Basic Data Structures: Stacks and Queues

References

See the chapter 10.1 in [CLRS] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms (3rd Edition). MIT Press and McGraw-Hill. 2009.

Data Structures

Data Structures and Algorithms

Outline

Definition

Stack: Abstract data type with the following operations:

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key
- `Key Pop()`: removes and returns most recently-added key

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key
- `Key Pop()`: removes and returns most recently-added key
- `Boolean Empty()`: are there any elements?

Balanced Brackets

Input: A string *str* consisting of '(', ')', '[', ']' characters.

Output: Return whether or not the string's parentheses and square brackets are balanced.

Balanced Brackets

Balanced:

- “([])[]()”,
- “((([([])]))())”

Unbalanced:

- “([]()”
- “][”

IsBalanced(*str*)

```
Stack stack
for char in str:
  if char in ['(', '[']:
    stack.Push(char)
  else:
    if stack.Empty(): return False
    top ← stack.Pop()
    if (top = '[' and char != ']') or
      (top = '(' and char != ')'):
      return False
return stack.Empty()
```

IsBalanced(*str*)

```
Stack stack
for char in str:
  if char in ['(', '[']:
    stack.Push(char)
  else:
    if stack.Empty(): return False
    top ← stack.Pop()
    if (top = '[' and char != ']') or
      (top = '(' and char != ')'):
      return False
return stack.Empty()
```

e.g. "((([([])])))"

IsBalanced(*str*)

```
Stack stack
for char in str:
  if char in ['(', '[']:
    stack.Push(char)
  else:
    if stack.Empty(): return False
    top ← stack.Pop()
    if (top = '[' and char != ']') or
      (top = '(' and char != ')'):
      return False
return stack.Empty()
```

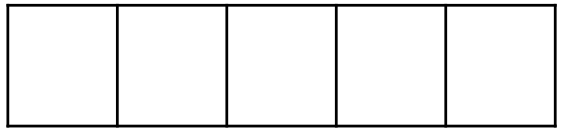
e.g. "(((”

Question

Given the unbalanced string "()([]", what character is on the top of the stack when the for loop is finished?

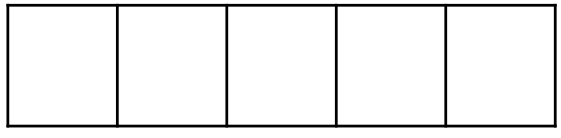
Stack Implementation with Array

numElements: 0



Stack Implementation with Array

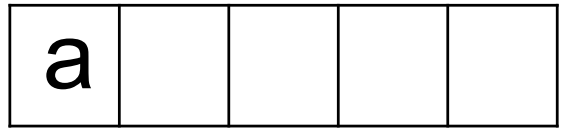
numElements: 0



`Push(a)`

Stack Implementation with Array

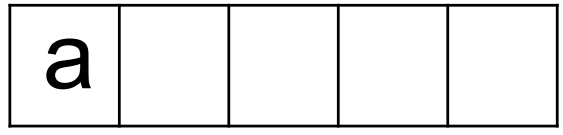
numElements: 1



`Push(a)`

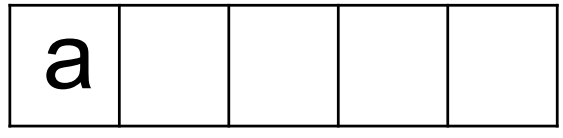
Stack Implementation with Array

numElements: 1



Stack Implementation with Array

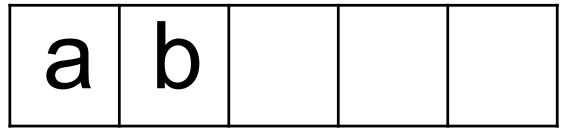
numElements: 1



`Push(b)`

Stack Implementation with Array

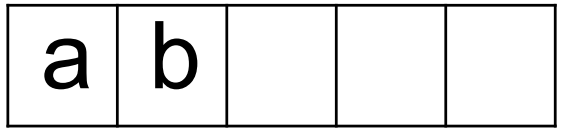
numElements: 2



`Push(b)`

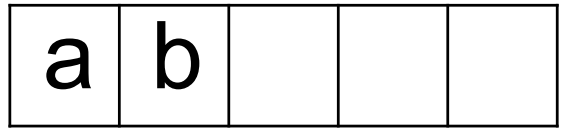
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

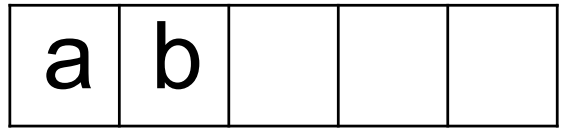
numElements: 2



`Top()`

Stack Implementation with Array

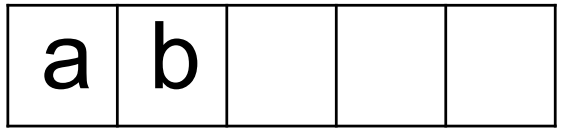
numElements: 2



`Top()` → `b`

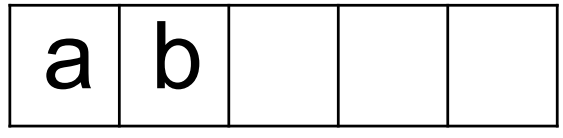
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

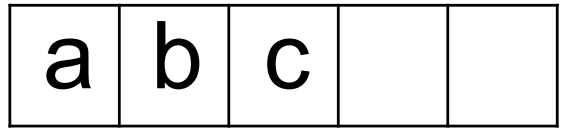
numElements: 2



`Push(c)`

Stack Implementation with Array

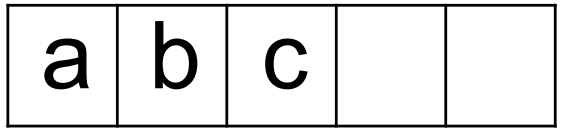
numElements: 3



`Push(c)`

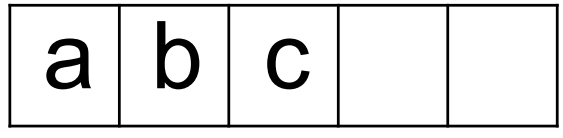
Stack Implementation with Array

numElements: 3



Stack Implementation with Array

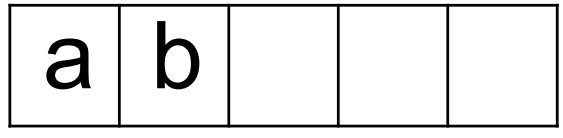
numElements: 3



`Pop()`

Stack Implementation with Array

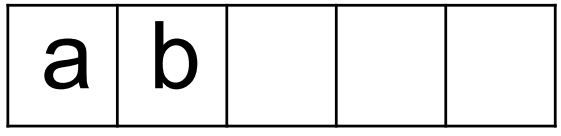
numElements: 2



`Pop()` → `c`

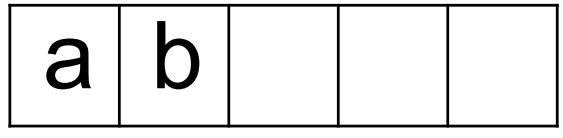
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

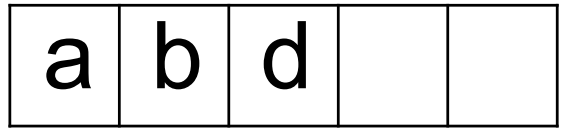
numElements: 2



`Push(d)`

Stack Implementation with Array

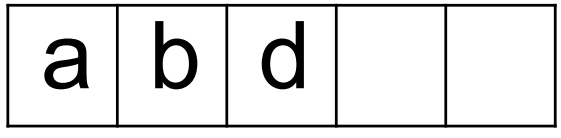
numElements: 3



`Push(d)`

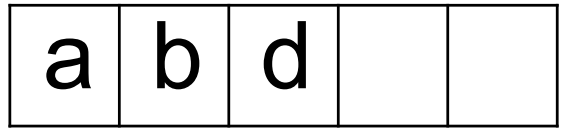
Stack Implementation with Array

numElements: 3



Stack Implementation with Array

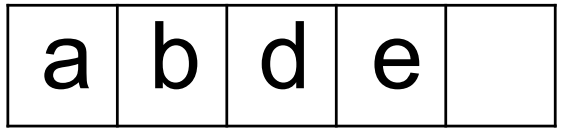
numElements: 3



`Push(e)`

Stack Implementation with Array

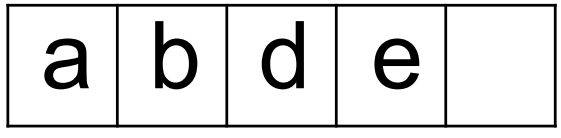
numElements: 4



`Push(e)`

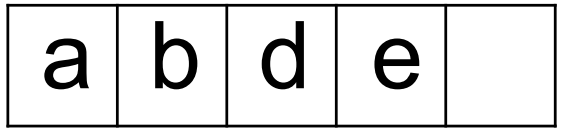
Stack Implementation with Array

numElements: 4



Stack Implementation with Array

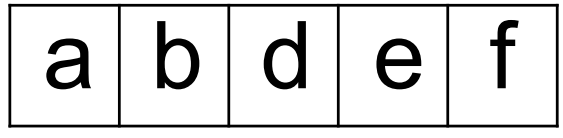
numElements: 4



`Push(f)`

Stack Implementation with Array

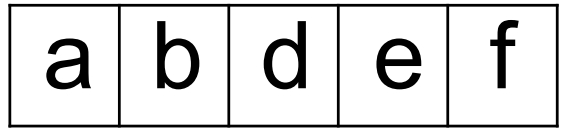
numElements: 5



`Push(f)`

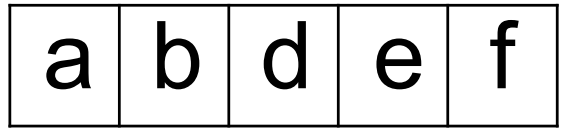
Stack Implementation with Array

numElements: 5



Stack Implementation with Array

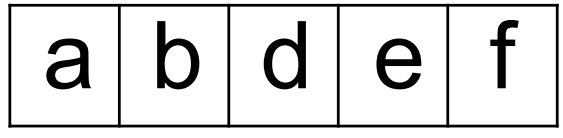
numElements: 5



`Push(g)`

Stack Implementation with Array

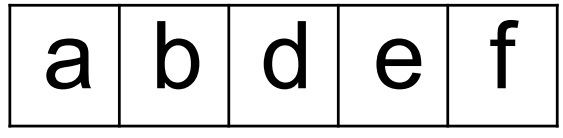
numElements: 5



`Push(g)` → `ERROR`

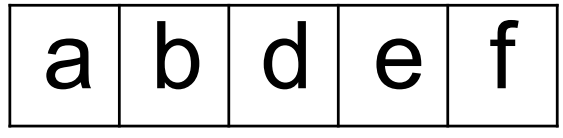
Stack Implementation with Array

numElements: 5



Stack Implementation with Array

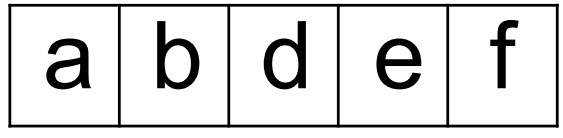
numElements: 5



`Empty()`

Stack Implementation with Array

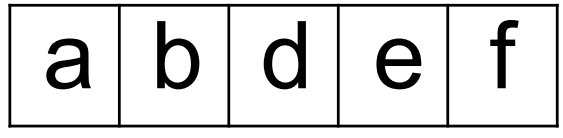
numElements: 5



`Empty()` → `False`

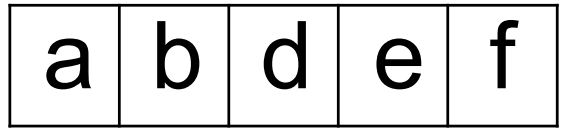
Stack Implementation with Array

numElements: 5



Stack Implementation with Array

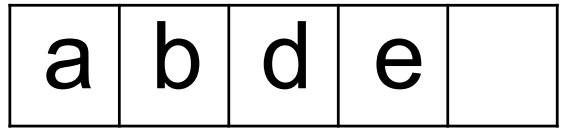
numElements: 5



Pop ()

Stack Implementation with Array

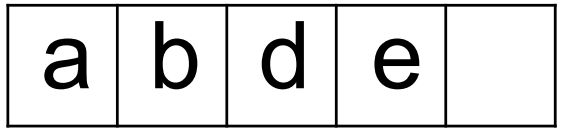
numElements: 4



`Pop()` → `f`

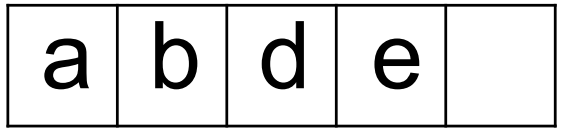
Stack Implementation with Array

numElements: 4



Stack Implementation with Array

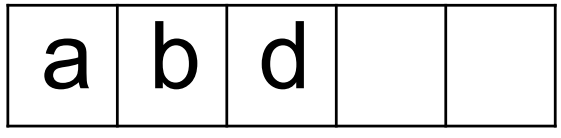
numElements: 4



Pop ()

Stack Implementation with Array

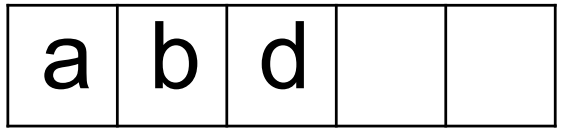
numElements: 3



`Pop()` → `e`

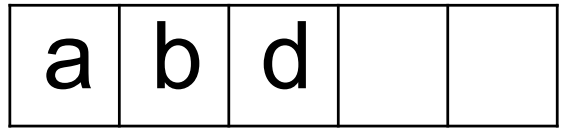
Stack Implementation with Array

numElements: 3



Stack Implementation with Array

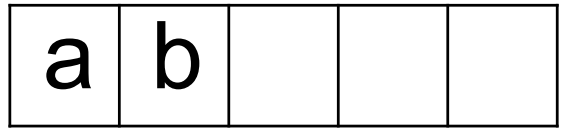
numElements: 3



Pop ()

Stack Implementation with Array

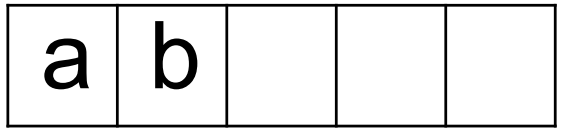
numElements: 2



`Pop()` → `d`

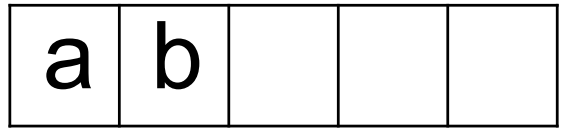
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

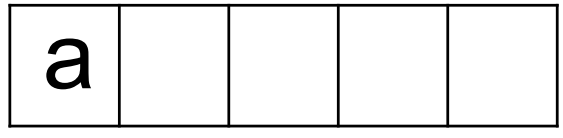
numElements: 2



Pop ()

Stack Implementation with Array

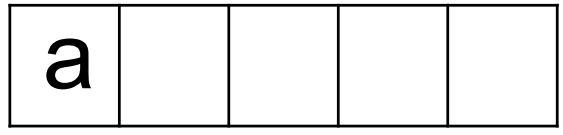
numElements: 1



`Pop()` → `b`

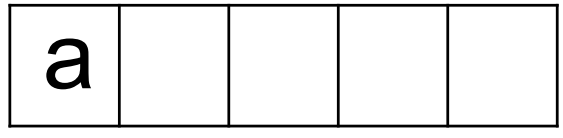
Stack Implementation with Array

numElements: 1



Stack Implementation with Array

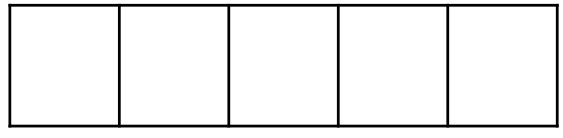
numElements: 1



Pop ()

Stack Implementation with Array

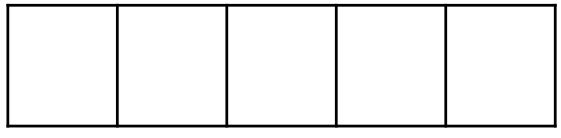
numElements: 0



`Pop()` → `a`

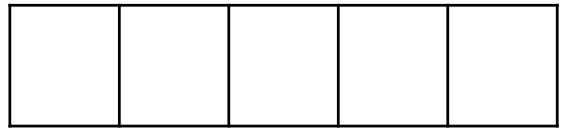
Stack Implementation with Array

numElements: 0



Stack Implementation with Array

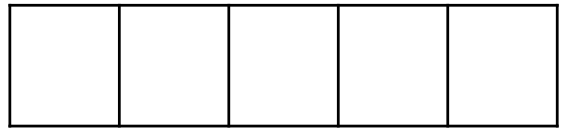
numElements: 0



`Empty()`

Stack Implementation with Array

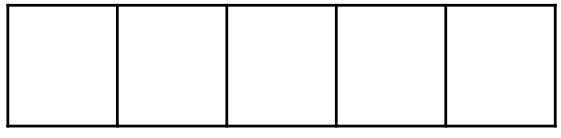
numElements: 0



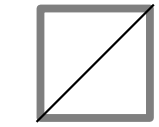
`Empty()` → `True`

Stack Implementation with Array

numElements: 0

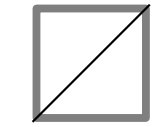


Stack Implementation with Linked List



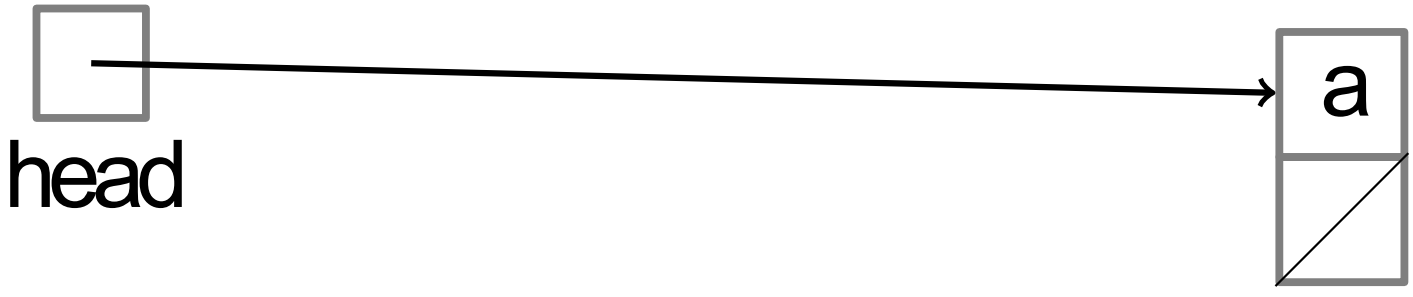
head

Stack Implementation with Linked List



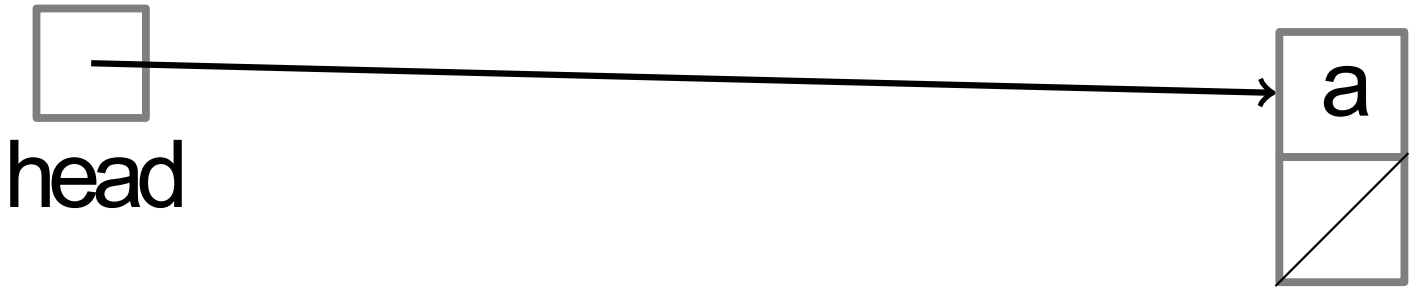
`Push(a)`

Stack Implementation with Linked List

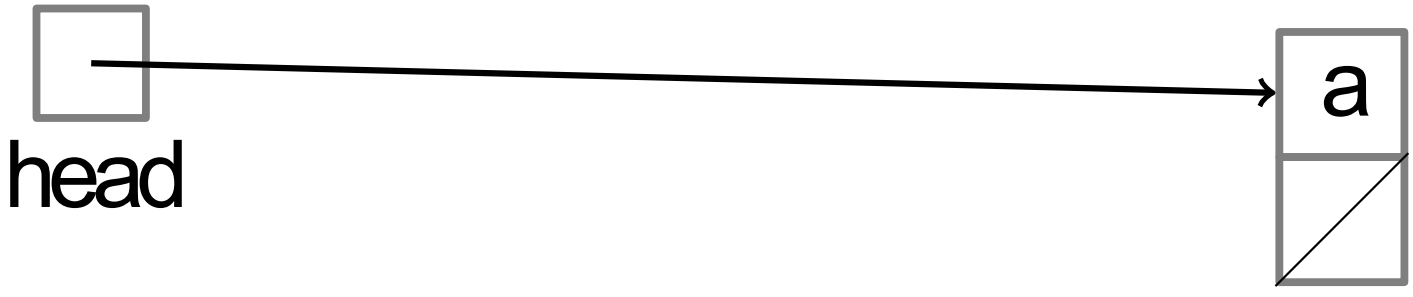


`Push(a)`

Stack Implementation with Linked List

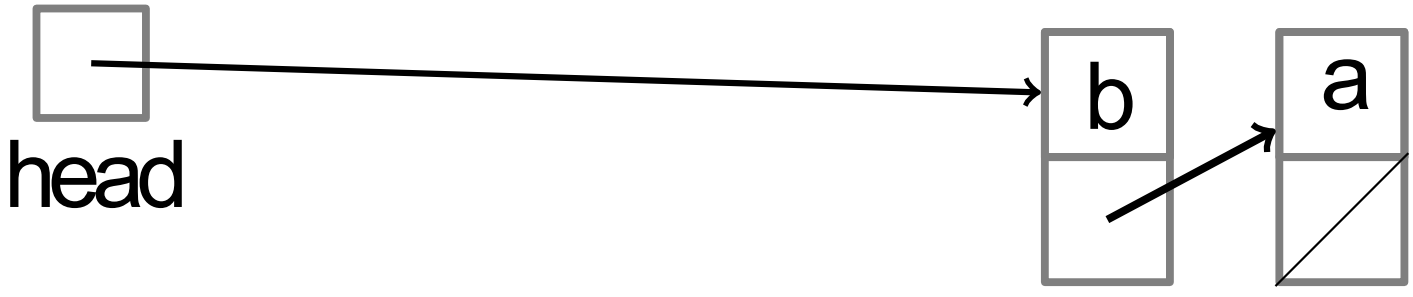


Stack Implementation with Linked List



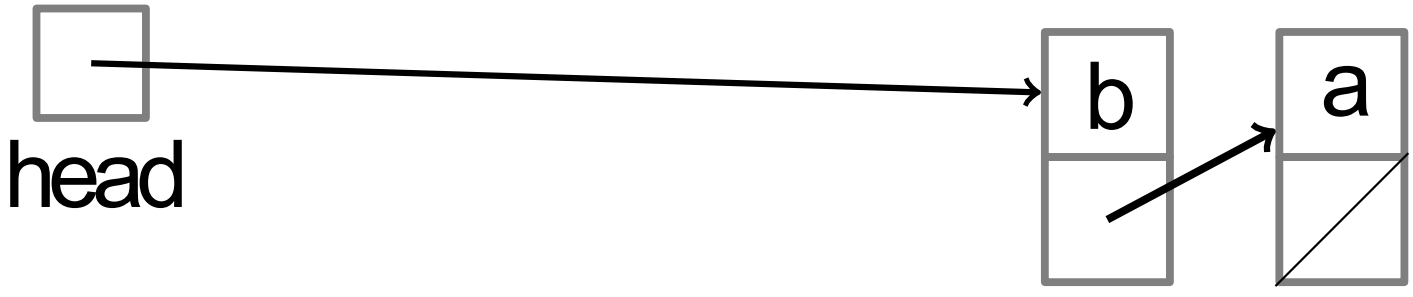
`Push(b)`

Stack Implementation with Linked List

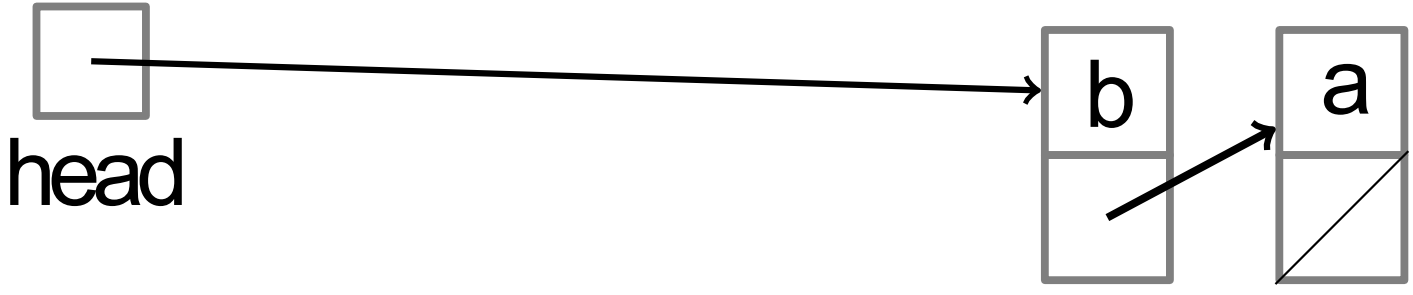


`Push(b)`

Stack Implementation with Linked List

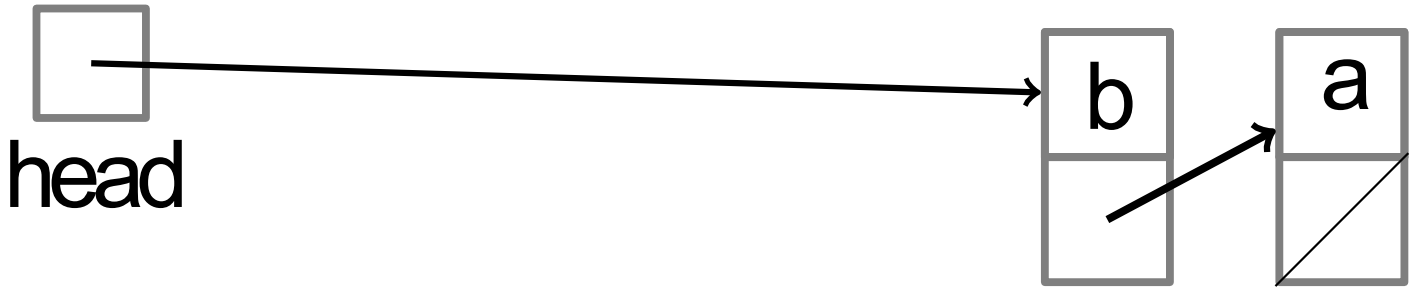


Stack Implementation with Linked List



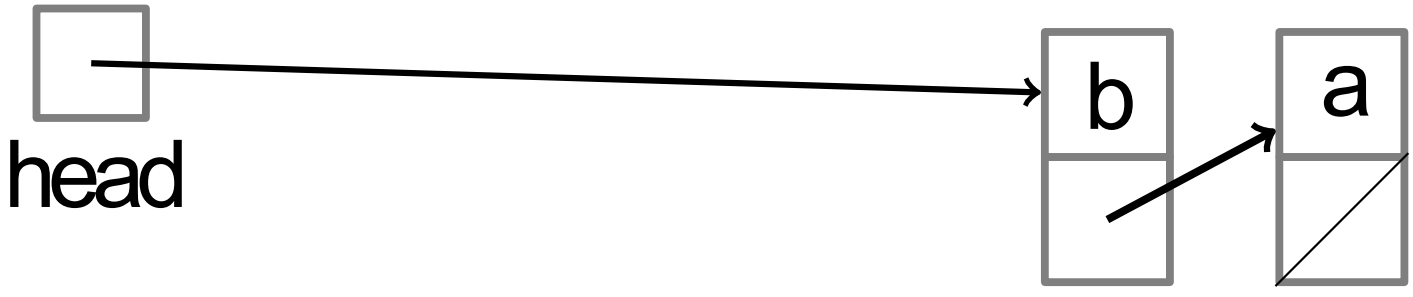
`Top()`

Stack Implementation with Linked List

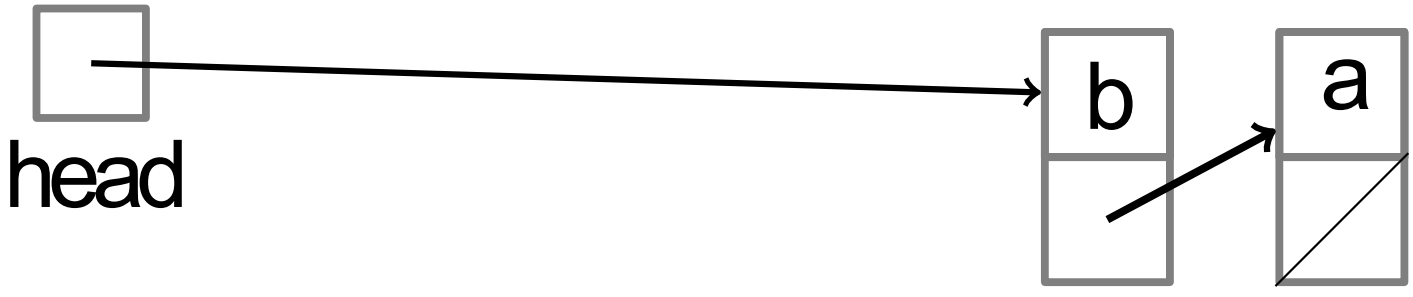


`Top()` → `b`

Stack Implementation with Linked List

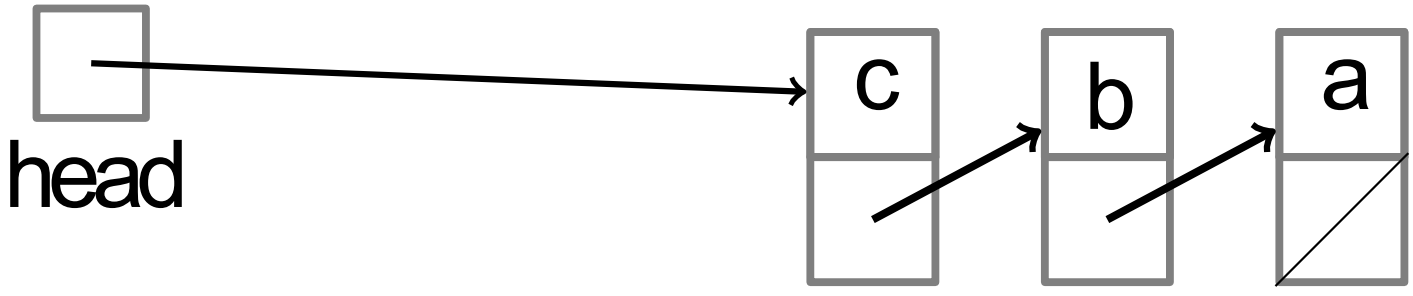


Stack Implementation with Linked List



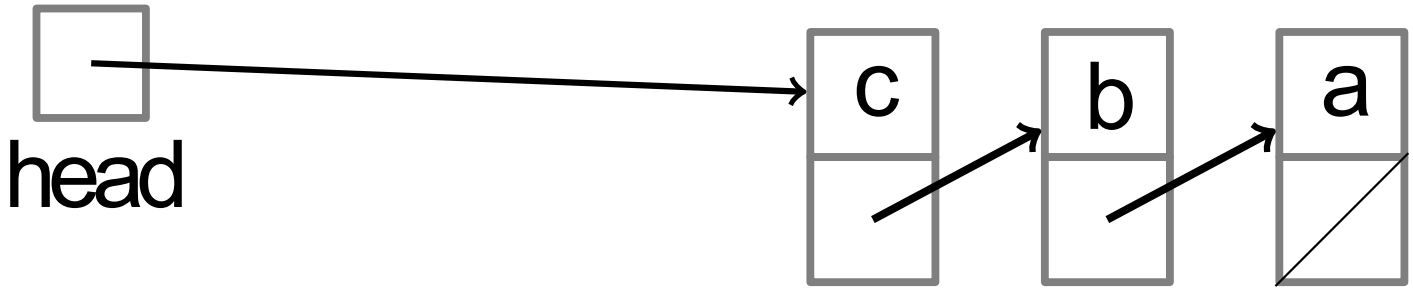
`Push(c)`

Stack Implementation with Linked List

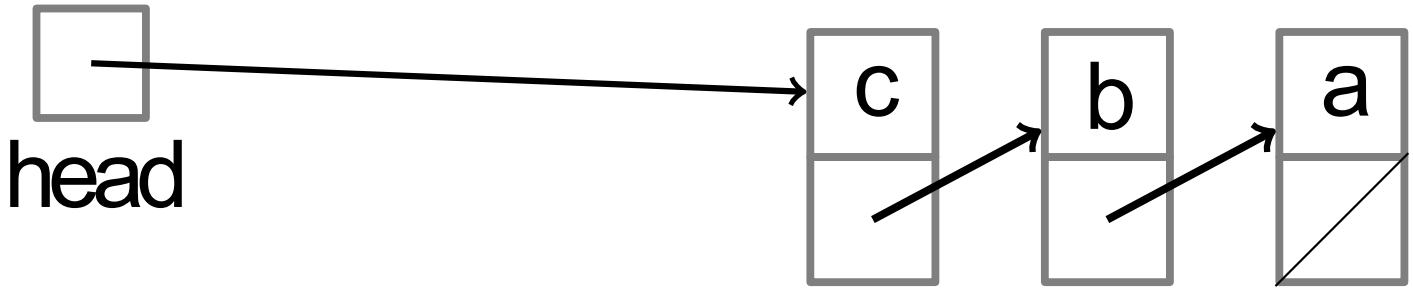


`Push(c)`

Stack Implementation with Linked List

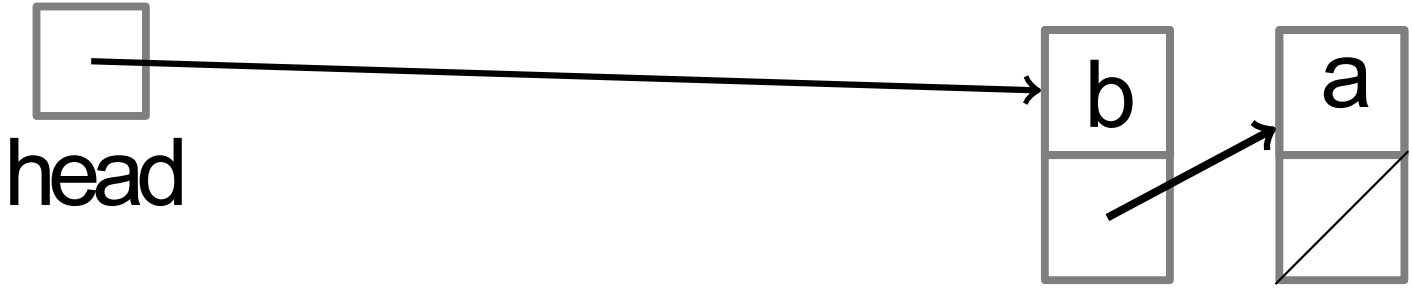


Stack Implementation with Linked List



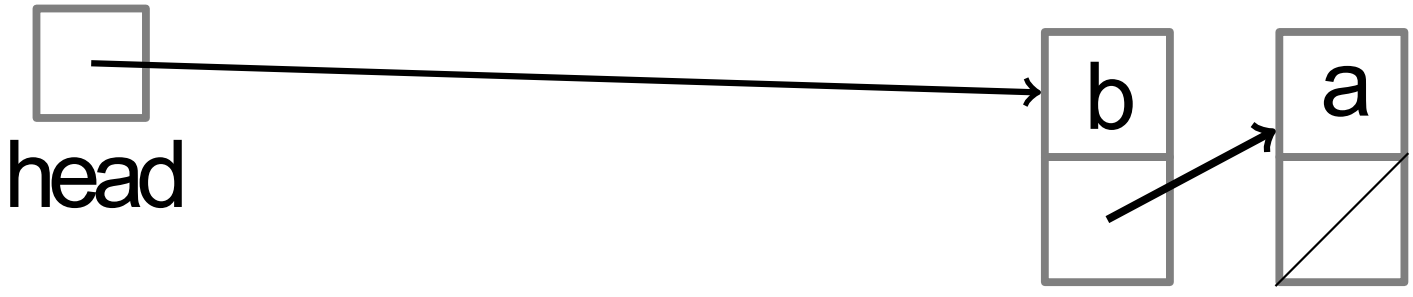
Pop ()

Stack Implementation with Linked List

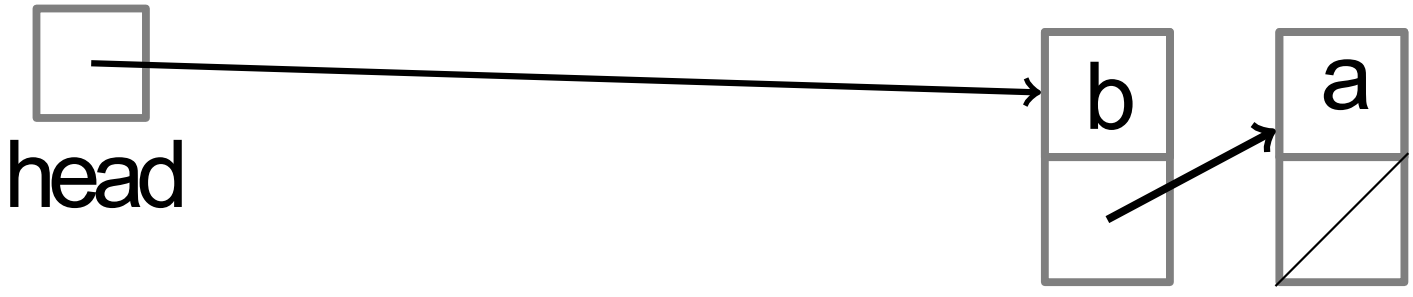


`Pop()` → `c`

Stack Implementation with Linked List

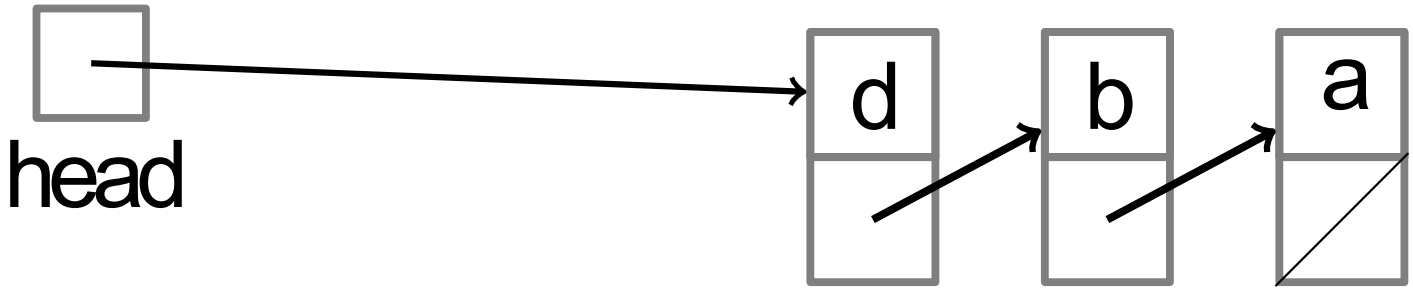


Stack Implementation with Linked List



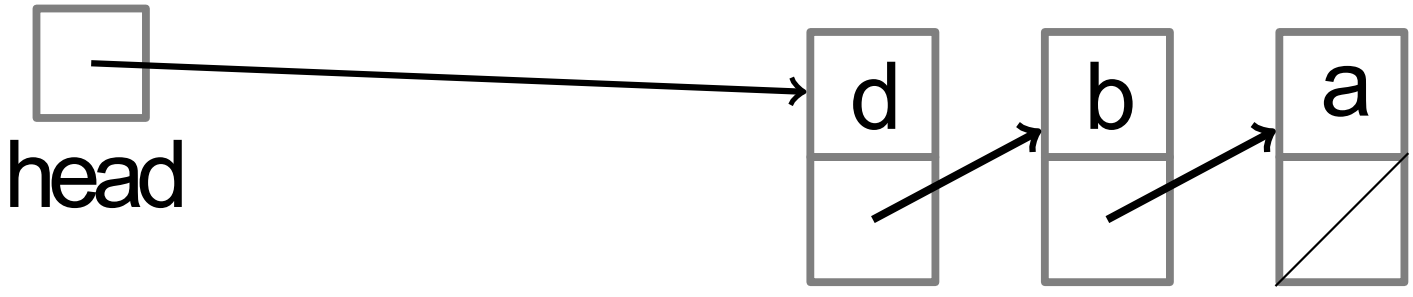
`Push(d)`

Stack Implementation with Linked List

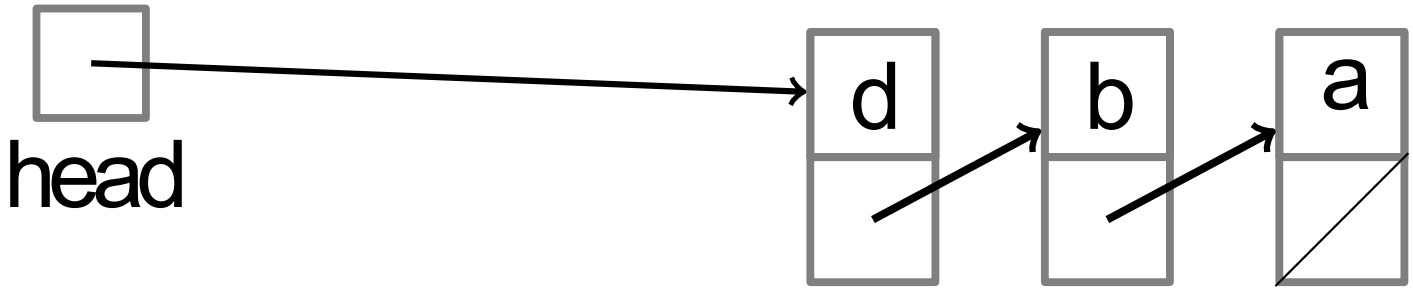


`Push(d)`

Stack Implementation with Linked List

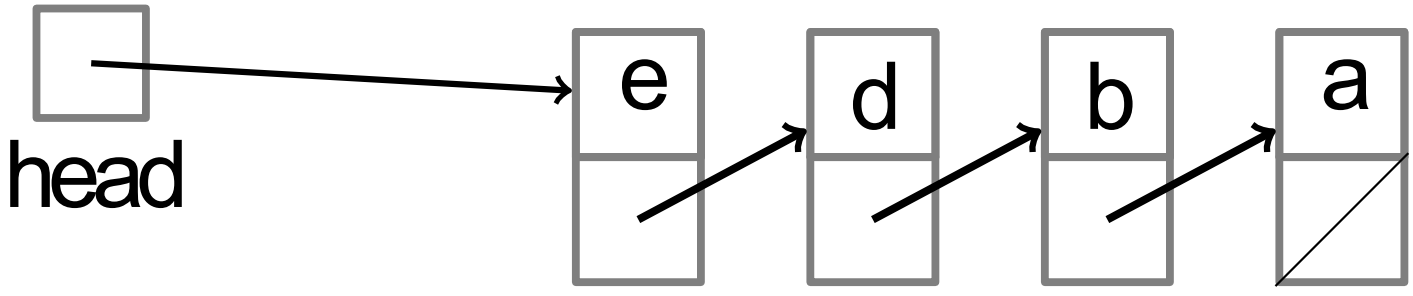


Stack Implementation with Linked List



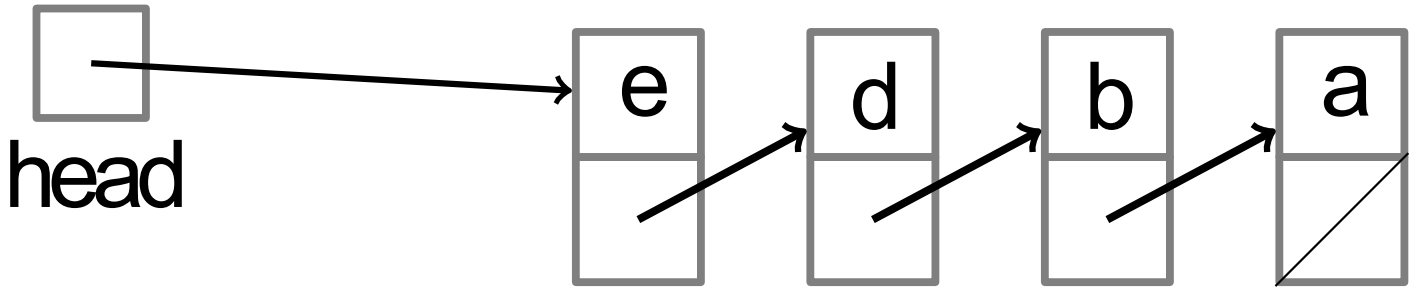
`Push(e)`

Stack Implementation with Linked List

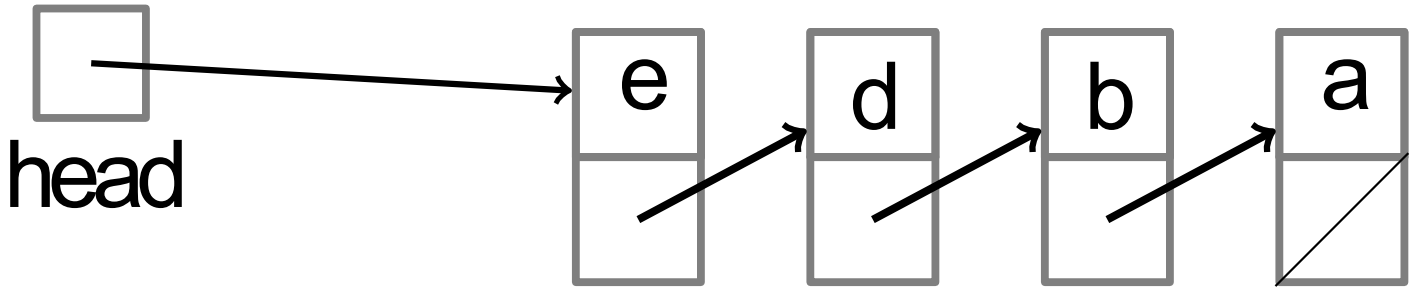


`Push(e)`

Stack Implementation with Linked List

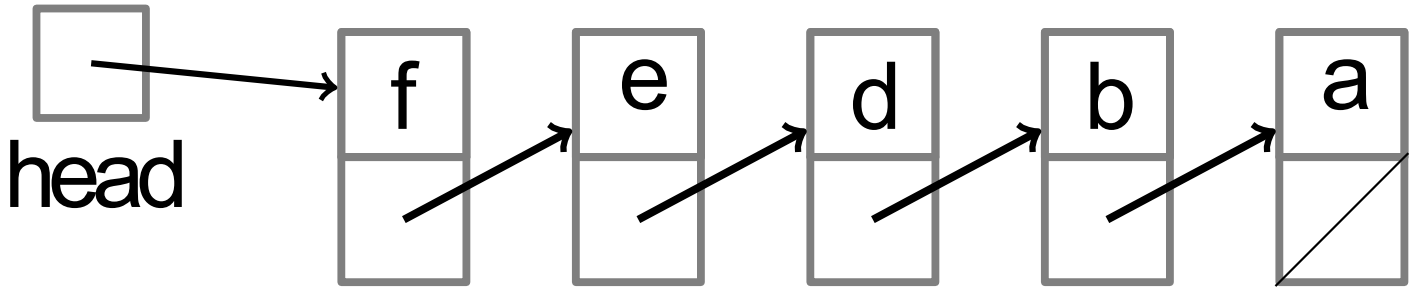


Stack Implementation with Linked List



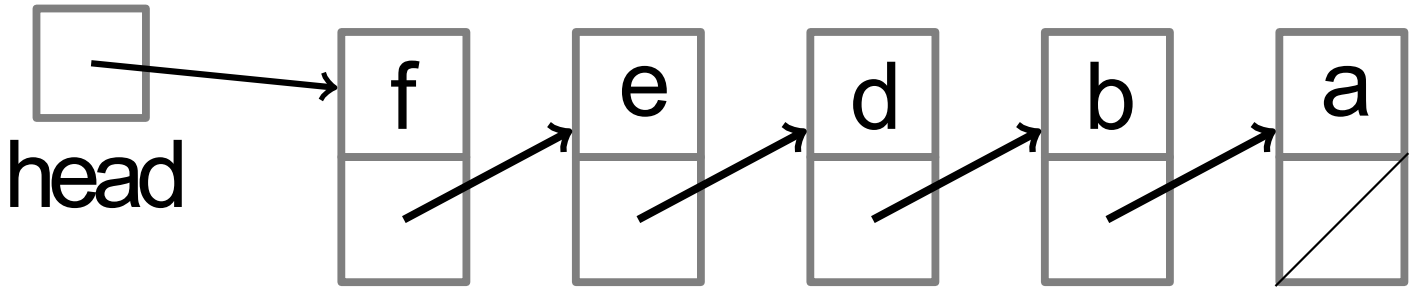
`Push(f)`

Stack Implementation with Linked List

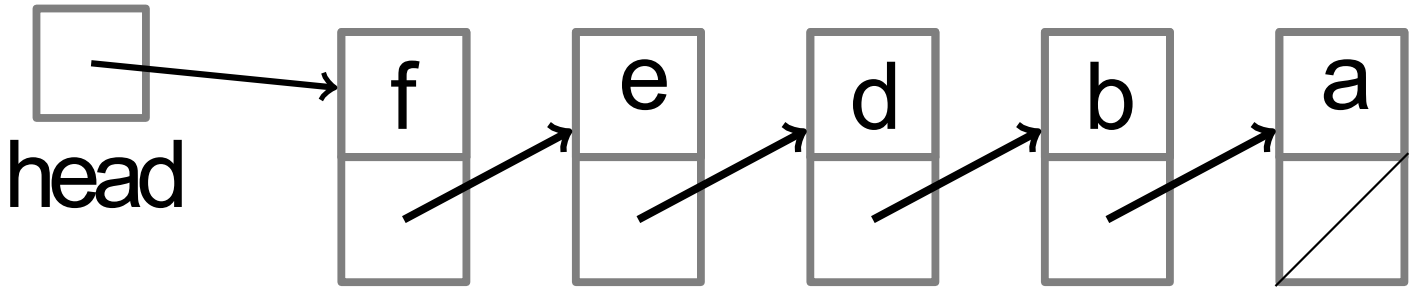


`Push(f)`

Stack Implementation with Linked List

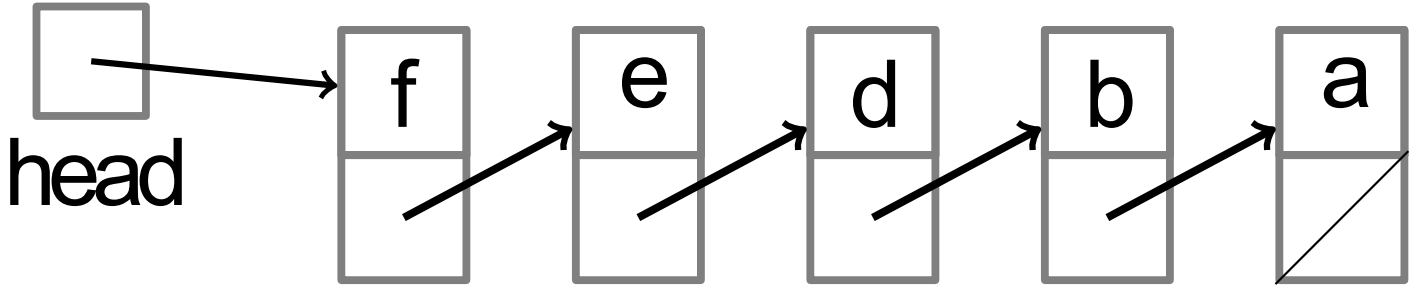


Stack Implementation with Linked List



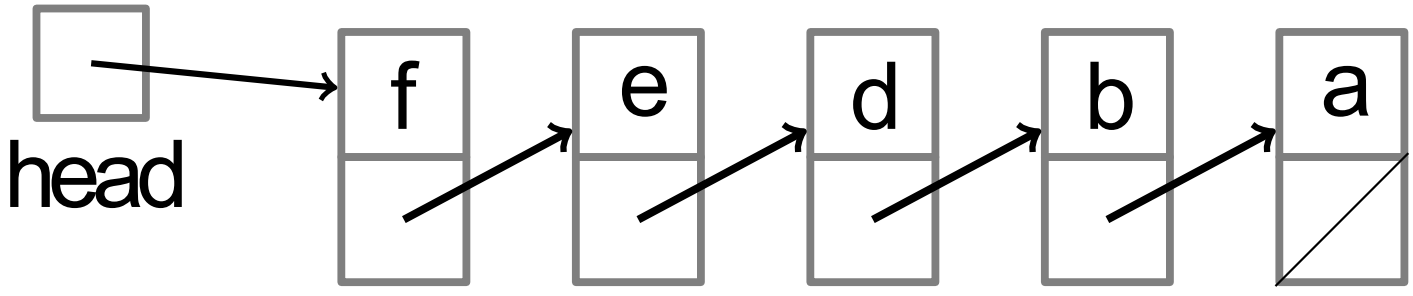
`Empty()`

Stack Implementation with Linked List

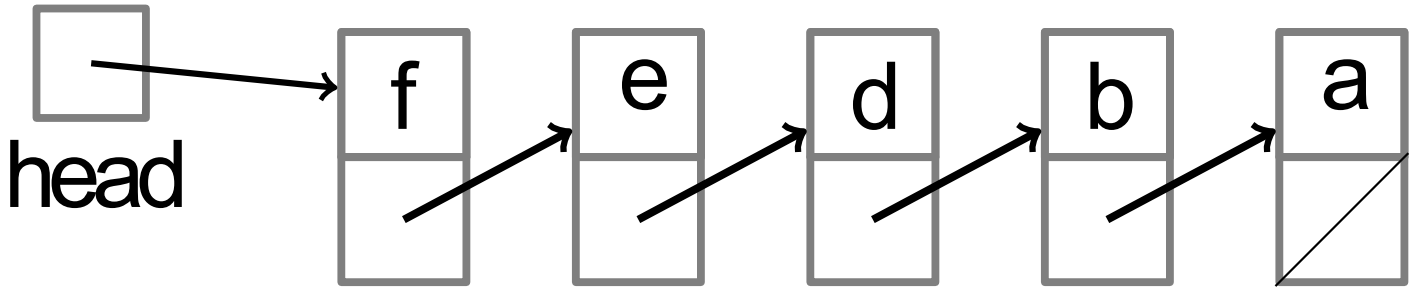


`Empty()` → `False`

Stack Implementation with Linked List

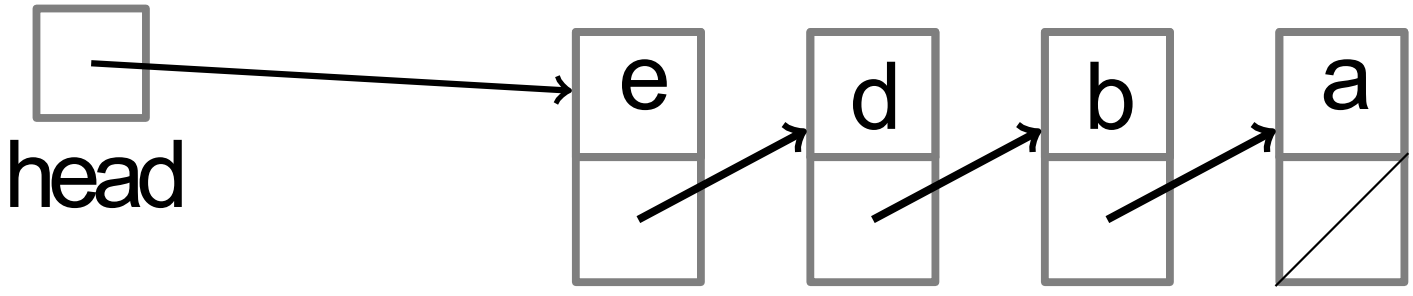


Stack Implementation with Linked List



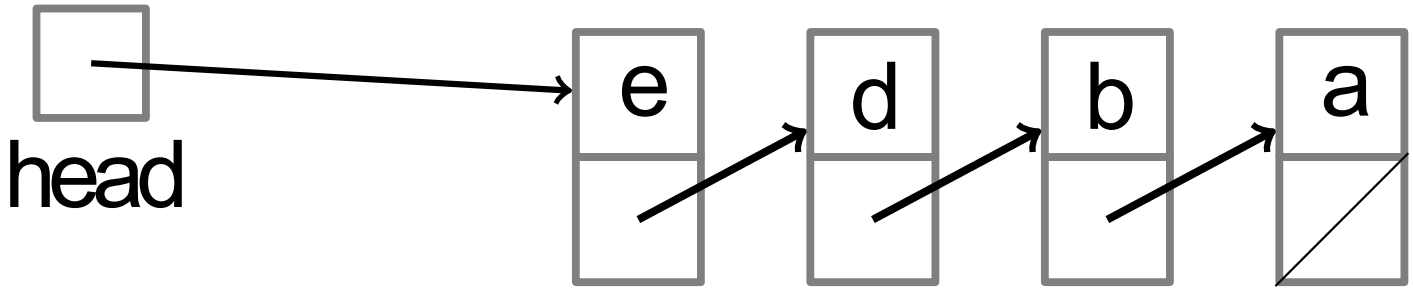
Pop()

Stack Implementation with Linked List

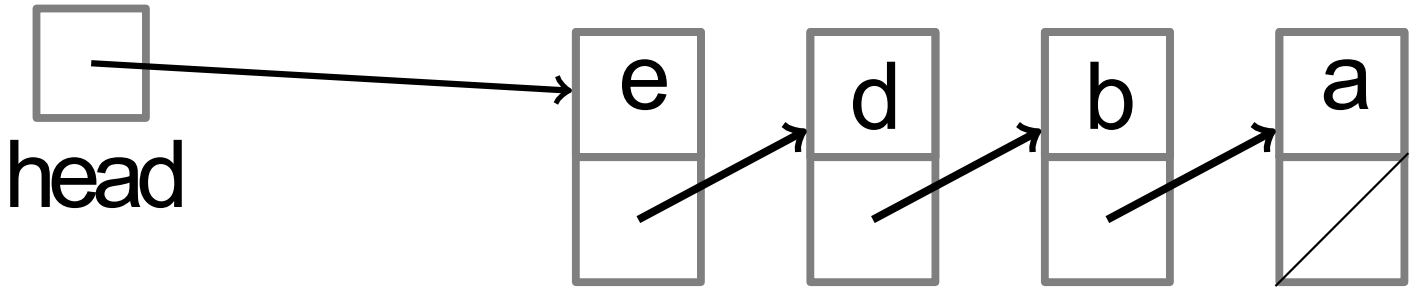


Pop()→ f

Stack Implementation with Linked List

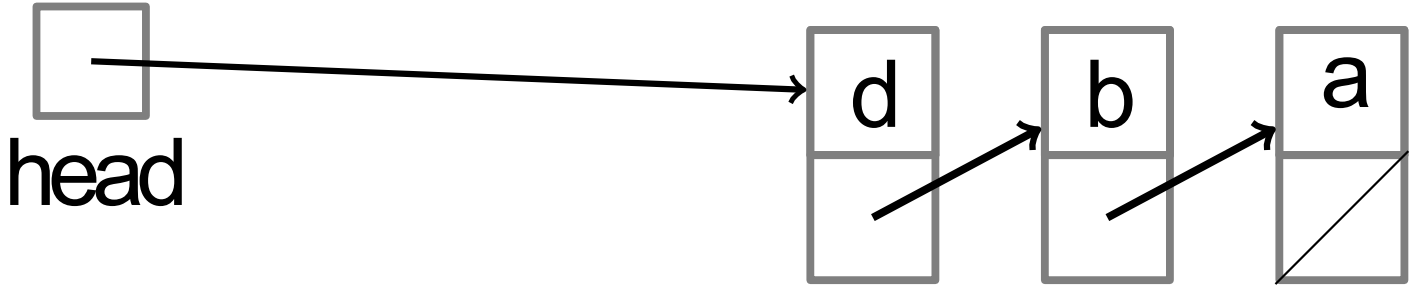


Stack Implementation with Linked List



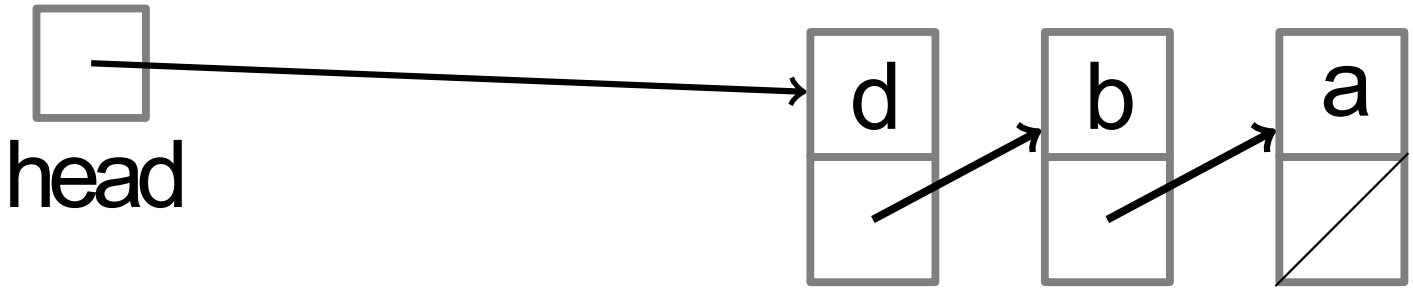
Pop()

Stack Implementation with Linked List

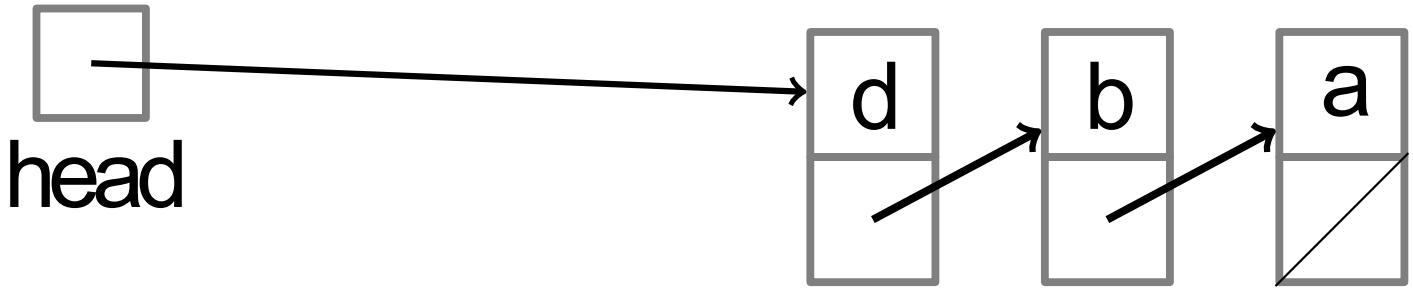


`Pop()` → `e`

Stack Implementation with Linked List

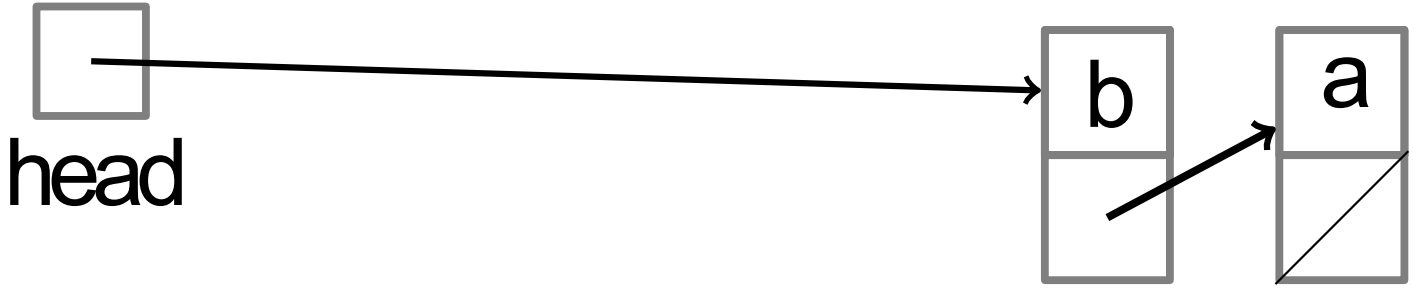


Stack Implementation with Linked List



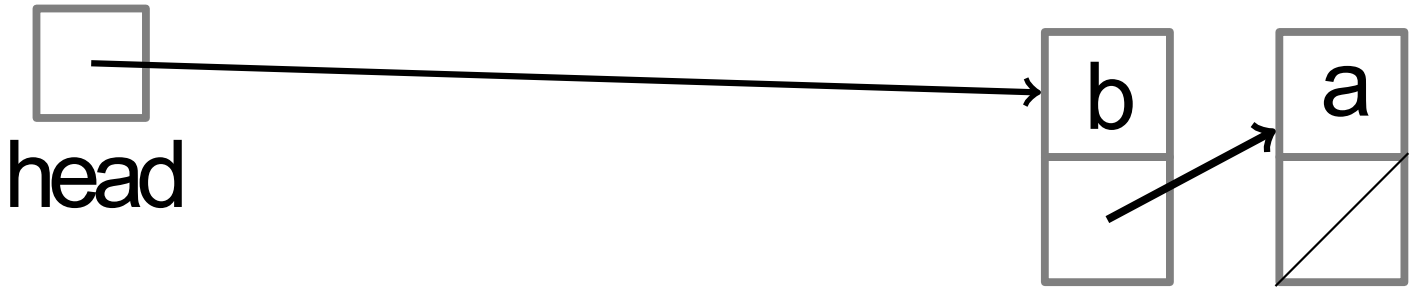
Pop()

Stack Implementation with Linked List

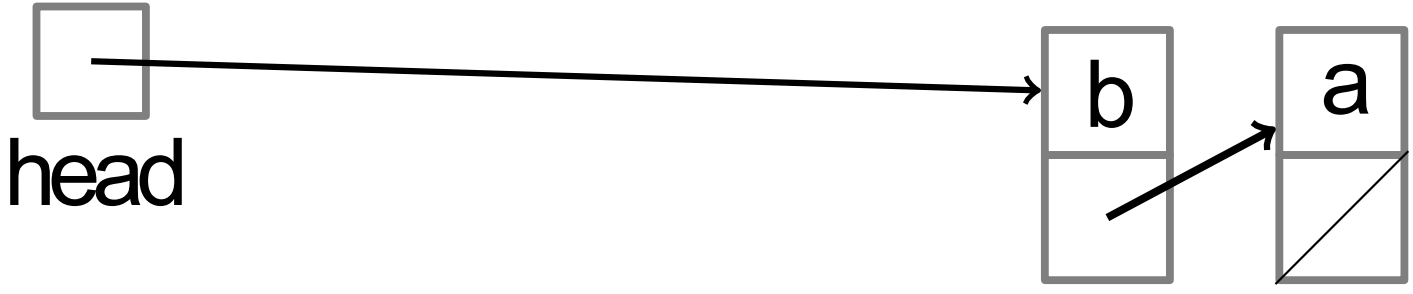


`Pop()` → `d`

Stack Implementation with Linked List

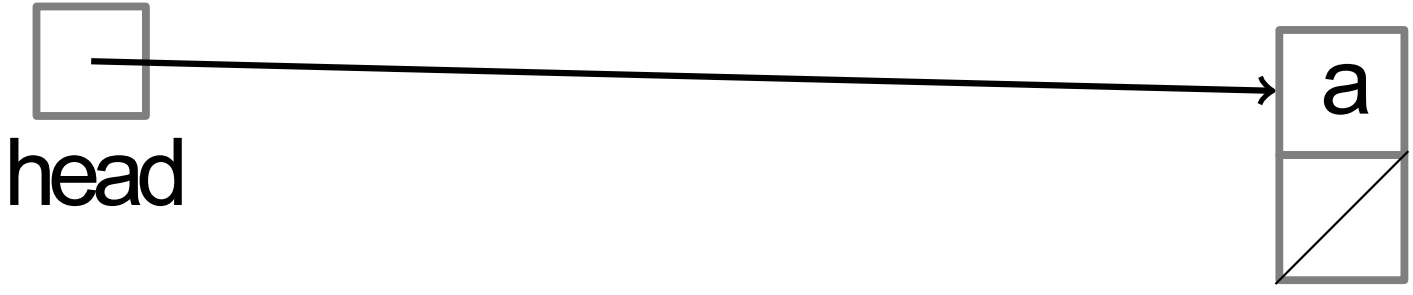


Stack Implementation with Linked List



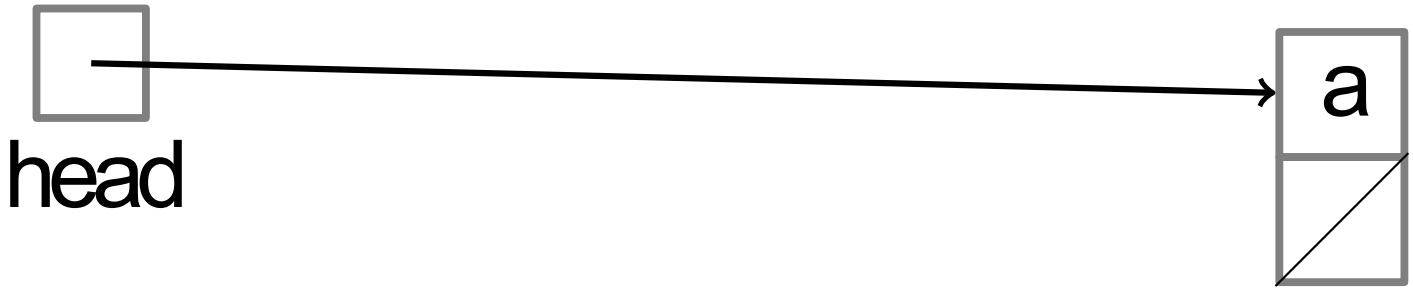
Pop()

Stack Implementation with Linked List

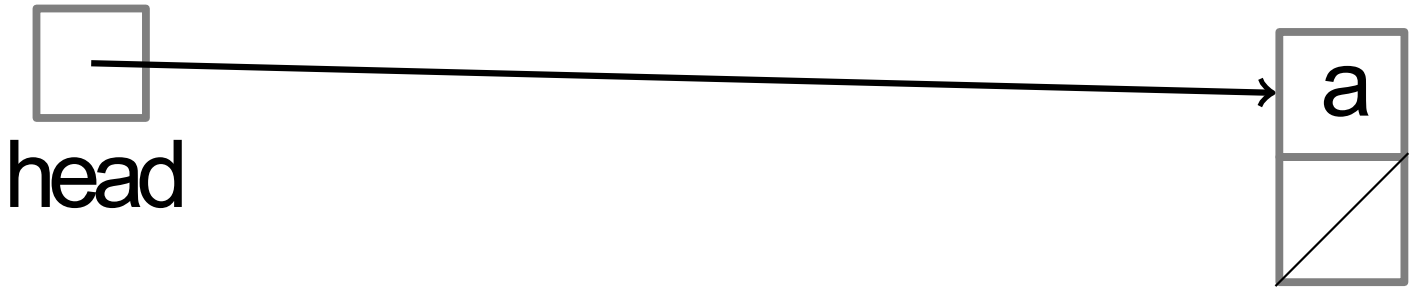


`Pop()` → `b`

Stack Implementation with Linked List

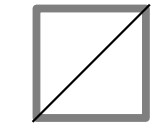


Stack Implementation with Linked List



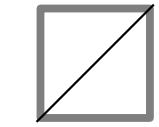
Pop ()

Stack Implementation with Linked List



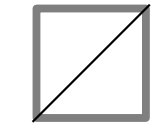
`Pop()` → `a`

Stack Implementation with Linked List



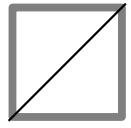
head

Stack Implementation with Linked List



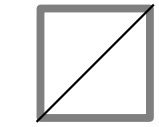
`Empty()`

Stack Implementation with Linked List



`Empty()` → `True`

Stack Implementation with Linked List



head

Stack Implementation with Linked List

Stack Implementation with Linked List

Stack Implementation with Linked List

Summary

- Stacks can be implemented with either an array or a linked list.

Summary

- Stacks can be implemented with either an array or a linked list.
- Each stack operation is $O(1)$: Push, Pop, Top, Empty.

Summary

- Stacks can be implemented with either an array or a linked list.
- Each stack operation is $O(1)$: Push, Pop, Top, Empty.
- Stacks are ocassionaly known as LIFO queues.

Outline

Definition

Queue: Abstract data type with the following operations:

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection
- `Key Dequeue()`: removes and returns least recently-added key

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection
- `Key Dequeue()`: removes and returns least recently-added key
- `Boolean Empty()`: are there any elements?

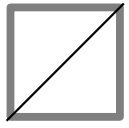
Definition

Queue: Abstract data type with the following operations:

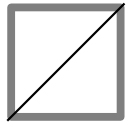
- `Enqueue(Key)`: adds key to collection
- `Key Dequeue()`: removes and returns least recently-added key
- `Boolean Empty()`: are there any elements?

FIFO: First-In, First-Out

Queue Implementation with Linked List

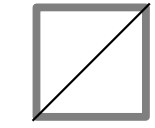


head

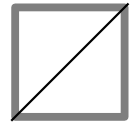


tail

Queue Implementation with Linked List



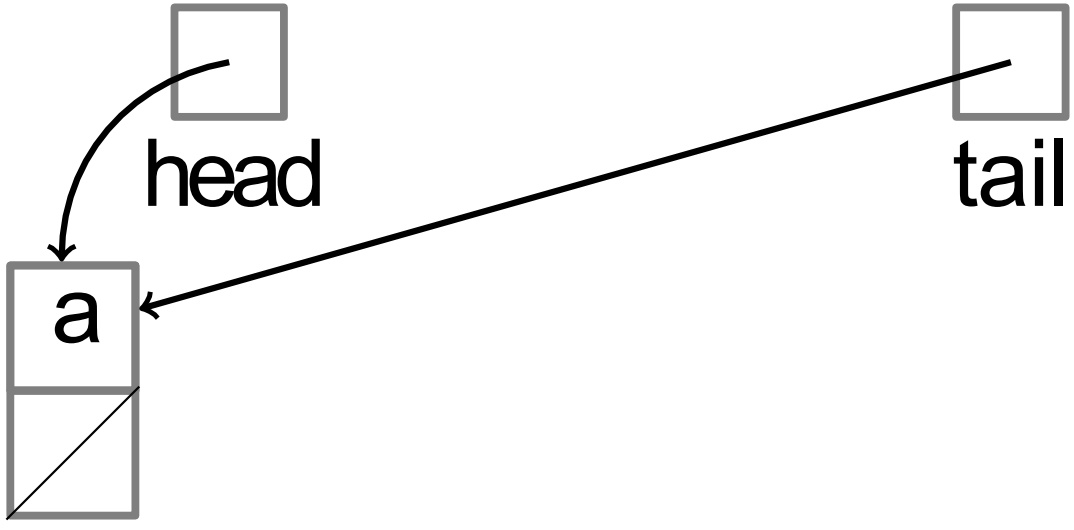
head



tail

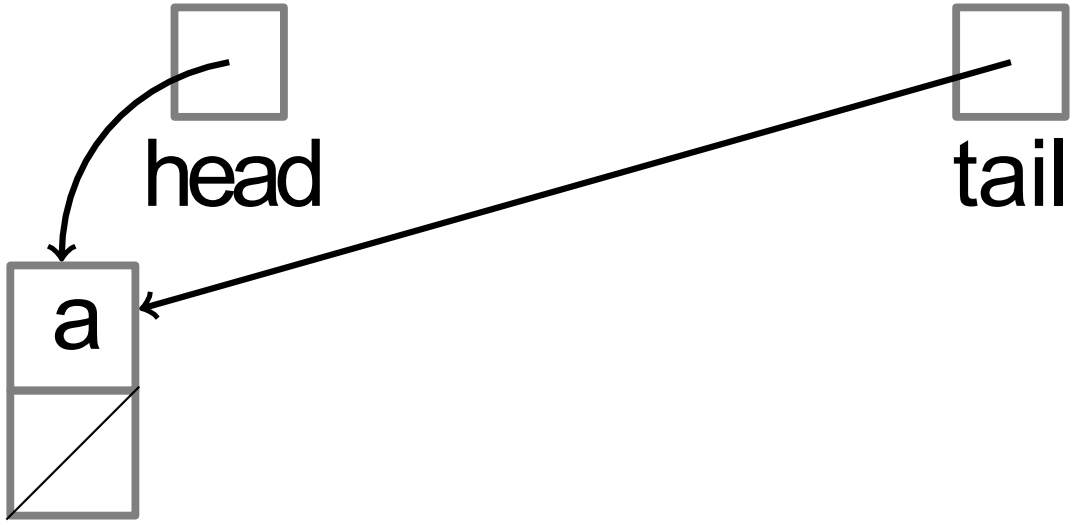
Enqueue(a)

Queue Implementation with Linked List

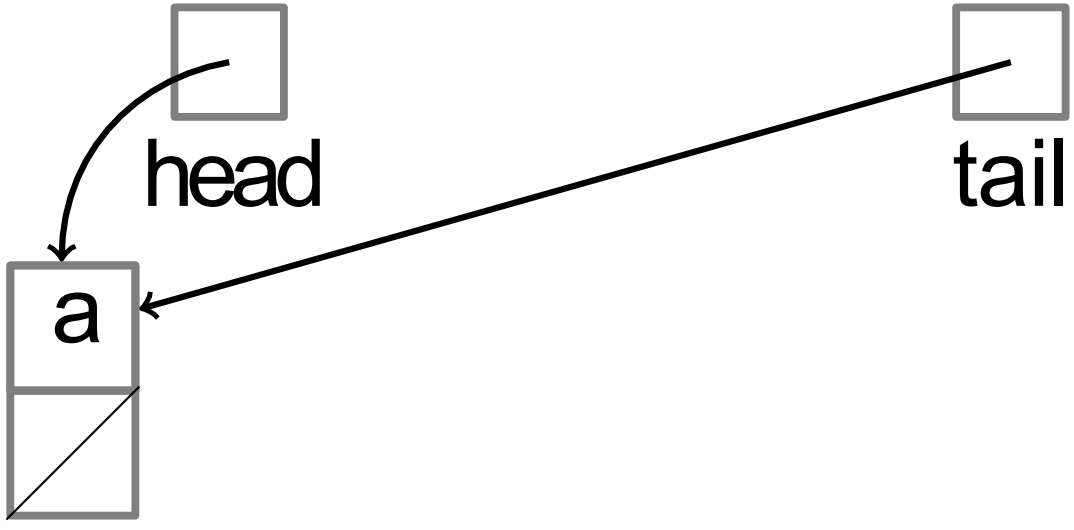


Enqueue(a)

Queue Implementation with Linked List

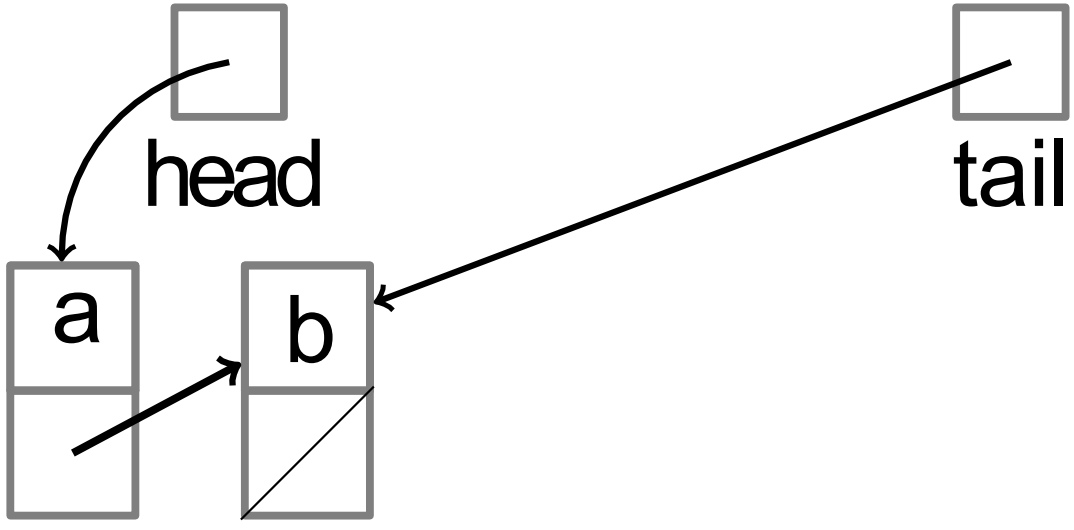


Queue Implementation with Linked List



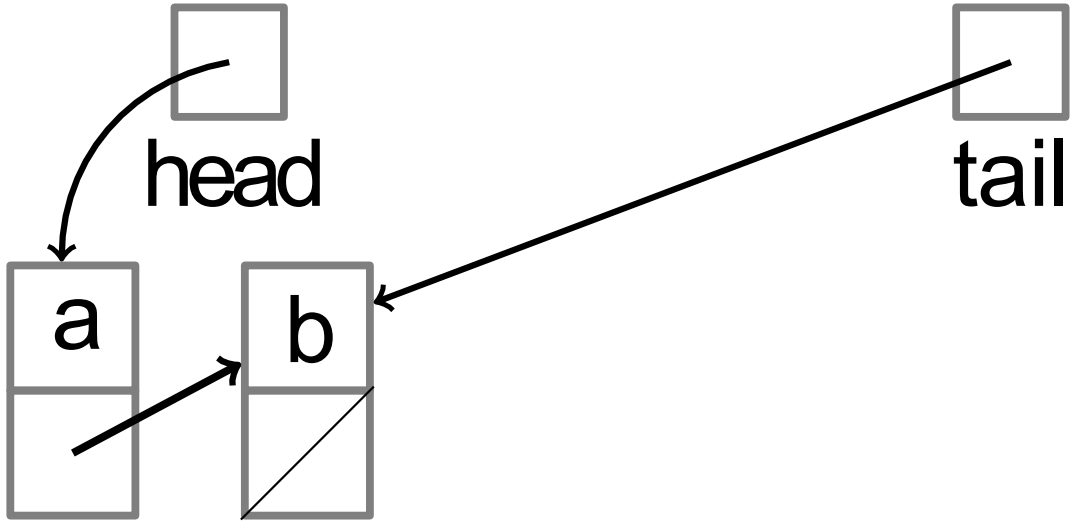
Enqueue (b)

Queue Implementation with Linked List

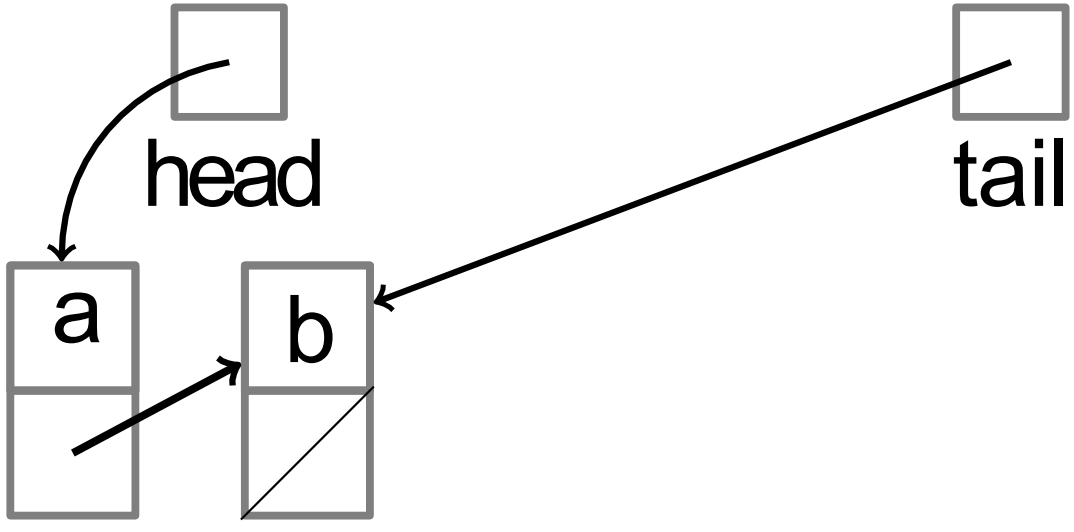


Enqueue (b)

Queue Implementation with Linked List

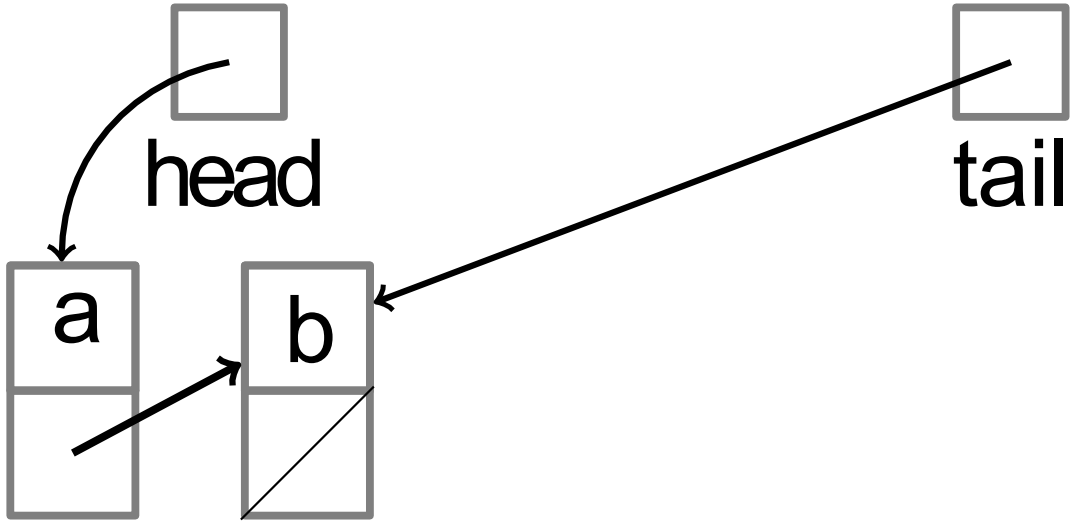


Queue Implementation with Linked List



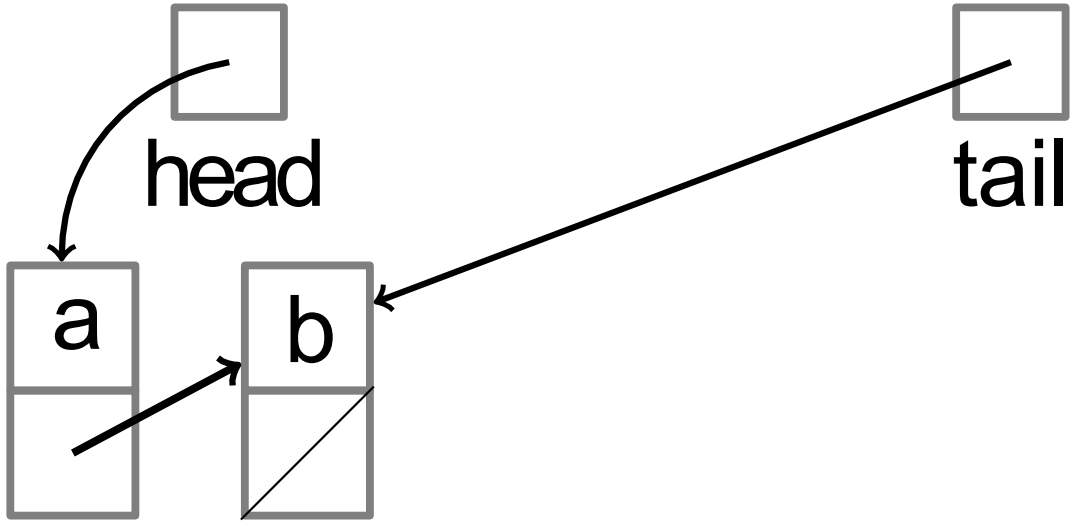
`Empty()`

Queue Implementation with Linked List

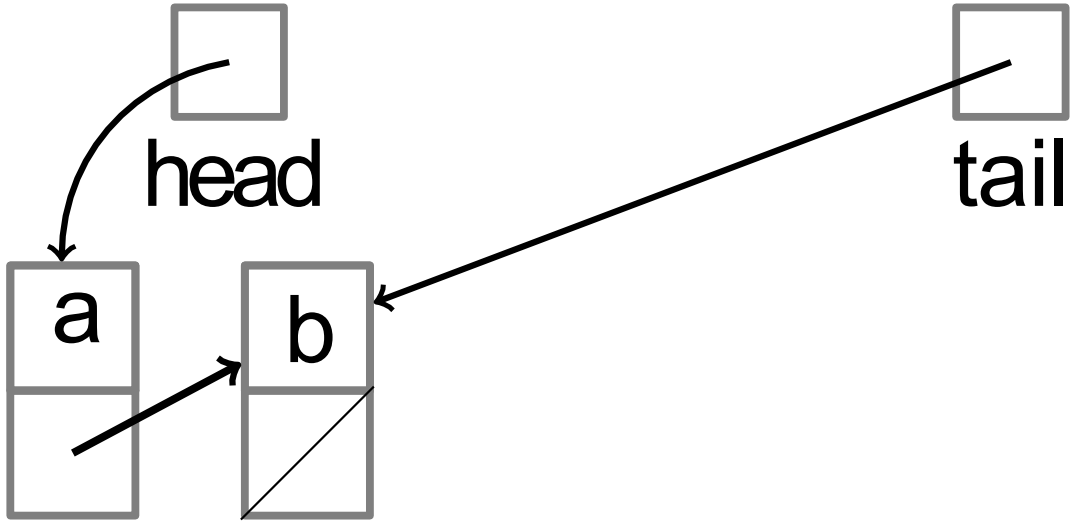


`Empty()` → `False`

Queue Implementation with Linked List

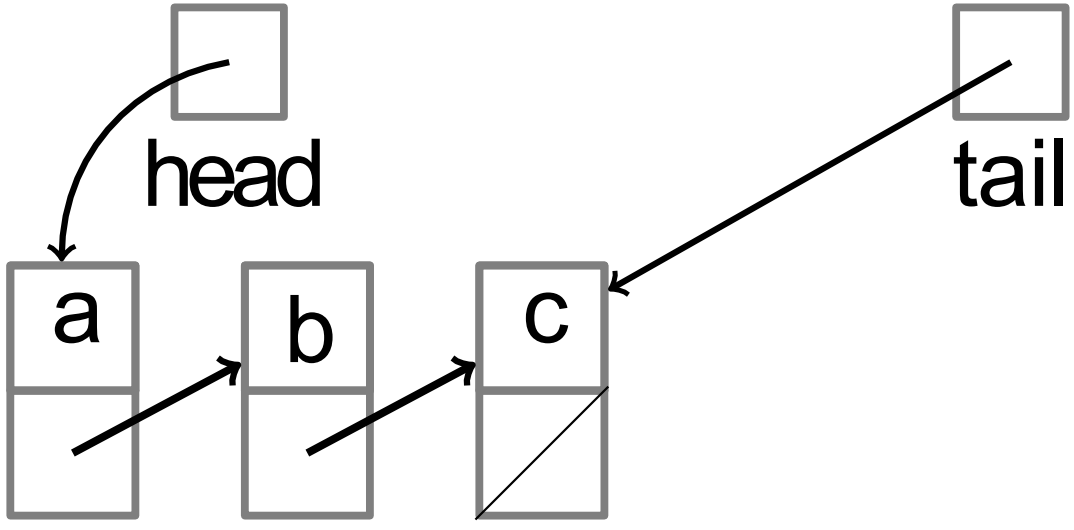


Queue Implementation with Linked List



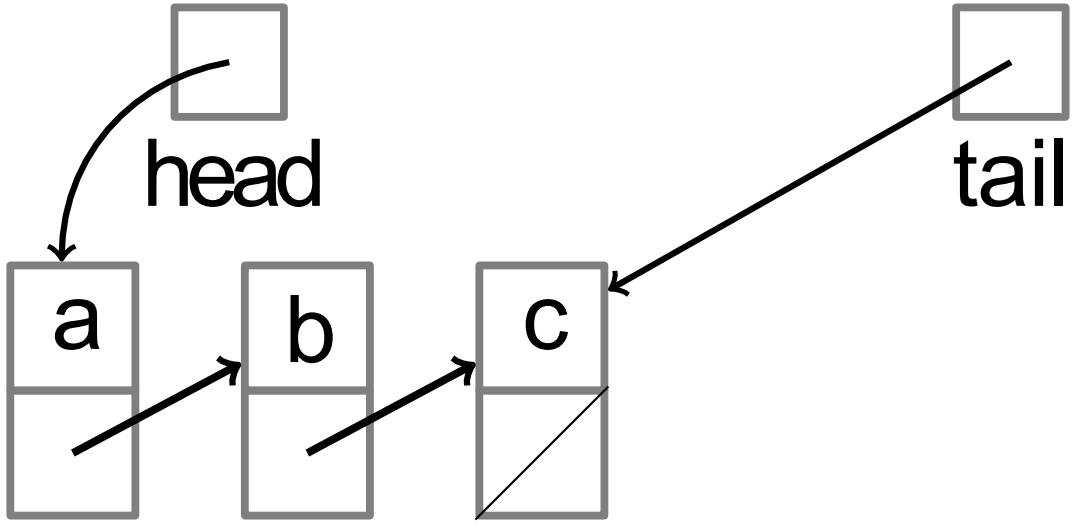
Enqueue(c)

Queue Implementation with Linked List

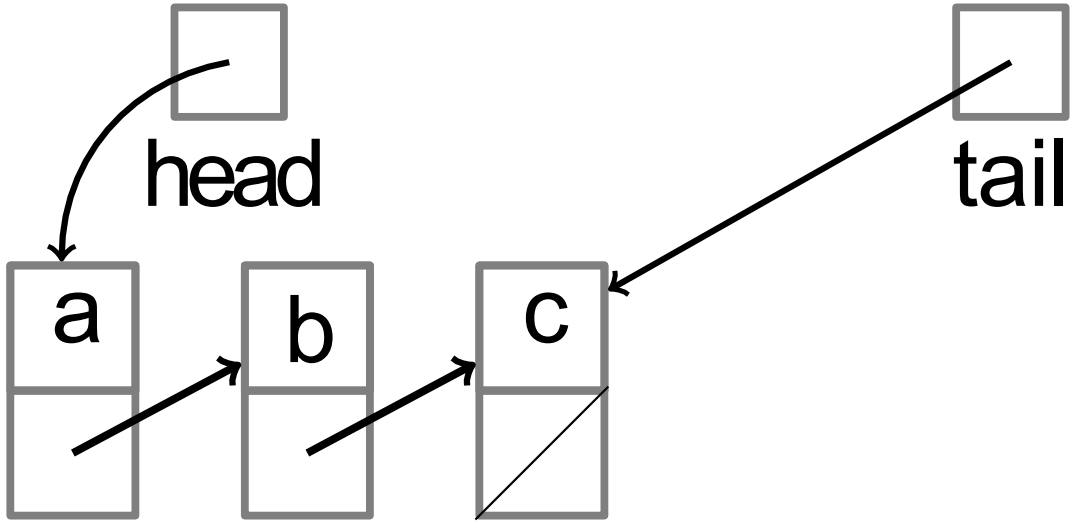


Enqueue(c)

Queue Implementation with Linked List

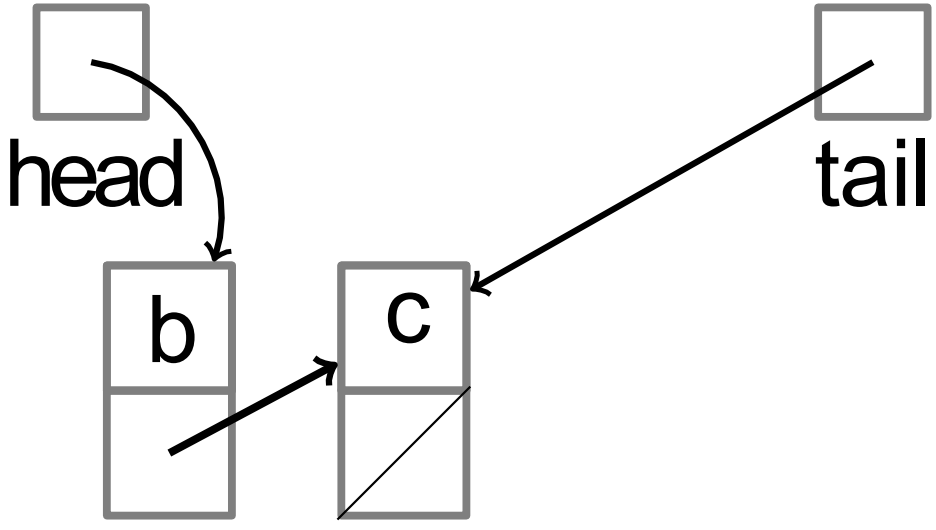


Queue Implementation with Linked List



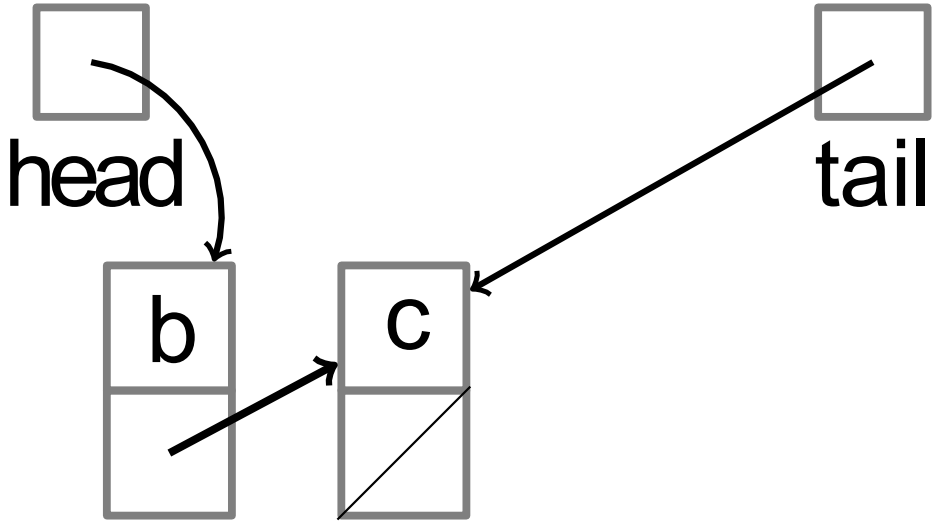
Dequeue()

Queue Implementation with Linked List

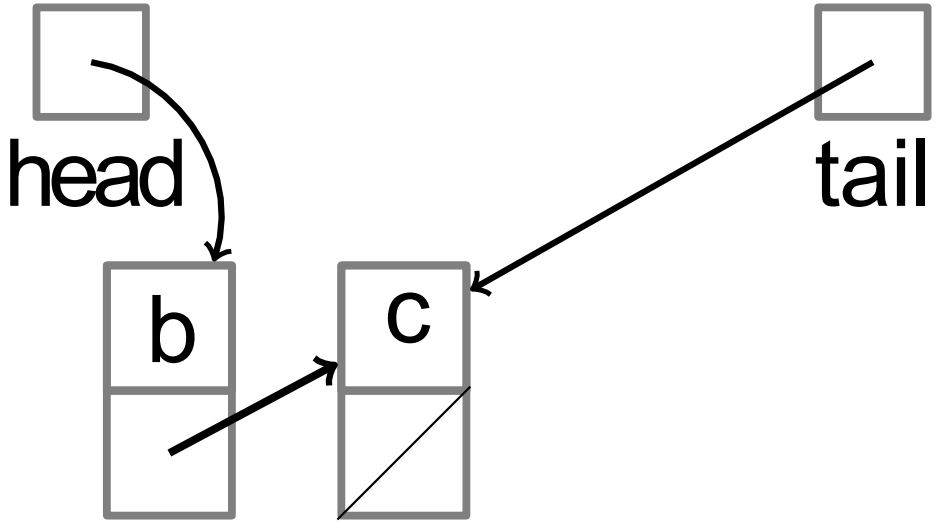


`Dequeue()` → `a`

Queue Implementation with Linked List

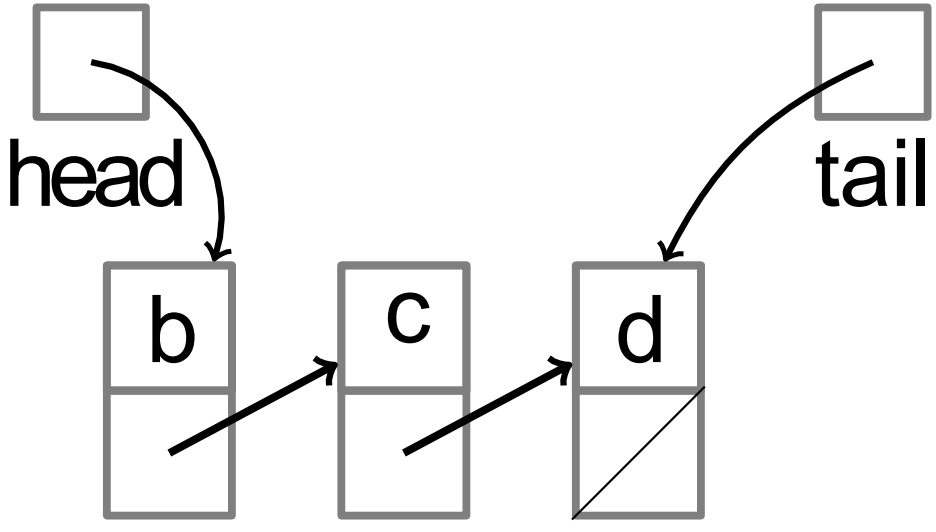


Queue Implementation with Linked List



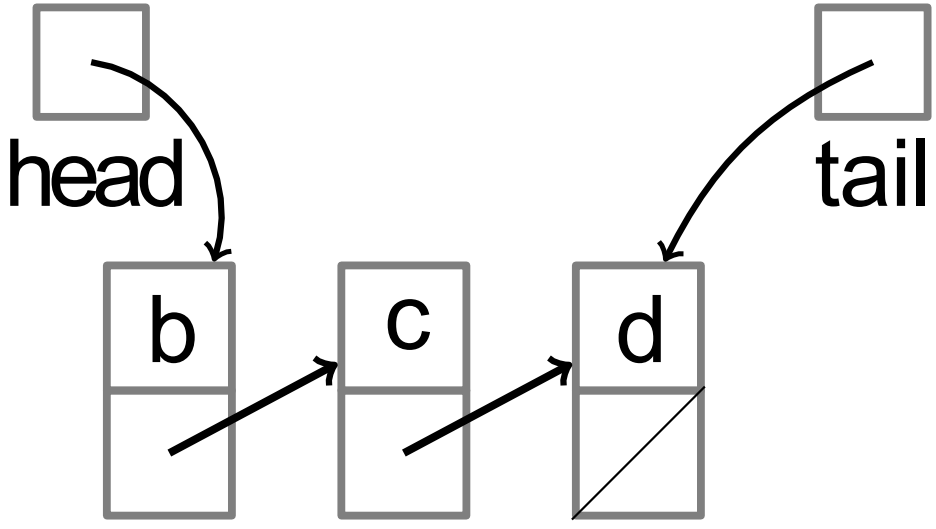
Enqueue (d)

Queue Implementation with Linked List

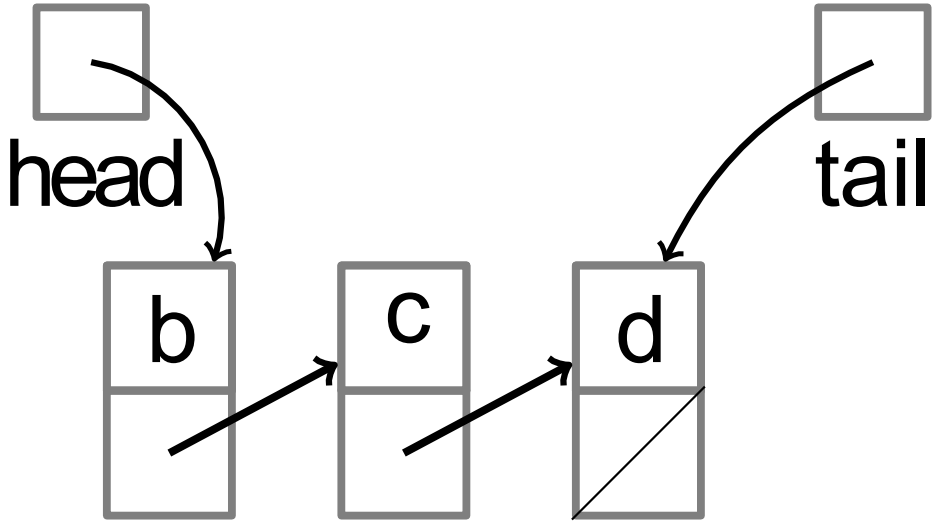


Enqueue (d)

Queue Implementation with Linked List

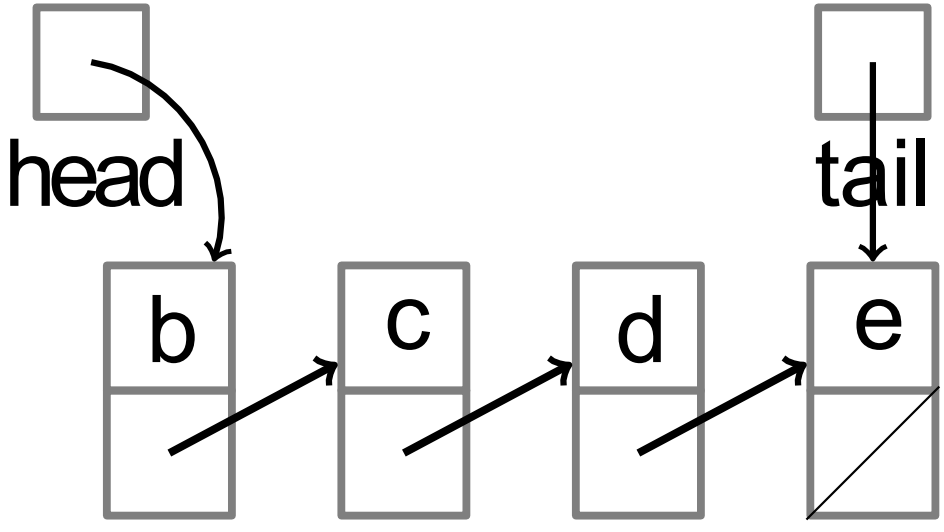


Queue Implementation with Linked List



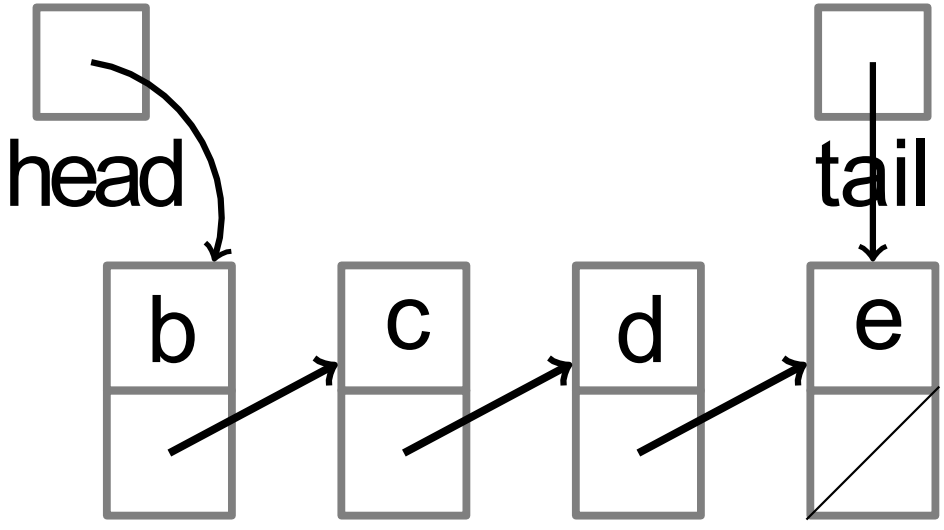
Enqueue(e)

Queue Implementation with Linked List

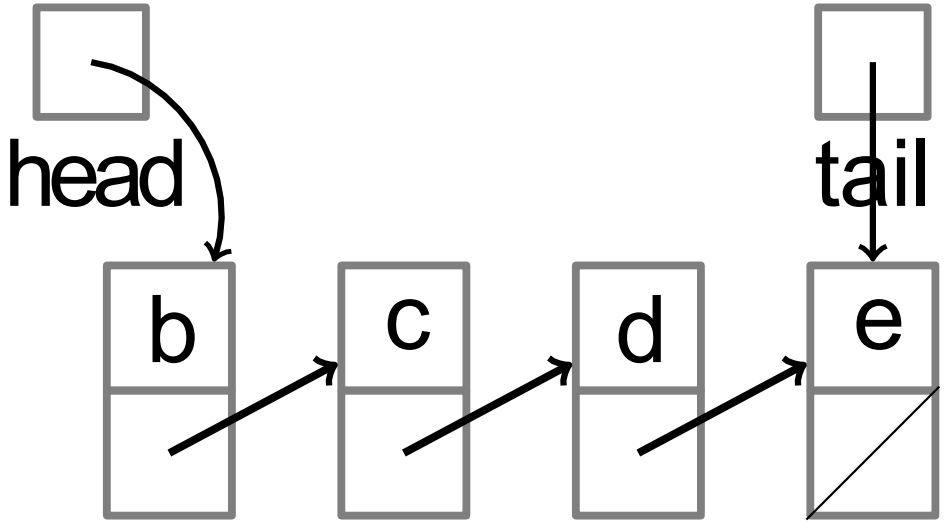


Enqueue(e)

Queue Implementation with Linked List

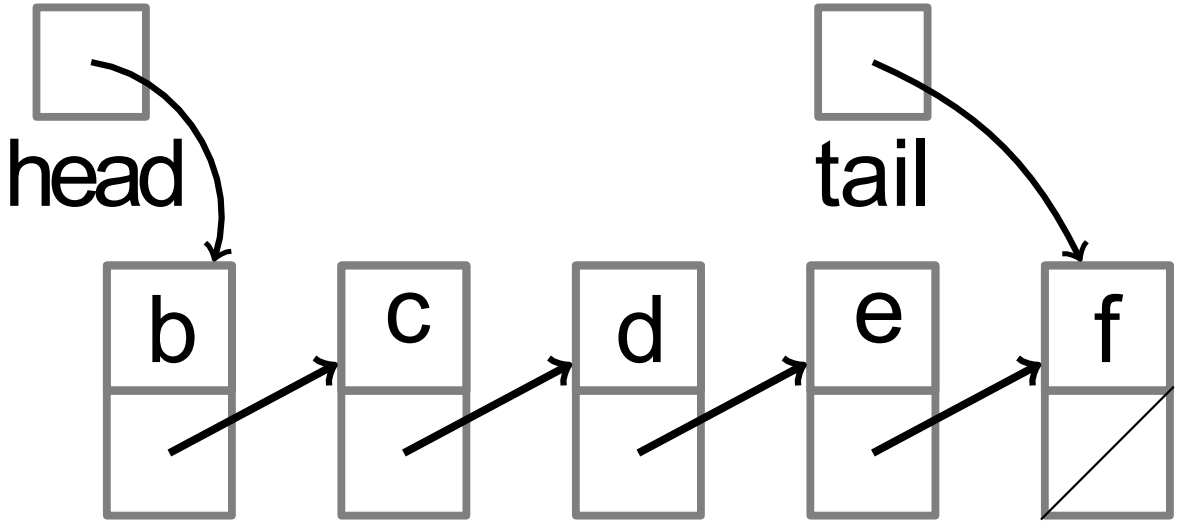


Queue Implementation with Linked List



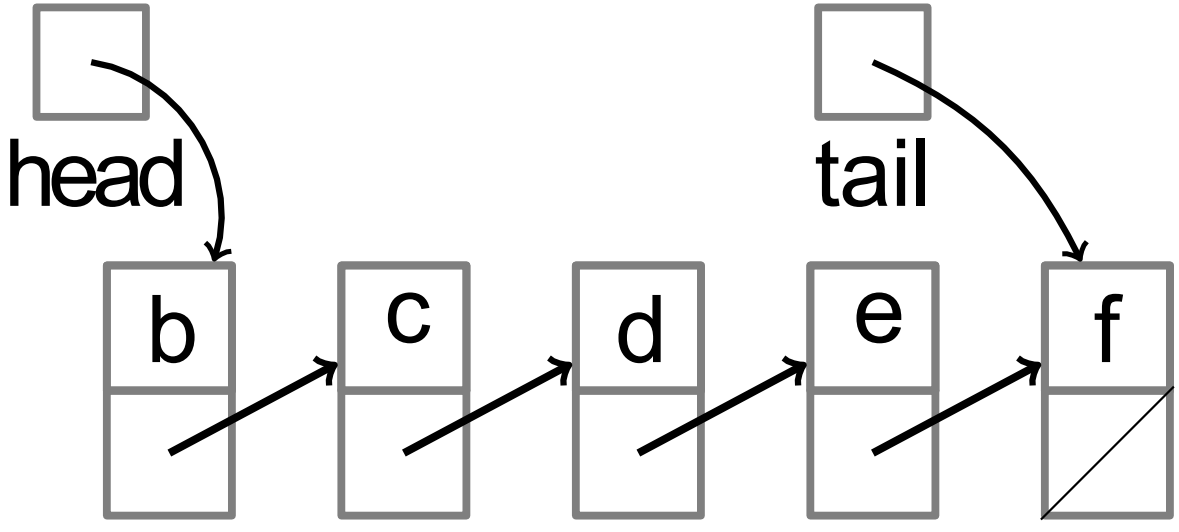
`Enqueue(f)`

Queue Implementation with Linked List

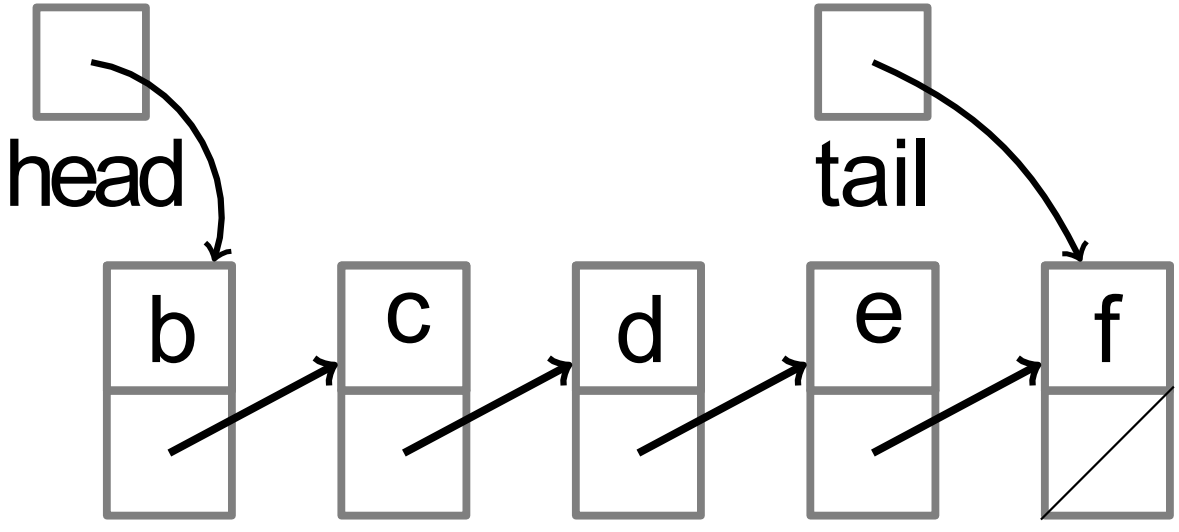


`Enqueue(f)`

Queue Implementation with Linked List

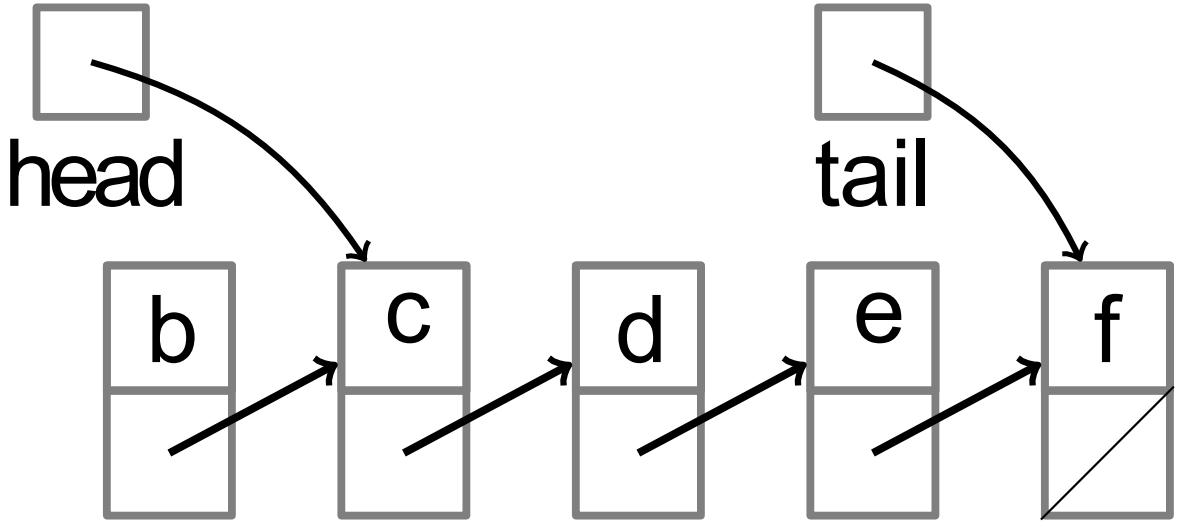


Queue Implementation with Linked List



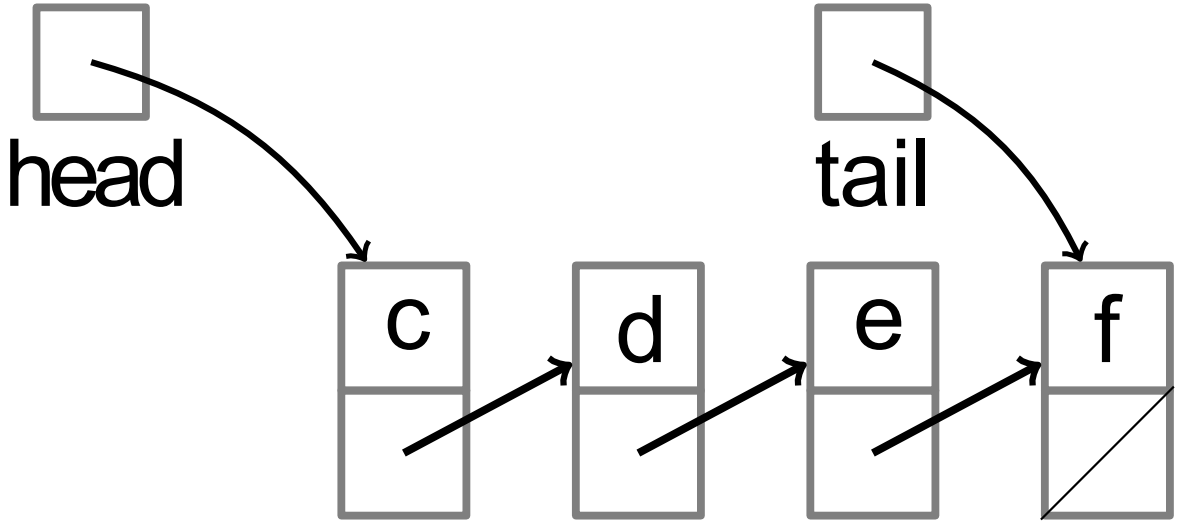
Dequeue()

Queue Implementation with Linked List

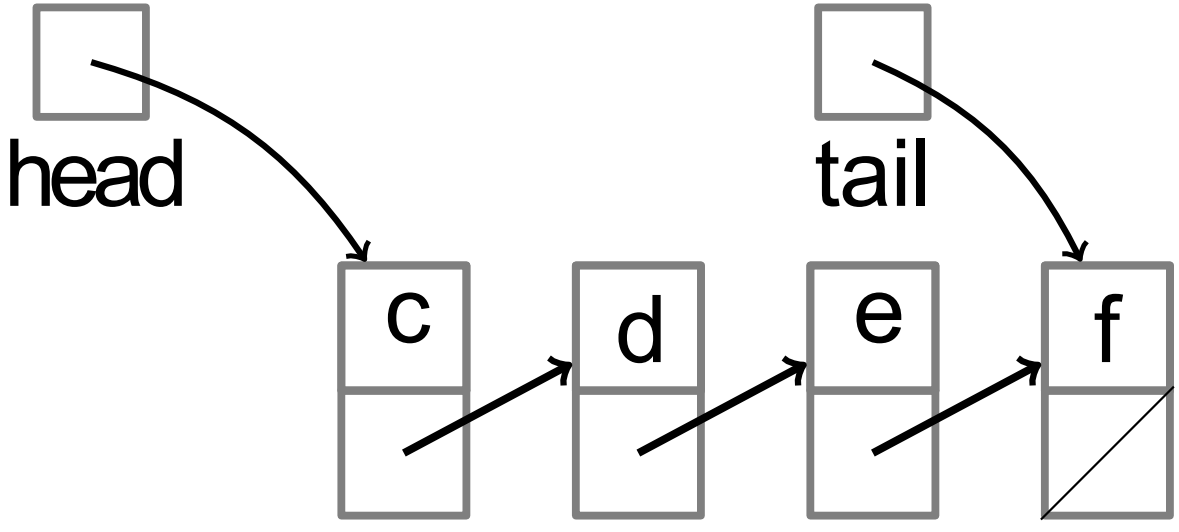


Dequeue()→ b

Queue Implementation with Linked List

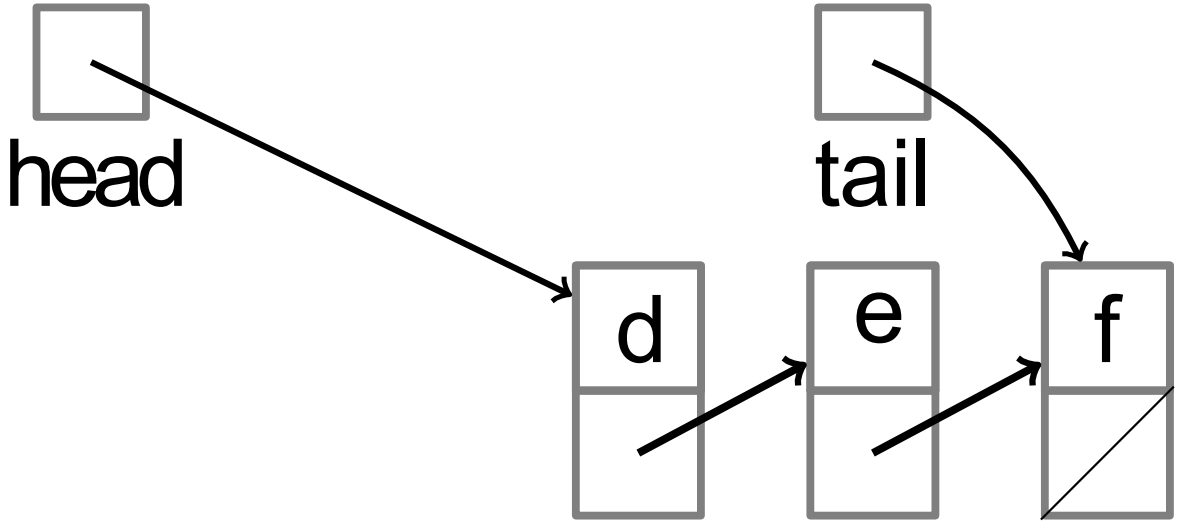


Queue Implementation with Linked List



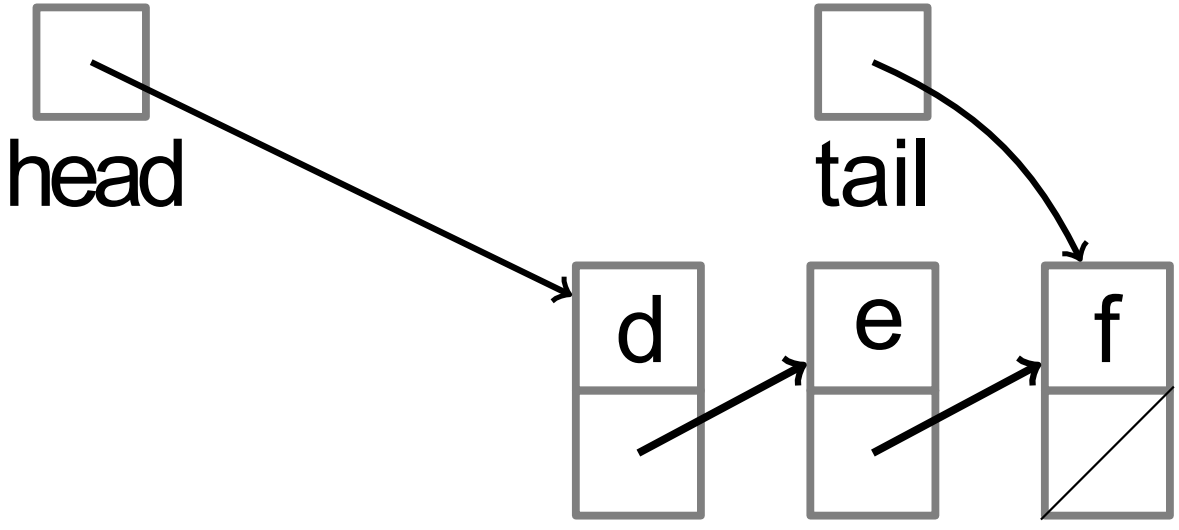
Dequeue()

Queue Implementation with Linked List

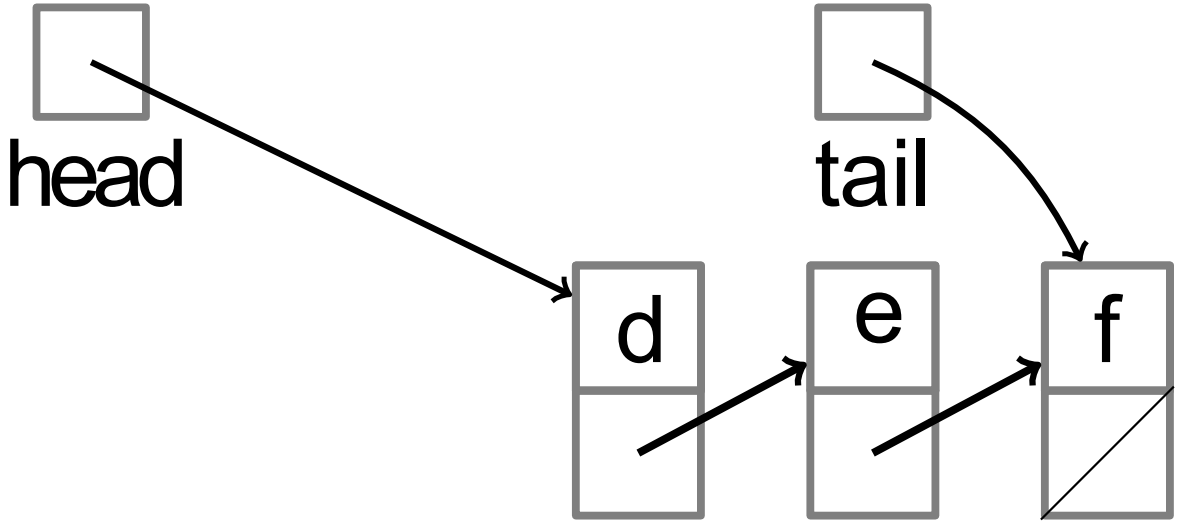


`Dequeue()` → `c`

Queue Implementation with Linked List

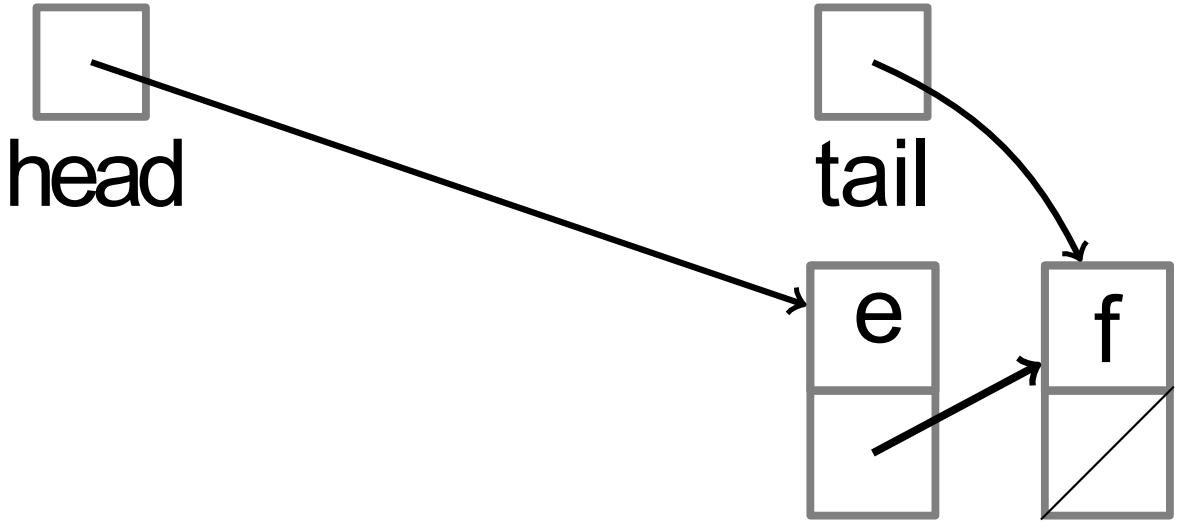


Queue Implementation with Linked List



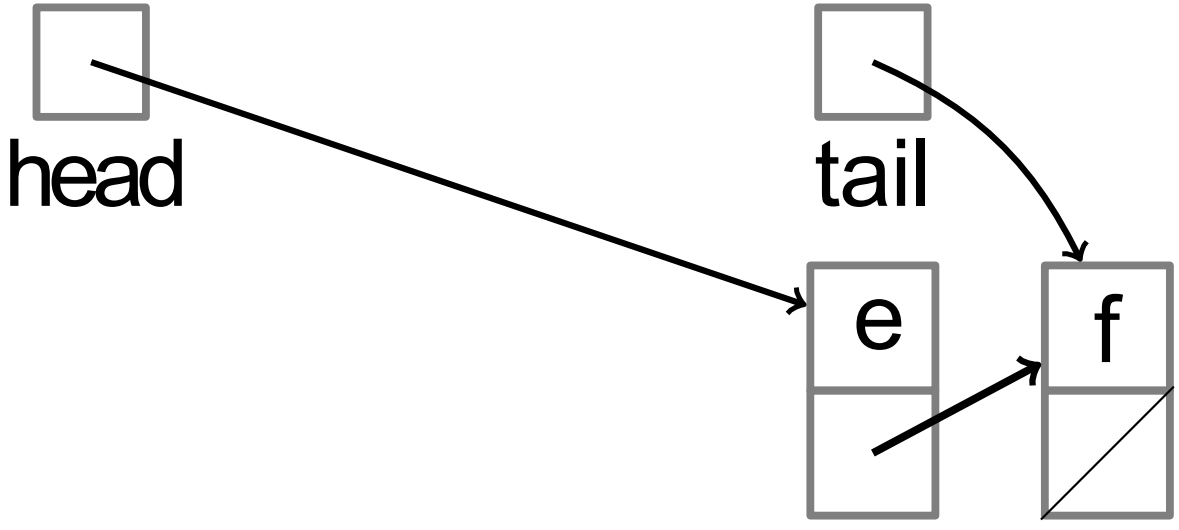
Dequeue()

Queue Implementation with Linked List

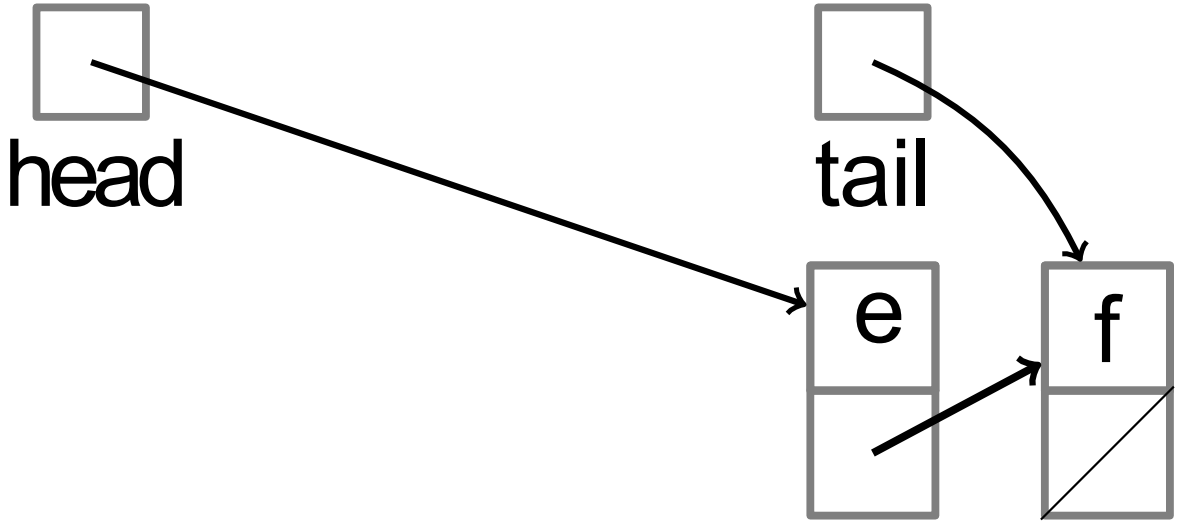


`Dequeue()` → `d`

Queue Implementation with Linked List

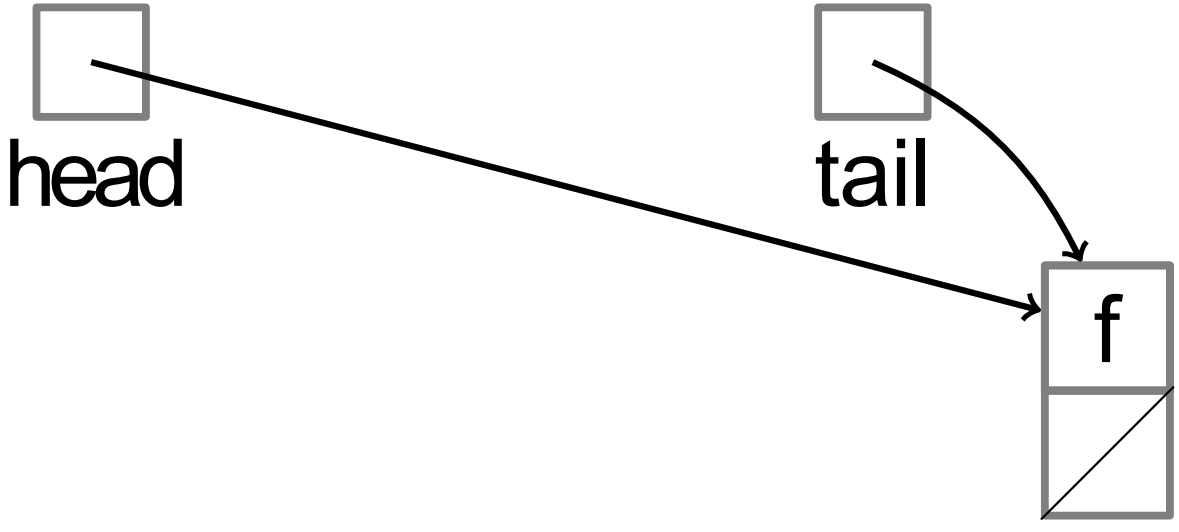


Queue Implementation with Linked List



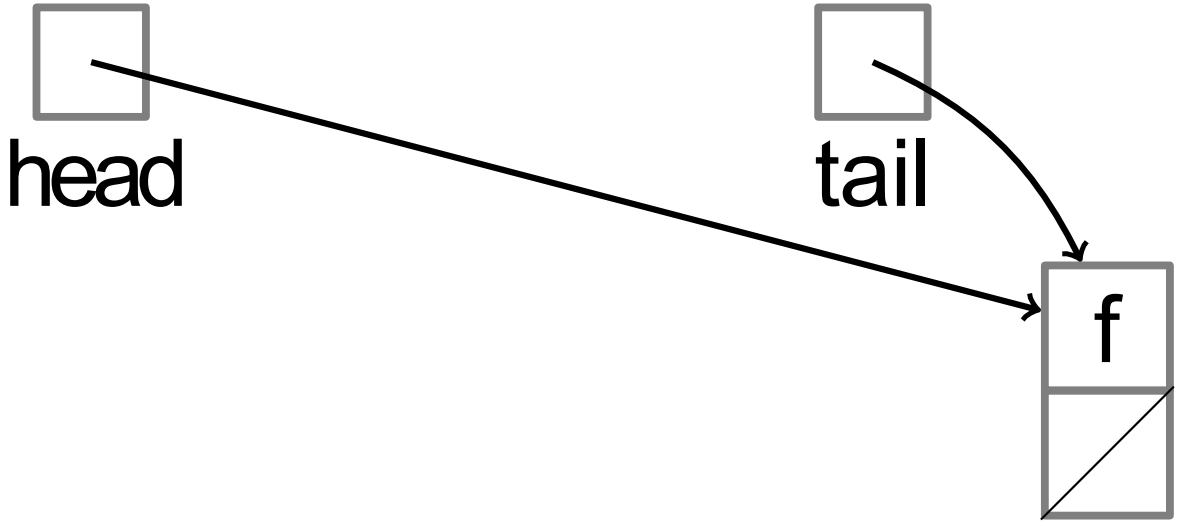
Dequeue()

Queue Implementation with Linked List

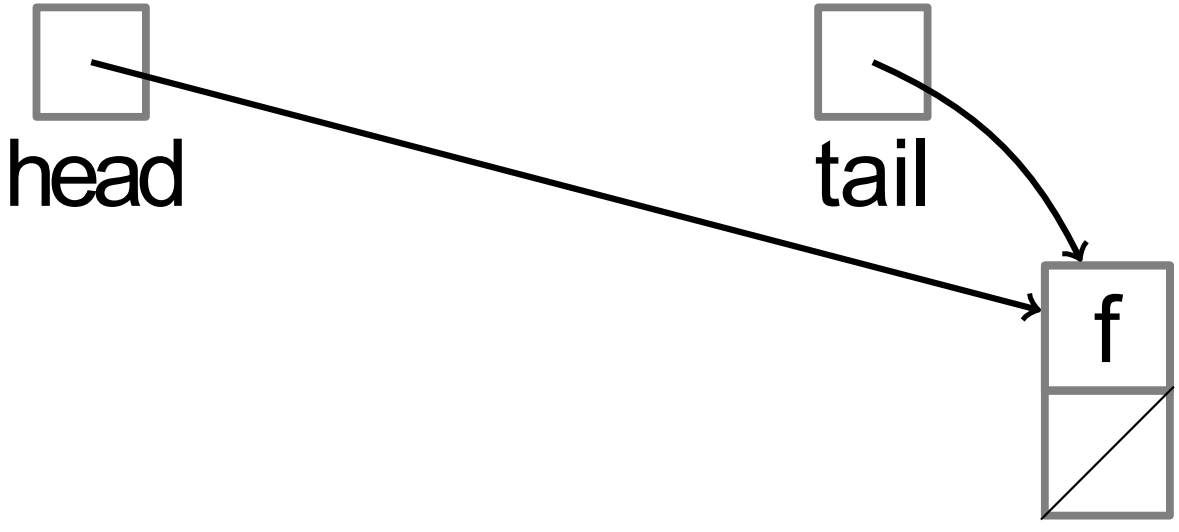


`Dequeue()` → `e`

Queue Implementation with Linked List

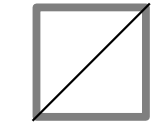


Queue Implementation with Linked List

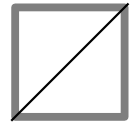


Dequeue()

Queue Implementation with Linked List



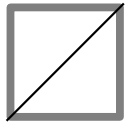
head



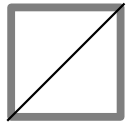
tail

`Dequeue()→ f`

Queue Implementation with Linked List

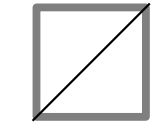


head

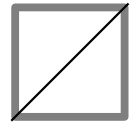


tail

Queue Implementation with Linked List



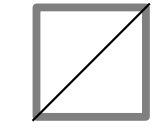
head



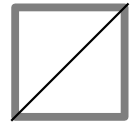
tail

`Empty()`

Queue Implementation with Linked List



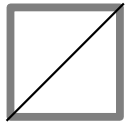
head



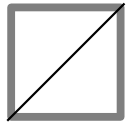
tail

`Empty()` → `True`

Queue Implementation with Linked List



head



tail

Queue Implementation with Linked List

- `Enqueue`: use `List.PushBack`

Queue Implementation with Linked List

- `Enqueue`: use `List.PushBack`
- `Dequeue`: use `List.TopFront` and `List.PopFront`

Queue Implementation with Linked List

- `Enqueue`: use `List.PushBack`
- `Dequeue`: use `List.TopFront` and `List.PopFront`
- `Empty`: use `List.Empty`

Question

What would happen if we used List.PushFront to implement Enqueue and List.TopBack and List.PopBack to implement Dequeue?

a)The queue would work correctly, but Dequeue would be O(n) time if the list were singly-linked with a tail pointer.

b)The queue would work correctly if the list were singly-linked with a tail pointer.

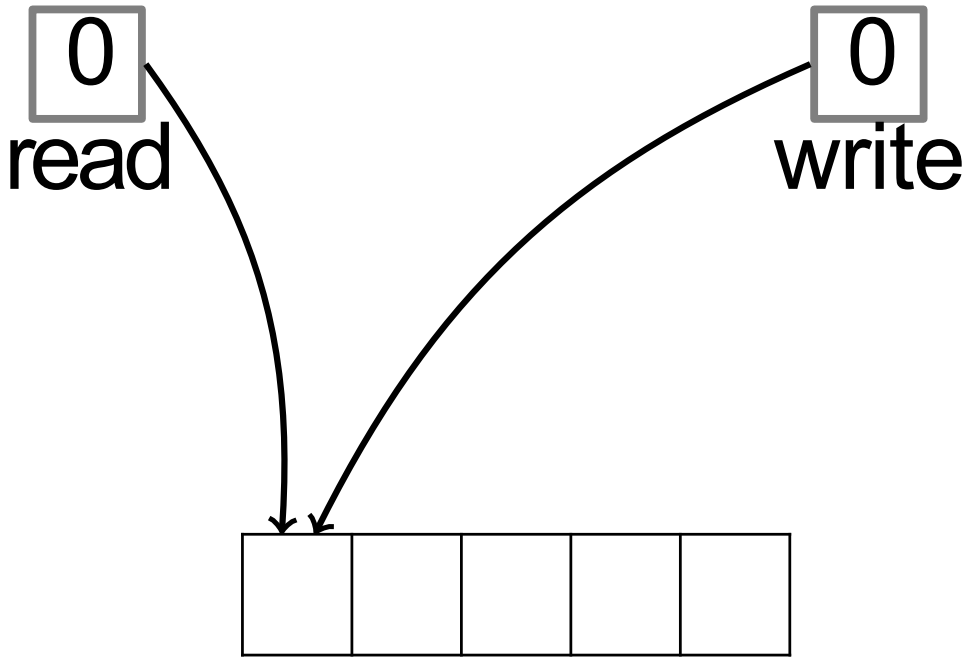
c)The queue would work correctly.

d)The queue wouldn't work right

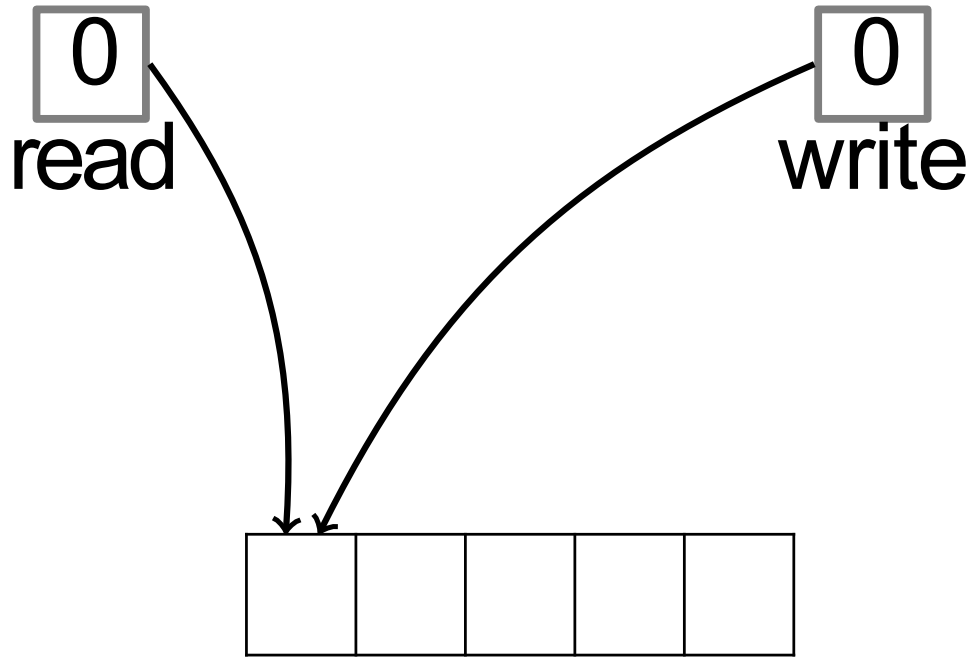
e)The queue would work correctly if the list were doubly-linked with a tail pointer.

f)The queue would work correctly, but Dequeue would be O(n) time if the list were doubly-linked with a tail pointer.

Queue Implementation with Array

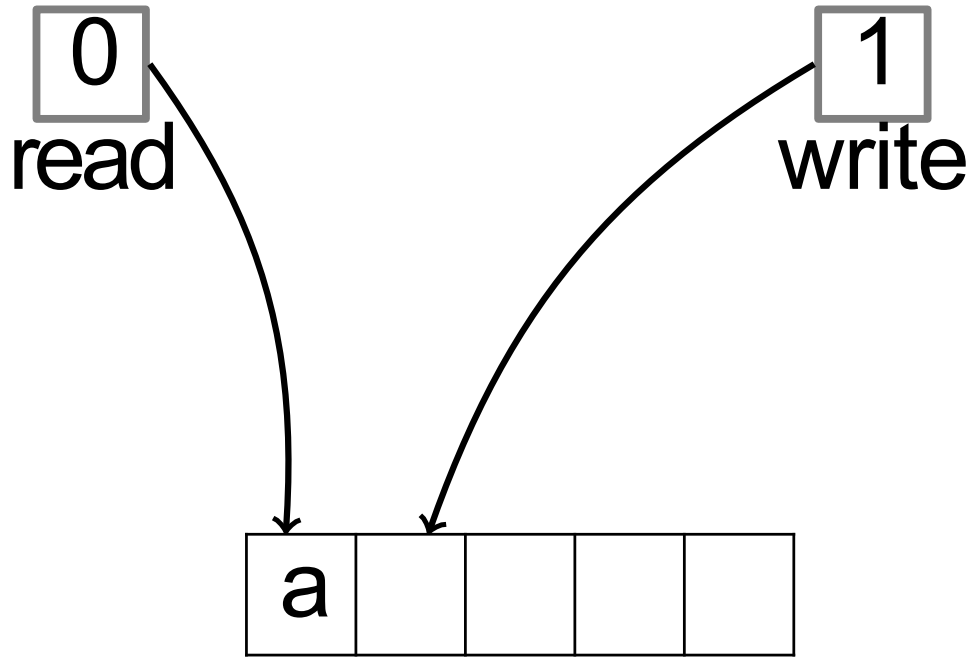


Queue Implementation with Array



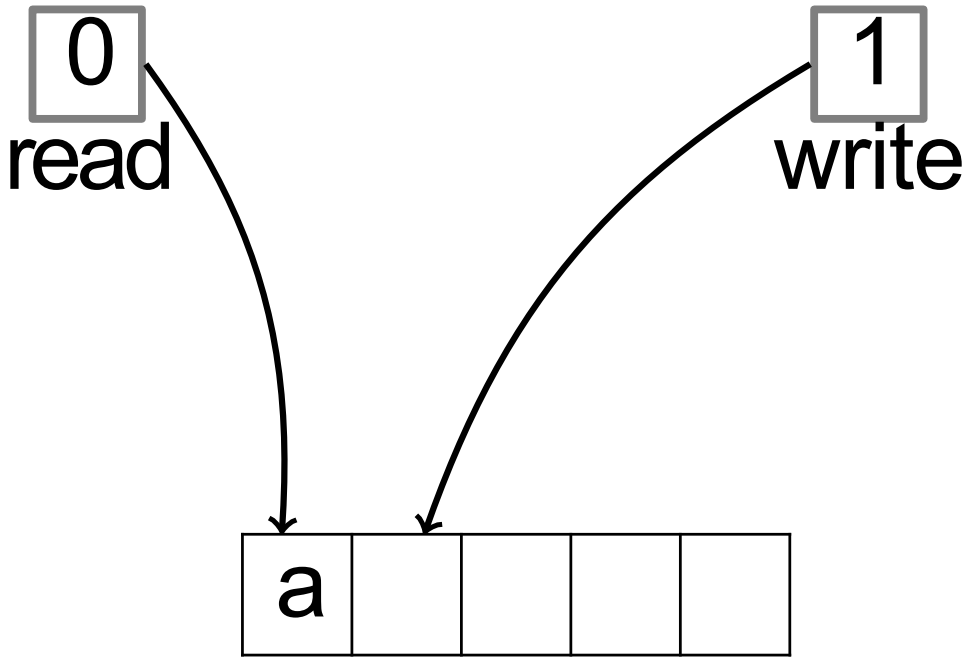
Enqueue(a)

Queue Implementation with Array

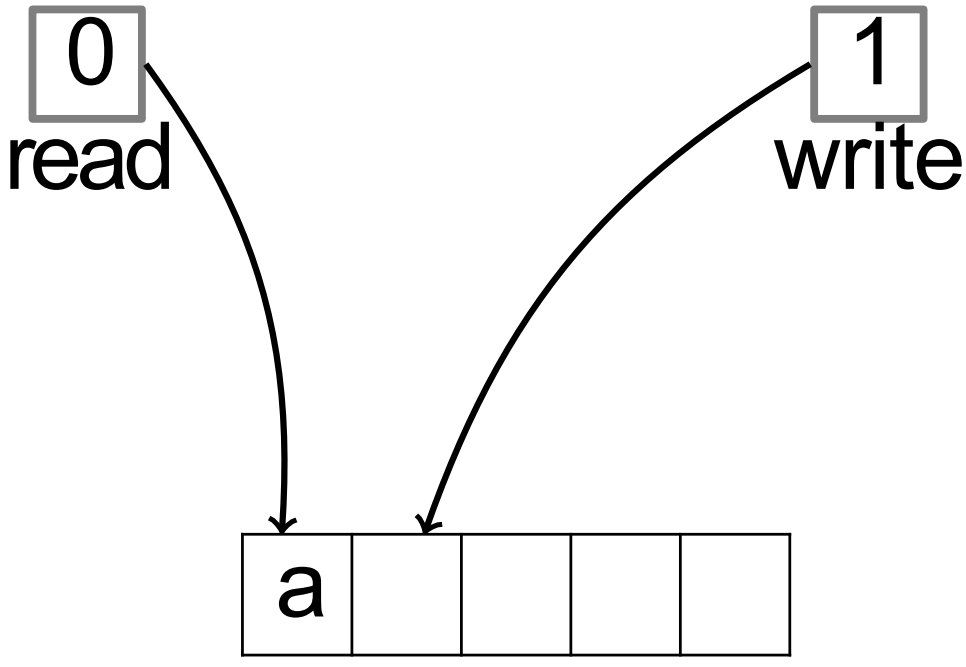


Enqueue(a)

Queue Implementation with Array

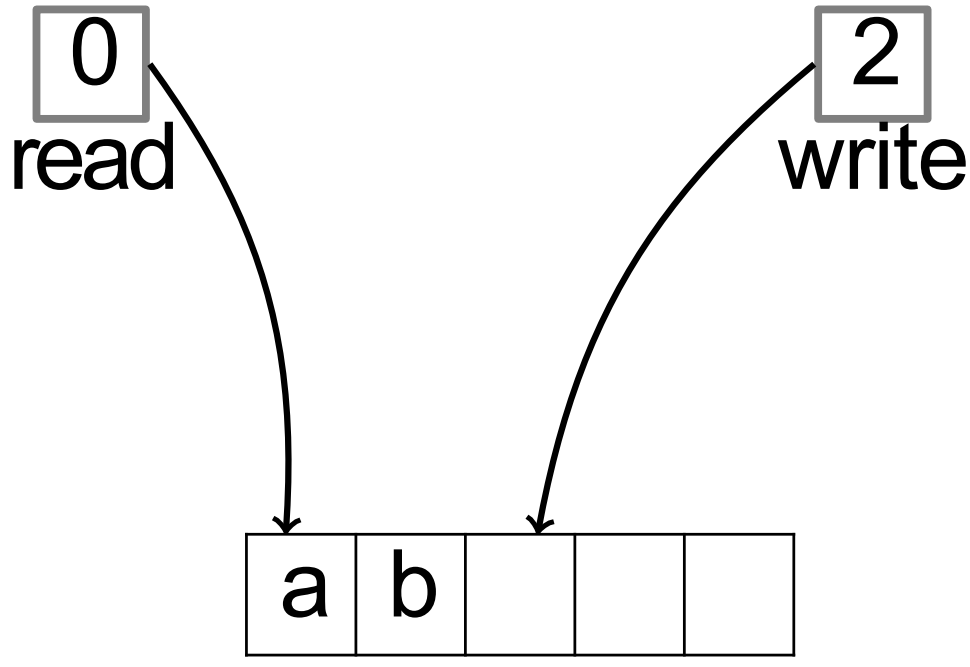


Queue Implementation with Array



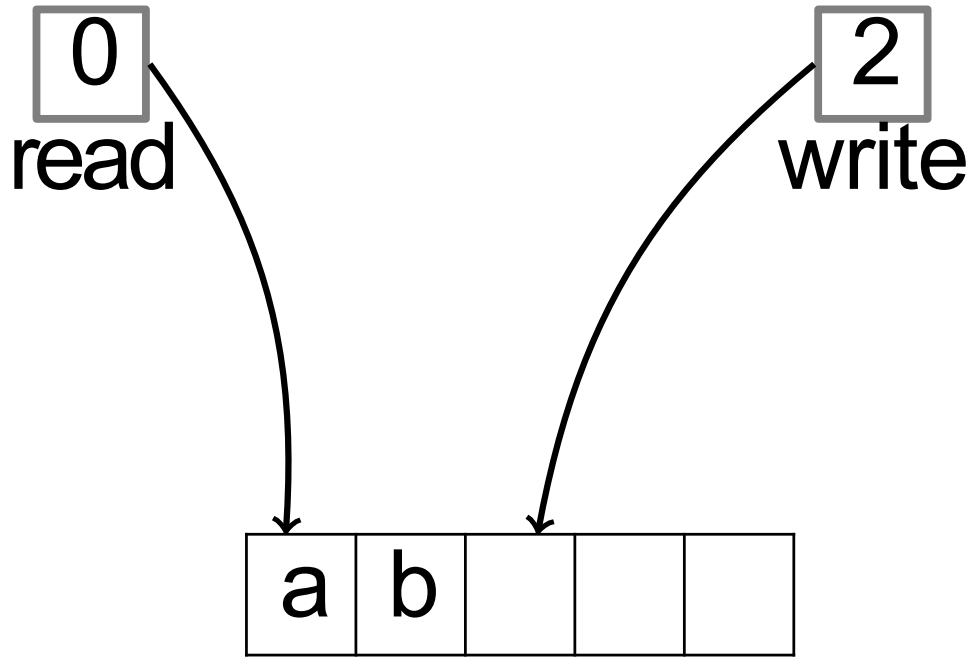
Enqueue (b)

Queue Implementation with Array

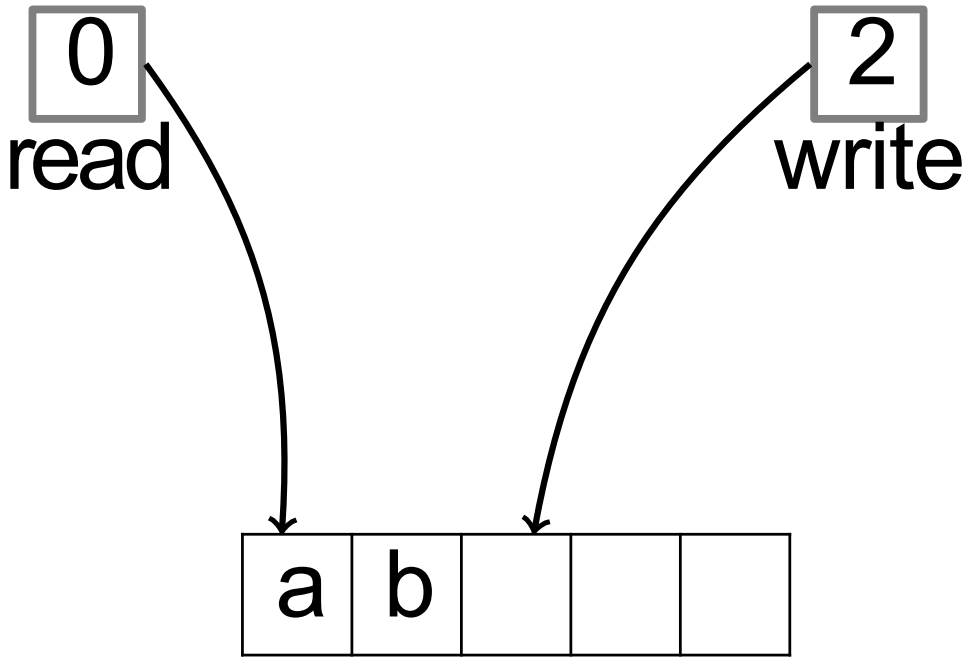


Enqueue (b)

Queue Implementation with Array

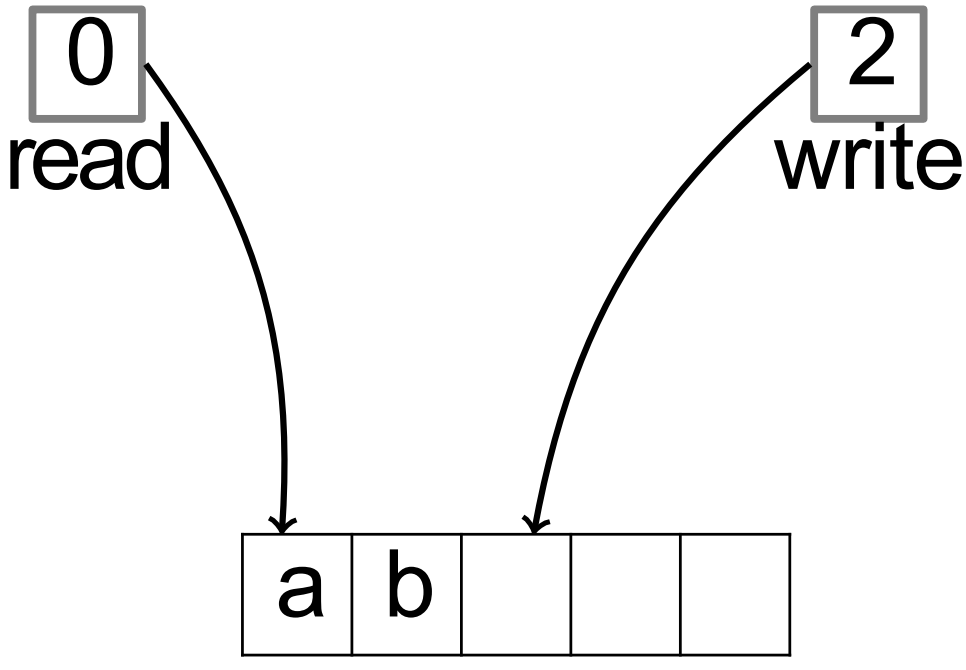


Queue Implementation with Array



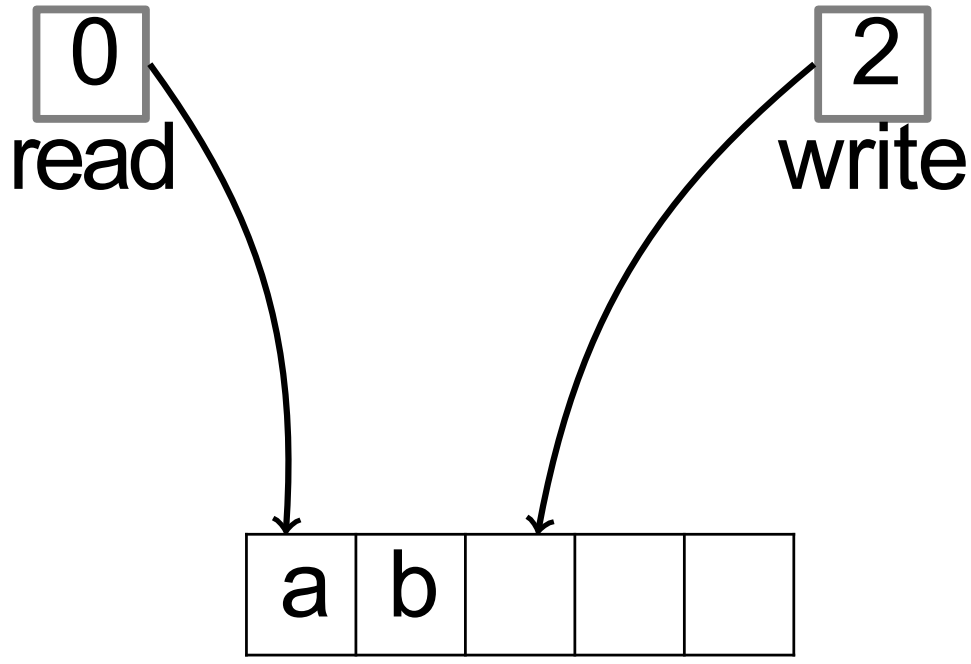
`Empty()`

Queue Implementation with Array

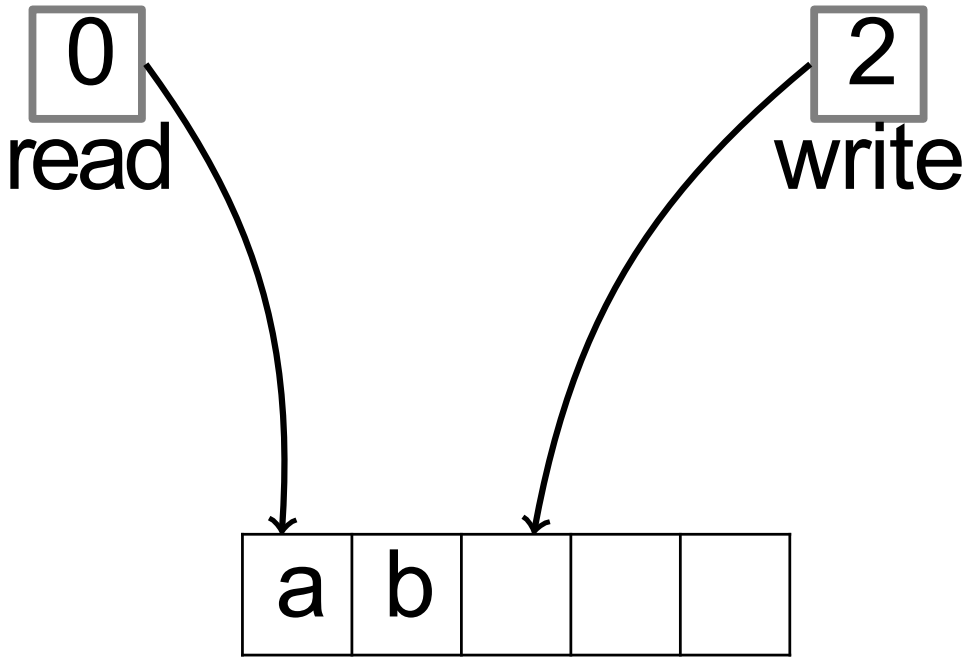


`Empty()` → `False`

Queue Implementation with Array

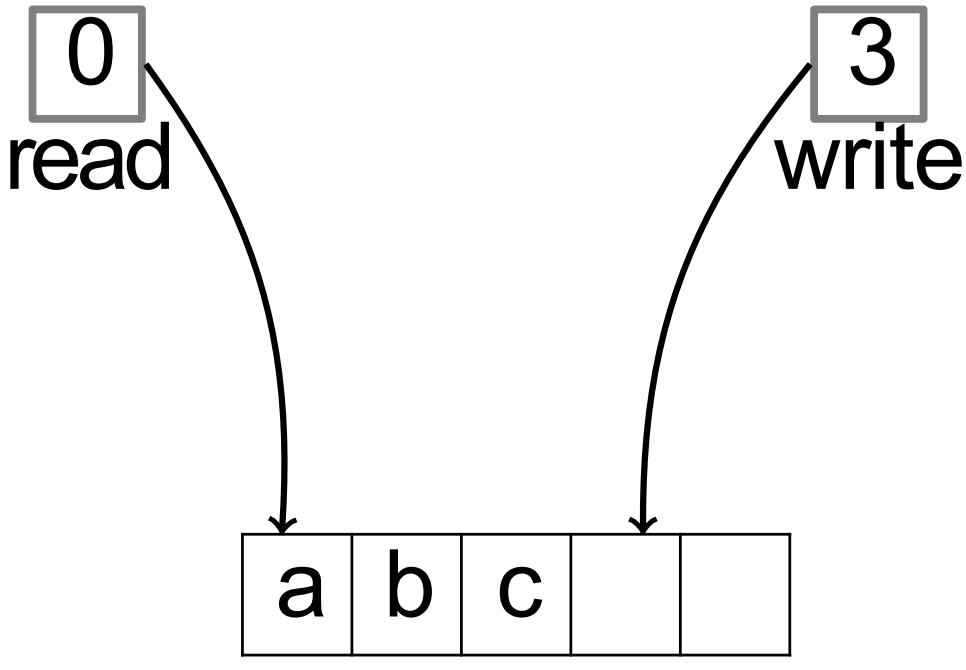


Queue Implementation with Array



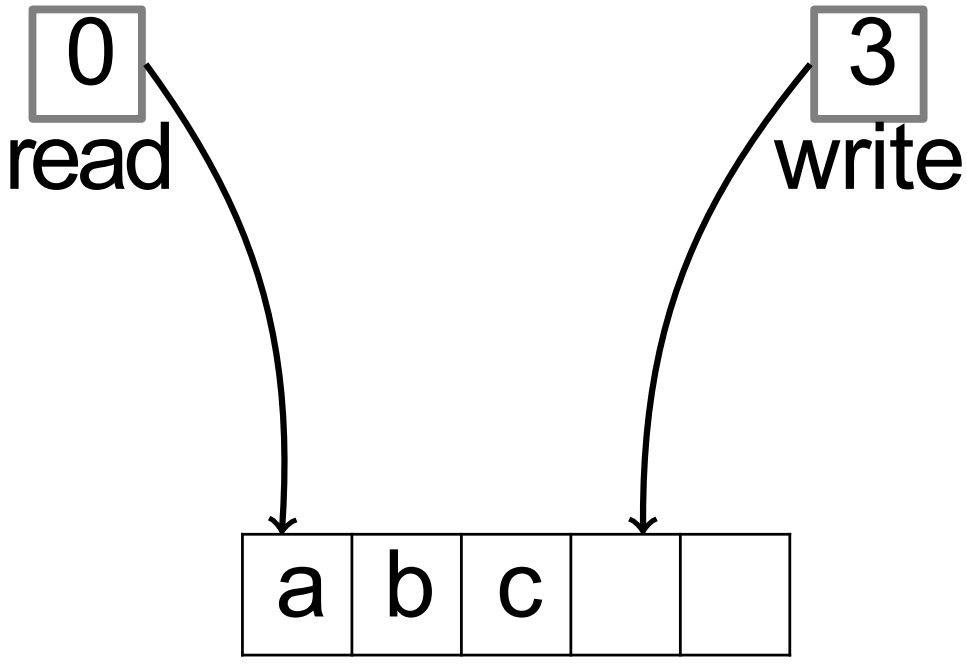
Enqueue(c)

Queue Implementation with Array

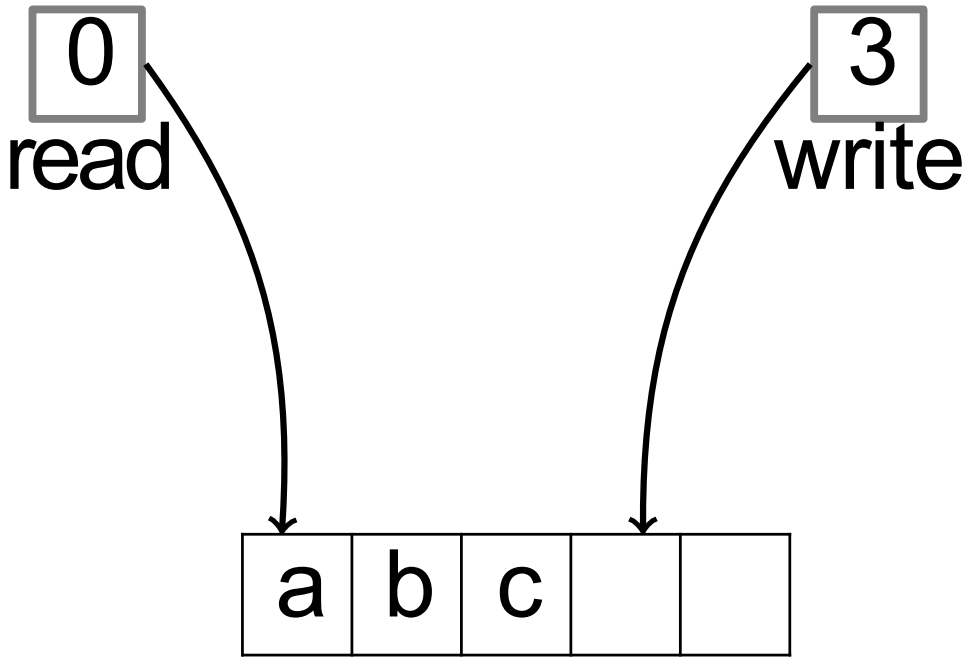


Enqueue(c)

Queue Implementation with Array

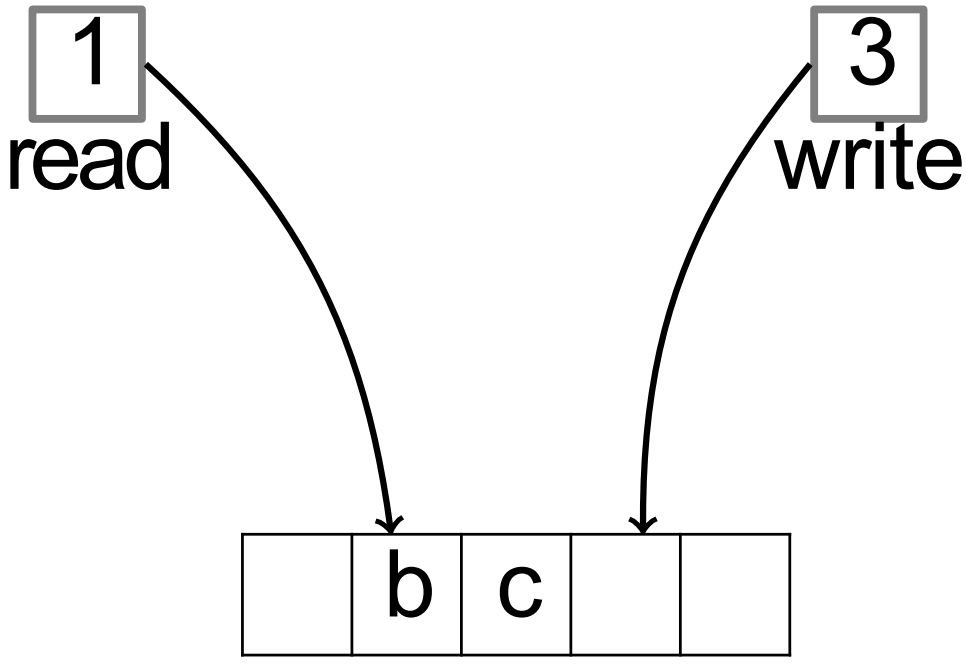


Queue Implementation with Array



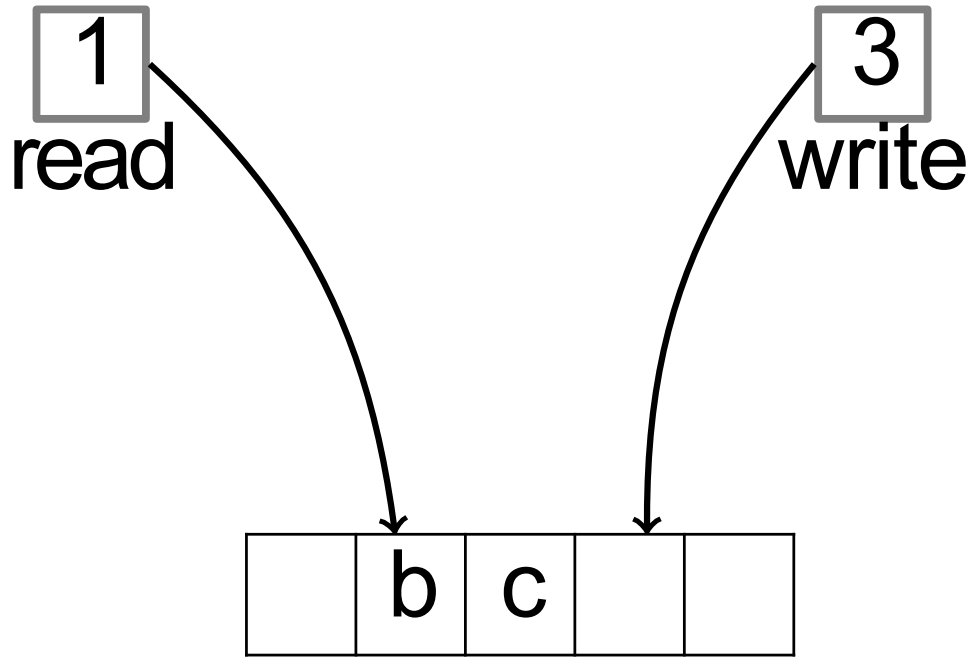
`Dequeue()`

Queue Implementation with Array

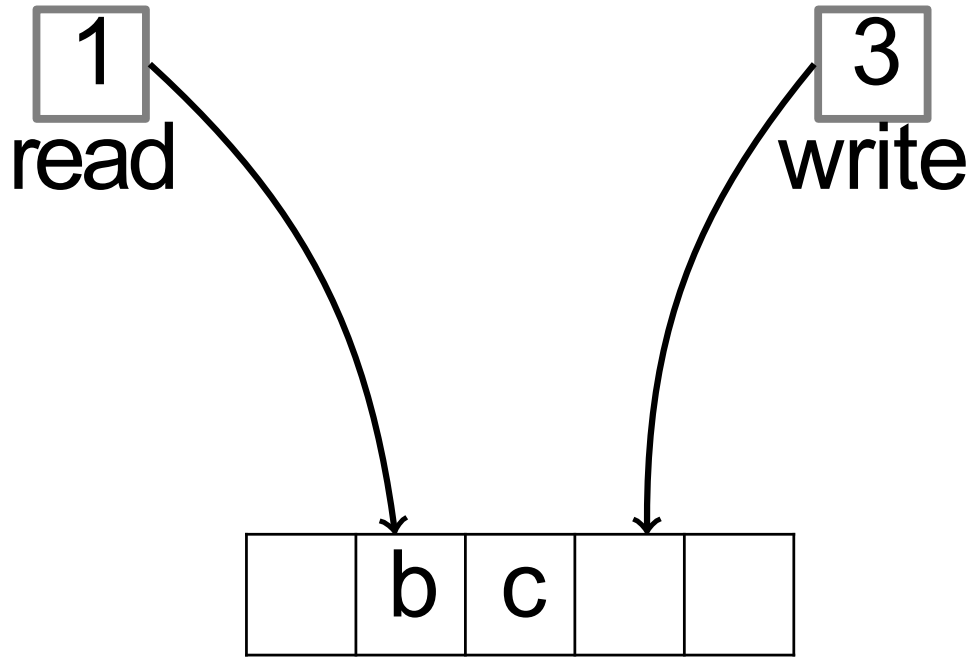


`Dequeue()` → a

Queue Implementation with Array

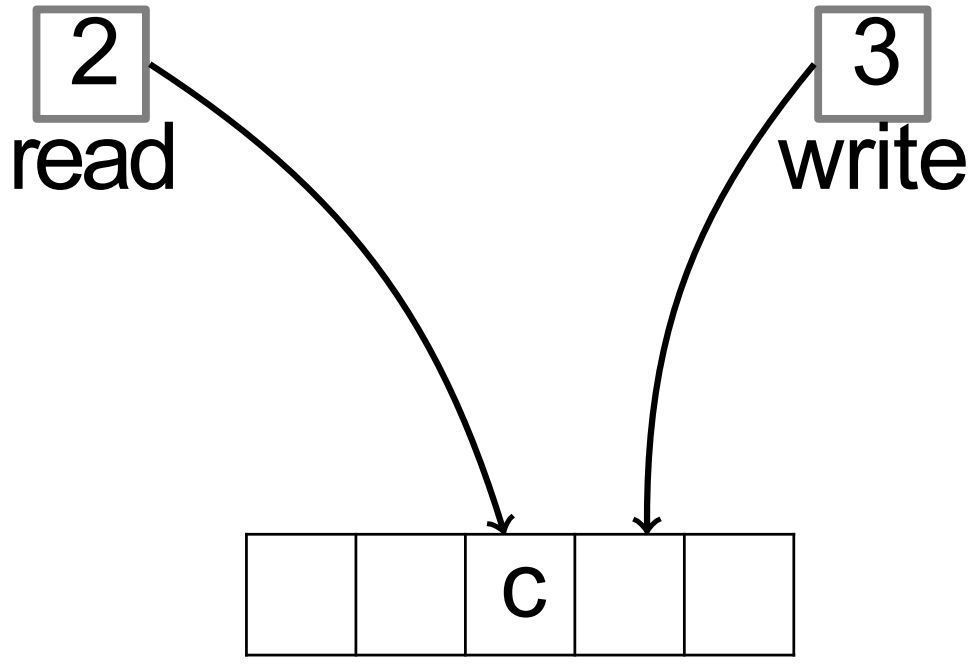


Queue Implementation with Array



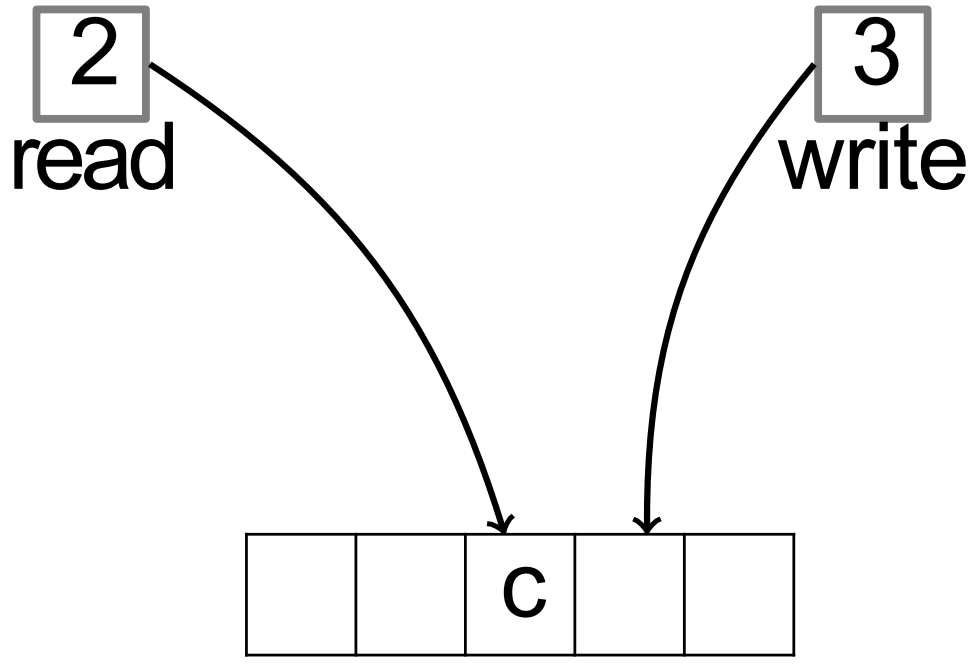
Dequeue()

Queue Implementation with Array

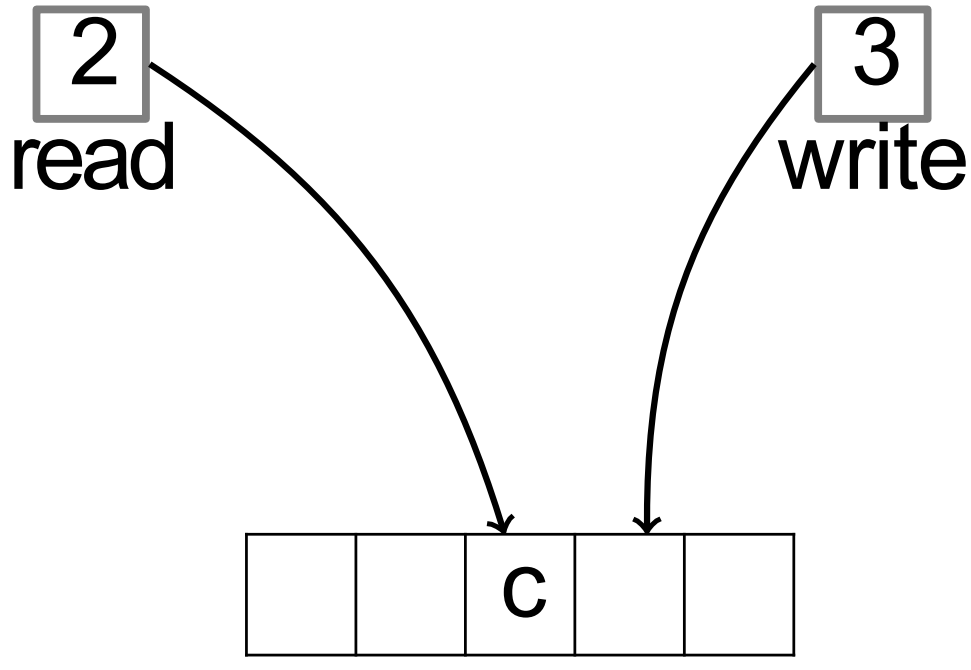


`Dequeue()` → `b`

Queue Implementation with Array

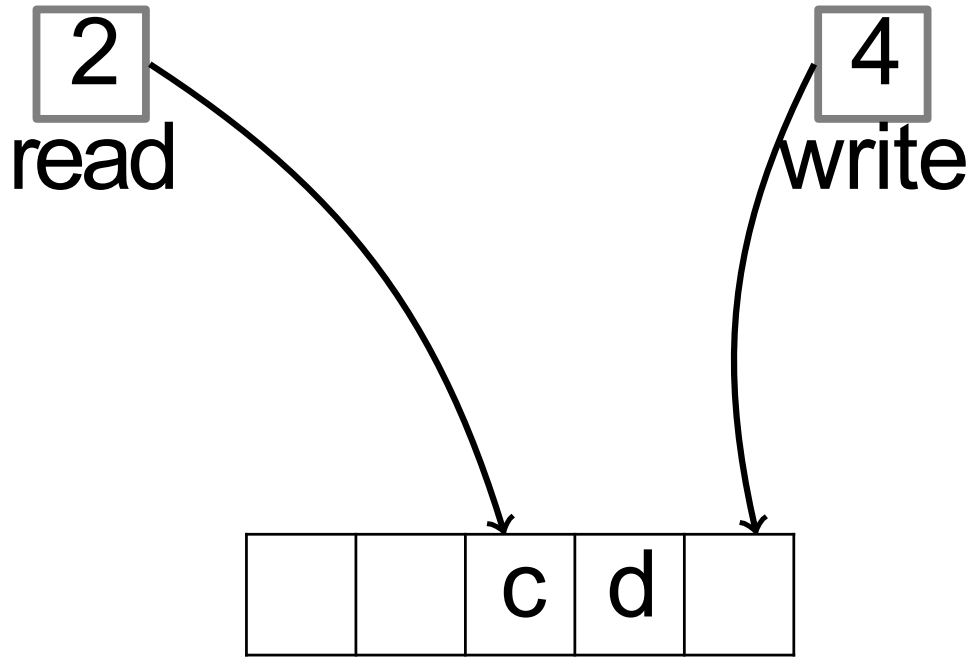


Queue Implementation with Array



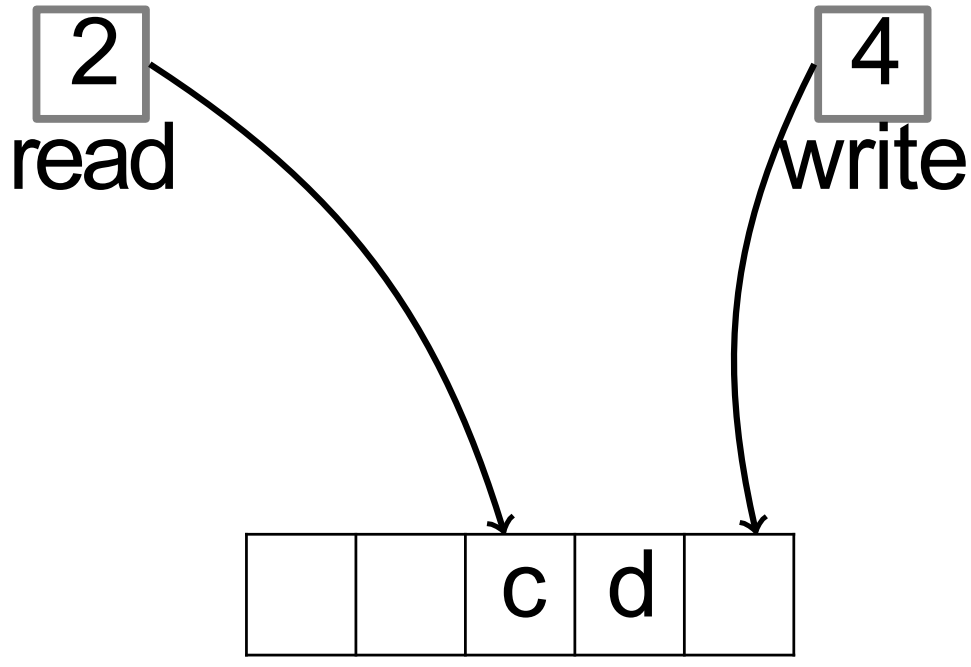
Enqueue (d)

Queue Implementation with Array

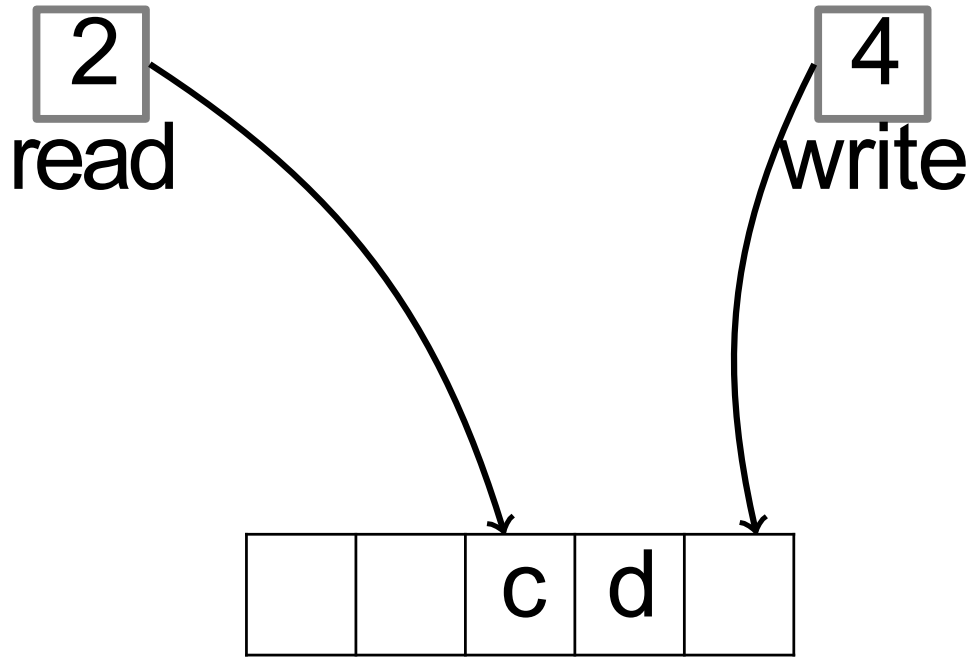


Enqueue (d)

Queue Implementation with Array

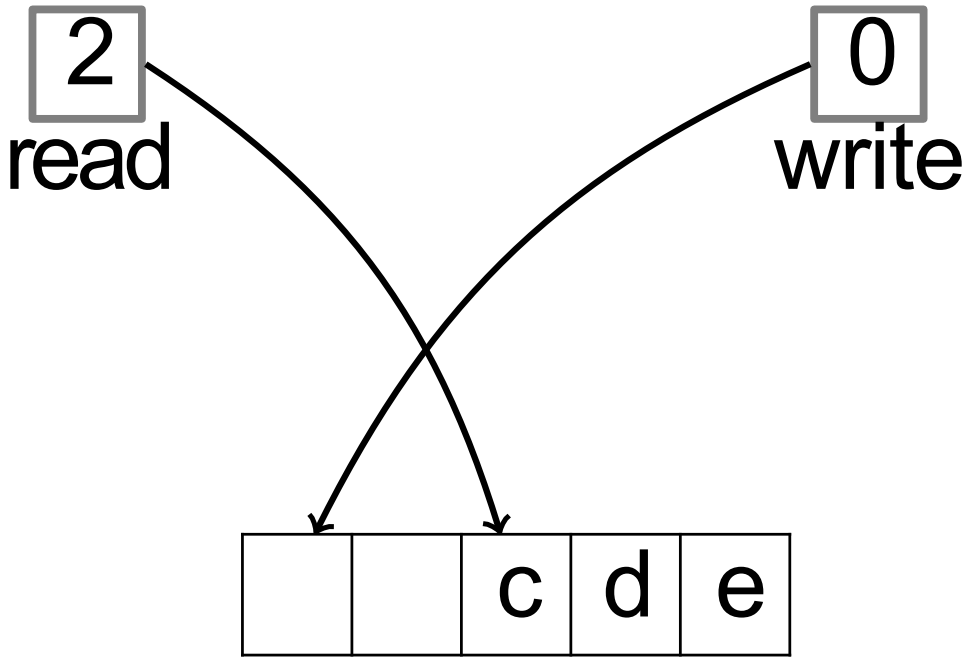


Queue Implementation with Array



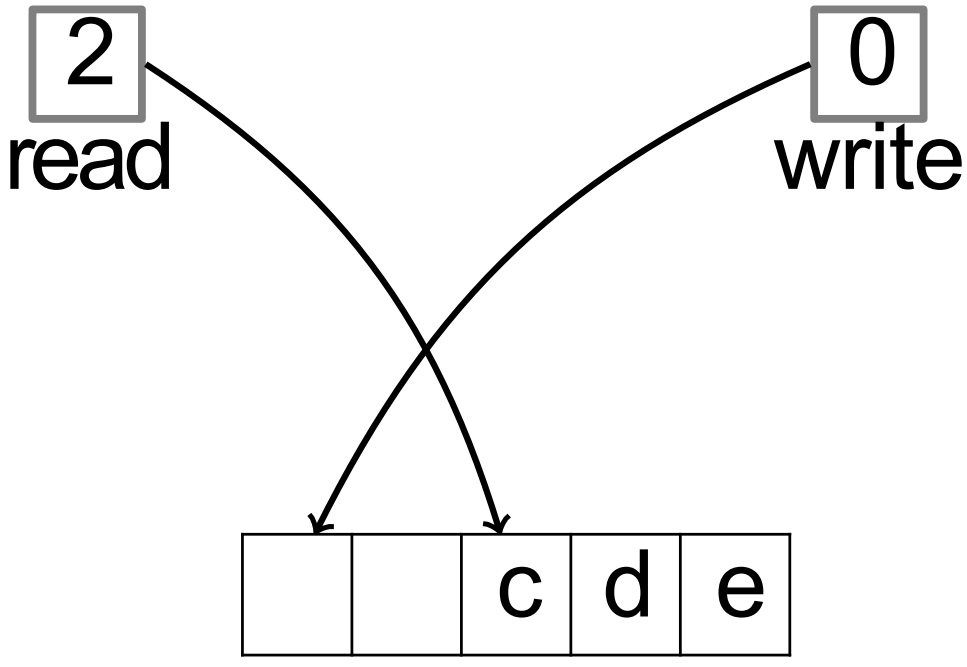
Enqueue (e)

Queue Implementation with Array

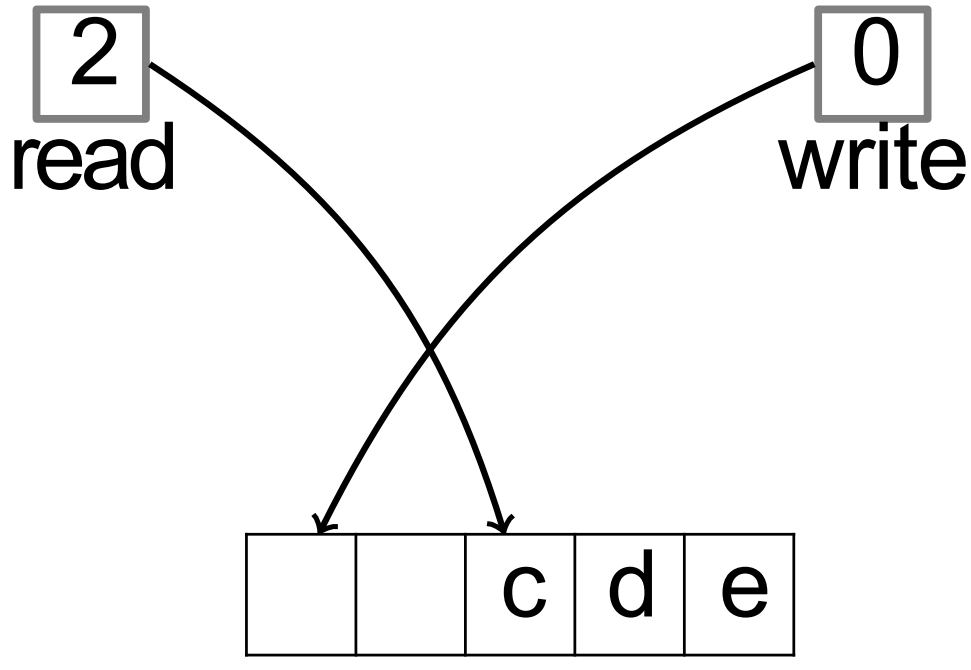


Enqueue (e)

Queue Implementation with Array

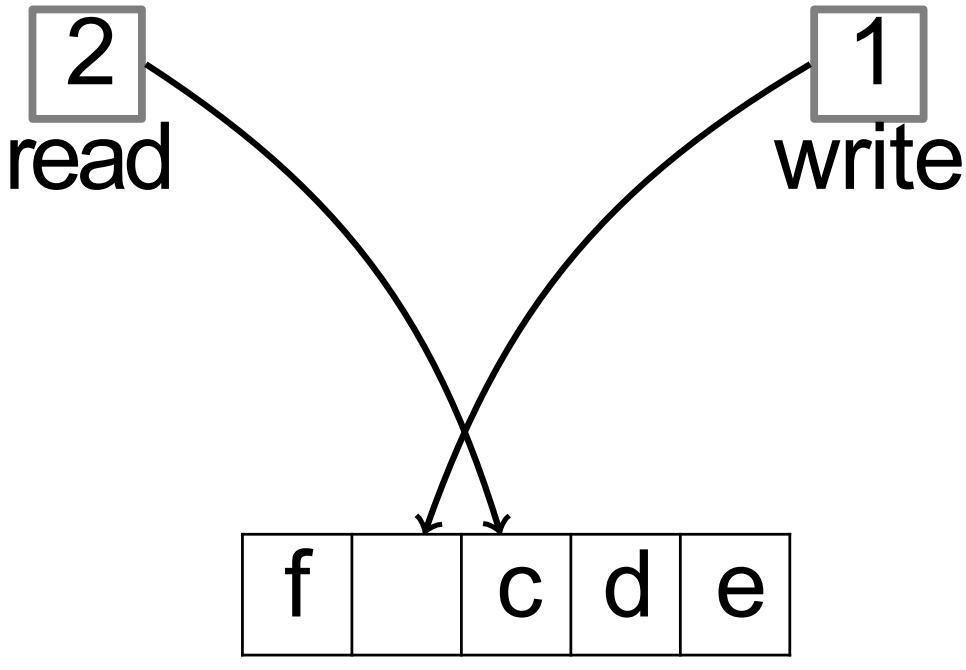


Queue Implementation with Array



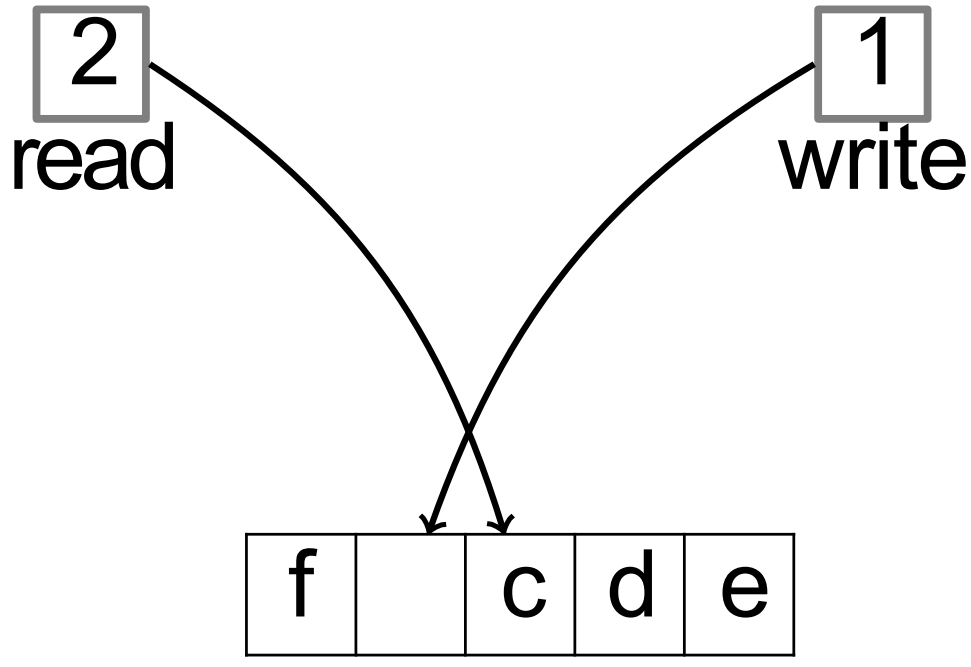
`Enqueue(f)`

Queue Implementation with Array

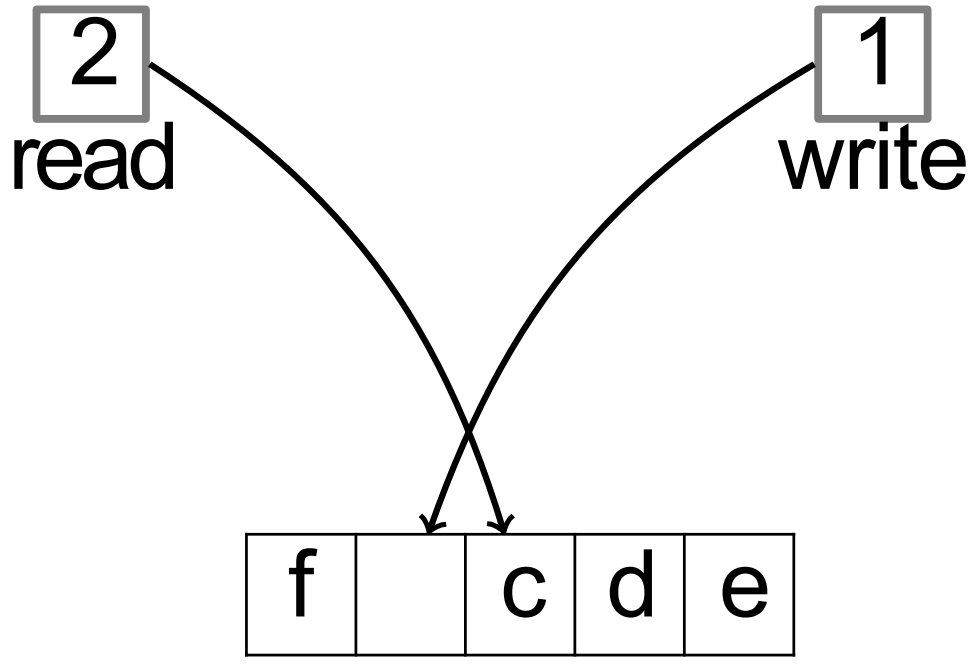


`Enqueue(f)`

Queue Implementation with Array

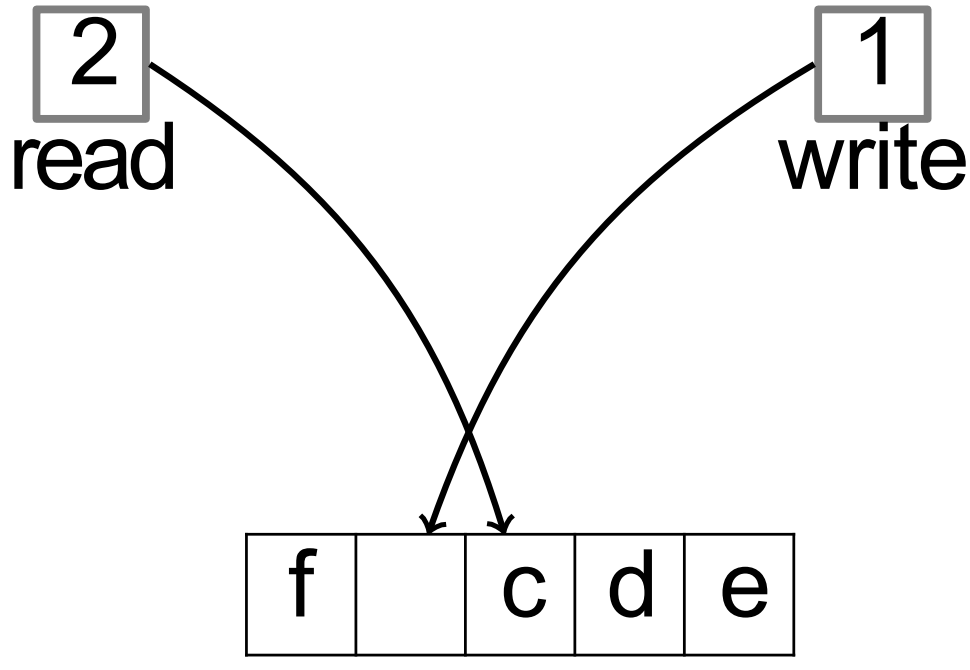


Queue Implementation with Array



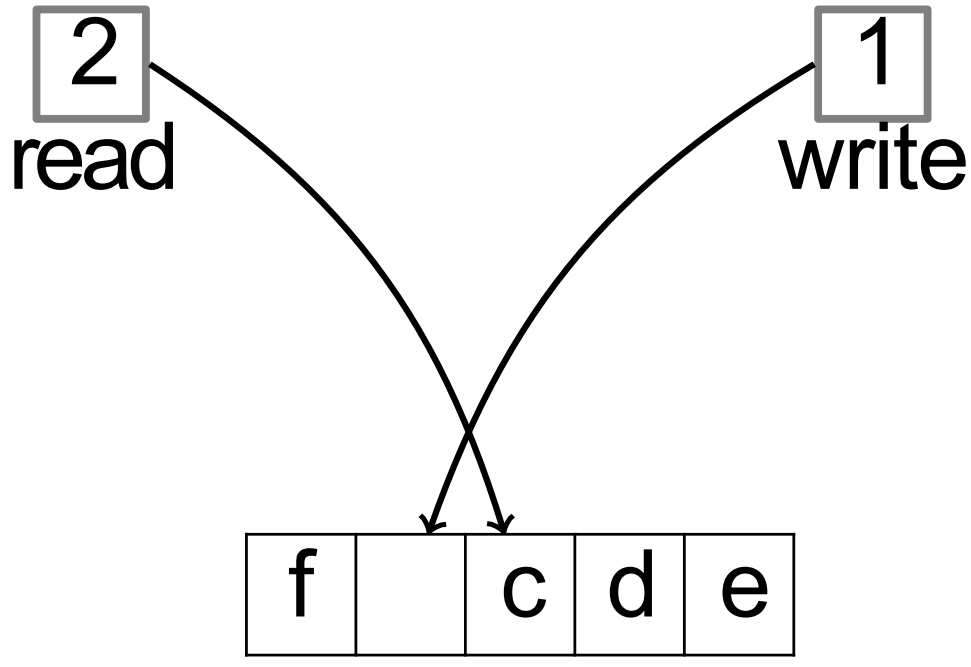
Enqueue (g)

Queue Implementation with Array

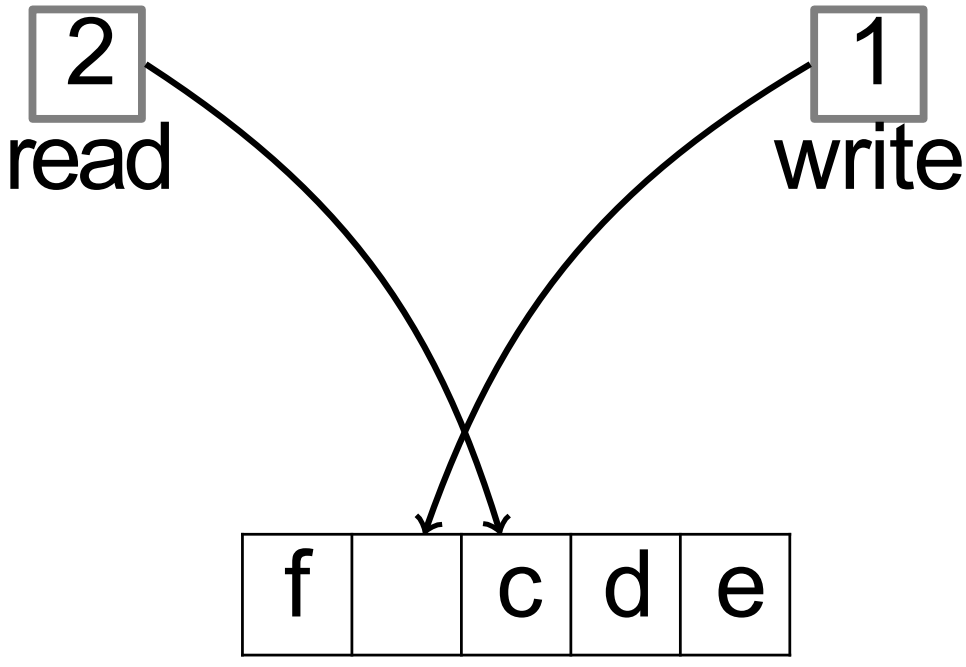


`Enqueue(g)` → `ERROR`

Queue Implementation with Array

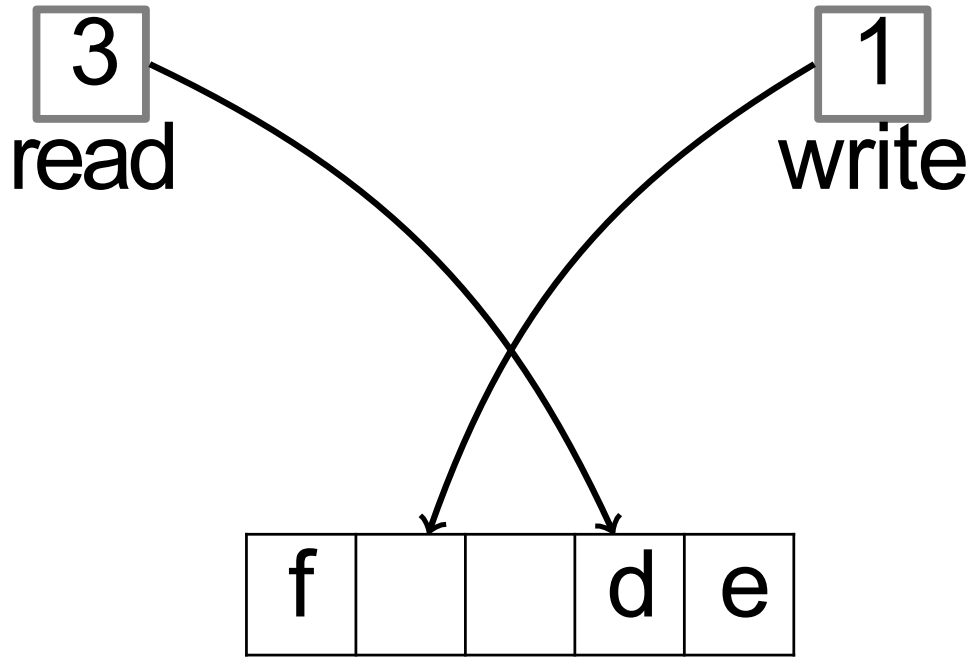


Queue Implementation with Array



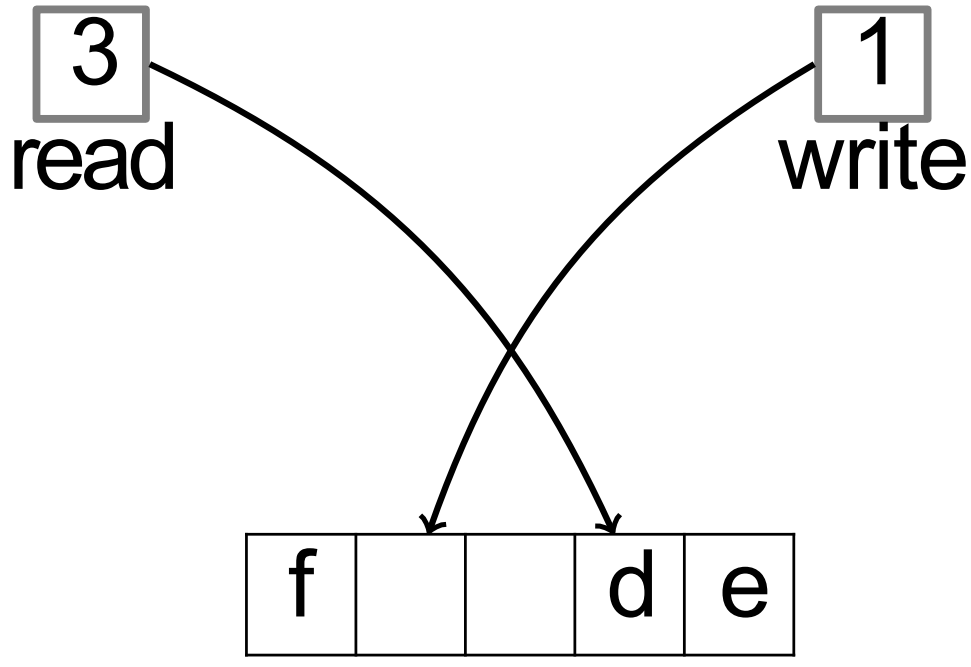

```
Dequeue()
```

Queue Implementation with Array

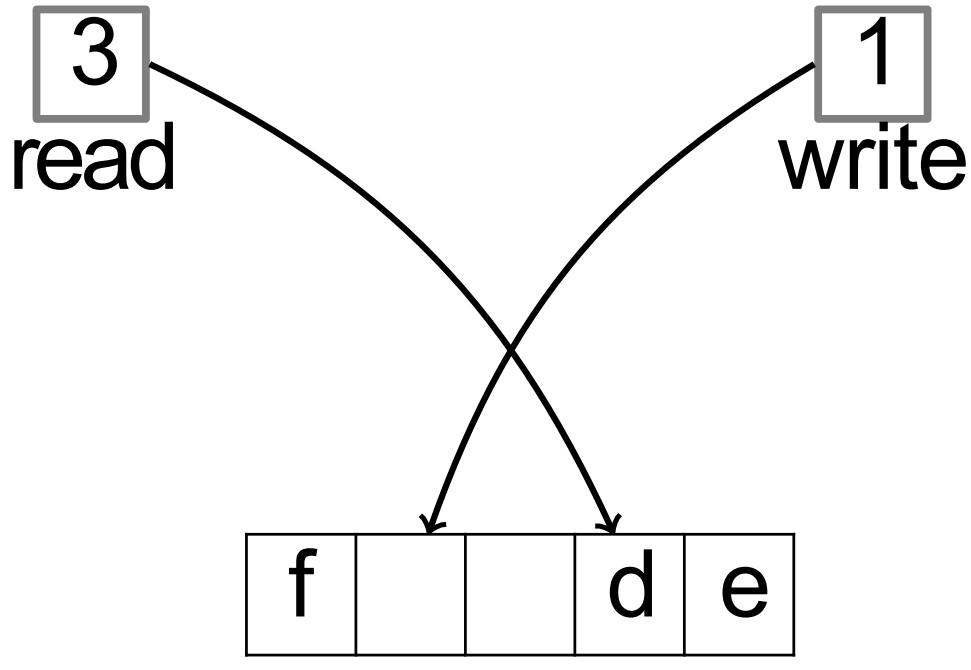


`Dequeue()` → `c`

Queue Implementation with Array

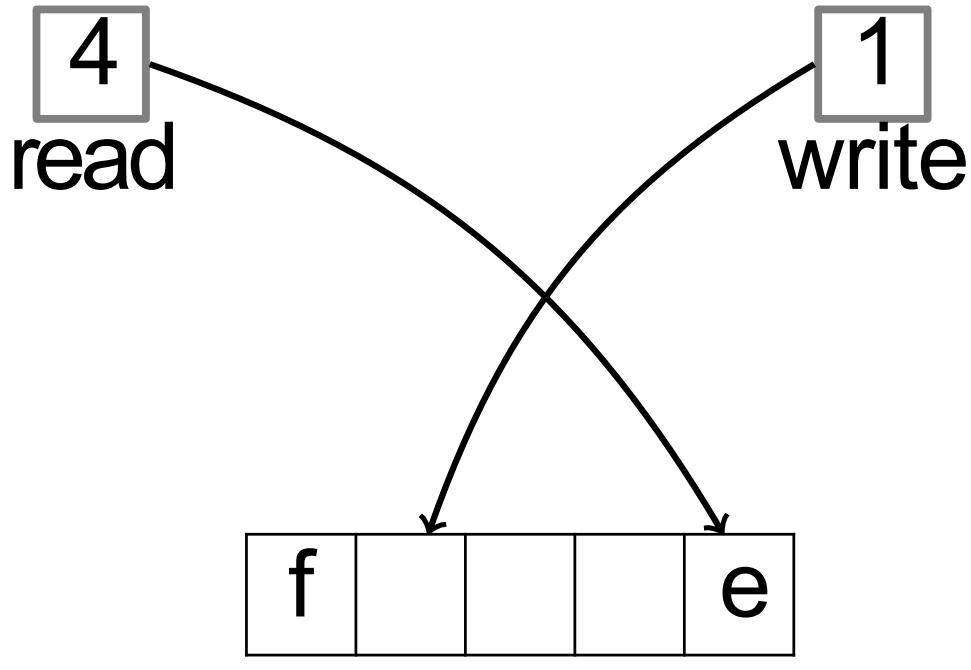


Queue Implementation with Array



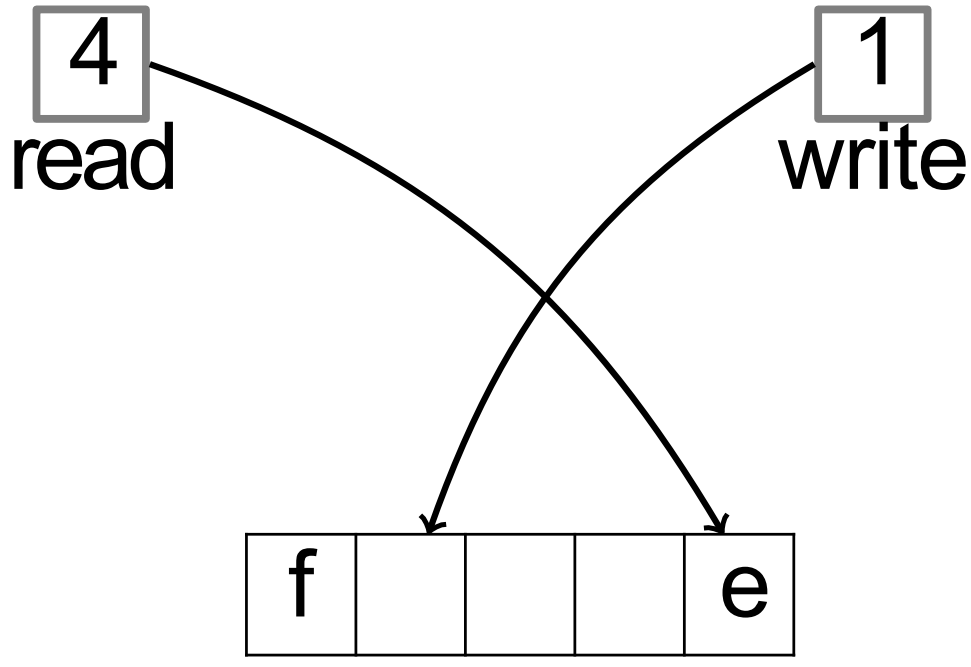
`Dequeue()`

Queue Implementation with Array

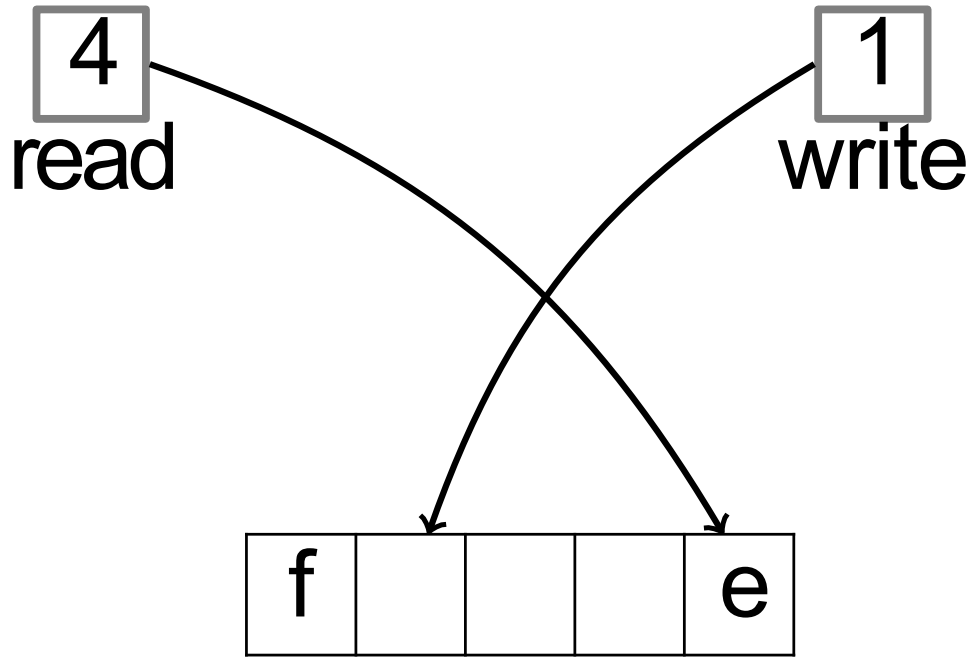


`Dequeue()` → `d`

Queue Implementation with Array

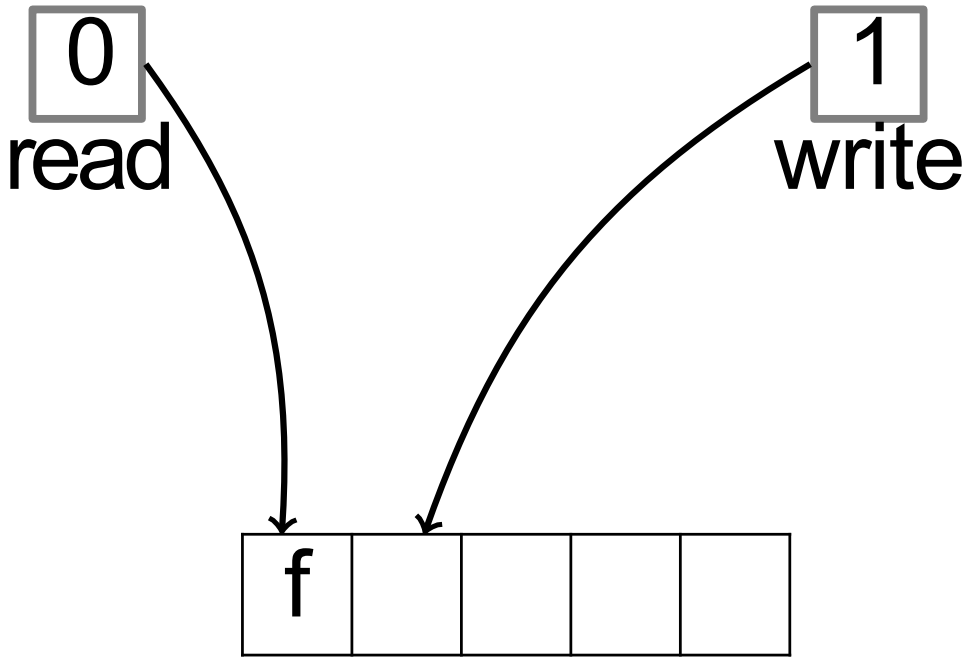


Queue Implementation with Array



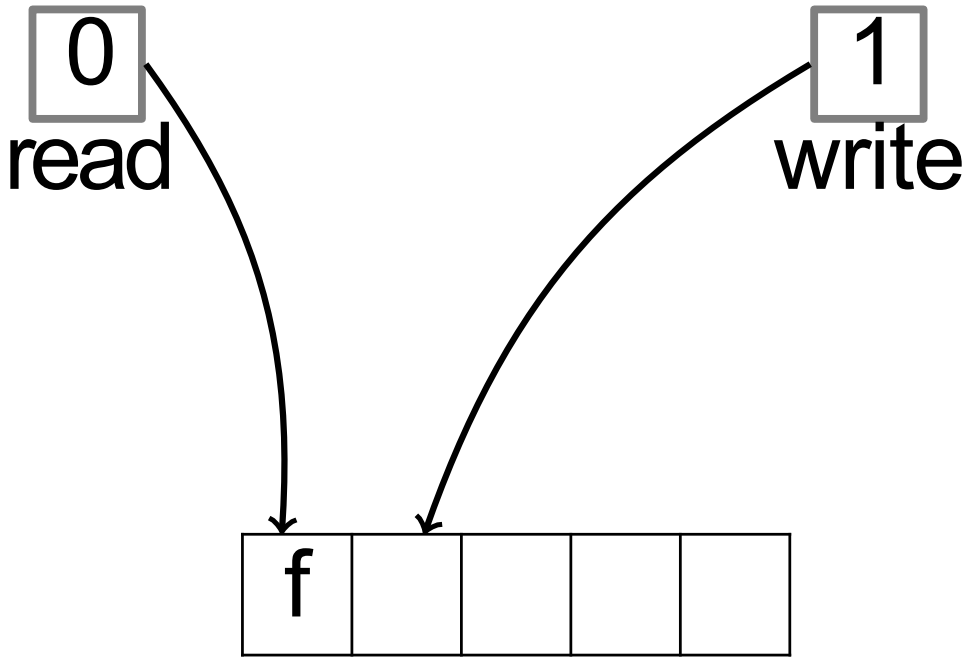

```
Dequeue()
```

Queue Implementation with Array

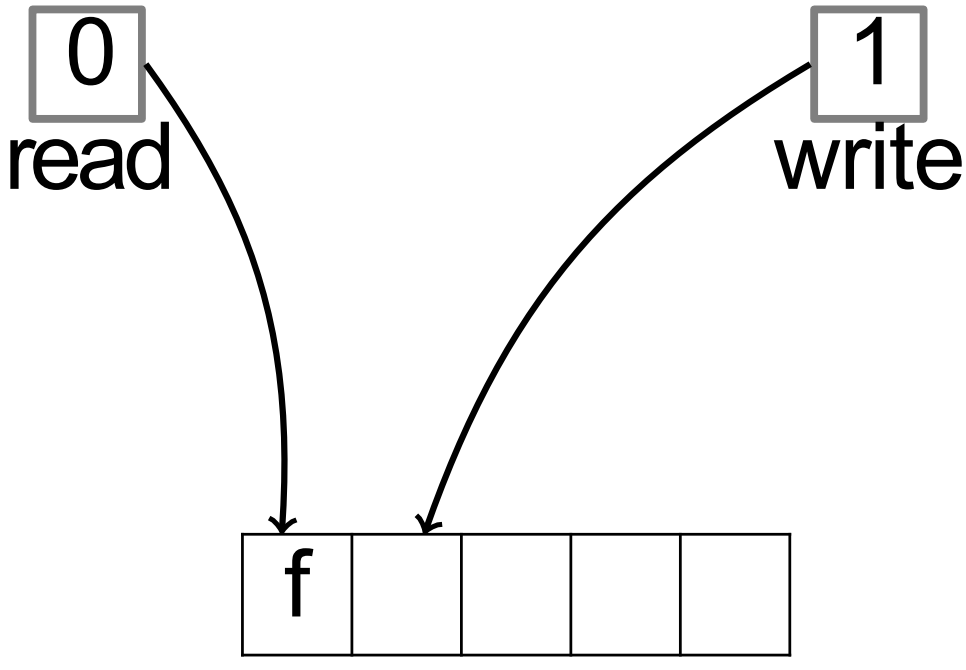


`Dequeue()` → `e`

Queue Implementation with Array

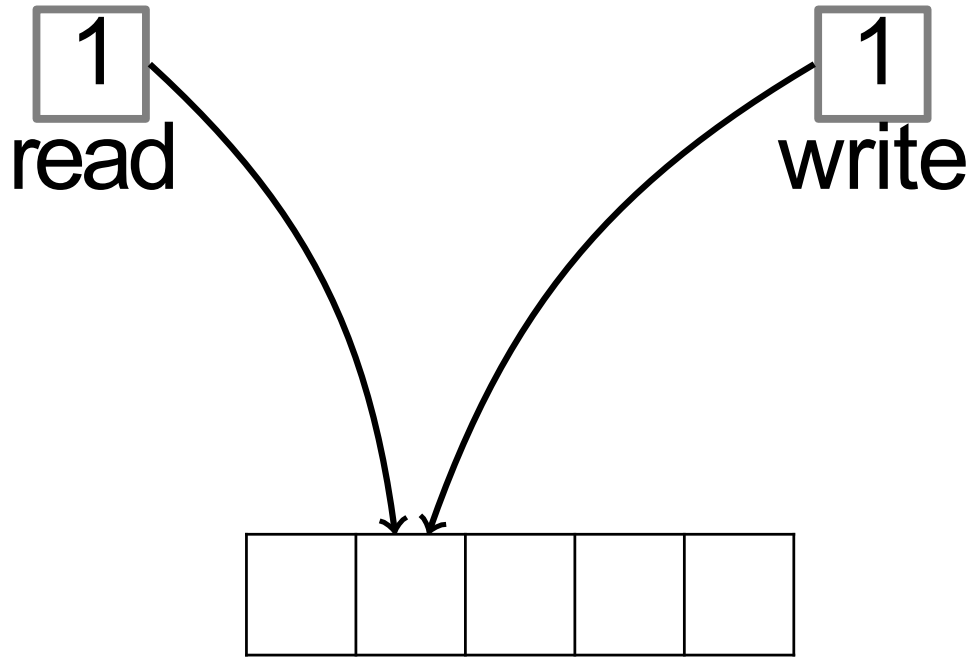


Queue Implementation with Array



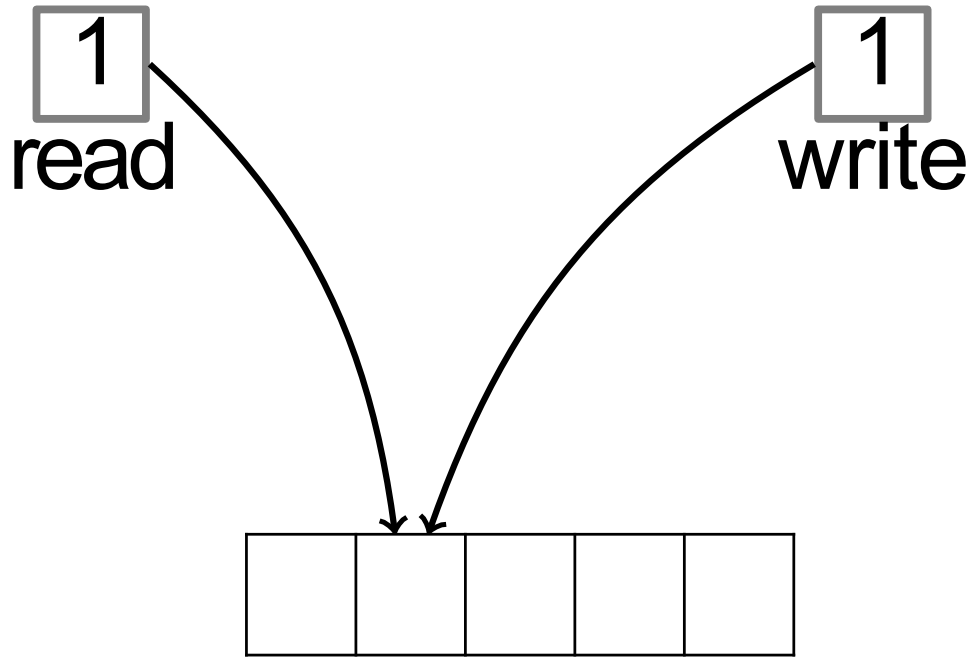

```
Dequeue()
```

Queue Implementation with Array

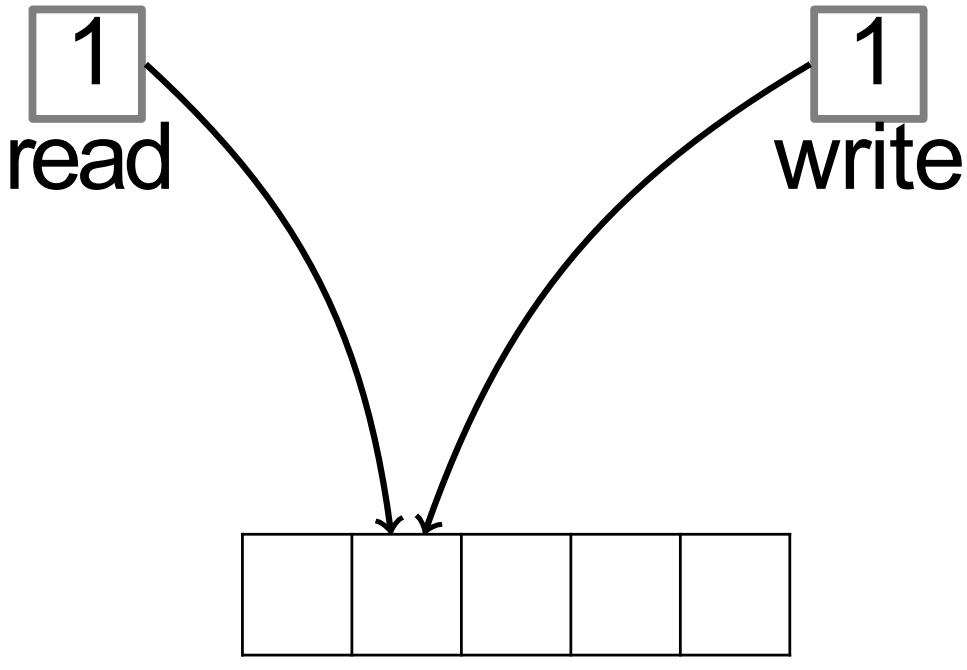


`Dequeue()` → `f`

Queue Implementation with Array

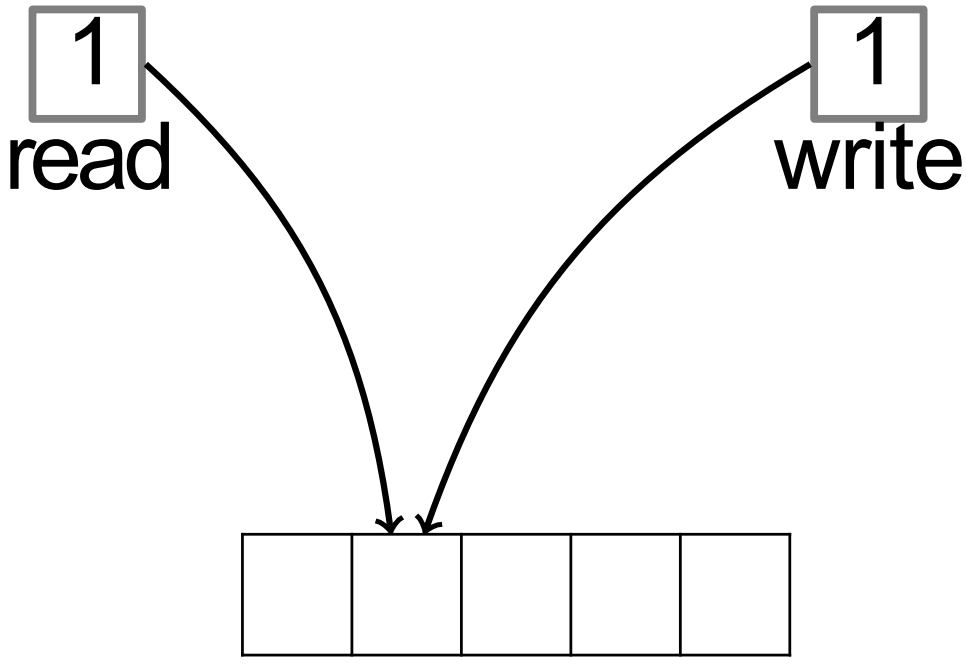


Queue Implementation with Array



`Empty()`

Queue Implementation with Array



`Empty()` → `True`

Summary

Summary

- Queues can be implemented with either a linked list (with tail pointer) or an array.

Summary

- Queues can be implemented with either a linked list (with tail pointer) or an array.
- Each queue operation is $O(1)$: Enqueue, Dequeue, Empty.