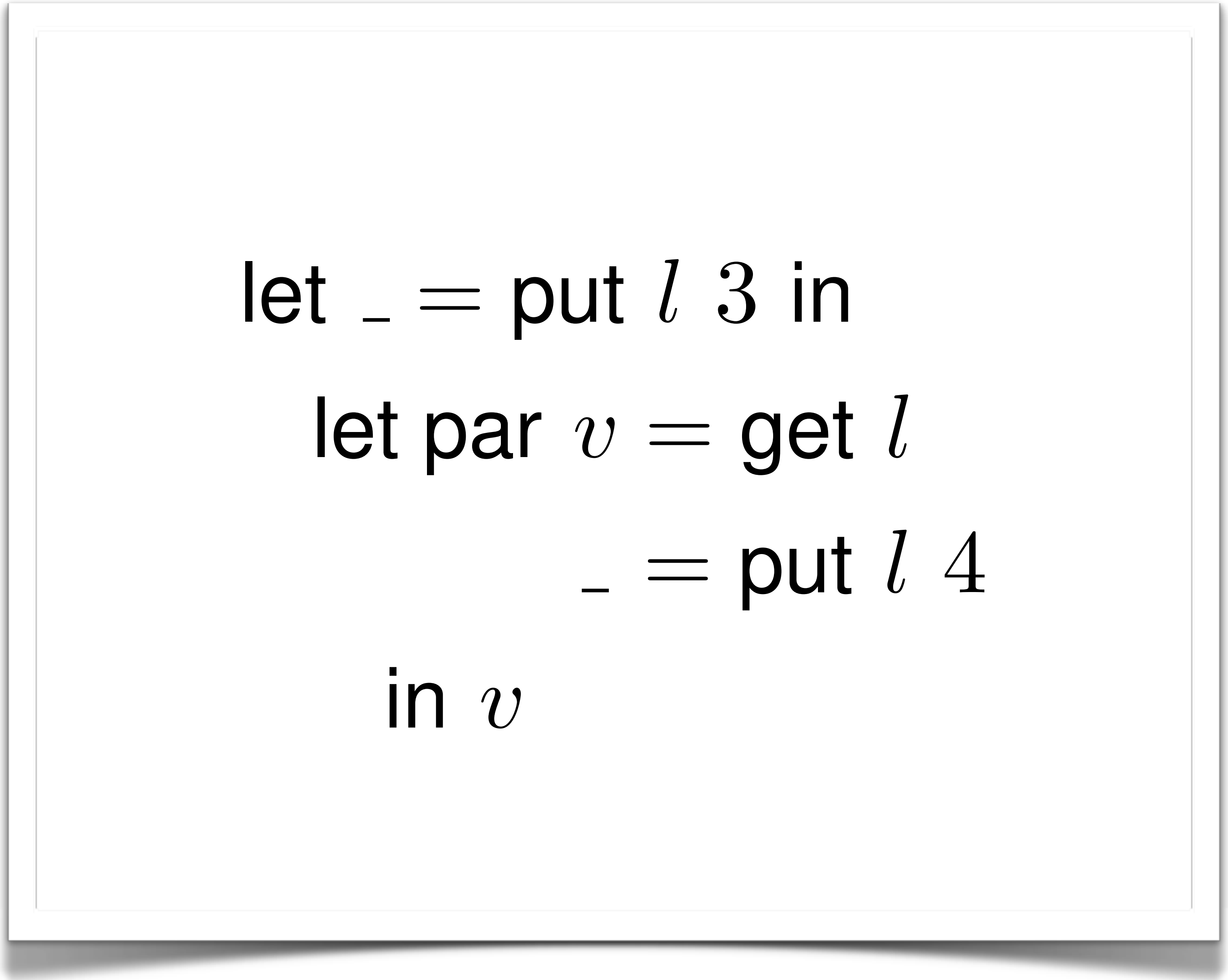
# A Lattice-Based Approach to Deterministic Parallelism with Shared State

**Lindsey Kuper** and Ryan R. Newton
Indiana University
Bloomington, Indiana, USA

Aarhus University
14 September 2012



```
let _ = put $l$ $3$ in
  let par $v$ = get $l$
              _ = put $l$ $4$
    in $v$
```

# What do we want?

# What do we want?

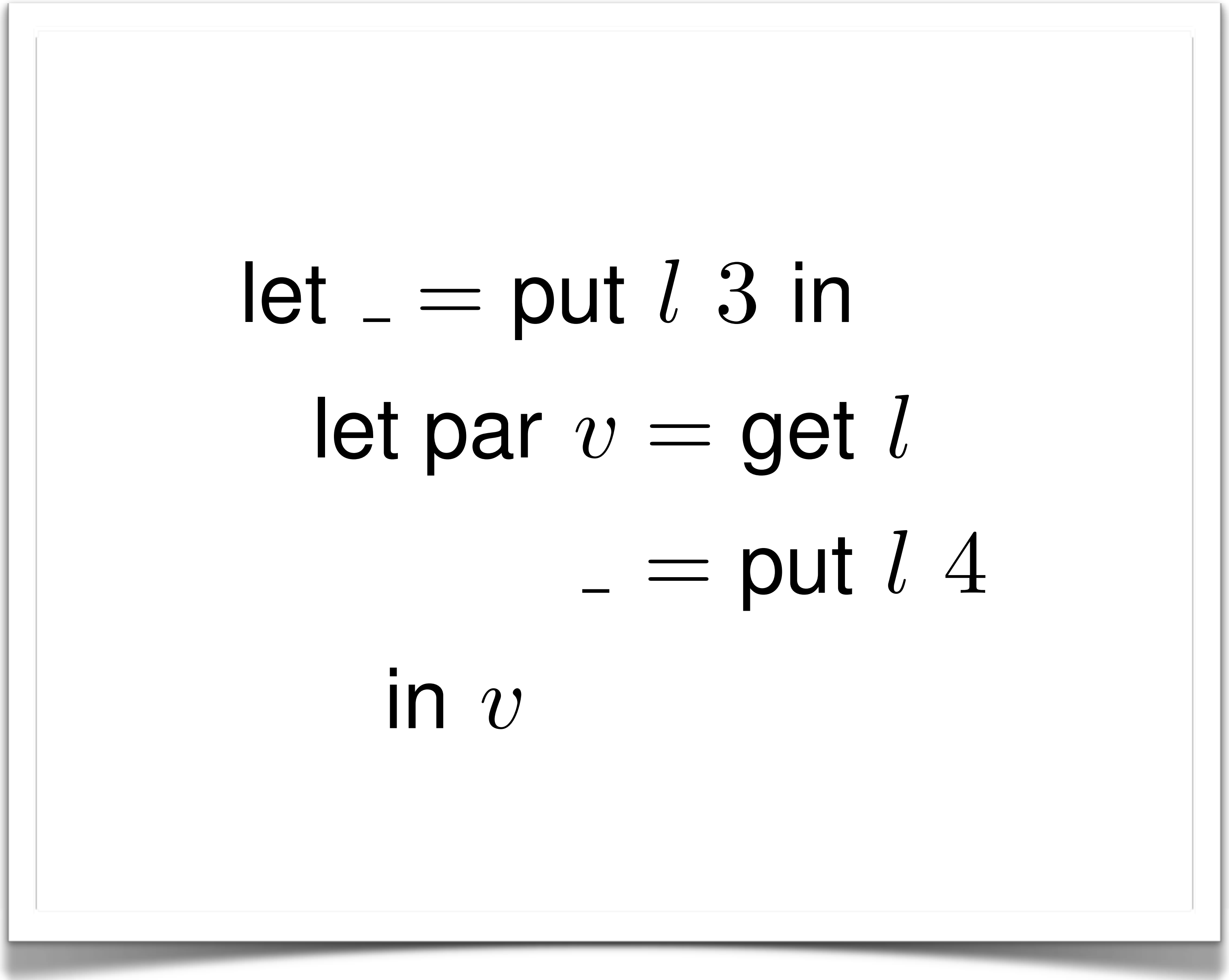
- A **deterministic** program is one that always produces the same observable result on multiple runs.

# What do we want?

- A **deterministic** program is one that always produces the same observable result on multiple runs.
- A **deterministic-by-construction** programming model is one that only allows deterministic programs to be written.

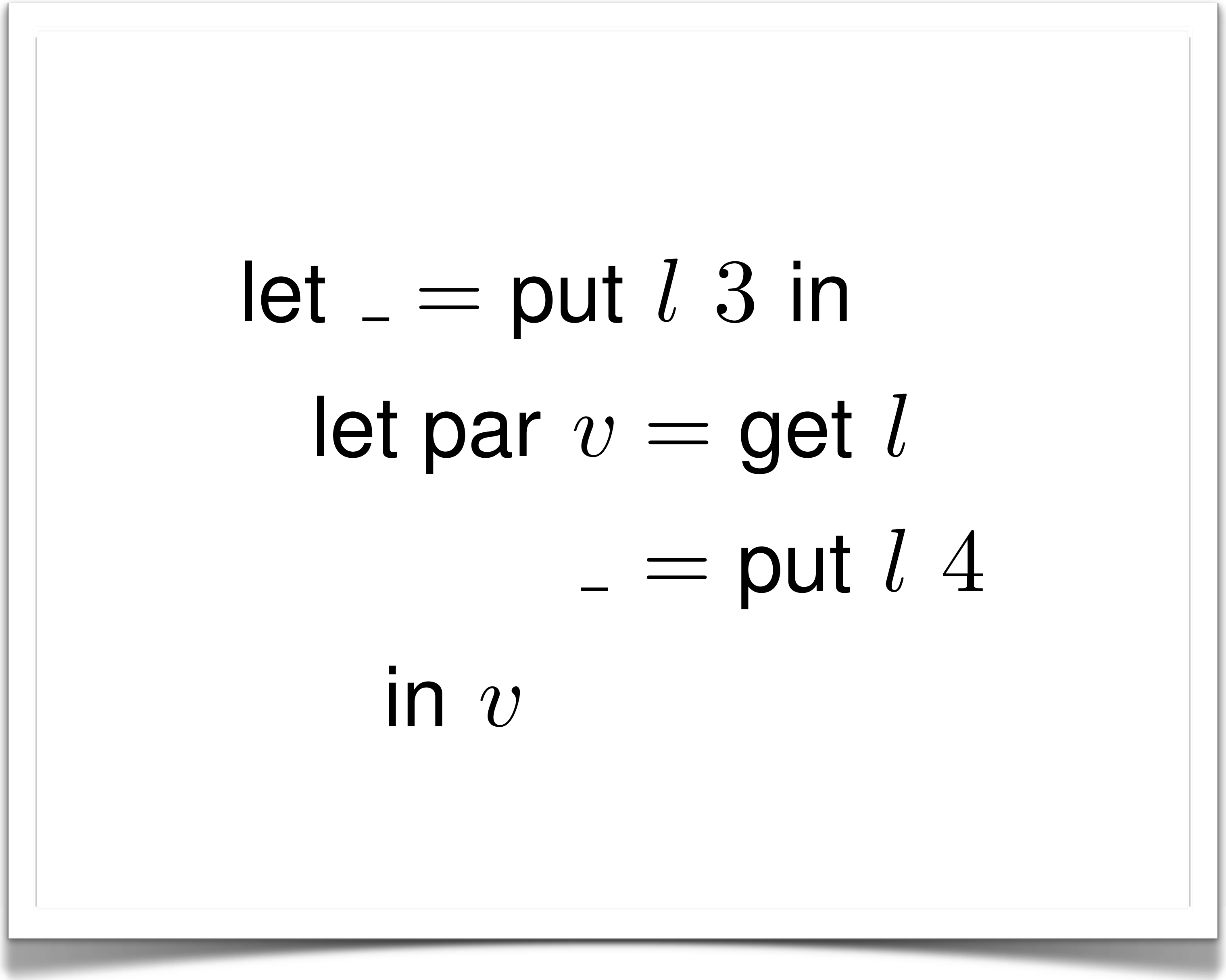
# What do we want?

- A **deterministic** program is one that always produces the same observable result on multiple runs.
- A **deterministic-by-construction** programming model is one that only allows deterministic programs to be written.
  - Examples: Kahn process networks, Intel Concurrent Collections, Haskell's monad-par, ...



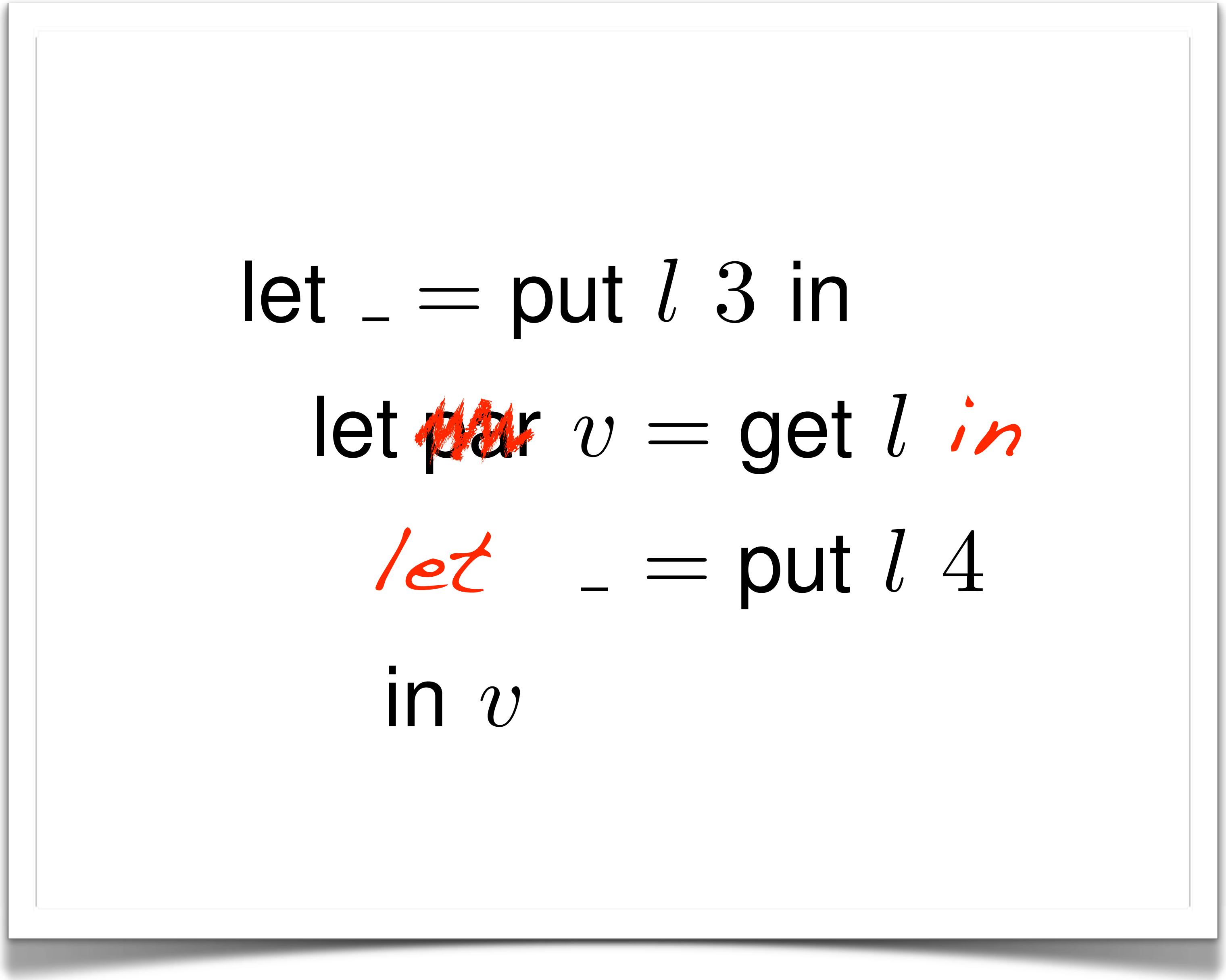
```
let _ = put l 3 in
  let par v = get l
              _ = put l 4
    in v
```

# Serialize?



```
let _ = put l 3 in
  let par v = get l
          _ = put l 4
    in v
```

# Serialize?



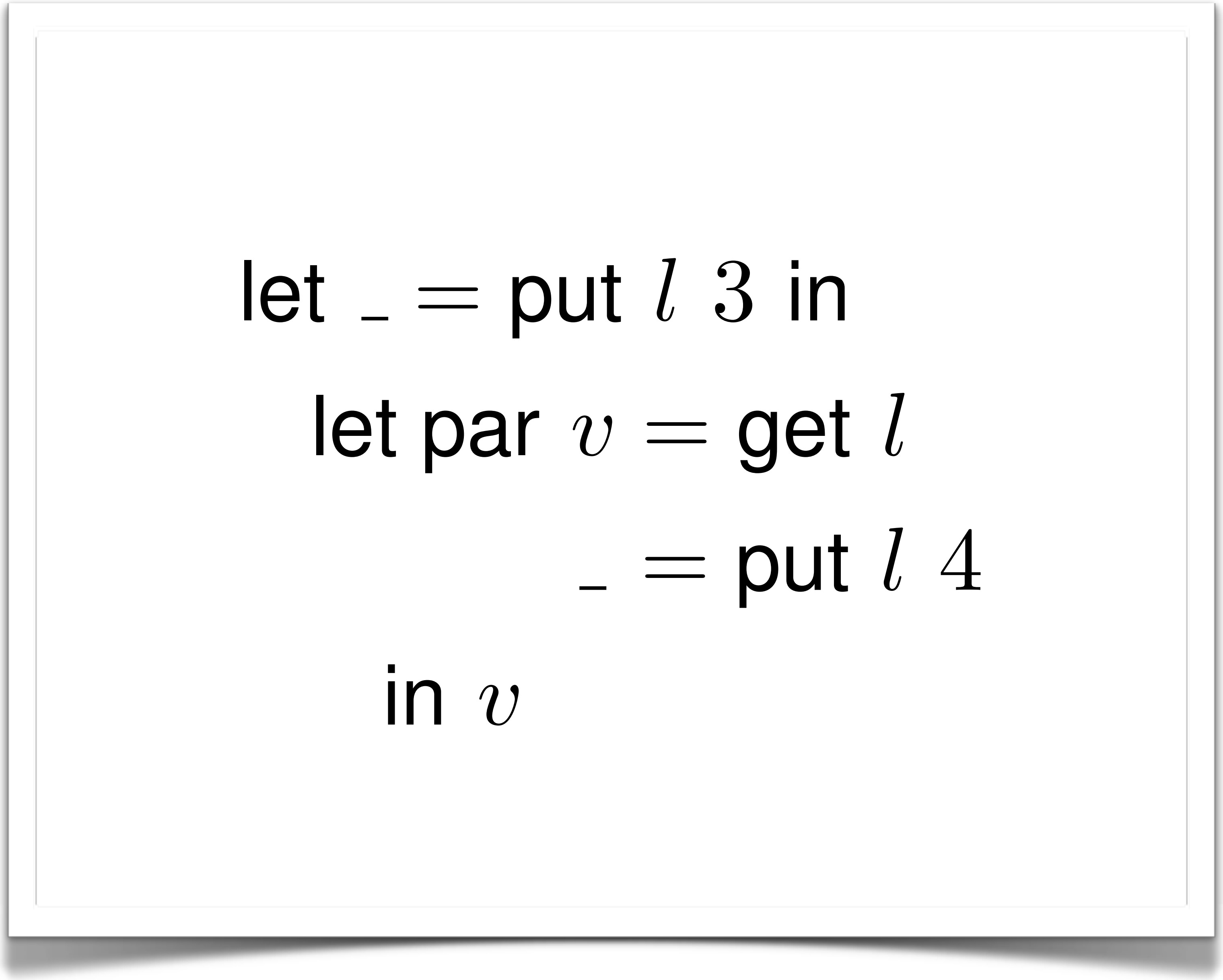
let $\_ = \mathsf{put}\ l\ 3$ in

  let ~~par~~ $v = \mathsf{get}\ l$ in

  let $\_ = \mathsf{put}\ l\ 4$

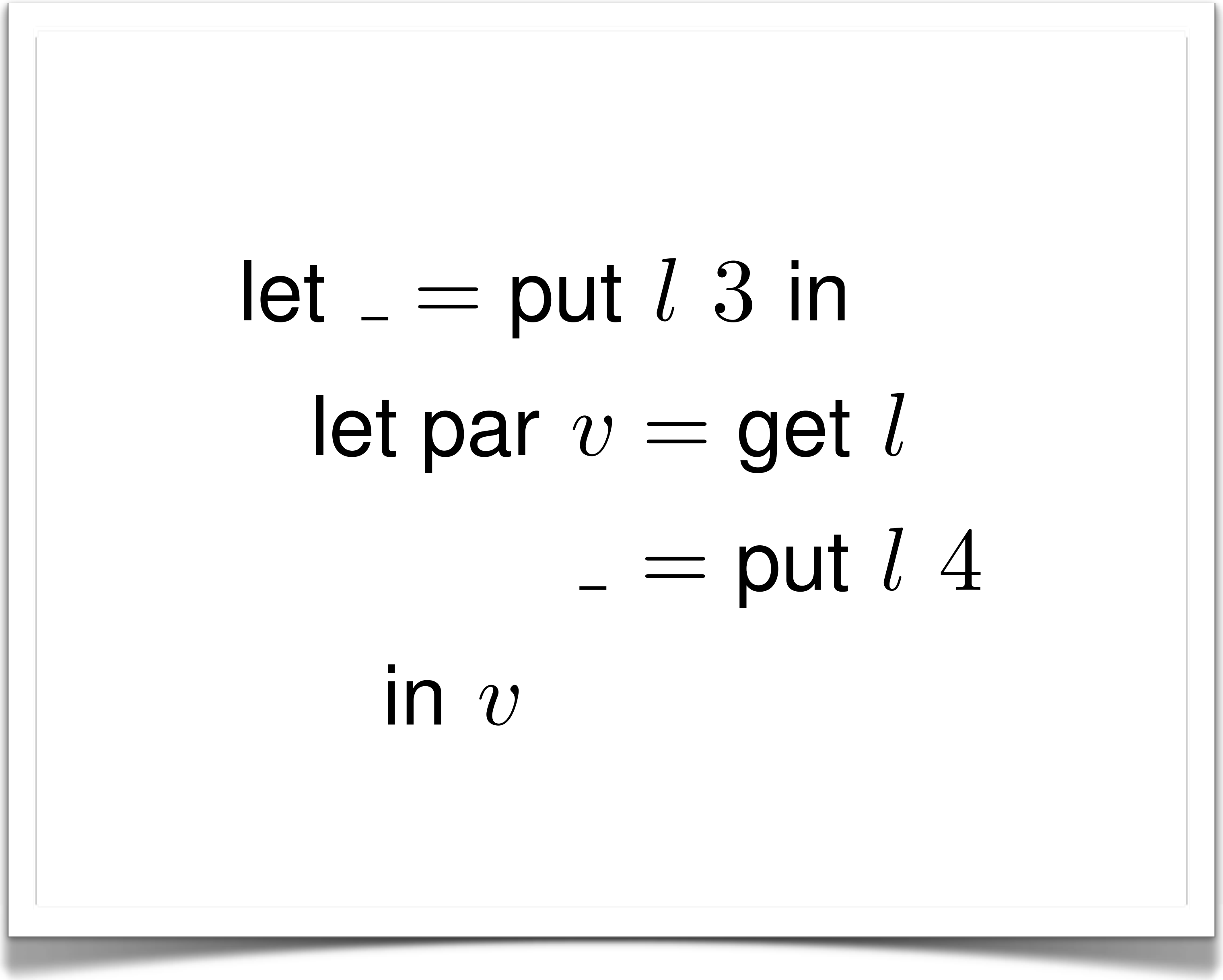
  in $v$

# Serialize?



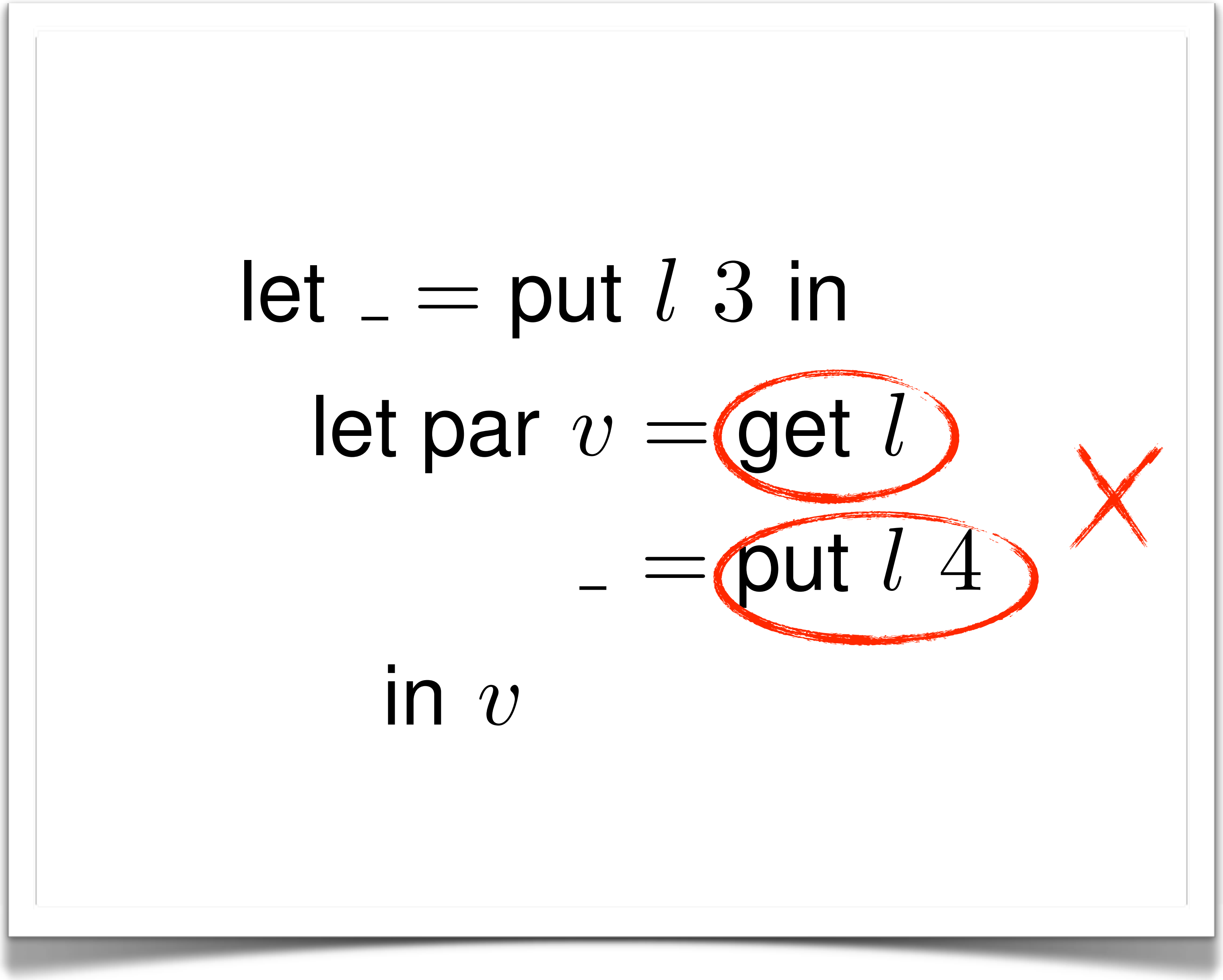
```
let _ = put $l$ $3$ in
  let par $v$ = get $l$
          _ = put $l$ $4$
    in $v$
```

# Disallow shared state?



```
let _ = put l 3 in
  let par v = get l
          _ = put l 4
    in v
```

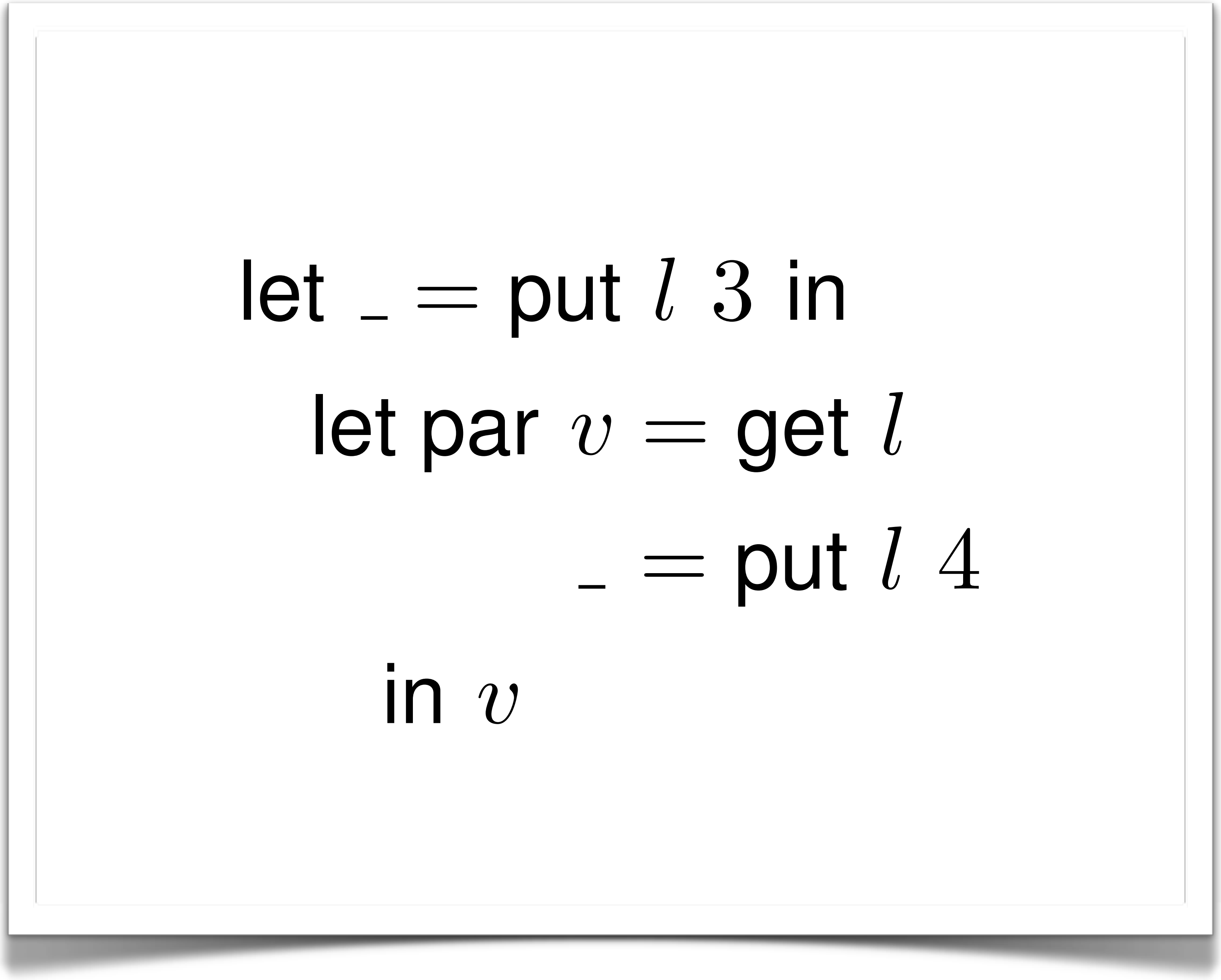
# Disallow shared state?



$$
\begin{aligned}
&\text{let } \_ = \text{put } l\ 3 \text{ in} \\
&\quad \text{let par } v = \text{get } l \\
&\quad \qquad\qquad \_ = \text{put } l\ 4 \\
&\quad \text{in } v
\end{aligned}
$$

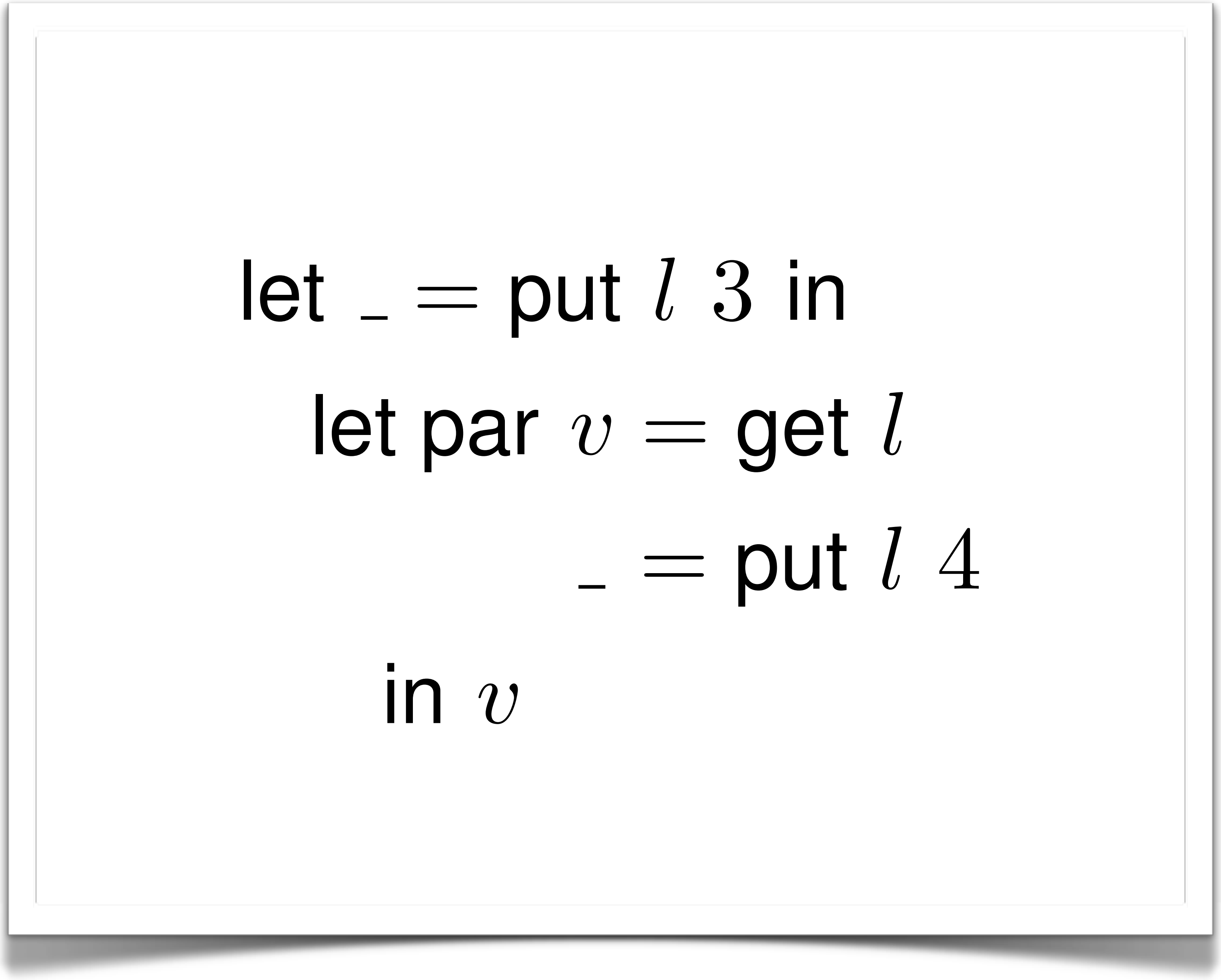
✗

# Disallow shared state?



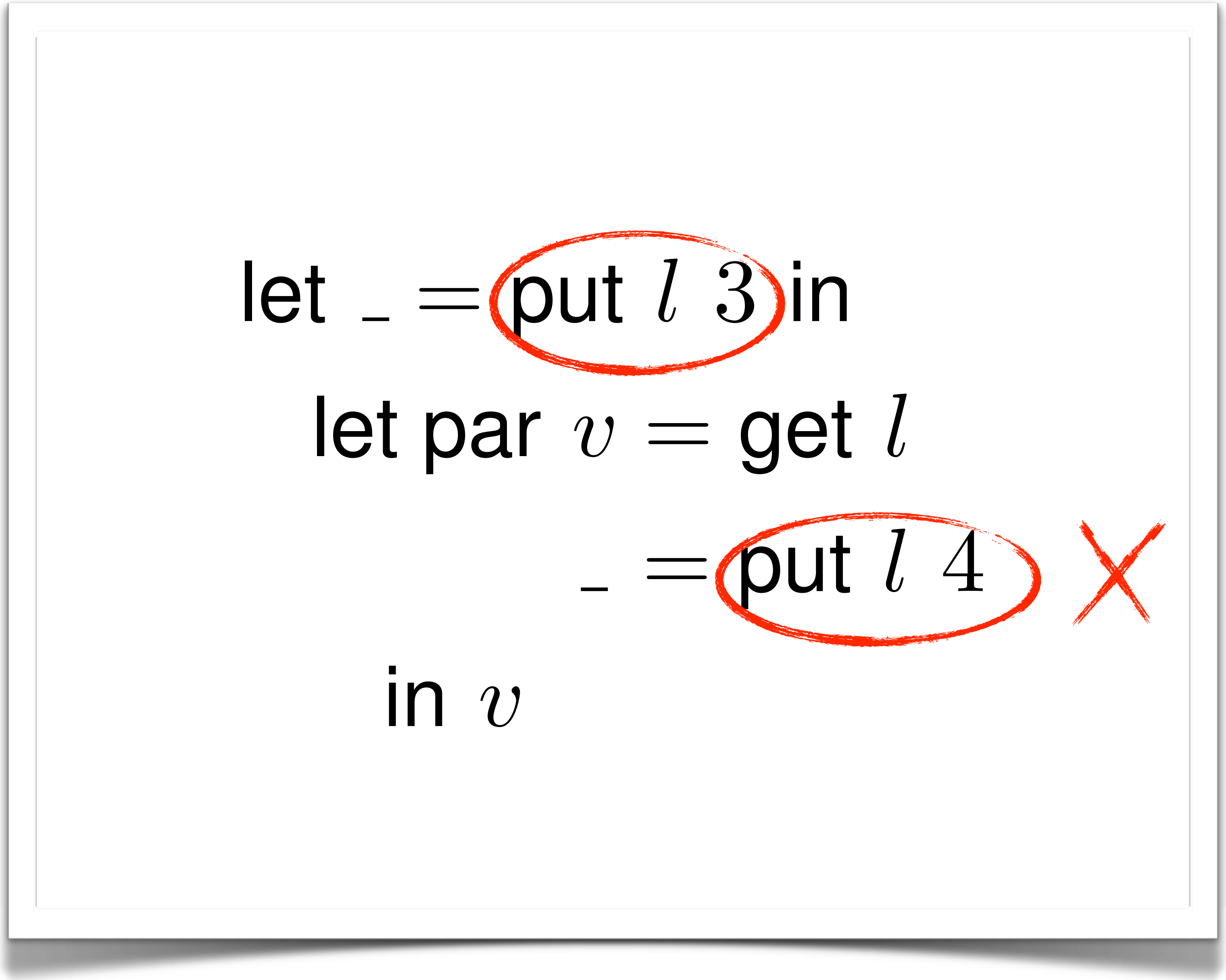
```
let _ = put $l$ $3$ in
  let par $v$ = get $l$
          _ = put $l$ $4$
    in $v$
```

# Disallow multiple assignment?



```
let _ = put $l$ $3$ in
  let par $v$ = get $l$
          _ = put $l$ $4$
    in $v$
```

# Disallow multiple assignment?



$$\begin{aligned} &\text{let } \_ = \boxed{\text{put } l\ 3} \text{ in}\\ &\quad \text{let par } v = \text{get } l\\ &\qquad\qquad\qquad \_ = \boxed{\text{put } l\ 4} \quad \times\\ &\quad \text{in } v \end{aligned}$$

# A few single-assignment languages

# A few single-assignment languages

- Historically:
  - Compel (Tesler and Enea, 1968)

# A few single-assignment languages

- Historically:
  - Compel (Tesler and Enea, 1968)
  - Id, I-Structures and IVars (Arvind *et al*., 1989)

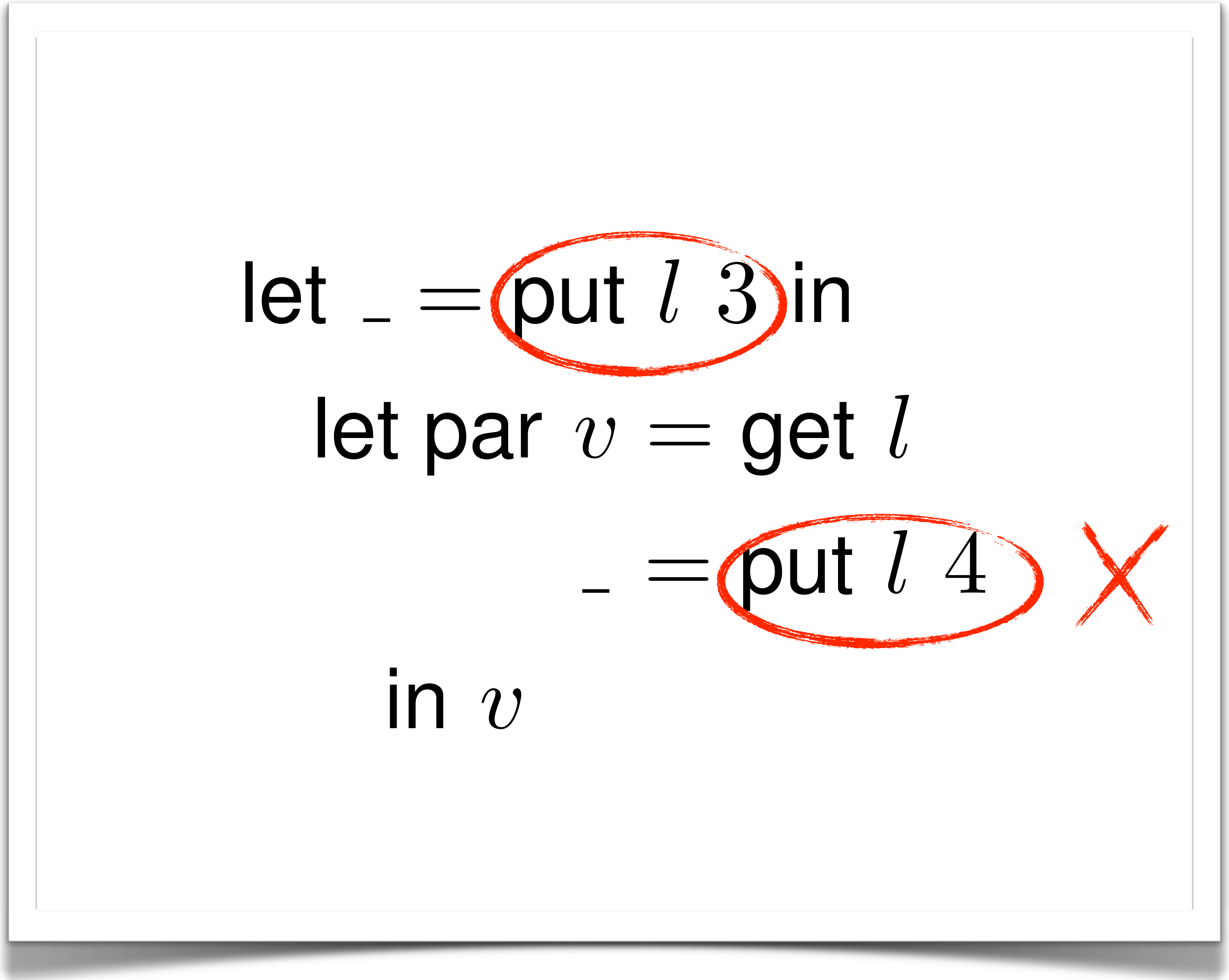
# A few single-assignment languages

- Historically:
  - Compel (Tesler and Enea, 1968)
  - Id, I-Structures and IVars (Arvind *et al.*, 1989)
- Today:
  - Intel Concurrent Collections (Budimlić *et al.*, 2010)
    - Specifically, *Featherweight CnC*

# A few single-assignment languages

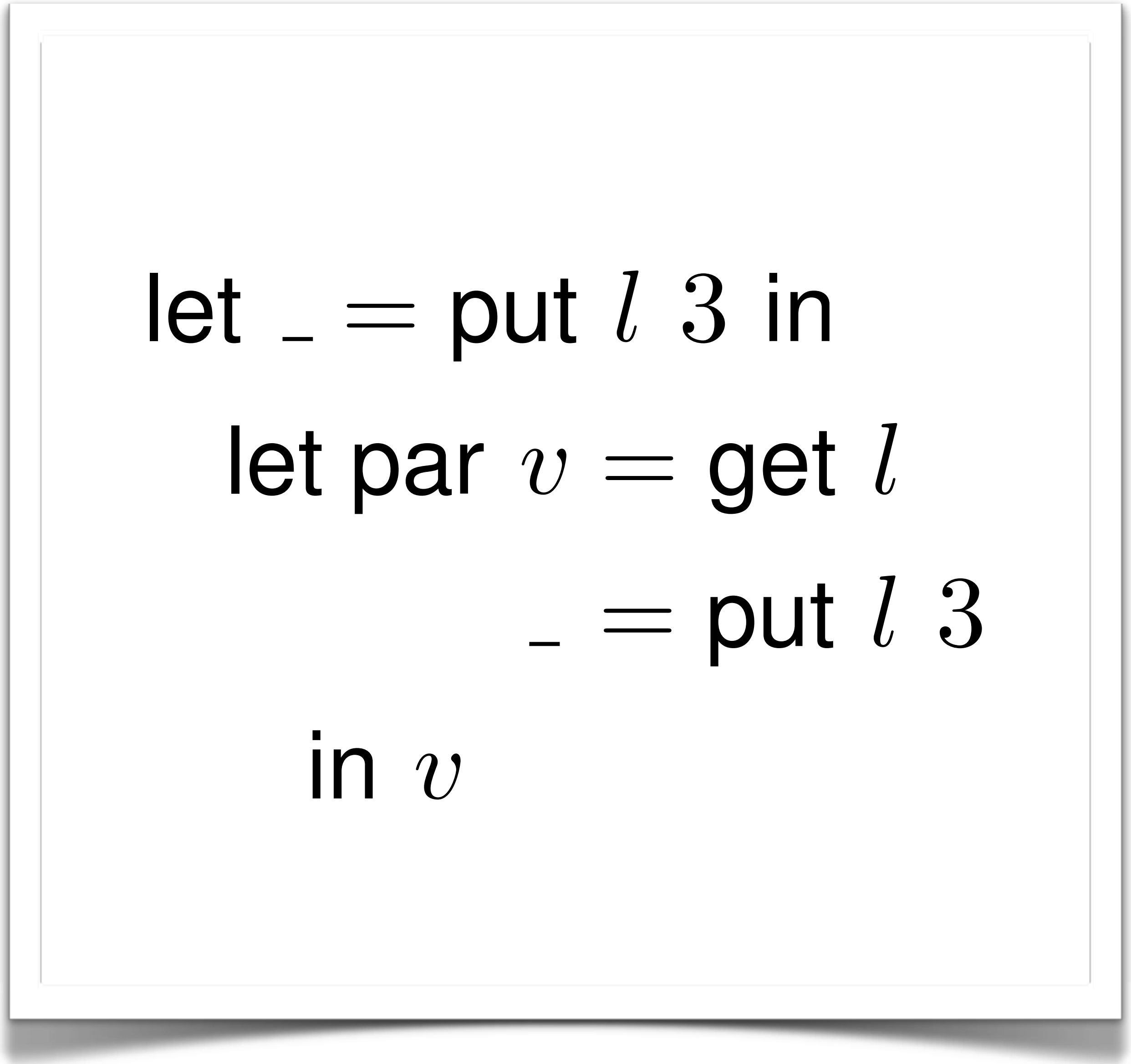
- Historically:
  - Compel (Tesler and Enea, 1968)
  - Id, I-Structures and IVars (Arvind *et al.*, 1989)
- Today:
  - Intel Concurrent Collections (Budimlić *et al.*, 2010)
    - Specifically, *Featherweight CnC*
  - monad-par for Haskell (Marlow *et al.*, 2011)

# Disallow multiple assignment?



```
let _ = put l 3 in
  let par v = get l
          _ = put l 4   ✗
  in v
```

# Deterministic programs that single-assignment forbids



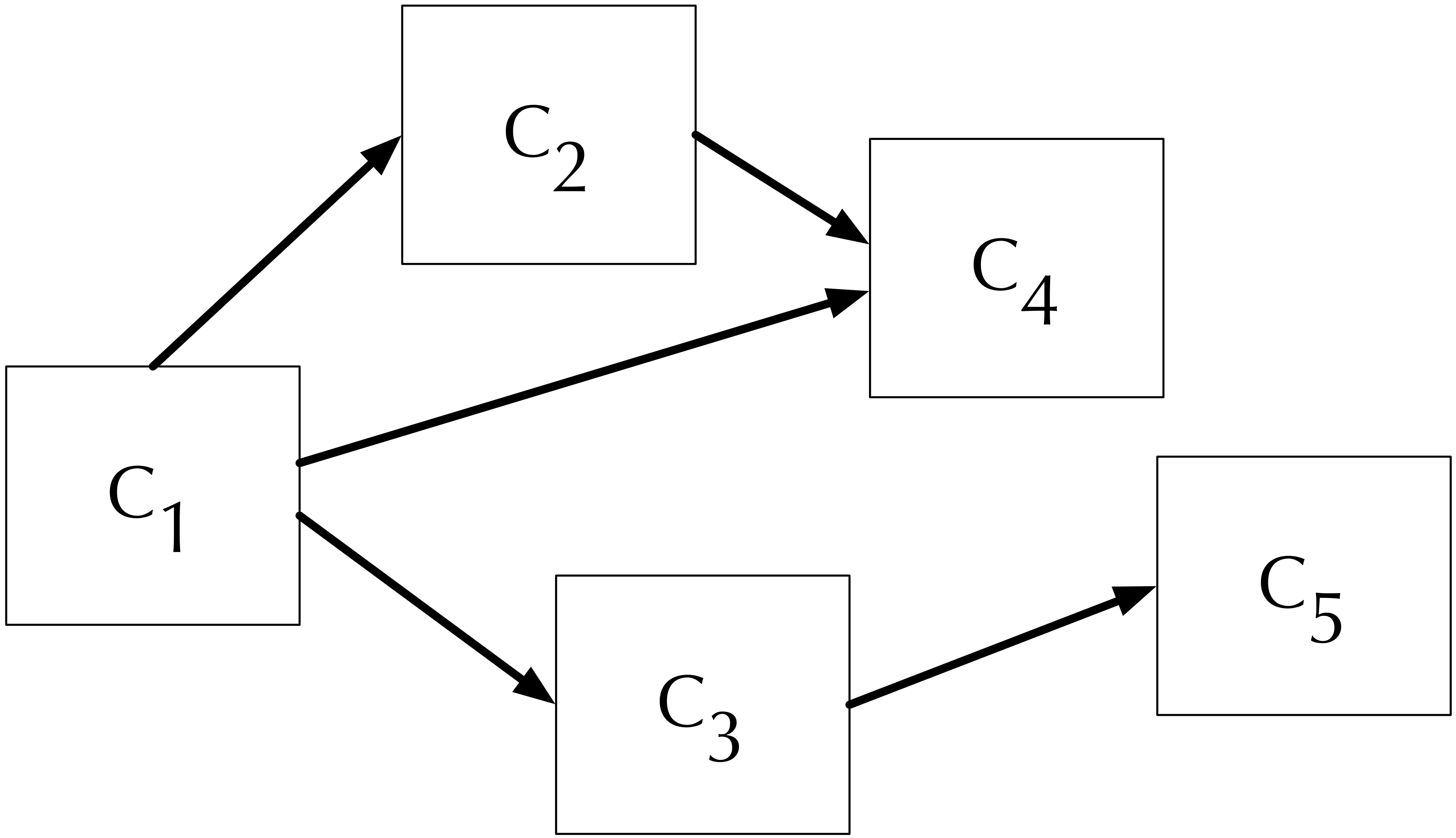
```
let $\_ =$ put $l$ $3$ in
  let par $v =$ get $l$
          $\_ =$ put $l$ $3$
    in $v$
```

# Deterministic programs that single-assignment forbids

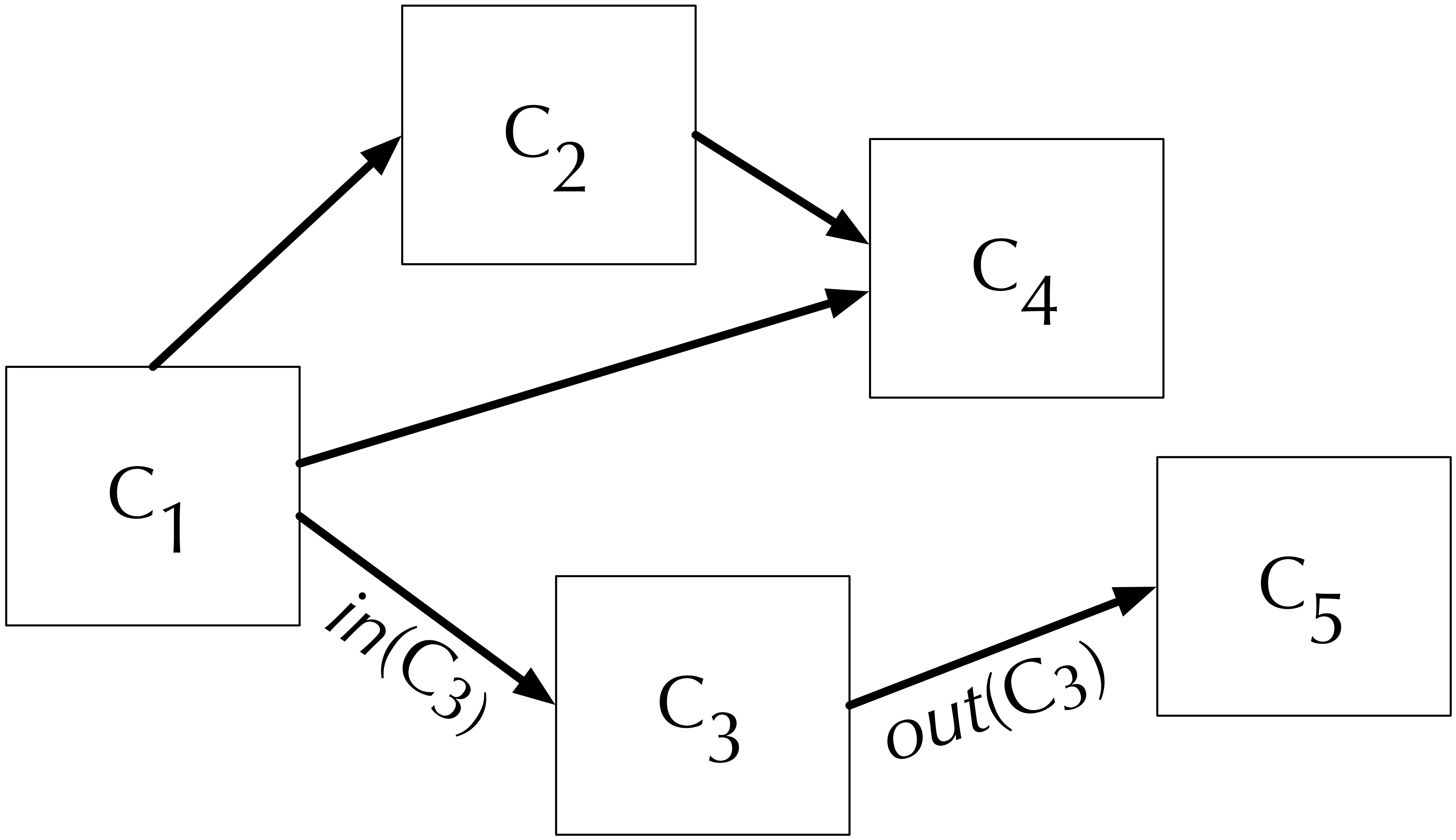
```
let $\_ =$ put $l\ 3$ in
  let par $v =$ get $l$
          $\_ =$ put $l\ 3$
    in $v$
```

```
let par $\_ =$ put $l\ (4, \bot)$
        $\_ =$ put $l\ (\bot, 3)$
  in let $v =$ get $l$ in $v$
```

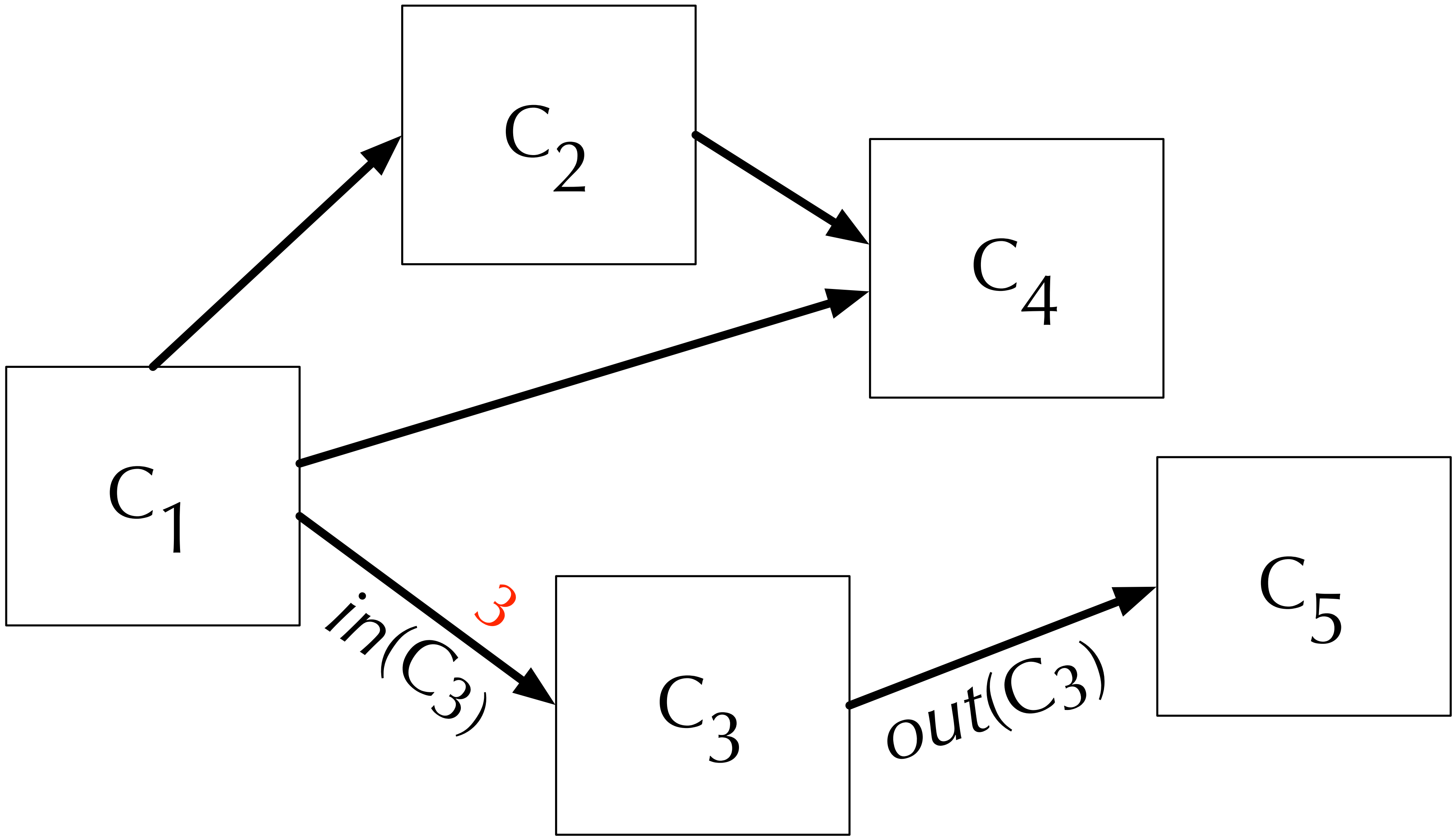
# Kahn process networks (Kahn, 1974)



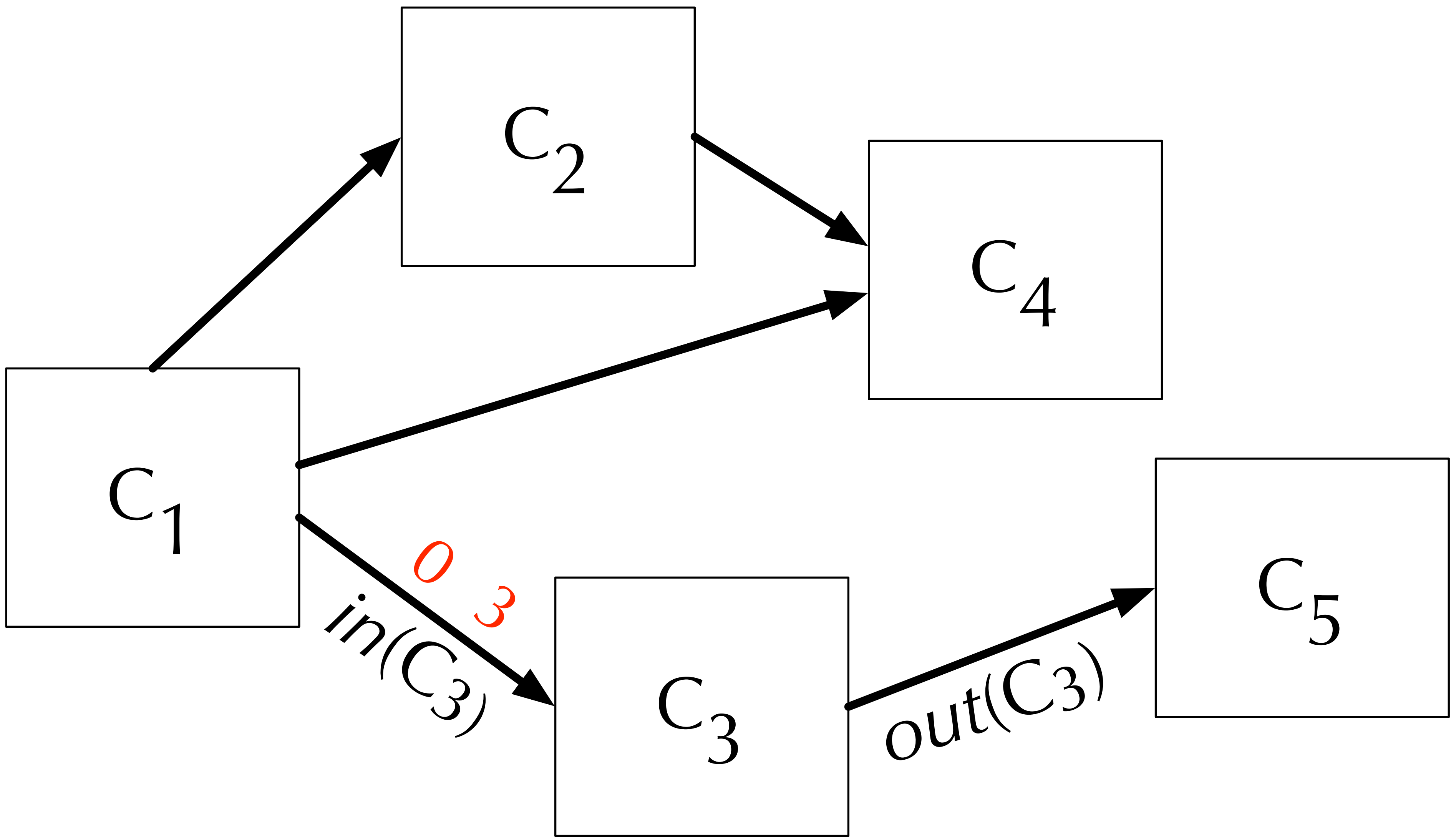
# Kahn process networks (Kahn, 1974)



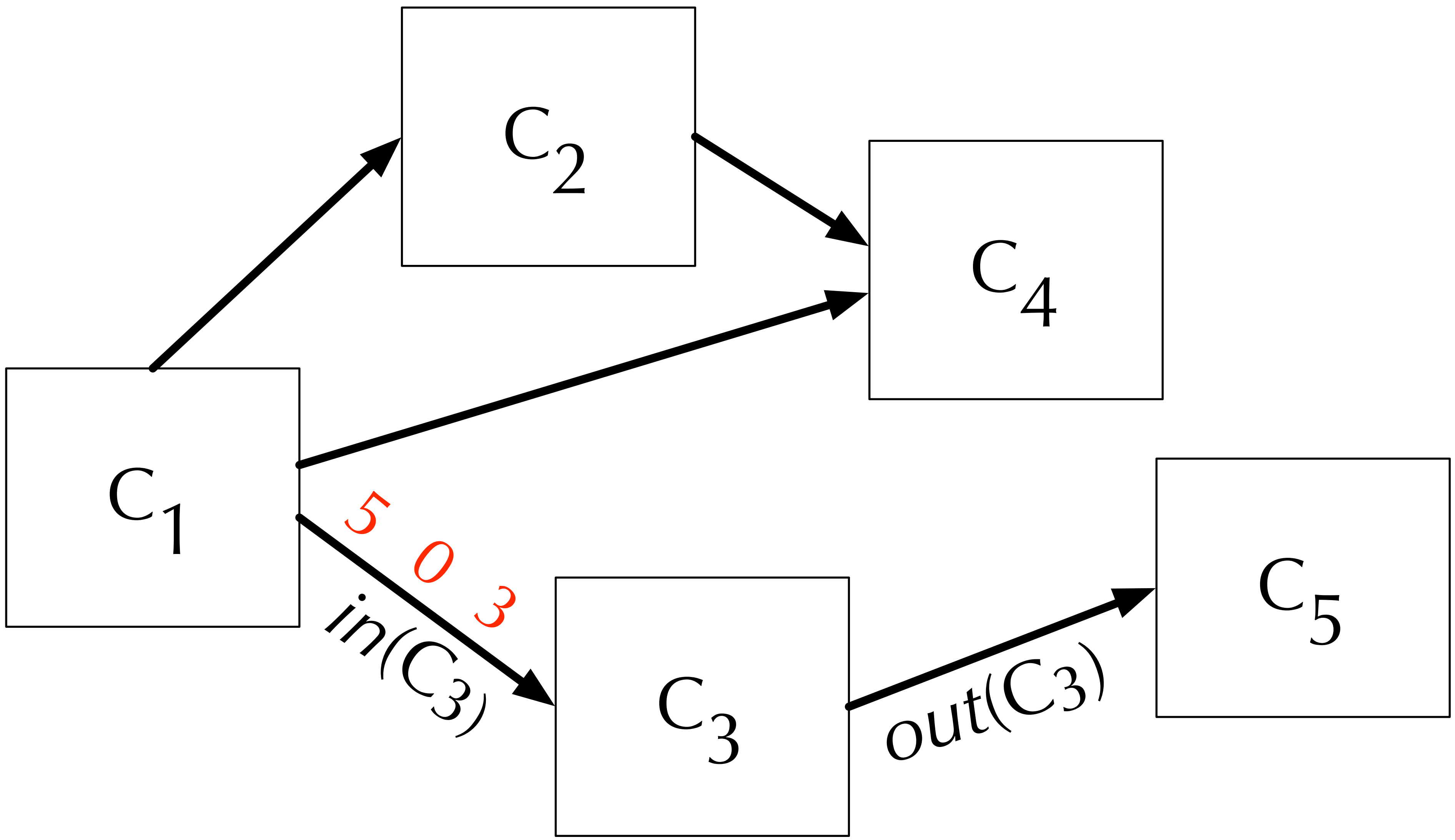
# Kahn process networks (Kahn, 1974)



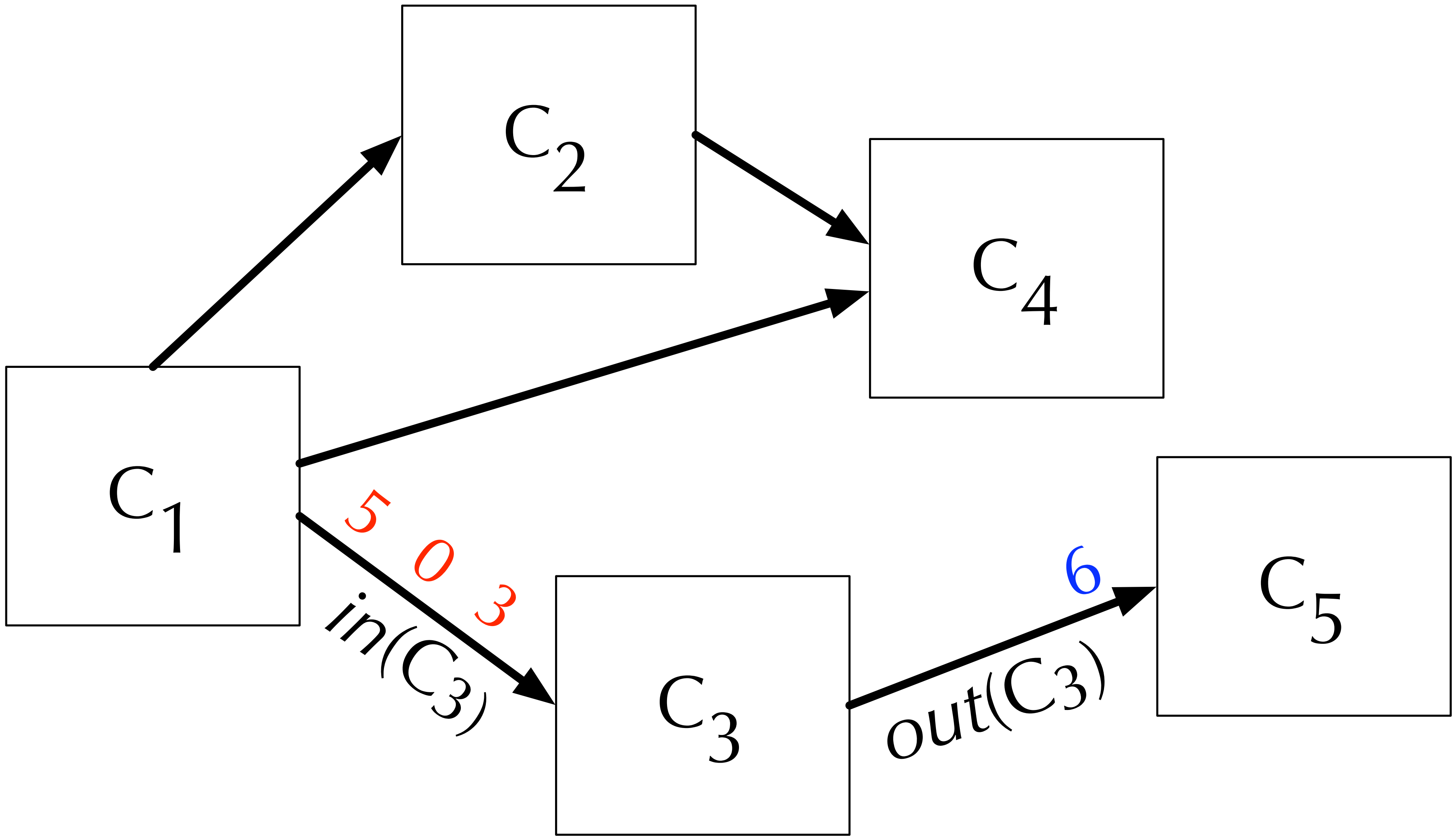
# Kahn process networks (Kahn, 1974)



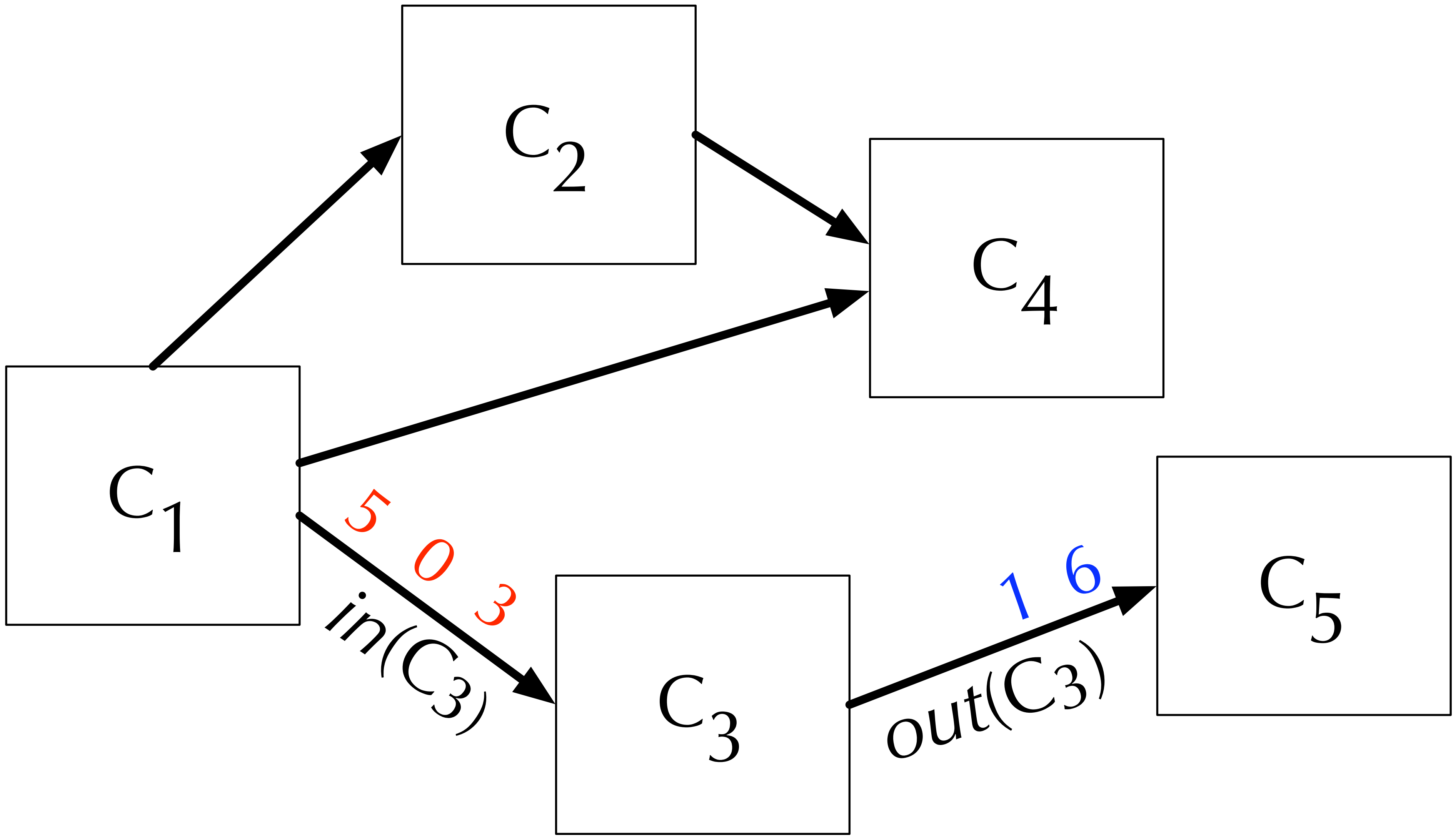
# Kahn process networks (Kahn, 1974)



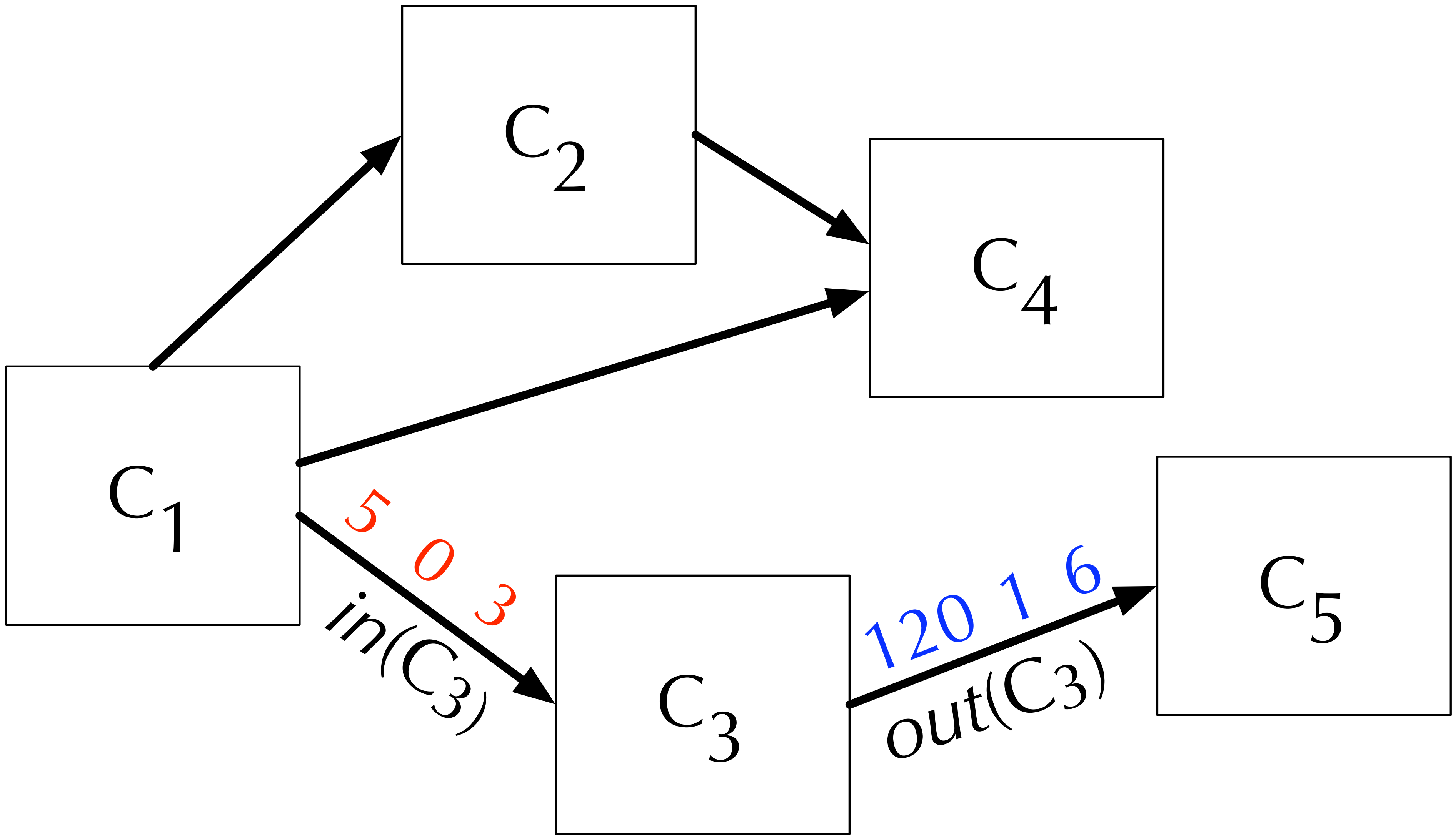
# Kahn process networks (Kahn, 1974)



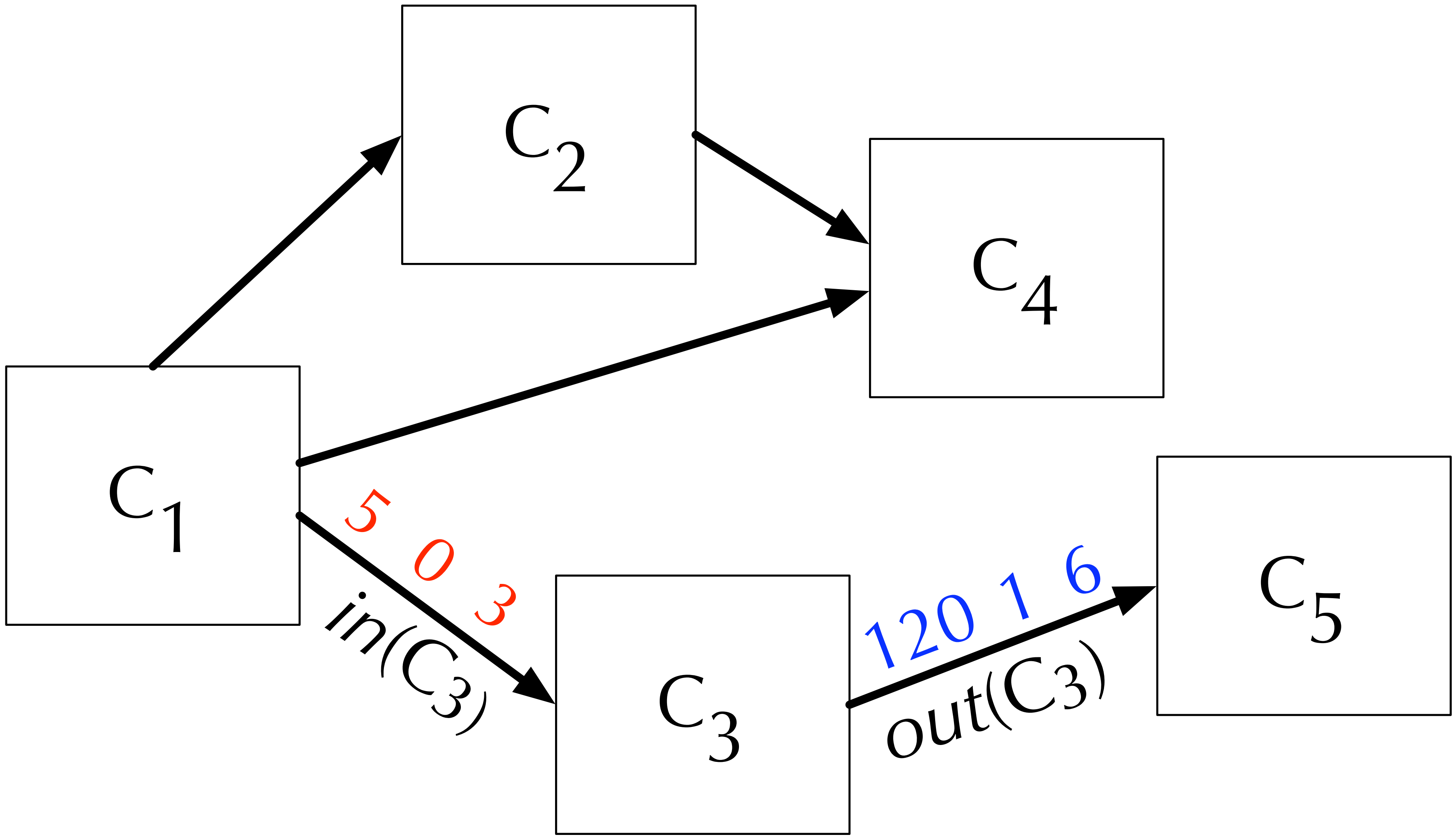
# Kahn process networks (Kahn, 1974)



# Kahn process networks (Kahn, 1974)

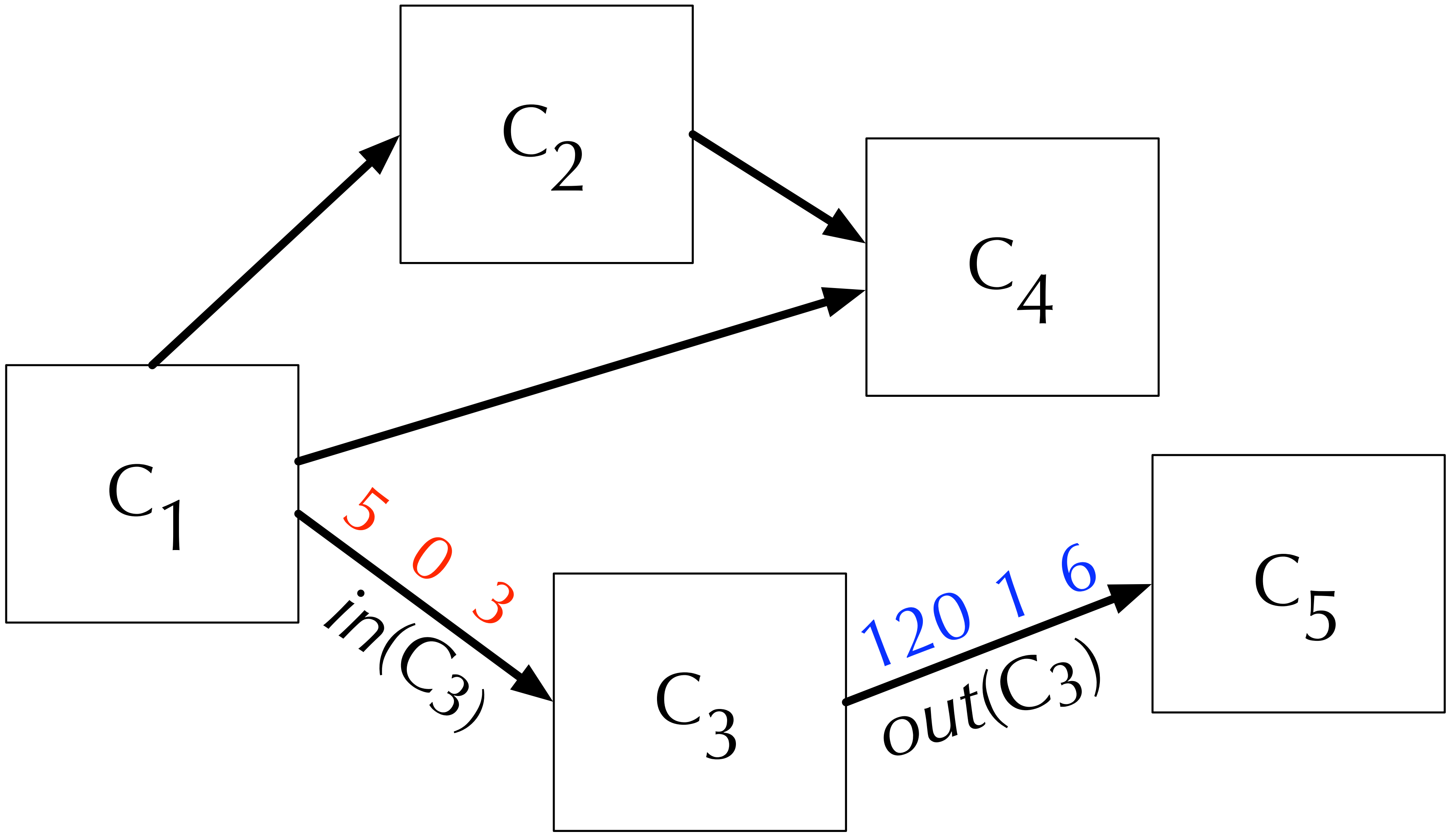


# Kahn process networks (Kahn, 1974)



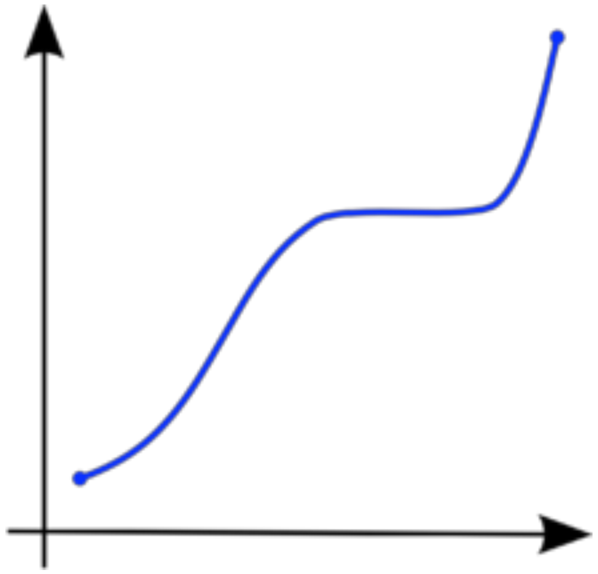
*hist*(*in*(C₃)): [3, 0, 5, ...]

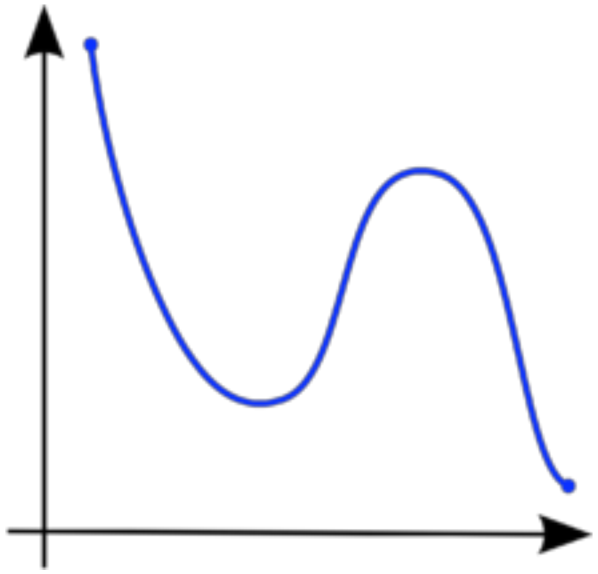
# Kahn process networks (Kahn, 1974)



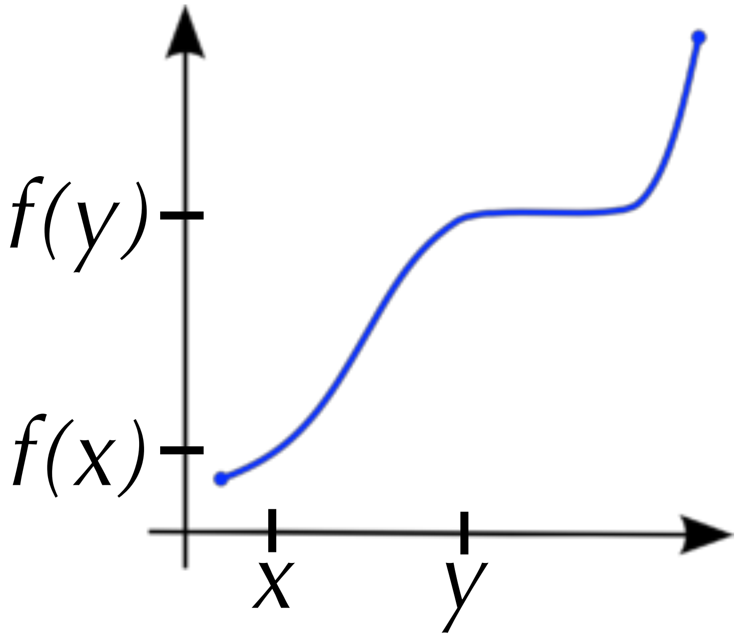
*hist*(*in*(C3)): [3, 0, 5, ...]  *hist*(*out*(C3)): [6, 1, 120, ...]

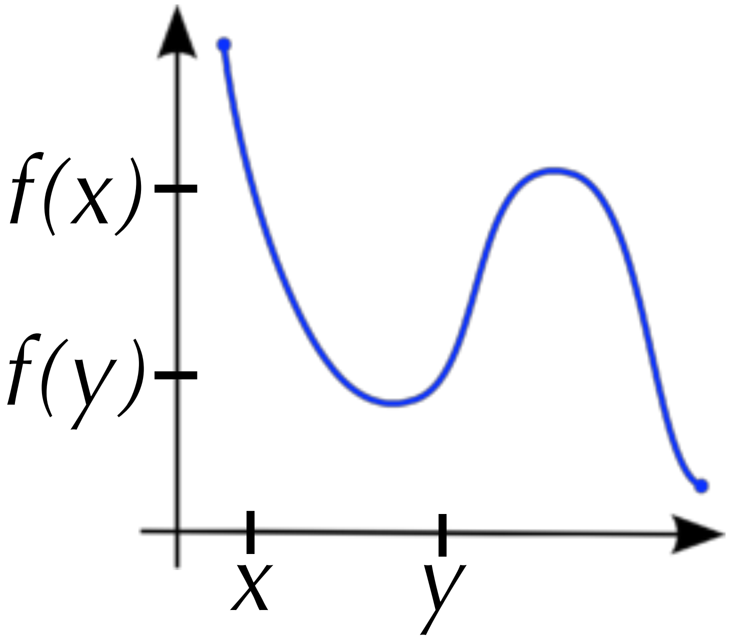
# Monotonicity



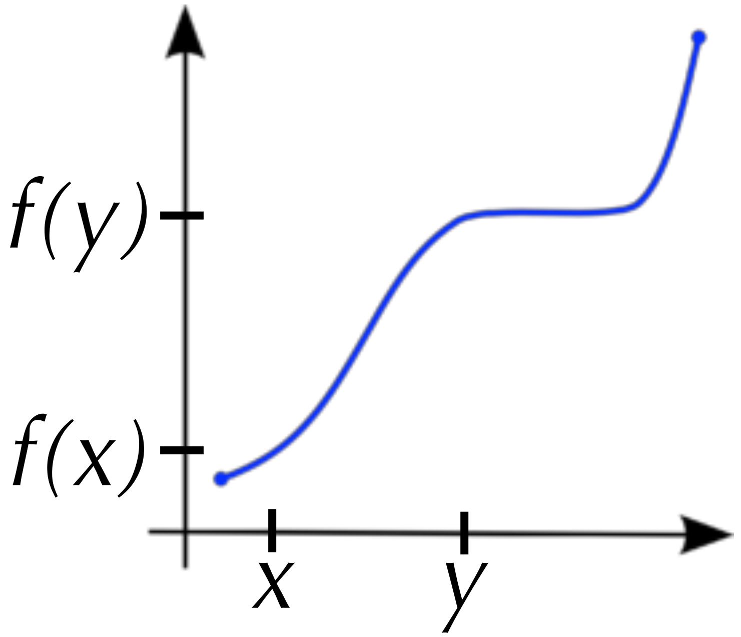


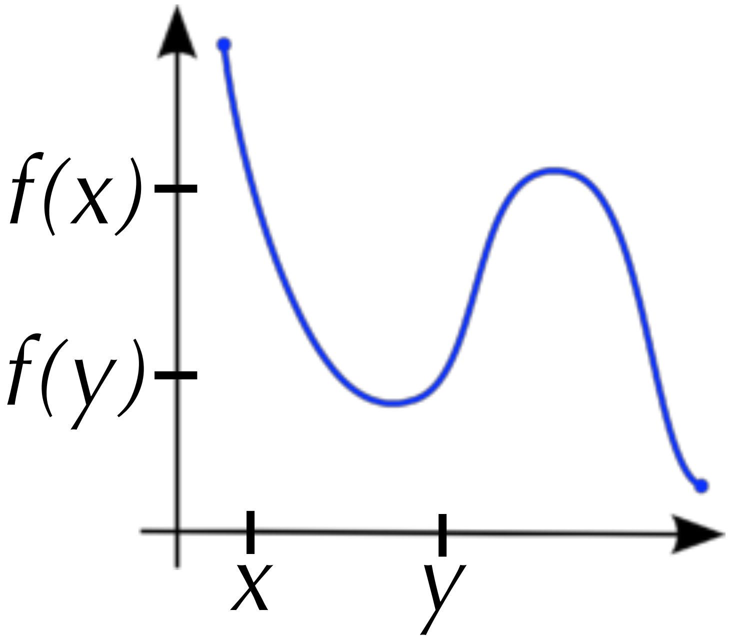
# Monotonicity





# Monotonicity

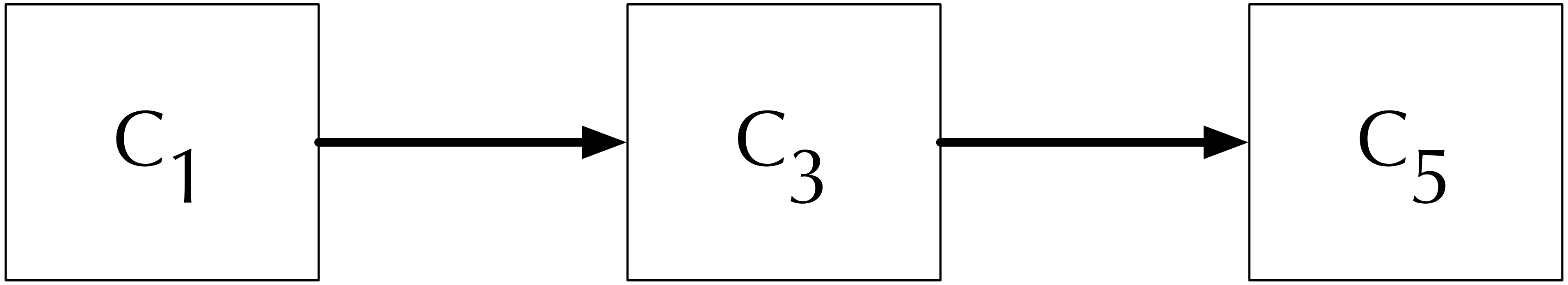




$f$ is monotonic iff $x \leq y \implies f(x) \leq f(y)$

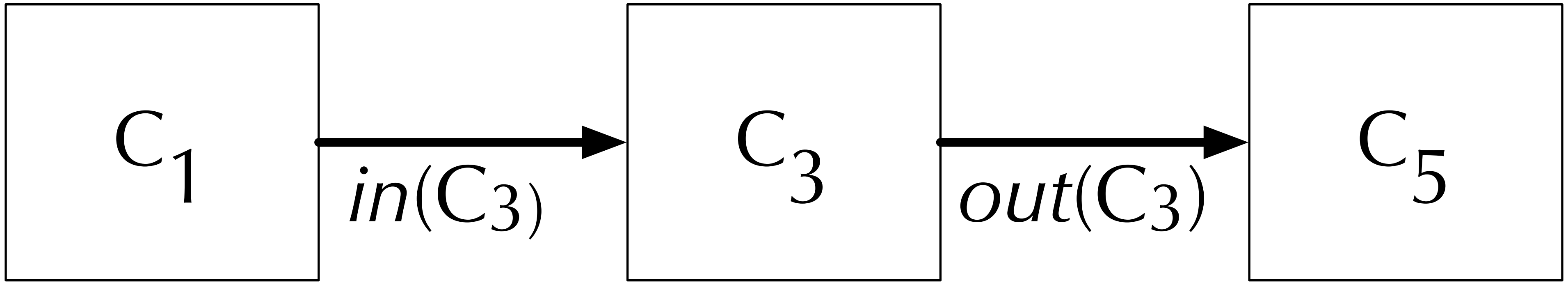
# Monotonicity in KPNs

$$f \text{ is monotonic iff } x \leq y \implies f(x) \leq f(y)$$



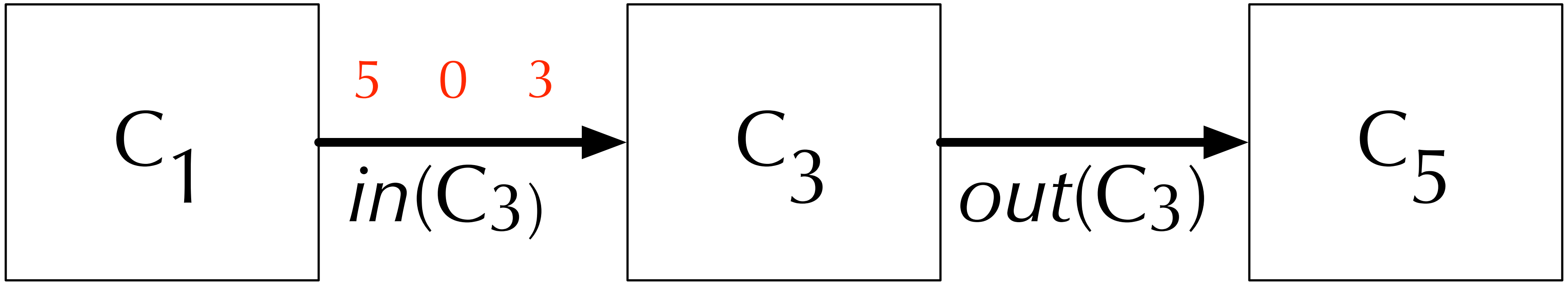
# Monotonicity in KPNs

$f$ is monotonic iff $x \leq y \implies f(x) \leq f(y)$



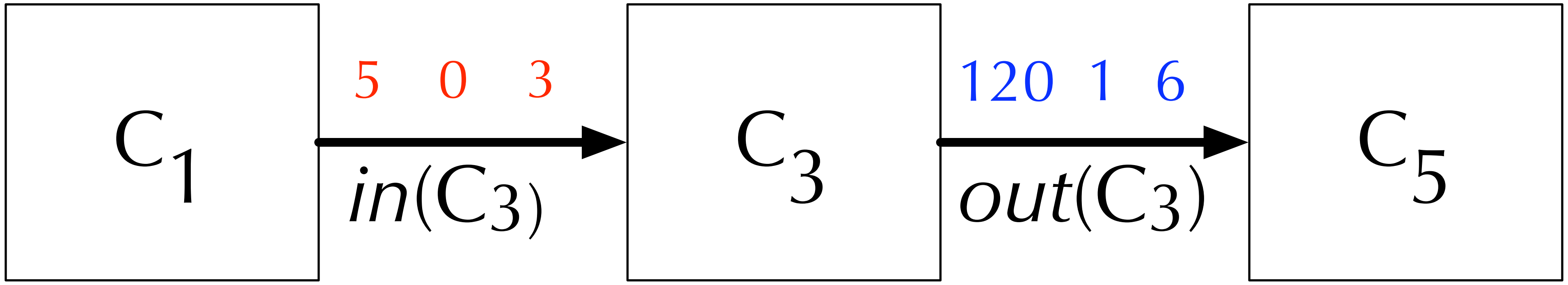
# Monotonicity in KPNs

$f$ is monotonic iff $x \leq y \implies f(x) \leq f(y)$



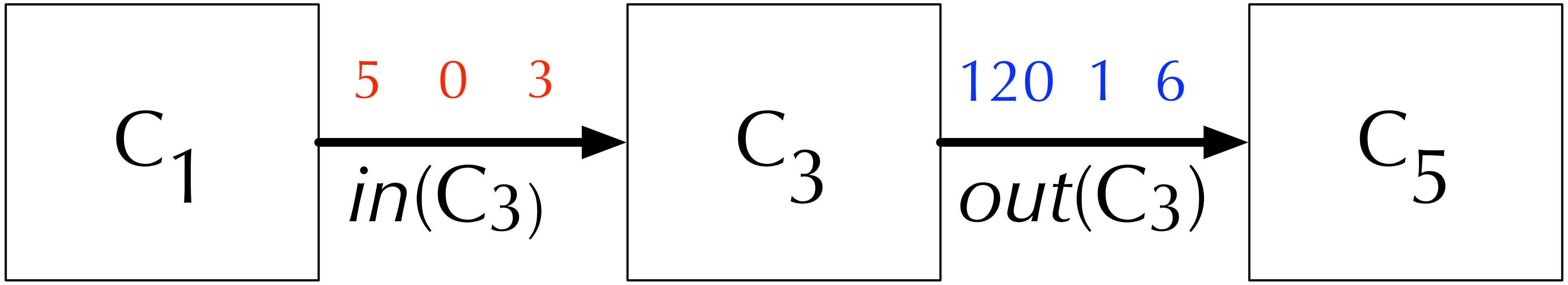
# Monotonicity in KPNs

$f$ is monotonic iff $x \leq y \implies f(x) \leq f(y)$



# Monotonicity in KPNs

$f$ is monotonic iff $x \leq y \implies f(x) \leq f(y)$



For KPNs, the ≤ relation is just *prefix-of*:

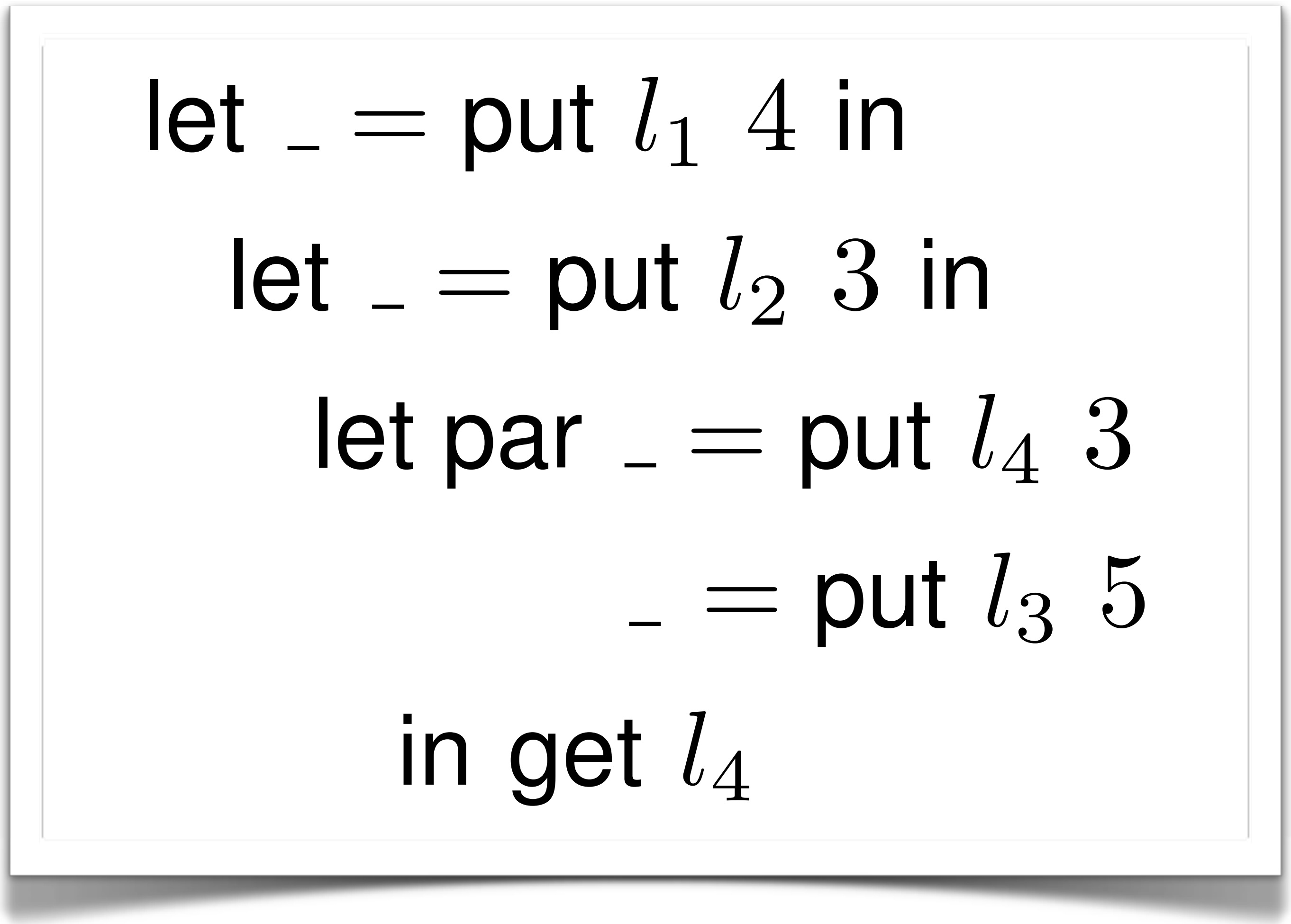
[3] *prefix-of* [3, 0] $\implies$ [6] *prefix-of* [6, 1]

[3, 0] *prefix-of* [3, 0, 5] $\implies$ [6, 1] *prefix-of* [6, 1, 120]

…

Monotonicity causes deterministic parallelism!

# Back to single-assignment languages



```
let _ = put $l_1$ $4$ in
   let _ = put $l_2$ $3$ in
      let par _ = put $l_4$ $3$
                 _ = put $l_3$ $5$
         in get $l_4$
```

Store:

# Back to single-assignment languages

```
let _ = put $l_1$ $4$ in
    let _ = put $l_2$ $3$ in
        let par _ = put $l_4$ $3$
                _ = put $l_3$ $5$
          in get $l_4$
```

Store:

| | |
|---|---|
| $l_1$ | $4$ |

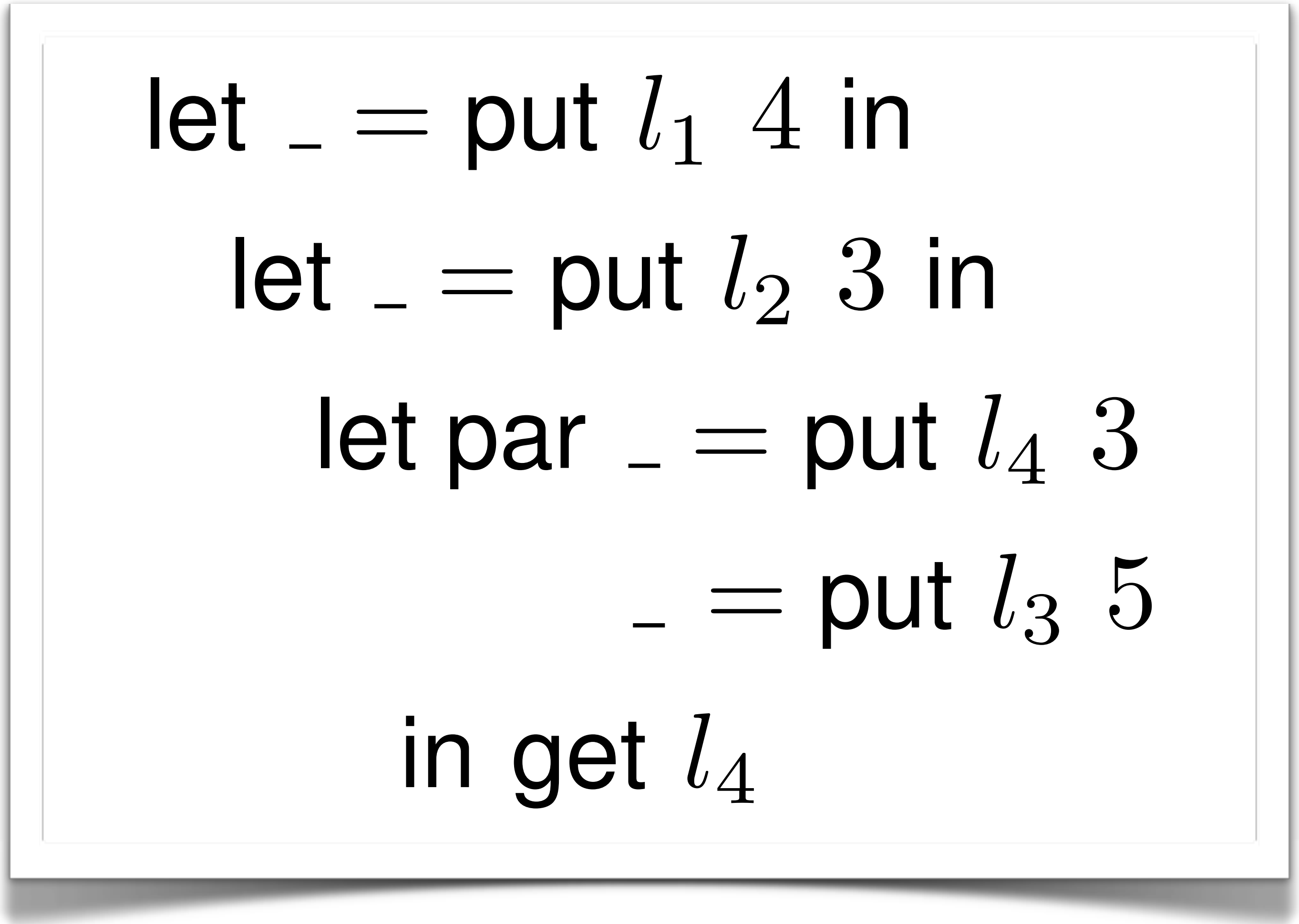
# Back to single-assignment languages

```
let _ = put $l_1$ $4$ in
  let _ = put $l_2$ $3$ in
    let par _ = put $l_4$ $3$
            _ = put $l_3$ $5$
      in get $l_4$
```

Store:

| | |
|---|---|
| $l_1$ | $4$ |
| $l_2$ | $3$ |

# Back to single-assignment languages



```
let _ = put $l_1$ $4$ in
  let _ = put $l_2$ $3$ in
    let par _ = put $l_4$ $3$
            _ = put $l_3$ $5$
      in get $l_4$
```

Store:

| | |
|---|---|
| $l_1$ | $4$ |
| $l_2$ | $3$ |
| $l_3$ | $5$ |

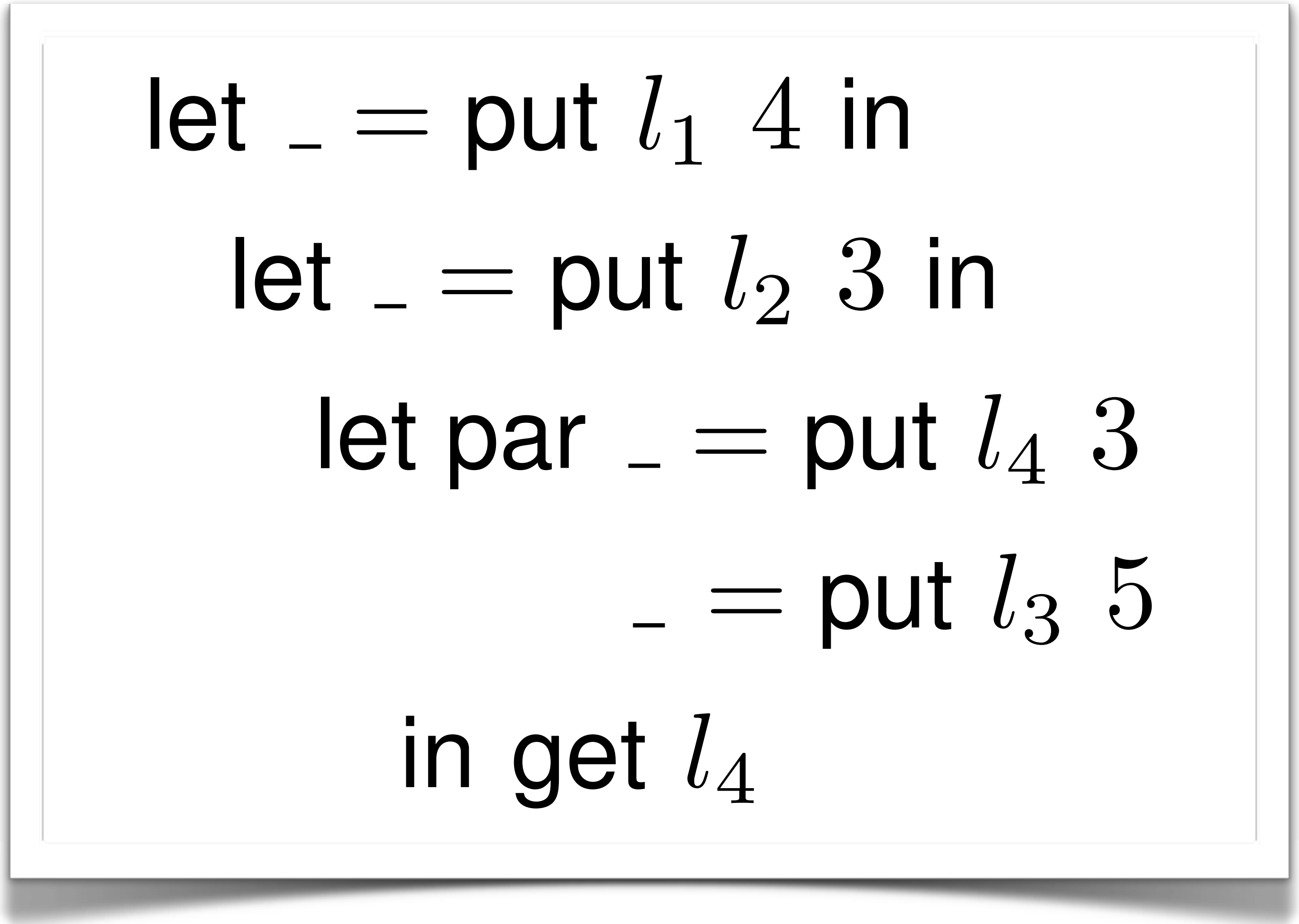
# Back to single-assignment languages

```
let _ = put $l_1$ 4 in
    let _ = put $l_2$ 3 in
        let par _ = put $l_4$ 3
                _ = put $l_3$ 5
            in get $l_4$
```

Store:

| | |
|---|---|
| $l_1$ | 4 |
| $l_2$ | 3 |
| $l_3$ | 5 |
| $l_4$ | 3 |

# Back to single-assignment languages



```
let _ = put $l_1$ $4$ in
  let _ = put $l_2$ $3$ in
    let par _ = put $l_4$ $3$
            _ = put $l_3$ $5$
      in get $l_4$
```

Store:

| | |
|---|---|
| $l_1$ | $4$ |
| $l_2$ | $3$ |
| $l_3$ | $5$ |
| $l_4$ | $3$ |

For stores, the ≤ relation is ⊆:

$$\{l_1 \rightarrow 4,\ l_2 \rightarrow 3\} \subseteq \{l_1 \rightarrow 4,\ l_2 \rightarrow 3,\ l_3 \rightarrow 5\} \Longrightarrow$$

$$\{l_1 \rightarrow 4,\ l_2 \rightarrow 3,\ l_4 \rightarrow 3\} \subseteq \{l_1 \rightarrow 4,\ l_2 \rightarrow 3,\ l_3 \rightarrow 5,\ l_4 \rightarrow 3\}$$

# Generalizing our notion of monotonicity

For stores, the $\leq$ relation is $\subseteq$:

$$\{l_1 \rightarrow 4,\ l_2 \rightarrow 3\} \subseteq \{l_1 \rightarrow 4,\ l_2 \rightarrow 3,\ l_3 \rightarrow 5\} \Longrightarrow$$

$$\{l_1 \rightarrow 4,\ l_2 \rightarrow 3,\ l_4 \rightarrow 3\} \subseteq \{l_1 \rightarrow 4,\ l_2 \rightarrow 3,\ l_3 \rightarrow 5,\ l_4 \rightarrow 3\}$$

- Given stores $S$ and $S'$, we say that $S \leq S'$ iff:
  - dom($S$) $\subseteq$ dom($S'$), and
  - for all locations $l$ in dom($S$), $S(l) = S'(l)$

# Generalizing our notion of monotonicity

For stores, the $\leq$ relation is $\subseteq$:

$$\{l_1 \to 4,\ l_2 \to 3\} \subseteq \{l_1 \to 4,\ l_2 \to 3,\ l_3 \to 5\} \implies$$

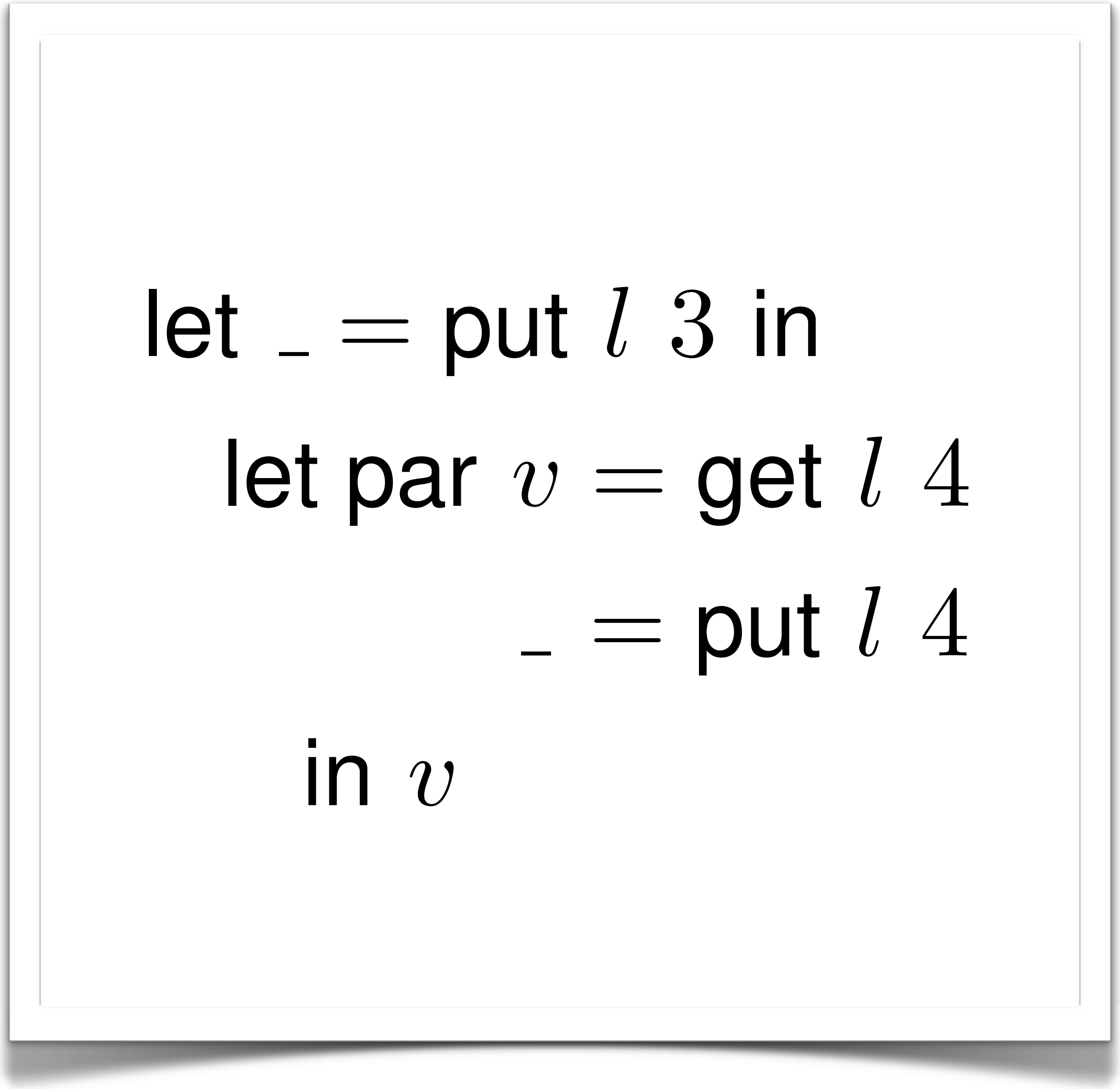
$$\{l_1 \to 4,\ l_2 \to 3,\ l_4 \to 3\} \subseteq \{l_1 \to 4,\ l_2 \to 3,\ l_3 \to 5,\ l_4 \to 3\}$$

- Given stores $S$ and $S'$, we say that $S \leq S'$ iff:
  - dom($S$) $\subseteq$ dom($S'$), and
  - for all locations $l$ in dom($S$), $S(l) \cancel{=} S'(l)$

$\leq$

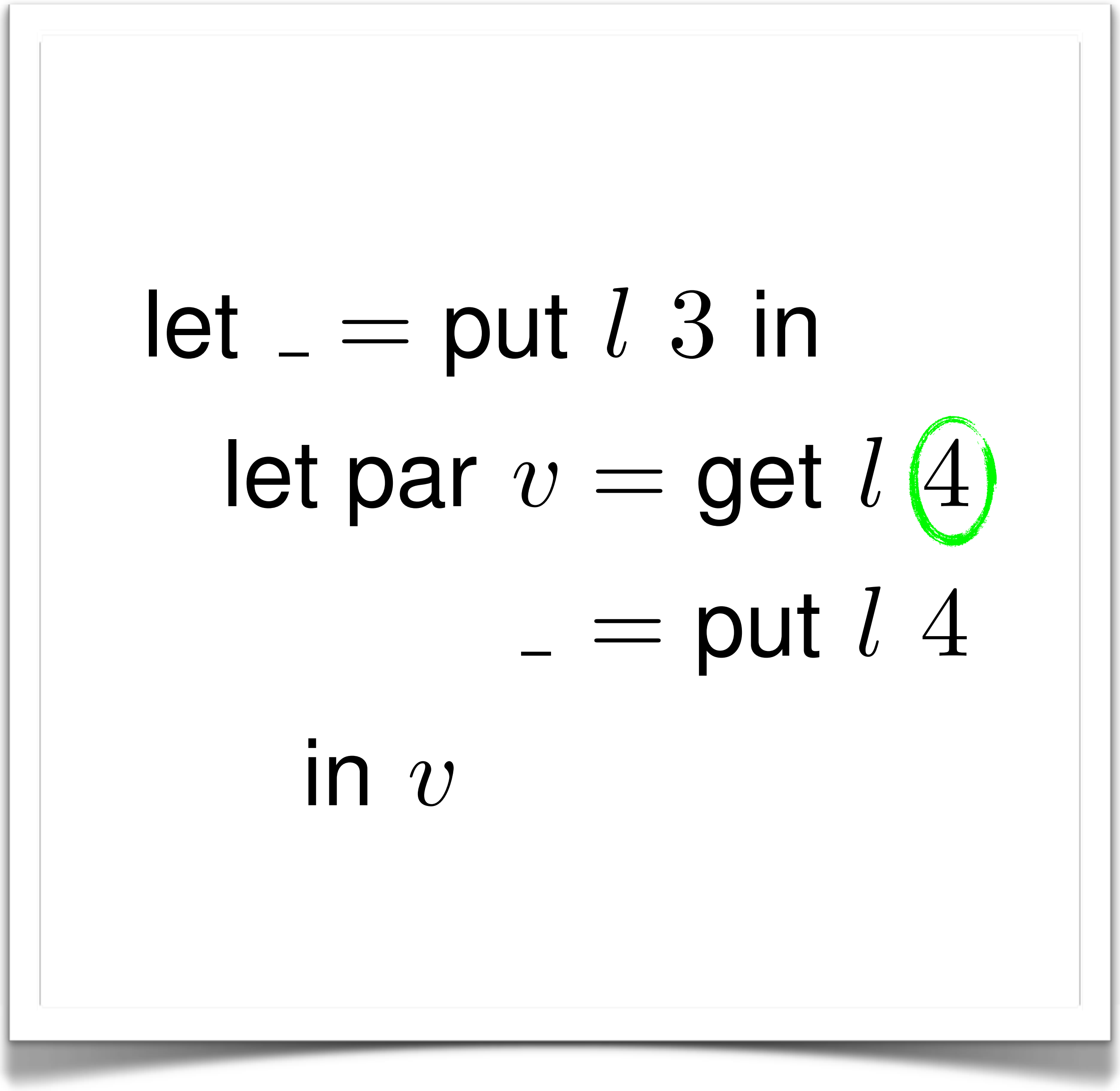
user-specified

# Idea: restrict reads to “threshold” reads



```
let _ = put l 3 in
  let par v = get l 4
          _ = put l 4
    in v
```

# Idea: restrict reads to “threshold” reads



```
let _ = put $l$ $3$ in
   let par $v$ = get $l$ $\textcircled{4}$
             _ = put $l$ $4$
      in $v$
```

# Idea: restrict reads to “threshold” reads

```
let $\_ =$ put $l$ $3$ in
  let par $v =$ get $l$ $\boxed{4}$
          $\_ =$ put $l$ $4$
  in $v$
```

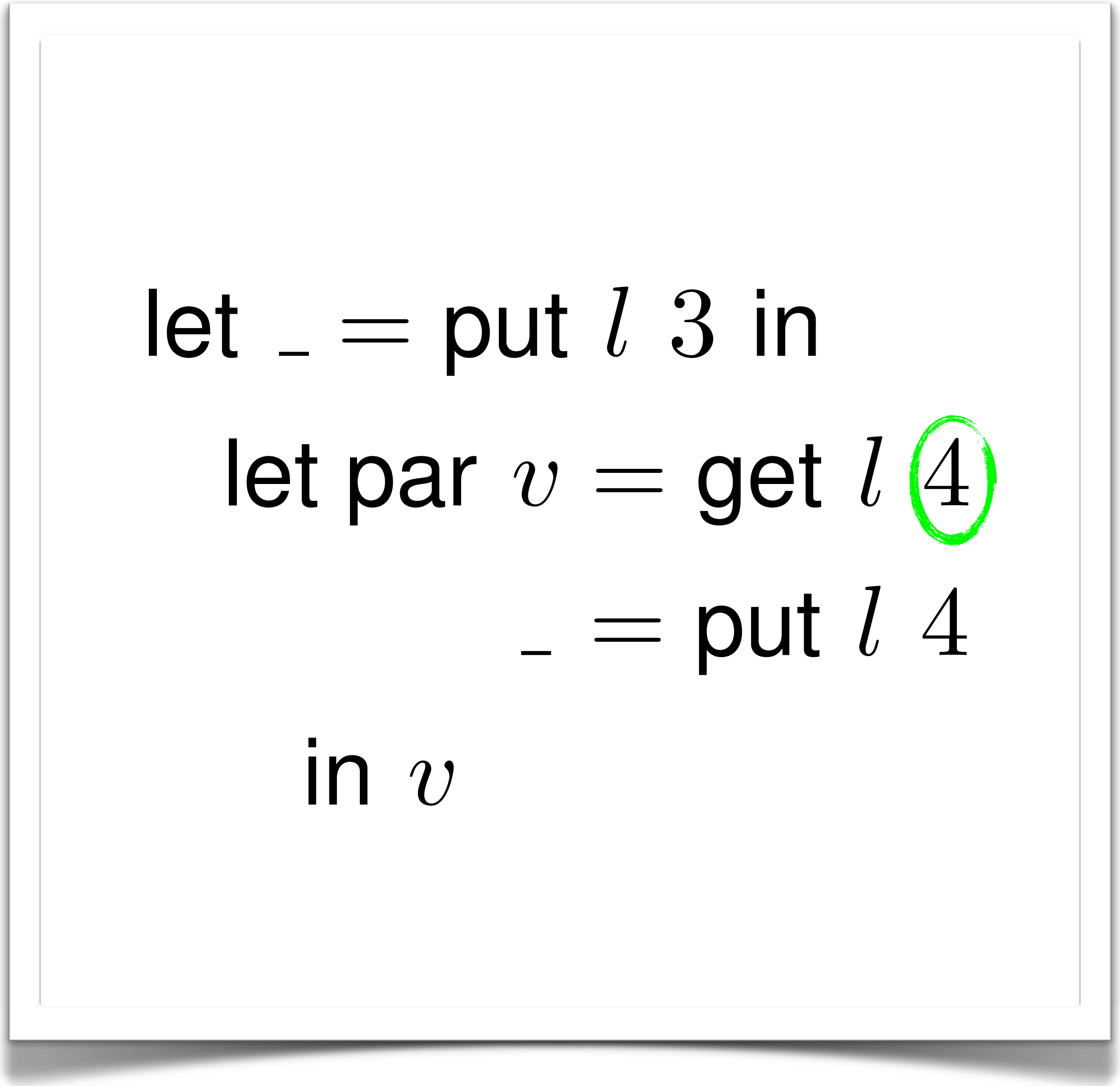
```
let $\_ =$ put $l$ $3$ in
  let par $v =$ get $l$ $4$
          $\_ =$ put $l$ $4$
          $\_ =$ put $l$ $5$
  in $v$
```

# Idea: restrict reads to “threshold” reads

```
let _ = put l 3 in
  let par v = get l 4
          _ = put l 4
    in v
```

```
let _ = put l 3 in
  let par v = get l 4
          _ = put l 4
          _ = put l 5
    in v
```

# Idea: restrict reads to “threshold” reads

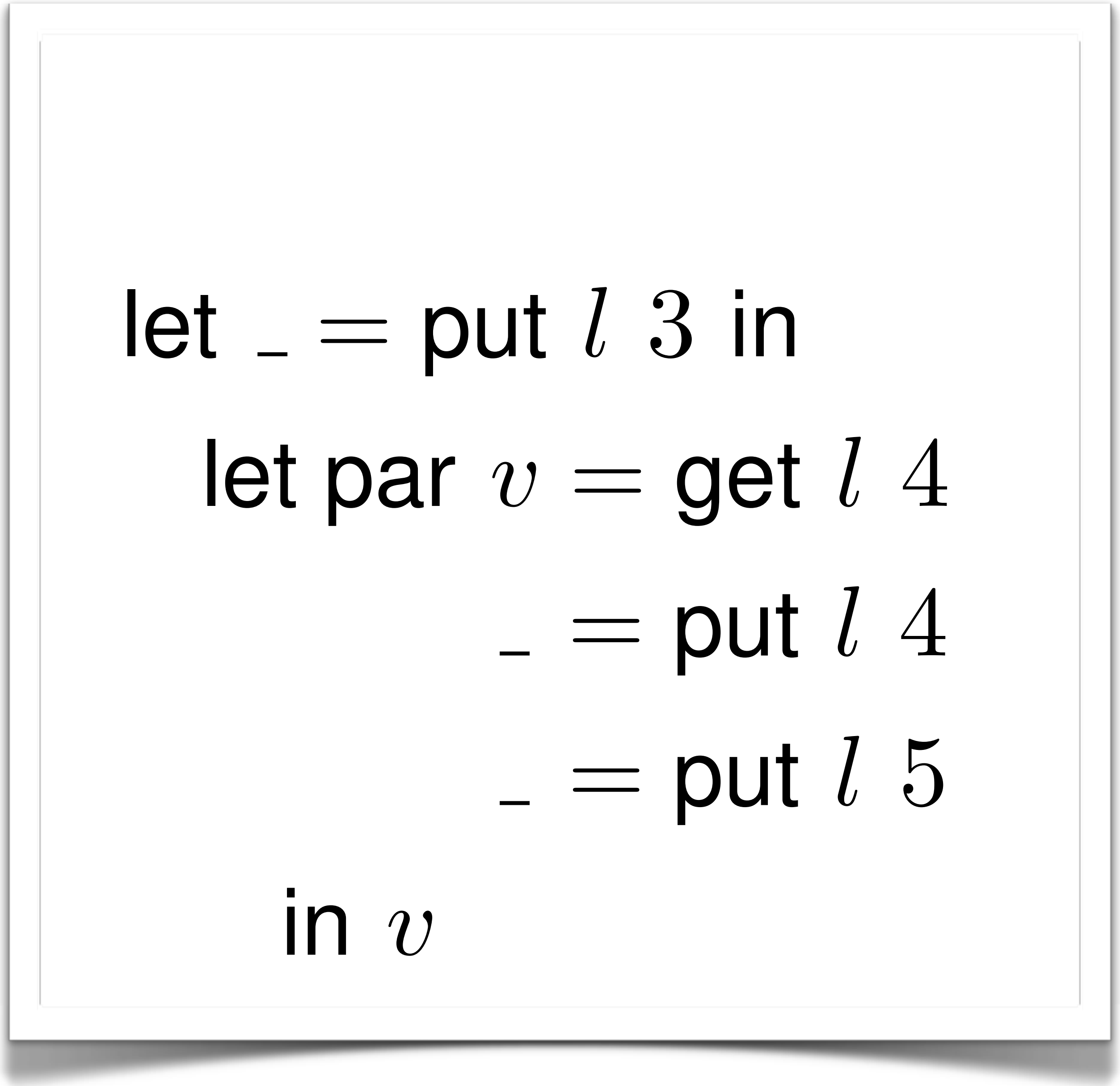


let $\_ =$ put $l\ 3$ in

let par $v =$ get $l\ \textcircled{4}$

$\_ =$ put $l\ 4$

in $v$



let $\_ =$ put $l\ 3$ in

let par $v =$ get $l\ 4$

$\_ =$ put $l\ 4$

$\_ =$ put $l\ 5$

in $v$

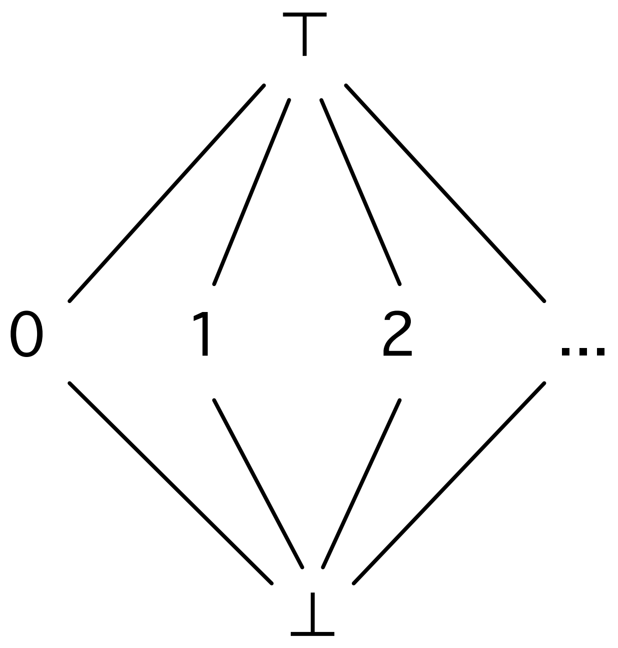
# Idea: restrict reads to “threshold” reads

$$
\begin{array}{l}
\text{let}\ \_ = \text{put}\ l\ 3\ \text{in} \\
\quad \text{let par}\ v = \text{get}\ l\ \boxed{4} \\
\quad \phantom{\text{let par}\ v} \_ = \text{put}\ l\ 4 \\
\quad \text{in}\ v
\end{array}
$$

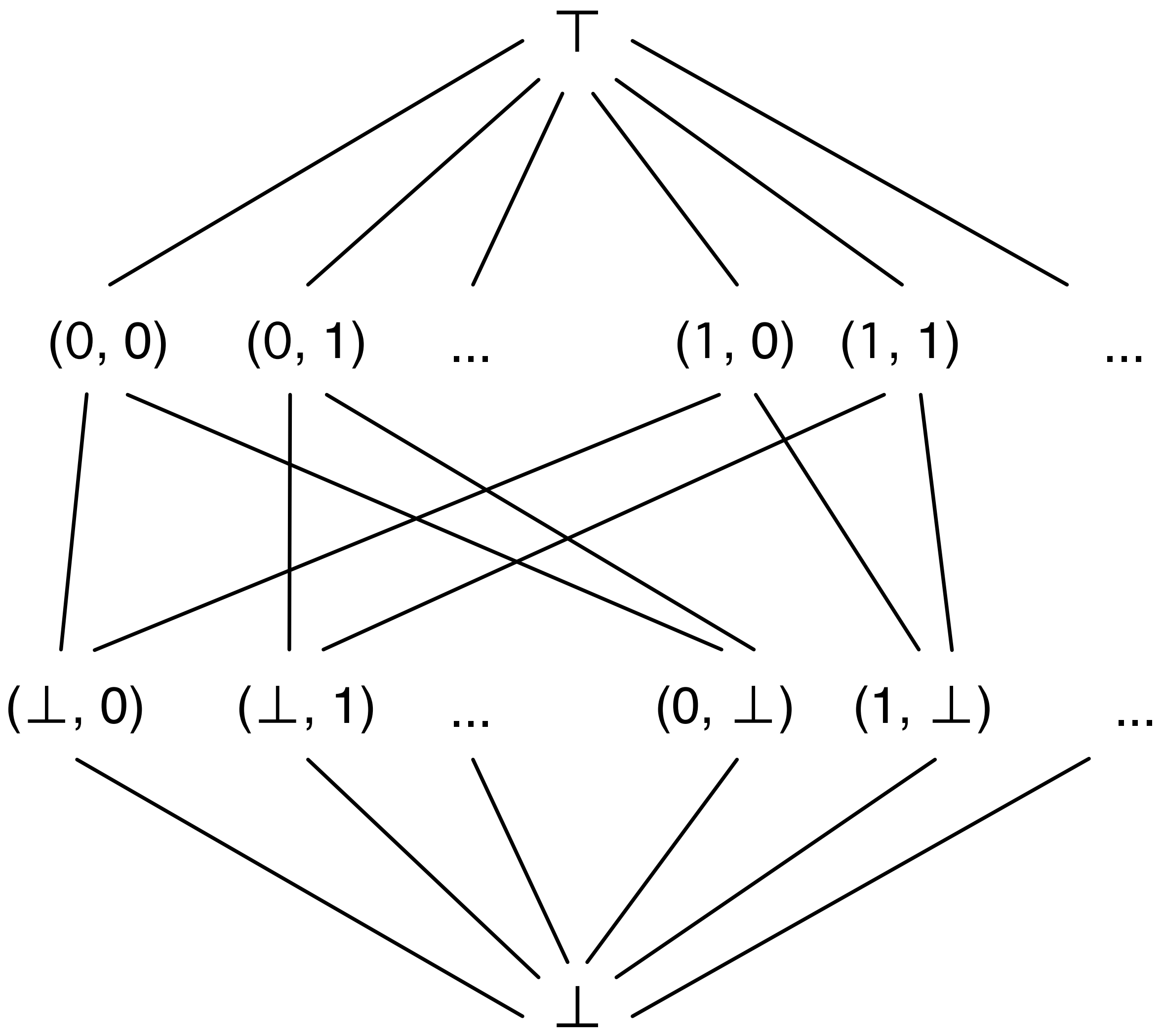
$$
\begin{array}{l}
\text{let}\ \_ = \text{put}\ l\ 3\ \text{in} \\
\quad \text{let par}\ v = \overset{\text{return 4}}{\boxed{\text{get}\ l\ 4}} \\
\quad \phantom{\text{let par}\ v} \_ = \text{put}\ l\ 4 \\
\quad \phantom{\text{let par}\ v} \_ = \text{put}\ l\ 5 \\
\quad \text{in}\ v
\end{array}
$$

Monotonically increasing writes
+ threshold reads
= deterministic parallelism

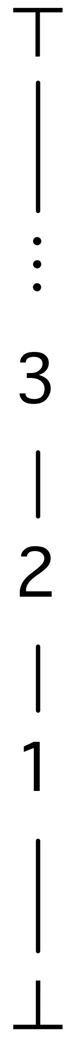
# Parameterizing our language: “LVars”



IVar

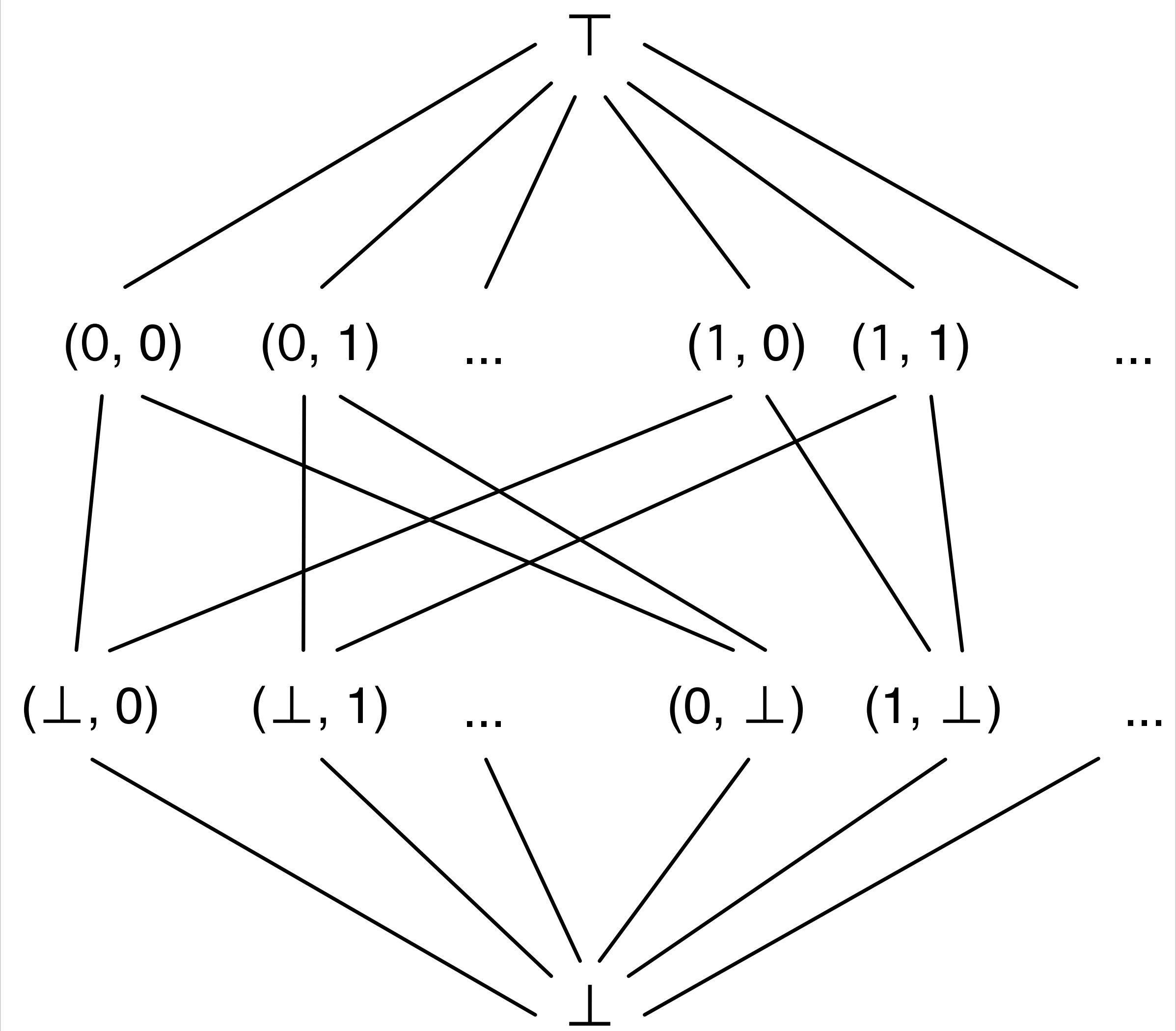


Pair of IVars



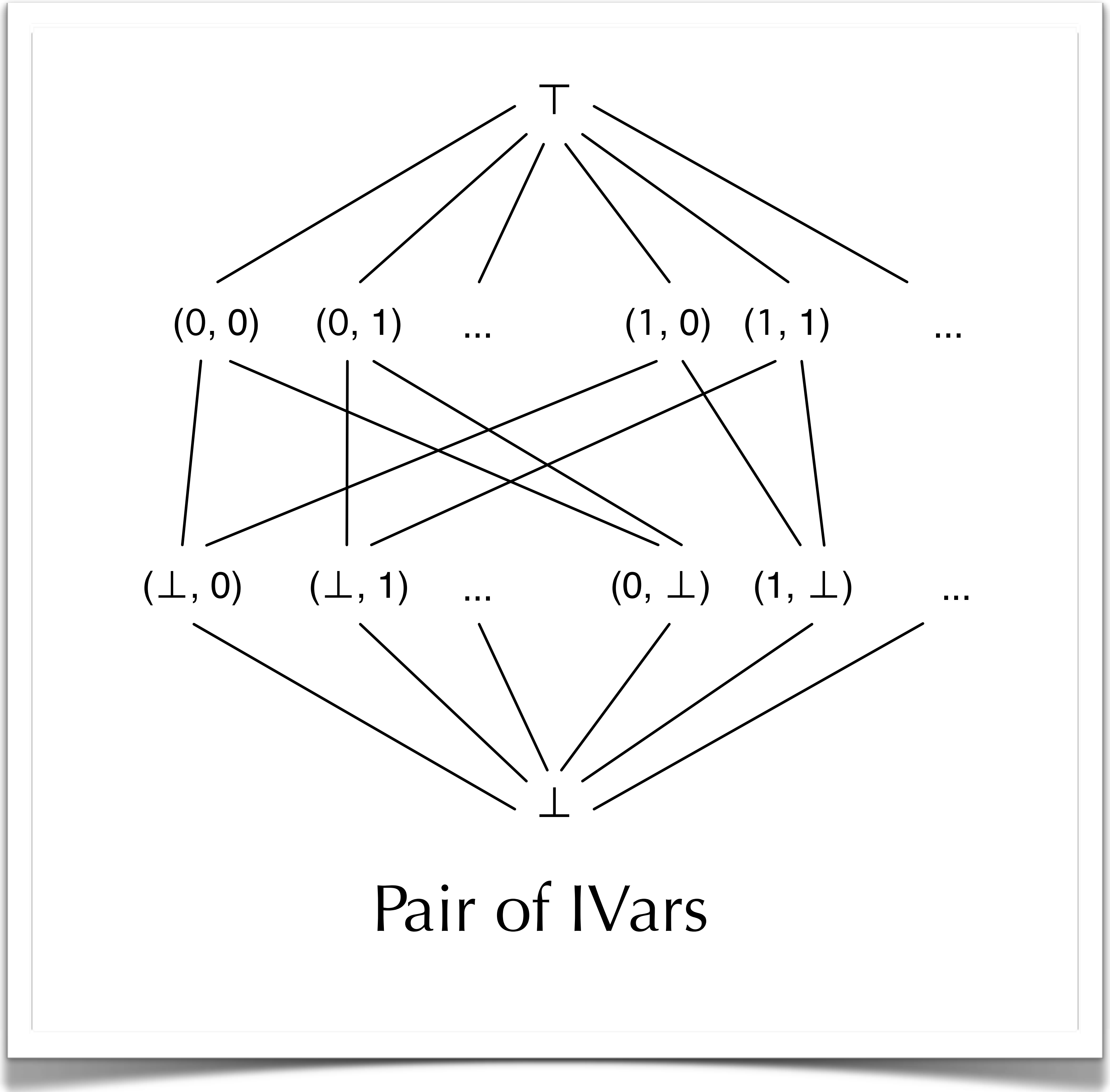
Counter

# Parameterizing our language: “LVars”



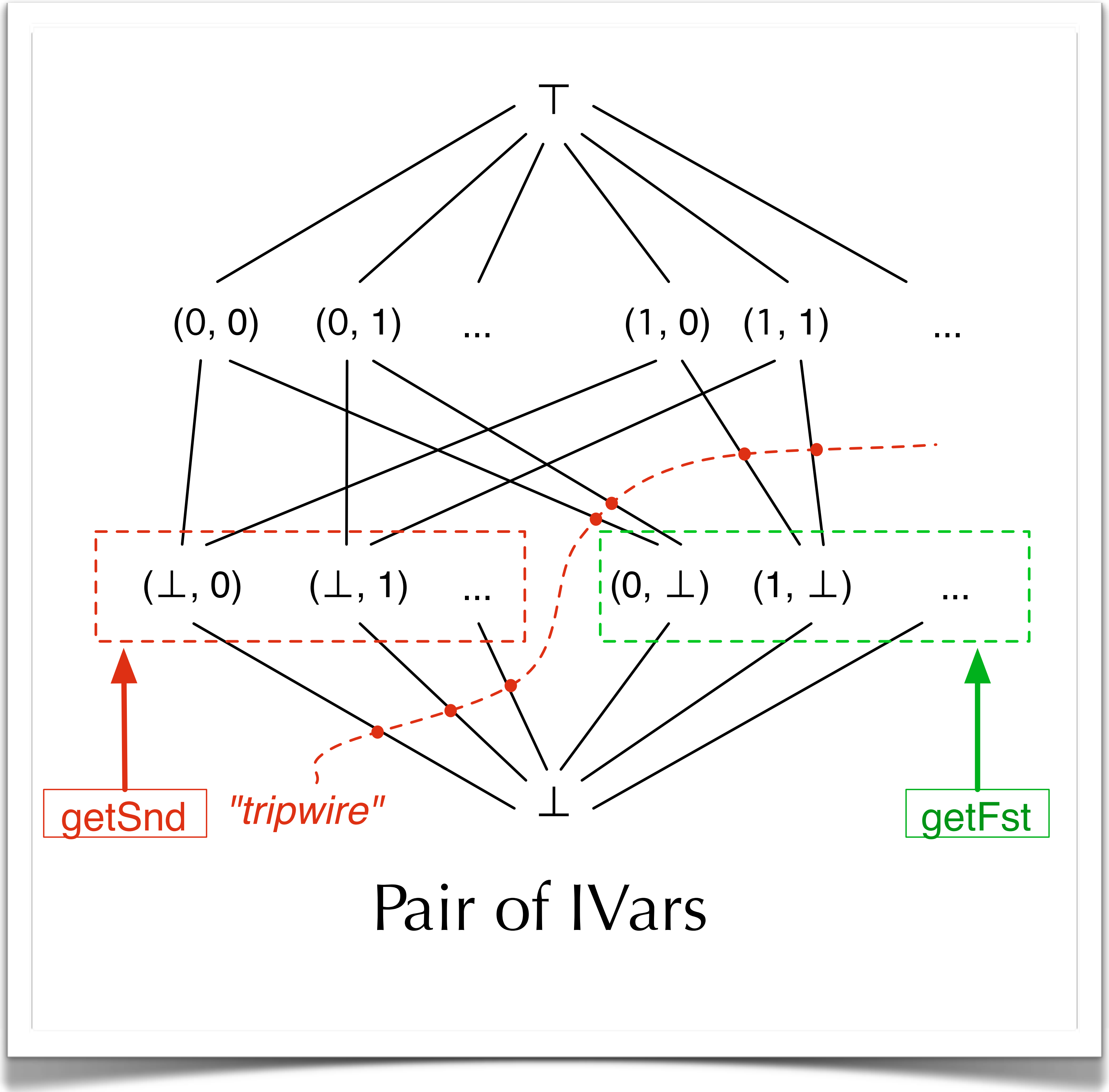
Pair of IVars

# Parameterizing our language: "LVars"



Pair of IVars

# Parameterizing our language: “LVars”



# Parameterizing our language: “LVars”

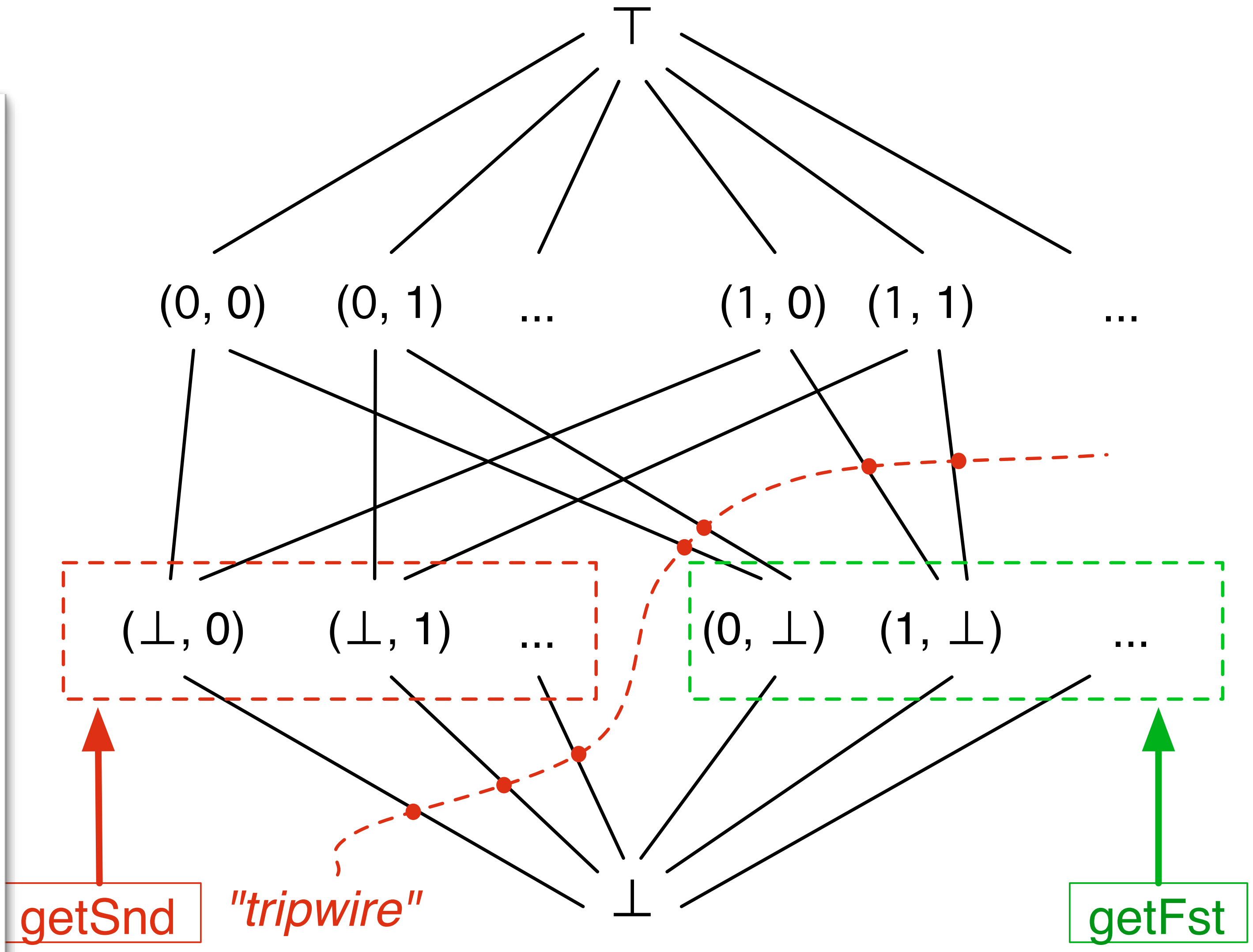
let $p$ = new in

  let $\_$ = put $p$ $\{(\bot, 4)\}$ in

    let par $v_1$ = getFst $p$

          $\_$ = put $p$ $\{(3, 4)\}$

     in $\ldots v_1 \ldots$



Pair of IVars

# Two take-aways

Monotonicity causes deterministic parallelism

Monotonically increasing writes
+ threshold reads
= deterministic parallelism

# More in our paper draft

# More in our paper draft

- Complete syntax and semantics

# More in our paper draft

- Complete syntax and semantics
- Proof of determinism
  - A “frame-rule-like” property
  - Location renaming is surprisingly tricky!

# More in our paper draft

- Complete syntax and semantics
- Proof of determinism
  - A "frame-rule-like" property
  - Location renaming is surprisingly tricky!
- Subsuming existing models
  - KPNs, CnC, monad-par

# More in our paper draft

- Complete syntax and semantics
- Proof of determinism
  - A “frame-rule-like” property
  - Location renaming is surprisingly tricky!
- Subsuming existing models
  - KPNs, CnC, monad-par
- Support for controlled nondeterminism
  - “probation” state

# Tak!

**Email:**
lkuper@cs.indiana.edu
**Twitter:** @lindsey
**Web:** cs.indiana.edu/~lkuper
**Research group:**
lambda.cs.indiana.edu

Photo by kakadu on Flickr.  Thanks!