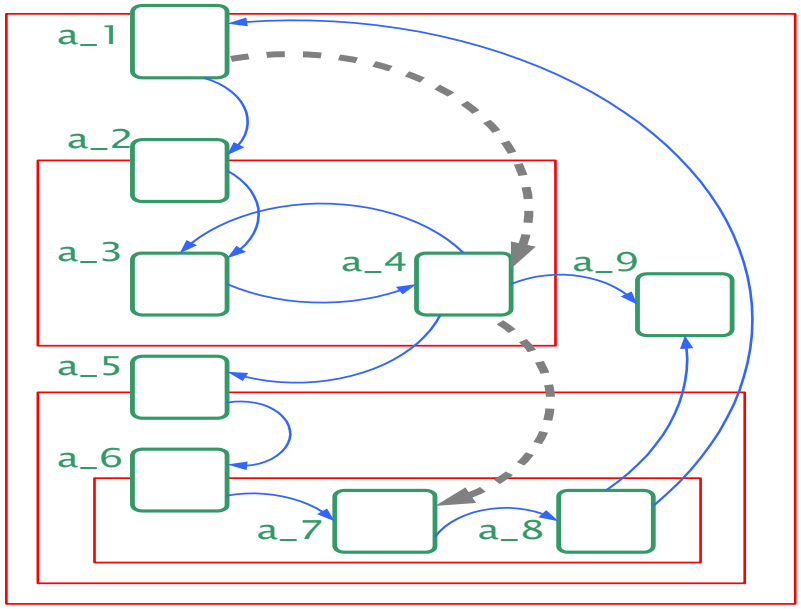
Ownership Application 14 – Memory Management

When deallocating object o, we would like to deallocate all the objects in o's box. This is safe, if we know that any path that leads to an object inside o's box, will go through o.

In other words, we need to know that o *dominates* all the objects in its box.

In other words, we want to have that
owners are dominators.

In the diagram, blue arrows are legal, and grey arrows are illegal.



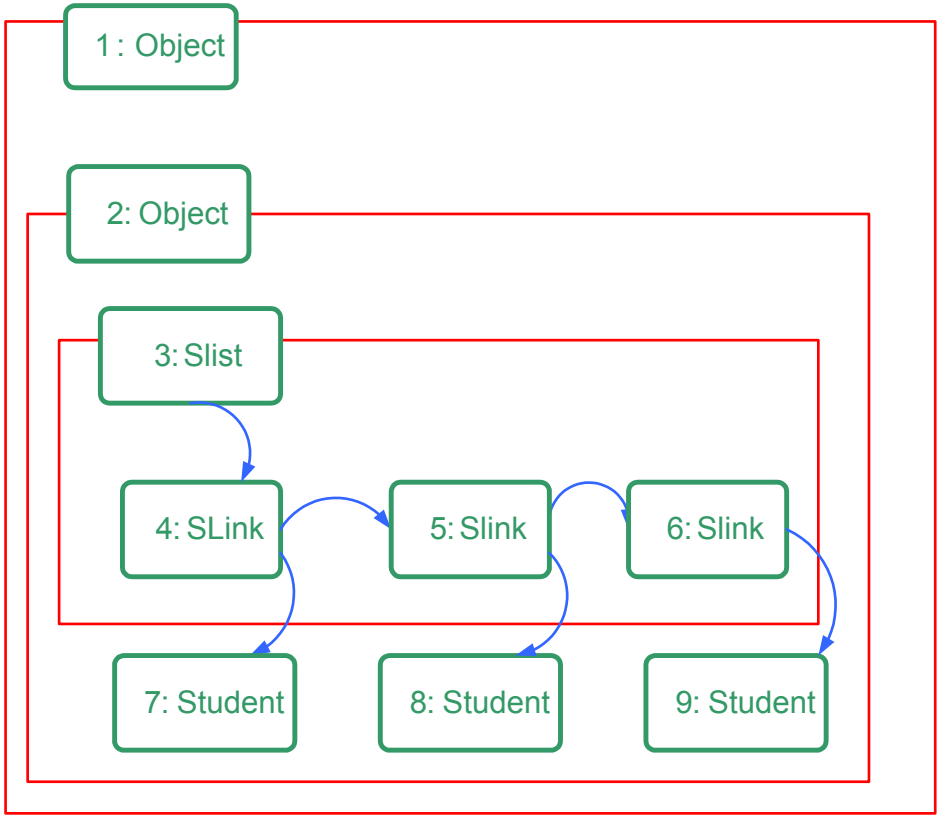
Owners as Dominators

Paths

Definition: At certain time of execution, characterized by heap χ, there is *a path* ($\langle\langle a_1, \ldots a_k \rangle\rangle$) from object a_1 to object a_k, iff a_i has some field pointing to object a_{i+1}, for all $i=1,\ldots k-1$.

$$\frac{\chi(a)(f) = a'}{\chi \vdash a \ \langle\langle a, a' \rangle\rangle \ a'}$$

$$\frac{\chi \vdash a_1 \ \langle\langle a_1, \ldots, a_k \rangle\rangle \ a_k \qquad \chi \vdash a_k \ \langle\langle a_k, \ldots, a_{k+m} \rangle\rangle \ a_{k+m}}{\chi \vdash a_1 \ \langle\langle a_1, \ldots, a_k, a_{k+1} \ldots a_{k+m} \rangle\rangle \ a_{k+m}}$$



Eg $X \vdash 3 \ll \ldots \gg 8$, but $X \nvdash 5 \ll \ldots \gg 7$.

Obviously, path-relationship changes with execution, i.e. possible that $e, X \rightsquigarrow e', X'$ and $X \vdash a \ll \ldots \gg a'$ but $X' \nvdash a \ll \ldots \gg a'$.

Object dominating another Object

Assume a fixed root object a_r.

Definition: At certain time of execution, characterized by heap X, object a *dominates* object a', iff all paths from a_r to a' lead through a, ie

$$X \vdash a \; dom \; a'$$

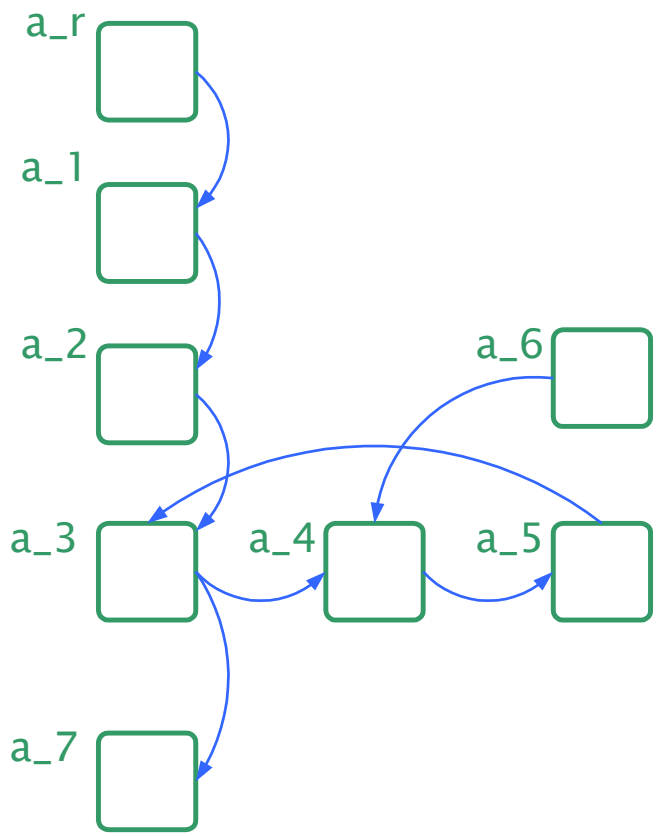
iff

$$X \vdash a\_r \; \langle\langle a\_r, \ldots, a\_k \rangle\rangle \; a' \quad \Rightarrow \quad a = a\_i \text{ for some } i \in \{1 \ldots k\}$$

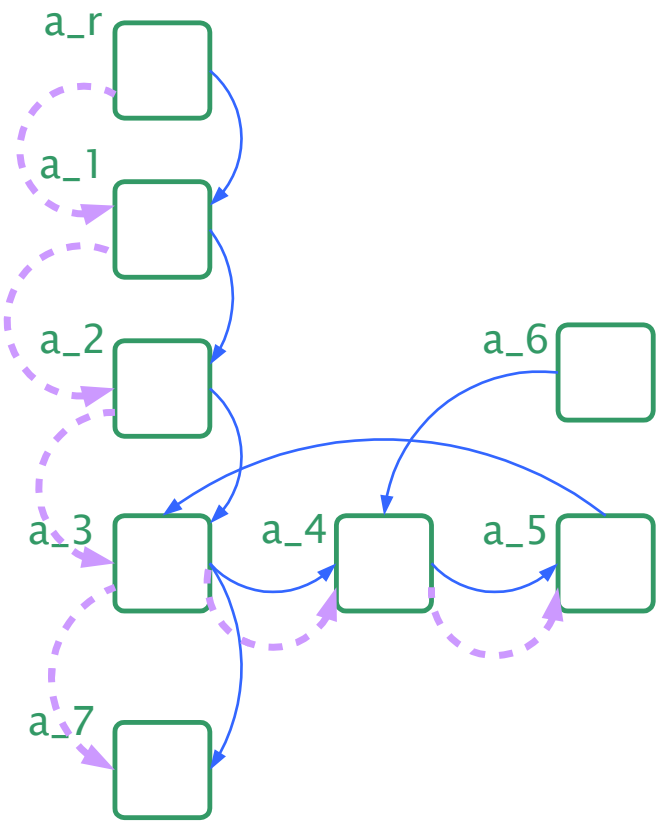
- Is the dom relationship transitive?
- Does $X \vdash a \; dom \; a'$ imply that $X \vdash a \; \langle\langle \ldots \rangle\rangle \; a'$?
- Does $X \vdash a \; dom \; a'$, $X \vdash a \; dom \; a''$ imply that $a' = a''$?
- Does $X \vdash a' \; dom \; a$, $X \vdash a'' \; dom \; a$ imply that $a' = a''$?

Example

for a state x_1:



the dom relationship is:



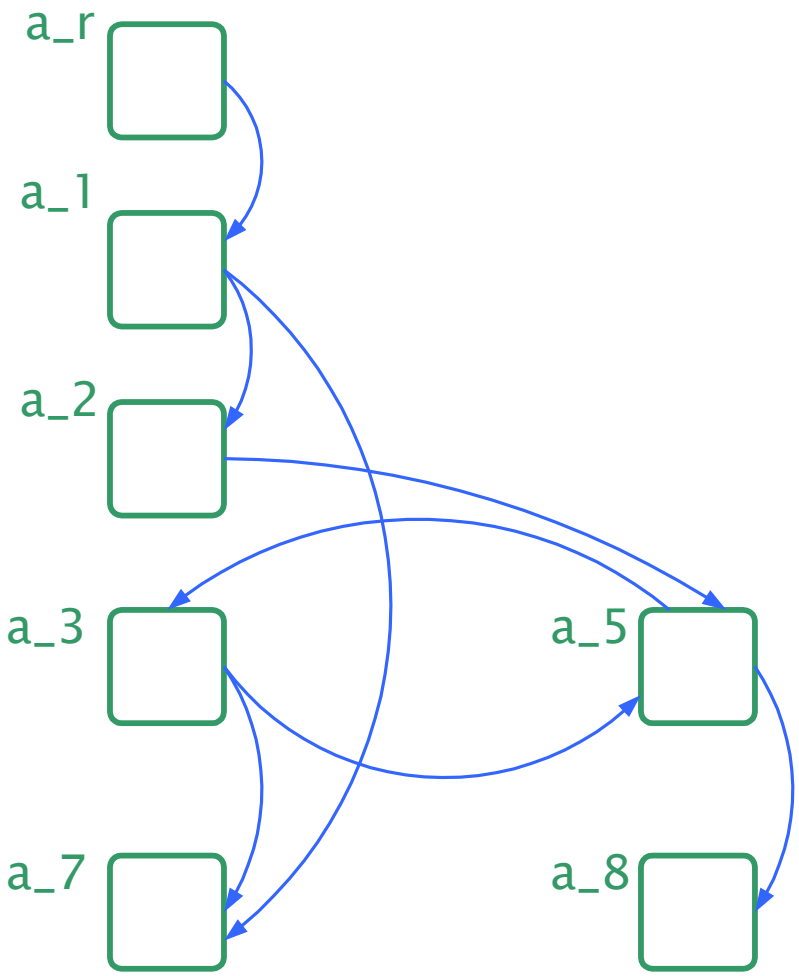
What about a_6?

Therefore, a dom a′ almost guarantees that a′ cannot be accessed from an object "outside" a, unless it "goes through" a. ….

☺ ☺ ☺

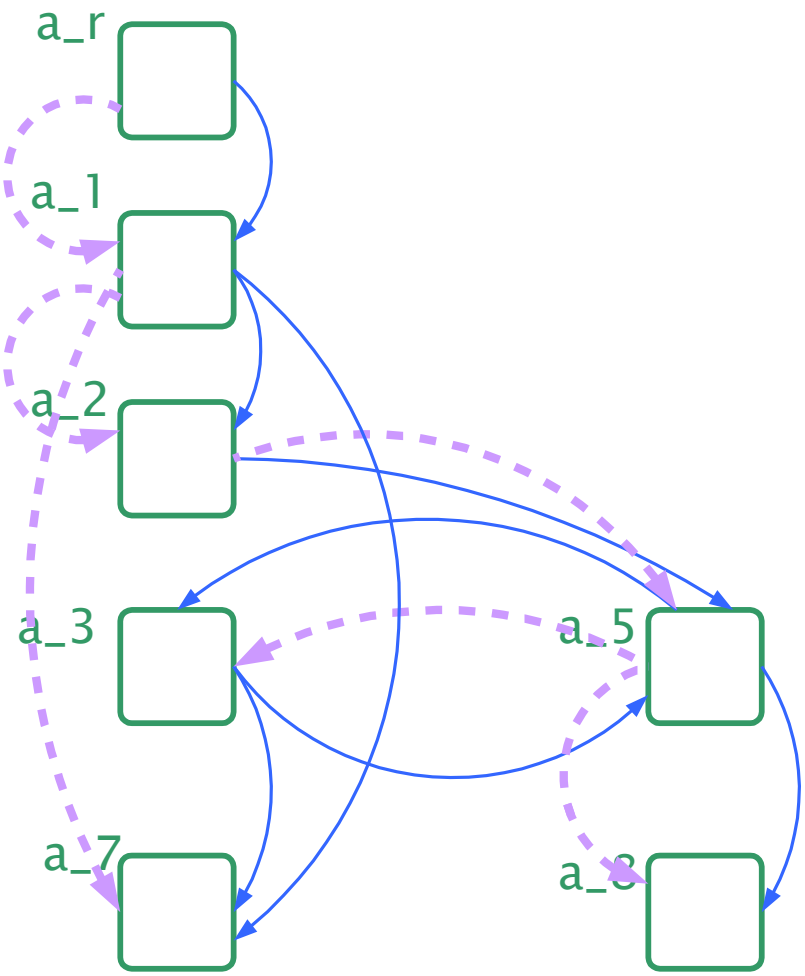
However, program execution may create new objects, may create new paths, and may destroy other paths …, eg …

later on, for a state x_2:



What happened to a_4, a_6?

the dom relationship is:



Therefore, the dom relationship is *not* invariant with program execution. In other words, it is possible to have $e, \chi \rightsquigarrow e', \chi'$ and $\chi \vdash a \ dom \ a'$ but $\chi' \nvdash a \ dom \ a'$.

☹ ☹ ☹

We will use a mapping from objects to objects which is invariant with program execution, and which, in a type correct program, will imply the dom relationship:

☺

We already have such an invariant mapping from object to object…

☺ ☺ ☺

Owners as dominators

The ownership mapping, owner, *respects* the dom relationship at a certain time of program execution iff the owners of all objects dominate them.

Definition

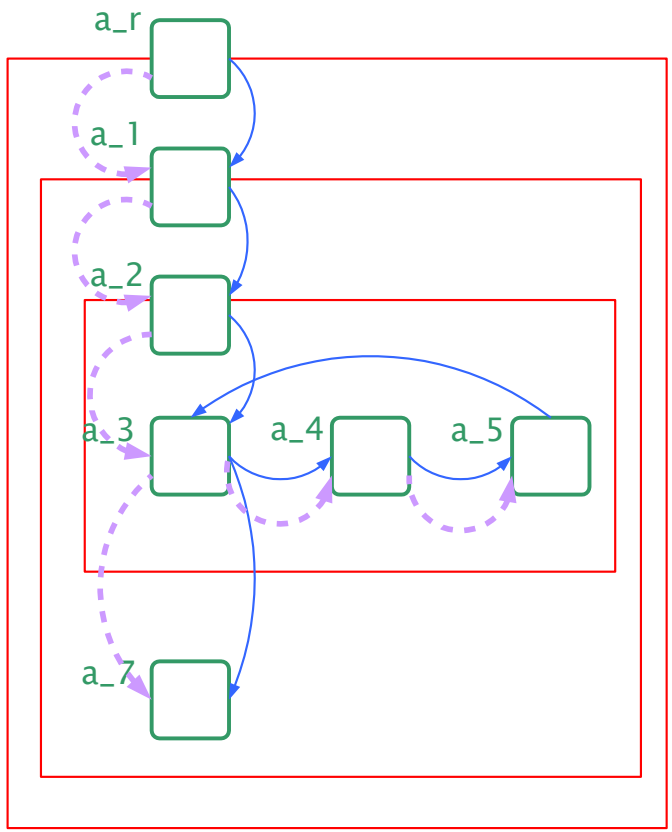
$$owner \vdash x \Diamond \quad \text{iff} \quad \text{for all } a: \quad x \vdash owner(a) \ dom \ a$$

- The owner mapping is independent of state x.
- The dom relationship is derived from the state x.
- We expect, that execution of a well typed program will preserve respect for dom, ie:

$$e \text{ well typed and } e, x \rightsquigarrow v, x' \text{ and } owner \vdash x \Diamond$$
$$\Rightarrow$$
$$owner \vdash x' \Diamond$$

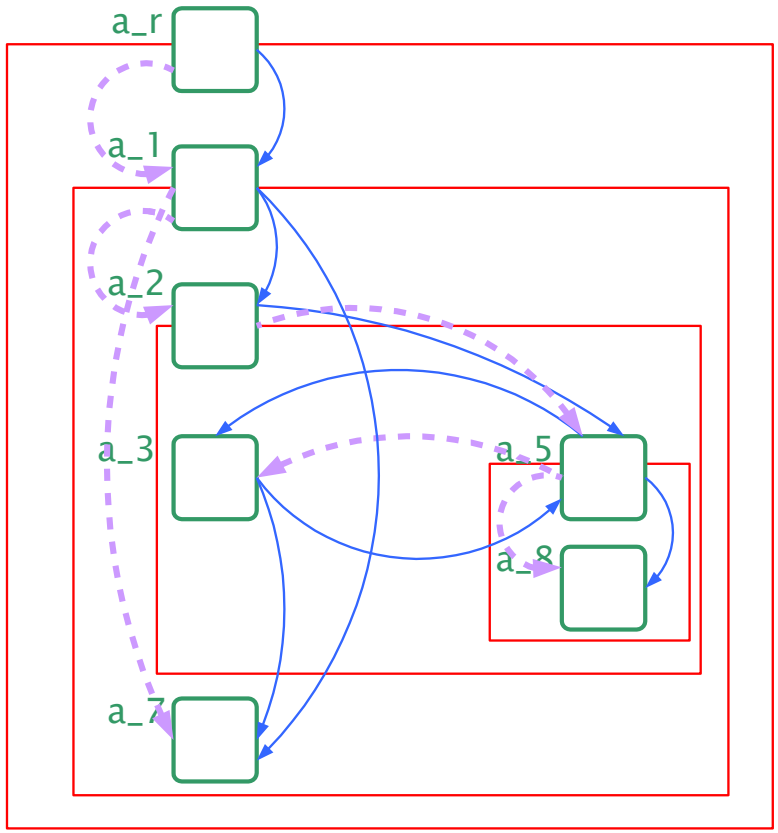
Consider a mapping o1(a_1)=a_r, o1(a_2)=a_1, o1(a_3)=a_2, o1(a_4)=a_2, o1(a_5)=a_2, o1(a_7)=a_1. Thus:

for x_1:



o1 ⊢ x_1 ◊

for x_2:



o1 ⊢ x_2 ◊

Notes

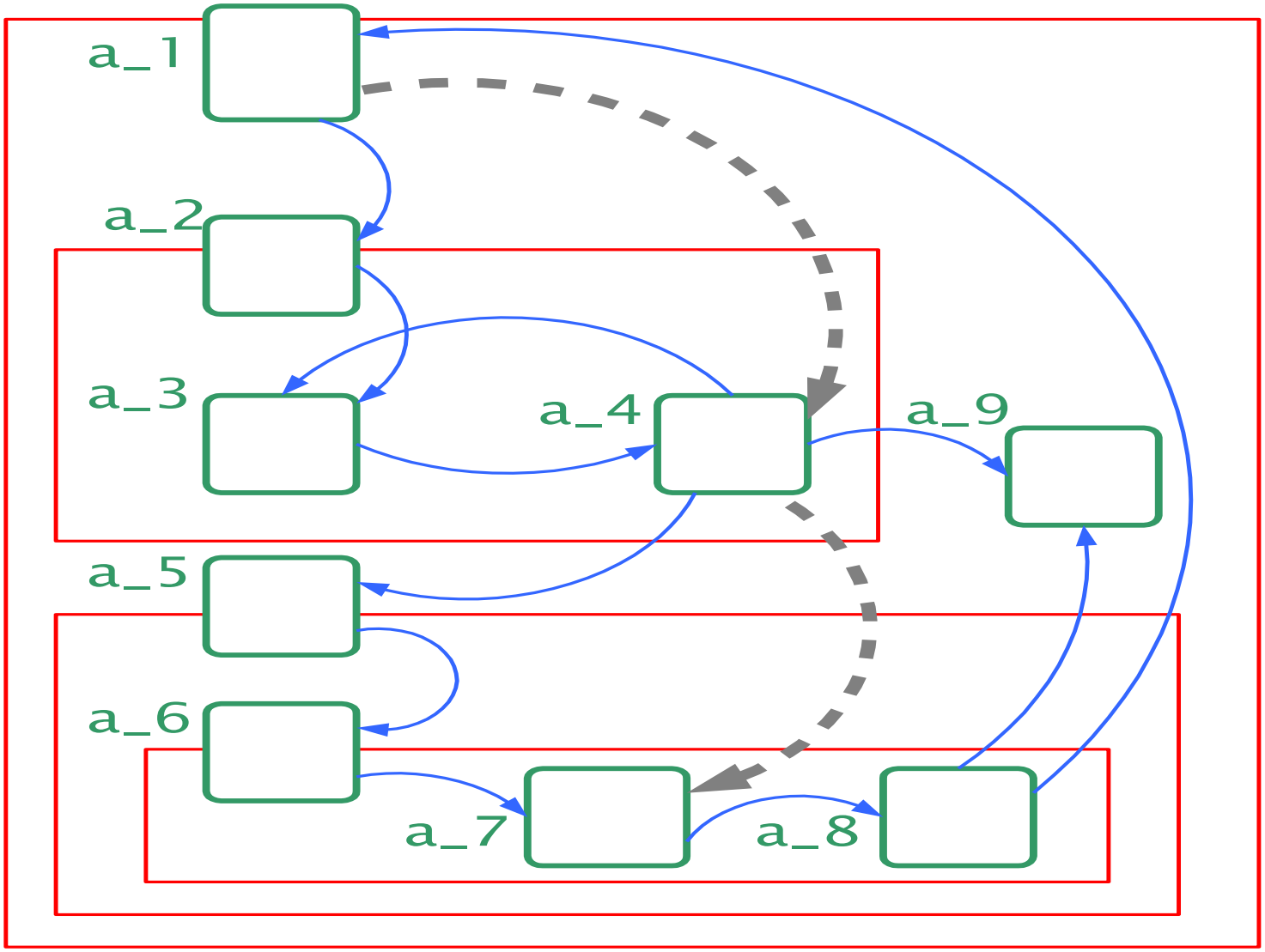
- The ownership boxes and the dom relationship are *not* part of the explicit state, however the paths are.
- During program execution, "new boxes" may be created
- Access "to the inside of the box" is only allowed from the direct owner of the box.
- We can formalize the above requirement through

  owner $\vdash \chi \Diamond$ iff

  $\forall a,a': \quad \chi \vdash a \ll a,a' \gg a' \quad \Rightarrow \quad \exists k \ owner^k(a) = owner(a')$

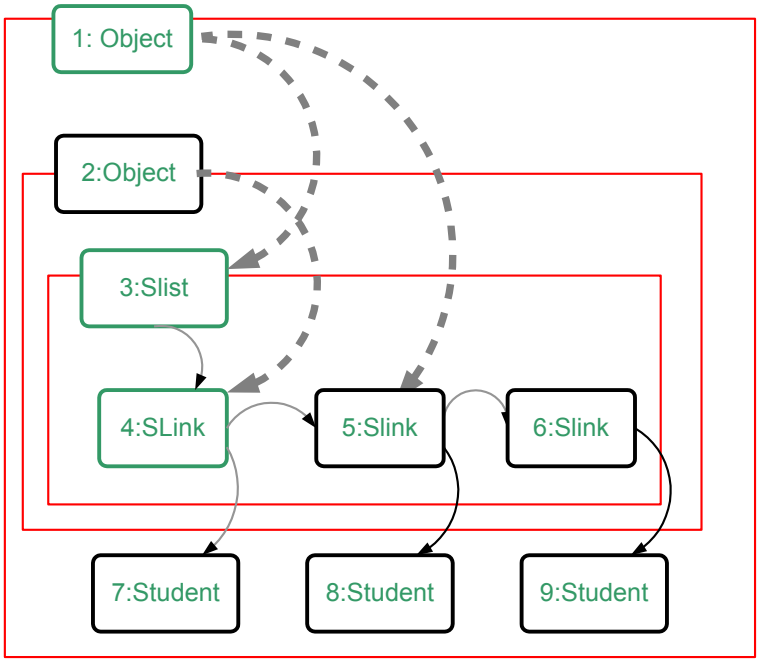
  ie, a reference from a to a' is only legal if a, or one of the owners of a is the owner of a' (Proof?).
- Above does not hold for paths, ie

  $\chi \vdash a \ll \ldots \gg a'$ does *not* imply $\exists k \ owner^k(a) = owner(a')$



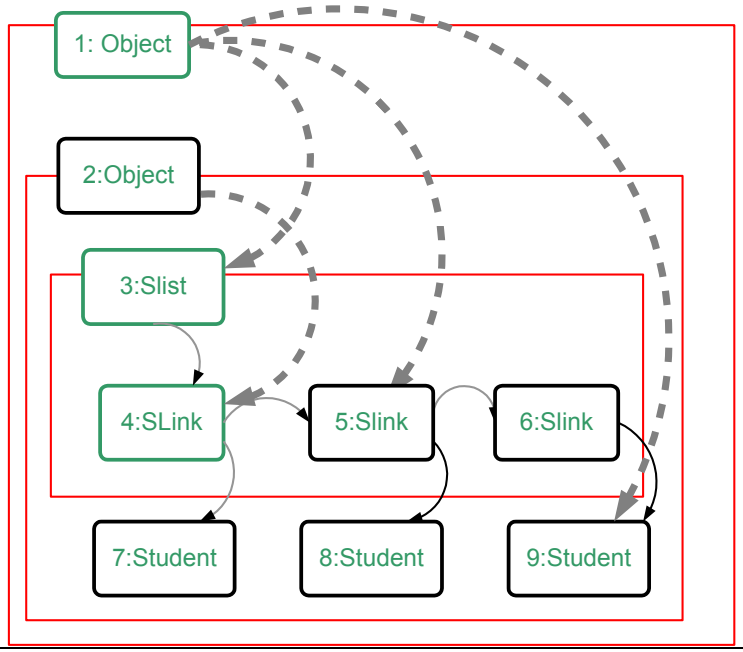
Blue arrows legal, grey dotted arrows are illegal.

For the type `SList<o2,o1>`



Thus,
- 3 "controls" 4, 5 & 6,
- 2 "controls" 3,
- 1 "controls" 2, 7, 8, 9.

and type `SList<o2,o2>`:



Thus,
- 3 "controls" 4, 5 & 6,
- 2 "controls" 3, 7, 8, 9.

Sufficient condition to statically ensure Owners as Dominators:

Consider the type

```
ClassId<r1,….rn>
```

Summary

- Ownership of objects characterizes the "location" of an object.
- Ownership is a relation across objects, not across classes.
- Privacy of members (eg C++) restricts scoping, but not aliasing, privacy is a relationship across classes.
- Many applications of ownership.
- Ownership types falvours have been incorporated into Scala, Rust, and X10 (for parallelism and concurrency), and into Real Time and Safety Critical Java (for memory management).
- In the next two talks we will discuss applications of ownership types to the actor paradigm, and to garbage collection.