

- Supposons que votre meilleur.e ami.e habite en Nouvelle-Zélande, et que vous désiriez lui jouer un sketch pour son anniversaire.
- Il y a 150 ans, vous auriez eu besoin de 80 jours...
- Il y a 50 ans, seuls 2-3 jours auraient suffi...
- Mais aujourd'hui, seules quelques minutes suffisent !
(si on excepte le temps qu'il vous faut pour préparer le sketch)
- Que se passe-t-il exactement pendant ces quelques minutes?

Première étape

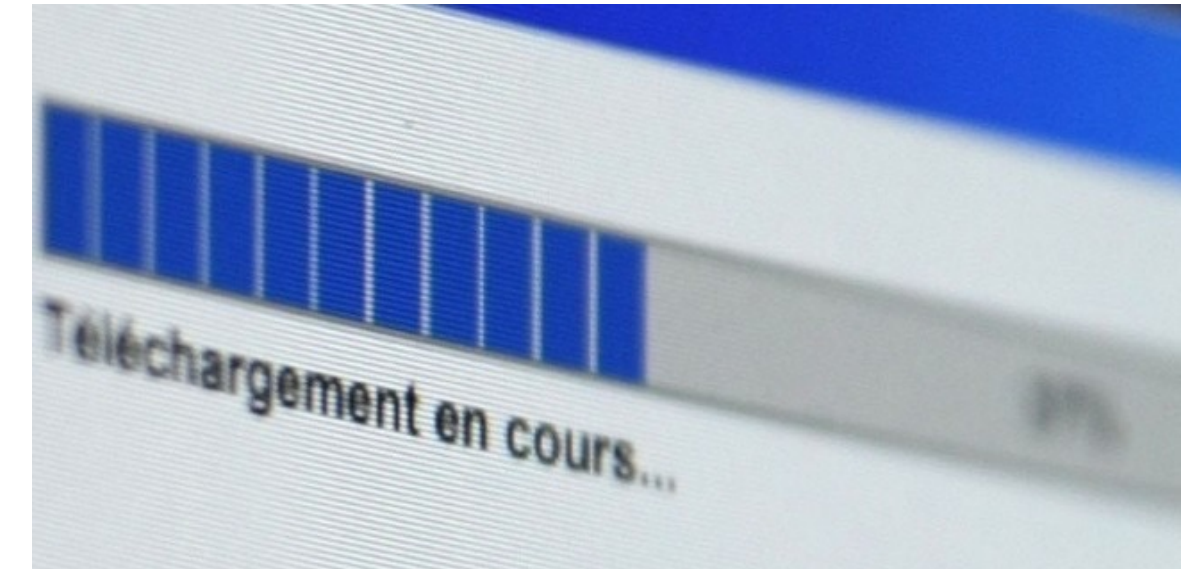
Avec des amis, vous préparez une vidéo amusante...



Le son et l'image sont enregistrés au format vidéo :

- Un signal **analogique** est converti en sa représentation **numérique**, au moyen d'un algorithme sophistiqué.
- Un algorithme de **correction d'erreurs** est utilisé pour enregistrer le fichier en mémoire.

Deuxième étape



Vous téléchargez la vidéo sur votre plateforme préférée...

(mais avant ça, vous réduisez sa taille au moyen d'un algorithme de **compression** pour que sa transmission ne pose pas de problèmes)

- Deux autres algorithmes de **correction d'erreurs** sont utilisés pour protéger la transmission des données :
 - de votre ordinateur/téléphone jusqu'à la prochaine borne wifi ;
 - sur internet.
- Un algorithme de **chiffrement** est également utilisé.

Troisième et dernière étape

Enfin, votre ami.e découvre la vidéo...

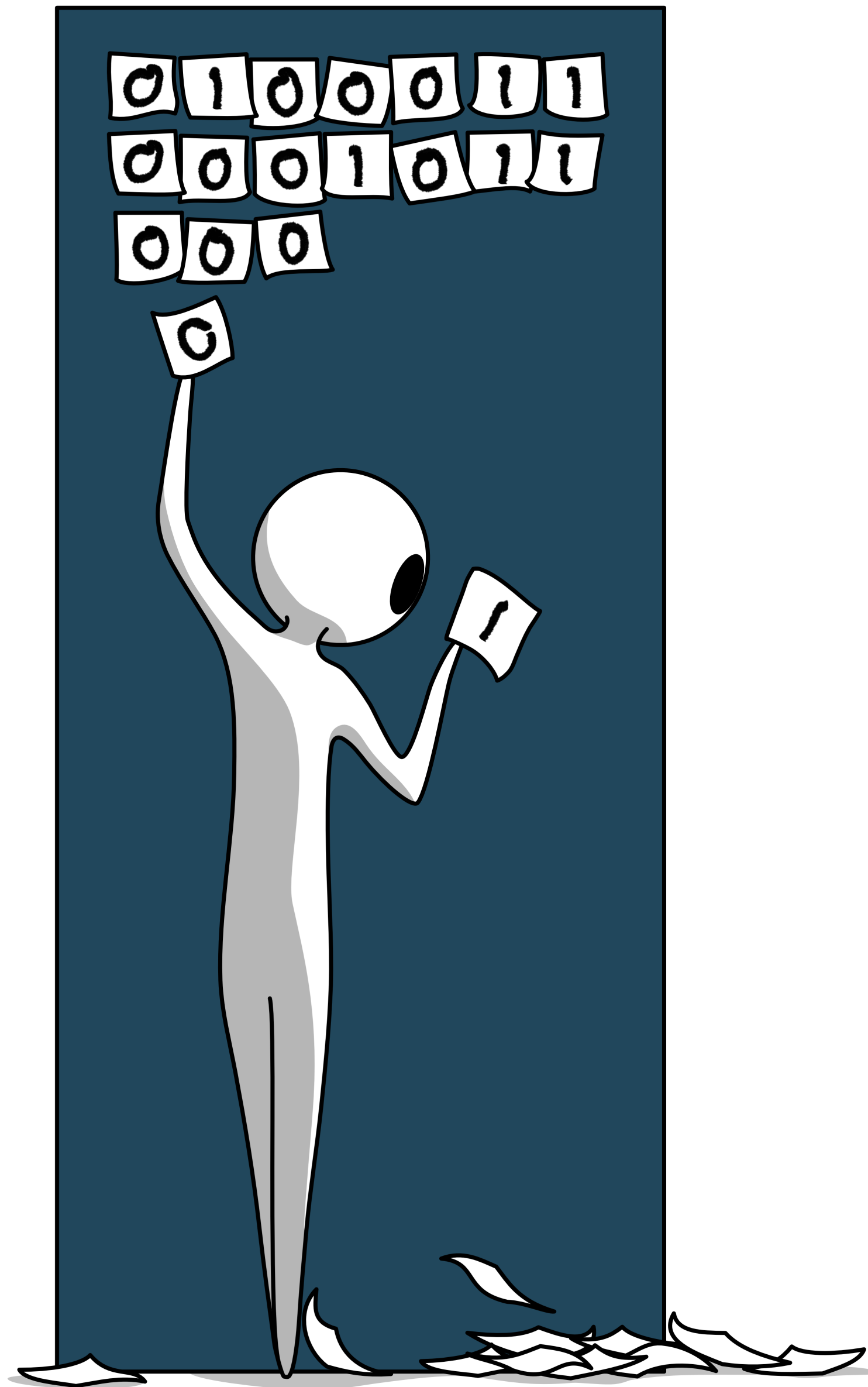


- Un ou deux algorithmes de **correction d'erreurs** sont à nouveau utilisés ici ;
- ainsi qu'un algorithme de **déchiffrement** ;
- et le signal est **reconstruit** à partir des données numériques.

- Dans nos gestes quotidiens, nous utilisons désormais un grand nombre d'algorithmes sophistiqués, souvent sans nous en rendre compte.
- L'omniprésence de ces algorithmes a, qu'on le veuille ou non, quelque peu changé notre manière de communiquer, de voyager, de voir le monde...
- Plusieurs contributions fondamentales, remontant pour la plupart à plus d'une septantaine d'années, ont permis la réalisation de ces moyens de communication modernes et l'avènement de notre ère digitale.
- Ce sont ces contributions que nous vous proposons de découvrir plus en détail dans les cours qui suivent.

Plan des semaines à venir

- Représentation de l'information :
 - Nombres entiers
 - Nombres réels
 - Implémentation concrète : circuits logiques et transistors
- Echantillonnage et reconstruction de signaux
- Entropie et compression de données
- Communication :
 - Correction d'erreurs
 - Réseaux
 - Cryptographie et sécurité



Information, Calcul et Communication

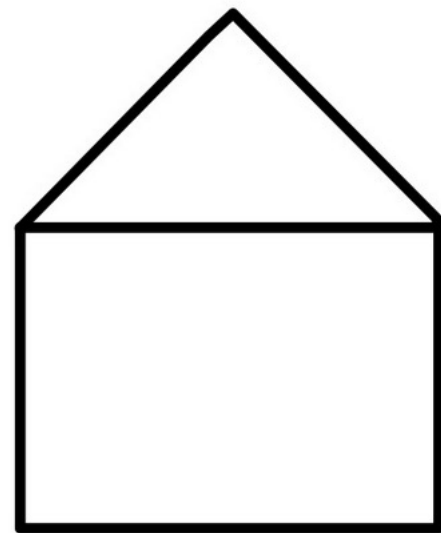
Représentation de
l'information

Olivier Lévêque

Il existe plusieurs façons de représenter une information.

Exemple:

- maison
- Haus
- casa
- chasa
- house
- domus



Mais encore:

- `.. _ . _ " ' ' _ _ _ _`
(code Morse)
- `77 65 73 83 79 78`
(code ASCII décimal)
- `4D 41 49 53 4F 4E`
(code ASCII hexadécimal)
- `01001101 01000001 ...`
(code ASCII binaire)

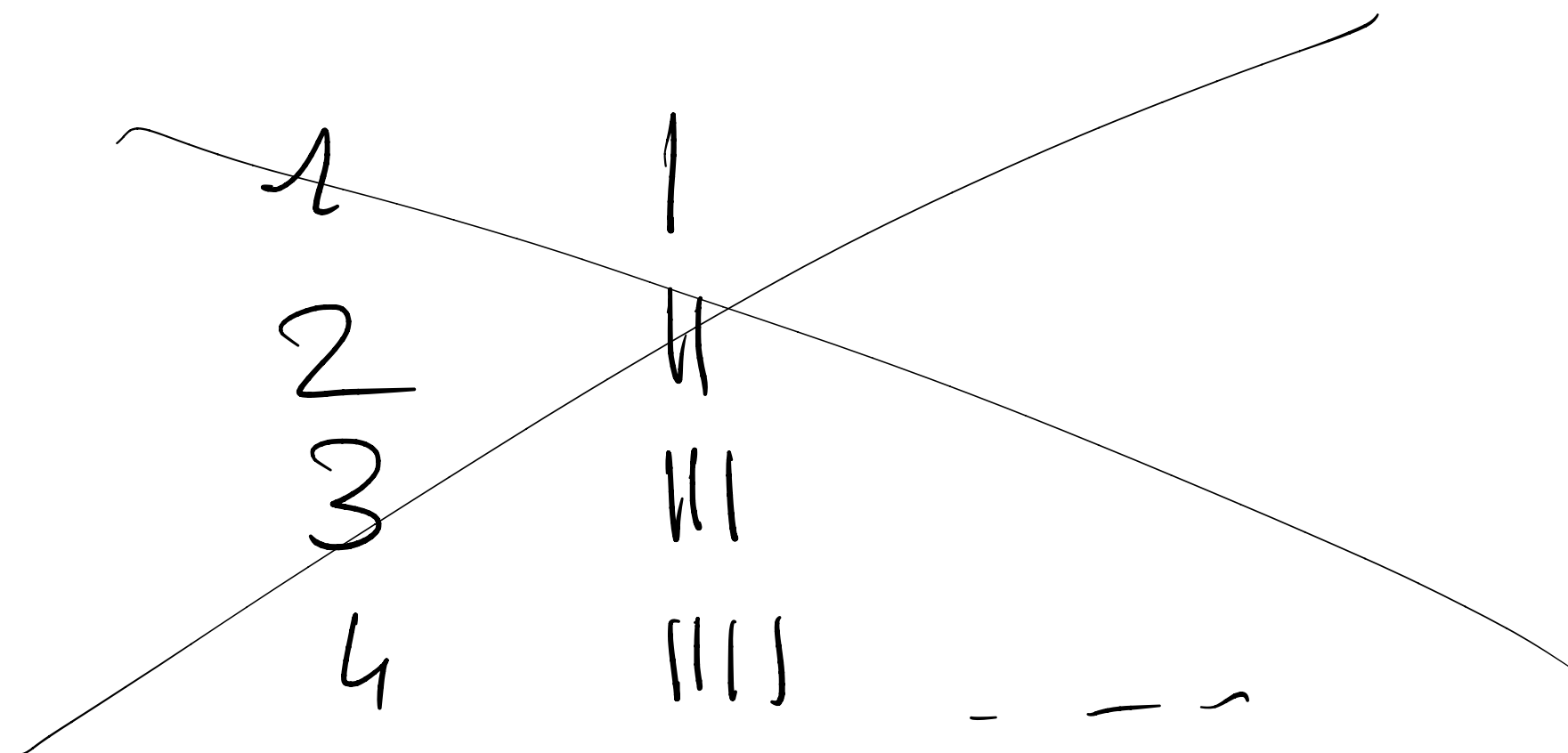
Pourquoi choisir la représentation binaire?

- Réduction à deux symboles (0 et 1) facile à implementer:

0 = circuit ouvert / 1 = circuit fermé

0 = tension de 0V / 1 = tension de 5V

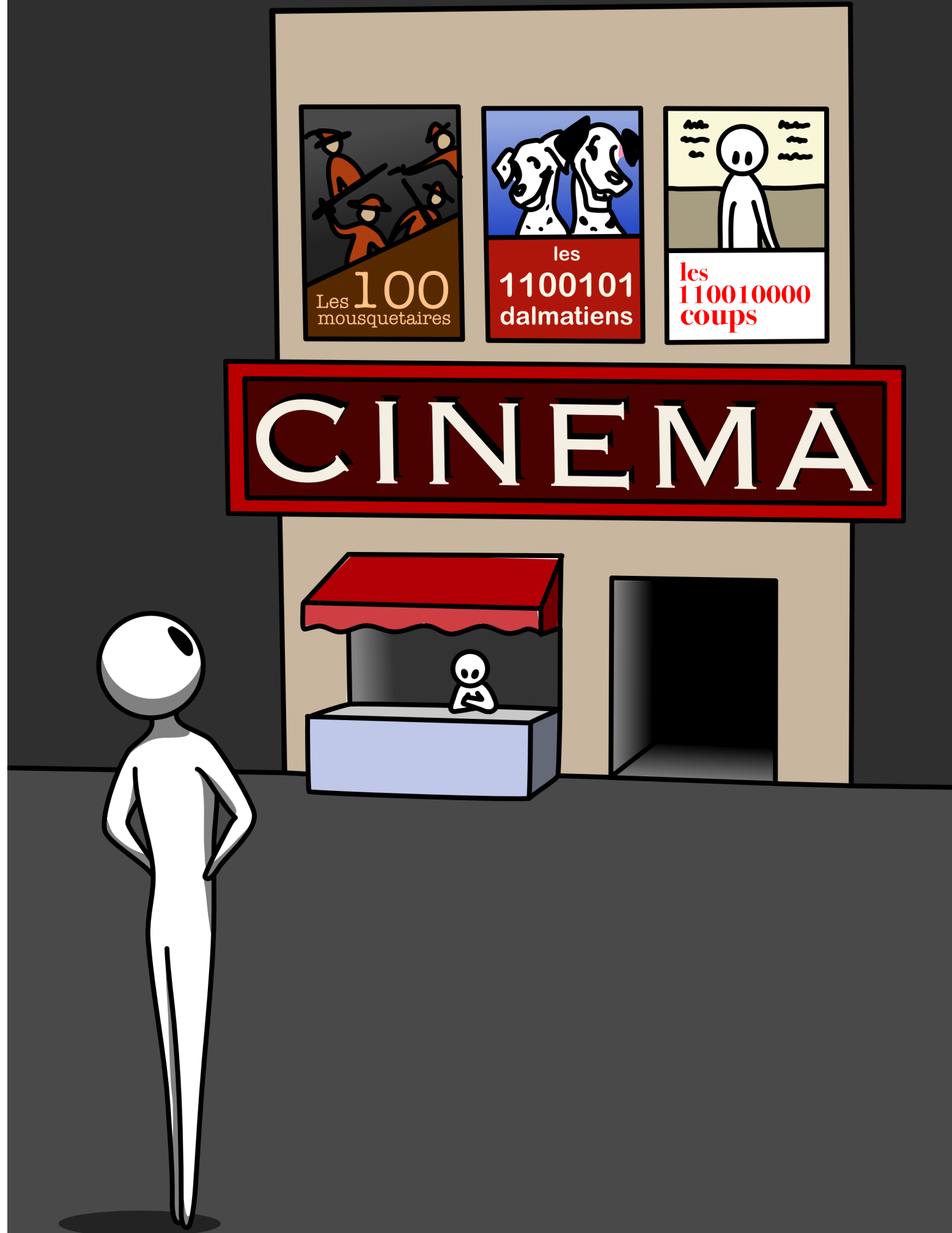
- Et au fait, pourquoi ne pas choisir *un seul* symbole (“1”, par exemple) ?



Avec n bits, on peut représenter 2^n éléments différents.

Exemples:

- 1 bit: “noir” (0) ou “blanc” (1)
- 2 bits: “à gauche” (00), “à droite” (01), “en haut” (10) ou “en bas” (11)
- 8 bits: 256 caractères différents (code ASCII étendu)
- 32 bits: plus de 4 milliards de caractères différents (code UTF-8)



Information, Calcul et Communication

Représentation binaire des nombres entiers

Olivier Lévêque

Représentation binaire des nombres entiers positifs

- Prenons un exemple de nombre entier positif: 1'984
- Ceci est une représentation ! (la représentation décimale)

$$1'984 = 1'000 + 900 + 80 + 4 = 1 \cdot 10^3 + 9 \cdot 10^2 + 8 \cdot 10^1 + 4 \cdot 10^0$$

(D'autres avant nous auraient écrit: M C M L X X X I V)

- Mais on peut aussi écrire: $1'984 = 1'024 + 512 + 256 + 128 + 64$
 $= 1 \cdot 2^{10} + 1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6$
 $+ 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$
 $\rightarrow 11111000000$ en binaire

EPFL Cas général

- Représentation décimale: $N = \sum_{j=0}^{m-1} c_j \cdot 10^j$ m chiffres $c_j \in \{0, 1, \dots, 9\}$

Nombre de chiffres nécessaires: $m = \underbrace{\lceil \log_{10}(N + 1) \rceil}_{\approx \log_{10}(N)}$

EPFL Cas général

- Représentation décimale: $N = \sum_{j=0}^{m-1} c_j \cdot 10^j$ m chiffres $c_j \in \{0,1, \dots, 9\}$

Nombre de chiffres nécessaires: $m = \lceil \log_{10}(N + 1) \rceil \approx \log_{10}(N)$

- Représentation binaire: $N = \sum_{i=0}^{n-1} b_i \cdot 2^i$ n bits $b_i \in \{0,1\}$

Nombre de bits nécessaires: $n = \lceil \log_2(N + 1) \rceil \approx \log_2(N)$

$$n = \log_2(N) = \frac{\log_{10}(N)}{\log_{10}(2)} = \frac{1}{\log_{10}(2)} \cdot m \approx \frac{1}{0.3} \cdot m \approx 3.3 m$$

EPFL Cas général

- Représentation décimale: $N = \sum_{j=0}^{m-1} c_j \cdot 10^j$ m chiffres $c_j \in \{0, 1, \dots, 9\}$

Nombre de chiffres nécessaires: $m = \lceil \log_{10}(N + 1) \rceil$

- Représentation binaire: $N = \sum_{i=0}^{n-1} b_i \cdot 2^i$ n bits $b_i \in \{0, 1\}$

Nombre de bits nécessaires: $n = \lceil \log_2(N + 1) \rceil$

- Attention! Avec n bits, on peut représenter 2^n nombres entiers différents:
les nombres de 0 (= 000 ... 0) à $2^n - 1$ (= 111 ... 1) **et donc pas 2^n lui-même!**

$$\begin{aligned} &= 1 + 2 + 4 + 8 + 16 + \dots + 2^{n-1} \\ &= 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \dots + 2^{n-1} \end{aligned}$$

- Représentation décimale: $N = \sum_{j=0}^{m-1} c_j \cdot 10^j$ m chiffres $c_j \in \{0,1, \dots, 9\}$

Nombre de chiffres nécessaires: $m = \lceil \log_{10}(N + 1) \rceil$

- Représentation binaire: $N = \sum_{i=0}^{n-1} b_i \cdot 2^i$ n bits $b_i \in \{0,1\}$

Nombre de bits nécessaires: $n = \lceil \log_2(N + 1) \rceil$

- Attention! Avec n bits, on peut représenter 2^n nombres entiers différents: les nombres de 0 (= 000 ... 0) à $2^n - 1$ (= 111 ... 1) **et donc pas 2^n lui-même!**

Exemple avec $n = 8$ bits: intervalle de 0 à $2^8 - 1 = 255$

- Addition:

$$\begin{array}{r} 111 \\ 0011 \\ + 0101 \\ \hline = 1000 \end{array} \quad \begin{array}{r} 3 \\ + 5 \\ \hline 8 \checkmark \end{array}$$

- Soustraction:

$$\begin{array}{r} 0_{10} \\ 0101 \\ - 0011 \\ \hline = 0010 \end{array} \quad \begin{array}{r} 5 \\ - 3 \\ \hline 2 \checkmark \end{array}$$

⚠ $0+0=0, 0+1=1+0=1, 1+1=10, 1+1+1=11$

Opérations binaires: multiplication et division

En binaire, multiplier et diviser par 2 est très facile:

(de même que multiplier et diviser par 10 est très facile en décimal)

$$\begin{array}{r} 0110 \quad 6 \\ \times 0010 \quad \times 2 \\ \hline = 1100 \quad 12 \end{array}$$

←

$$\begin{array}{r} 0110 \quad 6 \\ \div 0010 \quad \div 2 \\ \hline 0011 \quad 3 \end{array}$$

→

Opérations binaires: dépassement de capacité

Etant donné la limite imposée par le nombre de bits utilisés, des problèmes de *dépassement de capacité* (*overflow*) surviennent lorsqu'on effectue des opérations binaires et que le résultat attendu se trouve en dehors de l'intervalle des nombres représentables.

Exemples: ($n=4$)

$$\begin{array}{r}
 \textcircled{\times} \begin{array}{r} 111 \\ 1011 \\ + 0111 \\ \hline 0010 \\ \hline \hline \end{array} \\
 \begin{array}{r} 11 \\ + 7 \\ \hline 2 \quad ? \end{array}
 \end{array}$$

$$\begin{array}{r}
 \textcircled{\oplus} \begin{array}{r} 0111 \\ - 1011 \\ \hline 1100 \\ \hline \hline \end{array} \\
 \begin{array}{r} 7 \\ - 11 \\ \hline 12 \quad ? \end{array}
 \end{array}$$

$$\begin{array}{r}
 \textcircled{\times} \begin{array}{r} 1111 \\ 1111 \\ + 0001 \\ \hline 0000 \\ \hline \hline \end{array} \\
 \begin{array}{r} 15 \\ + 1 \\ \hline 0 \quad ? \end{array}
 \end{array}$$

nbs représentables: 0 à 15
 0000 à 1111
 (avec $n=4$ bits)

Représentation binaire des nombres entiers relatifs

- Avec n bits, on utilise la convention suivante pour représenter les nombres entiers relatifs (positifs et négatifs) :

$$N = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

$n=8$

}

$10000000 \rightarrow -128$

$11111111 \rightarrow -1$

$00000000 \rightarrow 0$

$01111111 \rightarrow +127$

- Exemple avec 8 bits: $N = -43$ est représenté par **11010101**, car

$$-43 = -128 + 64 + 16 + 4 + 1$$

$$= -\mathbf{1} \cdot 2^7 + \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$$

//

Représentation binaire des nombres entiers relatifs

Remarques:

- Le premier bit est le bit de signe (0 : nombre positif, 1 : nombre négatif)
- Avec 8 bits, les nombres représentables vont de -128 à $+127$:

$$\text{-128 = 10000000} \rightarrow \text{+127 = 01111111}$$

- $-1 = 11111111$ avec cette représentation
- $+128$ n'est *pas* représentable avec 8 bits !

Nbs entiers relatifs:

(ljs avec $n=4$ bits)

bit de signe + partie positive du nombre

$$\begin{array}{l} \text{ex: } \left\{ \begin{array}{l} -3 \longrightarrow 1 + 0011 \\ +3 \longrightarrow 0 + 0011 \end{array} \right. \end{array}$$

Dans cette représentation:

$$\begin{array}{r} \\ \\ \\ \hline 1 \end{array} \quad \begin{array}{l} -3 \\ +3 \\ -6 \quad ??? \end{array}$$

Comment trouver la rep. bin. de -43 ?

(avec $n=8$ bits)

Recette : • trouver la rep. bin. de $+43 = 32 + 8 + 2 + 1$

$$\rightarrow X = 00101011$$

• calculer le complément à 1 de X :

$$\begin{array}{r} 11111111 \\ - 00101011 \\ \hline 11010100 \end{array}$$

• calculer le complément à 2 de x :

$$\begin{array}{r} 11010100 \\ + 00000001 \\ \hline 11010101 \end{array}$$

Opérations binaires: addition et soustraction (bis)

■ Addition:

$$\begin{array}{r} 11 \\ 0011 \\ + 1001 \\ \hline = 1100 \end{array}$$

+3
-7
-4 ✓

~~⊗~~ 11 (n=4 bits)
1111 -1
+ 1110 -2

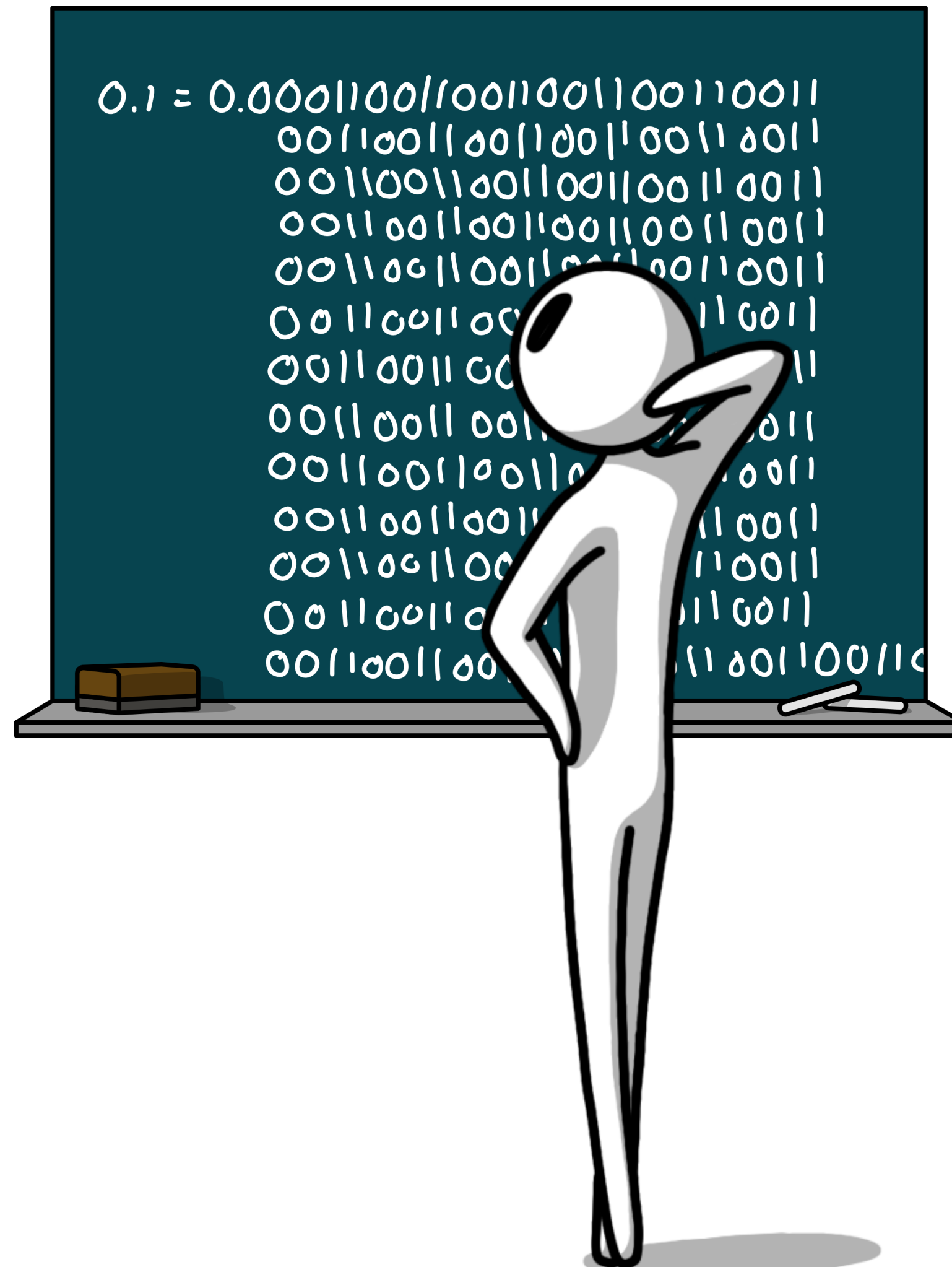
= 1101 -3 ✓

■ Soustraction:

nbs représentables: $-8 \rightarrow +7$

~~⊗~~ 11
1011 -5
+ 1011 -5

0110 +6 ??



Information, Calcul et Communication

Représentation binaire des nombres réels

Olivier Lévêque

Représentation binaire des nombres réels

Première remarque:

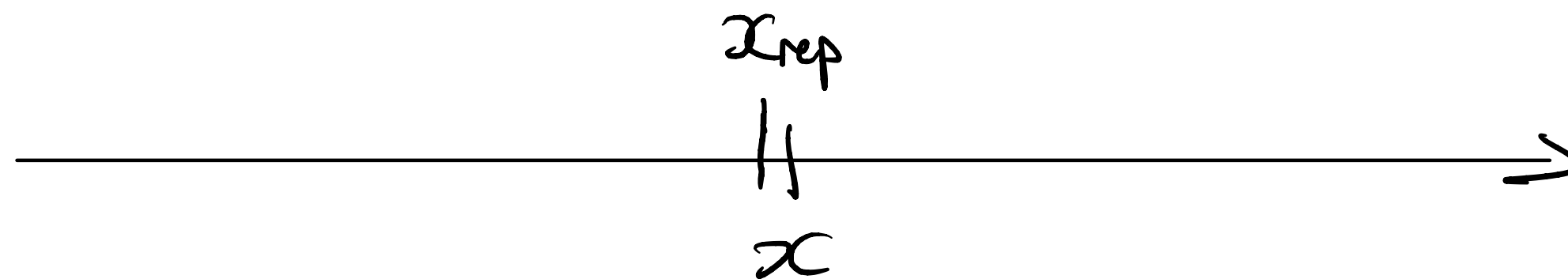
Avec un nombre fini de bits, on ne peut pas représenter *tous* les nombres réels de manière exacte ! (même si on se limite à l'intervalle fermé $[0,1]$)

Il faut donc s'attendre à effectuer des *erreurs* dans ce cas.

Soit x_{rep} la valeur représentée en binaire du nombre x . On définit:

l'erreur *absolue*: $|\Delta x| = |x - x_{rep}|$

l'erreur *relative*: $|\Delta x|/|x|$
(= "précision")



Représentation binaire des nombres réels

- Prenons un nombre entre 0 et 1, par exemple, $x = 0,375$:

$$x = 0,375 = 0,25 + 0,125 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} \quad \rightarrow \quad 011$$

(représentation exacte!)

- Autre exemple:

$$y = \pi - 3 = 0,1415926535 \dots = 0,125 + \dots$$

$$= 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + \dots \quad \rightarrow \quad 0010 \dots$$

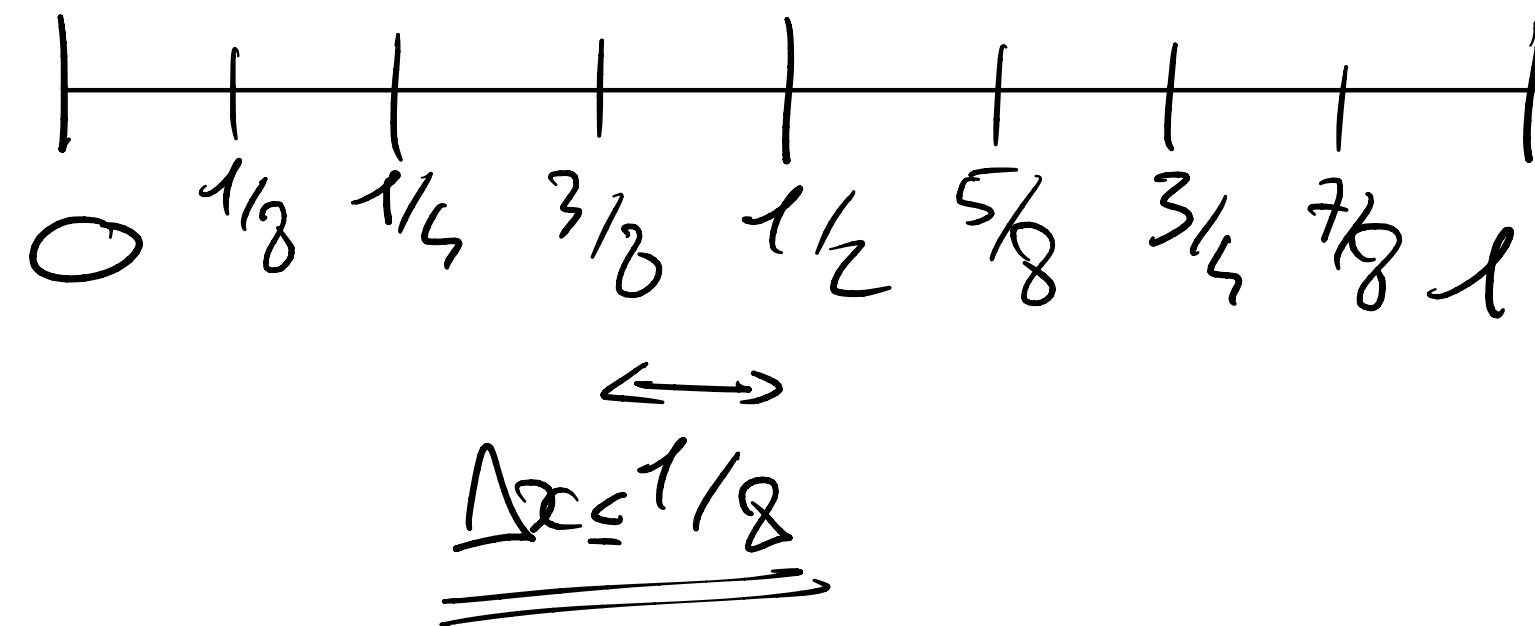
(représentation approximative!)

Représentation binaire des nombres réels: représentation en virgule fixe

- Nombres réels plus grand que 1:

n bits pour la partie entière, n bits pour la partie décimale

→ “tous” les nombres de 0 à 2^n avec une erreur absolue $\leq \frac{1}{2^n}$



avec n bits, pour la partie décimale d'un nombre,
 $\Delta x \leq \frac{1}{2^n}$

Représentation binaire des nombres réels: représentation en virgule fixe

- Erreur absolue avec cette représentation:

$$|\Delta x| = |x - x_{rep}| \leq 2^{-n}$$

si n bits sont utilisés pour la partie décimale. ✓

- Erreur relative?

$$|\Delta x| / |x|$$

peut être arbitrairement grande si x est proche de 0. ✗

Représentation binaire des nombres réels: représentation en virgule flottante

Pour pallier à ce problème de précision, on choisit plutôt la représentation suivante:

1. On garde la représentation en virgule fixe avec n bits pour les nombres réels dans l'intervalle $[1,2]$

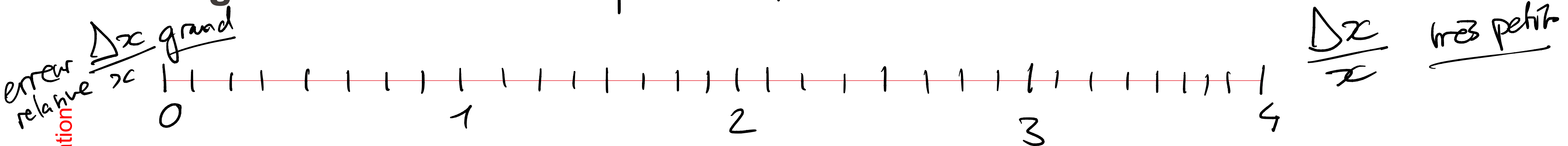
$$\rightarrow \text{erreur relative } |\Delta x| / |x| \leq 2^{-n} \quad \text{car } |x| \geq 1$$

2. On réplique cette représentation à toutes les échelles en la multipliant ou en la divisant par des puissances de 2.

Représentation binaire des nombres réels: représentation en virgule flottante

Schéma comparatif:

Virgule fixe: ($n=3$ bits par la partie décimale)



Virgule flottante:

