Taller Rust, sesión 13

18/03/2023

- [x] prog. funcional (map, reduce)
- [x] operaciones bitwise: |, &, ^, <<, >>
- [x] conversion int ⟷ bits ⟷ int

  float ⟷ bits ⟷ float
- [ ] Socket, comunicación mensajes + complejos.
- [ ] control del await.

Operaciones bitwise:

'1 0 1 0 1 1 1 1' → byte

8 bits

| (+, or)

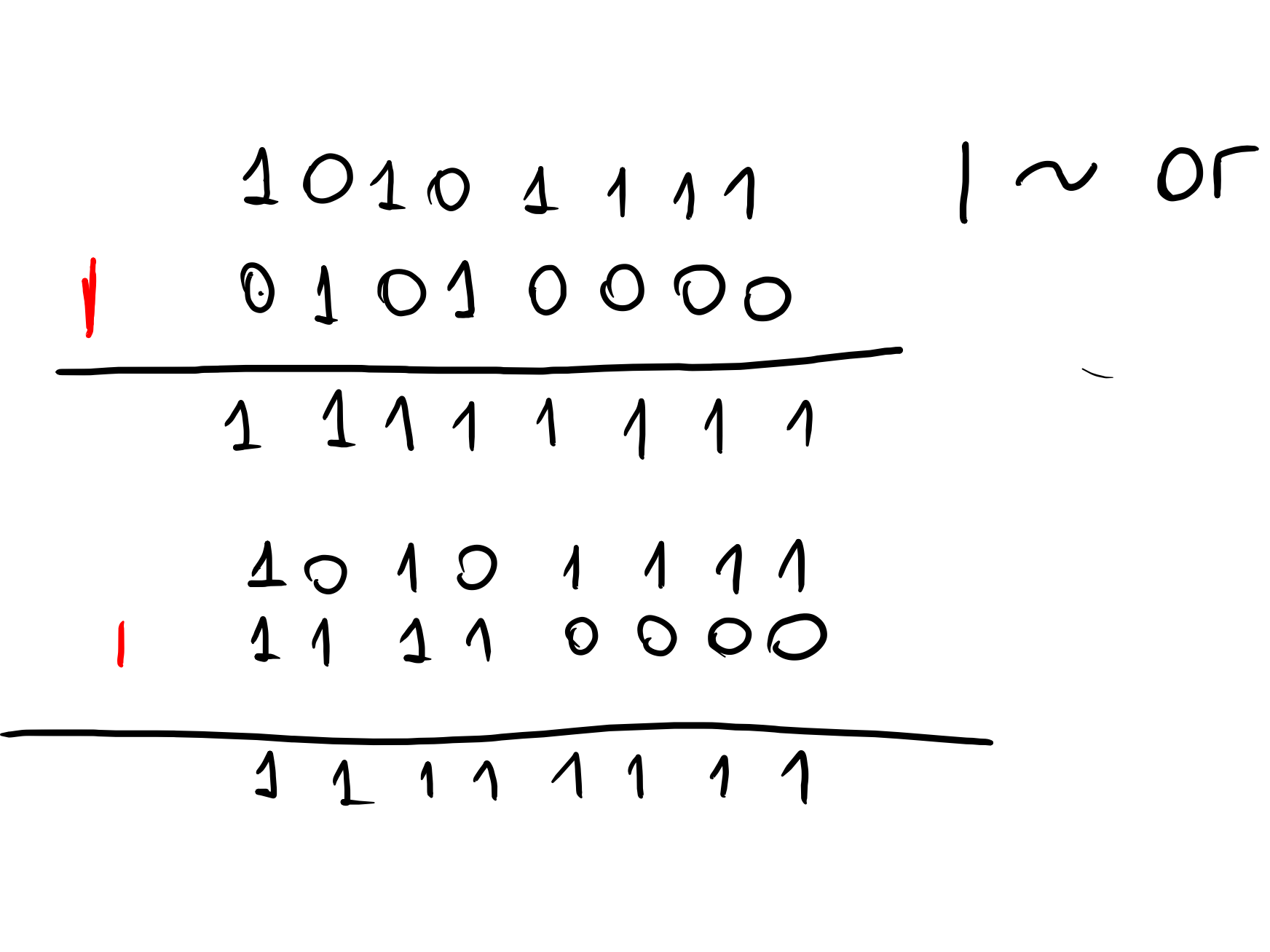
& (*, and)

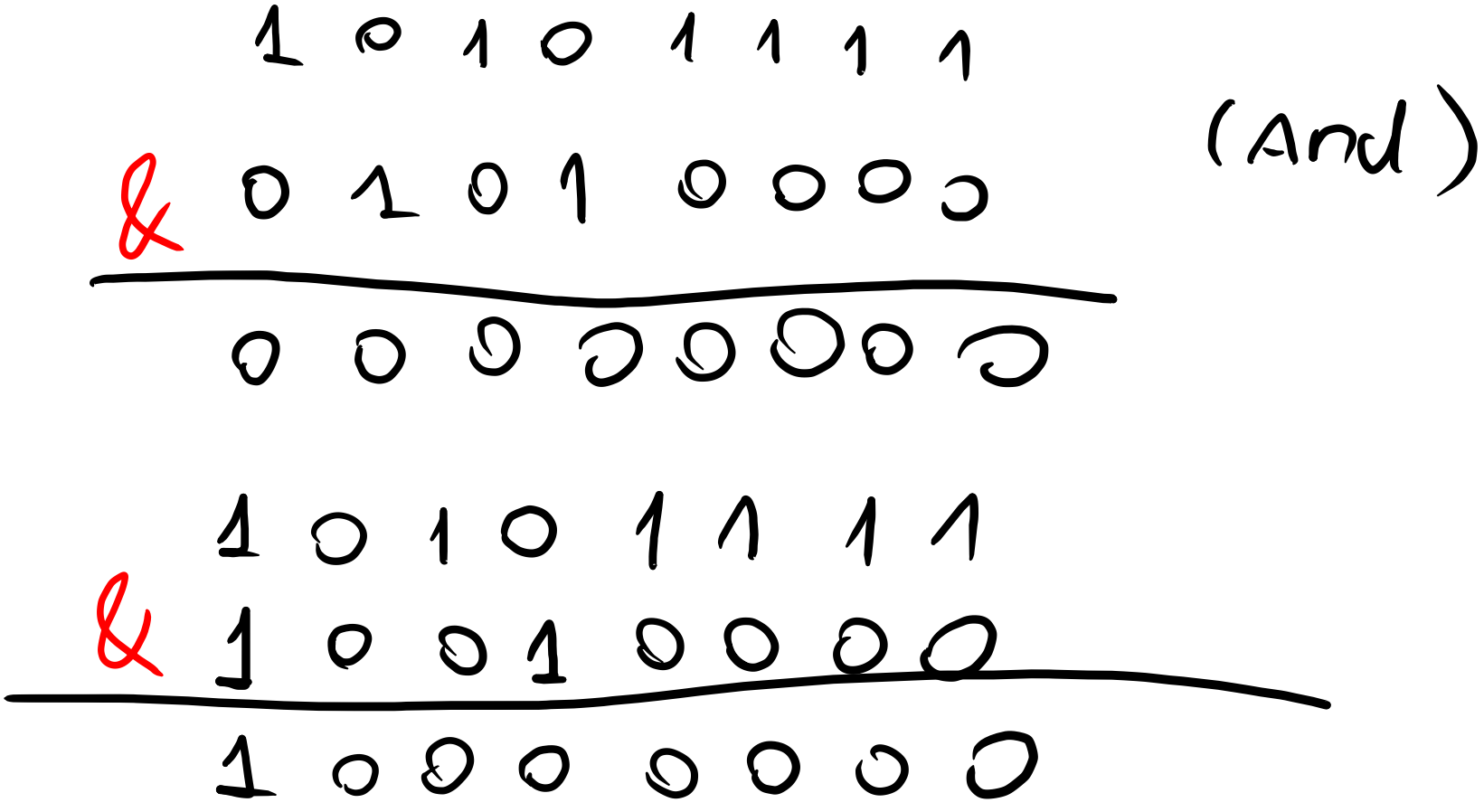
^ (XOR)

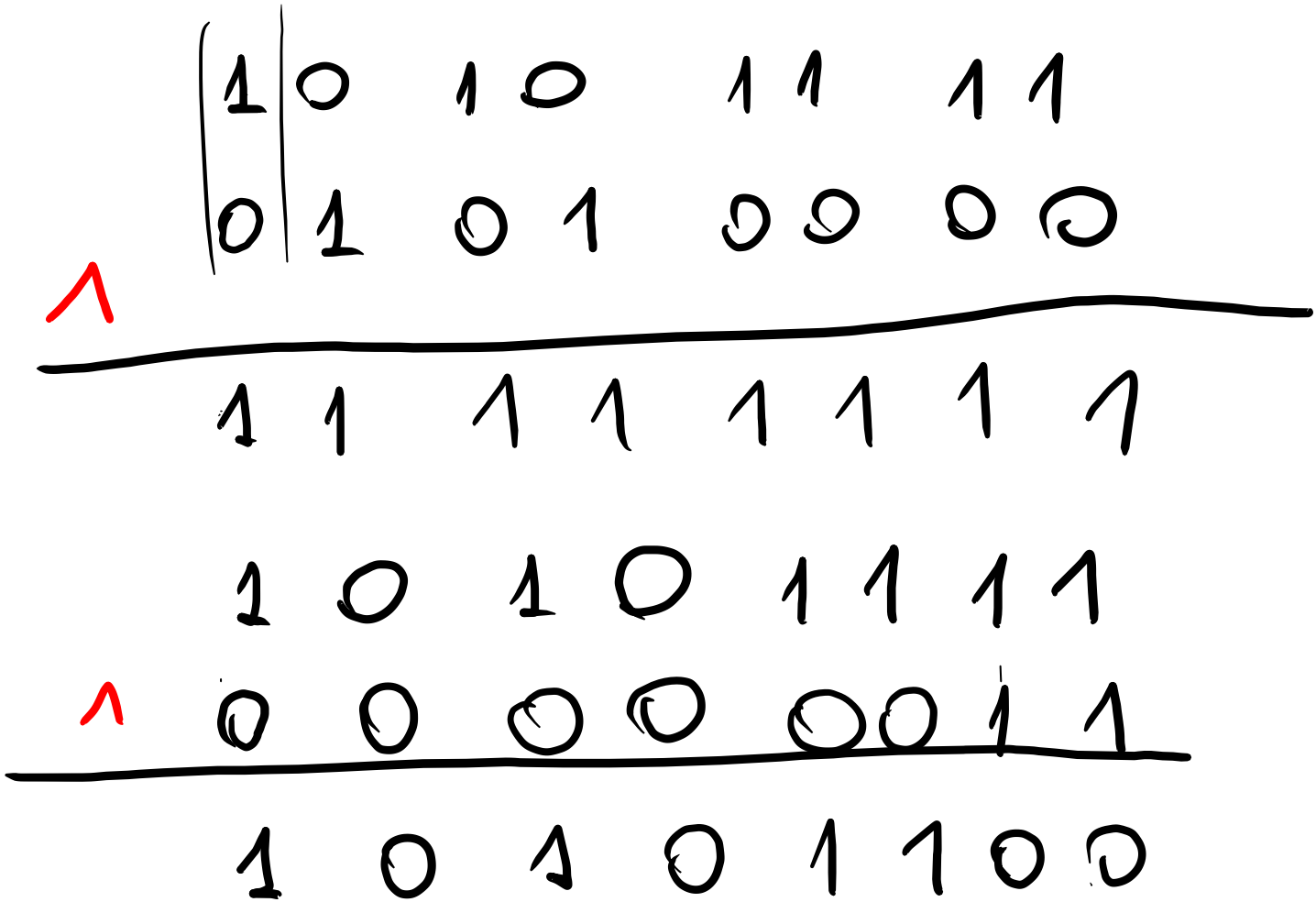
<< Desplazamiento ←

>> Desplazamiento →

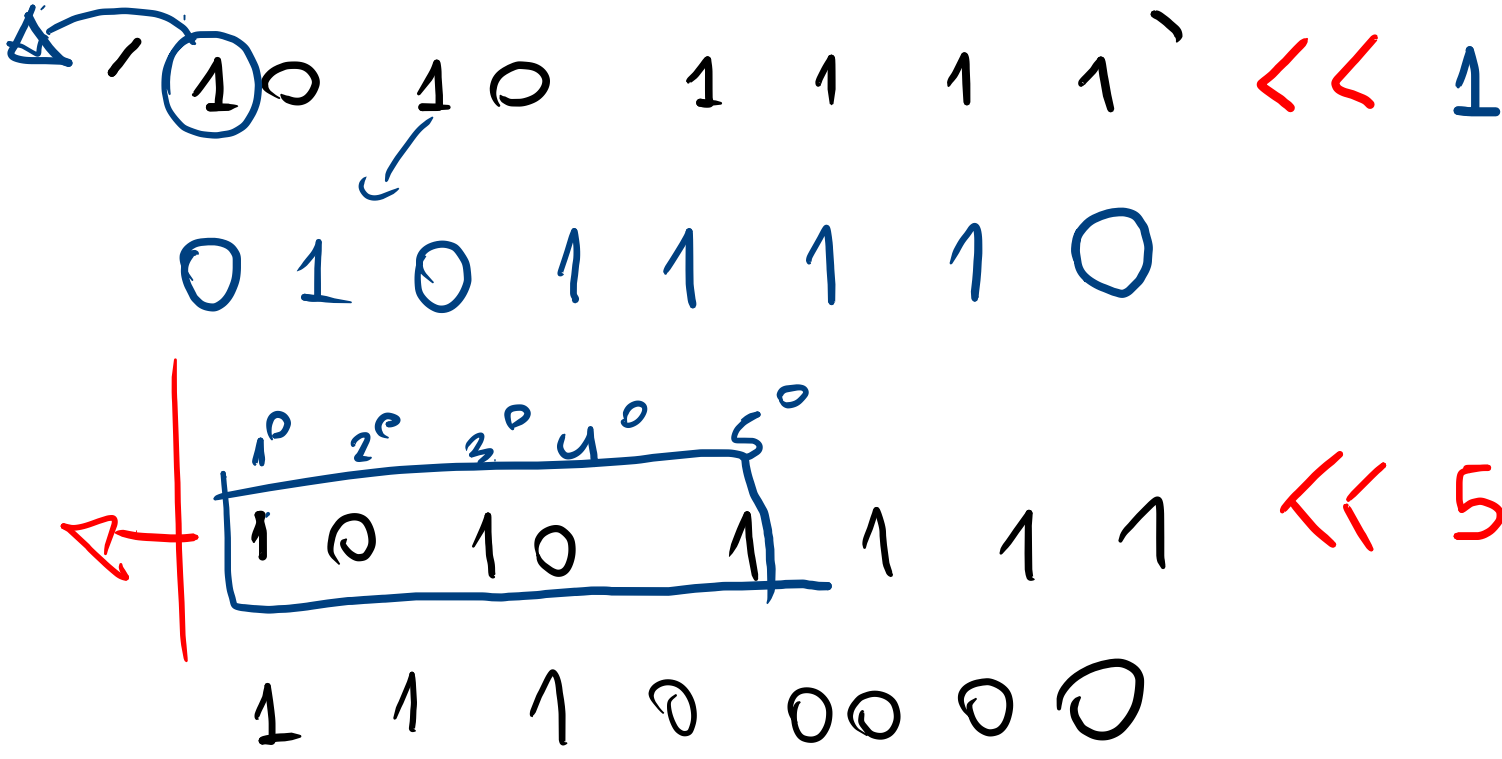
} CPU


1010 1111 | ~ or
0101 0000
1 1 1 1 1 1 1 1
1010 1111
1111 0000
1 1 1 1 1 1 1 1


1 0 1 0 1 1 1 1
& 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0
(And)
1 0 1 0 1 1 1 1
& 1 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0


1 0 1 0 1 1 1 1
0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 1
1 0 1 0 1 1 1 1
0 0 0 0 0 0 1 1
1 0 1 0 1 1 0 0

Deplazamiento (byte = 8 bit = u8)



¿ Qué bit es en posición 5?

8° 7° 6° 5° 4° 3° 2° 1°

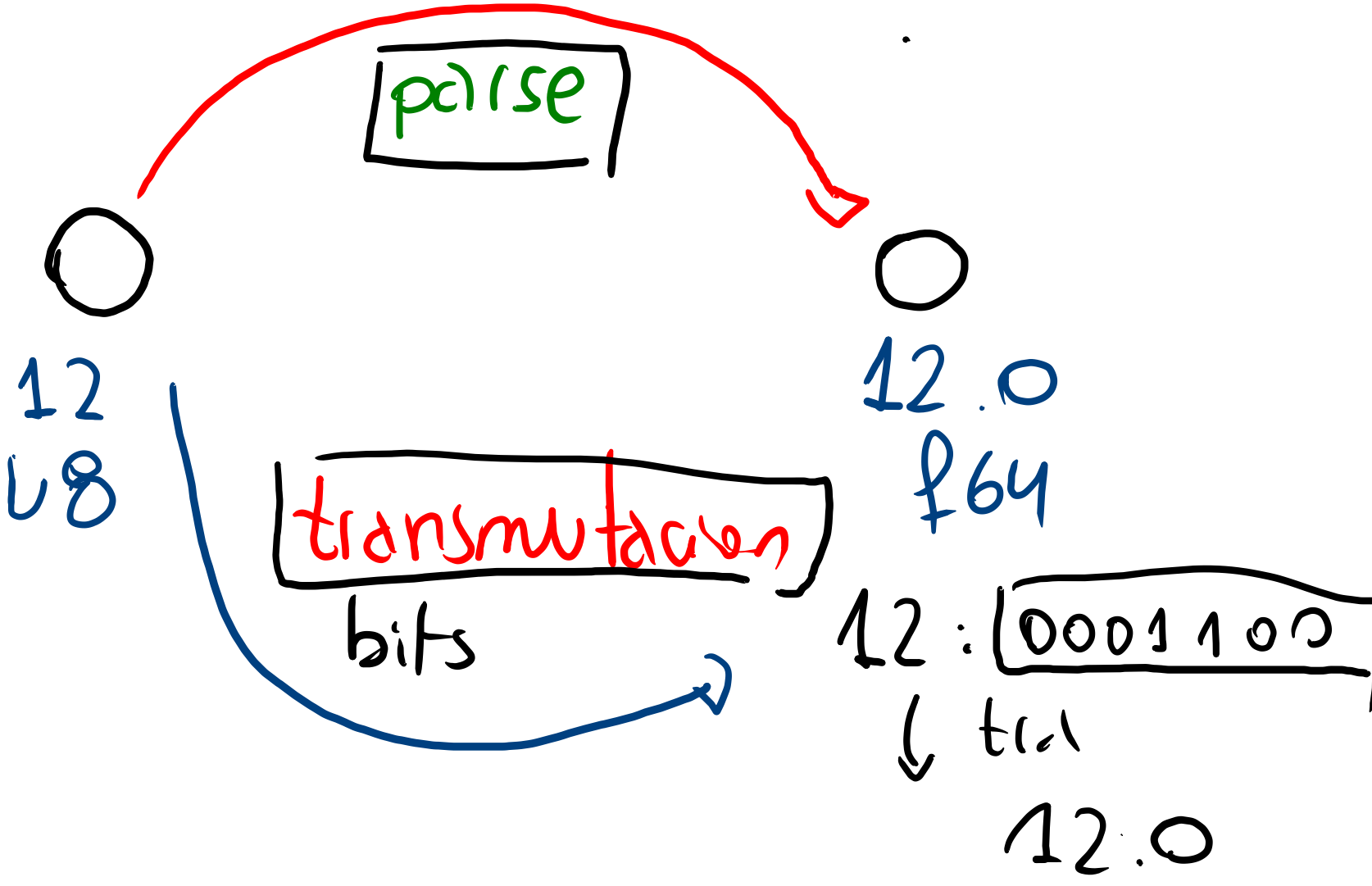
1 0 1 0 1 1 1 1 <<(?) = 3

indices: 7 6 5 4 3 2 1 0

0 1 1 1 1 0 0 0 >> ? = 7

// #2 de

0 0 0 0 0 0 0 0


parse
12
u8
12.0
f64
transmutation
bits
12: 0001100
try
12.0

mem :: transmute (Src, Dest)

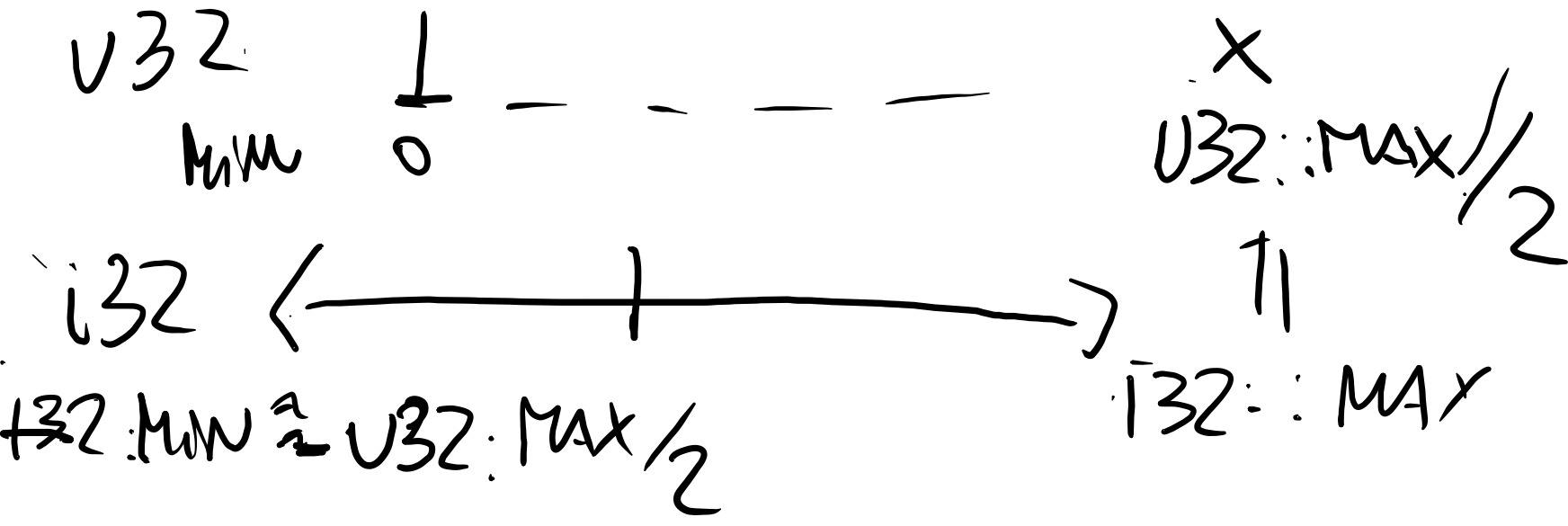
u8 ⟷ f64.

i32 ⟷ bites

(entero signado) Array bytes.

[0..255 ; 4]

32 bits → # repr.



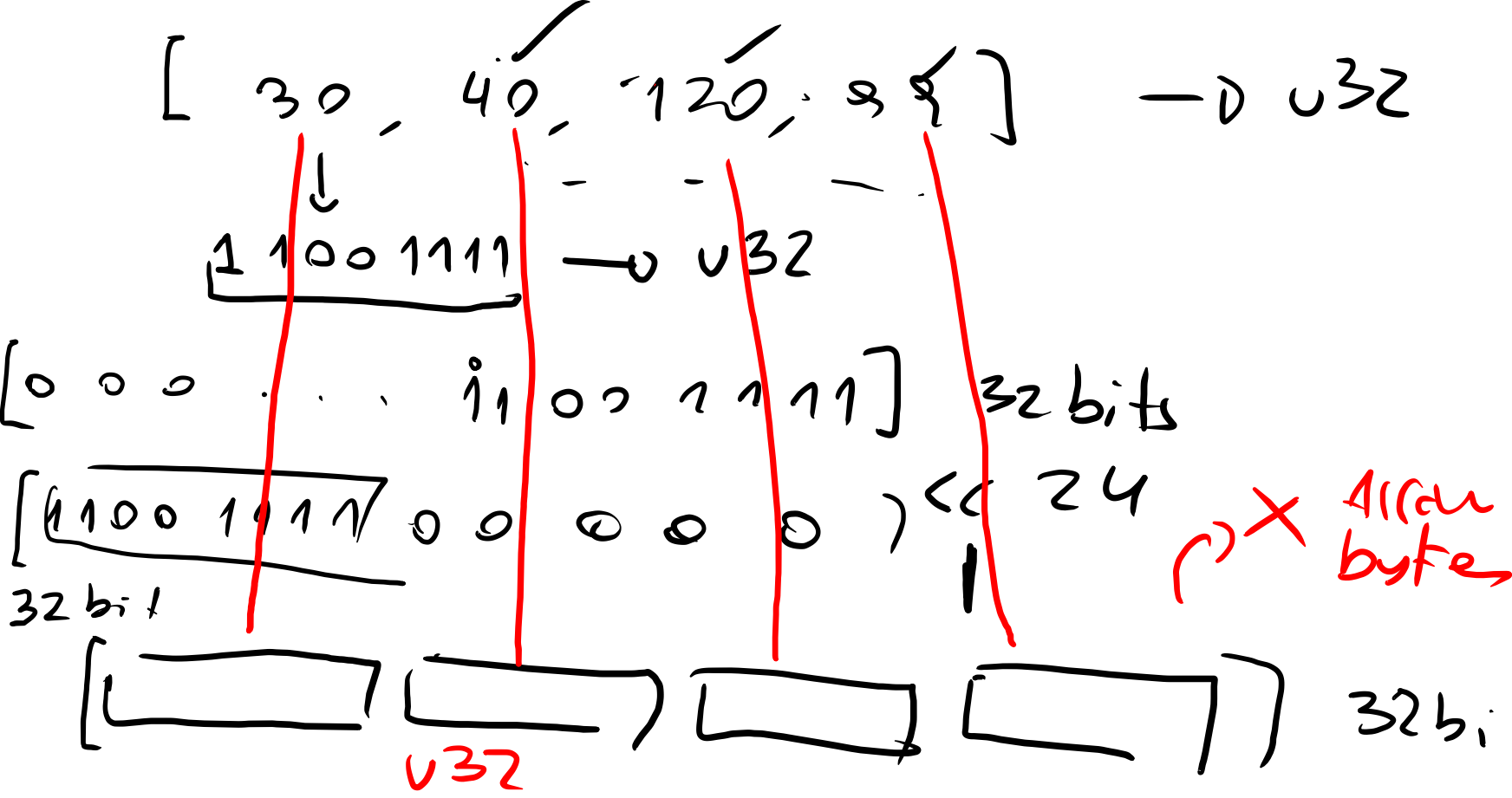
Antes de volver a convertir

saber el tipo destino

[225, 234, 255, 200] → u32 →

≠

[1..., 1...] i32 ⩽ 0


[30, 40, 120, 89] → u32
1 1 0 0 1 1 1 1 → u32
[0 0 0 . . . 1 1 0 0 1 1 1 1] 32 bits
[1 1 0 0 1 1 1 1 0 0 0 0 0 0] << 24
32 bit
x 4 (4 bytes)
u32
32 bi

conversion de floats f64

Prelude, op. funcionales

```
[a, b, c, d] -> iter().map(|v|{
                    sin(v)
                }).collect();
```

reduce → fold

u64_

[a, b, c, d] -> iter().fold(valor_inicial
0_u64,

| acc, v |

acc | fn(v)

});

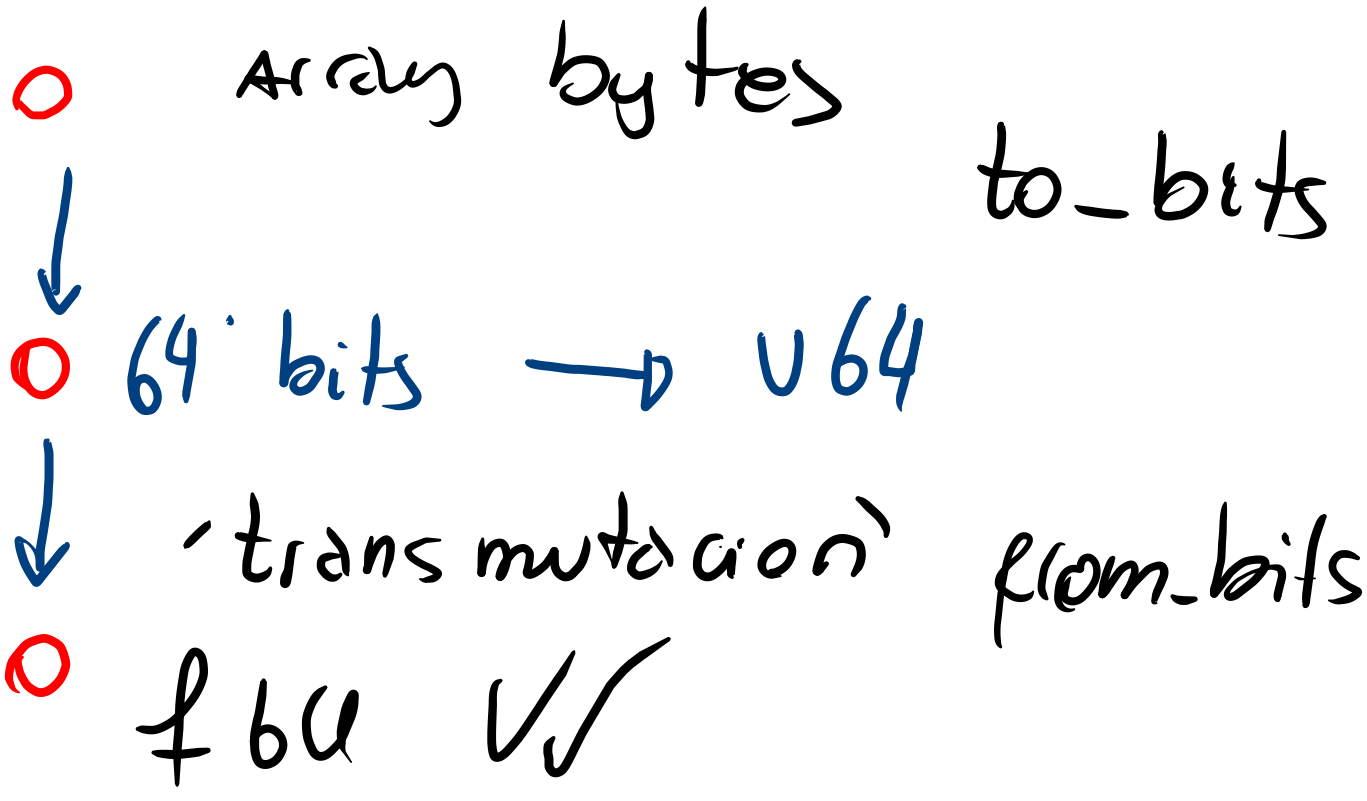
[fn(a), fn(b), fn(c), fn(d)]

fn(a) | fn(b) | fn(c) | fn(d)

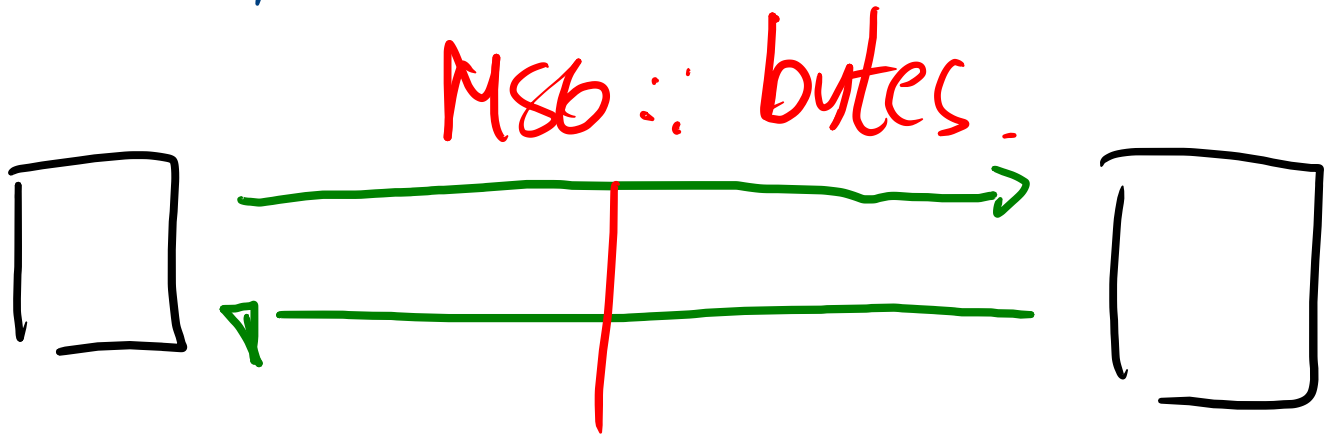
result

$f64

Rust



Socket Unix



Msg → Protocolo.

Protocolo Mensaje

4 bytes

u32

100.000.000 bytes

Zbyte

/ U16