

Software Verification (Fall 2013)

Lecture 5: Separation Logic

Part III

Chris Poskitt



In the previous lecture we saw that:

- separation logic is an extension of Hoare logic for shared mutable data structures
- program states are now modelled by **variable stores and heaps**
- **spatial connectives** allow assertions to focus on resources used by programs
- **frame rule** enables local reasoning

Next on the agenda

(1) model of program states for separation logic 

(2) assertions and spatial connectives 

(3) axioms and inference rules 

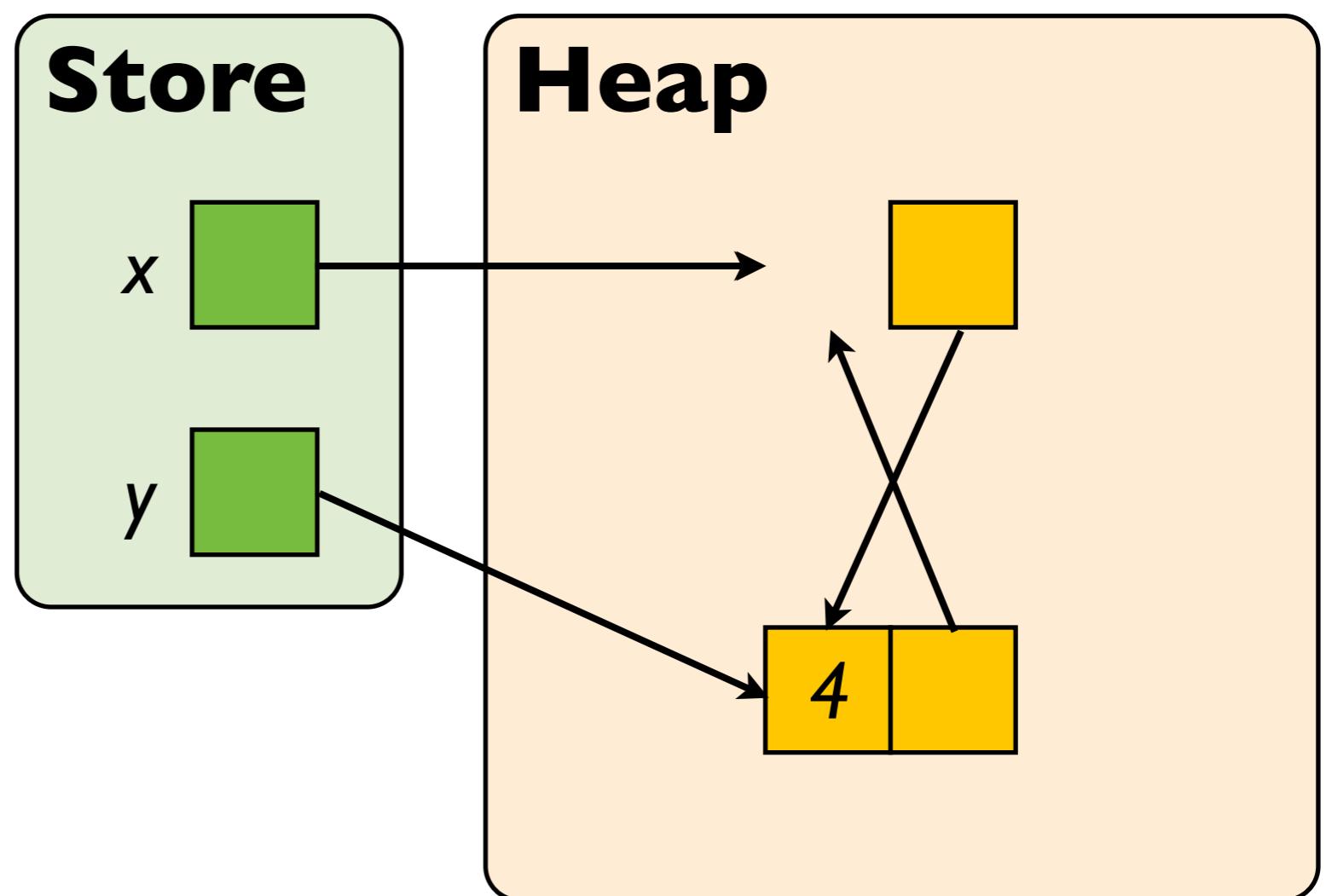
(4) program proofs

Exercise: prove this!

{emp}

```
x := cons(3,3);
y := cons(4,4);
[x+l] := y;
[y+l] := x;
y := x+l;
dispose x;
y := [y];
```

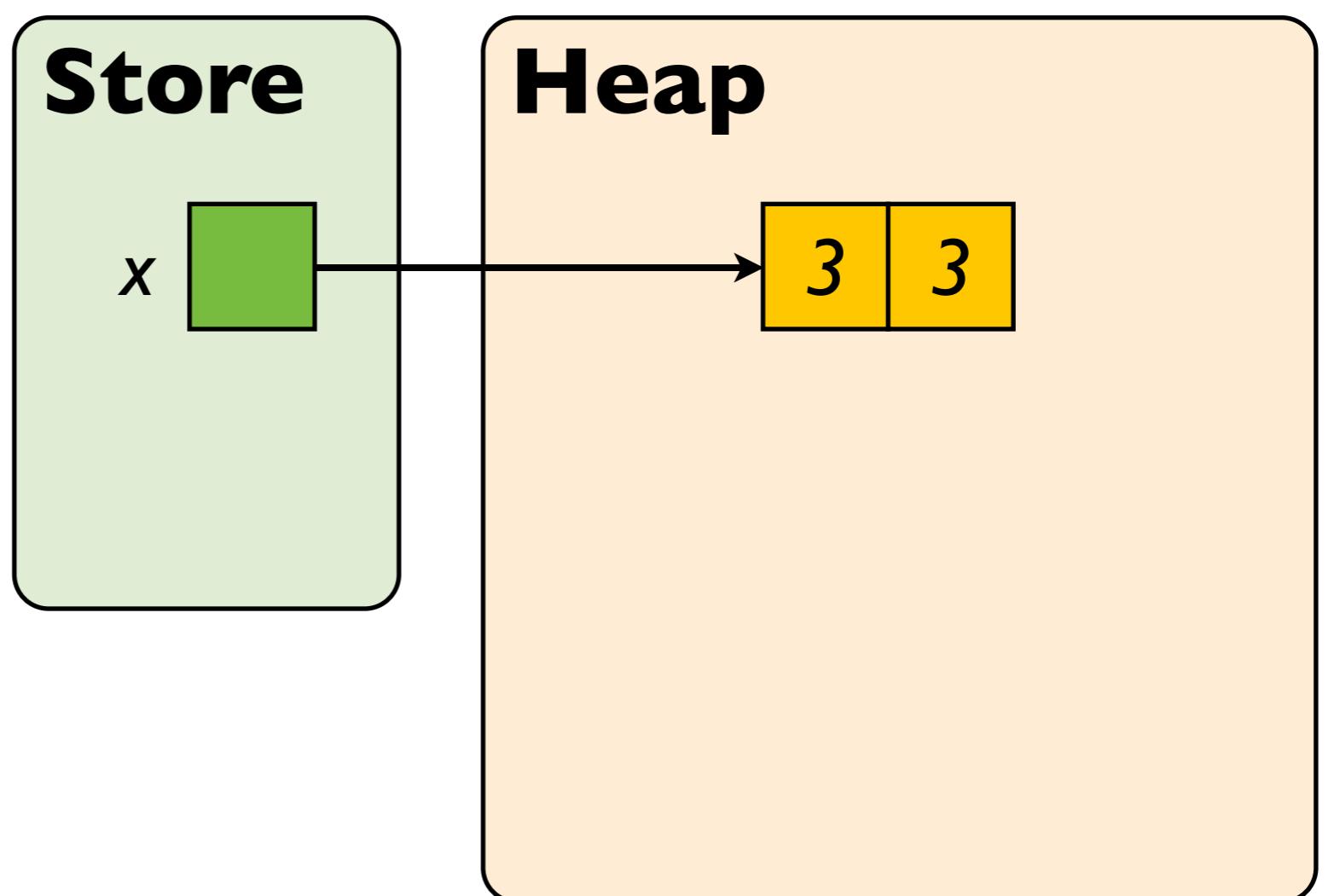
{y|->4 * true}



Proof outline

{emp}

x := cons(3,3);

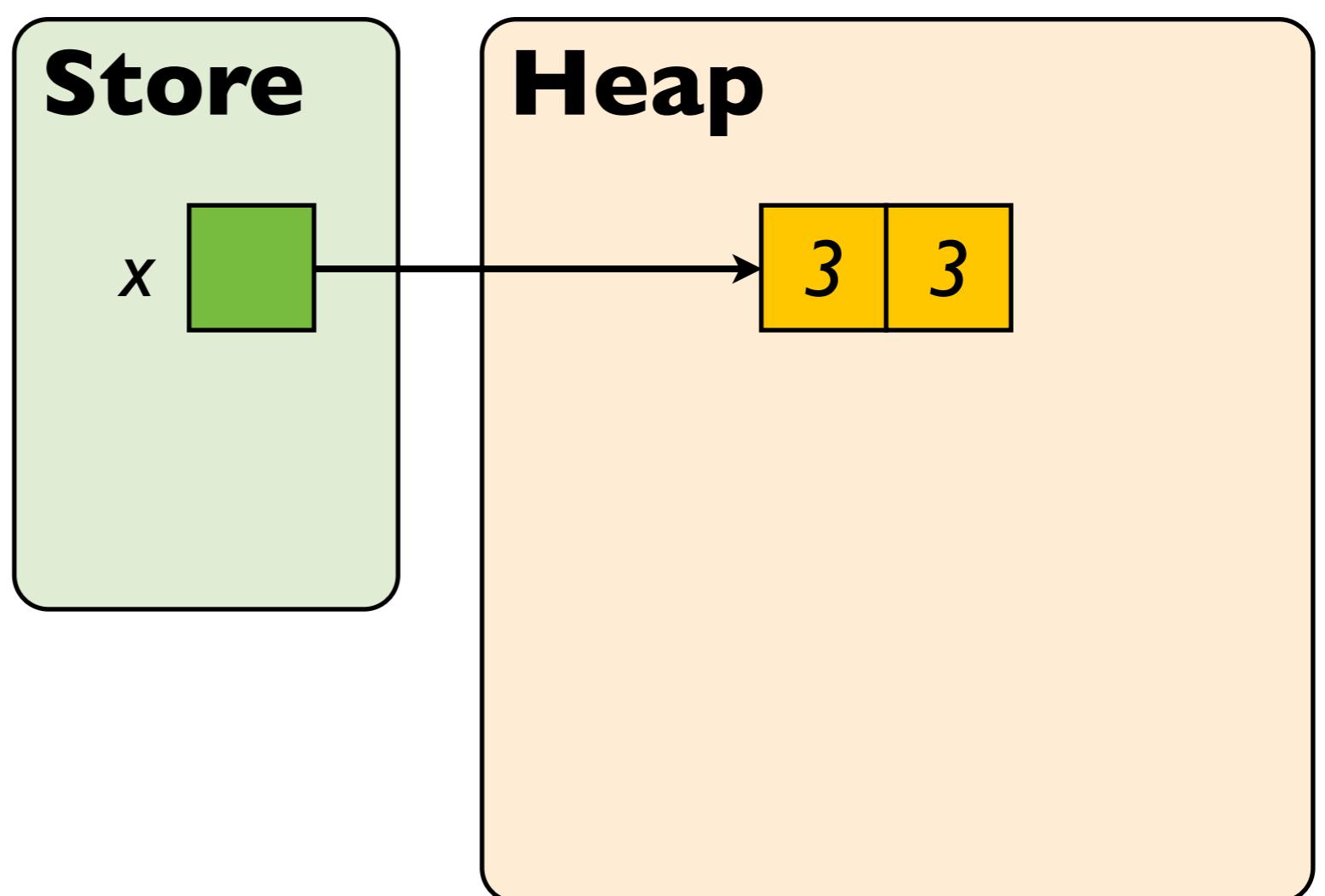


Proof outline

{emp}

x := cons(3,3);

{x |-> 3,3}



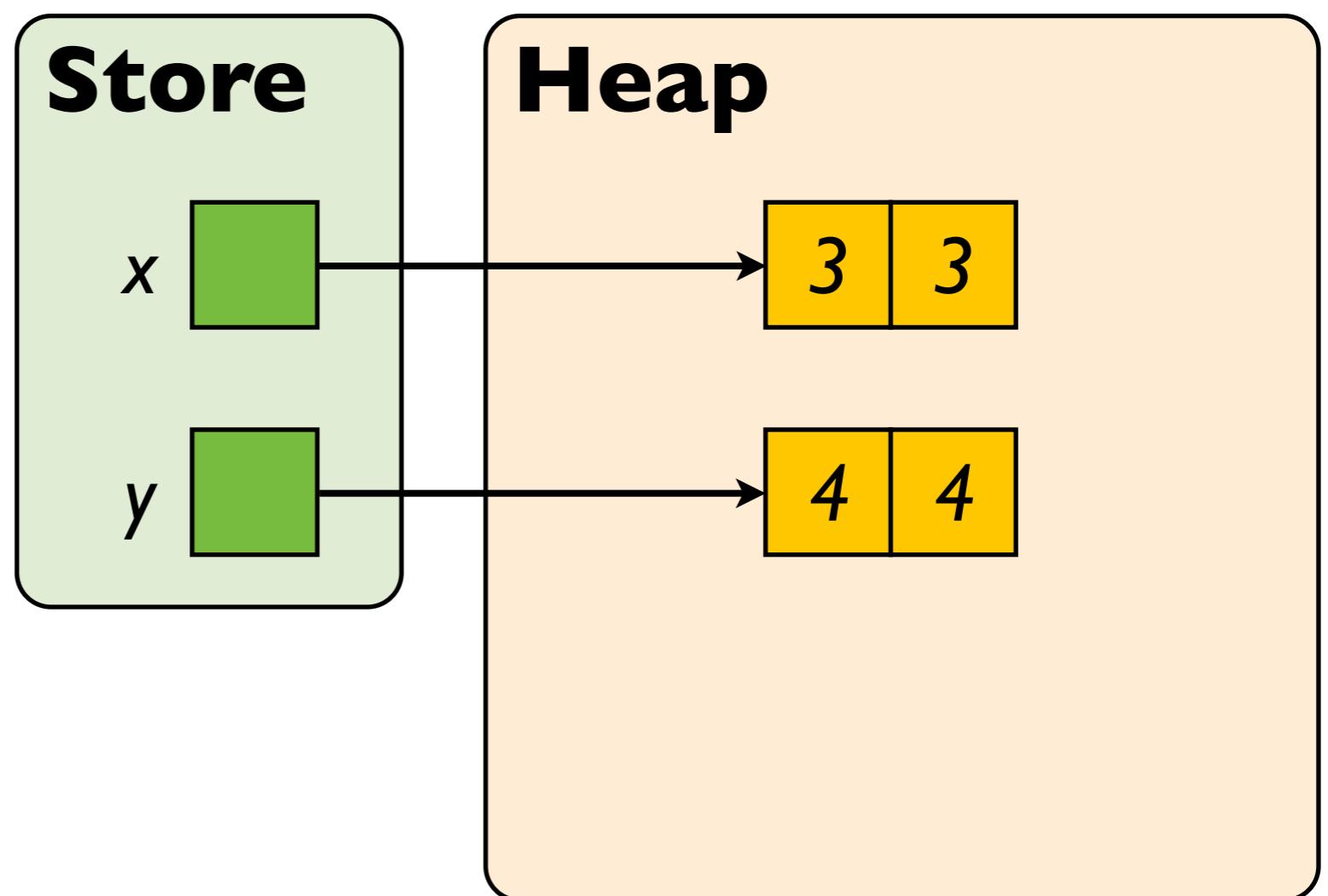
Proof outline

{emp}

x := cons(3,3);

{x |-> 3,3}

y := cons(4,4);



Proof outline

{emp}

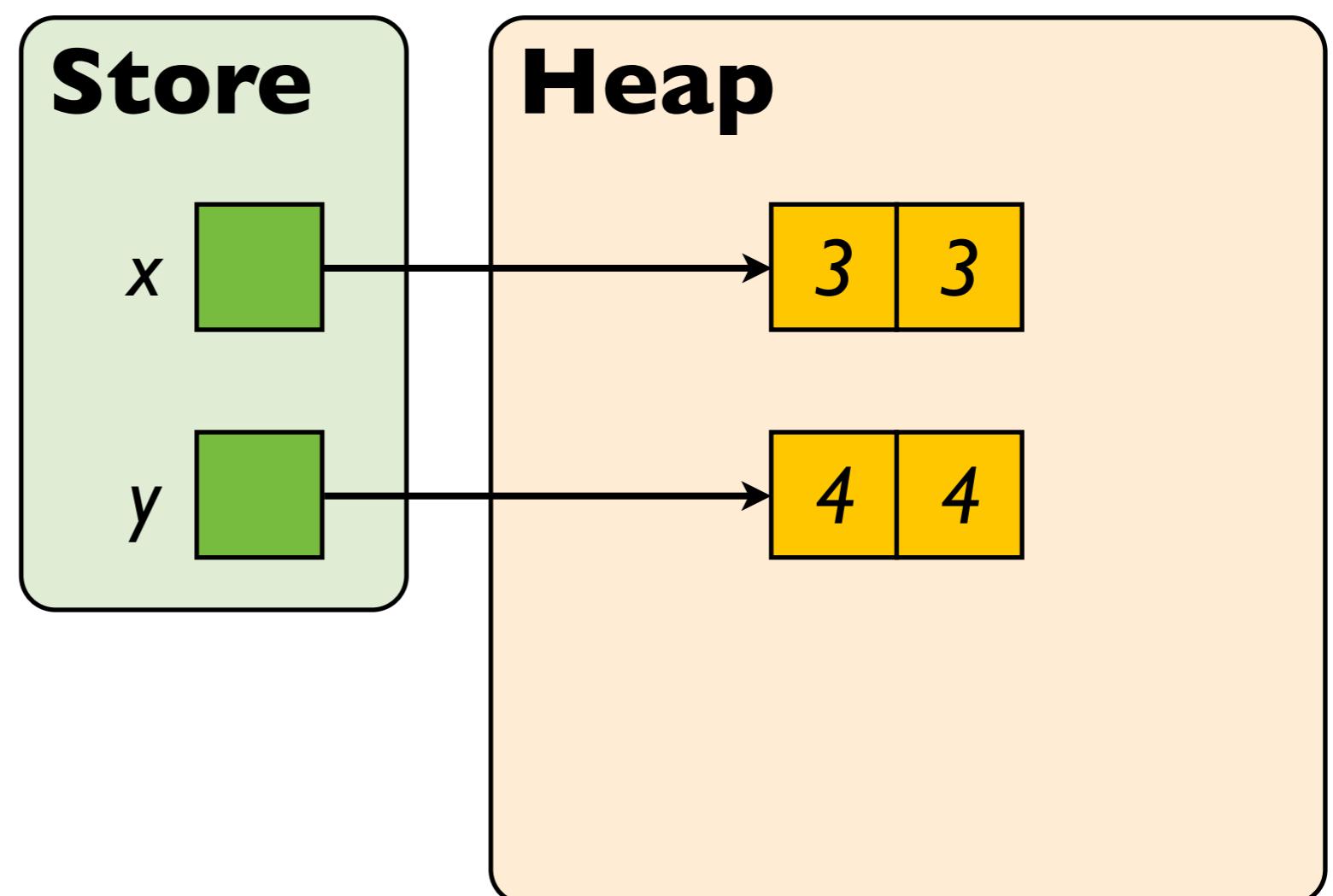
$x := \text{cons}(3,3);$

{ $x \rightarrow 3,3$ }

{ $x \rightarrow 3,3 * \text{emp}$ }

$y := \text{cons}(4,4);$

*rule of
consequence*



Proof outline

{emp}

x := cons(3,3);

{x |-> 3,3}

{x |-> 3,3 * emp}

{emp}

y := cons(4,4);

{y |-> 4,4}



frame rule!

{x |-> 3,3 * y |-> 4,4}

Proof outline

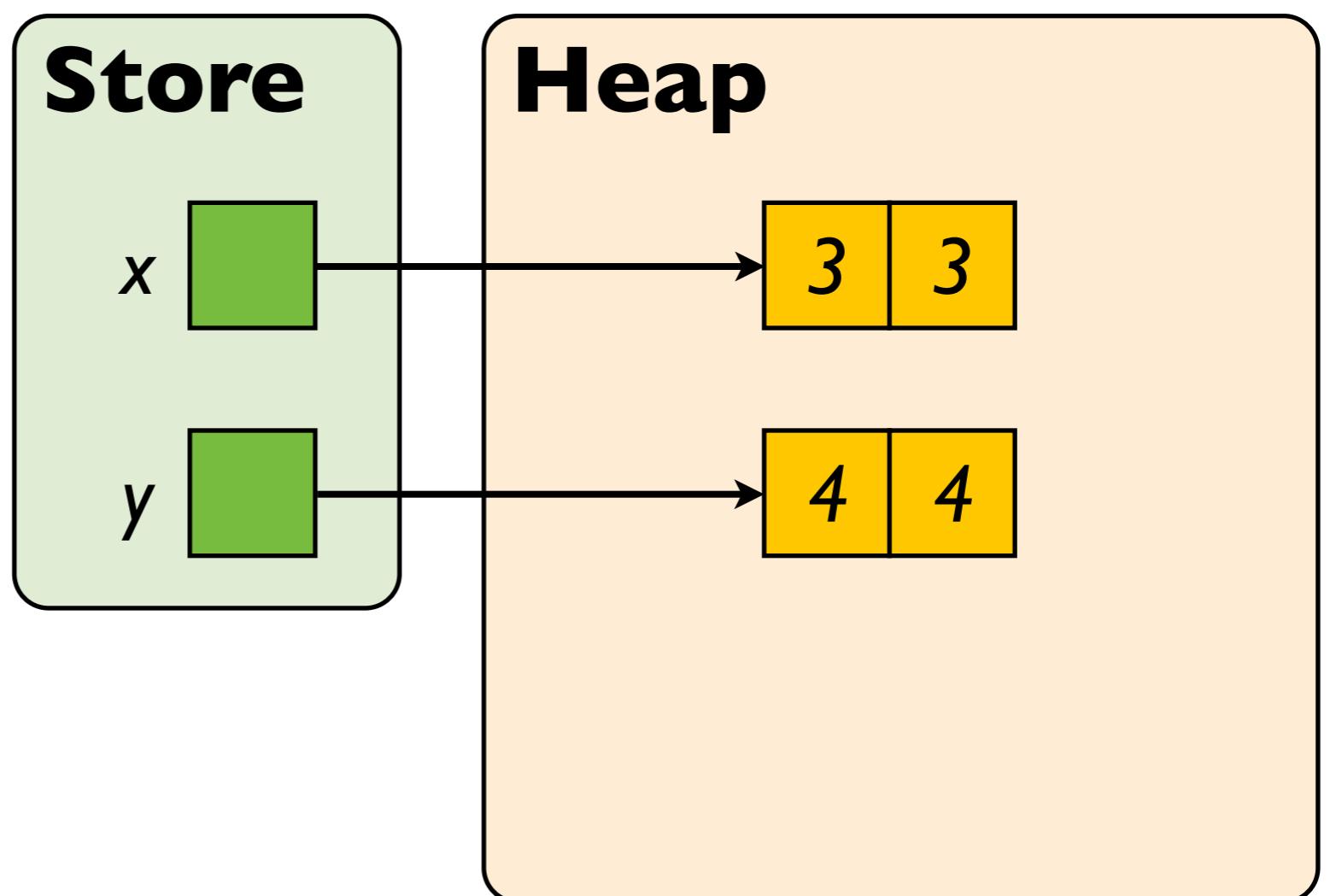
{emp}

x := cons(3,3);

{x |-> 3,3}

y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}



Proof outline

{emp}

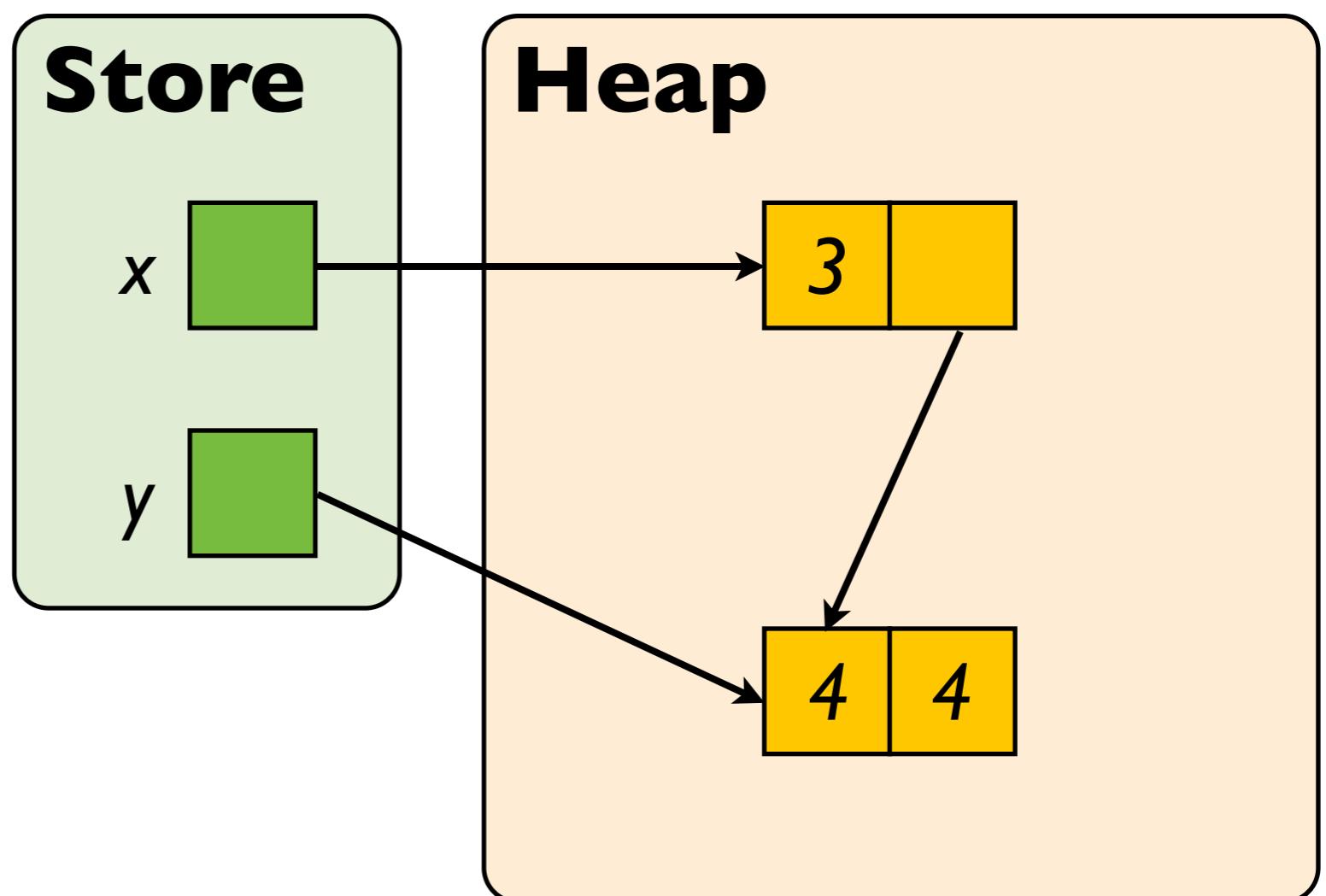
$x := \text{cons}(3,3);$

{ $x \rightarrow 3,3$ }

$y := \text{cons}(4,4);$

{ $x \rightarrow 3,3 * y \rightarrow 4,4$ }

$[x+1] := y;$



Proof outline

{emp}

$x := \text{cons}(3,3);$

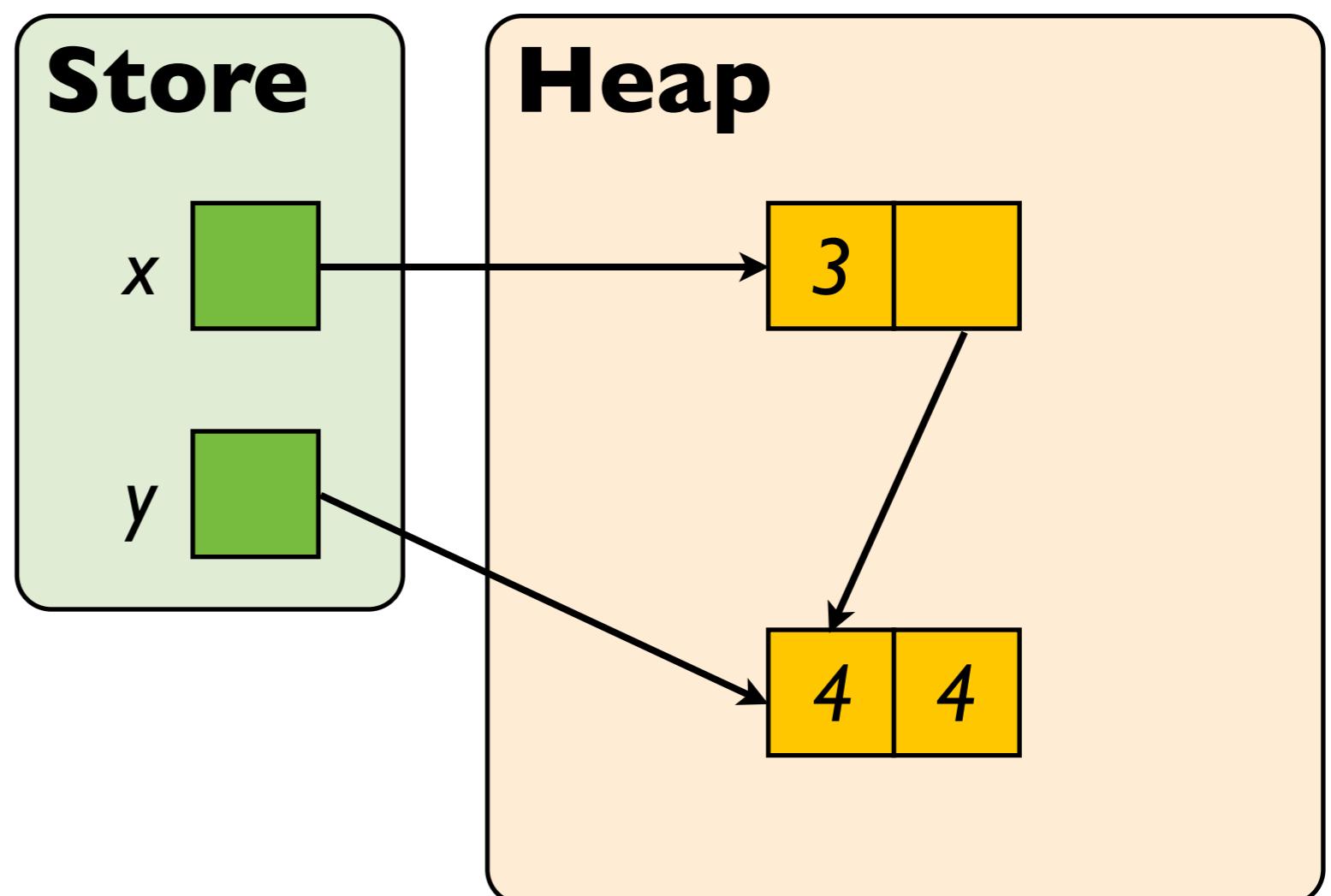
{ $x \rightarrow 3,3$ }

$y := \text{cons}(4,4);$

{ $x \rightarrow 3,3 * y \rightarrow 4,4$ }

{ $x \rightarrow 3 * x+1 \rightarrow 3$
 $* y \rightarrow 4,4$ }

$[x+1] := y;$



Proof outline

{emp}

$x := \text{cons}(3,3);$

{ $x |-> 3,3$ }

$y := \text{cons}(4,4);$

{ $x |-> 3,3 * y |-> 4,4$ }

{ $x |-> 3 * x+| |-> 3$
 $* y |-> 4,4$ }

$[x+|] := y;$

{ $x |-> 3 * x+| |-> y$
 $* y |-> 4,4$ }

{ $x+| |-> 3$
 $[x+|] := y;$
 $x+| |-> y$ }



frame rule!

Proof outline

{emp}

$x := \text{cons}(3,3);$

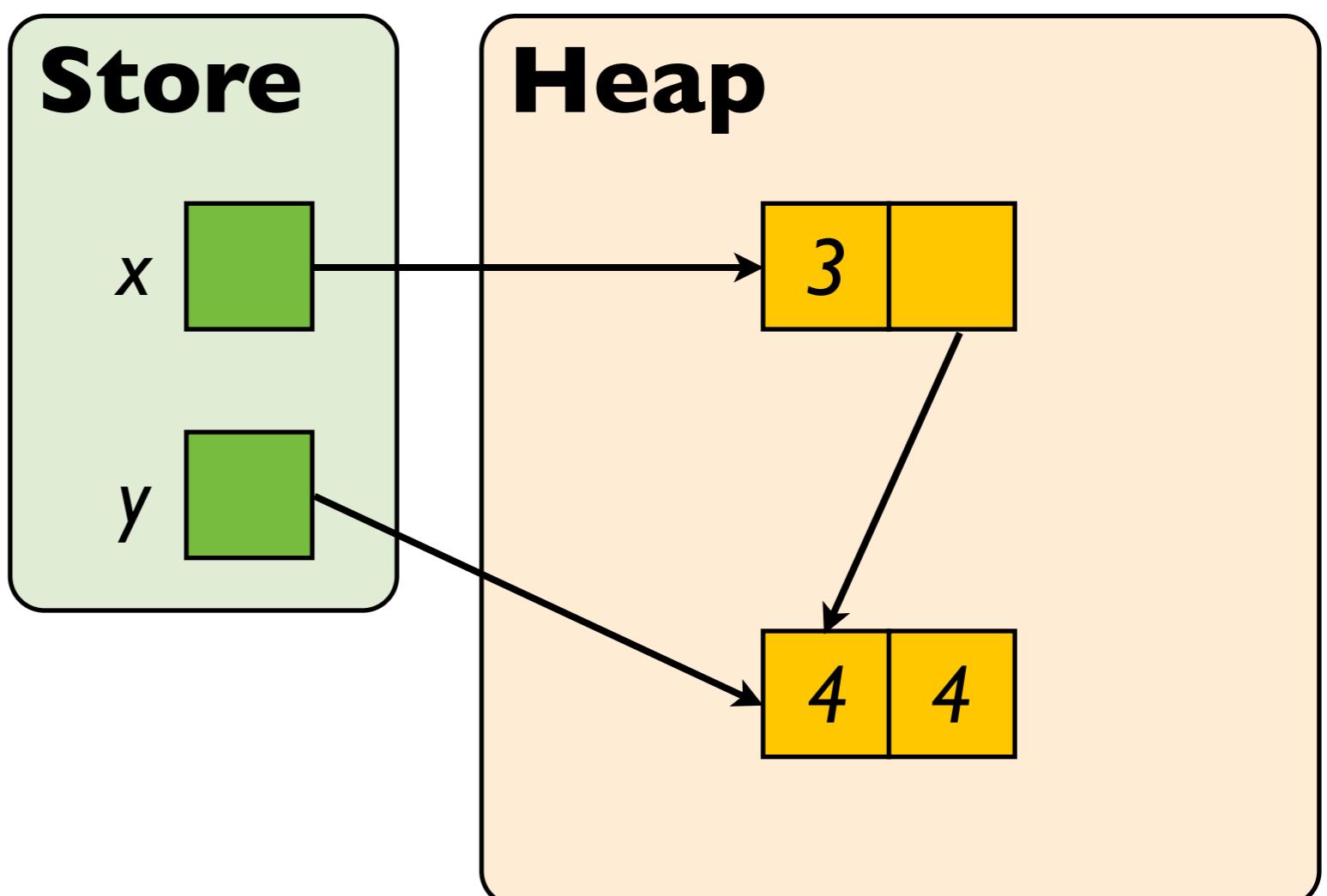
{ $x \rightarrow 3,3$ }

$y := \text{cons}(4,4);$

{ $x \rightarrow 3,3 * y \rightarrow 4,4$ }

$[x+1] := y;$

{ $x \rightarrow 3,y * y \rightarrow 4,4$ }



Proof outline

{emp}

$x := \text{cons}(3,3);$

{ $x \rightarrow 3,3$ }

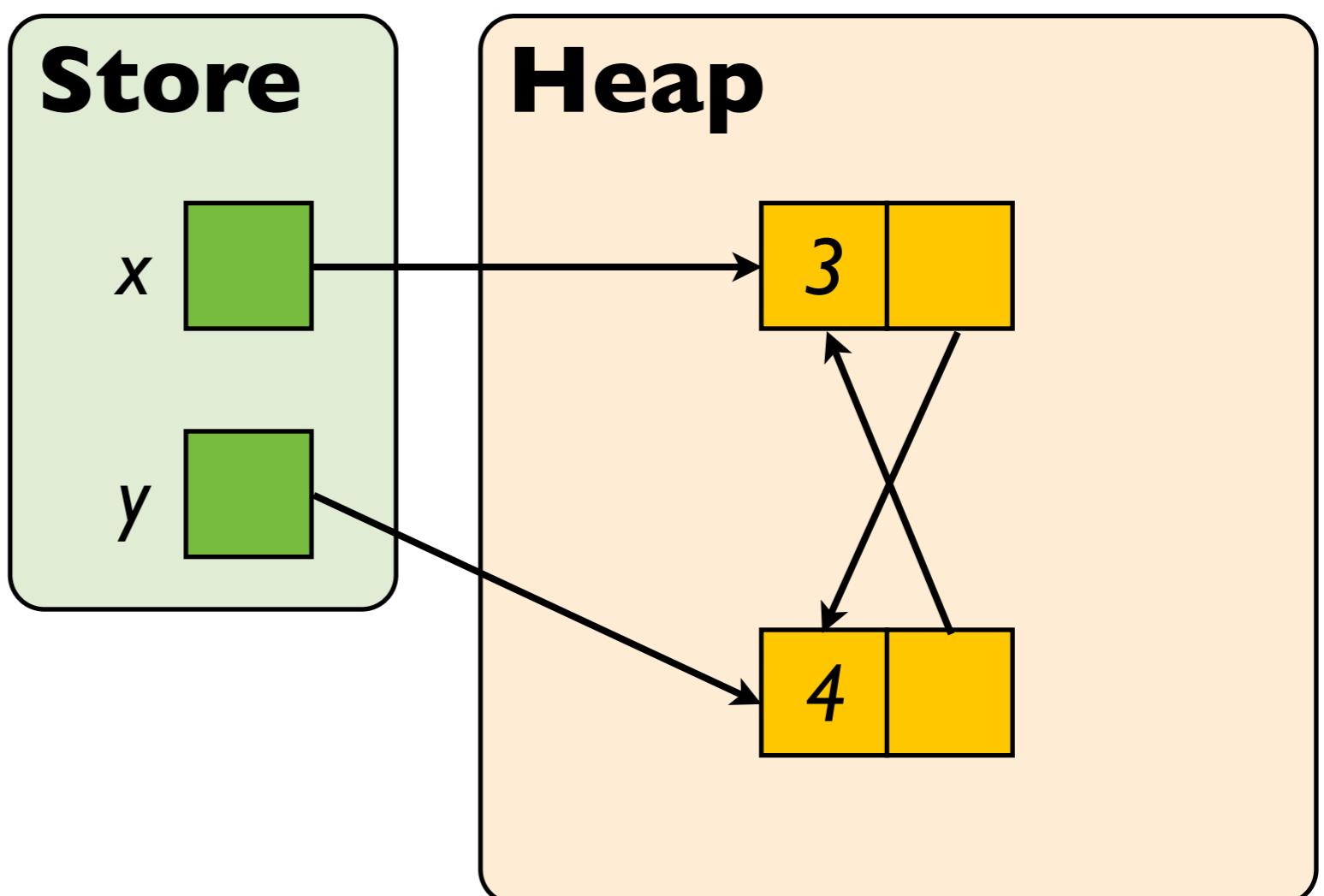
$y := \text{cons}(4,4);$

{ $x \rightarrow 3,3 * y \rightarrow 4,4$ }

$[x+1] := y;$

{ $x \rightarrow 3,y * y \rightarrow 4,4$ }

$[y+1] := x;$



Proof outline

{emp}

$x := \text{cons}(3,3);$

{ $x \rightarrow 3,3$ }

$y := \text{cons}(4,4);$

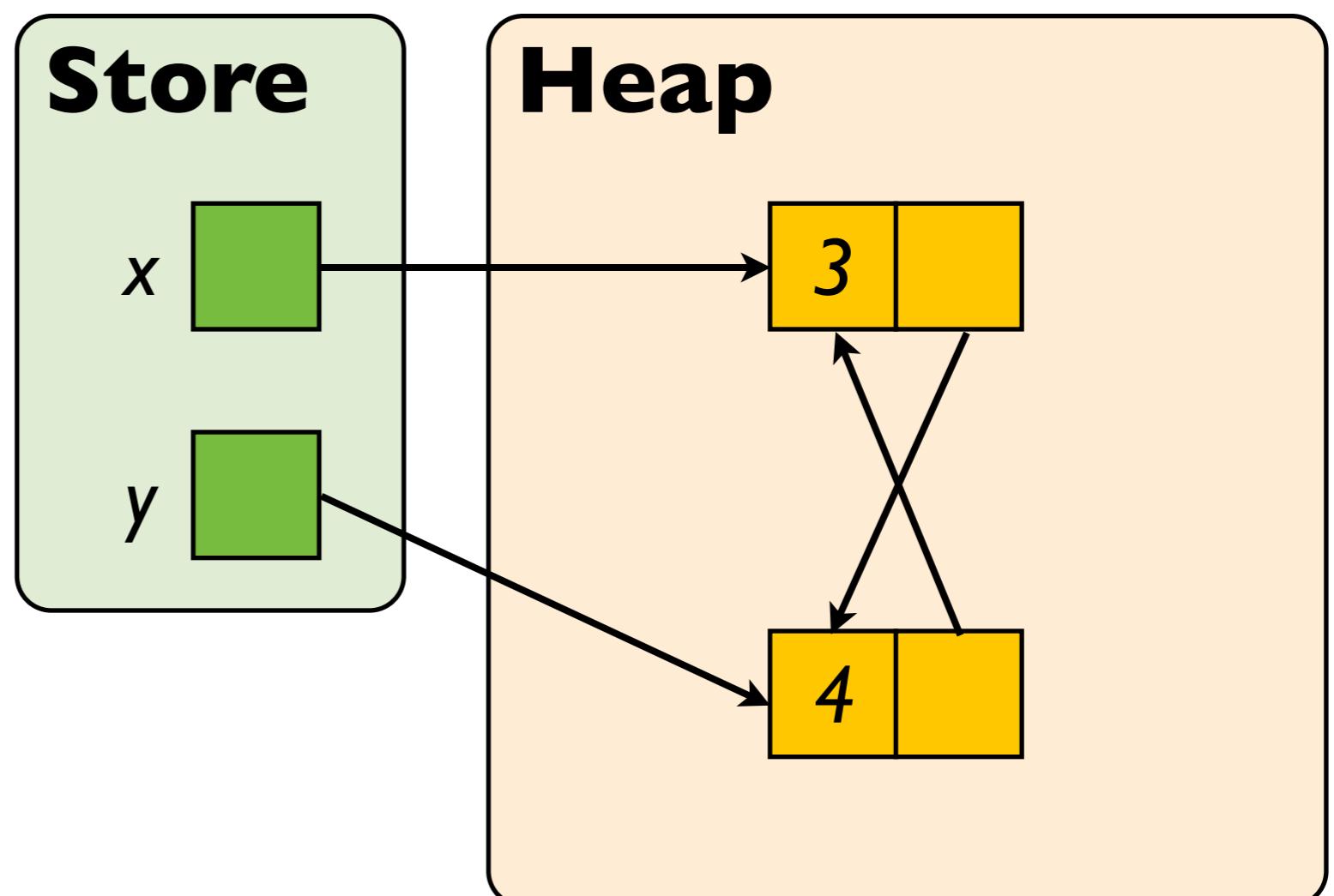
{ $x \rightarrow 3,3 * y \rightarrow 4,4$ }

$[x+1] := y;$

{ $x \rightarrow 3, y \rightarrow 4,4$ }

$[y+1] := x;$

{ $x \rightarrow 3, y \rightarrow 4, x$ }



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

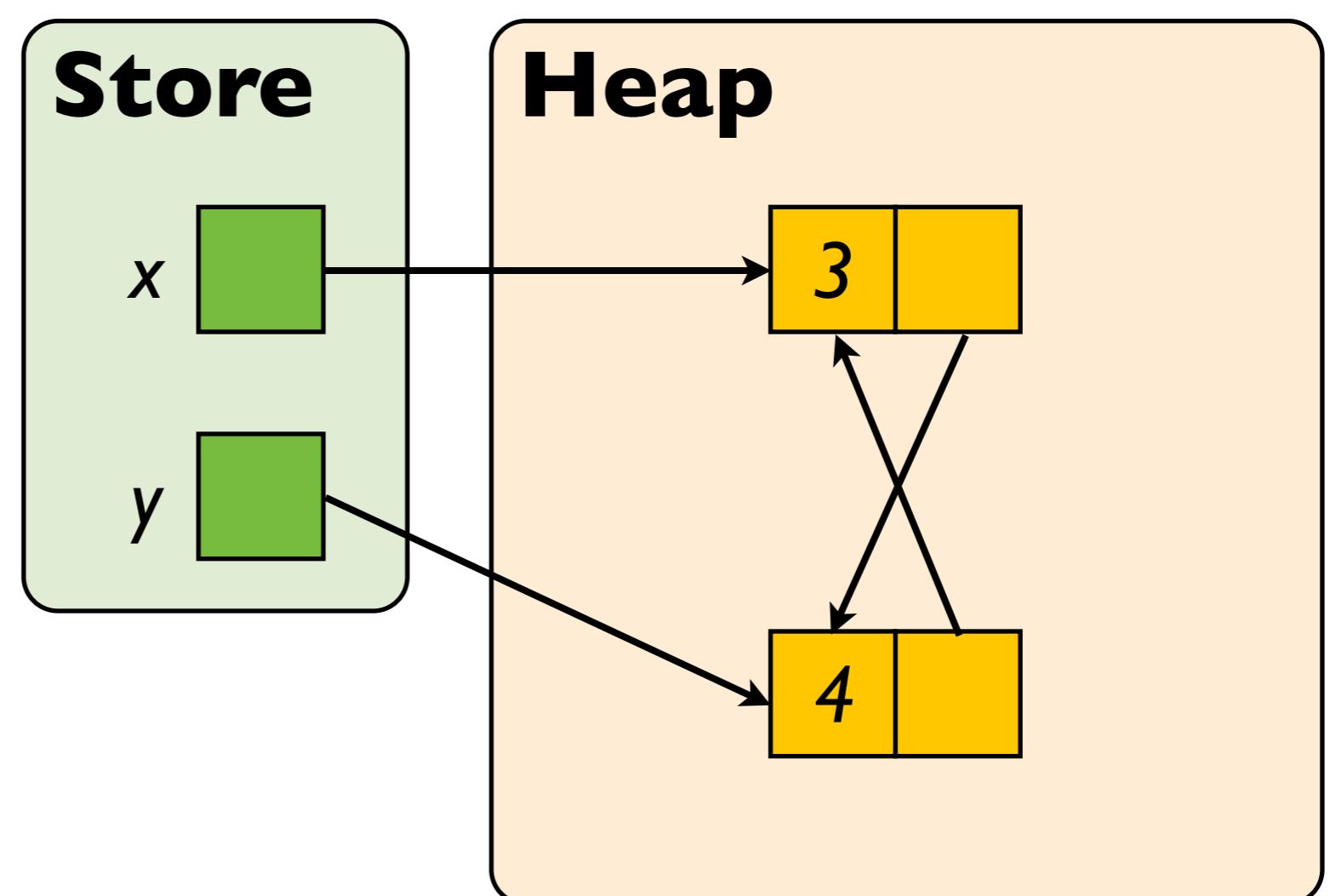
 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

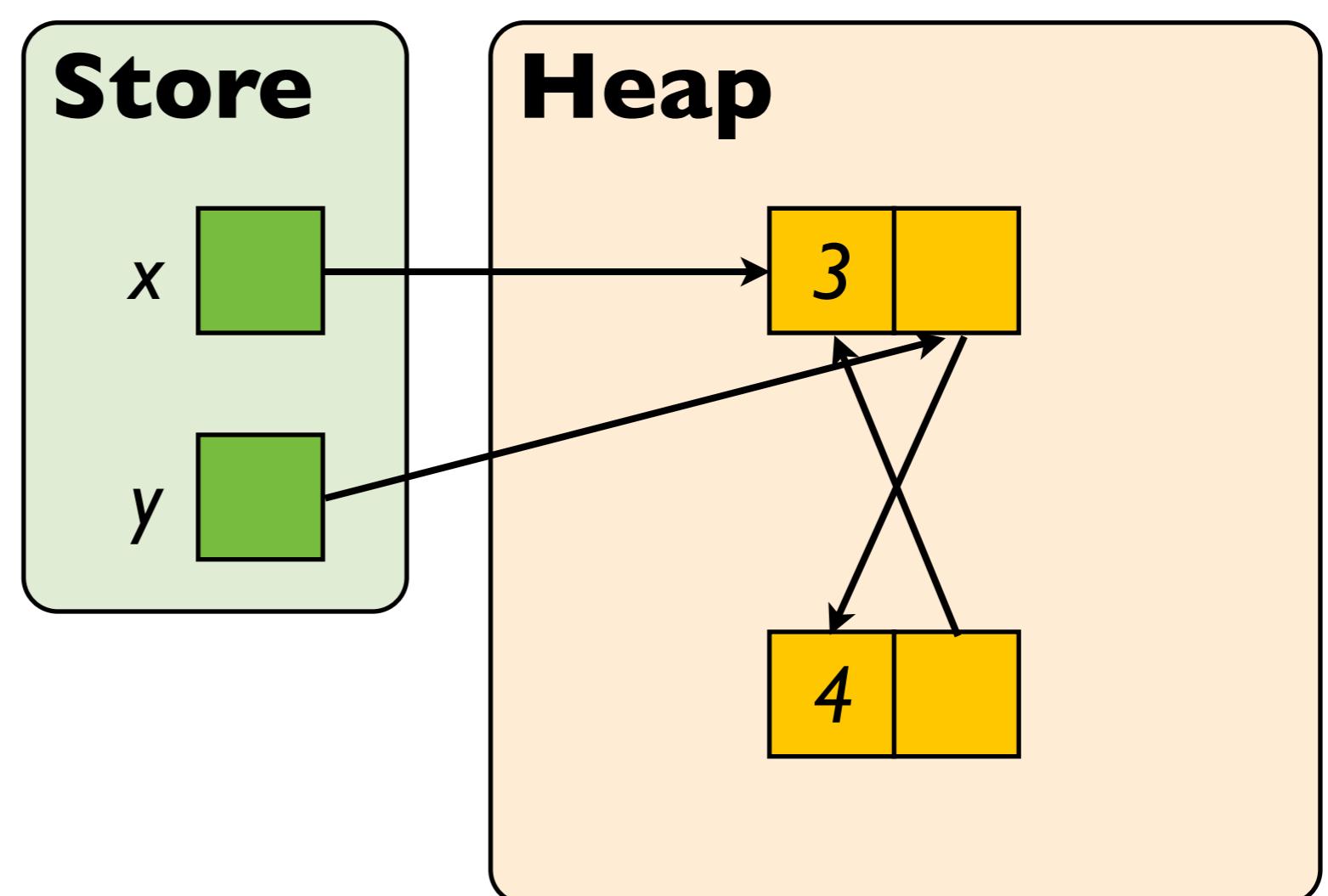
{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

Proof outline



via “forward” assignment axiom (from Hoare logic)

{x |-> 3,y * y |-> 4,x}

y := x+l

{x |-> 3,y^{old} * y^{old} |-> 4,x \wedge y = x+l }

{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

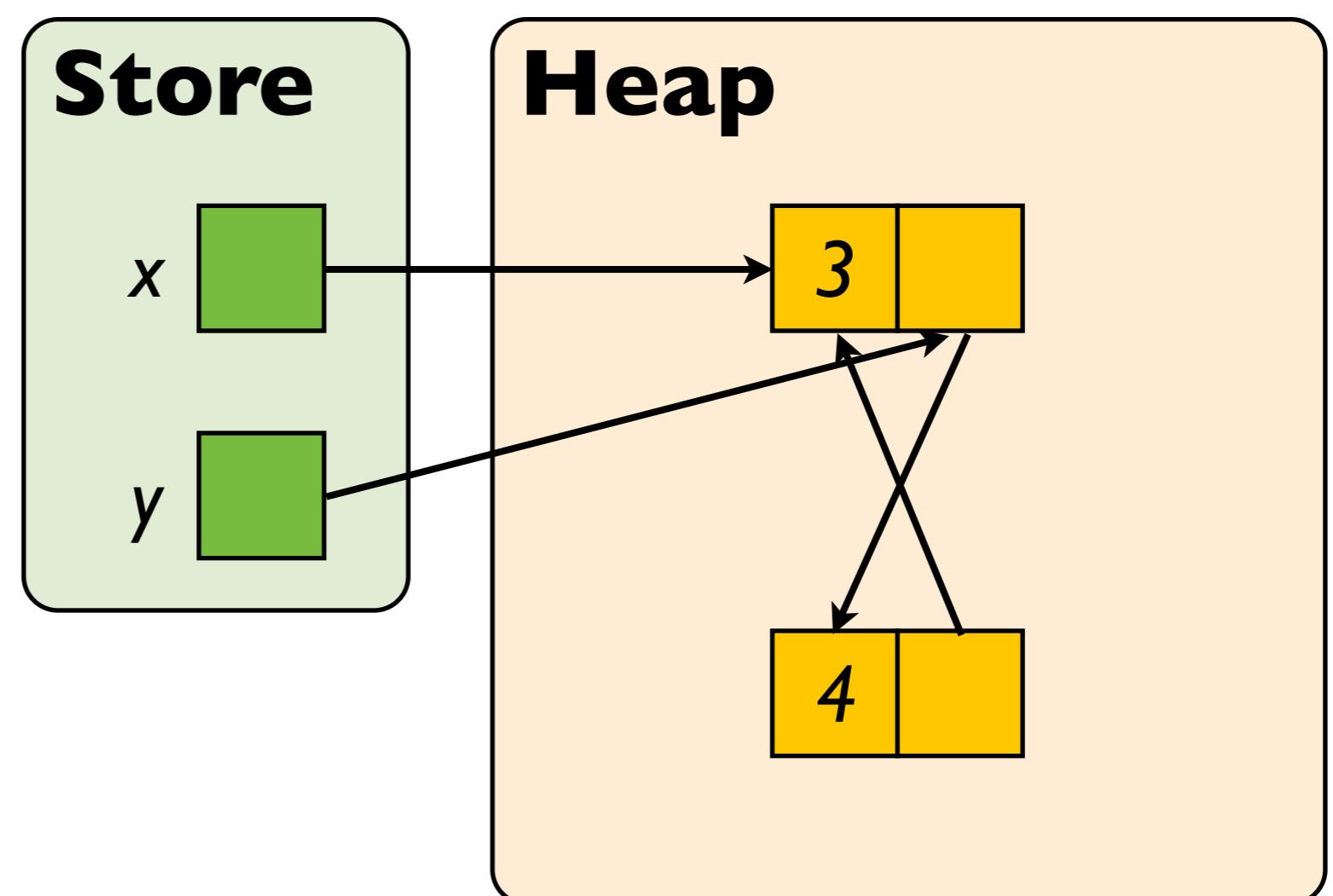
 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

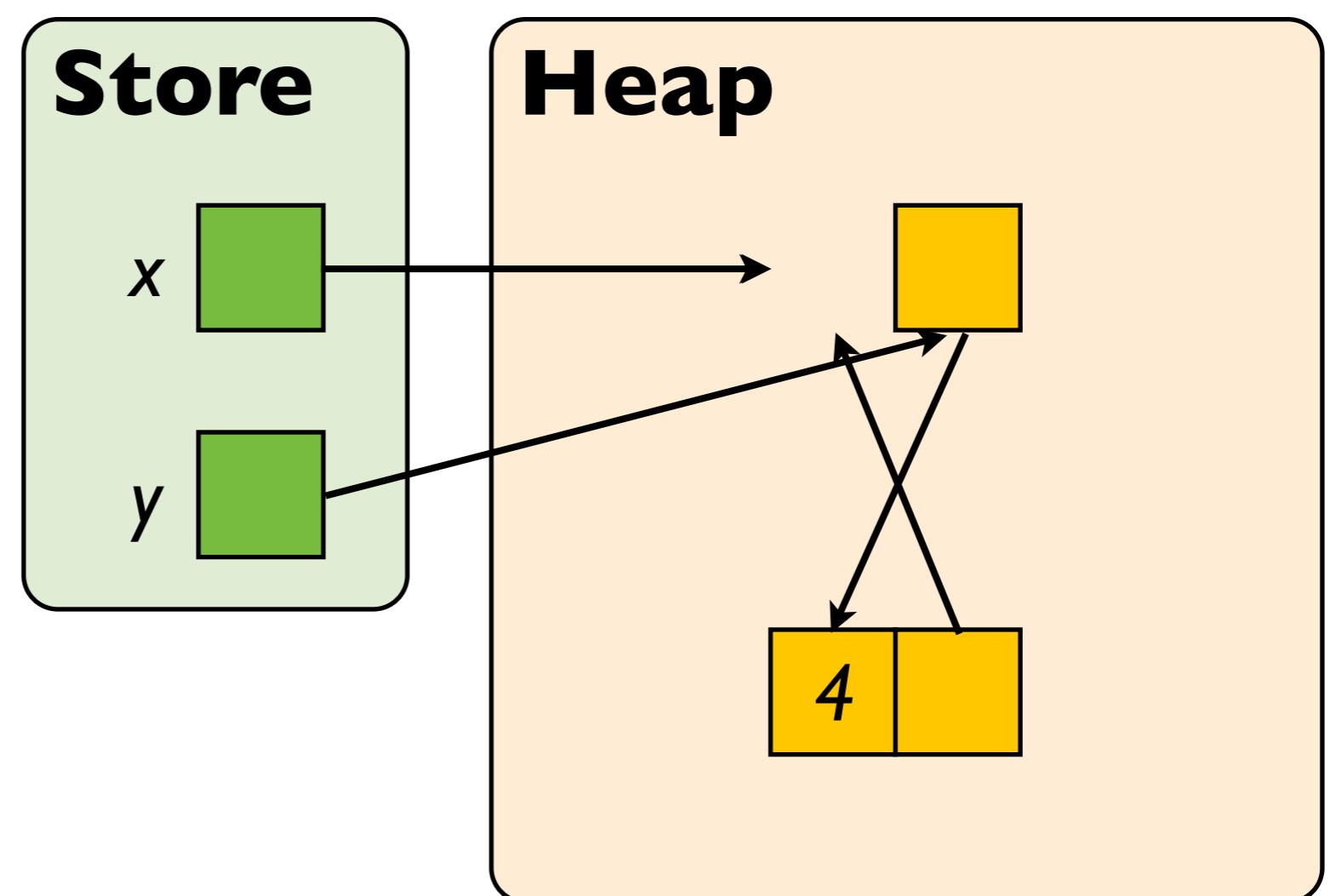
{x |-> 3,y * y |-> 4,x}

 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

dispose x;

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

 dispose x;

{emp * x+l |-> y^{old} *
y^{old} |-> 4,x ^ y = x+l}

Proof outline

{x |-> 3}

 dispose x;

{emp}



frame rule!

{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

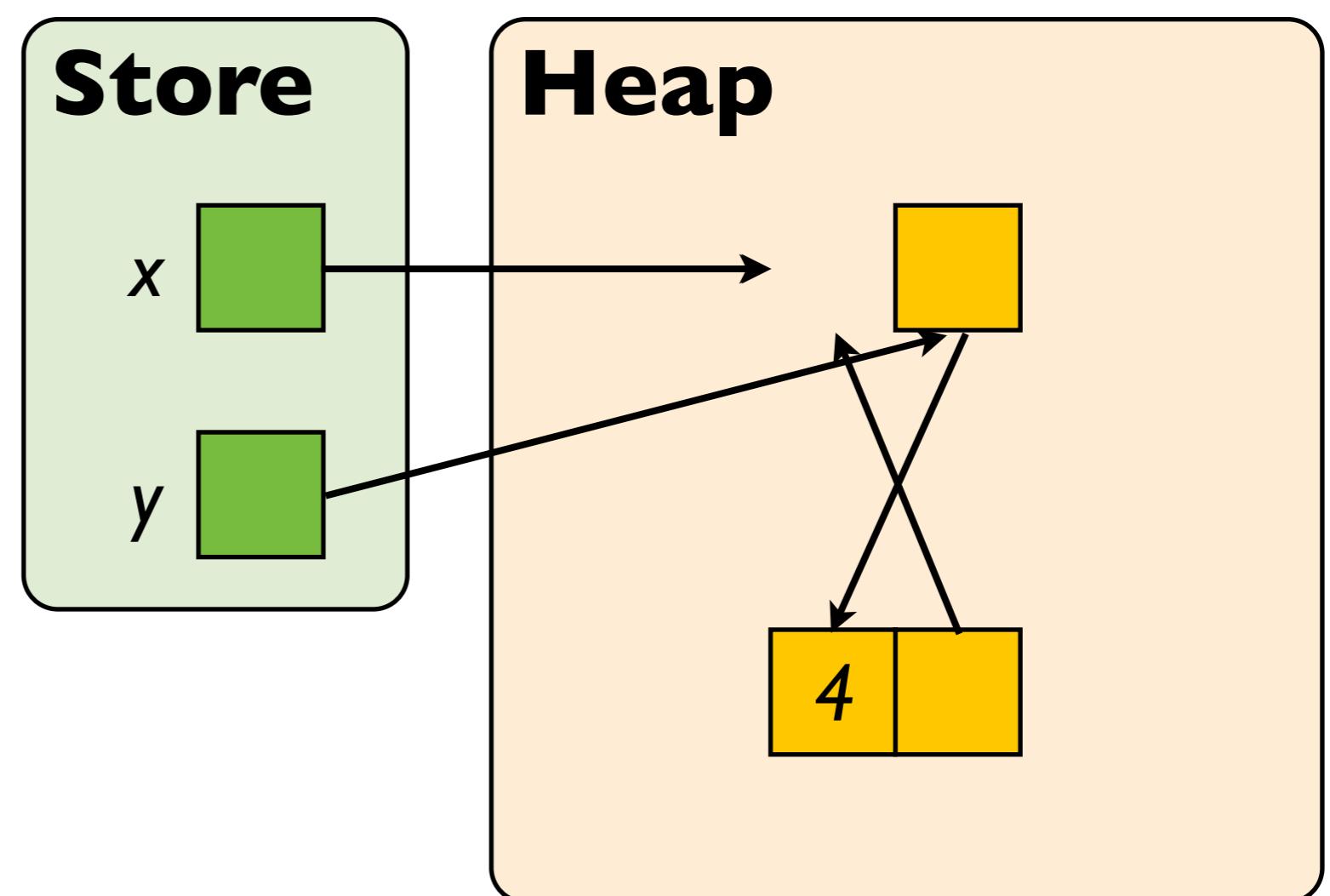
 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

dispose x;

{x+l |-> y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

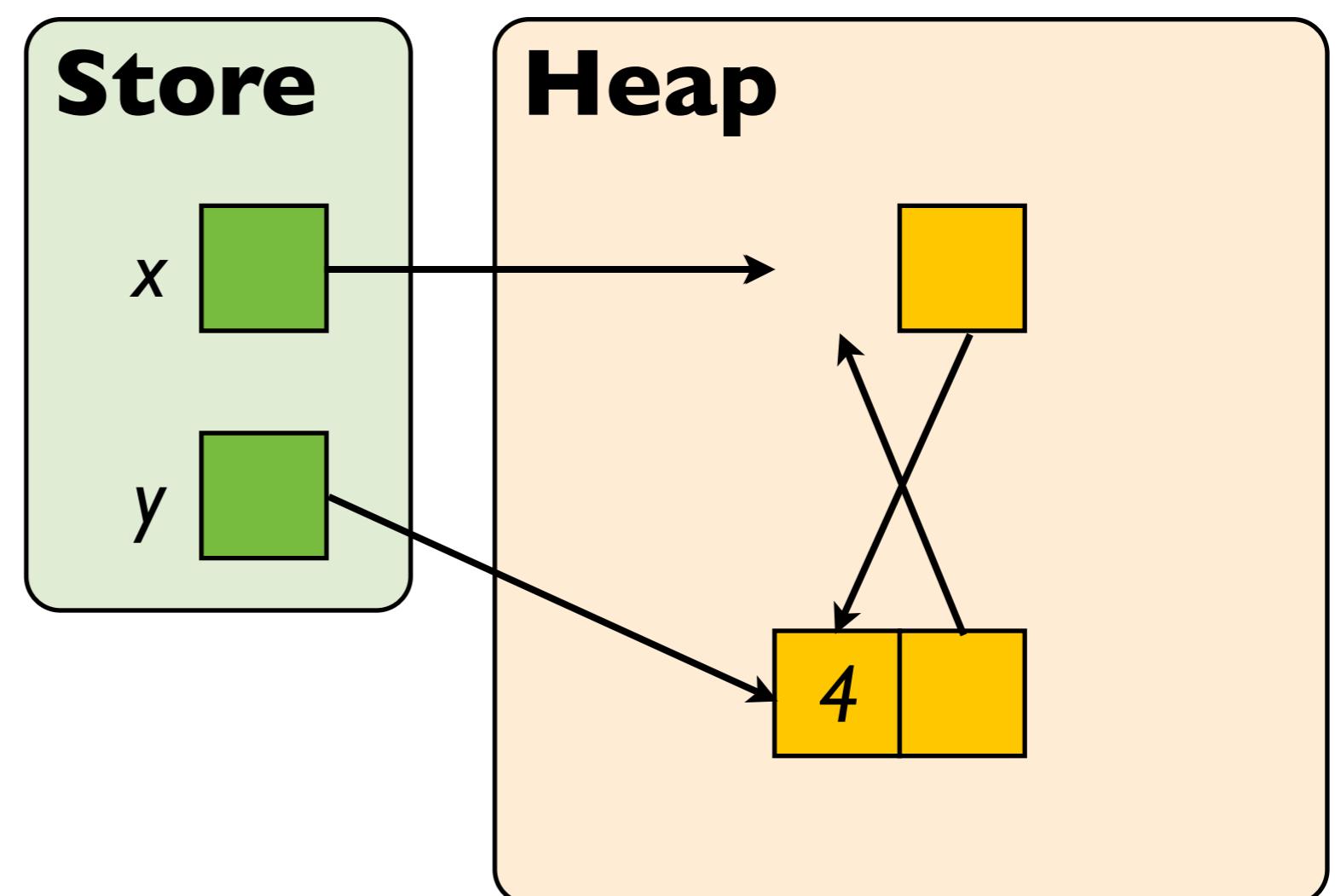
{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

dispose x;

{x+l |-> y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

y := [y];

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

 dispose x;

{x+l |-> y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

 y := [y];

Proof outline



frame rule and consequence!

{x+l = y ^ y |-> y^{old} }

y := [y];

{x+l |-> y^{old} ^ y^{old} = y}

{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

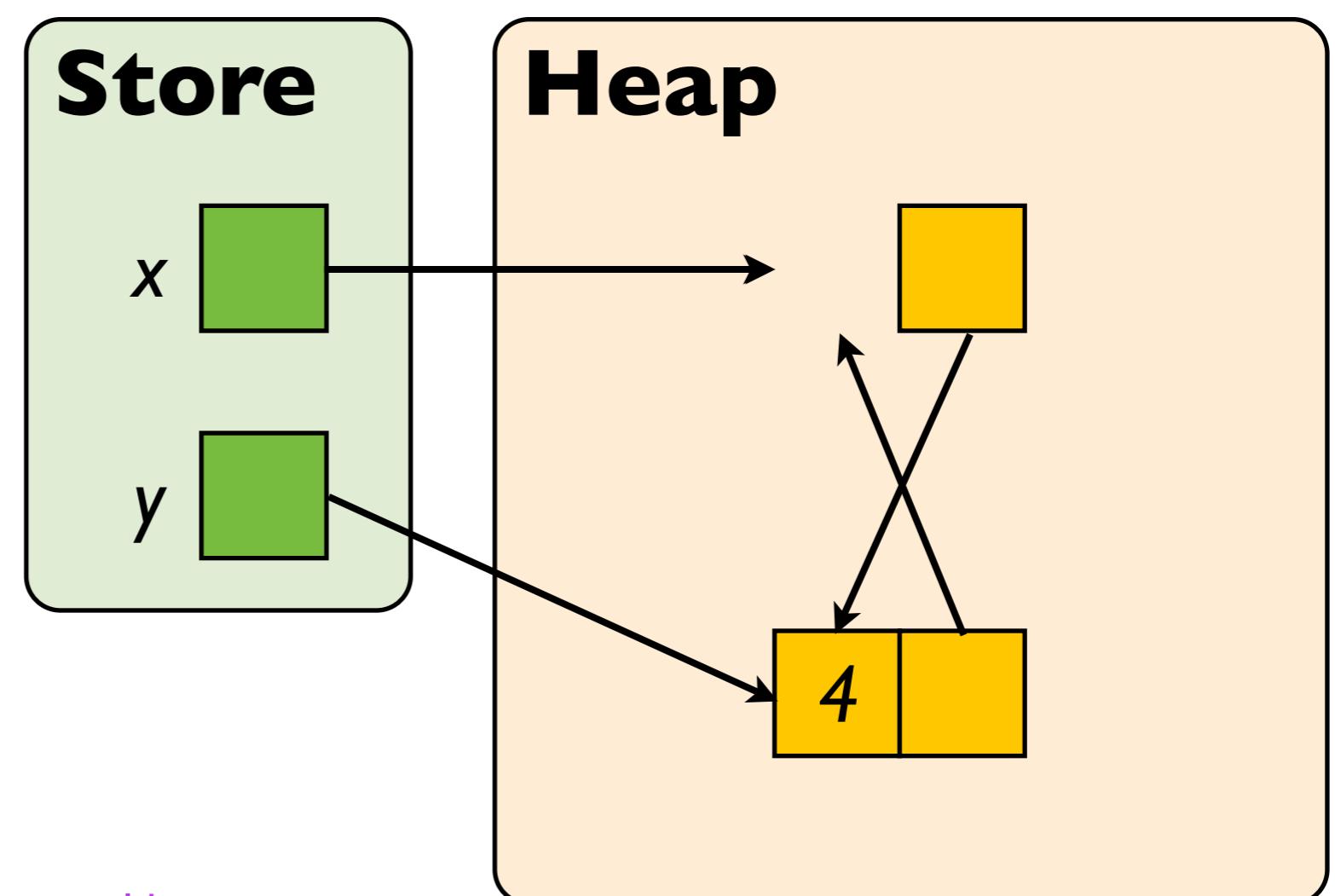
dispose x;

{x+l |-> y^{old} * y^{old} |-> 4,x
 ^ y = x+l }

 y := [y];

{x+l |-> y^{old} * y^{old} |-> 4,x ^ y = y^{old} }

Proof outline



{emp}

 x := cons(3,3);

{x |-> 3,3}

 y := cons(4,4);

{x |-> 3,3 * y |-> 4,4}

 [x+l] := y;

{x |-> 3,y * y |-> 4,4}

 [y+l] := x;

{x |-> 3,y * y |-> 4,x}

 y := x+l;

{x |-> 3,y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

dispose x;

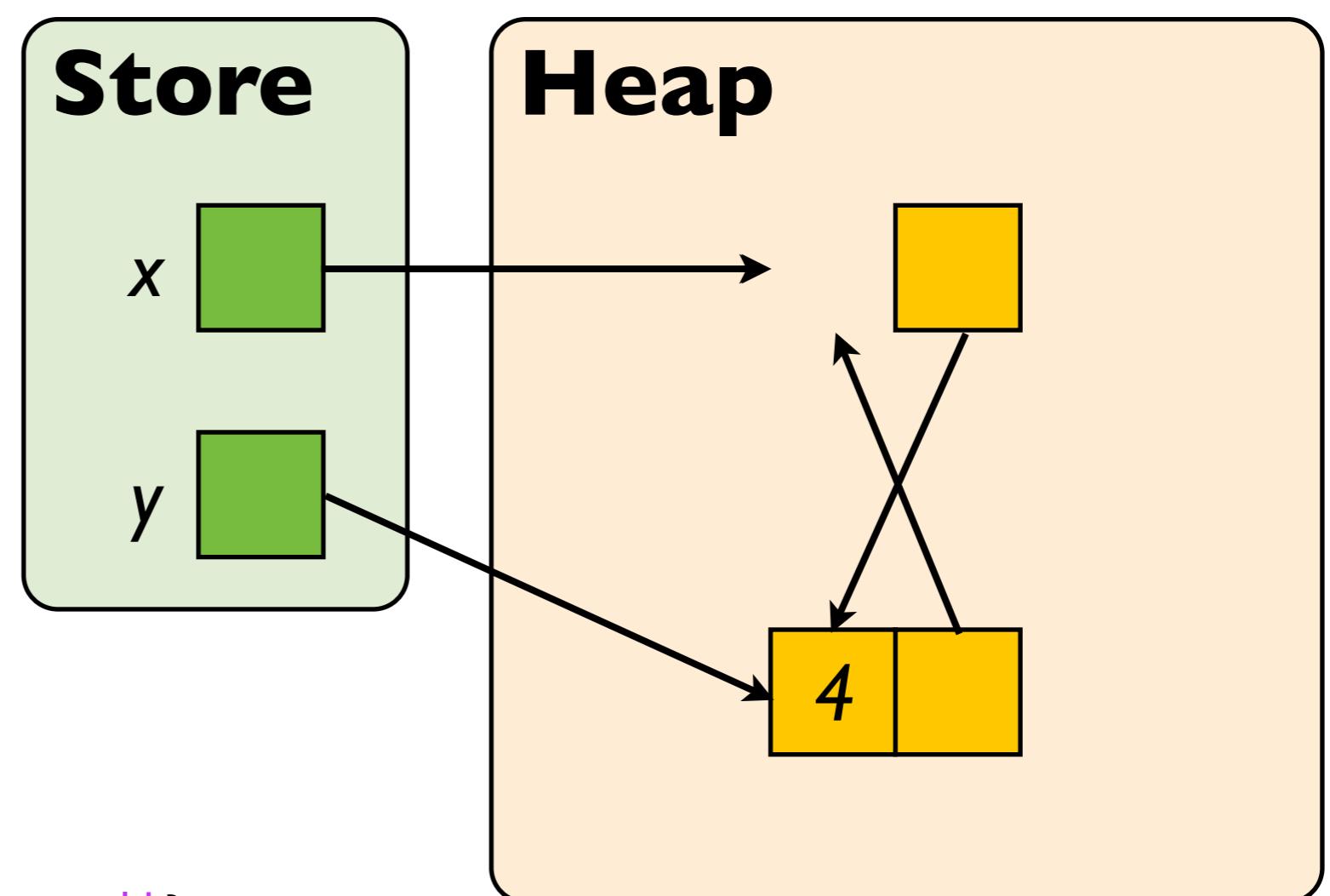
{x+l |-> y^{old} * y^{old} |-> 4,x
 ^ y = x+l}

 y := [y];

{x+l |-> y^{old} * y^{old} |-> 4,x ^ y = y^{old} }

{y |-> 4 * true}

Proof outline

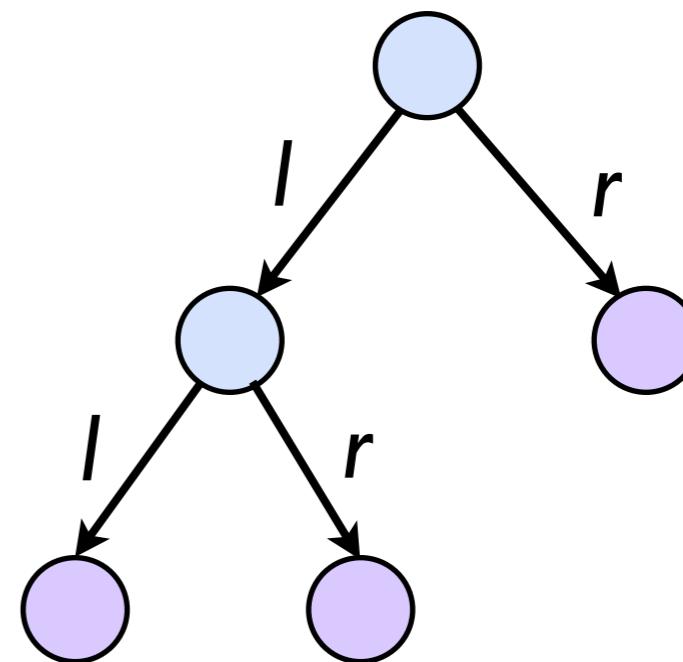


Inductive definitions in assertions

- heap portions in more realistic programs might comprise e.g. a tree, linked list
- helpful and concise to define such structures as predicates

Tree disposal

```
procedure DispTree(p)
local i, j;
if ¬isatom?(p) then
    i := p→l;
    j := p→r;
    DispTree(i)
    DispTree(j)
    dispose(p)
```



Tree predicate

$$\text{tree}(e) \iff$$

if $\text{isAtom}(e)$ then emp
else $\exists x, y. e \mapsto x, y * \text{tree}(x) * \text{tree}(y)$

- notes:
 - **isAtom(e)** returns true if e is an atomic value (e.g. characters) and not a location
 - if-then-else is easily compilable to logic (**how?**)

Tree disposal

```
procedure DispTree(p)
local i, j;
if ¬isatom?(p) then
    i := p→l;
    j := p→r;
    DispTree(i)
    DispTree(j)
    dispose(p)
```

{tree(p)} DispTree(p) {emp}

Tree disposal

```
procedure DispTree(p)
local i, j;
if ¬isatom?(p) then
    i := p→l;
    j := p→r;
    DispTree(i)
    DispTree(j)
    dispose(p)
```

we first:

- adjust for our store/heap model
- focus the proof on the crucial part



{tree(p)} DispTree(p) {emp}

Tree disposal proof

(from O'Hearn)

{ $P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)$ }

$i := [P];$

$j := [P + I];$

$\text{DispTree}(i);$

$\text{DispTree}(j);$

$\text{dispose}(P);$

$\text{dispose}(P + I);$

{emp}

Tree disposal proof

(from O'Hearn)

{ $P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)$ }

$i := [P];$

Tree disposal proof

(from O'Hearn)

$\{P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)\}$

$i := [P];$

$\{P \dashv\rightarrow x\}$
 $i := [P]$
 $\{P \dashv\rightarrow x \wedge x = i\}$



frame rule!

$\{P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y) \wedge x = i\}$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)\}$

Tree disposal proof

(from O'Hearn)

{ $P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)$ }

$i := [P];$

{ $P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)$ }

$j := [P + I];$

Tree disposal proof

(from O'Hearn)

$\{P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)\}$

$i := [P];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)\}$

$j := [P + I];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(j)\}$

Tree disposal proof

(from O'Hearn)

{ $P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)$ }

$i := [P];$

{ $P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)$ }

$j := [P + I];$

{ $P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(j)$ }

DispTree(i);

Tree disposal proof

(from O'Hearn)

$\{P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)\}$

$i := [P];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)\}$

$j := [P + I];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(j)\}$

$\text{DispTree}(i);$

$\{P \dashv\rightarrow x, y * \text{emp} * \text{tree}(j)\}$



frame rule!

$\{\text{tree}(i)\}$

$\text{DispTree}(i)$

$\{\text{emp}\}$

Tree disposal proof

(from O'Hearn)

$\{P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)\}$

$i := [P];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)\}$

$j := [P + I];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(j)\}$

$\text{DispTree}(i);$

$\{P \dashv\rightarrow x, y * \text{emp} * \text{tree}(j)\}$

$\text{DispTree}(j);$

Tree disposal proof

(from O'Hearn)

$\{P \dashv\rightarrow x, y * \text{tree}(x) * \text{tree}(y)\}$

$i := [P];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(y)\}$

$j := [P + I];$

$\{P \dashv\rightarrow x, y * \text{tree}(i) * \text{tree}(j)\}$

$\text{DispTree}(i);$

$\{P \dashv\rightarrow x, y * \text{emp} * \text{tree}(j)\}$

$\text{DispTree}(j);$

$\{P \dashv\rightarrow x, y * \text{emp} * \text{emp}\}$

{P |-> x,y * tree(x) * tree(y)}

i := [P];

{P |-> x,y * tree(i) * tree(y)}

j := [P+l];

{P |-> x,y * tree(i) * tree(j)}

DispTree(i);

{P |-> x,y * emp * tree(j)}

DispTree(j);

{P |-> x,y * emp * emp}

{P |-> x,y * tree(x) * tree(y)}

i := [P];

{P |-> x,y * tree(i) * tree(y)}

j := [P+l];

{P |-> x,y * tree(i) * tree(j)}

DispTree(i);

{P |-> x,y * emp * tree(j)}

DispTree(j);

{P |-> x,y * emp * emp}

dispose(p);

dispose(p+l);

{P |-> x,y * tree(x) * tree(y)}

i := [P];

{P |-> x,y * tree(i) * tree(y)}

j := [P+l];

{P |-> x,y * tree(i) * tree(j)}

DispTree(i);

{P |-> x,y * emp * tree(j)}

DispTree(j);

{P |-> x,y * emp * emp}

dispose(p);

dispose(p+l);

{emp * emp * emp}

{P |-> x,y * tree(x) * tree(y)}

i := [P];

{P |-> x,y * tree(i) * tree(y)}

j := [P+l];

{P |-> x,y * tree(i) * tree(j)}

DispTree(i);

{P |-> x,y * emp * tree(j)}

DispTree(j);

{P |-> x,y * emp * emp}

dispose(p);

dispose(p+l);

{emp * emp * emp}

{emp}

Next on the agenda

(1) model of program states for separation logic 

(2) assertions and spatial connectives 

(3) axioms and inference rules 

(4) program proofs 

In a nutshell



The **frame rule** is absolutely **key** to
separation logic proofs

$$\frac{\{p\} \quad C \quad \{q\}}{\{p * r\} \quad C \quad \{q * r\}}$$

Thank you! Questions?