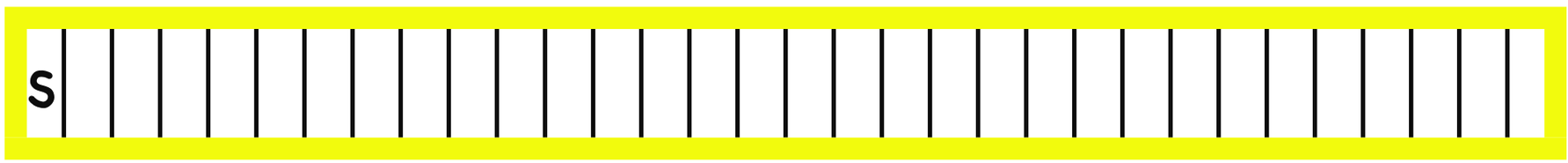
```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```

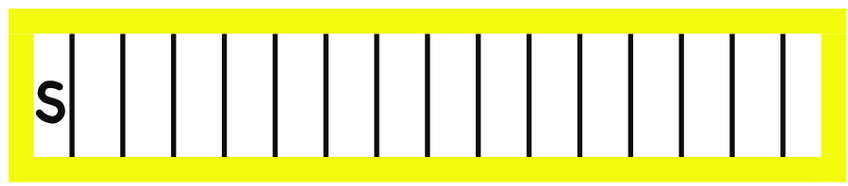


Soluzione Ex. 2

```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum += -1 * v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```

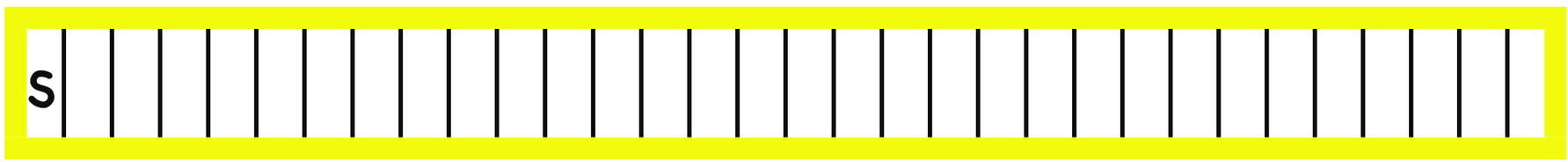




Soluzione Ex. 2

```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum += -1 * v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```



Soluzione Ex. 2

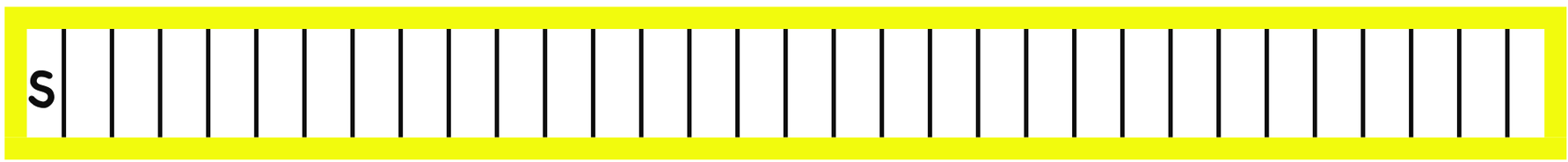
```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum += (long) (-1 * v[i].val);
            break;
    }

    prefix_sum(v, ++i);

}
```

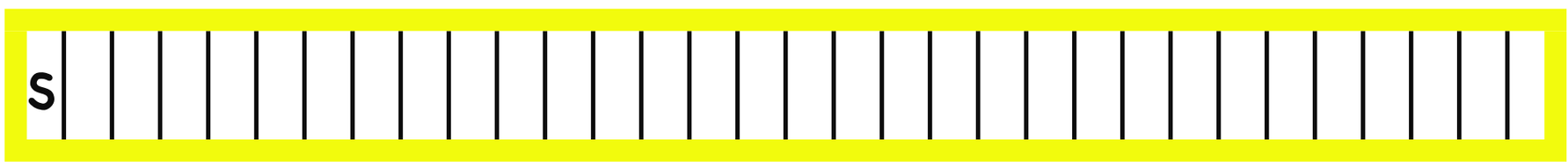
```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```

Soluzione Ex. 2

```
void prefix_sum (Element v[], int i){

    if(v[i].sign == 't') return;

    if(i>0) v[i].prefix_sum = v[i-1].prefix_sum;
    else v[i].prefix_sum=0;

    switch (v[i].sign) {
        case 'p':
            v[i].prefix_sum += v[i].val;
            break;
        case 'n':
            v[i].prefix_sum += -1 * (long)v[i].val;
            break;
    }

    prefix_sum(v, ++i);

}
```

```
typedef struct element {
    char sign;
    unsigned val;
    long prefix_sum;
} Element;
```





DOMANDE???

14/01/09 - Roberto Petroccia: petroccia@di.uniroma1.it

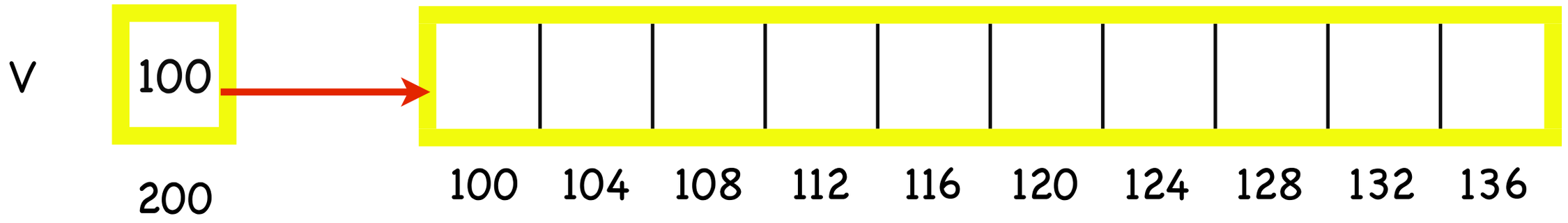
Allocazione memoria

int V[10];

int * V;
V = (int*) malloc (sizeof(int) * 10);

Allocazione memoria

int V[10];

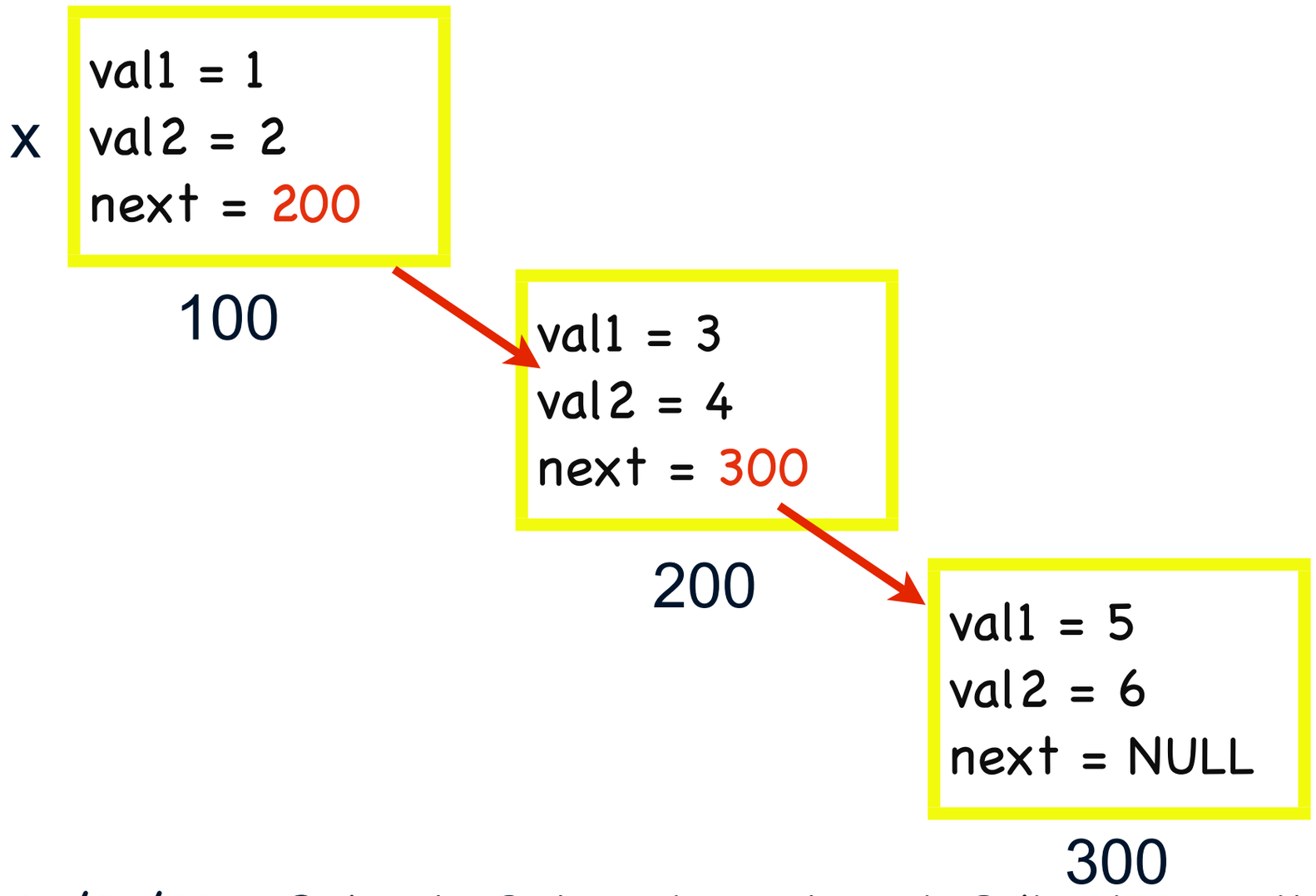



```
int * V;
V = (int*) malloc (sizeof(int) * 10);
```

Allocazione memoria

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

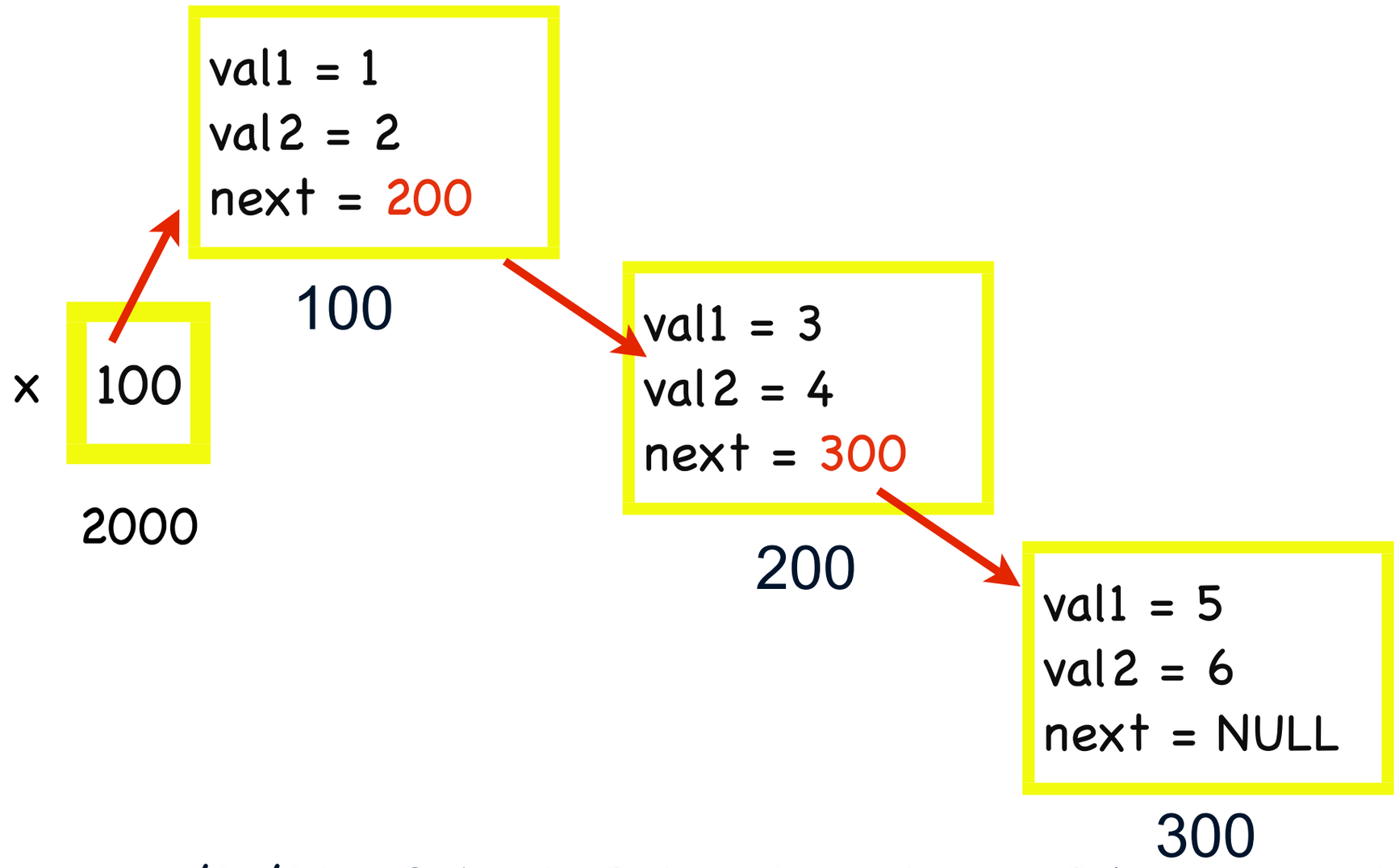
coppia x;



Allocazione memoria

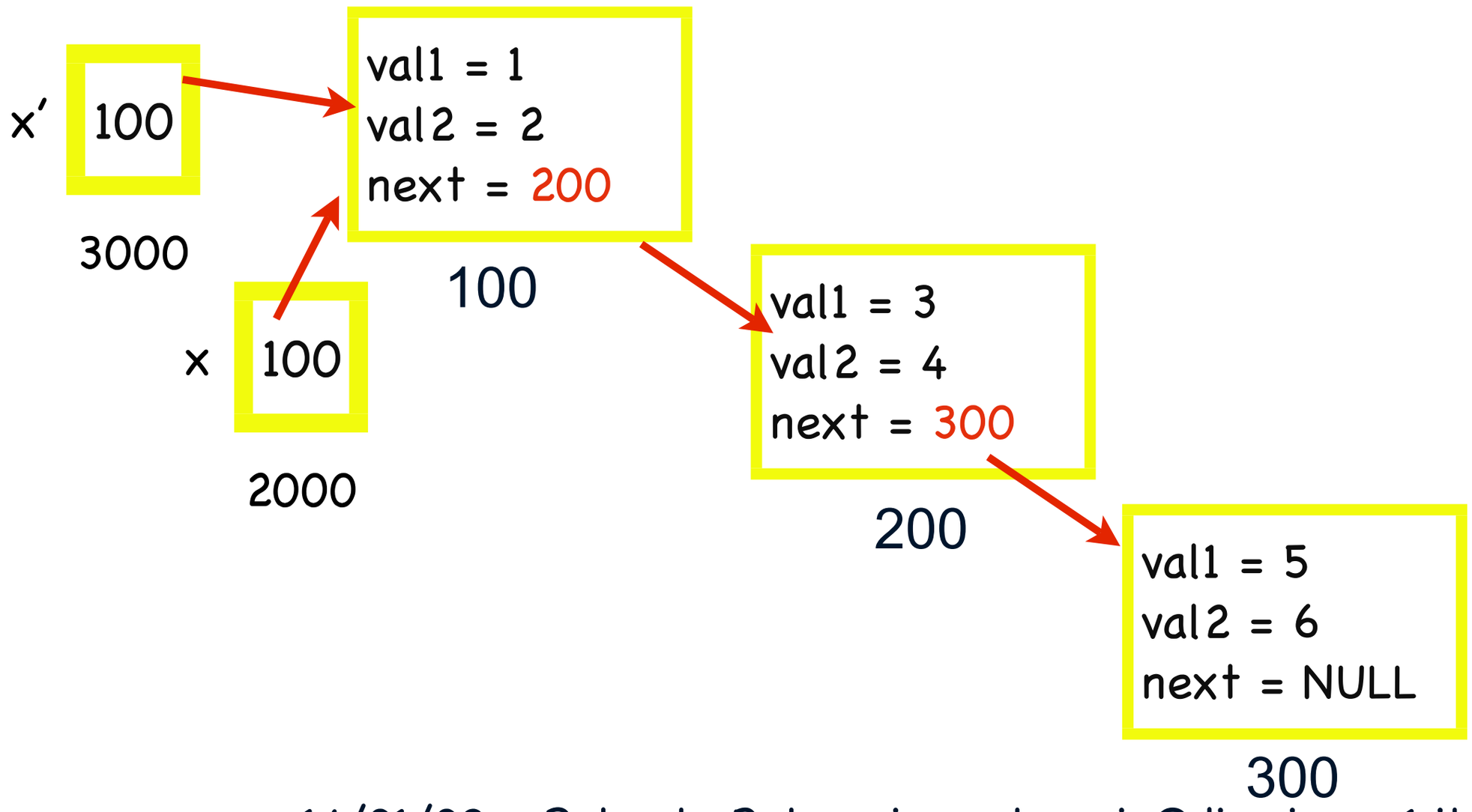
```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```

coppia* x;



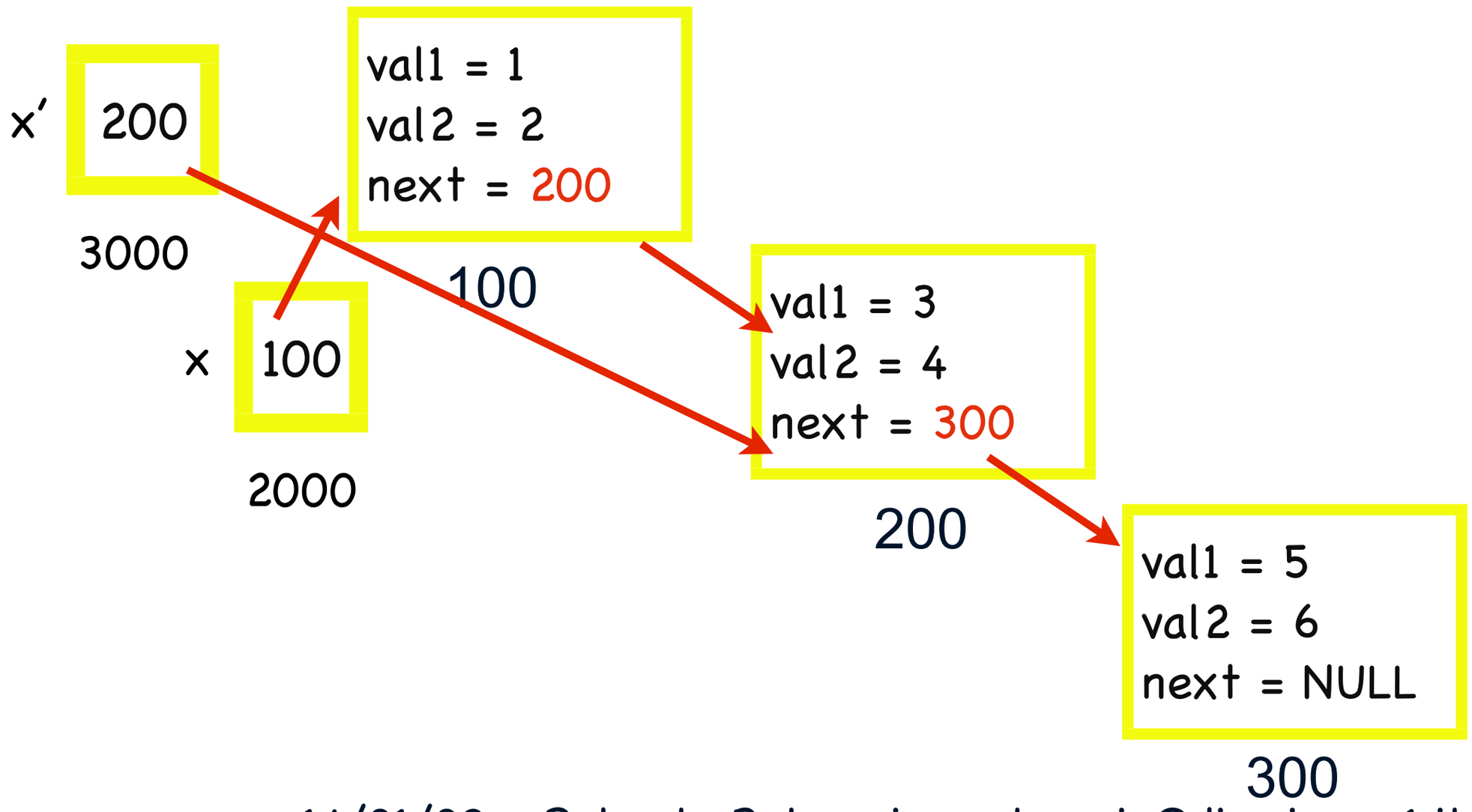
Allocazione memoria

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



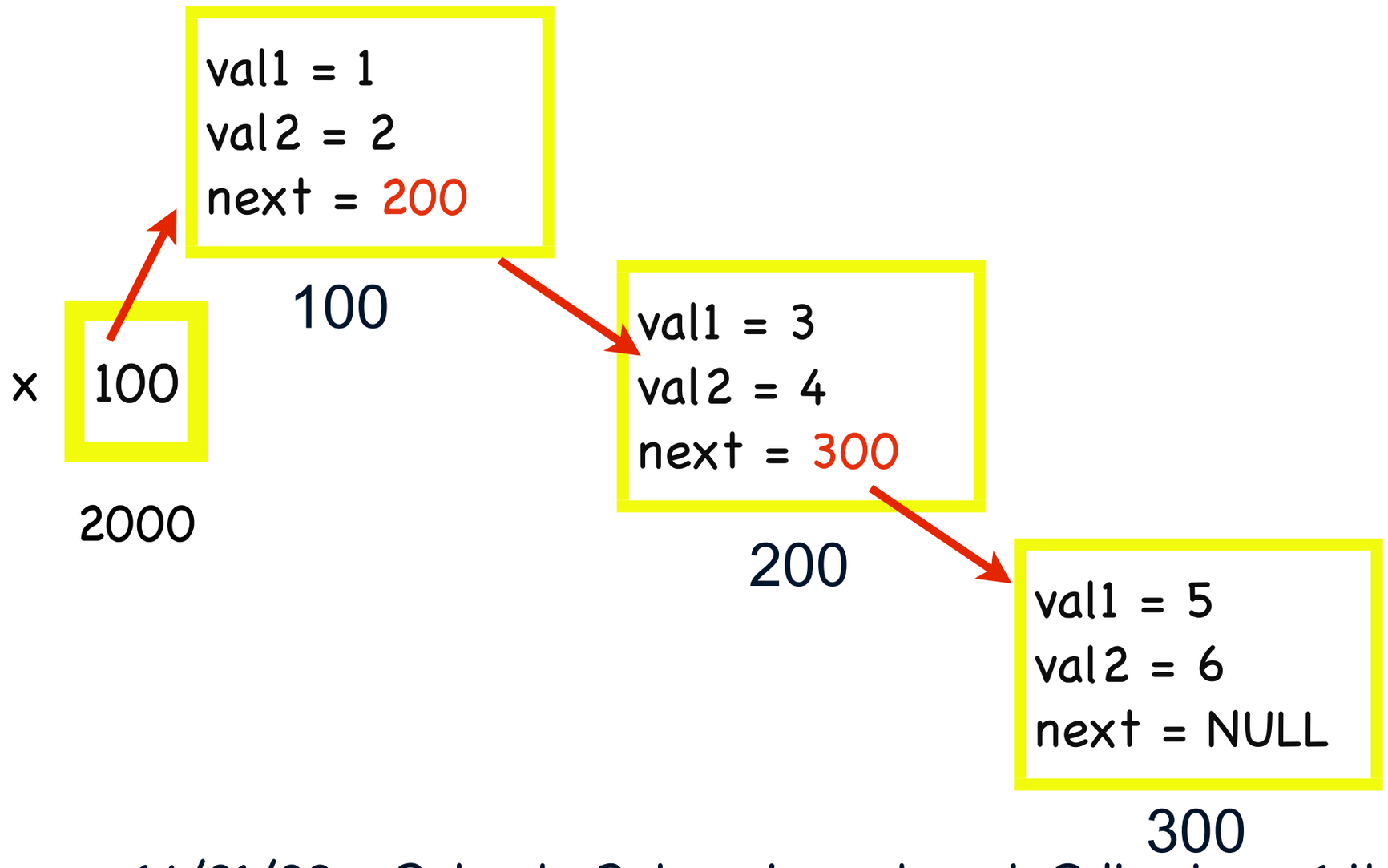
Allocazione memoria

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



Allocazione memoria

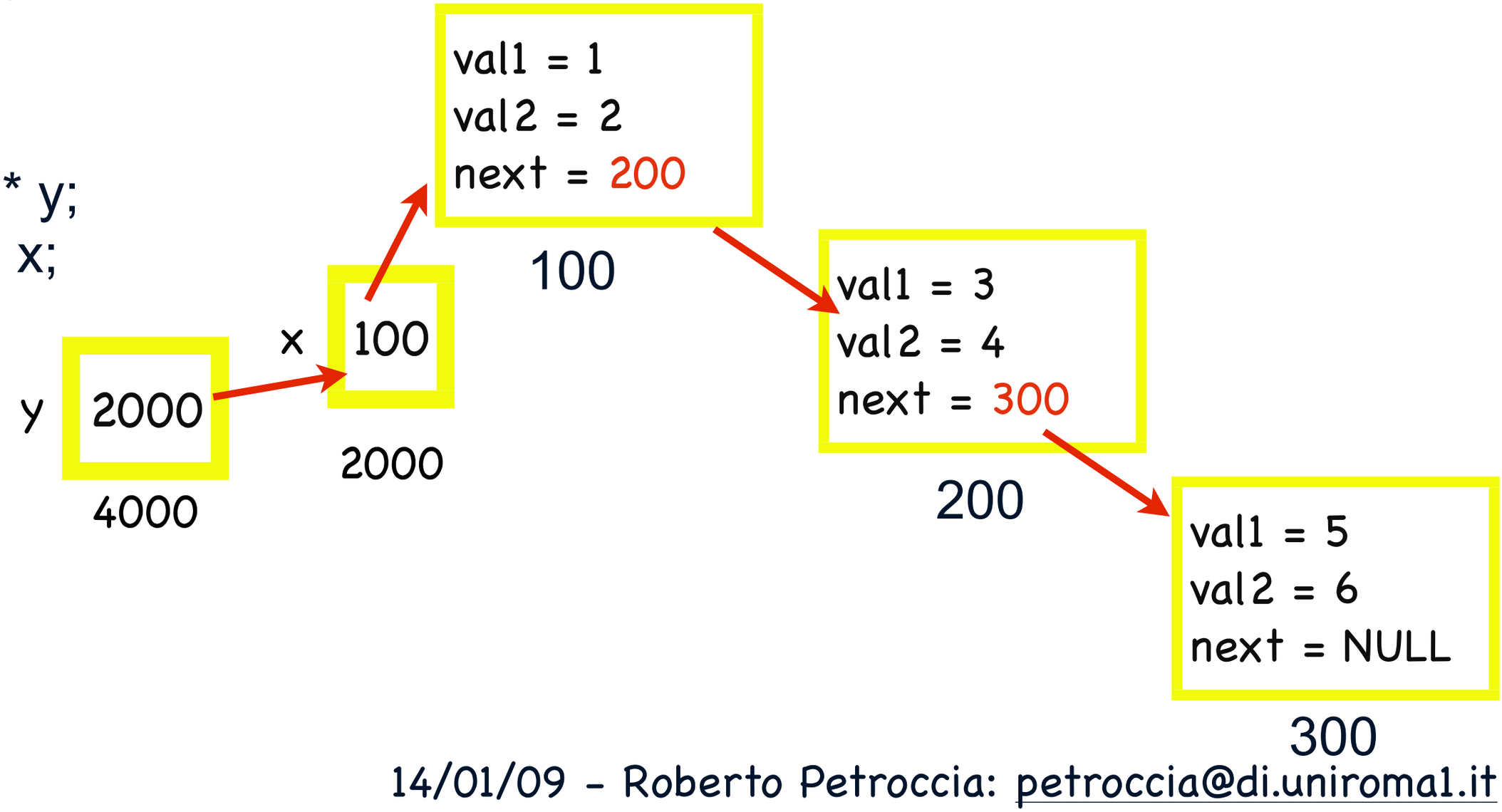
```
typedef struct coppia {
        int val1;
        int val2;
        struct coppia * next;
}coppia;
```



Allocazione memoria

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

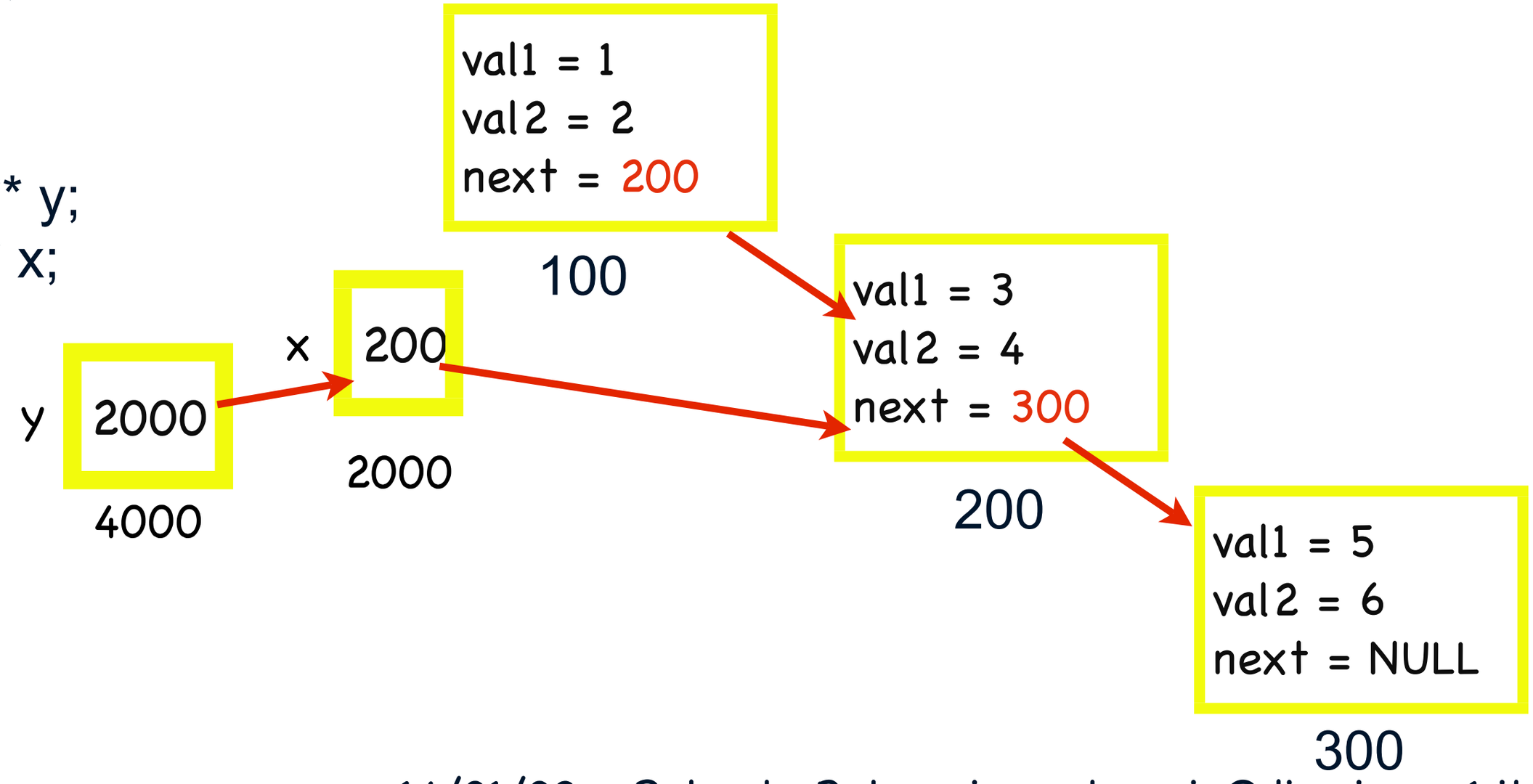
```
coppia** y;
coppia* x;
```



Allocazione memoria

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

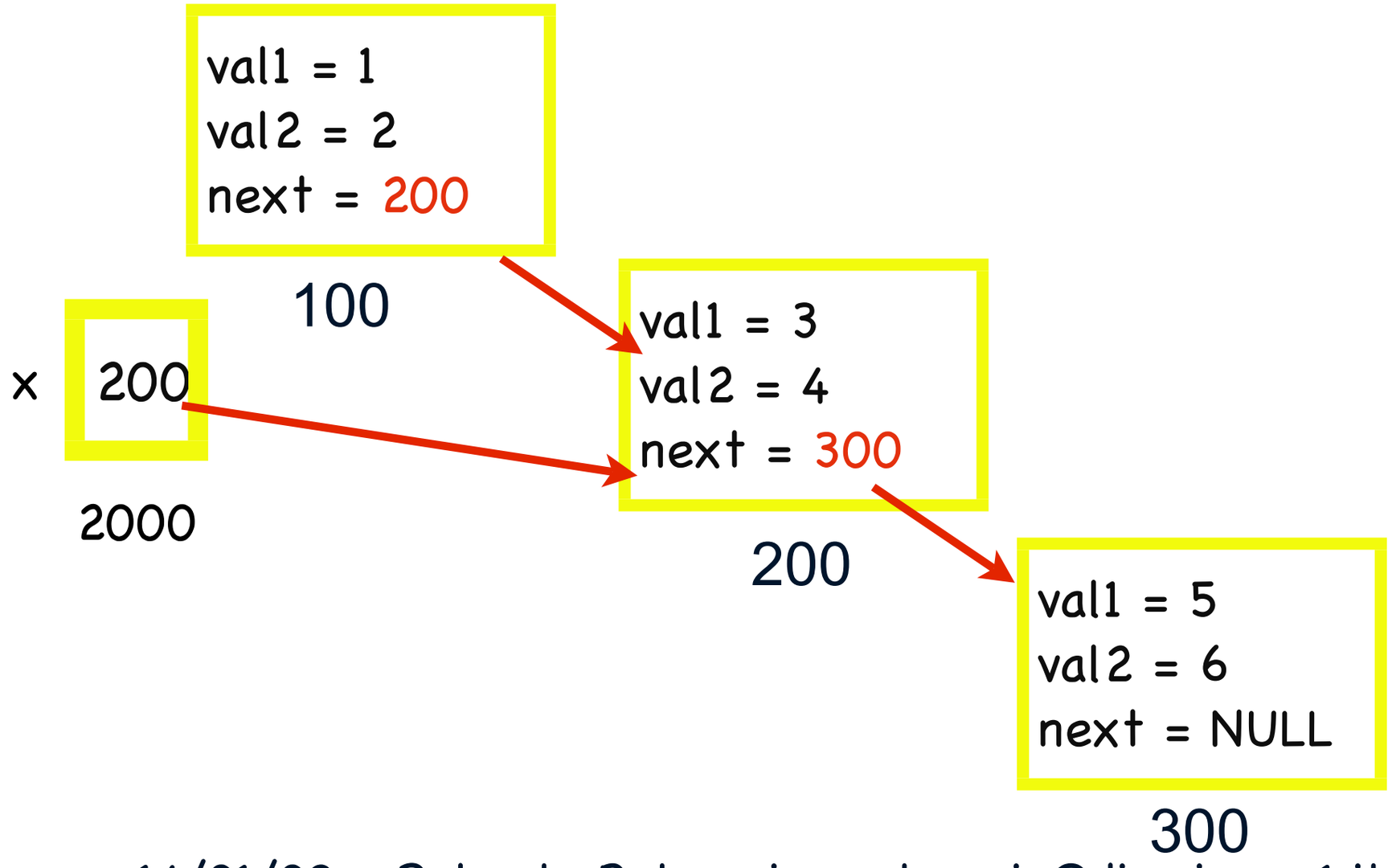
```
coppia** y;
coppia* x;
```



Allocazione memoria

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

```
coppia** y;
coppia* x;
```



LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

h NULL

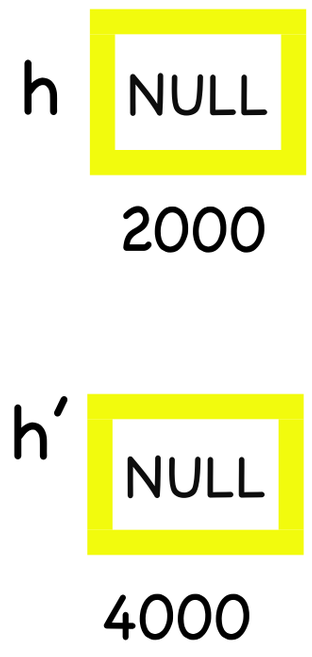
2000

LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

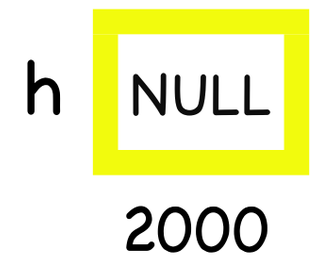


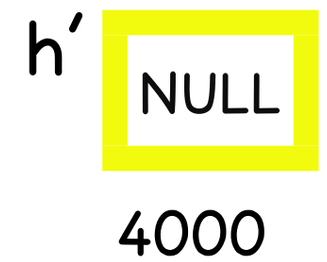
LISTE

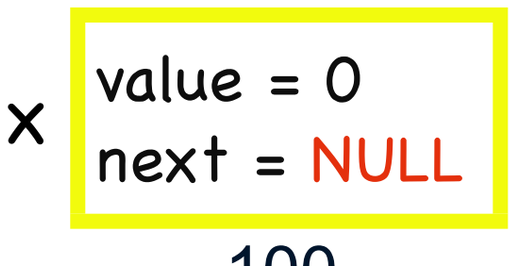
Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```





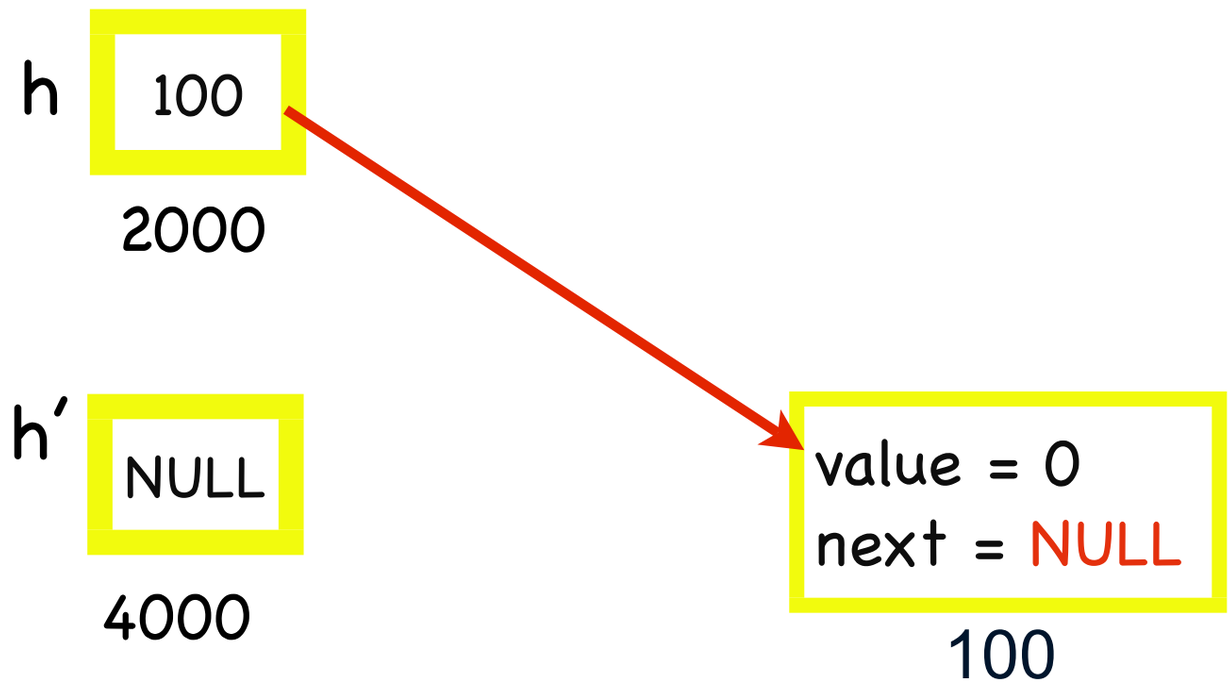


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

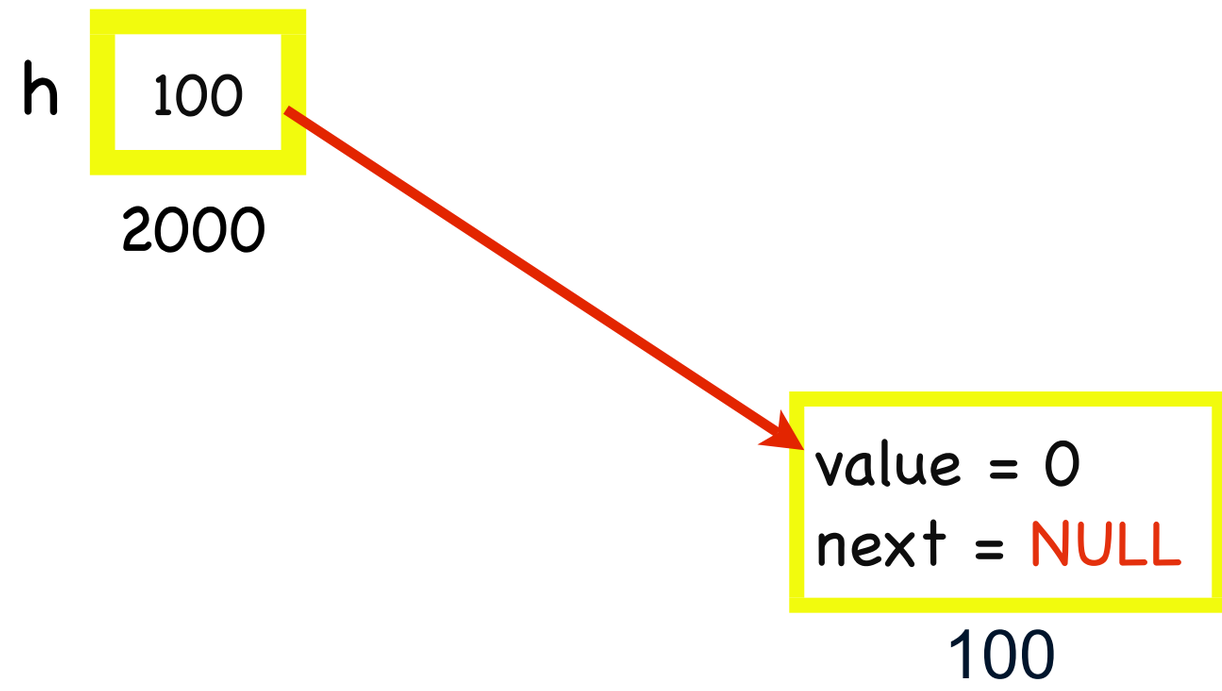


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

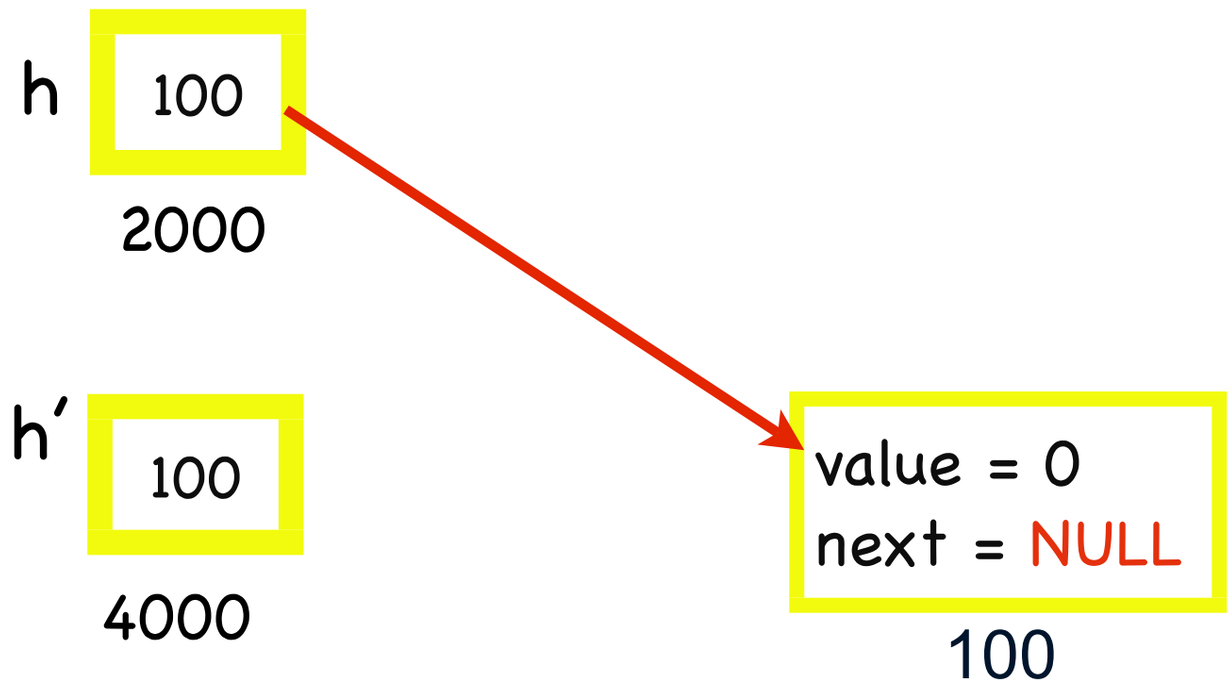


LISTE

Inserimento in testa

```
typedef struct lista {
     int value;
     struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
     listaPtr x = (listaPtr) malloc(sizeof(lista));
     x->value = v;
     x->next = NULL;

     if (h != NULL)  {
          x->next = h;
     }
     return x;
}
```

```
int main() {
     int i;
     listaPtr h;
     h = NULL;
     for (i = 0; i < 100; i++) {
          h = insertHead(h, i);
     }
     return 0;
}
```

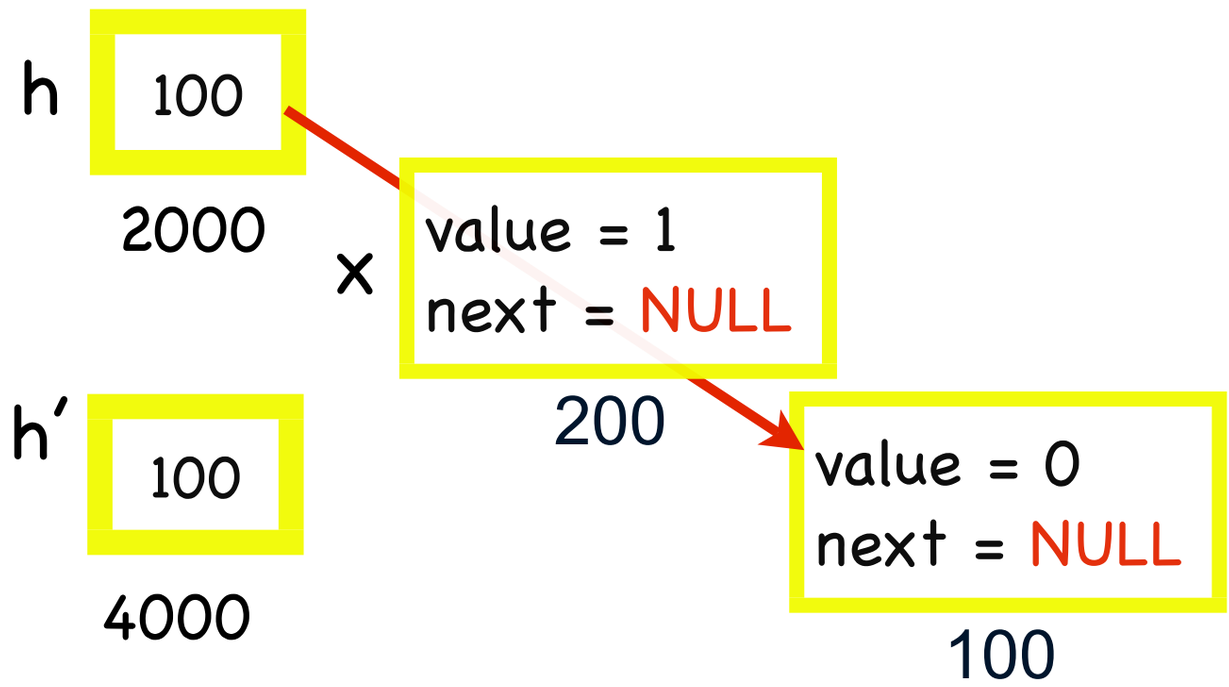


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

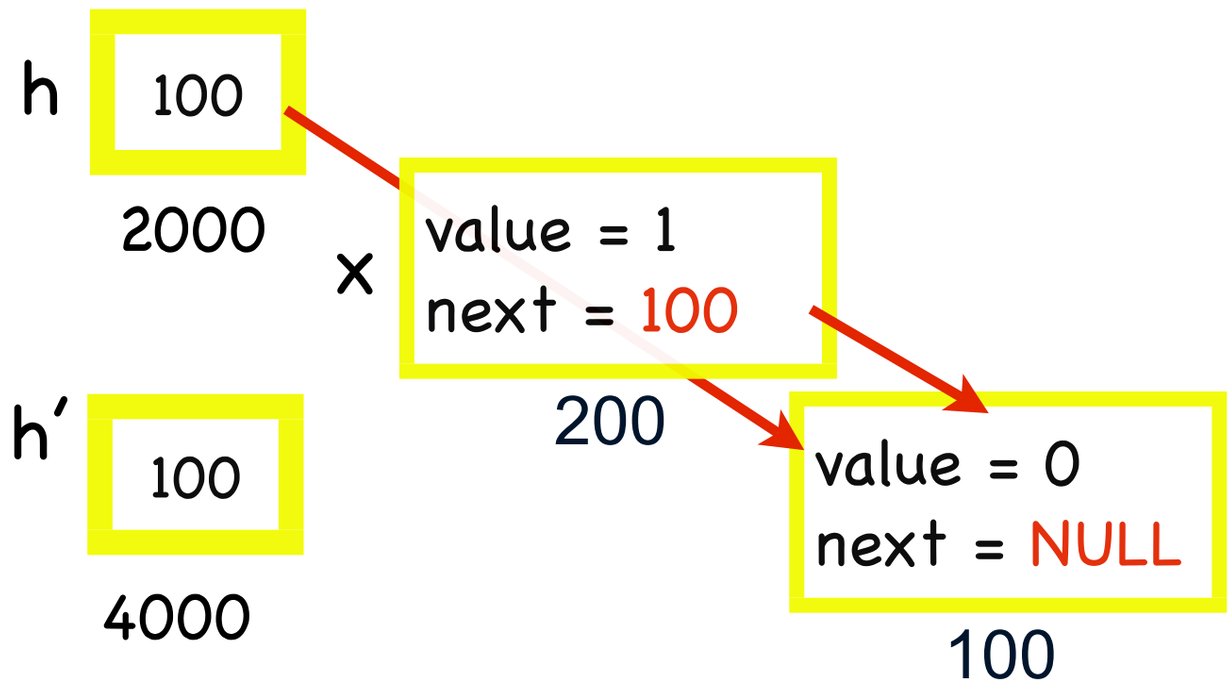


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

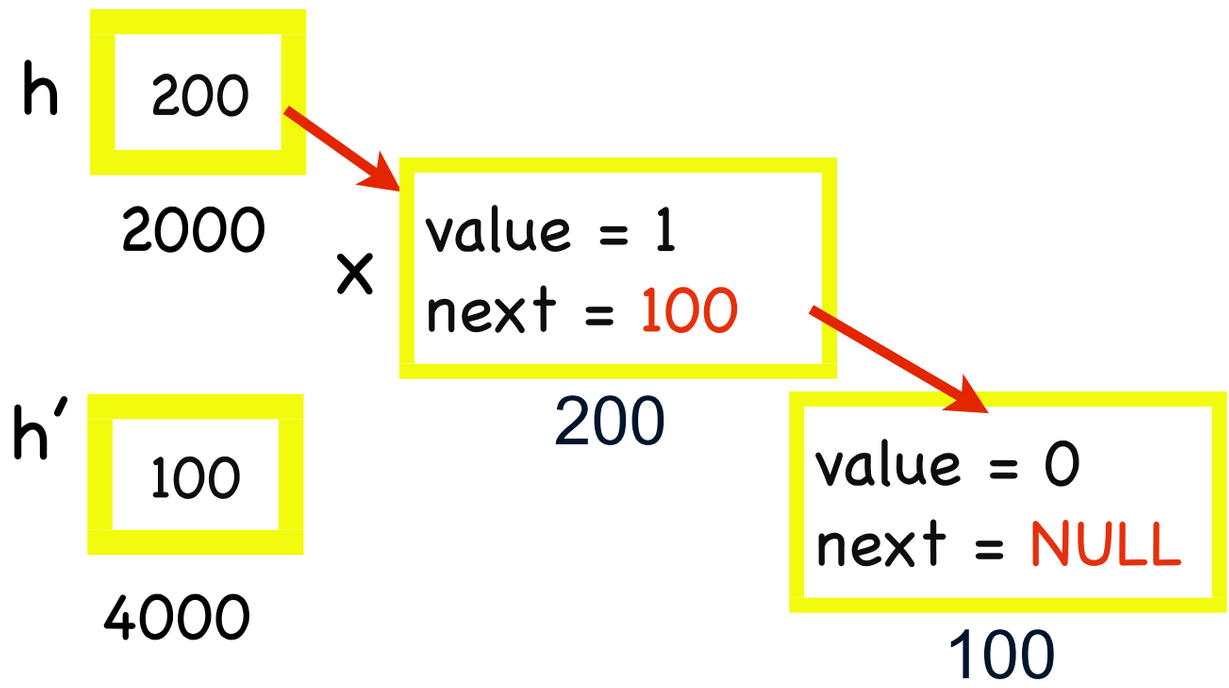


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```

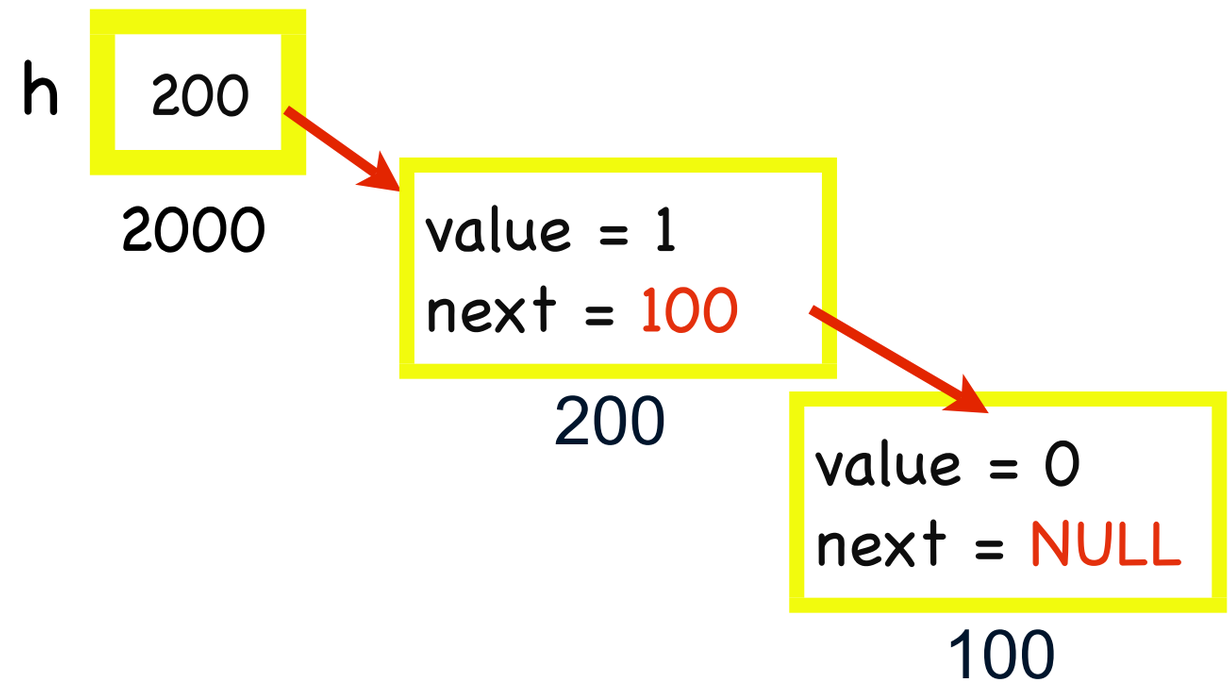


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

listaPtr insertHead(listaPtr h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = h;
    }
    return x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        h = insertHead(h, i);
    }
    return 0;
}
```



LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

h NULL

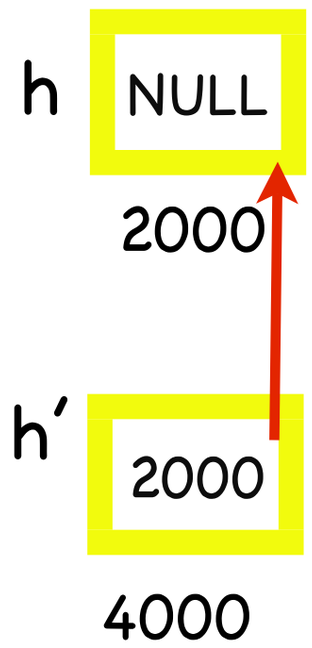
2000

LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```



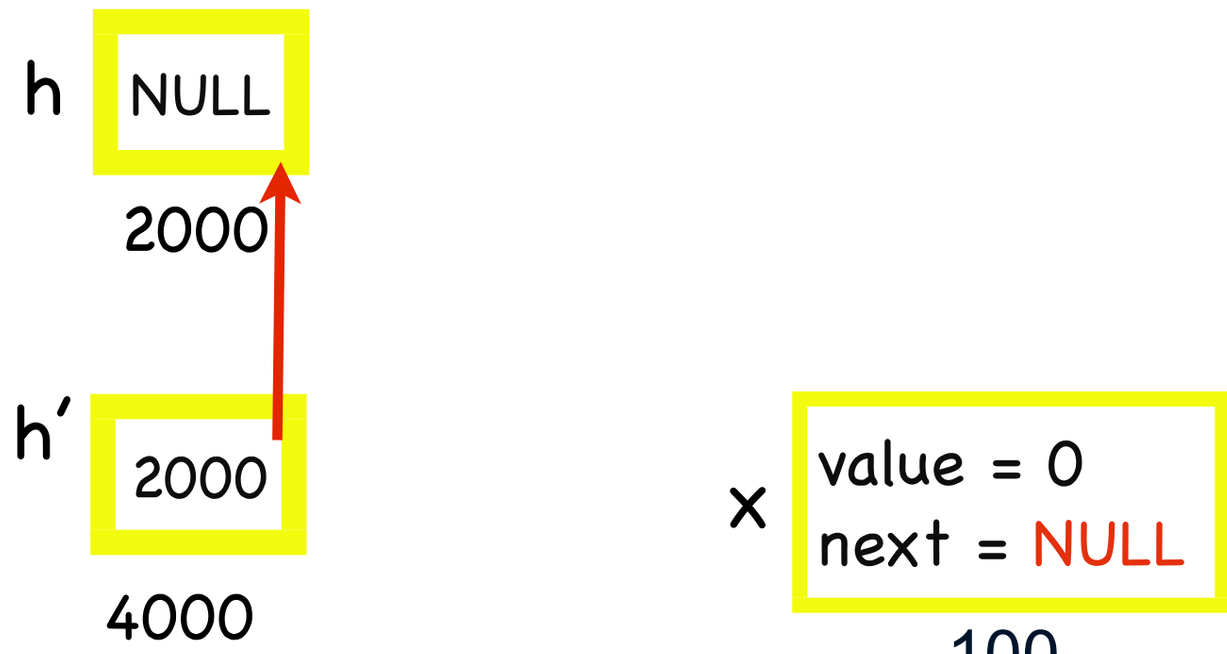
14/01/09 - Roberto Petroccia: petroccia@di.uniroma1.it

LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

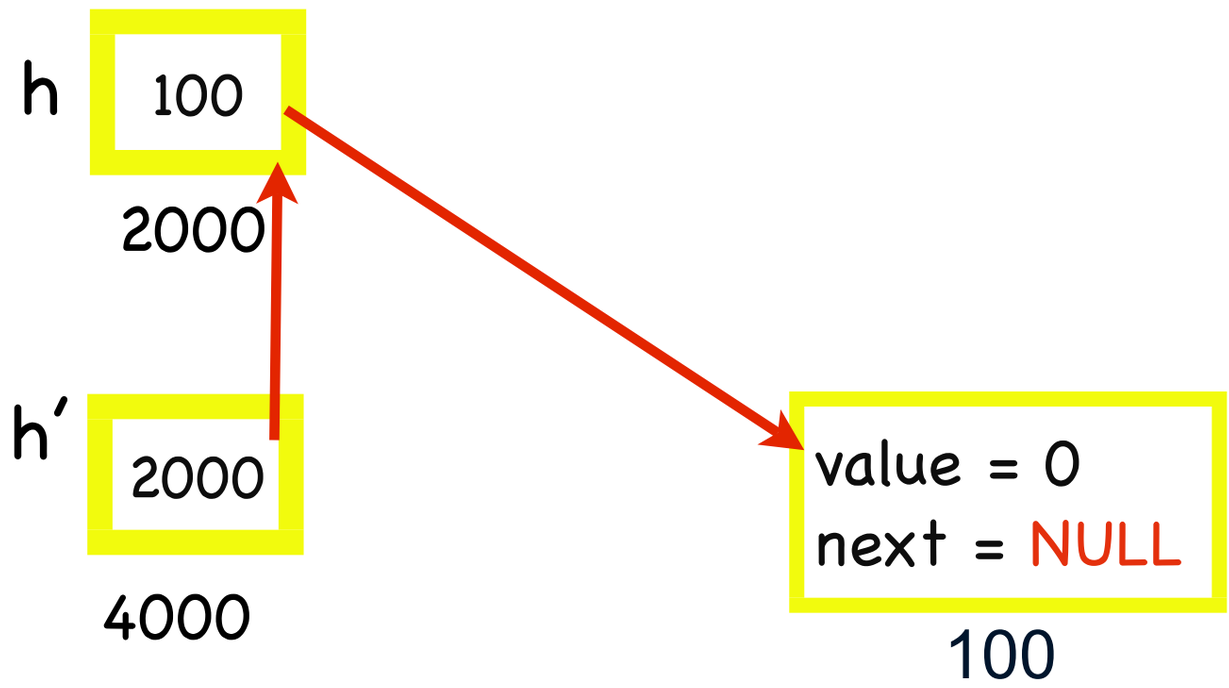


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

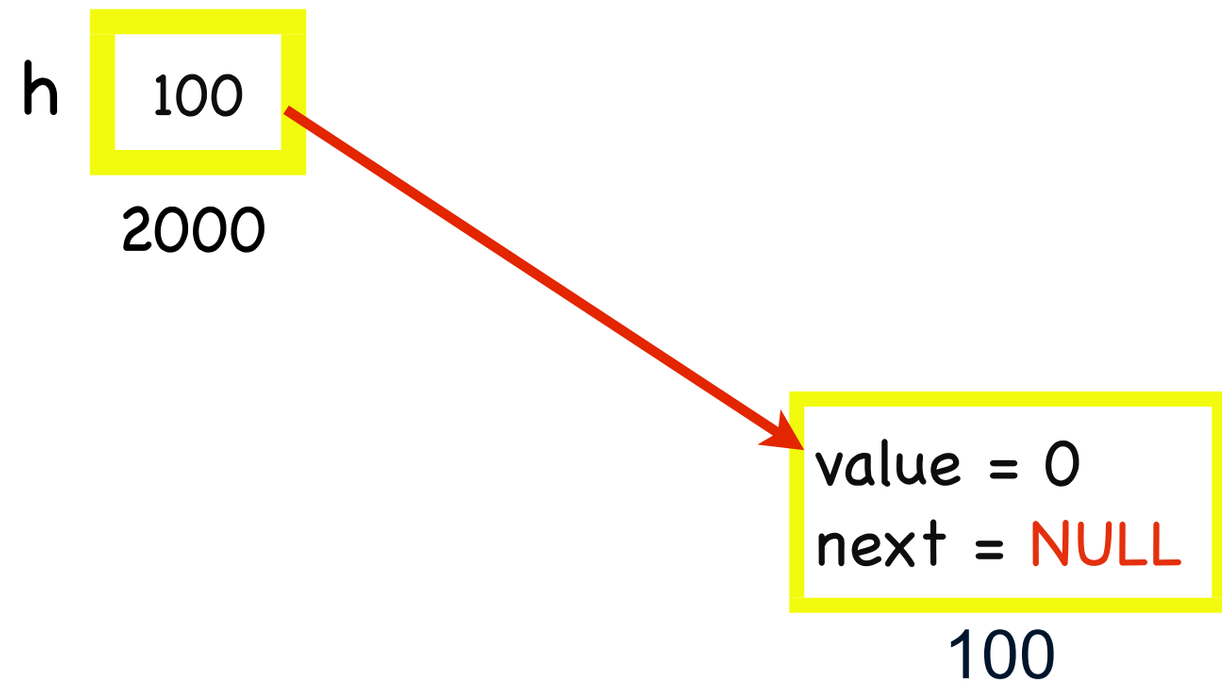


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

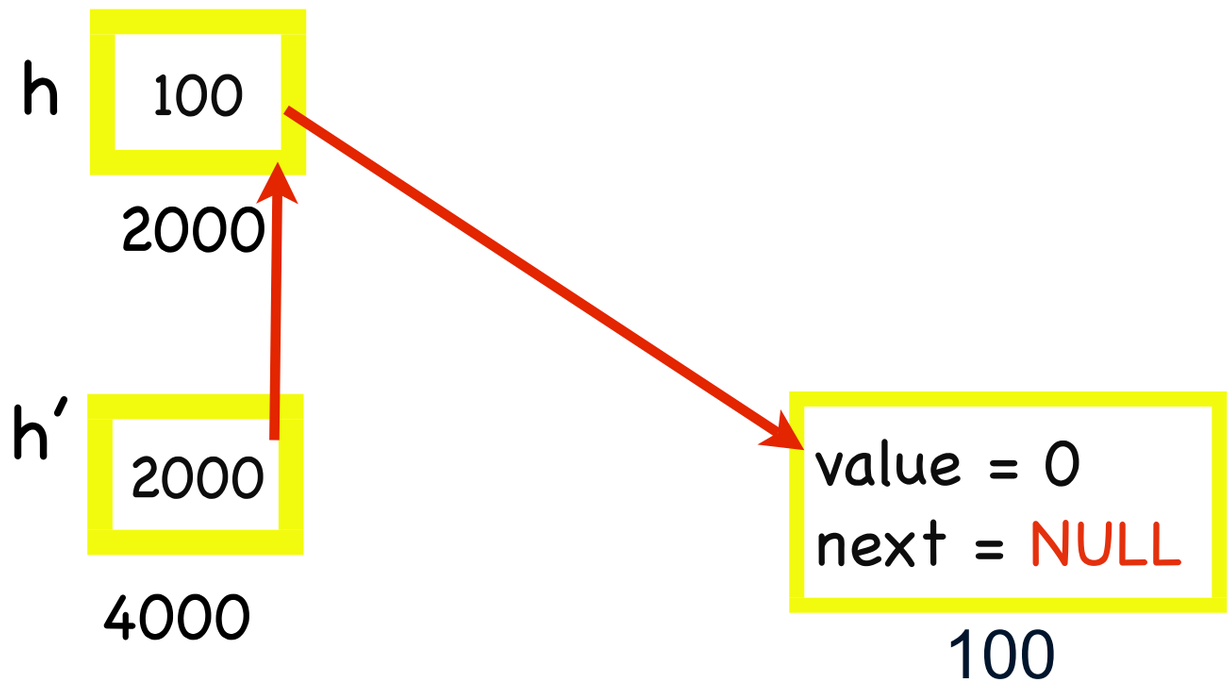


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

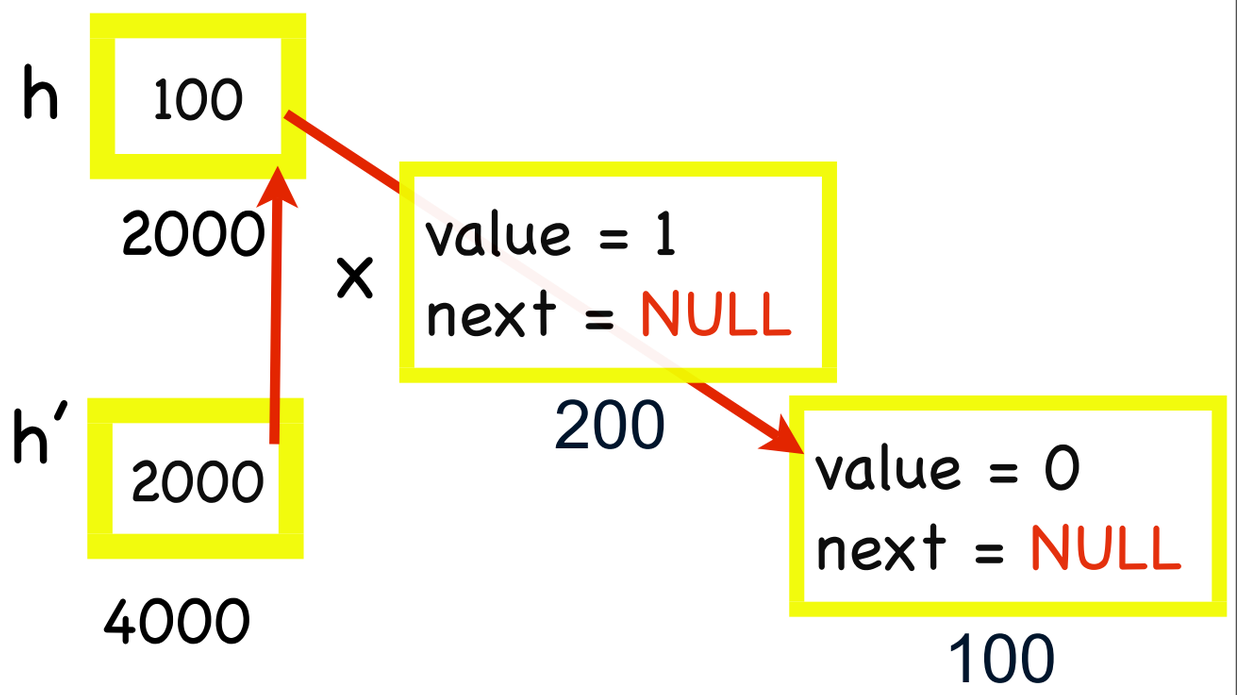


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

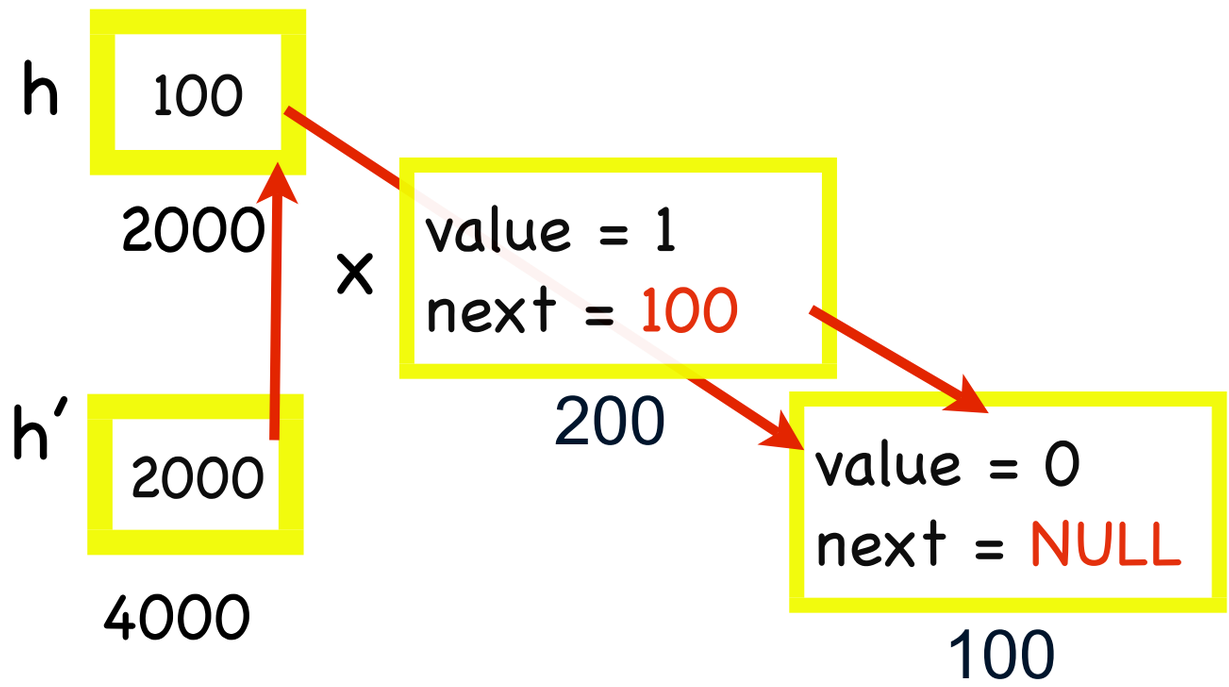


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

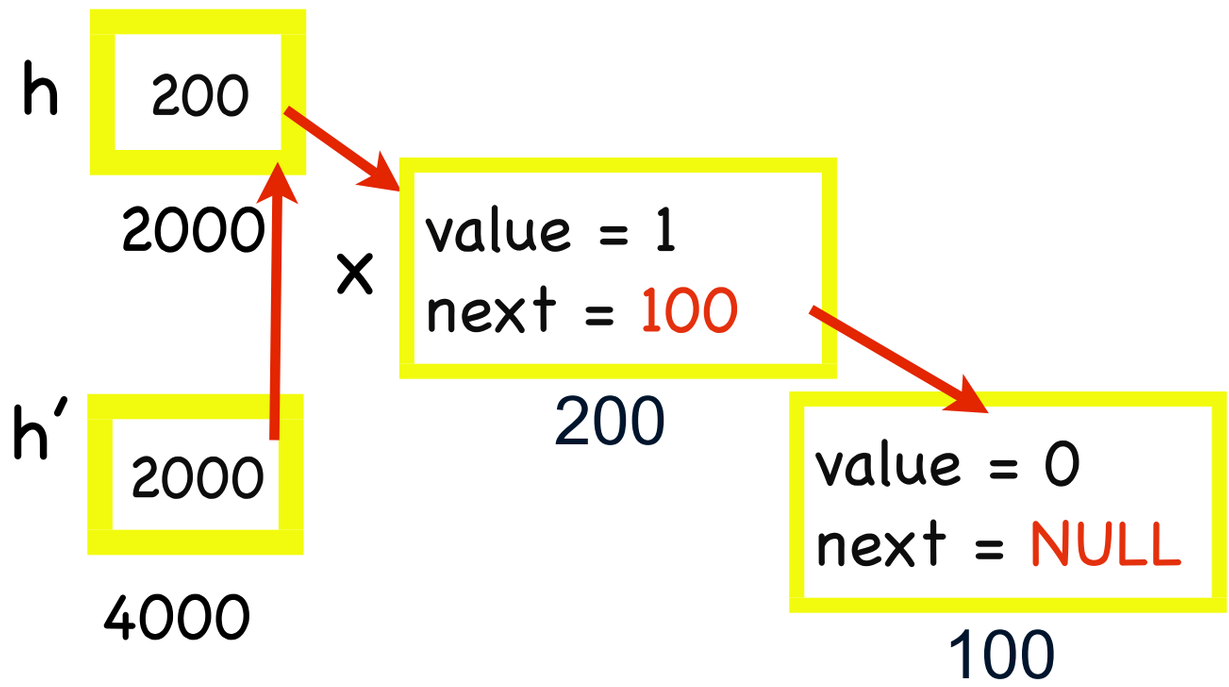


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```

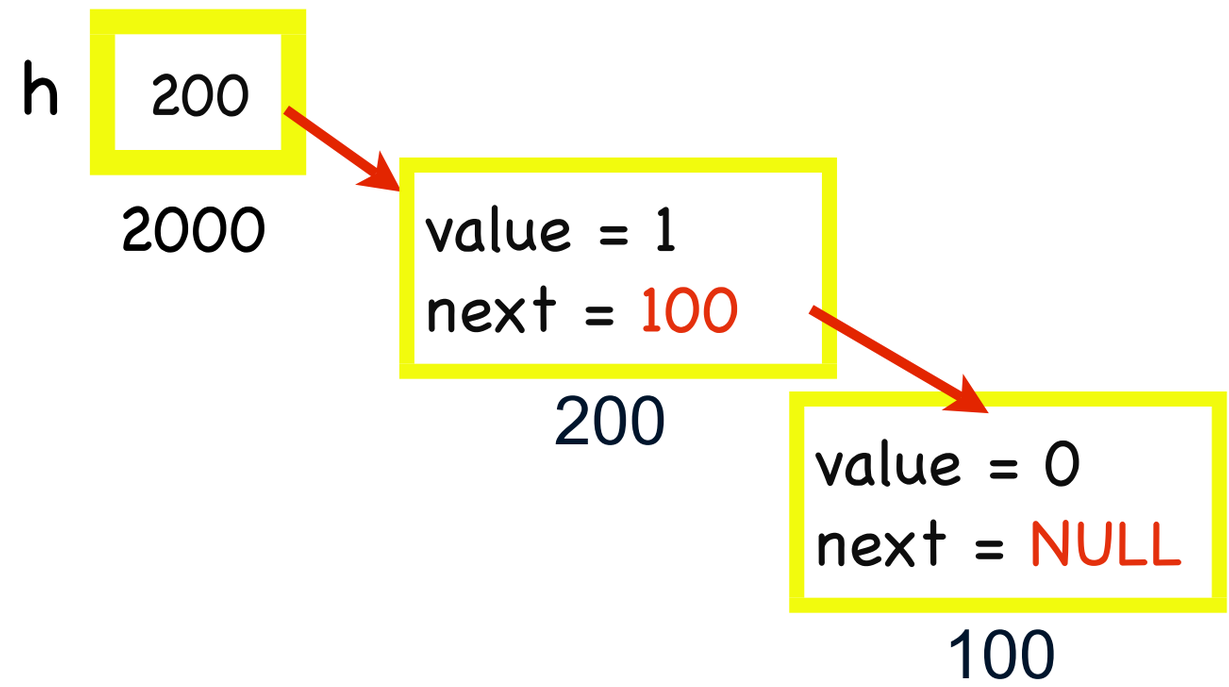


LISTE

Inserimento in testa

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;

void insertHead(listaPtr* h, int v) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;

    if (h != NULL)  {
        x->next = *h;
    }
    *h = x;
}
```

```
int main() {
    int i;
    listaPtr h;
    h = NULL;
    for (i = 0; i < 100; i++) {
        insertHead(&h, i);
    }
    return 0;
}
```



LISTE

Stampa Lista

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void printLista(listaPtr h) {
    if (h == NULL) {
        printf("Lista vuota\n");
    }
    else {
        while (h != NULL) {
            printf("%d - ", h->value);
            h = h->next;
        }
        printf("\n");
    }
}
```

```
void printListaRec(listaPtr h) {
    if (h != NULL) {
        printf("%d ", h->value);
        if (h->next != NULL) {
            printf("- ");
            printListaRec(h->next);
        }
        else {
            printf("\n");
        }
    }
}
```

LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

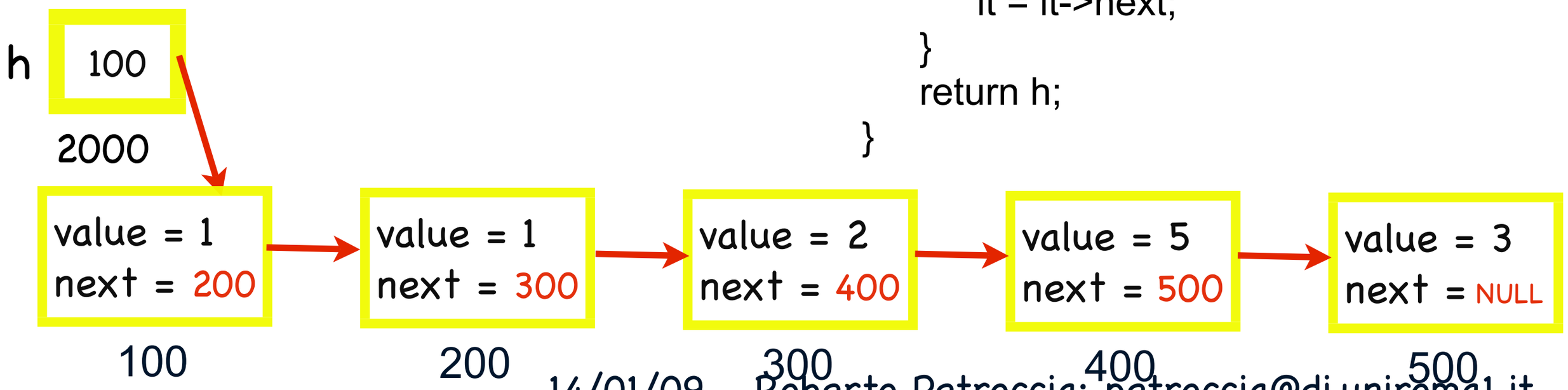
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```



LISTE

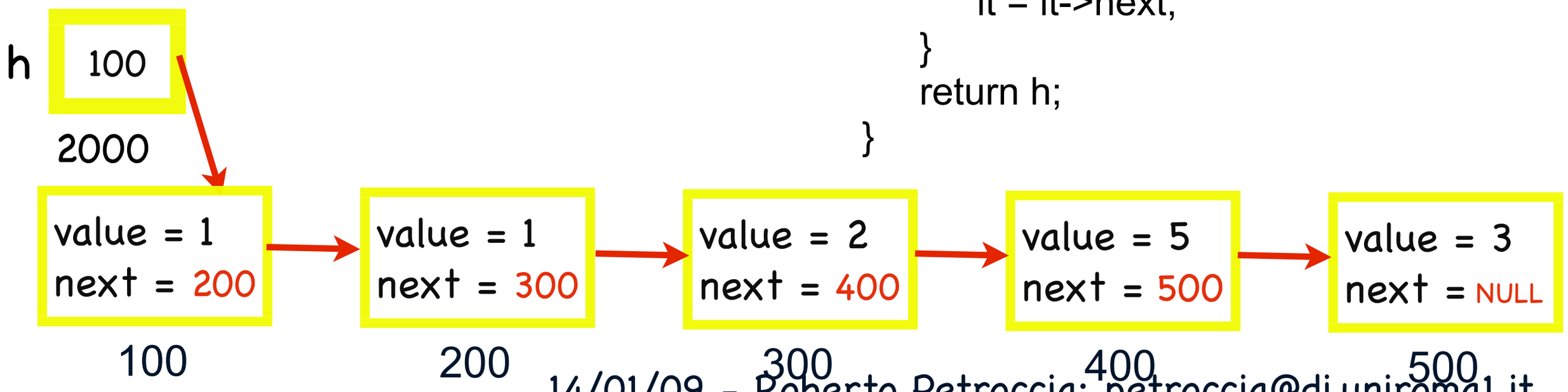
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 1

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```



LISTE

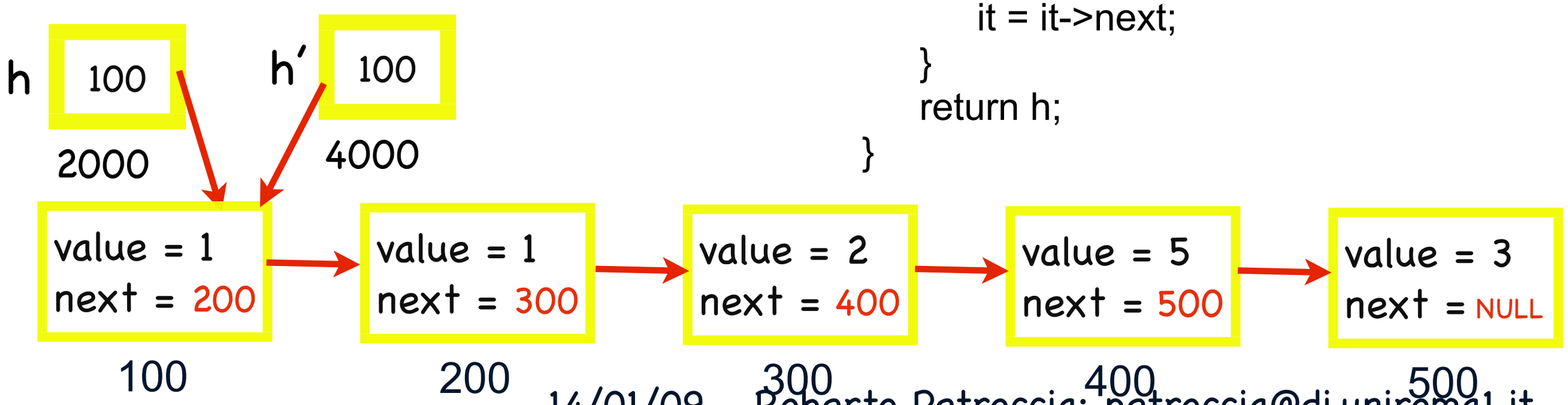
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 1



LISTE

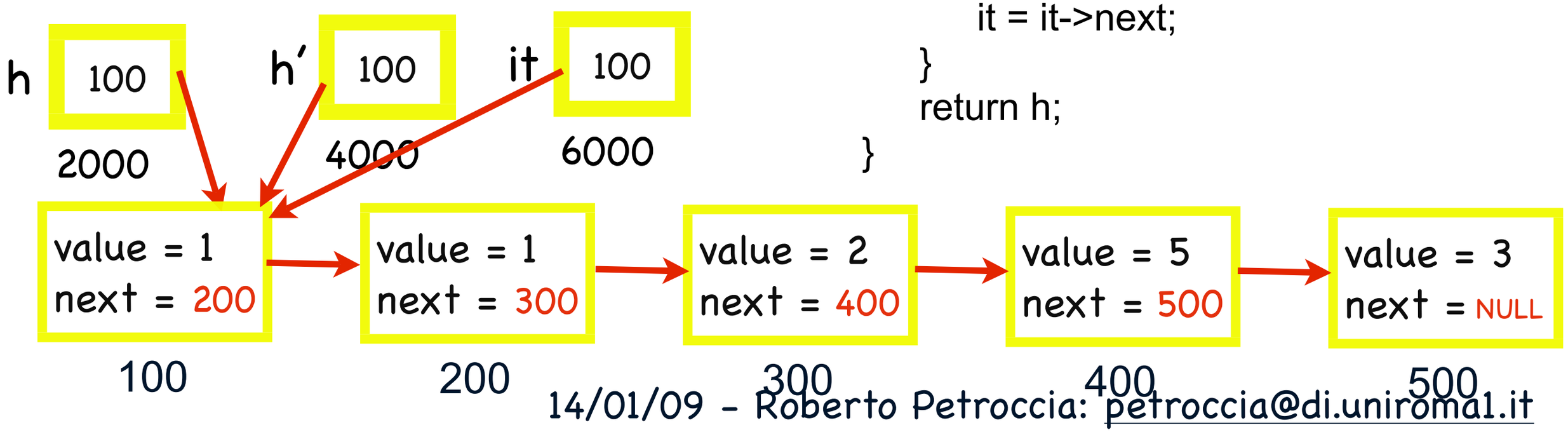
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 1



LISTE

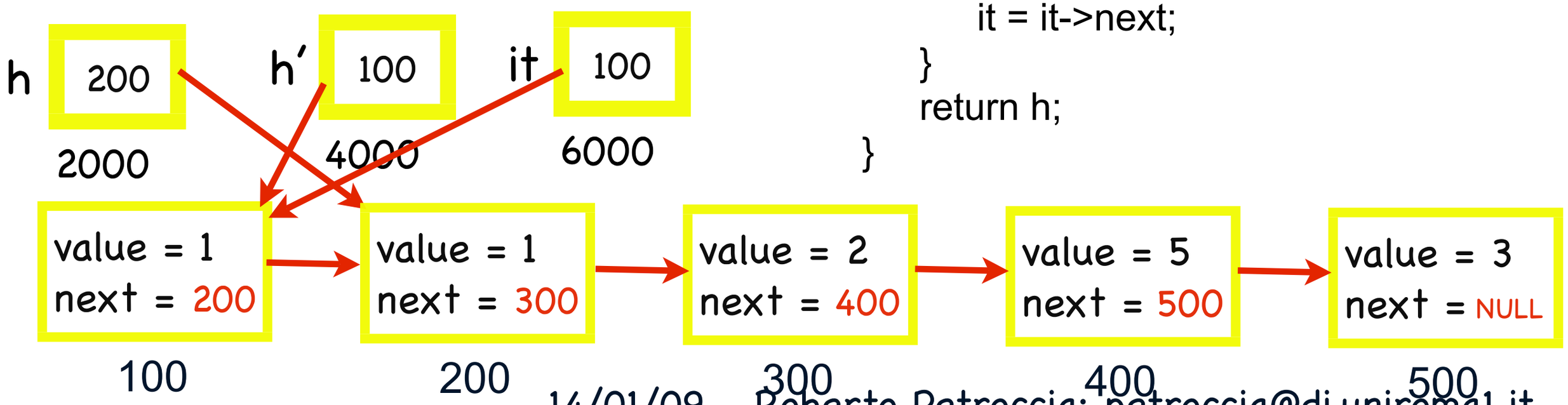
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 1



LISTE

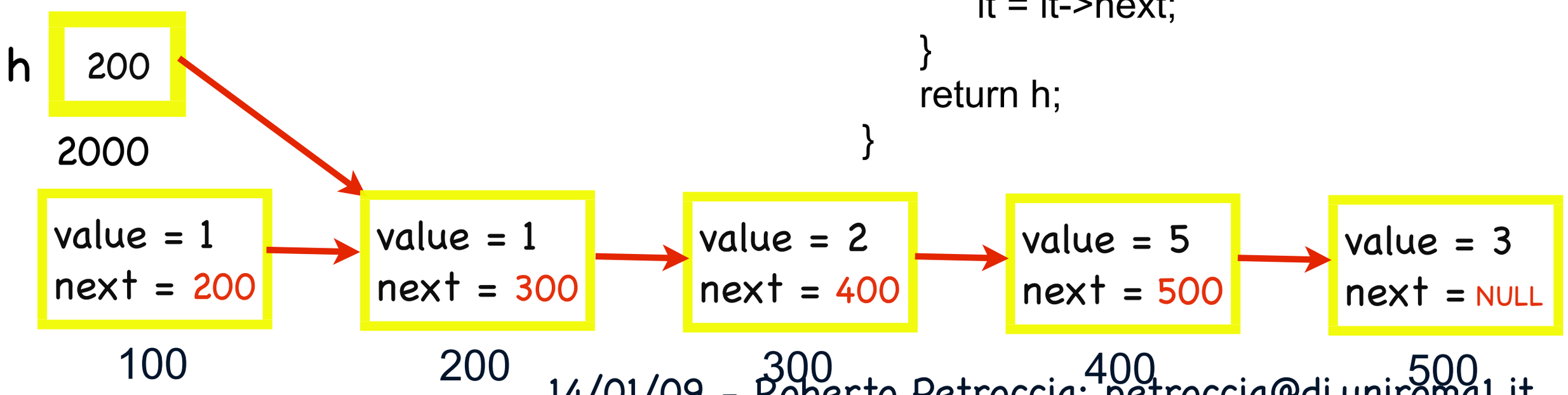
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 1

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```



LISTE

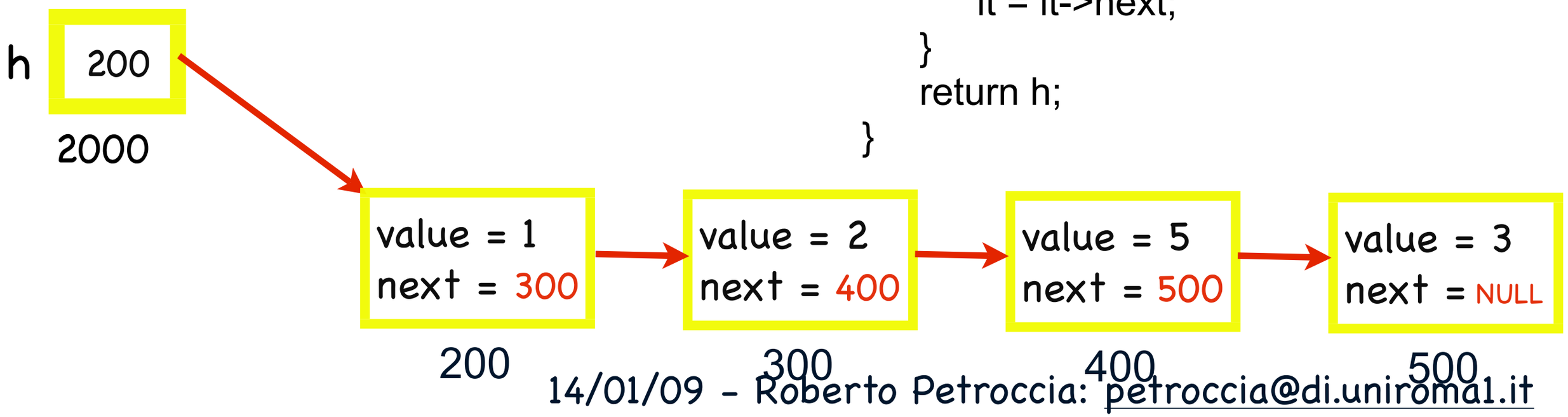
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 1

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```



LISTE

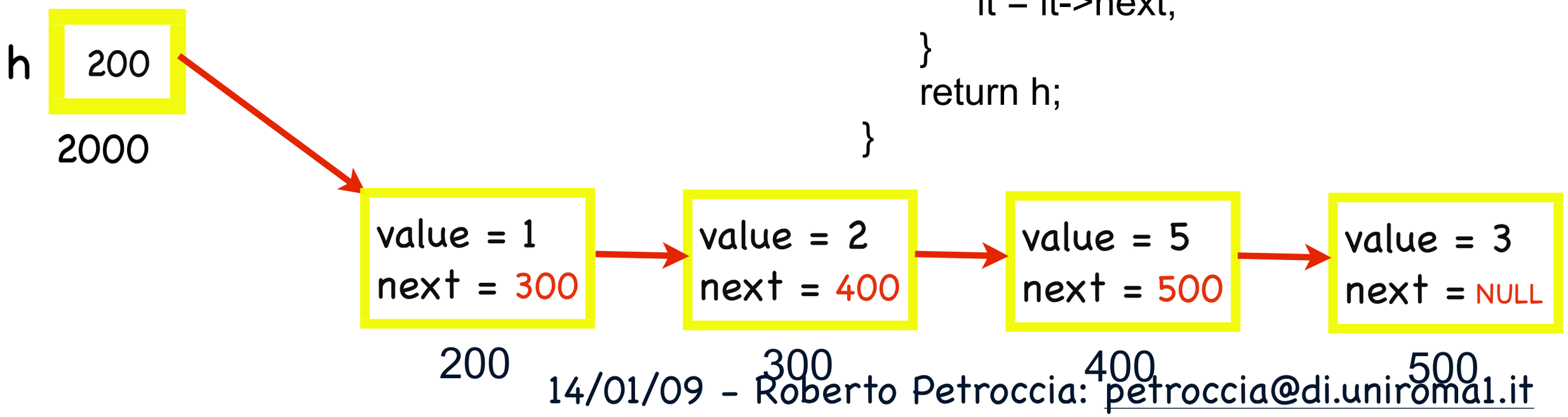
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 5

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```



LISTE

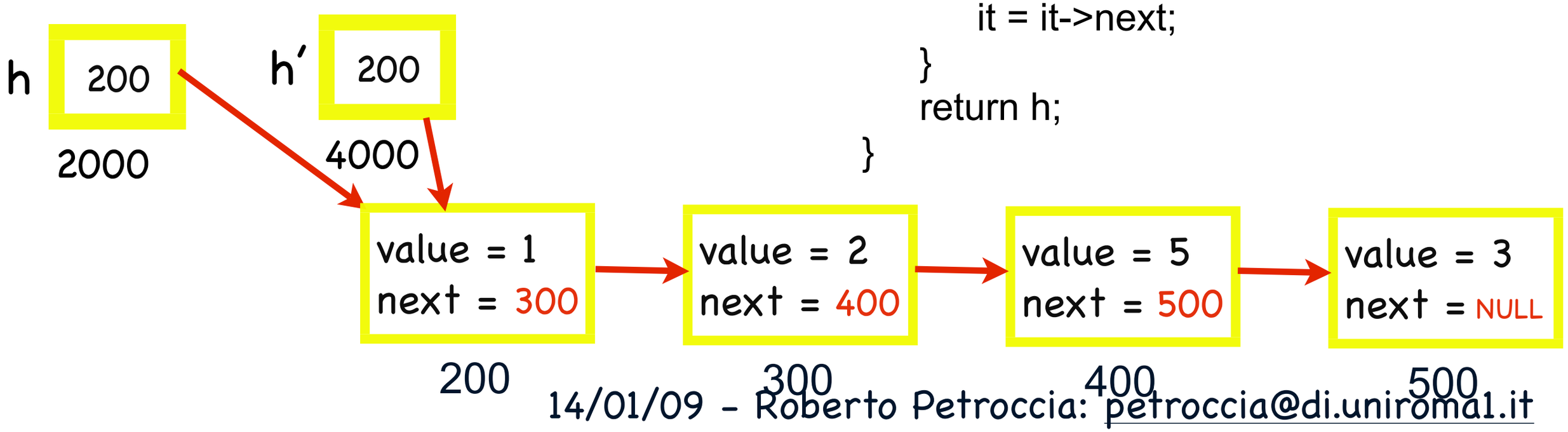
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 5



LISTE

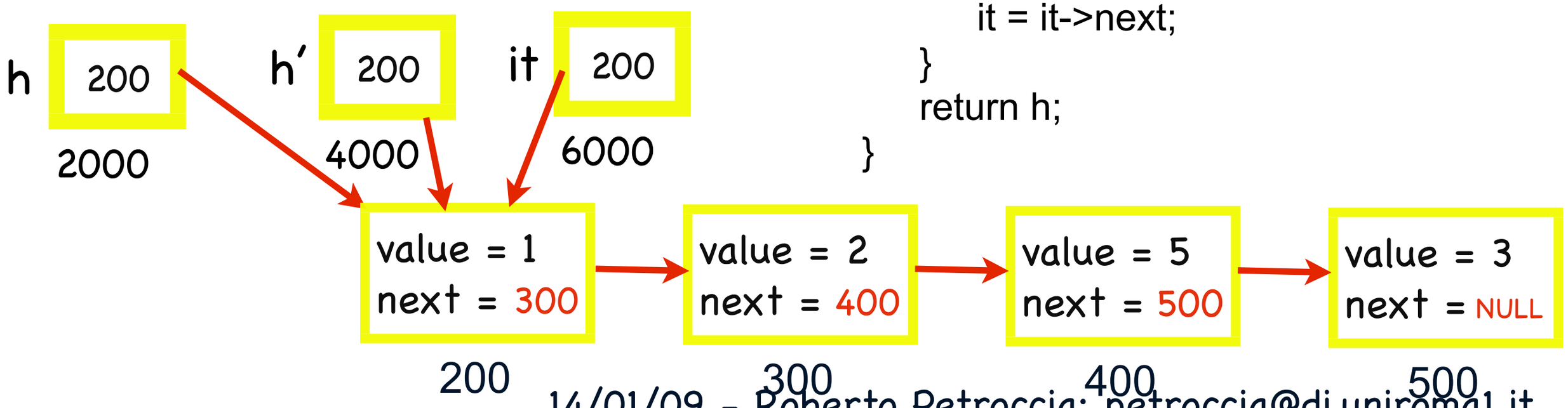
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 5



LISTE

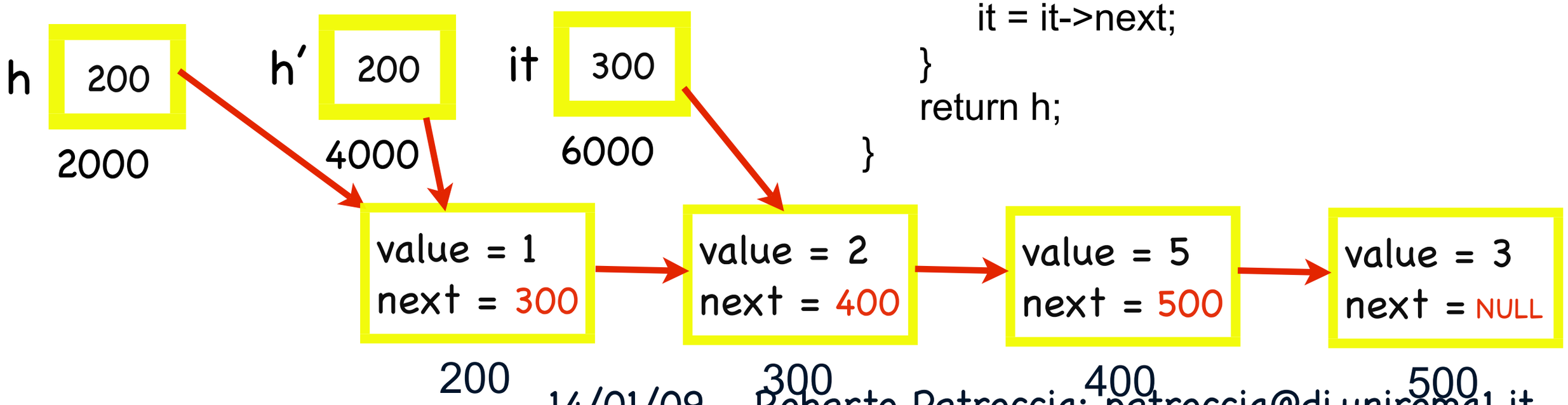
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 5



LISTE

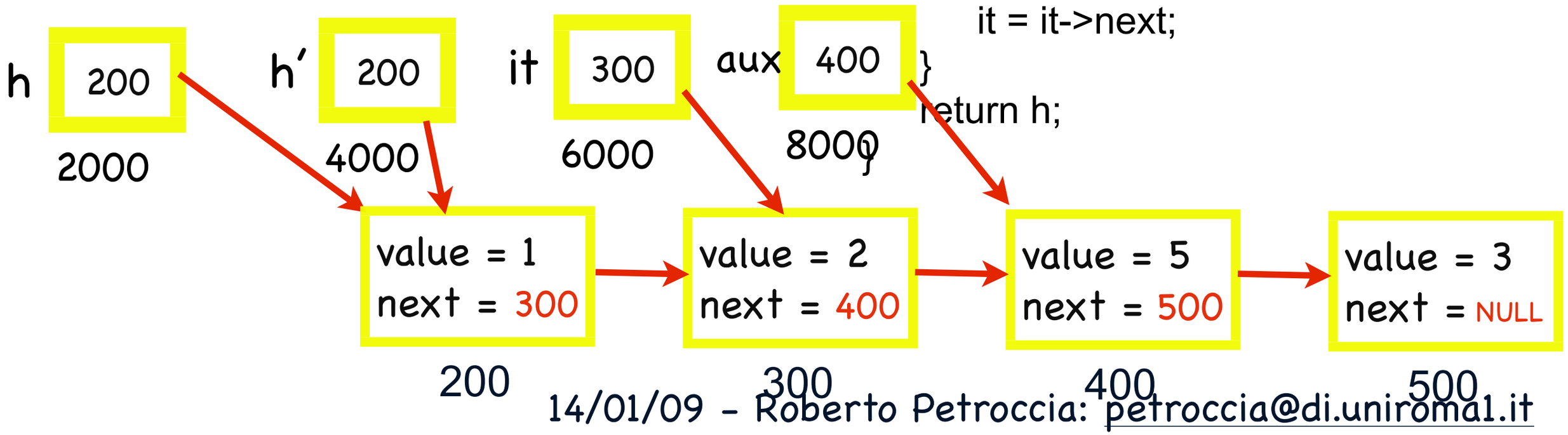
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 5



LISTE

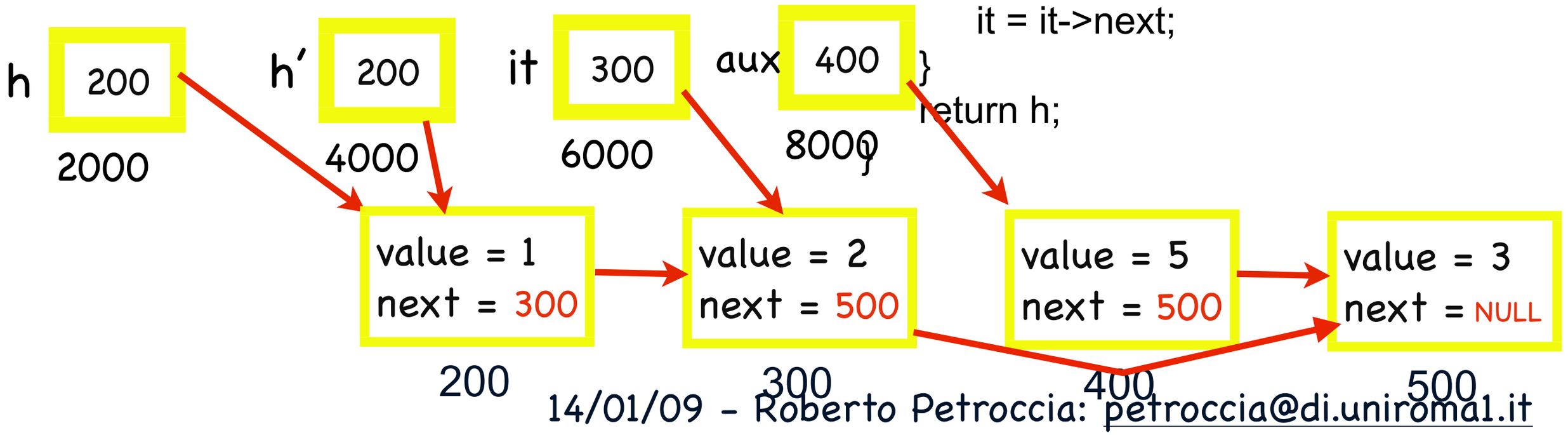
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 5



LISTE

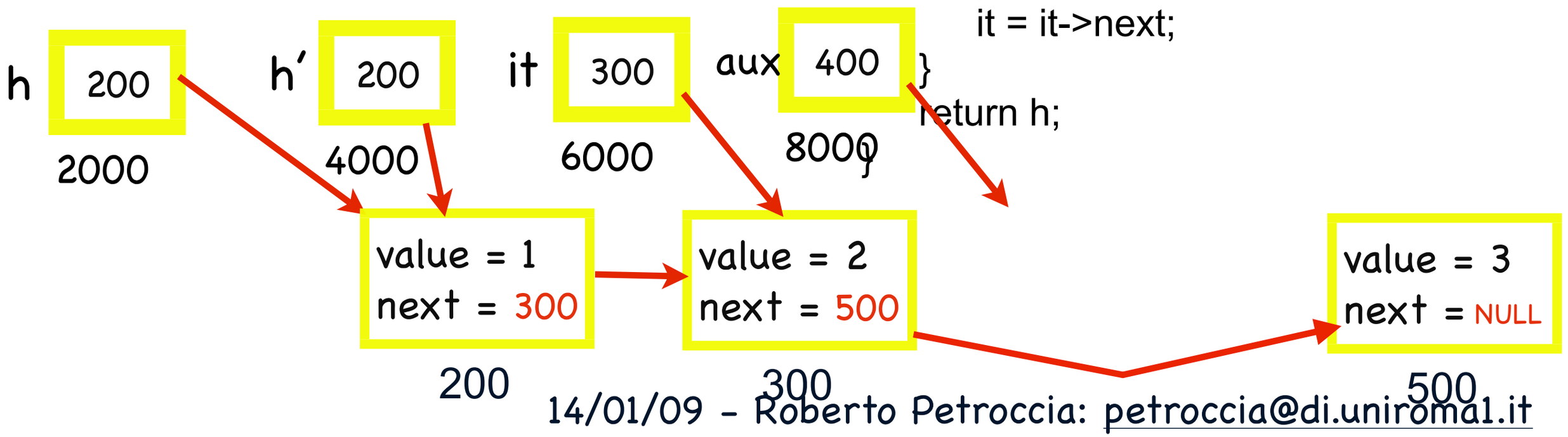
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```

v = 5



LISTE

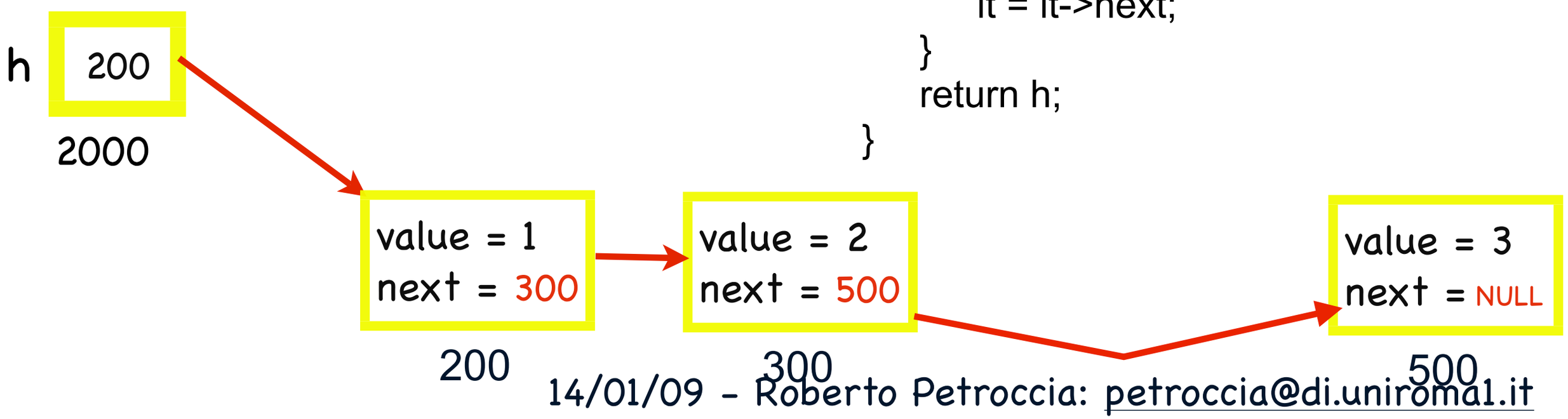
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 5

```
listaPtr eraseFirst(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it = h;
    if (h != NULL && h->value == v) {
        return h->next;
    }
    int check = 1;
    while (it->next != NULL && check) {
        if ((it->next)->value == v) {
            listaPtr aux = it->next;
            it->next = (it->next)->next;
            free(aux);
            check = 0;
        }
        it = it->next;
    }
    return h;
}
```



LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseFirstRic2(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr it;
    if (h->value != v) {
        h->next = eraseFirstRic2(h->next, v);
        return h;
    }
    it = h;
    h = h->next;
    free(it);
    return h;
}
```

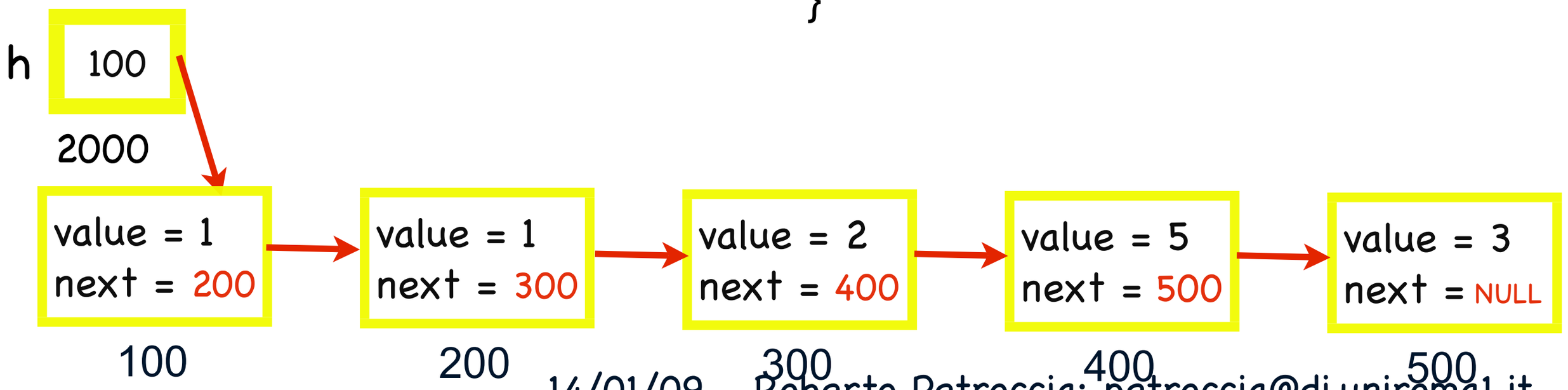
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

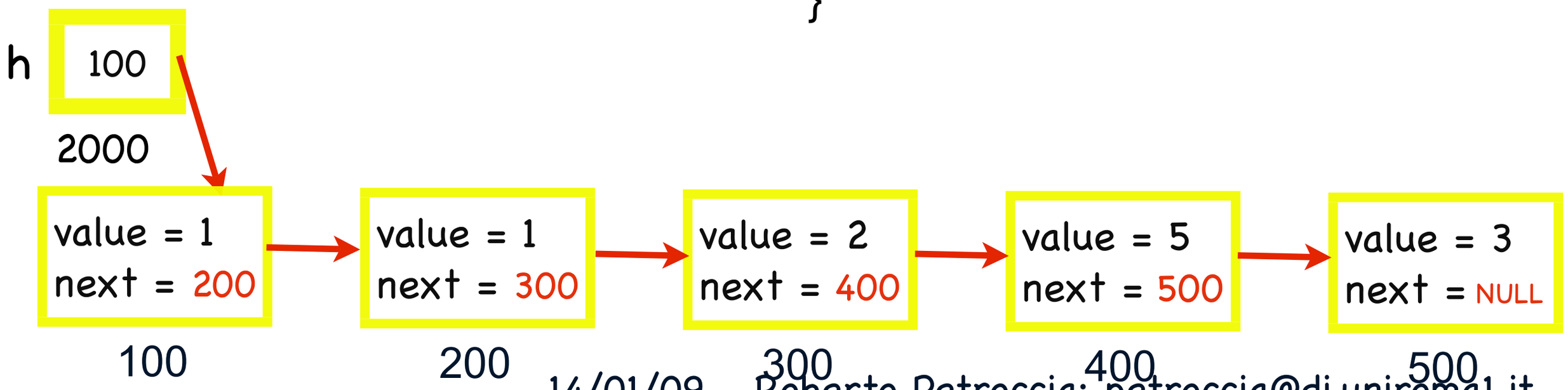
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 1

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

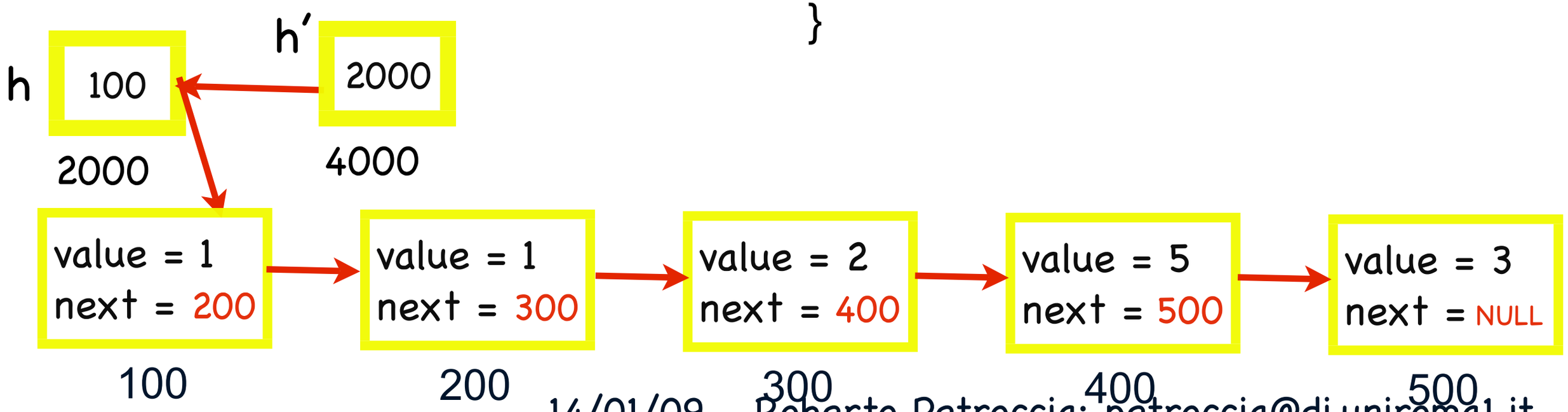
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 1

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



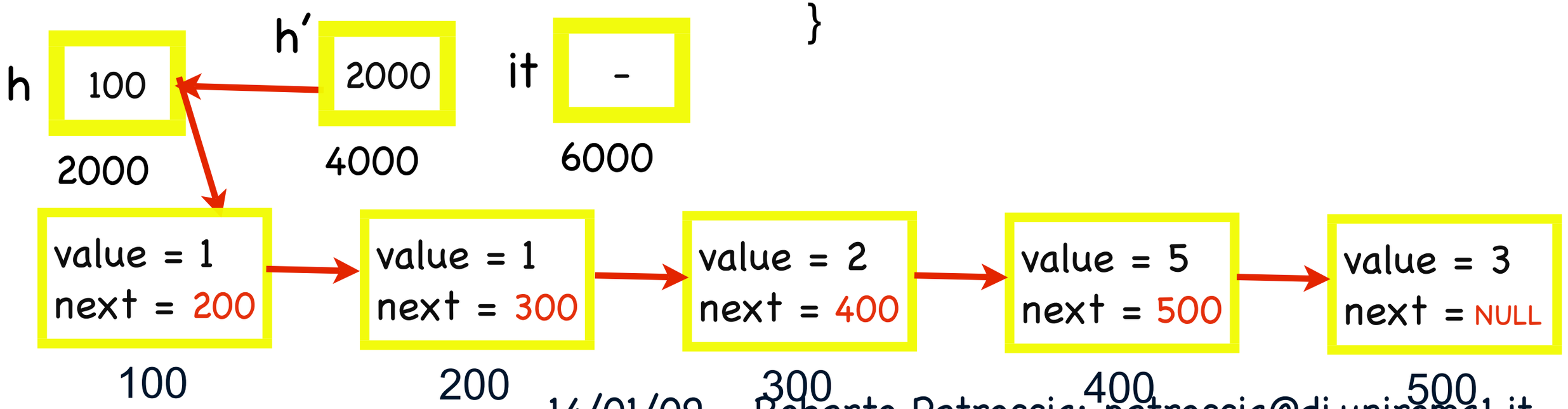
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



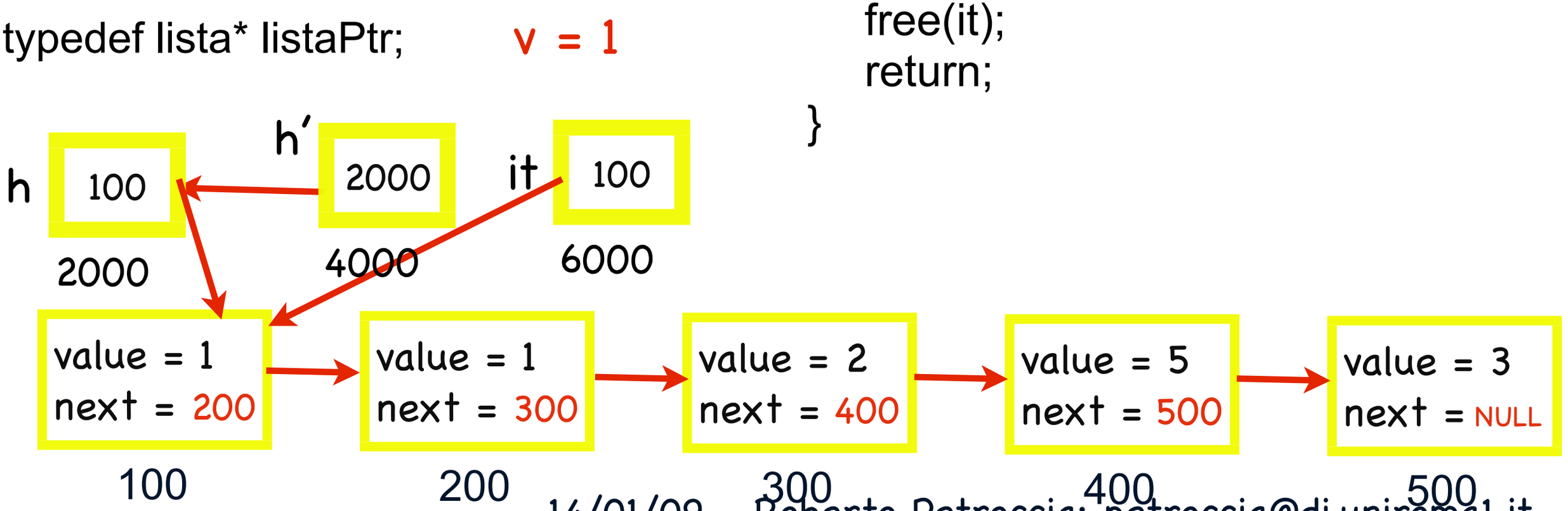
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



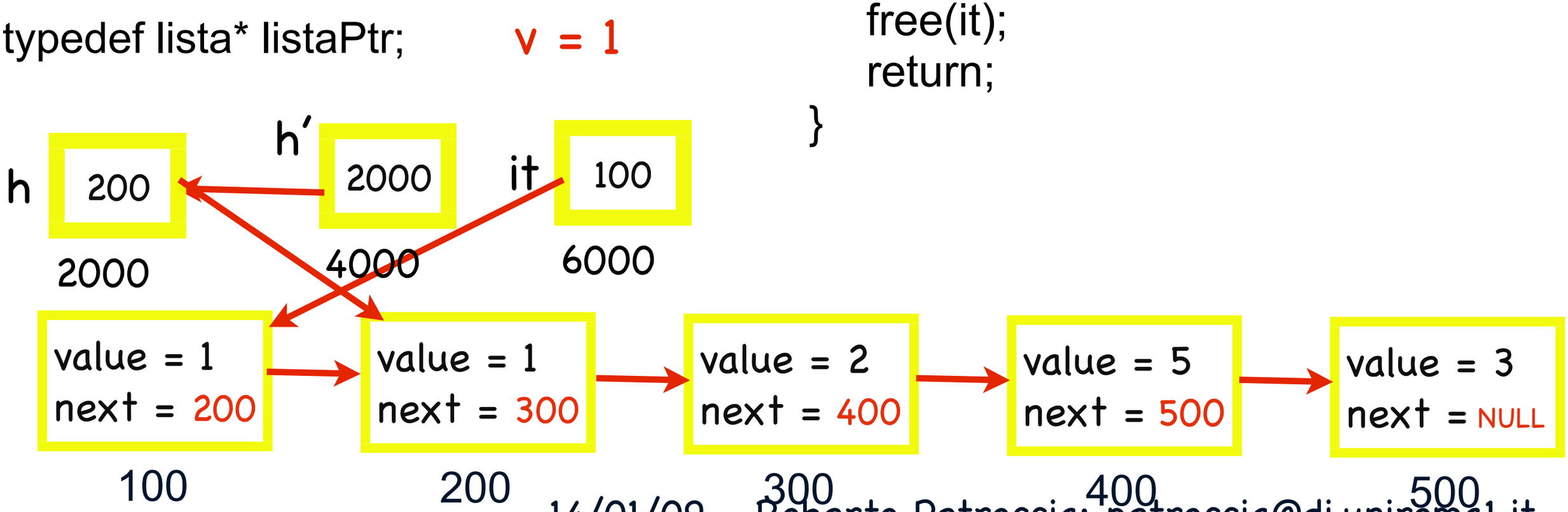
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



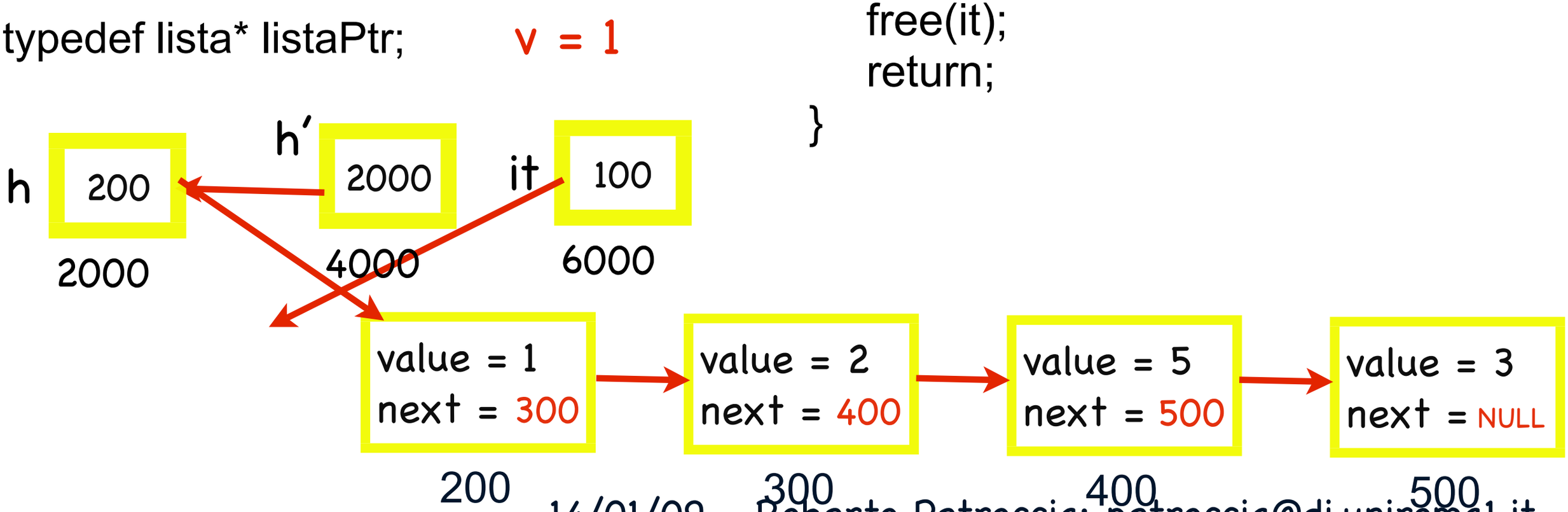
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

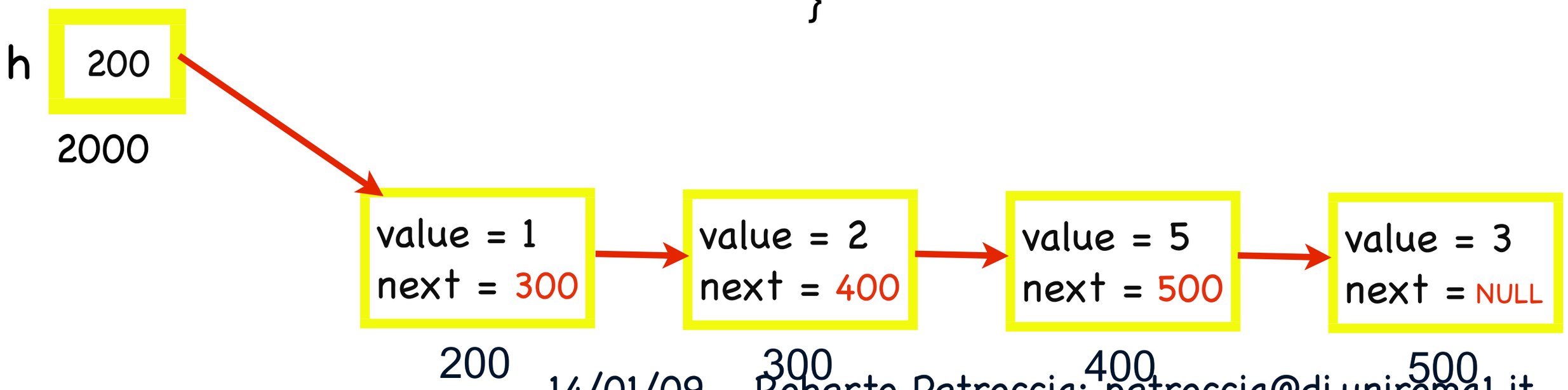
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 1

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

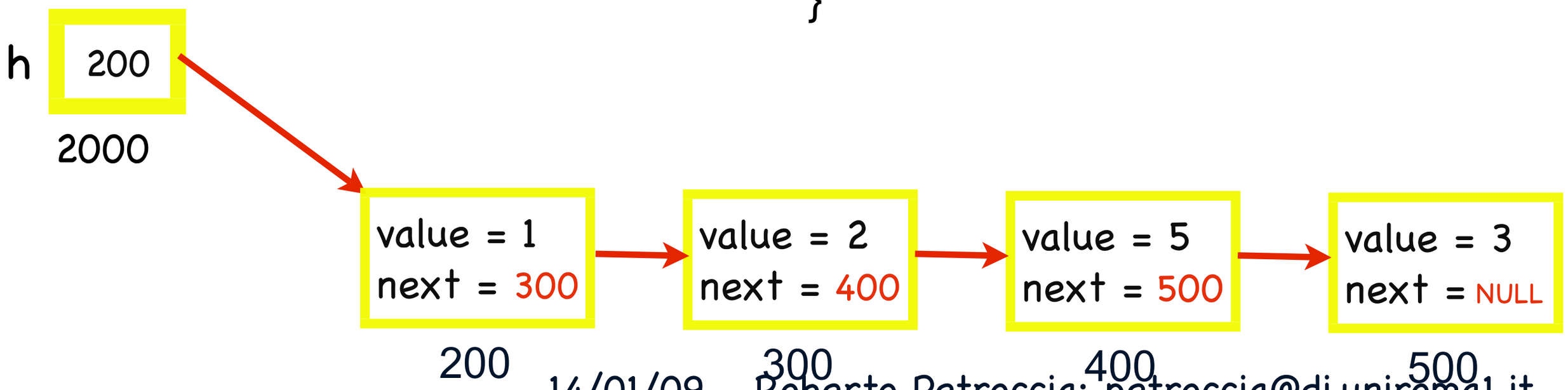
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 5

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

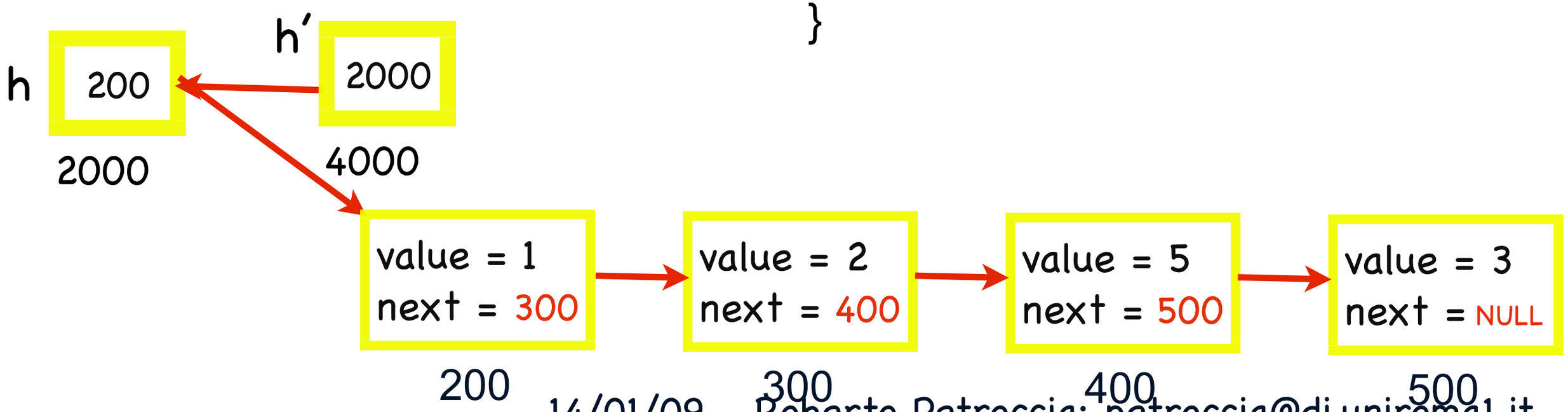
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 5

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



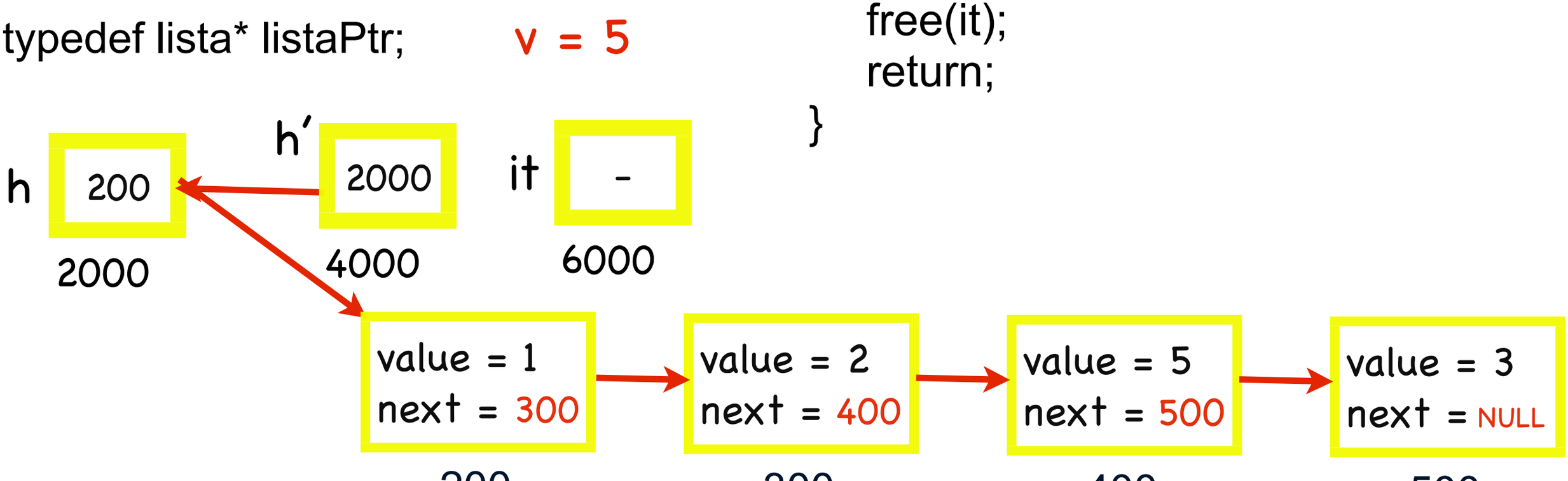
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



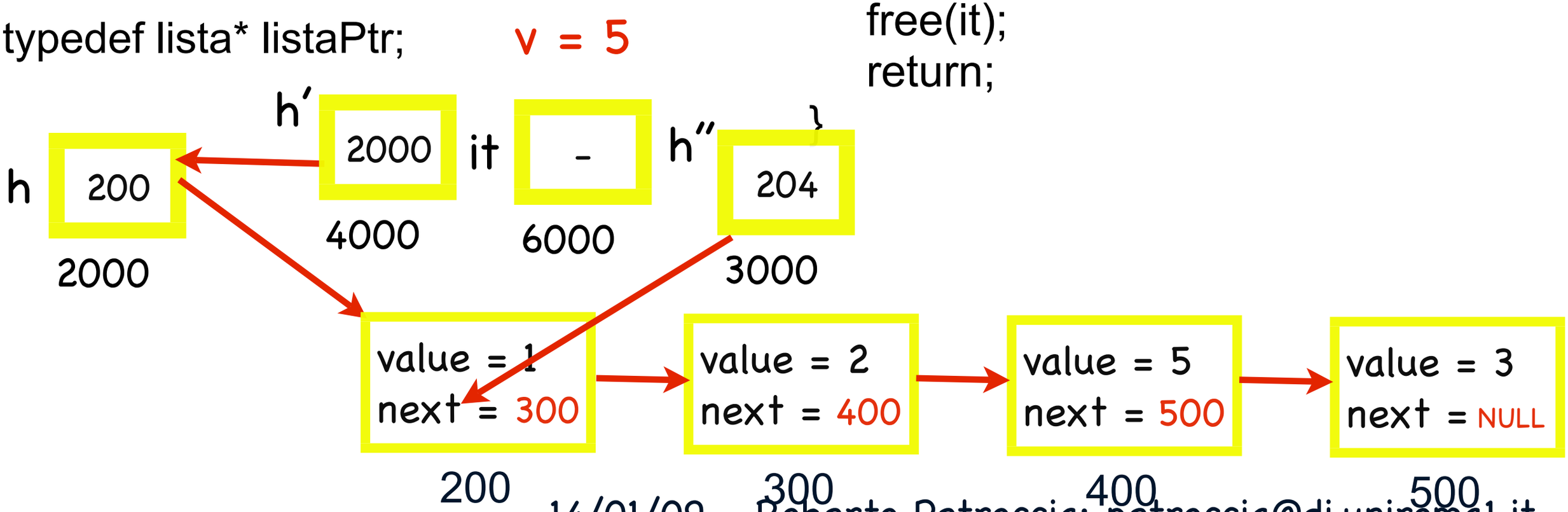
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



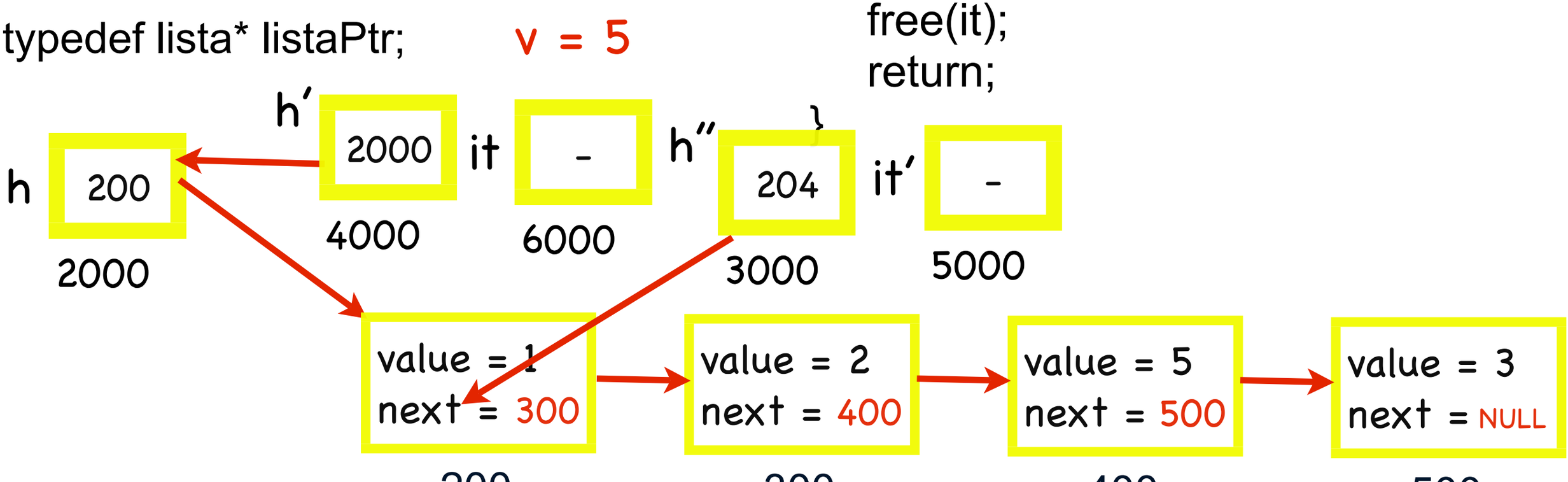
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



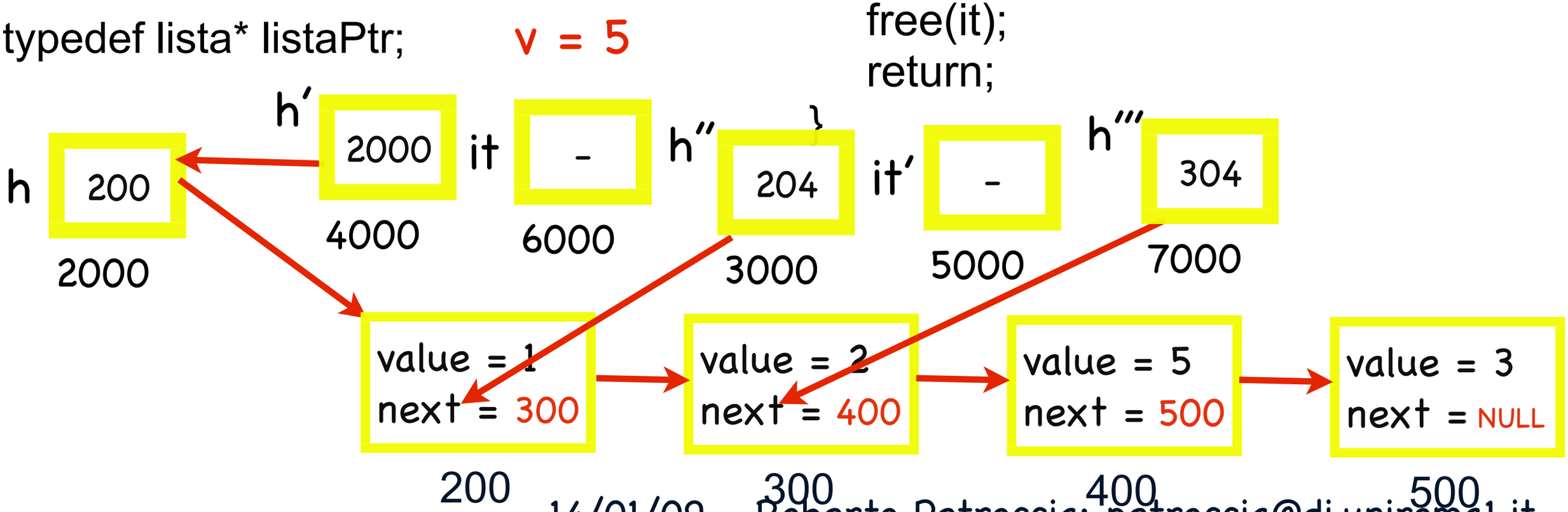
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



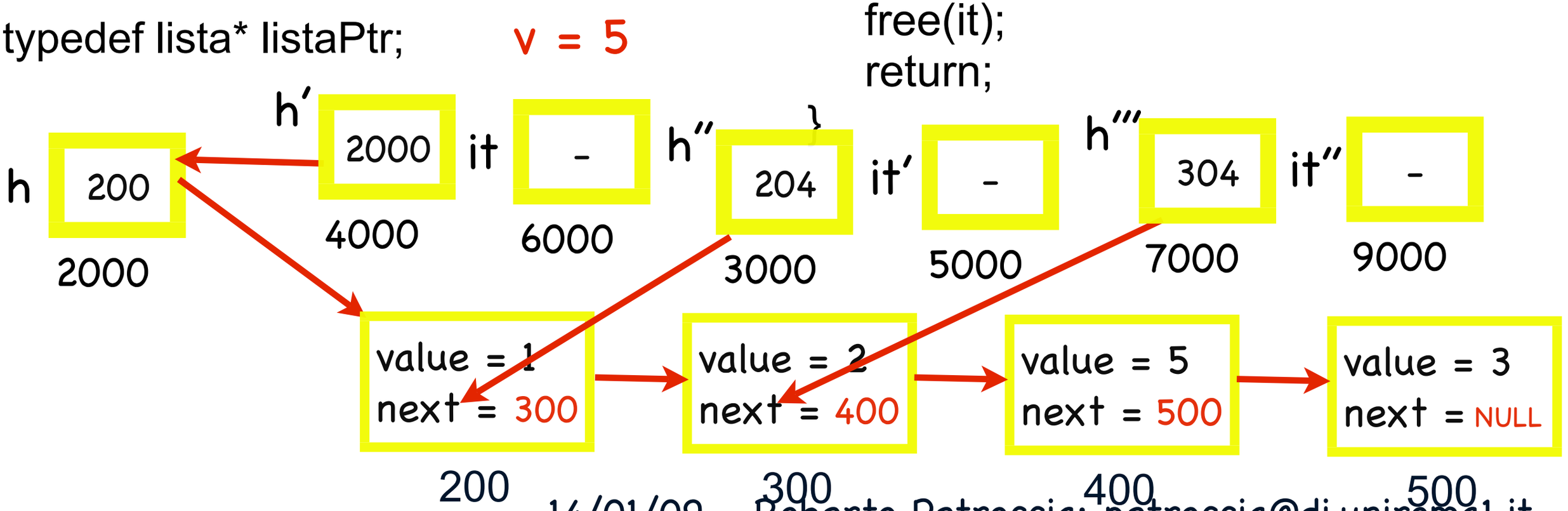
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



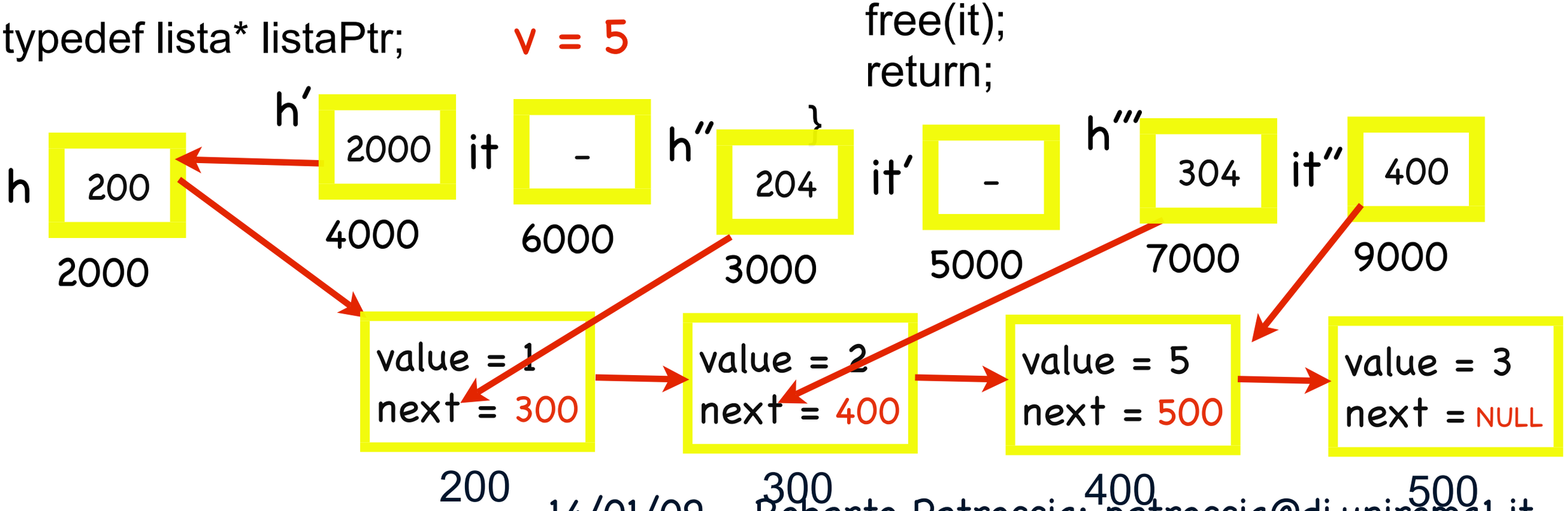
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



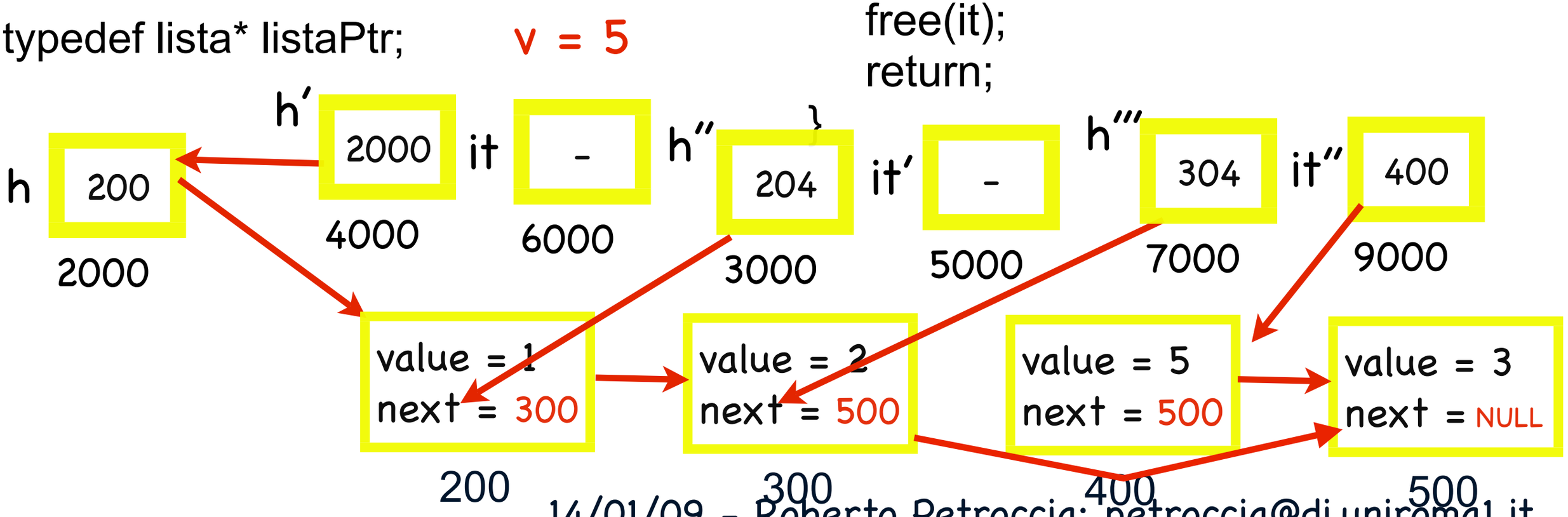
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



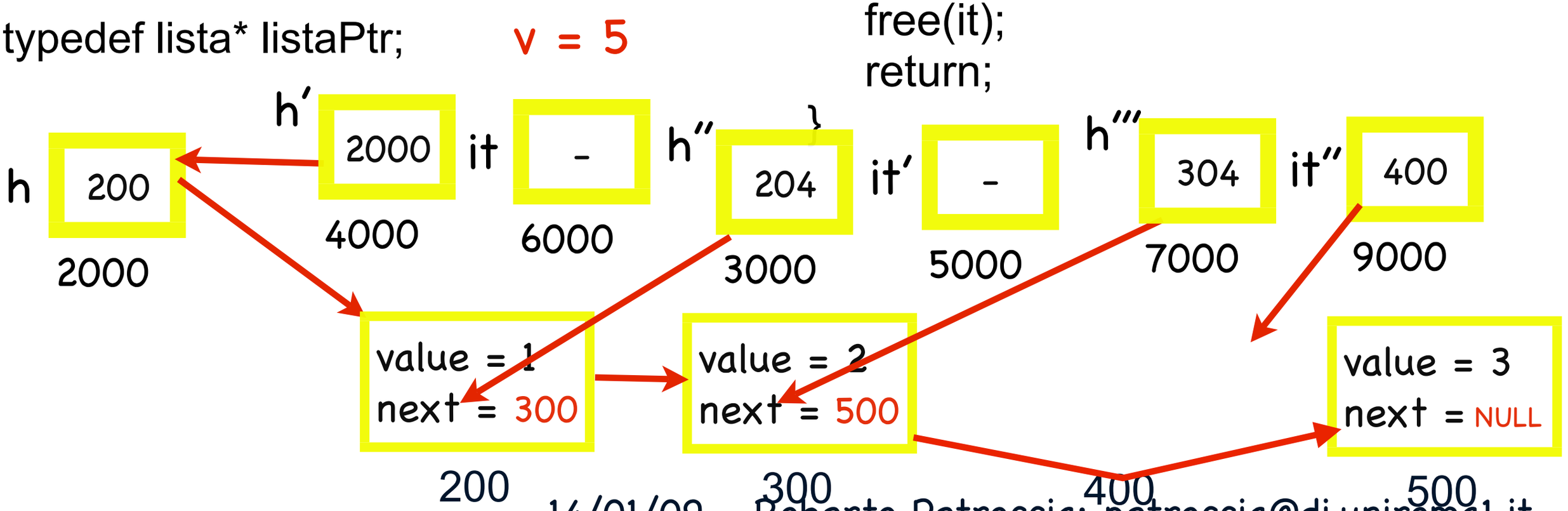
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



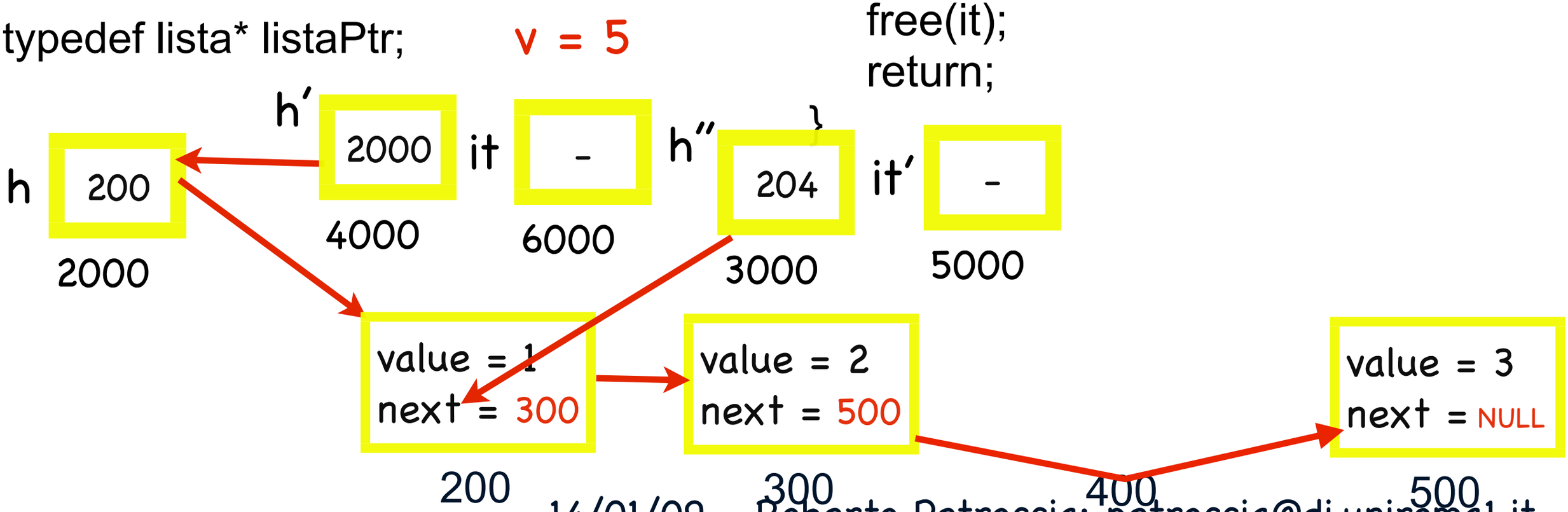
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



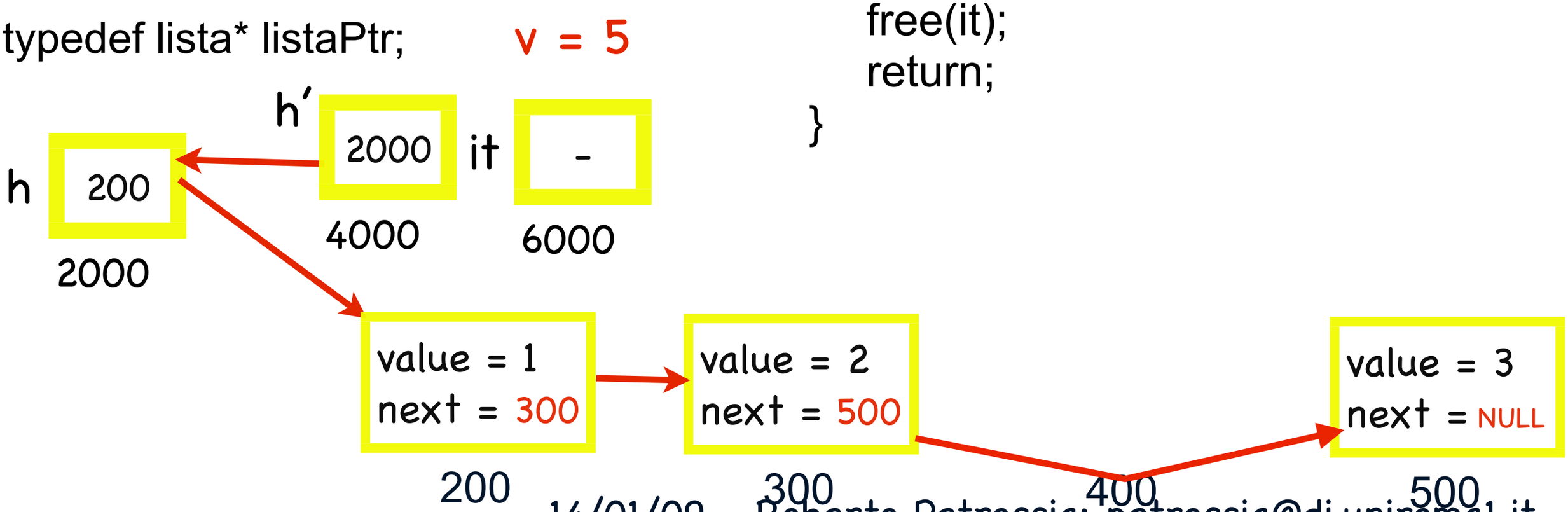
LISTE

Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

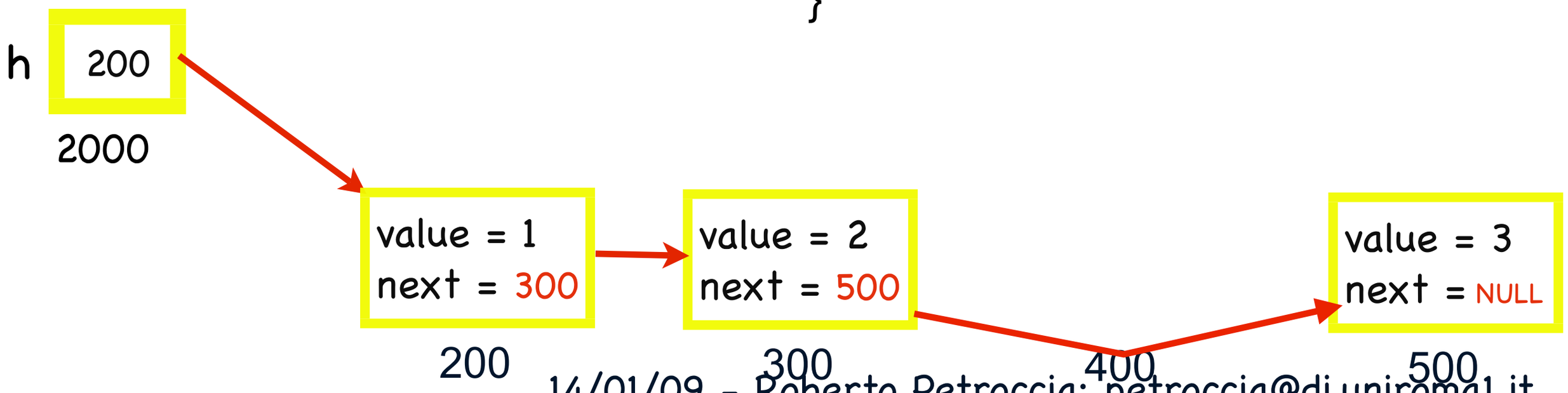
Elimina prima occorrenza

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

v = 5

```
void eraseFirstRic(listaPtr* h, int v) {
    if (*h == NULL) {
        return;
    }
    listaPtr it;
    if ((*h)->value != v) {
        eraseFirstRic(&((*h)->next), v);
        return;
    }
    it = *h;
    *h = (*h)->next;
    free(it);
    return;
}
```



LISTE

Elimina tutte occorrenze

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseAll(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr aux;
    listaPtr prev;
    prev = h;
    listaPtr it = h->next;
    while (it != NULL) {
        if (it->value == v) {
            listaPtr aux = it;
            prev->next = (it->next);
            free(aux);
            it = prev->next;
        }
        else {
            prev = it;
            it = it->next;
        }
    }
    if (h->value == v) {
        return h->next;
    }
    return h;
}
```

LISTE

Elimina tutte occorrenze

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
listaPtr eraseAllRic2(listaPtr h, int v) {
    listaPtr temp;
    if (h == NULL) {
        return h;
    }
    if ((h)->value == v) {
        temp = h;
        h = (h)->next;
        free(temp);
        return eraseAllRic2(h, v);
    }
    h->next = eraseAllRic2(h->next, v);
    return h;
}
```

LISTE

Elimina tutte occorrenze

dell'elemento v

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

```
void eraseAllRic(listaPtr* h, int v) {
    listaPtr temp;
    if (*h == NULL) {
        return;
    }
    if ((*h)->value == v) {
        temp = *h;
        *h = (*h)->next;
        free(temp);
        eraseAllRic(h, v);
        return;
    }
    eraseAllRic(&((*h)->next), v);
    return;
}
```

DOMANDE???

14/01/09 - Roberto Petroccia: petroccia@di.uniroma1.it

Esercizi su LISTE

1. Scrivere una funzione che calcoli la lunghezza (numero di elementi) di una lista
2. Scrivere una funzione che elimina le ripetizioni di uno stesso elemento dalla lista
3. Scrivere una funzione che fonde due liste senza l'allocazione di nuova memoria
4. Scrivere una funzione che inverte una lista di interi

TUTTE LE FUNZIONI POSSONO ESSERE IMPLEMENTATE IN MANIERA ITERATIVA E RICORSIVA

14/01/09 - Roberto Petroccia: petroccia@di.uniroma1.it