Parallel Complexity Analysis with Temporal Session Types

Ankush Das
Jan Hoffmann
Frank Pfenning

ICFP, Sep 26, 2018

Carnegie Mellon University
Computer Science Department

What is Parallel Complexity?

a.k.a. Span

What is Parallel Complexity?

a.k.a. Span

Total time of computation?

What is Parallel Complexity?

a.k.a. Span

Total time of computation?

Depends on amount of parallelism in system

What is Parallel Complexity?

a.k.a. Span

Total time of computation?

Depends on amount of parallelism in system

Data Dependencies	Wait for Messages	Data Races Shared Memory

Why Parallel Complexity?

Why Parallel Complexity?

Complexity of Parallel Algorithms

Blelloch (Comm. ACM '96)

Why Parallel Complexity?

Complexity of Parallel Algorithms

Blelloch (Comm. ACM '96)

Design of Optimal Scheduling Policies

Acar et. al. (JFP '16)

Why Parallel Complexity?

Complexity of Parallel Algorithms

Blelloch (Comm. ACM '96)

Design of Optimal Scheduling Policies

Acar et. al. (JFP '16)

Throughput and Latency of Streams

Mamouras et. al. (PLDI '17)

Why Parallel Complexity?

Complexity of Parallel Algorithms

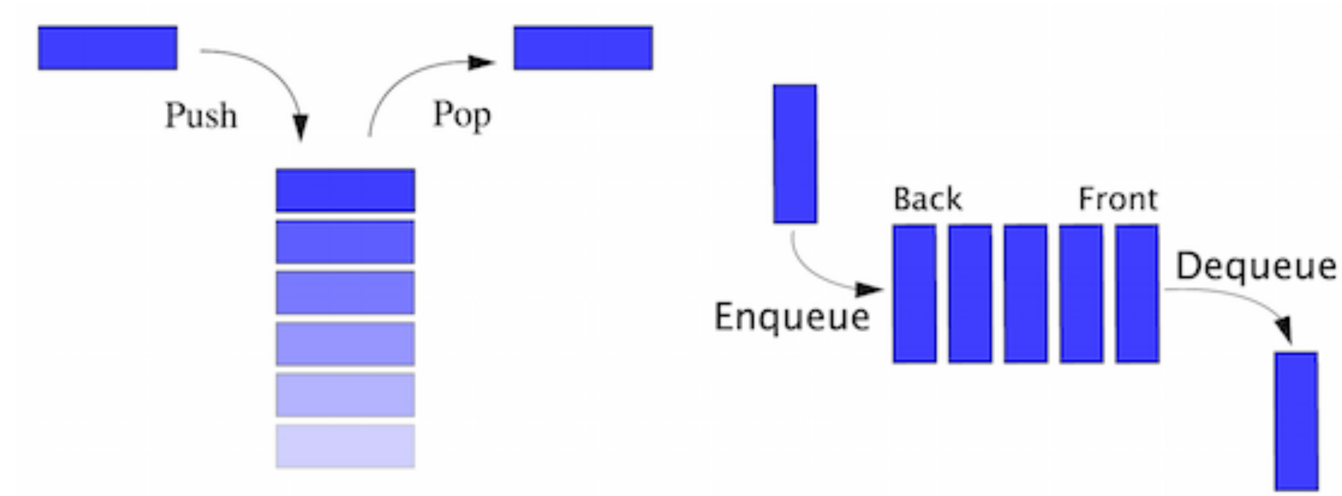
Blelloch (Comm. ACM '96)

Design of Optimal Scheduling Policies

Acar et. al. (JFP '16)

Throughput and Latency of Streams

Mamouras et. al. (PLDI '17)



Response Time of Concurrent Data Structures

Ellen and Brown (PODC '16)

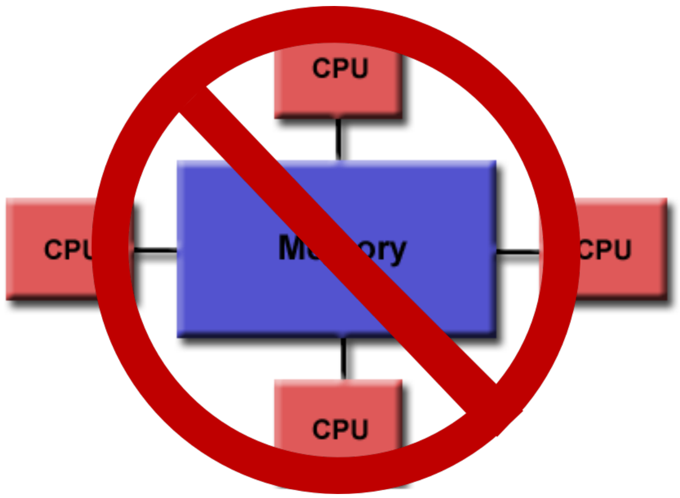
Why Session Types?

Why Session Types?

Concurrent Programs are hard to analyze!

Why Session Types?

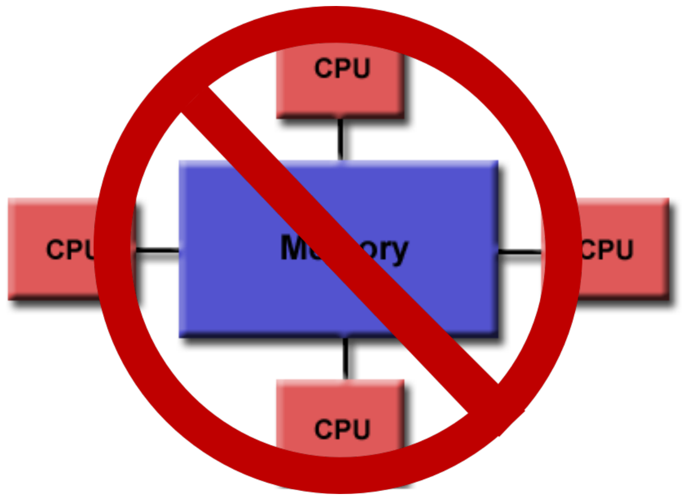
Concurrent Programs are hard to analyze!



No Shared Memory

Why Session Types?

Concurrent Programs are hard to analyze!

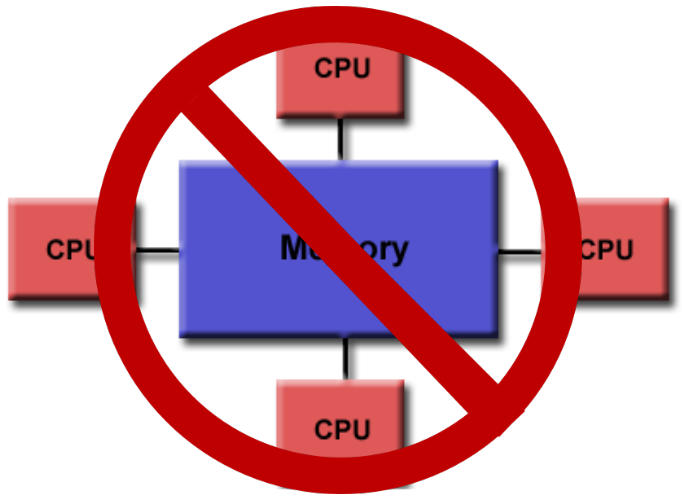


No Shared Memory

Types strictly enforce communication protocols

Why Session Types?

Concurrent Programs are hard to analyze!



No Shared Memory

Types strictly enforce communication protocols

Deadlock Freedom

What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Curry-Howard isomorphism with intuitionistic linear logic

What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Curry-Howard isomorphism with intuitionistic linear logic

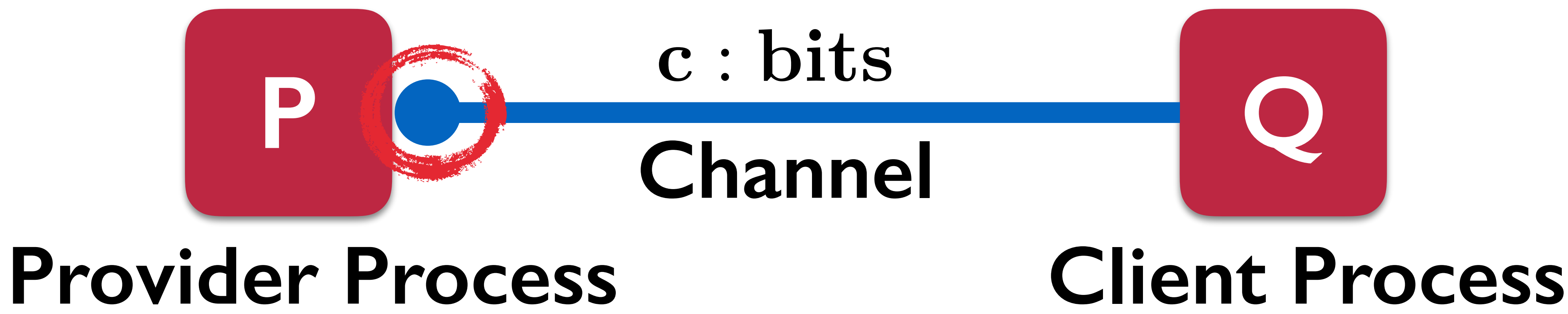
What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Curry-Howard isomorphism with intuitionistic linear logic

What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Curry-Howard isomorphism with intuitionistic linear logic

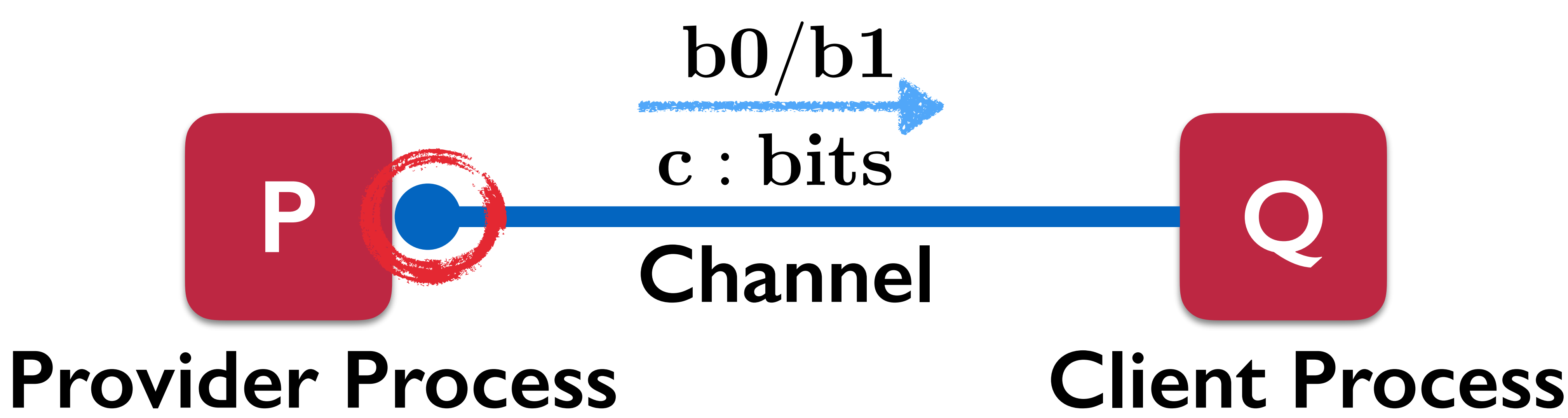
$$\mathrm{bits} = \oplus\{\mathrm{b0} : \mathrm{bits}, \mathrm{b1} : \mathrm{bits}\}$$



What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Curry-Howard isomorphism with intuitionistic linear logic

$$\text{bits} = \oplus\{\text{b0} : \text{bits}, \text{b1} : \text{bits}\}$$



Contributions

Contributions

Type system to analyze timings of message exchanges of session-typed programs

Contributions

Type system to analyze timings of message exchanges of session-typed programs

- types define the *timing* of message exchanges
- provides *precision* and *flexibility*
- proved *sound* w.r.t. *cost semantics* tracking time
- *conservative* extension to typical session type system
- applies to all *standard* session types examples
- can be *parameterized* to count resource of interest

How is time defined?

- Time is defined using a cost model
- *Cost model* assigns a time cost to each operation

How is time defined?

- **Time is defined using a cost model**
- ***Cost model*** **assigns a time cost to each operation**

$\mathcal{R}$ cost model

Unit delay after each receive

$\mathcal{RS}$ cost model

Unit delay after each receive and send

How is time defined?

- Time is defined using a cost model
- *Cost model* assigns a time cost to each operation

$\mathcal{R}$ cost model
Unit delay after each receive

$\mathcal{RS}$ cost model
Unit delay after each receive and send

- Expressed by inserting appropriate delays in the source code, only the delays cost time
- Programmer specifies cost model, compiler automatically inserts delays for type checking

$$\Omega \vdash \mathrm{P} :: (\mathrm{x} : \mathrm{S})$$

Definition of the Types

Track Message Rates

Track Message Rates

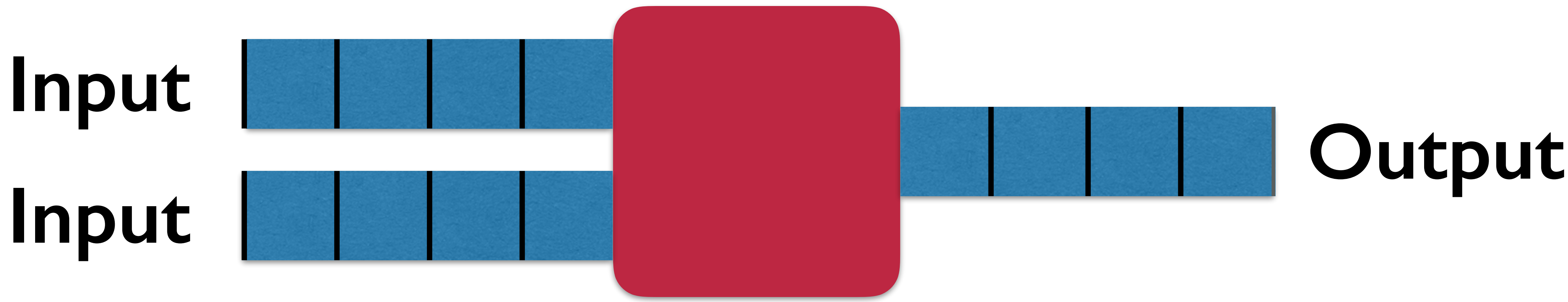
Compute output rate given input rate

Track Message Rates

Compute output rate given input rate

timing of messages ⇔ Parallel Complexity

Track Message Rates



Compute output rate given input rate

timing of messages ⇔ Parallel Complexity

Necessary:
need exact input/output rate to ensure compositionality

Sufficient:
span can be thought as timing of final message

Example: Bit Streams

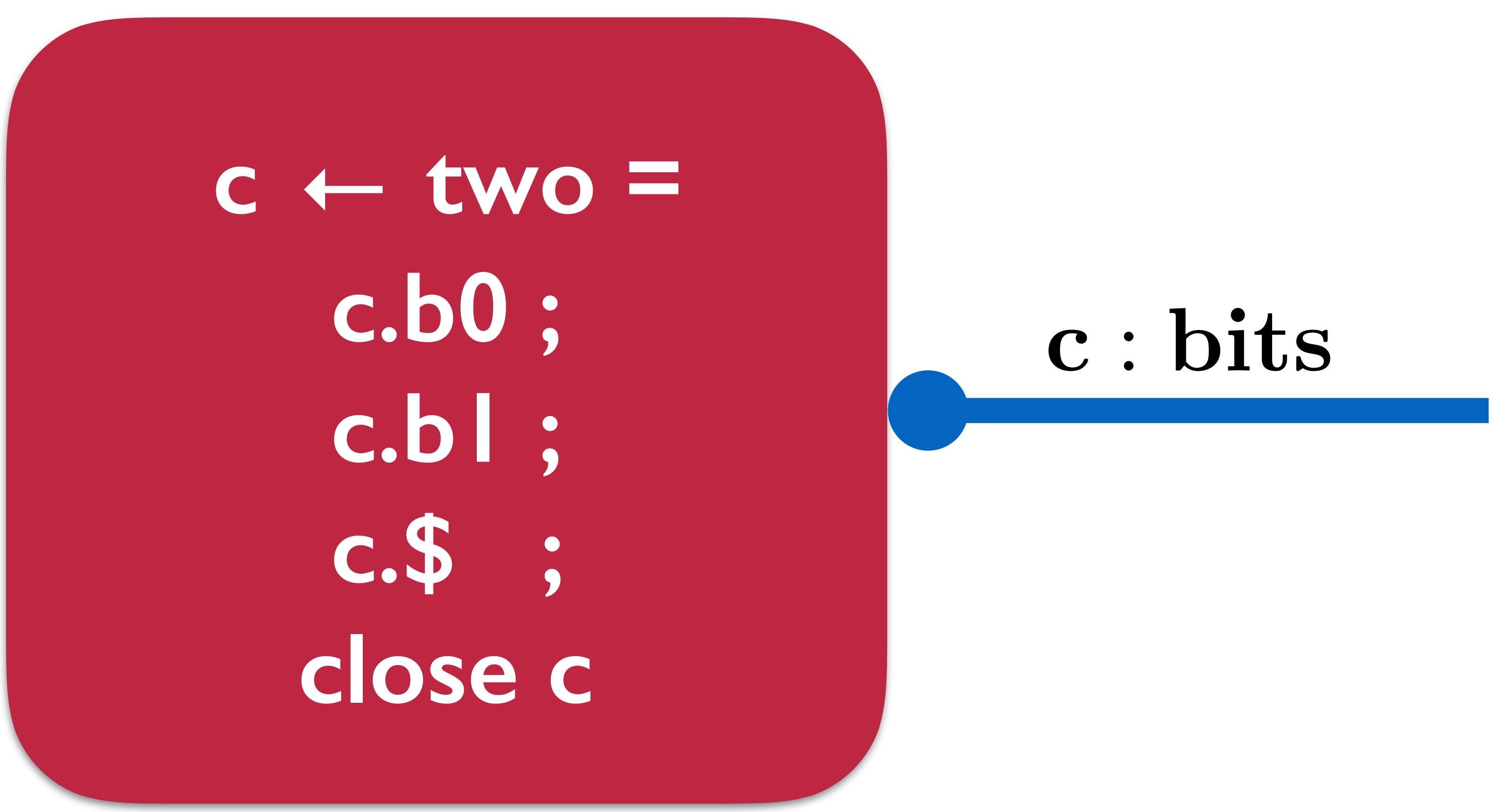
$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

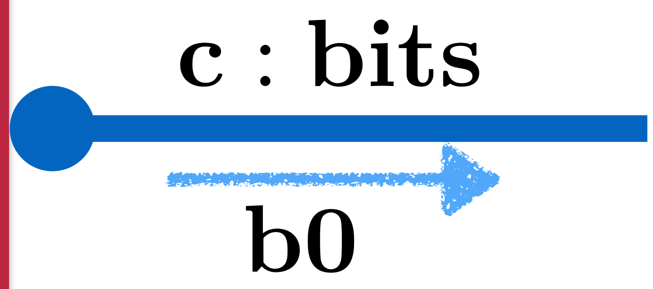


Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

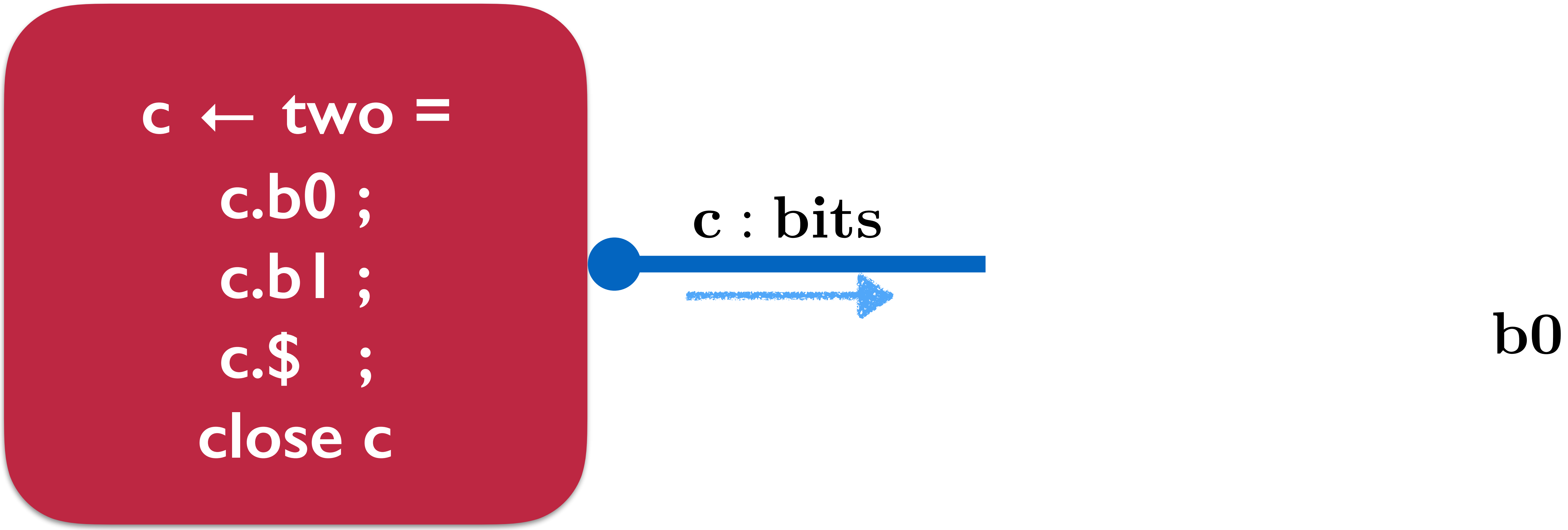
c ← two =
<u>c.b0 ;</u>
c.b1 ;
c.$;
close c



Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \underline{\mathbf{bits}}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$



Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ;
c.b1 ;
c.$  ;
close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \underline{\mathbf{bits}}, \$: \mathbf{1}\}$$

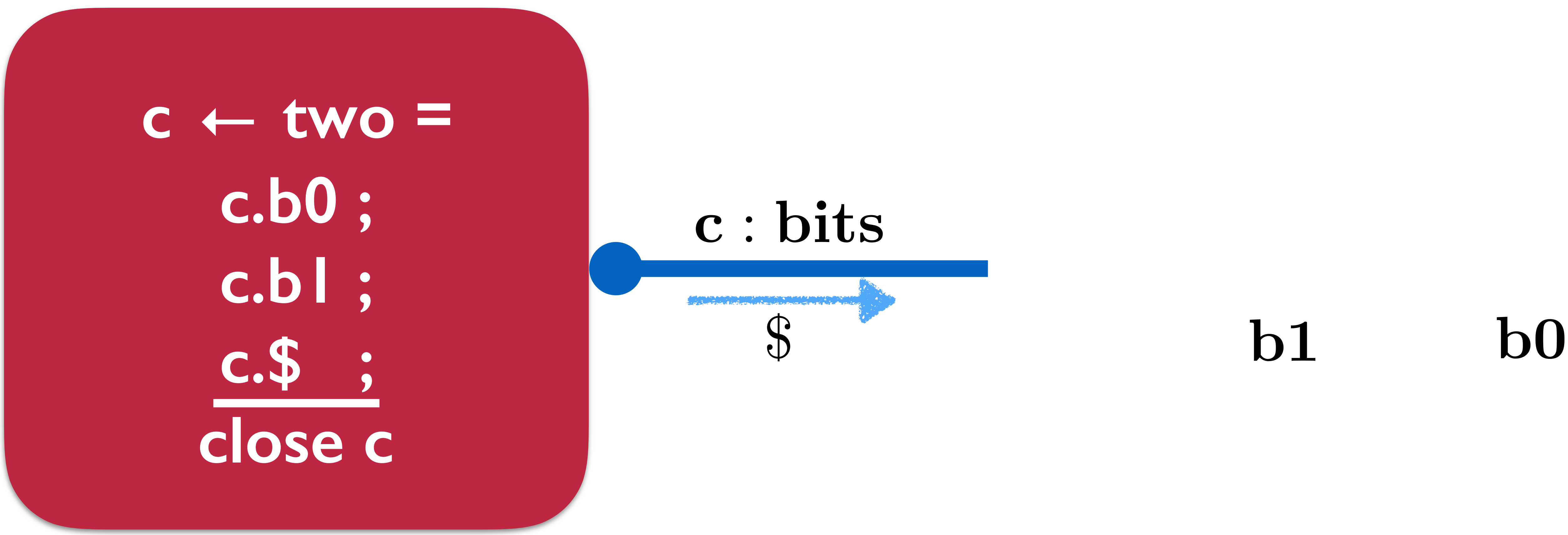
$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

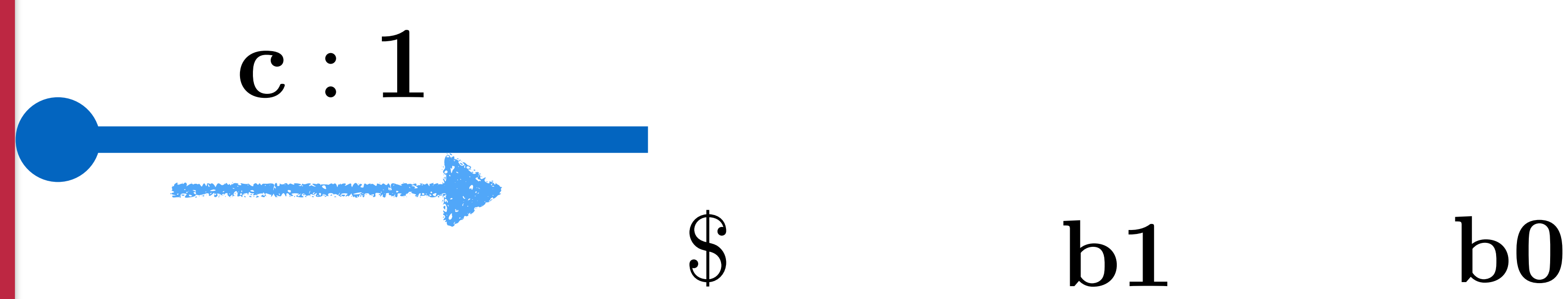


Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \underline{\mathbf{1}}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ;
c.b1 ;
c.$  ;
close c
```

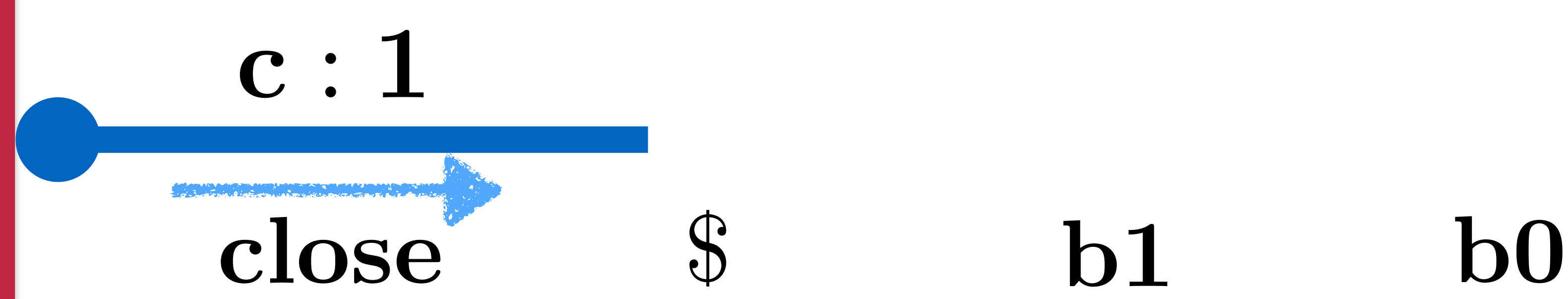


Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ;
c.b1 ;
c.$   ;
close c
```



Example: Bit Streams

$$\text{bits} = \oplus\{\text{b0} : \text{bits}, \text{b1} : \text{bits}, \$: 1\}$$

$$\cdot \vdash \text{two} :: (c : \text{bits})$$

Timing Information?

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$   ;
  close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Timing Information?

Sending a message causes unit delay

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```

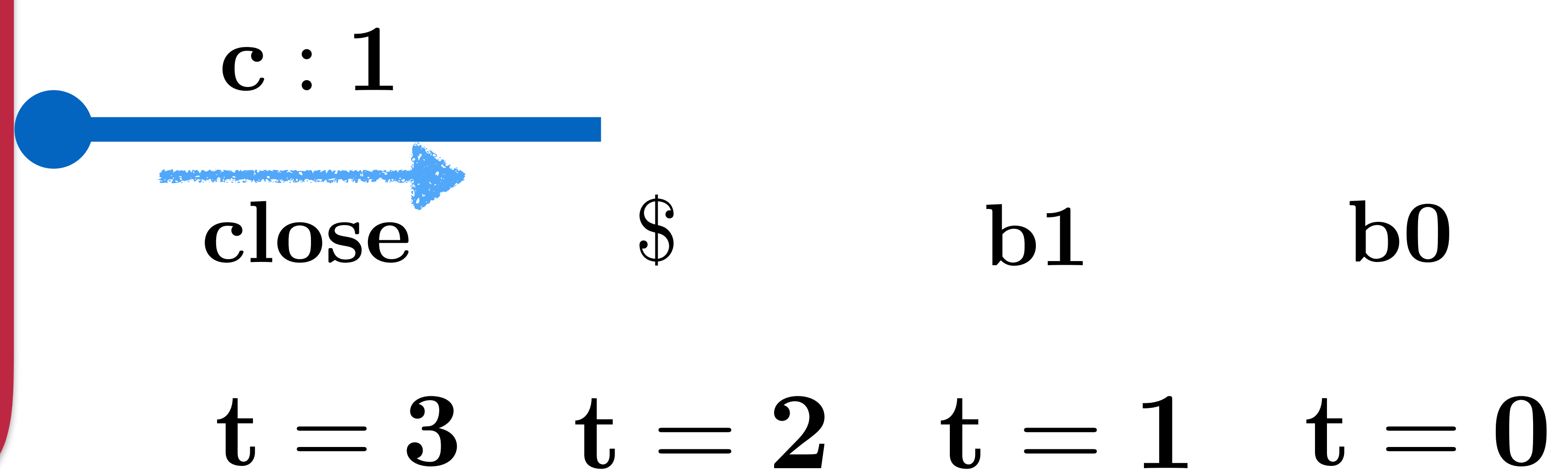
Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Timing Information?
Sending a message causes unit delay

```
c ← two =
c.b0 ;
c.b1 ;
c.$   ;
close c
```



Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

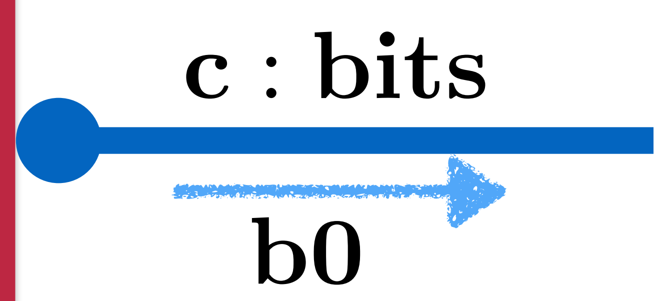
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
<u>c.b0 ;</u> delay ;
c.b1 ; delay ;
c.$; delay ;
close c



$$\mathbf{t = 0}$$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \underline{\bigcirc \mathbf{bits}}, \mathbf{b1} : \bigcirc \mathbf{bits}, \$: \bigcirc \mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$; delay ;
close c

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; <u>delay ;</u>
c.b1 ; delay ;
c.$; delay ;
close c

$\mathbf{b0}$

$\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

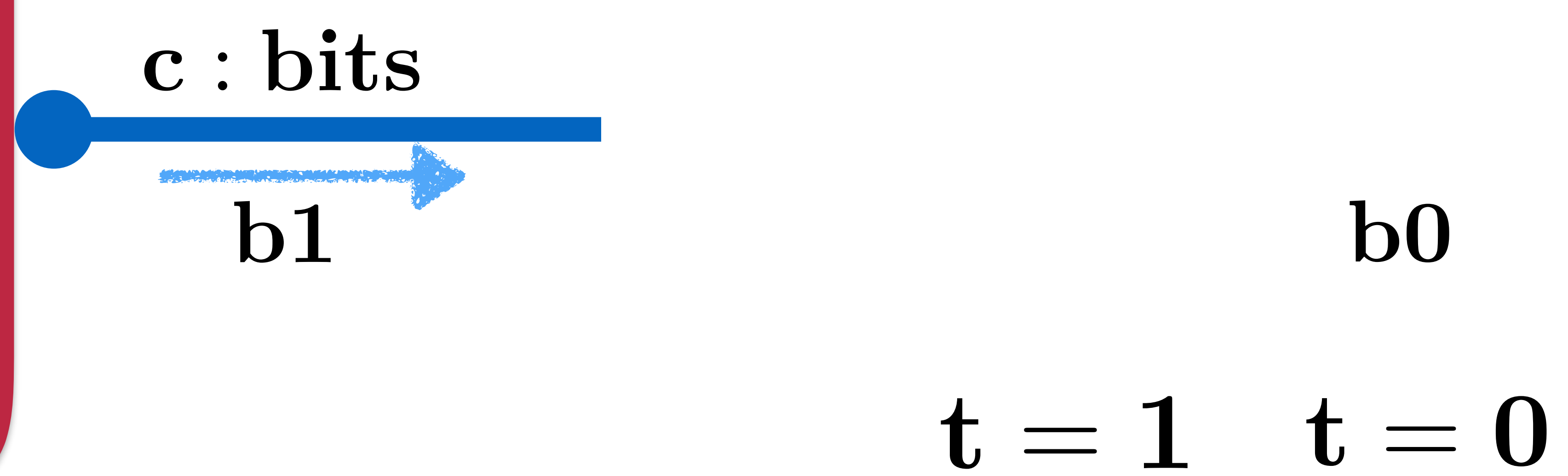
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$; delay ;
close c



Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \underline{\bigcirc\mathbf{bits}}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$; delay ;
close c

b1 b0

$\mathbf{t = 1}$ $\mathbf{t = 0}$

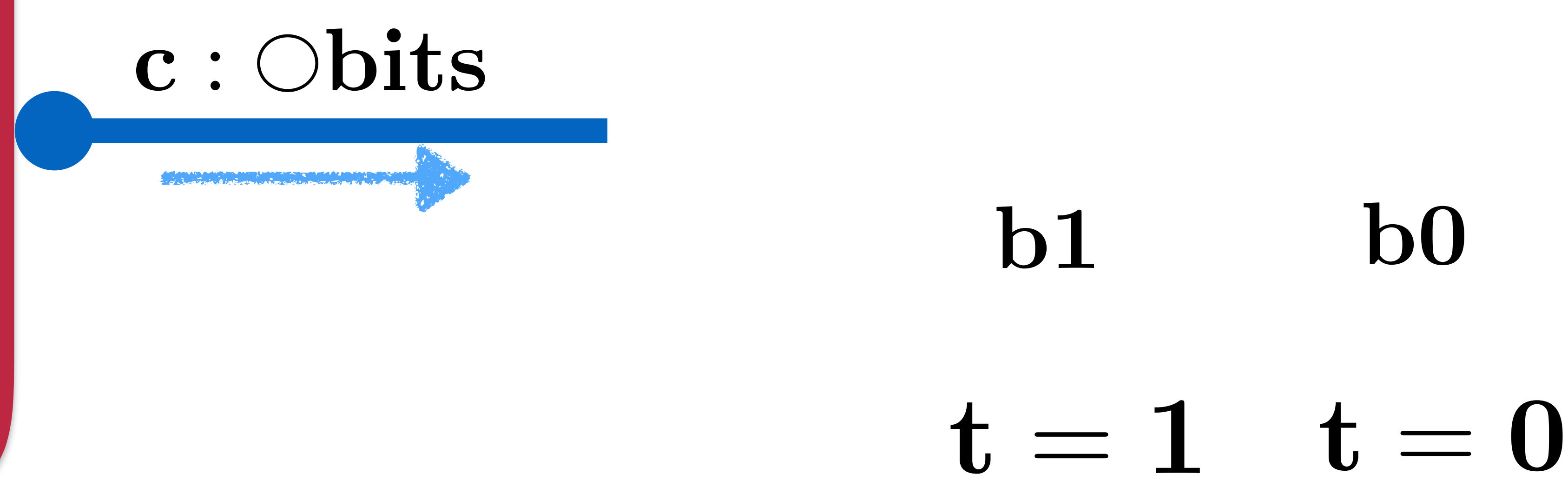
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; <u>delay ;</u>
c.$; delay ;
close c



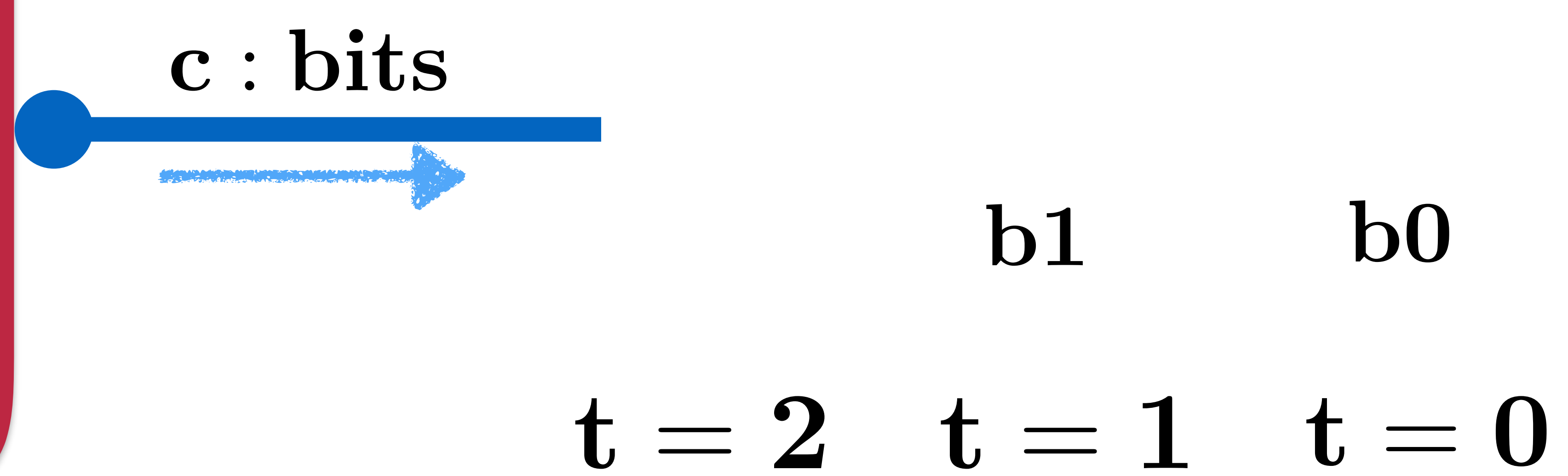
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$; delay ;
close c



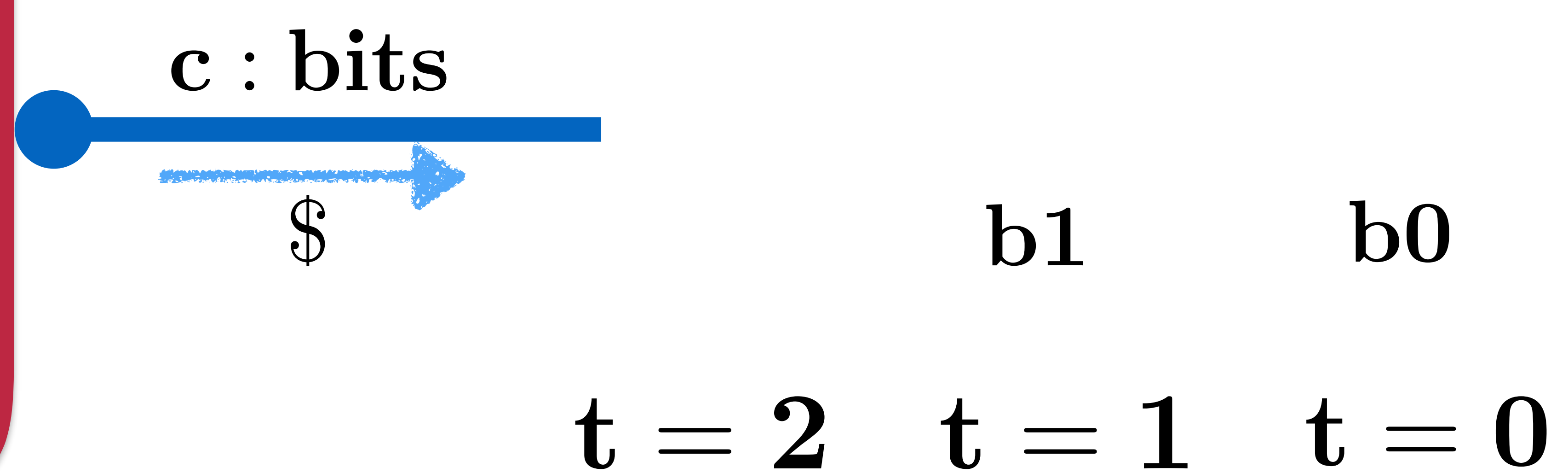
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$; delay ;
close c



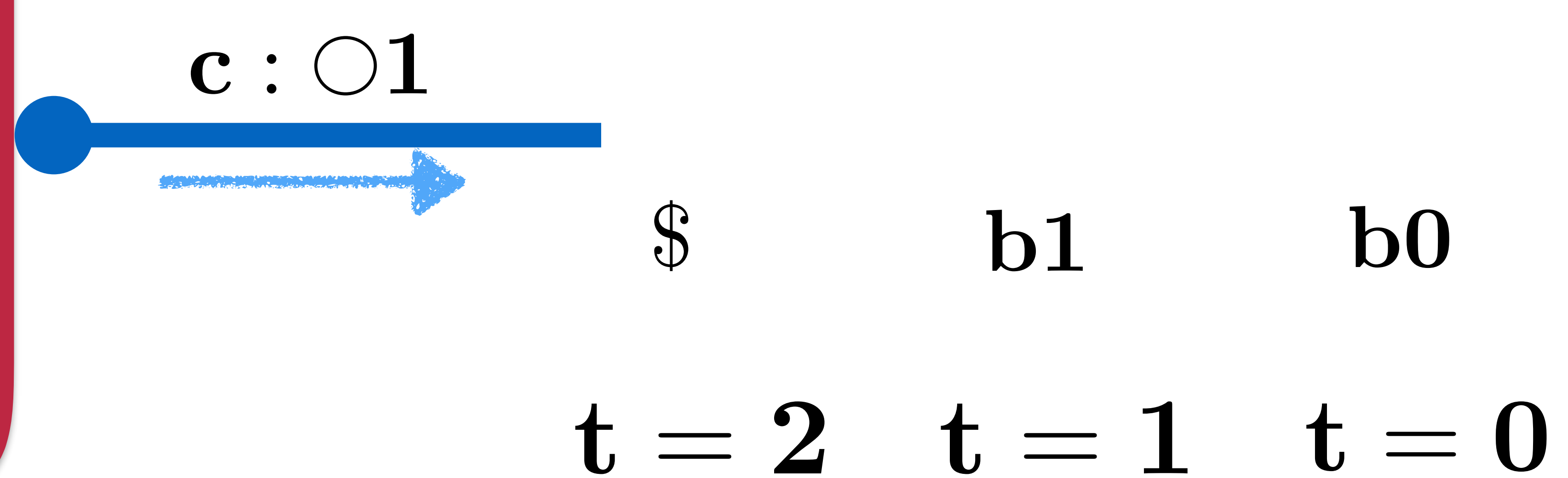
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc \mathbf{bits}, \mathbf{b1} : \bigcirc \mathbf{bits}, \$: \underline{\bigcirc \mathbf{1}}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$   ; delay ;
close c
```



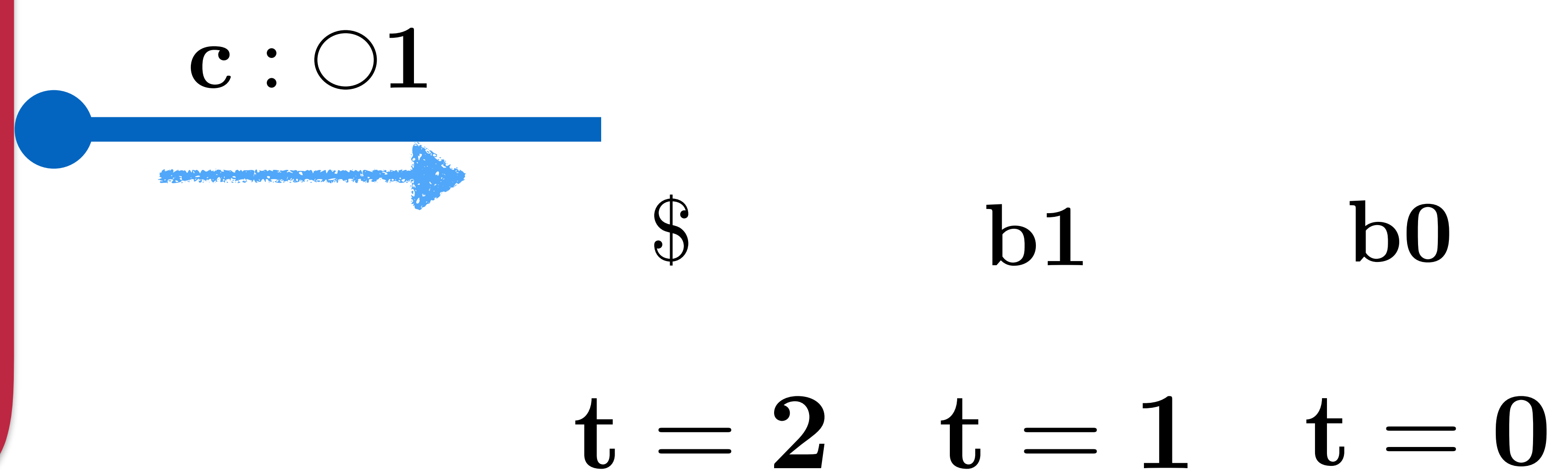
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$   ; delay ;
close c
```



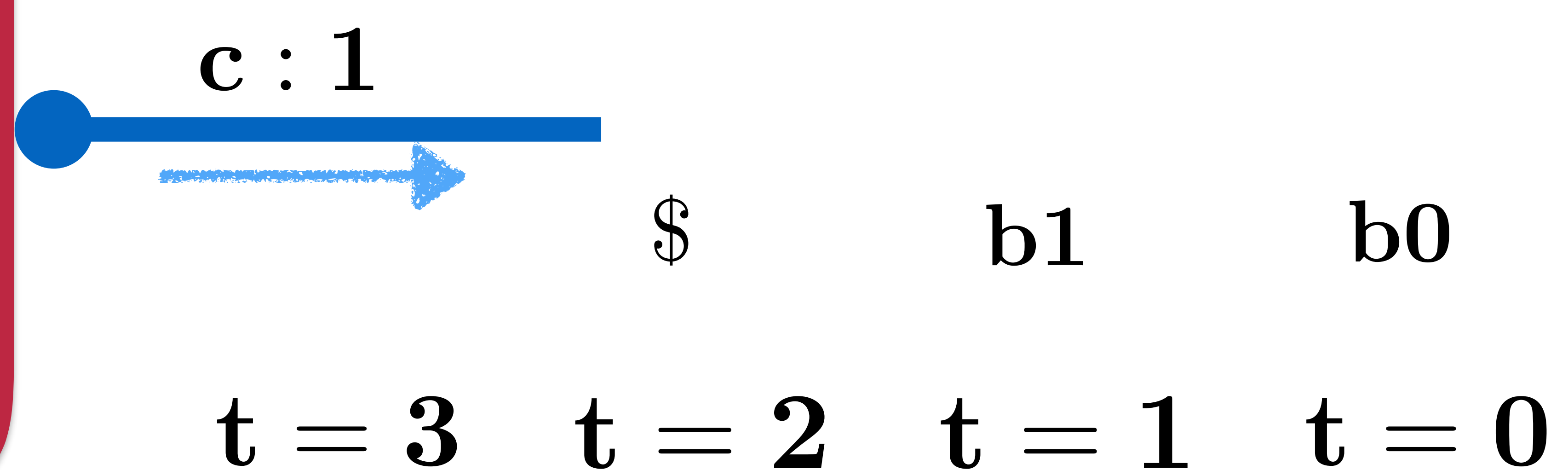
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc \mathbf{bits}, \mathbf{b1} : \bigcirc \mathbf{bits}, \$: \bigcirc \mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$; delay ;
close c



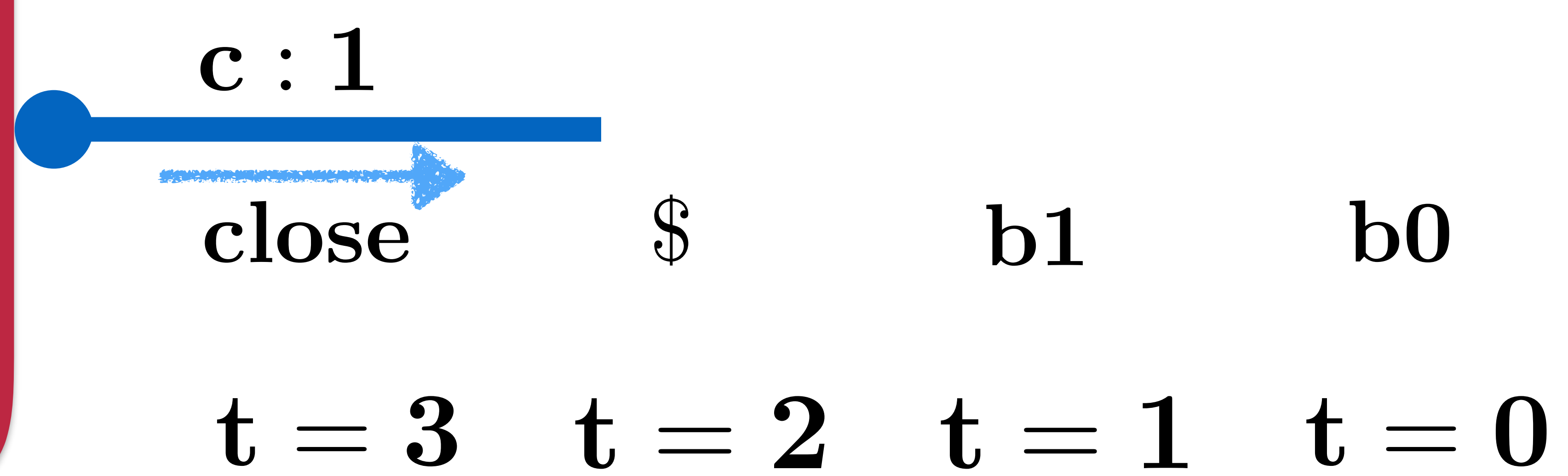
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$   ; delay ;
close c
```



Typing Rule (○)

applied pointwise

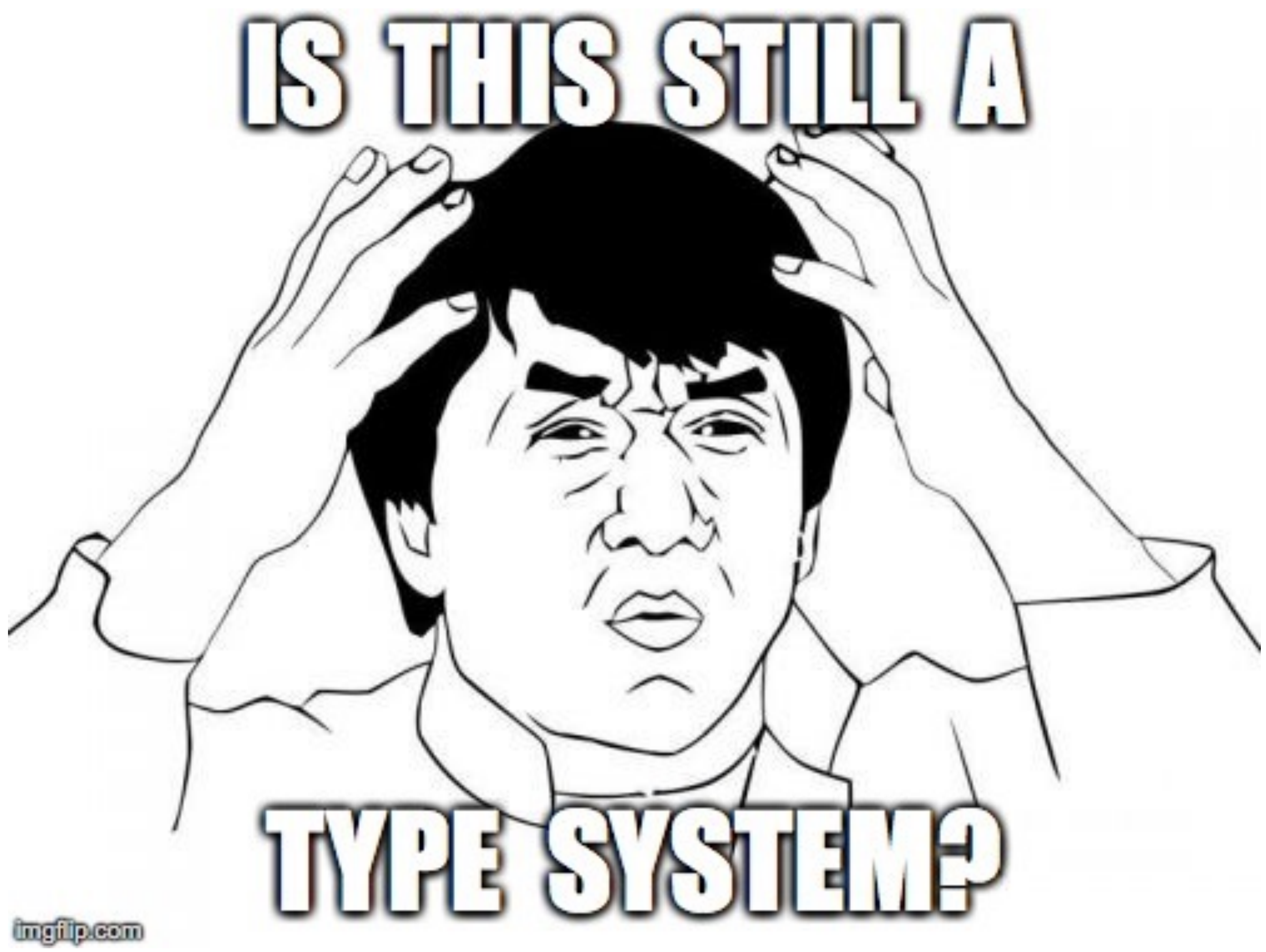
$$\frac{\mathbf{\Omega} \vdash \mathbf{P} :: (\mathbf{x} : \mathbf{S})}{\bigcirc\mathbf{\Omega} \vdash \mathbf{delay};\ \mathbf{P} :: (\mathbf{x} : \bigcirc\mathbf{S})}$$

Typing Rule (○)

applied pointwise

$$\frac{\mathbf{\Omega} \vdash \mathbf{P} :: (\mathbf{x} : \mathbf{S})}{\bigcirc\mathbf{\Omega} \vdash \mathbf{delay};\ \mathbf{P} :: (\mathbf{x} : \bigcirc\mathbf{S})}$$

breaks the locality property of type system!



Bit Streams

$\mathcal{R}$ **cost model**

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \mathbf{b1} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \$: \bigcirc^{\mathbf{r}}\mathbf{1}\}$$

Bit Streams

$\mathcal{R}$ **cost model**

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \mathbf{b1} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \$: \bigcirc^{\mathbf{r}}\mathbf{1}\}$$

$$\mathbf{x} : \mathbf{bits} \vdash \mathbf{copy} :: (\mathbf{y} : \bigcirc\mathbf{bits})$$

Bit Streams

$\mathcal{R}$ **cost model**

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \mathbf{b1} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \$: \bigcirc^{\mathbf{r}}\mathbf{1}\}$$

$$\mathbf{x} : \mathbf{bits} \vdash \mathbf{copy} :: (\mathbf{y} : \bigcirc\mathbf{bits})$$

$$\mathbf{x} : \mathbf{bits} \vdash \mathbf{plus1} :: (\mathbf{y} : \bigcirc\mathbf{bits})$$

Bit Streams

$\mathcal{R}$ **cost model**

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \mathbf{b1} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \$: \bigcirc^{\mathbf{r}}\mathbf{1}\}$$

$$\mathbf{x} : \mathbf{bits} \vdash \mathbf{copy} :: (\mathbf{y} : \bigcirc\mathbf{bits})$$

$$\mathbf{x} : \mathbf{bits} \vdash \mathbf{plus1} :: (\mathbf{y} : \bigcirc\mathbf{bits})$$

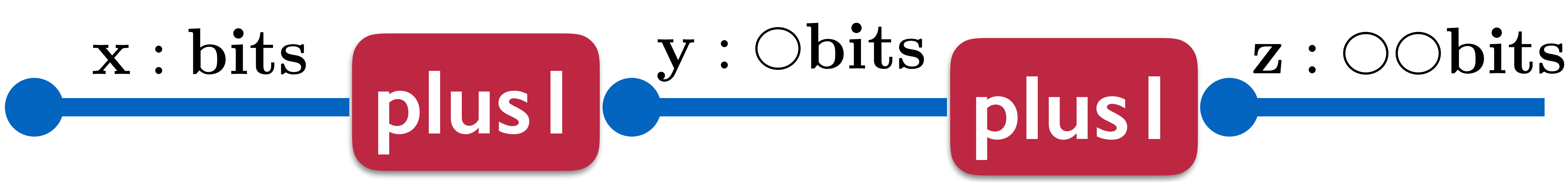
Bit Streams

$\mathcal{R}$ **cost model**

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \mathbf{b1} : \bigcirc^{\mathbf{r}}\mathbf{bits}, \$: \bigcirc^{\mathbf{r}}\mathbf{1}\}$$

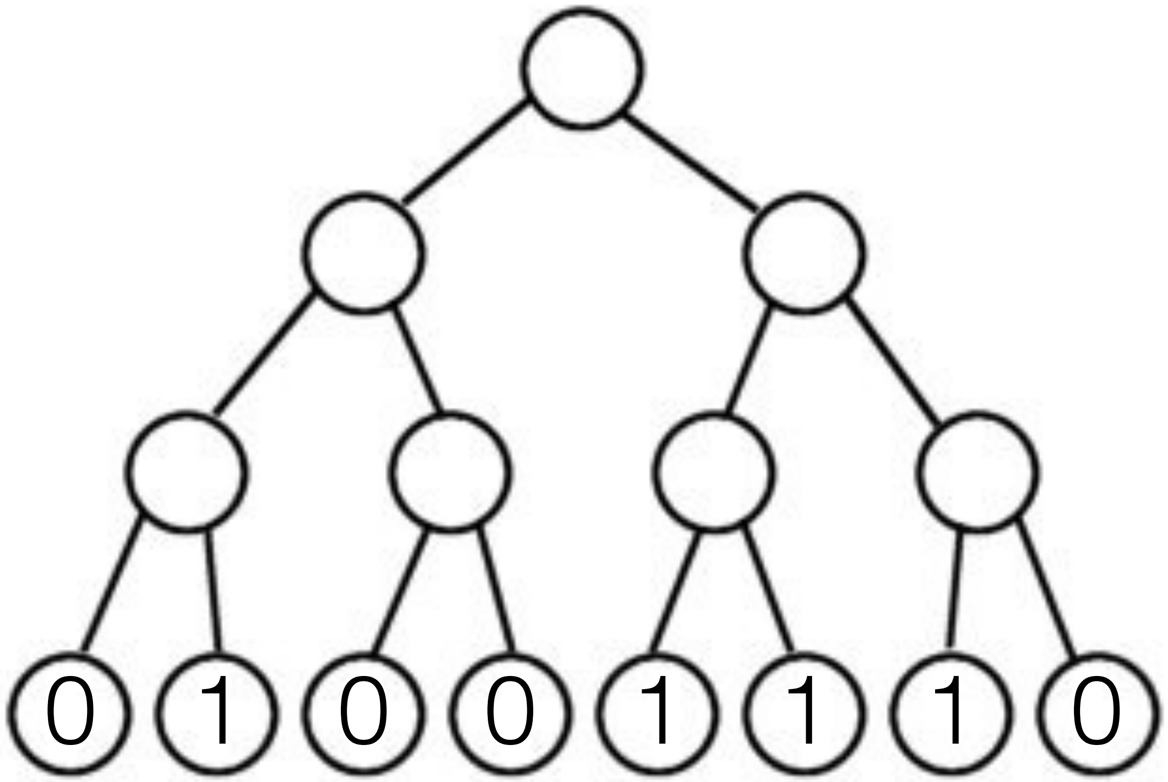
$\mathbf{x} : \mathbf{bits} \vdash \mathbf{copy} :: (\mathbf{y} : \bigcirc\mathbf{bits})$

$\mathbf{x} : \mathbf{bits} \vdash \mathbf{plus1} :: (\mathbf{y} : \bigcirc\mathbf{bits})$



$\mathbf{x} : \mathbf{bits} \vdash \mathbf{plus2} :: (\mathbf{z} : \bigcirc\bigcirc\mathbf{bits})$

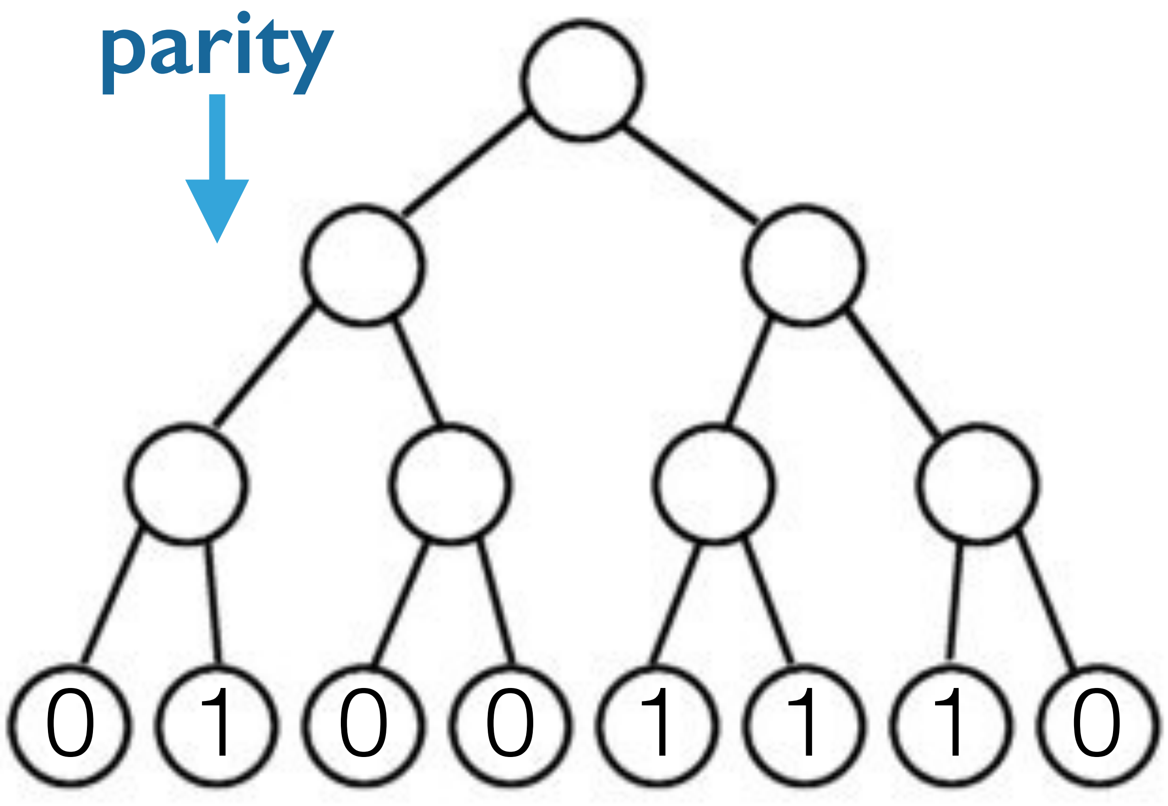
Fork-Join Parallelism



0s and 1s at the leaves

Compute the parity

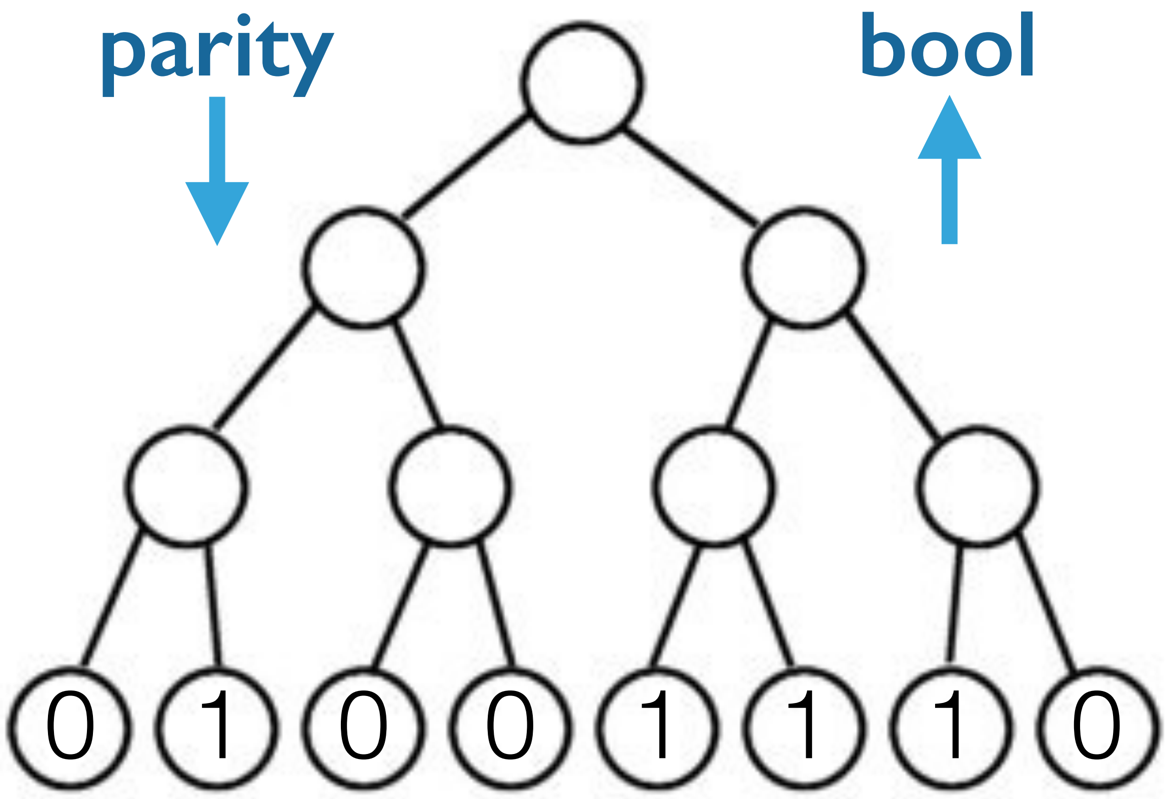
Fork-Join Parallelism



0s and 1s at the leaves

Compute the parity

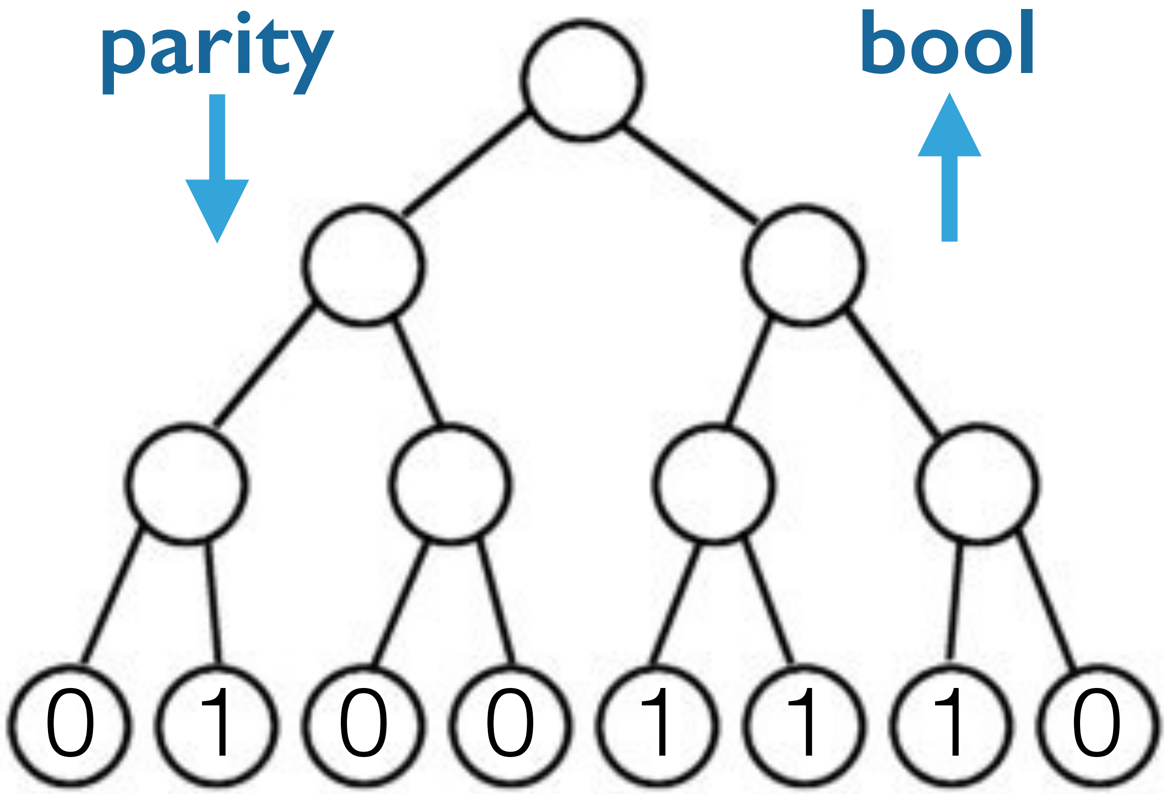
Fork-Join Parallelism



0s and 1s at the leaves

Compute the parity

Fork-Join Parallelism



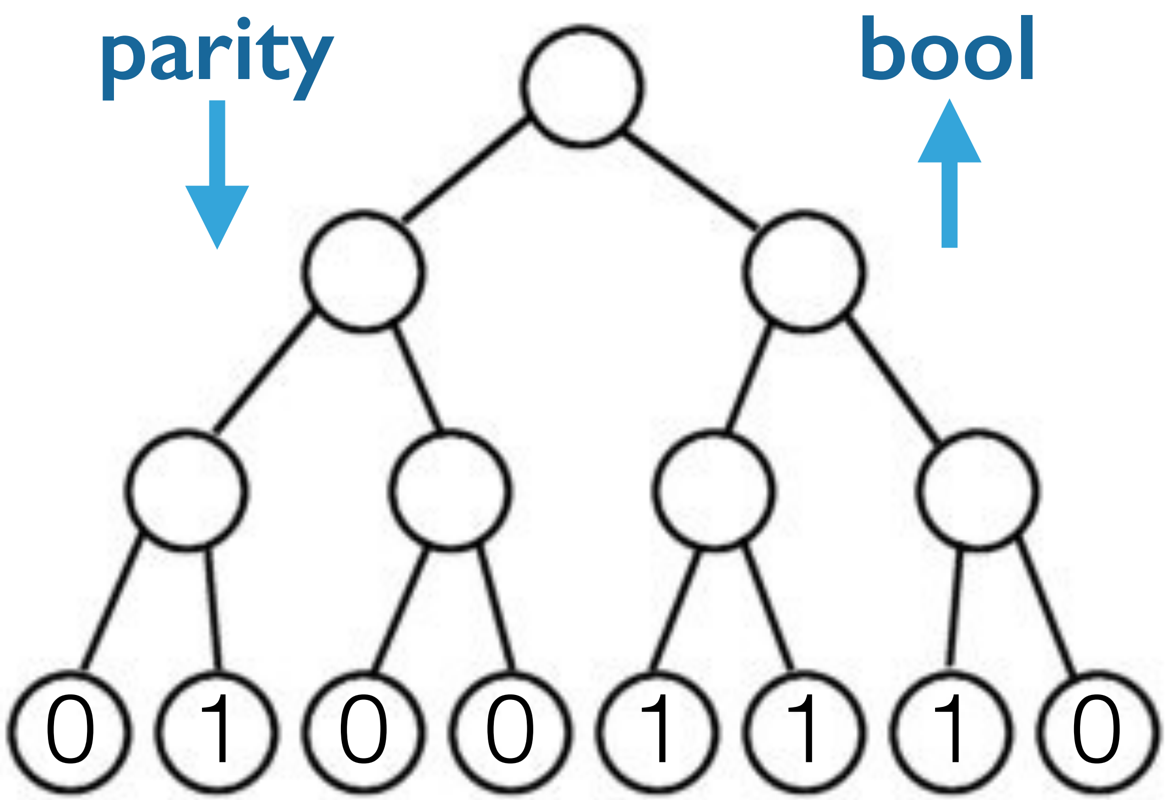
0s and 1s at the leaves

Compute the parity

$\mathcal{RS}$ cost model

$$\mathbf{tree[h]} = \&\{\mathbf{parity} : \bigcirc^{\mathbf{5h+3}}\mathbf{bool}\}$$

Fork-Join Parallelism



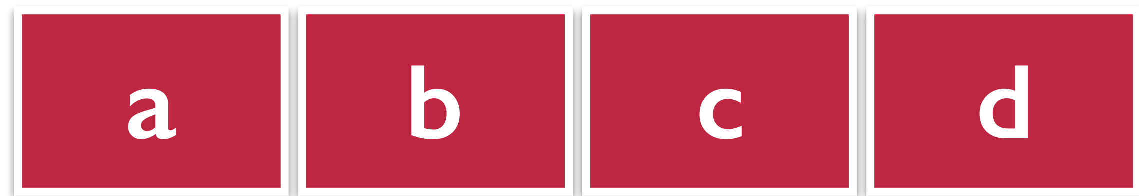
0s and 1s at the leaves

Compute the parity

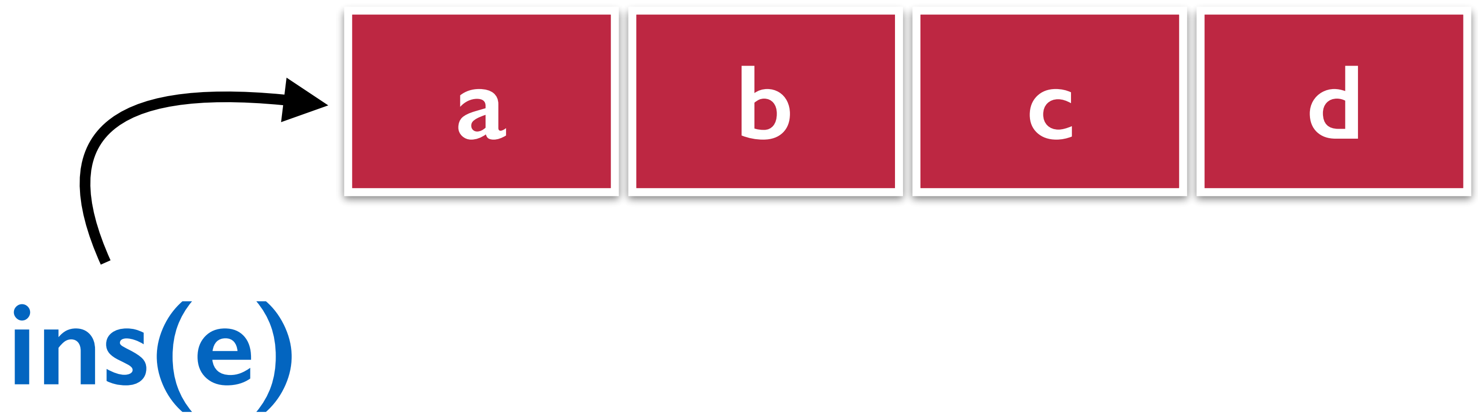
$\mathcal{RS}$ cost model $\quad \mathbf{tree[h]} = \&\{\mathbf{parity} : \bigcirc^{\mathbf{5h+3}}\mathbf{bool}\}$

Counting xors $\quad \mathbf{tree[h]} = \&\{\mathbf{parity} : \bigcirc^{\mathbf{h}}\mathbf{bool}\}$

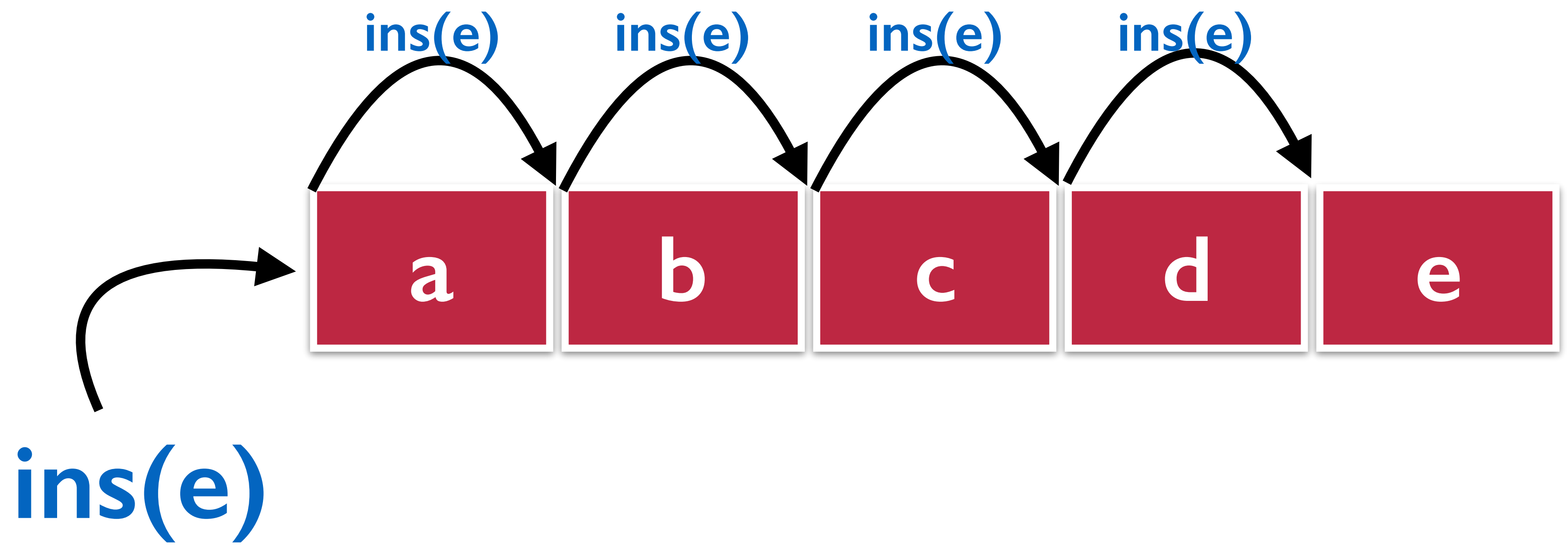
Can we type the queue?



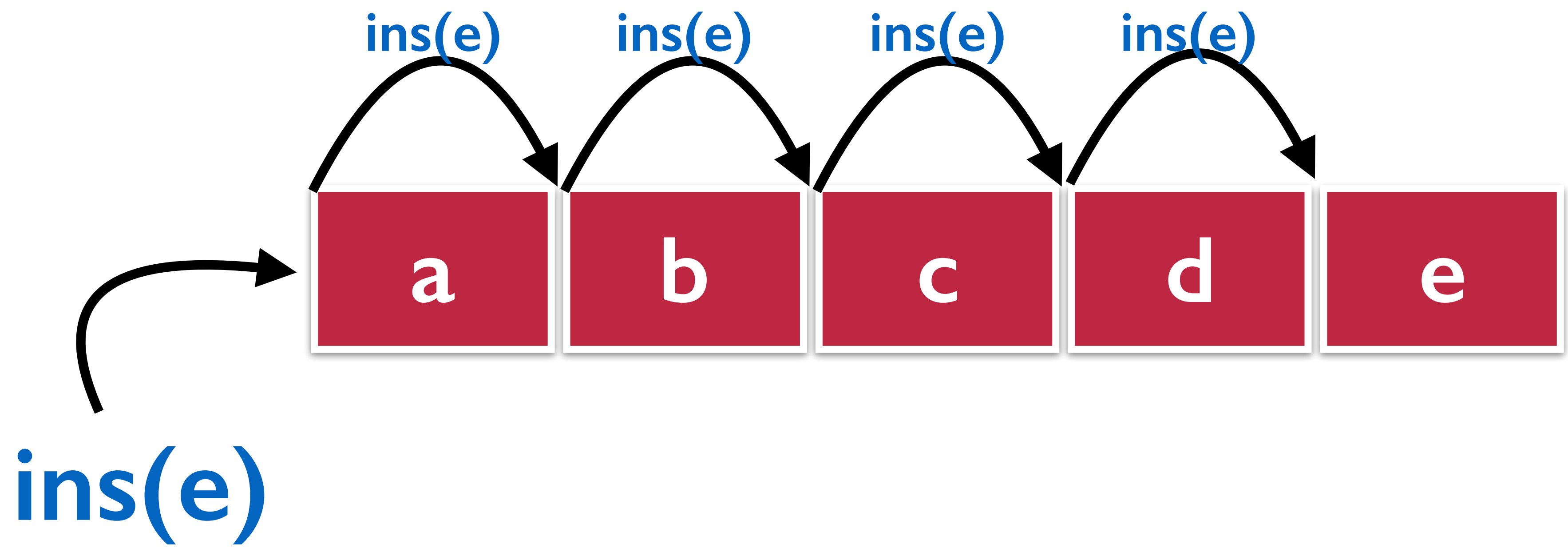
Can we type the queue?



Can we type the queue?

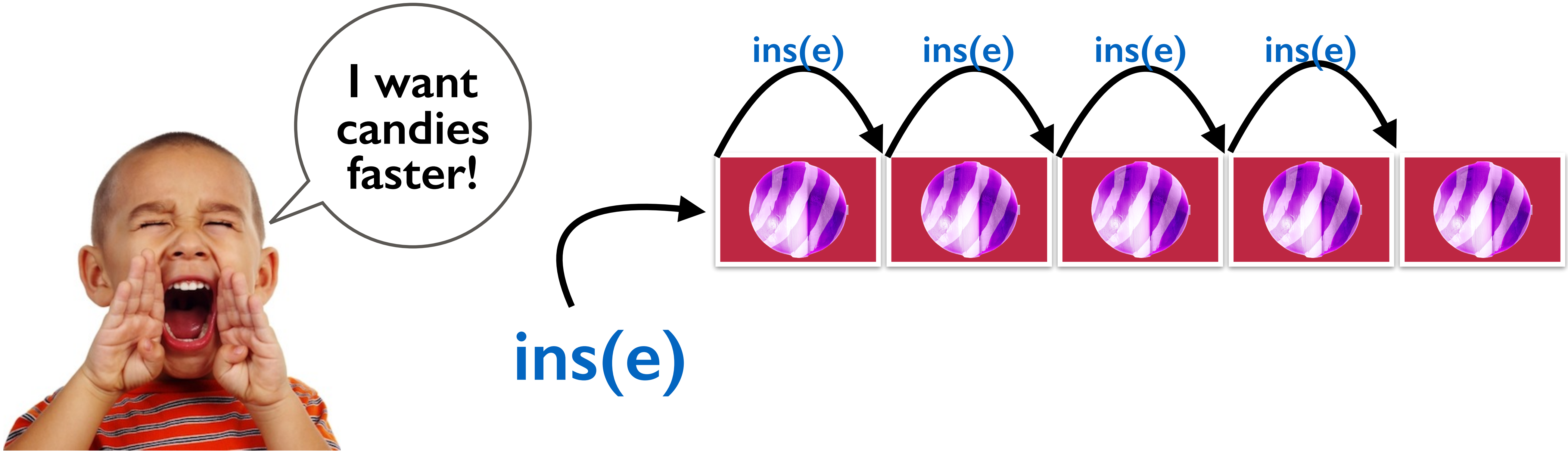


Can we type the queue?



- Next operator only expresses constant insertion rate
- But rate of insertion at the tail depends on the size of the queue — longer the queue, slower the rate
- To maintain a constant rate at the tail, new elements must be inserted at a faster rate than the previous one

Can we type the queue?



- Next operator only expresses constant insertion rate
- But rate of insertion at the tail depends on the size of the queue — longer the queue, slower the rate
- To maintain a constant rate at the tail, new elements must be inserted at a faster rate than the previous one

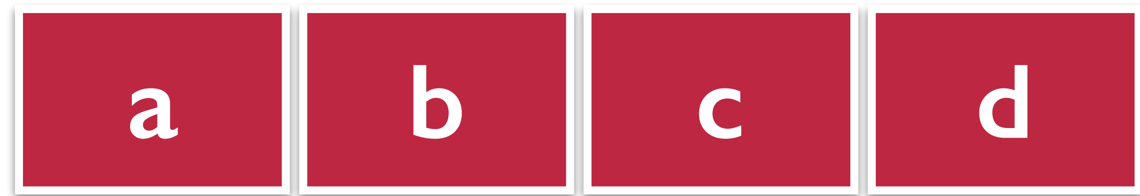
The Next Operator is too precise!

Adding Flexibility to the Type System

Providing Flexibility

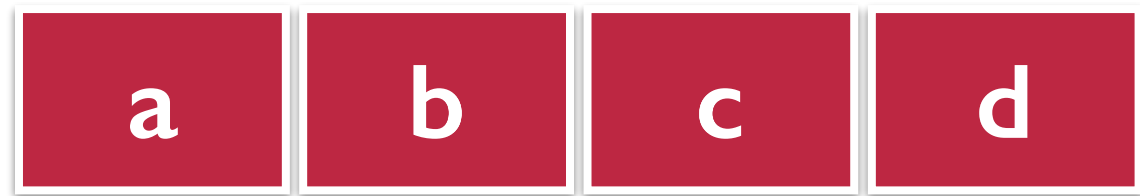
- **The Box Operator (□)**
  - Provider Action: always be ready to receive token
  - Client Action: eventually send the token
  - Provider doesn't know when the token will come, only the client does
  - Different from ◯ operator where both provider and client knew the timing of message exchange
- **The Diamond Operator (◇)**
  - Dual of the Box operator (provider and client flip)

Typing the Queue



$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

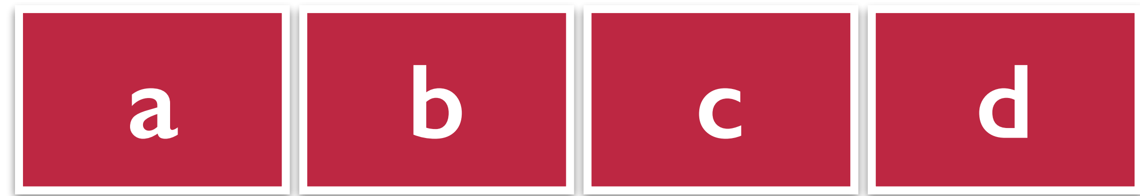
Typing the Queue



offers choice of ins/del

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\; \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\; \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

Typing the Queue

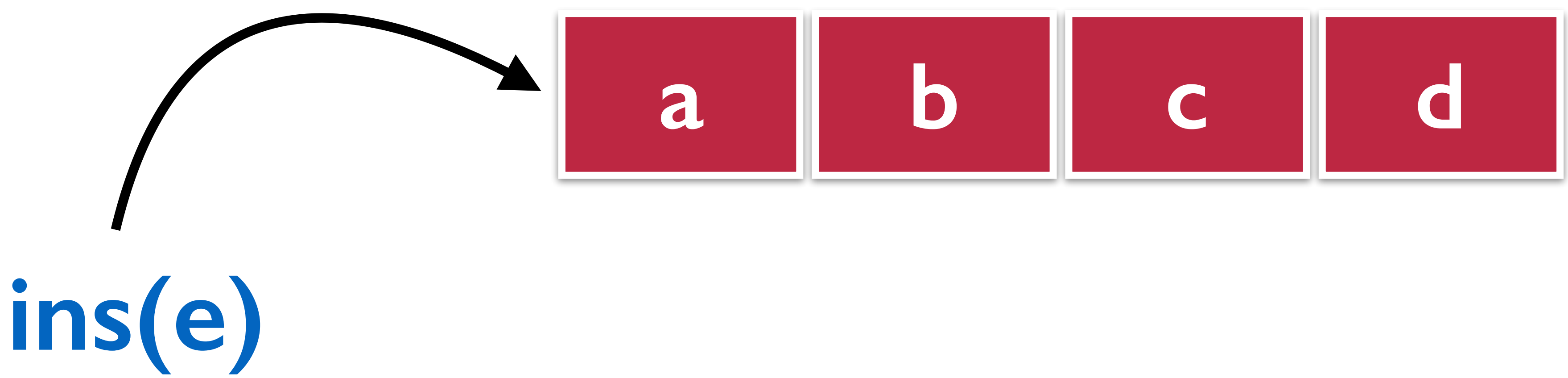


offers choice of ins/del

recv element of type A

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

Typing the Queue

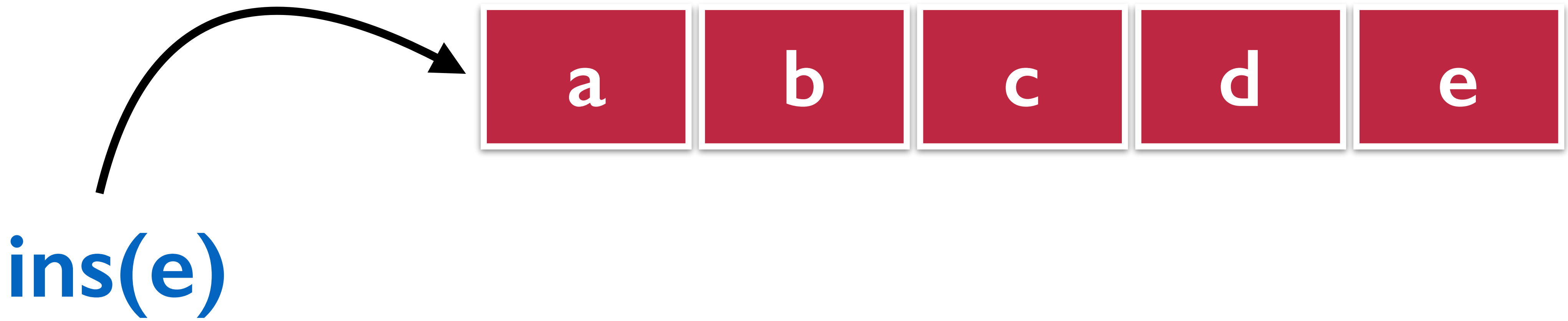


offers choice of ins/del

recv element of type A

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},\ \text{del} : \oplus\{\text{none} : \mathbf{1},\ \text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Typing the Queue

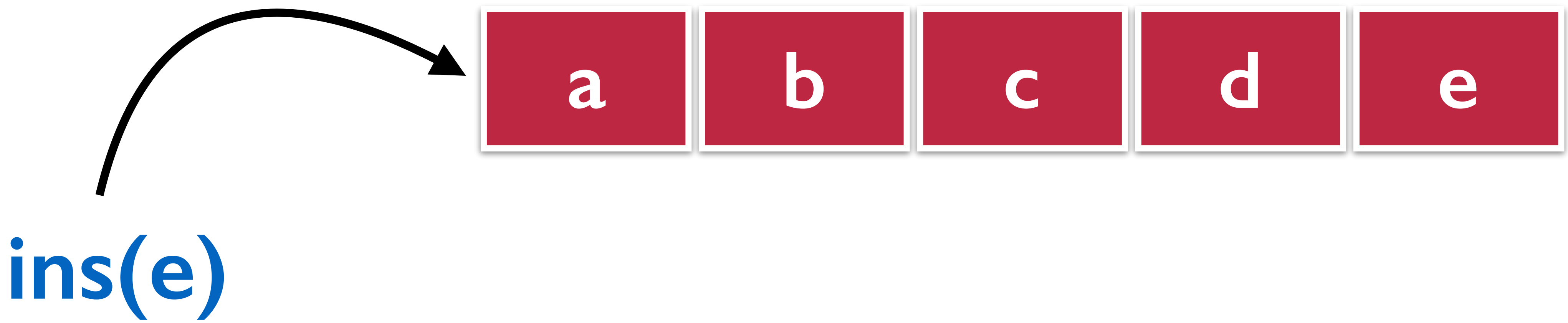


offers choice of ins/del

recv element of type A

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},$$
$$\mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

Typing the Queue



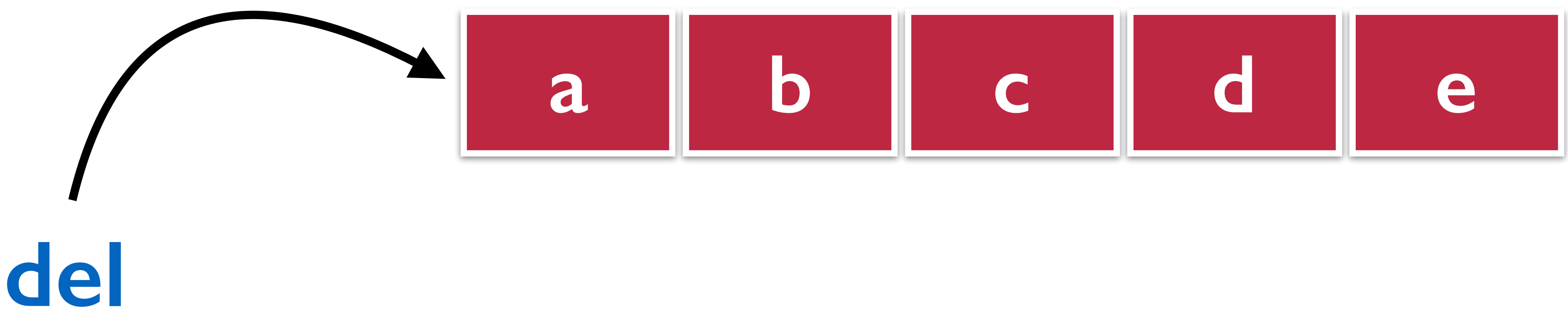
offers choice of ins/del

recv element of type A

behave as queue again

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

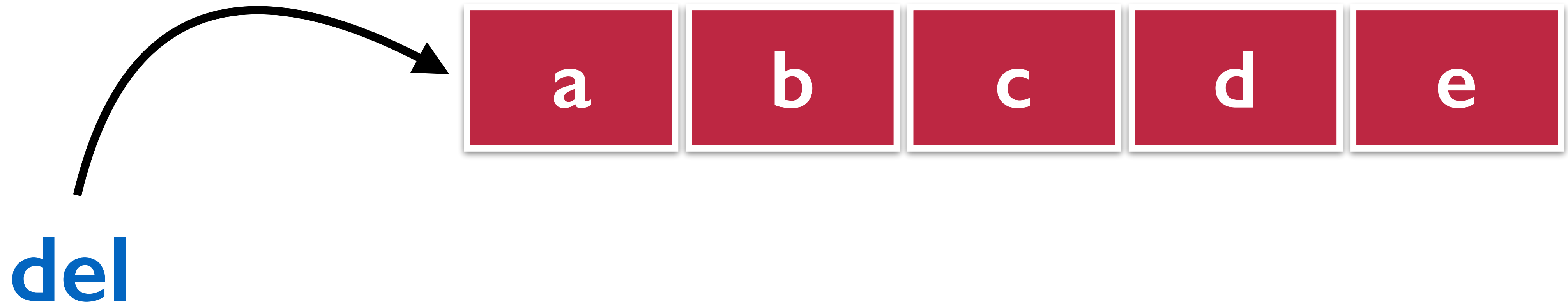
Typing the Queue



offers choice of ins/del

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},\ \text{del} : \oplus\{\text{none} : \mathbf{1},\ \text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Typing the Queue

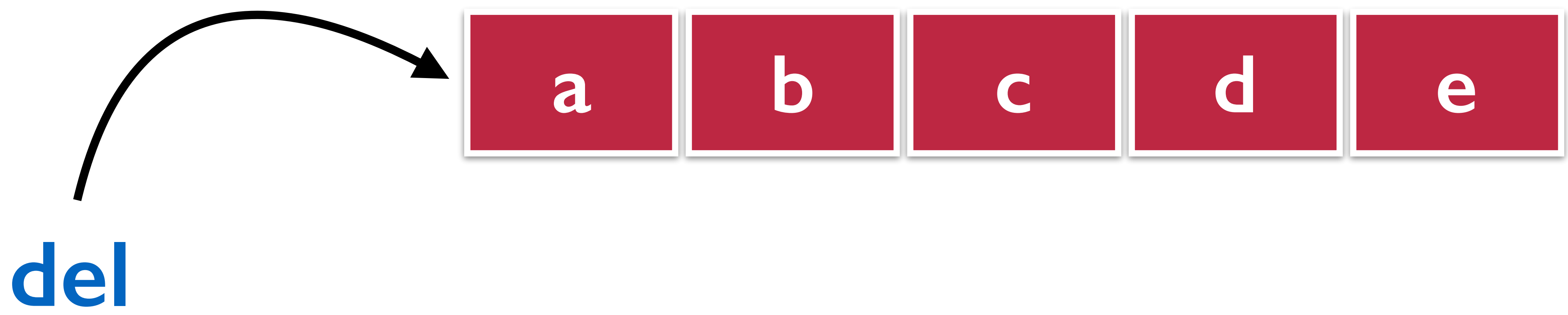


offers choice of ins/del

$$
\begin{aligned}
\text{queue}_{\mathbf{A}} = \&\{&\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}}, \\
&\text{del} : \oplus\{\text{none} : \mathbf{1}, \\
&\qquad\qquad \text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}
\end{aligned}
$$

send none if queue is empty

Typing the Queue



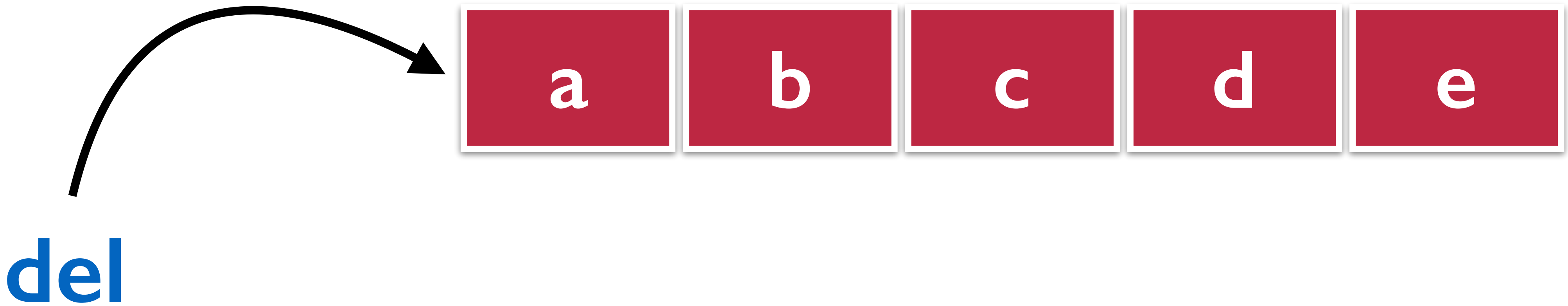
offers choice of ins/del

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},\ \text{del} : \oplus\{\text{none} : \mathbf{1},\ \text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

terminate

send none if queue is empty

Typing the Queue

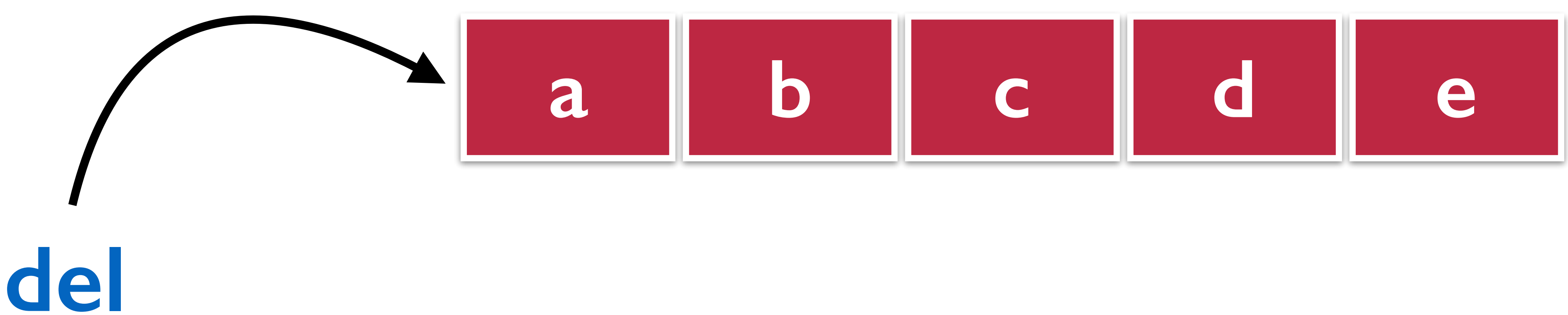


offers choice of ins/del

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

send some otherwise

Typing the Queue



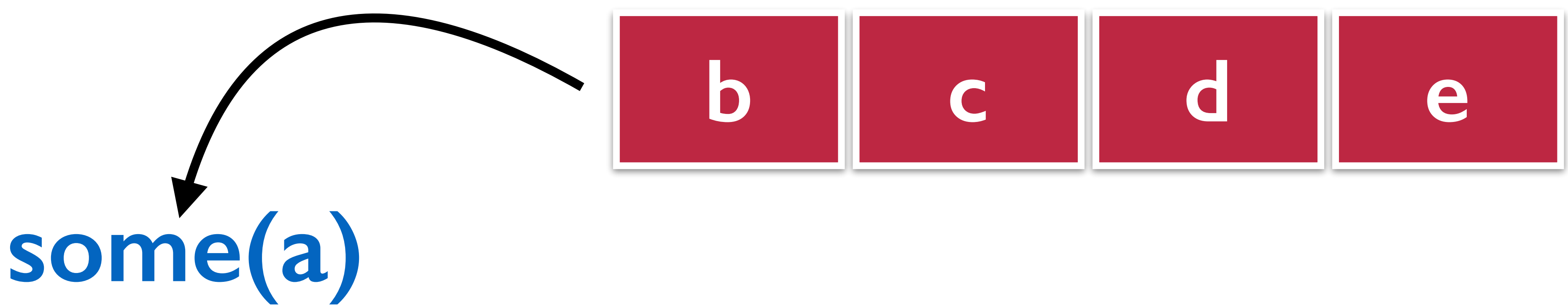
offers choice of ins/del

send element of type A

send some otherwise

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},\ \text{del} : \oplus\{\text{none} : \mathbf{1},\ \text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Typing the Queue



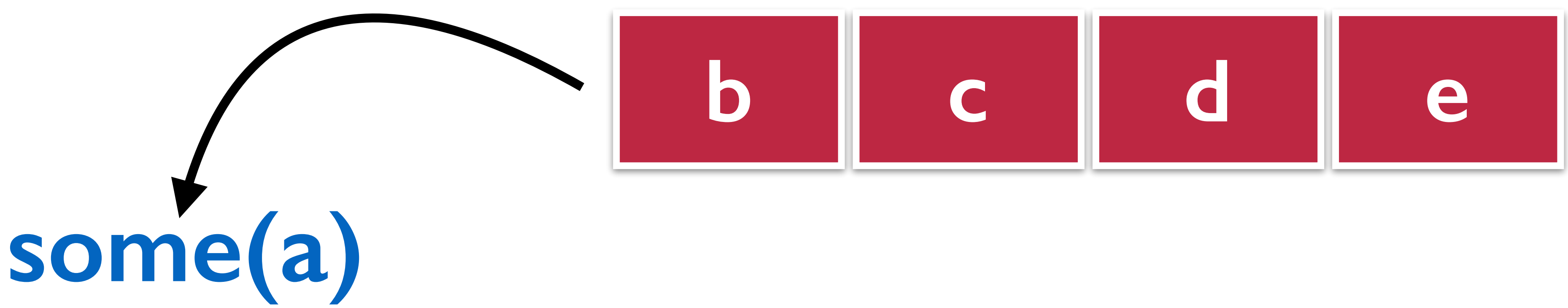
offers choice of ins/del

send element of type A

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

send some otherwise

Typing the Queue



offers choice of ins/del

send element of type A

behave as queue again

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

send some otherwise

Response Time of Queues

$$
\begin{aligned}
\mathbf{queue_A} = \Box\, \&\{&\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},\\
&\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},\\
&\qquad\qquad \mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}
\end{aligned}
$$

Response Time of Queues

$$\mathbf{queue_A} = \Box \,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response Time of Queues

$$\mathbf{queue_A} = \Box\,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response time for insertion: 3

Response Time of Queues

$$
\begin{aligned}
\mathbf{queue_A} = \Box\, \&\{\mathbf{ins} : \bigcirc(\Box \mathbf{A} \multimap \bigcirc^{\mathbf{3}} \mathbf{queue_A})\}, \\
\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc \mathbf{1}, \\
\mathbf{some} : \bigcirc(\Box \mathbf{A} \otimes \bigcirc \mathbf{queue_A})\}\}
\end{aligned}
$$

Can always accept ins/del messages

Response time for insertion: 3

Response time for deletion: 1

Response Time of Queues

$$\mathbf{queue_A} = \Box \,\&\{\mathbf{ins} : \bigcirc(\Box \mathbf{A} \multimap \bigcirc^3 \mathbf{queue_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc \mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box \mathbf{A} \otimes \bigcirc \mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response time for insertion: 3

Response time for deletion: 1

Precision

WE ARE HERE!

Flexibility

Typing Rules(□)

Typing Rules(□)

Exchanged token is a now! message

$$\frac{\Omega \text{ delayed}^{\Box} \quad \Omega \vdash P :: (x : S)}{\Omega \vdash \text{when? } x \,;\, P :: (x : \Box S)} \Box R$$

$\text{delayed}^{\Box} = \bigcirc^{*}\Box T$ → can be delayed indefinitely

Typing Rules(□)

Exchanged token is a now! message

$$\frac{\Omega\ \text{delayed}^{\Box} \qquad \Omega \vdash P :: (x : S)}{\Omega \vdash \text{when?}\ x\ ; P :: (x : \Box S)}\ \Box R$$

$\text{delayed}^{\Box} = \bigcirc^{*}\Box T$ → can be delayed indefinitely

$$\frac{\Omega, x : S \vdash Q :: (z : T)}{\Omega, x : \Box S \vdash \text{now!}\ x\ ; Q :: (z : T)}\ \Box L$$

Stacks vs Queues

$\mathcal{RS}$ **cost model**

$$\begin{aligned}\mathbf{stack_A} = \Box\ \&\{&\mathbf{ins} : \bigcirc(\Box \mathbf{A} \multimap \bigcirc \mathbf{stack_A}),\\ &\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc \mathbf{1},\\ &\qquad \mathbf{some} : \bigcirc(\Box \mathbf{A} \otimes \bigcirc \mathbf{stack_A})\}\}\end{aligned}$$

$$\begin{aligned}\mathbf{queue_A} = \Box\ \&\{&\mathbf{ins} : \bigcirc(\Box \mathbf{A} \multimap \bigcirc^{\mathbf{3}} \mathbf{queue_A}),\\ &\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc \mathbf{1},\\ &\qquad \mathbf{some} : \bigcirc(\Box \mathbf{A} \otimes \bigcirc \mathbf{queue_A})\}\}\end{aligned}$$

Stacks vs Queues

$\mathcal{RS}$ **cost model**

$$\mathbf{stack_A} = \Box \,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc\mathbf{stack_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{stack_A})\}\}$$

$$\mathbf{queue_A} = \Box \,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Which one's more efficient?

Stacks vs Queues

$\mathcal{RS}$ **cost model**

$$\mathbf{stack_A} = \Box\ \&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \underline{\bigcirc}\mathbf{stack_A}),\ \mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},\ \mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \underline{\bigcirc}\mathbf{stack_A})\}\}$$

$$\mathbf{queue_A} = \Box\ \&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \underline{\bigcirc^{3}}\mathbf{queue_A}),\ \mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},\ \mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \underline{\bigcirc}\mathbf{queue_A})\}\}$$

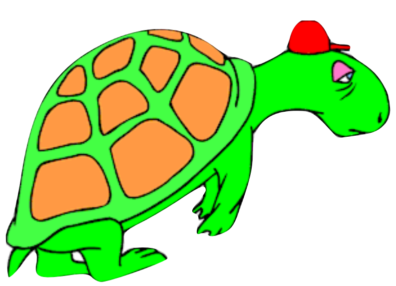
Which one's more efficient?

Stacks vs Queues

$\mathcal{RS}$ **cost model**

$$\mathbf{stack_A} = \Box\ \&\{\mathbf{ins} : \bigcirc(\Box \mathbf{A} \multimap \underline{\bigcirc}\mathbf{stack_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc \mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box \mathbf{A} \otimes \underline{\bigcirc}\mathbf{stack_A})\}\}$$

$$\mathbf{queue_A} = \Box\ \&\{\mathbf{ins} : \bigcirc(\Box \mathbf{A} \multimap \underline{\bigcirc^{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc \mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box \mathbf{A} \otimes \underline{\bigcirc}\mathbf{queue_A})\}\}$$



Which one's more efficient?

Features of Type System

- **Parametric:** time can be defined using a cost model
- **Compositional:** types describe individual processes, not just whole programs
- **Precise & Flexible:** $\bigcirc$ operator provides precision. $\Box$, $\Diamond$ operators provide flexibility
- **Conservative:** only added 3 type operators
- **General:** works on all standard examples
- **Automatic:** supports automatic type checking, type inference future work

Cost Semantics

$$\mathbf{proc}(\mathbf{c}, \mathbf{t}, \mathbf{P})$$

Process P offering along channel c at local time t

Cost Semantics

$$\mathrm{proc}(c, t, P)$$

Process P offering along channel c at local time t

Soundness Theorem:

message timings realized by the local clocks matches the timing predicted by the type system

What else is in the paper?

- Interaction of $\Box$, $\Diamond$ with $\bigcirc$ operators
- Sound and complete *subtyping* relation
- *Time Reconstruction* — inserting delay, now!, when? automatically from the program type
- *Cost Semantics* — each process stores a *local clock*, expresses timing at runtime, connected to the type system by a proof of *progress* and *preservation*
- Connection to the standard cost semantics
- Typing a set of processes at *different* local clocks

Conclusion

Conclusion

Type System

analyzes timing of message exchanges

Soundness Theorem

Cost Semantics

local clocks at each process

Properties

conservative extension, added 3 type operators

$\bigcirc$ provides precision, $\Box$, $\Diamond$ provide flexibility

Examples

throughput and latency of bit stream processors

response time of stacks vs queues

list examples: append, map, fold (many more in paper!)