Registres

Registres

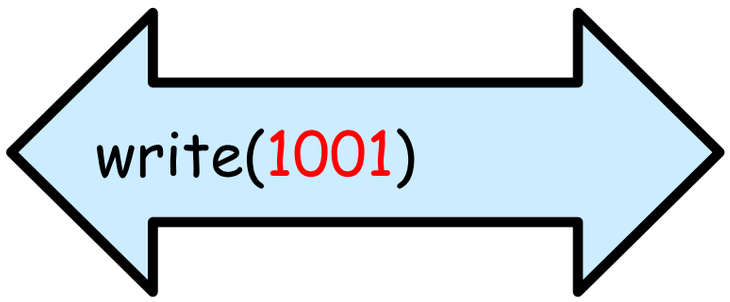
```
public interface Register<T> {
   public T read();
   public void write(T v);
}
```

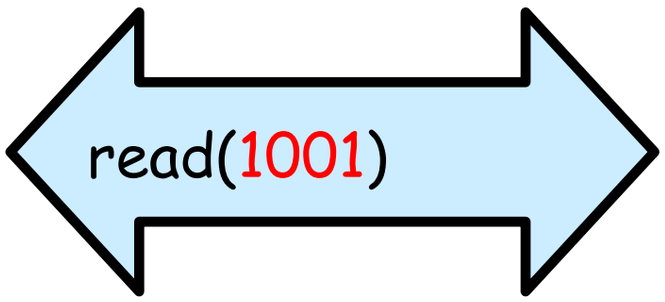
Registres

- SRSW
  - Single-reader single-writer
- MRSW
  - Multi-reader single-writer
- MRMW
  - Multi-reader multi-writer

Registre sûr (safe)

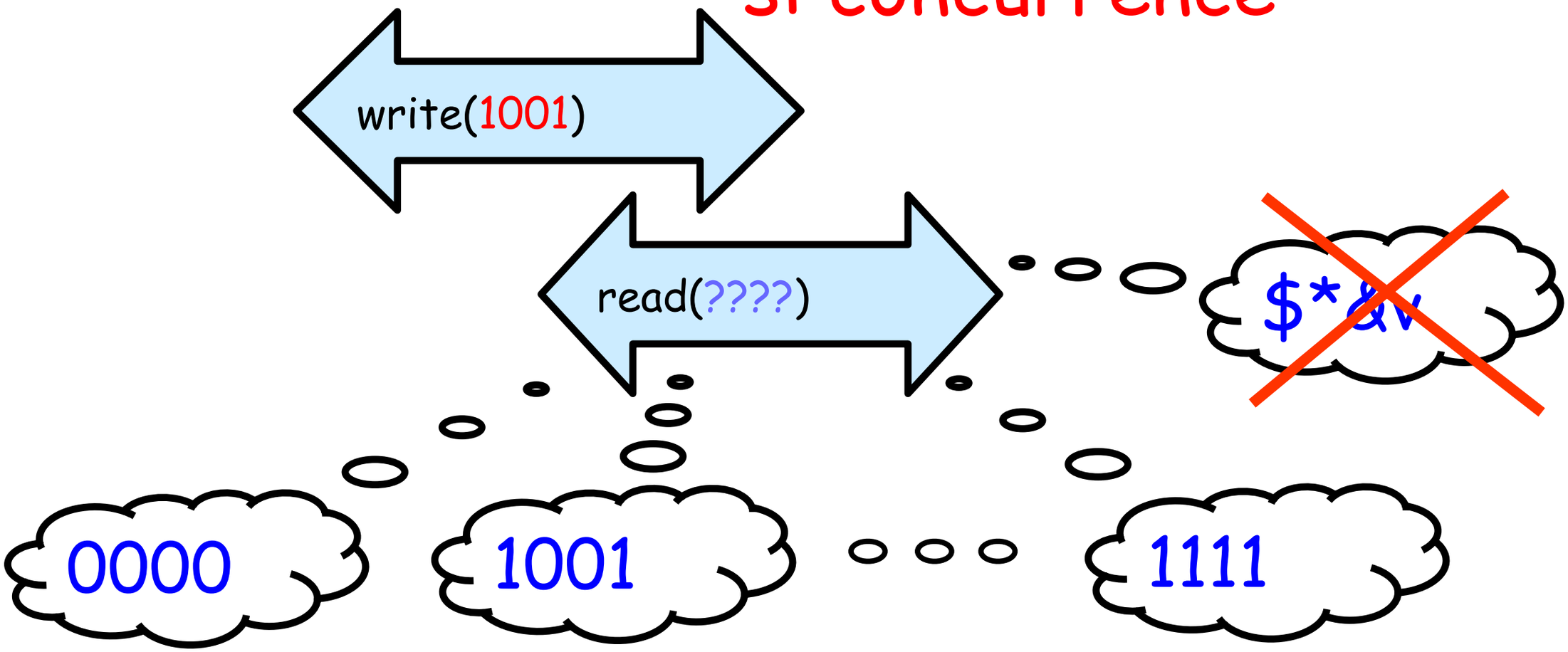
OK si non concurrence



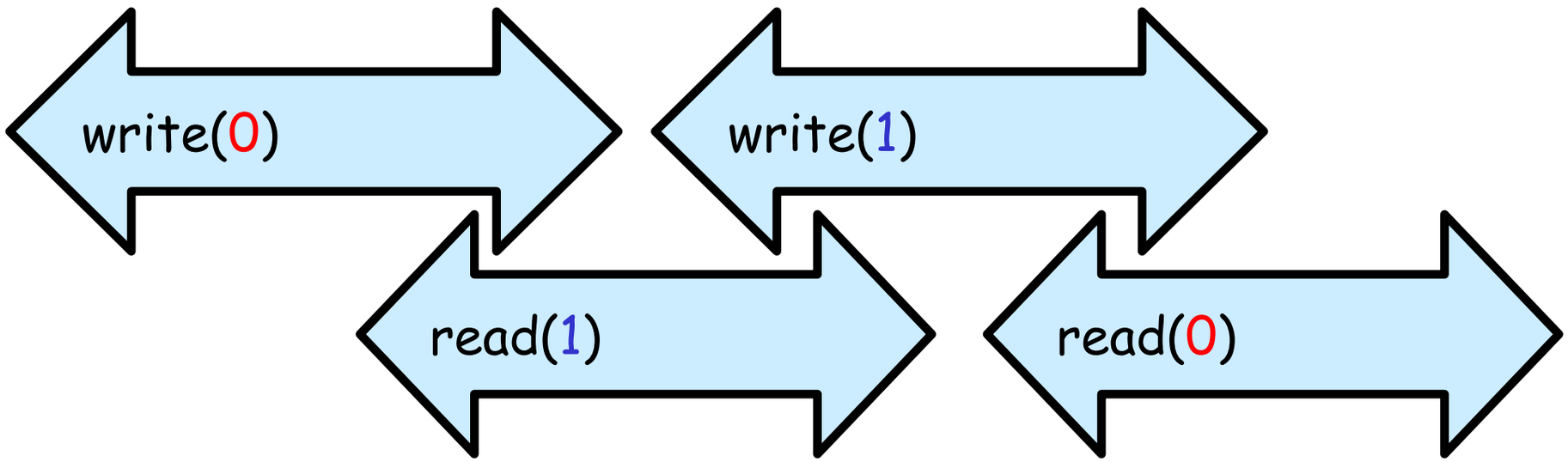


Registre Sûr

Une valeur « légale »
si concurrence

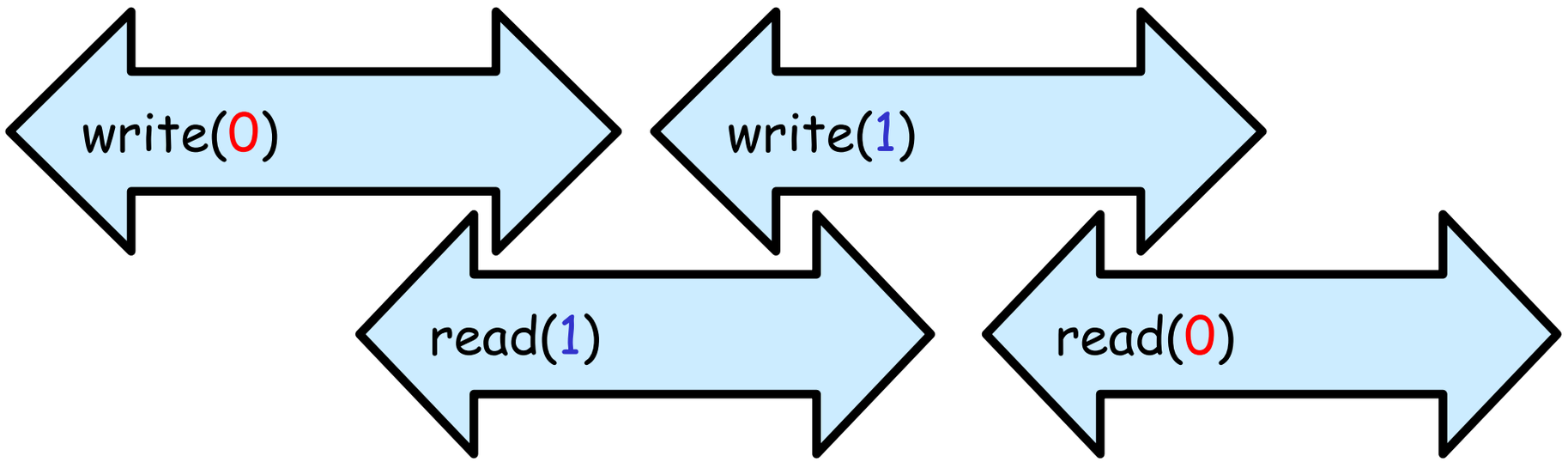


Registre Régulier



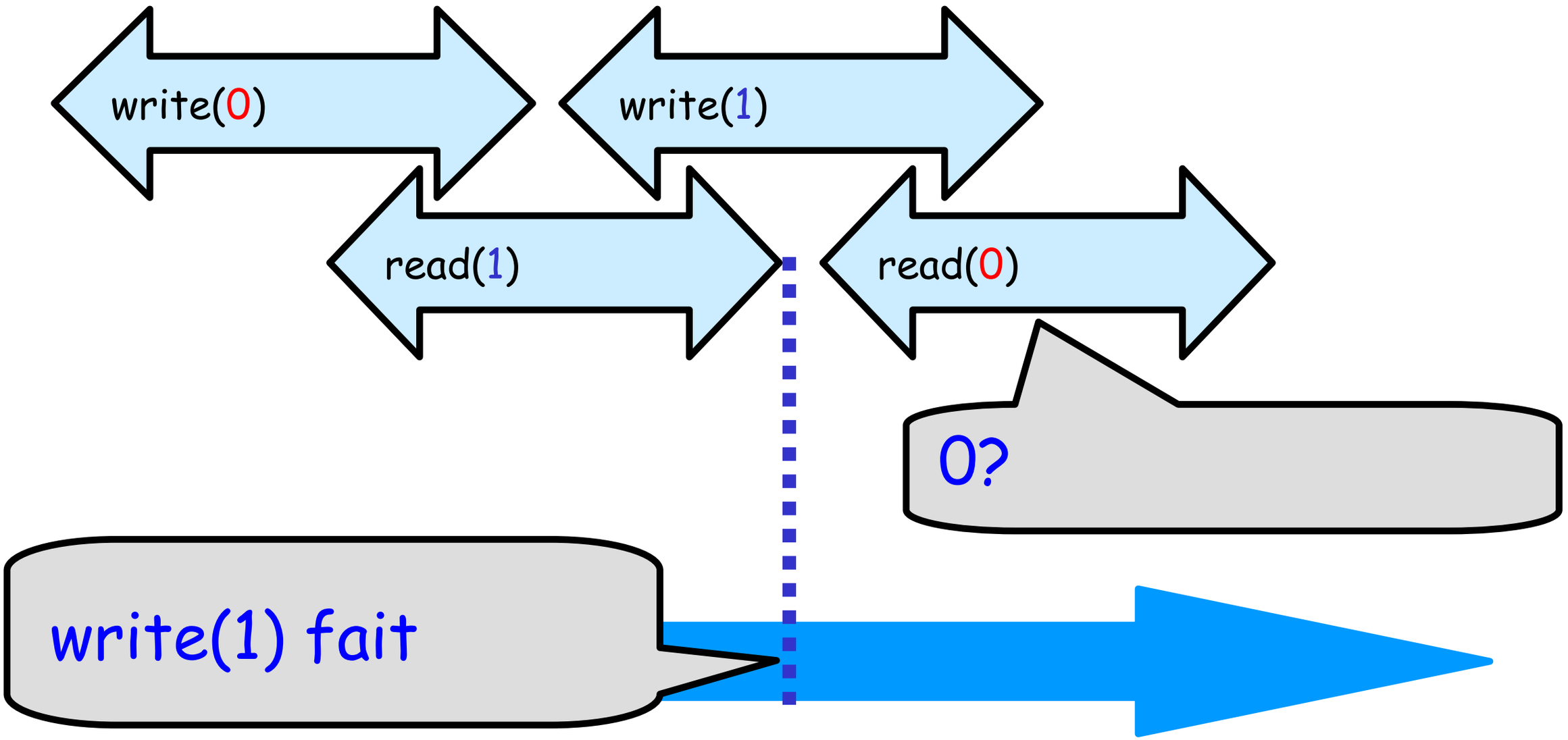
- Single Writer
- Readers:
  - ancienne valeur si non concurrence (sûr)
  - ancienne ou nouvelle si concurrence

Régulier?



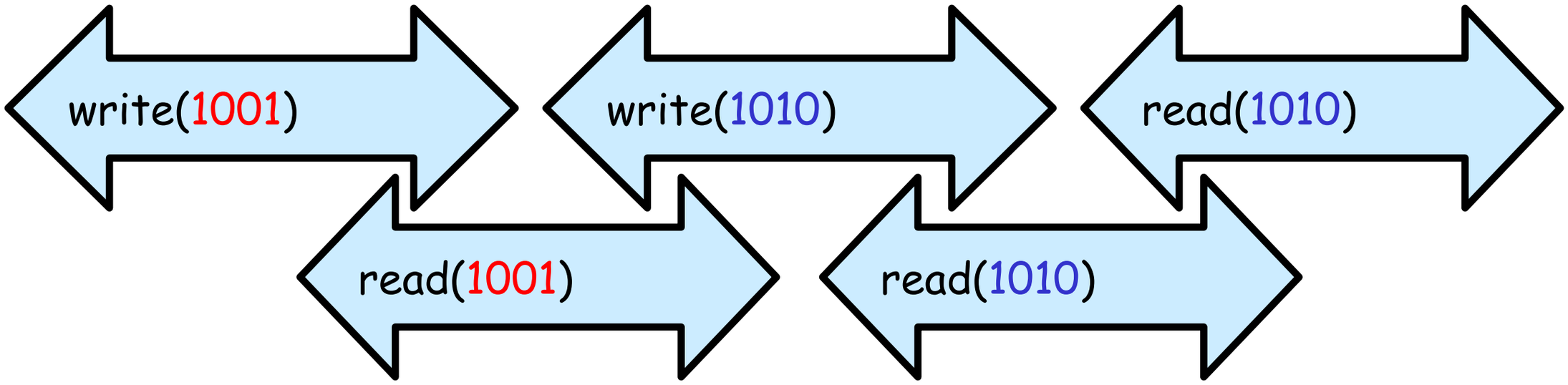
registres

Régulier ≠ Linéarisable



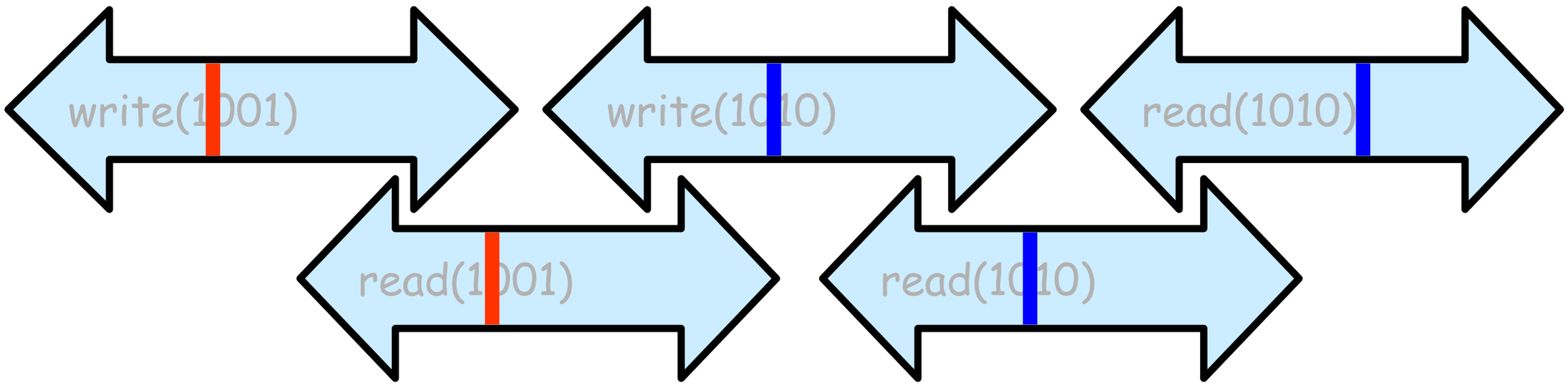
registres

Registre atomique



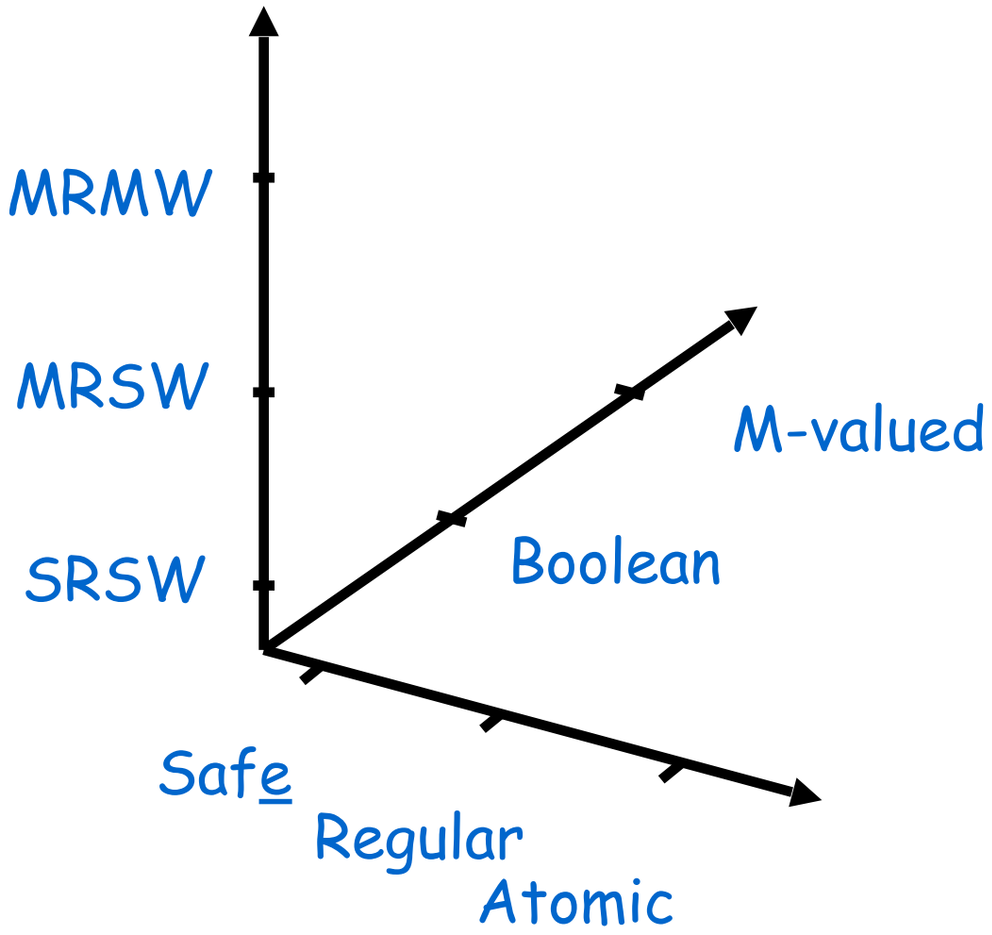
Linéarisable

Atomic Register



registres

Registres



Implementation Wait-Free

Definition: L'implementation d'un objet est **wait-free** si tout appel de méthode de l'objet termine (en un nombre fini de pas)

Pas d'exclusion mutuelle!

- Un Thread peut s'arrêter en section critique

Le programme…

- SRSW Booléen sûr
- MRSW Booléen sûr
- MRSW Booléen régulier
- MRSW regulier
- MRSW atomique
- MRMW atomique

Le programme…

- SRSW Booléen sûr
- MRSW Booléen sûr
- MRSW Booléen régulier
- MRSW regulier
- MRSW atomique
- MRMW atomique

Registres

```
public class SafeBoolMRSWRegister
 implements Register<Boolean> {
  public boolean read() { … }
  public void write(boolean x) { … }
}
```

Registres

```
public class SafeBoolMRSWRegister
 implements Register<Boolean> {
  public boolean read() { … }
  public void write(boolean x) { … }
}
```

propriété

type

nombre de writers
nombre de readers

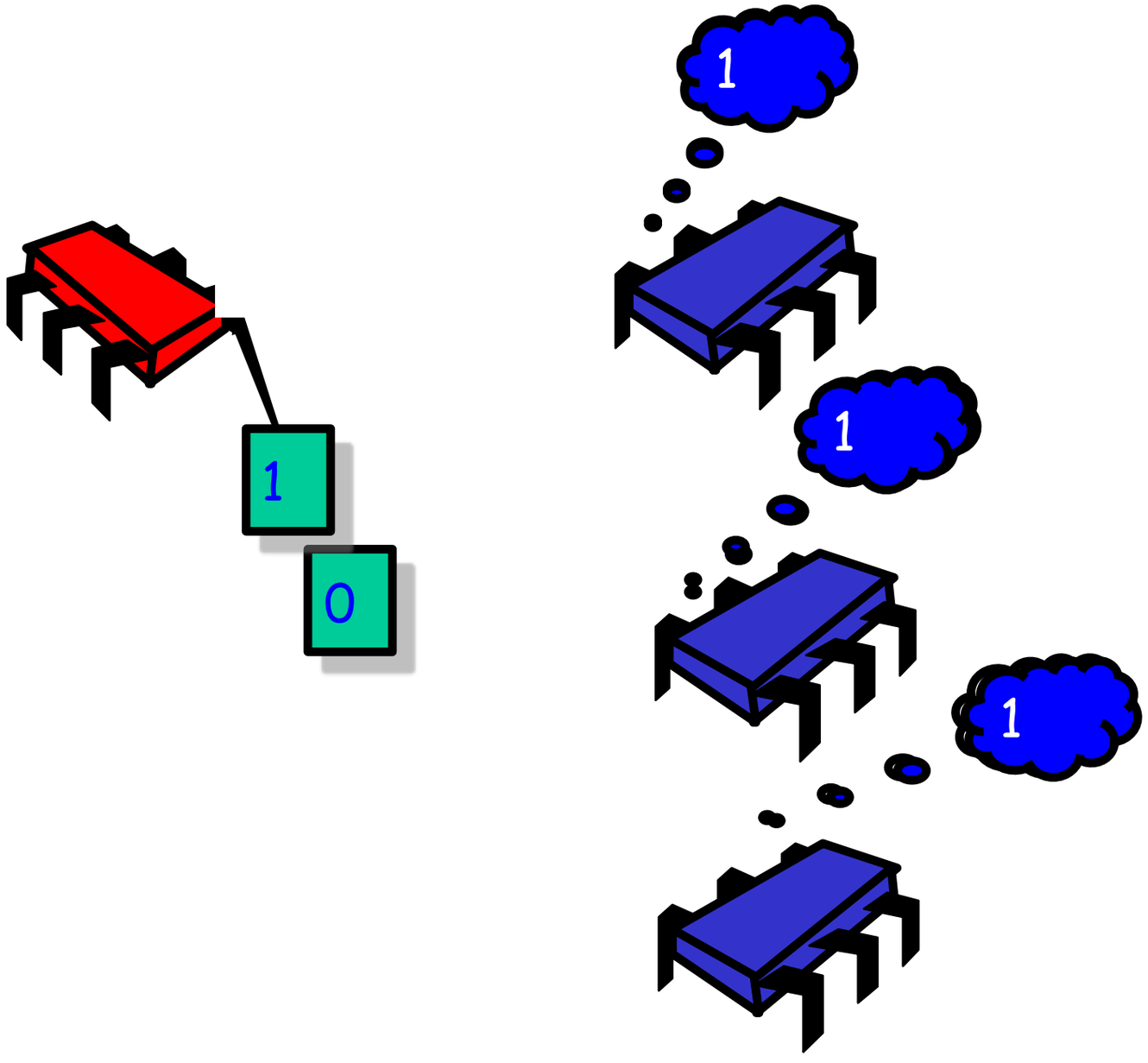
MRSW booléens sûrs à partir de SRSW booléens sûrs

```
public class SafeBoolMRSWRegister
 implements Register<Boolean> {
 private SafeBoolSRSWRegister[] r =
   new SafeBoolSRSWRegister[N];
  public void write(boolean x) {
   for (int j = 0; j < N; j++)
     r[j].write(x);
}
public boolean read() {
 int i = ThreadID.get();
 return r[i].read();
}}
```

Le programme…

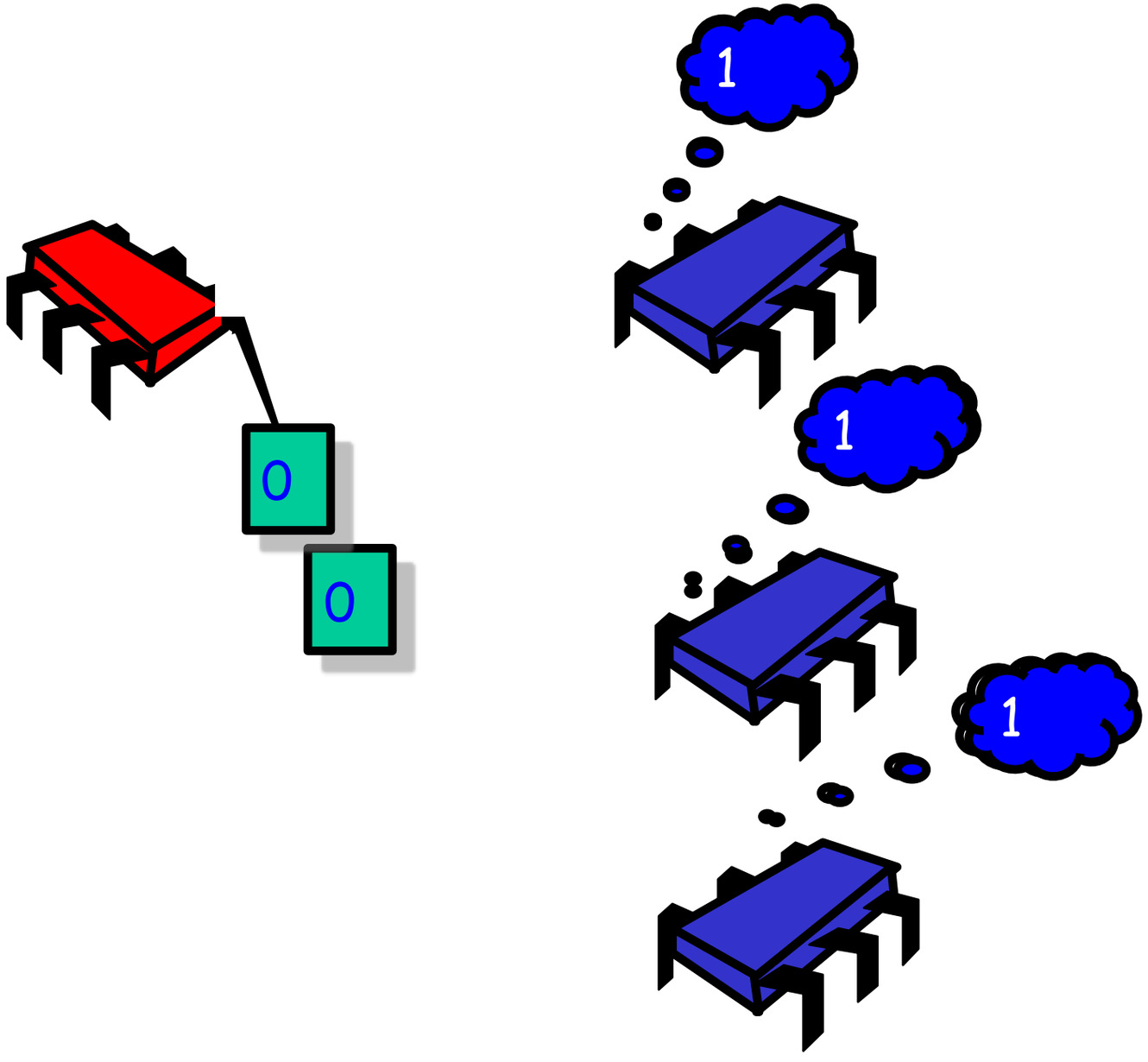
- SRSW Booléen sûr
- MRSW Booléen sûr
- MRSW Booléen régulier
- MRSW regulier
- MRSW atomique
- MRMW atomique

MRSW Booléens Réguliers à partir de MRSW Booléens Sûrs



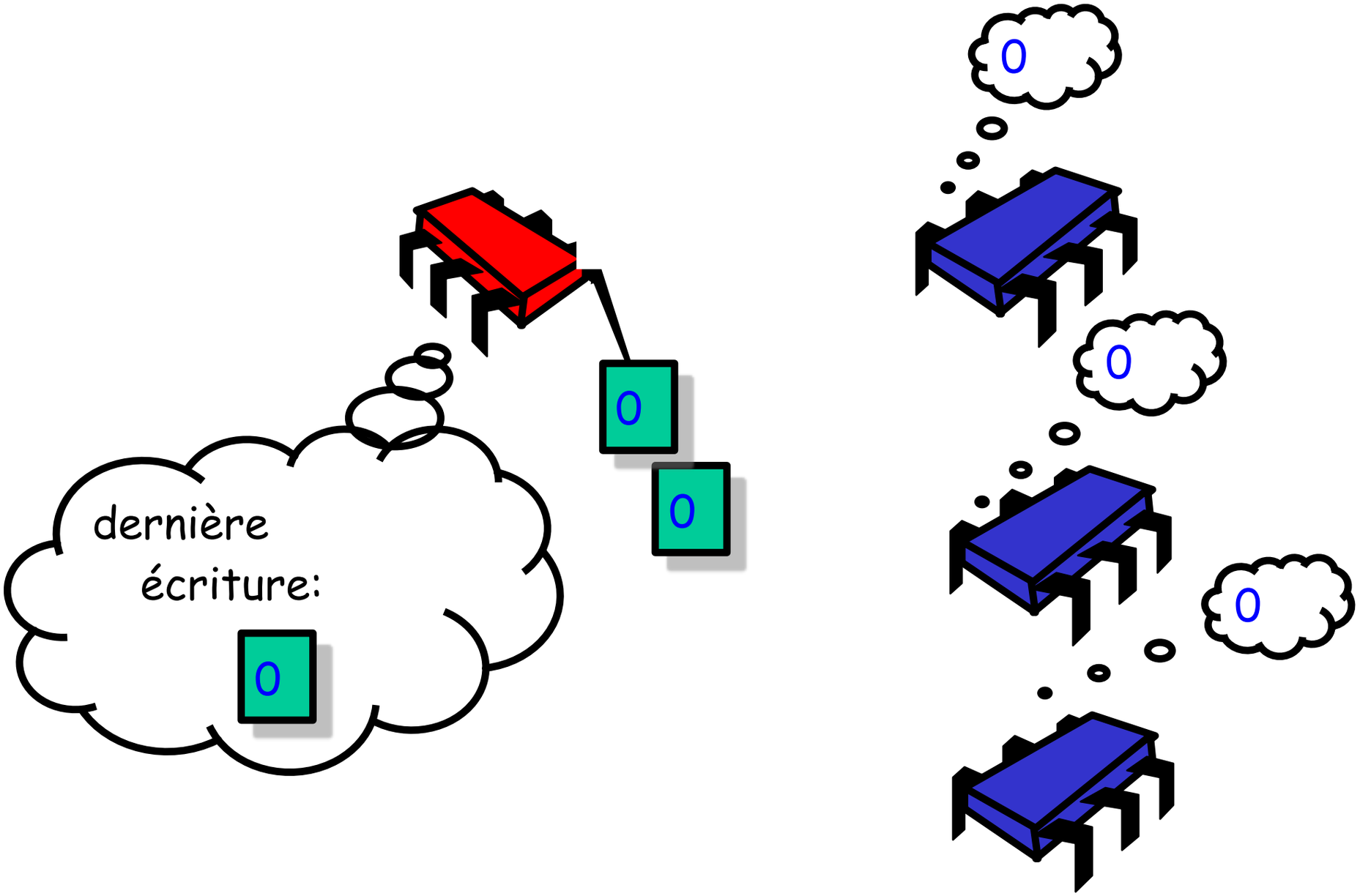
registres

MRSW Booléens Réguliers à partir de MRSW Booléens Sûrs



registres

MRSW Booléens Réguliers à partir de MRSW Booléens Sûrs

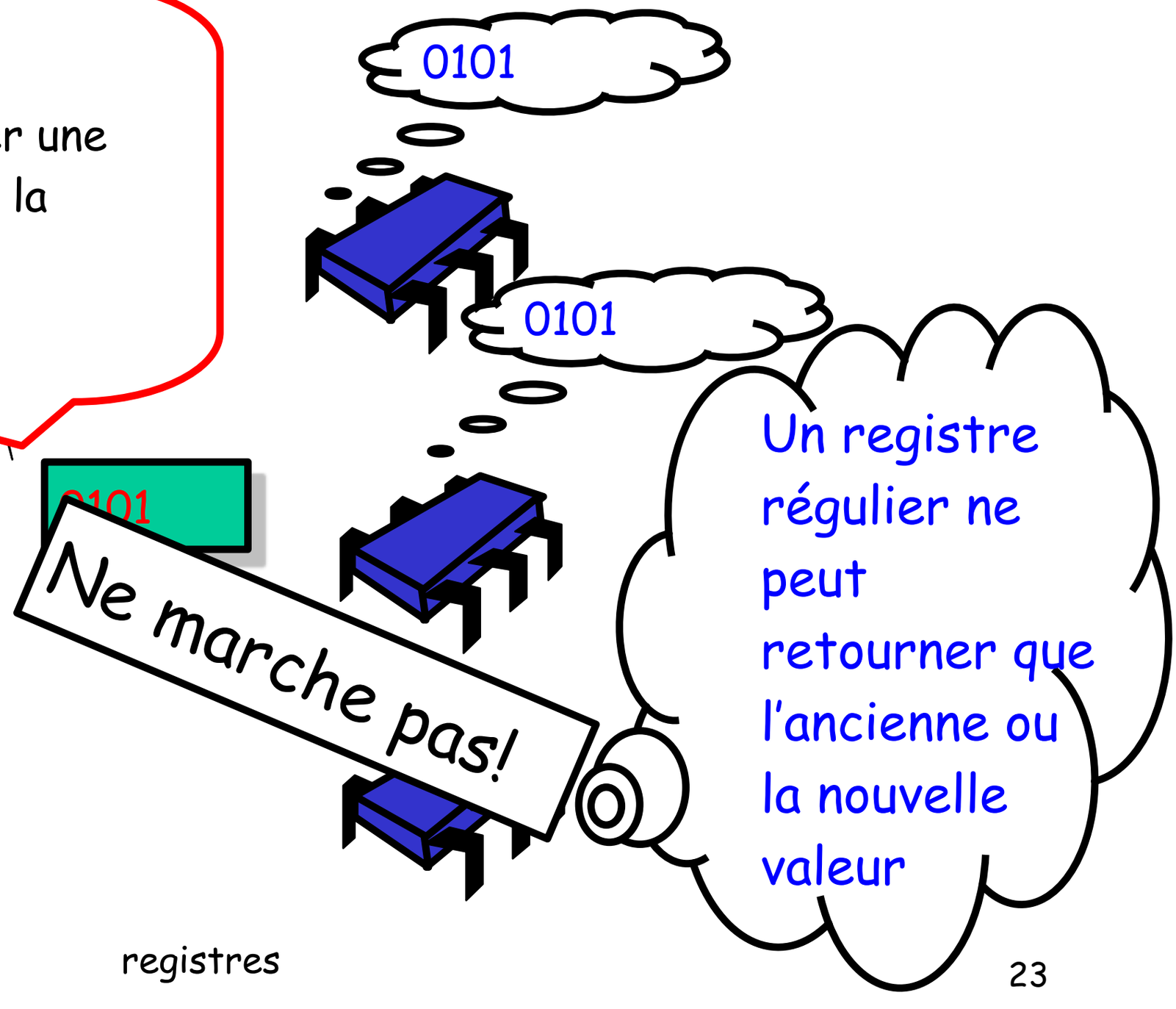


MRSW booléens réguliers à partir MRSW booléens sûrs

```
public class RegBoolMRSWRegister
 implements Register<Boolean> {
  private boolean old;
  private SafeBoolMRSWRegister value;
  public void write(boolean x) {
   if (old != x) {
    value.write(x);
    old = x;
  }}
public boolean read() {
 return value.read();
}}
```

et multi-valués?

Un registre sûr peut retourner une valeur qui n'est ni l'ancienne ni la nouvelle.



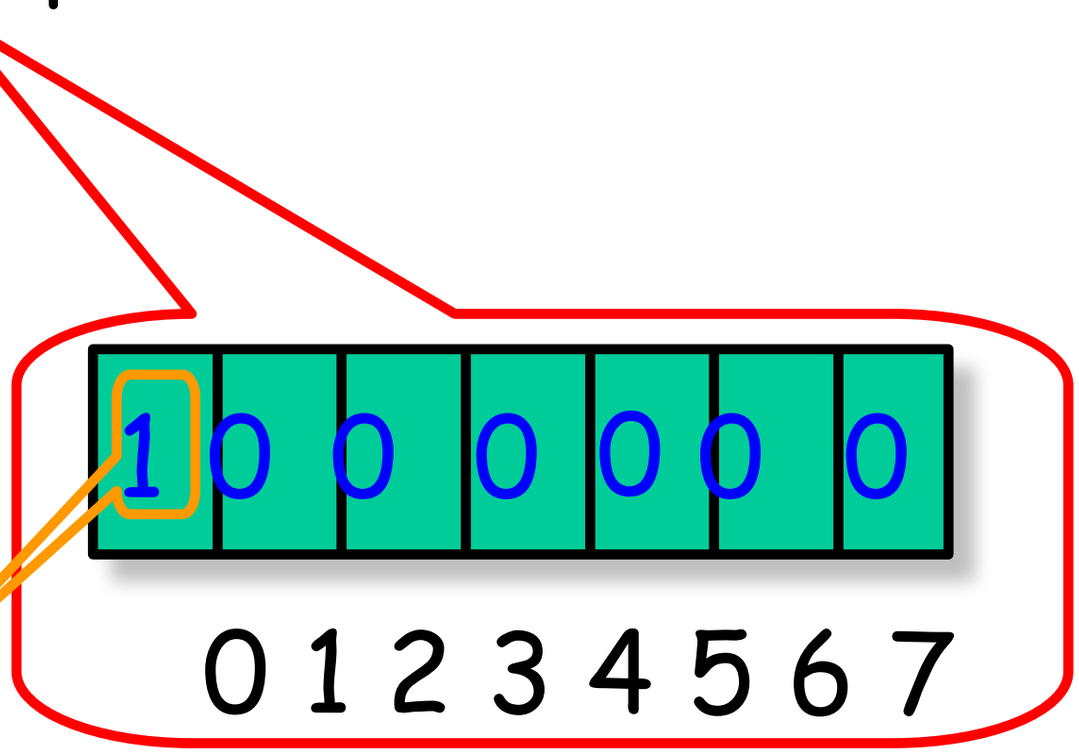
Le programme…

- SRSW Booléen sûr
- MRSW Booléen sûr
- MRSW Booléen régulier
- MRSW regulier
- MRSW atomique
- MRMW atomique

Multi-Valué

Représentation unaire:

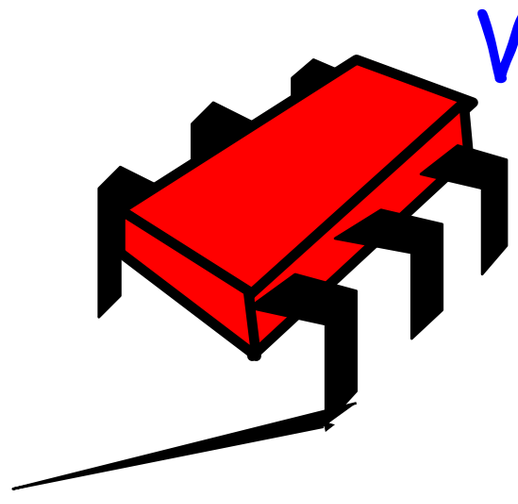
bit[i] -> valeur i

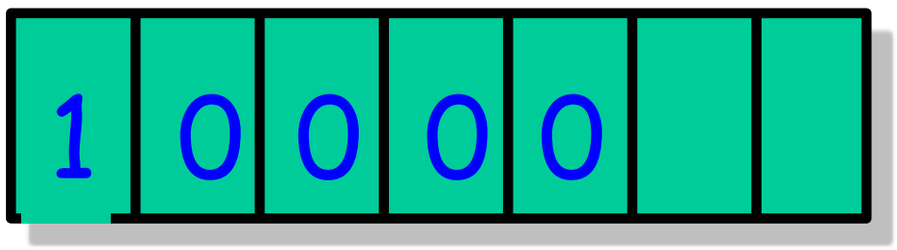


initialement 0

Ecriture

Write 5



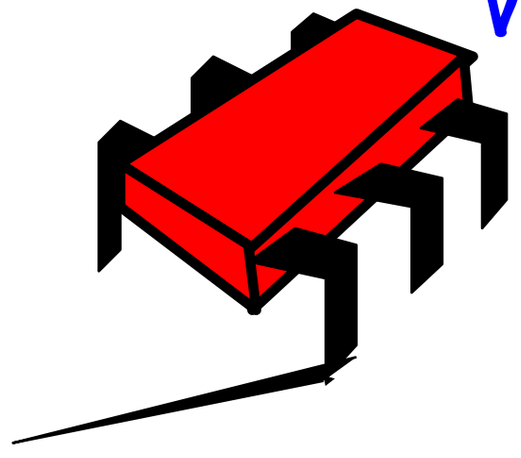


0 1 2 3 4 5 6 7

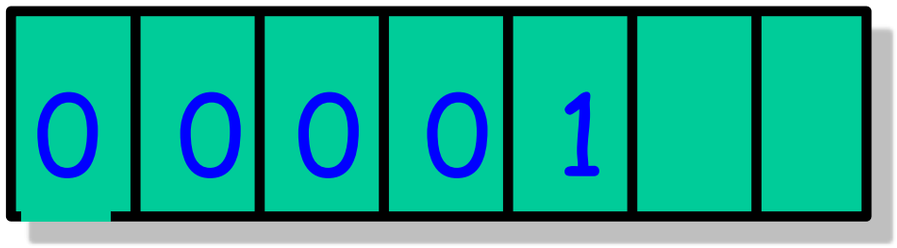
registres

Ecriture

Write 5

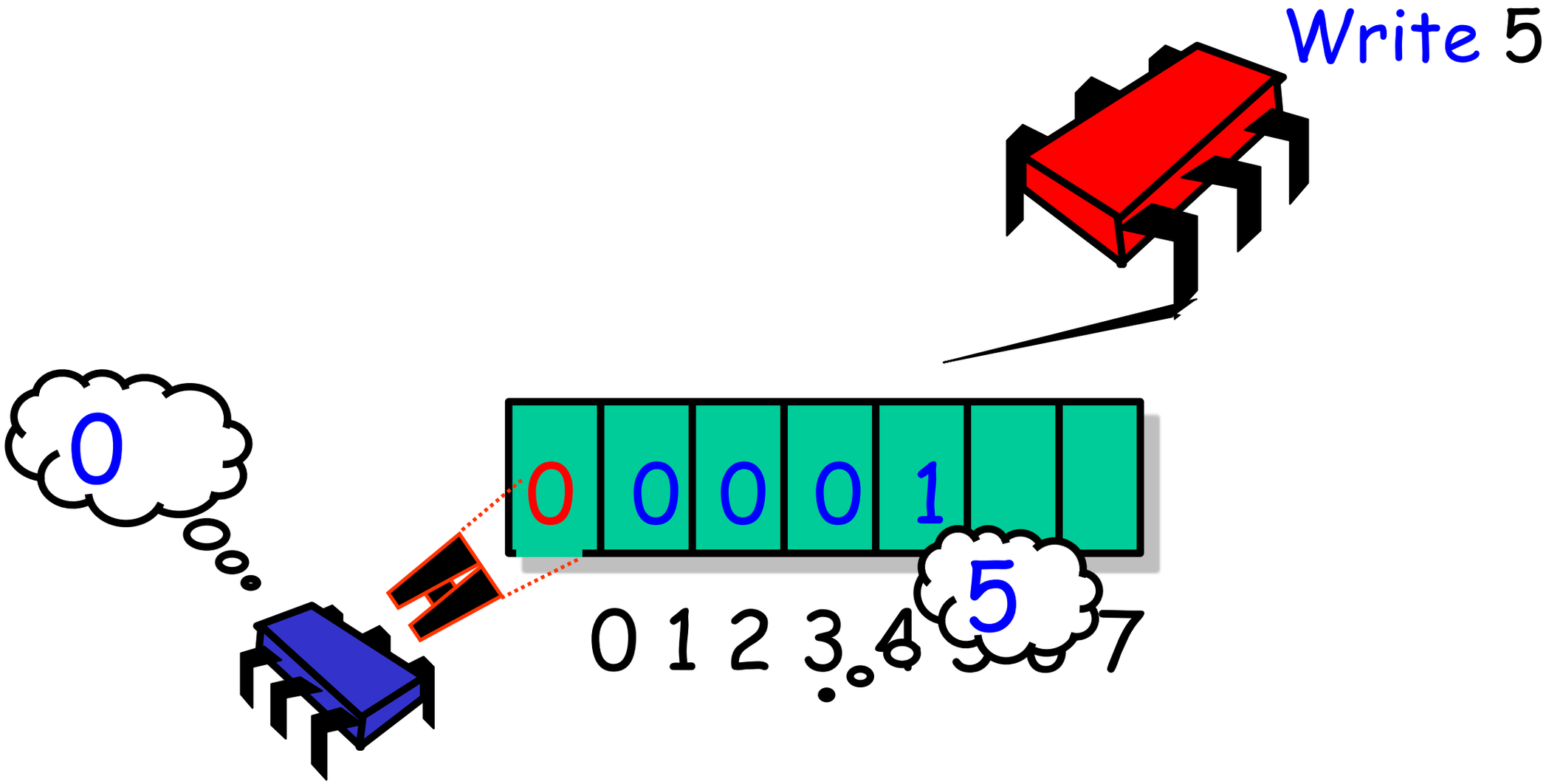


Initialement 0



0 1 2 3 4 5 6 7

Ecriture



MRSW Régulier Multi-valué à partir de MRSW Booléens Réguliers

```
public class RegMRSWRegister implements Register{
  RegBoolMRSWRegister[M] bit;

  public void write(int x) {
    this.bit[x].write(true);
    for (int i=x-1; i>=0; i--)
      this.bit[i].write(false);
  }

  public int read() {
    for (int i=0; i < M; i++)
      if (this.bit[i].read())
        return i;
   }}
```

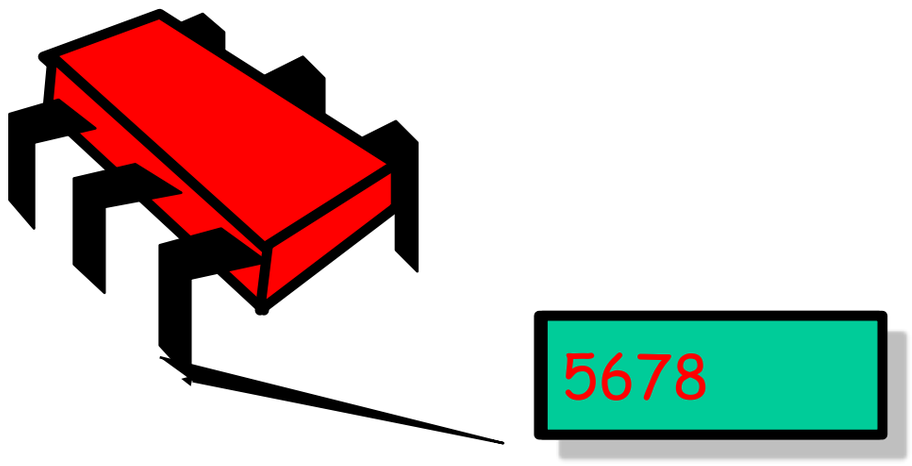
Le programme…

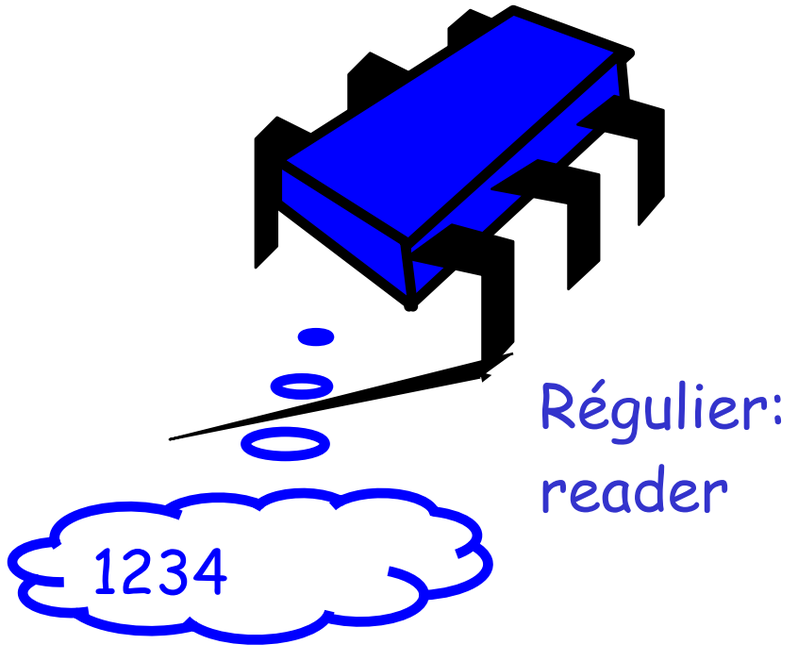
- SRSW Booléen sûr
- MRSW Booléen sûr
- MRSW Booléen régulier
- MRSW regulier
- MRSW atomique
- MRMW atomique

SWSR

SRSW Atomique à partir de SRSW Régulier

Régulier writer





Régulier: reader

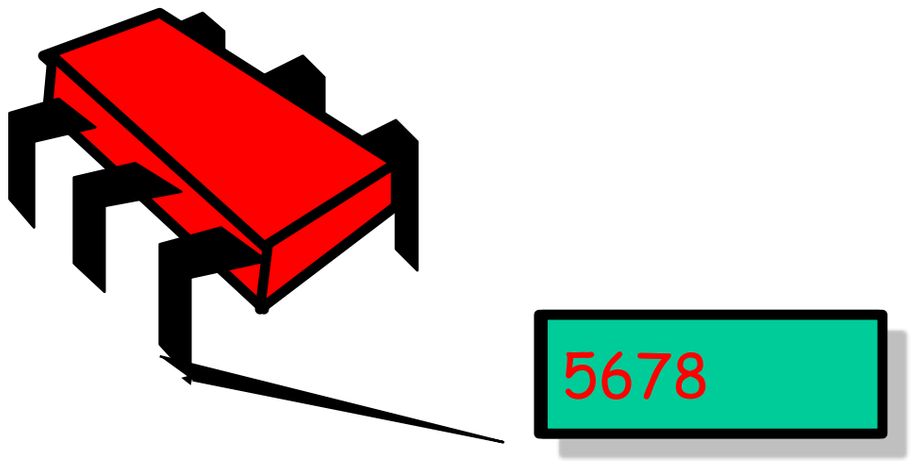
et pas 5678...

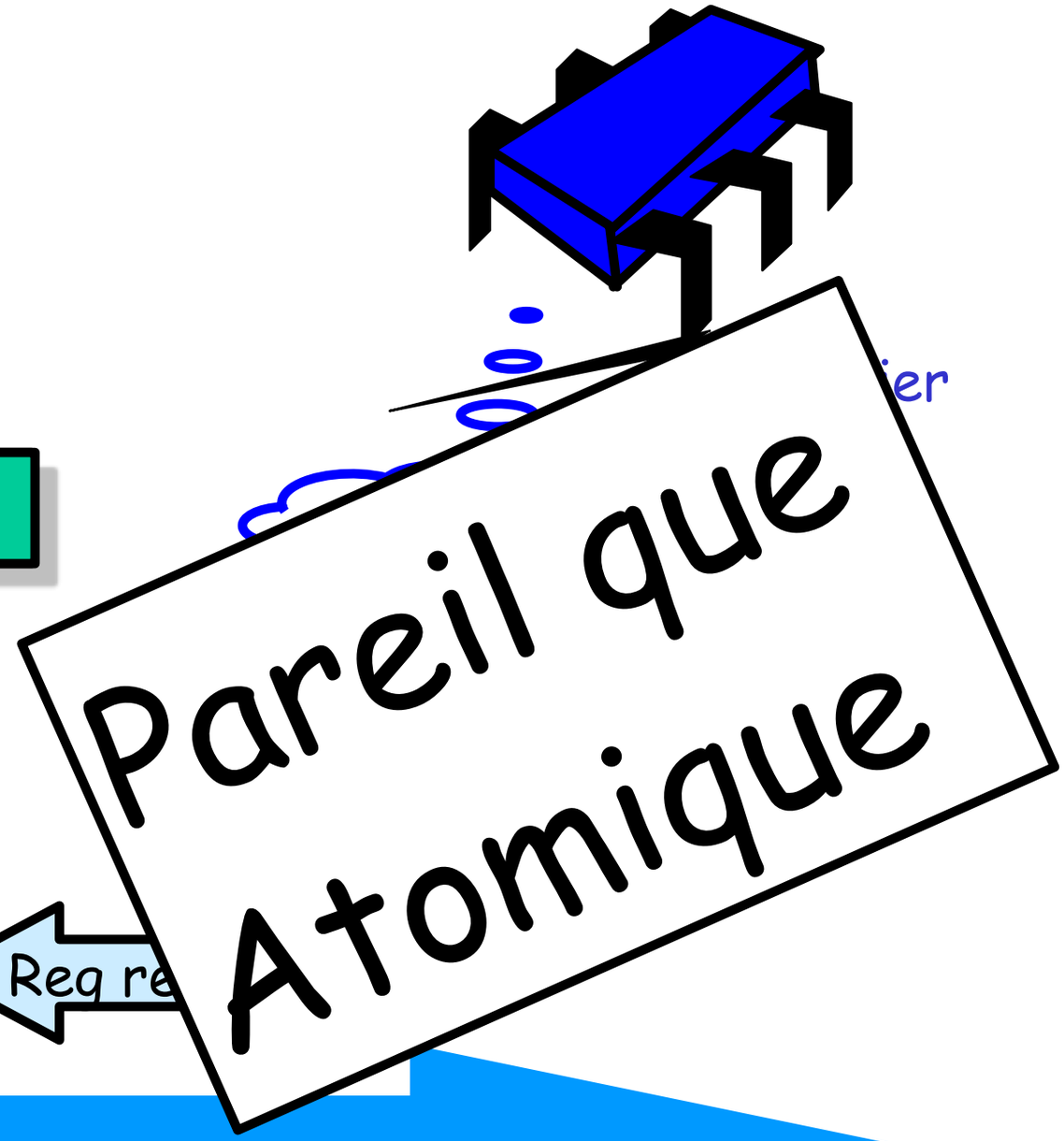
lecture concurrente

Problème?

SRSW Atomique à partir de SRSW Régulier

Régulier writer





initialement
1234

Reg write(5678)

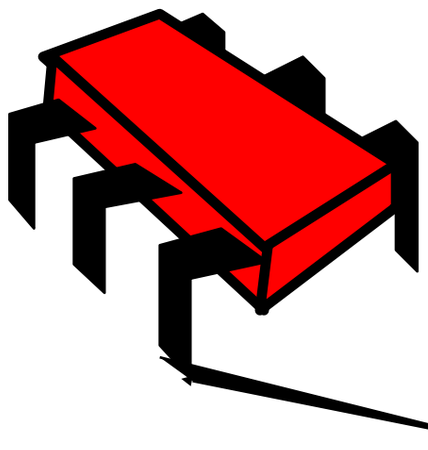
Reg re

time

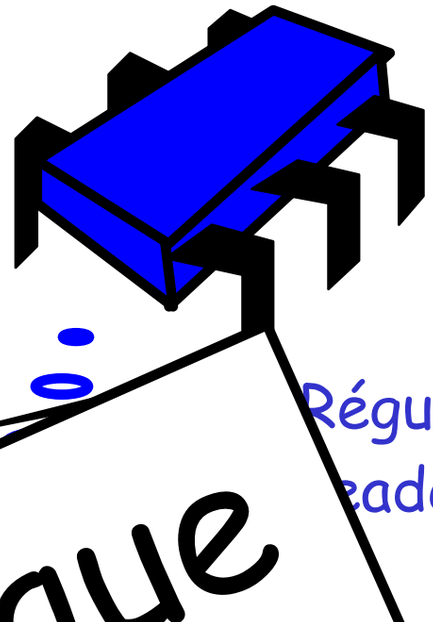
registres

SRSW Atomique à partir de SRSW Régulier

Régulier writer



5678



Régulier reader

Pareil que Atomique

Initialement
1234

Reg write(5678)

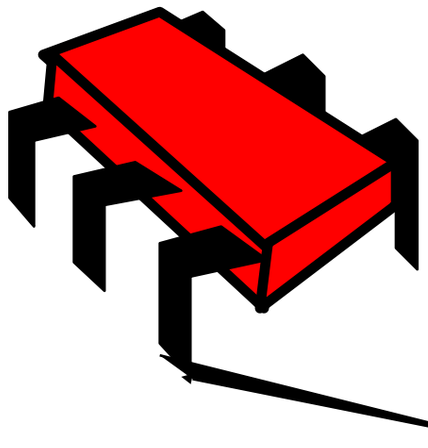
Reg read(1234)

time

registres

SRSW Atomique à partir de SRSW Régulier

Régulier writer



5678

Initialement
1234

Reg write(5678)

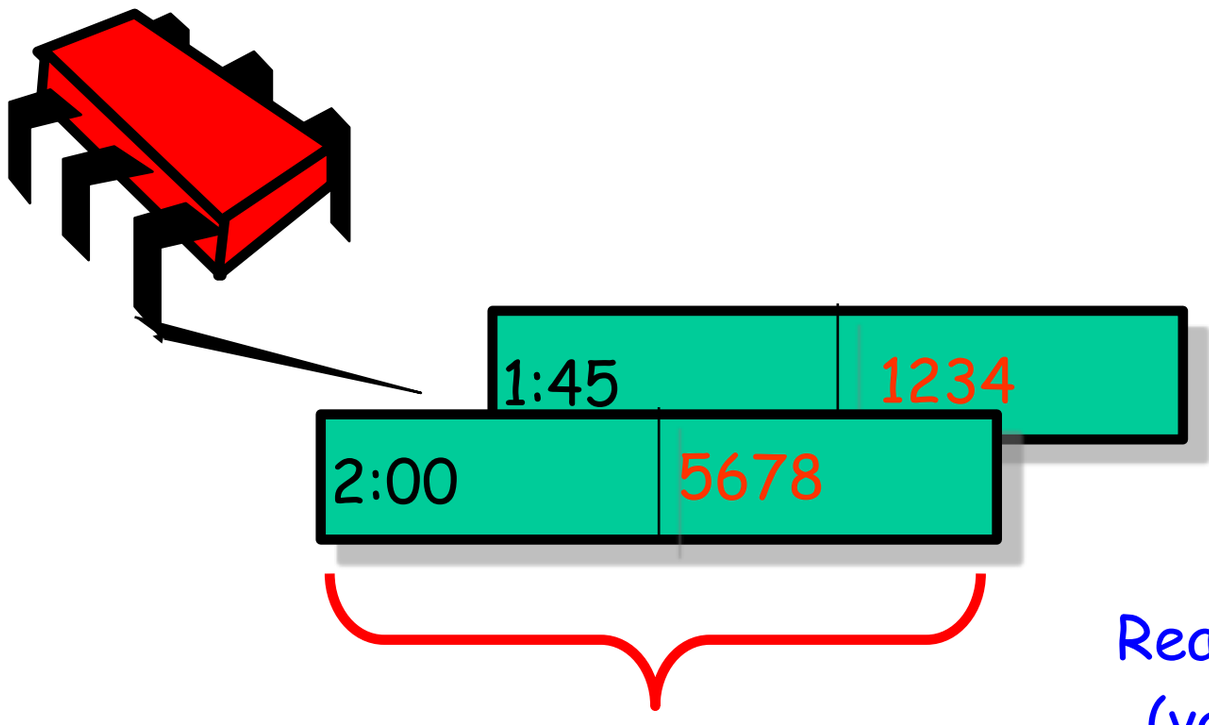
Reg read(5678)

Reg read(1234)

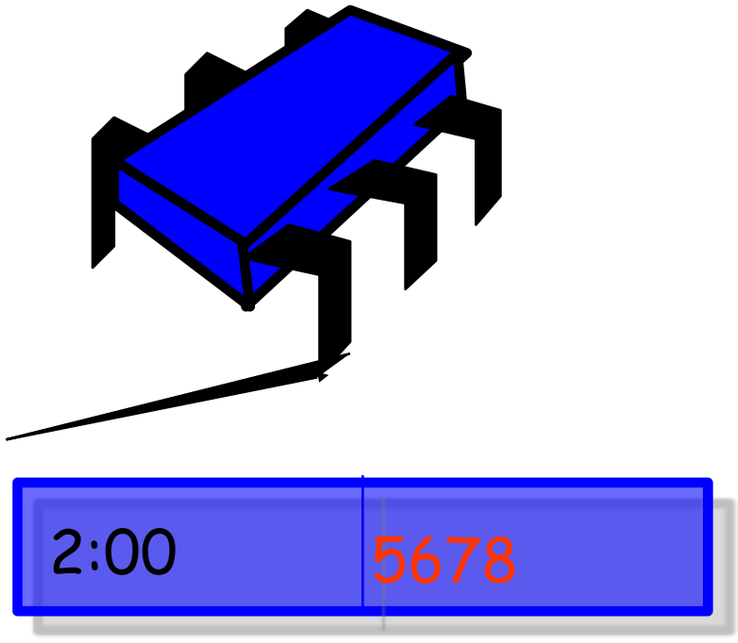
Write 5678
Réalisé

registres

Valeur avec Timestamp



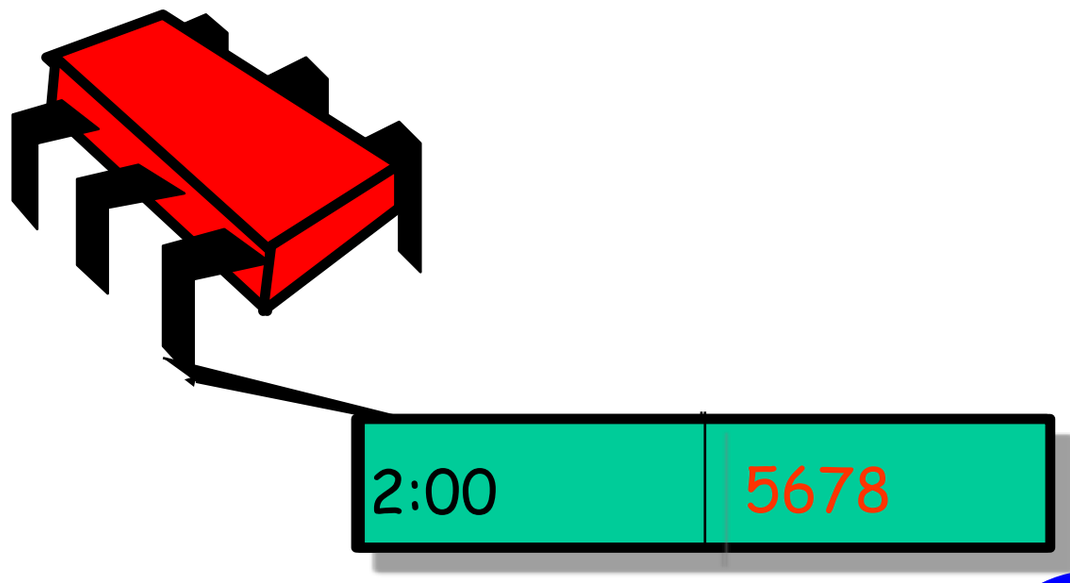
Writer écrit la valeur
et le timestamp

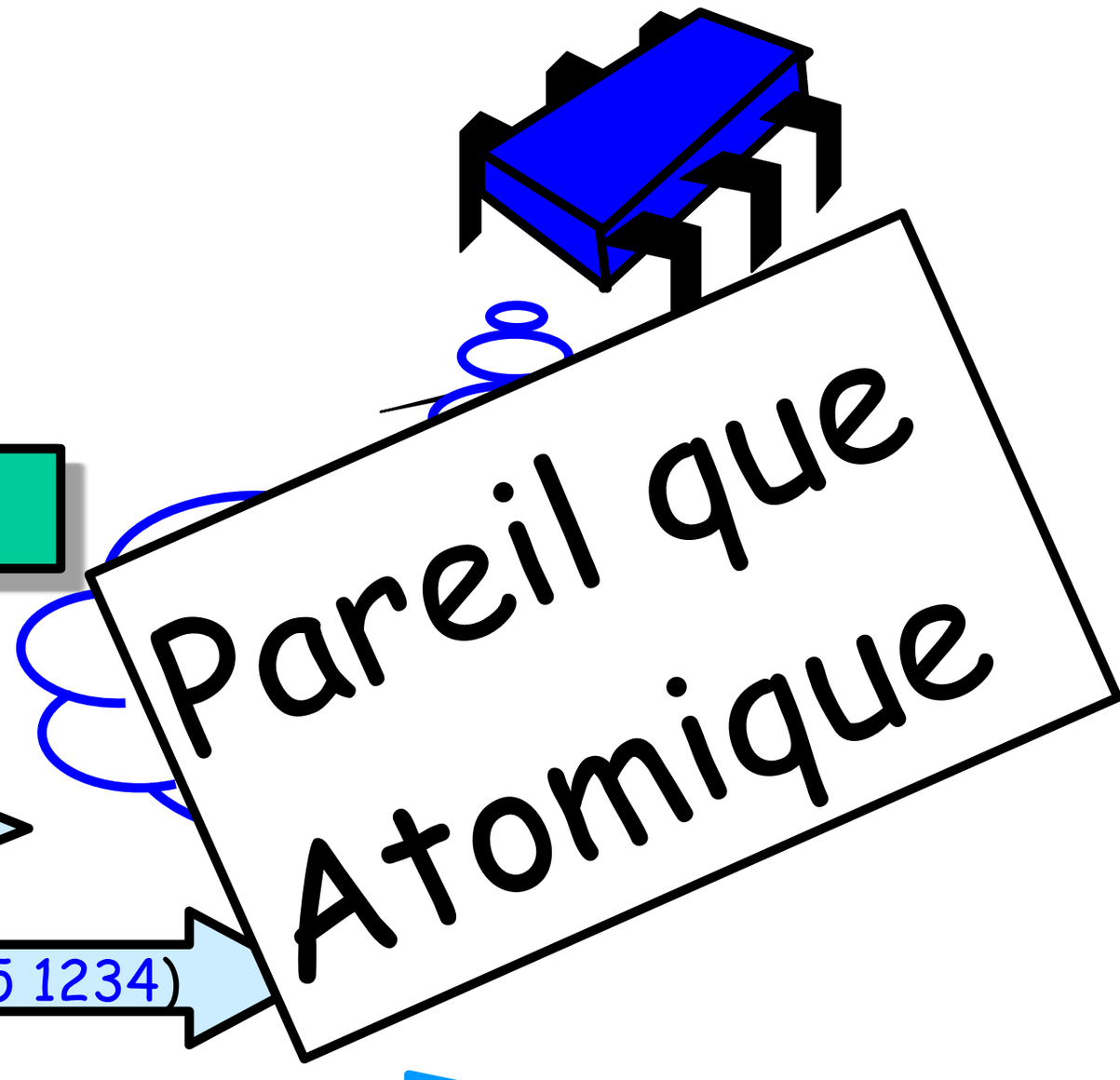


Reader sauve le dernier read (valeur,stamp) et retourne une nouvelle valeur seulement si le timestamp est plus grand

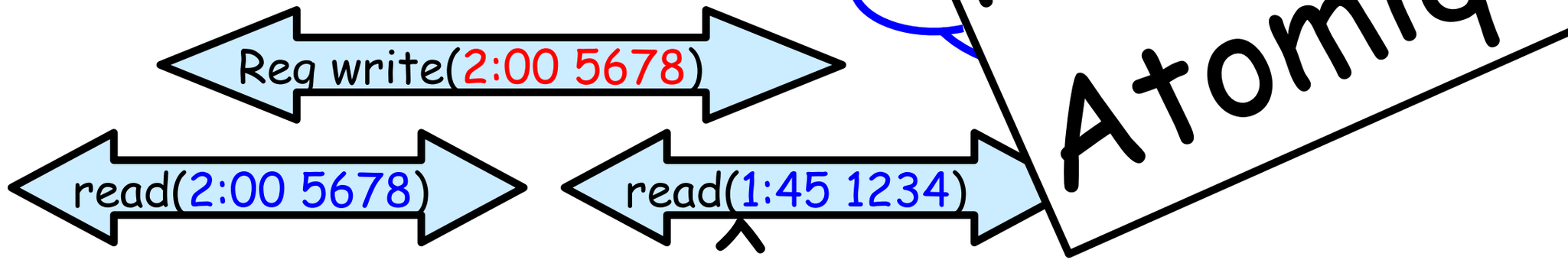
SRSW Atomique à partir de SRSW Régulier

writer



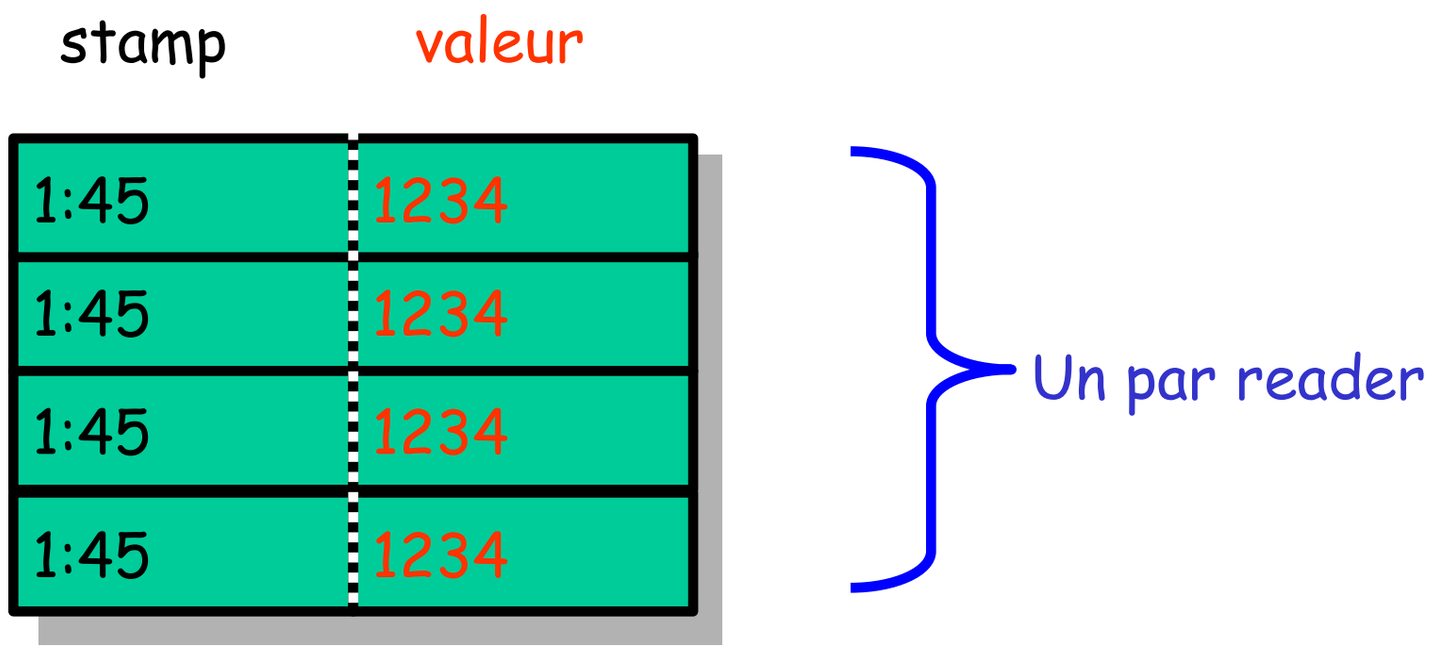


1:45
1234



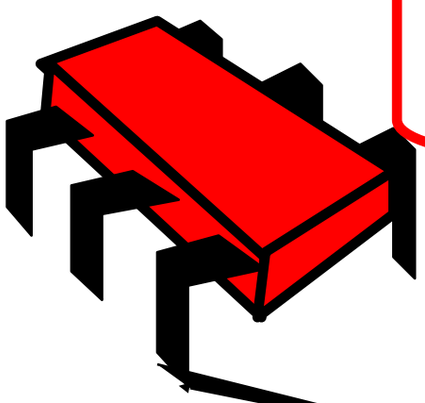
registres

de Single Reader Atomique à Multi-Reader Atomique



registres

Scenario

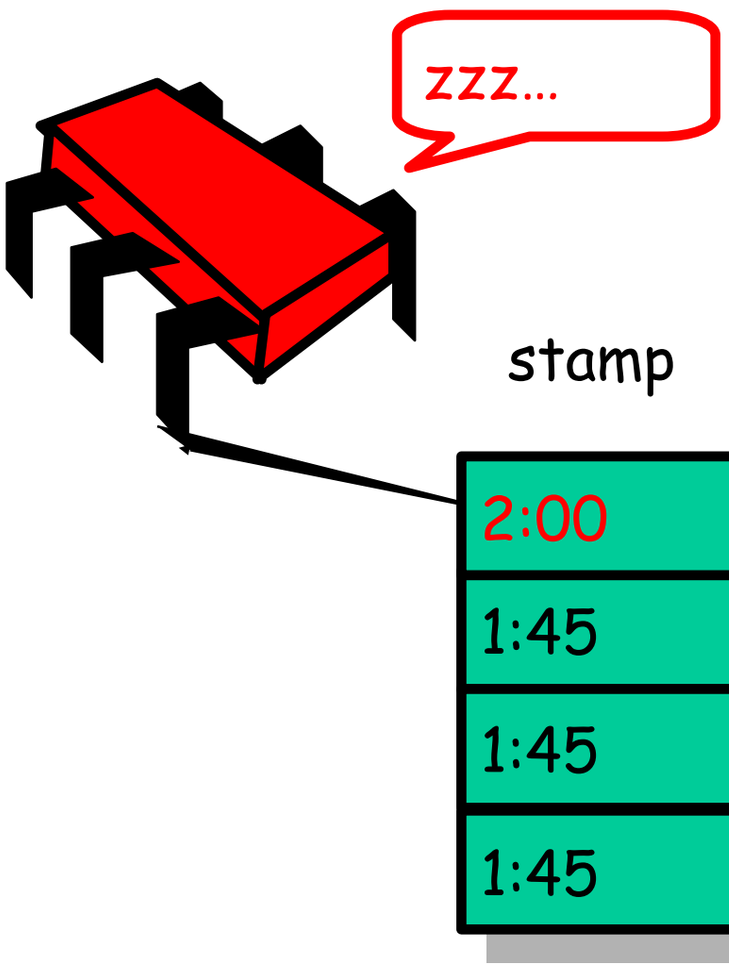


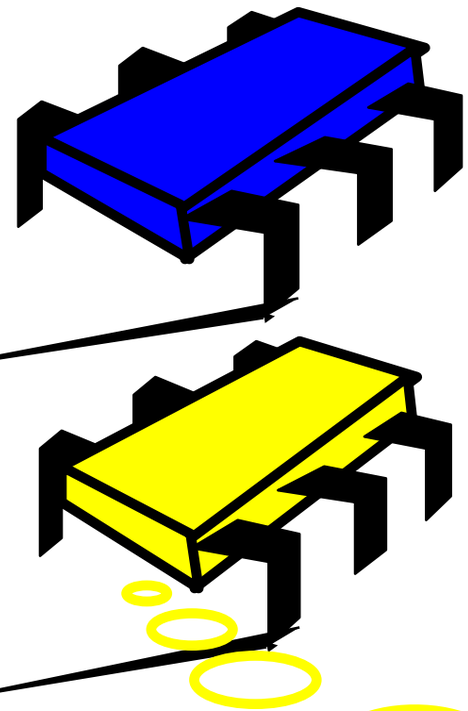
Writer commence
son write

stamp	valeur
2:00	5678
1:45	1234
1:45	1234
1:45	1234

registres

Scenario…





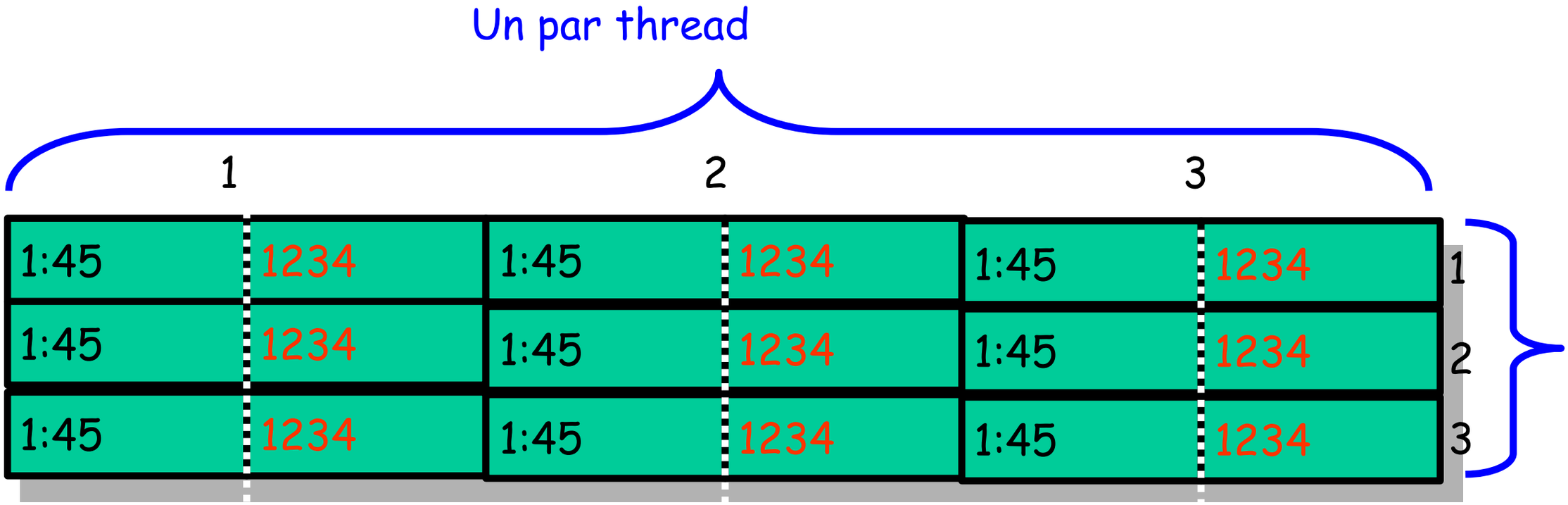
reader
lit

stamp	value
2:00	5678
1:45	1234
1:45	1234
1:45	1234

reader
(après)

Le Jaune retourne une value antérieure alors qu'il est après Bleu:non linéarisable!

Multi-Reader

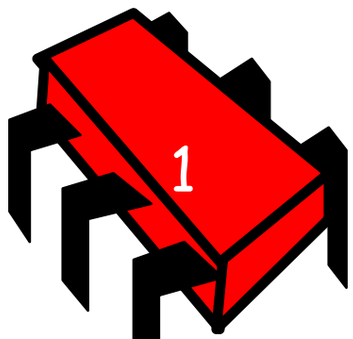


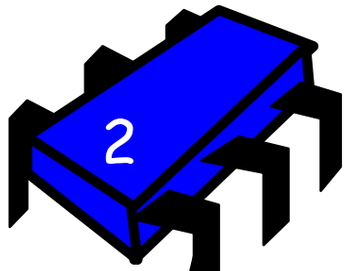
Multi-Reader

Writer écrit une colonne

2:00, 5678

reader
lit une ligne





1		2		3		
2:00	5678	1:45	1234	1:45	1234	1
2:00	5678	1:45	1234	1:45	1234	2
2:00	5678	1:45	1234	1:45	1234	3

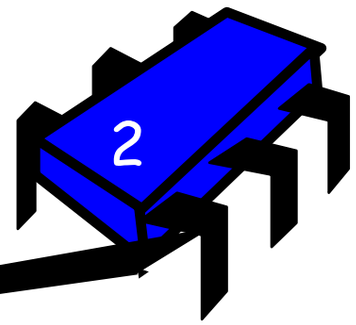
registres

i-Reader

zzz…après le second write

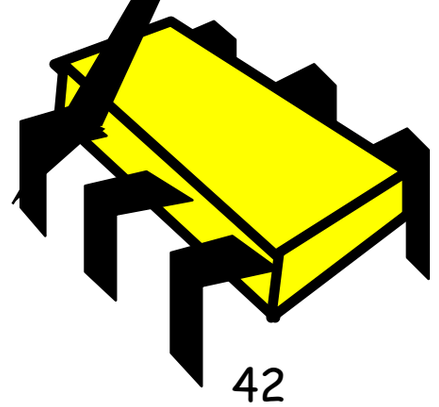
2:00, 5678

reader écrit la colonne pour dire aux autres ce qu'il a lu



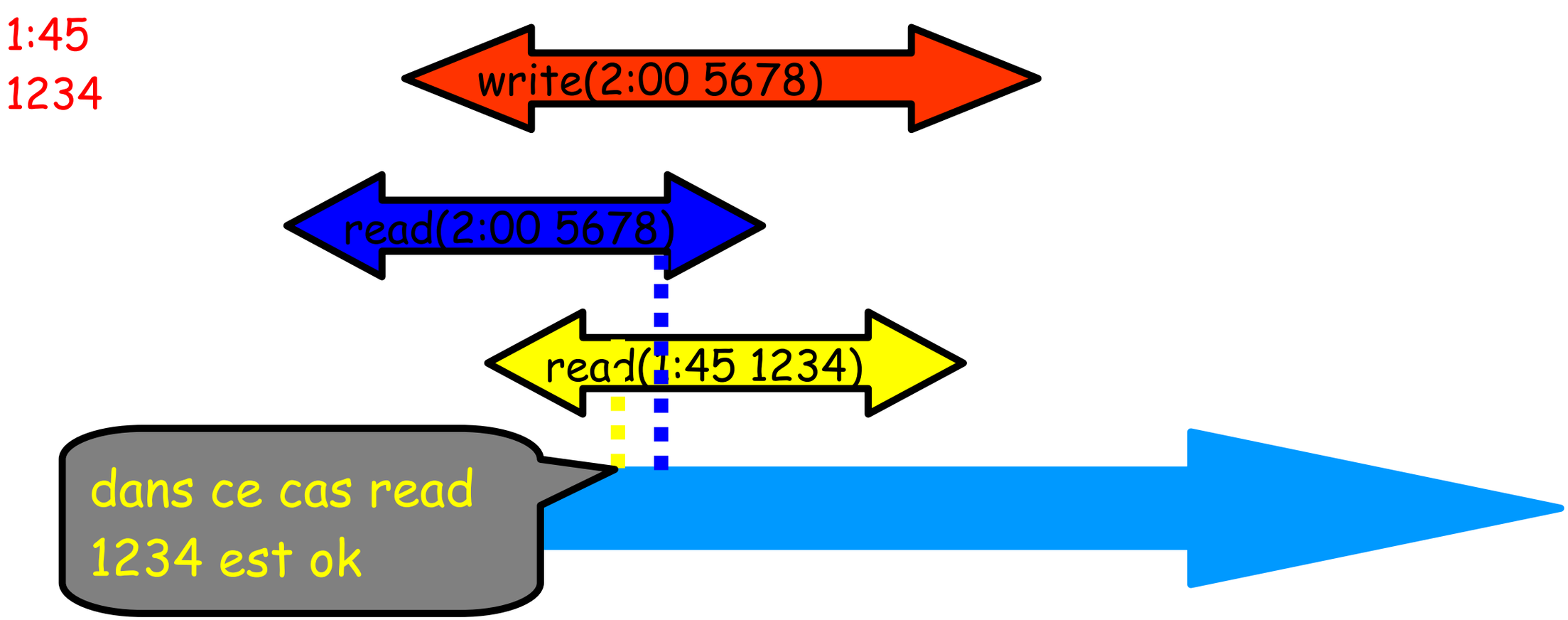
1		2		3	
2:00	5678	2:00	5678	1:45	1234
2:00	5678	2:00	5678	1:45	1234
1:45	1234	2:00	5678	1:45	1234

Jaune lira la nouvelle valeur dans la colonne écrite par Bleu



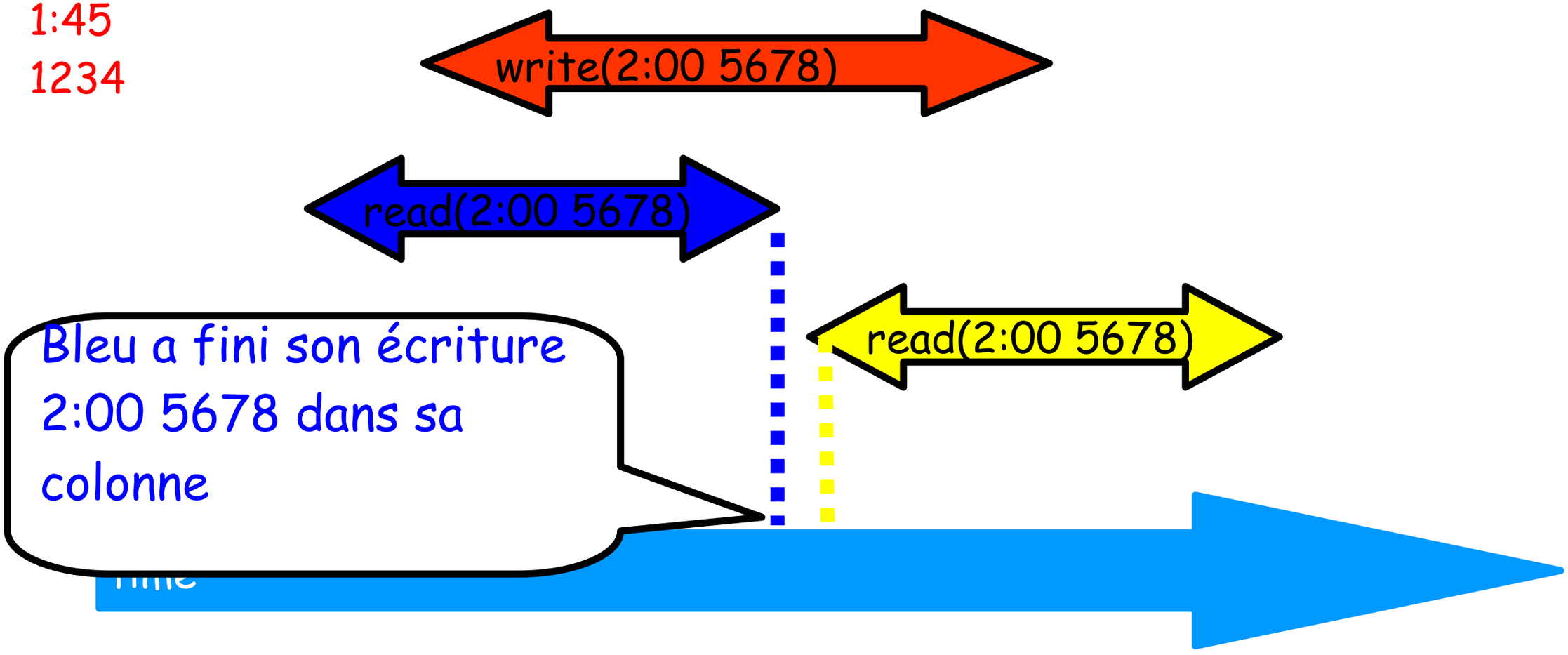
registres

Jaune peut-il rater la mise à jour de Bleu?
…seulement en cas de concurrence



Si non concurrence tout va bien!

1:45
1234

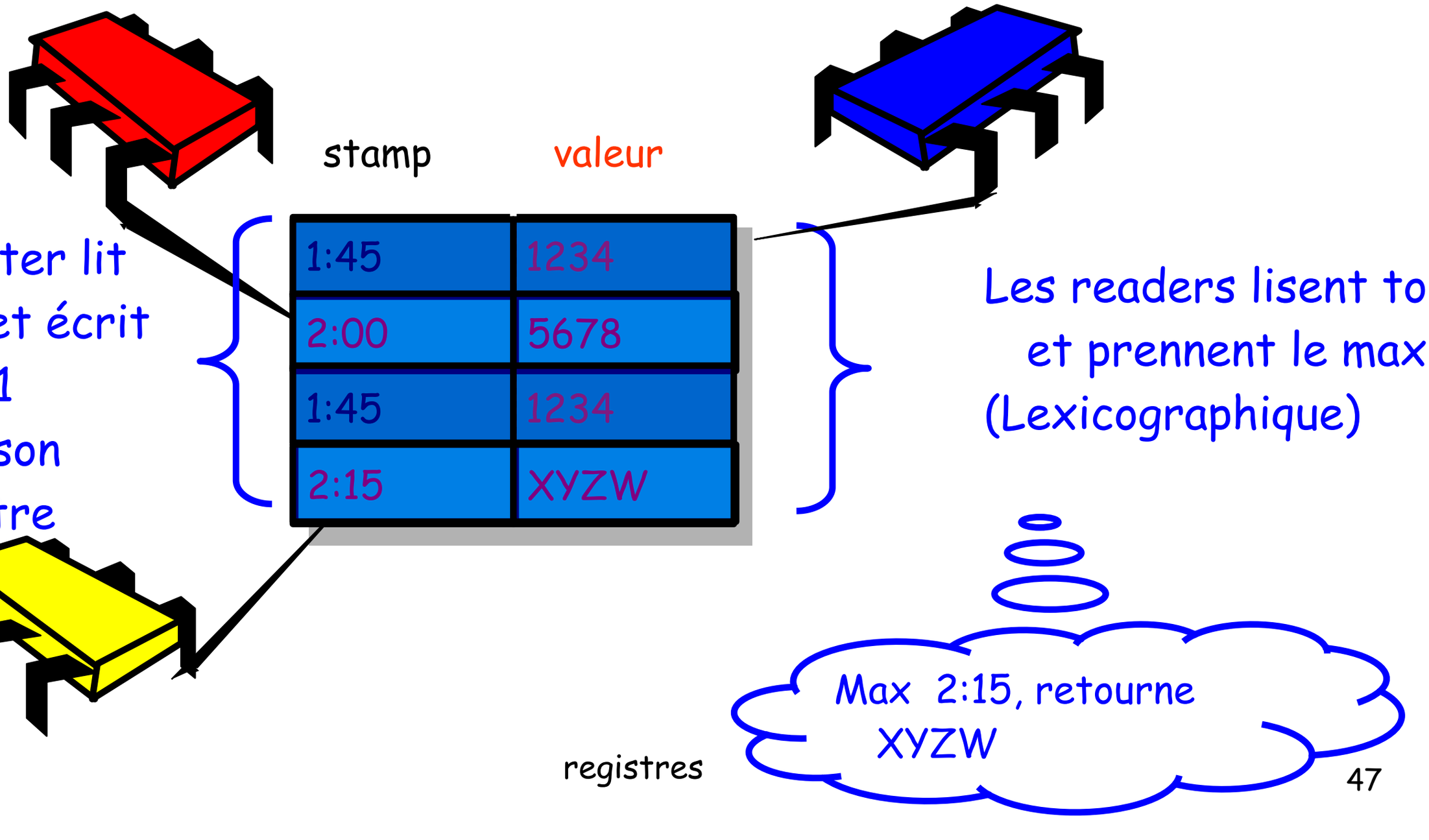


```
1  public class AtomicMRSWRegister<T> implements Register<T> {
2    ThreadLocal<Long> lastStamp;
3    private StampedValue<T>[][] a_table; // each entry is SRSW atomic
4    public AtomicMRSWRegister(T init, int readers) {
5      lastStamp = new ThreadLocal<Long>() {
6         protected Long initialValue() { return 0; };
7      };
8      a_table = (StampedValue<T>[][]) new StampedValue[readers][readers];
9      StampedValue<T> value = new StampedValue<T>(init);
10     for (int i = 0; i < readers; i++) {
11       for (int j = 0; j < readers; j++) {
12         a_table[i][j] = value;
13       }
14     }
15   }
16   public T read() {
17     int me = ThreadID.get();
18     StampedValue<T> value = a_table[me][me];
19     for (int i = 0; i < a_table.length; i++) {
20       value = StampedValue.max(value, a_table[i][me]);
21     }
22     for (int i = 0; i < a_table.length; i++) {
23       a_table[me][i] = value;
24     }
25     return value;
26   }
27   public void write(T v) {
28     long stamp = lastStamp.get() + 1;
29     lastStamp.set(stamp);
30     StampedValue<T> value = new StampedValue<T>(stamp, v);
31     for (int i = 0; i < a_table.length; i++) {
32       a_table[i][i] = value;
33     }
34   }
35 }
```

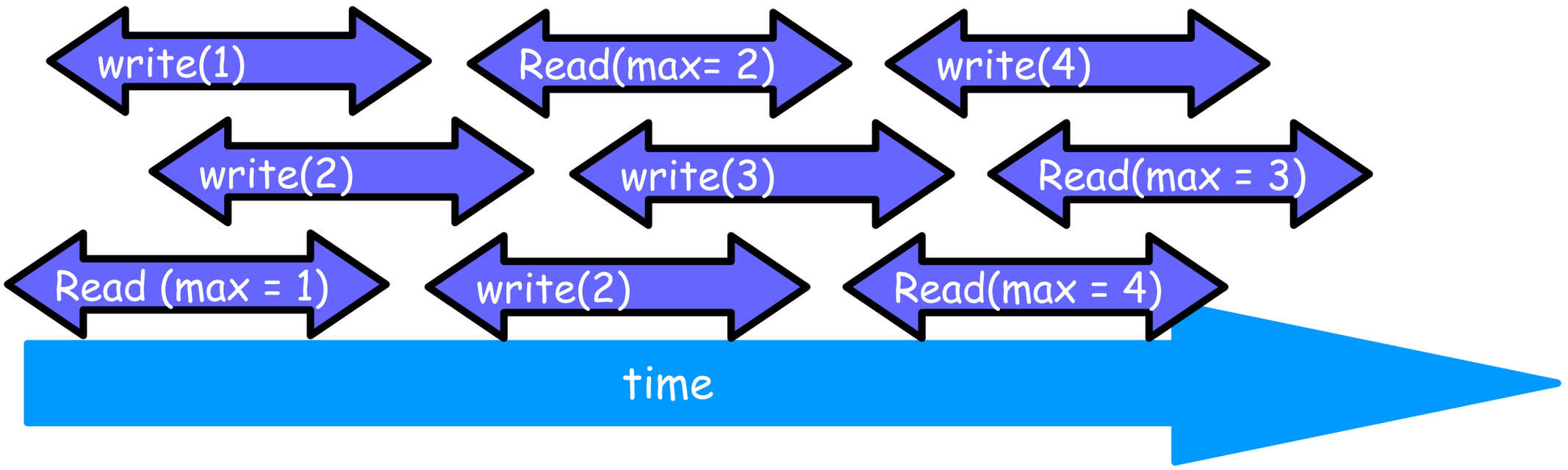
Le programme…

- SRSW Booléen sûr
- MRSW Booléen sûr
- MRSW Booléen régulier
- MRSW regulier
- MRSW atomique
- MRMW atomique

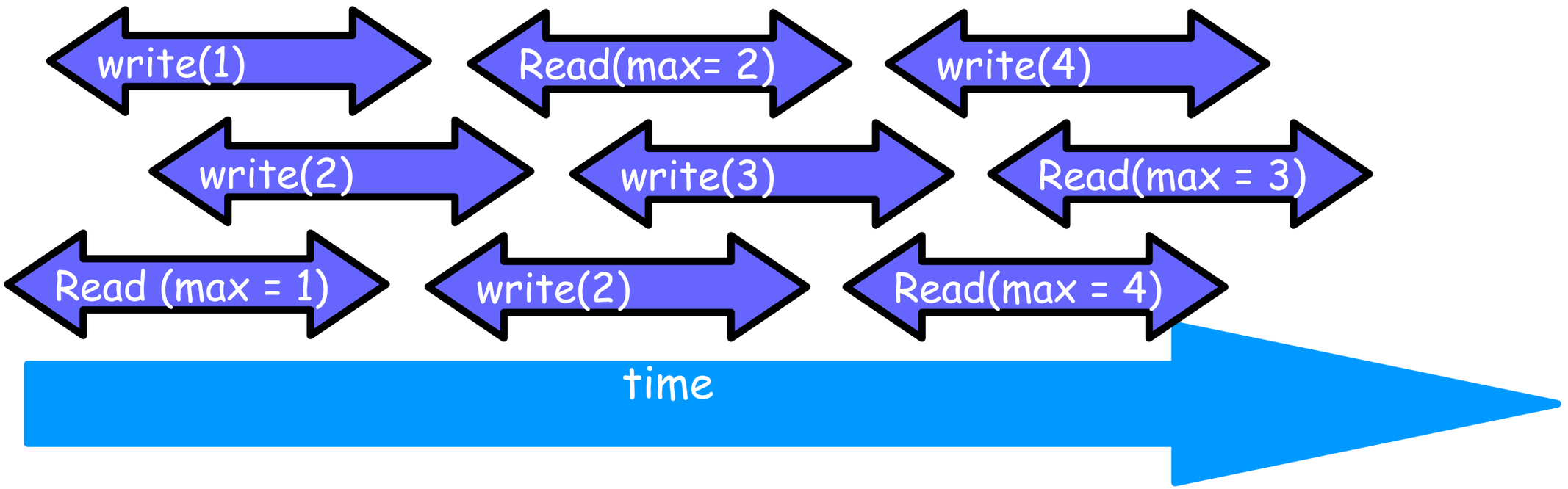
Multi-Writer Atomique à partir de Multi-Reader Atomique



Linearizsable

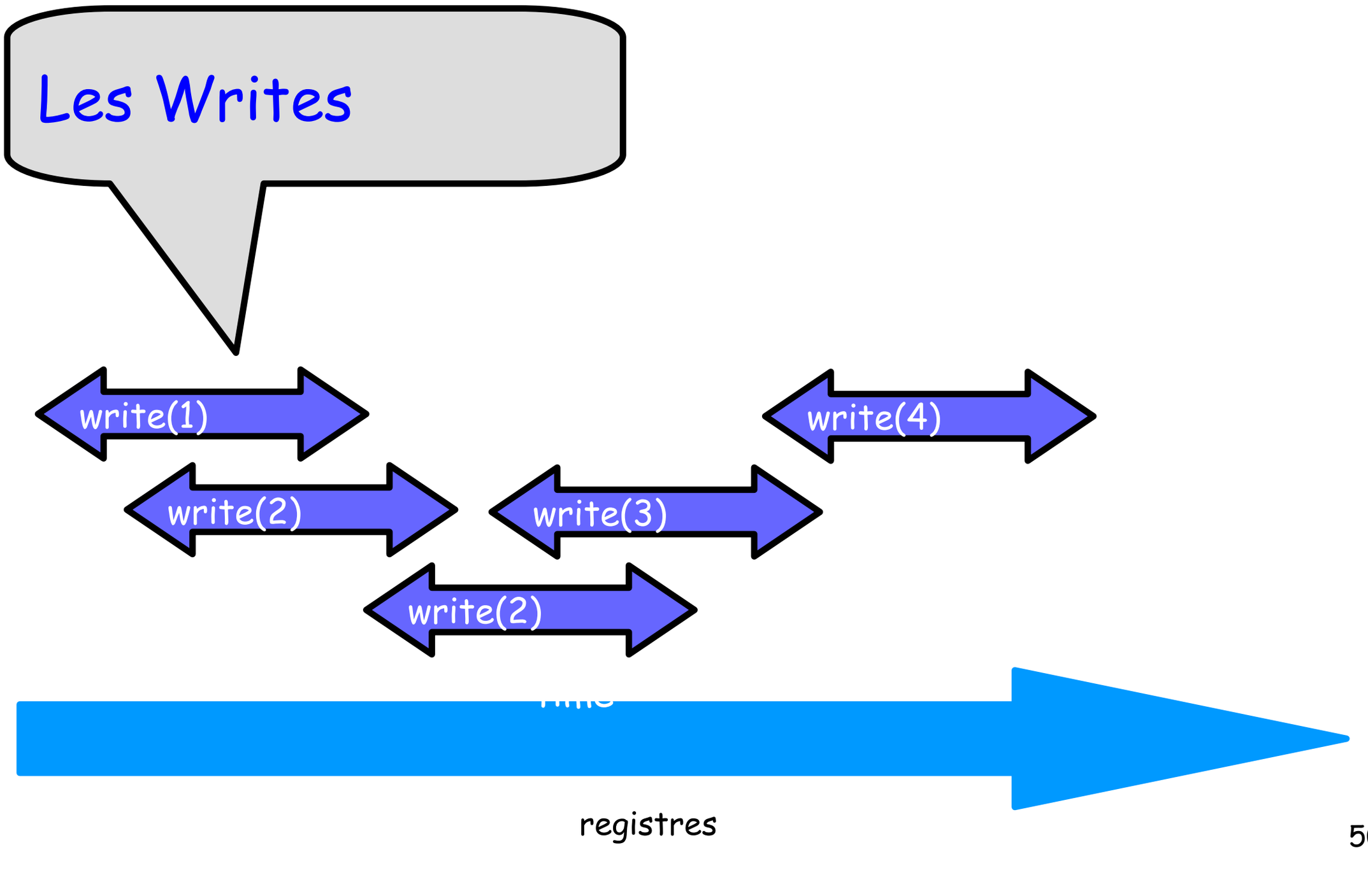


Points de Linéarisation

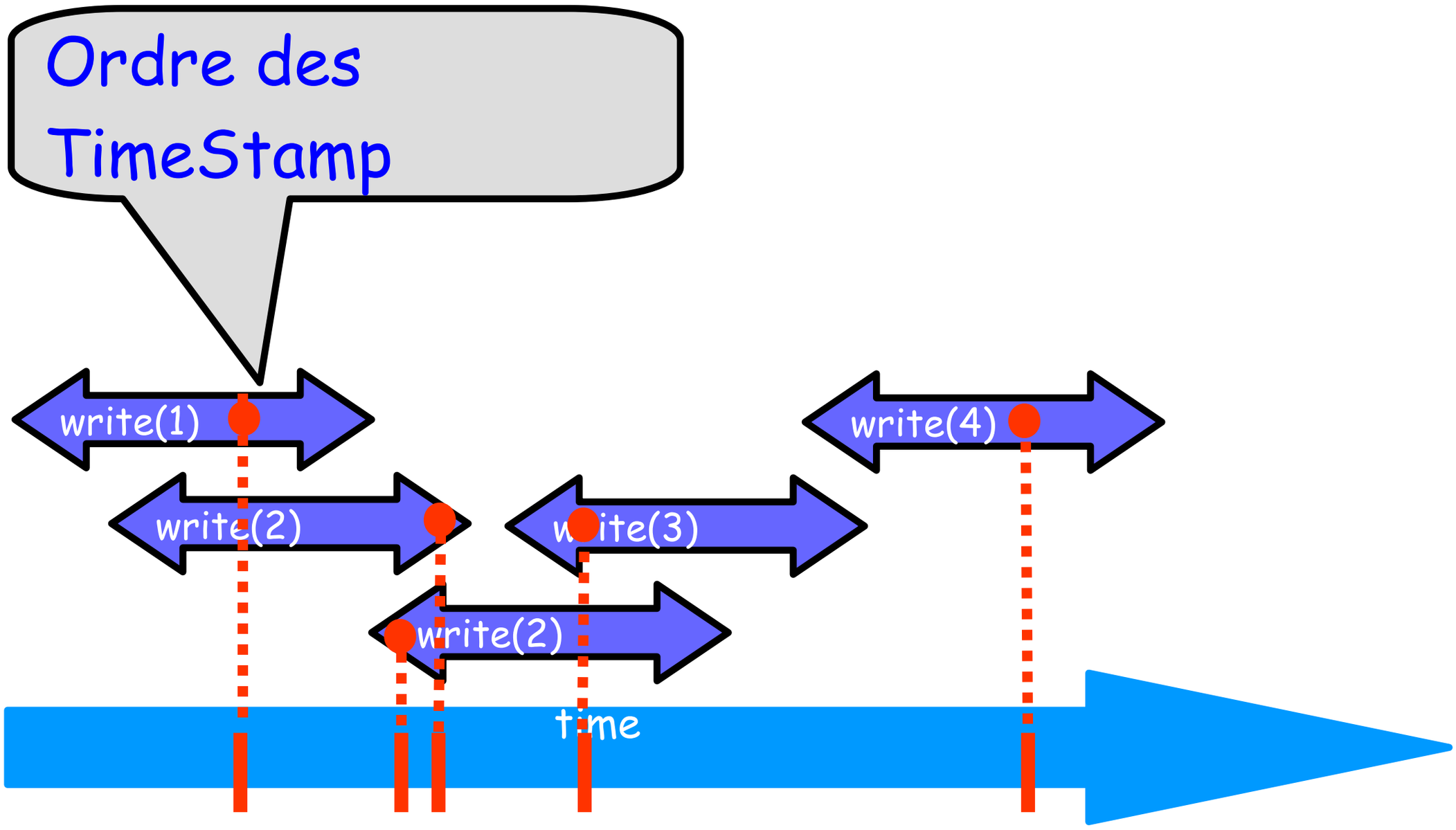


registres

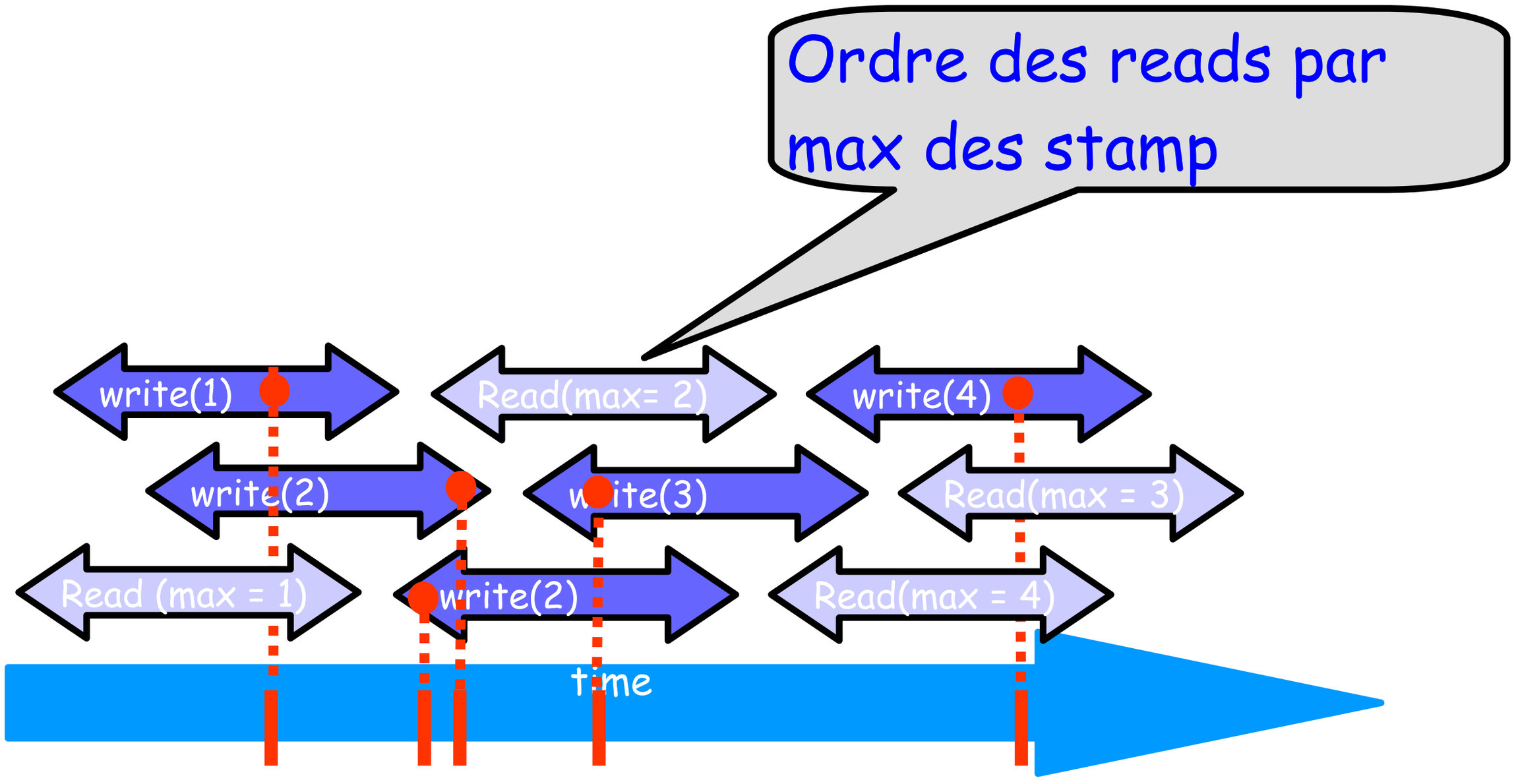
Points de Linéarisation



Points de Linéarisation

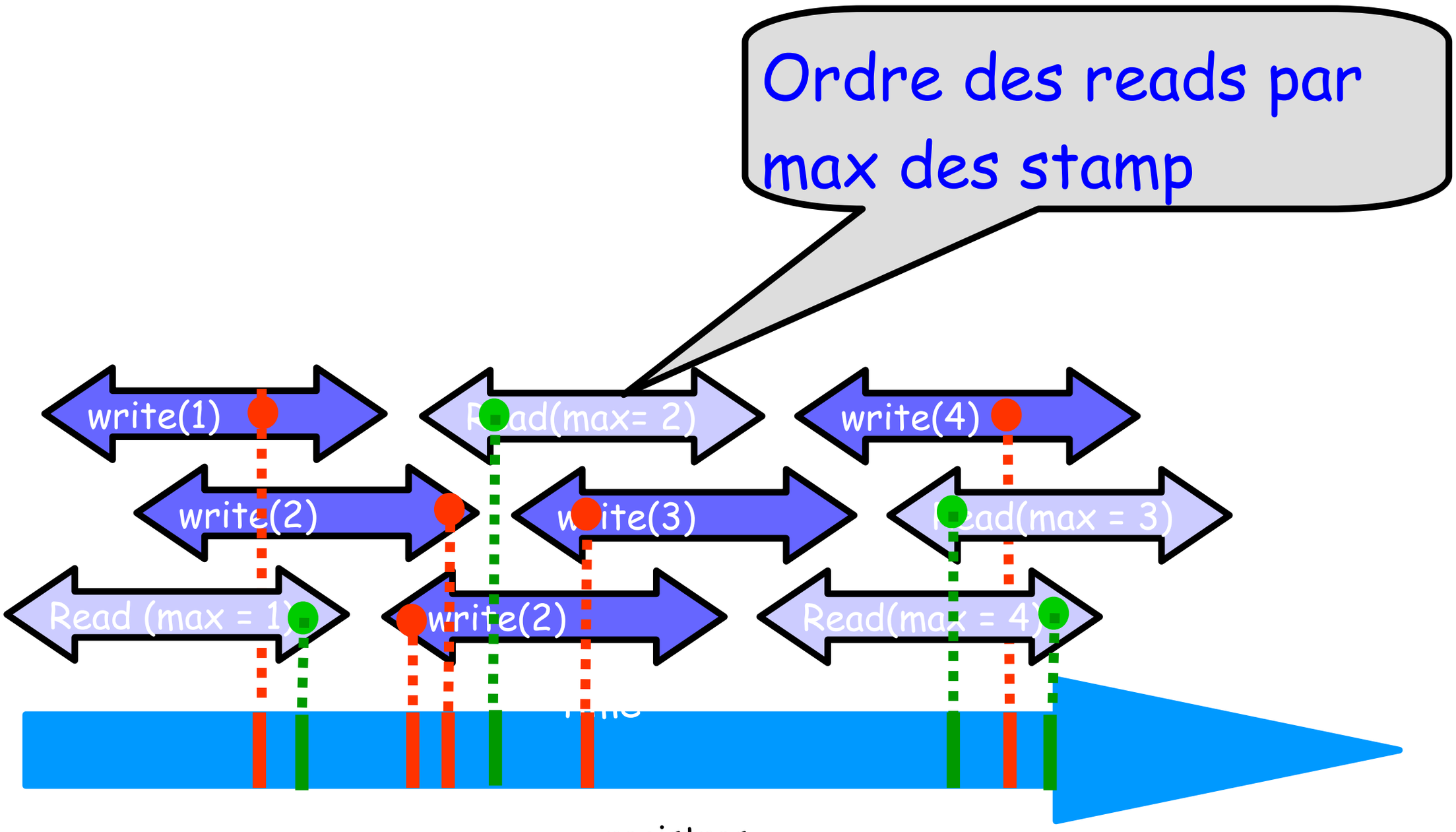


Points de Linéarisation



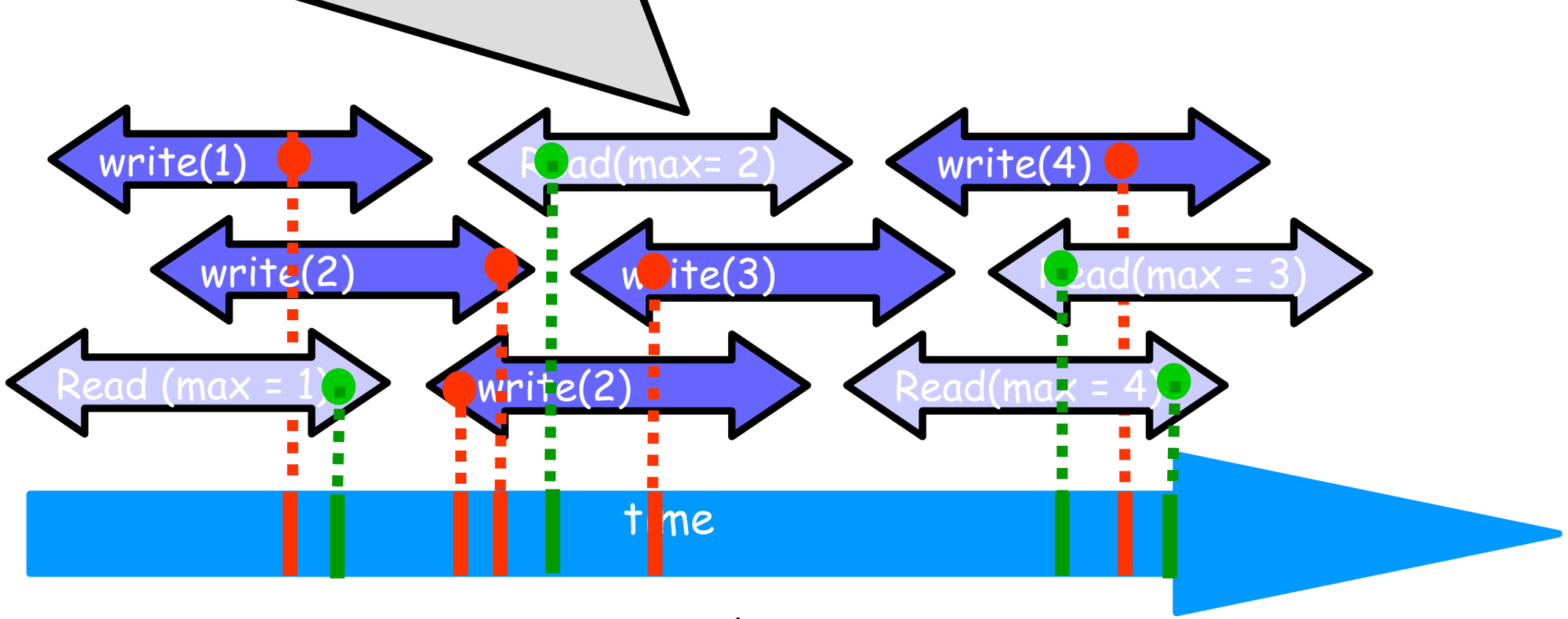
registres

Points de Linéarisation



Points de Linéarisation

Le point de linéarisation dépend de l'exécution



registres

```
1  public class AtomicMRMWRegister<T> implements Register<T>{
2    private StampedValue<T>[] a_table; // array of atomic MRSW registers
3    public AtomicMRMWRegister(int capacity, T init) {
4      a_table = (StampedValue<T>[]) new StampedValue[capacity];
5      StampedValue<T> value = new StampedValue<T>(init);
6      for (int j = 0; j < a_table.length; j++) {
7        a_table[j] = value;
8      }
9    }
10   public void write(T value) {
11     int me = ThreadID.get();
12     StampedValue<T> max = StampedValue.MIN_VALUE;
13     for (int i = 0; i < a_table.length; i++) {
14       max = StampedValue.max(max, a_table[i]);
15     }
16     a_table[me] = new StampedValue(max.stamp + 1, value);
17   }
18   public T read() {
19     StampedValue<T> max = StampedValue.MIN_VALUE;
20     for (int i = 0; i < a_table.length; i++) {
21       max = StampedValue.max(max, a_table[i]);
22     }
23     return max.value;
24   }
25 }
```