```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```

```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```

```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```

in =

```
"a" :: ("b" :: ("c" :: ("d" :: ("e" :: []))))
```

```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

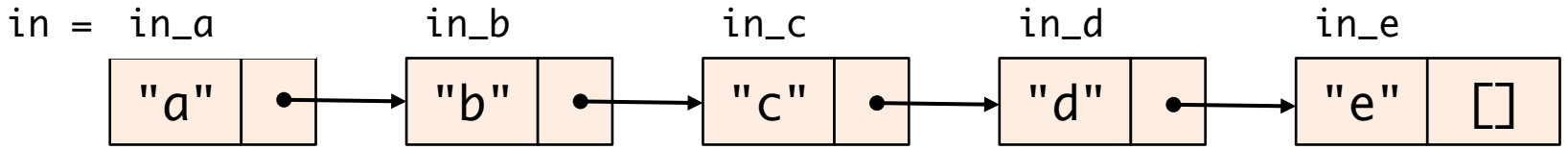
```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```

```
in =
  (::) "a" ((::) "b" ((::) "c" ((::) "d" ((::) "e" []))))
```

```
type List a
  ~= []
   | (::) a (List a)
```

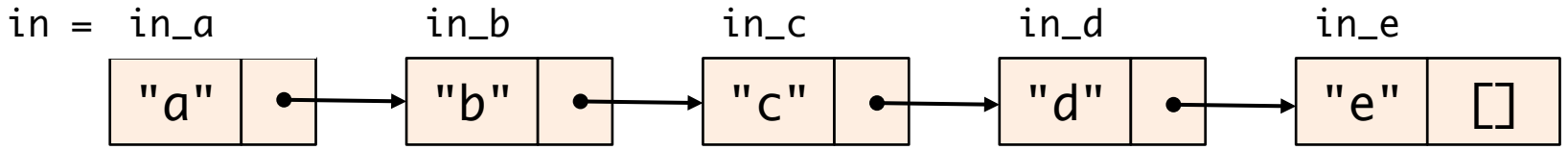
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

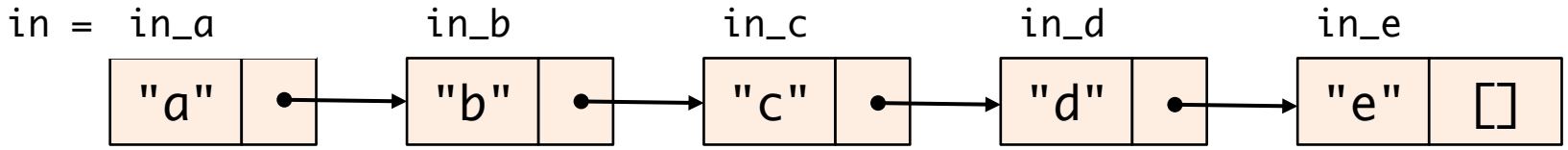
```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```




```
update in_a 2 "X"
```

```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```

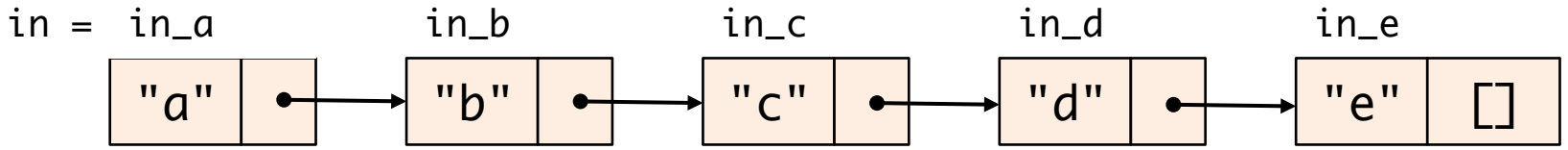



```
update in_a 2 "X"
```

```
xs = in_a  i = 2  y = "X"
```

```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

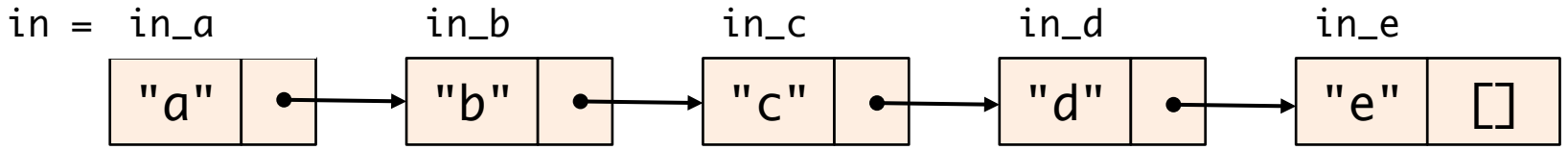
```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



update in_a 2 "X"	xs = in_a i = 2 y = "X"
	x = "a" rest = in_b

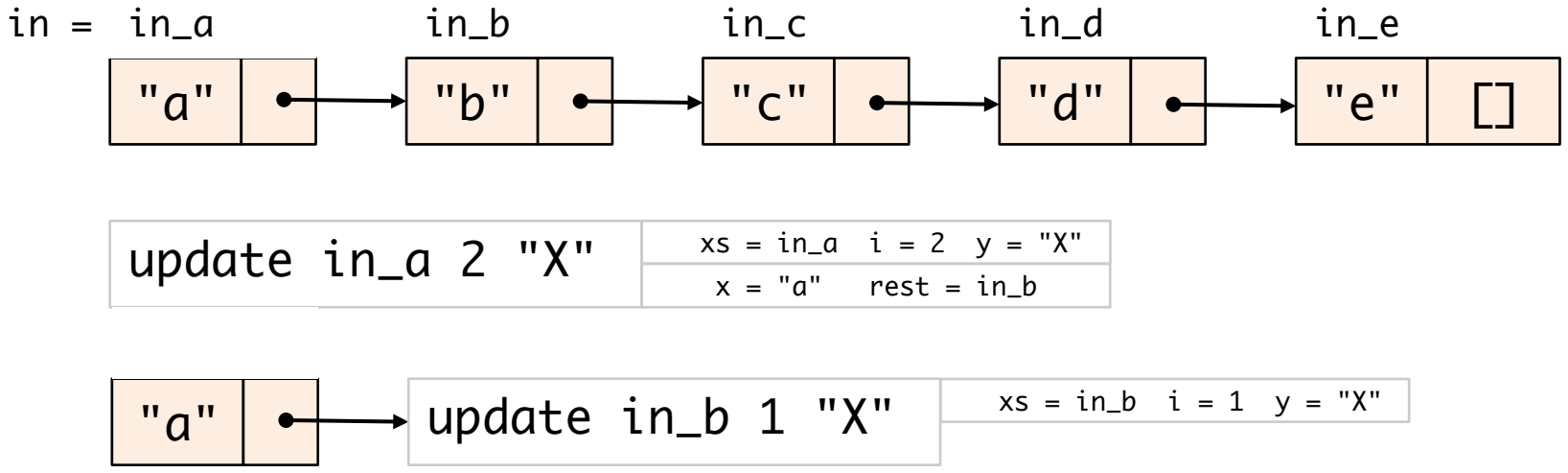
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



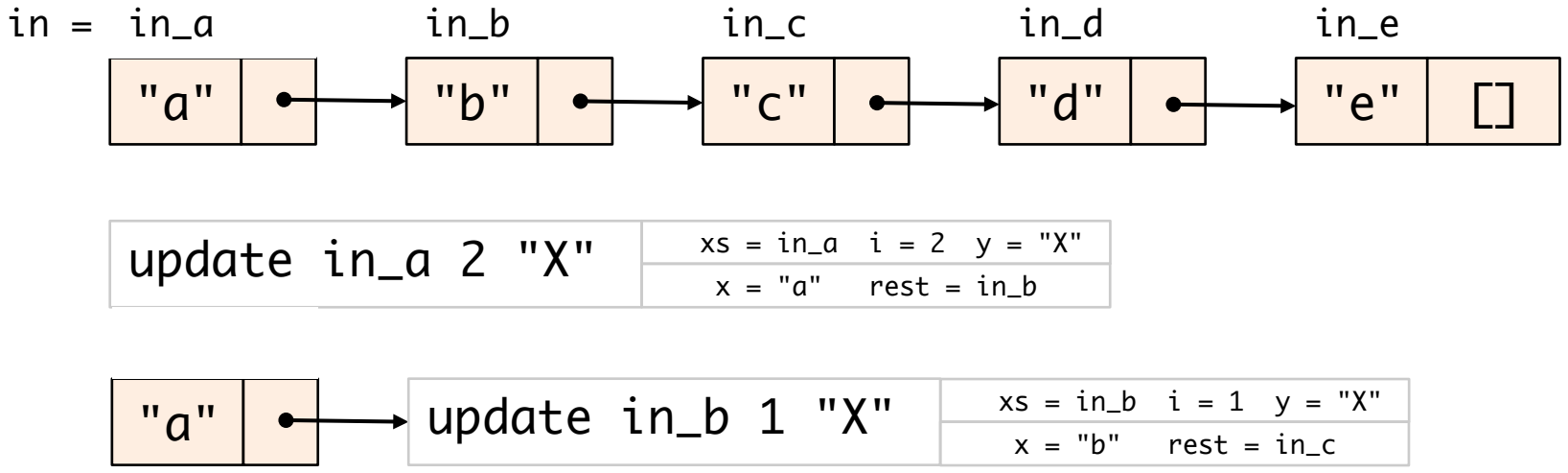
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



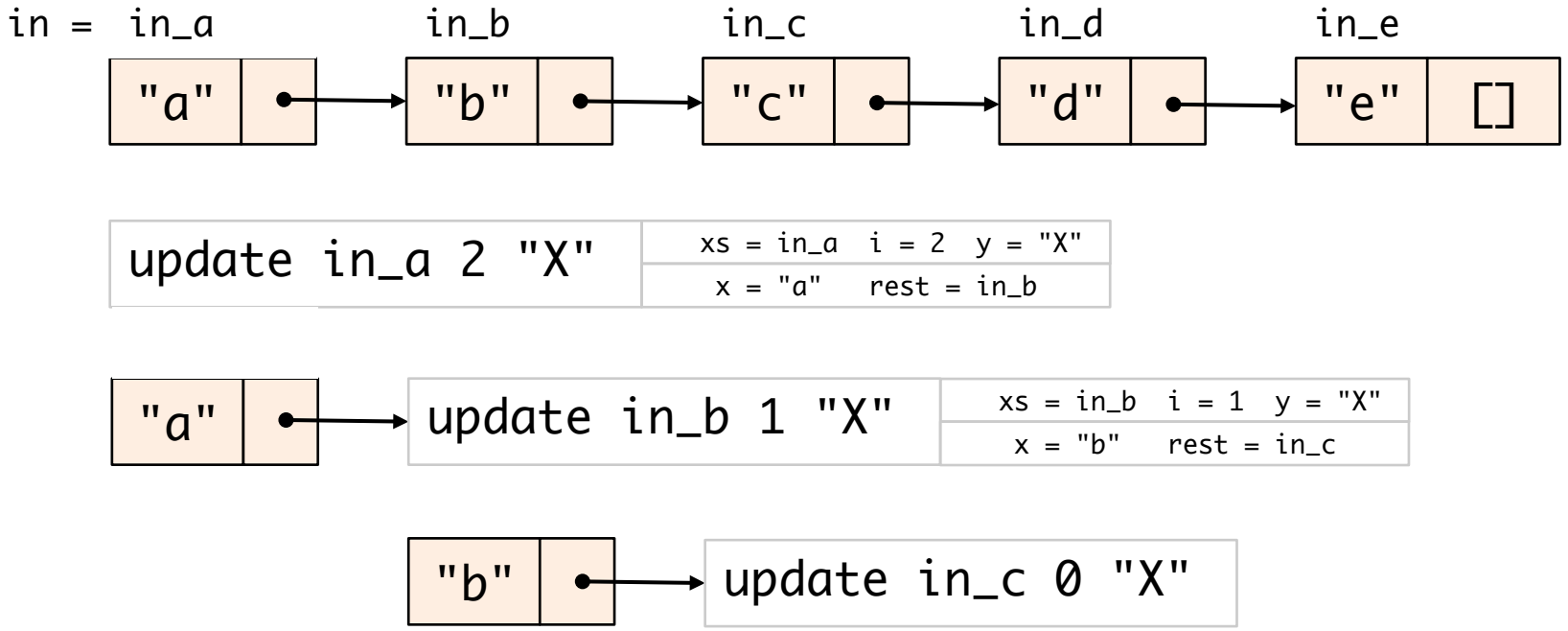
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



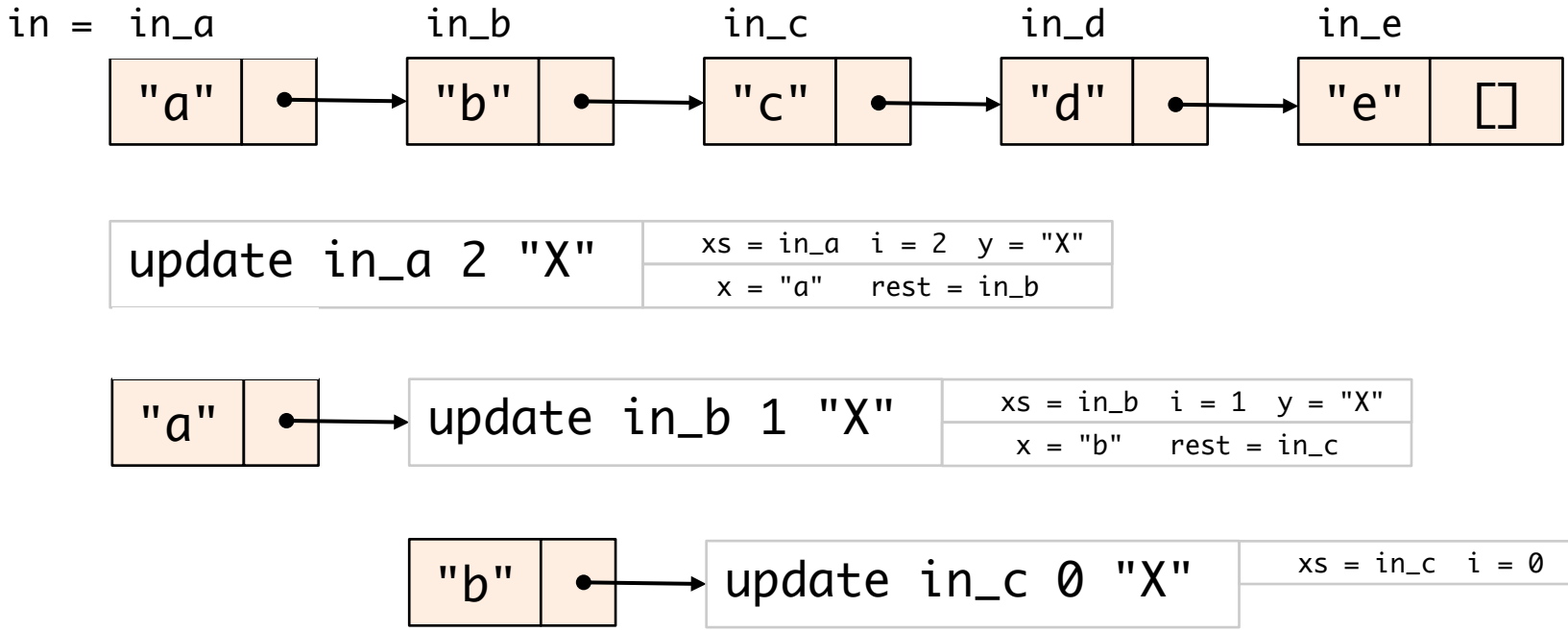
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



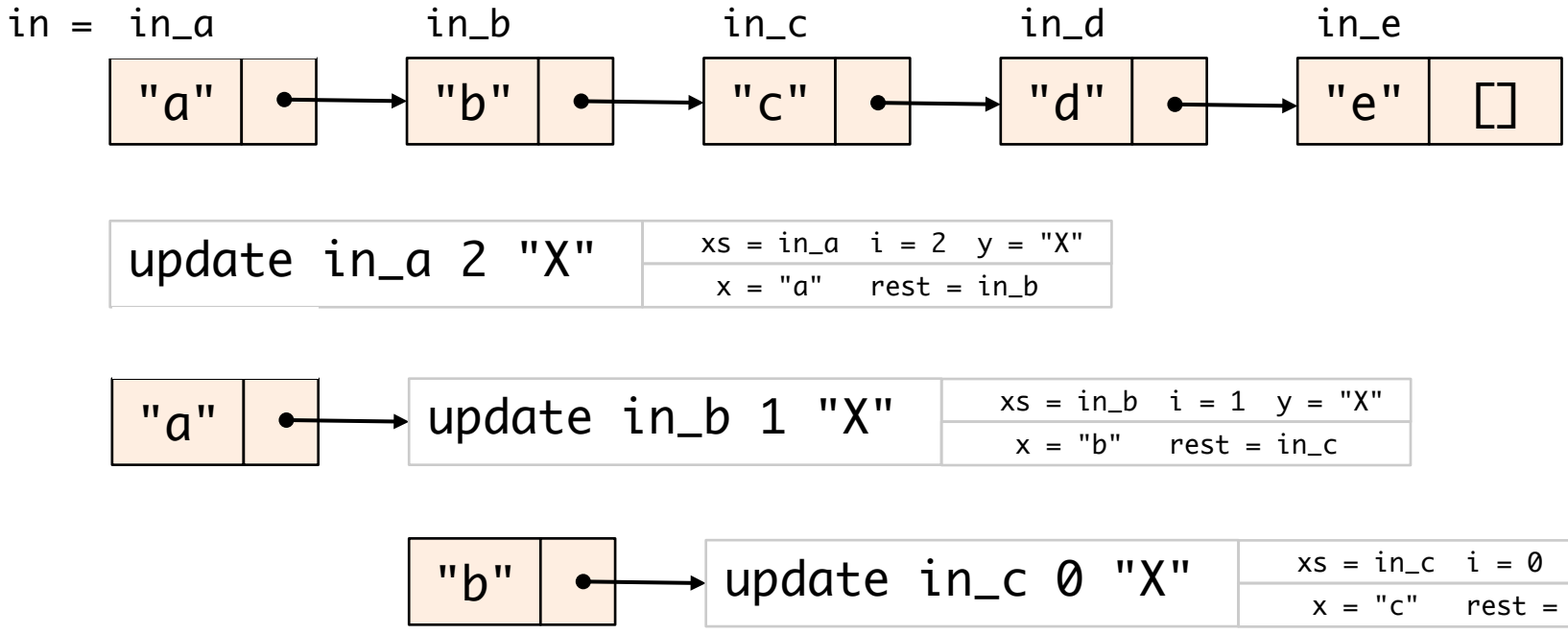
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



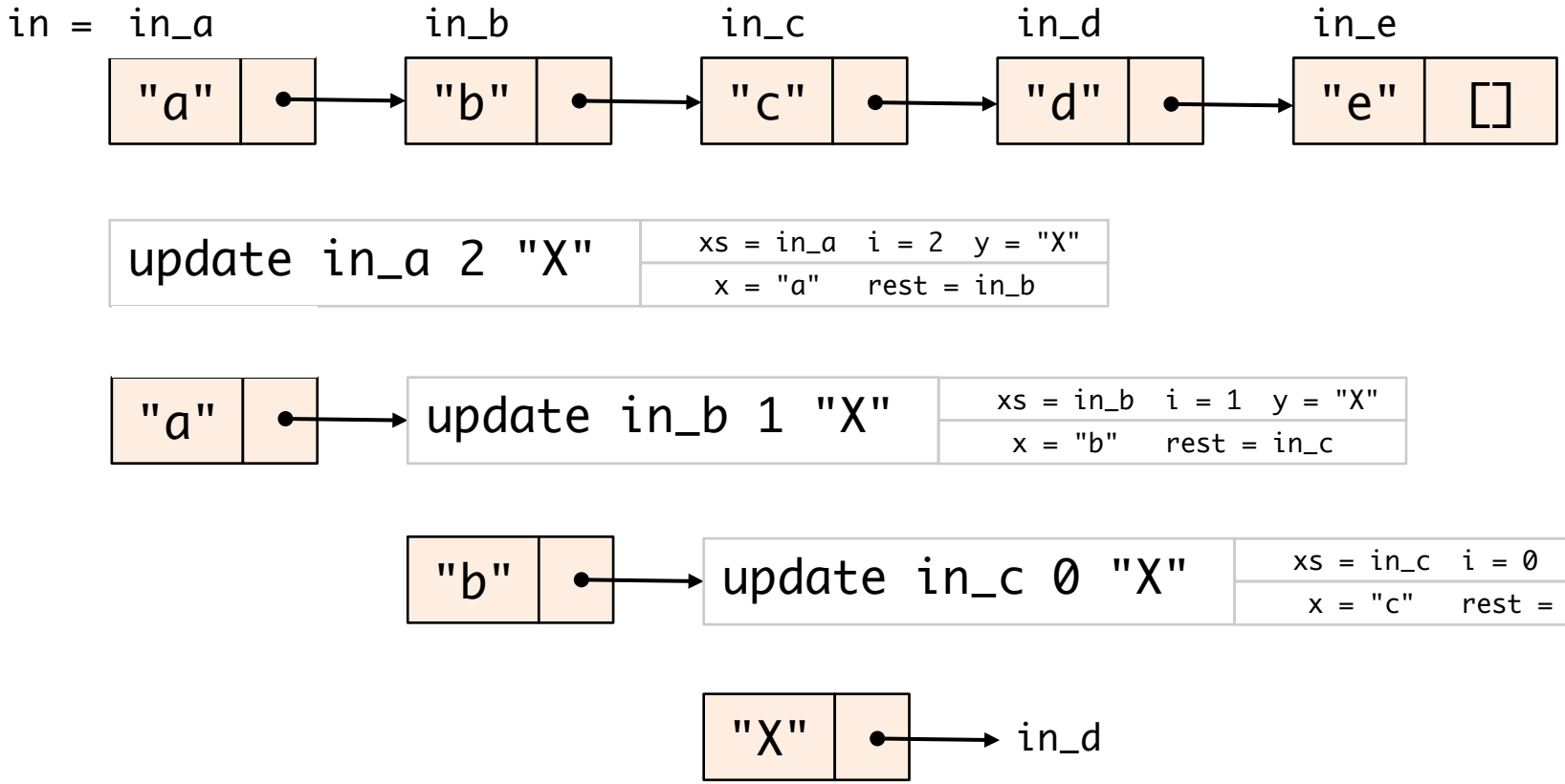
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



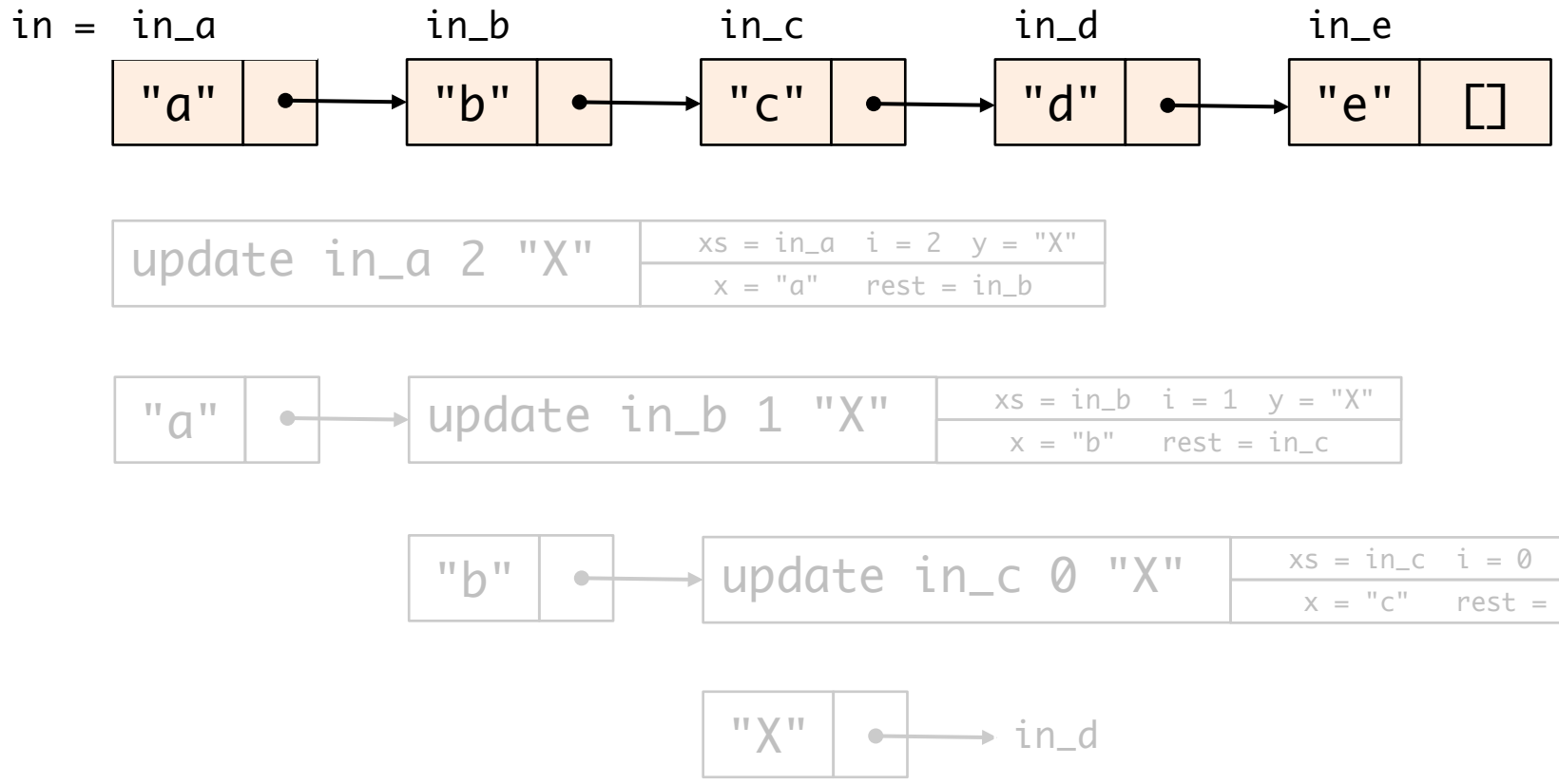
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



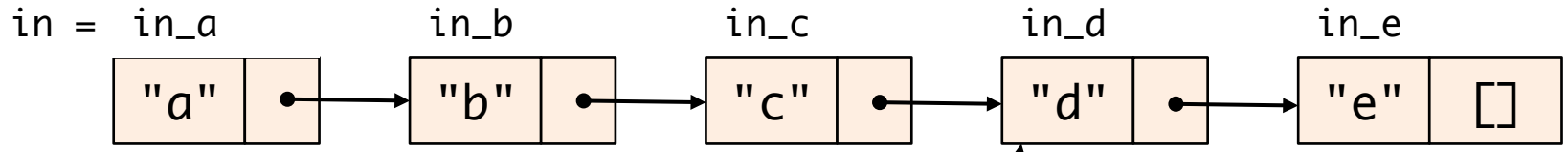
```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



```
let
  in  = ["a", "b", "c", "d", "e"]
  out = update in 2 "X"
in
  ...
```

```
update : List a -> Int -> a -> List a
update xs i y =
  case (xs, i) of
    ([], _) ->
      []
    (x::rest, 0) ->
      y :: rest
    (x::rest, _) ->
      x :: update rest (i-1) y
```



Two cons-cells shared

Three cons-cells allocated