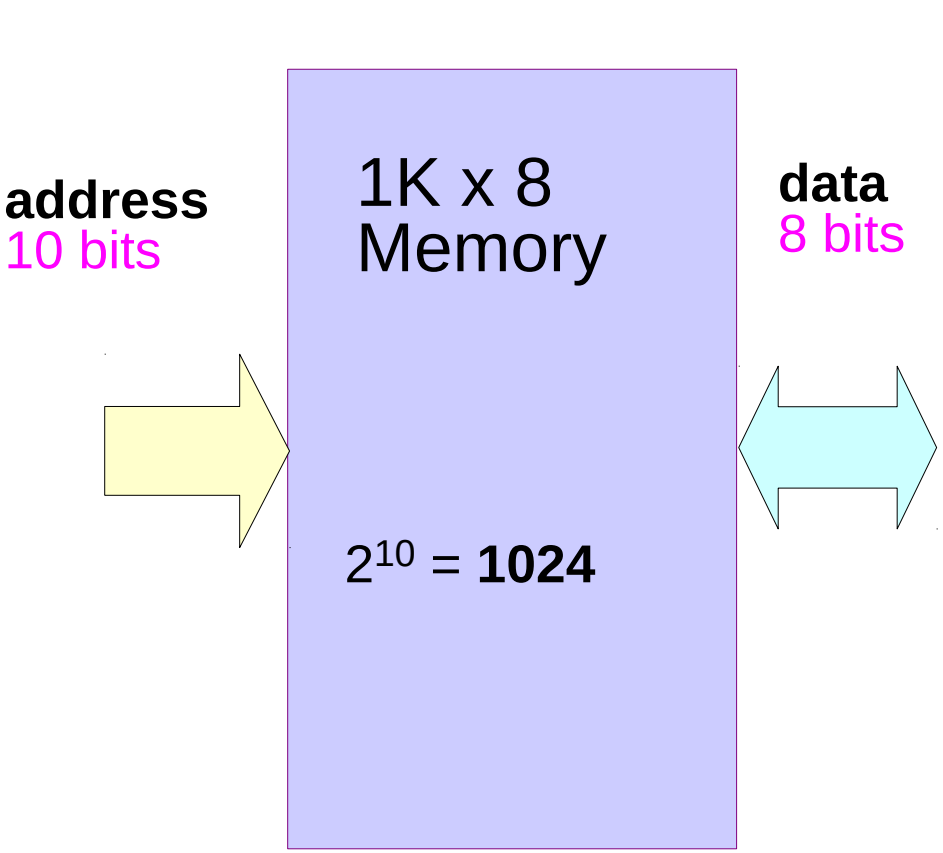
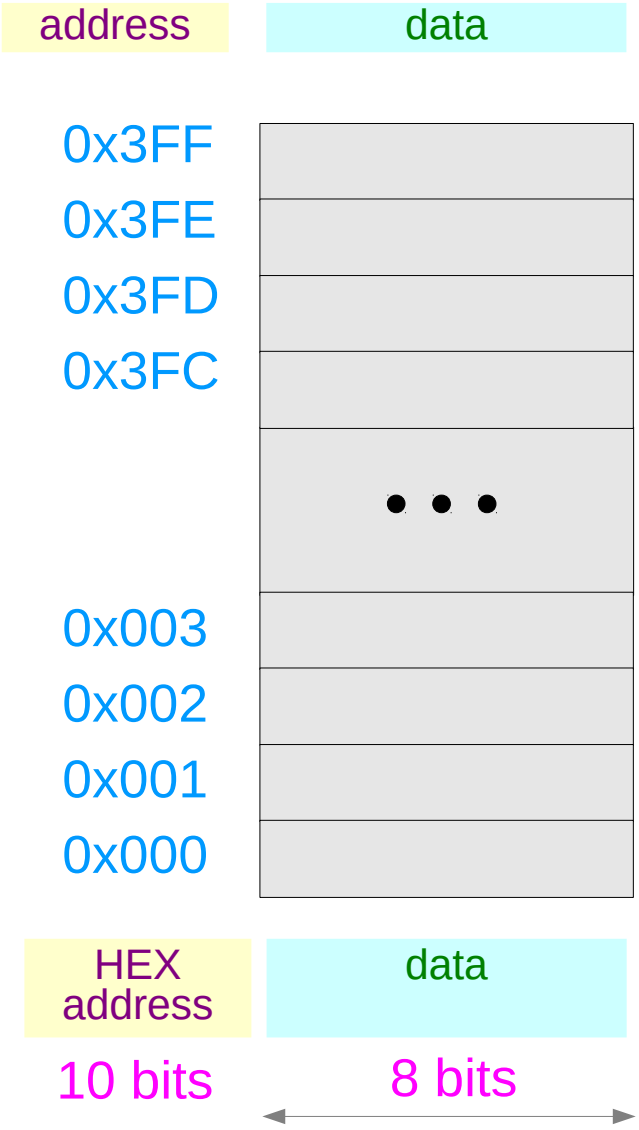
Background – Accessing Memory (1A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Byte Address and Data in a Memory

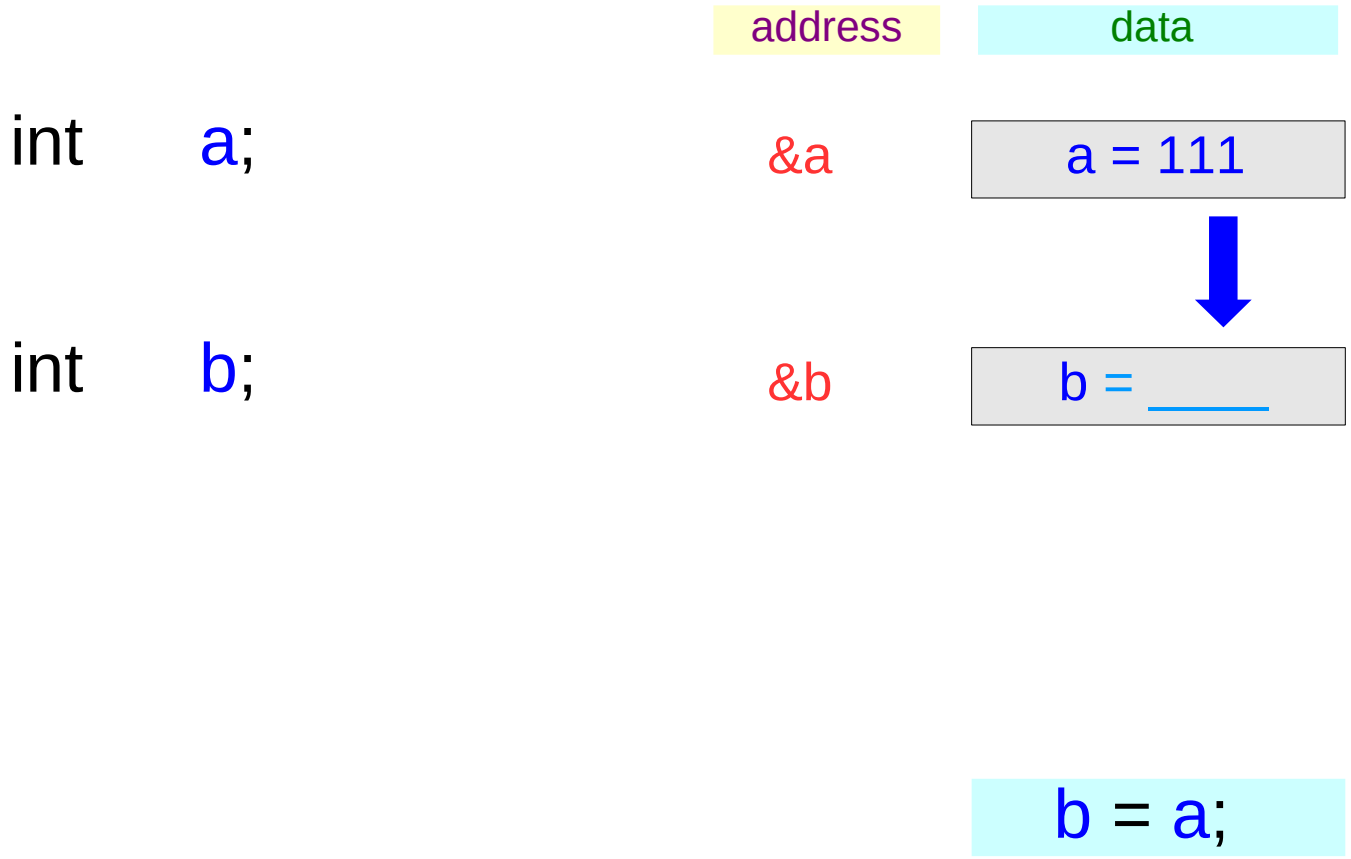




Variables and their addresses

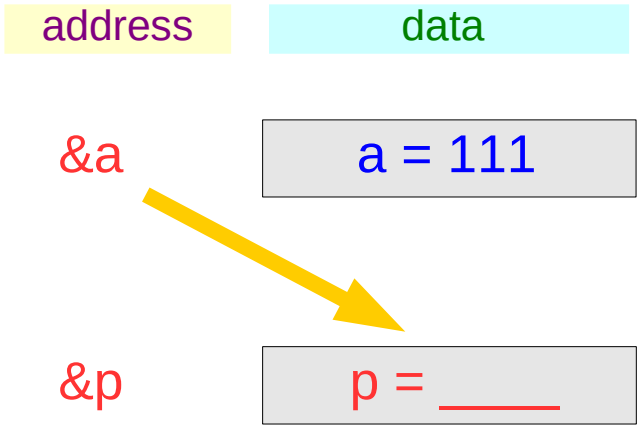
	address	data
int a;	&a	a
int * p;	&p	p

Assignment of a value



Assignment of an address

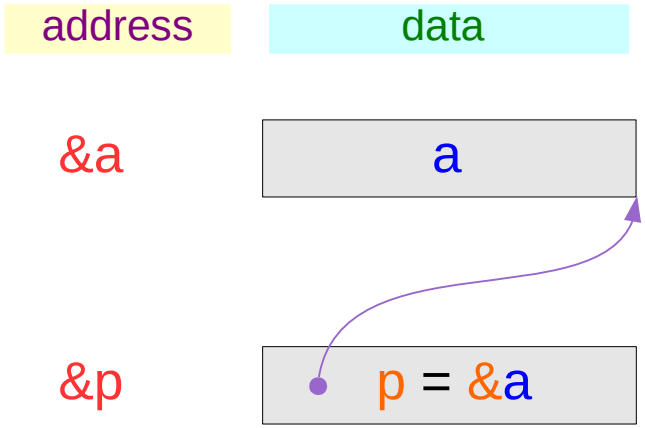
```
int     a;

int   * p;
```




```
p = &a;
```

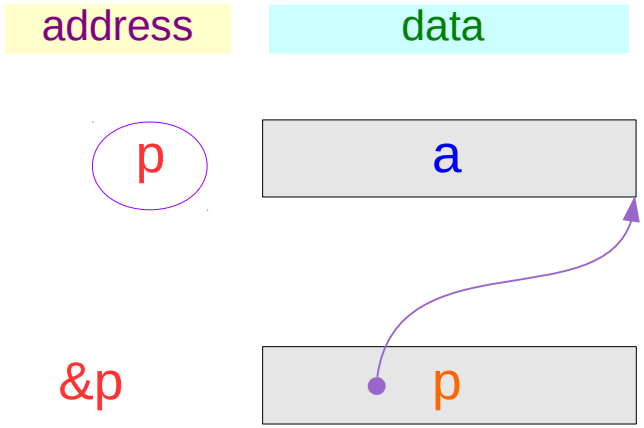
Variables with initializations

```
int     a;

int   * p = &a;
```



Pointed addresses : p

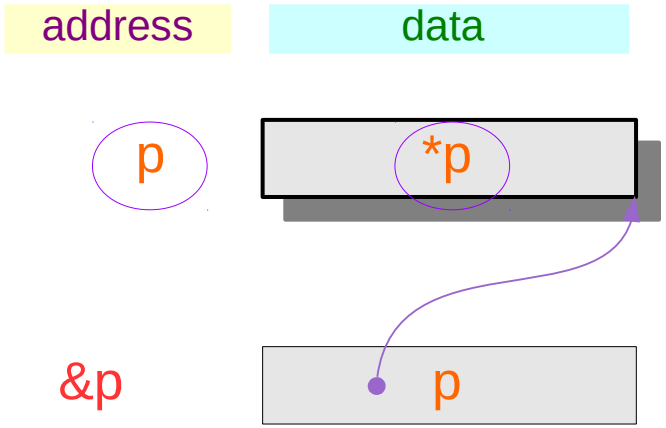
```
int     a;

int  *  p = &a;
```



p ≡ &a

Dereferenced Variable : *p

```
int     a;

int  *  p = &a;
```



assignment

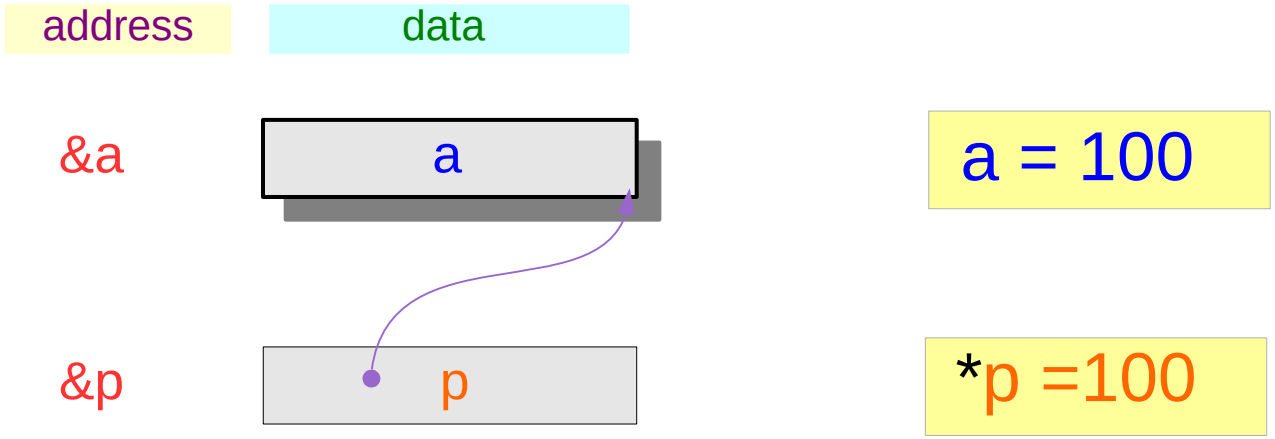
equivalence

p ≡ &a

*p ≡ *&a

*p ≡ a

Two way to access: a and *p



1) Read/Write a
2) Read/Write *p

Applications of pointers

1. Pass by Reference
2. Arrays

Pass by Reference

Variable Scopes

```
int func1 (int a, int b)
{
   int i, int j;
   ...
   ...

   ...
   ...
}
```

i and j's
variable scope

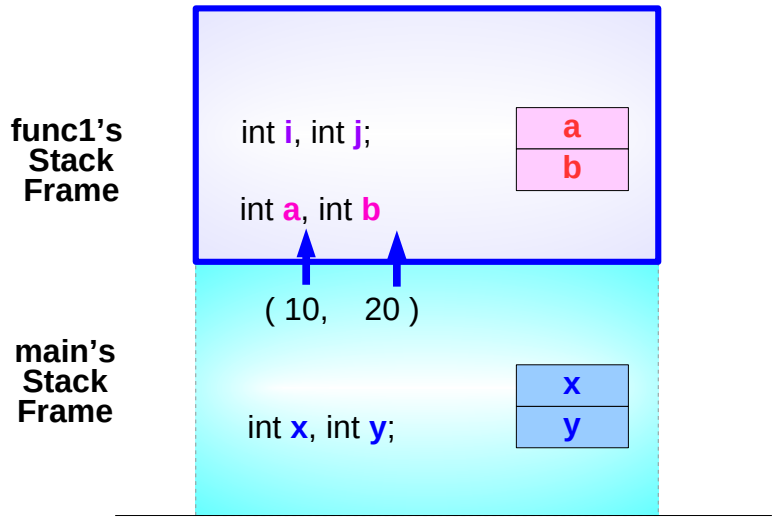
cannot access
each other

```
int main ()
{
   int x, int y;
   ...
   ...

   func1 ( 10, 20 );

   ...
   ...
}
```

x and y's
variable scope

Only **top** stack frame is active
and its variable can be accessed

Communications are performed
only through the **parameter** variables



Pass by Reference

```
int func1 (int* a, int* b)
{
   int i, int j;
   ...
   ...

   ...
   ...
}
```

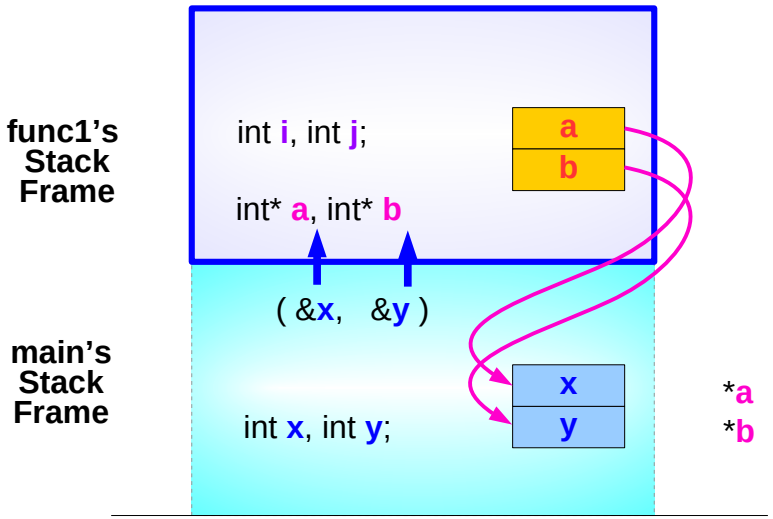
```
int main ()
{
   int x, int y;
   ...
   ...
   func1 ( &x, &y );

   ...
   ...
}
```

x and y are made known to **func1**
func1 can read / write x and y
through their addresses

*a
*b

a=&x
b=&y

x and y's
variable scope



Arrays

Accessing array elements – using an address

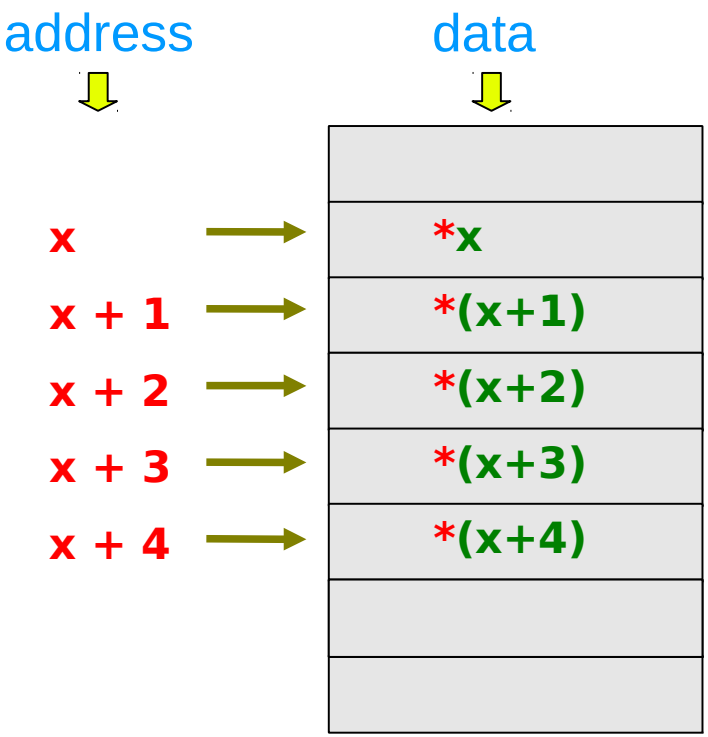
```
int     x[5];
```

x holds the *starting address*
*of **5** consecutive **int** variables*

5 int variables

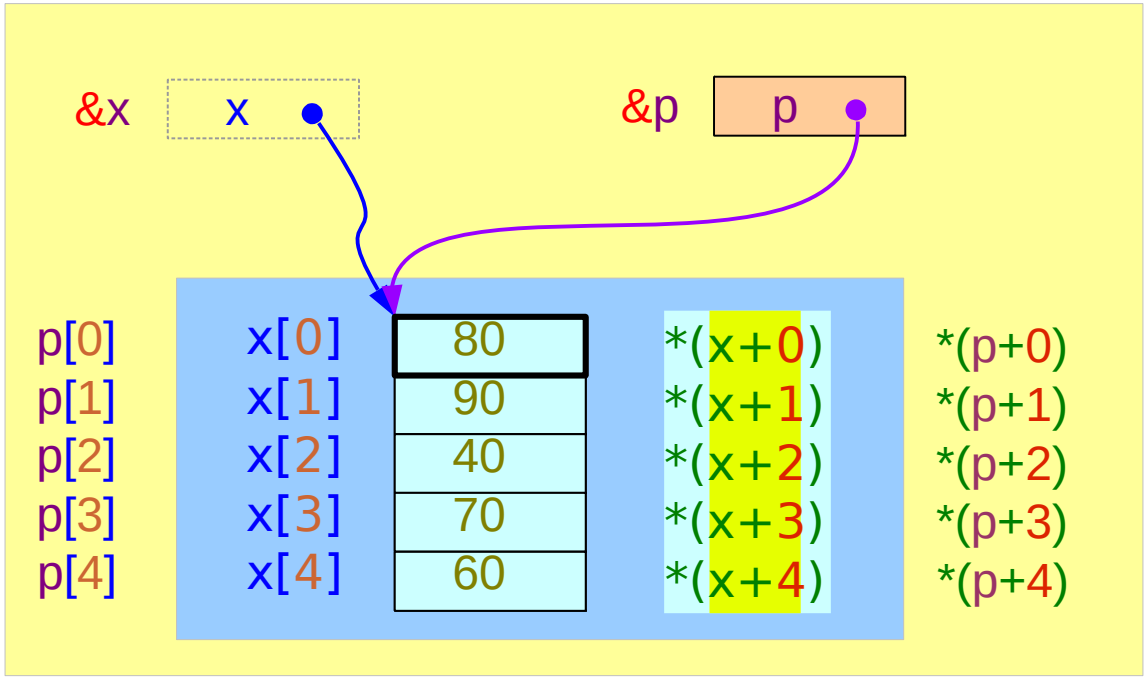
index	data
0	x[0]
1	x[1]
2	x[2]
3	x[3]
4	x[4]

cannot change
address x
(constant)



Accessing an Array with a Pointer Variable

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

int  *p = x;
```



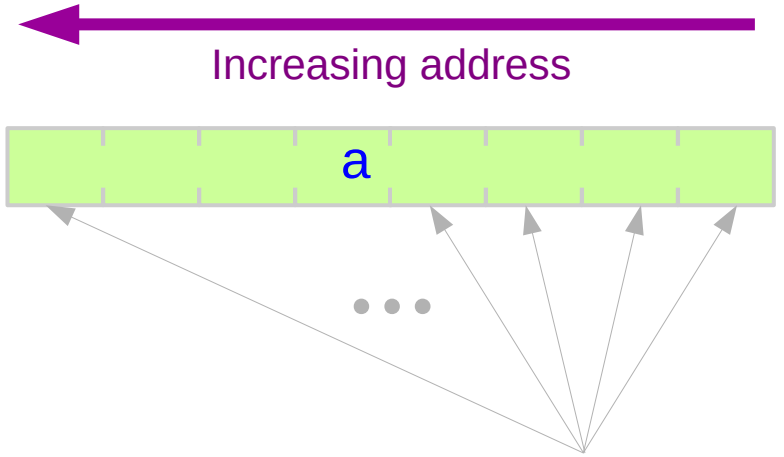
x is a constant symbol
cannot be changed

p is a variable
can point to other addresses

Byte Address
Little Endian
Big Endian

Byte Addresses

long a;



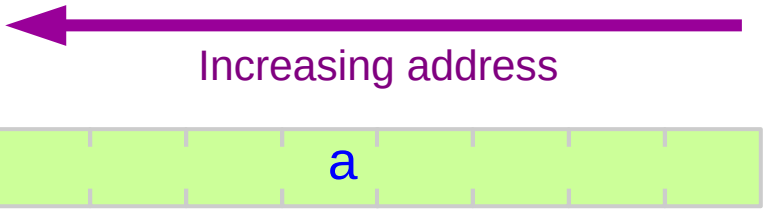
8-byte size data type

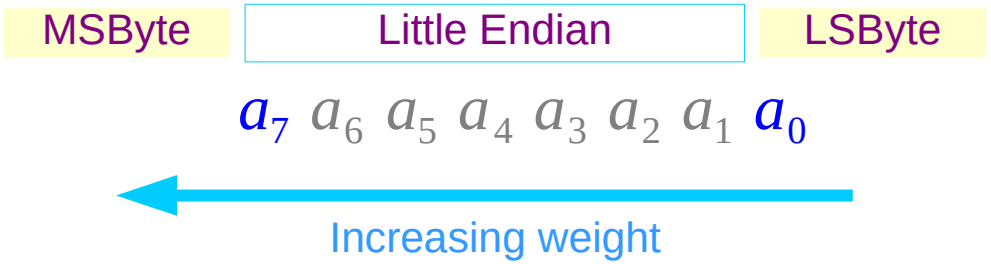
&a ?

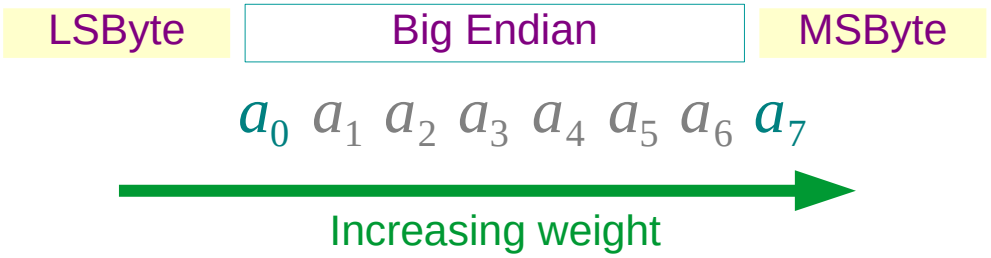
which byte?

Little / Big Endian Ordering of Bytes

long a;

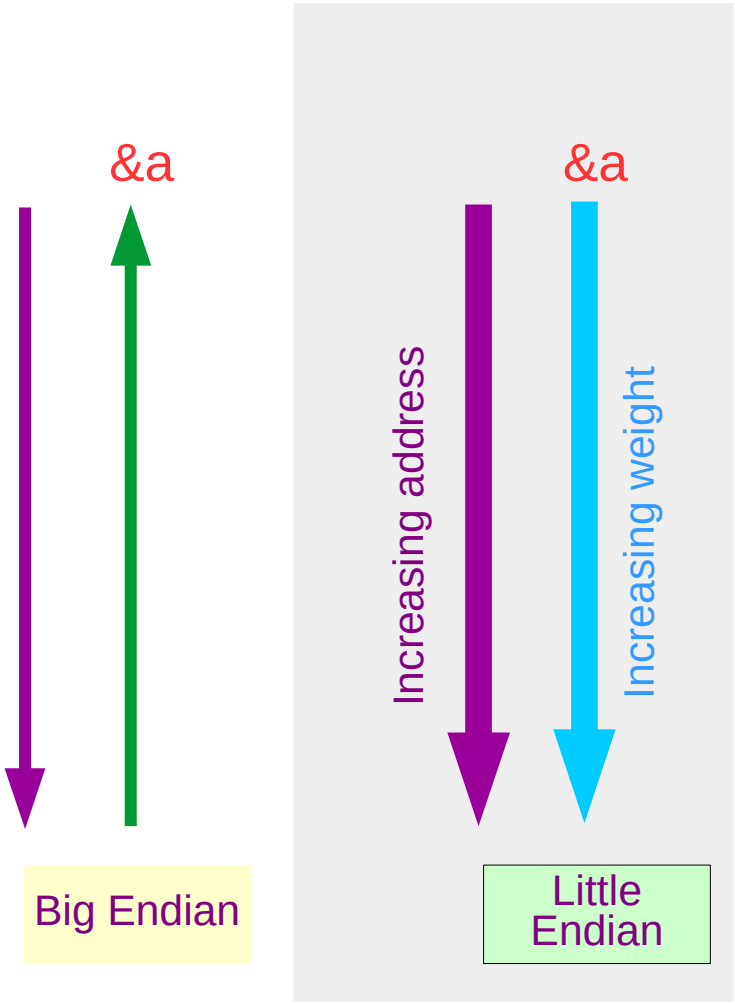




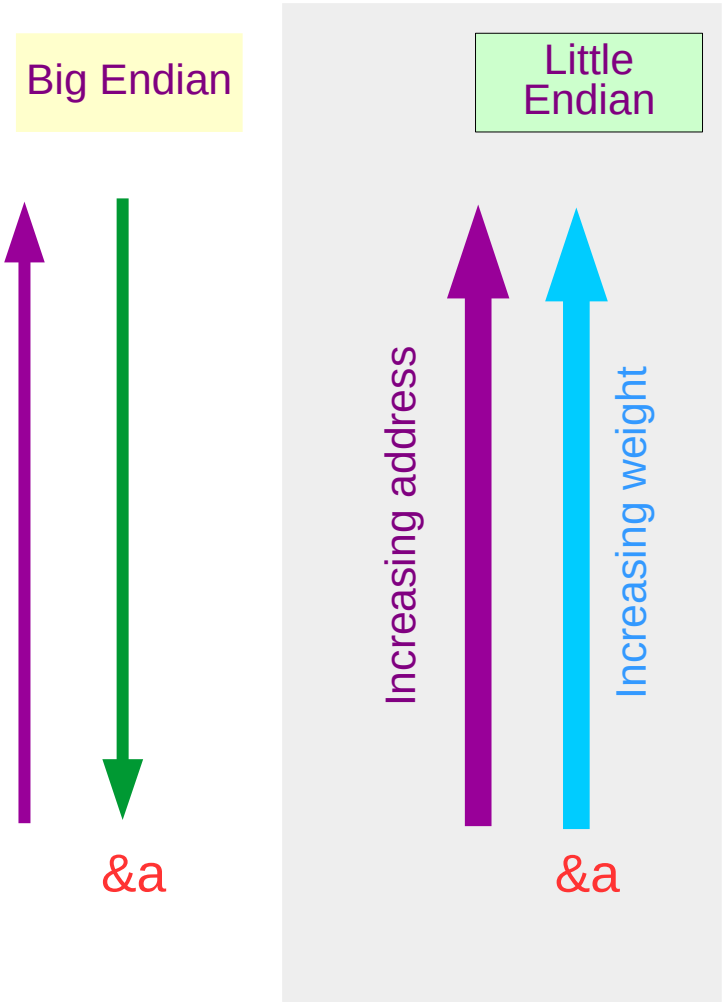


Increasing address, Increasing weight

downward, increasing address



upward, increasing address

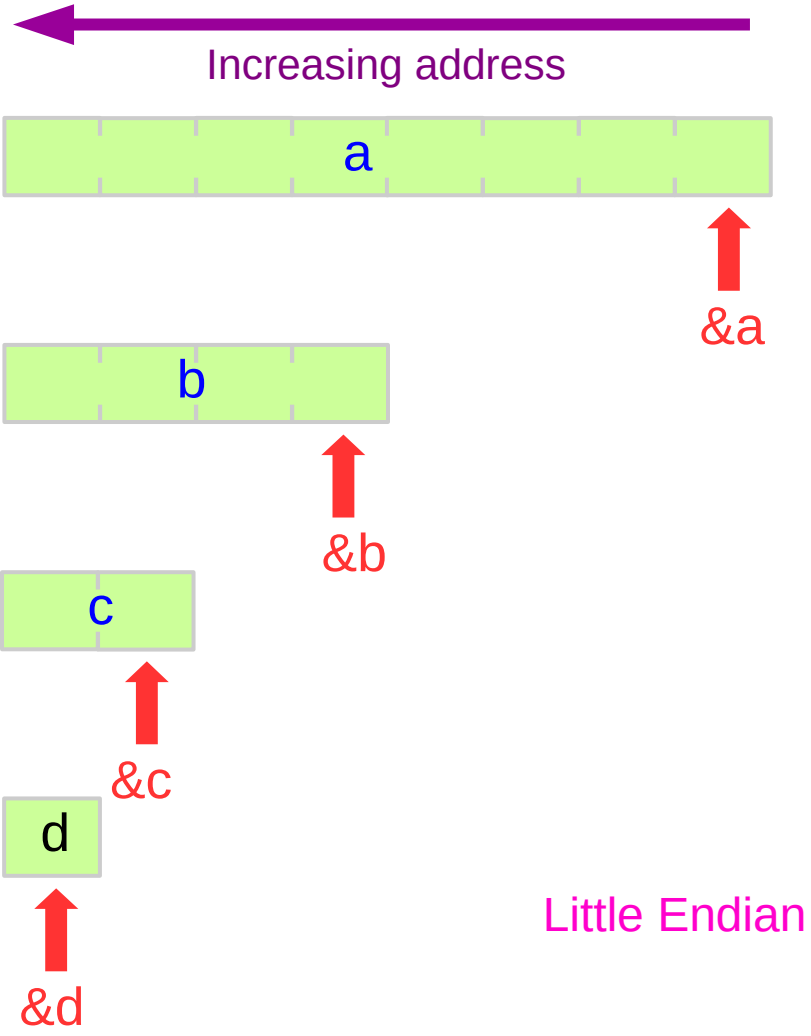


https://stackoverflow.com/questions/15620673/which-bit-is-the-address-of-an-integer

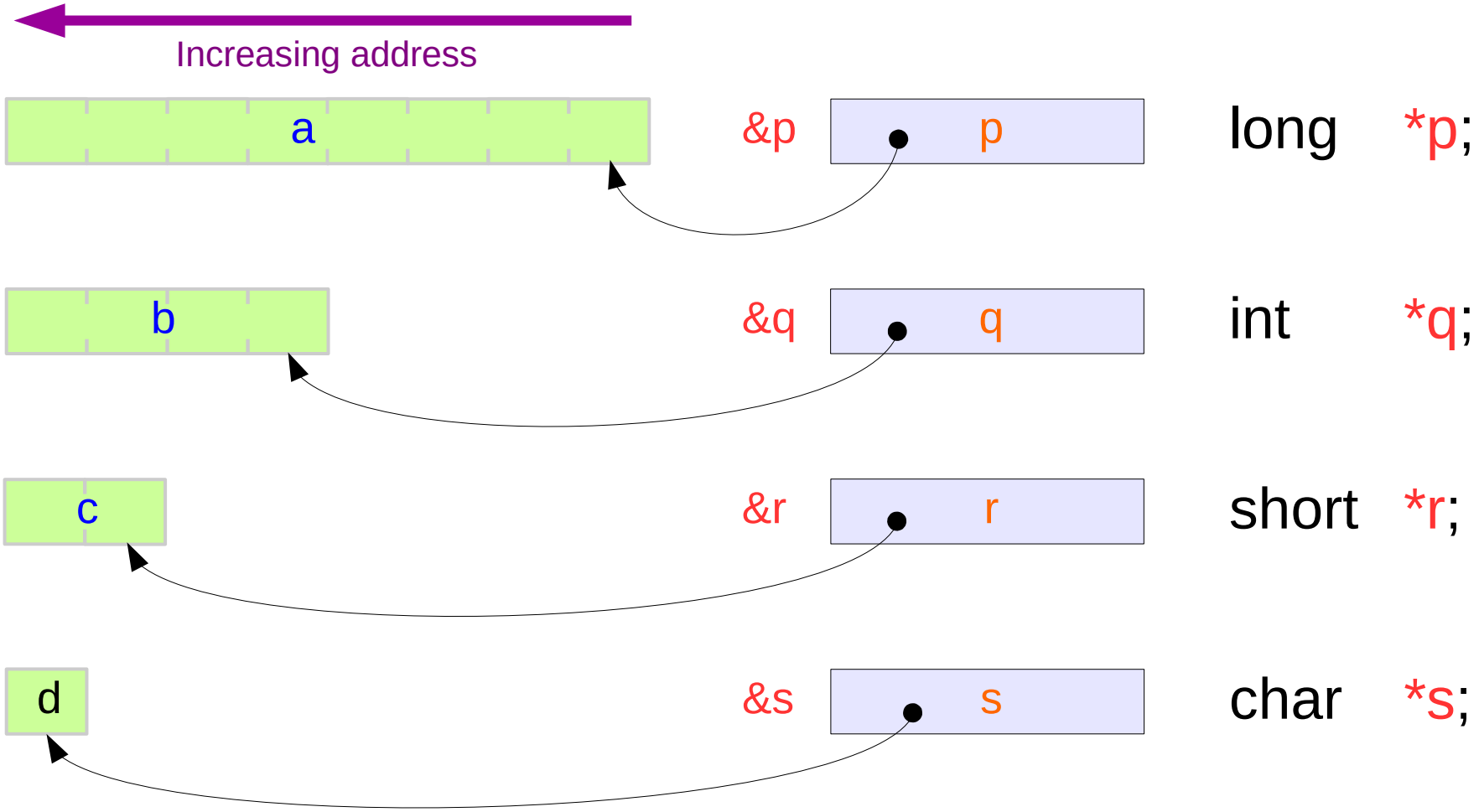
Pointer Types

Integer Type Variables and Their Addresses

```
long   a;
int    b;
short  c;
char   d;
```



Points to the LSByte



Aligning variables of different sizes

```
long   a;
int    b;
short  c;
char   d;
```

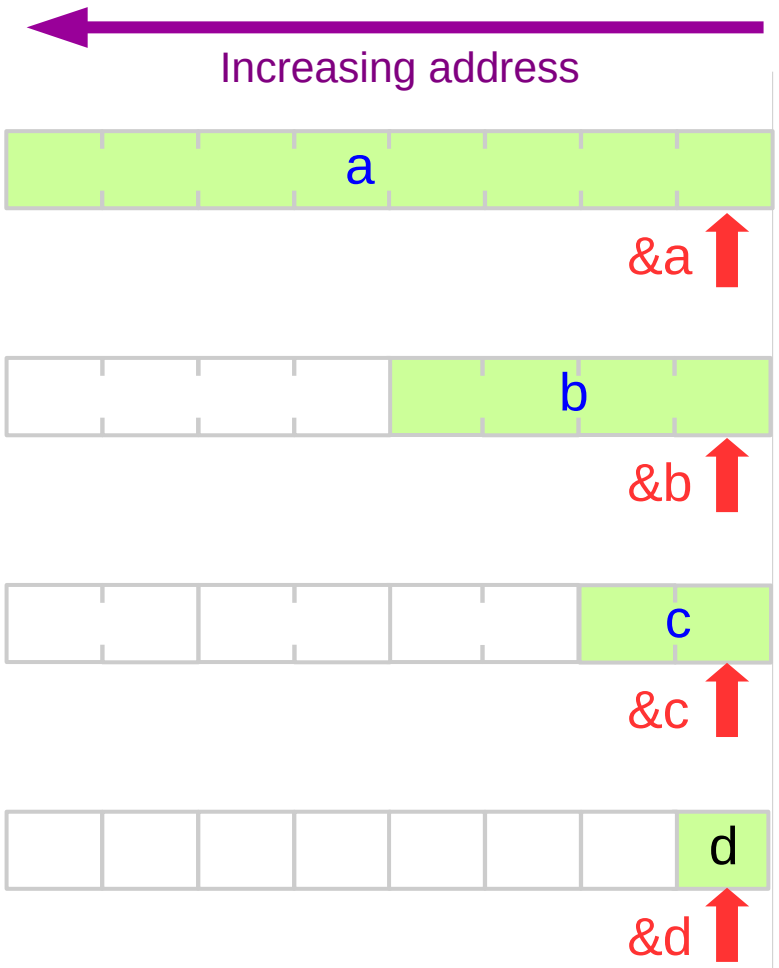
8-byte slots

4-byte slots

2-byte slots

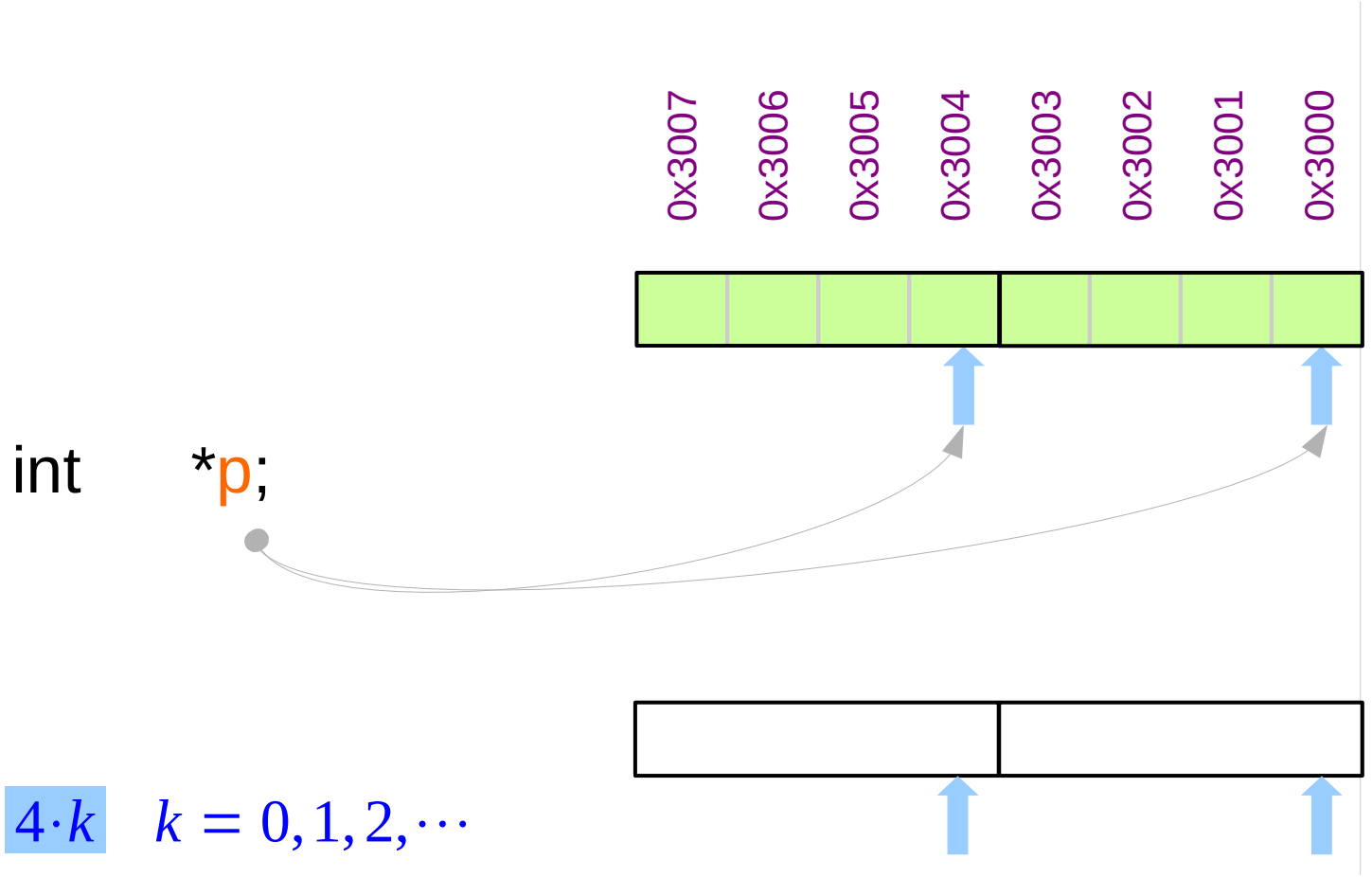
1-bytes slots

all these slots
are aligned

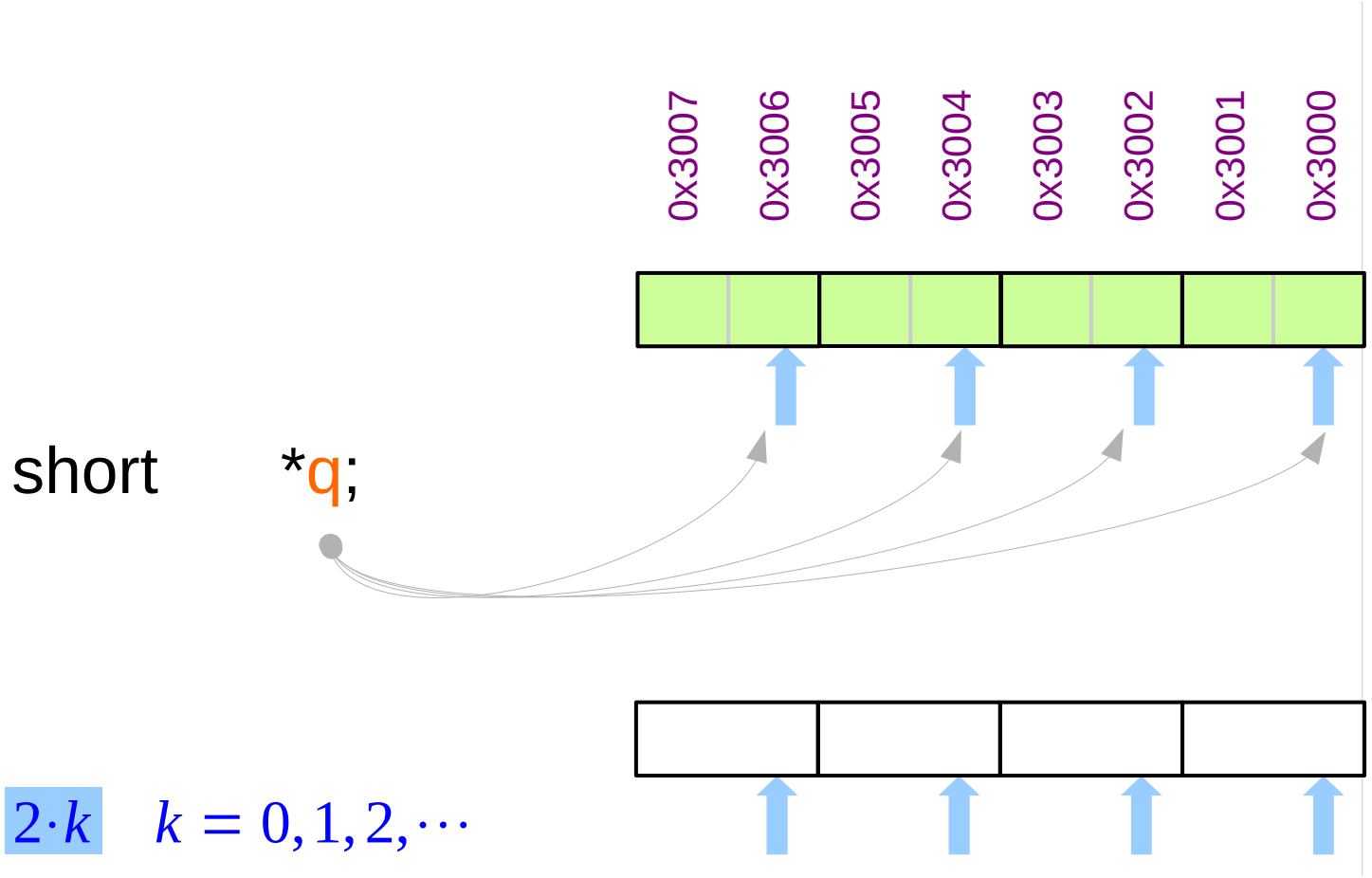


Memory Alignment
in the Little Endian

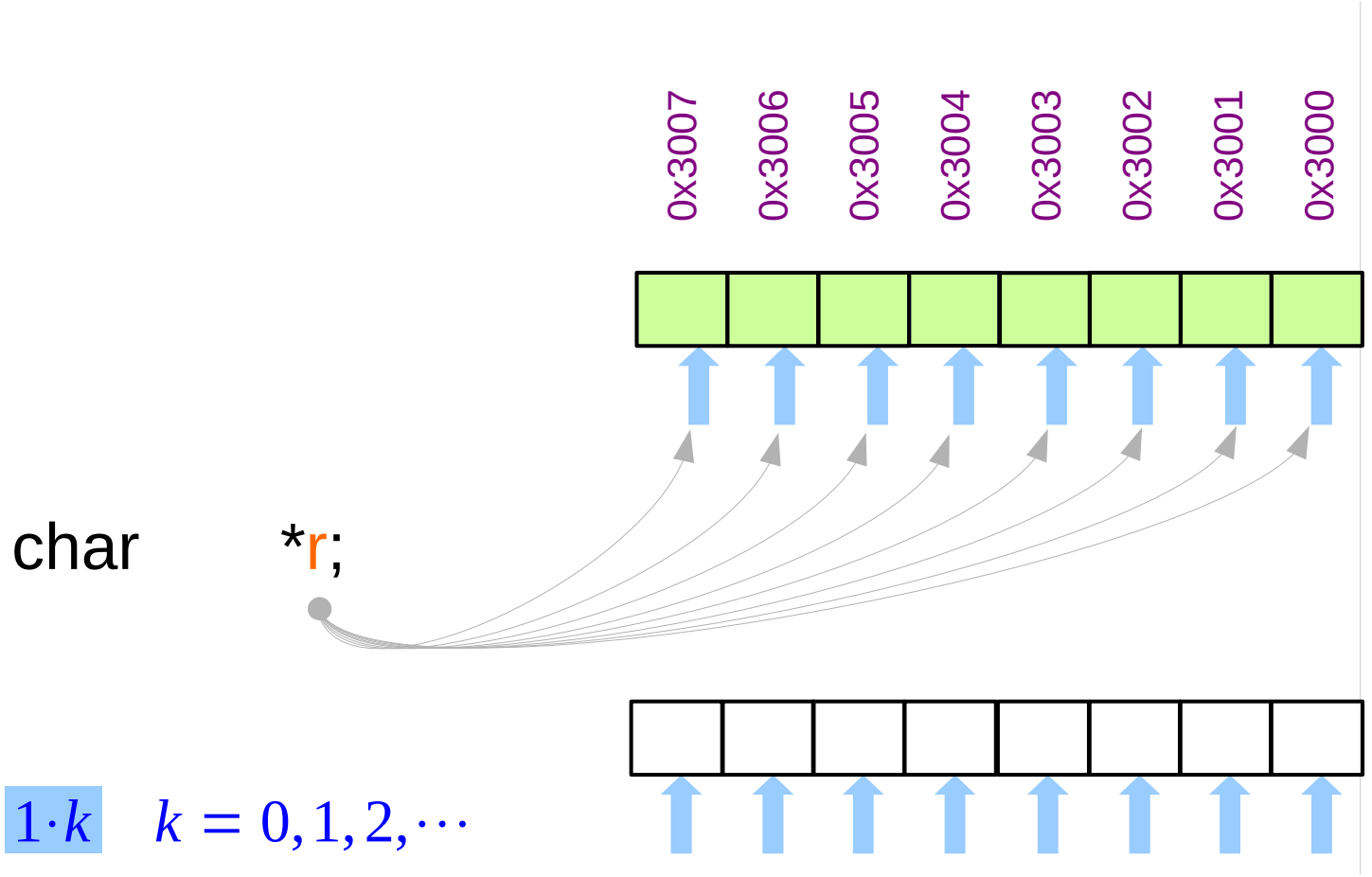
Possible addresses for **int** values



Possible addresses for **short** values

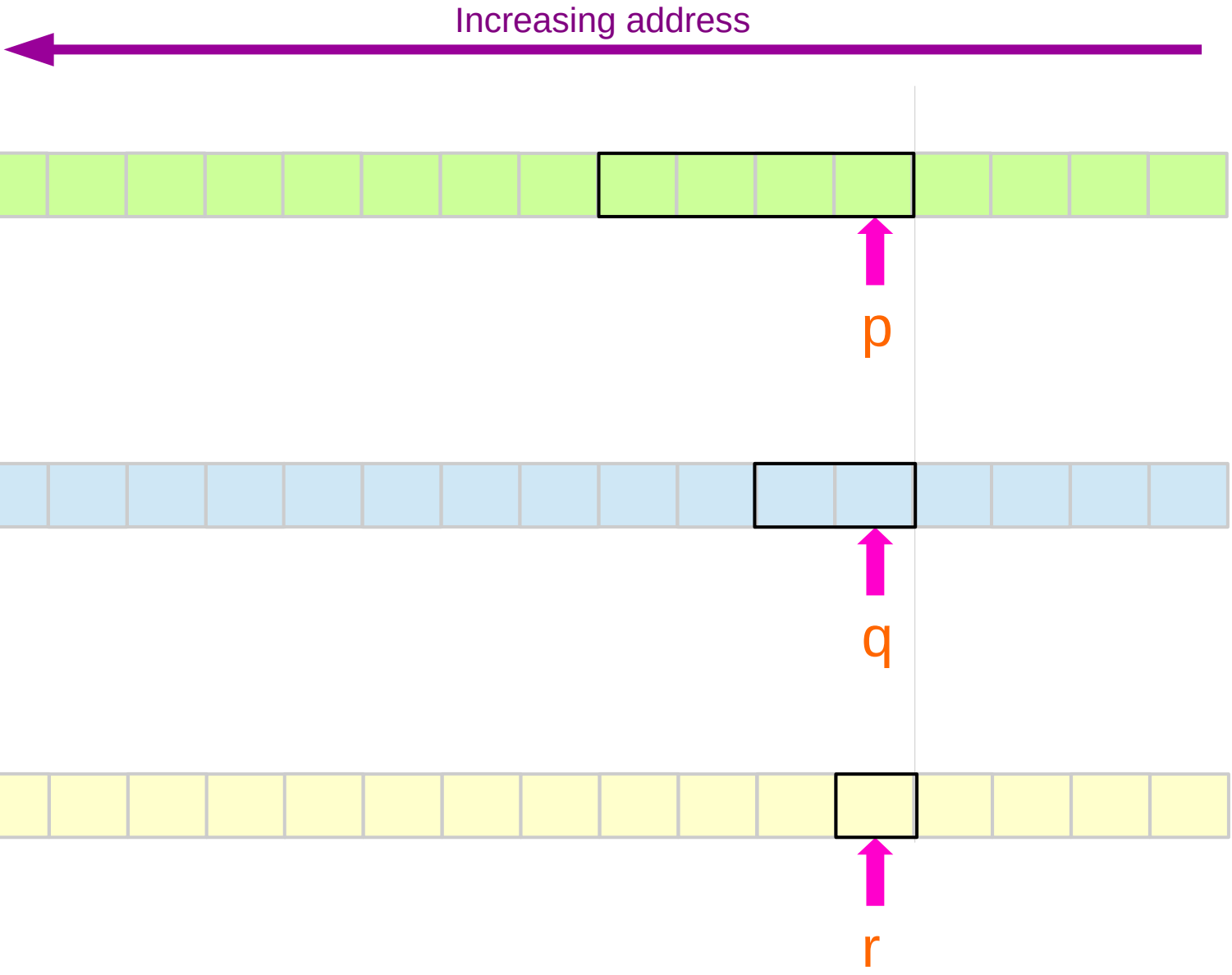


Possible addresses for **char** values

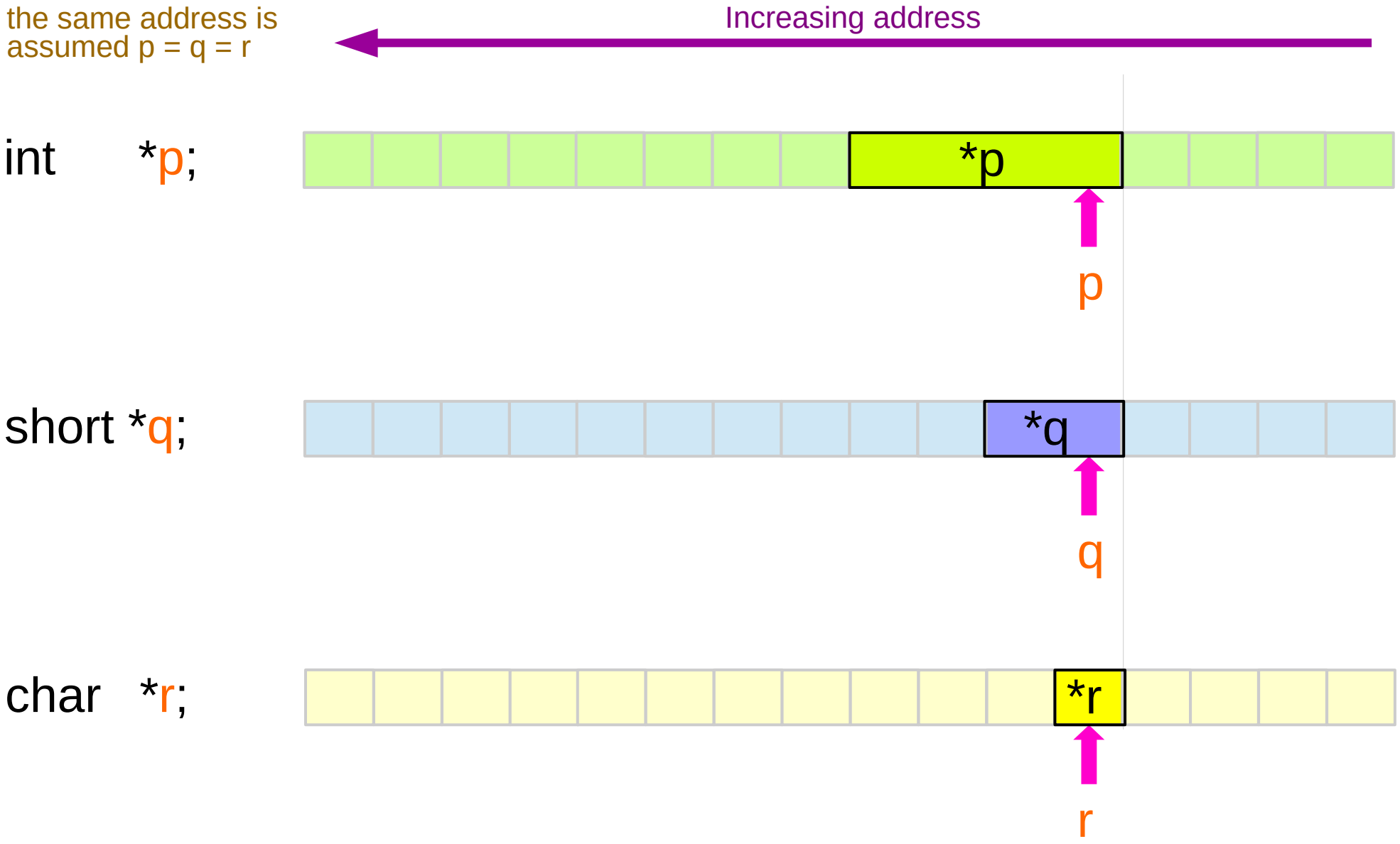


Data size at an address

the same address is
assumed p = q = r

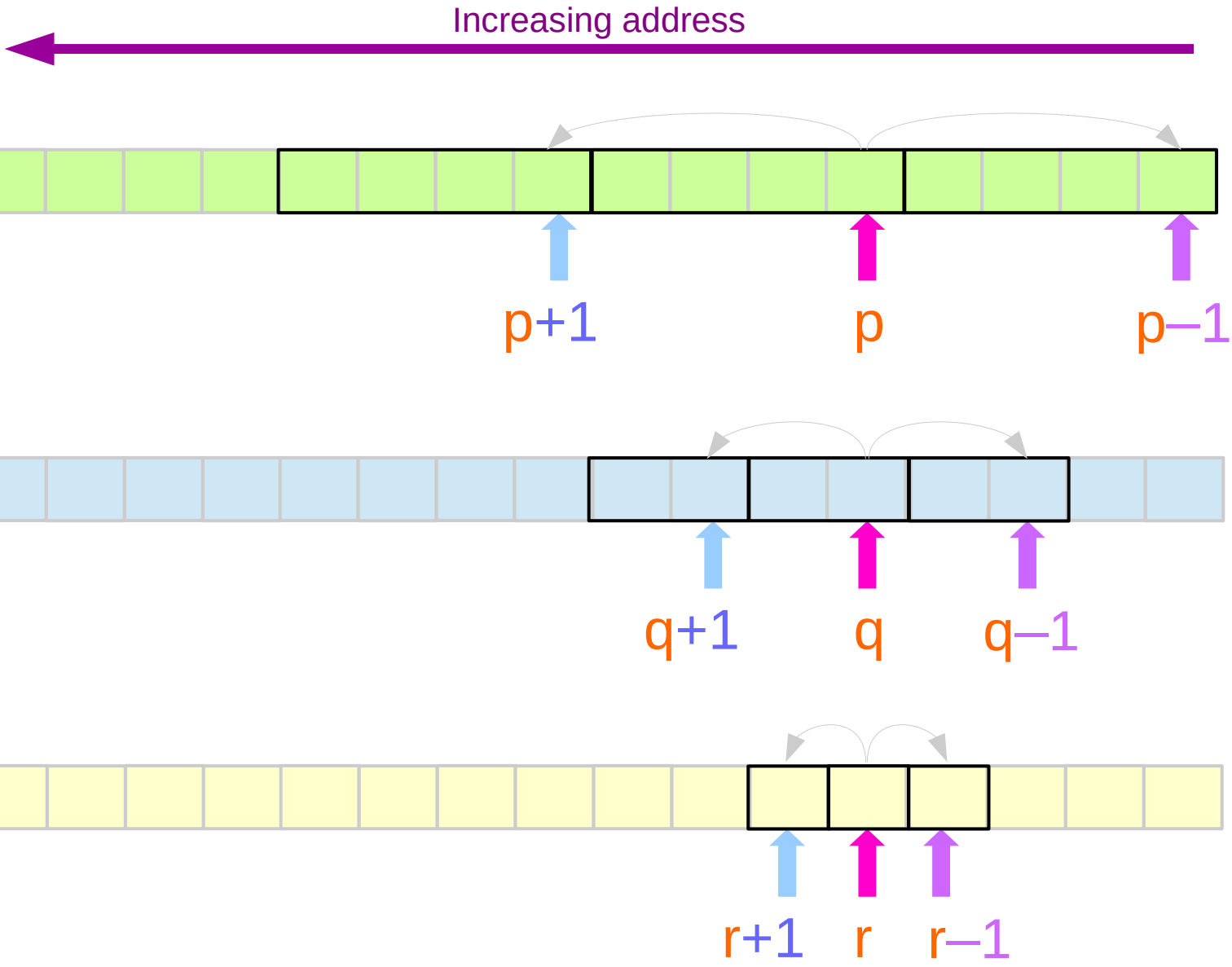


Associated data at the pointed addresses



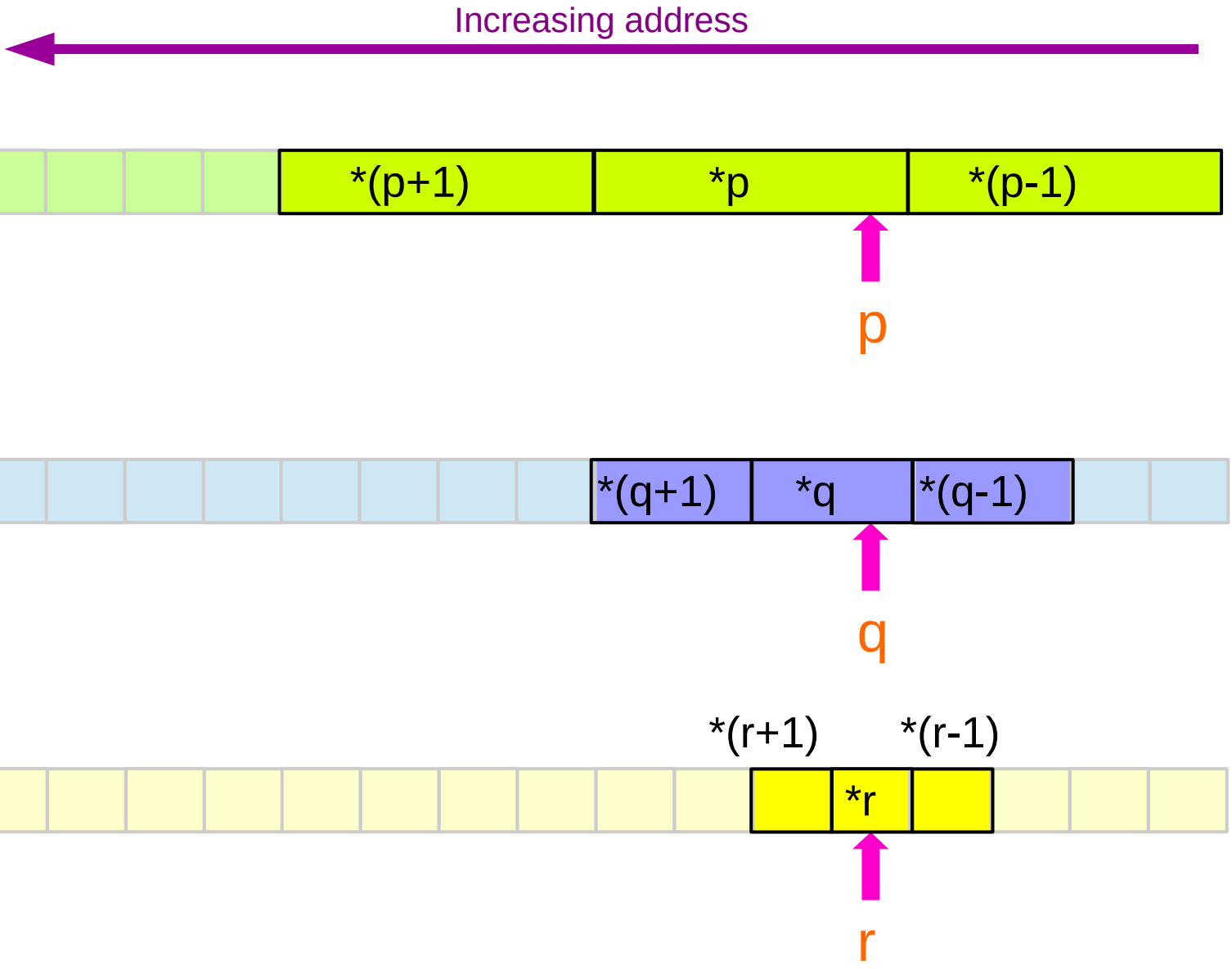
Incrementing / decrementing pointers

the same address is
assumed p = q = r



Dereferencing the pointers

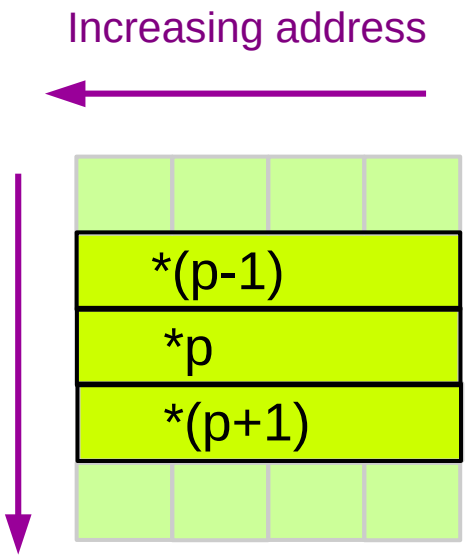
the same address is
assumed p = q = r

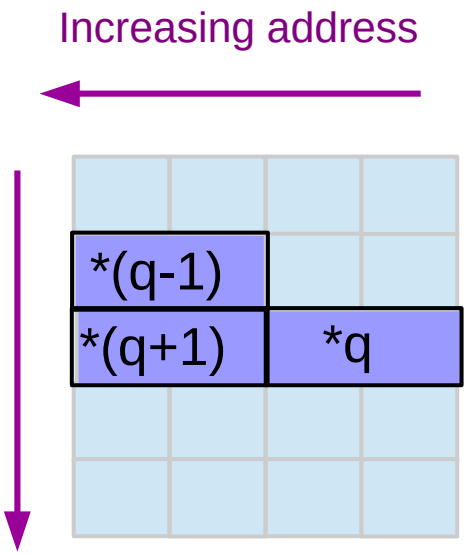


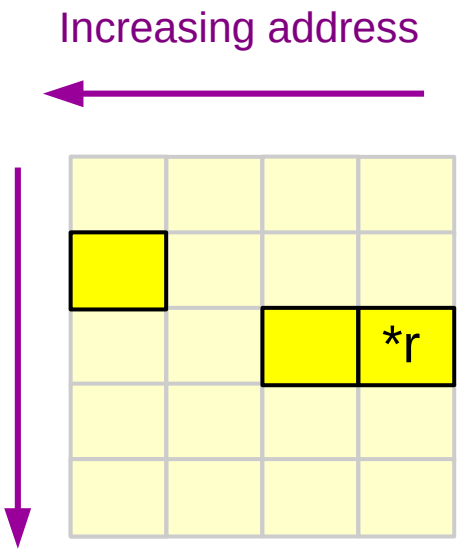
32-bit word view

the same address is
assumed p = q = r

1 word = 4 bytes = 4 (8 bits) = 32 bits





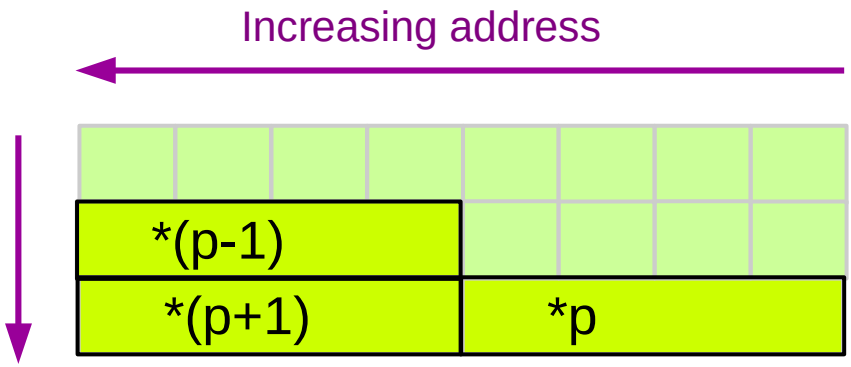


int *p;

short *q;

char *r;

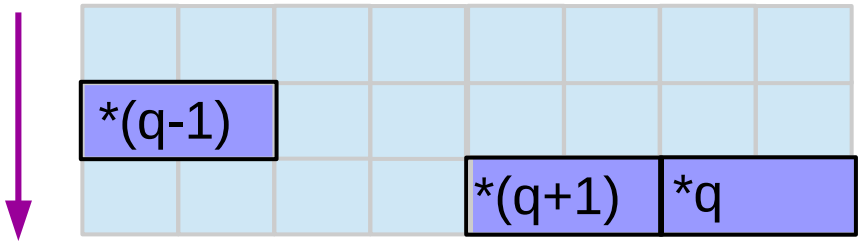
64-bit word view



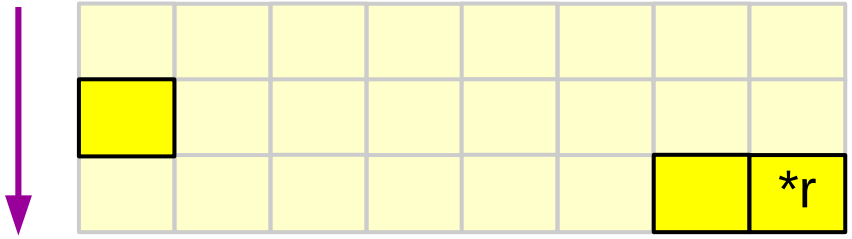
1 word = 8 bytes = 8 (8 bits) = 64 bits

the same address is
assumed p = q = r

int *p;

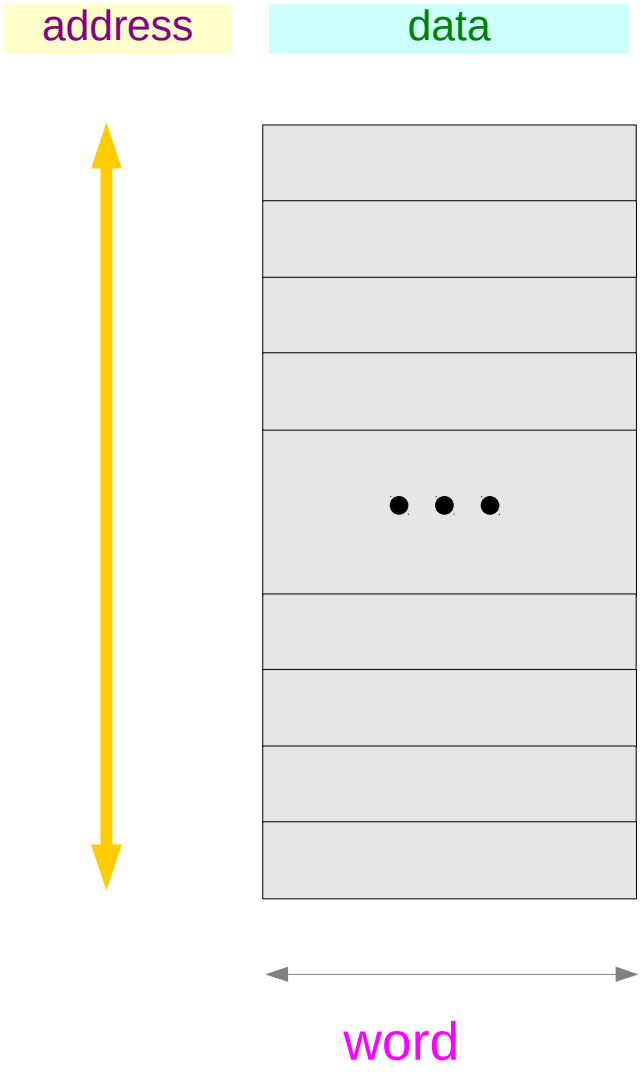


short *q;



char *r;

32-bit vs. 64-bit machines



data unit per memory access	
32-bit	64-bit
word size	
32-bit	64-bit
max address space	
2^32	2^64

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun