Mixing Metaphors

Actors as Channels and Channels as Actors

Simon Fowler

University of Edinburgh

Joint work with Sam Lindley and Philip Wadler

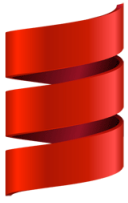
November 10, 2016

THE UNIVERSITY of EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

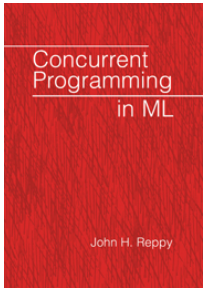
Communication-centric Programming Languages

Go

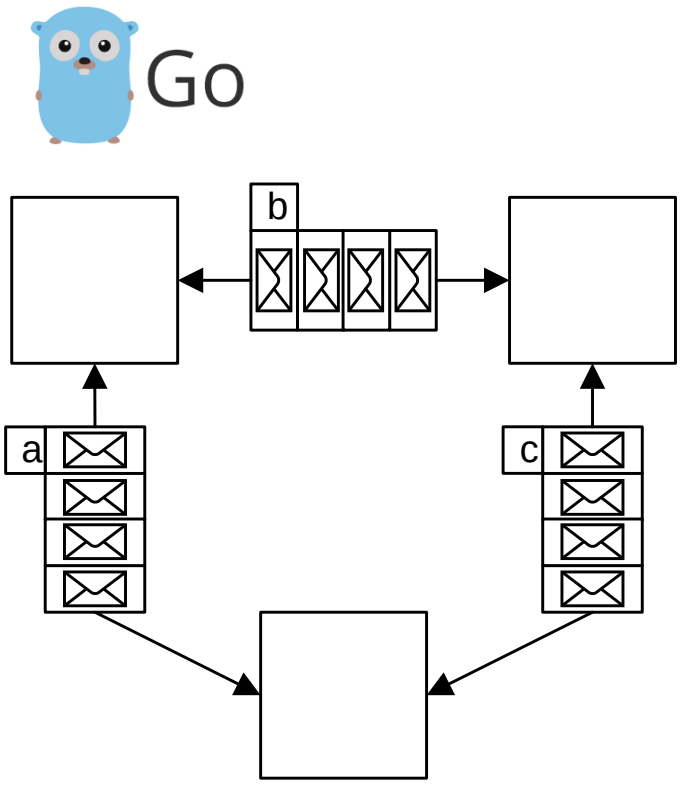


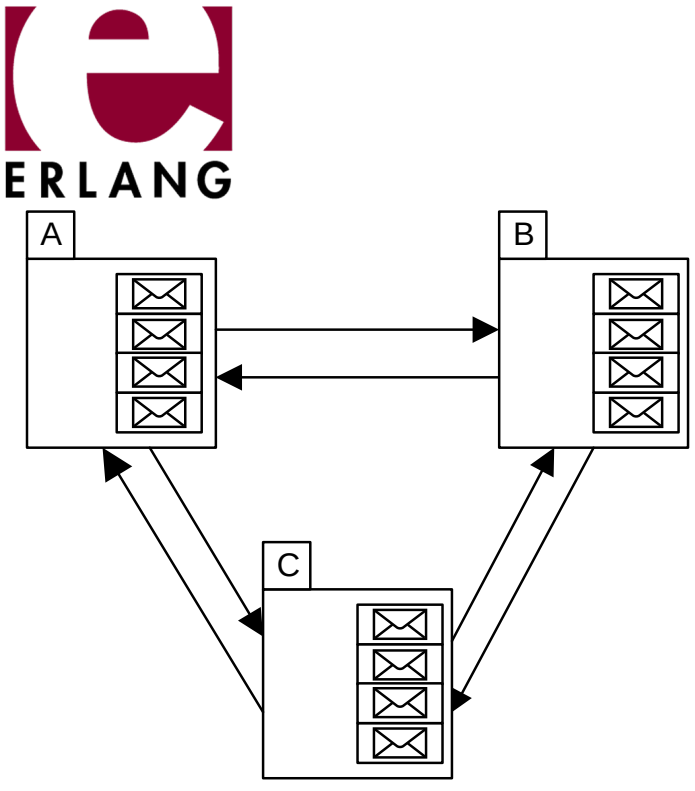
Scala

elixir

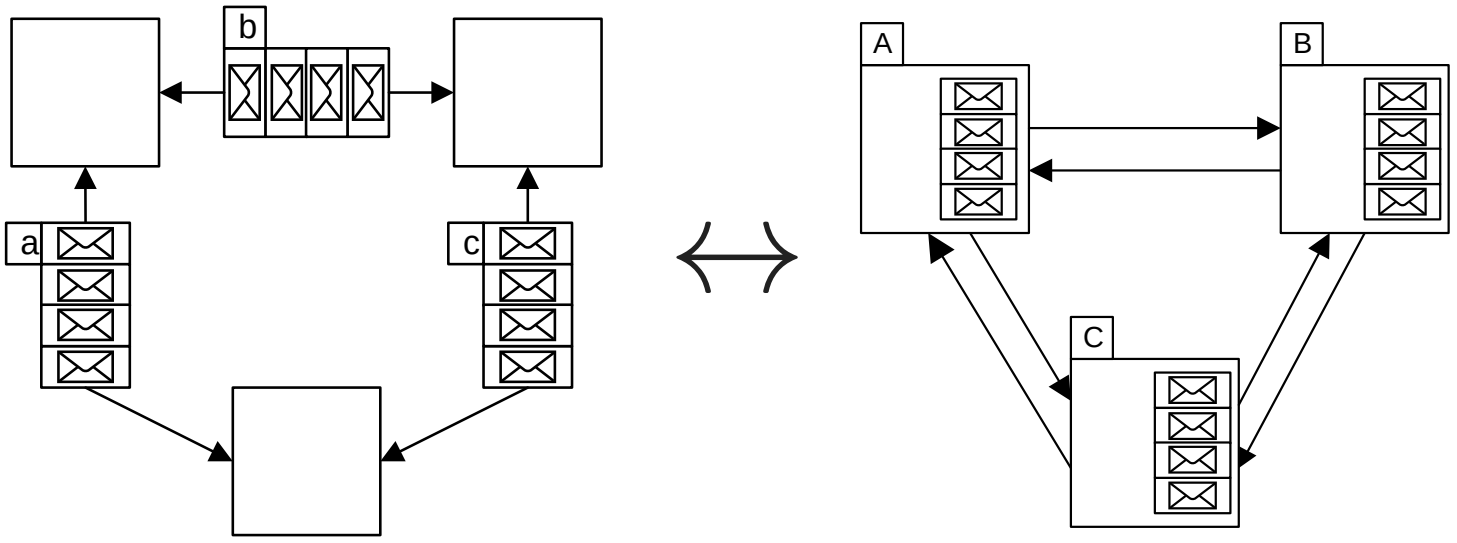


Channels and Actors





Why compare the two?



→ Concise formalisations of how the models fit into functional programming languages

→ Clear up confusion between the two models

→ Towards behavioural typing disciplines!

λ_{ch}: A concurrent λ-calculus for asynchronous channels

Syntax

$$
\begin{array}{rlll}
\text{Types} & A, B & ::= & \mathbf{1} \mid A \to B \mid \mathsf{ChanRef}(A) \\
\text{Variables} & \alpha & ::= & x \mid a \\
\text{Values} & V, W & ::= & \alpha \mid \lambda x.M \mid () \\
\text{Computations} & L, M, N & ::= & V\,W \\
 & & \mid & \mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N \mid \mathsf{return}\ V \\
 & & \mid & \mathsf{fork}\ M \mid \mathsf{give}\ V\,W \mid \mathsf{take}\ V \mid \mathsf{newCh}
\end{array}
$$

- → Separation of values and computations
- → Set of communication and concurrency primitives

λ_{act} Typing Rules

Typing for communication and concurrency primitives

GIVE
$$\frac{\Gamma \vdash V : A \quad \Gamma \vdash W : \mathsf{ChanRef}(A)}{\Gamma \vdash \mathsf{give}\ V\ W : \mathbf{1}}$$

TAKE
$$\frac{\Gamma \vdash V : \mathsf{ChanRef}(A)}{\Gamma \vdash \mathsf{take}\ V : A}$$

FORK
$$\frac{\Gamma \vdash M : \mathbf{1}}{\Gamma \vdash \mathsf{fork}\ M : \mathbf{1}}$$

NEWCH
$$\frac{}{\Gamma \vdash \mathsf{newCh} : \mathsf{ChanRef}(A)}$$

Note that fork returns **1**: processes are anonymous!

Modelling Concurrent Behaviour: Configurations

$$\mathcal{C} ::= \mathcal{C}_1 \parallel \mathcal{C}_2 \mid (\nu a)\mathcal{C} \mid a(\overrightarrow{V}) \mid M$$

Concurrent λ-calculi model concurrent behaviour by reduction on a language of configurations.

- $\mathcal{C}_1 \parallel \mathcal{C}_2$: parallel composition
- $(\nu a)\mathcal{C}$: name restriction
- $a(\overrightarrow{V})$: buffer with name a containing values of $\overrightarrow{V}$
- M: term evaluating as a thread

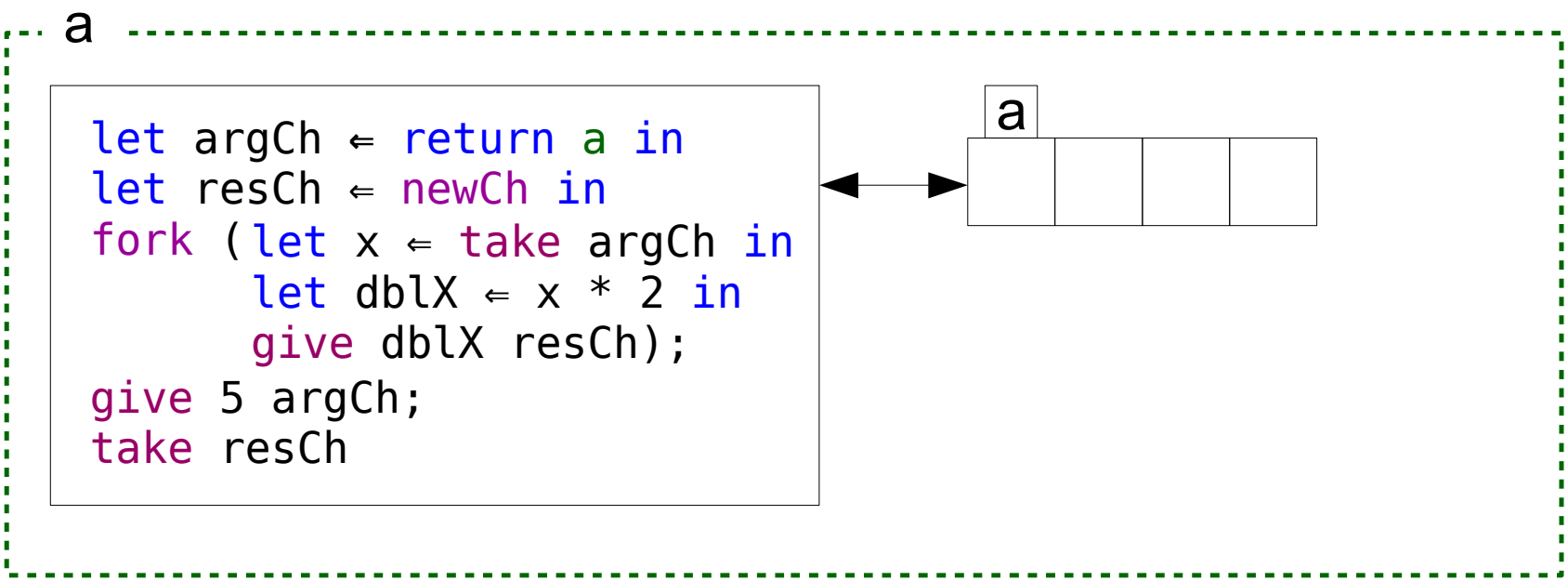
Also configuration typing rules for well-formed configurations.

λ_{ch} Semantics, by Example

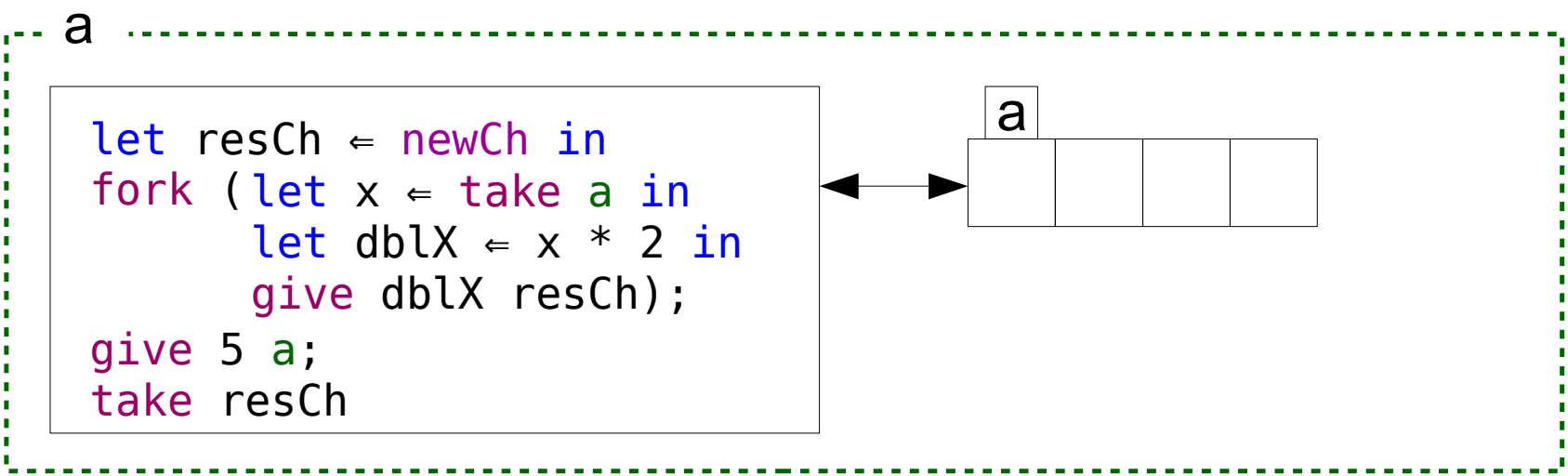
```
let argCh ⇐ newCh in
let resCh ⇐ newCh in
fork (let x ⇐ take argCh in
      let dblX ⇐ x * 2 in
      give dblX resCh);
give 5 argCh;
take resCh
```

- → Create two channels *argCh* and *resCh*
- → Spawn a process which:
  - → Receives a value along the argument channel
  - → Doubles it
  - → Sends it along the result channel
- → Then, send a value along the argument channel, and wait for the result

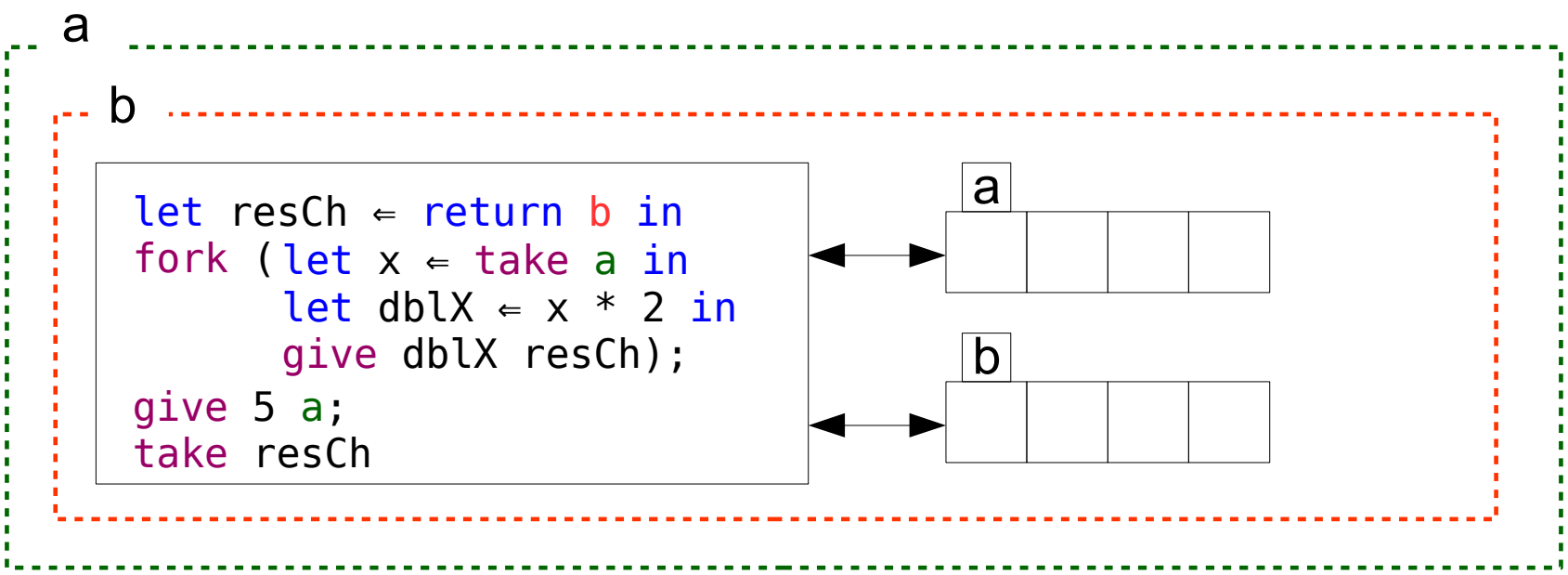
λ_{ch} Semantics, by Example



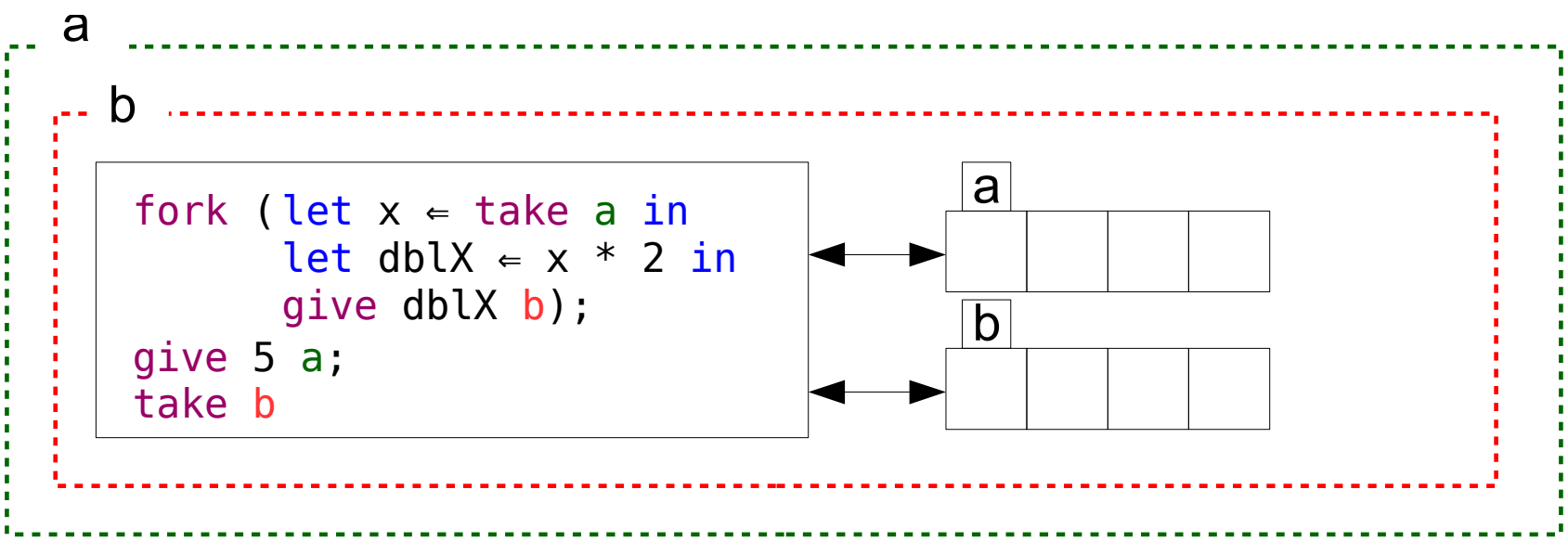
λ_{ch} Semantics, by Example



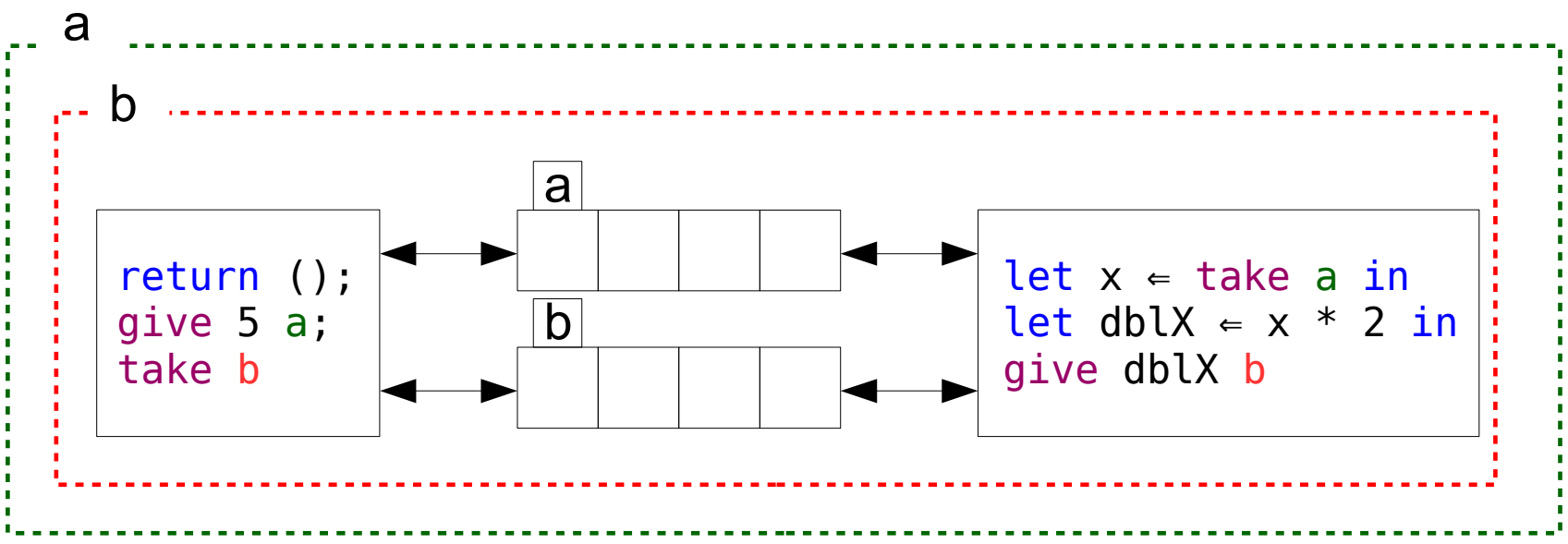
λ_{ch} Semantics, by Example



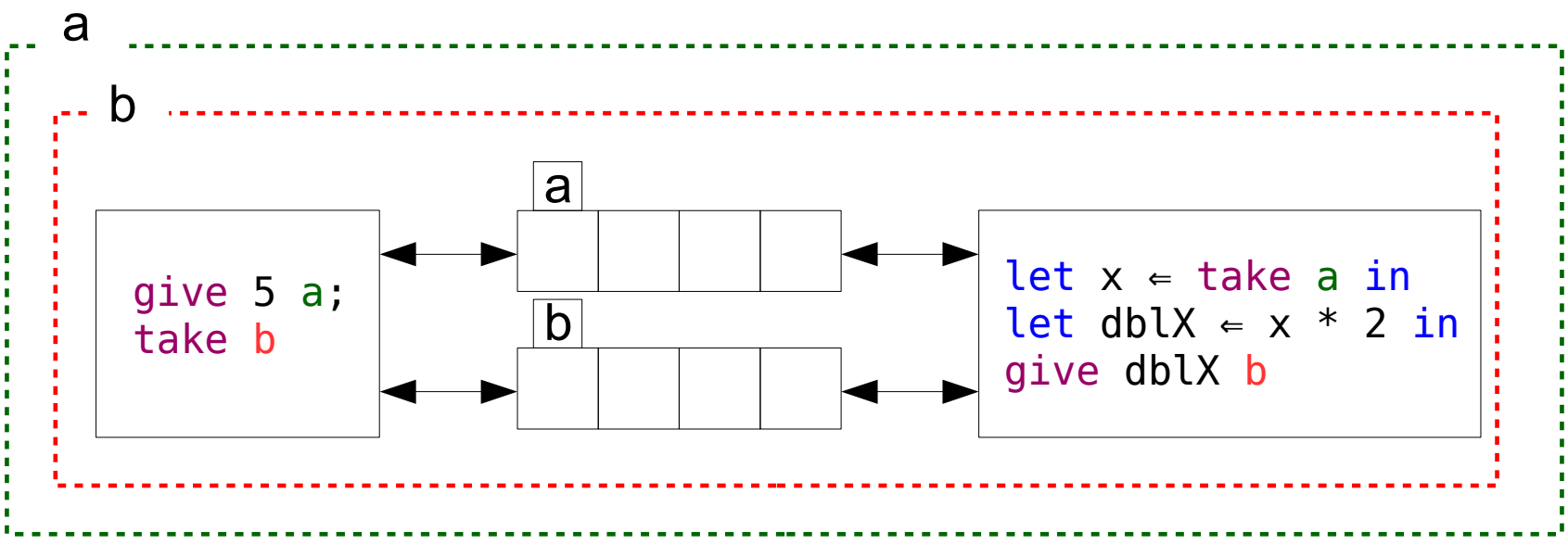
λ_{ch} Semantics, by Example



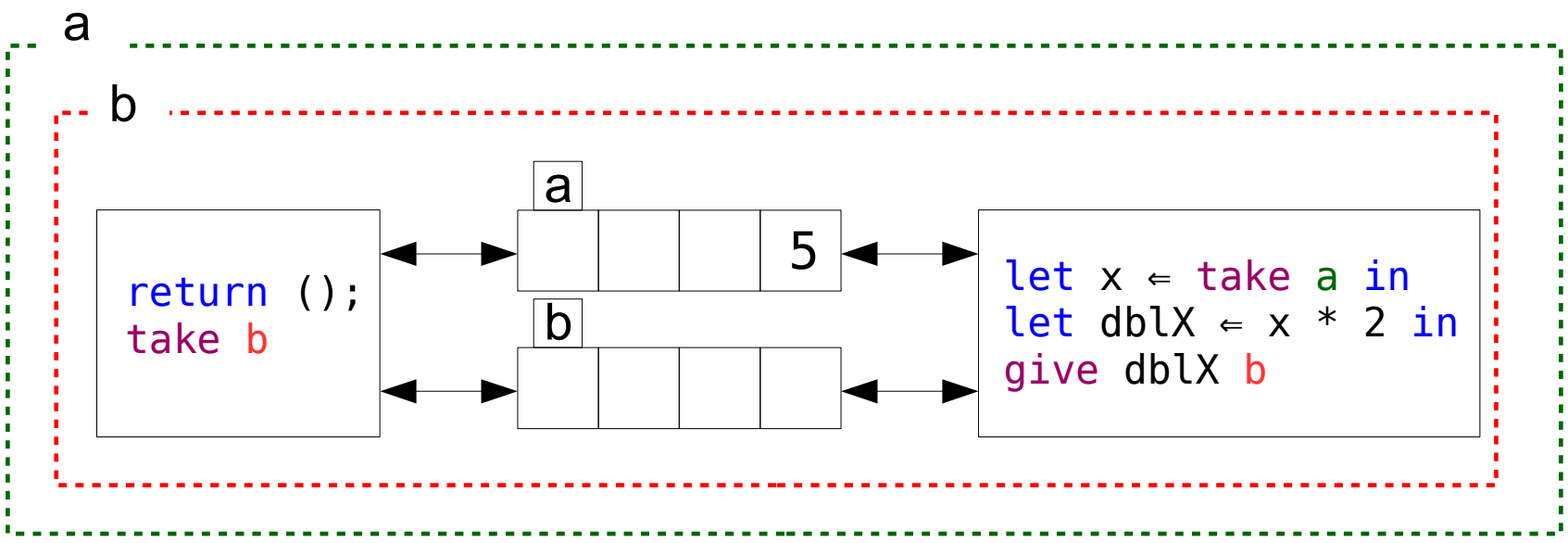
λ_{ch} Semantics, by Example



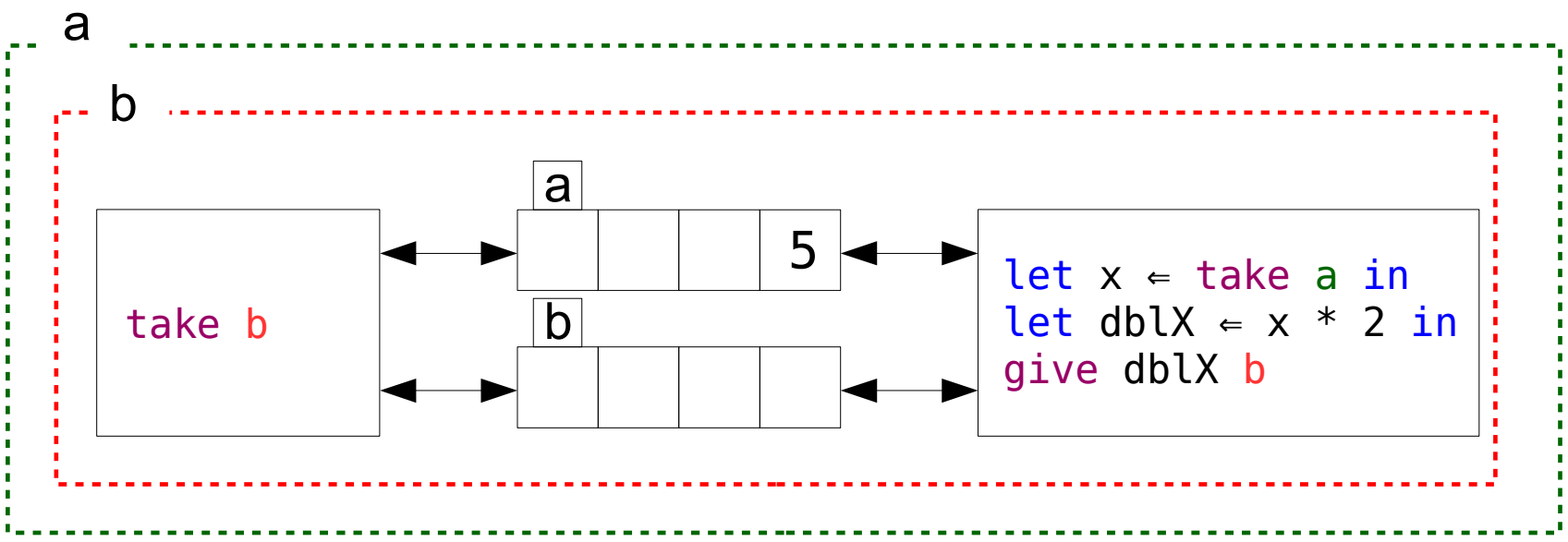
λ_{ch} Semantics, by Example



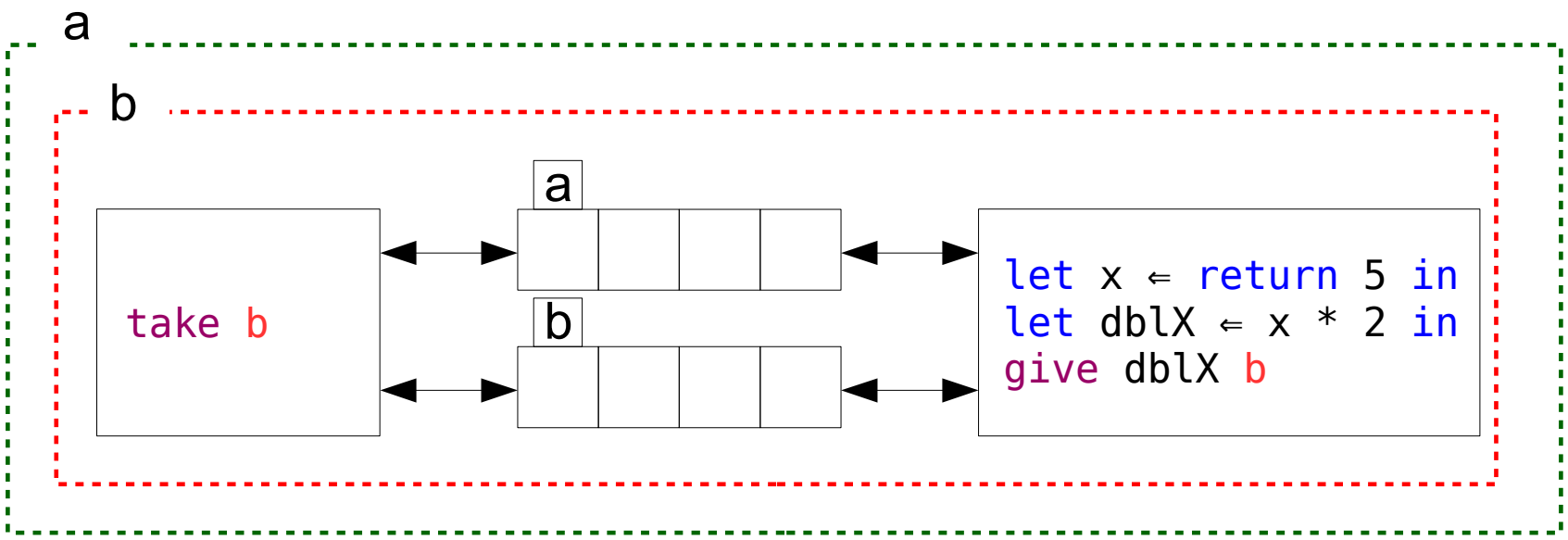
λ_{ch} Semantics, by Example



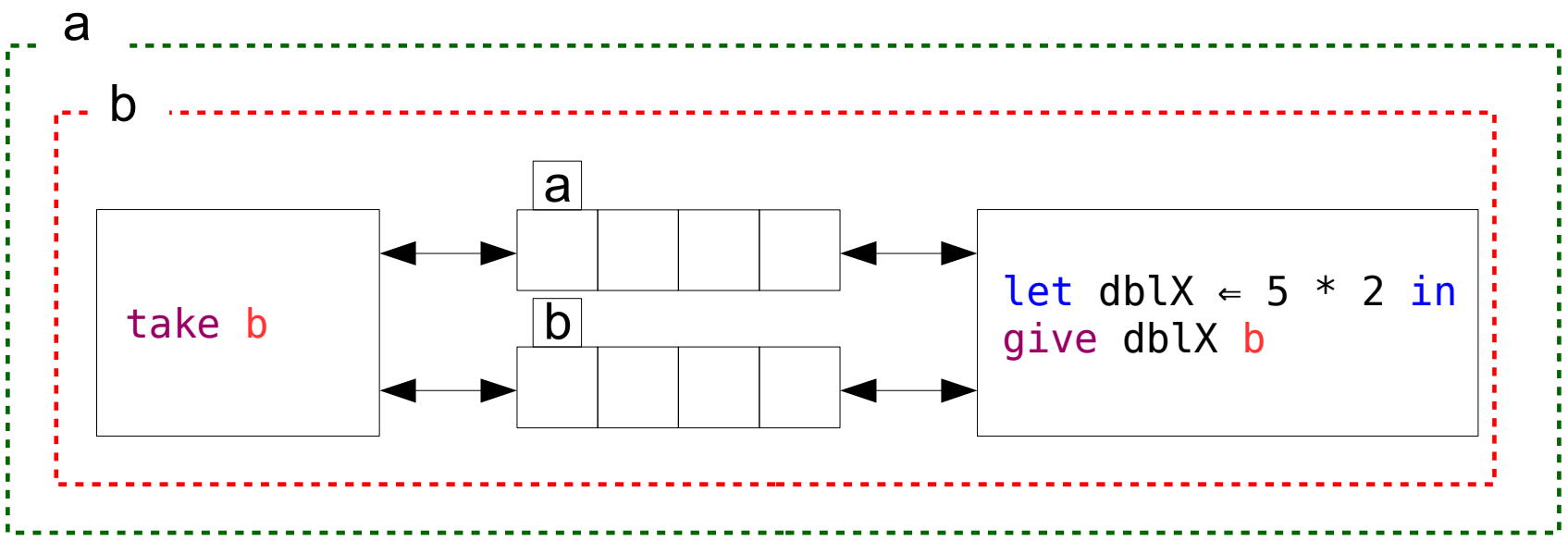
λ_{ch} Semantics, by Example



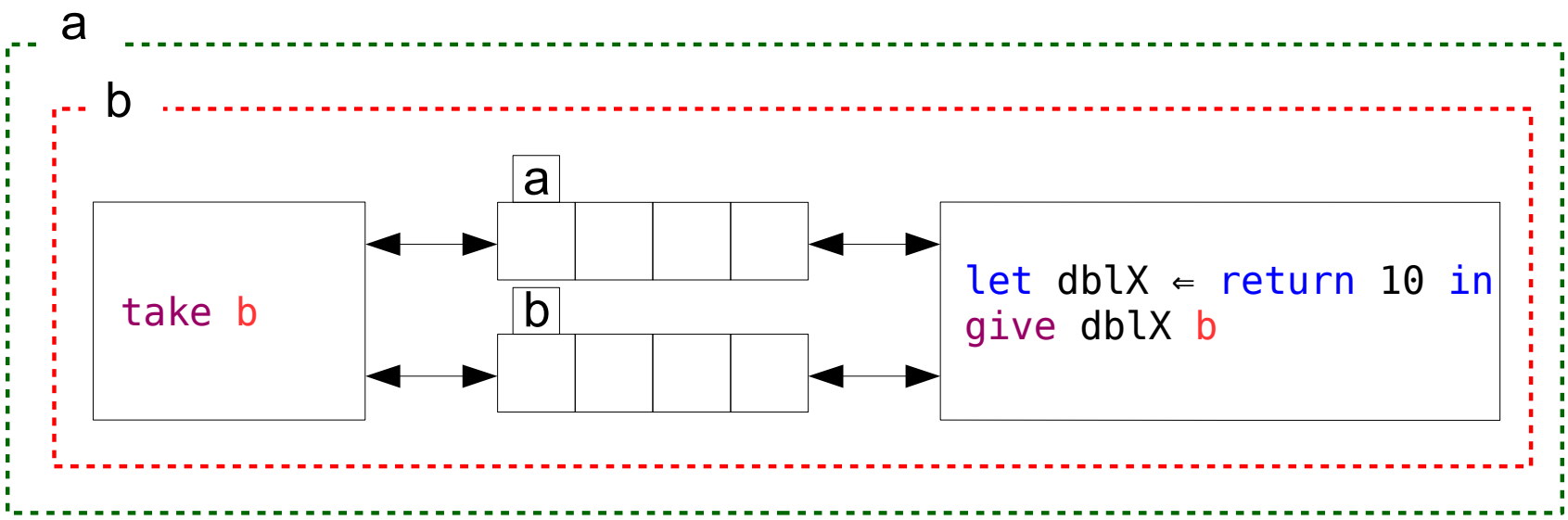
λ_{ch} Semantics, by Example



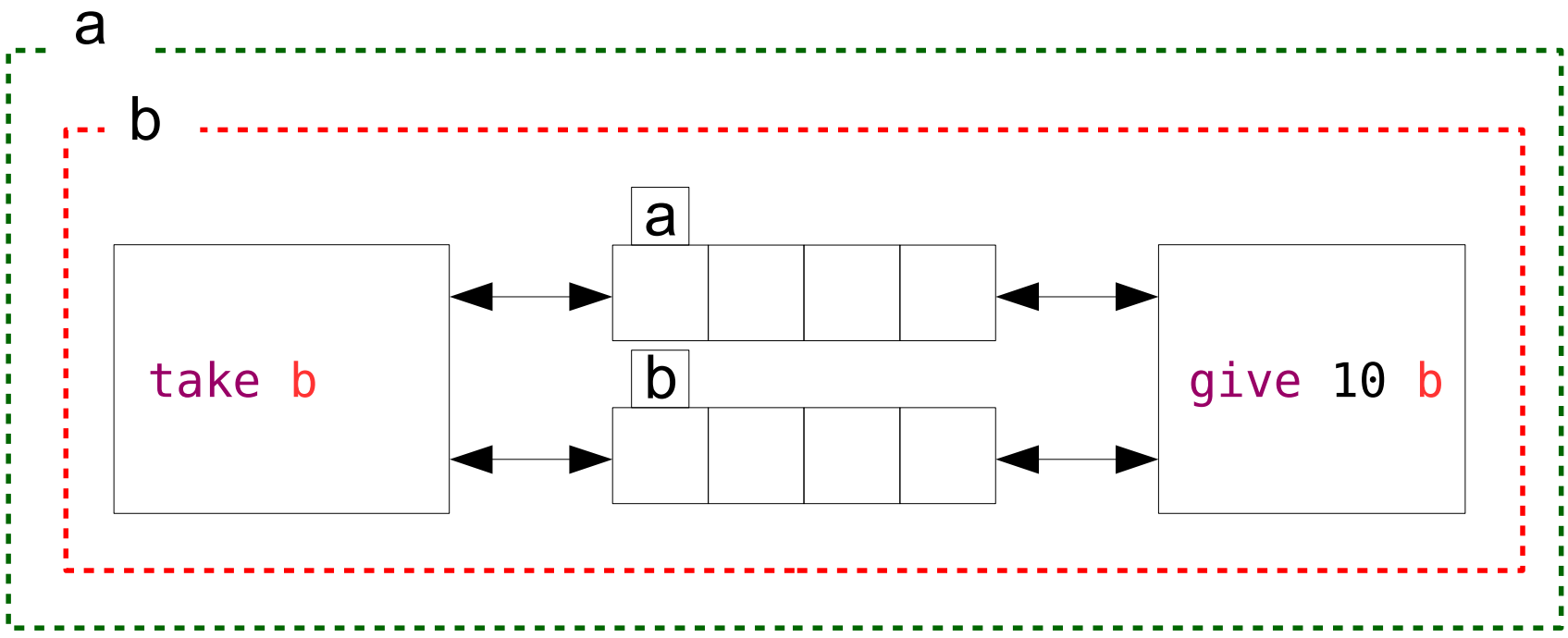
λ_{ch} Semantics, by Example



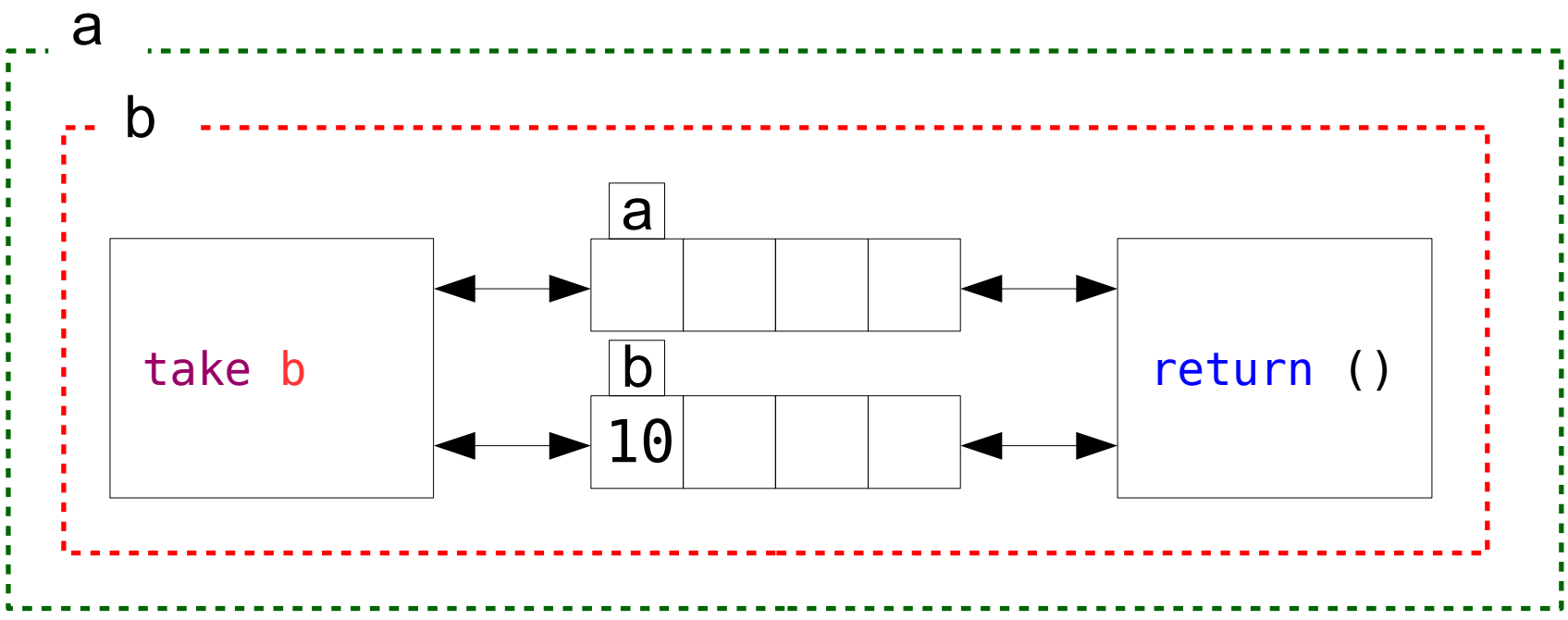
λ_{ch} Semantics, by Example



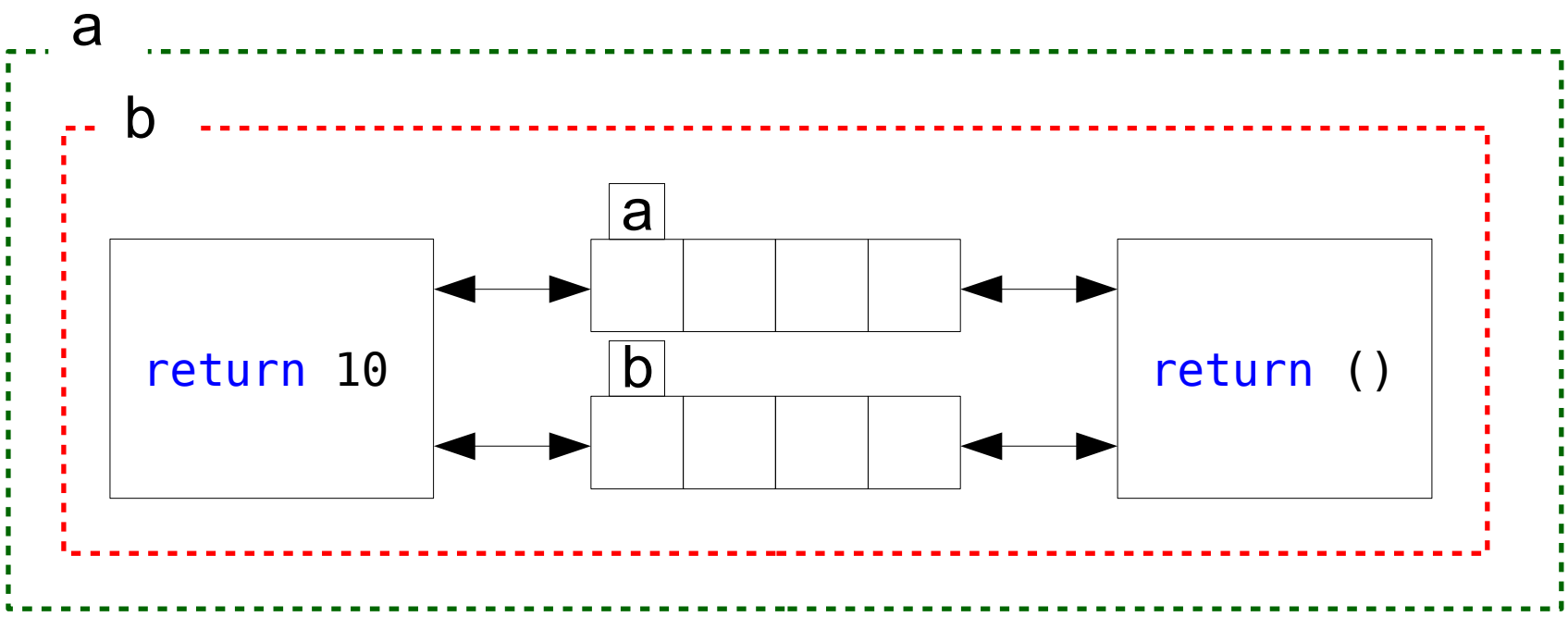
λ_{ch} Semantics, by Example



λ_{ch} Semantics, by Example



λ_{ch} Semantics, by Example



λ_{act}: A concurrent λ-calculus for type-parameterised actors

Syntax

$$
\begin{array}{rlll}
\text{Types} & A, B, C & ::= & \mathbf{1} \mid A \to^{C} B \mid \mathsf{ActorRef}(A) \\
\text{Variables} & \alpha & ::= & x \mid a \\
\text{Values} & V, W & ::= & \alpha \mid \lambda x.M \mid () \\
\text{Computations} & L, M, N & ::= & V\,W \\
 & & \mid & \mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N \mid \mathsf{return}\ V \\
 & & \mid & \mathsf{spawn}\ M \mid \mathsf{send}\ V\,W \mid \mathsf{receive} \mid \mathsf{self}
\end{array}
$$

Note: receive takes no arguments!

λ_{act} Typing Rules

As we are considering type-parameterised actors, each actor process has a typed mailbox. Therefore our term typing judgement becomes:

$$\boxed{\Gamma \mid B \vdash M : A}$$

"Under environment Γ and with the ability to receive messages of type *B*, term *M* has type *A*".

λ_{act} Typing Rules

Adapting abstraction and application

Abs

$$\frac{\Gamma, x : A \mid C \vdash M : B}{\Gamma \vdash \lambda x.M : A \to^{C} B}$$

App

$$\frac{\Gamma \vdash V : A \to^{C} B \quad \Gamma \vdash W : A}{\Gamma \mid C \vdash V\,W : B}$$

Communication and Concurrency Primitives

Send

$$\frac{\Gamma \vdash V : A \quad \Gamma \vdash W : \mathsf{ActorRef}(A)}{\Gamma \mid B \vdash \mathsf{send}\ V\,W : \mathbf{1}}$$

Recv

$$\frac{}{\Gamma \mid A \vdash \mathsf{receive} : A}$$

Spawn

$$\frac{\Gamma \mid A \vdash M : \mathbf{1}}{\Gamma \mid B \vdash \mathsf{spawn}\ M : \mathsf{ActorRef}(A)}$$

Self

$$\frac{}{\Gamma \mid A \vdash \mathsf{self} : \mathsf{ActorRef}(A)}$$

λ_{act} Configurations

$$\boxed{\mathcal{C} ::= \mathcal{C}_1 \parallel \mathcal{C}_2 \mid (\nu a)\mathcal{C} \mid \langle a, N, \overrightarrow{V} \rangle}$$

Parallel composition and name restriction are as before. No separate buffers and terms!

$$\boxed{\langle a, N, \overrightarrow{V} \rangle}$$

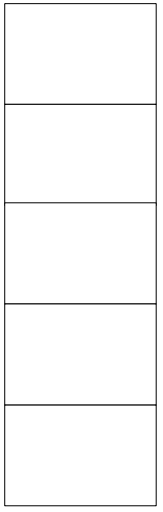
Actor with name a, evaluating term N, with mailbox $\overrightarrow{V}$. Terms must be evaluated in an actor context!

λ_{act} Semantics, by Example

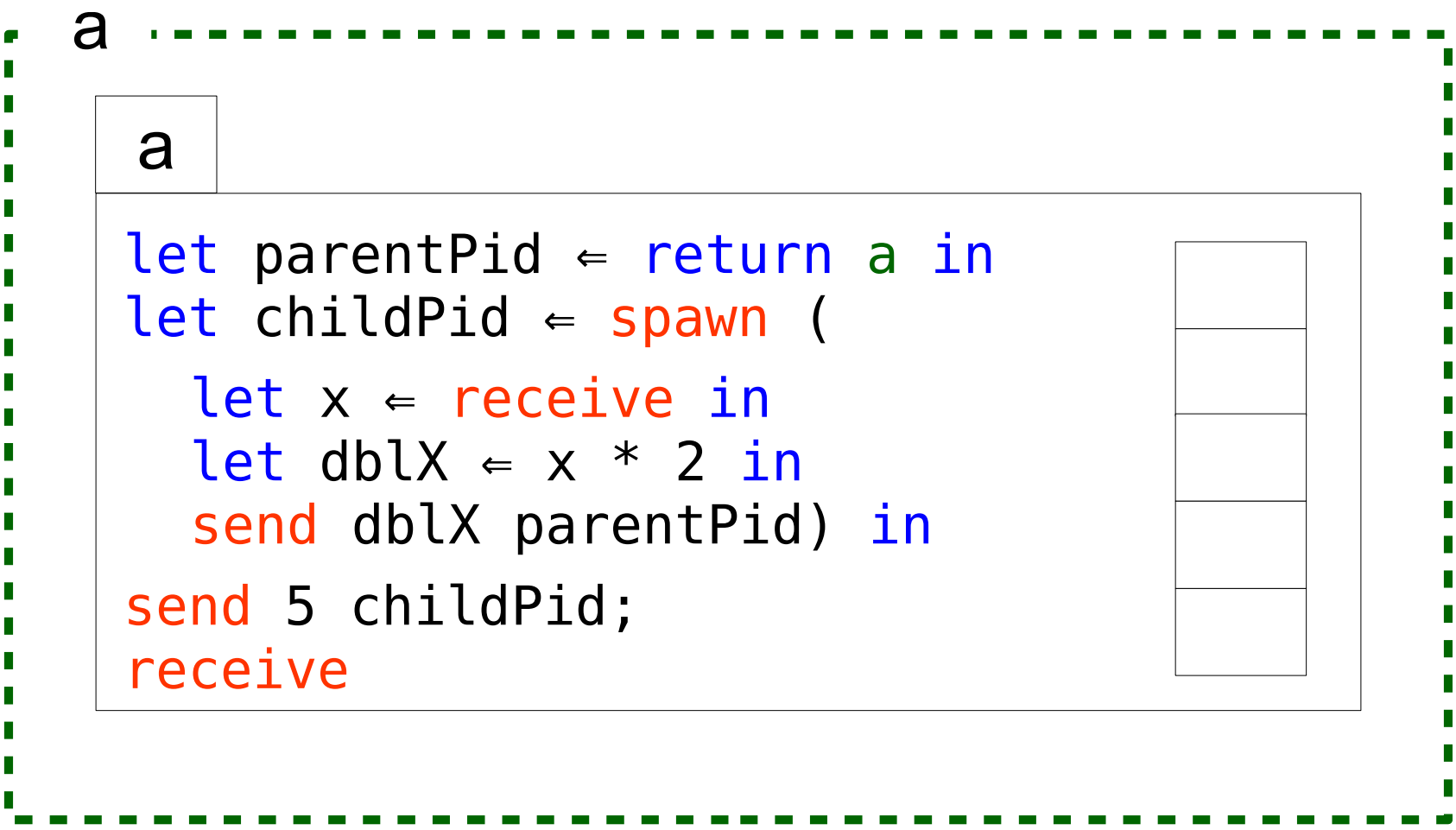
a

a

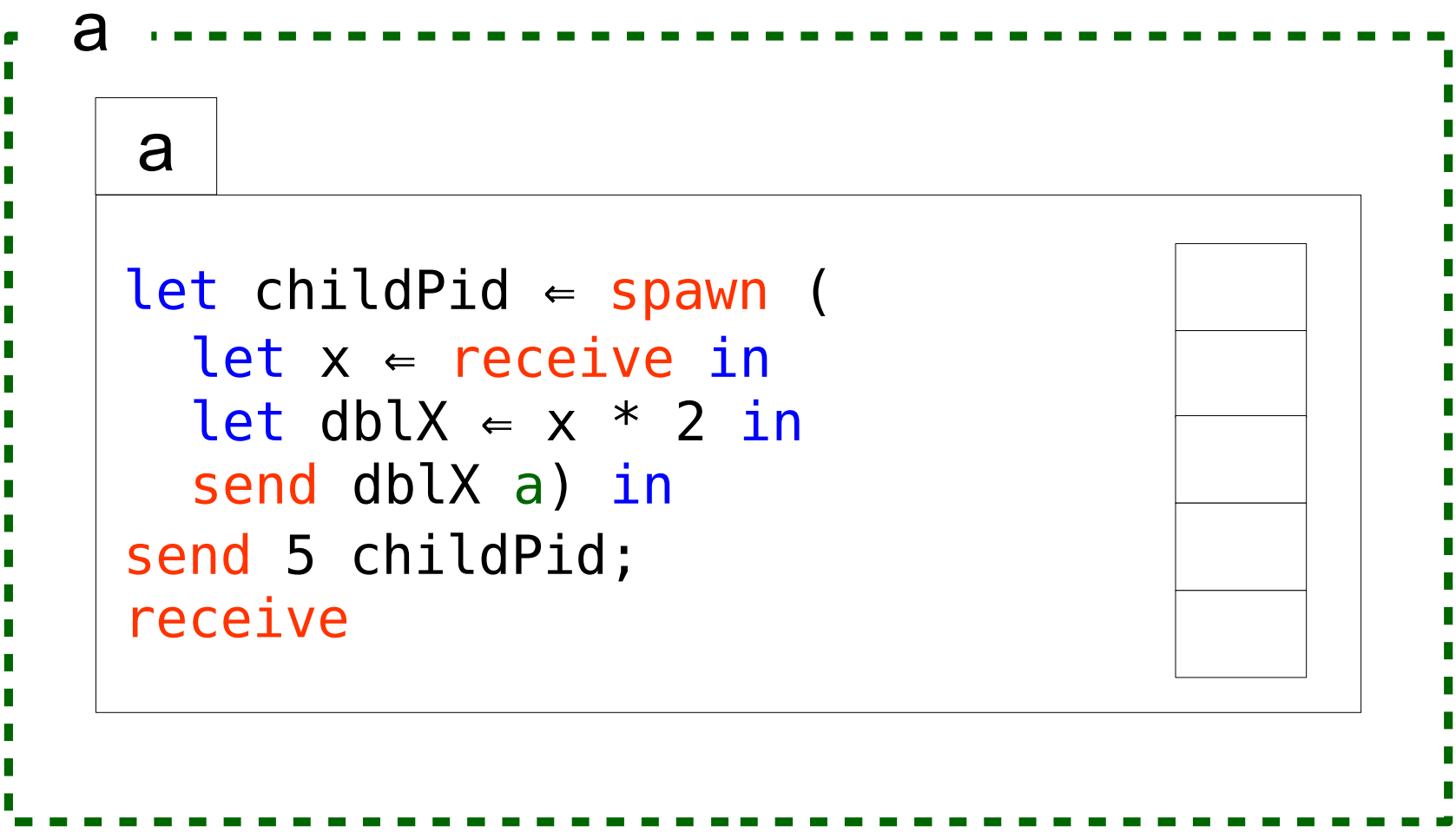
```
let parentPid ⇐ self in
let childPid ⇐ spawn (
  let x ⇐ receive in
  let dblX ⇐ x * 2 in
  send dblX parentPid) in
send 5 childPid;
receive
```



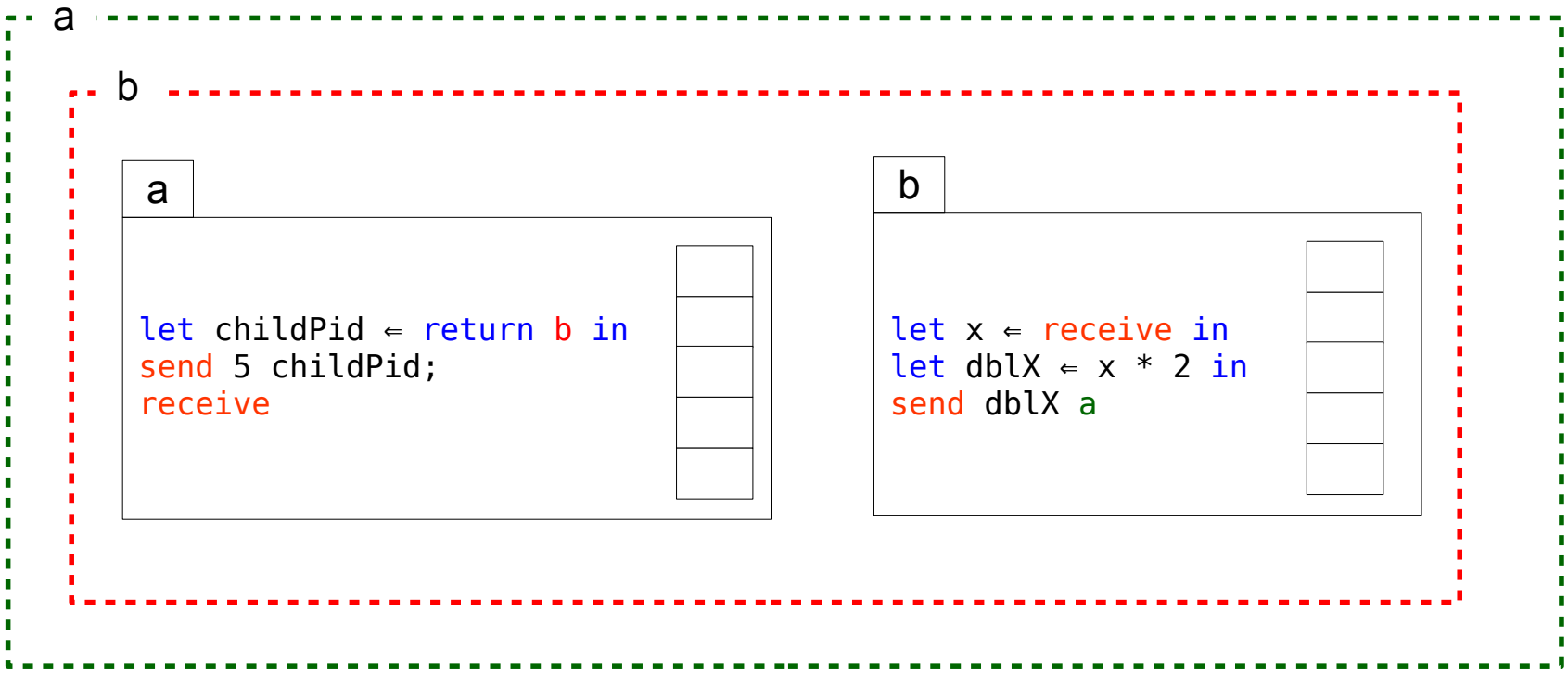
λ_{act} Semantics, by Example



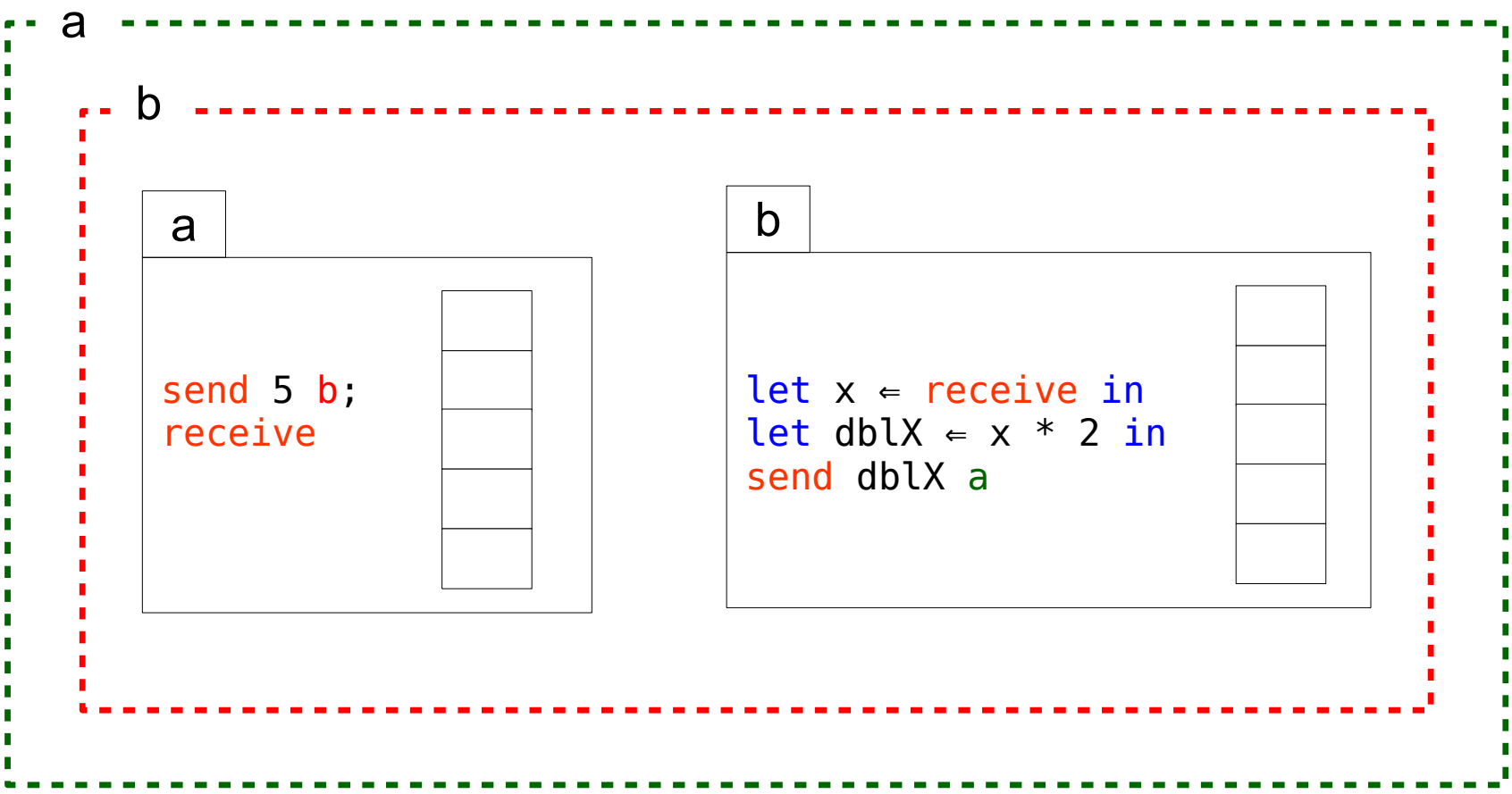
λ_{act} Semantics, by Example



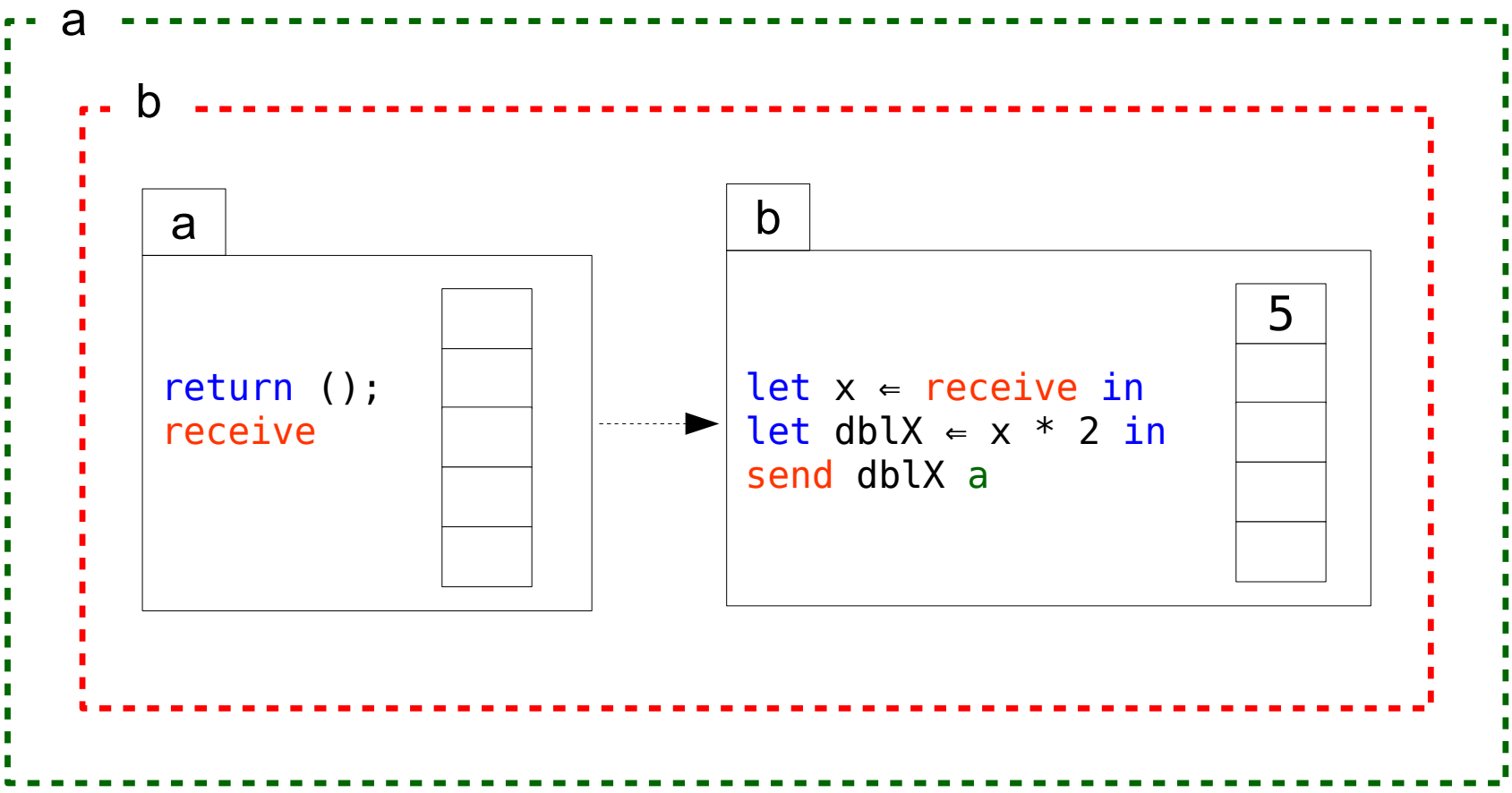
λ_{act} Semantics, by Example



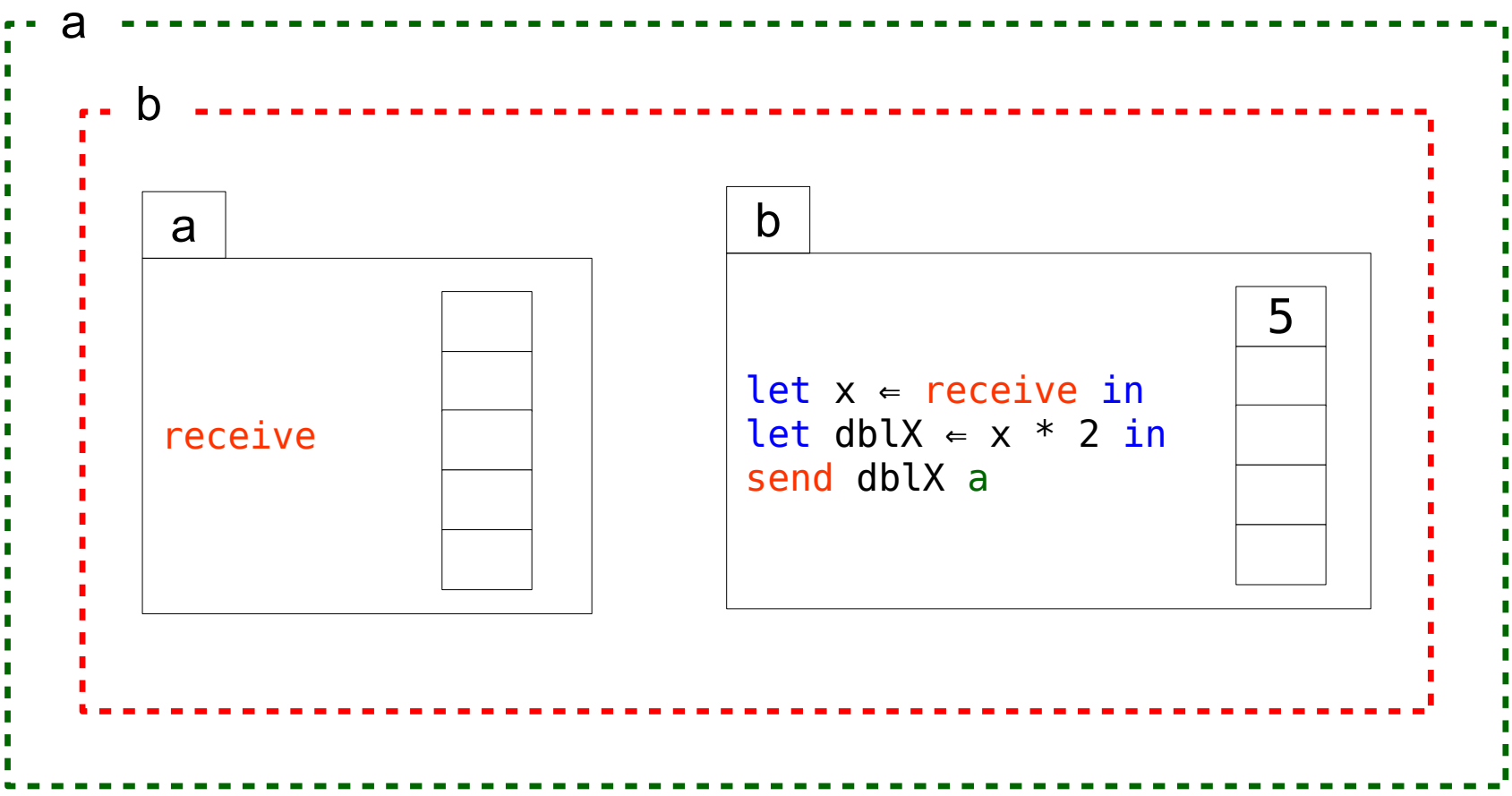
λ_{act} Semantics, by Example



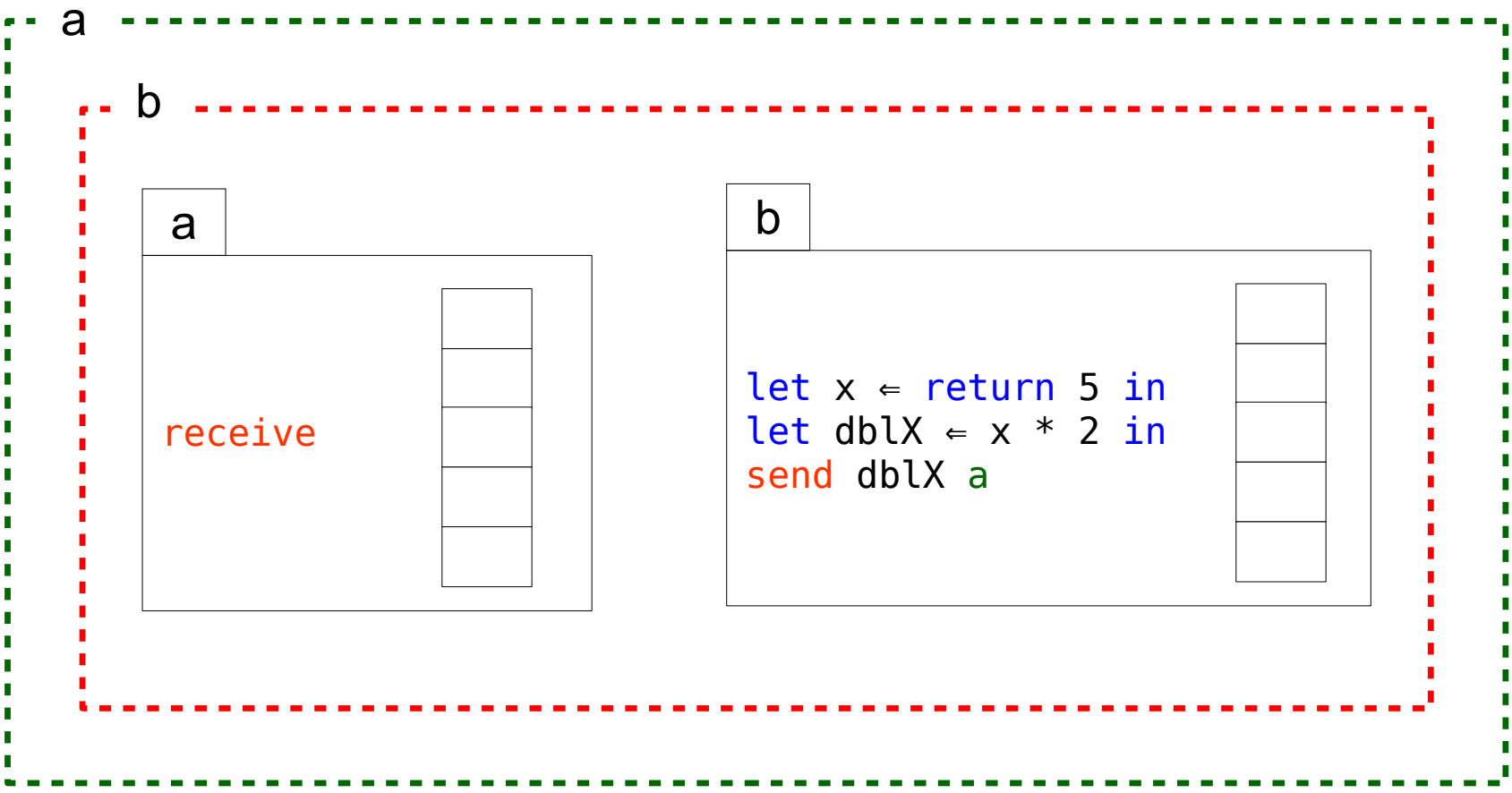
λ_{act} Semantics, by Example



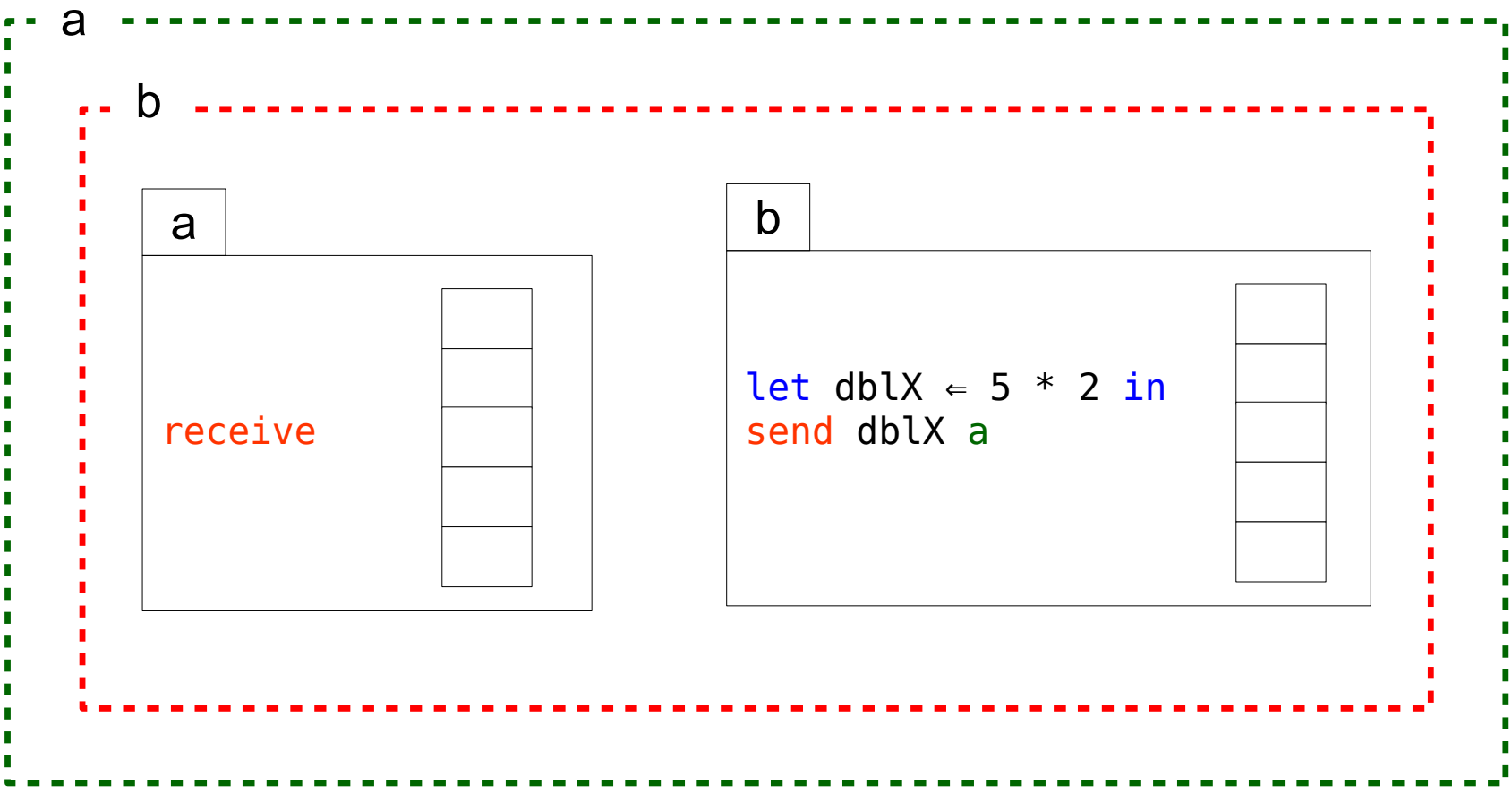
λ_{act} Semantics, by Example



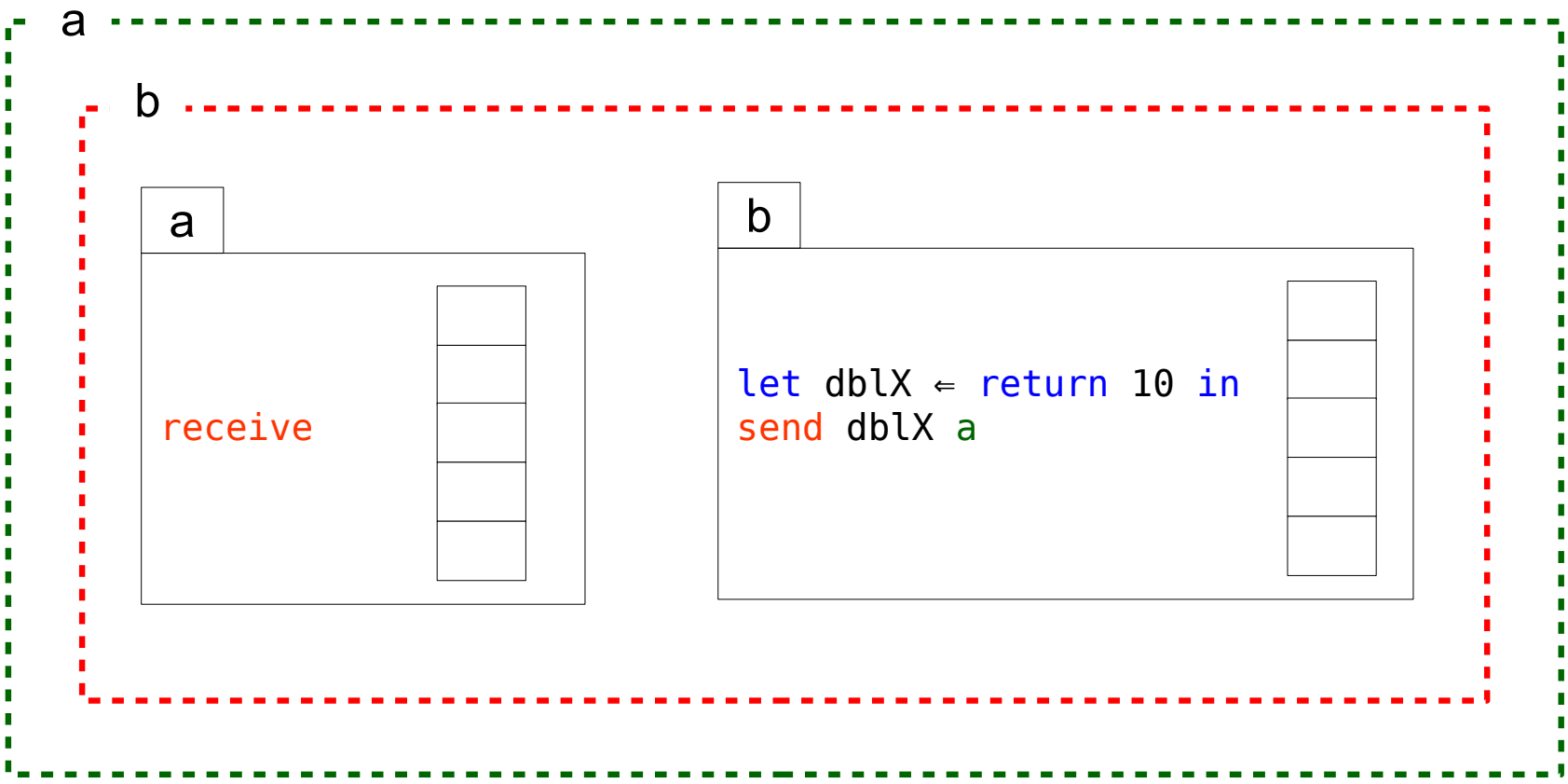
λ_{act} Semantics, by Example



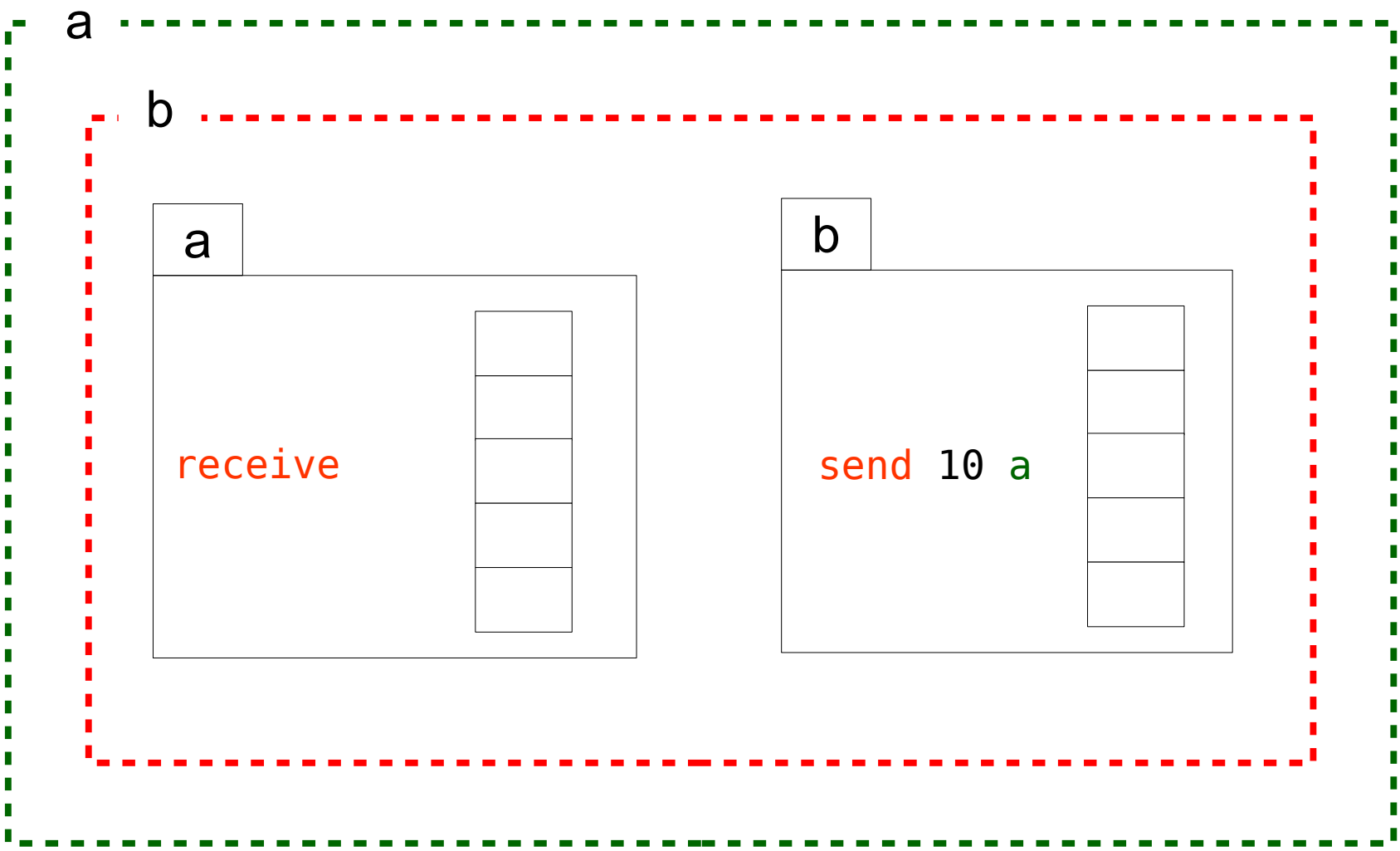
λ_{act} Semantics, by Example



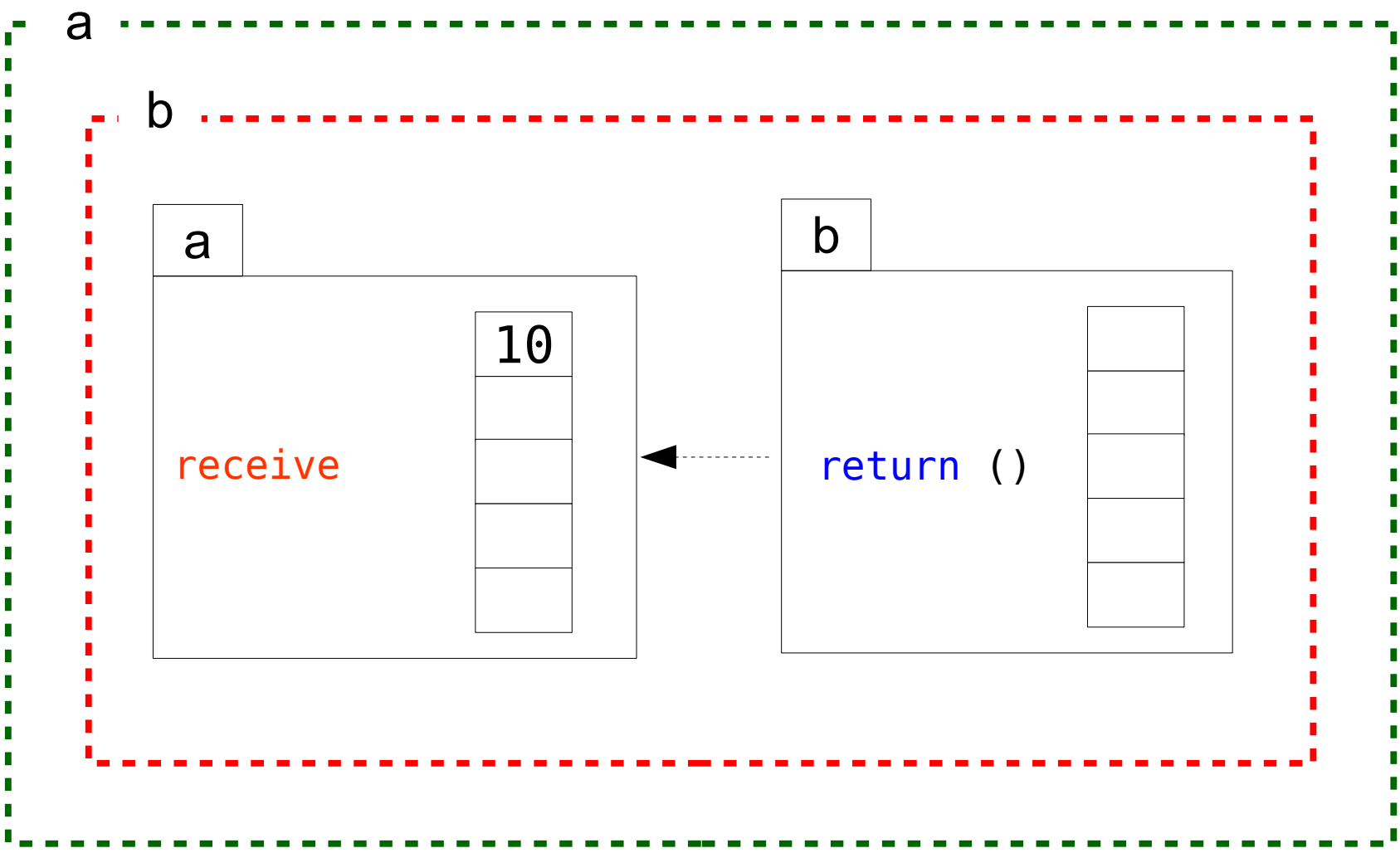
λ_{act} Semantics, by Example



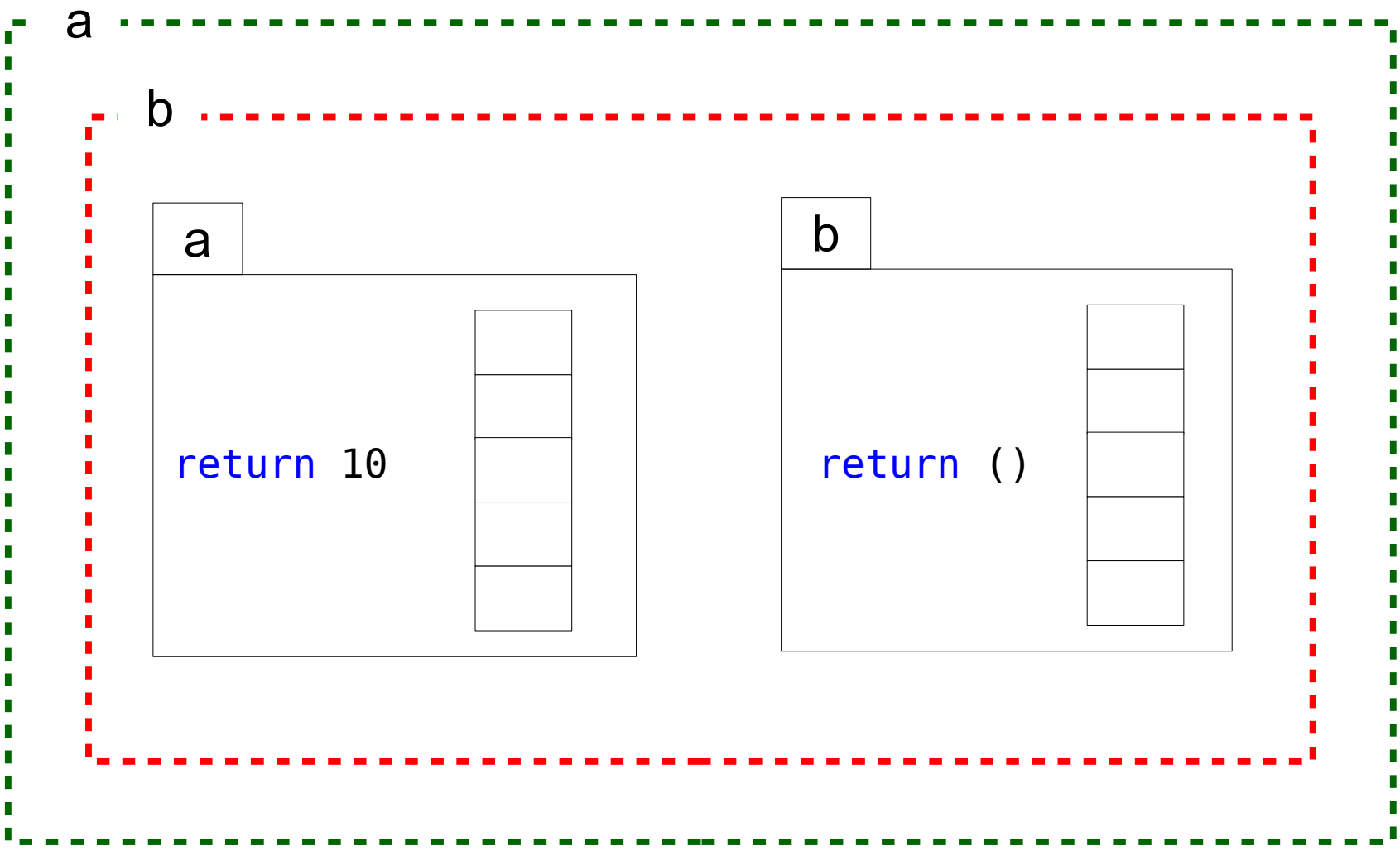
λ_{act} Semantics, by Example



λ_{act} Semantics, by Example

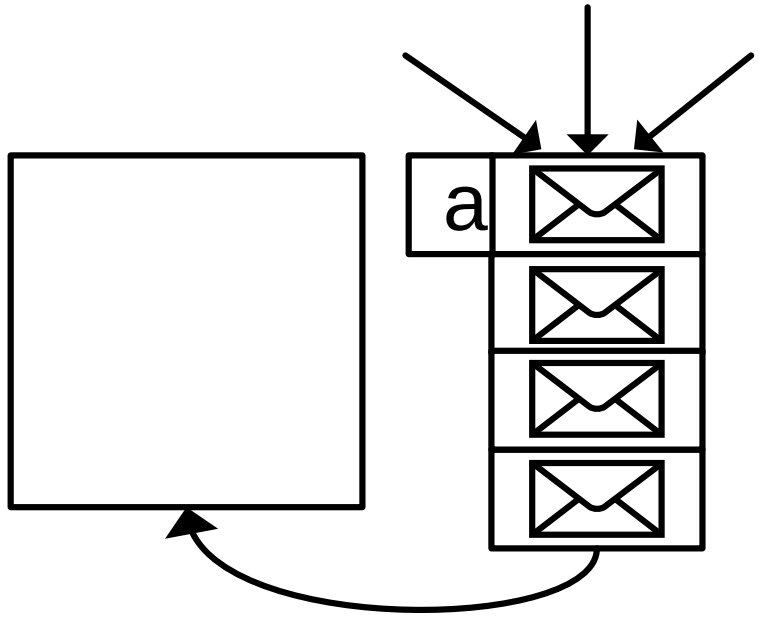


λ_{act} Semantics, by Example



From λ_{act} to λ_{ch}

The Idea



Main idea: emulate the mailbox using a channel, which is ‘pinned’ to a process.

The Translation, Formally (1)

Main technical idea: thread the 'mailbox channel' through the translation on terms.

Translation on Types $\boxed{[\![A]\!]}$

$$
\begin{aligned}
[\![\mathsf{ActorRef}(A)]\!] &= \mathsf{ChanRef}([\![A]\!]) \\
[\![\mathbf{1}]\!] &= \mathbf{1} \\
[\![A \to^{C} B]\!] &= [\![A]\!] \to \mathsf{ChanRef}([\![C]\!]) \to [\![B]\!]
\end{aligned}
$$

Translation on Values $\boxed{[\![V]\!]}$

$$[\![x]\!] = x \qquad [\![a]\!] = a \qquad [\![\lambda x.M]\!] = \lambda x.\lambda ch.([\![M]\!]\ ch) \qquad [\![()]\!] = ()$$

The Translation, Formally (2)

Translation on communication and concurrency primitives $\boxed{[\![M]\!]\ x}$

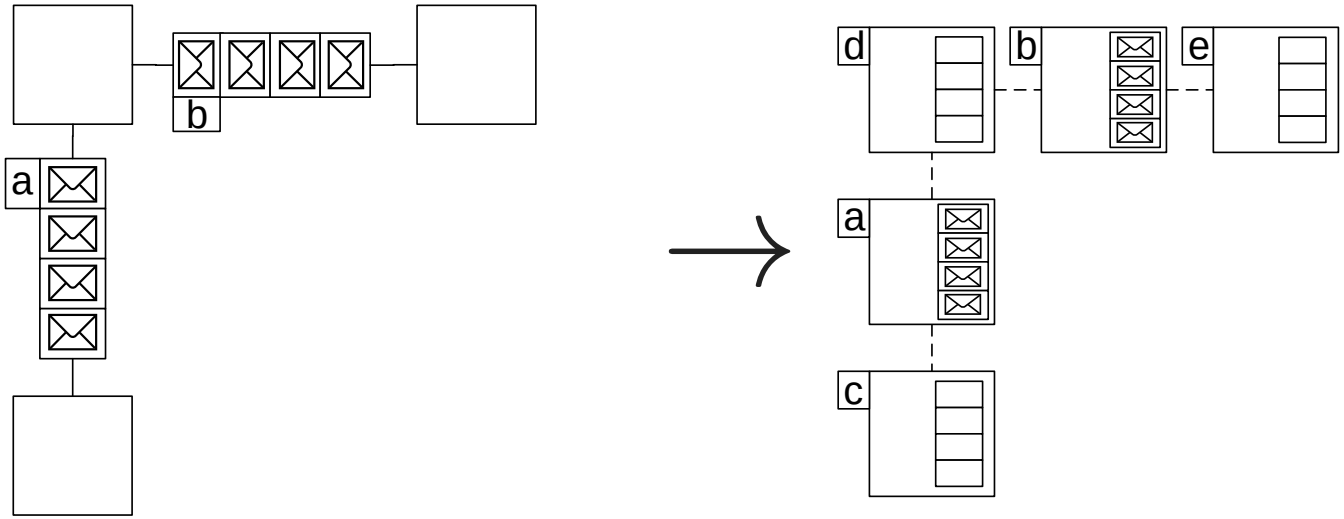
$$
\begin{array}{rcl}
[\![\mathsf{self}]\!]\ ch & = & \mathsf{return}\ ch \\
[\![\mathsf{receive}]\!]\ ch & = & \mathsf{take}\ ch \\
[\![\mathsf{spawn}\ M]\!]\ ch & = & \mathsf{let}\ chMb \Leftarrow \mathsf{newCh}\ \mathsf{in} \\
 & & \mathsf{fork}\ ([\![M]\!]\ chMb); \\
 & & \mathsf{return}\ chMb \\
[\![\mathsf{send}\ V\ W]\!]\ ch & = & \mathsf{give}\ ([\![V]\!])\ ([\![W]\!])
\end{array}
$$

Translation on configurations $\boxed{[\![\mathcal{C}]\!]}$

$$
\begin{array}{rcl}
[\![\mathcal{C}_1 \parallel \mathcal{C}_2]\!] & = & [\![\mathcal{C}_1]\!] \parallel [\![\mathcal{C}_2]\!] \\
[\![(\nu a)\mathcal{C}]\!] & = & (\nu a)\,[\![\mathcal{C}]\!] \\
[\![\langle a, M, \overrightarrow{V} \rangle]\!] & = & a([\![\overrightarrow{V}]\!]) \parallel ([\![M]\!]\ a)
\end{array}
$$

From λ_{ch} to λ_{act}

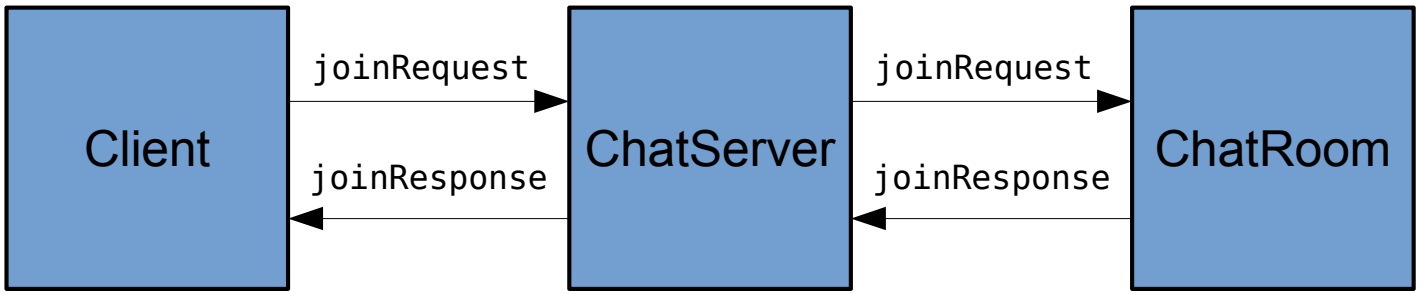
The Idea



Emulate each channel using an actor: can receive requests to queue or dequeue values. Must do this to preserve ability to pass channel endpoints!

Type Pollution Rears its Ugly Head...

Problem with typed actors: type pollution! Type of an ActorRef must handle all types that the actor can receive.



ChatServer reference has type
ActorRef(`joinRequest` + `joinResponse`)

Dealing with type pollution

Type pollution is problematic for the translation—which types are given to mailboxes of actors emulating threads?

- → Require all channels in the system to have the same type
- → If not, perform coalescing translation to assign general enough type to all mailboxes

Can then write judgements of the form $\{B\}\ \Gamma \vdash M : A$

- → Under environment Γ, where all channels carry values of type B, M has type A.

Translation (1)

body is an event loop which receives requests to either store a value, or dequeue a value and send it back to a given PID.

Translation on types $\boxed{(\!|\, A \,|\!)\ B}$

$$
\begin{aligned}
(\!|\, \mathsf{Chan}\, |\!)\ C &= \mathsf{ActorRef}((\!|\, C \,|\!)\ C + \mathsf{ActorRef}((\!|\, C \,|\!)\ C)) \\
(\!|\, A \to B \,|\!)\ C &= ((\!|\, A \,|\!)\ C) \to^{(\!|\, C \,|\!)\ C} ((\!|\, B \,|\!)\ C)
\end{aligned}
$$

Translation on communication and concurrency primitives $\boxed{(\!|\, M \,|\!)}$

$$
\begin{aligned}
(\!|\, \mathsf{fork}\ M \,|\!) &= \mathsf{let}\ x \Leftarrow \mathsf{spawn}\ (\!|\, M \,|\!)\ \mathsf{in}\ \mathsf{return}\ () \\
(\!|\, \mathsf{give}\ V\ W \,|\!) &= \mathsf{send}\ (\mathsf{inl}\ (\!|\, V \,|\!))\ (\!|\, W \,|\!) \\
(\!|\, \mathsf{take}\ V \,|\!) &= \mathsf{let}\ \mathit{selfPid} \Leftarrow \mathsf{self}\ \mathsf{in} \\
&\quad\ \mathsf{send}\ (\mathsf{inr}\ \mathit{selfPid})\ (\!|\, V \,|\!); \\
&\quad\ \mathsf{receive} \\
(\!|\, \mathsf{newCh} \,|\!) &= \mathsf{spawn}\ (\mathsf{body}\ (\mathbf{[\,]}, \mathbf{[\,]}))
\end{aligned}
$$

Translation (2)

Translation on configurations

$(\!| \mathcal{C} |\!)$

$$
\begin{array}{rcll}
(\!| \mathcal{C}_1 \parallel \mathcal{C}_2 |\!) & = & (\!| \mathcal{C}_1 |\!) \parallel (\!| \mathcal{C}_2 |\!) & \\
(\!| (\nu a)\mathcal{C} |\!) & = & (\nu a)(\!| \mathcal{C} |\!) & \\
(\!| M |\!) & = & (\nu a)(\langle a, (\!| M |\!), \epsilon \rangle) & a \text{ is a fresh name} \\
(\!| a(\overrightarrow{V}) |\!) & = & \langle a, \text{body}\ ((\!| \overrightarrow{V} |\!), \mathbf{[]}), \epsilon \rangle & \\
 & & \text{where } (\!| \overrightarrow{V} |\!) = (\!| V_0 |\!) :: \ldots :: (\!| V_n |\!) :: \mathbf{[]} &
\end{array}
$$

Wrapping Up

Parting Thoughts

- → Two small models for languages based on channels, and languages based on actors, and translations between them.
- → Also: extensions for λ_{act} with synchronous calls (sidesteps type pollution); λ_{ch} with nondeterministic choice
  - → Relating channels with choice to actors is still a challenging open problem
- → Given many insights into behaviourally-typed versions – that's up next!

Thanks!