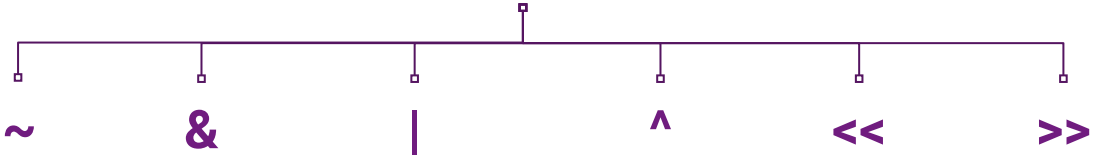
Bitwise Operators



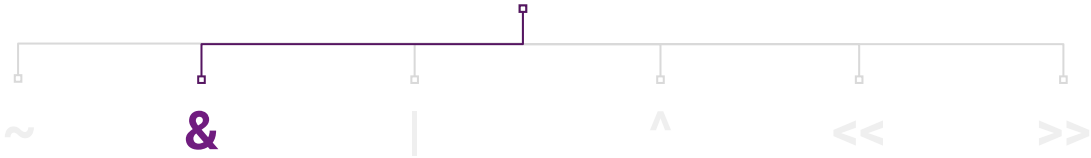
Bitwise Operators

~ & | ^ << >>

$(7)_d$ → 0 0 0 0 0 1 1 1

~7 → 1 1 1 1 1 0 0 0 =-8

Bitwise Operators



$(7)_d$ → 0 0 0 0 0 1 1 1

&

$(4)_d$ → 0 0 0 0 0 1 0 0

0 0 0 0 0 1 0 0 = 4

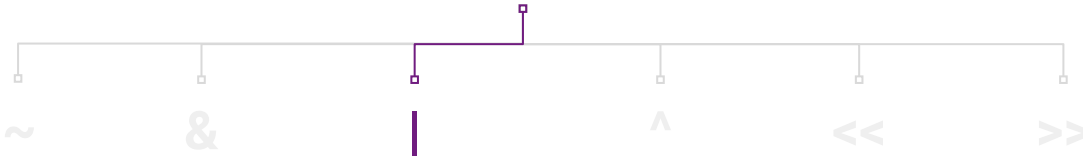
$(-7)_d$ → 1 1 1 1 1 0 0 1

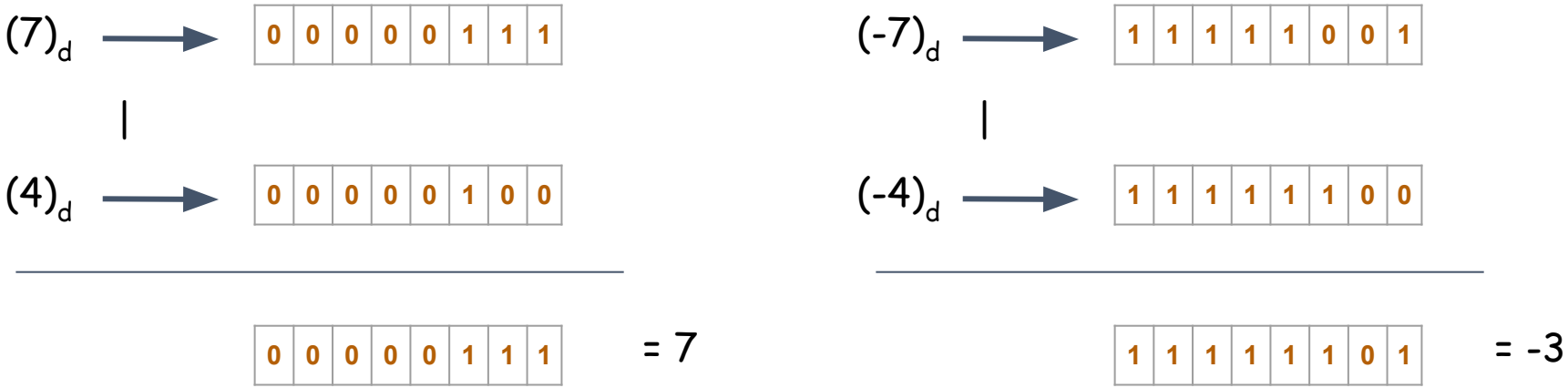
&

$(-4)_d$ → 1 1 1 1 1 1 0 0

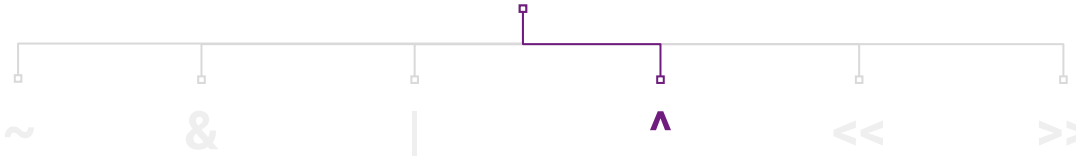
1 1 1 1 1 0 0 0 = -8

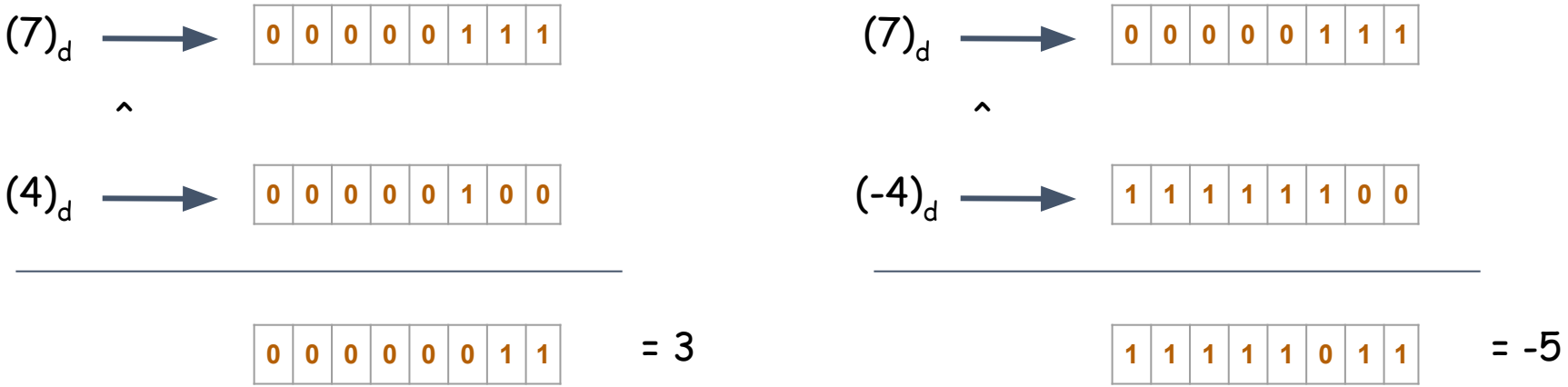
Bitwise Operators



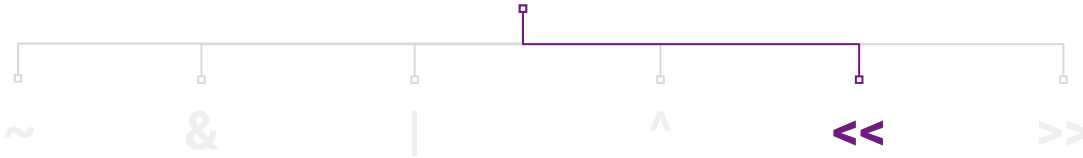


Bitwise Operators





Bitwise Operators



$(7)_d$ → | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

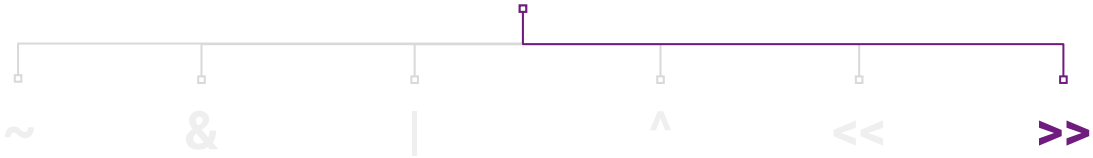
$(-4)_d$ → | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

---

7 <<2 → | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | = 28

-4 <<2 → | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | = -16

Bitwise Operators



Pay Attention

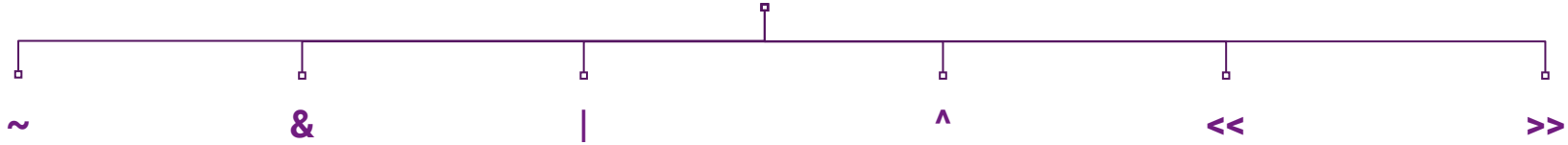
$(7)_d$ → 0 0 0 0 0 1 1 1

7 >>2 → 0 0 0 0 0 0 0 1 = 1

$(-4)_d$ → 1 1 1 1 1 1 0 0

-4 >>2 → 1 1 1 1 1 1 1 1 = -1

Bitwise Operators



$(7)_d$ = 0 0 0 0 0 1 1 1

~7 = 1 1 1 1 1 0 0 0 = -8

$(7)_d$ = 0 0 0 0 0 1 1 1

&

$(4)_d$ = 0 0 0 0 0 1 0 0

0 0 0 0 0 1 0 0 = 4

$(7)_d$ = 0 0 0 0 0 1 1 1

/

$(4)_d$ = 0 0 0 0 0 1 0 0

0 0 0 0 0 1 1 1 = 7

$(7)_d$ = 0 0 0 0 0 1 1 1

^

$(4)_d$ = 0 0 0 0 0 1 0 0

0 0 0 0 0 0 1 1 = 3

$(7)_d$ = 0 0 0 0 0 1 1 1

7 <<2 = 0 0 0 1 1 1 0 0 = 28

$(7)_d$ = 0 0 0 0 0 1 1 1

7 >>2 = 0 0 0 0 0 0 0 1 = 1

$(-4)_d$ = 1 1 1 1 1 1 0 0

-4 <<2 = 1 1 1 1 0 0 0 0 = -16

$(-4)_d$ = 1 1 1 1 1 1 0 0

-4 >>2 = 1 1 1 1 1 1 1 1 = -1

Pay Attention

Masks

(Defines which bits you want to keep)

All zeros

mask = 0

All ones

mask = ~0

one 1 at location (i)

mask = 1<<i

one 0 at location (i)

mask = ~(1<<i)

(i) bits with 1s at the beginning

mask = (1<<i) -1

(i) bits with 1s at the end

mask = ~((1<<(n-i)) -1)

(i) bits with 1s in the middle

mask = high - low

Masks

$1 << i$	⇢	2^i
$N \& (1 << i)$	⇢	Check the i^{th} bit (1 or 0)
$N \mid (1 << i)$	⇢	Set the i^{th} bit to (make it 1)
$N \& \sim(1 << i)$	⇢	Clear the i^{th} bit (make it 0)
$N \& (-N)$	⇢	The rightmost 1
$\sim N + 1$	⇢	The 2's complement
$(1 << i) - 1 == N$	⇢	Check if all bits in N are $(1)_s$

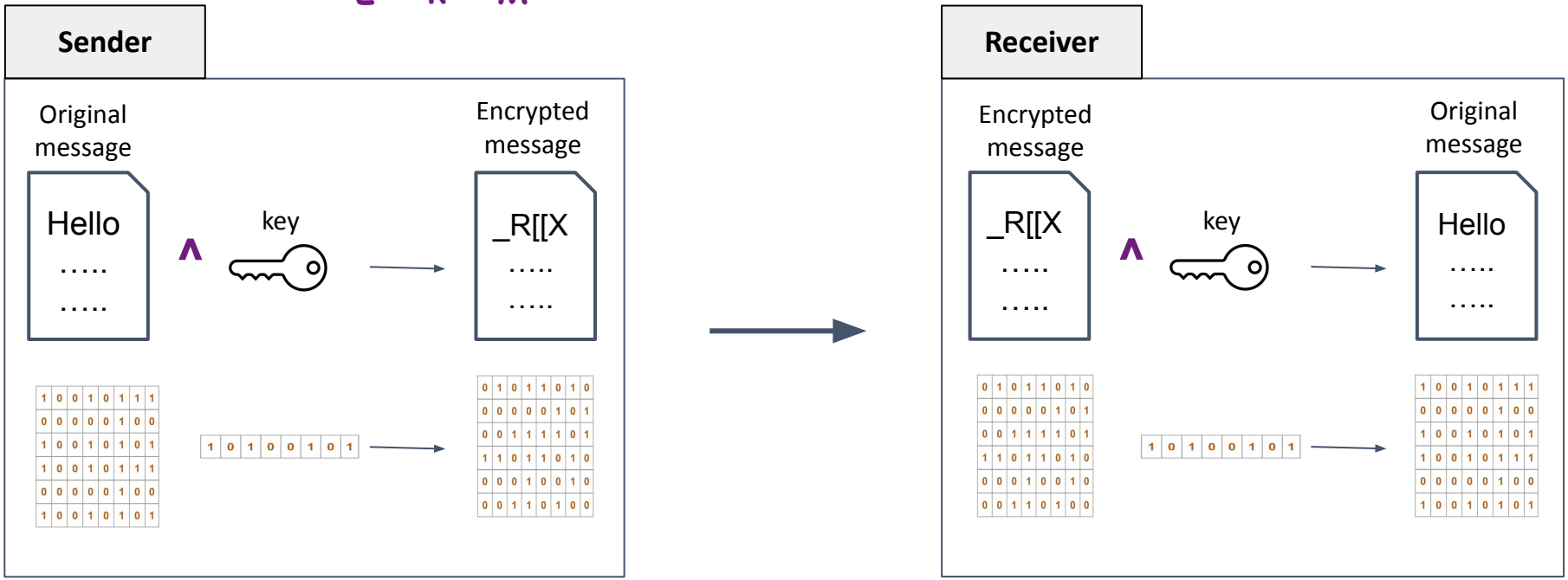
Tricks

`1 << i`	→	2^i
`N & (1 << i)`	→	Check the i^{th} bit (1 or 0)
`N | (1<<i)`	→	set the i^{th} bit (make it 1)
`N & ~(1<<i)`	→	clear the i^{th} bit (make it 0)
`N & (-N)`	→	The rightmost 1
`~N + 1`	→	The 2's complement

Bitwise Operators Applications

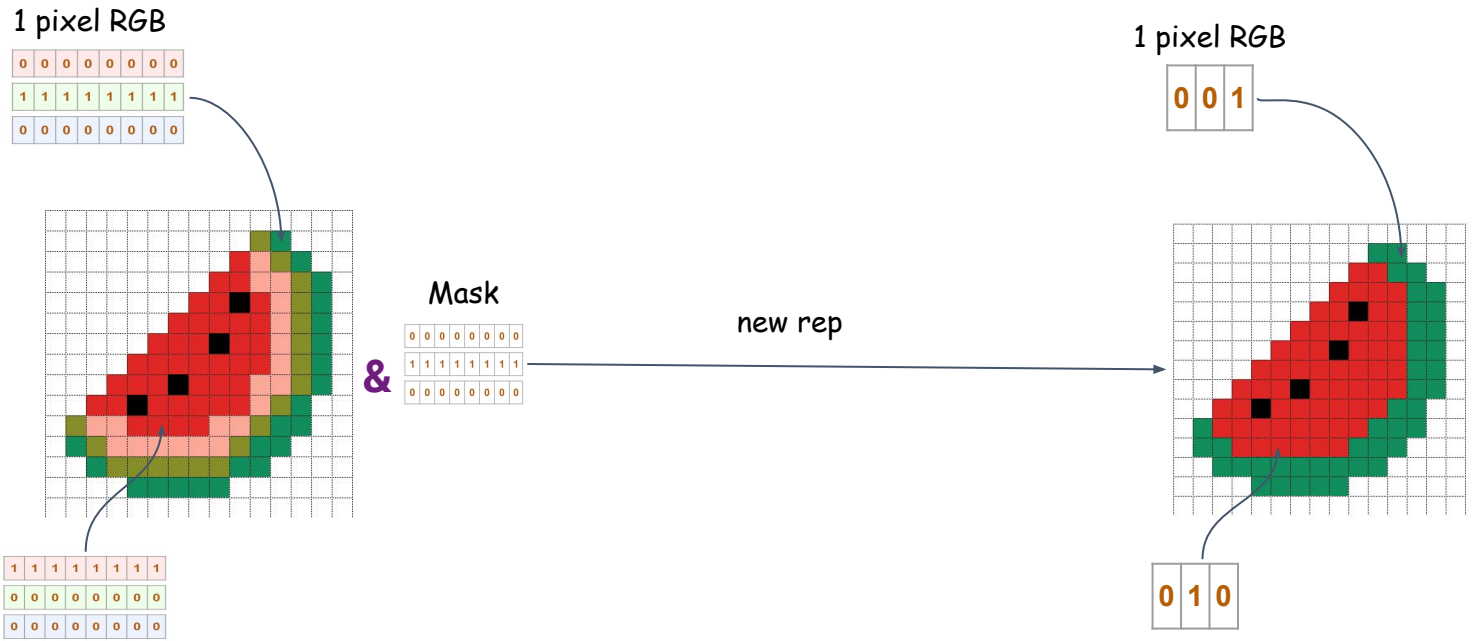
Data Encryption | Data Compression | Faster Algorithms

```
M ^ k = E
E ^ k = M
```



Bitwise Operators Applications

Data Encryption | **Data Compression** | Faster Algorithms



15 X 15 X 3 X 8 = 5400 bits
= 675 bytes

15 X 15 X 3 = 675 bits
= 85 bytes

Bitwise Operators Applications

Data Encryption

Data Compression

Faster Algorithms

Traditional Method

```
bool isPowerOfTwo(int x){
  if(x == 0){
      return false;
  }else{
    while(x % 2 == 0) x /= 2;
    return (x == 1);
  }
}
```

Using bitwise Method

```
bool isPowerOfTwo(int x){
  return (x && !(x & (x - 1)));
}
```