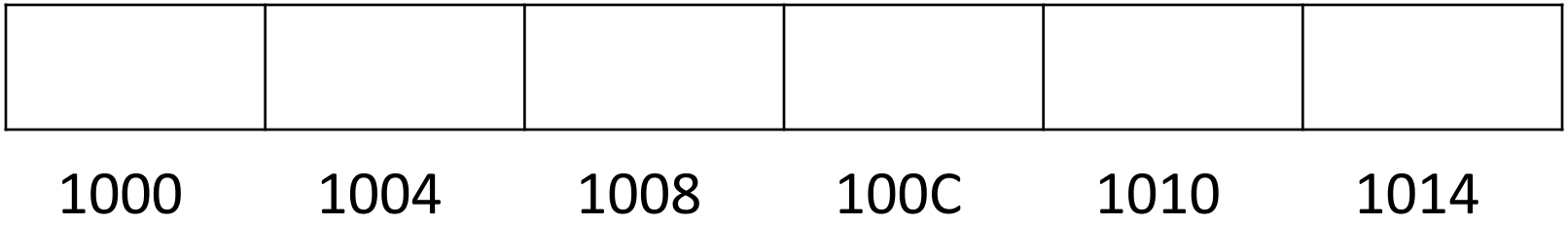
Pointer

Memory Addresses

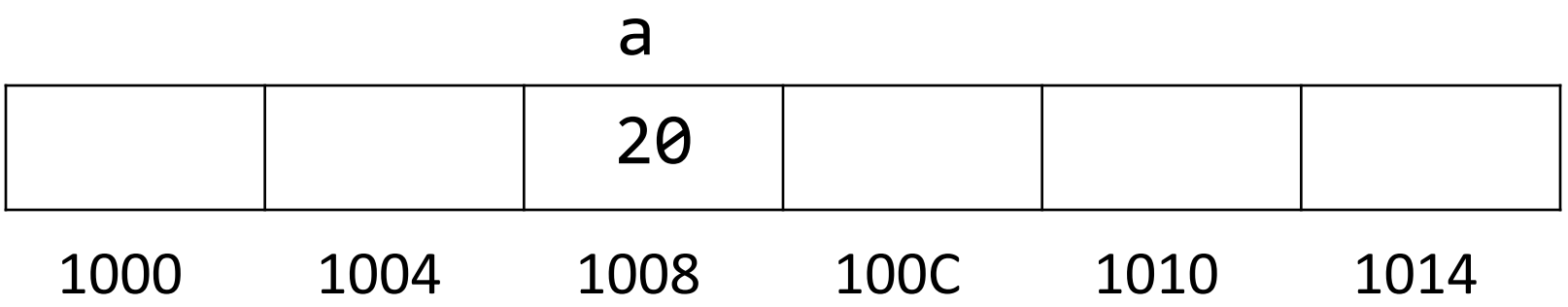
- Particular location in memory are identified by their address
- This addresses are 32 bit or 64 bit depending on computer architecture



Memory Address of variables

- The address of a variable can be determined by applying the address operator (&)

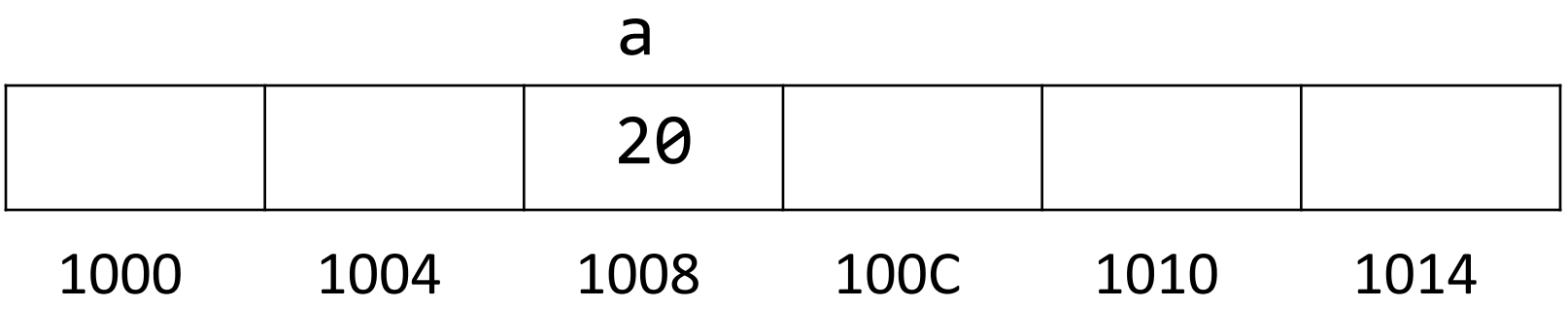
```
int a = 20;
```



Memory Address of variables

- The address of a variable can be determined by applying the address operator (&)
- Addresses can be printed using the %p format specifier

```
int a = 20;
Memory address of variable a = &a = 1008
```



Pointer

Variables that stores memory addresses.

Declaring Pointer

```
int *p;
```

This is a pointer variable p which should contain the memory address of an integer variable

Assigning Pointer

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

Assigns memory address of variable a (which is an int) to pointer variable p

Initializing Pointer

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

NULL Pointer

Pointers should always be initialized when they are declared
If there is no value to use to initialize then use NULL to initialize the pointer.

```
int *p = NULL;
```

Indirection (*) Operator

* - in a variable declaration statement indicates that a particular variable is a pointer type.

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

- in executable statements it refers to the data pointed to by the pointer

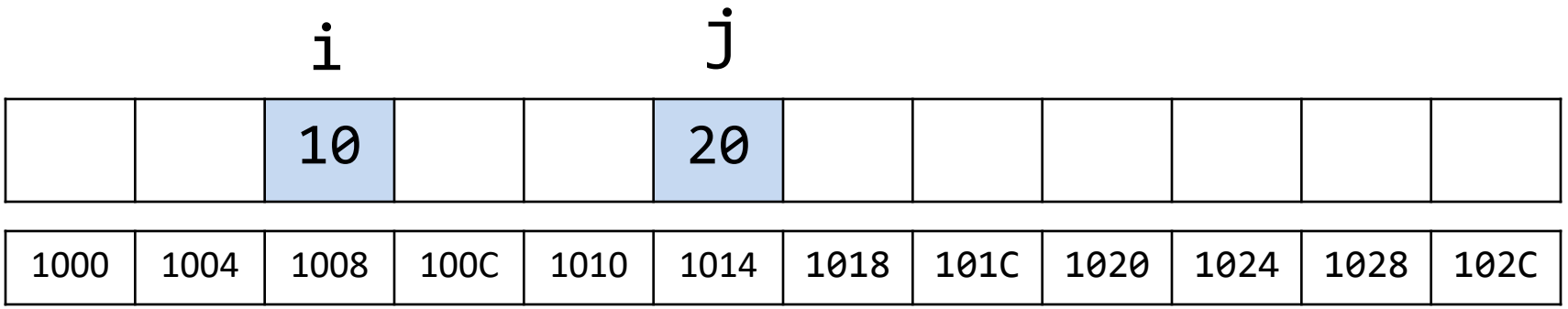
```
*p = 30;
```

Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
p2 = NULL;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```

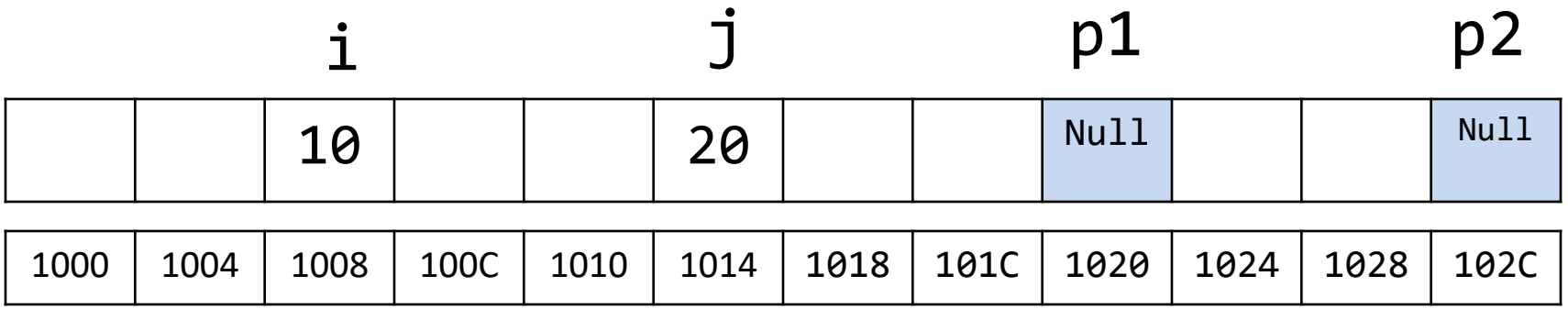
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



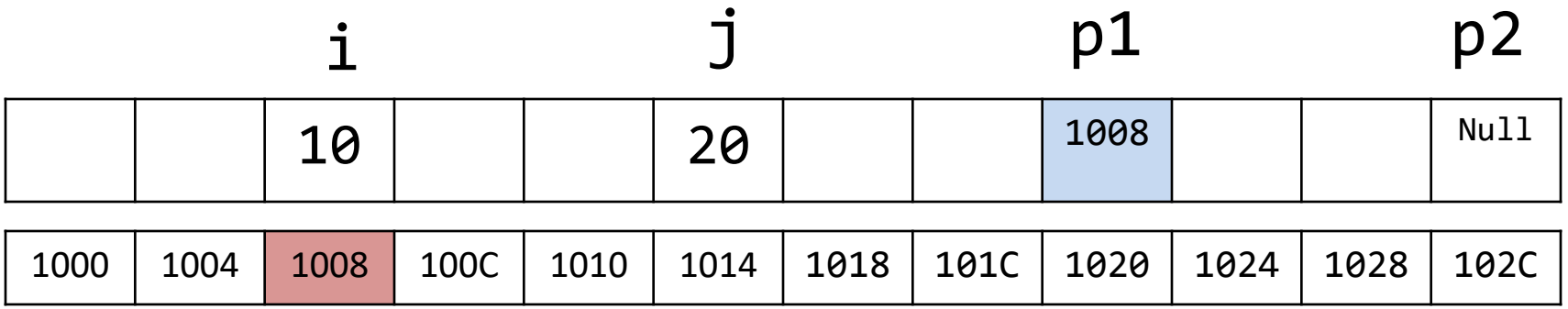
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



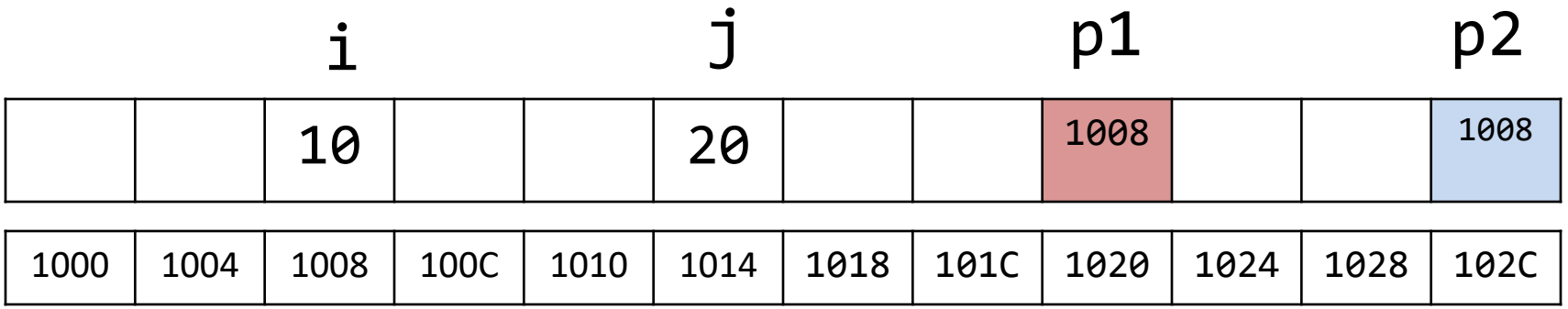
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



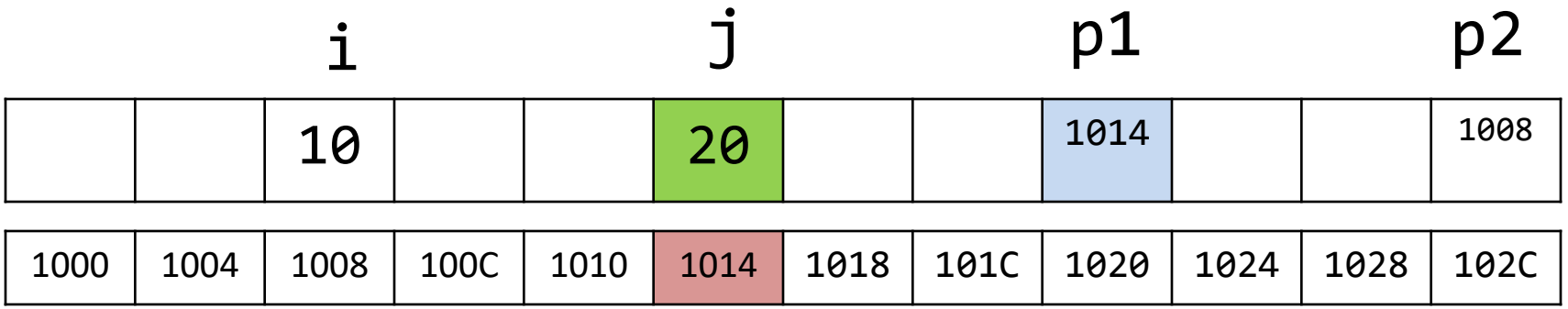
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



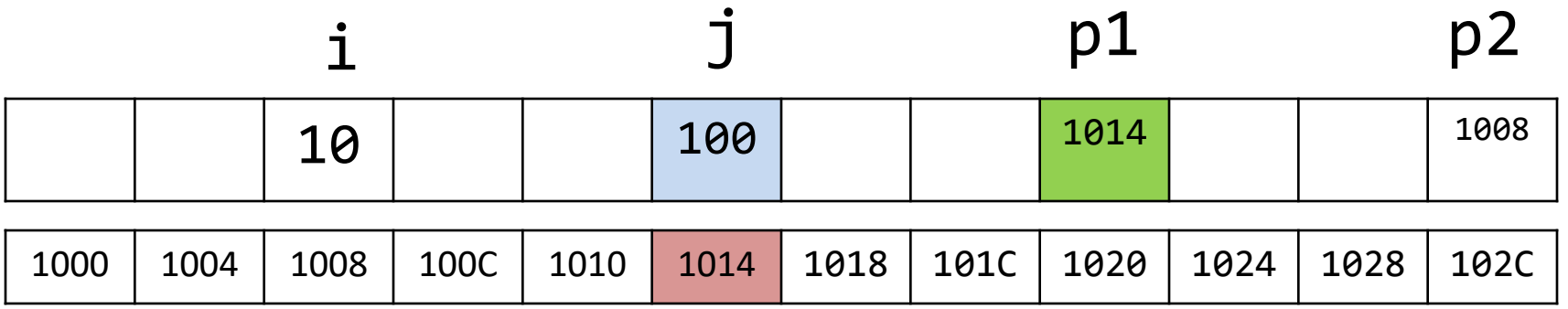
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



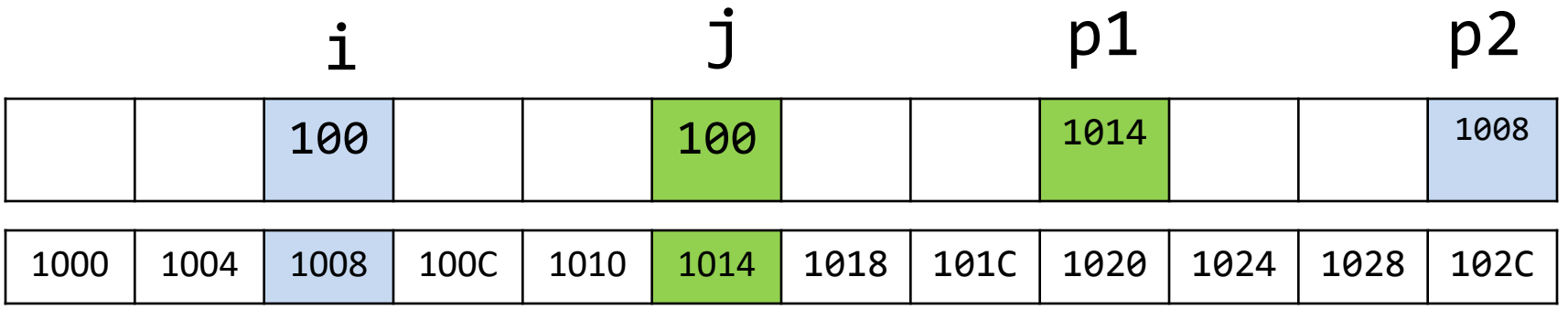
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



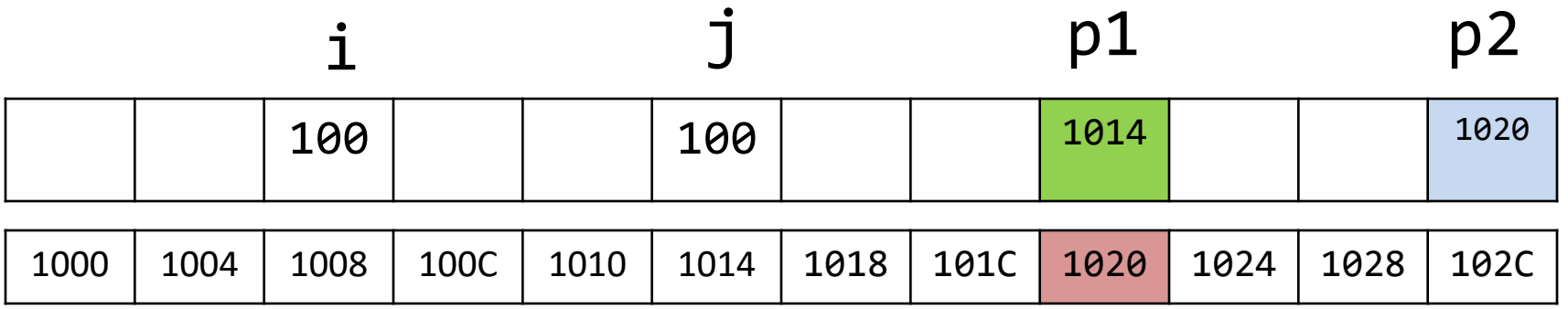
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



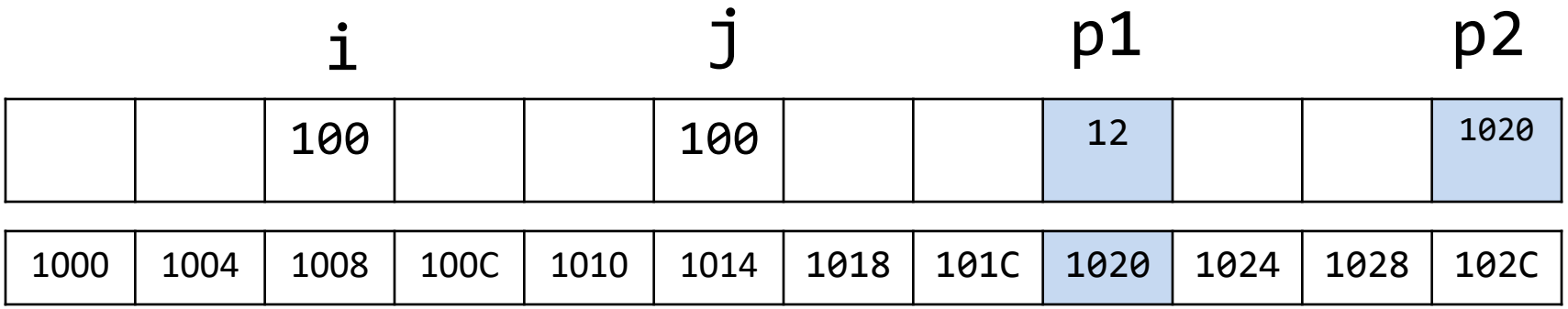
Pointer

```
int i = 10, j = 20;
int *p1 = NULL, *p2 = NULL;
p1 = &i;
p2 = p1;
p1 = &j;
*p1 = 100;
*p2 = *p1;
p2 = &p1;
```



Pointer

```
*p2 = 12;
```



Pointer Arithmetic

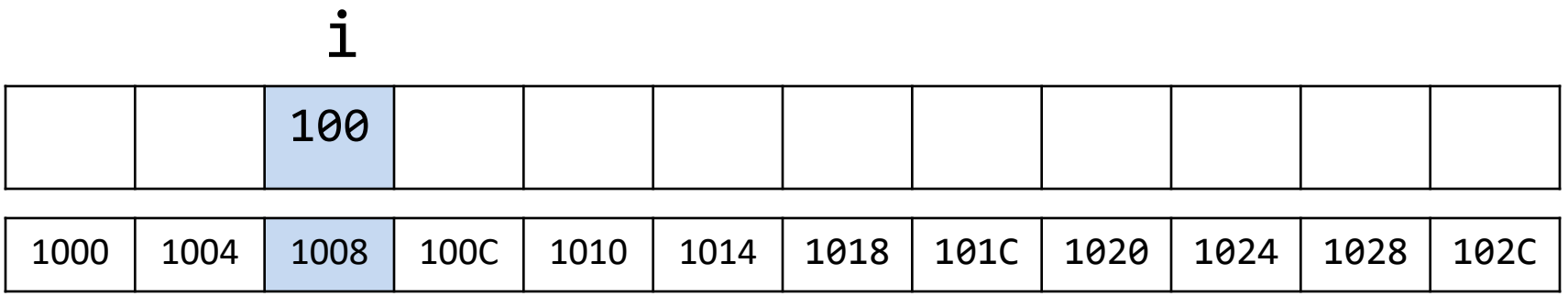
- integers may be added to or subtracted from addresses.

```
int *p = &a;
p = p + 2;
p = p – 2;
```

- Arithmetic operations on addresses actually occur in steps of the size of the thing pointed to by the address.

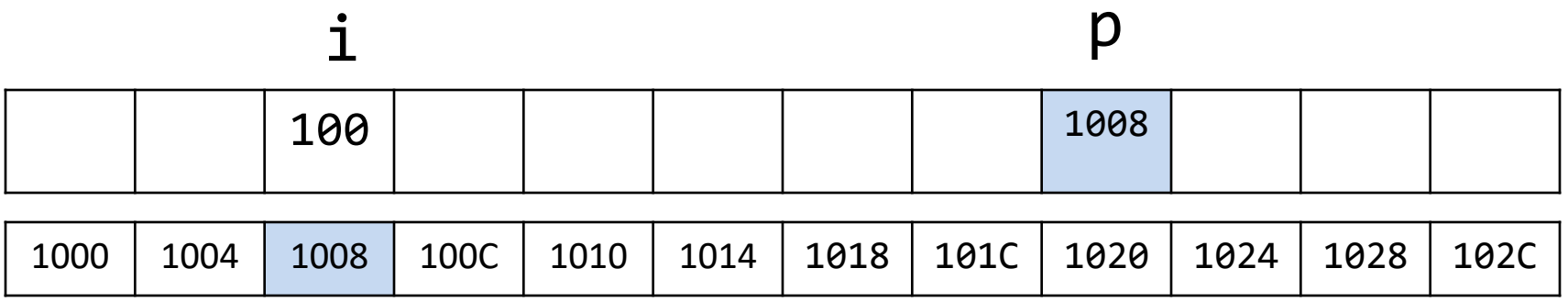
Pointer Arithmetic(1)

```
int i = 100;
int *p = &i;
p = p + 4;
*p = 15
```



Pointer Arithmetic(1)

```
int i = 100;
int *p = &i;
p = p + 4;
*p = 15
```

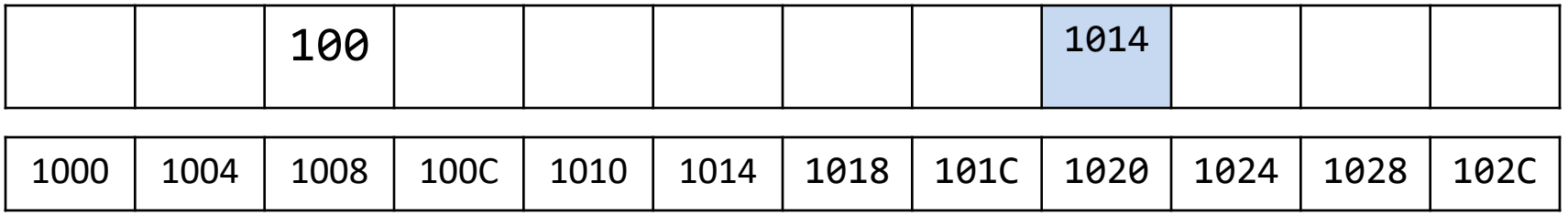


Pointer Arithmetic(1)

```
int i = 100;
int *p = &i;
p = p + 4;
*p = 15
```

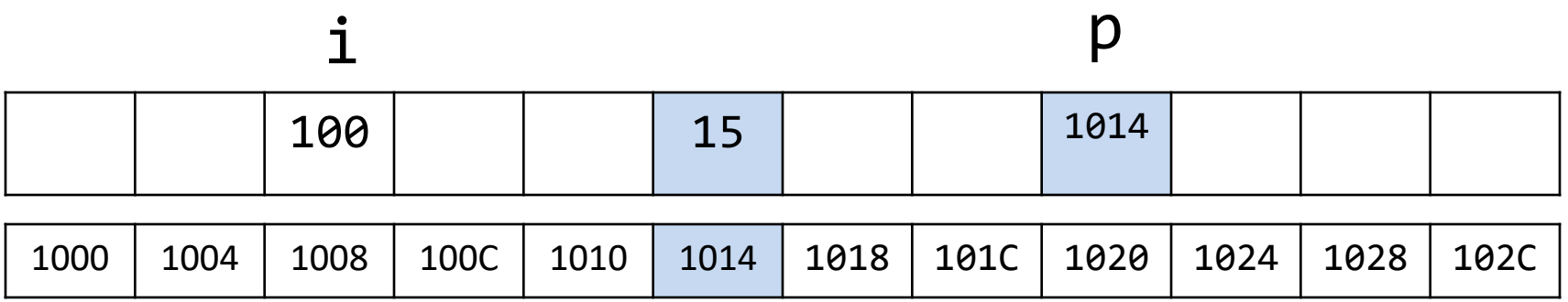
1008 + 4

i p



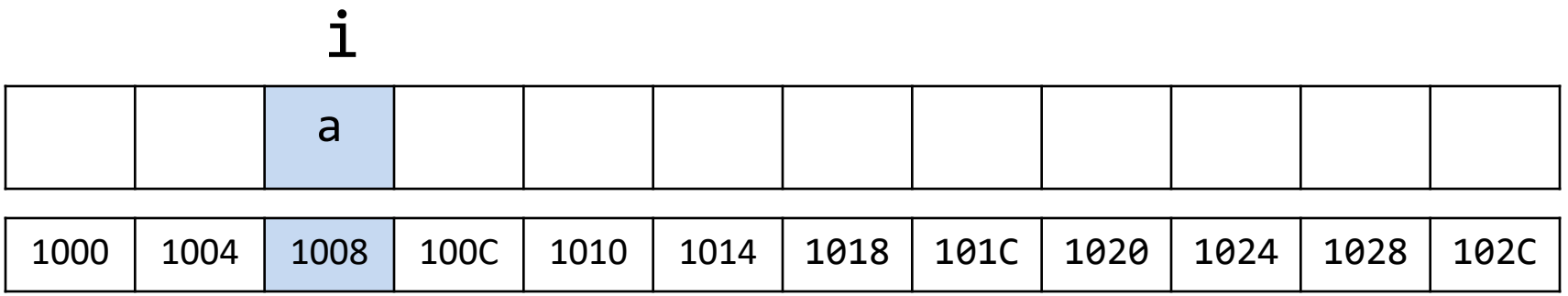
Pointer Arithmetic(1)

```
int i = 100;
int *p = &i;
p = p + 4;
*p = 15
```



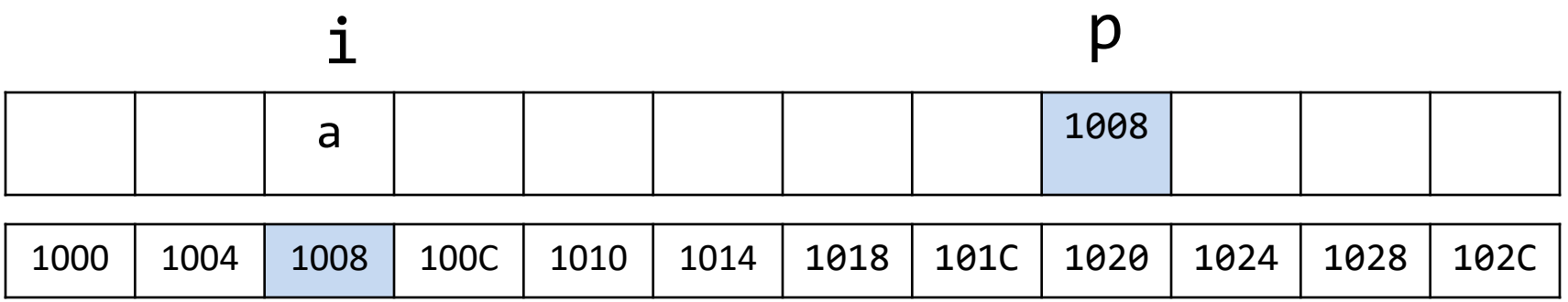
Pointer Arithmetic(2)

```
char i = ‘a’;
char *p = &i;
p = p + 4;
*p = ‘c’
```



Pointer Arithmetic(2)

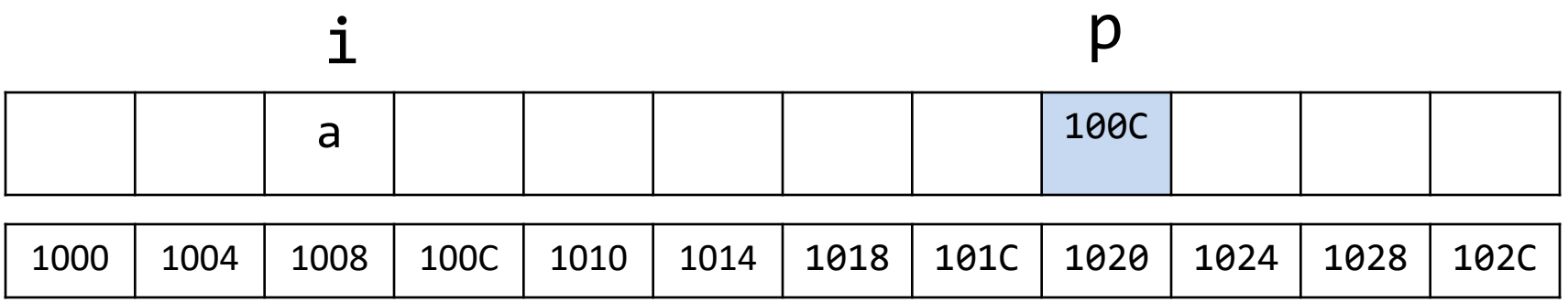
```
char i = ‘a’;
char *p = &i;
p = p + 4;
*p = ‘c’
```



Pointer Arithmetic(2)

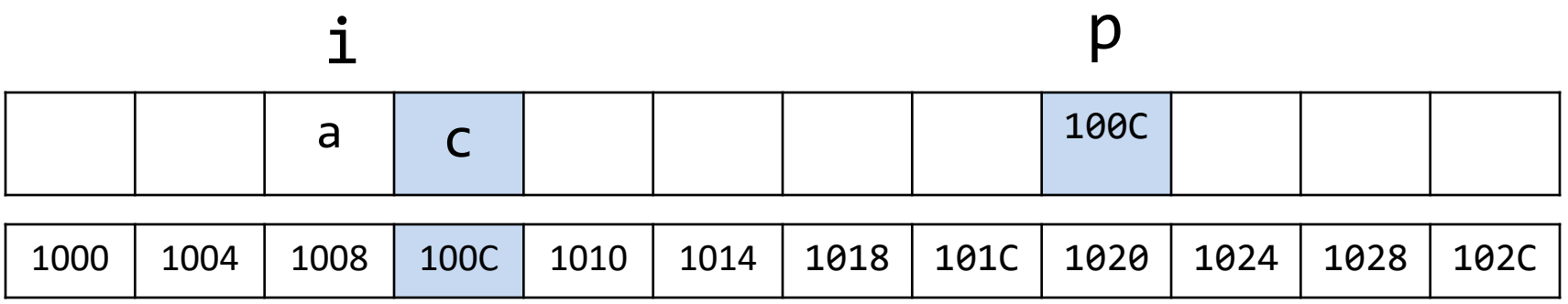
```
char i = ‘a’;
char *p = &i;
p = p + 4;
*p = ‘c’
```

1008 + 4



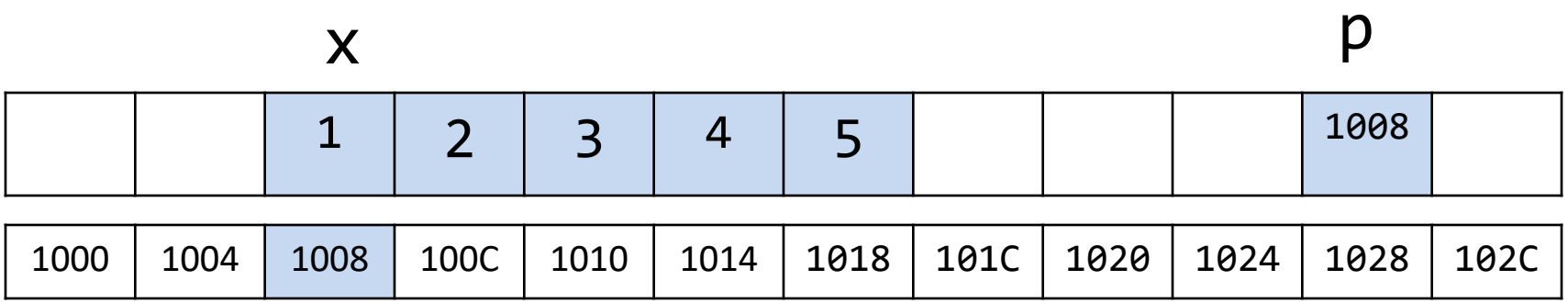
Pointer Arithmetic(2)

```
char i = ‘a’;
char *p = &i;
p = p + 2;
*p = ‘c’
```

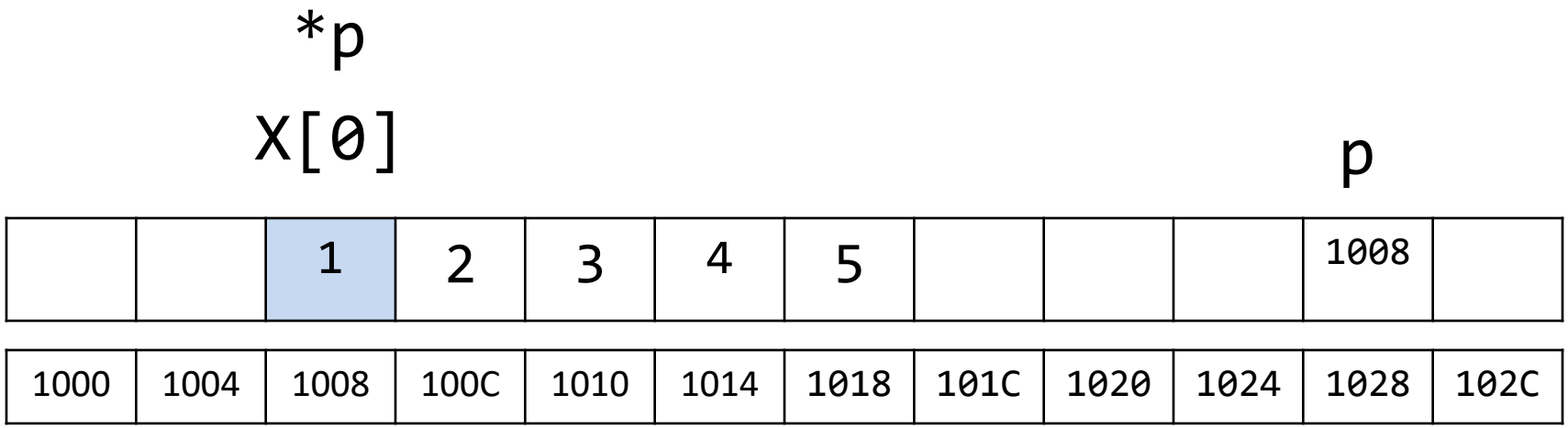


Pointers & Arrays

```
int x[5] = {1, 2, 3, 4, 5},
int *p = x;
```



Pointers & Arrays

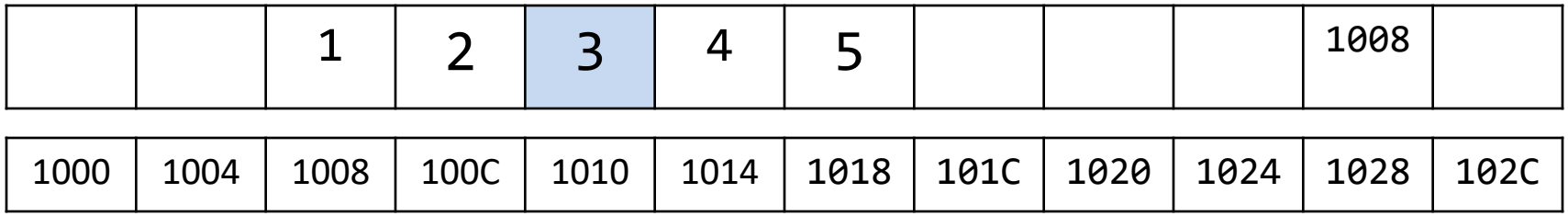


Pointers & Arrays

*(p + 2)

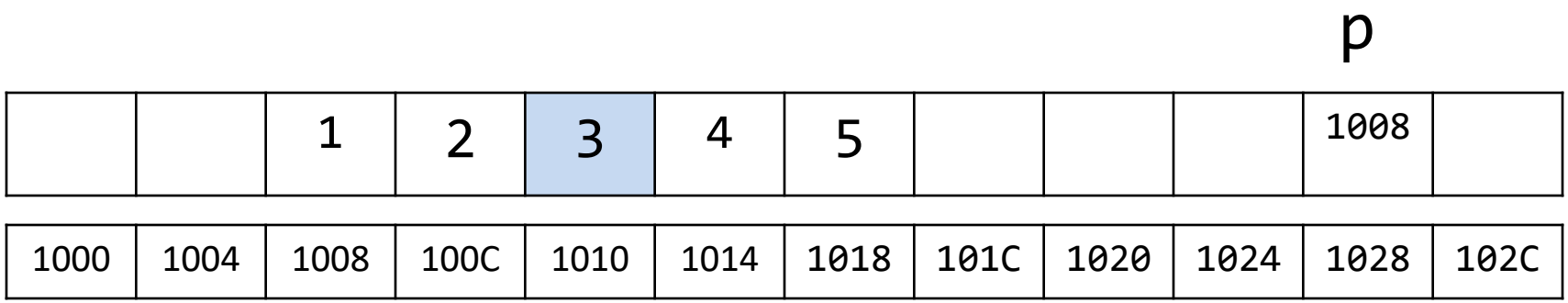
X[2]

p



Pointers & Arrays

```
X[i] = * (p + i)
```



Pointers & Arrays (1)

```
int x[5];
int *p = x;
```

```
scanf("%d", & x[i]);
```

```
scanf("%d", (p + i));
```

```
printf("%d ", x[i]);
```

```
printf("%d ", *(p + i));
```

Pointers & Arrays (2)

```
int x[5][5];
int *p = x;
```

```
scanf("%d", & x[i][j]);
```

```
scanf("%d", (p + i*C + j));
```

```
printf("%d ", x[i][j]);
```

```
printf("%d ", *(p + i*C + j));
```