A diagrammatic introduction to Separation Logic

John Wickerson, TU Berlin

Two 90-minute lectures, part of a course on 'Quality Assurance of Embedded Systems'

January 2013

Outline

- **A “VeriFast” introduction to Hoare logic**
- List reversal in Hoare logic
- List reversal in separation logic
- Proof rules for separation logic
- Program variables as resource

Hoare logic

- Invented by Tony Hoare (now at Microsoft Research Cambridge, UK) in 1969
- A formal mathematical system based on annotating program code with **assertions** that must hold whenever execution reaches that point
- Basic unit is the **Hoare triple**, written {p} C {q}
  - Hoare's original notation was p {C} q

Demo: simple examples in VeriFast

Meaning of Hoare triple

- What does {p} C {q} mean?
  - If C begins execution in a state satisfying p then any final state it reaches will satisfy q

Rules of Hoare logic

$$\frac{\{p \land b\}\ C\ \{p\}}{\{p\}\ \text{while } b \text{ do } C\ \{p \land \neg b\}}$$

$$\frac{\{p \land b\}\ C_1\ \{q\} \qquad \{p \land \neg b\}\ C_2\ \{q\}}{\{p\}\ \text{if } b \text{ then } C_1 \text{ else } C_2\ \{q\}}$$

$$\frac{}{\{p\}\ \text{skip}\ \{p\}}$$

$$\frac{\{p\}\ C_1\ \{q\} \qquad \{q\}\ C_2\ \{r\}}{\{p\}\ C_1\ ;\ C_2\ \{r\}}$$

Outline

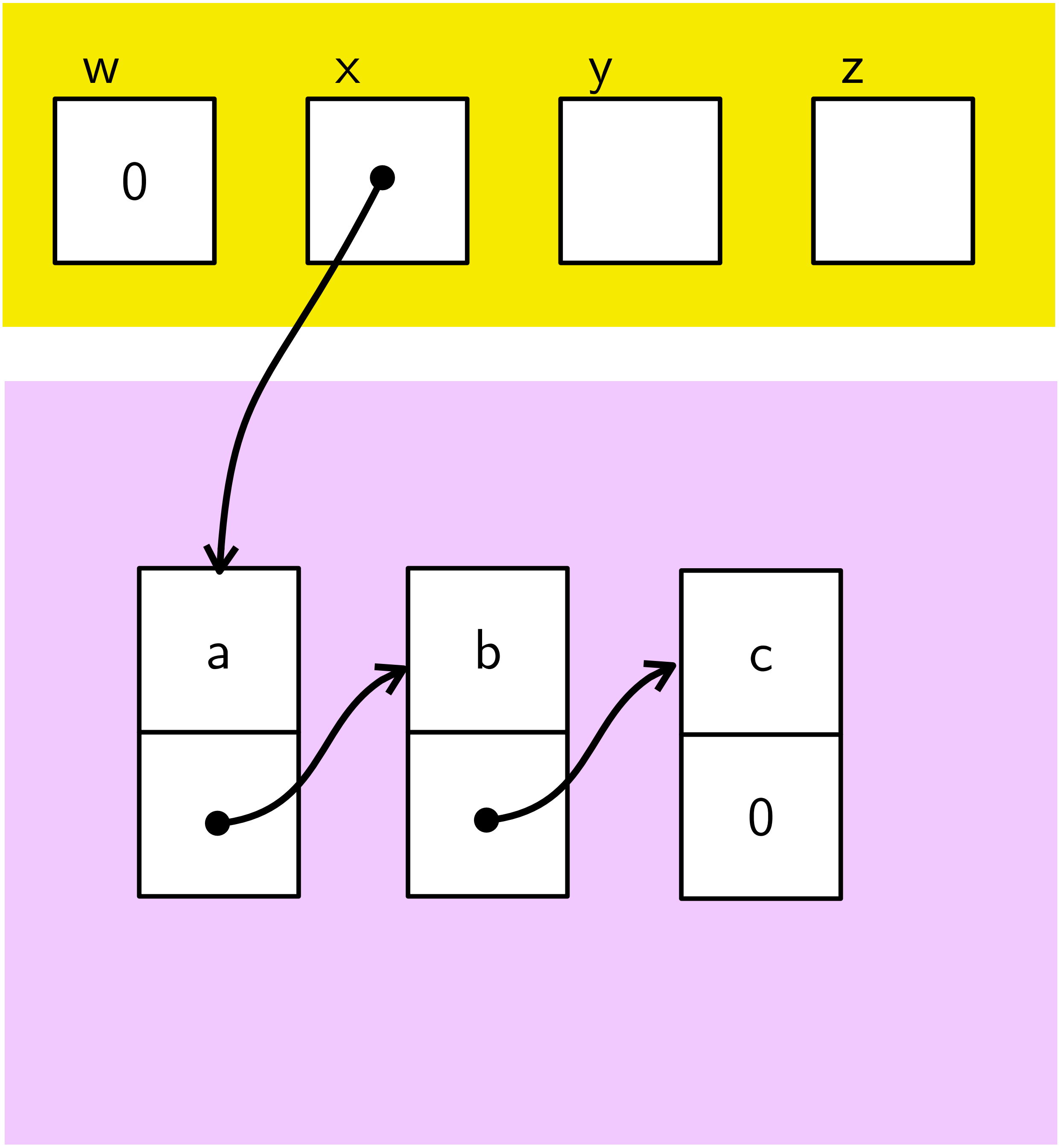
- A “VeriFast” introduction to Hoare logic
- **List reversal in Hoare logic**
- List reversal in separation logic
- Proof rules for separation logic
- Program variables as resource

Proof of list reverse

list δ x

```
w := 0;
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w

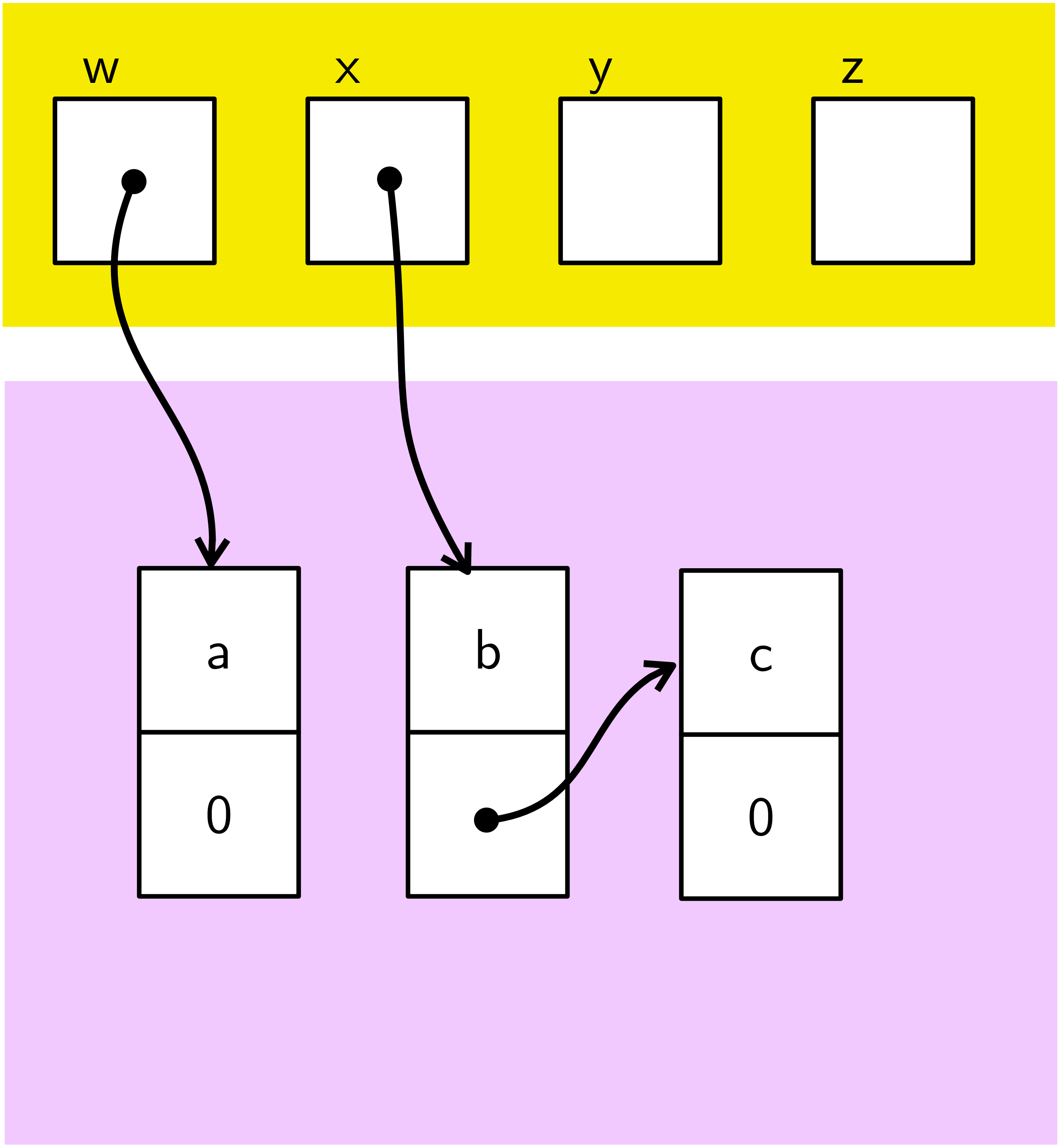


Proof of list reverse

list δ x

```
w := 0;
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w

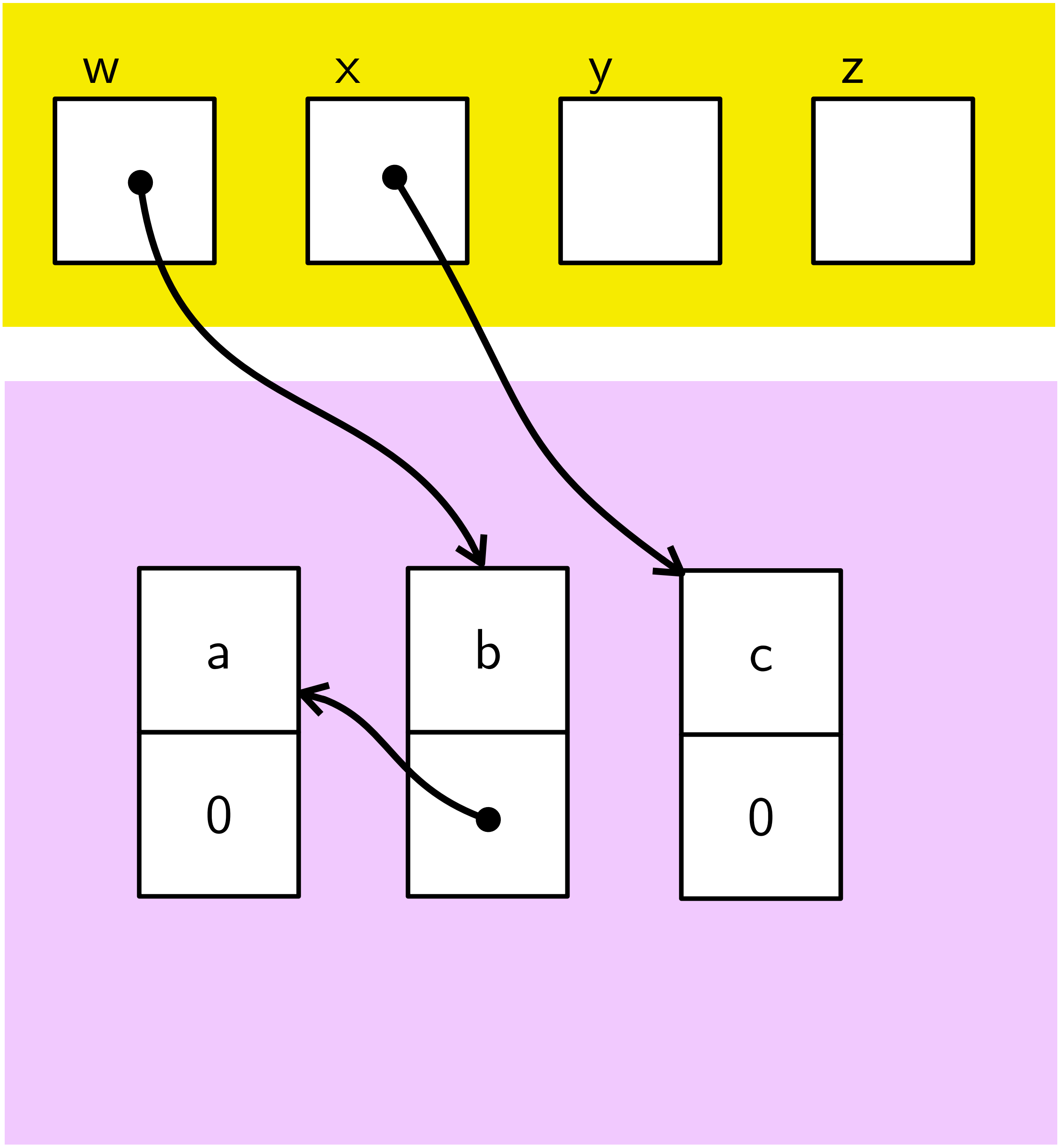


Proof of list reverse

list δ x

```
w := 0;
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w

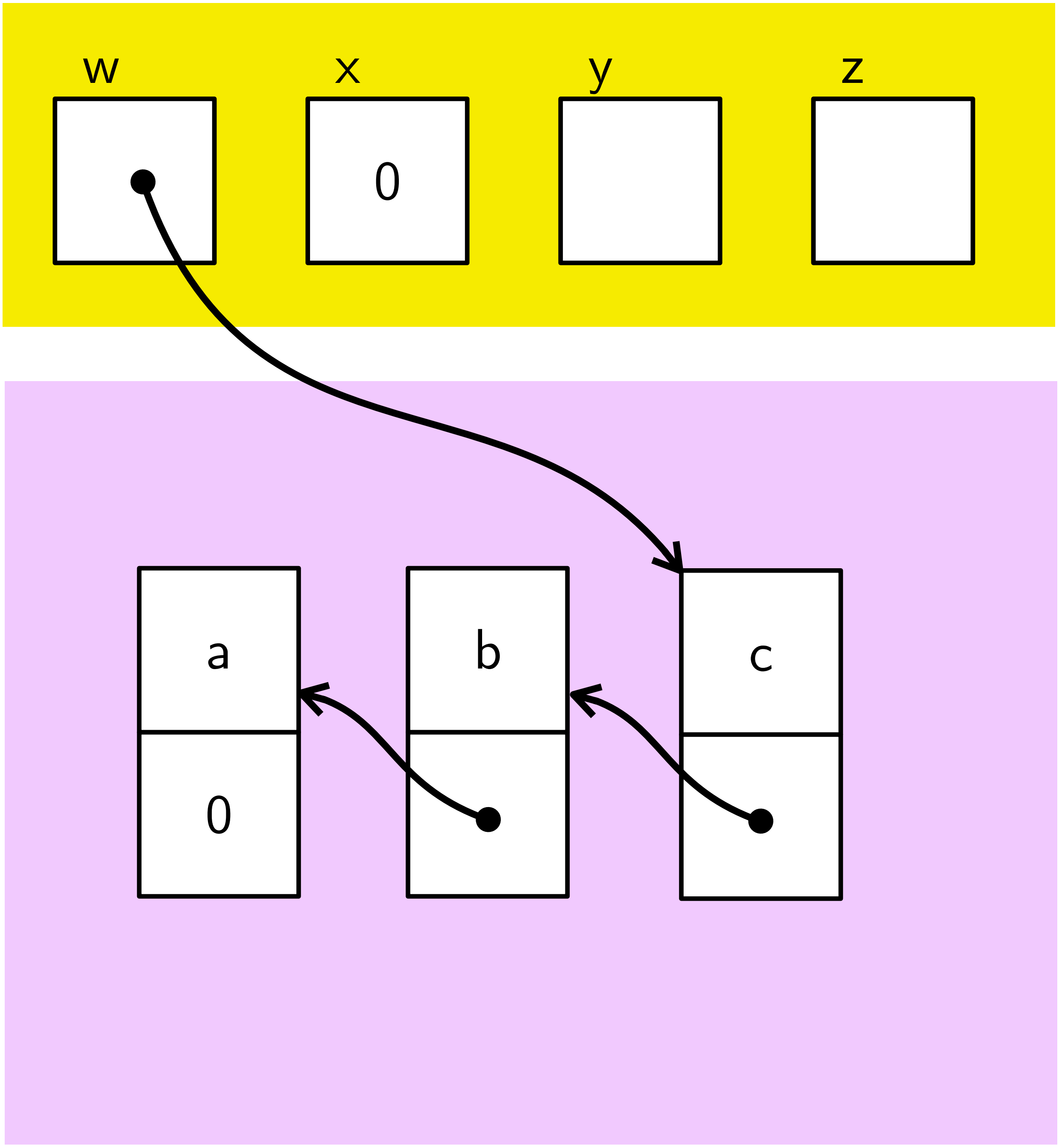


Proof of list reverse

list δ x

```
w := 0;
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w

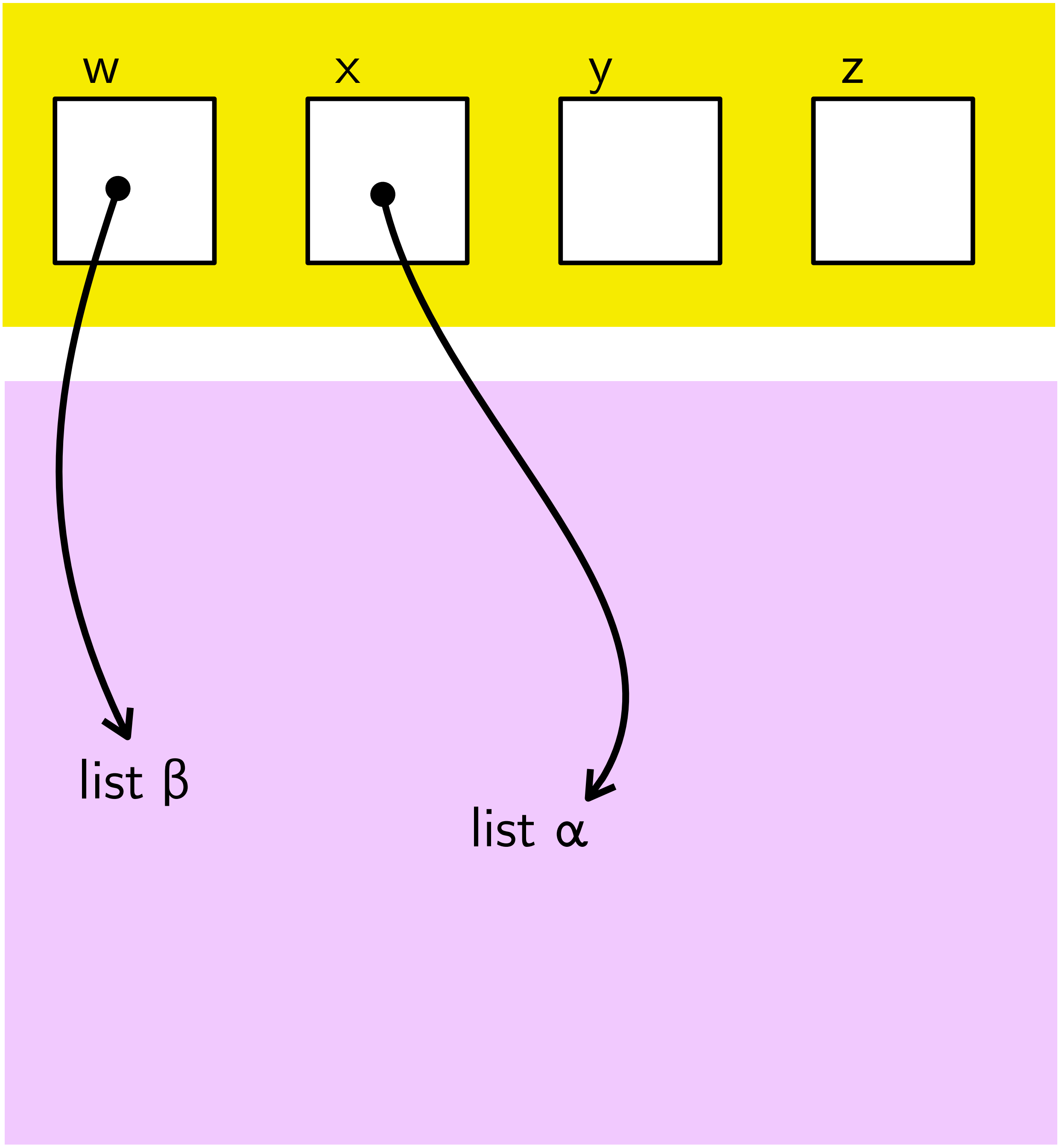


Proof of list reverse

list δ x

```
w := 0;
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w



$\delta = -\beta \cdot \alpha$

Proof of list reverse

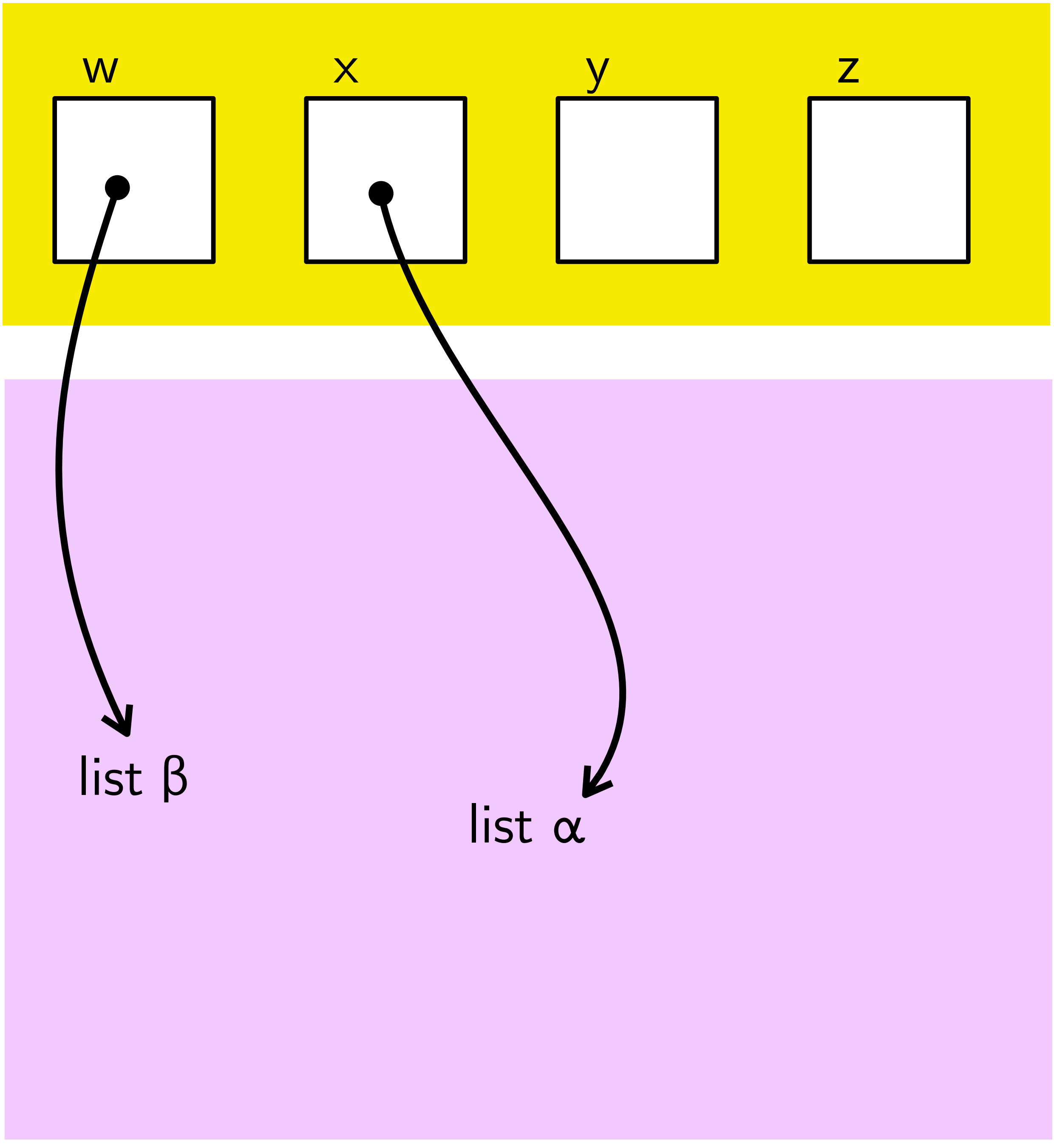
list δ x

```
w := 0;
```

∃α,β. list α x ∧ list β w ∧ δ = -β·α

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w



δ = -β · α

Proof of list reverse

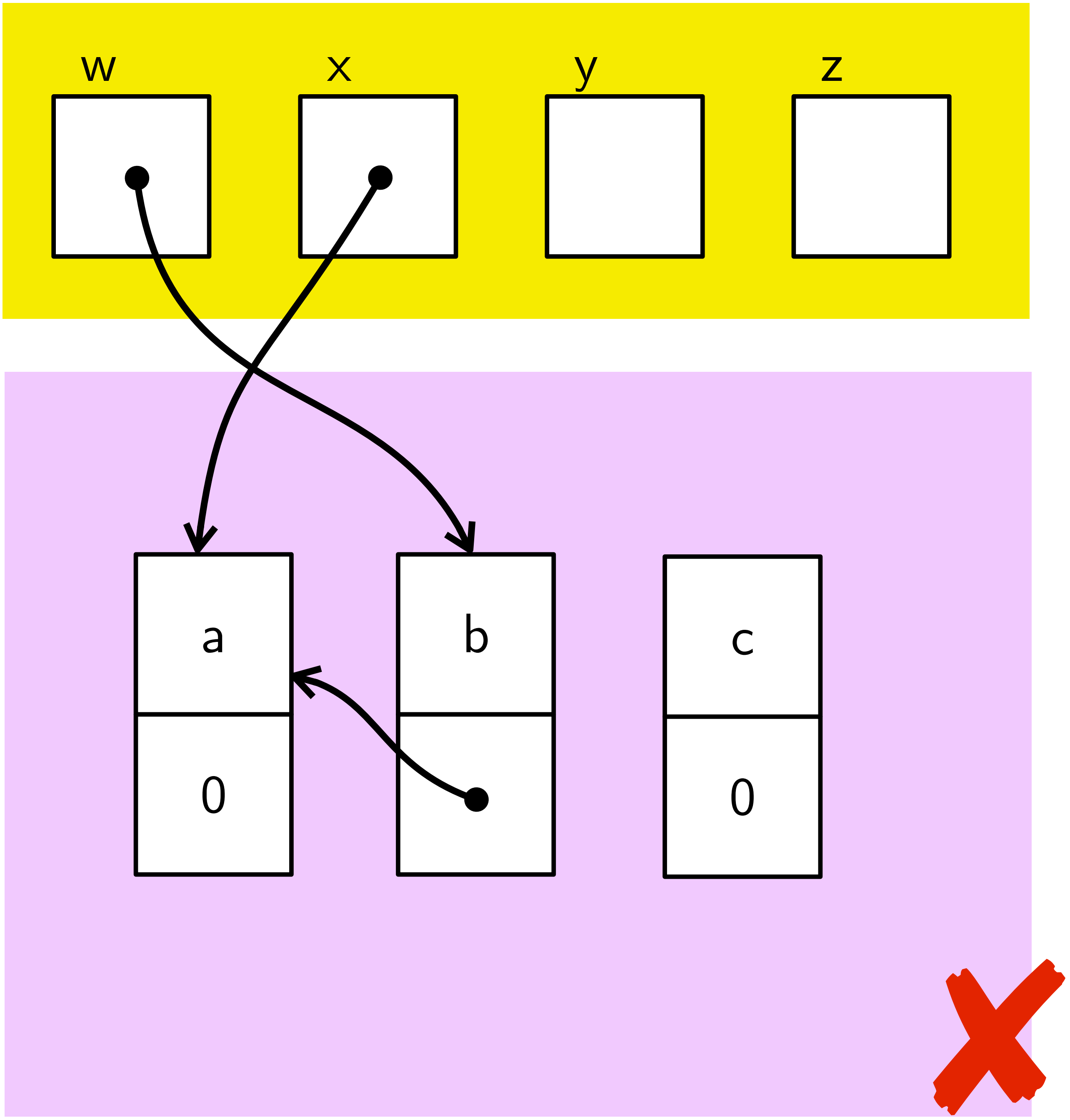
list δ x

```
w := 0;
```

∃α,β. list α x ∧ list β w ∧ δ = -β·α

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w



Proof of list reverse

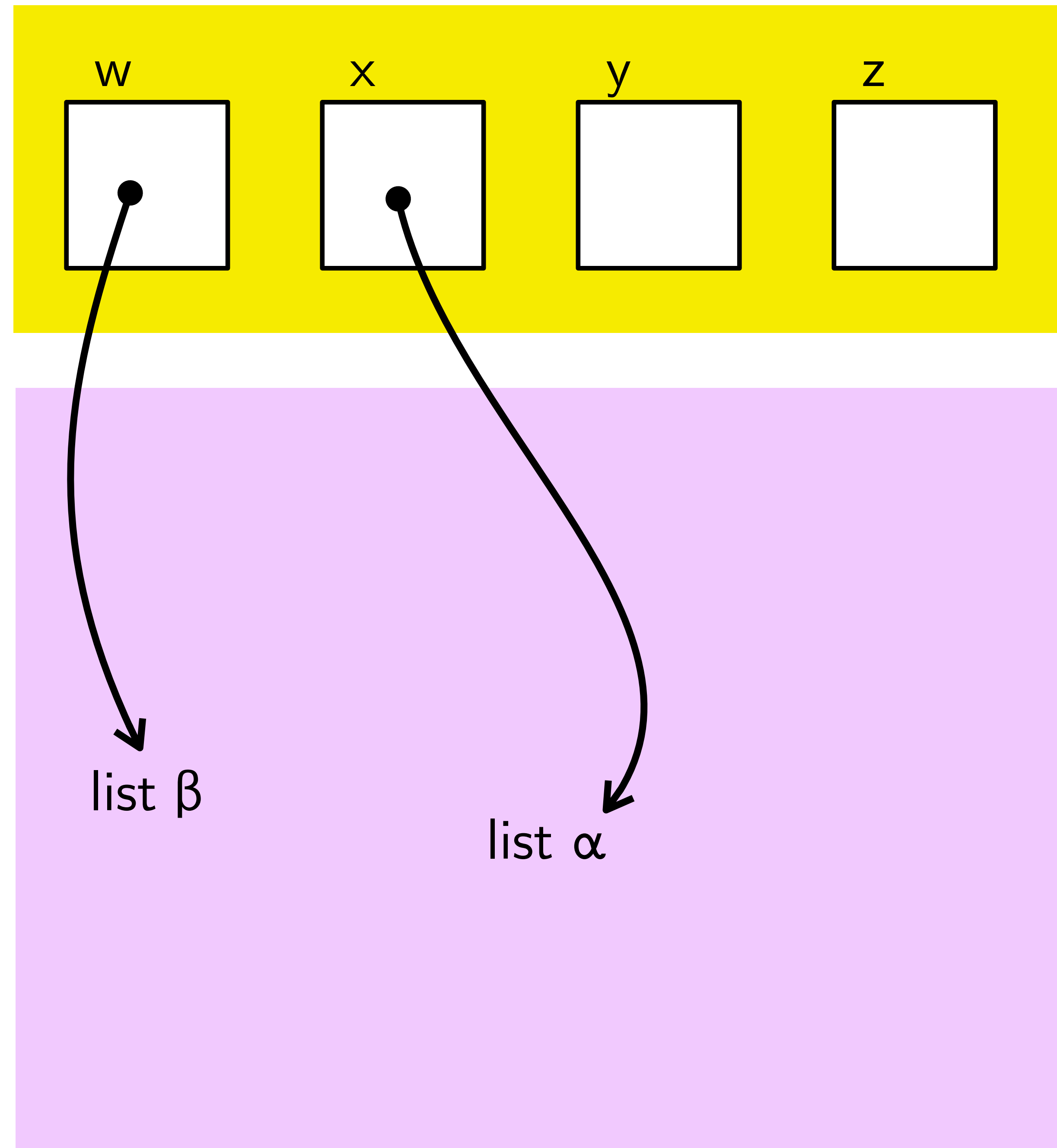
list δ x

```
w := 0;
```

$\exists \alpha,\beta.$ list α x $\wedge$ list β w $\wedge$ $\delta = -\beta \cdot \alpha$ $\wedge$
($\forall$z. reach(x,z) $\wedge$ reach(w,z) $\Rightarrow$ z=0)

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w



$\delta = -\beta \cdot \alpha$

Proof of list reverse

list δ x

listreverse(x,w)

list -δ w

Proof of list reverse

list δ x ∧ list ε y

listreverse(x,w)

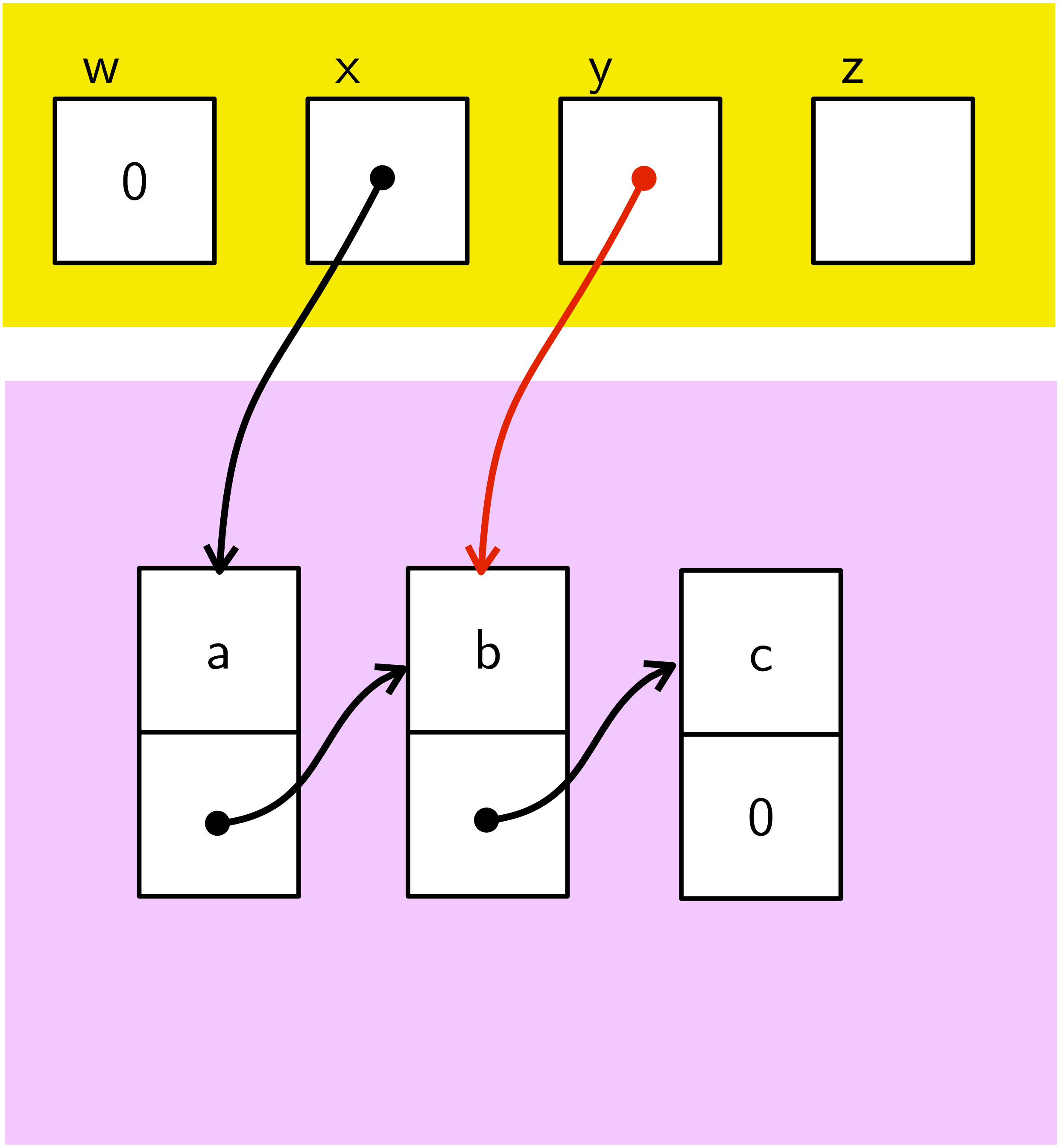
list -δ w

Proof of list reverse

list δ x ∧ list ε y

listreverse(x,w)

list -δ w

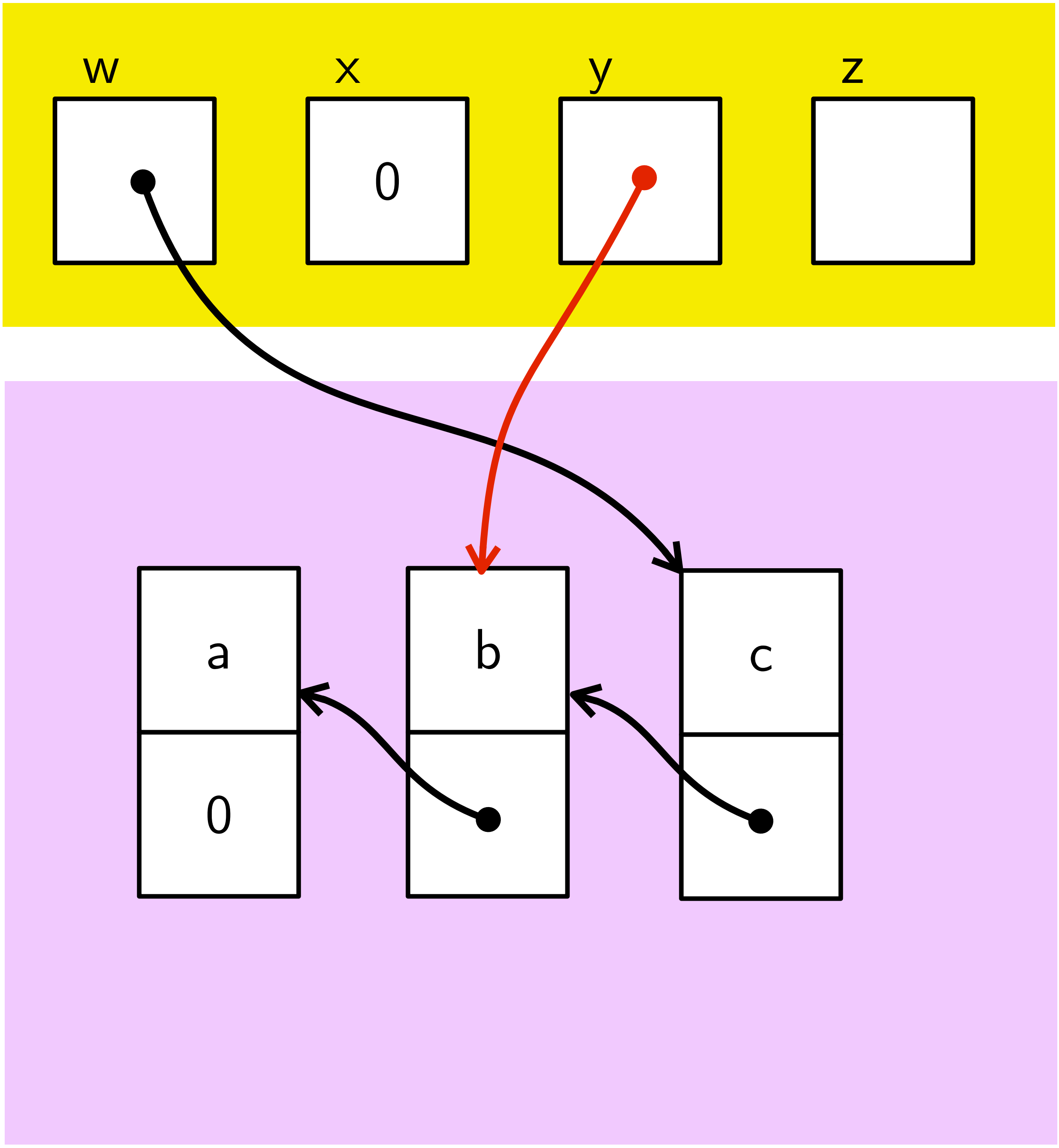


Proof of list reverse

list δ x ∧ list ε y

listreverse(x,w)

list -δ w



Proof of list reverse

list δ x $\wedge$ list ε y

$\wedge$ ($\forall$z. reach(x,z) $\wedge$ reach(y,z) $\Rightarrow$ z=0)

listreverse(x,w)

list -δ w

Proof of list reverse

$\text{list } \delta\ x \wedge \text{list } \varepsilon\ y$

$\wedge\ (\forall z.\ \text{reach}(x,z) \wedge \text{reach}(y,z) \Rightarrow z{=}0)$

```
w := 0;
```

$\exists \alpha,\beta.\ \text{list } \alpha\ x \wedge \text{list } \beta\ w \wedge \delta = \text{-}\beta{\cdot}\alpha\ \wedge$

$(\forall z.\ \text{reach}(x,z) \wedge \text{reach}(w,z) \Rightarrow z{=}0)$

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

$\text{list } \text{-}\delta\ w$

Proof of list reverse

$\text{list}\ \delta\ x \wedge \text{list}\ \varepsilon\ y$

$\wedge\ (\forall z.\ \text{reach}(x,z) \wedge \text{reach}(y,z) \Rightarrow z=0)$

```
w := 0;
```

$\exists \alpha,\beta.\ \text{list}\ \alpha\ x \wedge \text{list}\ \beta\ w \wedge \delta = -\beta\cdot\alpha\ \wedge$

$(\forall z.\ \text{reach}(x,z) \wedge \text{reach}(w,z) \Rightarrow z=0)$

$\wedge\ \text{list}\ \varepsilon\ y$

$\wedge\ (\forall z.\ (\text{reach}(x,z) \vee \text{reach}(w,z))$

$\quad \wedge\ \text{reach}(y,z) \Rightarrow z=0)$

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

$\text{list}\ -\delta\ w$

Proof of list reverse

$\text{list } \delta\ x \wedge \text{list } \varepsilon\ y$

$\wedge\ (\forall z.\ \text{reach}(x,z) \wedge \text{reach}(y,z) \Rightarrow z=0)$

```
w := 0;
```

$\exists \alpha,\beta.\ \text{list } \alpha\ x \wedge \text{list } \beta\ w \wedge \delta = -\beta\cdot\alpha\ \wedge$

$(\forall z.\ \text{reach}(x,z) \wedge \text{reach}(w,z) \Rightarrow z=0)$

$\wedge\ \text{list } \varepsilon\ y$

$\wedge\ (\forall z.\ (\text{reach}(x,z) \vee \text{reach}(w,z))$

$\quad \wedge\ \text{reach}(y,z) \Rightarrow z=0)$

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

$\text{list } -\delta\ w \wedge \text{list } \varepsilon\ y$

$\wedge\ (\forall z.\ \text{reach}(x,z) \wedge \text{reach}(y,z) \Rightarrow z=0)$

Proof of list reverse

list δ x $\wedge$ list ε y

$\wedge$ ($\forall$z. reach(x,z) $\wedge$ reach(y,z) $\Rightarrow$ z=0)

listreverse(x,w)

list -δ w $\wedge$ list ε y

$\wedge$ ($\forall$z. reach(x,z) $\wedge$ reach(y,z) $\Rightarrow$ z=0)

Outline

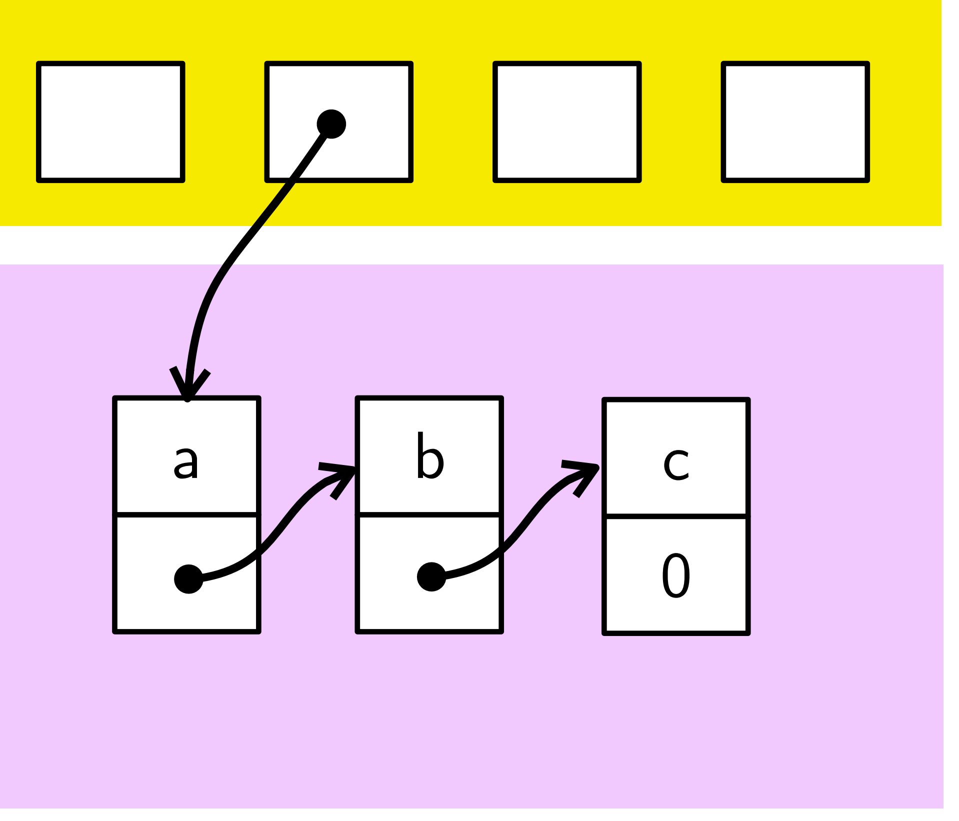
- A "VeriFast" introduction to Hoare logic
- List reversal in Hoare logic
- **List reversal in separation logic**
- Proof rules for separation logic
- Program variables as resource

The need for separation

The 'list' predicate:

$$\text{list } [\,]\ x \stackrel{\text{def}}{=} (x = 0)$$

$$\text{list } (a{::}\alpha)\ x \stackrel{\text{def}}{=} (\exists y.\ [x] = a \land [x{+}1] = y \land \text{list } \alpha\ y)$$



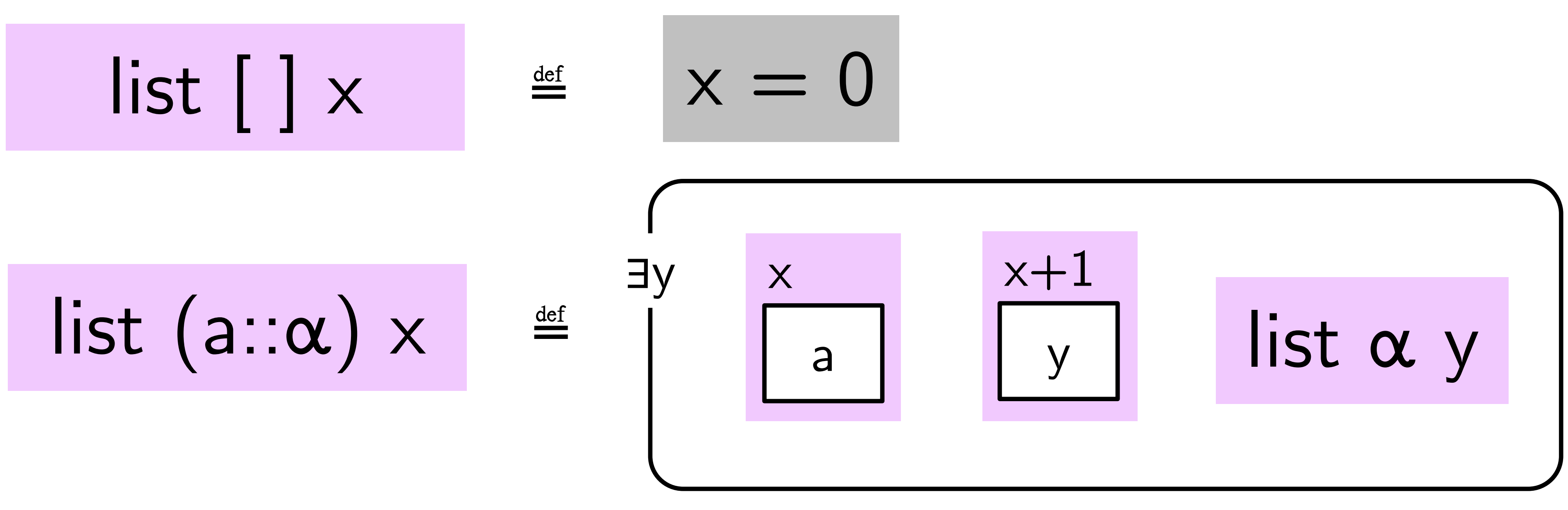
The need for separation

The ‘list’ predicate:

$$\text{list}\ [\,]\ x \stackrel{\text{def}}{=} (x = 0)$$

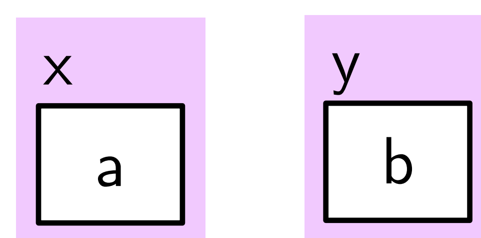
$$\text{list}\ (a{::}\alpha)\ x \stackrel{\text{def}}{=} (\exists y.\ [x] = a \wedge [x+1] = y \wedge \text{list}\ \alpha\ y)$$

In separation logic:



The need for separation

$$[x] = a \;\wedge\; [y] = b$$



The need for separation

list δ x

```
w := 0;
```

∃α,β. list α x ∧ list β w ∧ δ = -β·α ∧
(∀z. reach(x,z) ∧ reach(w,z) ⇒ z=0)

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list -δ w

The need for separation

list δ x

```
w := 0;
```

$\exists\alpha$ $\exists\beta$ list α x list β w $\delta = -\beta\cdot\alpha$

```
while (x≠0) do {
    z := [x+1];
    [x+1] := w;
    w := x;
    x := z;
}
```

list $-\delta$ w

The need for separation

list δ x

listreverse(x,w)

list -δ w

The need for separation

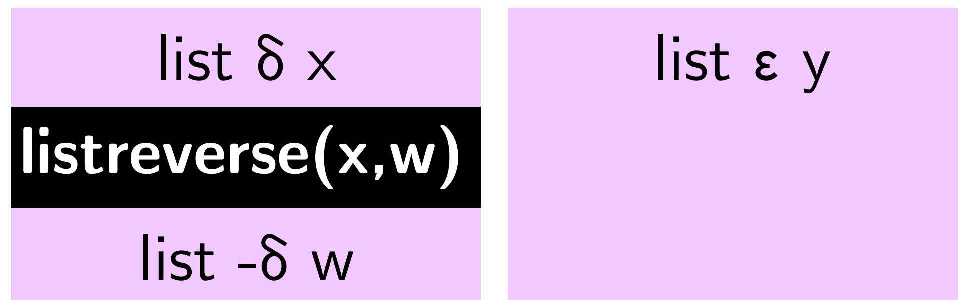
list δ x

listreverse(x,w)

list -δ w

list ε y

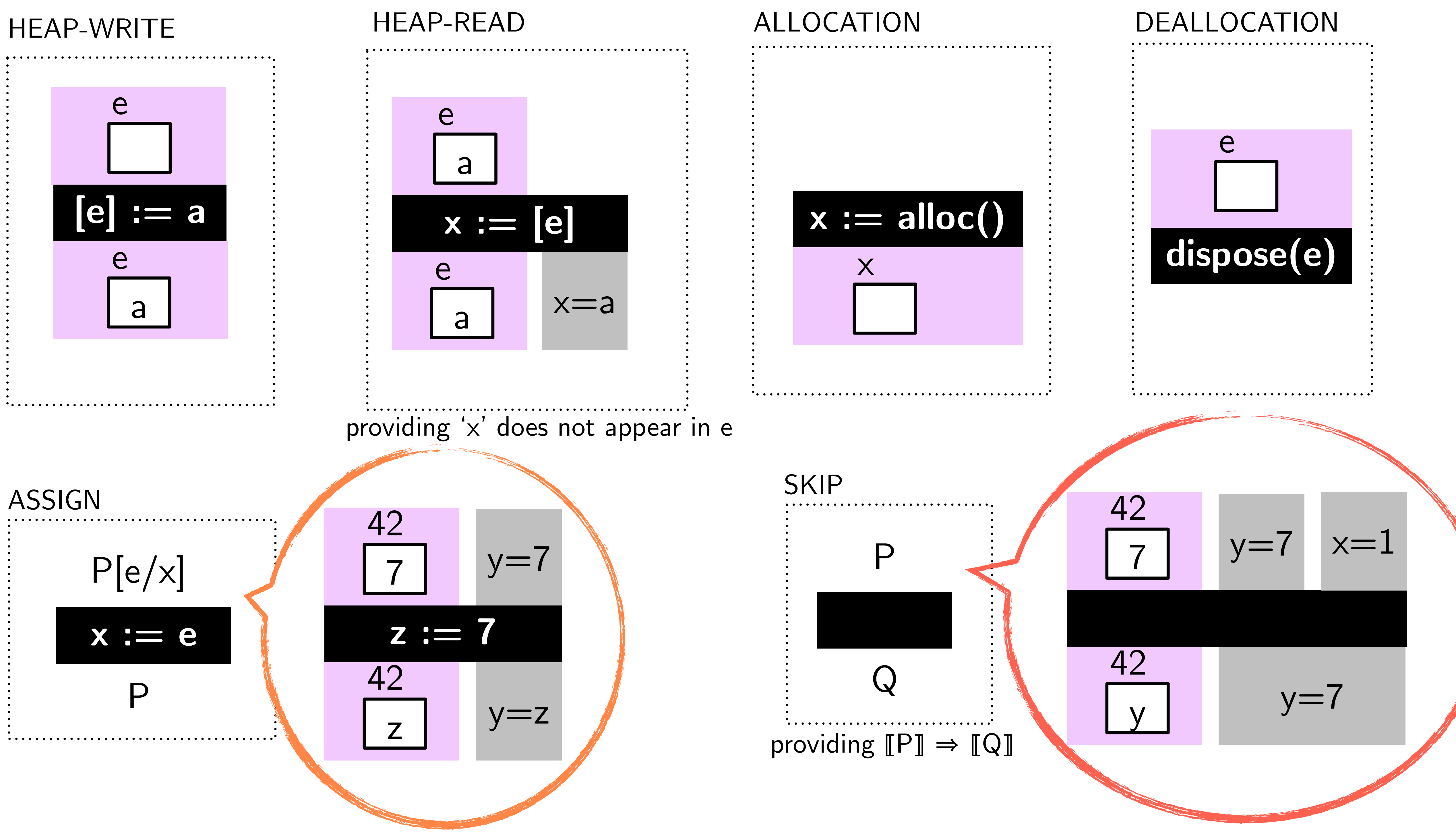
The need for separation



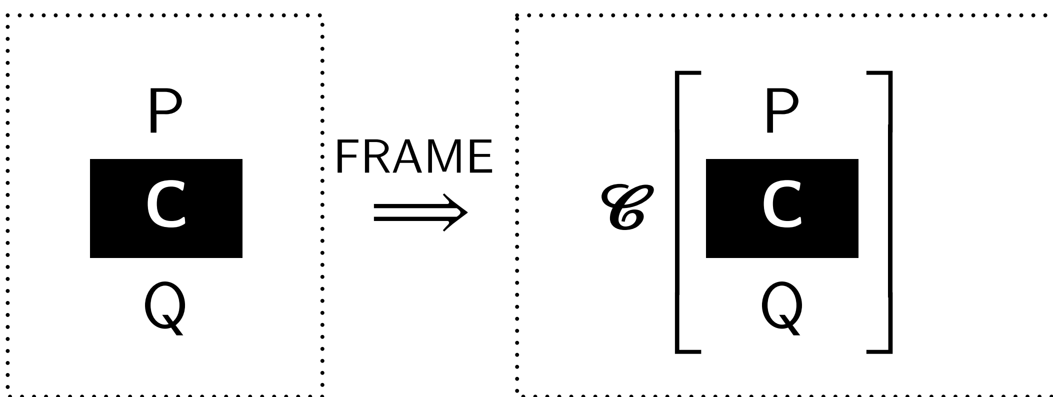
Outline

- A “VeriFast” introduction to Hoare logic
- List reversal in Hoare logic
- List reversal in separation logic
- **Proof rules for separation logic**
- Program variables as resource

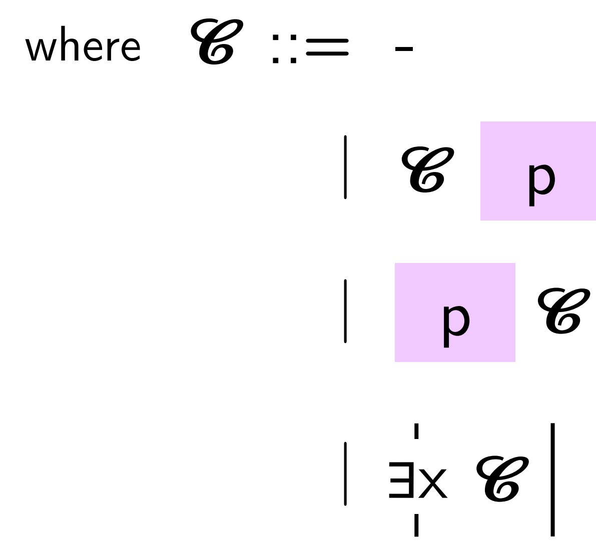
Proof rules for separation logic



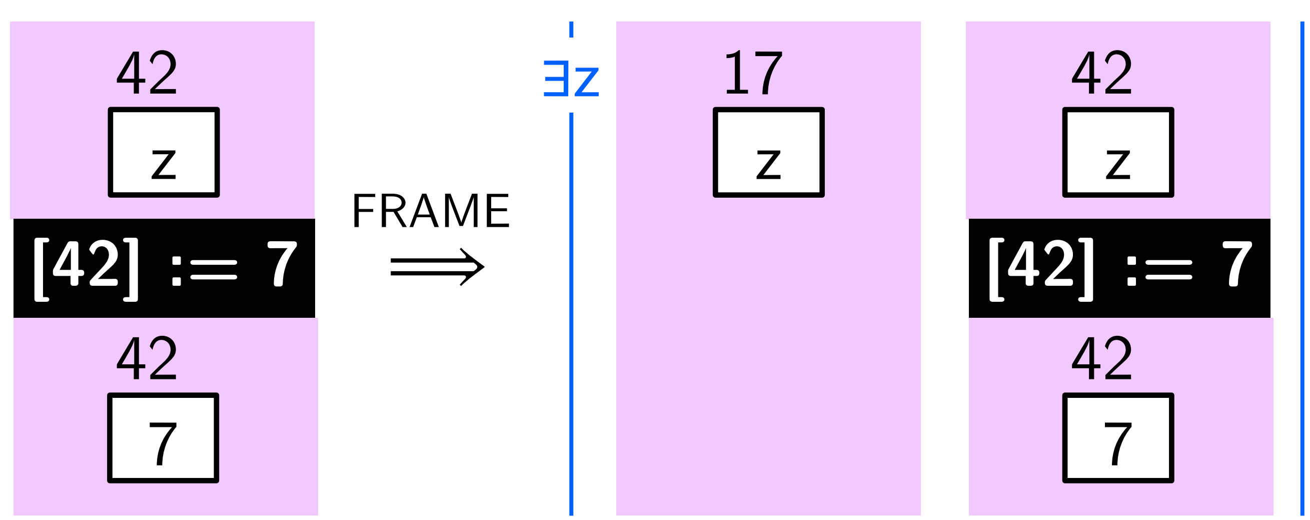
Proof rules for separation logic



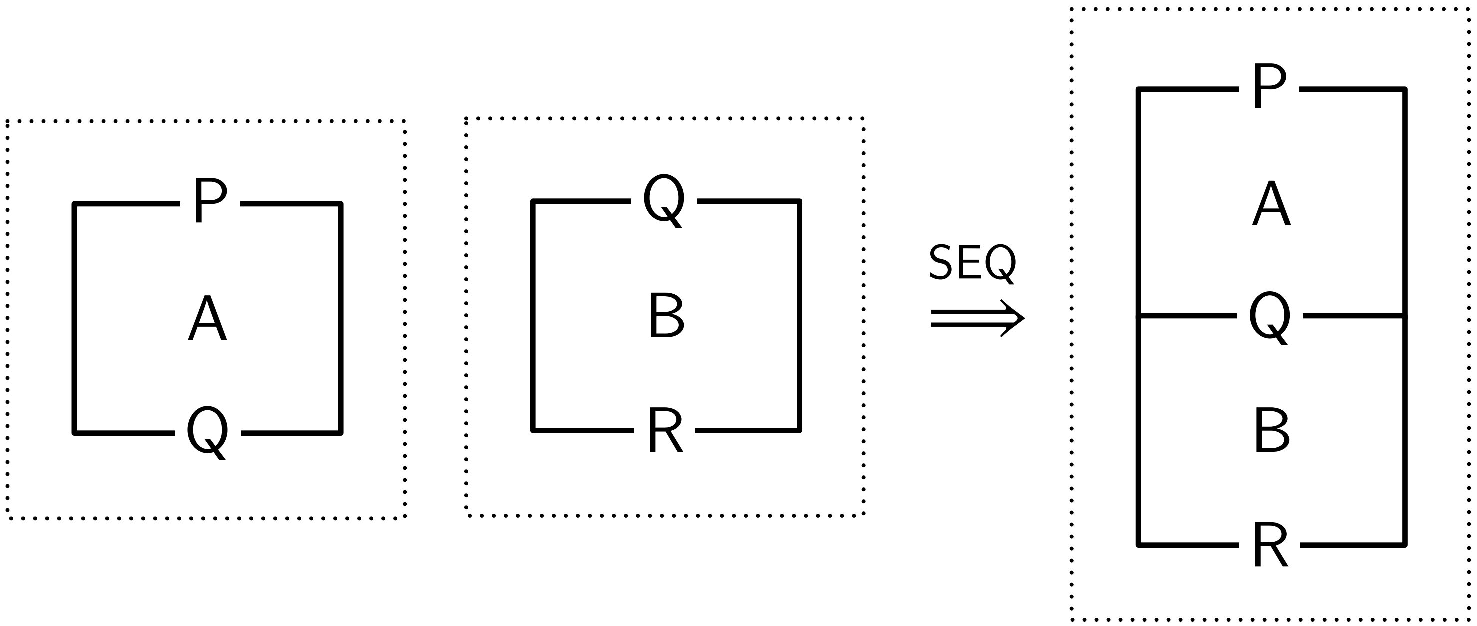
provided $wr(C) \cap rd(\mathscr{C}) = \varnothing$



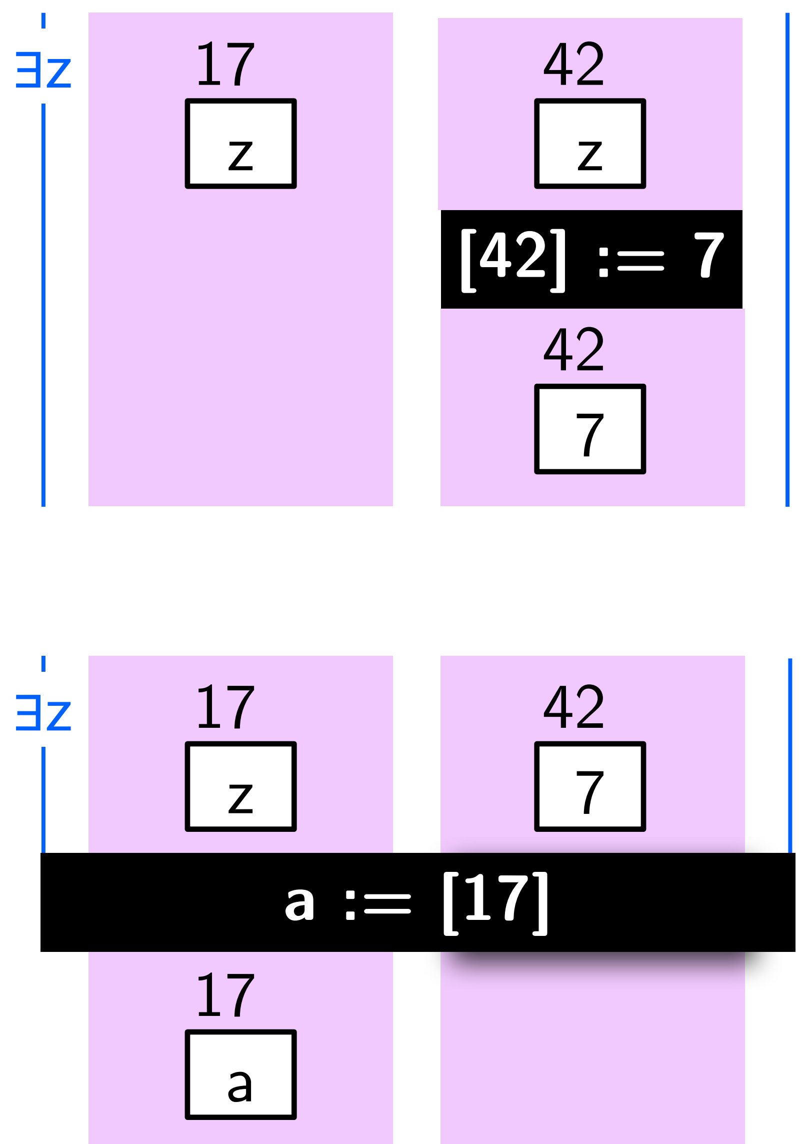
Example:



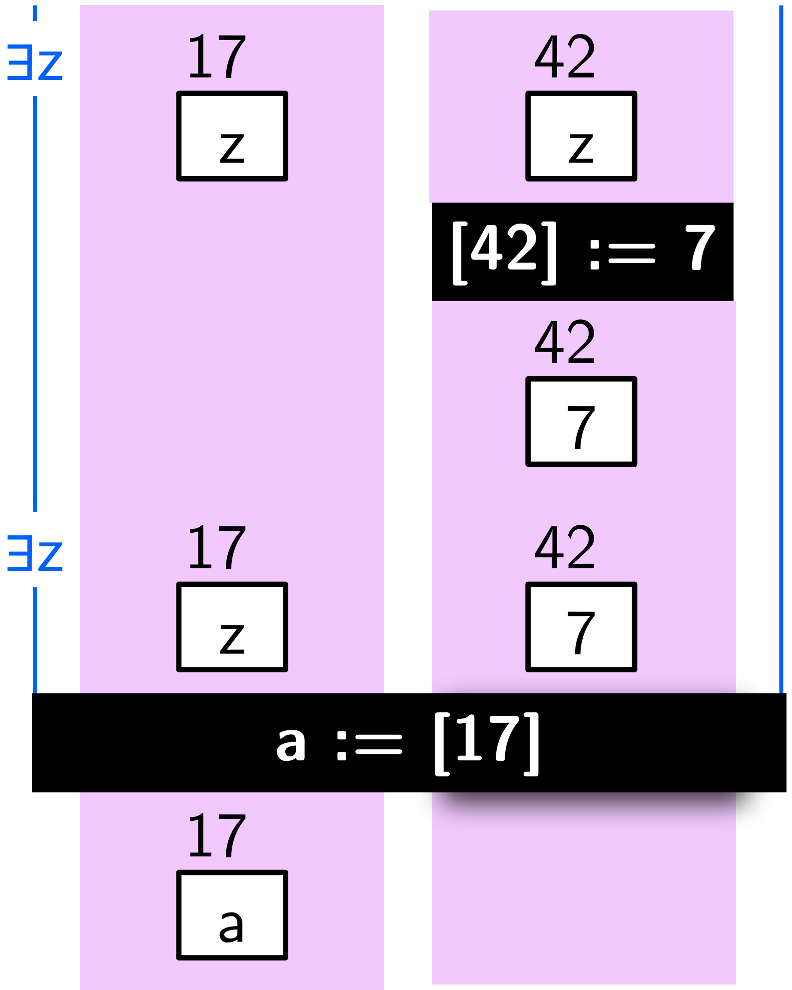
Proof rules for separation logic



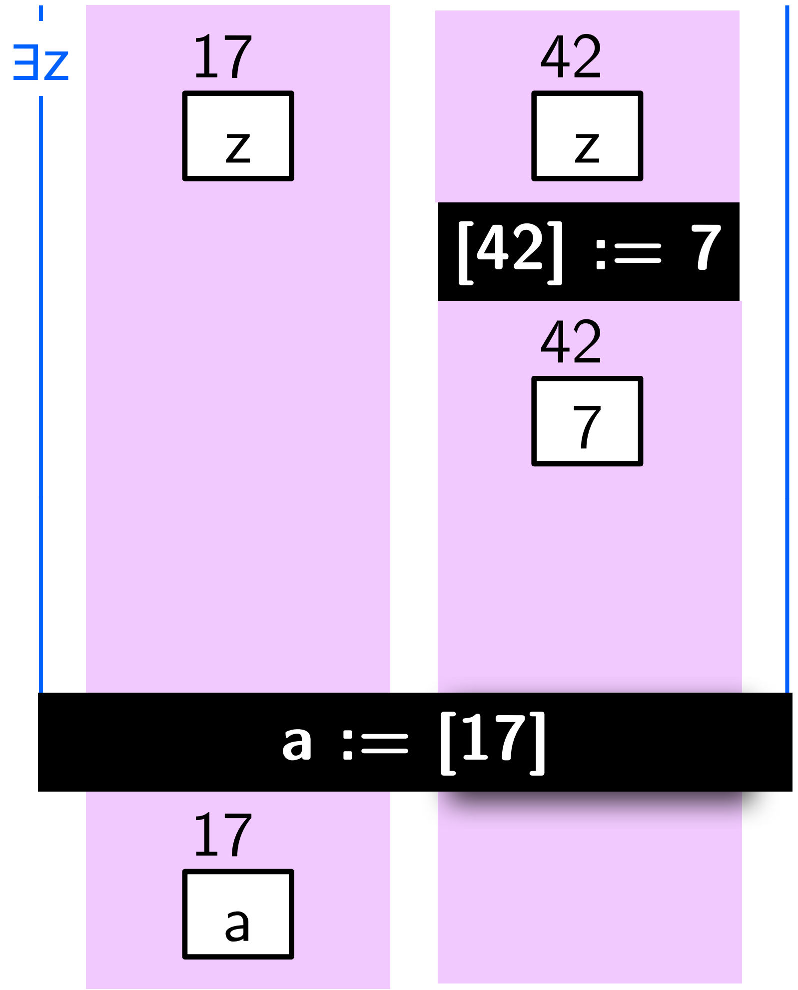
Proof rules for separation logic



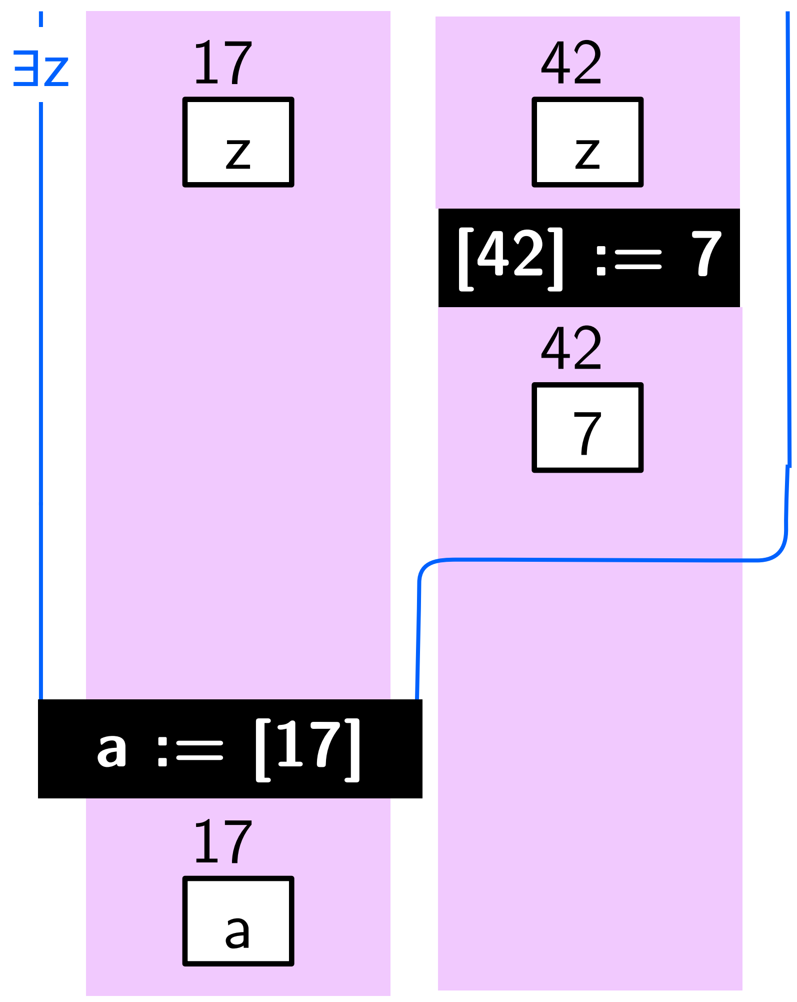
Proof rules for separation logic



Proof rules for separation logic



Proof rules for separation logic



Three assignments

$[x]{=}0 \wedge [y]{=}0 \wedge [z]{=}0$

[x] := 1

$[x]{=}1 \wedge [y]{=}0 \wedge [z]{=}0$

[y] := 1

$[x]{=}1 \wedge [y]{=}1 \wedge [z]{=}0$

[z] := 1

$[x]{=}1 \wedge [y]{=}1 \wedge [z]{=}1$

Three assignments

$[x]=0 \wedge [y]=0 \wedge [z]=0 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

[x] := 1

$[x]=1 \wedge [y]=0 \wedge [z]=0 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

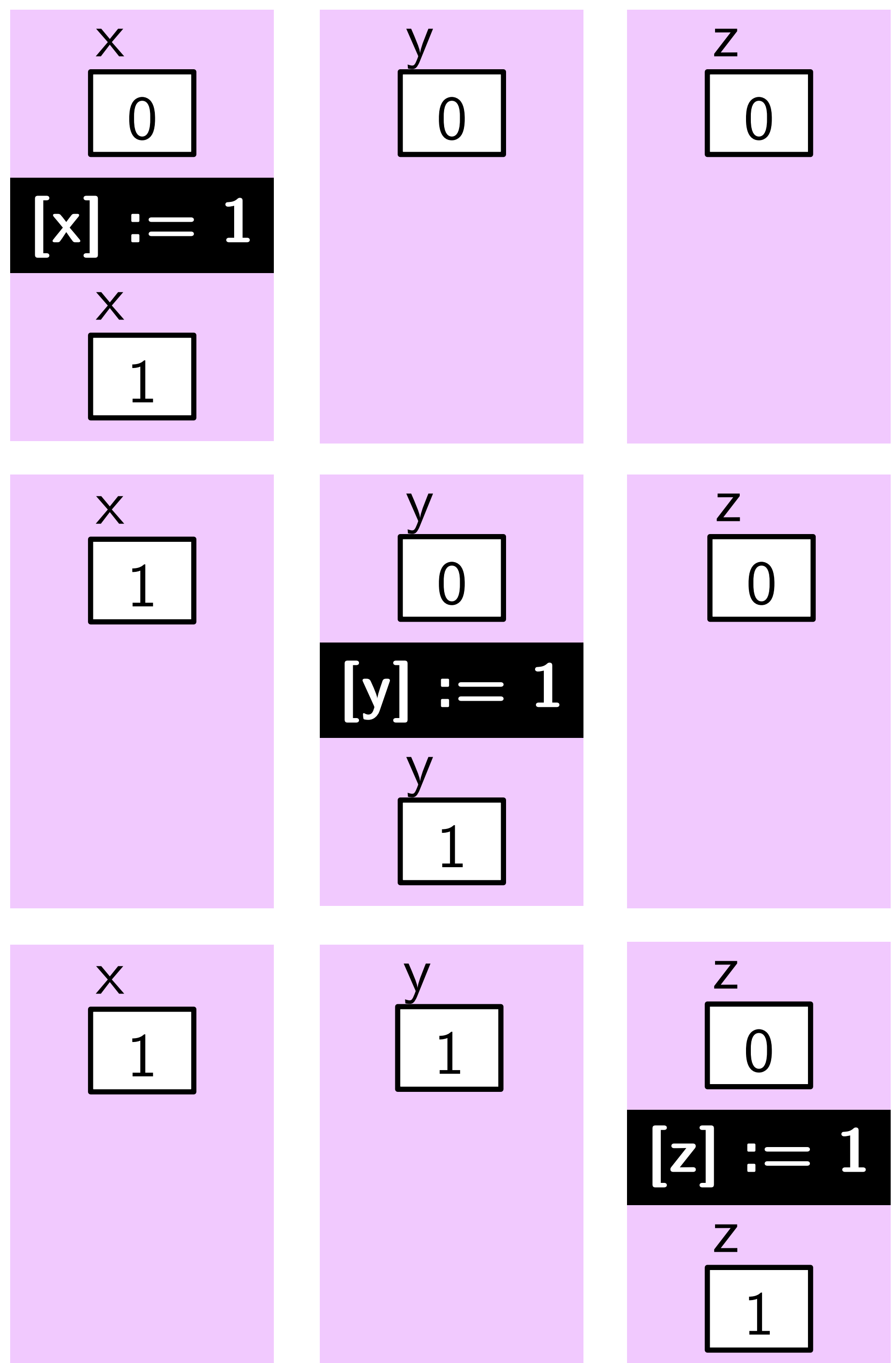
[y] := 1

$[x]=1 \wedge [y]=1 \wedge [z]=0 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

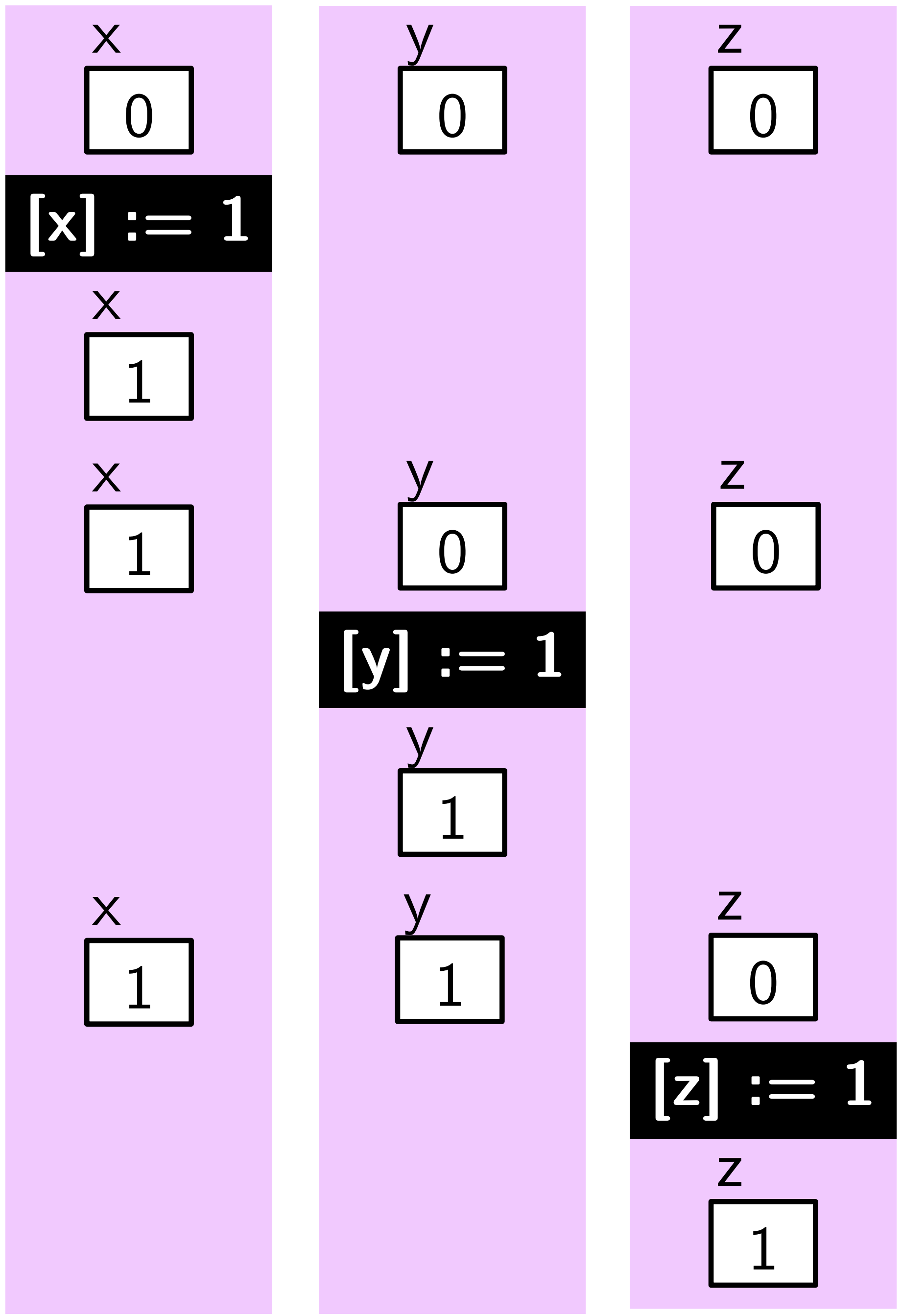
[z] := 1

$[x]=1 \wedge [y]=1 \wedge [z]=1 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

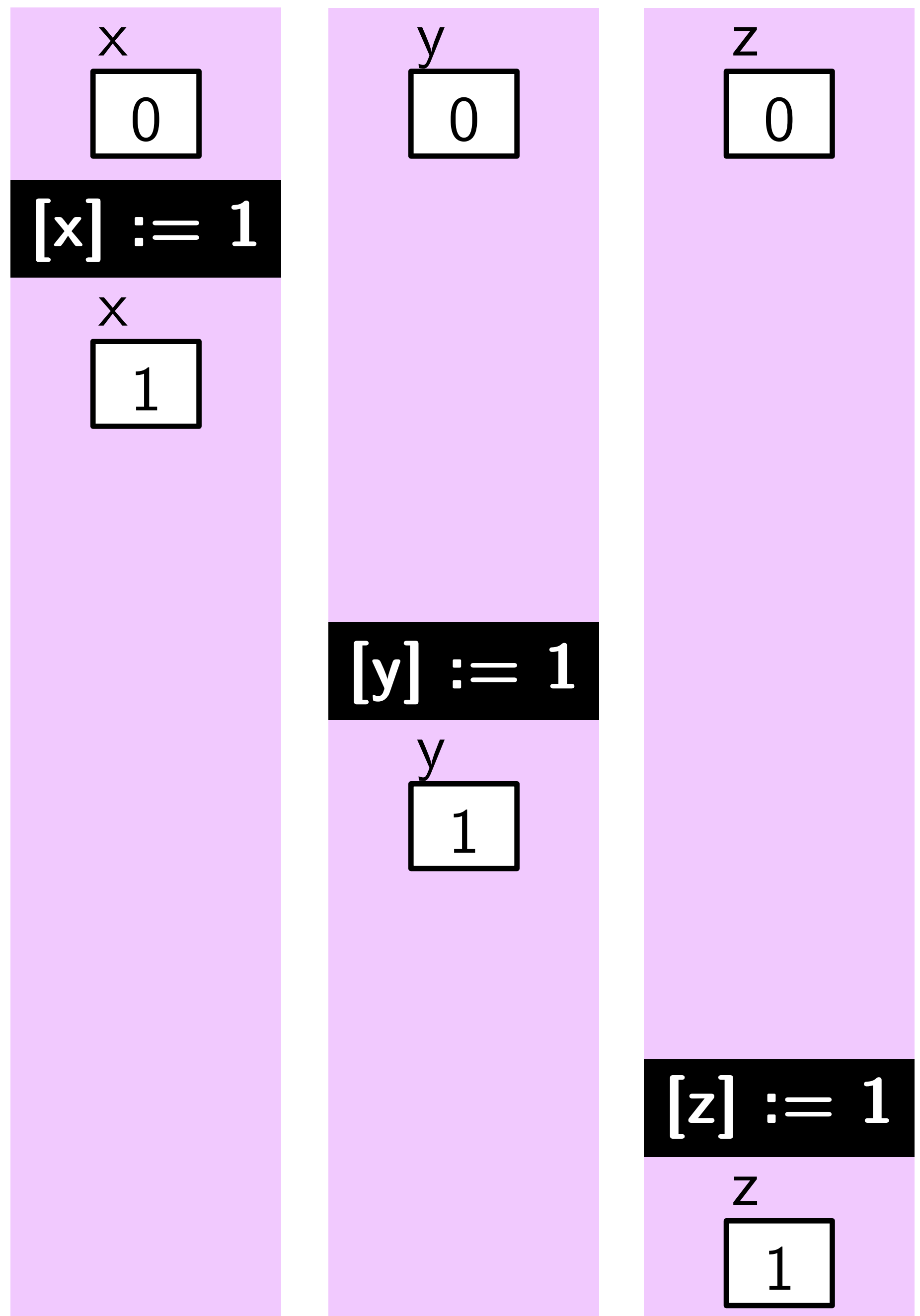
Three assignments



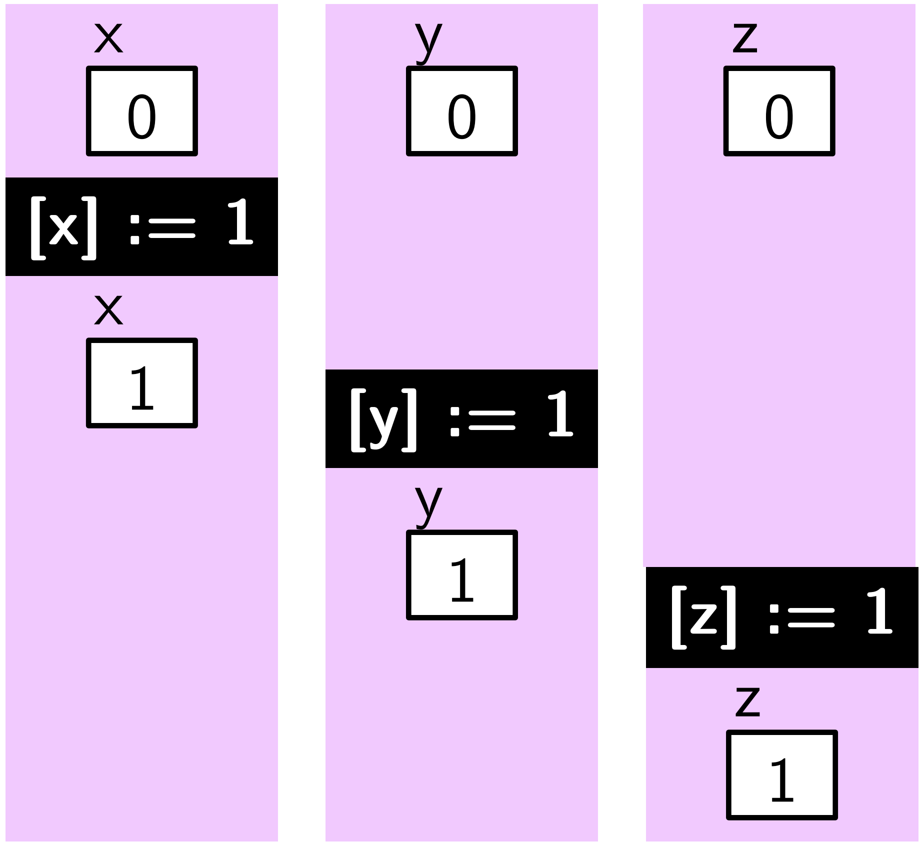
Three assignments



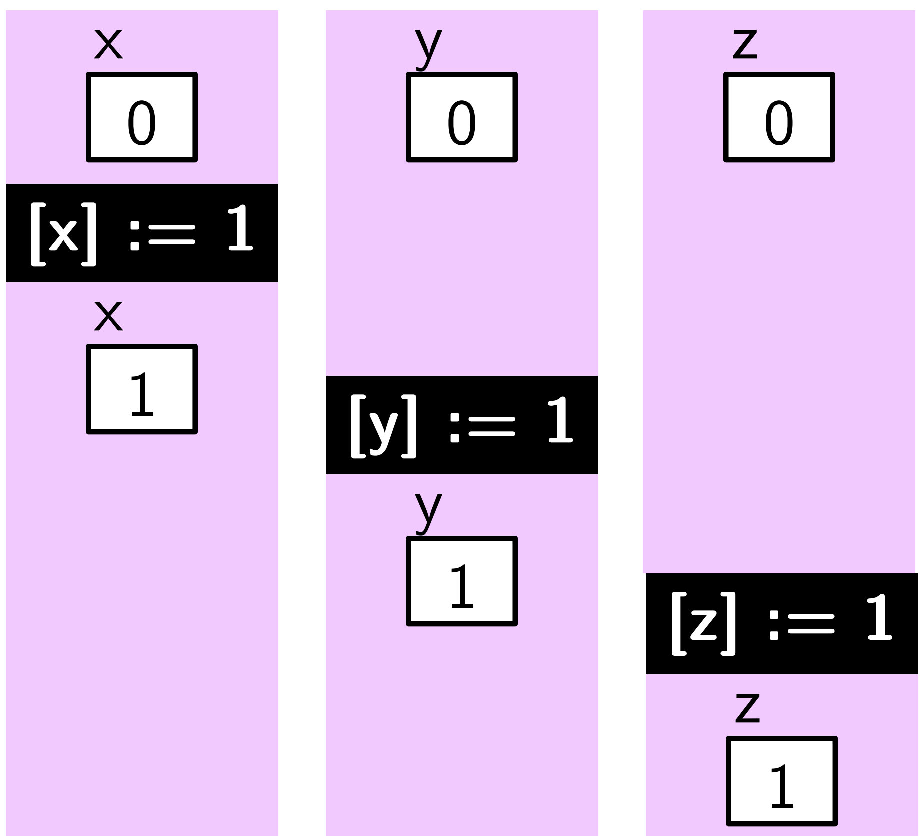
Three assignments



Three assignments



Three assignments



$[x]=0 \wedge [y]=0 \wedge [z]=0 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

[x] := 1

$[x]=1 \wedge [y]=0 \wedge [z]=0 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

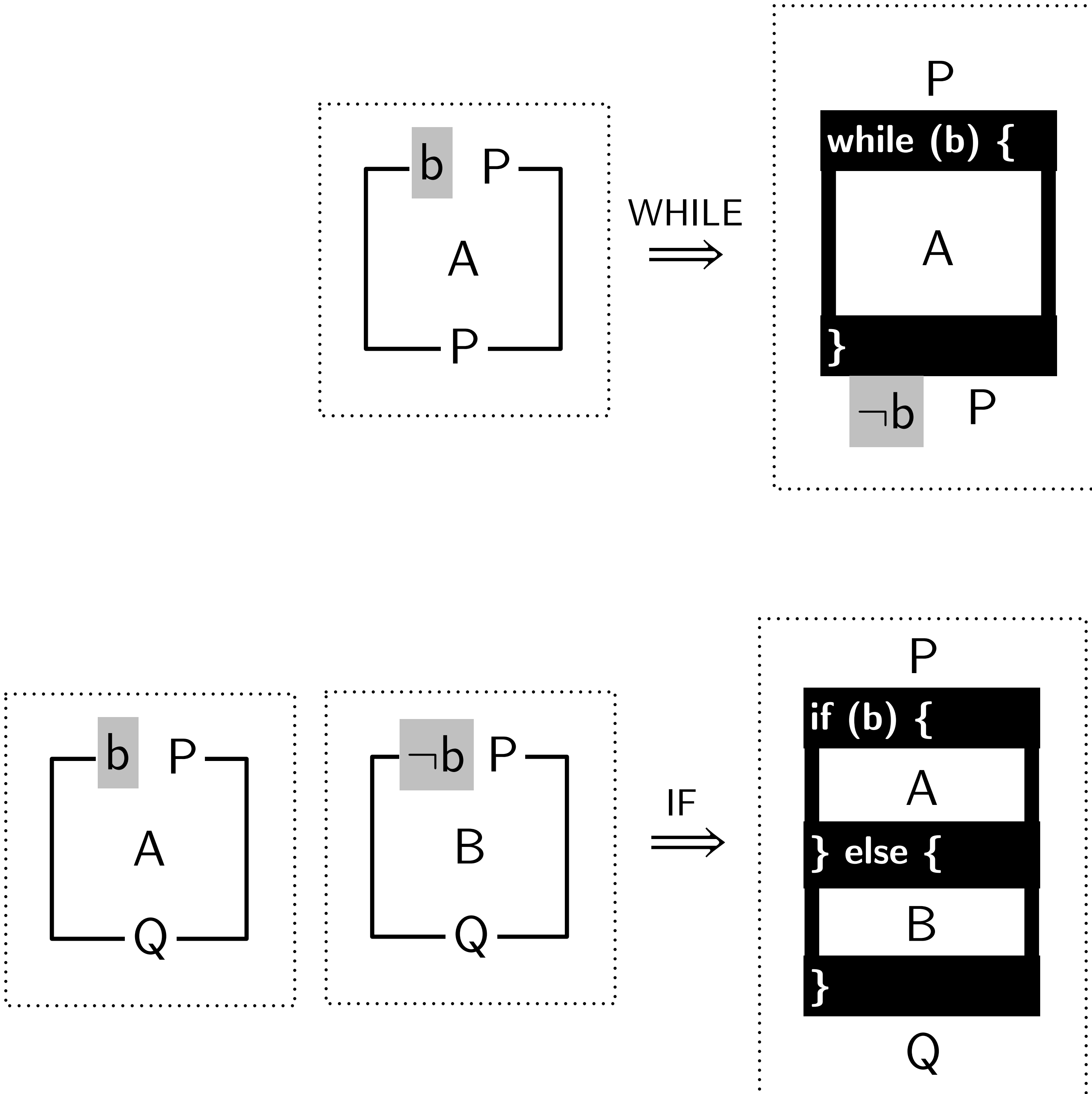
[y] := 1

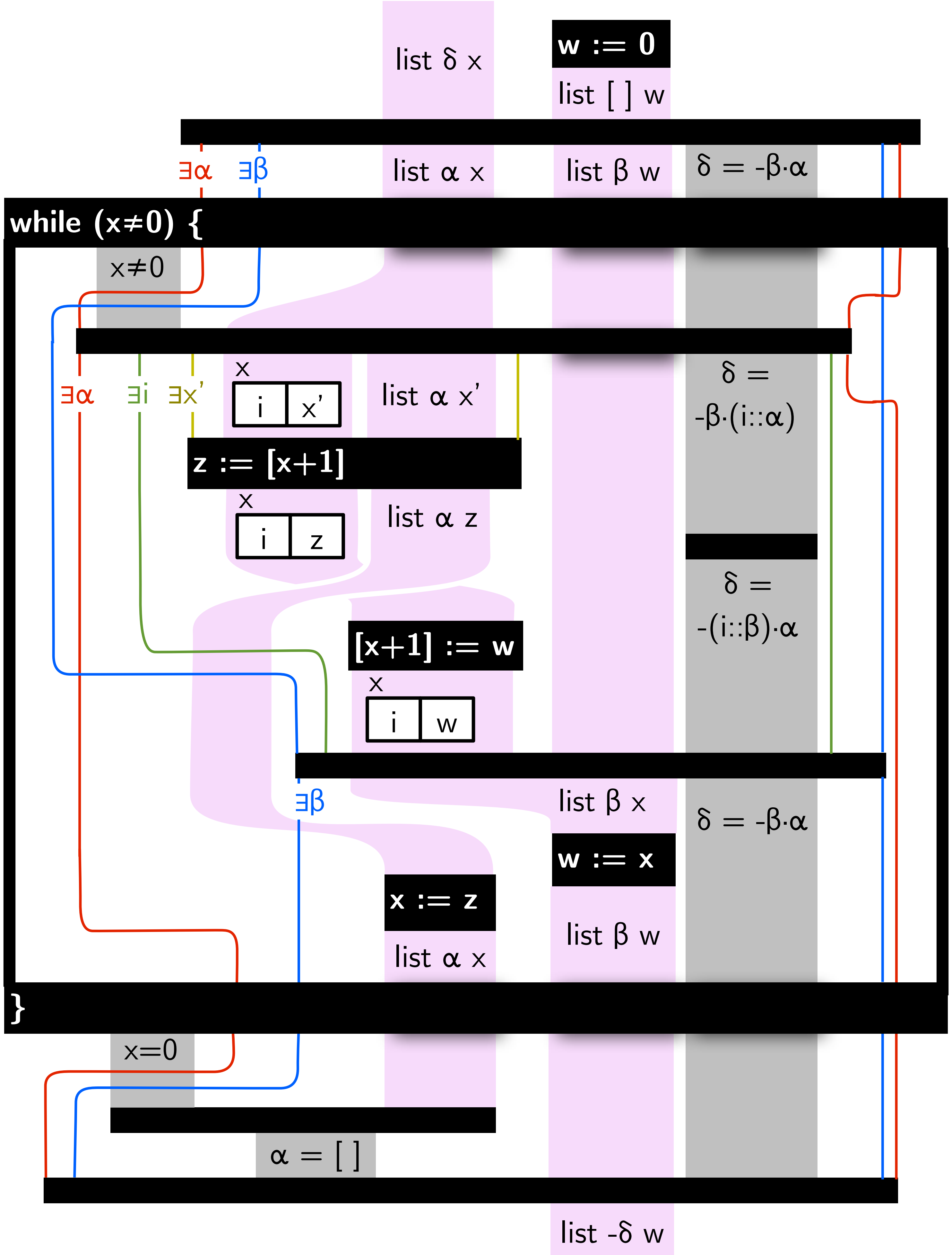
$[x]=1 \wedge [y]=1 \wedge [z]=0 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

[z] := 1

$[x]=1 \wedge [y]=1 \wedge [z]=1 \wedge x \neq y \wedge y \neq z \wedge x \neq z$

Proof rules for separation logic





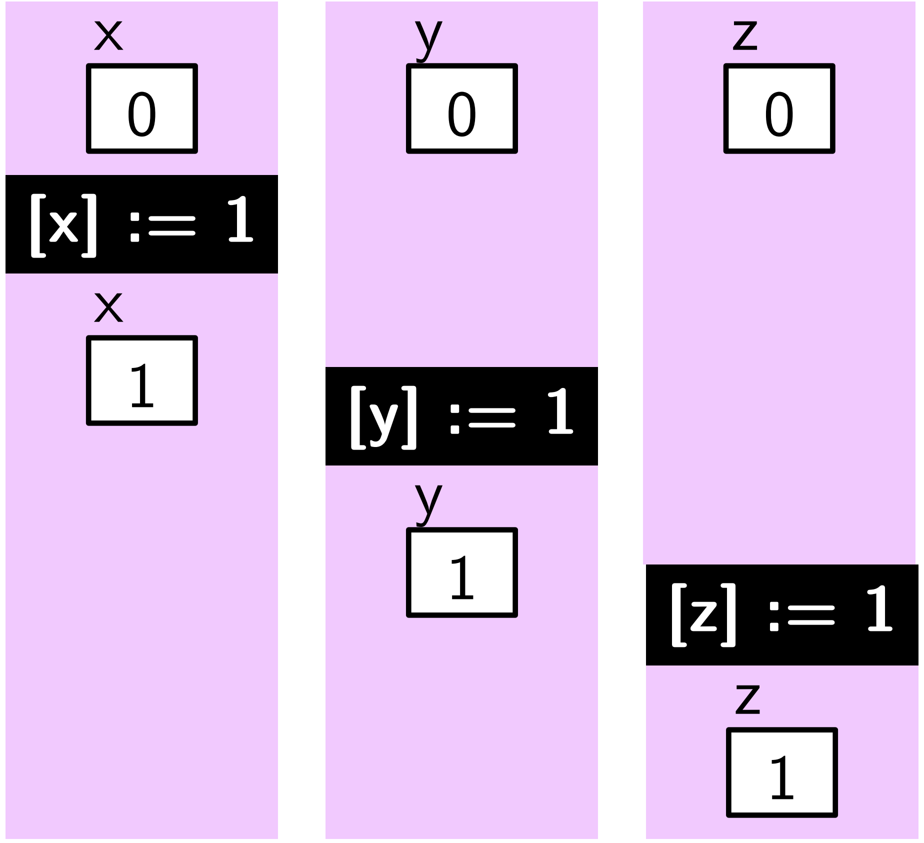
list δ x
w := 0
list [ ] w
∃α
∃β
list α x
list β w
δ = -β·α
while (x≠0) {
x≠0
∃α
∃i
∃x'
x
i
x'
list α x'
δ =
-β·(i::α)
z := [x+1]
x
i
z
list α z
δ =
-(i::β)·α
[x+1] := w
x
i
w
∃β
list β x
δ = -β·α
w := x
x := z
list β w
list α x
}
x=0
α = [ ]
list -δ w

Demo: List reverse in VeriFast

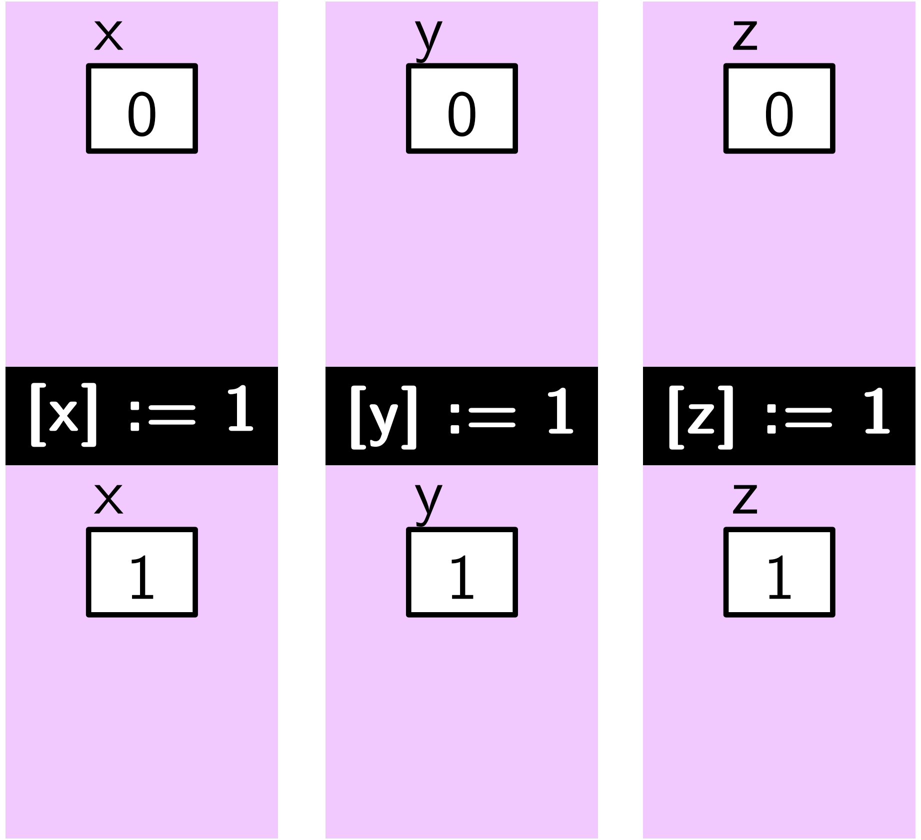
Outline

- A “VeriFast” introduction to Hoare logic
- List reversal in Hoare logic
- List reversal in separation logic
- Proof rules for separation logic
- **Program variables as resource**

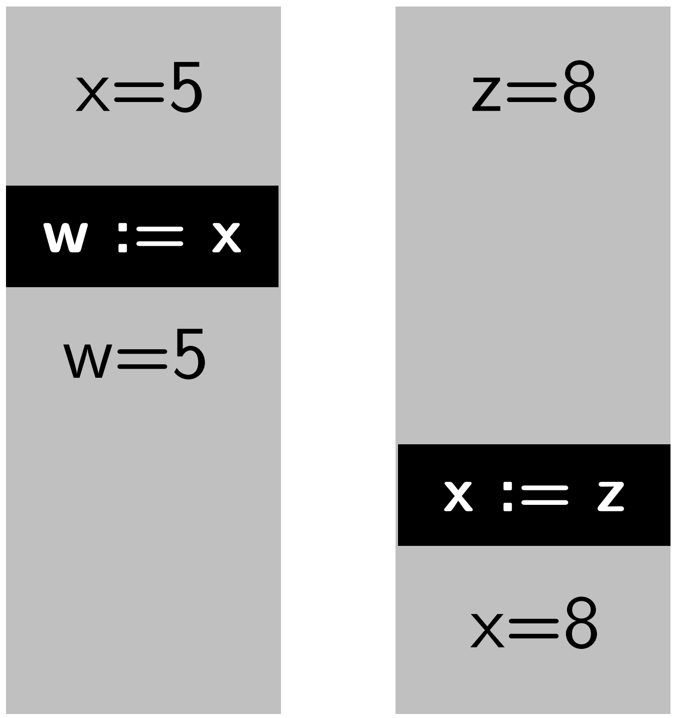
Three assignments



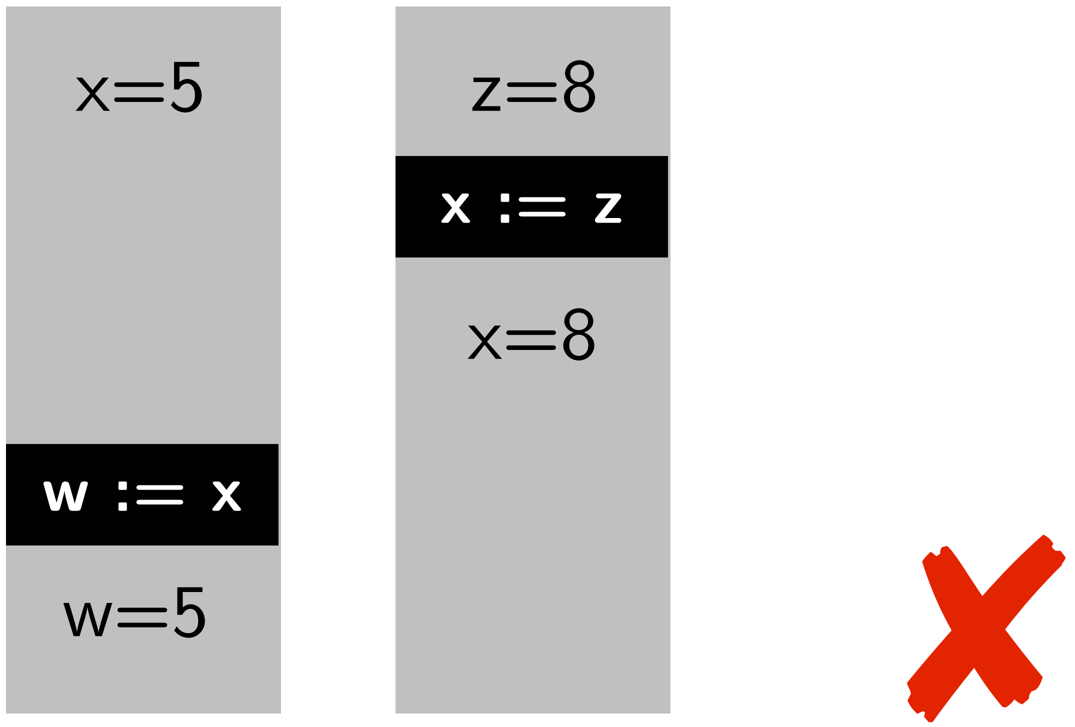
Three assignments



Variables as resource

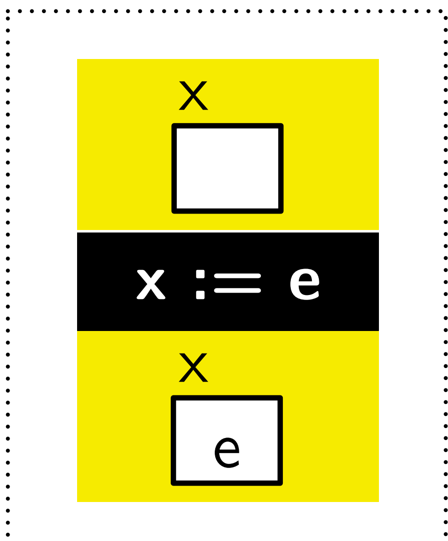


Variables as resource

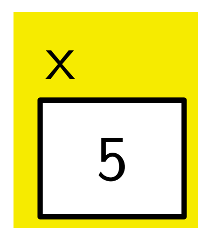


Variables as resource

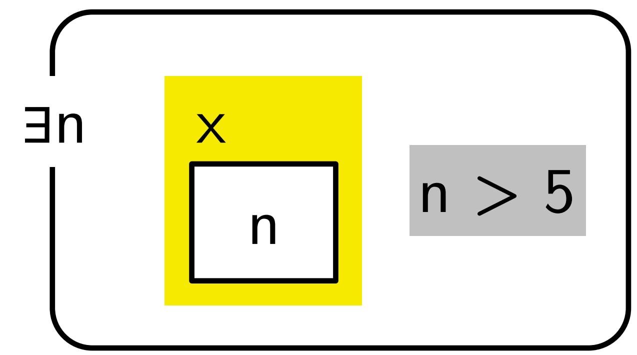
ASSIGN



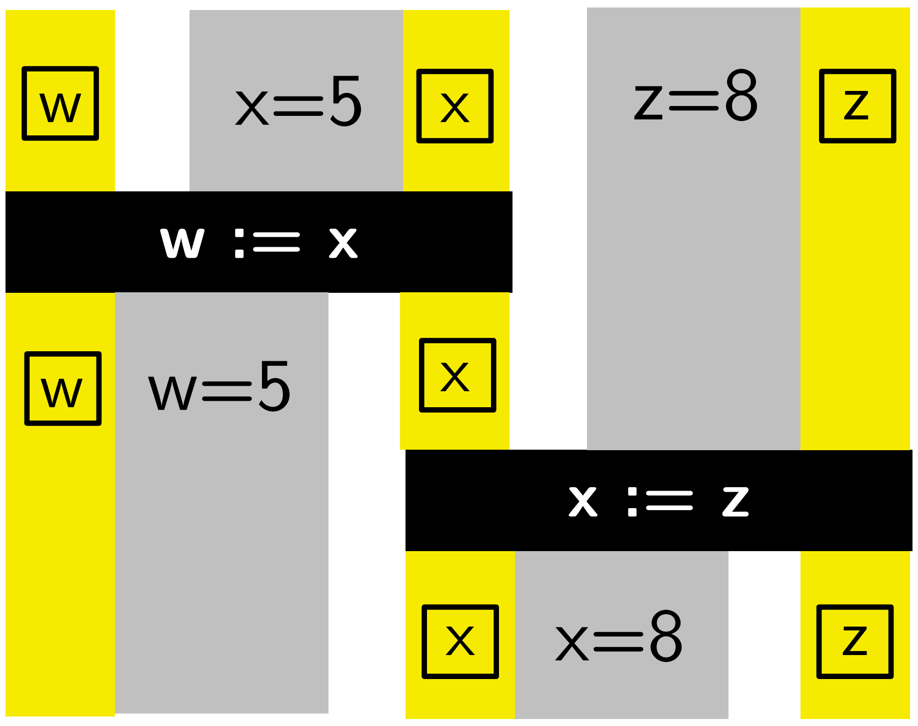
$x = 5$



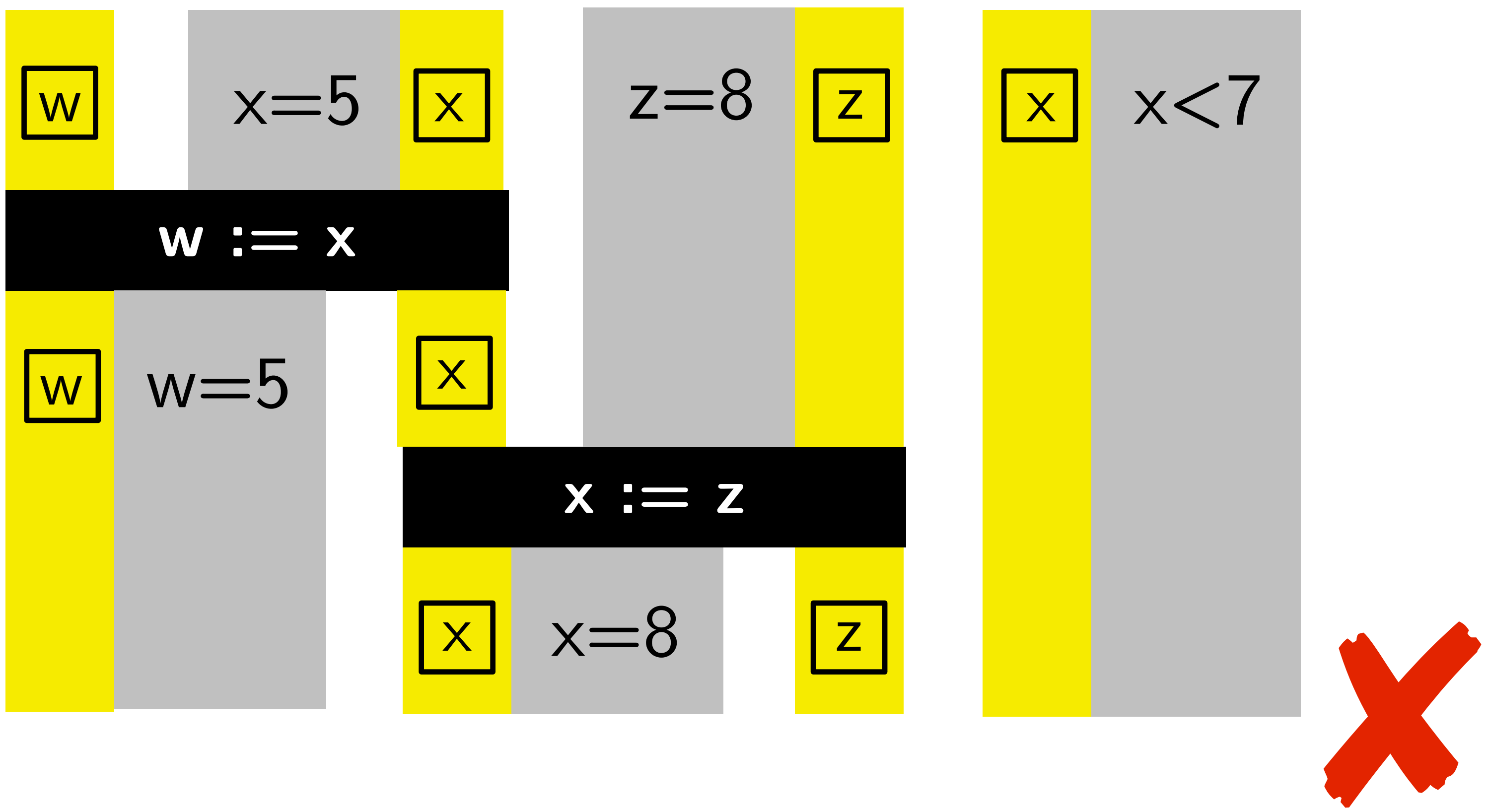
$x > 5$



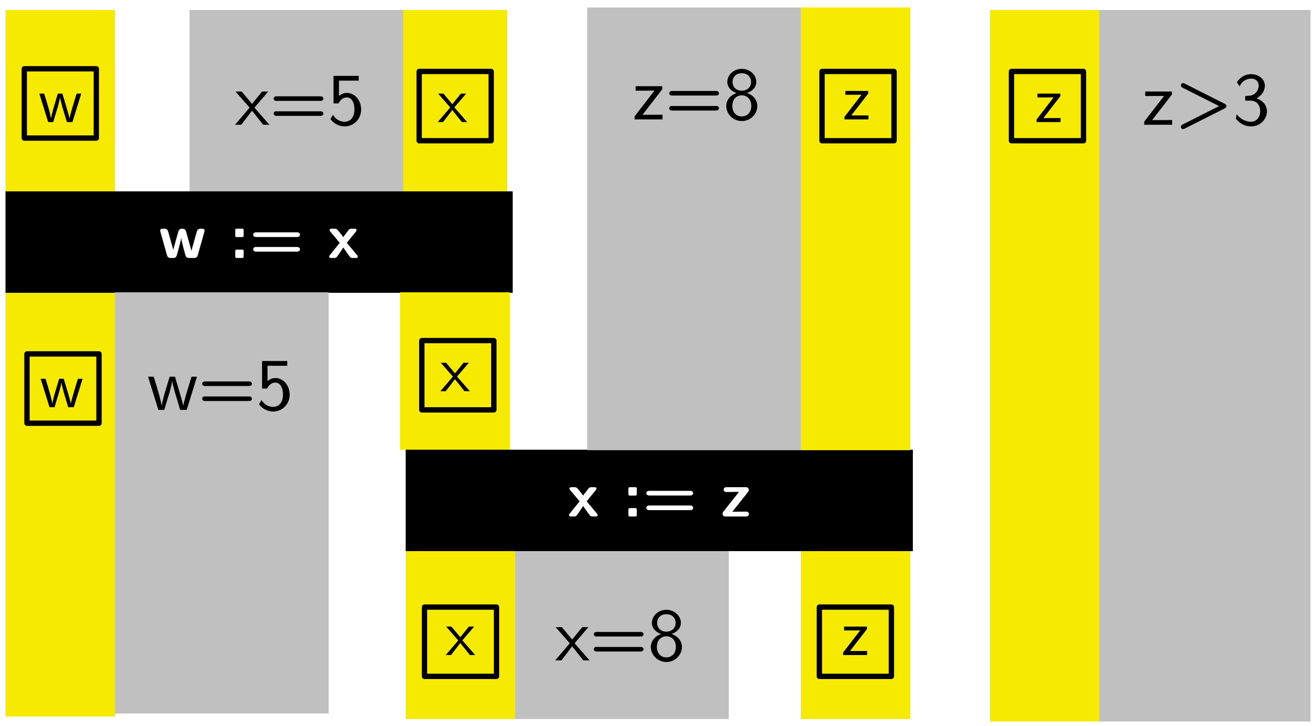
Variables as resource



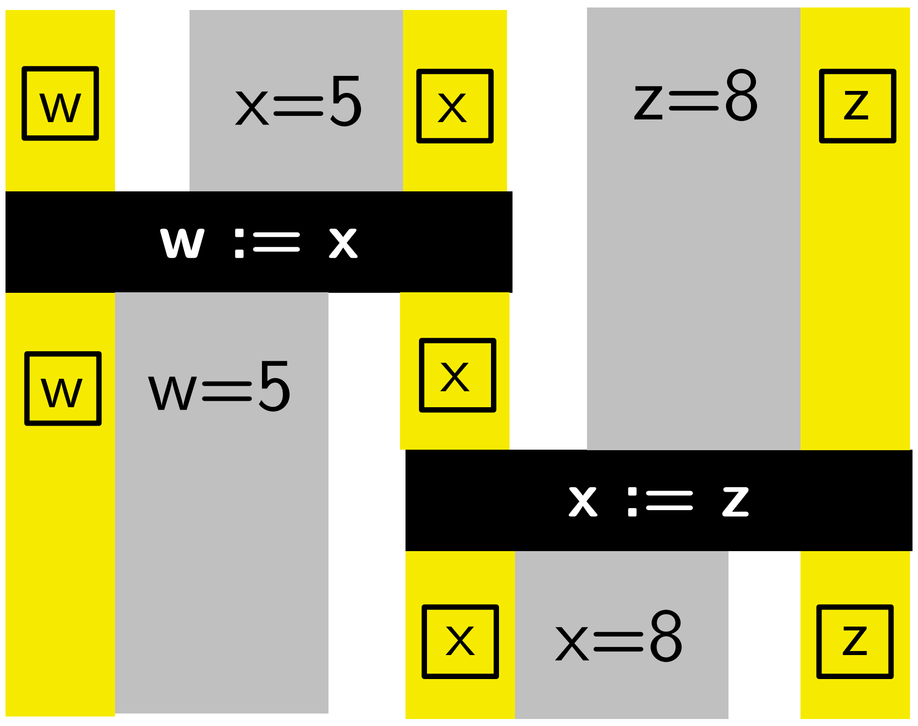
Variables as resource



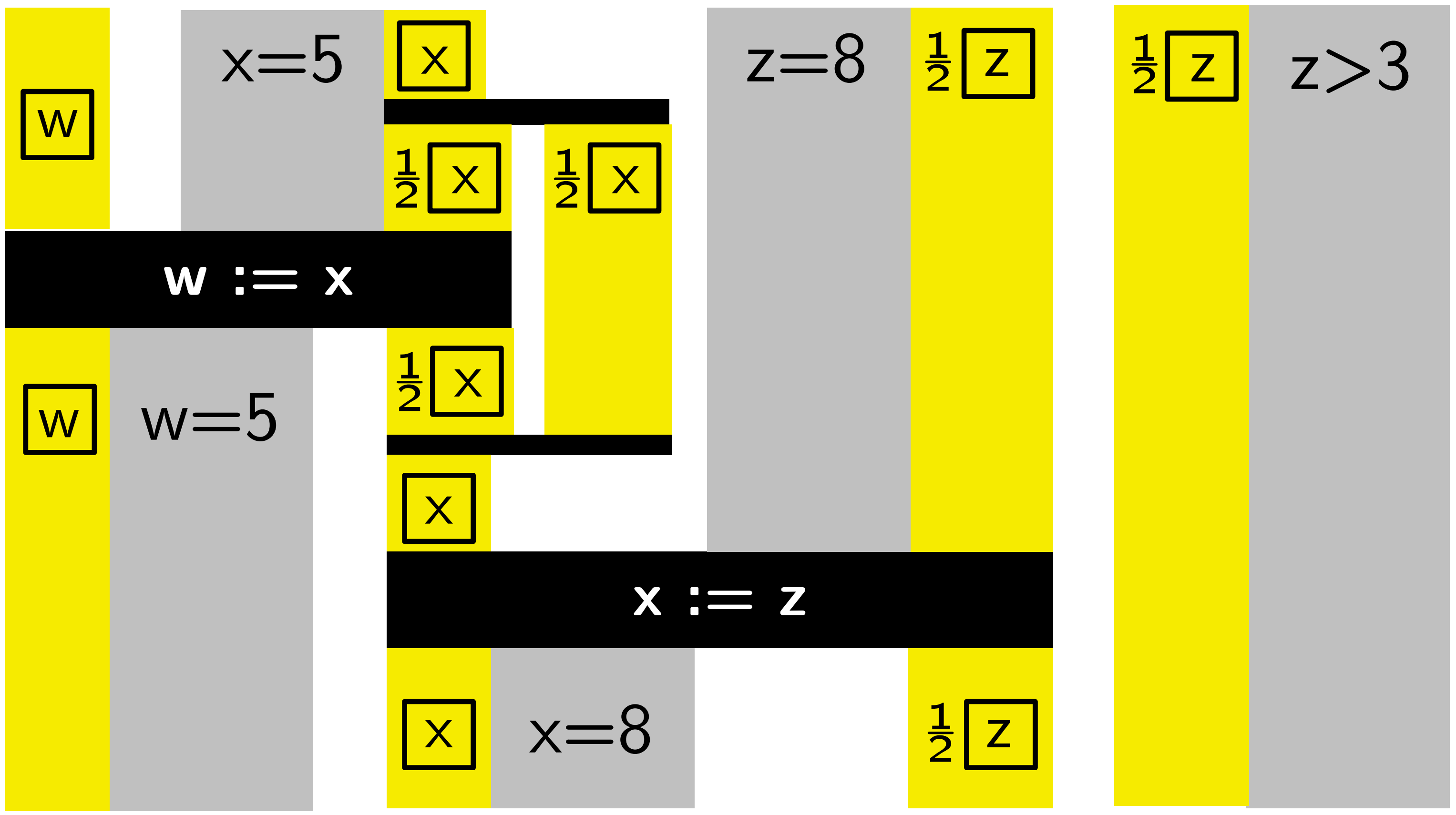
Variables as resource



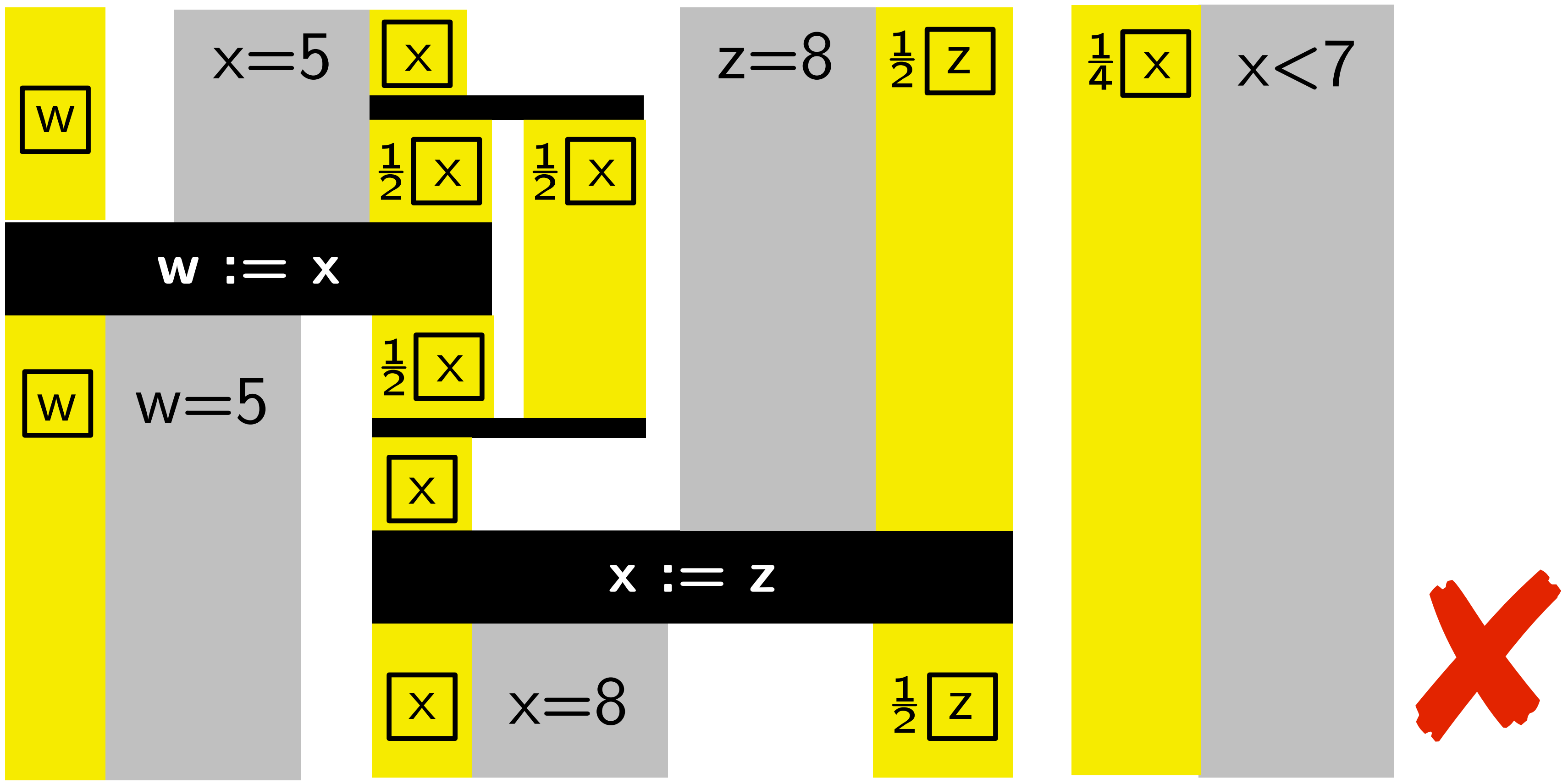
Variables as resource

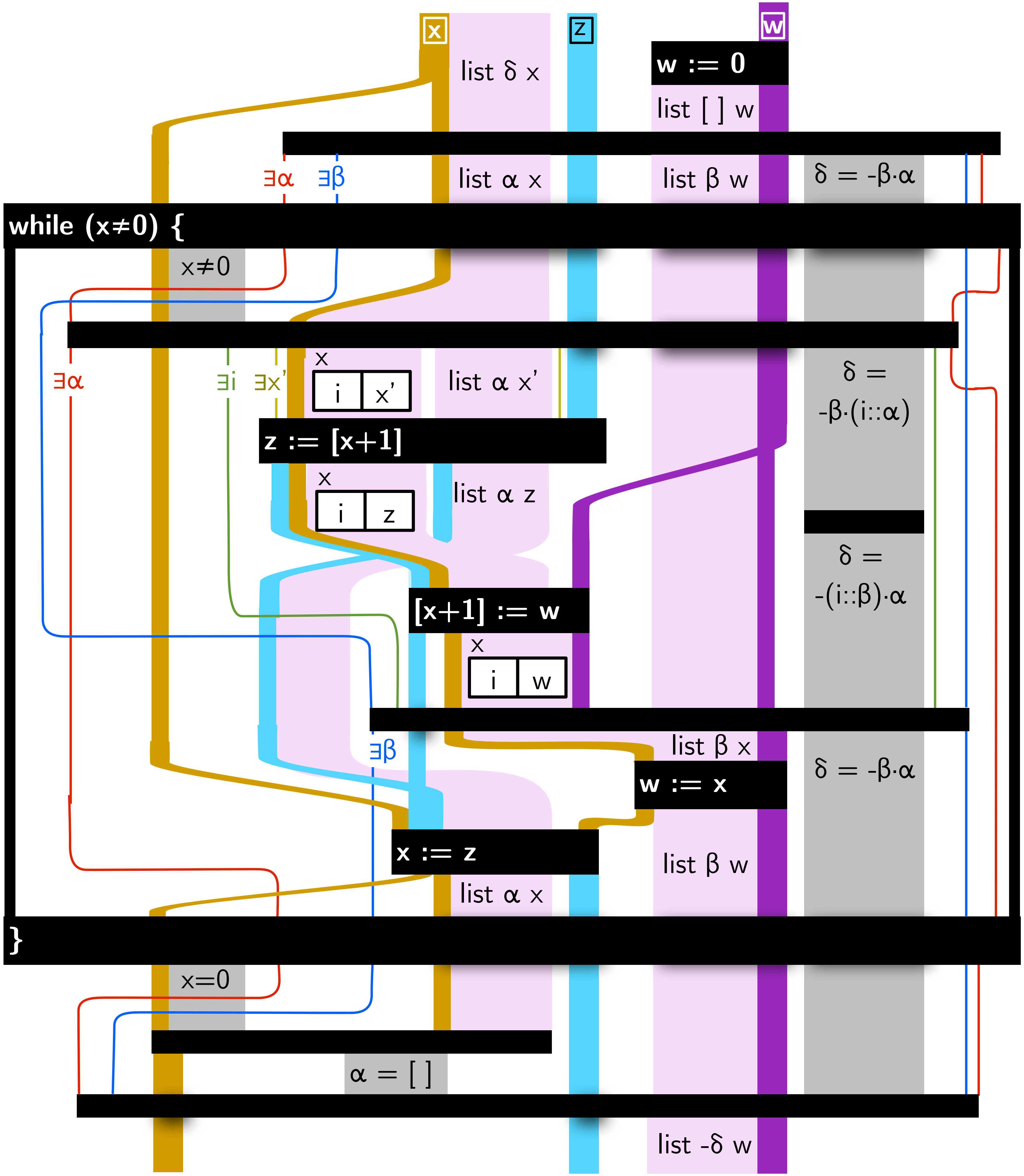


Variables as resource



Variables as resource




x
z
w
list δ x
w := 0
list [ ] w
∃α ∃β
list α x
list β w
δ = -β·α
while (x≠0) {
x≠0
∃α
∃i ∃x'
x
i x'
list α x'
δ =
-β·(i::α)
z := [x+1]
x
i z
list α z
δ =
-(i::β)·α
[x+1] := w
x
i w
∃β
list β x
w := x
δ = -β·α
list β w
x := z
list α x
}
x=0
α = [ ]
list -δ w

Some further reading

Separation Logic: A Logic for Shared Mutable Data Structures. By John C. Reynolds. In Proceedings of LICS, 2002.
Available from http://www.cs.cmu.edu/~jcr/seplogic.pdf

The main reference for newcomers to separation logic.

Ribbon Proofs for Separation Logic. By John Wickerson, Mike Dodds and Matthew Parkinson. In Proceedings of ESOP, 2013.
Available from http://www.cl.cam.ac.uk/~jpw48/ribbons_esop13.pdf

Introduces the 'graphical' reading of separation logic.

Variables as Resource in Separation Logic. By Richard Bornat, Cristiano Calcagno and Hongseok Yang. In Proceedings of MFPS, 2006.
Available from http://www.eis.mdx.ac.uk/staffpages/r_bornat/papers/bornatentcs.pdf

Introduces the idea of treating program variables as 'resources'.