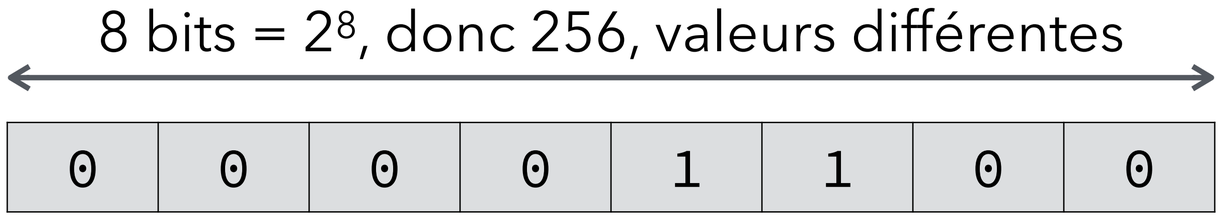
Types entiers

Pratique de la programmation orientée-objet

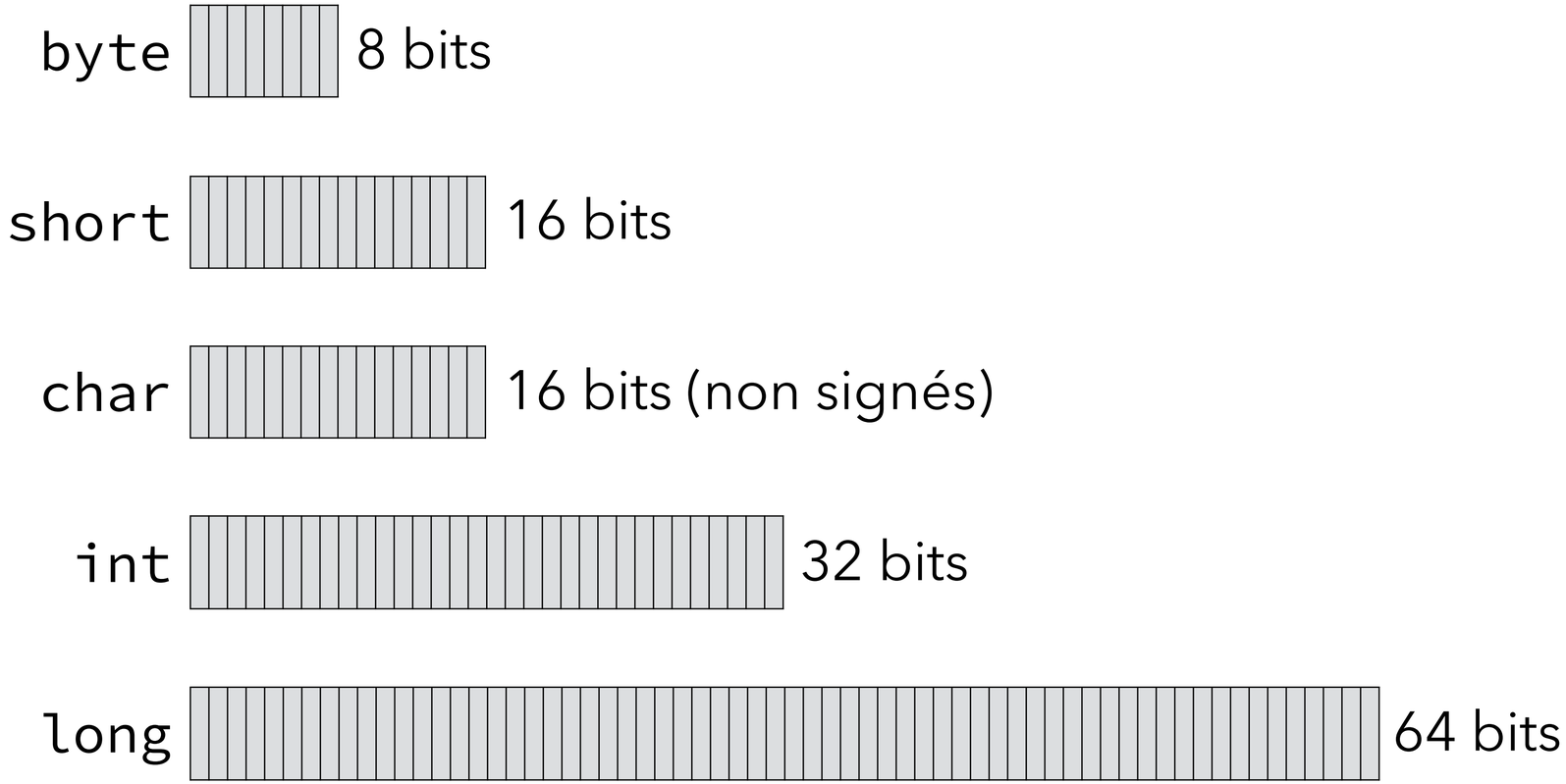
Michel Schinz

« Entier » = vecteur de bits



bit (***bi****nary digi****t***) = chiffre binaire : 0 ou 1

« Entiers » Java



Interprétation non signée

Un vecteur de bits peut être interprété comme un entier (positif) en binaire (base 2) :

poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
bit	1	0	0	0	1	1	0	0

bit de poids (le plus) fort, MSB

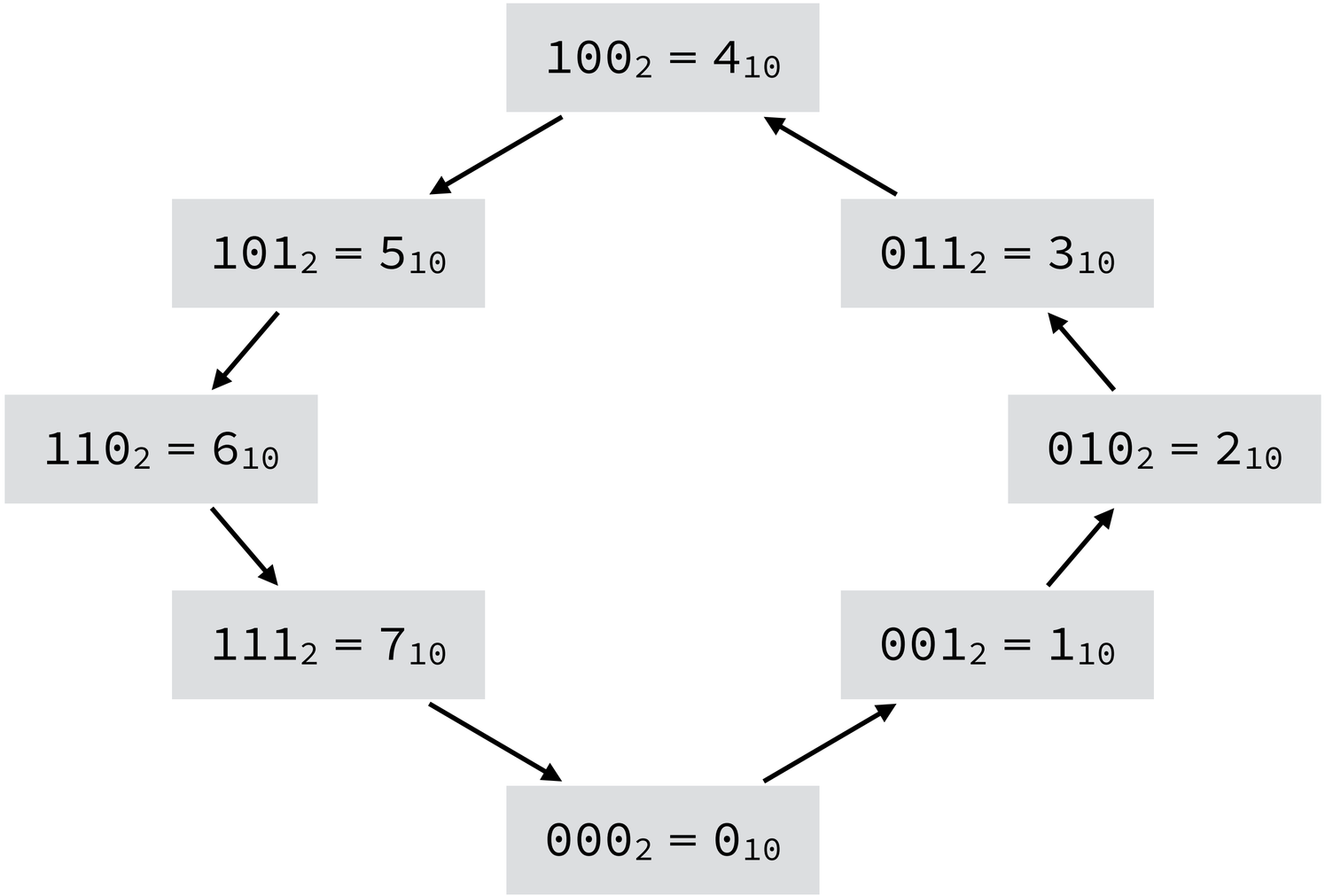
bit de poids (le plus) faible, LSB

Interprété comme un entier, ce vecteur de 8 bits vaut :

$2^7 + 2^3 + 2^2 = 128 + 8 + 4 = 140$

Interprétation non signée

Valeurs positives seulement (en Java, `char` est interprété ainsi.)



Interprétation signée

Un vecteur de bits peut aussi être interprété comme un entier **signé** en binaire, en inversant le signe du poids fort :

poids	$-(2^7)$	2^6	2^5	2^4	2^3	2^2	2^1	2^0
bit	1	0	0	0	1	1	0	0

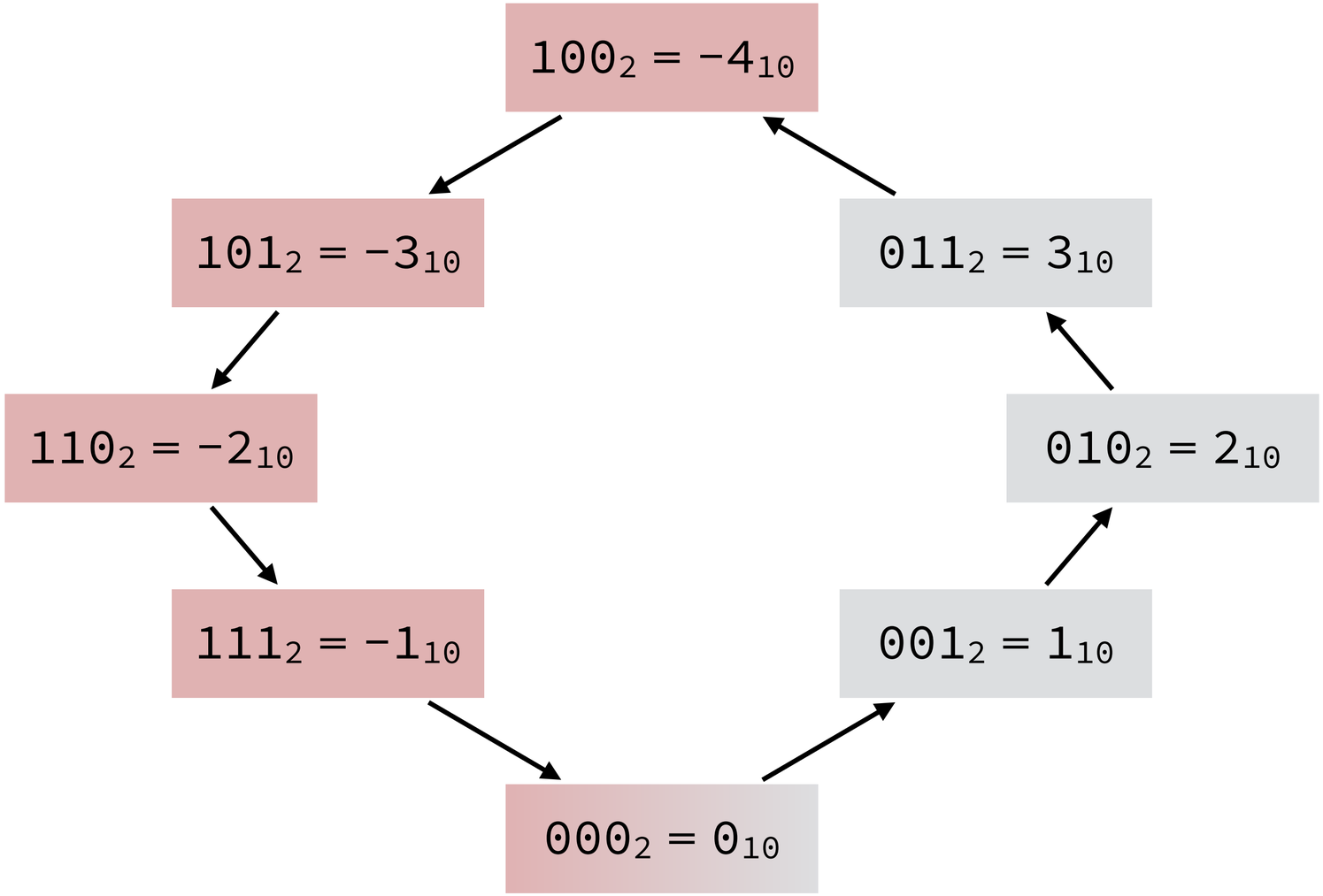
Cette interprétation s'appelle le **complément à deux** (*two's complement*).

Interprété comme un entier signé (en complément à deux), ce vecteur de 8 bits vaut :

$-(2^7) + 2^3 + 2^2 = -128 + 8 + 4 = -116$

Interprétation signée

Complément à deux (en Java : `byte`, `short`, `int` et `long`.)



Dépassements de capacité

Addition

+	-4	-3	-2	-1	0	1	2	3
-4	0	1	2	3	-4	-3	-2	-1
-3	1	2	3	-4	-3	-2	-1	0
-2	2	3	-4	-3	-2	-1	0	1
-1	3	-4	-3	-2	-1	0	1	2
0	-4	-3	-2	-1	0	1	2	3
1	-3	-2	-1	0	1	2	3	-4
2	-2	-1	0	1	2	3	-4	-3
3	-1	0	1	2	3	-4	-3	-2

Dépassement de capacité dans ~25% des cas

Dépassements de capacité

Soustraction

–	-4	-3	-2	-1	0	1	2	3
-4	0	-1	-2	-3	-4	3	2	1
-3	1	0	-1	-2	-3	-4	3	2
-2	2	1	0	-1	-2	-3	-4	3
-1	3	2	1	0	-1	-2	-3	-4
0	-4	3	2	1	0	-1	-2	-3
1	-3	-4	3	2	1	0	-1	-2
2	-2	-3	-4	3	2	1	0	-1
3	-1	-2	-3	-4	3	2	1	0

Dépassement de capacité dans ~25% des cas

Dépassements de capacité

Multiplication

*	-4	-3	-2	-1	0	1	2	3
-4	0	-4	0	-4	0	-4	0	-4
-3	-4	1	-2	3	0	-3	2	-1
-2	0	-2	-4	2	0	-2	-4	2
-1	-4	3	2	1	0	-1	-2	-3
0	0	0	0	0	0	0	0	0
1	-4	-3	-2	-1	0	1	2	3
2	0	2	-4	-2	0	2	-4	-2
3	-4	-1	2	-3	0	3	-2	1

Dépassement de capacité dans ~40% des cas

Dépassements de capacité

Division

/	-4	-3	-2	-1	0	1	2	3
-4	1	1	2	-4	–	-4	-2	-1
-3	0	1	1	3	–	-3	-1	-1
-2	0	0	1	2	–	-2	-1	0
-1	0	0	0	1	–	-1	0	0
0	0	0	0	0	–	0	0	0
1	0	0	0	-1	–	1	0	0
2	0	0	-1	-2	–	2	1	0
3	0	-1	-1	-3	–	3	1	1

Dépassement de capacité dans ~2% des cas

Types « entiers » Java

byte (8 bits)

00001100	12
11110100	-12

short (16 bits)

0000000000001100	12
1111111111110100	-12

char (16 bits)

0000000000001100	12
1111111111110100	65524

int (32 bits)

00000000000000000000000000001100	12
11111111111111111111111111110100	-12

+ **long** (64 bits)

Valeurs limites

Type	Minimum	Maximum
`byte`	-128	127
`short`	-32 768	32 767
`char`	0	65 535
`int`	-2 147 483 648	2 147 483 647
`long`	-9 223 372 036 854 775 808	9 223 372 036 854 775 807

Opérateurs bit à bit

complément (~)

	~
0	1
1	0

~ 1 1 1 1 0 0 0 0

=

0 0 0 0 1 1 1 1

Opérateurs bit à bit

et (&)

&	0	1
0	0	0
1	0	1

1	1	1	1	0	0	0	0

&

0	0	1	1	1	1	0	0

=

0	0	1	1	0	0	0	0

ou inclusif (|)

|	0	1
0	0	1
1	1	1

1	1	1	1	0	0	0	0

|

0	0	1	1	1	1	0	0

=

1	1	1	1	1	1	0	0

ou exclusif (^)

^	0	1
0	0	1
1	1	0

1	1	1	1	0	0	0	0

^

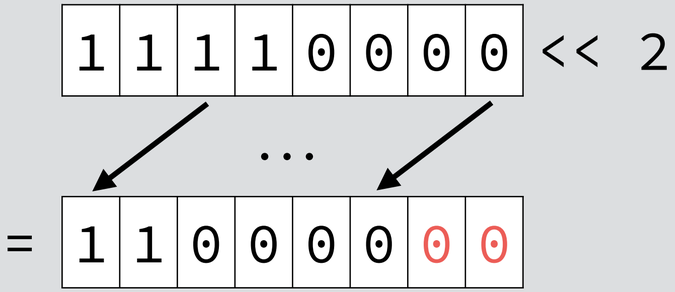
0	0	1	1	1	1	0	0

=

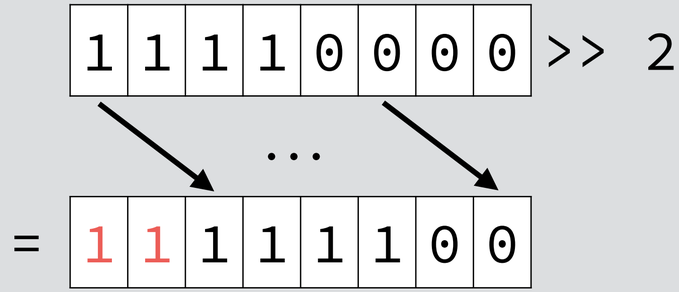
1	1	0	0	1	1	0	0

Décalages

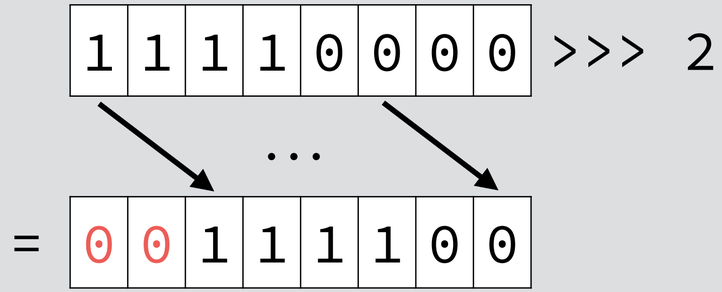
gauche (<<)



droite (>>)



droite logique (>>>)

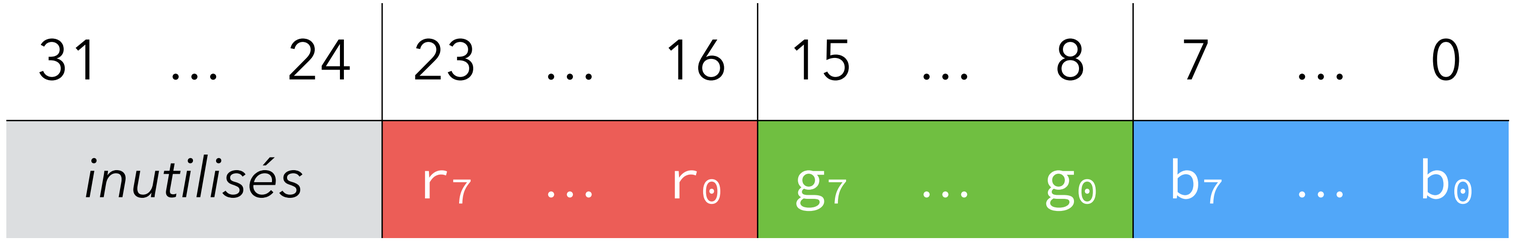


Exemple : couleurs empaquetées

Couleur empaquetée

Une couleur est représentée par trois composantes : R (rouge), G (vert) et B (bleu).
Ces composantes peuvent être empaquetées dans un seul entier de 32 bits :

- chaque composante est un entier de 8 bits,
- ces 3×8 = 24 bits sont placées côte à côte dans les 24 bits de poids faible de l'entier 32 bits.



Notation des entiers Java

```java
// Notation binaire
int x = 0b10010001101001010101111001101;
// Notation hexadécimale (base 16)
int x = 0x1234ABCD;
// Notation décimale
int x = 305441741;
```

En hexadécimal : 1 chiffre = 4 bits

0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1
1				2				3				4				A				B				C				D			

La notation hexadécimale convient bien aux couleurs empaquetées.
Par exemple `0xFF_00_00` représente un rouge pur (255,0,0).

Extraction du vert

```
int rgb = …;
int g = (rgb >> 8) & 0xFF;
```

`rgb`

31…24	23…16	15…8	7…0
???	$r_7 \ldots r_0$	$g_7 \ldots g_0$	$b_7 \ldots b_0$

`rgb >> 8`

31…24	23…16	15…8	7…0
???	???	$r_7 \ldots r_0$	$g_7 \ldots g_0$

`(rgb >> 8) & 0xFF`

31…24	23…16	15…8	7…0
0…0	0…0	0…0	$g_7 \ldots g_0$