Manipuler efficacement les motifs binaires

Plan:

- *Table des classes de priorités des opérateurs du C++*
- *Rappel des propriétés des opérateurs logiques*
- *les opérateurs bit à bit : & ¦ ^ ~ << >>*

niveaux de priorités des opérateurs

Plus de 50 opérateurs
sur 17 niveaux de priorités

Associativité: pour les opérateurs de même priorité

Gauche->Droite / Left-to-Right
Droite->Gauche / Right-to-Left

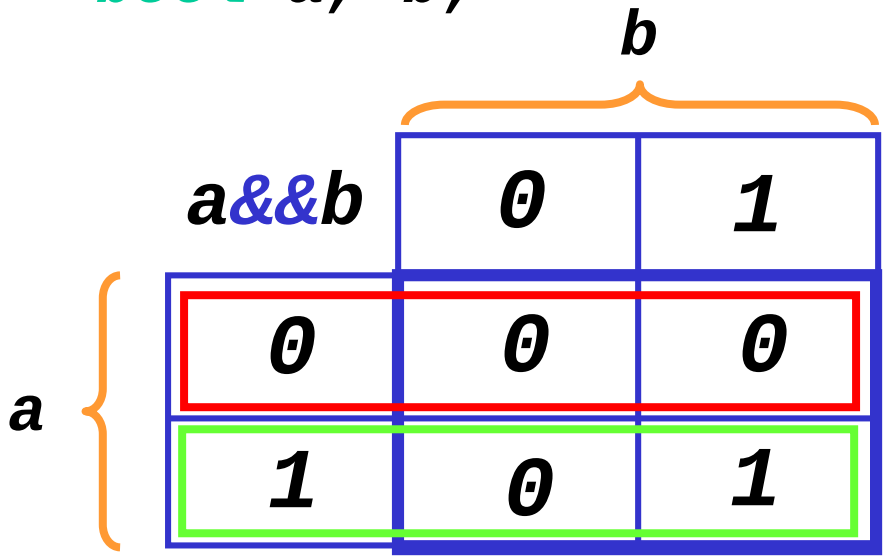
Opérateurs bit à bit

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++ a--`	Suffix/postfix increment and decrement	
	`type``()` *`type`*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`. ->`	Member access	
3	`++a --a`	Prefix increment and decrement	Right-to-left
	`+a -a`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(`*`type`*`)`	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`co_await`	await-expression (C++20)	
	`new new[]`	Dynamic memory allocation	
	`delete delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer-to-member	Left-to-right
5	`a*b a/b a%b`	Multiplication, division, and remainder	
6	`a+b a-b`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
10	`== !=`	For relational operators = and ≠ respectively	
11	`&`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left
	`throw`	throw operator	
	`co_yield`	yield-expression (C++20)	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+= -=`	Compound assignment by sum and difference	
	`*= /= %=`	Compound assignment by product, quotient, and remainder	
	`<<= >>=`	Compound assignment by bitwise left shift and right shift	
	`&= ^= |=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right

cppreference.com

Rappel : Le ET logique

```
bool a, b;
```




```
0 && b =  0
1 && b =  b
```

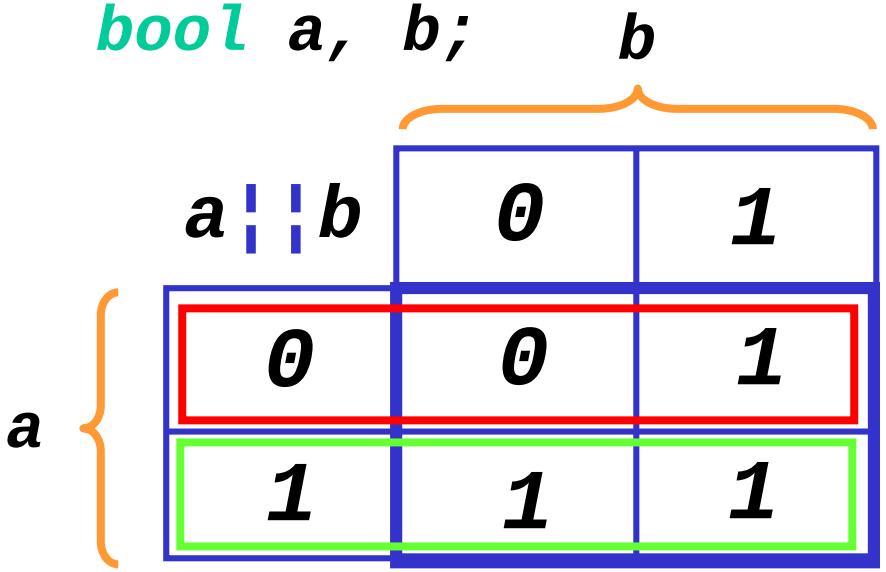
ET LOGIQUE

Evaluation paresseuse du ET logique:

*L'opérande **droit** n'est PAS évalué*
*si celui de **gauche** vaut 0*

Rappel: Le OU logique

```
bool a, b;
```



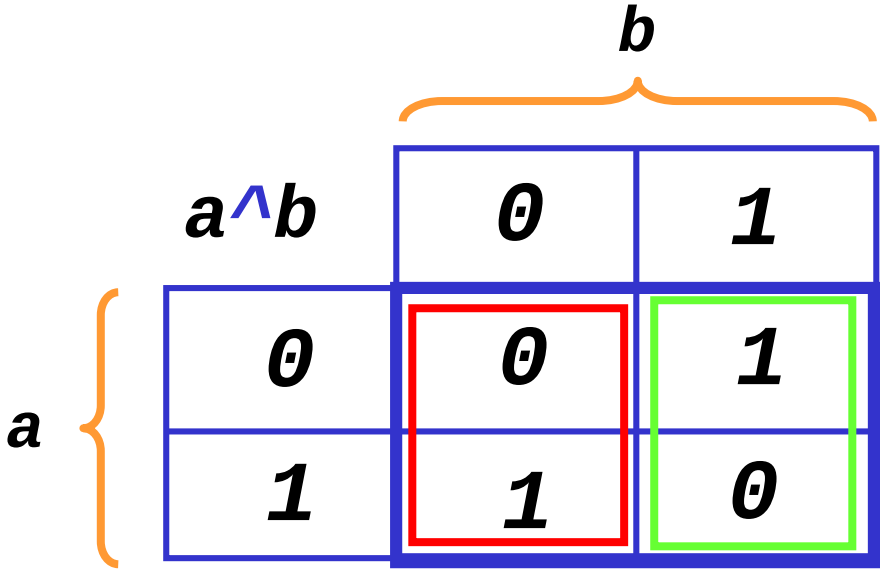

```
0 || b =  b
1 || b =  1
```

OU LOGIQUE

Evaluation paresseuse du OU logique:

*L'opérande **droit** n'est PAS évalué*
*si celui de **gauche** vaut 1*

Rappel : Le OU Exclusif



OU Exclusif

a^0 = a

a^1 = Négation de a

Exemples avec les opérateurs bit à bit

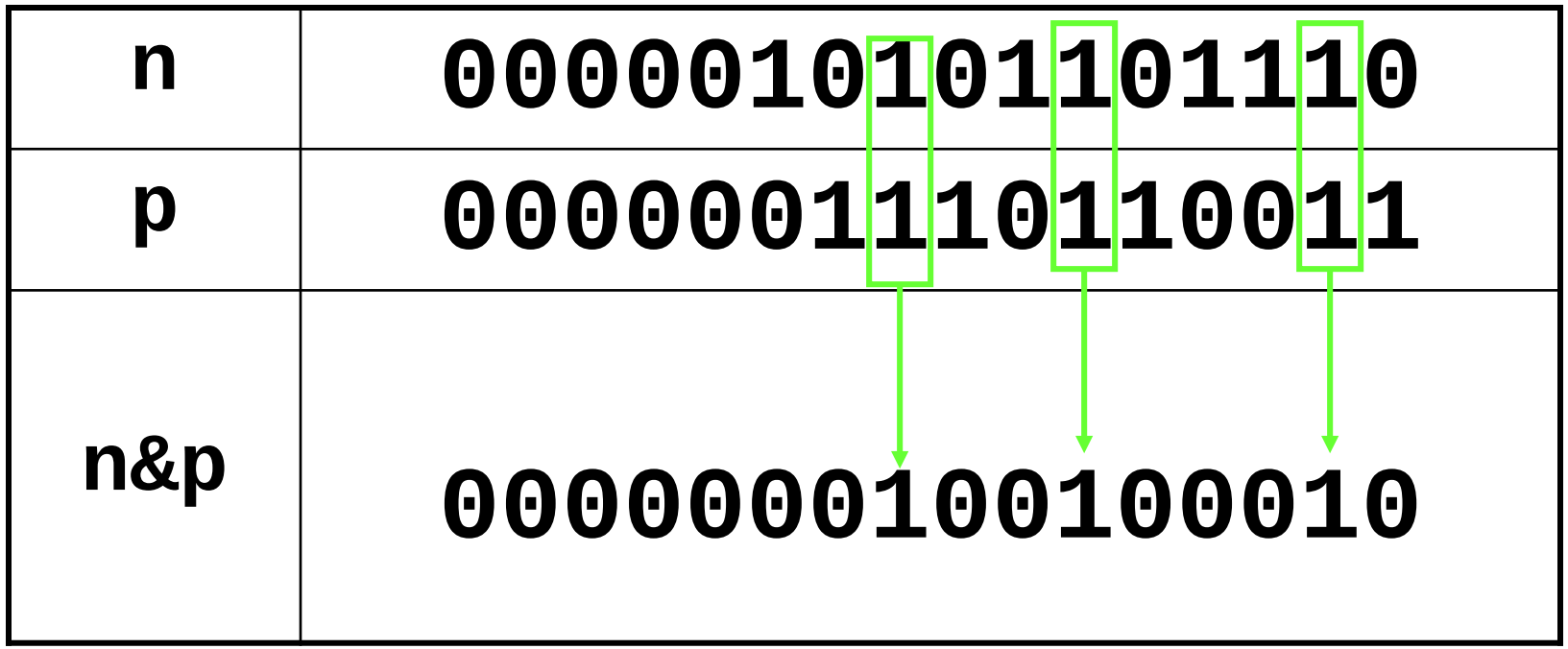
soit n et p deux variables de type entier

n	**0000010101101110**
p	**0000001110110011**

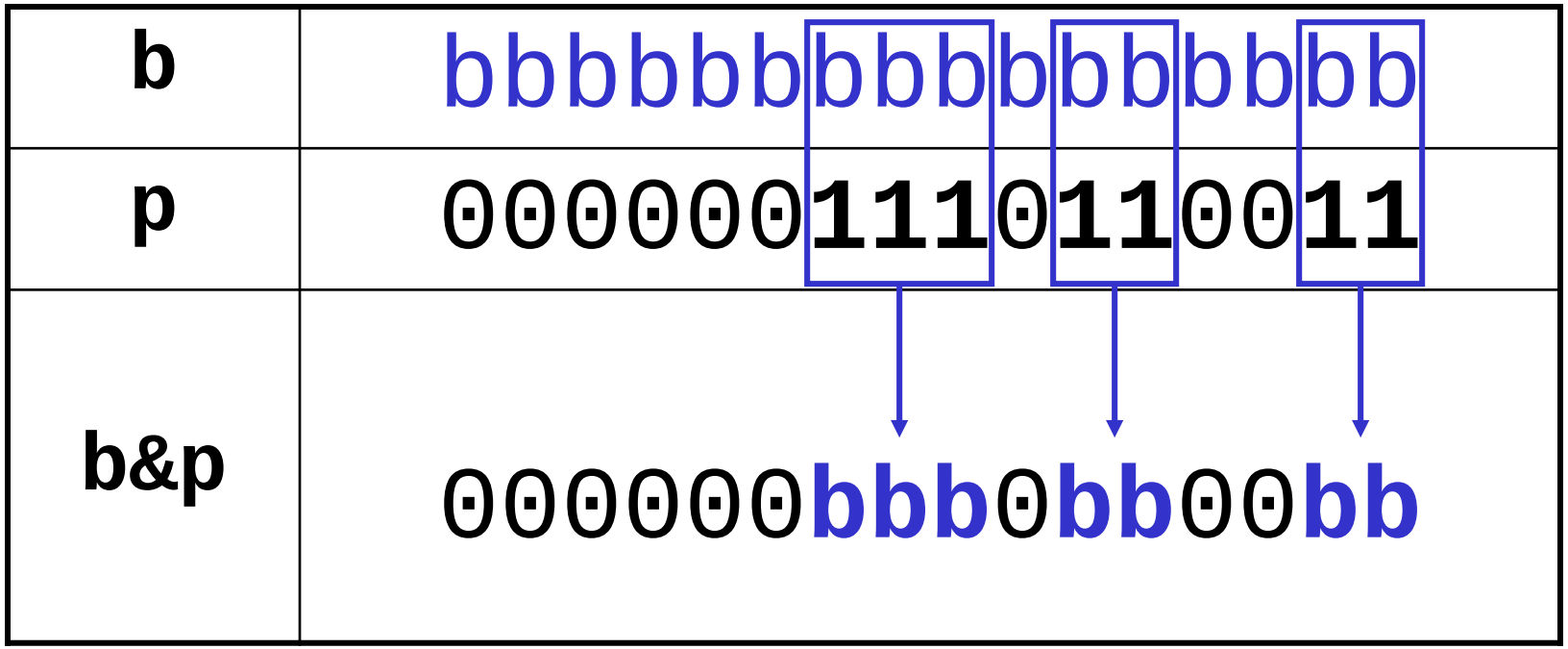
ET-Logique n & p

n	0000010101101110
p	0000001110110011
n&p	0

n & p


n
0000010101101110
p
0000001110110011
n&p
0000000100100010

b & 1 donne b
b & 0 donne 0



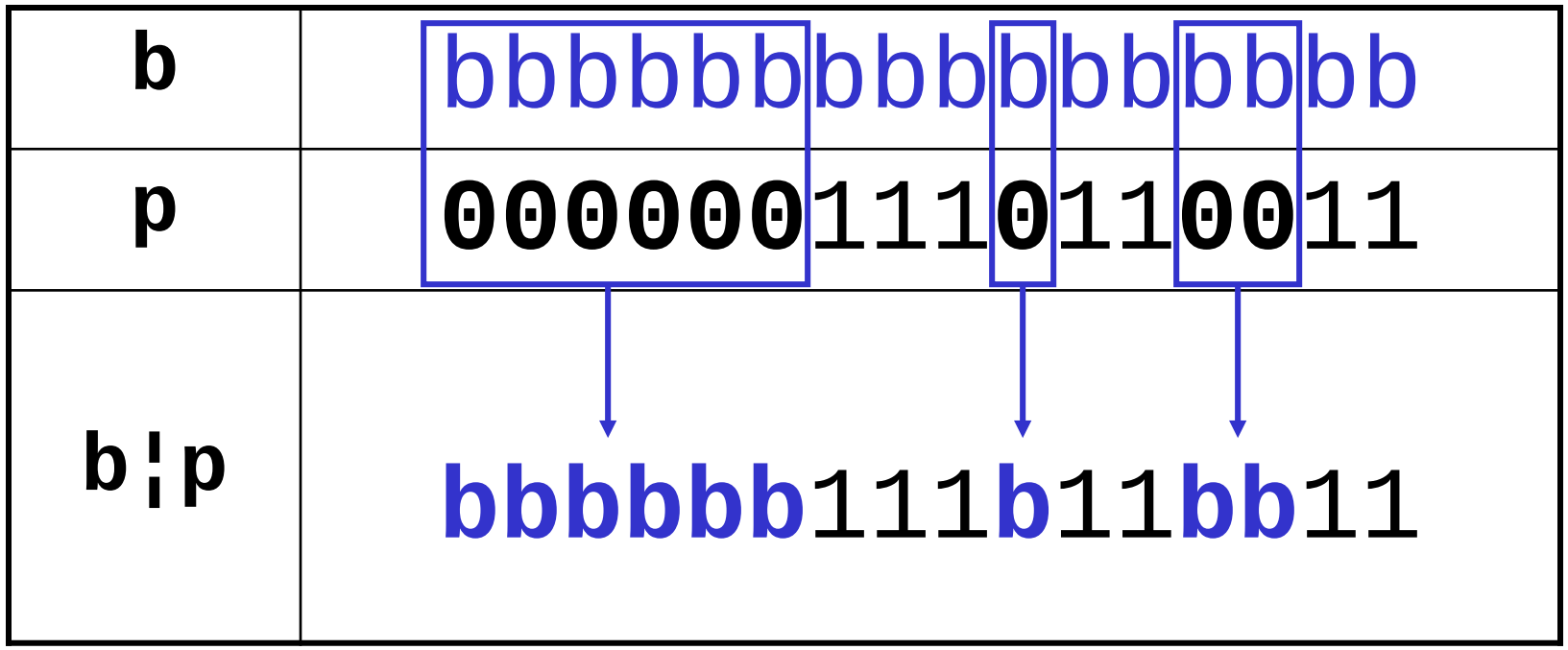
OU_Logique *n ¦ p*

n	0000010101101110
p	0000001110110011
n¦p	1

$n \nmid p$

n	0000010101101110
p	0000001110110011
n∤p	0000011111111111

b ¦ 1 donne 1
b ¦ 0 donne b



Negation bit-à-bit ~n

n	0000010101101110
~n	1

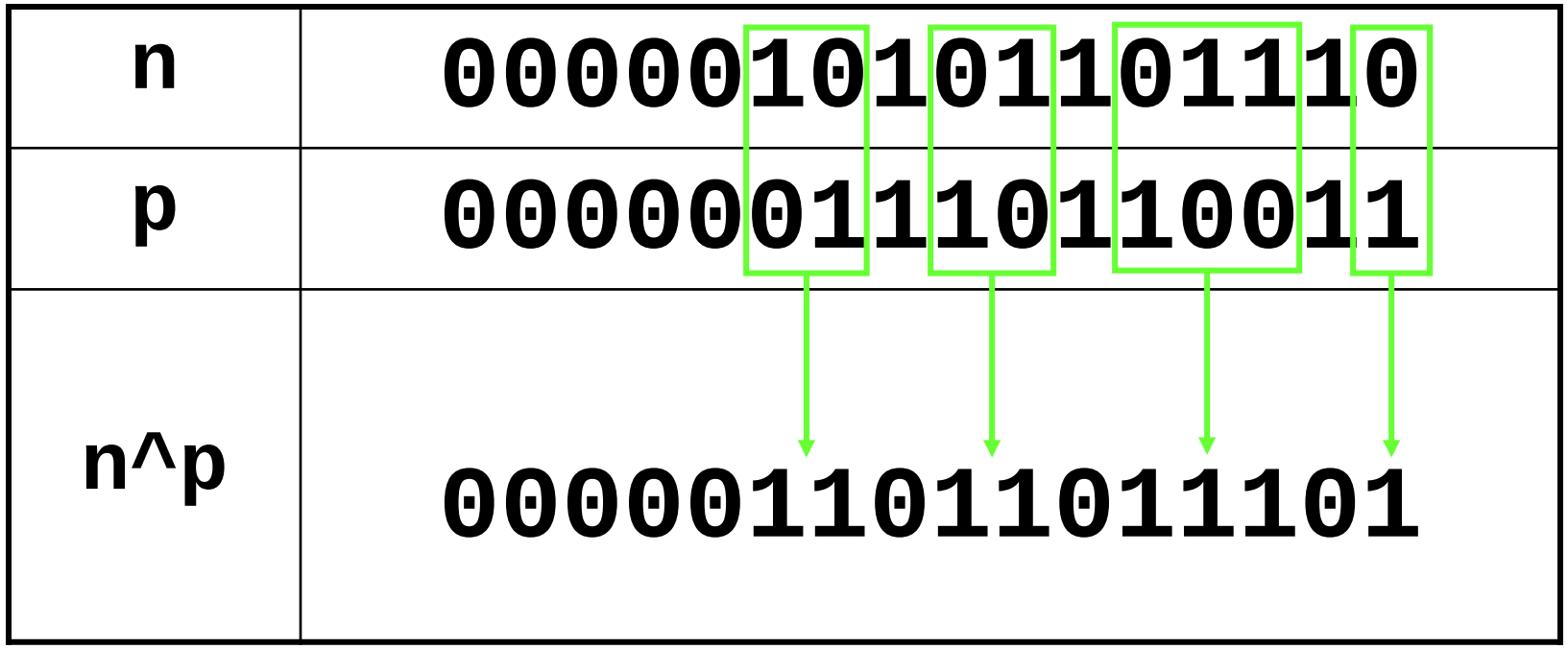
~n

n	0000010101101110
~n	1111101010010001

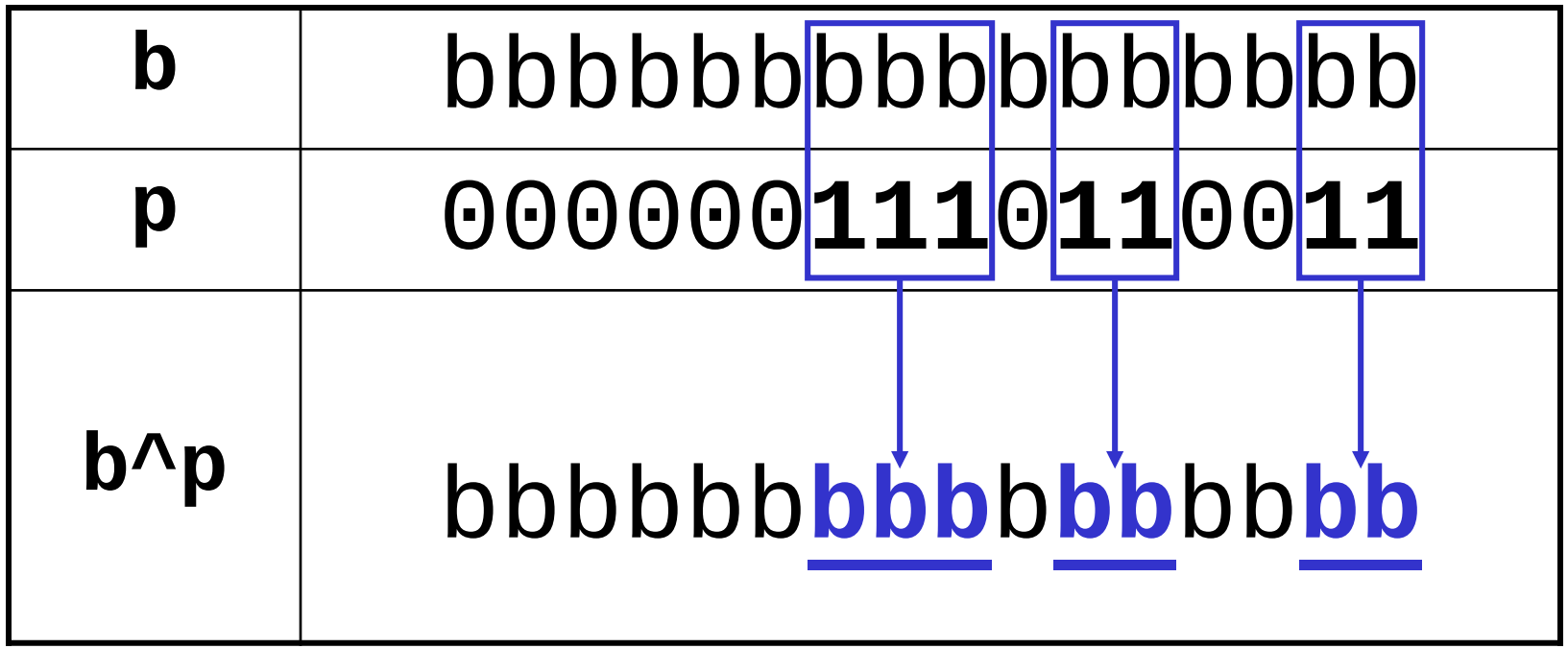
OU_Exclusif n ^ p

n	0000010101101110
p	0000001110110011
n^p	1

n ^ p


n
00000101011011110
p
00000011101100011
n^p
00000110110111101

b ^ 1 donne ~b (noté b)
b ^ 0 donne b



Décalage du motif binaire vers la gauche *: n << 3*

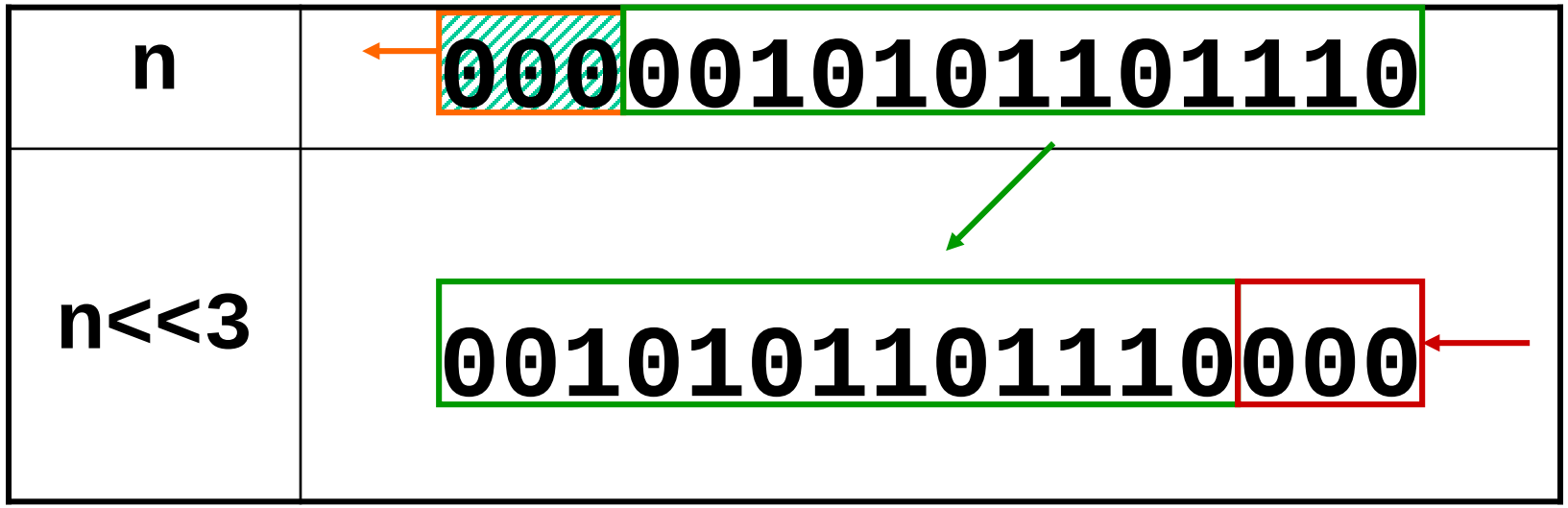
n	0000010101101110
n<<3	

n << 3

n	0000001010111011110
n<<3	

n << 3

équivalent à une multiplication par 2^3



Décalage du motif binaire vers la droite : *n >> 3*

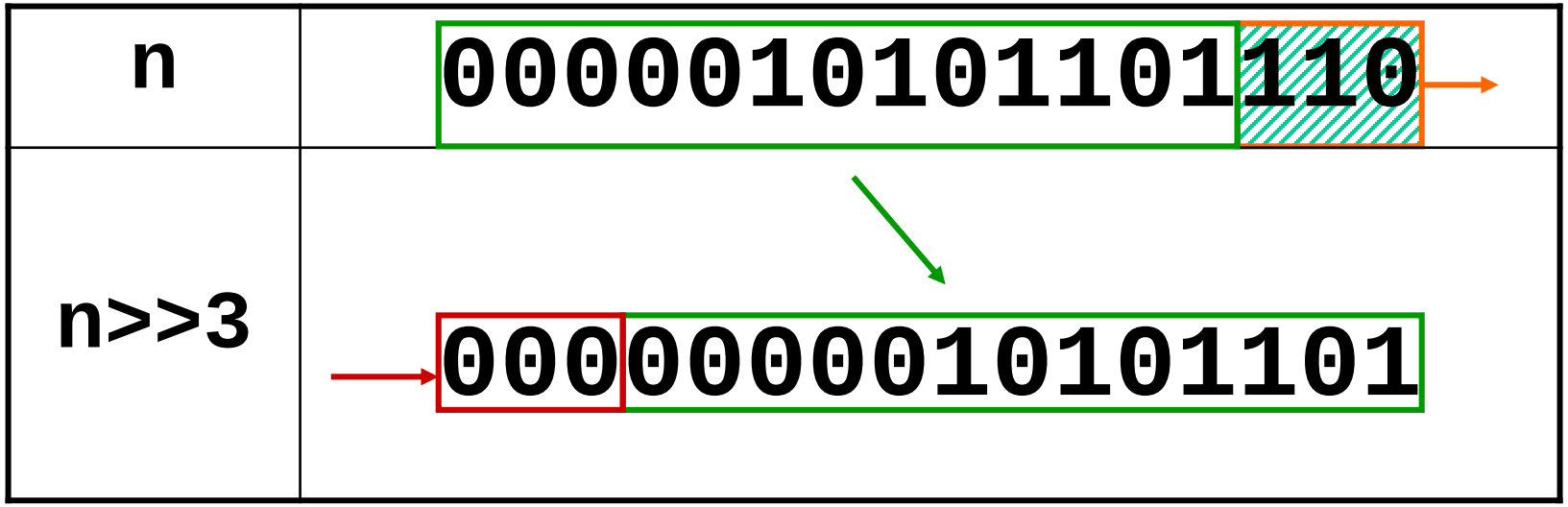
n	0000010101101110
n>>3	

n >> 3

n	00000101011101110 →
n>>3	

n >> 3

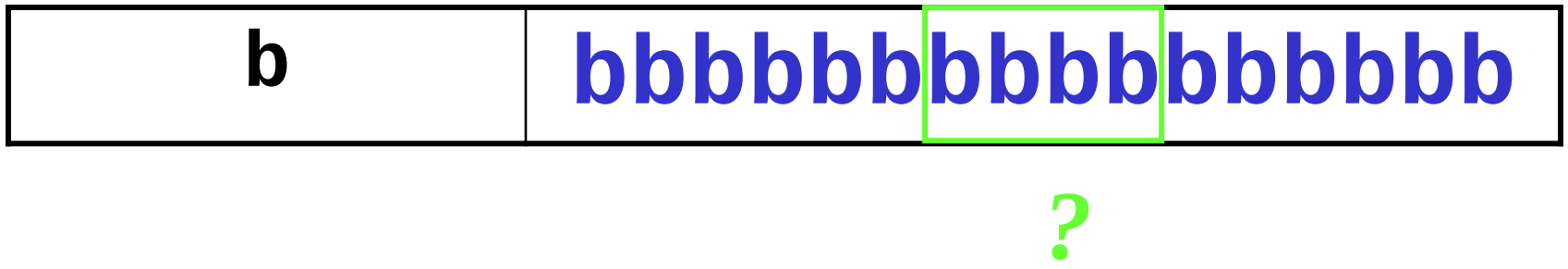
équivalent à une division entière par 2^3



Attention: comportement dépend de la machine pour les nombres signés négatifs

Ici pas de problème car le bit de poids fort est 0, donc peu importe si le nombre est de type int ou unsigned int, dans tous les cas des 0 sont introduits

Application système embarqué 1:
extraction d'un champ de bit
à l'aide d'un masque **m** avec décalage **d**



extraction d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbb`
`m = 0xF`	`0000000000001111`

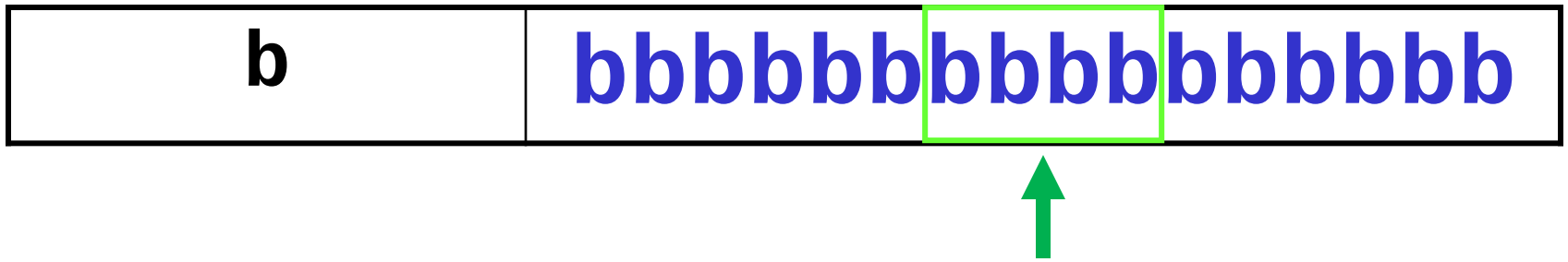
extraction d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbbbb`
`m = 0xF`	`00000000000001111`
`b>>6`	`??????bbbbbbbbbb`

extraction d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

b	bbbbbbbbbbbbbbbb
m = 0xF	0000000000001111
b>>6	??????bbbbbbbbbb
m&(b>>6)	0000000000bbbb

Application système embarqué 2:
insertion d'un champ de bit
à l'aide d'un masque **m** avec décalage **d**



on veut ranger
une nouvelle valeur ici

par exemple: v = 0010

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbb`
`m = 0xF`	`0000000000001111`

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbb`
`m = 0xF`	`0000000000001111`
`m<<6`	`0000001111000000`

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

b	bbbbbbbbbbbbbbbb
m = 0xF	0000000000001111
m<<6	0000001111000000
~(m<<6)	1111110000111111

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

b	bbbbbbbbbbbbbbbb
m = 0xF	0000000000001111
m<<6	0000001111000000
~(m<<6)	1111110000111111
b& ~(m<<6)	bbbbbb0000bbbbbb

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbb`
`b& ~(m<<6)`	`bbbbbbb0000bbbbbb`
`v = 0x2`	`0000000000000010`

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbb`
`b& ~(m<<6)`	`bbbbbbb0000bbbbbb`
`v = 0x2`	`0000000000000010`
`v << 6`	`0000000010000000`

insertion d'un champ de bit
*à l'aide d'un masque **m** avec décalage **d***

`b`	`bbbbbbbbbbbbbbbb`
`b& ~(m<<6)`	`bbbbbbb0000bbbbbb`
`v << 6`	`0000000010000000`
`(b&~(m<<6)) | (v << 6)`	`bbbbbbb0010bbbbbb`

Application 3: gestion de l'ouverture des fichiers en C++

Exemple: il existe plusieurs constantes définissant les options possibles pour ouvrir un **stream**

Il est autorisé d'activer plusieurs options: **écriture** + **binaire + app**

L'expression **`option1 ¦ option2 ¦ option3`**

fusionne ces 3 options dans un seul motif binaire entier

qui est passé à la méthode open() du stream