

SHEEP CLONING

Paley Li, Nicholas Cameron, and James Noble

Object cloning

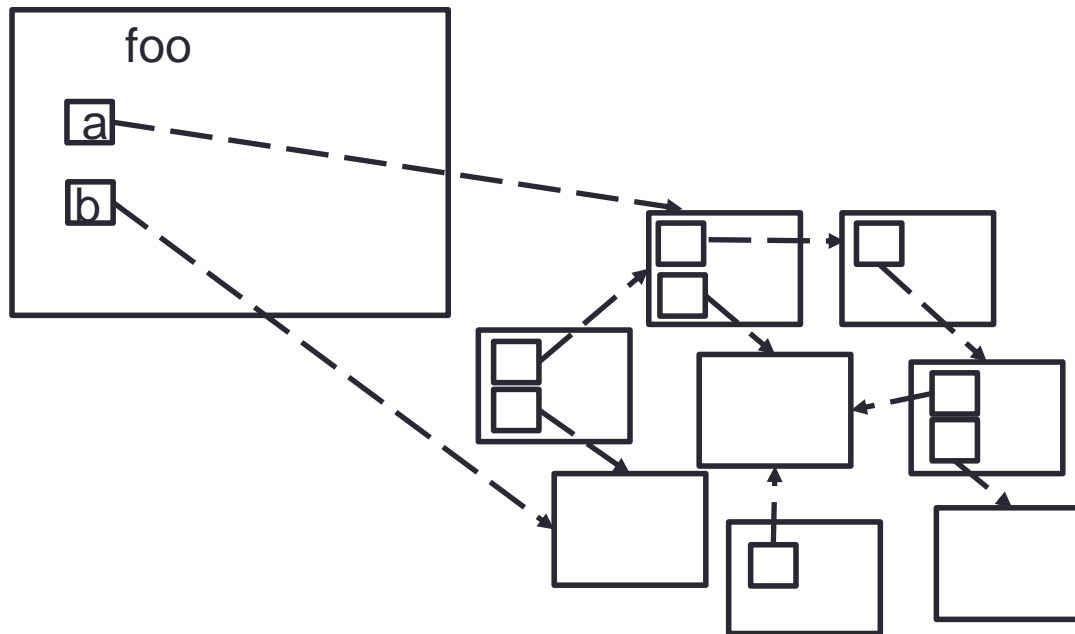
- How do you do object cloning?

Shallow cloning

- Copies an object and alias the references in that object.

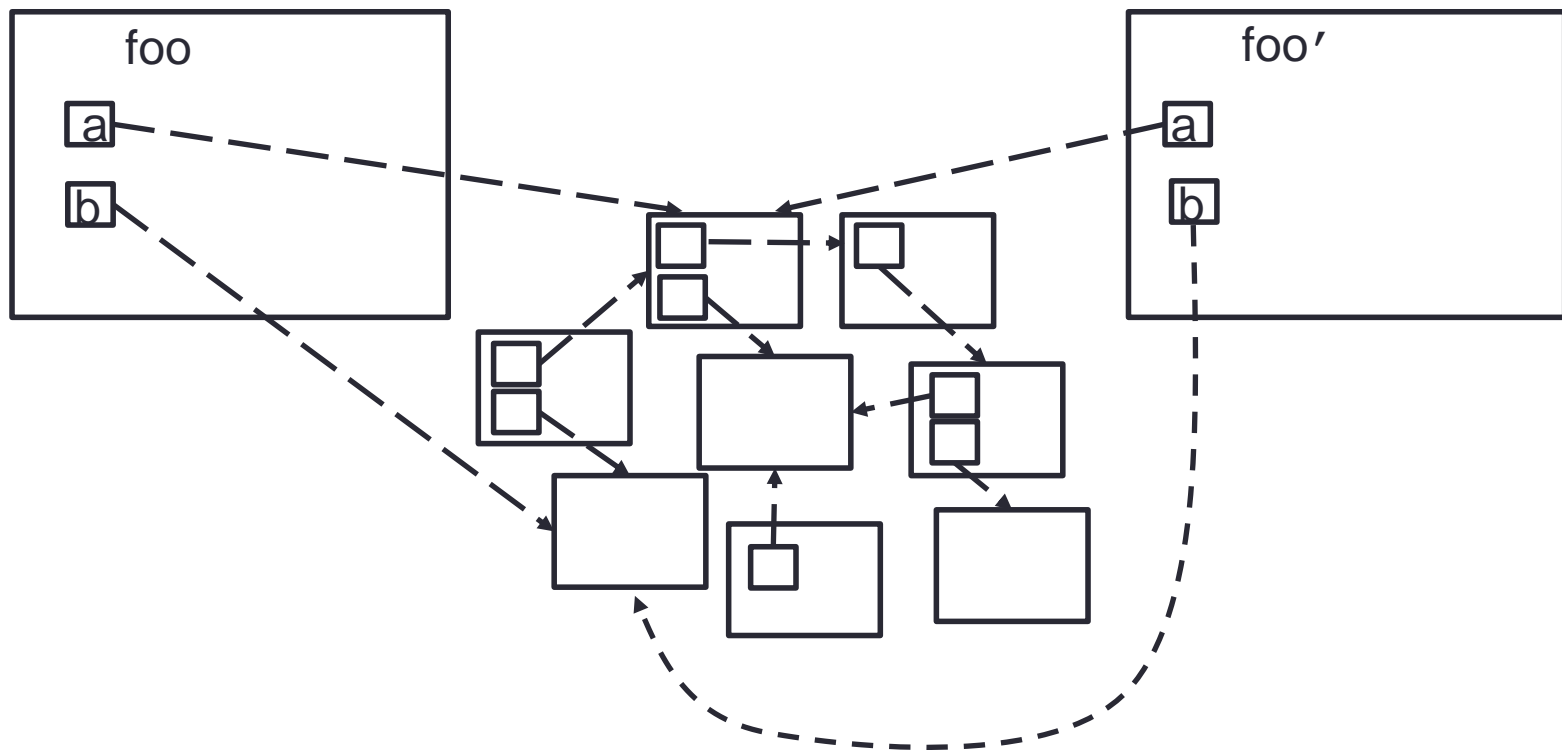
Shallow cloning

- Copies an object and alias the references in that object.



Shallow cloning

- Copies an object and alias the references in that object.

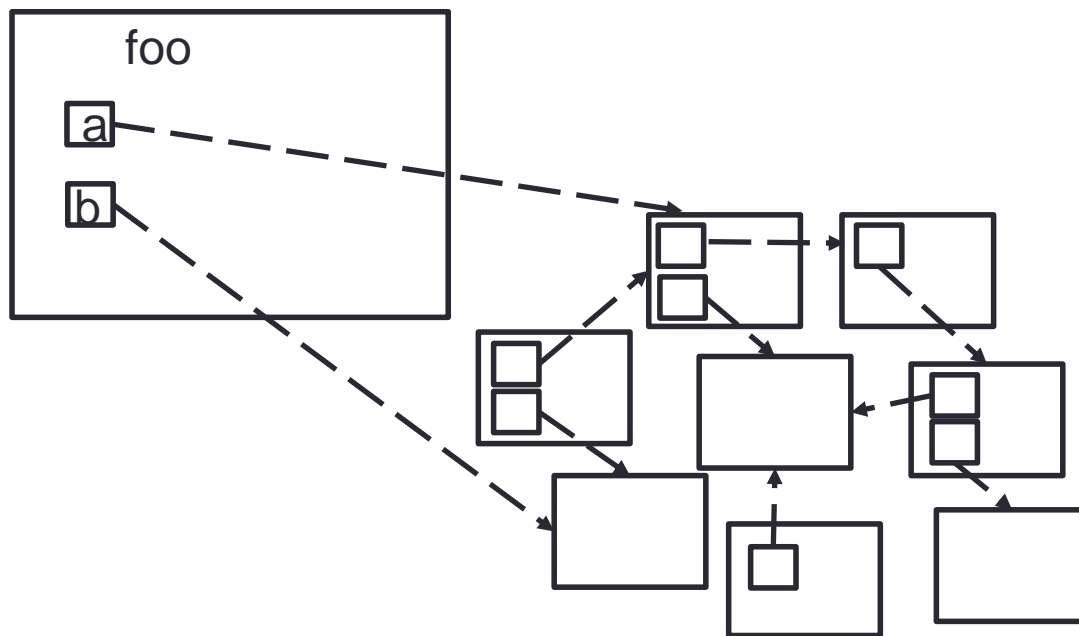


Deep cloning

- Copies the object and its referenced objects.

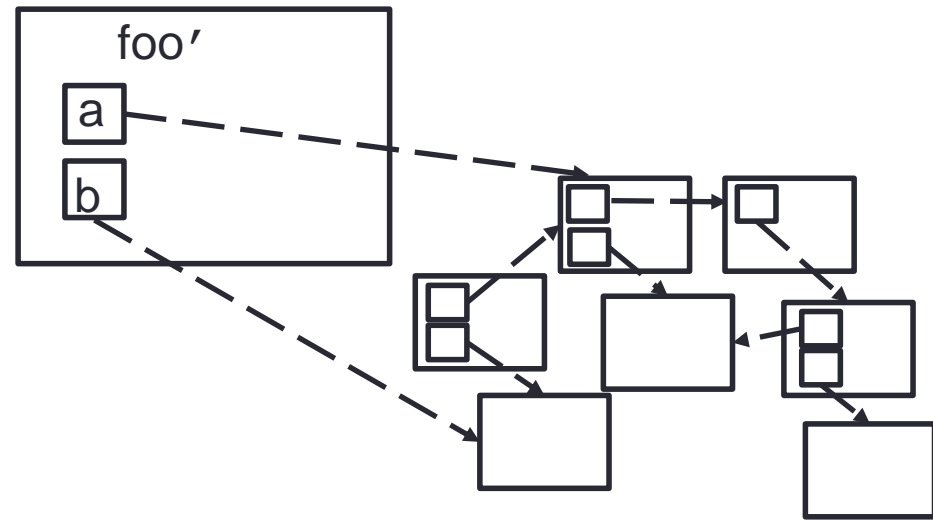
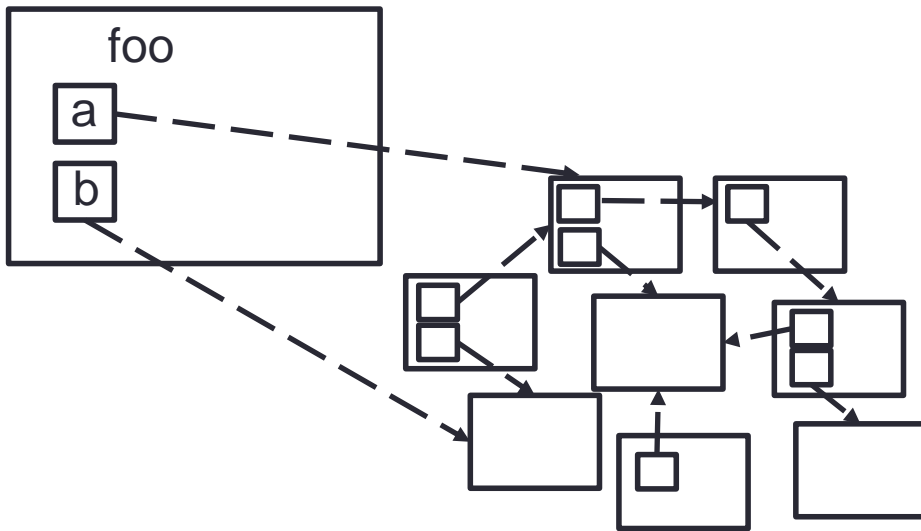
Deep cloning

- Copies the object and its referenced objects.



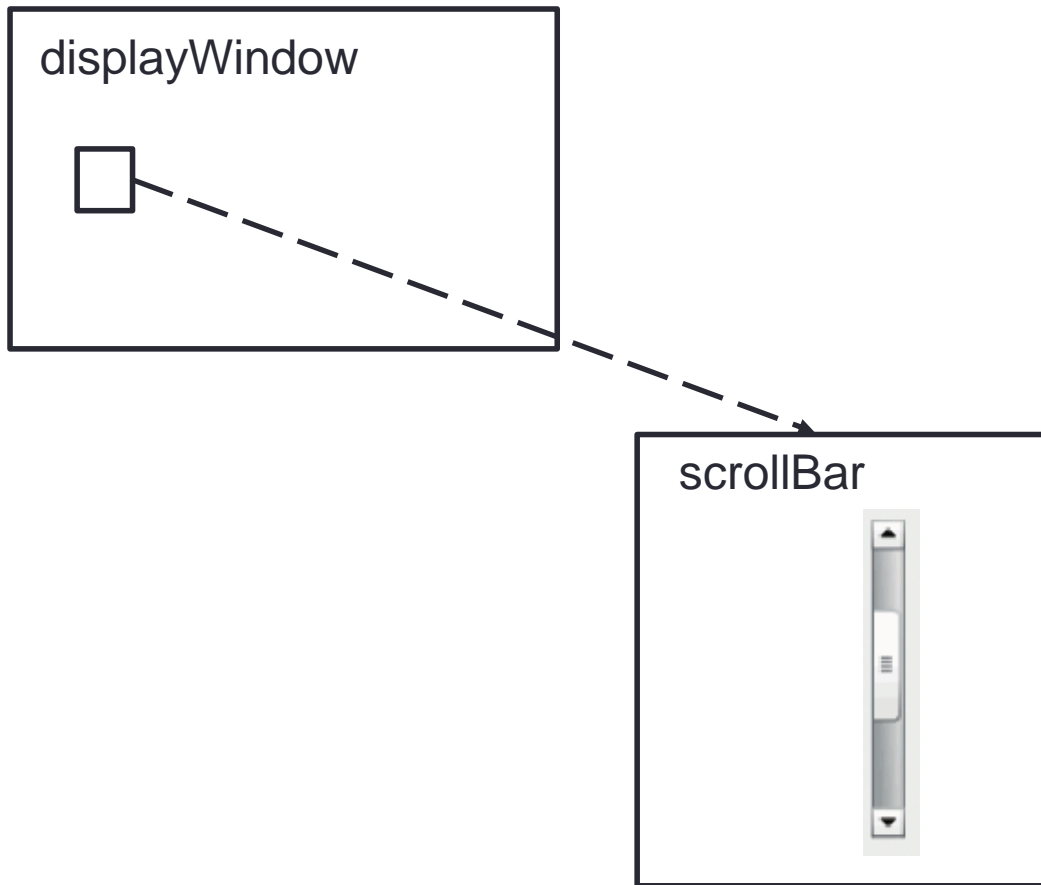
Deep cloning

- Copies the object and its referenced objects.



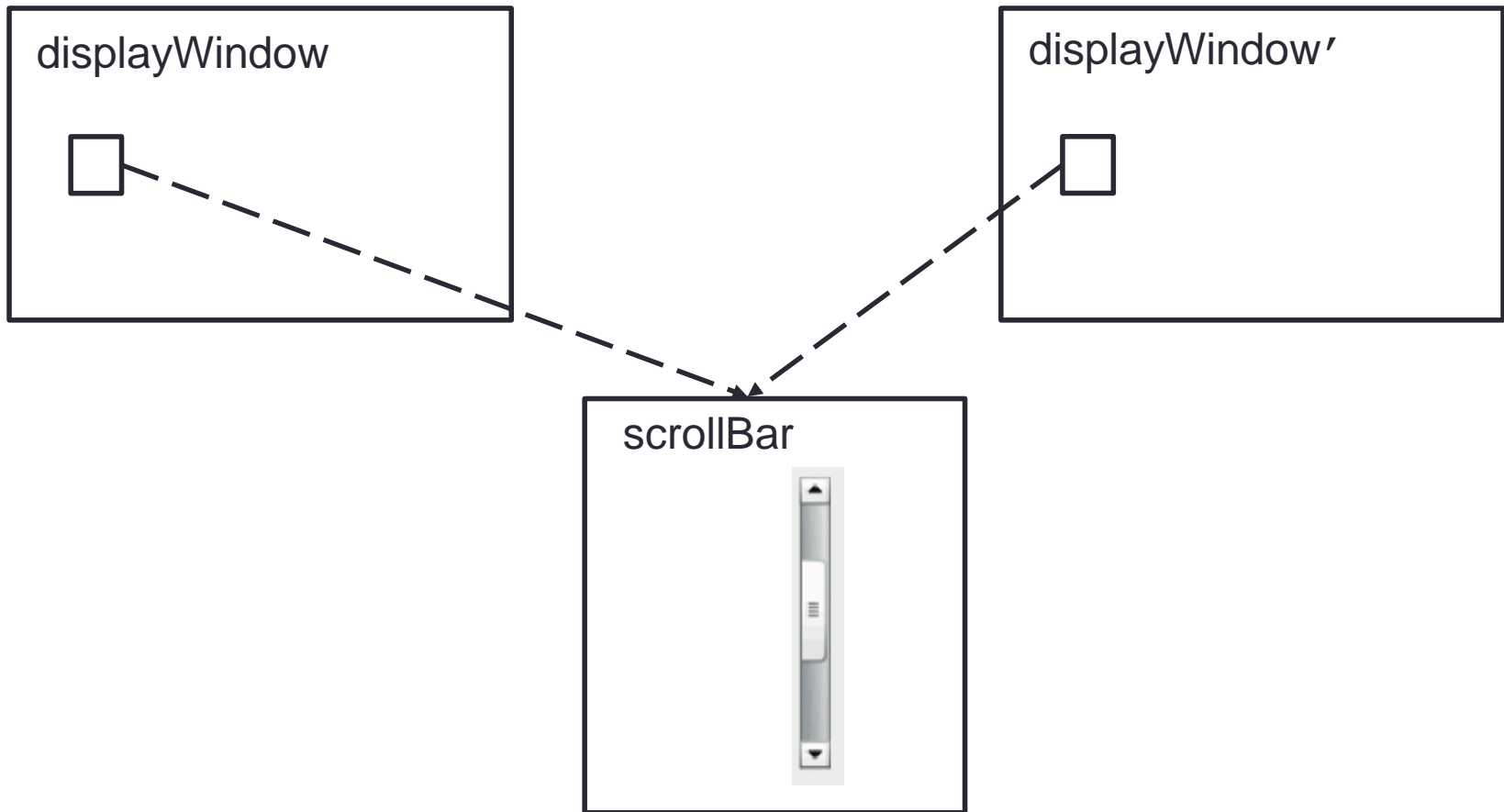


- Shallow cloning is too shallow





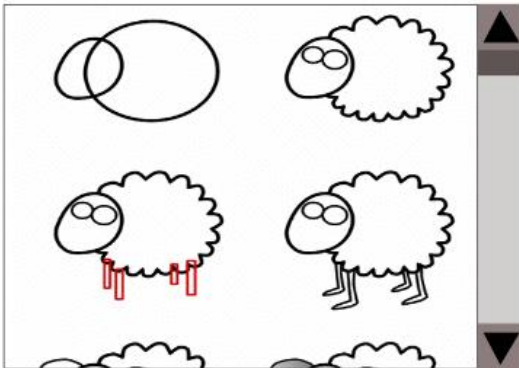
- Shallow cloning is too shallow



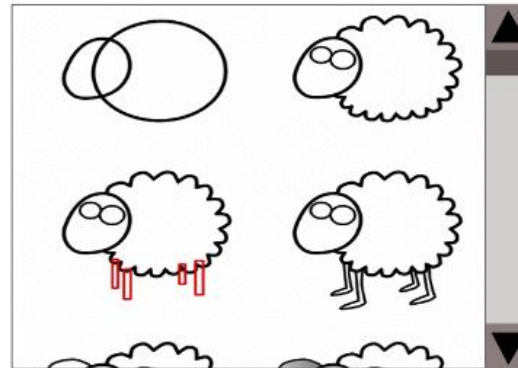


- Shallow cloning is too shallow

displayWindow



displayWindow'



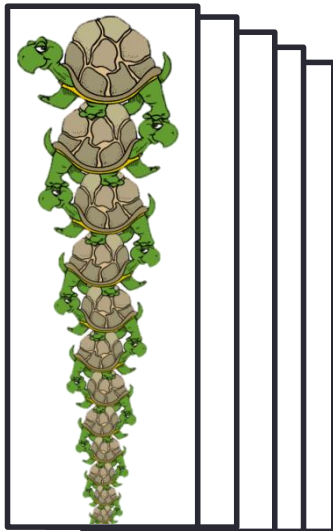


- Deep cloning is too deep

displayWindow

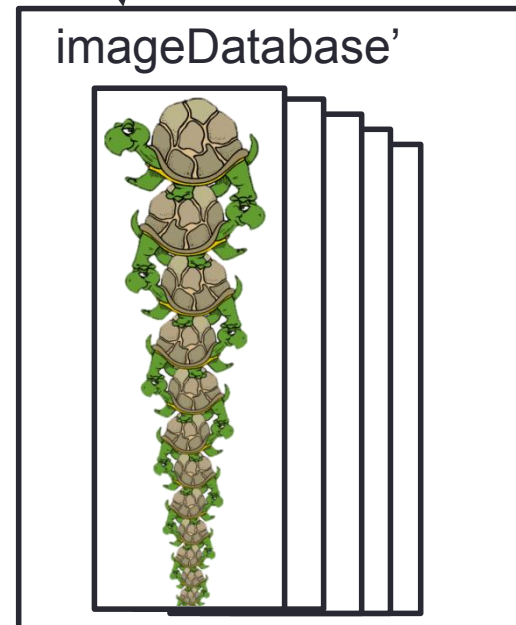
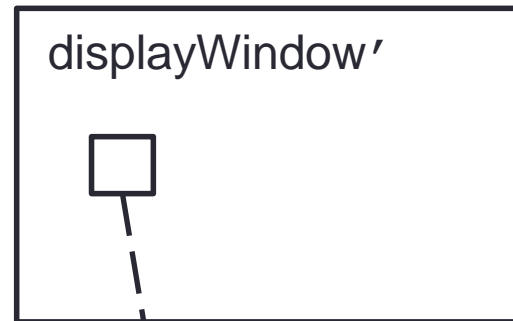
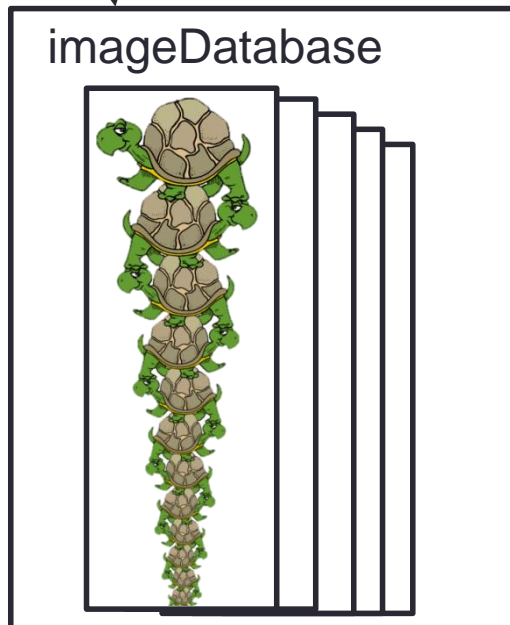


imageDatabase





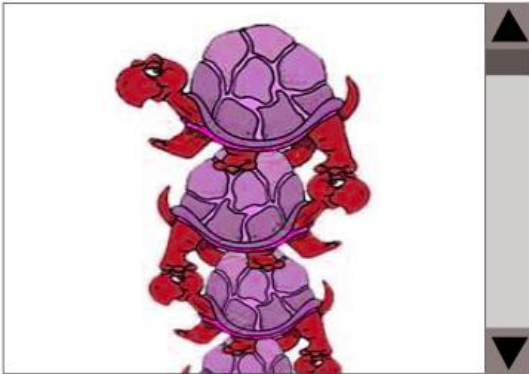
- Deep cloning is too deep



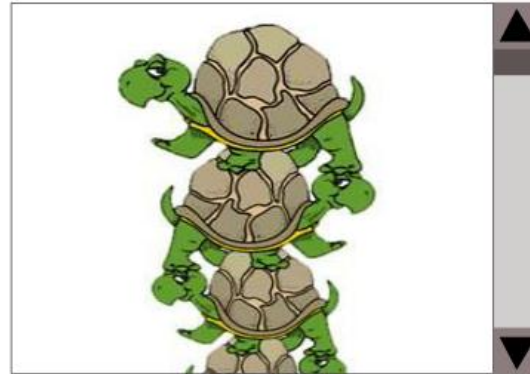


- Deep cloning is too deep

displayWindow



displayWindow'



Common practices

- Cloning in Java (Cloneable) and C# (ICloneable):
 - Default clone() method is shallow.
 - Defining deep cloning is inconvenient and prone to bugs.
 - Requires type casting.

```
class Foo implements Cloneable
{
    public Object clone(){
        try{
            return super.clone();
        }
        catch( CloneNotSupportedException e )
        {
            return null;
        }
    }
}
```

Common practices

- Cloning in C++ :
 - Copy constructors and assignment operators.
- Cloning in Eiffel :
 - Inherit shallow and deep cloning from the ANY class.

```
class C feature
  ...
end

class D feature
  x: C
  y: expanded C

  test is
    do
      x := y -- forbidden
      x := clone(y)
      x := deep_clone(y)
      x.clone(y)
      x.deep_clone(y)
    end
  end
end
```


Common practices

- Most practices still suffer from the flaws of shallow and deep cloning.
- Not automated.
 - “Programmer knows best” - they have to define their own cloning.
- What if we have the information to produce more sensible clones, but had overlooked it?

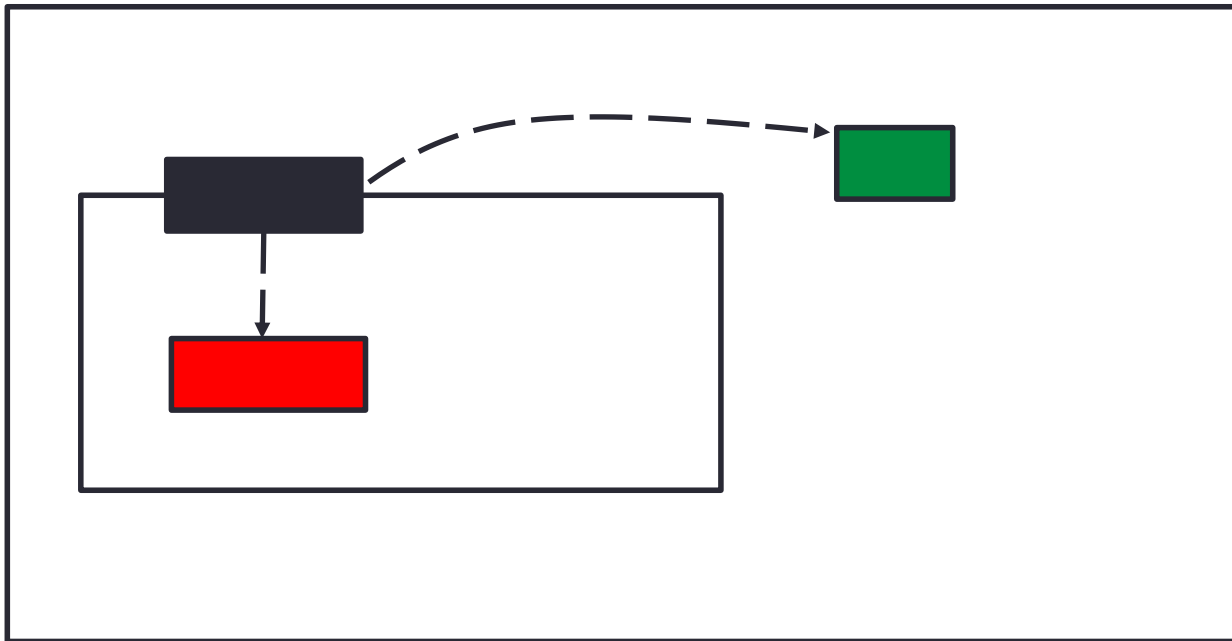


- The ideal model

- We aim to formalise a cloning model that is just right.
- It needs to be able to identify areas that are “important” to an object.
- Only copy those “important” areas.

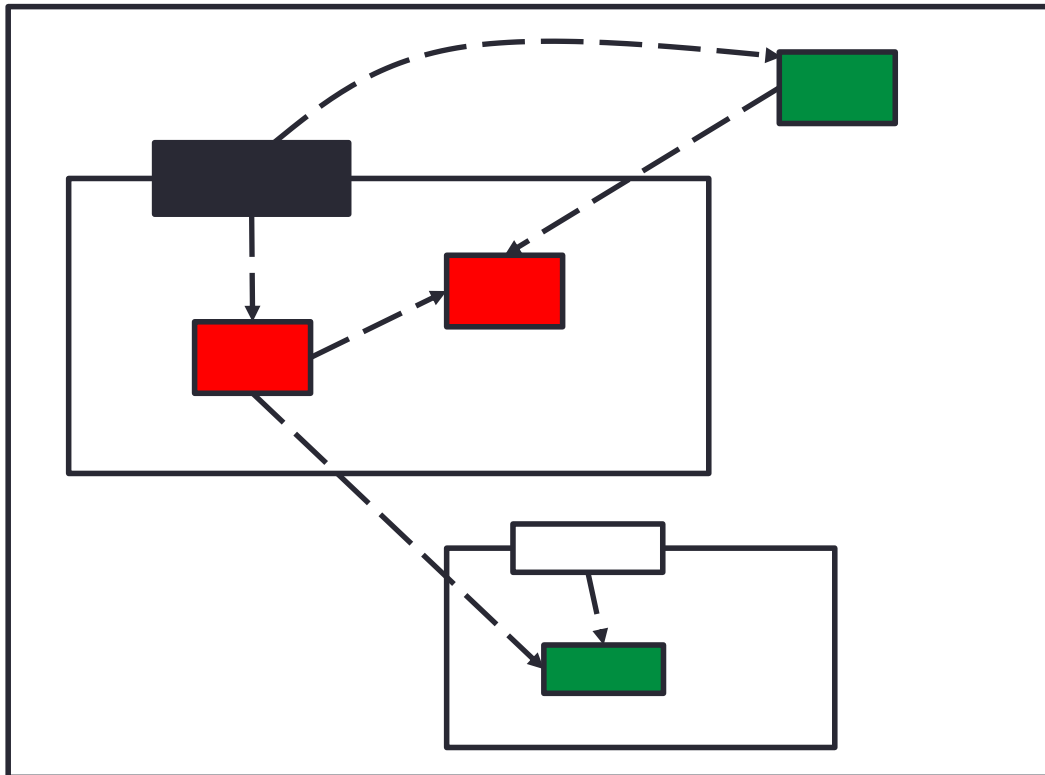
Ownership Types

- Ownership types enforce a hierarchical topology over the heap.



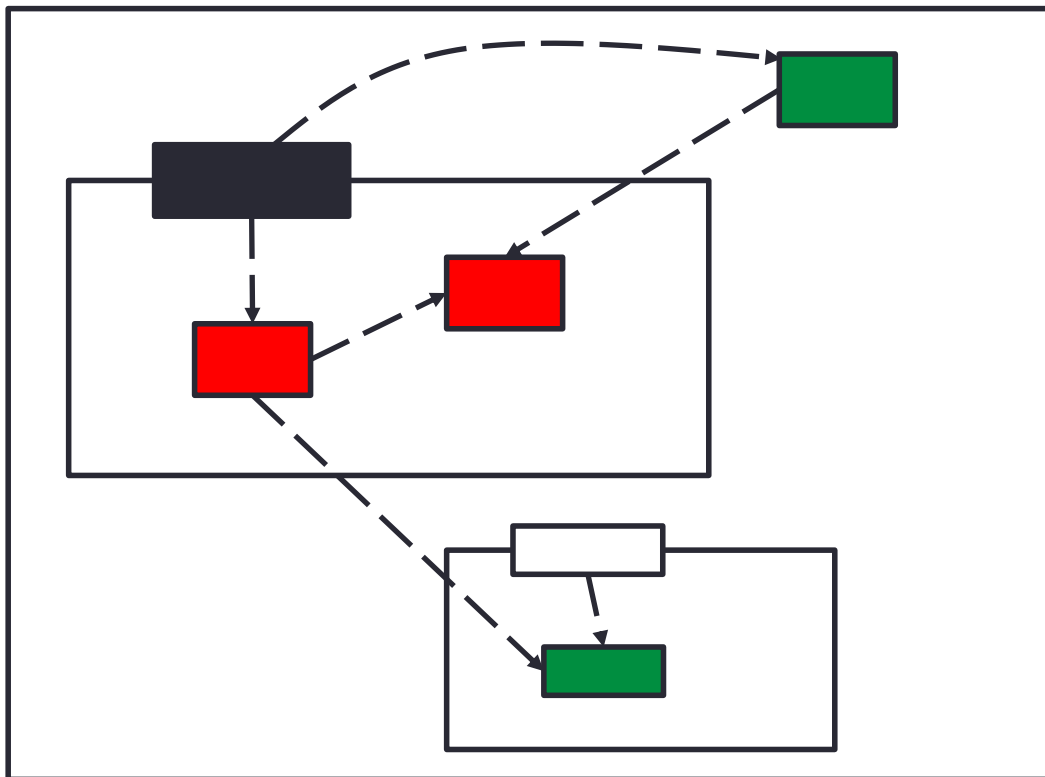
Ownership Types

- Context is the formal set of objects owned by an object.
- Representation is the set of objects which are conceptually part of an object.



Ownership Types

- Context is the formal set of objects owned by an object.
- Representation is the set of objects which are conceptually part of an object.

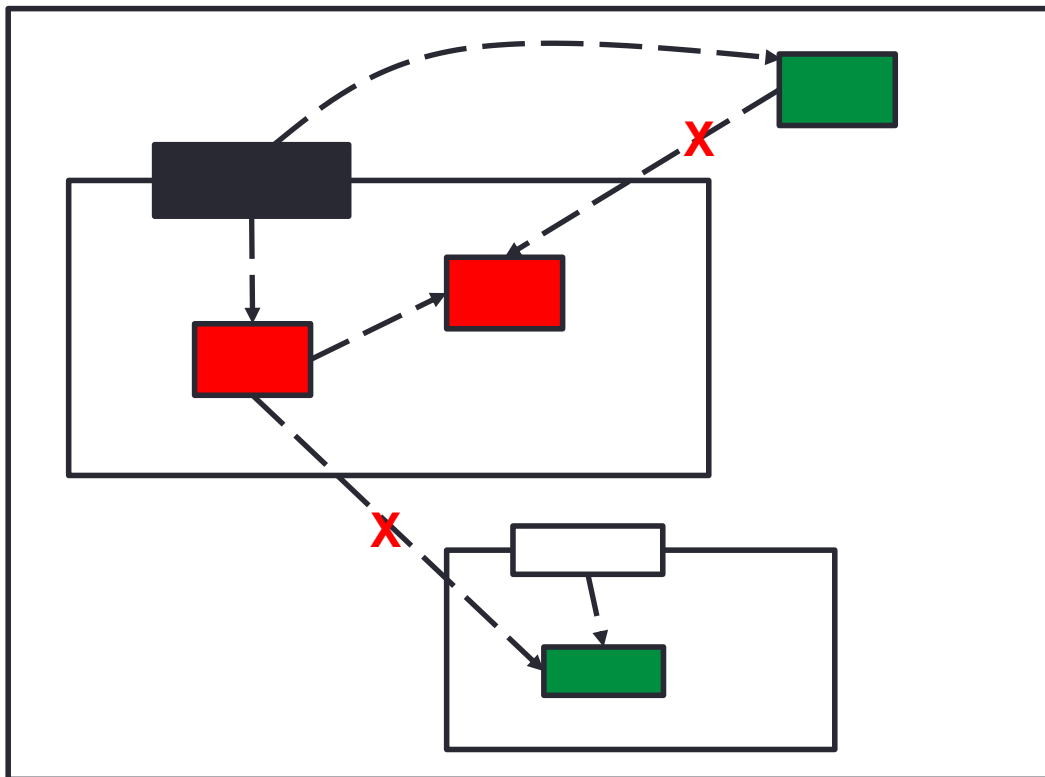


Representation = context =



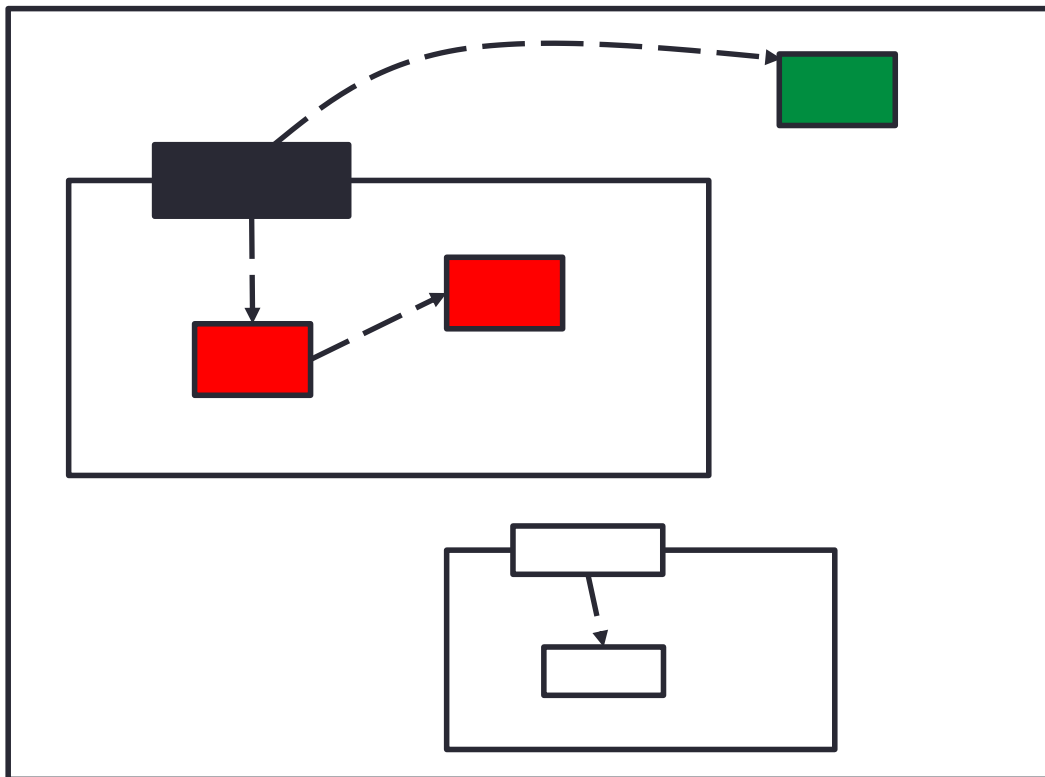
Deep Ownership

- All reference paths to an object must pass through that object's owner.
- Also known as owners-as-dominators.



Deep Ownership

- All reference paths to an object must pass through that object's owner.
- Also known as owners-as-dominators.



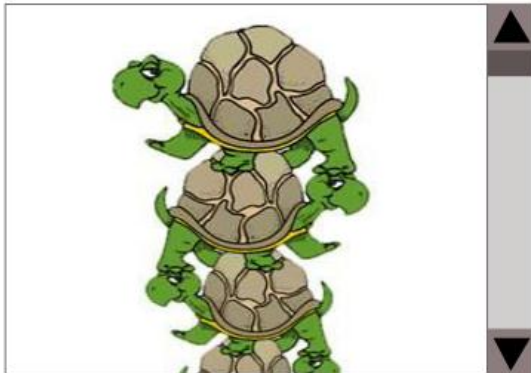
Sheep = Shallow + Deep

- Utilises ownership types to identify the “important bits” of each object.
- Cloning an object’s representation:
 - Copies every object **inside** the object’s context.
 - Aliases every reference to objects **outside** the object’s context.



- Sheep cloning is just right!

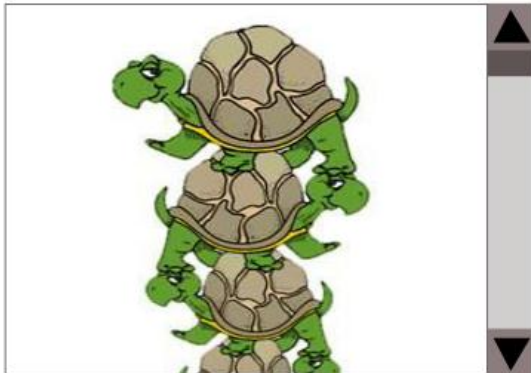
displayWindow



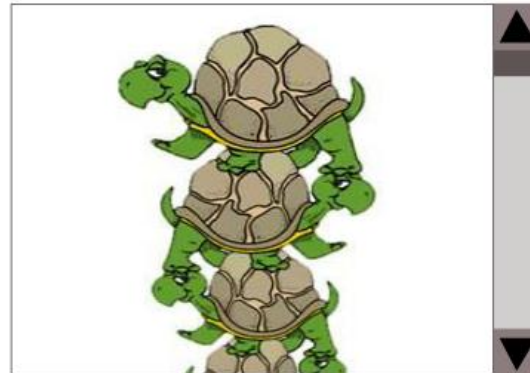


- Sheep cloning is just right!

displayWindow



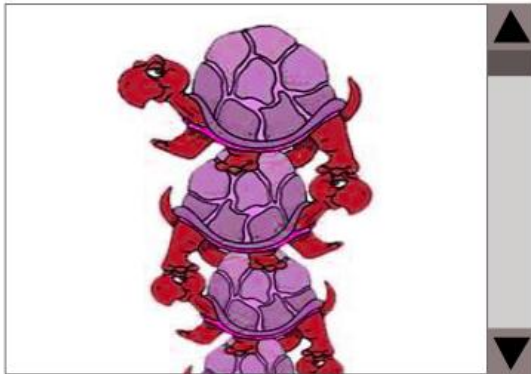
displayWindow'



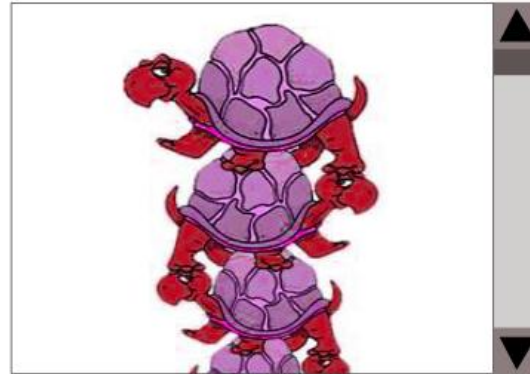


- Sheep cloning is just right!

displayWindow



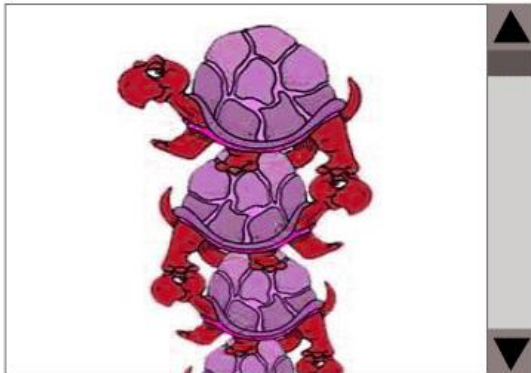
displayWindow'



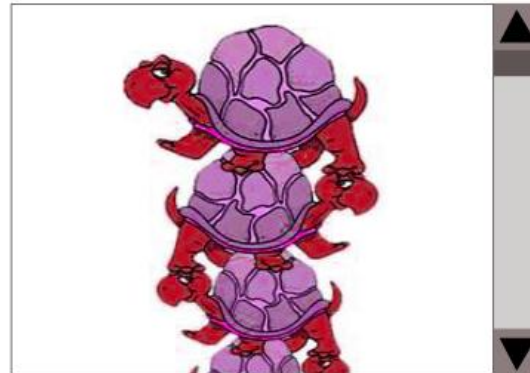


- Sheep cloning is just right!

displayWindow



displayWindow'



Sheep cloning

- We have formalised sheep cloning in an ownership system with deep ownership.
- We have proved soundness and an assortment of correctness property of our formalism.

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq o_u \rangle \{ \overline{N} f; \overline{M} \}$ *class declarations*
 $M ::= N m(N x) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \frac{\gamma : T, o : \top}{\gamma : T, o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$

$v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*

$\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*

$\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*

$\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*

x *variables*

ι *object address*

err *errors*

null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \frac{\gamma : T, \overline{o} : \top}{\gamma : T, \overline{o} : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o} \preceq \overline{o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, \overline{f \rightarrow v}\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq o_u \rangle \{ \overline{N} f; \overline{M} \}$ *class declarations*
 $M ::= N m(N x) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$Q ::= \text{class } C \langle \overline{o_l} \preceq x \preceq \overline{o_u} \rangle \{ \overline{N} \text{ f}; \overline{M} \}$ *class declarations*
 $M ::= N \text{ m}(N \text{ x}) \{ \text{return } e; \}$ *method declarations*

$T ::= N \mid \top$ *type*
 $N ::= o : C \langle \overline{o} \rangle$ *class type*
 $o ::= \gamma \mid \text{world} \mid \text{owner}$ *owners*

$e ::= \text{null} \mid \gamma \mid \gamma.f \mid \gamma.f = e \mid \gamma.m(e)$ *expressions*
 $\quad \mid \text{new } o : C \langle \overline{o} \rangle \mid \text{sheep}(e) \mid v$
 $v ::= \text{null} \mid \iota$ *values*

$\gamma ::= x \mid \text{this} \mid \iota$ *expression variables and addresses*
 $\Gamma ::= \overline{\gamma : T}, \overline{o : \top}$ *variable environments*
 $\mathcal{E} ::= \overline{o \preceq o}$ *owners environments*

$\mathcal{H} ::= \overline{\iota \rightarrow \{N, f \rightarrow v\}}$ *heaps*
 $\text{map} ::= \overline{\{\iota \rightarrow \iota\}}$ *map*

$x \preceq o$ *owners relation*
 x *variables*
 ι *object address*
 err *errors*
 null *null expression*

A touch of formal

$$\frac{\mathcal{E}; \Gamma \vdash e : T}{\mathcal{E}; \Gamma \vdash \text{sheep}(e) : T}$$

(T-SHEEP)

A touch of formal

$$\frac{\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

A touch of formal

$$\frac{\text{SheepAux}(\textcircled{v}, \textcircled{v}, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(\textcircled{v}); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

Original object

A touch of formal

$$\frac{\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

Original object

A touch of formal

$$\frac{\text{SheepAux}(\overset{\text{Original object}}{\circlearrowleft} v, \overset{\text{Original object}}{\circlearrowright} v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(\overset{\text{Original object}}{\circlearrowleft} v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

A touch of formal

$$\frac{\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

Original heap

A touch of formal

Map

$$\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \{\iota \rightarrow \iota'\}$$

$$\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'$$

(R-SHEEP)

A touch of formal

$$\frac{\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

Sheep clone

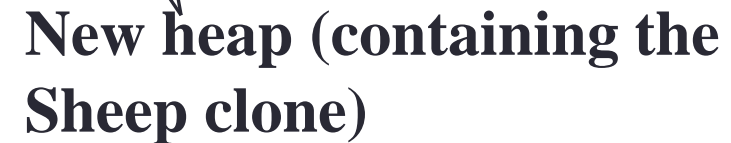


A touch of formal

$$\frac{\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \overline{\{\iota \rightarrow \iota'\}}}{\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

New heap (containing the Sheep clone)

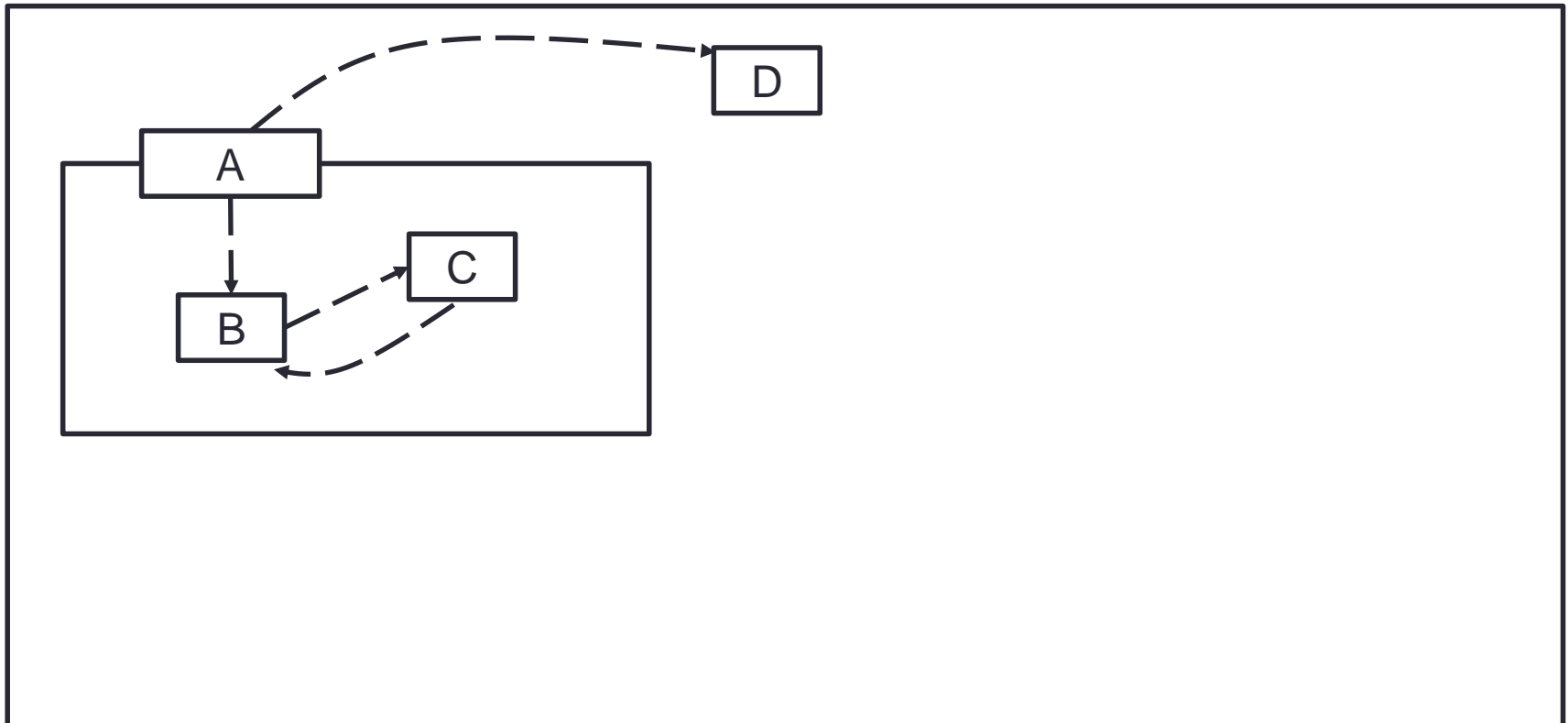


A touch of formal

- **SheepAux function:**
 - **R-SheepInside:** Copies the object if it is inside the original object.
 - **R-SheepOutside:** Creates an alias to the object if the object is outside the original object.
 - **R-SheepRef:** Creates a reference to an existing Sheep clone of an object using the **Map**.
 - **R-SheepNull:** Returns a null, when Sheep cloning a null.

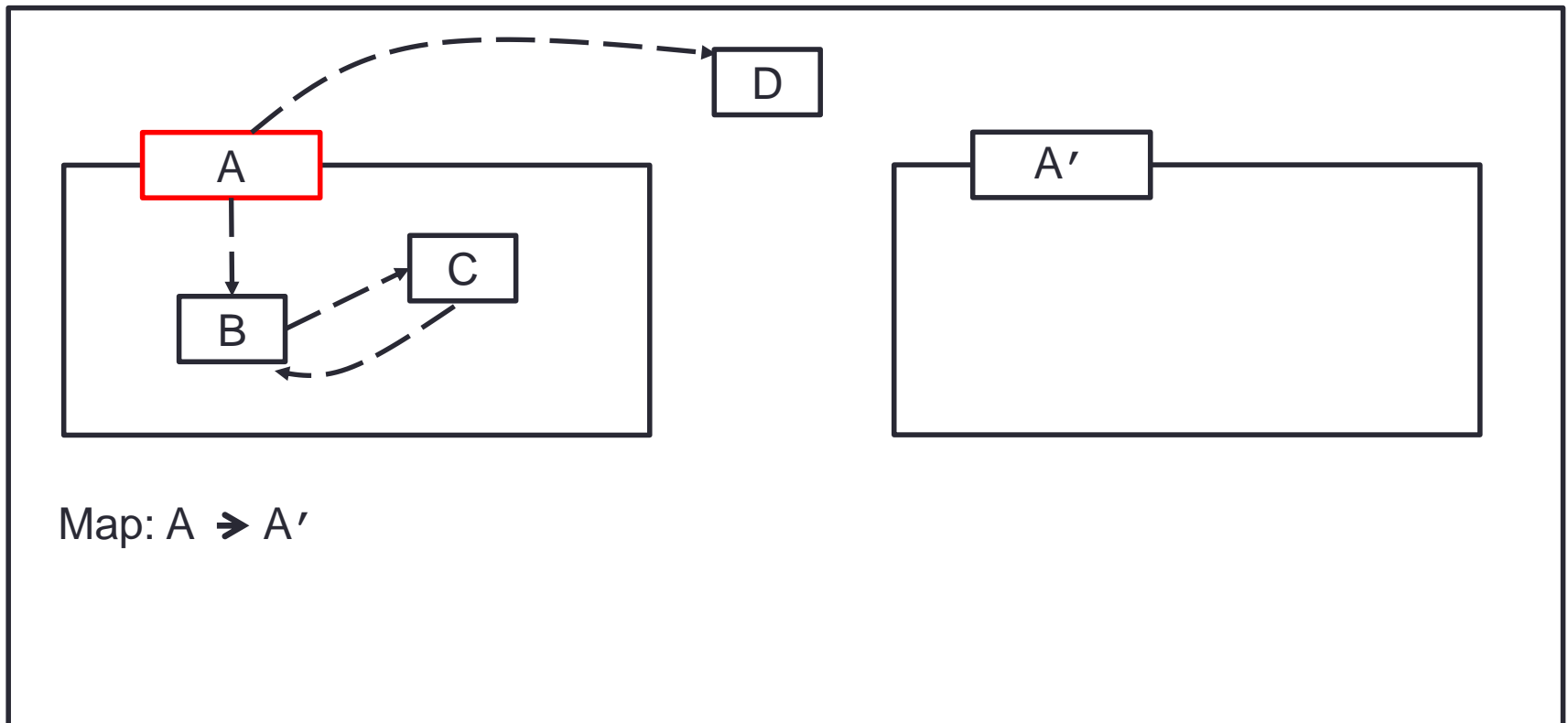
Can we clone it?

- Lets Sheep clone object A.



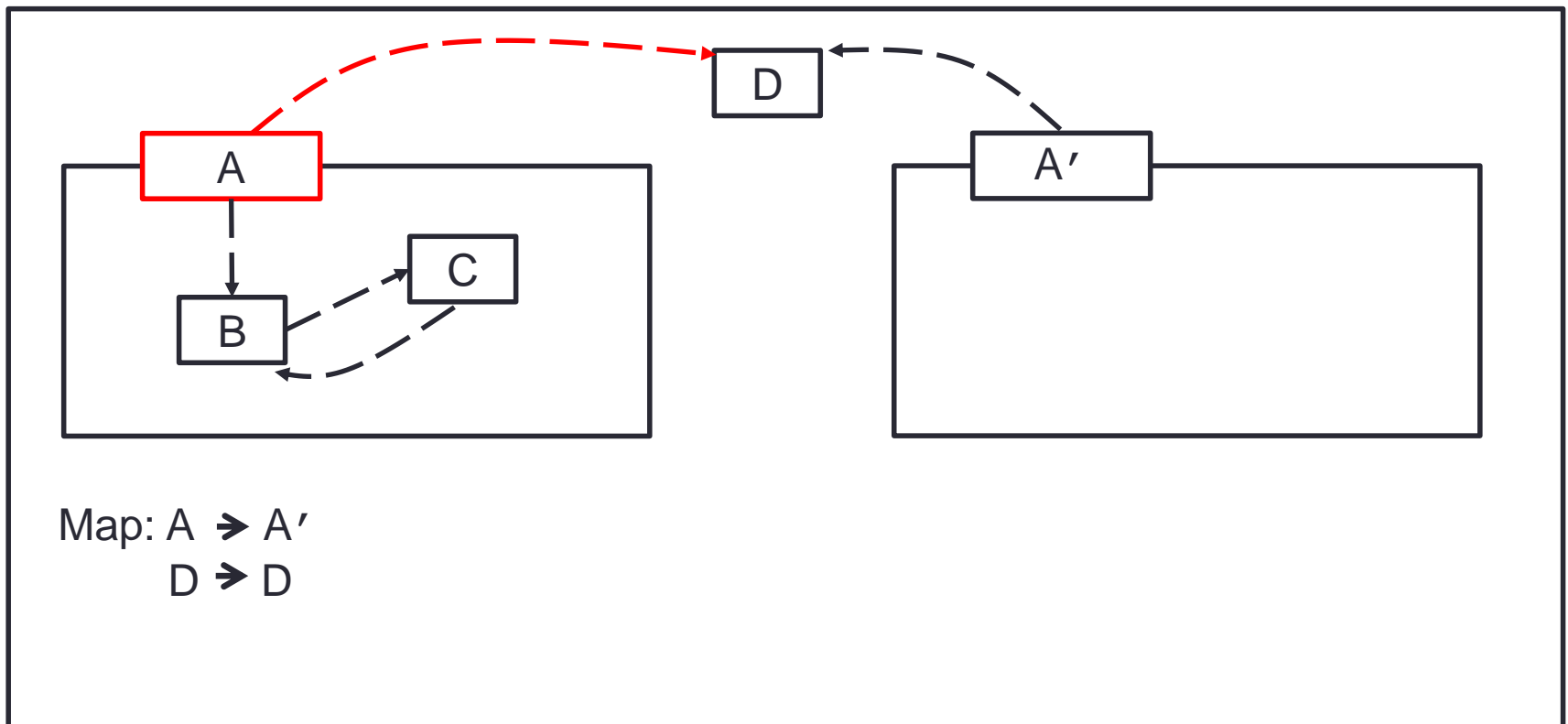
Can we clone it?

- `R-SheepInside` creates the object `A'` by copying `A`.



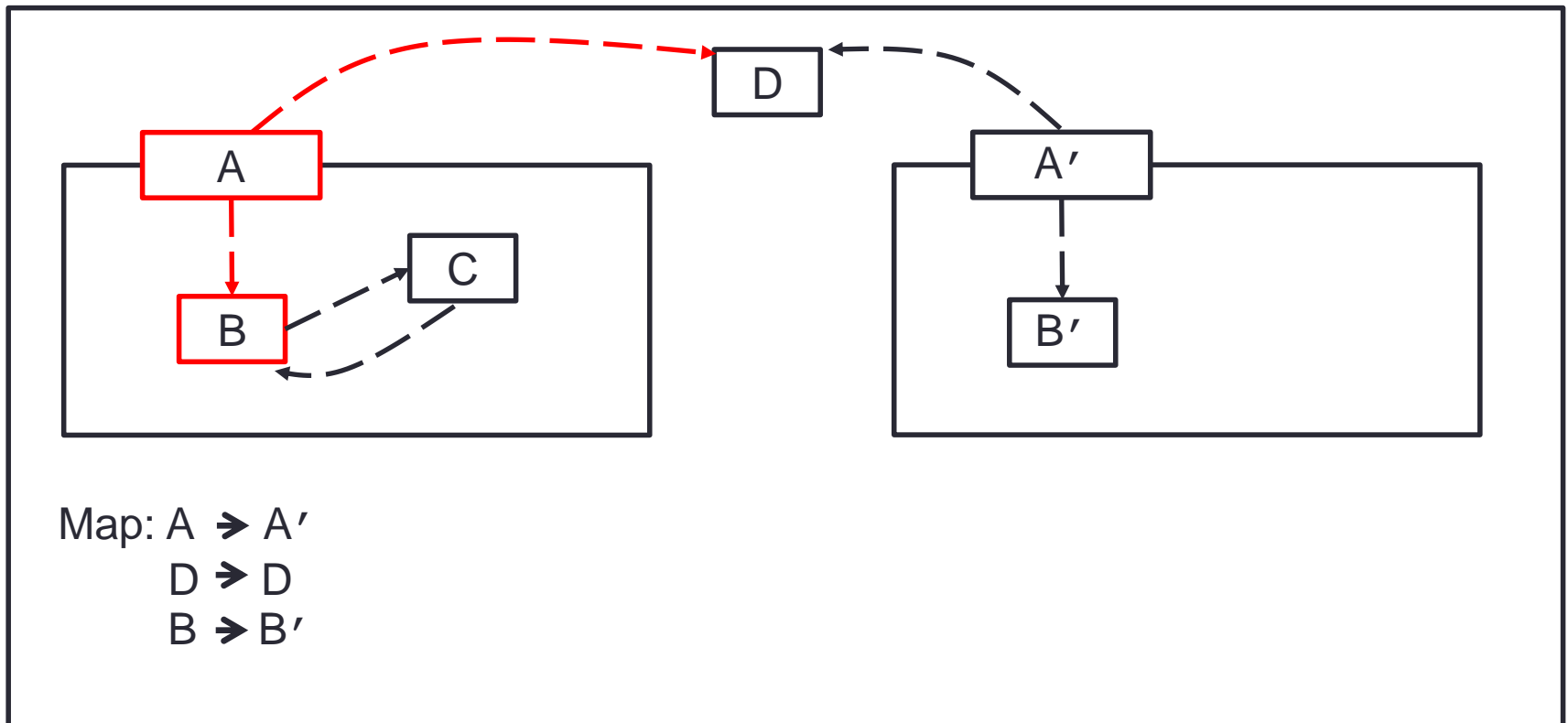
Can we clone it?

- `R-SheepOutside` creates an alias to `D`.



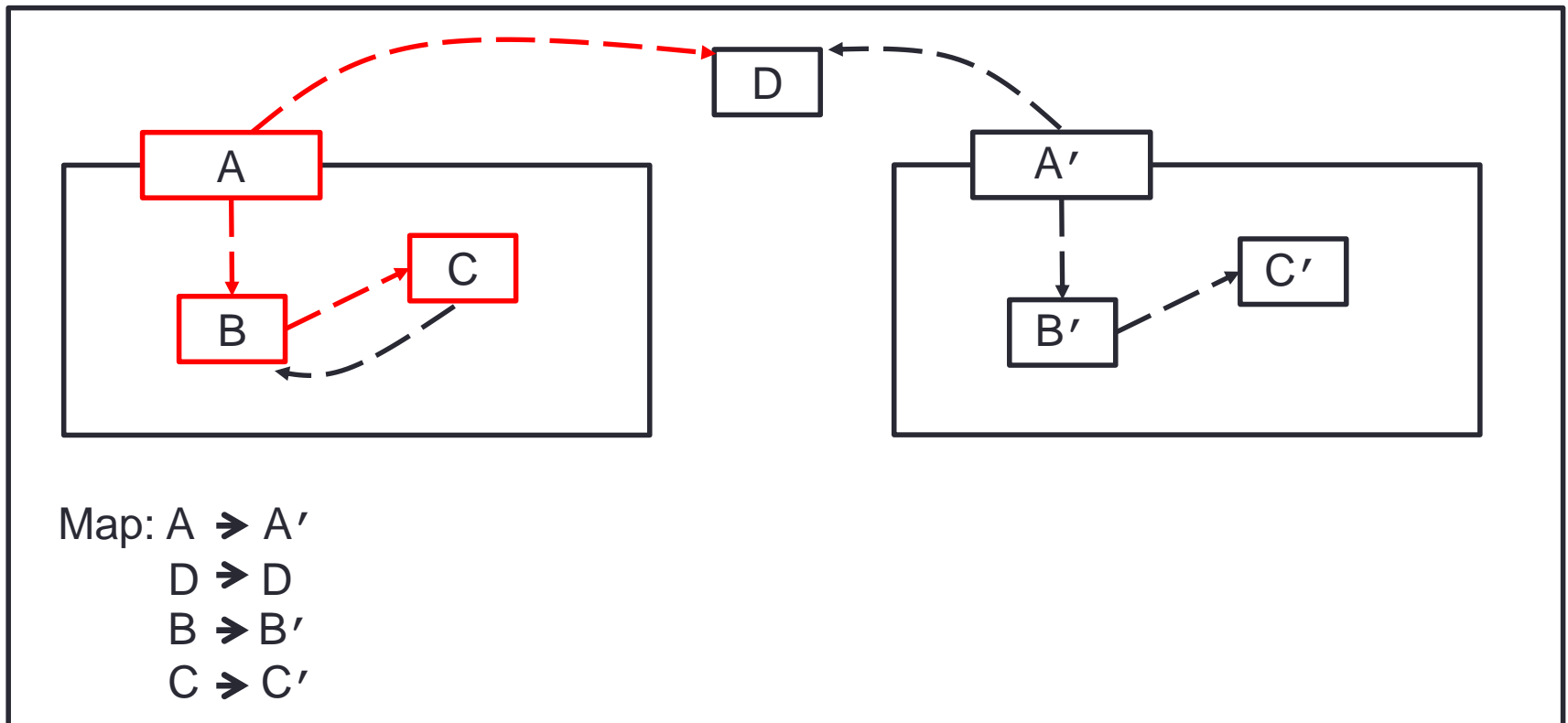
Can we clone it?

- `R-SheepInside` creates the object `B'` by copying `B`.



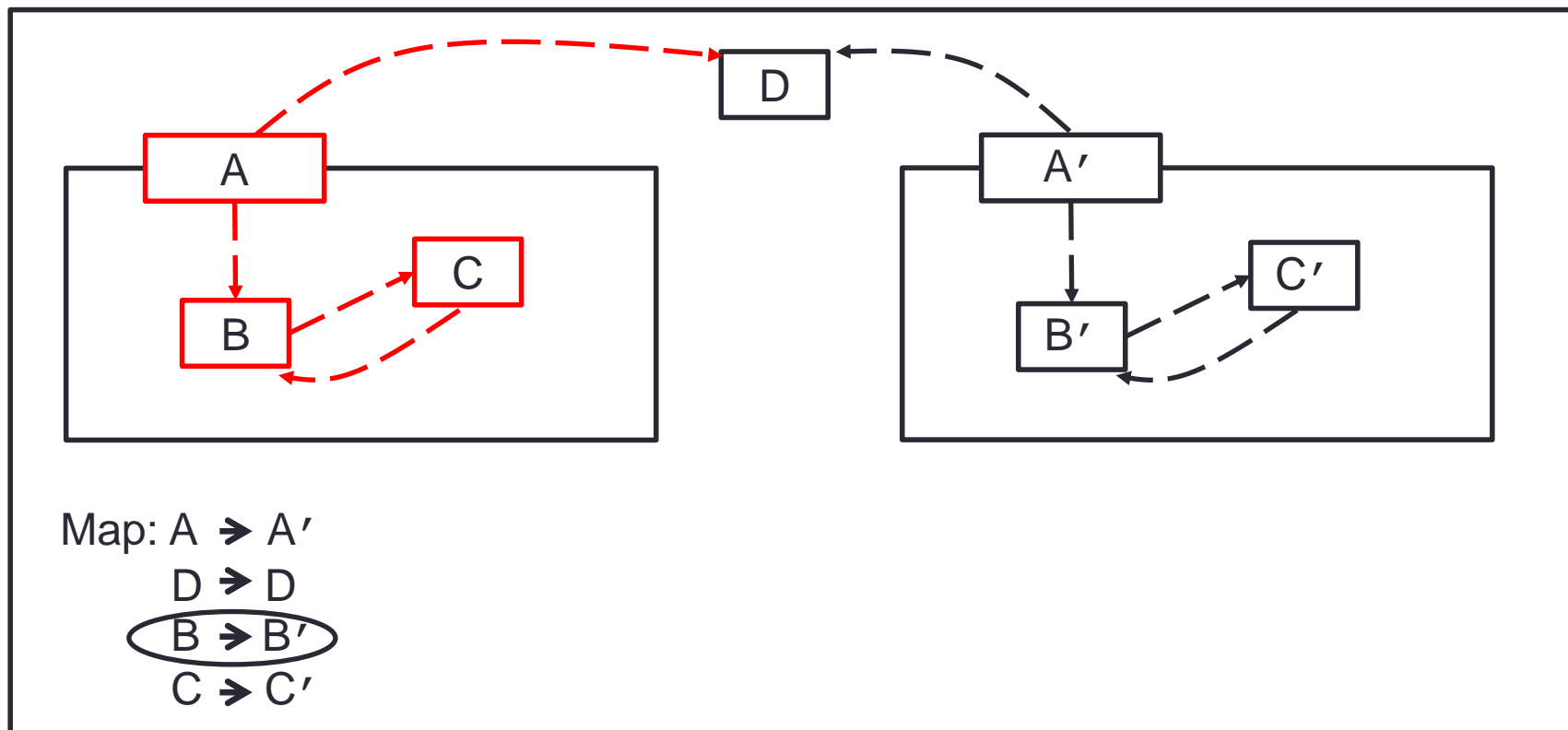
Can we clone it?

- `R-SheepInside` creates the object `C'` by copying `C`.



.... Yes we can!

- R-SheepRef creates the reference from object C' to object B' using the map.



Proving the formalism

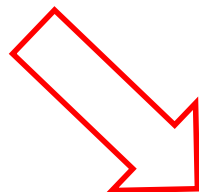
Subject reduction case: R-SHEEP.

For all \mathcal{H} , \mathcal{H}' , v , v' , and N , if $\mathcal{H} \vdash \text{sheep}(v) : N$ and $\vdash \mathcal{H} \text{ OK}$ and $\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'$ then $\mathcal{H}' \vdash v' : N$ and $\vdash \mathcal{H}' \text{ OK}$.

Proving the formalism

Subject reduction case: R-SHEEP.

For all $\mathcal{H}, \mathcal{H}', v, v'$, and N , if $\mathcal{H} \vdash \text{sheep}(v) : N$ and $\vdash \mathcal{H}$ OK and $\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'$ then $\mathcal{H}' \vdash v' : N$ and $\vdash \mathcal{H}'$ OK.



$$\frac{\text{SheepAux}(v, v, \mathcal{H}, \emptyset) = v'; \mathcal{H}'; \{\overline{\iota \rightarrow \iota'}\}}{\text{sheep}(v); \mathcal{H} \rightsquigarrow v'; \mathcal{H}'}$$

(R-SHEEP)

Proving the formalism

Lemma: Mapped type preserves type well-formedness.

For all \mathcal{H} , map , and N , if $\vdash \mathcal{H} \text{ OK}$, $\mathcal{H} \vdash \text{map} \text{ OK}$, and $\mathcal{H} \vdash N \text{ OK}$ then $\mathcal{H} \vdash \text{map}(N) \text{ OK}$.

$$\frac{\text{map} = \overline{\{\iota \mapsto \iota'\}}}{\text{map}(N) = \overline{[\iota'/\iota] N}}$$

Proving the formalism

Lemma: Mapped type preserves type well-formedness.

For all \mathcal{H} , map , and N , if $\vdash \mathcal{H} \text{ OK}$, $\mathcal{H} \vdash \text{map} \text{ OK}$, and $\mathcal{H} \vdash N \text{ OK}$ then $\mathcal{H} \vdash \text{map}(N) \text{ OK}$.

$$\frac{\text{map} = \overline{\{\iota \mapsto \iota'\}}}{\text{map}(\textcircled{N}) = \overline{[\iota'/\iota]} N}$$

Proving the formalism

Lemma: Mapped type preserves type well-formedness.

For all \mathcal{H} , map , and N , if $\vdash \mathcal{H} \text{ OK}$, $\mathcal{H} \vdash \text{map} \text{ OK}$, and $\mathcal{H} \vdash N \text{ OK}$ then $\mathcal{H} \vdash \text{map}(N) \text{ OK}$.

$$\frac{\text{map} = \overline{\{\iota \mapsto \iota'\}}}{\text{map}(N) = \overline{[\iota'/\iota] N}}$$

Proving the formalism

Lemma: Mapped type preserves type well-formedness.

For all \mathcal{H} , map , and N , if $\vdash \mathcal{H} \text{ OK}$, $\mathcal{H} \vdash \text{map} \text{ OK}$, and $\mathcal{H} \vdash N \text{ OK}$ then $\mathcal{H} \vdash \text{map}(N) \text{ OK}$.

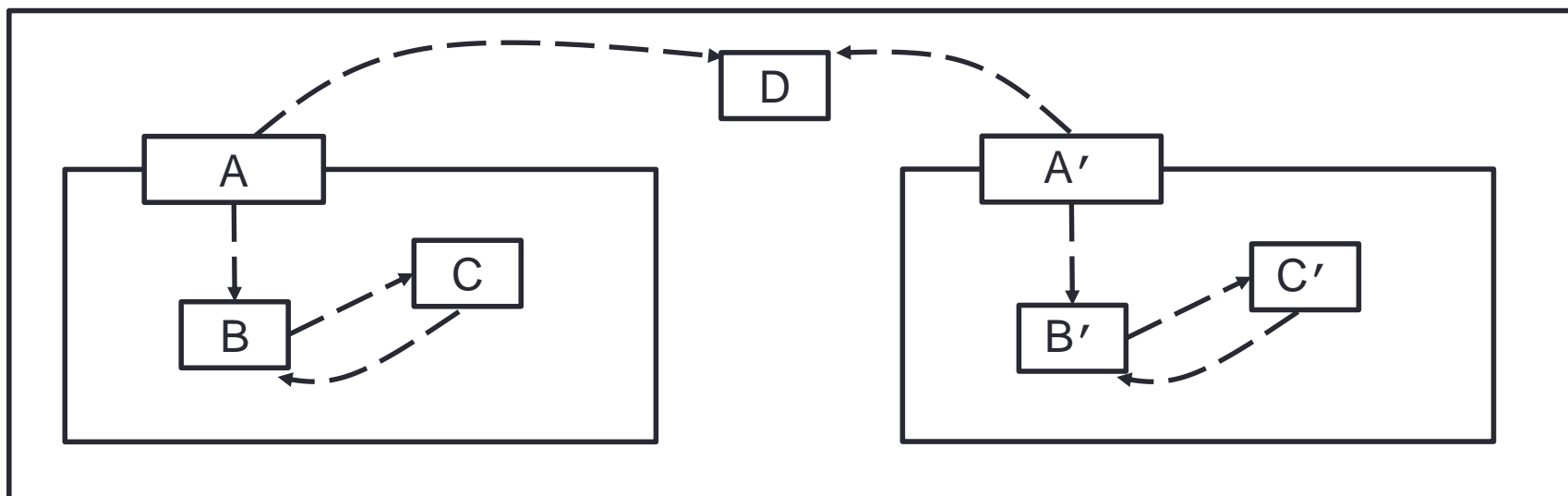
$$\frac{\text{map} = \overline{\{\iota \mapsto \iota'\}}}{\text{map}(N) = \overline{[\iota'/\iota] N}}$$

Correctness of the formalism

Correctness property: Sheep Cloning creates a new object.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ then $\iota' \notin \text{dom}(\mathcal{H})$ and $\iota \neq \iota'$.

Where: $\iota = A$
 $\iota' = A'$

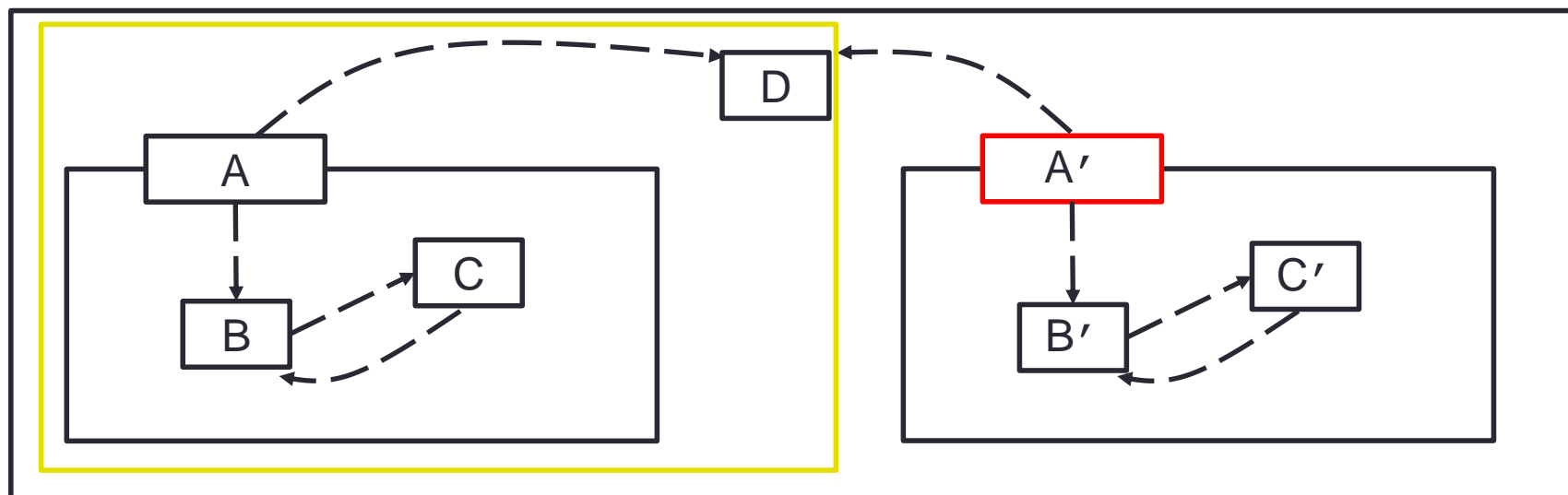


Correctness of the formalism

Correctness property: Sheep Cloning creates a new object.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , **if** $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ **then** $\iota' \notin \text{dom}(\mathcal{H})$ and $\iota \neq \iota'$.

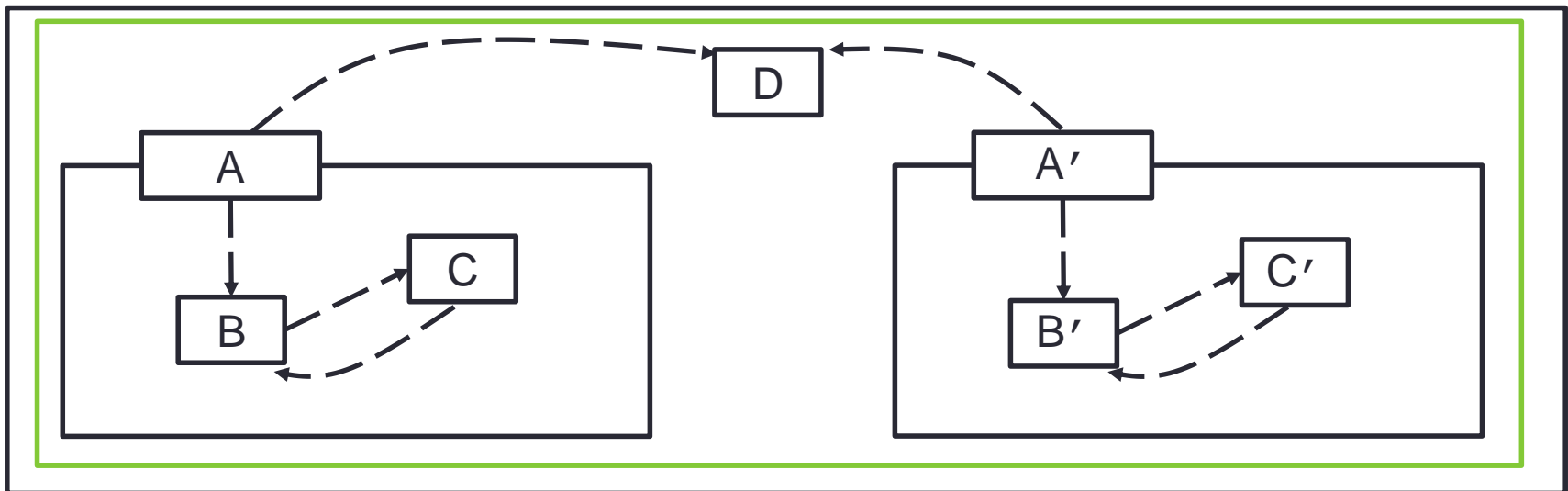
Where: $\iota = A$
 $\iota' = A'$



Correctness of the formalism

Correctness property: Sheep Cloning creates a sub-heap that contains the new object.

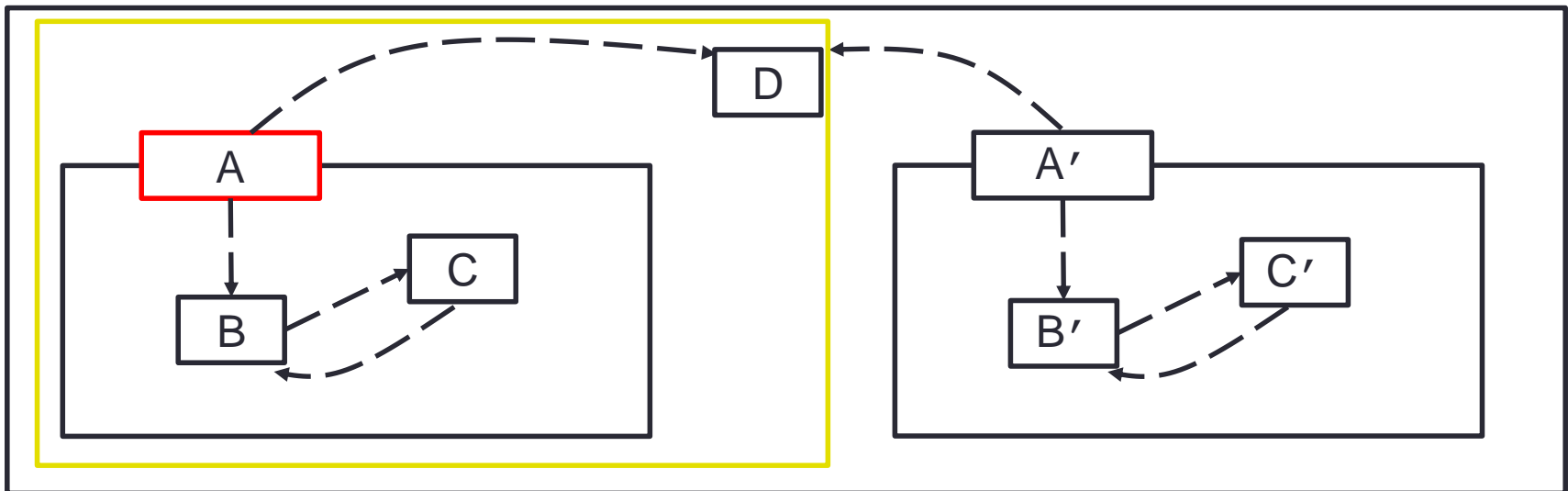
For all \mathcal{H} , \mathcal{H}' , \mathcal{H}'' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ and $\iota' \neq \iota$ then $\exists \mathcal{H}''$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \in \text{dom}(\mathcal{H}'')$ and $\iota \in \text{dom}(\mathcal{H})$.



Correctness of the formalism

Correctness property: Sheep Cloning creates a sub-heap that contains the new object.

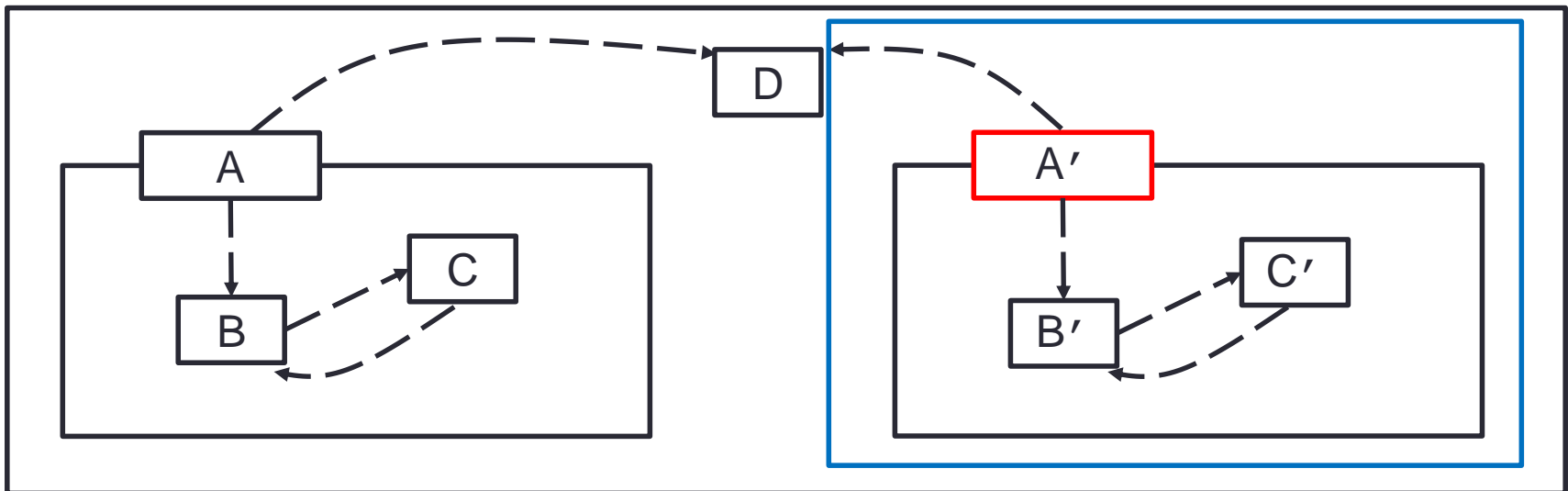
For all \mathcal{H} , \mathcal{H}' , \mathcal{H}'' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ and $\iota' \neq \iota$ then $\exists \mathcal{H}''$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \in \text{dom}(\mathcal{H}'')$ and $\iota \in \text{dom}(\mathcal{H})$.



Correctness of the formalism

Correctness property: Sheep Cloning creates a sub-heap that contains the new object.

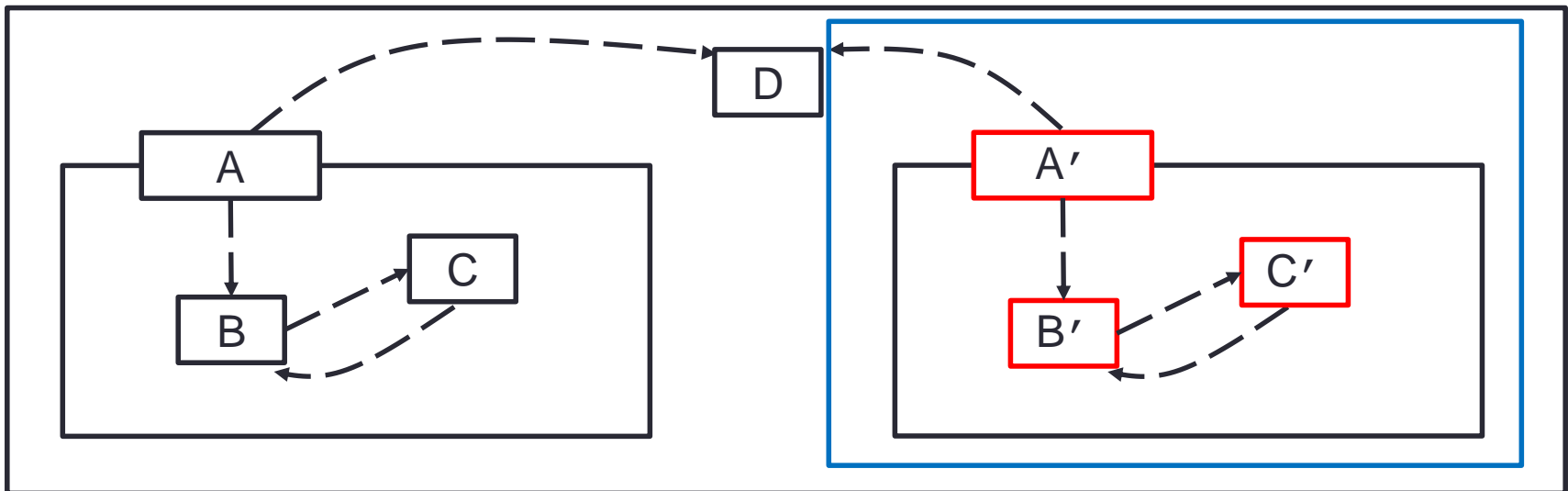
For all \mathcal{H} , \mathcal{H}' , \mathcal{H}'' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ and $\iota' \neq \iota$ then $\exists \mathcal{H}''$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \in \text{dom}(\mathcal{H}'')$ and $\iota \in \text{dom}(\mathcal{H})$.



Correctness of the formalism

Correctness property: All new objects are in the representation of the clone, and all objects in that representation are new.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ then $\iota'' \in \text{dom}(\mathcal{H}'')$ if and only if $\mathcal{H}' \vdash \iota'' \preceq \iota'$.

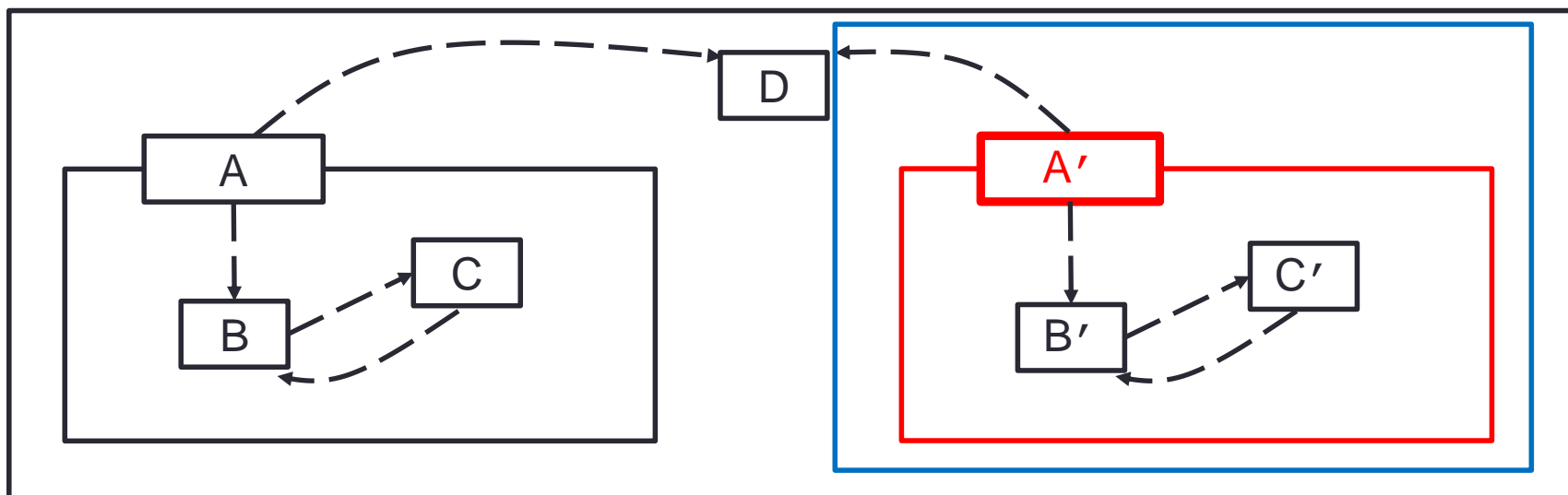


Correctness of the formalism

Correctness property: All new objects are in the representation of the clone, and all objects in that representation are new.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ then $\iota'' \in \text{dom}(\mathcal{H}'')$ if and only if $\mathcal{H}' \vdash \iota'' \preceq \iota'$.

$$A' \preceq A'$$

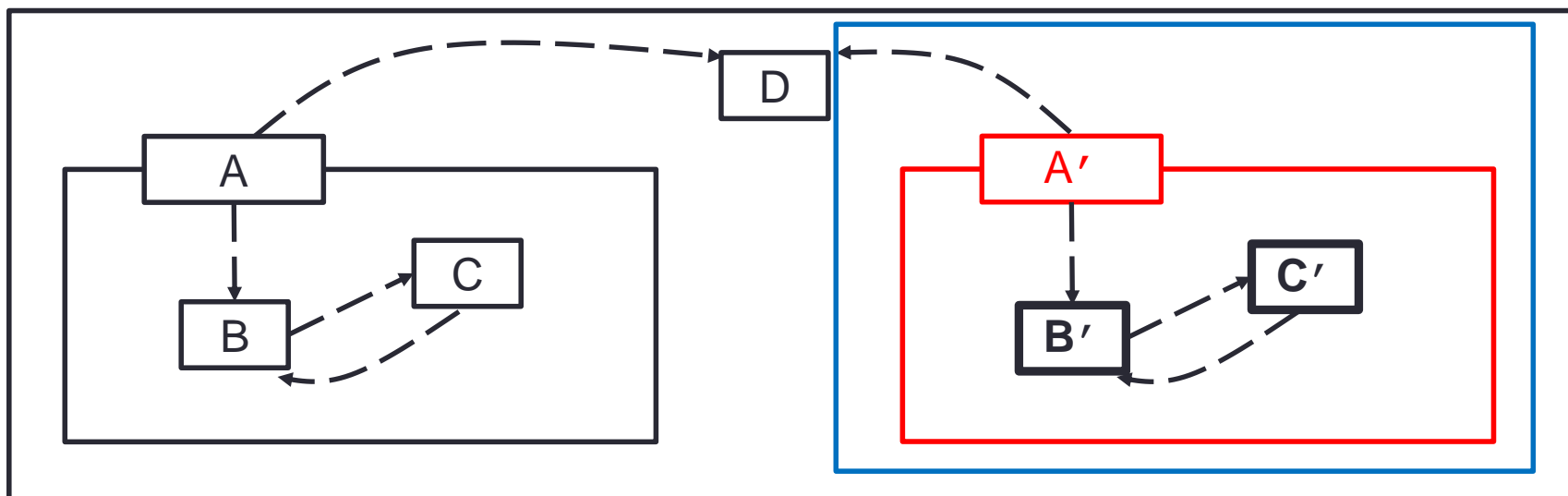


Correctness of the formalism

Correctness property: All new objects are in the representation of the clone, and all objects in that representation are new.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ then $\iota'' \in \text{dom}(\mathcal{H}'')$ if and only if $\mathcal{H}' \vdash \iota'' \preceq \iota'$.

$$\mathbf{B'} \preceq \mathbf{A'}, C' \preceq \mathbf{A'}$$

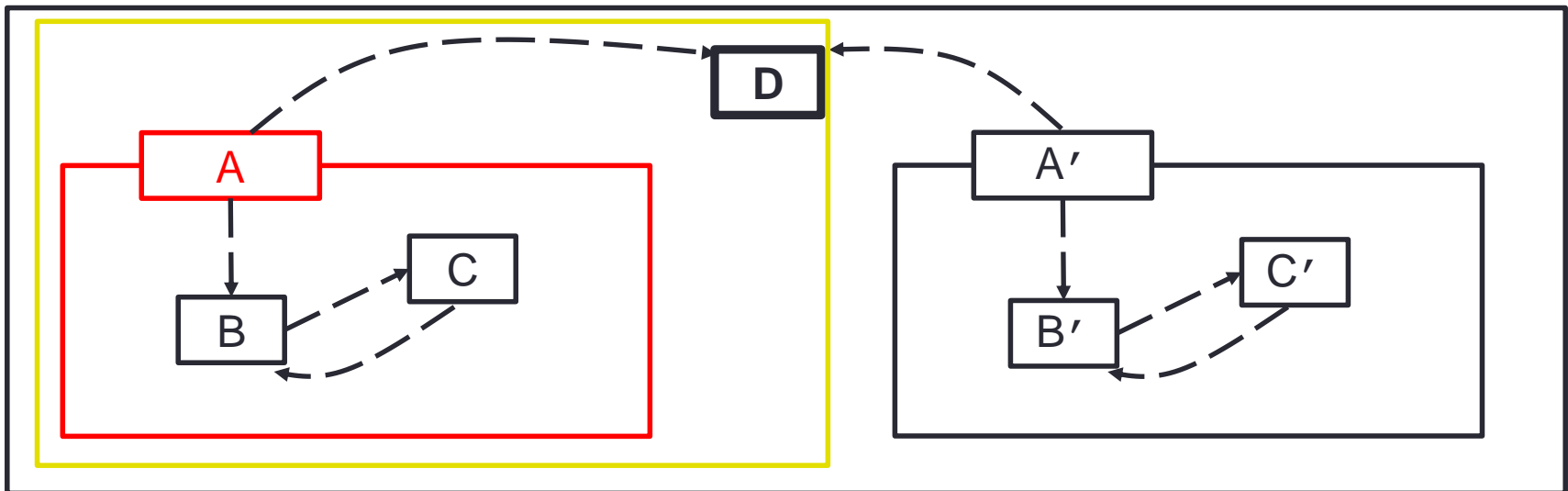


Correctness of the formalism

Correctness property: All objects outside the cloned object are outside the clone.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\iota' \neq \iota$ and $\forall \iota'' \in \text{dom}(\mathcal{H})$ and $\mathcal{H}' \vdash \iota \preceq \iota''$ then $\mathcal{H}' \vdash \iota' \preceq \iota''$.

$$A \preceq D$$

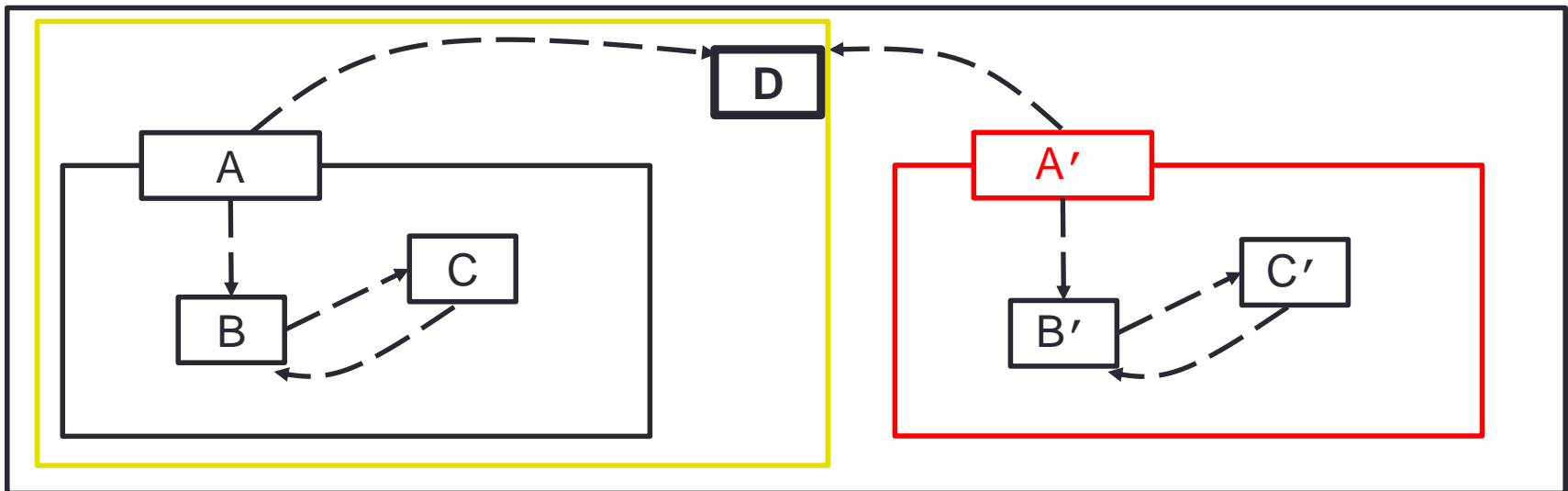


Correctness of the formalism

Correctness property: All objects outside the cloned object are outside the clone.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\iota' \neq \iota$ and $\forall \iota'' \in \text{dom}(\mathcal{H})$ and $\mathcal{H}' \vdash \iota \preceq \iota''$ then $\mathcal{H}' \vdash \iota' \preceq \iota''$.

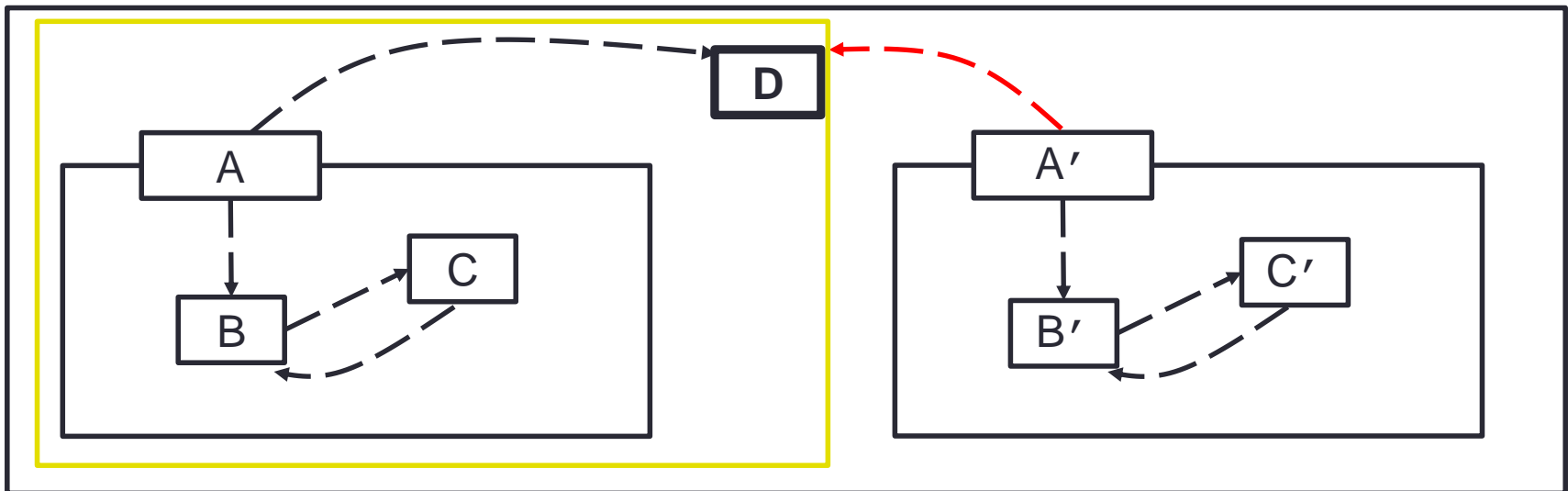
$$A' \preceq D$$



Correctness of the formalism

Correctness property: Sheep Cloning does not introduce references to the cloned object's representation.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , **if** $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ and $\forall \mathbf{f} \mapsto \iota'' \in \text{range}_{\downarrow 2}(\mathcal{H}'')$ where $\iota'' \in \text{dom}(\mathcal{H})$ **then** $\mathcal{H}' \vdash \iota \preceq \iota''$.

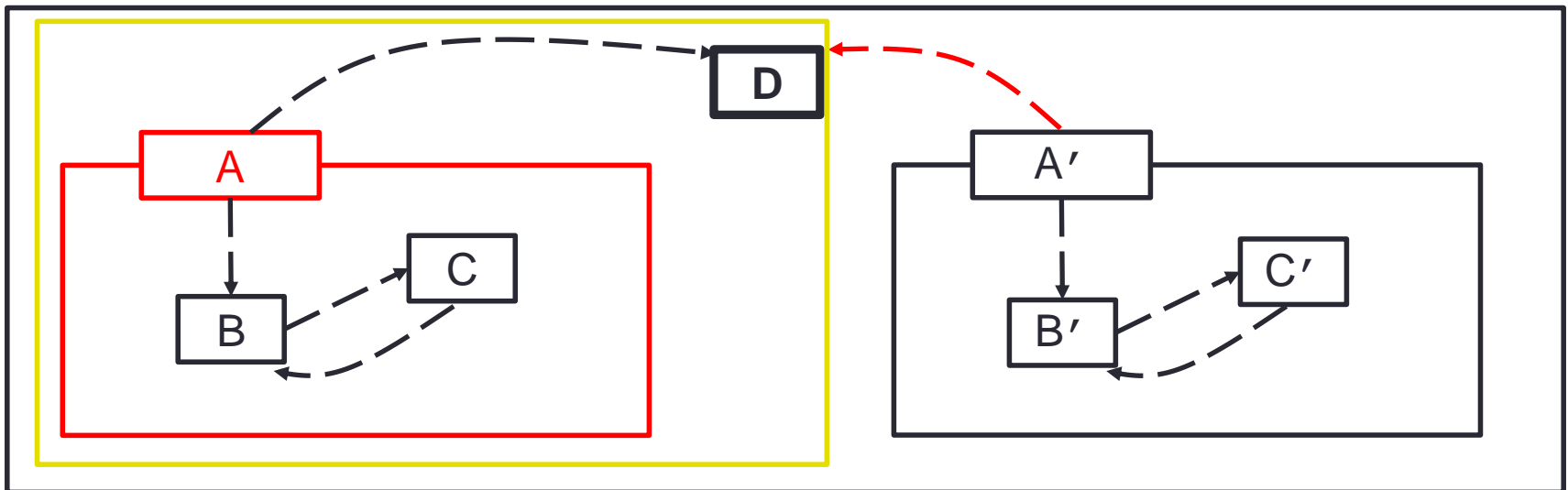


Correctness of the formalism

Correctness property: Sheep Cloning does not introduce references to the cloned object's representation.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ and $\forall \mathbf{f} \mapsto \iota'' \in \text{range}_{\downarrow 2}(\mathcal{H}'')$ where $\iota'' \in \text{dom}(\mathcal{H})$ then $\mathcal{H}' \vdash \iota \preceq \iota''$.

$$A \preceq D$$

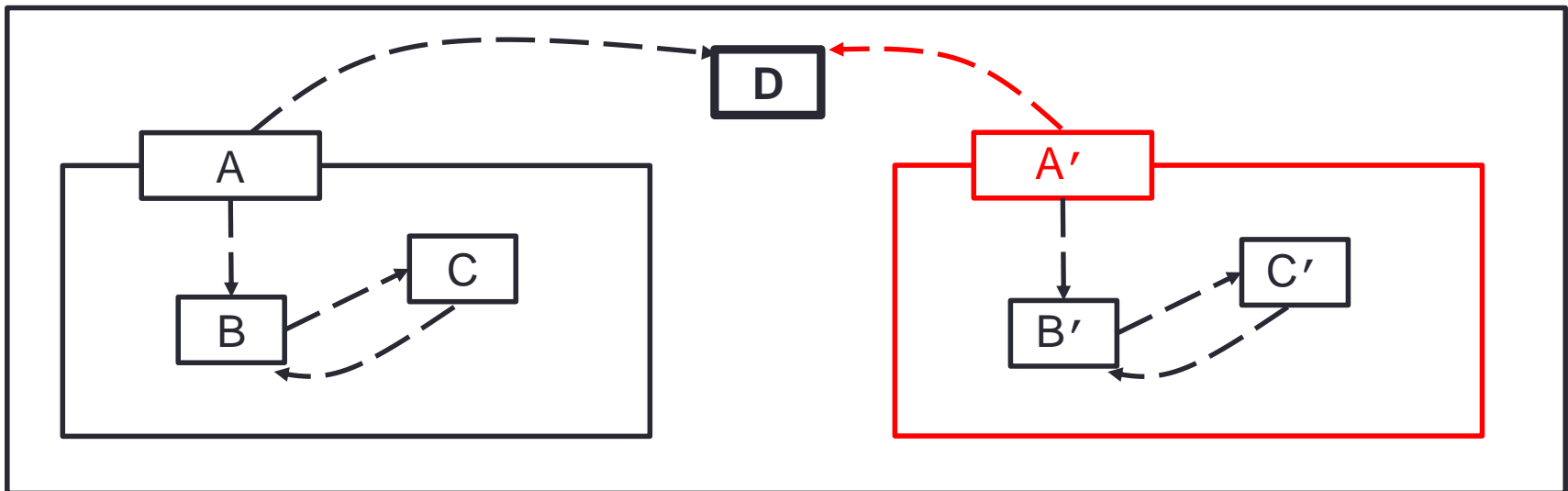


Correctness of the formalism

Correctness property: For all references from an object inside the clone to an object outside the clone, there is a reference to the same object from inside the cloned object.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H} \text{ OK}$ and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ and $\forall \mathbf{f} \mapsto \iota'' \in \text{range}_{\downarrow 2}(\mathcal{H}'')$ and $\mathcal{H}' \vdash \iota' \preceq \iota''$ then $\exists \mathbf{f}' \mapsto \iota'' \in \mathcal{H}(\iota^*) \downarrow_2$ where $\mathcal{H}' \vdash \iota^* \preceq \iota$.

$$A' \preceq D$$

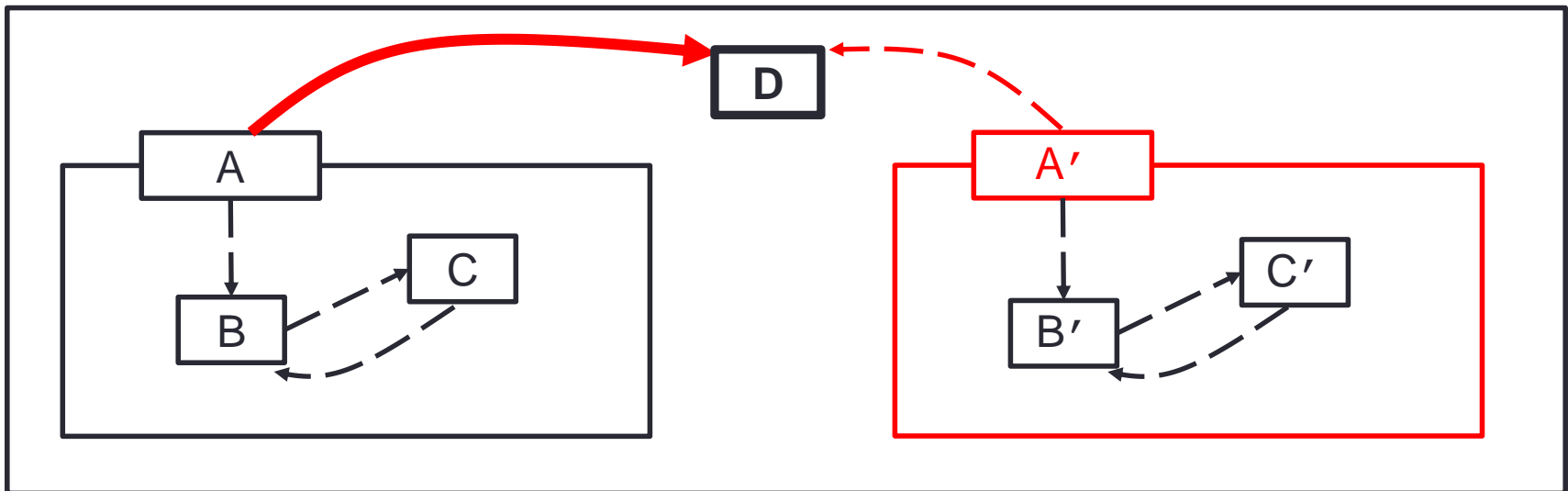


Correctness of the formalism

Correctness property: For all references from an object inside the clone to an object outside the clone, there is a reference to the same object from inside the cloned object.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H} \text{ OK}$ and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ where $\mathcal{H}' = \mathcal{H}$, \mathcal{H}'' and $\iota' \neq \iota$ and $\forall \mathbf{f} \mapsto \iota'' \in \text{range}_{\downarrow 2}(\mathcal{H}'')$ and $\mathcal{H}' \vdash \iota' \preceq \iota''$ then $\exists \mathbf{f}' \mapsto \iota'' \in \mathcal{H}(\iota^*)_{\downarrow 2}$ where $\mathcal{H}' \vdash \iota^* \preceq \iota$.

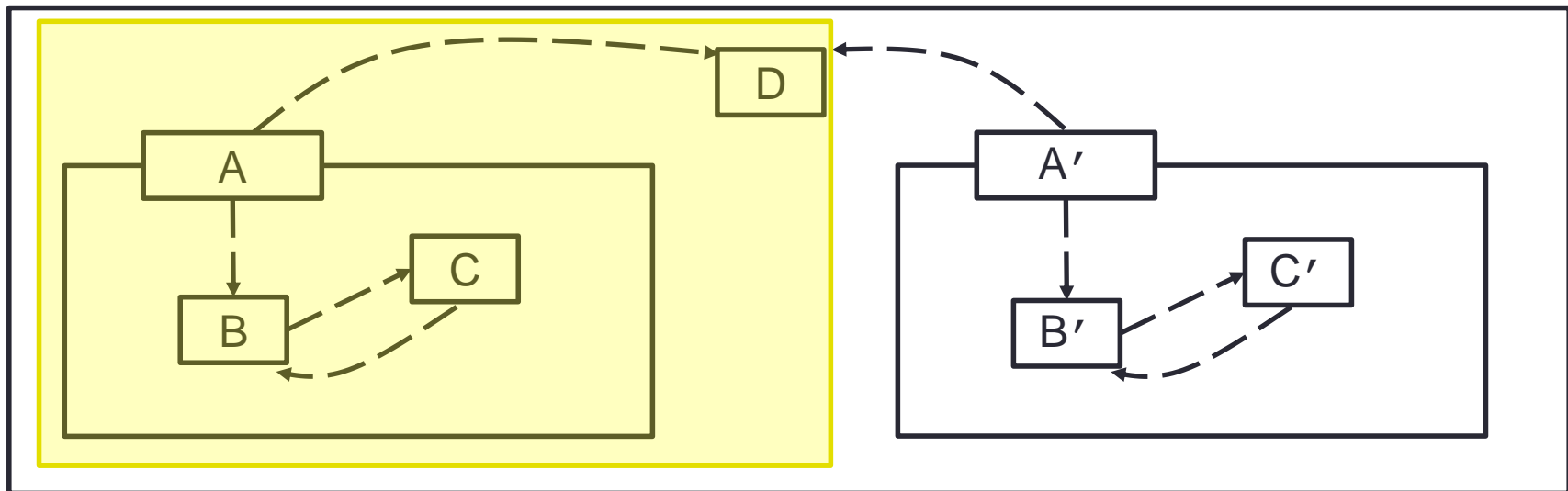
$$A' \preceq D$$



Correctness of the formalism

Correctness property: Sheep Cloning preserves owners-as-dominators.

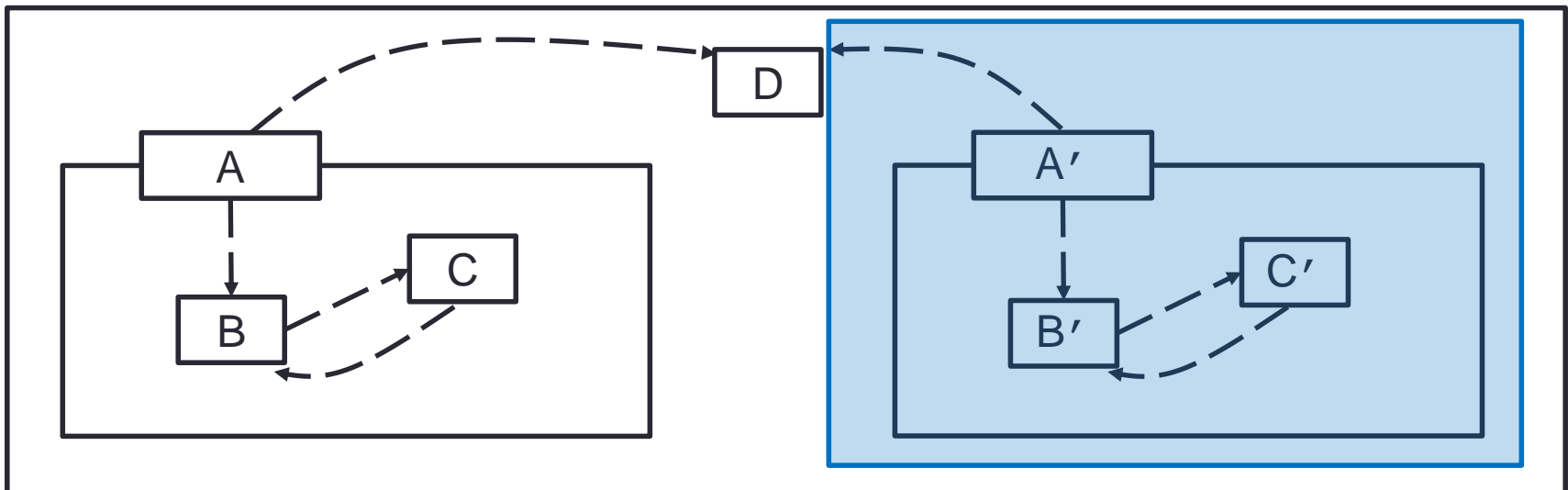
For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ and \mathcal{H} preserves owners-as-dominators then \mathcal{H}' preserves owners-as-dominators.



Correctness of the formalism

Correctness property: Sheep Cloning preserves owners-as-dominators.

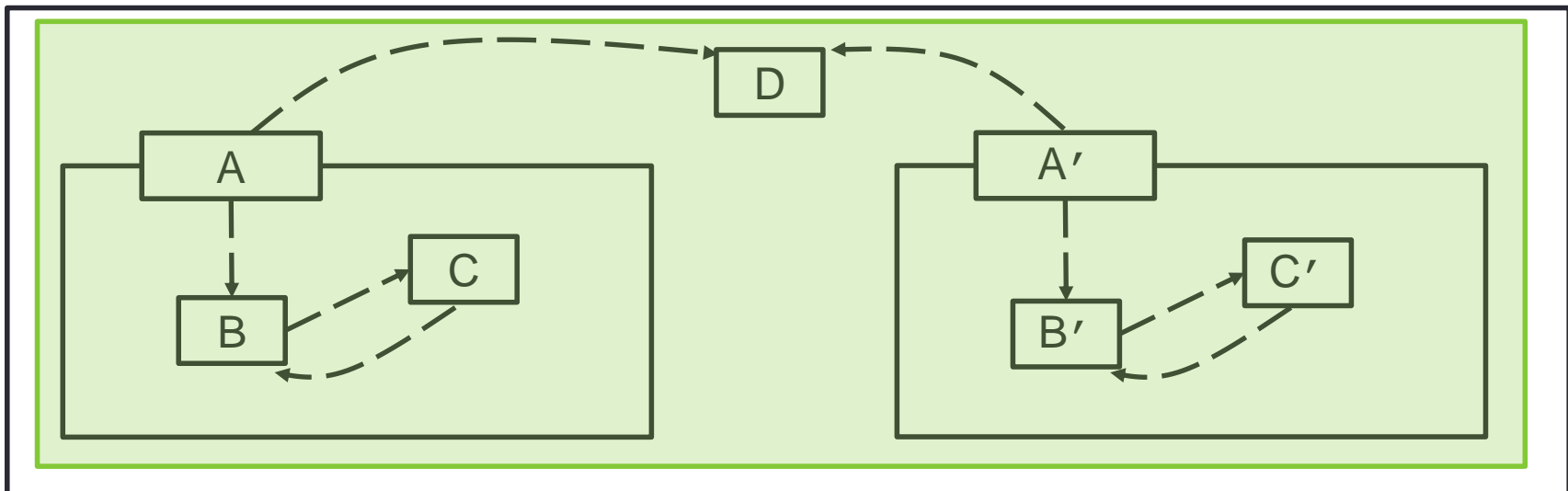
For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ and \mathcal{H} preserves owners-as-dominators then \mathcal{H}' preserves owners-as-dominators.



Correctness of the formalism

Correctness property: Sheep Cloning preserves owners-as-dominators.

For all \mathcal{H} , \mathcal{H}' , ι , and ι' , if $\vdash \mathcal{H}$ OK and $\text{sheep}(\iota); \mathcal{H} \rightsquigarrow \iota'; \mathcal{H}'$ and \mathcal{H} preserves owners-as-dominators then \mathcal{H}' preserves owners-as-dominators.



Summary

- Shallow is too shallow.
- Deep is too deep.
- Sheep = shallow + deep.
- Formalised sheep cloning.
- Proved soundness and correctness.

Thank you.

Questions?