Disjointness Domains for Fine-Grained Aliasing

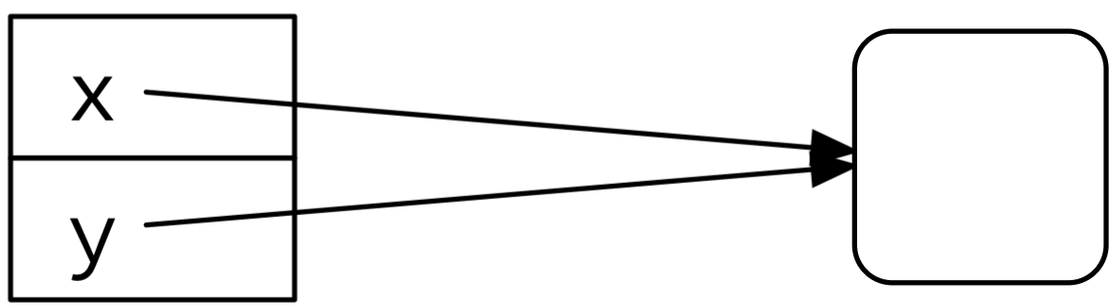
Local + Static Reasoning About Shared Mutable State

Stephan Brandauer | @sbrandauer | http://stbr.me
Dave Clarke
Tobias Wrigstad

Uppsala University

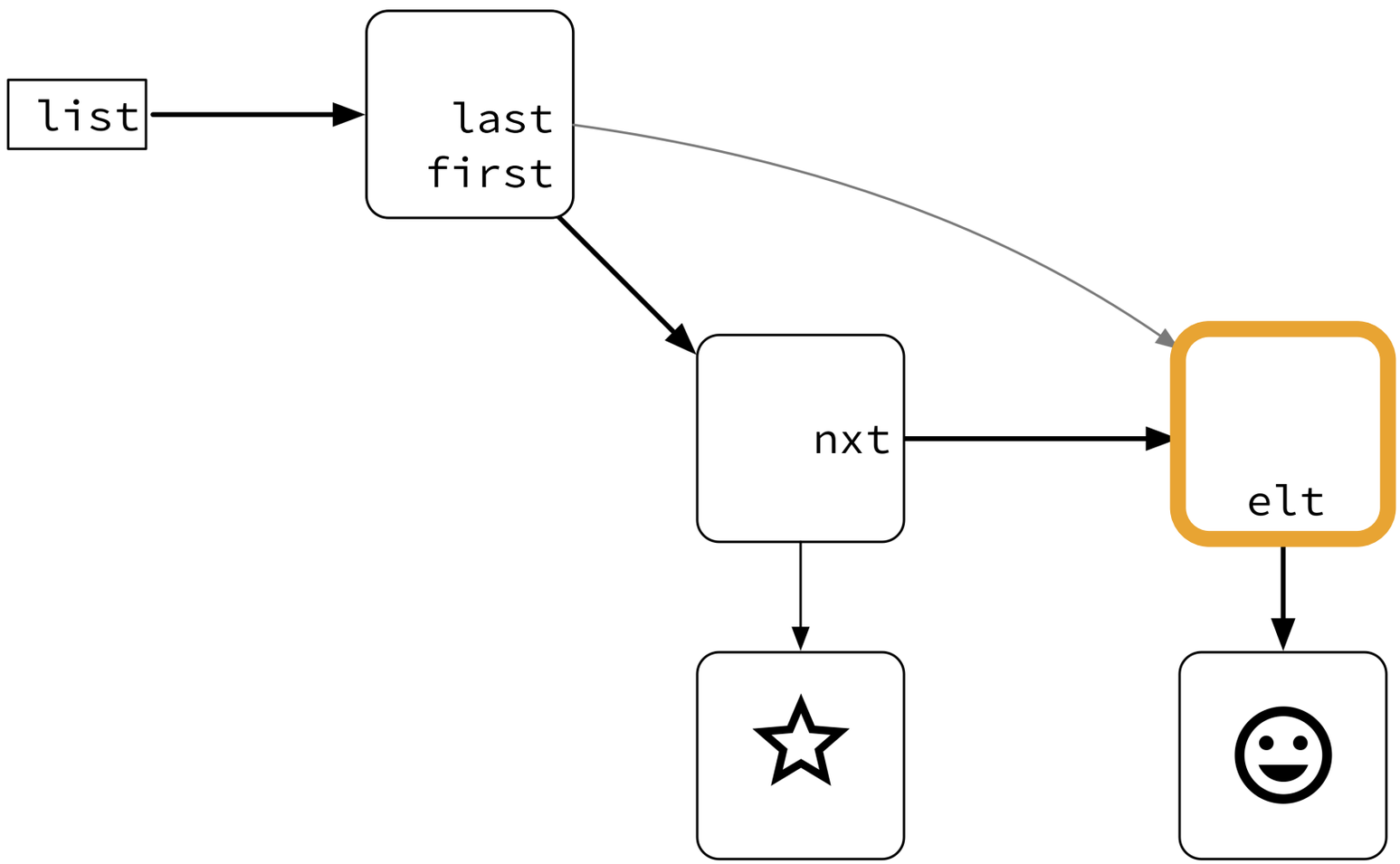
Aliasing

$$x \equiv y$$



Aliasing

list.first.nxt

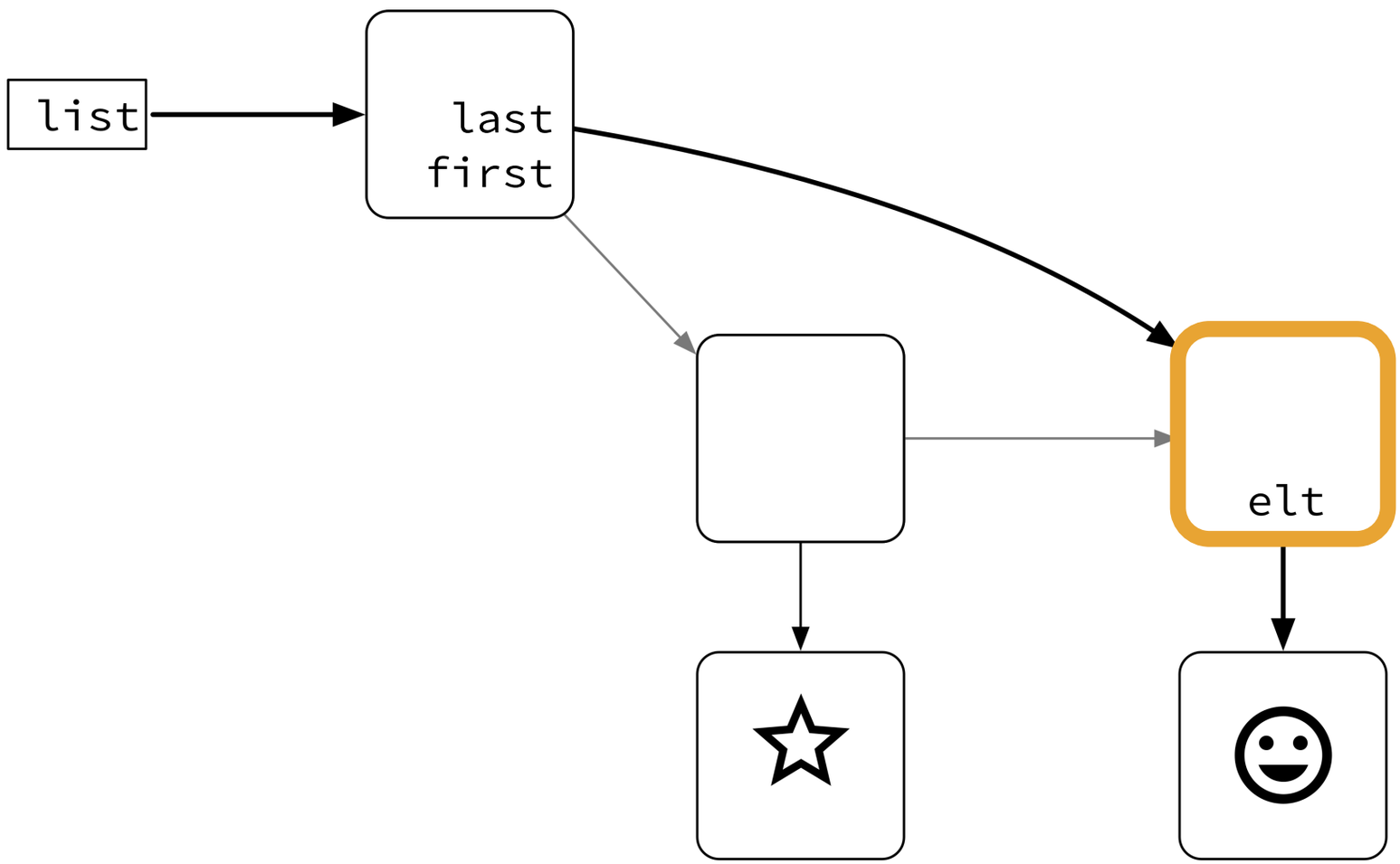


Aliasing

`list.first.nxt`

≡

`list.last`



Aliasing: Blessing/Curse

Three Examples

	Aliased	Unaliased
CORRECTNESS	?	?
PARALLELISM	?	?
EXPRESSIVITY	?	?

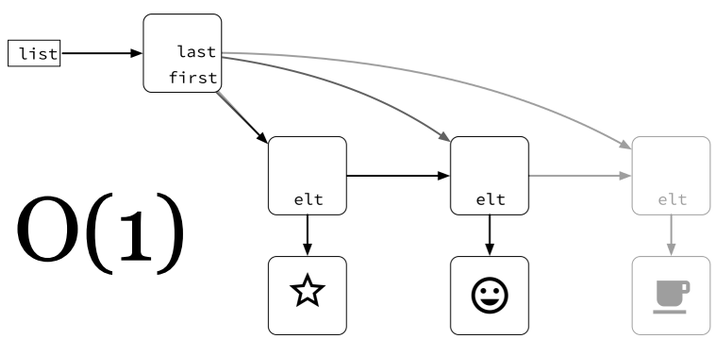
Three Examples

	Aliased	Unaliased
`x.close();` `y.read();`	?	?
PARALLELISM	?	?
EXPRESSIVITY	?	?

Three Examples

	Aliased	Unaliased
`x.close();` `y.read();`	?	?
`x.set(12) || y.set(13)`	?	?
EXPRESSIVITY	?	?

Three Examples

	Aliased	Unaliased
`x.close();` `y.read();`	?	?
`x.set(12) || y.set(13)`	?	?
 O(1)	?	?

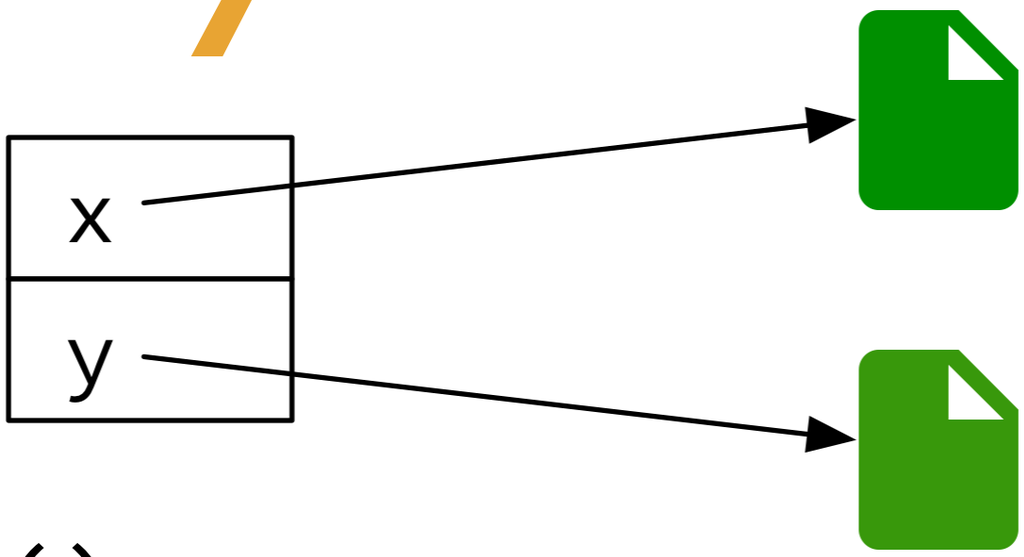
DESIGN OPTION No. 1:

All Vars/Fields May Alias

All Vars May Alias

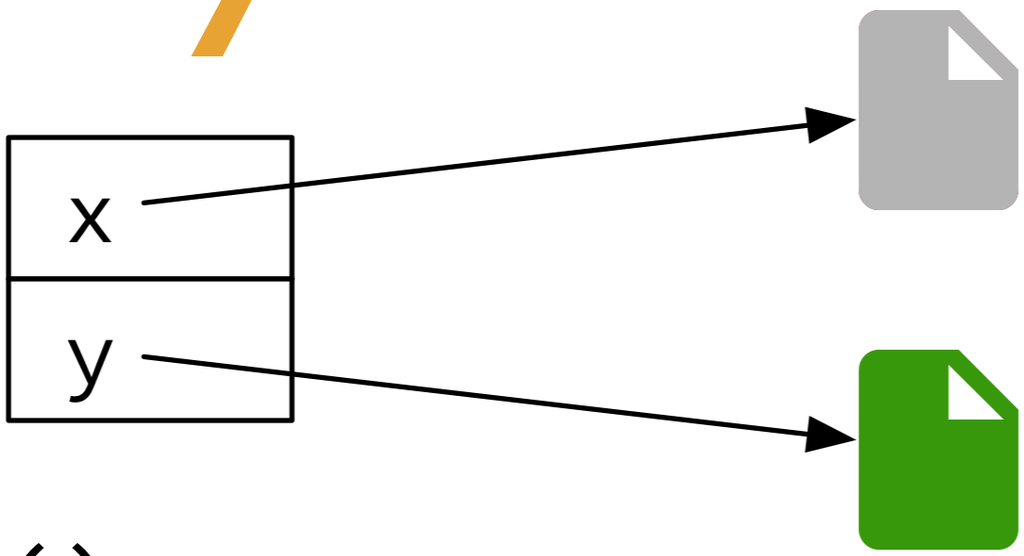
```
x.close();
y.read();
```

All Vars May Alias



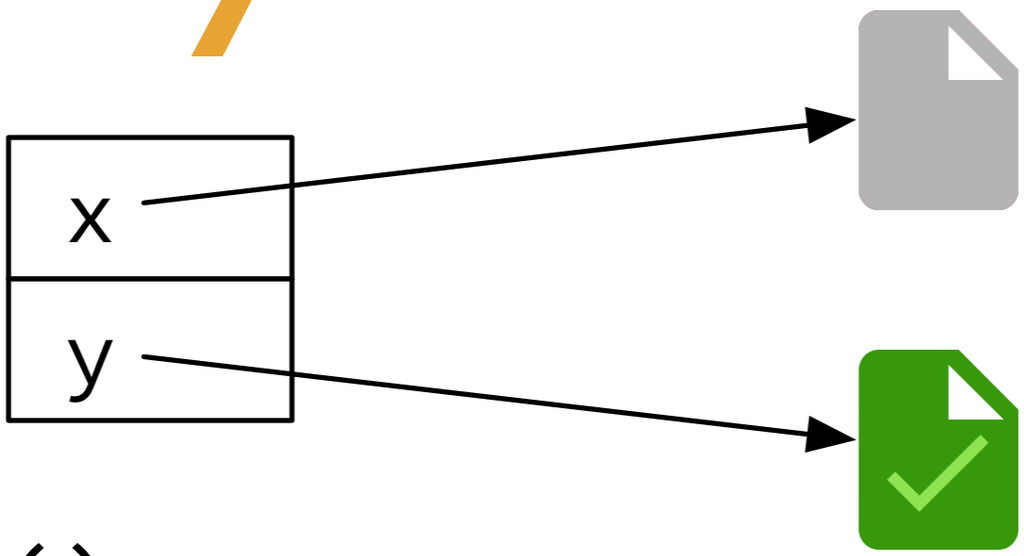

```
x.close();
y.read();
```

All Vars May Alias



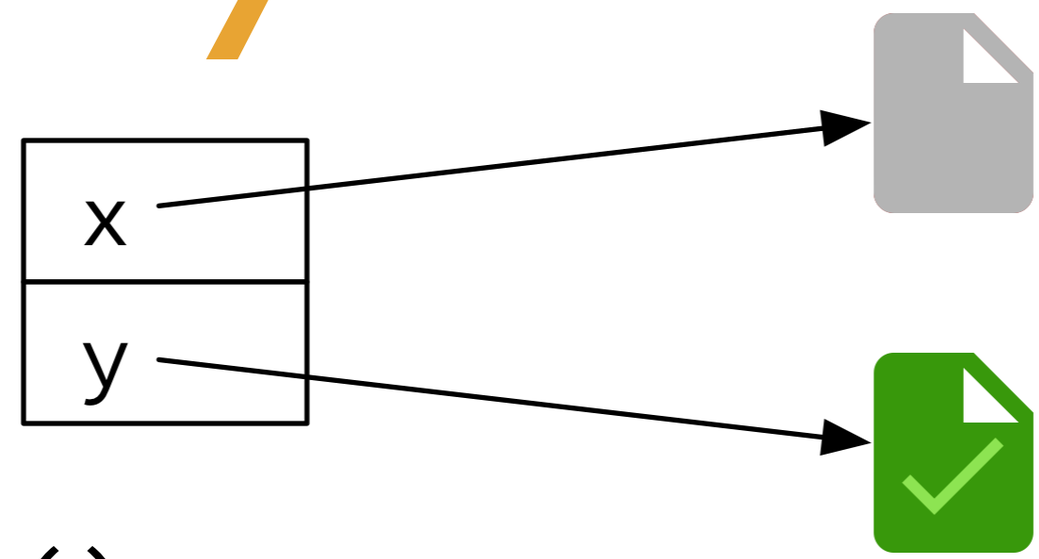

```
x.close();
y.read();
```

All Vars May Alias

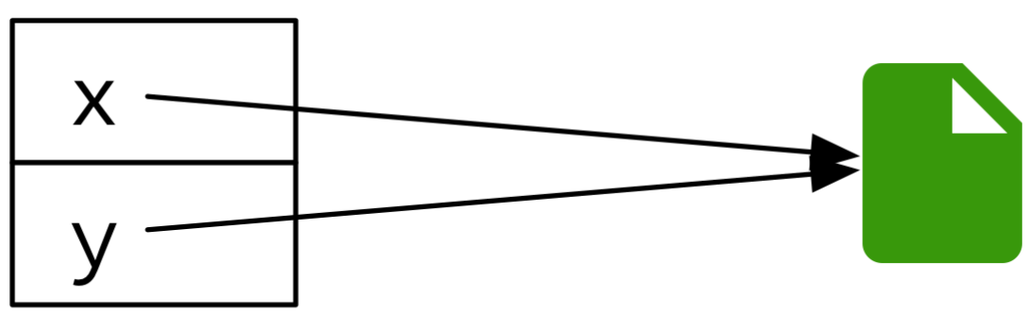



```
x.close();
y.read();
```

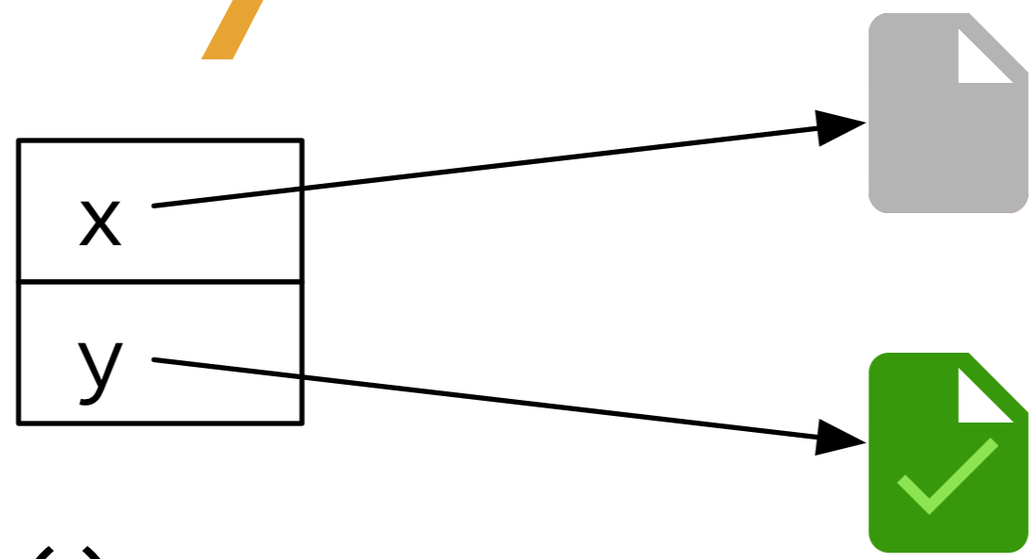
All Vars May Alias



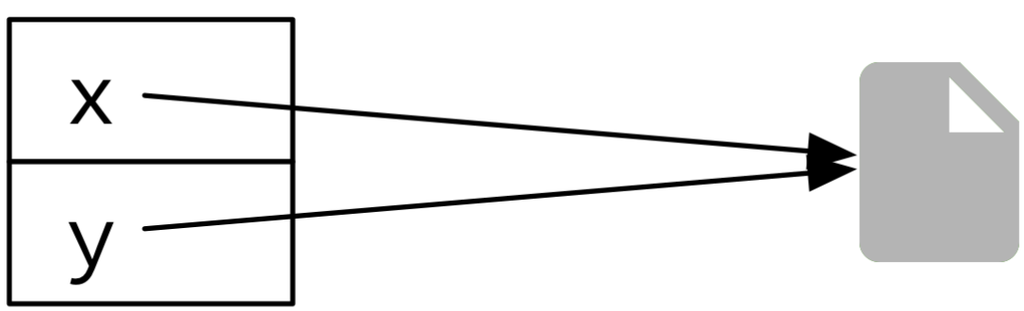

```
x.close();
y.read();
```



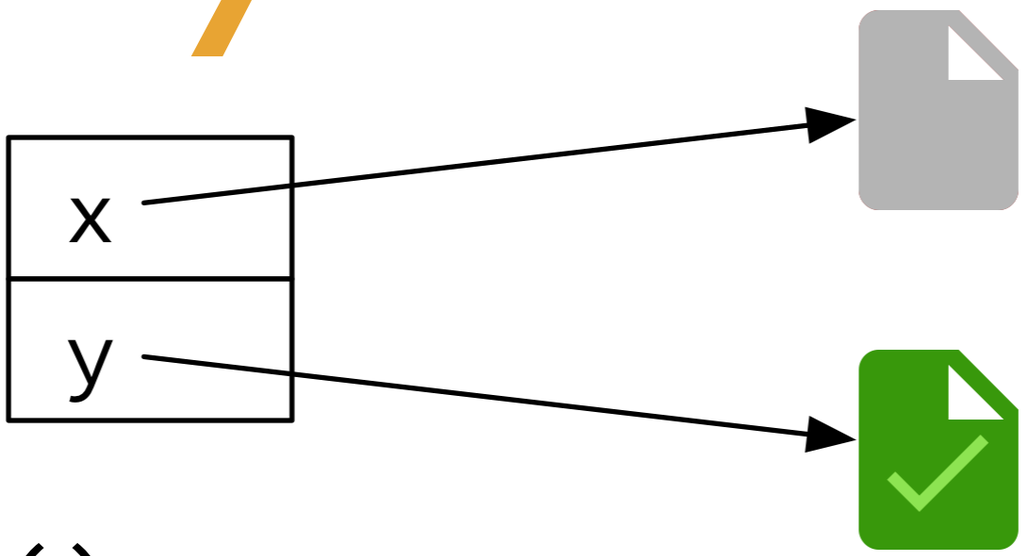
All Vars May Alias



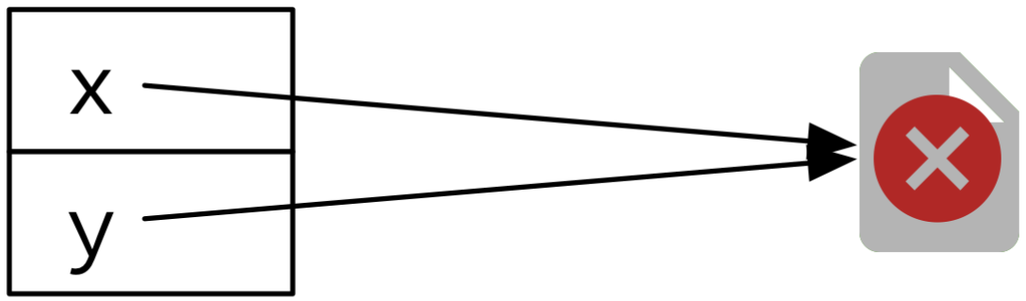

```
x.close();
y.read();
```



All Vars May Alias




```
x.close();
y.read();
```



All Vars May Alias

```
x.close();
y.read();
```

All Vars May Alias

```
x.close();
y.read();
```

✓ $\Leftrightarrow x \not\equiv y$

All Vars May Alias

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

All Vars May Alias

```
if(      ){
  ;
  closeRead(a,b);
  ;
}
closeRead(r,s);
;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

All Vars May Alias

```
if(      ){
  ;
  closeRead(a,b);
  ;
}
closeRead(r,s);
;
```

```
while (      ) {
  closeRead(v,w);
}
;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

All Vars May Alias

```
if(    ){
  ____;
  closeRead(a,b);
  ____;
}
closeRead(r,s);
_______;
```

```
while (_____) {
  closeRead(v,w);
}
_______;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

```
_____;
if (_____){
  _____;
  closeRead(t,u);
}
```

All Vars May Alias

```
if(      ){
  ;
  closeRead(a,b);
  ;
}
closeread(r,s);
;
```

```
while (      ) {
  closeRead(v,w);
}
;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

```
;
if (      ){
  ;
  closeRead(t,u);
}
```

All Vars May Alias

```
if(    ){
  ____;
  closeRead(a,b);
  ____;
}
closeread(r,s);
_____;
```

```
while (_____) {
  closeRead(v,w);
}
_____;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

```
_____;
if (_____){
  _____;
  closeRead(t,u);
}
```

All Vars May Alias

```
if(      ){
  ;
  closeRead(a,b);
  ;
}
closeread(r,s);
;
```

```
while (      ) {
  closeRead(v,w);
}
;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

```
;
if (      ){
  ;
  closeRead(t,u);
}
```

All Vars May Alias

```
if(    ){
  ____;
  closeRead(a,b);
  ____;
}
closeread(r,s);
____;
```

```
while (____) {
  closeRead(v,w);
}
____;
```

```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

```
____;
if (____){
  ____;
  closeRead(t,u);
}
```

All Vars May Alias

```
if(    ){
  ____;
  closeRead(a,b);
  ____;
}
closeread(r,s);
____;
```

```
while (____) {
  closeRead(v,w);
}
____;
```

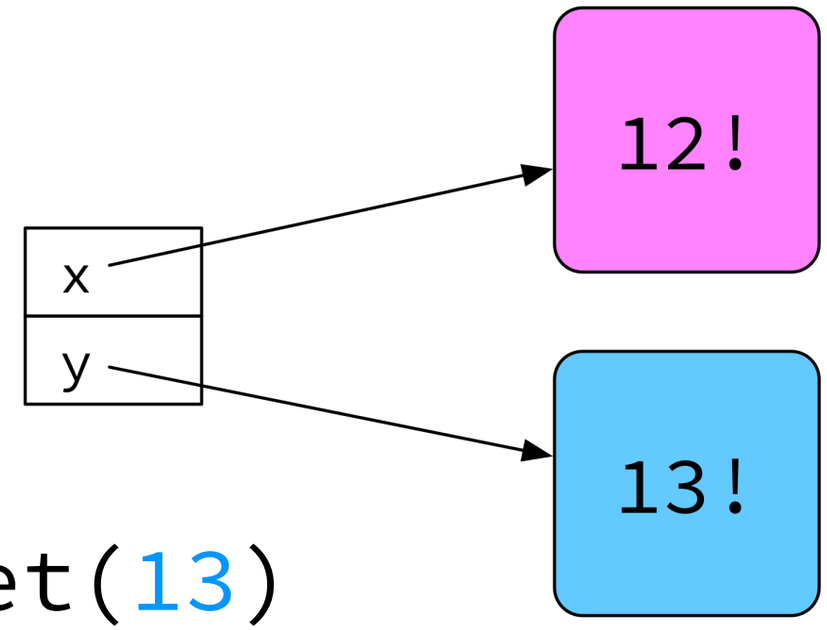
```
def closeRead(x, y) {
  x.close();
  y.read();
}
```

```
____;
if (____){
  ____;
  closeRead(t,u);
}
```

All Vars May Alias

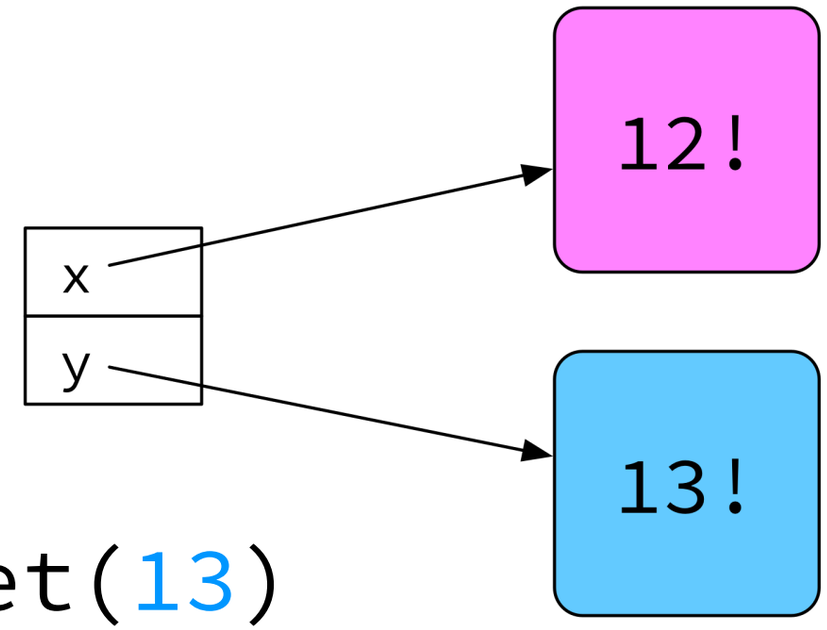
```
x.set(12) || y.set(13)
```

All Vars May Alias

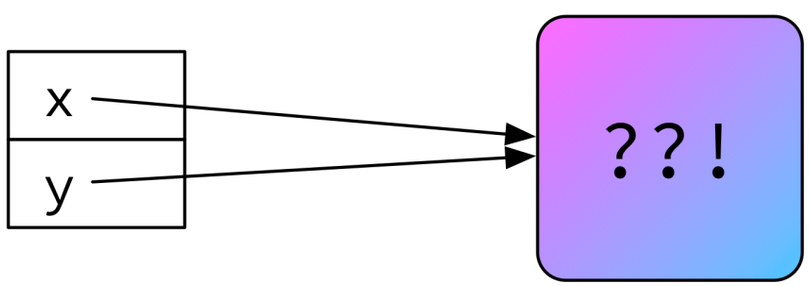



```
x.set(12) || y.set(13)
```

All Vars May Alias




```
x.set(12) || y.set(13)
```



All Vars May Alias

```
x.set(12) || y.set(13)
```

All Vars May Alias

```
x.set(12) || y.set(13)
```

✓ $\Leftrightarrow x \not\equiv y$

All Vars May Alias

```
def setPar(x, y) {
  x.set(12) || y.set(13);
}
```

All Vars May Alias

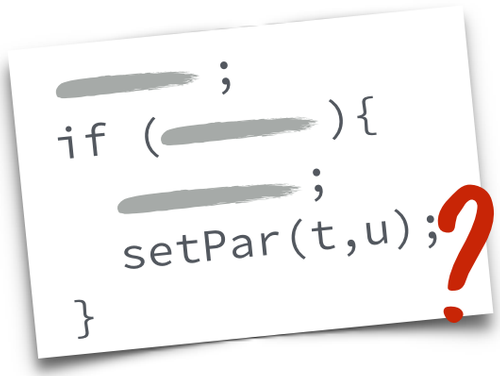
```
____ ;
if (____ ){
   ____ ;
   setPar(t,u);
}
```

```
while (____ ) {
   setPar(v,w);
}
____ ;
```

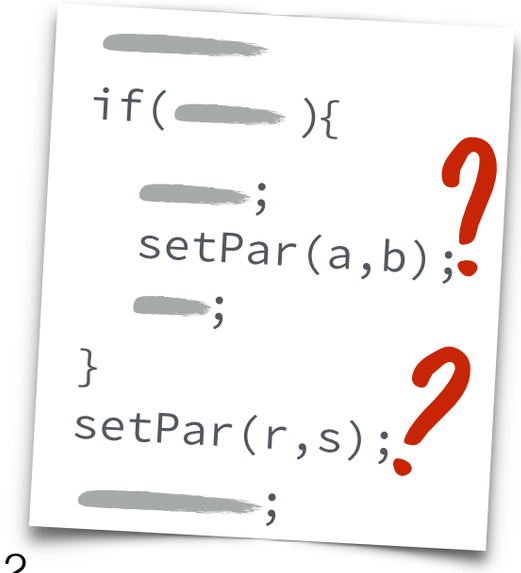
```
def setPar(x, y) {
   x.set(12) || y.set(13);
}
```

```
____
if(____ ){
   ____ ;
   setPar(a,b);
   ____ ;
}
setPar(r,s);
____ ;
```

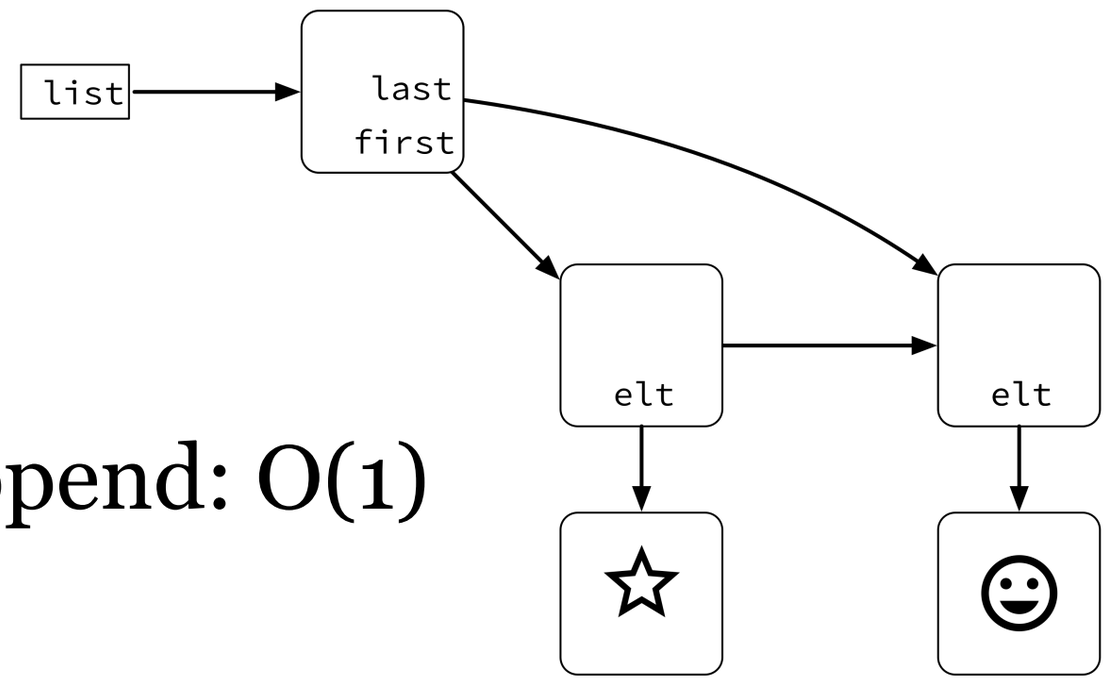
All Vars May Alias




```
def setPar(x, y) {
  x.set(12) || y.set(13);
}
```

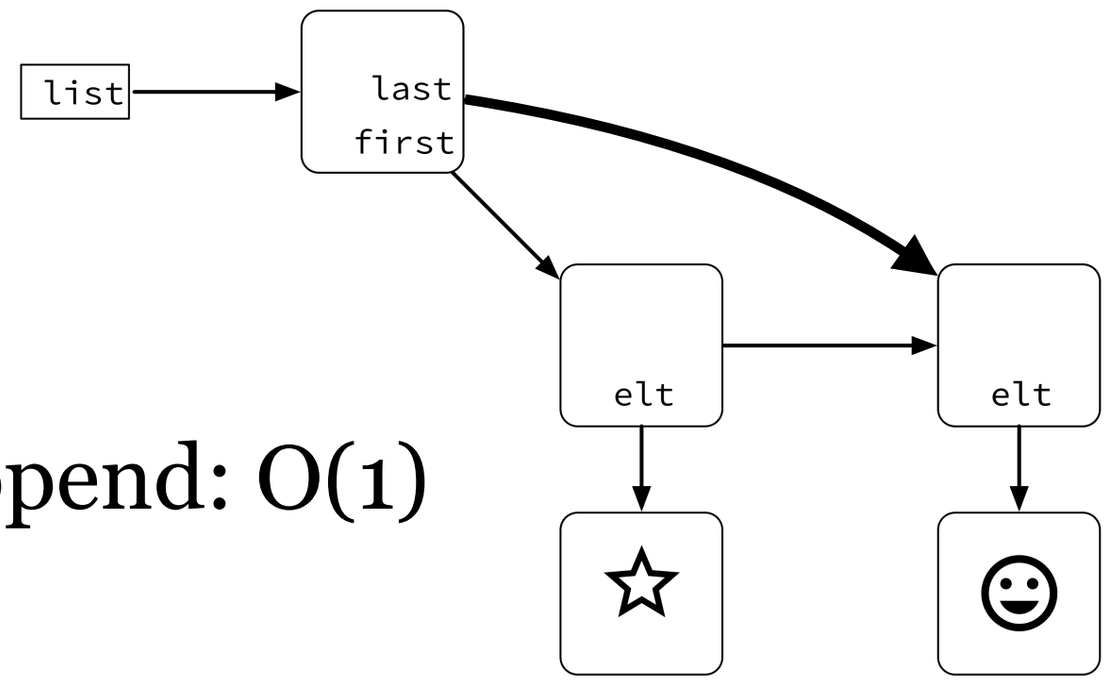


All Vars May Alias



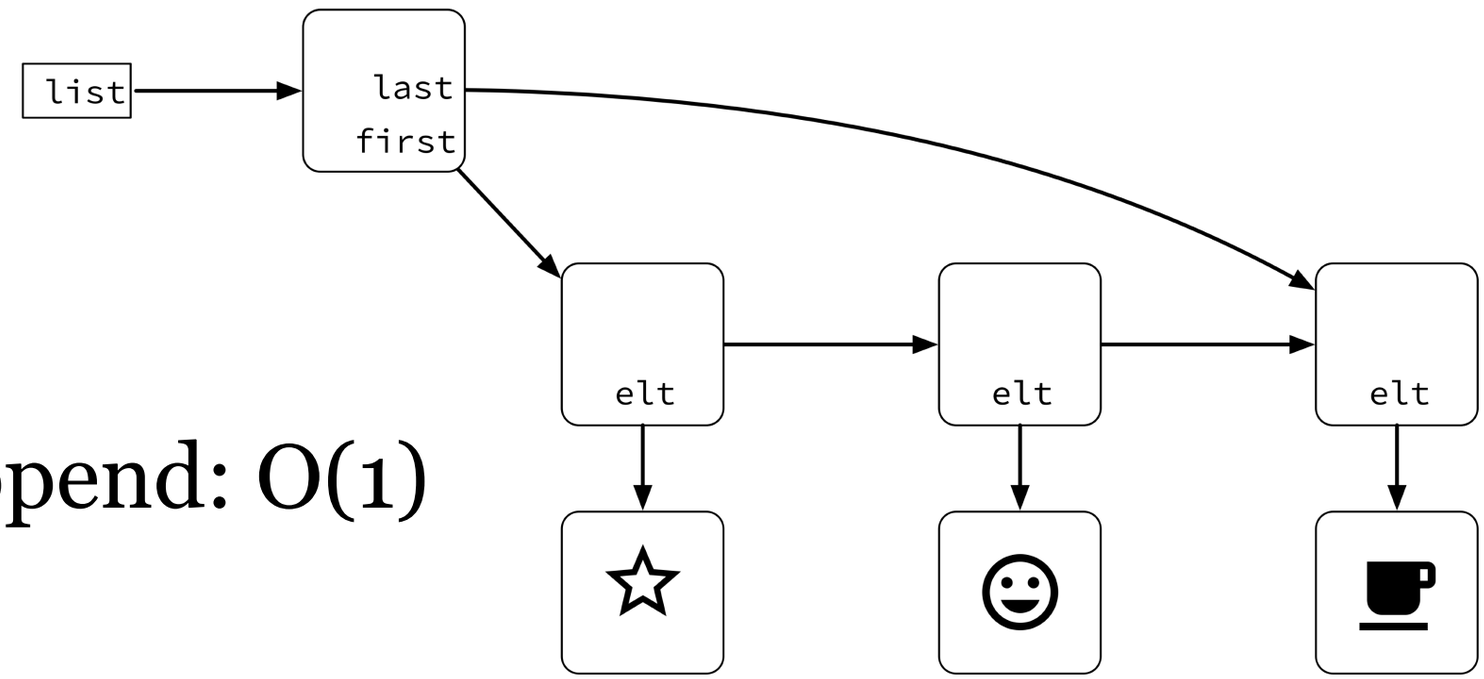
List append: O(1)

All Vars May Alias



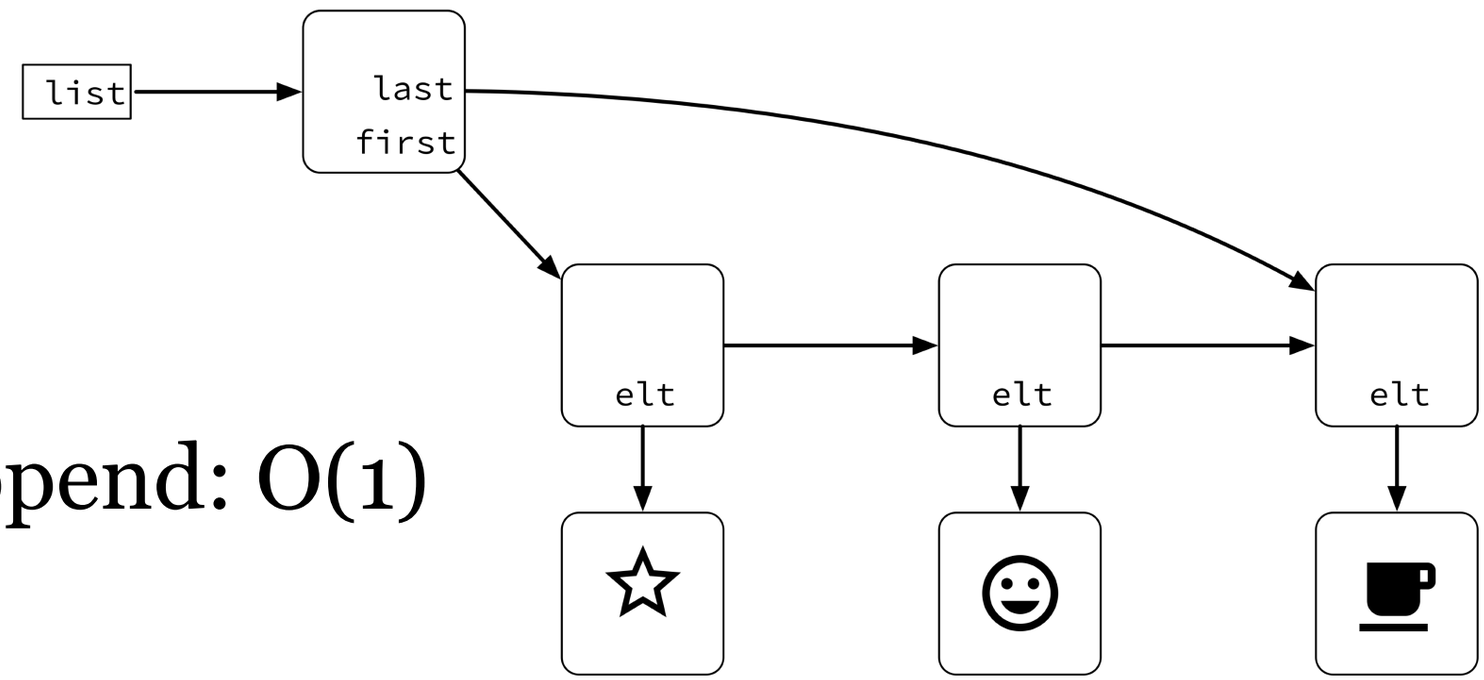
List append: O(1)

All Vars May Alias



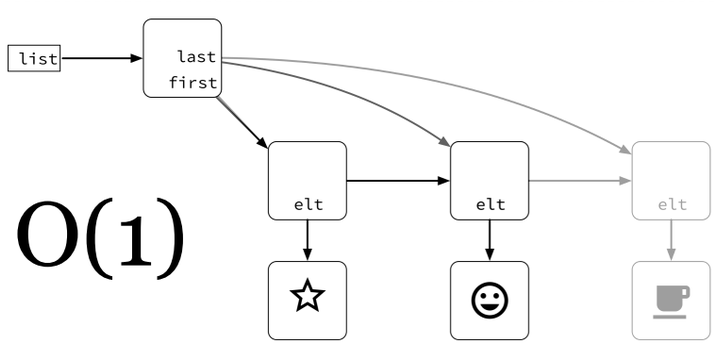
List append: O(1)

All Vars May Alias

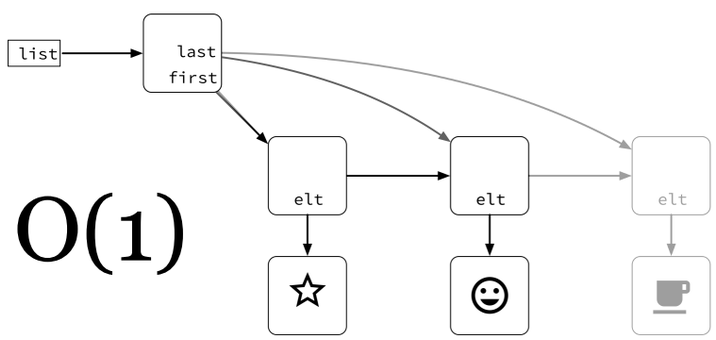


List append: O(1)

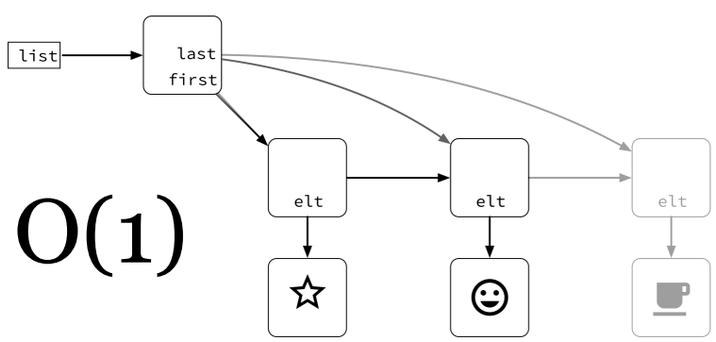
Aliased

	Aliased	Unaliased
`x.close();` `y.read();`		
`x.set(12) || y.set(13)`		
 O(1)		

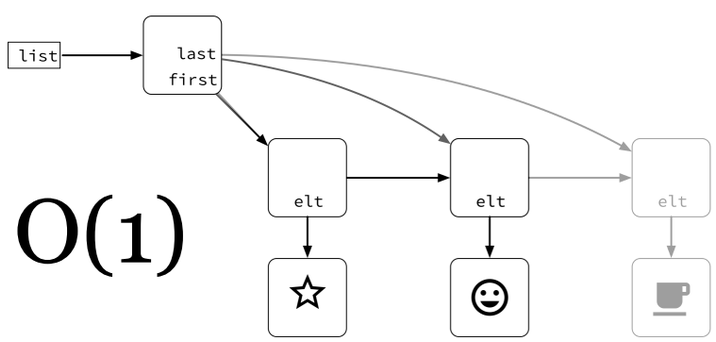
Aliased

	Aliased	Unaliased
`x.close();` `y.read();`	✕	
`x.set(12) || y.set(13)`		
 O(1)		

Aliased

	Aliased	Unaliased
`x.close();` `y.read();`	✕	
`x.set(12) || y.set(13)`	✕	
 O(1)		

Aliased

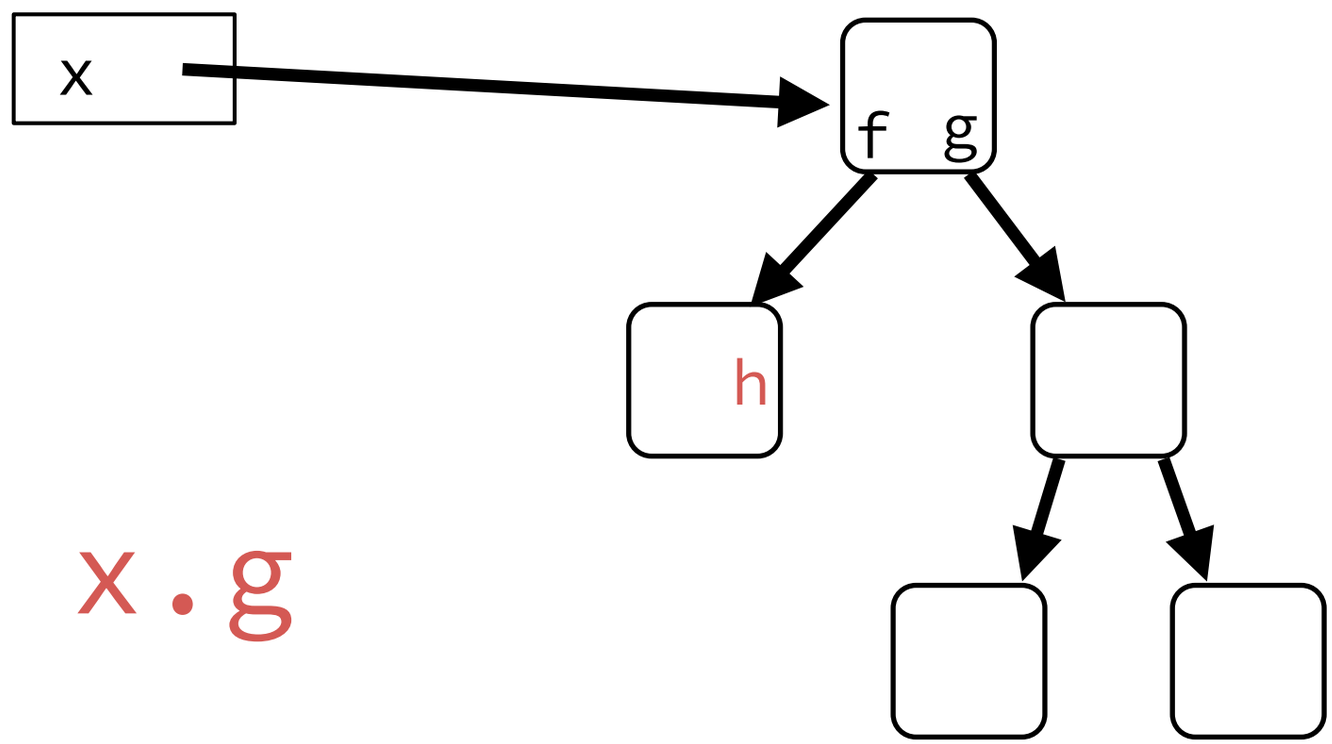
	Aliased	Unaliased
`x.close();` `y.read();`	✗	
`x.set(12) || y.set(13)`	✗	
 O(1)	✓	

DESIGN OPTION NO. 2:

No Vars/Fields May Alias

No Vars May Alias

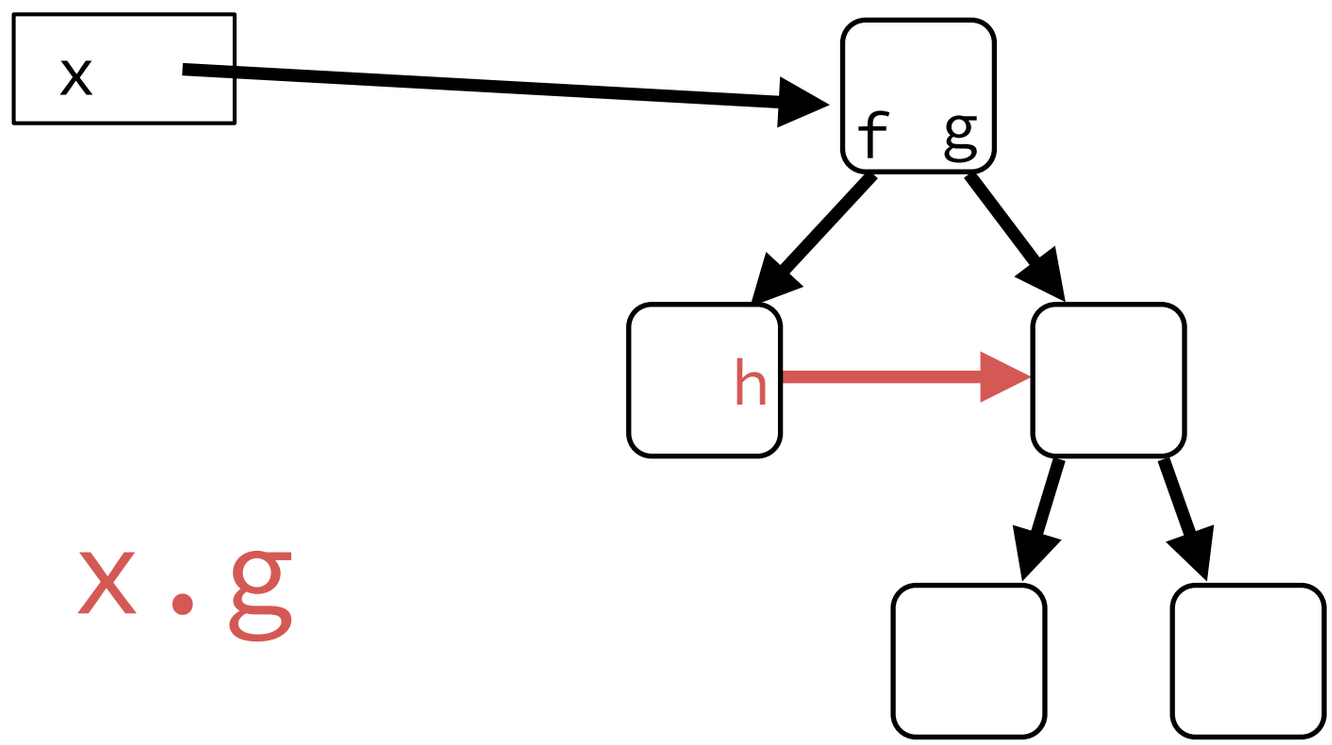
"Unique References"




```
x.f.h := x.g
```

No Vars May Alias

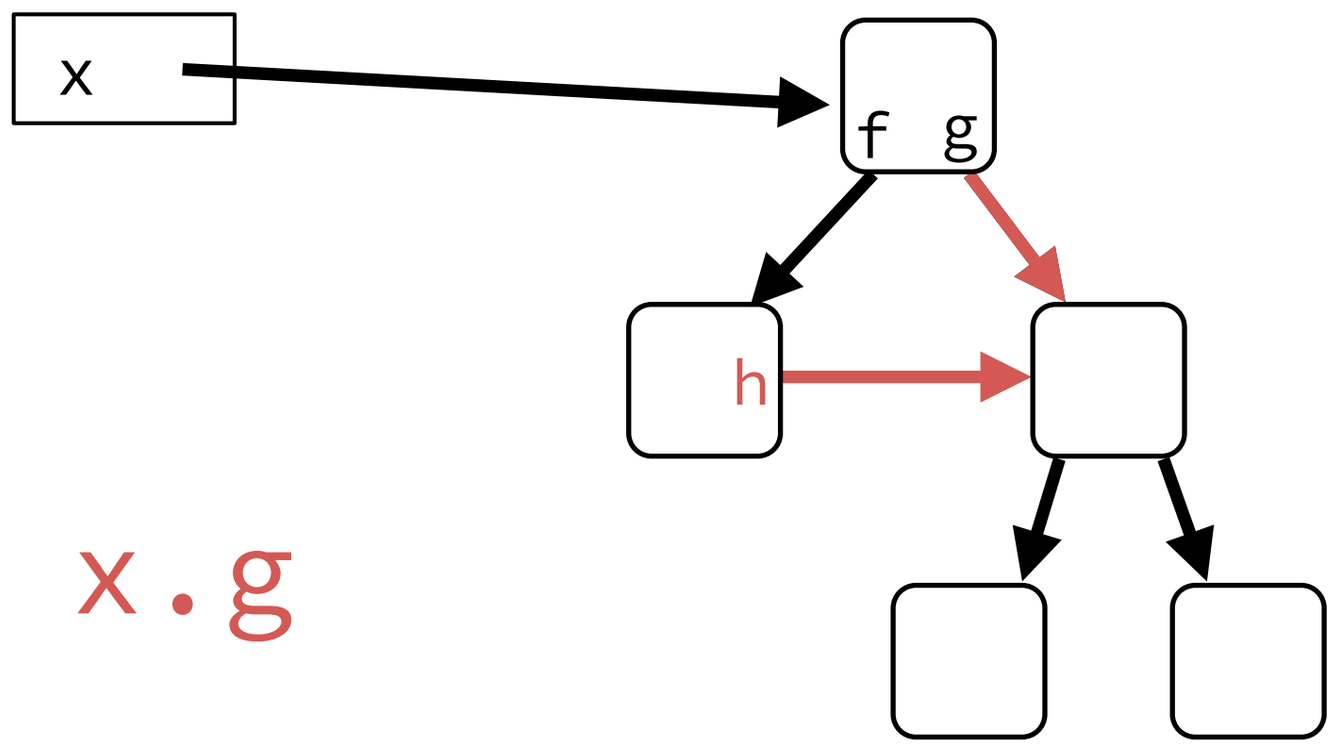
"Unique References"




```
x.f.h := x.g
```

No Vars May Alias

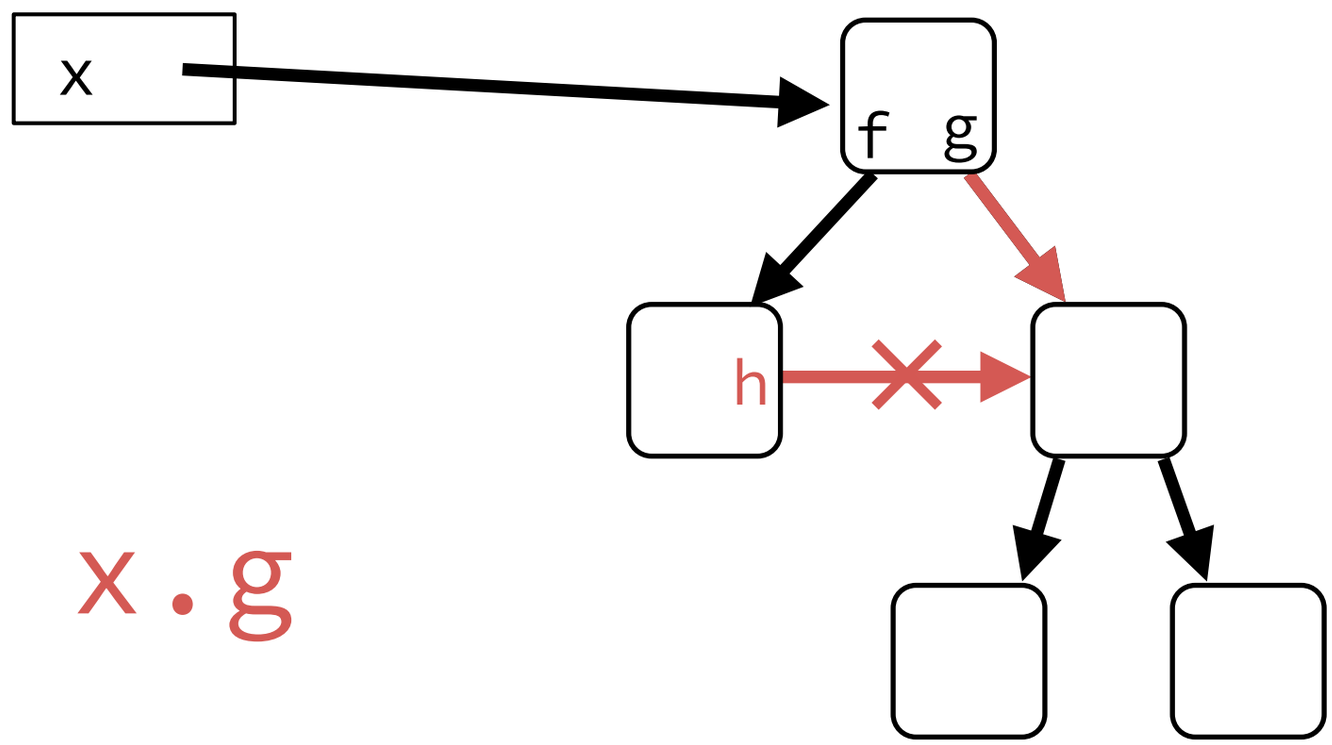
"Unique References"




```
x.f.h := x.g
```

No Vars May Alias

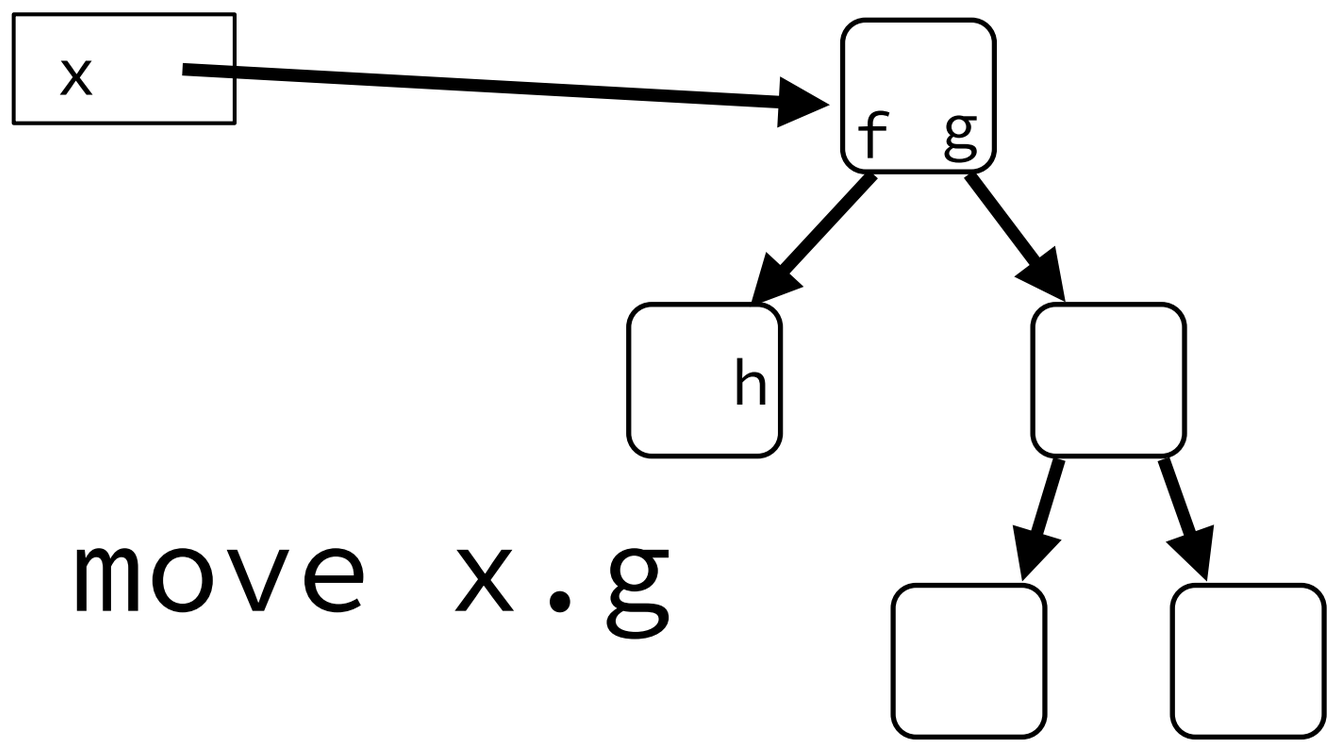
"Unique References"



x.f.h := x.g

No Vars May Alias

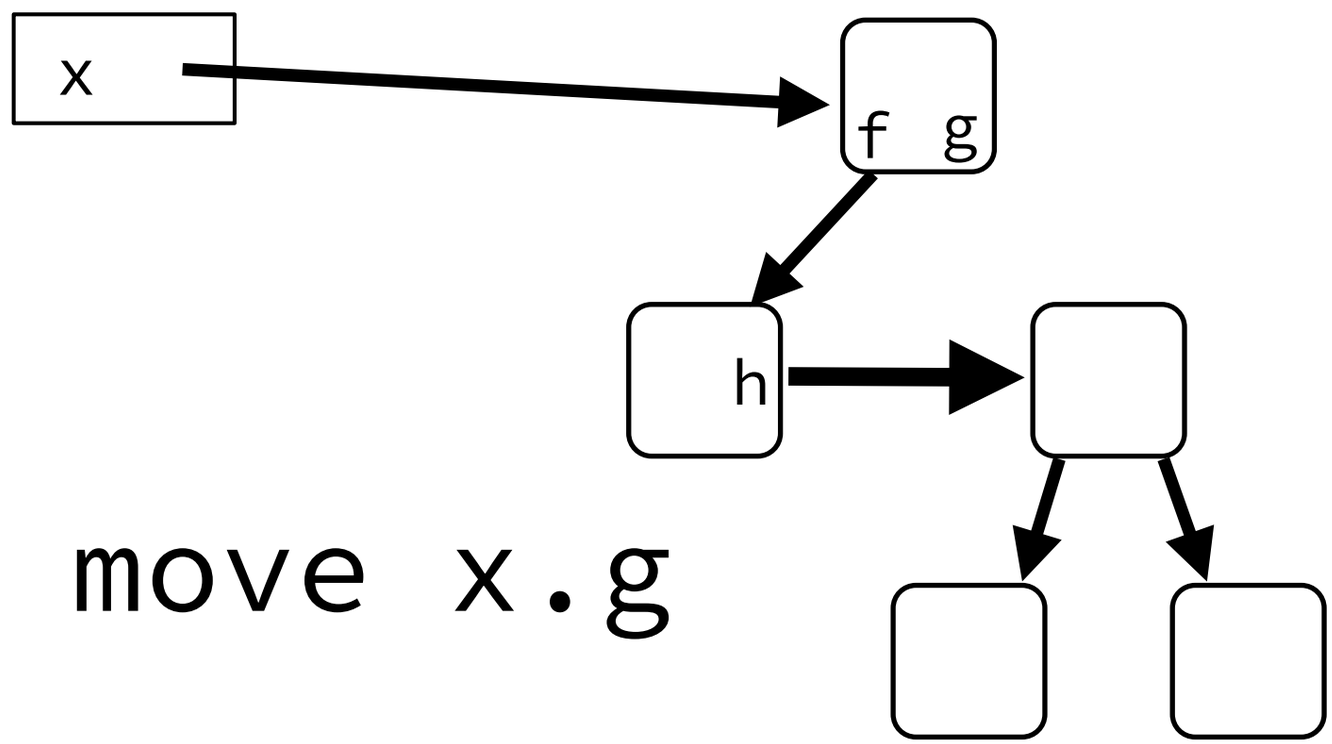
aka "Linear Types"
"Unique References"




```
x.f.h := move x.g
```

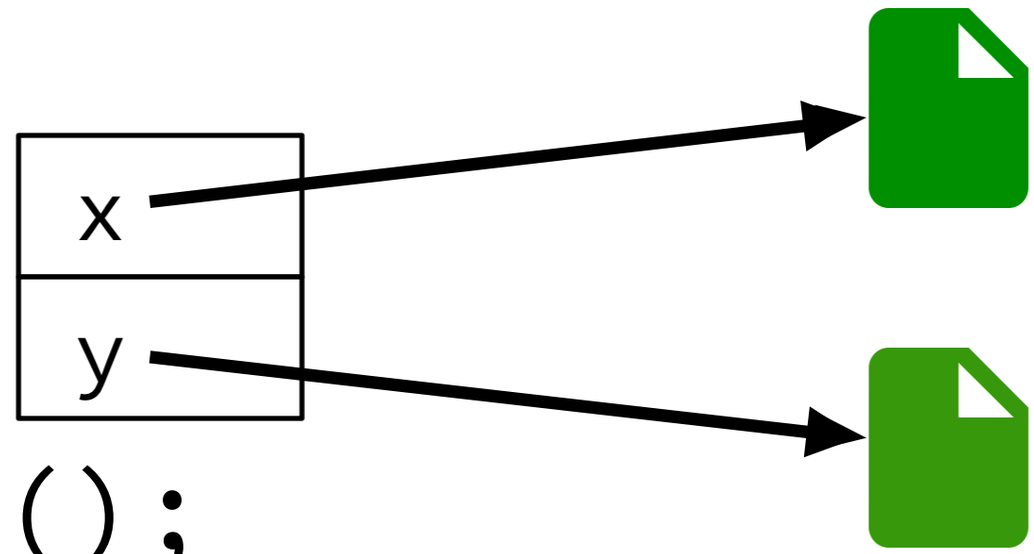
No Vars May Alias

aka "Linear Types"
"Unique References"

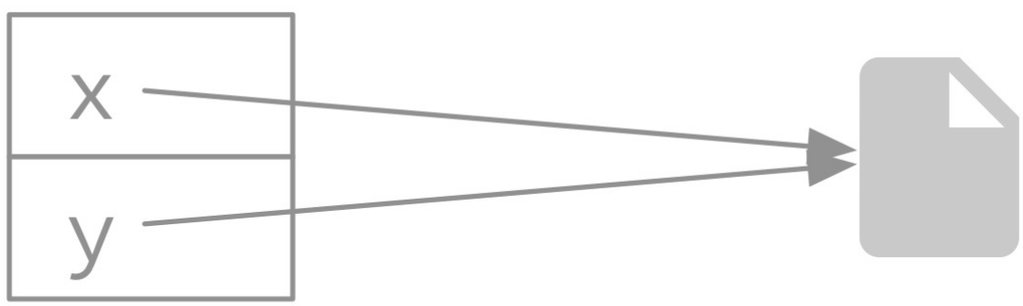



```
x.f.h := move x.g
```

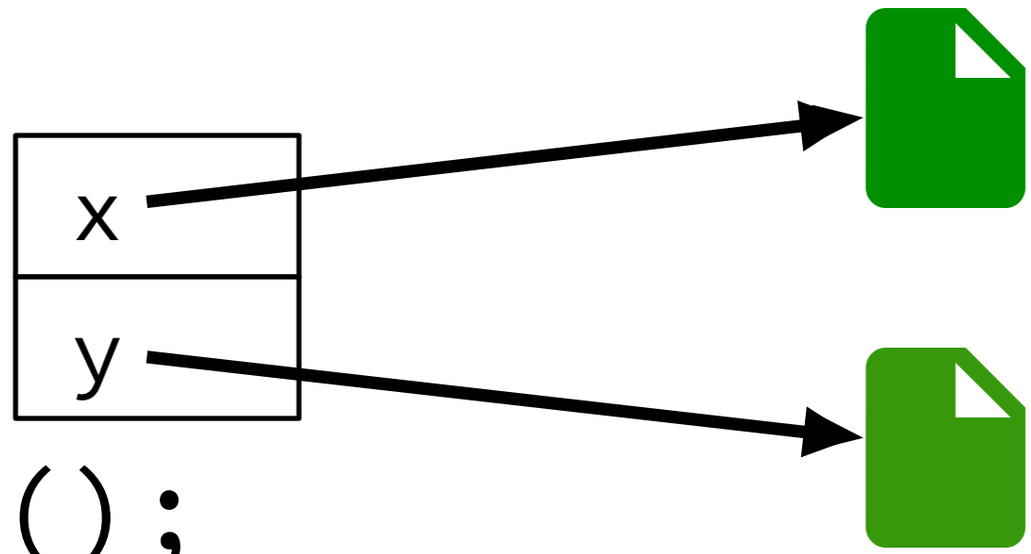
No Vars May Alias



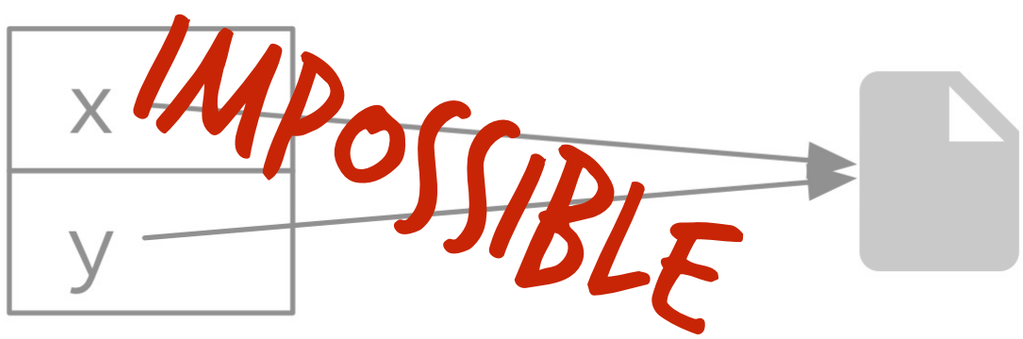

```
x.close();
y.read();
```

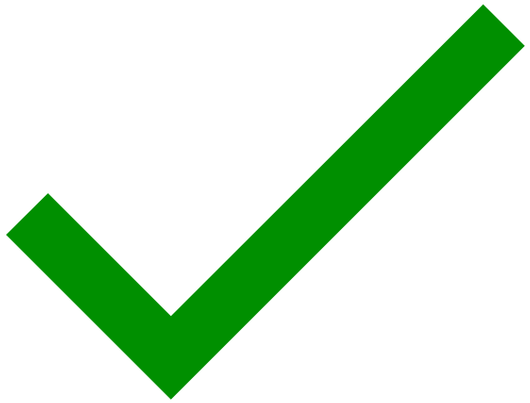


No Vars May Alias

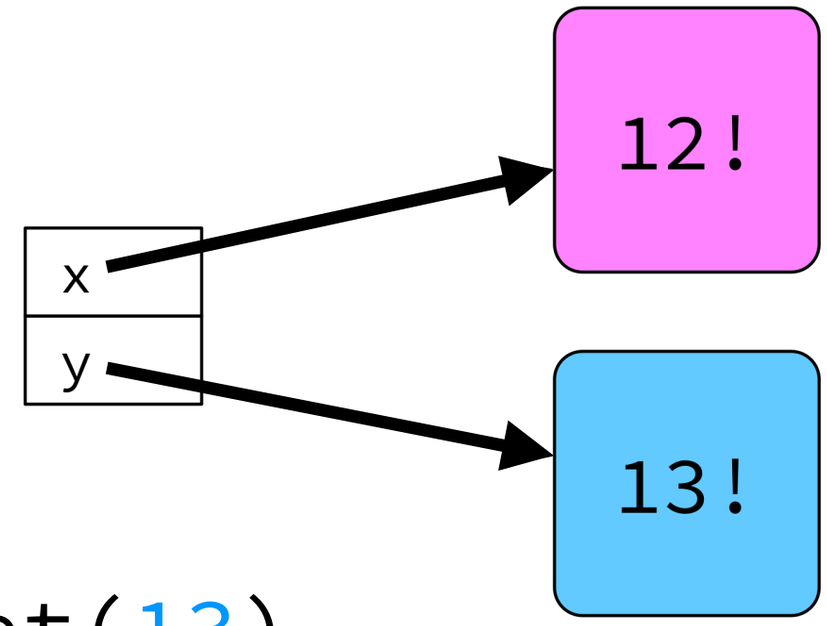



```
x.close();
y.read();
```

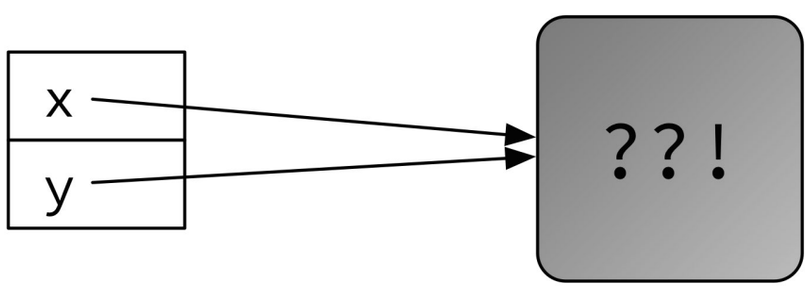




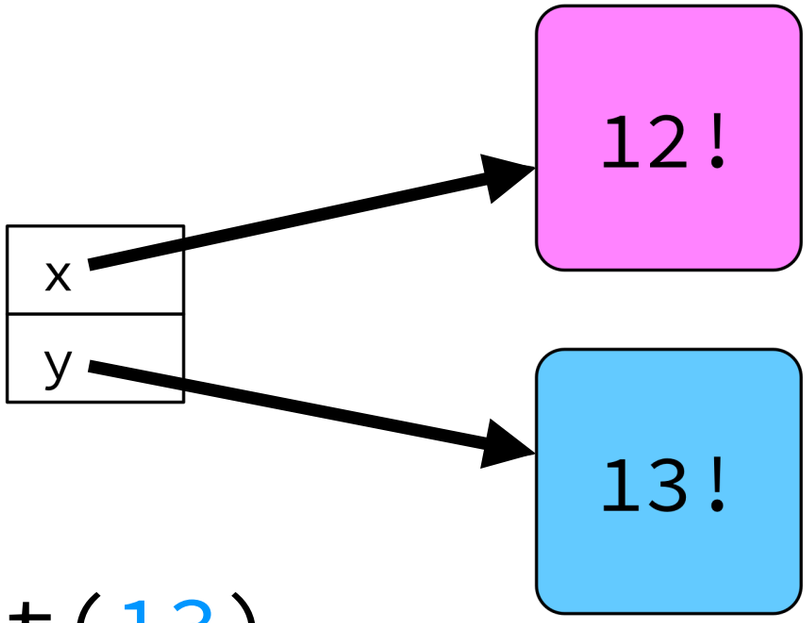
No Vars May Alias



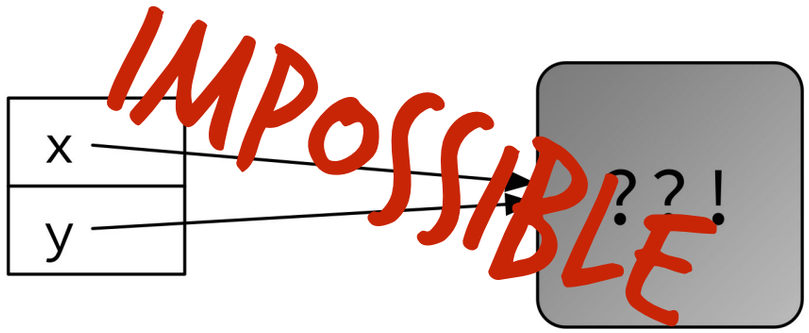

```
x.set(12) || y.set(13)
```

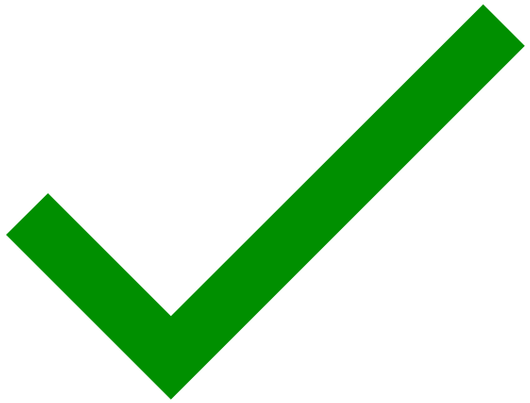


No Vars May Alias

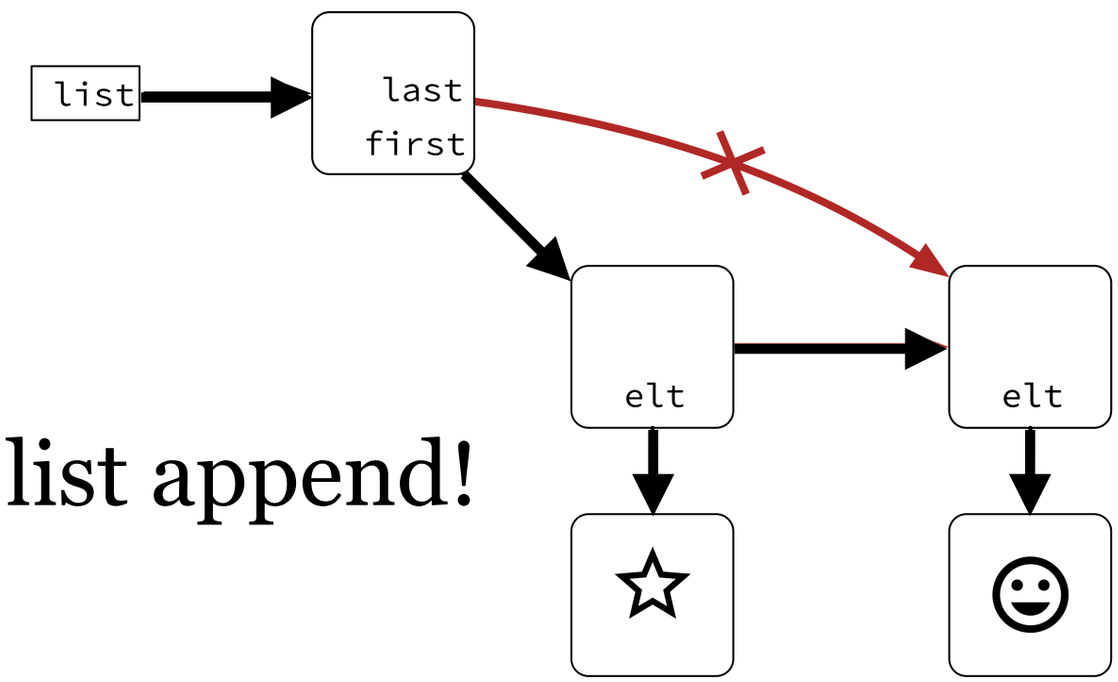



```
x.set(12) || y.set(13)
```



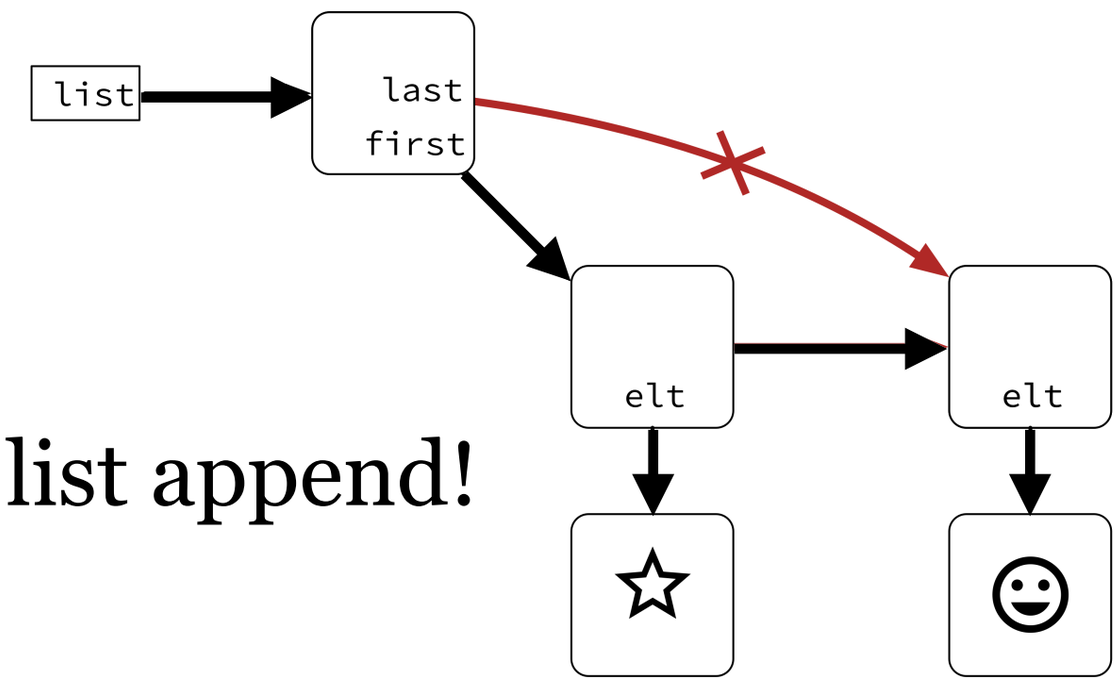


No Vars May Alias



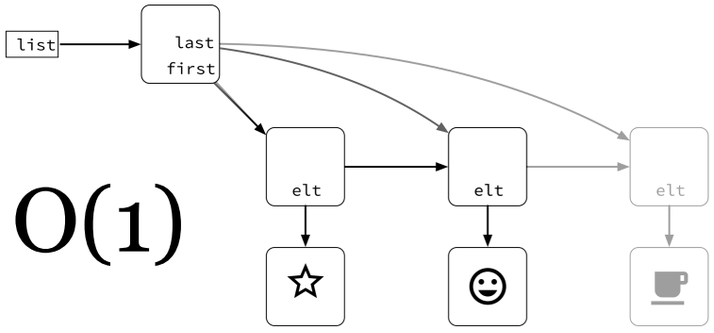
No efficient list append!

No Vars May Alias

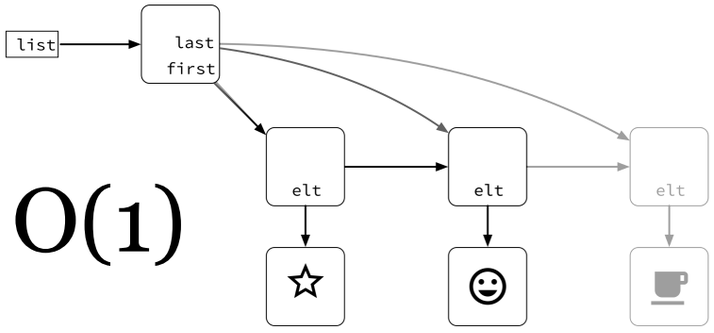


No efficient list append!

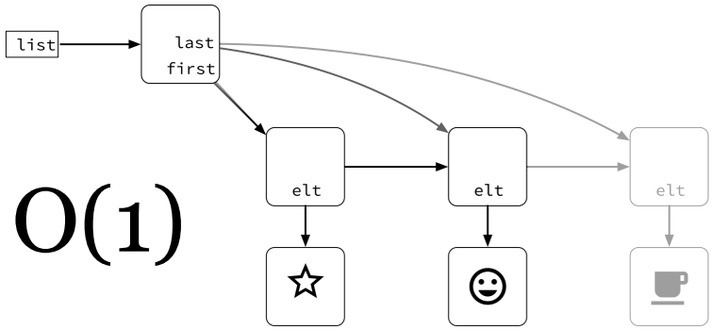
Aliased vs Unaliased

	Aliased	Unaliased
`x.close();` `y.read();`	✗	
`x.set(12) ||` `y.set(13)`	✗	
 O(1)	✓	

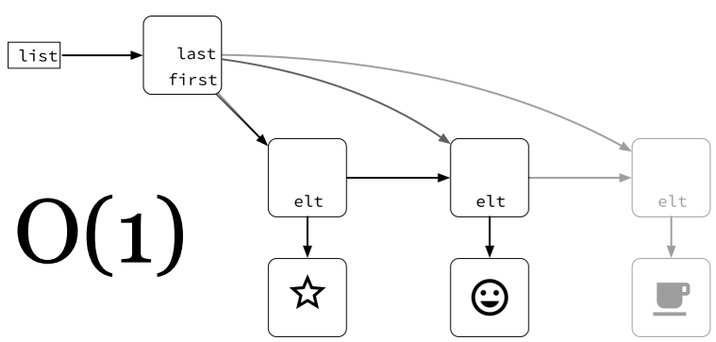
Aliased vs Unaliased

	Aliased	Unaliased
`x.close();` `y.read();`	✗	✓
`x.set(12) ||` `y.set(13)`	✗	
 O(1)	✓	

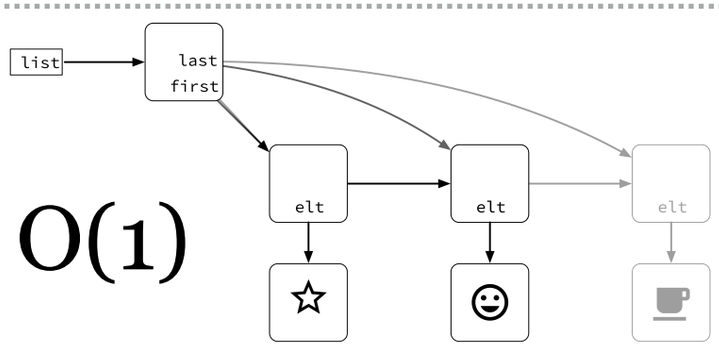
Aliased vs Unaliased

	Aliased	Unaliased
`x.close();` `y.read();`	✗	✓
`x.set(12) ||` `y.set(13)`	✗	✓
 O(1)	✓	

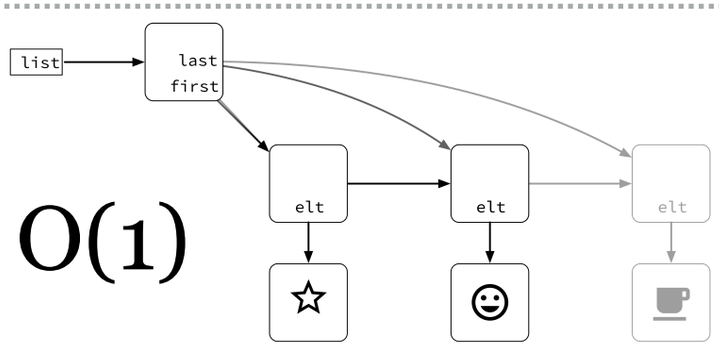
Aliased vs Unaliased

	Aliased	Unaliased
`x.close();` `y.read();`	✗	✓
`x.set(12) ||` `y.set(13)`	✗	✓
O(1) 	✓	✗

Contribution

	Aliased	?	Unaliased
`x.close();` `y.read();`	✗	✓	✓
`x.set(12) ||` `y.set(13)`	✗	✓	✓
 O(1)	✓	✓	✗

Contribution

	Aliased	Fine-Grained Aliasing	Unaliased
`x.close();` `y.read();`	✗	✓	✓
`x.set(12) ||` `y.set(13)`	✗	✓	✓
 O(1)	✓	✓	✗

CONTRIBUTION:

Fine-Grained Aliasing

Aliased Domains

```
def foo() {

}
```

unique

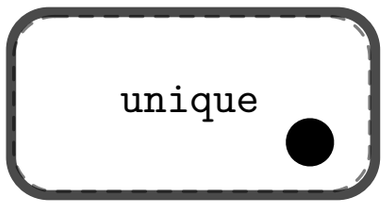
Aliased Domains


```
def foo() {
  let x : File#unique = fopen(…)
  in …

}
```

Aliased Domains

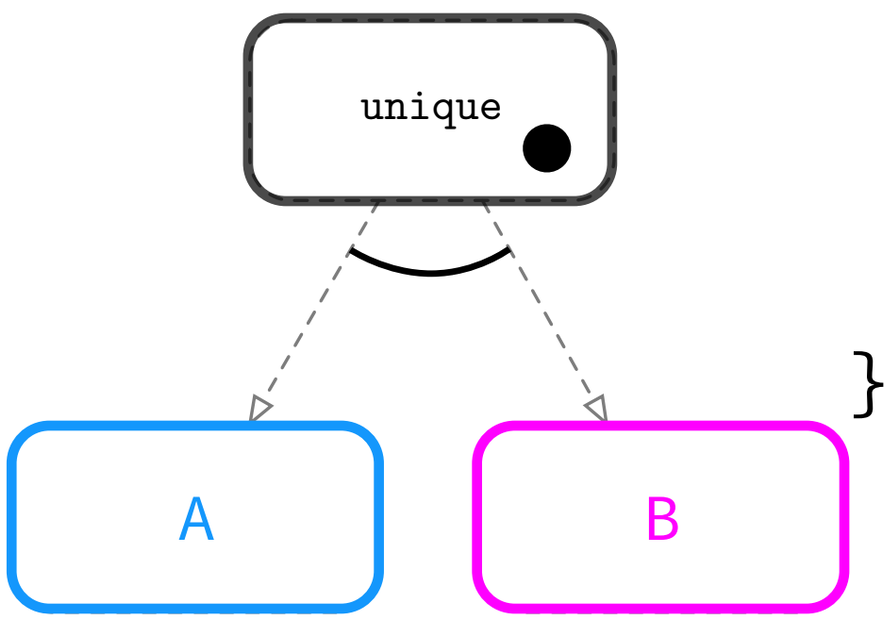

```
def foo() {
  let x : File#unique = fopen(…)
  in …

}
```

Aliased Domains



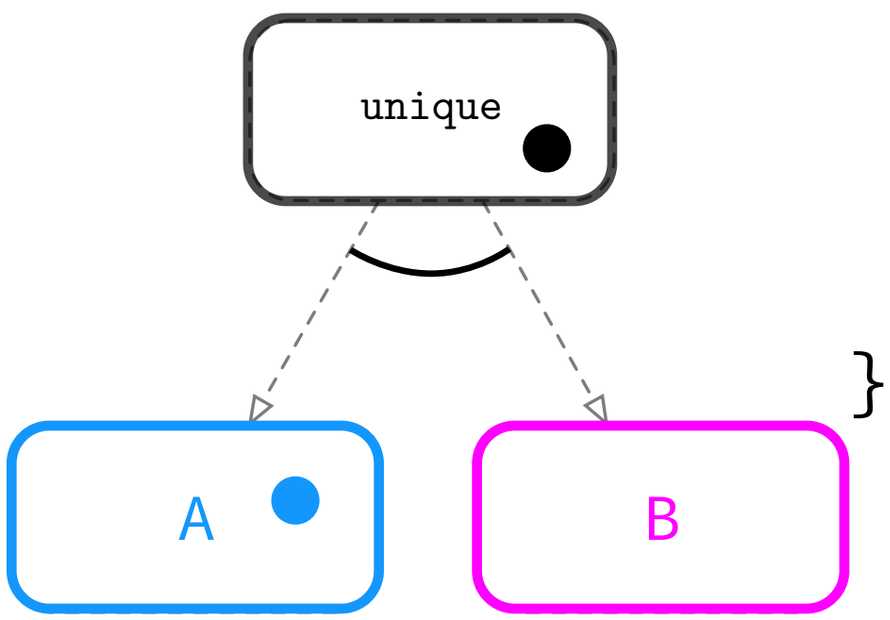

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
      …
    }
}
```

Aliased Domains



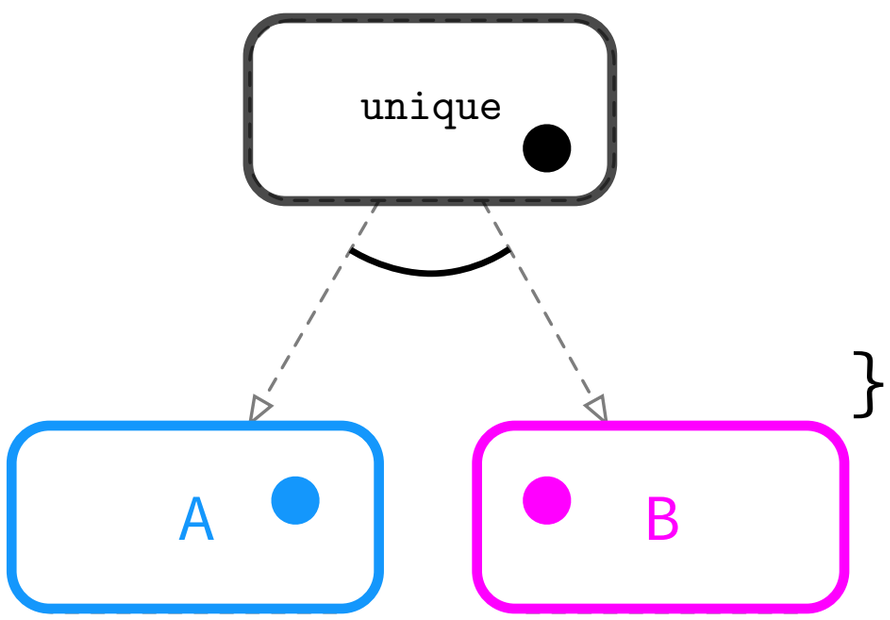

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

Aliased Domains



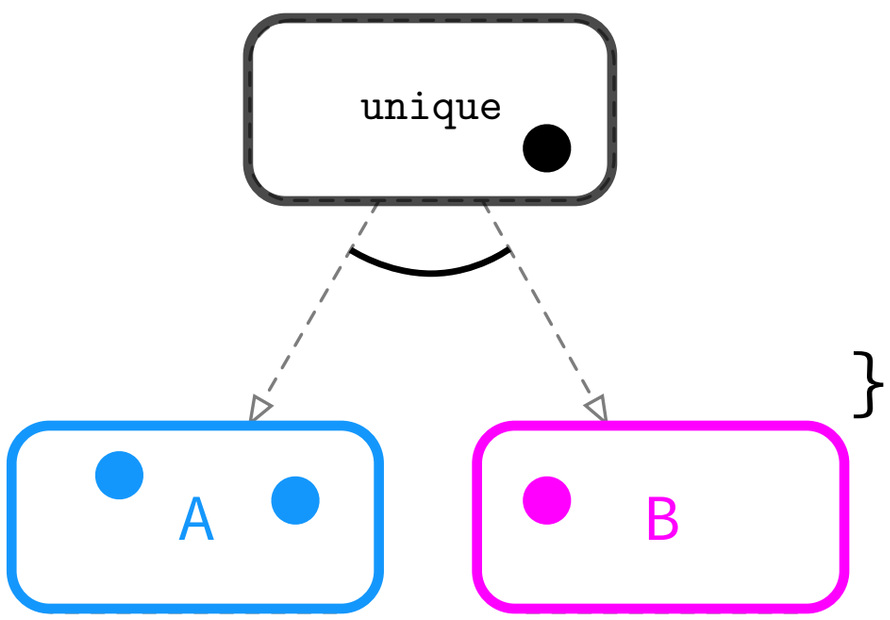

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

Aliased Domains



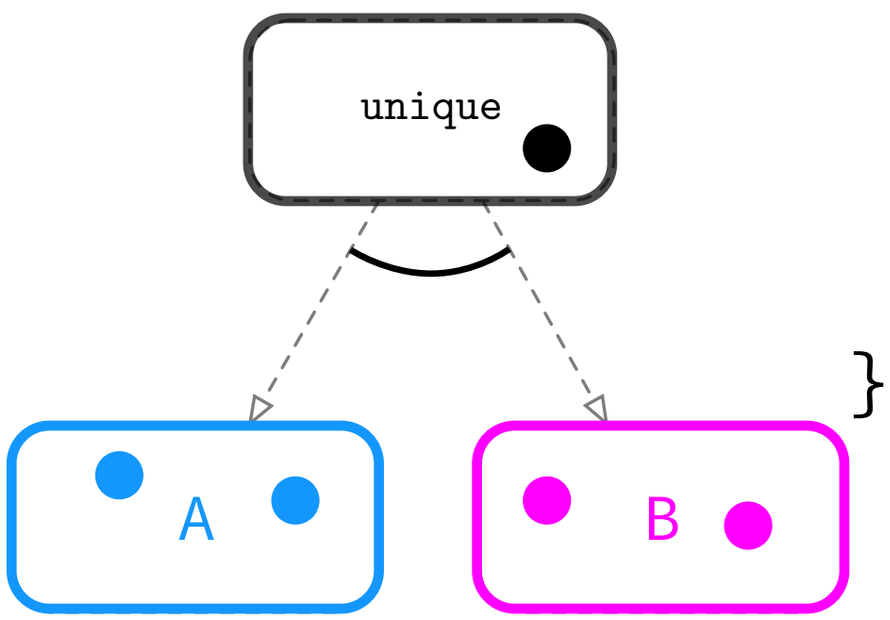

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

Aliased Domains




```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

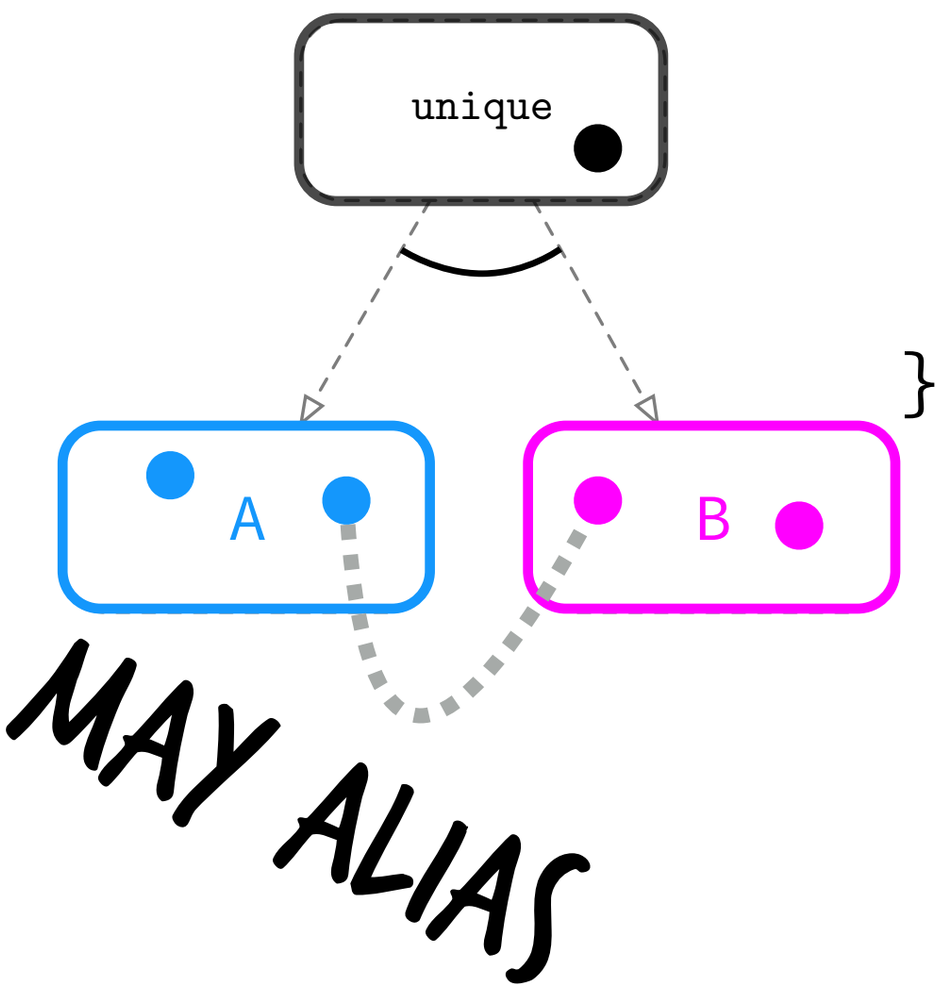
Aliased Domains



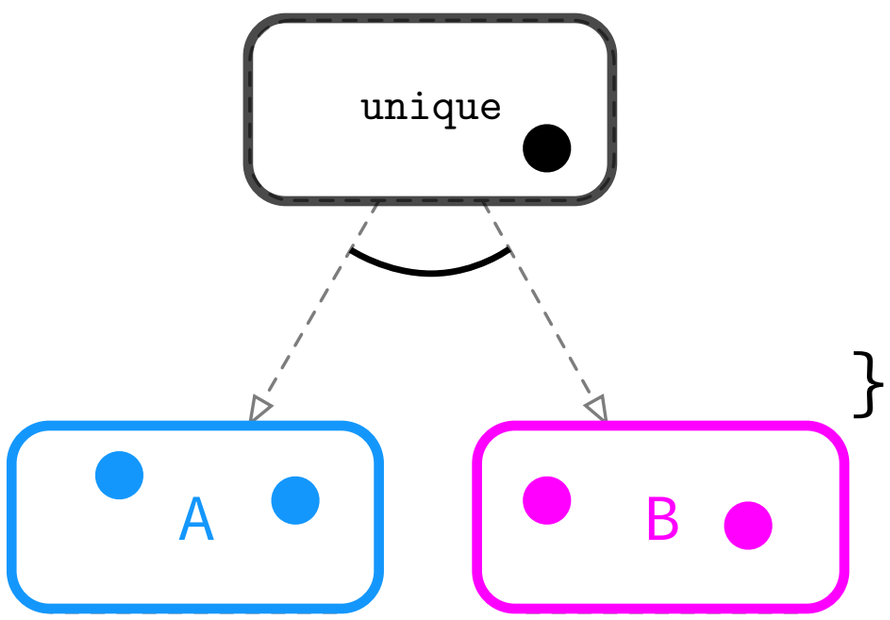

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

Aliased Domains

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

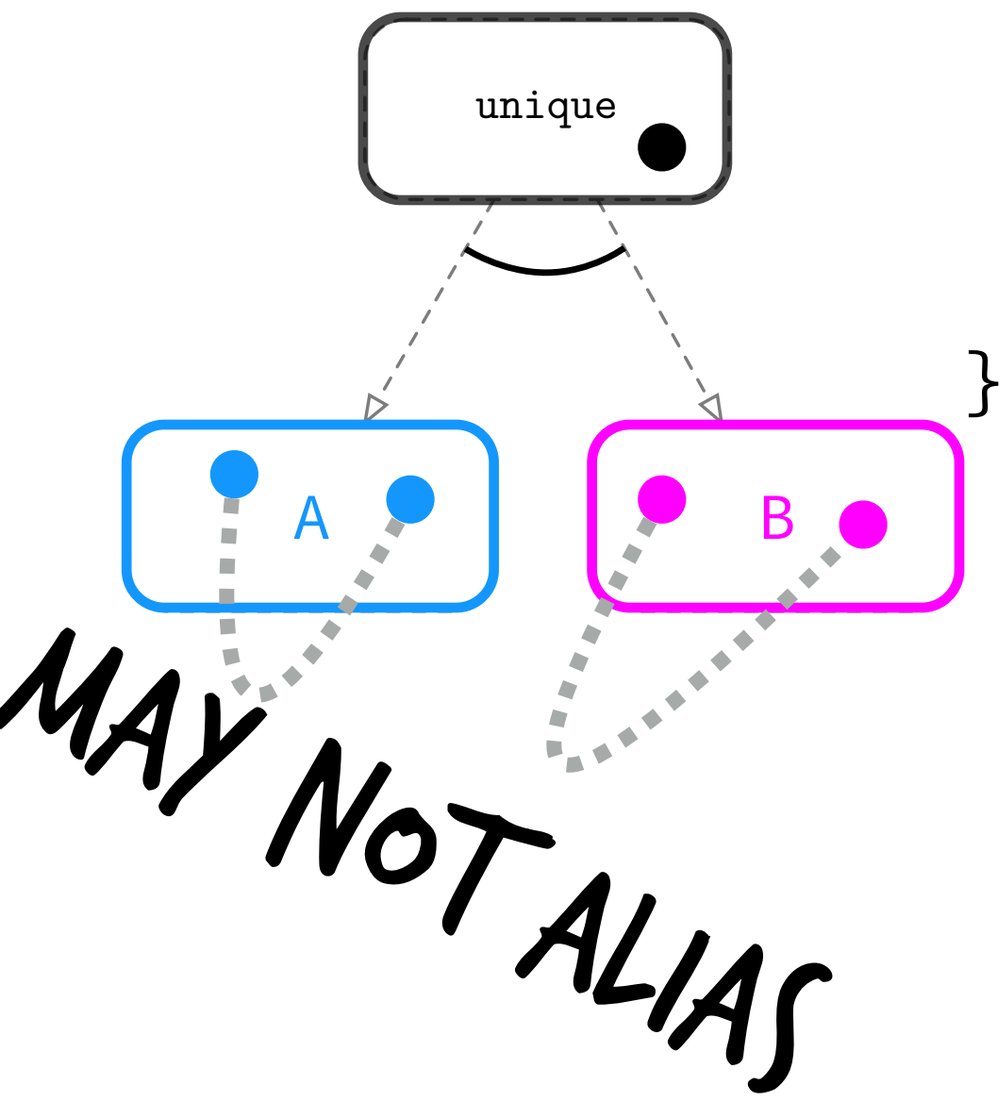


Aliased Domains




```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

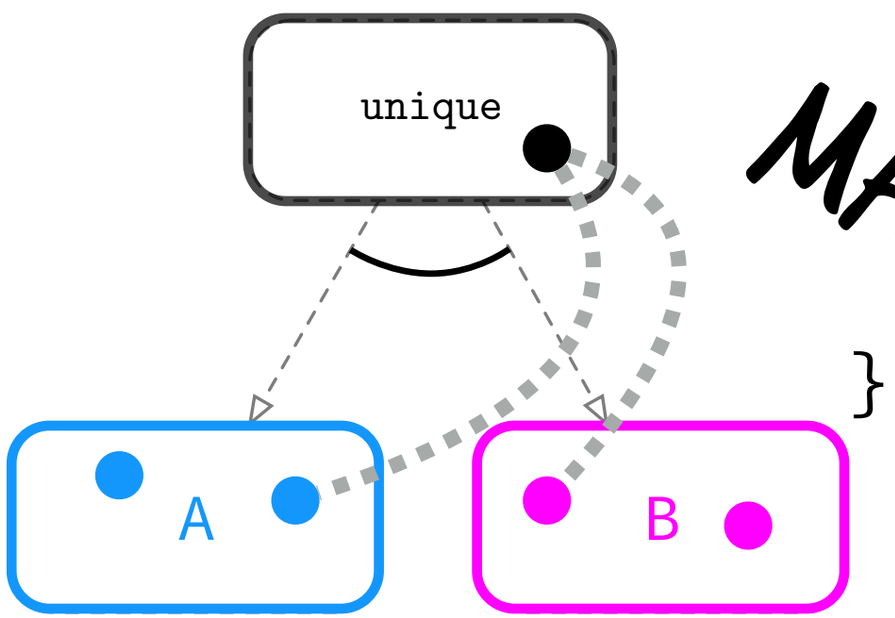
Aliased Domains




```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

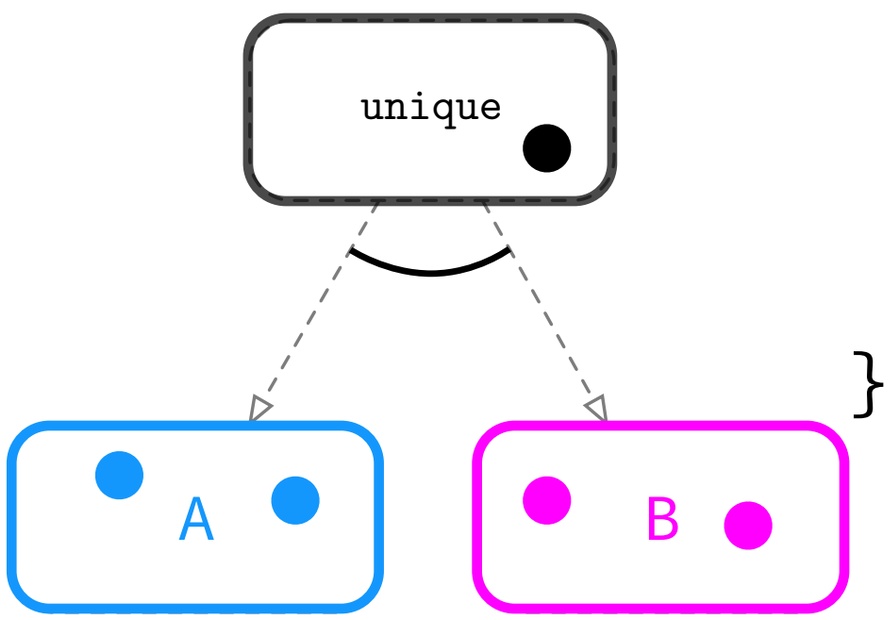
Aliased Domains

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
    …
  }
```



MAY NOT ALIAS

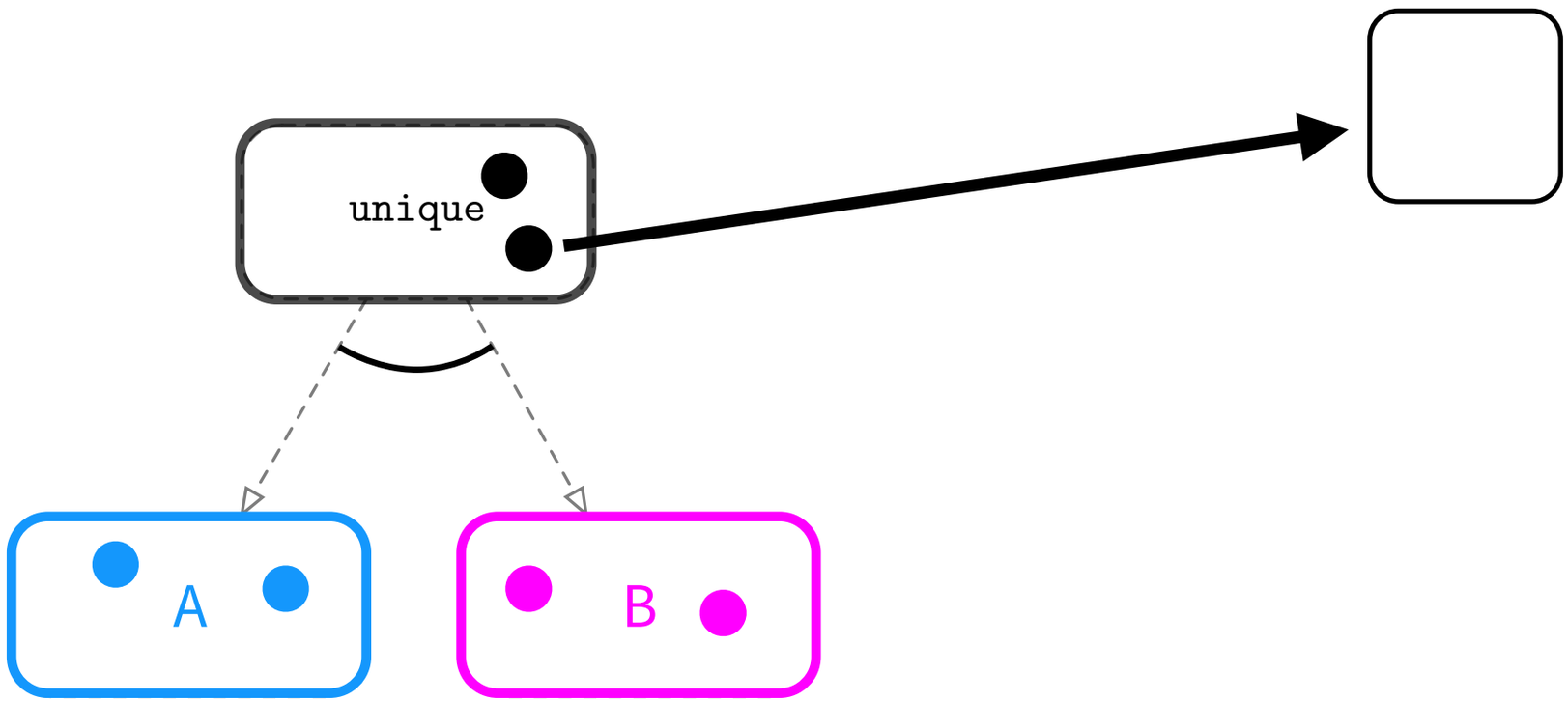
Aliased Domains



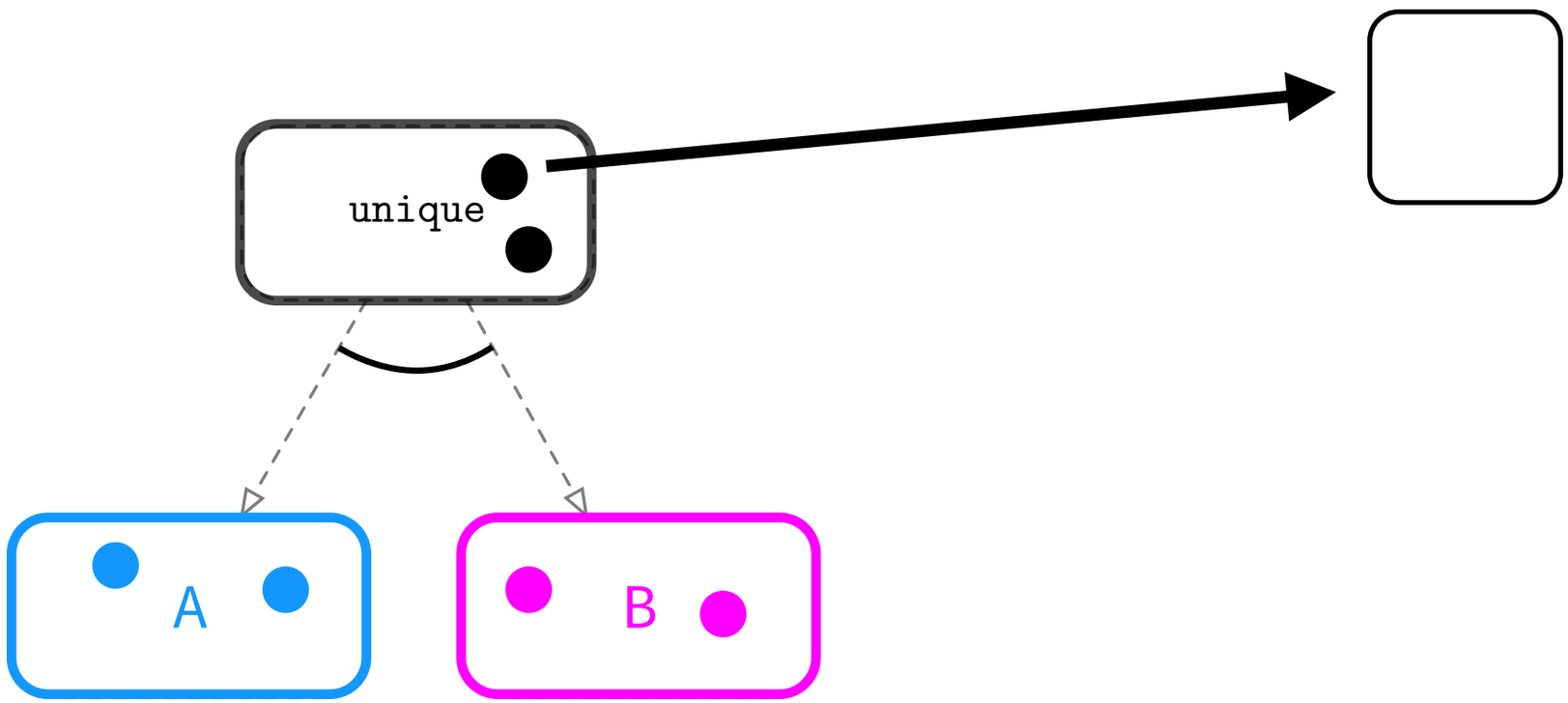

```
def foo() {
  let x : File#unique = fopen(…)
  in domain (A,B) extend unique {
        …
  }
}
```

LIMITED ALIASING!

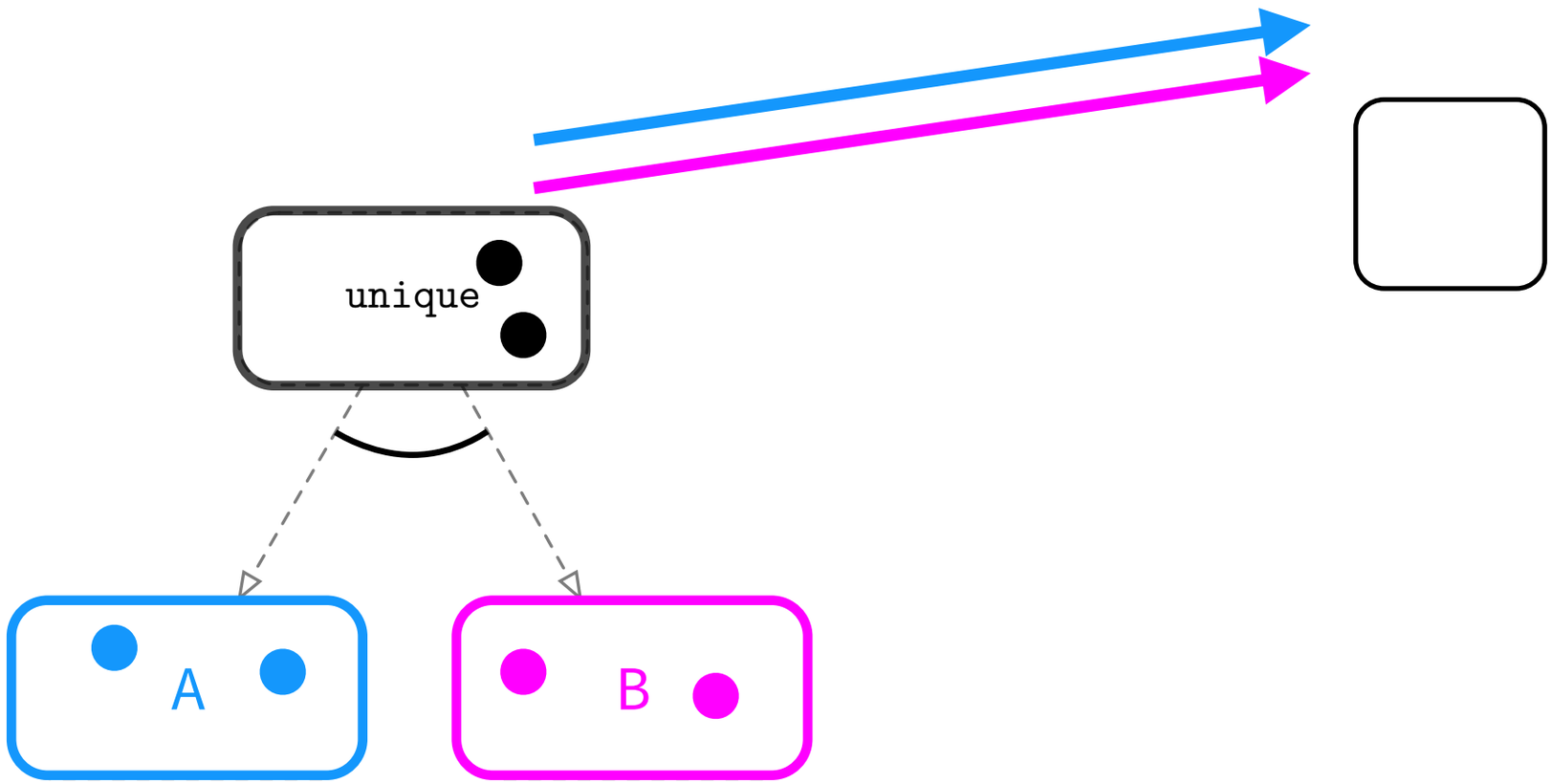
Aliased Domains



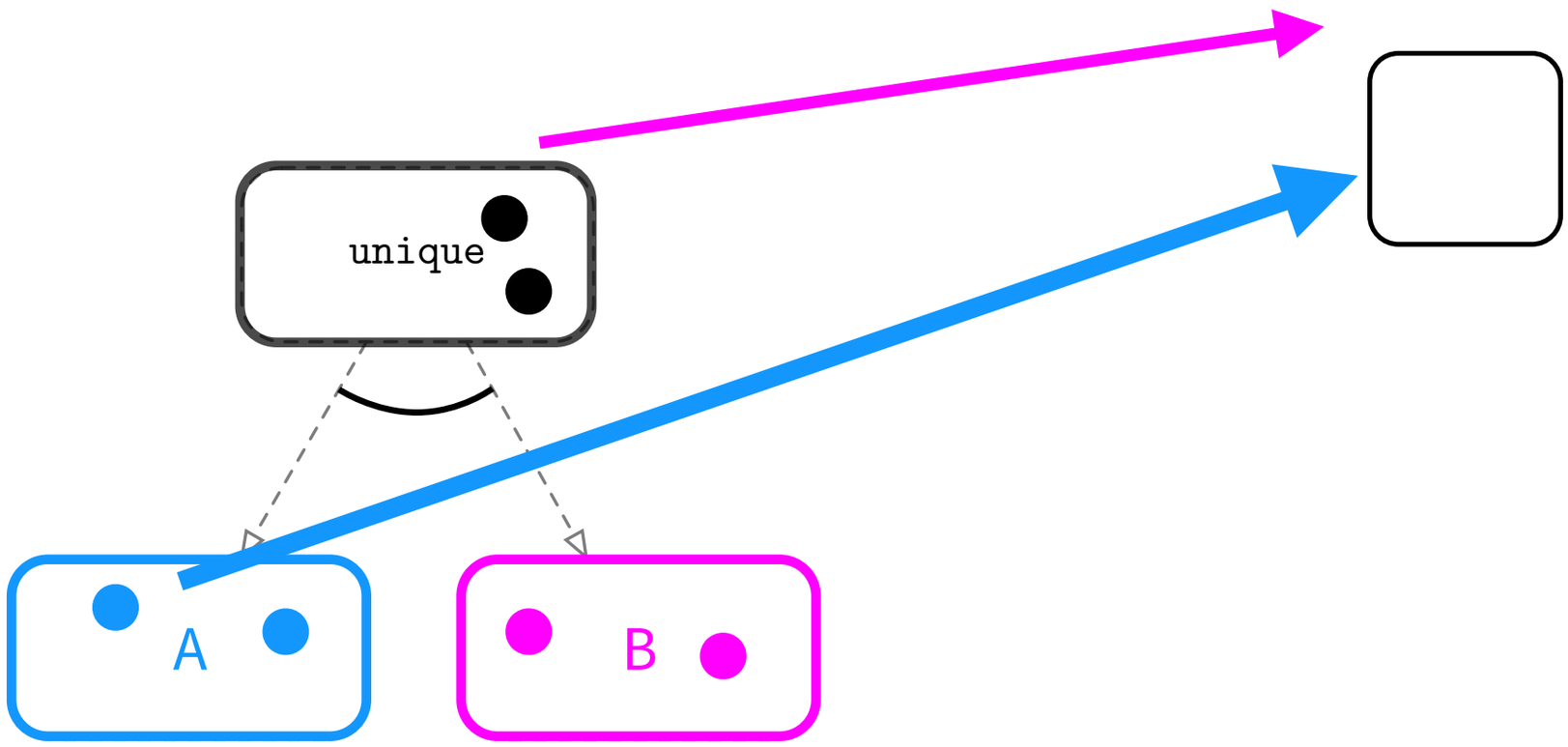
Aliased Domains



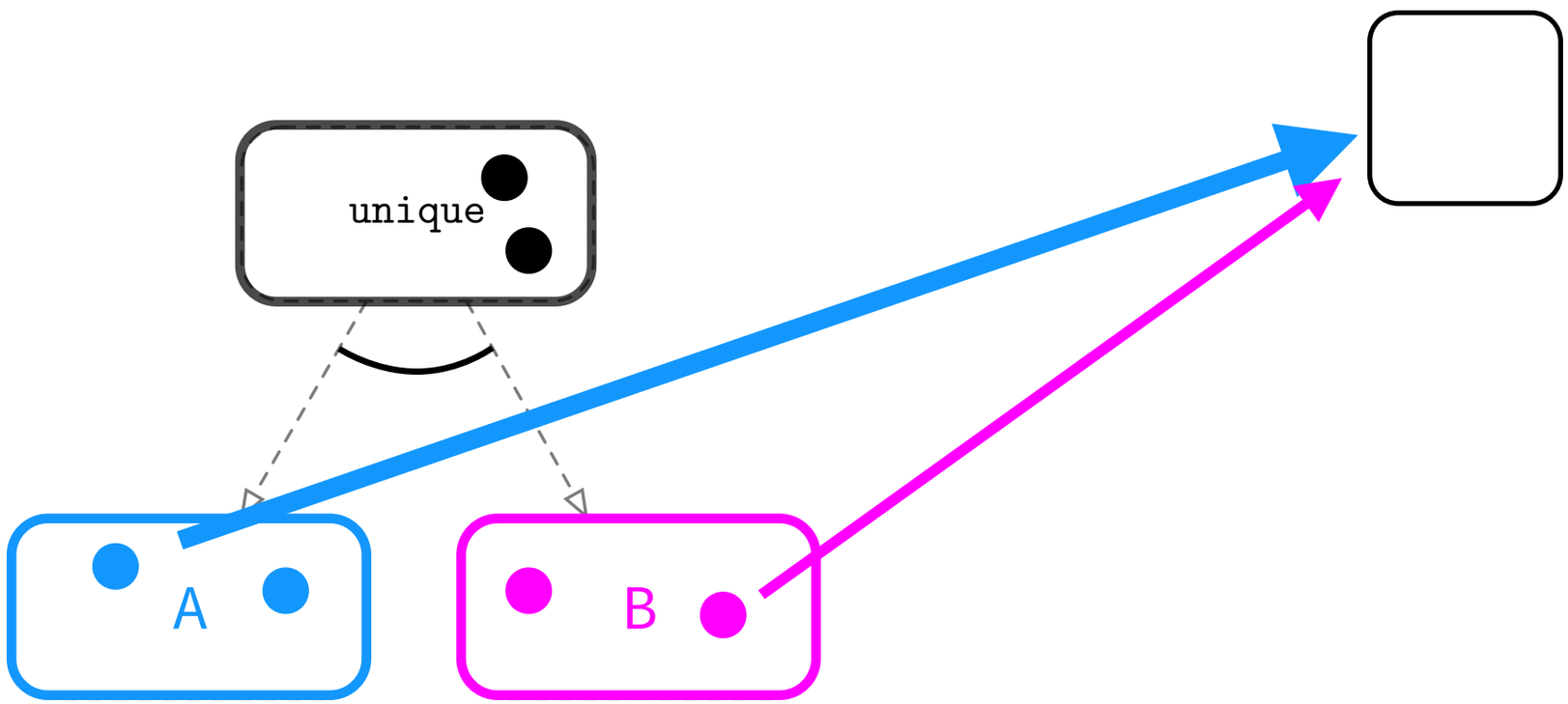
Aliased Domains



Aliased Domains



Aliased Domains



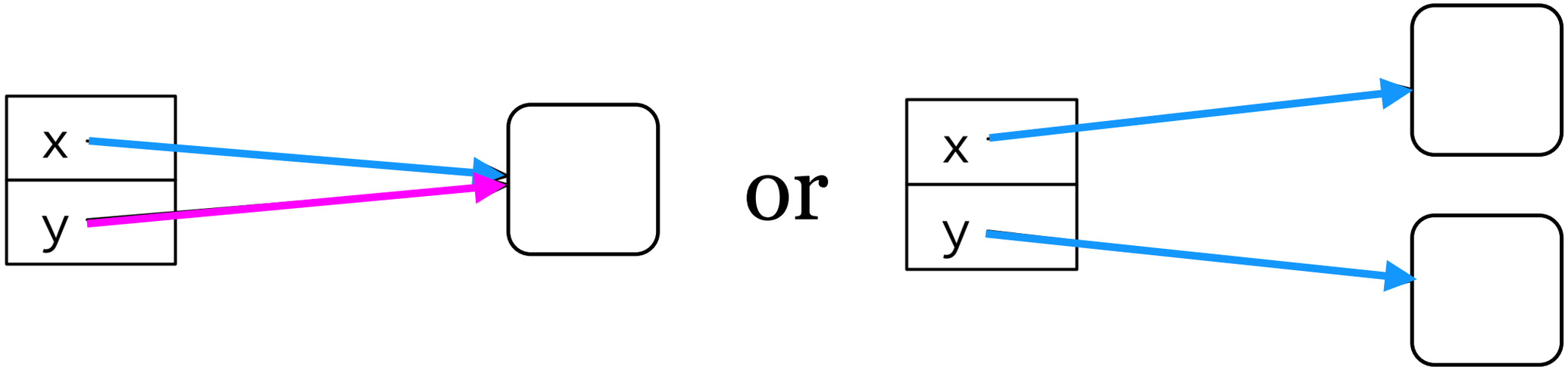
There´s more!

getting uniqueness back, avoiding destructive reads, domains that may NOT alias, ...

From May-not-Alias to Object Disjointness

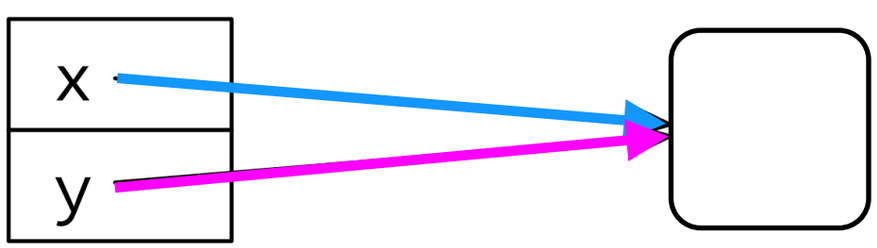
From May-not-Alias to Object Disjointness

We know whether two variables may alias.
But not yet about object disjointness!

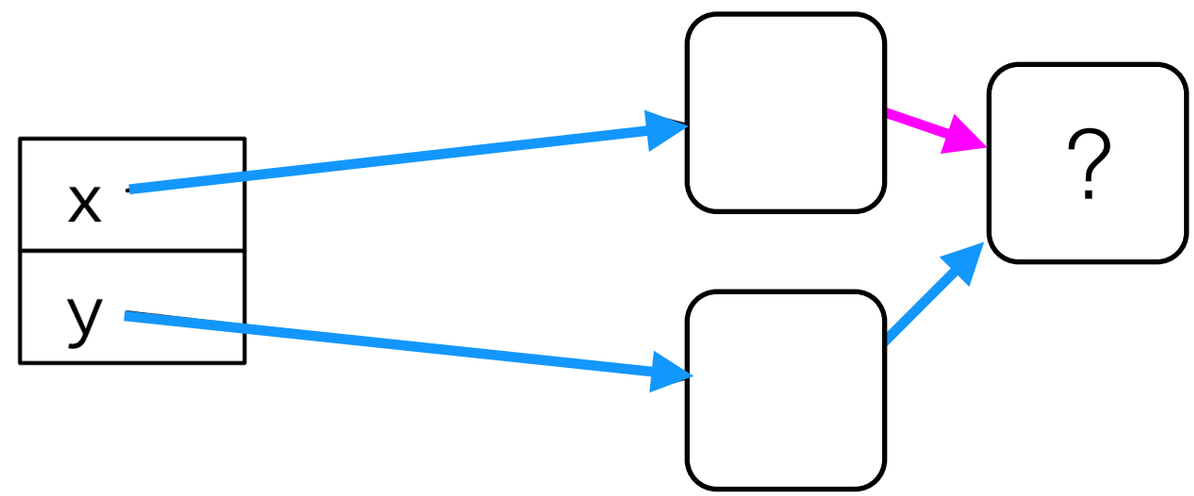


From May-not-Alias to Object Disjointness

We know whether two variables may alias.
But not yet about object disjointness!



or



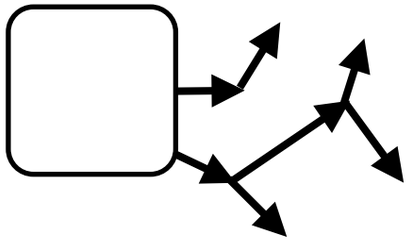
One Weird Trick for Object Disjointness

Code	*Runtime*

`class List {`
`  Obj #unique element;`
`  List#unique next;`
`}`
	`List`

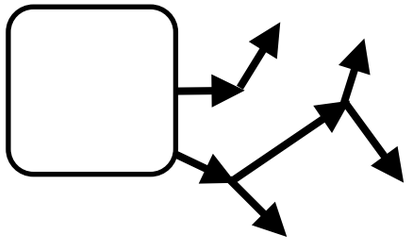
One Weird Trick for Object Disjointness

Code	*Runtime*

`class List {`
`  Obj #unique element;`
`  List#unique next;`
`}`
	`List`
	 

One Weird Trick for Object Disjointness

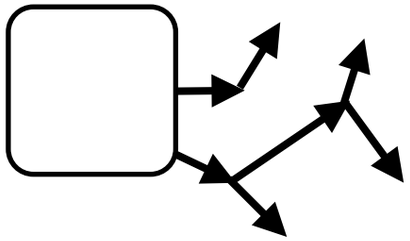
Code	*Runtime*

`class List {`
`  Obj #unique element;`
`  List#unique next;`
`}`
	`List`
	 

Let's add domain class parameters!

One Weird Trick for Object Disjointness

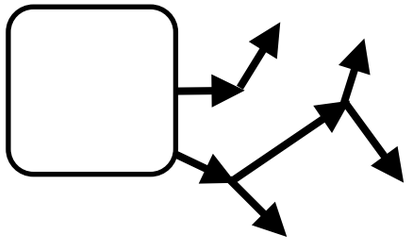
Code	*Runtime*

`class List[Data] {`
`  Obj #unique element;`
`  List#unique next;`
`}`
	`List`
	 

Let's add domain class parameters!

One Weird Trick for Object Disjointness

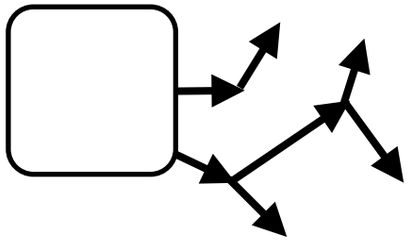
Code	*Runtime*

`class List[Data] {`
`  Obj #Data  element;`
`  List#unique next;`
`}`
	`List`
	 

Let's add domain class parameters!

One Weird Trick for Object Disjointness

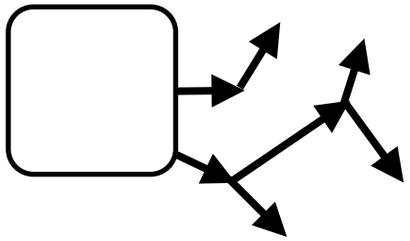
Code	*Runtime*

`class List[Data] {`
`  Obj #Data  element;`
`  List[Data]#unique next;`
`}`
	`List`
	 

Let's add domain class parameters!

One Weird Trick for Object Disjointness

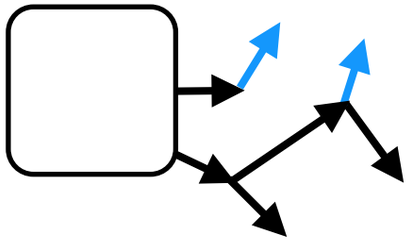
Code	*Runtime*

`class List[Data] {`
`  Obj #Data  element;`
`  List[Data]#unique next;`
`}`
	`List[■]`
	 

Let's add domain class parameters!

One Weird Trick for Object Disjointness

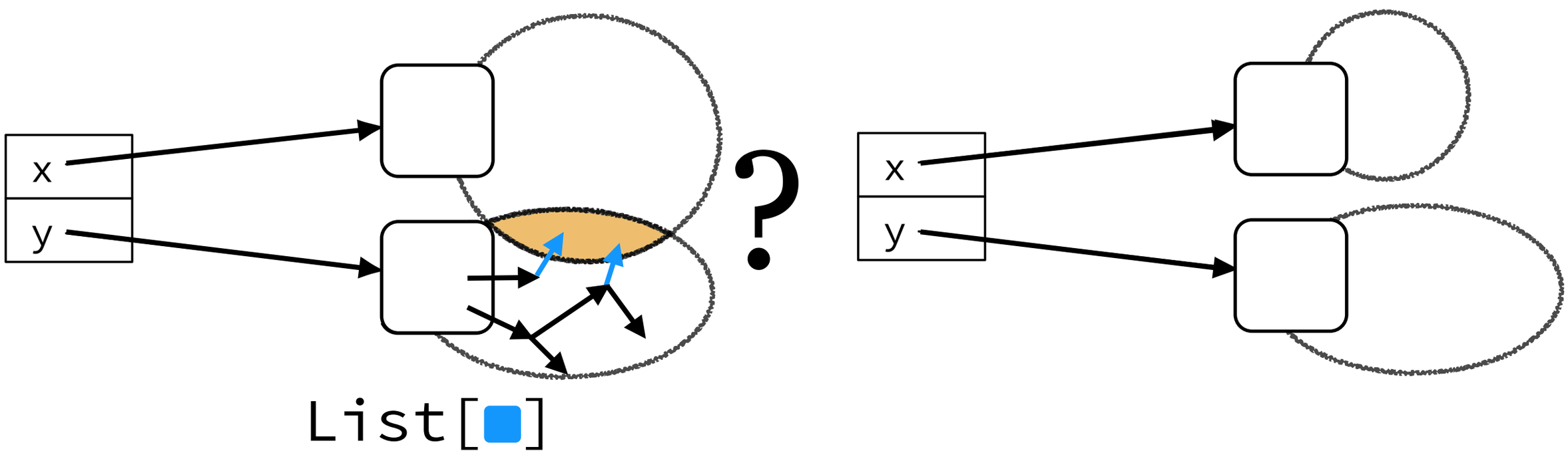
Code	*Runtime*

`class List[Data] {`
`  Obj #Data  element;`
`  List[Data]#unique next;`
`}`
	`List[■]`
	 

Let's add domain class parameters!

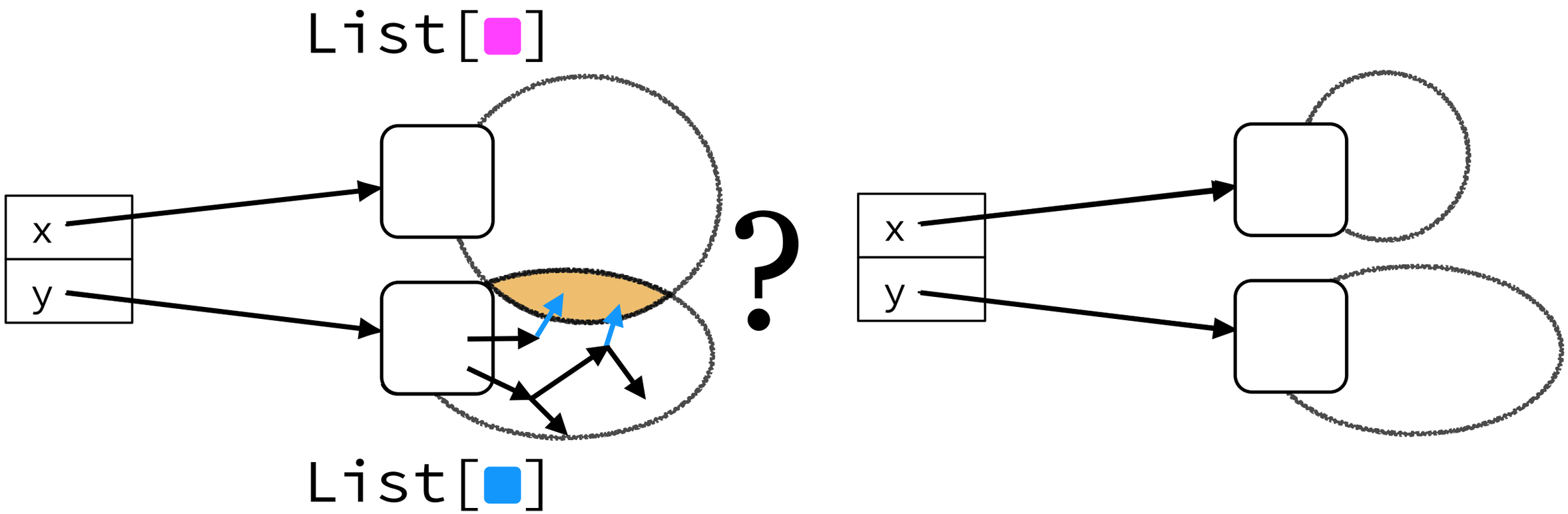
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



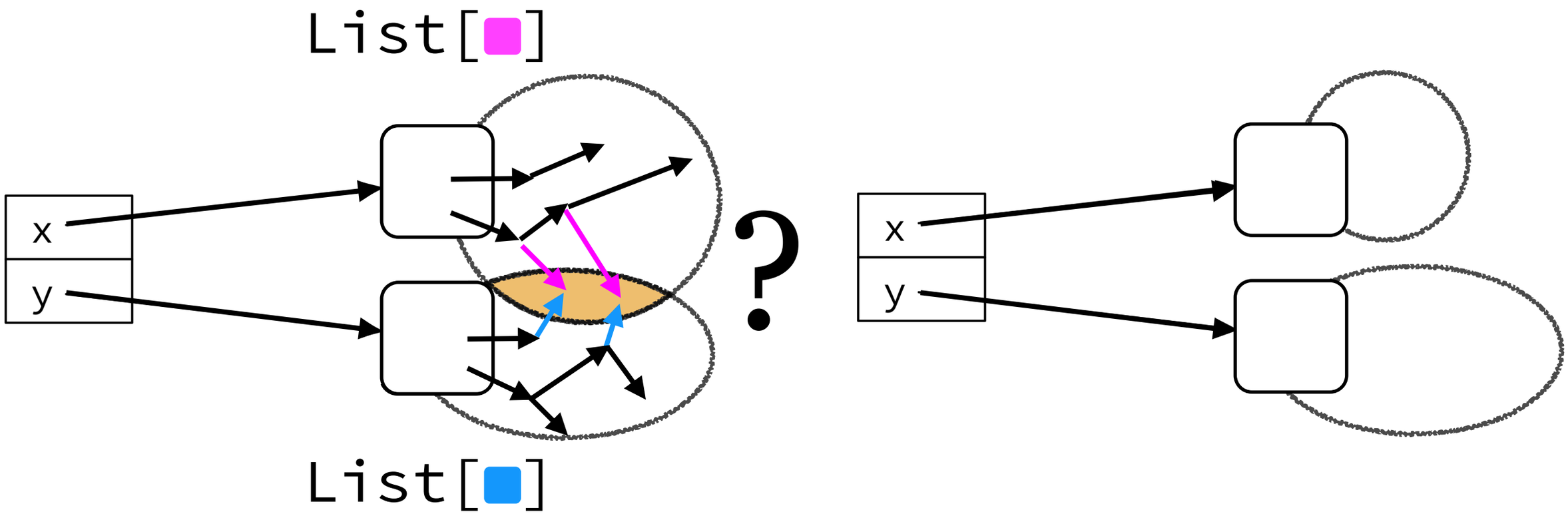
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



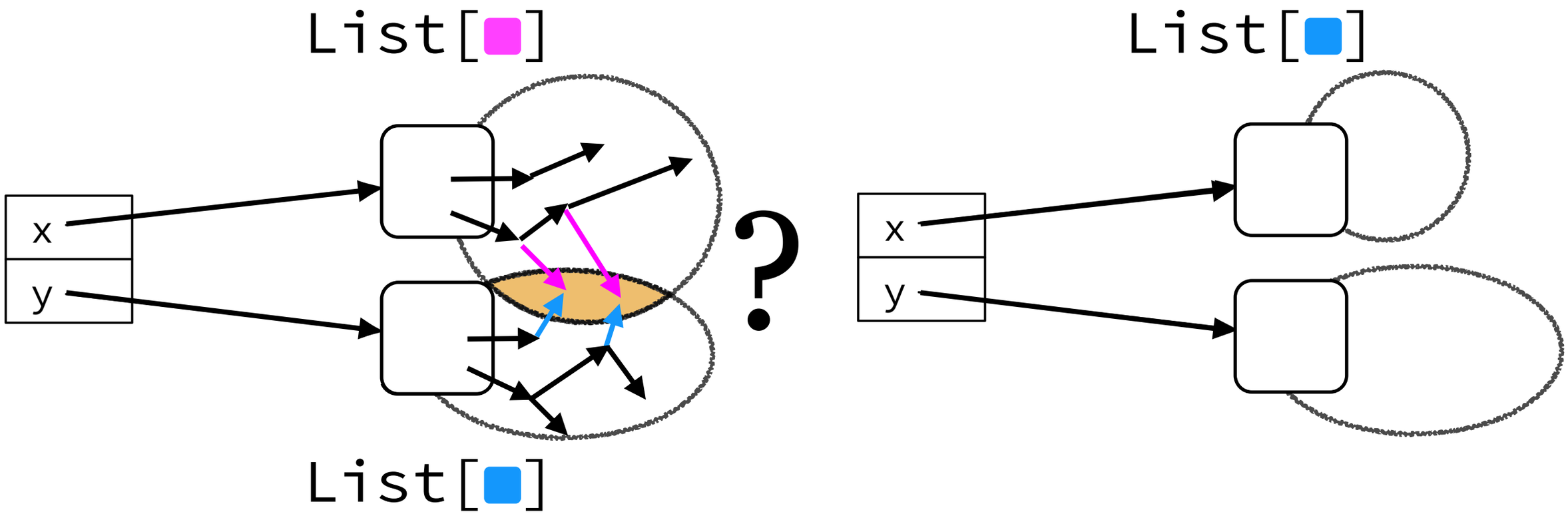
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



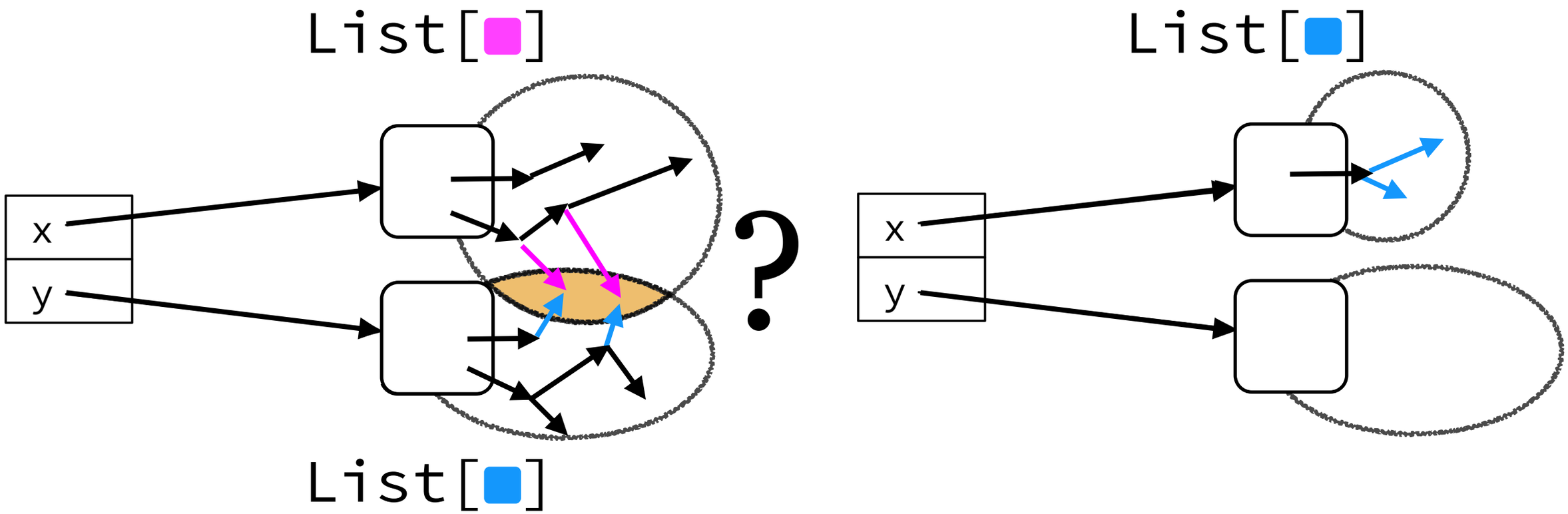
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



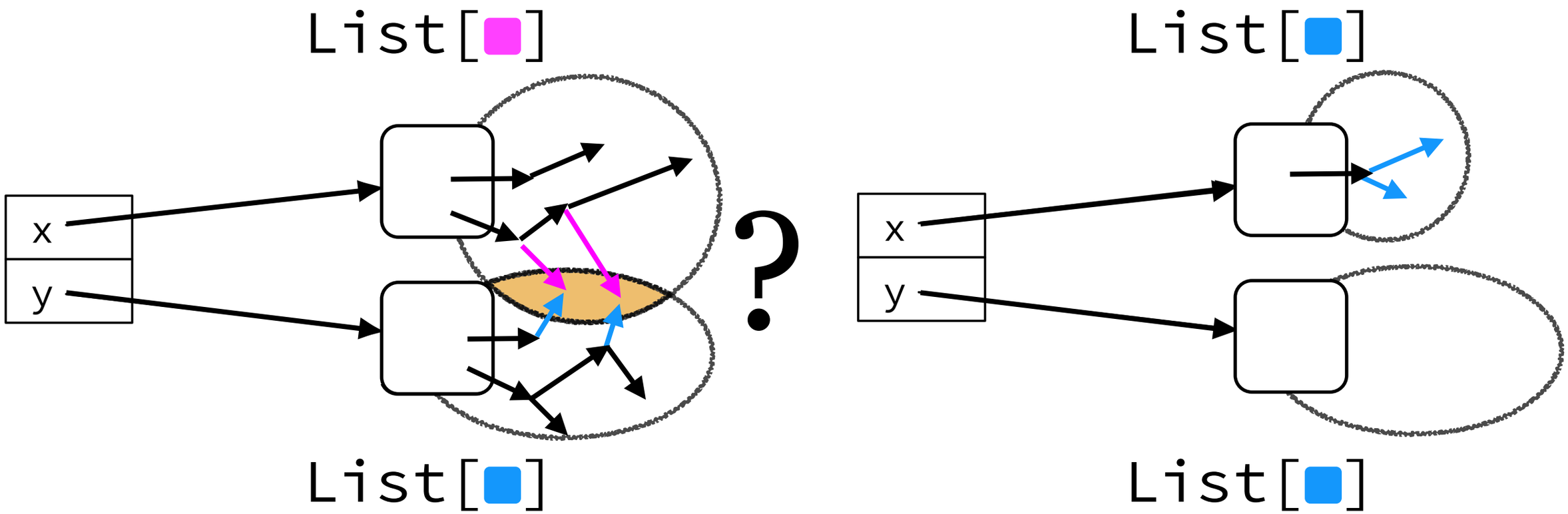
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



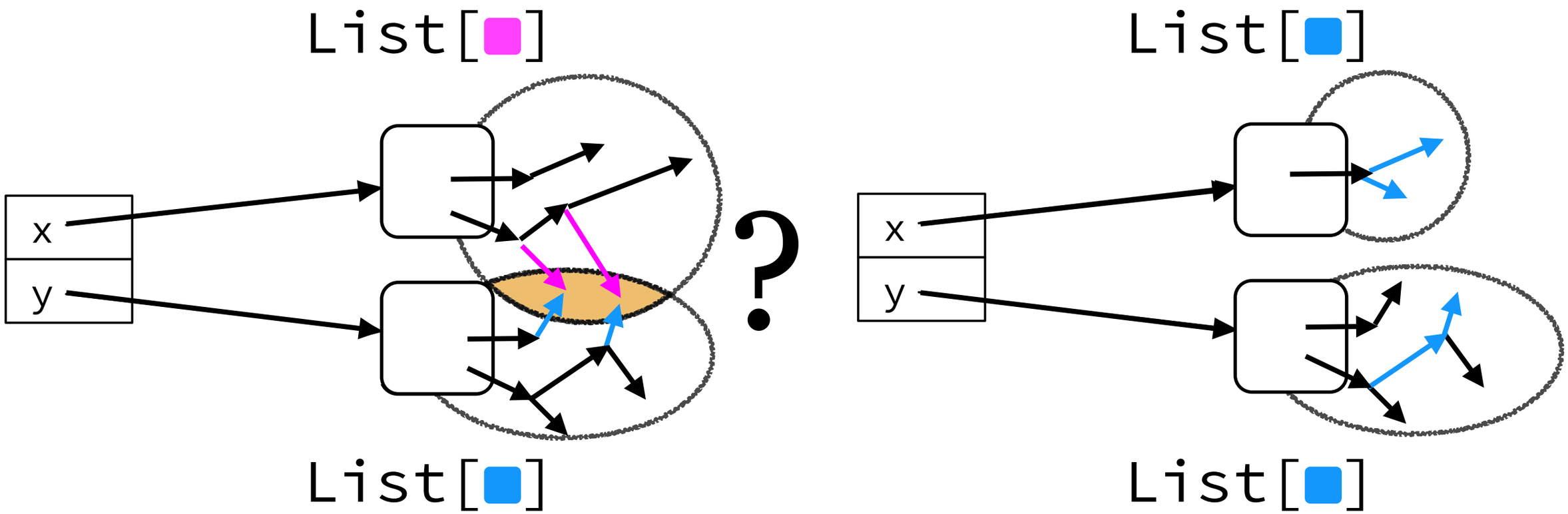
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



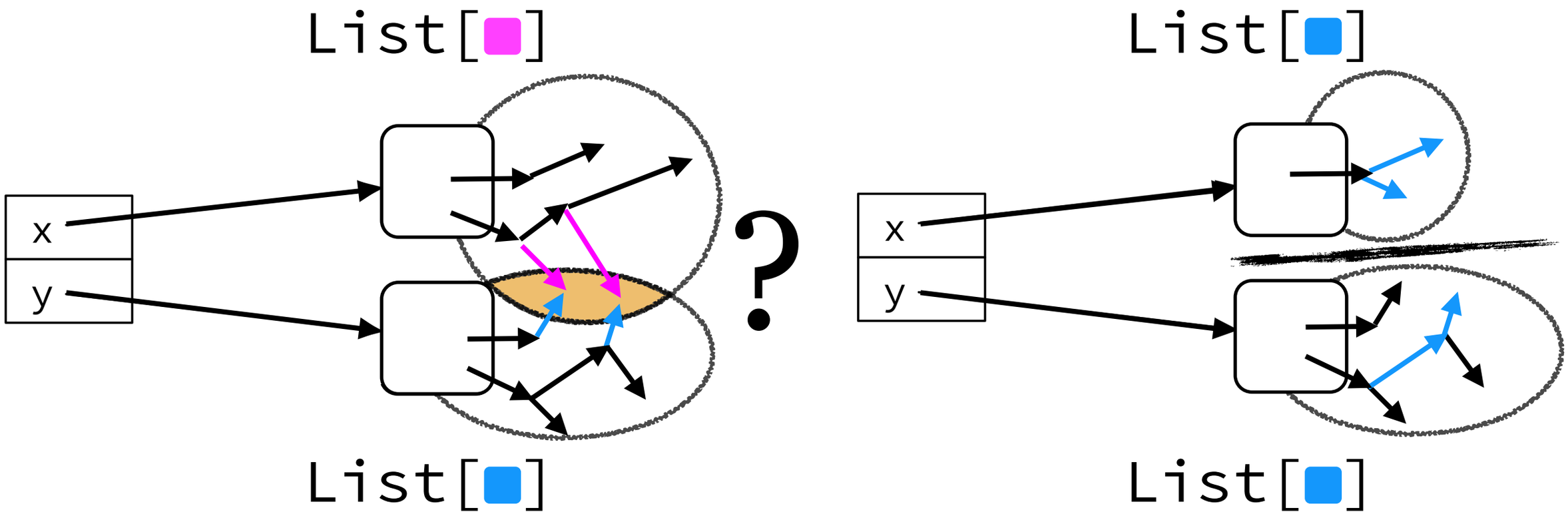
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



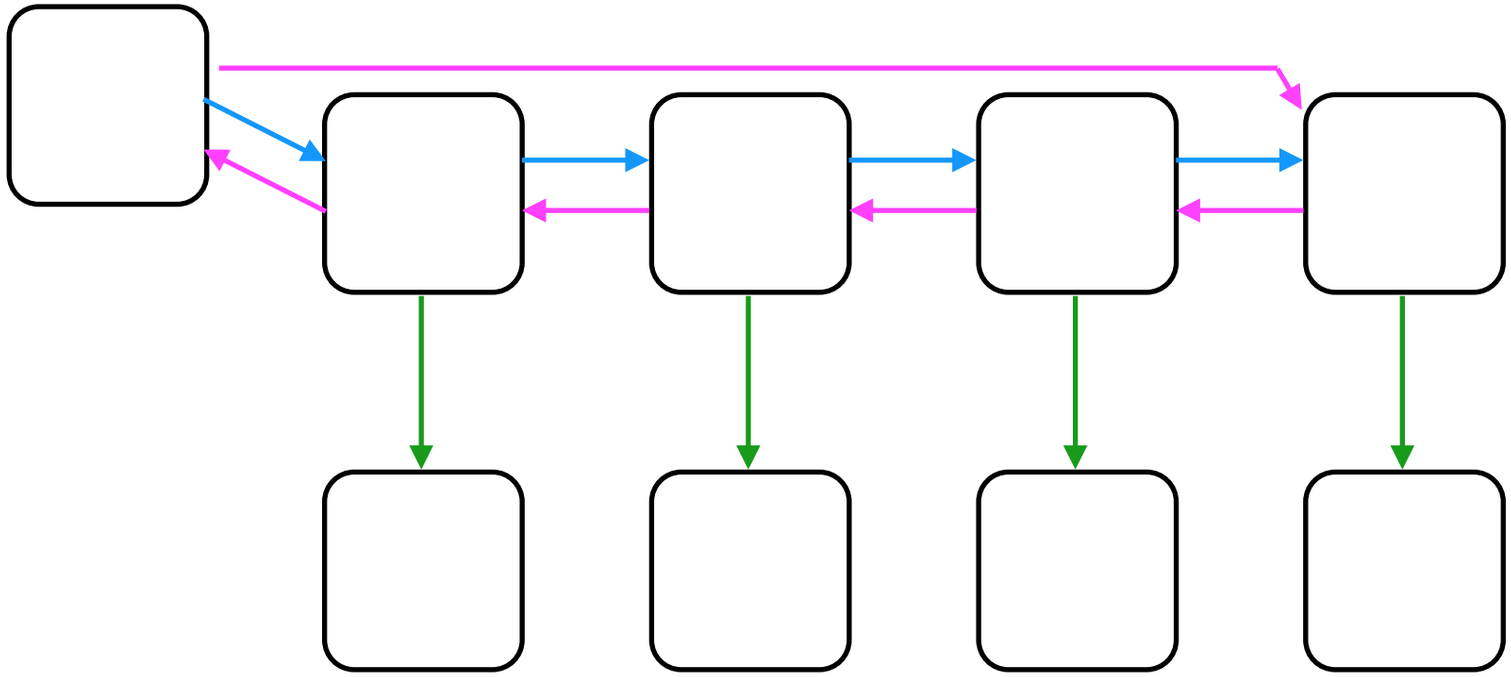
From May-not-Alias to Object Disjointness

3) We can tell whether two *objects* may share any data:



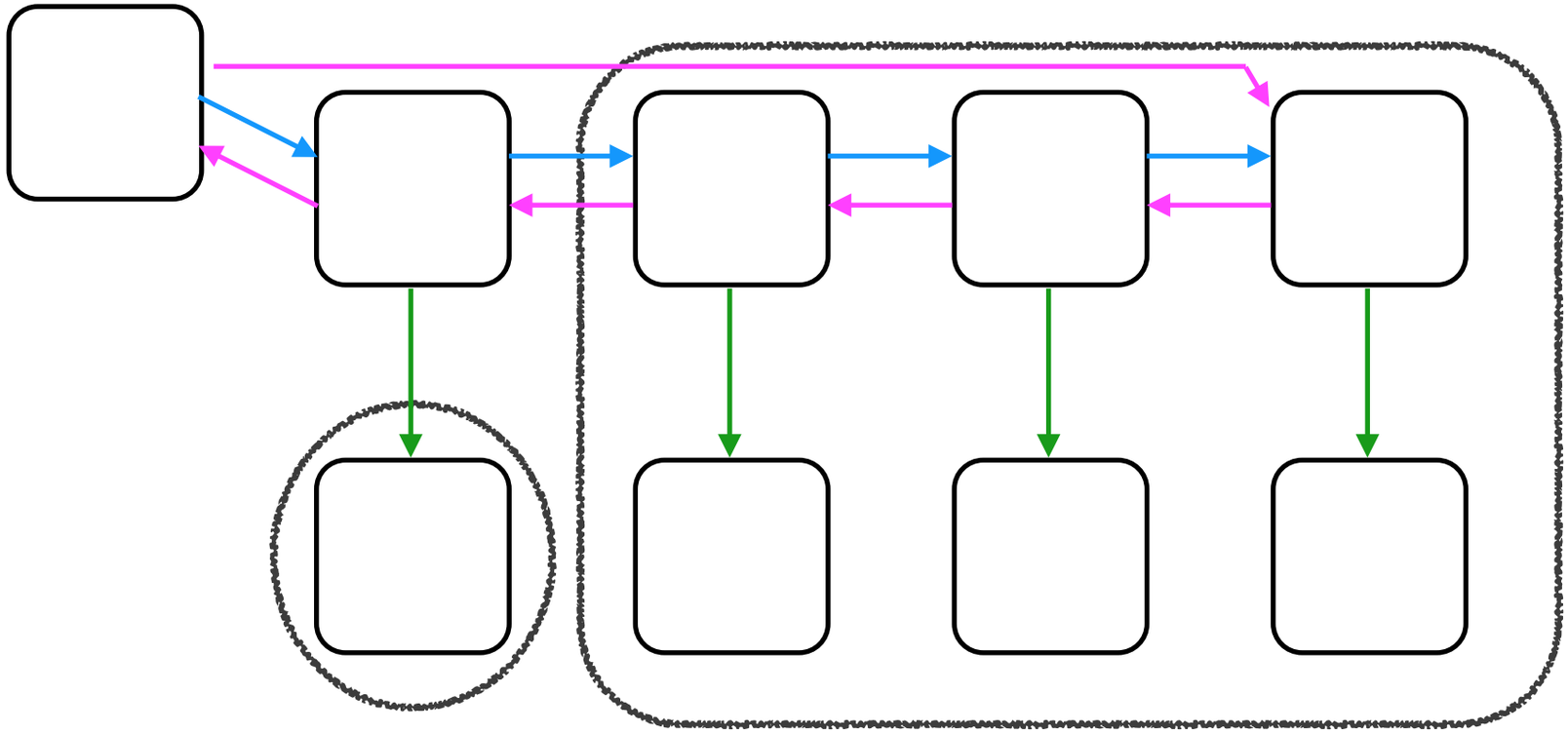
Parallelism

Doubly linked list: parallel + deterministic `foreach` method.



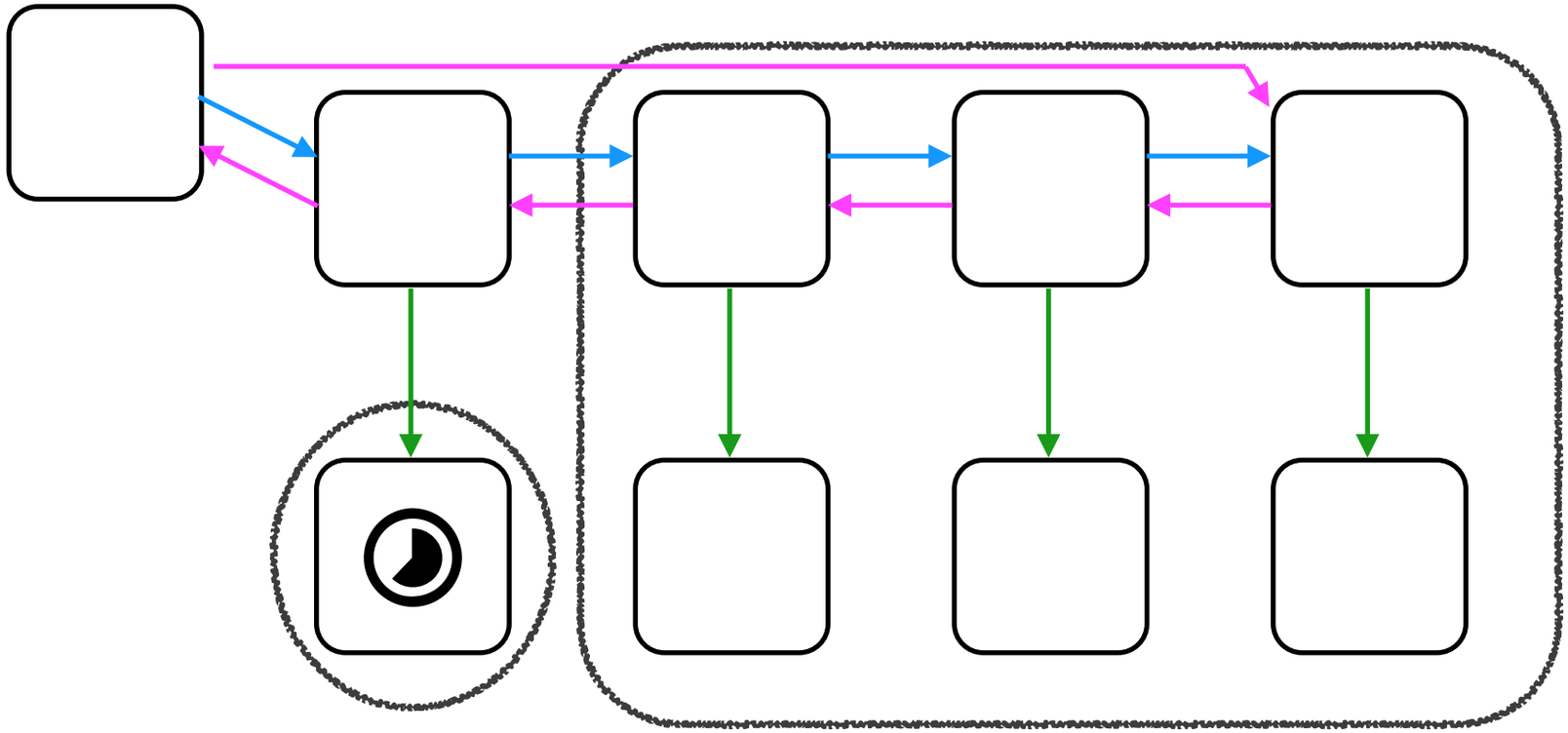
Parallelism

Doubly linked list: parallel + deterministic `foreach` method.



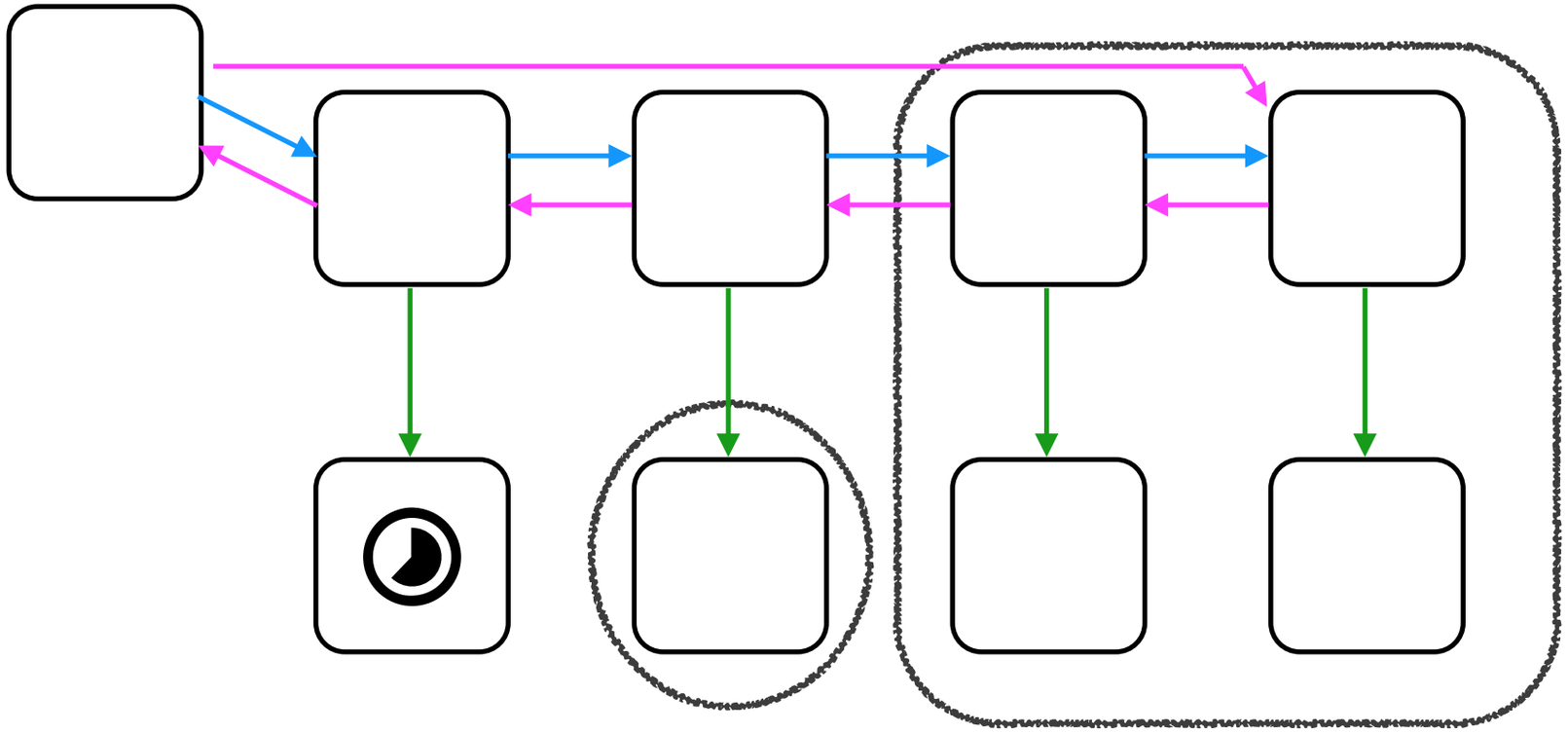
Parallelism

Doubly linked list: parallel + deterministic `foreach` method.



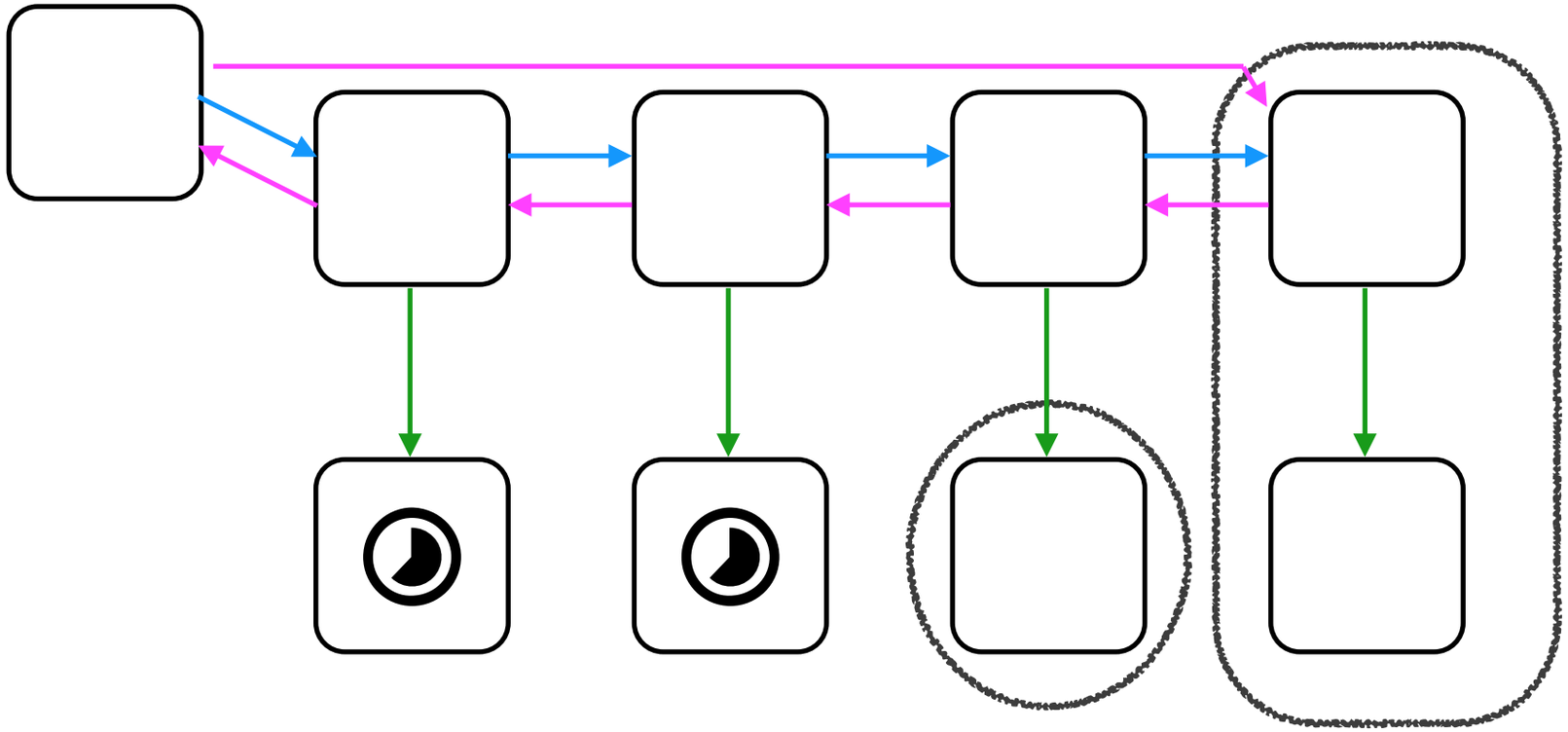
Parallelism

Doubly linked list: parallel + deterministic `foreach` method.



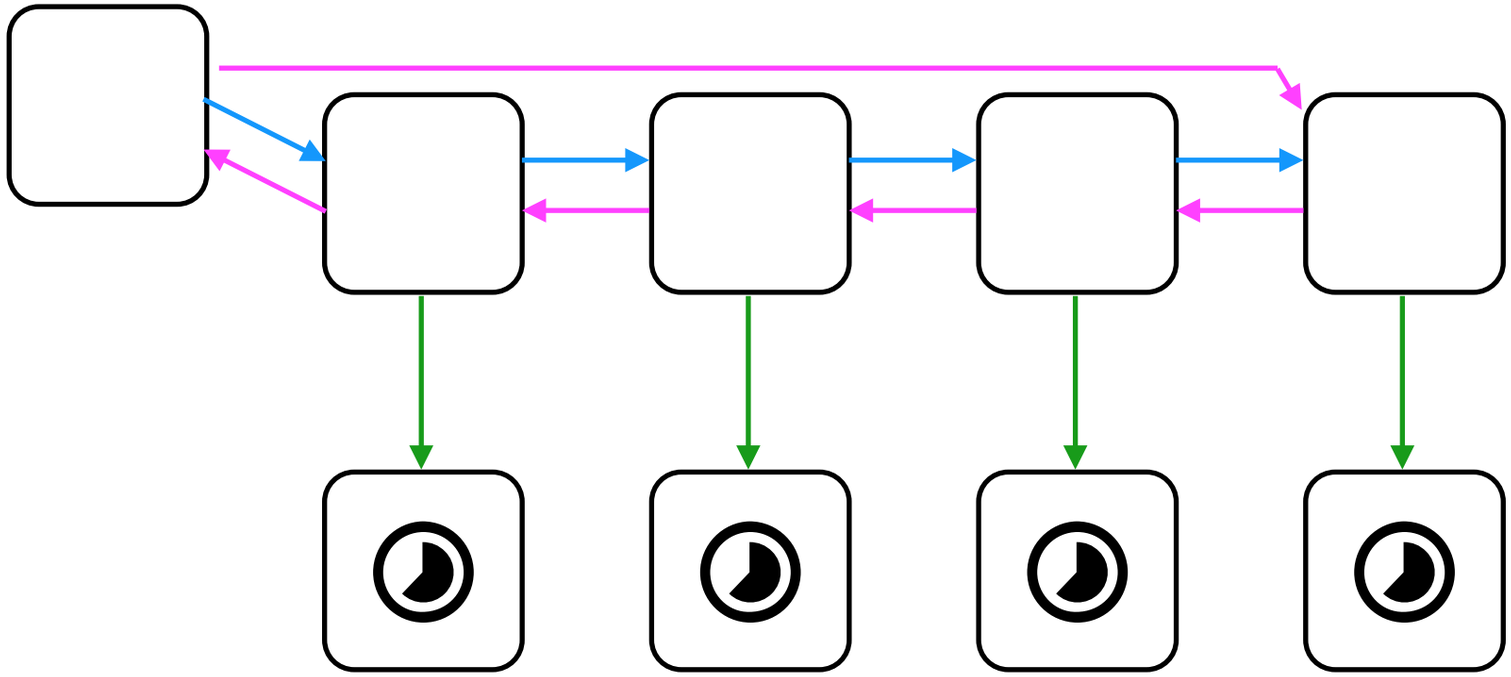
Parallelism

Doubly linked list: parallel + deterministic `foreach` method.

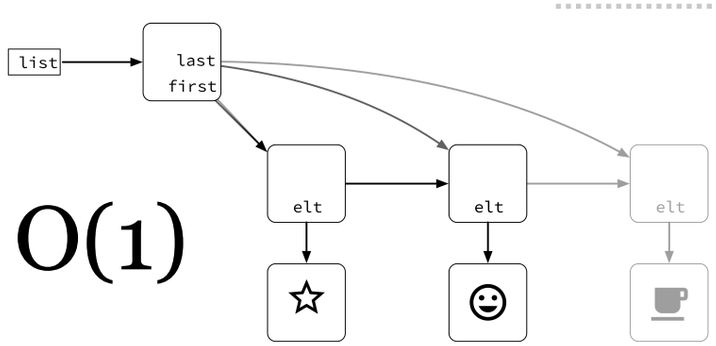


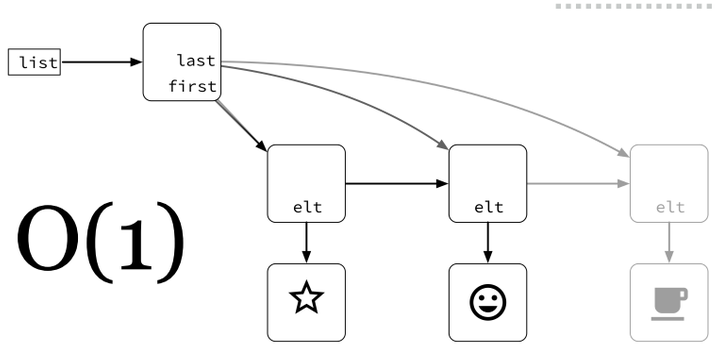
Parallelism

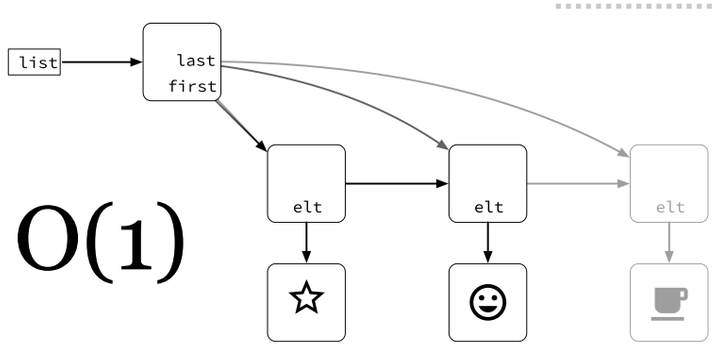
Doubly linked list: parallel + deterministic `foreach` method.

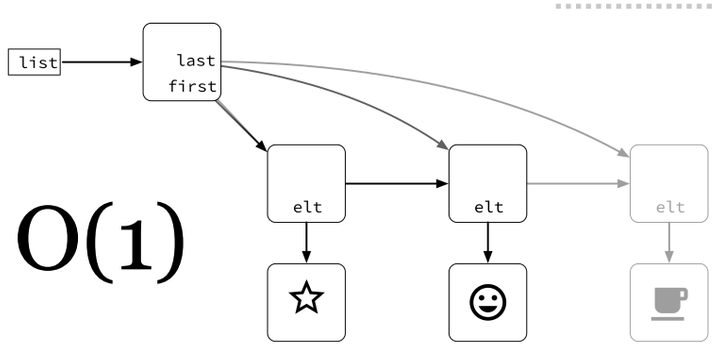


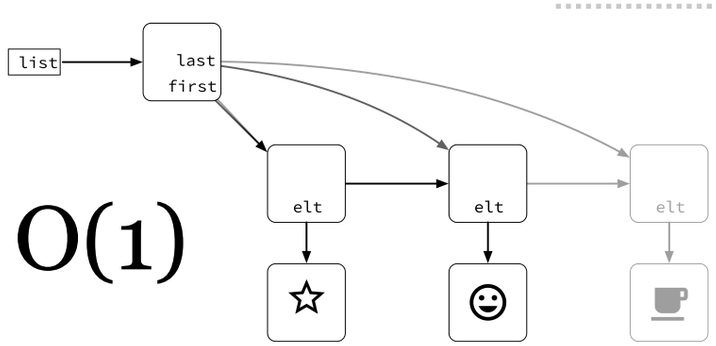
Wrapping Up

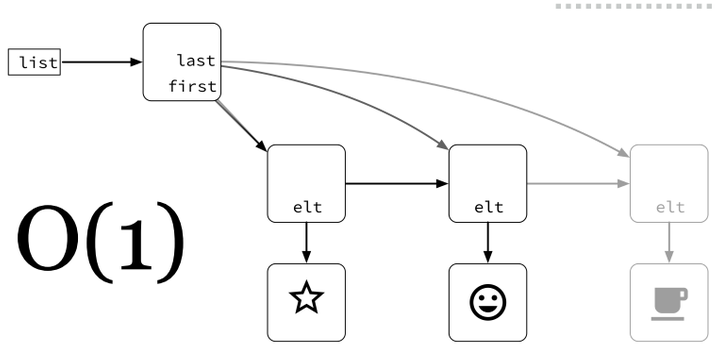
	Aliased	**Fine-Grained Aliasing**	Unaliased
`x.close();` `y.read();`			
`x.set(12) ||` `y.set(13)`			
 O(1)			

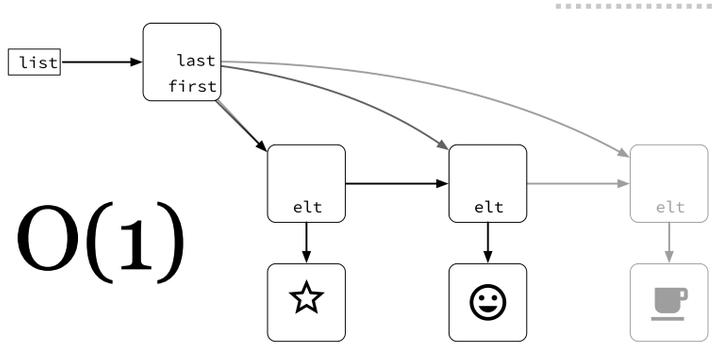
	Aliased	**Fine-Grained Aliasing**	Unaliased
`x.close();` `y.read();`			
`x.set(12) ||` `y.set(13)`			
 O(1)	✓		

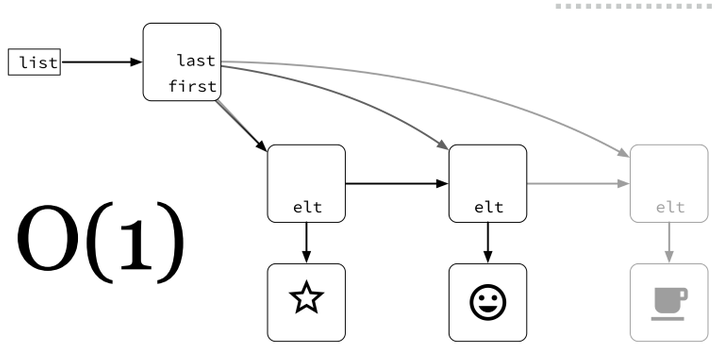
	Aliased	Fine-Grained Aliasing	Unaliased
`x.close();` `y.read();`	✗		
`x.set(12) ||` `y.set(13)`	✗		
 O(1)	✓		

	Aliased	Fine-Grained Aliasing	Unaliased
`x.close();` `y.read();`	✗		✓
`x.set(12) ||` `y.set(13)`	✗		✓
 O(1)	✓		

	Aliased	**Fine-Grained Aliasing**	Unaliased
`x.close();` `y.read();`	✗		✓
`x.set(12) ||` `y.set(13)`	✗		✓
 O(1)	✓		✗

	Aliased	Fine-Grained Aliasing	Unaliased
`x.close();` `y.read();`	✗	✓	✓
`x.set(12) ||` `y.set(13)`	✗	✓	✓
 O(1)	✓	✓	✗

	Aliased	Fine-Grained Aliasing	Unaliased
`x.close();` `y.read();`	✗	✓	✓
`x.set(12) ||` `y.set(13)`	✗	✓	✓
 O(1)	✓	✓	✗

	Aliased	Fine-Grained Aliasing	Unaliased
`x.close();` `y.read();`	✗	✓	✓
`x.set(12) ||` `y.set(13)`	✗	✓	✓
 O(1)	✓	✓	✗

Thank You

Unique to fully shared, gradually.

Sane Defaults: Annotate only ‘bad’ behaviour.

Variable Aliasing + Object Disjointness + Encapsulation.

Direct translation from Java-like programs.

Example: Doubly linked list + parallel foreach + iterators.

@sbrandauer | http://stbr.me

Thank You

Unique to fully shared, gradually.

Sane Defaults: Annotate only ‘bad’ behaviour.

Variable Aliasing + Object Disjointness + Encapsulation.

Direct translation from Java-like programs.

Example: Doubly linked list + parallel foreach + iterators.

PREPRINT ON SPLASH WEBSITE!

@sbrandauer | http://stbr.me

Extra Slides

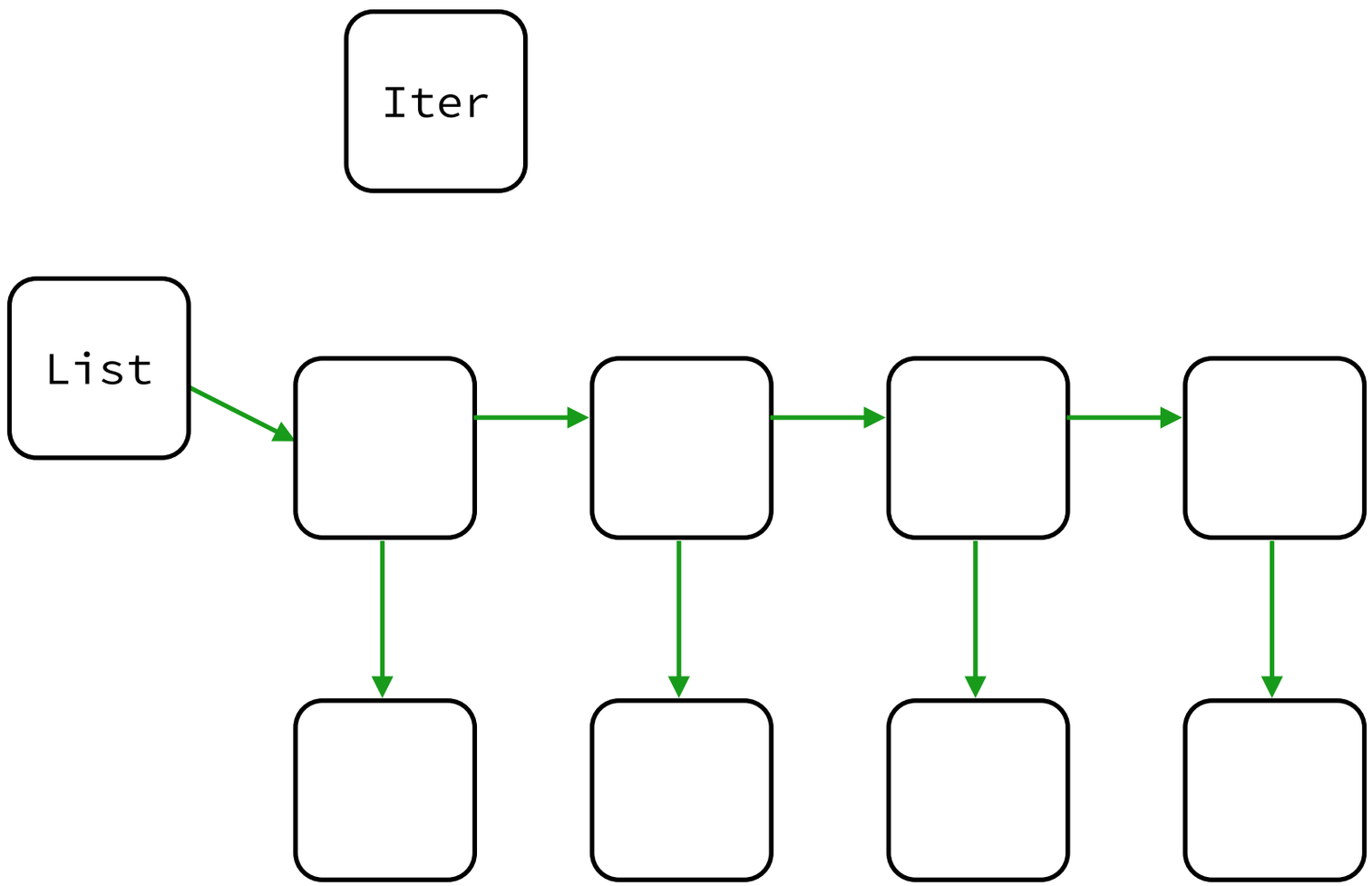
Compare to Rust?

- Rust makes shared data read-only, we make shared data sequential.

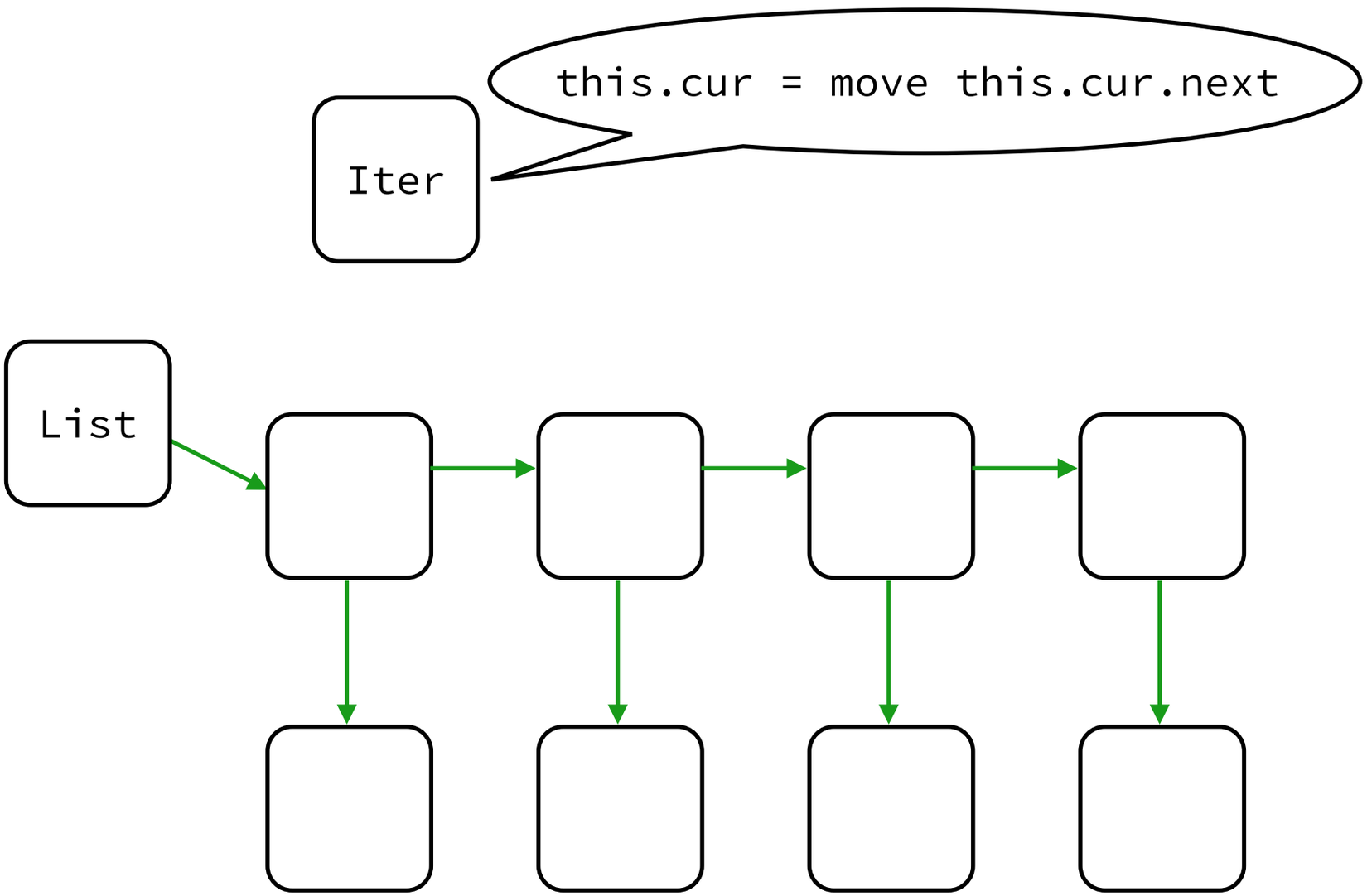
Compare to Fractional Permissions?

- Fractional Permissions make shared data read-only. We make shared data sequential.
- We have a general way to borrow from variables — see paper.
- Direct translation: java-like programs -> using disjointness domains badly.

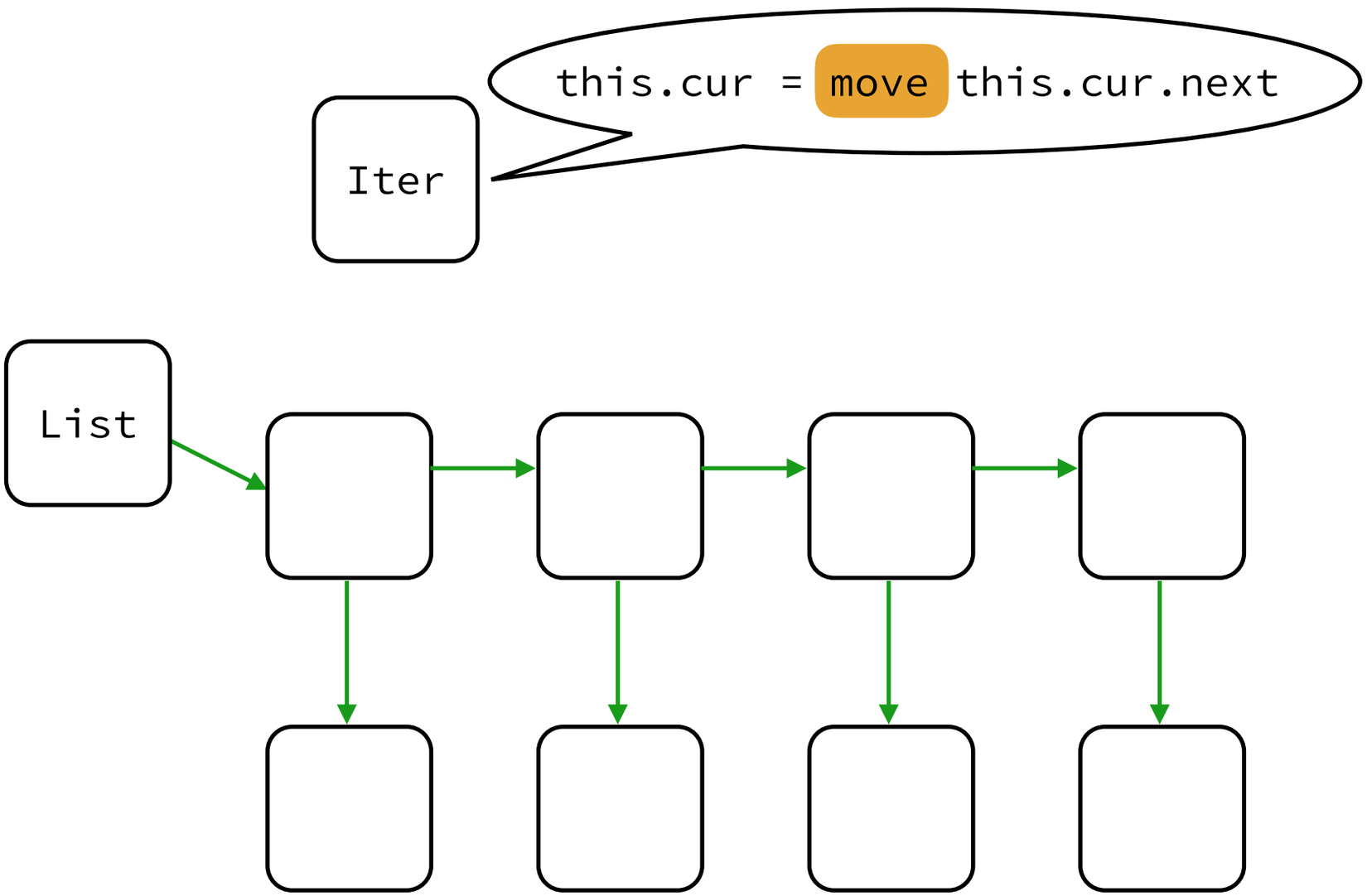
Iteration



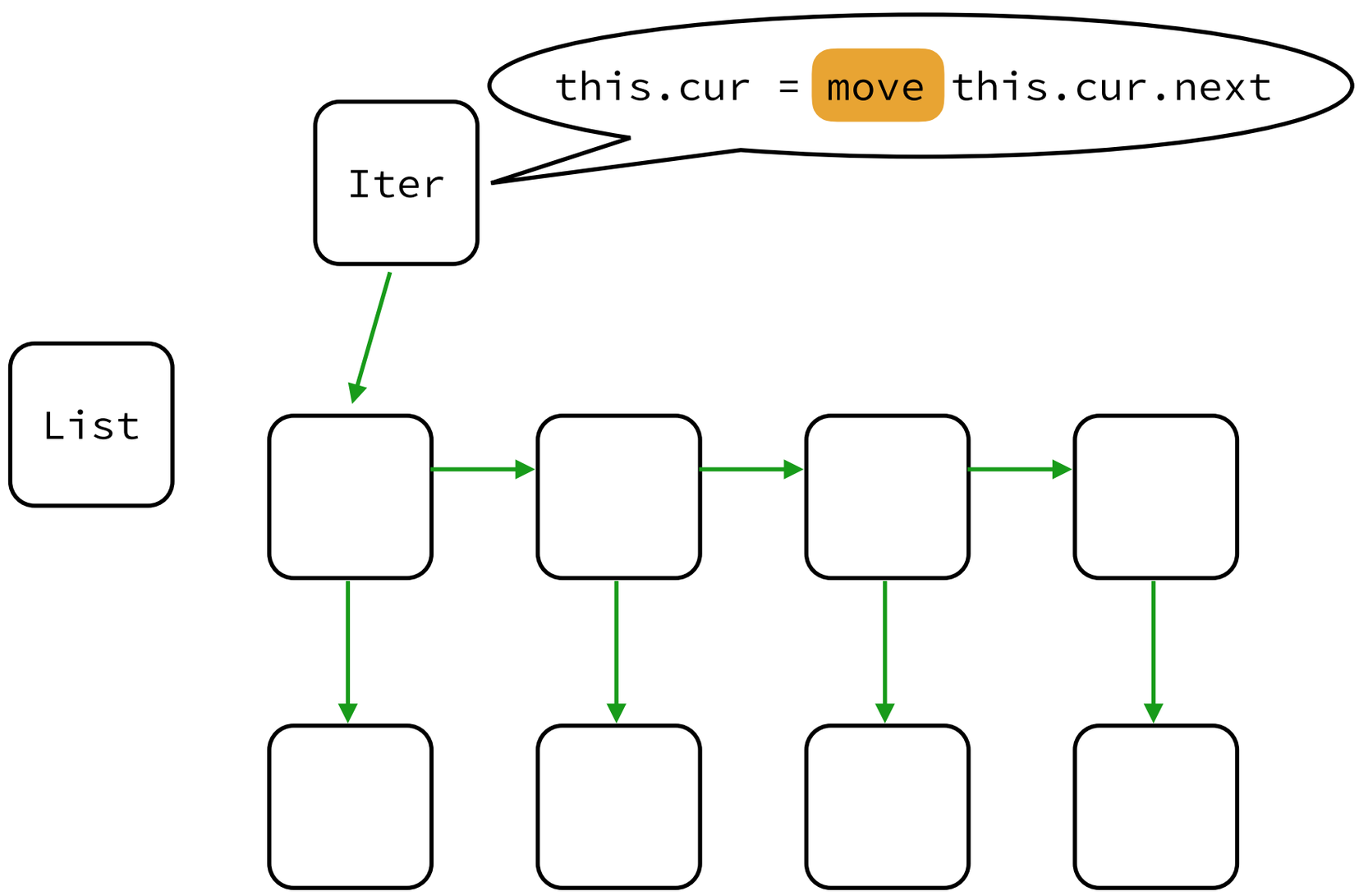
Iteration



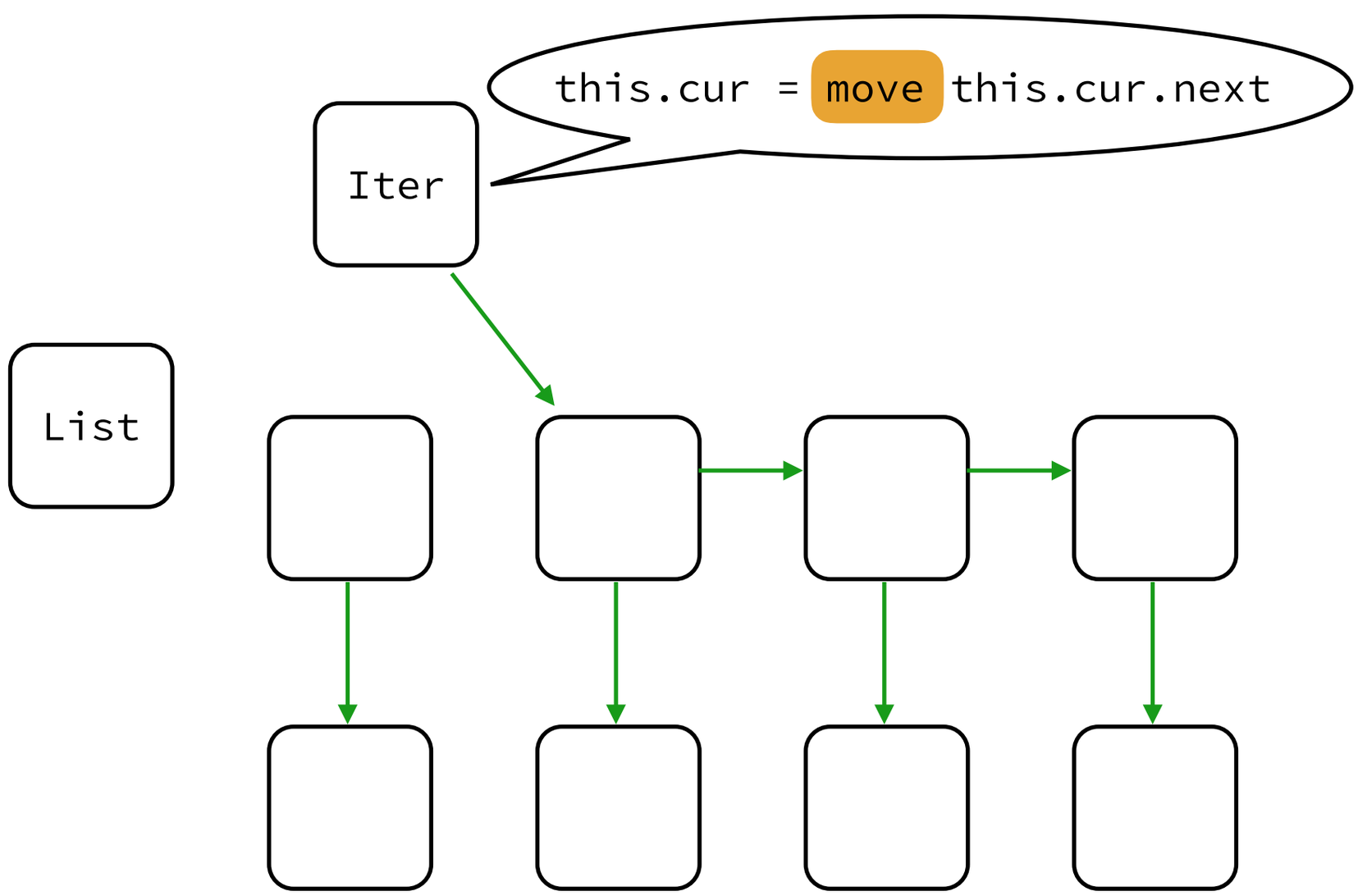
Iteration



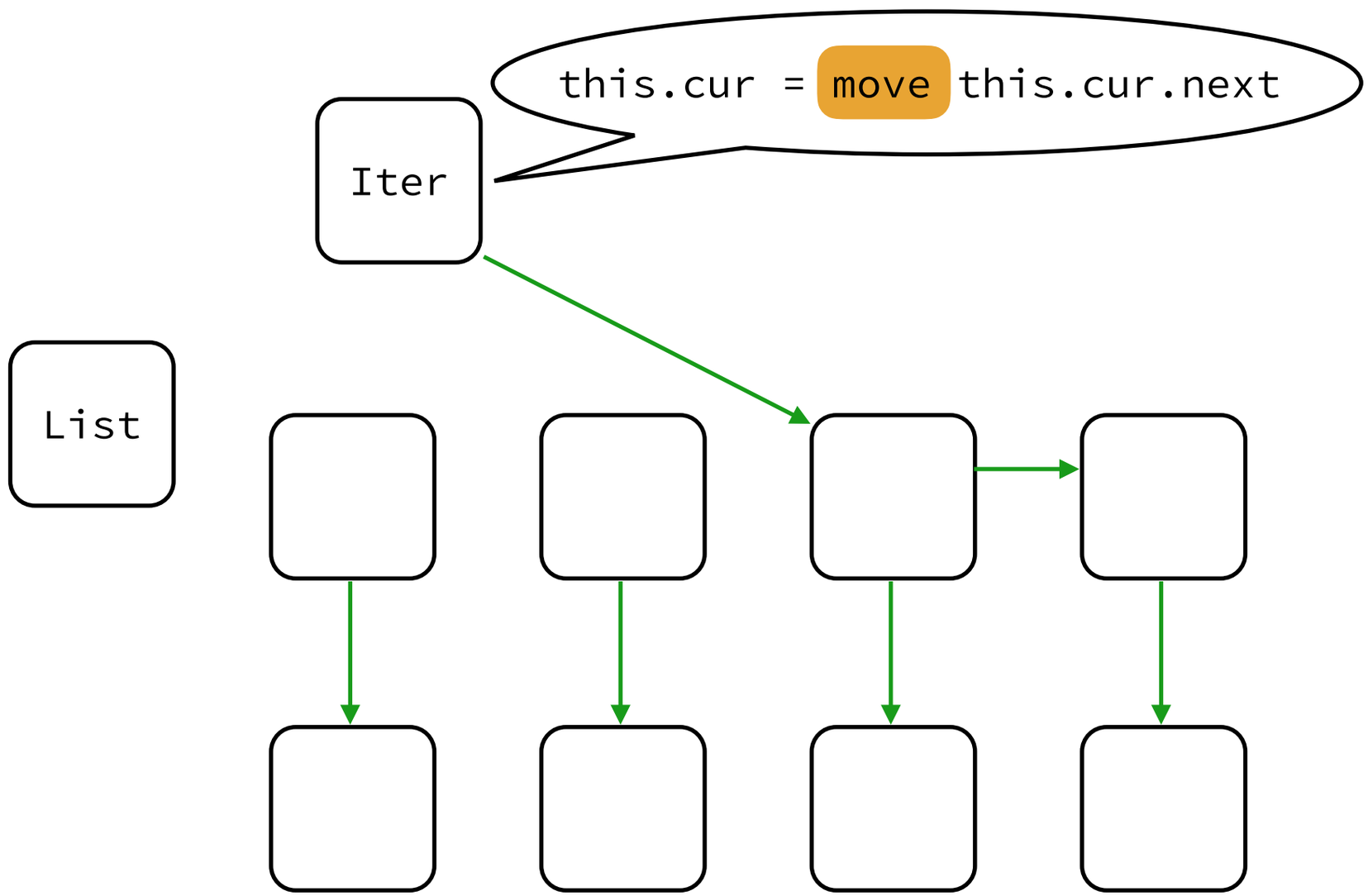
Iteration



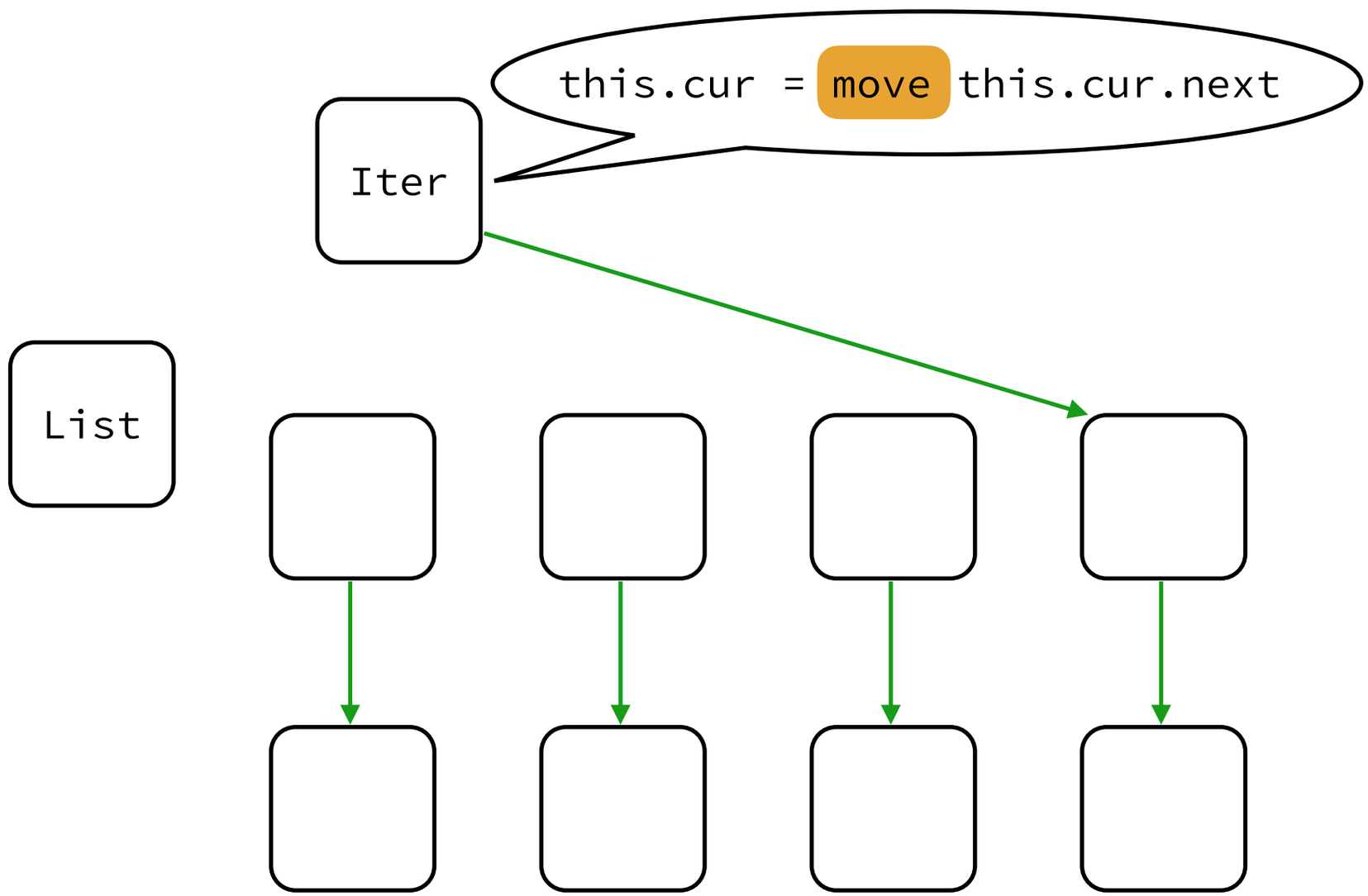
Iteration



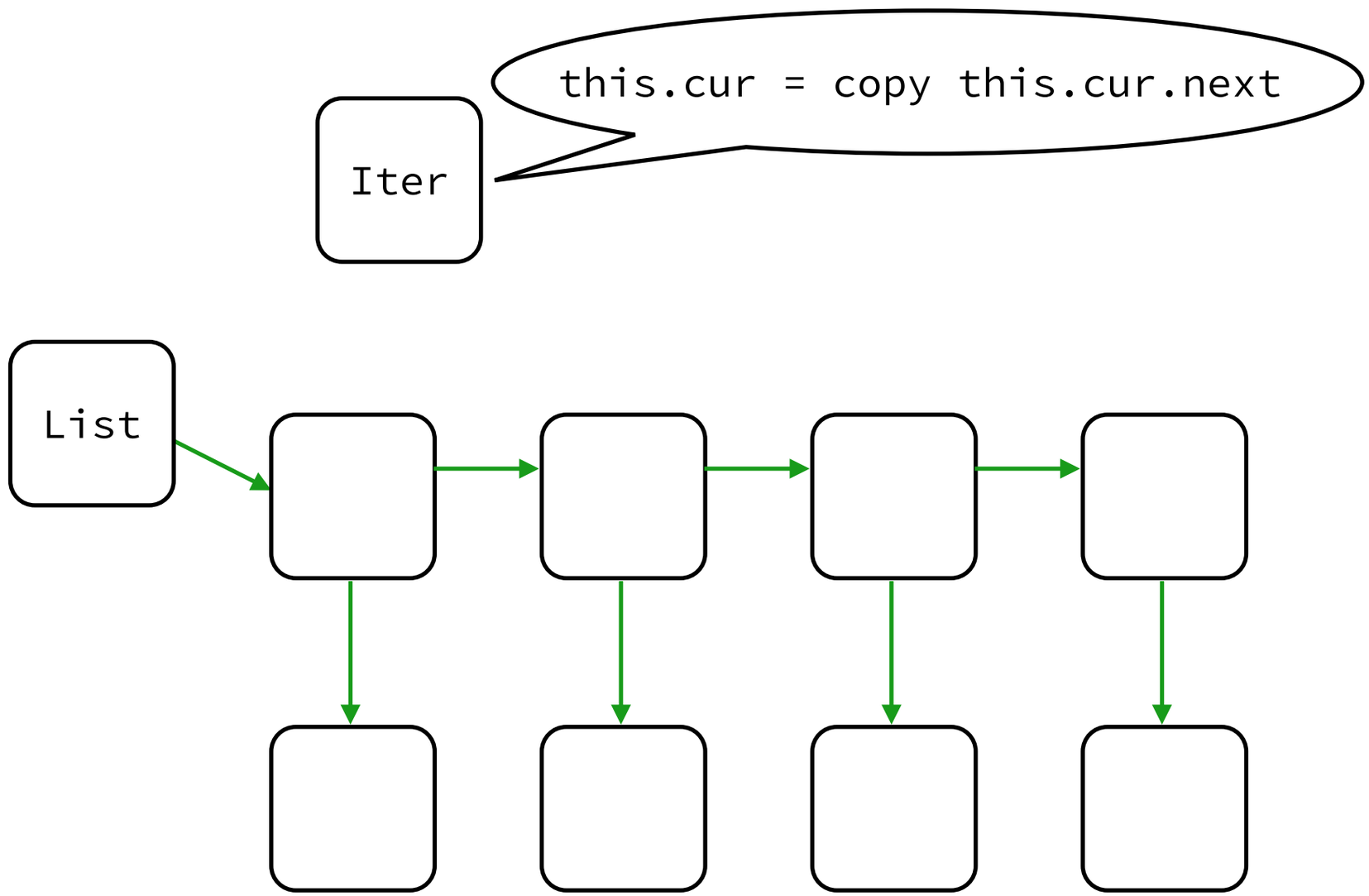
Iteration



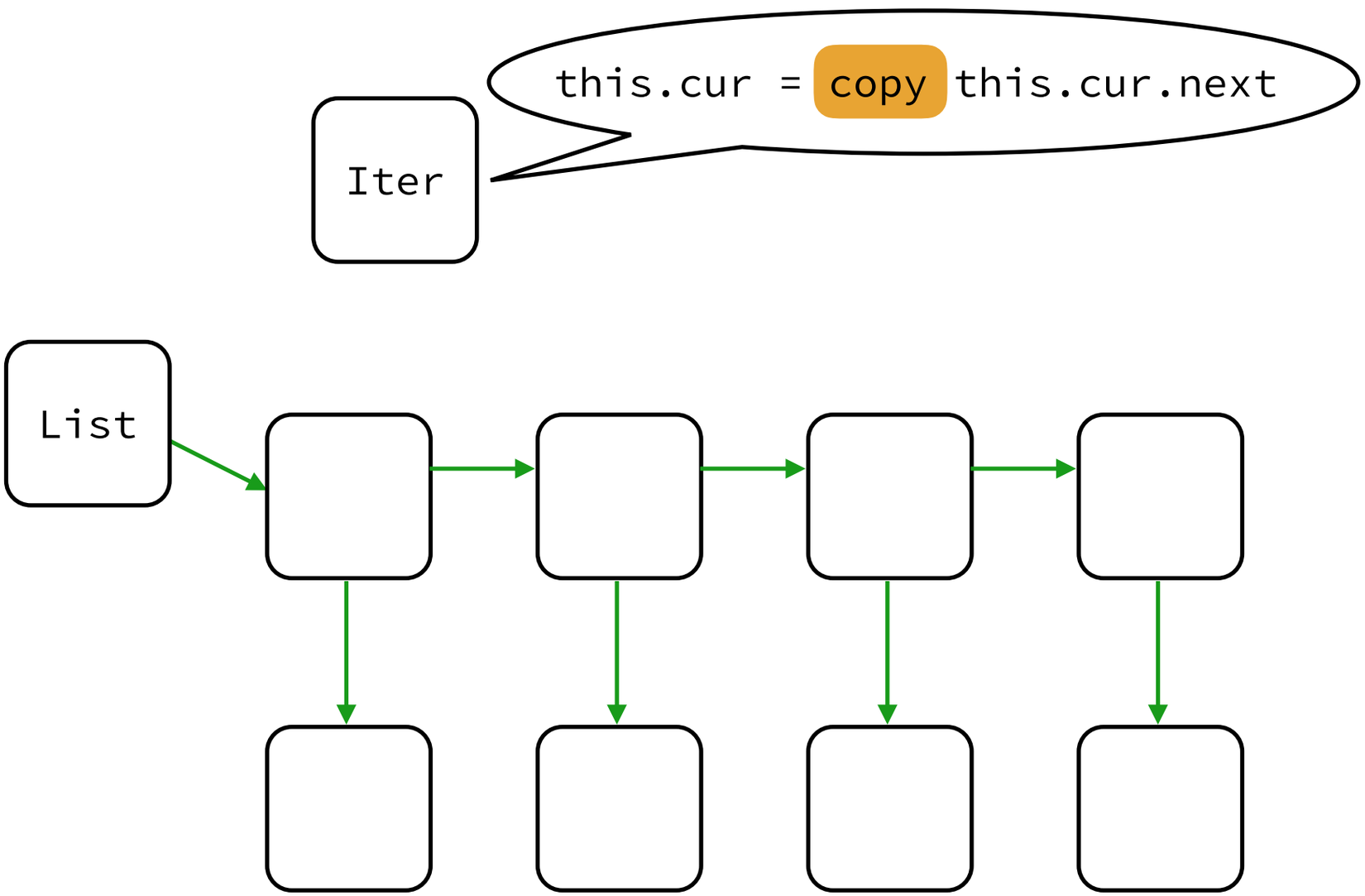
Iteration



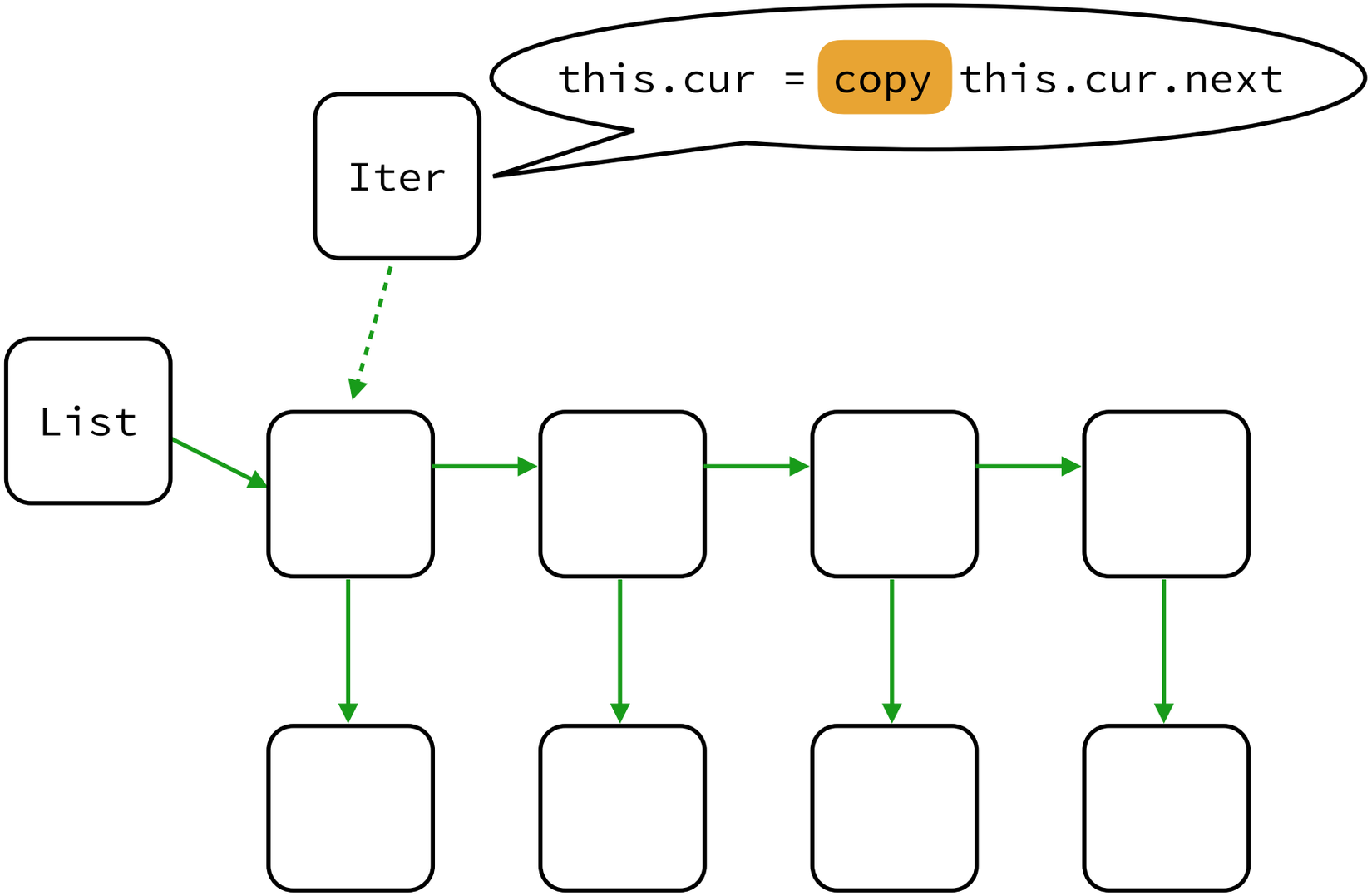
Iteration



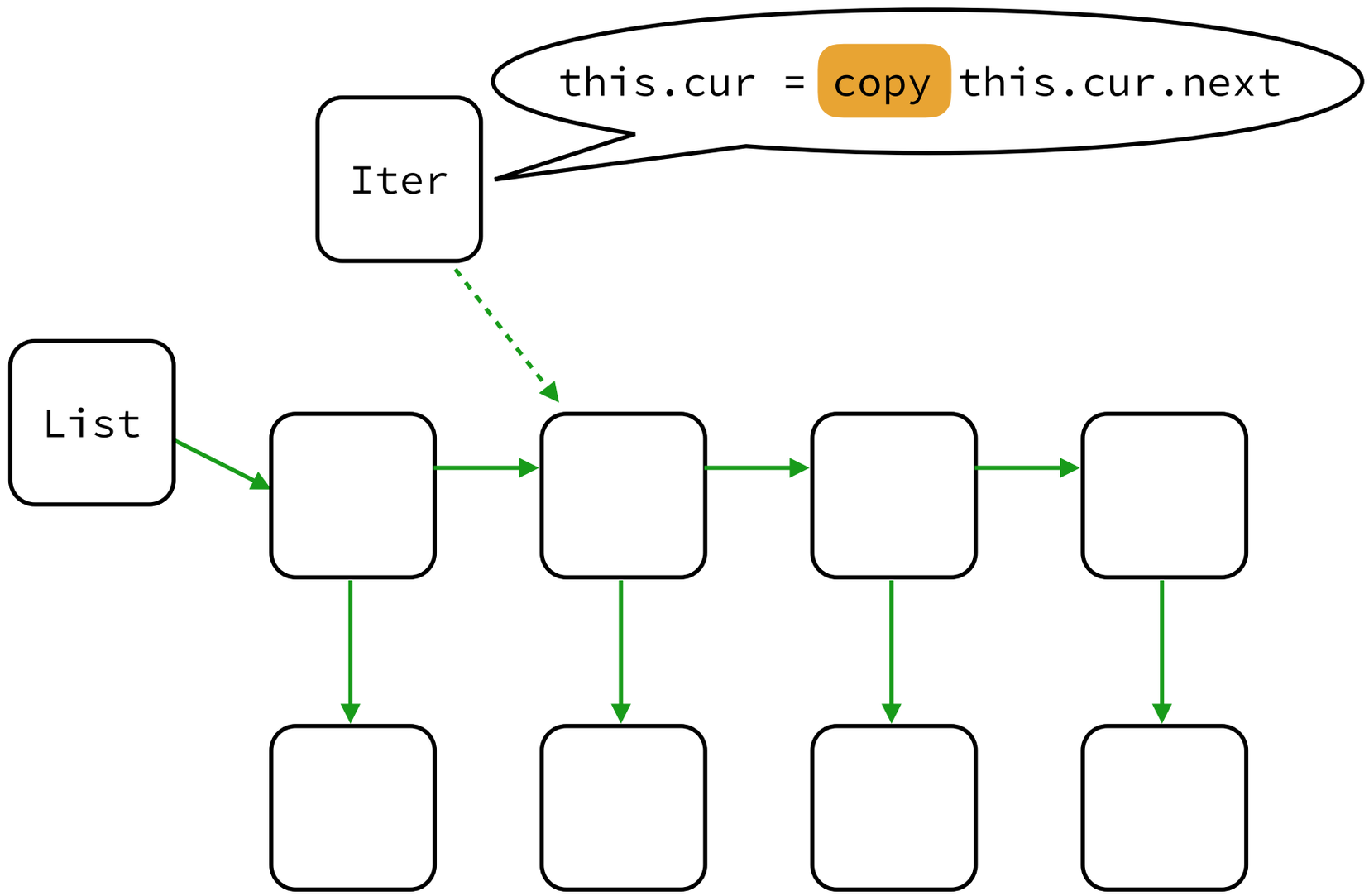
Iteration



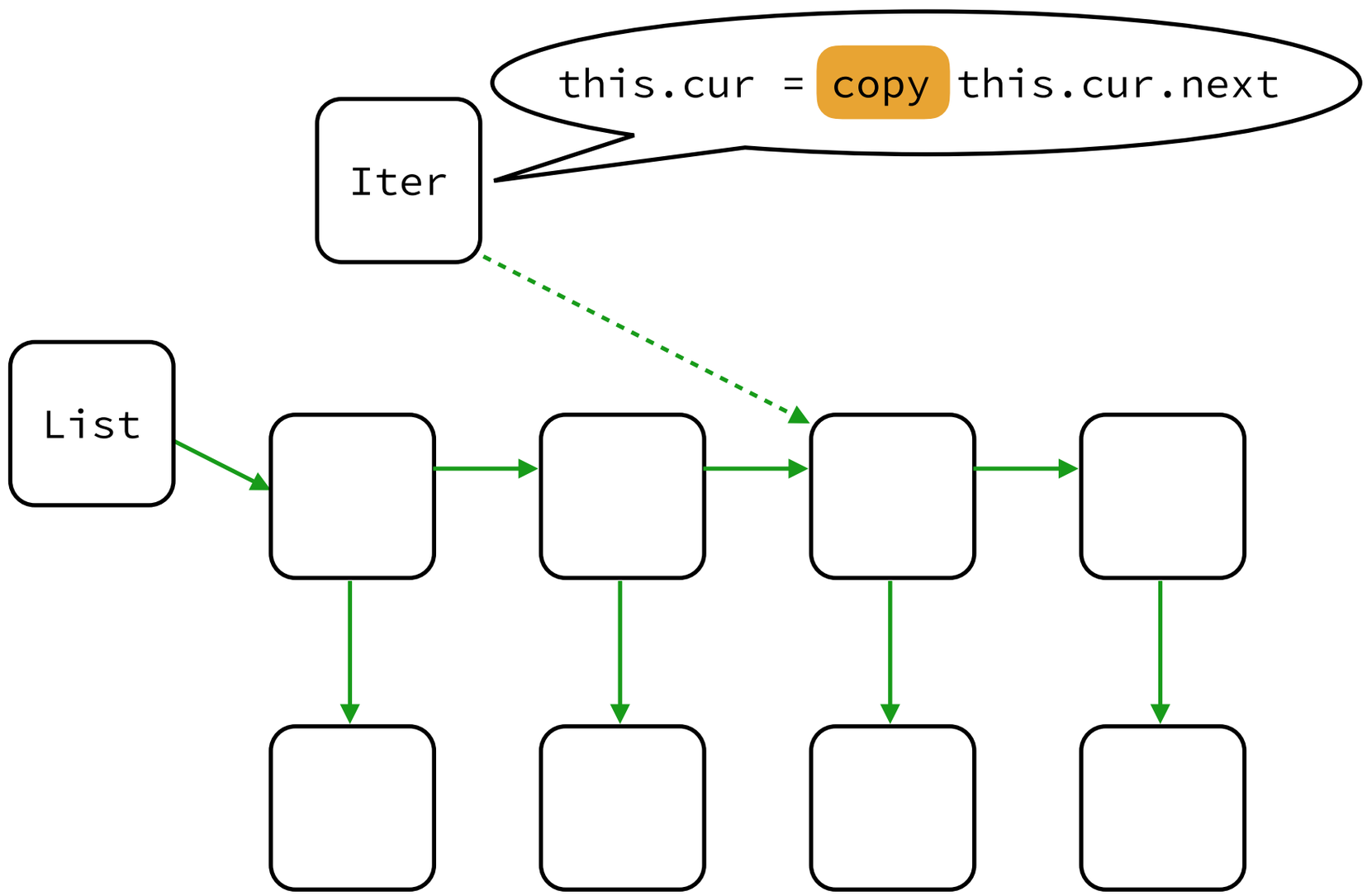
Iteration



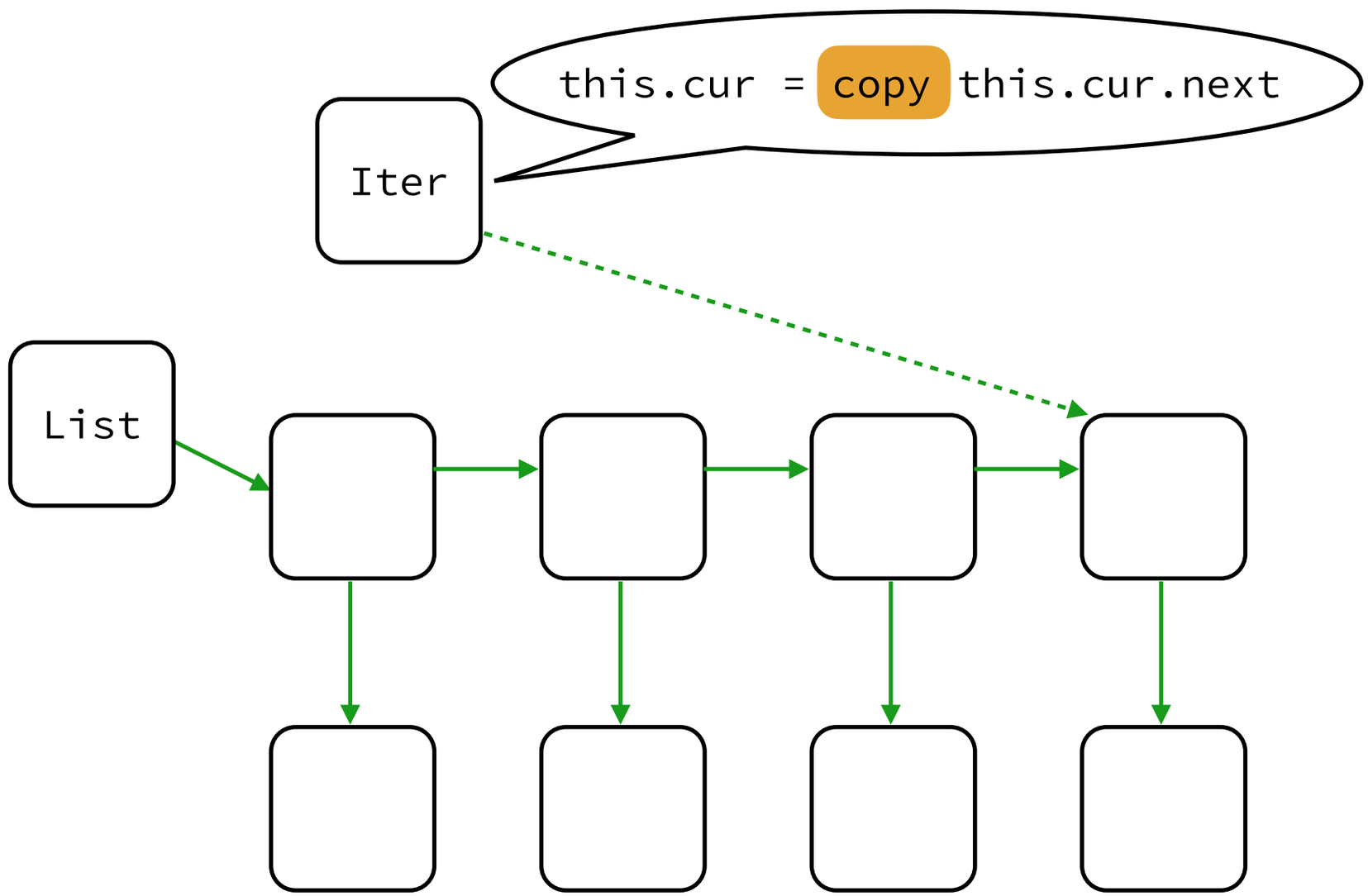
Iteration



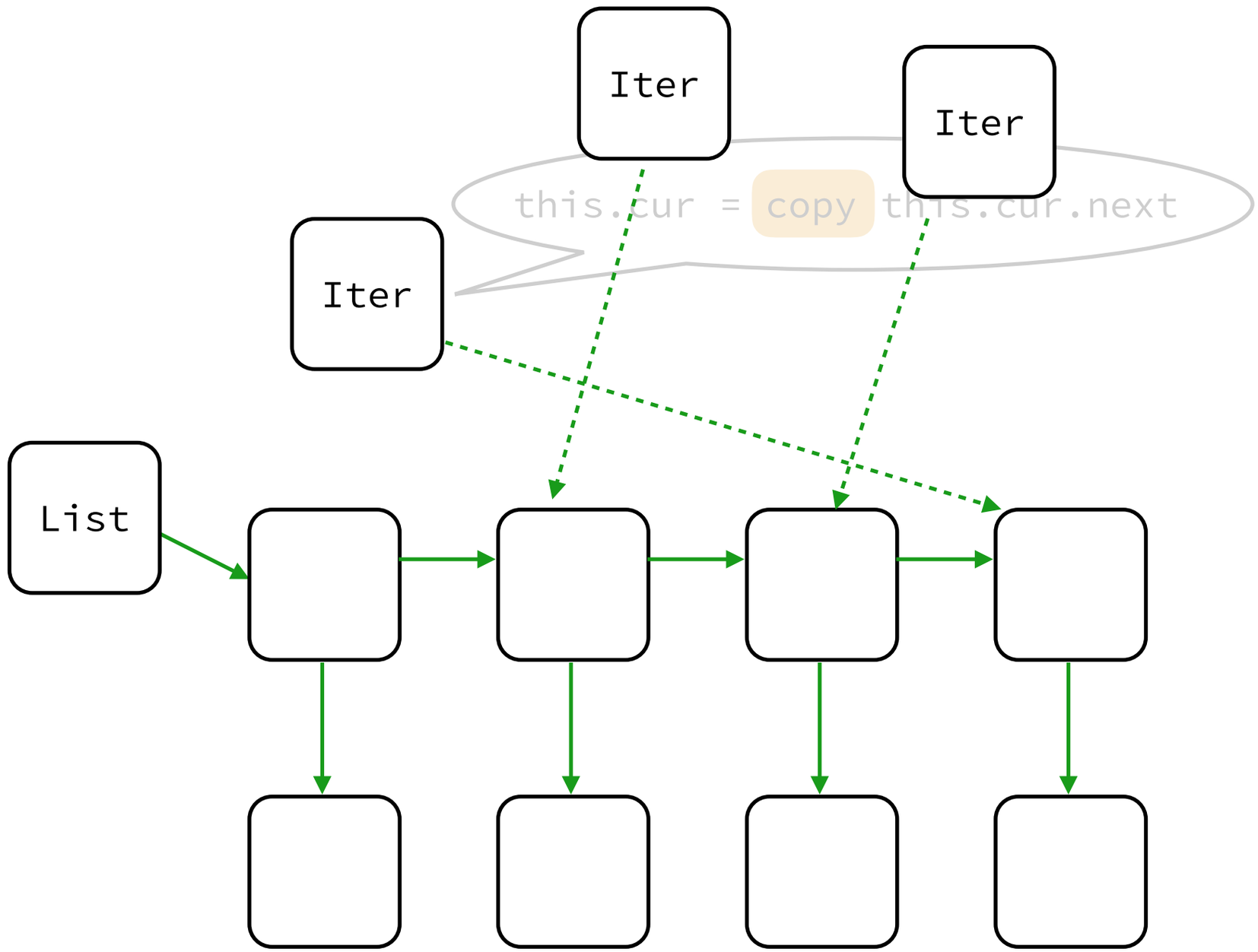
Iteration



Iteration



Iteration



Iteration