Lecture
11_7.1

Pointers

Alessandro SAVINO
Politecnico di Torino (Italy)

alessandro.savino@polito.it

www.testgroup.polito.it

License Information

Disclaimer

- **We disclaim any warranties or representations as to the accuracy or completeness of this material.**
- **Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.**
- **Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.**

Goal

- **This lecture presents a global overviews of problems and issues related to dynamic memory allocation and pointers usage**

Prerequisites

- **Basic knowledge of C programming language**

Homework

– **None**

Outline

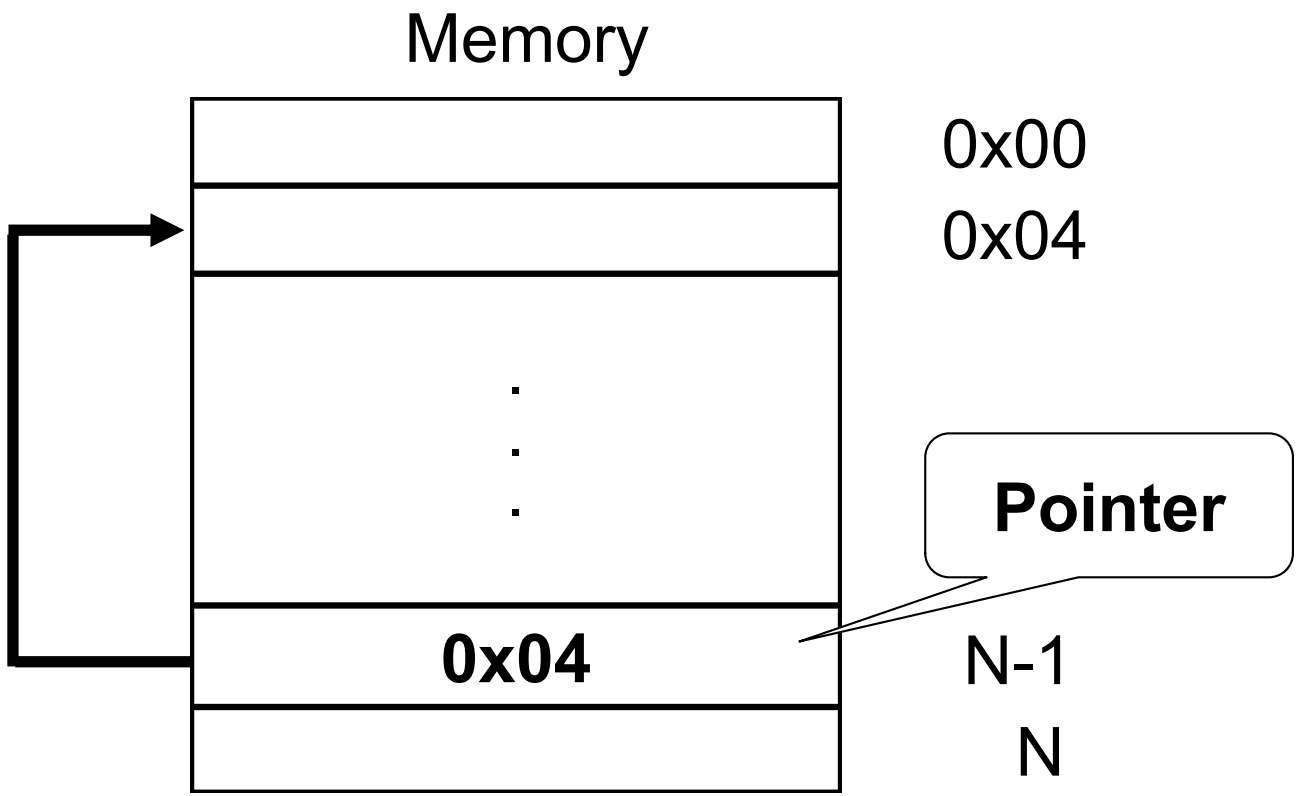
- **Pointers:**
  - **Definition**
  - **Initialization**
  - **Operators**
  - **Variable Reference**
  - **Pointers Arithmetic's**

Outline

- **Pointers:**
  - **Definition**
  - **Initialization**
  - **Operators**
  - **Variable Reference**
  - **Pointers Arithmetic's**

Pointers - Definition

- **A pointers is a variable containing a memory address**



Pointers - Definition (cnt'd)

- **While defining a pointer, it requires to use the * special character**

Pointers – Definition *(cnt'd)*

- **Example of pointer definition:**
  - `int *p;`

p **holds memory address of an *integer* variable**

Pointers – Definition *(cnt'd)*

- **Example of pointer definition:**
  - `int *p;`
  - `char *p;`

***p* holds memory address of *char* variable**

Pointers – Definition *(cnt'd)*

- **Example of pointer definition:**
  - `int *p;`
  - `char *p;`
  - `double *p;`

***p* holds memory address of *double* variable**

Pointers – Definition *(cnt'd)*

- **Example of pointer definition:**
  - `int *p;`
  - `char *p;`
  - `double *p;`
  - `void *p;`

***p* holds *generic* memory address**

Pointers – Definition *(cnt'd)*

- **Example of pointer definition:**
  - `int *p;`
  - `char *p;`
  - `double *p;`
  - `void *p;`

Each pointer contains a memory address

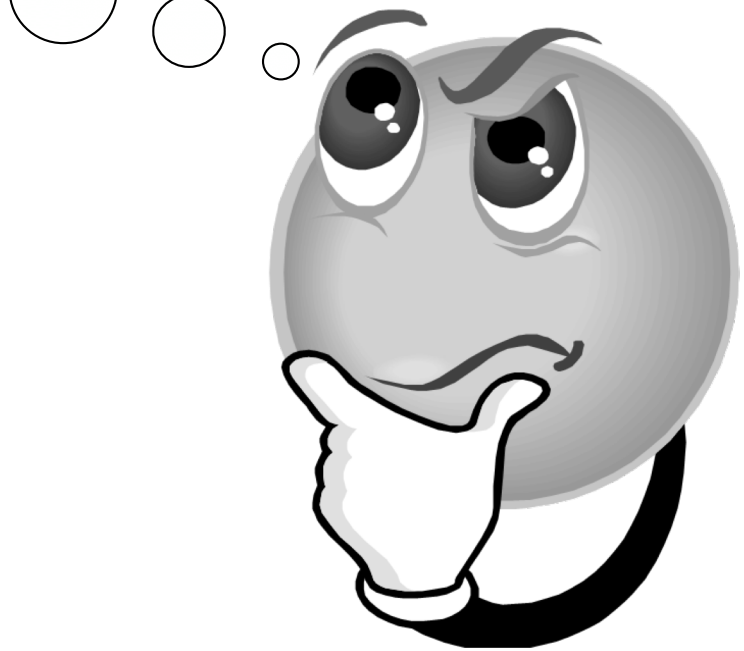
Pointers – Definition *(cnt'd)*

- **Example of pointer definition:**
  - `int *p;`
  - `char *p;`
  - `double *p;`
  - `void *p;`

Each pointer contains a memory address

All pointers have the same size regardless what they point to !!!

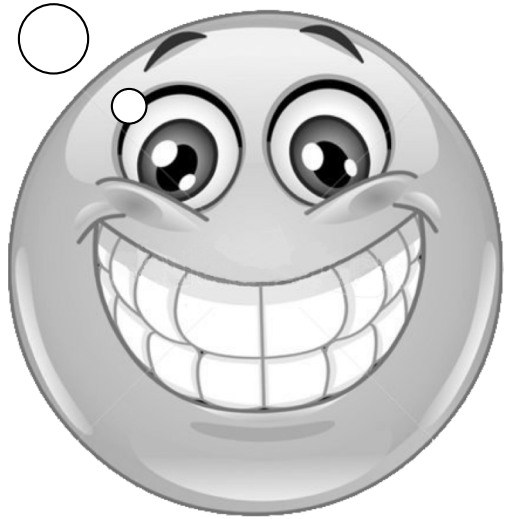
Why do we need to specify the pointer type?



We need to know what type of variable we are going to get when we *dereference* (i.e., use) the pointer

Implicit conversions are illegal!

```
int *p;
double f;
p = &f; //ERROR
```



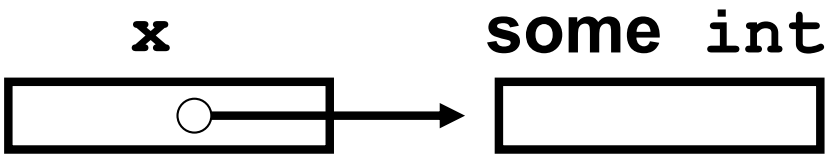
Pointers – Definition (cnt'd)

- **Multiple pointers definition require a `*` before each variable definition:**
  - `int *p, *p1, *x, **y;`

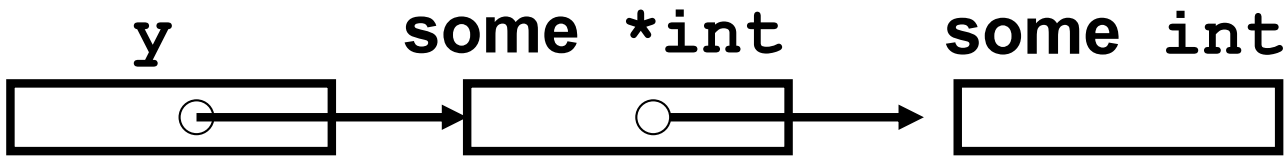
Pointers – Definition *(cnt'd)*

- **Multiple pointers definition require a `*` before each variable definition:**
  - `int *p, *p1, *x, **y;`

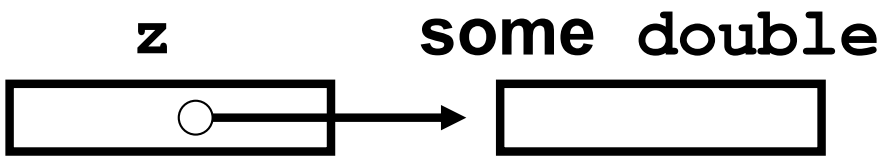
`int *x;`



`int **y;`



`double *z;`



Outline

- **Pointers:**
  - **Definition**
  - **Initialization**
  - **Operators**
  - **Variable Reference**
  - **Pointers Arithmetic's**

Pointers – Initialization

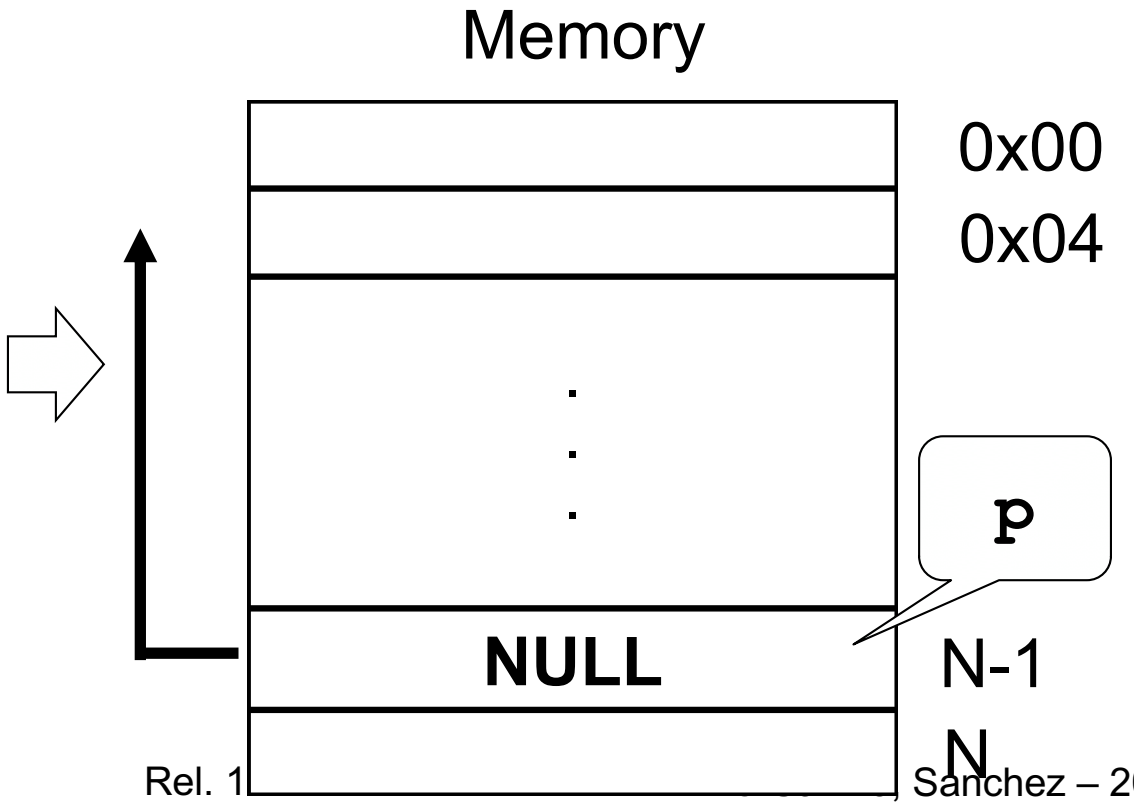
- **A pointer can be initialized to:**
  - **0**
  - **NULL / nullptr**
  - **an address**

Pointers – Initialization *(cnt'd)*

- **A pointer can be initialized to:**
  - **0**
  - **NULL / nullptr**
  - **an address**

The pointer points to <u>nothing</u>

```
int *p = NULL
```

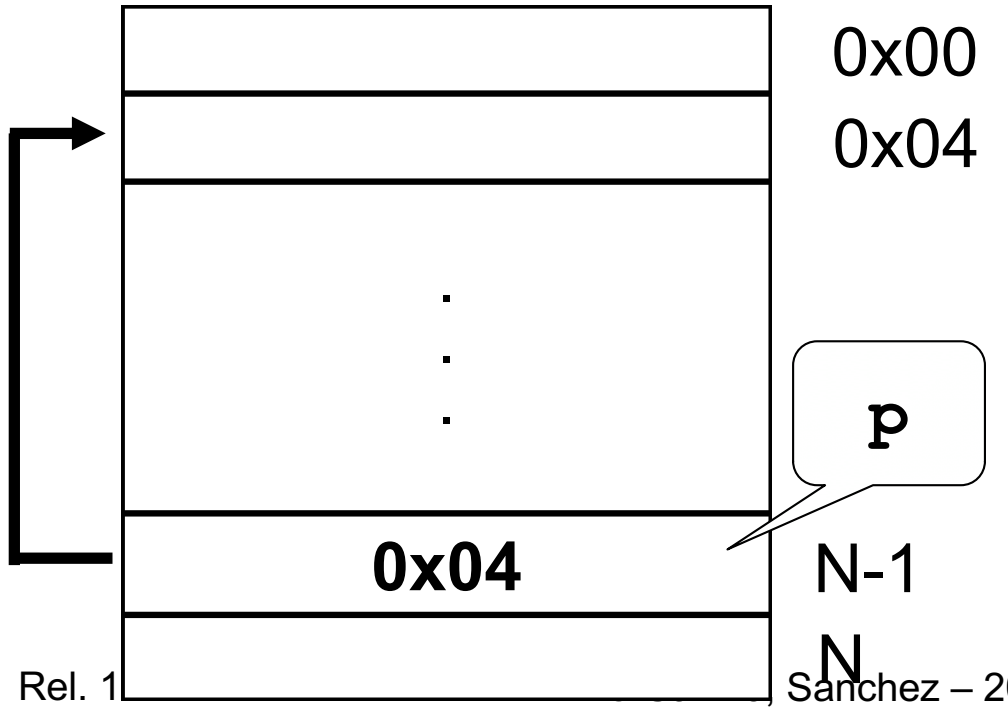


Pointers – Initialization *(cnt'd)*

- **A pointer can be initialized to:**
  - **0**
  - **NULL**
  - **an address**

The pointer points to <u>an address</u>

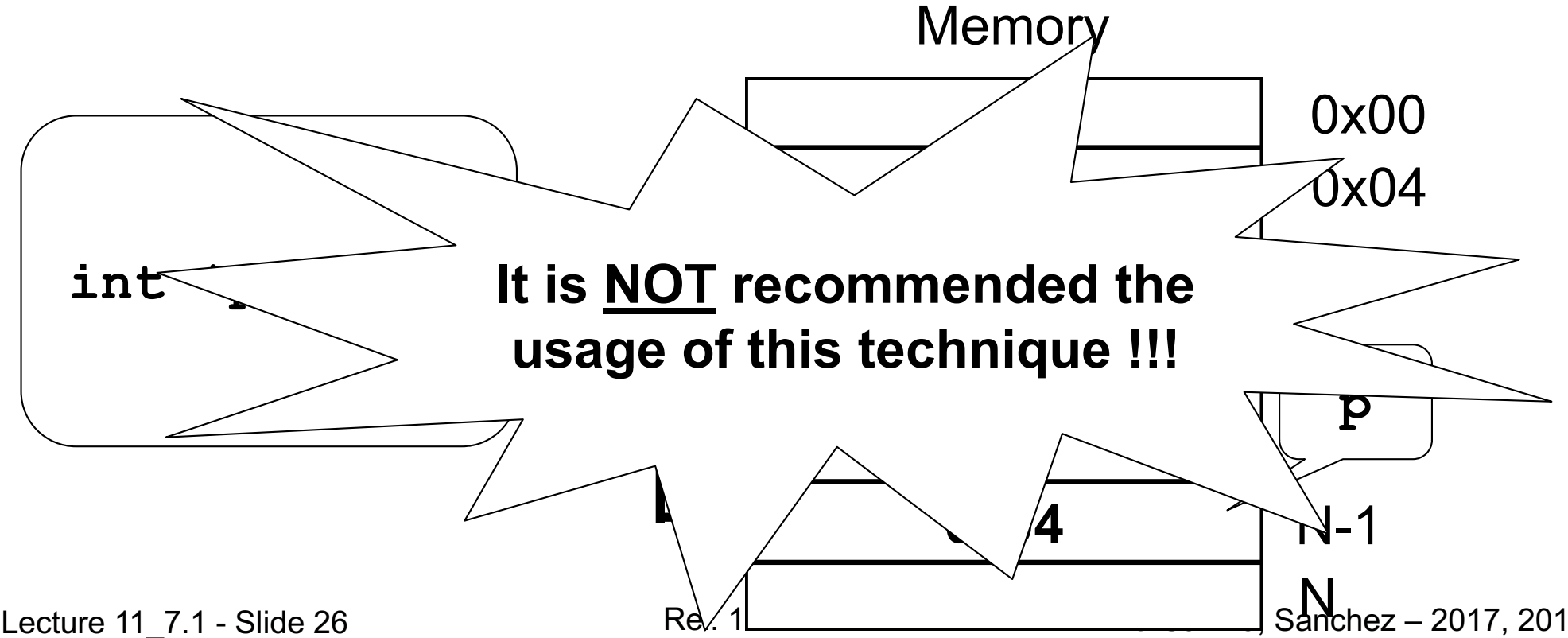
```
int *p = 0x04
```



Pointers – Initialization *(cnt'd)*

- **A pointer can be initialized to:**
  - **0**
  - **NULL**
  - **an address**

The pointer points to <u>an address</u>



It is <u>NOT</u> recommended the usage of this technique !!!

Outline

- **Pointers:**
  - **Definition**
  - **Initialization**
  - **Operators**
  - **Variable Reference**
  - **Pointers Arithmetic's**

Pointers – Operators

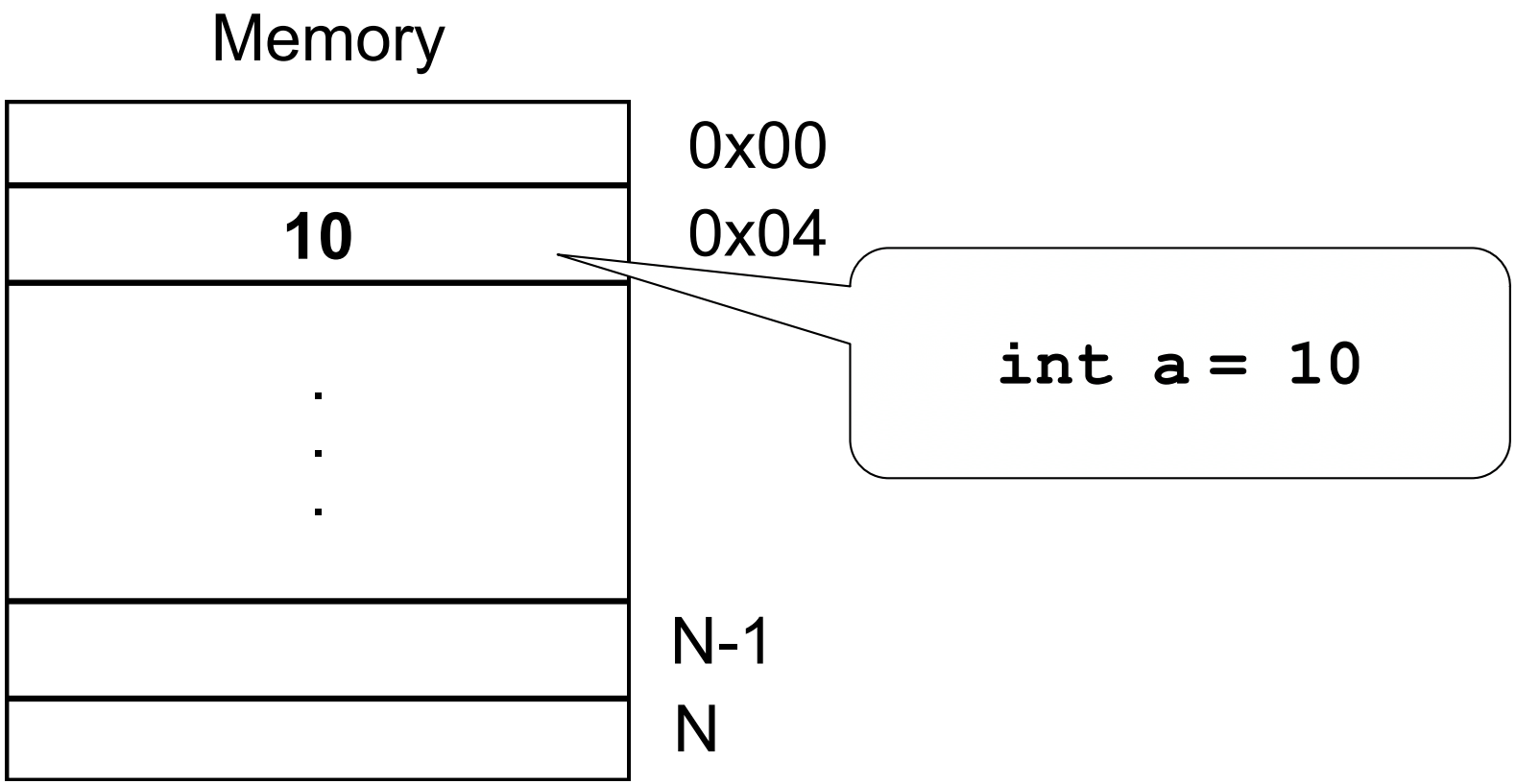
- **Two operators can be used with pointers:**
  - **&**
  - *****

Pointers – & Operator

- **The & operator is called the *reference* operator**
- **It returns the address of an operand.**

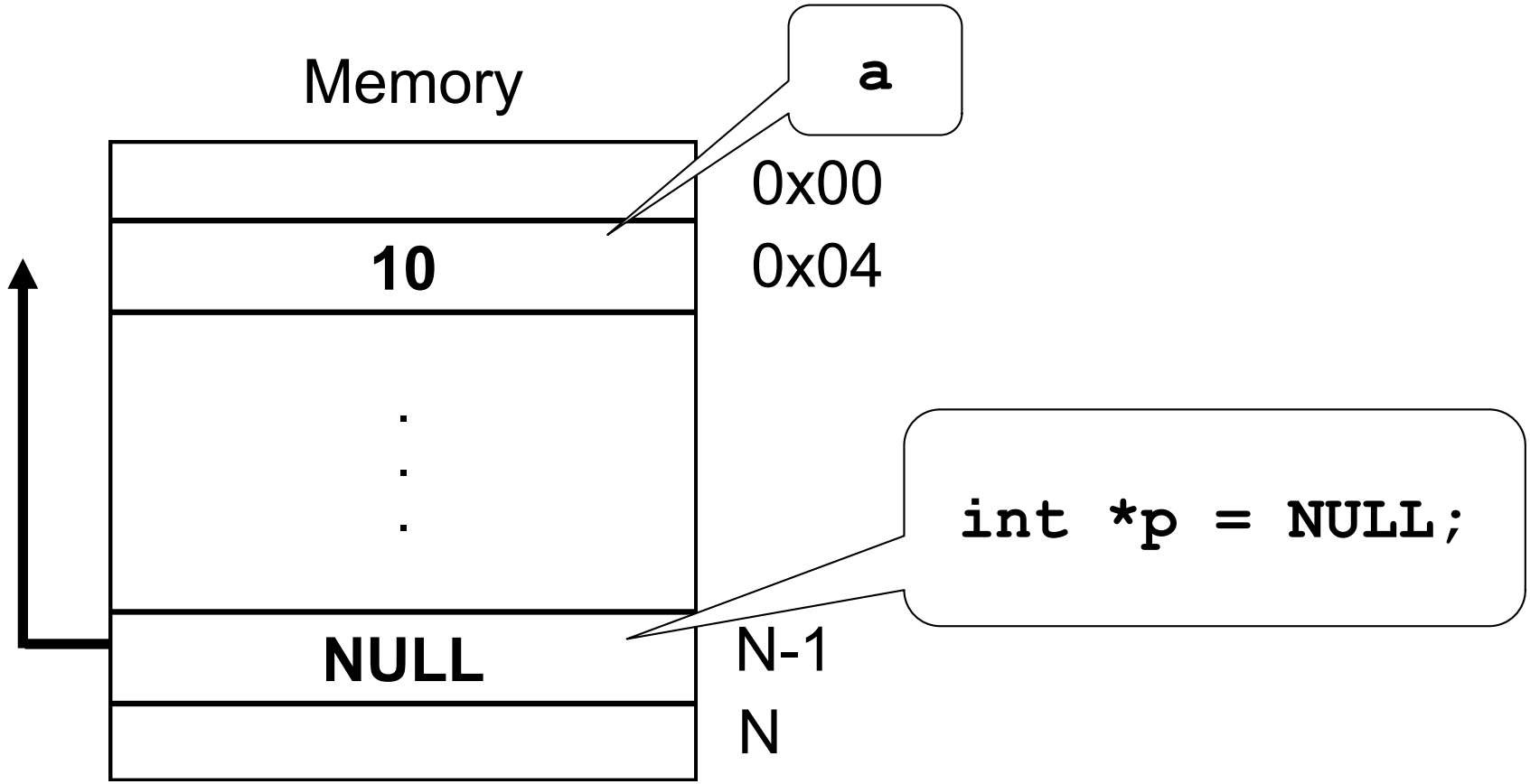
Pointers – & Operator

- **The & operator is called the *reference* operator**
- **It returns the address of an operand.**



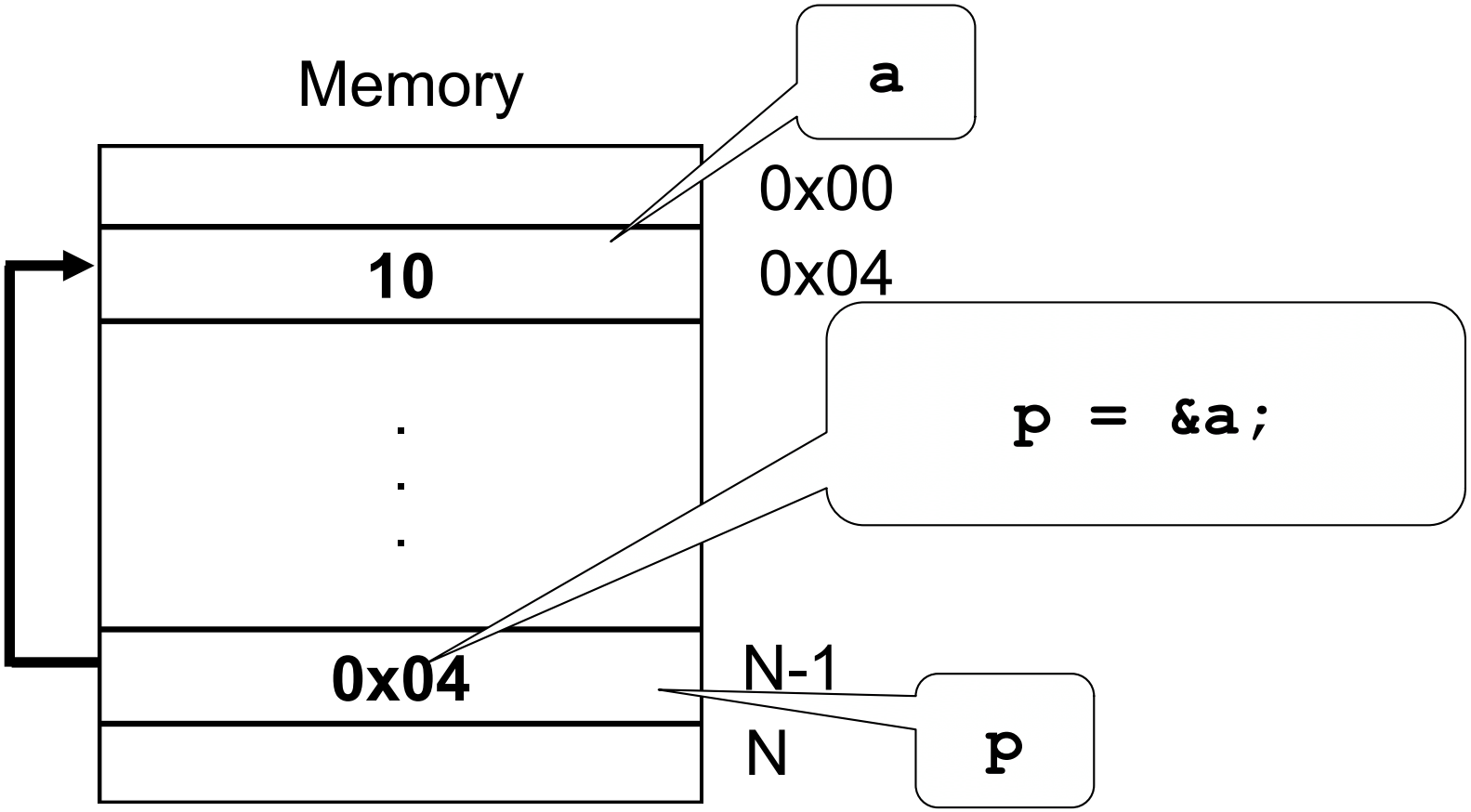
Pointers – & Operator

- **The & operator is called the *reference* operator**
- **It returns the address of an operand.**



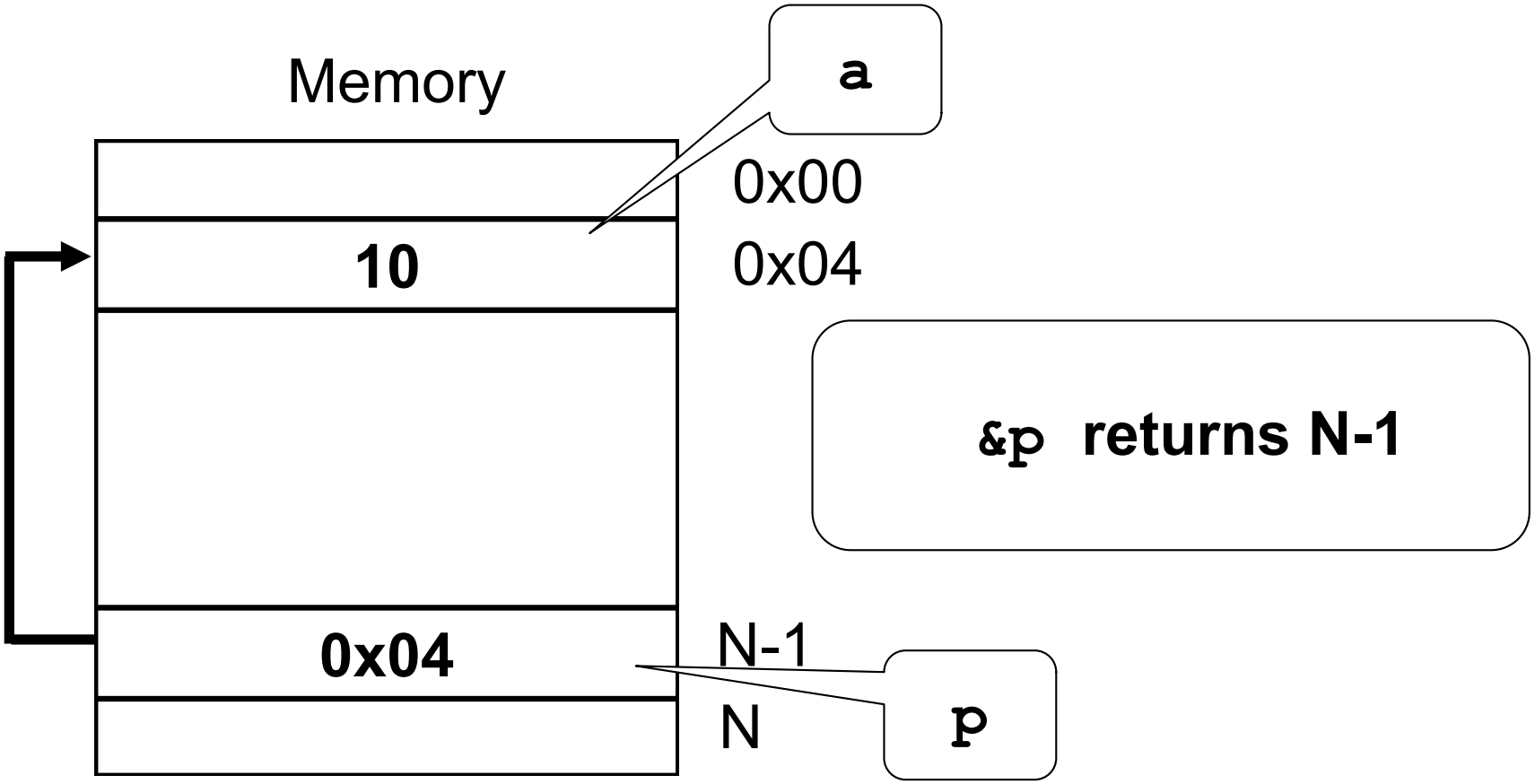
Pointers – & Operator

- **The & operator is called the *reference* operator**
- **It returns the address of an operand.**



Pointers – & Operator

- The & operator is called the *reference* operator
- It returns the address of an operand.

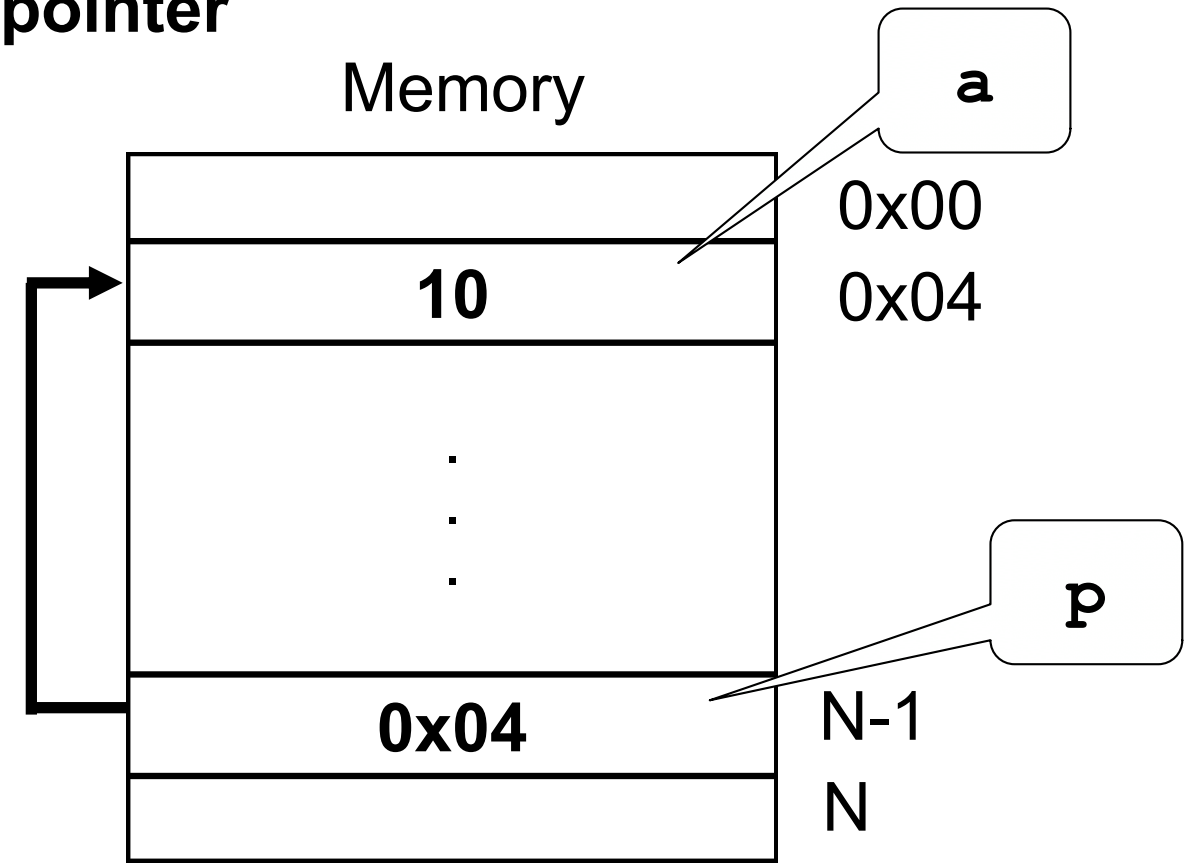


*Pointers – * Operator*

- **The * operator is called the *indirection* or *dereferencing* operator**
- **Returns the value of the object pointed by the pointer**

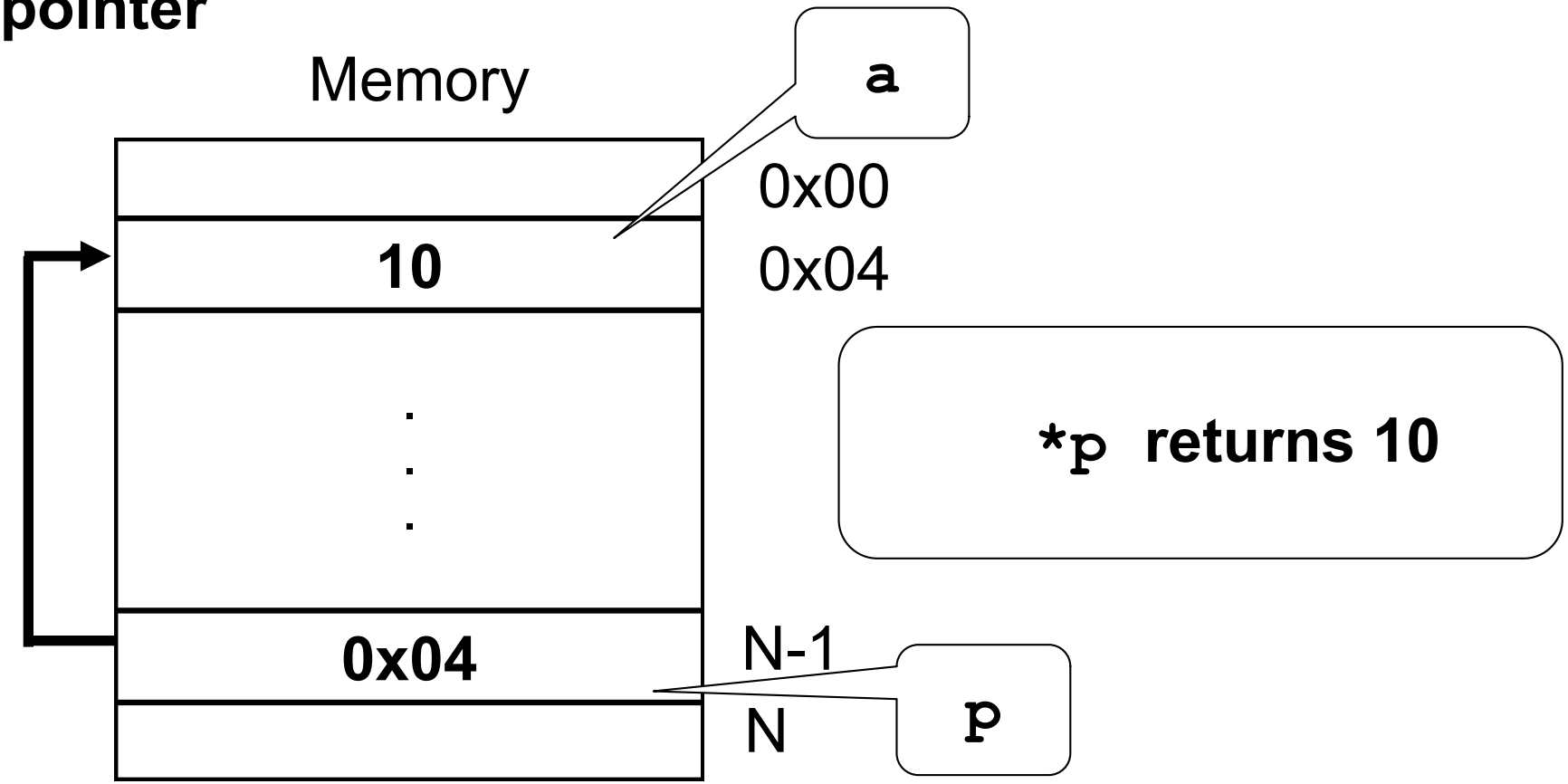
Pointers – * Operator

- **The * operator is called the *indirection* or *dereferencing* operator**
- **Returns the value of the object pointed by the pointer**



Pointers – * Operator

- **The * operator is called the *indirection* or *dereferencing* operator**
- **Returns the value of the object pointed by the pointer**



Pointers – Operators Example

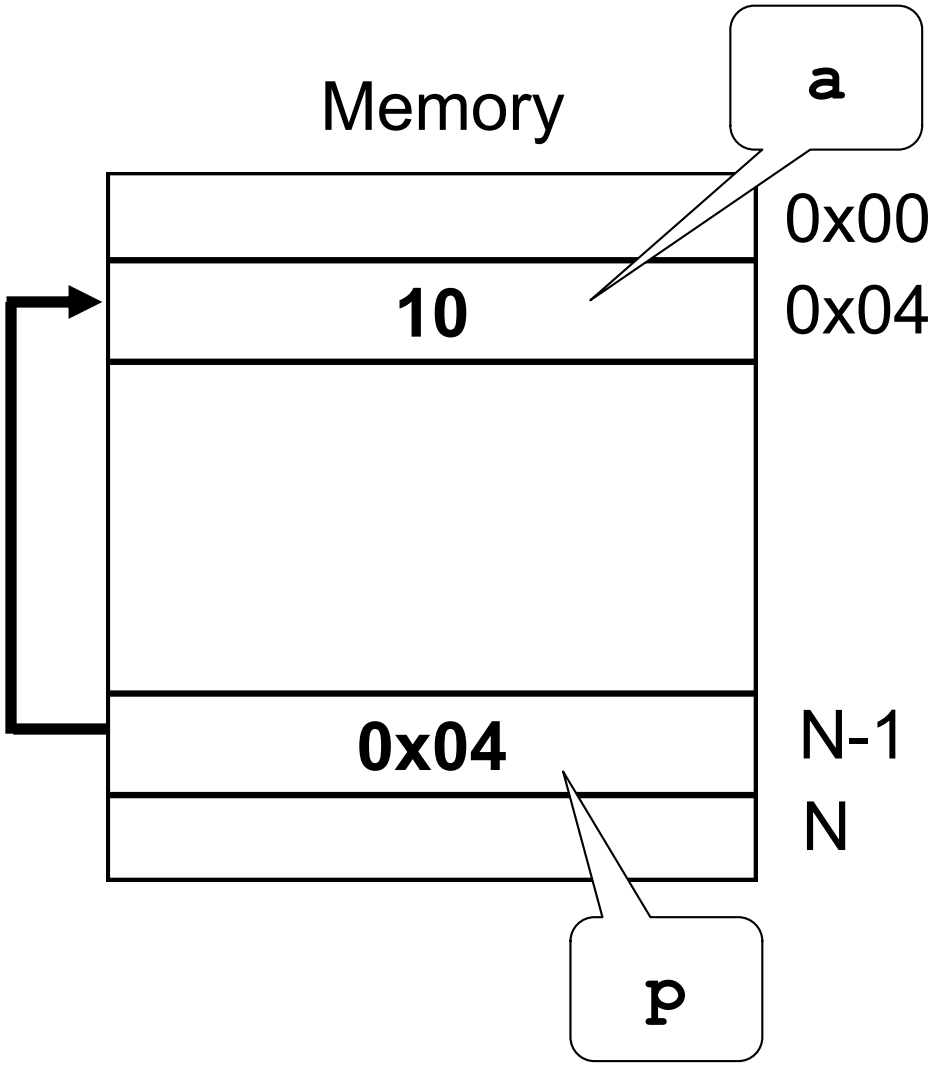
```
int x = 1;
int y = 2;

int *ip;  /*ip is a pointer to int */

ip  = &x; /*ip now points to x*/

y = *ip;  /*y is now 1*/

*ip = 0;  /*x is now 0 */
```

Pointers – Operators

- **The `*` and the `&` are inverse: they cancel each other**

Pointers – Operators

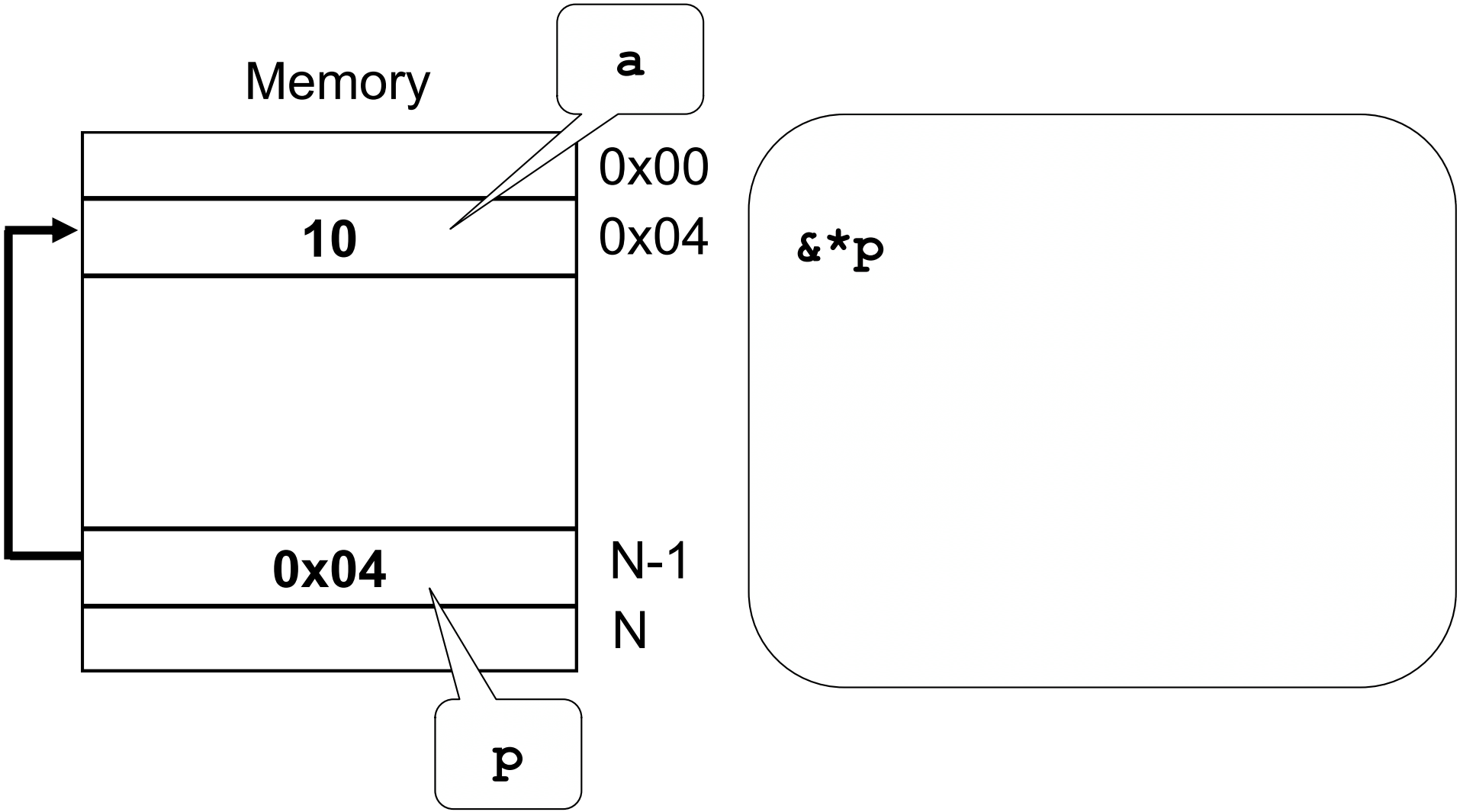
- **The * and the & are inverse: they cancel each other**



&p return N-1
***p return 10**

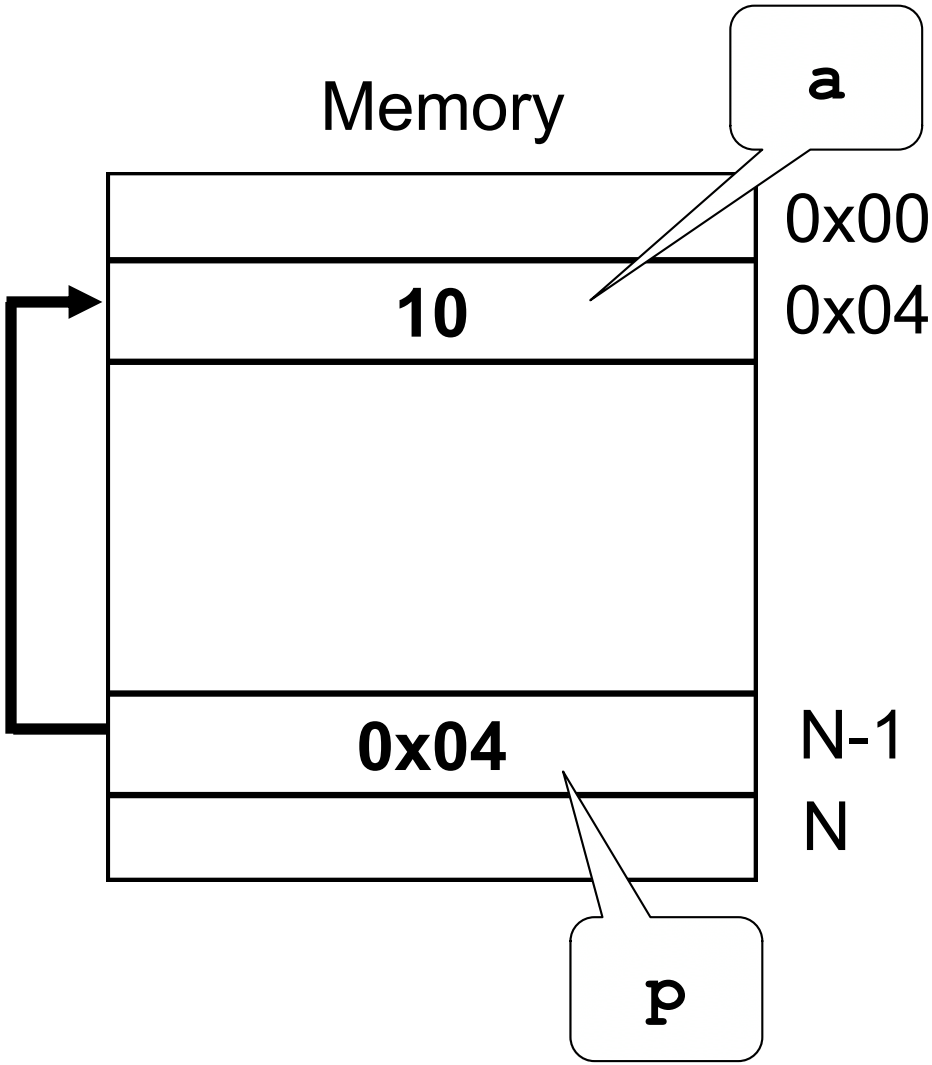
Pointers – Operators

- **The * and the & are inverse: they cancel each other**



Pointers – Operators

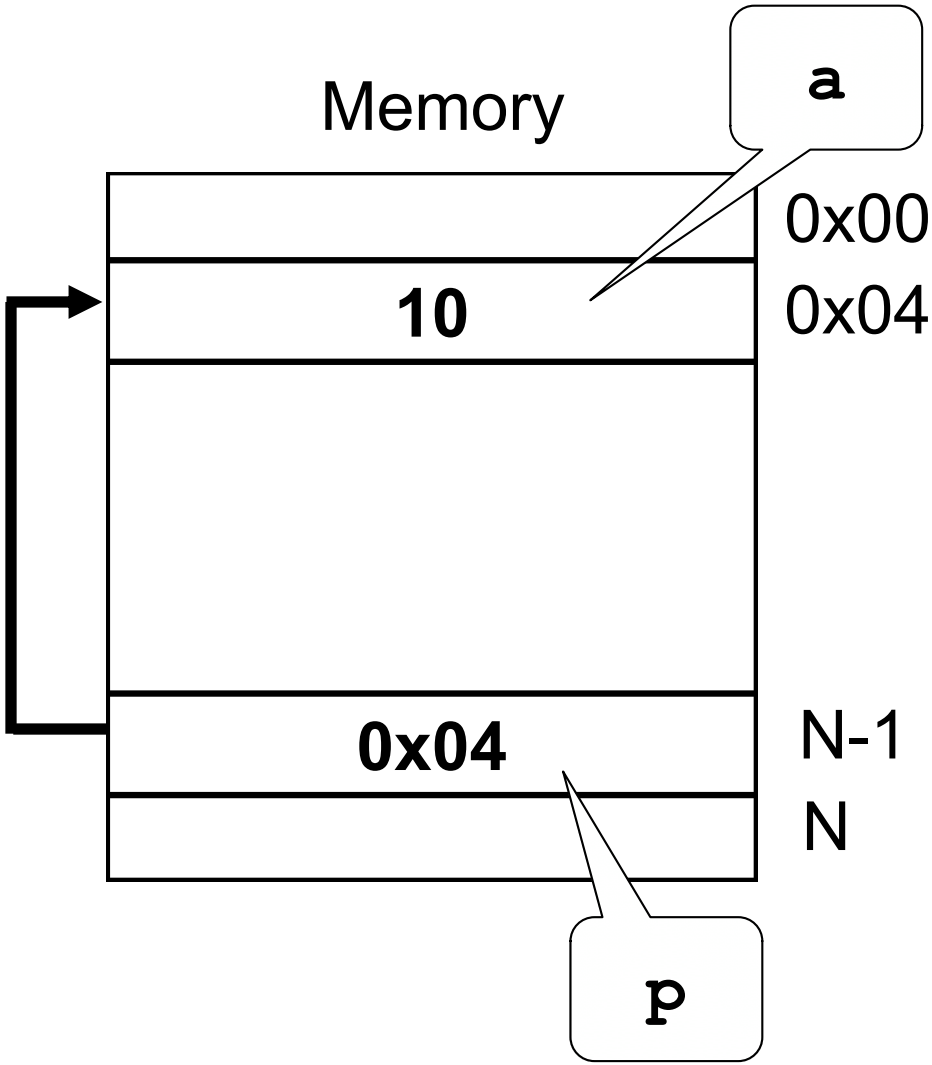
- **The * and the & are inverse: they cancel each other**




```
&*p ≡ &(*p)
```

Pointers – Operators

- **The `*` and the `&` are inverse: they cancel each other**

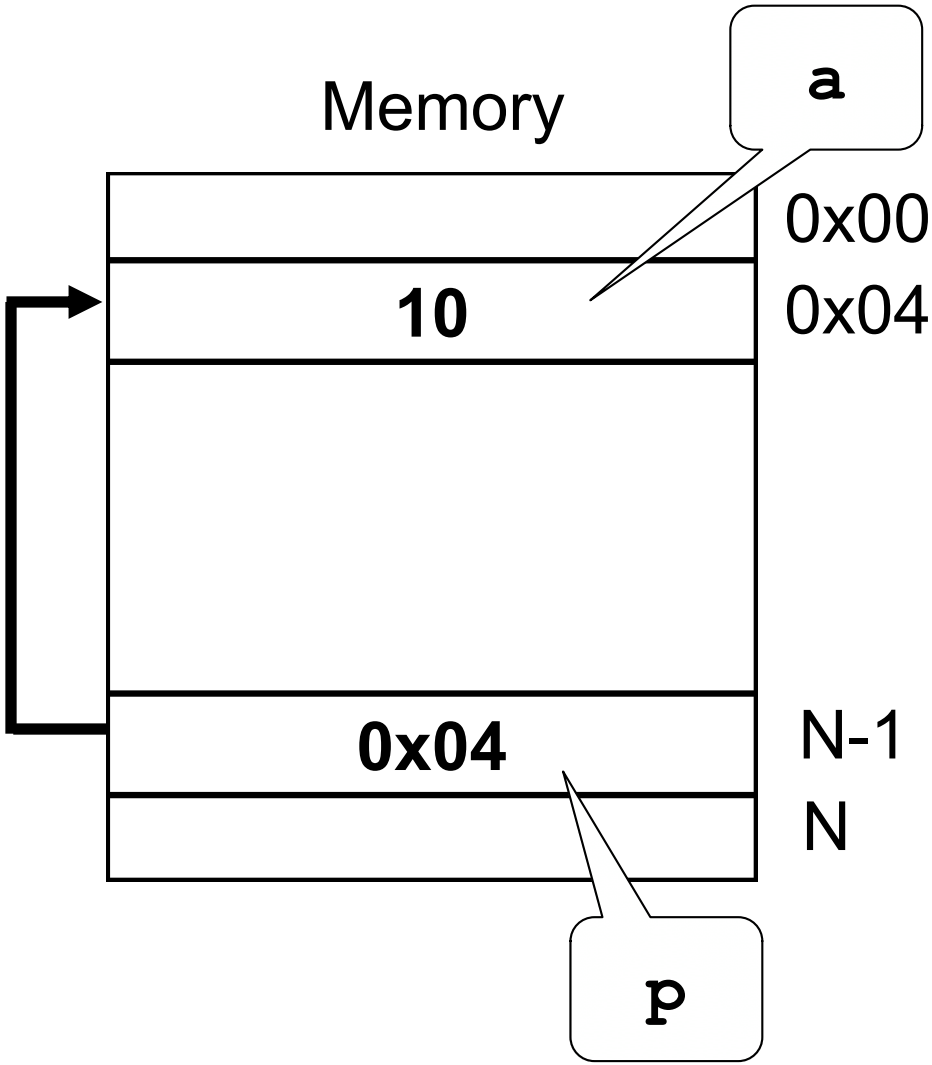


`&*p` ≡ `&(*p)`

1. **`*p` :**
   returns the content of cell pointed by `p` (i.e., the variable `a`)

Pointers – Operators

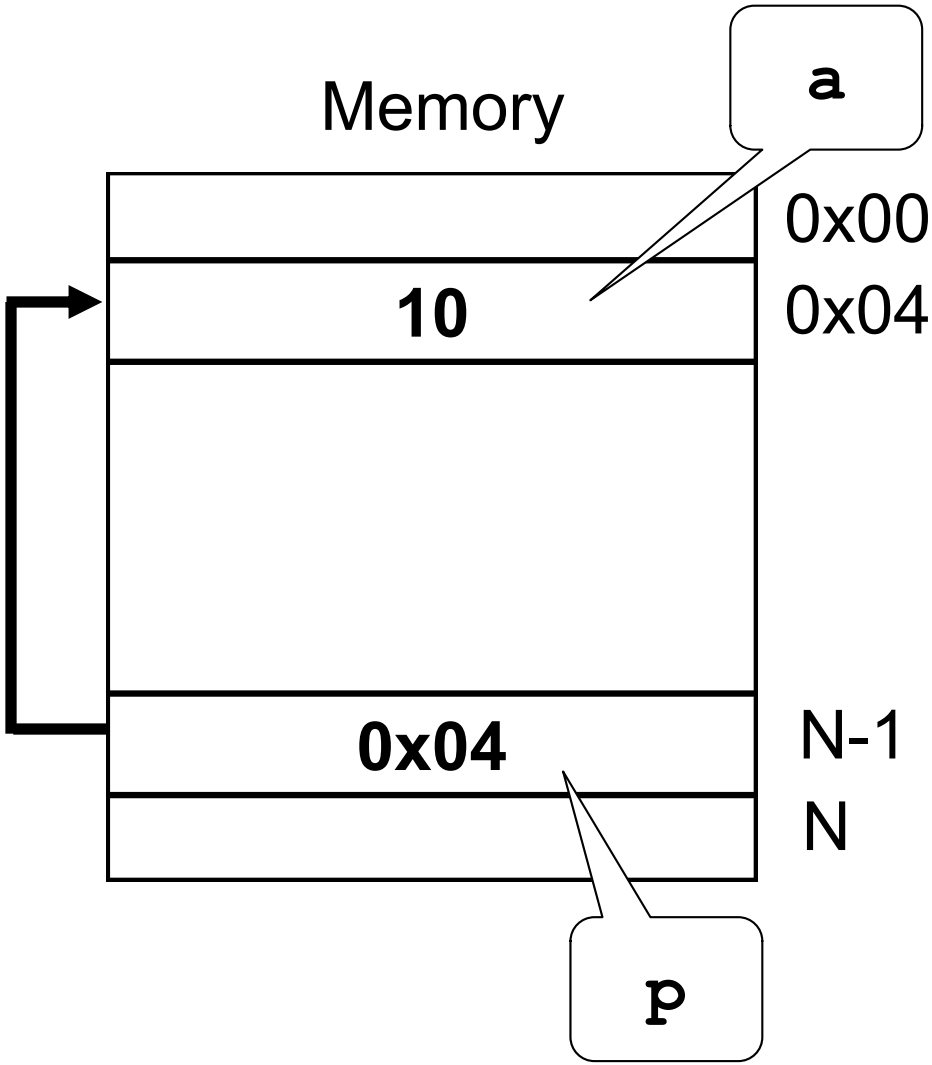
- **The * and the & are inverse: they cancel each other**




```
&*p ≡ &(*p)
   1.  *p ≡ a
```

Pointers – Operators

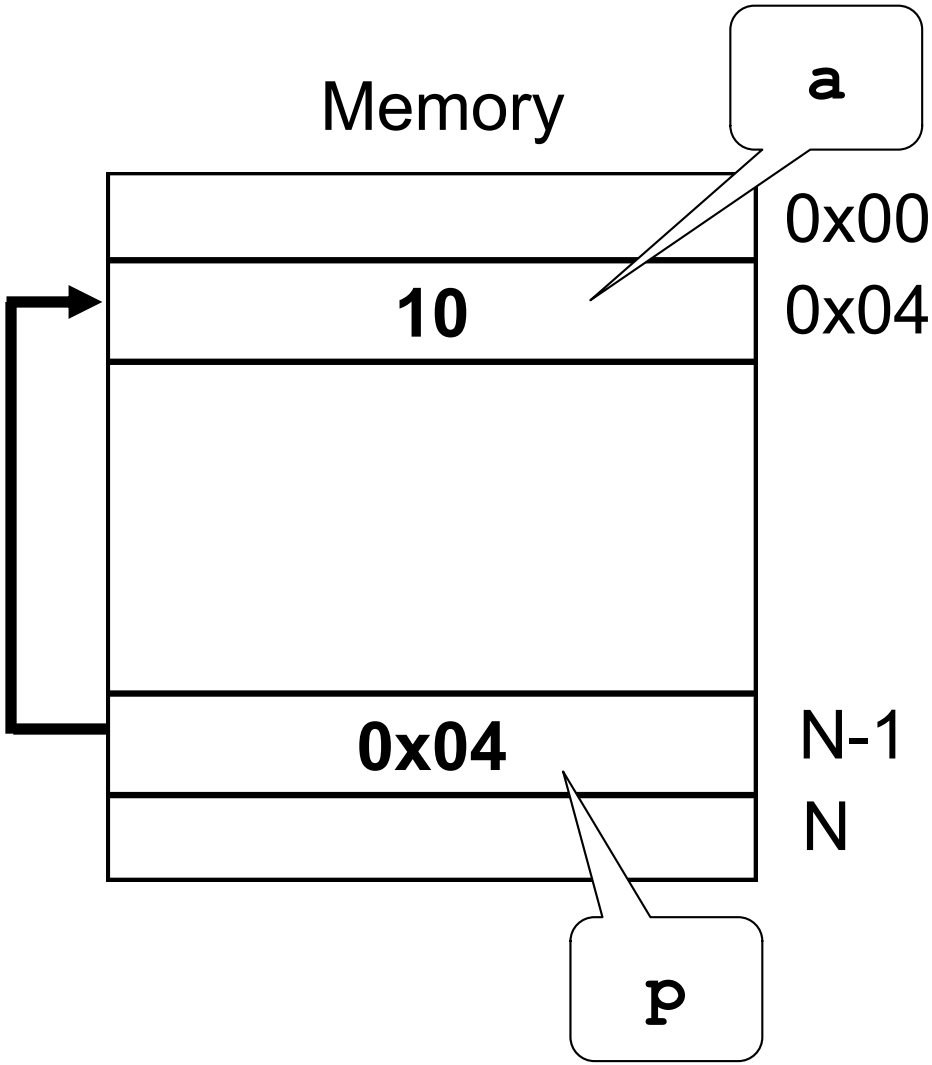
- **The * and the & are inverse: they cancel each other**




```
&*p ≡ &(*p) ≡ &(a)
   1.  *p ≡ a
```

Pointers – Operators

- **The * and the & are inverse: they cancel each other**

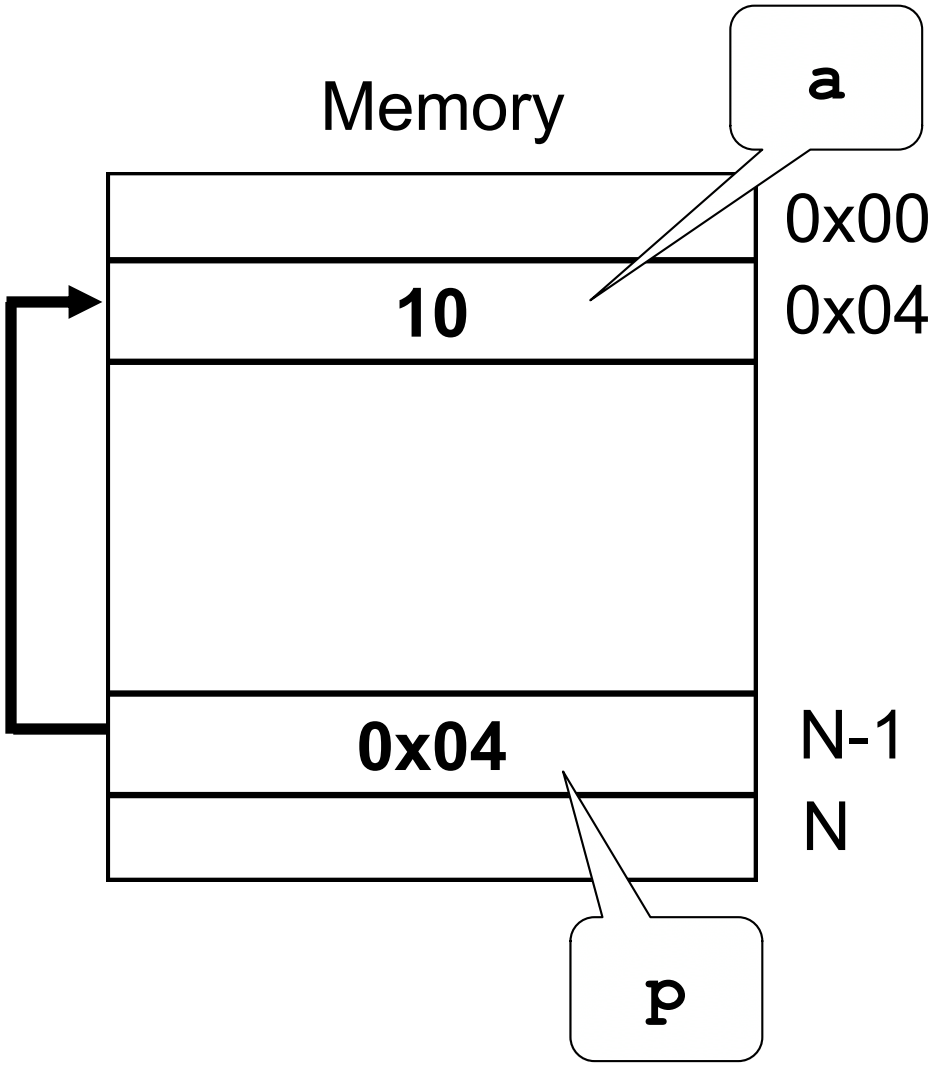


`&*p ≡ &(*p) ≡ &(a)`

1. **`*p ≡ a`**
2. **`&(a)`: returns the address of the variable `a`**

Pointers – Operators

- **The * and the & are inverse: they cancel each other**

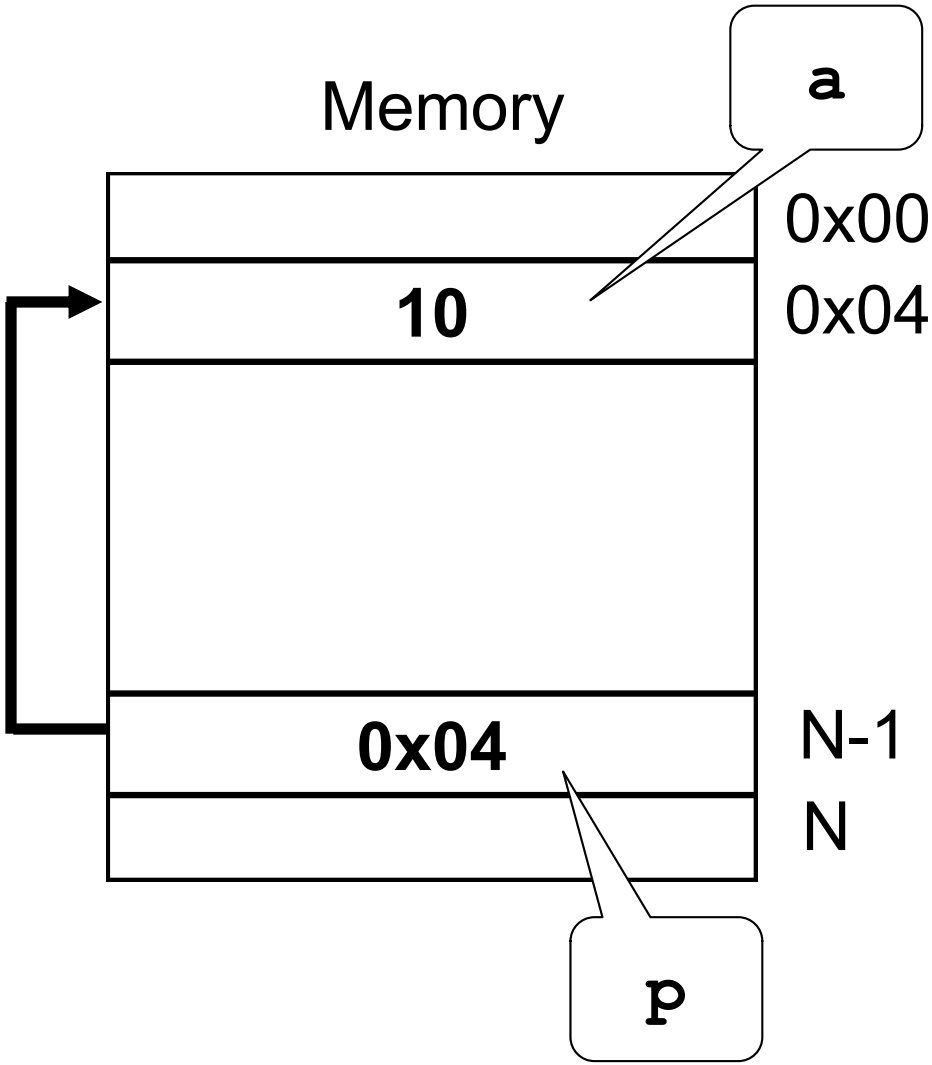



```
&*p ≡ &(*p) ≡ &(a)
```

1. `*p ≡ a`
2. `&(a) ≡ 0x04`

Pointers – Operators

- **The * and the & are inverse: they cancel each other**

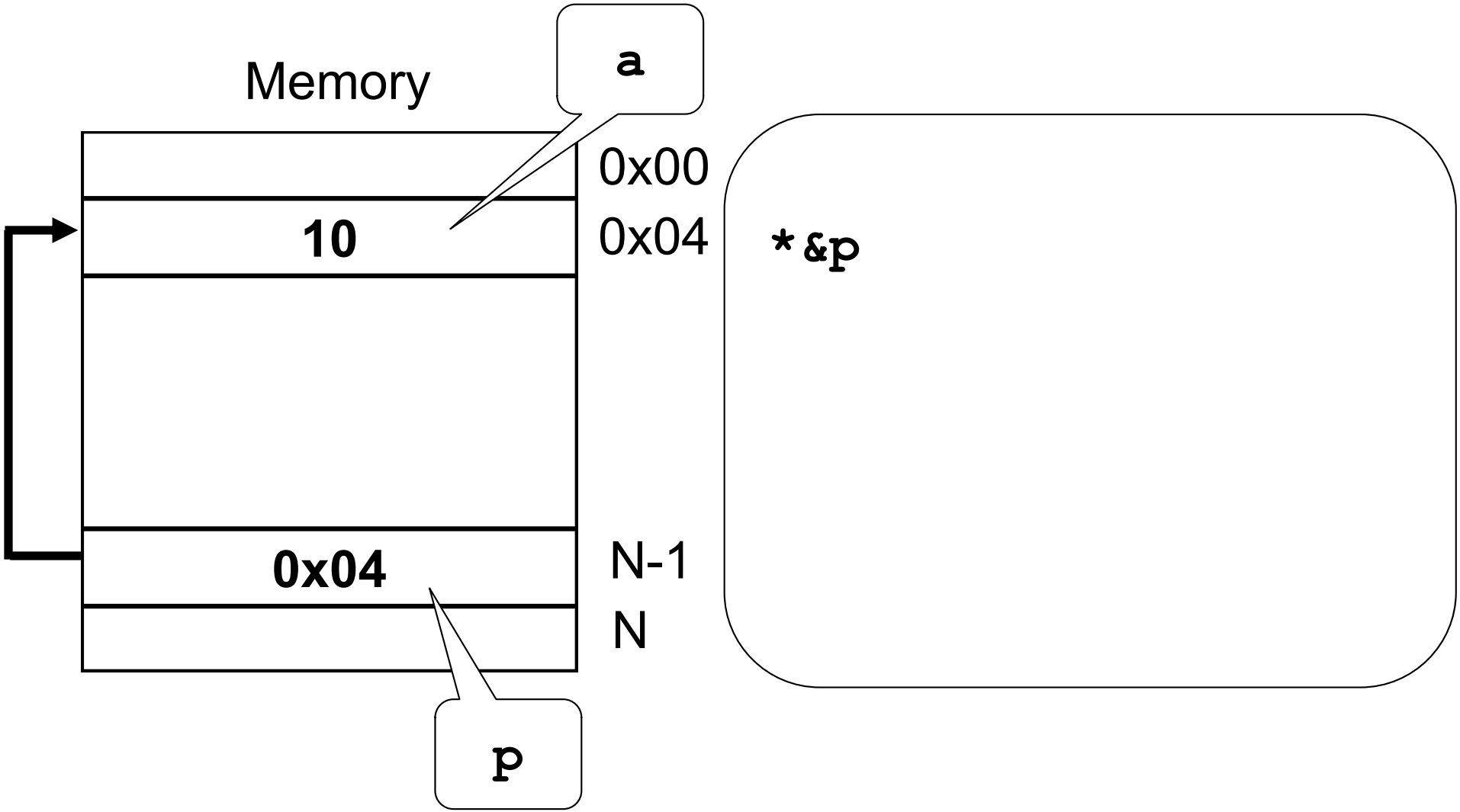



```
&*p ≡ &(*p) ≡ &(a) ≡
0x04
```

1. `*p ≡ a`
2. `&(a) ≡ 0x04`

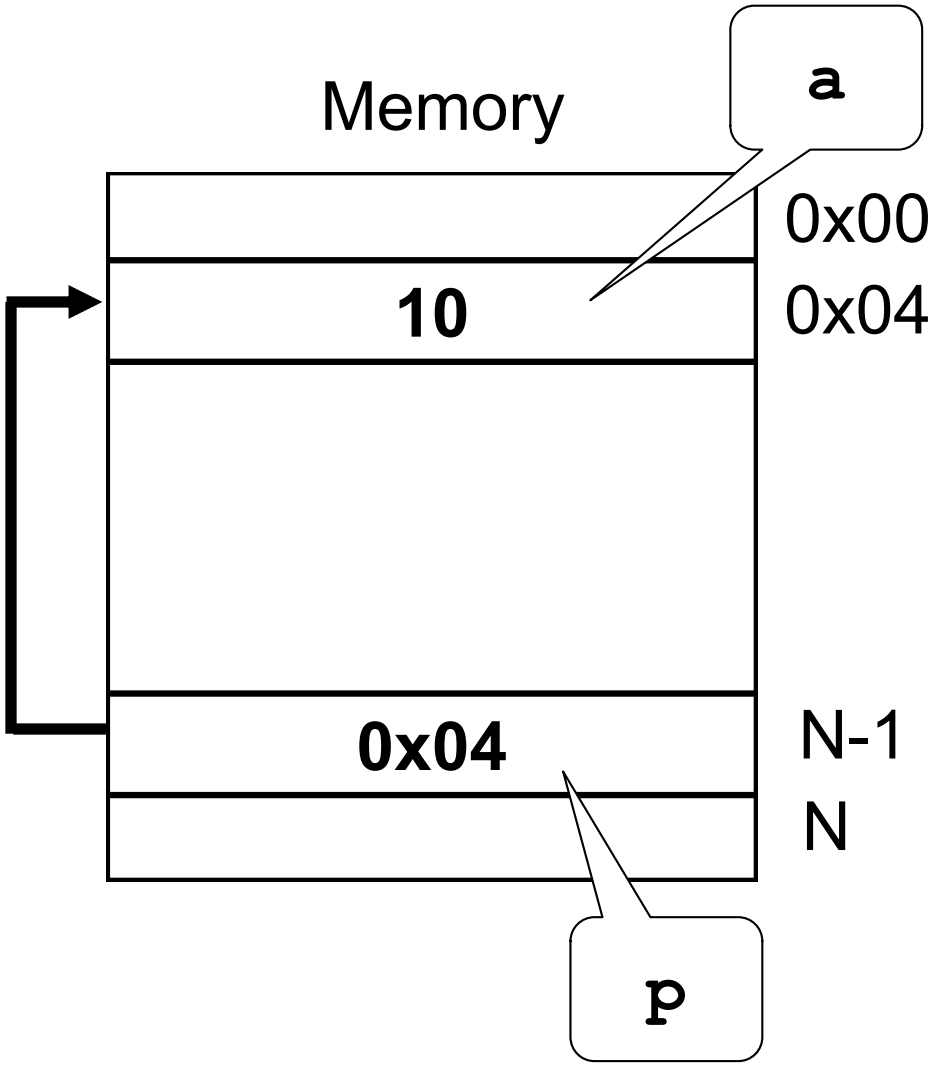
Pointers – Operators

- **The * and the & are inverse: they cancel each other**



Pointers – Operators

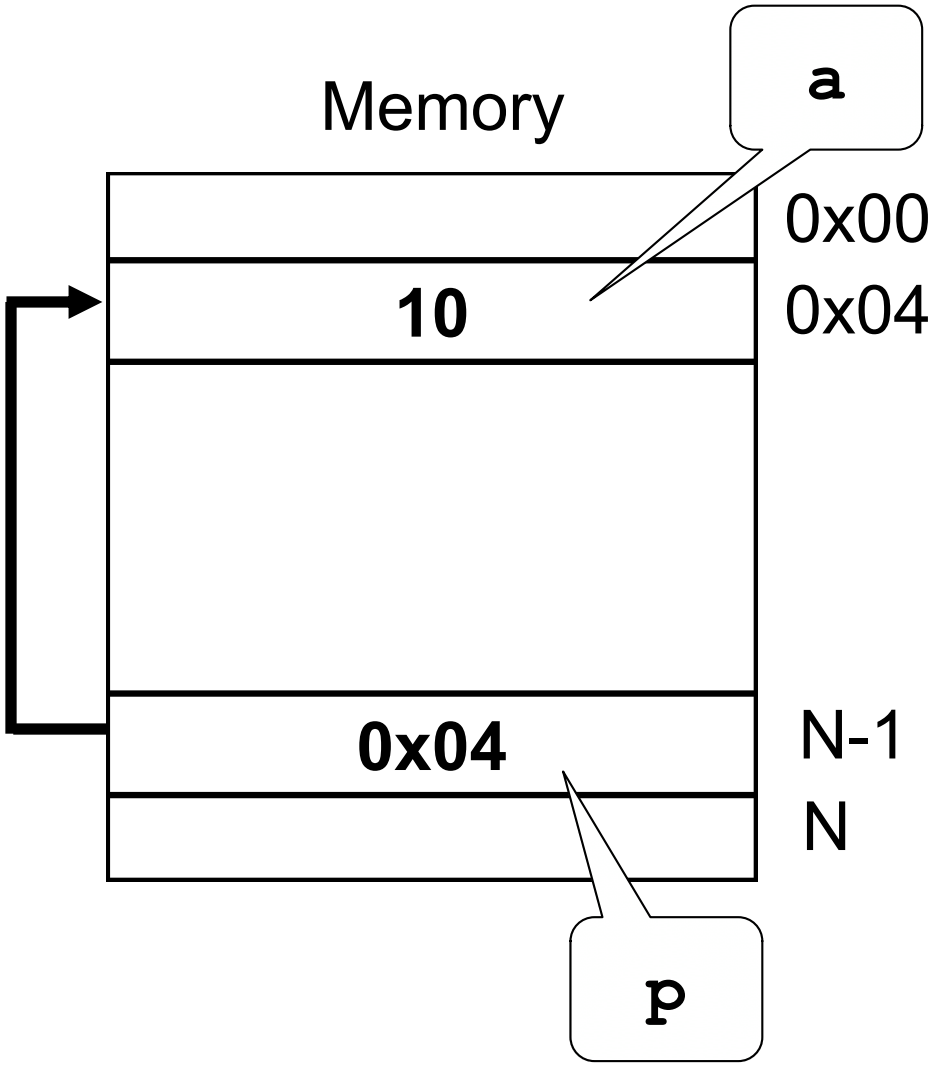
- **The * and the & are inverse: they cancel each other**




```
*&p ≡ *(&p)
```

Pointers – Operators

- **The * and the & are inverse: they cancel each other**

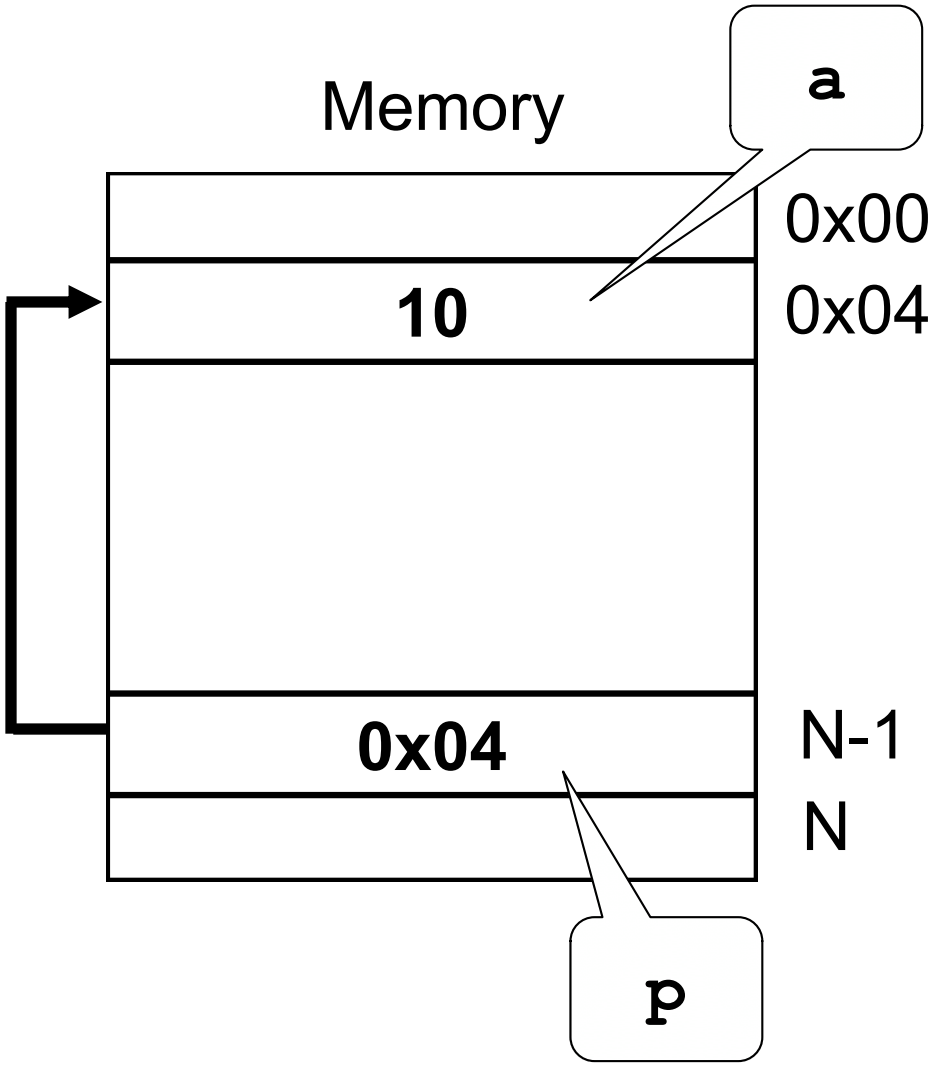


`*&p` ≡ `*(&p)`

1. **`&p` :**
   returns the address of the pointer `p` (i.e., N-1)

Pointers – Operators

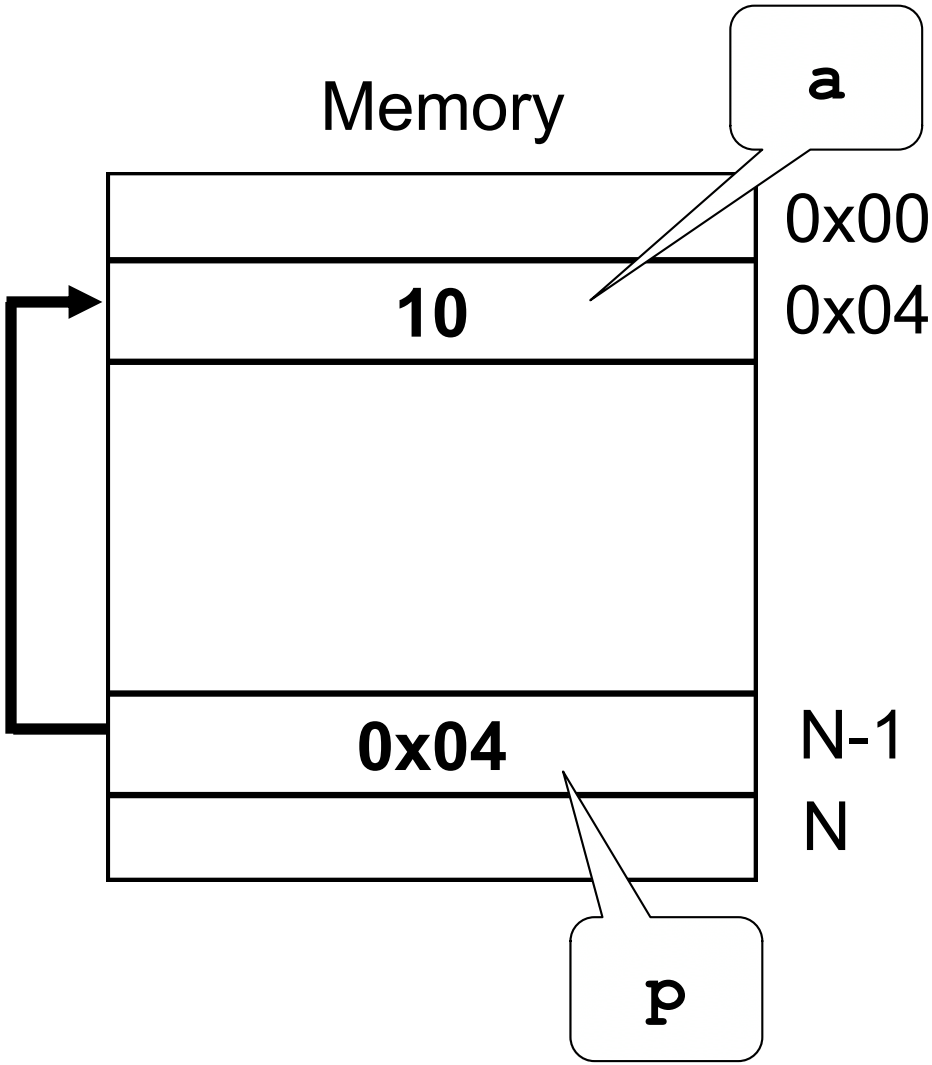
- **The * and the & are inverse: they cancel each other**




```
*&p ≡ *(&p)
   1.  &p ≡ N-1
```

Pointers – Operators

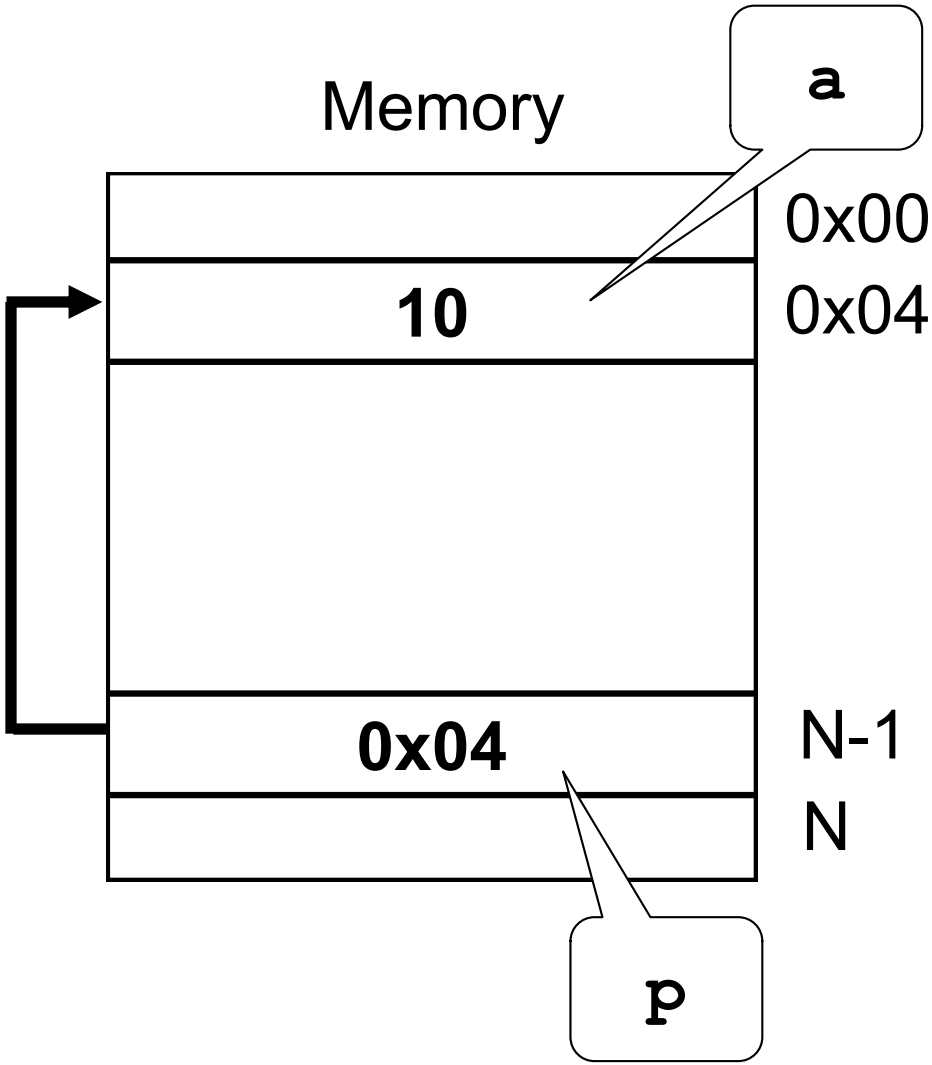
- **The * and the & are inverse: they cancel each other**




```
*&p ≡ *(&p) ≡ *(N-1)
   1.  &p ≡ N-1
```

Pointers – Operators

- **The * and the & are inverse: they cancel each other**

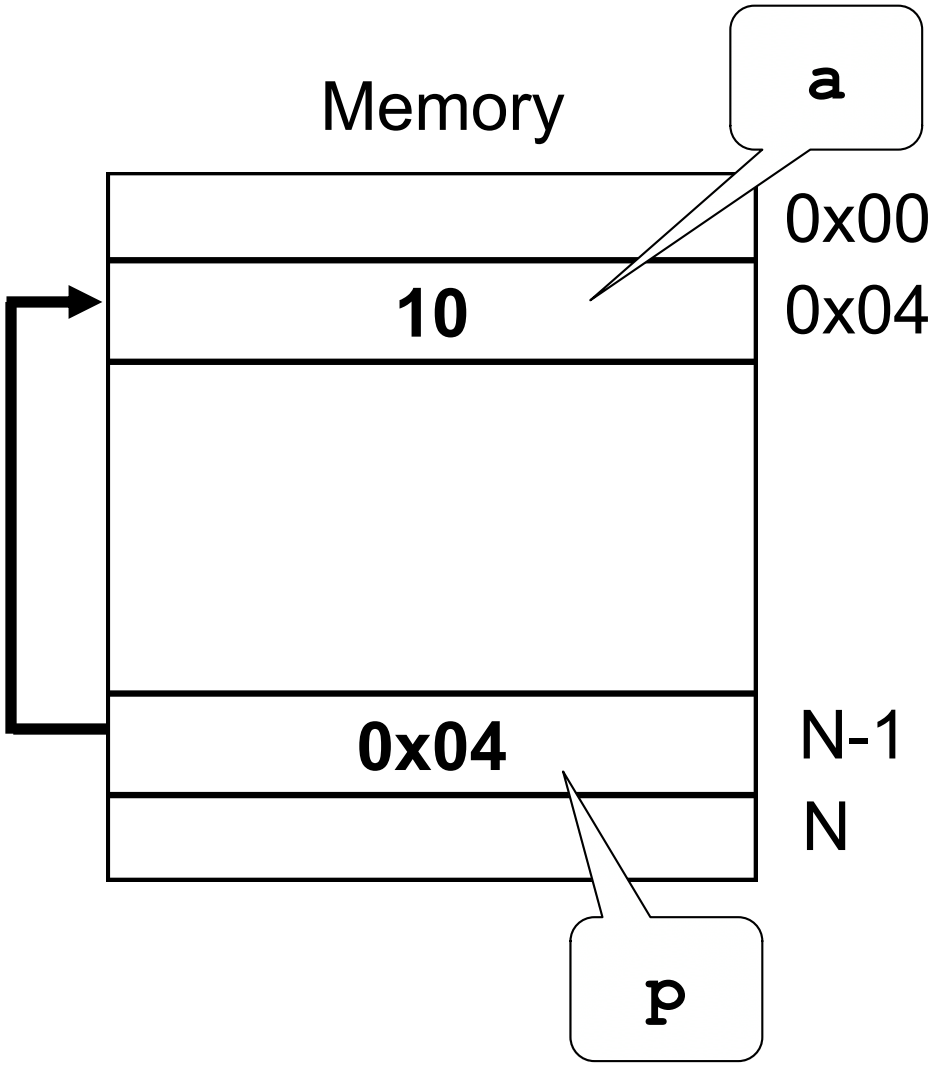


`*&p ≡ *(&p) ≡ *(N-1)`

1. **`&p ≡ N-1`**
2. **`*(N-1)`: returns the content of the cell at the `N-1` address**

Pointers – Operators

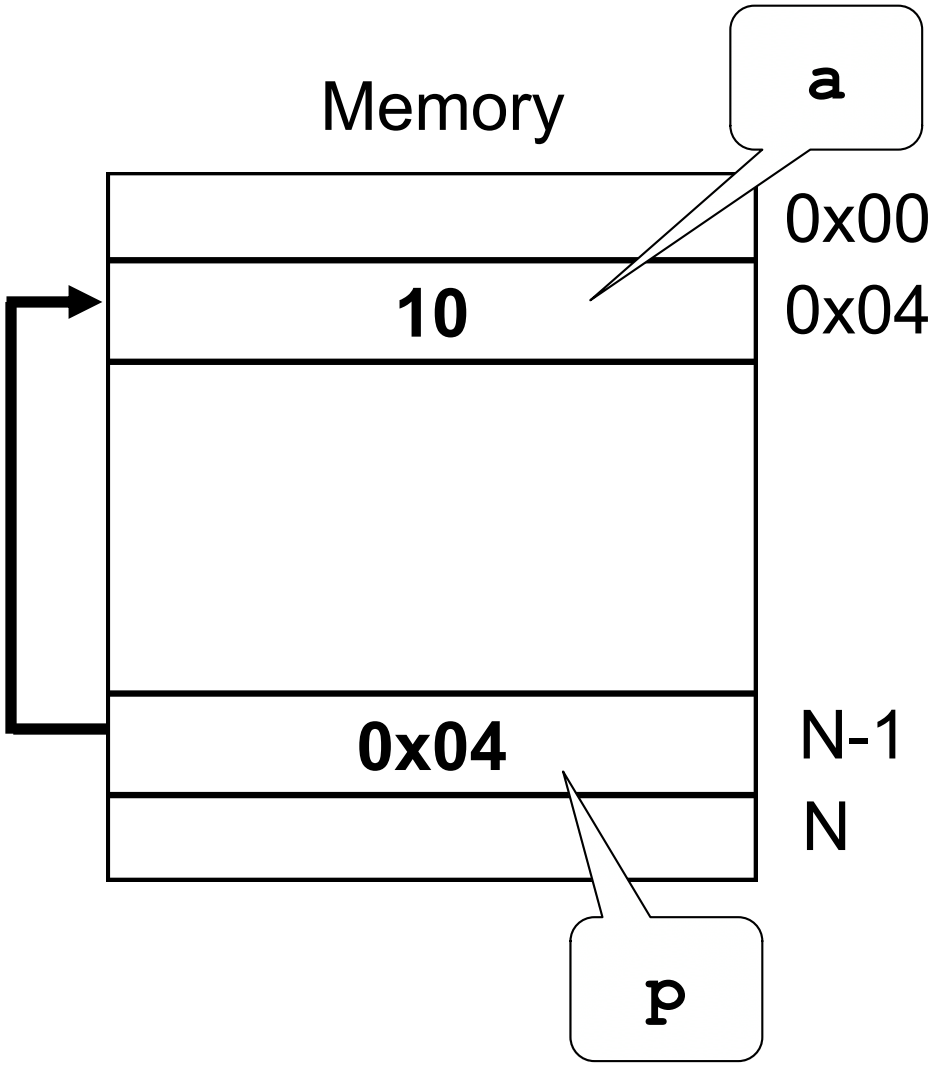
- **The * and the & are inverse: they cancel each other**




```
*&p ≡ *(&p) ≡ *(N-1)
   1.  &p ≡ N-1
   2.  *(N-1) ≡0x04
```

Pointers – Operators

- **The * and the & are inverse: they cancel each other**

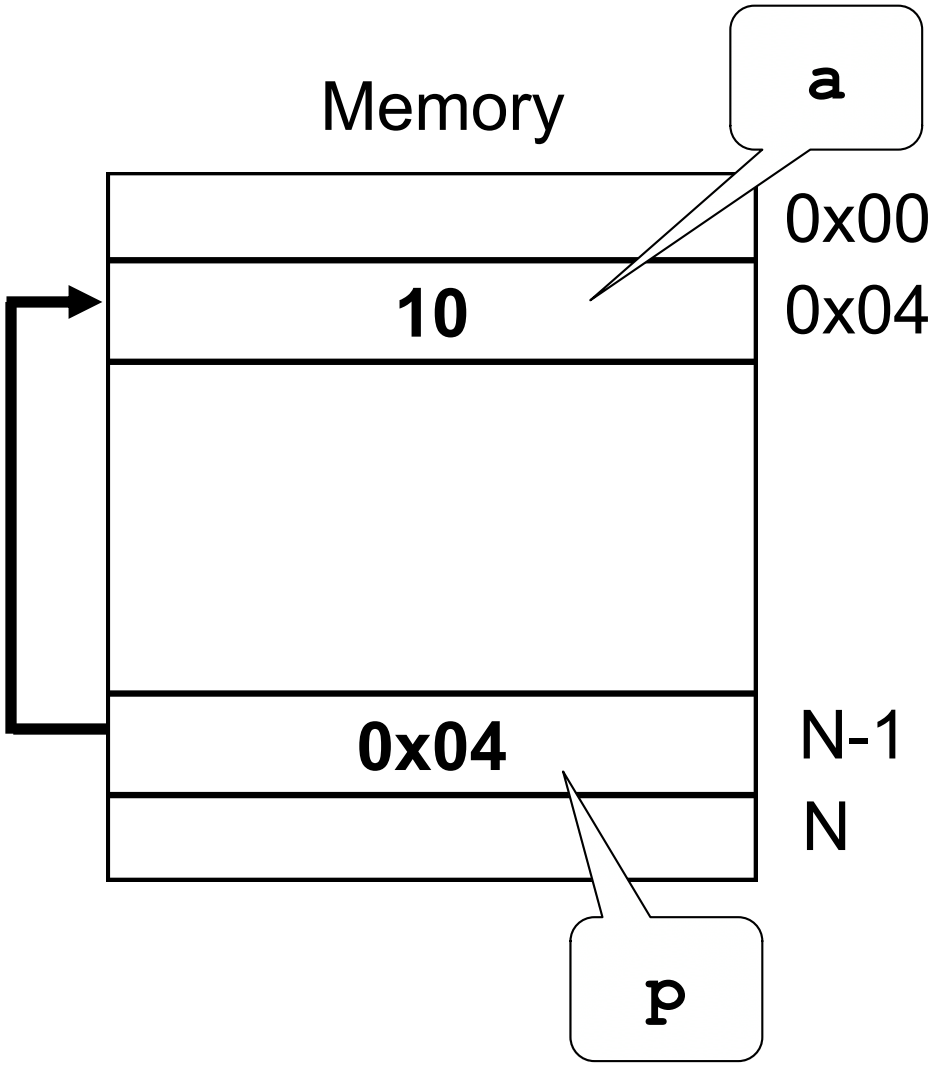



```
*&p ≡ *(&p) ≡ *(N-1) ≡
0x04
```

1. `&p ≡ N-1`
2. `*(N-1) ≡0x04`

Pointers – Operators

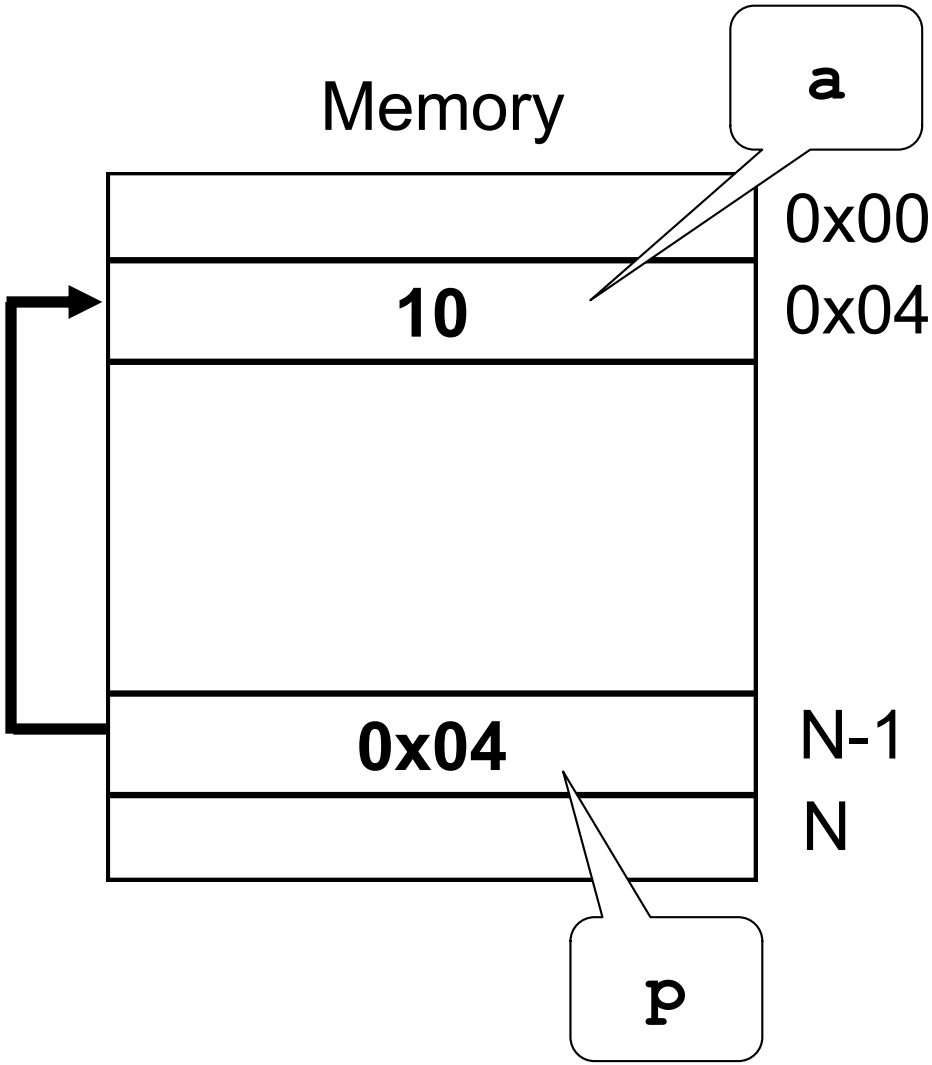
- **The * and the & are inverse: they cancel each other**




```
*&p ≡ 0x04
&*p ≡ 0x04
```

Pointers – Operators

- **The * and the & are inverse: they cancel each other**




```
*&p ≡ 0x04
&*p ≡ 0x04

*&p ≡ &*p
```

Outline

- **Pointers:**
  - **Definition**
  - **Initialization**
  - **Operators**
  - **Variable Reference**
  - **Pointers Arithmetic's**

How to reference a variable

- **There are two ways to use (i.e., *reference*) a variable:**
  - **Direct Variable Reference**
  - **Indirect Variable Reference**

Direct Variable Reference

- **<u>Direct</u> reference to a variable:**

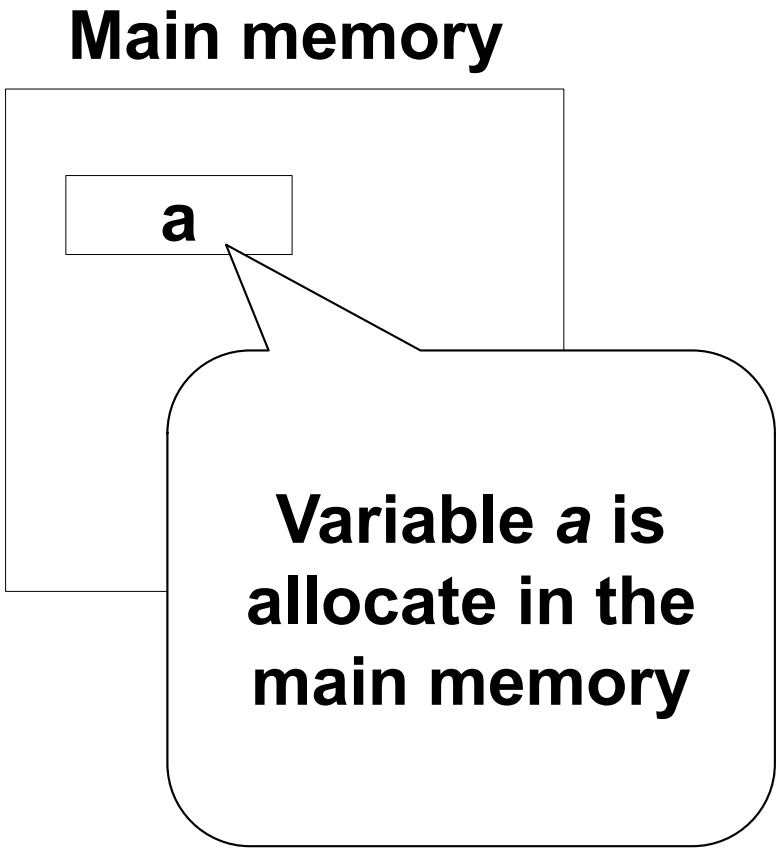
```
int a;
…
a=10;
```

Main memory

Direct Variable Reference (cnt'd)

- **<u>Direct</u> reference to a variable:**

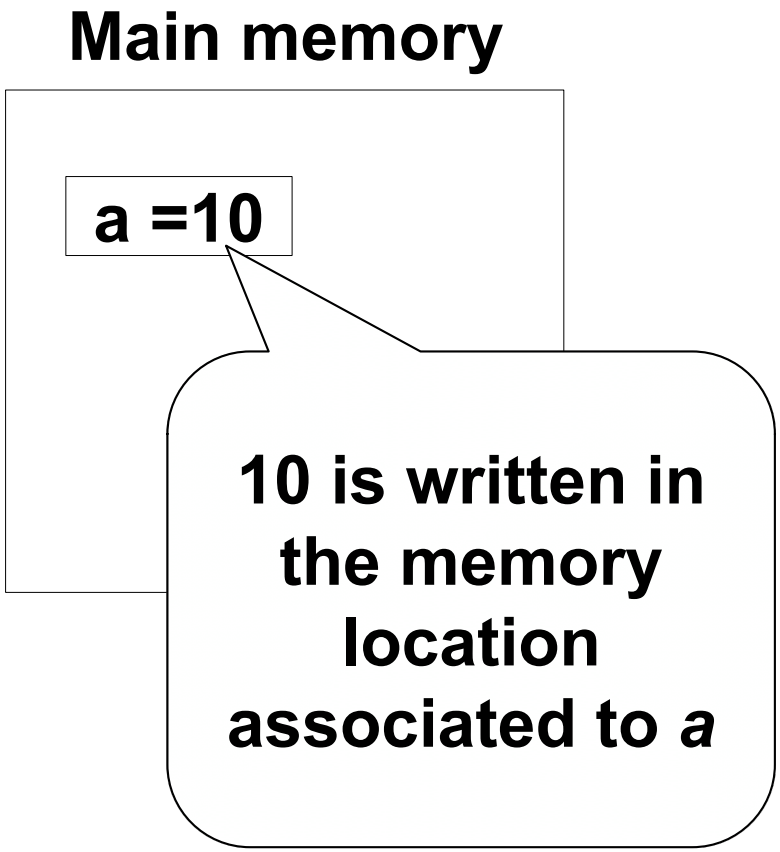
```
▷ int a;
  …
  a=10;
```



Direct Variable Reference

- **<u>Direct</u> reference to a variable:**

```
int a;
…
▷ a=10;
```



Indirect Variable Reference

- **<u>Indirect</u> reference to a variable :**

```
int a;
int *p;
…
p = &a;
*p = 10;
```

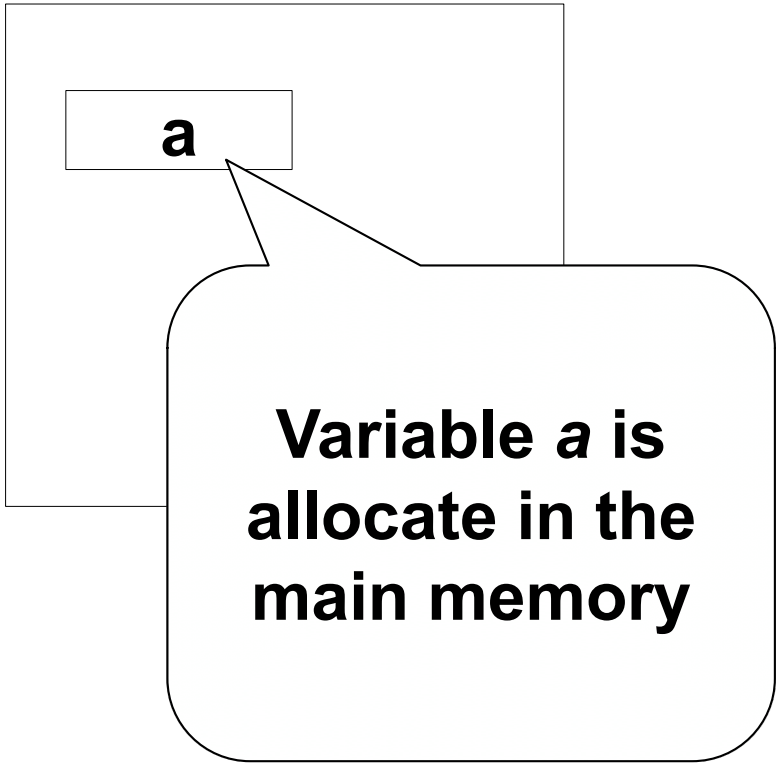
Main memory

Indirect Variable Reference (cnt'd)

- **<u>Indirect</u> reference to a variable:**

```
▷ int a;
  int *p;
  …
  p = &a;
  *p = 10;
```

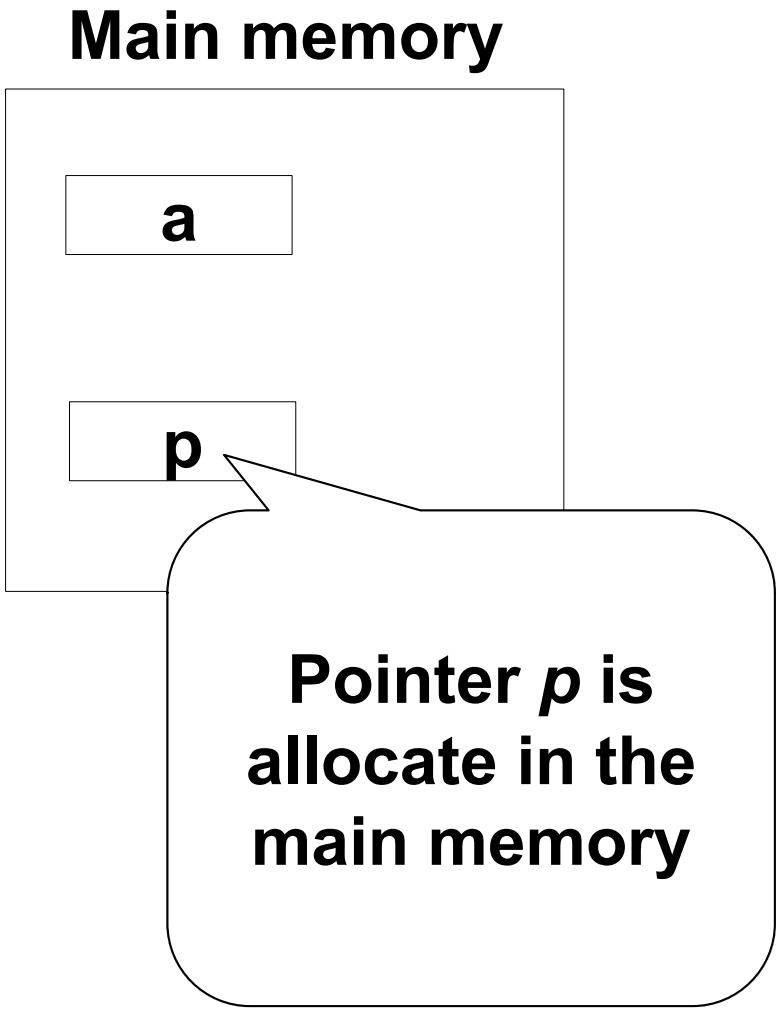
Main memory



Indirect Variable Reference (cnt'd)

- **<u>Indirect</u> reference to a variable:**

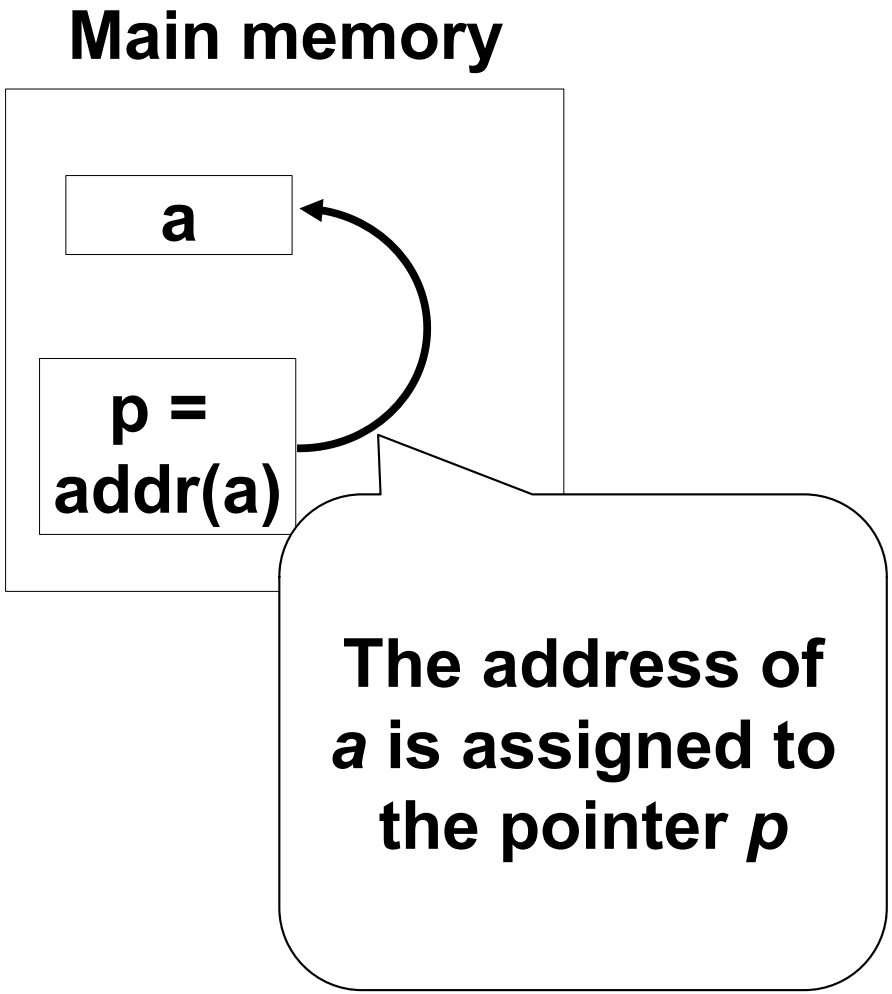
```
  int a;
▷ int *p;
  …
  p = &a;
  *p = 10;
```



Indirect Variable Reference (cnt'd)

- **<u>Indirect</u> reference to a variable:**

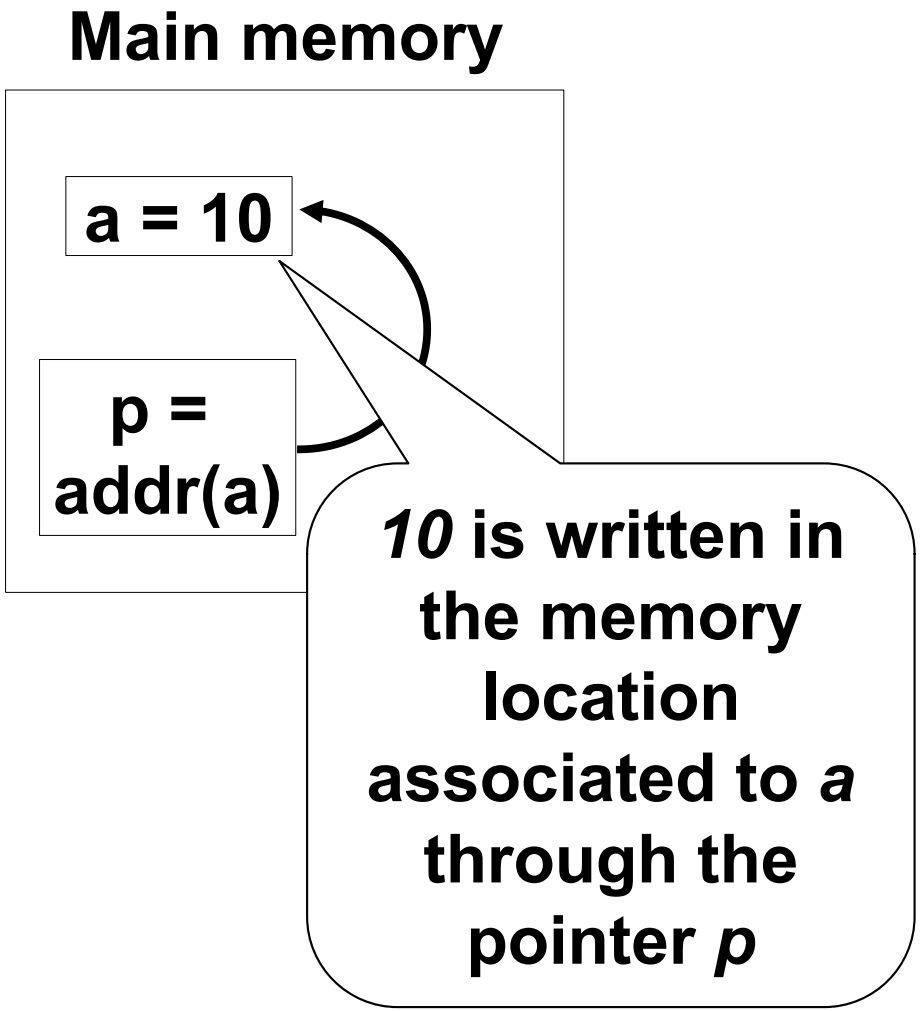
```
int a;
int *p;
…
▷ p = &a;
*p = 10;
```



Indirect Variable Reference (cnt'd)

- **<u>Indirect</u> reference to a variable:**

```
int a;
int *p;
…
p = &a;
▷ *p = 10;
```



Outline

- **Pointers:**
  - **Definition**
  - **Initialization**
  - **Operators**
  - **Variable Reference**
  - **Pointers Arithmetic's**

Pointers Arithmetic

- **The pointer arithmetic includes <u>Increment</u> and <u>Decrement</u> operations, only**

Using Pointers to scan Arrays

- **The pointer arithmetic includes <u>Increment</u> and <u>Decrement</u> operations, only**
- **Examples:**
  - `p = p + 5;`
  - `p = p - 10;`
  - `p++;`

If `p` is a pointer to an integer, after this instruction `p` will point to <u>5 "integers"</u> after `p`

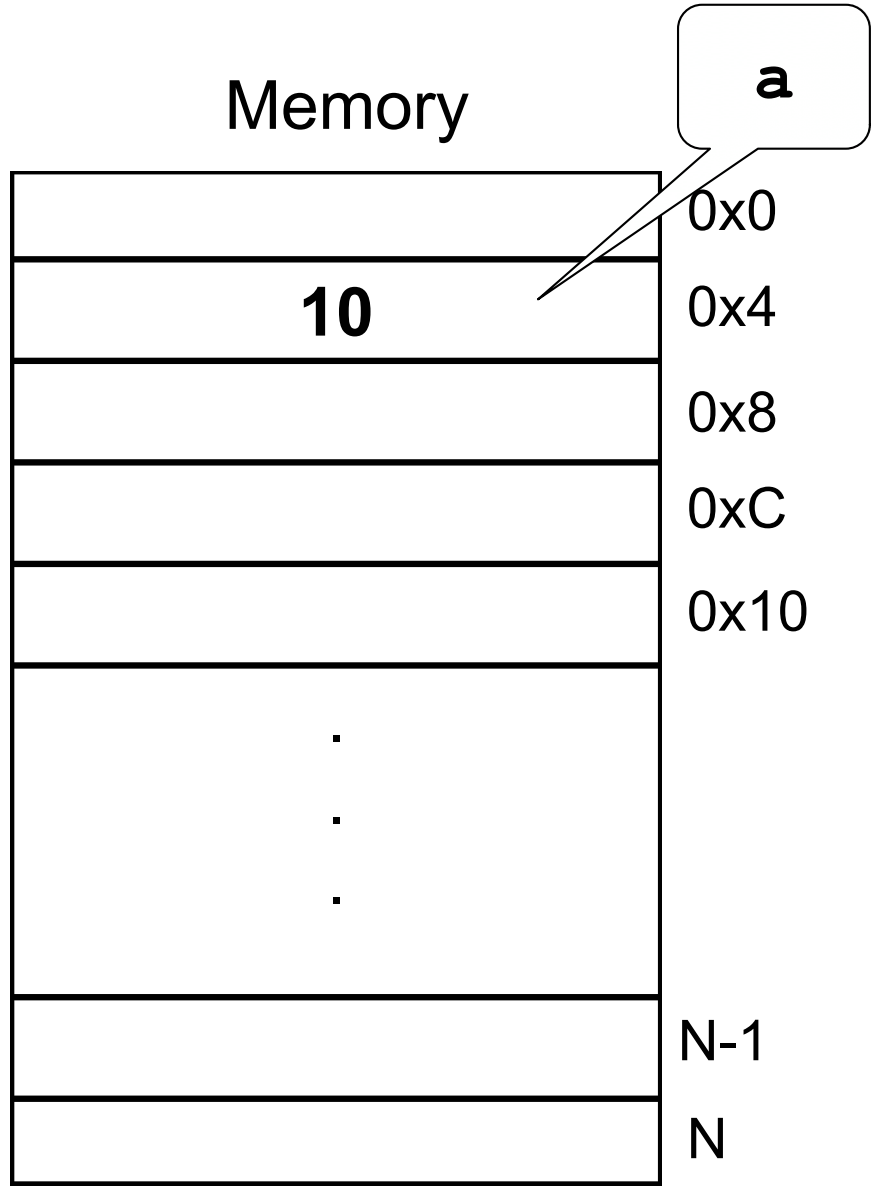
Using Pointers to scan Arrays

- **The pointer arithmetic includes <u>Increment</u> and <u>Decrement</u> operations, only**
- **Examples:**
  - `p = p + 5;`
  - `p = p - 10;`
  - `p++;`

The <u>ACTUAL</u> increment depends on the <u>SIZE</u> of the pointed data

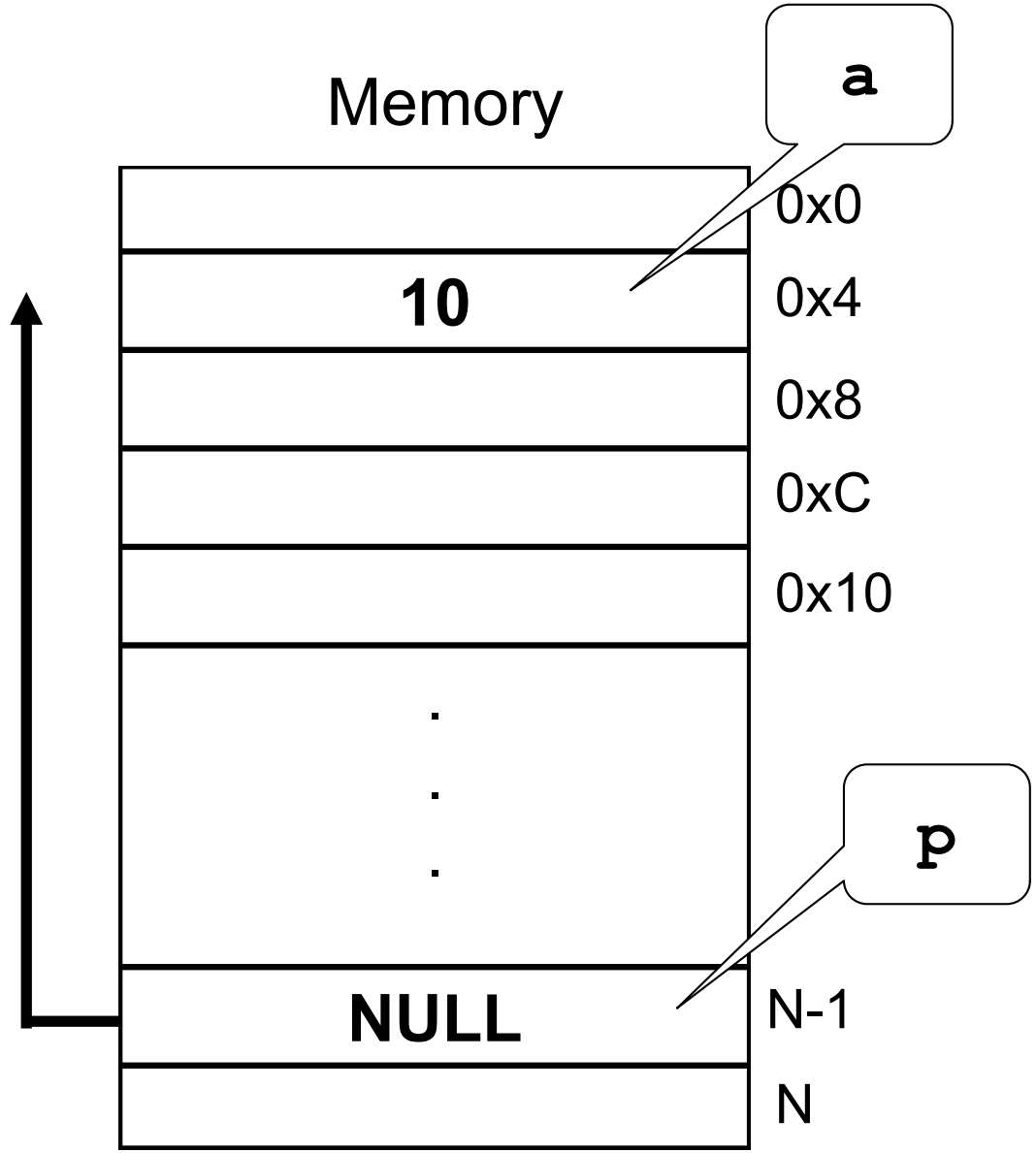
Pointers Arithmetic's – Examples

```
▷ int a = 10;
  int *p = NULL;

  p = &a;
  p = p + 3;
  p = p - 2;
  p++;
  p--;
```



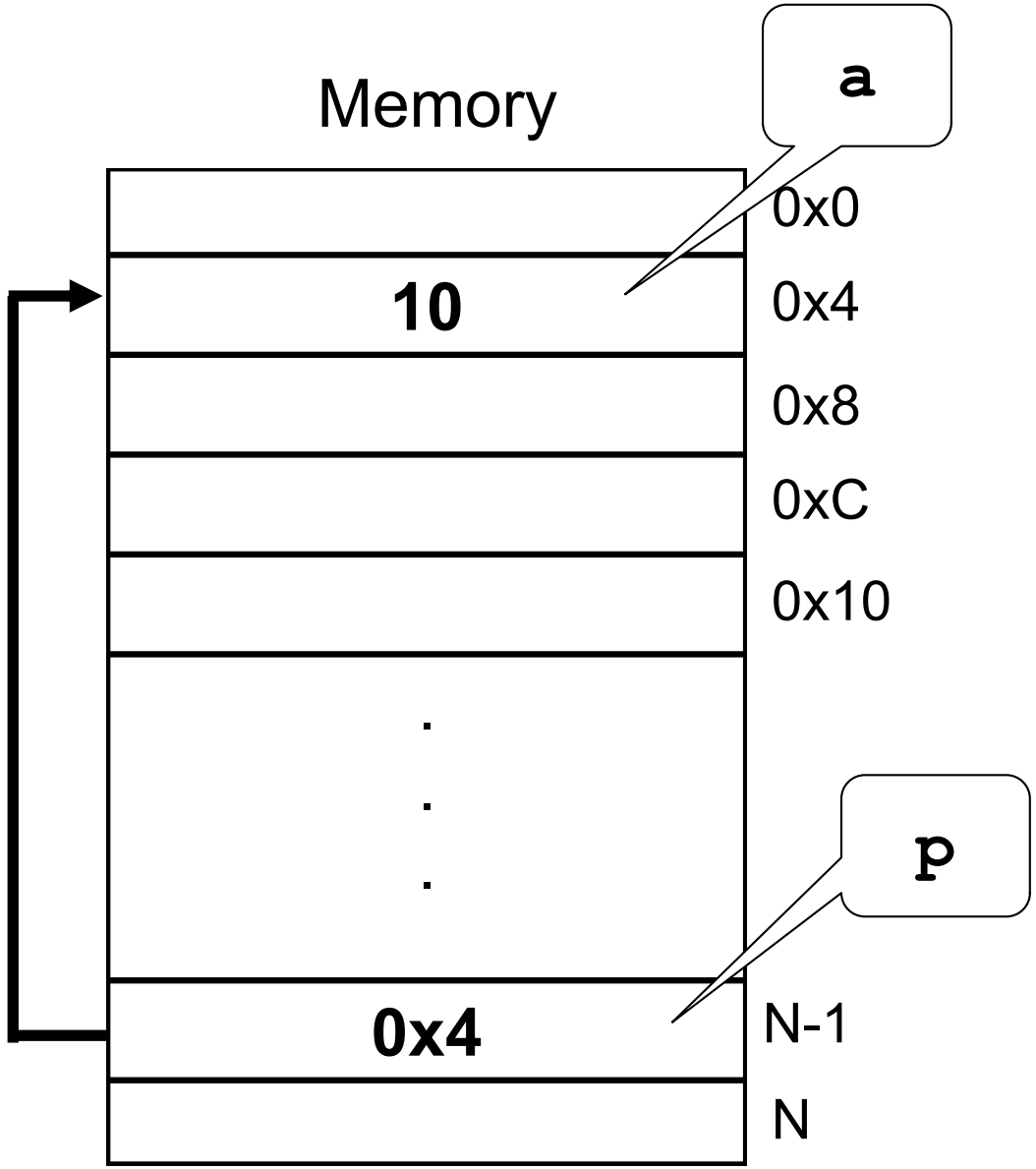
Pointers Arithmetic's – Examples

```
int a = 10;
int *p = NULL;

p = &a;
p = p + 3;
p = p - 2;
p++;
p--;
```



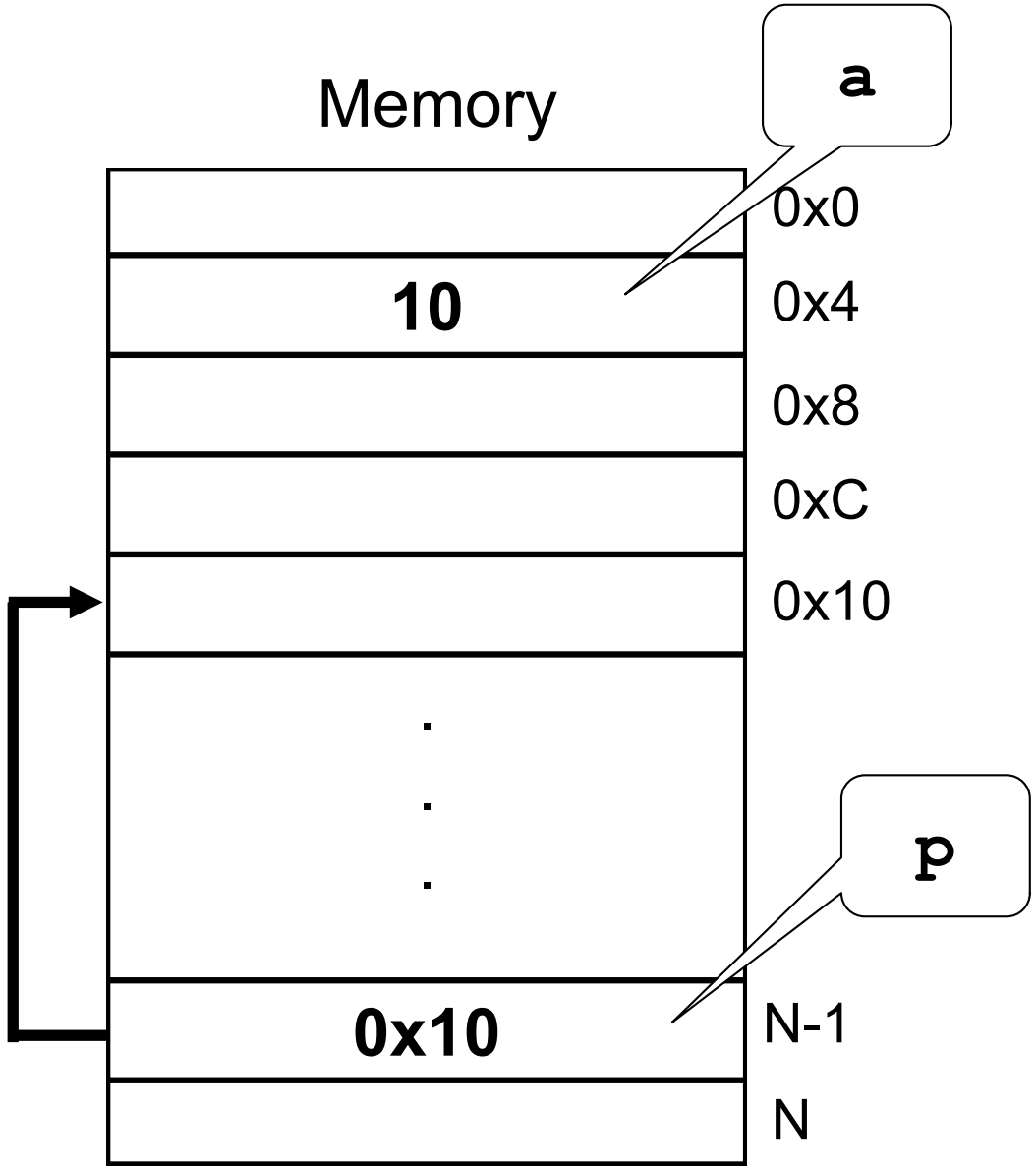
Pointers Arithmetic's – Examples

```
int a = 10;
int *p = NULL;

▷ p = &a;
p = p + 3;
p = p - 2;
p++;
p--;
```



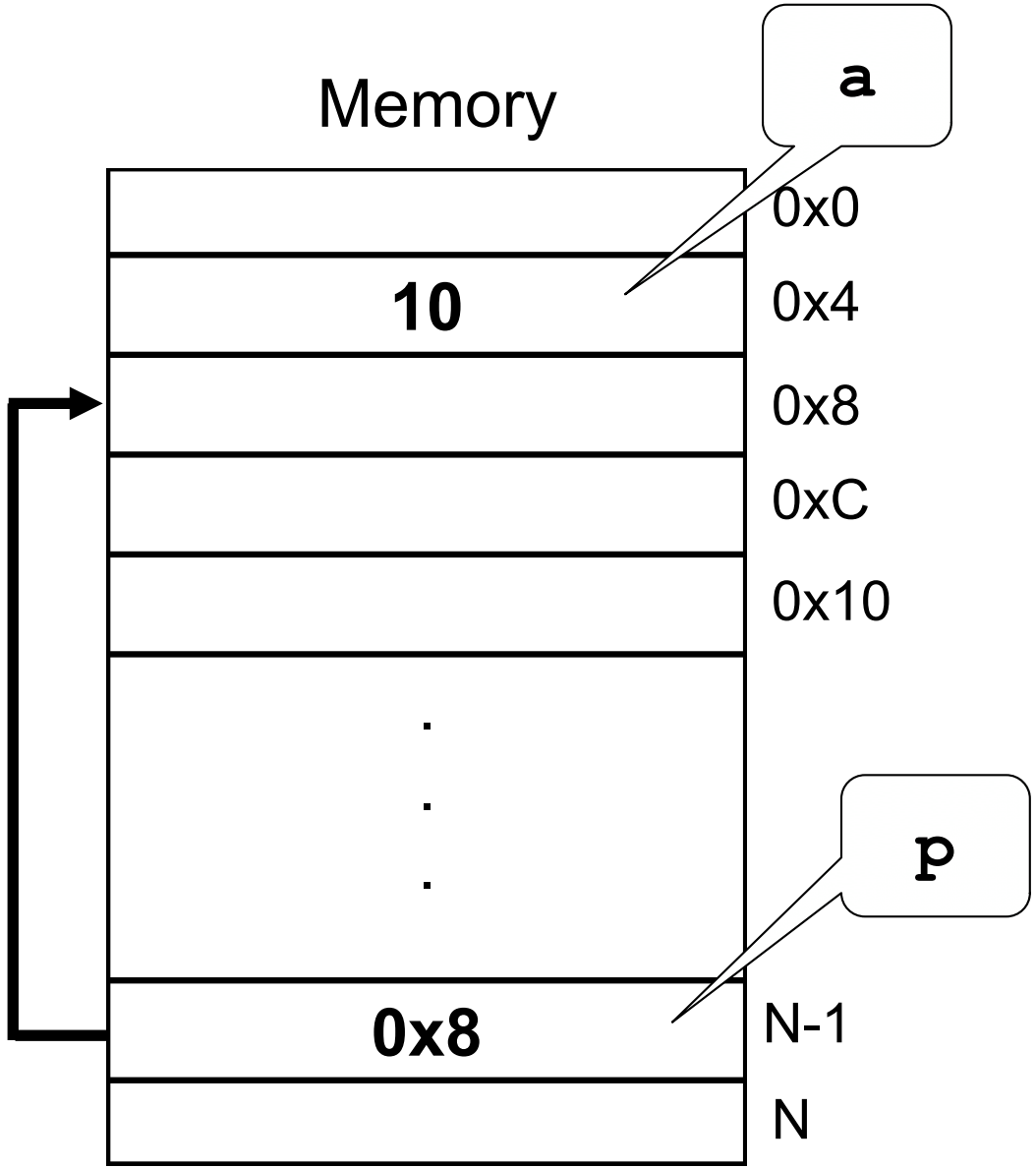
Pointers Arithmetic's – Examples

```
int a = 10;
int *p = NULL;

p = &a;
▷ p = p + 3;
p = p - 2;
p++;
p--;
```



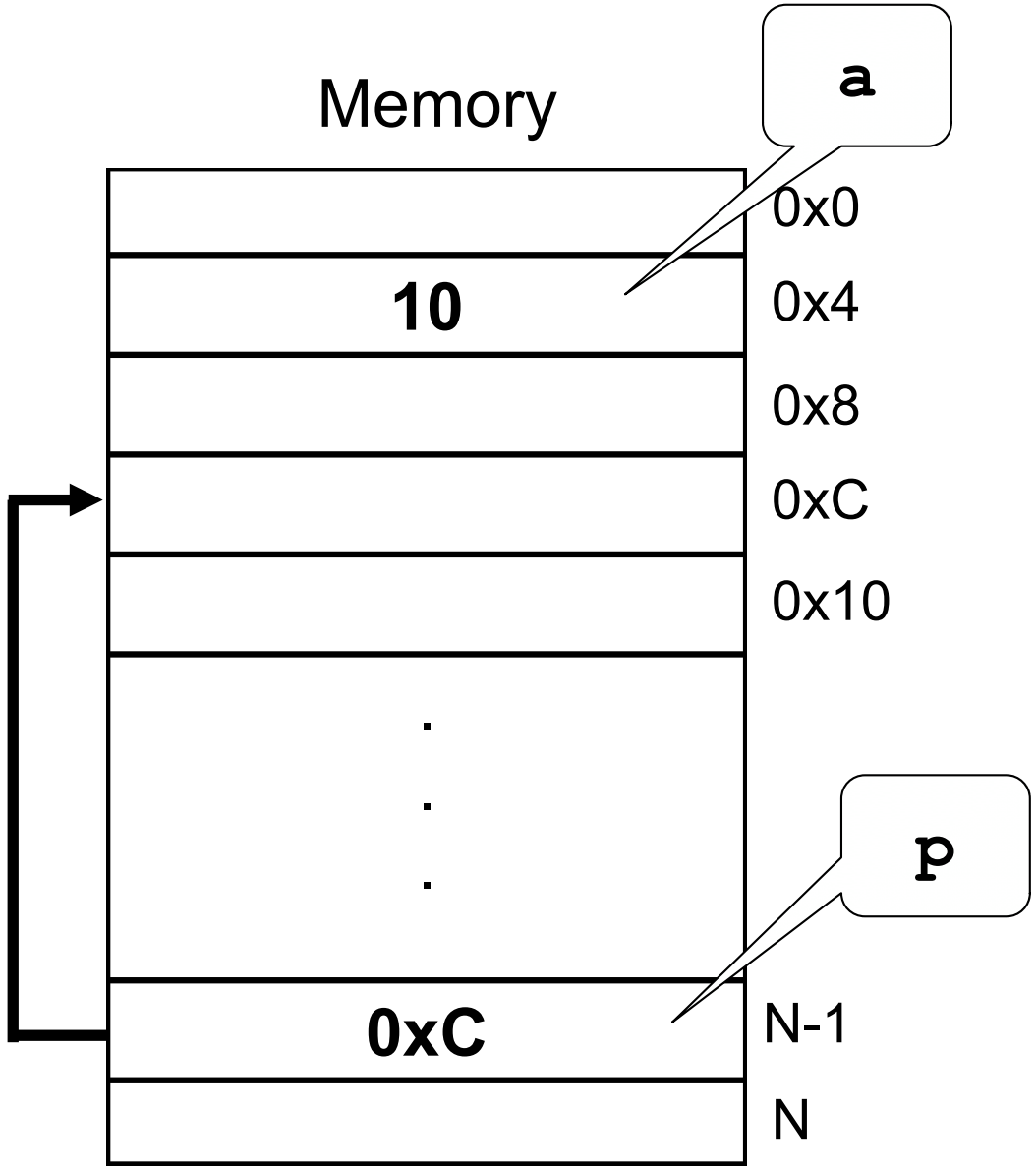
Pointers Arithmetic's – Examples

```
int a = 10;
int *p = NULL;

p = &a;
p = p + 3;
▷ p = p - 2;
p++;
p--;
```



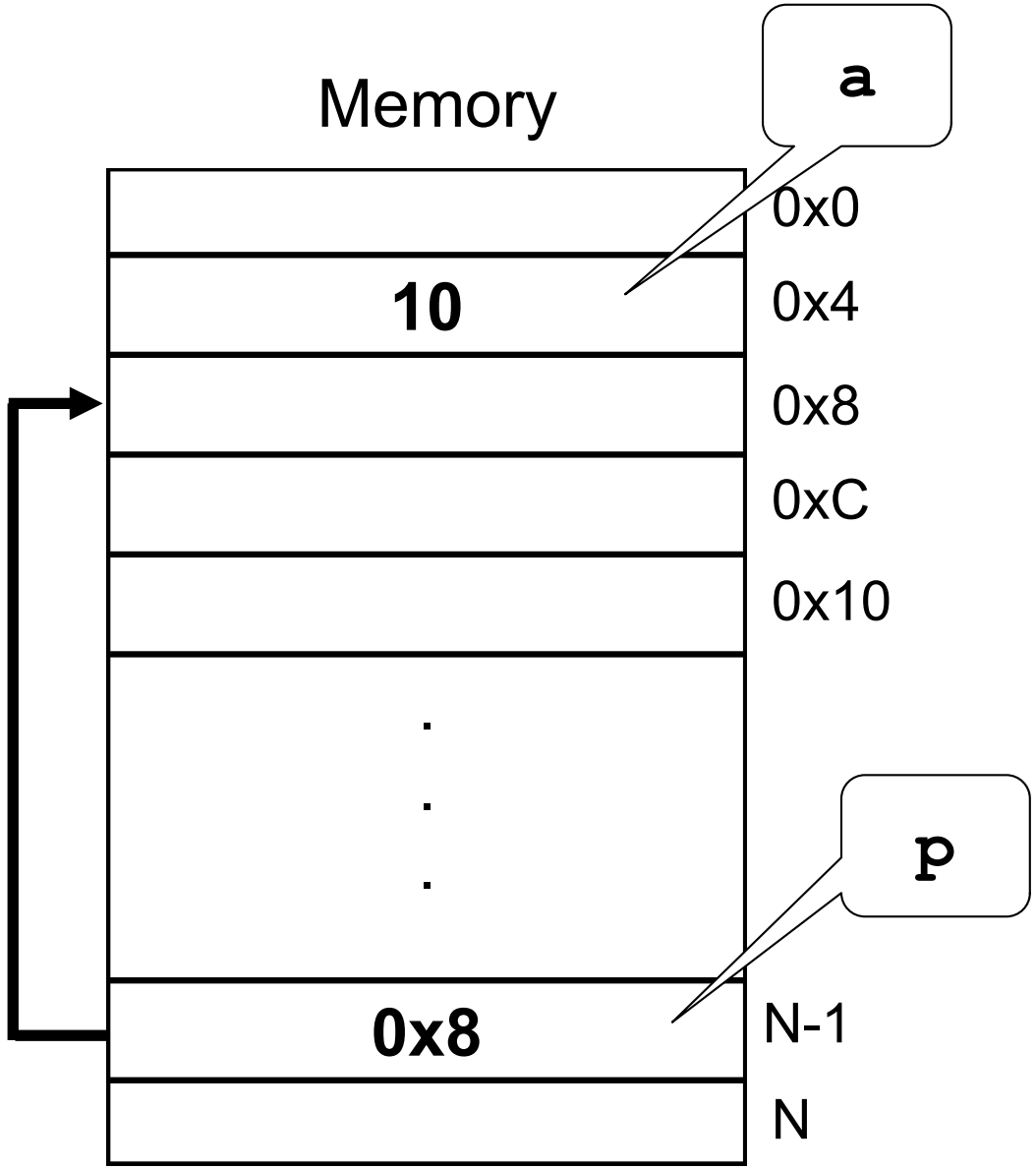
Pointers Arithmetic's – Examples

```
int a = 10;
int *p = NULL;

p = &a;
p = p + 3;
p = p - 2;
▷ p++;
p--;
```



Pointers Arithmetic's – Examples

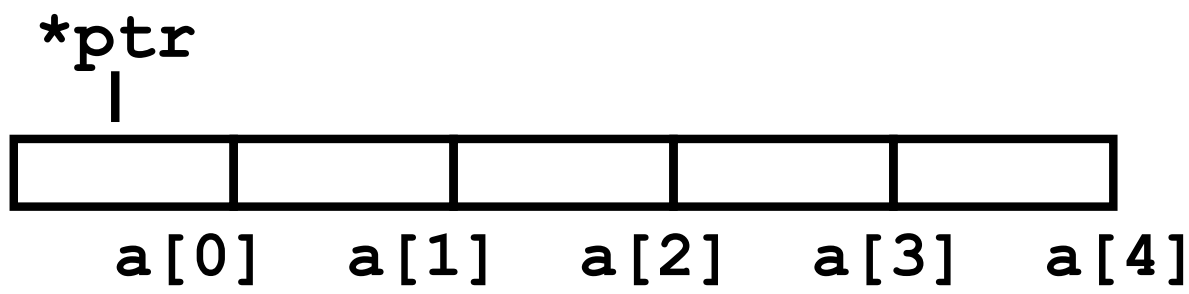
```
int a = 10;
int *p = NULL;

p = &a;
p = p + 3;
p = p - 2;
p++;
▷ p--;
```



Pointers Arithmetic's – vectors

- **Integer math operations can be used with pointers.**
- **If you increment a pointer, it will be increased by the size of whatever it points to.**

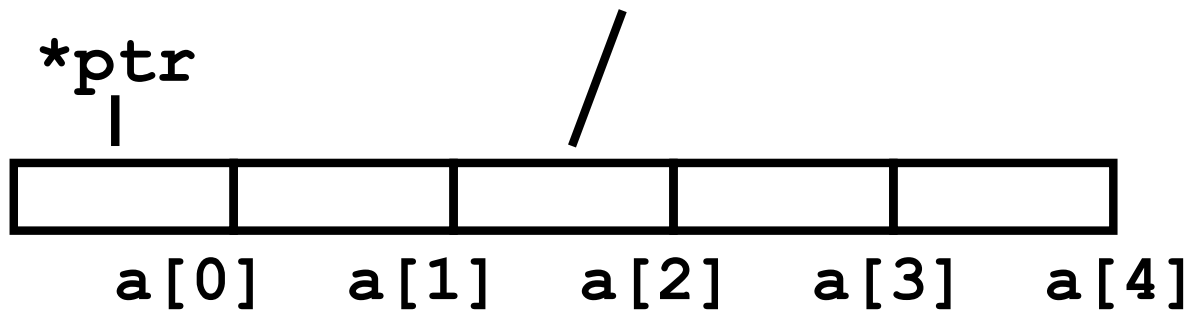
```
int a[5];
int *ptr = a;
```



Pointers Arithmetic's – vectors

- **Integer math operations can be used with pointers.**
- **If you increment a pointer, it will be increased by the size of whatever it points to.**

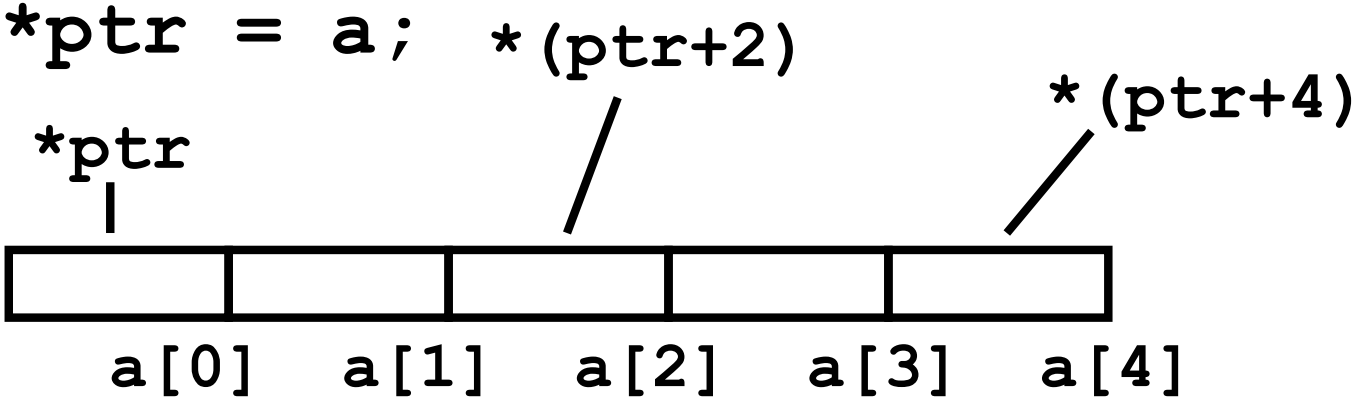
```
int a[5];
int *ptr = a;  *(ptr+2)
```



Pointers Arithmetic's – vectors

- **Integer math operations can be used with pointers.**
- **If you increment a pointer, it will be increased by the size of whatever it points to.**

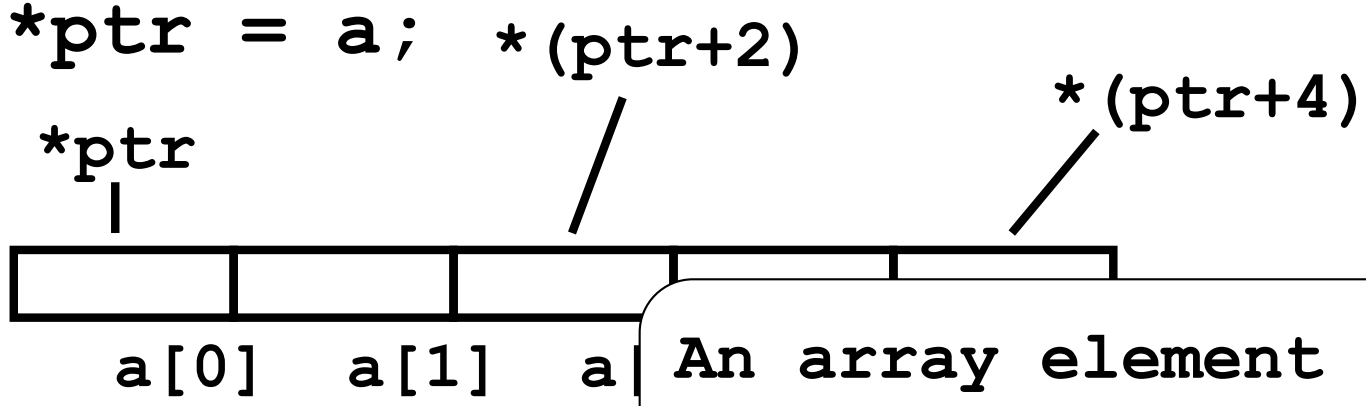
```
int a[5];
int *ptr = a;
```



Pointers Arithmetic's – vectors

- **Integer math operations can be used with pointers.**
- **If you increment a pointer, it will be increased by the size of whatever it points to.**

```
int a[5];
int *ptr = a;
```



An array element can be accessed both by a[i] and *(ptr+i)


Малые Автюхи, Калинковичский район, Республики Беларусь