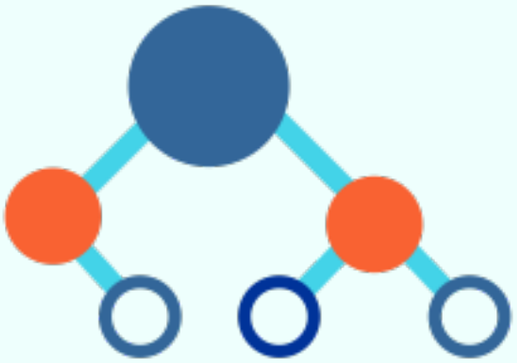
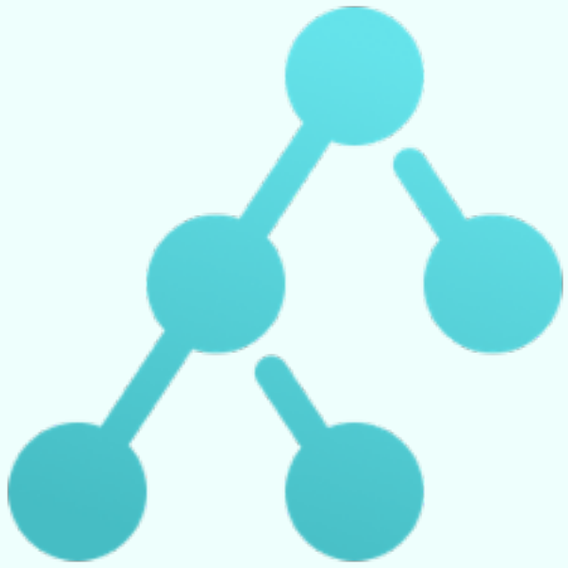
Data Structures

Exercise Session



Marco Vassena



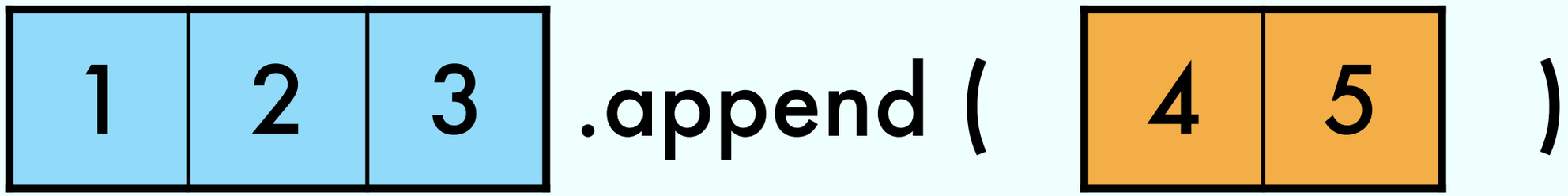
Exercise 1 from 12/08

Analyze the time complexity

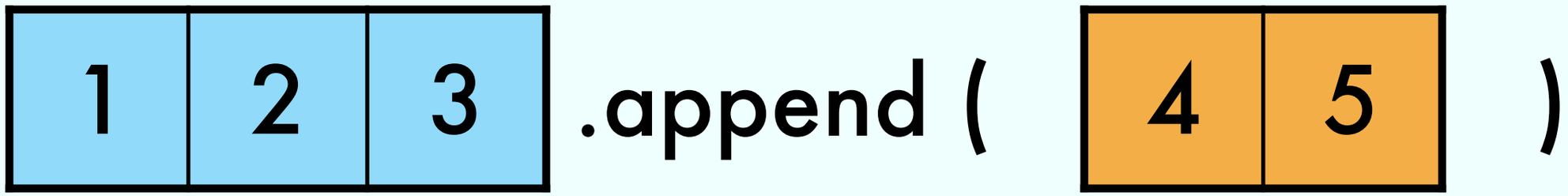
```
for(int r = 0; r < M; r++)
    for(int c = 0; c < N; c++)
        stack.push(c);
```

in terms of `M`, `N` and `|stack|`

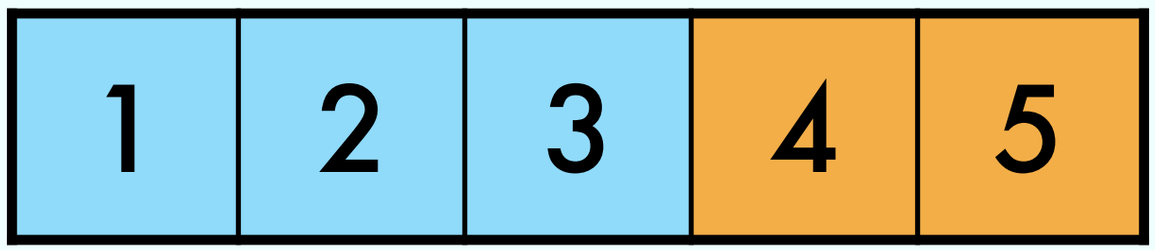
Exercise 3 from 12/04



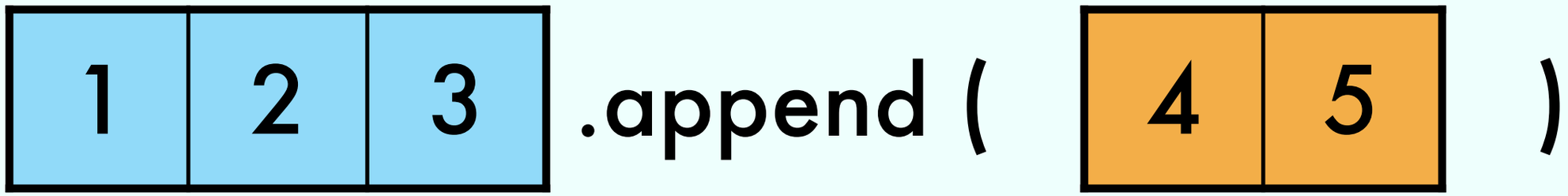
Exercise 3 from 12/04



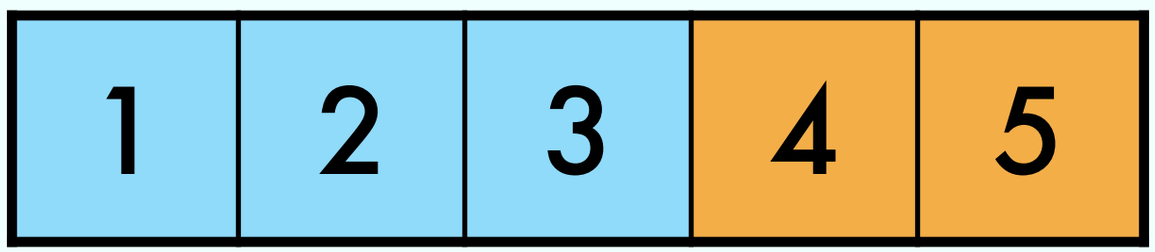
=



Exercise 3 from 12/04



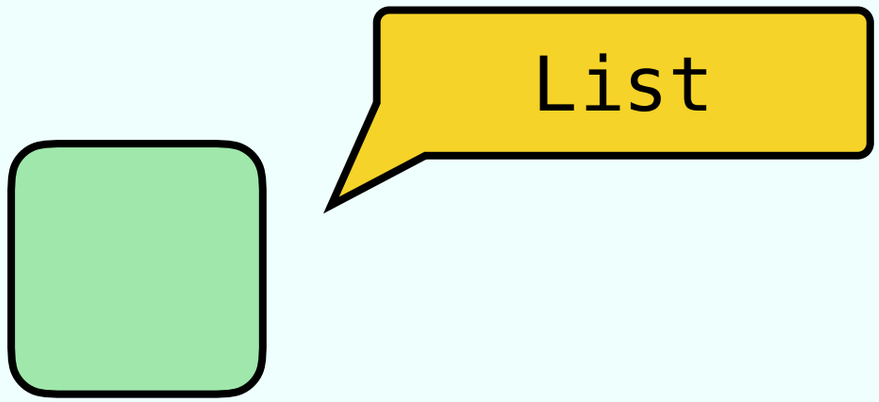
=



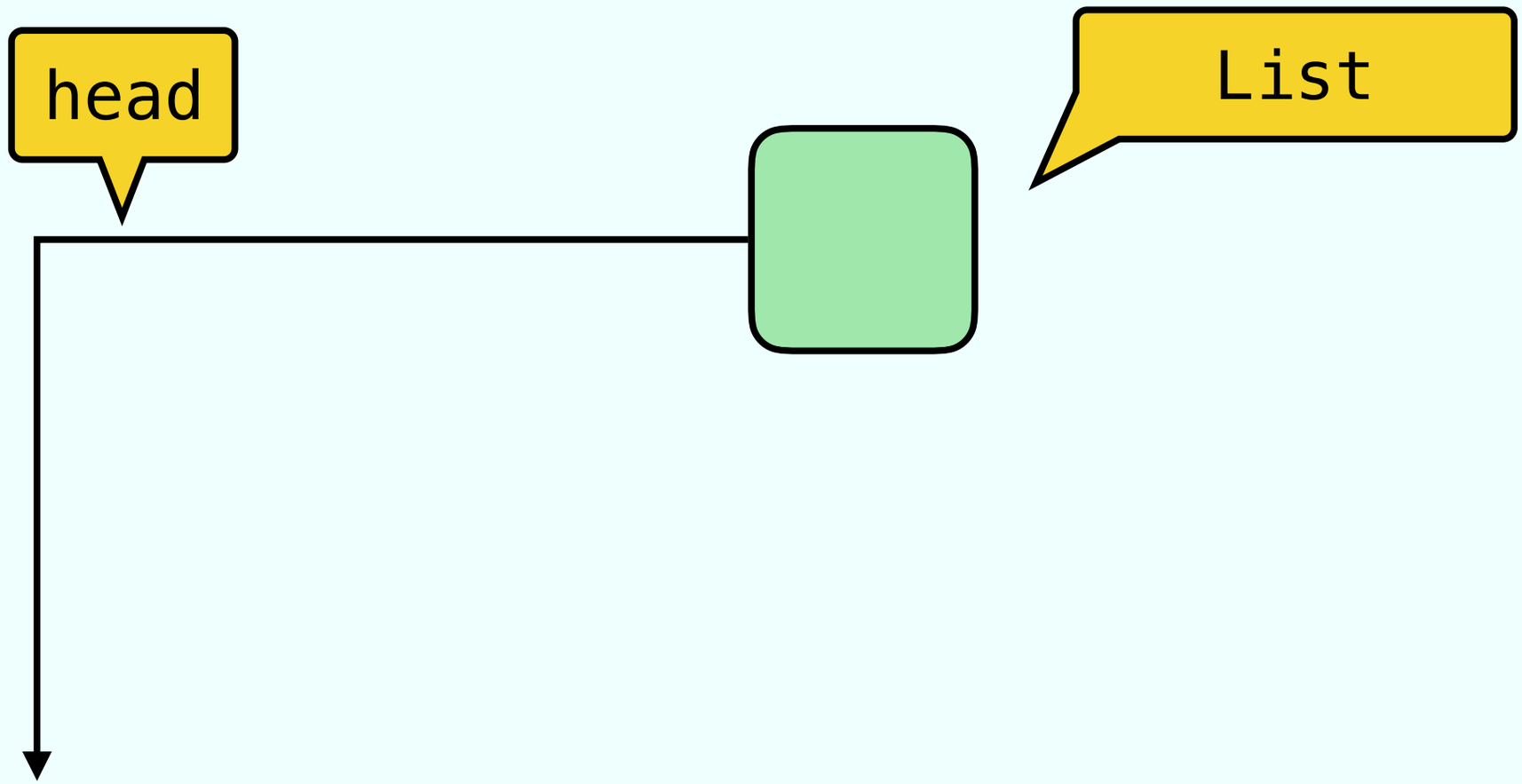
Implement append in O(1)

Linked List

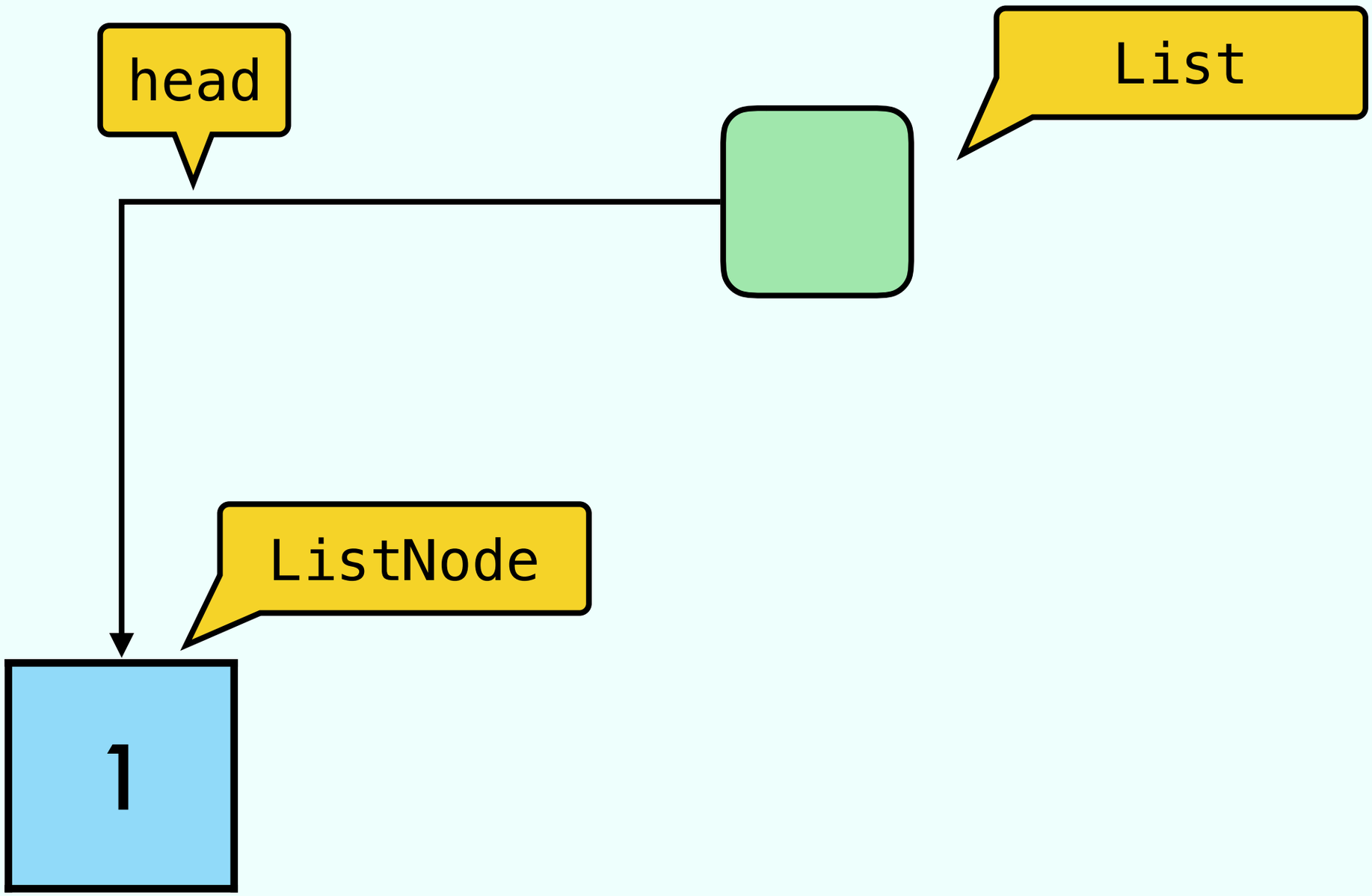
Linked List



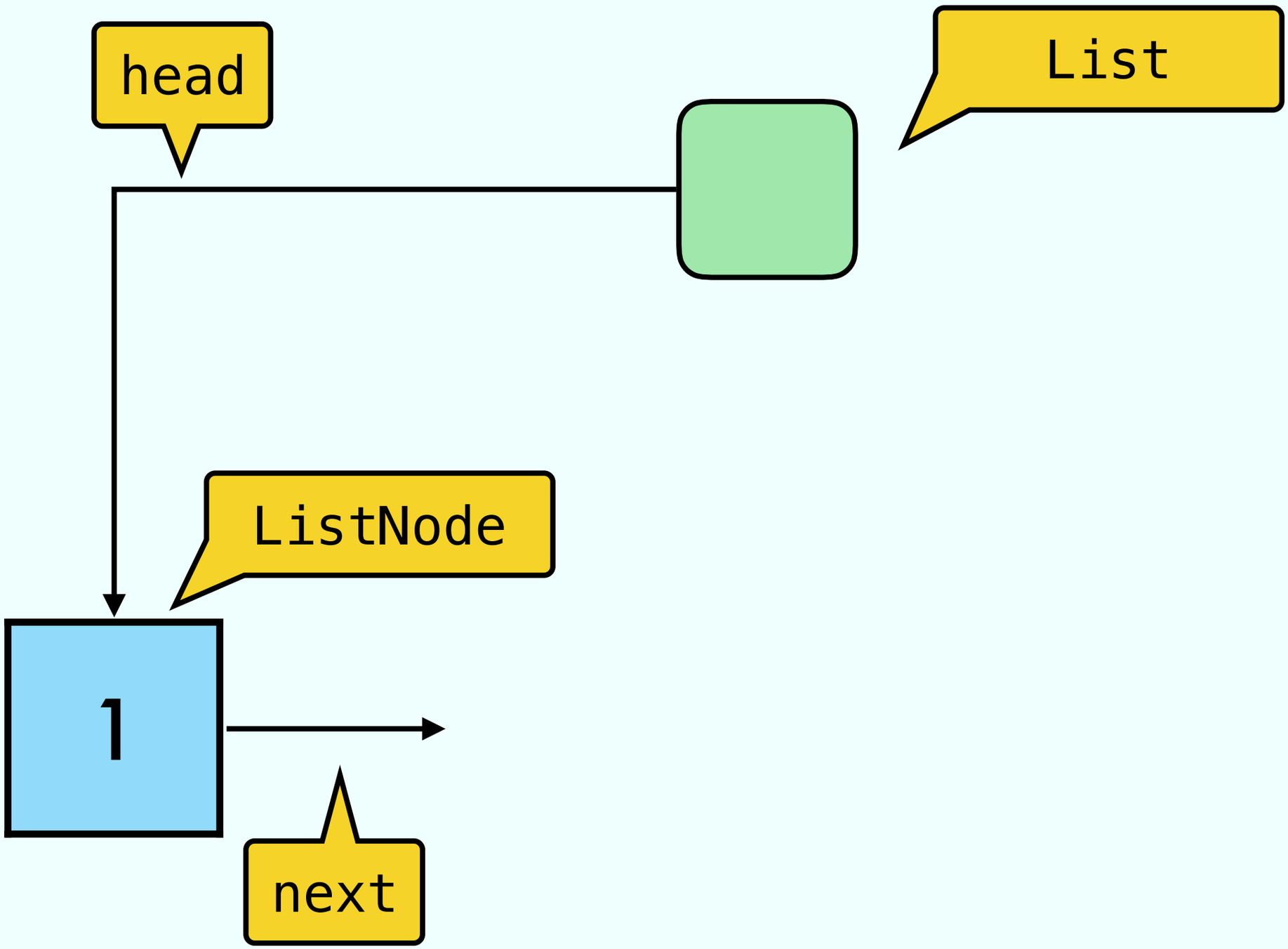
Linked List



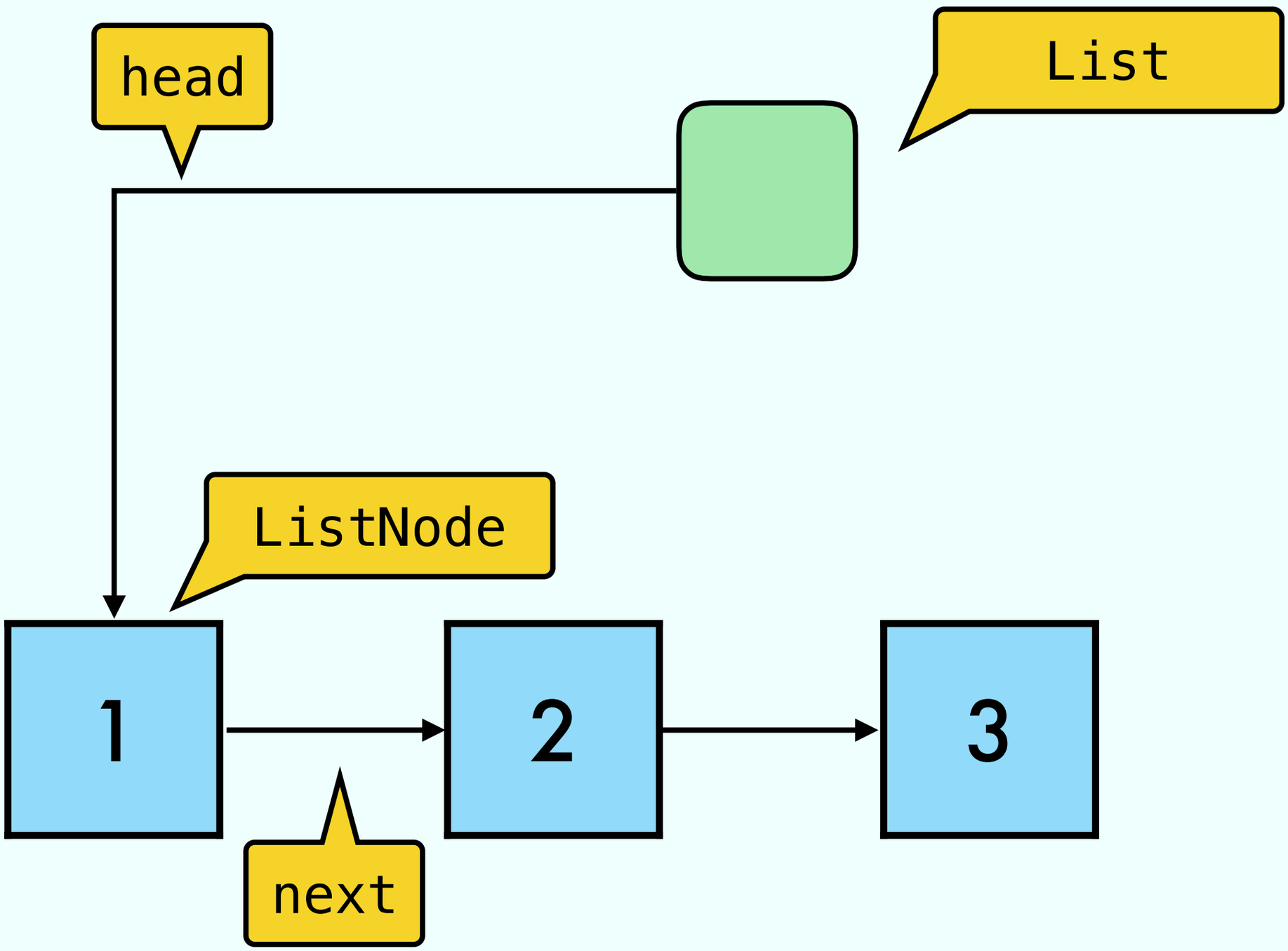
Linked List



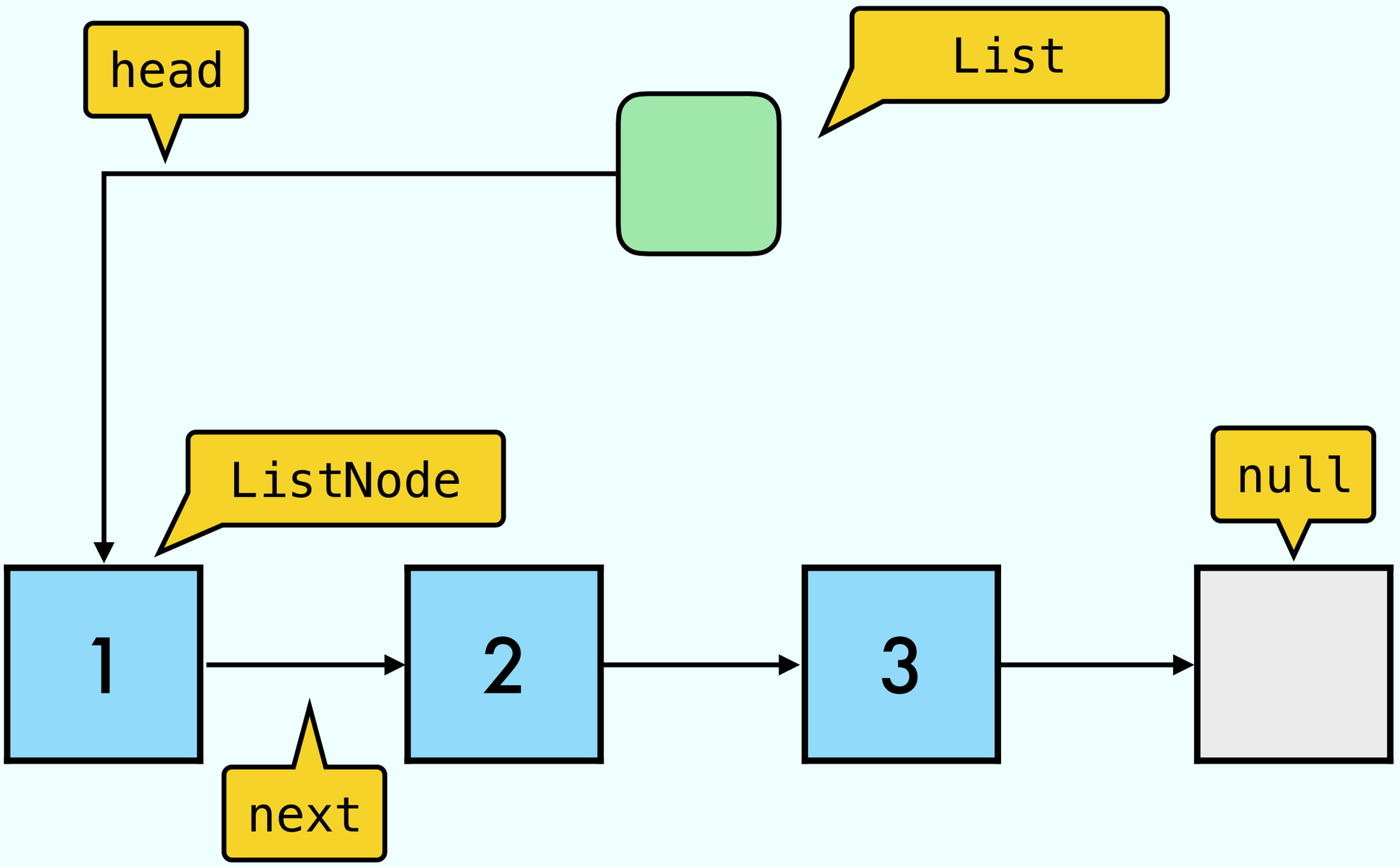
Linked List

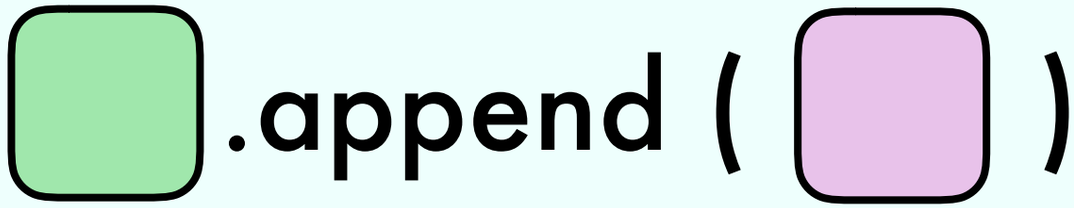


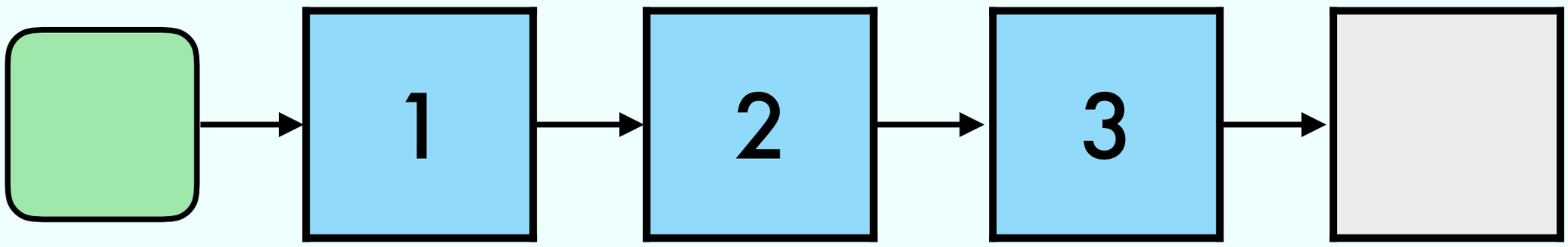
Linked List

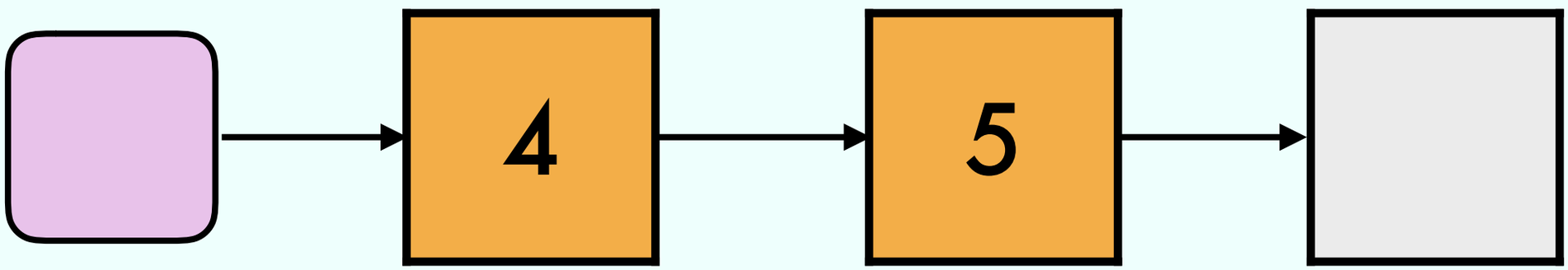


Linked List

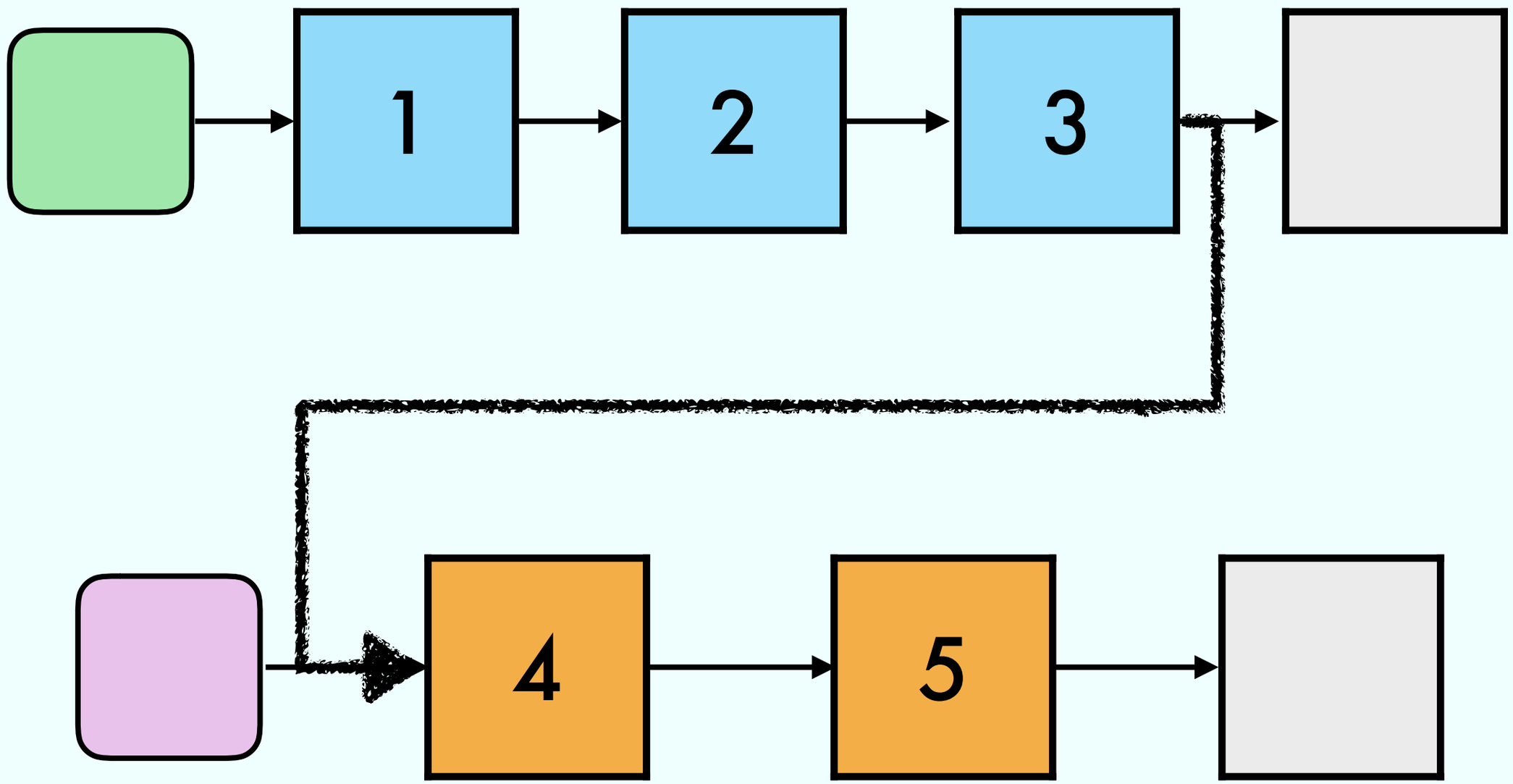



.append ( )


1
2
3

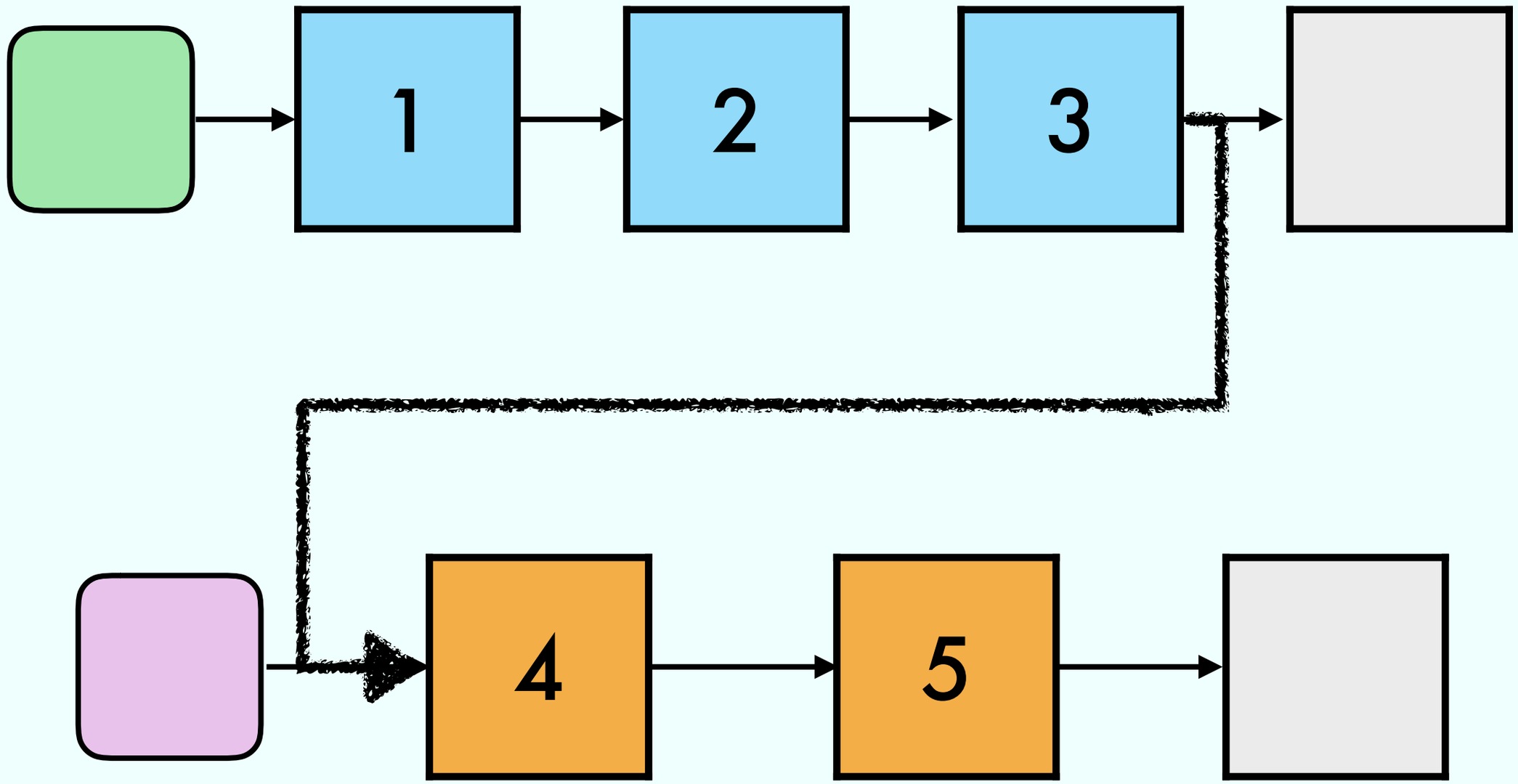

4
5


.append ( )


1
2
3
4
5

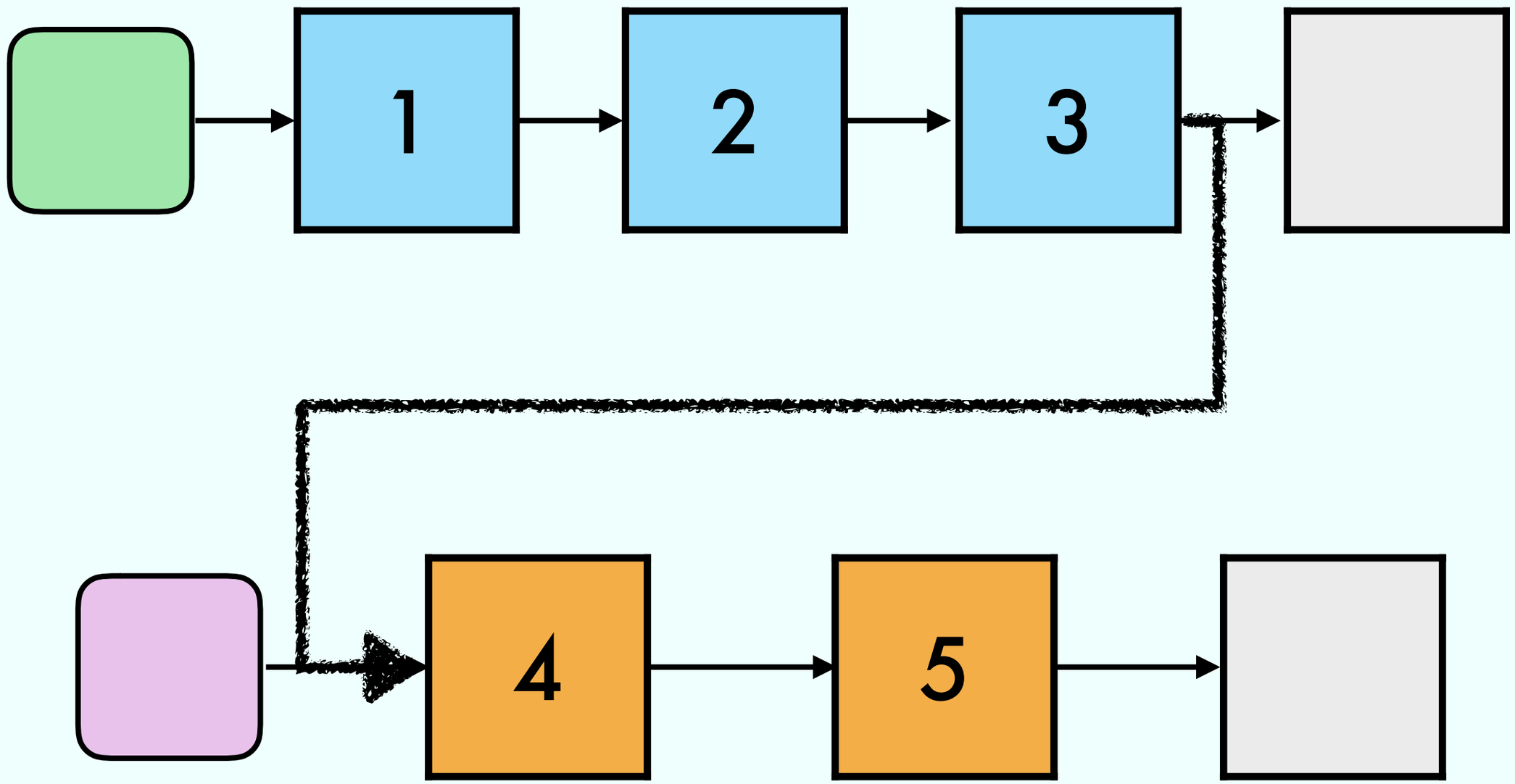
O(N)

.append ( )

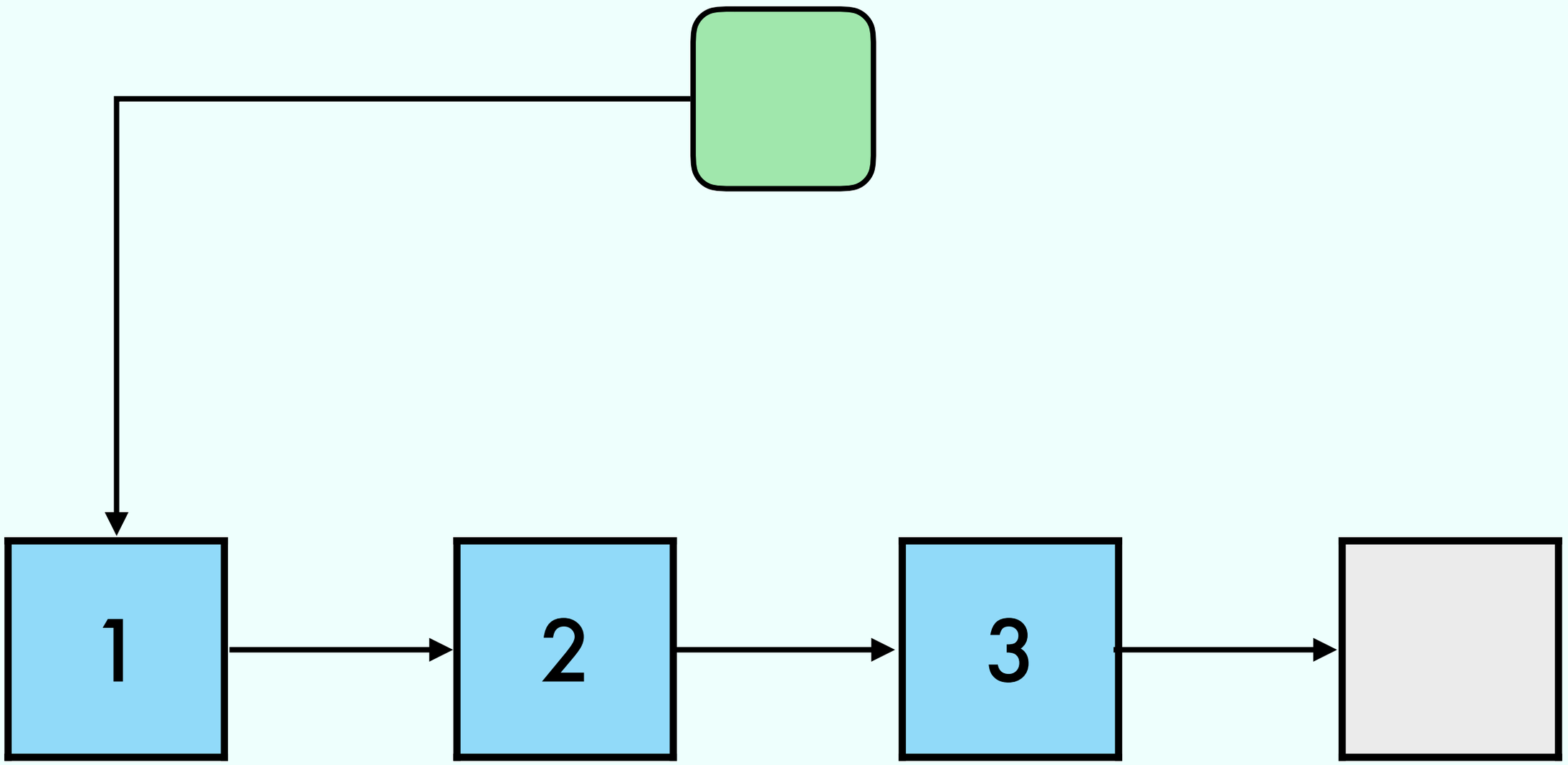


O(N)

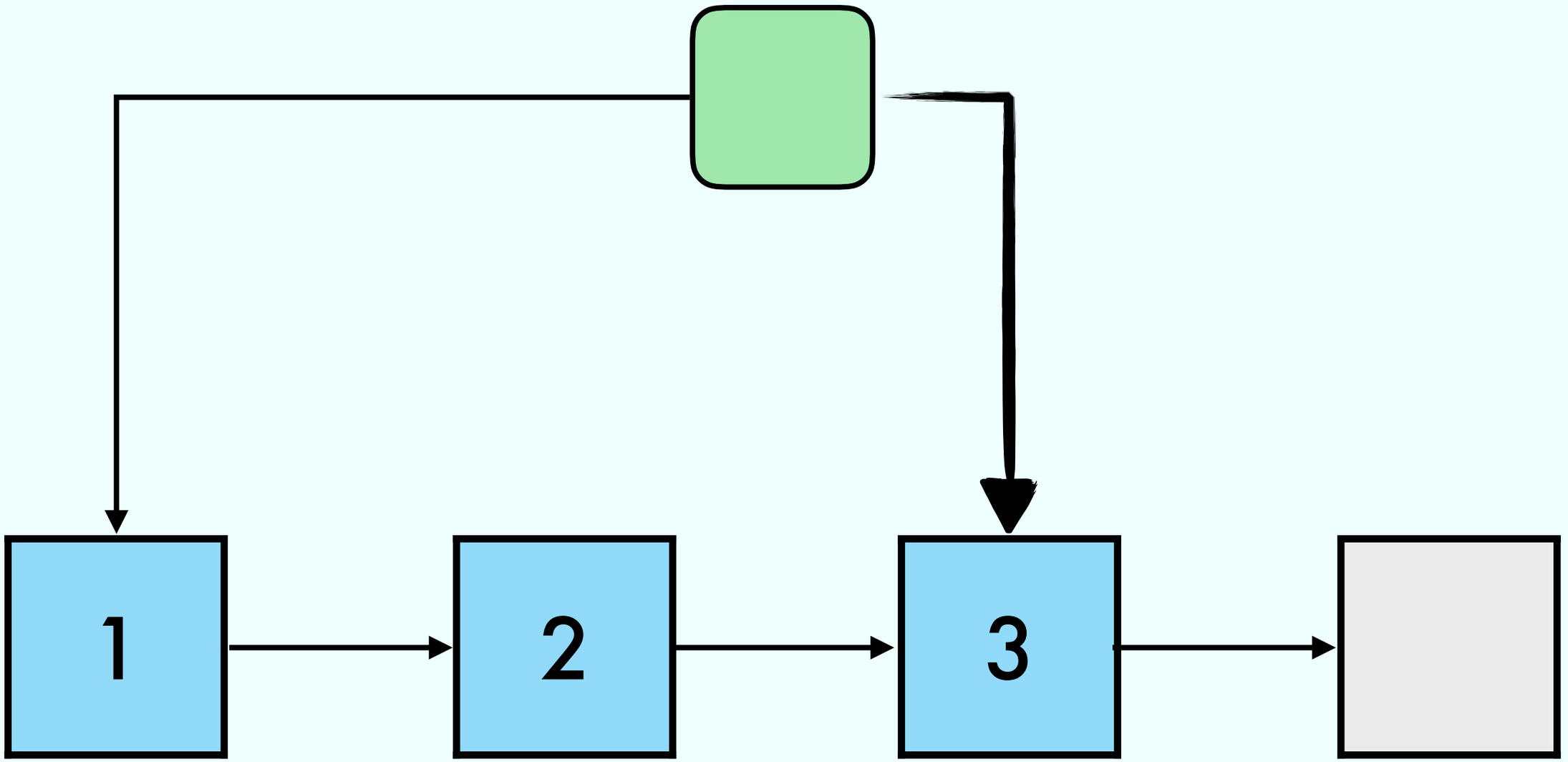
.append ( )



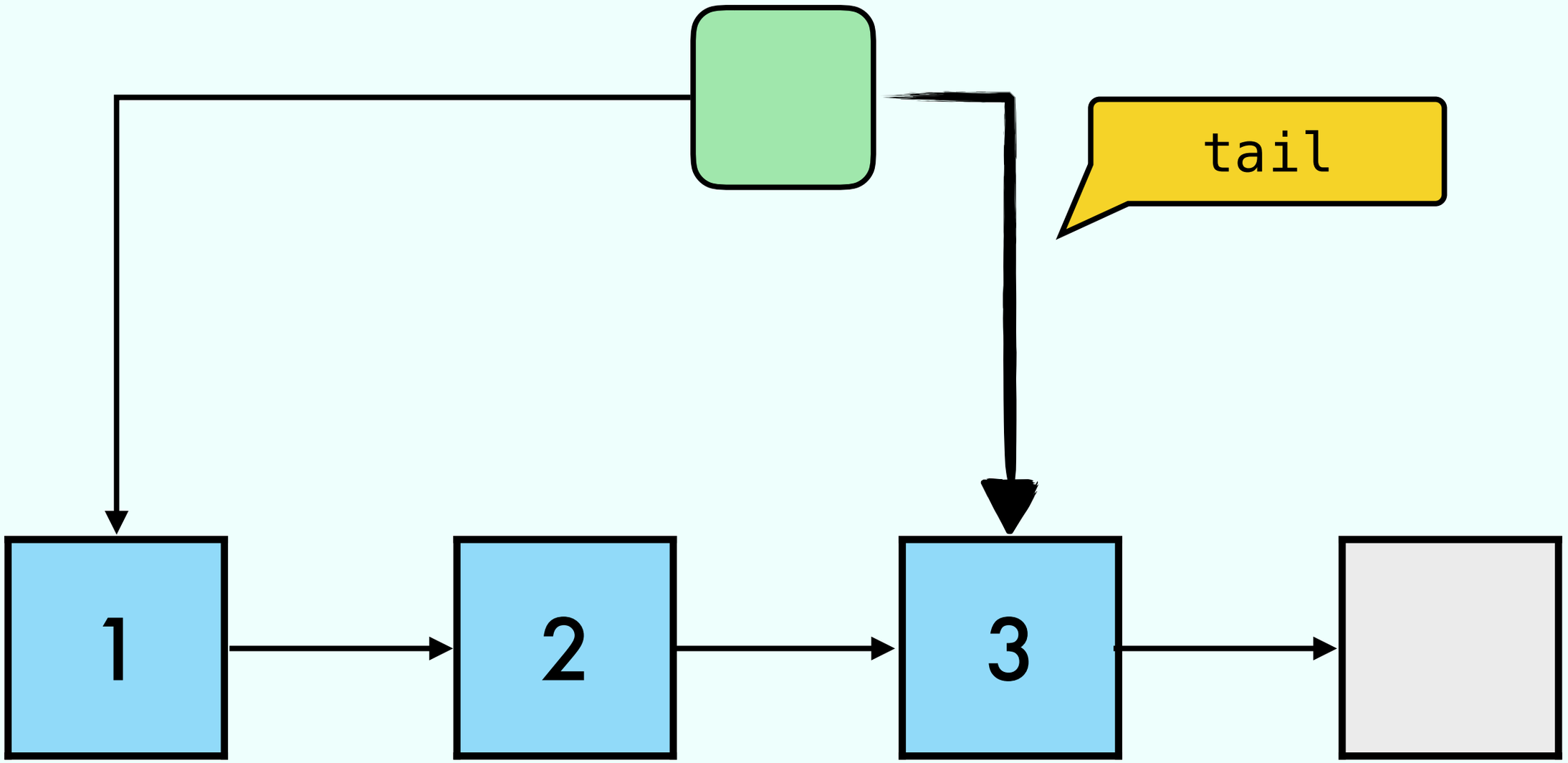
Linked List with pointer to last



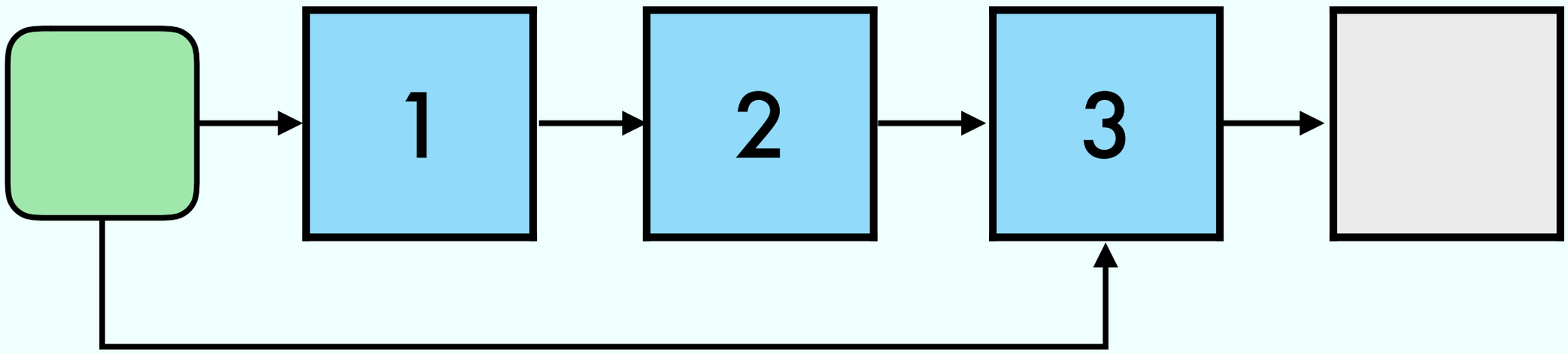
Linked List with pointer to last

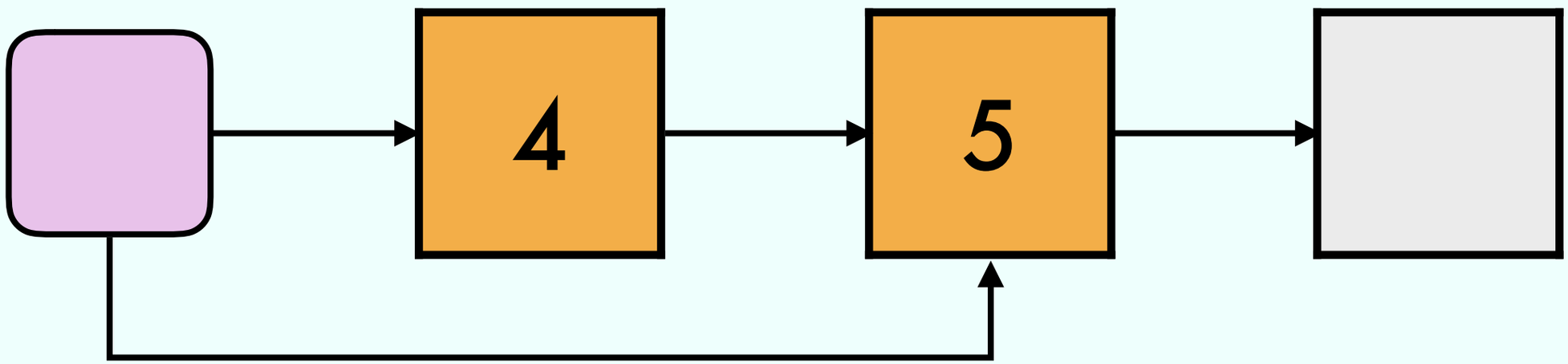


Linked List with pointer to last

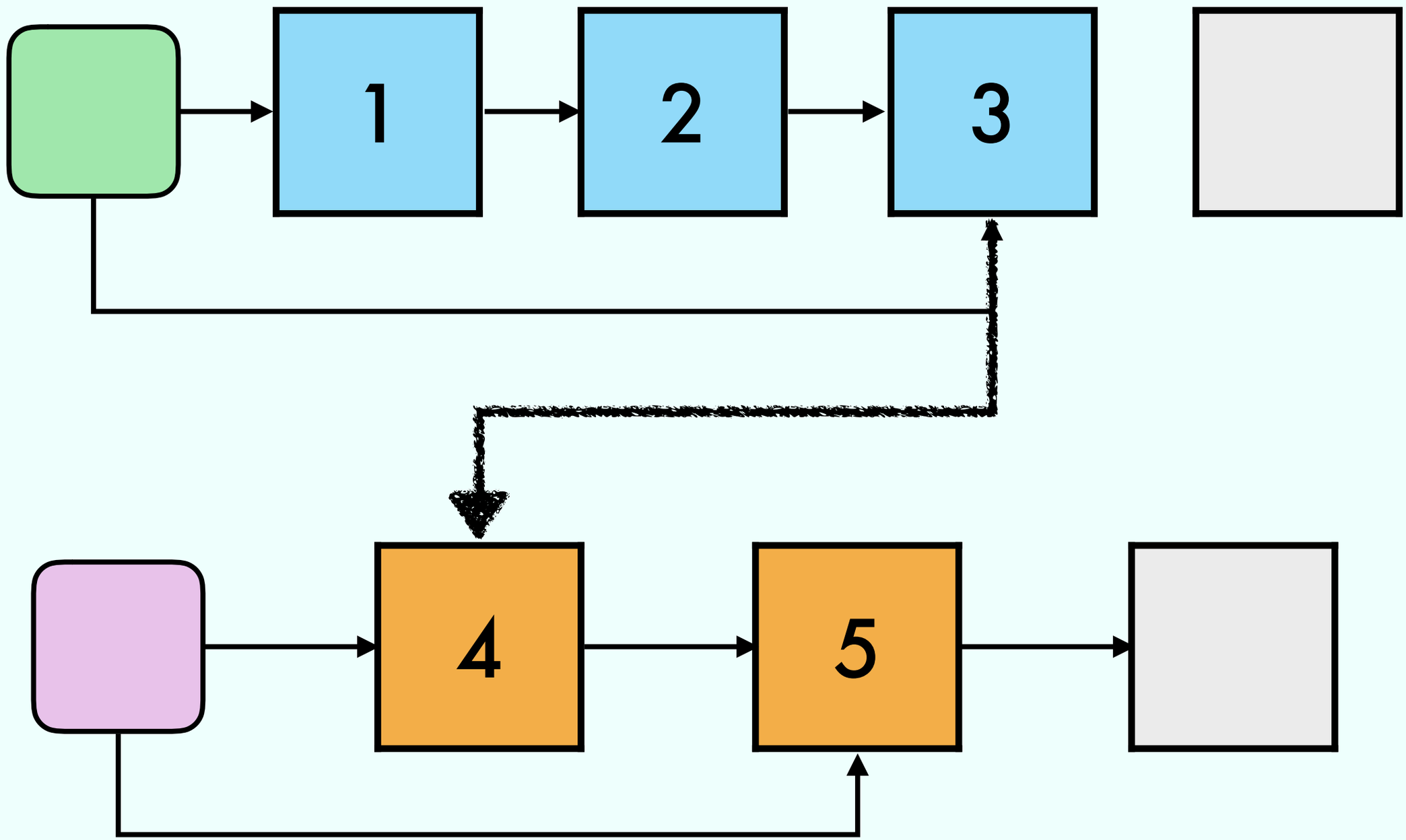


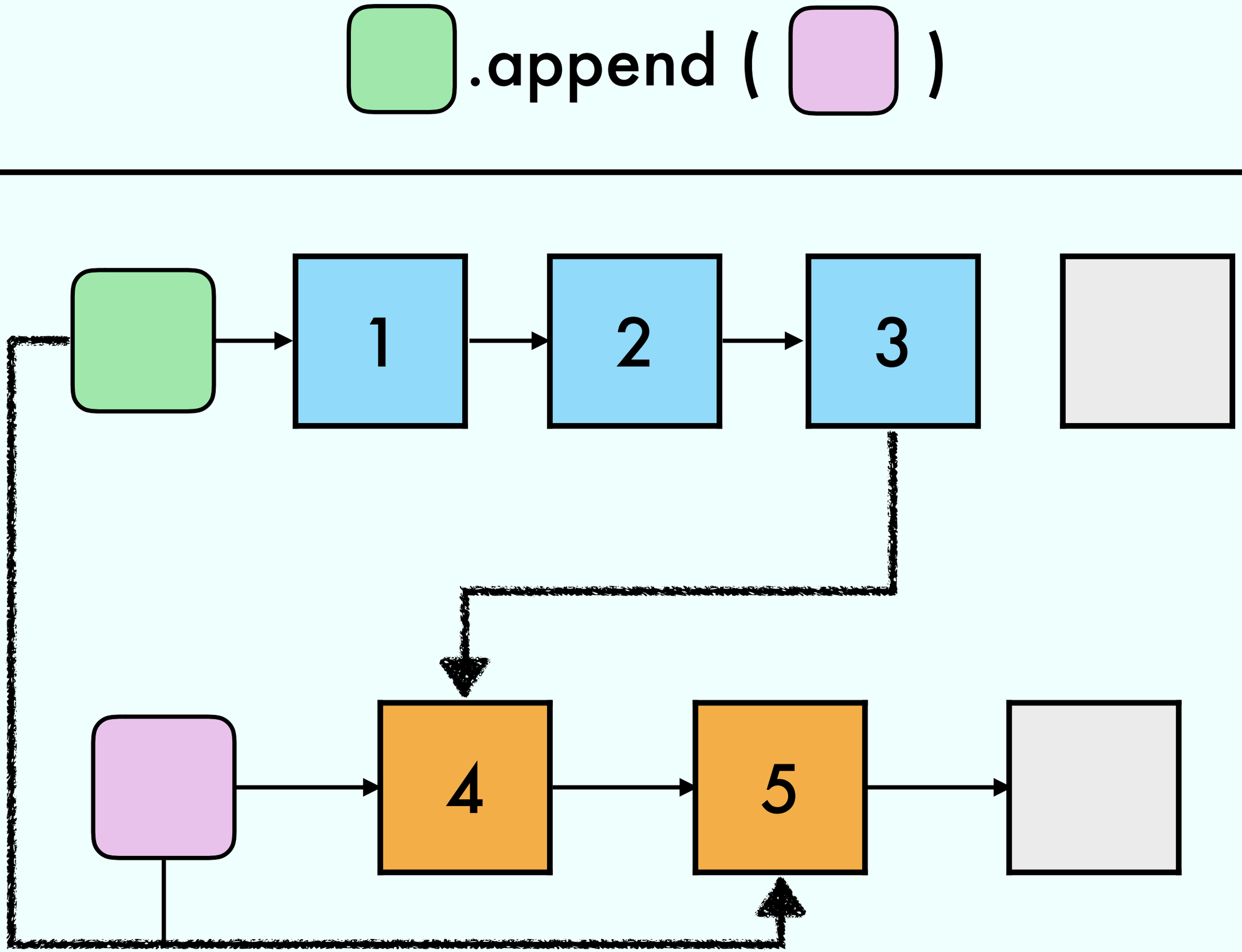
.append ( )





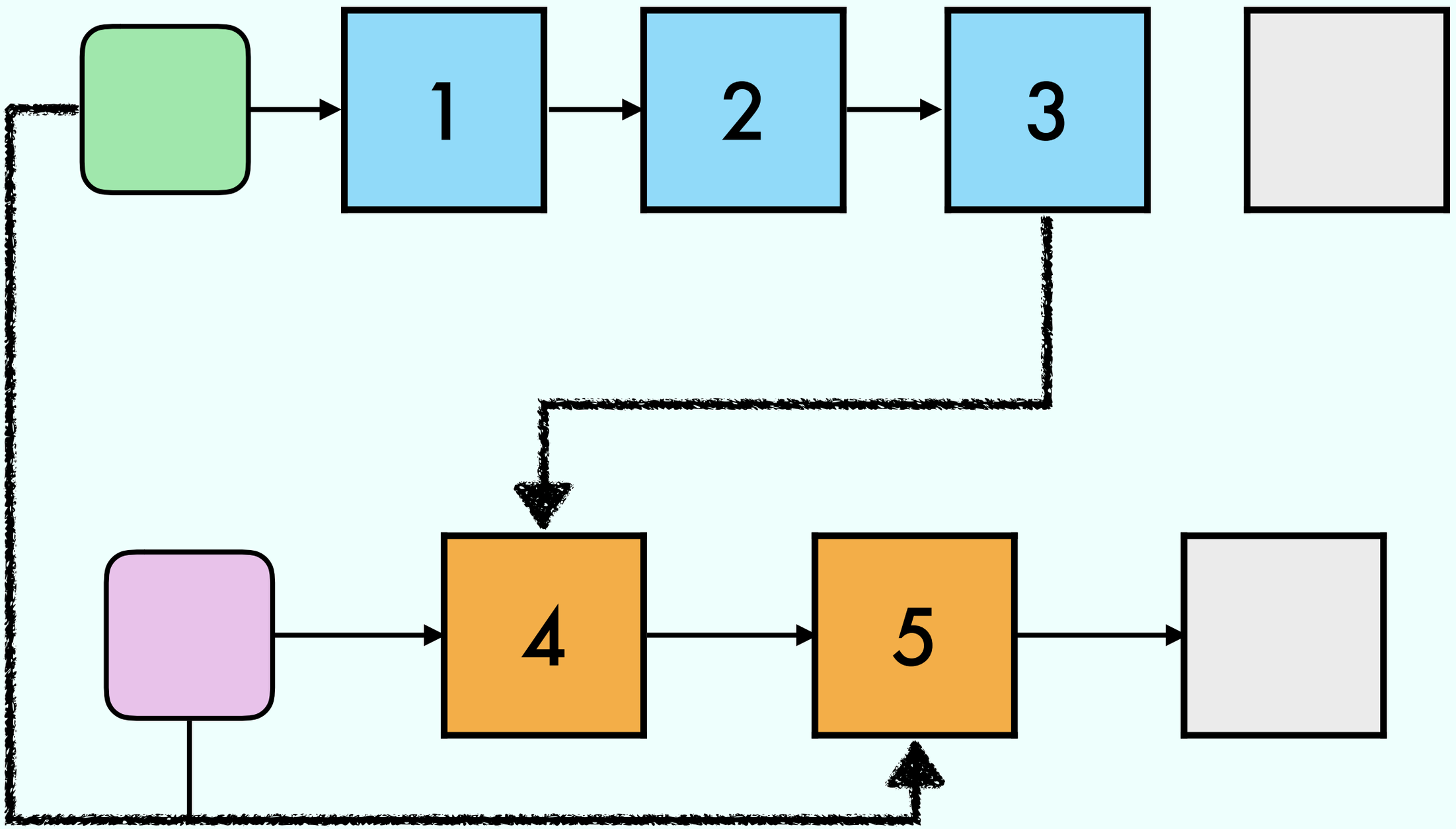
.append ( )




.append ( )
1
2
3
4
5

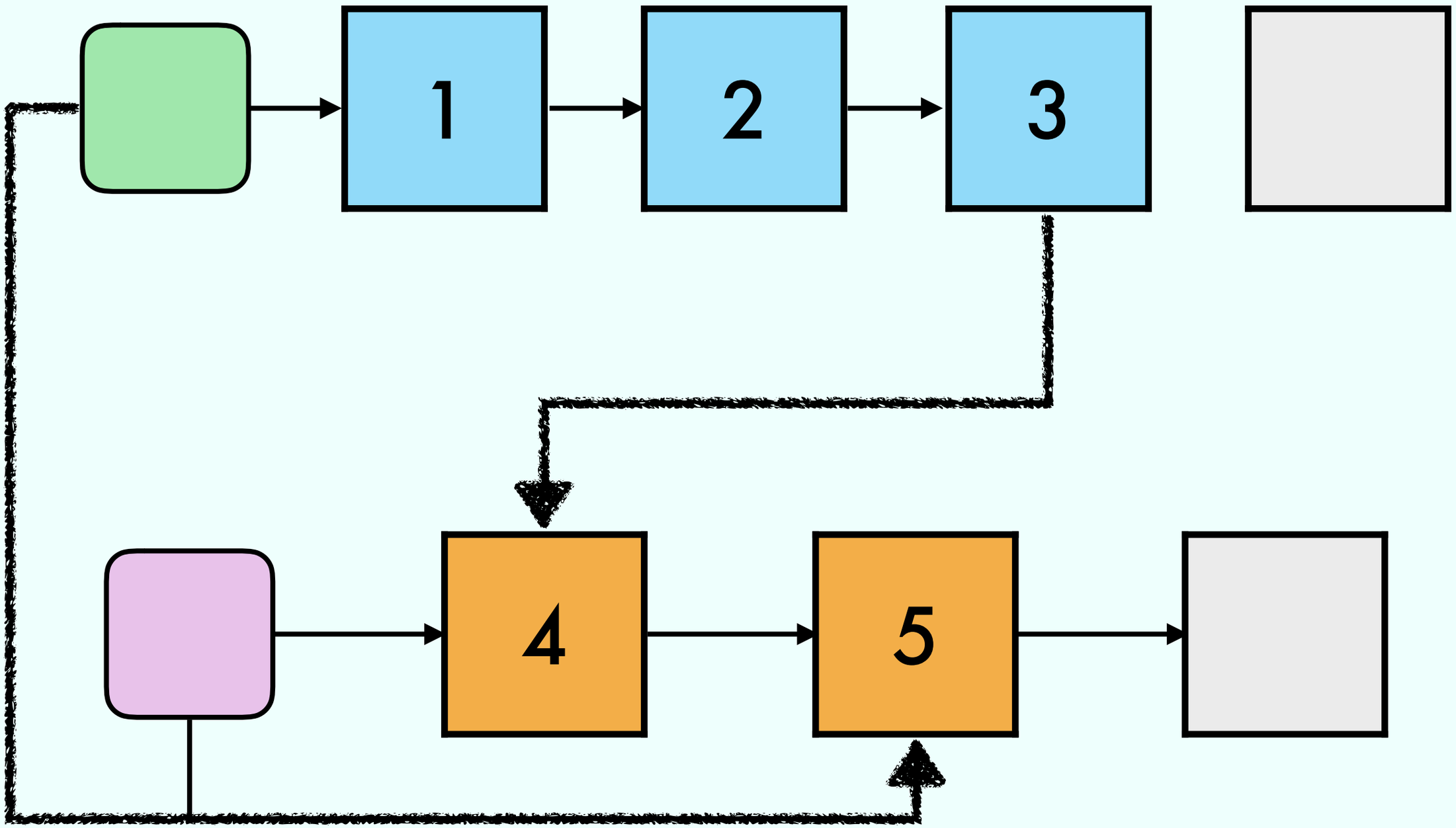
O(1)

.append ( )



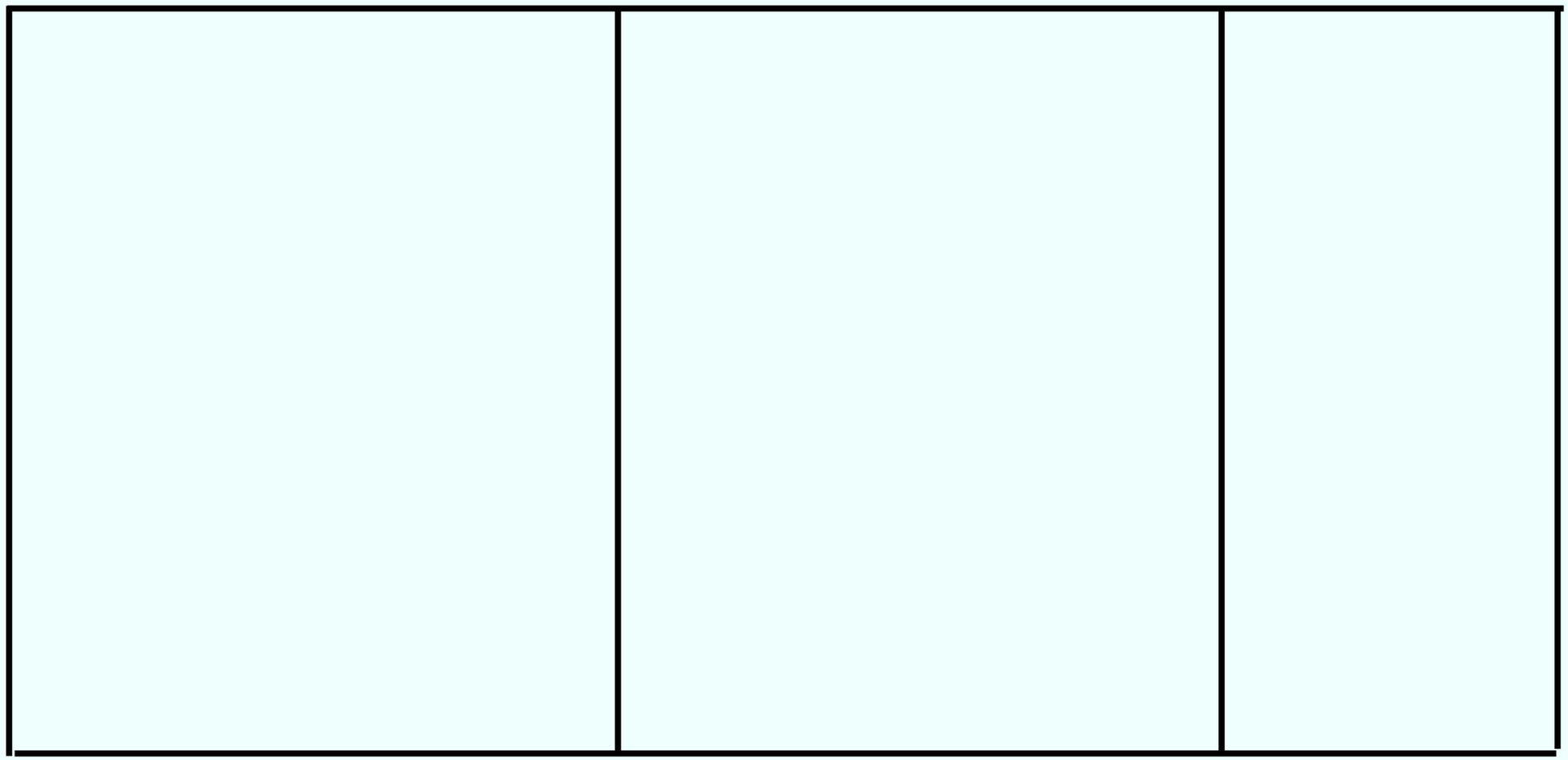
O(1)

.append ( )



Exercise 3.25a

Stack:



Exercise 3.25a

Stack:

push(x)		

Exercise 3.25a

Stack:

push(x)	O(1)	

Exercise 3.25a

Stack:

push(x)	O(1)	☑

Exercise 3.25a

Stack:

push(x) pop()	O(1)	

Exercise 3.25a

Stack:

push(x)	O(1)	☑
pop()	O(1)	

Exercise 3.25a

Stack:

push(x)	O(1)	☑
pop()	O(1)	☑

Exercise 3.25a

Stack:

push(x)	O(1)	☑
pop()	O(1)	☑
findMin()		

Exercise 3.25a

Stack:

push(x)	O(1)	☑
pop()	O(1)	☑
findMin()	O(1)	

Exercise 3.25a

Stack:

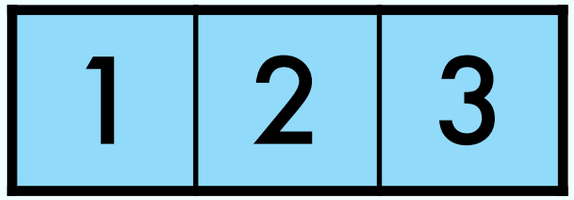
push(x)	O(1)	☑
pop()	O(1)	☑
findMin()	O(1)	

Exercise 3.29

Print a singly linked list in reverse in constant space:

Exercise 3.29

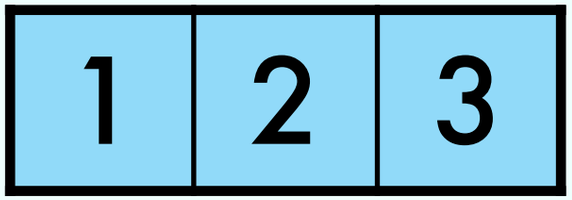
Print a singly linked list in reverse in constant space:



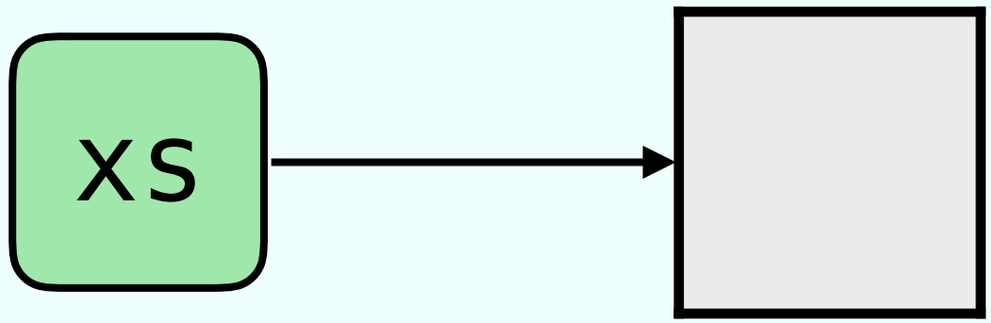
.printRev () // O(1) memory

Exercise 3.29

Print a singly linked list in reverse in constant space:



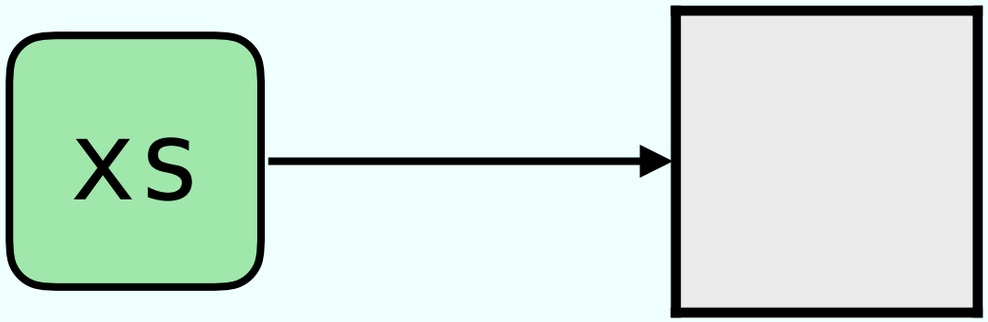
.printRev () // O(1) memory



here

prev

next

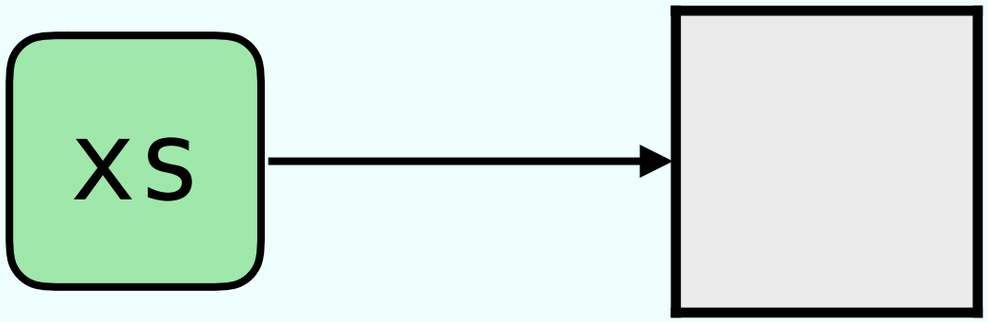


here

prev

next

Init



here

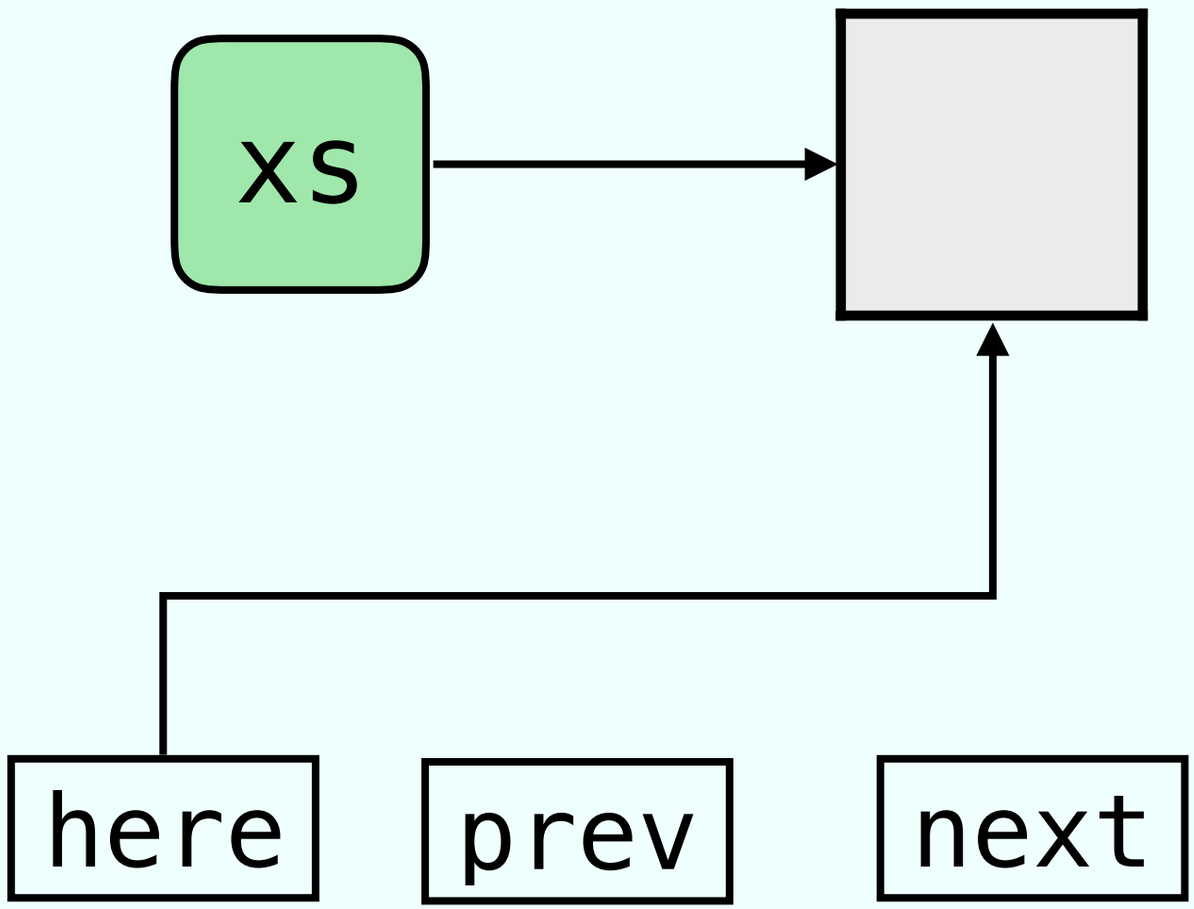
prev

next

Init

here = xs.head

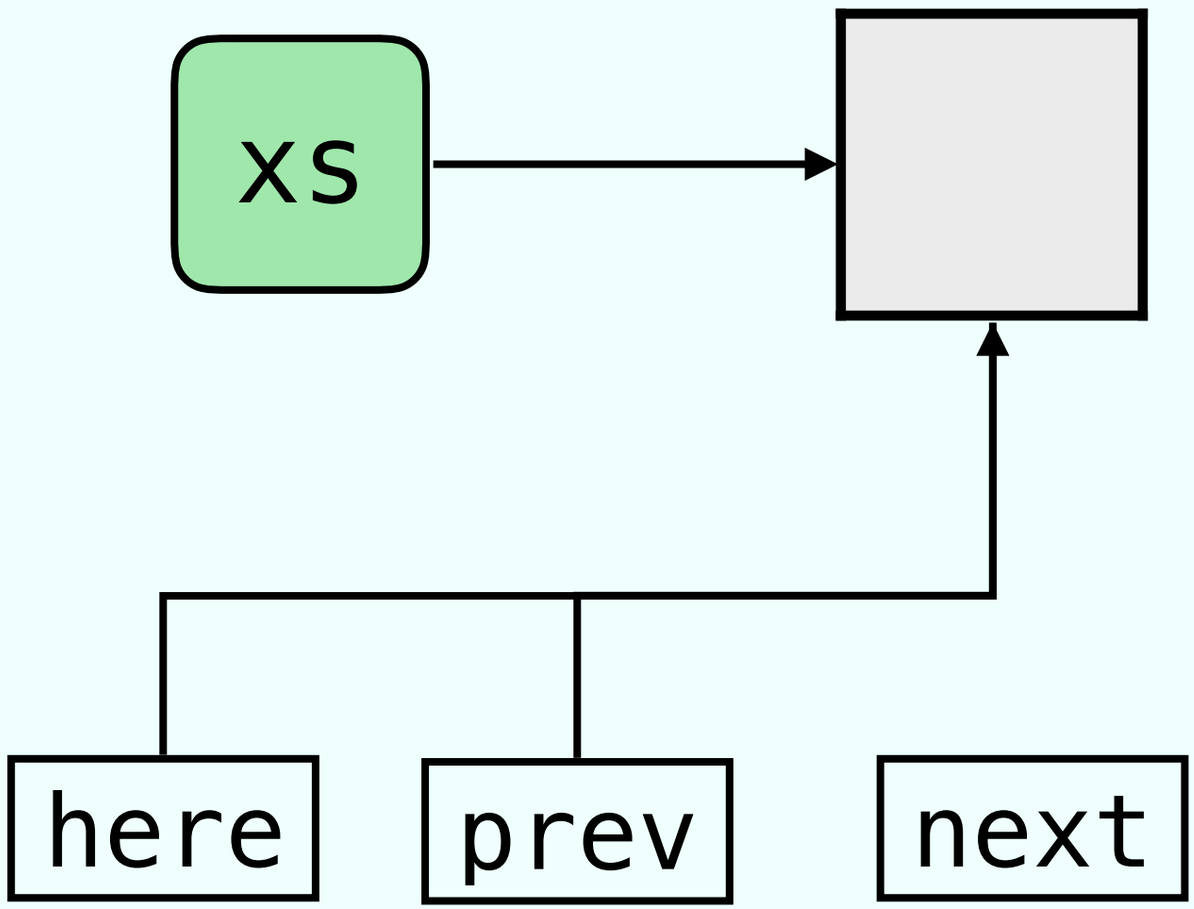
prev = null



Init

here = xs.head

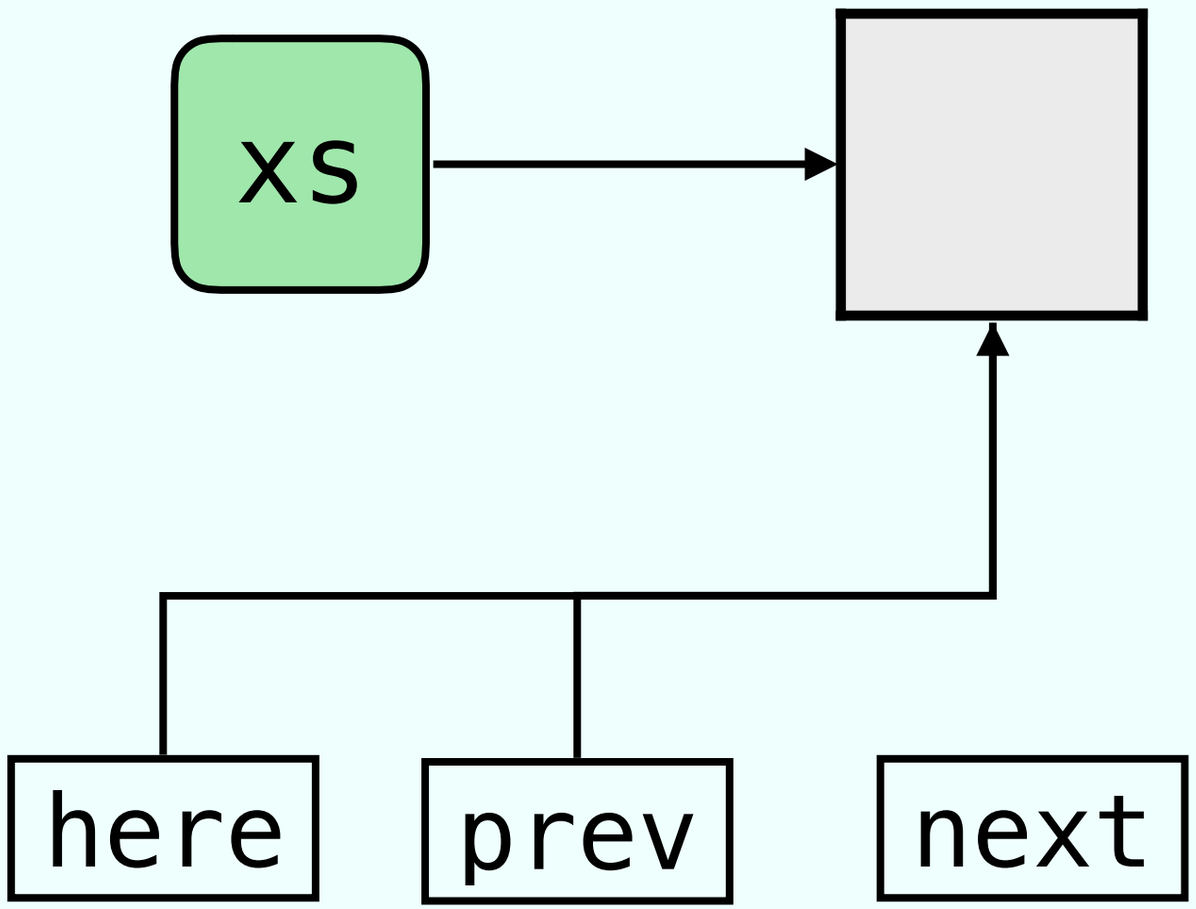
prev = null



Init

here = xs.head

prev = null

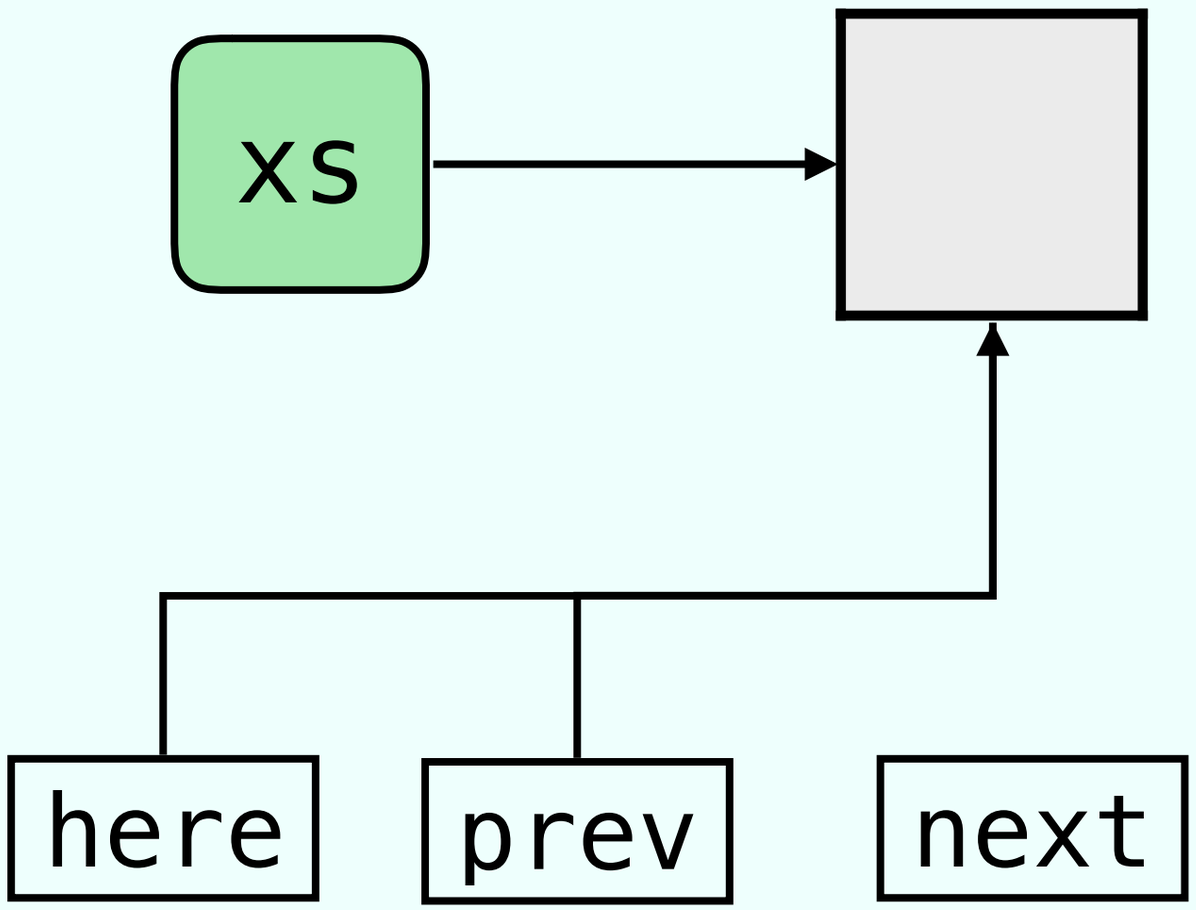


Init

here = xs.head

prev = null

Rev Loop

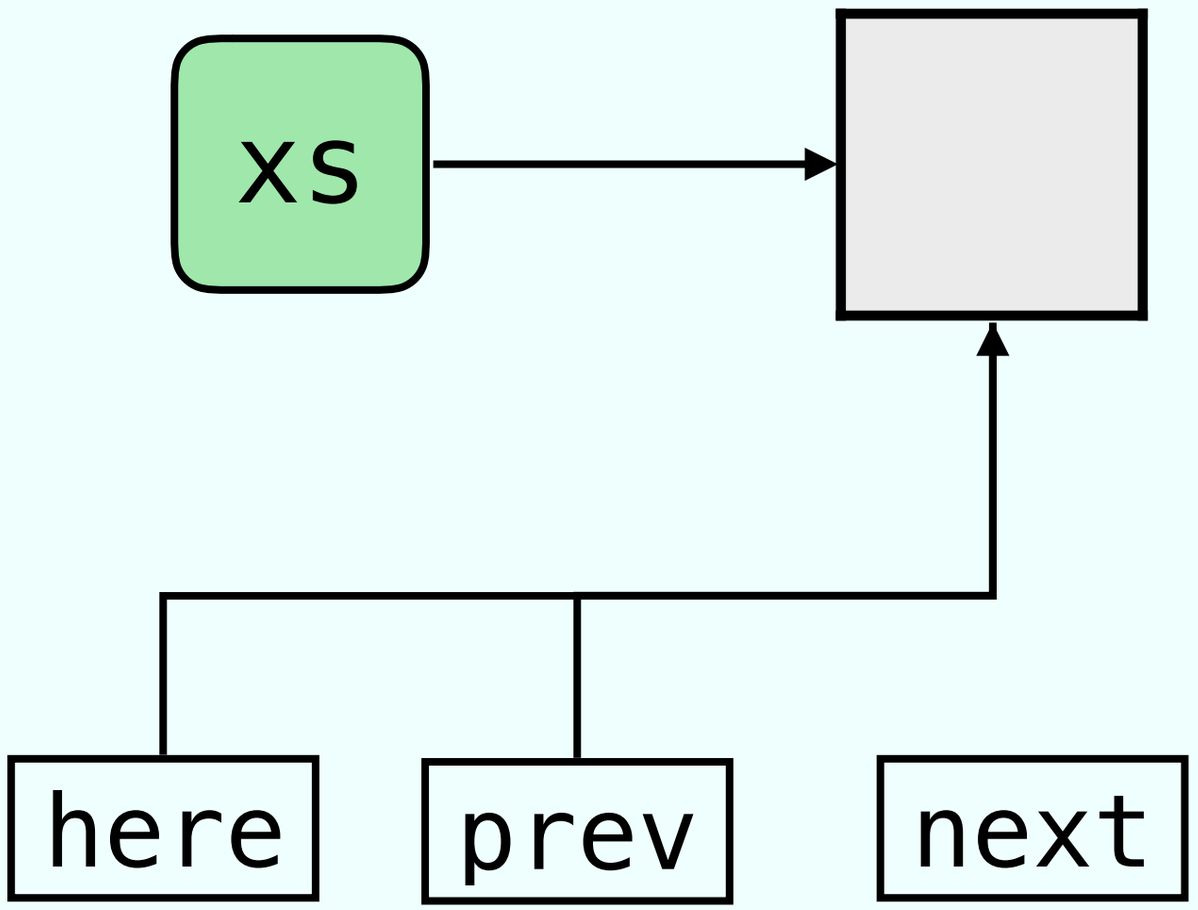


Init

```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```



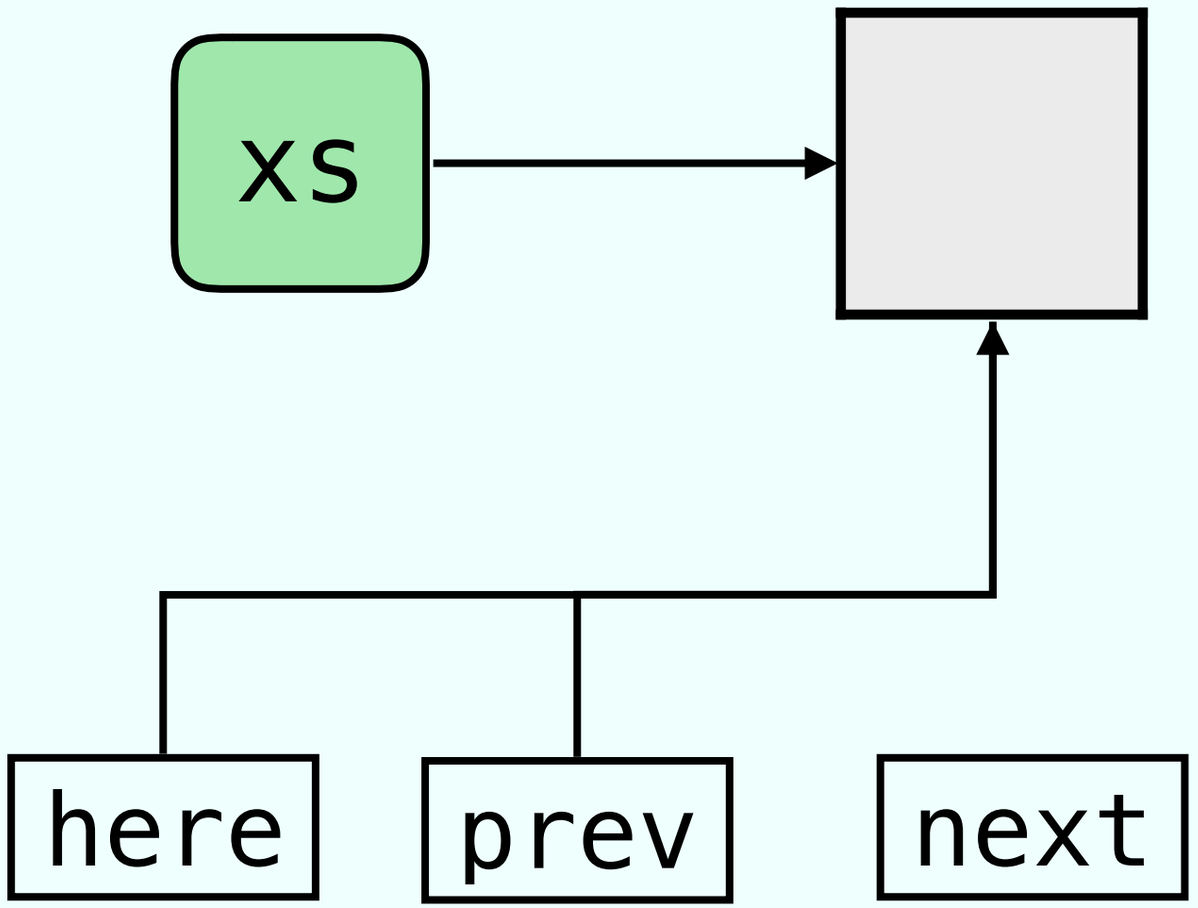
Init

```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc



Init

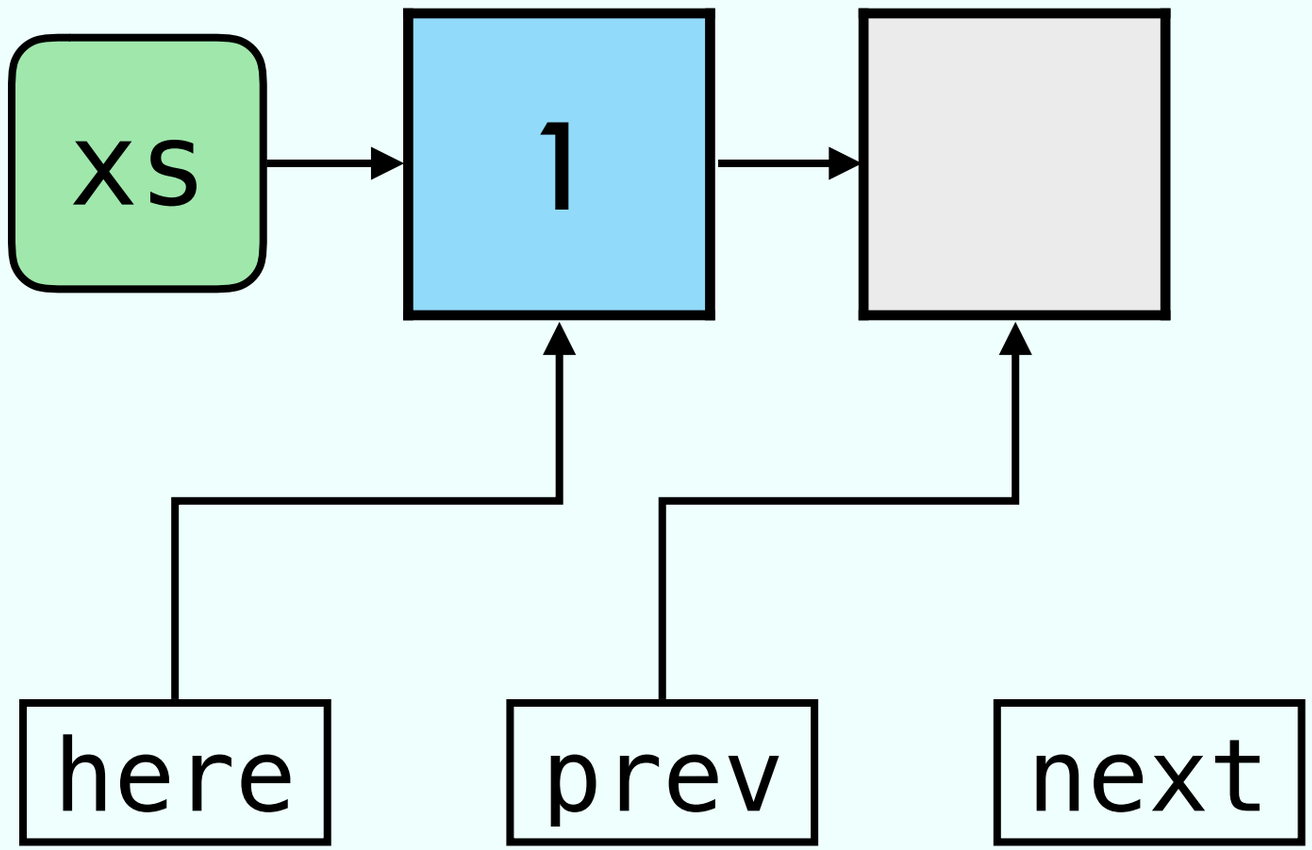
```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

```
xs.head = prev
```



Init

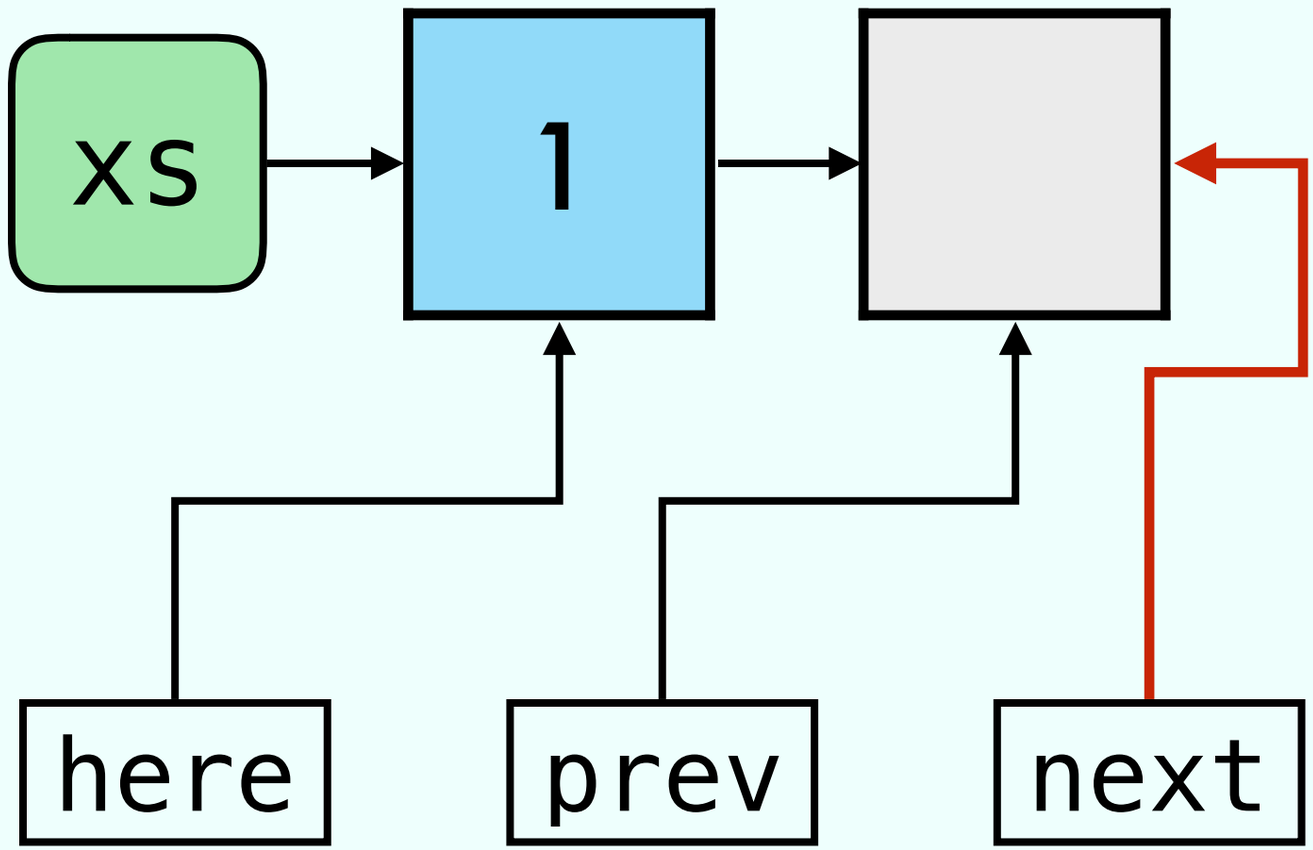
```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

```
xs.head = prev
```



Init

here = xs.head

prev = null

Rev Loop

while (here ≠ null) do

next = here.next

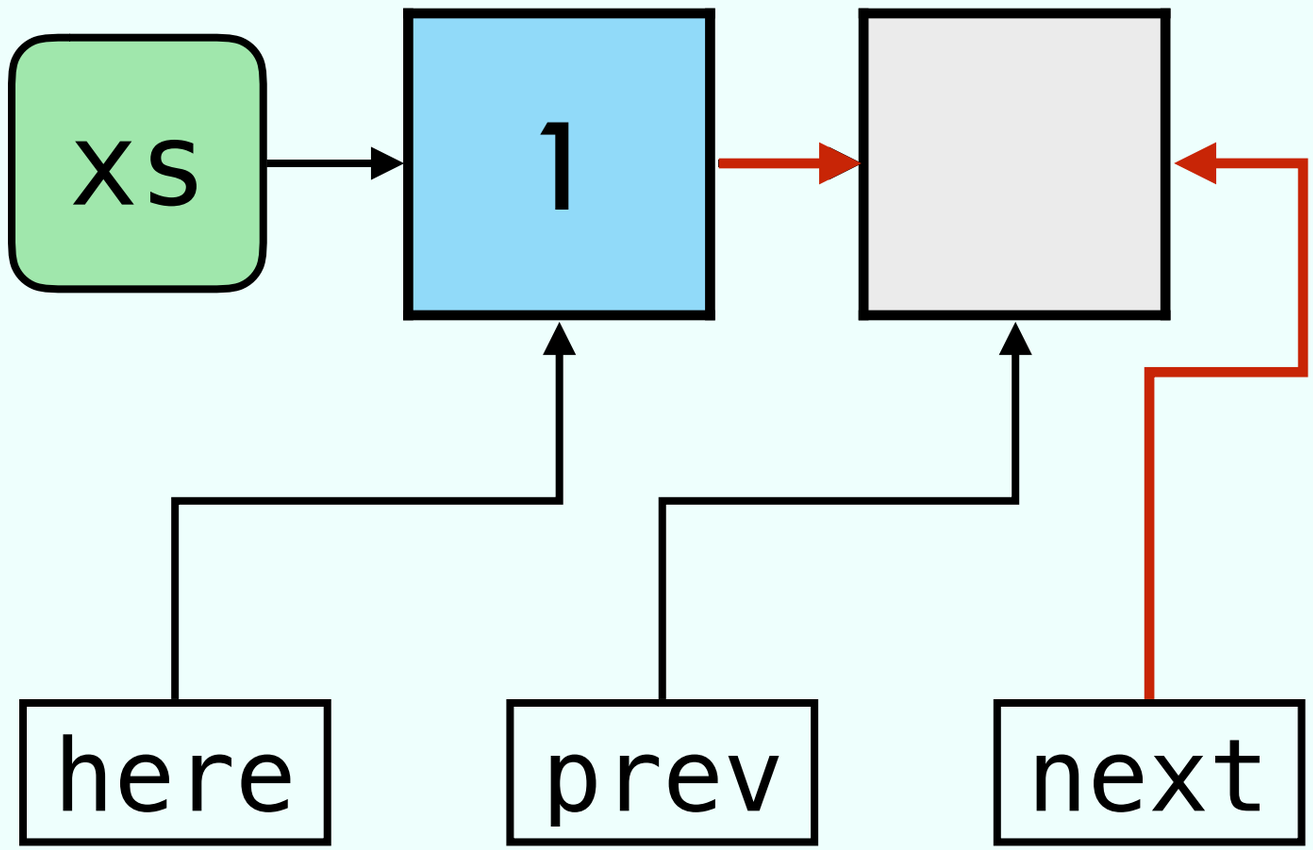
here.next = prev

prev = here

here = next

Conc

xs.head = prev



Init

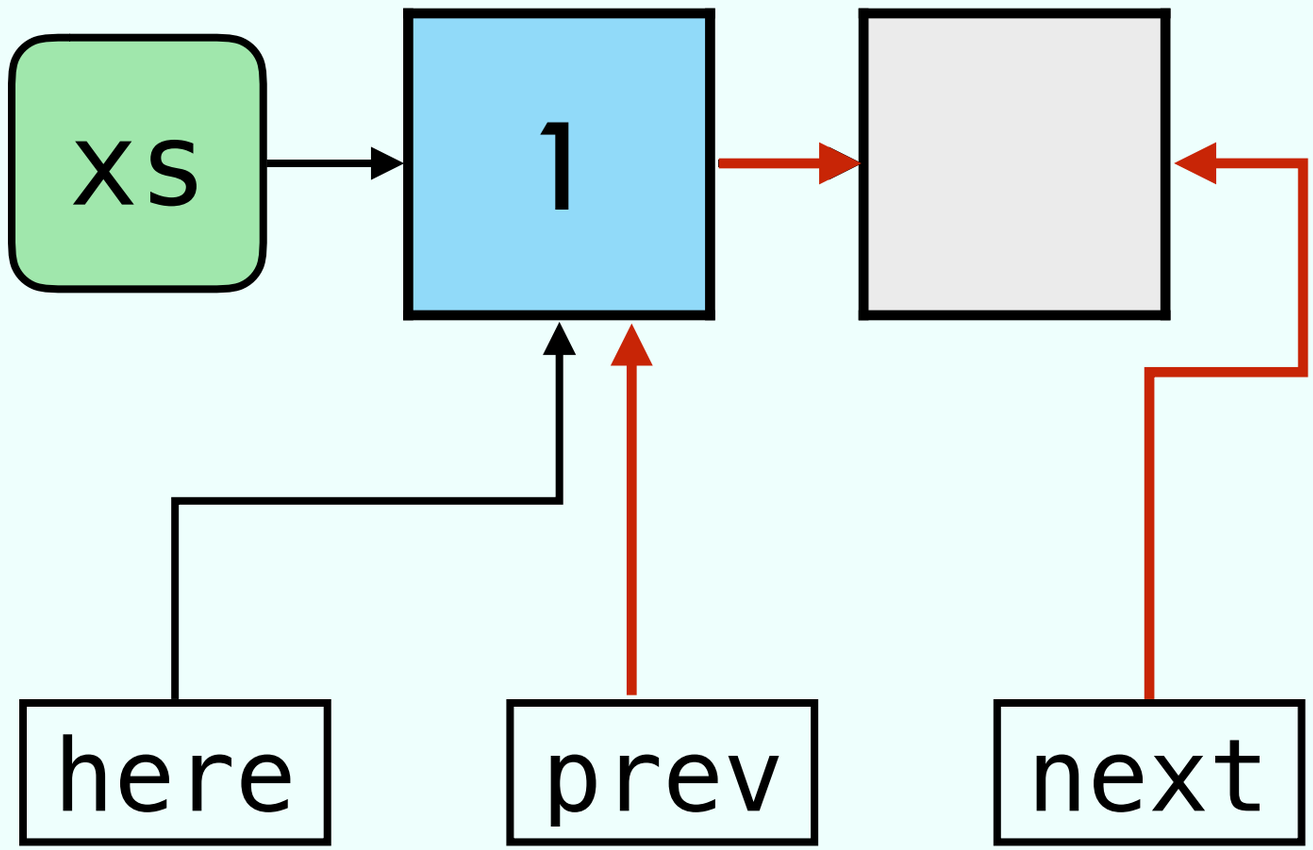
```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

```
xs.head = prev
```



Init

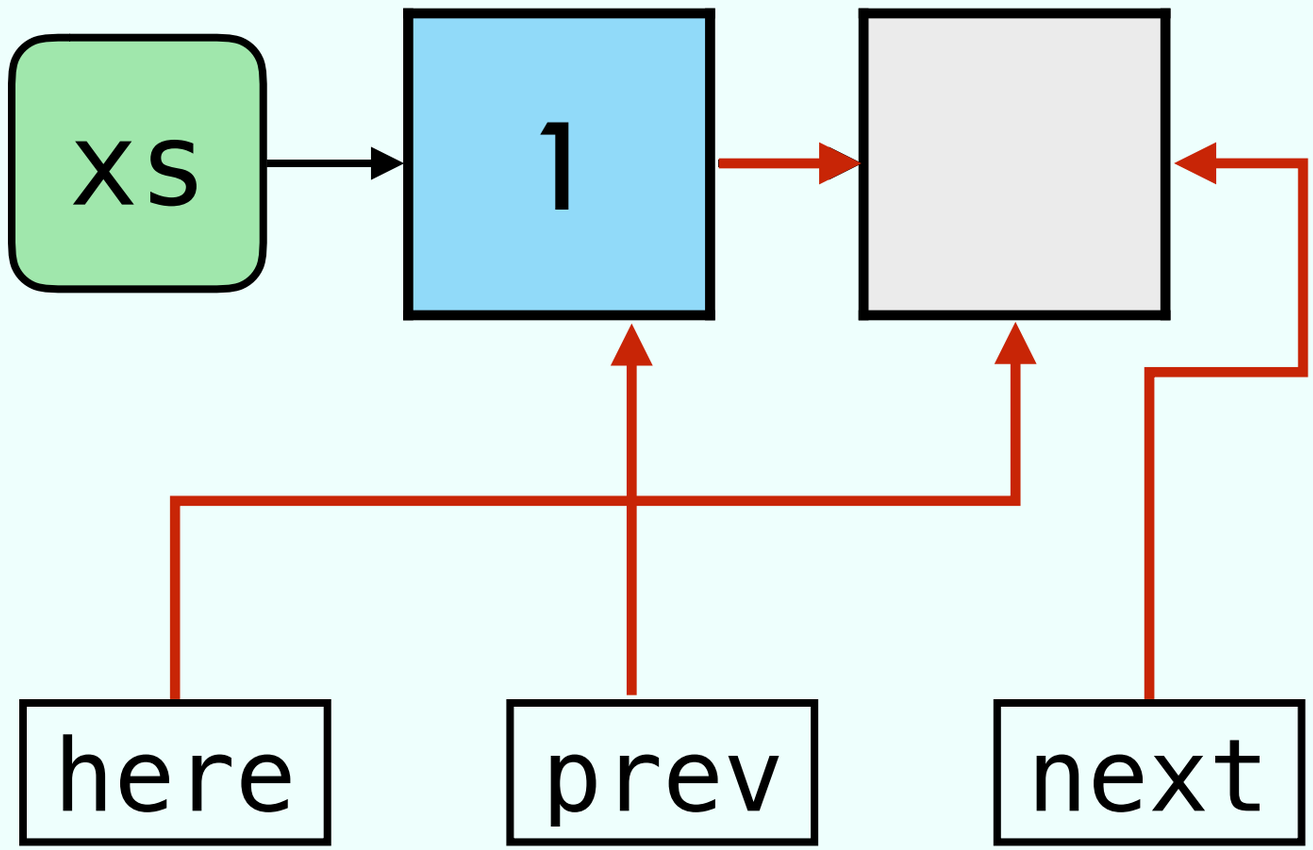
```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

```
xs.head = prev
```



Init

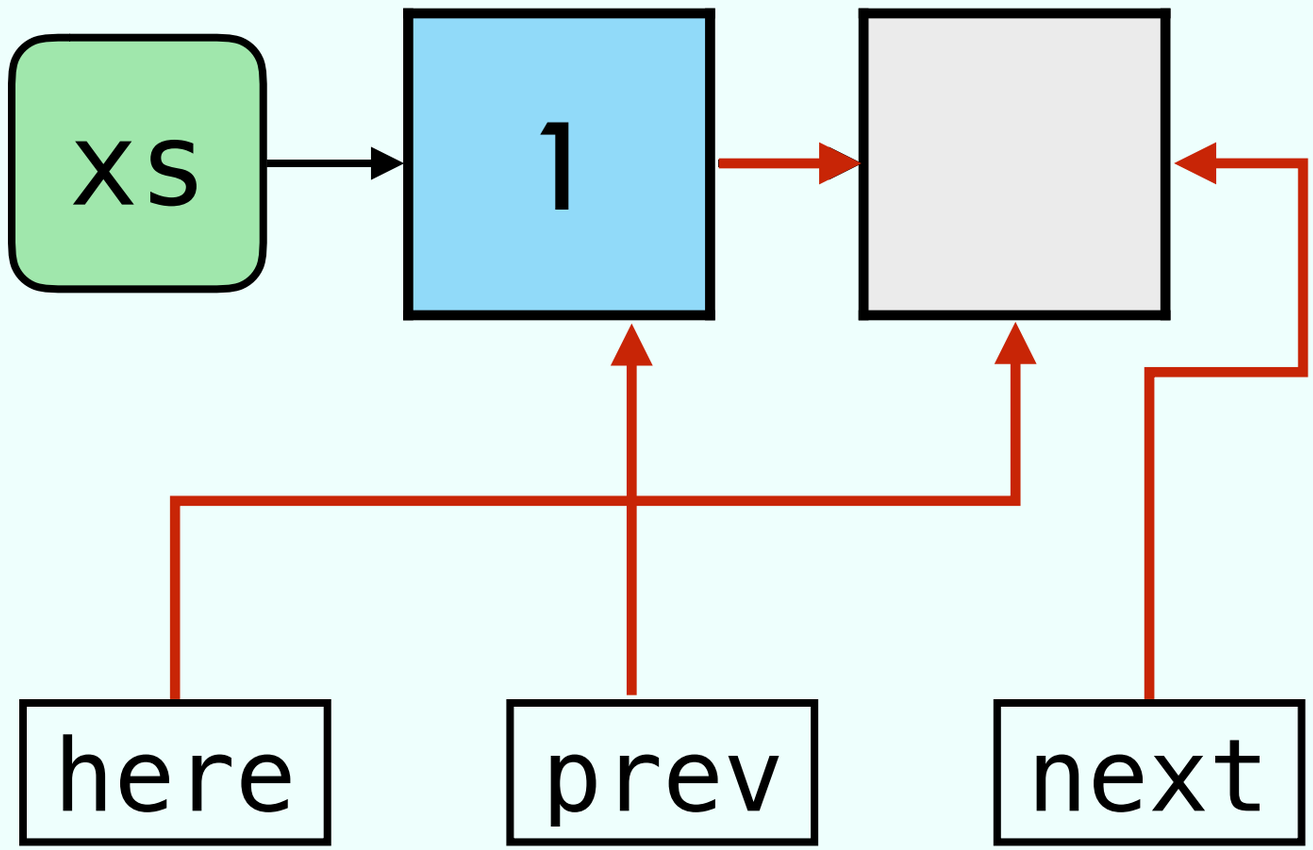
```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

```
xs.head = prev
```



Init

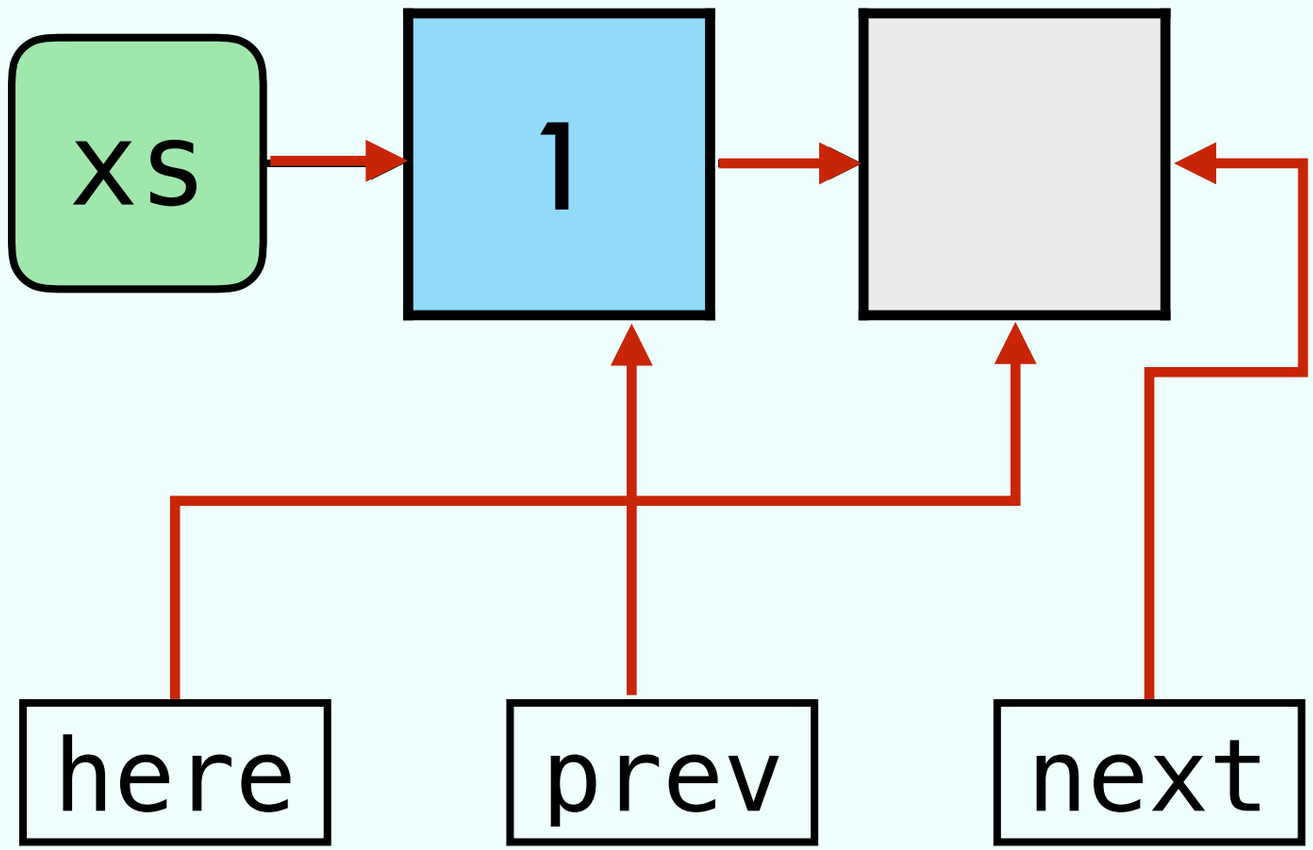
```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

```
xs.head = prev
```



Init

```
here = xs.head
prev = null
```

Rev Loop

```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```

Conc

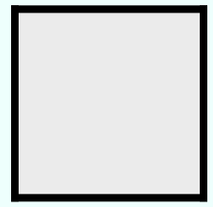
```
xs.head = prev
```

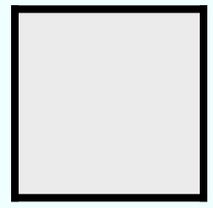
Exercise 5 from13/04

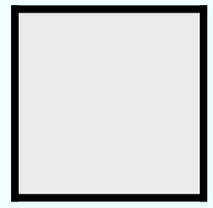
Dynamic Array:

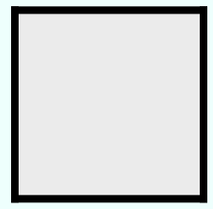
- `ins(x)` // Insert in first empty position
- `del()` // Removes the last element

Operations	Dynamic Array

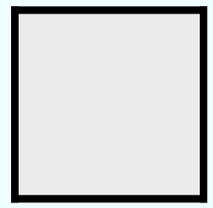
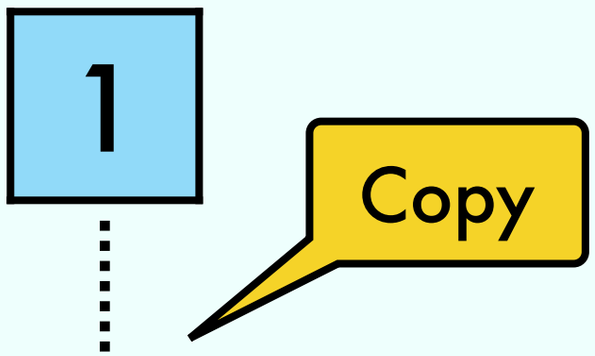
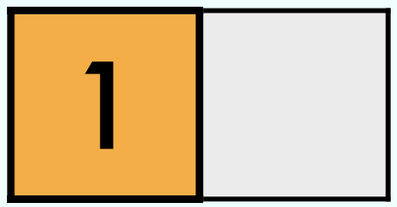
Operations	Dynamic Array
new( )	

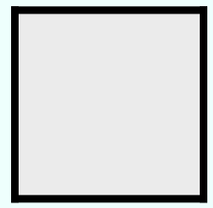
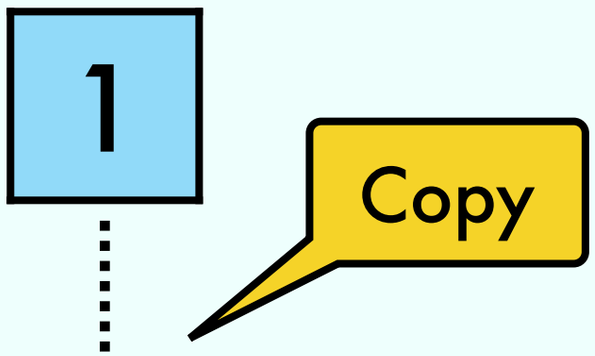
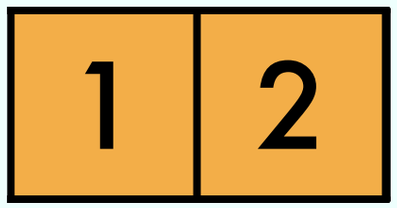
Operations	Dynamic Array
`new()`	
`ins(1)`	

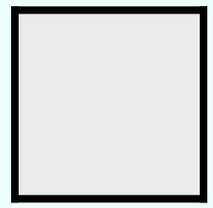
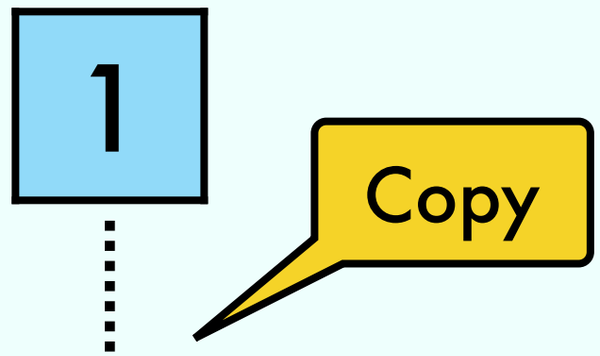
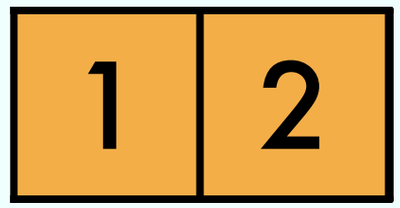
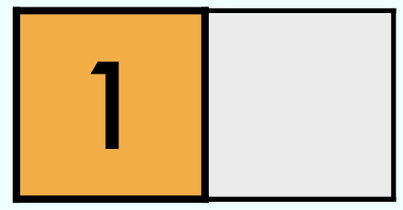
Operations	Dynamic Array
`new()`	
`ins(1)`	
`ins(2)`	

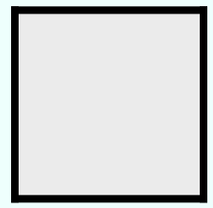
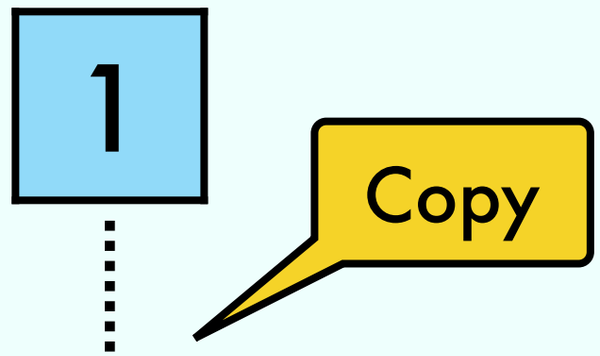
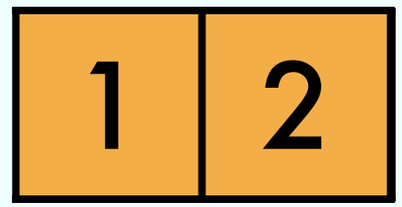
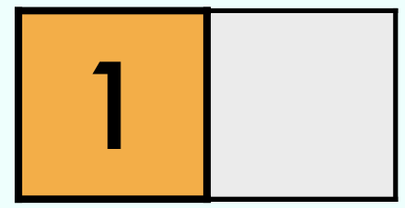
Operations	Dynamic Array
`new()`	
`ins(1)`	
`ins(2)`	

Double the size if full

Operations	Dynamic Array
`new()`	
`ins(1)`	
Double the size if full	
`ins(2)`	

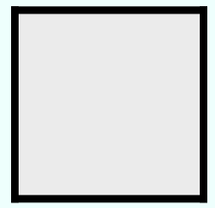
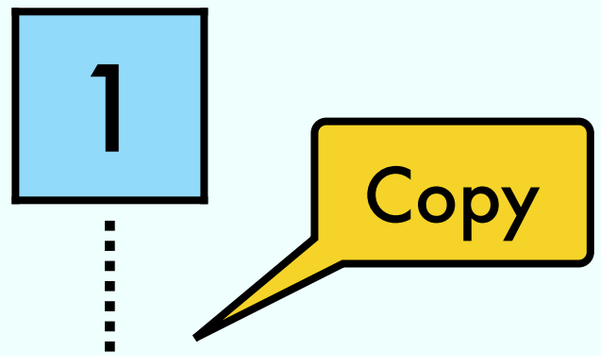
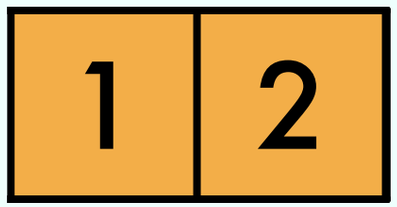
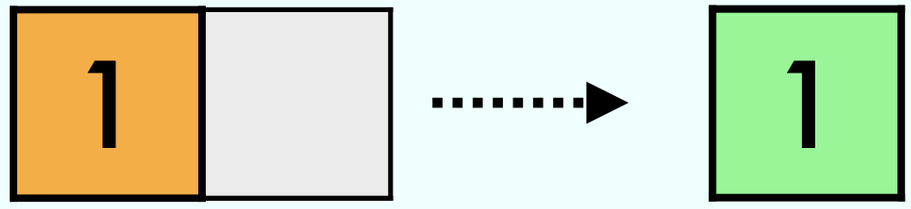
Operations	Dynamic Array
`new()`	
`ins(1)`	
Double the size if full	
`ins(2)`	

Operations	Dynamic Array
`new()`	
`ins(1)`	
Double the size if full	
`ins(2)`	
`del()`	

Operations	Dynamic Array
`new()`	
`ins(1)`	
`ins(2)`	
`del()`	

Double the size if full

Resize if half-empty

Operations	Dynamic Array
`new()`	
`ins(1)`	
Double the size if full	
`ins(2)`	
Resize if half-empty	
`del()`	

Exercise 5 from13/04

For every N exists

S_N : Sequence of N operations

such that

$T(S_N) = \Omega(N^2)$