Science informatique

CM9

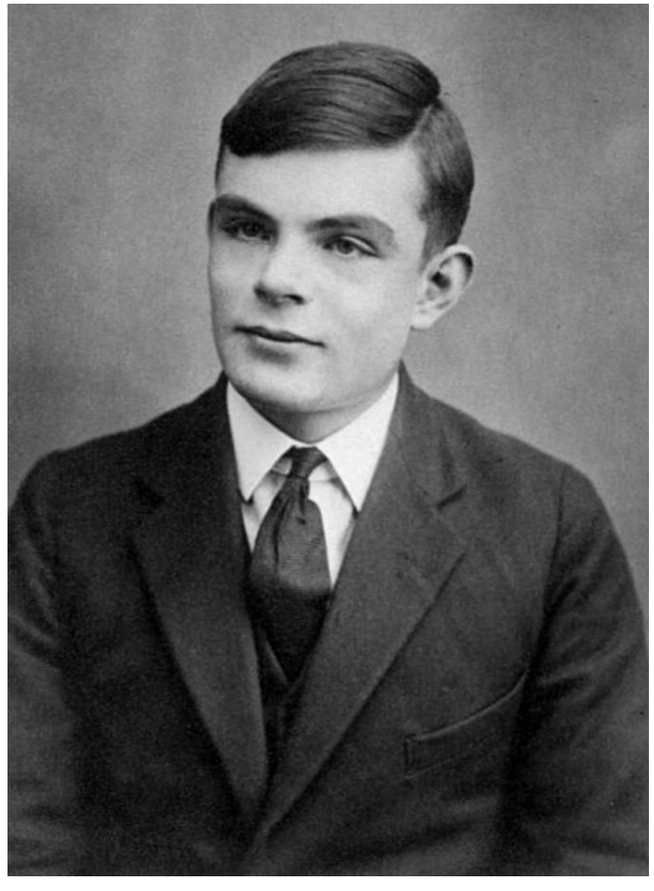
Antonio E. Porreca
aeporreca.org/scienceinfo

Problèmes indécidables

Thèse de Church-Turing

Tout ce qu'on peut calculer, on peut le calculer
avec une machine de Turing (ou en pseudo-code,
ou en Python, ou d'autres modèles équivalents)

🤝

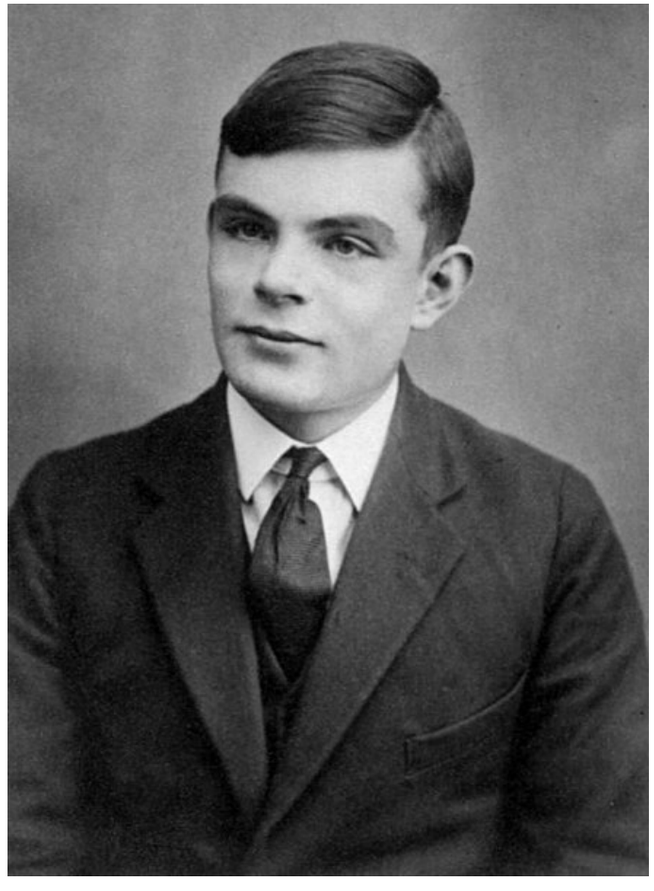


Thèse de Church-Turing

🤔

Tout ce qu'on peut calculer, on peut le calculer
avec une machine de Turing (ou en pseudo-code,
ou en Python, ou d'autres modèles équivalents)

🤝



Mais… est-ce qu’il existe des choses qu’on ne peut pas calculer ?

Programmes en entrée

```
def appliquer(f, x):
    return f(x)
```

Programmes en entrée

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

Programmes en entrée

```
def appliquer(f, x):
    return f(x)

def carré(x):
    return x**2

>>> appliquer(carré, 2)
```

Programmes en entrée

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
4
```

Programmes en entrée

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
4
>>> appliquer(carré, 3)
```

Programmes en entrée

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
4
>>> appliquer(carré, 3)
9
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
9
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
9
>>> appliquer(composer(successeur, carré), 2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
9
>>> appliquer(composer(successeur, carré), 2)
5
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
9
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
9
>>> composer(successeur, carré)(2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
9
>>> composer(successeur, carré)(2)
5
```

Un exemple plus intéressant : la fonction dérivée

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                       # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                       # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                       # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
2
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                       # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
2 4.000999999999699
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                       # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
2 4.000999999999699
3 6.000999999999479
4 8.001000000000037
```

Exercice 3 du TD7

Quoi d’autre ?

```
def afficher(x):
    print('voilà', x)
```

Quoi d’autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
```

Quoi d’autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
```

Quoi d’autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
```

Quoi d'autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
```

Quoi d'autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
>>> afficher([1, 2, 3])
```

Quoi d’autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
>>> afficher([1, 2, 3])
voilà [1, 2, 3]
```

Quoi d'autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
>>> afficher([1, 2, 3])
voilà [1, 2, 3]
>>> afficher(carré)
```

Quoi d’autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
>>> afficher([1, 2, 3])
voilà [1, 2, 3]
>>> afficher(carré)
voilà <function carré at 0x1035c83a0>
```

Quoi d'autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
>>> afficher([1, 2, 3])
voilà [1, 2, 3]
>>> afficher(carré)
voilà <function carré at 0x1035c83a0>
>>> afficher(afficher)
```

Quoi d'autre ?

```
def afficher(x):
    print('voilà', x)


>>> afficher(5)
voilà 5
>>> afficher('salut')
voilà salut
>>> afficher([1, 2, 3])
voilà [1, 2, 3]
>>> afficher(carré)
voilà <function carré at 0x1035c83a0>
>>> afficher(afficher)
voilà <function afficher at 0x1035c8790>
```

```
afficher(afficher)
```

```
afficher(afficher)
```

Ceci n'est pas un appel récursif.

Mais… est-ce qu’il existe des choses qu’on ne peut pas calculer ?

Malheureusement, oui !

- Il existe des fonctions mathématiquement bien définies…
- …avec aucun algorithme pour les calculer
- L’exemple classique est le problème de l’arrêt, dû à Alan Turing lui-même
- Il n’y a pas d’algorithme qui prend en entrée un programme, une entrée pour le programme et établit toujours correctement si le programme s’arrête sur cette entrée

Exercice 4 du TD7

Programmes qui s’arrêtent, ou pas

Programmes qui s’arrêtent, ou pas

```
def terminer(x):
    print('je termine')
```

Programmes qui s’arrêtent, ou pas

```
def terminer(x):
    print('je termine')
```

```
def boucler(x):
    while True:
        print('je boucle')
```

Programmes qui s’arrêtent, ou pas

```
def terminer(x):
    print('je termine')
```

```
def boucler(x):
    while True:
        print('je boucle')
```

```
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

Si on pouvait calculer l’arrêt…

```
def terminer(x):
    print('je termine')
```

```
def boucler(x):
    while True:
        print('je boucle')
```

```
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

Si on pouvait calculer l'arrêt…

```
def terminer(x):
    print('je termine')
```

```
def boucler(x):
    while True:
        print('je boucle')
```

```
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```
from turing import s_arrête   # si seulement !
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```
from turing import s_arrête   # si seulement !

>>> s_arrête(terminer, 3)
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```
from turing import s_arrête   # si seulement !

>>> s_arrête(terminer, 3)
True
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```python
from turing import s_arrête   # si seulement !
```

```
>>> s_arrête(terminer, 3)
True
>>> s_arrête(boucler, 5)
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```python
from turing import s_arrête   # si seulement !
```

```
>>> s_arrête(terminer, 3)
True
>>> s_arrête(boucler, 5)
False
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```python
from turing import s_arrête   # si seulement !

>>> s_arrête(terminer, 3)
True
>>> s_arrête(boucler, 5)
False
>>> s_arrête(terminer_parfois, 7)
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```python
from turing import s_arrête   # si seulement !

>>> s_arrête(terminer, 3)
True
>>> s_arrête(boucler, 5)
False
>>> s_arrête(terminer_parfois, 7)
True
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```python
from turing import s_arrête   # si seulement !

>>> s_arrête(terminer, 3)
True
>>> s_arrête(boucler, 5)
False
>>> s_arrête(terminer_parfois, 7)
True
>>> s_arrête(terminer_parfois, -2)
```

Si on pouvait calculer l'arrêt…

```python
def terminer(x):
    print('je termine')
```

```python
def boucler(x):
    while True:
        print('je boucle')
```

```python
def terminer_parfois(x):
    if x >= 0:
        print('je termine')
    else:
        while True:
            print('je boucle')
```

```python
from turing import s_arrête   # si seulement !
```

```
>>> s_arrête(terminer, 3)
True
>>> s_arrête(boucler, 5)
False
>>> s_arrête(terminer_parfois, 7)
True
>>> s_arrête(terminer_parfois, -2)
False
```

Si on pouvait calculer l’arrêt…

```
def afficher(x):
    print('voilà', x)
```

```
>>> s_arrête(afficher, 3)
```

Si on pouvait calculer l’arrêt…

```
def afficher(x):
    print('voilà', x)
```

```
>>> s_arrête(afficher, 3)
True
```

Si on pouvait calculer l'arrêt…

```
def afficher(x):
    print('voilà', x)
```

```
>>> s_arrête(afficher, 3)
True
>>> s_arrête(afficher, 'salut')
```

Si on pouvait calculer l'arrêt…

```
def afficher(x):
    print('voilà', x)
```

```
>>> s_arrête(afficher, 3)
True
>>> s_arrête(afficher, 'salut')
True
```

Si on pouvait calculer l'arrêt…

```
def afficher(x):
    print('voilà', x)
```

```
>>> s_arrête(afficher, 3)
True
>>> s_arrête(afficher, 'salut')
True
>>> s_arrête(afficher, afficher)
```

Si on pouvait calculer l'arrêt…

```
def afficher(x):
    print('voilà', x)
```

```
>>> s_arrête(afficher, 3)
True
>>> s_arrête(afficher, 'salut')
True
>>> s_arrête(afficher, afficher)
True
```

La fonction diagonale

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

La fonction diagonale

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

```
>>> diagonale(afficher)
```

```
def afficher(x):
    print('voilà', x)
```

La fonction diagonale

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

```
>>> diagonale(afficher)
je boucle
je boucle
je boucle
je boucle
```

```
def afficher(x):
    print('voilà', x)
```

La fonction diagonale

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

```
>>> diagonale(boucler)
```

```
def boucler(x):
    while True:
        print('je boucle')
```

La fonction diagonale

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

```
>>> diagonale(boucler)
je termine
>>>
```

```
def boucler(x):
    while True:
        print('je boucle')
```

La fonction diagonale

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')


>>> diagonale(diagonale)
```

Montrons que la fonction `s_arrête` n'existe pas

Rappel : `s_arrête` est censée toujours répondre correctement

Si `diagonale(diagonale)` s'arrête, alors…

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` s'arrête, alors…

on termine ici

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` s'arrête, alors…

on termine ici

donc ça doit être faux

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` s'arrête, alors…

on termine ici

donc ça doit être faux

mais `x == diagonale`

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` s'arrête, alors…

on termine ici

donc ça doit être faux

mais `x == diagonale`

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

donc `s_arrête` s'est trompée !

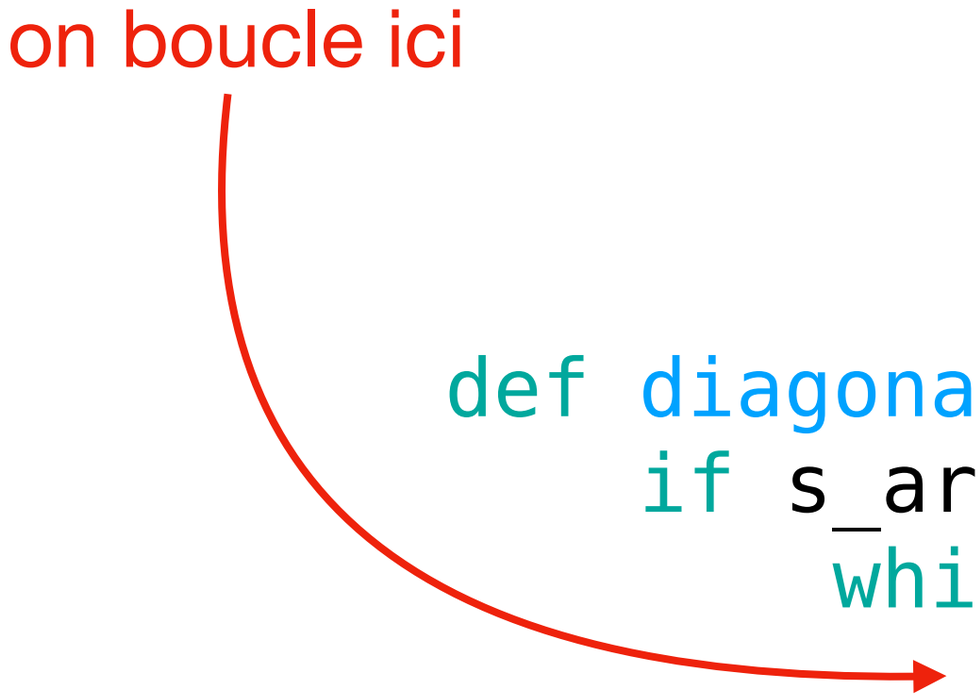
Alors il faut que `diagonale(diagonale)` ne s’arrête pas !

Si `diagonale(diagonale)`

ne s’arrête pas, alors…

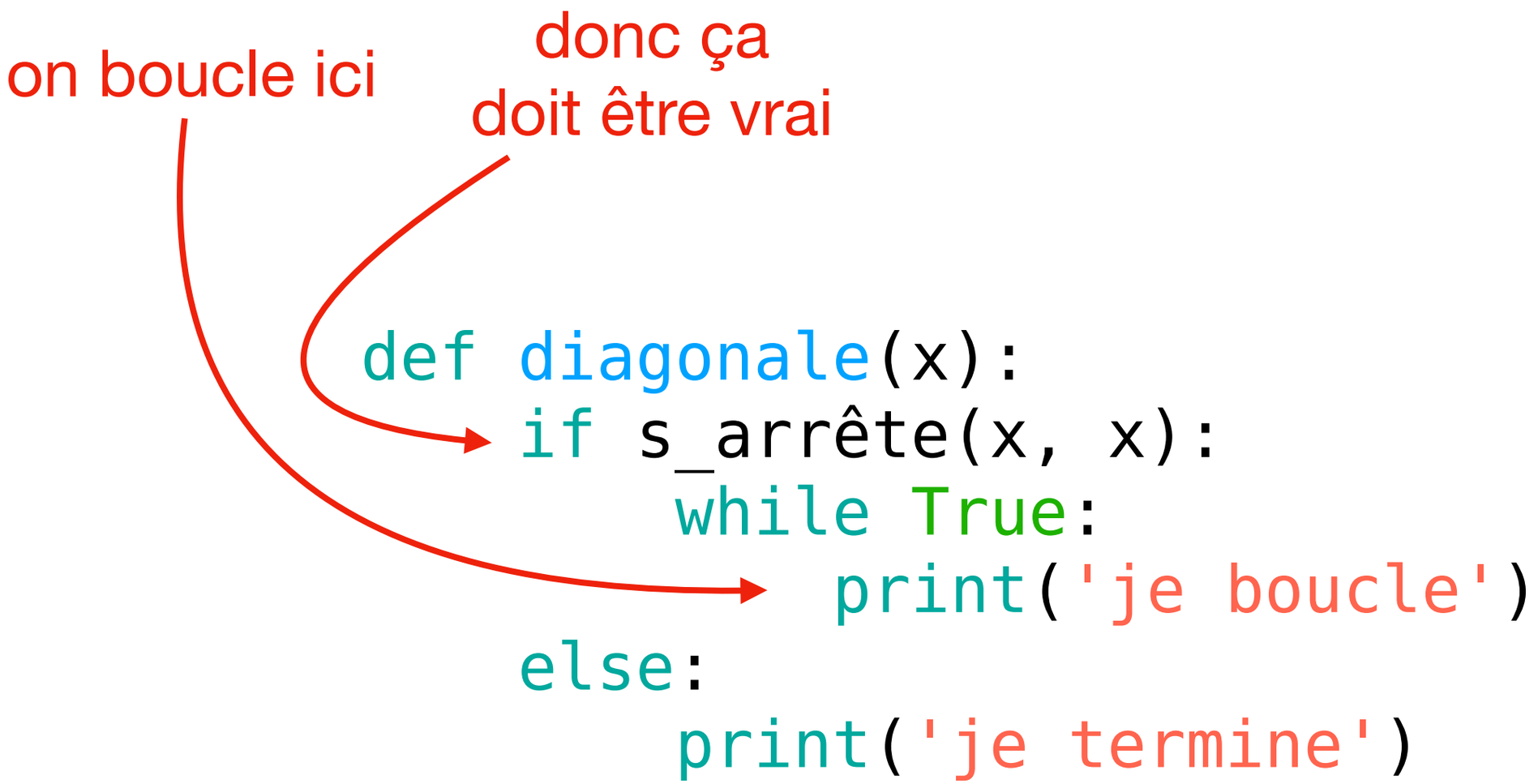
```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` ne s'arrête pas, alors…




```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` ne s'arrête pas, alors…



Si `diagonale(diagonale)` ne s'arrête pas, alors…

on boucle ici

donc ça doit être vrai

mais `x == diagonale`

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

Si `diagonale(diagonale)` ne s'arrête pas, alors…

on boucle ici

donc ça doit être vrai

mais `x == diagonale`

```
def diagonale(x):
    if s_arrête(x, x):
        while True:
            print('je boucle')
    else:
        print('je termine')
```

donc `s_arrête` s'est trompée encore une fois !

`diagonale(diagonale)` s'arrête ou pas ?

- Si `diagonale(diagonale)` s'arrête,
  on a une contradiction (`s_arrête` se trompe)

- Alors il faudrait qu'elle ne s'arrête pas,
  mais ça aussi est contradictoire, pour la même raison !

- Mais soit `diagonale` s'arrête sur l'entrée `diagonale`,
  soit elle ne s'arrête pas sur cette entrée !

Il n’y a pas d’algorithme pour le problème de l’arrêt

- `s_arrête(diagonale, diagonale)` ne peut être ni vrai ni faux, alors que c’est bien l’un des deux cas

- Donc c’est n’est pas vrai que la fonction `s_arrête` résout le problème de l’arrêt ☹️

- On peut répéter le même raisonnement pour n’importe quelle autre fonction candidate `s_arrête_2`, avec le même résultat

- On peut conclure qu’il n’y a pas d’algorithme qui résout le problème

Ça servirait à quoi un algorithme pour le problème de l'arrêt ?

- Déjà on pourrait tricher à l'examen de Science informatique, pour les questions sur la terminaison des algorithmes 😉

- Plus en général, ça arrive de se tromper en écrivant du code et que parfois nos programmes ne terminent pas quand il faudrait, donc ce serait un outil pratique

- Et on peut démontrer que plein de problèmes de débogage (debugging) sont équivalents au problème de l'arrêt

Problèmes solubles avec un algorithme pour l'arrêt

- Est-ce que tôt ou tard la variable x prend la valeur 7 dans ce programme ?

- Est-ce que ce programme affiche quelque chose à l'écran ?

- Est-ce que la proposition mathématique formalisée en entrée (par exemple, $\forall n > 2 \ \nexists a, b, c : a^n + b^n = c^n$) est démontrable ou non ?

- Établir si ce dernier problème admet un algorithme s'appelle le Entscheidungsproblem (problème de décision), est c'est justement pour y répondre « non » que Turing a inventé sa machine !

Est-ce que ce programme s’arrête sur toute entrée ?

```
def collatz(n):
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3*n + 1
print("c'est bon")
```

Exercice 5 du TD7

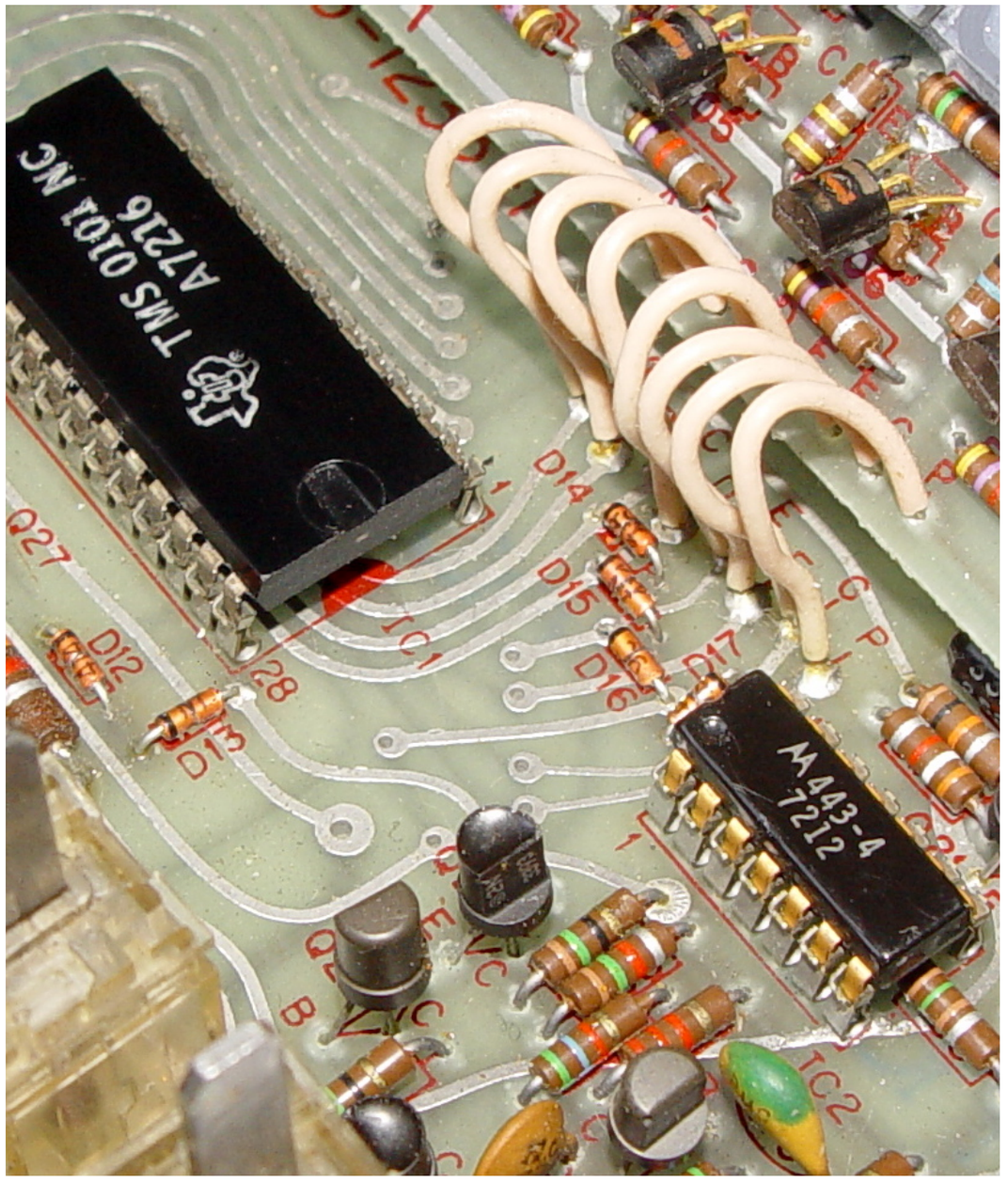
Modèles de calcul non conventionnels

Ordinateurs électroniques

- Construits avec des circuits électriques…
- …qui sont basés,
  entre autre, sur les lois de l’électromagnétisme
- Par exemple la loi d’Ohm :

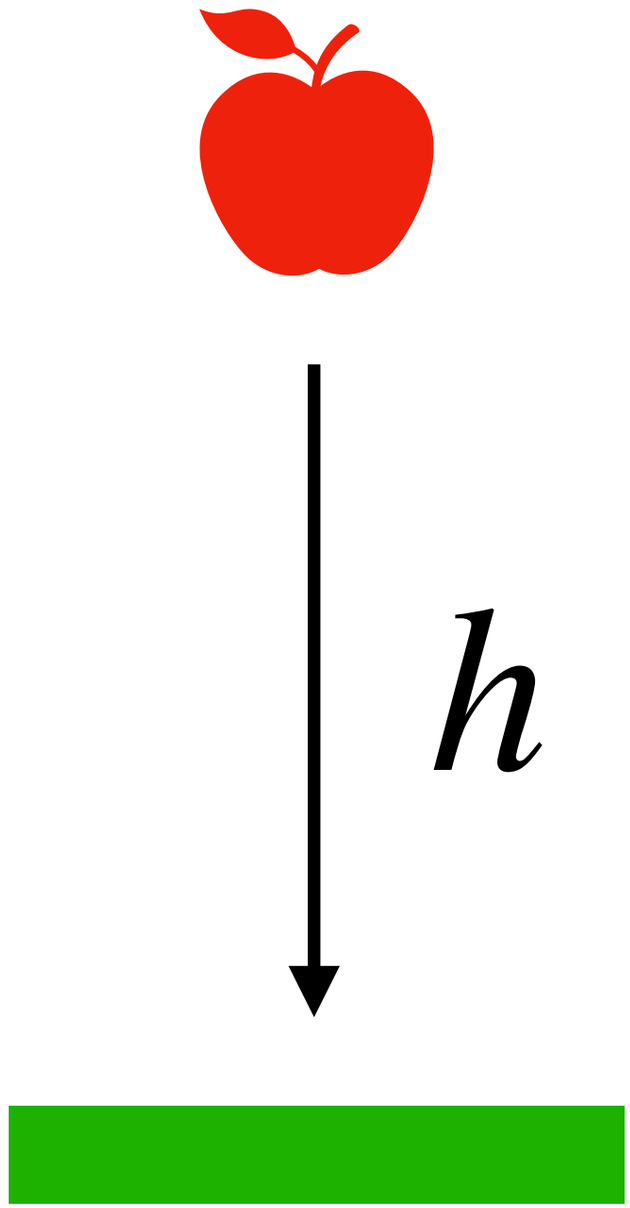
$$U = R \times I$$

tension — **résistance** — **intensité de courant**

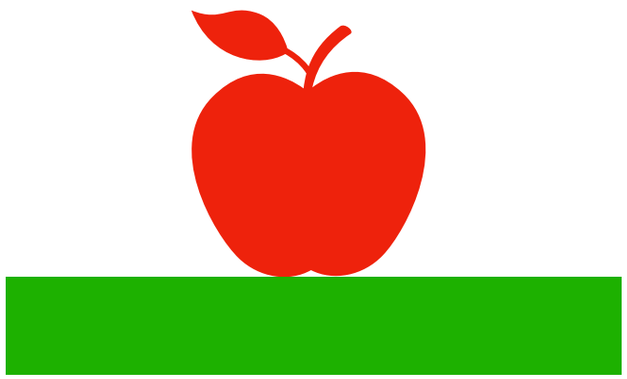


Mais on peut aussi exploiter d’autres phénomènes physiques !

Calculer avec la gravité

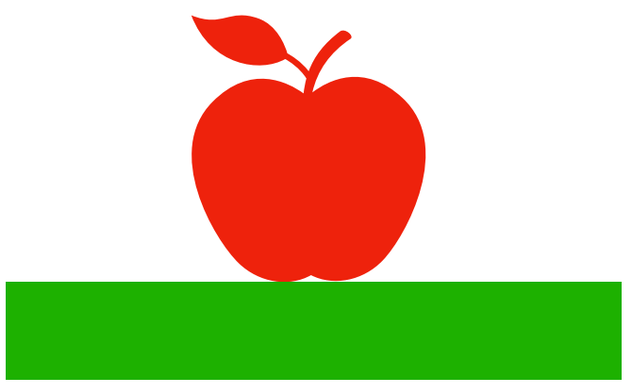


Calculer avec la gravité



Calculer avec la gravité

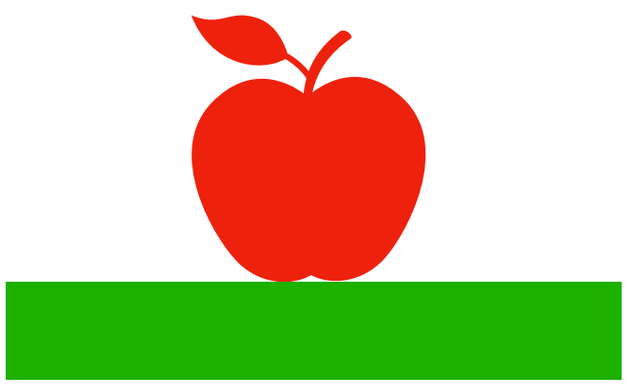
$$t = \sqrt{\frac{2h}{g}}$$



Calculer avec la gravité

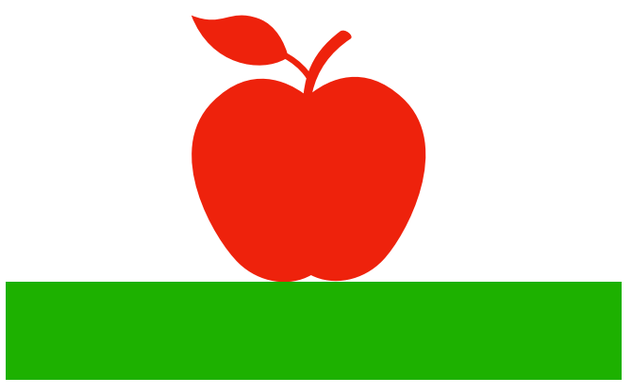
$$t = \sqrt{\frac{2h}{g}}$$

g : 9,81 m/s^2



Calculer avec la gravité

$$t = \sqrt{\frac{2h}{g}}$$

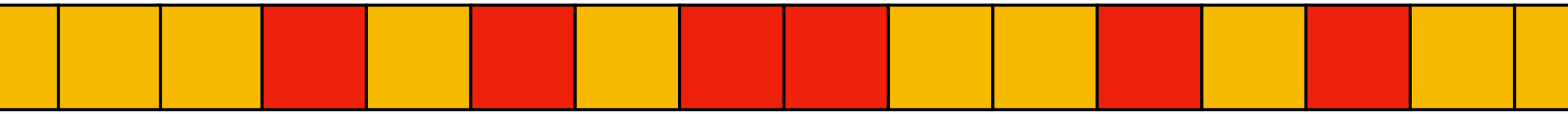


$$h = \frac{xg}{2}$$

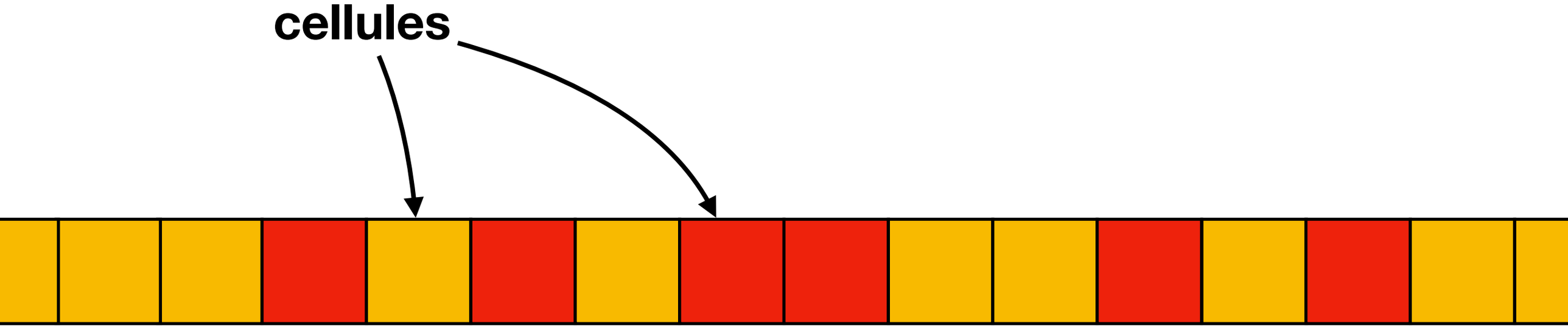
$$t = \sqrt{x}$$

Automates cellulaires

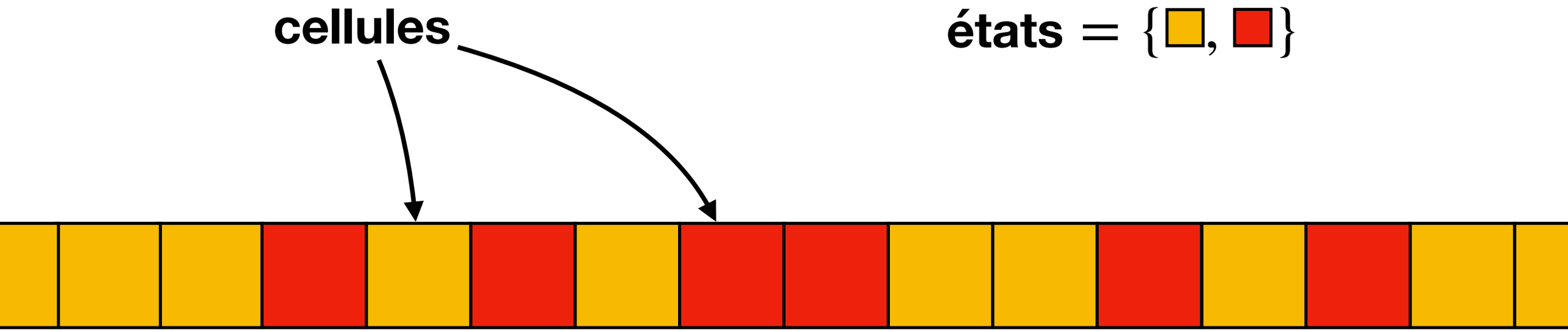
Automates cellulaires 1D



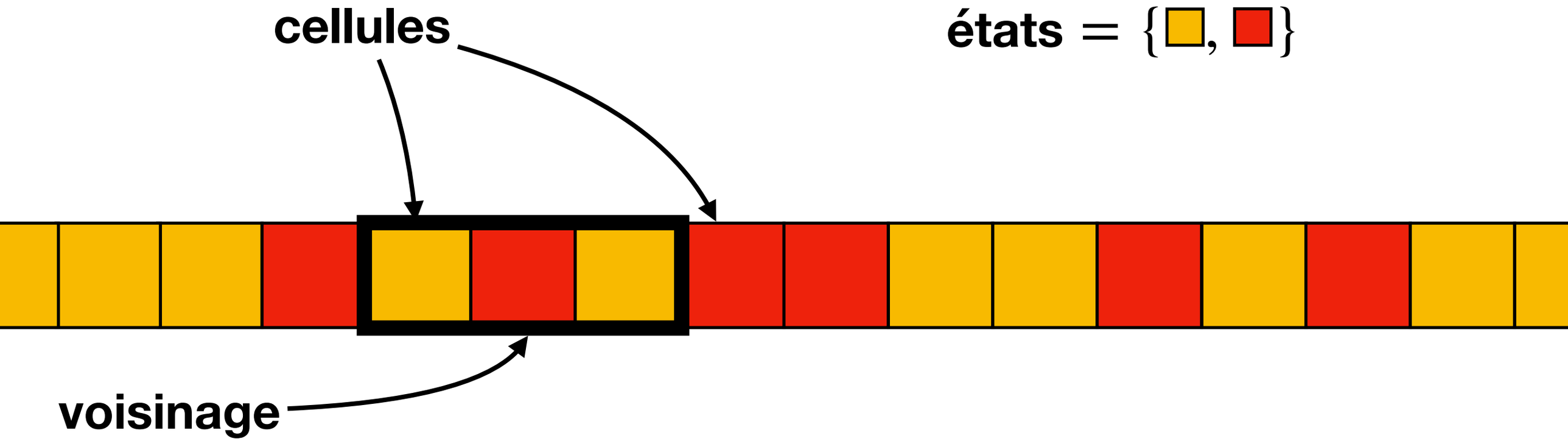
Automates cellulaires 1D



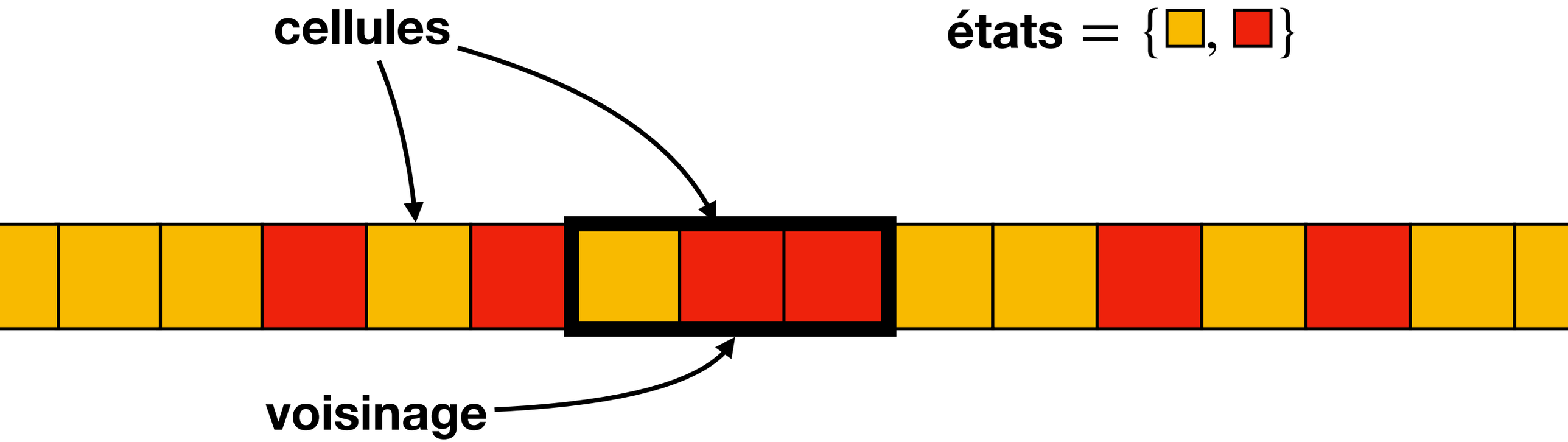
Automates cellulaires 1D



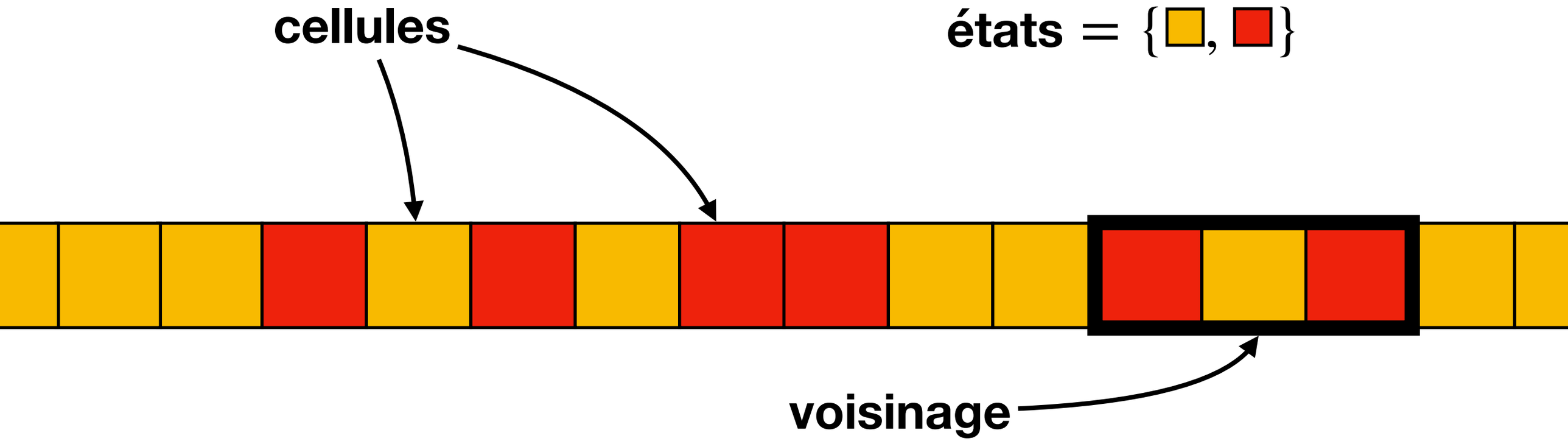
Automates cellulaires 1D



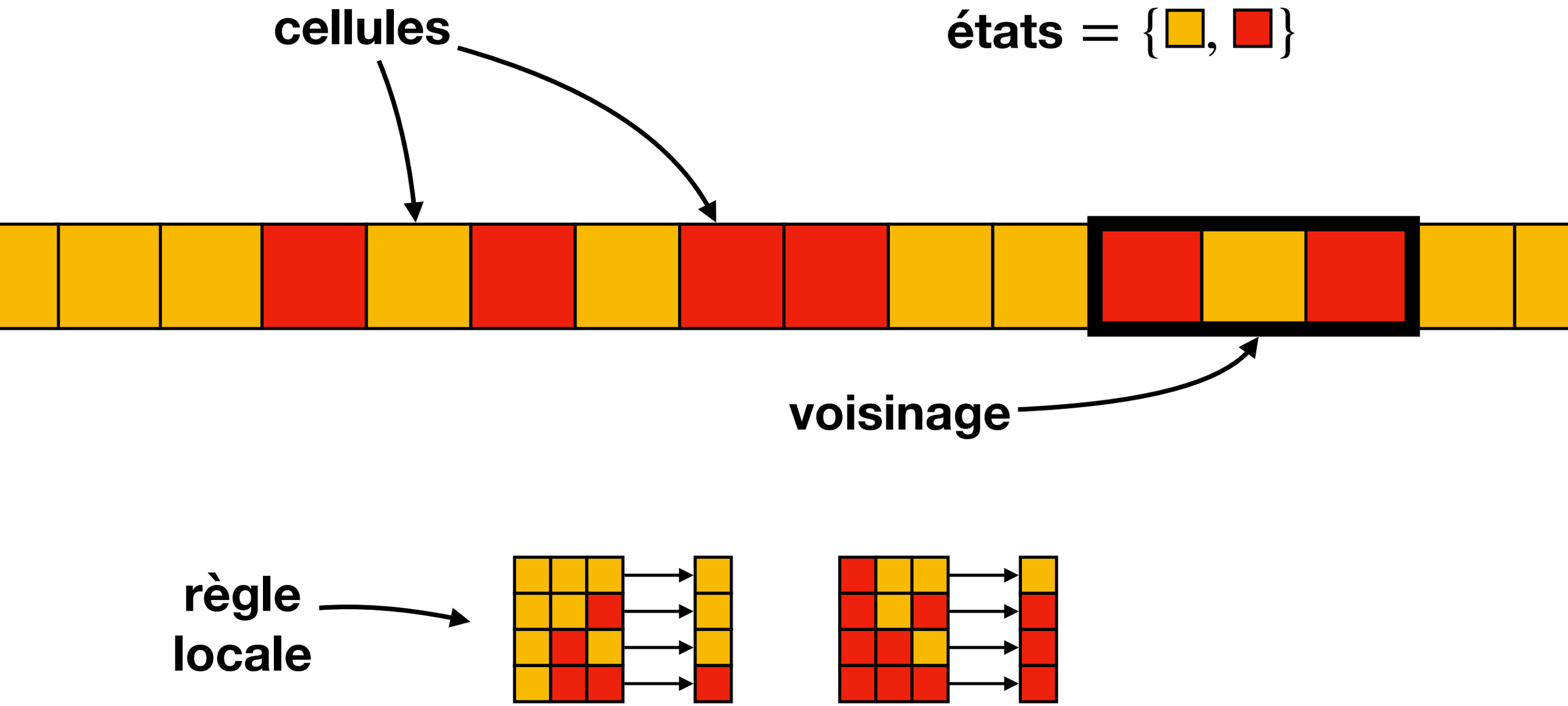
Automates cellulaires 1D



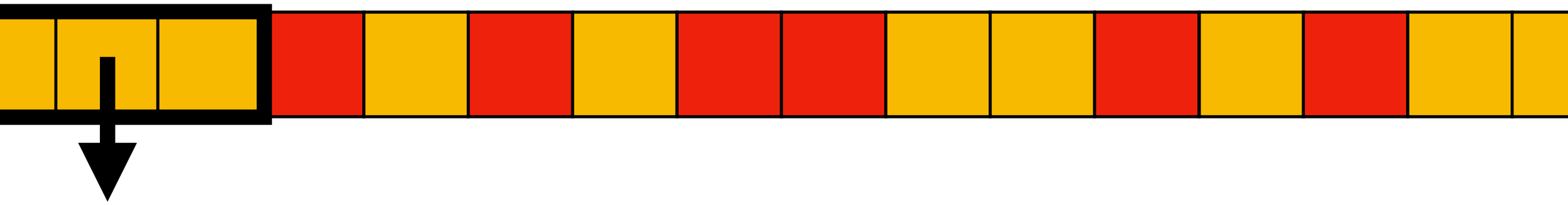
Automates cellulaires 1D



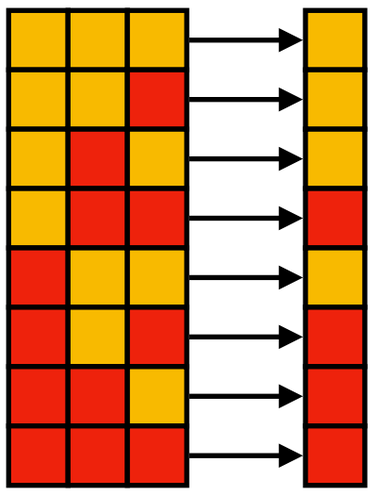
Automates cellulaires 1D



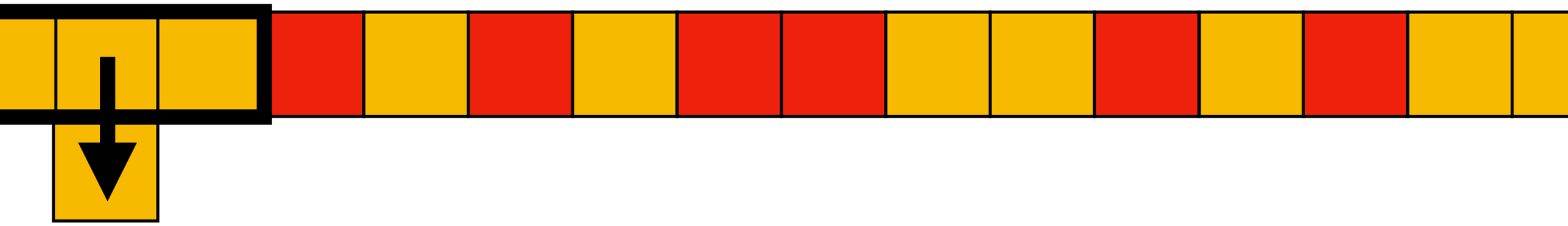
Automates cellulaires 1D



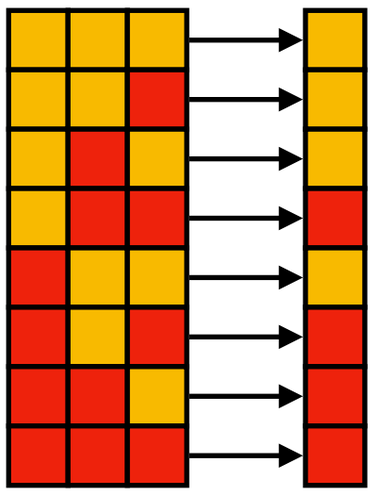
règle locale



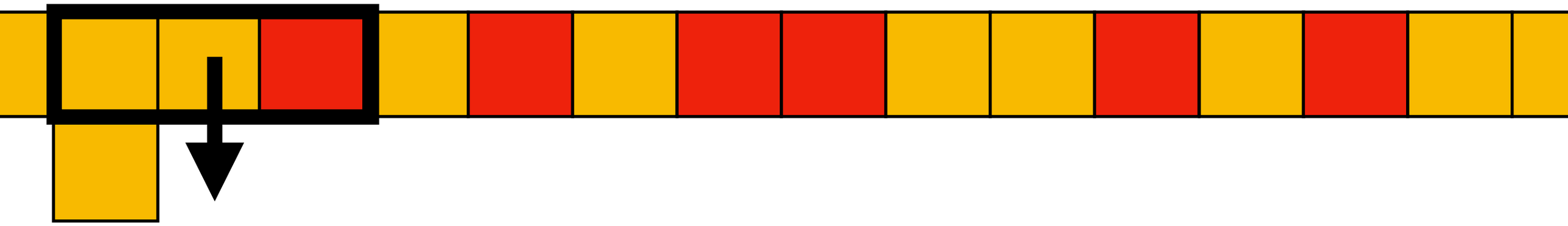
Automates cellulaires 1D



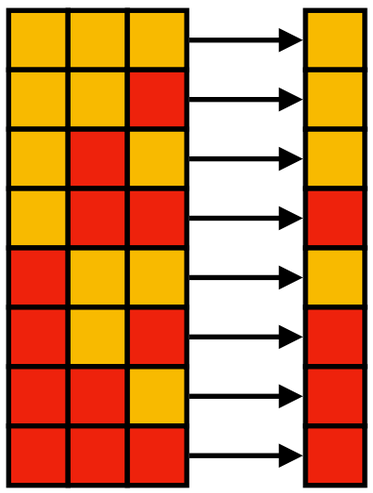
règle locale



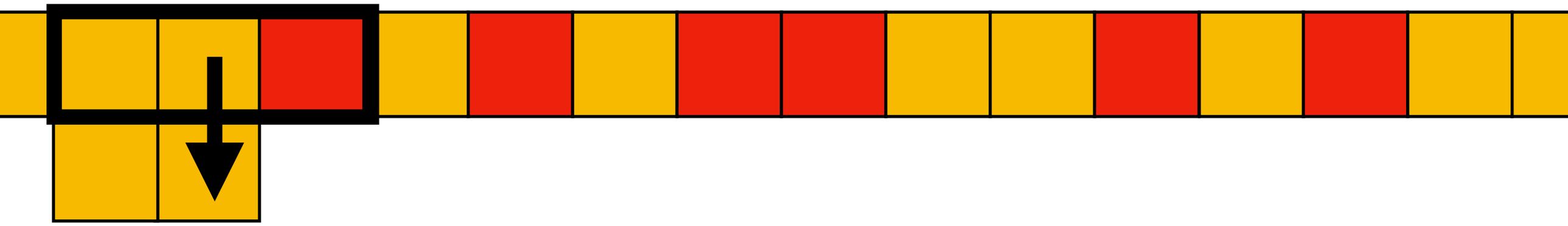
Automates cellulaires 1D



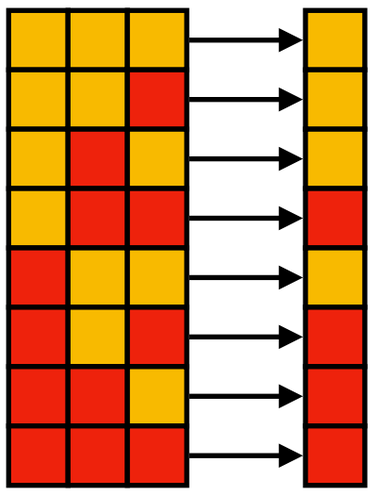
règle locale



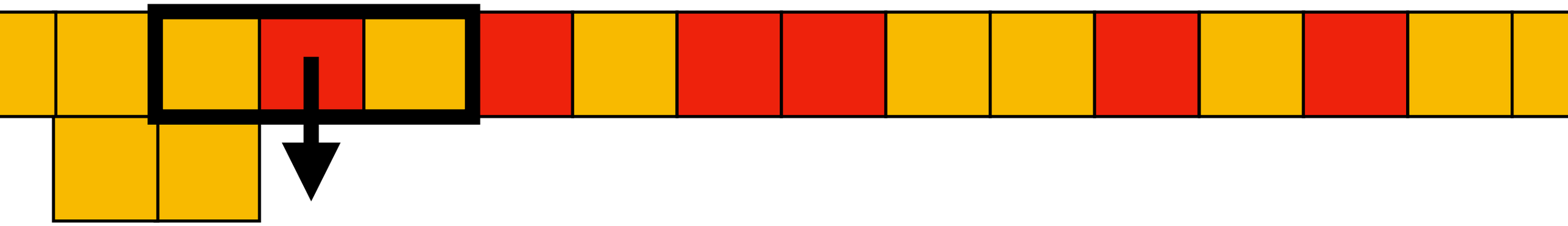
Automates cellulaires 1D



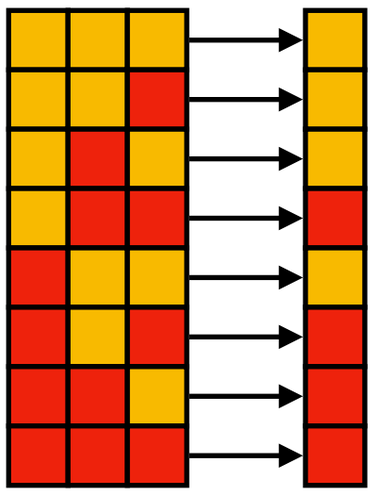
règle locale



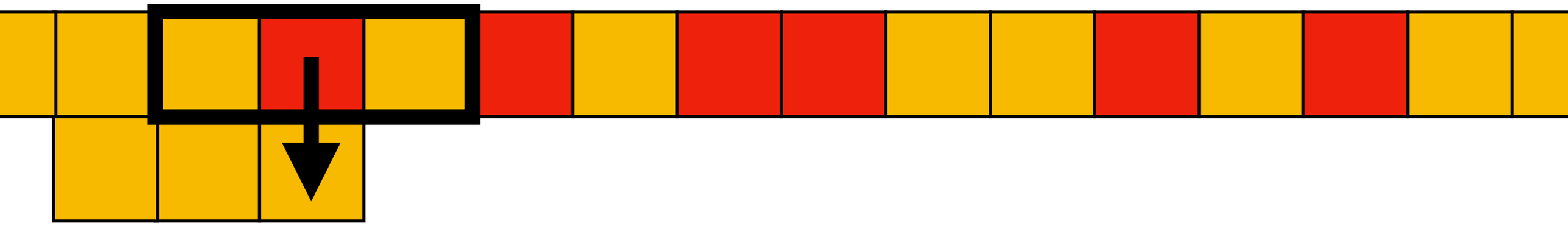
Automates cellulaires 1D



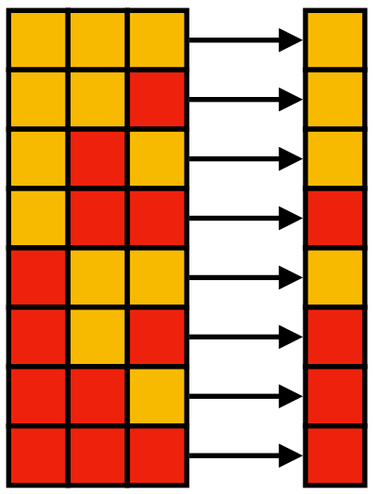
règle locale



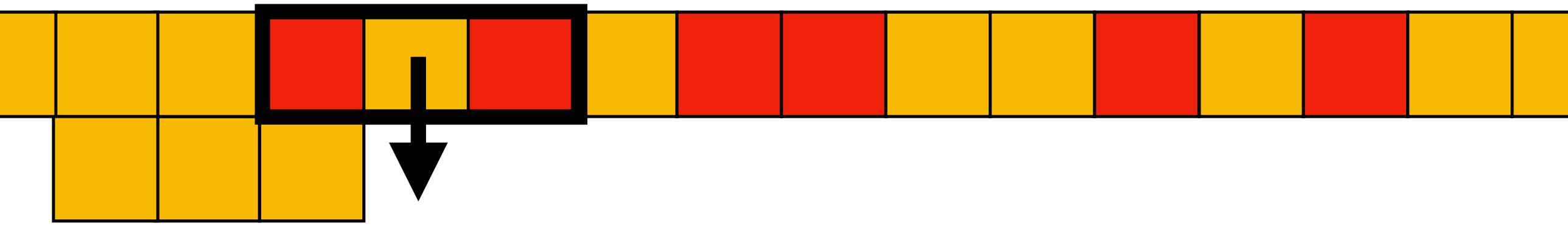
Automates cellulaires 1D



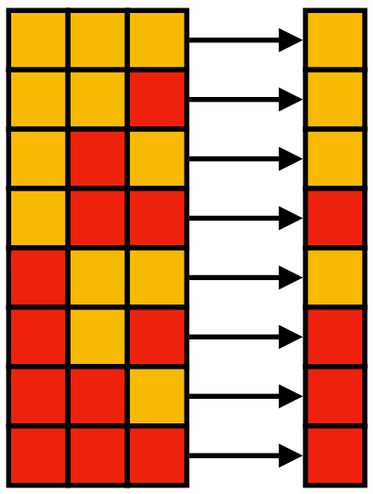
règle locale



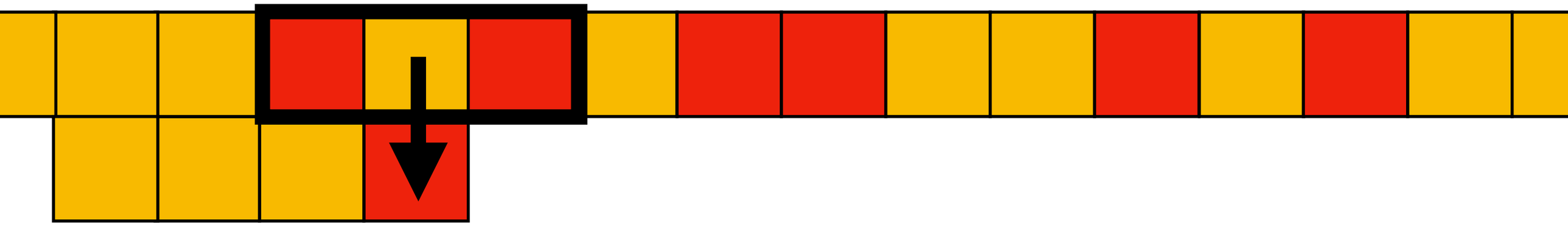
Automates cellulaires 1D



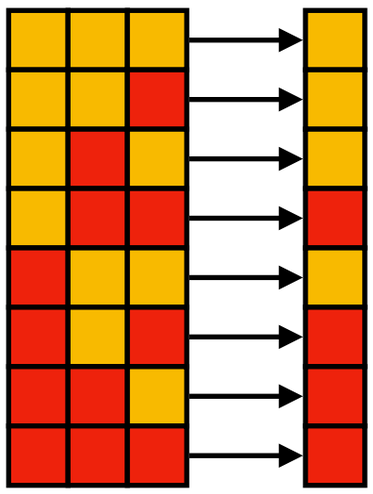
règle locale



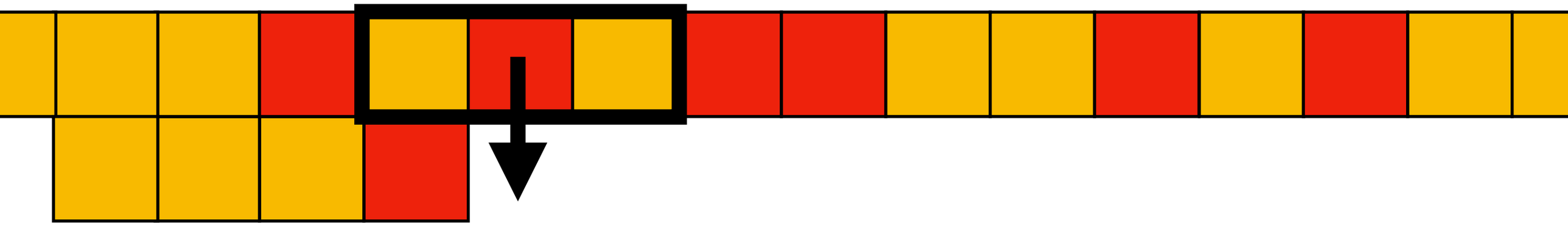
Automates cellulaires 1D



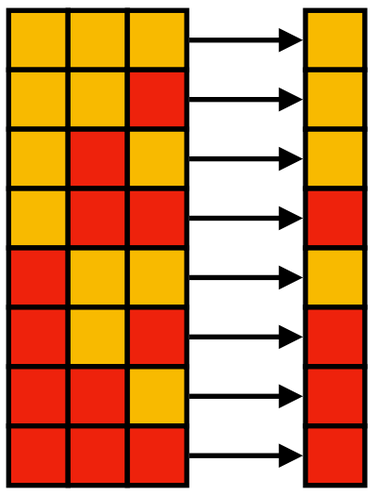
règle locale



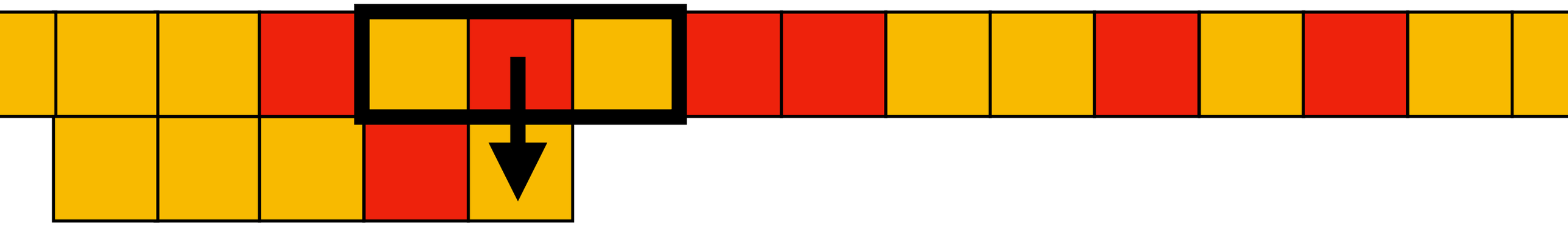
Automates cellulaires 1D



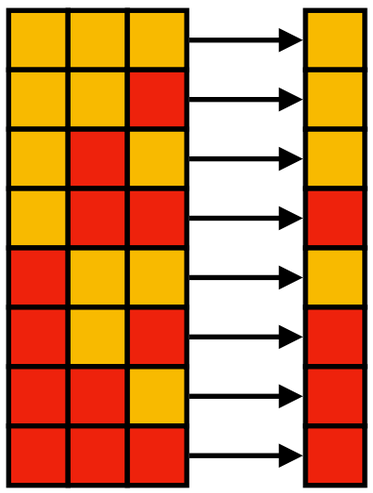
règle locale



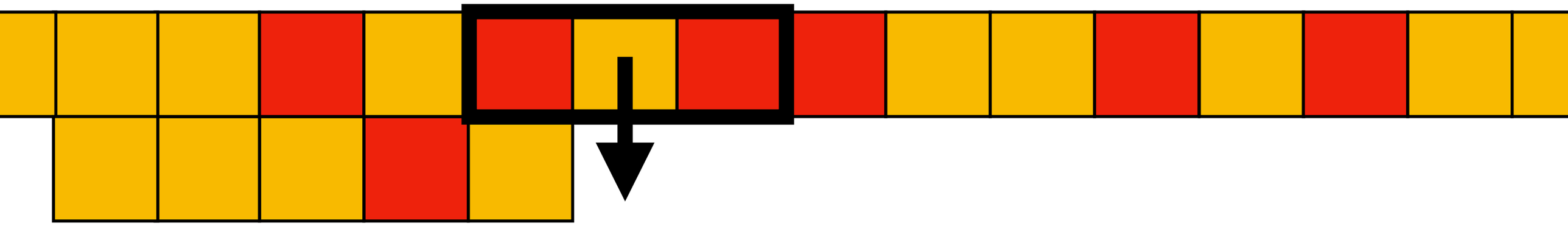
Automates cellulaires 1D



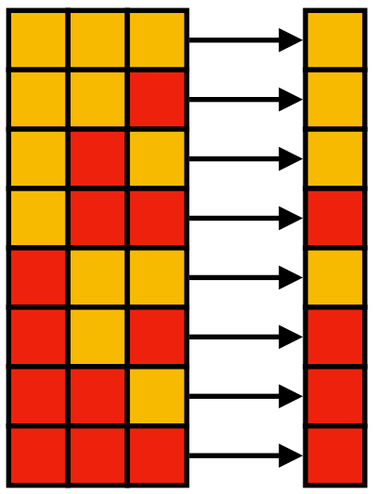
règle locale



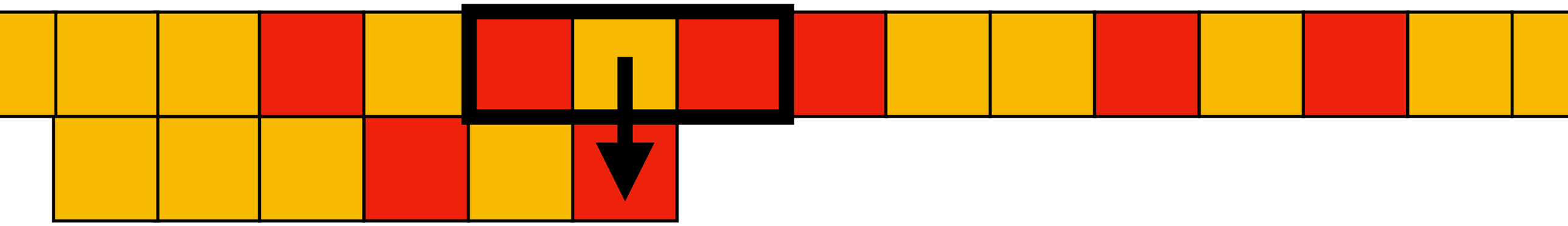
Automates cellulaires 1D



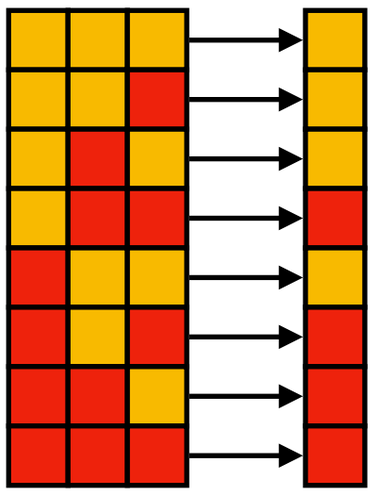
règle locale



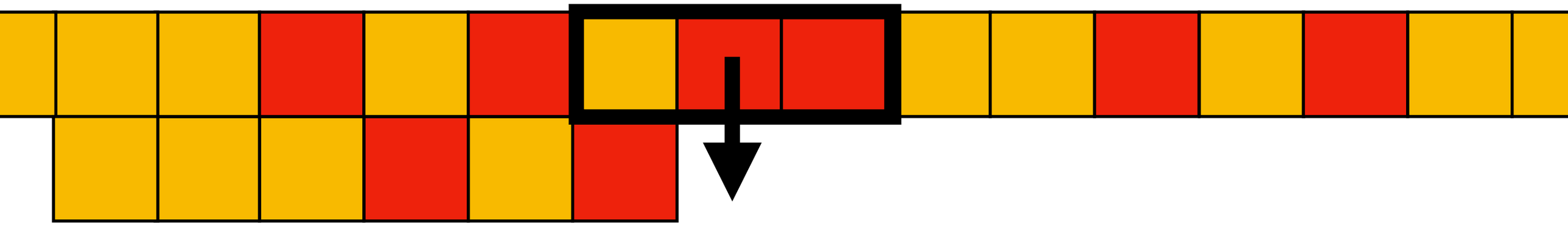
Automates cellulaires 1D



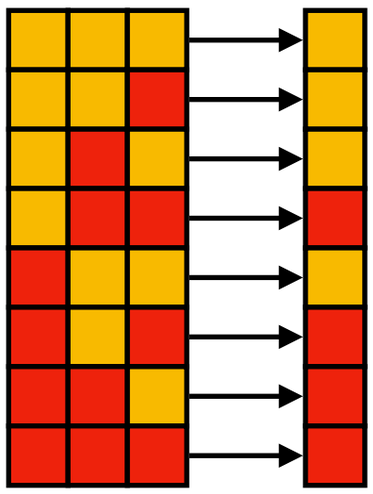
règle locale



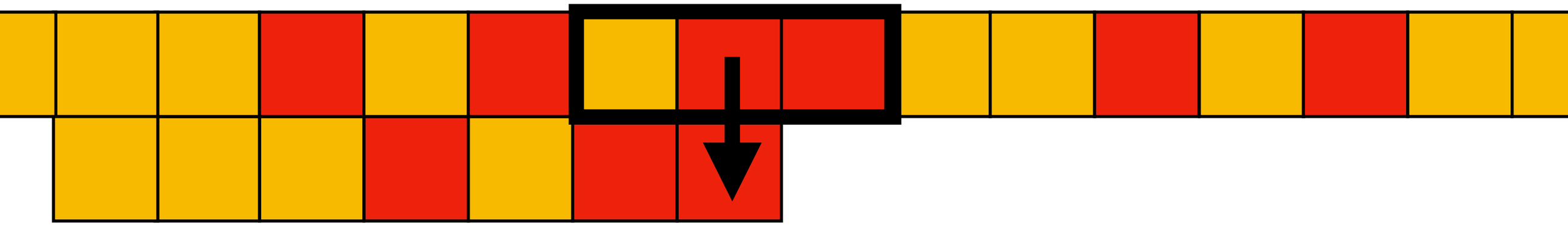
Automates cellulaires 1D



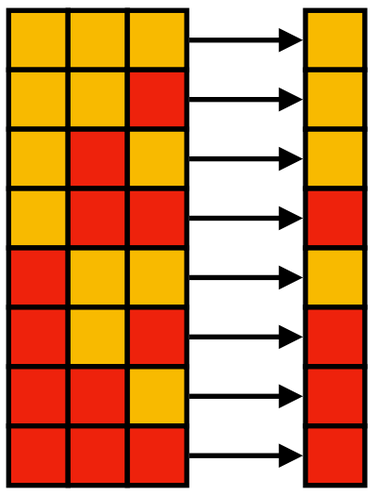
règle locale



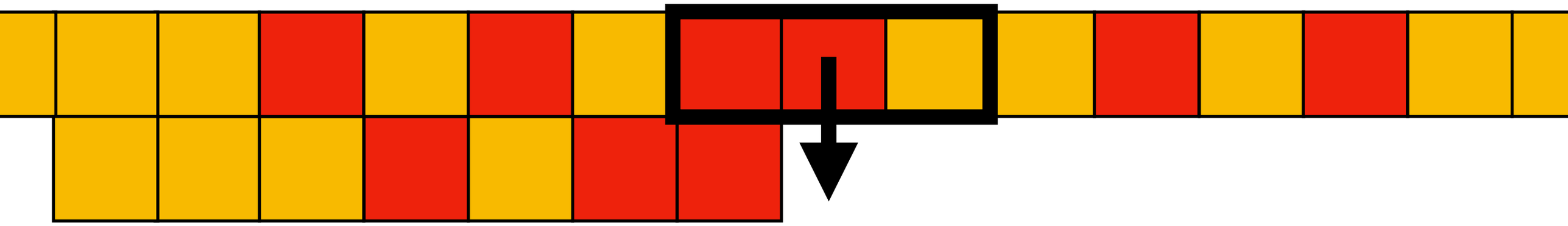
Automates cellulaires 1D



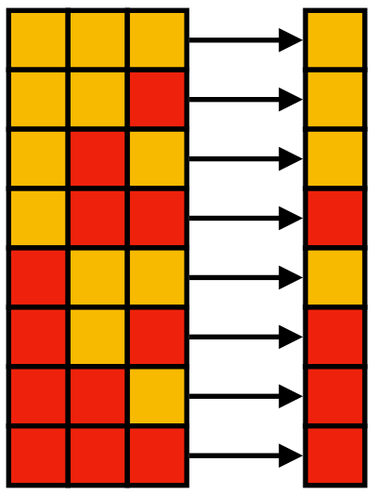
règle locale



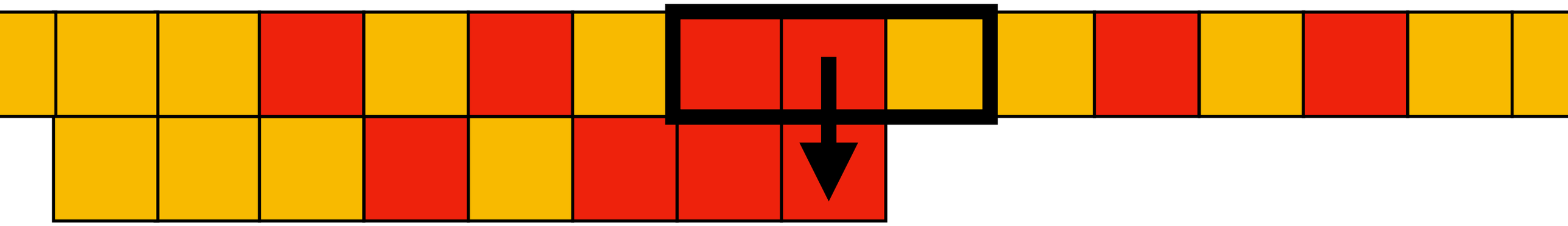
Automates cellulaires 1D



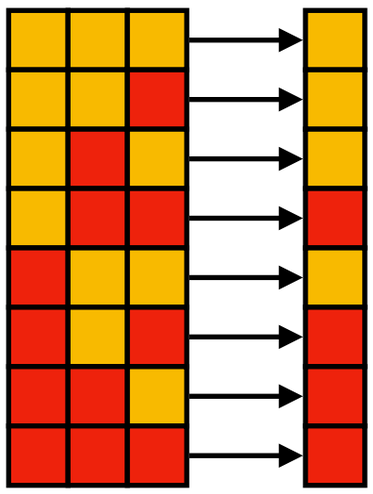
règle locale



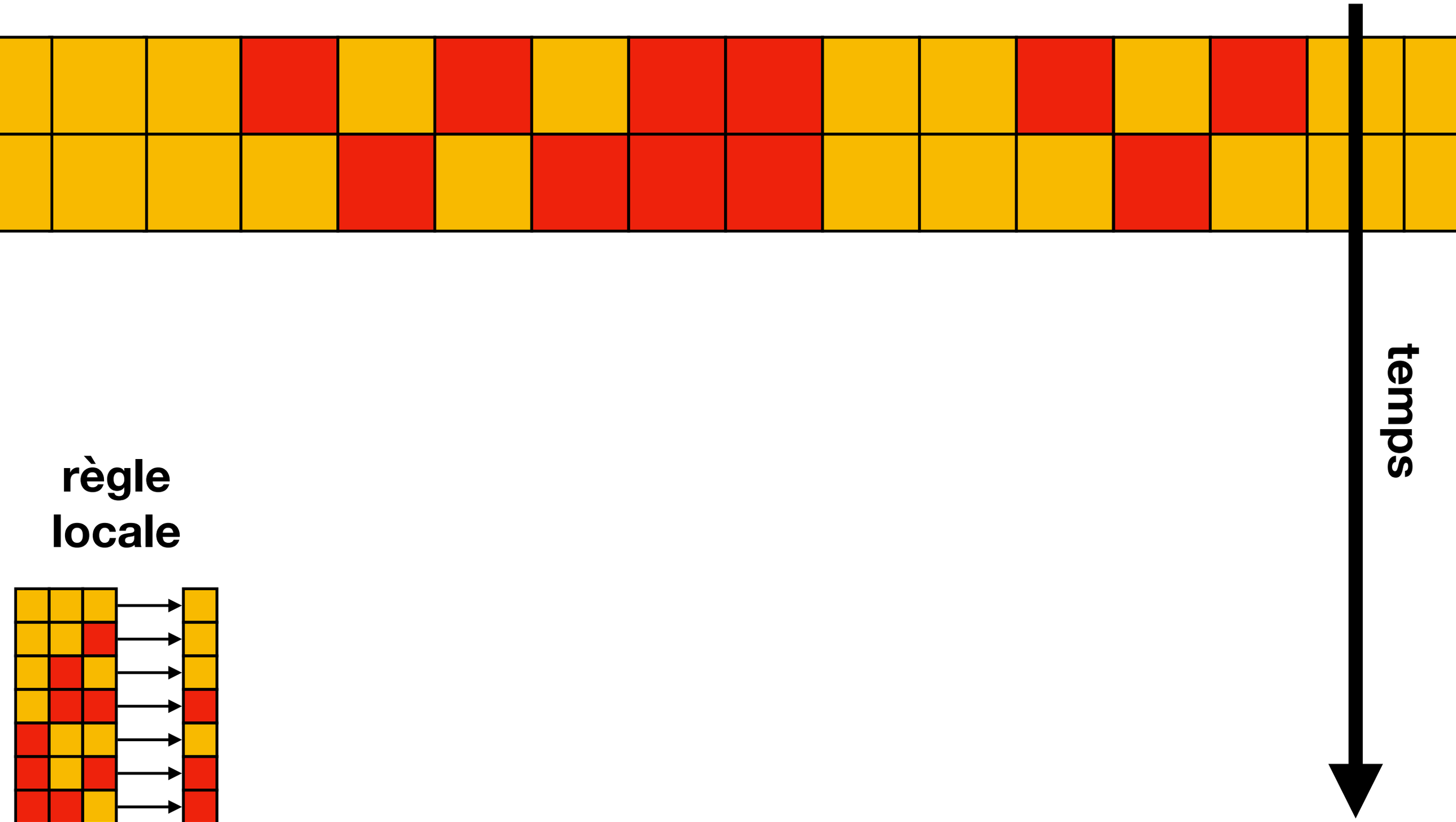
Automates cellulaires 1D



règle locale

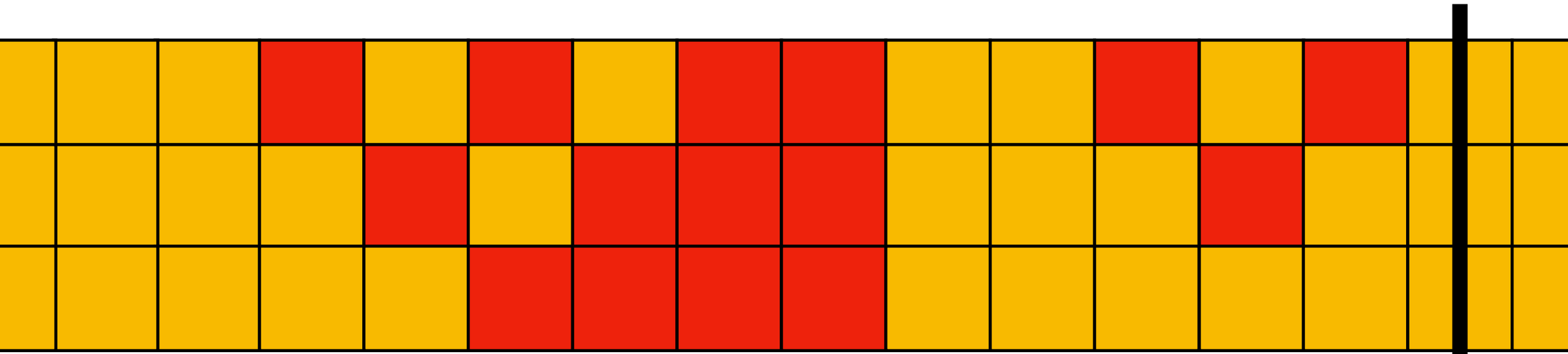


Automates cellulaires 1D

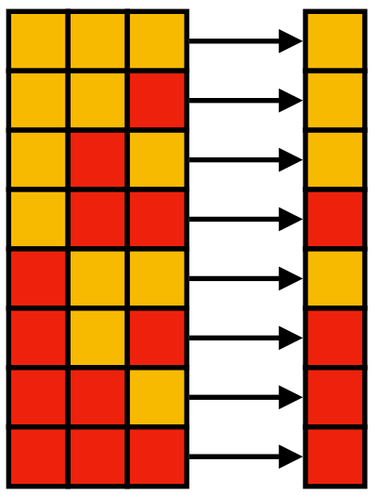


règle locale

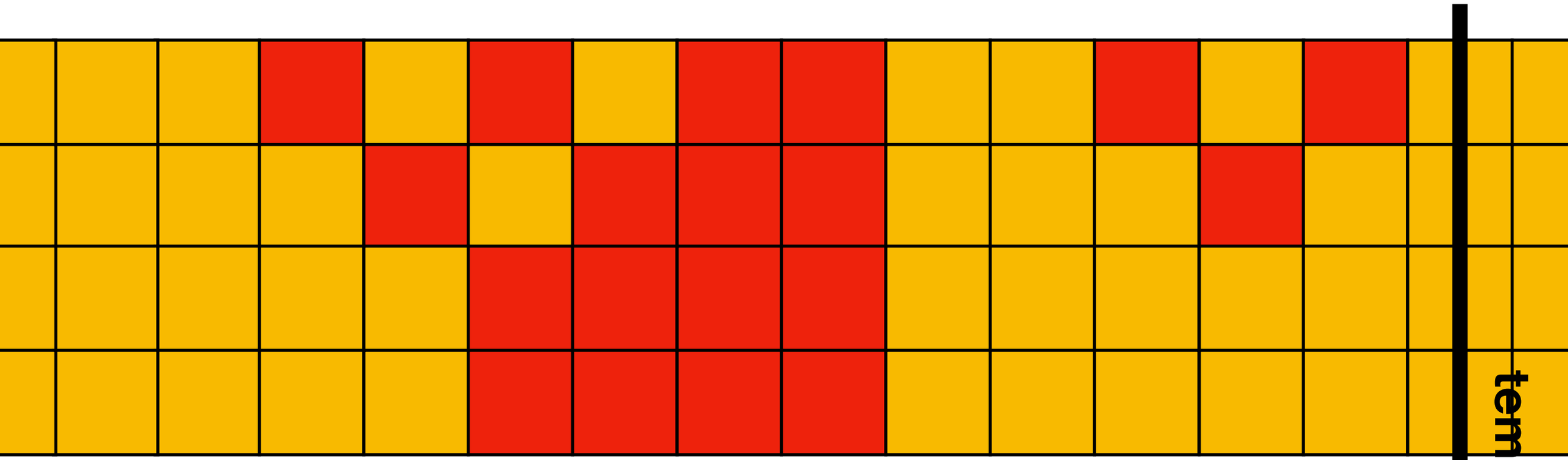
Automates cellulaires 1D



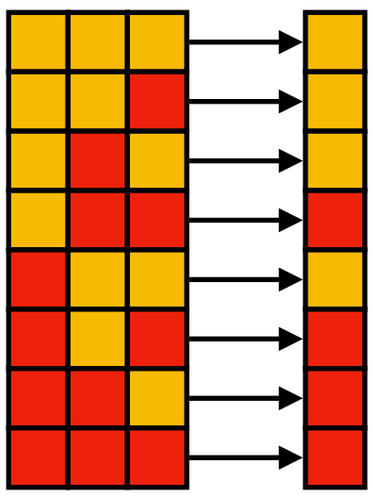
règle locale



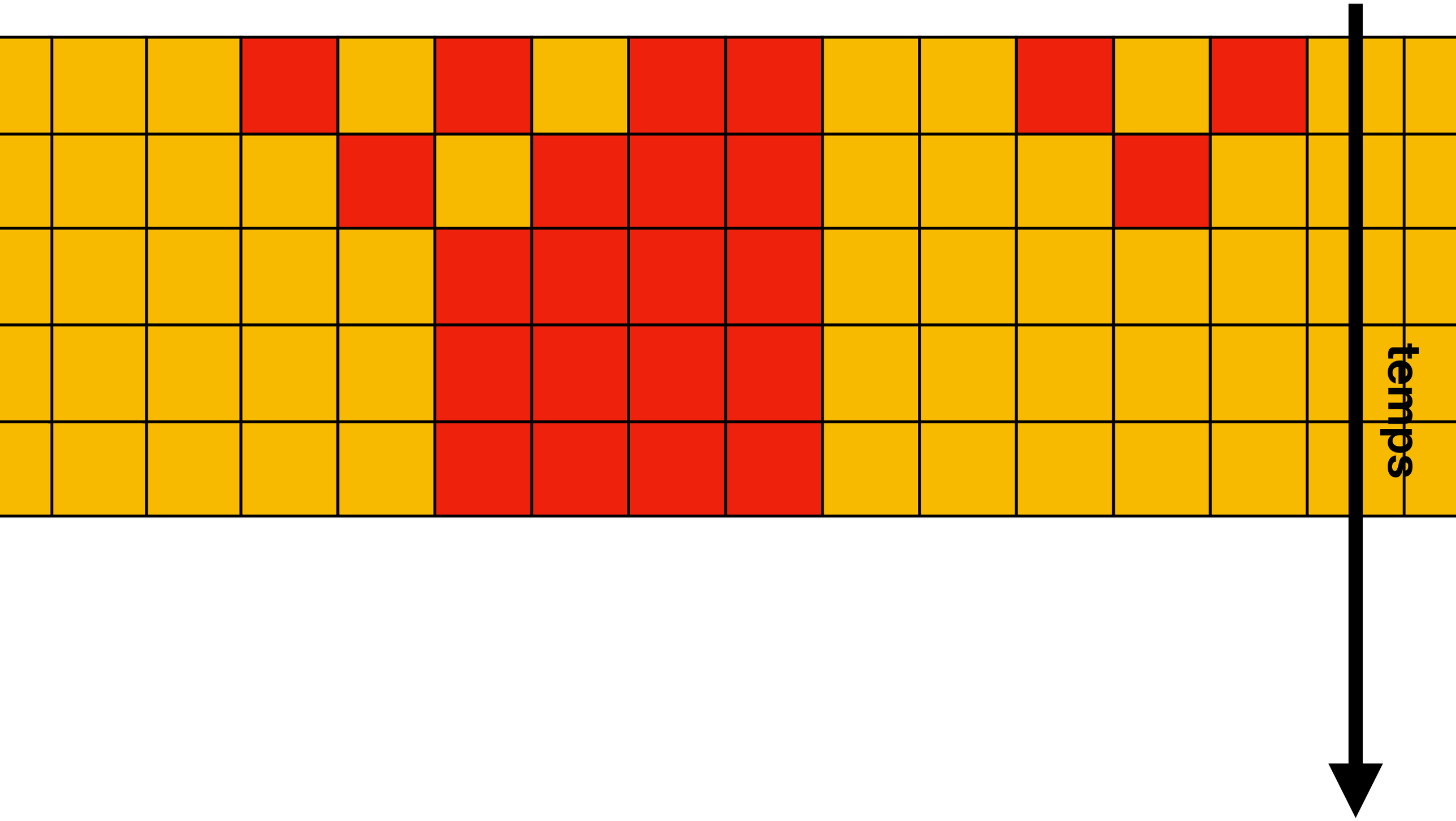
Automates cellulaires 1D



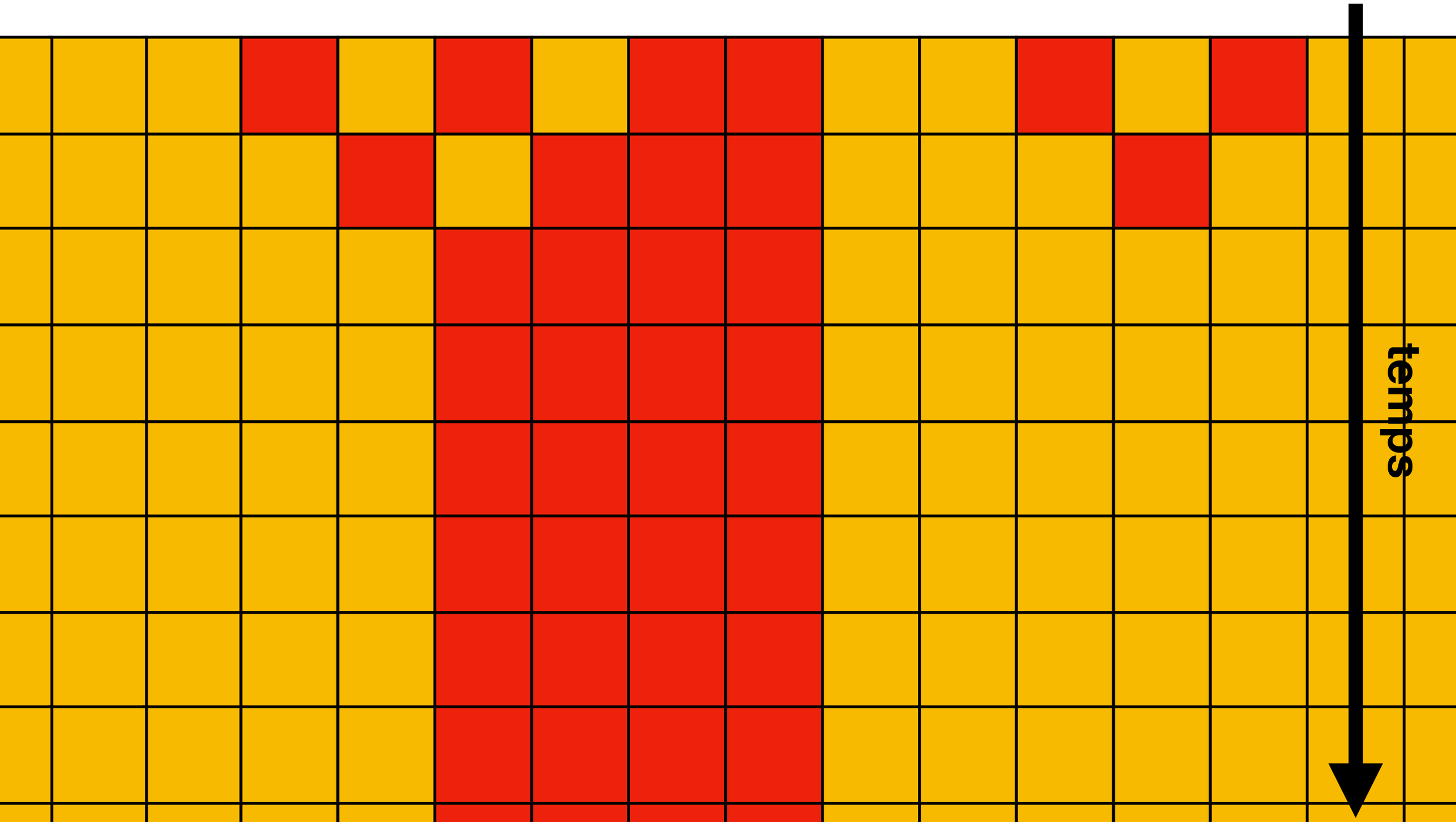
règle locale



Automates cellulaires 1D

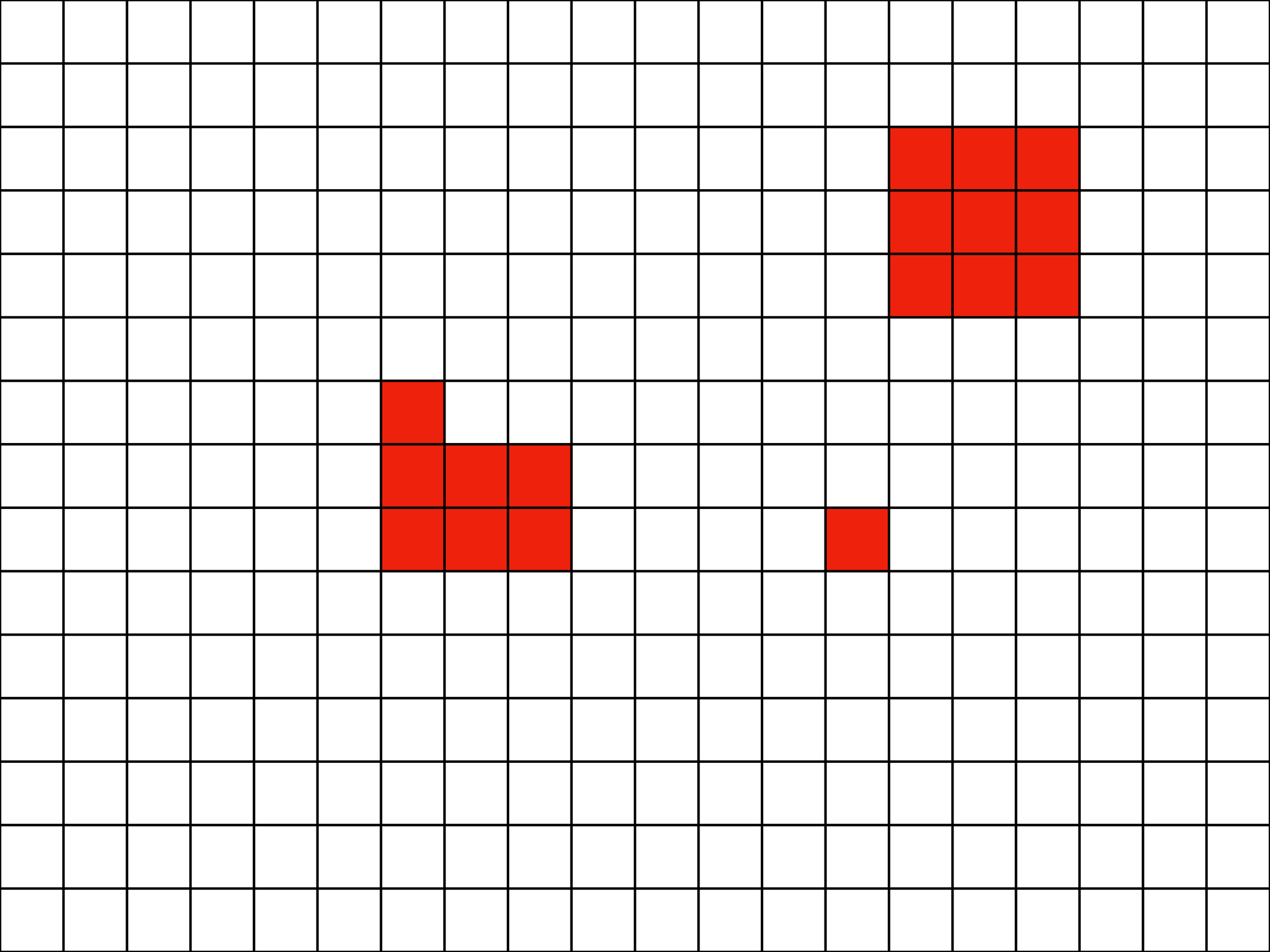


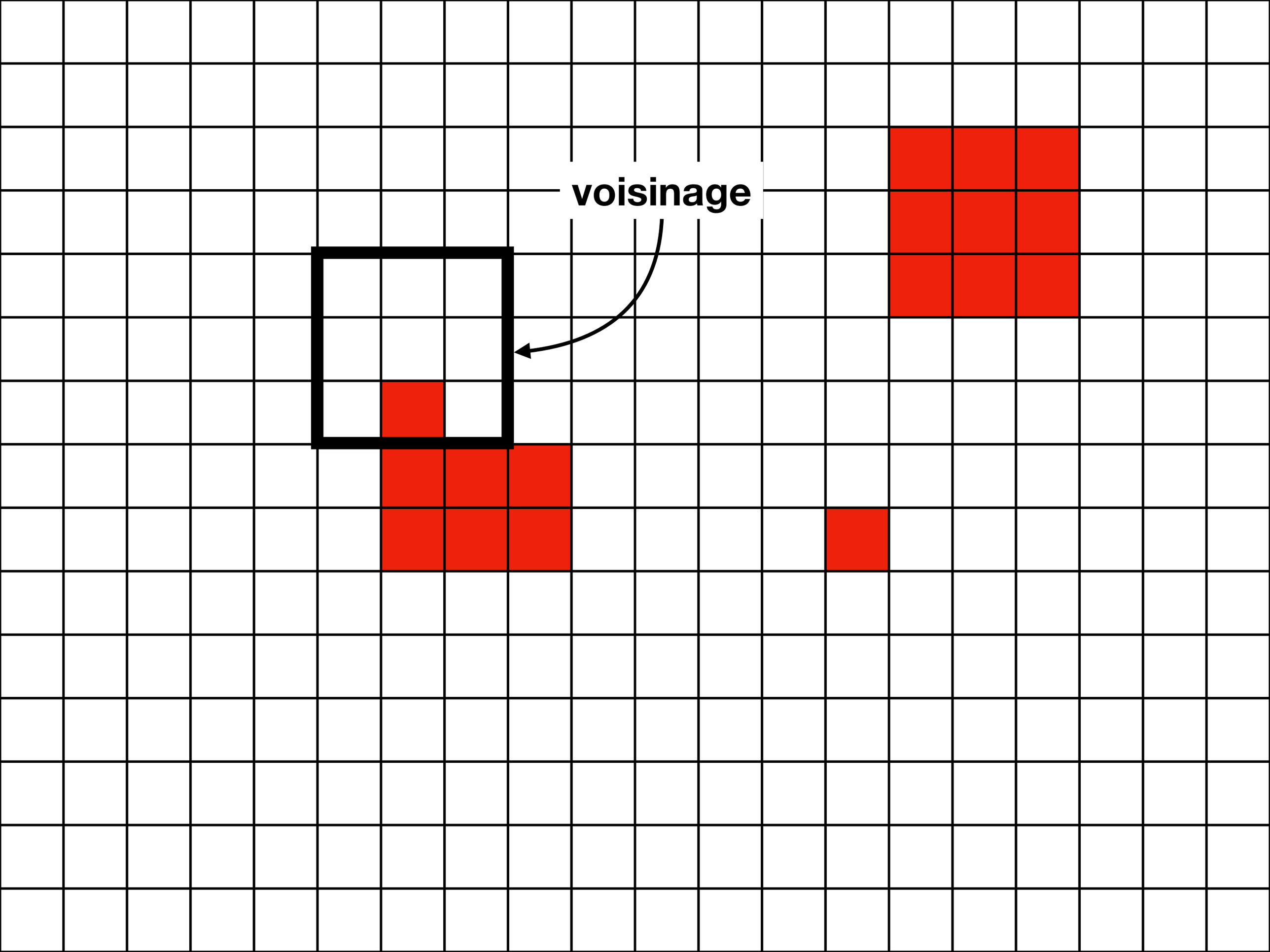
Automates cellulaires 1D



Exercice 1 du TD8

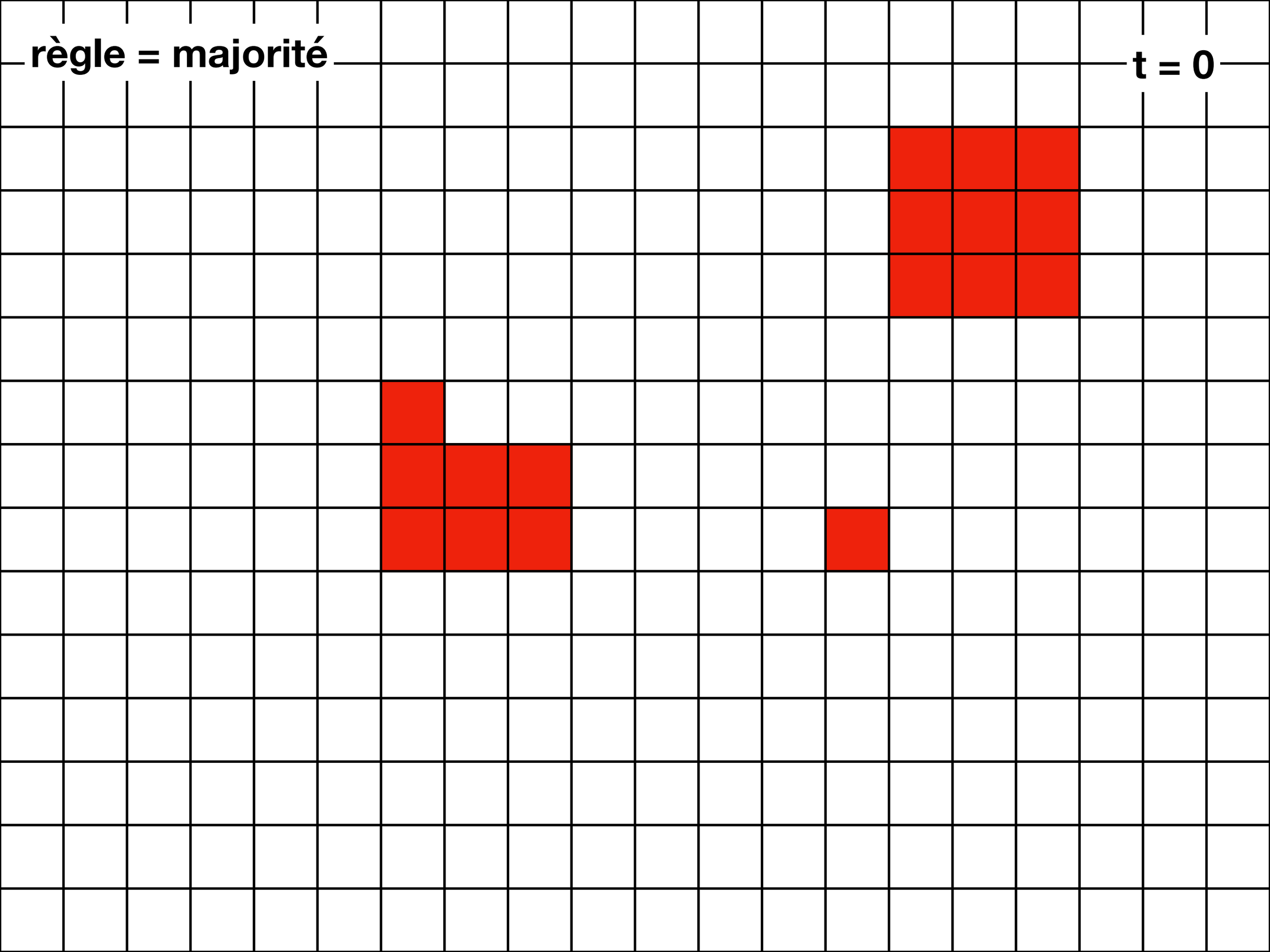
Automates cellulaires 2D




voisinage

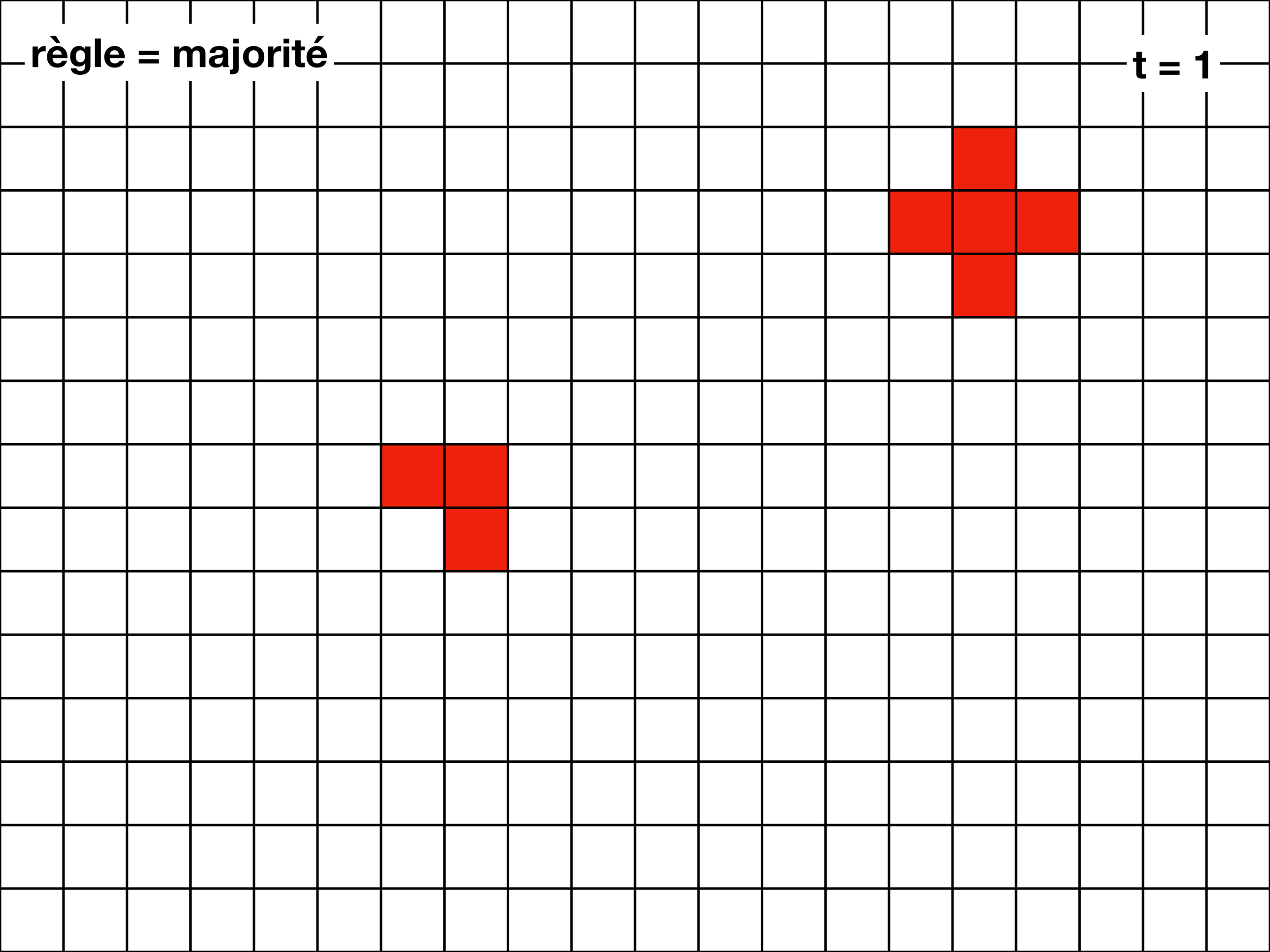
règle = majorité

t = 0



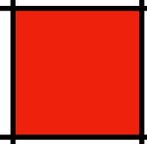
règle = majorité

t = 1



règle = majorité

t = 2



règle = majorité

t = 3

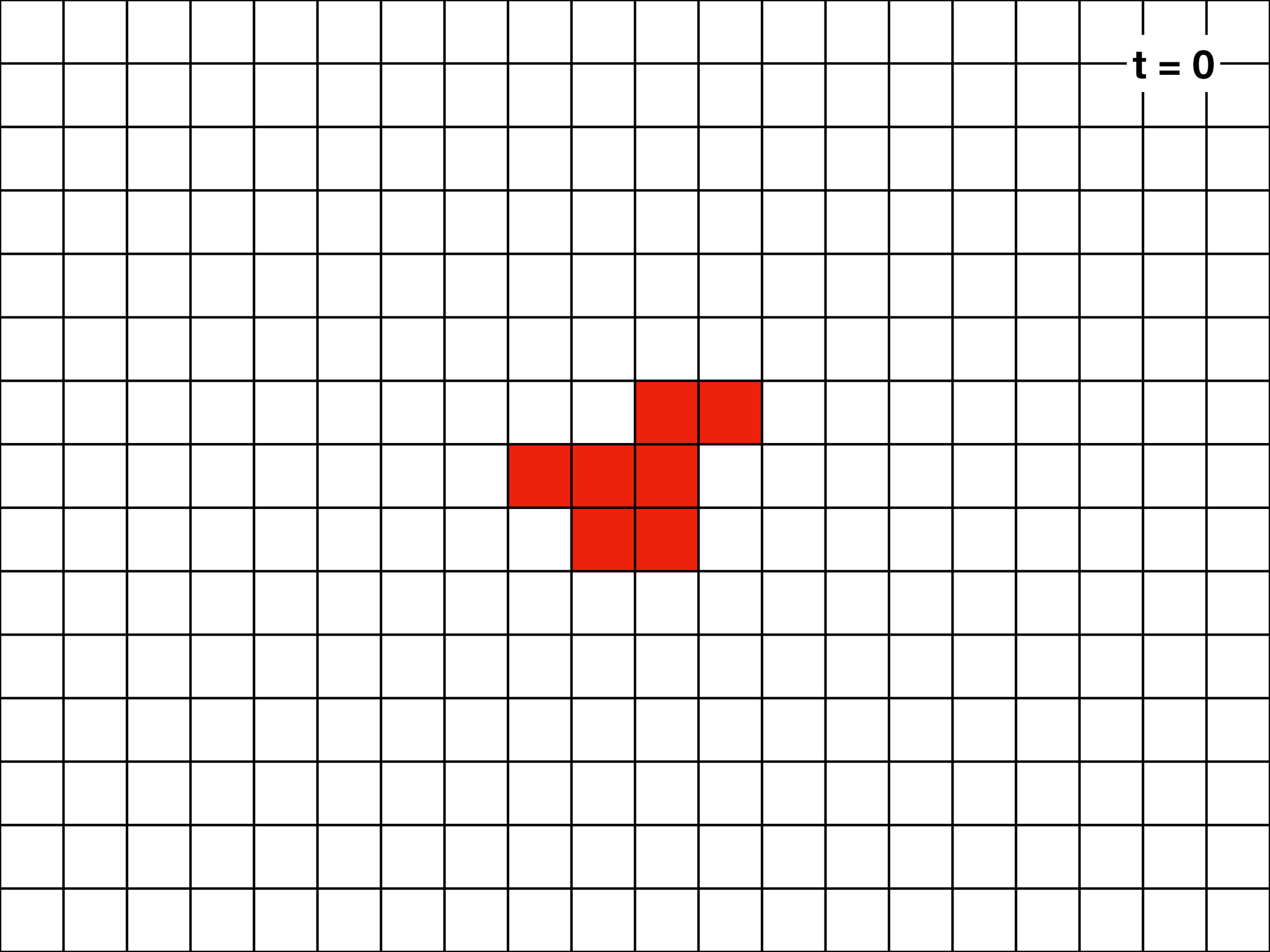
règle = majorité

t = 4

Le jeu de la vie

Les règles du jeu de la vie

- C’est un automate cellulaire 2D (à priori) infini
- Il y a deux états : vivant ■ et mort □
- Si une cellule vivante a 2 ou 3 voisins vivants, elle reste vivante
- Si une cellule vivante a moins de 2 voisins vivants
  ou plus de 3, elle meurt (par isolement ou surpopulation)
- Si une cellule morte a exactement 3 voisins vivants,
  elle devient vivante
- Les autres cellules mortes restent mortes


t = 0

t = 1

t = 2

t = 3

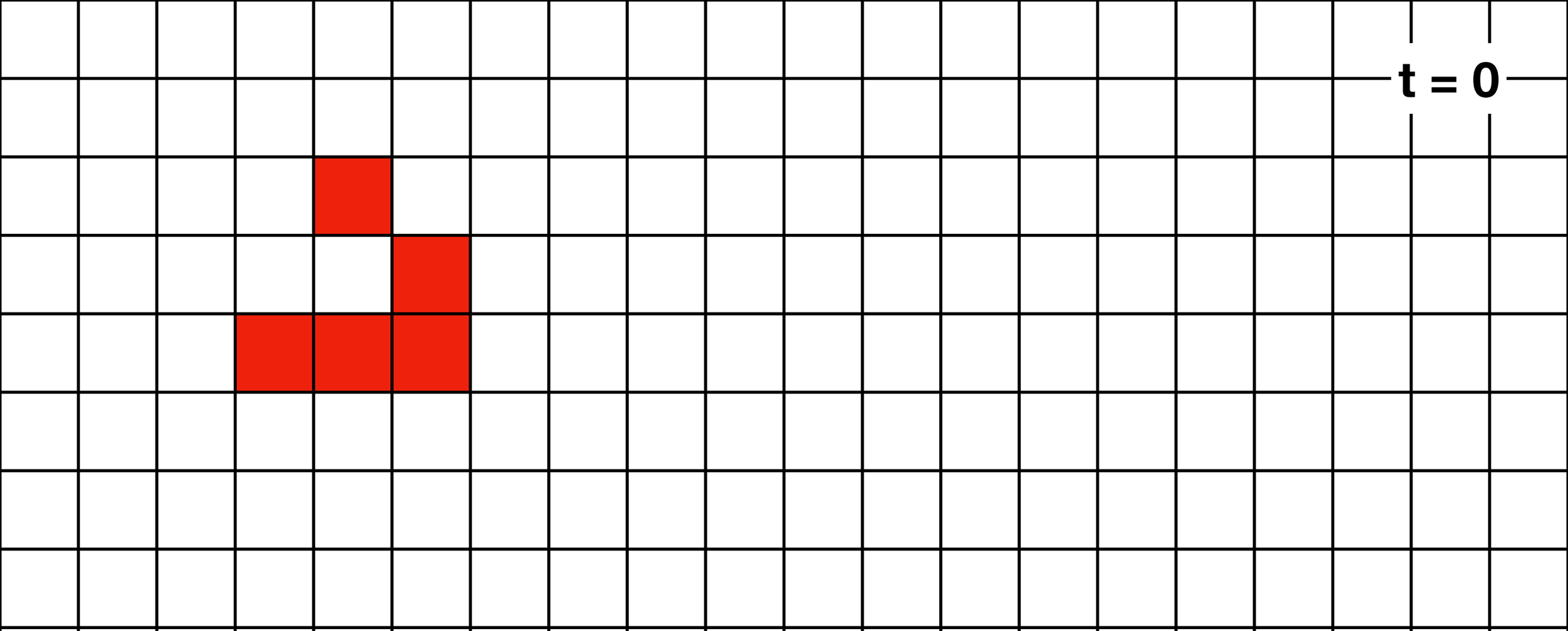
t = 4

Exercice 2 du TD8

Expériences sur

https://playgameoflife.com

Planeurs dans le jeu de la vie


t = 0

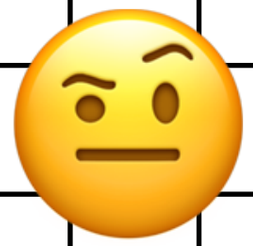
t = 1

t = 2

t = 3

t = 4

t = 4



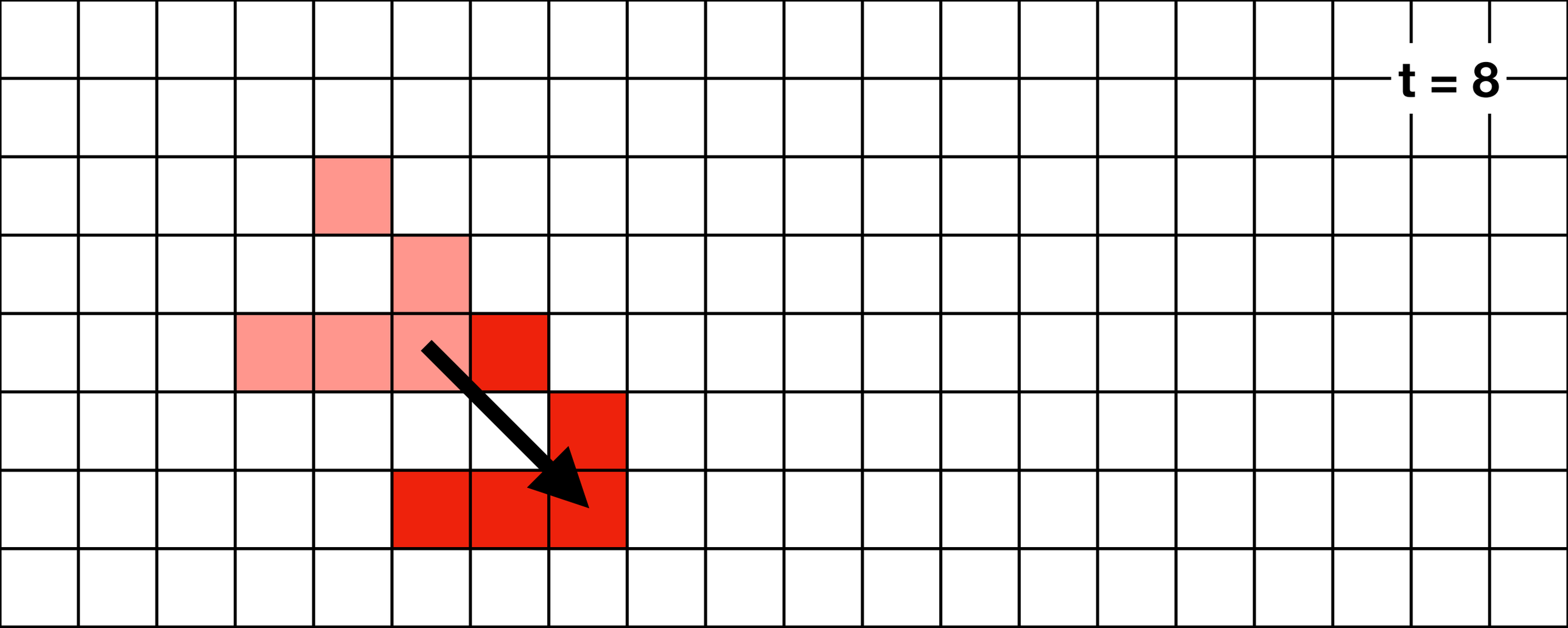
t = 5

t = 6

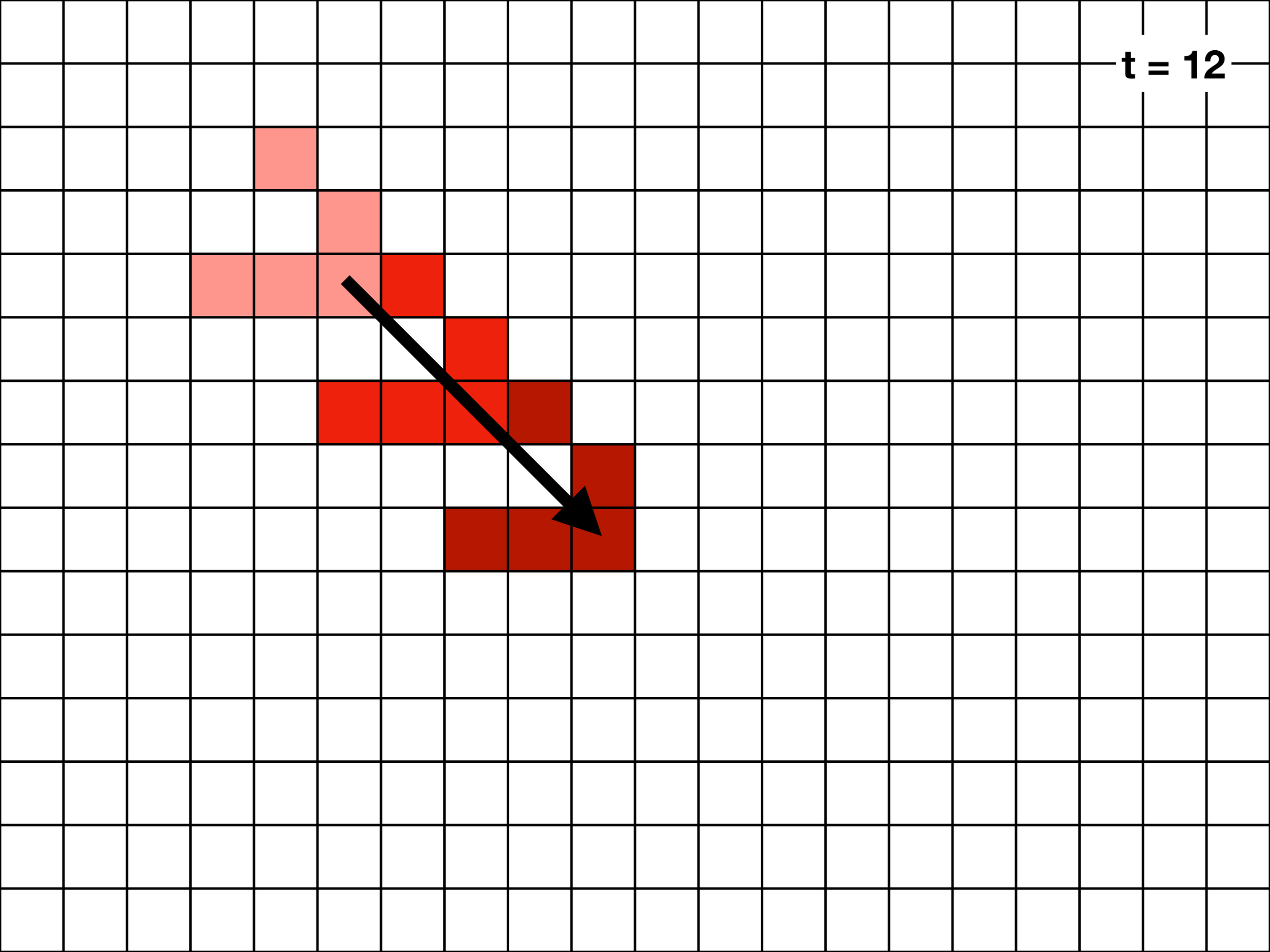
t = 7

t = 8

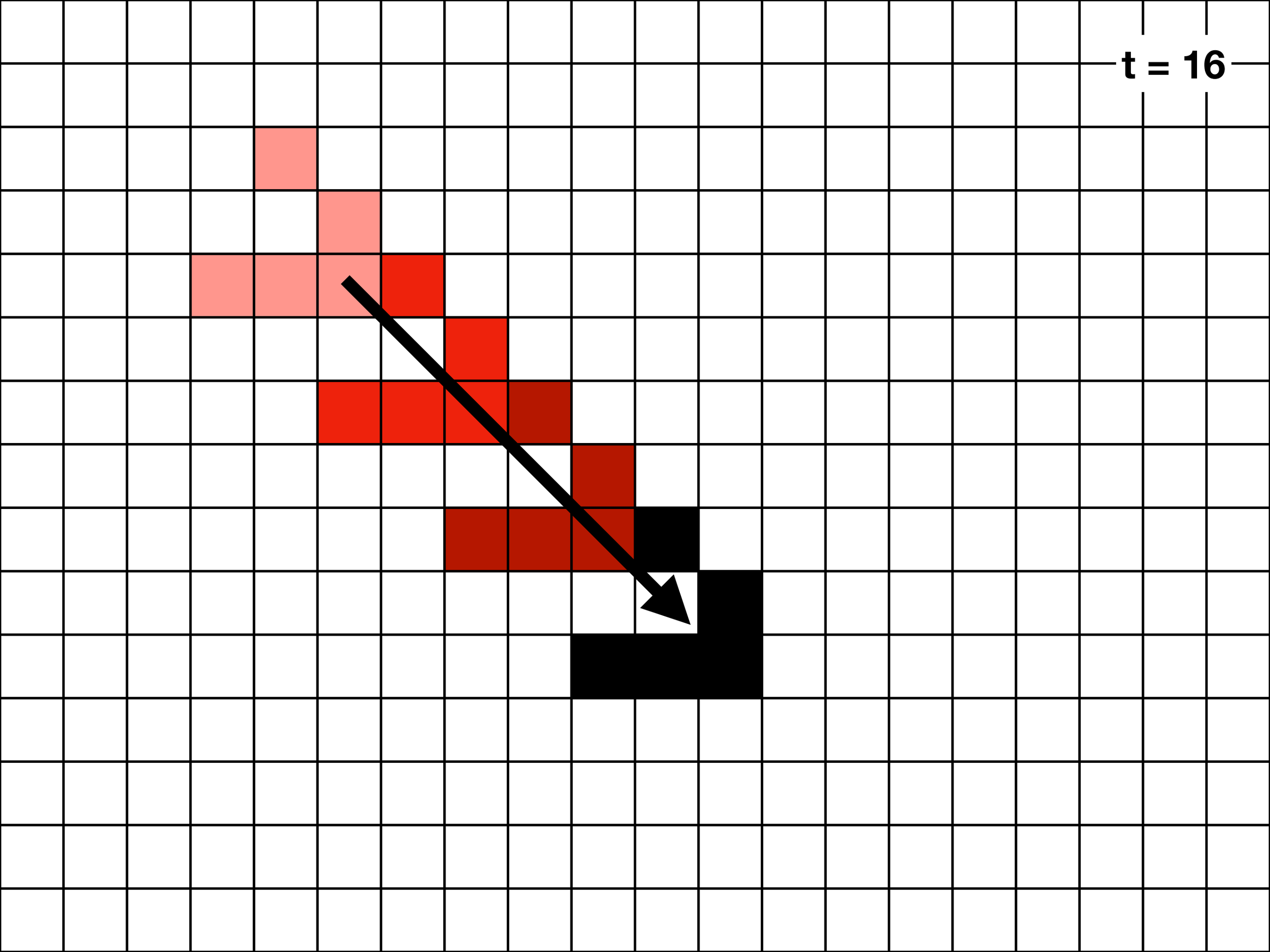
t = 8



t = 12



t = 16



Impossibilité du vote par majorité

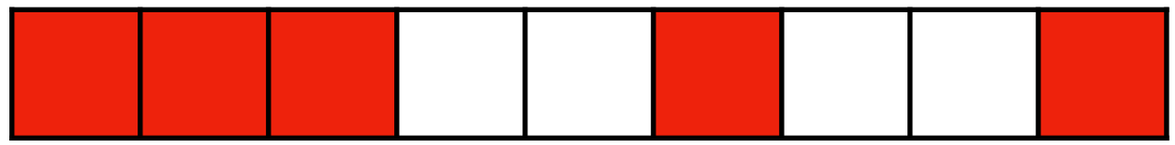
Un problème insoluble pour les automates cellulaires

- Il n'existe pas de règle pour automates cellulaires finis sur 2 états ■ et □…

- …avec conditions aux bords périodiques (la cellule à gauche de la première est la dernière et vice-versa)…

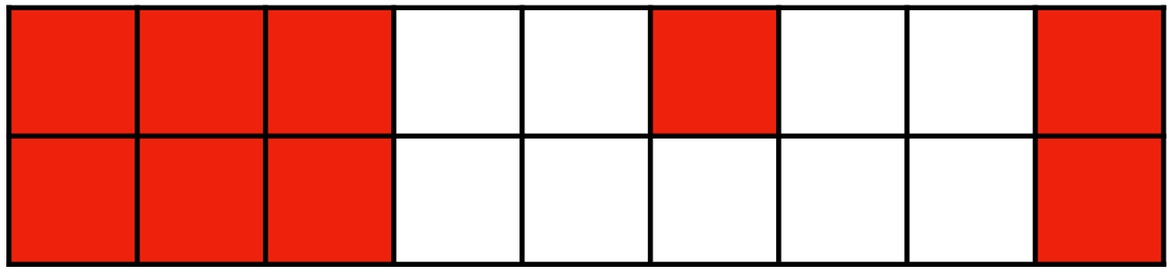
- …telle que pour chaque configuration initiale, l'automate converge sur une configuration uniforme de la couleur de la majorité des cellules de la configuration initiale

La règle de majorité locale ne marche pas !



le rouge devrait gagner, mais…

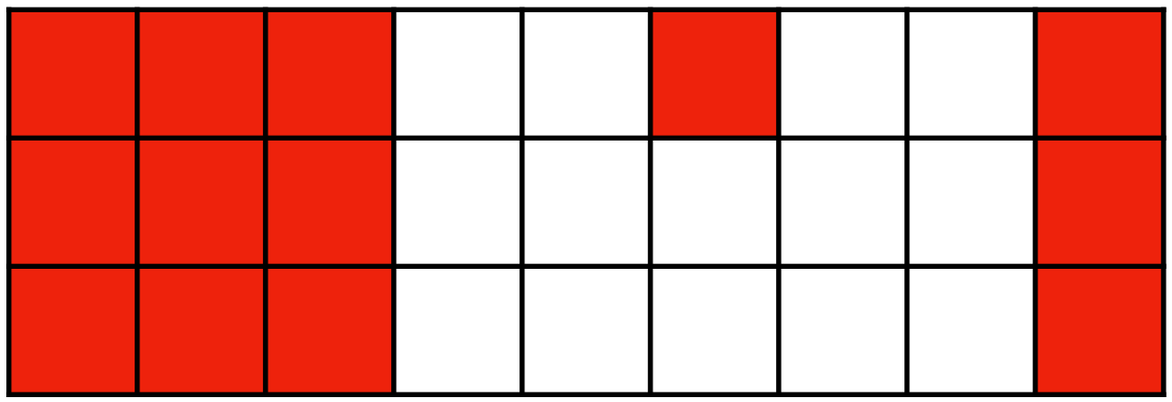
La règle de majorité locale ne marche pas !



le blanc
maintenant
à la majorité !

temps

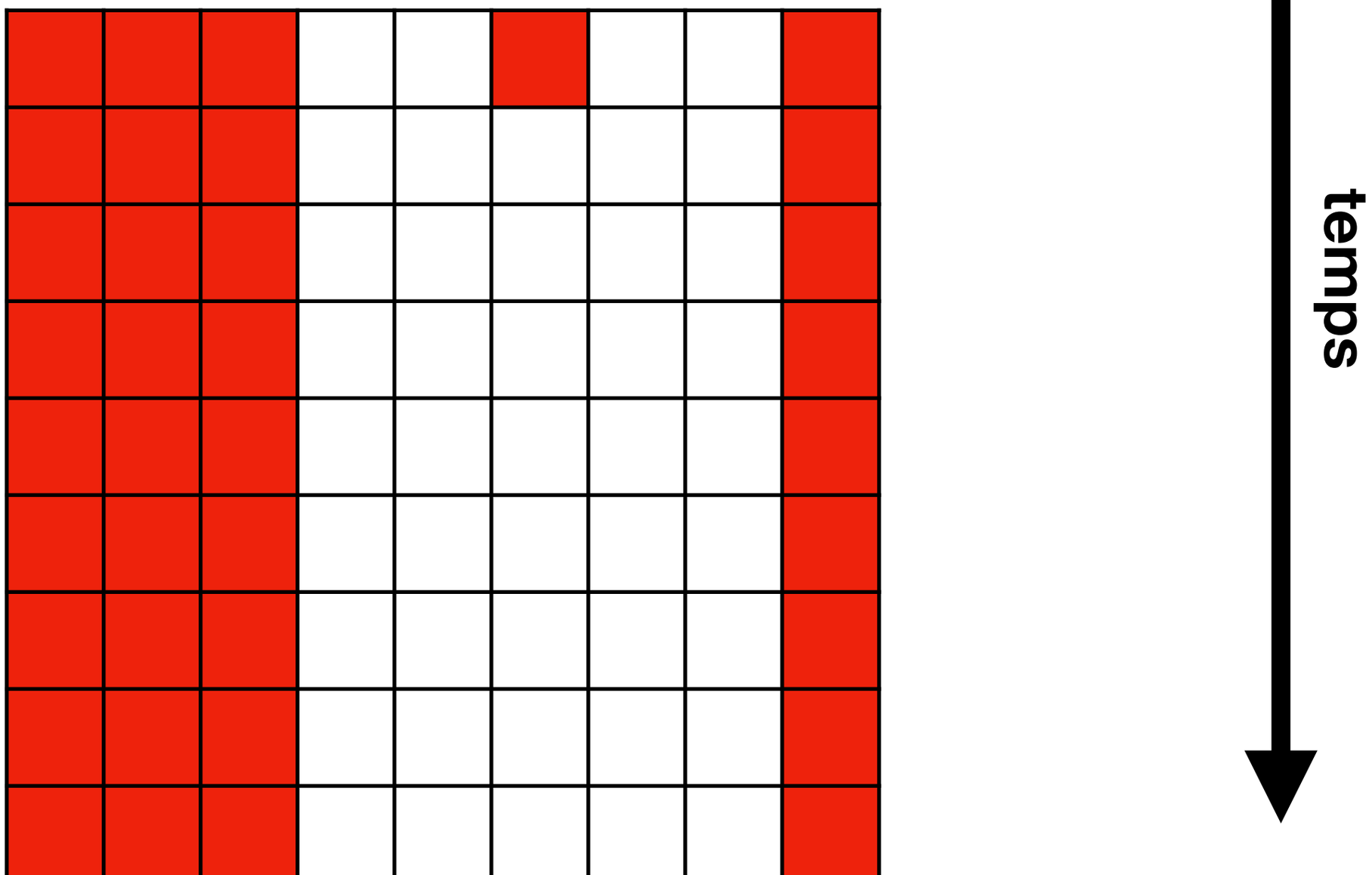
La règle de majorité locale ne marche pas !



et en plus
ça se répète…

temps

La règle de majorité locale ne marche pas !



Machines autoréplicatives

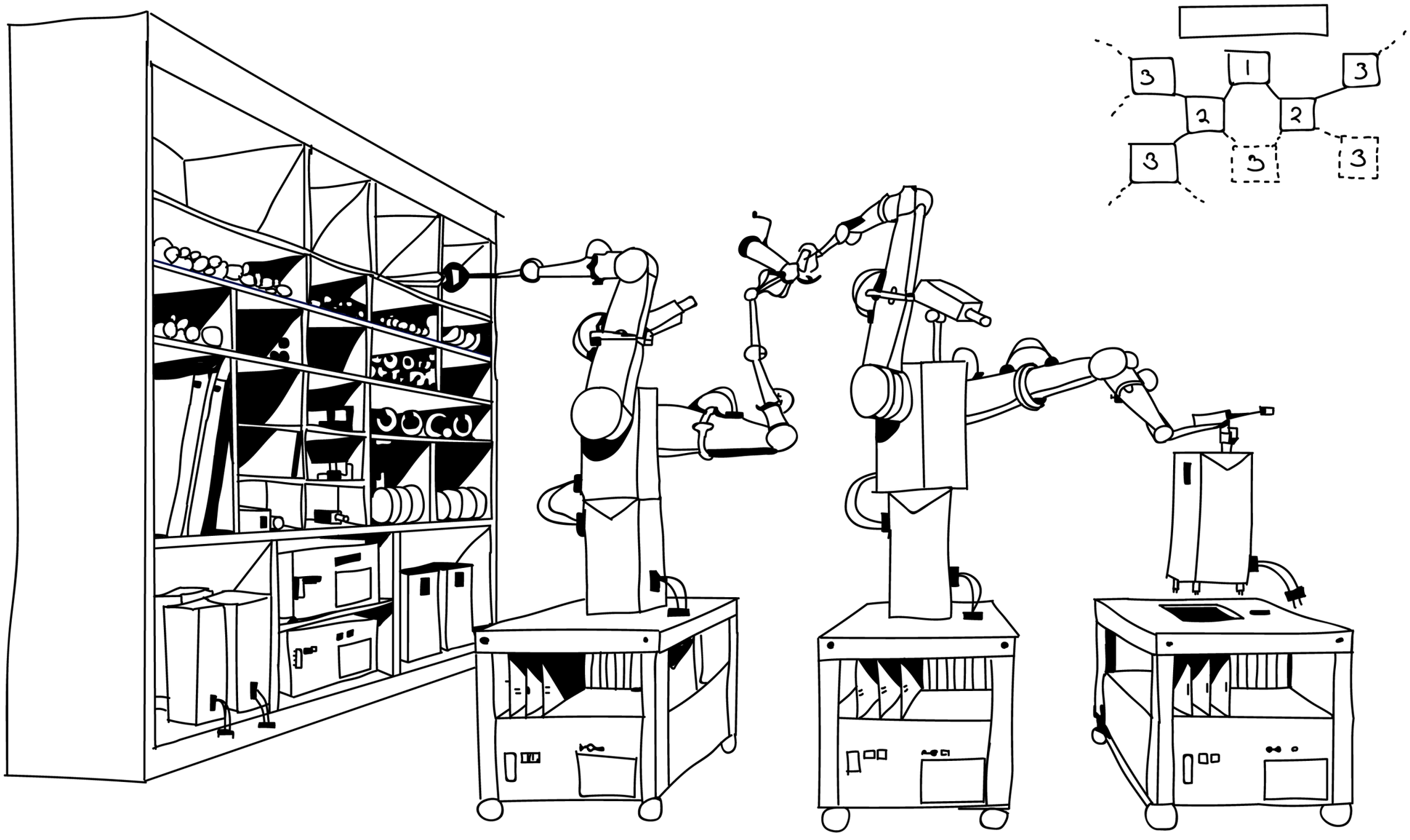
Machines autoréplicatives et automates cellulaires

- Existe-t-il une machine capable de construire une copie identique d'elle-même ?

- D'ailleurs ça arrive tout le temps en biologie… 🦠

- John von Neumann a étudié le problème (années 40) dans le domaine des automates cellulaires

- La réponse est oui !

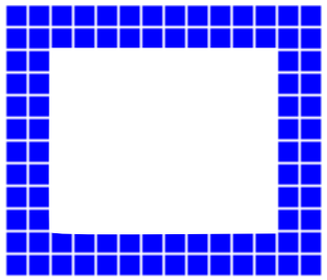

3
1
3
2
2
3
3
3

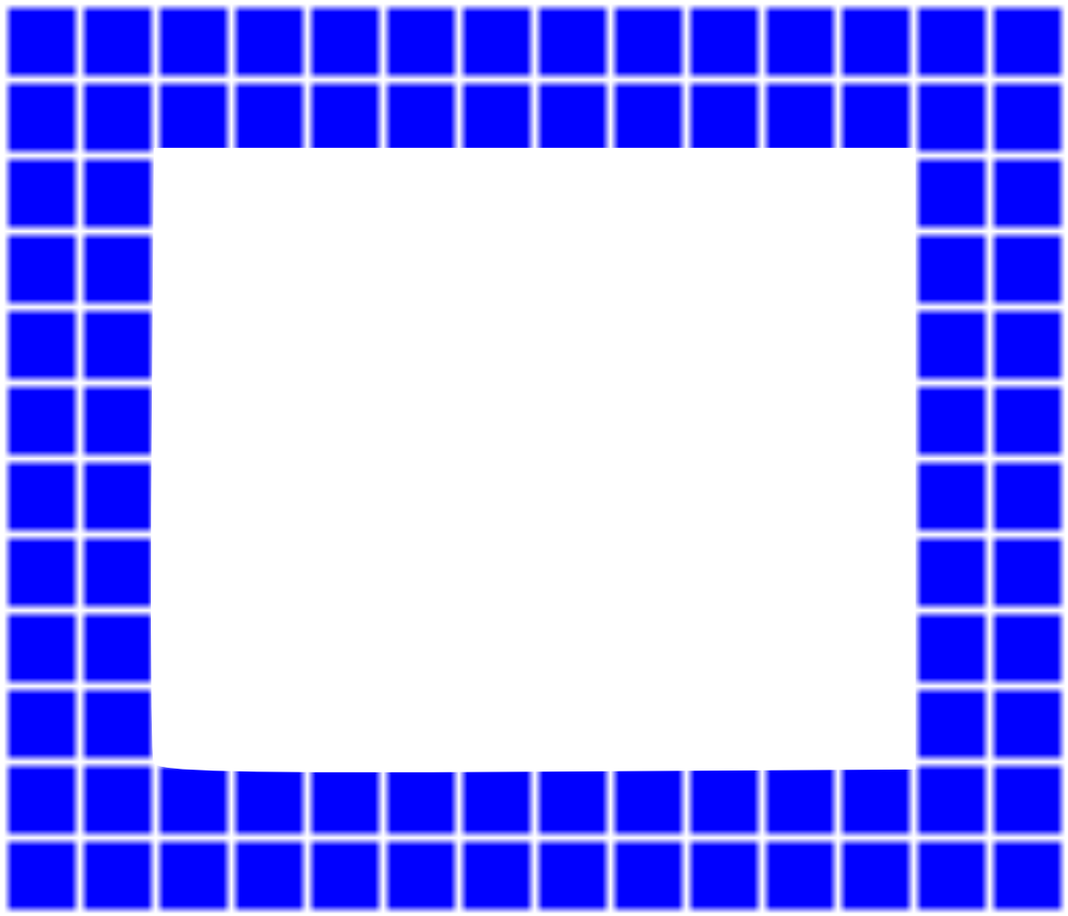
La règle de parité (XOR) pour les automates cellulaires 2D

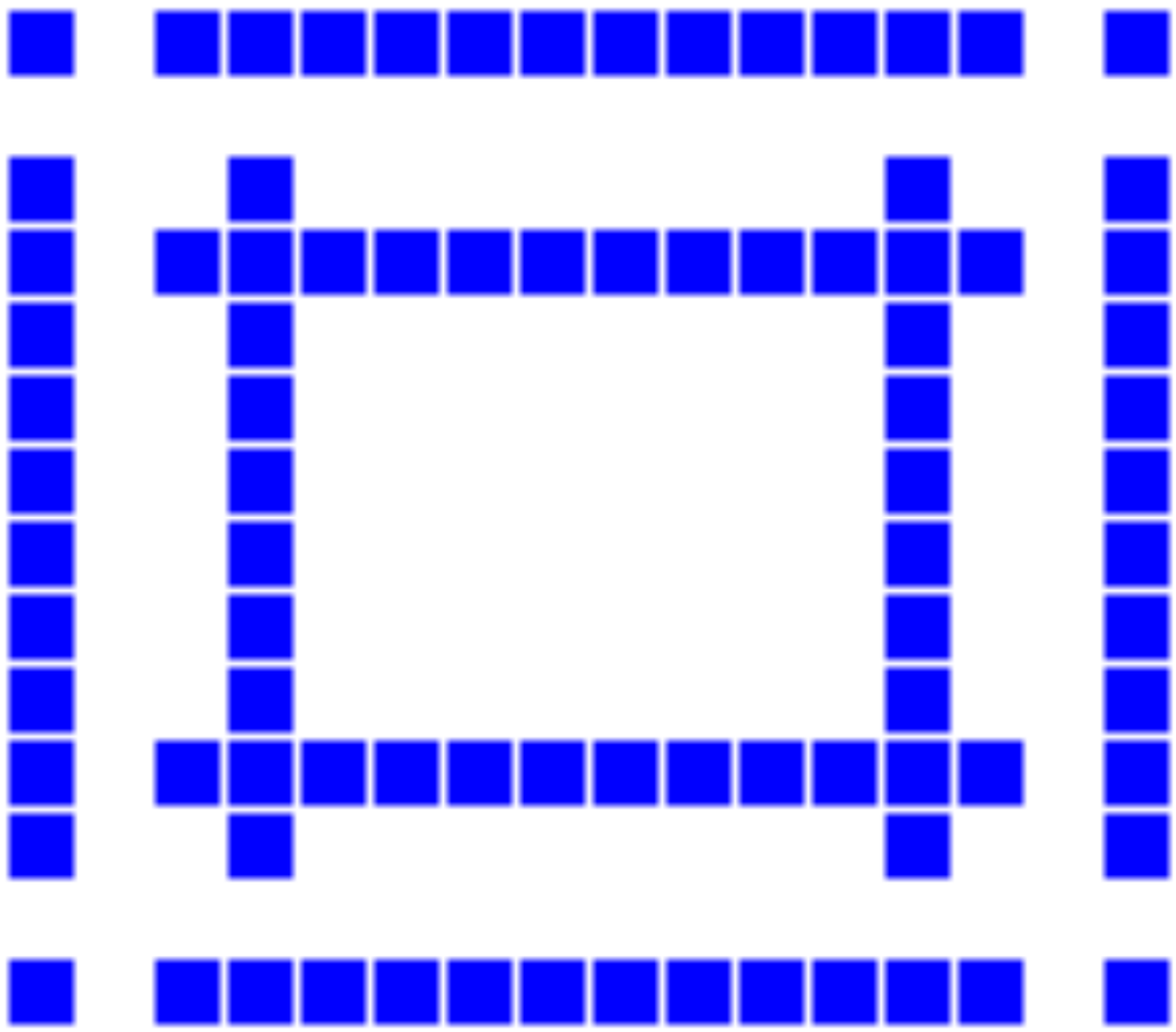
- L’état de chaque cellule à l’étape suivante est ■ si le nombre de ■ dans son voisinage (y compris elle-même) est impaire

- Sinon son état suivant est □

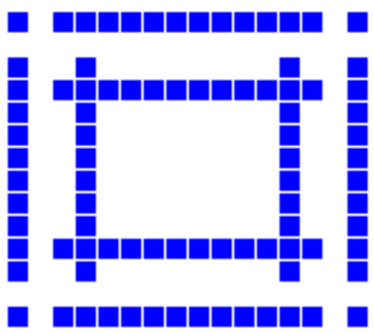
Une machine autoréplicative pour la règle de parité



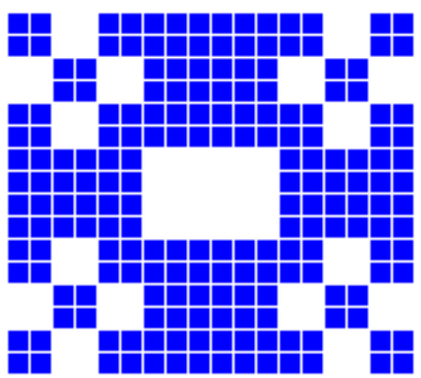




Une machine autoréplicative pour la règle de parité

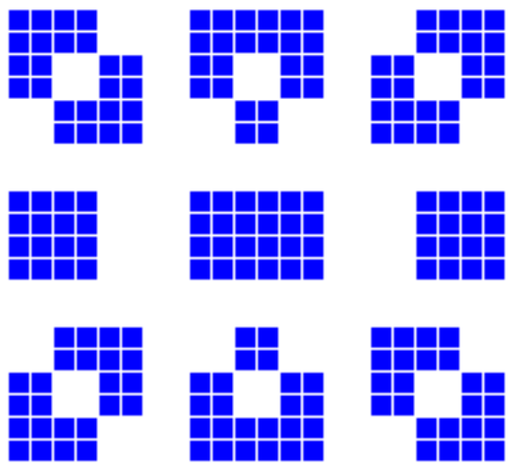


Une machine autoréplicative pour la règle de parité

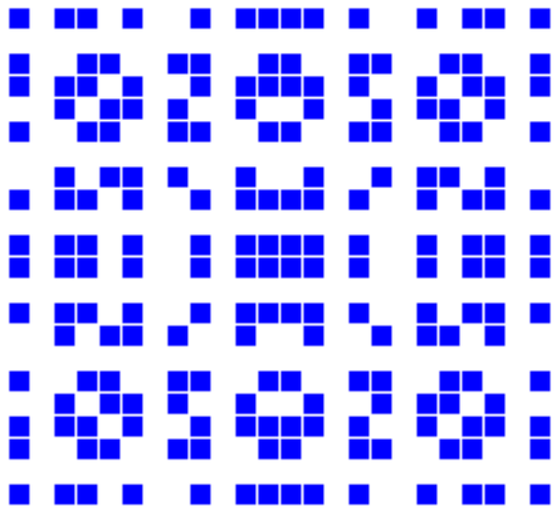


Une machine autoréplicative pour la règle de parité

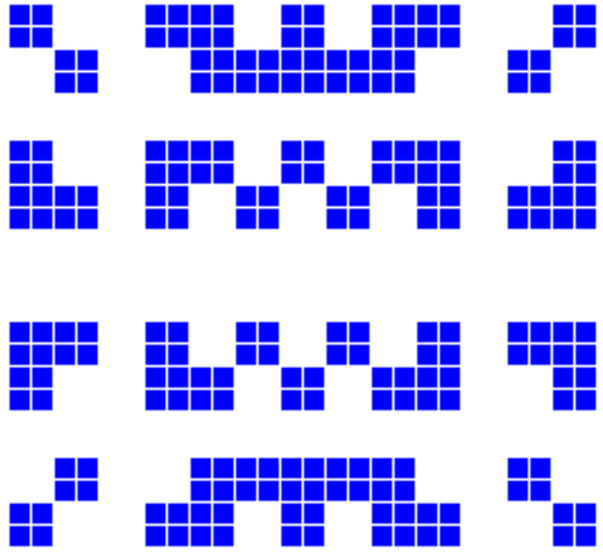
Une machine autoréplicative pour la règle de parité



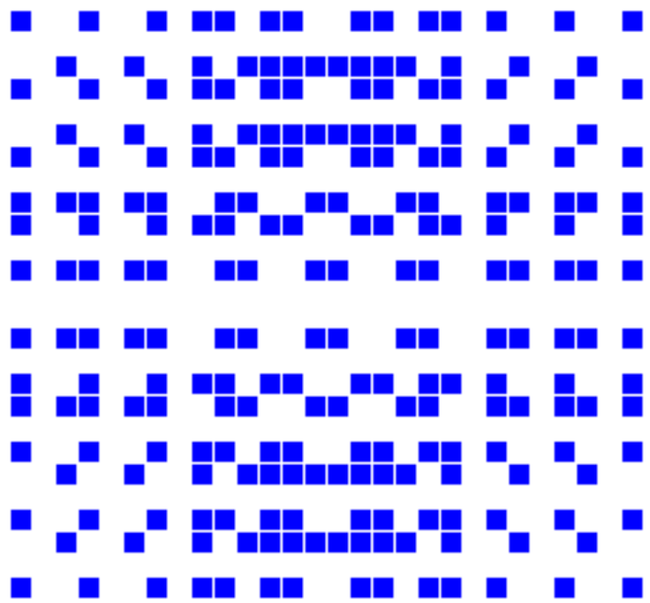
Une machine autoréplicative pour la règle de parité



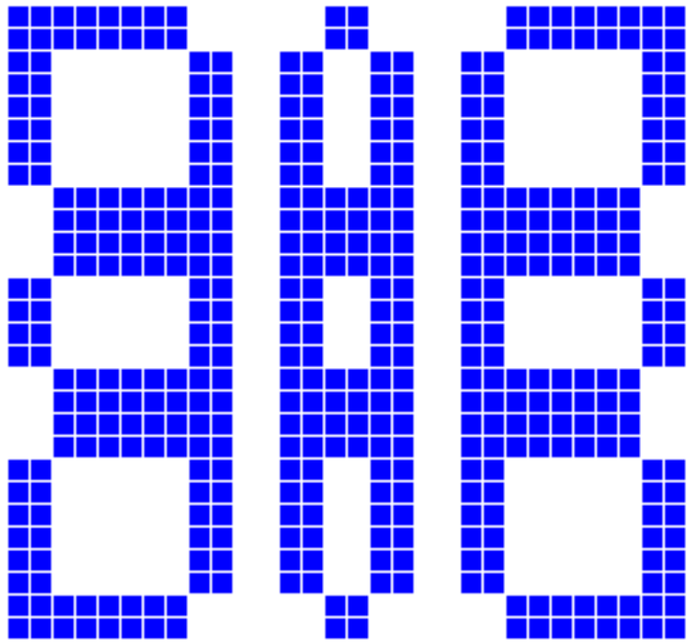
Une machine autoréplicative pour la règle de parité



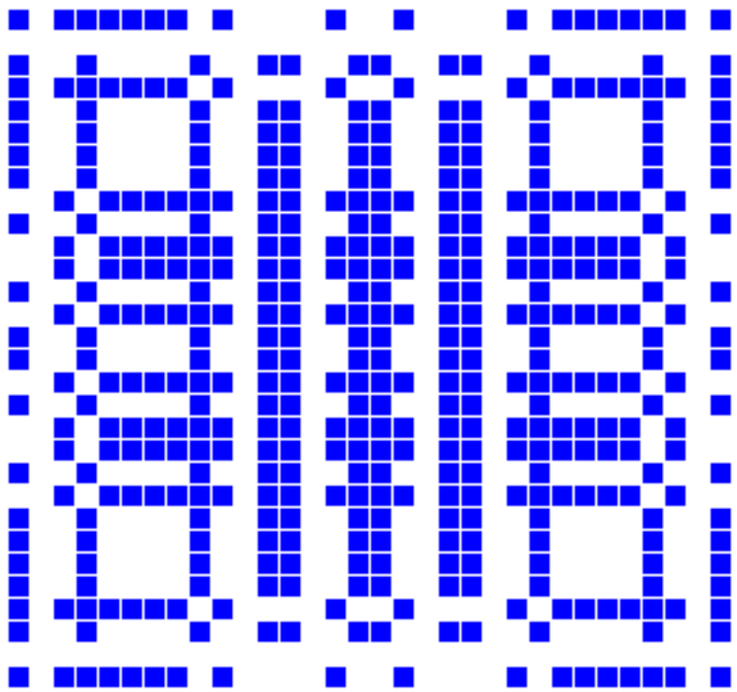
Une machine autoréplicative pour la règle de parité



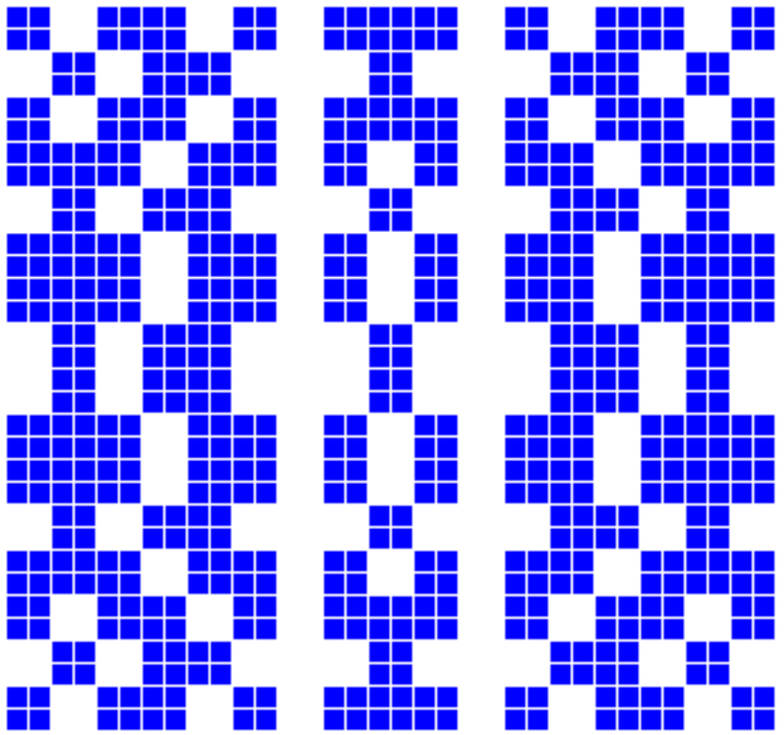
Une machine autoréplicative pour la règle de parité



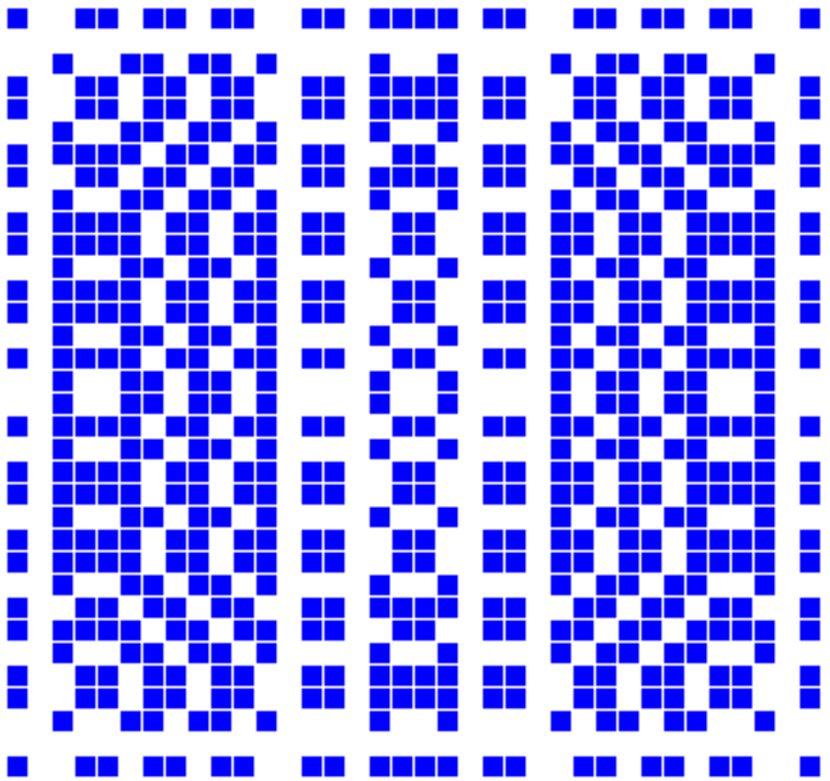
Une machine autoréplicative pour la règle de parité



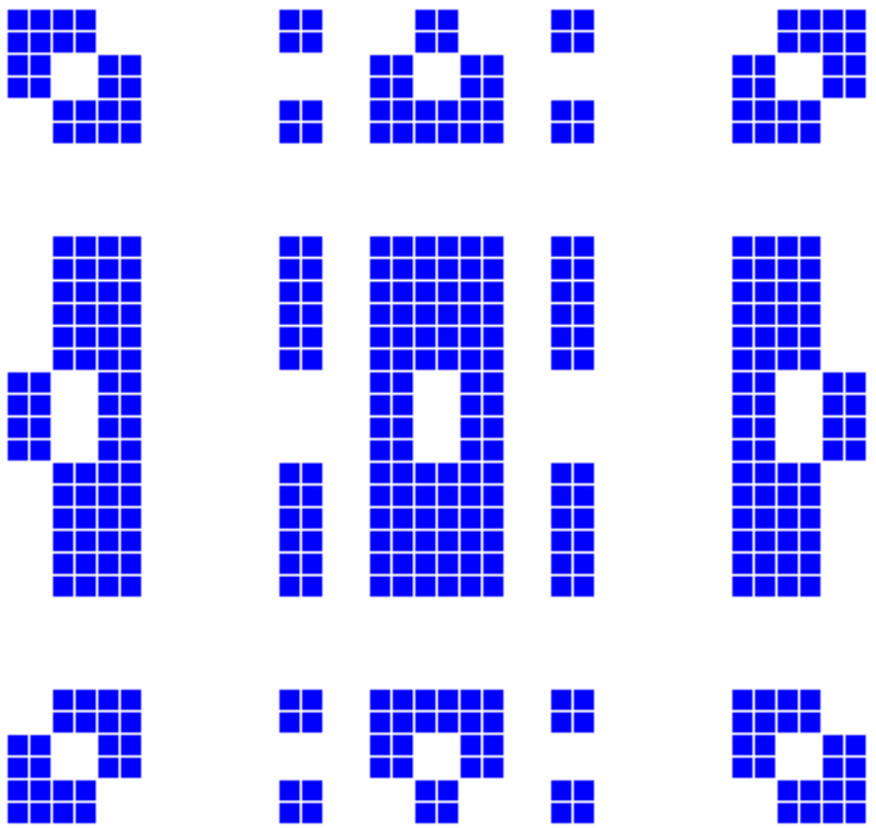
Une machine autoréplicative pour la règle de parité



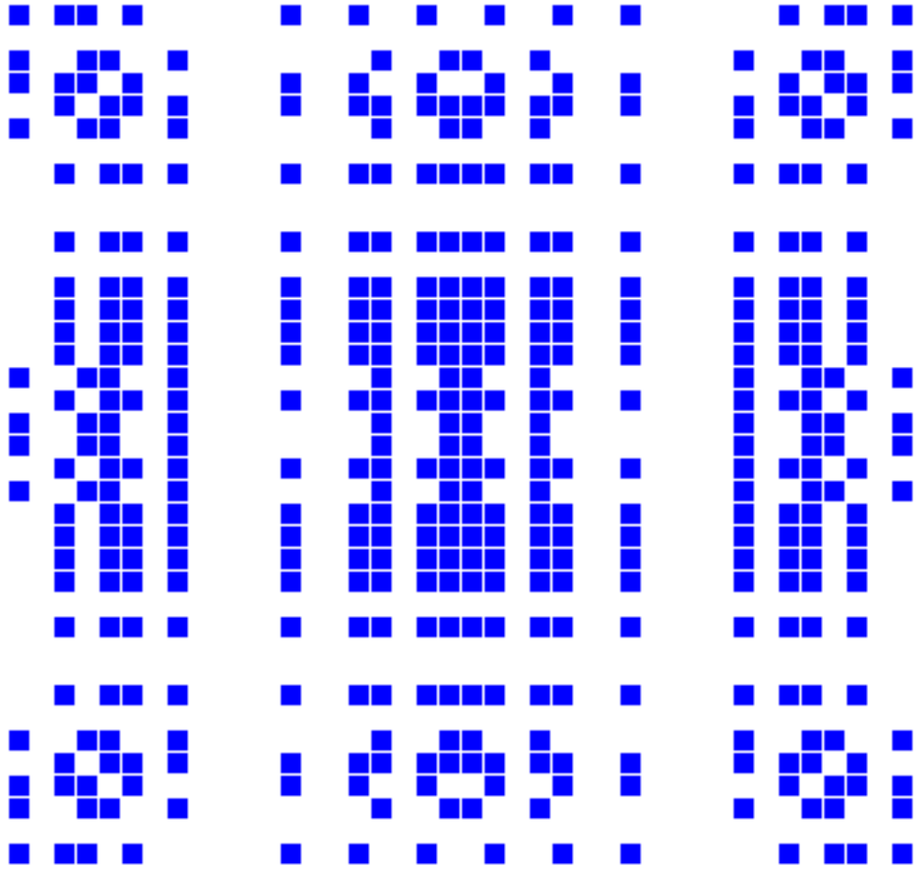
Une machine autoréplicative pour la règle de parité



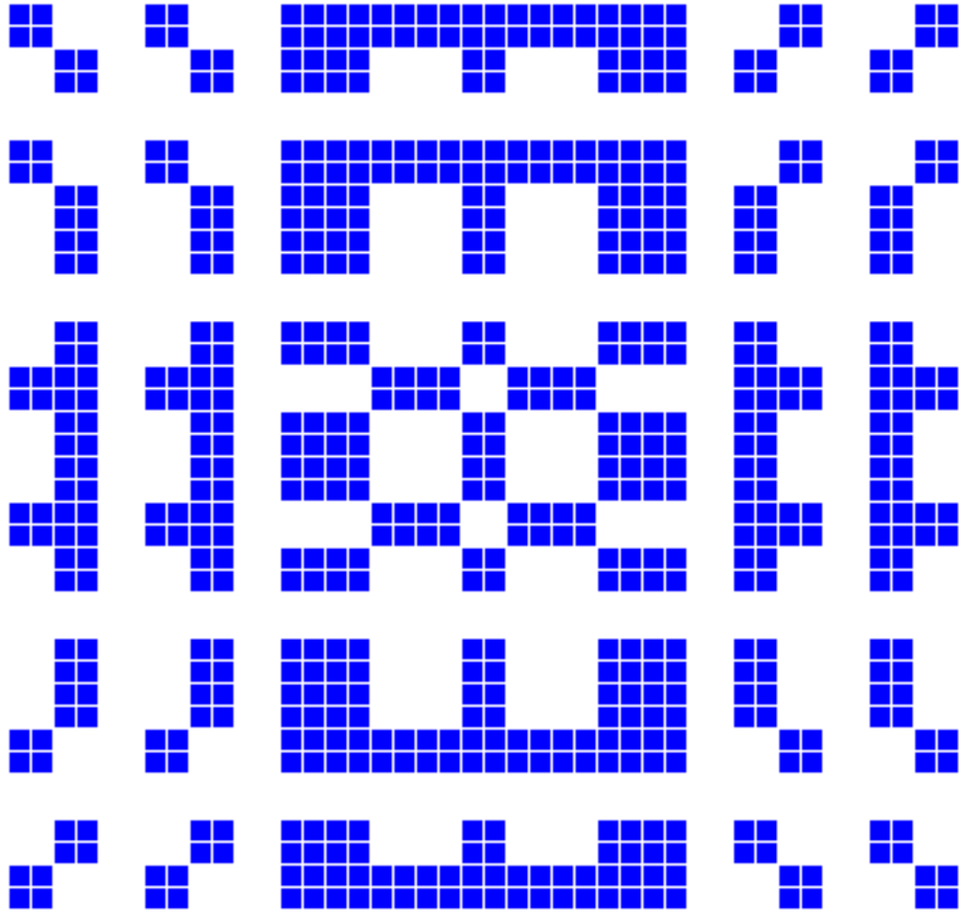
Une machine autoréplicative pour la règle de parité



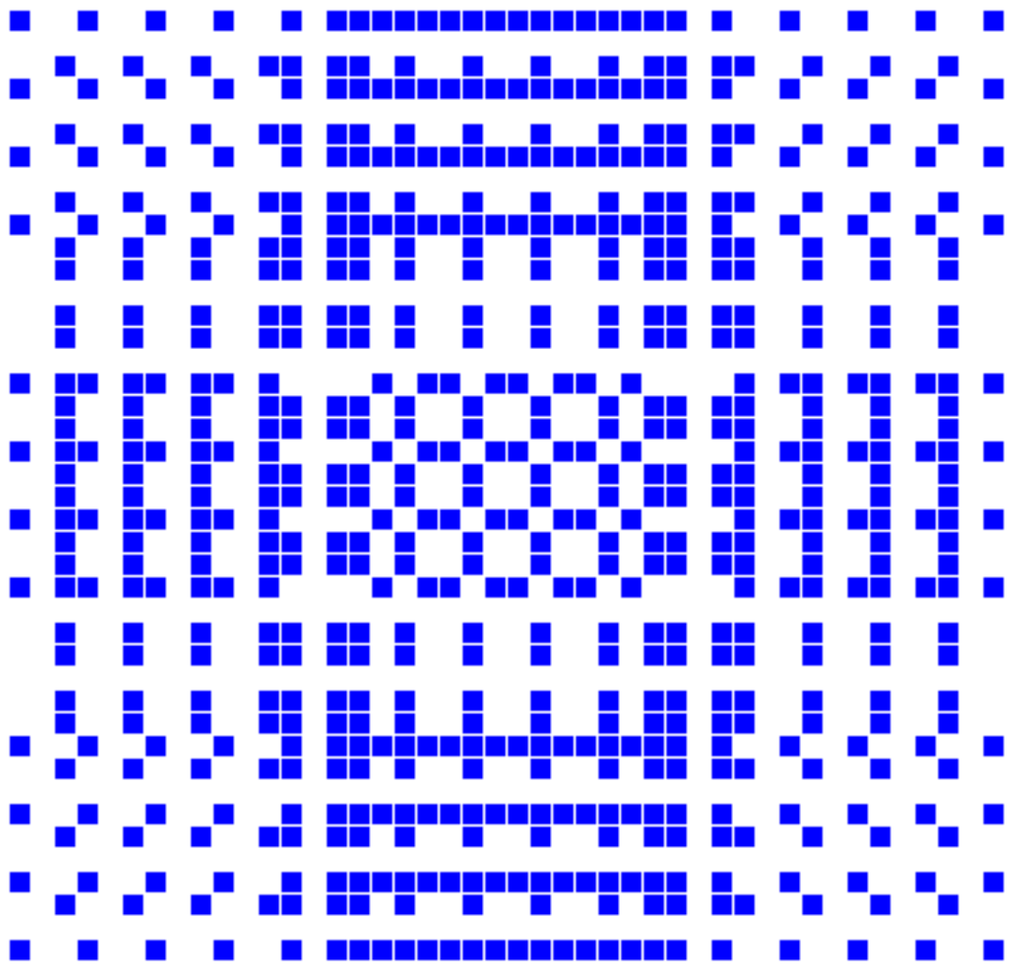
Une machine autoréplicative pour la règle de parité



Une machine autoréplicative pour la règle de parité



Une machine autoréplicative pour la règle de parité



Une machine autoréplicative pour la règle de parité