Puntatori: definizione

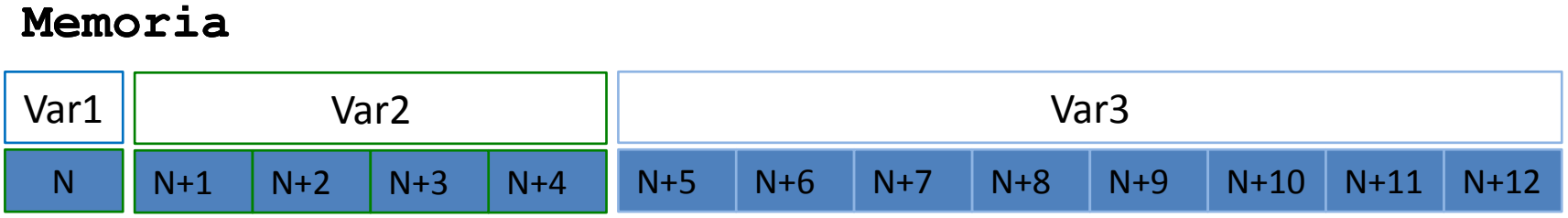
- La memoria (insieme di bytes) di un computer e' un insieme di celle da un byte numerate (indirizzate) in modo consecutivo
- Se le variabili `var1` e `var2` sono adiacenti, i loro indirizzi di memoria sono rispettivamente `N` e `N+dim`, dove `dim` è la dimensione di `var1`
- Esempi:

  `var1` di tipo `char`, `dim=1 byte (su AMD Opteron)`

  `var2` di tipo `int`, `dim=4 bytes (su AMD Opteron)`

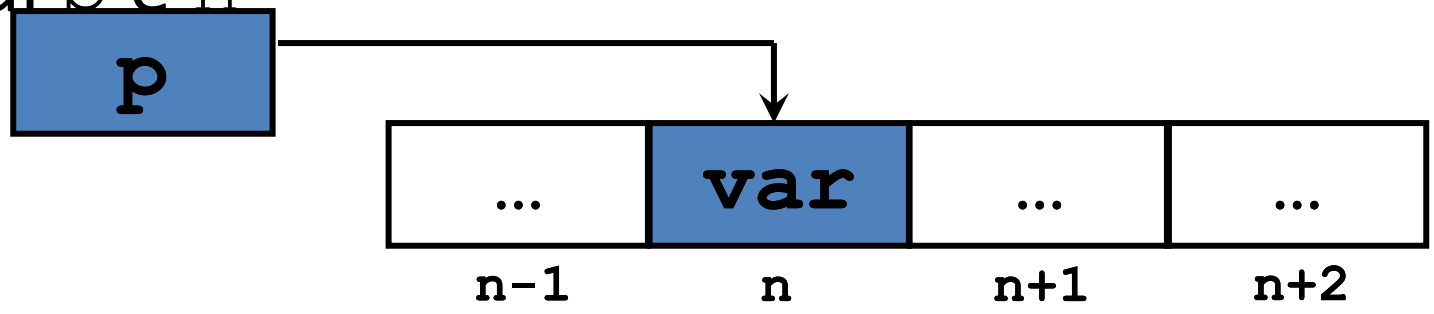
  `var3` di tipo `double`, `dim=8 bytes (su AMD Opteron)`

- Graficamente:



Puntatori: definizione

- Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile
- Se la variabile `var` si trova all'indirizzo di memoria `n` e `p` è un puntatore a `var`, il valore di `p` è `n`



Puntatori: dichiarazione

- Ogni puntatore deve "puntare" ad uno specifico tipo di dati:

```
char *p1;          puntatore a char
int *p2;           puntatore a int
double *p3;        puntatore a double
In questo modo determiniamo come
incrementare (decrementare) il
puntatore di 1, 2, .. unita'
```

Puntatori: l'operatore &

- L'operatore unario & fornisce l'indirizzo di una variabile:

```
int i;
int *pi;
pi = &i;
```

  Ora `pi` punta ad `i`: il valore di p e' l'indirizzo della prima cella di memoria occupata dalla variabile `i`.

- L'operatore & restituisce un intero: la memoria e' un insieme numerabile di bytes

Puntatori: l'operatore *

- L'operatore unario * fornisce il valore della variabile puntata:

```
int i = 10;
int *pi;

pi = &i;
cout<<(*pi)<<endl;
```

Stampa 10

- L'operatore unario * si usa per modificare il valore della variabile puntata:

```
*pi = 45;
```

Ora `i` vale `45`

```
cout<<(*pi)<<endl;
```

Stampa 45

```
cout<<i<<endl;
```

Stampa sempre 45

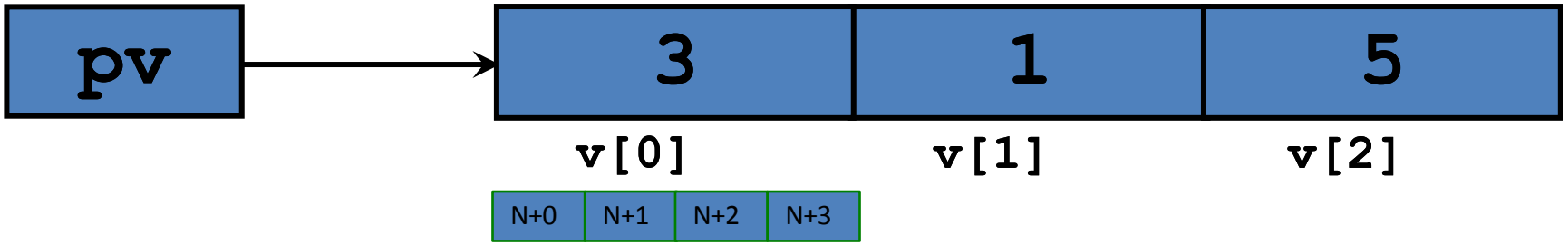
Aritmetica dei puntatori

- Non tutte le operazioni aritmetiche hanno senso tra puntatori perché essi rappresentano indirizzi di memoria (variabili intere)
- Sono ammesse:
  - Addizione tra un puntatore ed una espressione a risultato intero
    -> Fanno avanzare il puntatore
  - Sottrazione tra un puntatore ed una espressione a risultato intero -> Fanno retrocedere il puntatore
- Sono ammesse la differenza e il confronto tra 2 puntatori
  - Solo se puntano ad elementi di uno stesso array
  - Negli altri casi dipende dall'implementazione
- Il compilatore esegue il calcolo dell'indirizzo in base al tipo dell'oggetto puntato
- Prodotto/rapporto/somma di puntatori?!?!

Puntatori e vettori: esempio

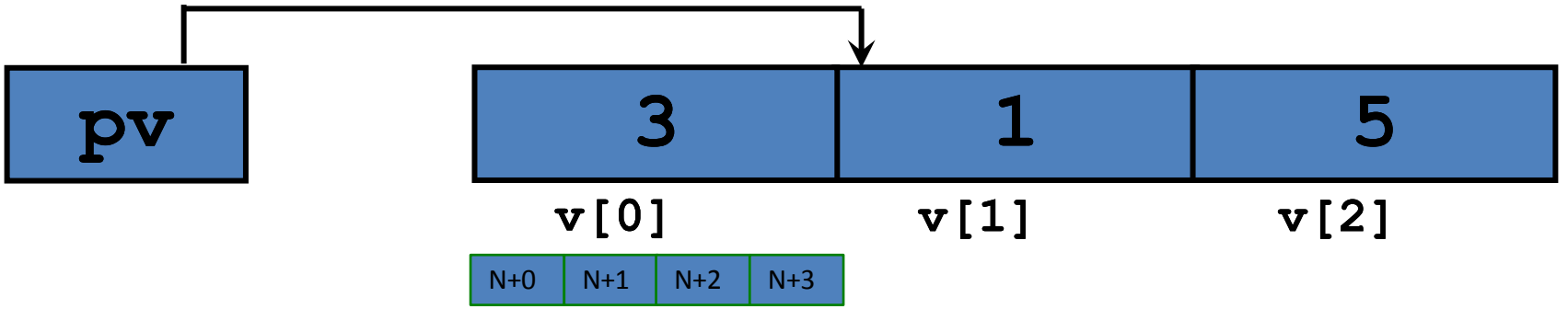
```
int v[3] = {3, 1, 5}, *pv;
pv = &v[0];
```

`pv` punta a `v[0]`



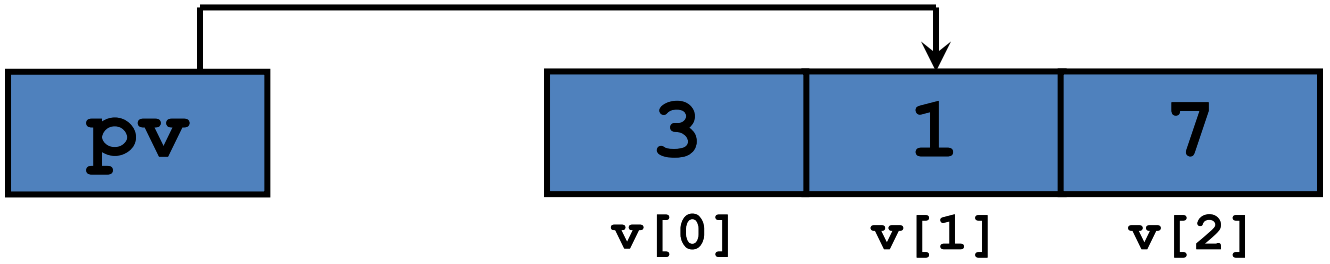

```
pv++;
```

ora `pv` punta a `v[1]`

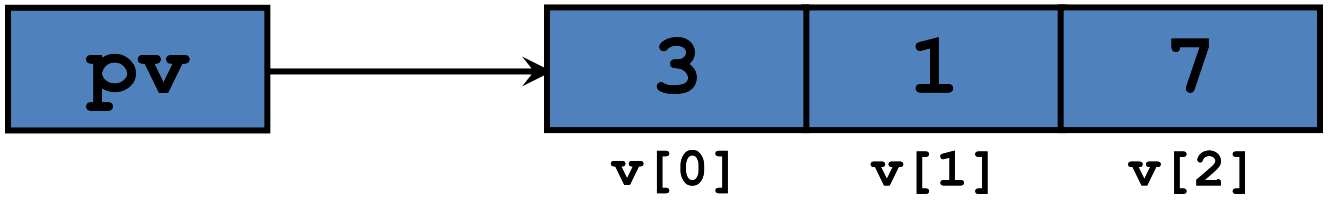


Puntatori e vettori

`*(pv+1) = 7;` `pv punta a v[1], (pv+1) a v[2]`



`--pv;` ora `pv` punta a `v[0]`



Puntatori e vettori

- Il nome di un vettore è una costante con valore pari all'indirizzo del suo primo elemento:

```
int v[3], *pv;
pv = v;        uguale a    pv = &v[0];
```

- Il puntatore però è una variabile, il nome di un vettore NO!

```
pv = pv + 2;          OK!
v = v + 2;            ERRORE!
```