Integer Clocks: Thin Buffers and Thick Wires

Adrien Guatto
Joint work with A. Cohen, L. Gérard, L. Mandel and M. Pouzet

PARKAS team, DIENS & INRIA

SYNCHRON 2012

What this is about

Programming streaming applications with:

- high developper productivity
- efficient execution on a parallel machine
- strong safety guarantees
- modular compilation

The path taken

We'll embrace and extend n-synchrony and Lucy-n.

Outline

Outline

Lucy-n

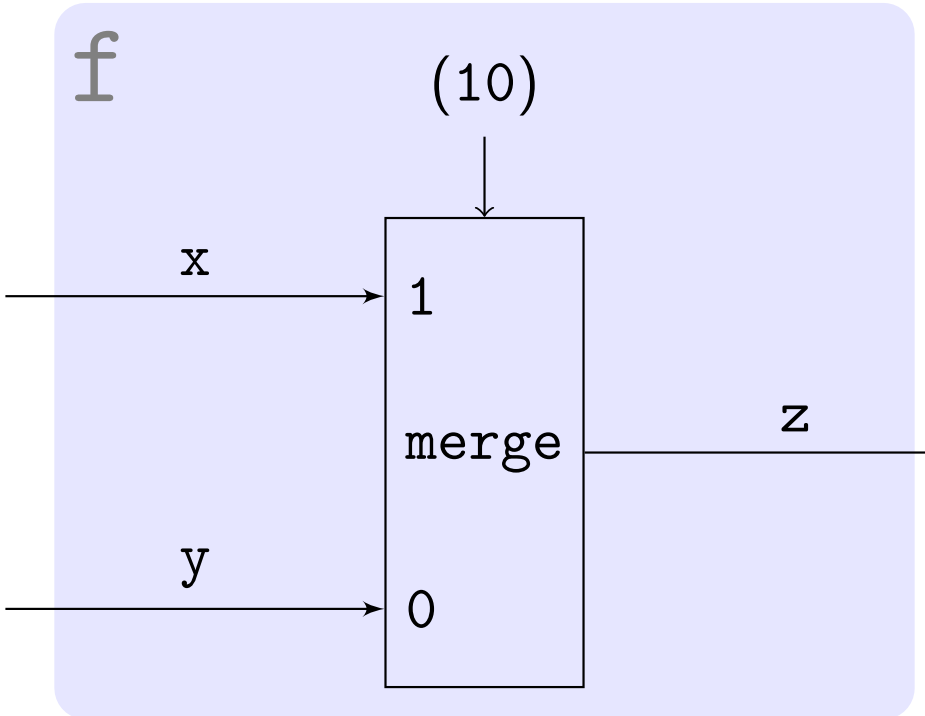
The language [MPP10]

- minimalistic data-flow synchronous language
- inspired by Lustre and Lucid Synchrone
- restricted to ultimately periodic sampling/fusion conditions
- accepts Kahn networks with bounded buffers

The compiler

- computes sufficient buffer sizes through *clock inference*
- generate code

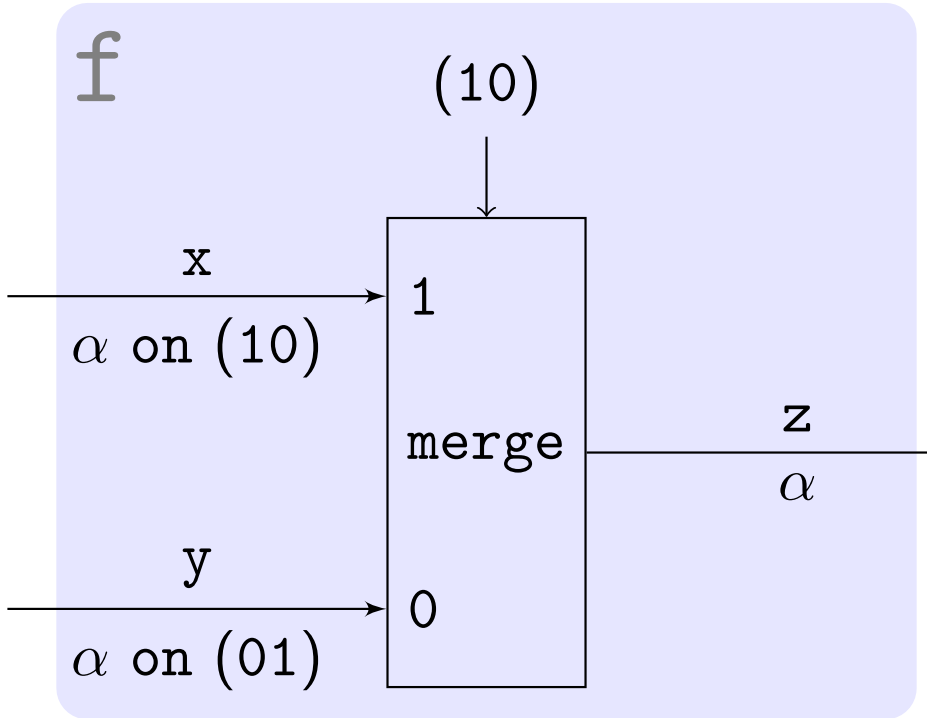
My First Lucy-n Program™




```
let node f (x, y) = z
  where
    rec z =
      merge (10) x y
```

x	x_0	.	x_1	.	x_2	.	x_3	.	...
y	.	y_0	.	y_1	.	y_2	.	y_3	...
z	x_0	y_0	x_1	y_1	x_2	y_2	x_3	y_3	...

My First Lucy-n Program™

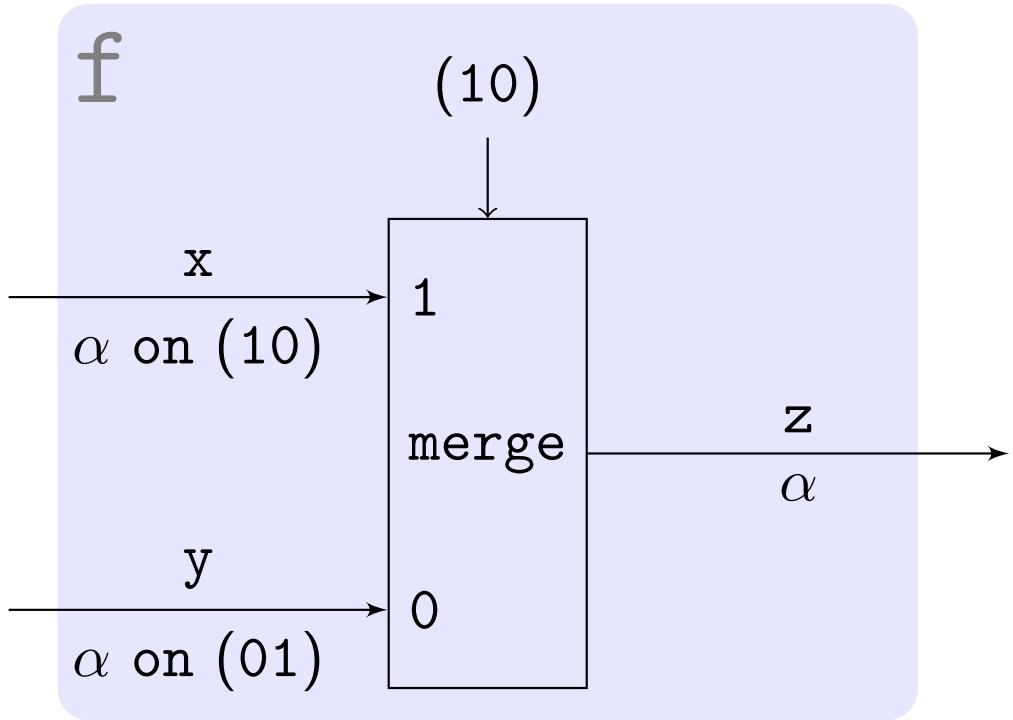



```
let node f (x, y) = z
  where
    rec z =
      merge (10) x y
```

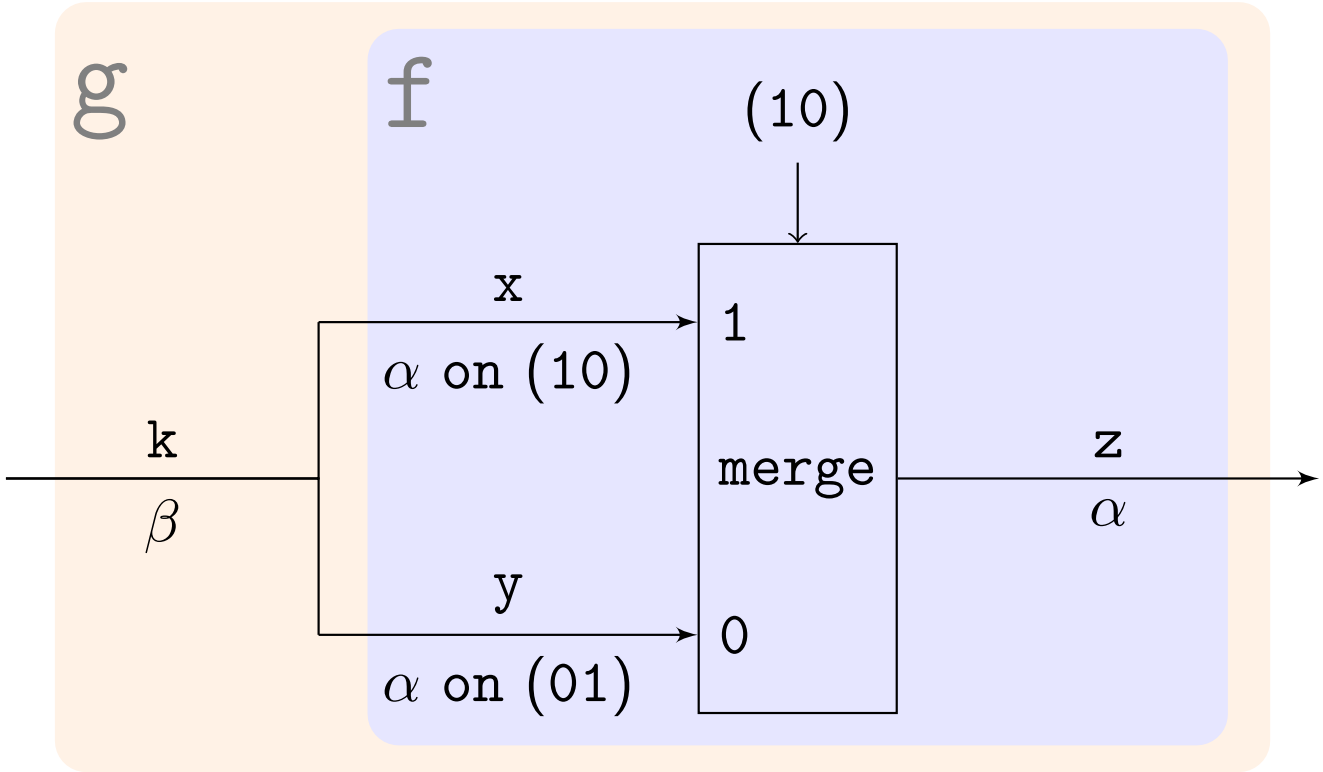
x	x_0	.	x_1	.	x_2	.	x_3	.	...
y	.	y_0	.	y_1	.	y_2	.	y_3	...
z	x_0	y_0	x_1	y_1	x_2	y_2	x_3	y_3	...

```
val f :: forall 'a. ('a on (10) * 'a on (01)) -> 'a
```

Stutter: synchronous composition



Stutter: synchronous composition

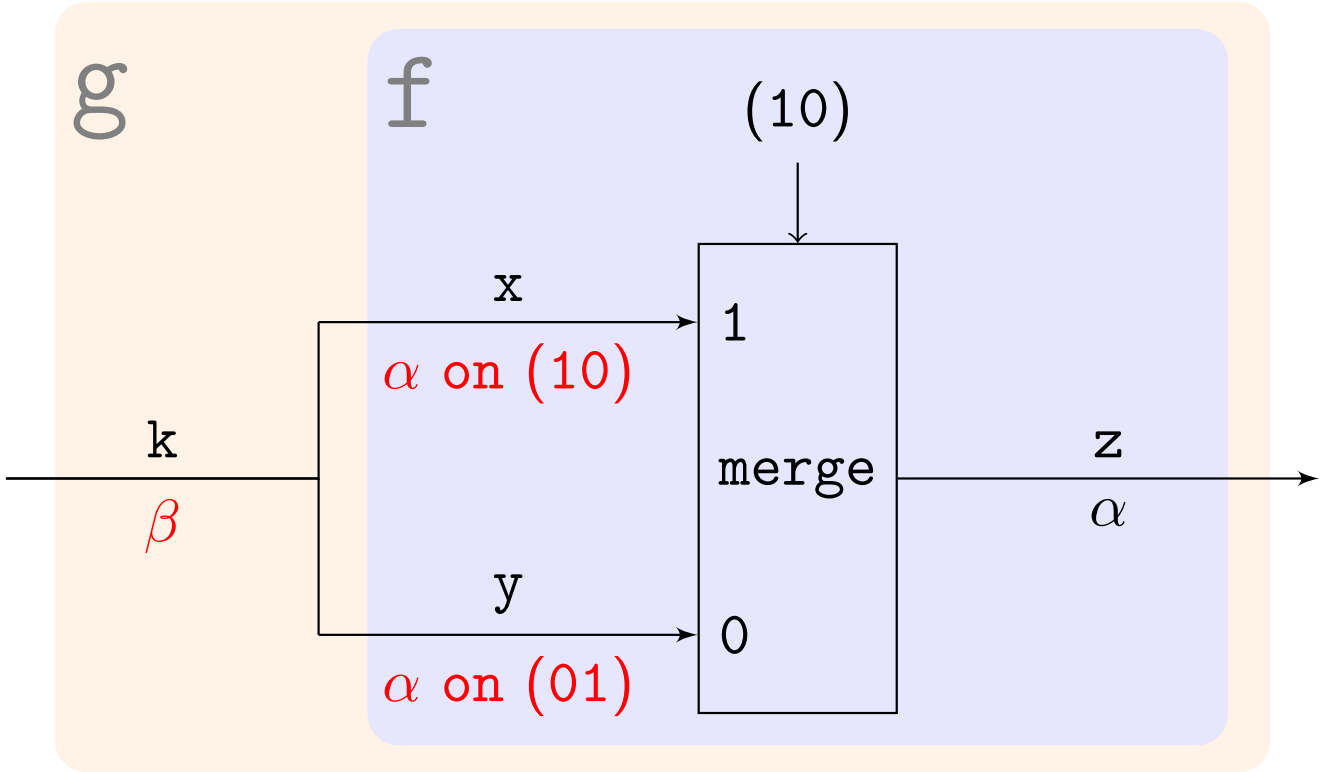



```
let node g k = f (k, k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{B}^{\omega} \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \equiv \alpha \text{ on } (01)$$

Stutter: synchronous composition



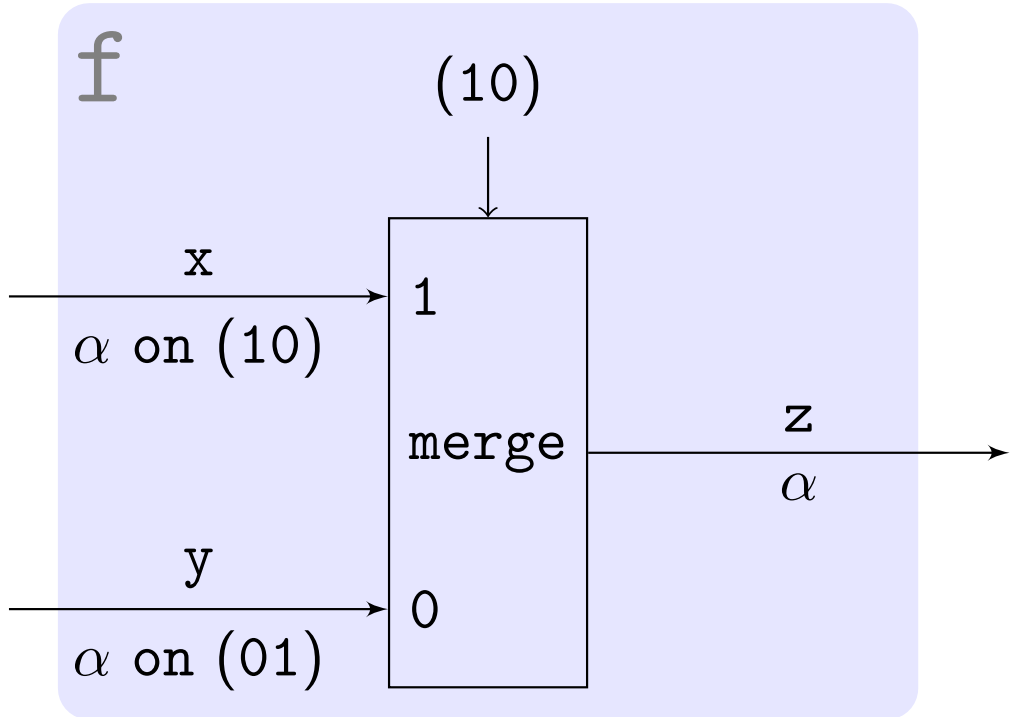

```
let node g k = f (k, k)
```

The program is well-clocked iff:

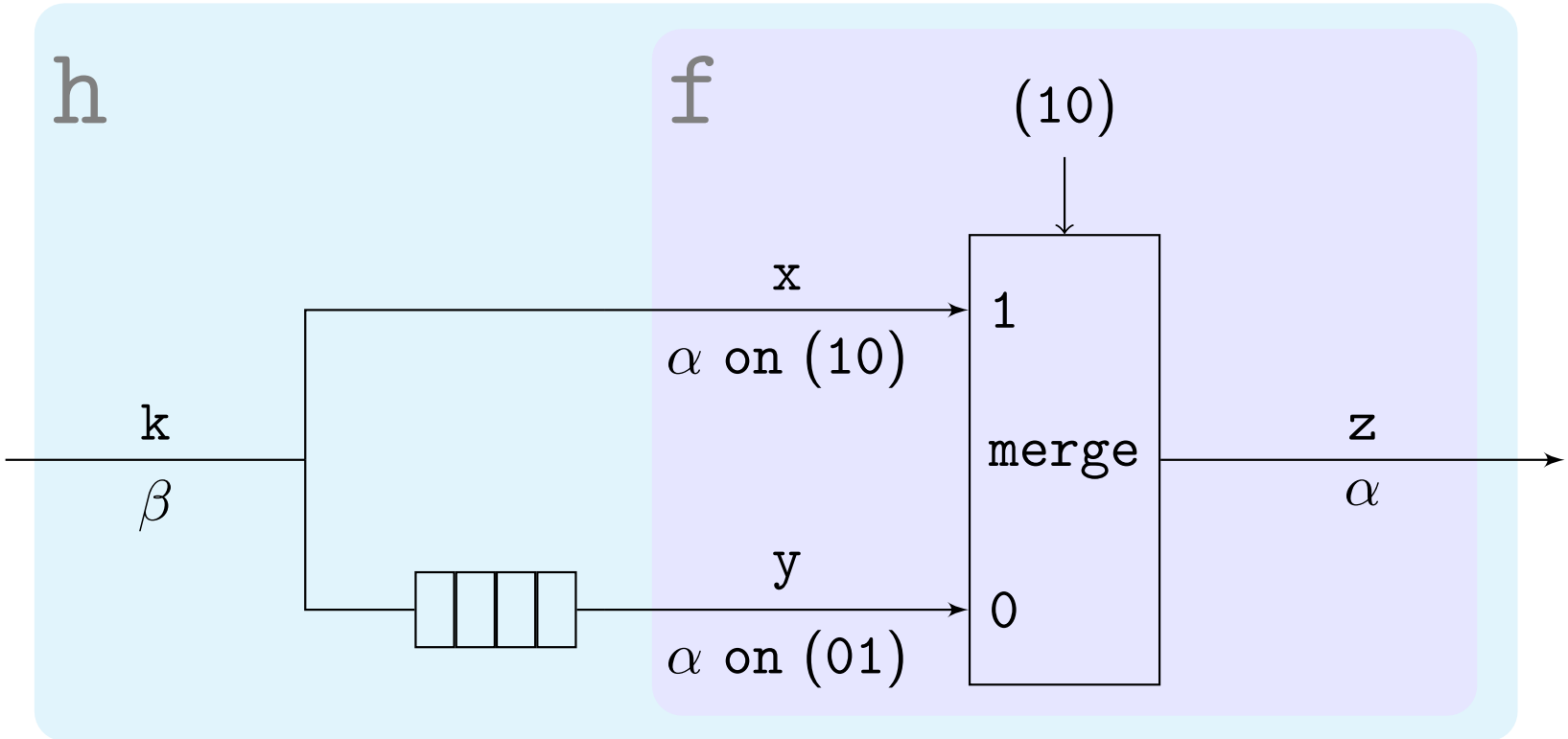
$$\exists \alpha, \beta \in \mathbb{B}^{\omega} \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \equiv \alpha \text{ on } (01)$$

Rejected: no α with equal sets of odd and even instants!

Stutter: n-synchronous composition



Stutter: n-synchronous composition

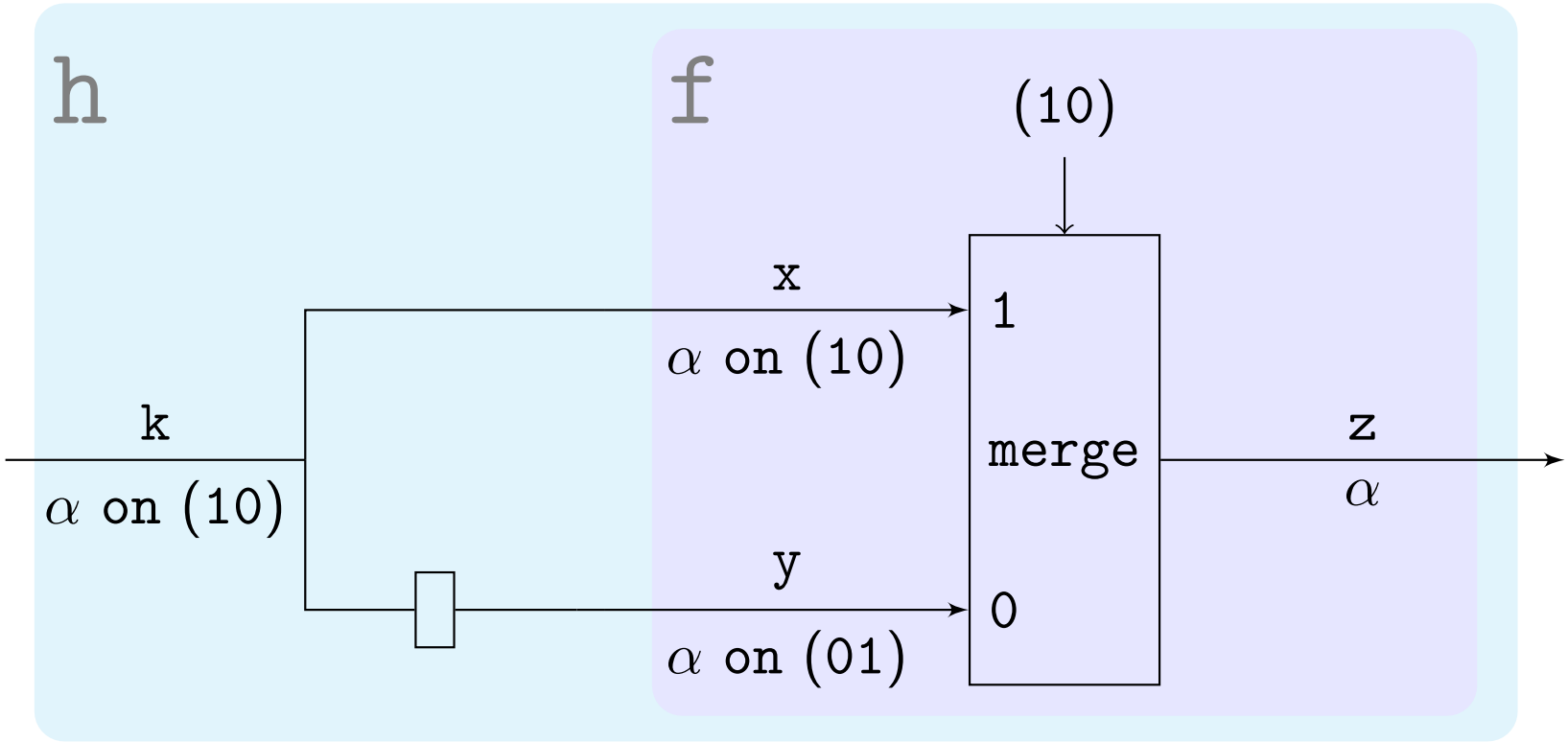



```
let node h k = f (k, buffer k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{B}^{\omega} \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \text{ and } \beta <: \alpha \text{ on } (01)$$

Stutter: n-synchronous composition




```
let node h k = f (k, buffer k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{B}^\omega \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \text{ and } \beta <: \alpha \text{ on } (01)$$

One solution: any α, $\beta = \alpha$ on (10)

Outline

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)										
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0									
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.								
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1							
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1	0						
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1	0	.					
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1	0	.	1				
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1	0	.	1	0			
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1	0	.	1	0	.		
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
0(10)	0	.	1	0	.	1	0	.	1	...
(101) on 0(10)										

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
$0(10)$	0	.	1	0	.	1	0	.	1	...
(101) on $0(10)$	0	0	1	0	0	1	0	0	1	...

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
$0(10)$	0	.	1	0	.	1	0	.	1	...
(101) on $0(10) = (001)$	0	0	1	0	0	1	0	0	1	...

Boolean clocks 101

$$\mathbb{B}^\omega \triangleq \mathbb{N} \to \mathbb{B}$$

$$\begin{array}{lllll} (0.w_1) & \text{on} & w_2 & = & 0.(w_1 \text{ on } w_2) \\ (1.w_1) & \text{on} & (b.w_2) & = & b.(w_1 \text{ on } w_2) \end{array}$$

Example:

(101)	1	0	1	1	0	1	1	0	1	...
$0(10)$	0	.	1	0	.	1	0	.	1	...
(101) on $0(10) = (001)$	0	0	1	0	0	1	0	0	1	...

Another one:

(101)	1	0	1	1	0	1	1	0	1	...
(011)	0	.	1	1	.	0	1	.	1	...
(101) on $(011) = (001100101)$	0	0	1	1	0	0	1	0	1	...

Boolean clocks and buffers

Cumulative function $\mathcal{O} : \mathbb{B}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} \mathcal{O}_w(0) & = & 0 \\ \mathcal{O}_{0.w}(i) & = & \mathcal{O}_w(i-1) \\ \mathcal{O}_{1.w}(i) & = & 1 + \mathcal{O}_w(i-1) \end{array}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Boolean clocks and buffers

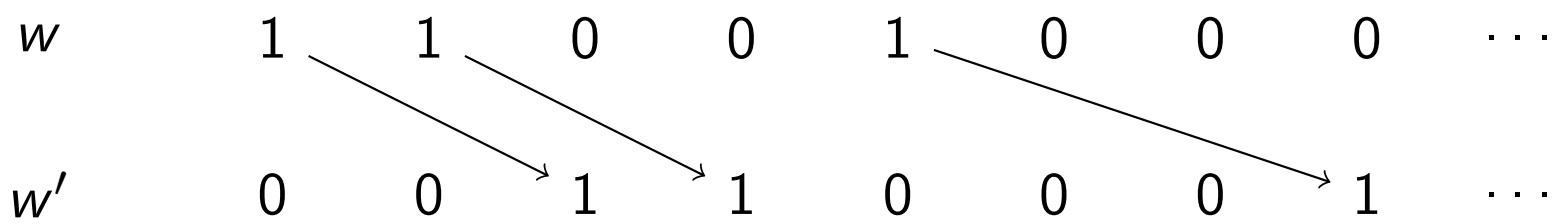
Cumulative function $\mathcal{O} : \mathbb{B}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} \mathcal{O}_w(0) & = & 0 \\ \mathcal{O}_{0.w}(i) & = & \mathcal{O}_w(i-1) \\ \mathcal{O}_{1.w}(i) & = & 1 + \mathcal{O}_w(i-1) \end{array}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Example: $(11001000) <: (00110001)$



Boolean clocks and buffers

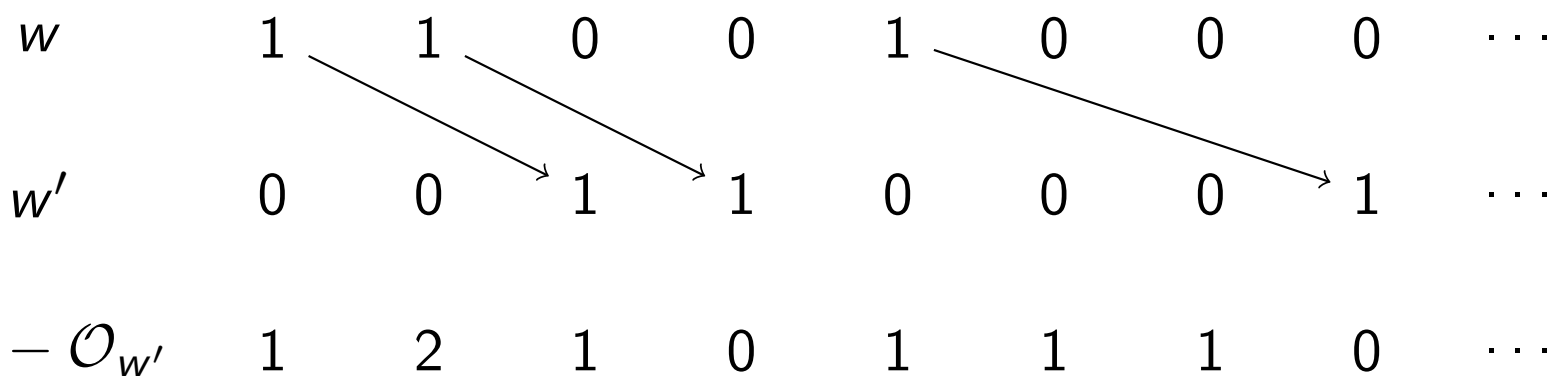
Cumulative function $\mathcal{O} : \mathbb{B}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} \mathcal{O}_w(0) & = & 0 \\ \mathcal{O}_{0.w}(i) & = & \mathcal{O}_w(i-1) \\ \mathcal{O}_{1.w}(i) & = & 1 + \mathcal{O}_w(i-1) \end{array}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Example: $(11001000) <: (00110001)$



Boolean clocks and buffers

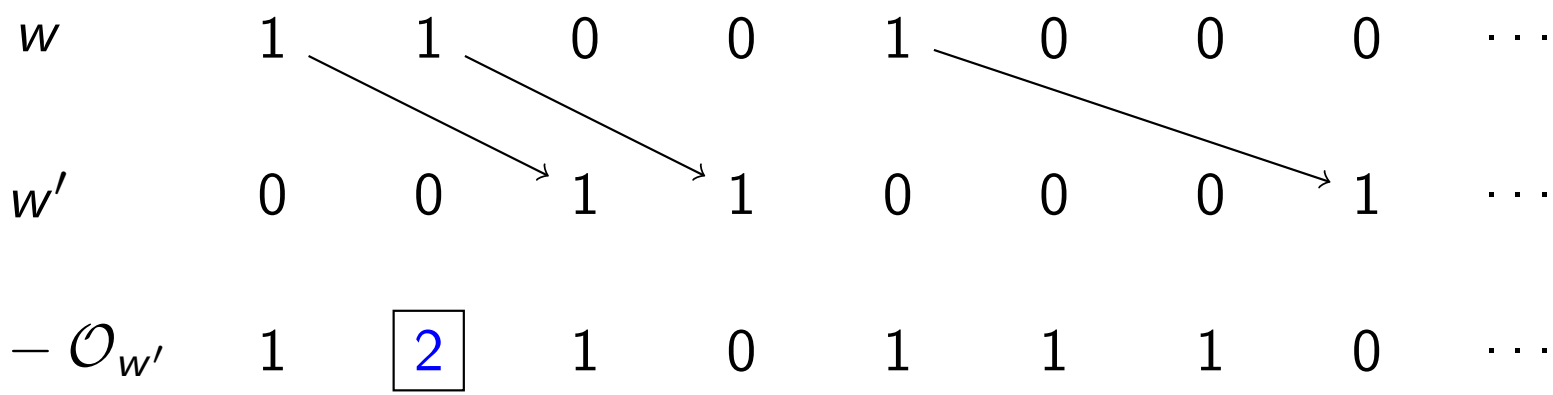
Cumulative function $\mathcal{O} : \mathbb{B}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} \mathcal{O}_w(0) & = & 0 \\ \mathcal{O}_{0.w}(i) & = & \mathcal{O}_w(i-1) \\ \mathcal{O}_{1.w}(i) & = & 1 + \mathcal{O}_w(i-1) \end{array}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Example: $(11001000) <: (00110001)$



Outline

Clocking and buffer sizing

Clocking rules:

$$\textsc{Merge}\quad \frac{\Gamma \vdash e_1 :: ck \texttt{ on } \texttt{w} \qquad \Gamma \vdash e_2 :: ck \texttt{ on not } w}{\Gamma \vdash \texttt{merge } w\ e_1\ e_2 :: ck} \qquad \textsc{Buffer}\quad \frac{\Gamma \vdash e :: ck \qquad ck <: ck'}{\Gamma \vdash \texttt{buffer } e :: ck'} \qquad \ldots$$

Induced constraint systems (equations or/and inequations):

$$\left\{ \begin{array}{rcl} p_1 \texttt{ on } w_1 & = & p'_1 \texttt{ on } w'_1 \\ & \ldots & \\ p_n \texttt{ on } w_n & <: & p'_n \texttt{ on } w'_n \end{array} \right\}$$

Solved with various linear programming-related techniques.

Causality analysis in Lucy-n

Sufficient condition à la Lustre / SCADE

- We reject feedback loops not crossing a delay...

Causality analysis in Lucy-n

Sufficient condition à la Lustre / SCADE

- We reject feedback loops not crossing a delay...
- ... but what is an n-synchronous delay?!

Causality analysis in Lucy-n

Sufficient condition à la Lustre / SCADE

- We reject feedback loops not crossing a delay...
- ... but what is an n-synchronous delay?!

Loose buffers

- A buffer $w <: w'$ is said "loose" when $w \ll: w'$
- $w \ll: w' \triangleq w <: w'$ and $\forall i \geq 1, \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$
- Thus a loose buffer never ever needs its current input to produce its current output!

Imperative code generation

Handling buffers

- We translate buffers into `push`/`pop` pairs
- For non-loose buffers, `pop` depends on `push`
- For loose buffers, `push` depends on `pop`!

Translation

We then compile in a standard way [BCHP08]:

- any topological ordering is now a valid schedule
- clocks are used to generate control code
- equations are straightforwardly translated to statements

Outline

A tale of two FIFOs

- Well-clocked Lucy-n $\Rightarrow$ well-behaved Kahn network

A tale of two FIFOs

- Well-clocked Lucy-n $\Rightarrow$ well-behaved Kahn network
- Kahn networks are deterministic and parallel

A tale of two FIFOs

- Well-clocked Lucy-n $\Rightarrow$ well-behaved Kahn network
- Kahn networks are deterministic and parallel
- . . . problem solved?

A tale of two FIFOs

- Well-clocked Lucy-n $\Rightarrow$ well-behaved Kahn network
- Kahn networks are deterministic and parallel
- . . . problem solved?

DEMO

A tale of two FIFOs

- Well-clocked Lucy-n $\Rightarrow$ well-behaved Kahn network
- Kahn networks are deterministic and parallel
- . . . problem solved?

DEMO

- Naive distribution is easy
  - only visible data-flow dependencies
  - no reaction to absence of values

A tale of two FIFOs

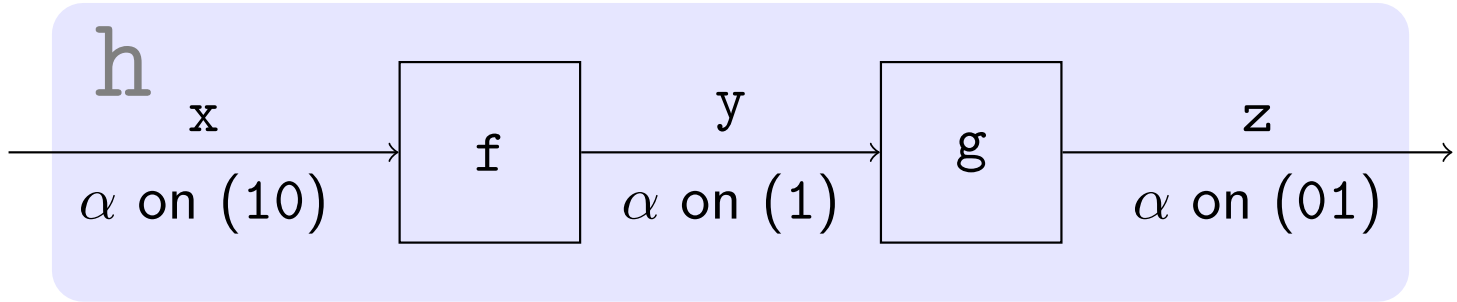
- Well-clocked Lucy-n $\Rightarrow$ well-behaved Kahn network
- Kahn networks are deterministic and parallel
- . . . problem solved?

DEMO

- Naive distribution is easy
  - only visible data-flow dependencies
  - no reaction to absence of values
- Desynchronization is hard:
  - fine-grained communication encroached in the language
  - hardware law:
    $|\, 1 \times \texttt{push(c, 2, \&v)} \,| > |\, 2 \times \texttt{push(c, 1, \&v[i])} \,|$

Outline

Desynchronization: a simple idea

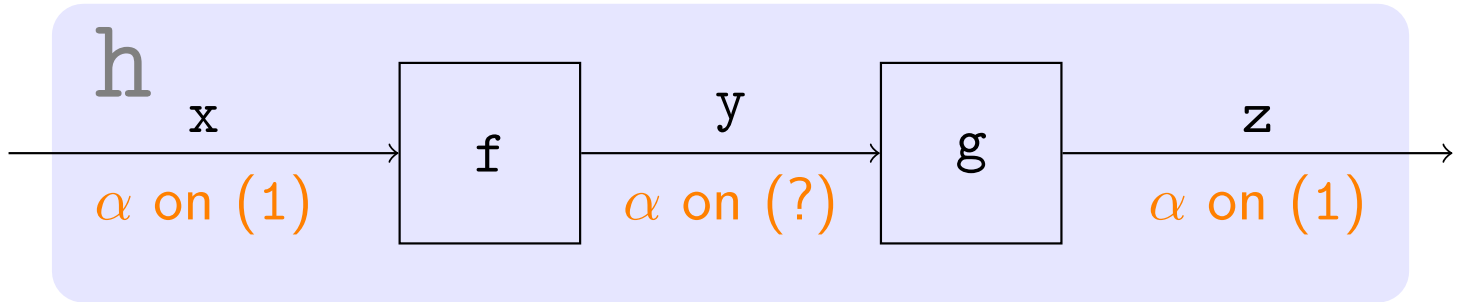



```
int a, b;
unsigned int i = 0;
f_mem mem;
while(1) {
  if(i++ % 2 == 0)
    pop(x, 1, &a);
  b = f_step(a, mem);
  push(y, 1, &b);
}
```

```
int a, b;
unsigned int i = 0;
g_mem mem;
while(1) {
  pop(y, 1, &a);
  b = g_step(a, gem);
  if(i++ % 2 == 1)
    push(z, 1, &b);
}
```

Computation-to-bulk transfer ratio: 1/2

Desynchronization: a simple idea

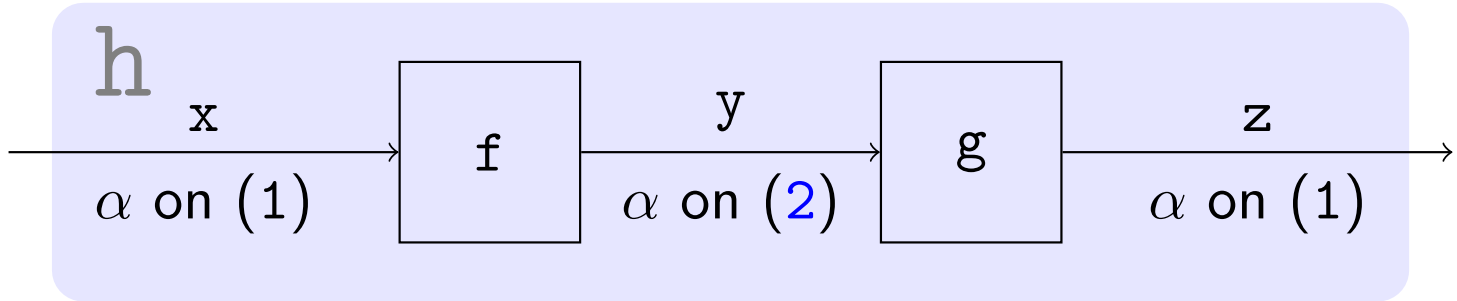



```
int a, b;
unsigned int i = 0;
f_mem mem;
while(1) {
  if(i++ % 2 == 0)
    pop(x, 1, &a);
  b = f_step(a, mem);
  push(y, 1, &b);
}
```

```
int a, b;
unsigned int i = 0;
g_mem mem;
while(1) {
  pop(y, 1, &a);
  b = g_step(a, gem);
  if(i++ % 2 == 1)
    push(z, 1, &b);
}
```

Computation-to-bulk transfer ratio: 1?

Desynchronization: a simple idea

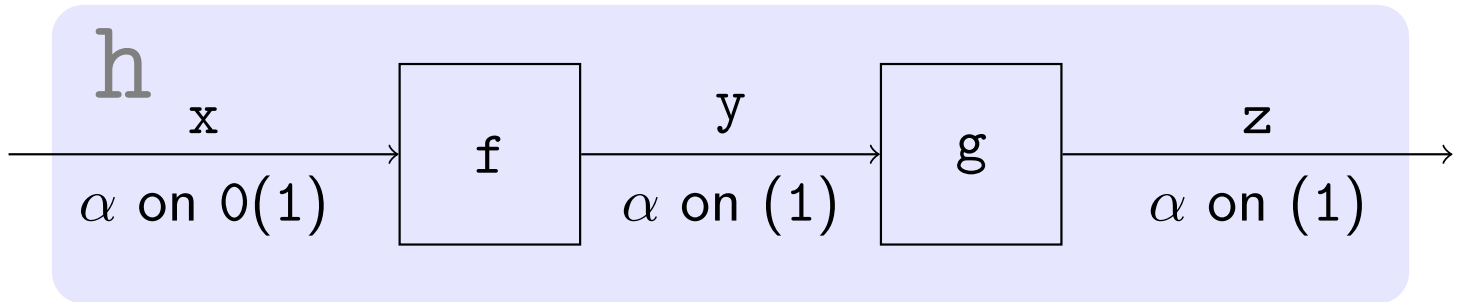



```
int a, b[2];
unsigned int i = 0;
f_mem mem;
while(true) {
  a = pop(x);
  for(i = 0; i < 2; i++)
    b[i] = f_step(a, mem);
  push(y, 2, &b);
}
```

```
int a[2], b;
unsigned int i = 0;
g_mem mem;
while(true) {
  pop(y, 2, &a);
  for(i = 0; i < 2; i++)
    b = g_step(a[i], gem);
  push(z, 1, &b);
}
```

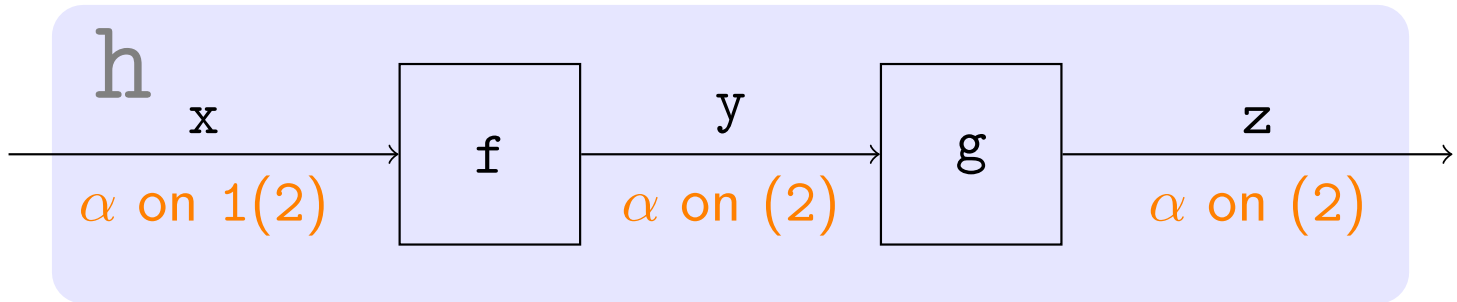
Computation-to-bulk transfer ratio: 1

Desynchronization: no free beer



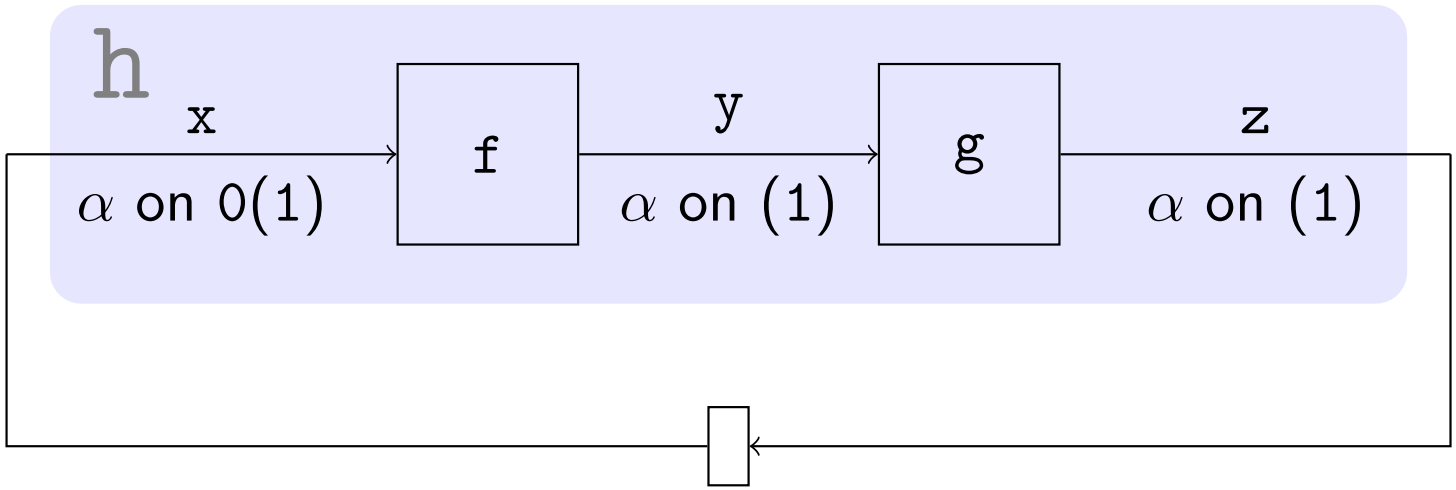
Computation-to-bulk transfer ratio: 1

Desynchronization: no free beer



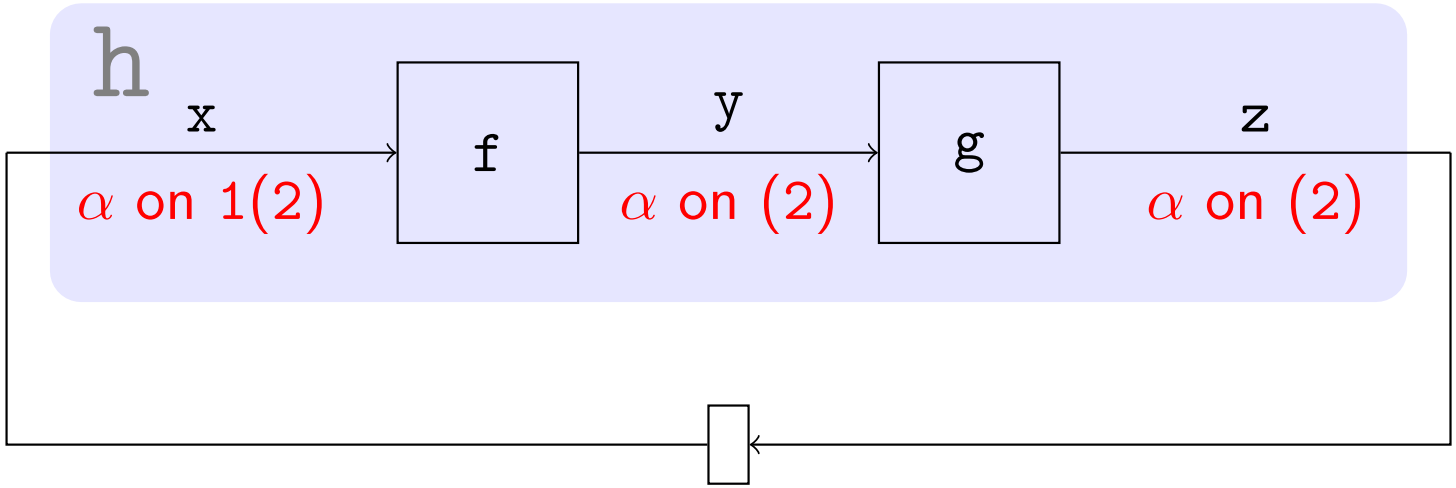
Computation-to-bulk transfer ratio: 2?

Desynchronization: no free beer



Computation-to-bulk transfer ratio:	1
Buffer:	Loose

Desynchronization: no free beer



Computation-to-bulk transfer ratio: $\times$

Buffer: Tight

DEADLOCK

Outline

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\dots.n_n.w_2) \quad = \quad (\Sigma_{1 \leq i \leq p} n_i).(w_1 \;\; \text{on} \;\; w_2)$$

Example:

(103)	1	0	3	1	0	3	...
0(10)							
(103) on 0(10)							

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1	0		3		1	0		3				...
0(10)	0	.											
(103) on 0(10)													

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \; = \; (\Sigma_{1 \leq i \leq p} n_i).(w_1 \; \text{on} \; w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1 0
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1 0 1
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1 0 1 0
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1 0 1 0 .
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1 0 1 0 . 1
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	$1\ 0\ 3\ 1\ 0\ 3\ \ldots$
$0(10)$	$0\ .\ 1\,0\,1\ 0\ .\ 1\,0$
(103) on $0(10)$	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \quad = \quad (\Sigma_{1\leq i\leq p} n_i).(w_1 \quad \text{on} \quad w_2)$$

Example:

(103)	1 0 3 1 0 3 ...
0(10)	0 . 1 0 1 0 . 1 0 1 ...
(103) on 0(10)	

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \;=\; (\Sigma_{1 \le i \le p} n_i).(w_1 \;\; \text{on} \;\; w_2)$$

Example:

(103)	1	0	3	1	0	3	...
0(10)	0	.	1 0 1	0	.	1 0 1	...
(103) on 0(10)	0	0	2	0	0	2	...

Integer clocks 101

$$\mathbb{N}^\omega \triangleq \mathbb{N} \to \mathbb{N}$$

$$(p.w_1) \quad \text{on} \quad (n_1.\ldots.n_n.w_2) \;=\; (\Sigma_{1\leq i\leq p} n_i).(w_1 \;\text{ on }\; w_2)$$

Example:

(103)	1	0	3	1	0	3	...
0(10)	0	.	1 0 1	0	.	1 0 1	...
(103) on 0(10) = (002)	0	0	2	0	0	2	...

Integer clocks and buffers

Cumulative function $\mathcal{O} : \mathbb{N}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{array}{rcl} \mathcal{O}_w(0) & = & 0 \\ \mathcal{O}_{n.w}(i) & = & n + \mathcal{O}_w(i-1) \end{array}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Integer clocks and buffers

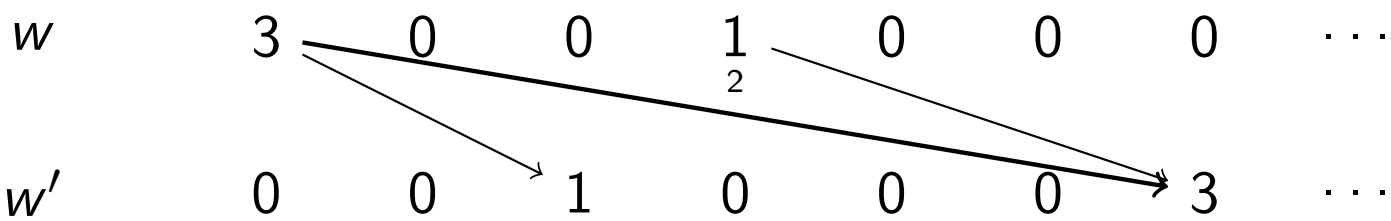
Cumulative function $\mathcal{O} : \mathbb{N}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{aligned} \mathcal{O}_w(0) &= 0 \\ \mathcal{O}_{n.w}(i) &= n + \mathcal{O}_w(i-1) \end{aligned}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Example: $(3001000) <: (0010003)$



Integer clocks and buffers

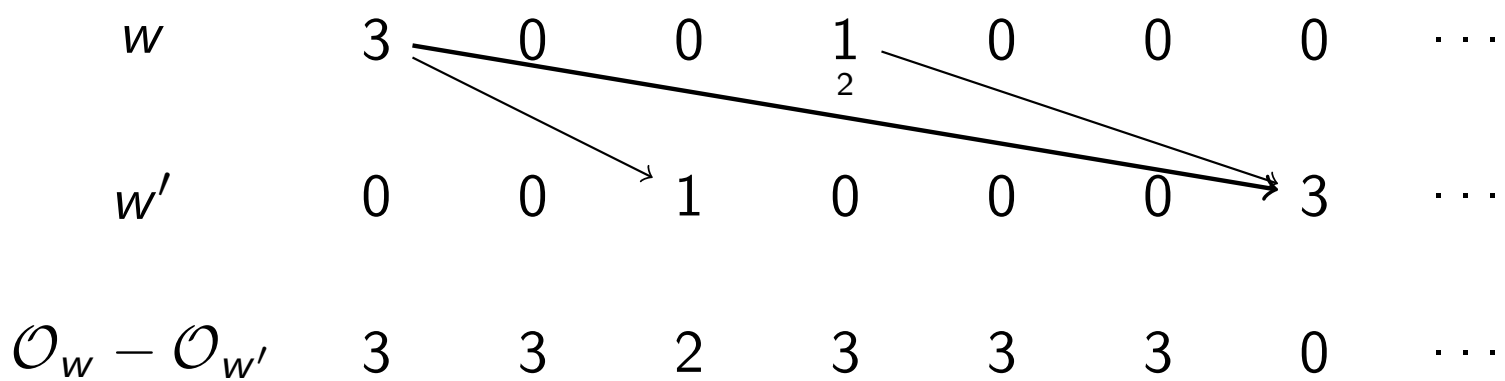
Cumulative function $\mathcal{O} : \mathbb{N}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{aligned} \mathcal{O}_w(0) &= 0 \\ \mathcal{O}_{n.w}(i) &= n + \mathcal{O}_w(i-1) \end{aligned}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Example: $(3001000) <: (0010003)$



Integer clocks and buffers

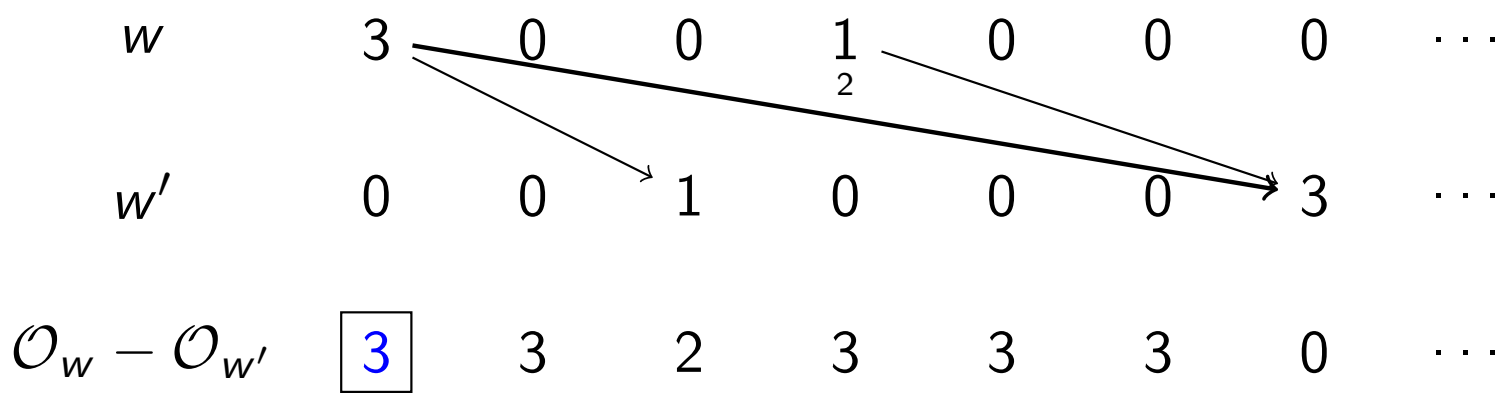
Cumulative function $\mathcal{O} : \mathbb{N}^\omega \times \mathbb{N} \to \mathbb{N}$

$$\begin{aligned} \mathcal{O}_w(0) &= 0 \\ \mathcal{O}_{n.w}(i) &= n + \mathcal{O}_w(i-1) \end{aligned}$$

Adaptability relation $<:$

$$w <: w' \quad \Leftrightarrow \quad \exists k \geq 0, \forall i \in \mathbb{N}, 0 \leq \mathcal{O}_w(i) - \mathcal{O}_{w'}(i) \leq k$$

Example: $(3001000) <: (0010003)$



Integer clocks: strict adaptability

$$w \ll: w' \triangleq w <: w' \text{ and } \forall i \geq 1. \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$$

However. . .

Integer clocks: strict adaptability

$$w \ll: w' \triangleq w <: w' \text{ and } \forall i \geq 1. \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$$

However...

$$\forall w_1, w_2, w_3 \in \mathbb{B}^\omega \quad w_1 \ll: w_2 \quad \Rightarrow \quad w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2$$

Integer clocks: strict adaptability

$$w \ll: w' \triangleq w <: w' \text{ and } \forall i \geq 1. \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$$

However...

$$\begin{array}{lllll} \forall w_1, w_2, w_3 \in \mathbb{B}^\omega & w_1 \ll: w_2 & \Rightarrow & w_3 \texttt{ on } w_1 \ll: w_3 \texttt{ on } w_2 \\ \forall w_1, w_2, w_3 \in \mathbb{N}^\omega & w_1 \ll: w_2 & \not\Rightarrow & w_3 \texttt{ on } w_1 \ll: w_3 \texttt{ on } w_2 \end{array}$$

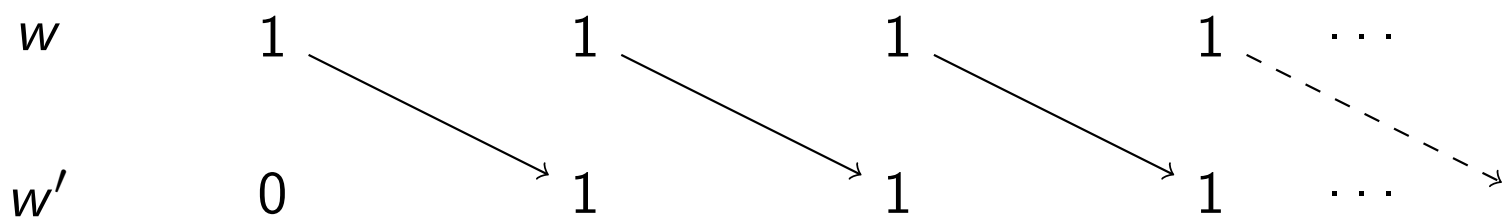
Integer clocks: strict adaptability

$$w \ll: w' \triangleq w <: w' \text{ and } \forall i \geq 1. \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$$

However...

$$\forall w_1, w_2, w_3 \in \mathbb{B}^\omega \quad w_1 \ll: w_2 \Rightarrow w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2$$
$$\forall w_1, w_2, w_3 \in \mathbb{N}^\omega \quad w_1 \ll: w_2 \not\Rightarrow w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2$$

Example: $(1) \ll: 0(1)$ but $(2) \not\ll: 1(2)$



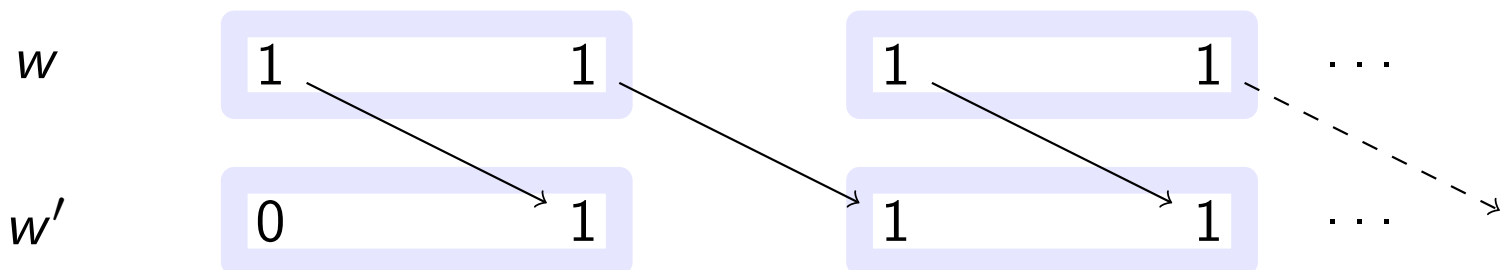
Integer clocks: strict adaptability

$$w \ll: w' \triangleq w <: w' \text{ and } \forall i \geq 1. \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$$

However...

$$\begin{array}{lllll} \forall w_1, w_2, w_3 \in \mathbb{B}^\omega & w_1 \ll: w_2 & \Rightarrow & w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2 \\ \forall w_1, w_2, w_3 \in \mathbb{N}^\omega & w_1 \ll: w_2 & \not\Rightarrow & w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2 \end{array}$$

Example: $(1) \ll: 0(1)$ but $(2) \not\ll: 1(2)$



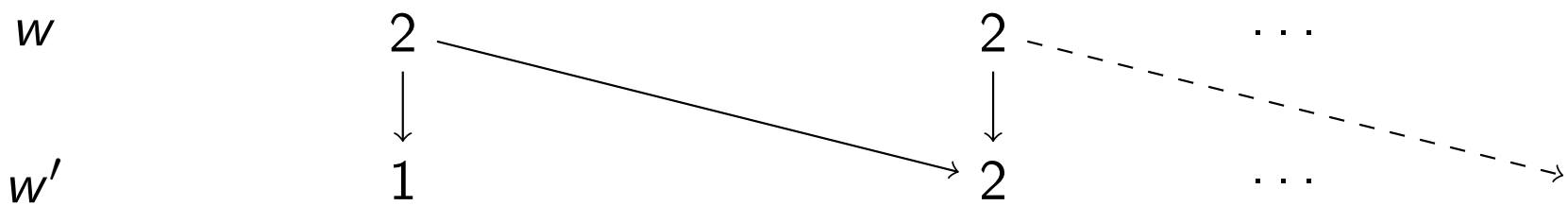
Integer clocks: strict adaptability

$$w \ll: w' \triangleq w <: w' \text{ and } \forall i \geq 1. \mathcal{O}_w(i-1) \geq \mathcal{O}_{w'}(i)$$

However...

$$\forall w_1, w_2, w_3 \in \mathbb{B}^\omega \quad w_1 \ll: w_2 \quad \Rightarrow \quad w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2$$
$$\forall w_1, w_2, w_3 \in \mathbb{N}^\omega \quad w_1 \ll: w_2 \quad \not\Rightarrow \quad w_3 \text{ on } w_1 \ll: w_3 \text{ on } w_2$$

Example: $(1) \ll: 0(1)$ but $(2) \not\ll: 1(2)$



Outline

μNIR syntax

e	$::=$	c	constant
	$\mid$	x	variable
	$\mid$	$f\ (e, \ldots, e)$	application
	$\mid$	`merge` $bw\ e\ e$	fusion
	$\mid$	e `when` bw	sampling
	$\mid$	`buffer` e	buffer
bw	$::=$	$b^*(b^+)$	UP binary word
nw	$::=$	$n^*(n^+)$	UP integer word
ck	$::=$	α	clock variable
	$\mid$	ck `on` nw	clock sampling
eq	$::=$	$(x, \ldots, x) = e$	equation
	$\mid$	eq `and` eq	parallel composition
d	$::=$	`let node` $f\ (x, \ldots, x) = e$	node definition

μNIR semantics

Denotational semantics

- As usual, of two kinds: Kahn and (int-)synchronous

μNIR semantics

Denotational semantics

- As usual, of two kinds: Kahn and (int-)synchronous
- Distinguished by the underlying nature of streams

μNIR semantics

Denotational semantics

- As usual, of two kinds: Kahn and (int-)synchronous
- Distinguished by the underlying nature of streams
- Realized in Haskell

μNIR semantics

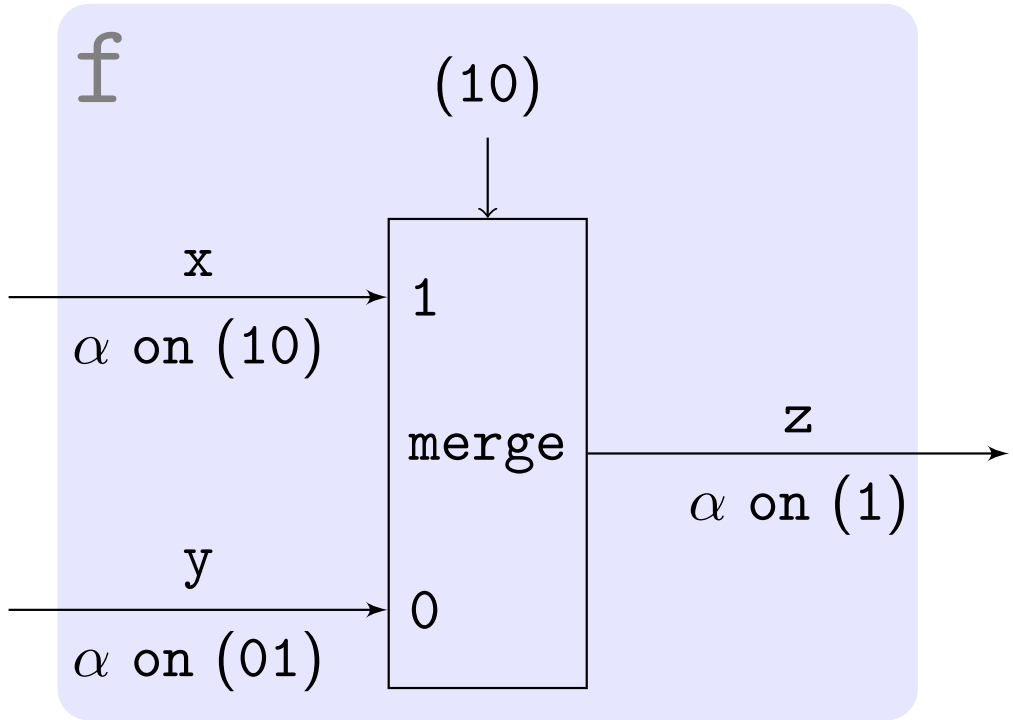
Denotational semantics

- As usual, of two kinds: Kahn and (int-)synchronous
- Distinguished by the underlying nature of streams
- Realized in Haskell

Haskell types

- Kahn semantics: `type Stream a = [a]`
- int-synchronous semantics: `type Stream a = [SList a]`

Revisiting stutter

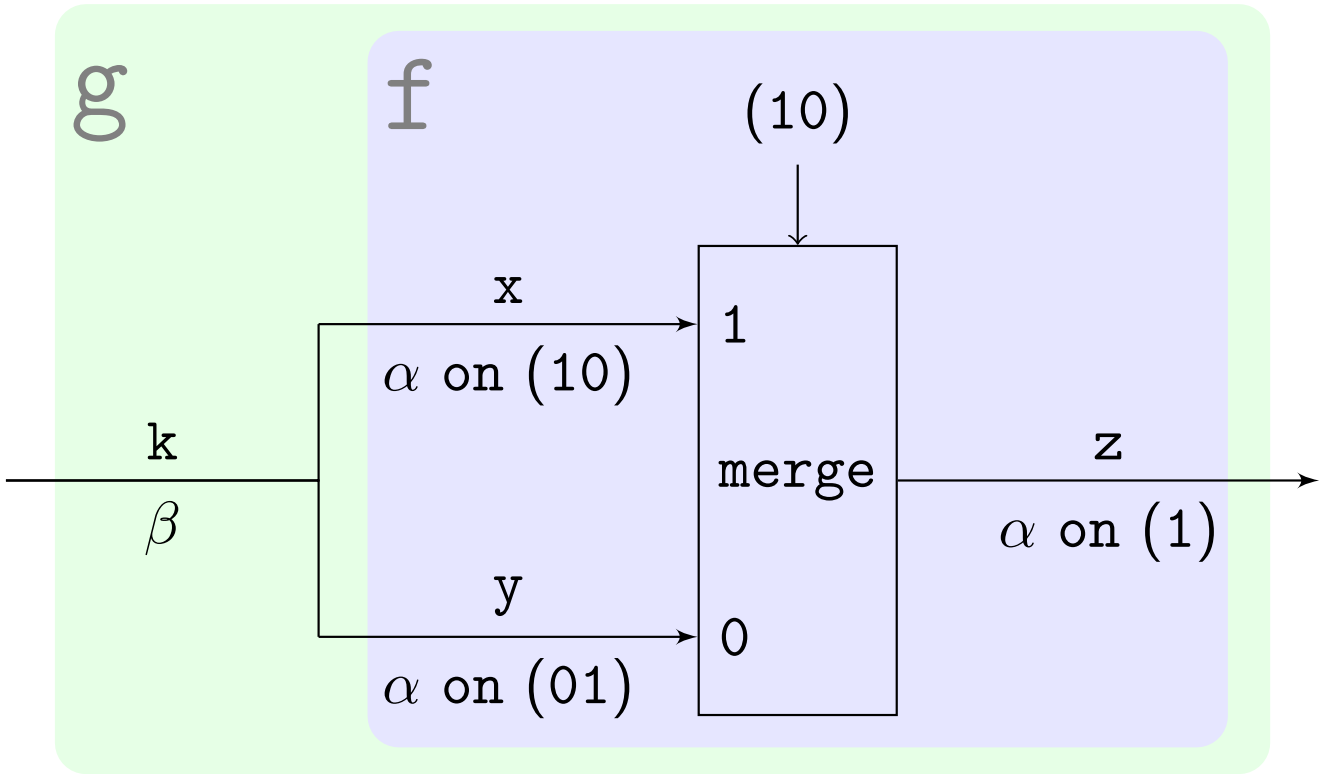



```
let node g k = f (k, k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{N}^{\omega} \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \equiv \alpha \text{ on } (01)$$

Revisiting stutter

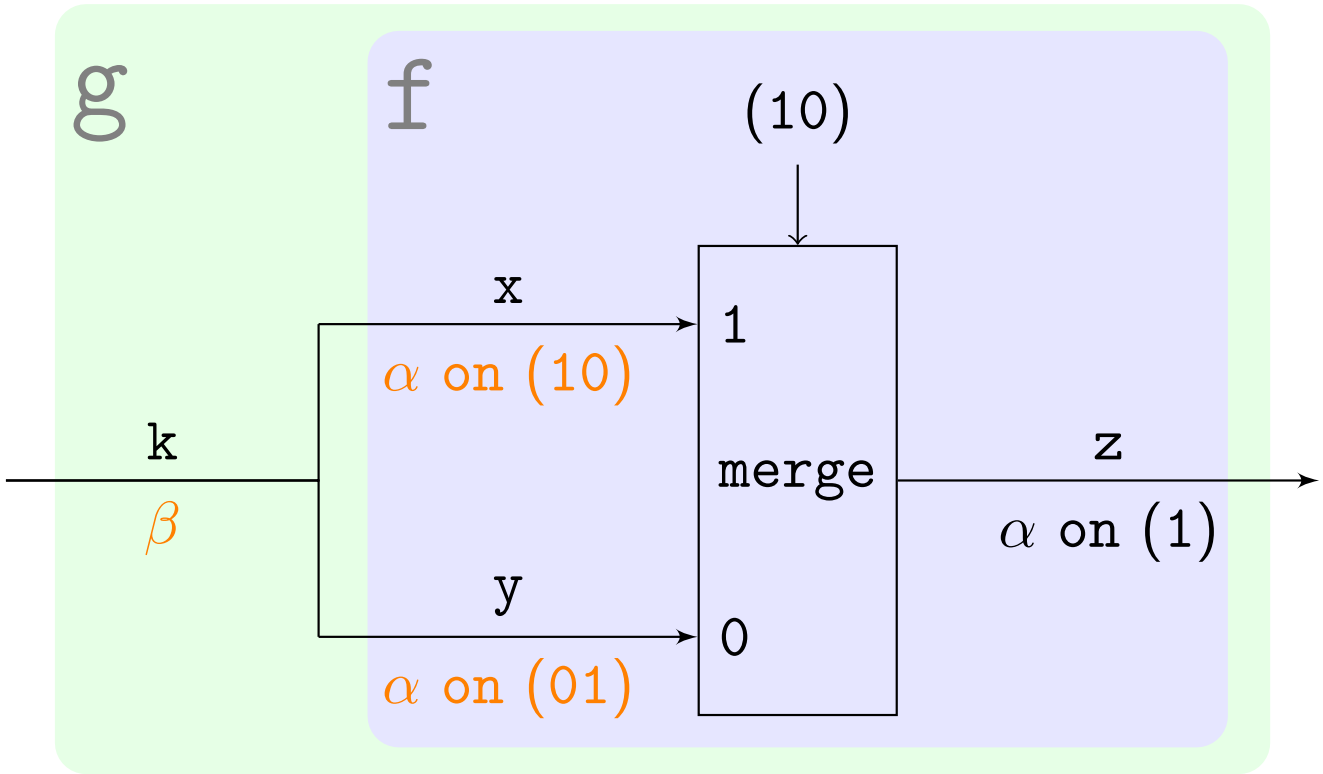



```
let node g k = f (k, k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{N}^{\omega} \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \equiv \alpha \text{ on } (01)$$

Revisiting stutter

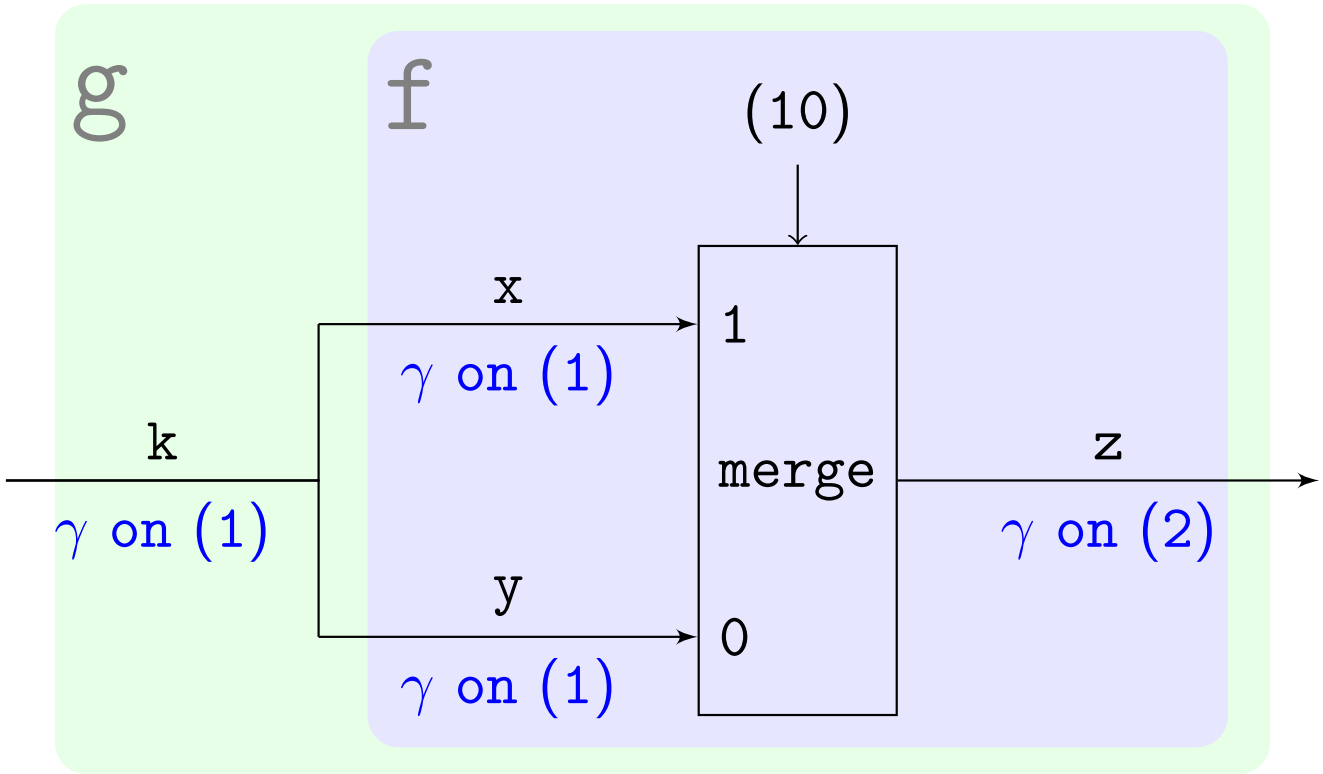



```
let node g k = f (k, k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{N}^{\omega} \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \equiv \alpha \text{ on } (01)$$

Revisiting stutter




```
let node g k = f (k, k)
```

The program is well-clocked iff:

$$\exists \alpha, \beta \in \mathbb{N}^\omega \text{ s.t. } \beta \equiv \alpha \text{ on } (10) \equiv \alpha \text{ on } (01)$$

$$\forall \gamma \in \mathbb{N}^\omega, \quad \alpha = \gamma \text{ on } (2)$$
$$\beta = \gamma \text{ on } (2) \text{ on } (10) = \gamma \text{ on } (2) \text{ on } (01) = \gamma$$

Outline

Causality and modularity (1/2)

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

```
void f_step(f_mem *mem,
            int *out) {
  int b, o;
  if(mem->cpt > 0)
    b = mem->b;
  if(mem->cpt == 0)
    o = 0;
  else
    o = 1 + b;
  mem->b = o;
  mem->cpt++;
  *out = o;
}
```

- Same causality criterion
- Can we have o on some clock α on (2)?

Causality and modularity (2/2)

- Clock desynchronization $\equiv$ Loop distribution
- Impossible here: tight dependency
- Once scheduled, can be freely called with any clock!

```
void f_step(f_mem *mem, int *out) {
  int b, o;
  for(int i = 0; i < 2; i++) {
    if(mem->cpt > 0)
      b = mem->b;
    o = (mem->cpt == 0 ? 0 : 1 + b);
    mem->b = o;
    mem->cpt++;
    *out++ = o;
  }
}
```

Sketch of software code generation

Same principles...

- Static scheduling
- Clock-based control code
- Equations to statement
- Distinguish between memory and local variables

Sketch of software code generation

Same principles...

- Static scheduling
- Clock-based control code
- Equations to statement
- Distinguish between memory and local variables

... different details!

Sketch of software code generation

Same principles...

- Static scheduling
- Clock-based control code
- Equations to statement
- Distinguish between memory and local variables

... different details!

- Stream representation: arrays (and array windows)

Sketch of software code generation

Same principles...

- Static scheduling
- Clock-based control code
- Equations to statement
- Distinguish between memory and local variables

... different details!

- Stream representation: arrays (and array windows)
- Control code: `for` loops

Outline

Related work

Lots of related work, from synchronous programming to automatic parallelization:

- Tangentially related: distribution of synchronous programs
  - lots of work from the SIGNAL community
  - quasi-synchronous model from Caspi et al.
  - ...
- Desynchronization of synchronous programs
  - OCREP (Girault)
  - Reactive domains in REACTIVEML (Pasteur)
  - Futures in HEPTAGON (Gérard)
- STREAMIT and *DF (SDF, CSDF...) models in general
- Automatic parallelization of loop nests (polyhedral model)
- ...

Conclusion

Take-home points

- Uniform theory of modular data-flow desynchronization
- Now completely understood thanks to n-synchronous theory
- Not restricted to ultimately periodic fusion/sampling

Saved for future talk

- Full definition of the Haskell operators (ask me!)
- Complete NIR language, clock calculus, `block` construct
- Compilation to software and circuits

Next questions

- Program transformations enabled by integer clocks
- Semantic characterization of the effect of scheduling

Conclusion

Take-home points

- Uniform theory of modular data-flow desynchronization
- Now completely understood thanks to n-synchronous theory
- Not restricted to ultimately periodic fusion/sampling

Saved for future talk

- Full definition of the Haskell operators (ask me!)
- Complete NIR language, clock calculus, `block` construct
- Compilation to software and circuits

Next questions

- Program transformations enabled by integer clocks
- Semantic characterization of the effect of scheduling

Conclusion

Thank you!

Take-home points

- Uniform theory of modular data-flow desynchronization
- Now completely understood thanks to n-synchronous theory
- Not restricted to ultimately periodic fusion/sampling

Saved for future talk

- Full definition of the Haskell operators (ask me!)
- Complete NIR language, clock calculus, `block` construct
- Compilation to software and circuits

Next questions

- Program transformations enabled by integer clocks
- Semantic characterization of the effect of scheduling


Darek Biernacki, Jean-Louis Colaco, Grégoire Hamon, and Marc Pouzet, Clock-directed Modular Code Generation of Synchronous Data-flow Languages, ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES) (Tucson, Arizona), June 2008.


Louis Mandel, Florence Plateau, and Marc Pouzet, Lucy-n: a n-synchronous extension of Lustre, Tenth International Conference on Mathematics of Program Construction (MPC 2010) (Québec, Canada), June 2010.