A Lattice-Based Approach to Deterministic Parallelism with Shared State

Lindsey Kuper and Ryan R. Newton
Indiana University

University of California, Berkeley
August 16, 2012

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

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, ...

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

Serialize?

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

Serialize?

let $\_ =$ put l 3 in

let ~~par~~ $v =$ get l in

let $\_ =$ put l 4

in v

Serialize?

$$
\begin{array}{l}
\mathsf{let}\ \_ = \mathsf{put}\ l\ 3\ \mathsf{in} \\
\quad \mathsf{let\ par}\ v = \mathsf{get}\ l \\
\qquad\qquad\quad \_ = \mathsf{put}\ l\ 4 \\
\qquad \mathsf{in}\ v
\end{array}
$$

Disallow shared state?

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

Disallow shared state?

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

Disallow shared state?

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

Disallow multiple assignment?

$$
\begin{array}{l}
\mathsf{let}\ \_ = \mathsf{put}\ l\ 3\ \mathsf{in} \\
\quad \mathsf{let\ par}\ v = \mathsf{get}\ l \\
\qquad\qquad\quad\ \_ = \mathsf{put}\ l\ 4 \\
\quad\ \ \mathsf{in}\ v
\end{array}
$$

Disallow multiple assignment?

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

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?

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

Deterministic programs that single-assignment forbids

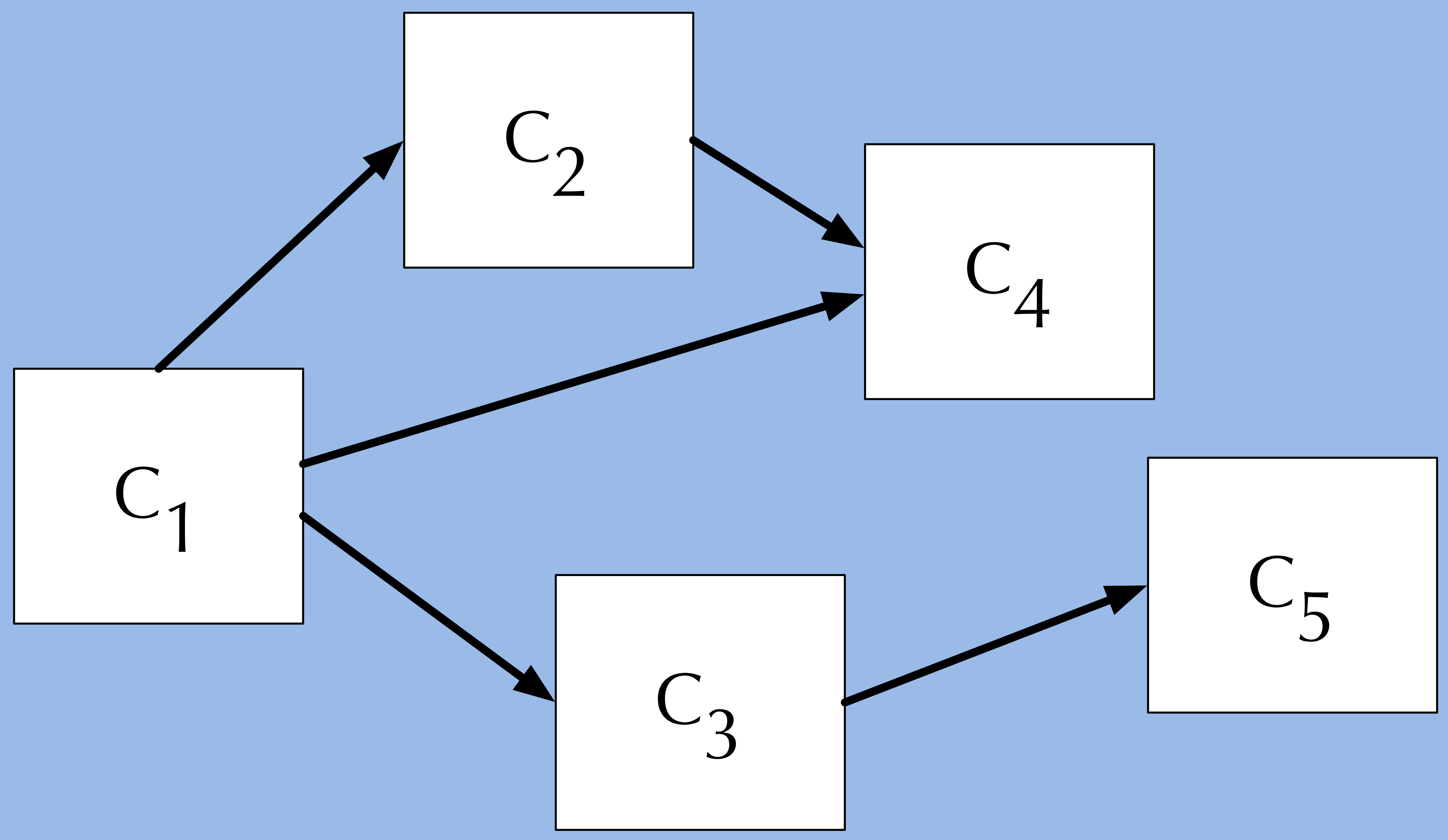
$$
\begin{array}{l}
\text{let } \_ = \text{put } l\ 3 \text{ in} \\
\quad \text{let par } v = \text{get } l \\
\qquad\qquad\quad \_ = \text{put } l\ 3 \\
\quad\quad \text{in } v
\end{array}
$$

Deterministic programs that single-assignment forbids

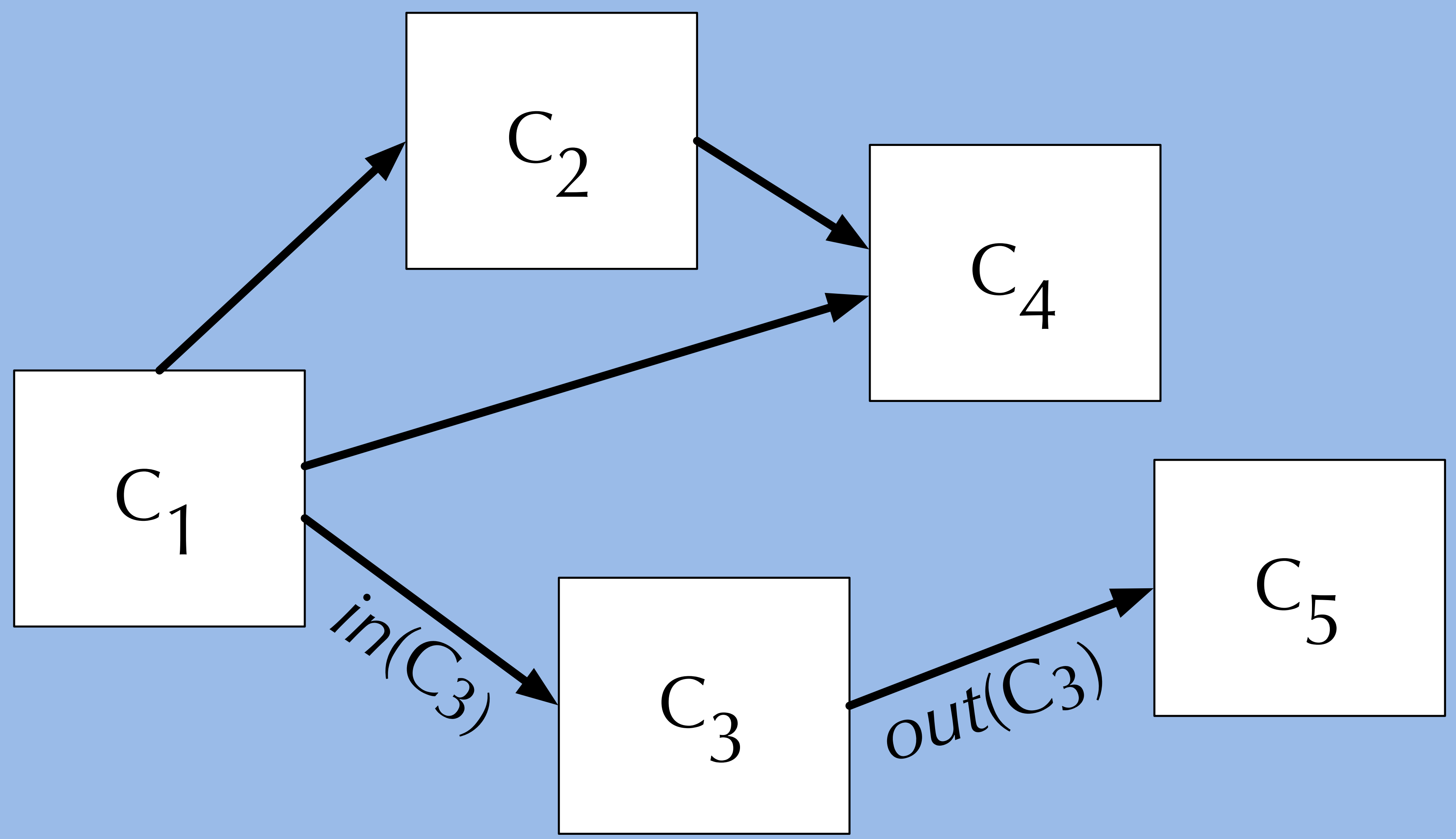
$$
\begin{array}{l}
\text{let } \_ = \text{put } l\ 3 \text{ in} \\
\quad \text{let par } v = \text{get } l \\
\qquad\qquad\quad \_ = \text{put } l\ 3 \\
\quad\quad \text{in } v
\end{array}
$$

$$
\begin{array}{l}
\text{let par } \_ = \text{put } l\ (4, \bot) \\
\qquad\quad\ \ \_ = \text{put } l\ (\bot, 3) \\
\quad \text{in let } v = \text{get } l \text{ in } v
\end{array}
$$

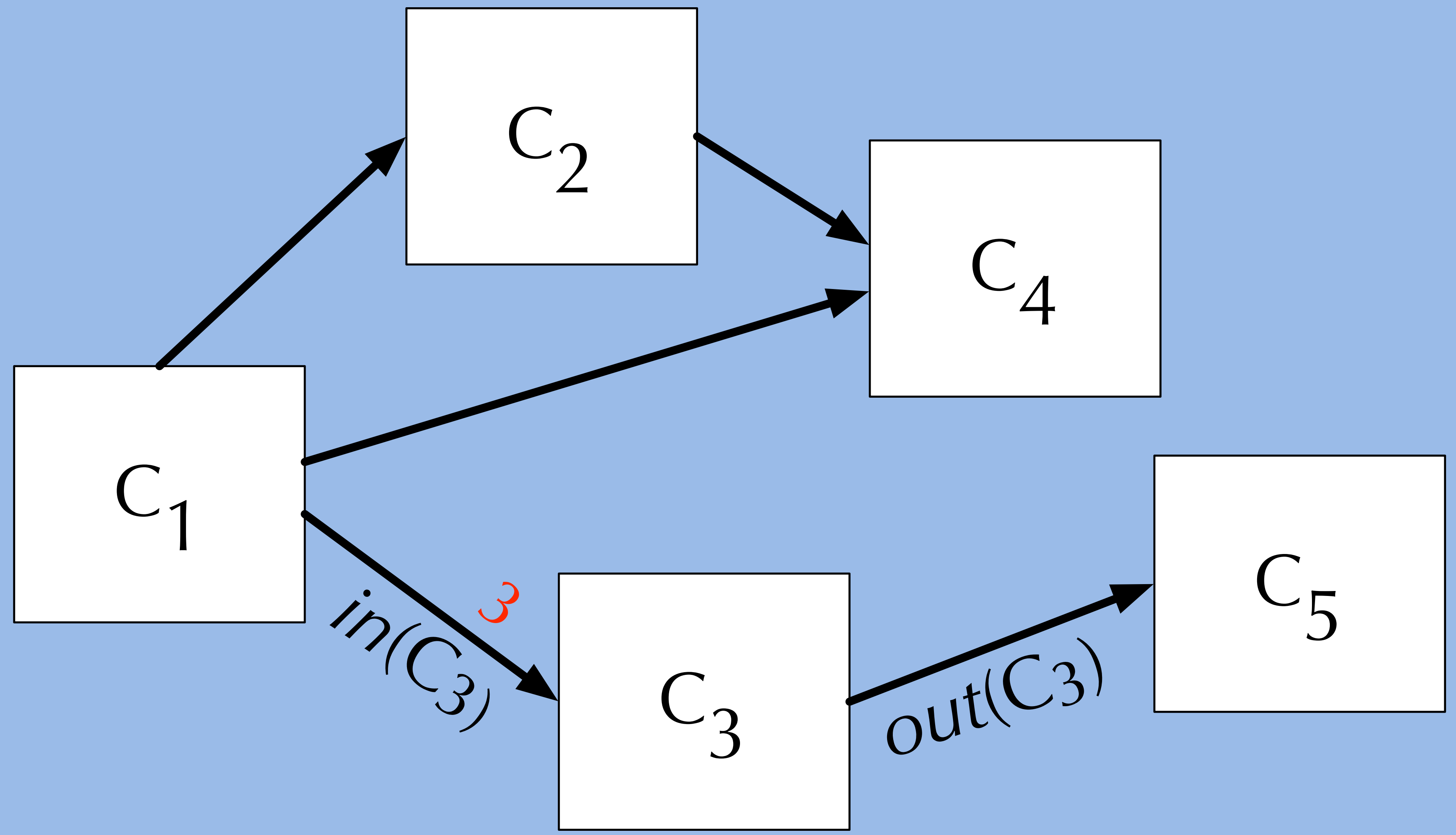
Kahn process networks (Kahn, 1974)



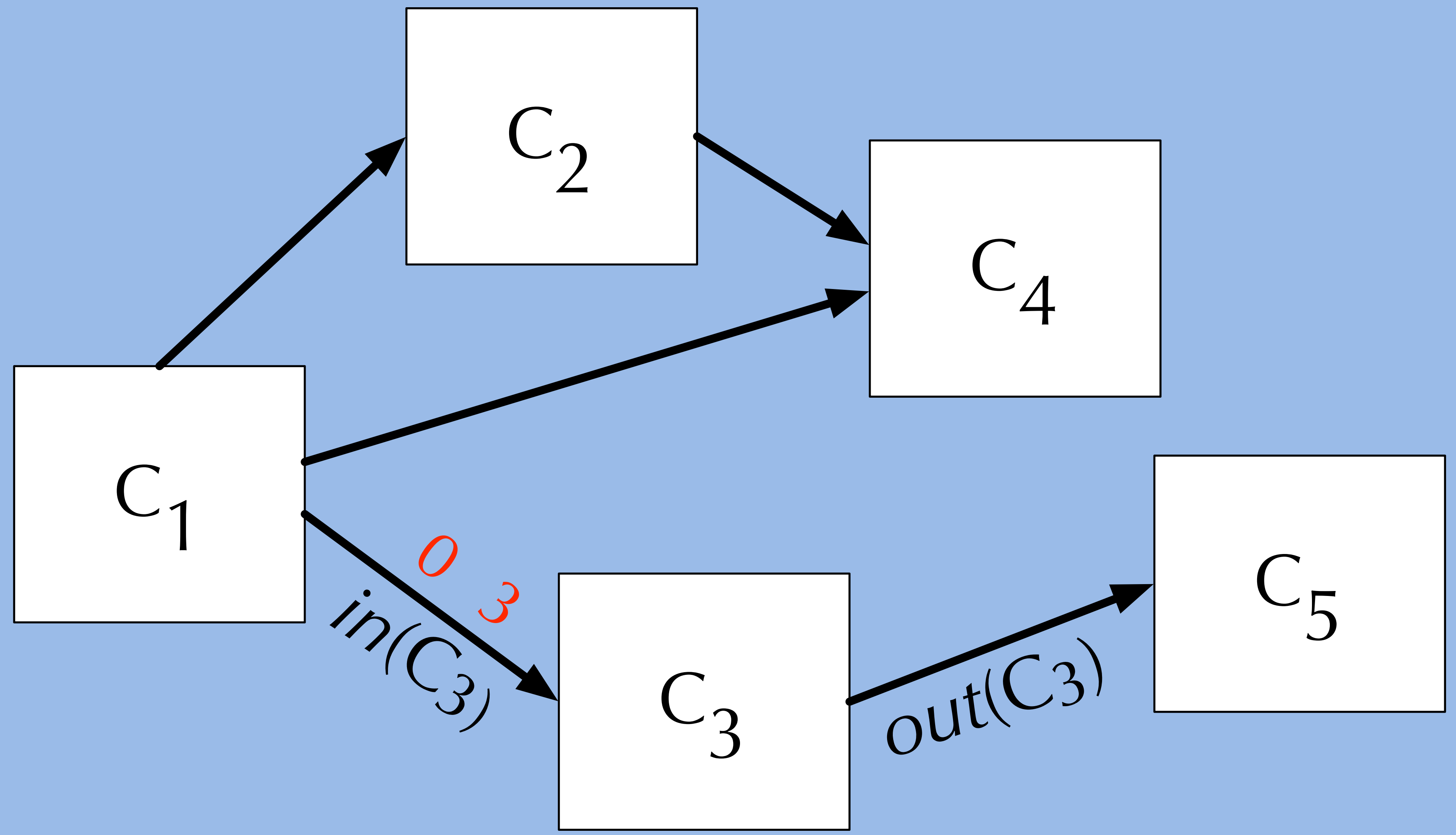
Kahn process networks (Kahn, 1974)



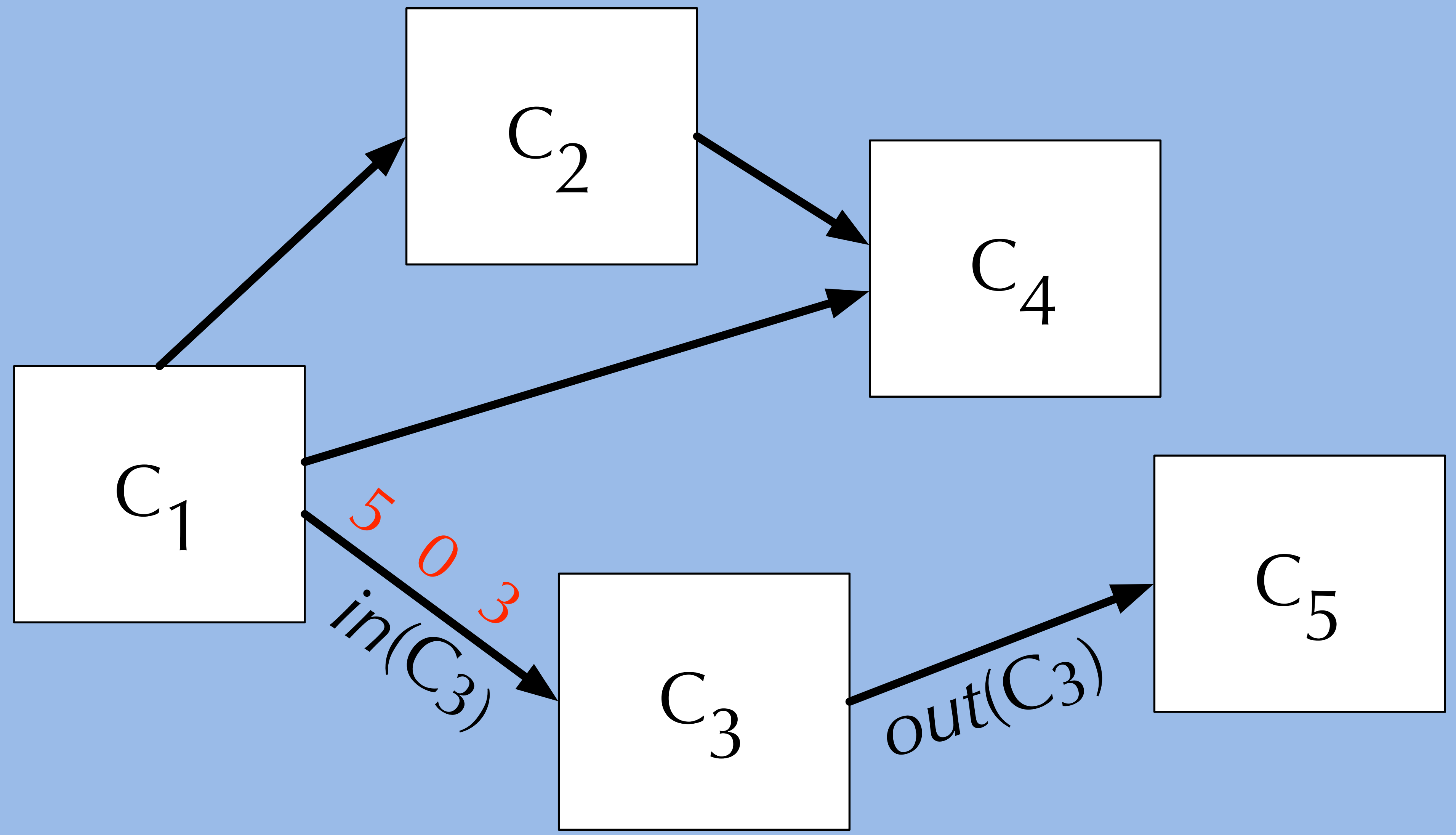
Kahn process networks (Kahn, 1974)



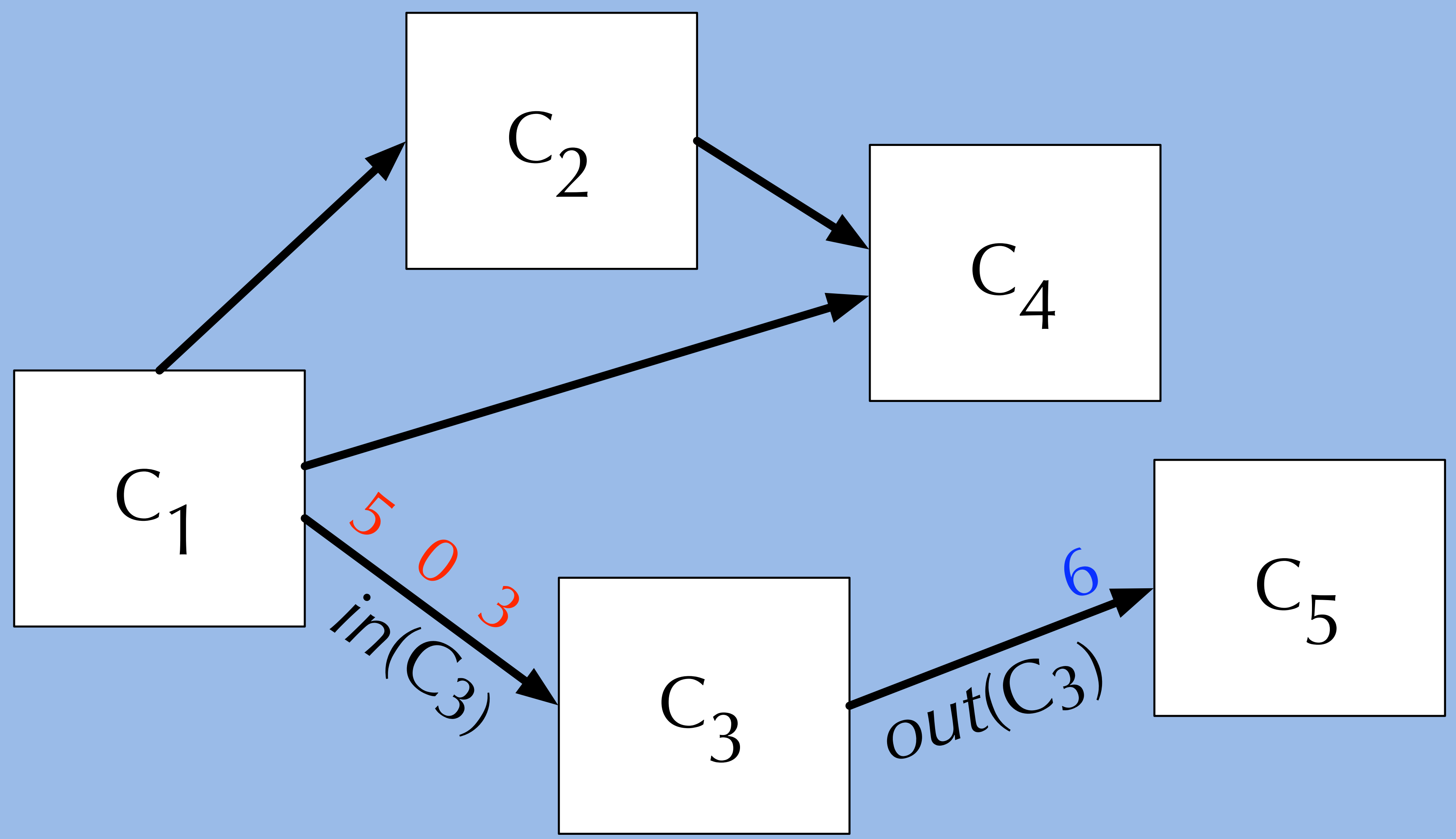
Kahn process networks (Kahn, 1974)



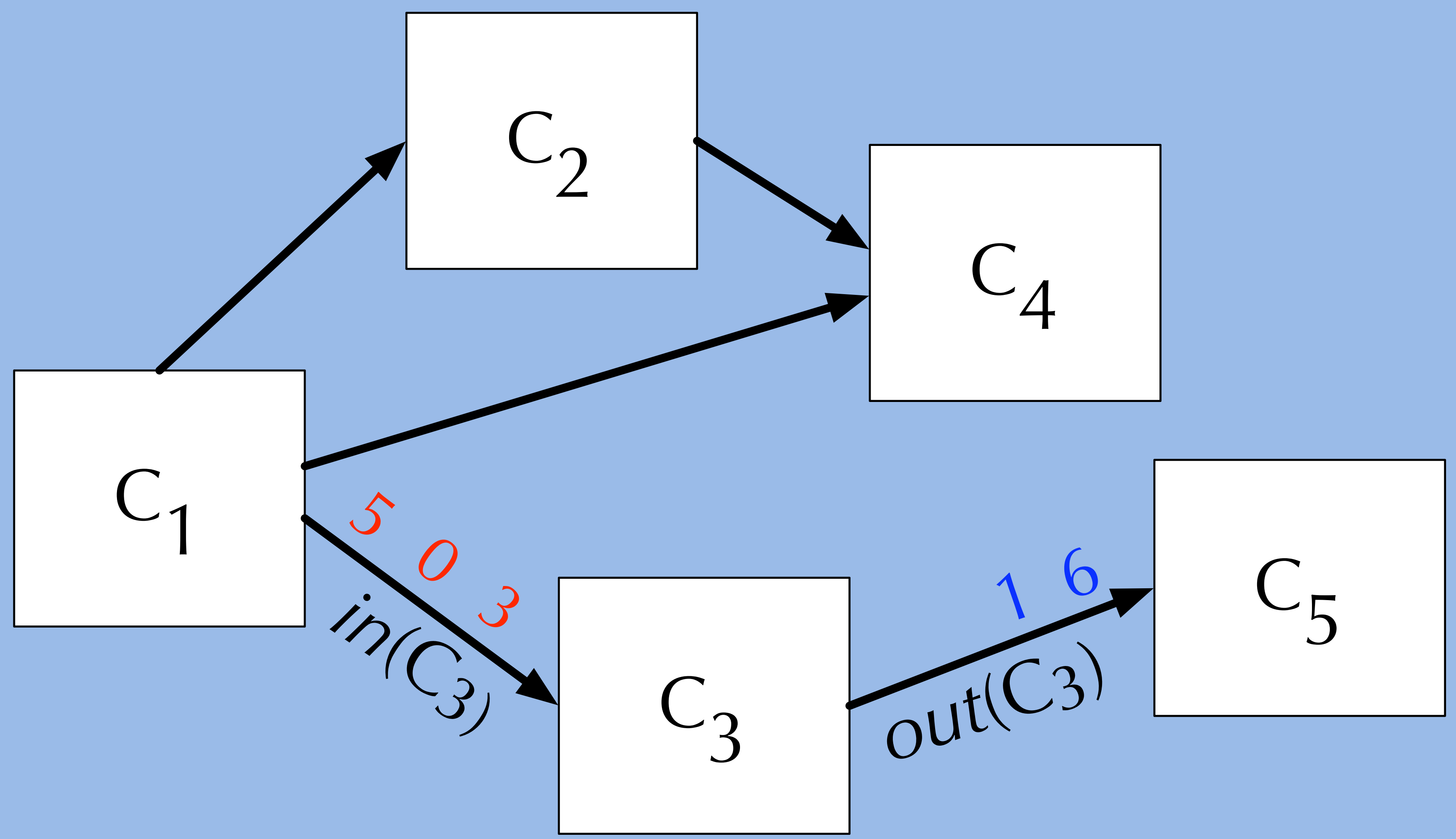
Kahn process networks (Kahn, 1974)



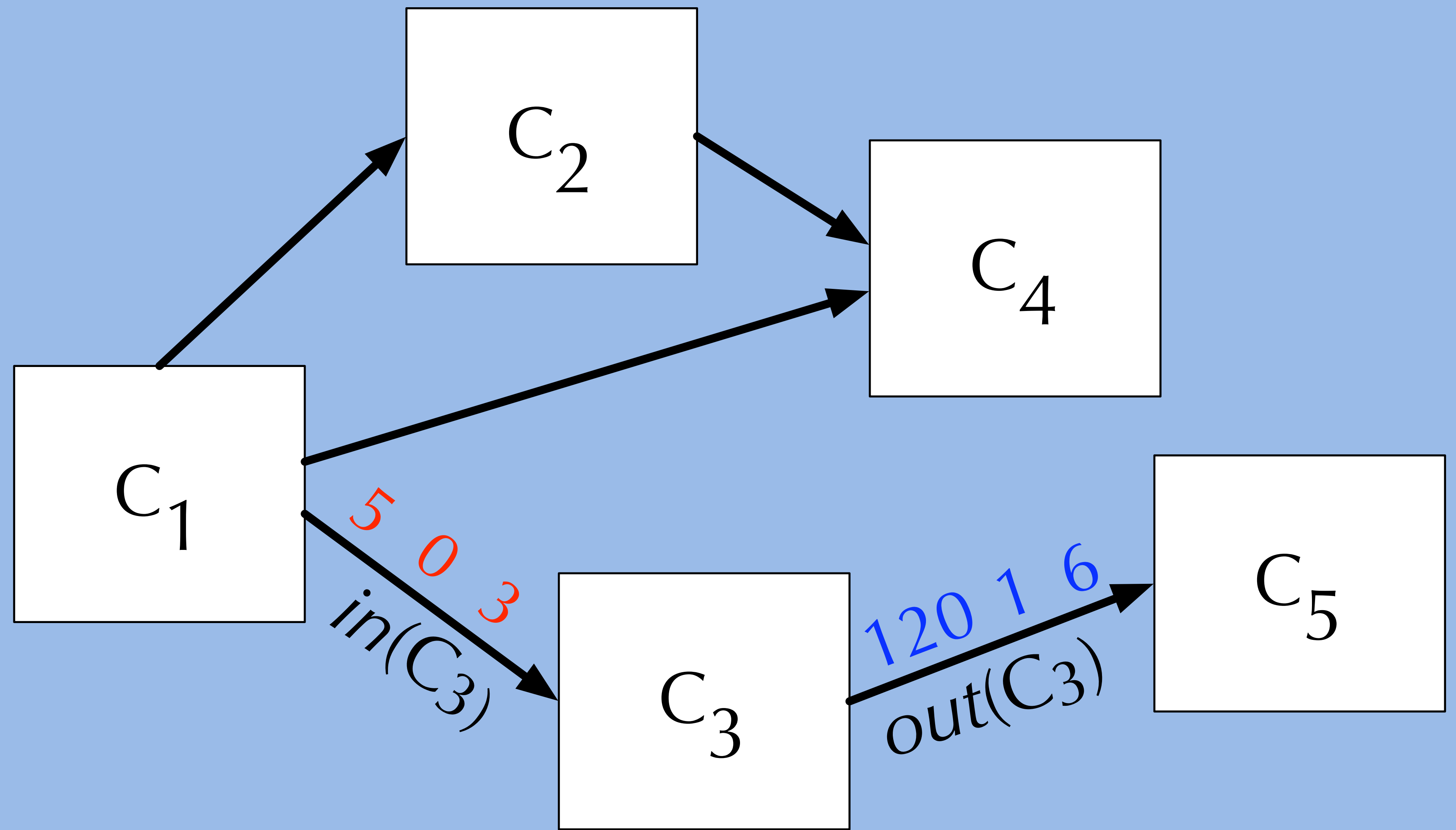
Kahn process networks (Kahn, 1974)



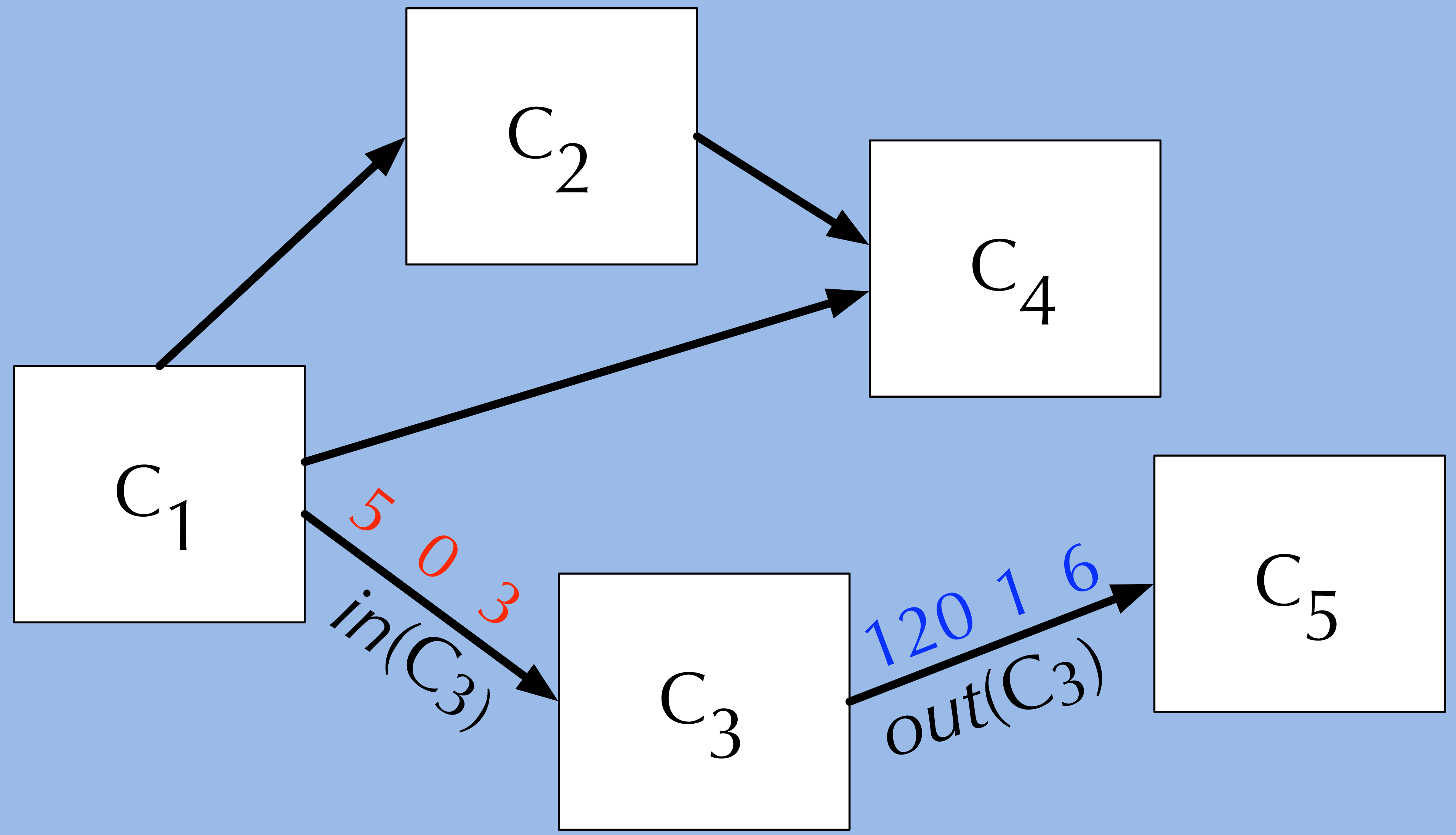
Kahn process networks (Kahn, 1974)



Kahn process networks (Kahn, 1974)

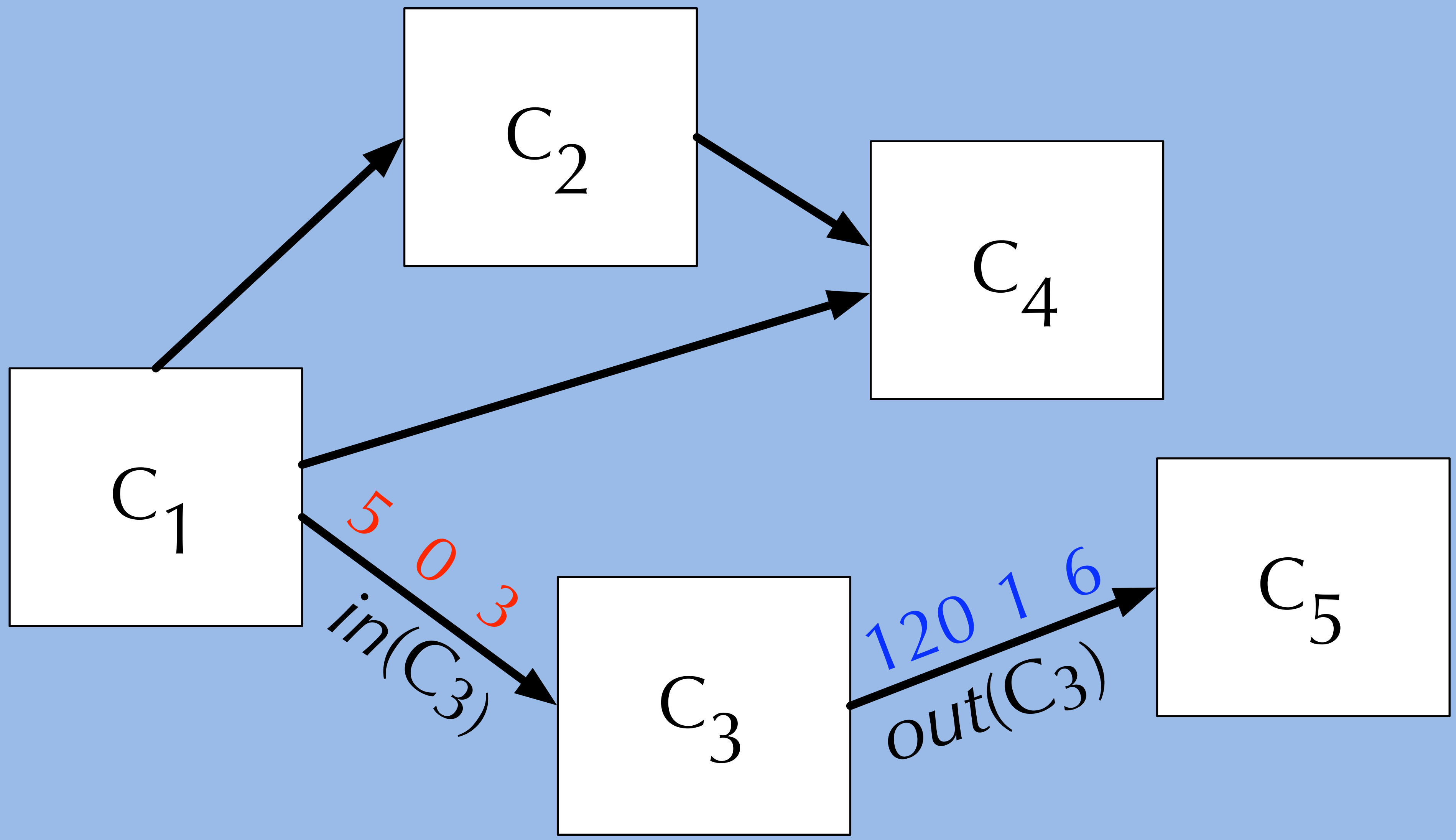


Kahn process networks (Kahn, 1974)



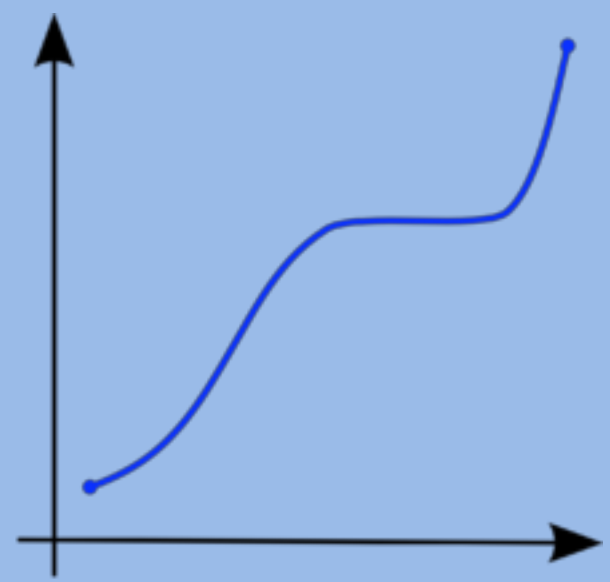
hist(*in*(C$_3$)): [3, 0, 5, ...]

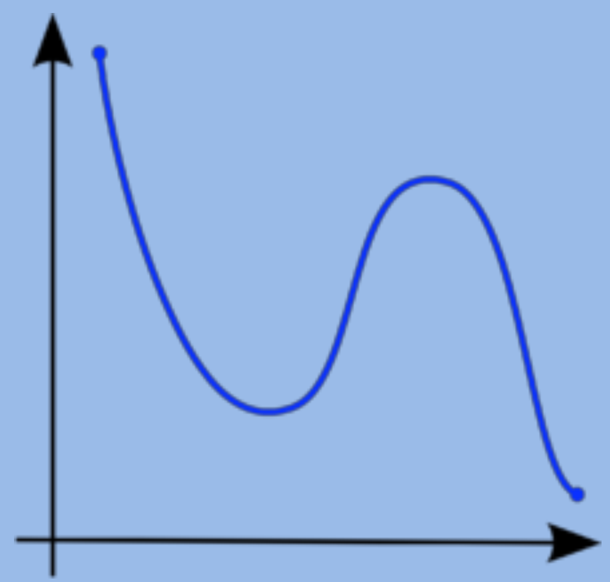
Kahn process networks (Kahn, 1974)



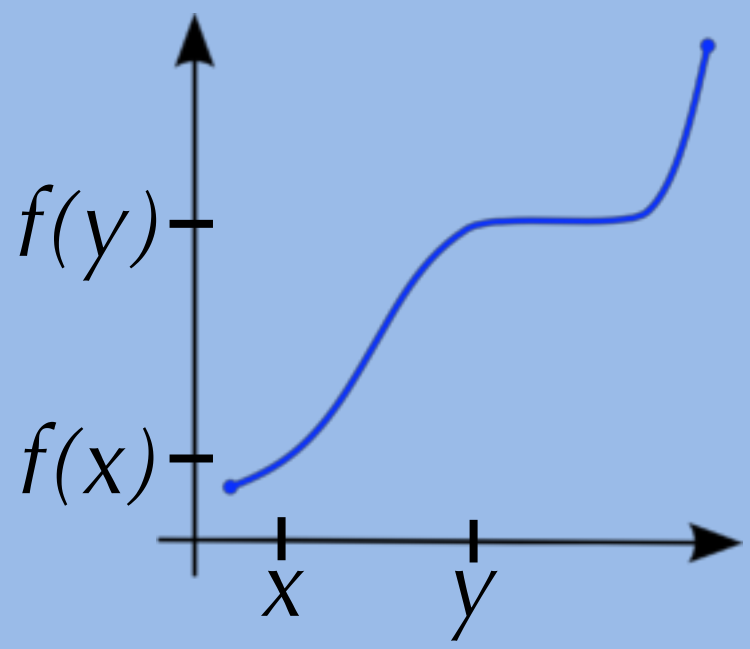
hist(*in*(C3)): [3, 0, 5, ...] *hist*(*out*(C3)): [6, 1, 120, ...]

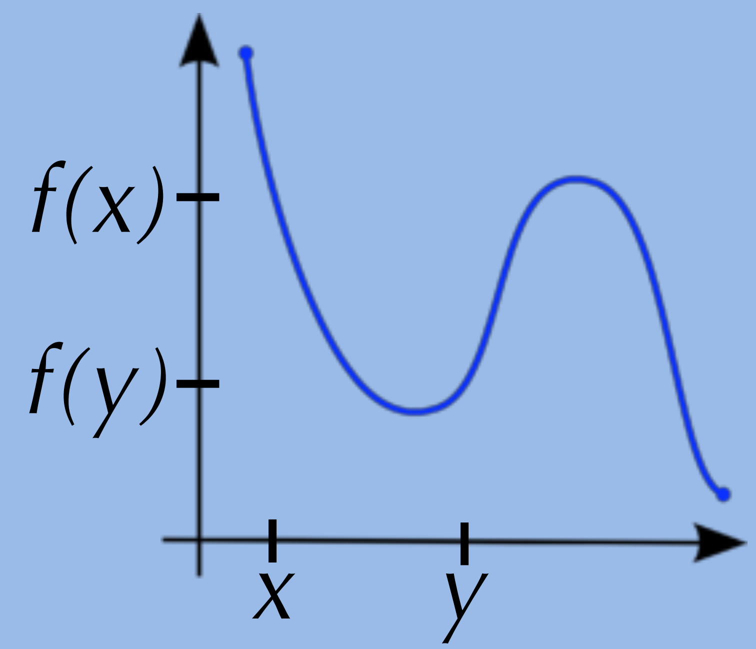
Monotonicity



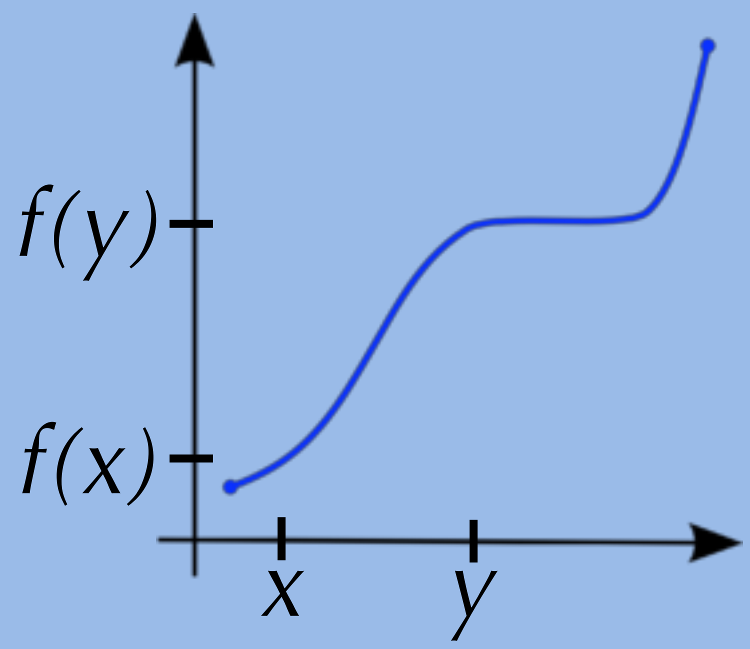


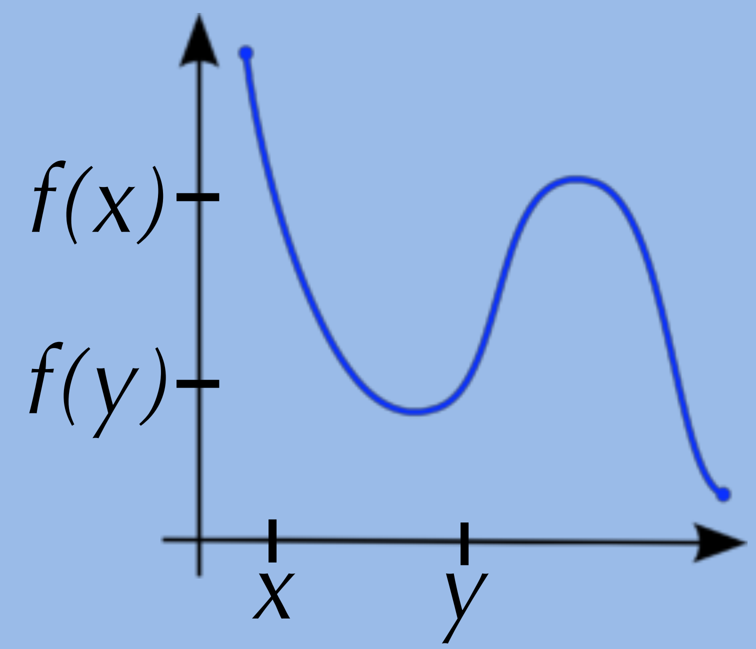
Monotonicity





Monotonicity

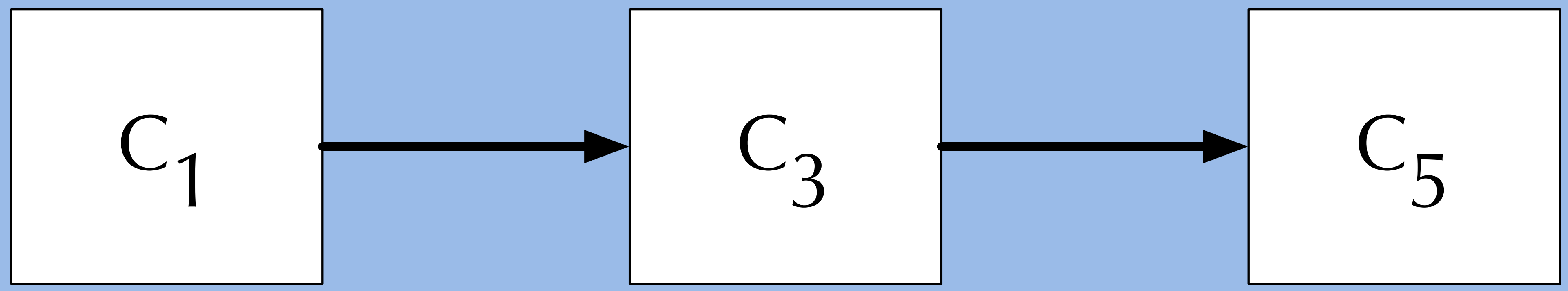




f is monotonic iff $x \leq y \implies f(x) \leq f(y)$

Monotonicity in KPNs

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



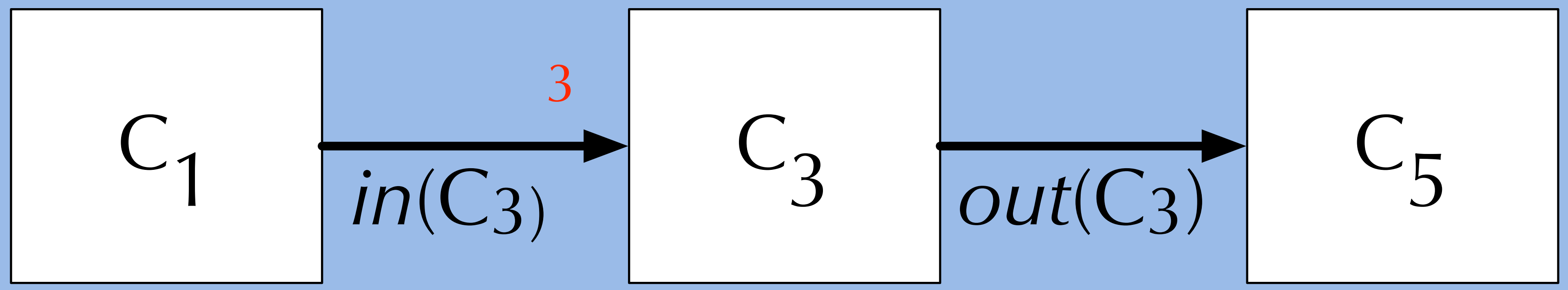
Monotonicity in KPNs

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

$$\boxed{C_1} \xrightarrow[\mathit{in}(C_3)]{} \boxed{C_3} \xrightarrow[\mathit{out}(C_3)]{} \boxed{C_5}$$

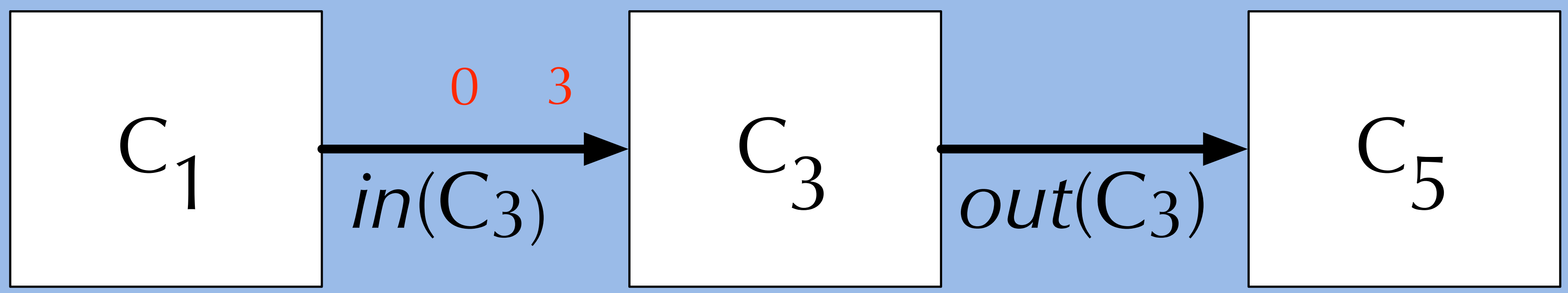
Monotonicity in KPNs

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



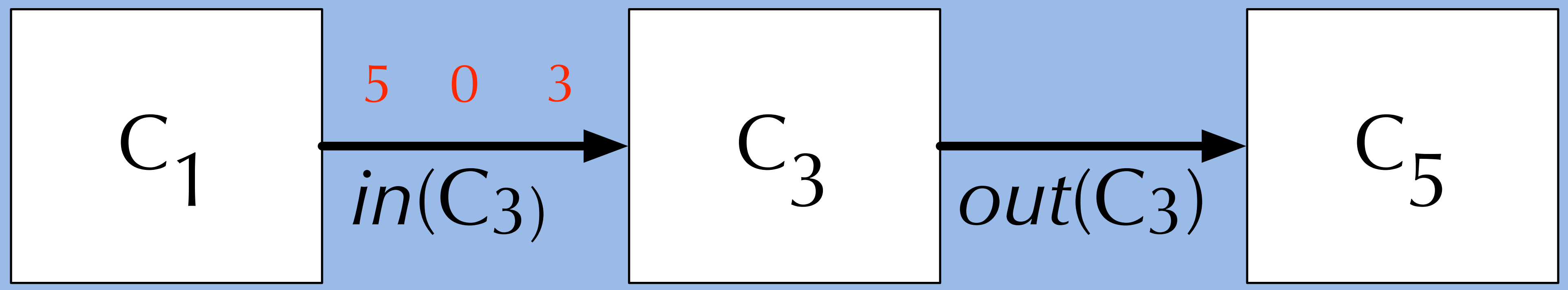
Monotonicity in KPNs

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



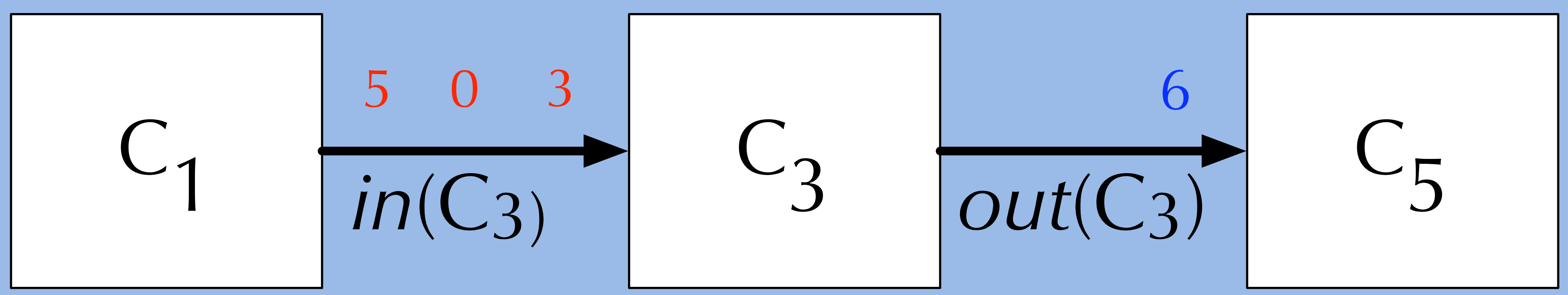
Monotonicity in KPNs

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



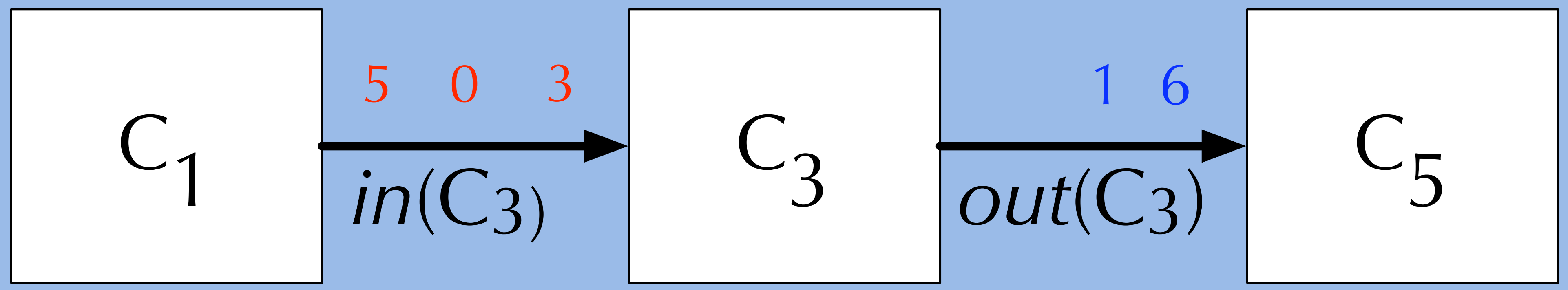
Monotonicity in KPNs

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



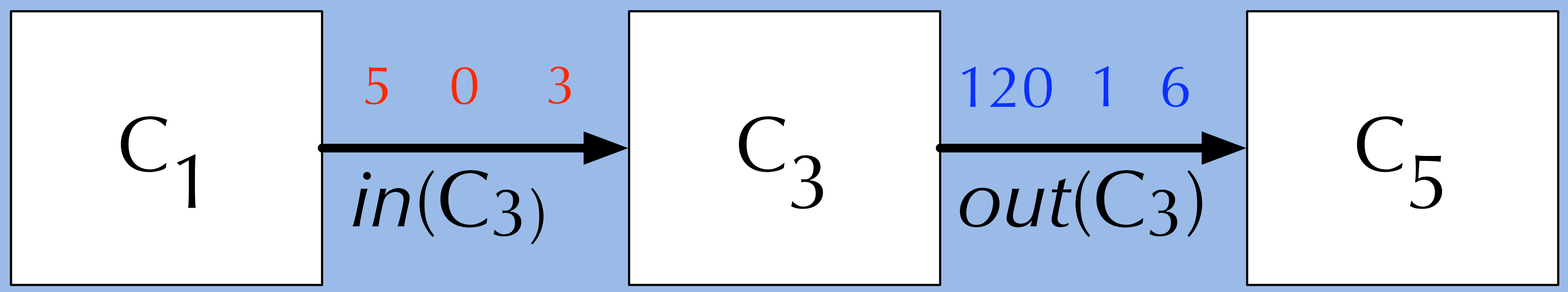
Monotonicity in KPNs

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



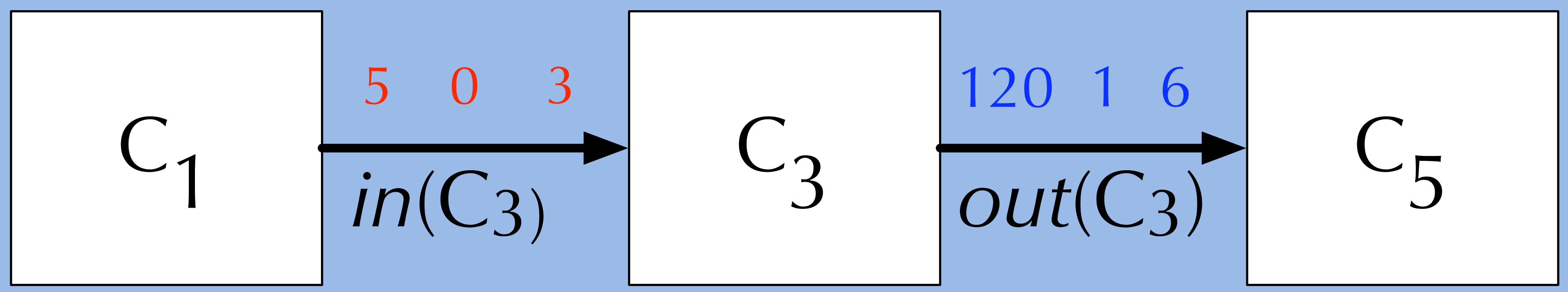
Monotonicity in KPNs

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



Monotonicity in KPNs

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



For KPNs, the $\leq$ 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

$$
\begin{array}{l}
\text{let } \_ = \text{put } l_1 \; 4 \text{ in} \\
\quad \text{let } \_ = \text{put } l_2 \; 3 \text{ in} \\
\quad\quad \text{let par } \_ = \text{put } l_4 \; 3 \\
\quad\quad\quad\quad\quad\;\; \_ = \text{put } l_3 \; 5 \\
\quad\quad\quad \text{in get } l_4
\end{array}
$$

Store:

Back to single-assignment languages

$$\begin{array}{l}
\text{let } _ = \text{put } l_1 \; 4 \text{ in} \\
\quad \text{let } _ = \text{put } l_2 \; 3 \text{ in} \\
\qquad \text{let par } _ = \text{put } l_4 \; 3 \\
\qquad \qquad \qquad _ = \text{put } l_3 \; 5 \\
\qquad \quad \text{in get } l_4
\end{array}$$

Store:

l_1	4

Back to single-assignment languages

$$
\begin{array}{l}
\text{let } \_ = \text{put } l_1 \ 4 \text{ in} \\
\quad \text{let } \_ = \text{put } l_2 \ 3 \text{ in} \\
\qquad \text{let par } \_ = \text{put } l_4 \ 3 \\
\qquad\qquad\quad\;\; \_ = \text{put } l_3 \ 5 \\
\qquad\quad \text{in get } l_4
\end{array}
$$

Store:

l_1	4
l_2	3

Back to single-assignment languages

$$
\begin{array}{l}
\text{let } \_ = \text{put } l_1 \ 4 \text{ in} \\
\quad \text{let } \_ = \text{put } l_2 \ 3 \text{ in} \\
\qquad \text{let par } \_ = \text{put } l_4 \ 3 \\
\qquad\qquad\qquad \_ = \text{put } l_3 \ 5 \\
\qquad\quad \text{in get } l_4
\end{array}
$$

Store:

l_1	4
l_2	3
l_3	5

Back to single-assignment languages

$$
\begin{array}{l}
\text{let } \_ = \text{put } l_1\ 4 \text{ in} \\
\quad \text{let } \_ = \text{put } l_2\ 3 \text{ in} \\
\quad\quad \text{let par } \_ = \text{put } l_4\ 3 \\
\quad\quad\quad\quad\quad\quad \_ = \text{put } l_3\ 5 \\
\quad\quad\quad \text{in get } l_4
\end{array}
$$

Store:

l_1	4
l_2	3
l_3	5
l_4	3

Back to single-assignment languages

$$
\begin{array}{l}
\text{let \_} = \text{put } l_1 \; 4 \text{ in} \\
\quad \text{let \_} = \text{put } l_2 \; 3 \text{ in} \\
\quad\quad \text{let par \_} = \text{put } l_4 \; 3 \\
\quad\quad\quad\quad\quad\quad \text{\_} = \text{put } l_3 \; 5 \\
\quad\quad\quad \text{in get } l_4
\end{array}
$$

Store:

l_1	4
l_2	3
l_3	5
l_4	3

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

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

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

Generalizing our notion of monotonicity

For stores, the ≤ relation is ⊆:

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

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

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

Generalizing our notion of monotonicity

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

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

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

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

Idea: restrict reads

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

Idea: restrict reads

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

Idea: restrict reads

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

$$\begin{aligned}&\text{let } \_ = \text{put } l\ 3 \text{ in}\\ &\quad \text{let par } v = \text{get } l\ 4\\ &\qquad\qquad \_ = \text{put } l\ 4\\ &\qquad\qquad \_ = \text{put } l\ 5\\ &\quad \text{in } v\end{aligned}$$

Idea: restrict reads

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

$$\begin{aligned}
&\text{let } _ = \text{put } l\ 3 \text{ in} \\
&\quad \text{let par } v = \text{get } l\ 4 \\
&\qquad\qquad\quad _ = \text{put } l\ 4 \\
&\qquad\qquad\quad _ = \text{put } l\ 5 \\
&\quad\ \text{in } v
\end{aligned}$$

Idea: restrict reads

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

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

Idea: restrict reads

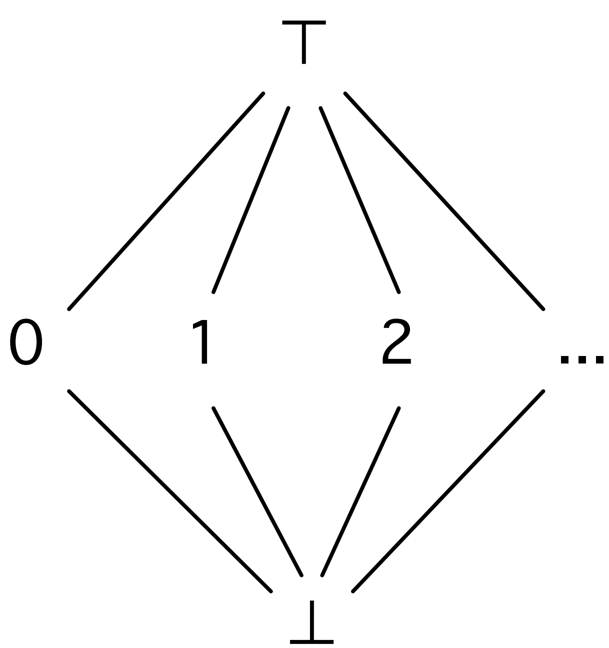
$$\begin{aligned}
&\text{let } _ = \text{put } l\ 3 \text{ in}\\
&\quad \text{let par } v = \text{get } l\ 4\\
&\quad\quad\quad\quad\ \ _ = \text{put } l\ 4\\
&\quad\ \ \text{in } v
\end{aligned}$$

return 4

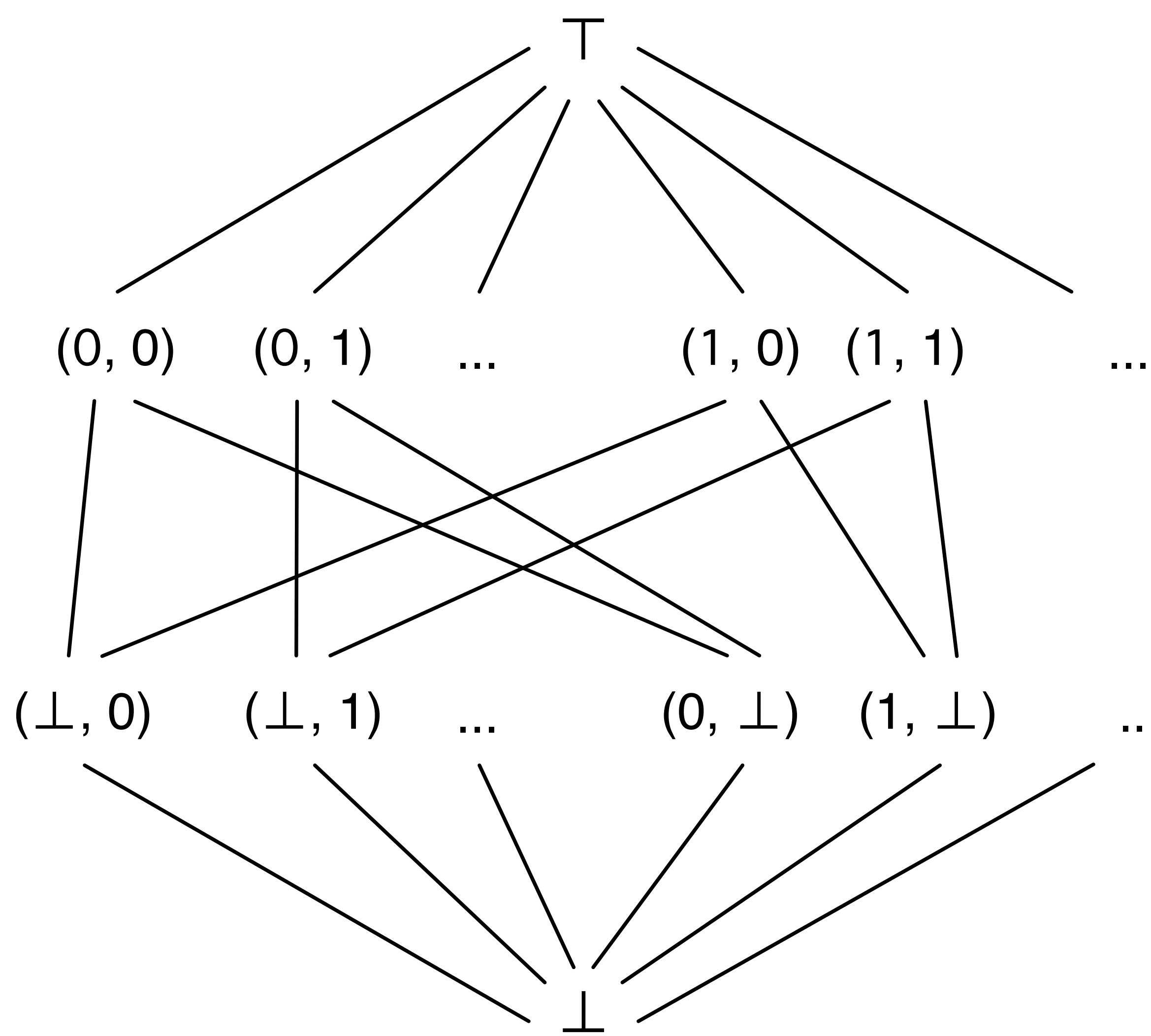
$$\begin{aligned}
&\text{let } _ = \text{put } l\ 3 \text{ in}\\
&\quad \text{let par } v = \text{get } l\ 4\\
&\quad\quad\quad\quad\ \ _ = \text{put } l\ 4\\
&\quad\quad\quad\quad\ \ _ = \text{put } l\ 5\\
&\quad\ \ \text{in } v
\end{aligned}$$

Monotonically increasing writes
+ restricted reads
= deterministic-by-construction parallelism

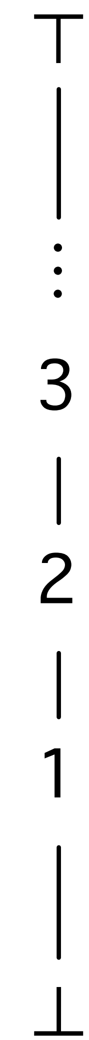
Parameterizing our language: “LVars”



IVar

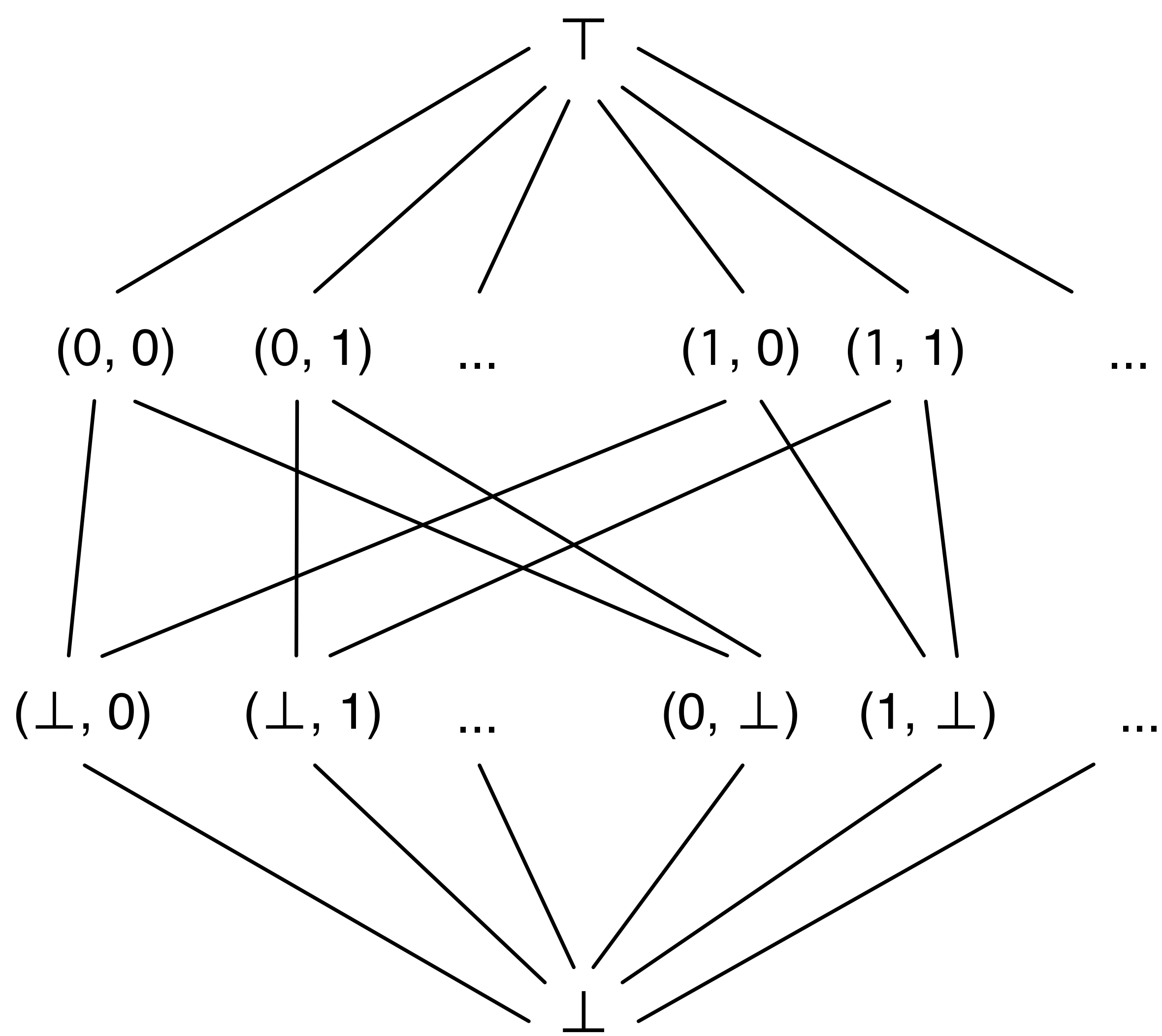


Pair of IVars



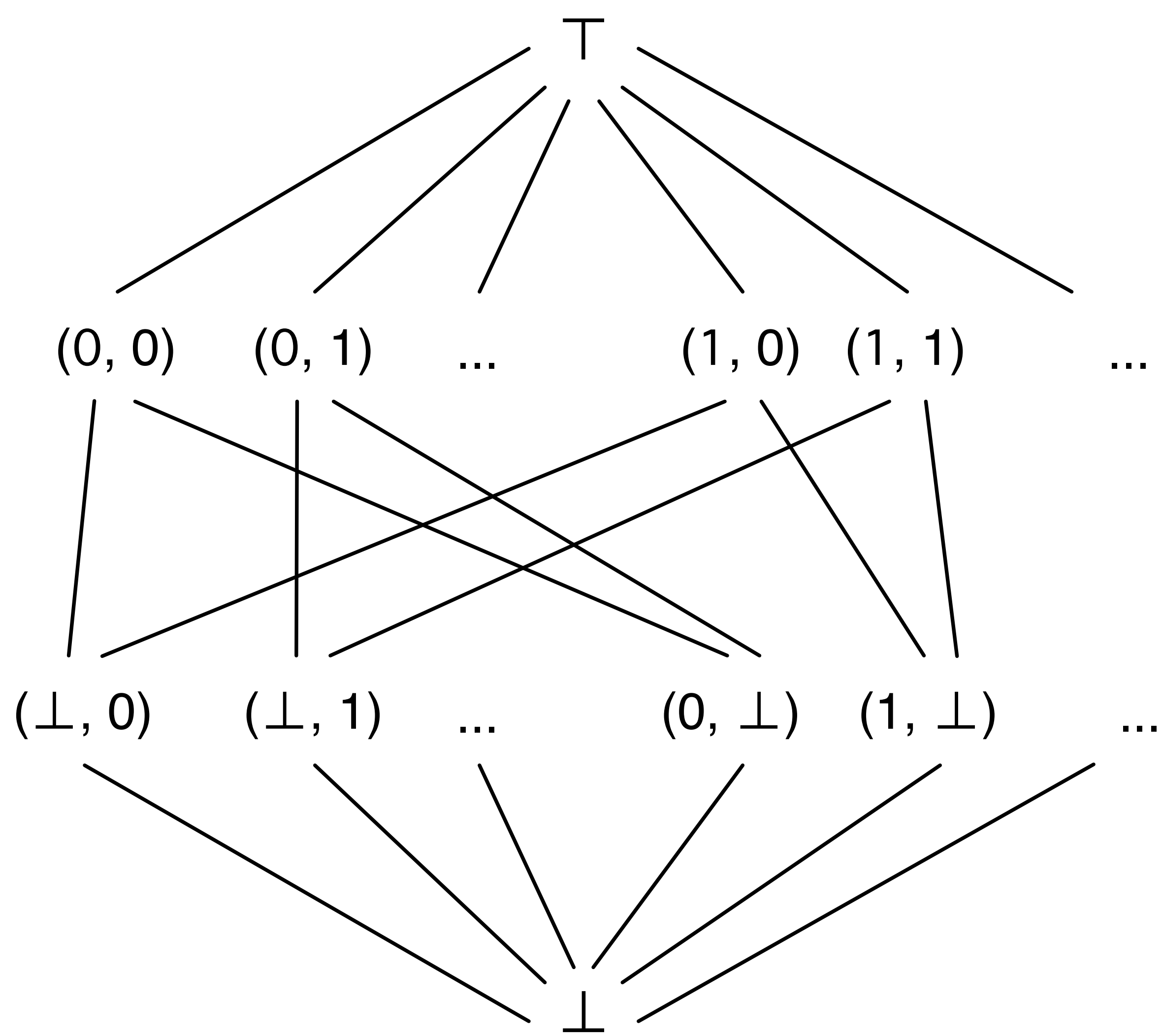
Counter

Parameterizing our language: “LVars”



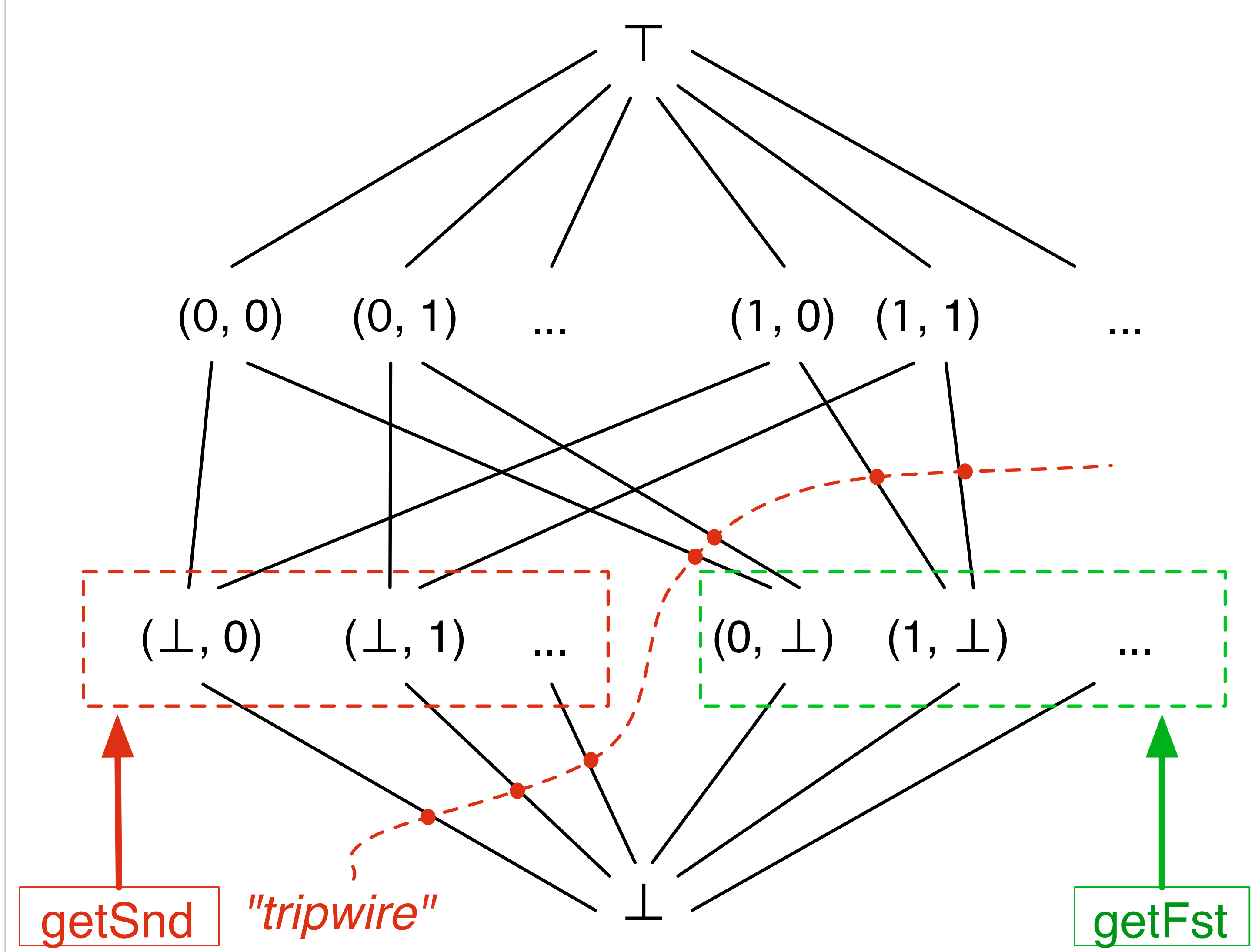
Pair of IVars

Parameterizing our language: "LVars"



Pair of IVars

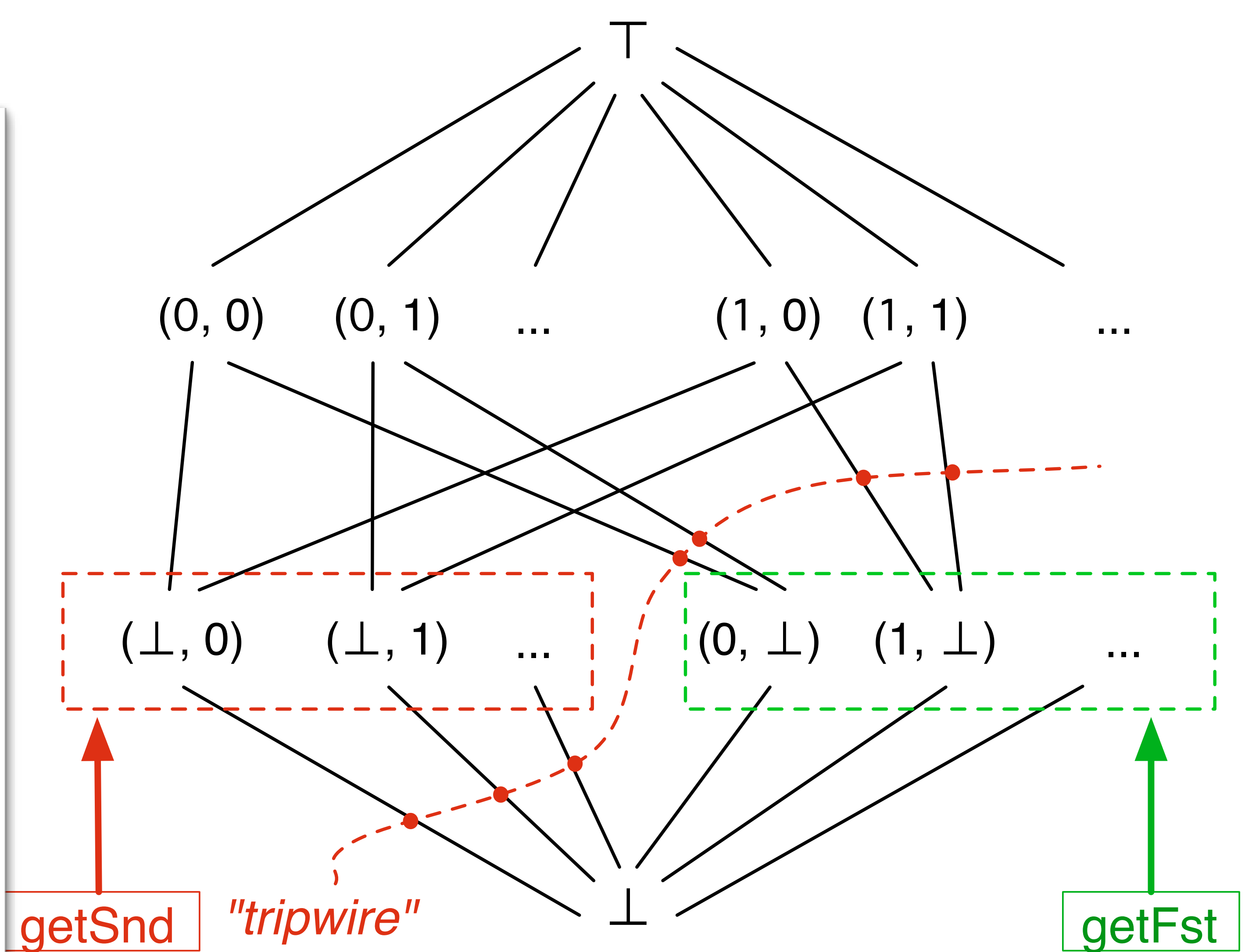
Parameterizing our language: “LVars”



Pair of IVars

Parameterizing our language: "LVars"

$$
\begin{aligned}
&\text{let } p = \text{new in} \\
&\quad \text{let } \_ = \text{put } p \ \{(\bot, 4)\} \text{ in} \\
&\qquad \text{let par } v_1 = \text{getFst } p \\
&\qquad\qquad\quad \_ = \text{put } p \ \{(3, 4)\} \\
&\qquad \text{in } \ldots v_1 \ldots
\end{aligned}
$$



Pair of IVars

More in our paper draft and TR

More in our paper draft and TR

- Complete syntax and semantics

More in our paper draft and TR

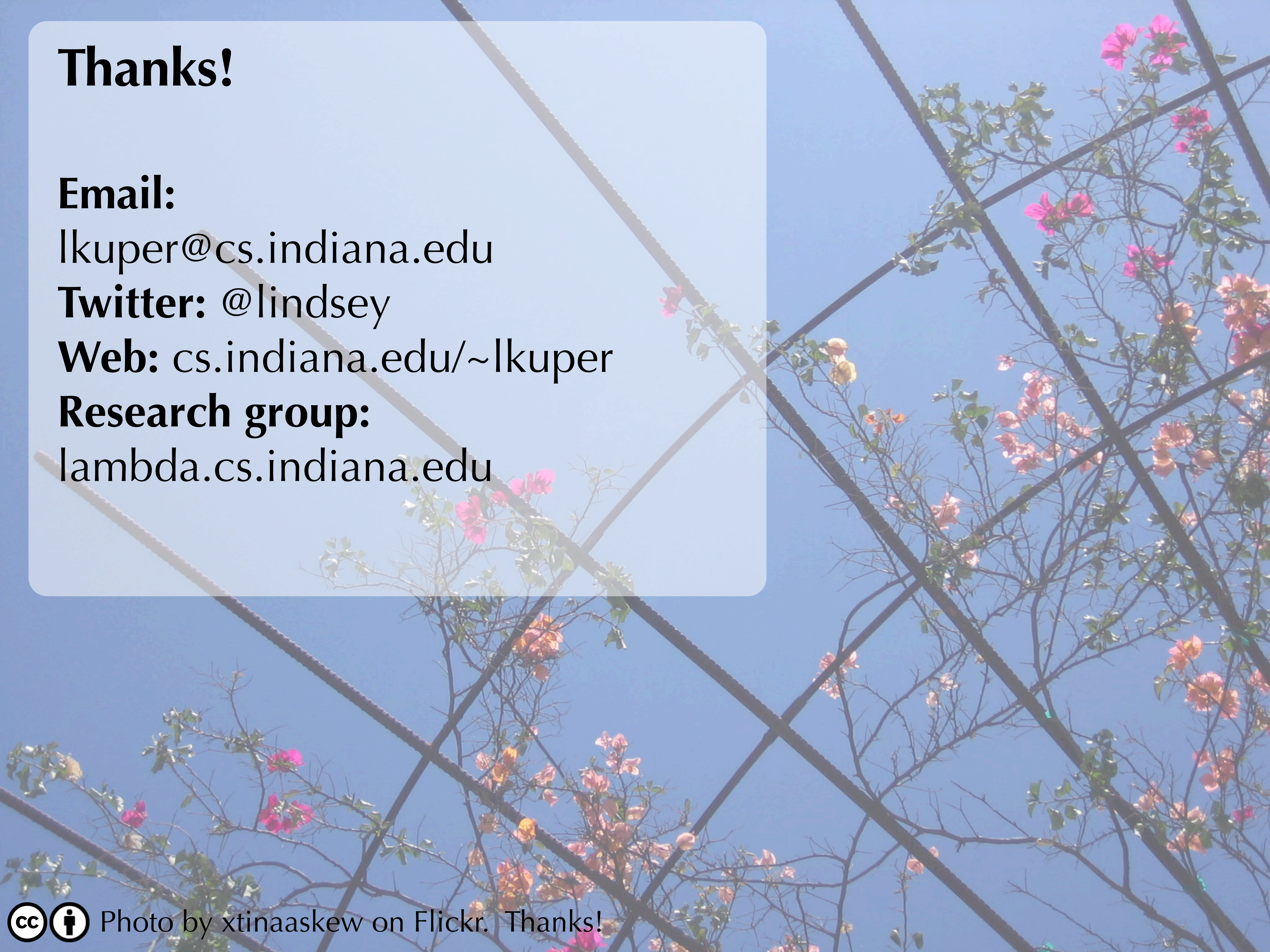
- Complete syntax and semantics
- Proof of determinism
  - A “frame-rule-like” property
  - Location renaming is surprisingly tricky!

More in our paper draft and TR

- 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 and TR

- 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



Thanks!

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