



# **Lezione 3**

Introduzione ai puntatori



# Uno sguardo alla RAM

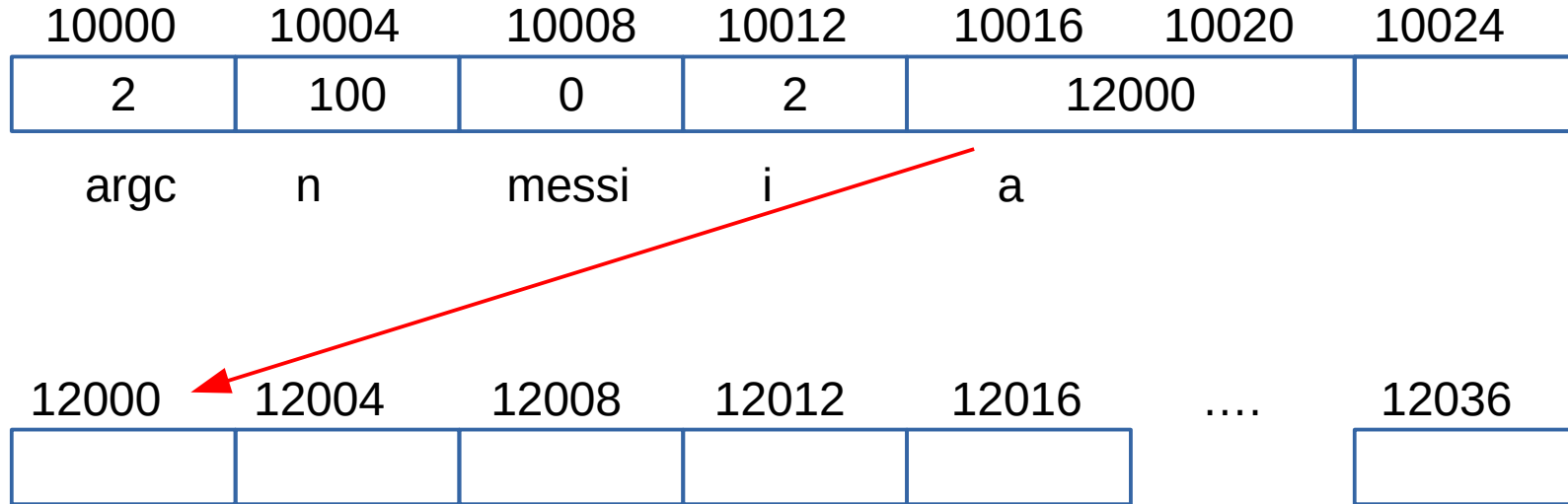
- La RAM del computer consiste in bytes, ad esempio 512MB = 536870912, singolarmente indirizzabili
- Ogni byte è individuato da un indirizzo, tra 0 e 536870911
- Il nostro programma occupa una porzione di questa RAM, diciamo a partire dalla posizione 10,000

Supponiamo il nostro programma contenga delle variabili intere (4 byte)

10000	10004	10008	10012	10016	10020	10024
2	100	0	2			
argc	n	messi	i			

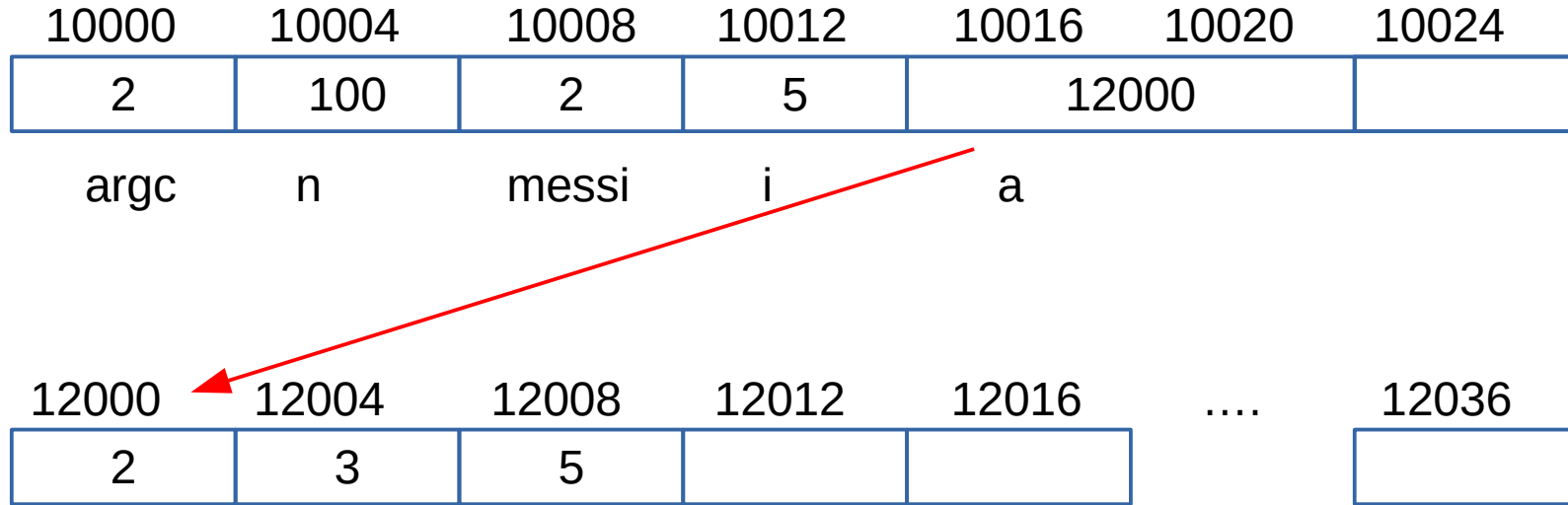
Situazione prima della chiamata a **malloc()**: solo variabili intere

Supponiamo il nostro programma contenga delle variabili intere (4 byte)



Dopo la **malloc**: 40 byte riservati dal sistema per il mio array

Supponiamo il nostro programma contenga delle variabili intere (4 byte)



Dopo aver scritto i primi tre elementi nell'array `a[ ]`

# realloc e free

10000	10004	10008	10012	10016	10020	10024
2	100	2	5	12000		

argc

n

messi

i

a

12000	12004	12008	12012	12016	....	12036
2	3	5				

esecuzione di **a = realloc(a,80)**

13000	13004	13008	13012	....	13076

# realloc e free

10000	10004	10008	10012	10016	10020	10024
2	100	2	5	13000		
argc	n	messi	i	a		

dopo esecuzione di **a = realloc(a,80)**

13000	13004	13008	13012	....	13076
2	3	5			

# realloc e free

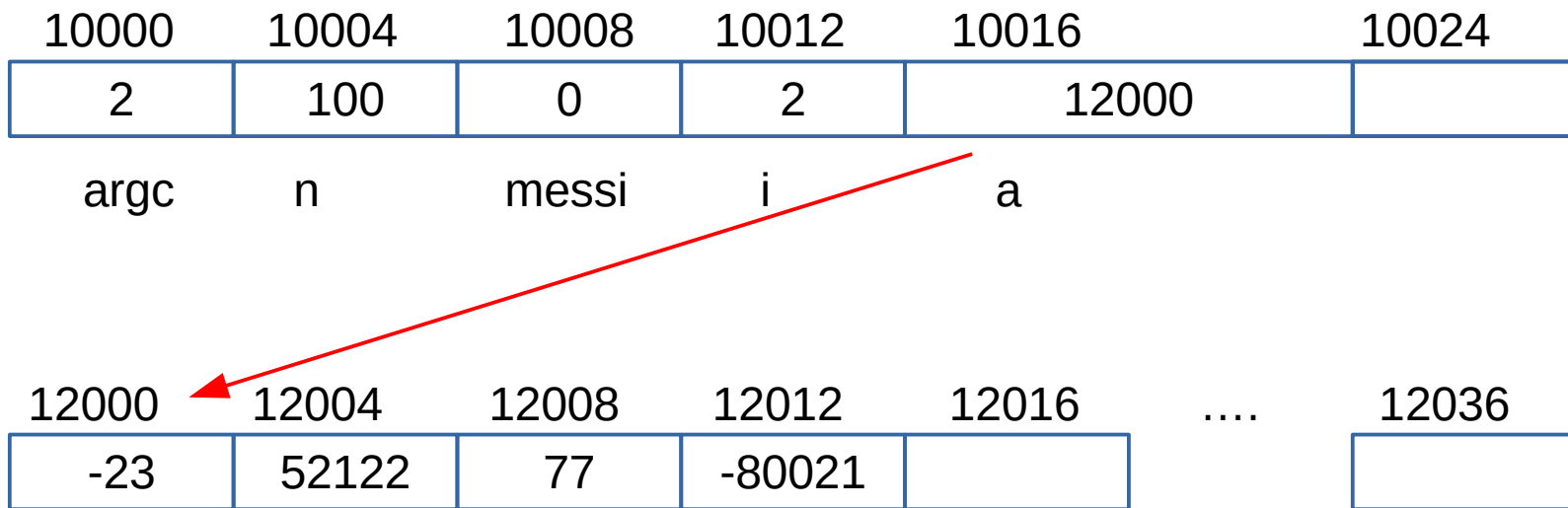
10000	10004	10008	10012	10016	10020	10024
2	100	2	5	13000		
argc	n	messi	i	a		

dopo esecuzione di **free(a)**

?



# La memoria restituita da malloc non è inizializzata!



Dopo la malloc: 40 byte riservati dal sistema per il mio array



# Da ricordare

- malloc serve per creare un array
- realloc per cambiarne le dimensioni
- free per distruggerlo
- un array è individuato dalla sua posizione in memoria del suo primo elemento
- tocca a noi non accedere **mai** a zone di memoria non assegnate a noi

# Puntatori

- una variabile che contiene l'indirizzo in memoria di un'altra variabile è detta ***puntatore***
- il tipo di un puntatore è individuato dal tipo della variabile puntata seguito ad un \*. Ad esempio: **`int *`**
- I puntatori contengono tutti la stessa cosa (un indirizzo) ma il compilatore distingue tra **`int *`**, **`long *`**, **`double *`**, etc.

# Estrarre l'indirizzo

- un altro modo per generare puntatori è l'operatore &
- data una variabile `w` se scrivo `&w` ottengo l'indirizzo di `w` nella RAM

10000	10004	10008	10012	10016	10020	10024
2	100	1	2	12000		
argc	n	messi	i	a		

Esempi: `&messi` → 10008, `&a` → 10016

# Cosa serve l'indirizzo?

- si usa solo con l'operatore `*` (dereferenziazione)
- esempio molto importante:

```
// metto in p l'indirizzo in RAM di messi  
int *p = &messi;  
// incremento n del valore della variabile a cui punta p (= messi)  
n += *p;  
// scrivo 7 nella variabile a cui punta p (= messi)  
*p = 7;
```

# Cosa serve l'indirizzo?

Gli operatori **&** e **\*** servono per “aggirare” il fatto che nel C quando chiamo una funzione i parametri sono passati per valore

```
main() {  
    ...  
    int n=7;  
    fun(n);  
    ...  
}
```

```
int fun(int a) {  
    ...  
    x = a*a;  
    a += x*y+a;  
    ...  
}
```

nulla di ciò che  
viene fatto in **fun**  
può alterare il  
valore di **n**

# Cosa serve l'indirizzo?

Gli operatori **&** e **\*** servono per “aggirare” il fatto che nel C quando chiamo una funzione i parametri sono passati per valore

```
main() {  
    ...  
    int n=7;  
    zun(&n);  
    ...  
}
```

```
int zun(int *p) {  
    ...  
    *p=5;  
    ...  
}
```

l'esecuzione di  
**\*p=5** scrive il  
valore 5 dentro **n**