Réseaux de Kahn à rafales et horloges entières

JFLA 2014

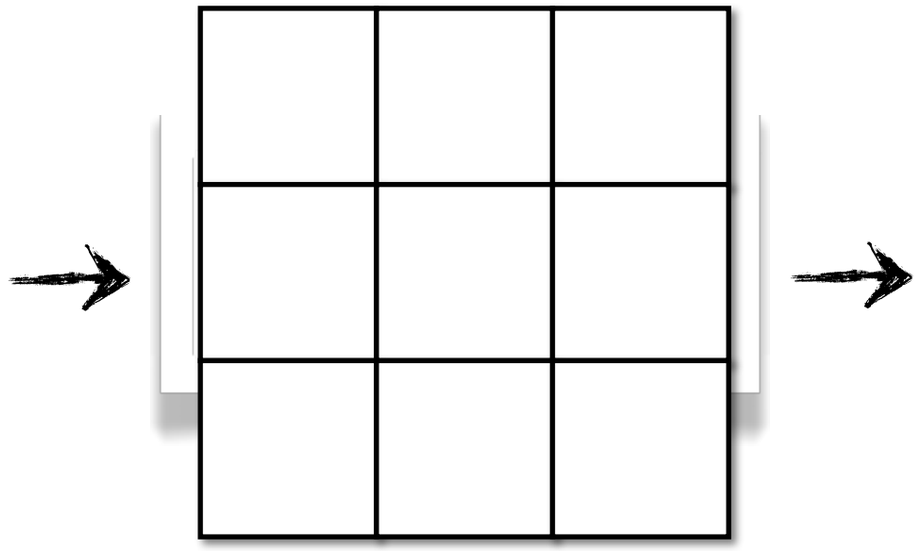
Adrien Guatto
PARKAS, DI ENS Paris

Louis Mandel
Collège de France

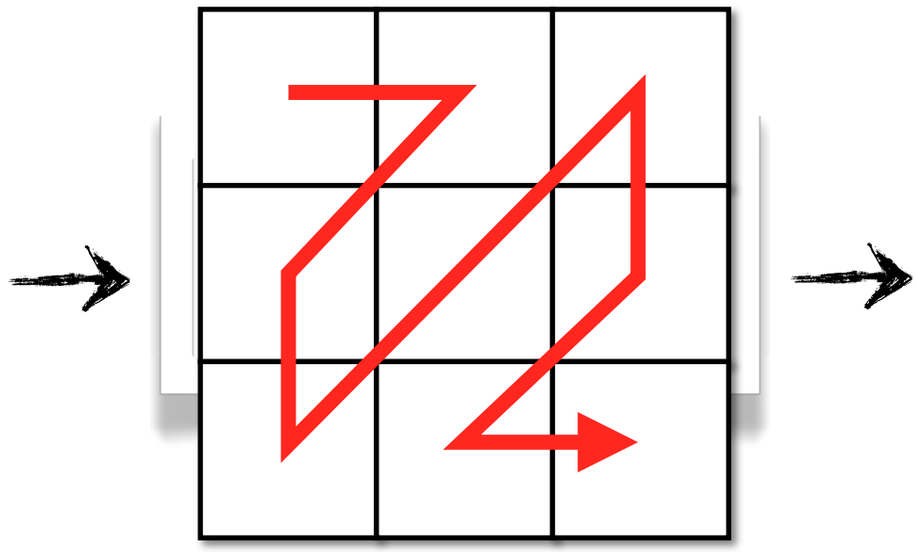
Parcours Zig-Zag JPEG

Parcours Zig-Zag JPEG

Parcours Zig-Zag JPEG



Parcours Zig-Zag JPEG

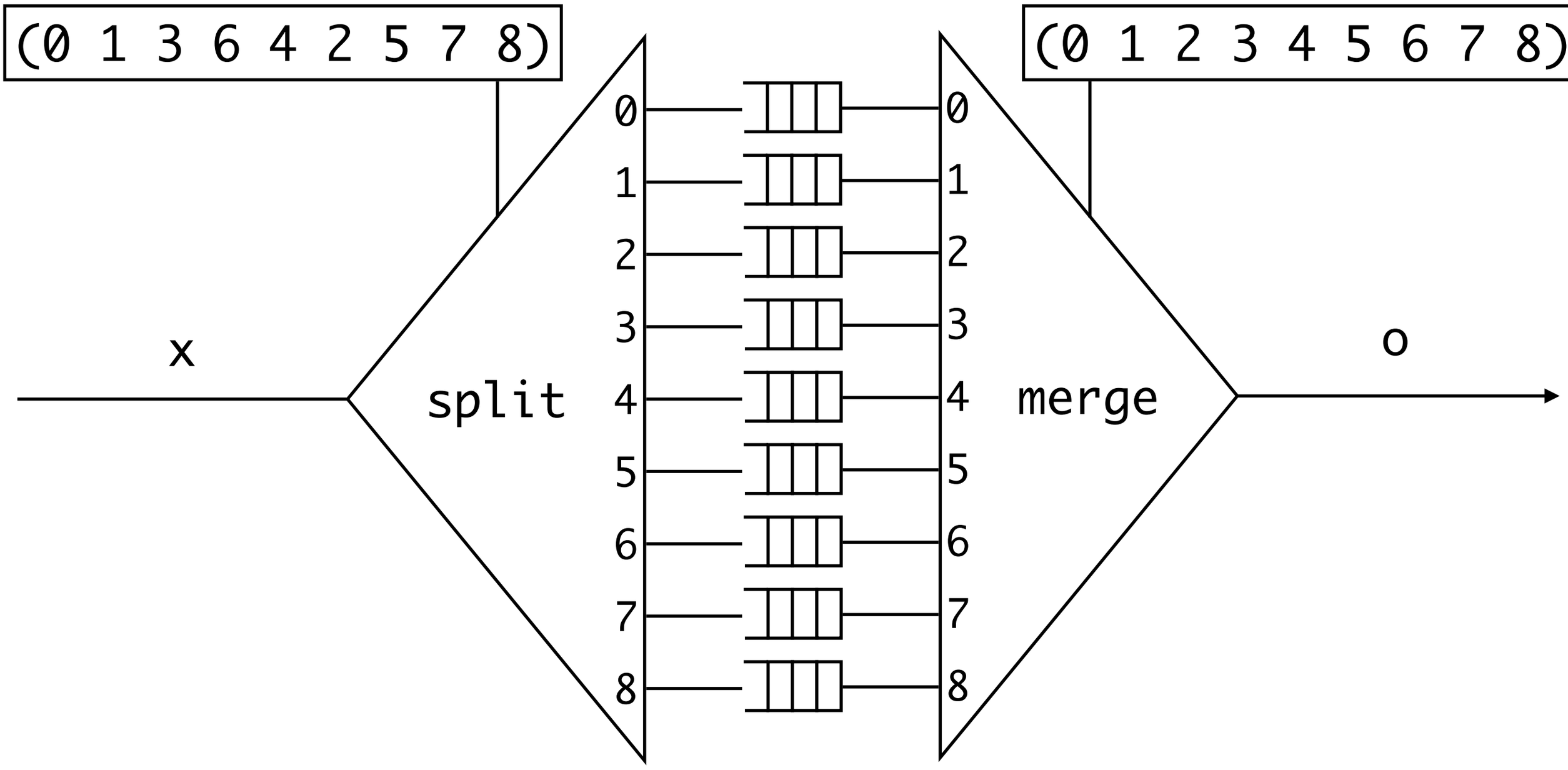


Parcours Zig-Zag JPEG

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, 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
```

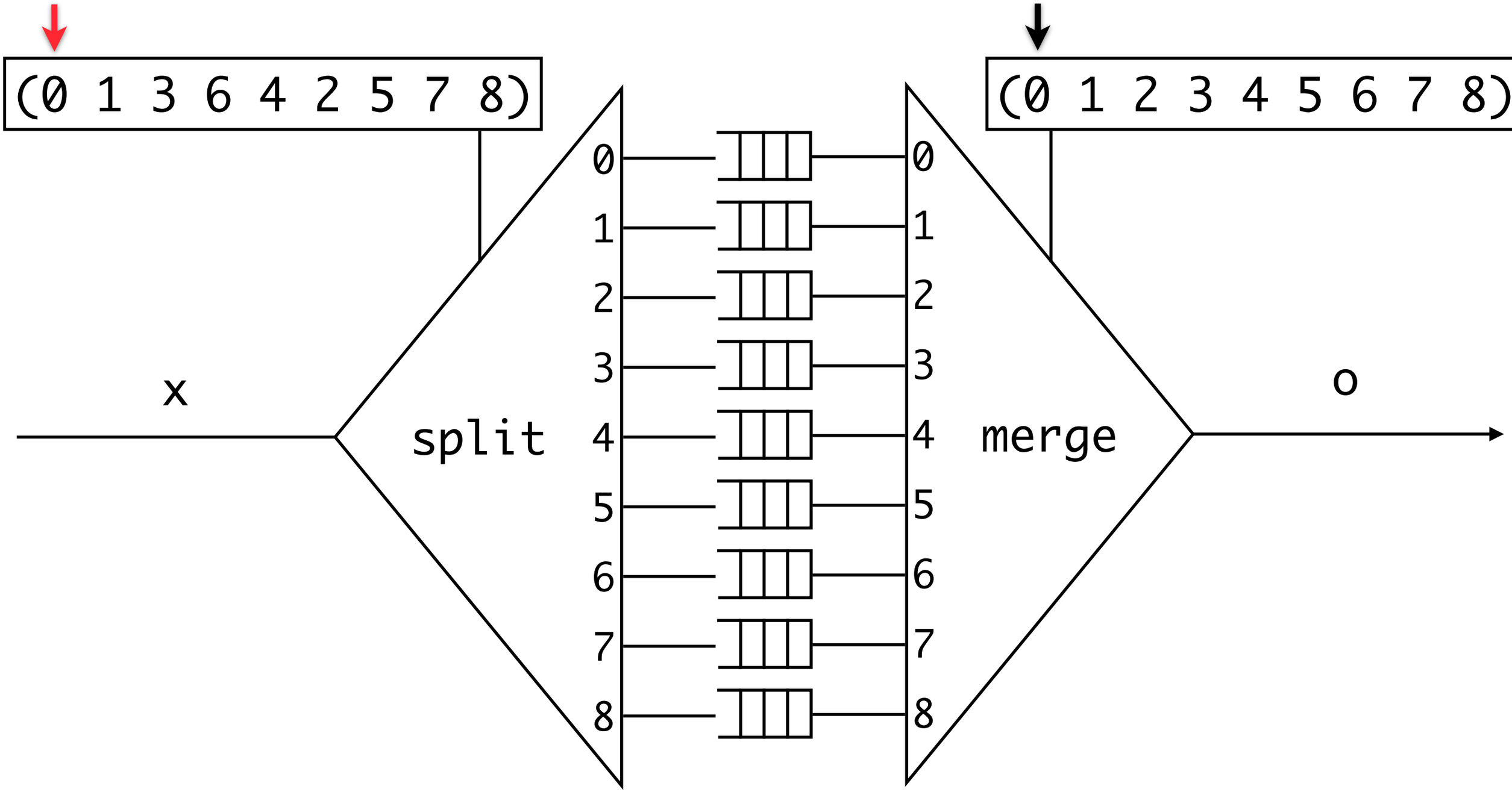
Parcours Zig-Zag JPEG en AcidS

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

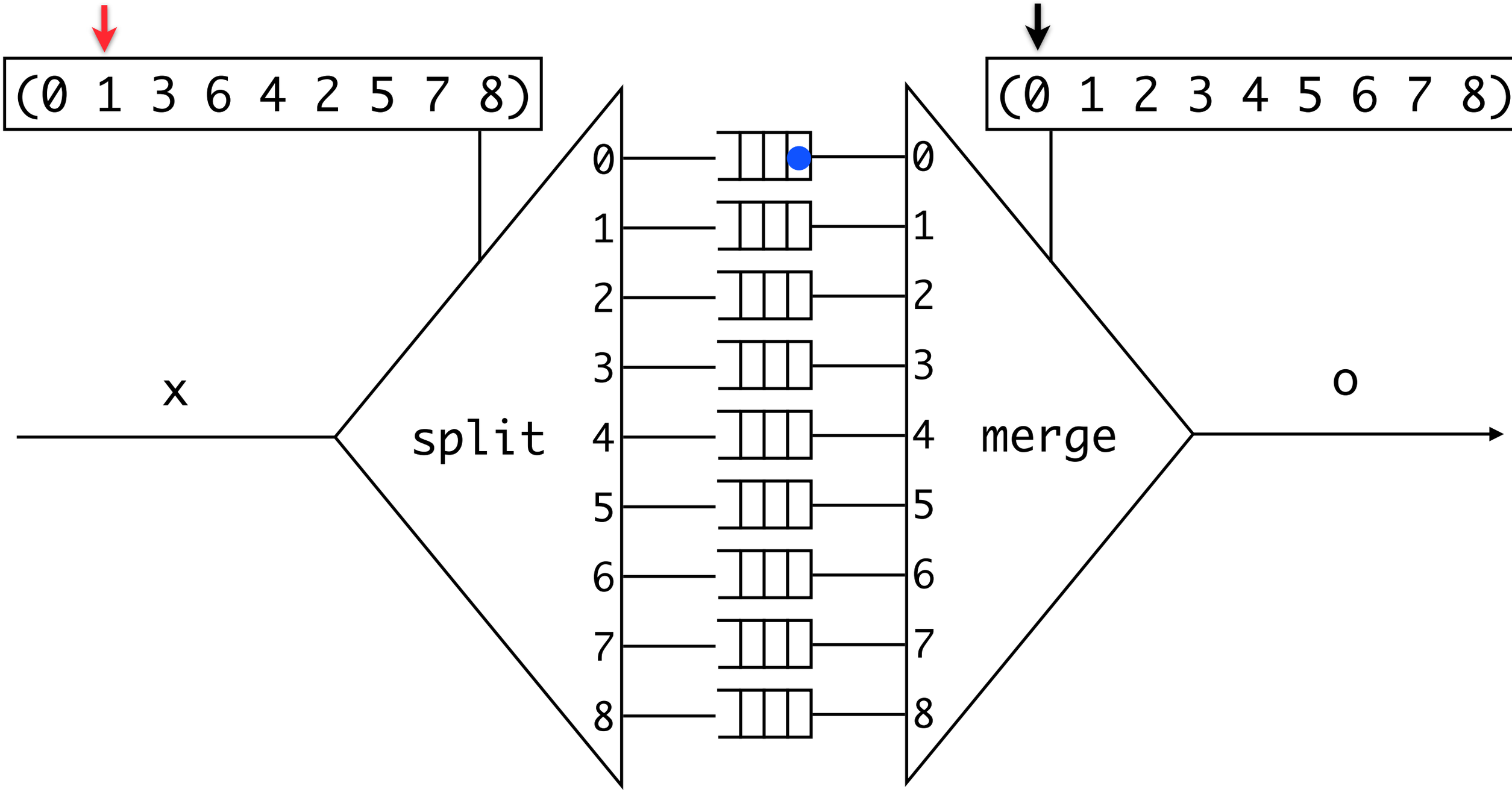
```
$ asc -i zigzag.as
val zigzag
  : 'x -> 'x
  :: 'a -> 'a on 0^3(1)
```

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

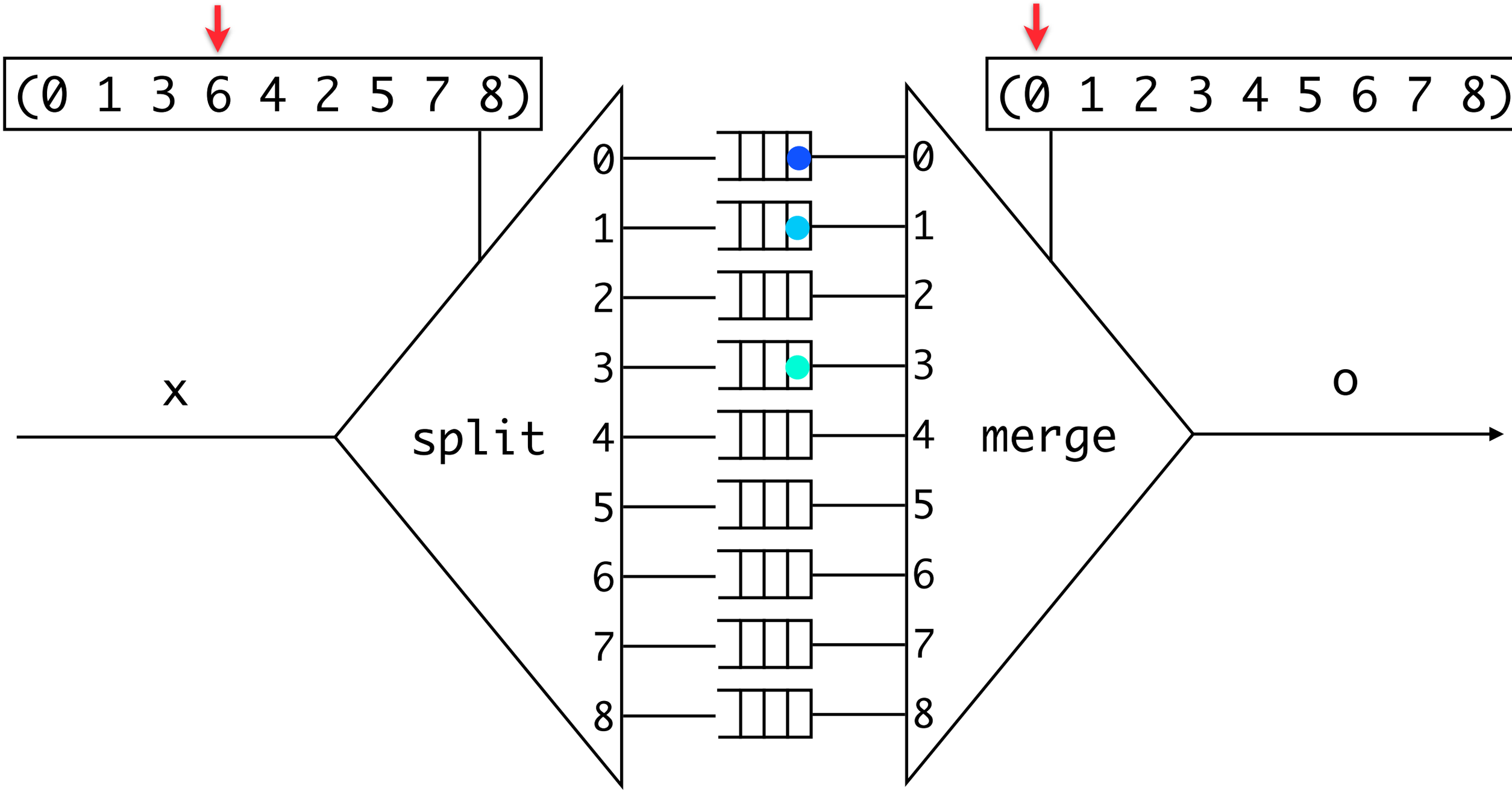
Parcours Zig-Zag JPEG, dataflow scalaire



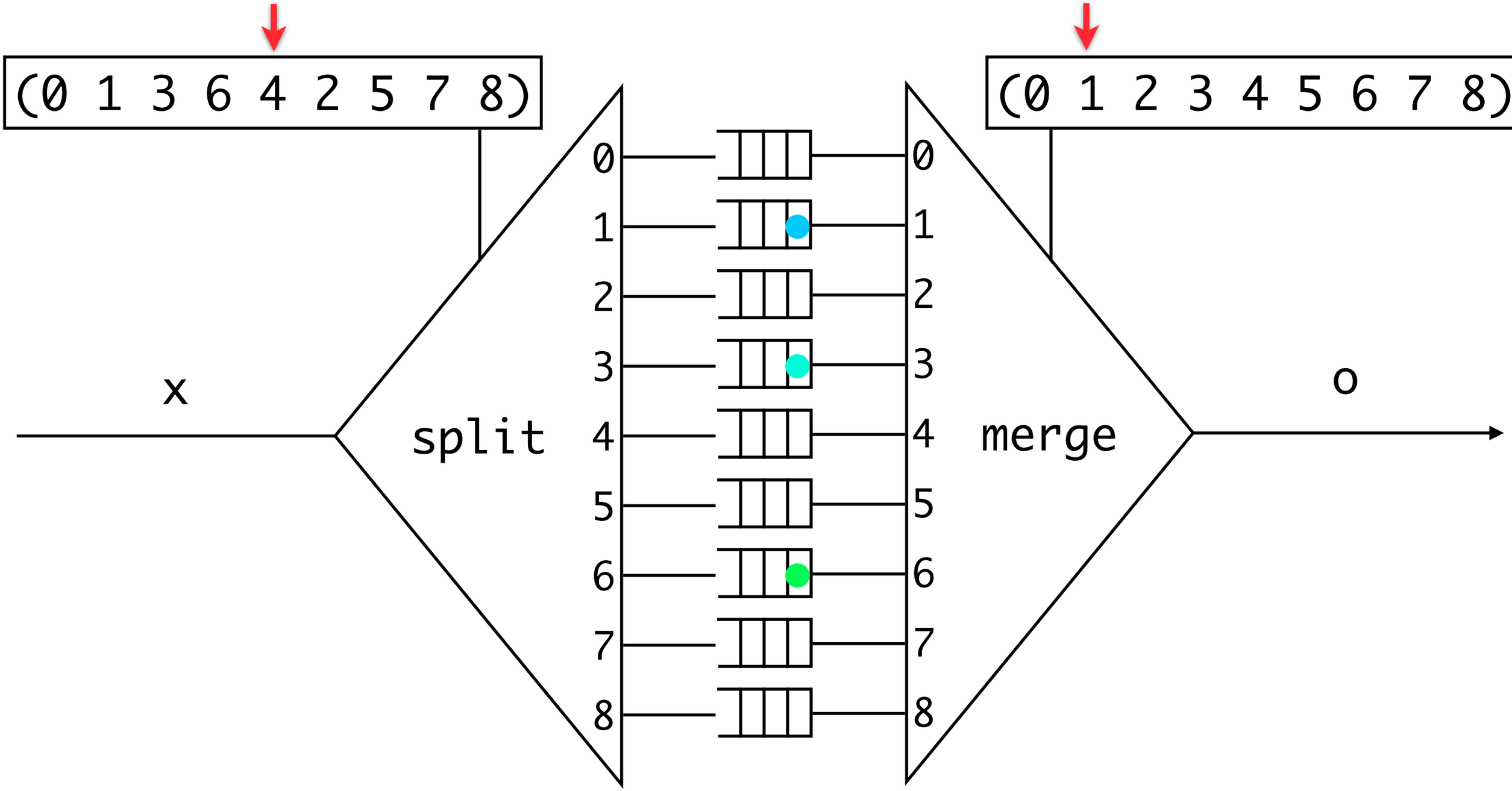
Parcours Zig-Zag JPEG, dataflow scalaire



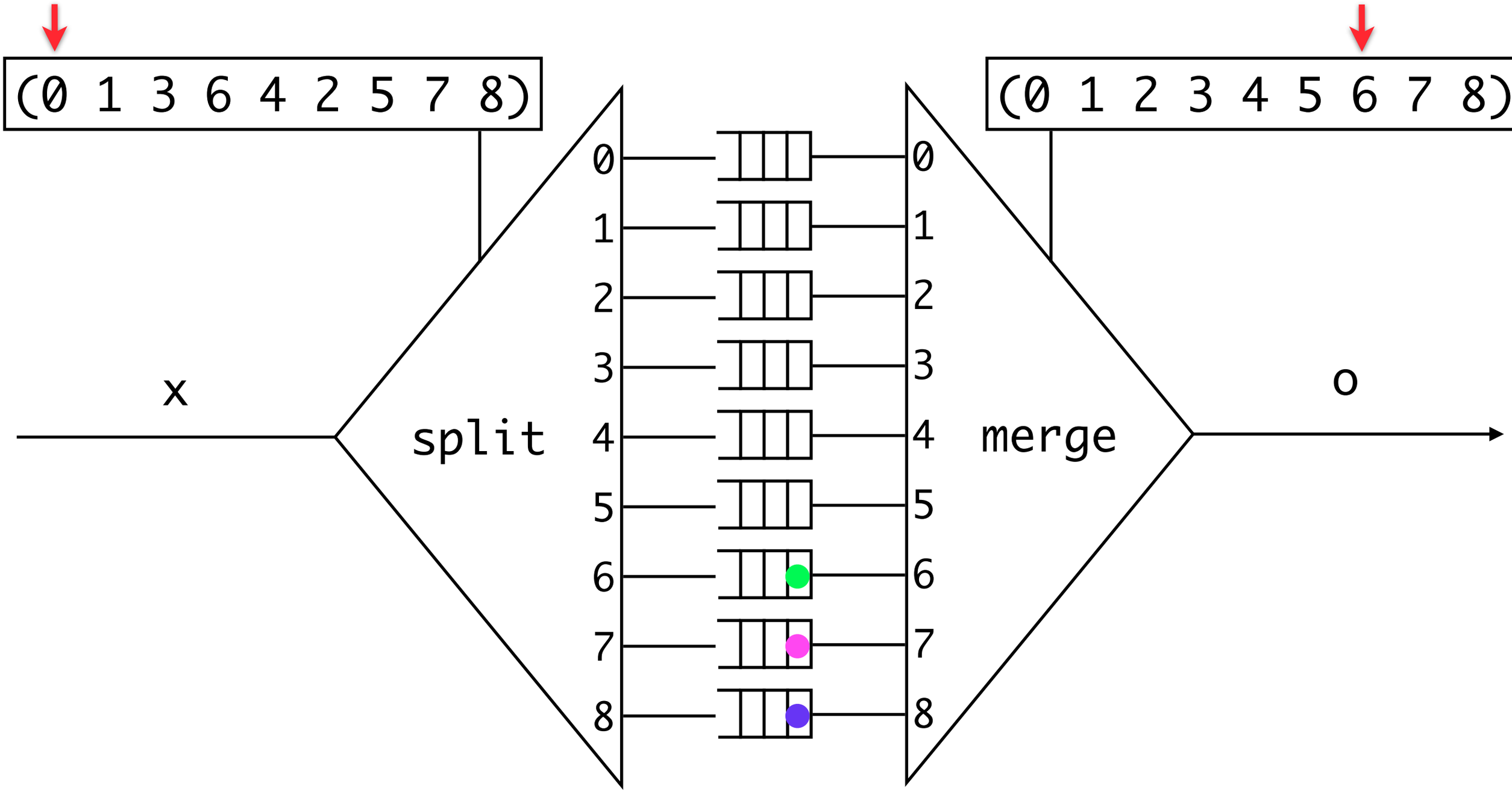
Parcours Zig-Zag JPEG, dataflow scalaire



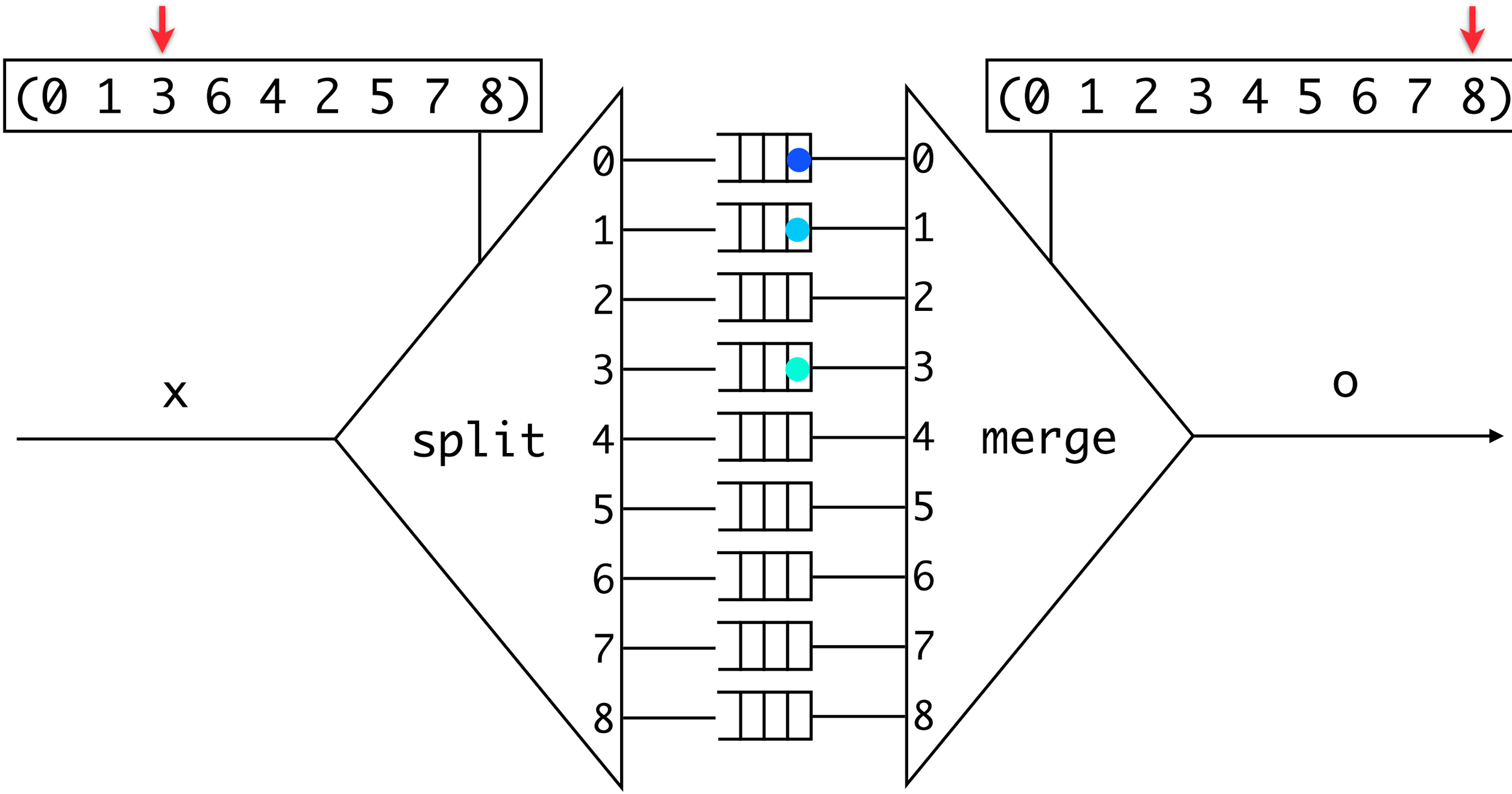
Parcours Zig-Zag JPEG, dataflow scalaire



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)
```

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

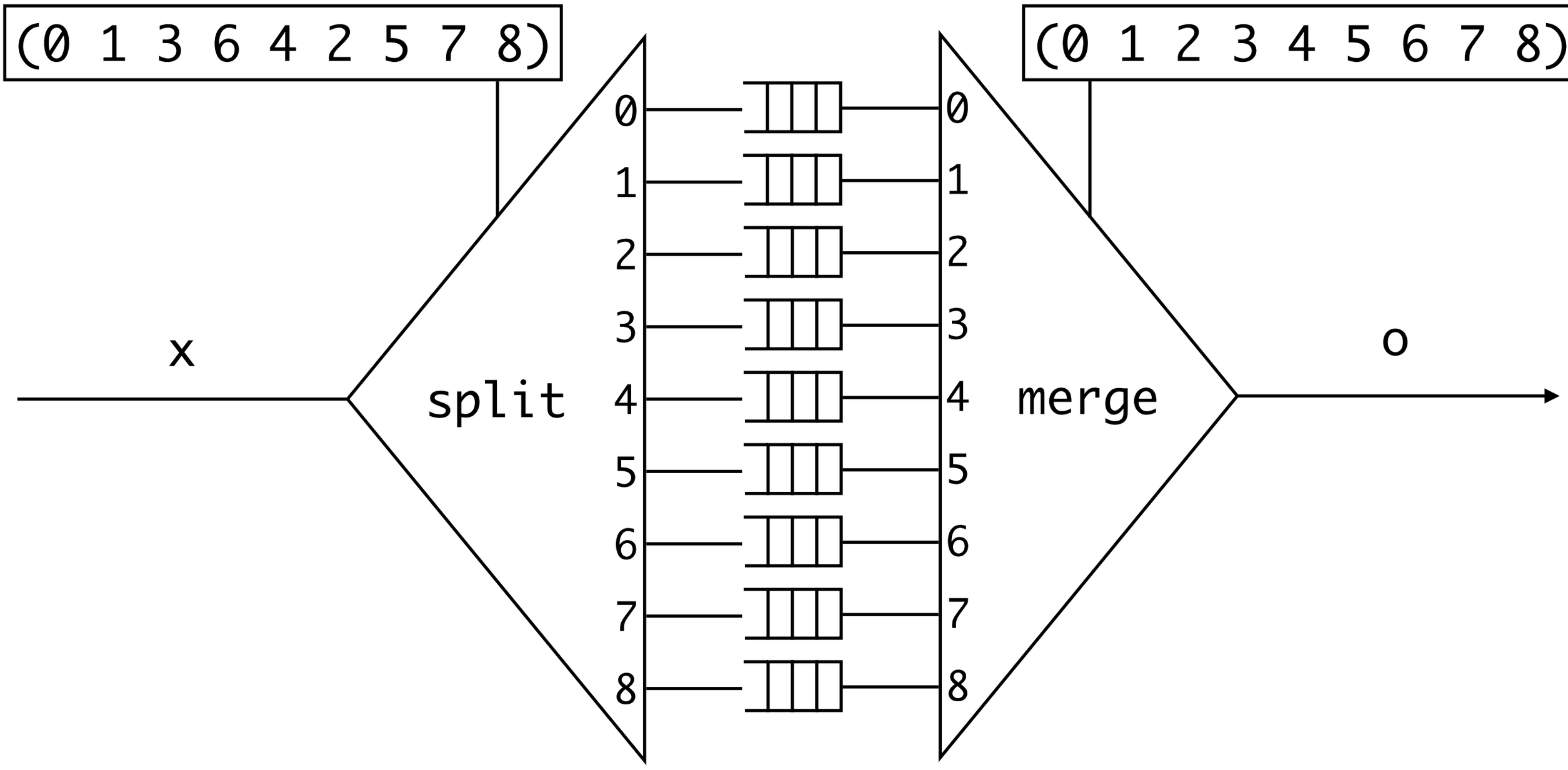
Parcours Zig-Zag JPEG en AcidS

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

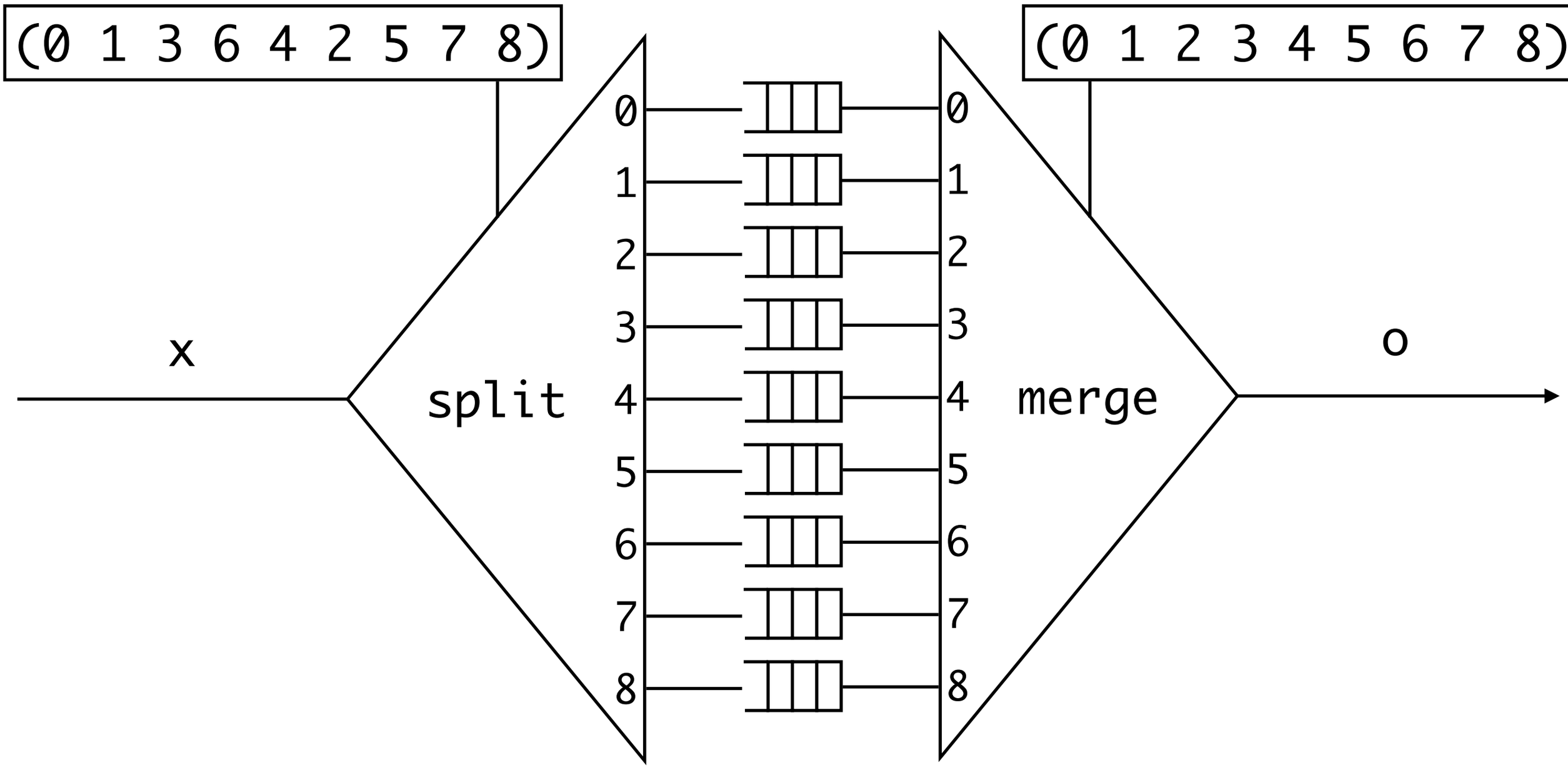
```
$ asc -i zigzag.as -max_burst 9
val zigzag
: 'x -> ‘x
:: 'a on (9) -> 'a on (9)
```

```
                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

Le langage

Causalité

Conclusion

Le langage

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

Syntaxe

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

Syntaxe

$$
\begin{array}{rcl}
e & ::= & \left.\begin{array}{l} c \\ x \\ f\ e \\ \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\ e\ \texttt{where}\ eq^* \end{array}\right. \\
 & | & \texttt{merge}\ ce\ e\ e \\
 & | & e\ \texttt{when}\ ce \\
 & | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
 & | & \texttt{buffer}\ e \\
eq & ::= & x = e \\
ce & ::= & c^*(c^+) \\
 & | & ce\ =\ c
\end{array}
$$

Fonctions

Syntaxe

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

Syntaxe

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

Syntaxe

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

Syntaxe

$$
\begin{array}{rcl}
e & ::= & c \\
 & | & x \\
 & | & f\ e \\
 & | & \texttt{let node}\ f\ x\ =\ e\ \texttt{in}\ e \\
 & | & e\ \texttt{where}\ eq^* \\
 & | & \texttt{merge}\ ce\ e\ e \\
 & | & e\ \texttt{when}\ ce \\
 & | & \texttt{split}\ e\ \texttt{by}\ ce\ \texttt{with}\ c^* \\
 & | & \texttt{buffer}\ e \\
eq & ::= & x = e \\
ce & ::= & c^*(c^+) \quad \text{Mot ultimement périodique} \\
 & | & ce\ =\ c
\end{array}
$$

Syntaxe

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

Syntaxe

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

Sémantique des types

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

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

Sémantique des types

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

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

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

Sémantique des types

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 des types

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], ...

clock

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

Sémantique des termes : constantes

Sémantique de Kahn

Sémantique synchrone

Sémantique des termes : constantes

Sémantique de Kahn $[\![c]\!]^K = c^\omega$

$$[\![\ 'a'\ ]\!]^K = [\ 'a', 'a', 'a' \ldots$$

Sémantique synchrone

Sémantique des termes : constantes

Sémantique de Kahn $\quad [\![c]\!]^K = c^\omega$

$$[\![\ 'a'\ ]\!]^K = [\ 'a', 'a', 'a' \ldots$$

Sémantique synchrone

$$[\![\ \texttt{'a'}\ ]\!]^S \quad =_? \quad [\ [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ]\ \ldots$$

Sémantique des termes : constantes

Sémantique de Kahn $\quad [\![c]\!]^K = c^\omega$

$$[\![\ 'a'\ ]\!]^K = [\ 'a', 'a', 'a' \ldots$$

Sémantique synchrone

$$\begin{aligned} [\![\ \texttt{'a'}\ ]\!]^S &=_? [\ [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ] \ldots \\ &=_? [\ [\ \texttt{'a'}, \texttt{'a'}\ ], [\ ], [\ \texttt{'a'}, \texttt{'a'}, \texttt{'a'}\ ] \ldots \end{aligned}$$

Sémantique des termes : constantes

Sémantique de Kahn $\quad [\![c]\!]^K = c^\omega$

$$[\![\ 'a'\ ]\!]^K = [\ 'a', 'a', 'a' \ldots$$

Sémantique synchrone

$$\begin{aligned} [\![\ \texttt{'a'}\ ]\!]^S &=_? [\ [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ], [\ \texttt{'a'}\ ] \ldots \\ &=_? [\ [\ \texttt{'a'}, \texttt{'a'}\ ], [\ ], [\ \texttt{'a'}, \texttt{'a'}, \texttt{'a'}\ ] \ldots \\ &=_? \ldots \end{aligned}$$

Sémantique des termes : constantes

Sémantique de Kahn $\quad [\![c]\!]^K = c^\omega$

$$[\![\ 'a'\ ]\!]^K = [\ 'a',\ 'a',\ 'a' \ldots$$

Sémantique synchrone $\quad [\![c :: ck]\!]^S = pack_{[\![ck]\!]}(c^\omega)$

$$\begin{array}{lll} [\![\ \texttt{'a'}\ ]\!]^S & =? & [\ [\ \texttt{'a'}\ ],\ [\ \texttt{'a'}\ ],\ [\ \texttt{'a'}\ ],\ [\ \texttt{'a'}\ ]\ \ldots \\ & =? & [\ [\ \texttt{'a'}, \texttt{'a'}\ ],\ [\ ],\ [\ \texttt{'a'}, \texttt{'a'}, \texttt{'a'}\ ]\ \ldots \\ & =? & \ldots \end{array}$$

Sémantique des termes : opérateurs

Sémantique de Kahn

Sémantique synchrone

Sémantique des termes : opérateurs

Sémantique de Kahn

$$[\![e_1 + e_2]\!]^K = map2\ (+)\ [\![e_1]\!]^K\ [\![e_2]\!]^K$$

Sémantique synchrone

Sémantique des termes : opérateurs

Sémantique de Kahn

$$[\![e_1 + e_2]\!]^K = map2\ (+)\ [\![e_1]\!]^K\ [\![e_2]\!]^K$$

Sémantique synchrone

$$[\![e_1 + e_2]\!]^S = map2\ (map2\ (+))\ [\![e_1]\!]^S\ [\![e_2]\!]^S$$

Sémantique des termes : opérateurs

Sémantique synchrone

$$[\![e_1 + e_2]\!]^S = map2\ (map2\ (+))\ [\![e_1]\!]^S\ [\![e_2]\!]^S$$

Sémantique des termes : opérateurs

Sémantique synchrone

$$[\![e_1 + e_2]\!]^S = map2\ (map2\ (+))\ [\![e_1]\!]^S\ [\![e_2]\!]^S$$

$[\![e_1]\!]^S$	$[0,1,2]$	$[]$	$[3,4]$	$[5]$	$\ldots$
$[\![e_2]\!]^S$	$[1,3,4]$	$[]$	$[5,6]$	$[8]$	$\ldots$
$[\![e_1]\!]^S + [\![e_2]\!]^S$	$[1,4,6]$	$[]$	$[8,10]$	$[13]$	$\ldots$

Sémantique des termes : opérateurs

Sémantique synchrone

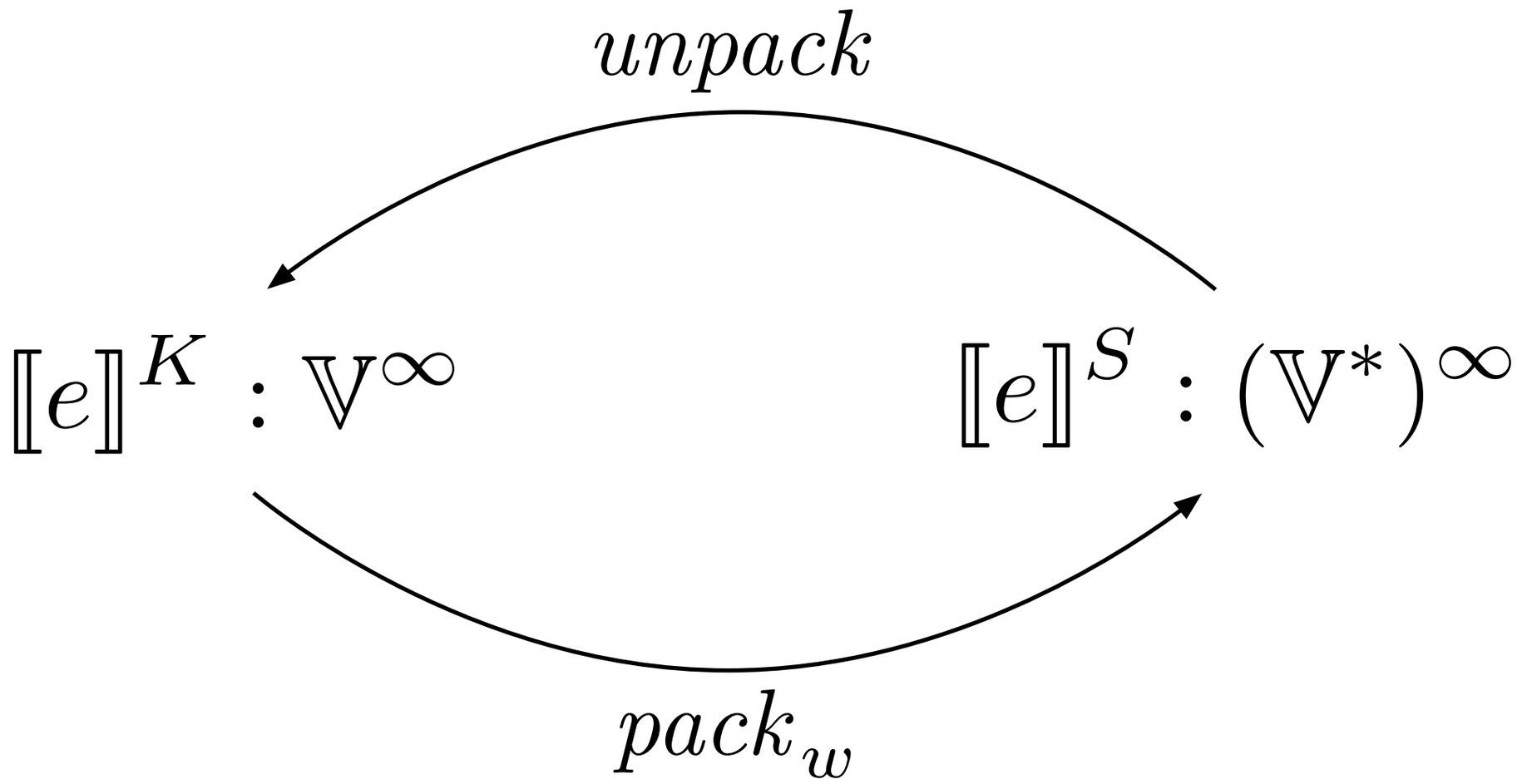
$$[\![e_1 + e_2]\!]^S = map2\ (map2\ (+))\ [\![e_1]\!]^S\ [\![e_2]\!]^S$$

$[\![e_1]\!]^S$	$[0,1,2]$	$[]$	$[3,4]$	$[5]$	$\ldots$
$[\![e_2]\!]^S$	$[1,3,4]$	$[]$	$[5,6]$	$[8]$	$\ldots$
$[\![e_1]\!]^S + [\![e_2]\!]^S$	$[1,4,6]$	$[]$	$[8,10]$	$[13]$	$\ldots$

$[\![e_1]\!]^S$	$[0,1]$	$[]$	$[2,3,4]$	$[5]$	$\ldots$
$[\![e_2]\!]^S$	$[1,3,4]$	$[]$	$[5,6,8]$	$[]$	$\ldots$
$[\![e_1]\!]^S + [\![e_2]\!]^S$	$[1,4]$	$[]$	$[7,9,12]$	$[]$	$\ldots$

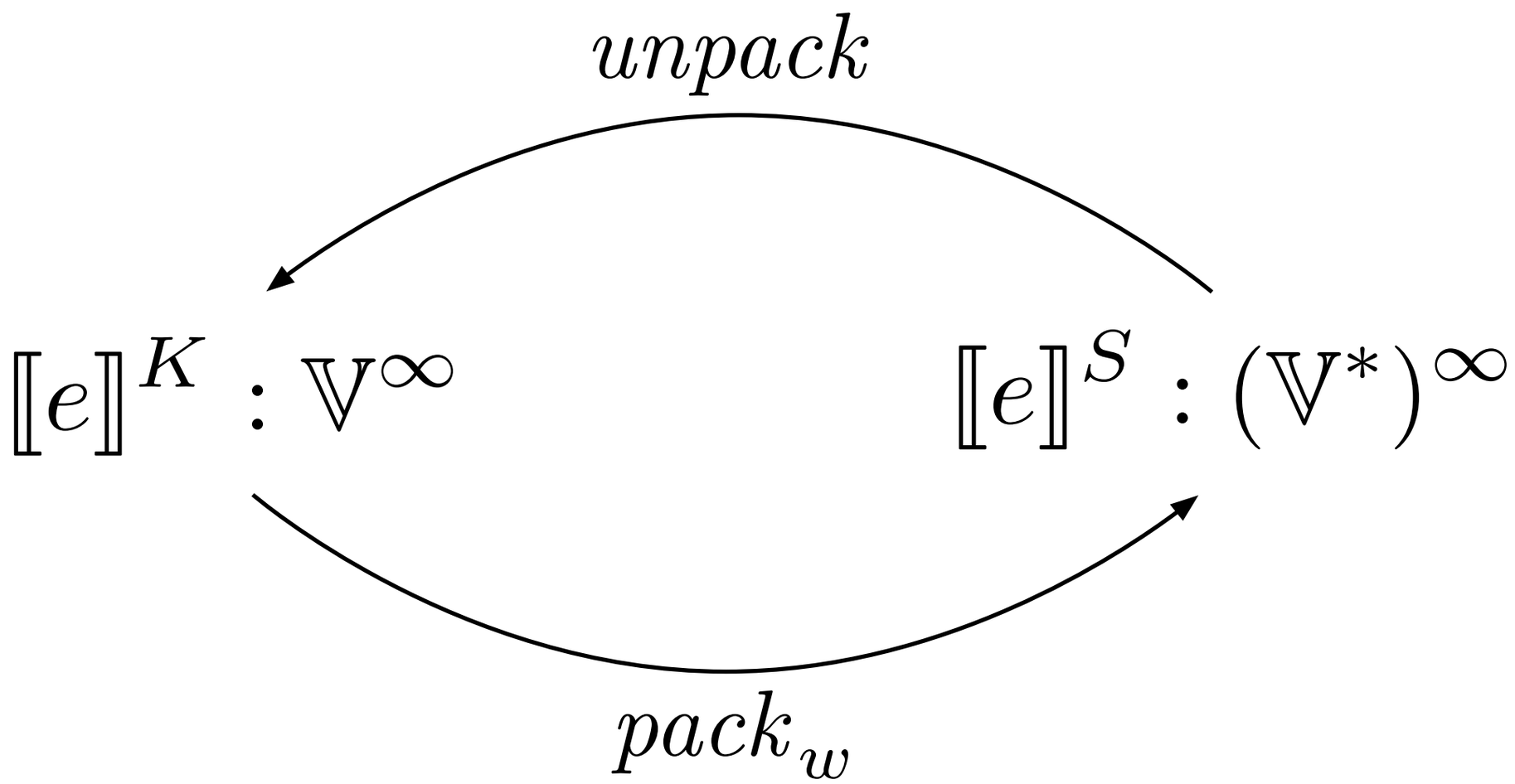
Adéquation des sémantiques

On veut $unpack\ (\llbracket \quad e \qquad \rrbracket^S) = \llbracket e \rrbracket^K$



Adéquation des sémantiques

On veut $unpack\ (\llbracket \vdash e :: ck \rrbracket^S) = \llbracket e \rrbracket^K$



Typage d'horloges

$$\frac{}{\Gamma \vdash c :: ck}$$

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck}$$

$$\ldots$$

Adéquation : les constantes

$$\frac{}{\Gamma \vdash c :: ck}$$

Une constante est bien typée si...

Adéquation : les constantes

$$\frac{}{\Gamma \vdash c :: ck}$$

Une constante est bien typée si...

$$unpack\ (\llbracket c :: ck \rrbracket^S) \quad = \quad \llbracket c \rrbracket^K$$

Adéquation : les constantes

$$\frac{}{\Gamma \vdash c :: ck}$$

Une constante est bien typée si...

$$\begin{array}{lrcl} & unpack\ (\llbracket c :: ck \rrbracket^S) & = & \llbracket c \rrbracket^K \\ \Leftrightarrow & unpack\ (pack_{\llbracket ck \rrbracket}\ c^\omega) & = & c^\omega \end{array}$$

Adéquation : les constantes

$$\frac{}{\Gamma \vdash c :: ck}$$

Une constante est bien typée si...

$$\begin{array}{lrcl} & unpack\ (\llbracket c :: ck \rrbracket^S) & = & \llbracket c \rrbracket^K \\ \Leftrightarrow & unpack\ (pack_{\llbracket ck \rrbracket}\ c^\omega) & = & c^\omega \\ \Leftrightarrow & . & & \end{array}$$

Adéquation : les opérateurs

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck}$$

Une addition est bien typée si...

Adéquation : les opérateurs

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck}$$

Une addition est bien typée si...

$$unpack\ (map2\ F\ x\ y) \quad = \quad F\ (unpack\ x)\ (unpack\ y)$$

$$\text{avec} \begin{cases} F & = & map2\ (+) \\ x & = & [\![e_1]\!]^S \\ y & = & [\![e_2]\!]^S \end{cases}$$

Adéquation : les opérateurs

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck}$$

Une addition est bien typée si...

$$\begin{aligned} \mathit{unpack}\ (\mathit{map2}\ F\ x\ y) &= F\ (\mathit{unpack}\ x)\ (\mathit{unpack}\ y) \\ &\text{avec} \begin{cases} F &= \mathit{map2}\ (+) \\ x &= [\![e_1]\!]^S \\ y &= [\![e_2]\!]^S \end{cases} \\ \Leftarrow \quad \forall i, F\ (x_i.x_{i+1})\ (y_i.y_{i+1}) &= (F\ x_i\ y_i).(F\ x_{i+1}\ y_{i+1}) \end{aligned}$$

Adéquation : les opérateurs

$$\frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck}$$

Une addition est bien typée si...

$$\begin{array}{rrcl}
 & \mathit{unpack}\ (\mathit{map2}\ F\ x\ y) & = & F\ (\mathit{unpack}\ x)\ (\mathit{unpack}\ y) \\
 & & \text{avec} & \begin{cases} F & = & \mathit{map2}\ (+) \\ x & = & [\![e_1]\!]^S \\ y & = & [\![e_2]\!]^S \end{cases} \\
\Leftarrow & \forall i, F\ (x_i.x_{i+1})\ (y_i.y_{i+1}) & = & (F\ x_i\ y_i).(F\ x_{i+1}\ y_{i+1}) \\
\Leftarrow & \forall i, |x_i| & = & |y_i|
\end{array}$$

Typage d'horloges

$$\frac{}{\Gamma \vdash c :: ck} \qquad \frac{\Gamma \vdash e_1 :: ck \qquad \Gamma \vdash e_2 :: ck}{\Gamma \vdash e_1 + e_2 :: ck}$$

$$\frac{\Gamma \vdash e :: ck \qquad \Gamma \vdash ce :: ck}{\Gamma \vdash e \texttt{ when } ce :: ck \texttt{ on } ce} \qquad \frac{\Gamma \vdash e :: ck \qquad ck <: ck'}{\Gamma \vdash \texttt{buffer } e :: ck'}$$

...

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

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

Correction du typage d'horloges

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

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

Plan

Introduction

Le langage

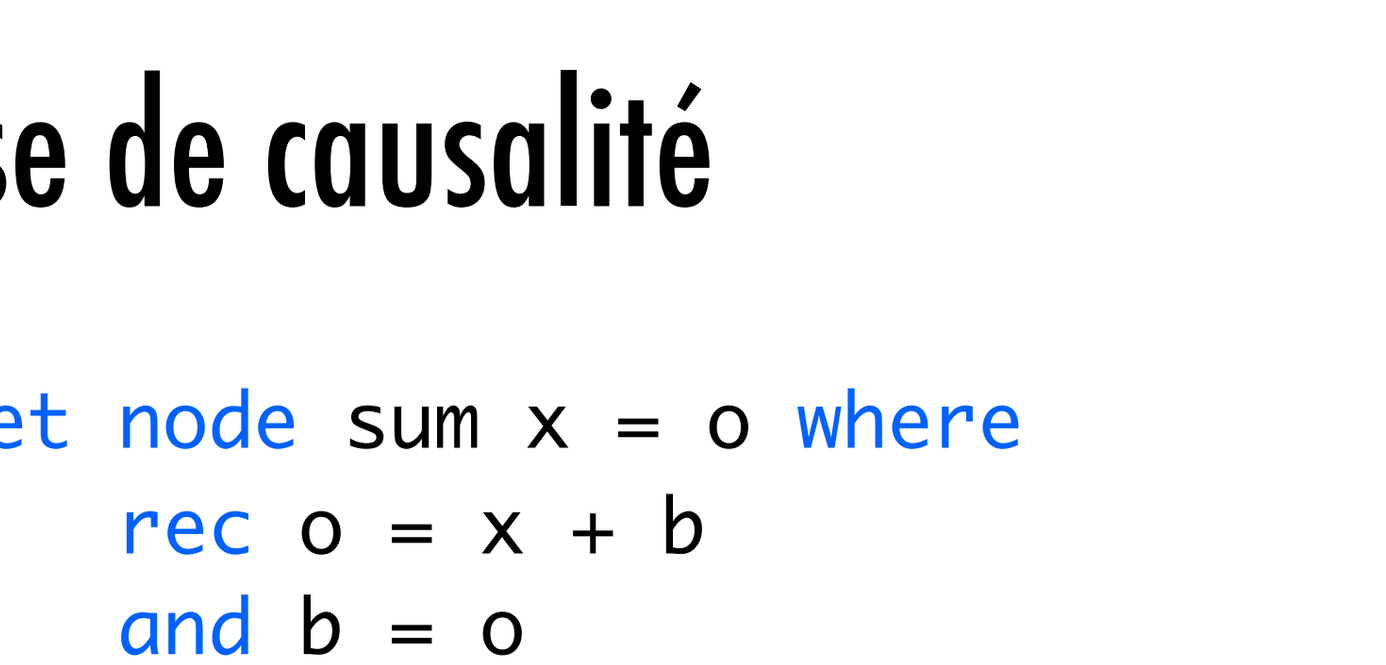
Causalité

Conclusion

Analyse de causalité

```
let node sum x = o where
    rec o = x + b
    and b = o
```

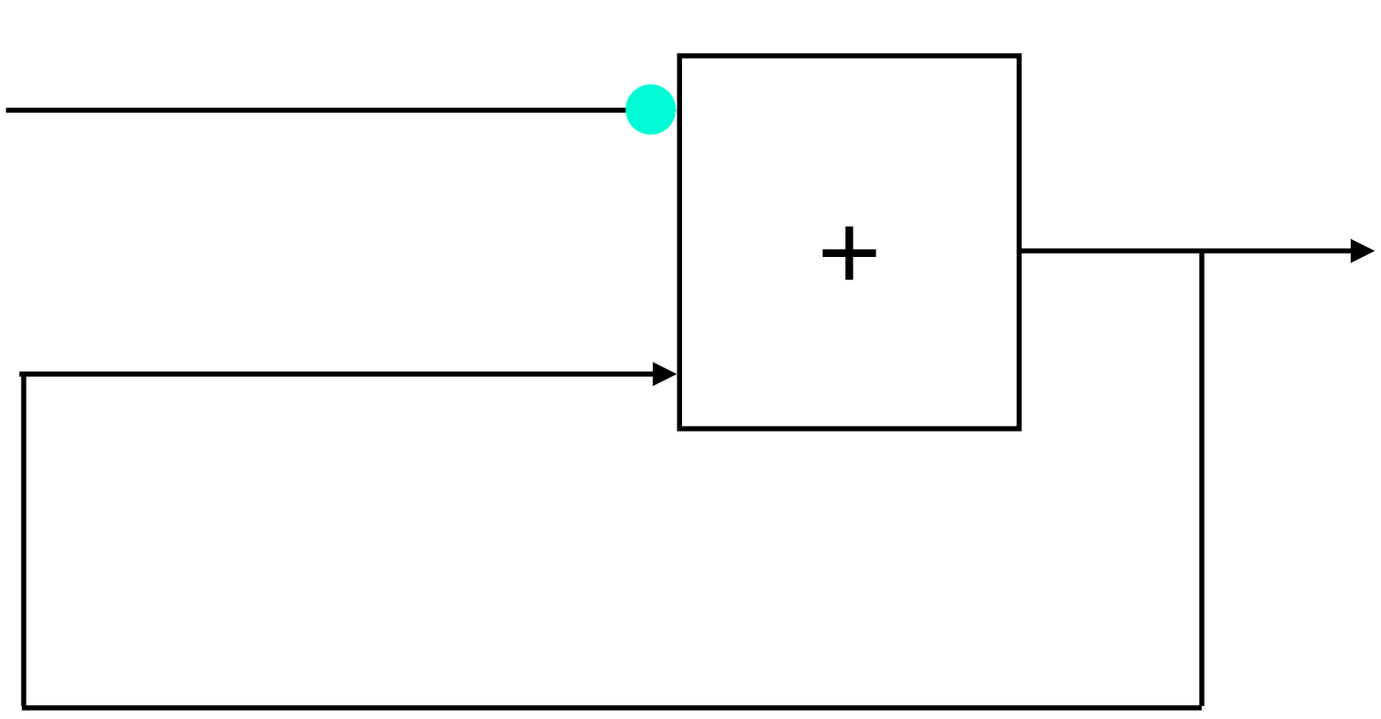
```
sum :: 'a -> 'a
```



Analyse de causalité

```
let node sum x = o where
    rec o = x + b
    and b = o
```

```
sum :: 'a -> 'a
```

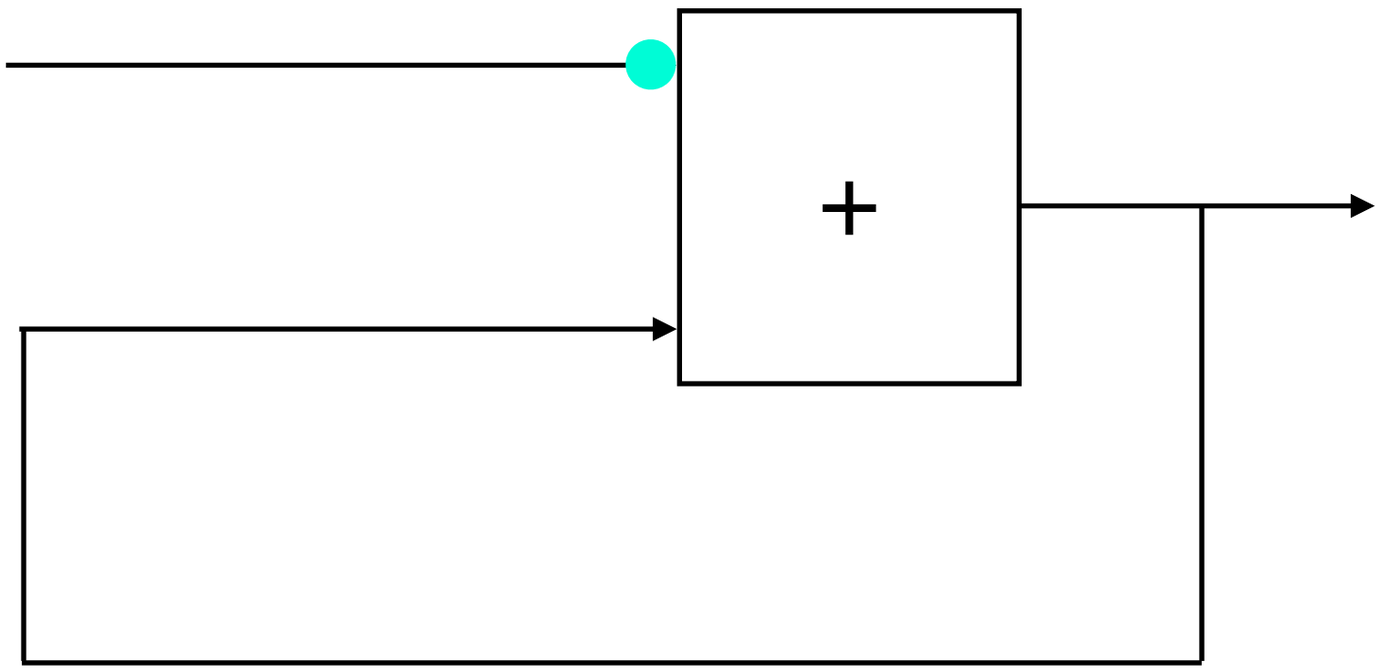


Analyse de causalité

Rejeté

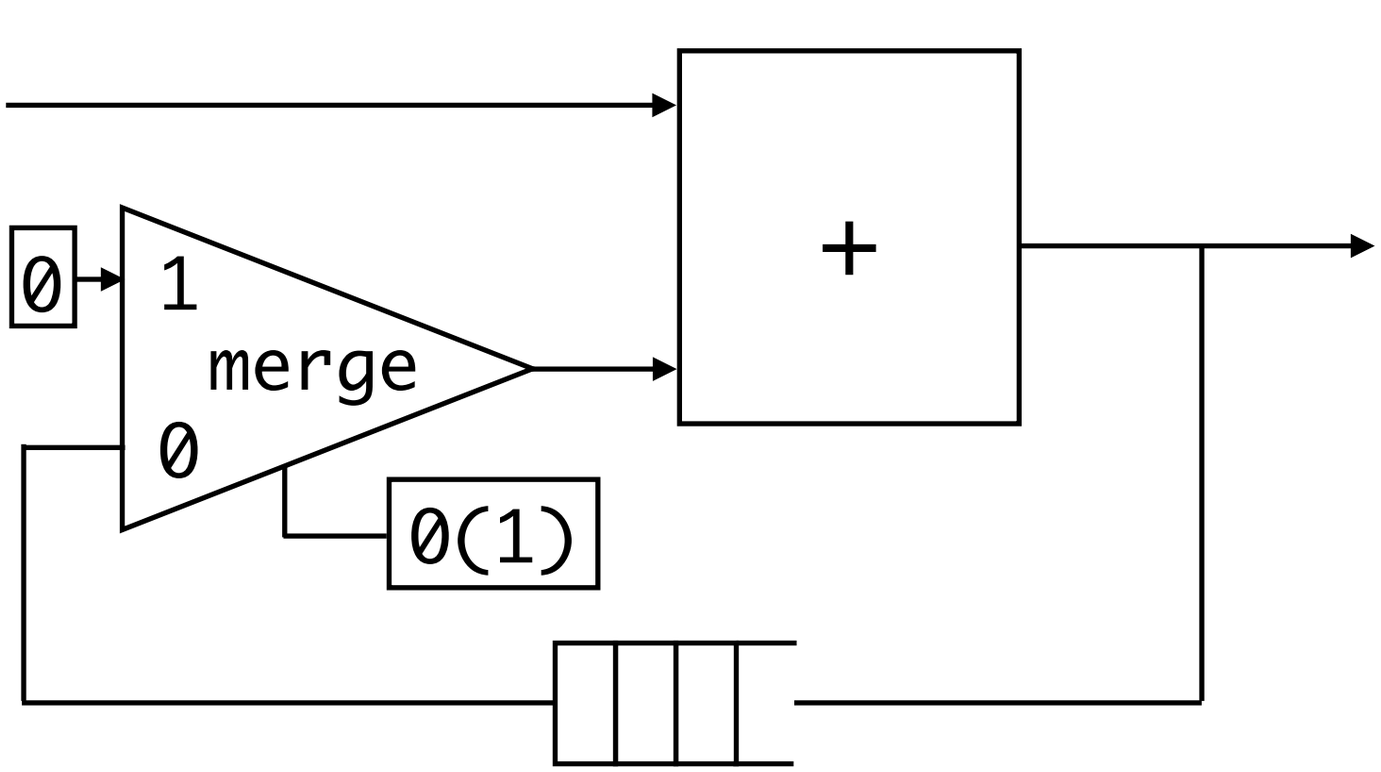
```
sum x = o where
  b = x + b
  o = o
```

```
sum :: 'a -> 'a
```



Analyse de causalité

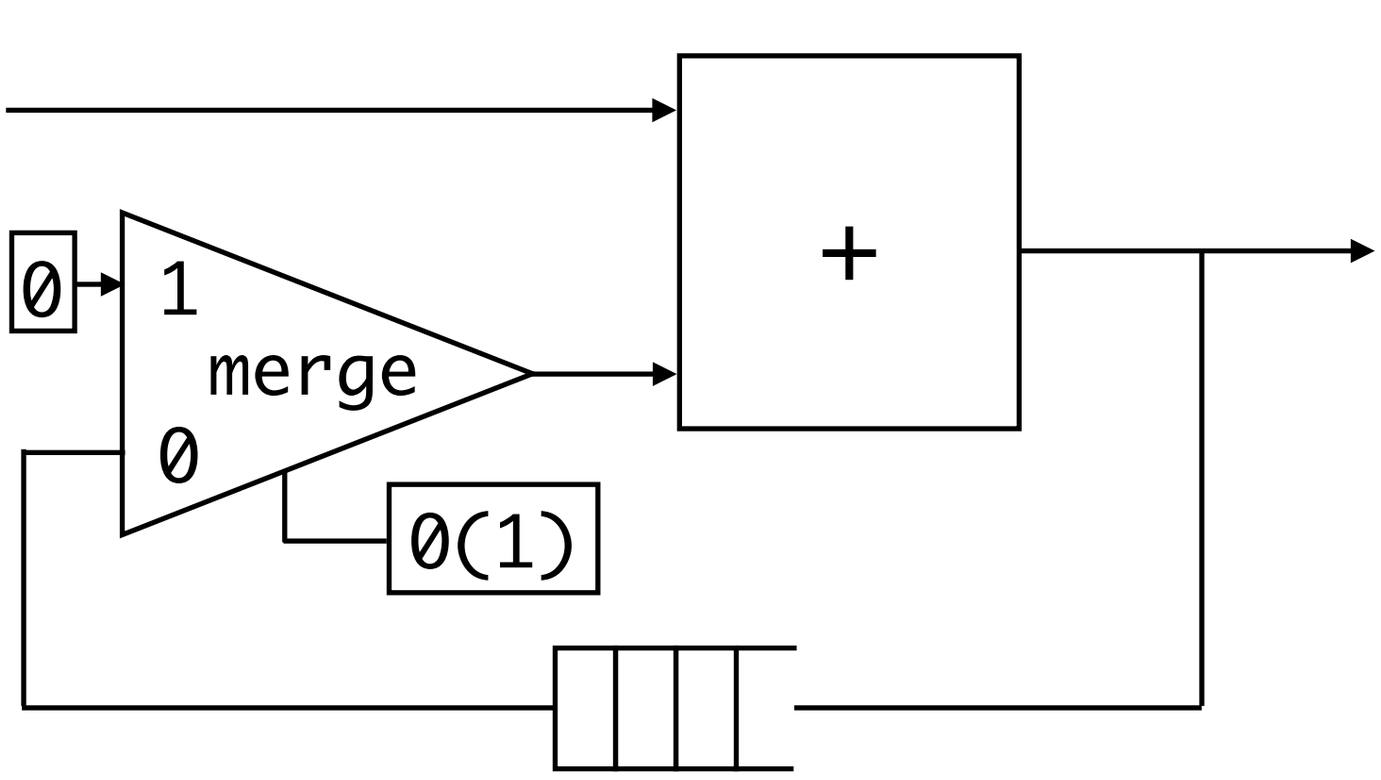
```
let node sum x = o where
    rec o = merge true(false) 0 b
    and b = buffer (x + o)
```



Analyse de causalité

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

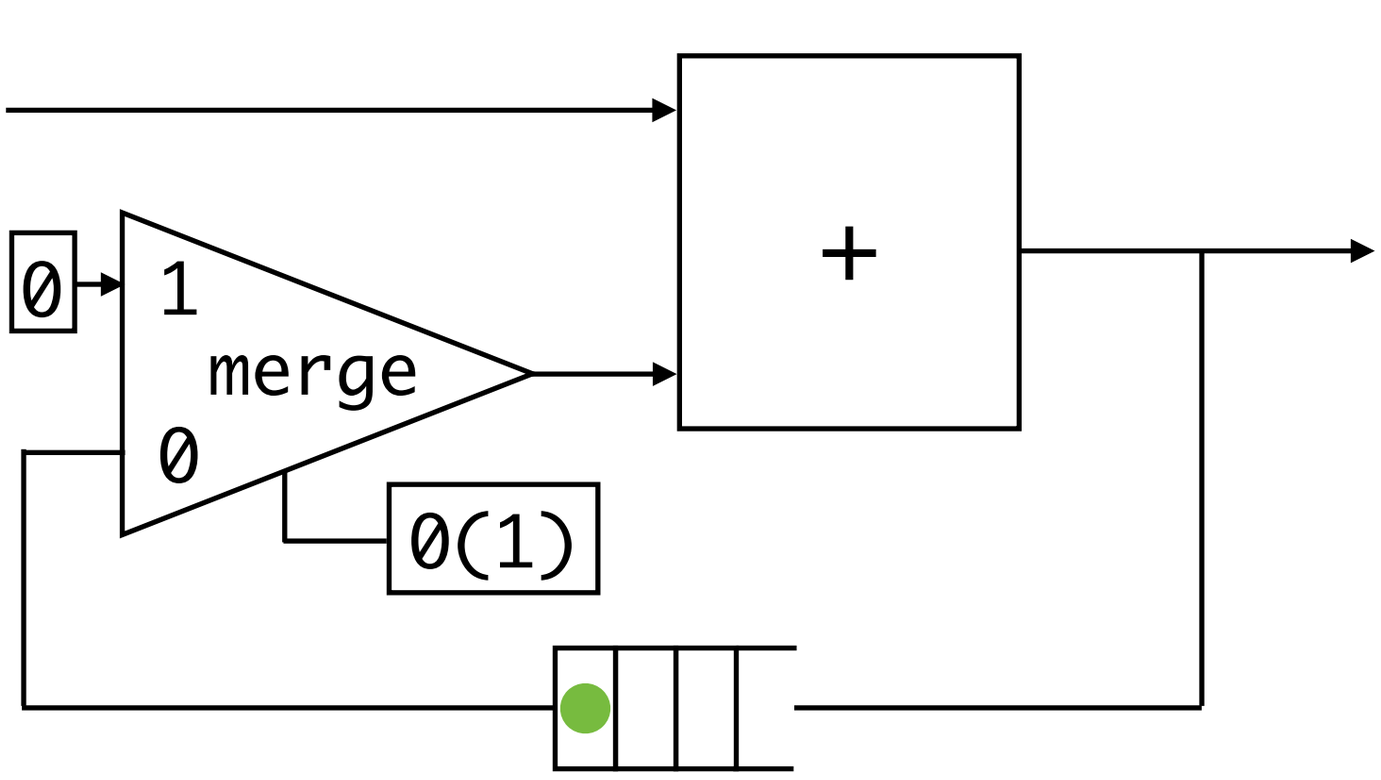
```
sum :: 'a -> 'a
```



Analyse de causalité

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

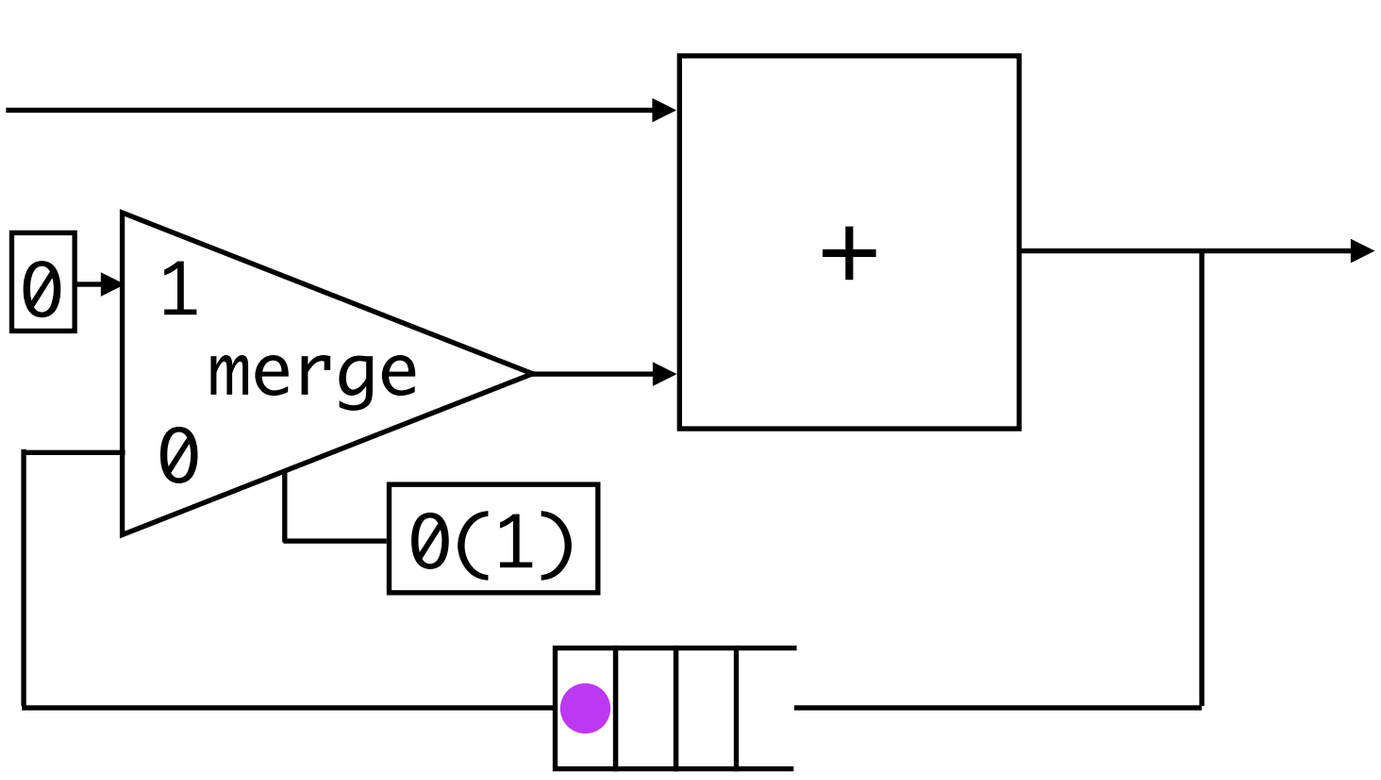
```
sum :: 'a -> 'a
```



Analyse de causalité

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

```
sum :: 'a -> 'a
```

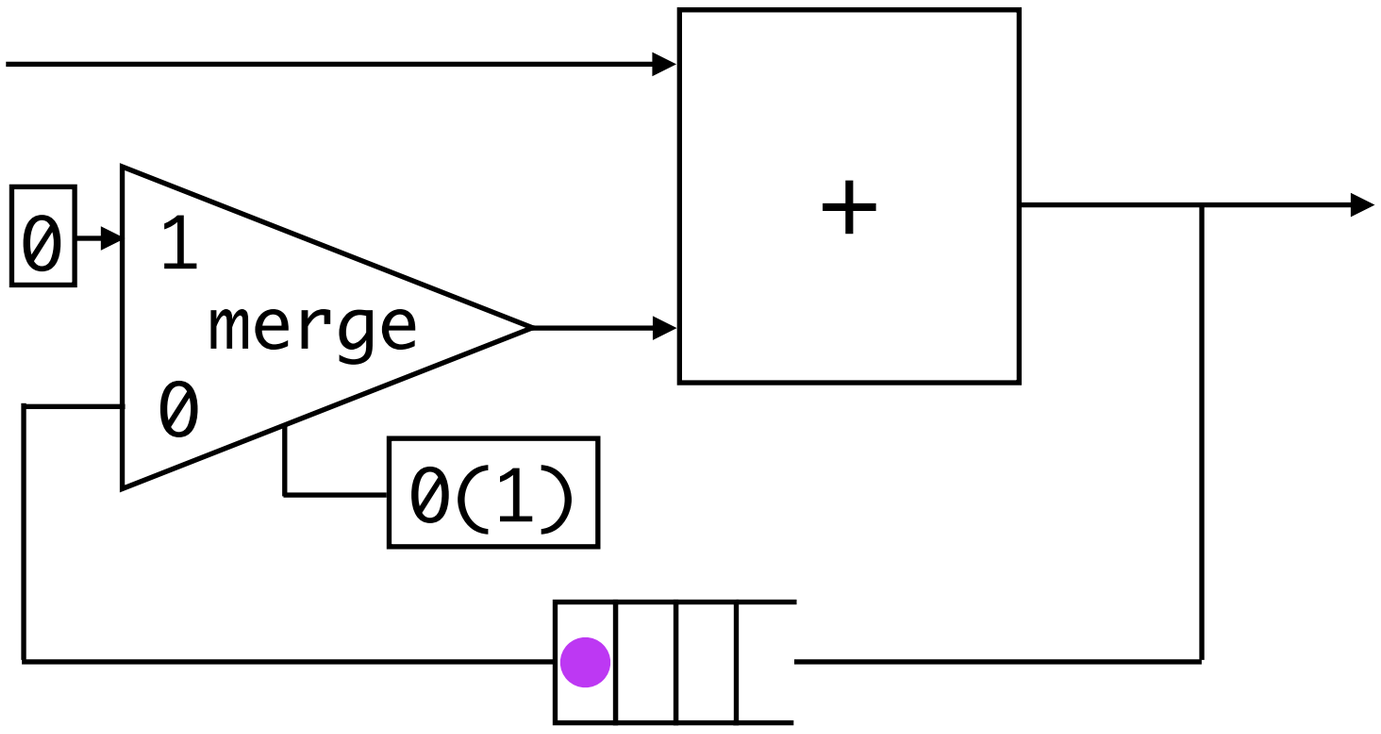


Analyse de causalité

```
let node sum x =
    rec o = merge true(false) 0 b
    and b = buffer (x + o)
```

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

```
sum :: 'a -> 'a
```



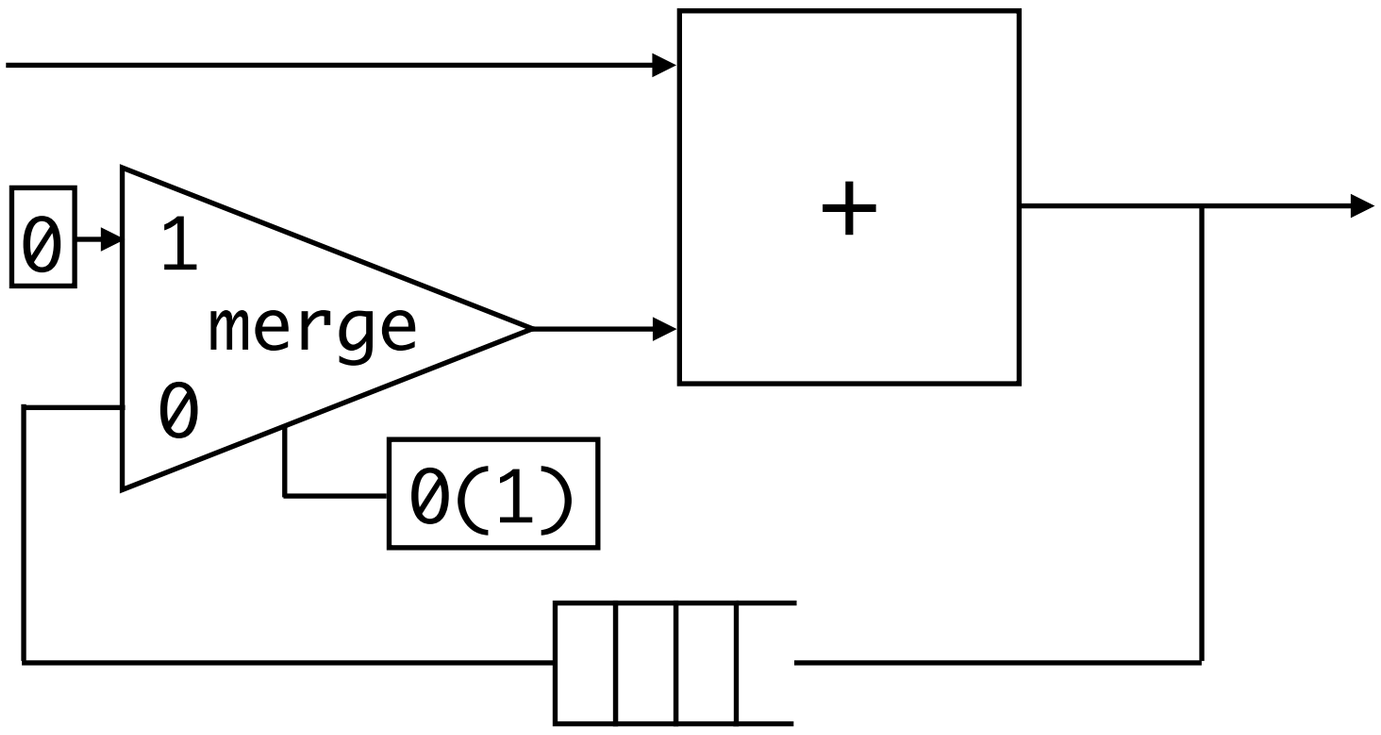
Analyse de causalité

Accepté

```
sum x =
    = merge true(false) 0 b
    = buffer (x + o)
```

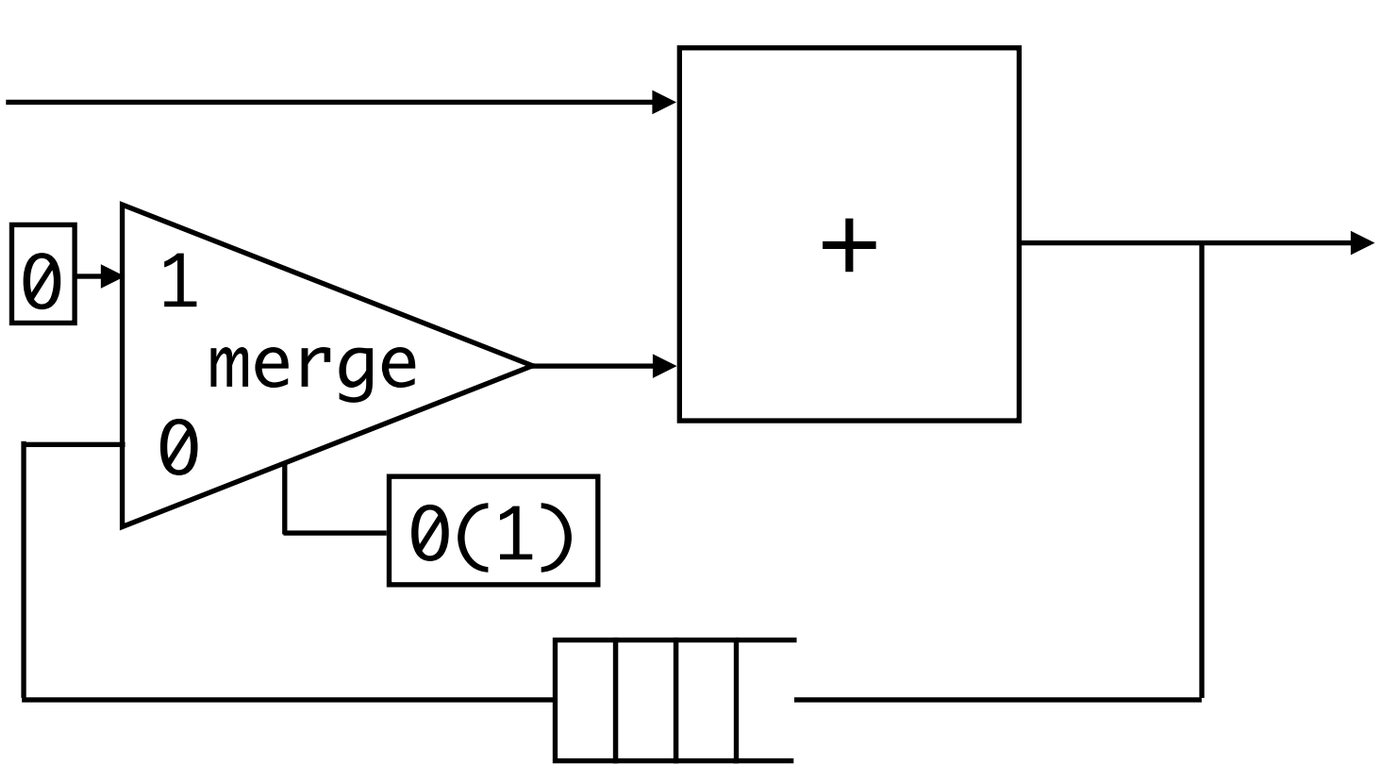
$(1) \ll : 0(1)$

```
sum :: 'a -> 'a
```



Analyse de causalité

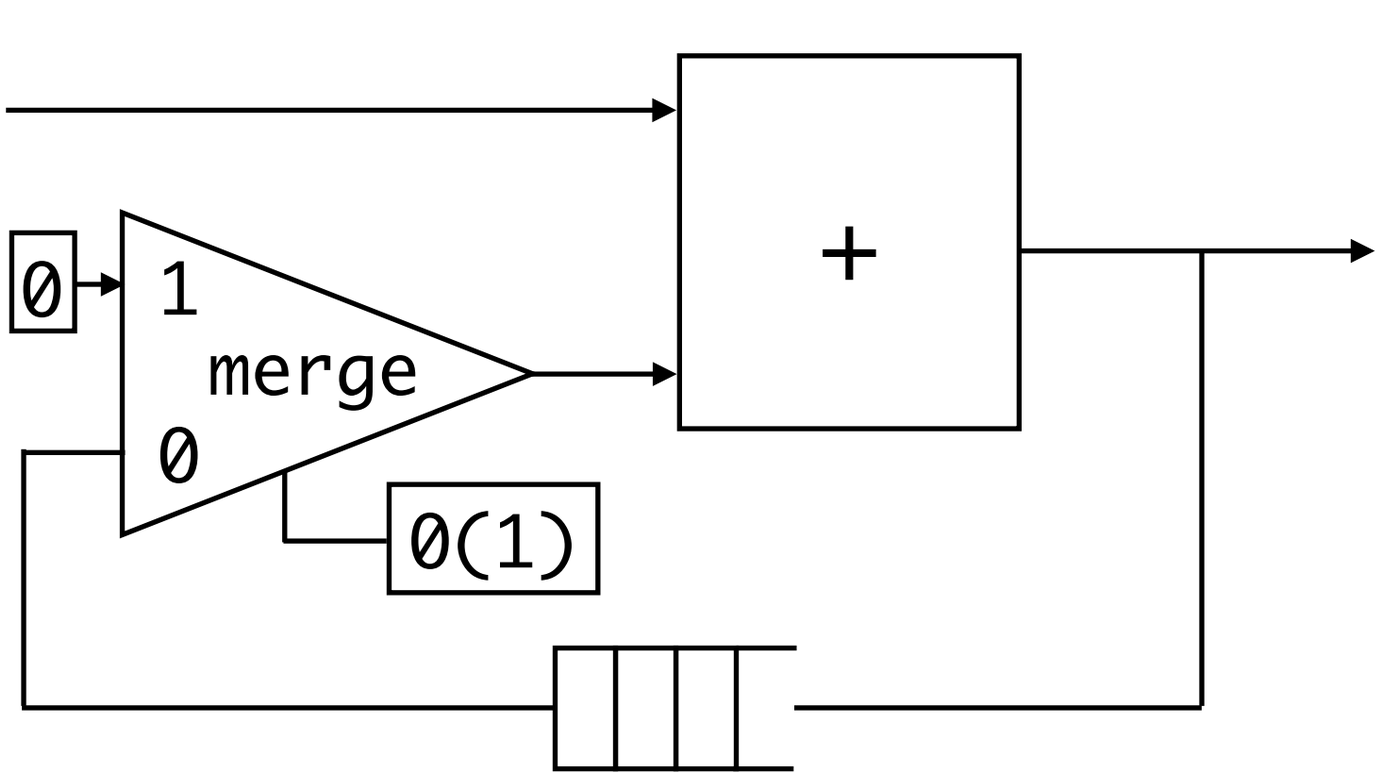
```
let node sum x = o where
    rec o = merge true(false) 0 b
    and b = buffer (x + o)
```



Analyse de causalité

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

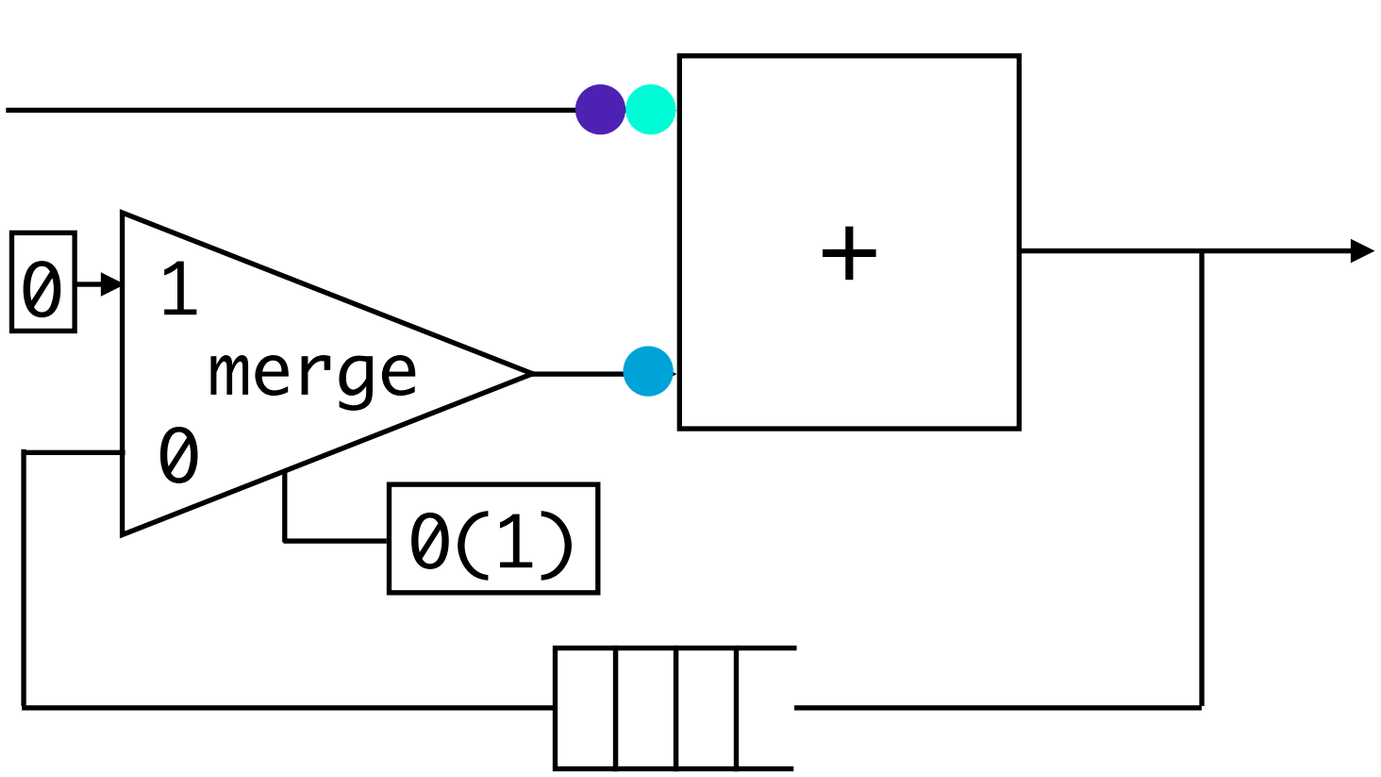
```
sum :: 'a on (2) -> 'a on (2)
```



Analyse de causalité

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

```
sum :: 'a on (2) -> 'a on (2)
```

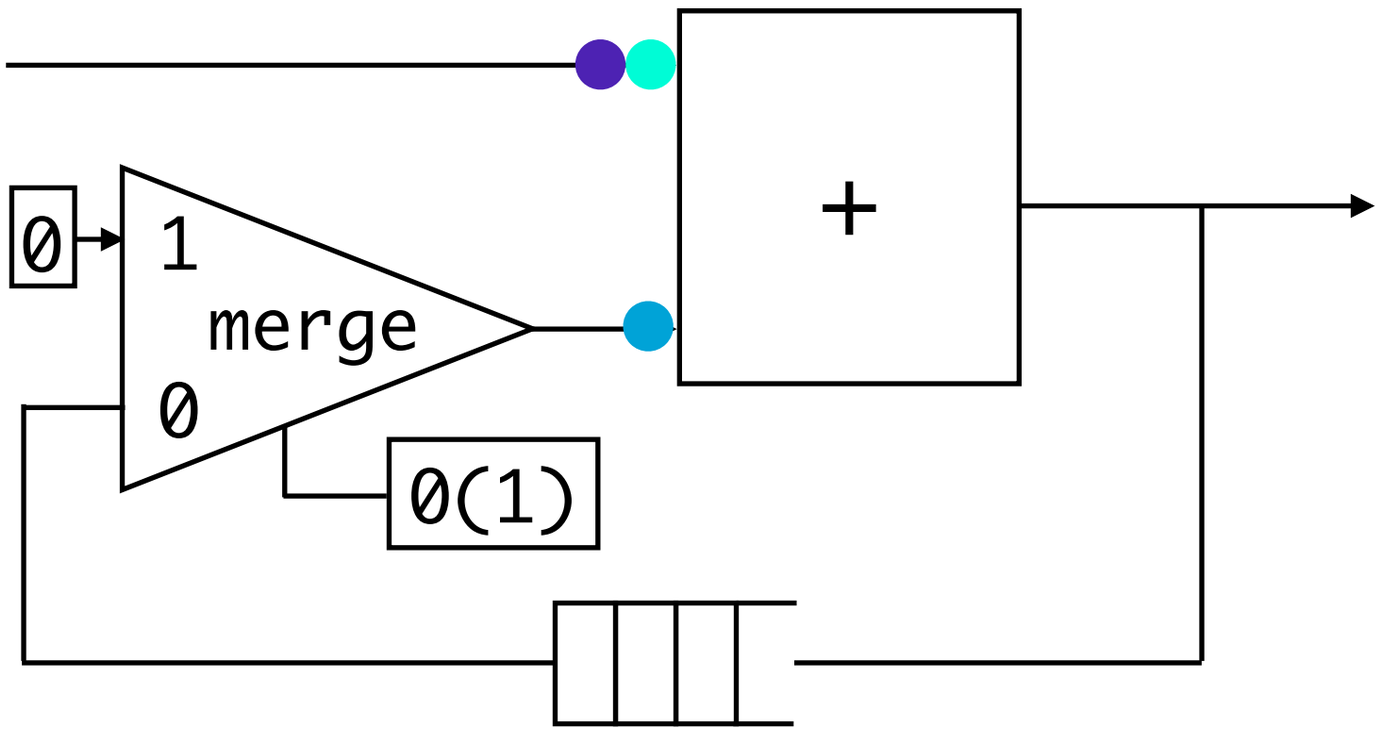


Analyse de causalité

```
let node sum x =
    rec o = merge true(false) 0 b
    and b = buffer (x + o)
```

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

```
sum :: 'a on (2) -> 'a on (2)
```



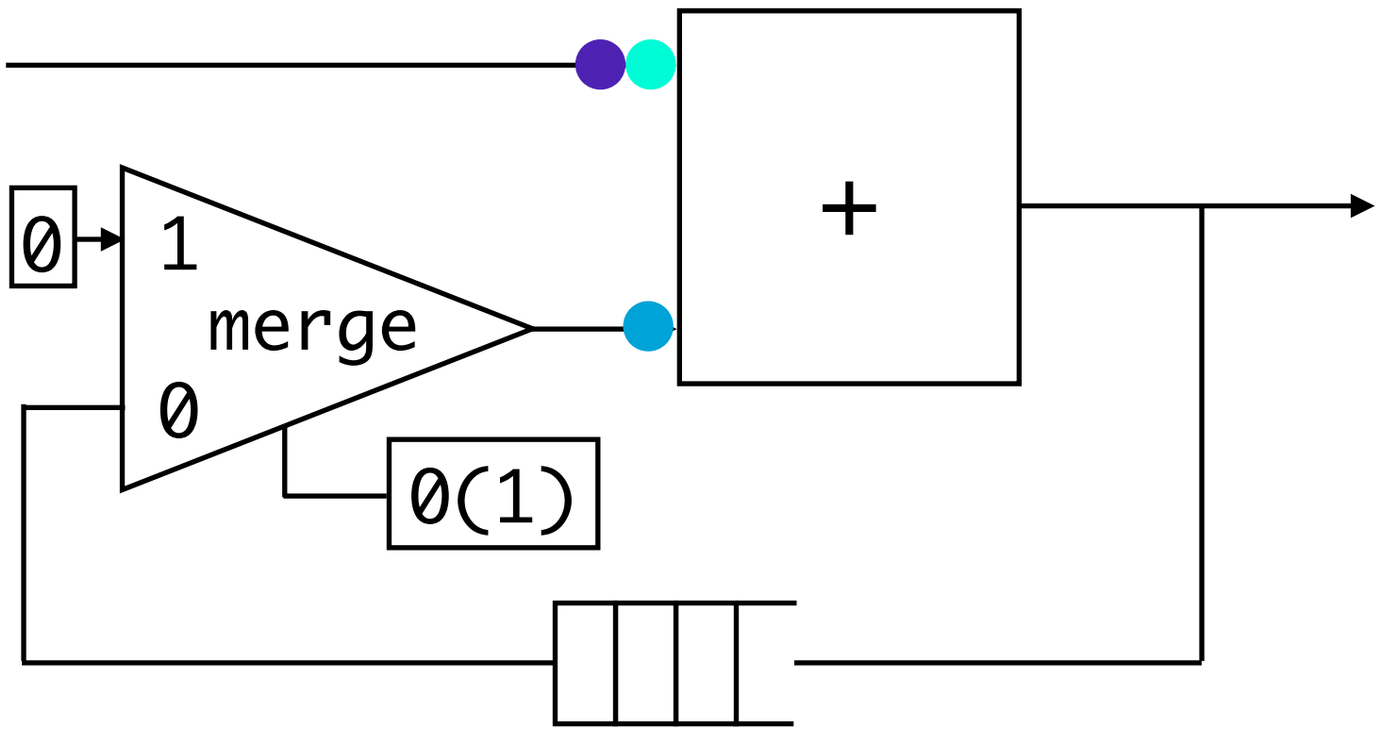
Analyse de causalité

```
sum x = let rec b = merge true(false) 0 b
        in b = buffer (x + o)
```

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

Rejeté

```
sum :: 'a on (2) -> 'a on (2)
```



Plan

Introduction

Le langage

Causalité

Conclusion

Conclusion

- Un langage fonctionnel avec du temps flexible
- Un système de type pour :
  - vérifier un ordonnancement statique
  - calculer la taille des buffers
- Des problèmes de causalité nouveaux
- Travaux connexes : déforestation, stream fusion...

Conclusion

Merci !

- Un langage fonctionnel avec du temps flexible
- Un système de type pour :
  - vérifier un ordonnancement statique
  - calculer la taille des buffers
- Des problèmes de causalité nouveaux
- Travaux connexes : déforestation, stream fusion...