Informatica - Mod. Programmazione
Lezione 05

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica
Università di Bergamo

Vettori

- Come fare a memorizzare **Insiemi di Valori**?
- Esempio: Scrivere un programma che legge 5 numeri e, terminato l'inserimento, li stampa nell'ordine di inserimento

Vettori

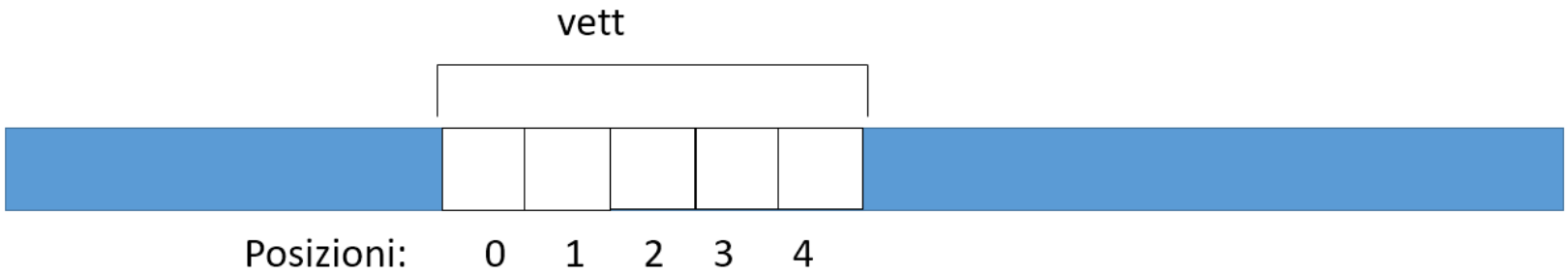
- Esempio: Scrivere un programma che legge 5 numeri e, terminato l'inserimento, li stampa nell'ordine di inserimento
- **Non è possibile risolverlo con l'approccio ad accumulo**.
- Possibile soluzione: definiamo 5 variabili,
  scriviamo 5 istruzioni di lettura
  scriviamo 5 istruzioni di stampa
- Funziona, ma se i valori da leggere sono $1'000$?
  Serve qualche cosa d'altro

Vettori/Array

```
int vett[5];
```

- Con questa definizione, diciamo che la variabile `vett` è un **vettore** che contiene **5 elementi**, tutti di tipo intero.
- Il termine **Array** è sinonimo di **Vettore**

Nella Memoria Centrale

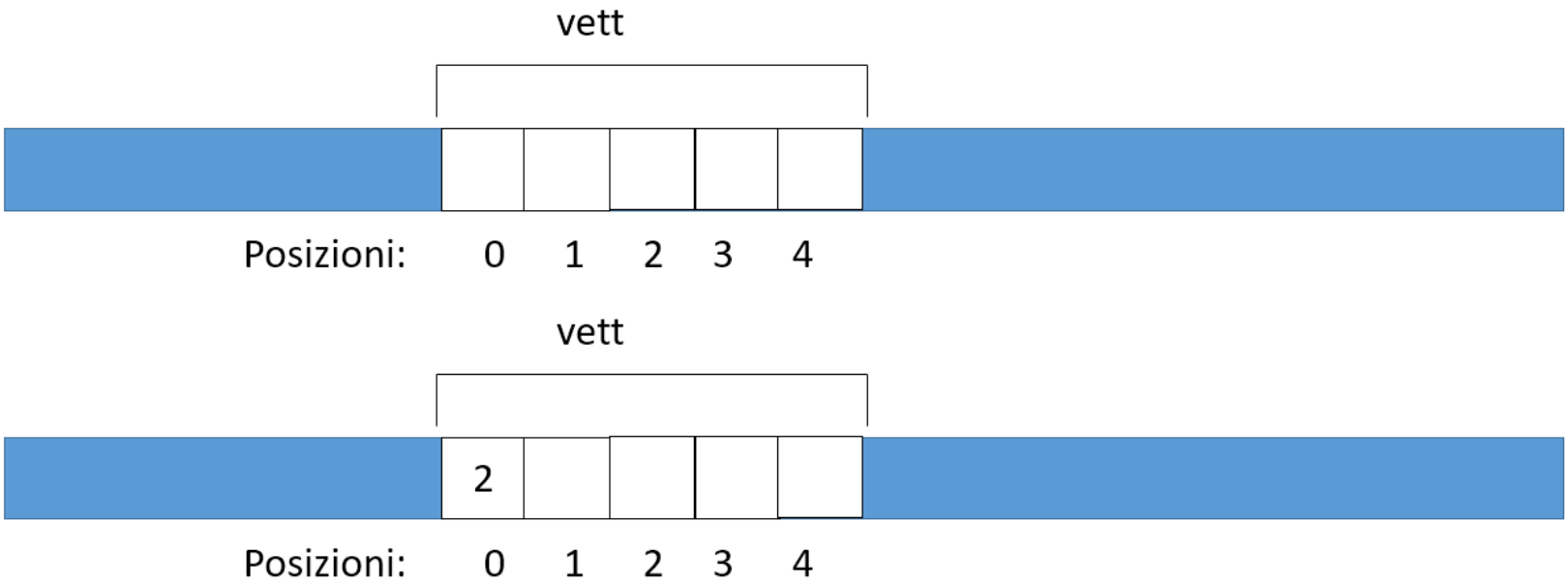


Vettori/Array

```
vett[0] = 2;
cout << vett[0];
```

- Per gestire il singolo elemento del vettore,
- il nome della variabile è seguito da una coppia di parentesi quadre, con all'interno l'indice di posizione dell'elemento
- La prima istruzione assegna il valore 2 all'elemento in posizione 0
  la seconda istruzione stampa il valore dell'elemento 0

Nella Memoria Centrale



Vettori/Array

```
int i=0;
vett[i] = 2;
cout << vett[i];
```

- Nelle parentesi quadre possiamo mettere un'espressione
- Nella versione più semplice, l'espressione è composta da una sola variabile
- Conseguenza: possiamo gestire TUTTI gli elementi del vettore con un ciclo

Programma: Vettori 01.cpp

```
int main()
{
    int vett[5];
    int i;

    for(i=0; i < 5; i++)
    {
        cout << "Inserire un valore: ";
        cin >> vett[i];
    }

    cout << endl;
    for(i=0; i < 5; i++)
        cout << "Posizione " << (i+1)
             << " valore: " << vett[i] << endl;

    return 0;
}
```

Vettori/Array

- Attenzione a come vengono scritti i cicli per gestire gli array:
  `for(i=0; i < 5; i++)`
- infatti, occorre generare esattamente le posizioni degli elementi nel vettore
  con N elementi
  da 0 a $N - 1$
- È diventata una prassi:
  a parte casi particolari, i cicli a contatore partono **sempre** da 0.

Vettori/Array

- Se invece di avere 5 elementi ne abbiamo 10?
- Facile, basta cambiare tutti i 5 in 10
- Uhm ...

Programma: Vettori 02.cpp

```
int main()
{
    int vett[10];
    int i;

    for(i=0; i < 10; i++)
    {
        cout << "Inserire un valore: ";
        cin >> vett[i];
    }

    cout << endl;
    for(i=0; i < 10; i++)
        cout << "Posizione " << (i+1)
             << " valore: " << vett[i] << endl;

    return 0;
}
```

Vettori/Array

- Uhm ...
- Troppo fragile: se non si cambiano tutti i 5 in 10, il programma non funziona correttamente
  CONSEGUENZA: tempo perso inutilmente per sistemarlo
- Soluzione: introdurre le **Costanti Simboliche**

Costanti Simboliche (Macro)

```
#define SIZE 10
```

- Al simbolo `SIZE` viene associata la sequenza di caratteri `10` (non il valore numerico)
- Tutte le volte che il pre-compilatore incontra `SIZE` lo sostituisce con `10` (macro-sostituzione)
- In questo modo, si rende il programma parametrico rispetto all'effettivo numero di elementi da gestire
- È prassi mettere le istruzioni `#define` all'inizio del file

Programma: Vettori 03.cpp

```
#define SIZE 10

int main()
{
    int vett[SIZE];
    int i;

    for(i=0; i < SIZE; i++)
    {
        cout << "Inserire un valore: ";
        cin >> vett[i];
    }

    cout << endl;
    for(i=0; i < SIZE; i++)
       cout << "Posizione " << (i+1)
            << " valore: " << vett[i] << endl;

    return 0;
}
```

Costanti Tipizzate

```
const int SIZE = 10;
```

- `SIZE` viene definita **come se fosse** una variabile che non cambia il suo valore iniziale
  `const` dice proprio questo
- Quindi, a differenza di prima, ora `SIZE` ha un tipo e `10` è il valore assegnatole, di tipo intero

Programma: Vettori 04.cpp

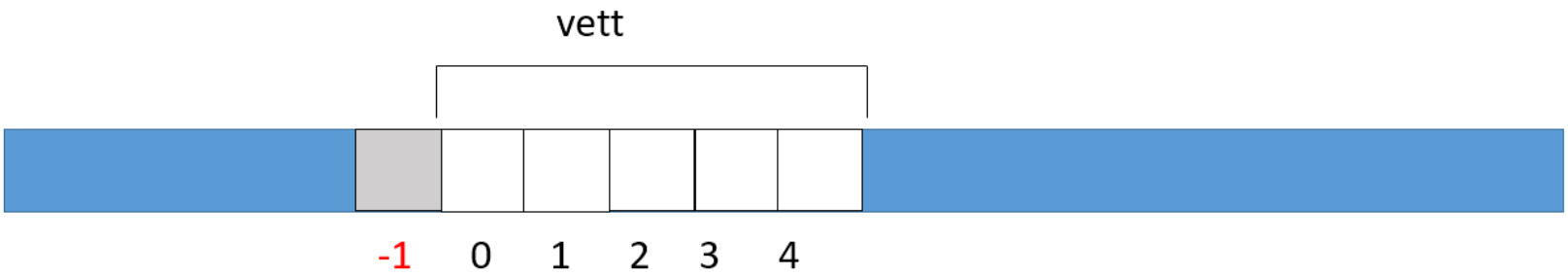
```
int main()
{
    const int SIZE = 10;
    int vett[SIZE];
    int i;

    for(i=0; i < SIZE; i++)
    {
        cout << "Inserire un valore: ";
        cin >> vett[i];
    }

    cout << endl;
    for(i=0; i < SIZE; i++)
       cout << "Posizione " << (i+1)
            << " valore: " << vett[i] << endl;

    return 0;
}
```

Trappole dei Vettori

```
vett[-1] = 10;
```

- Ovviamente, non esiste un elemento con posizione negativa
- Ma per ragioni che capiremo fra non molto, né durante la compilazione né durante l'esecuzione, viene segnalato o generato un errore.

Nella Memoria Centrale



Trappole dei Vettori

- Per ragioni di efficienza,
- non viene fatto alcun controllo sul fatto di andare fuori dallo spazio riservato al vettore
- In caso di assegnamento, **sporchiamo** la memoria
- Effetti possibili:
  - lo spazio modificato non era usato, quindi non facciamo danni
  - lo spazio modificato era usato da altre variabili, quindi facciamo **danni subdoli**
  - lo spazio modificato è fuori dallo spazio riservato al programma, il sistema operativo interviene (crash del programma)

Sfida

Programma: Sopra_Media.cpp

- Scrivere un programma che legge da tastiera una serie di 10 numeri interi.
- Terminata la lettura, il programma stampa i i valori superiori alla media dei valori nella serie, in ordine inverso di inserimento

Programma: No_Duplicati.cpp

- Scrivere un programma che legge da tastiera una serie di 10 numeri interi.
- Ogni volta che un valore viene letto, occorre verificare che non sia stato letto in precedenza:
  se ciò accade, si chiede all'utente di re-inserire il valore.
- Terminata la lettura, il programma stampa la sequenza in ordine di inserimento.

Indirizzi

- Come è organizzata la memoria centrale?
- È un gigantesco vettore di Byte
  ogni Byte (o cella) ha una ben precisa posizione
  la prima cella ha posizione 0
- La posizione di una cella di memoria prende il nome di **INDIRIZZO**

Indirizzi e Variabili

- Ogni variabile **Occupa** un certo spazio di memoria
  Le celle contigue necessarie per rappresentare il valore
- Quindi ha una certa posizione di memoria: quella del primo Byte
- L'indirizzo di una variabile è l'indirizzo del primo Byte (cella) usato dalla variabile

Indirizzi e Variabili

- Come ottenere l'indirizzo di una variabile a in C/C++?

  ```
  &a
  ```

- L'operatore & usato davanti ad una variabile fornisce il suo indirizzo

Indirizzi e Variabili

- Che fare con l'indirizzo di una variabile?
- Stamparlo non ha alcuna utilità
- Occorre memorizzarlo
  in un'altra variabile

Indirizzi e Variabili

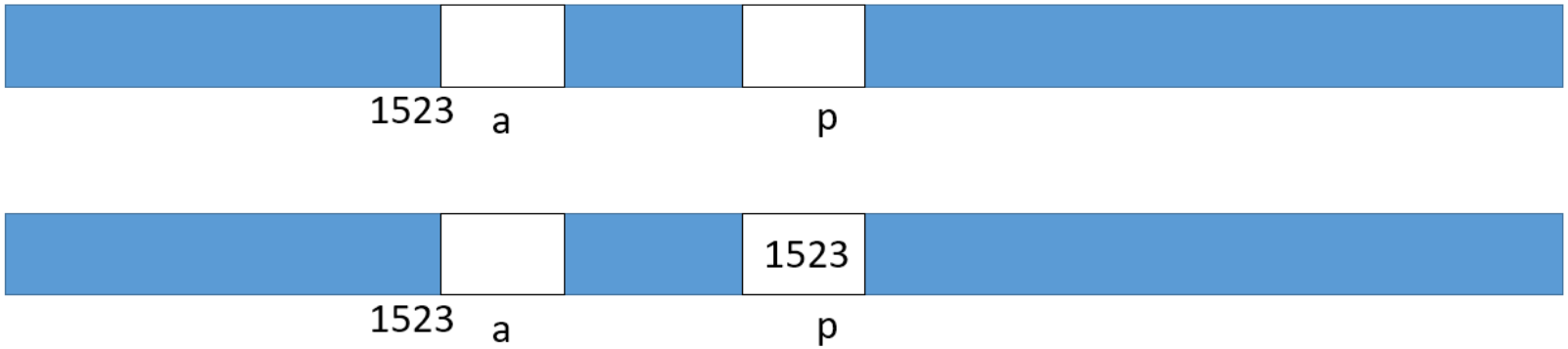
```
int *p;
p = &a;
```

- La variabile `p` memorizza l'**indirizzo di una variabile che contiene un numero intero**.
- Il tipo di `p` è

  `int *`

  che indica che il valore è l'indirizzo di una variabile che memorizza un numero intero
- Dopo l'assegnamento, `p` contiene l'indirizzo di `a`

Nella Memoria Centrale



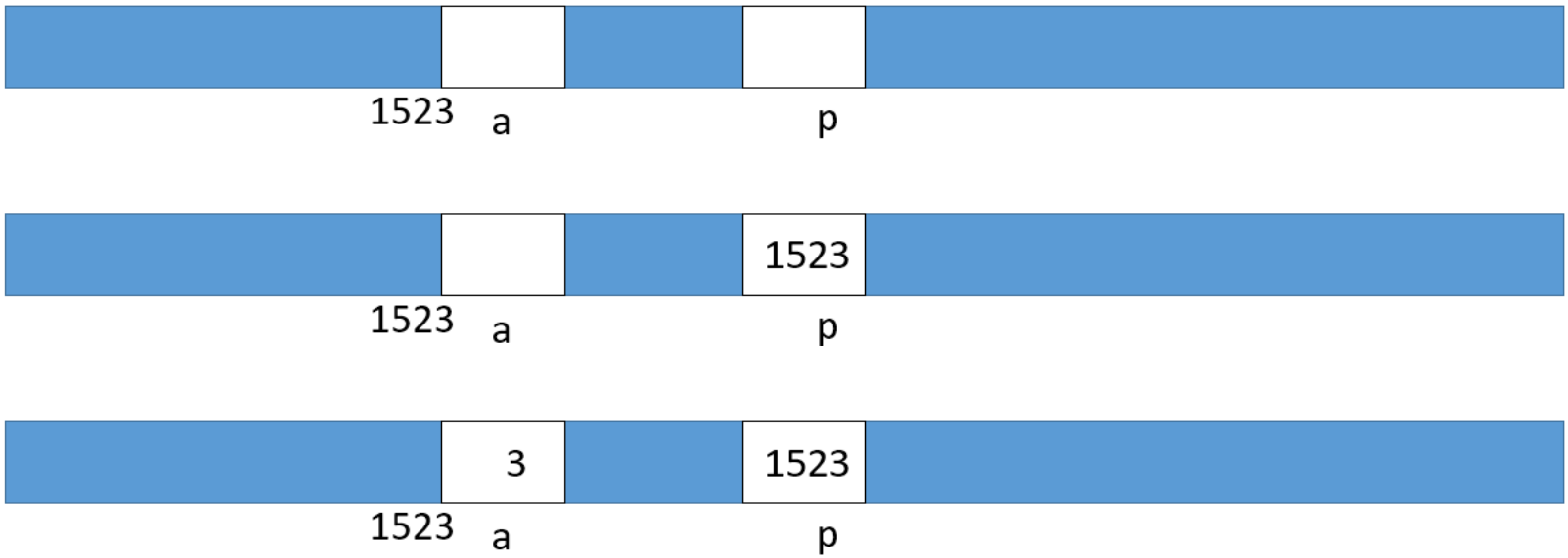
Uso dell'Indirizzo

- Una volta che `p` è stata inizializzata con l'indirizzo di `a`
- possiamo accedere allo spazio di memoria di `a` partendo da `p`

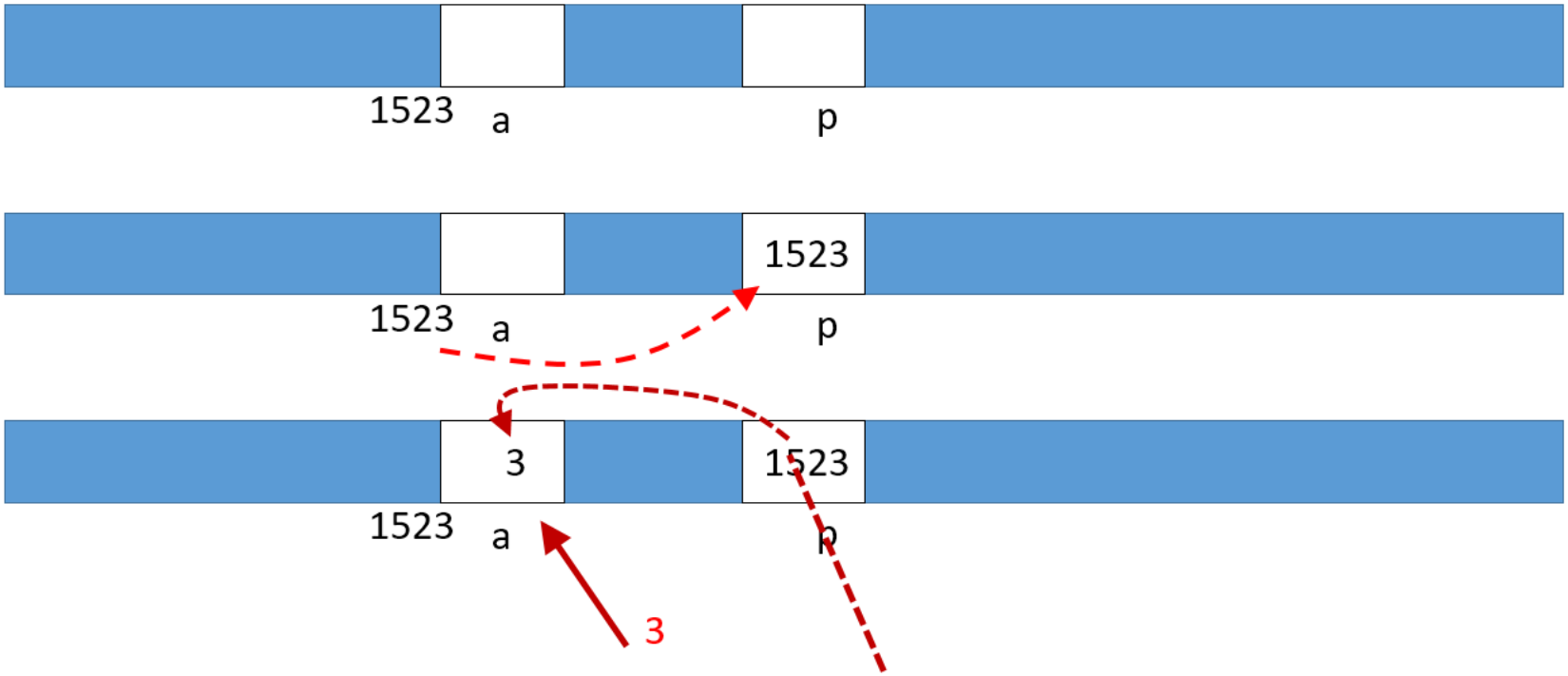
```
*p = 3;
```

- Diciamo che l'indirizzo viene **de-referenziato**

Nella Memoria Centrale



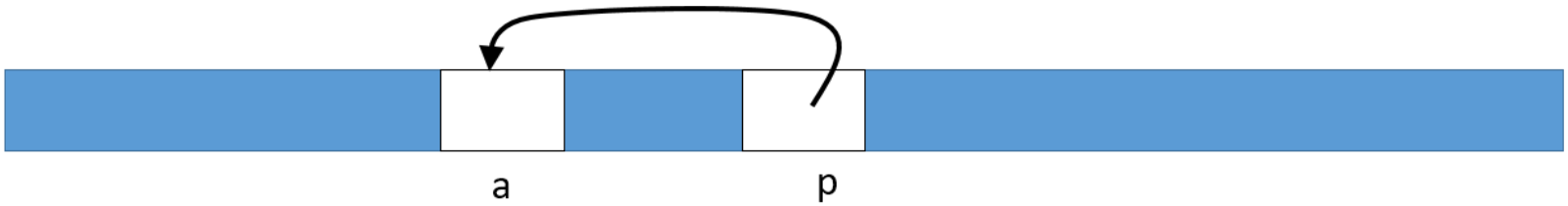
Nella Memoria Centrale



Ma Perché PUNTATORE?

- Per vedere che cosa succede, non usiamo i numeri
- usiamo le frecce: dal centro della variabile `p` al bordo della variabile `a`
- Diciamo che `p` **PUNTA ad** `a`
- Quindi, `p` è un **PUNTATORE**

Nella Memoria Centrale



Programma: Puntatori_01.cpp

```
int main()
{
    int a=1;
    int *p;

    cout << "Valore di a: " << a << endl;
    p = &a;
    *p = 3;

    cout << "Valore di a: " << a << endl;

    return 0;
}
```

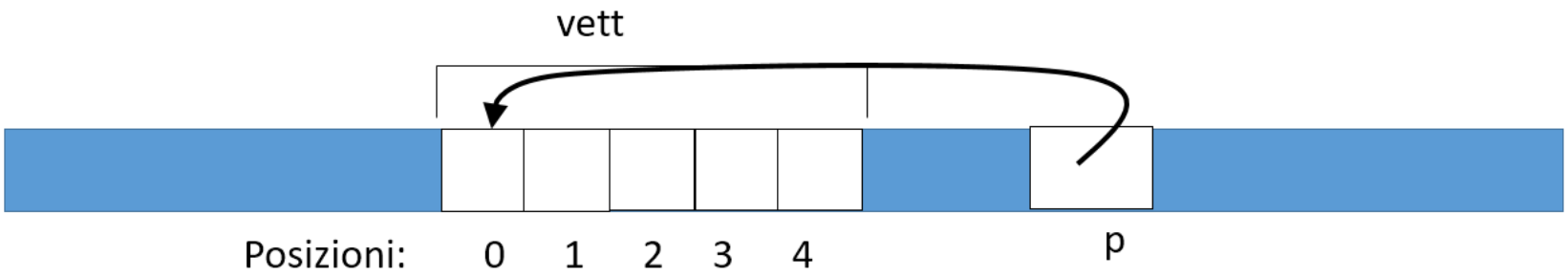
```
Valore di a: 1
Valore di a: 3
```

Puntatori e Vettori

- Consideriamo un vettore e l'indirizzo del primo elemento

```
int vett[5];
int *p;
p = &vett[0];
```

Nella Memoria Centrale



Puntatori e Vettori

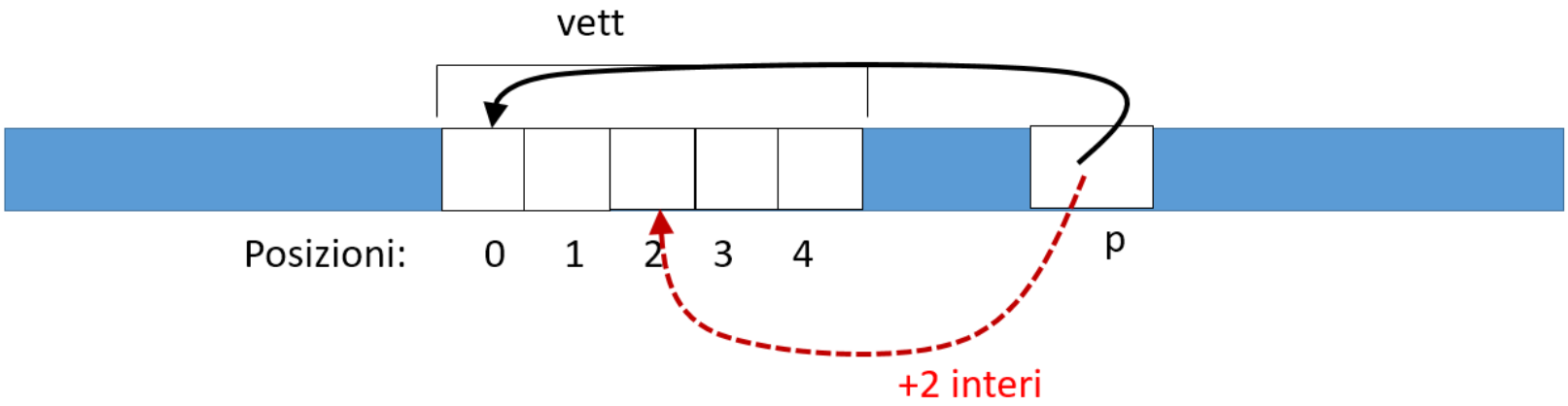
- E l'indirizzo degli altri elementi?
  Per esempio, l'elemento 2

  `&vett[2]`
- Qual è la relazione con il valore di `p`?
  Vi sono esattamente due elementi interi
- **Aritmetica dei Puntatori**

  `p + 2`

  Parte dall'indirizzo memorizzato in `p` e lo **spiazza** (incrementa) di due elementi interi
  Si ottiene lo stesso valore di `&vett[2]`

Nella Memoria Centrale



Puntatori e Vettori

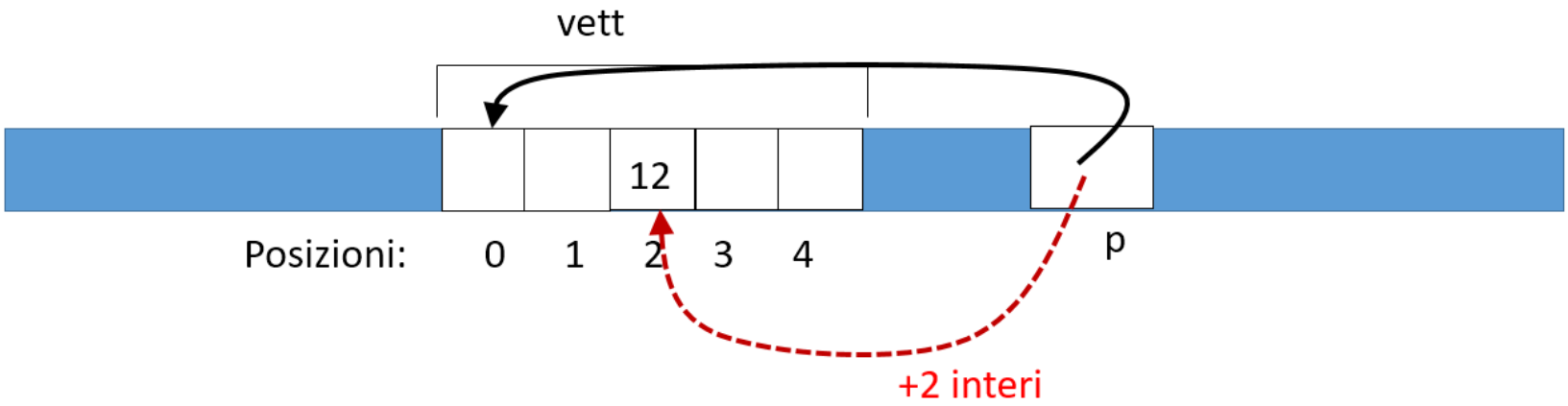
- Per accedere al valore dell'elemento 2?
- Si deve **de-referenziare** l'indirizzo così ottenuto

```
*(p + 2) = 12;
```

- Ma questo è equivalente a scrivere

```
vett[2] = 12;
```

Nella Memoria Centrale



Puntatori e Vettori

- Come vengono gestiti realmente i vettori?
- La variabile del vettore, senza parentesi, indica l'**indirizzo del primo elemento del vettore**
- Di conseguenza, si può scrivere

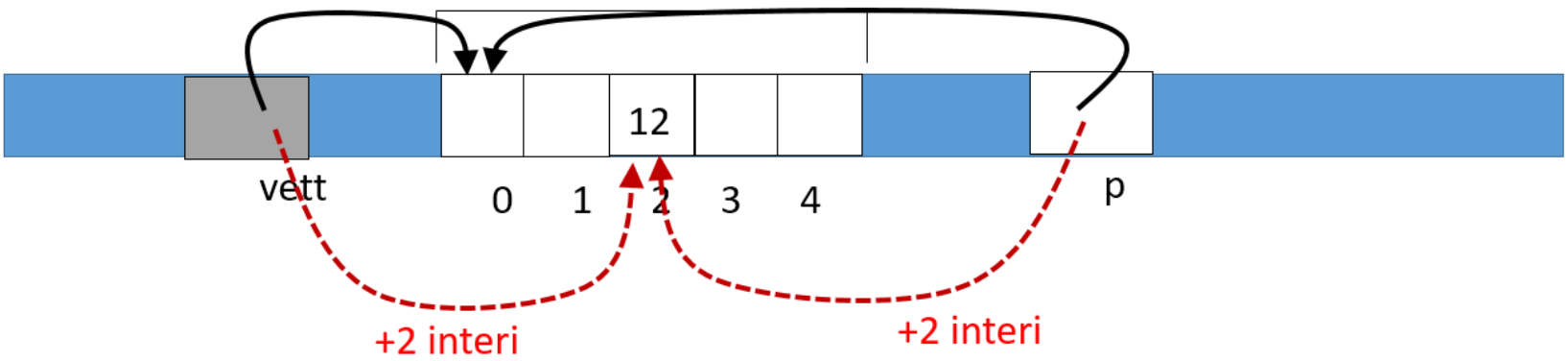
```
p = vett;
```

  invece di

```
p = &vett[0];
```

- **È come se**, oltre agli elementi del vettore, ci fosse anche lo spazio per l'indirizzo

Nella Memoria Centrale



Puntatori e Vettori

- Quando usiamo il vettore
  in realtà stiamo usando il puntatore
- Il compilatore traduce

```
vett[2] = 12;
```

  e

```
*(vett + 2) = 12;
```

  allo stesso modo
- Sembra che le due notazioni (puntatori e vettori) siano equivalanti
- è vero?

Puntatori e Vettori

- **SÌ**, è vero, infatti

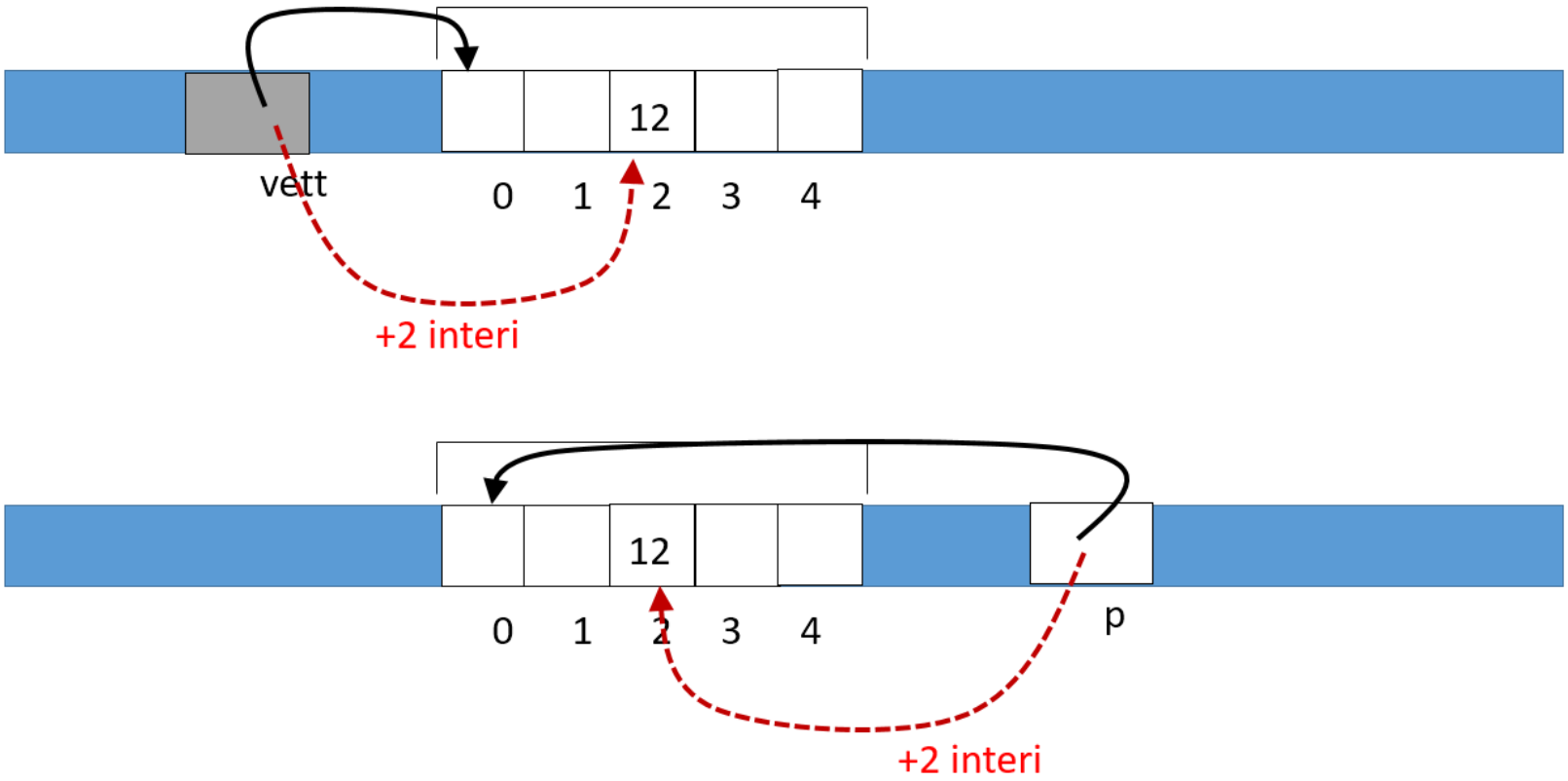
```
*(p + 2) = 12;
```

  e

```
p[2] = 12;
```

  sono equivalenti
- Ma come? il vettore `p` non esiste
- Non ha importanza, la notazione vettoriale è una notazione comoda per gestire i puntatori

Nella Memoria Centrale



Puntatori e Vettori

- Si chiama
  equivalenza puntatori/vettori
- ATTENZIONE: vale per i valori, non per le variabili definite come vettori

Puntatori e Vettori

- `int vett[5];`
- La variabile `vett` in realtà non esiste
- Il suo valore viene calcolato al volo
- `vett = ...`
  dà ERRORE di compilazione
  perché `vett` **NON è un l-value**
- l-value: qualcosa che può stare a SINISTRA di un assegnamento (l sta per left)

Programma: Puntatori_02.cpp

```
int main()
{
    const int SIZE=5;
    int vett[SIZE];
    int i;
    int *p;

    p = vett;

    for(i=0; i < SIZE; i++)
    {
        cout << "Inserire un valore: ";
        cin >> *(p + i);
    }
```

Programma: Puntatori_02.cpp

```
    cout << endl;
    for(i=0; i < SIZE; i++)
        cout << "Posizione " << (i+1)
             << " valore: " << p[i] << endl;

    return 0;
}
```