Sémantique et compilation d'un langage synchrone fonctionnel à rafales

Adrien Guatto et Albert Cohen Louis Mandel Marc Pouzet

ENS Paris

Inria

Collège de France

ENS Paris / UPMC

Plan

Introduction

Horloges Entières

Le langage

Un compilateur expérimental

Conclusion

Plan

Introduction

Horloges Entières

Le langage

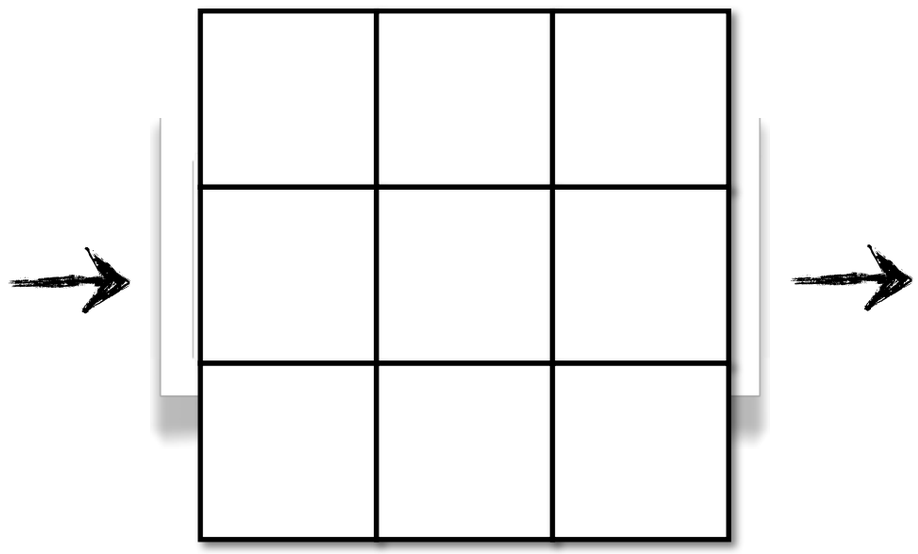
Un compilateur expérimental

Conclusion

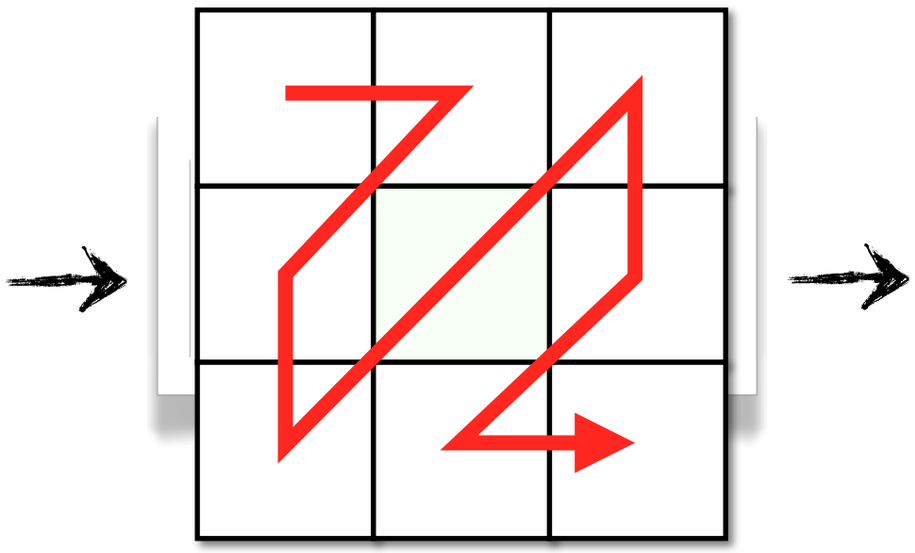
Parcours Zig-Zag JPEG

Parcours Zig-Zag JPEG

Parcours Zig-Zag JPEG

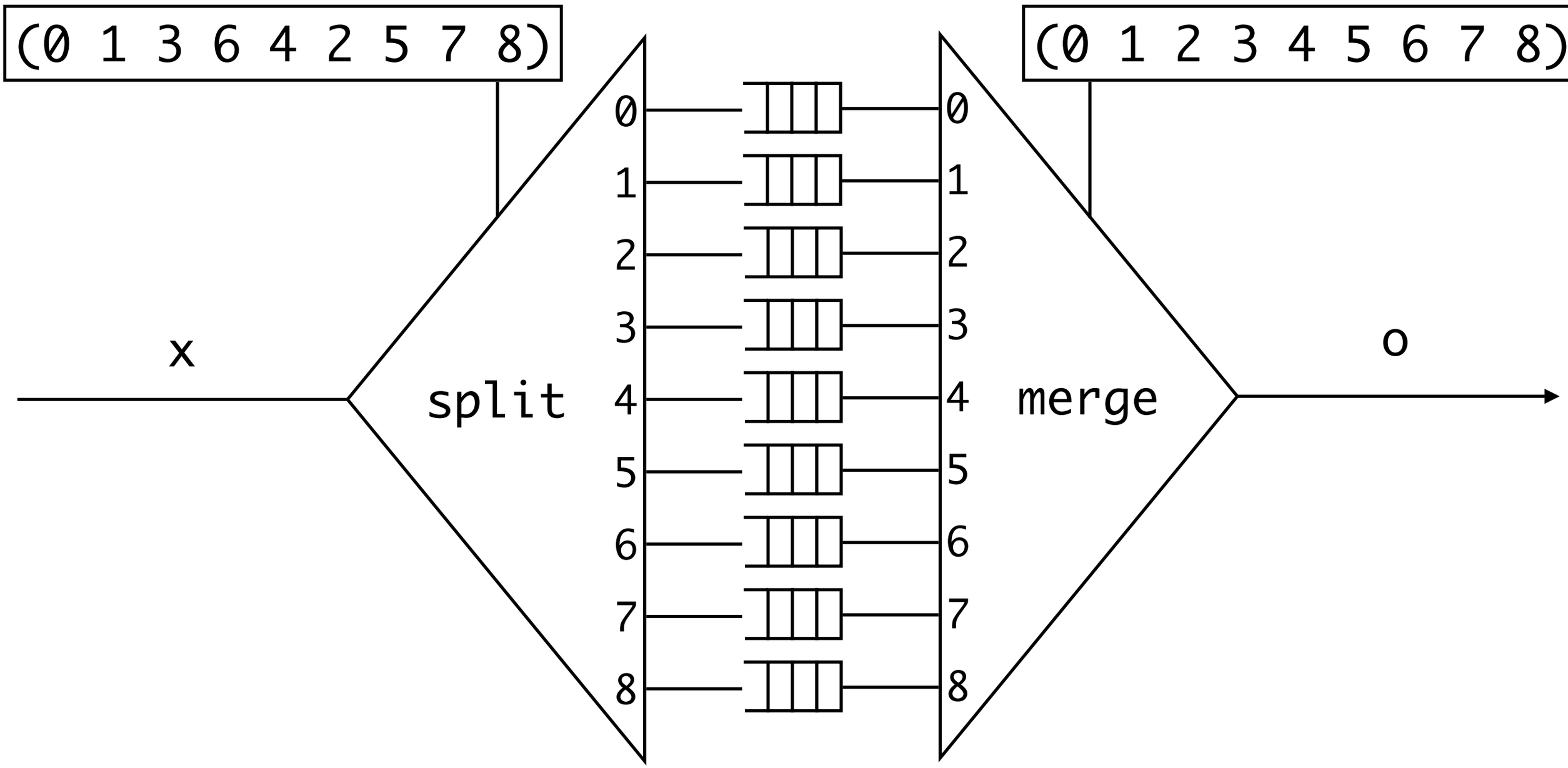


Parcours Zig-Zag JPEG



Parcours Zig-Zag JPEG

Parcours Zig-Zag JPEG, dataflow



Parcours Zig-Zag JPEG en AcidS

```
let node zigzag x = o where
    rec (x0, x1, x2, x3, x4, x5, x6, x7, x8)
          = split x
              with (0 1 3 6 4 2 5 7 8)
              by (0, 1, 2, 3, 4, 5, 6, 7, 8)
and o = merge (0 1 2 3 4 5 6 7 8) with
            | 0 -> buffer x0
            | 1 -> buffer x1
             ...
            | 8 -> buffer x8
          end
```

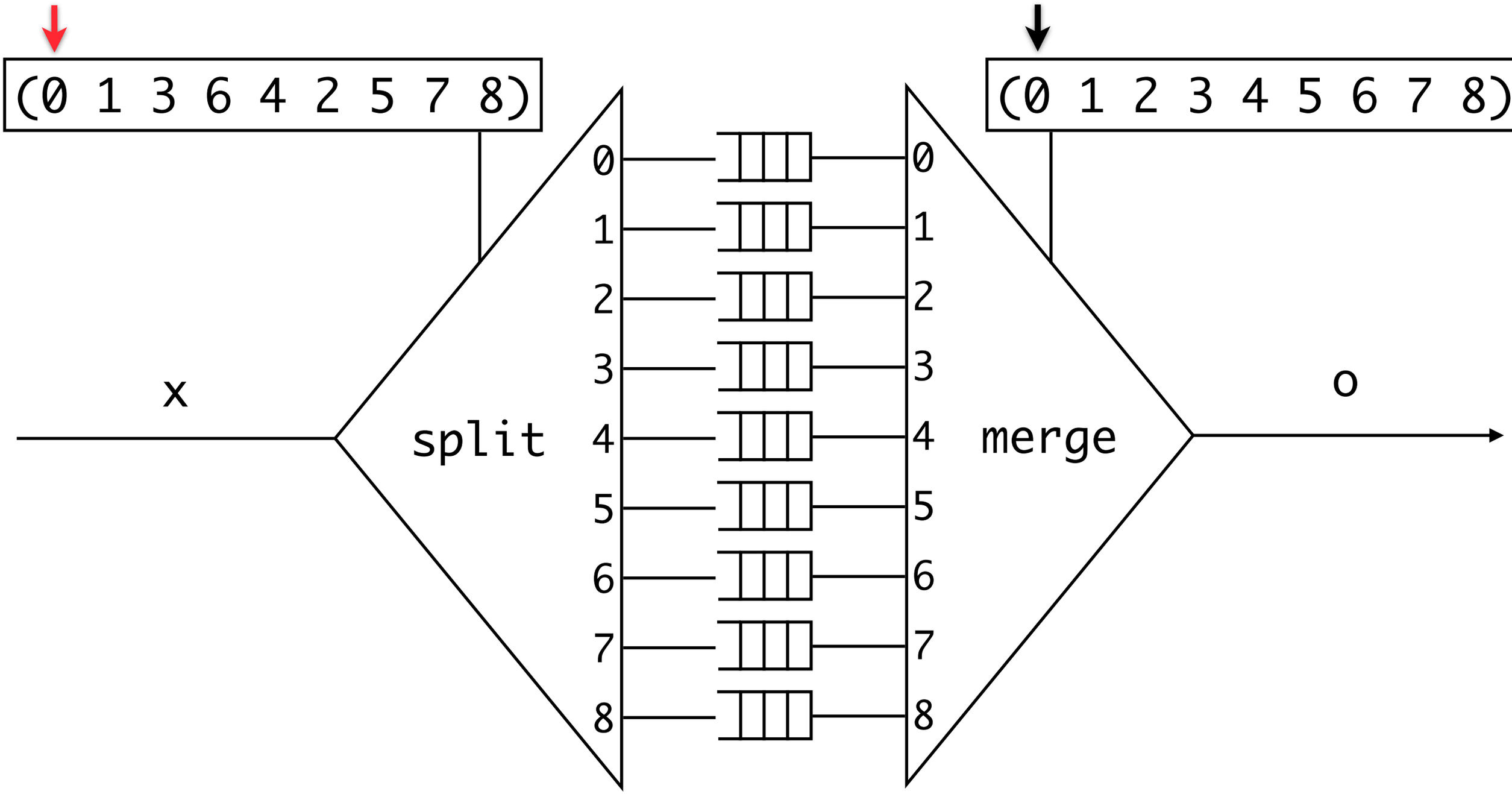
Parcours Zig-Zag JPEG en AcidS

```
let node zigzag x = o where
    rec (x0, x1, x2, x3, x4, x5, x6, x7, x8)
```

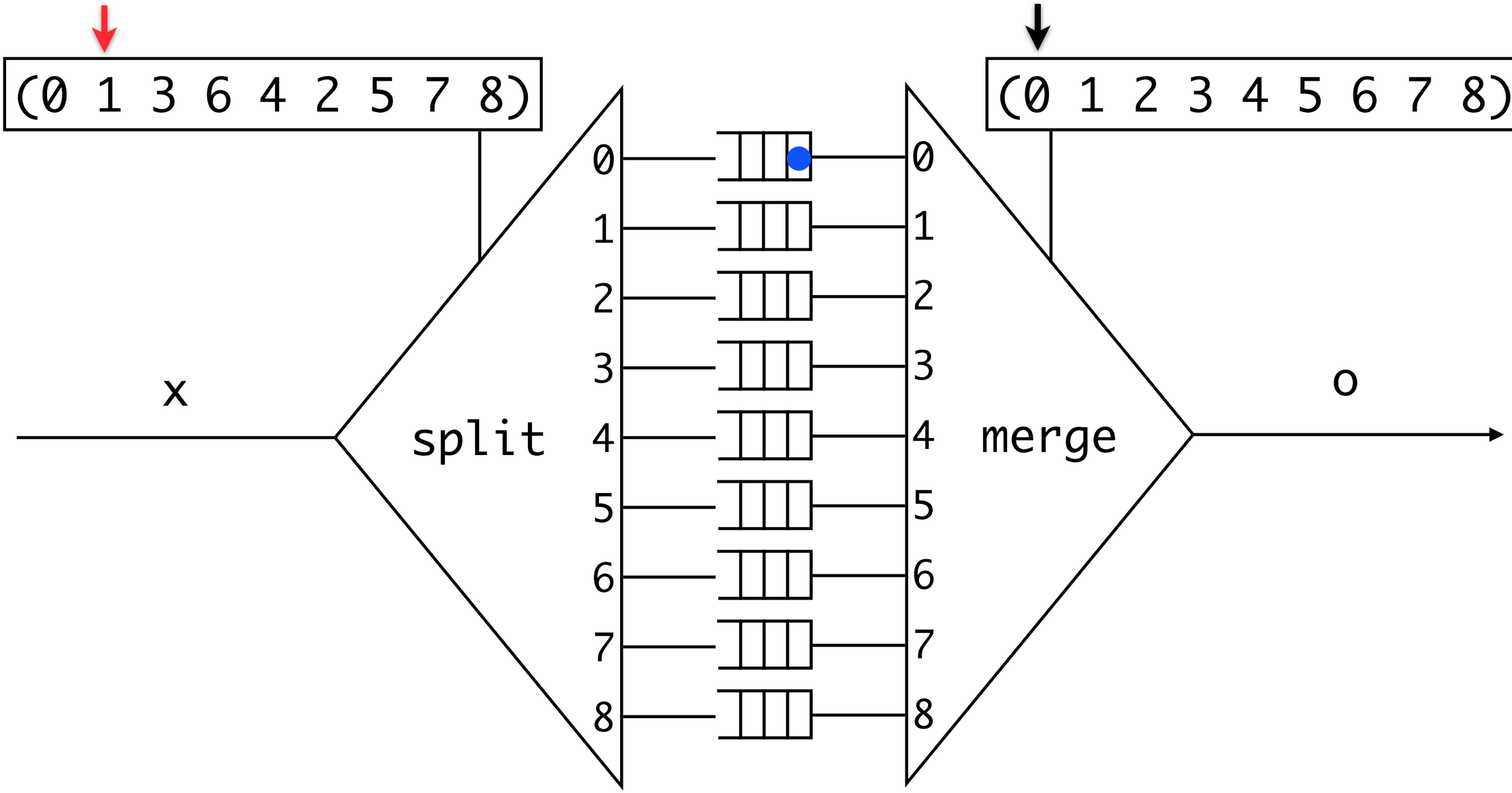
```
$ asc -i zigzag.as
val zigzag
  : 'x -> ‘x
  :: 'a -> 'a on 0^3(1)
```

```
                    ...
                  | 8 -> buffer x8
              end
```

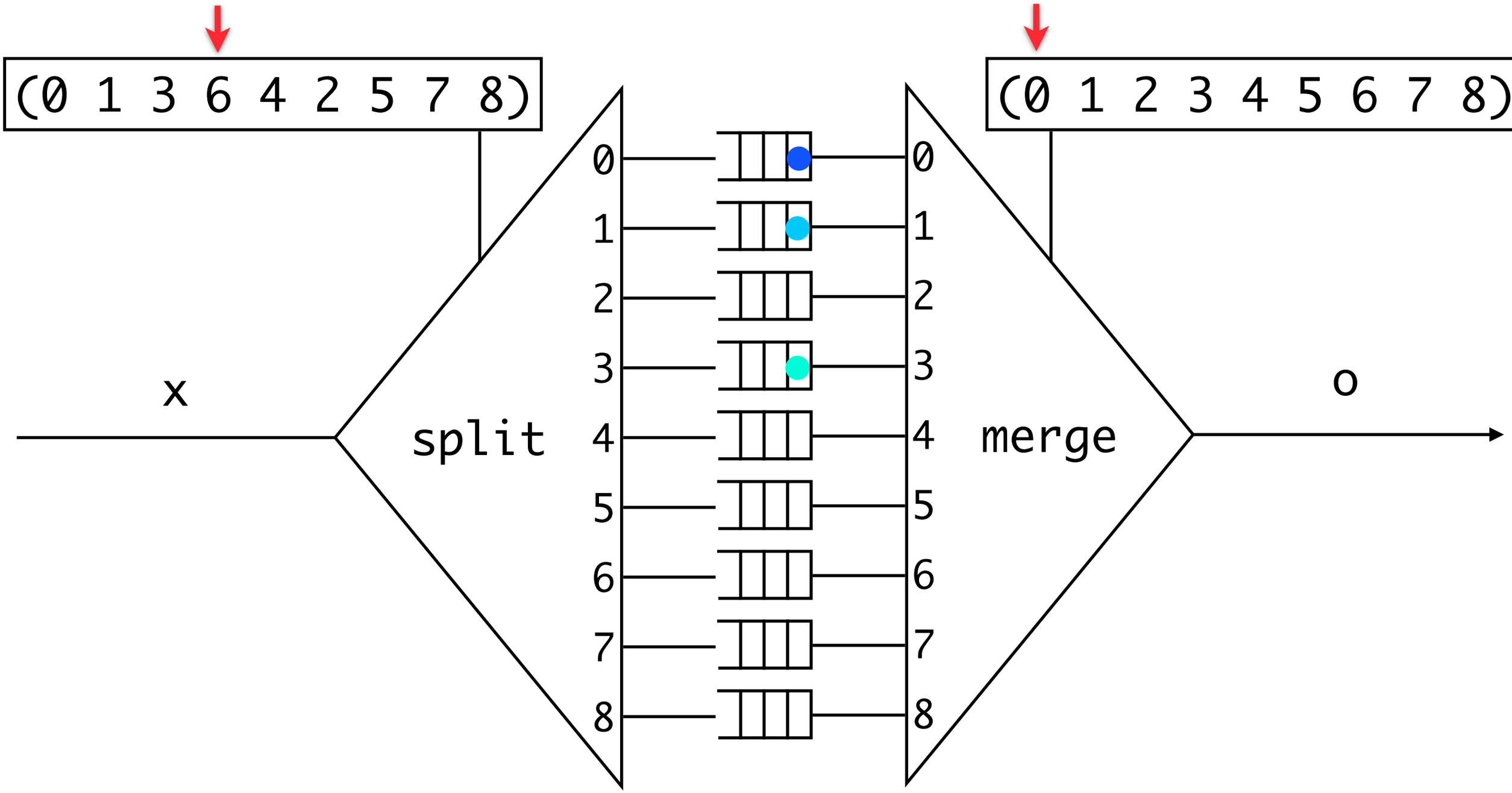
Parcours Zig-Zag JPEG, dataflow scalaire



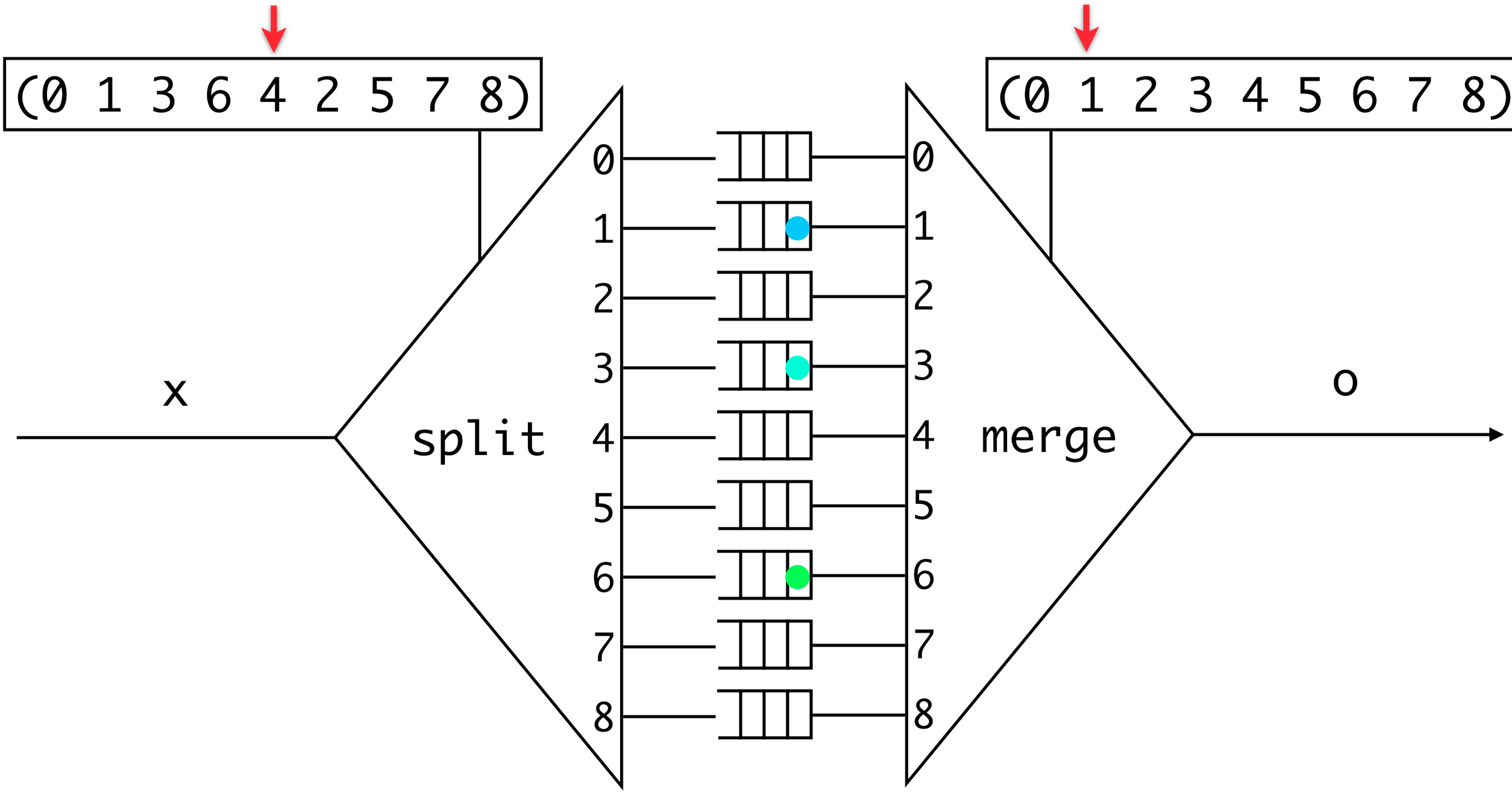
Parcours Zig-Zag JPEG, dataflow scalaire



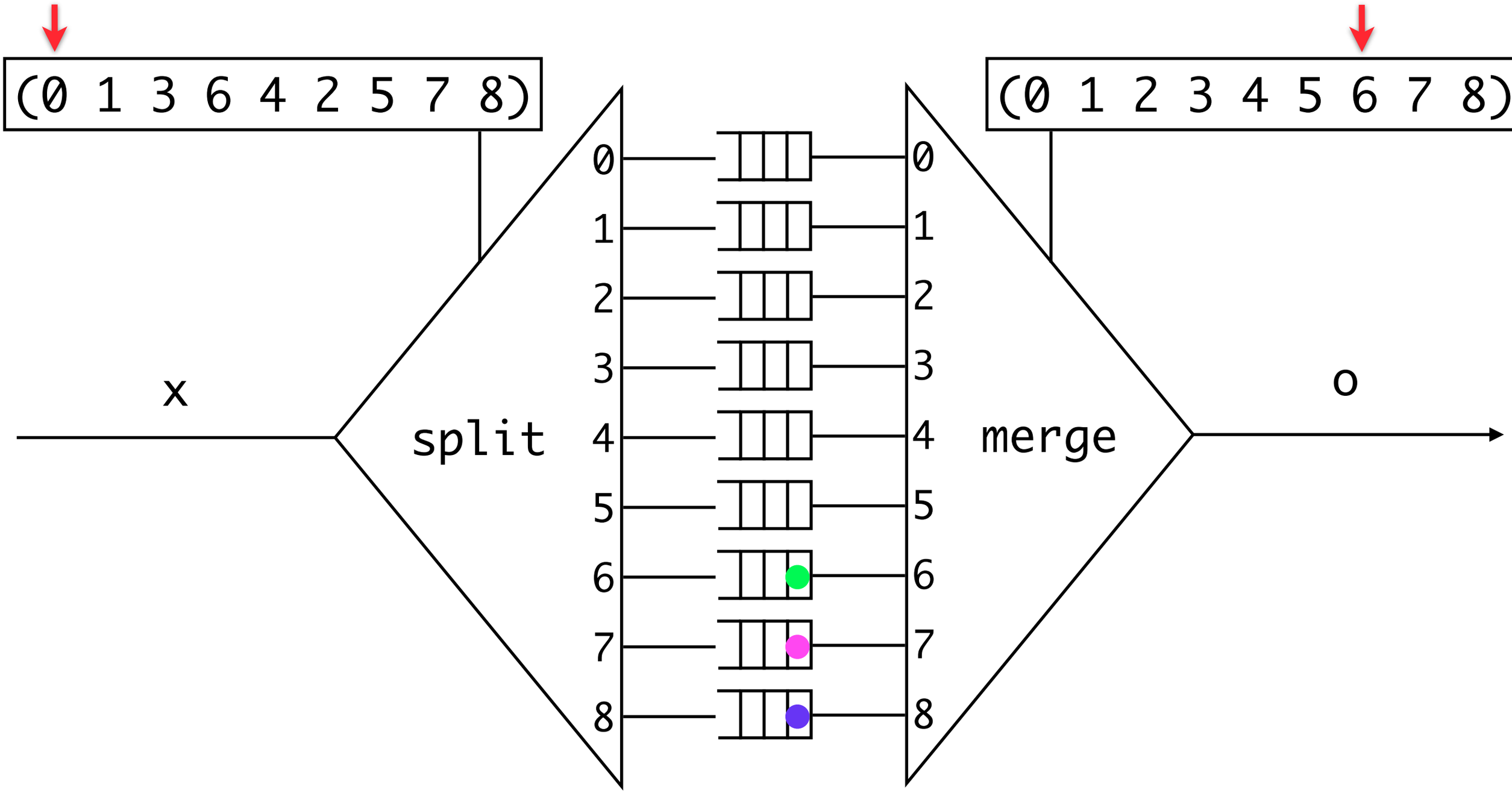
Parcours Zig-Zag JPEG, dataflow scalaire



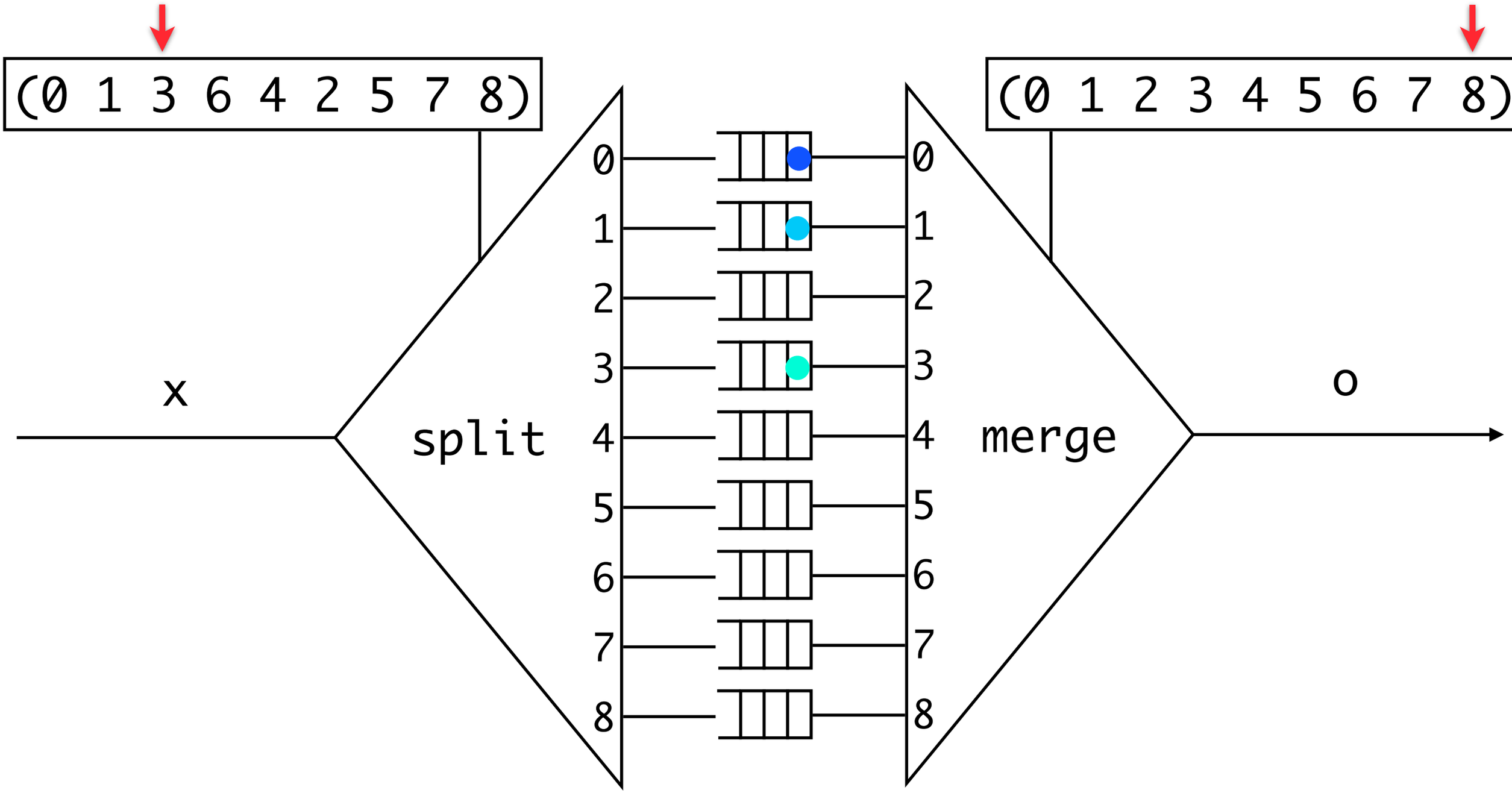
Parcours Zig-Zag JPEG, dataflow scalaire



Parcours Zig-Zag JPEG, dataflow scalaire



Parcours Zig-Zag JPEG, dataflow scalaire



Parcours Zig-Zag JPEG en AcidS

```
let node zigzag x = o where
    rec (x0, x1, x2, x3, x4, x5, x6, x7, x8)
```

```
$ asc -i zigzag.as
val zigzag
  : 'x -> ‘x
  :: 'a -> 'a on 0^3(1)
```

```
                    ...
                  | 8 -> buffer x8
              end
```

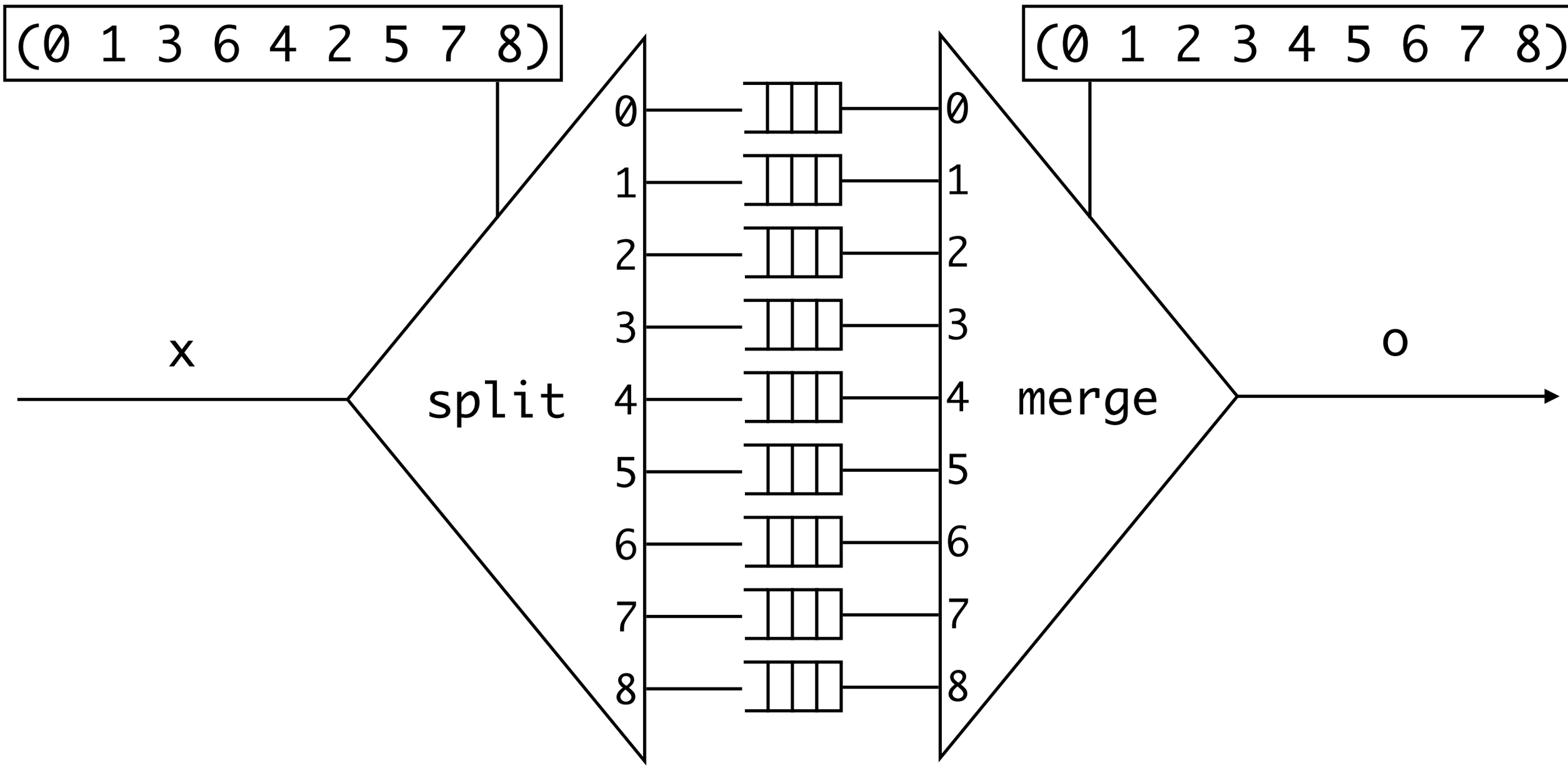
Parcours Zig-Zag JPEG en AcidS

```
let node zigzag x = o where
        (x0, x1, x2, x3, x4, x5, x6, x7, x8)
```

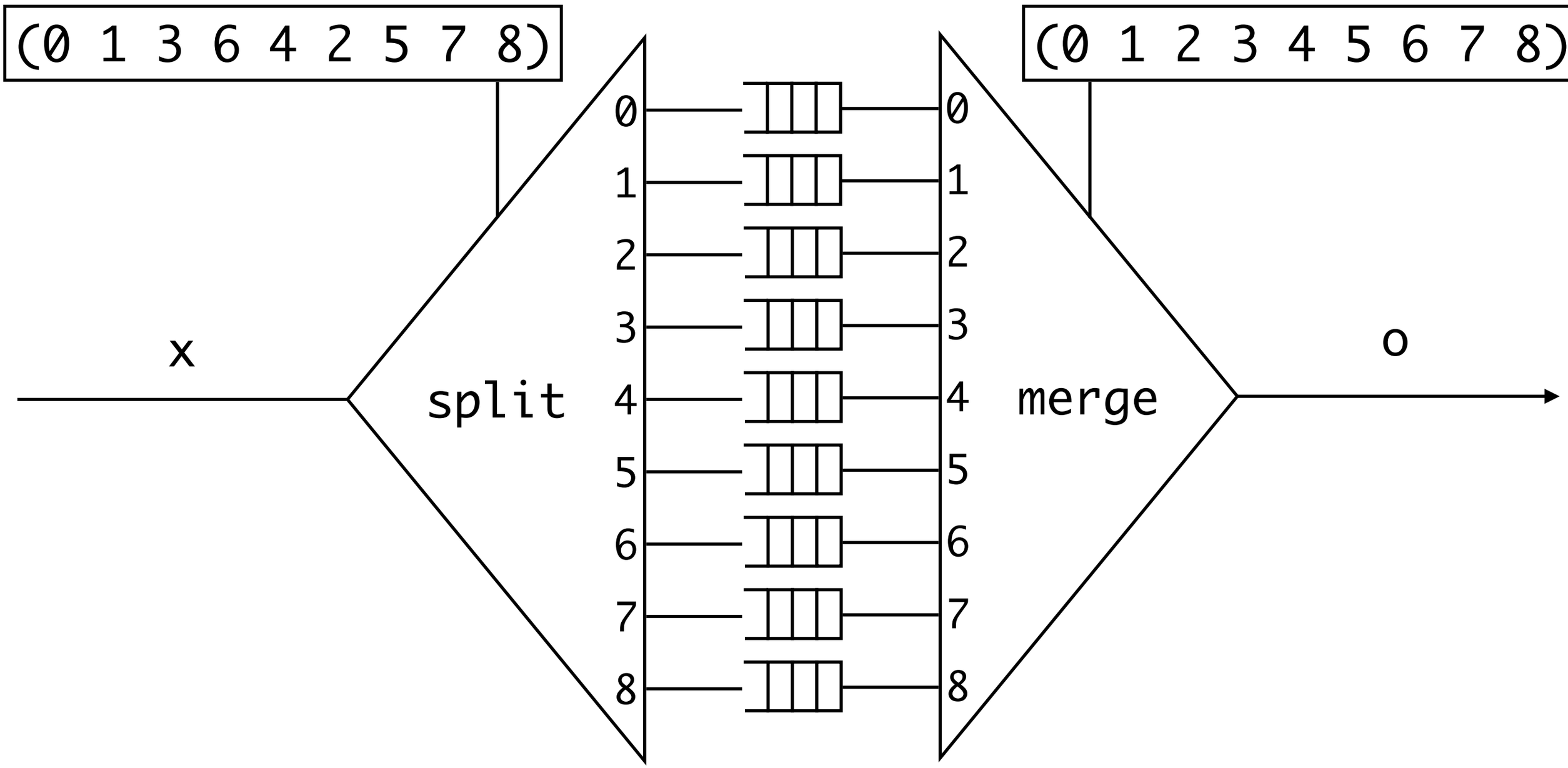
```
$ asc -i zigzag.as -max_burst 9
val zigzag
: 'x -> 'x
:: 'a on (9) -> 'a on (9)
```

```
                buffer x1
            ...
          | 8 -> buffer x8
    end
```

Parcours Zig-Zag JPEG, dataflow rafales



Parcours Zig-Zag JPEG, dataflow rafales



Parcours Zig-Zag JPEG, variation (ASAP)

```
$ asc -i zigzag.as -max_burst 3
val zigzag
 : 'x -> 'x
 :: ???
```

Parcours Zig-Zag JPEG, variation (ASAP)

```
$ asc -i zigzag.as -max_burst 3
val zigzag
 : 'x -> 'x
 :: ???
```

Demandons au compilateur...

Parcours Zig-Zag JPEG, variation (ASAP)

```
$ asc -i zigzag.as -max_burst 3
val zigzag
 : 'x -> 'x
 ::'a on (3) -> 'a on 0(3)
```

Demandons au compilateur...

Parcours Zig-Zag JPEG, variation (ASAP)

```
$ asc -i zigzag.as -max_burst 3
val zigzag
 : 'x -> 'x
 ::'a on (3) -> 'a on 0(3)
```

Demandons au compilateur...

Parcours Zig-Zag JPEG, variation (ASAP)

```
$ asc -i zigzag.as -max_burst 3
val zigzag
 : 'x -> 'x
 ::'a on (3) -> 'a on 0(3)
```

Demandons au compilateur...

Parcours Zig-Zag JPEG, variation (ASAP)

```
$ asc -i zigzag.as -max_burst 3
val zigzag
 : 'x -> 'x
 ::'a on (3) -> 'a on 0(3)
```

Demandons au compilateur...

Plan

Introduction

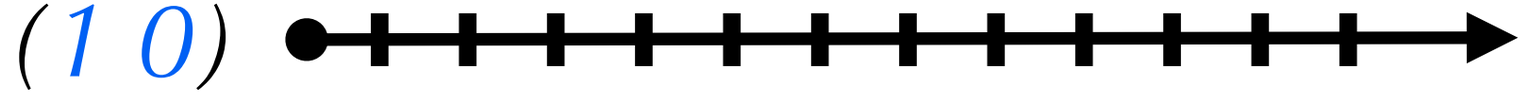
Horloges Entières

Le langage

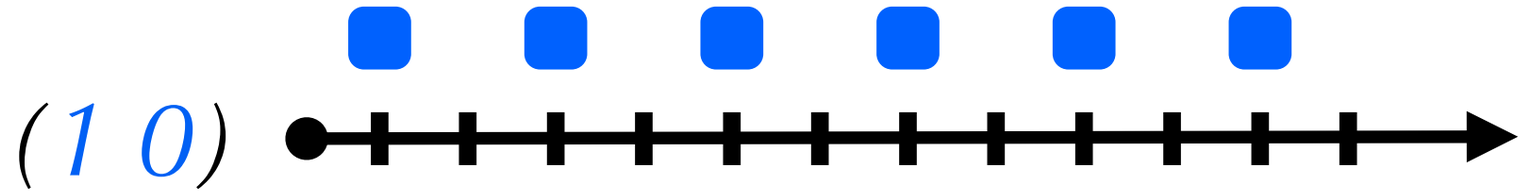
Un compilateur expérimental

Conclusion

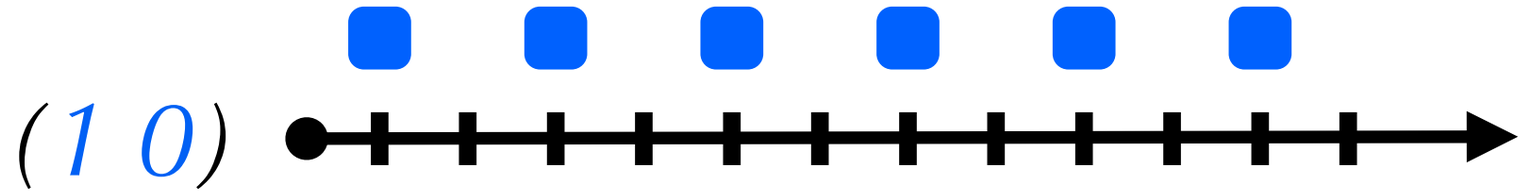
Horloges binaires

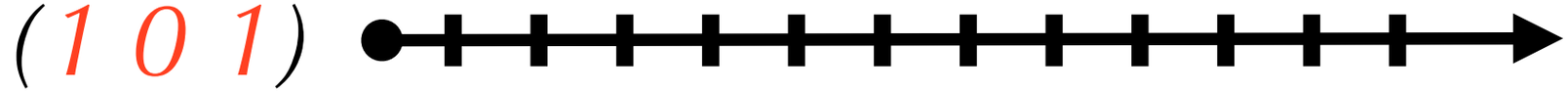


Horloges binaires

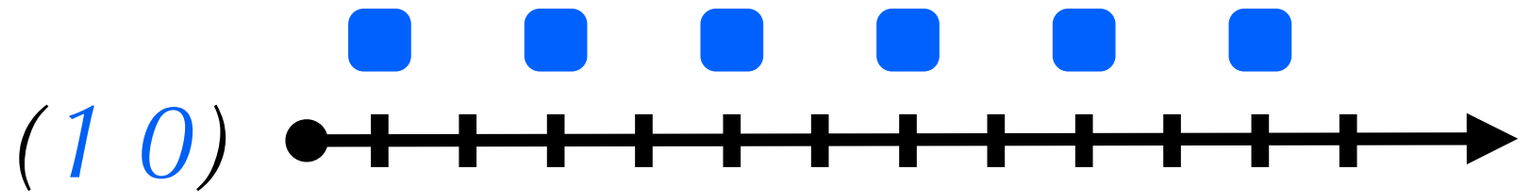


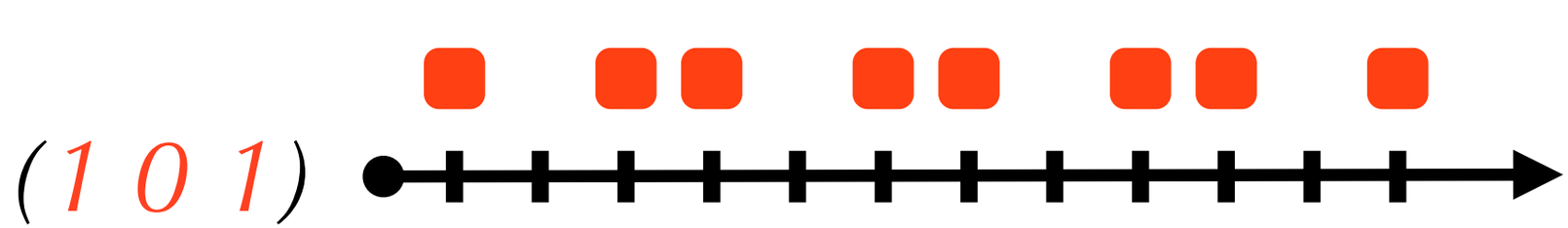
Horloges binaires



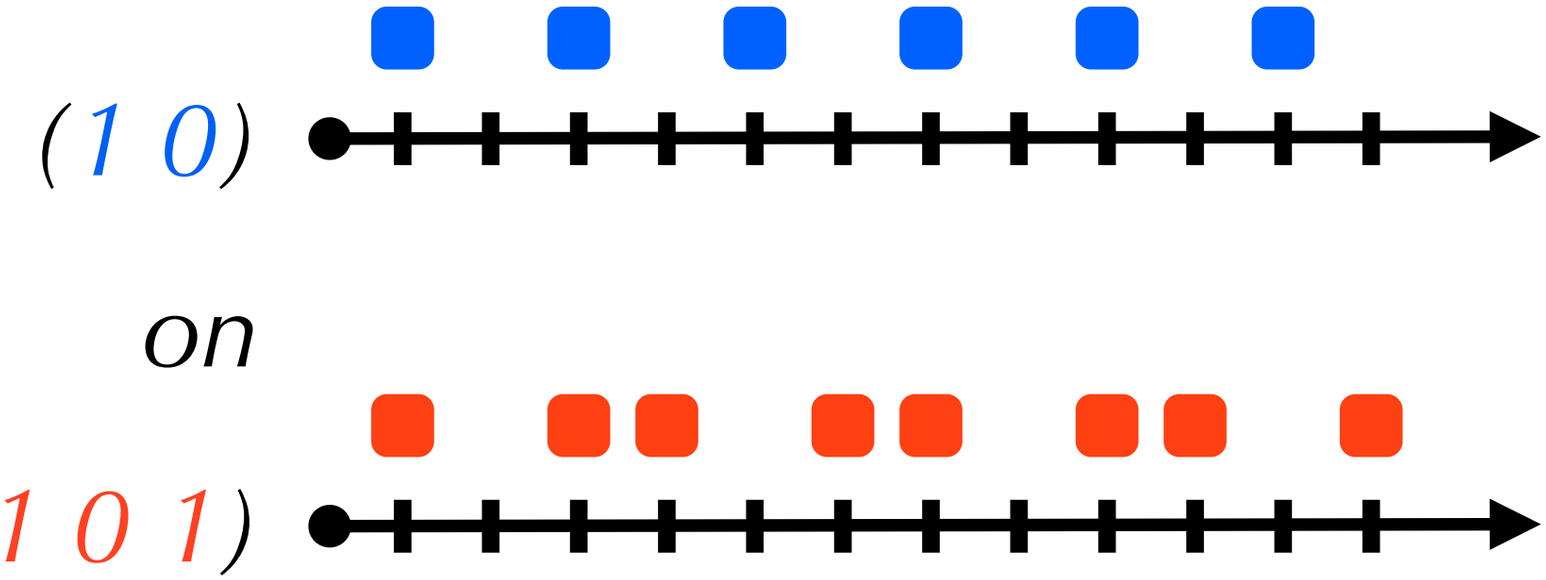


Horloges binaires

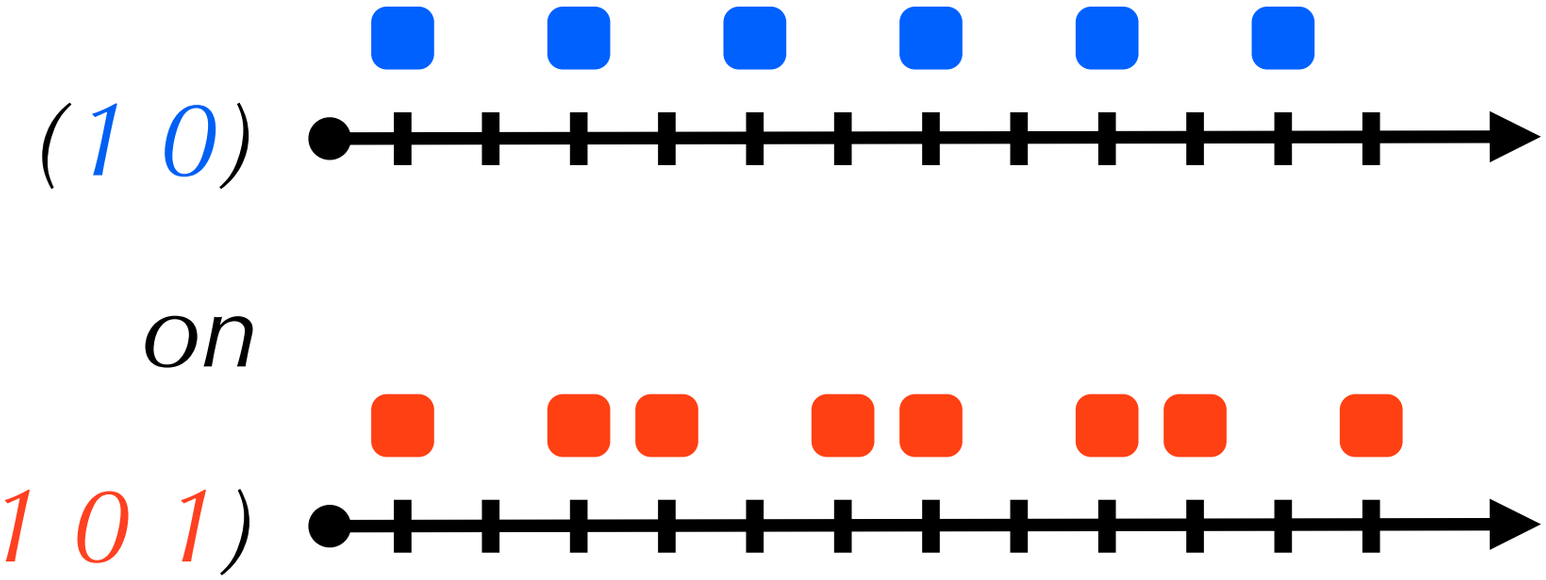




Horloges binaires

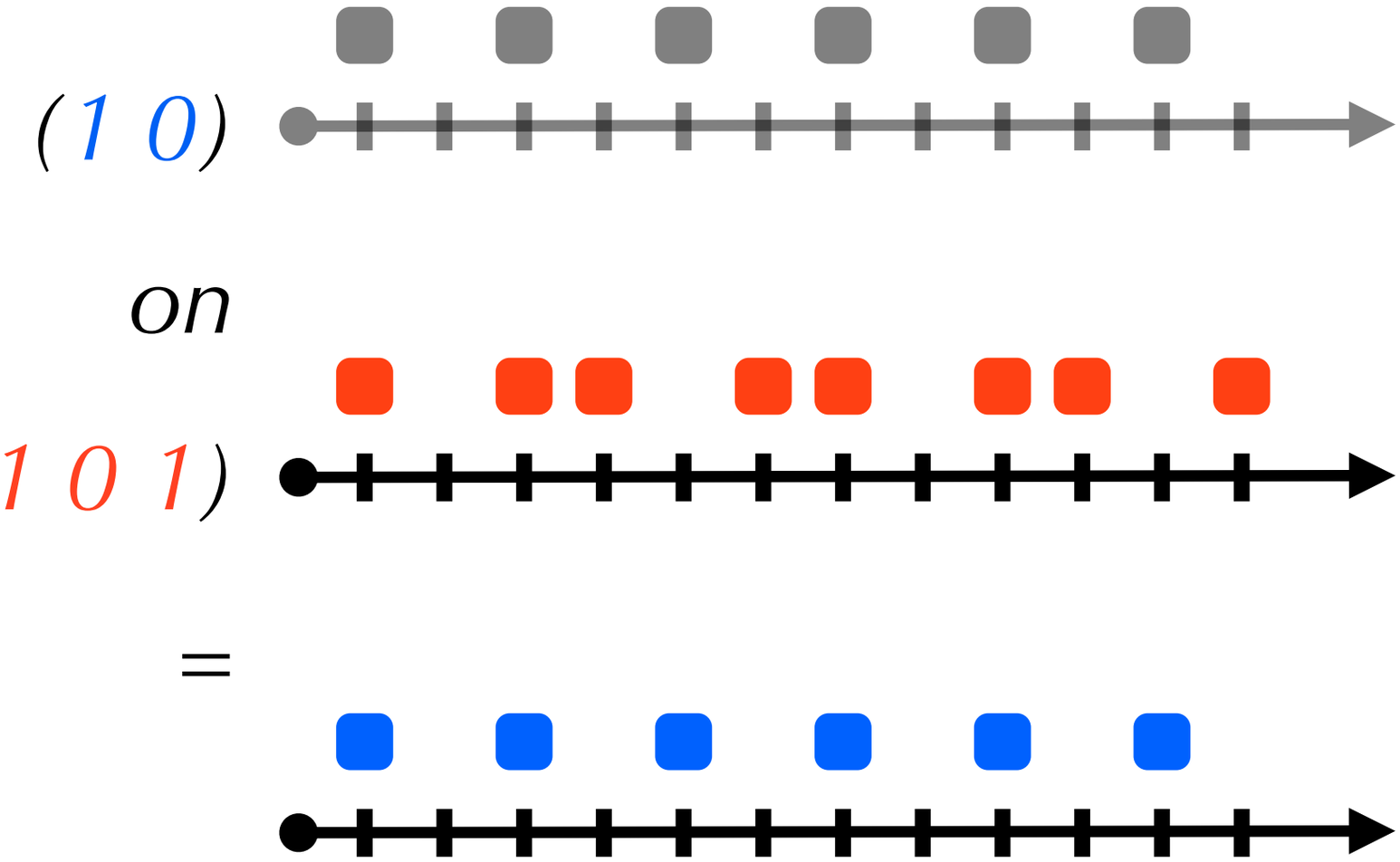


Horloges binaires

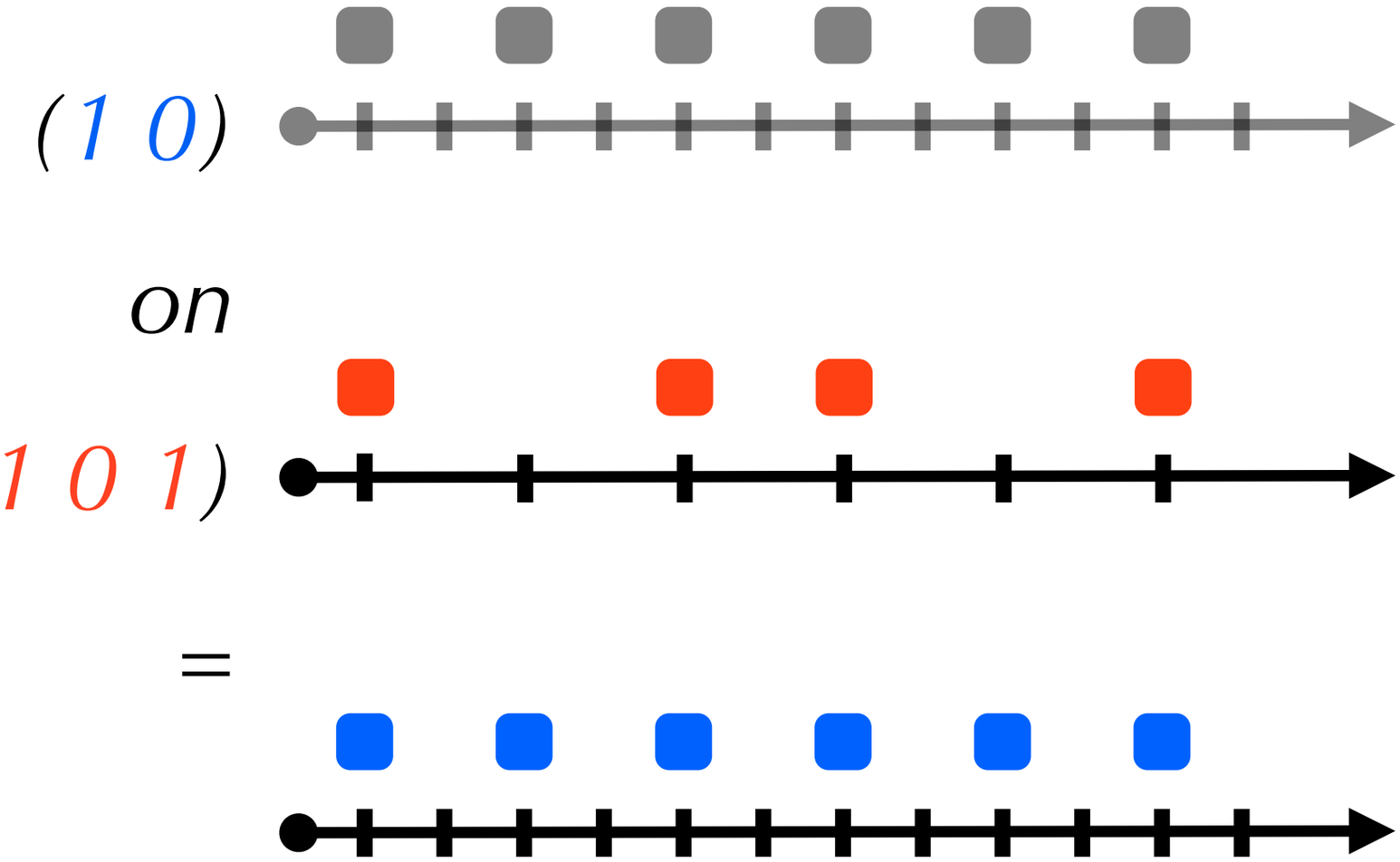


=

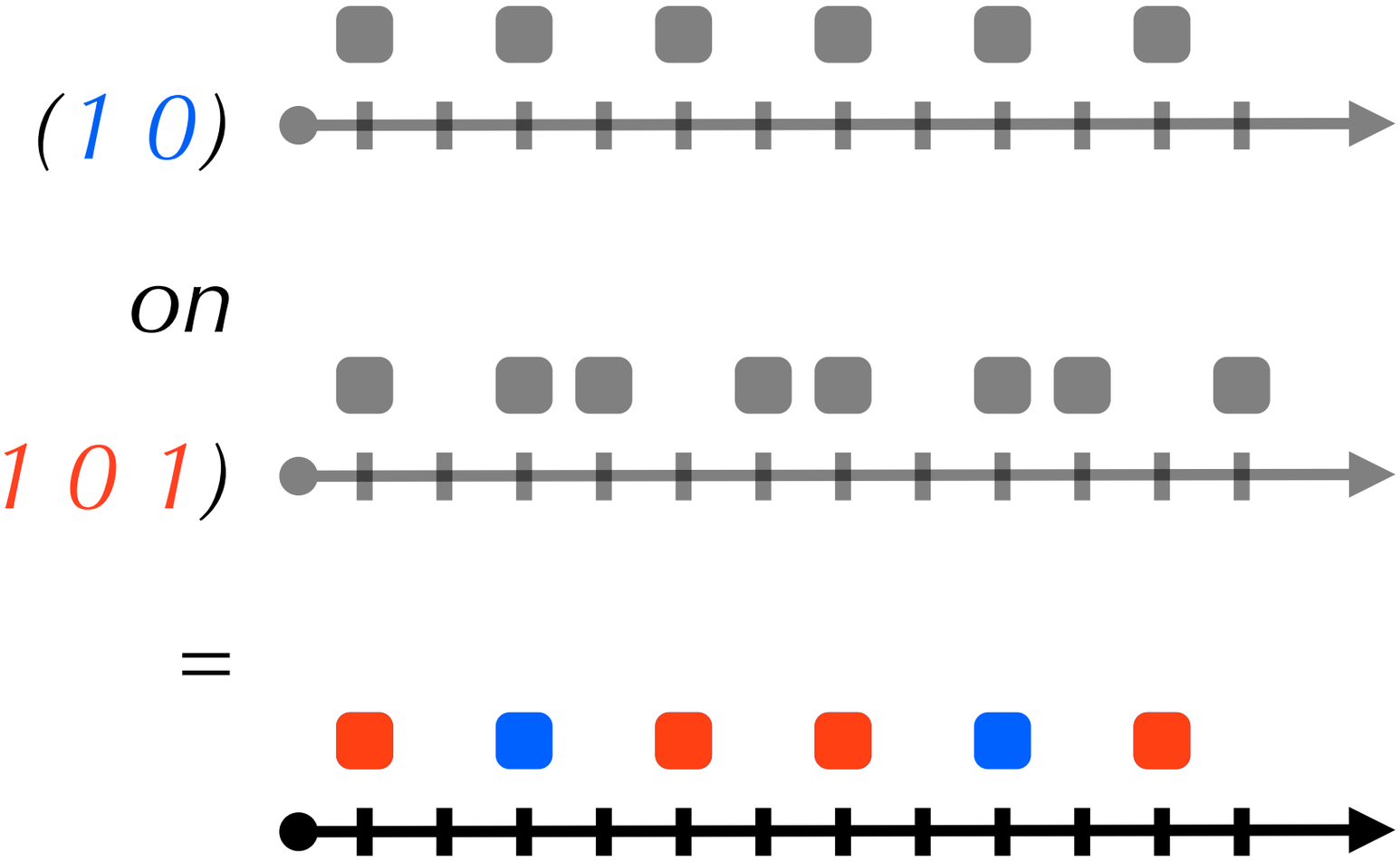
Horloges binaires



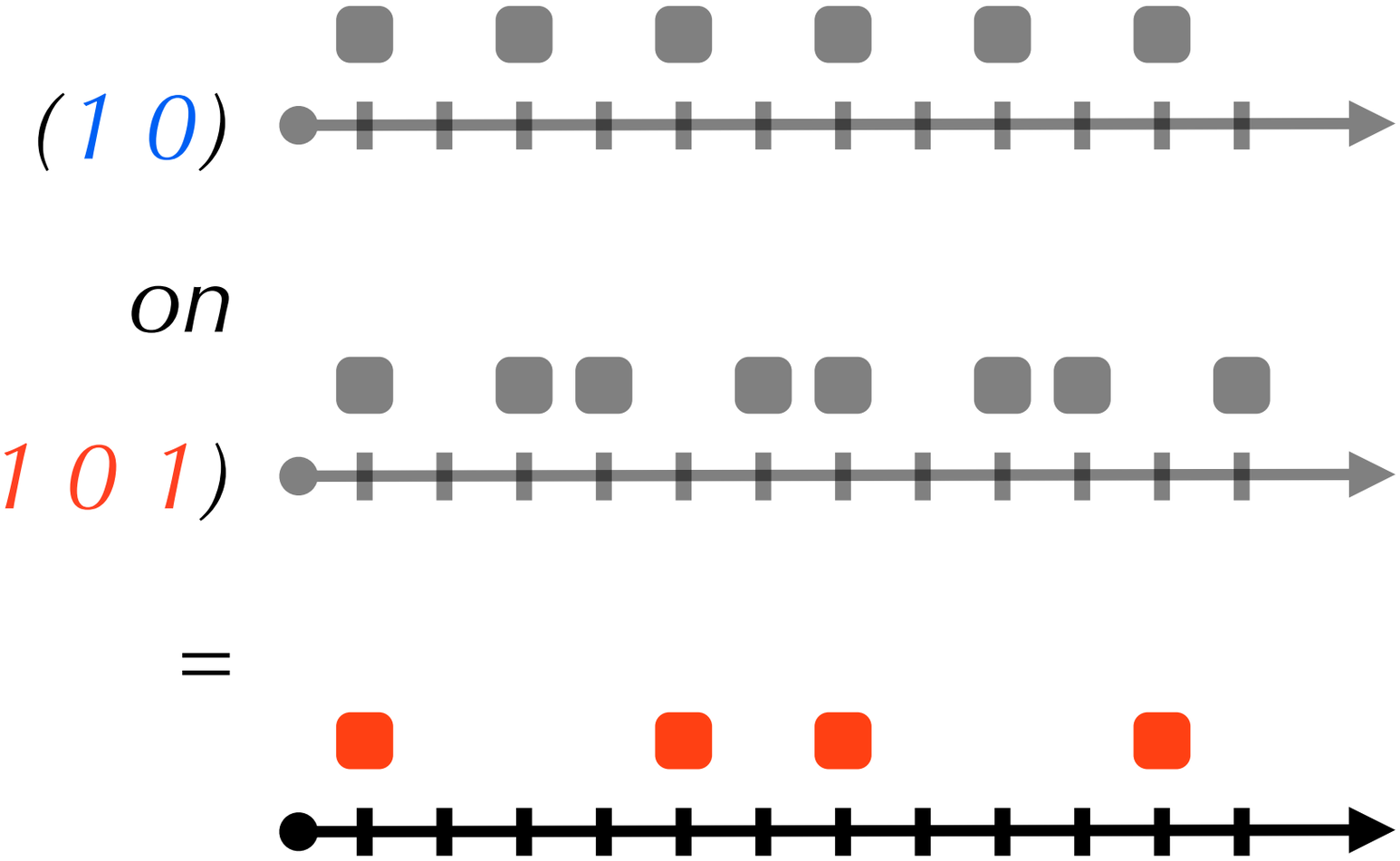
Horloges binaires



Horloges binaires

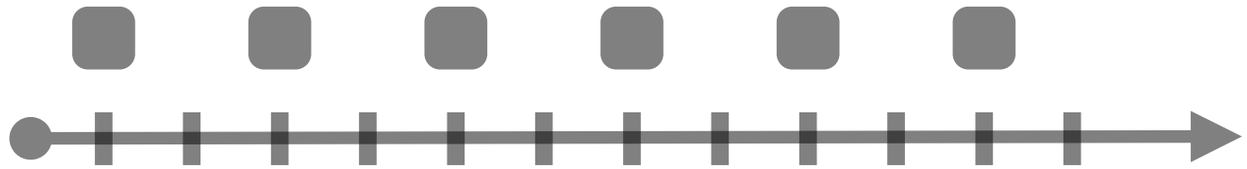


Horloges binaires



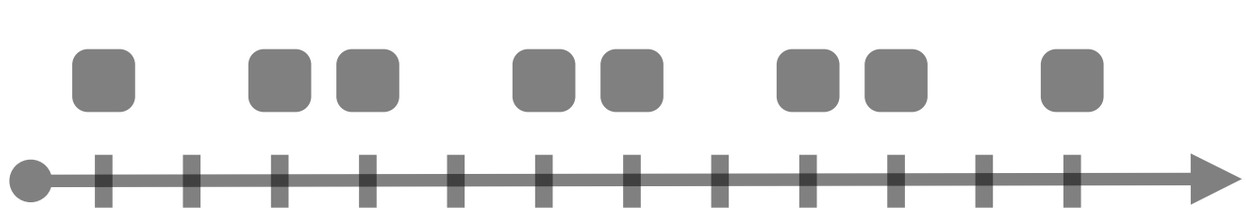
Horloges binaires

(1 0)



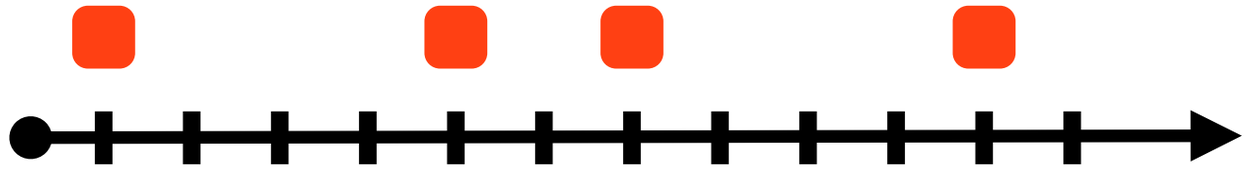
on

(1 0 1)

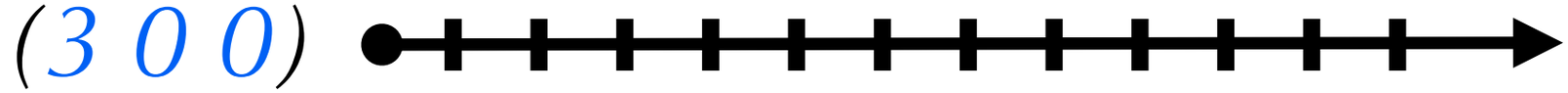


=

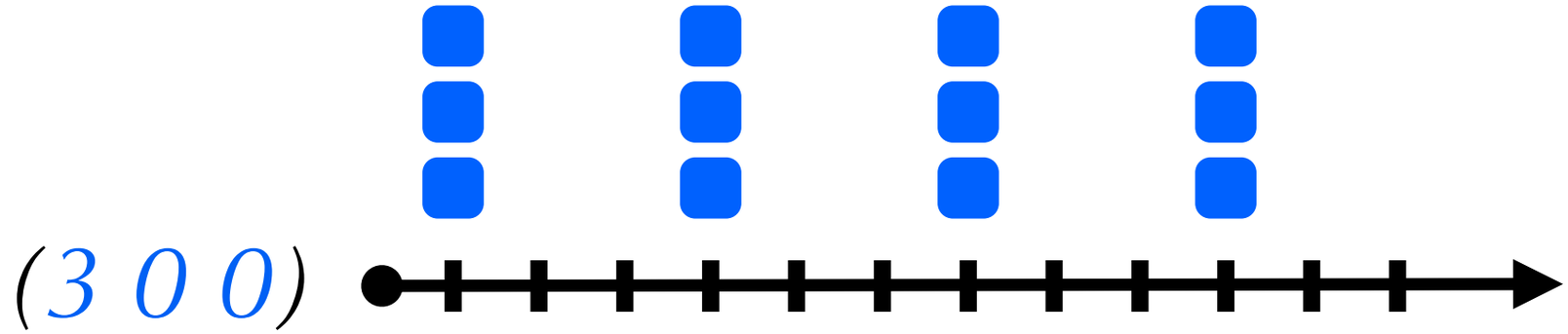
(1 0 0 0 1 0)



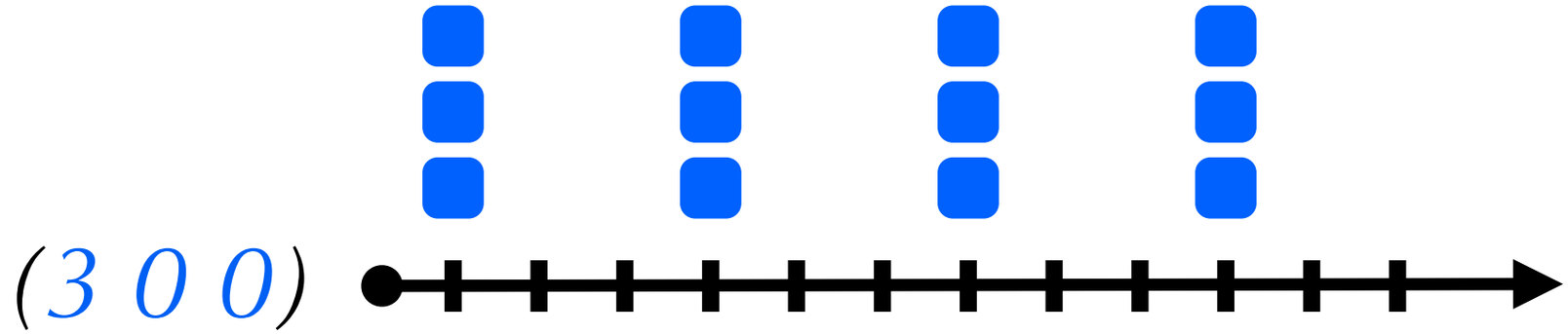
Horloges entières

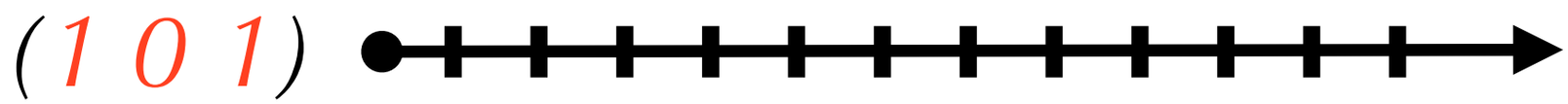


Horloges entières

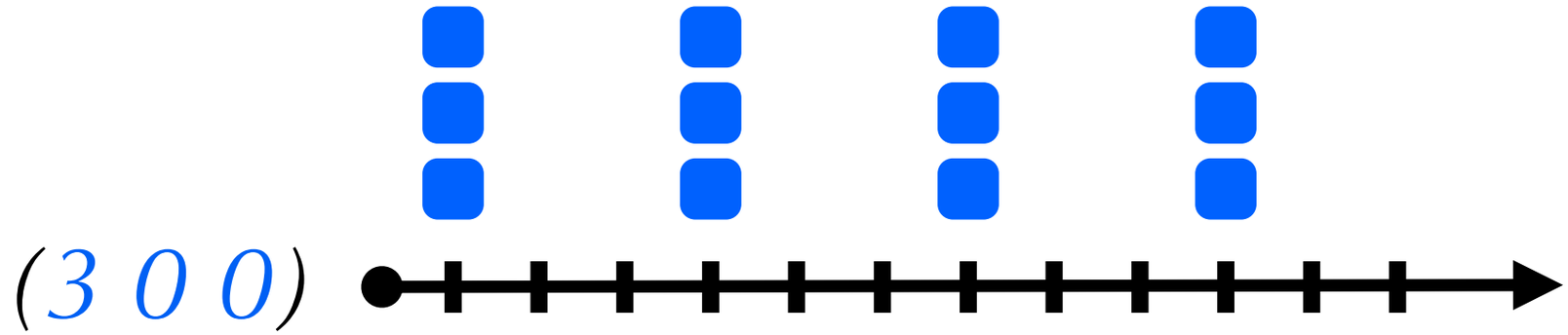


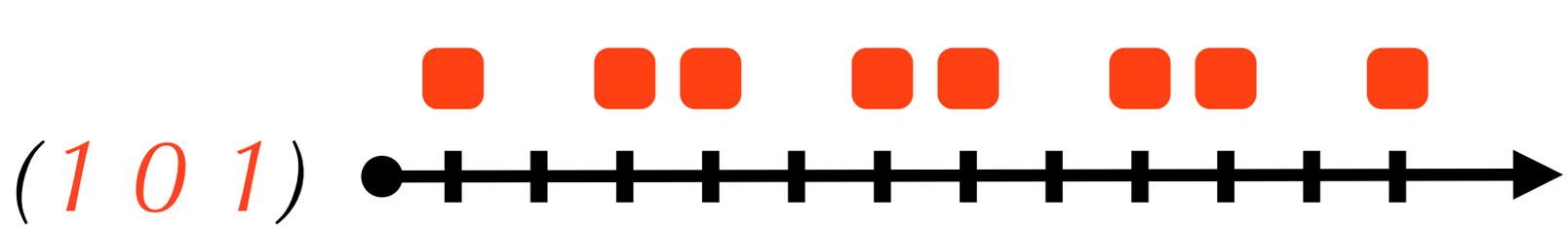
Horloges entières



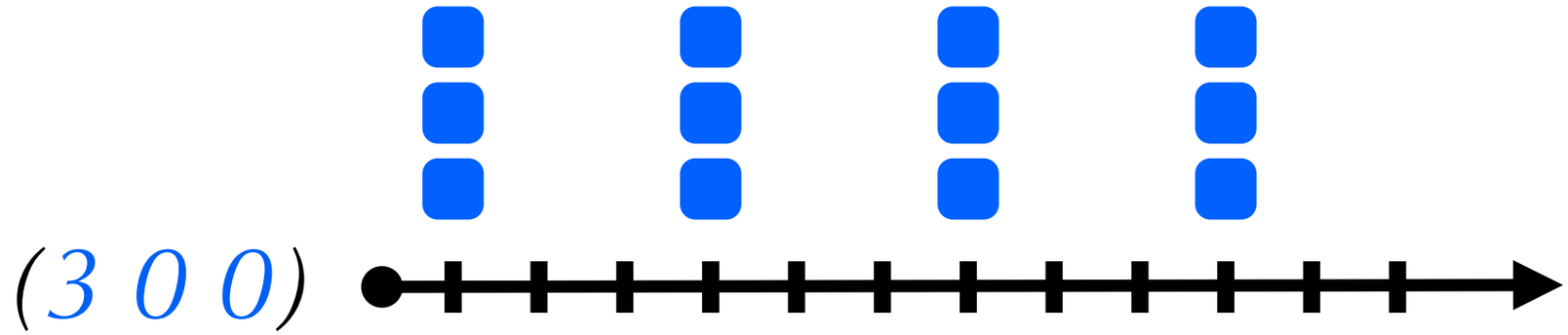


Horloges entières

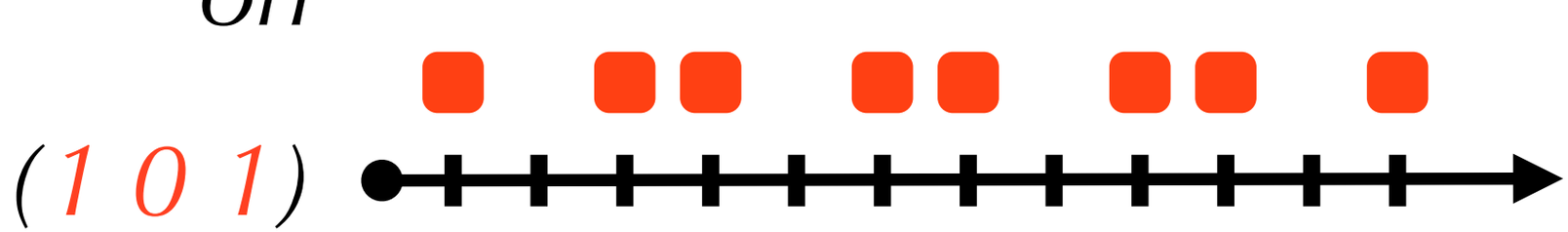




Horloges entières

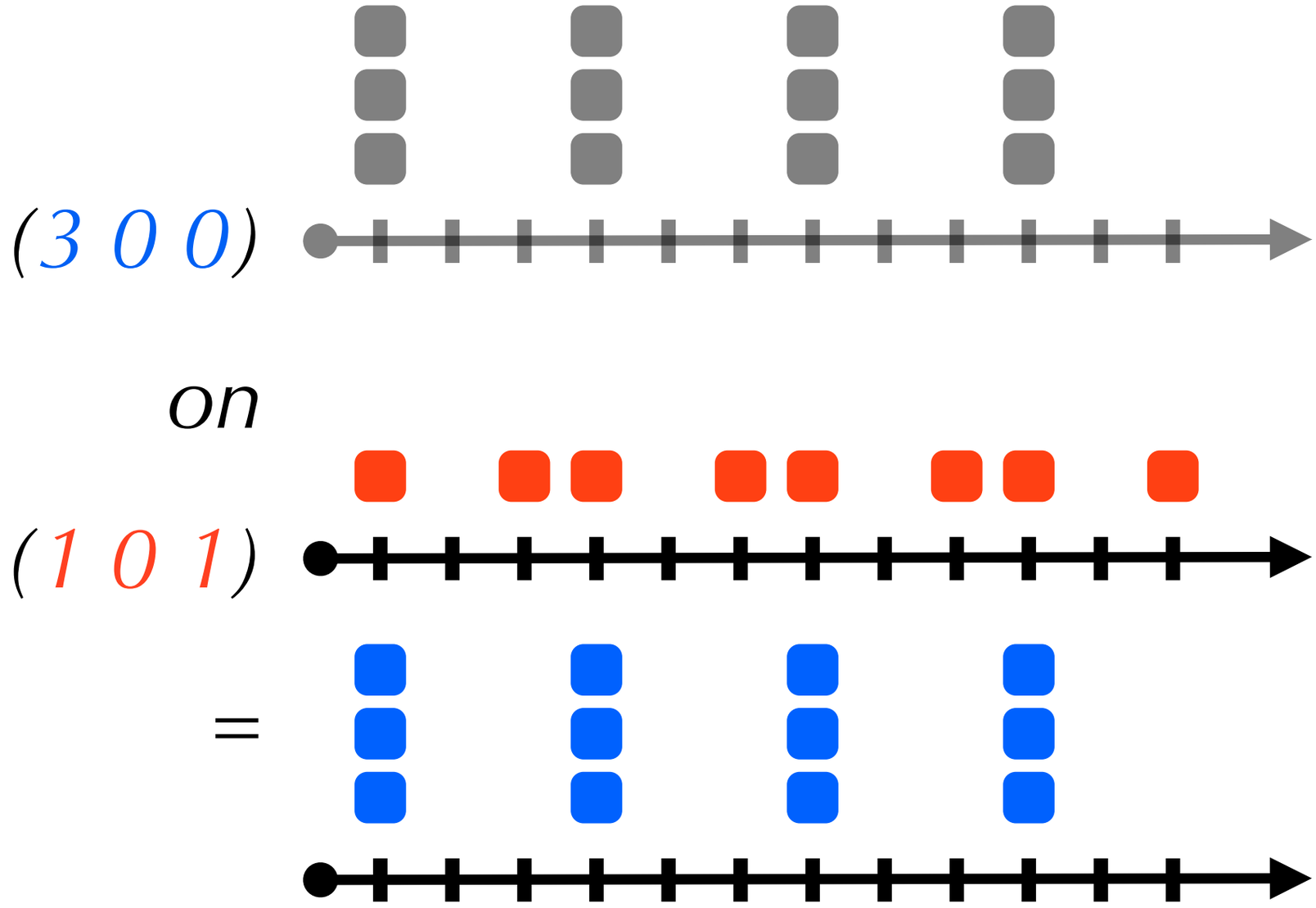


on

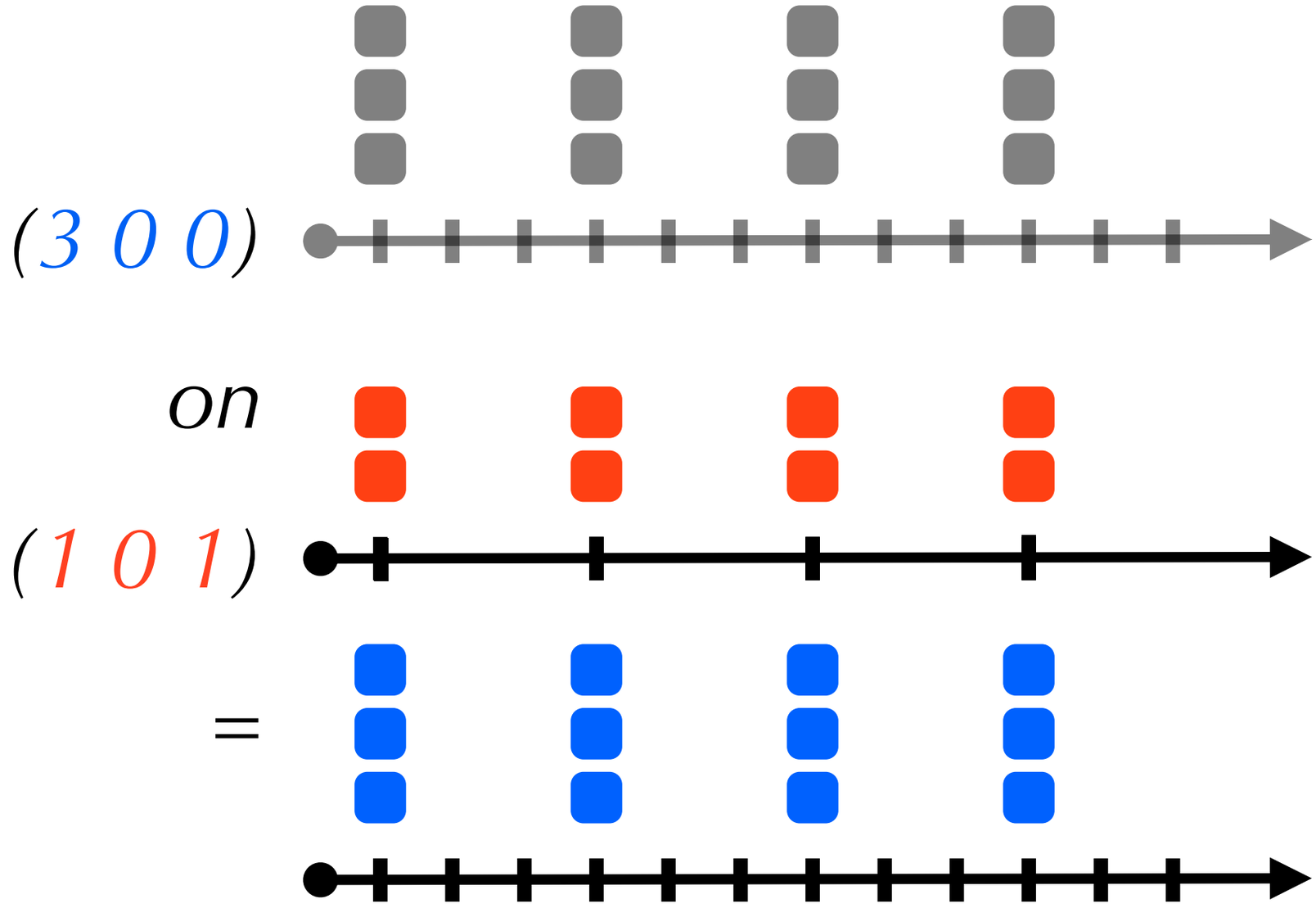


=

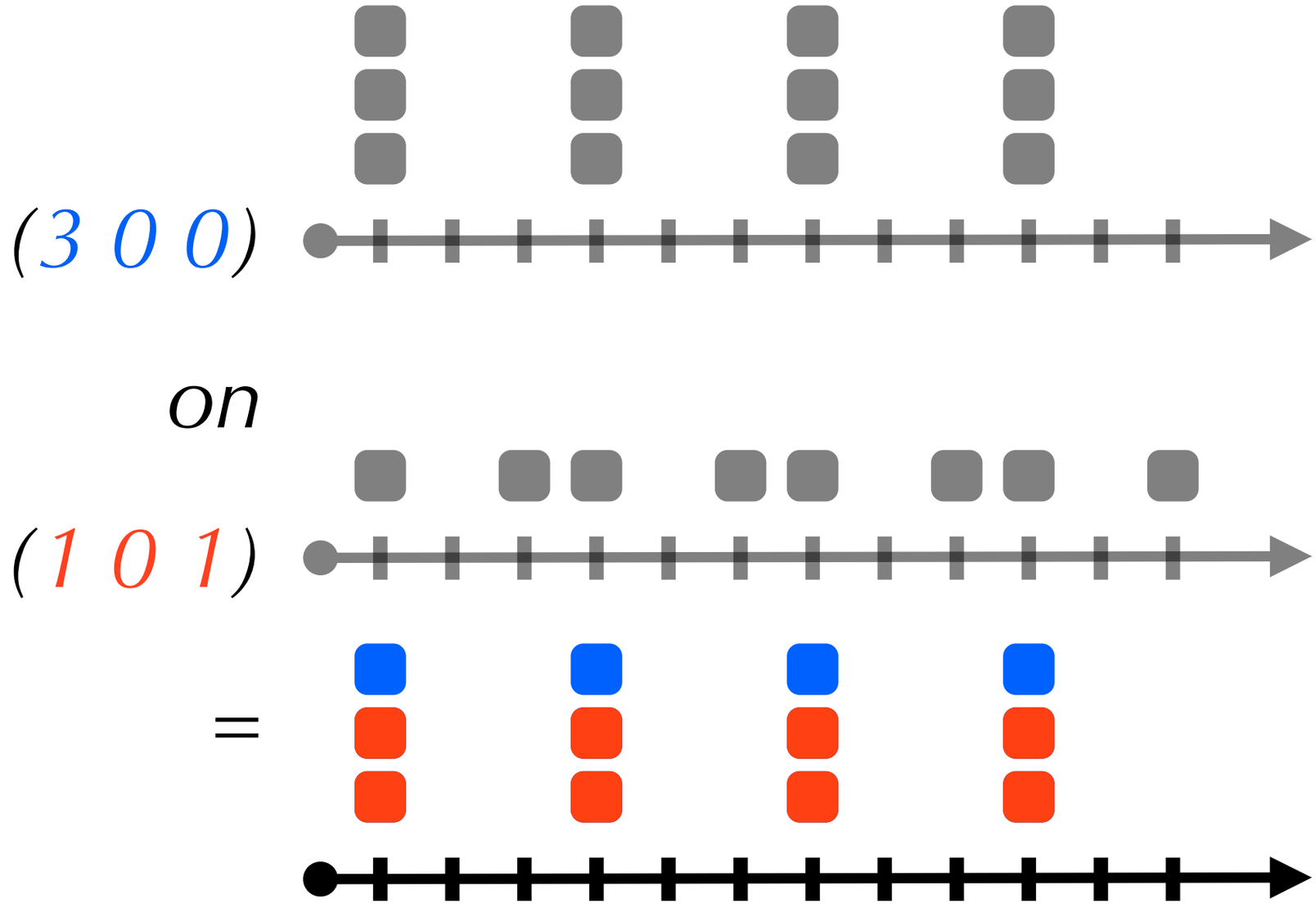
Horloges entières



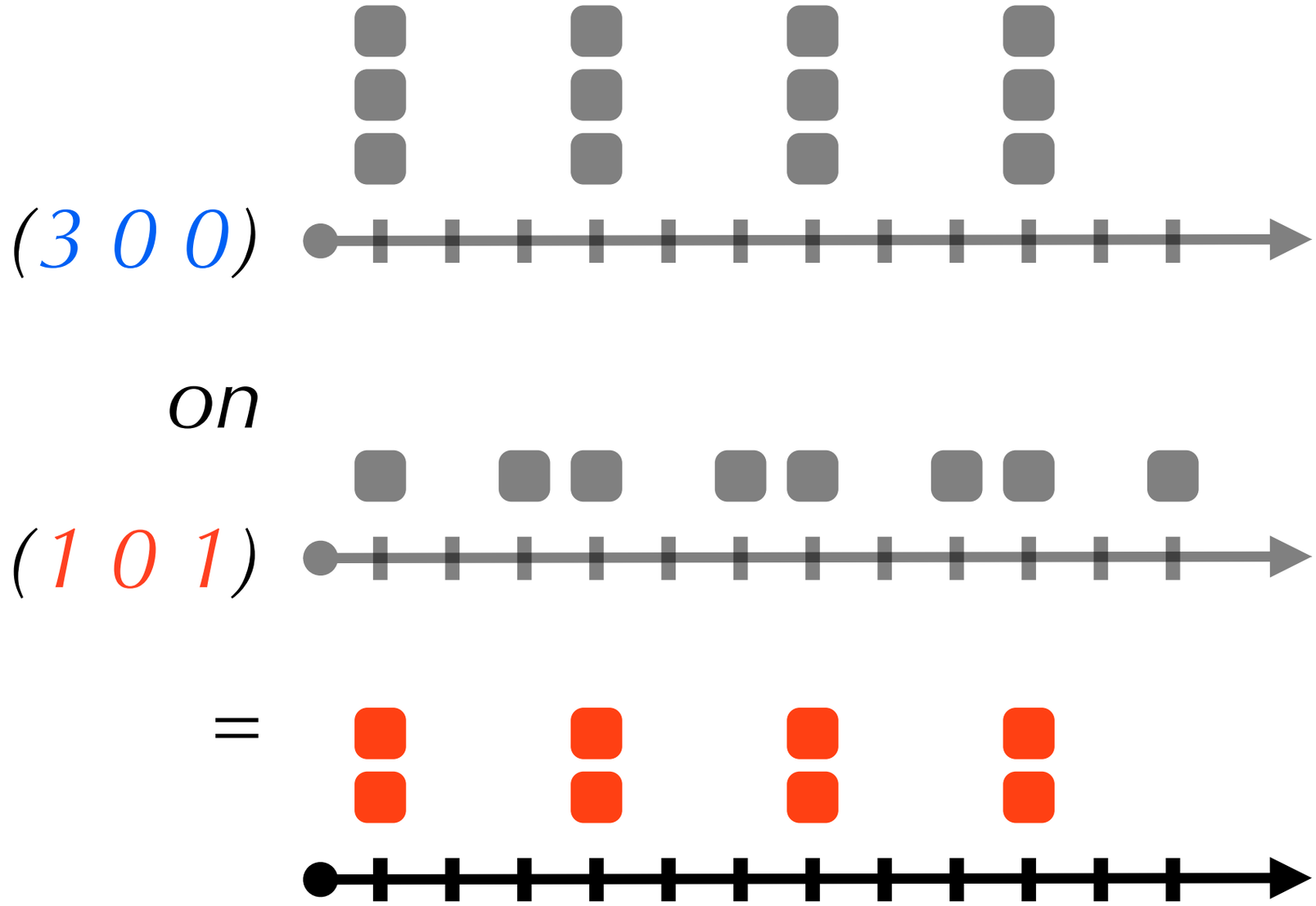
Horloges entières



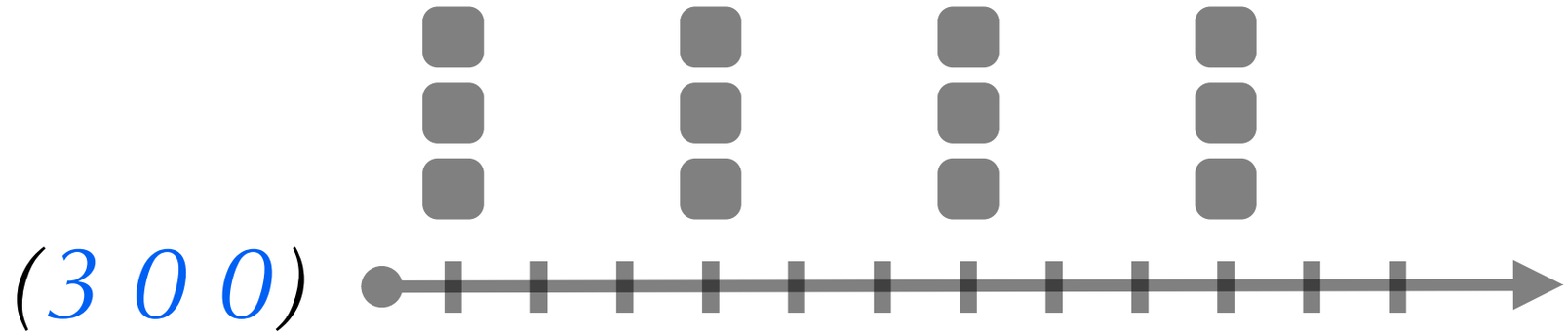
Horloges entières



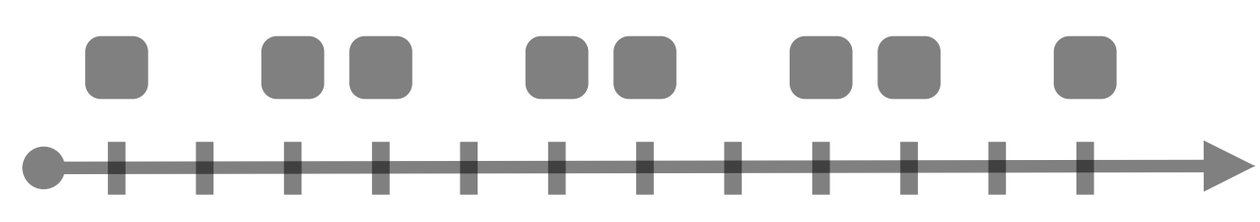
Horloges entières



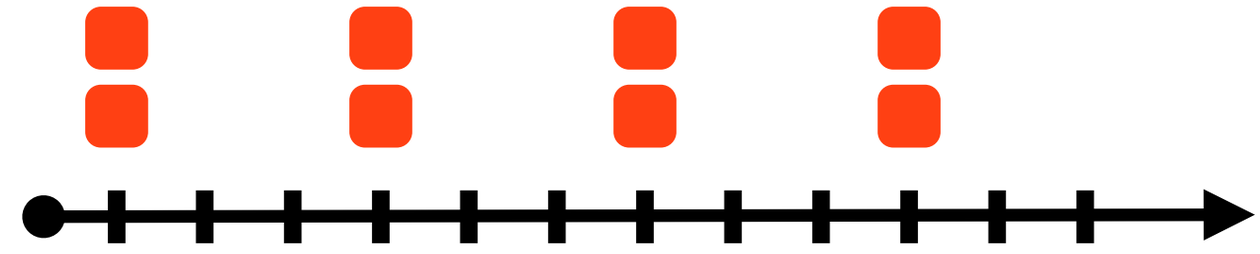
Horloges entières



on



=



Structure algébrique des horloges

Definition:

$$w \in \mathbb{N}^\omega$$

Composition:

$$(n.w)\ on\ (k_1 \ldots k_n.w') = (\sum_{1 \leq i \leq n} k_i).(w\ on\ w')$$

Buffers:

$$w <: w' \ = \ \forall i.\ 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq S$$

with $\mathcal{O}_w(i) = \sum_{1 \leq j \leq i} w[j]$

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  c1 on (1 0 0)    <:  c2 on (1 0)
   c1 on (0 1 0)    <:  c2 on (1 0)
c1                  <:  c3 on (0 2)     }
```

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  c1 on (1 0 0)    <:  c2 on (1 0)
   c1 on (0 1 0)    <:  c2 on (1 0)
   c1               <:  c3 on (0 2)      }
```

Une solution possible :

```
{  c1               =   (1)
   c2               =   (0 1)
   c3               =   (1)              }
```

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  (1) on (1 0 0)   <:  (0 1) on (1 0)
   (1) on (0 1 0)   <:  (0 1) on (1 0)
   (1)              <:  (1) on (0 2)    }
```

Une solution possible :

```
{  c1   =  (1)
   c2   =  (0 1)
   c3   =  (1)                         }
```

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  (1 0 0)          <:  (0 1 0)
   (0 1 0)          <:  (0 1 0)
   (1)              <:  (0 2)          }
```

Une solution possible :

```
{  c1               =   (1)
   c2               =   (0 1)
   c3               =   (1)            }
```

Inéquations d'horloges

Trouver des horloge[illegible] Taille un [illegible]es que...

```
{  (1 0 0)        <:  (0 1 0)
   (0 1 0)        <:  (0 1 0)
   (1)            <:  (0 2)          }
```

Une solution possible :

```
{  c1             =   (1)
   c2             =   (0 1)
   c3             =   (1)            }
```

Inéquations d'horloges

Trouver des horloges telles que...

Taille un

Taille zéro

```
{  (1 0 0)      <:  (0 1 0)
   (0 1 0)      <:  (0 1 0)
   (1)          <:  (0 2)          }
```

Une solution possible :

```
{  c1           =   (1)
   c2           =   (0 1)
   c3           =   (1)            }
```

Inéquations d'horloges

Trouver des horloges telles que...

Taille un

Taille zéro

Taille un

$$\left\{ \begin{array}{ll} (1\ 0\ 0) & <: \quad (0\ 1\ 0) \\ (0\ 1) & <: \quad (0\ 1\ 0) \\ (1) & <: \quad (0\ 2) \end{array} \right.$$

Une solution possible :

$$\left\{ \begin{array}{lll} c1 & = & (1) \\ c2 & = & (0\ 1) \\ c3 & = & (1) \end{array} \right.$$

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  c1 on (1 0 0)    <:  c2 on (1 0)
   c1 on (0 1 0)    <:  c2 on (1 0)
   c1               <:  c3 on (0 2)      }
```

Une autre solution :

```
{  c1               =   (3 0 0)
   c2               =   (2 0 0)
   c3               =   (1)              }
```

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  (3 0 0) on (1 0 0)   <:  (2 0 0) on (1 0)
   (3 0 0) on (0 1 0)   <:  (2 0 0) on (1 0)
   (3 0 0)              <:  (1) on (0 2)      }
```

Une autre solution :

```
{  c1                   =   (3 0 0)
   c2                   =   (2 0 0)
   c3                   =   (1)               }
```

Inéquations d'horloges

Trouver des horloges c1, c2 et c3 telles que...

```
{  (1 0 0)          <:  (1 0 0)
   (1 0 0)          <:  (1 0 0)
   (3 0 0)          <:  (0 2)          }
```

Une autre solution :

```
{  c1               =   (3 0 0)
   c2               =   (2 0 0)
   c3               =   (1)            }
```

Inéquations d'horloges

Trouver des horloges telles que...

Taille zéro

```
{  (1 0 0)          <:  (1 0 0)
   (1 0 0)          <:  (1 0 0)
   (3 0 0)          <:  (0 2)          }
```

Une autre solution :

```
{  c1               =   (3 0 0)
   c2               =   (2 0 0)
   c3               =   (1)            }
```

Inéquations d'horloges

Trouver des horloges telles que...

Taille zéro

Taille zéro

```
{  (1 0 0)        <:  (1 0 0)
   (1 0 0)        <:  (1 0 0)
   (3 0 0)        <:  (0 2)          }
```

Une autre solution :

```
{  c1             =   (3 0 0)
   c2             =   (2 0 0)
   c3             =   (1)            }
```

Inéquations d'horloges

Trouver des horloges telles que...

Taille zéro

Taille zéro

Taille trois

```
{  (1 0 0)        <:  (1 0 0)
   (1 0 0)        <:  (1 0 0)
   (3 0 0)        <:  (0 2)        }
```

Une autre solution :

```
{  c1             =   (3 0 0)
   c2             =   (2 0 0)
   c3             =   (1)          }
```

Plan

Introduction

Horloges Entières

Le langage

Un compilateur expérimental

Conclusion

Le langage

- Langage fonctionnel de premier ordre
- Manipule uniquement des flots typés
- n-synchrone: Lustre + `buffer`, à la Lucy-n
- Inférence d'horloges entières

Syntaxe

$$
\begin{array}{rcl}
e & ::= & x \\
& | & f\ e \\
& | & \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\
& | & e\ \texttt{where}\ eq^* \\
& | & \texttt{merge}\ ce\ e\ e \\
& | & e\ \texttt{when}\ ce \\
& | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
& | & \texttt{buffer}\ e \\
eq & ::= & x = e \\
ce & ::= & p \\
& | & x \\
& | & ce\ =\ c
\end{array}
$$

Syntaxe

$$
\begin{array}{rcl}
e & ::= & x \qquad \text{Fonctions} \\
& | & f\ e \\
& | & \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\
& | & e\ \texttt{where}\ eq^* \\
& | & \texttt{merge}\ ce\ e\ e \\
& | & e\ \texttt{when}\ ce \\
& | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
& | & \texttt{buffer}\ e \\
eq & ::= & x = e \\
ce & ::= & p \\
& | & x \\
& | & ce\ =\ c
\end{array}
$$

Syntaxe

$$
\begin{array}{rcl}
e & ::= & x \\
 & | & f\ e \\
 & | & \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\
 & | & e\ \texttt{where}\ eq^* \\
 & | & \texttt{merge}\ ce\ e\ e \qquad \text{Contrôle} \\
 & | & e\ \texttt{when}\ ce \\
 & | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
 & | & \texttt{buffer}\ e \\
eq & ::= & x = e \\
ce & ::= & p \\
 & | & x \\
 & | & ce\ =\ c
\end{array}
$$

Syntaxe

$$
\begin{array}{rcl}
e & ::= & x \\
& | & f\ e \\
& | & \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\
& | & e\ \texttt{where}\ eq^* \\
& | & \texttt{merge}\ ce\ e\ e \\
& | & e\ \texttt{when}\ ce \\
& | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
& | & \texttt{buffer}\ e \qquad \text{Mémoire} \\
eq & ::= & x = e \\
ce & ::= & p \\
& | & x \\
& | & ce\ =\ c
\end{array}
$$

Syntaxe

$$
\begin{array}{rcl}
e & ::= & x \\
& | & f\ e \\
& | & \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\
& | & e\ \texttt{where}\ eq^* \\
& | & \texttt{merge}\ ce\ e\ e \\
& | & e\ \texttt{when}\ ce \\
& | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
& | & \texttt{buffer}\ e \\
eq & ::= & x = e \\
ce & ::= & p \\
& | & x \\
& | & ce\ =\ c
\end{array}
$$

Sémantique

Sémantique de Kahn $[\![\texttt{Bool}]\!]^K = \mathbb{B}^\infty$

Sémantique

Sémantique de Kahn $[\![\texttt{Bool}]\!]^K = \mathbb{B}^\infty$

[true, true, false, false, false, true, false, true, false...

Sémantique

Sémantique de Kahn $\quad [\![\texttt{Bool}]\!]^K = \mathbb{B}^\infty$

[true, true, false, false, false, true, false, true, false...

Sémantique synchrone $\quad [\![\texttt{Bool}]\!]^S = (\mathbb{B}^*)^\infty$

Sémantique

Sémantique de Kahn $[\![\texttt{Bool}]\!]^K = \mathbb{B}^\infty$

[true, true, false, false, false, true, false, true, false...

Sémantique synchrone $[\![\texttt{Bool}]\!]^S = (\mathbb{B}^*)^\infty$

[[true, true], [], [false, false, false], [true, false], [true], [false], ...

Sémantique

Sémantique de Kahn $\quad [\![\texttt{Bool}]\!]^K = \mathbb{B}^\infty$

[true, true, false, false, false, true, false, true, false...

Sémantique synchrone $\quad [\![\texttt{Bool}]\!]^S = (\mathbb{B}^*)^\infty$

[[true, true], [], [false, false, false], [true, false], [true], [false], ...

[2, 0, 3, 2, 1, 1, ...

Sémantique

Sémantique de $\texttt{bl}\rrbracket^K = \mathbb{B}^\infty$

Unique

[true, true, false, false, false, true, false, true, false...

Sémantiqu$|^S = (\mathbb{B}^*)^\infty$

Dépendante des types d'horloge

[[true, true], [], [false, false, false], [true, false], [true], [false], ...

[2, 0, 3, 2, 1, 1, ...

Typage d'horloges

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck} \qquad \frac{\Gamma \vdash e :: ck \qquad \Gamma \vdash ce :: ck}{\Gamma \vdash e \texttt{ when } ce :: ck \texttt{ on } ce}$$

$$\frac{}{\Gamma \vdash c :: ck} \qquad \frac{\Gamma \vdash e :: ck \qquad ck <: ck'}{\Gamma \vdash \texttt{buffer } e :: ck'}$$

$$\ldots$$

Typage d'horloges

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck \qquad \Gamma \vdash e :: ck \qquad \Gamma \vdash ce :: ck}{\ldots}$$

En pratique : inférence d'horloge

- Accumuler les contraintes en parcourant l'AST
- Les réduire en contraintes sur les mots périodiques
- Réduire ces dernières à un problème de PLNE

. . .

Correction du typage d'horloges

La sémantique de Kahn et la sémantique synchrone d'un programme bien typé coïncident

Correction du typage d'horloges

La sémantique de Kahn et la sémantique synchrone d'un programme bien typé coïncident

$$unpack([\![\vdash e :: ck]\!]^S) = [\![e]\!]^K$$

Plan

Introduction

Horloges Entières

Le langage

Un compilateur expérimental

Conclusion

Schéma de compilation

$$[\![f]\!]^K : A^\infty \to B^\infty$$

Schéma de compilation

$$[\![f]\!]^K : A^\infty \to B^\infty$$

Clocking

$$[\![f]\!]^S : (A^*)^\infty \to (B^*)^\infty$$

Schéma de compilation

$$[\![f]\!]^K : A^\infty \to B^\infty$$

Clocking

$$[\![f]\!]^S : (A^*)^\infty \to (B^*)^\infty$$

Codegen

$$(\!|f|\!) : \exists S.(S \times (S \times A^* \to S \times B^*))$$

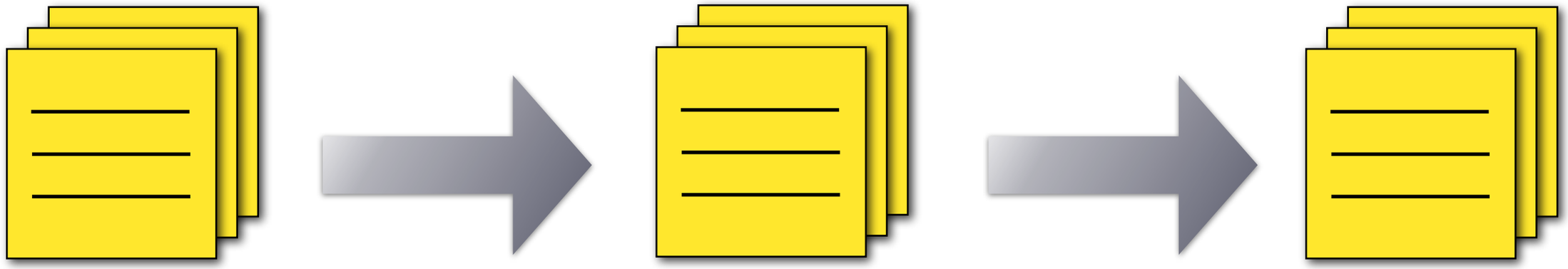
Objectifs de la génération de code

```
let node zigzag x = o where
    rec (x0, x1, x2, x3, x4, x5, x6, x7, x8)
          = split x
            with (0 1 3 6 4 2 5 7 8)
            by (0, 1, 2, 3, 4, 5, 6, 7, 8)
    and o = merge (0 1 2 3 4 5 6 7 8) with
              | 0 -> buffer x0
              | 1 -> buffer x1
               ...
              | 8 -> buffer x8
            end
```

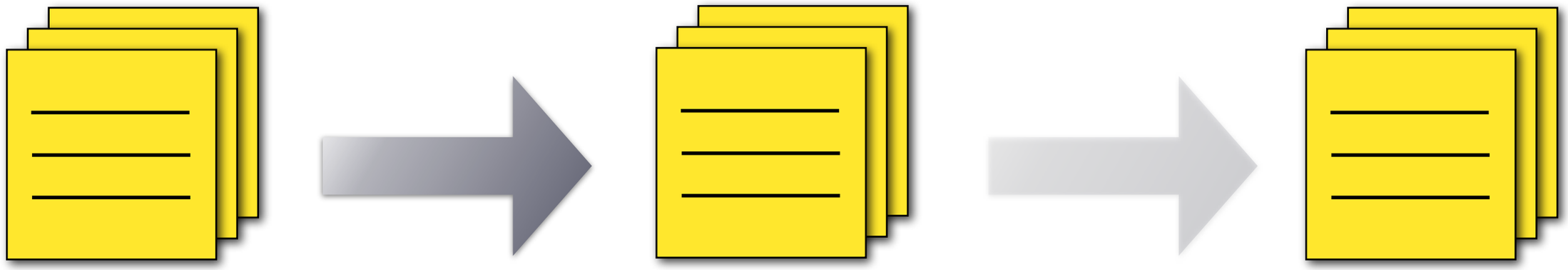
Objectifs de la génération de code

```
void zigzag_step(int x[9], int o[9]) {
  for (int i = 0; i < 9; i++) {
    switch(i) {
    case 0:
       o[0] = x[i]; break;
      case 1:
        o[1] = x[i]; break;
      case 2:
        o[3] = x[i]; break;
       ...
    }
  }
}
```

Flot de compilation, en bref



Flot de compilation, en bref

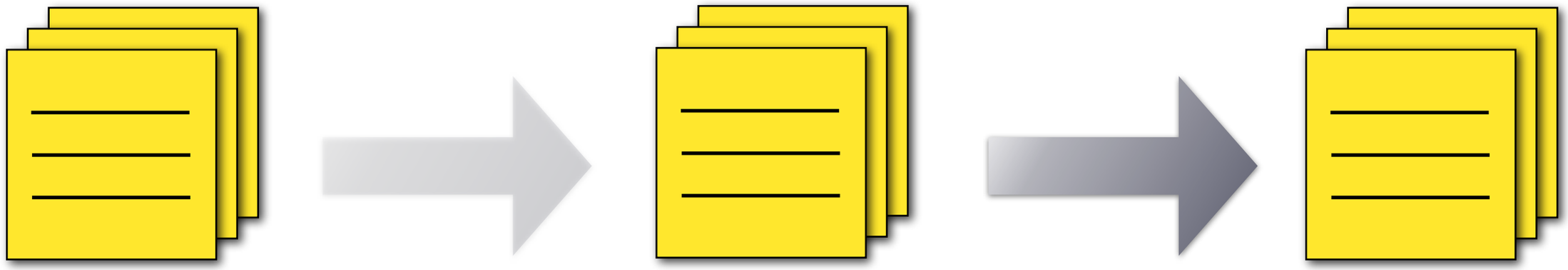


AcidS NIR C

Front-end

- Parsing/Scoping/Typage
- Inférence d'horloge
- Analyse de causalité
- Desugaring

Flot de compilation, en bref



Middle-end/Back-end

- Optimisations
- Explicitation format des données & contrôle
- Séquentialisation
- Production de code impératif

Analyse de causalité

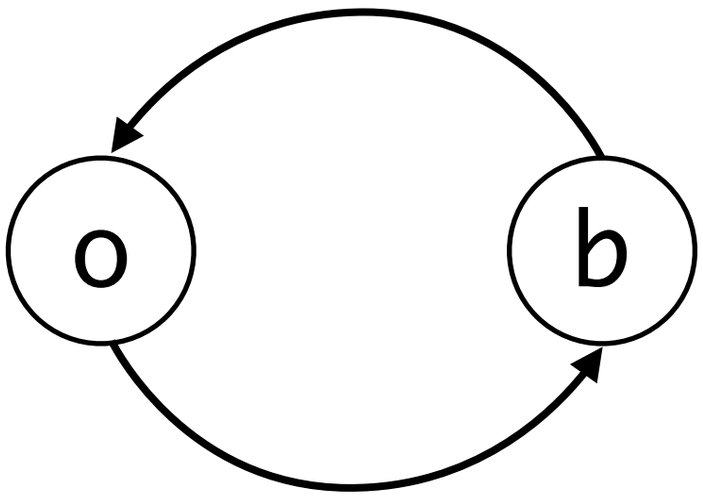
```
let node bad () = o where
    rec o = b
    and b = o
```

```
bad :: 'a1 -> 'a
```

Analyse de causalité

```
let node bad () = o where
    rec o = b
    and b = o
```

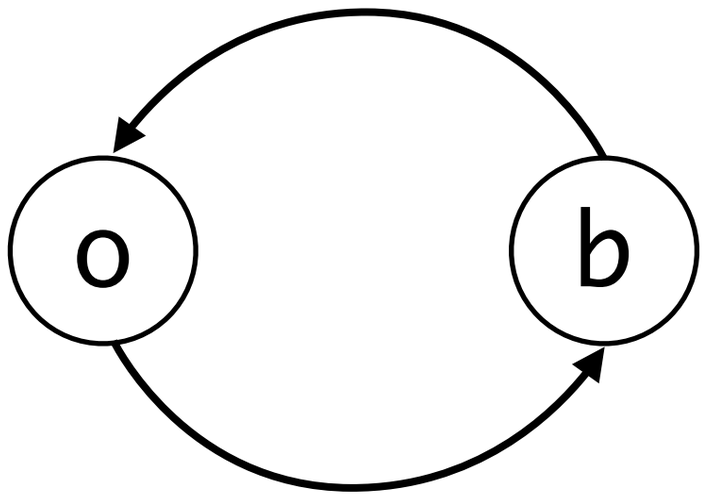
```
bad :: 'a1 -> 'a
```



Analyse de causalité

```
let node bad () = o where
    rec o = b
    and b = o
```

```
bad :: 'a1 -> 'a
```

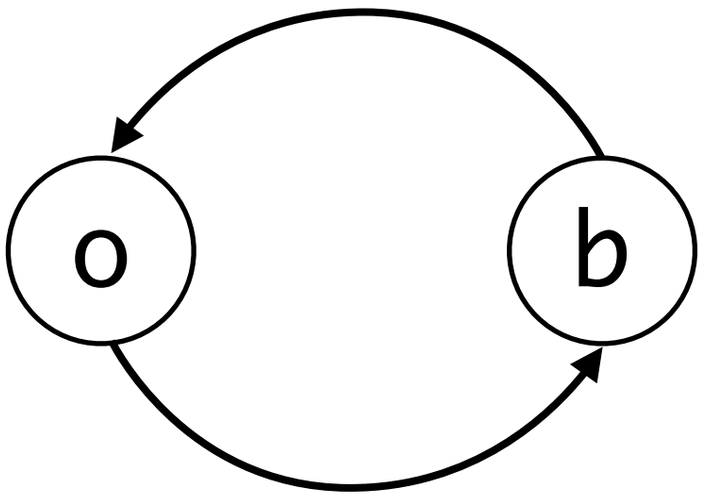


Refusé.

Analyse de causalité

```
let node nat () = o where
    rec o = merge true(false) 0 b
    and b = 1 + buffer o
```

```
nat :: 'a1 -> 'a
```

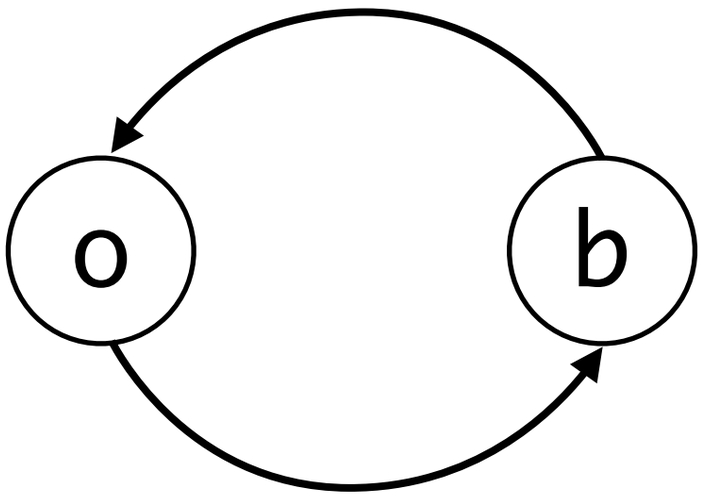


Analyse de causalité

```
let node nat () = o w
    rec o = merge tru(false) 0 b
    and b = 1 + buffer o
```

$(1) \ll: 0(1)$

```
nat :: 'a1 -> 'a
```

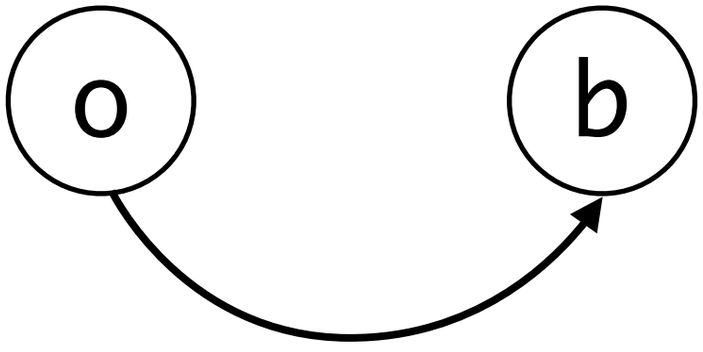


Analyse de causalité

```
let node nat () = o w
    rec o = merge tru(false) 0 b
    and b = 1 + buffer o
```

$(1) \ll: 0(1)$

```
nat :: 'a1 -> 'a
```

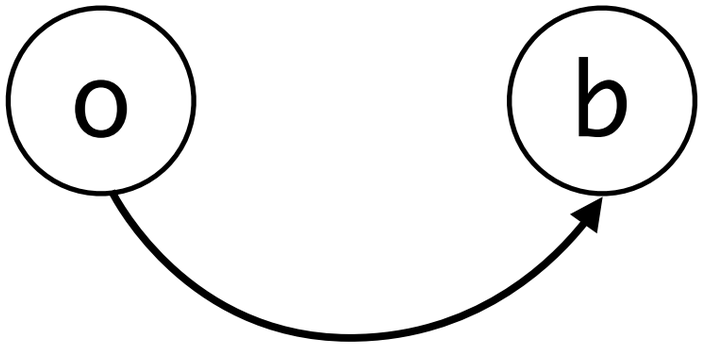


Analyse de causalité

```
let node nat () = o w
    rec o = merge tru(false) 0 b
    and b = 1 + buffer o
```

$(1) \ll: 0(1)$

```
nat :: 'a1 -> 'a
```

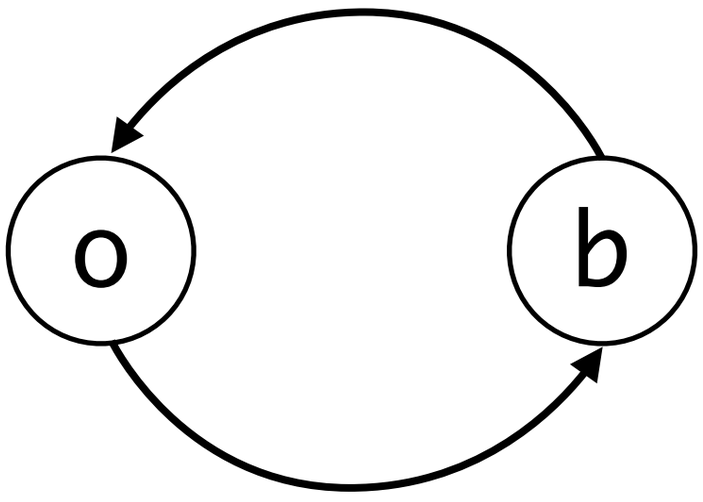


Accepté.

Analyse de causalité

```
let node nat () = o where
    rec o = merge true(false) 0 b
    and b = 1 + buffer o
```

```
nat :: 'a1 -> 'a on (2)
```

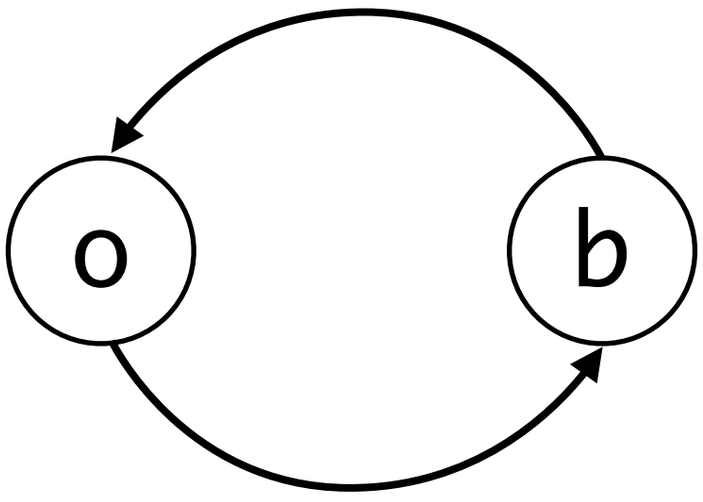


Analyse de causalité

```
let node nat () = o w
    rec o = merge tr  (false) 0 b
    and b = 1 + buffer o
```

$(2) \not\ll : 1(2)$

```
nat :: 'a1 -> 'a on (2)
```

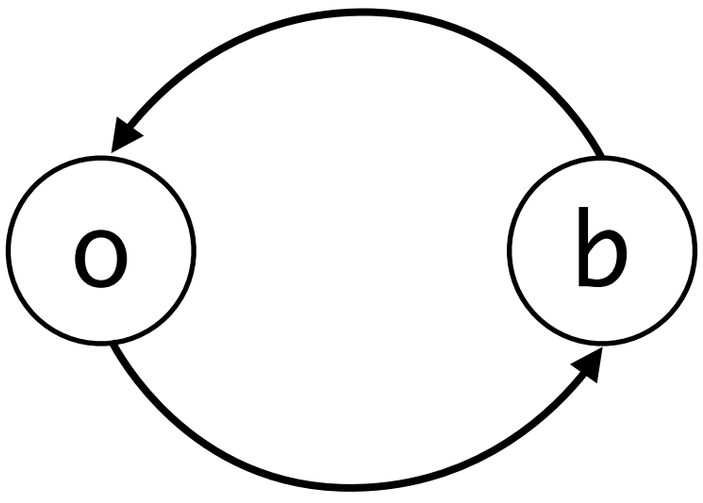


Analyse de causalité

```
let node nat () = o w
    rec o = merge tr  (false) 0 b
    and b = 1 + buffer o
```

$(2) \not\ll : 1(2)$

```
nat :: 'a1 -> 'a on (2)
```



Refusé.

Le compilateur `asc` en pratique

- Compile un plus gros langage
- Horloges traduites en boucles et tableaux
- Cible C, plus tard VHDL

Le compilateur `asc` en pratique

- Compile un plus gros langage
- Horloges traduites en boucles et tableaux
- Cible C, plus tard VHDL

État courant

- Front-end fait, y compris clocking et causalité
- Génération de code naïve quasi terminée
- Meilleur clocking et codegen: beaucoup de boulot
- 20 kLoC de code OCaml

Plan

Introduction

Horloges Entières

Le langage

Un compilateur expérimental

Conclusion

Travail futur

- Beaucoup de boulot sur la résolution de contraintes
- Beaucoup de boulot sur la génération de meilleur code, séquentiel et parallèle
- Beaucoup de boulot sur l'expressivité du langage
- Beaucoup de boulot...

Conclusion

- Un point de vue «langage» sur l'ordonnancement
- De vieux principes dans un nouveau cadre :
  - Typage d'horloge
  - Inférence de boucles et tableaux
- Travaux connexes: *DF, désynchronisation...
- Impossible sans Lustre, Lucid Synchrone, Lucy-n

Conclusion

Merci !

- Un point de vue «langage» sur l'ordonnancement
- De vieux principes dans un nouveau cadre :
  - Typage d'horloge
  - Inférence de boucles et tableaux
- Travaux connexes: *DF, désynchronisation...
- Impossible sans Lustre, Lucid Synchrone, Lucy-n