

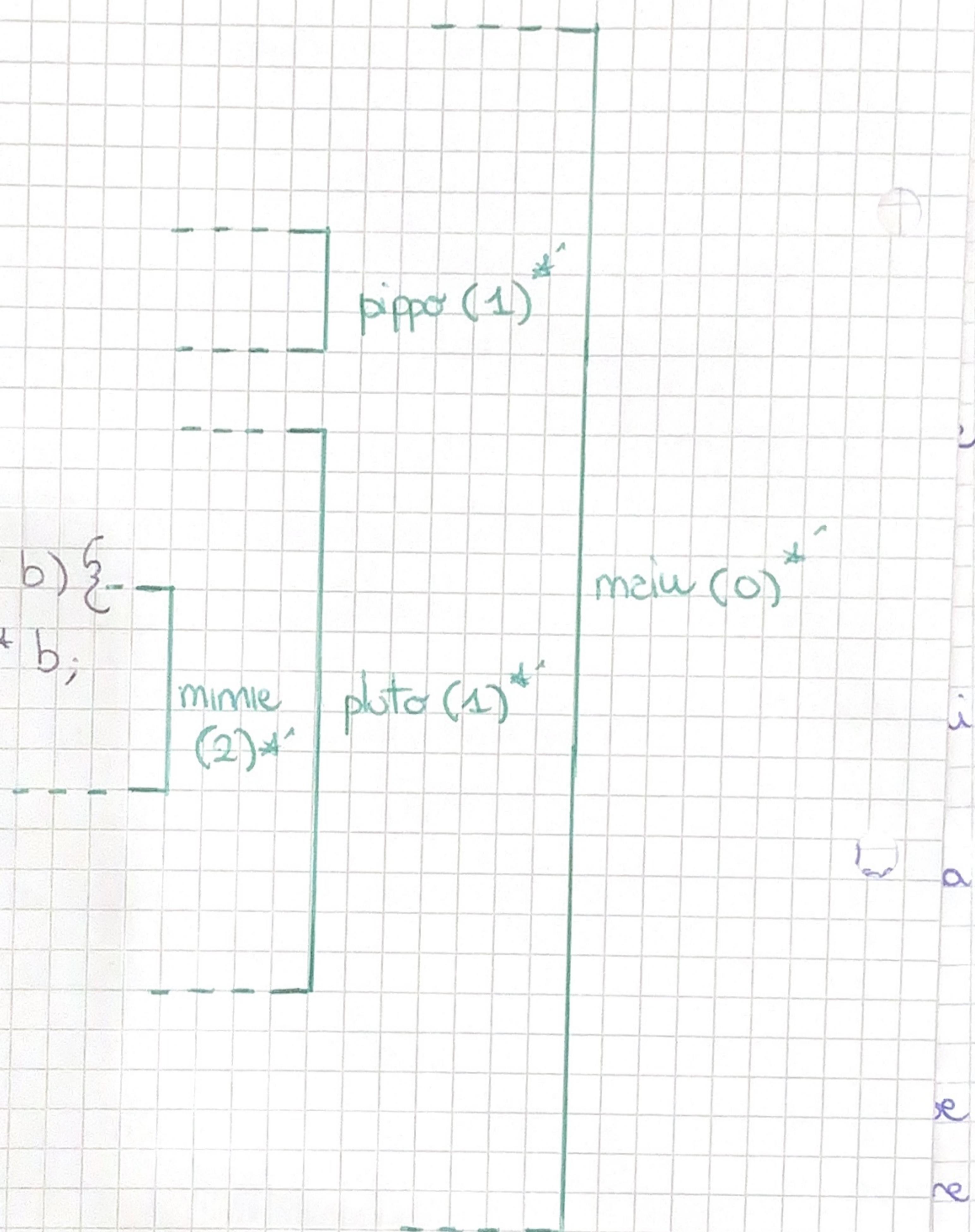
ESEMPIO

ESEMPIO 1) Si consideri il seguente programma. Si dica cosa viene calcolato dall'assegnamento in corso di scoping statico (mostrando come vengono calcolati i link statici e come vengano risolti i riferimenti non locali) e in caso di scoping dinamico (mostrando l'evoluzione della CRT e come vengano risolti i riferimenti non locali). Codice:

RIGHE

```

1. {
2.     int a = 2;
3.     void pippo() {
4.         a = a + 1;
5.     }
6.     void pluto() {
7.         a = 1;
8.         void minnie(int b) {
9.             int a = 2 * b;
10.            pippo();
11.        }
12.    pippo();
13.    minnie(3);
14. }
15. a = 3;
16. pluto();
17. }
```



* tra parentesi il valore di Sd;

Esistono due tipologie di scope:

- (a) SCOPE STATICO, in cui un nome non locale è risolto nel blocco di che testualmente lo ricopre;
- (b) SCOPE DINAMICO, in cui un nome non locale è risolto nella chiamata attivata più di recente e non ancora terminata.

La CRT (tabella centrale dei riferimenti) si riferisce allo scope dinamico.
Quest'ultimo, a livello pratico, può essere implementato con una tecnica
di shallow access, nella quale le variabili locali sono poste in delle
strutture dati centrali. Ed ecco che entra in gioco la CRT, una tabella
con le seguenti caratteristiche:

- 1) una entry è variabile nel programma;
- 2) ogni entry ha un puntatore ad una lista di elementi;
- 3) ogni lista di elementi contiene informazioni necessarie per
accedere ad un eventuale ambiente di riferimento.

SCOPING STATICO:

PASSO 1) si calcola la profondità del codice dato. Si indica con
 Sd la profondità statica, ovvero la profondità di
annidamento della definizione della procedura corrispondente
al RDA (record di attivazione, pila/stack).

Vedi parte verde sul codice!

PASSO 2) si procede con l'esecuzione del codice, contemporaneo, ad ogni
chiamata, i link statici e mostrando come vengono
risolti i riferimenti sui locali. Si ricorda che la formula
per calcolare i link statici è: $Sd(\text{STATIC DEEP})$

$$K = Sd(\text{chiamante}) - Sd(\text{chiamato}) + 1;$$

Mentre i link statici dipendono dalle dichiarazioni effettive
e dalla formula di sopra, i link dinamici puntano sempre
al chiamante, così da formare la catena dinamica.

Inoltre, il numero di volte (N) necessario per risalire la catena
statica è dato dalla differenza tra la profondità statica della
procedura dove viene utilizzato l'identificatore, meno la
 (P)
profondità statica della procedura che definisce tale identificatore.

$$N = Sd(P) - Sd(D)$$

In quei casi si indica l'abbreviazione di catena statica, avere la catena di link statici che collegano le istanze del RDA.

PASSO 3) Ora abbiamo tutto il necessario per autorizzare i link statici e i valori progressivi.

② maius invoca la funzione pluto:

$$\text{LINK STATICO} \rightarrow Sd(\text{main}) - Sd(\text{pluto}) + 1 = 0 - 1 + 1 = 0$$

$$CS(\text{pluto}) = \text{indirizzo}(\text{main})$$

$$\text{Riferimento maius locale di 'a'} \rightarrow N = Sd(\text{pluto}) - Sd(\text{main})$$

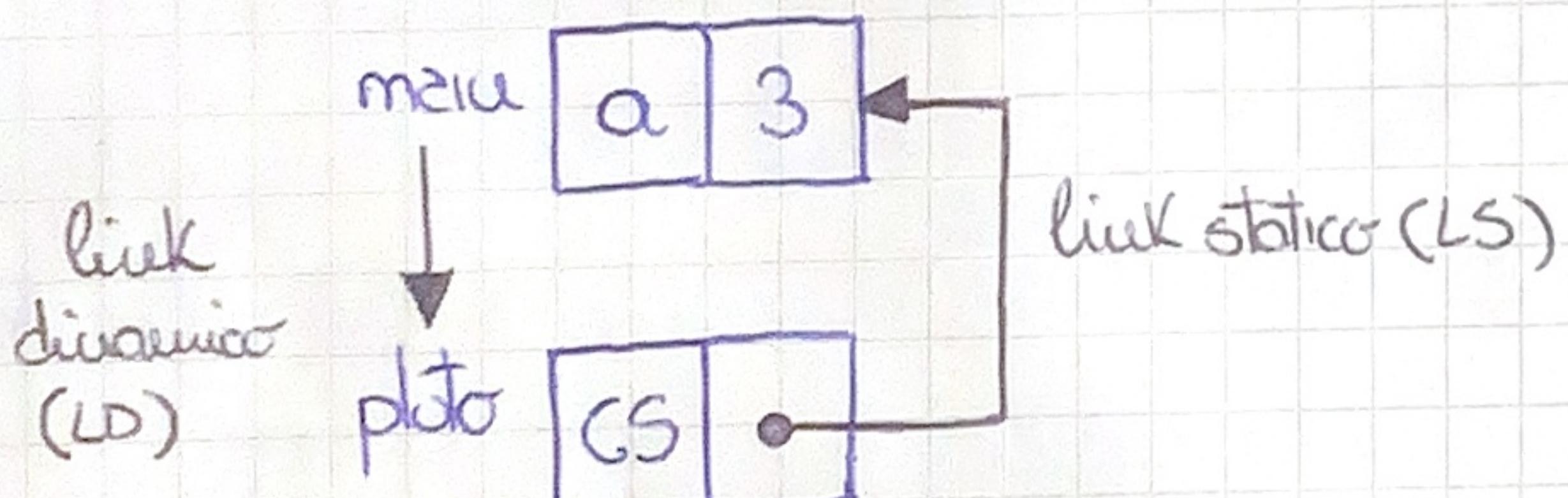
INITIO:

main	a	2
------	---	---

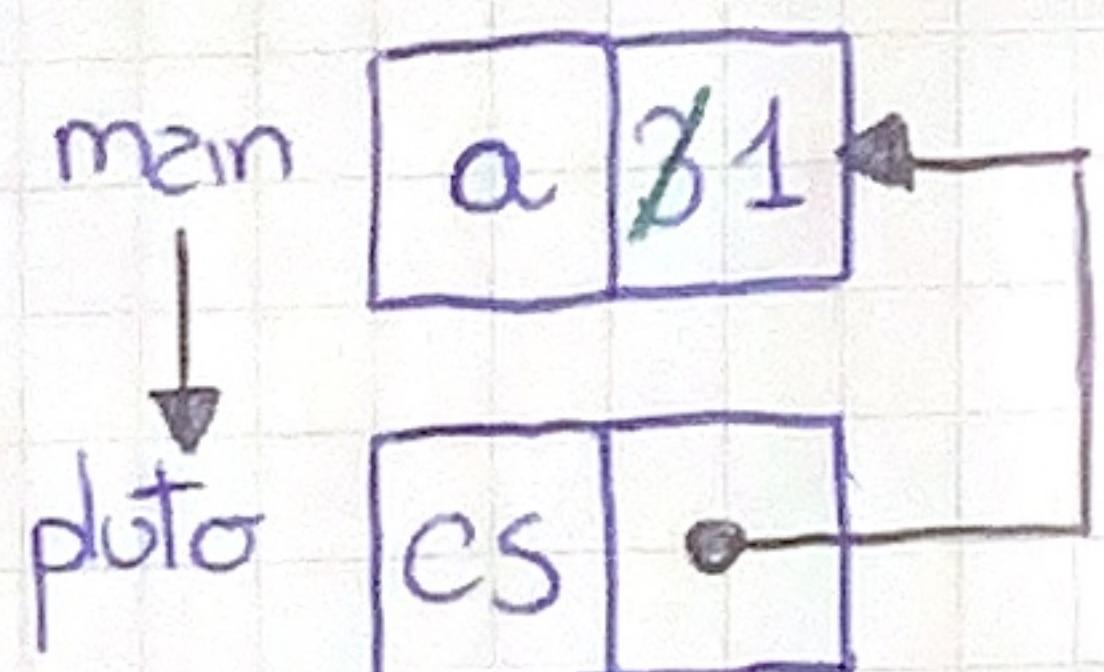
$$= 1 - 0 = 1$$

main	a	3
------	---	---

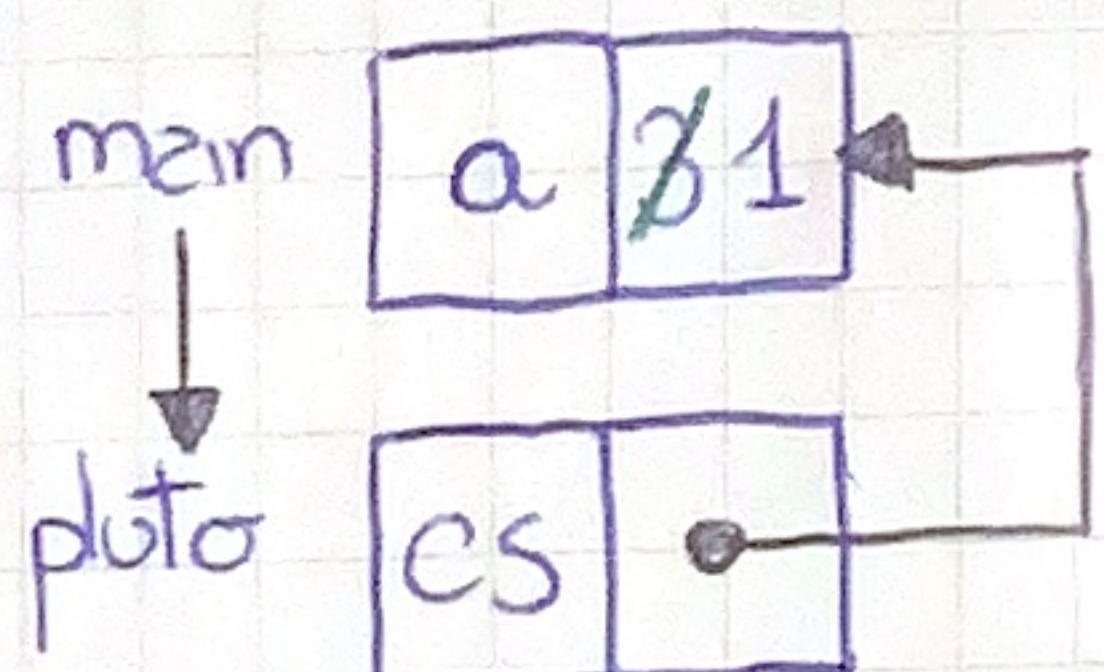
RIGA 15 (sempre modifica del main):



RIGA 16 (main chiama pluto):



RIGA 7 (pluto modifica 'a', che essendo reusa dichiarazione, fa riferimento alla 'a' del main):



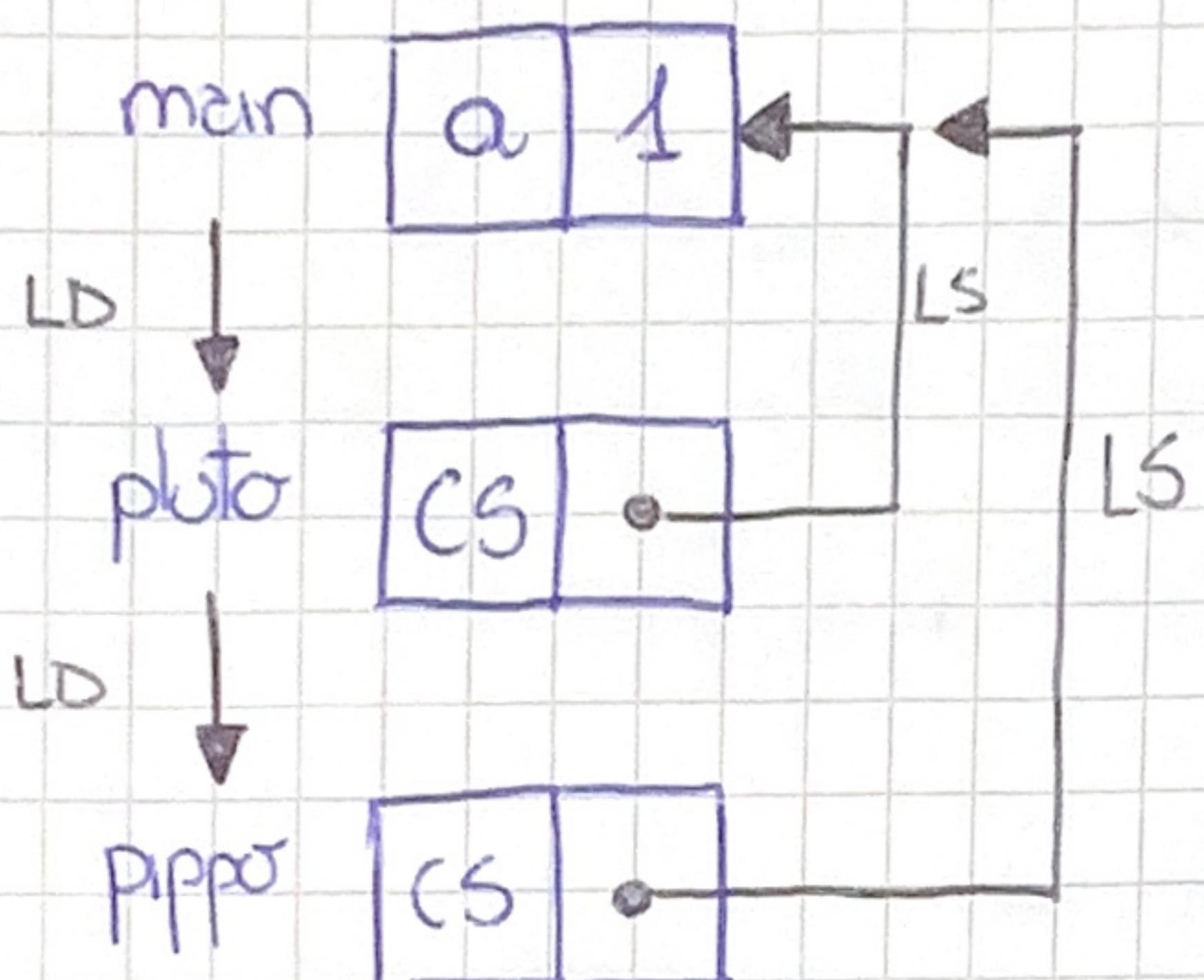
b) pluto invoca la funzione pippo :

$$\text{LINK STATICO} \rightarrow Sd(\text{pluto}) - Sd(\text{pippo}) + 1 = 1 - 1 + 1 = 1 ;$$

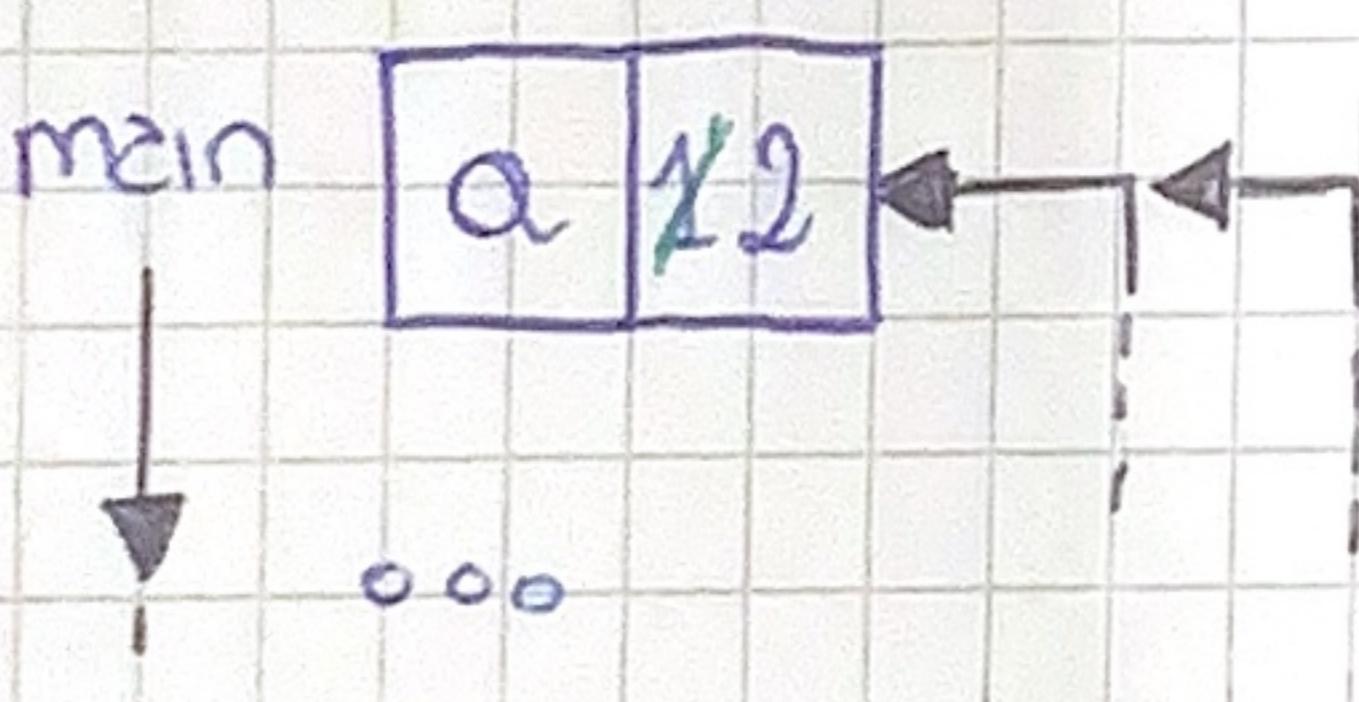
$$CS \rightarrow CS(\text{pippo}) = \text{indirizzo}(CS(\text{pluto})) = \text{indirizzo}(\text{main}) ;$$

$$\begin{aligned} \text{Riferimento mai locale 'a'} \rightarrow Sd(\text{pippo}) - Sd(\text{main}) \\ = 1 - 0 = 1 ; \end{aligned}$$

RIGA 12 (da pluto viene invocato pippo) :



RIGA 4 (dentro pluto, viene incrementata la 'a' mai locale) :



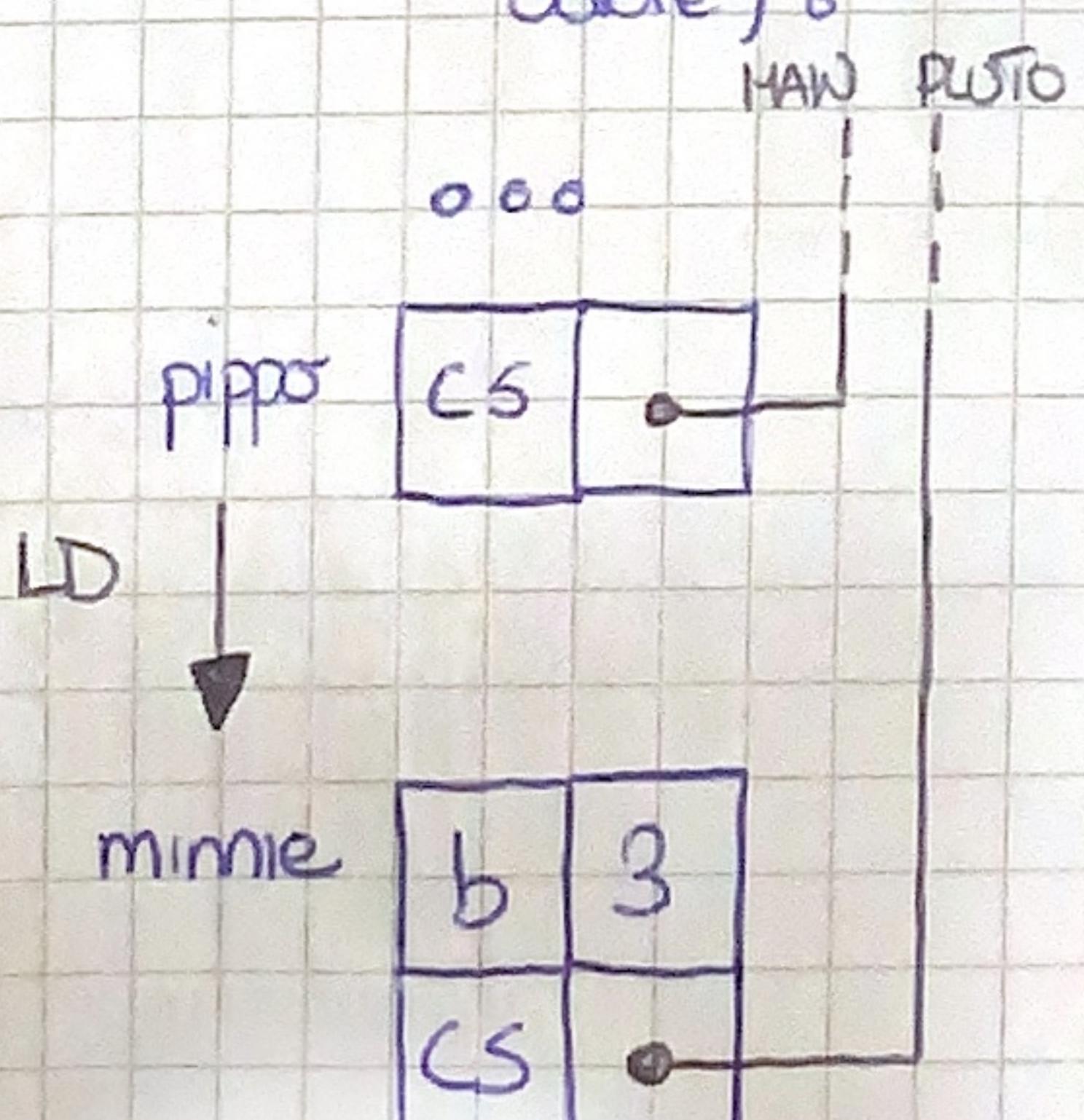
c) pluto invoca la funzione minnie

$$\text{LINK STATICO} \rightarrow Sd(\text{pluto}) - Sd(\text{minnie}) + 1 = 1 - 2 + 1 = 0 ;$$

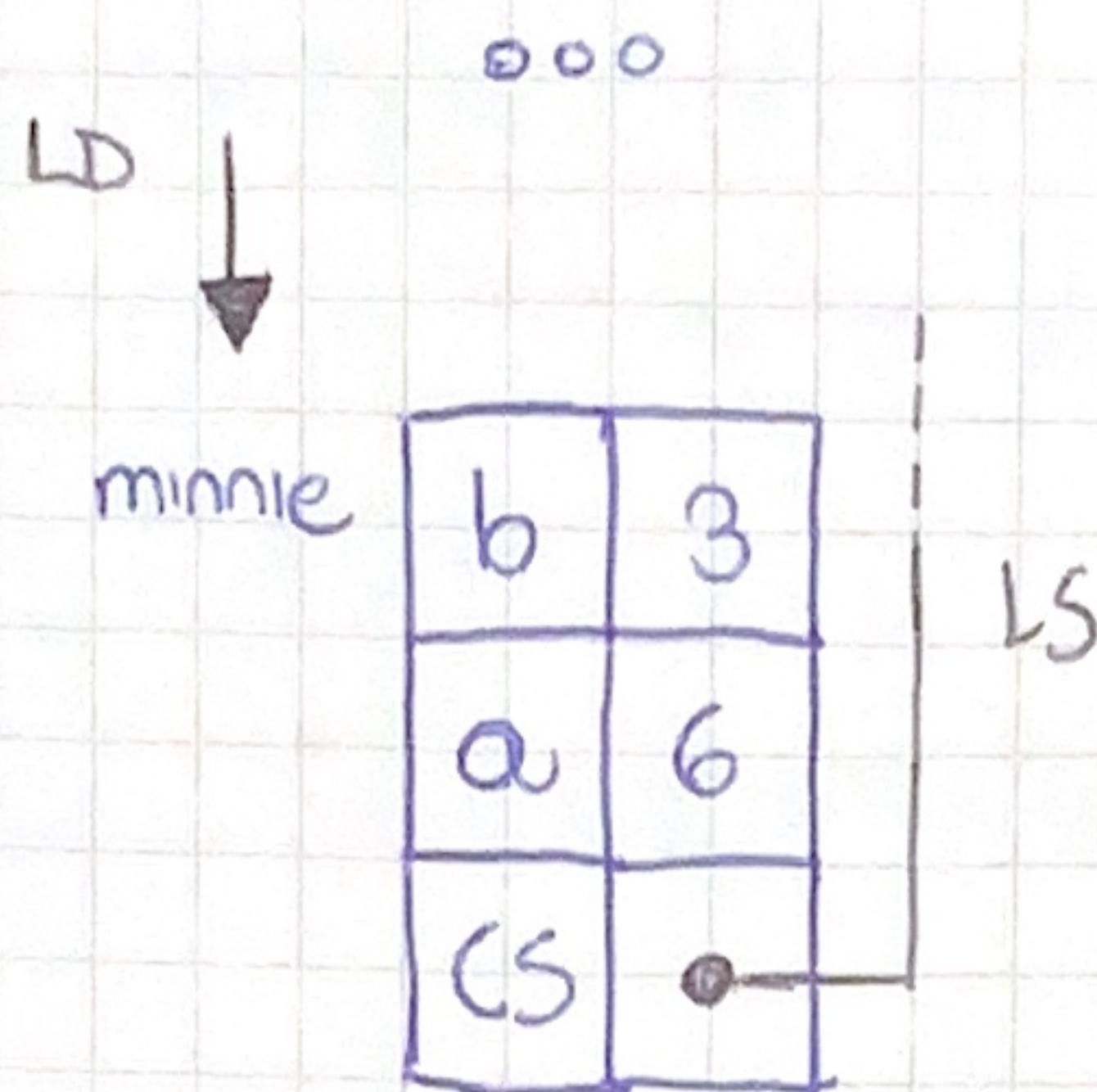
$$CS \rightarrow CS(\text{minnie}) = \text{indirizzo}(\text{pluto})$$

Riferimento mai locale \rightarrow NESSUNO (a non è stata),

RIGA 13 (vengono invocate minnie con passaggio di parametro per valore) :



RIGA 9 (venerdì dichiarata una var. 'a' che localmente riapre l'altro)



(d) minnie invoca B facendo pippo :

$$\text{LINK STATICO} \rightarrow S_d(\text{minnie}) - S_d(\text{pippo}) + 1 = 2 - 1 + 1 = 2;$$

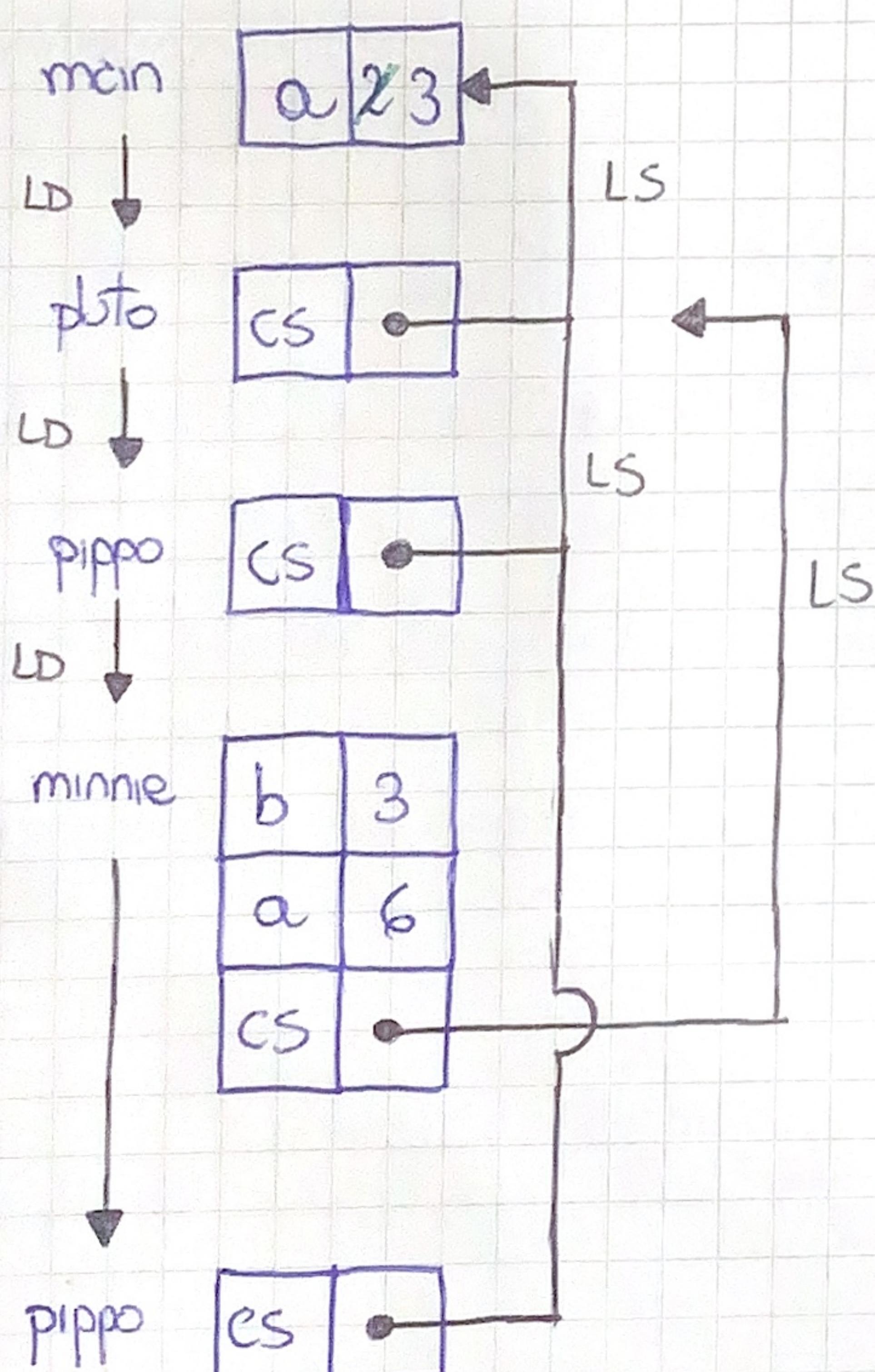
$$CS(\text{pippo}) \rightarrow \text{indirizzo}(\text{minnie});$$

LS da indirizzo (CS(minnie))

Riferimento es. locale 'a' :

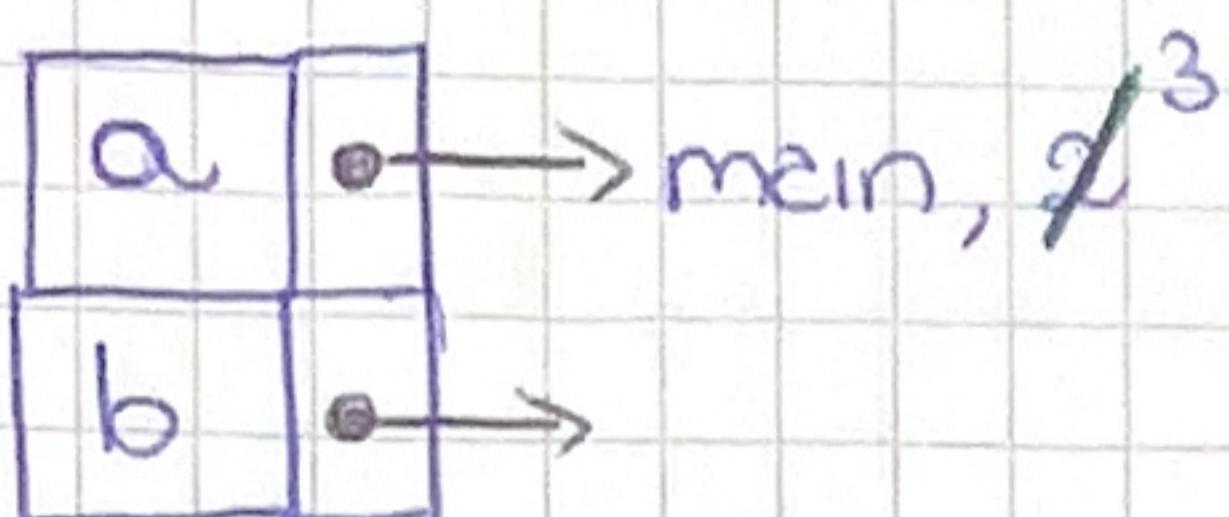
$$N = S_d(\text{pippo}) - S_d(\text{min}) = 1 - 0 = 1;$$

RIGA 4 (pippo opera l'ultima op. sulla 'a' globale, quindi la prossima figura è quella finale)

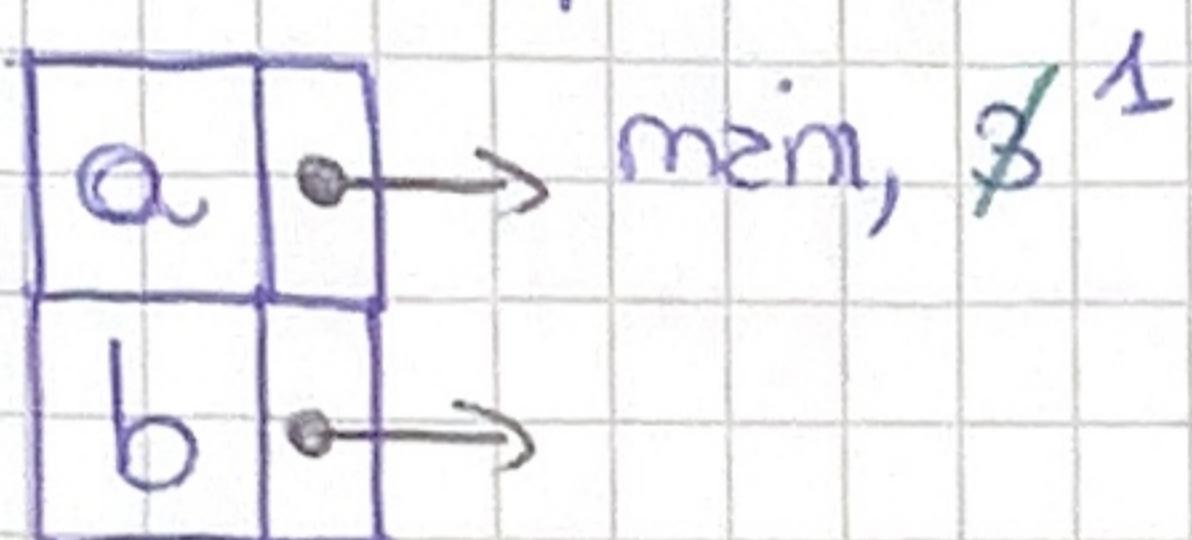


SCOPO DIAMICO :

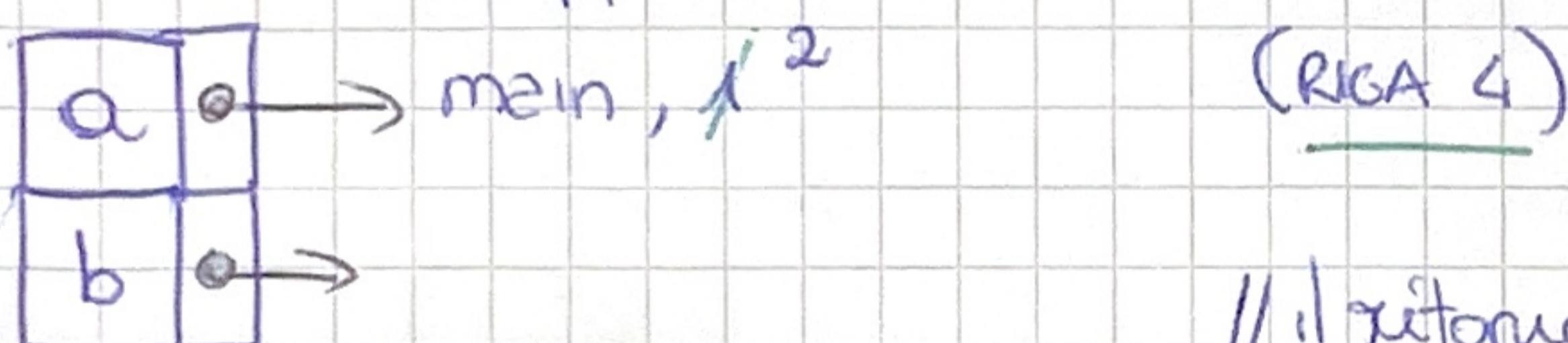
- Ⓐ Il main inizializza $a=2$ e poi aggiorna a 3 :



- Ⓑ main invoca pluto (RIGA 16) che imposta $a=1$:

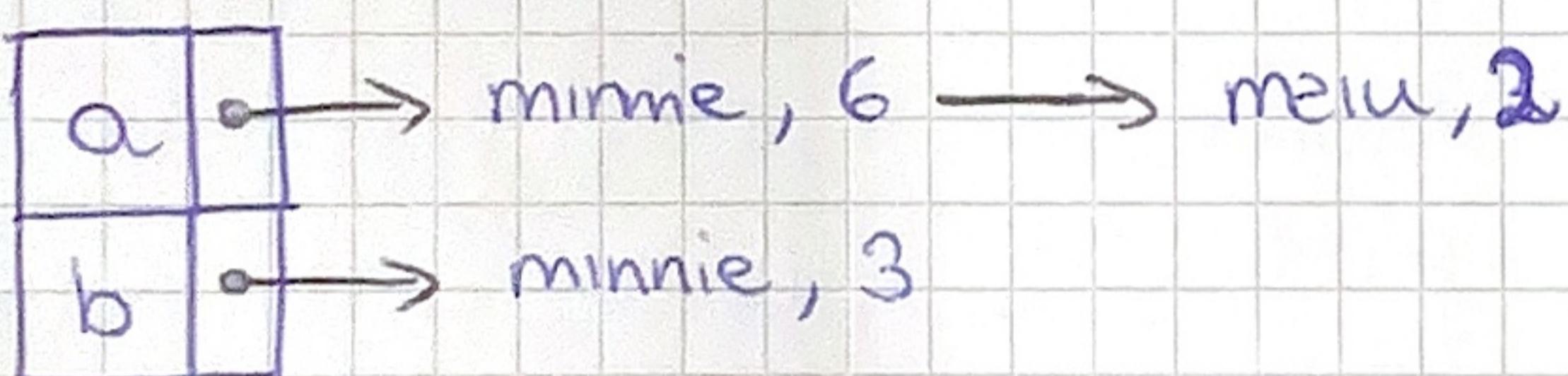


- Ⓒ pluto invoca pippo, quindi a viene incrementata di uno:

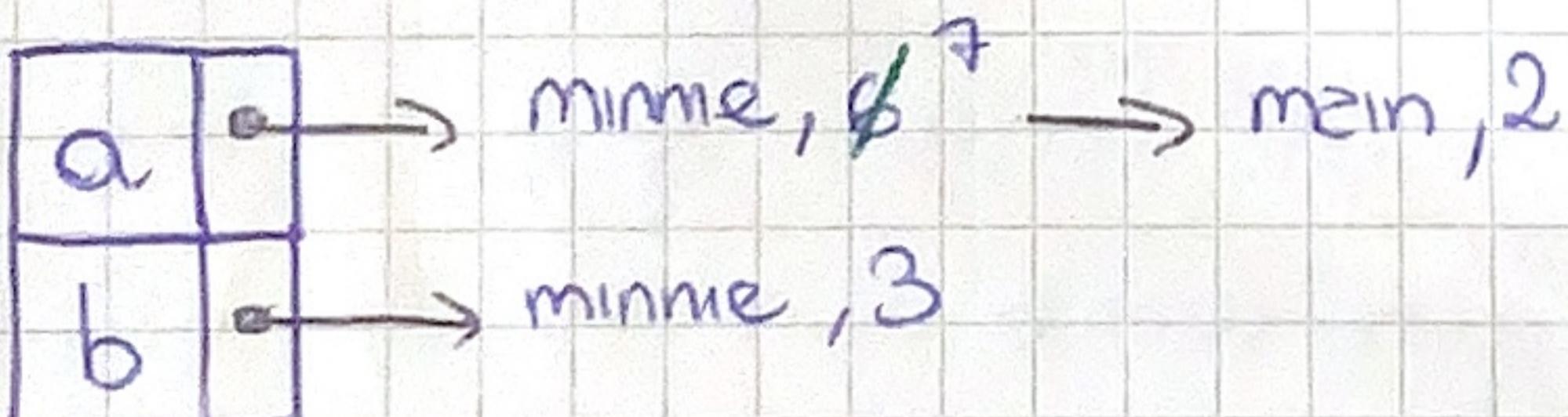


// il ritorno da pippo non cancella nulla dalla CPT dato che pippo con tutte variabili non locali

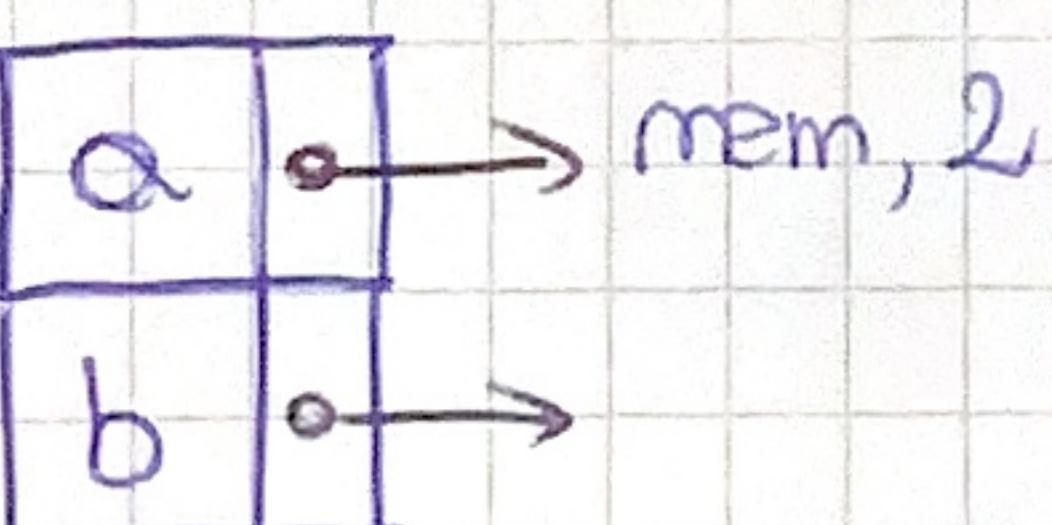
- Ⓓ pluto invoca minnie, istanziando b e facendo una nuova var. 'a':



- Ⓔ minnie invoca pippo, che incrementa b 'a' passata dinamicamente (shallow binding) ossia b 'a' di minnie e non quella globale:



- Ⓕ ritorno da minnie: gli elementi 'a' e 'b' di minnie vengono riavvolti perché la chiamata si è cancellata. Quindi segue la CPT finale:



Esercizio 2) Descrivere brevemente cosa sia un ambiente dinamico, la memoria e le derivazioni della semantica dinamica per assegnamento e per espressioni, risolvendo la seguente derivazione:

$$\rho = [x \mapsto 2, y \mapsto ly], \quad \sigma = [ly \mapsto 4]$$

$$\text{dove } y := (x+4)^* y;$$

Nella semantica dinamica, per descrivere l'esecuzione di un programma, è fondamentale tenere traccia del suo stato. Questo stato è composto principalmente da due elementi:

1) AMBIENTE DINAMICO: l'ambiente è una fn. o una mappa che associa -
(' ρ ') indirizzi delle domande, ma c'è anche statico!

-- i nomi delle variabili (gli identificatori) alle loro locazioni di memoria. In alcuni casi può associare anche i nomi ai colori, ma è maggiormente diffusa la versione detta.

È 'dinamico' nel senso che la sua composizione può cambiare durante l'esecuzione del programma.

es: $\rho = [x \mapsto lx]$ significa che la var. 'x' è associata alla locazione di memoria lx ;

2) MEMORIA: la memoria, o store, è una funzione che associa le locazioni di memoria agli effettivi valori contenuti. Cambia ogni volta che una var. viene assegnata ad un nuovo valore, oppure quando viene allocate / deallocate memorie per struttura dati.

es: $\sigma = [lx \mapsto 5]$ significa che nella locazione lx è contenuto il valore 5;

La semantica dinamica può essere, infine, fatta per assegnamento o per espressioni. Nel 1° caso, durante la derivazione viene descritto anche

Come gli assegnamenti modificano la memoria, mentre nel 2° caso, le espressioni vengono valutate per produrre un valore.

ESERCIZIO:

1) REGOLA INIZIALE:

$$\frac{P \vdash <((x+4)^*y), \sigma> \xrightarrow{*} (k_0, \sigma)}{P \vdash <y^{\circ} = (x+4)^*y, \sigma> \longrightarrow <y^{\circ} = k_0, \sigma>}$$

2) VALUTIAMO L'ESPRESSONE: valutazione di $(x+4)^*y$

$$\frac{P \vdash <(x+4), \sigma> \xrightarrow{*} (k_1, \sigma)}{P \vdash <(x+4)^*y, \sigma> \longrightarrow <k_1 * y, \sigma>}$$

3) VALUTIAMO L'ESPRESSONE: valutazione di $(x+4)$

$$\frac{P \vdash <(x+4), \sigma> \longrightarrow <\textcircled{2}, \sigma>} {P \vdash <(x+4), \sigma> \longrightarrow <2+4, \sigma>} \quad \Rightarrow P(x) = 2$$

4) VALUTIAMO L'ESPRESSONE: calcoliamo $(2+4)$

$$P \vdash <2+4, \sigma> \longrightarrow <\underbrace{6}_{k_1}, \sigma>$$

quindi

$$P \vdash <x+4, \sigma> \xrightarrow{*} <6, \sigma> \Rightarrow k_1 = 6$$

5) TORNIAMO INDIETRO: valutiamo $k_1 * y$

$$\frac{P \vdash <y, \sigma> \longrightarrow <4, \sigma>}{P \vdash <6 * y, \sigma> \longrightarrow <6 * \textcircled{4}, \sigma>} \quad \Rightarrow P(y) = P_y$$

6) TORNIAMO INDIETRO: calcoliamo $6 * 4$

$$P \vdash <6 * 4, \sigma> \longrightarrow <\underbrace{24}_{k_0}, \sigma>$$

$$\sigma(P_y) = 4$$

$$P \mapsto \langle (x+4)*y, \sigma \rangle \rightarrow^* \langle 24, \sigma \rangle$$

$$P \mapsto \langle y := (x+4)*y, \sigma \rangle \rightarrow \langle y = 24, \sigma \rangle$$

7) ESEGUIAMO $y = 24$:

$$P \mapsto \langle y = 24, \sigma \rangle \rightarrow \sigma := [P_y \mapsto 24], P(y) = P_y.$$

Esercizio 3) Descrivere passaggio di parametri, per valore e per riferimento, ed eseguire in entrambi i modi il seg. codice:

```
void fun(int a, int b) {
```

$$a = a + 5;$$

$$b = 2b;$$

}

```
void main() {
```

$$\text{int } x = 3;$$

$$\text{int } y = 2;$$

```
fun(x,y);
```

}

Il passaggio di parametri definisce come i dati vengano trasferiti ad una funzione. Con il passaggio per valore, la funzione riceve una copia del dato, e quindi le modifiche del parametro interno non influenzano l'originale. Con il passaggio per riferimento, la funzione riceve un riferimento (indirizzo) del dato originale; le modifiche al parametro interno alterano direttamente la variabile originale.

PASSAGGIO PER VALORE:

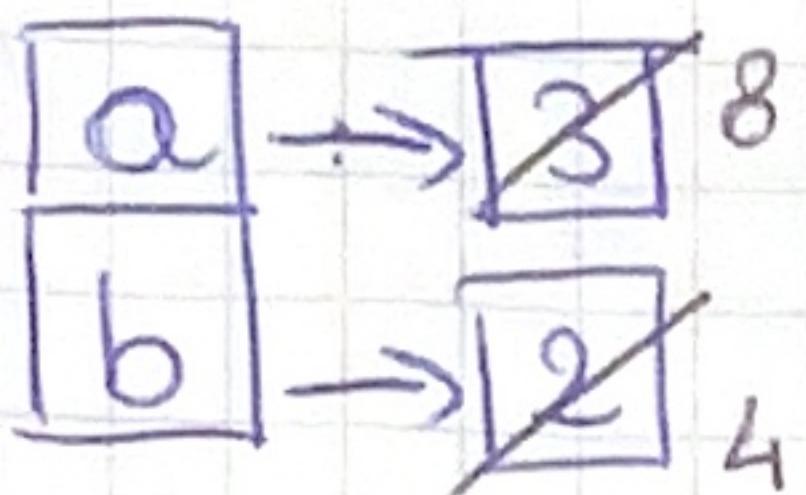
1) man:

x	→	3
y	→	2

2) chiamata a funzione fun

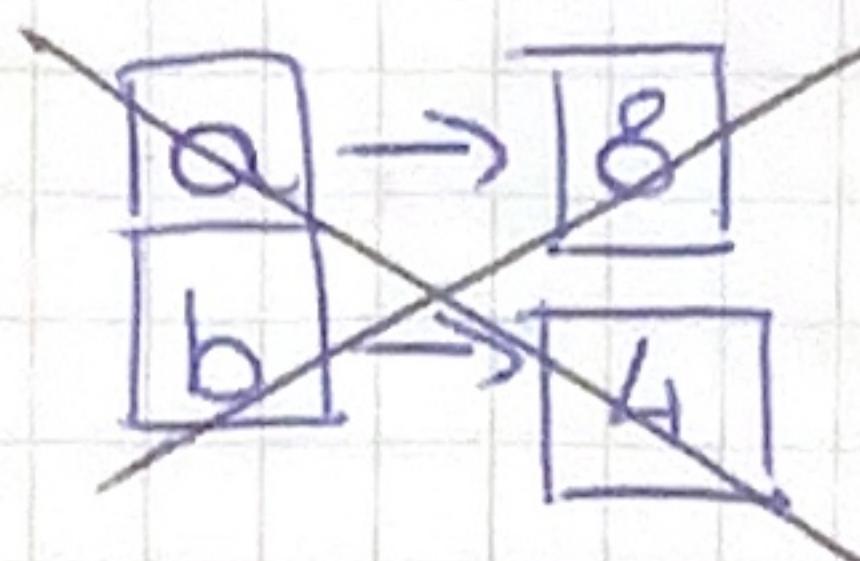
a	→	3
b	→	2

3) esecuzione fun:

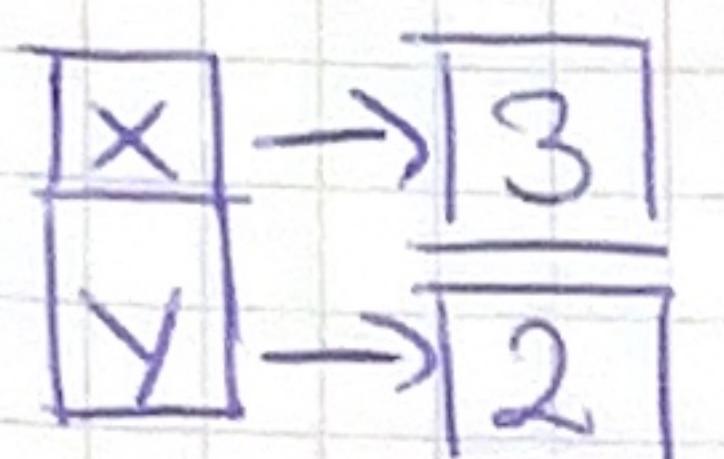


4) terminazione funzione fun:

(si eliminano a e b)

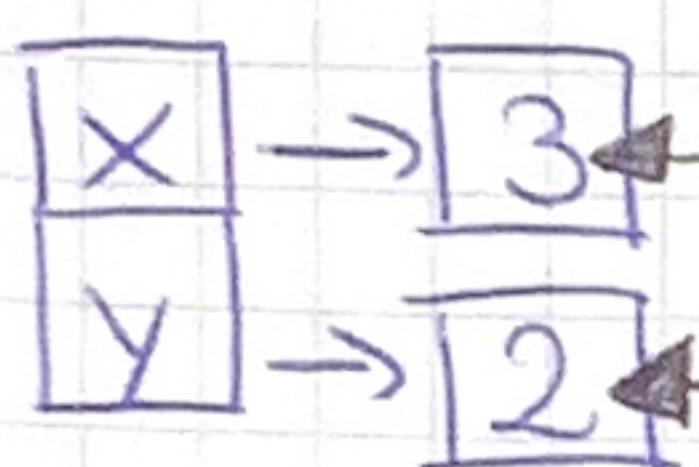


5) main finale:

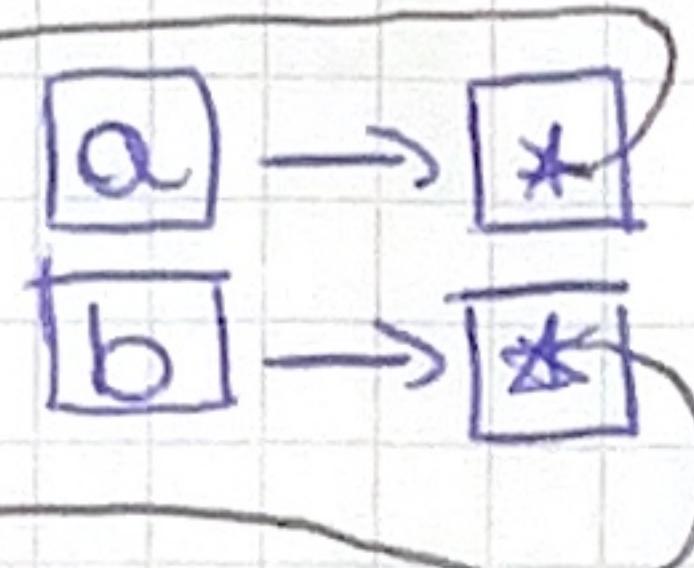


PASSAGGIO PER RIFERIMENTO:

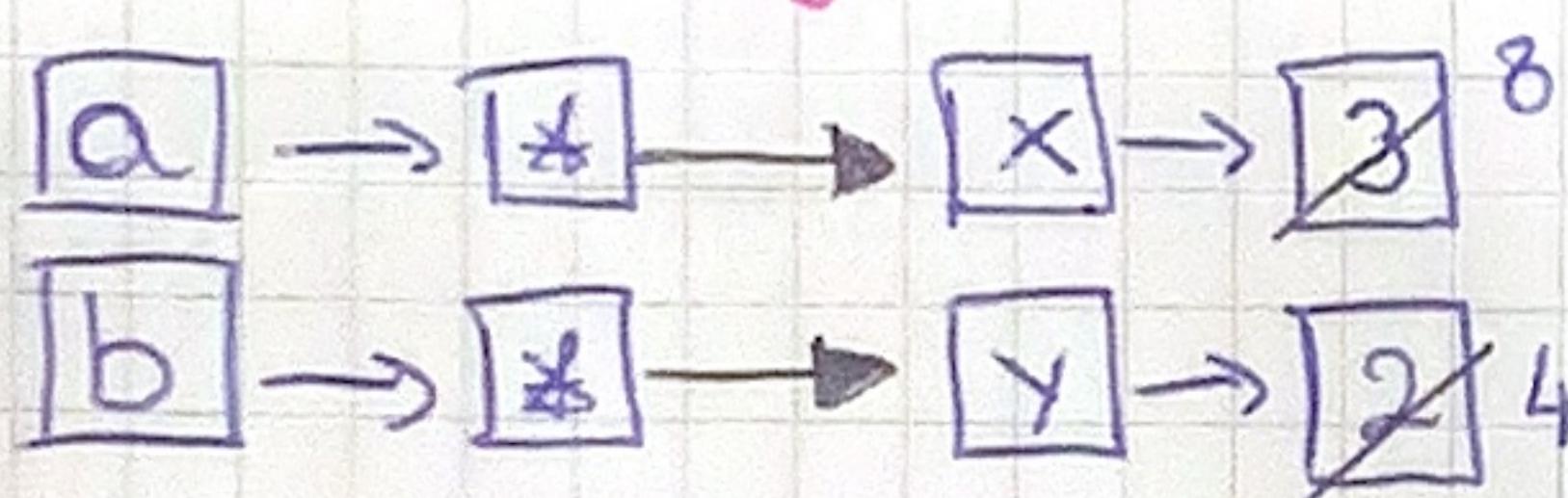
1) main:



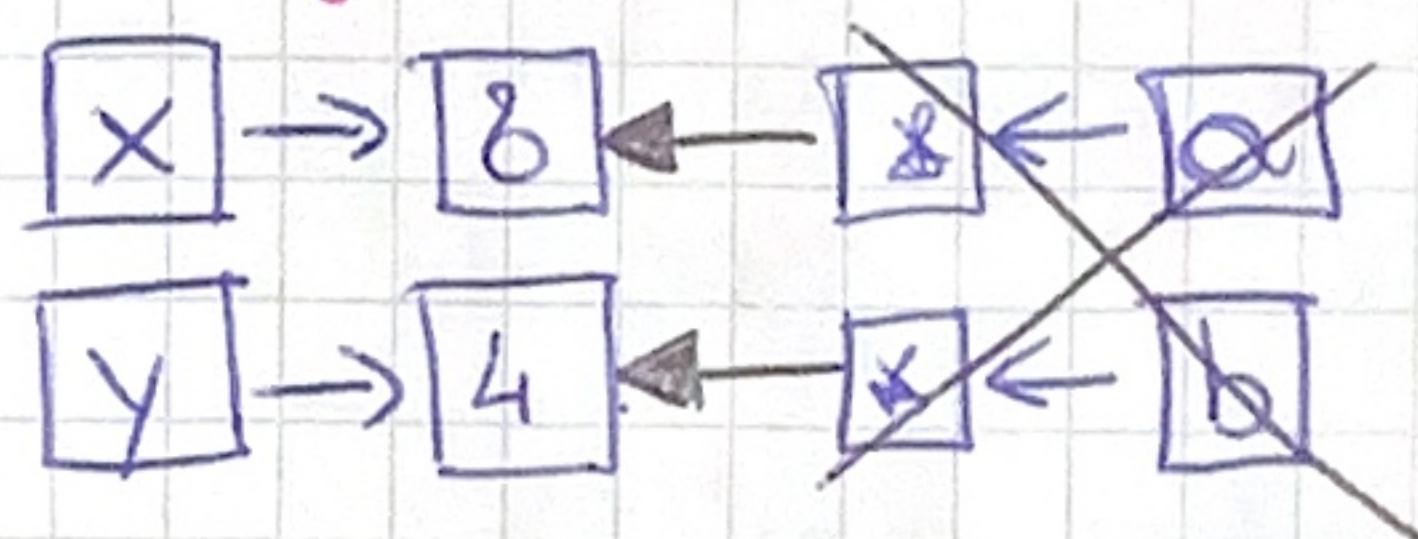
2) chiamata di fun:



3) esecuzione fun:



4) main finale (distruttore a e b):



Esercizio 4) Definire la ricorsione, la ricorsione in coda, esplicando a parole cosa fa il seguente programma, e proponendone una versione in versione 'ricorsione in coda'. Programma:

```
int function(list list) {  
    if (length(list) == 0) then  
        return 0;  
    else  
        return (car(list)) + function(cdr(list));  
}
```

non è in coda

Testiamo il programma con l'input `List(5,6,7)`:

RESET
↑

1) `function(5,6,7) → if (length = 0) NO!`

↳ else : $5 + \text{function}(6,7)$

$$13 + 5 = 18$$

2) `function(6,7) → if (length = 0) NO!`

↳ else : $6 + \text{function}(7)$

$$6 + 7 = 13$$

3) `function(7) → if (length = 0) NO!`

↳ else : $7 + \text{function}(0)$

$$7 + 0 = 7$$

4) `function(0) → if (length = 0) SI! return 0;`

Σ

Nuova funzione con 'ricorsione in coda':

```
int function(List list) {  
    return fix2(list, 0);  
}
```

```
int fix2(List list, int res) {  
    if (length(list) == 0)  
        return res;  
    else  
        fix2(cdr(list), car(list) + res);  
}
```

Il aggiunto parametro che memorizza il risultato intermedio; ora il caso base ritorna direttamente il risultato finale.

Esecuzione:

`function((5,6,7))` chiamata `fix2((5,6,7), 0);`

1) $\text{fun2}((5,6,7), 0) \rightarrow \text{if}(\text{length} == 0) \text{ NO!}$
 $\hookrightarrow \text{else: } \underline{\text{fun2}((6,7), 5+0)}$

2) $\text{fun2}((6,7), 5) \rightarrow \text{if}(\text{length} == 0) \text{ NO!}$
 $\hookrightarrow \text{else: } \underline{\text{fun2}((7), 5+6)}$

3) $\text{fun2}((7), 11) \rightarrow \text{if}(\text{length} == 0) \text{ NO!}$
 $\hookrightarrow \text{else: } \underline{\text{fun2}((), 11+7)}$

4) $\text{fun2}((), 18) \rightarrow \text{if}(\text{length} == 0) \text{ SI!} \rightarrow \text{return } \underline{\text{res} = 18}$

Esercizio 5) Descrivere ambiente statico e verificare se nell'ambiente

$\Delta = [y \mapsto \text{intloc}, x \mapsto \text{int}]$ l'assegnamento $y := (x+4)^* y$
è ben formato.

L'ambiente statico è una funzione o una mappa che associa gli identificatori (nomi) ai loro tipi di dato. La semantica statica si basa quindi sull'ambiente statico (Δ) per scoprire compiti come il controllo dei tipi. Il suo scopo è quindi garantire che le operazioni nel programma siano applicate a valori dei tipi appropriati, prevenendo errori che altrimenti si manifesterebbero solo a run-time. Perché un'espressione sia ben formata dobbiamo certificare che le operazioni sulle siano tra tipi di dati compatibili, e allo stesso tempo che gli assegnamenti vengano fatti su locazioni di memoria del tipo giusto. Esercizio:

$\Delta = [y \mapsto \text{intloc}, x \mapsto \text{int}]$

LOCAZIONE VALORE

Sappiamo quindi da $\Delta(y)$ che l'assegnamento di y avviene correttamente.

$$y := (x + 4) * y$$

assegnamento: si aspetta intloc, quindi y deve essere una locazione. Come detto, lo è!

a destra dell'assegnamento, dobbiamo avere un tipo di dato valido, in questo caso rispetto a y , ossia int

Dobbiamo quindi verificare che $\Delta \vdash (x + 4) * y : \text{int}$

$$\frac{\Delta \vdash (x+4) : \text{int} \quad \Delta \vdash y : \text{int}}{\Delta \vdash (x+4) * y : \text{int}}$$

] si legge: parte sotto
è vera sse parte sopra!

*^① $y : \text{int}$ lo sappiamo già dall'ambiente statico,
che dice $\Delta(y) = \underline{\text{intloc}}$;

Dobbiamo quindi verificare che $\Delta \vdash x + 4 : \text{int}$

$$\frac{\Delta \vdash x : \text{int} \quad \Delta \vdash 4 : \text{int}^{\circledR}}{\Delta \vdash x + 4 : \text{int}}$$

*^② vero sempre

*^③ vero dall'ambiente, $\Delta(x) = \text{int}$

Abbiamo quindi dimostrato che l'assegnamento, in questo ambiente, è ben formato.

Esercizio 6) Definire brevemente scoping (statico/dinamico), link statico e tabella CRT. Risolvere con entrambi gli scoping, raccontandone l'evoluzione, il seguente codice:

```
1. void main() {  
2.     int x;  
3.     int y;  
4.     void fun1() {  
5.         int z = x + y;  
6.     }  
7.     void fun2() {  
8.         y = y + 1;  
9.     }  
10.    void fun3() {  
11.        int y = x + 2;  
12.        fun2();  
13.        fun1();  
14.    }  
15.    x = 5;  
16.    y = 2;  
17.    fun3();  
18.}
```

Diagramma di scoping:

- main: Scoping dinamico (SD) = 0, local vars: x, y.
- fun1: SD = 1, local var: z.
- fun2: SD = 1, local var: y.
- fun3: SD = 1, local var: y.

Lo scoping è ciò che determina l'ambito di visibilità di una variabile. Esistono due strategie fondamentali di scoping, e variano appunto nel

ambiente in cui viene considerata all'interno di un certo contesto di programmazione:

- SCOPING STATICO: si guarda dove le variabili sono definite nel codice, a tempo di compilazione;
- SCOPING DINAMICO: si guarda la successione di chiamate a funzione nell'ambiente di esecuzione (run-time);

Per calcolare passo passo l'elaborazione di entrambe si utilizzano reciprocamente:

- **LINK STATICO**: puntatore / collegamento che risale i contesti lessicali calcolate tramite la formula:

$$Scd(\text{chiamante}) - Scd(\text{chiamato}) + 1 = K.$$

Se $K=0$, il link statico punta direttamente all'indirizzo del chiamante, altrimenti si risale la catena di $K=n$ passi;

- **TABELLA CRT**: tabella centrale dei riferimenti, che contiene una lista di elementi e informazioni necessarie per accedere ad un eventuale ambiente di riferimento;

ESEMPIO:

SCOPING STATICO)

- 1) main inizializza x e y , RIGA 15 e 16 :

main	x	5
	y	2

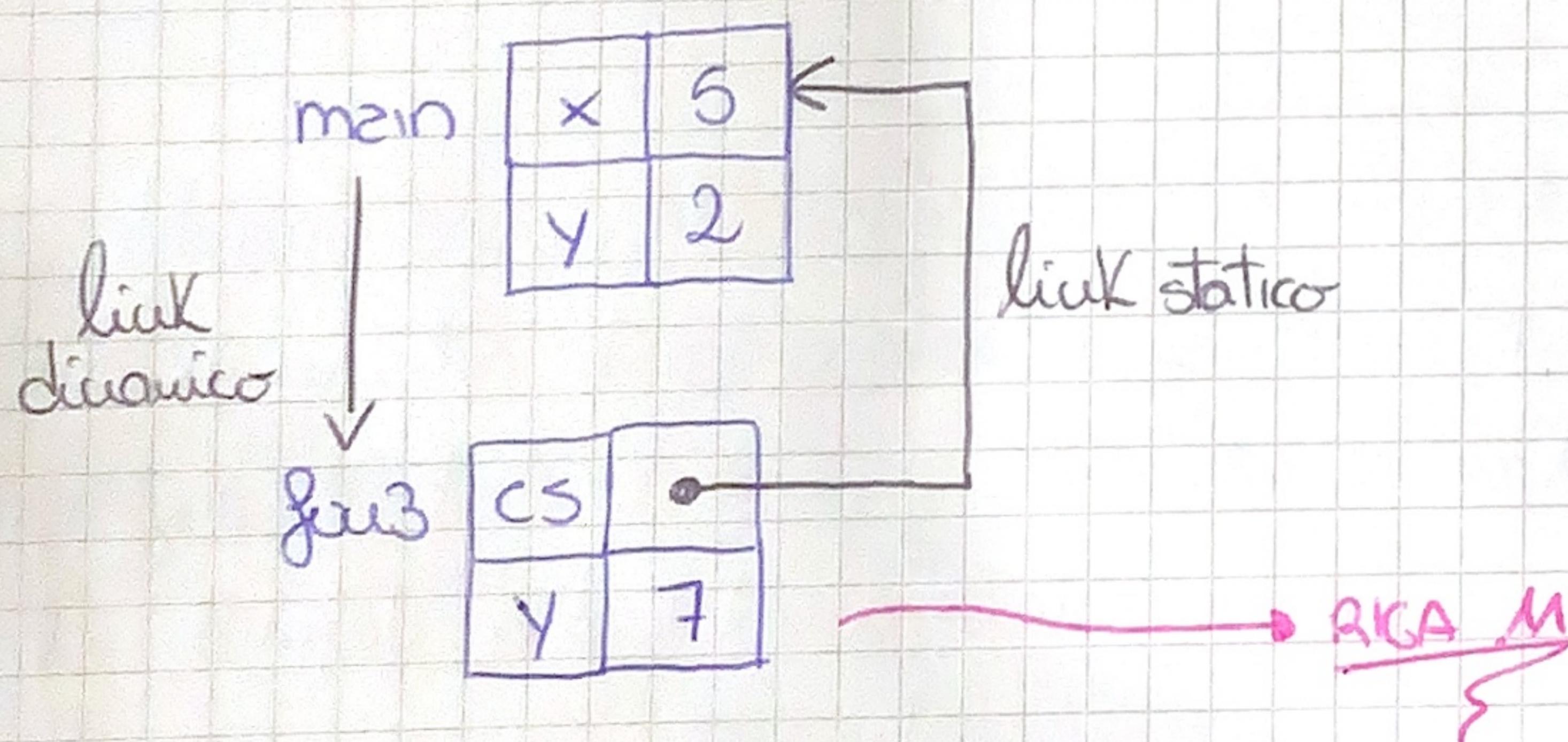
- 2) main chiama f(x3) :

$$Scd(\text{main}) - Scd(f(x3)) + 1 = 0 - 1 + 1 = 0$$

$CS(f(x3)) = \text{indirizzo}(\text{main})$

Variabili: y è locale a f(x3), x invece fa riferimento

$$\text{non locale} \rightarrow N = Scd(f(x3)) - Scd(\text{main}) = 1 - 0 = 1$$

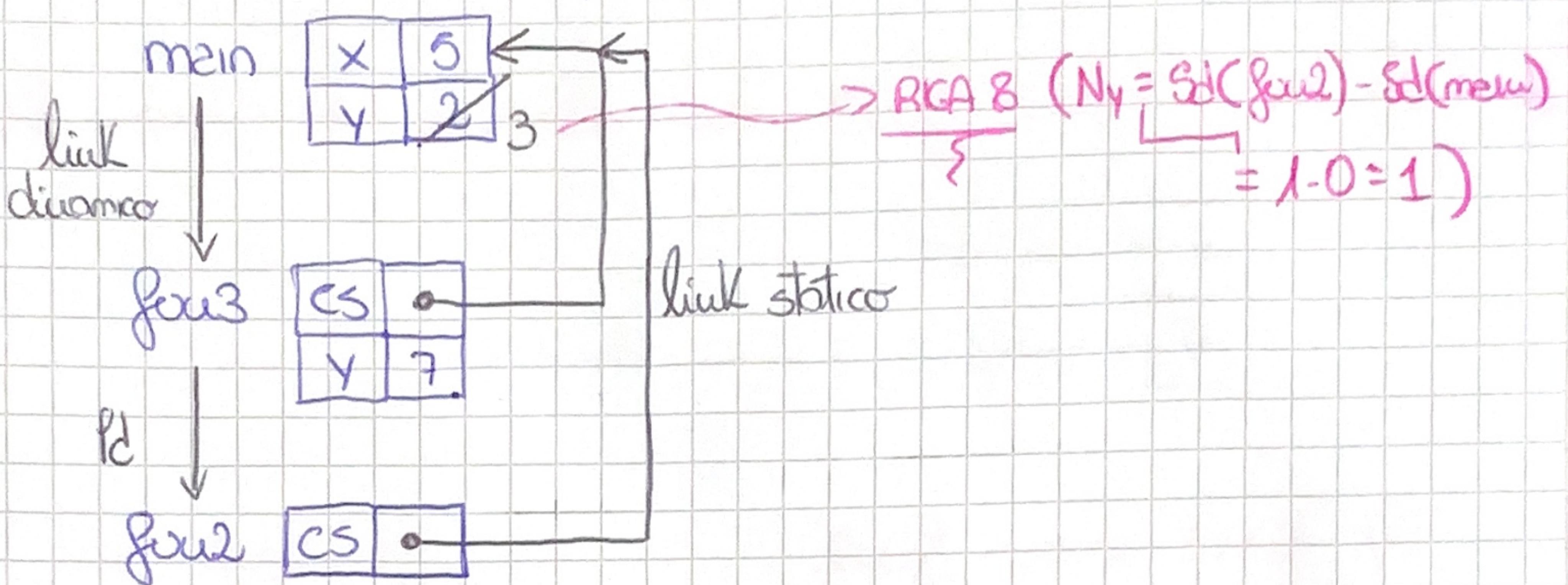


3) fun3 invoca fun2 (RIGA 12) e fun2 opera su y non locale:

$$Sc(fun3) - Sc(fun2) + 1 = 1 - 1 + 1 = 1$$

$$CS(fun2) = \text{indirizzo}(CS(fun3)) = \text{indirizzo main}$$

Variabili: x e y riferimenti non locali a main



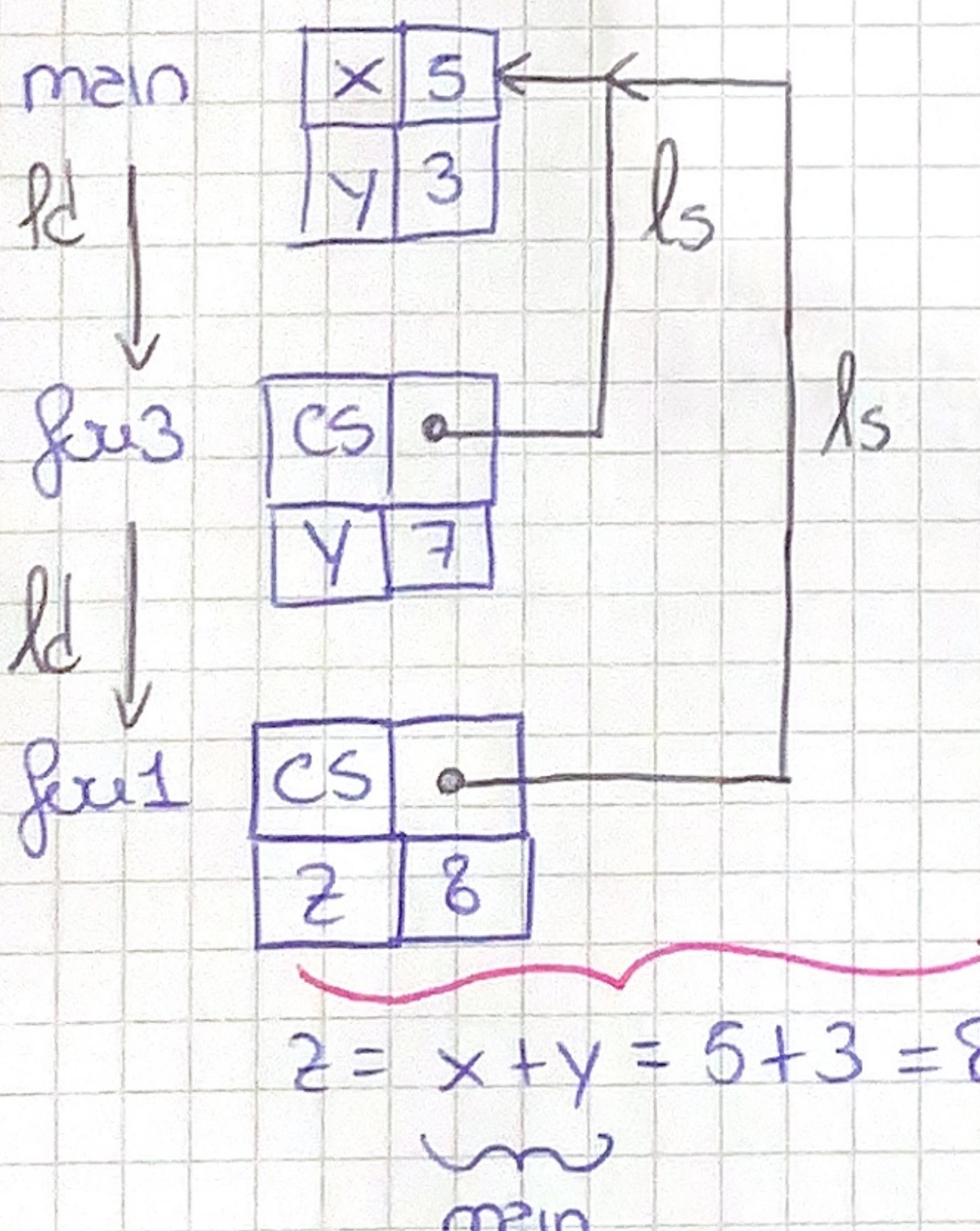
4) fun2 termina e viene distruita, fun3 chiama fun1 (RIGA 13),
fun1 crea z e calcola:

$$\text{Catenata: } Sc(fun3) - Sc(fun1) + 1 = 1 - 1 + 1 = 1$$

$$CS(fun1) = \text{indirizzo}(CS(fun3)) = \text{indirizzo main}$$

Variabili: x e y riferimenti non locali, quindi

$$N_{x,y} = Sc(fun1) - Sc(\text{main}) = 1 - 0 = 1$$



RIGA 5

5) Termina tutto!!

SCOPING DINAMICO :

1) main inizializza x e y :

x	•	→ 5, main
y	•	→ 2, main

2) main invoca fun3, crea nuova y + esecuzione fun3

x	•	→ main, 5
y	•	→ fun3, 7 → main, 2

3) fun3 invoca fun2

x	•	→ main, 5
y	•	→ fun3, 7 → main, 2

$y = 7 + 8$
|
4) fun2 exec ($y = y + 1$)

x	•	→ main, 5
y	•	→ fun3, 8 → main, 2

4) fun2 termina, fun3 chiama fun1

x	•	→ main, 5
y	•	→ fun3, 8 → main, 2
z	•	→ x

5) fun1 exec $z = x + y$

x	•	→ main, 5
y	•	→ fun3, 8 → main, 2
z	•	→ fun1, 13

6) Termina tutto!

ESECUZIONI SUL' INTRODUZIONE STRUTTURALE:

1) Dati u , che cerca i nodi di un albero, e h , che calcola l'altezza
di un albero, dimostrare che $\forall T \in Btree, n(T) \leq 2^{h(T)+1} - 1$.

$$n := \begin{cases} n(\text{foglio}) = 1 \\ n(\text{branch}(T_1, T_2)) = 1 + n(T_1) + n(T_2) \end{cases}$$

$$h := \begin{cases} h(\text{foglia}) = 0 \\ h(\text{branch}(T_1, T_2)) = 1 + \max(h(T_1), h(T_2)) \end{cases}$$

CASO BASE :

$$T = \log(1/\alpha) \quad n(T) = n(\log(1/\alpha)) = 1$$

$$h(\tau) = h(\text{foglio}) = 0.$$

$$\text{Calcoliamo } 2^{h(\tau)+1} - 1 = 2^0 + 1 - 1 = 2^1 - 1 = 2 - 1 = 1;$$

$$\text{Quando } \Rightarrow n(\tau) = 2^{R(\tau)+1} - 1 = 1 \quad \checkmark$$

Caso WDMW: $T = \text{brauch}(T_1, T_2)$

$$\text{hp: } n(\tau_1) \leq 2^{h(\tau_1)+1} - 1 \quad \text{demonstrating } n(\tau) \leq 2^{h(\tau)+1} - 1$$

Abbiamo che

$$\lambda) \quad n(\tau) = 1 + n(\tau_1) + n(\tau_2)$$

$$2) h(\tau) = 1 + \max(h(\tau_1), h(\tau_2)) \Rightarrow h \geq h(\tau_1) + 1 \\ \Rightarrow h \geq h(\tau_2) + 1$$

Quindi

$$n(\tau) = 1 + n(\tau_1) + n(\tau_2) \leq 1 + (2^{h(\tau_1)+1} - 1) + (2^{h(\tau_2)+1} - 1) \leq \dots$$

$$\dots \leq 2^h + 2^h - 1 \leq 2(2^h) - 1 \leq 2^{h+1} - 1 = 2^{h(r)} - 1 \quad \checkmark$$

Esercizio 2) Definiamo P come un 'conta foglie'. Dimostrare che $l(\tau) \leq 2^{h(\tau)}$
 $\forall \tau \in \text{Btree}$.

$$P := \begin{cases} P(\text{foglia}) = 1 \\ P(\text{branch } (\tau_1, \tau_2)) = P(\tau_1) + P(\tau_2) \end{cases}$$

BASE: con $\tau = \text{foglia}$

$$P(\tau) = 1, P(\tau) = 0 \text{ quindi } 1 \leq 2^0 = 1 \quad \checkmark$$

$$\text{HP WD: } l(\tau_1) \leq 2^{h(\tau_1)} \sim P(\tau_1) \leq 2^{h(\tau_1)}$$

PASSO WD: dimostriamo che $P(\tau) \leq 2^{h(\tau)}$

$$P(\tau) = P(\tau_1) + P(\tau_2) \sim h(\tau) = 1 + \max(h(\tau_1), h(\tau_2))$$

$$\Rightarrow P(\tau) = P(\tau_1) + P(\tau_2) \stackrel{\text{hp}}{\leq} 2^{h(\tau_1)} + 2^{h(\tau_2)} \stackrel{1}{\leq} 2^h + 2^h = 2(2^h) \\ \stackrel{1}{=} 2^{h+1} = 2^{h(\tau)} \quad \checkmark$$

Esercizio Scoping + completamento:

Es 1) I useranno codice funzione 'fun' t.c. ritorni lo stesso valore nelle due chiamate con scoping statico, ma valori diversi con scoping dinamico.

```
1 int x;
```

① IMPLEMENTA ②

```
void exec() {
```

```
    for (int j = 2; j <= 3; j++) {
```

```
        int y = 1;
```

```
        x = fun();
```

```
}
```

```
exec();
```

```
}
```

```
② { int j = 1;  
    { int fun() {  
        return j  
    } }
```

SCOPING STATICO:

La chiamata ad ogni interazione ritorna la j globale che abbiamo aggiunto, dato l'sd della funzione;

SCOPING DINAMICO:

La chiamata ritorna la j del ciclo for, che vale 2 alla 1^a iterazione, e 3 alla 2^a;

Esercizio 2) $\int_{\text{int } a=0}^5$

① IMPLEMENTA (1)

while ($a \leq 1$) {

 int x;

 ② IMPLEMENTA (2)

$x = \text{fun}();$

 } $a++;$

① : int $x = 5;$

 int fun() {

 return x;

}

...

② : int $x = a;$

...

...

SCOPING STATICO : x vale sempre 5!

SCOPING DINAMICO : x vale 0 alla 1^a iterazione, 1 alla 2^a;

riente
cedo =

2+5

ESERCIZIO SCOPING + BINDING:

- Definire brevemente scoping statico + dinamico, e dire quando servono le regole di binding e definire. Analizzare il codice in tutti i modi:

```

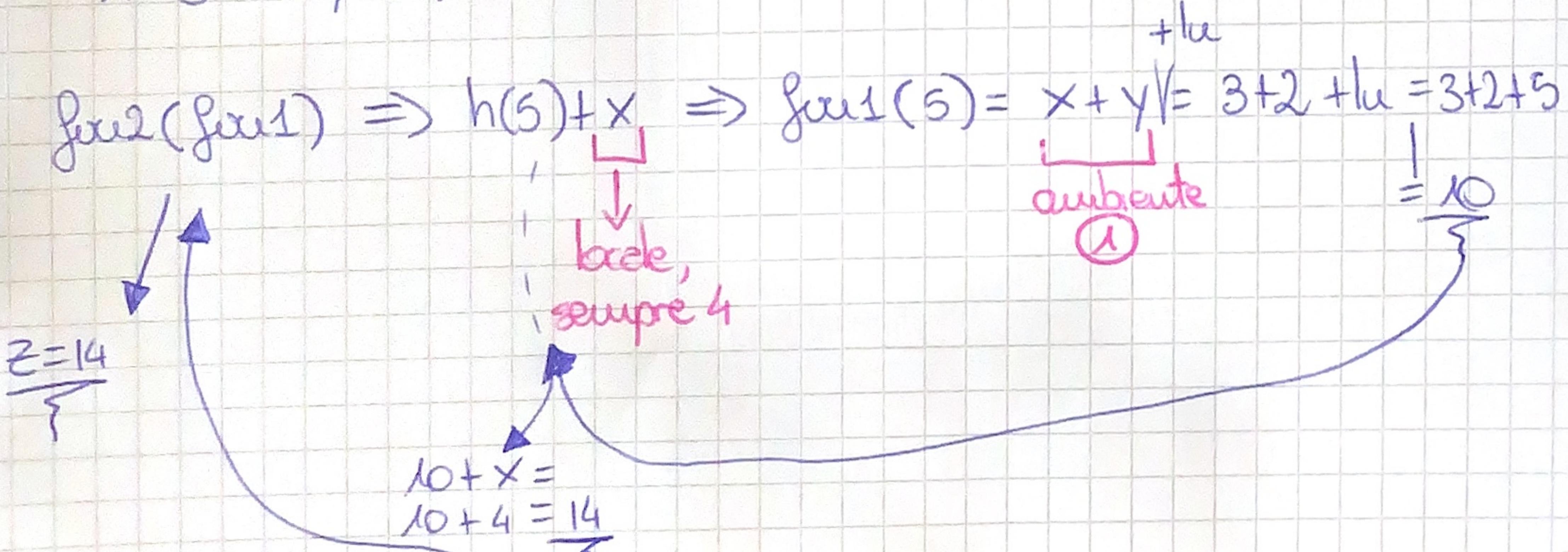
int x=3;
int y=2;
int fux1(int ln) {
    return x+y+ln;
}
int fux2(int h(int b)) {
    int x=4;
    int y=4;
    return h(5)+x;
}
{
    int x=6;
    int y=3;
    int z=fux2(fux1);
}
    
```

ambiente statico, del blocco di definizione; ①
ambiente non locale;
servono le regole di scoping;
DINAMICO + SHALLOW!
ambiente del blocco di chiamata effettiva mediante il parametro attuale; ③
DINAMICO + DEEP! ②
ambiente del blocco in cui viene creato il legame tra parametro attuale e formale;
servono le regole di binding;

Servono le regole di scoping quando una procedura ha un ambiente non locale. Servono anche le regole di binding quando una procedura con ambiente non locale è passata come parametro ad un'altra procedura. Definiti con il verde i vari ambienti presenti e usati.

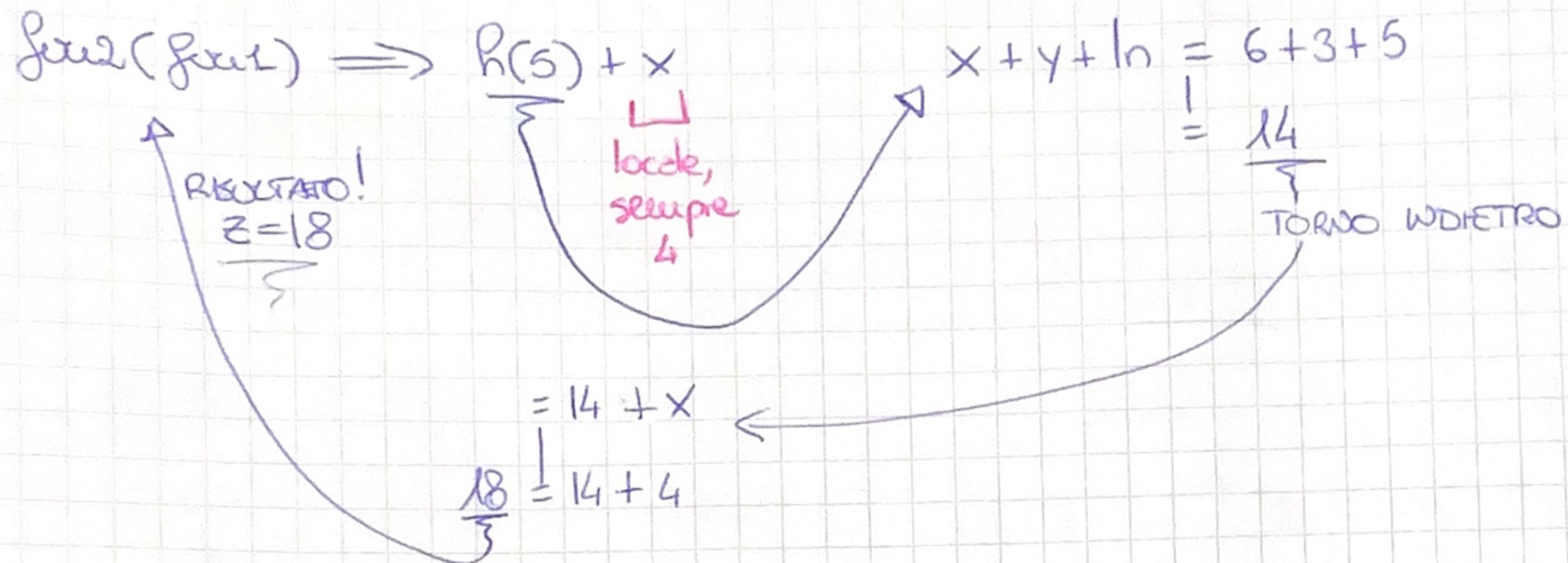
SCOPING STATICO: per fux1 l'ambiente di riferimento è ① dove

$$x=3 \wedge y=2.$$



SCOPING DINAMICO + DEEP:

per `fuz1` l'ambiente è ②, dae $x=6 \sim y=3$. Esecuzione:



SCOPING DINAMICO + SHALLOW:

con shallow, l'ambiente di `fuz1` è quello della chiamata effettiva della funzione, ossia $h(5)$. Quindi l'ambiente è ③ con $x=4 \sim y=4$.

