Unifying Models of Data Flow

Tony Hoare

Microsoft Research

Marktoberdorf, August 2010

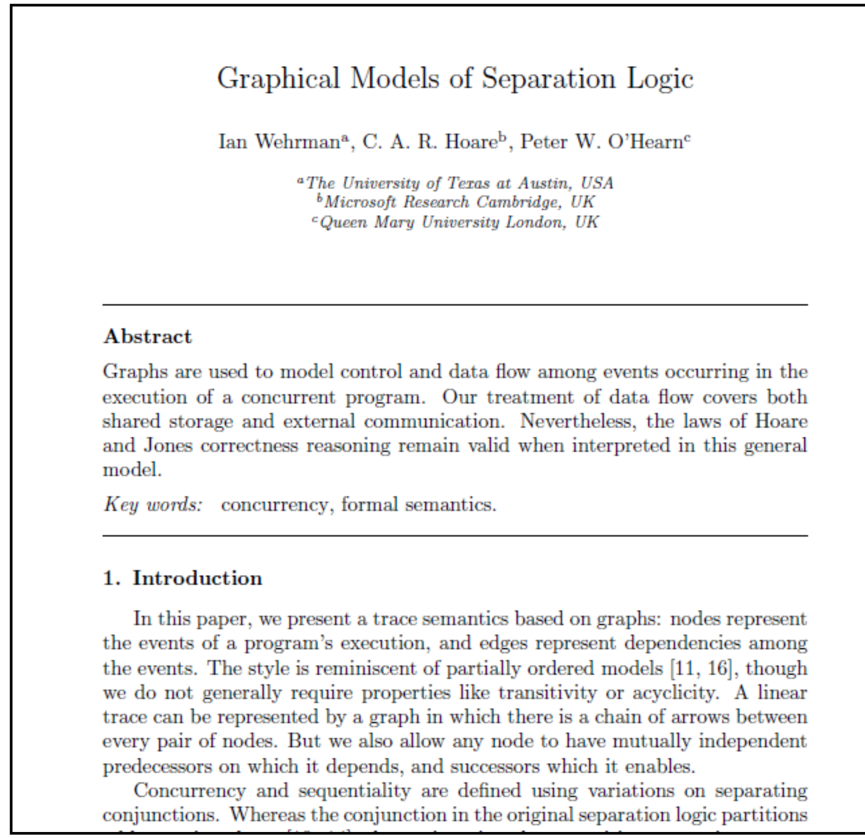
Unifying…

- Memory
  - shared/private, weakly/strongly consistent
- Communication
  - synchronised/buffered, reliable/unreliable
- Allocation
  - dynamic/nested, disposed/collected
- Concurrency
  - threads/processes, coarse/fine-grained

Reference

Unifying Models of Control Flow

- Ian Wehrman, C.A.R. Hoare and Peter O'Hearn. Graphical Models of Separation Logic. In *Engineering Methods and Tools for Software Safety and Security*, M. Broy et al. (eds.), IOS Press, pp. 177-202, 2009.


Graphical Models of Separation Logic

Ian Wehrman[a], C. A. R. Hoare[b], Peter W. O'Hearn[c]

[a] *The University of Texas at Austin, USA*
[b] *Microsoft Research Cambridge, UK*
[c] *Queen Mary University London, UK*

Abstract

Graphs are used to model control and data flow among events occurring in the execution of a concurrent program. Our treatment of data flow covers both shared storage and external communication. Nevertheless, the laws of Hoare and Jones correctness reasoning remain valid when interpreted in this general model.

Key words: concurrency, formal semantics.

1. Introduction

In this paper, we present a trace semantics based on graphs: nodes represent the events of a program's execution, and edges represent dependencies among the events. The style is reminiscent of partially ordered models [11, 16], though we do not generally require properties like transitivity or acyclicity. A linear trace can be represented by a graph in which there is a chain of arrows between every pair of nodes. But we also allow any node to have mutually independent predecessors on which it depends, and successors which it enables.

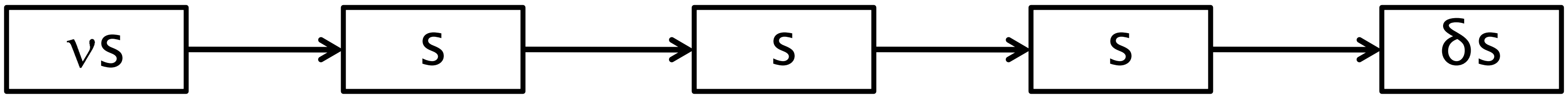
Concurrency and sequentiality are defined using variations on separating conjunctions. Whereas the conjunction in the original separation logic partitions

The behaviour of a resource

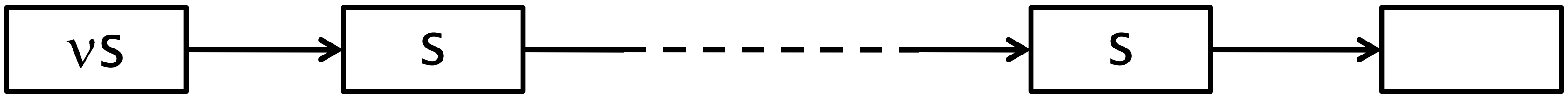
- ...is recorded as a trace of all events in which it has engaged...
  - drawn as boxes 
- ...with direct dependencies between them...
  - drawn as arrows 

  - target cannot occur without/before source
- ...along which data may flow

A sequential trace



The sequential design pattern



Defined in relational algebra as:

$$\nu\ (s \rightarrow s)^{+}\ \delta$$

A Graph is

□ the carrier set of events

$\longrightarrow$ $\subseteq$ □ × □ a relation between events

s, c, x, δ, … $\subseteq$ □ a collection of subsets of events

m and n are relations

- e (m) f means (e, f) $\in$ m
- m n is their relational composition
  - e (m n) f $\triangleq$ $\exists$g. e (m) g & g (n) f
- the identity relation is defined by
  - e (Id) f $\triangleq$ (e = f)
- the universal relation is defined by
  - e **U** f $\triangleq$ true

Relational operators

- Converse: $e \leftarrow f \triangleq f \rightarrow e$, or:

  $$e\ (m^{\cup})\ f \triangleq f\ (m)\ e$$

- Kleene star: $\leq \ \triangleq\ (\rightarrow)^*$, where
  - $(m)^* = Id \cup (m) \cup (m\ m) \cup (m\ m\ m) \cup \ldots$
  - $(m)^+ = (m) \cup (m\ m) \cup (m\ m\ m) \cup \ldots$

Relational properties

- If $\rightarrow$ is acyclic, $\leq$ is antisymmetric:

  $(\leq \cap \geq) \subseteq \mathrm{Id}$

  or, in predicate calculus:

  $\forall e,f.\ e \leq f \ \&\ e \geq f \Rightarrow e = f$

- If m is a (partial) function then:

  $m^{\cup}\ m \subseteq \mathrm{Id}$

  or, in predicate calculus:

  $\forall e,f,g.\ e\ (m)\ f \ \&\ e\ (m)\ g \Rightarrow f = g$

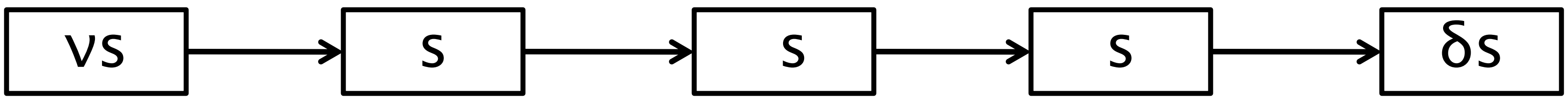
A Resource

- ...is represented by the set of events in which it has engaged
- We use set-valued variables to range over resources
  - c, d, … (channels)
  - x, y, … (variables)
  - r, s, … (etc.)

Sets and relations

- A set of events s is represented as a relation: $e\ (s)\ f \triangleq e \in s\ \&\ e = f$
- Set intersection corresponds to relational composition: $s \cap t = st$
- $(s \rightarrow t)$ is a relation containing all arrows with source in s and target in t
- We can lift relations to sets:
  $s\ [m]\ t \triangleq \forall e,f.\ e \in s\ \&\ f \in t \Rightarrow e\ (m)\ f$

A sequential trace



- Each event in s has at most one successor and one predecessor :
  $(s \leftarrow s \rightarrow s) \subseteq Id$ and $(s \rightarrow s \leftarrow s) \subseteq Id$

- Dependency is a total order on s :
  $(s \leftarrow s)^* \cup (s \rightarrow s)^* = s\ \mathbf{U}\ s$

Sets of events

ν	allocate	δ	dispose
$:=$	assign	$=:$	fetch
!	output	?	input
$\Downarrow$	acquire	$\Uparrow$	release

Conjunction

- Suppose w is a data value. Then:

x :=	the set of all assignments to x
x =:	the set of all fetches of x
x := w	the set of assignments of w to x
x =: w	the set of fetches of w from x
νs	the set of all allocations of s
δs	the set of all disposals of s

Allocation and disposal

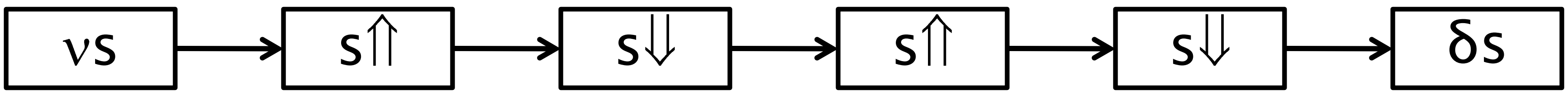
- Allocation is the first event of a resource s: $\nu s \; [\leq] \; s$
- Each resource s has one allocation event: $|\nu s| = 1$
- Disposal is similar

Implementation

- Allocation may be:
  - global,
  - on stack, or
  - on heap

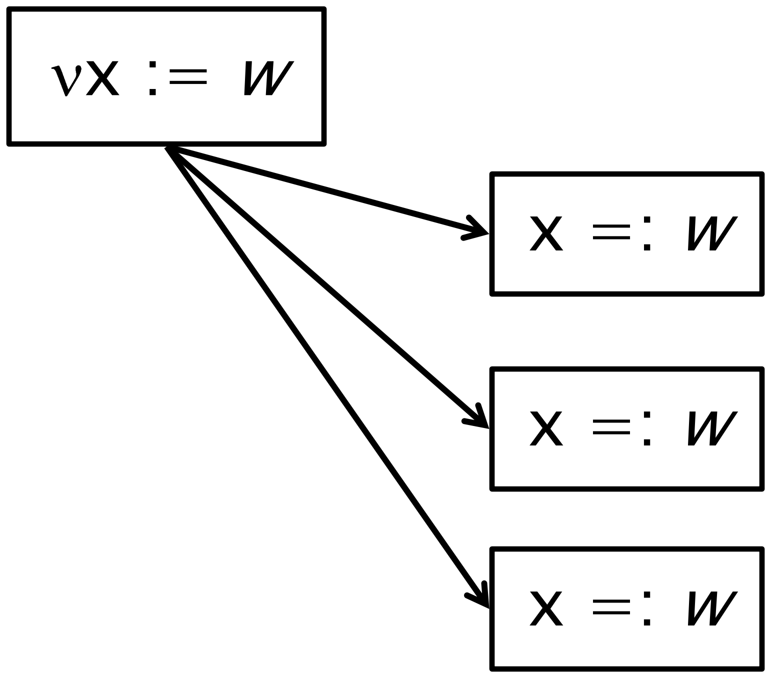
- Disposal may be:
  - from stack,
  - by mark-and-sweep garbage collection,
  - by operating system,
  - by switching off, or
  - by ultimate disposal of hardware

A semaphore s



$$
\begin{aligned}
s \to s \;\subseteq\; & (\nu \to \Uparrow) \cup \\
& (\Uparrow \to \Downarrow) \cup \\
& (\Downarrow \to \Uparrow) \cup \\
& (\Downarrow \to \delta) \cup \\
& (\nu \to \delta)
\end{aligned}
$$

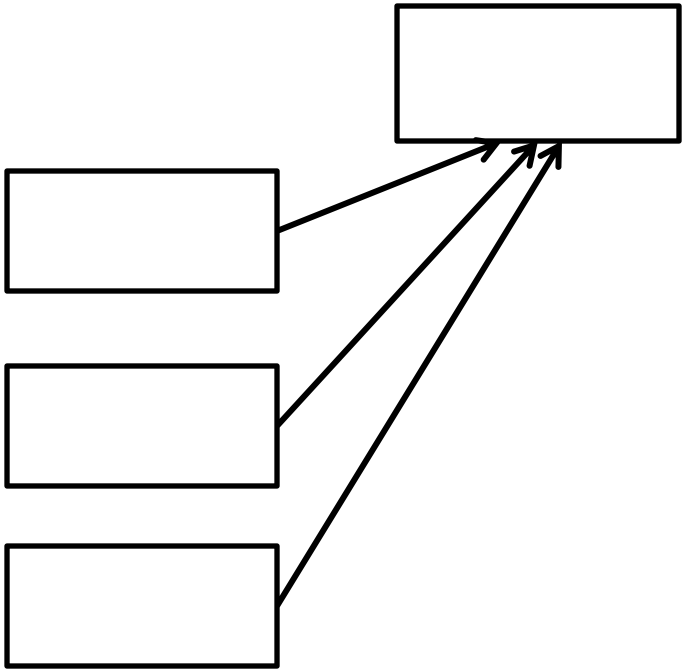
A parameter x



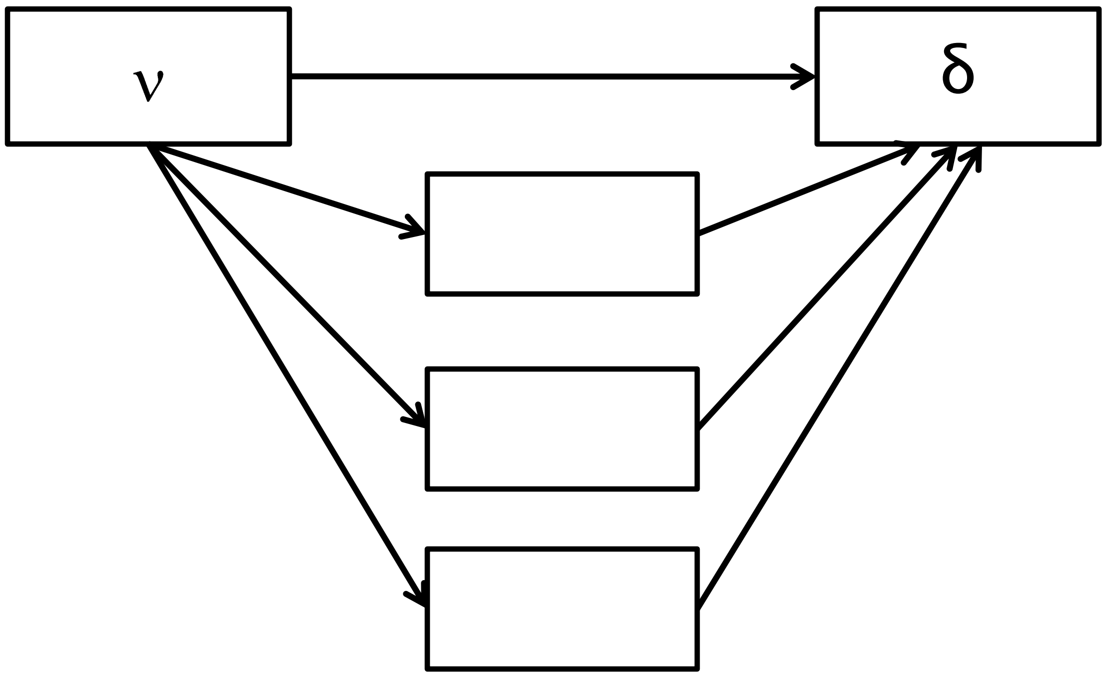
$$x \rightarrow x \leftarrow x \subseteq Id$$

$$x \rightarrow x \subseteq \{(e,f) \mid \exists w.\ e \in (\nu x := w) \ \&\ f \in (x =: w)\}$$

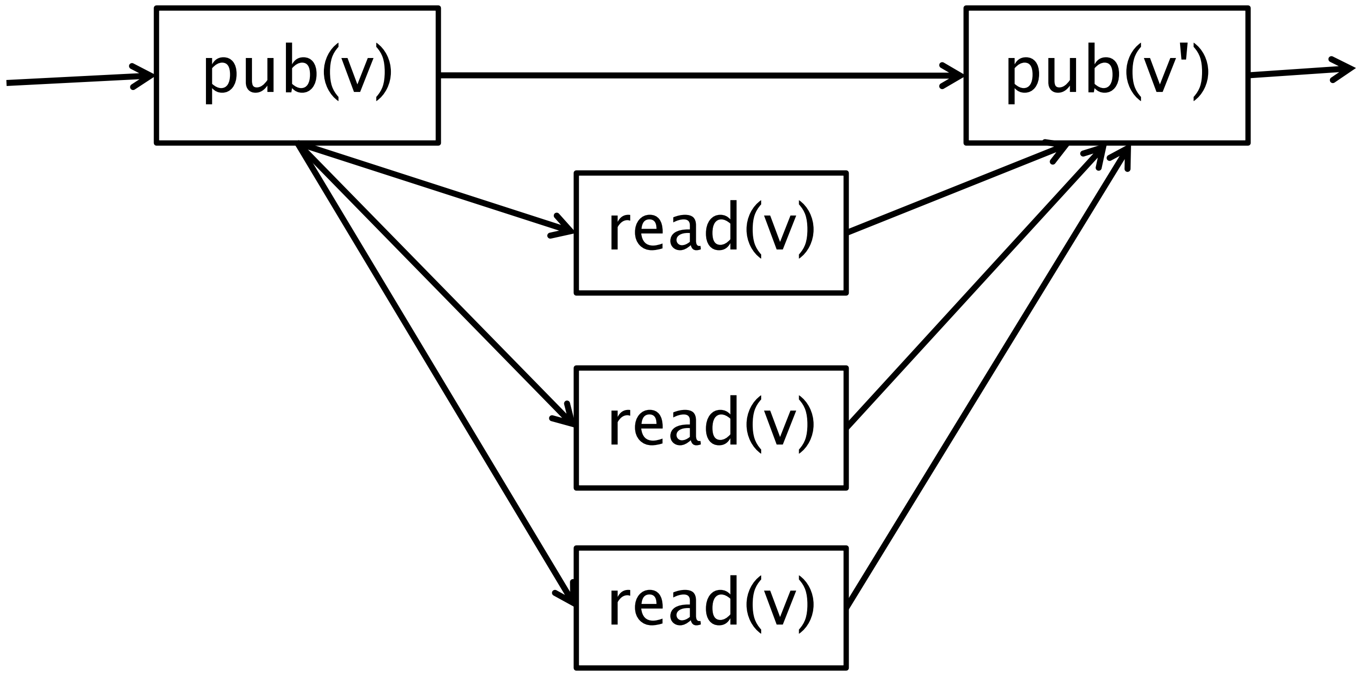
Fan-in



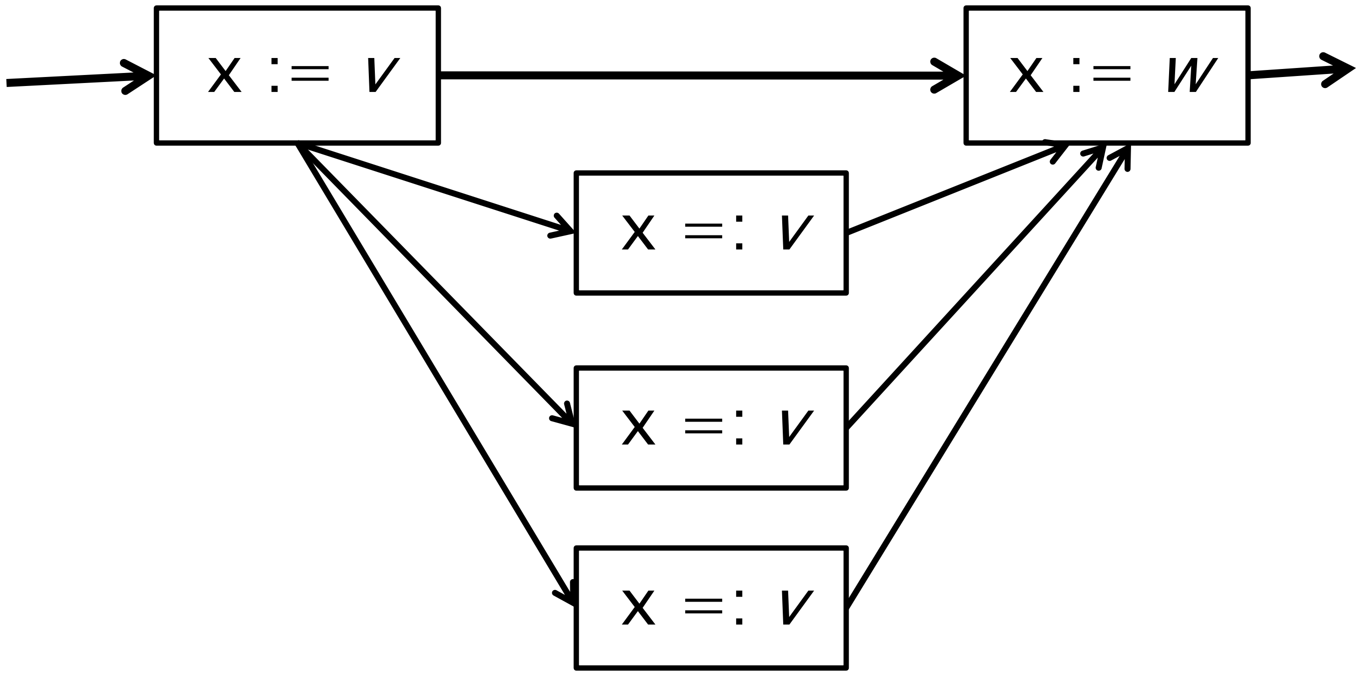
Concurrent Resource



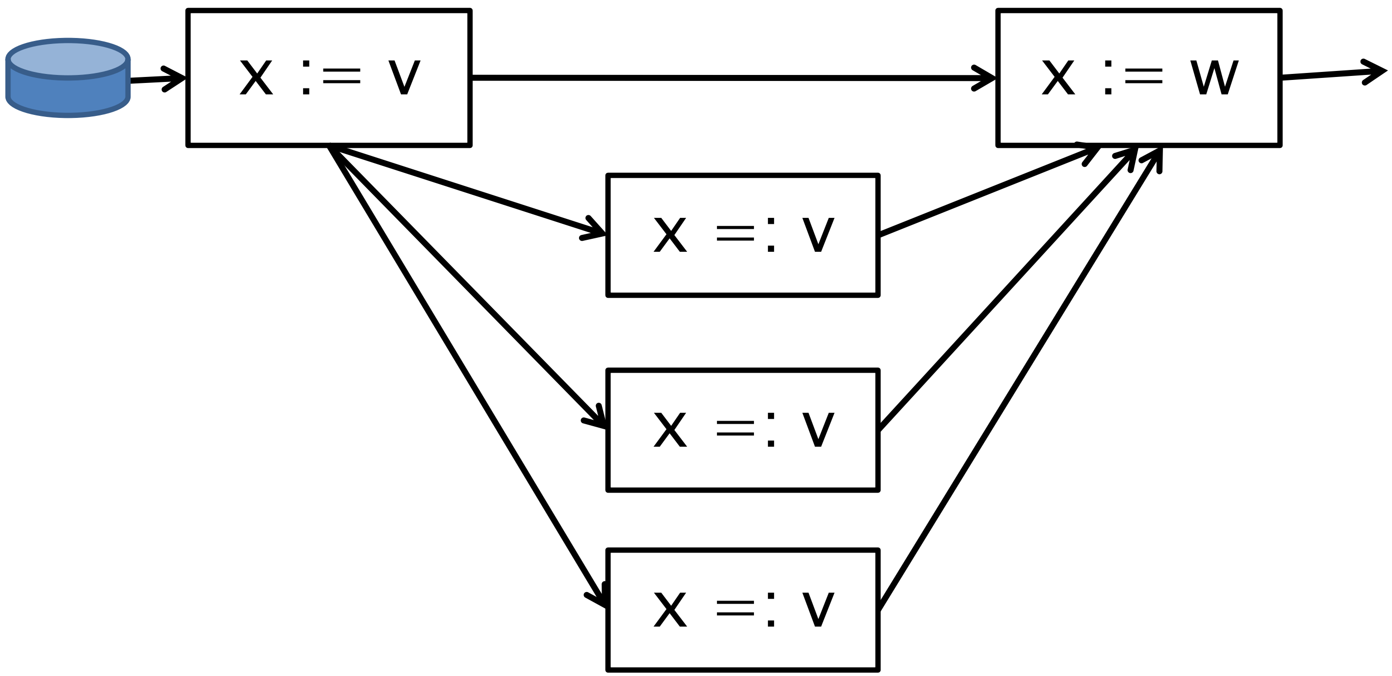
Publication



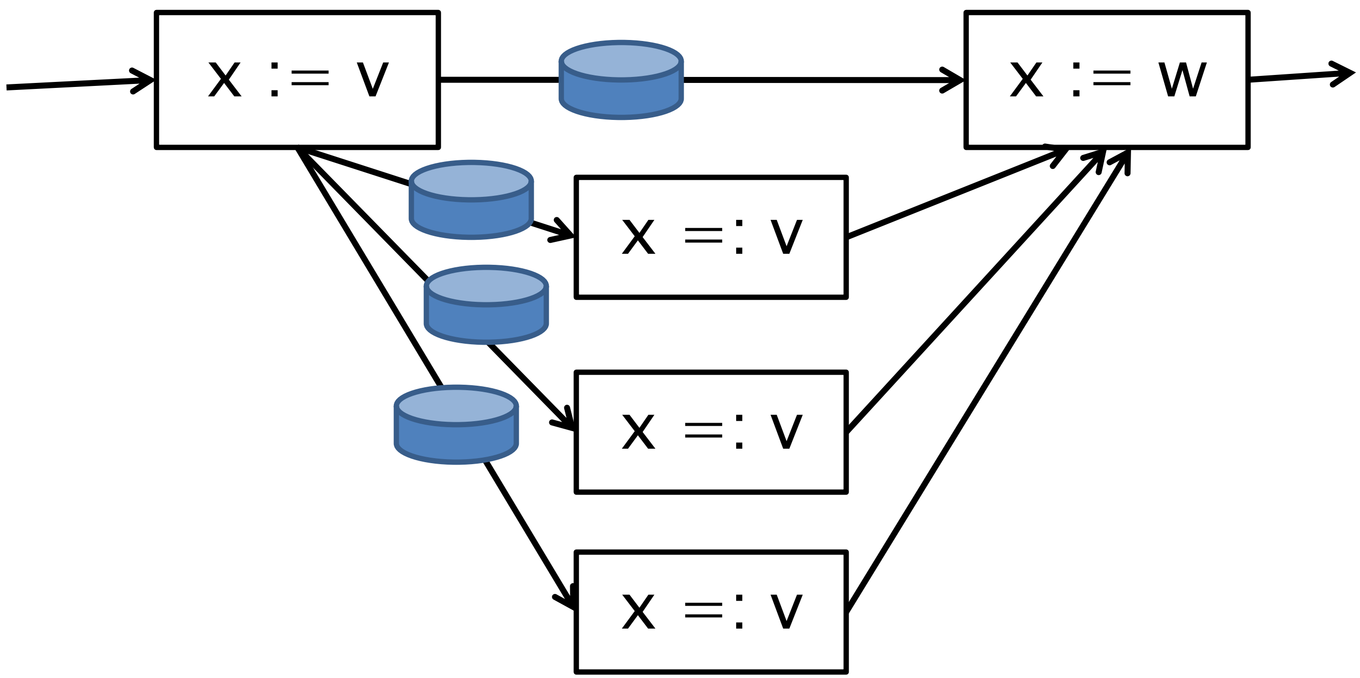
Assignment



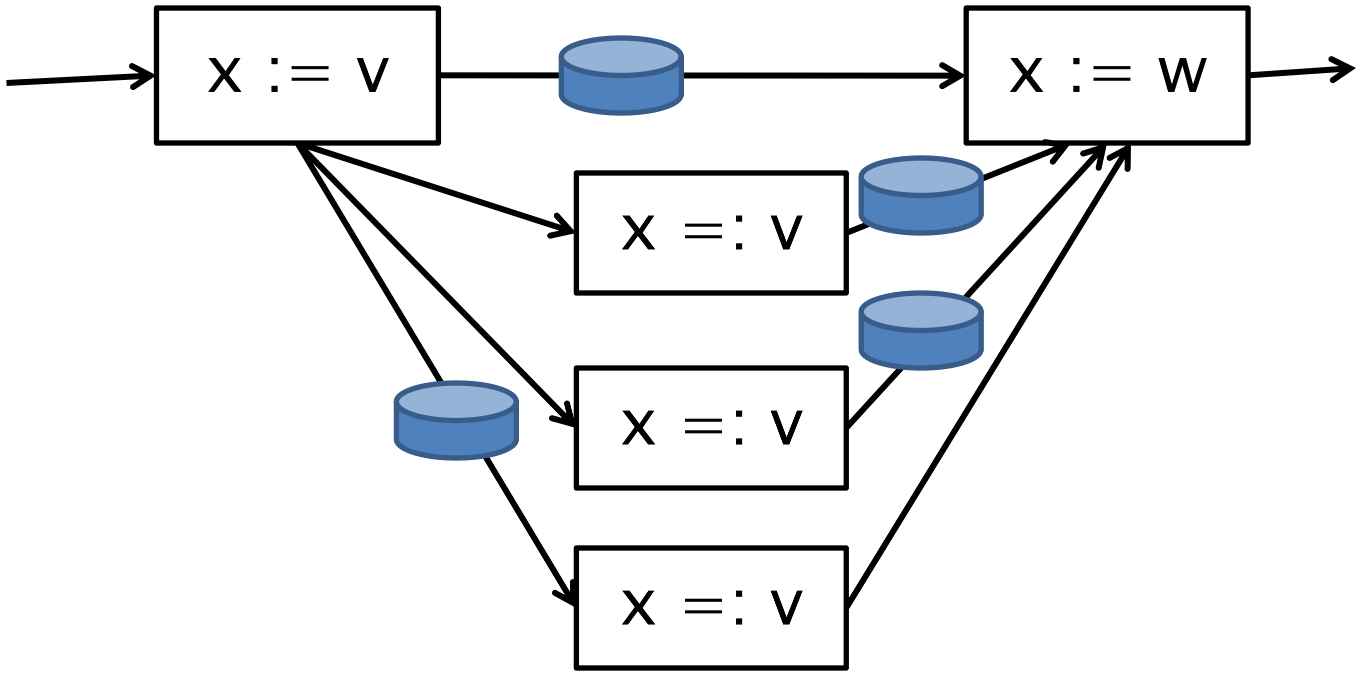
The token game (1)



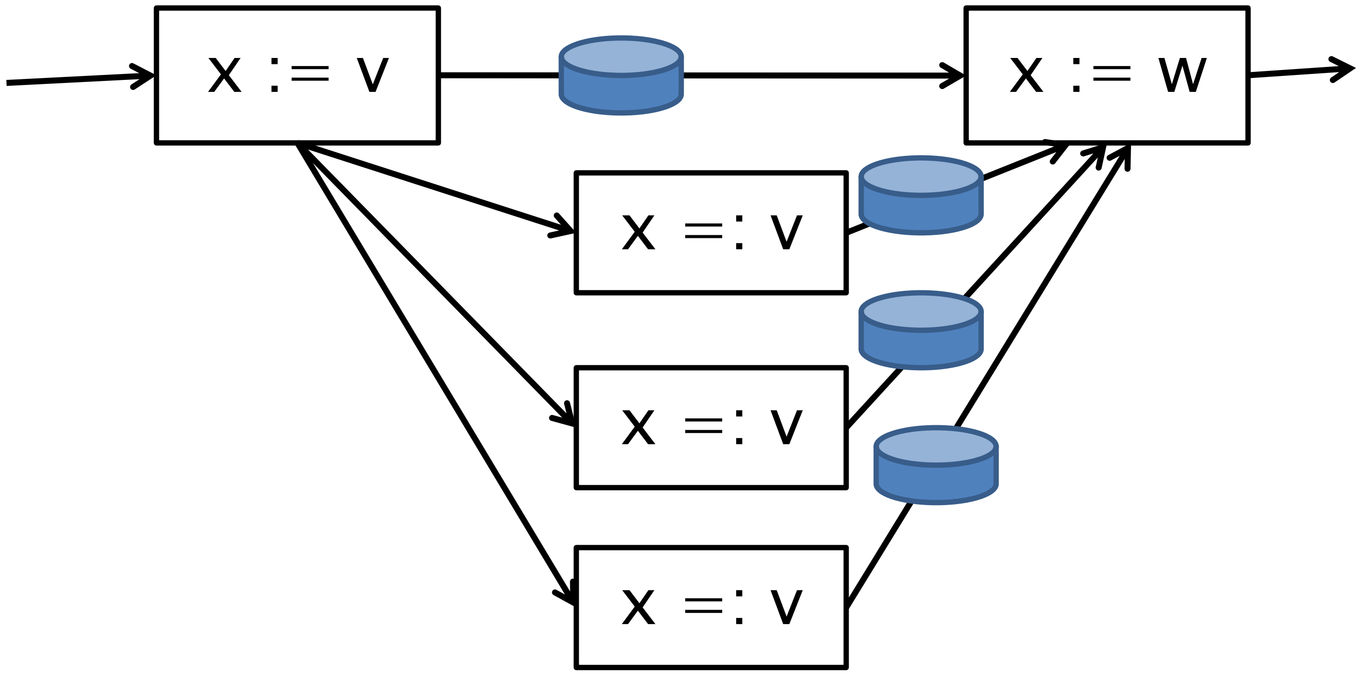
The token game (2)



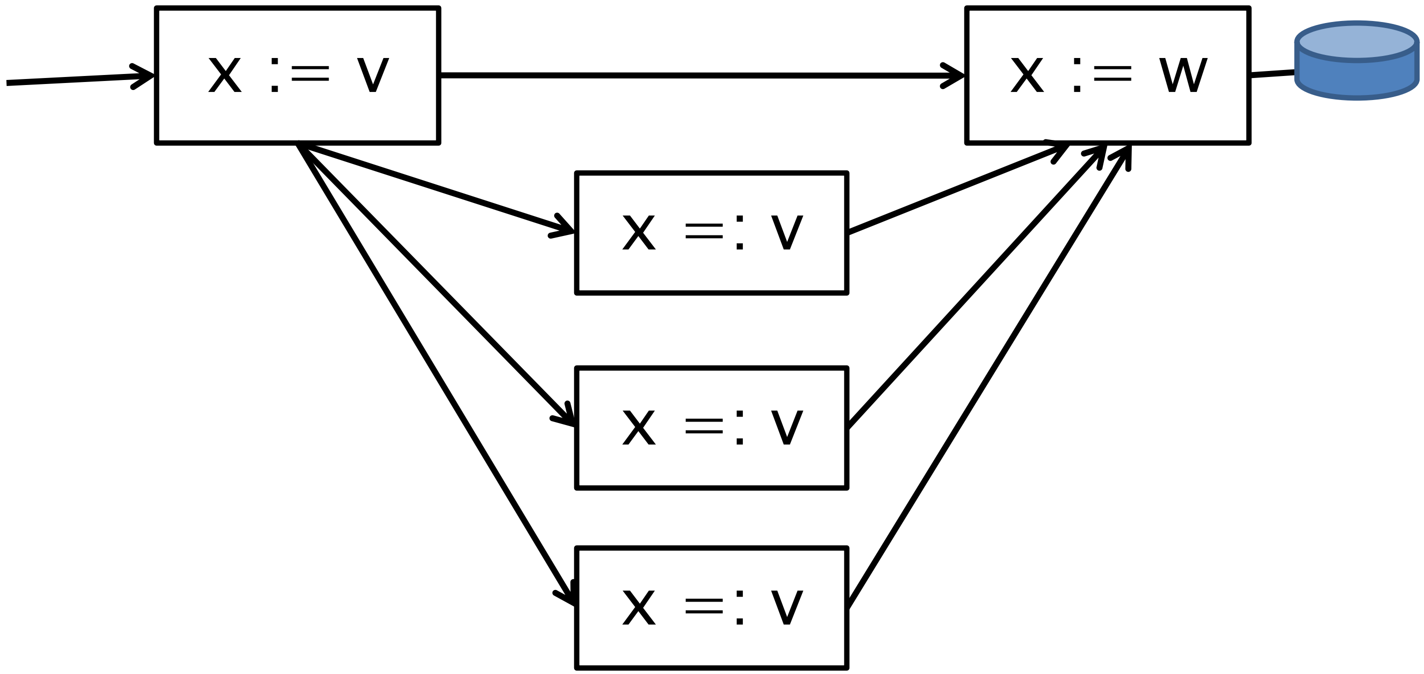
The token game (3)



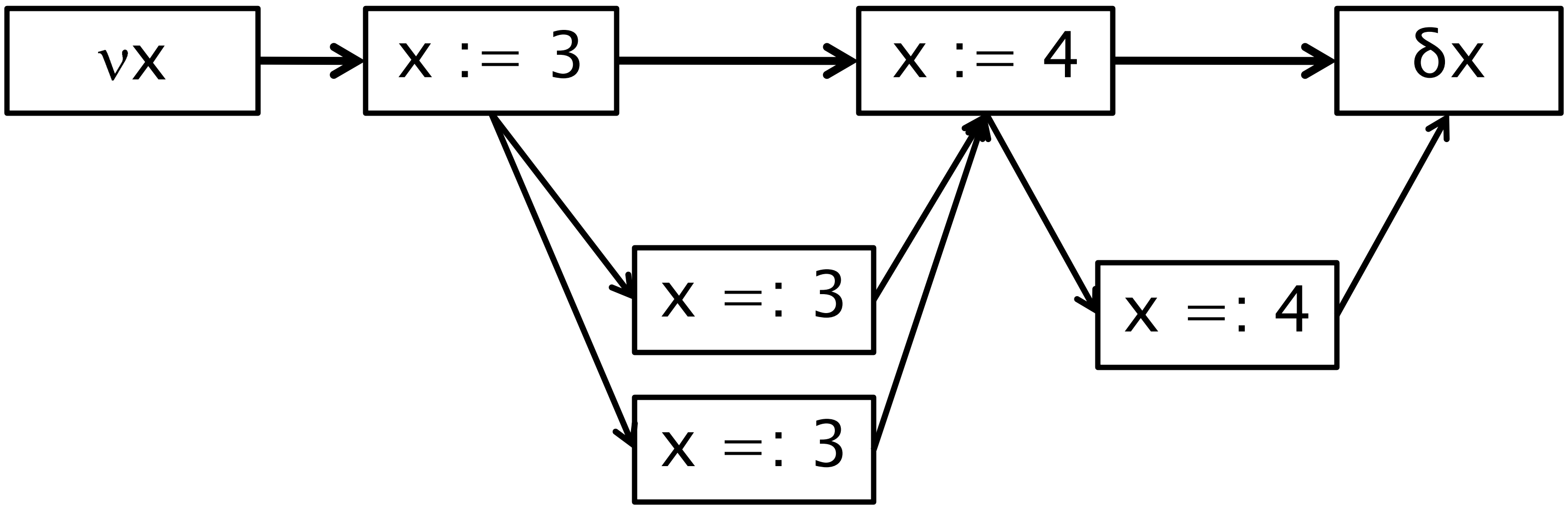
The token game (4)



The token game (5)

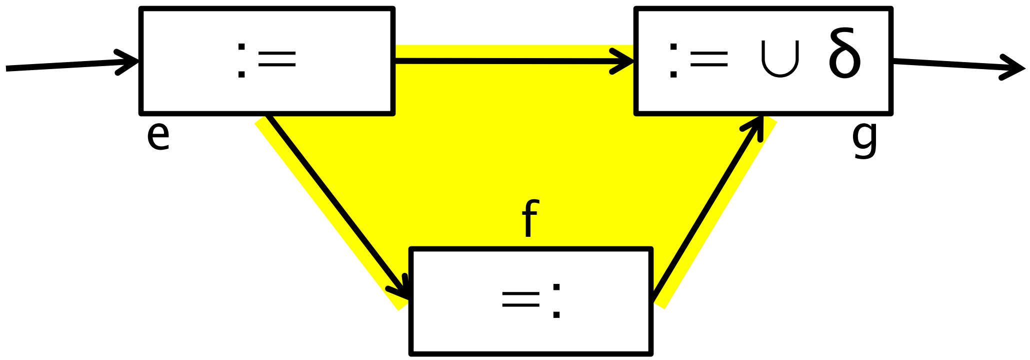


A variable



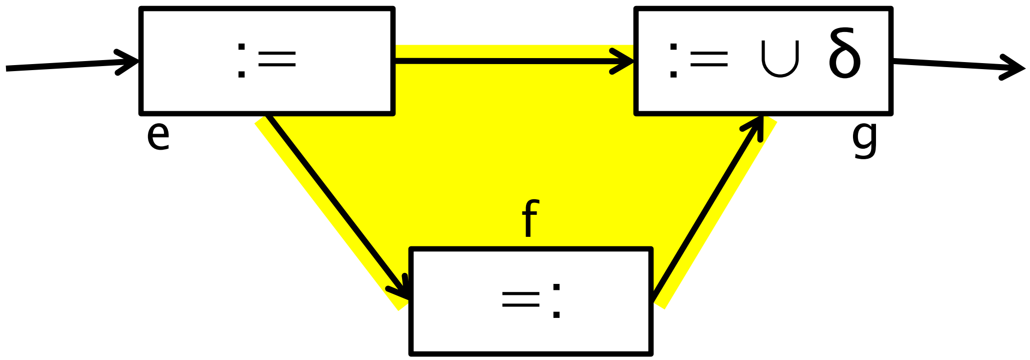
$$
\begin{aligned}
x \to x \subseteq\ & (\nu \to {:=}) \cup (\nu \to {=:}) \cup \\
& ({:=} \to {:=}) \cup ({:=} \to {=:}) \cup \\
& ({=:} \to {:=}) \cup ({=:} \to {:=}) \cup \\
& (\nu \to \delta) \cup ({:=} \to \delta) \cup ({=:} \to \delta)
\end{aligned}
$$

A closed triangle (1)



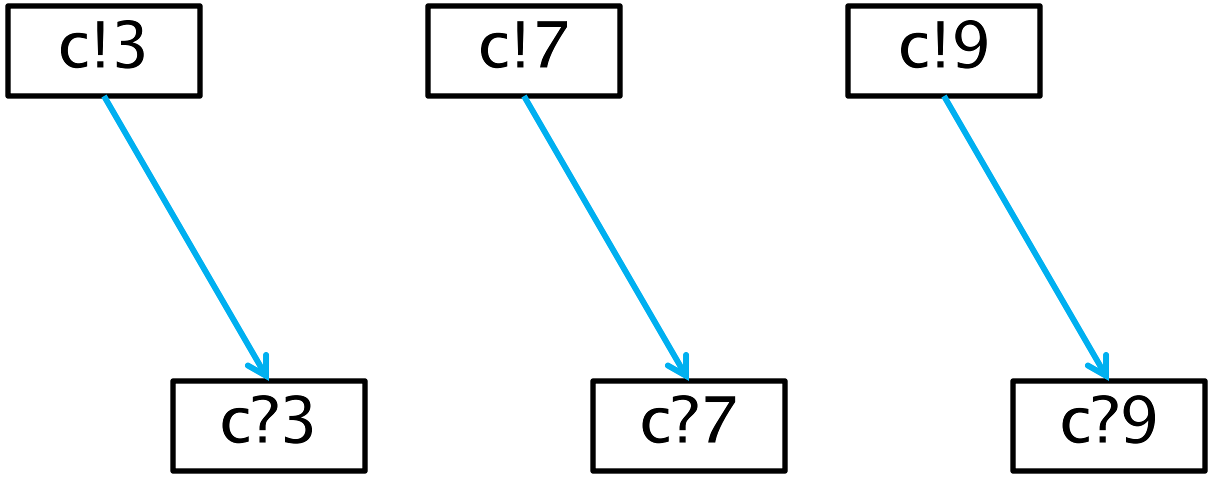
- $(:=)\rightarrow(=:)\rightarrow(:= \cup \delta)\leftarrow(:=) \quad \subseteq \quad (:=)$
- similar to: $(:=)\rightarrow(=:)\leftarrow(:=) \quad \subseteq \quad (:=)$

A closed triangle (2)

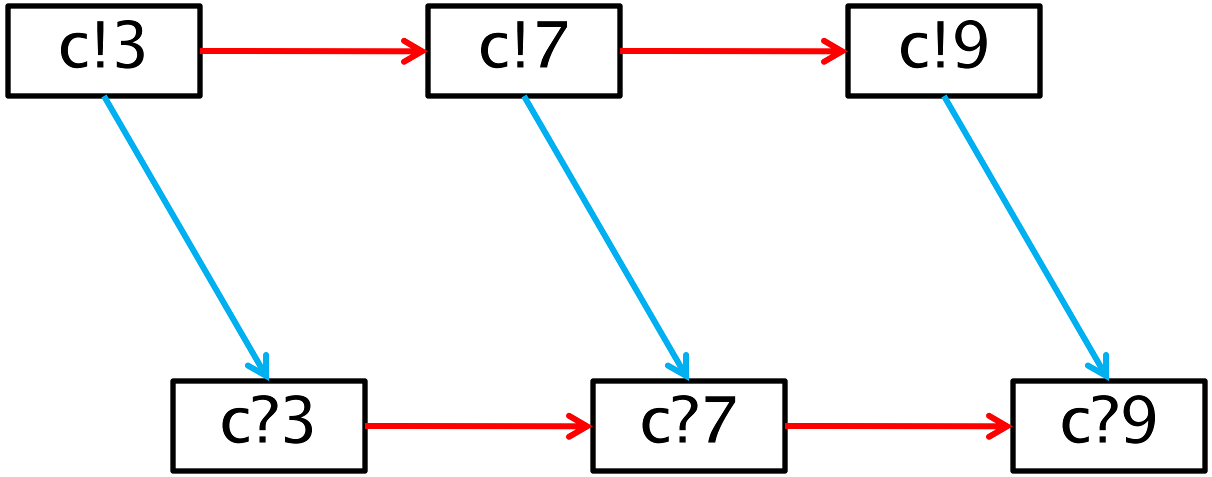


- $(:=) \rightarrow (:= \cup \ \delta) \leftarrow (=:) \leftarrow (:=) \quad \subseteq \quad (:=)$

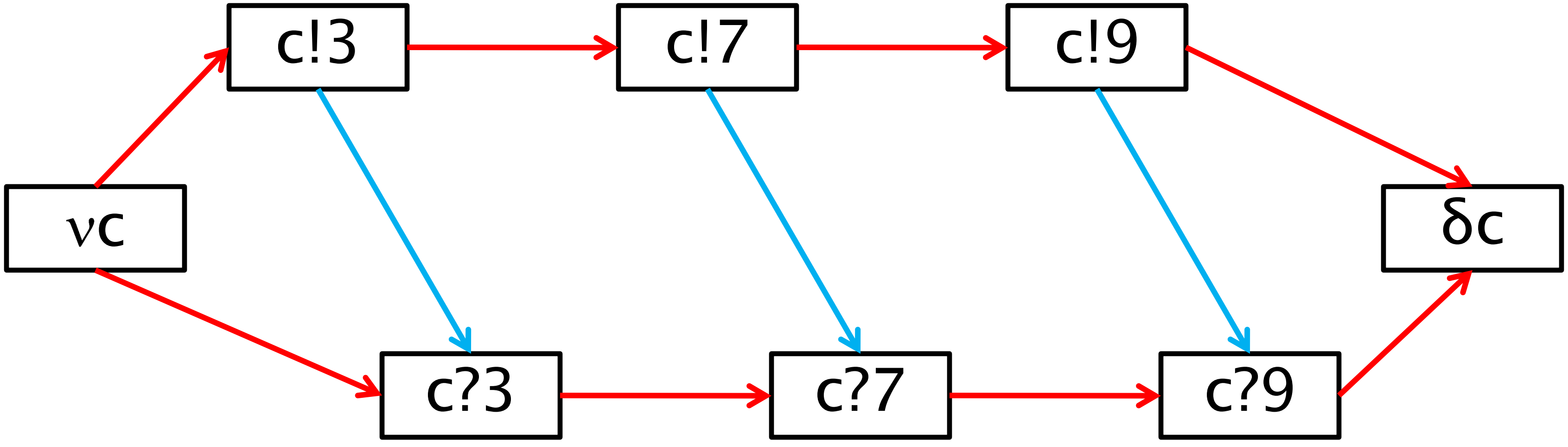
Communication



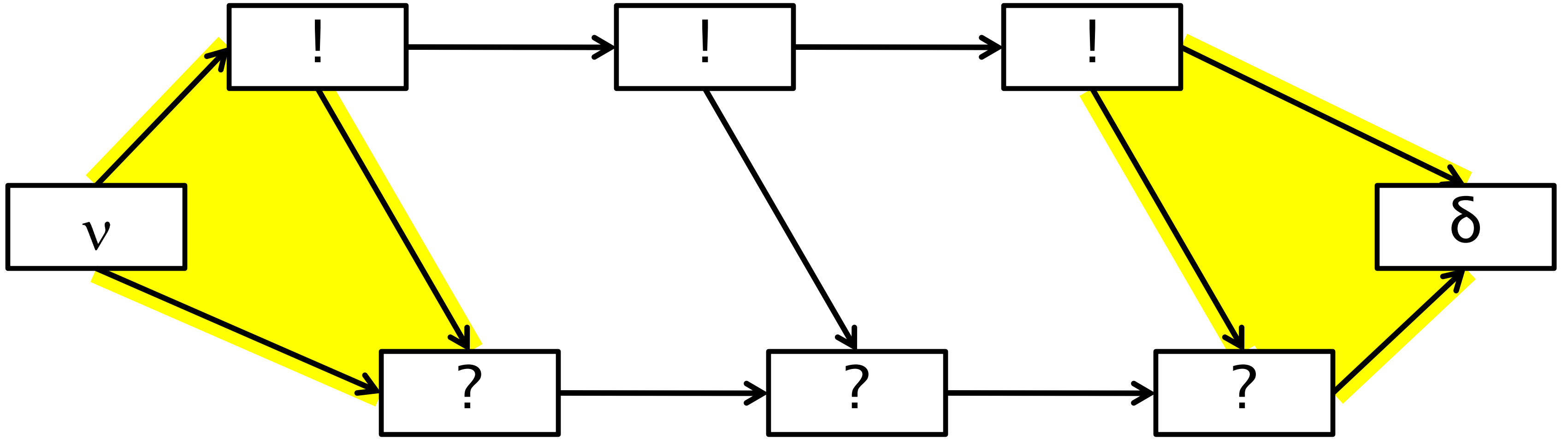
Sequential outputs/inputs



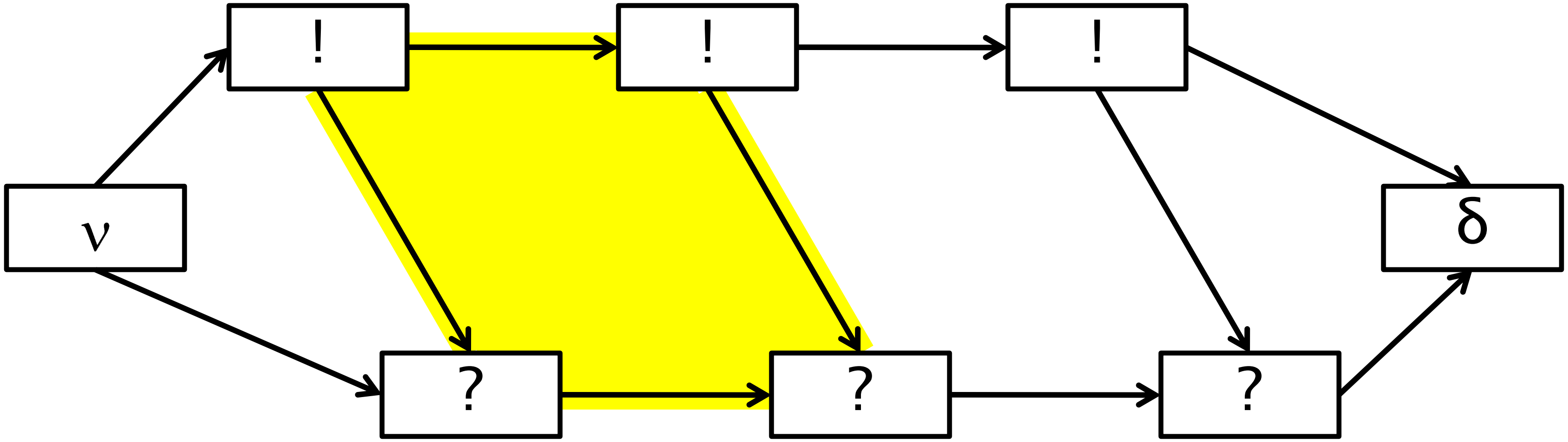
Channel



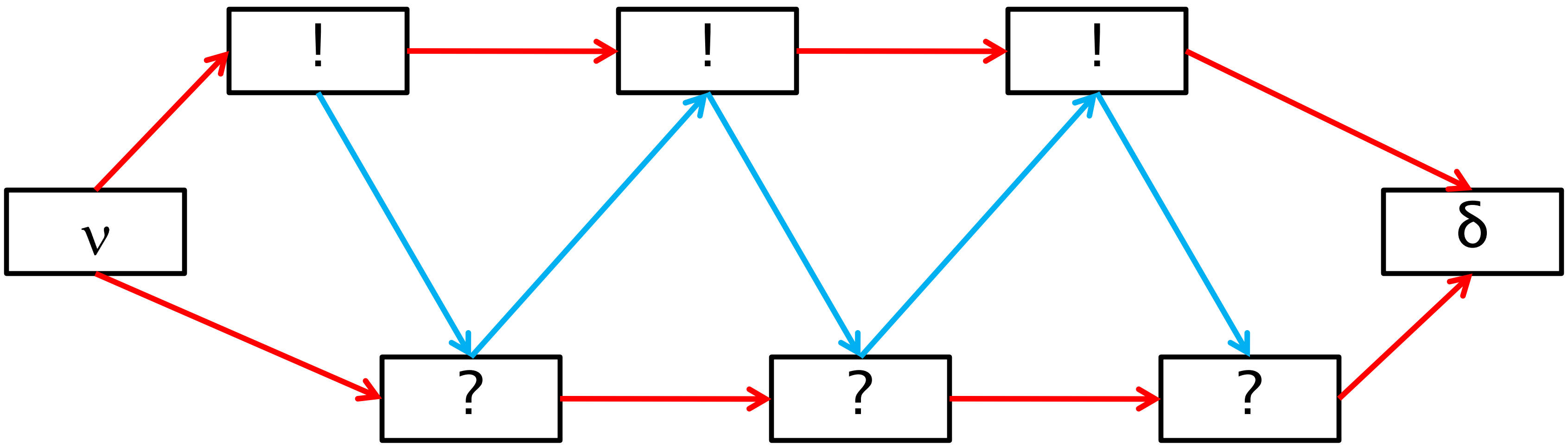
Closed triangles



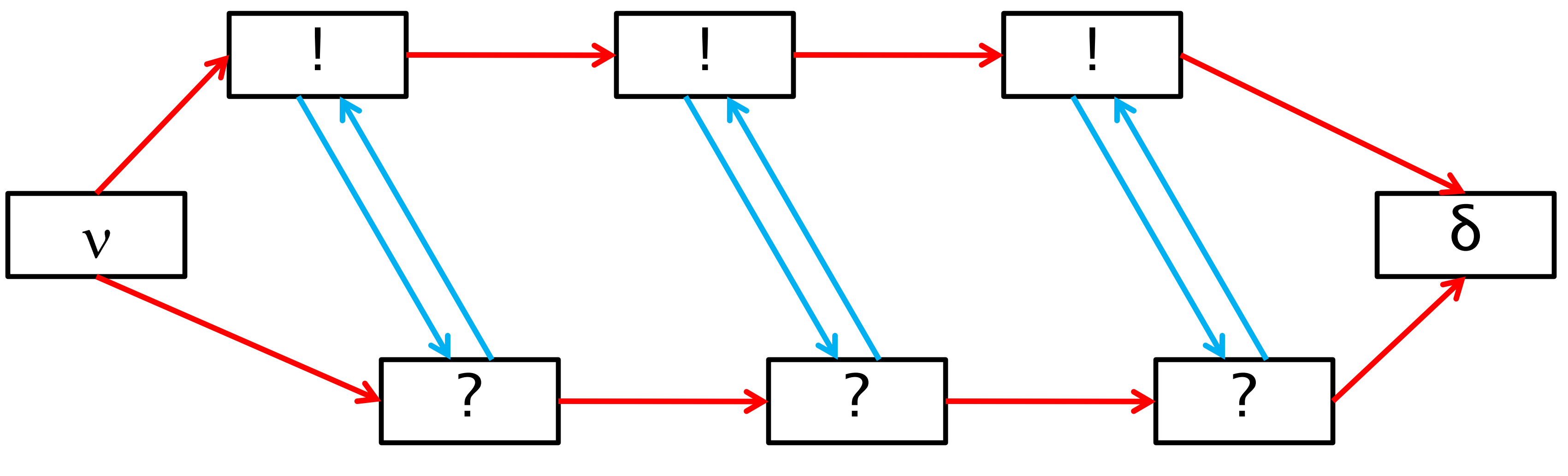
Closed rectangles



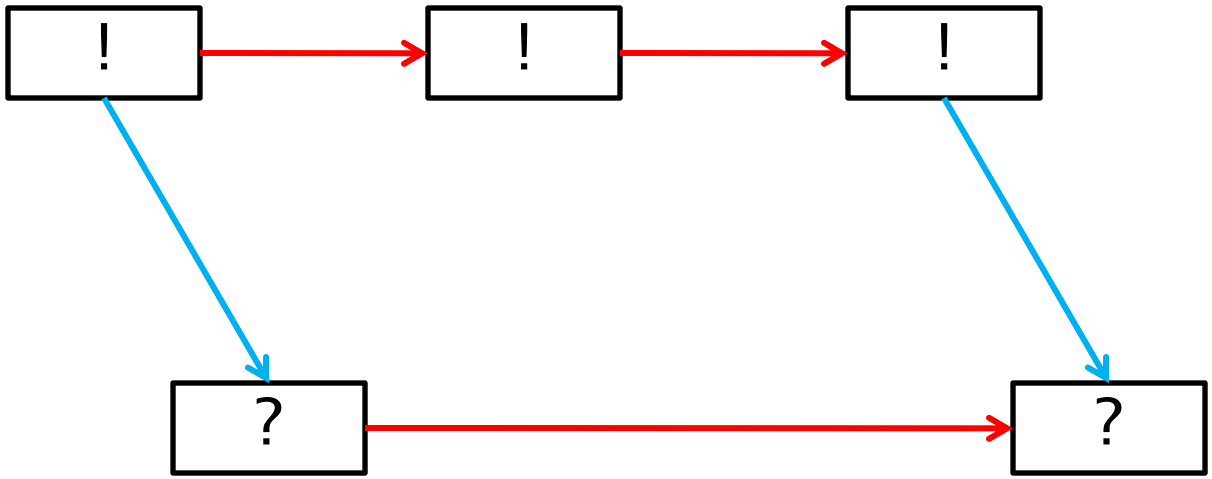
Singly-buffered channel



Zero-buffered channel



A lossy channel



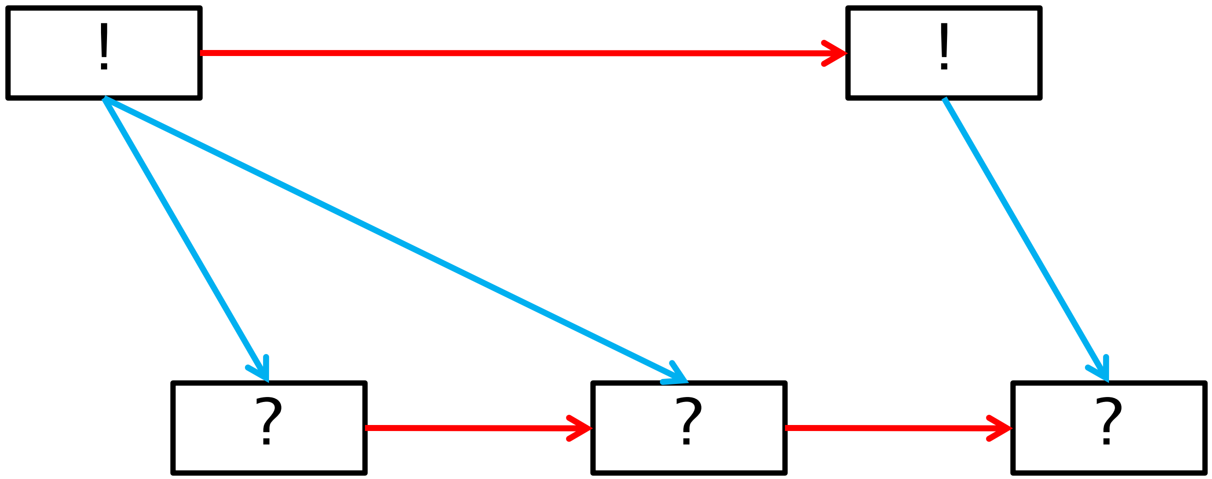
For a <u>lossless</u> channel,

$! \to ?$ is a total relation on outputs

$! \subseteq ! \to ? \leftarrow !$

$\forall e \in ! \,.\, \exists f \in ? \,.\, e \to f$

A stuttering channel

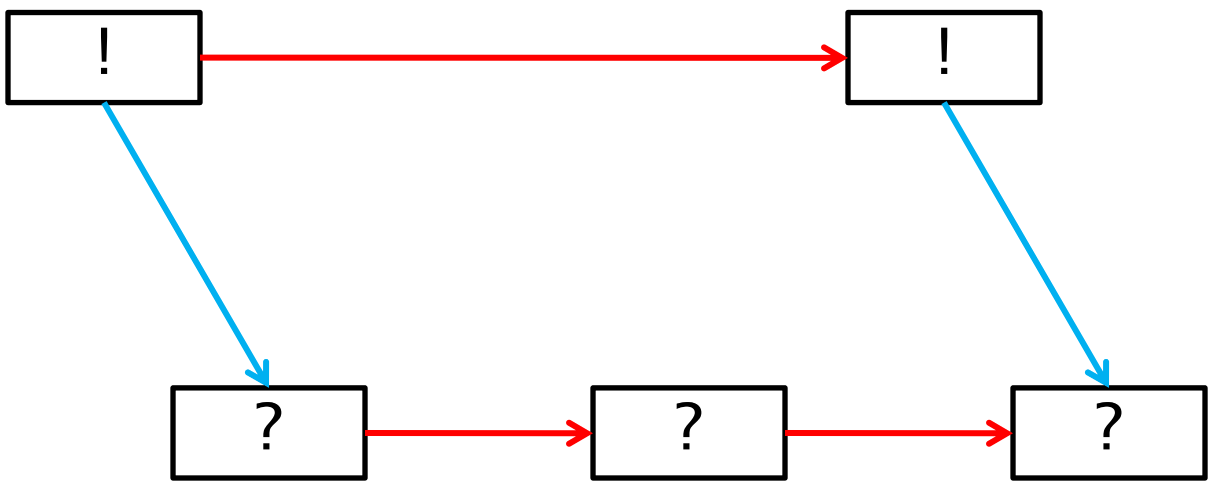


For a <u>non-stuttering</u> channel,

! → ? is a (partial) function

? ← ! → ? ⊆ ?

A fraudulent channel



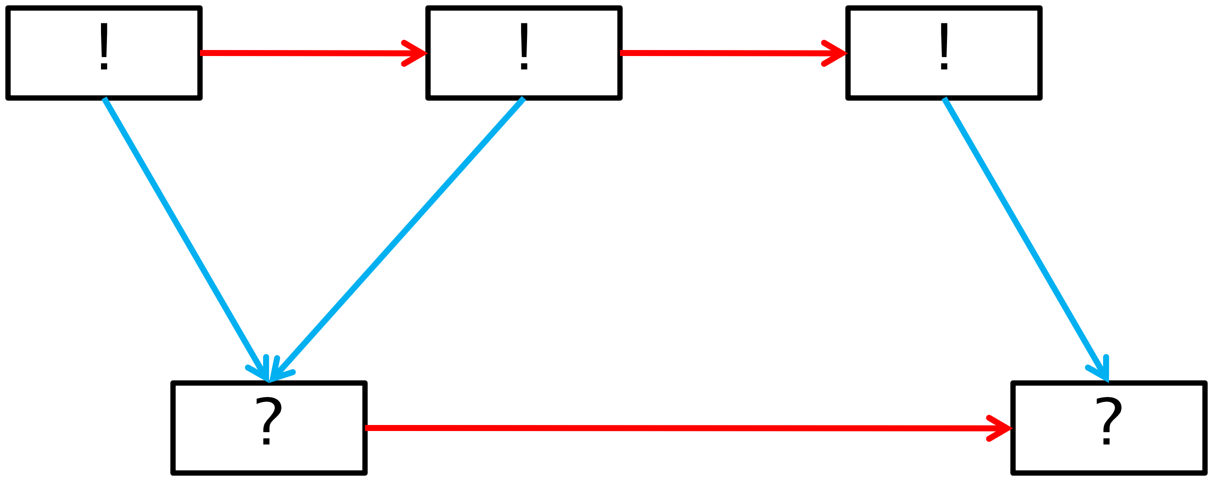
For a <u>non-fraudulent</u> channel,

$! \rightarrow ?$ is surjective

$? \subseteq ? \rightarrow ! \leftarrow ?$

$\forall e \in ? \,.\, \exists f \in ! \,.\, f \rightarrow e$

A confusing channel



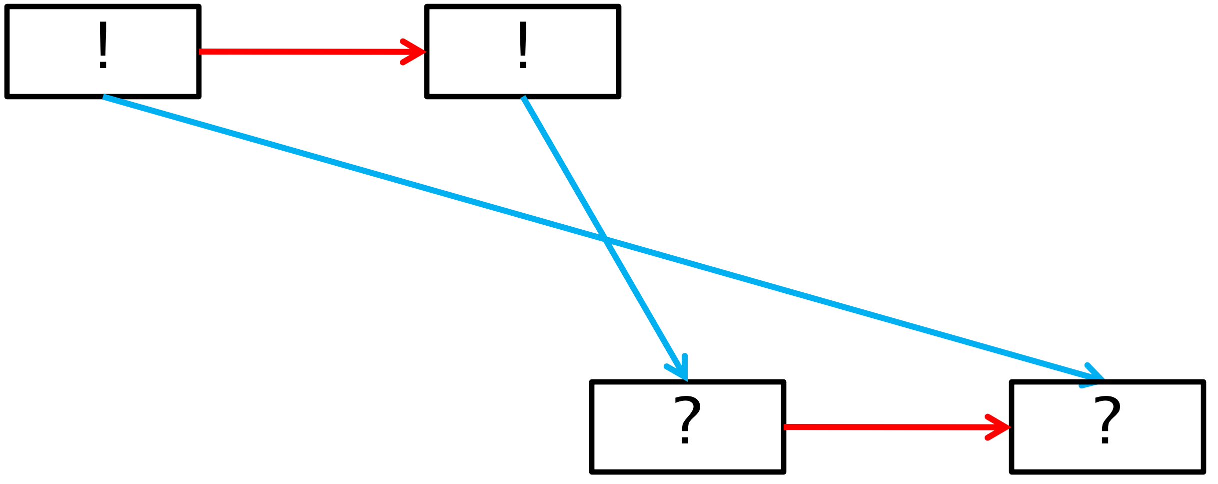
For a non-confusing channel,

$! \rightarrow ?$ is injective

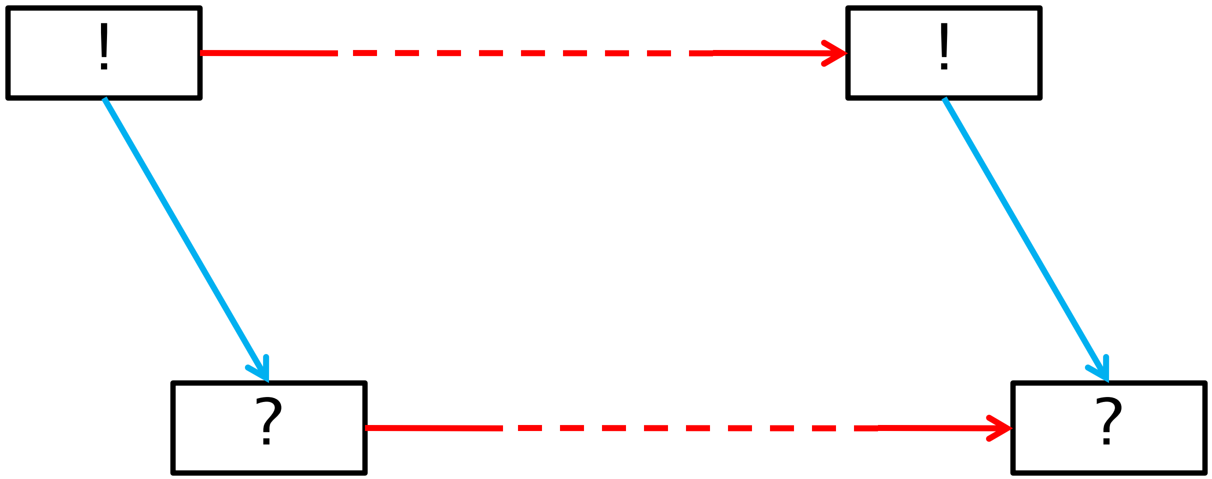
$! \rightarrow ? \leftarrow ! \subseteq \mathrm{Id}$

$\forall e, e', f \,.\; e \rightarrow f \;\&\; e' \rightarrow f \;\Rightarrow\; e = e'$

An overtaking channel

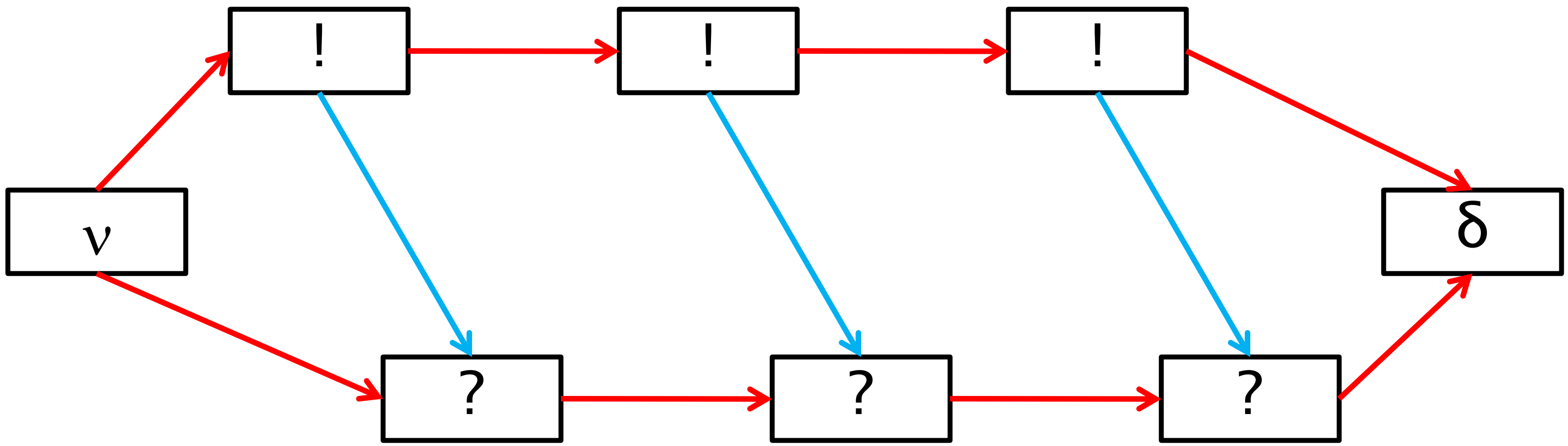


An order-preserving channel

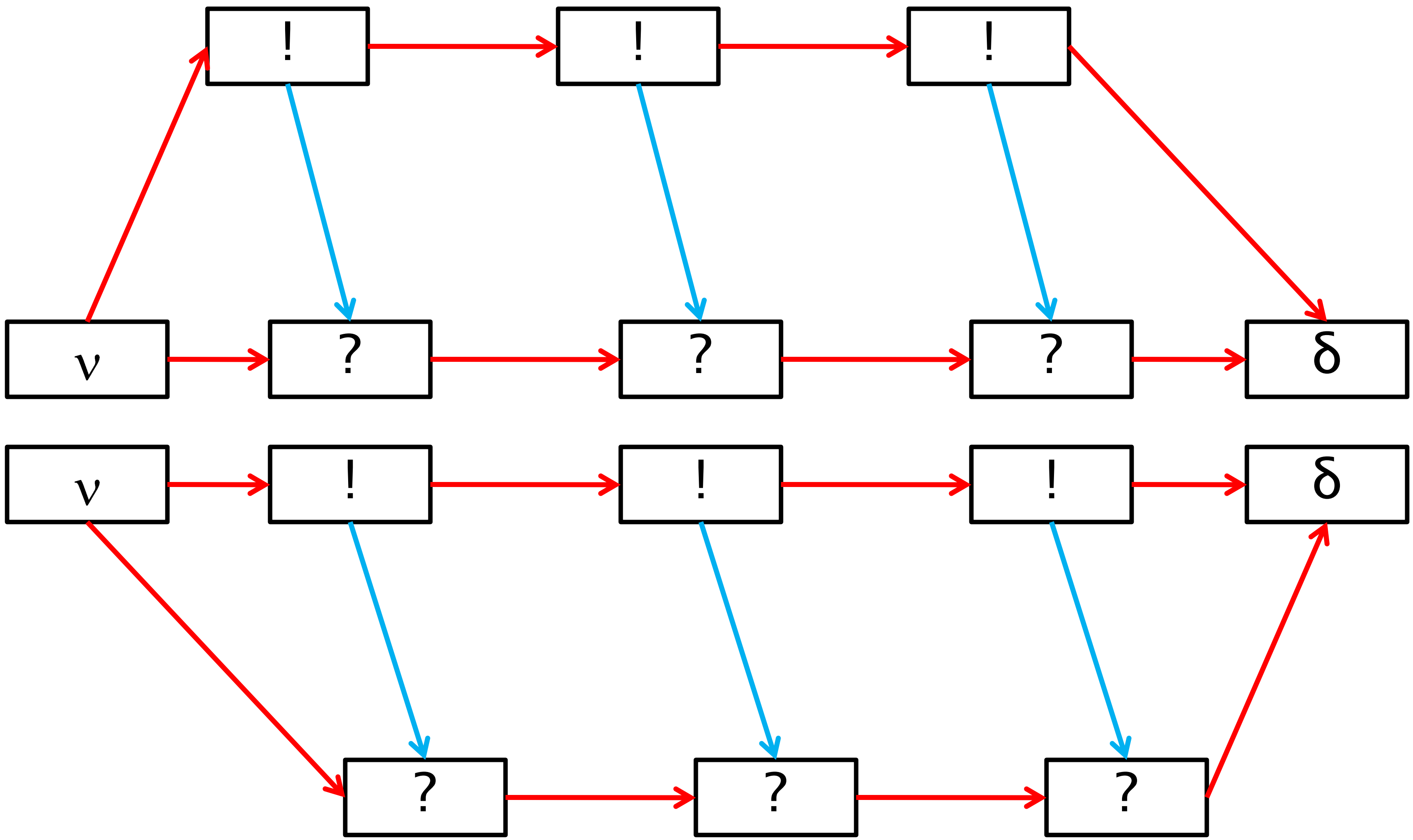


$$(! \color{red}{\rightarrow} \color{black}{!})^+ \color{cyan}{\rightarrow} \color{black}{?} = ! \color{cyan}{\rightarrow} \color{black}{(? \color{red}{\rightarrow} \color{black}{?})^+}$$

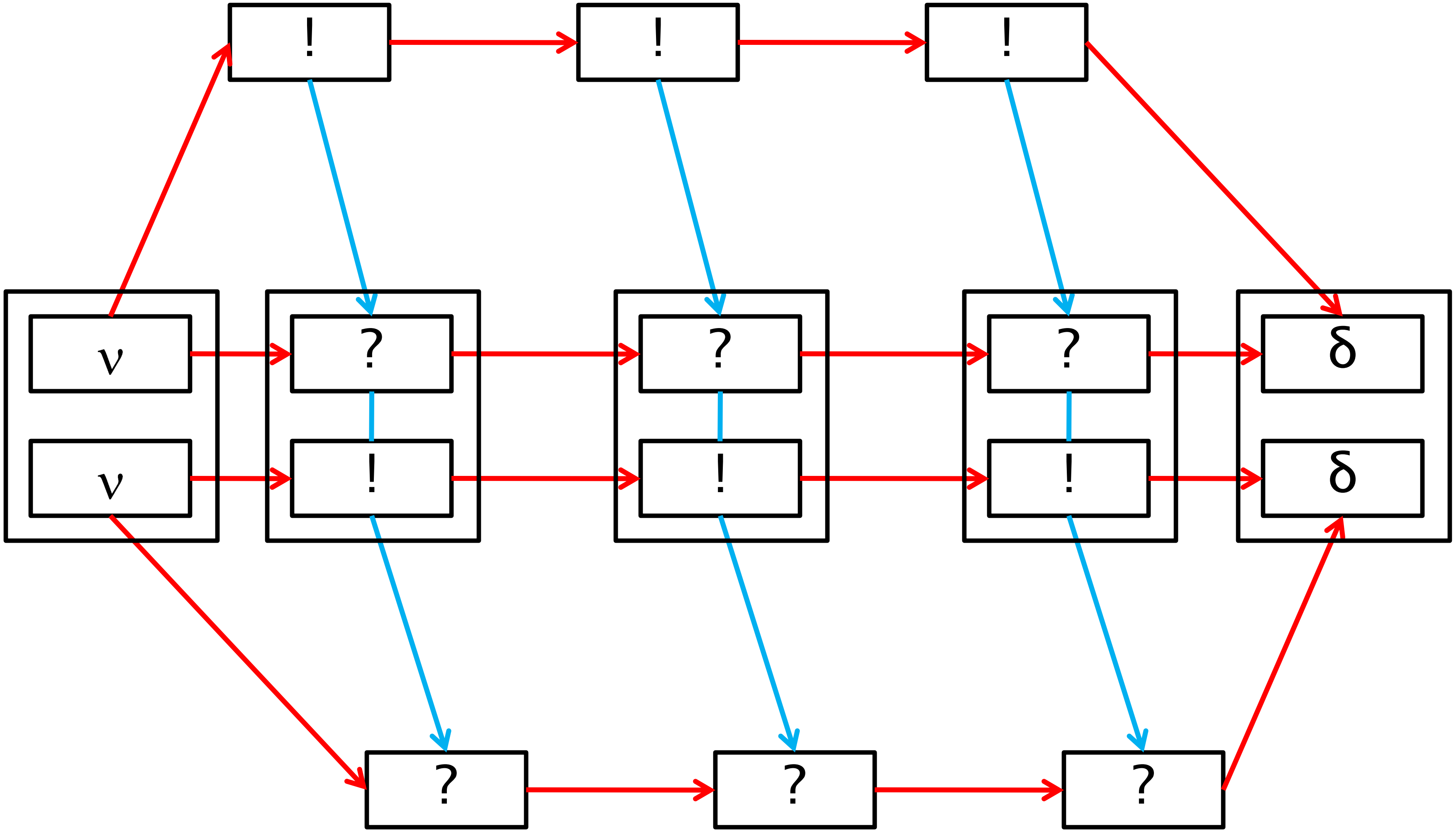
New channels from old



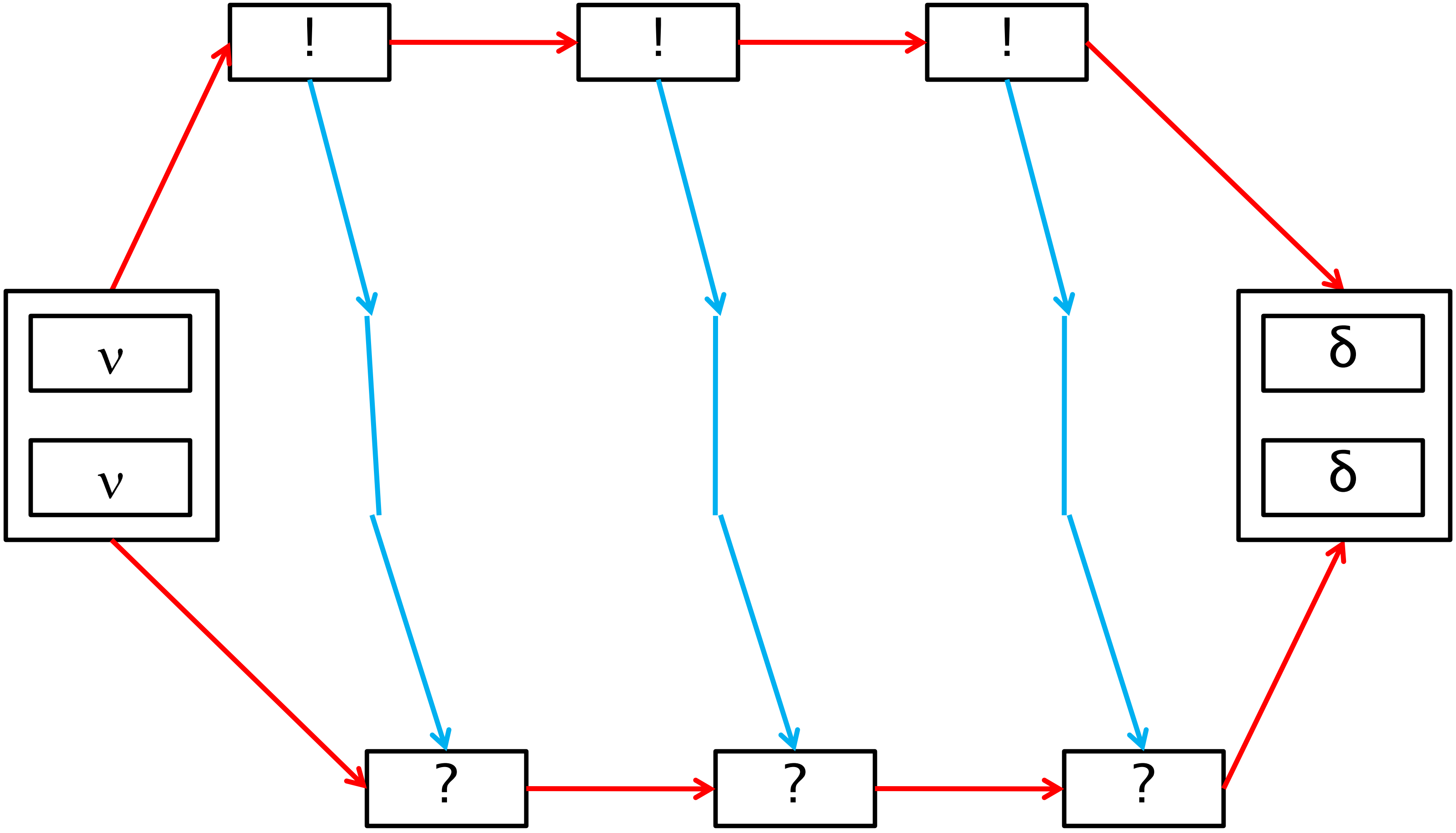
Take two channels...



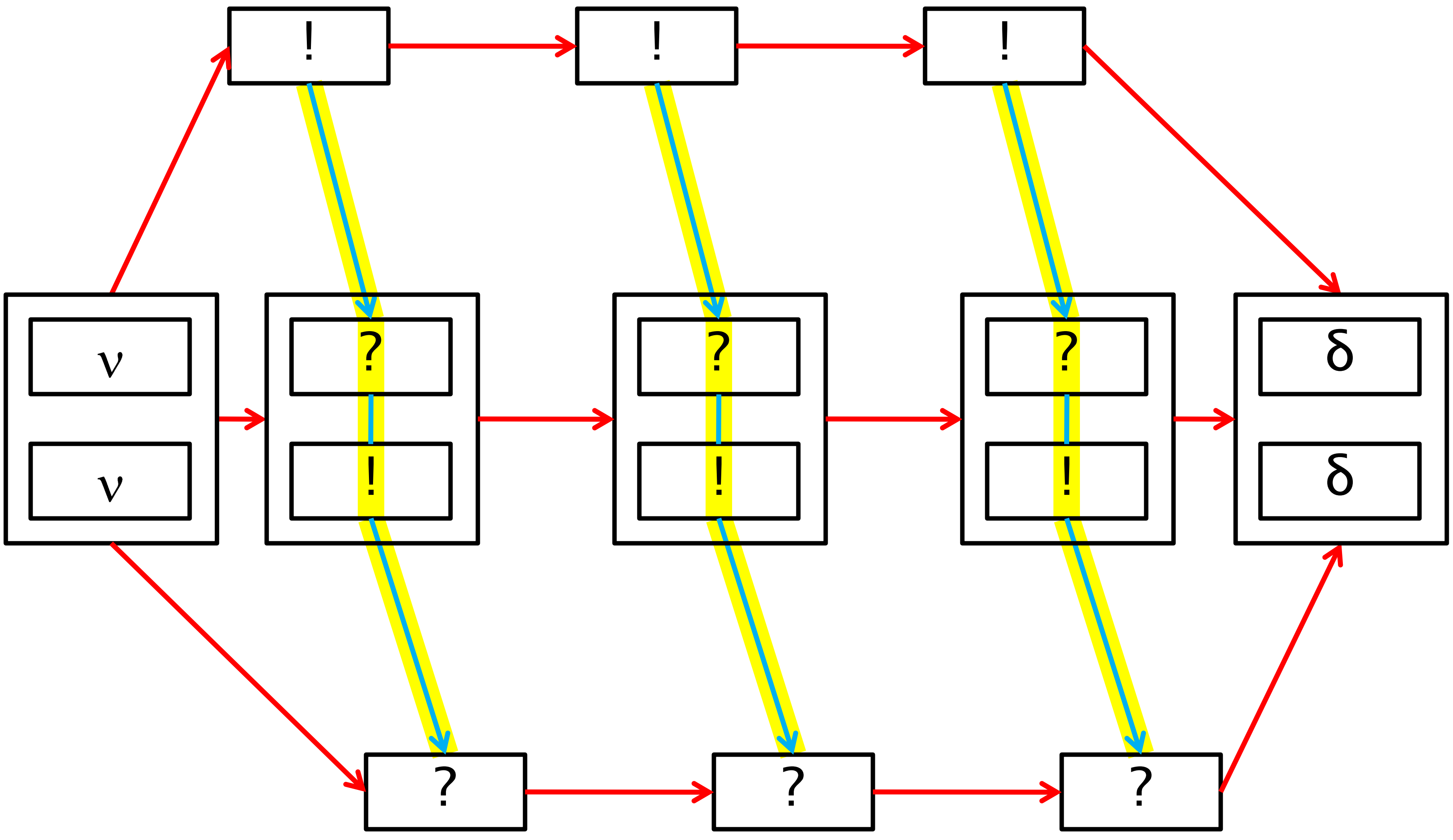
...connect them...



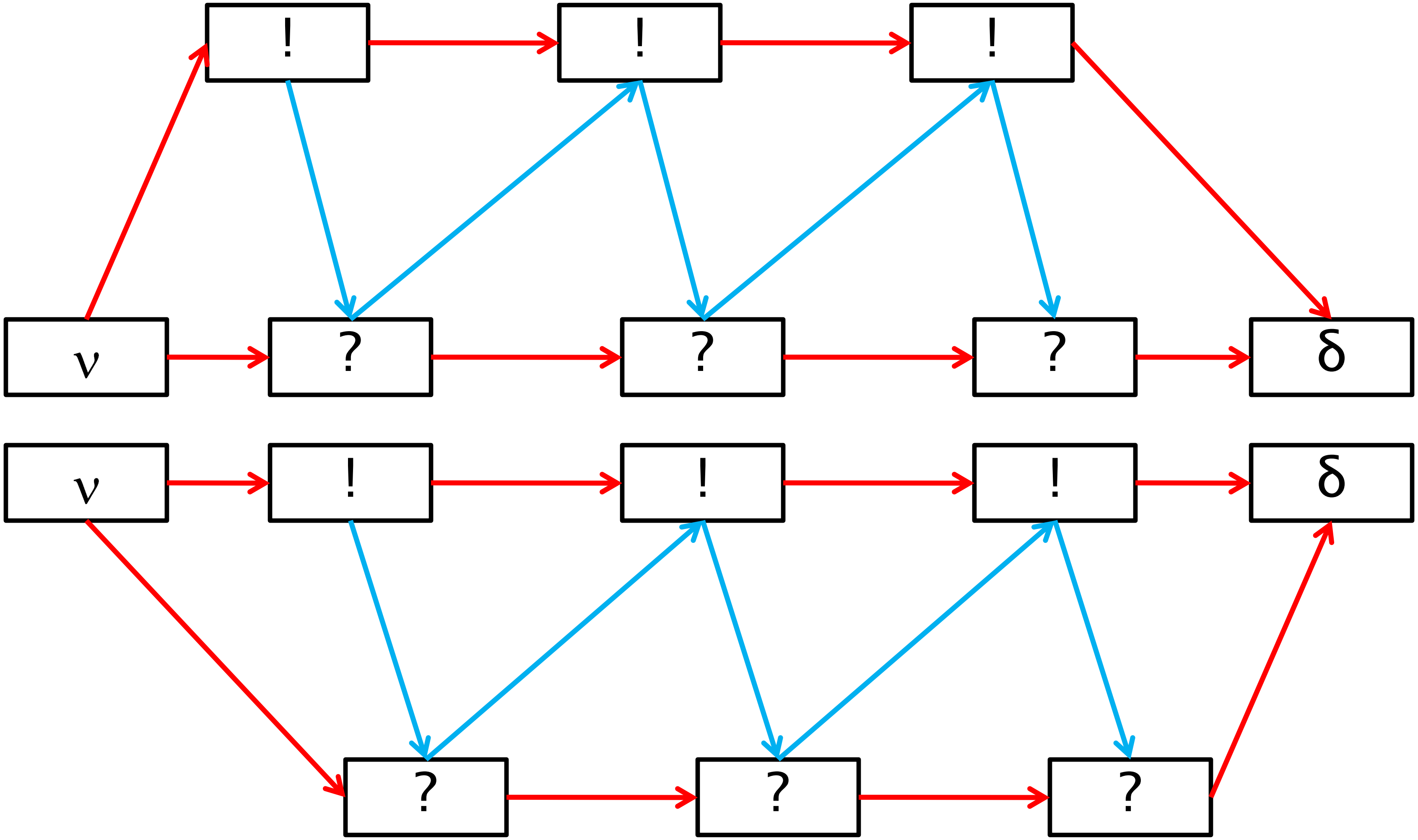
... abstract internal events...



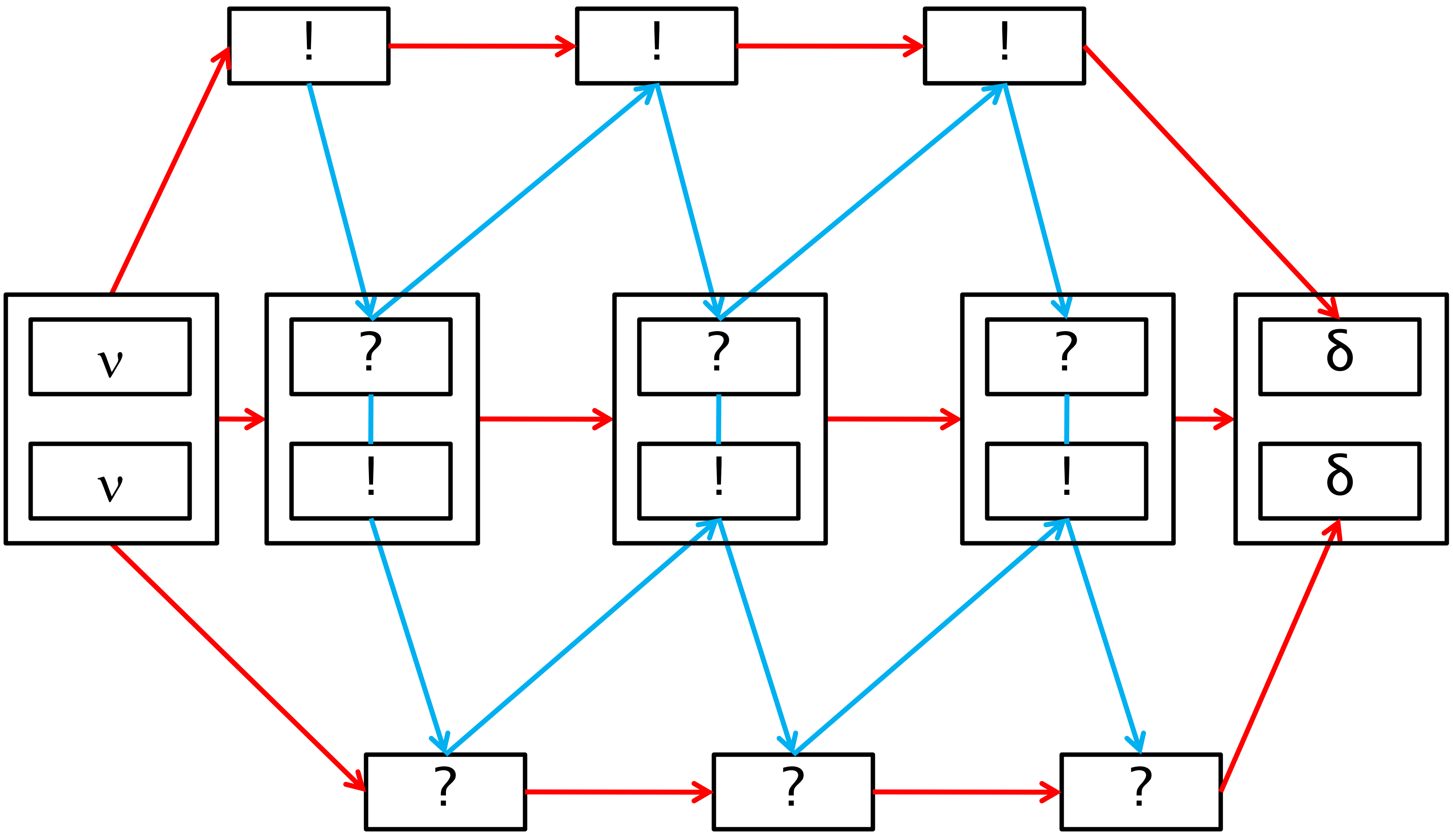
... result: a single channel



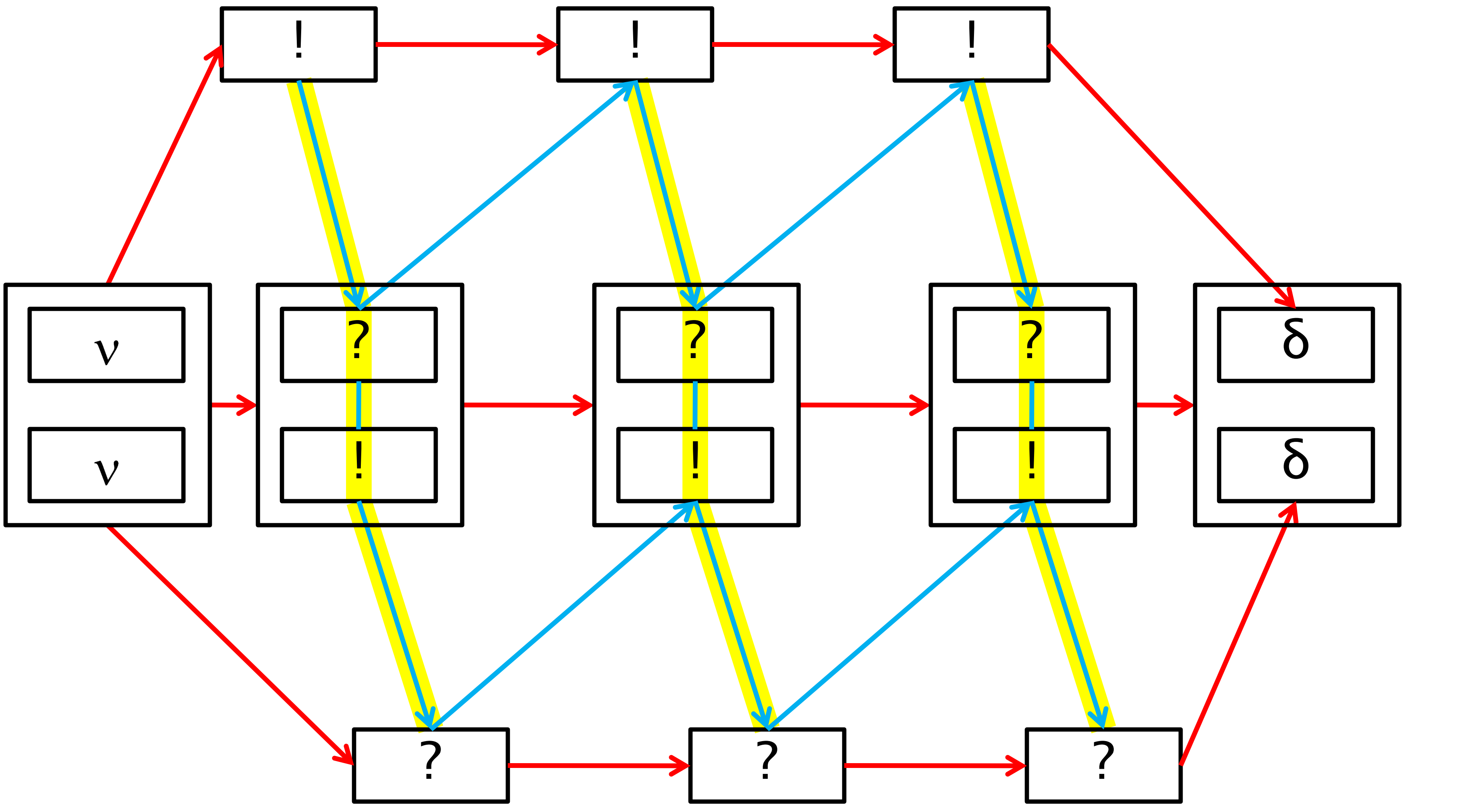
Two singly-buffered channels…



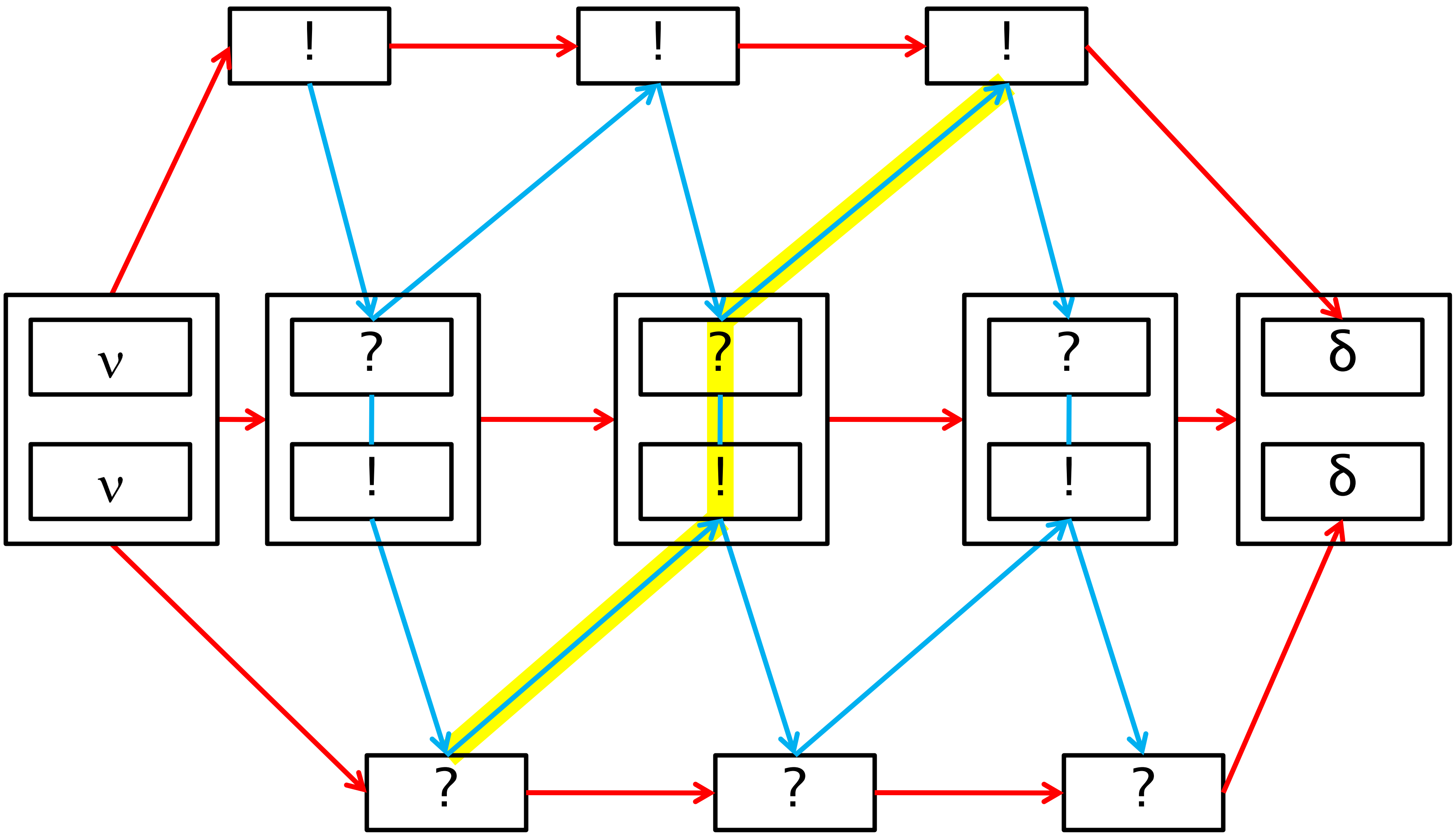
...connected...



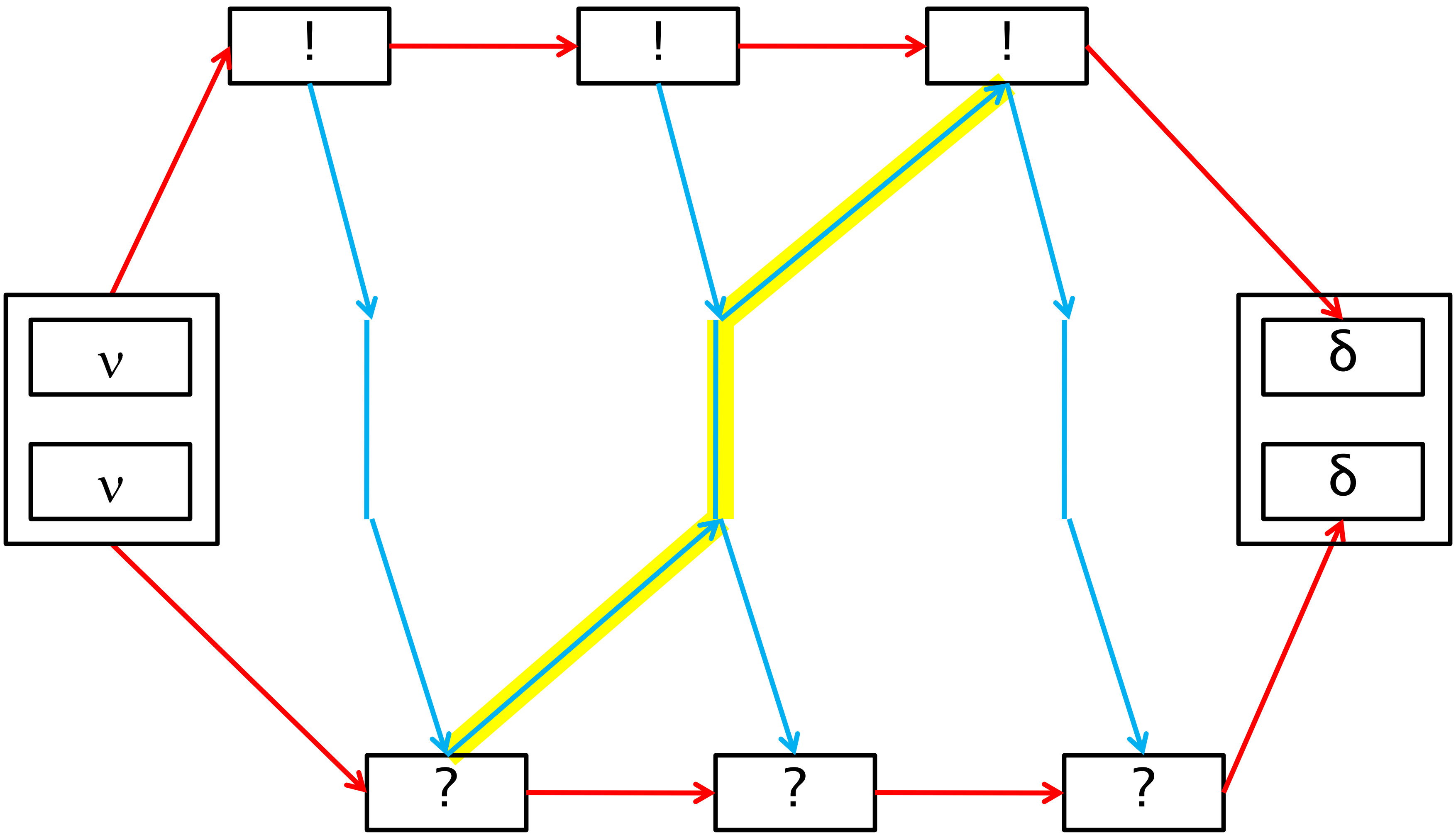
… doubly-buffered channel



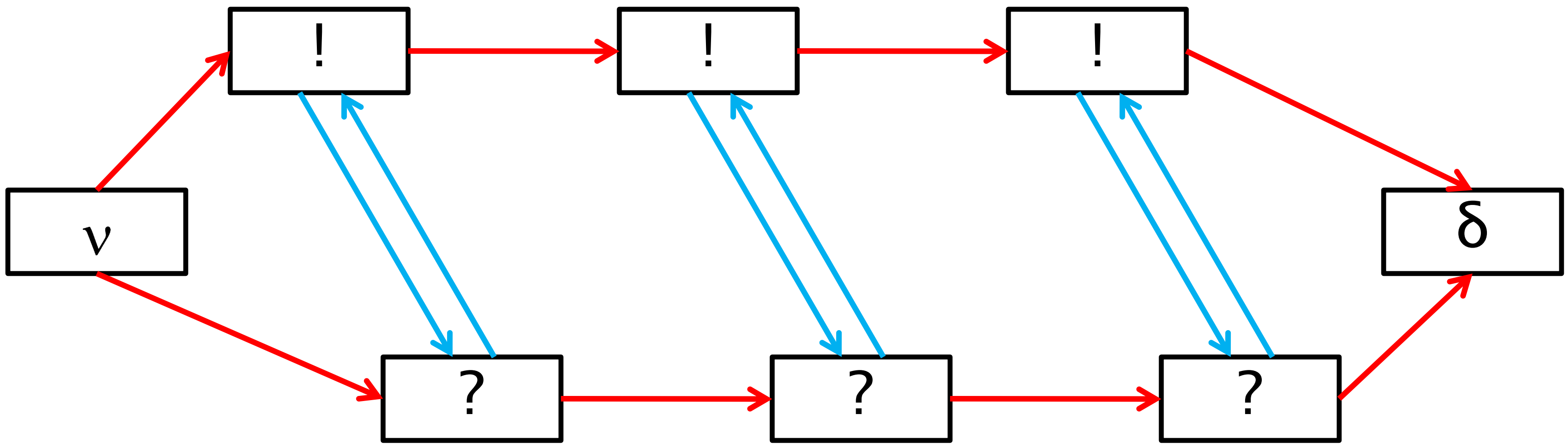
... doubly-buffered channel



... doubly-buffered channel



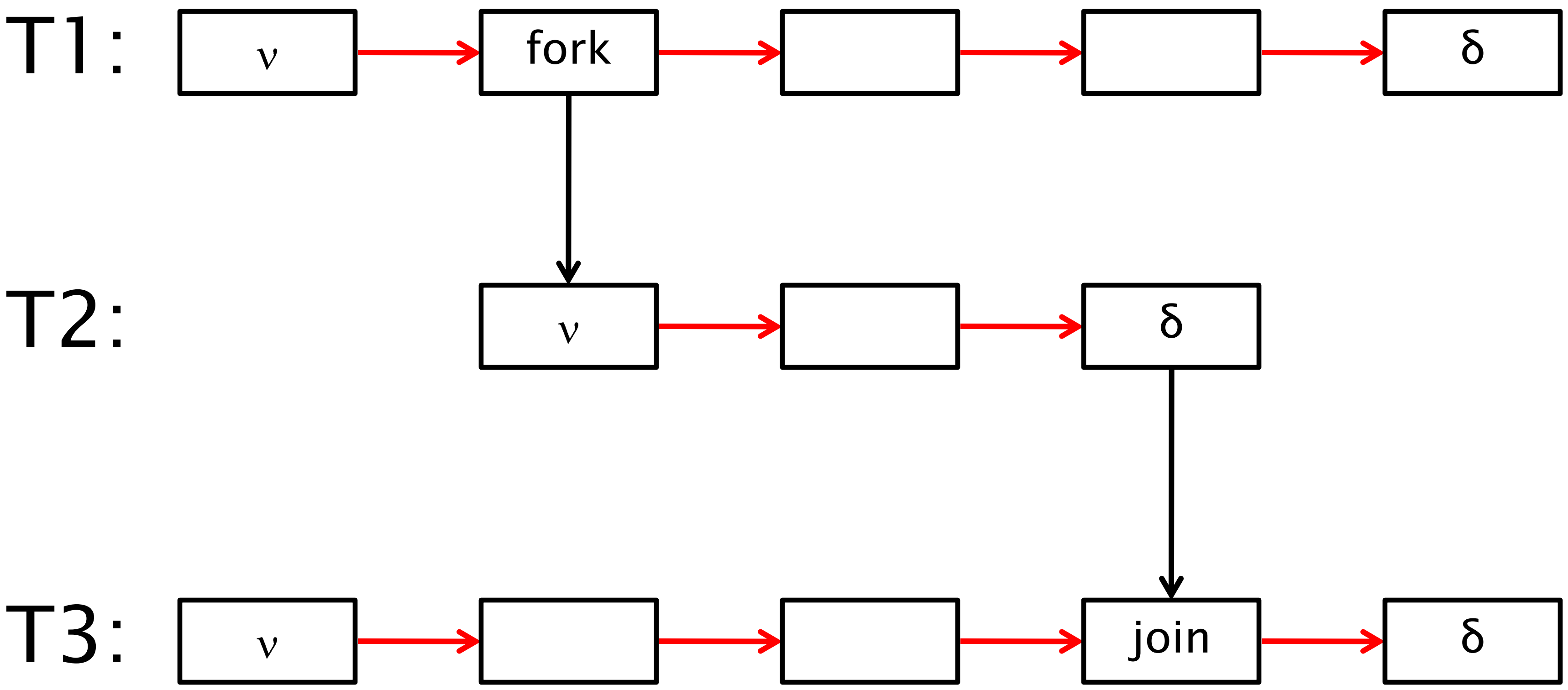
Zero-buffered channel



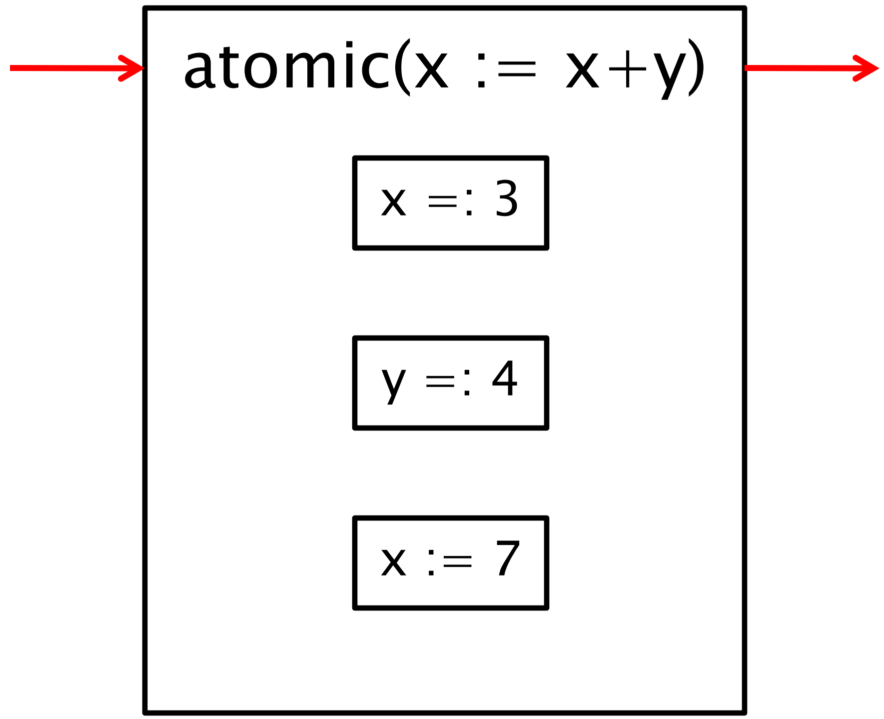
Exercises

- What is the effect of linking a zero-buffered channel to another channel?
- Implement a singly-buffered channel by means of two semaphores to synchronise input with output, and a variable to hold the content of the buffer.

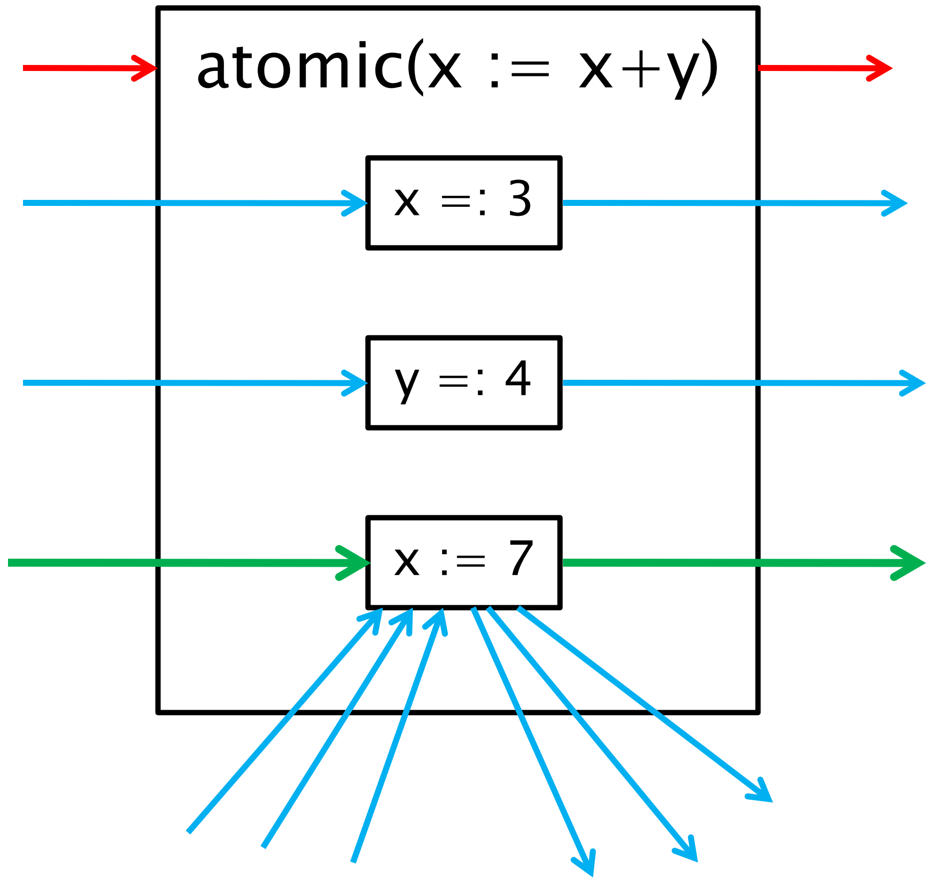
Threads



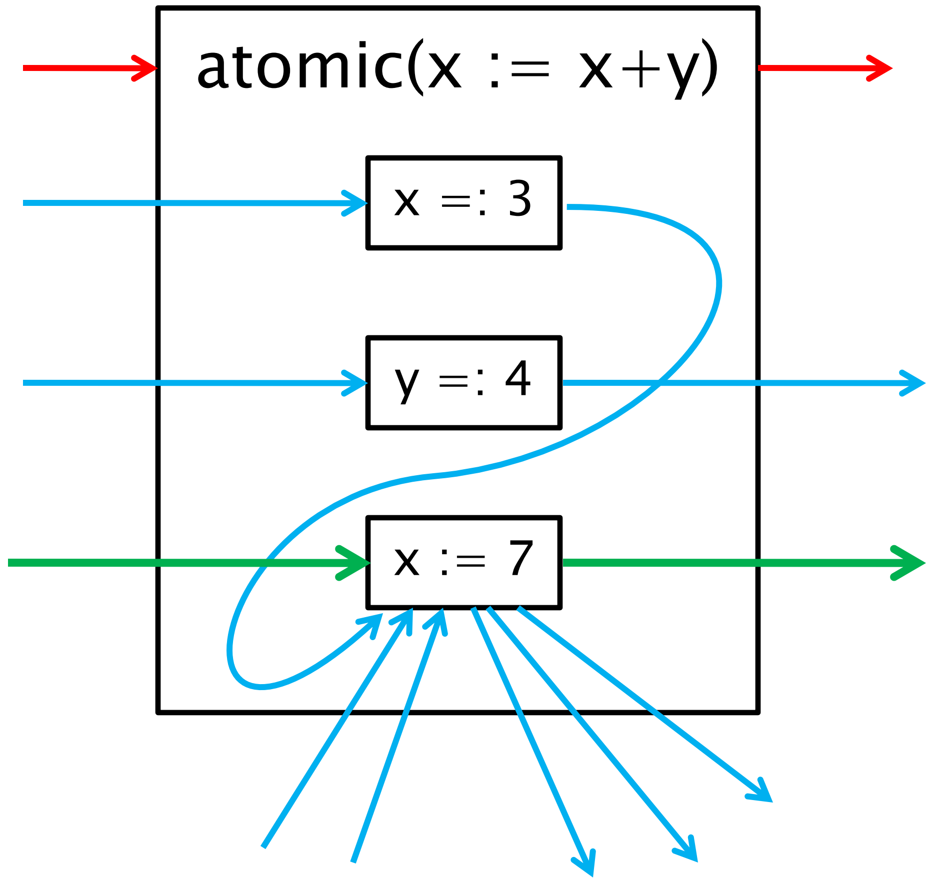
An atomic assignment (1)



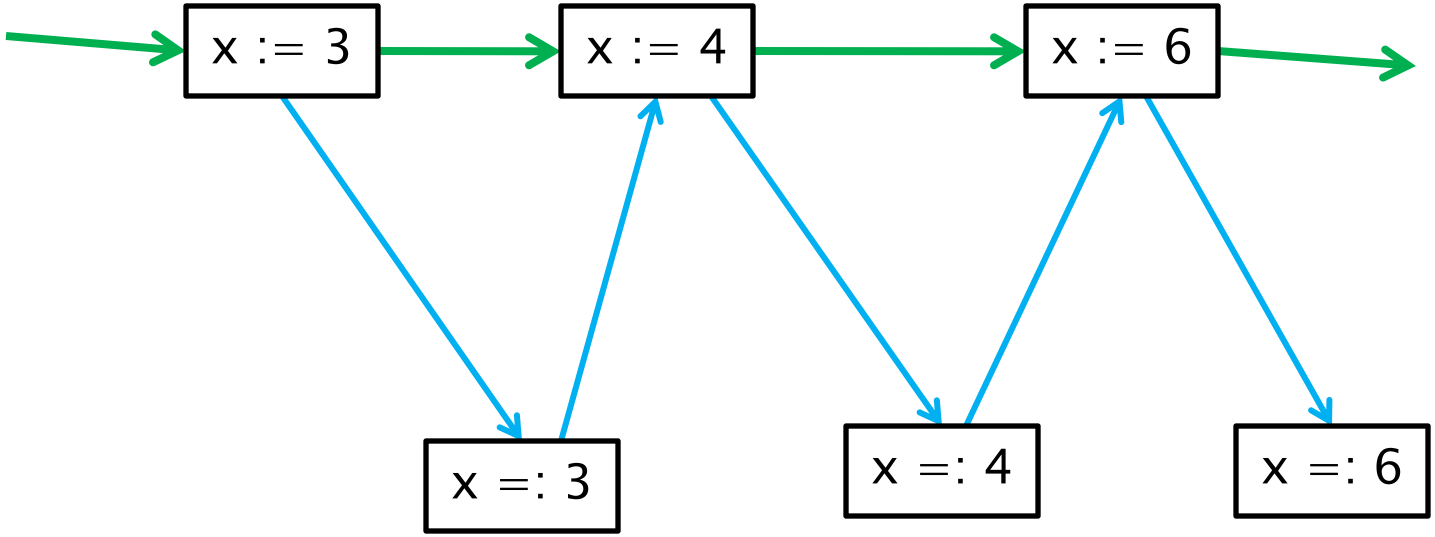
An atomic assignment (2)



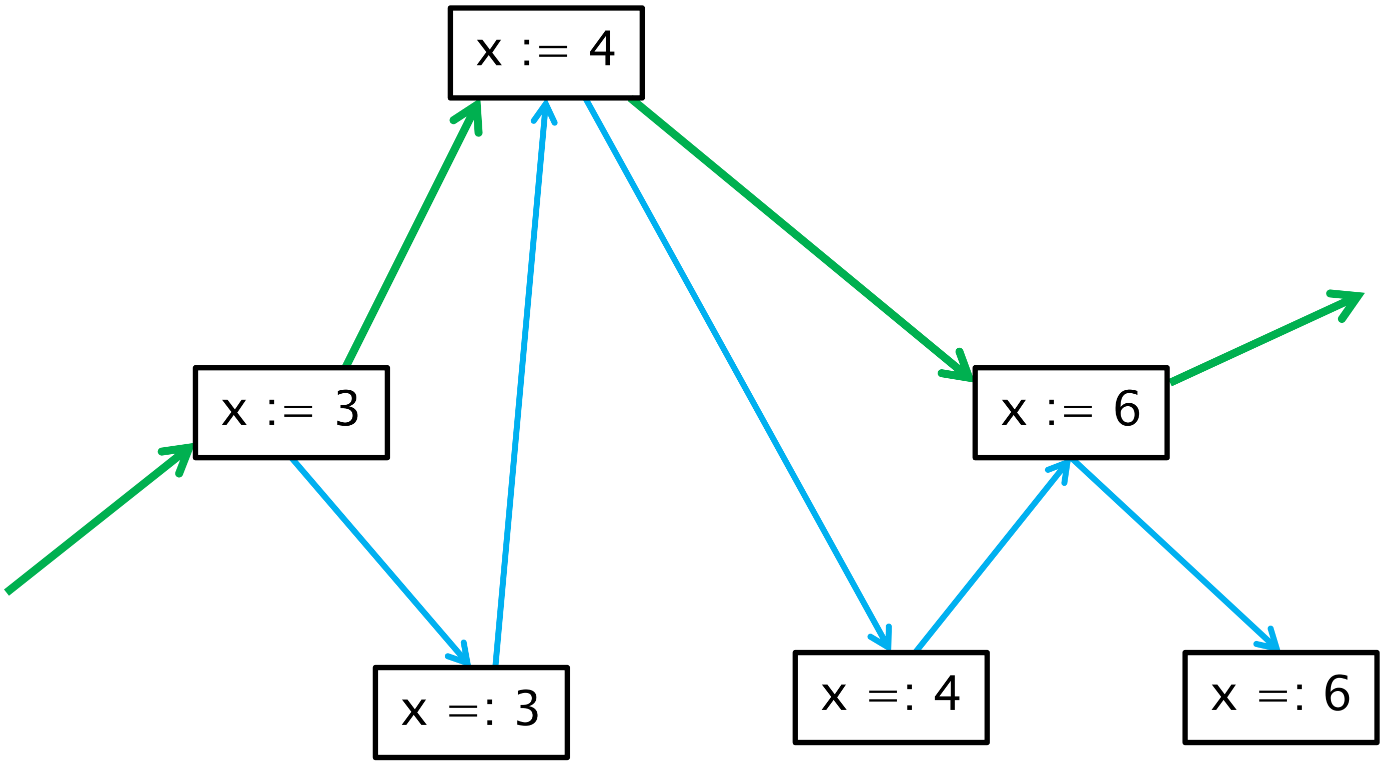
An atomic assignment (3)



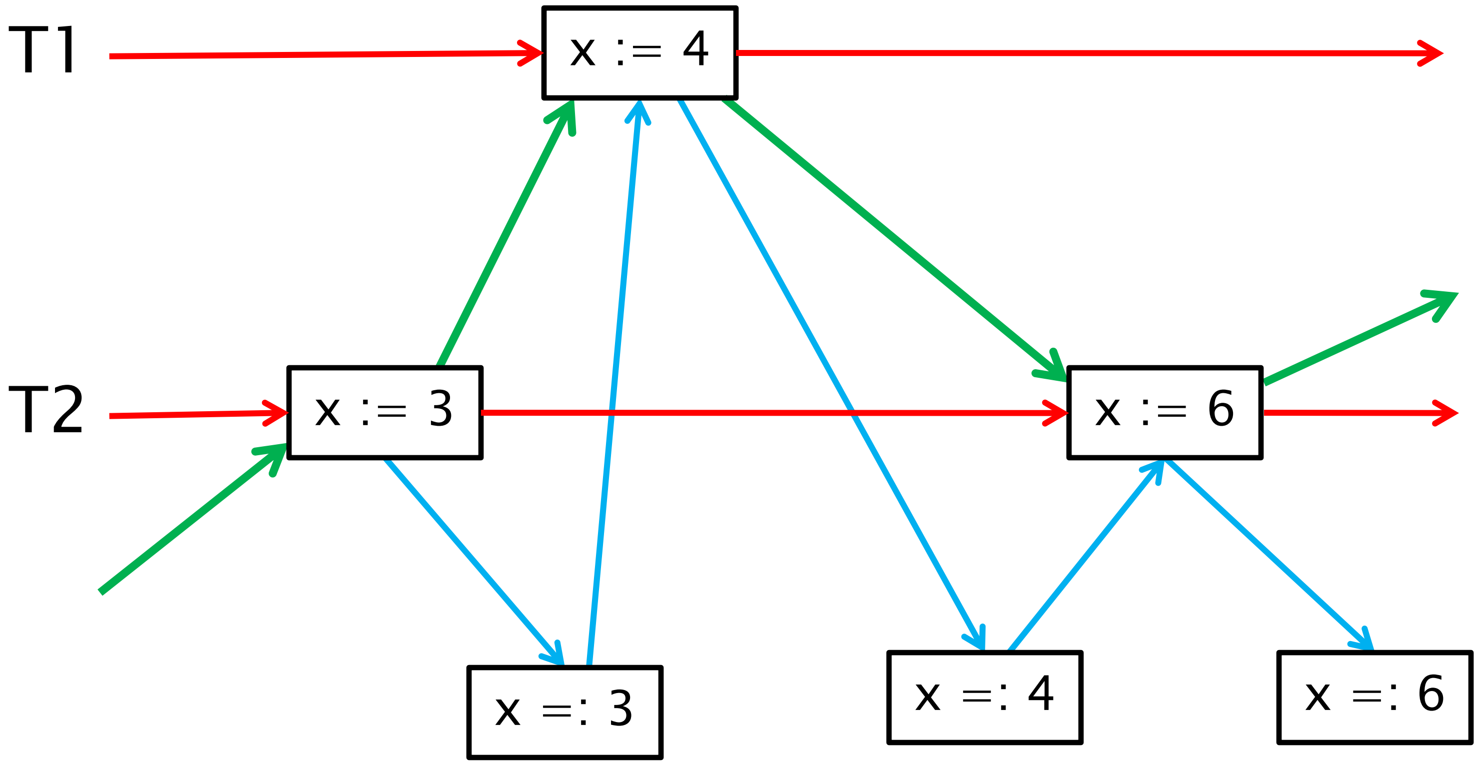
A shared variable (1)



A shared variable (2)



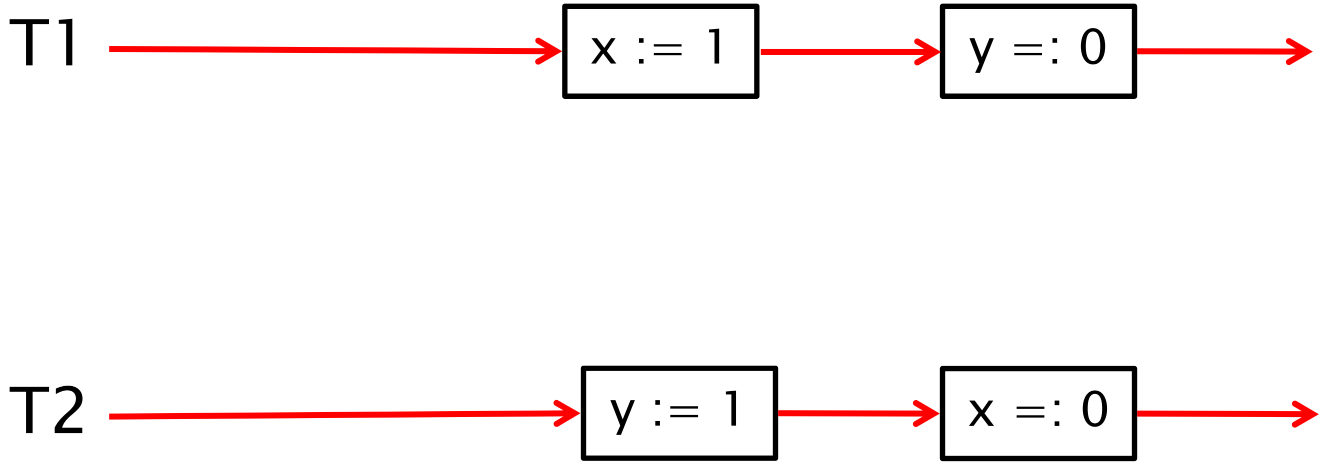
A shared variable(3)



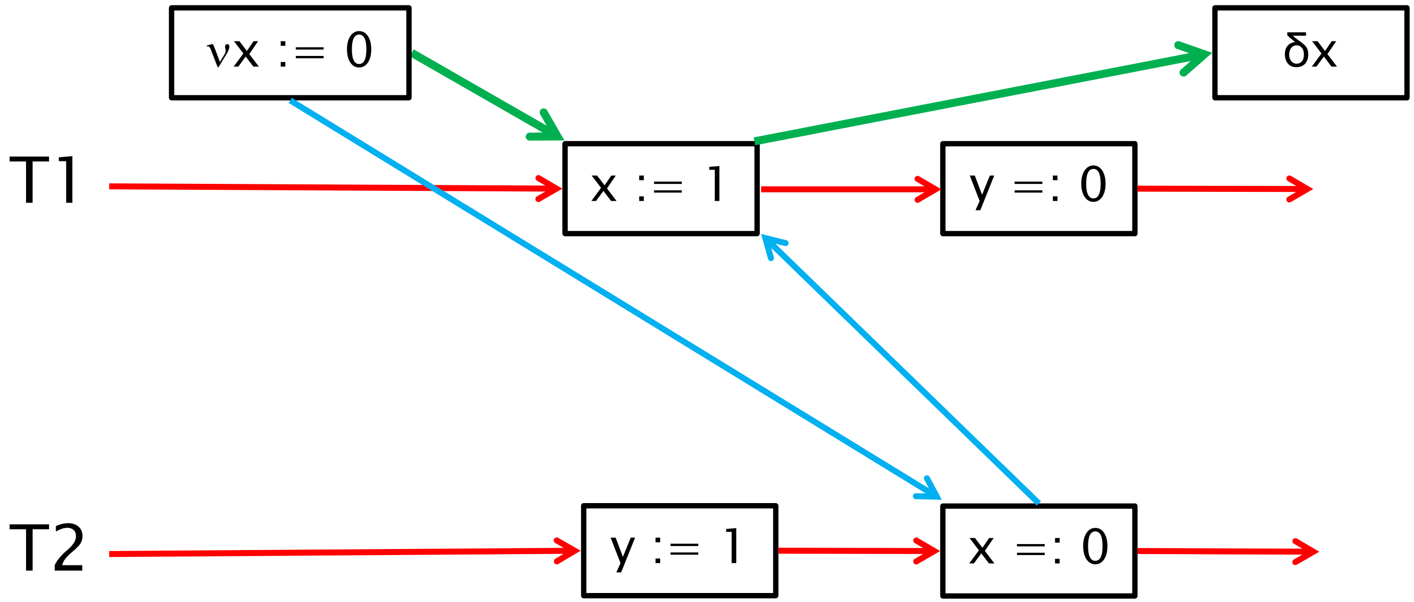
Weakly consistent memory

- as implemented in multi-core architecture...
- ...complicates shared variable behaviour...
- ...both in definition and in use

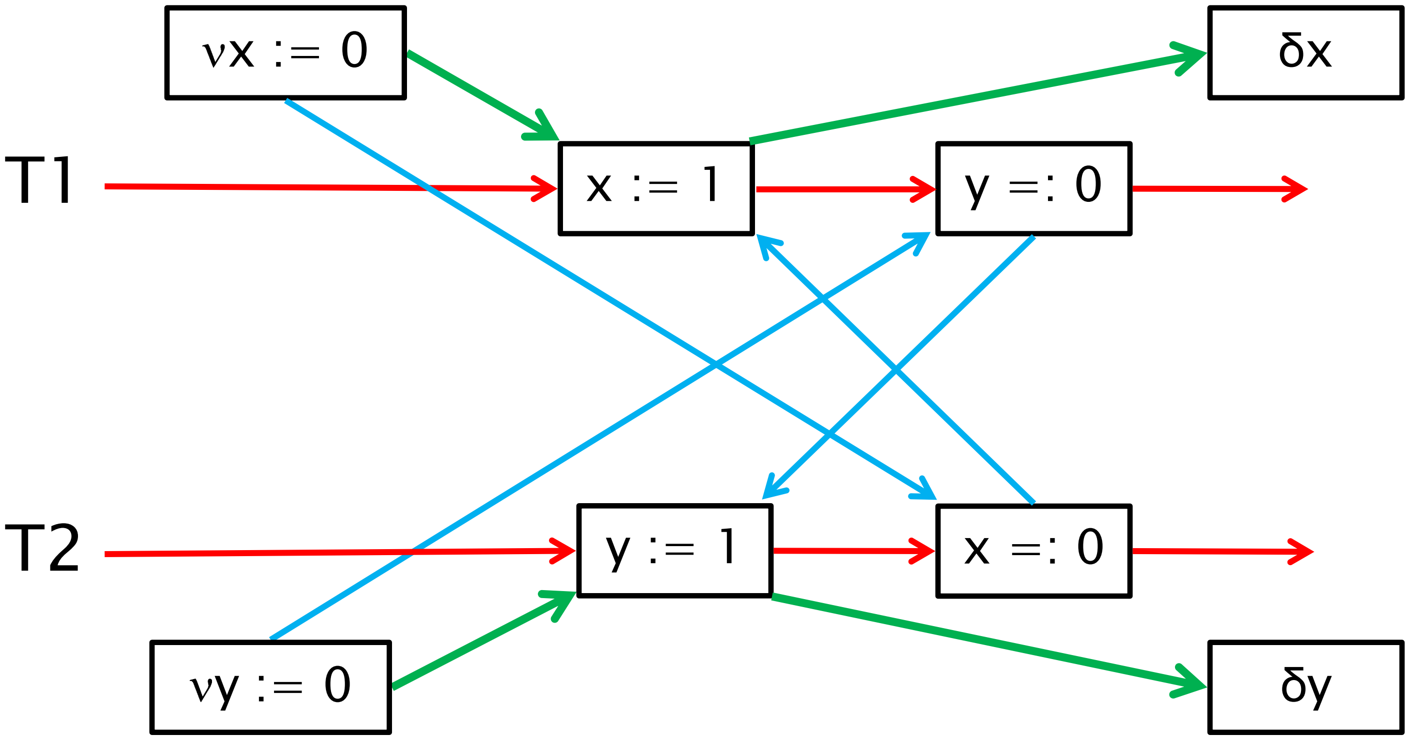
Litmus test



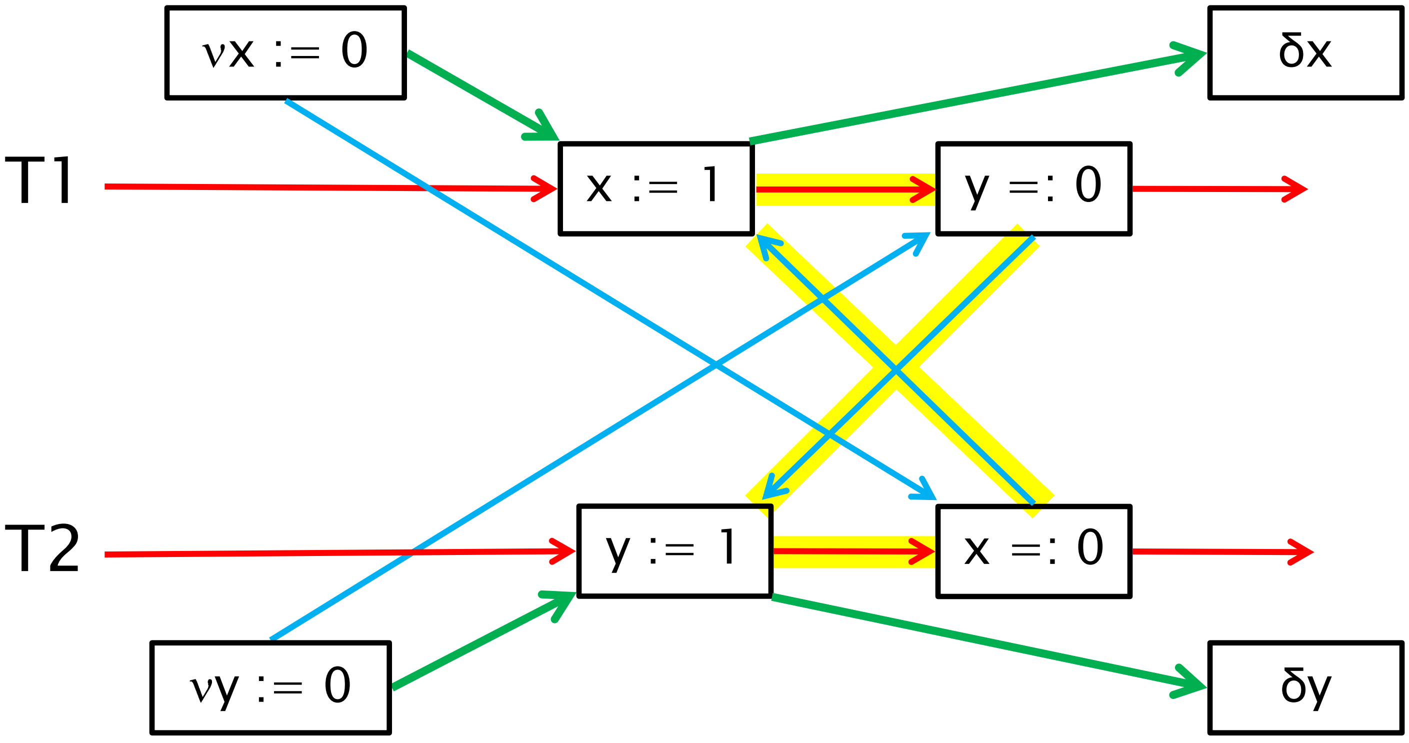
Litmus test



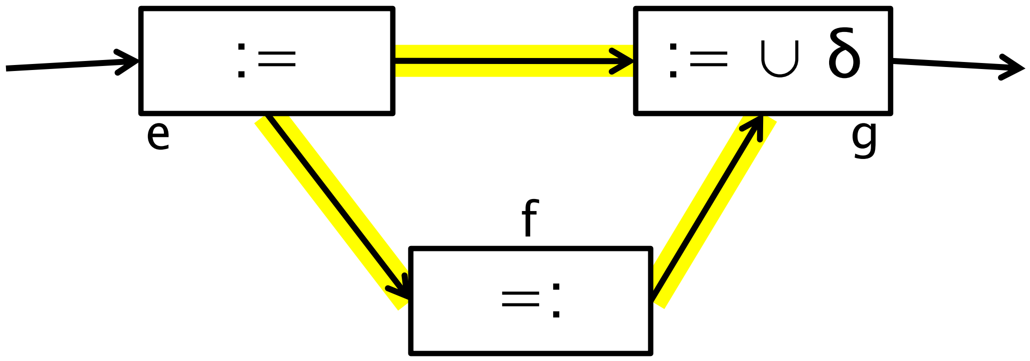
Litmus test

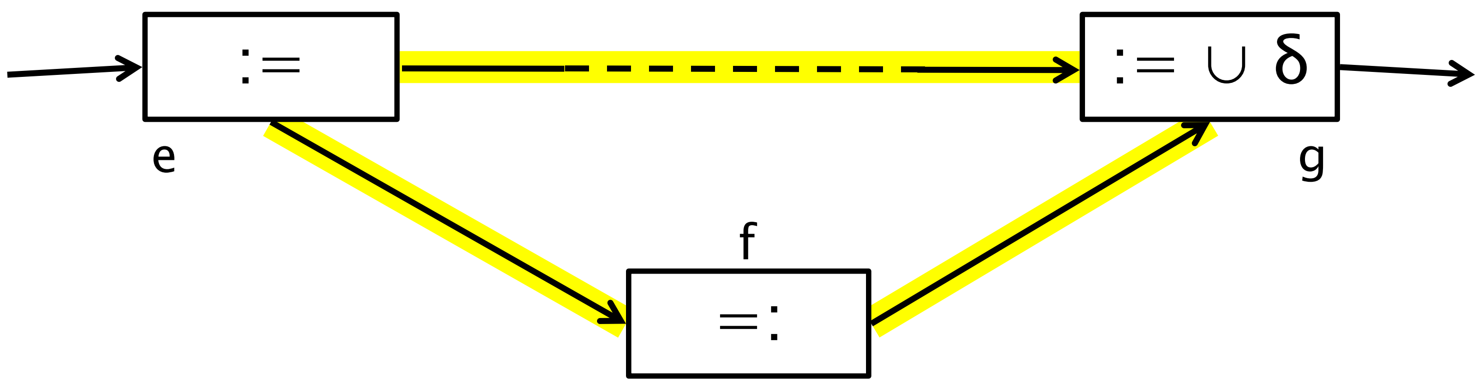


Litmus test

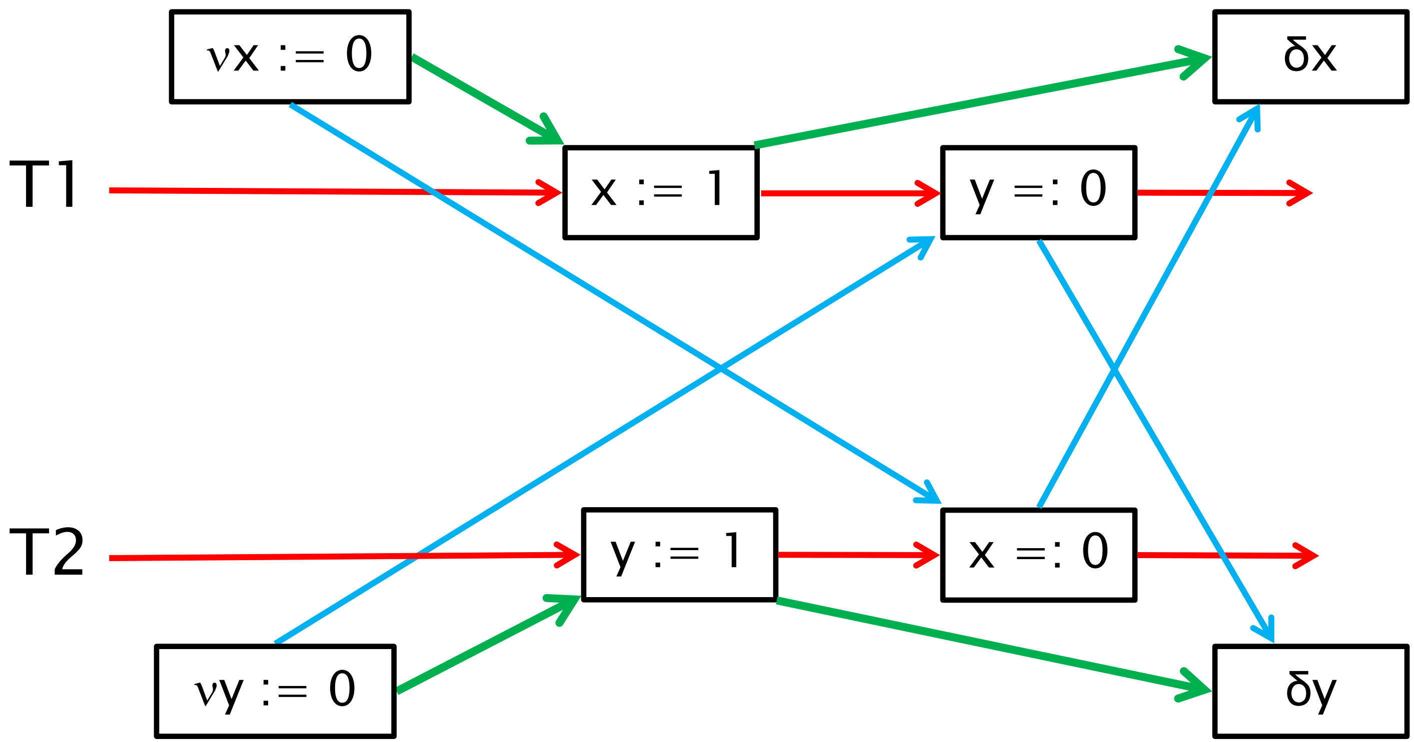


Relaxed triangle

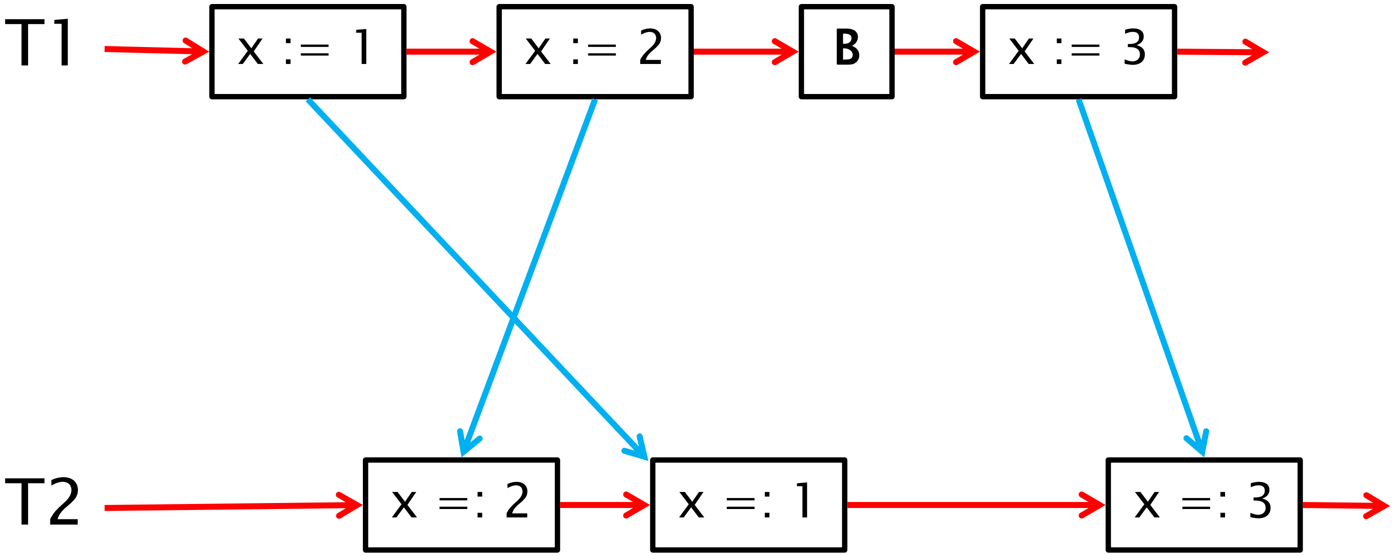




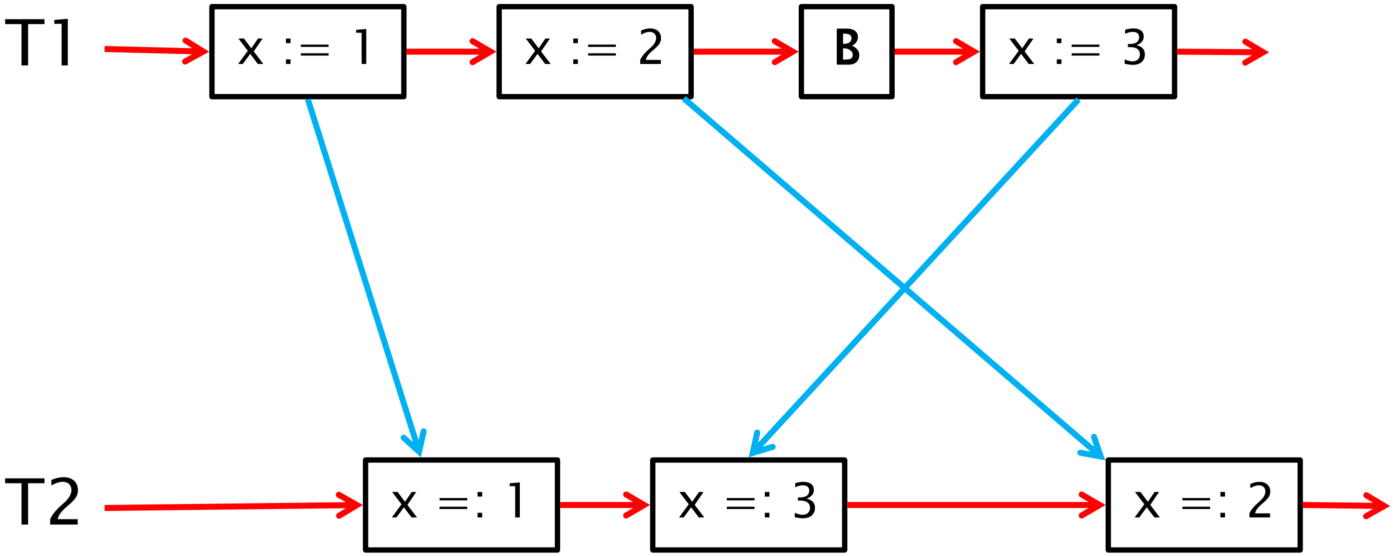
Litmus test



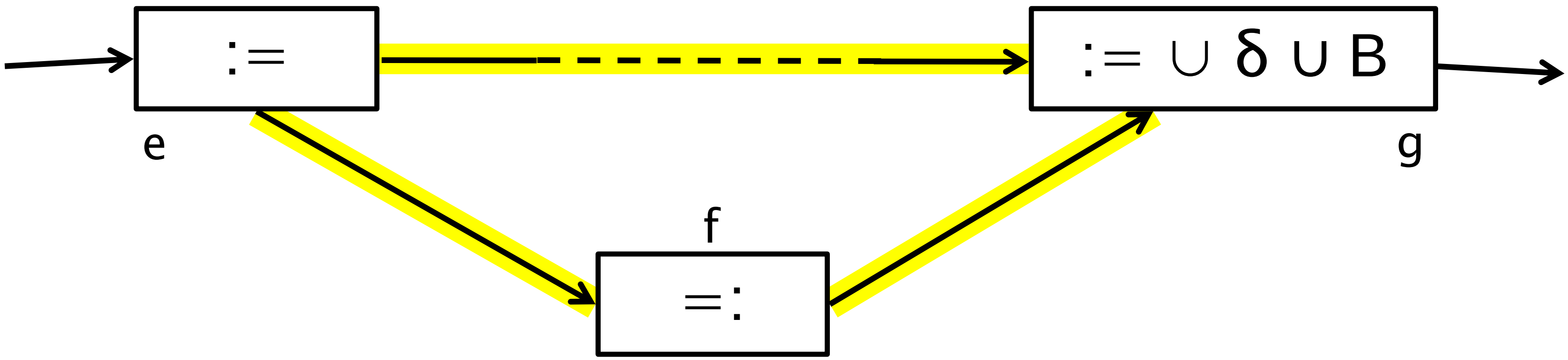
Memory barriers



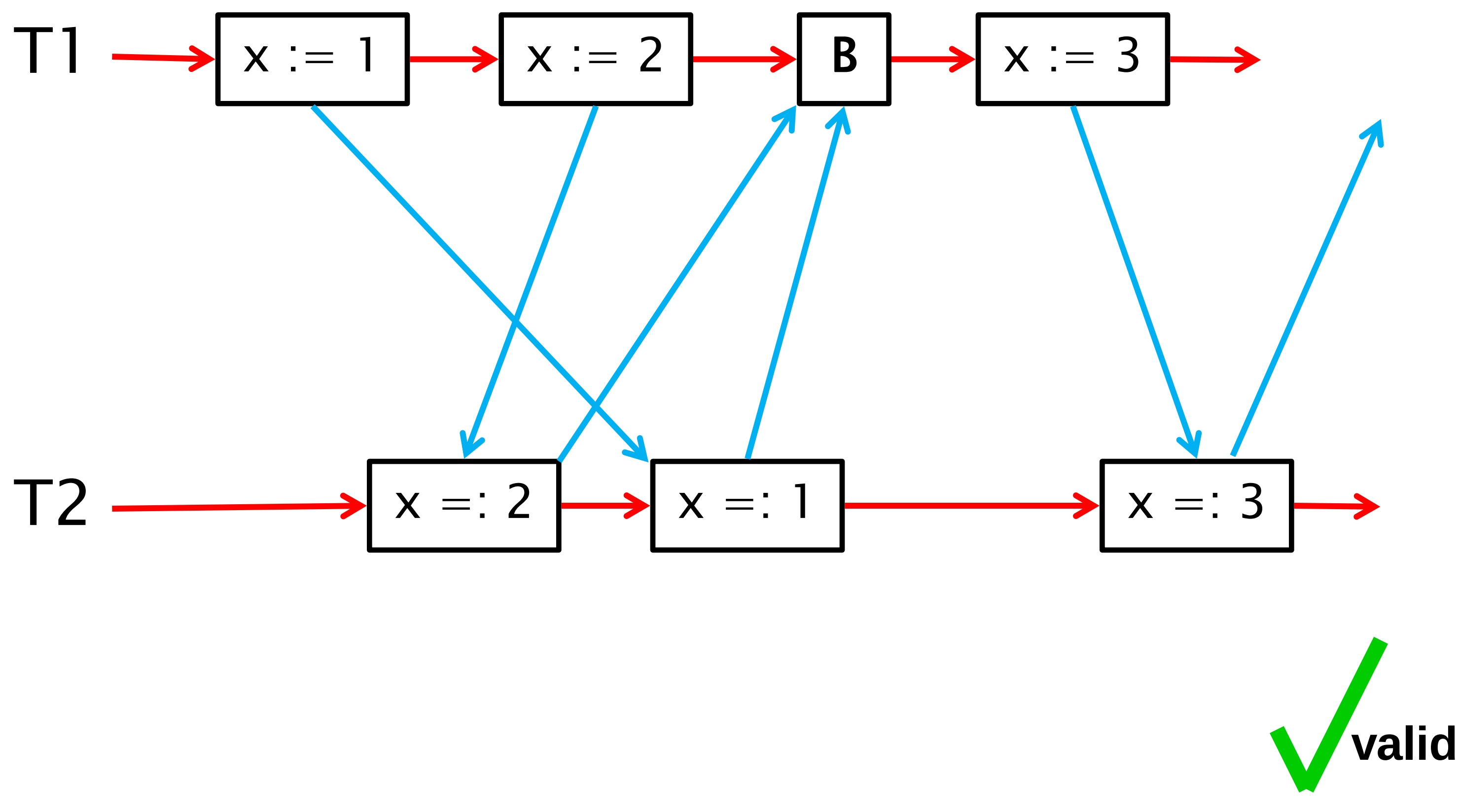
Memory barriers



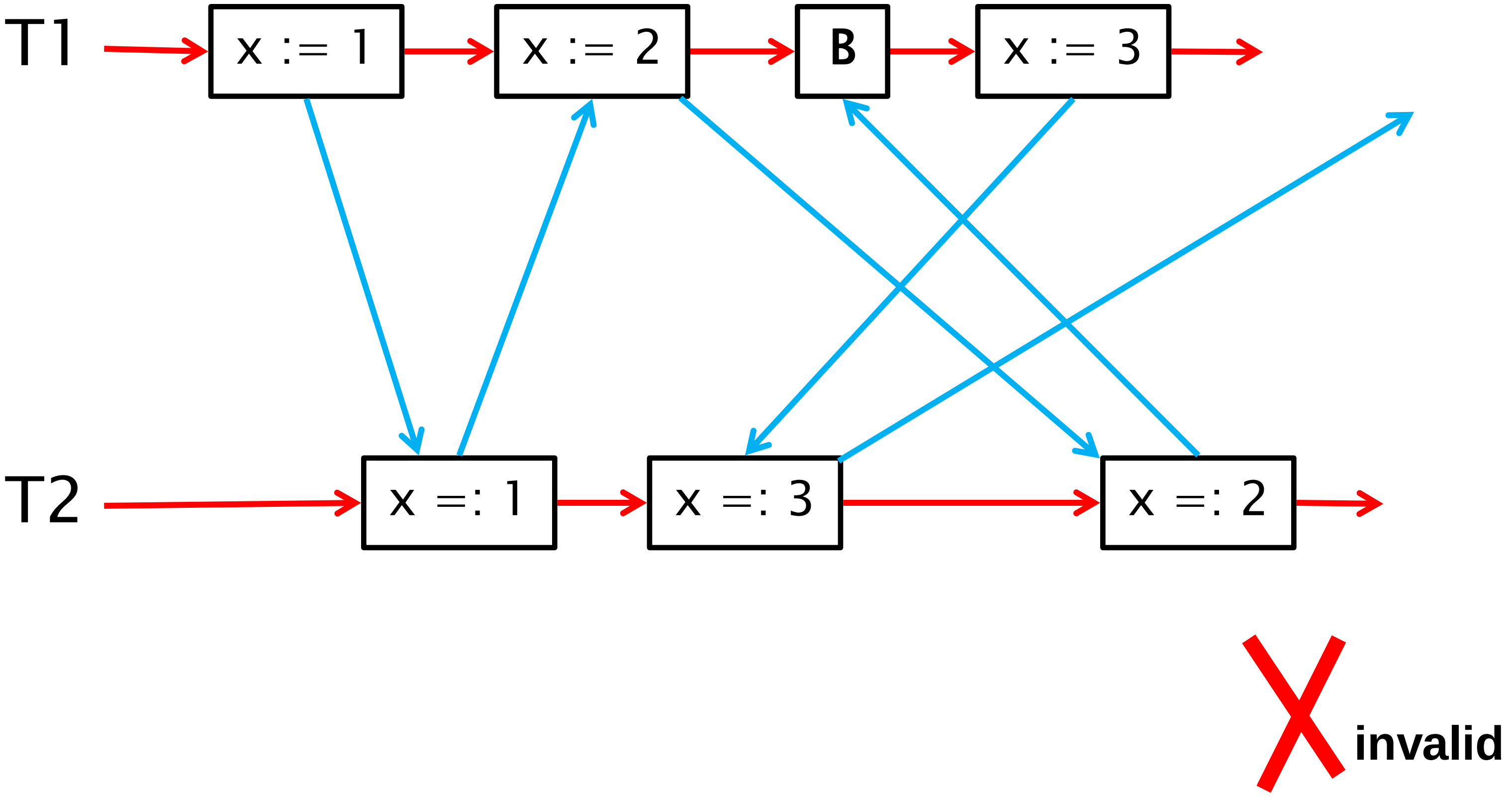
Memory barriers



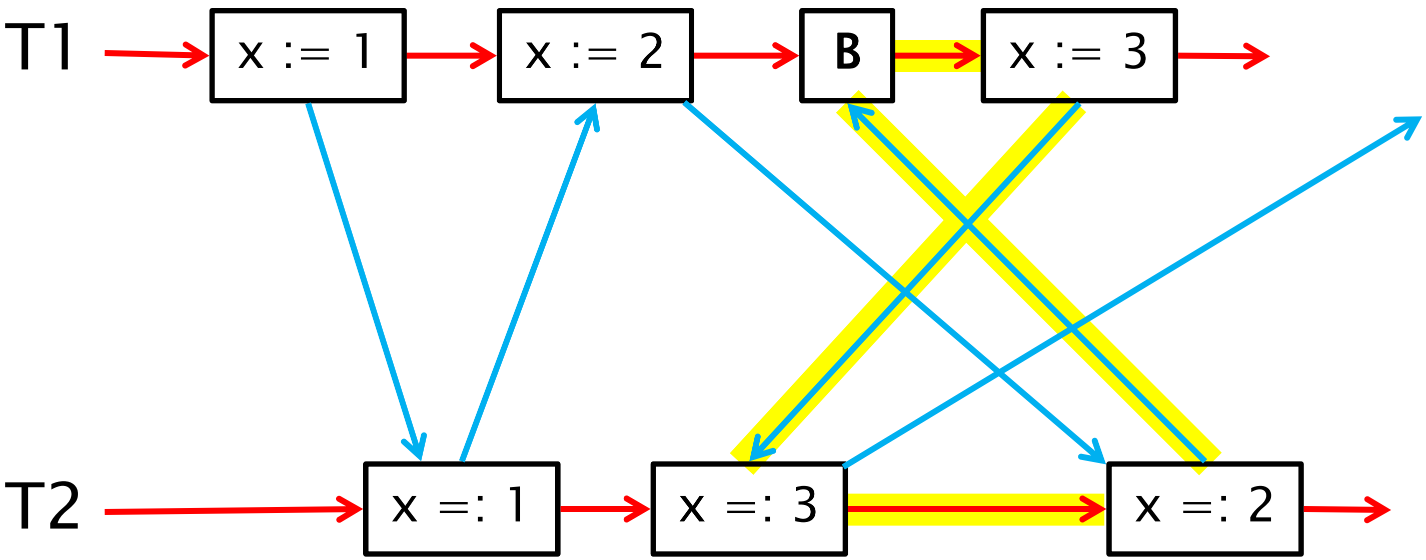
Memory barriers



Memory barriers

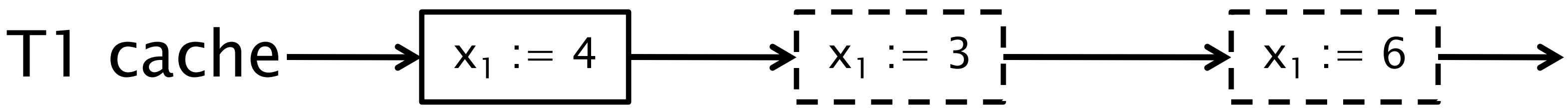


Memory barriers

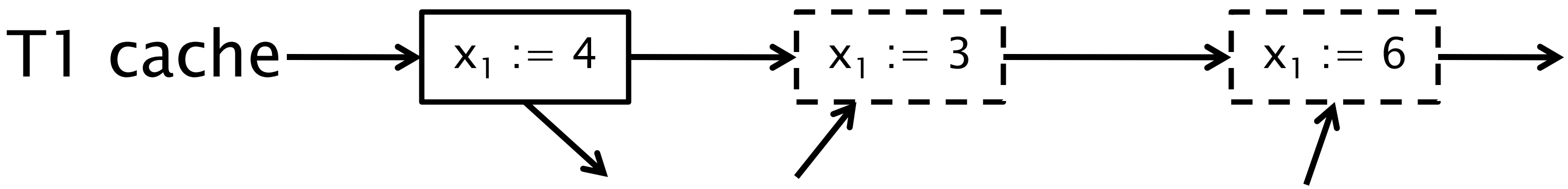


X invalid

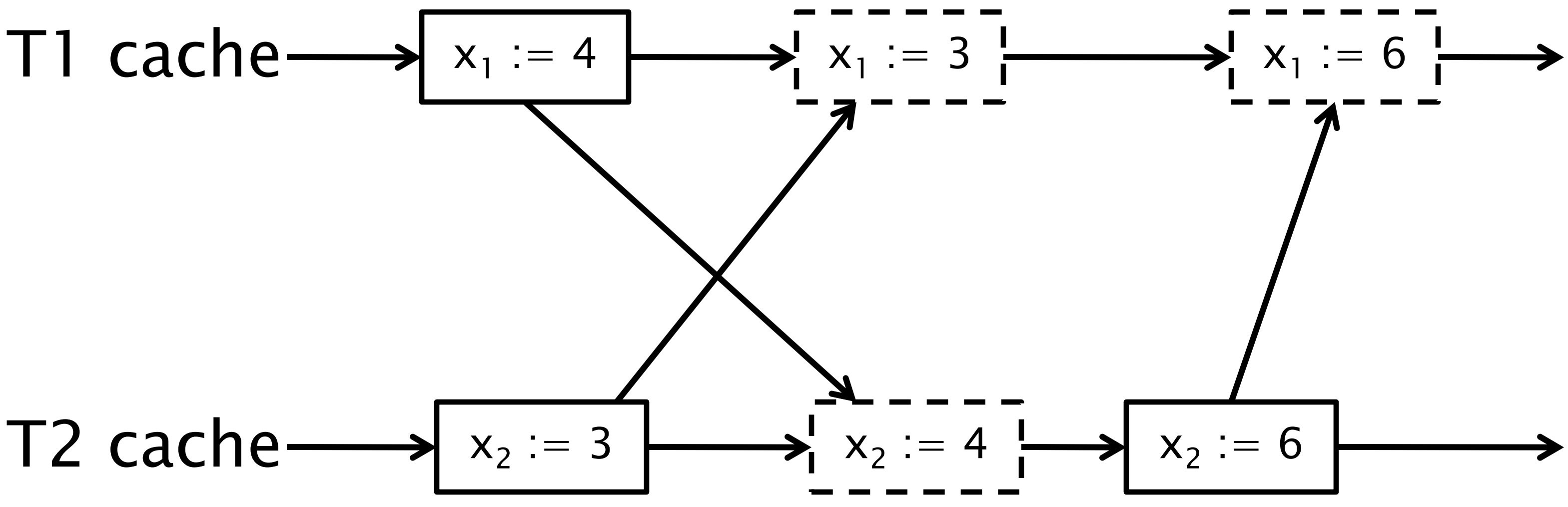
Cache (1)



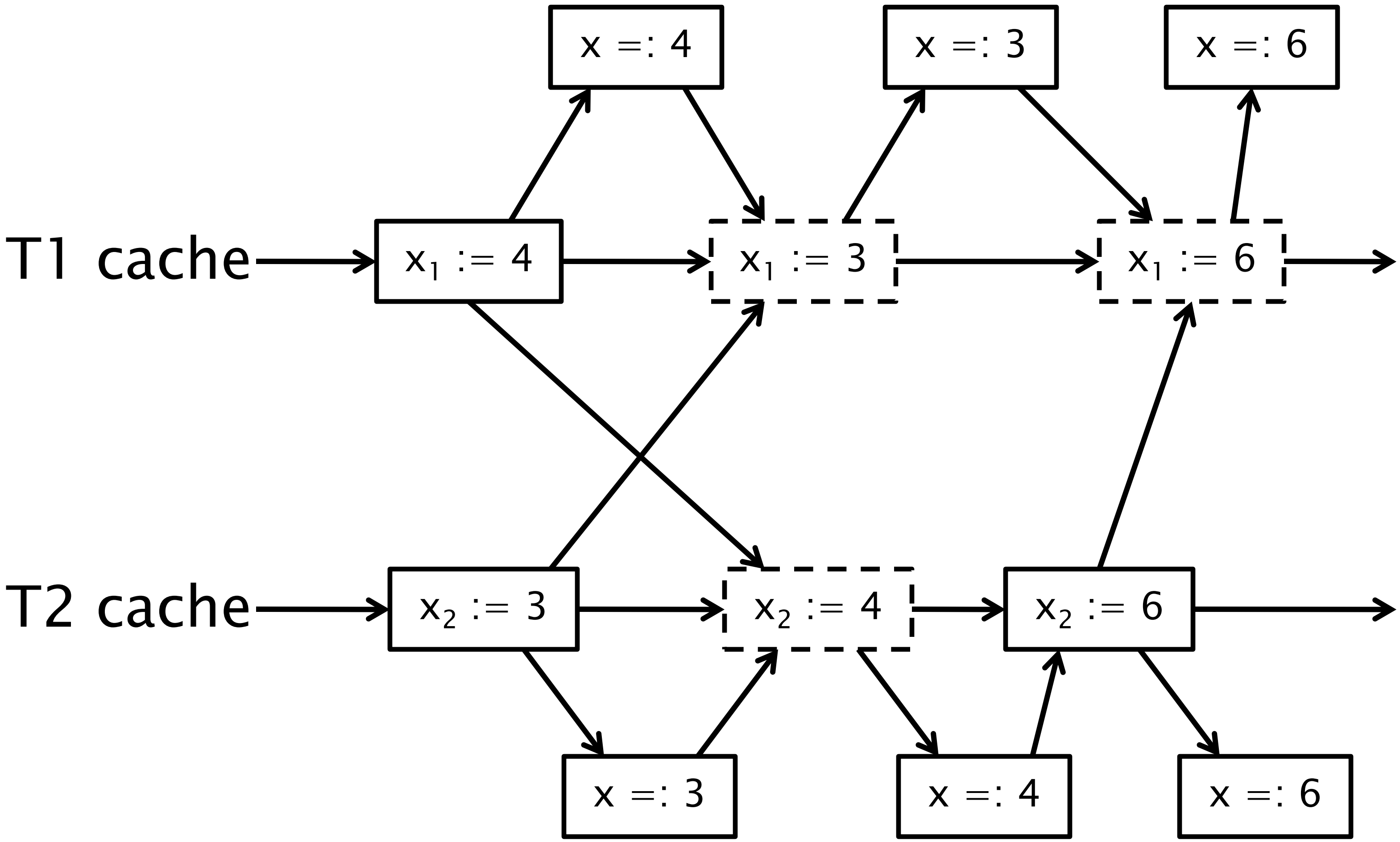
Cache (2)



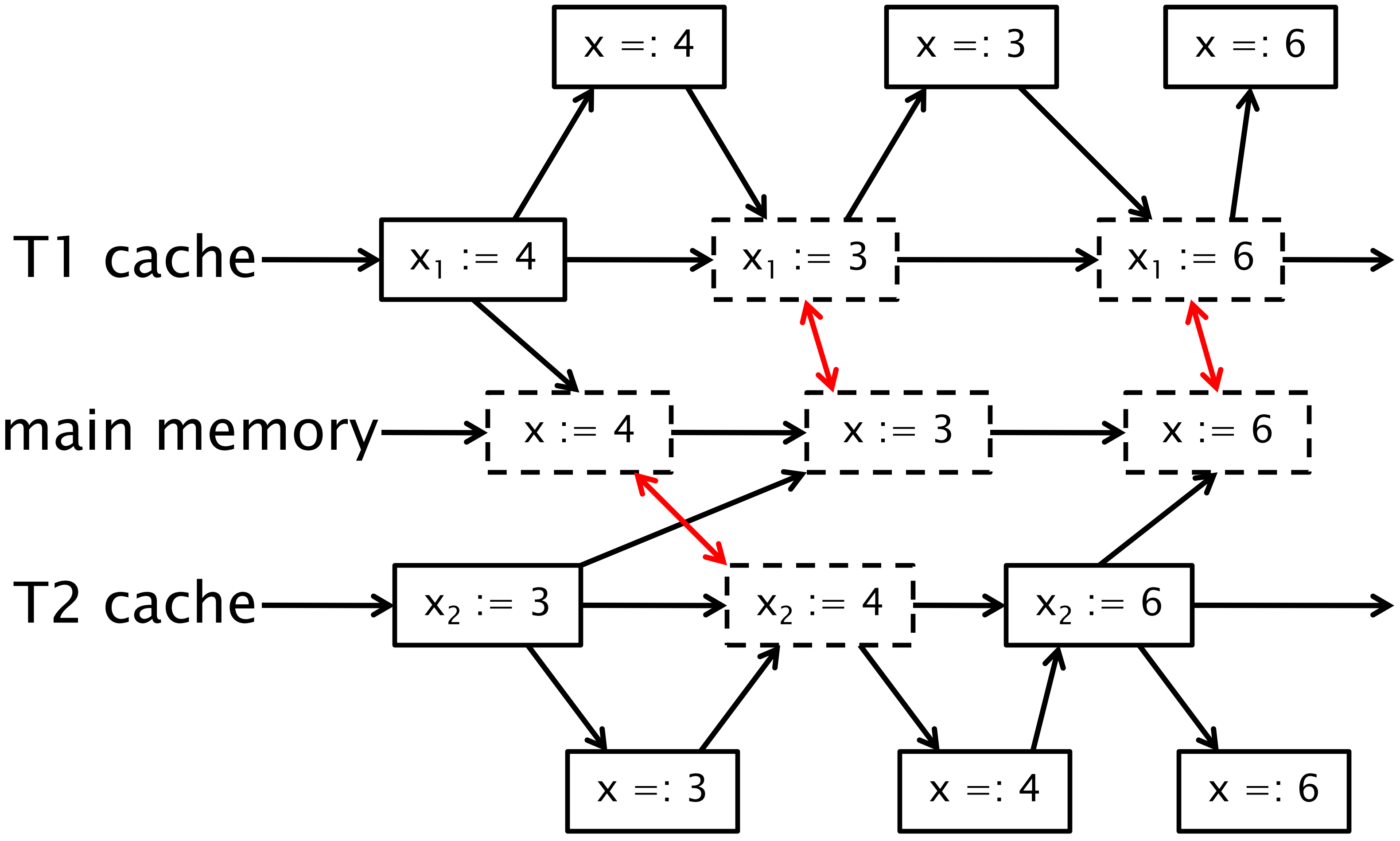
A second cache



Partial store ordering



Total store ordering



Summary

- Data flow is a primitive concept,
  - adequate to describe the dynamic behaviour of many kinds of computing resource.
- Relational calculus,
  - illustrated by labelled graphs,
  - provides a general framework adequate for a unifying theory of data flow

Acknowledgements

- Jay Misra
- Bernhard Möller
- Jørgen Steensgaard
- Viktor Vafeiadis
- Peter Höfner
- And especially John Wickerson