Representing Numbers In Memory

Mark Allman
mallman@case.edu

EECS 325/425
Fall 2018

"To workers I'm just another drone,
To Ma Bell I'm just another phone,
I'm just another statistic on a sheet"

Representing Numbers

Representing Numbers

- How is some number like 67,305,985 stored?

Representing Numbers

- How is some number like 67,305,985 stored?

```
unsigned int x = 67305985;
```

Representing Numbers

- How is some number like 67,305,985 stored?

unsigned int x = 67305985;

- First step: stored in binary. So,

```
>>> bin (67305985)
'0b100000000110000001000000001'
```

Representing Numbers

Representing Numbers

```
>>> bin (67305985)
'0b100000000110000001000000001'
```

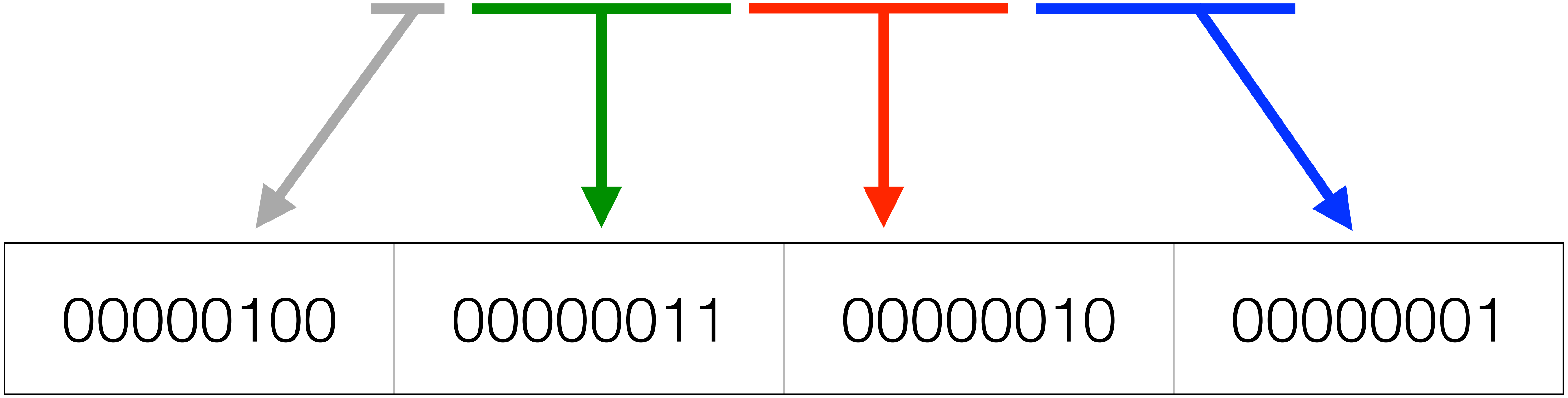
Representing Numbers

```
>>> bin (67305985)
'0b100000000110000001000000001'
```

00000100	00000011	00000010	00000001

Representing Numbers

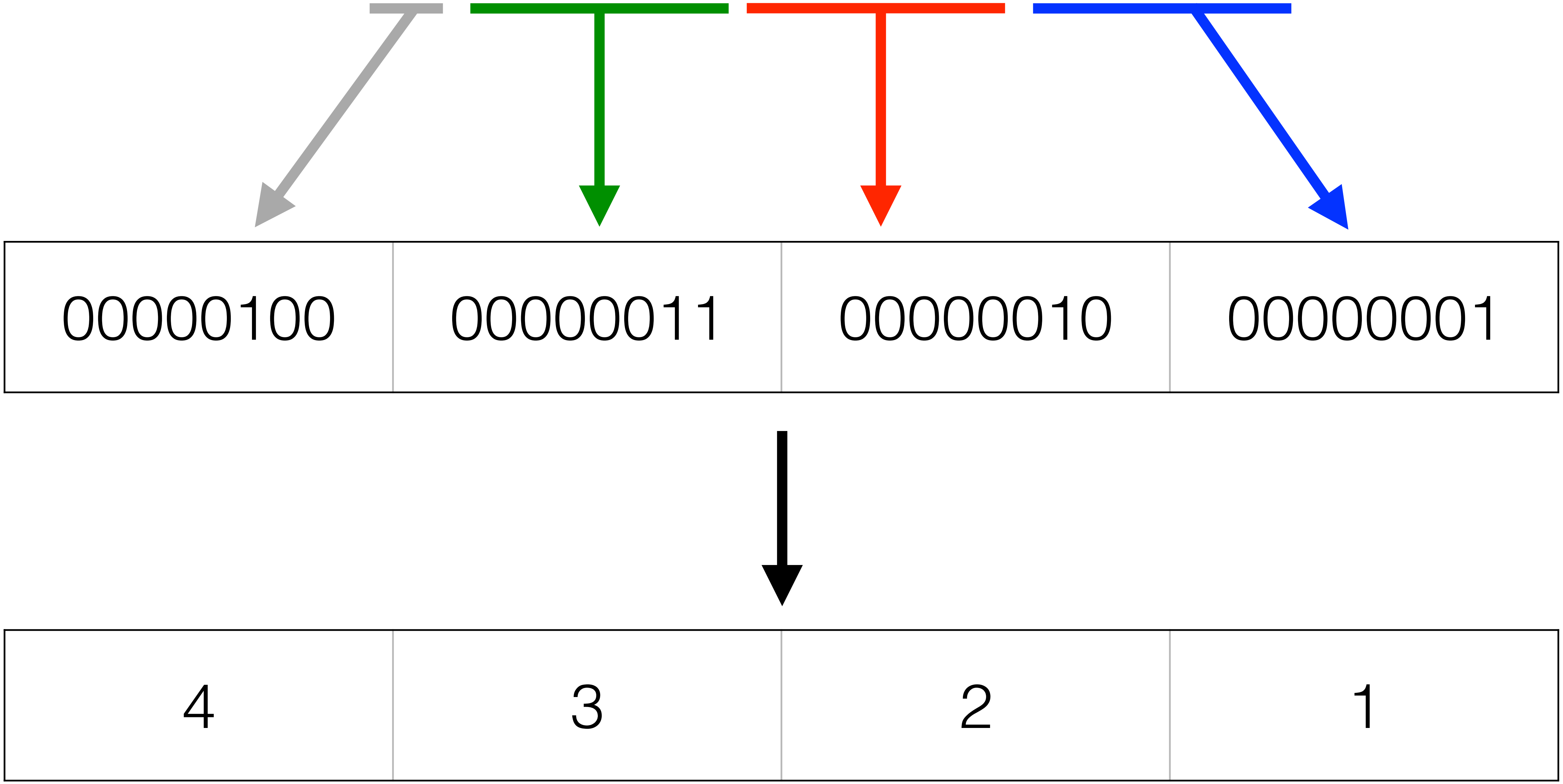
```
>>> bin (67305985)
'0b100000001100000001000000001'
```



00000100	00000011	00000010	00000001

Representing Numbers

```
>>> bin (67305985)
'0b100000000110000001000000001'
```

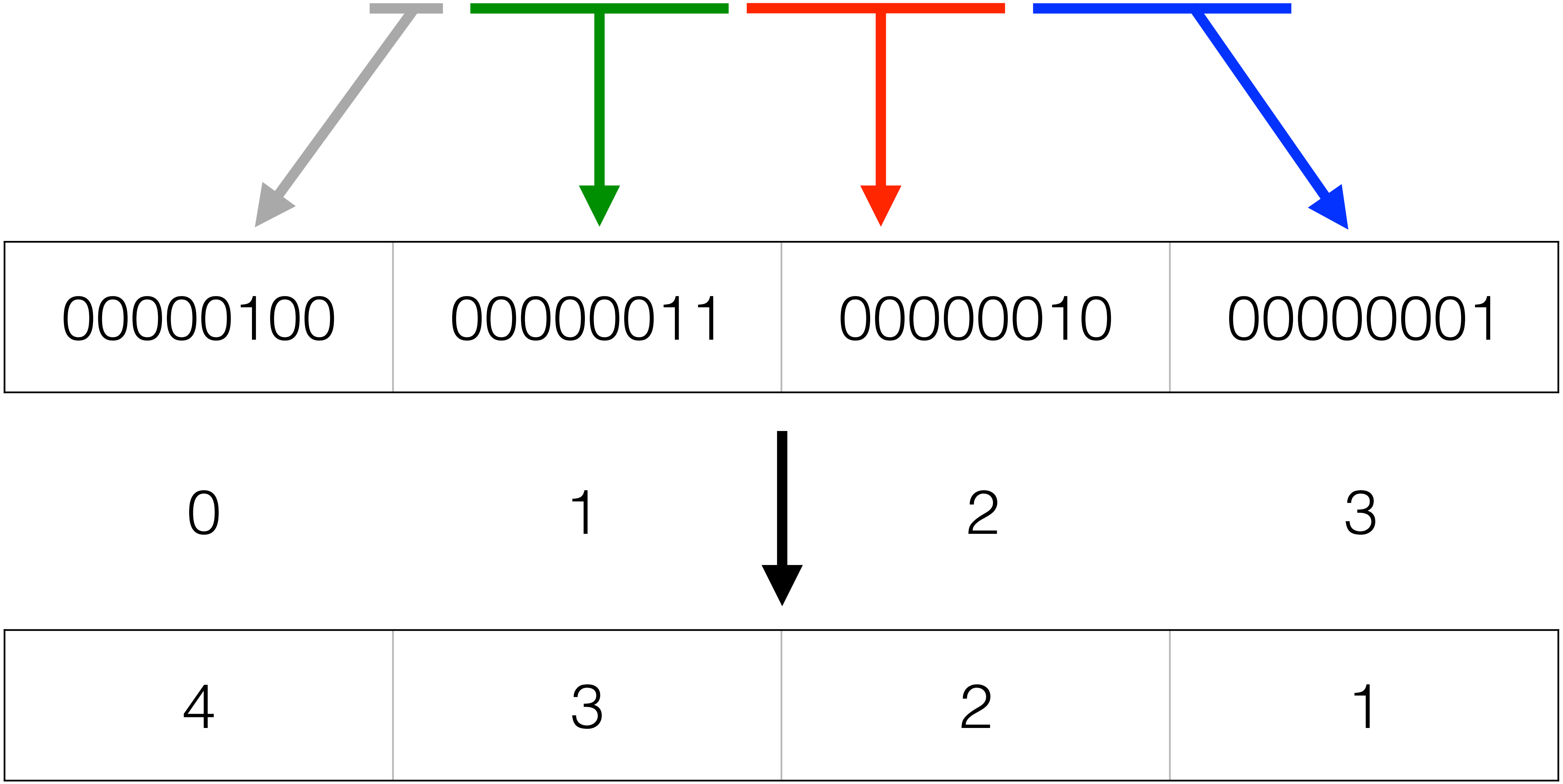


00000100	00000011	00000010	00000001

4	3	2	1

Representing Numbers

```
>>> bin (67305985)
'0b100000000110000001000000001'
```



00000100	00000011	00000010	00000001
0	1	2	3
4	3	2	1

Peeking Into Memory

```
int main ()
{
    unsigned int x = 67305985;
    unsigned char bytes [4];
    unsigned short i;

    printf ("sizeof (unsigned int) == %lu\n",
            sizeof (unsigned int));
    memcpy (bytes,&x,sizeof (unsigned int));
    for (i = 0; i < sizeof (unsigned int); i++)
        printf ("position: %d value: %d\n", i, bytes [i]);
}
```

Peeking Into Memory

```
int main ()
{
    unsigned int x = 67305985;
    unsigned char bytes [4];
    unsigned short i;

    printf ("sizeof (unsigned int) == %lu\n",
            sizeof (unsigned int));
    memcpy (bytes,&x,sizeof (unsigned int));
    for (i = 0; i < sizeof (unsigned int); i++)
        printf ("position: %d value: %d\n", i, bytes [i]);
}
```

```
% ./foo
sizeof (unsigned int) == 4
```

Peeking Into Memory

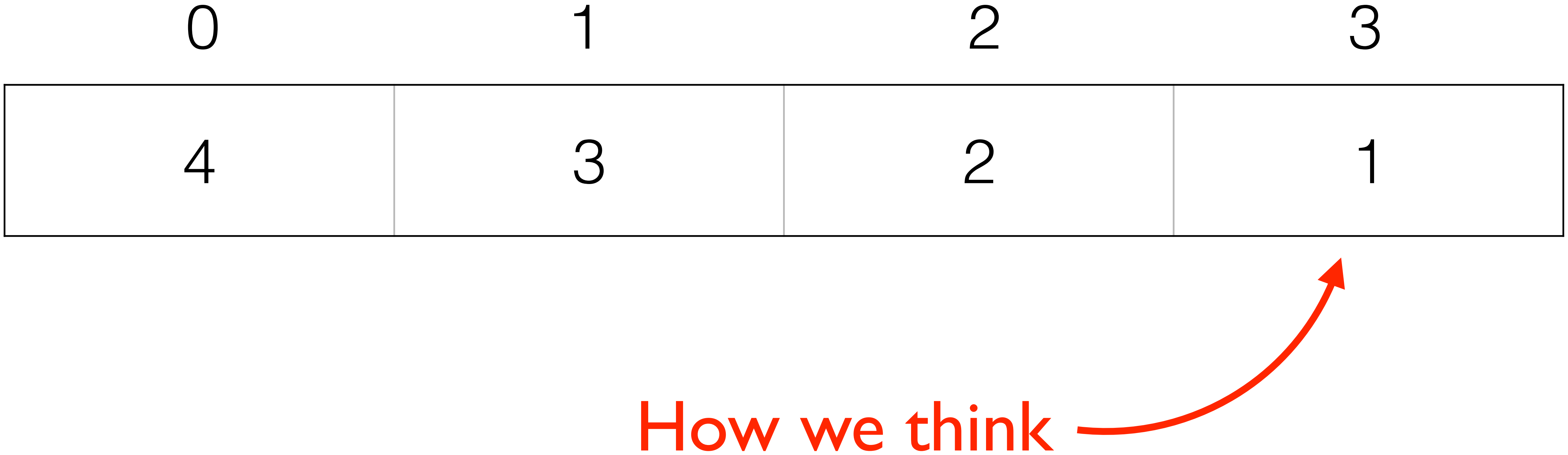
```
int main ()
{
    unsigned int x = 67305985;
    unsigned char bytes [4];
    unsigned short i;

    printf ("sizeof (unsigned int) == %lu\n",
            sizeof (unsigned int));
    memcpy (bytes,&x,sizeof (unsigned int));
    for (i = 0; i < sizeof (unsigned int); i++)
        printf ("position: %d value: %d\n", i, bytes [i]);
}
```

```
% ./foo
sizeof (unsigned int) == 4
position: 0 value: 1
position: 1 value: 2
position: 2 value: 3
position: 3 value: 4
```

OH NO!

0	1	2	3
4	3	2	1

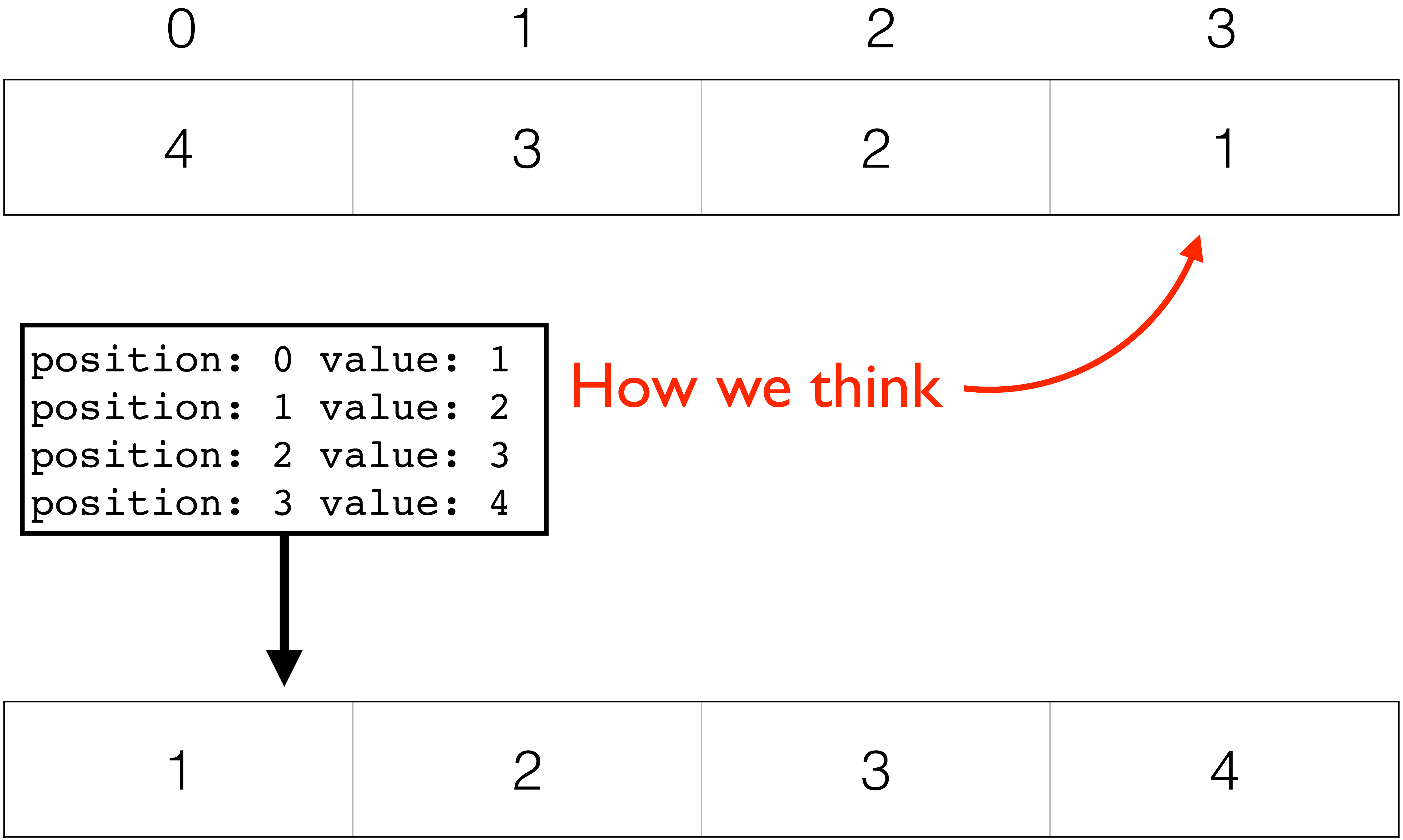
OH NO!



0	1	2	3
4	3	2	1

How we think

OH NO!



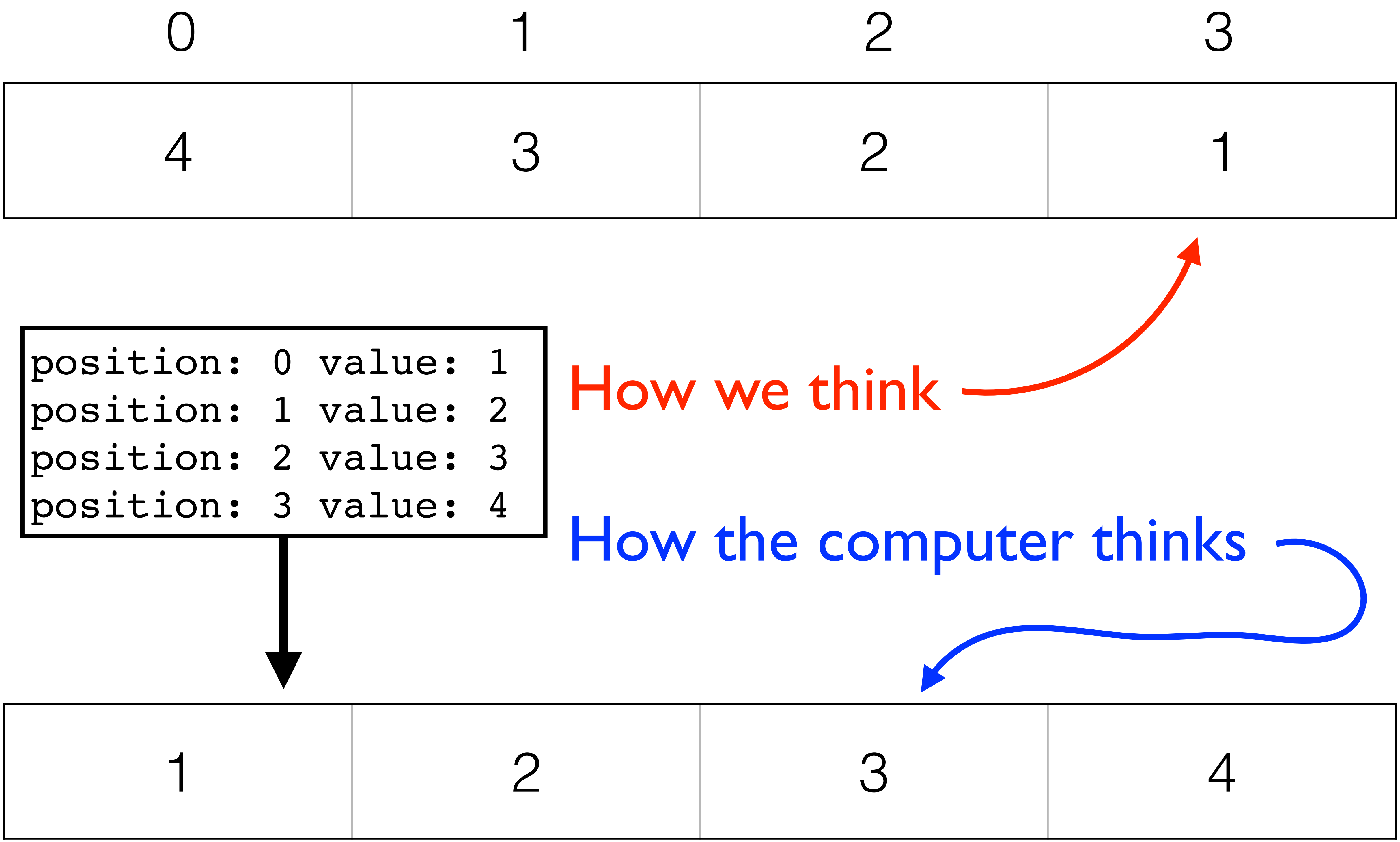
0	1	2	3
4	3	2	1

```
position: 0 value: 1
position: 1 value: 2
position: 2 value: 3
position: 3 value: 4
```

How we think

1	2	3	4

OH NO!



0	1	2	3
4	3	2	1

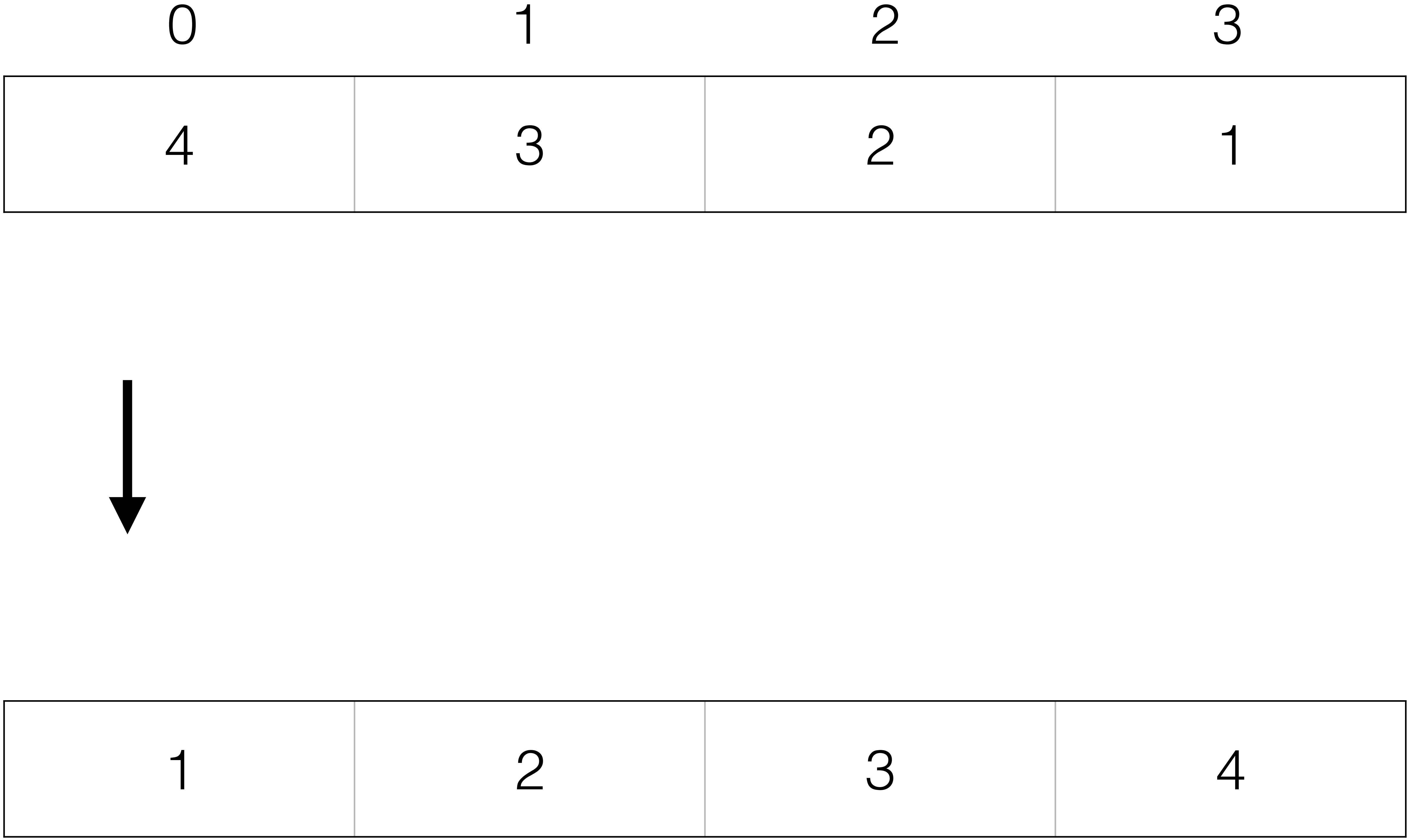
```
position: 0 value: 1
position: 1 value: 2
position: 2 value: 3
position: 3 value: 4
```

How we think

How the computer thinks

1	2	3	4

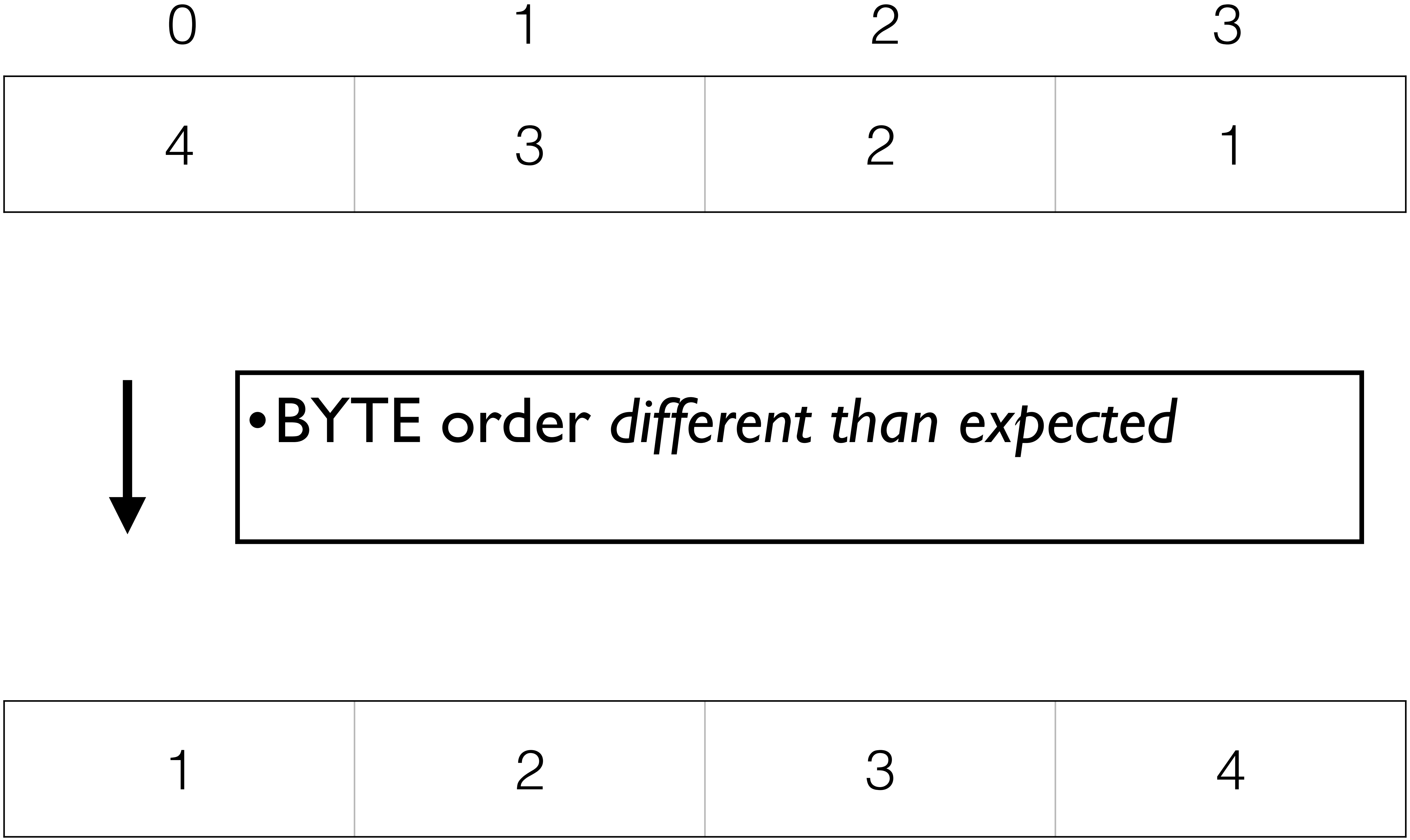
Byte-vs-Bit Order



0	1	2	3
4	3	2	1

1	2	3	4

Byte-vs-Bit Order

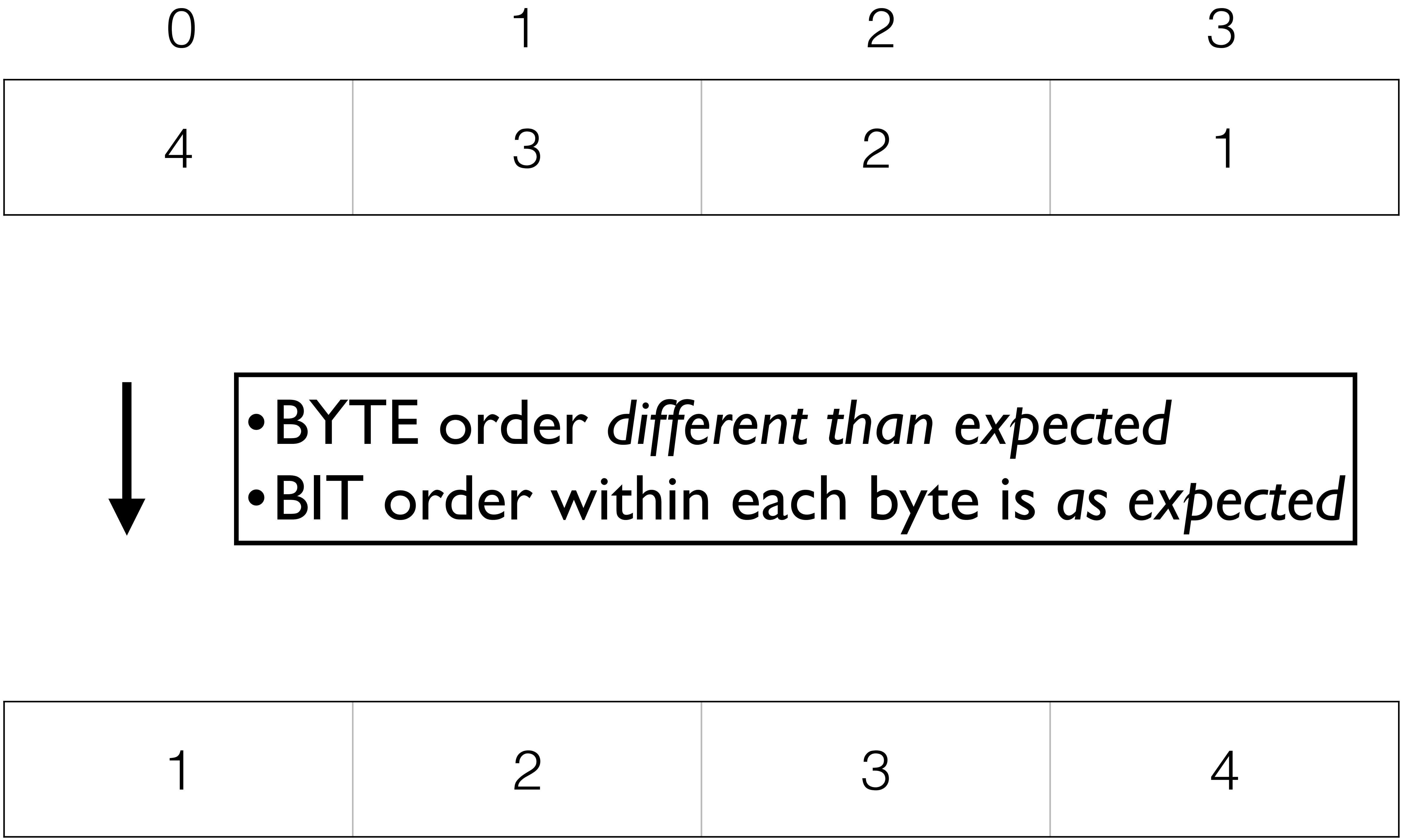


0	1	2	3
4	3	2	1

- BYTE order *different than expected*

1	2	3	4

Byte-vs-Bit Order

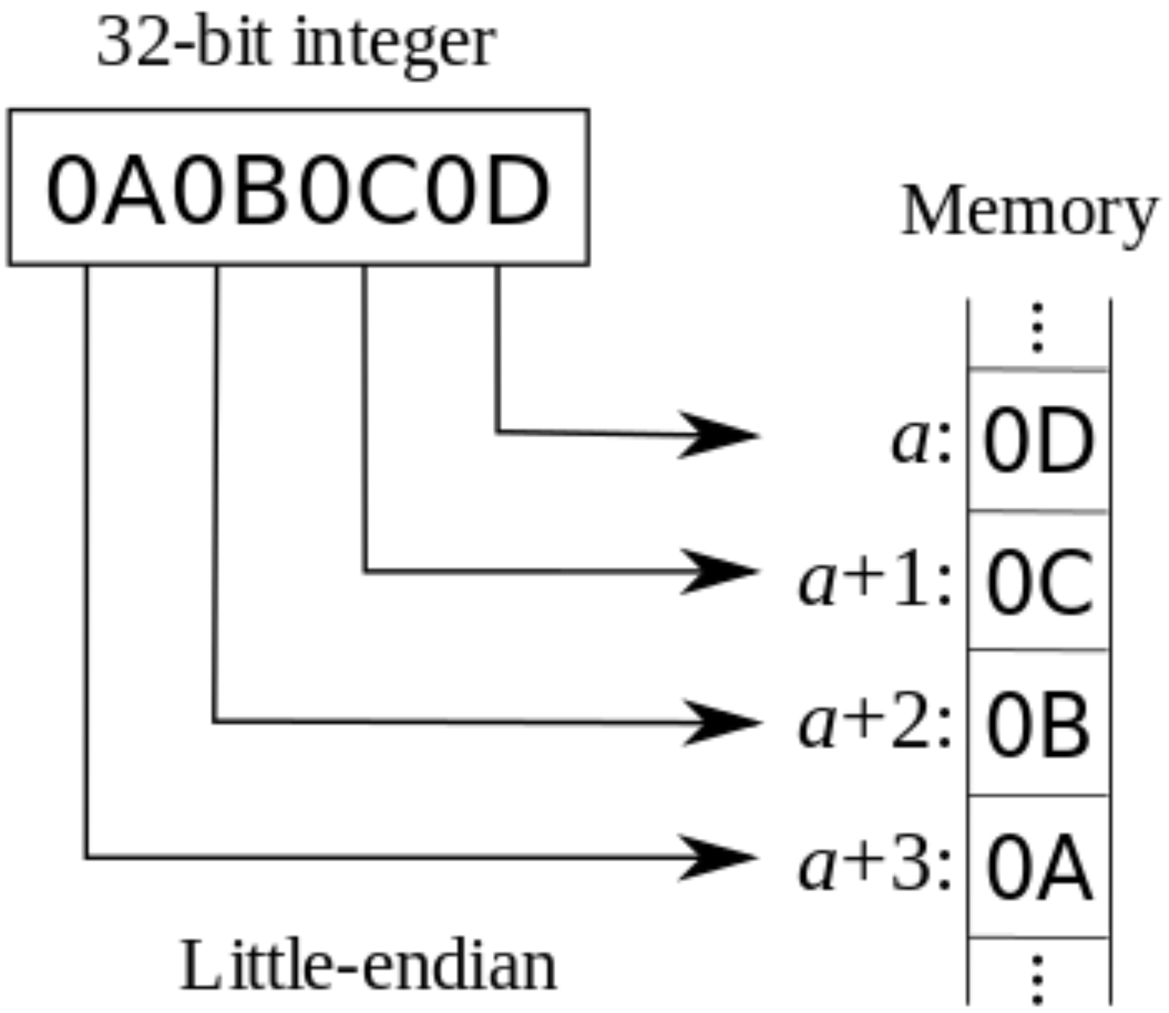


0	1	2	3
4	3	2	1

- BYTE order *different than expected*
- BIT order within each byte is *as expected*

1	2	3	4

Little Endian



So What?

So What?

- As long as the computer thinks in an internally consistent way this is perfectly fine …

  … even if that is different from how we think!

  … even if that is different from how another computer thinks!

So What?

- As long as the computer thinks in an internally consistent way this is perfectly fine …

  … even if that is different from how we think!

  … even if that is different from how another computer thinks!

- So, what’s the big deal?
  - I.e., you have probably never before thought about this, so who cares?

Networks Causin’ Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (x);
}
```

Host B:

```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    printf ("y == %d\n", y);
}
```

Networks Causin' Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (x);
}
```

Host B:

```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    printf ("y == %d\n", y);
}
```

1	2	3	4

Networks Causin' Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (x);
}
```

Host B:

```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    printf ("y == %d\n", y);
}
```

1	2	3	4

Networks Causin' Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (x);
}
```

Host B:

```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    printf ("y == %d\n", y);
}
```

1	2	3	4

```
y == 16909058
```

Networks Causin’ Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (x);
}
```

Host B:

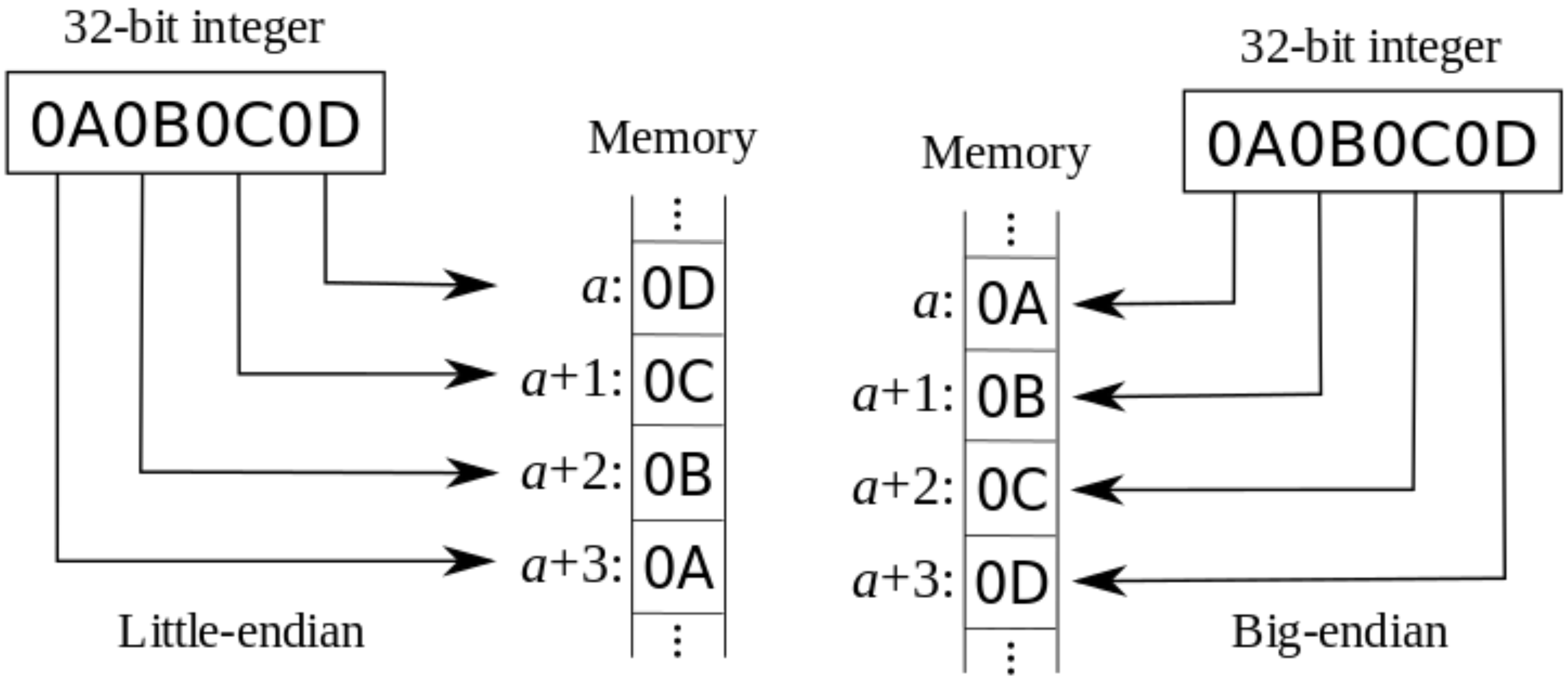
```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    printf ("y == %d\n", y);
}
```

1	2	3	4

```
y == 16909058
```

WHY?

Big vs. Little Endian



Network Byte Order

Network Byte Order

- TCP/IP networks use “Network Byte Order” (i.e., big endian)

Network Byte Order

- TCP/IP networks use “Network Byte Order” (i.e., big endian)

- Routines that deal with this for you:
  - before sending: *htonl (), htons ()*
  - after receiving: *ntohl (), ntohs ()*
- These swap bytes *when needed*

Dealin’ With Network Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (htonl (x));
}
```

Host B:

```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    y = ntohl (y)
    printf ("y == %d\n", y);
}
```

Dealin' With Network Problems

Host A:

```
main ()
{
    unsigned int x = 67305985;

    SendToHostB (htonl (x));
}
```

Host B:

```
main ()
{
    unsigned int y;

    y = RecvFromHostA ();
    y = ntohl (y)
    printf ("y == %d\n", y);
}
```

```
y == 67305985
```