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 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);
```

O(1)

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

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);
```

Exactly N times

O(1)

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

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);
```

Exactly M times

Exactly N times

O(1)

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

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);
```

Exactly M times

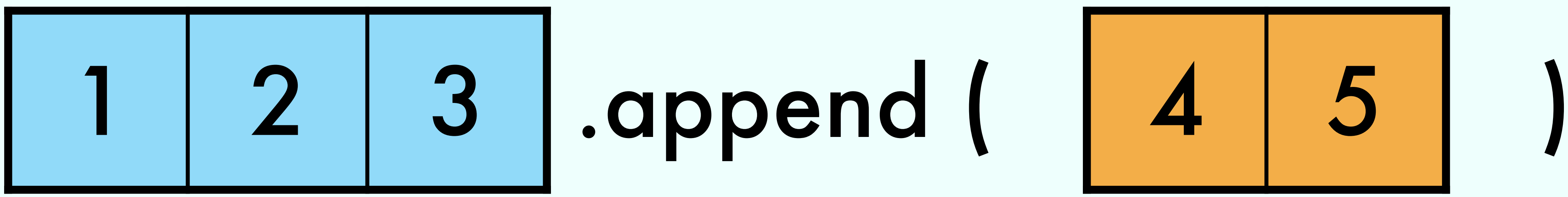
Exactly N times

O(1)

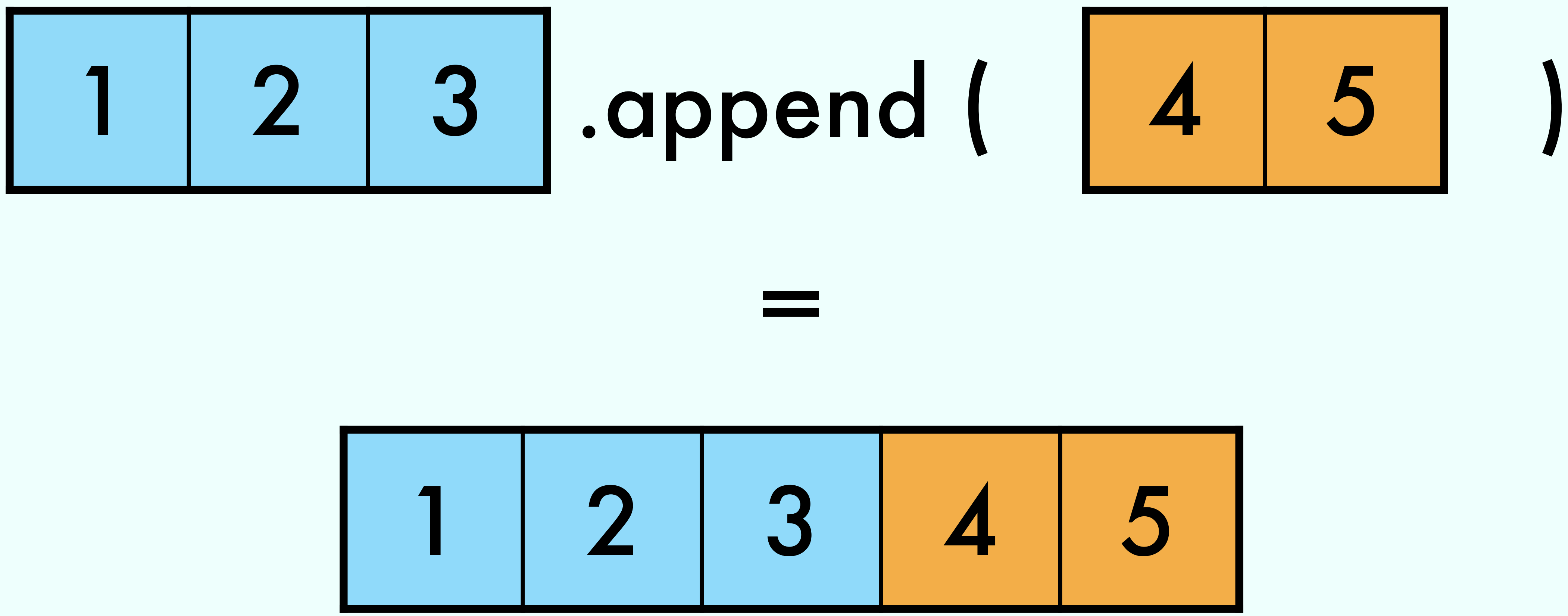
in terms of `M`, `N` and `|stack|`

$\Theta(MN)$

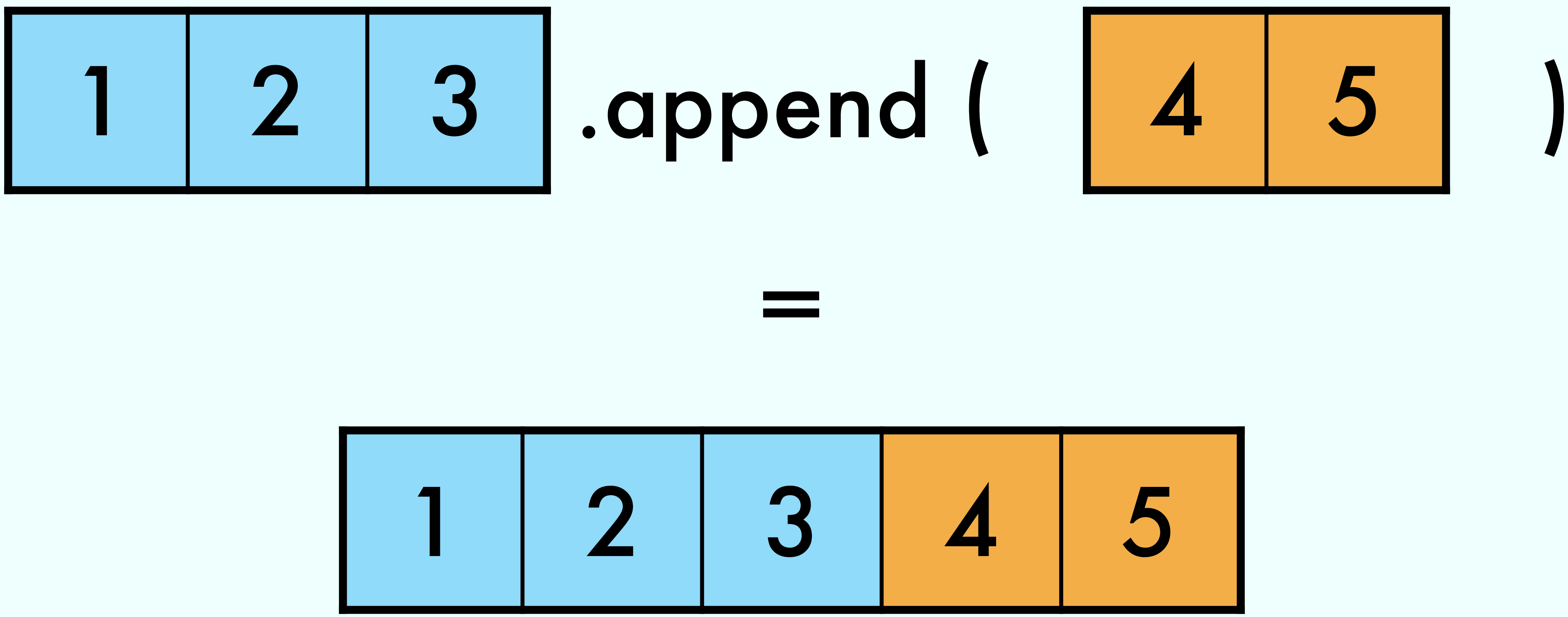
Exercise 3 from 12/04



Exercise 3 from 12/04



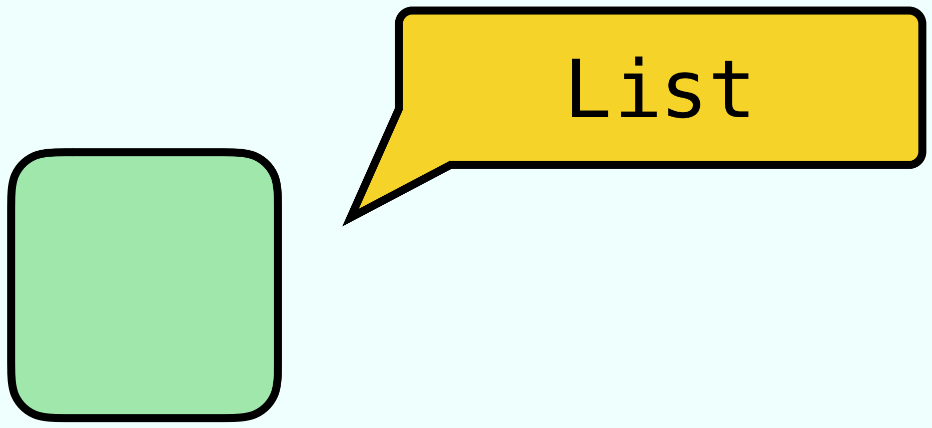
Exercise 3 from 12/04



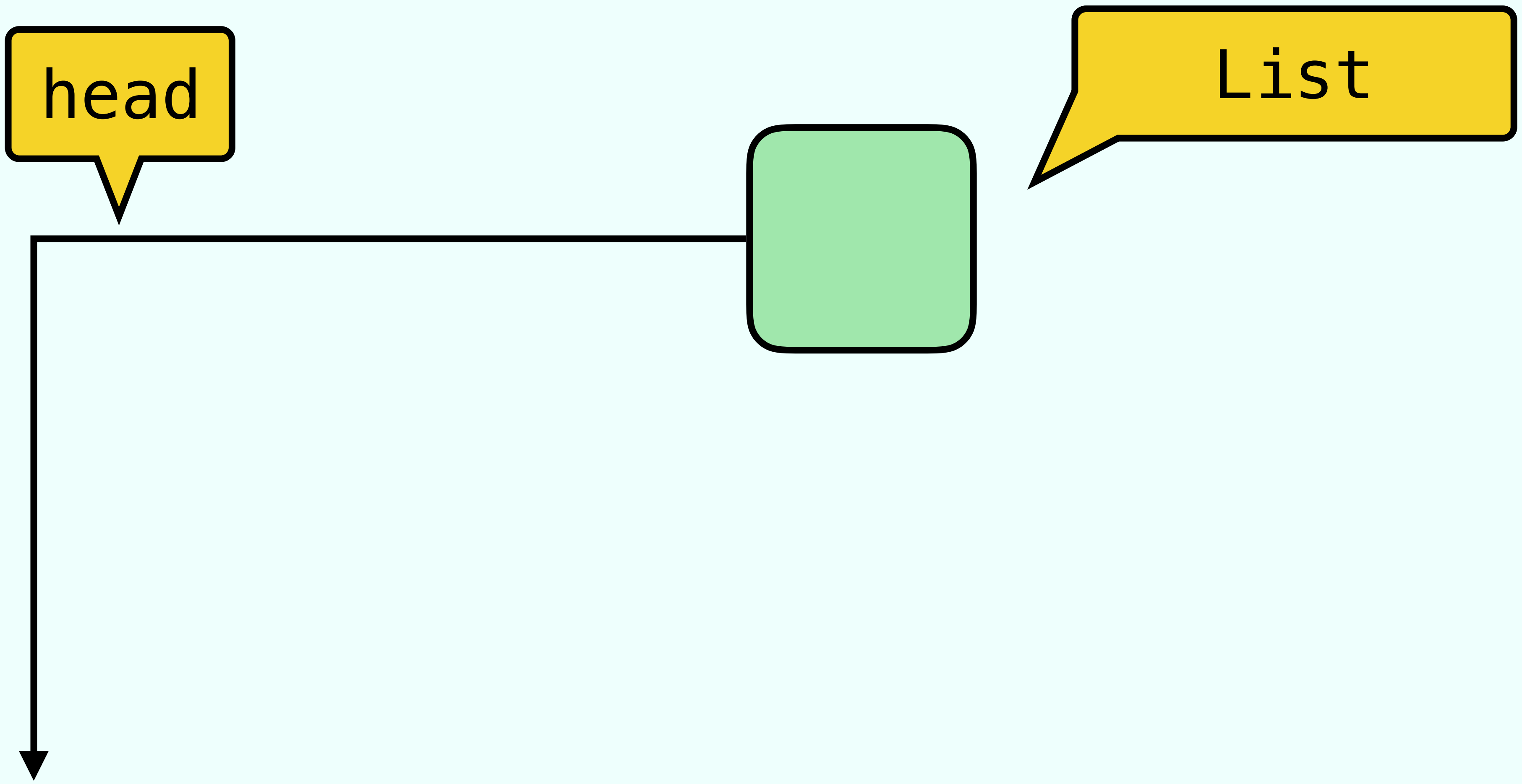
Implement append in O(1)

Linked List

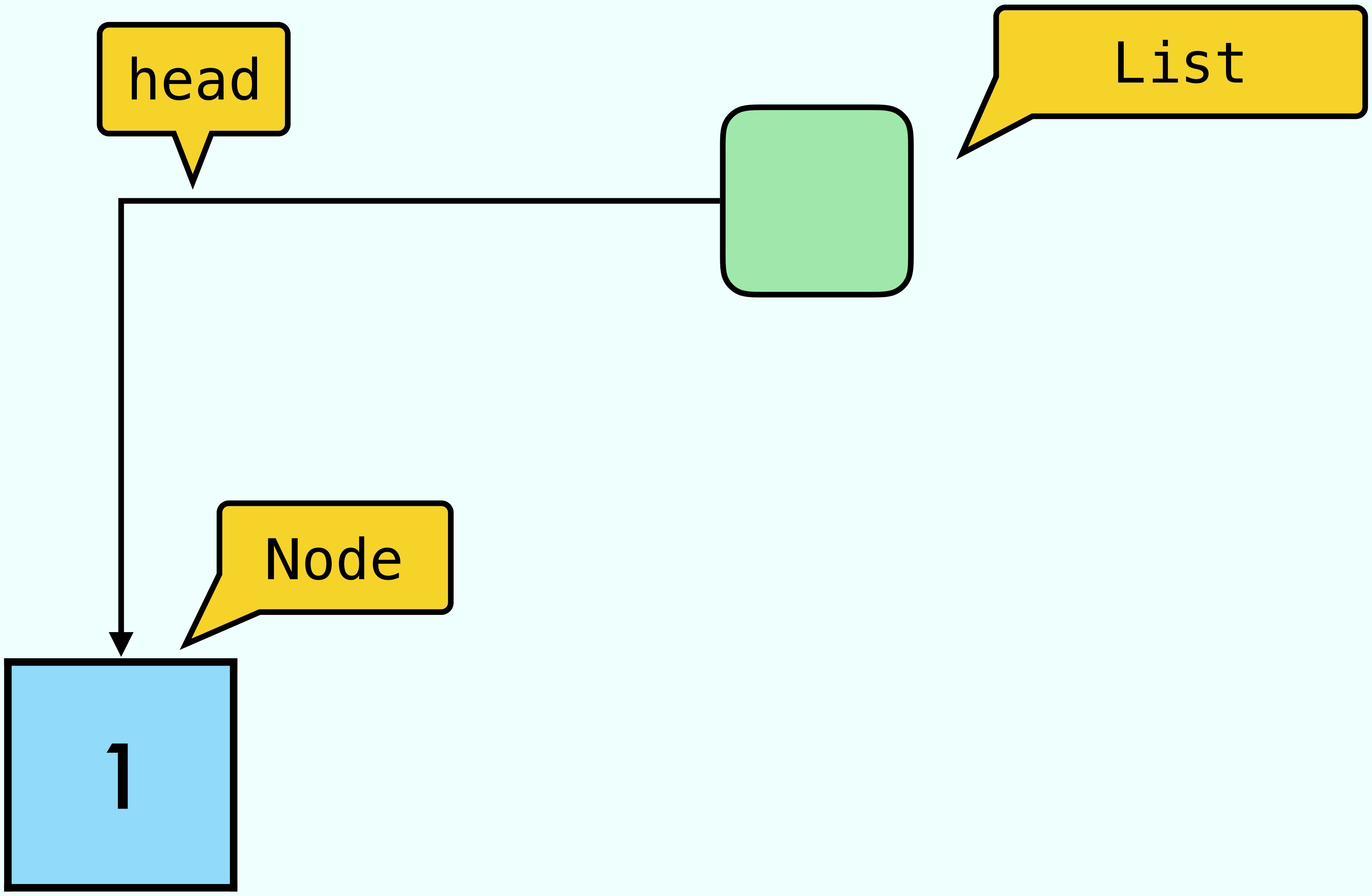
Linked List



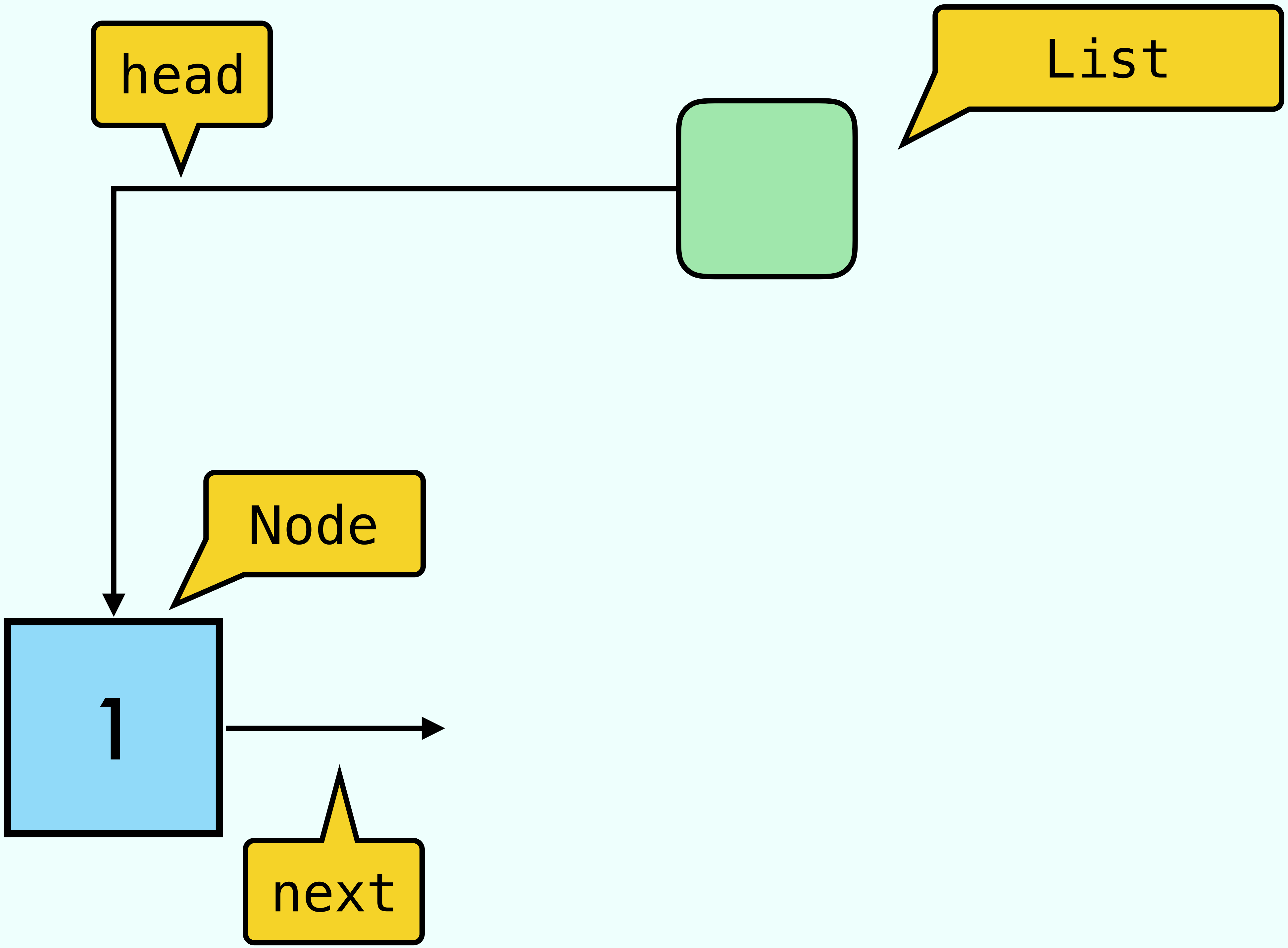
Linked List



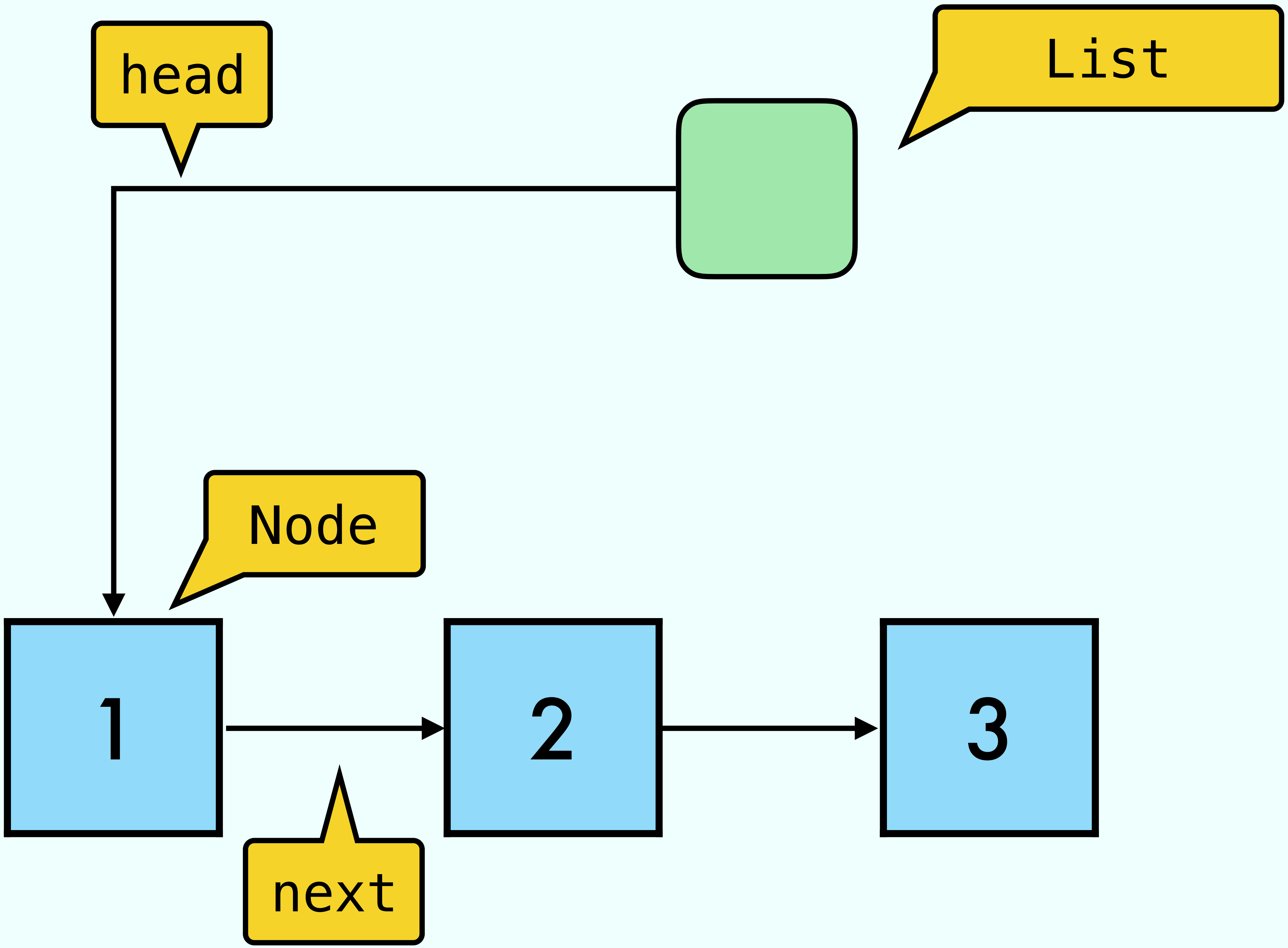
Linked List



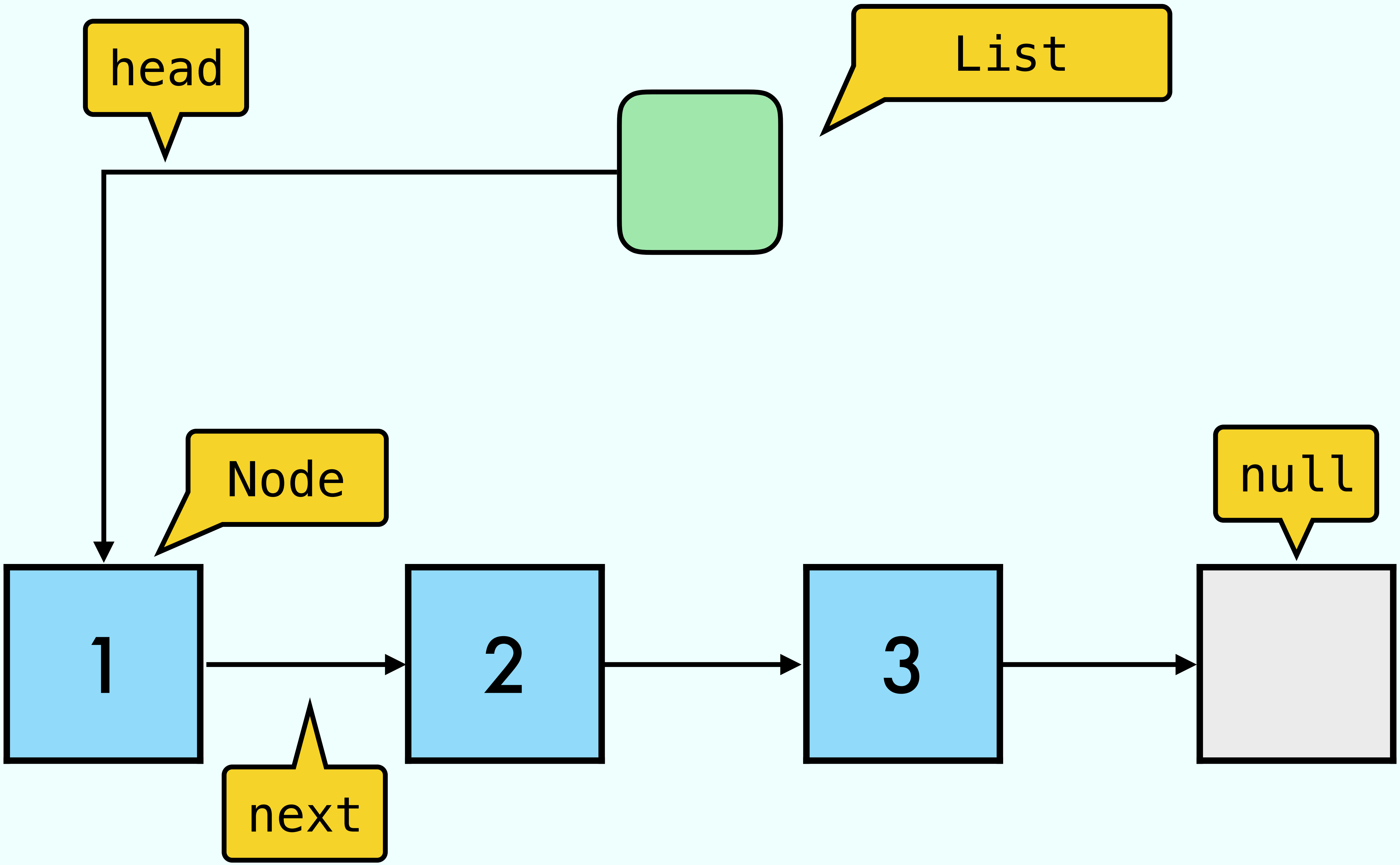
Linked List

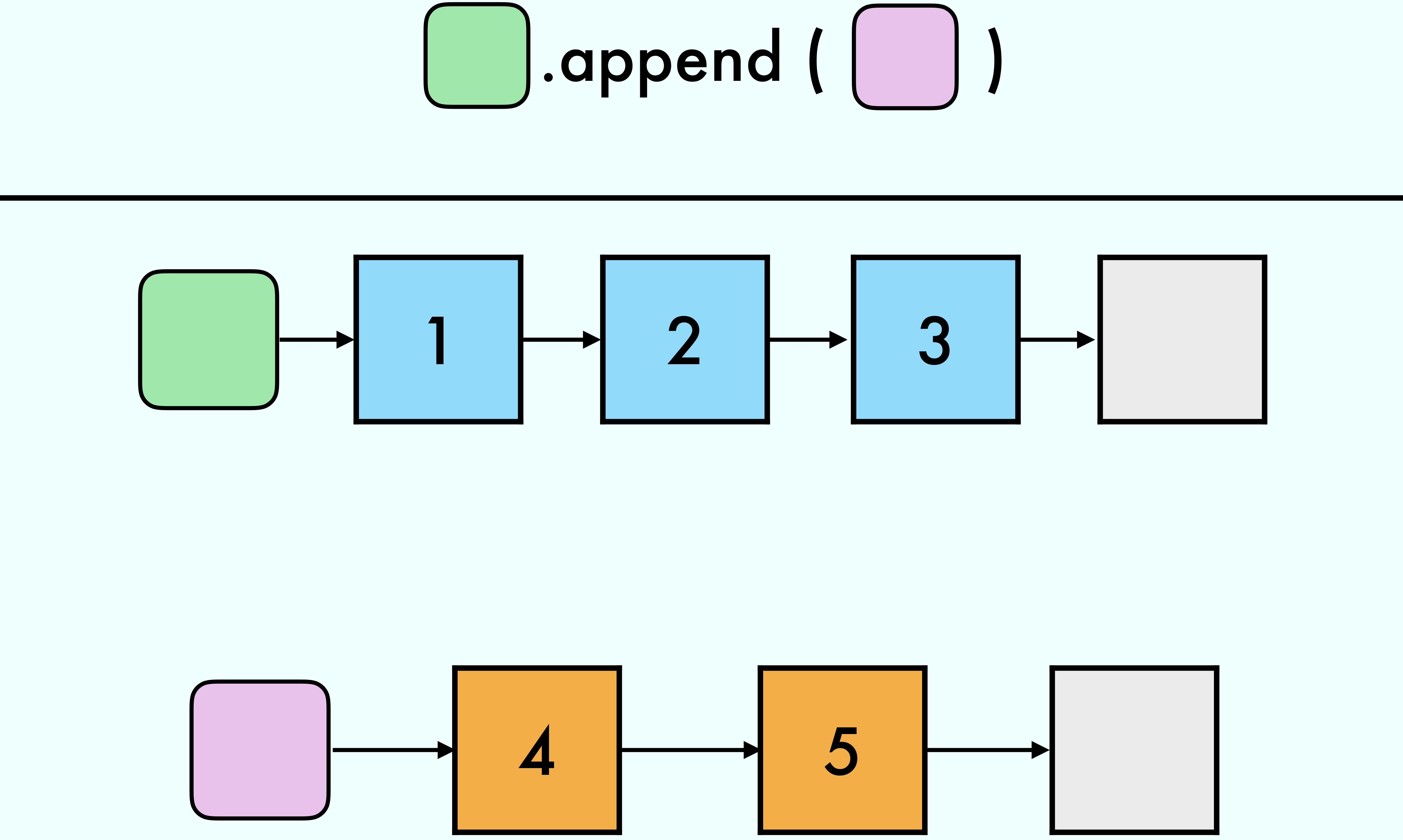


Linked List

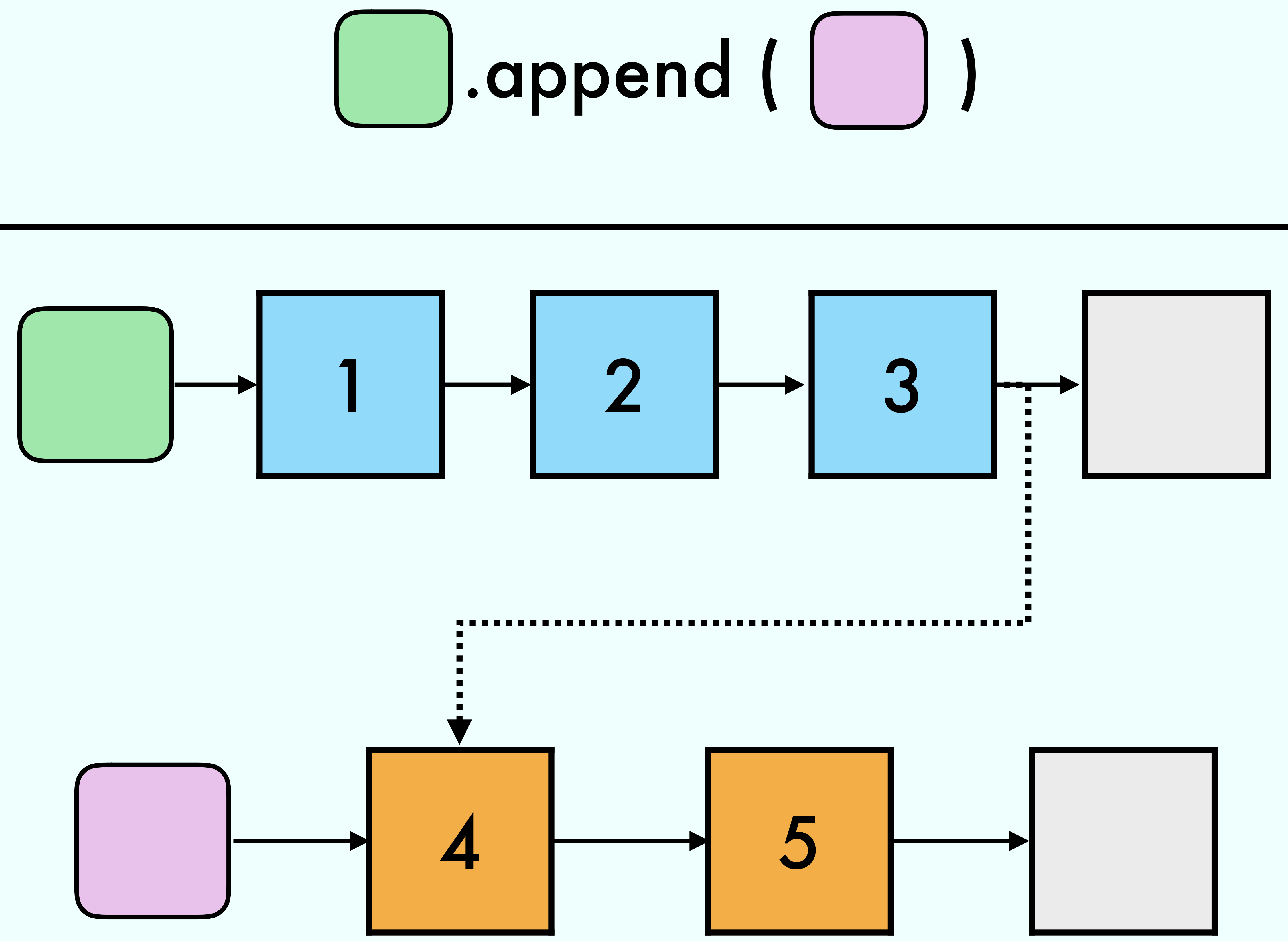


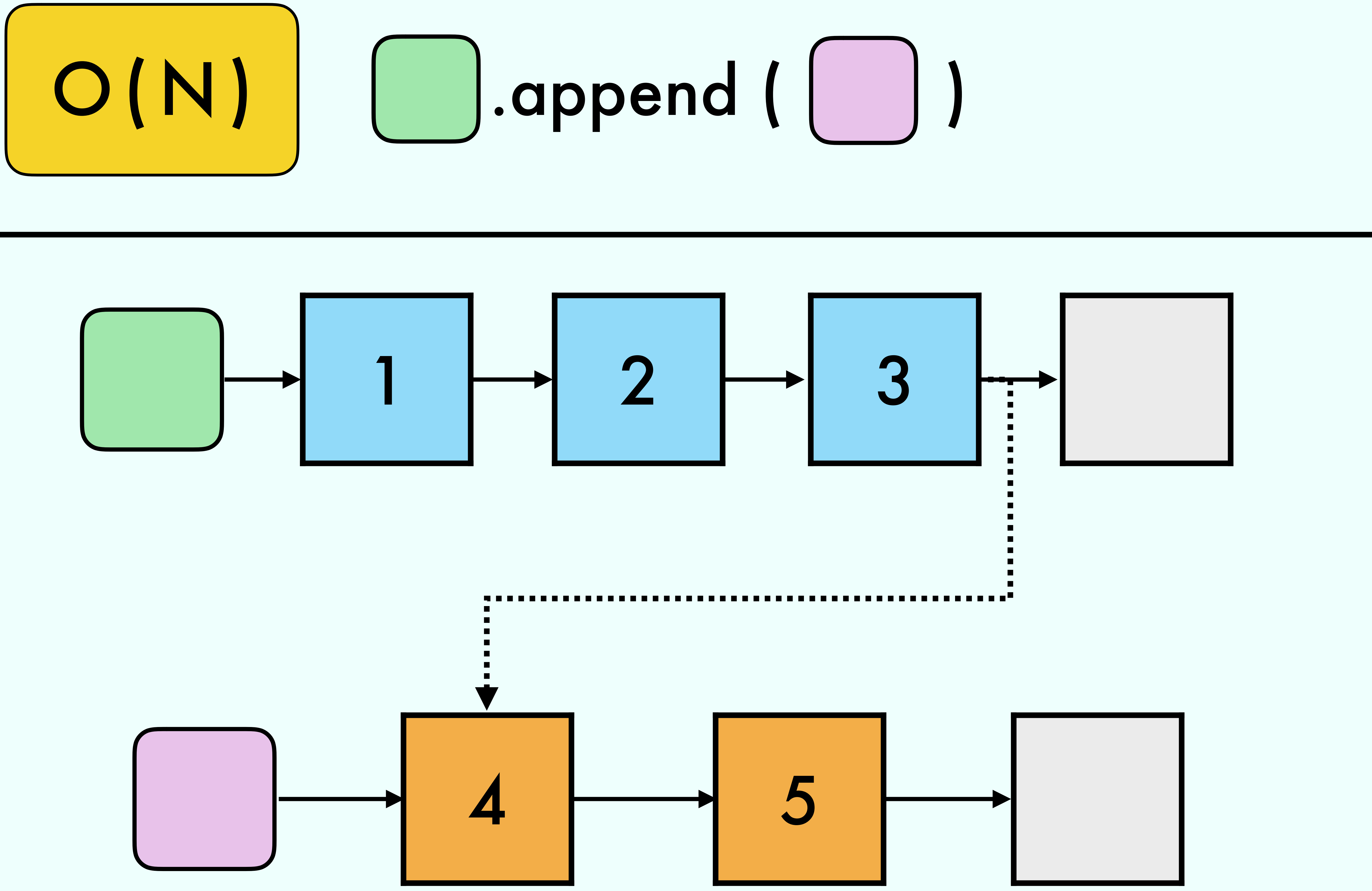
Linked List

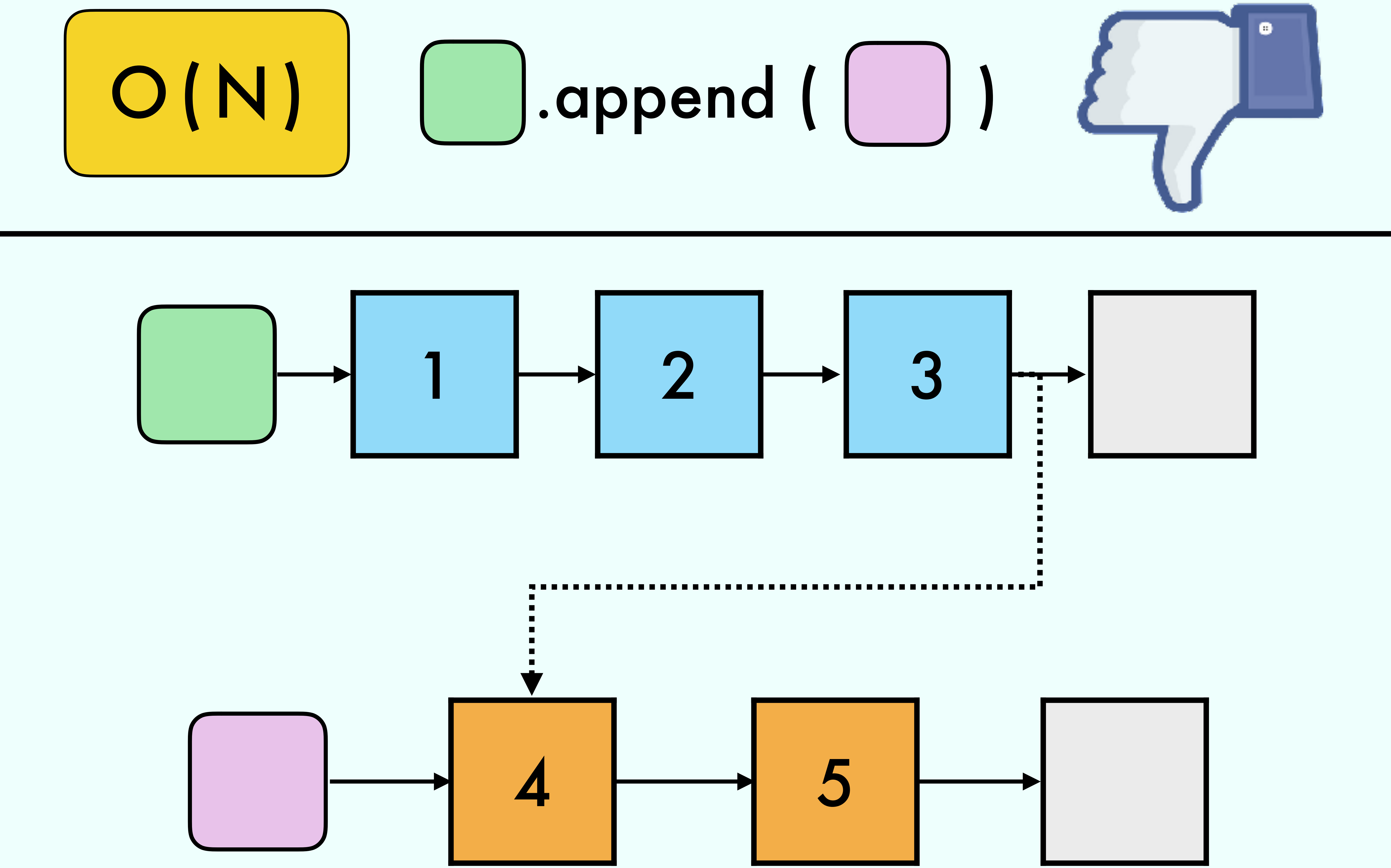




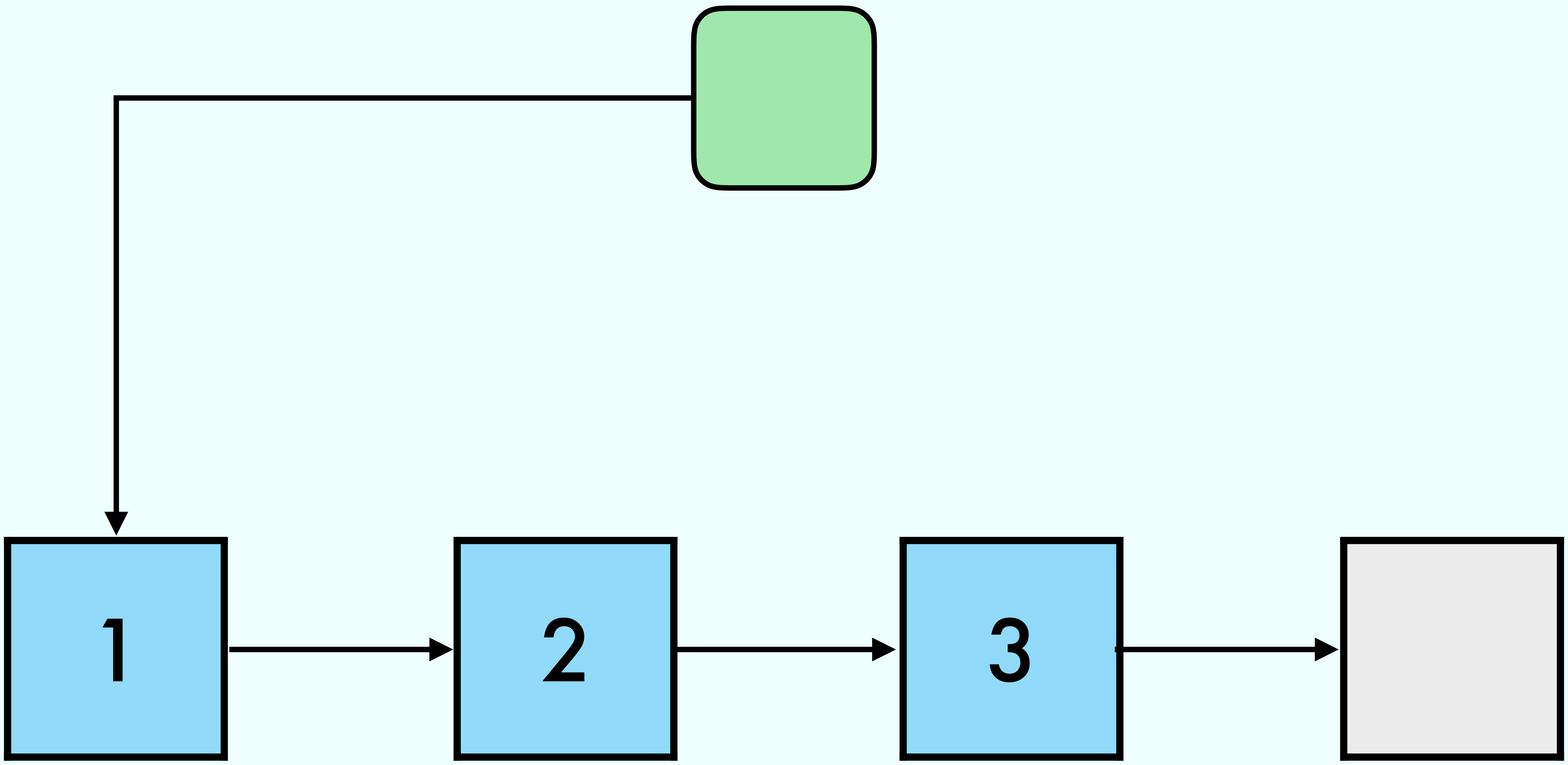
.append ( )
1
2
3
4
5


.append ( )
1
2
3
4
5

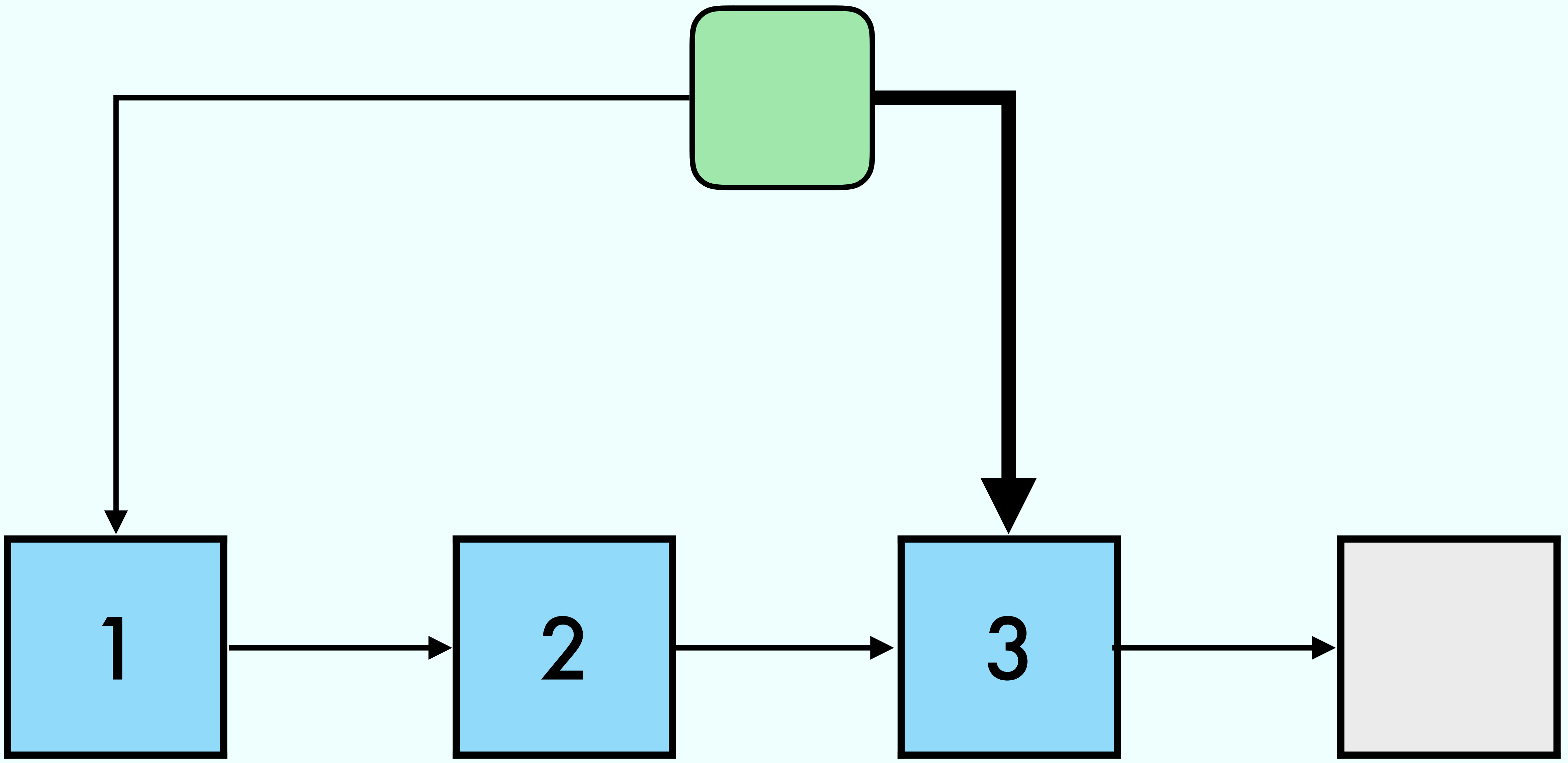

O(N)
.append ( )
1
2
3
4
5


O(N)
.append ( )
1
2
3
4
5

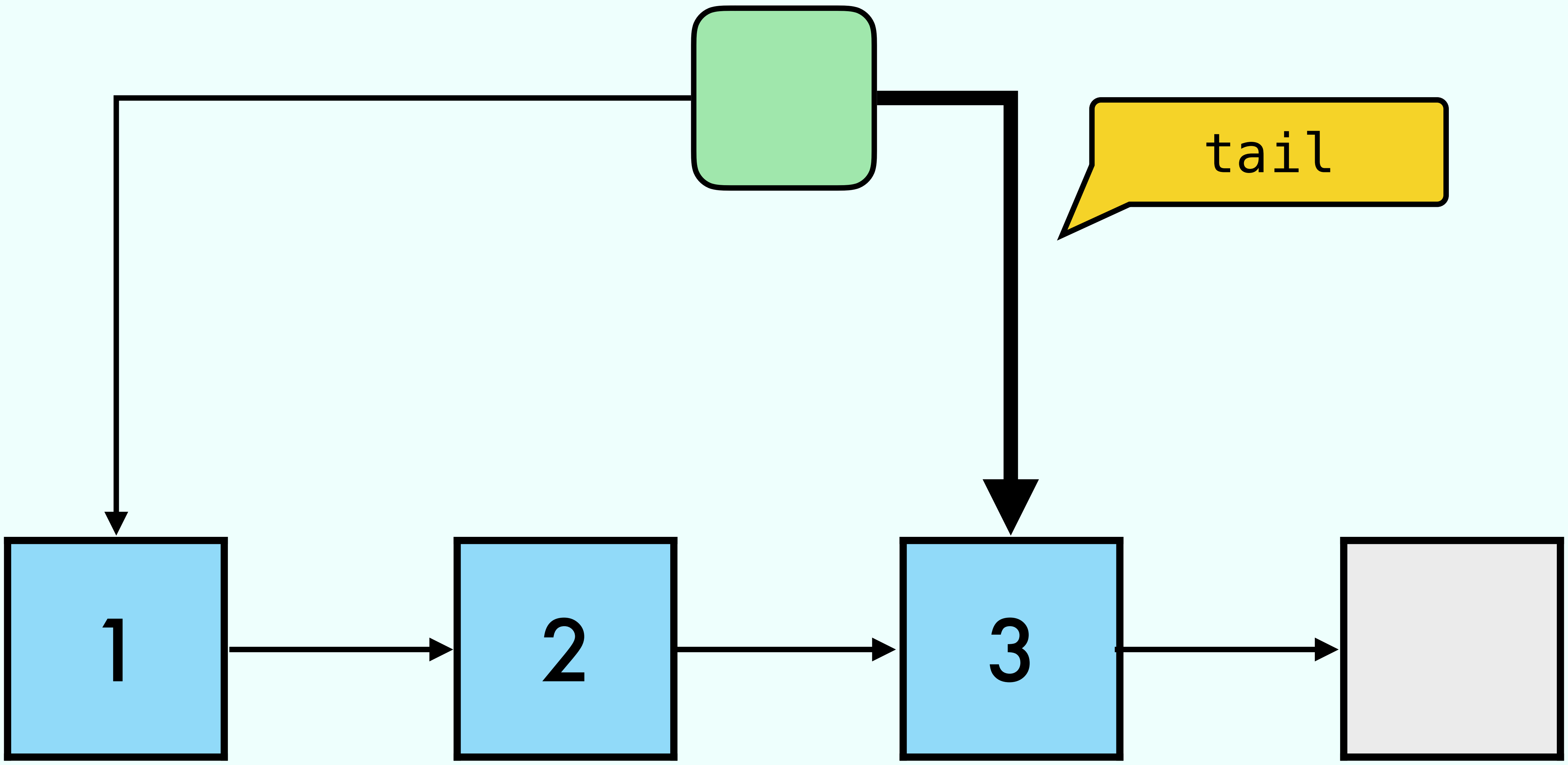
Linked List with pointer to last

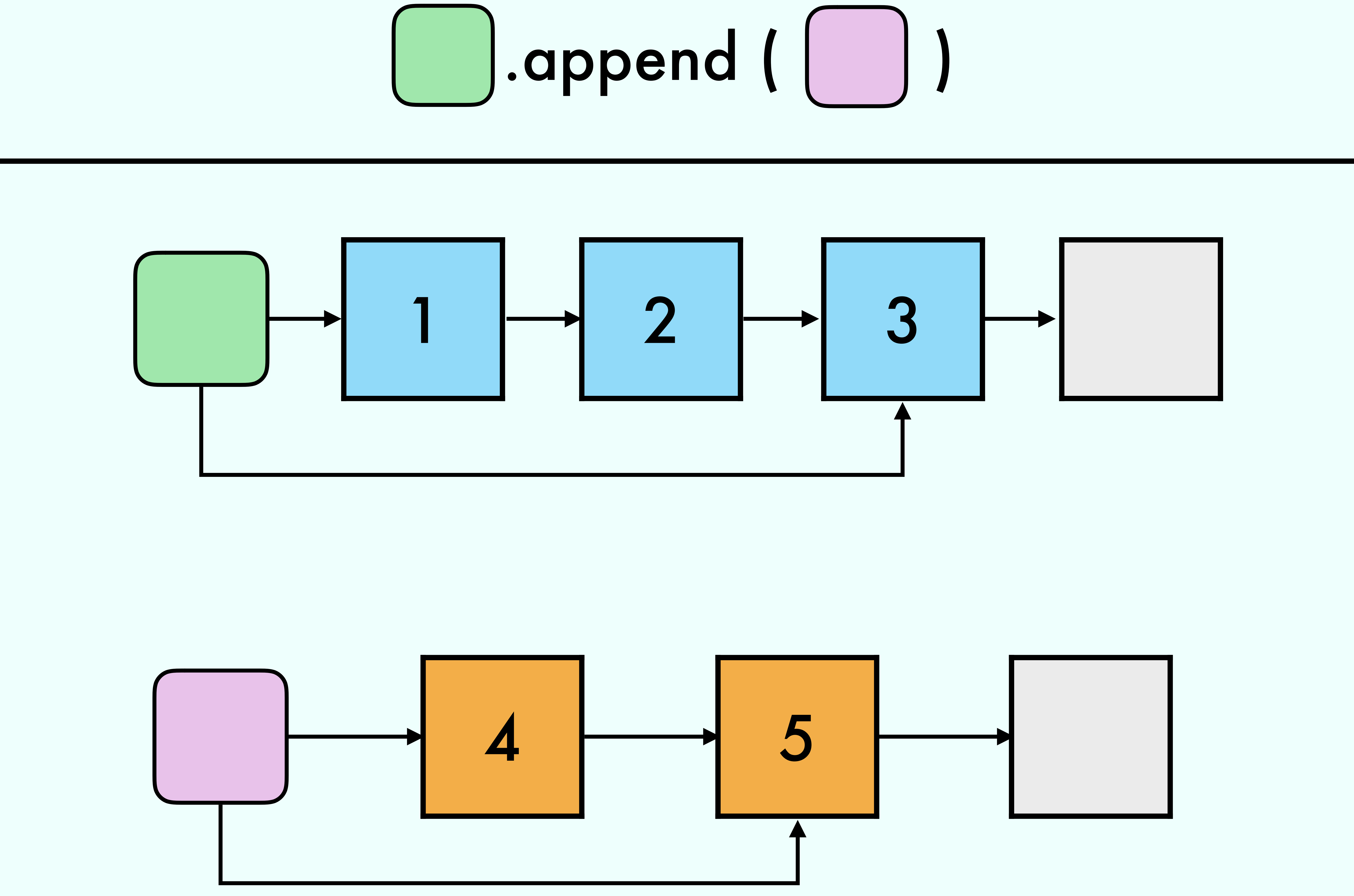


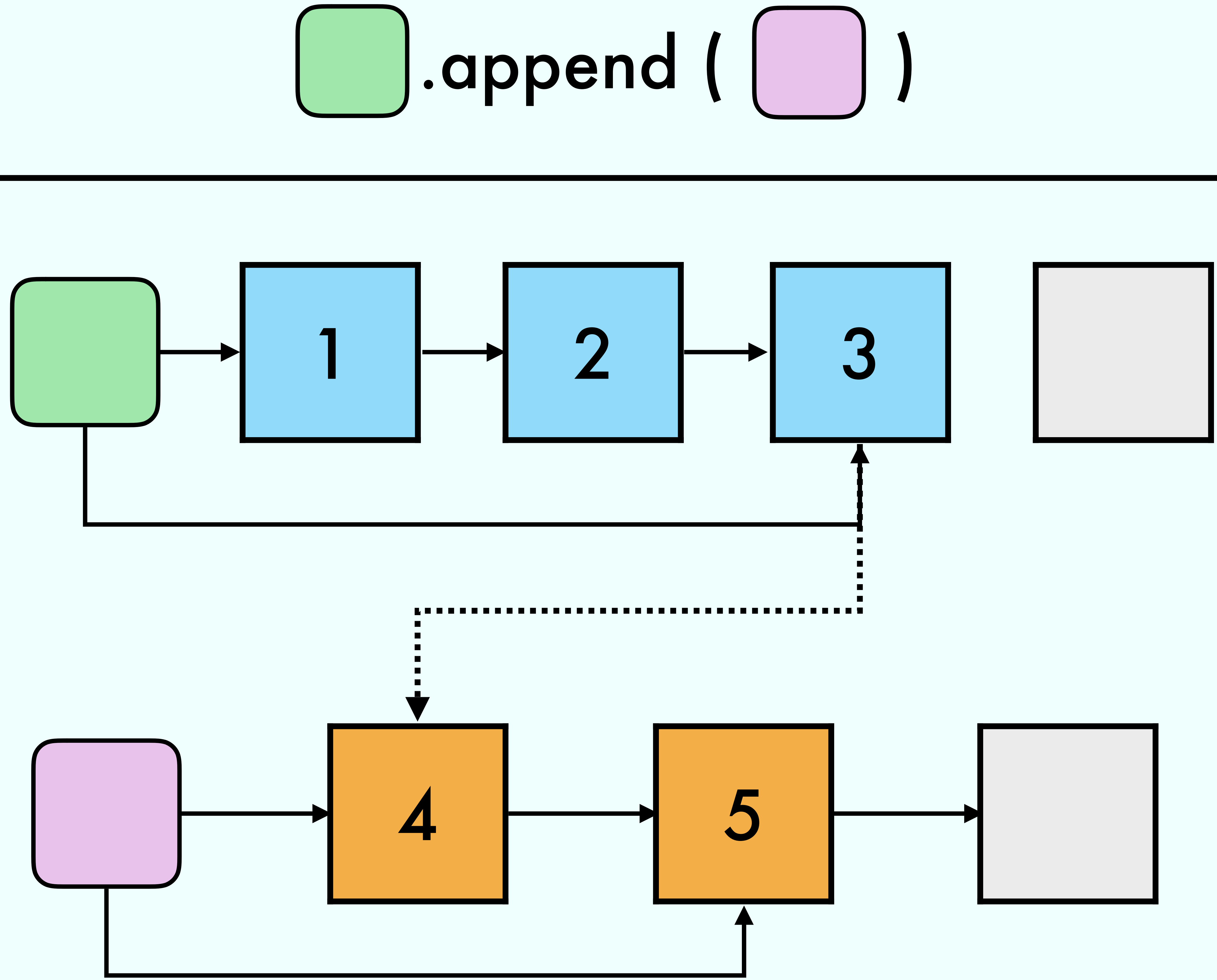
Linked List with pointer to last

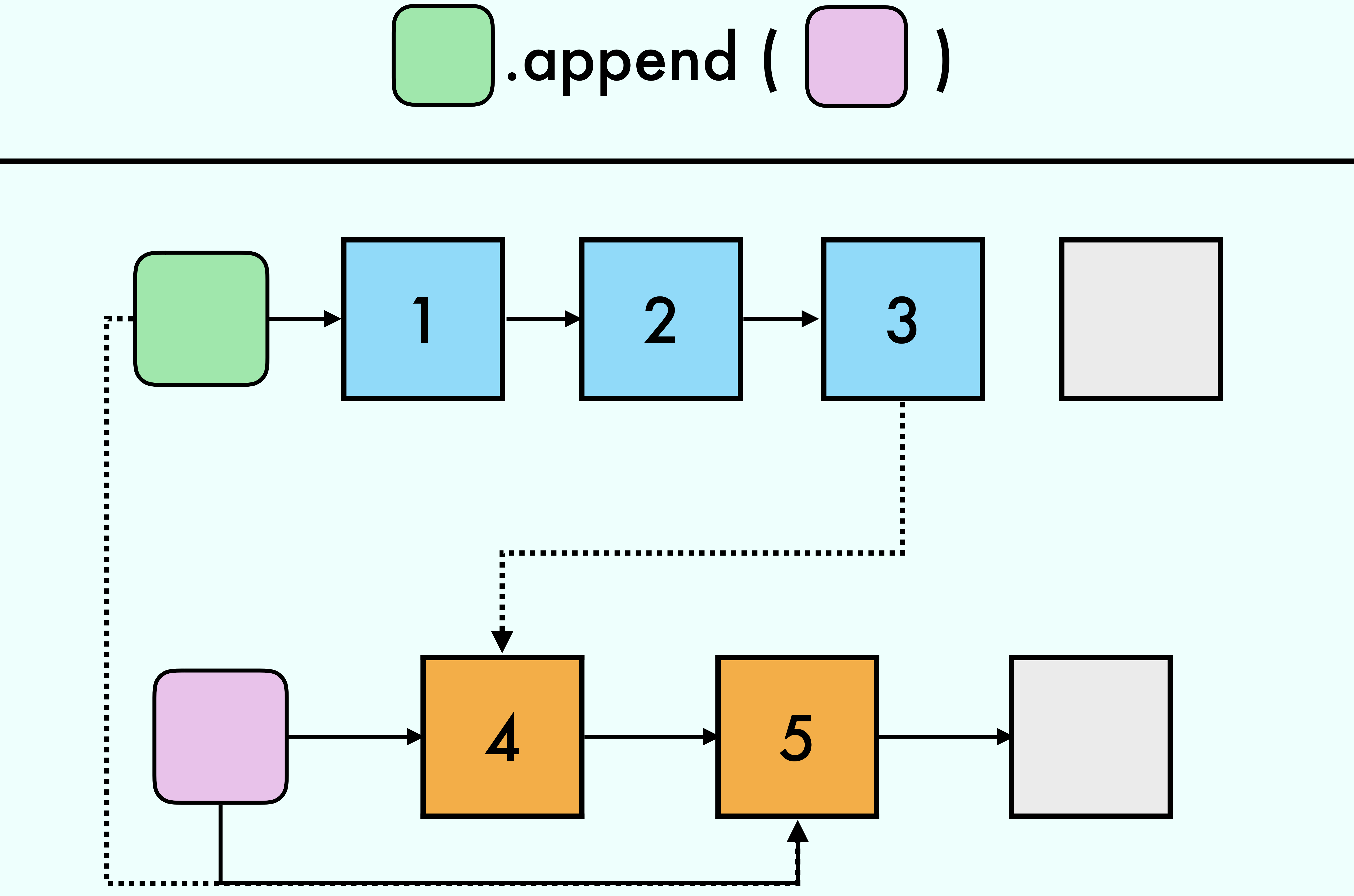


Linked List with pointer to last

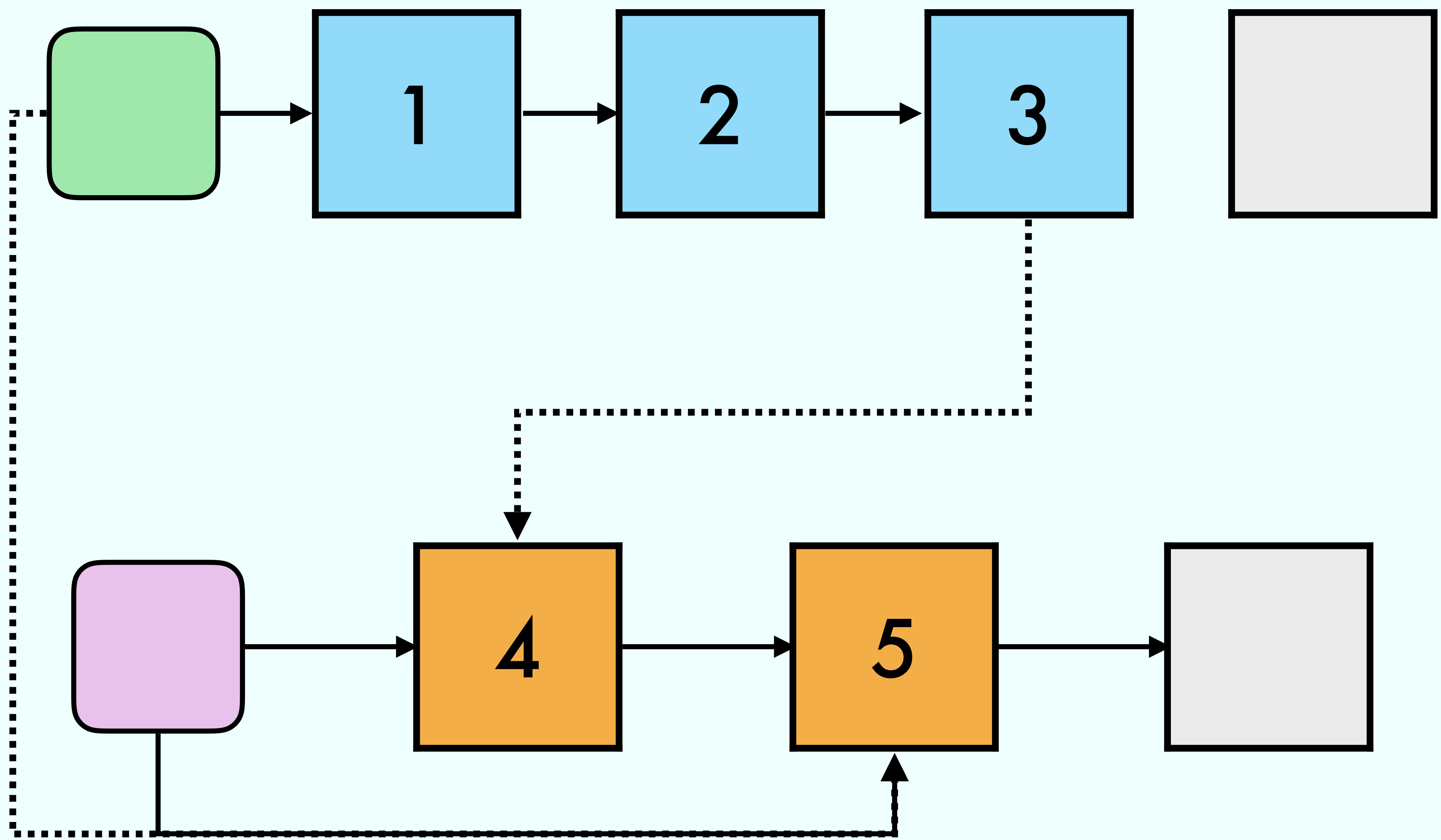



.append ( )
1
2
3
4
5


.append ( )
1
2
3
4
5


.append ( )
1
2
3
4
5

O(1) .append ( )



Exercise 3.25a

Define a data structure with

Exercise 3.25a

Define a data structure with

push(x)		

Exercise 3.25a

Define a data structure with

push(x)	O(1)	

Exercise 3.25a

Define a data structure with

push(x)	O(1)	☑

Exercise 3.25a

Define a data structure with

push(x)	O(1)	
pop()		

Exercise 3.25a

Define a data structure with

push(x)	O(1)	☑
pop()	O(1)	

Exercise 3.25a

Define a data structure with

push(x)	O(1)	☑
pop()	O(1)	☑

Exercise 3.25a

Define a data structure with

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

Exercise 3.25a

Define a data structure with

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

Exercise 3.25a

Define a data structure with

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:

| 1 | 2 | 3 |.printRev()   // O(1) memory

Exercise 3.29

Print a singly linked list in reverse in constant space:

1	2	3

.printRev() // O(1) memory

```
3
2
1
```

Exercise 3.29

Print a singly linked list in reverse in constant space:

1	2	3

.printRev()

3

2

1

```
void printRev() {
  list.reverse();
  for (int x : list)
    print(x);
  list.reverse();
}
```

Reverse in Place

Reverse in Place

Initialize

```
here = list.head
prev = null
```

Reverse in Place

Initialize

```
here = list.head
prev = null
```

First node processed

Reverse in Place

Initialize

```
here = list.head     First node processed
prev = null          Previous node
```

Reverse in Place

Initialize

```
here = list.head        First node processed
prev = null             Previous node
```

Reverse Loop

Shifting

Reverse

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

Reverse in Place

Initialize

```
here = list.head
prev = null
```

First node processed

Previous node

Reverse Loop

Shifting

Reverse

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

Save next node

Reverse in Place

Initialize

```
here = list.head
prev = null
```

First node processed

Previous node

Reverse Loop

Shifting

Reverse

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

Save next node

Reversing

Reverse in Place

Initialize

```
here = list.head          First node processed
prev = null               Previous node
```

Reverse Loop

Shifting

Reverse

```
while (here ≠ null) do
  next = here.next        Save next node
  here.next = prev        Reversing
  prev = here             Save previous node
  here = next
```

Reverse in Place

Initialize

here = list.head — First node processed
prev = null — Previous node

Reverse Loop

Shifting

Reverse

while (here ≠ null) do
  next = here.next — Save next node
  here.next = prev — Reversing
  prev = here — Save previous node
  here = next — Shift to next node

Reverse in Place

Initialize

```
here = list.head            First node processed
prev = null                 Previous node
```

Reverse Loop

Shifting

Reverse

```
while (here ≠ null) do
  next = here.next          Save next node
  here.next = prev          Reversing
  prev = here               Save previous node
  here = next               Shift to next node
```

Conclusion

Reverse in Place

Initialize

```
here = list.head        First node processed
prev = null             Previous node
```

Reverse Loop

Shifting

Reverse

```
while (here ≠ null) do
  next = here.next      Save next node
  here.next = prev      Reversing
  prev = here           Save previous node
  here = next           Shift to next node
```

Conclusion

```
list.head = prev
```

Reverse in Place

Initialize

```
here = list.head        First node processed
prev = null             Previous node
```

Reverse Loop

Shifting

Reverse

```
while (here ≠ null) do
  next = here.next      Save next node
  here.next = prev      Reversing
  prev = here           Save previous node
  here = next           Shift to next node
```

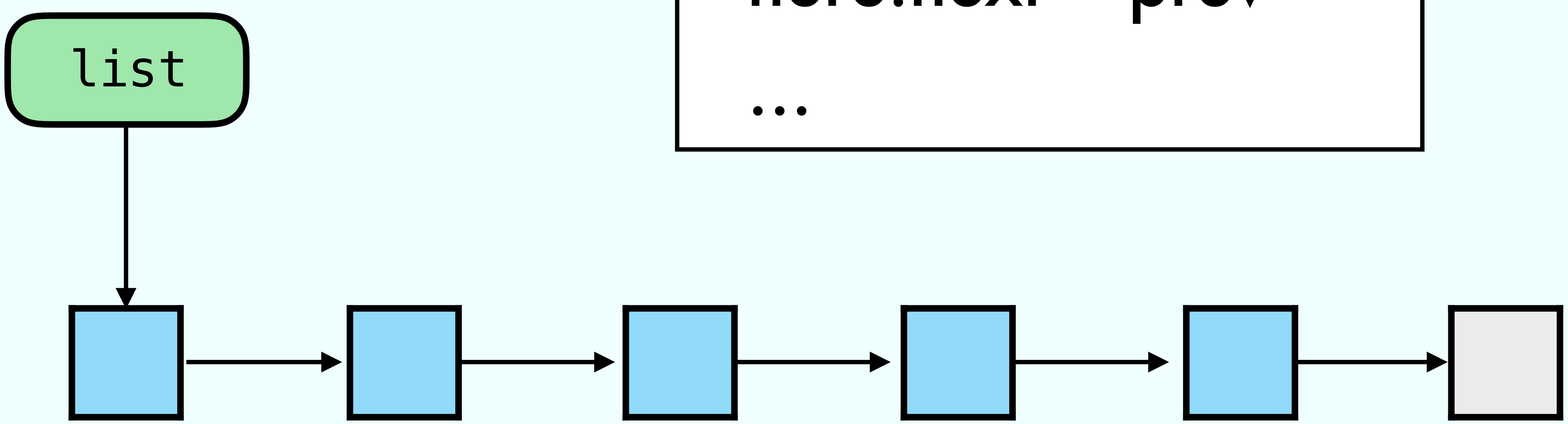
Conclusion

```
list.head = prev        last node becomes the head
```

Initiatlization and First Iteration

```
here = list.head
prev = null
```

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

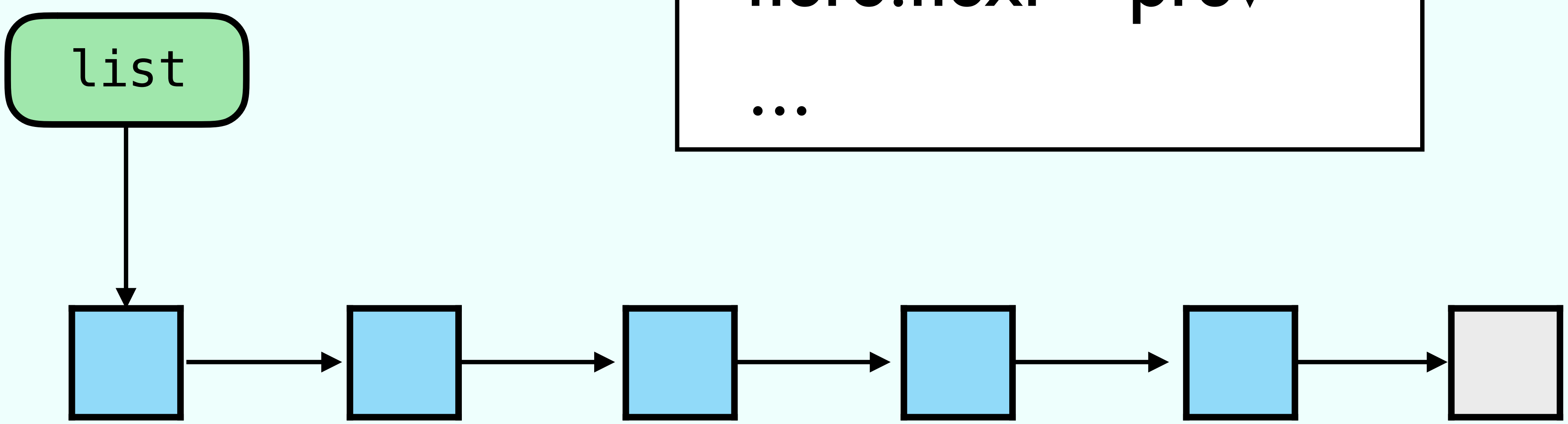


Initiatlization and First Iteration

Initialization

```
here = list.head
prev = null
```

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

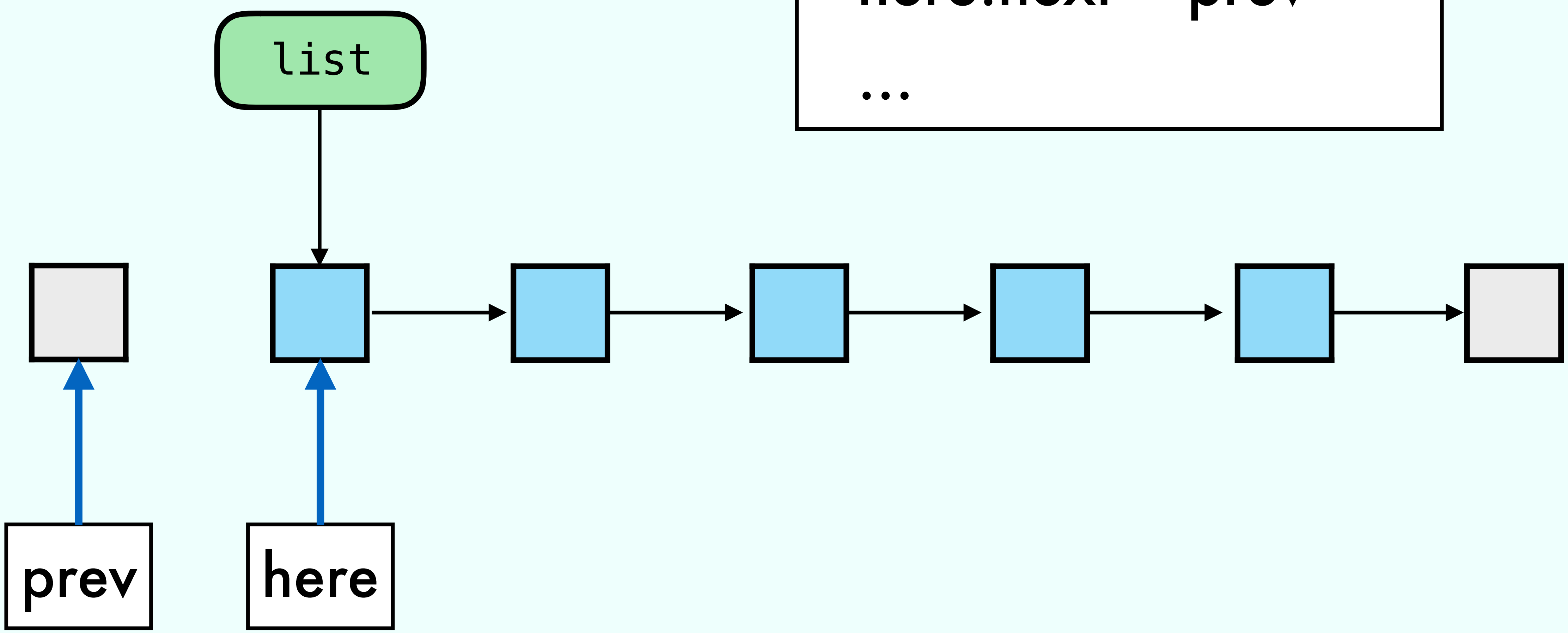


Initiatlization and First Iteration

Initialization

```
here = list.head
prev = null
```

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

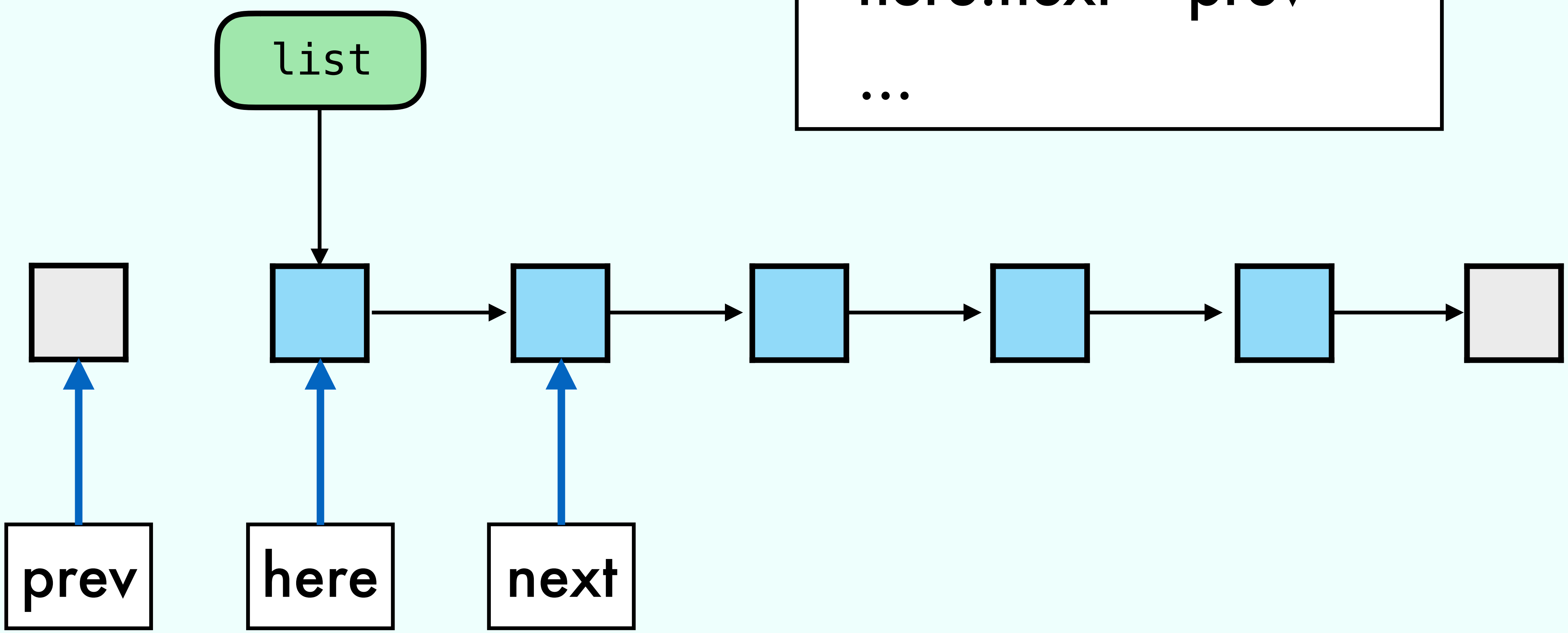


Initiatlization and First Iteration

Initialization

```
here = list.head
prev = null
```

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



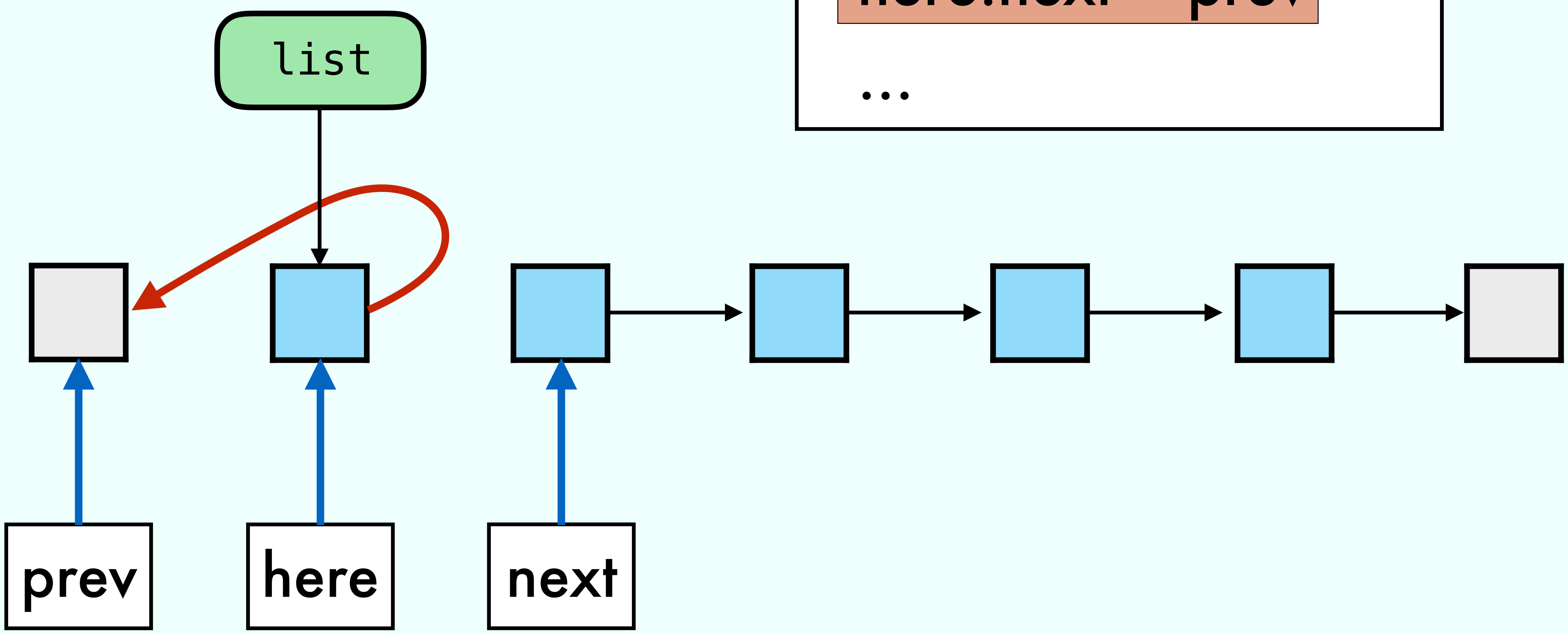
Initiatlization and First Iteration

Initialization

Reverse

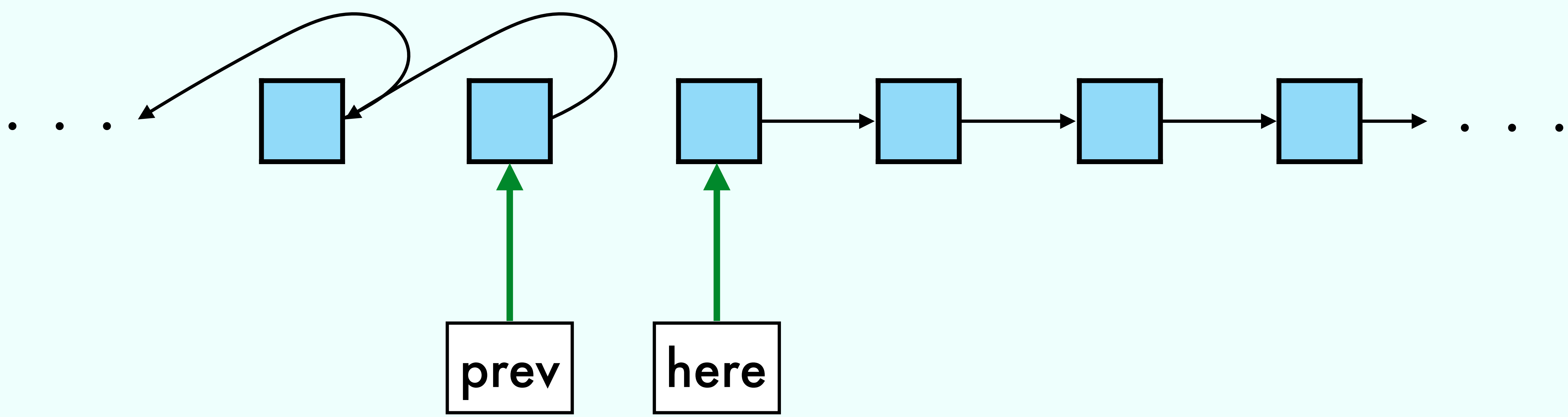
```
here = list.head
prev = null
```

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



Reverse Loop

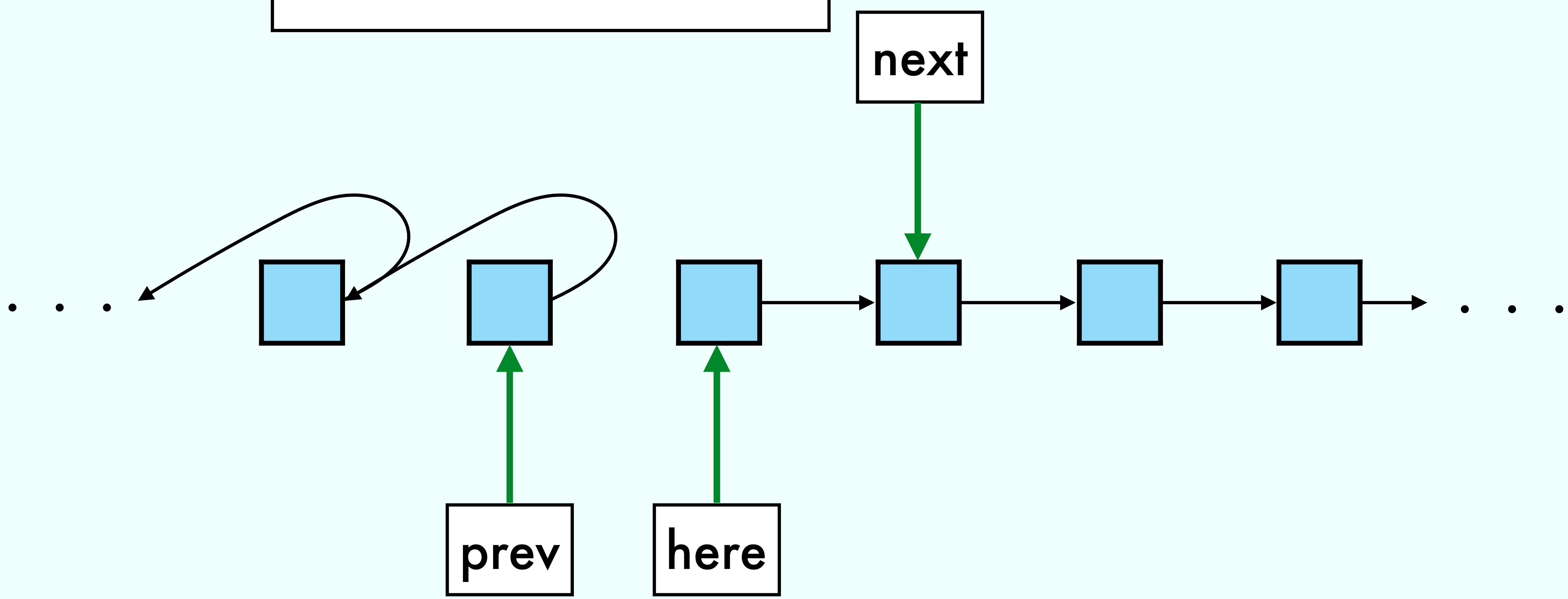
```
while (here ≠ null) do
  next = here.next
  here.next = prev
  prev = here
  here = next
```



Reverse Loop

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

Shifting

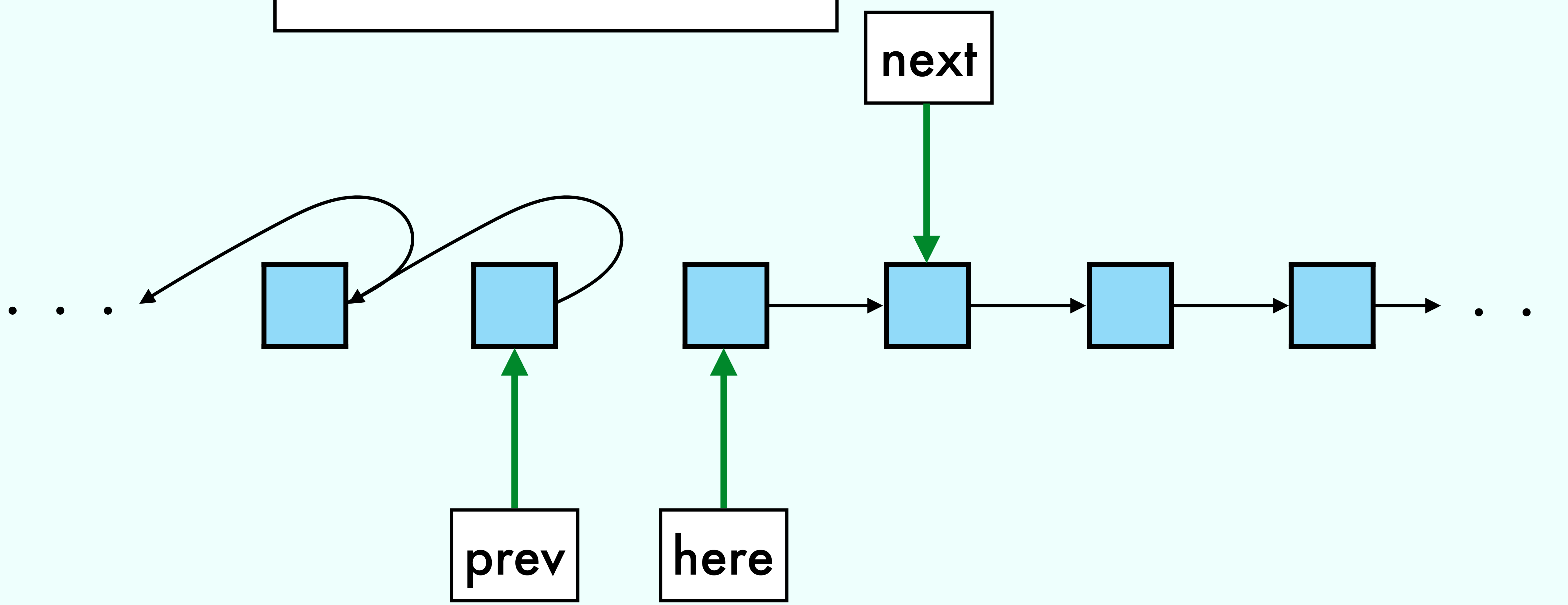


Reverse Loop

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

Shifting

Reverse

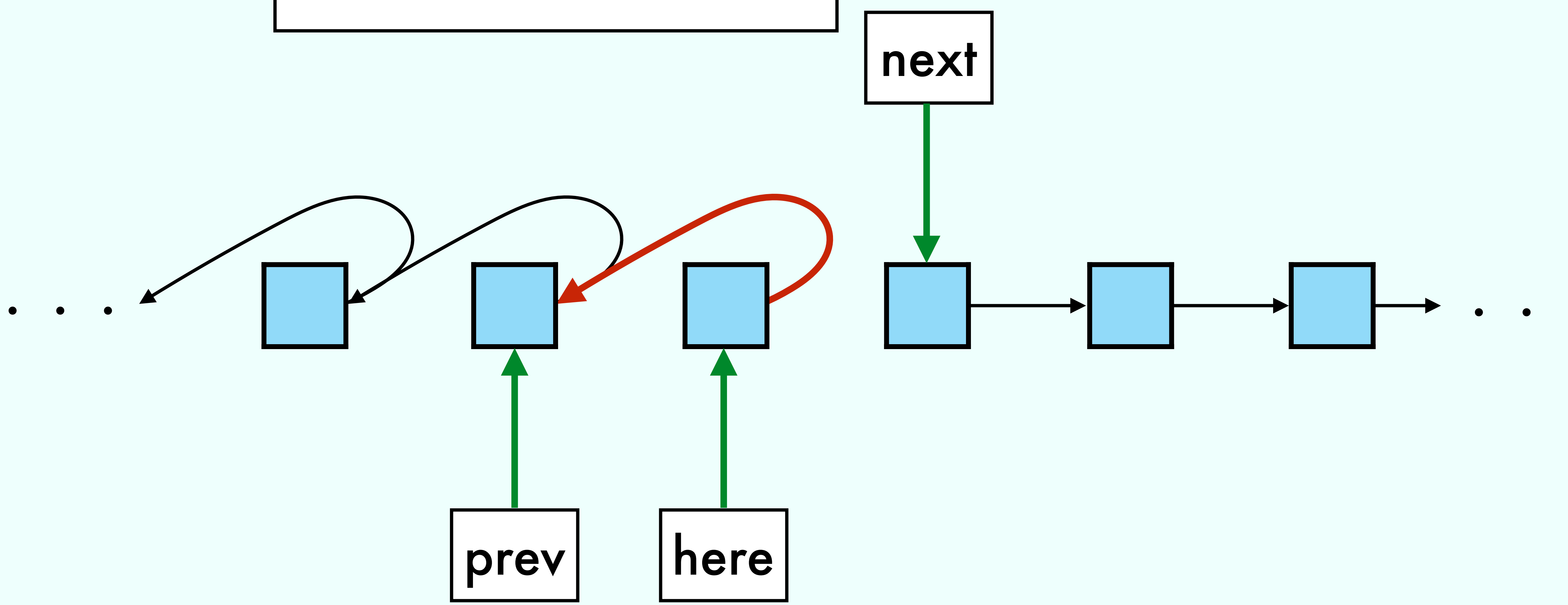


Reverse Loop

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

Shifting

Reverse

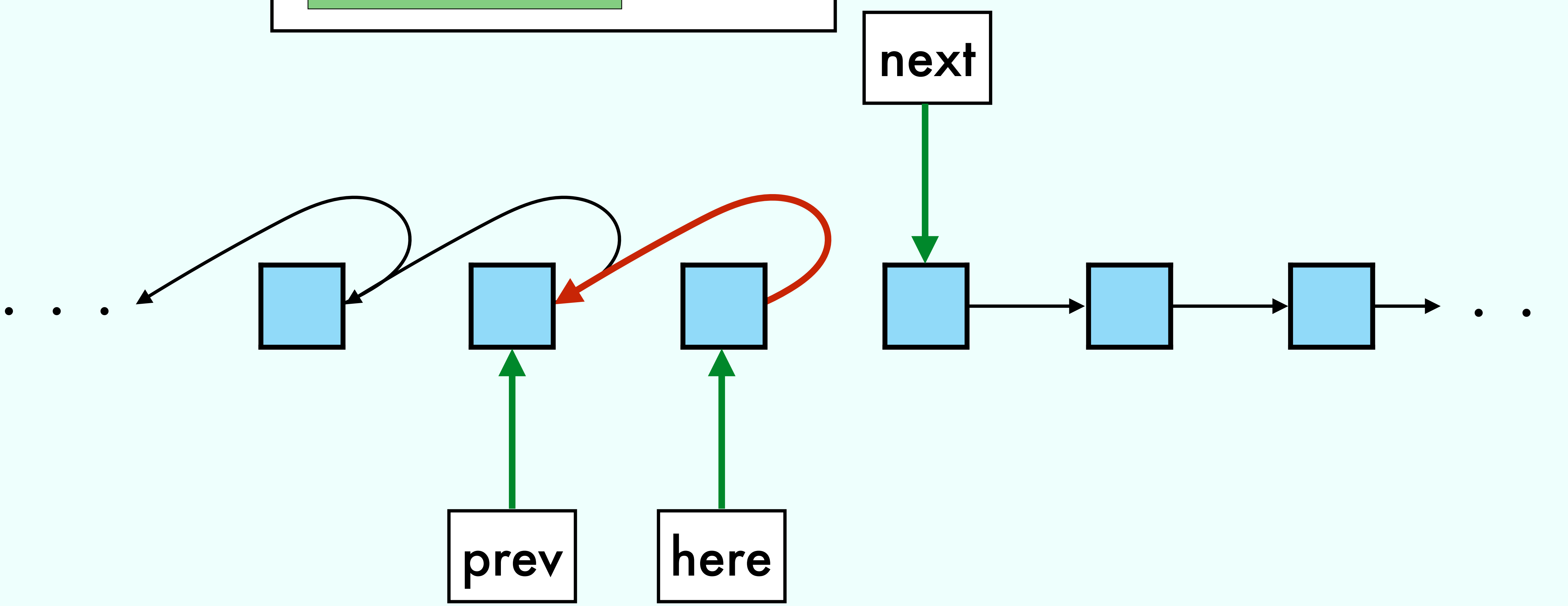


Reverse Loop

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

Shifting

Reverse

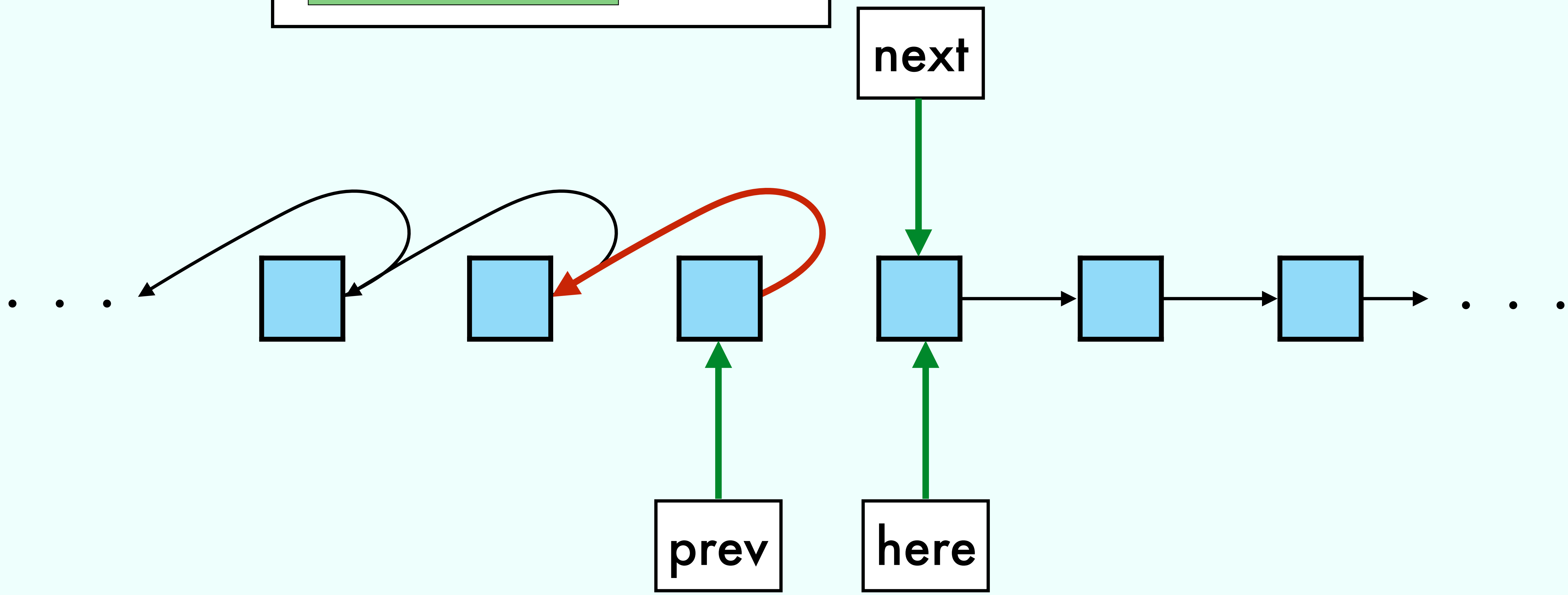


Reverse Loop

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

Shifting

Reverse

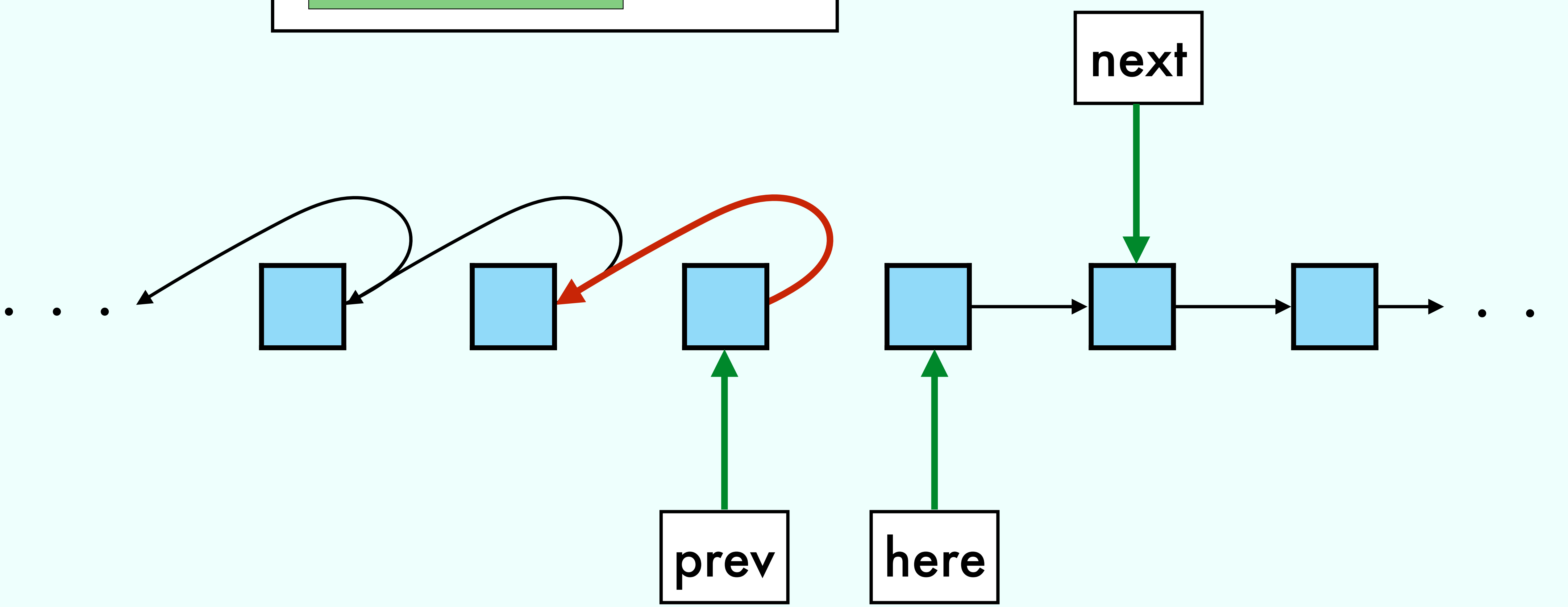


Reverse Loop

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

Shifting

Reverse

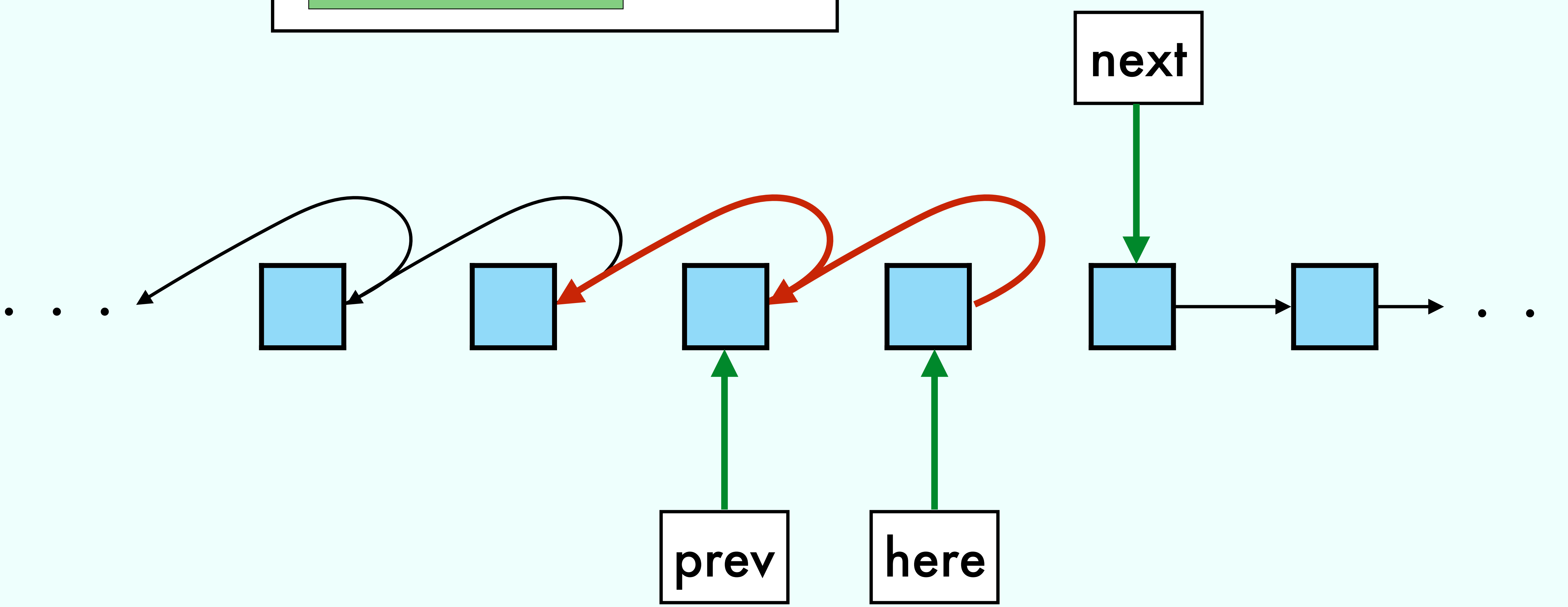


Reverse Loop

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

Shifting

Reverse

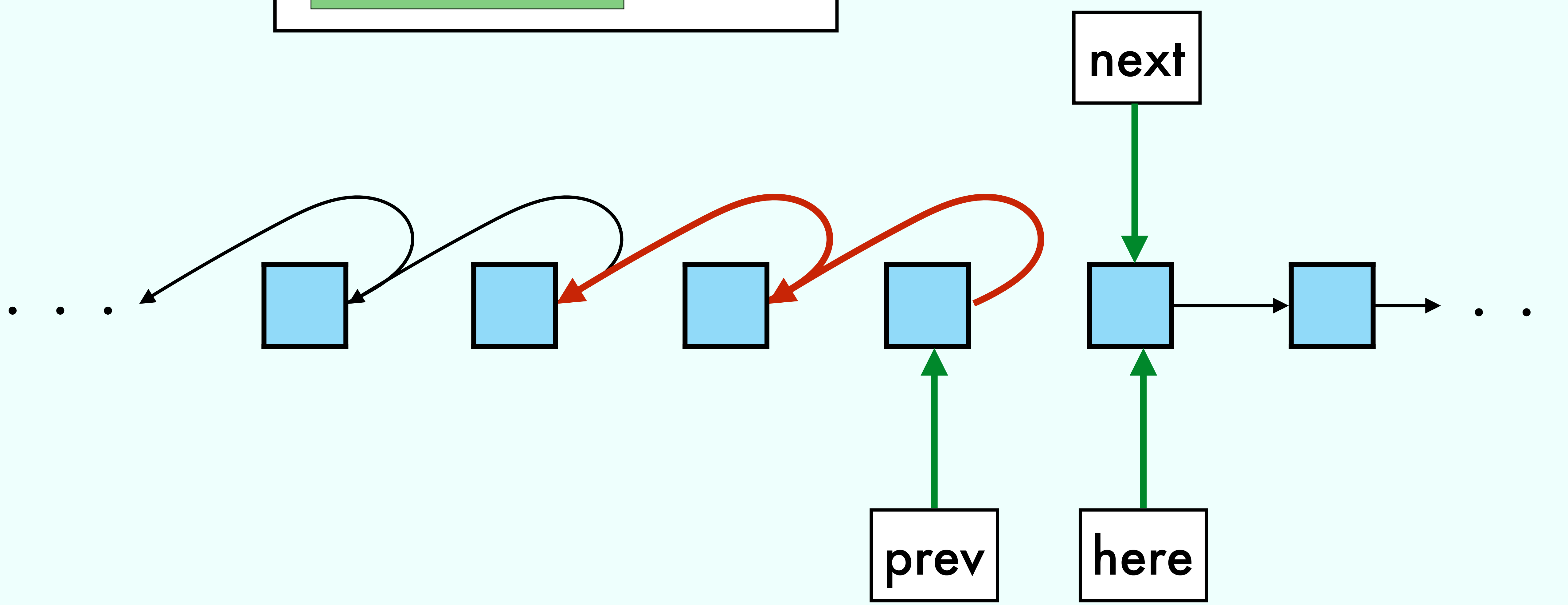


Reverse Loop

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

Shifting

Reverse

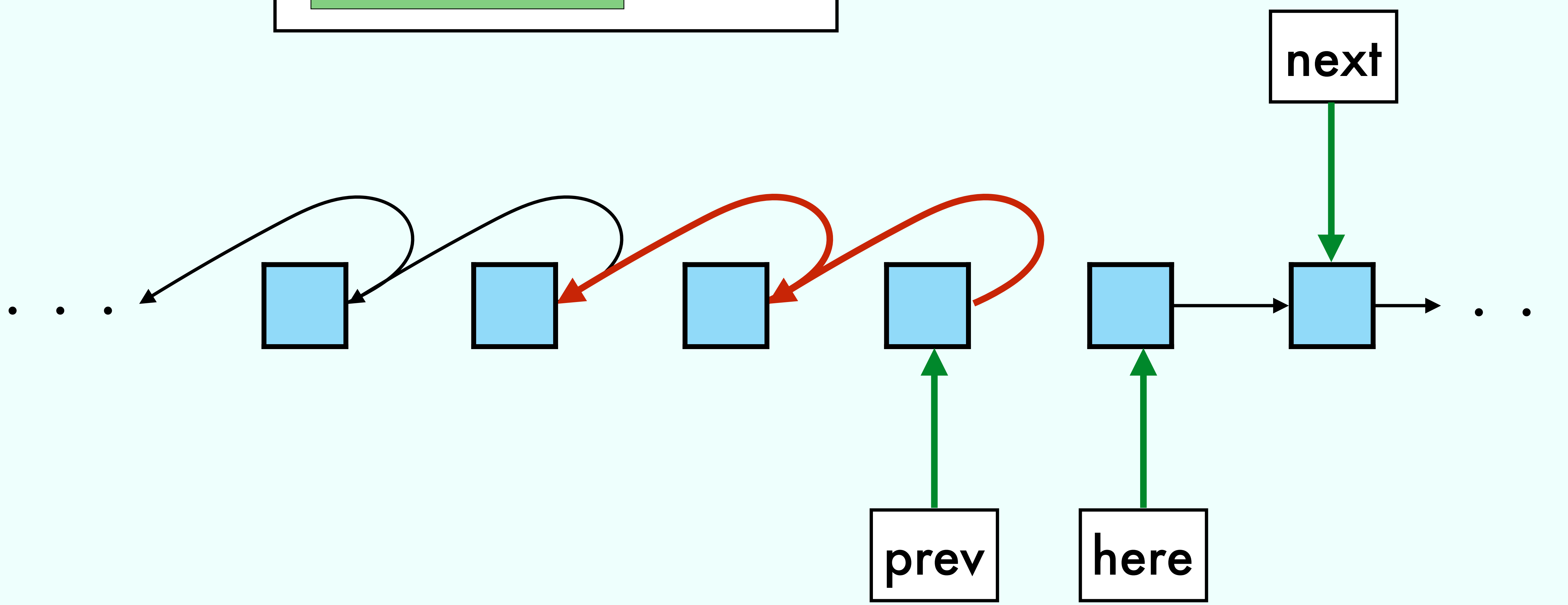


Reverse Loop

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

Shifting

Reverse

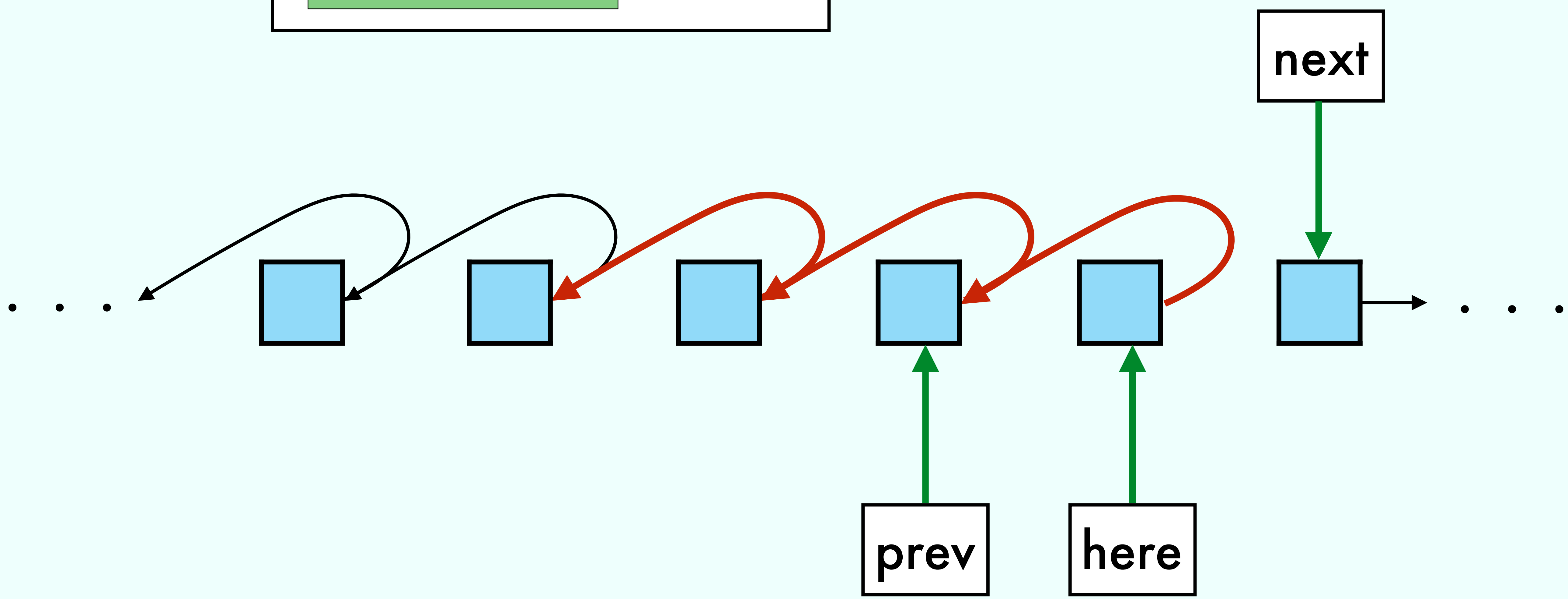


Reverse Loop

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

Shifting

Reverse

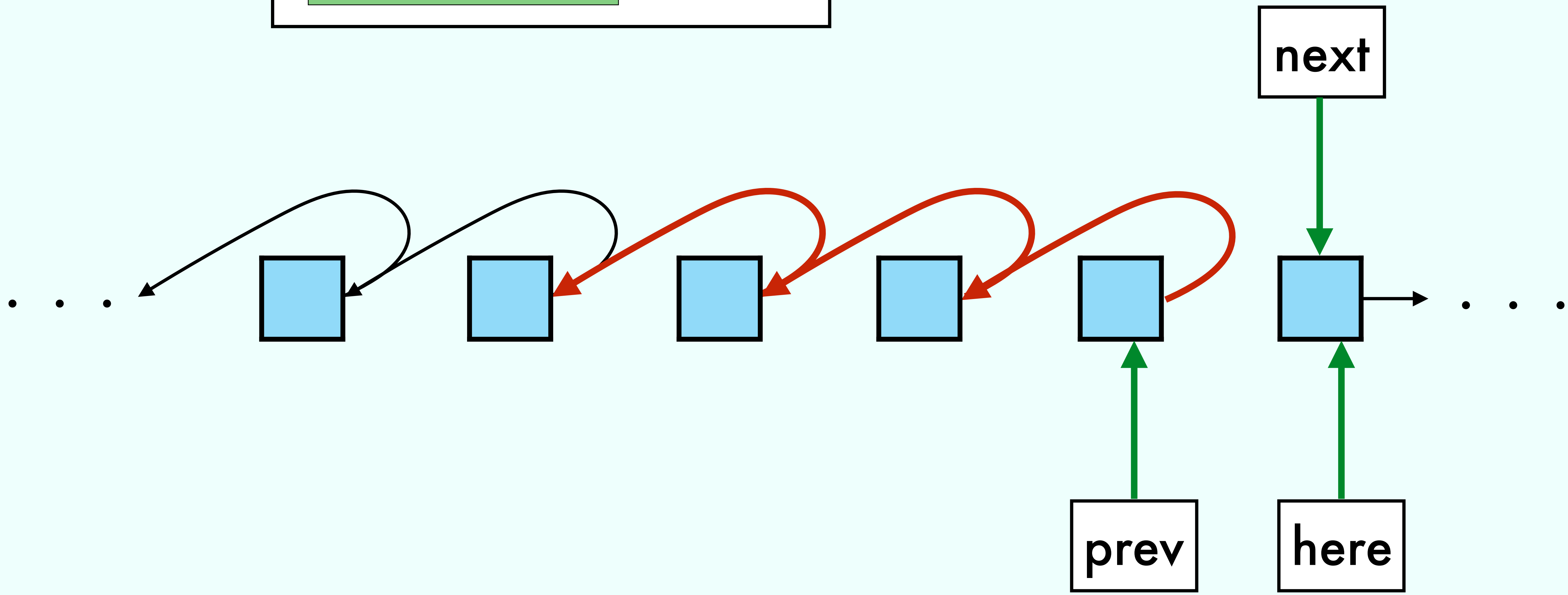


Reverse Loop

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

Shifting

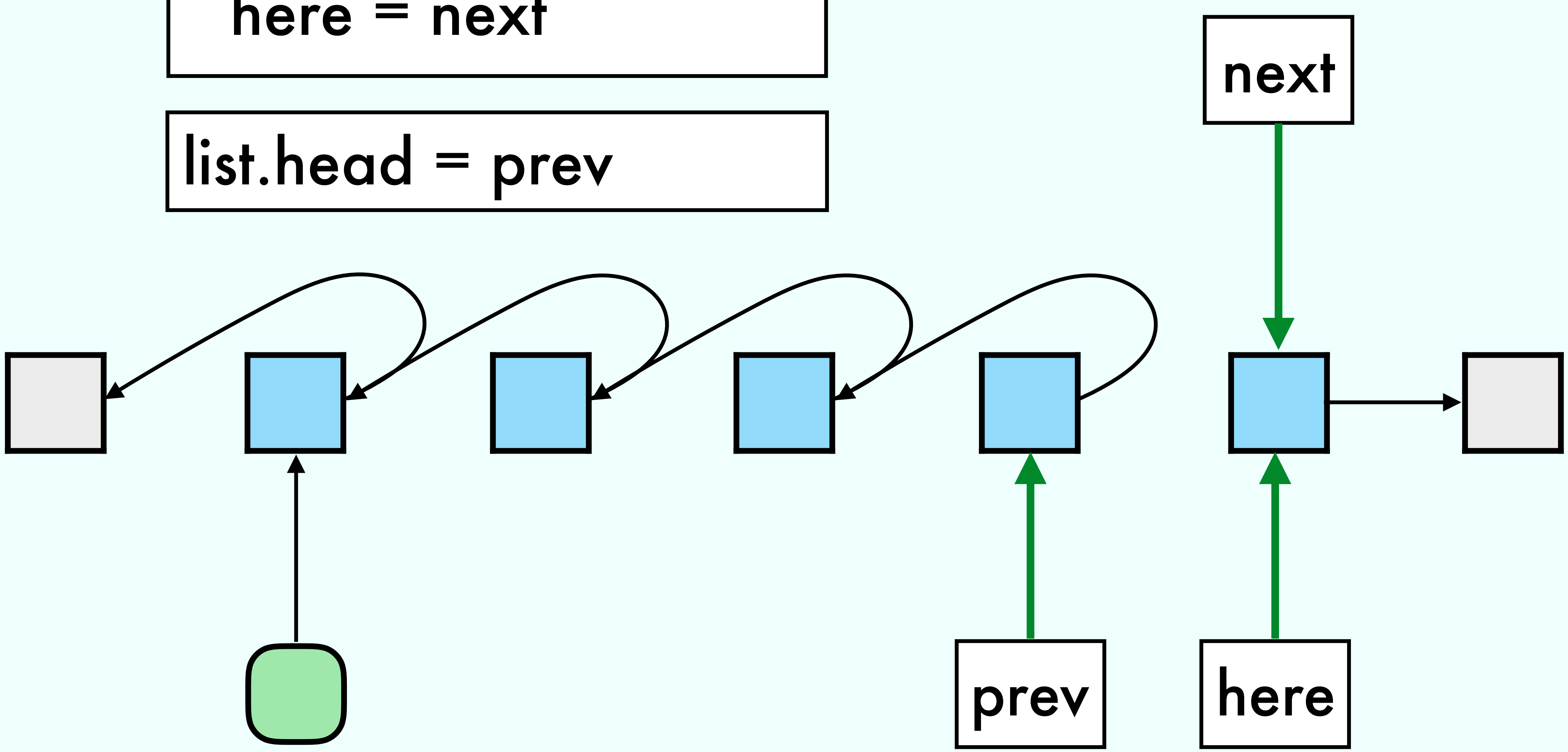
Reverse



Last Iteration and Conclusion

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

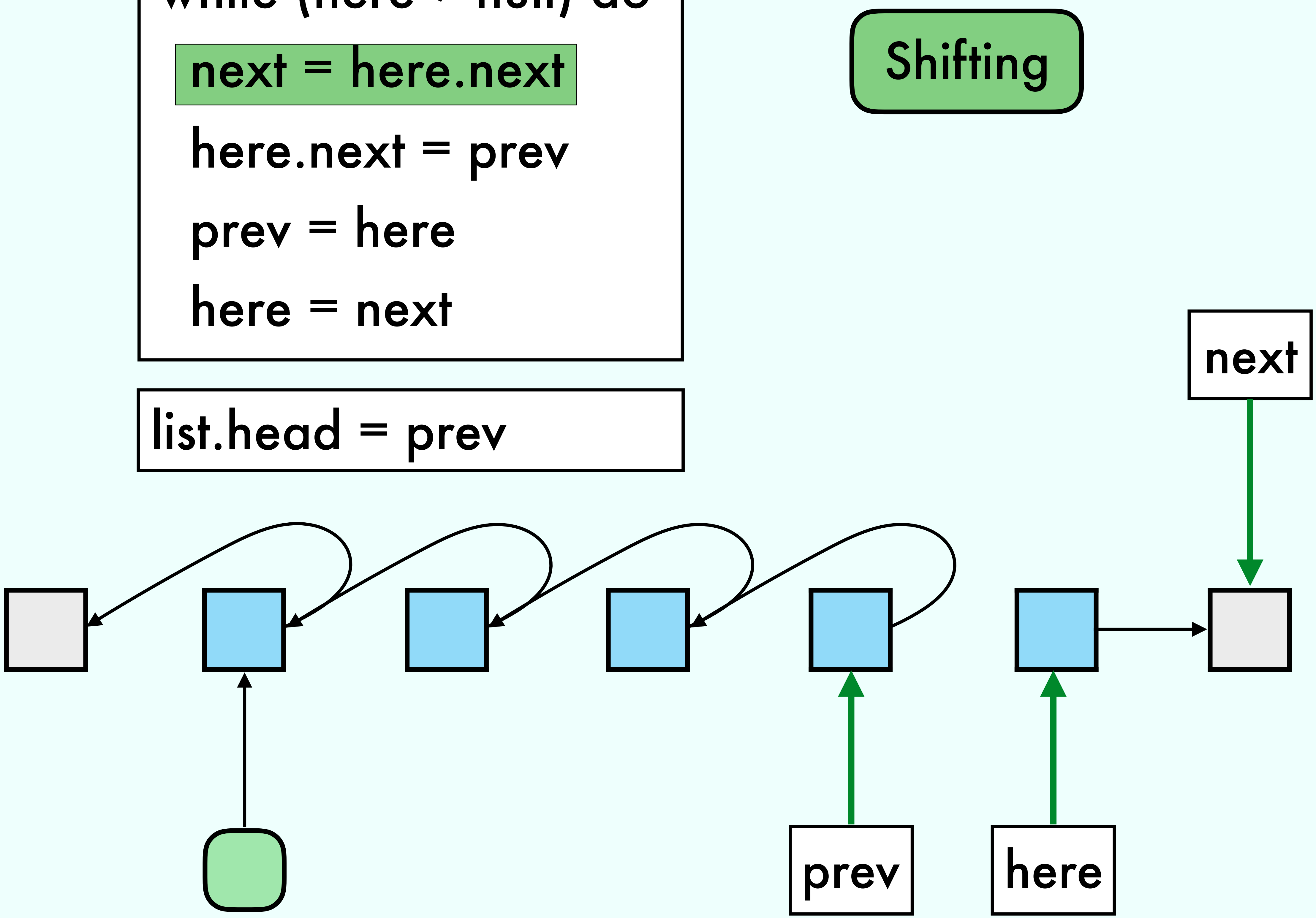
```
list.head = prev
```



Last Iteration and Conclusion

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

```
list.head = prev
```



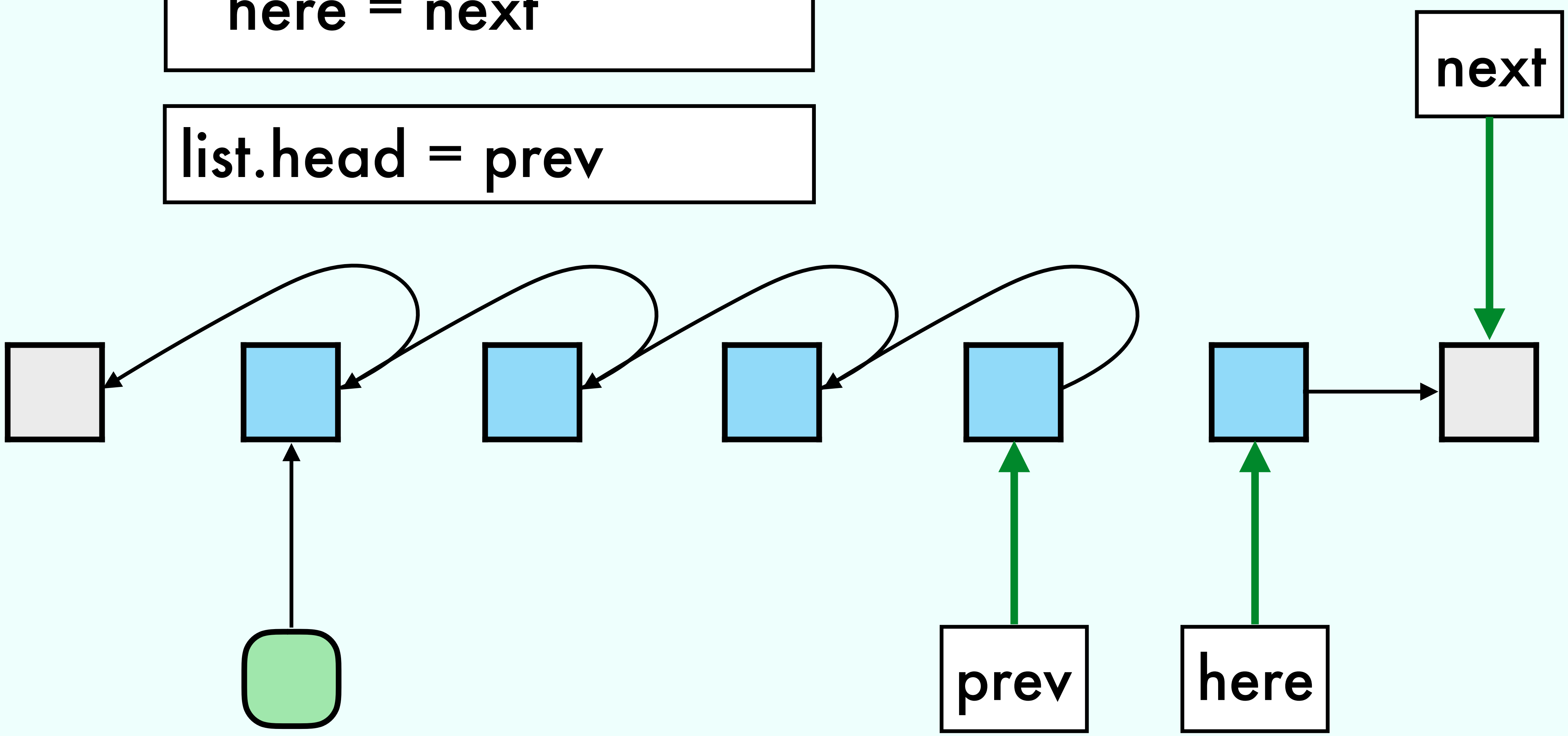
Last Iteration and Conclusion

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

```
list.head = prev
```

Shifting

Reverse



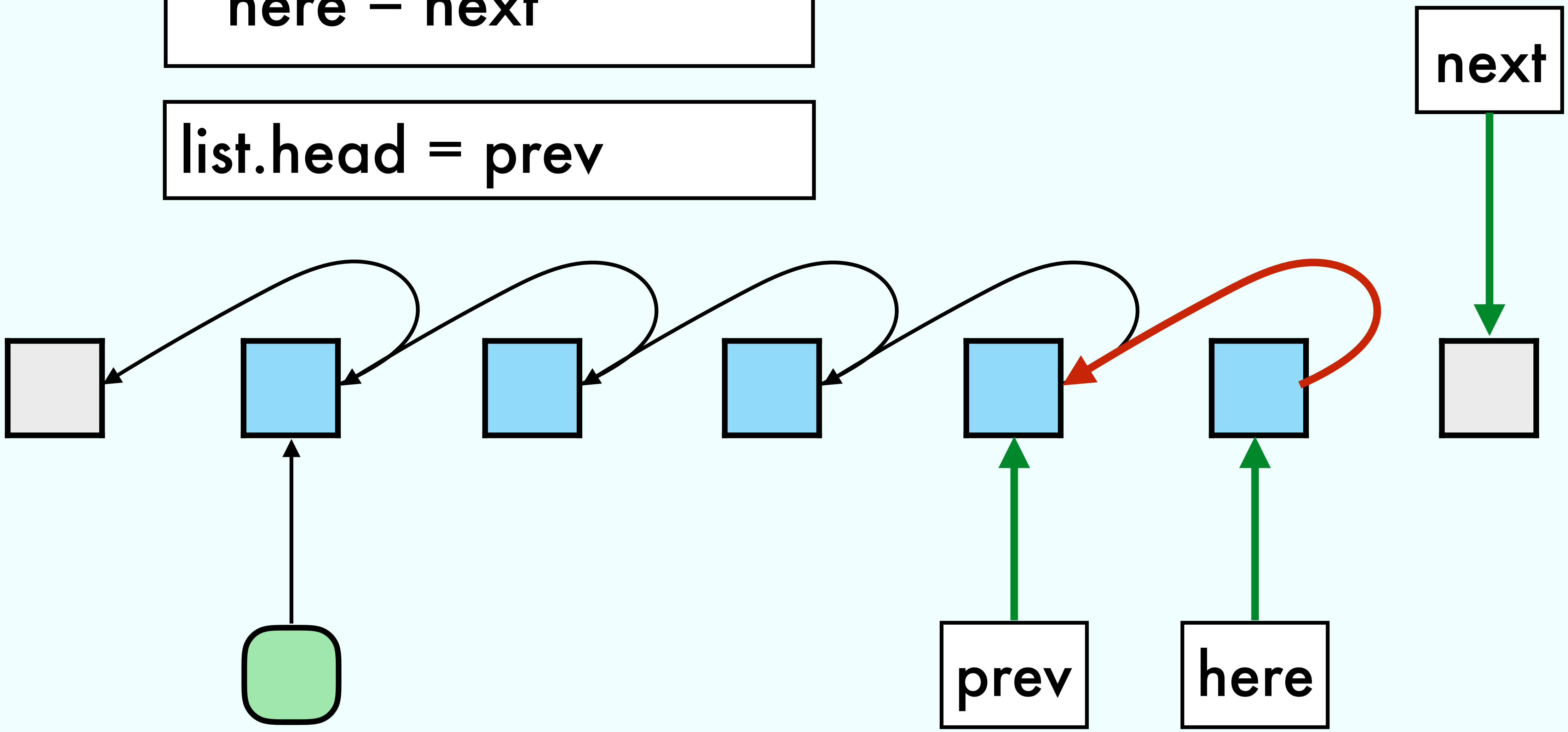
Last Iteration and Conclusion

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

```
list.head = prev
```

Shifting

Reverse

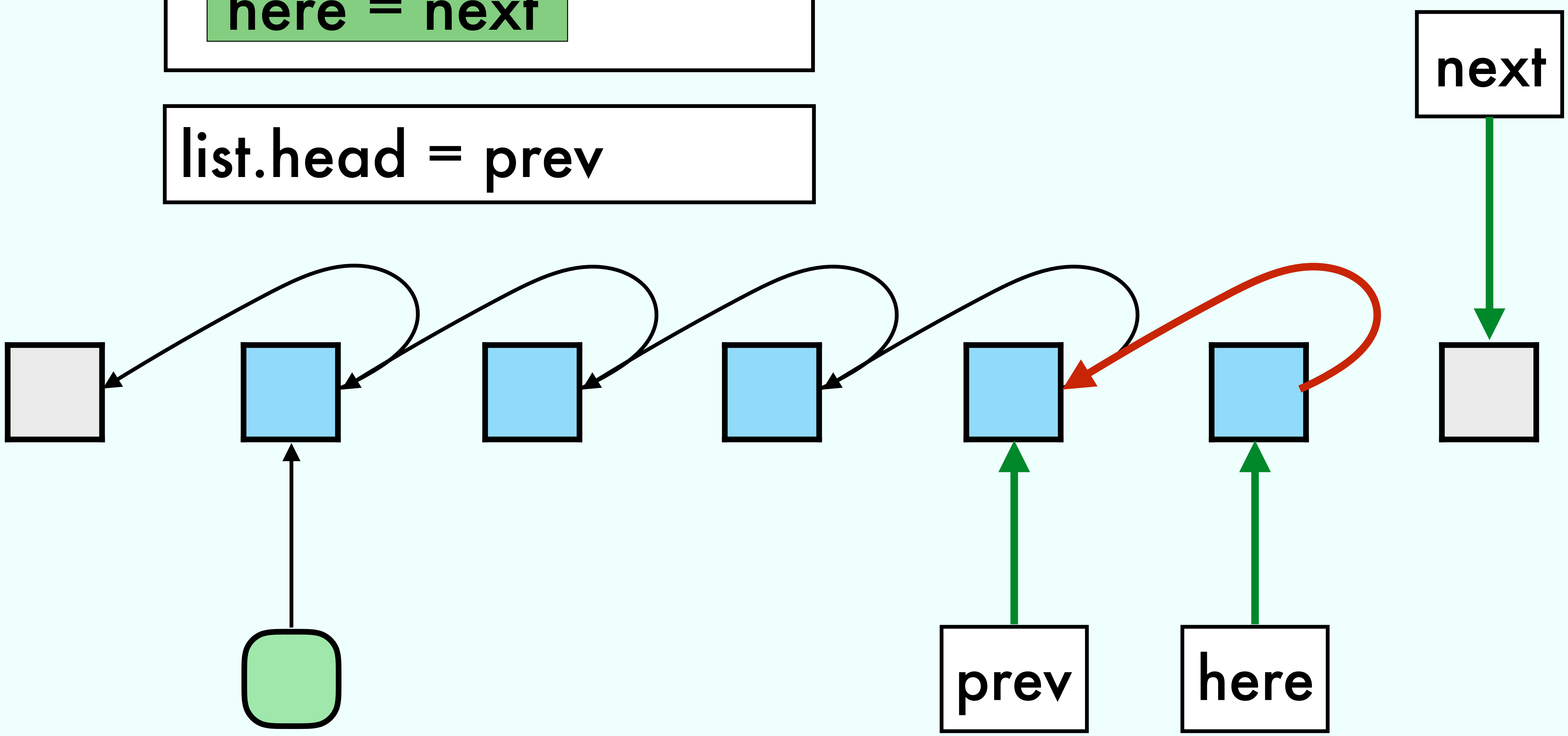


Last Iteration and Conclusion

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

Shifting

Reverse



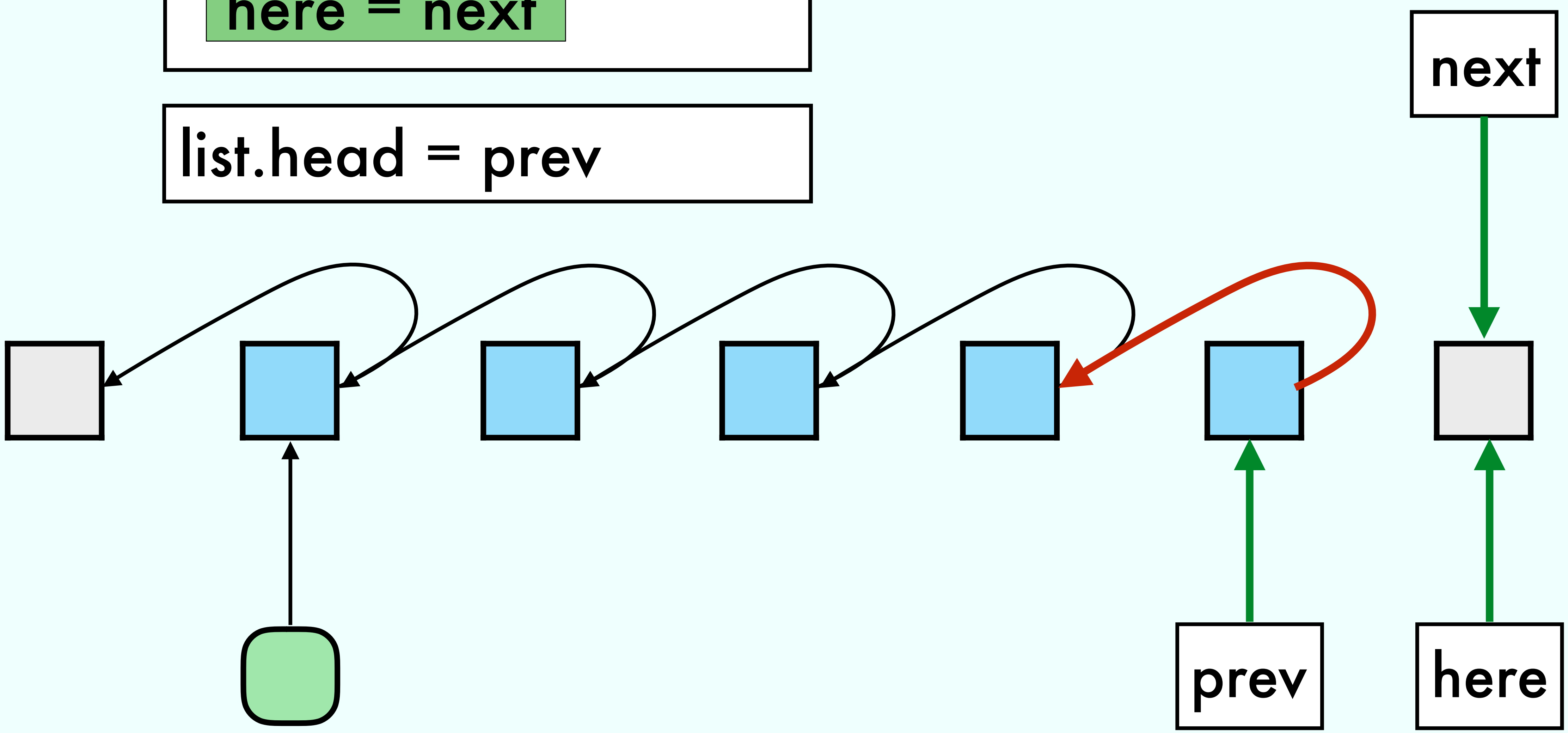
Last Iteration and Conclusion

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

```
list.head = prev
```

Shifting

Reverse



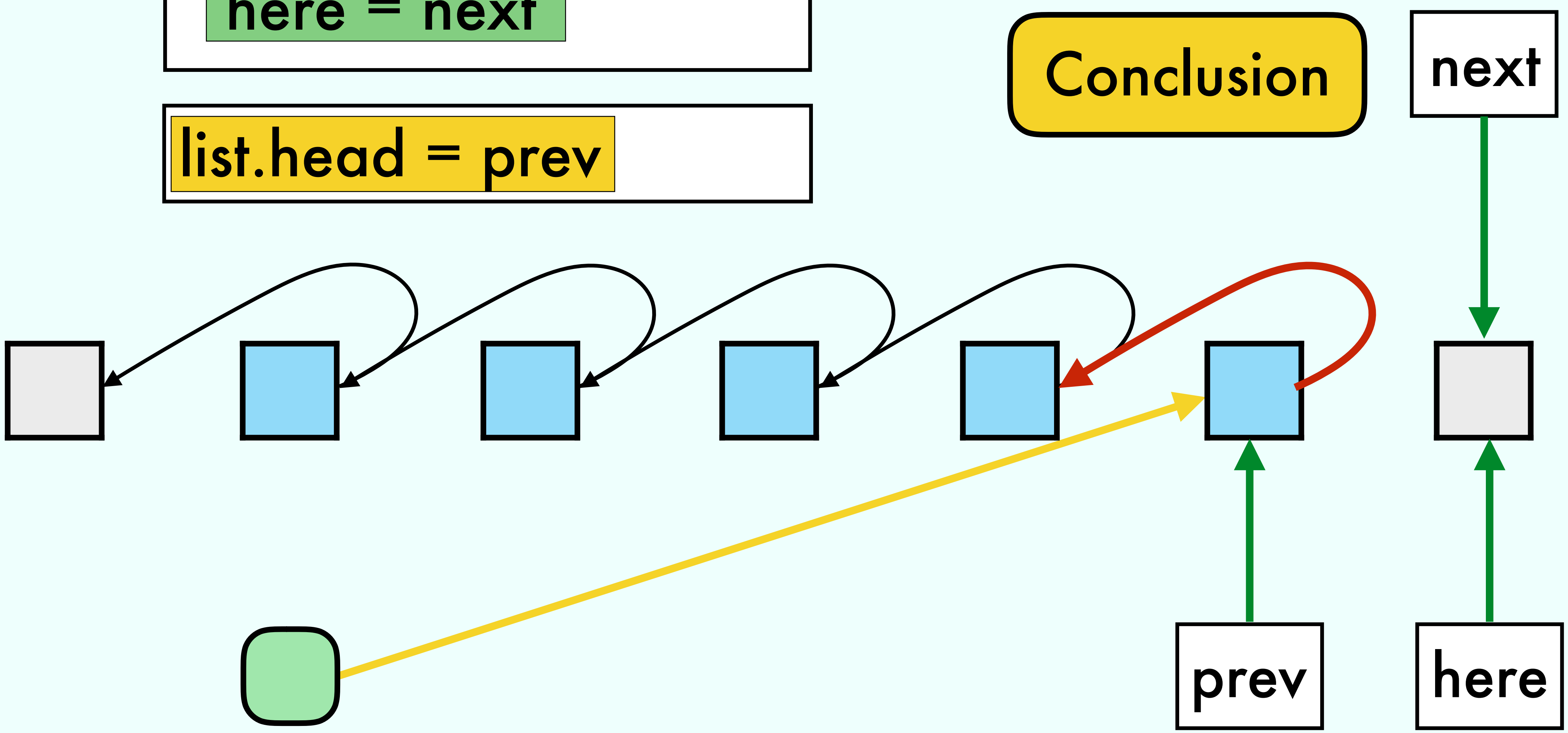
Last Iteration and Conclusion

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

Shifting

Reverse

Conclusion

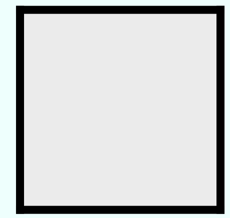


Exercise 5 from13/04

Dynamic Array with operations:

- `new()` // Create empty array with length 1
- `ins(x)` // Insert in first empty position
- `del()` // Remove last element

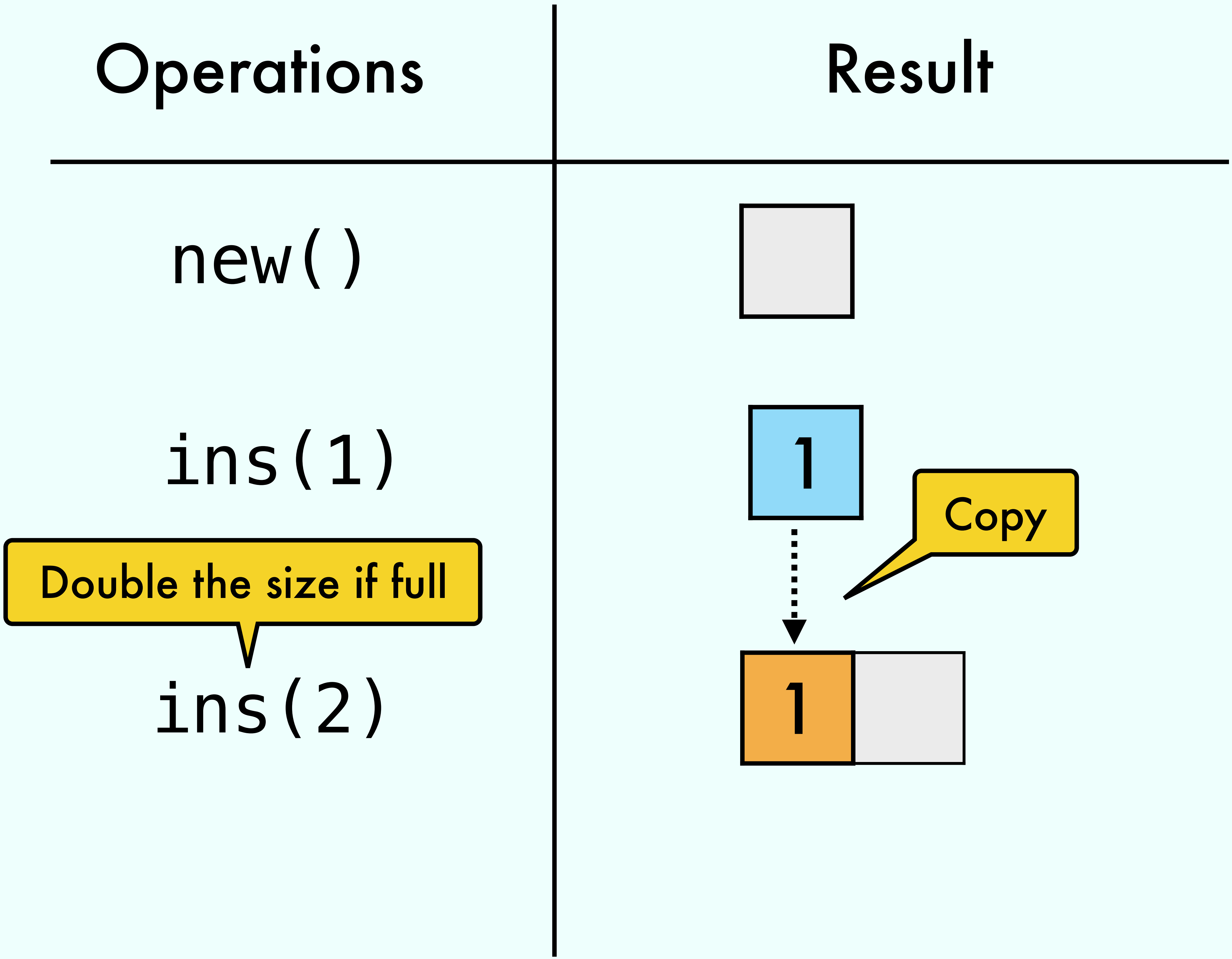
Operations	Result

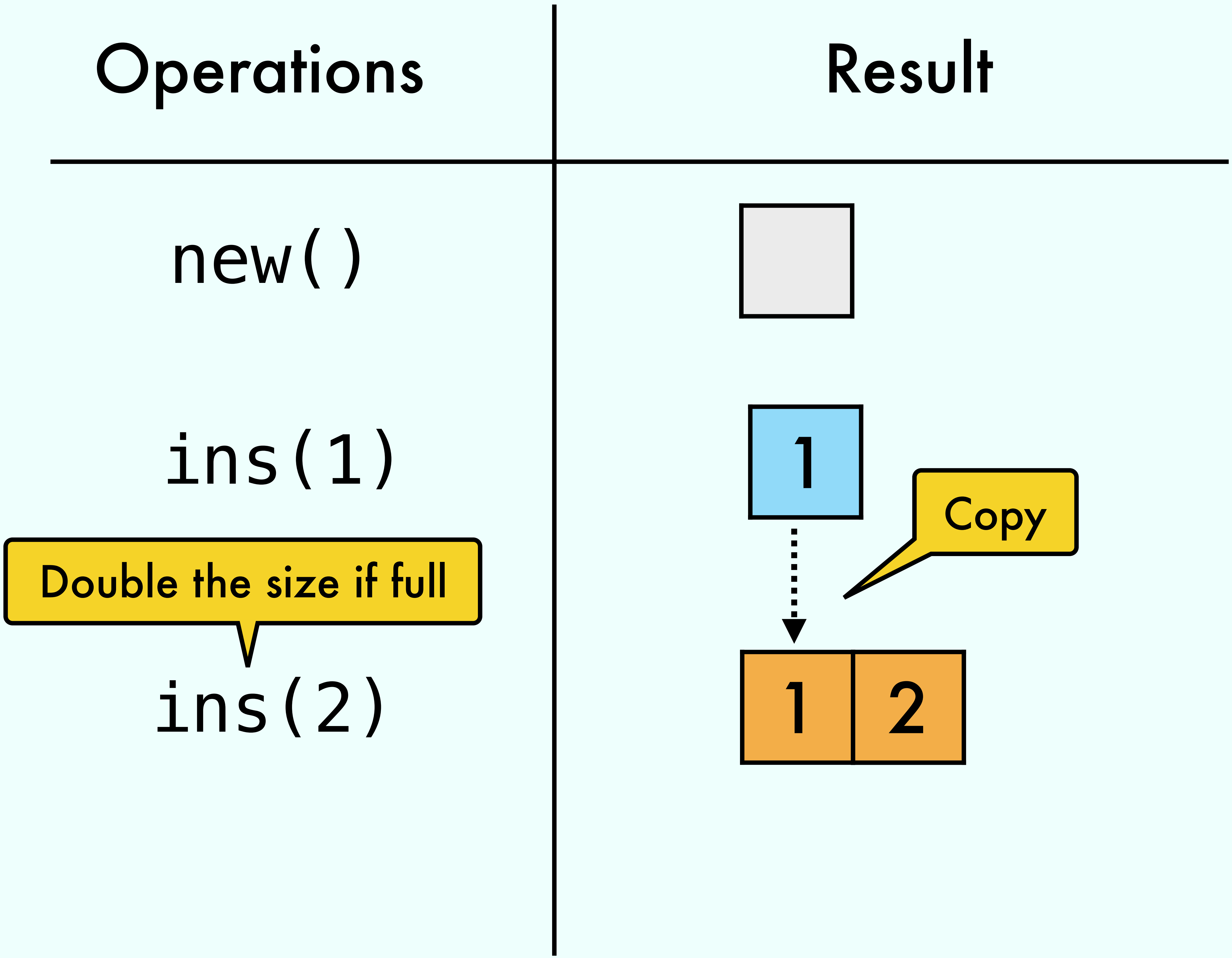
Operations	Result
new( )	

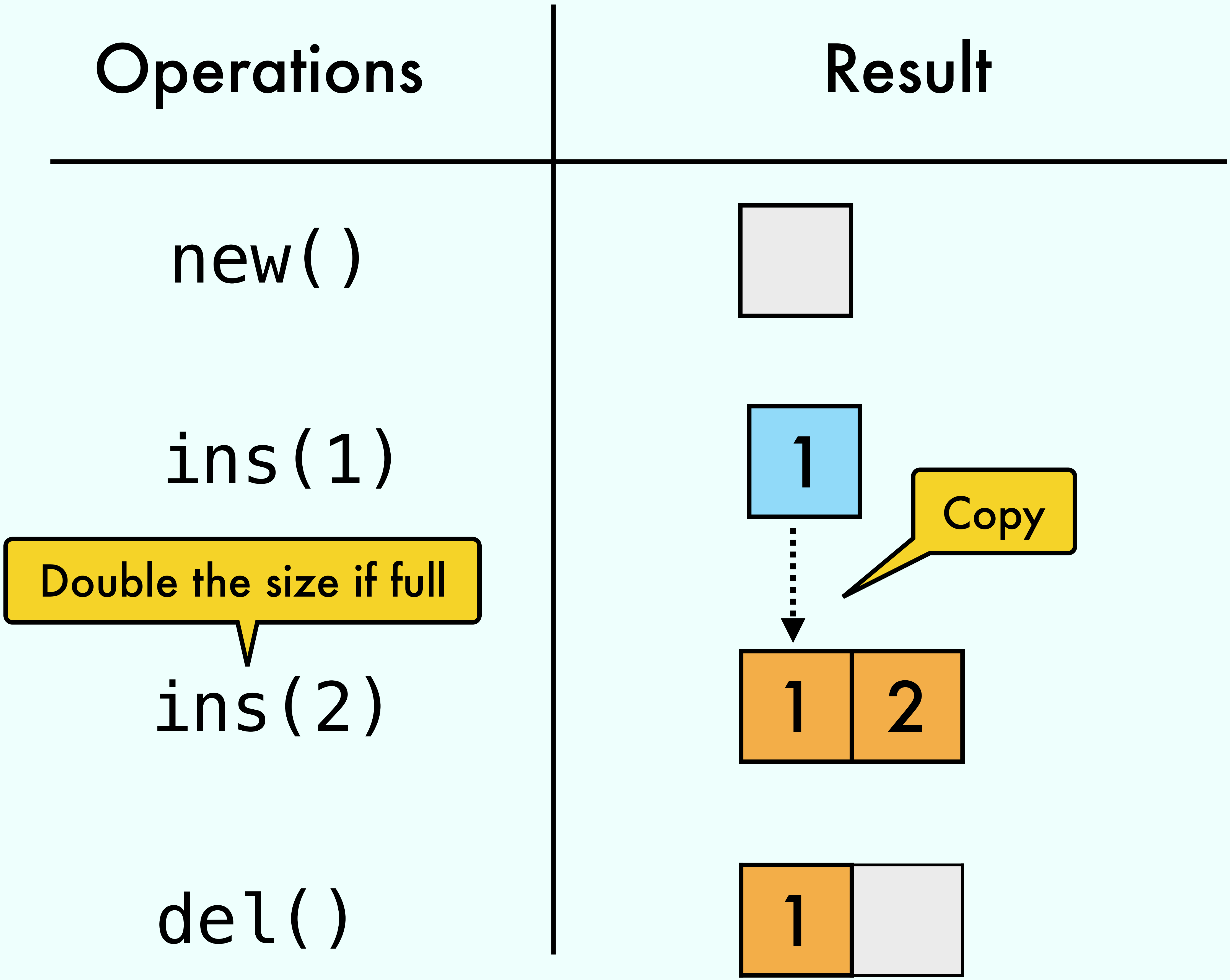
Operations	Result
`new()`	
`ins(1)`	1

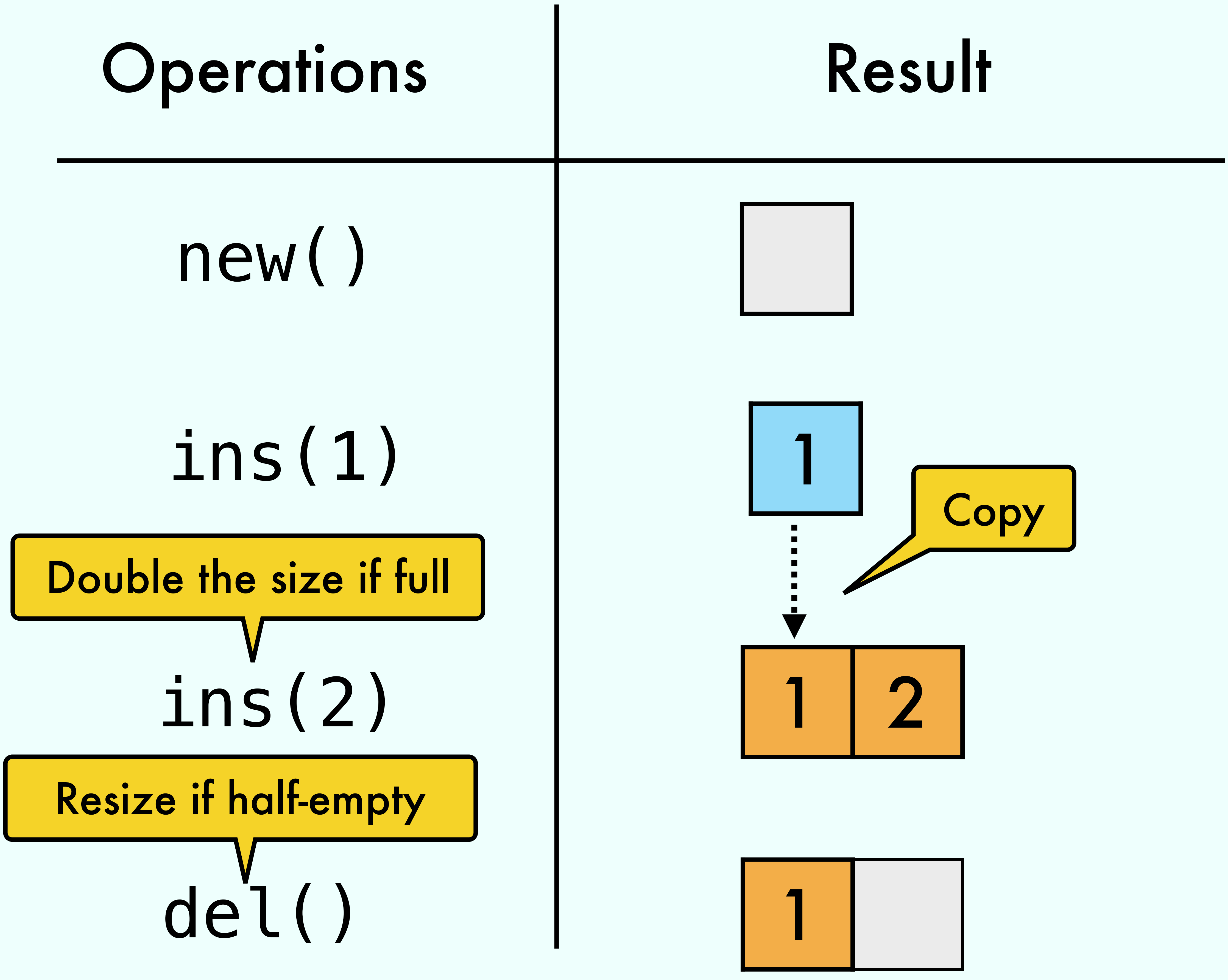
Operations	Result
`new()`	
`ins(1)`	1
`ins(2)`	

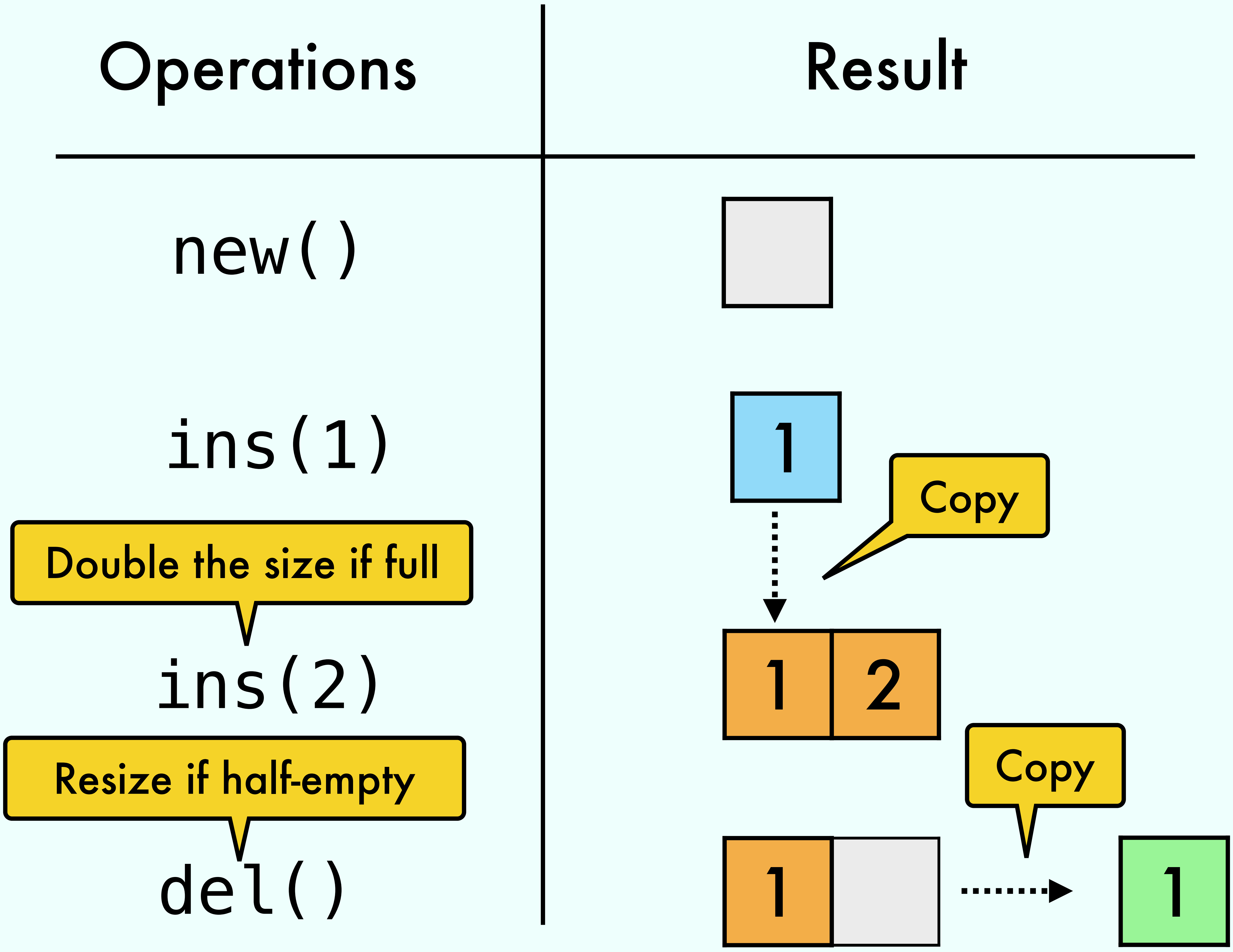
Operations	Result
`new()`	
`ins(1)`	1
Double the size if full `ins(2)`	


Operations
Result
new()
ins(1)
1
Copy
Double the size if full
ins(2)
1


Operations
Result
new()
ins(1)
1
Copy
Double the size if full
ins(2)
1
2


Operations
Result
new()
ins(1)
1
Copy
Double the size if full
ins(2)
1
2
del()
1


Operations
Result
new()
ins(1)
1
Copy
Double the size if full
ins(2)
1
2
Resize if half-empty
del()
1


Operations
Result
new()
ins(1)
1
Copy
Double the size if full
ins(2)
1
2
Copy
Resize if half-empty
del()
1
1

Exercise 5 from13/04

For every N exists

a sequence of N operations S_N

such that

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