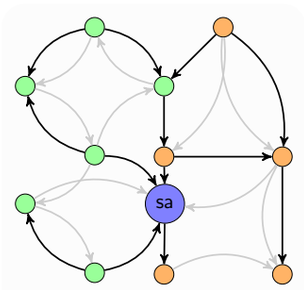
Actors without Borders:

Amnesty for Imprisoned State

Elias Castegren, Tobias Wrigstad

PLACES'17, Uppsala

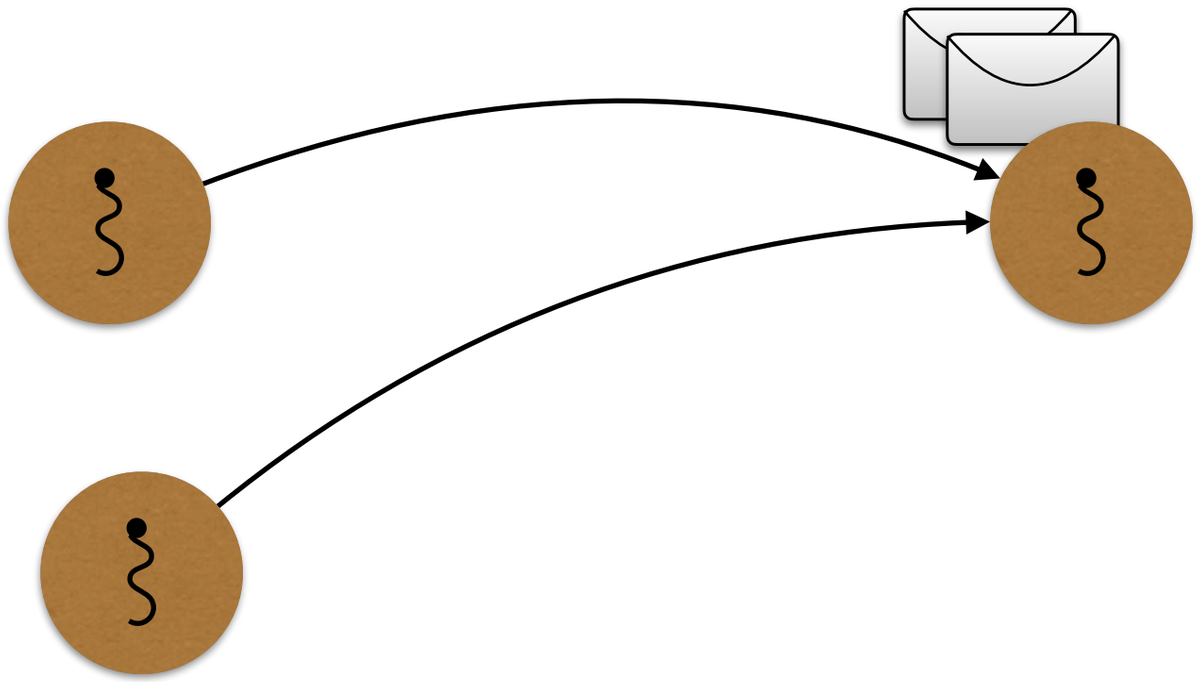
UPMARC



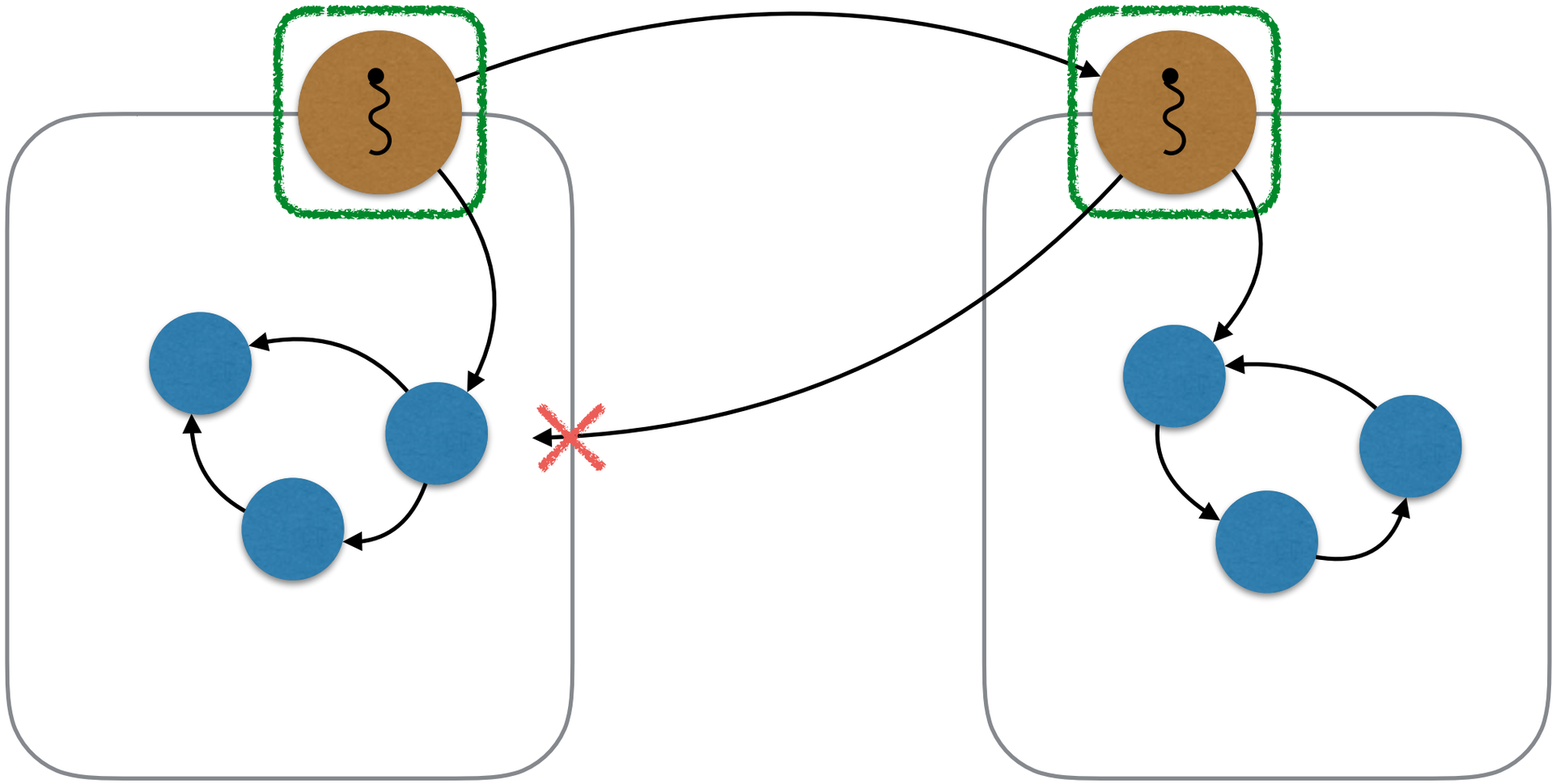
Structured Aliasing

UPSCALE

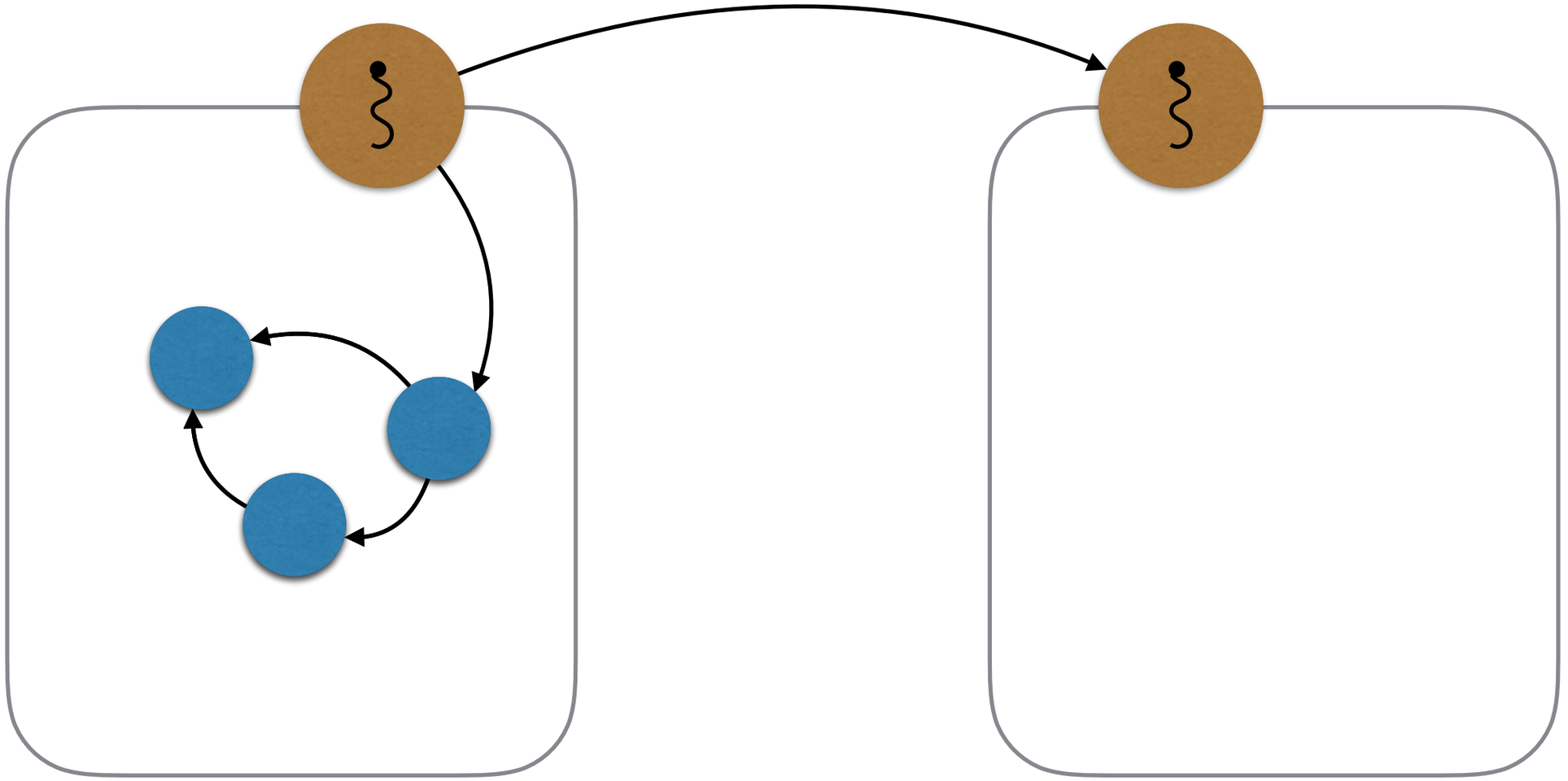
The Actor Model



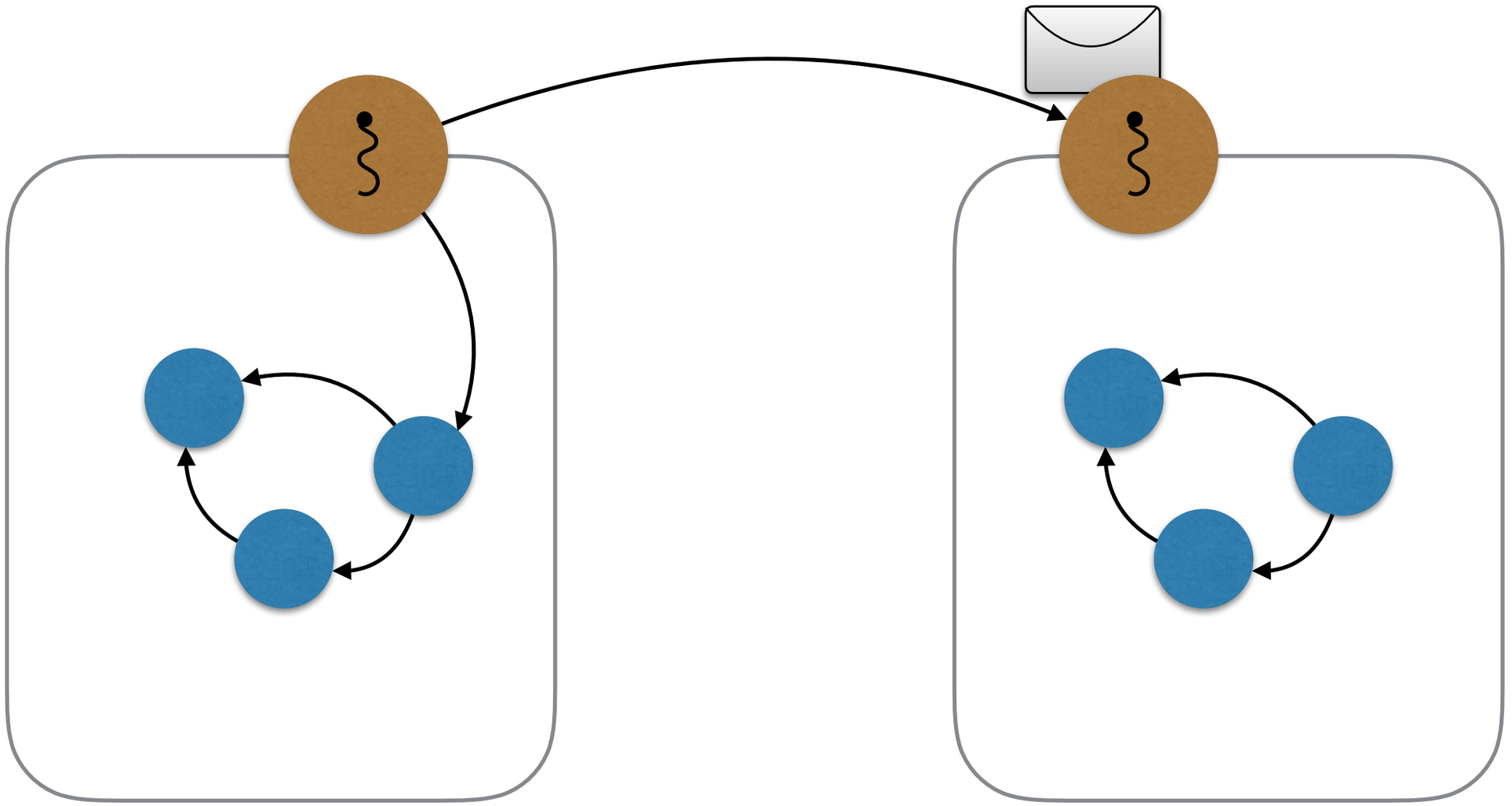
Actor Isolation allows Sequential Reasoning



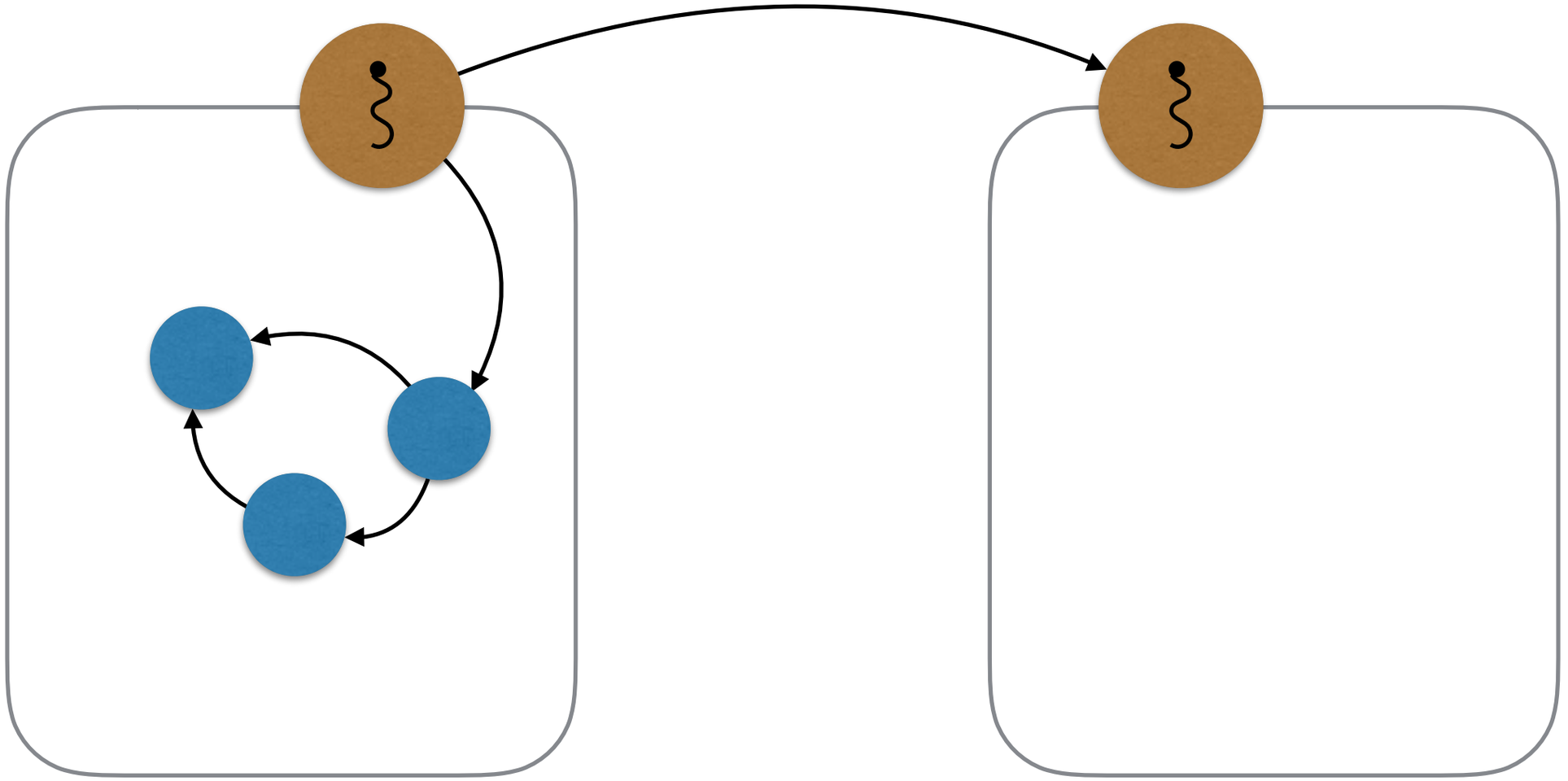
Passing Data Between Actors (by copy)



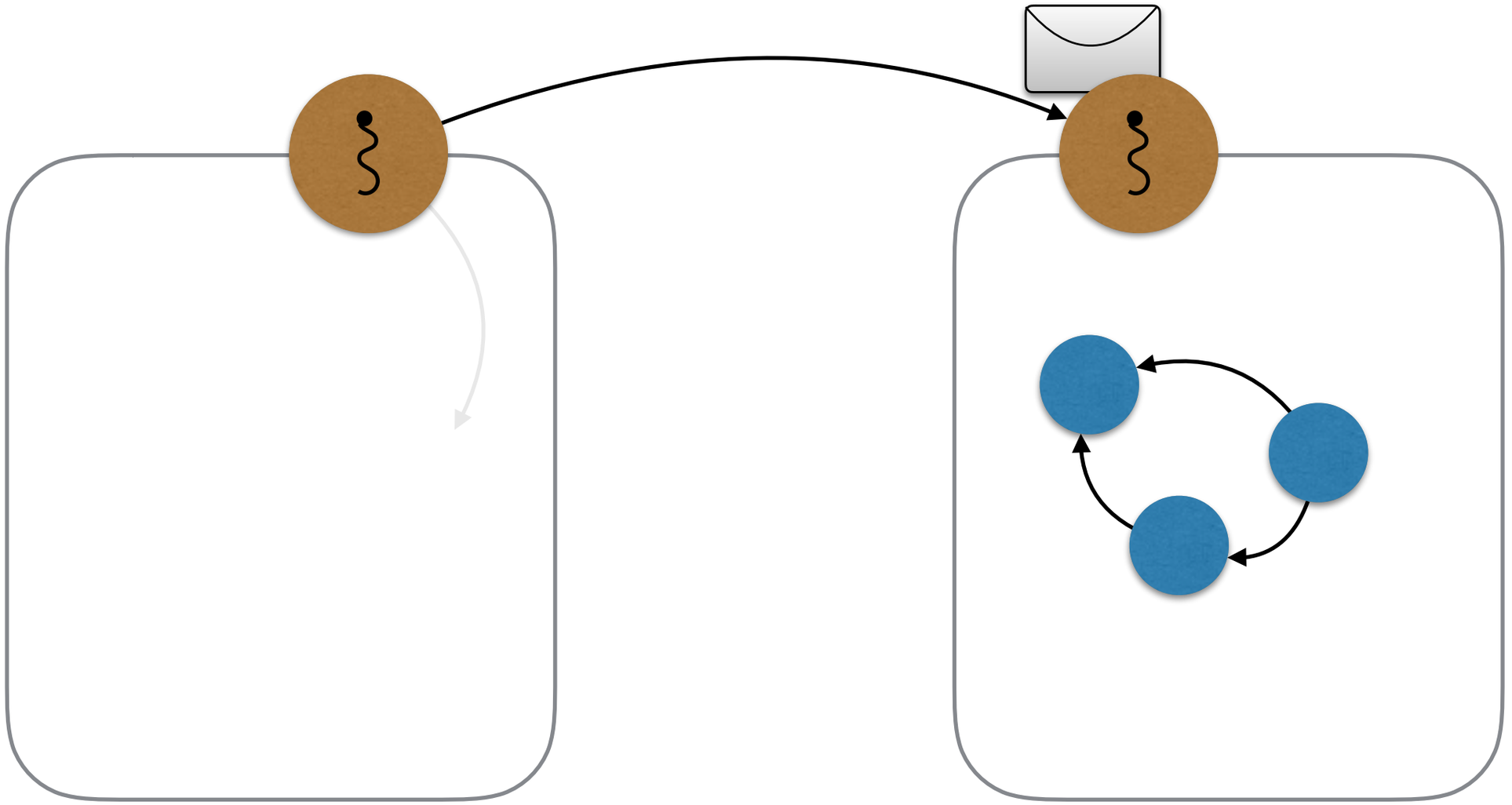
Passing Data Between Actors (by copy)



Passing Data Between Actors (by transfer)

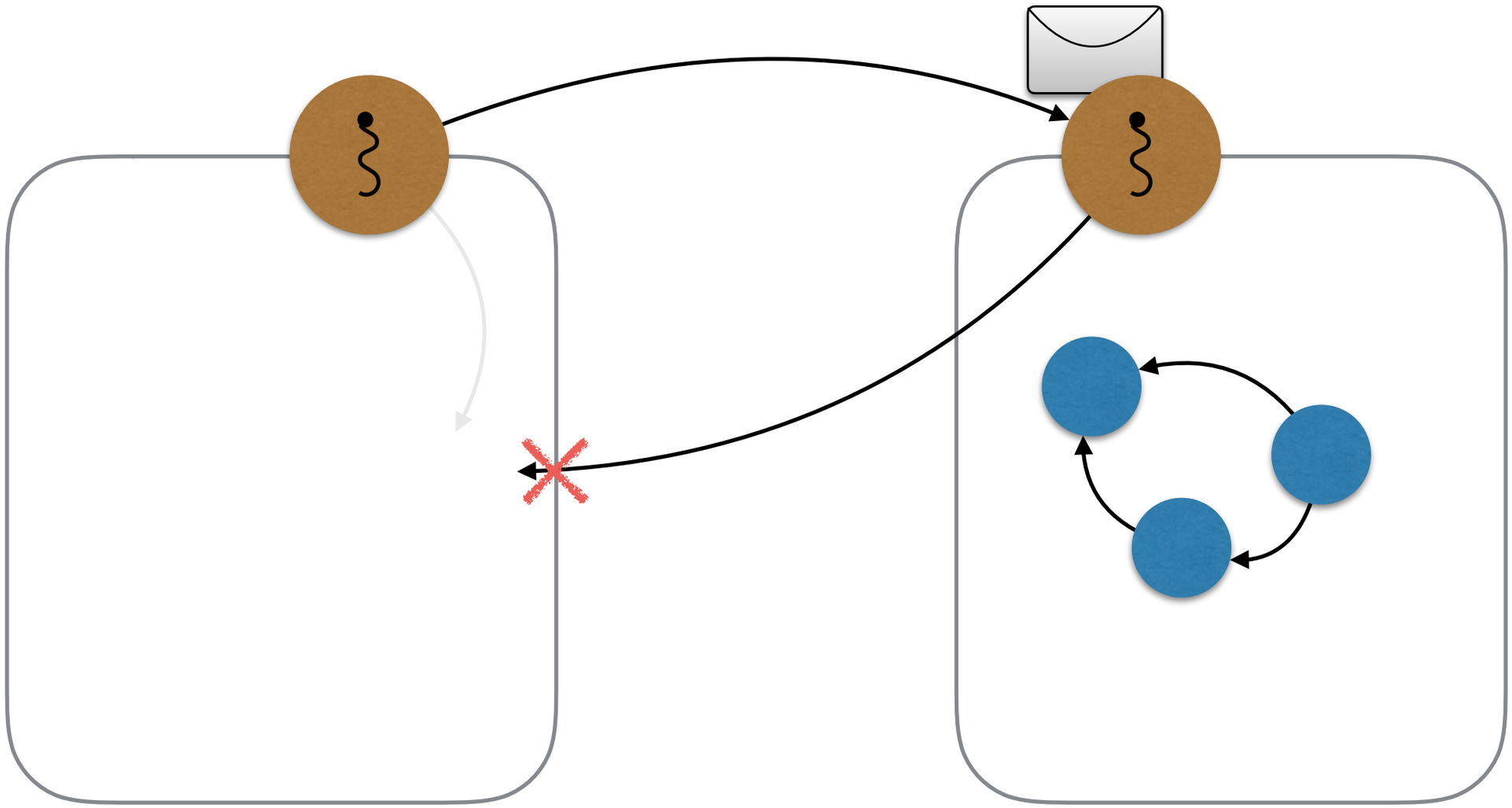


Passing Data Between Actors (by transfer)

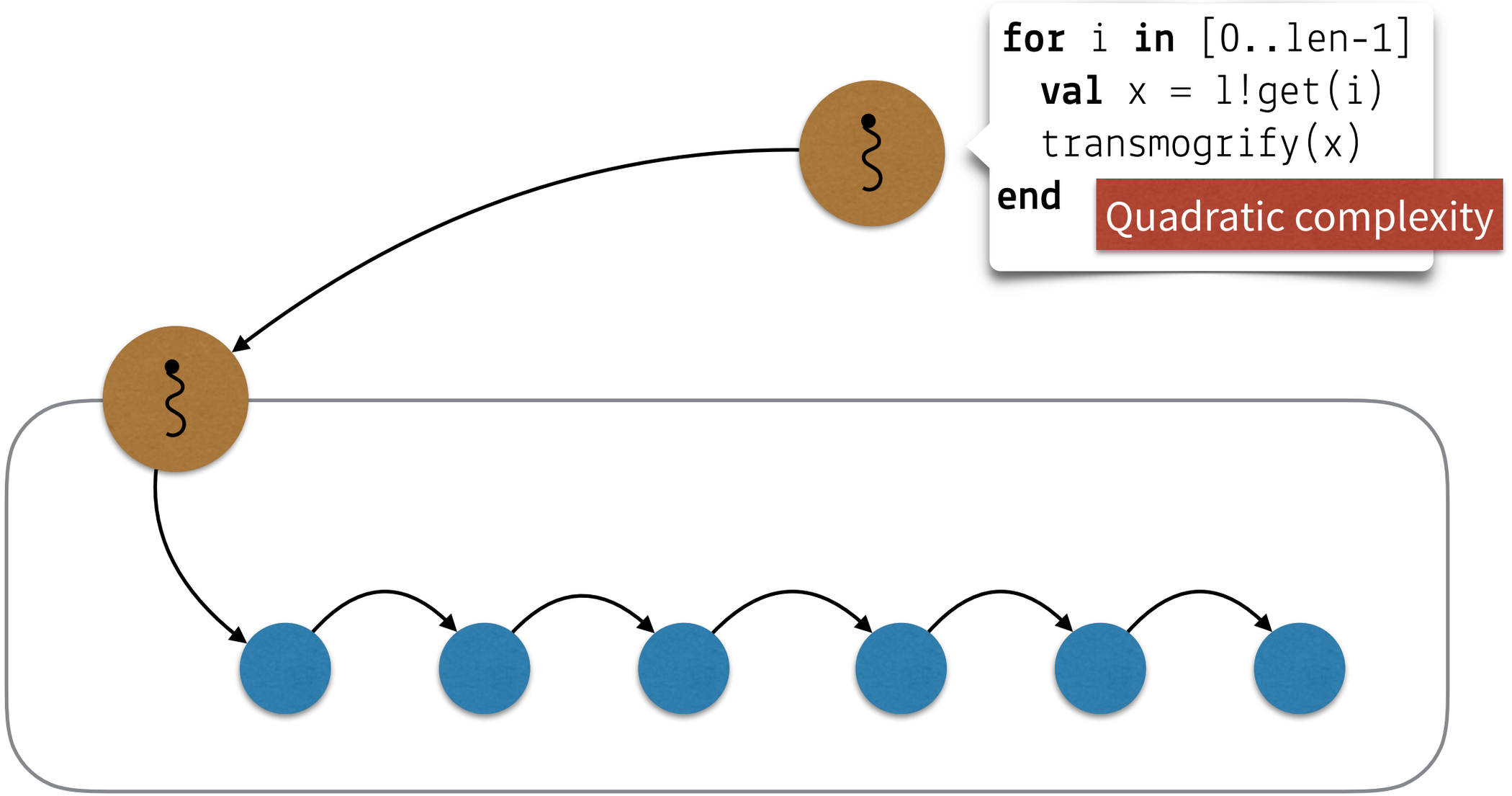


Passing Data Between Actors (by transfer)

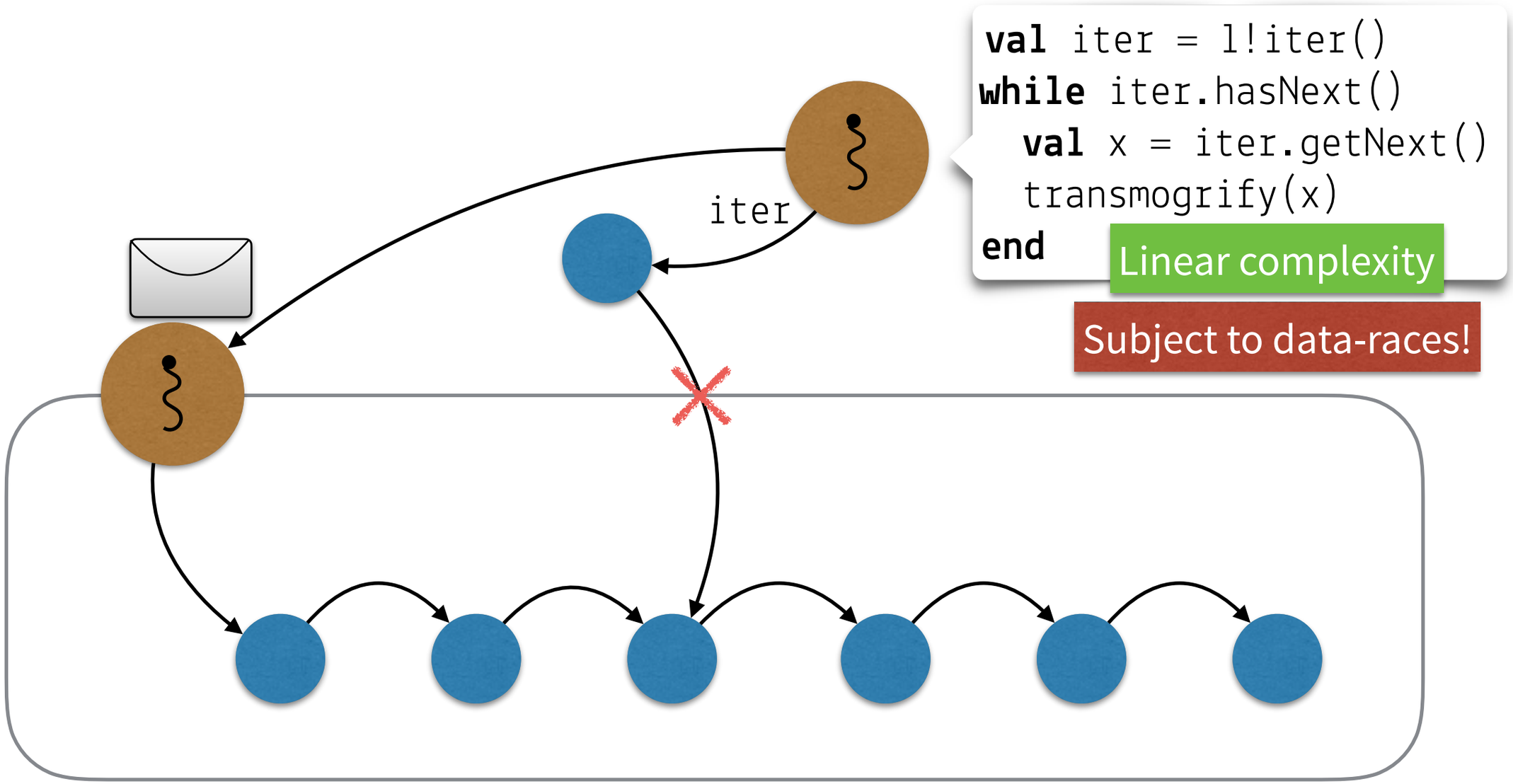
Strong Encapsulation



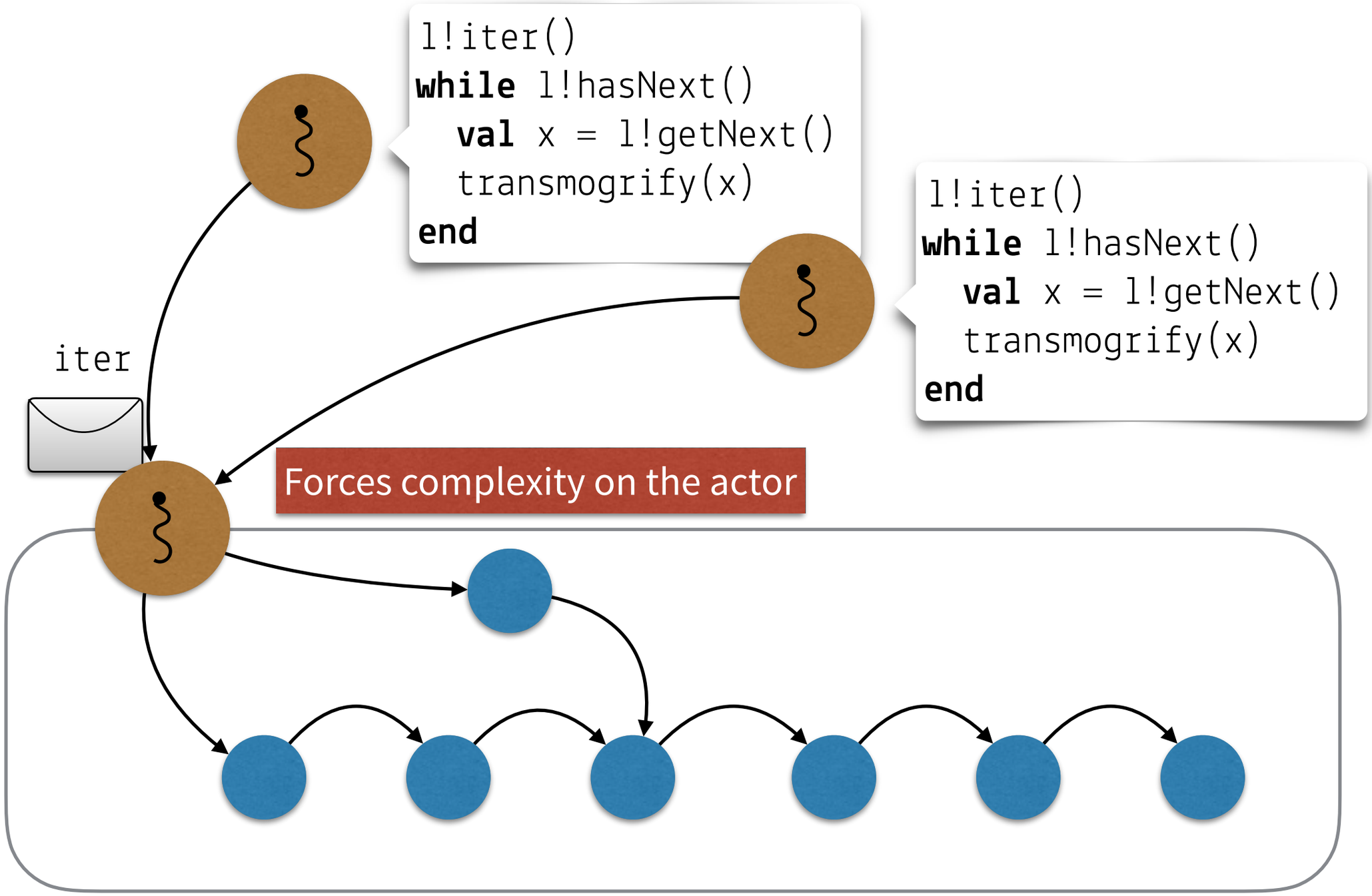
(Too) Strong Encapsulation



Solution? Breaking Actor Isolation



Solution? Keeping Complexity Isolated



Problem Overview

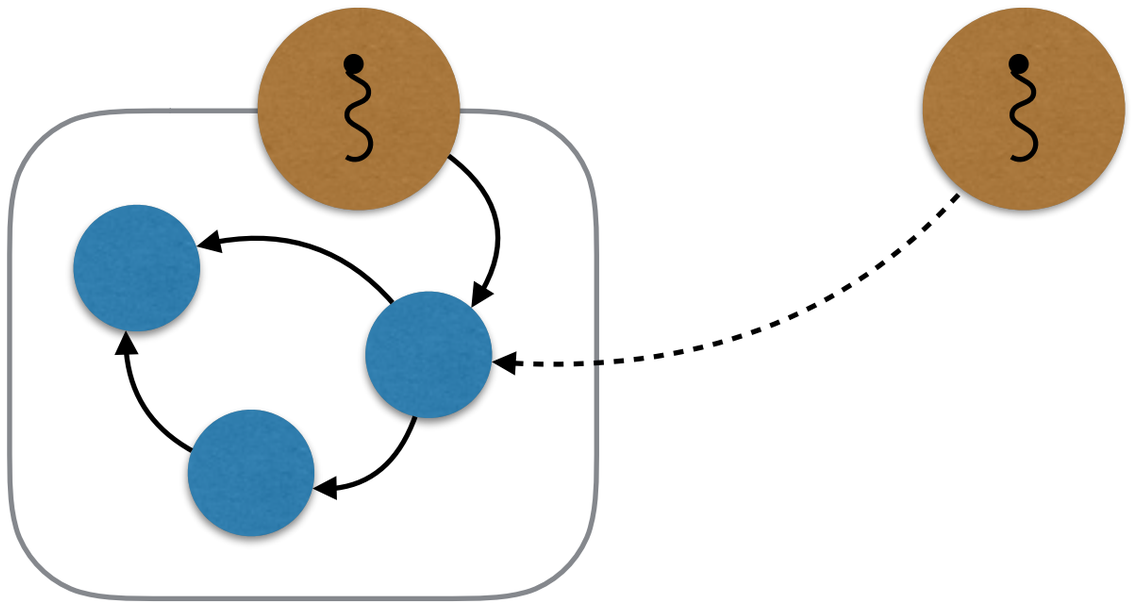
- Actors rely on isolation for sequential reasoning

  - Too strong for certain patterns

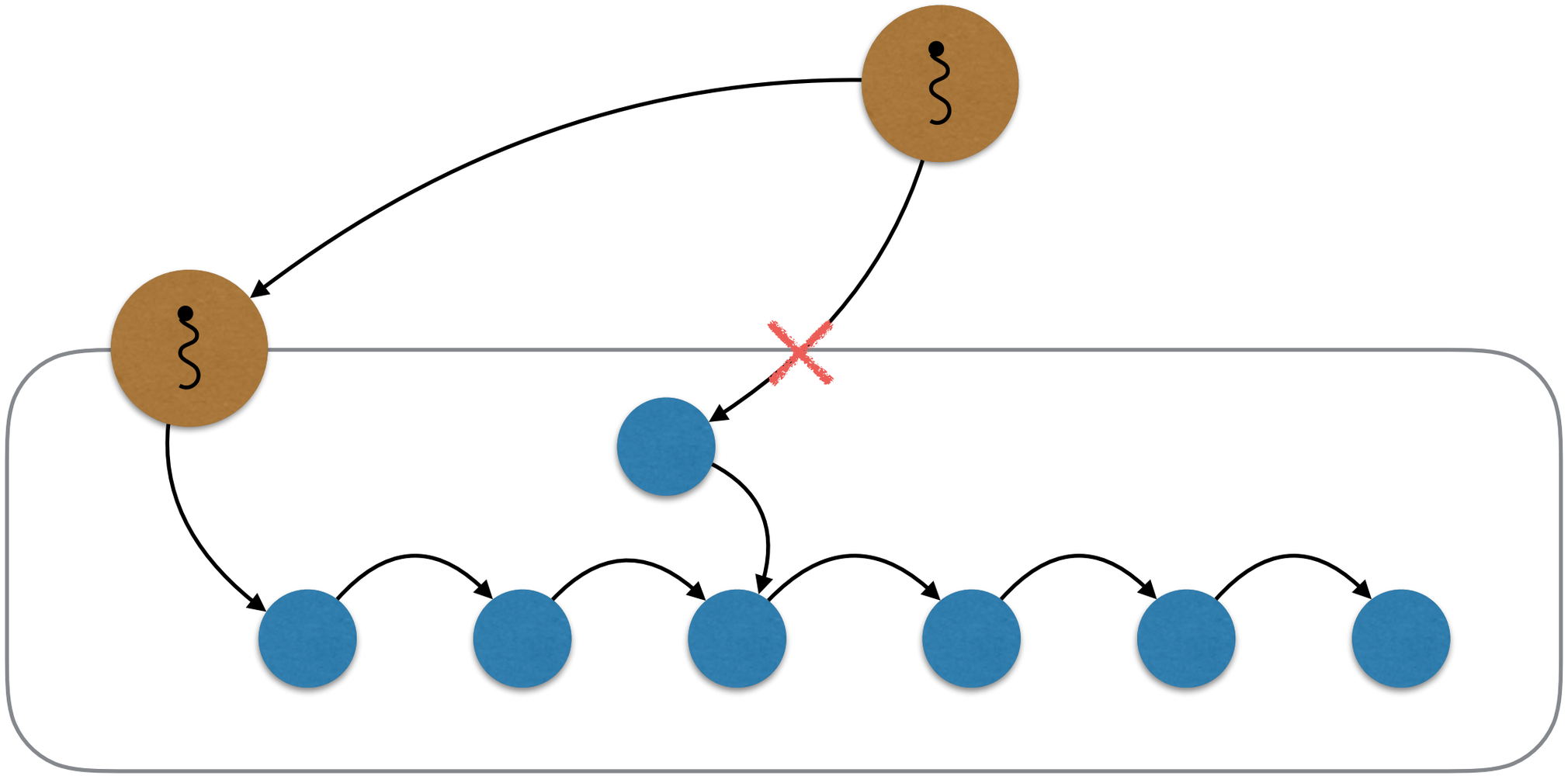
- Breaking actor isolation prevents sequential reasoning

- Extending actors to handle these patterns make them complex

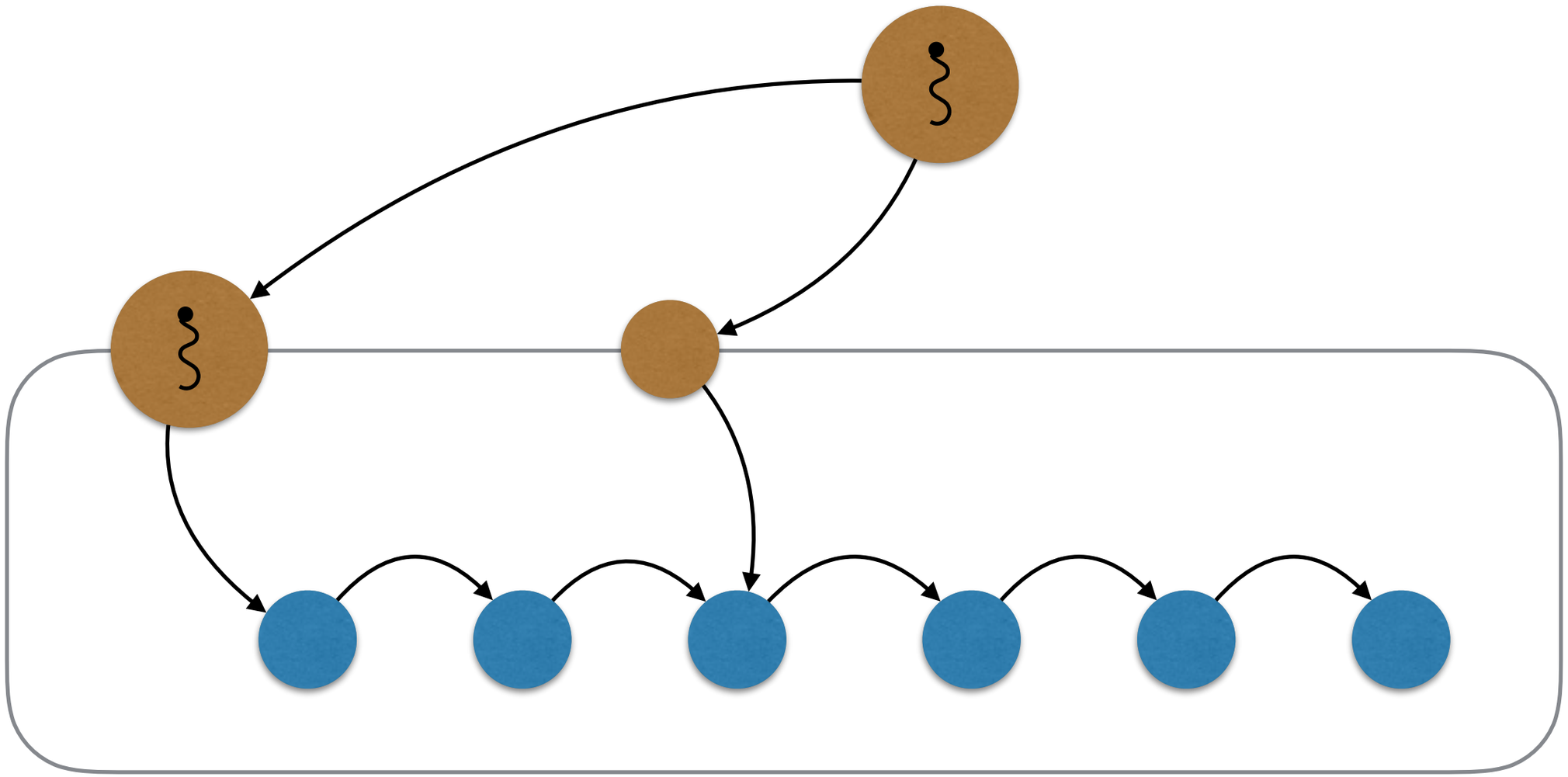
- **Observation:**
  Holding a reference to an isolated object is OK as long as only the owner accesses it



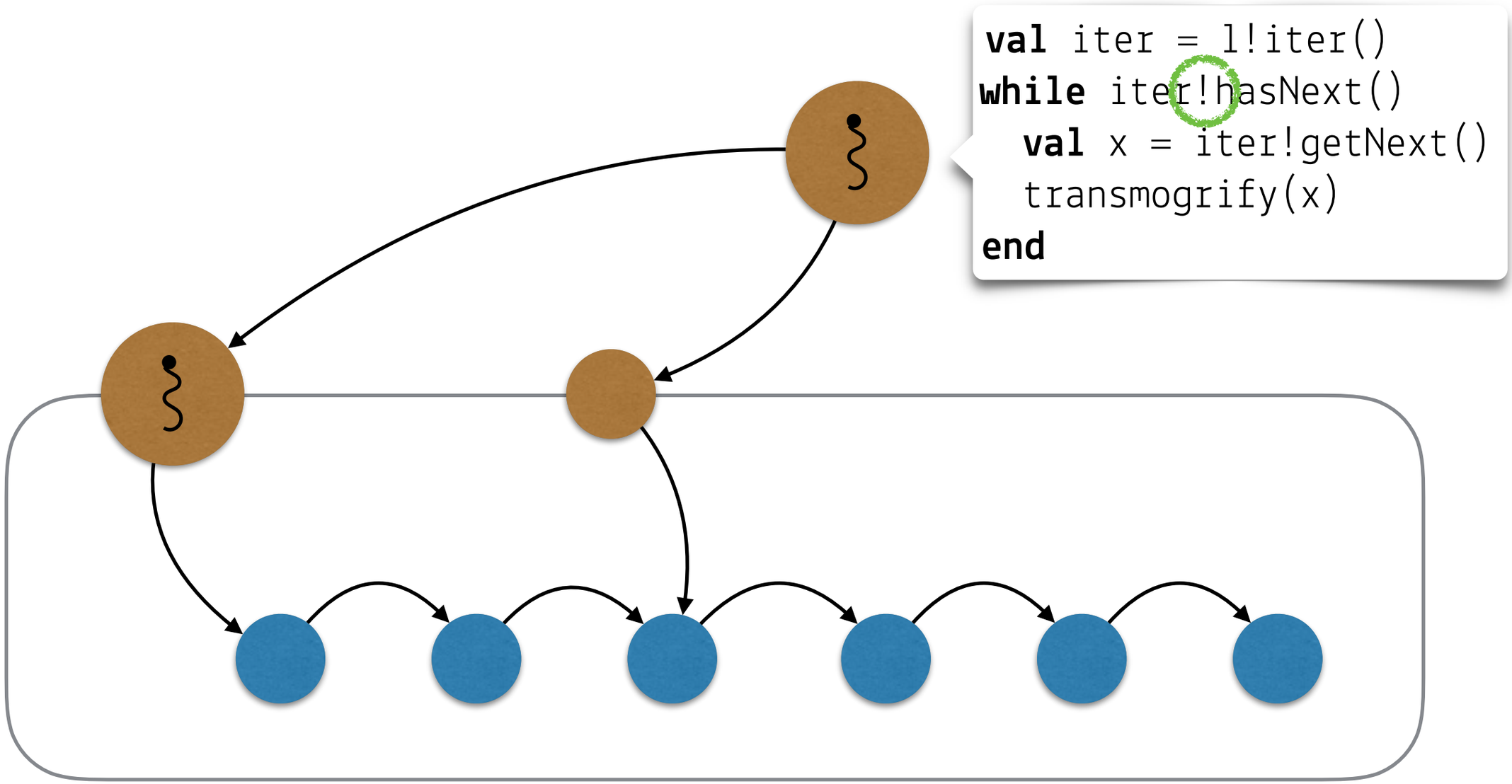
Relax Actor Isolation by Bestowing Activity



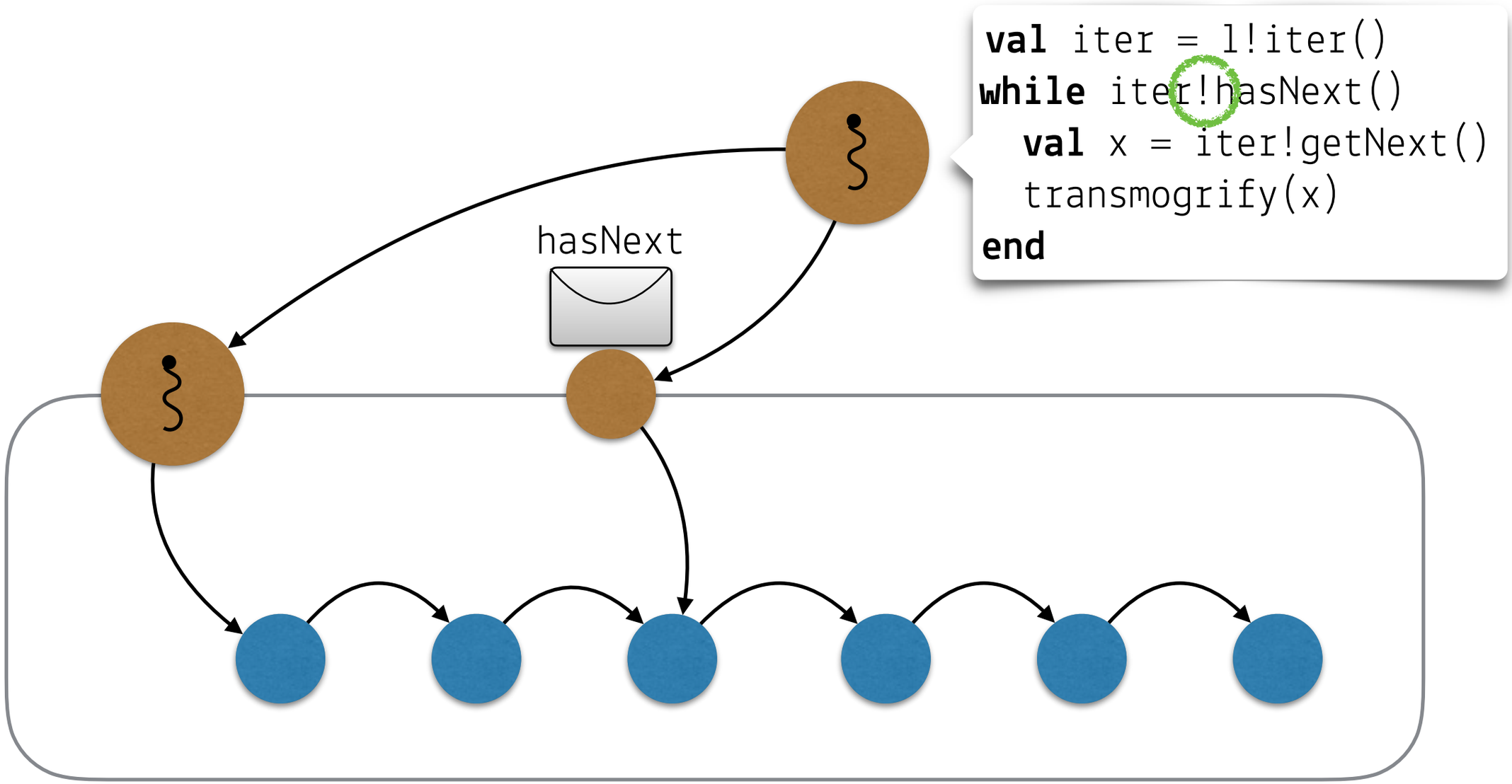
Relax Actor Isolation by Bestowing Activity



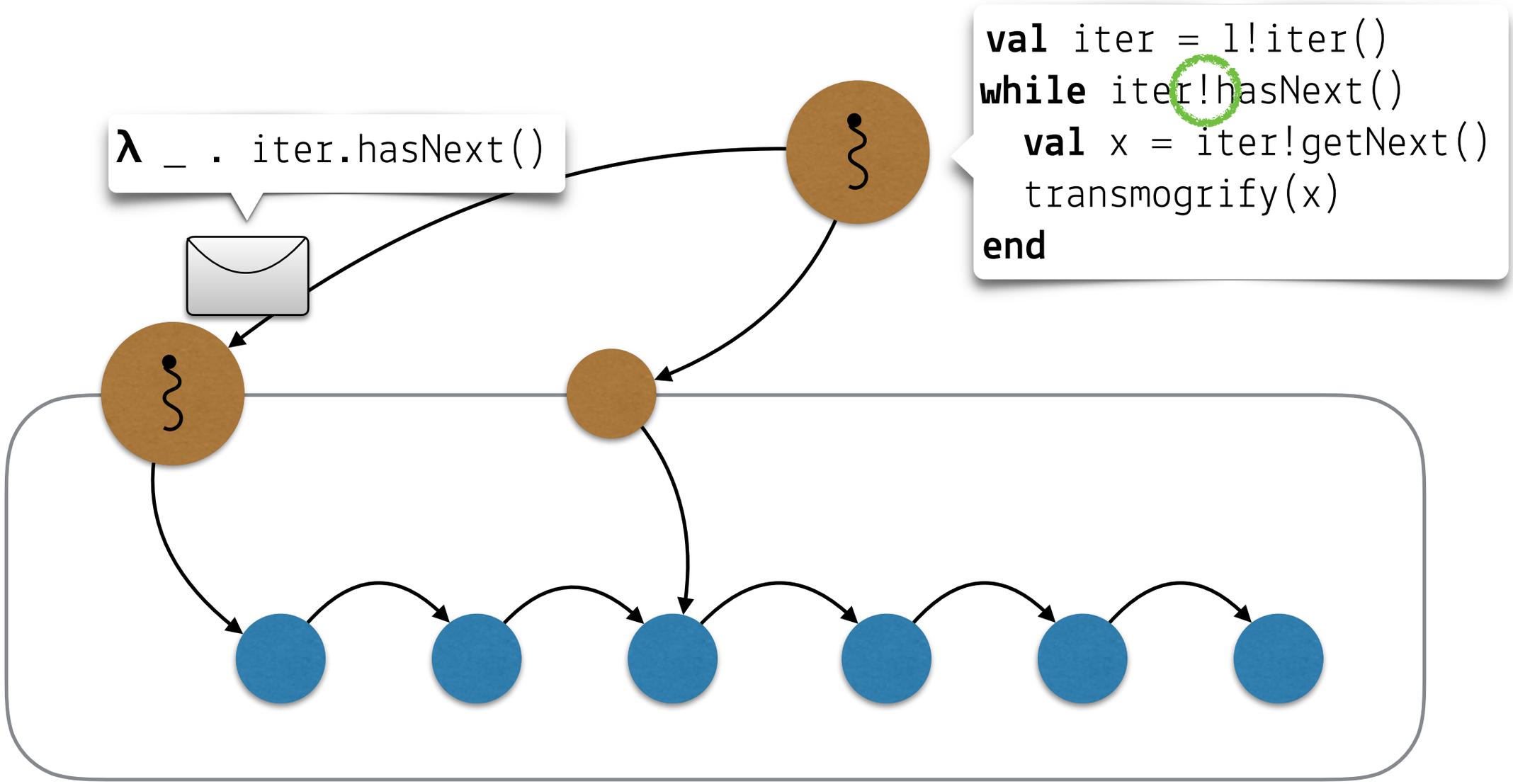
Relax Actor Isolation by Bestowing Activity



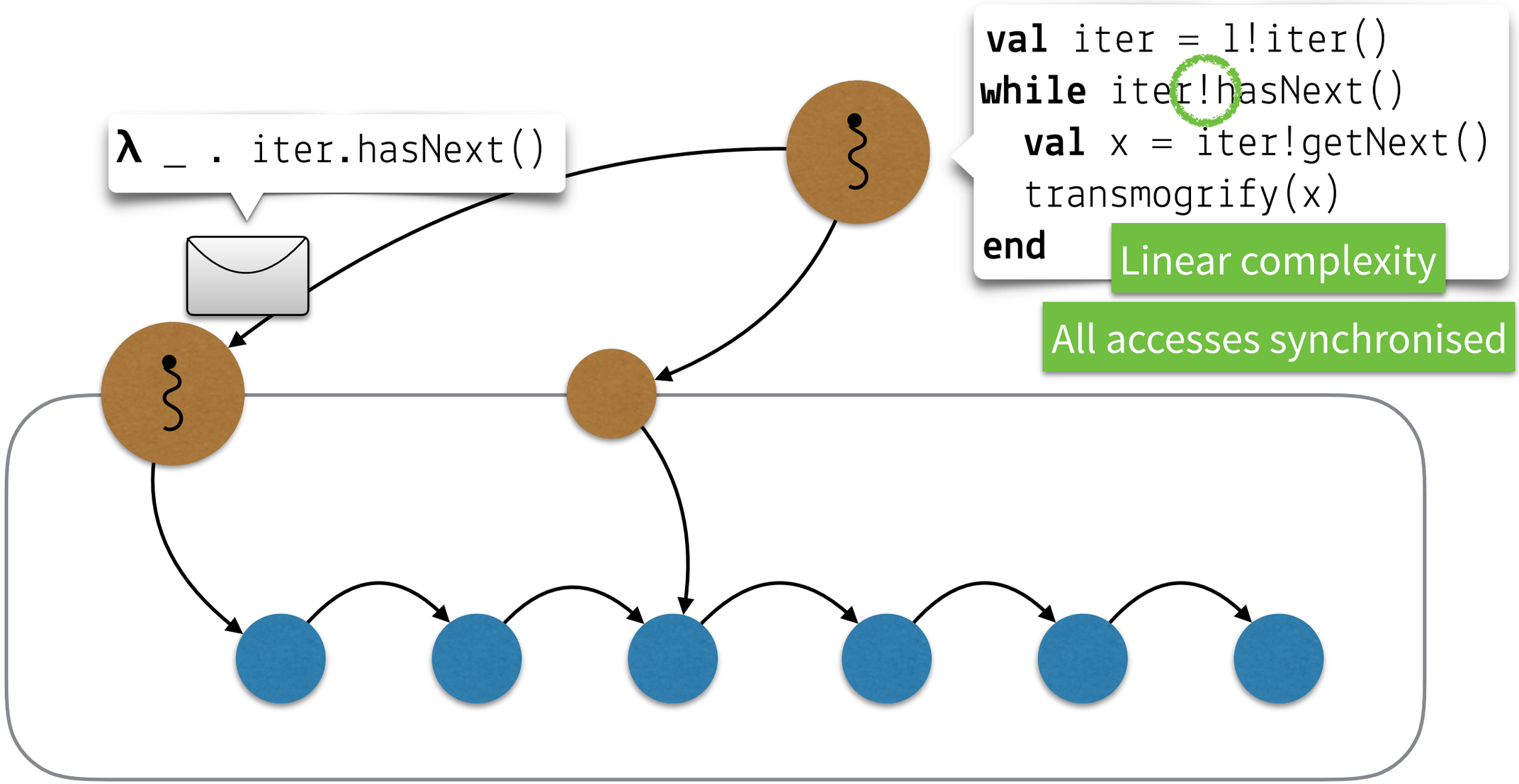
Relax Actor Isolation by Bestowing Activity



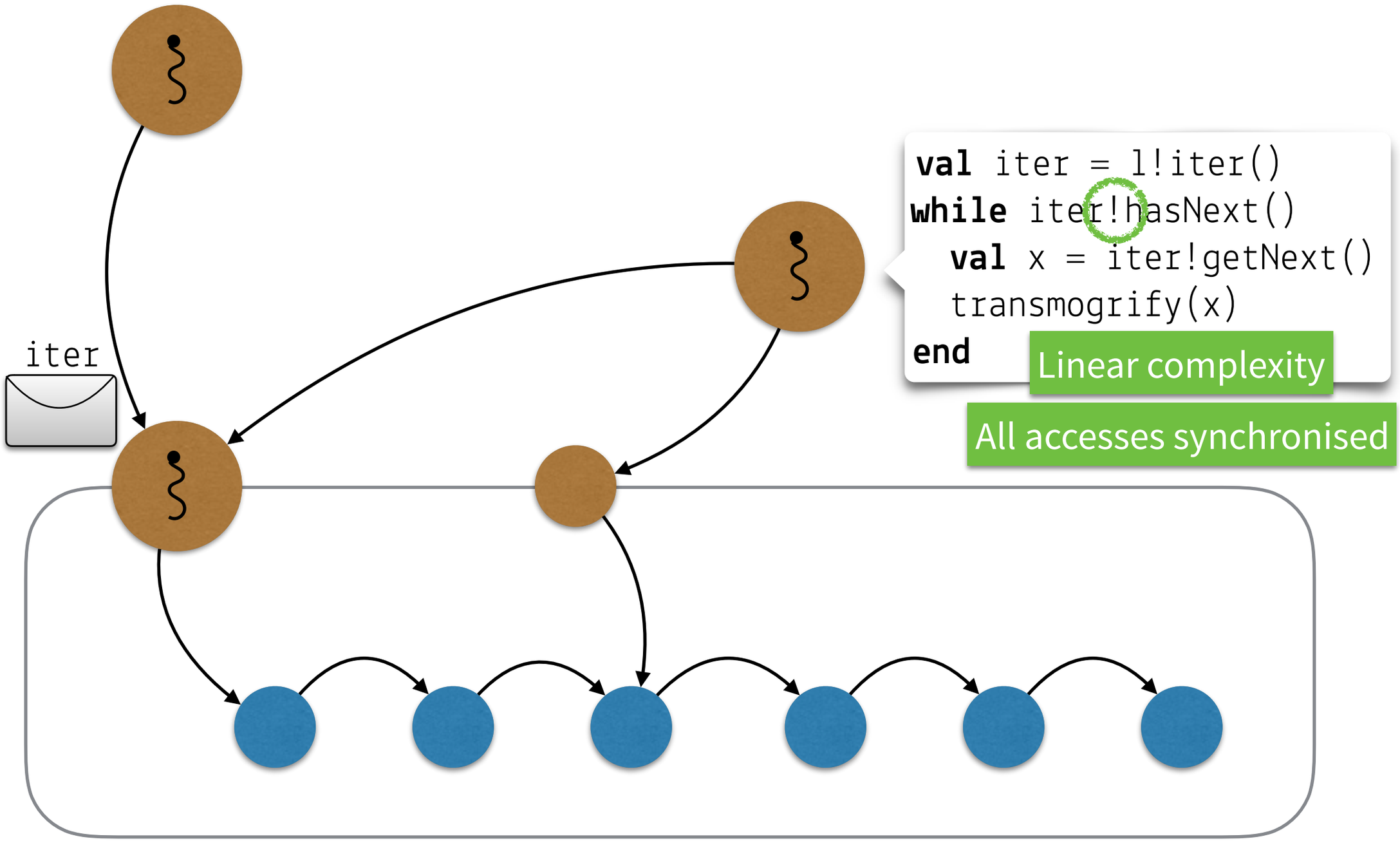
Relax Actor Isolation by Bestowing Activity



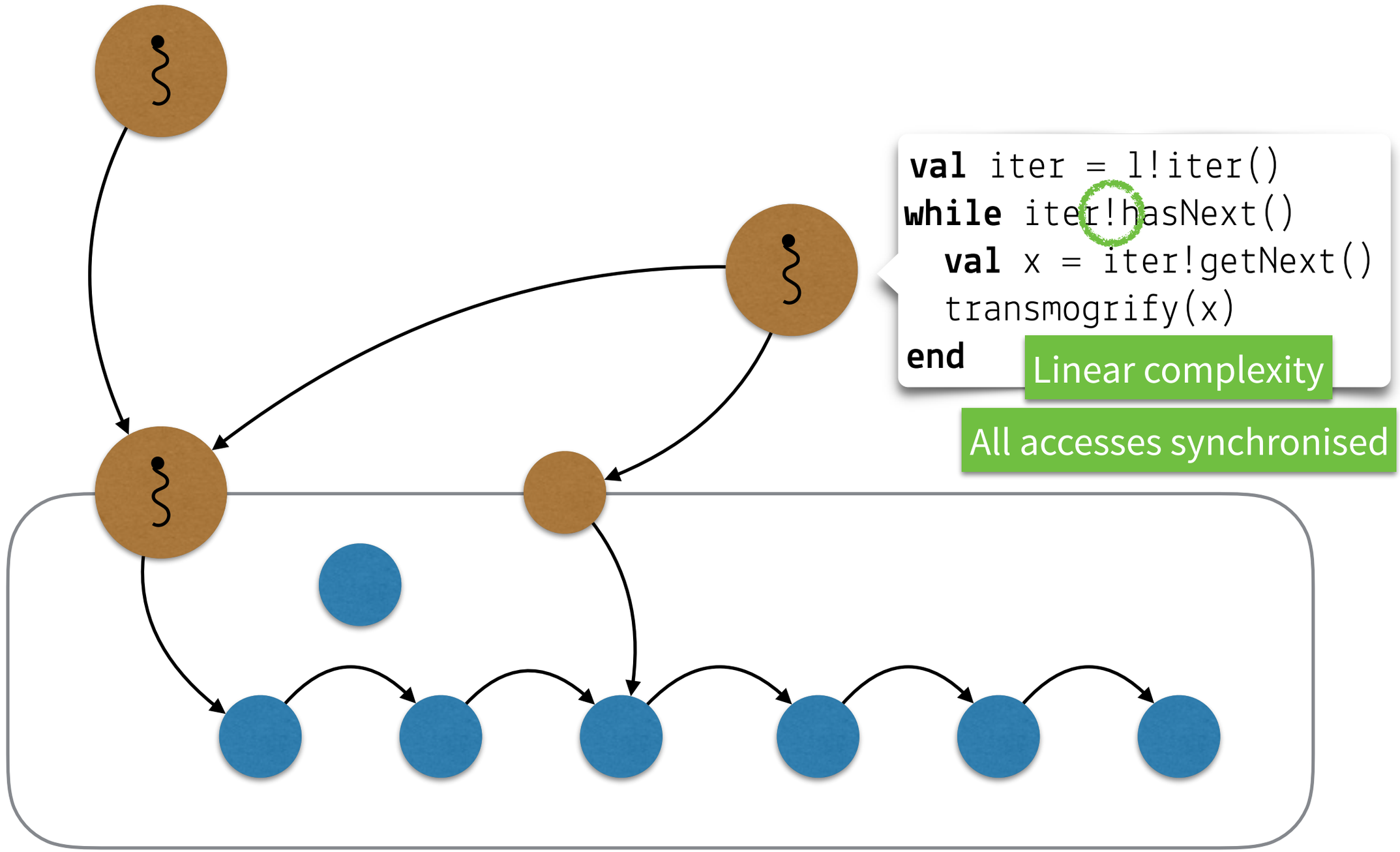
Relax Actor Isolation by Bestowing Activity



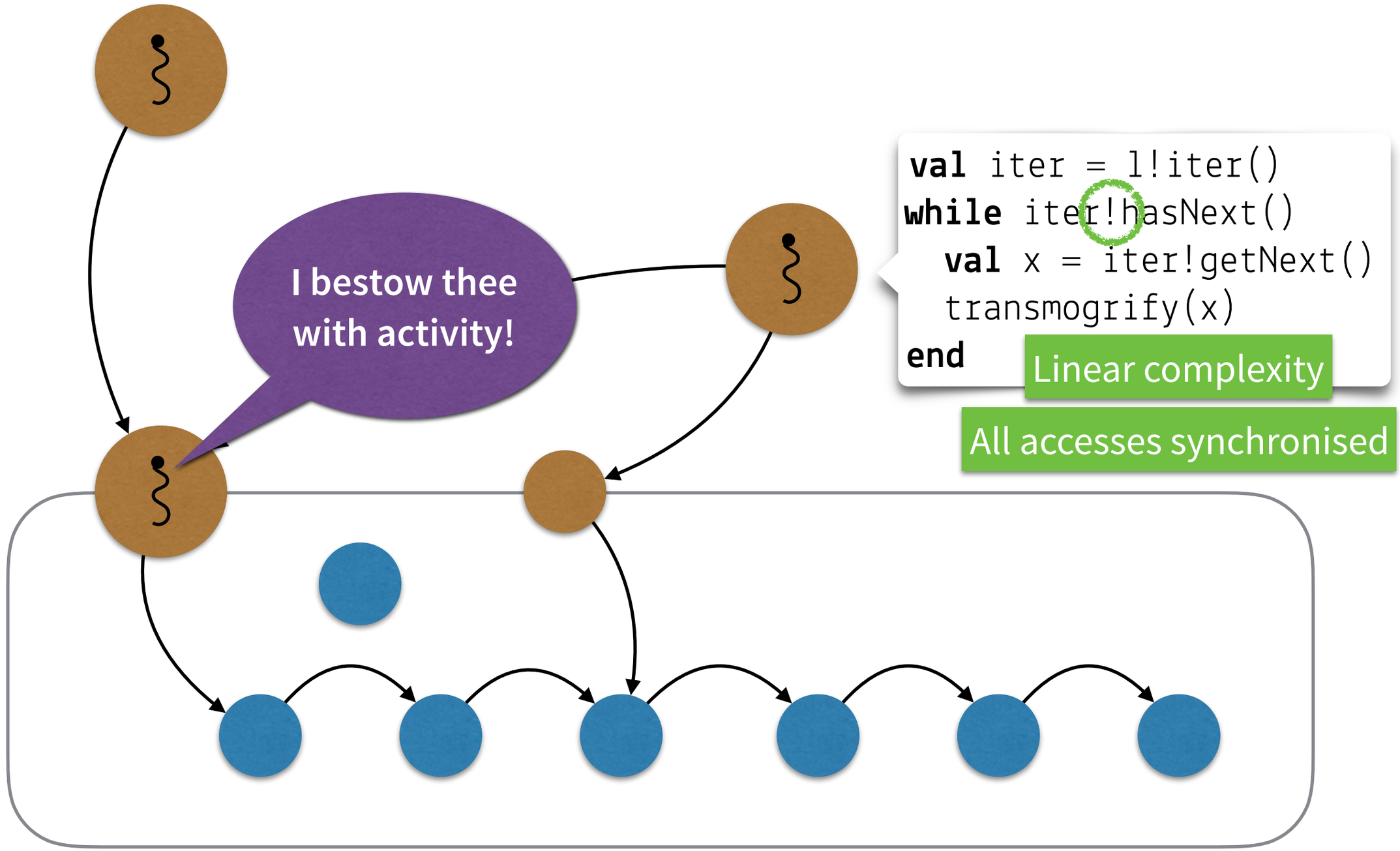
Relax Actor Isolation by Bestowing Activity



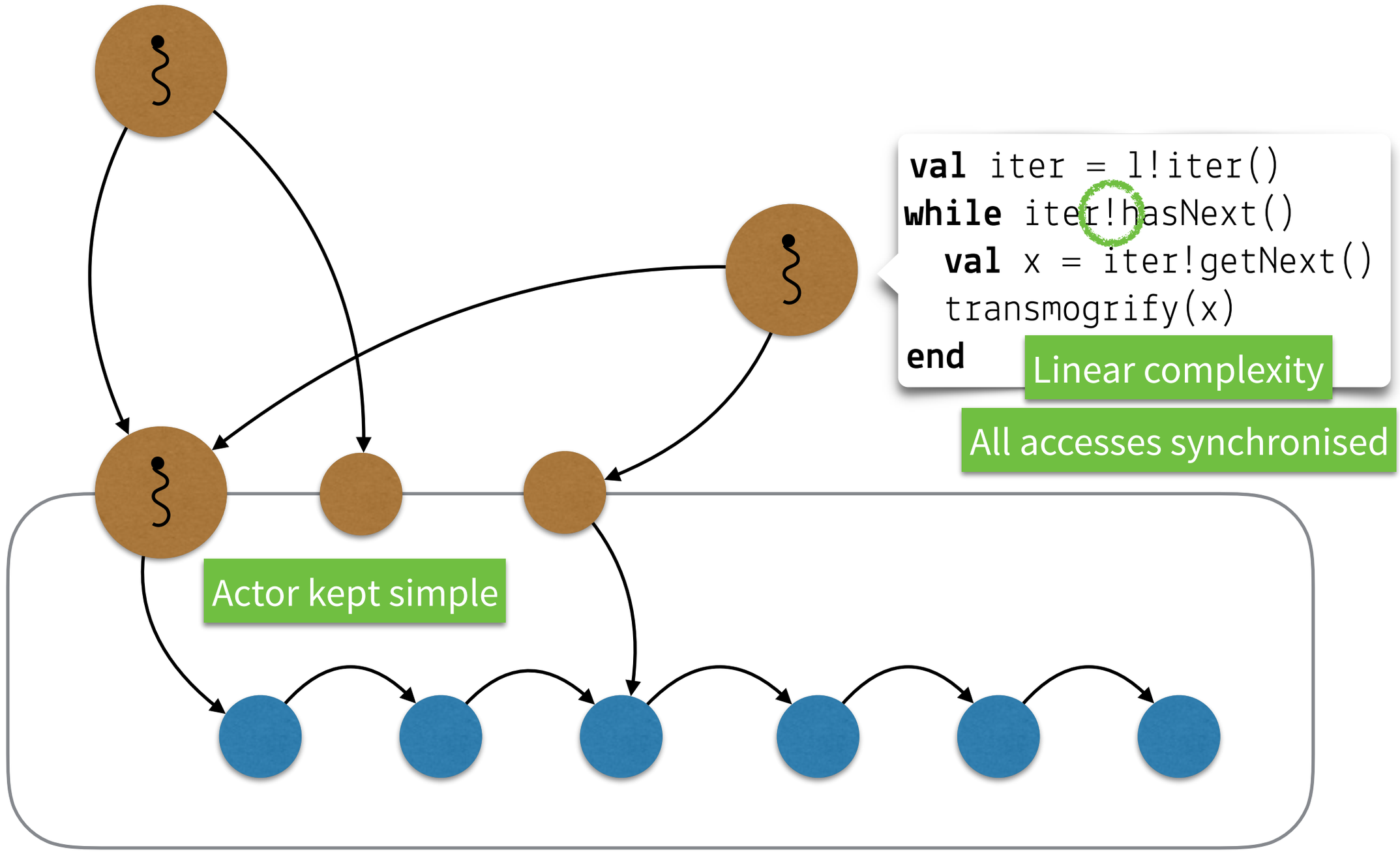
Relax Actor Isolation by Bestowing Activity



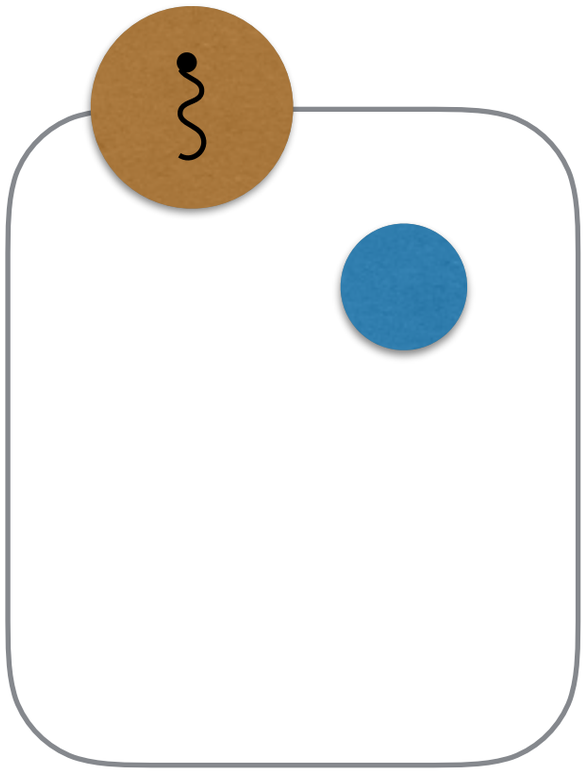
Relax Actor Isolation by Bestowing Activity



Relax Actor Isolation by Bestowing Activity



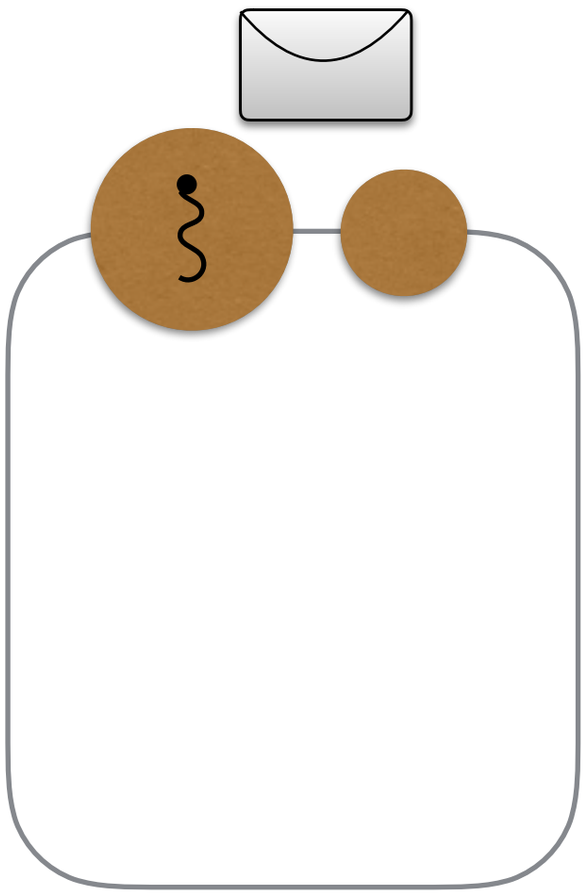
Delegating Parts of an Actor's Interface



```
actor A
  var c : C
  def foo() : unit
    …
  end
  def bar() : unit
    …
  end
  def getC : bestowed[C]
    bestow this.c
  end
end
```

```
class C
  def beep() : unit
    …
  end
  def boop() : unit
    …
  end
end
```

Delegating Parts of an Actor's Interface



```
actor A
  var c : C
  def foo() : unit
    …
  end
  def bar() : unit
    …
  end
  def getC : bestowed[C]
    bestow this.c
  end
end
```

```
class C
  def beep() : unit
    …
  end
  def boop() : unit
    …
  end
end
```

Formalising Bestowed References

- Syntax

$$
\begin{array}{lcl}
e & ::= & x \mid e\ e \mid e!v \mid e.\texttt{mutate}() \mid \textbf{new}\ \tau \mid \textbf{bestow}\ e \mid v \\
v & ::= & \lambda x : \tau . e \mid () \mid id \mid \iota \mid \iota_{id}
\end{array}
\qquad
\begin{array}{lcl}
\tau & ::= & \alpha \mid \mathsf{p} \mid \tau \to \tau \mid \texttt{Unit} \\
\alpha & ::= & \mathsf{c} \mid \mathbf{B}(\mathsf{p})
\end{array}
$$

Formalising Bestowed References

- Syntax

Message send

Bestowing

Passive type

$$e ::= x \mid e\ e \mid e!v \mid e.\texttt{mutate}() \mid \textbf{new}\ \tau \mid \textbf{bestow}\ e \mid v \qquad \tau ::= \alpha \mid \mathsf{p} \mid \tau \to \tau \mid \texttt{Unit}$$

$$v ::= \lambda x : \tau . e \mid () \mid id \mid \iota \mid \iota_{id} \qquad \alpha ::= \mathsf{c} \mid \mathbf{B}(\mathsf{p})$$

Bestowed type

Actor type

Bestowed value

- Static semantics

$\boxed{\Gamma \vdash e : \tau}$

(Expressions)

$$\text{E-VAR}\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \text{E-APPLY}\ \frac{\Gamma \vdash e : \tau' \to \tau \quad \Gamma \vdash e' : \tau'}{\Gamma \vdash e\,e' : \tau} \qquad \text{E-NEW-PASSIVE}\ \frac{}{\Gamma \vdash \textbf{new}\,\mathsf{p} : \mathsf{p}} \qquad \text{E-NEW-ACTOR}\ \frac{}{\Gamma \vdash \textbf{new}\,\mathsf{c} : \mathsf{c}}$$

$$\text{E-MUTATE}\ \frac{\Gamma \vdash e : \mathsf{p}}{\Gamma \vdash e.\texttt{mutate}() : \texttt{Unit}} \qquad \text{E-BESTOW}\ \frac{\Gamma \vdash e : \mathsf{p}}{\Gamma \vdash \textbf{bestow}\,e : \mathbf{B}(p)} \qquad \text{E-SEND}\ \frac{\Gamma \vdash e : \alpha \quad \Gamma_\alpha, x : \mathsf{p} \vdash e' : \tau' \quad \nexists\, \iota\ .\ \iota \in e'}{\Gamma \vdash e!\lambda x : \mathsf{p}.e' : \texttt{Unit}}$$

$$\text{E-FN}\ \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash (\lambda x : \tau . e) : \tau \to \tau'} \qquad \text{E-UNIT}\ \frac{}{\Gamma \vdash () : \texttt{Unit}} \qquad \text{E-LOC}\ \frac{}{\Gamma \vdash \iota : \mathsf{p}} \qquad \text{E-ID}\ \frac{}{\Gamma \vdash id : \mathsf{c}} \qquad \text{E-BESTOWED}\ \frac{}{\Gamma \vdash \iota_{id} : \mathbf{B}(p)}$$

Formalising Bestowed References

- Syntax

Message send

Bestowing

Passive type

$$e ::= x \mid e\ e \mid e!v \mid e.\texttt{mutate}() \mid \textbf{new}\ \tau \mid \textbf{bestow}\ e \mid v \qquad \tau ::= \alpha \mid \mathsf{p} \mid \tau \to \tau \mid \texttt{Unit}$$

$$v ::= \lambda x : \tau . e \mid () \mid id \mid \iota \mid \iota_{id} \qquad \alpha ::= \mathsf{c} \mid \mathbf{B}(\mathsf{p})$$

Bestowed type

Bestowed value

Actor type

- Static semantics

$\boxed{\Gamma \vdash e : \tau}$

(Expressions)

E-VAR
$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

E-APPLY
$$\frac{\Gamma \vdash e : \tau' \to \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash e\,e' : \tau}$$

E-NEW-PASSIVE
$$\frac{}{\Gamma \vdash \textbf{new}\,\mathsf{p} : \mathsf{p}}$$

E-NEW-ACTOR
$$\frac{}{\Gamma \vdash \textbf{new}\,\mathsf{c} : \mathsf{c}}$$

E-MUTATE
$$\frac{\Gamma \vdash e : \mathsf{p}}{\Gamma \vdash e.\texttt{mutate}() : \texttt{Unit}}$$

E-FN
$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash (\lambda x : \tau . e) : \tau \to \tau'}$$

E-UNIT
$$\frac{}{\Gamma \vdash () : \texttt{Unit}}$$

E-LOC
$$\frac{}{\Gamma \vdash \iota : \mathsf{p}}$$

E-ID
$$\frac{}{\Gamma \vdash id : \mathsf{c}}$$

E-BESTOWED
$$\frac{}{\Gamma \vdash \iota_{id} : \mathbf{B}(p)}$$

E-BESTOW
$$\frac{\Gamma \vdash e : \mathsf{p}}{\Gamma \vdash \textbf{bestow}\ e : \mathbf{B}(p)}$$

E-SEND
$$\frac{\Gamma \vdash e : \alpha \qquad \Gamma_\alpha, x : \mathsf{p} \vdash e' : \tau' \qquad \nexists\, \iota\ .\ \iota \in e'}{\Gamma \vdash e!\lambda x : \mathsf{p} . e' : \texttt{Unit}}$$

Formalising Bestowed References

- Dynamic semantics

$$\boxed{id \vdash \langle H, e\rangle \hookrightarrow \langle H', e'\rangle} \qquad \textit{(Evaluation of expressions)}$$

EVAL-SEND-ACTOR

$$\frac{H(id') = (\iota, L, Q, e) \qquad H' = H[id' \mapsto (\iota, L, v\,Q, e)]}{id \vdash \langle H, id' ! v\rangle \hookrightarrow \langle H', ()\rangle}$$

EVAL-SEND-BESTOWED

$$\frac{H(id') = (\iota', L, Q, e) \qquad H' = H[id' \mapsto (\iota', L, (\lambda x : \mathsf{p}.v\,\iota)\,Q, e)]}{id \vdash \langle H, \iota_{id'} ! v\rangle \hookrightarrow \langle H', ()\rangle}$$

EVAL-APPLY

$$\frac{e' = e[x \mapsto v]}{id \vdash \langle H, (\lambda x : \tau.e)\,v\rangle \hookrightarrow \langle H, e'\rangle}$$

EVAL-MUTATE

$$\frac{}{id \vdash \langle H, \iota.\texttt{mutate}()\rangle \hookrightarrow \langle H, ()\rangle}$$

EVAL-BESTOW

$$\frac{}{id \vdash \langle H, \mathbf{bestow}\,\iota\rangle \hookrightarrow \langle H, \iota_{id}\rangle}$$

EVAL-NEW-PASSIVE

$$\frac{H(id) = (\iota, L, Q, e) \qquad \iota' \textbf{ fresh} \qquad H' = H[id \mapsto (\iota, L \cup \{\iota'\}, Q, e)]}{id \vdash \langle H, \mathbf{new}\,\mathsf{p}\rangle \hookrightarrow \langle H', \iota'\rangle}$$

EVAL-NEW-ACTOR

$$\frac{id' \textbf{ fresh} \qquad \iota' \textbf{ fresh} \qquad H' = H[id' \mapsto (\iota', \{\iota'\}, \epsilon, ())]}{id \vdash \langle H, \mathbf{new}\,\alpha\rangle \hookrightarrow \langle H', id'\rangle}$$

EVAL-CONTEXT

$$\frac{id \vdash \langle H, e\rangle \hookrightarrow \langle H', e'\rangle}{id \vdash \langle H, E[e]\rangle \hookrightarrow \langle H', E[e']\rangle}$$

Formalising Bestowed References

- Dynamic semantics

$id \vdash \langle H, e\rangle \hookrightarrow \langle H', e'\rangle$ *(Evaluation of expressions)*

EVAL-SEND-ACTOR

$$\frac{H(id') = (\iota, L, Q, e) \qquad H' = H[id' \mapsto (\iota, L, v\,Q, e)]}{id \vdash \langle H, id' ! v\rangle \hookrightarrow \langle H', ()\rangle}$$

EVAL-SEND-BESTOWED

$$\frac{H(id') = (\iota', L, Q, e) \qquad H' = H[id' \mapsto (\iota', L, (\lambda x : \mathsf{p}.v\,\iota)\,Q, e)]}{id \vdash \langle H, \iota_{id'} ! v\rangle \hookrightarrow \langle H', ()\rangle}$$

EVAL-APPLY

$$\frac{e' = e[x \mapsto v]}{id \vdash \langle H, (\lambda x : \tau.e)\,v\rangle \hookrightarrow \langle H, e'\rangle}$$

EVAL-MUTATE

$$\frac{}{id \vdash \langle H, \iota.\mathtt{mutate}()\rangle \hookrightarrow \langle H, ()\rangle}$$

EVAL-BESTOW

$$\frac{}{id \vdash \langle H, \mathbf{bestow}\,\iota\rangle \hookrightarrow \langle H, \iota_{id}\rangle}$$

EVAL-NEW-PASSIVE

$$\frac{H(id) = (\iota, L, Q, e) \qquad \iota' \textbf{ fresh} \qquad H' = H[id \mapsto (\iota, L \cup \{\iota'\}, Q, e)]}{id \vdash \langle H, \mathbf{new}\,\mathsf{p}\rangle \hookrightarrow \langle H', \iota'\rangle}$$

EVAL-NEW-ACTOR

$$\frac{id' \textbf{ fresh} \qquad \iota' \textbf{ fresh} \qquad H' = H[id' \mapsto (\iota', \{\iota'\}, \epsilon, ())]}{id \vdash \langle H, \mathbf{new}\,\alpha\rangle \hookrightarrow \langle H', id'\rangle}$$

EVAL-CONTEXT

$$\frac{id \vdash \langle H, e\rangle \hookrightarrow \langle H', e'\rangle}{id \vdash \langle H, E[e]\rangle \hookrightarrow \langle H', E[e']\rangle}$$

Properties of the Formalism

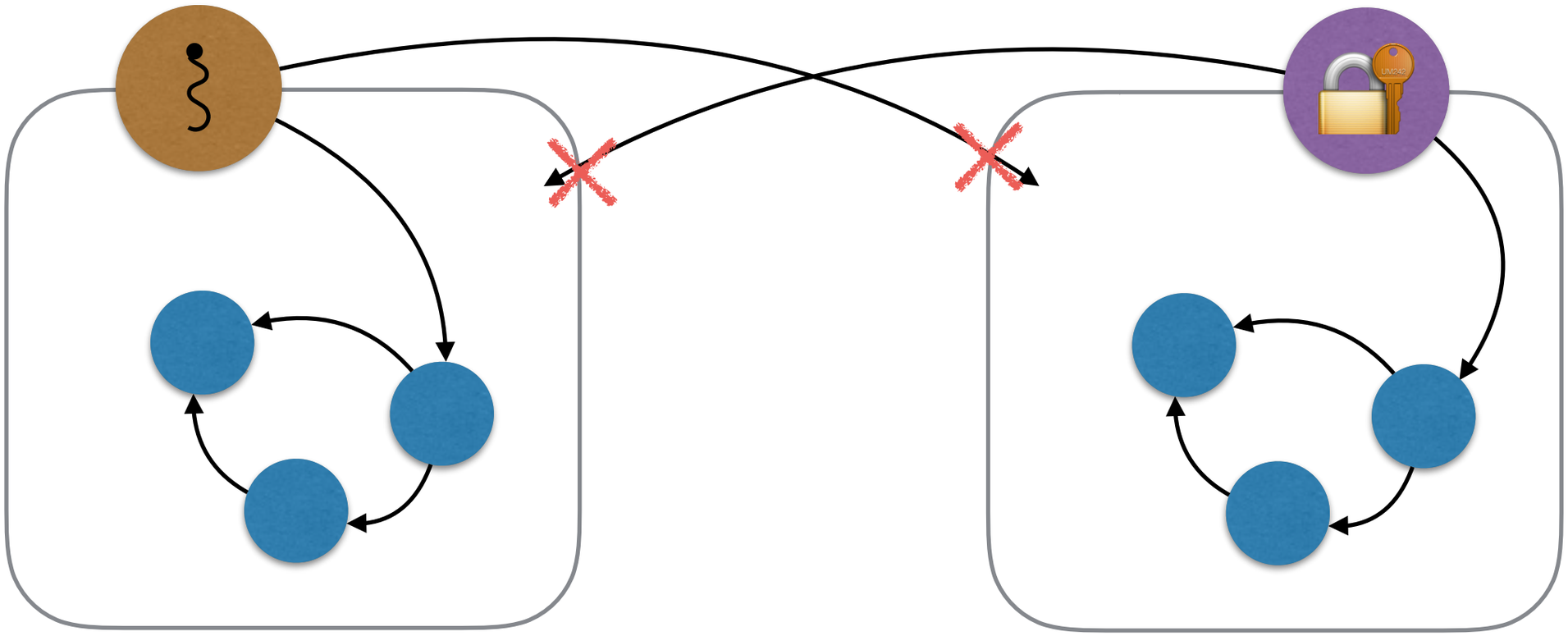
- **Progress:** $\vdash H \implies (\exists H' \,.\, H \hookrightarrow H') \;\vee\; (\forall id \in \mathbf{dom}(H) \,.\, H(id) = (\iota, L, \epsilon, v))$
- **Preservation:** $\vdash H \wedge H \hookrightarrow H' \implies \vdash H'$
- **Data-race freedom:** Two actors will never mutate the same passive object

$$\left(\begin{array}{c} id_1 \neq id_2 \\ \wedge\, H(id_1) = (\iota_1, L_1, Q_1, \iota.\texttt{mutate}()) \\ \wedge\, H(id_2) = (\iota_2, L_2, Q_2, \iota'.\texttt{mutate}()) \end{array}\right) \implies \iota \neq \iota'$$

The Big Picture

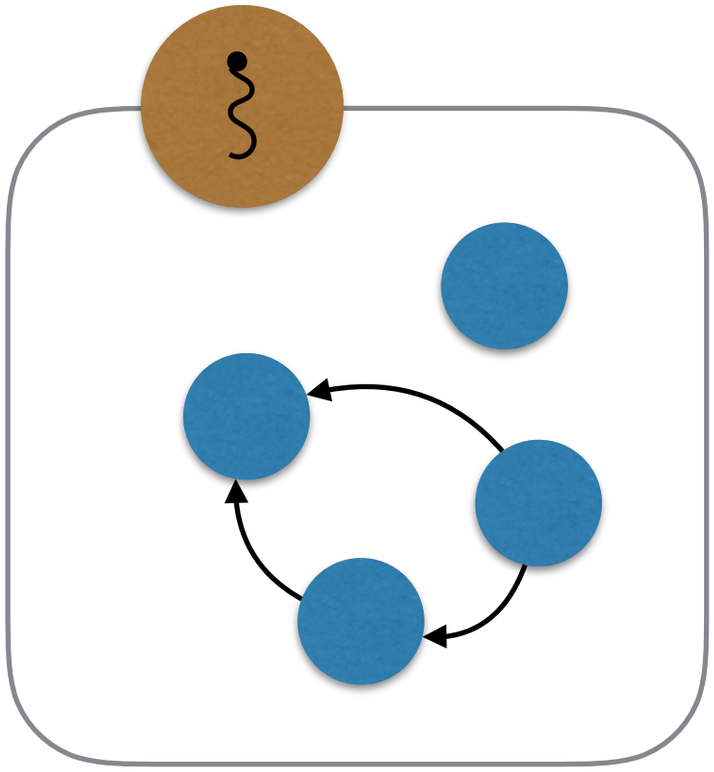
- Kappa is a capability based type-system for concurrent OO-programming [ECOOP'16]

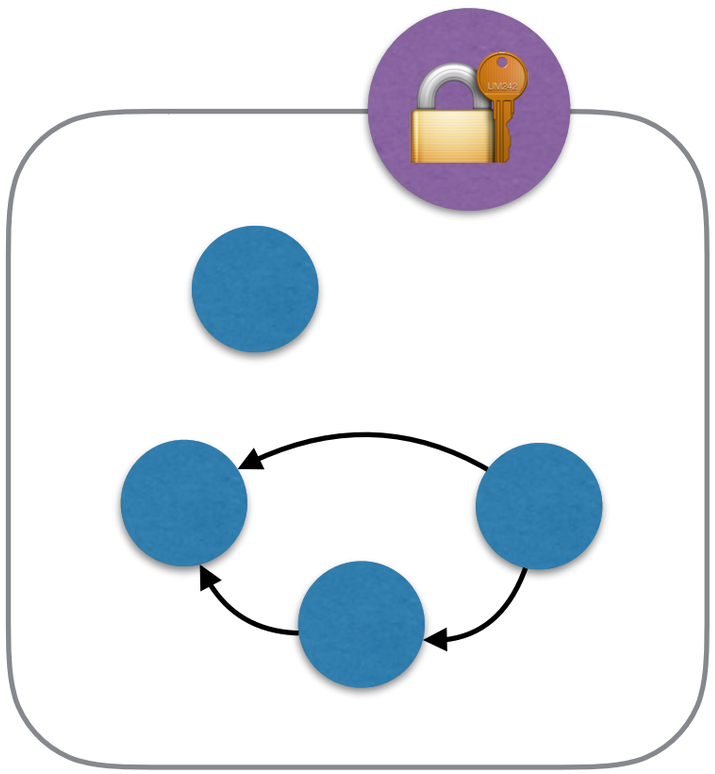
  - Tracks the boundaries of objects to achieve strong encapsulation



Generalizing Bestowed References

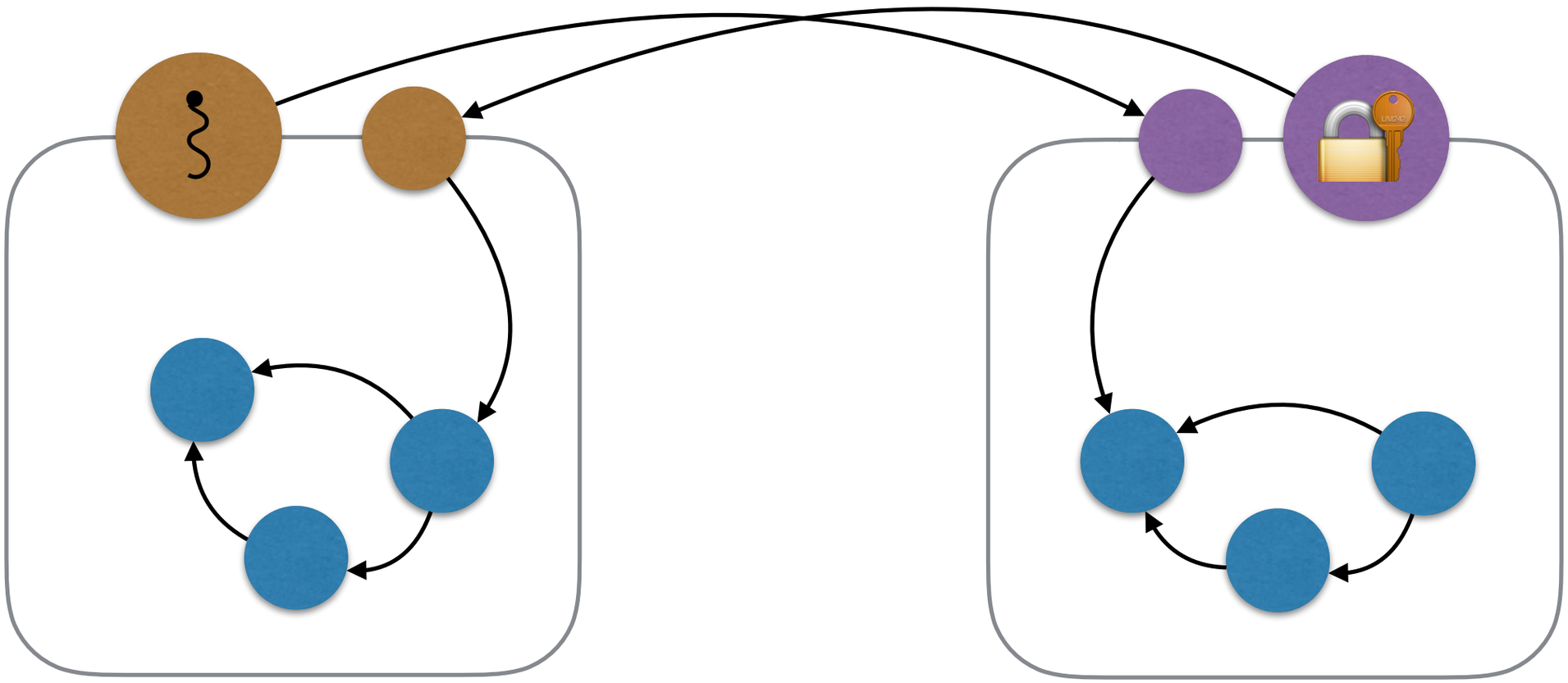
- Bestowed references provide a thread-safe way to break encapsulation





Generalizing Bestowed References

- Bestowed references provide a thread-safe way to break encapsulation



Future Work

- Implementation in Encore

```
active class Foo
    var box : IntBox
    def bestowBox() : Bestowed[IntBox]
        bestow this.box
    end
end

active class Main
    def main() : unit
        var b = new Foo()
        var box = get(b ! bestowBox())
        box ! inc()
    end
end
```

- Enriched formalism with copying and ownership transfer

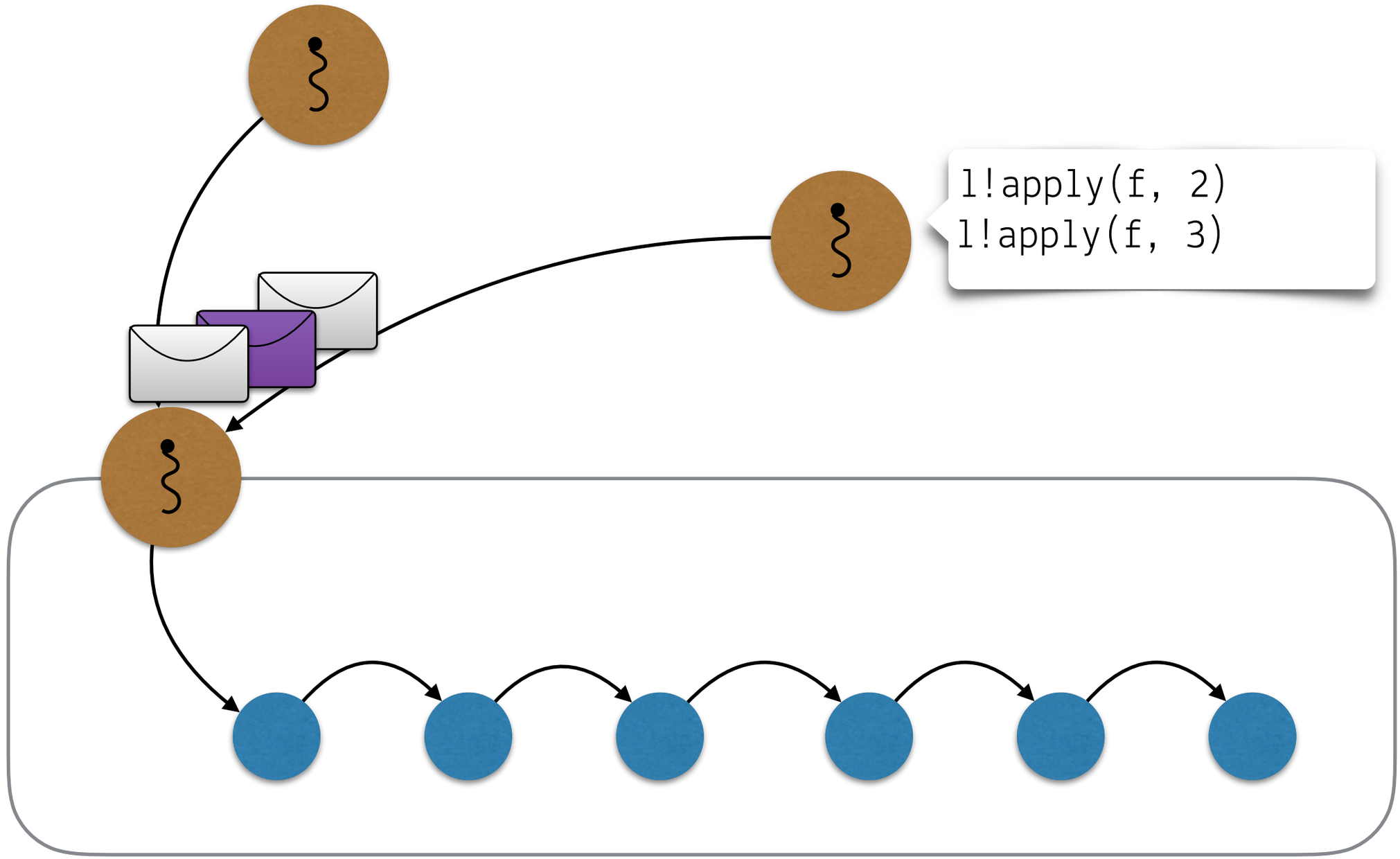
Summary

- Actor isolation can be relaxed by **bestowing** encapsulated objects with activity
  - All accesses will be synchronised via the message queue of the owning actor
  - Actors do not need to know the implementation of their bestowed objects
  - A bestowed object does not need to know that it is bestowed
- The same kind of relaxed encapsulation works for locks
  - All accesses will be synchronised via the lock of the owning object
- **Also in the paper:**

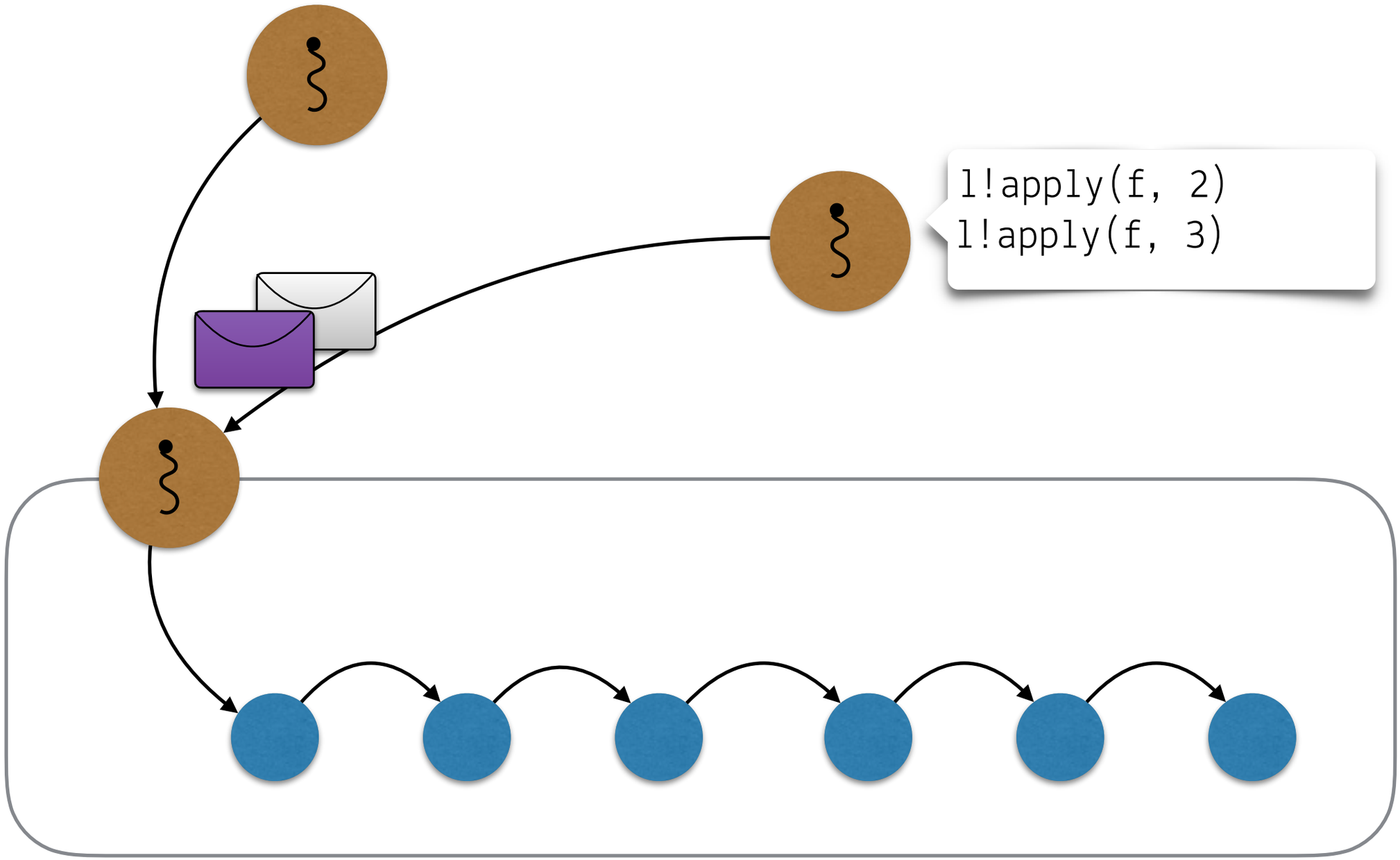
  Atomic blocks to group operations
  Implementation sketches
  Polymorphic concurrency control

Tack! Frågor?

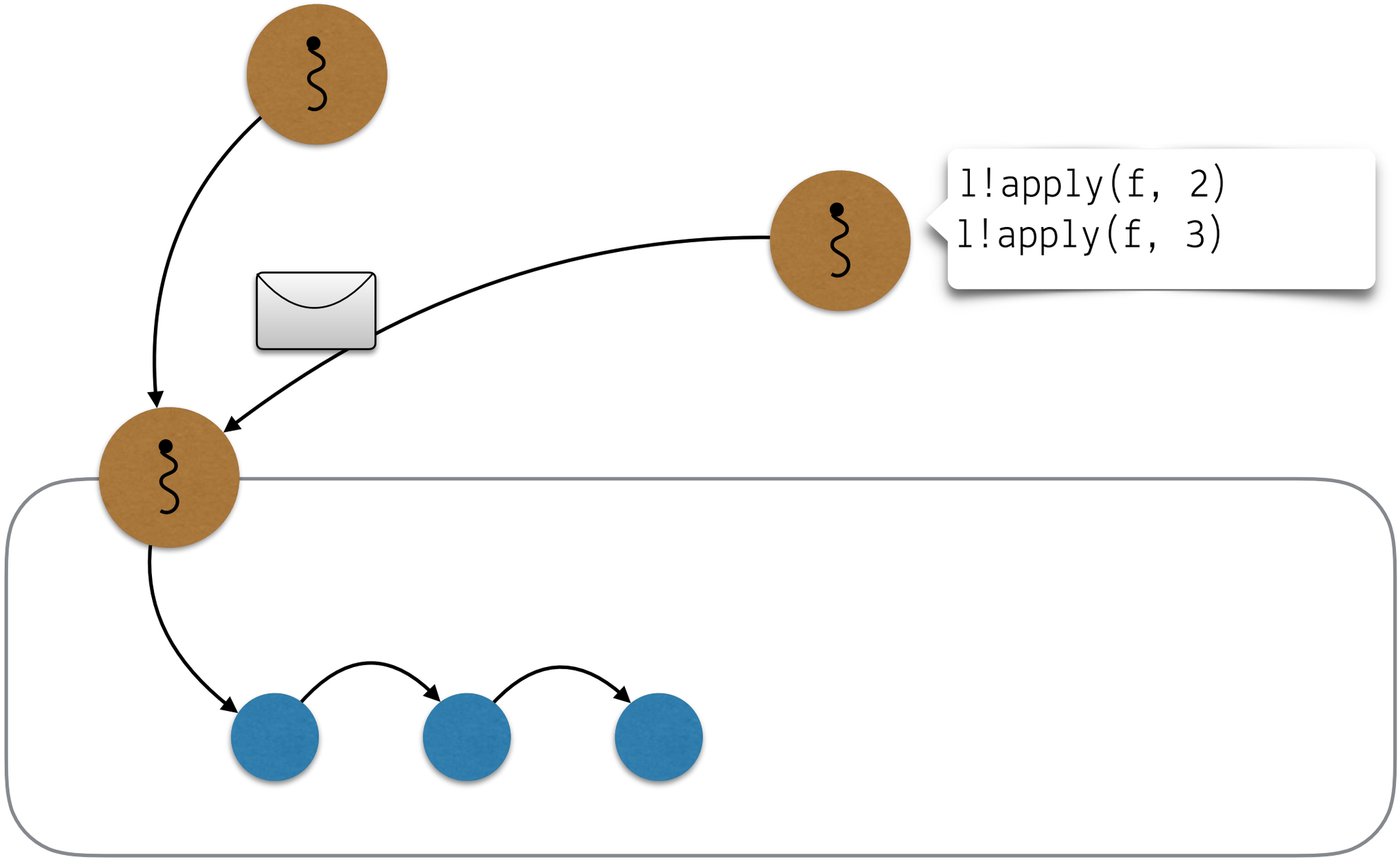
Incompatible Interleaving



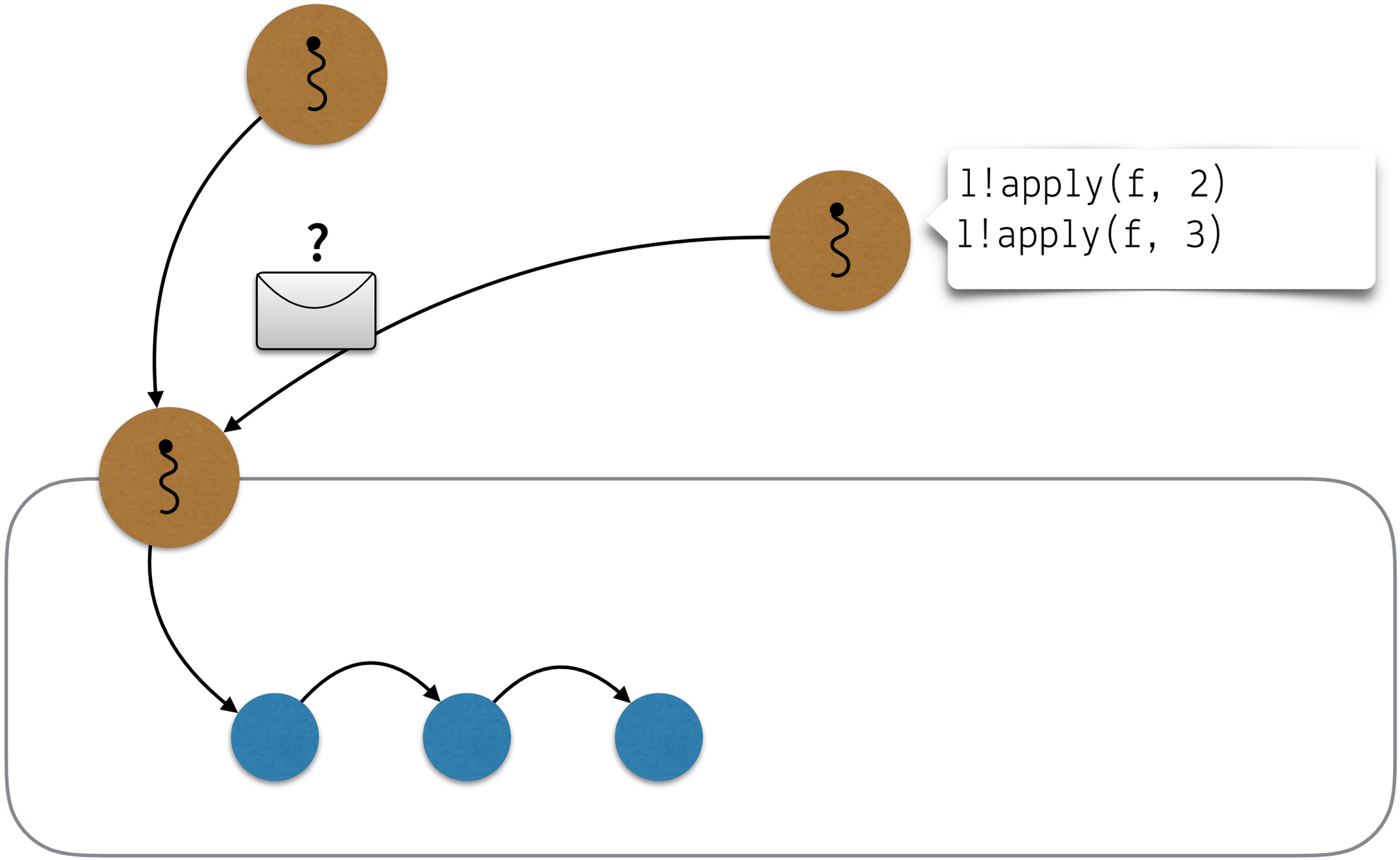
Incompatible Interleaving



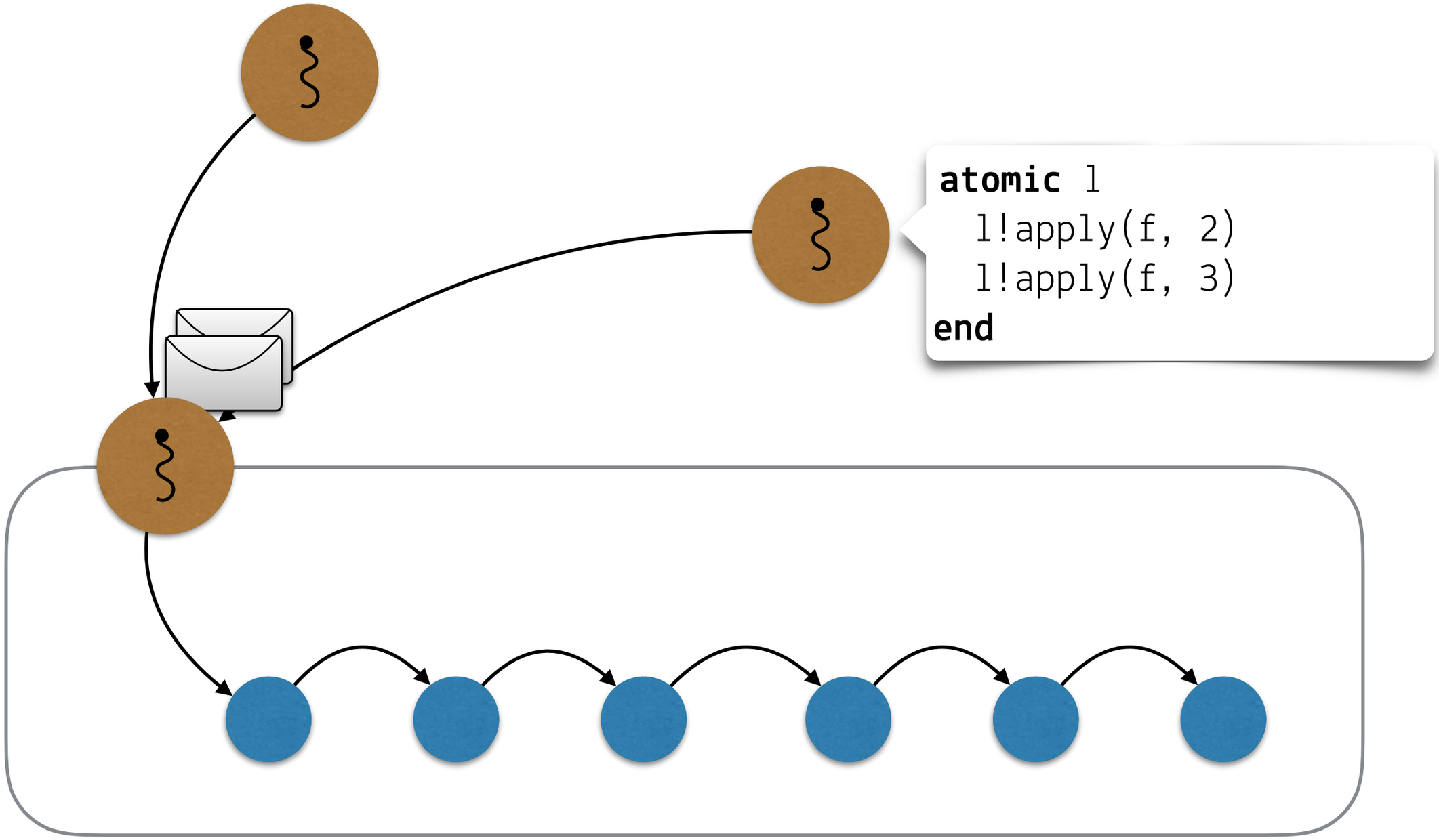
Incompatible Interleaving



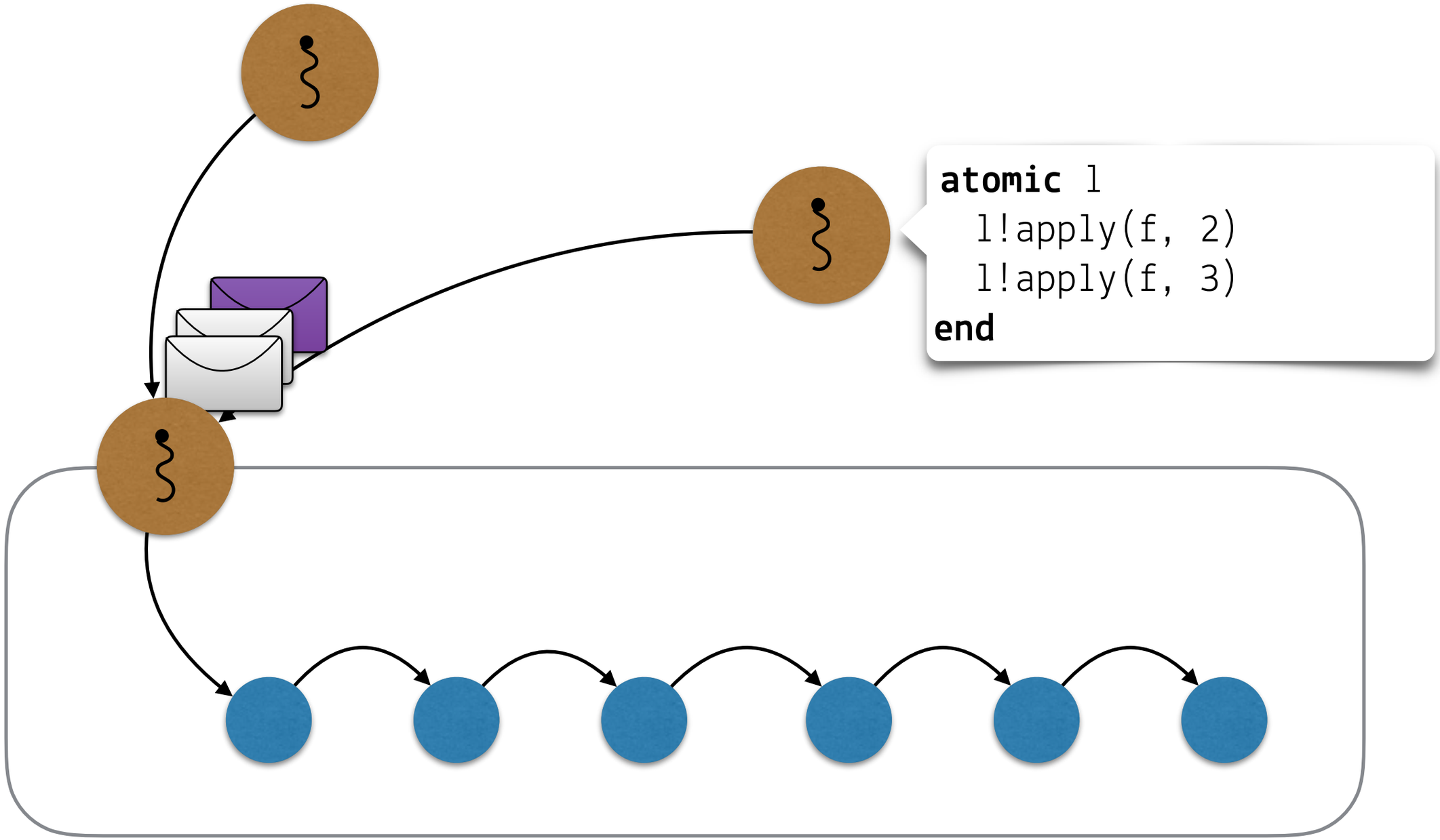
Incompatible Interleaving



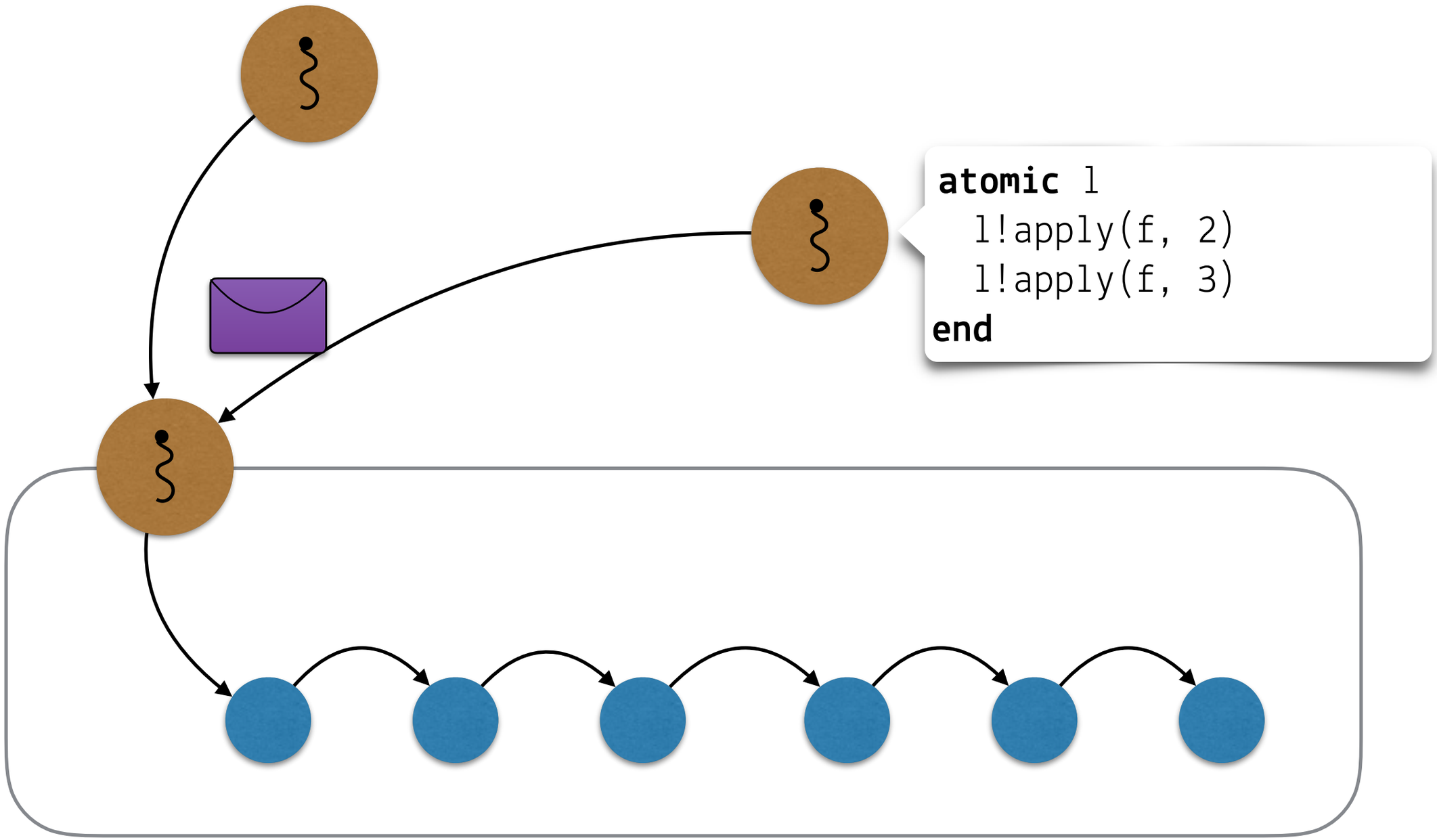
Grouping Messages



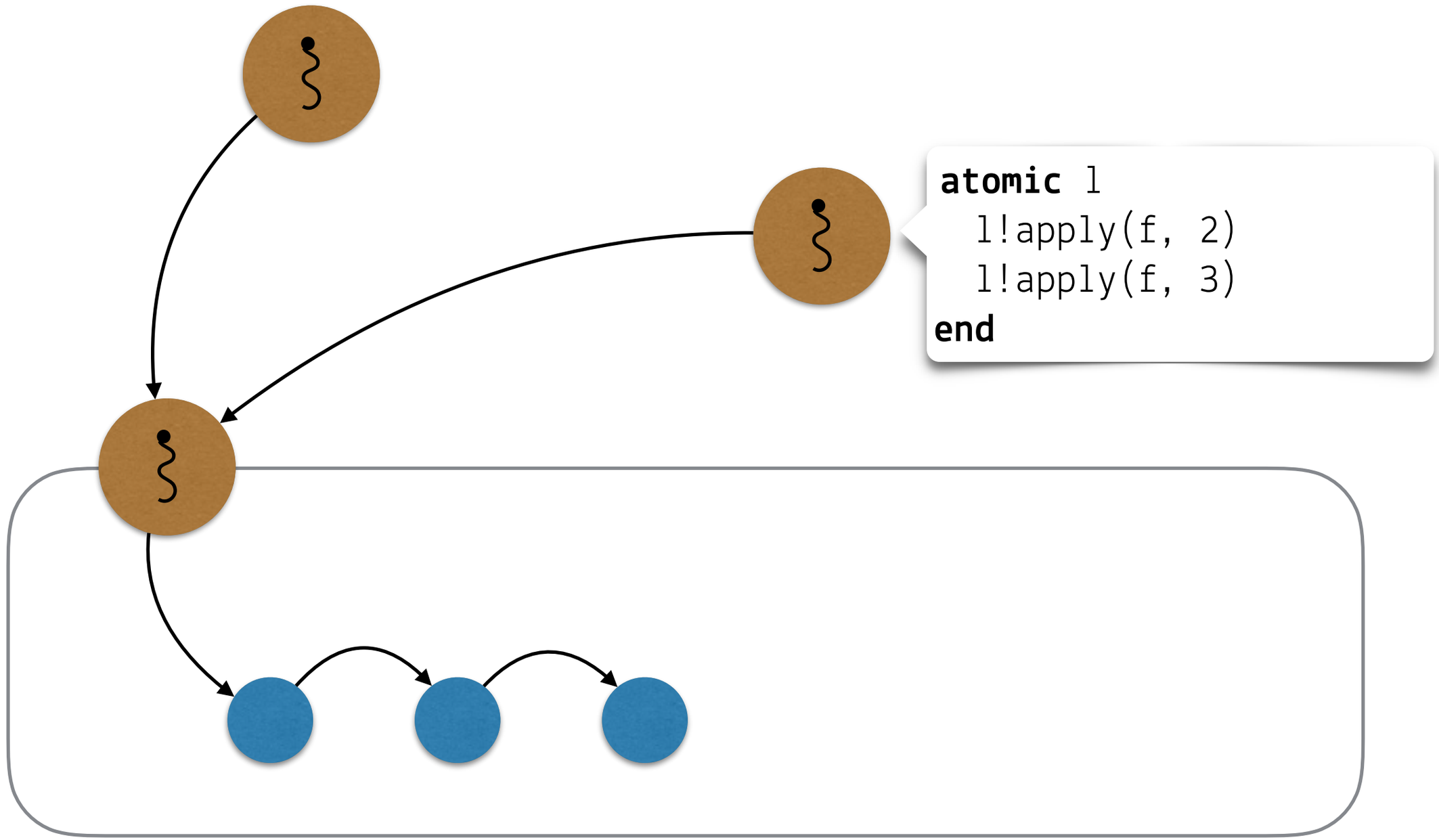
Grouping Messages



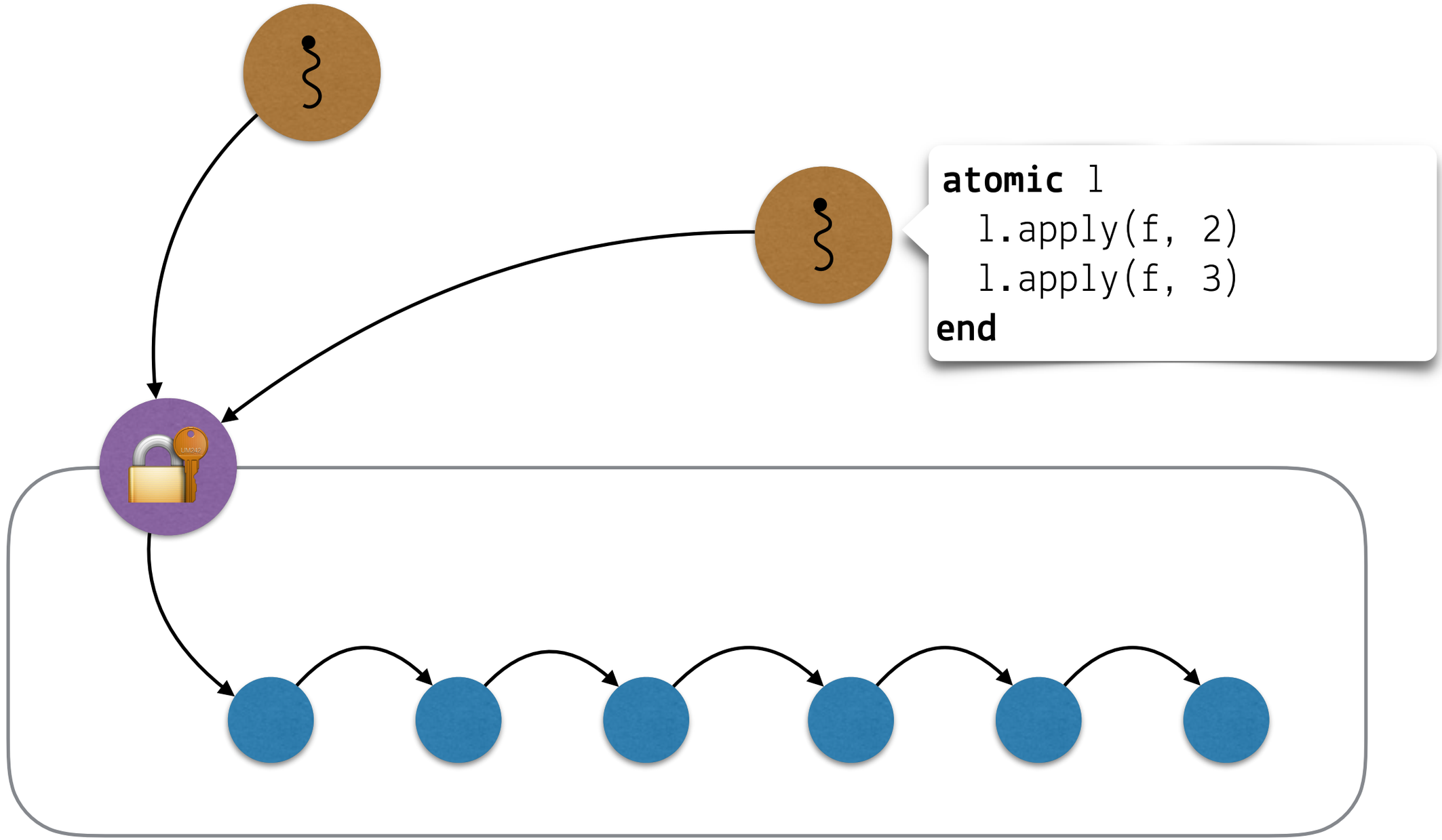
Grouping Messages



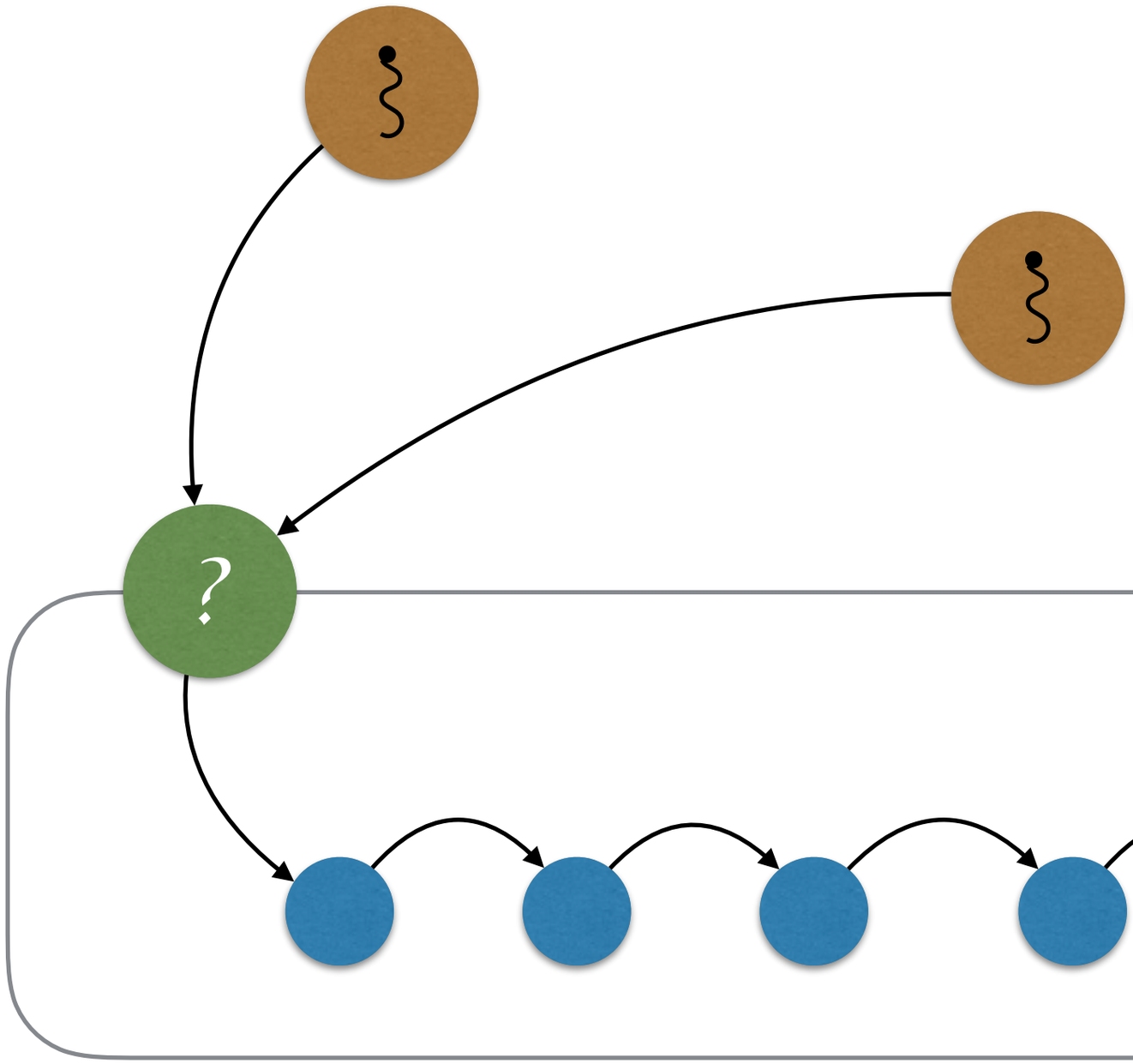
Grouping Messages



Grouping Locked operations

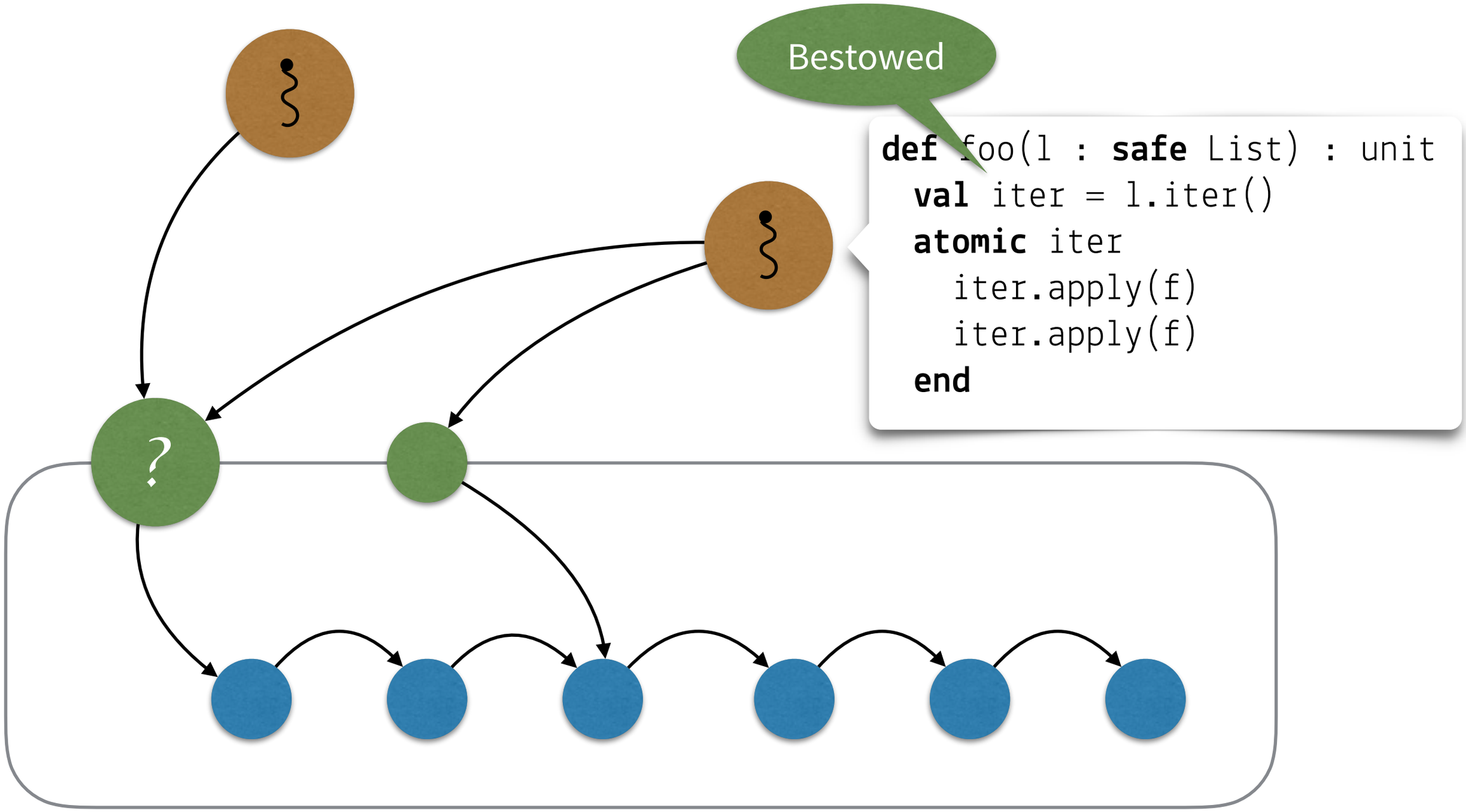


Polymorphic Concurrency Control




```
def foo(l : safe List) : unit
  val iter = l.iter()
  atomic iter
    iter.apply(f)
    iter.apply(f)
end
```

Polymorphic Concurrency Control



Tack! Frågor?

Summary

- Actor isolation can be relaxed by **bestowing** encapsulated objects with activity

  - All accesses will be synchronised via the message queue of the owning actor

  - Actors do not need to know the implementation of their bestowed objects

  - A bestowed object does not need to know that it is bestowed

- The same kind of relaxed encapsulation works for locks

  - All accesses will be synchronised via the lock of the owning object

- **Also in the paper:**

  Atomic blocks to group operations
  Implementation sketches
  Polymorphic concurrency control