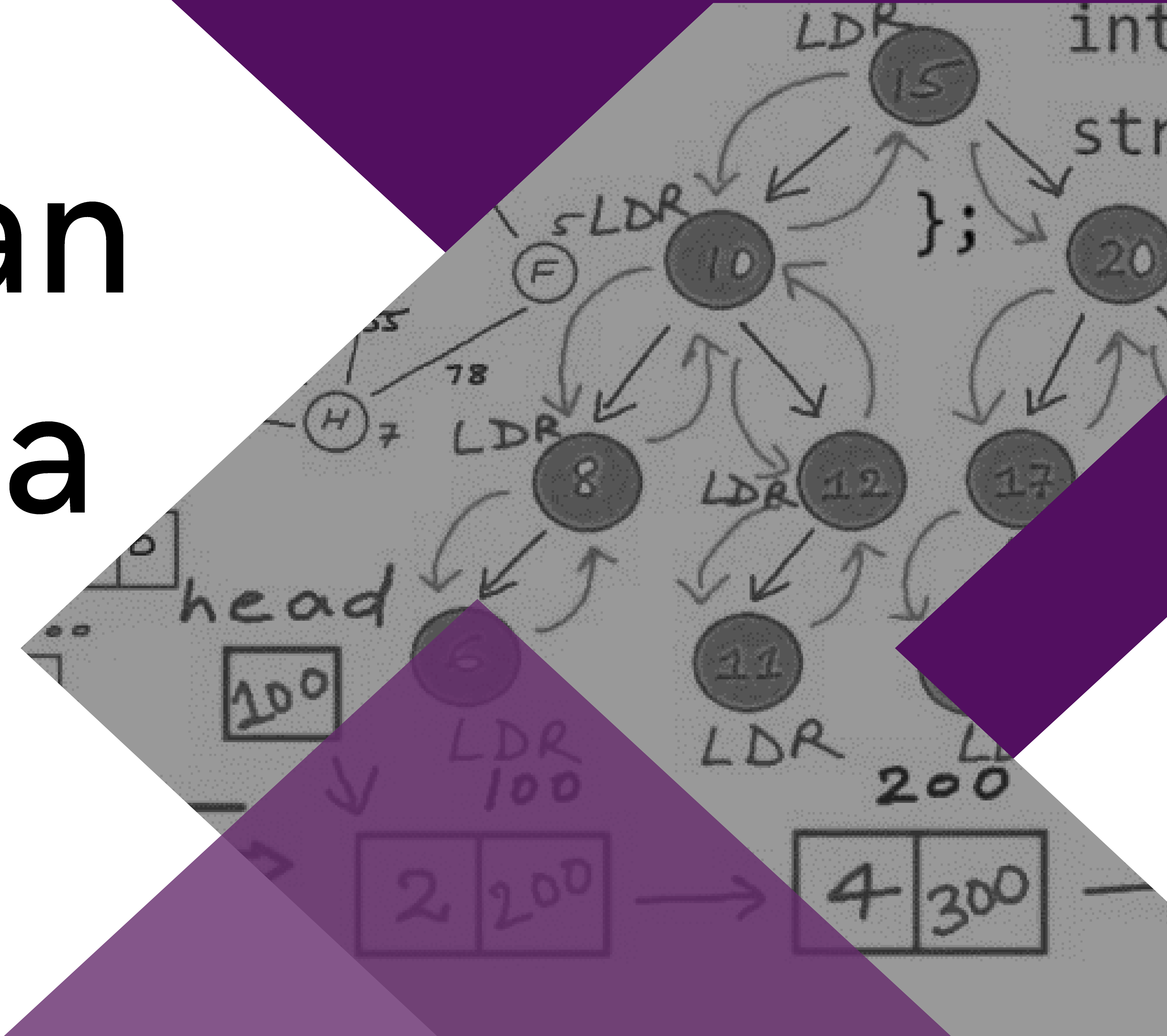
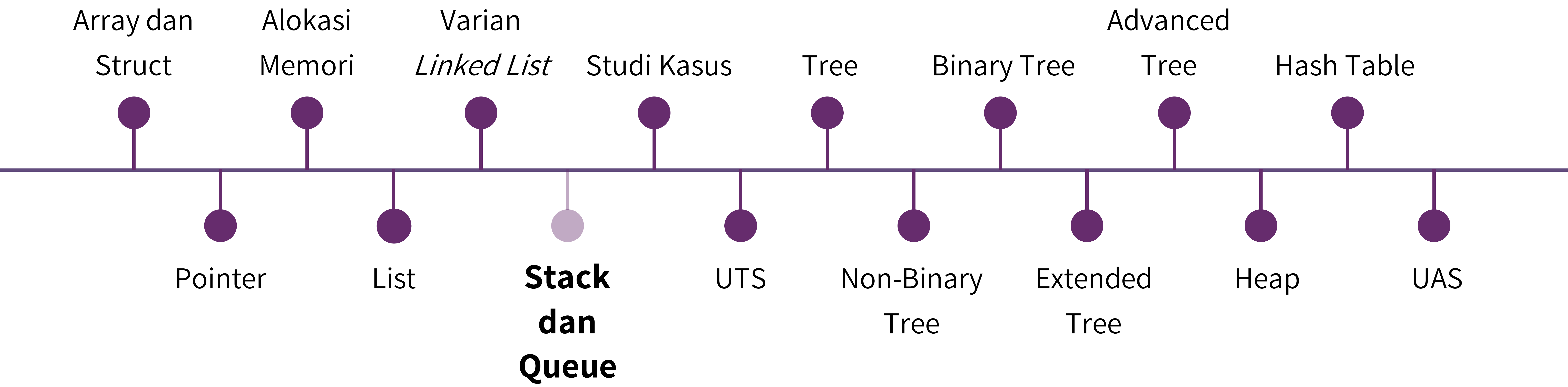
Algoritma dan Struktur Data



Pekan 6



Array dan Struct

Pointer

Alokasi Memori

List

Varian *Linked List*

Stack dan Queue

Studi Kasus

UTS

Tree

Non-Binary Tree

Binary Tree

Extended Tree

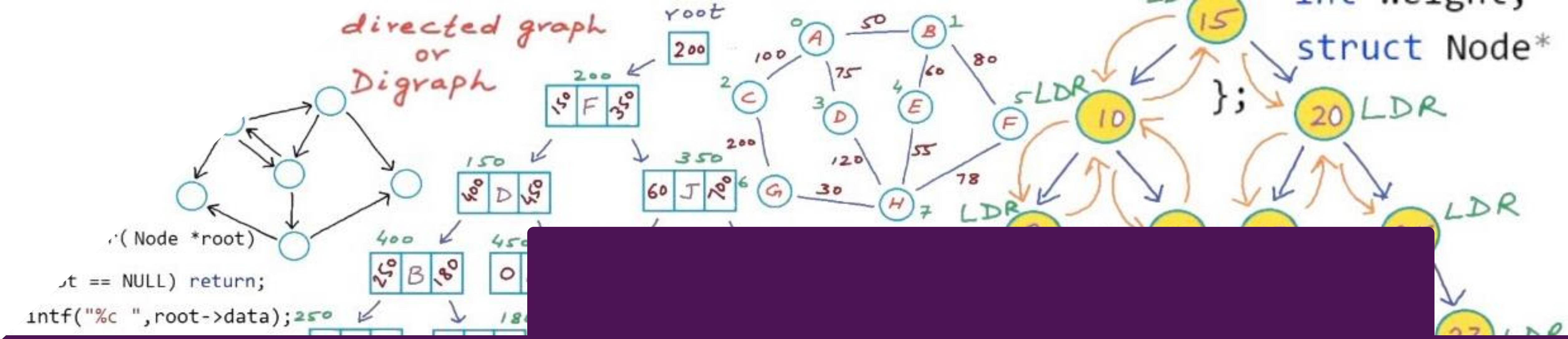
Advanced Tree

Heap

Hash Table

UAS

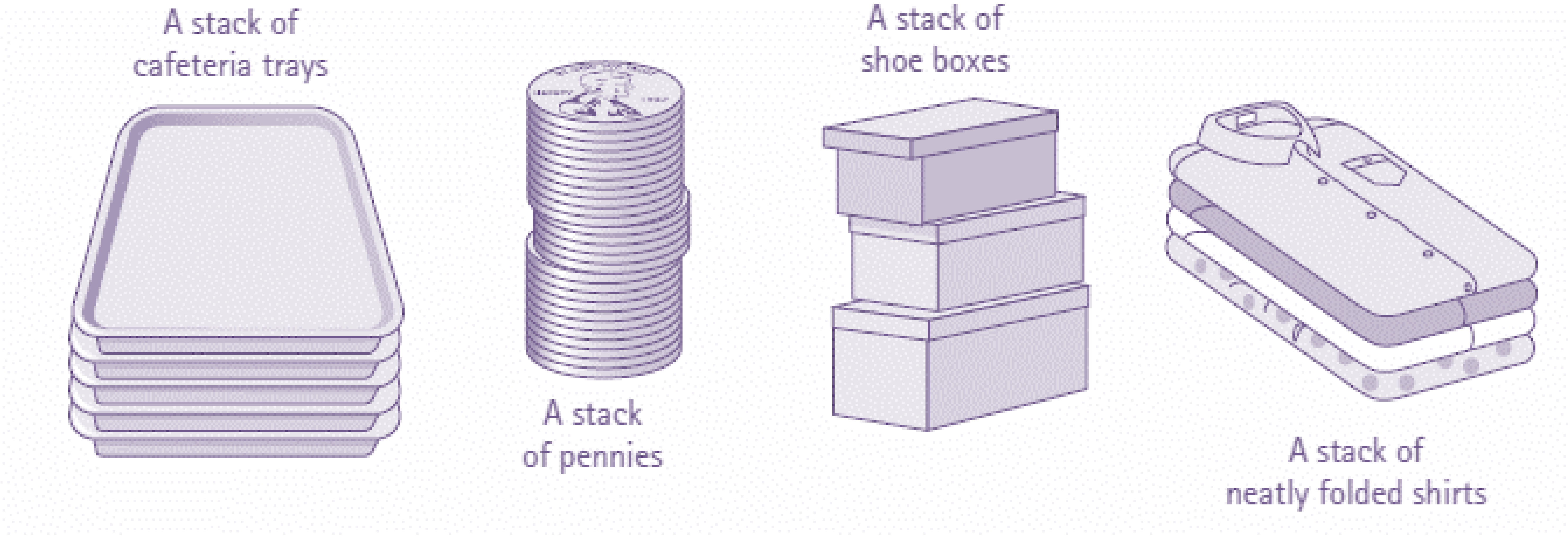
Tujuan



1	Mahasiswa memahami konsep stack dan operasinya
2	Mahasiswa memahami konsep queue dan operasinya
3	Mahasiswa mampu mengimplementasikan ADT stack dan queue dalam bentuk array
4	Mahasiswa mampu mengimplementasikan ADT stack dan queue dalam bentuk linked list.

ADT - Stack

Stack


A stack of cafeteria trays
A stack of pennies
A stack of shoe boxes
A stack of neatly folded shirts

Stack

An abstract data type in which elements are added and removed from only one end; a “last in, first out” (LIFO) structure.

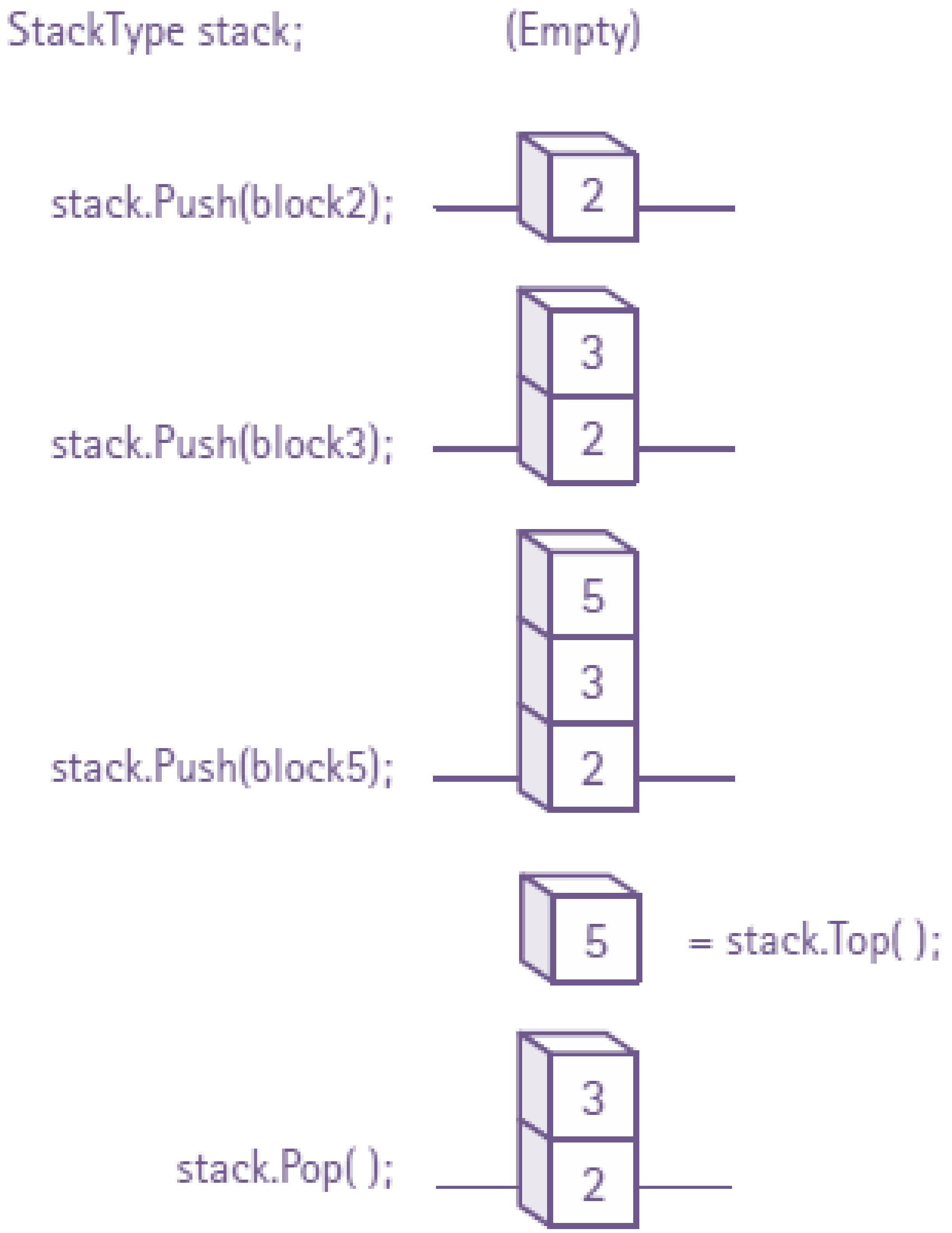
A last-in, first-out (LIFO) data structure that operates much like a pile of papers: we add new elements to the top of the stack and remove elements starting with the top of the stack.

A list data structure in which elements are inserted in and removed from the same end, the top of the stack

Stack Operations

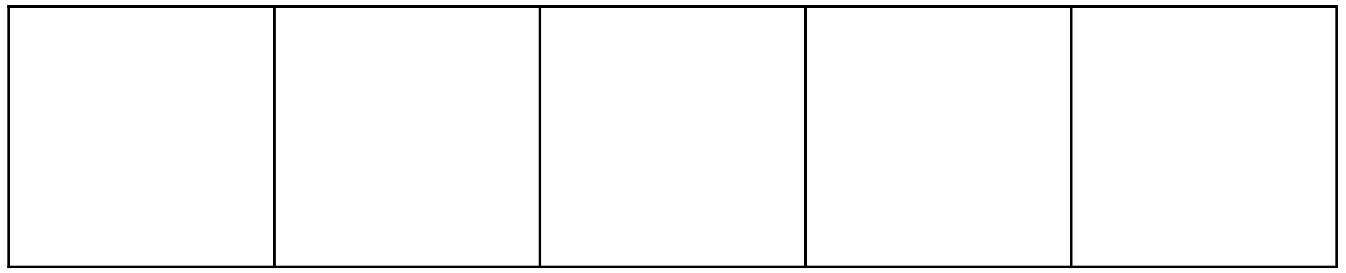
Push(key)	Add a new element to the top of the stack.
Pop()	Remove the element from the top of the stack and return it.

Top()	Return most recently added key
IsEmpty()	Check whether there are any elements



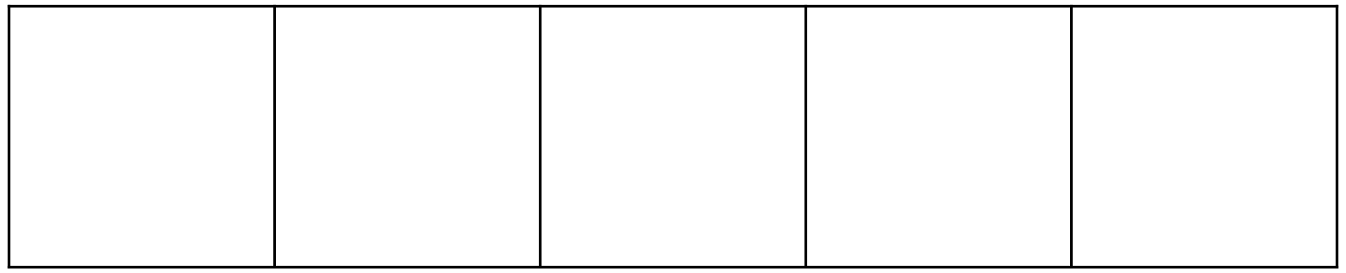
Stack Implementation with Arrays

numElements:0



Stack Implementation with Arrays

numElements:0



```
Push(a)
```

Stack Implementation with Arrays

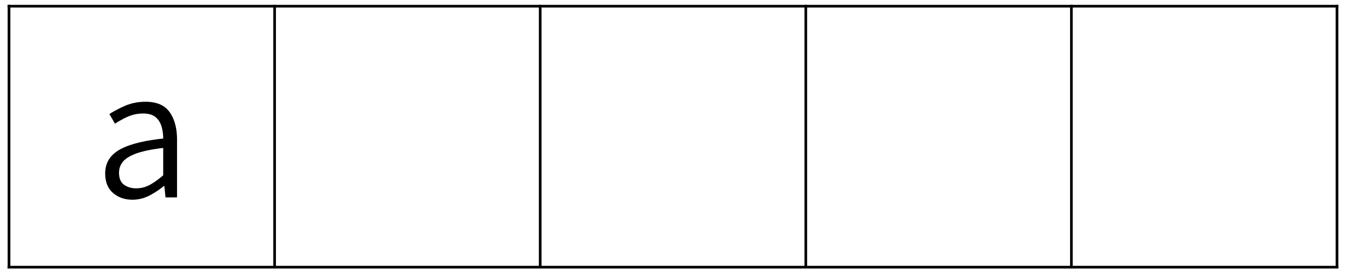
numElements:1

a				

```
Push(a)
```

Stack Implementation with Arrays

numElements:1



a				

Stack Implementation with Arrays

numElements:1

a				

`Push(b)`

Stack Implementation with Arrays

numElements:2

a	b			

```
Push(b)
```

Stack Implementation with Arrays

numElements:2

a	b			

Stack Implementation with Arrays

numElements:2

a	b			

`Top()`

Stack Implementation with Arrays

numElements:2

a	b			

`Top()` → 5

Stack Implementation with Arrays

numElements:2

a	b			

```
Push(c)
```

Stack Implementation with Arrays

numElements:3

a	b	c		

```
Push(c)
```

Stack Implementation with Arrays

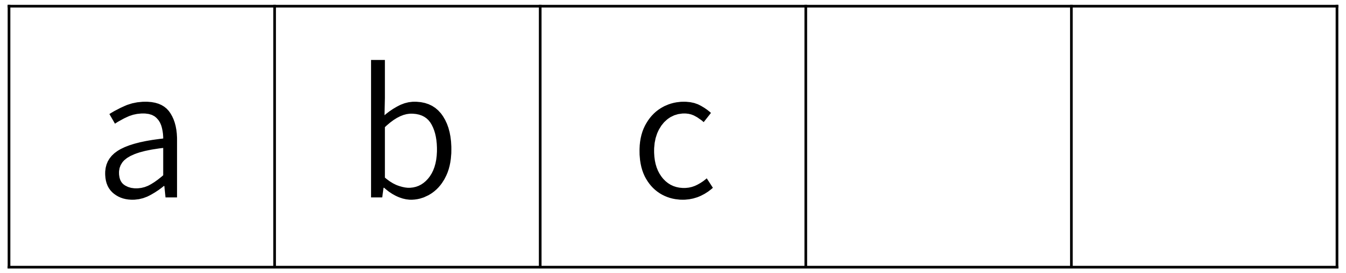
numElements:3

a	b	c		

```
Push(c)
```

Stack Implementation with Arrays

numElements:3



a	b	c		

Stack Implementation with Arrays

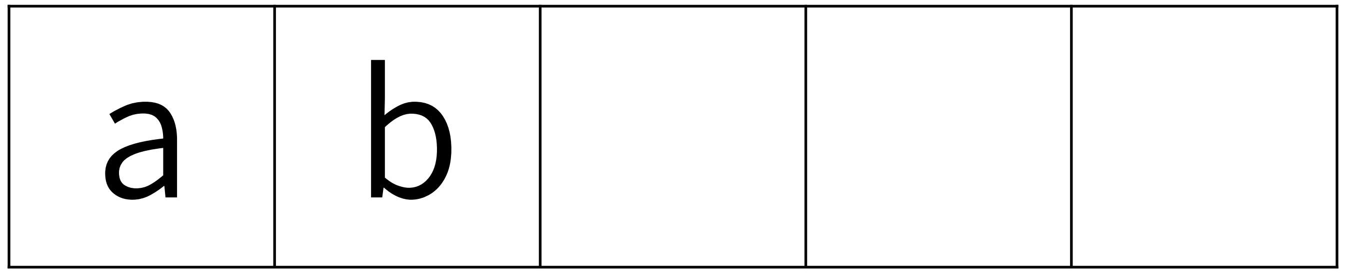
numElements:3

a	b	c		

```
Pop()
```

Stack Implementation with Arrays

numElements:2



a	b			

Pop() → c

Stack Implementation with Arrays

numElements:2

a	b			

Stack Implementation with Arrays

numElements:2

a	b			

```
Push(d)
```

Stack Implementation with Arrays

numElements:3

a	b	d		

```
Push(d)
```

Stack Implementation with Arrays

numElements:3

a	b	d		

Stack Implementation with Arrays

numElements:3

a	b	d		

`Push(e)`

Stack Implementation with Arrays

numElements:4

a	b	d	e	

```
Push(e)
```

Stack Implementation with Arrays

numElements:4

a	b	d	e	

Stack Implementation with Arrays

numElements:4

a	b	d	e	

`Push(f)`

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

```
Push(f)
```

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

```
Push(g)
```

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

`Push(g)→ ERROR`

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

`Empty()`

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

`Empty()` → `False`

Stack Implementation with Arrays

numElements:5

a	b	d	e	f

`Pop()`

Stack Implementation with Arrays

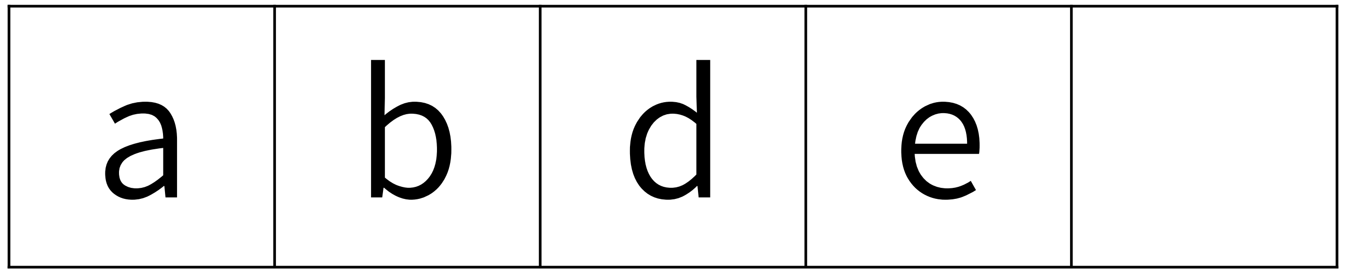
numElements:4

a	b	d	e	

`Pop()→f`

Stack Implementation with Arrays

numElements:4



a	b	d	e	

Stack Implementation with Arrays

numElements:4

a	b	d	e	

`Pop()`

Stack Implementation with Arrays

numElements:3

a	b	d		

Pop() →e

Stack Implementation with Arrays

numElements:3

a	b	d		

Stack Implementation with Arrays

numElements:3

a	b	d		

`Pop()`

Stack Implementation with Arrays

numElements:2

a	b			

```
Pop()→d
```

Stack Implementation with Arrays

numElements:2

a	b			

Stack Implementation with Arrays

numElements:2

a	b			

```
Pop()
```

Stack Implementation with Arrays

numElements:1

a				

`Pop() → b`

Stack Implementation with Arrays

numElements:1

a				

Stack Implementation with Arrays

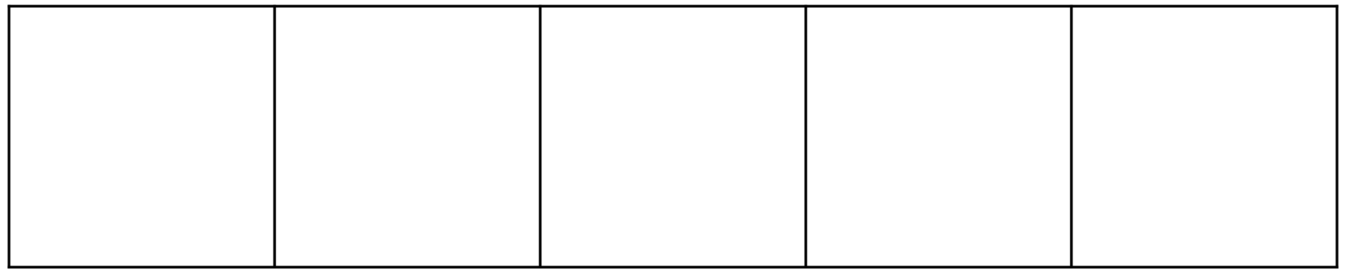
numElements:1

a				

`Pop()`

Stack Implementation with Arrays

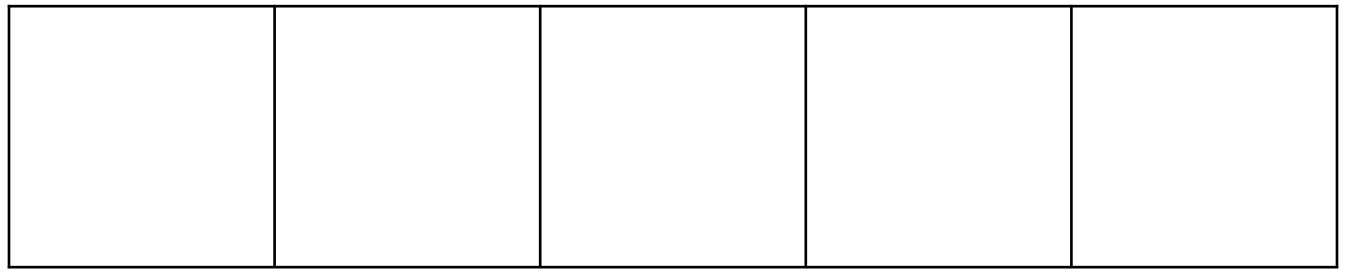
numElements:0



`Pop()→ a`

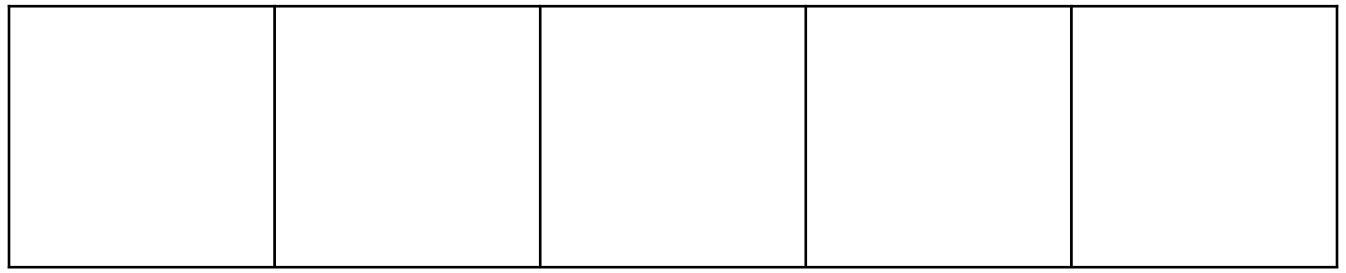
Stack Implementation with Arrays

numElements:0



Stack Implementation with Arrays

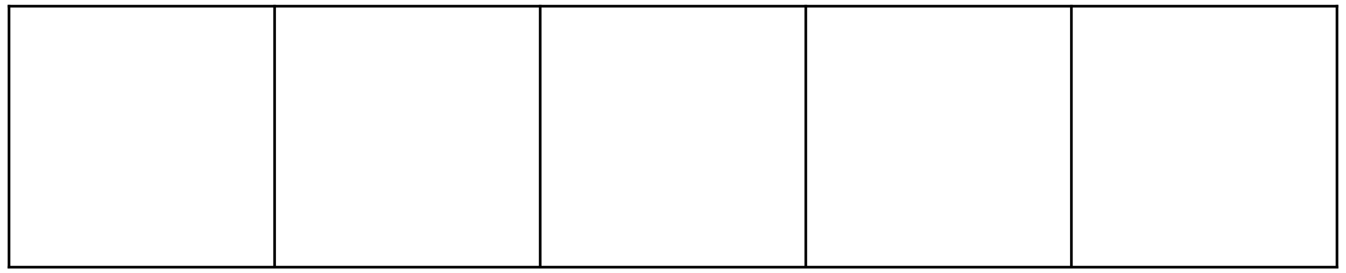
numElements:0



```
Empty()
```

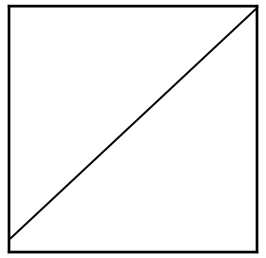
Stack Implementation with Arrays

numElements:0



`Empty()→ True`

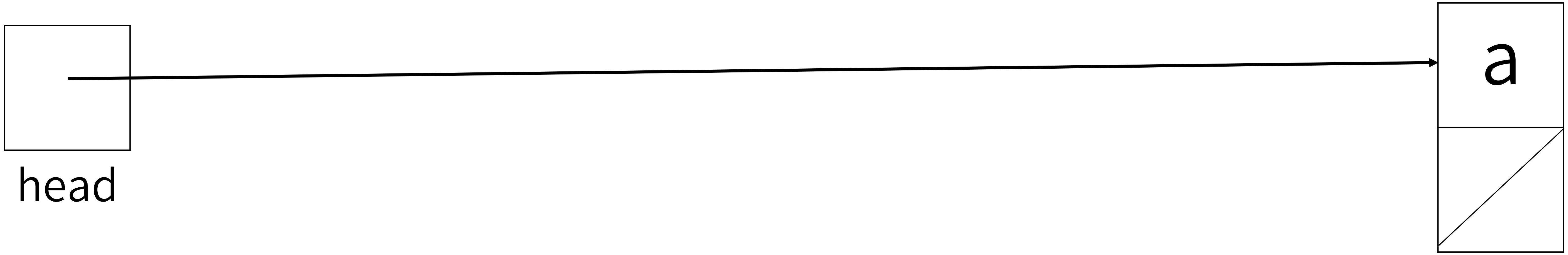
Stack Implementation with Linked List



head

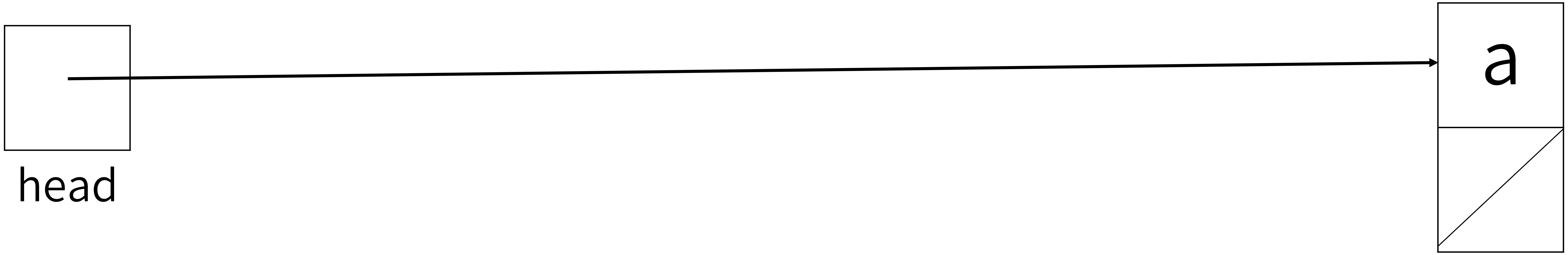
`Push(a)`

Stack Implementation with Linked List

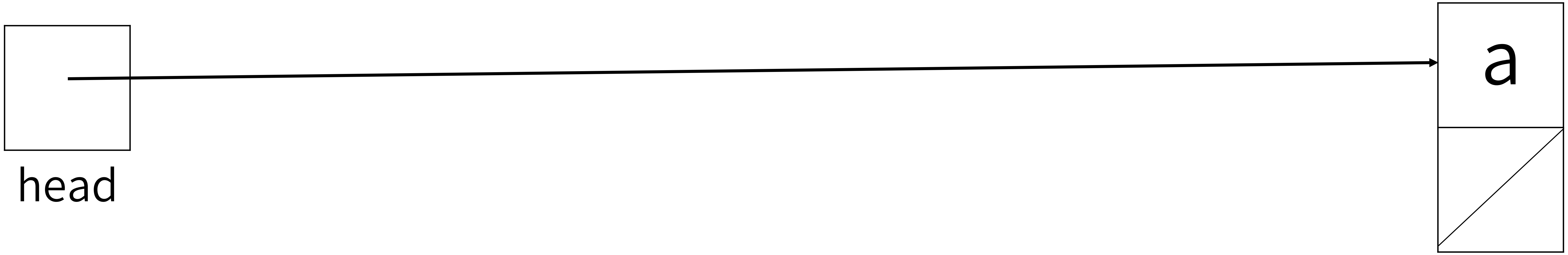


```
Push(a)
```

Stack Implementation with Linked List

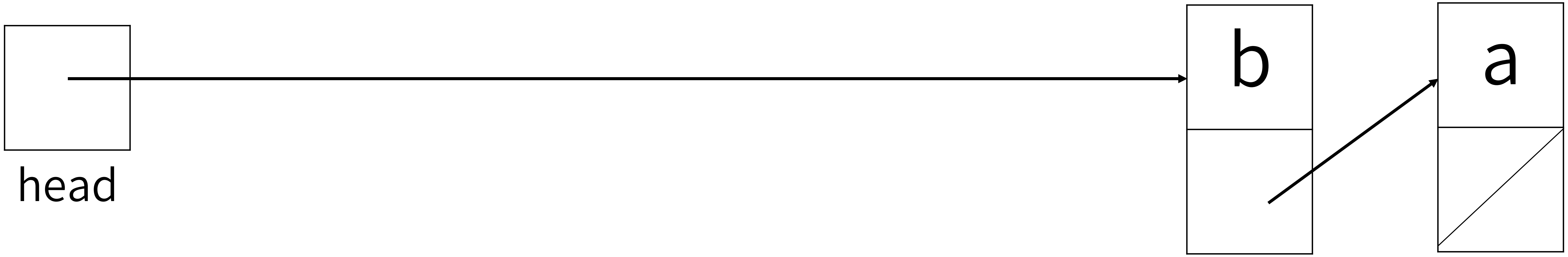


Stack Implementation with Linked List



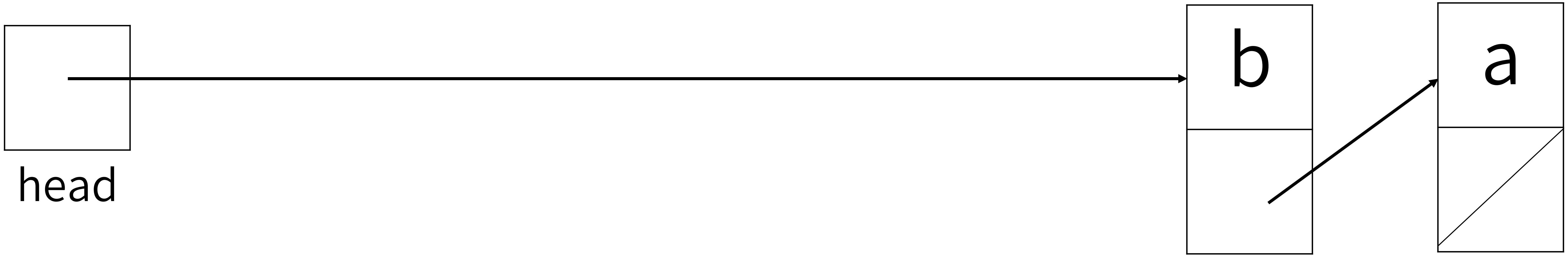
`Push(b)`

Stack Implementation with Linked List

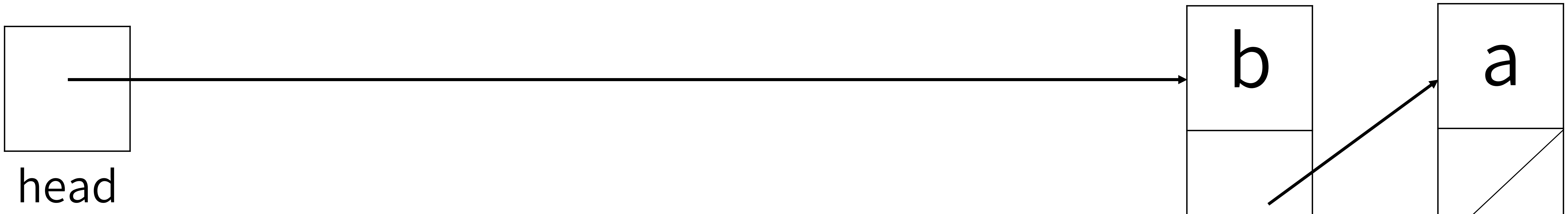


`Push(b)`

Stack Implementation with Linked List

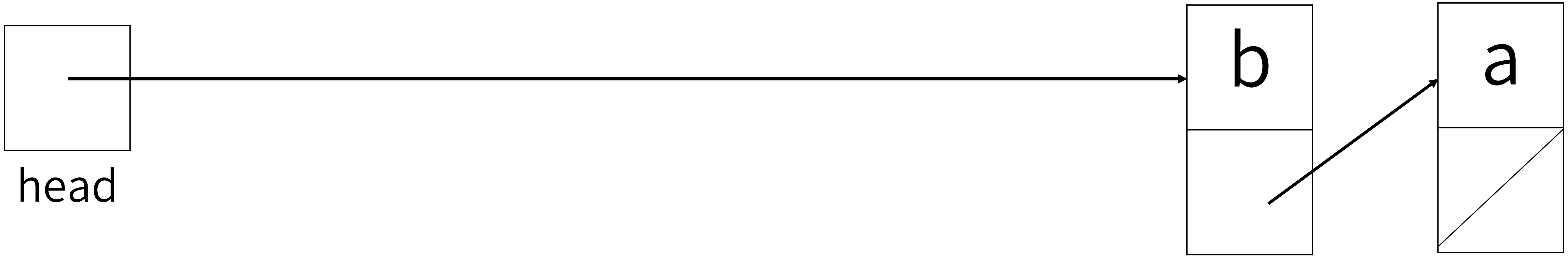


Stack Implementation with Linked List



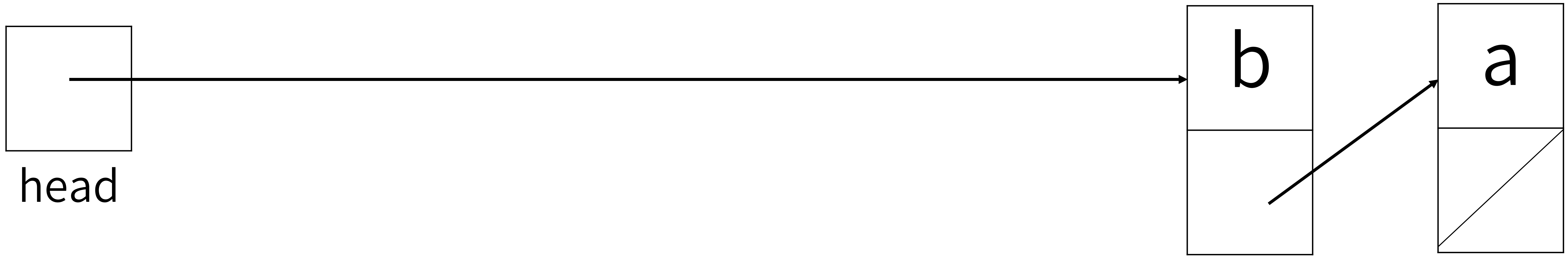
`Top()`

Stack Implementation with Linked List

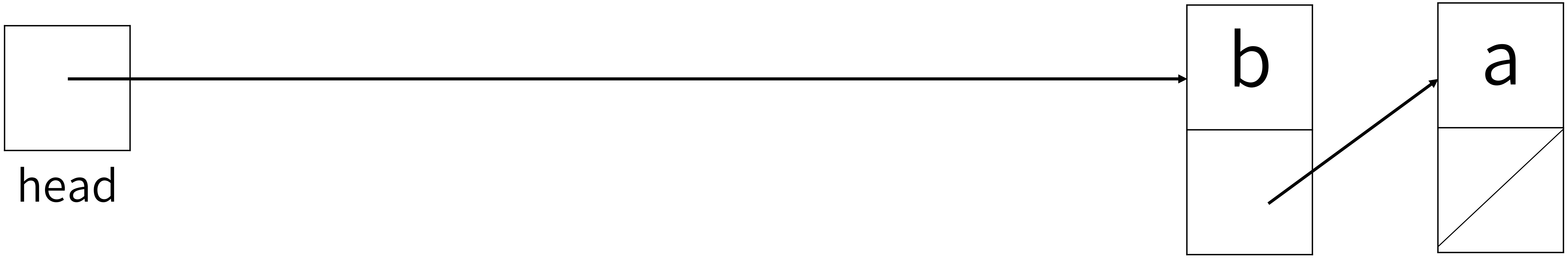


`Top()` → `b`

Stack Implementation with Linked List

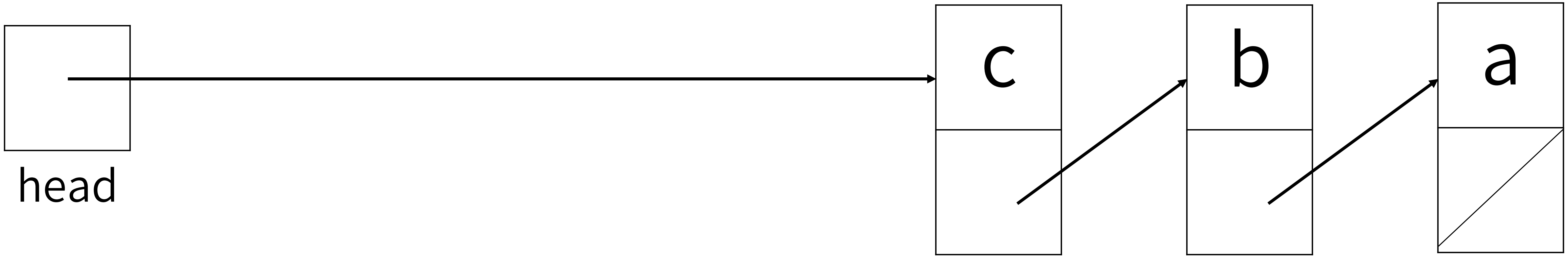


Stack Implementation with Linked List



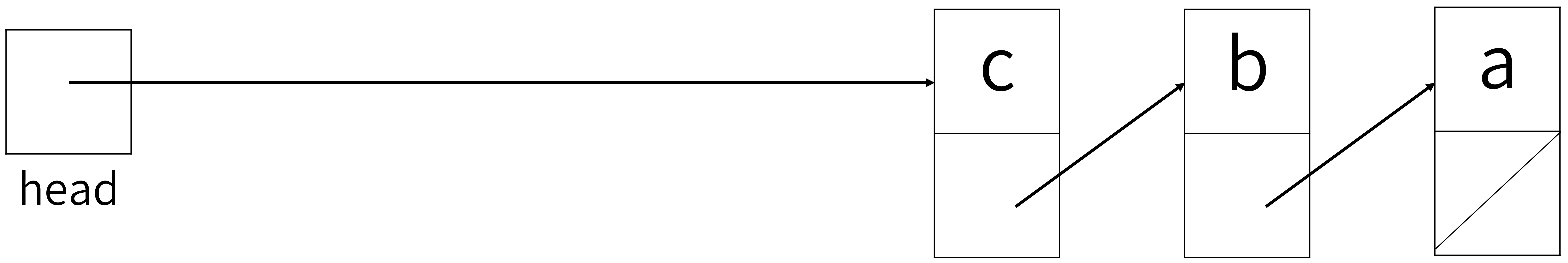
`Push(c)`

Stack Implementation with Linked List

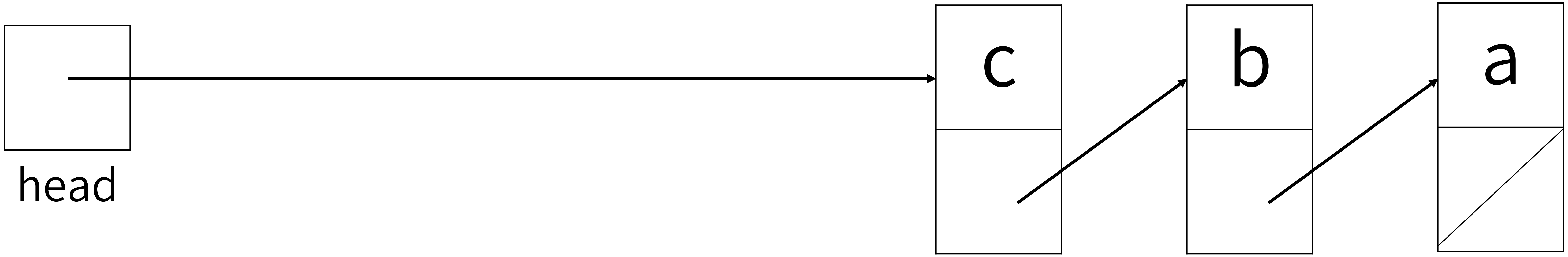


`Push(c)`

Stack Implementation with Linked List

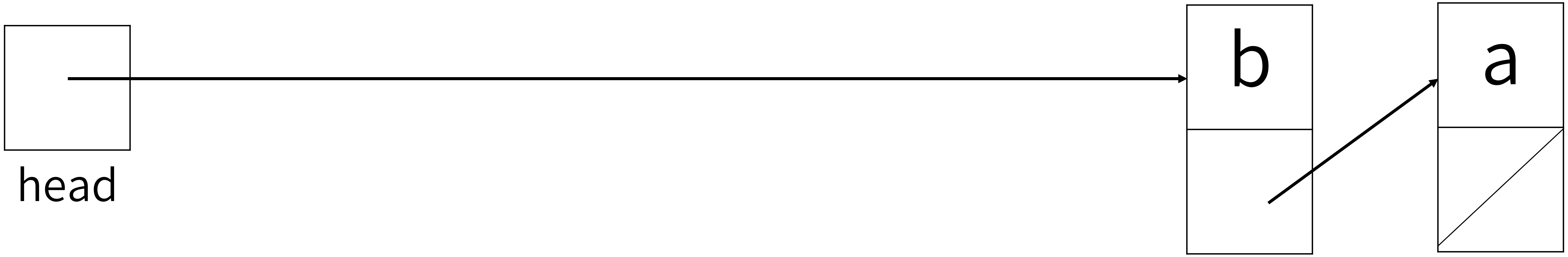


Stack Implementation with Linked List



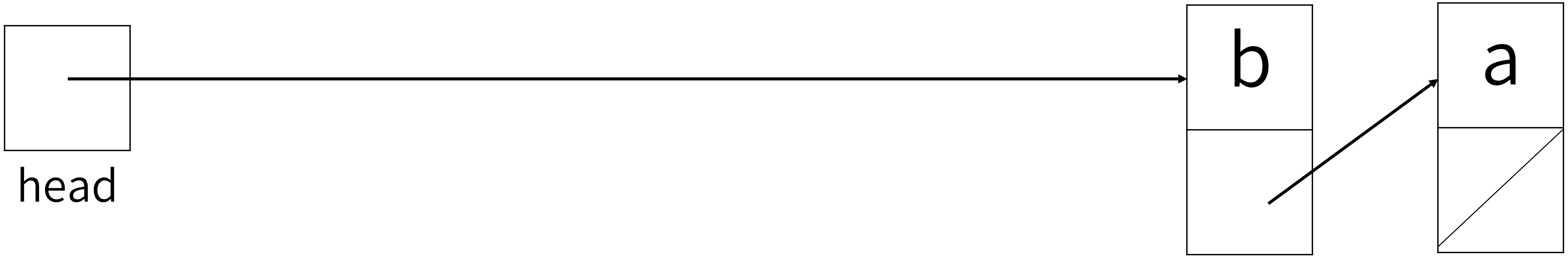
`Pop()`

Stack Implementation with Linked List



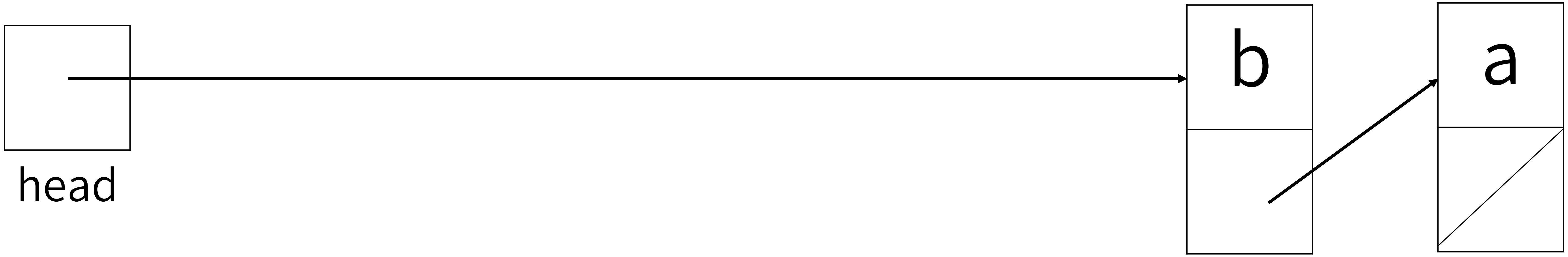
`Pop()`

Stack Implementation with Linked List

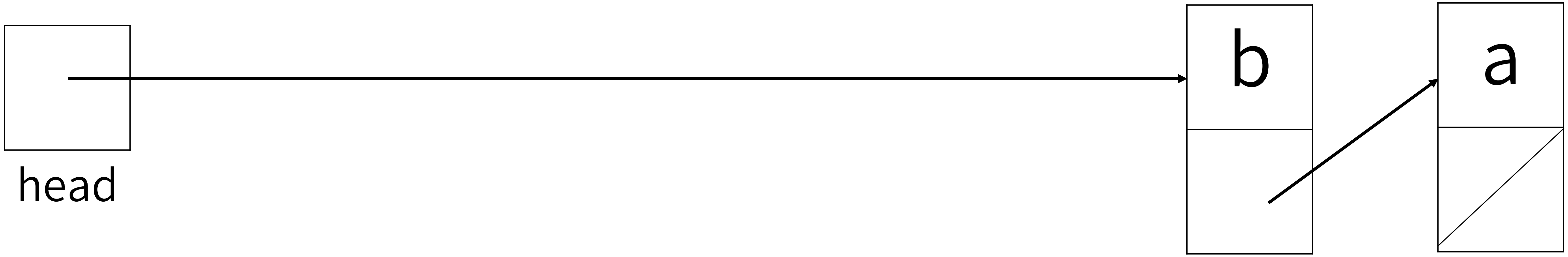


`Pop()→ c`

Stack Implementation with Linked List

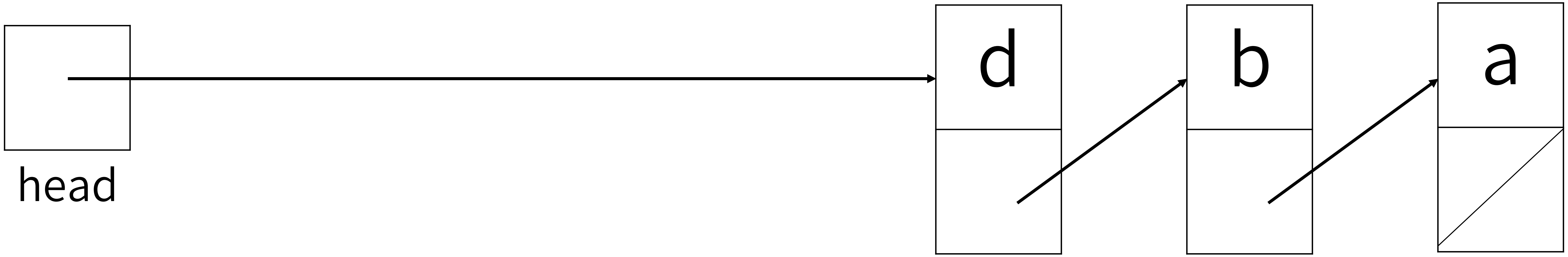


Stack Implementation with Linked List



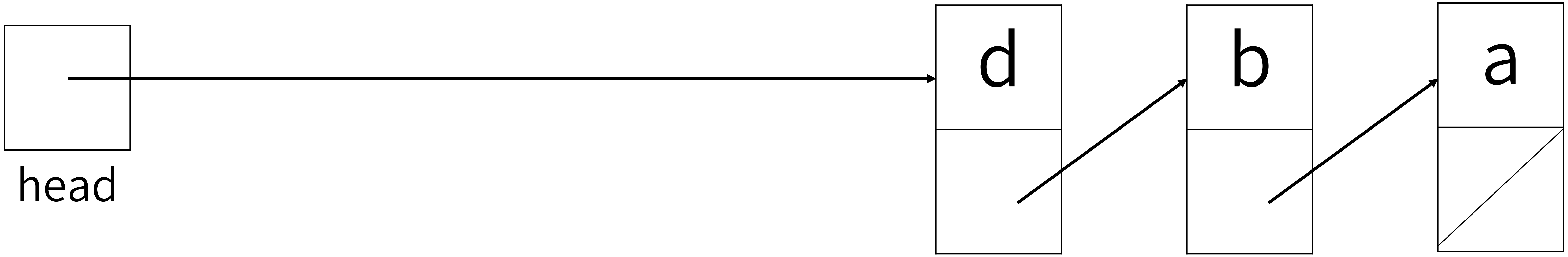
`Push(d)`

Stack Implementation with Linked List

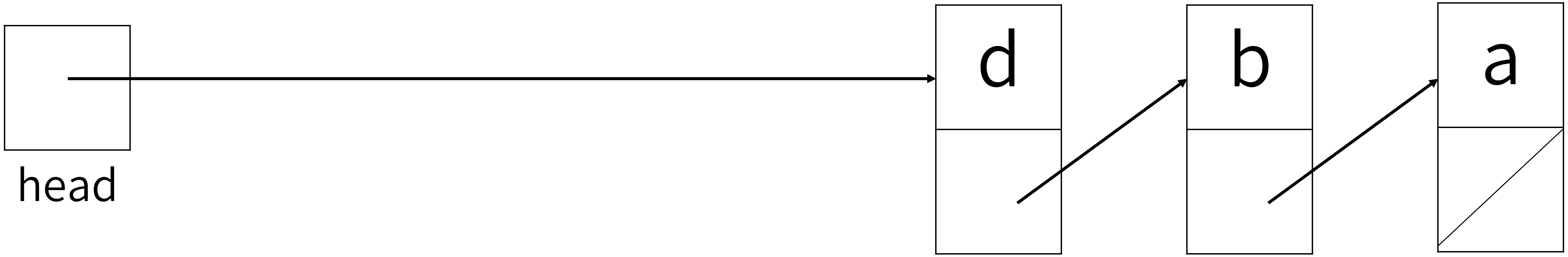


`Push(d)`

Stack Implementation with Linked List

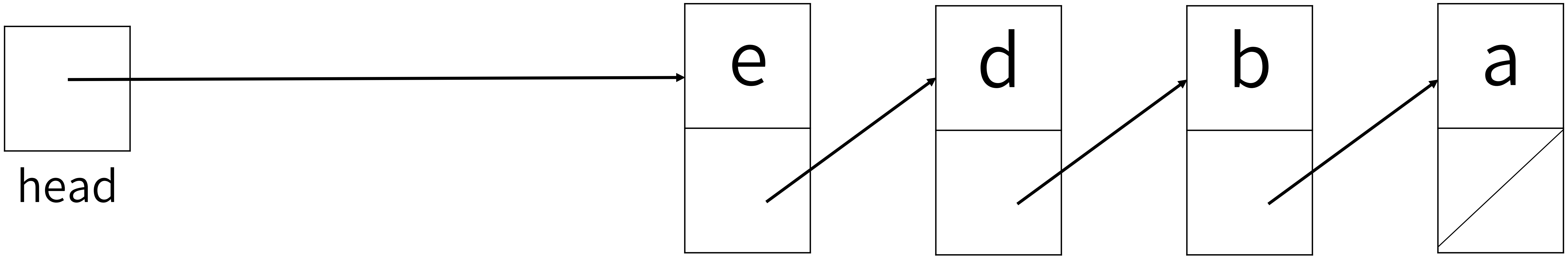


Stack Implementation with Linked List



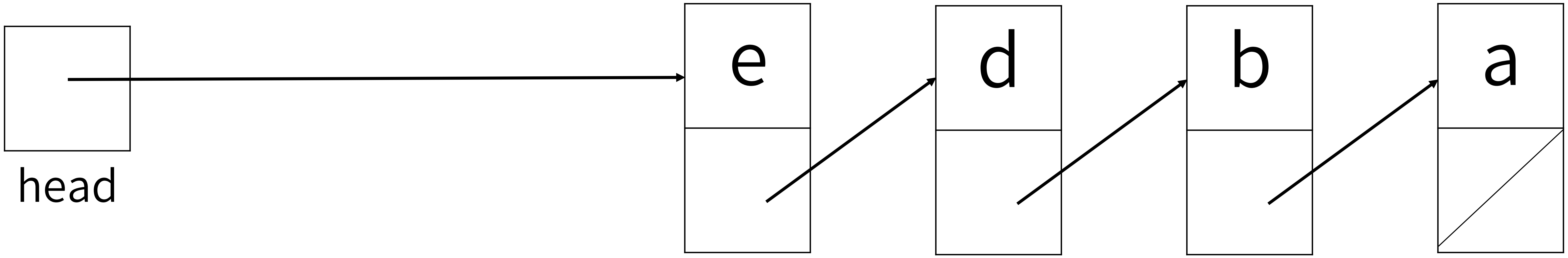
`Push(e)`

Stack Implementation with Linked List

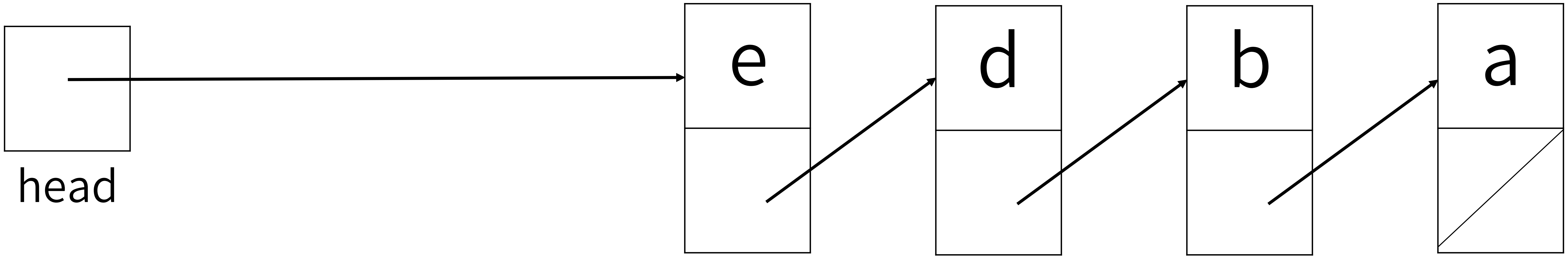


Push(e)

Stack Implementation with Linked List

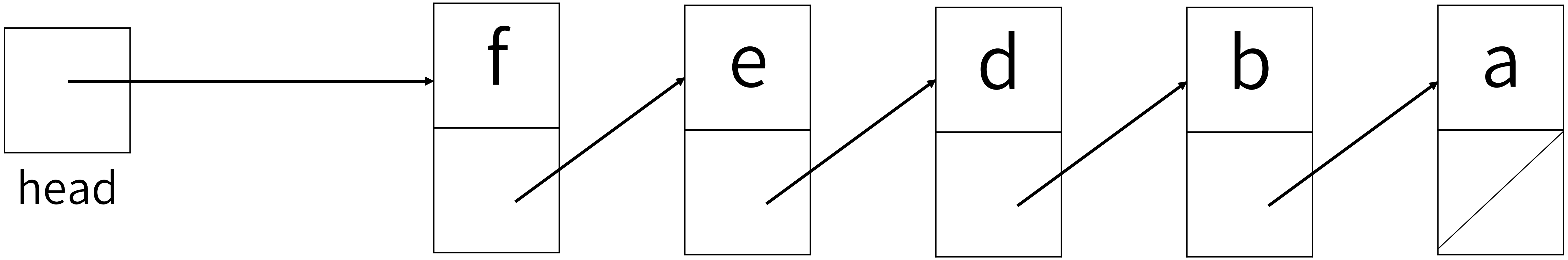


Stack Implementation with Linked List



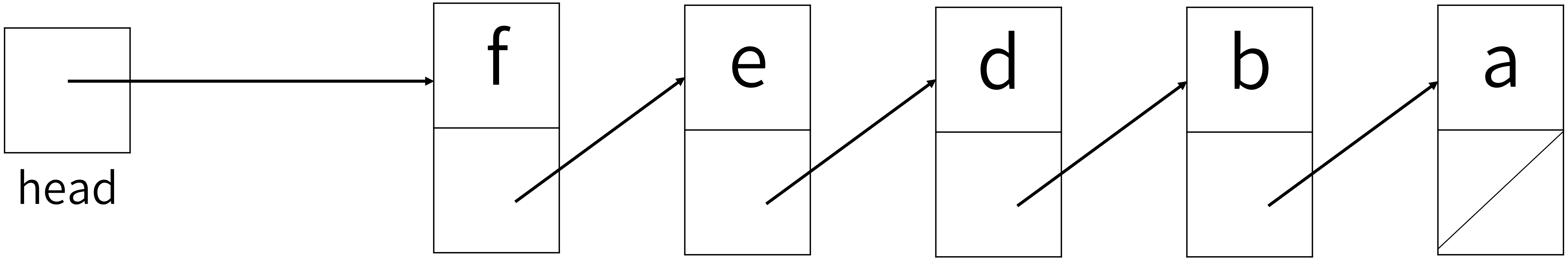
`Push(f)`

Stack Implementation with Linked List

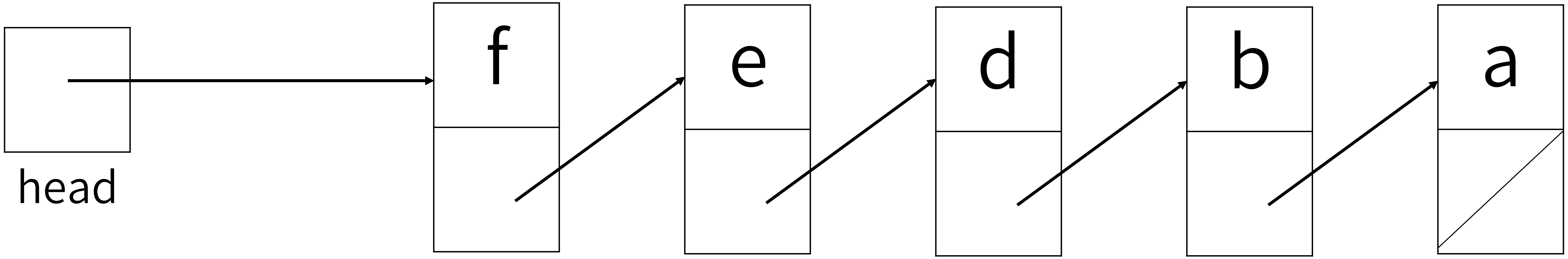


`Push(f)`

Stack Implementation with Linked List

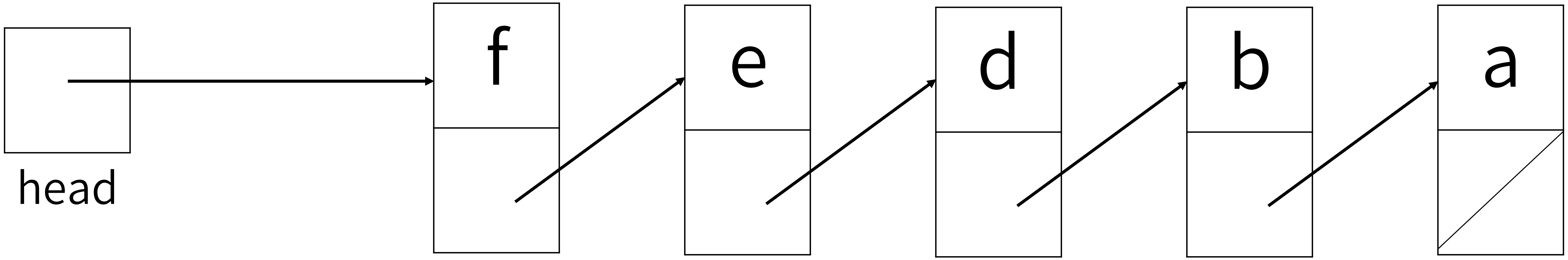


Stack Implementation with Linked List



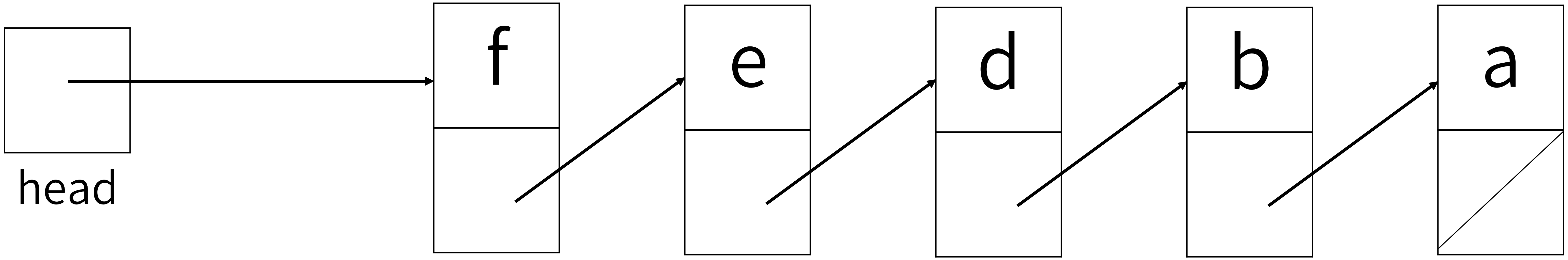
`Empty()`

Stack Implementation with Linked List



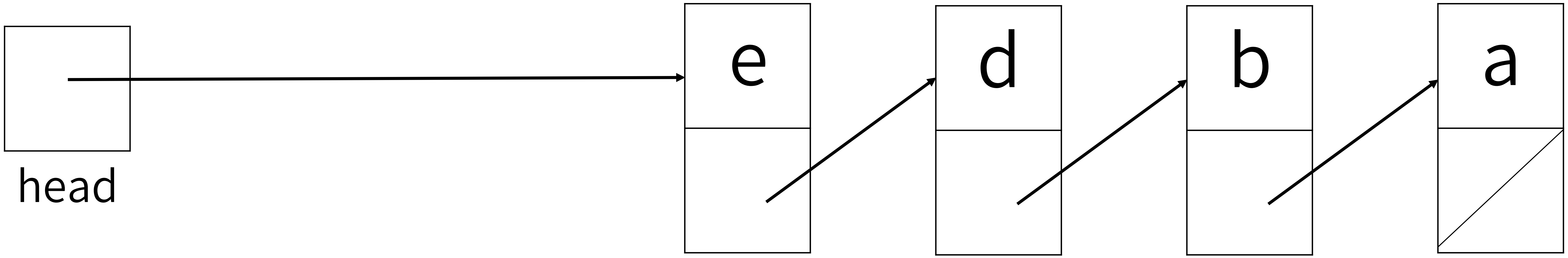
`Empty() → False`

Stack Implementation with Linked List



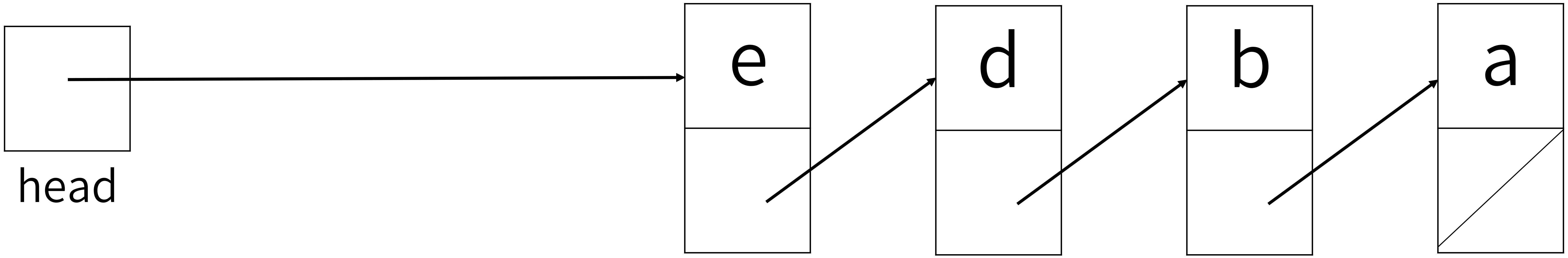
Pop()

Stack Implementation with Linked List

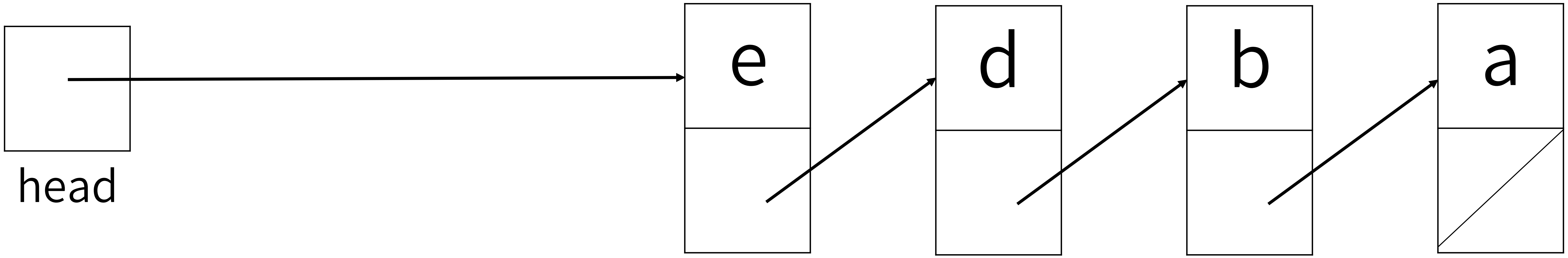


`Pop() → f`

Stack Implementation with Linked List

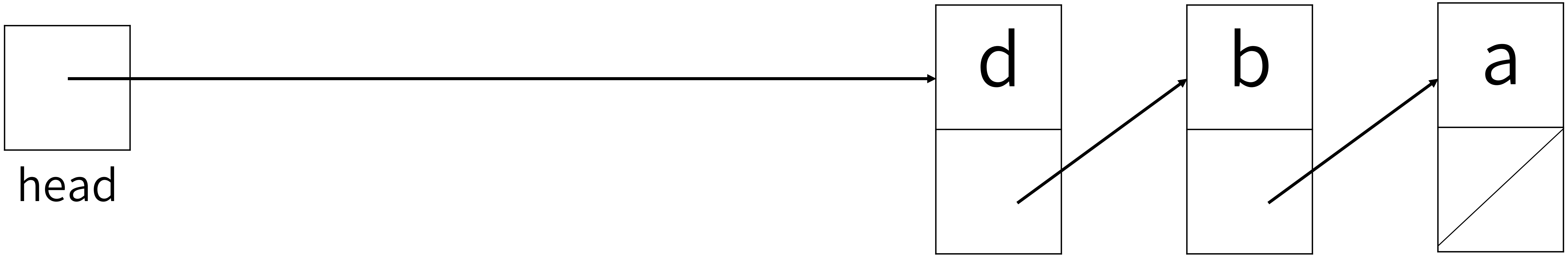


Stack Implementation with Linked List



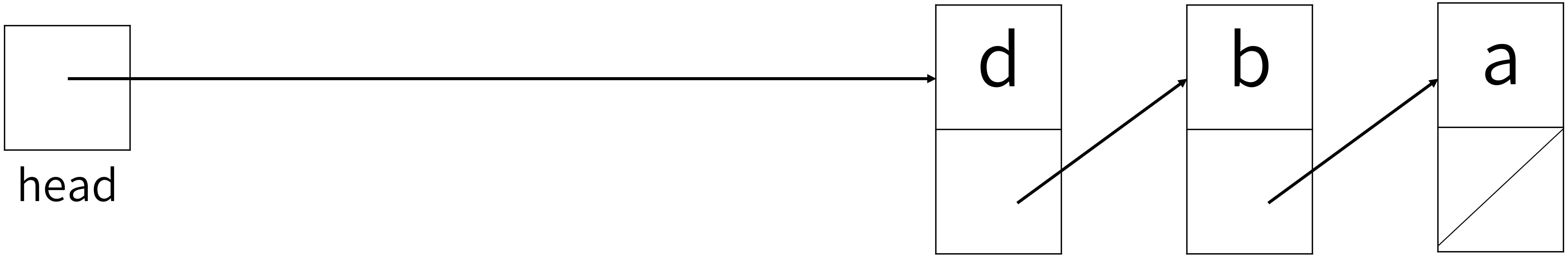
`Pop()`

Stack Implementation with Linked List

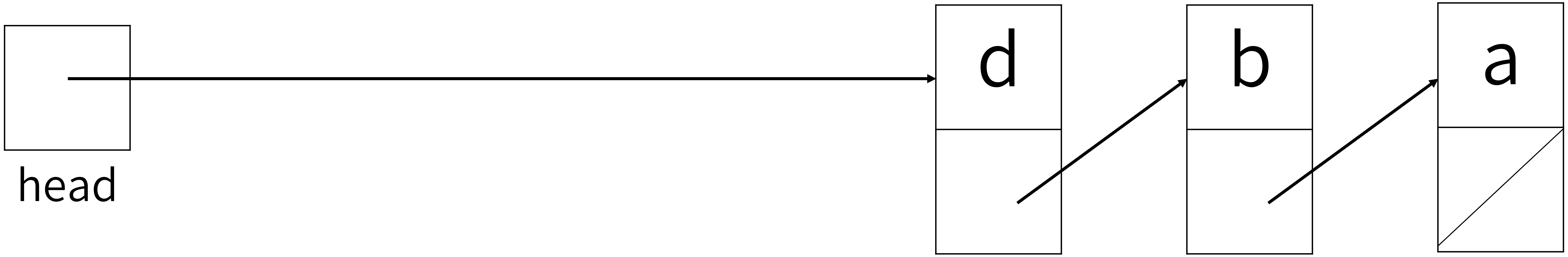


`Pop() → e`

Stack Implementation with Linked List

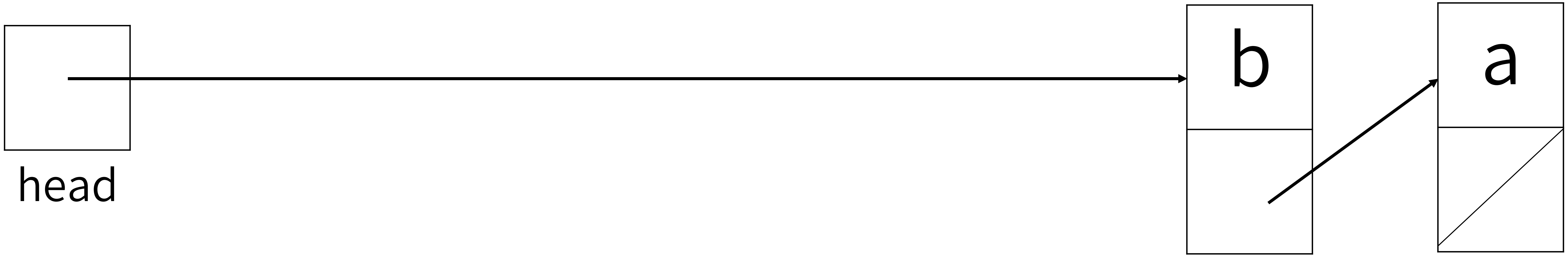


Stack Implementation with Linked List



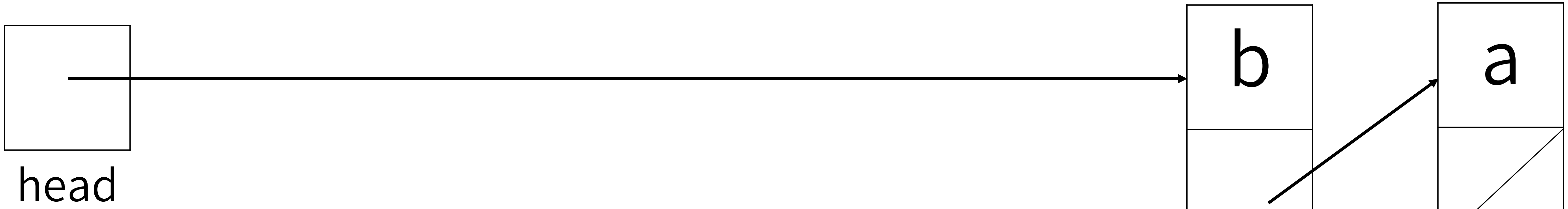
`Pop()`

Stack Implementation with Linked List

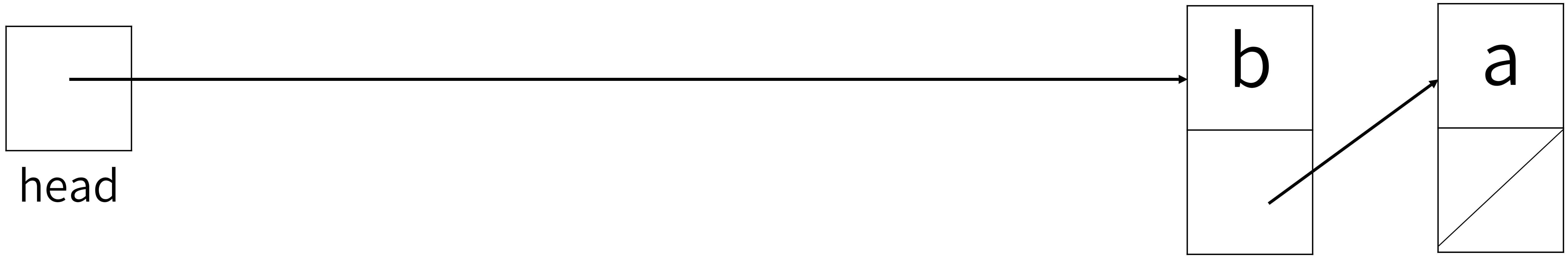


`Pop() → d`

Stack Implementation with Linked List

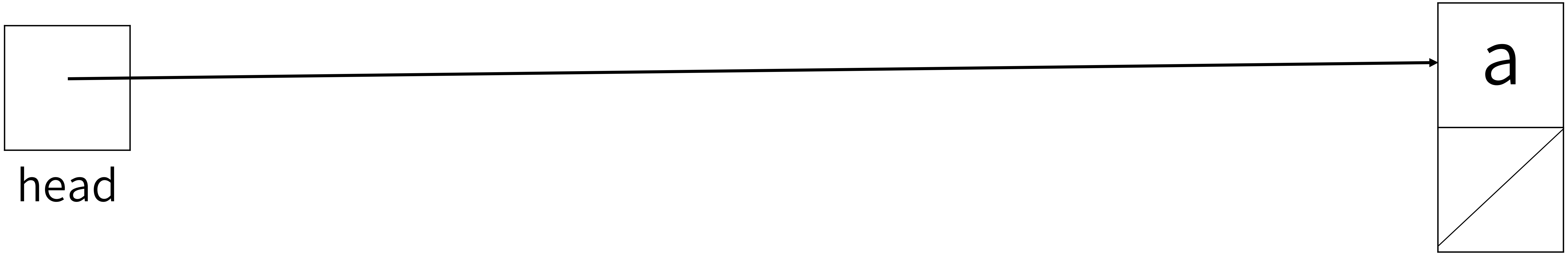


Stack Implementation with Linked List



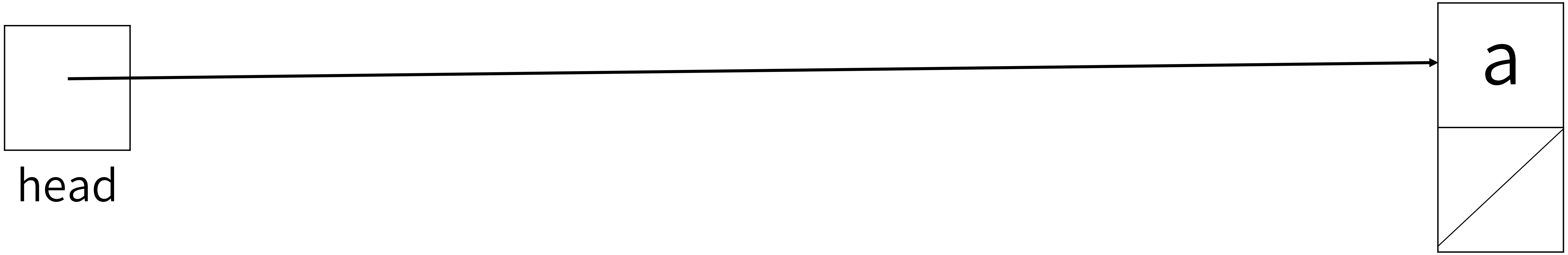
`Pop()`

Stack Implementation with Linked List



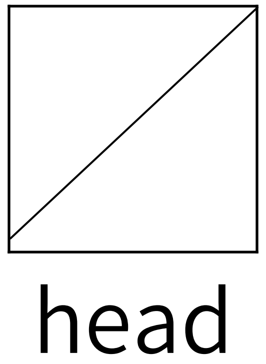
`Pop() → b`

Stack Implementation with Linked List



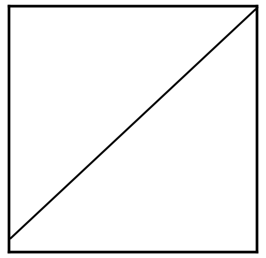
`Pop()`

Stack Implementation with Linked List



`Pop()→ a`

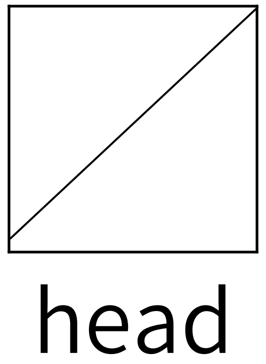
Stack Implementation with Linked List



head

```
Empty()
```

Stack Implementation with Linked List



`Empty() → True`

Summary

- Stacks can be implemented with either an array or a linked list.
- Each stack operation is O(1):Push, Pop,Top,Empty.
- Implementation with linked list:
  - `Push : PushFront()`
  - `Top : TopFront()`
  - `Pop : TopFront() + PopFront()`
- Stacks are occasionally known as LIFO queues.

ADT - Queue

Queue

Queue

An abstract data type in which element are added to the rear and remove from the front; a “first in, first out” (FIFO) structure.

A queue is a first-in, first-out (FIFO) data structure that operates like the line at your favorite coffee shop: we add new elements at the back of the queue and remove old elements from the front.

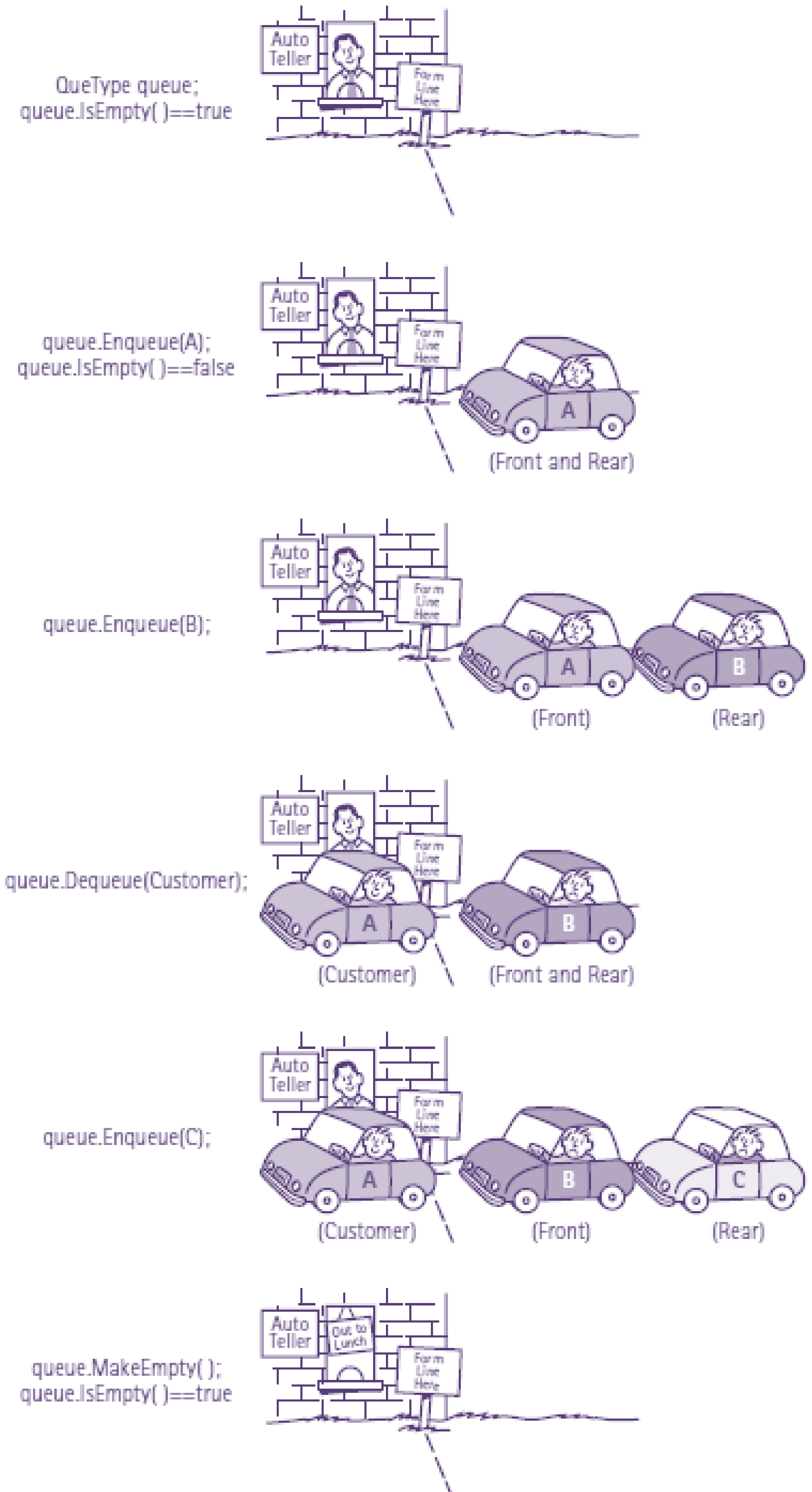
A list data structure in which elements are inserted at one end and removed from the other end.

Queue Operations

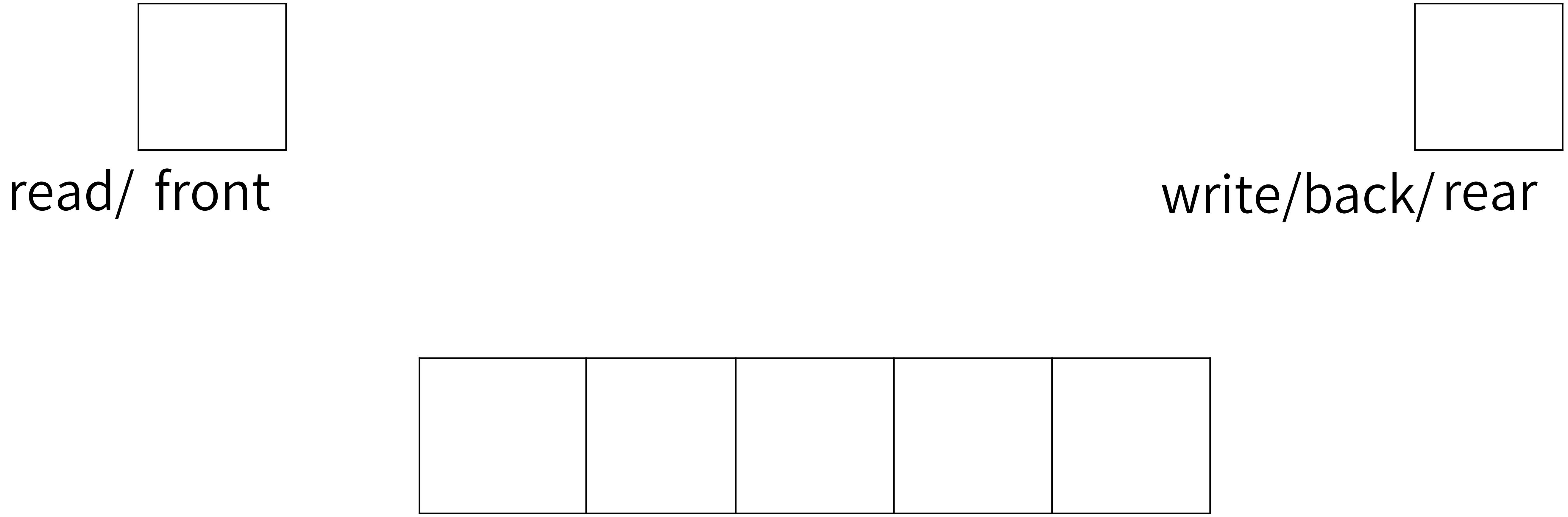
Enqueue(key) Add a new element to the back of the queue.

Dequeue(key) Remove the element from the front of the queue and return it.

IsEmpty() Check whether there are any elements



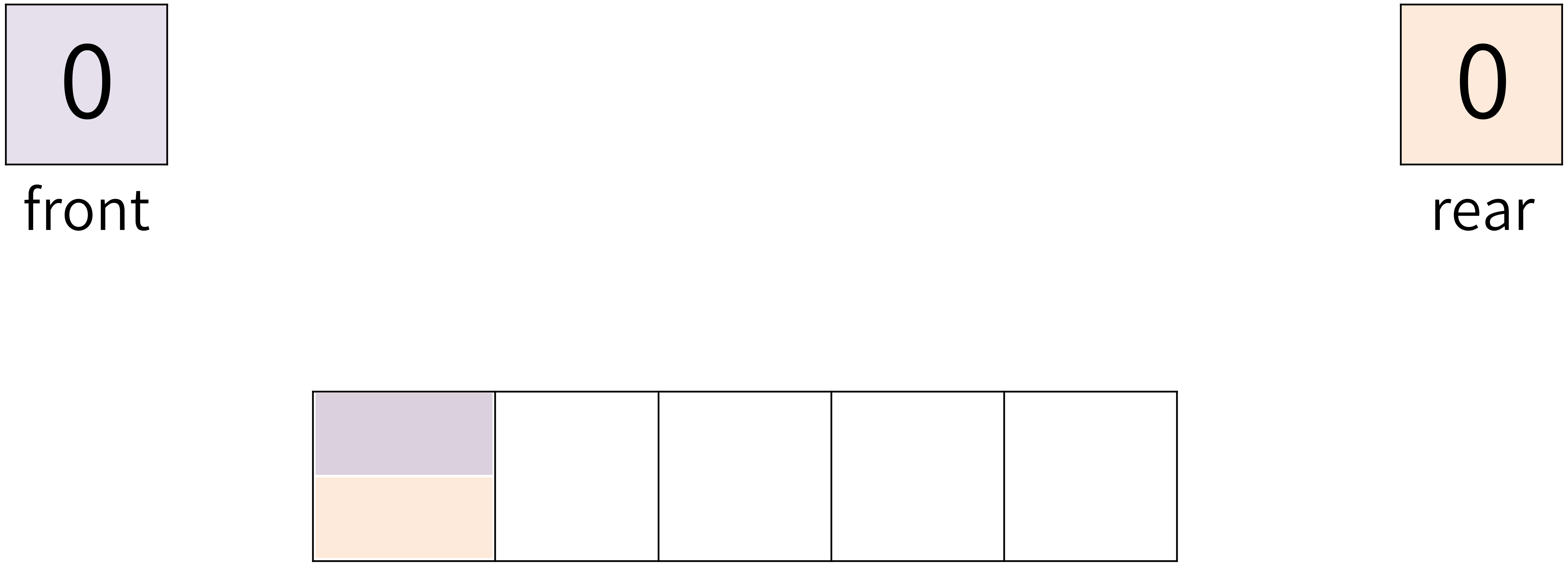
Queue Implementation with Arrays



read/ front

write/back/ rear

Queue Implementation with Arrays



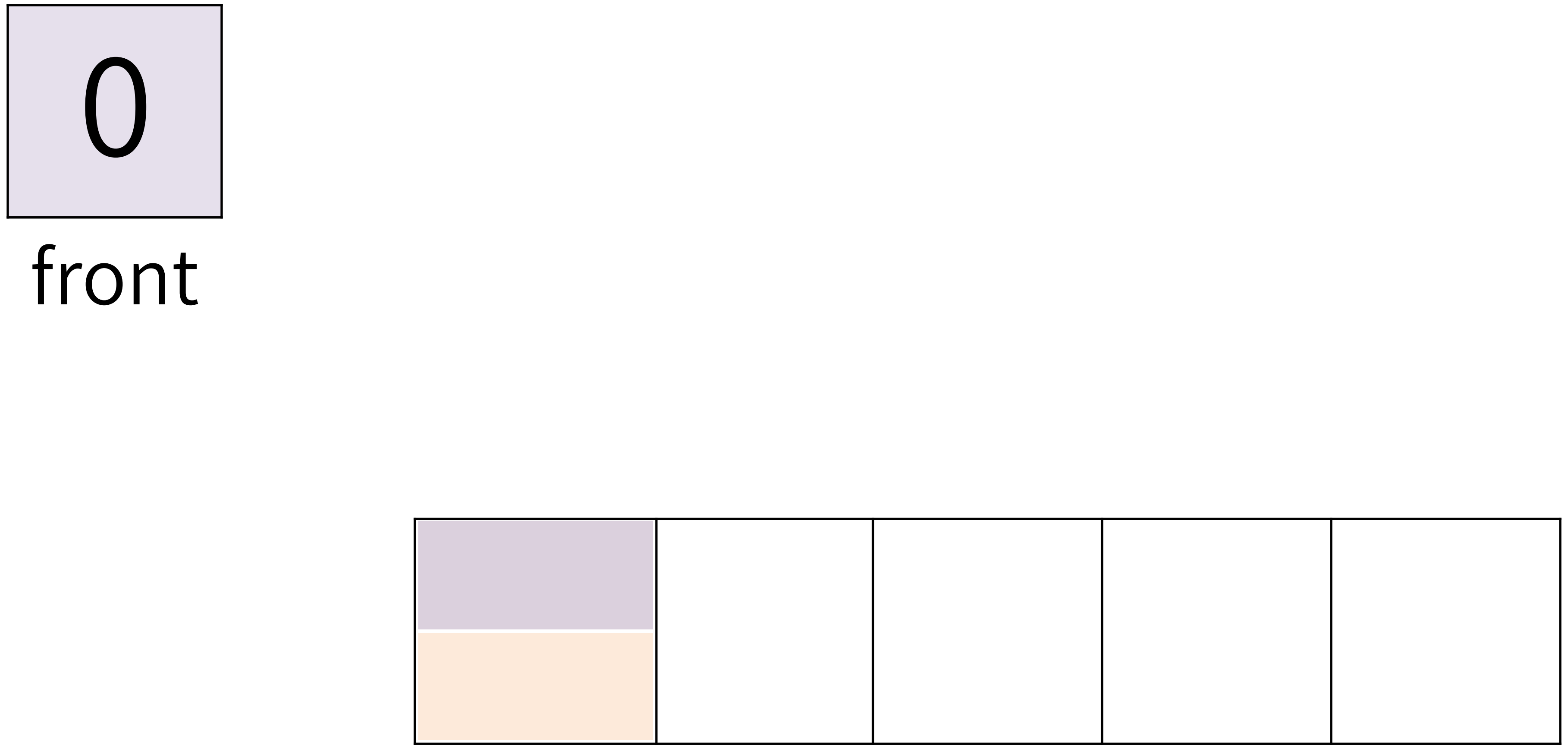
0

front

0

rear

Queue Implementation with Arrays



0

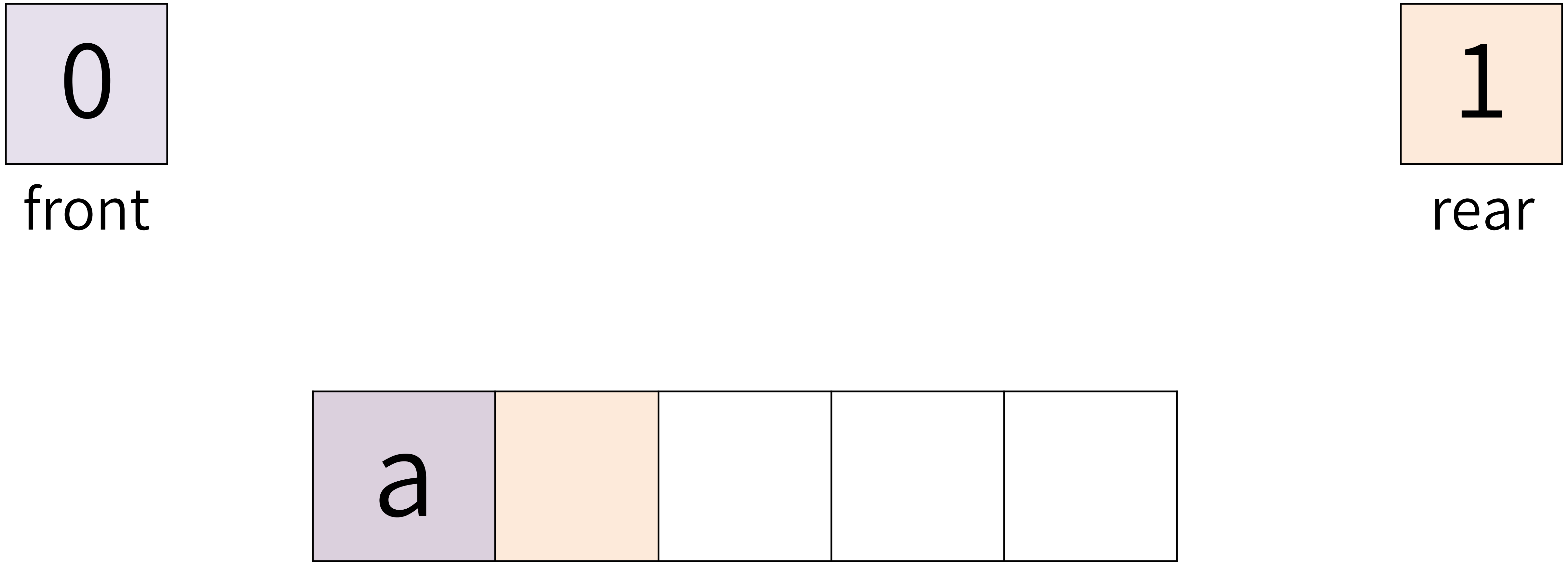
front

0

rear

`Enqueue(a)`

Queue Implementation with Arrays



0

front

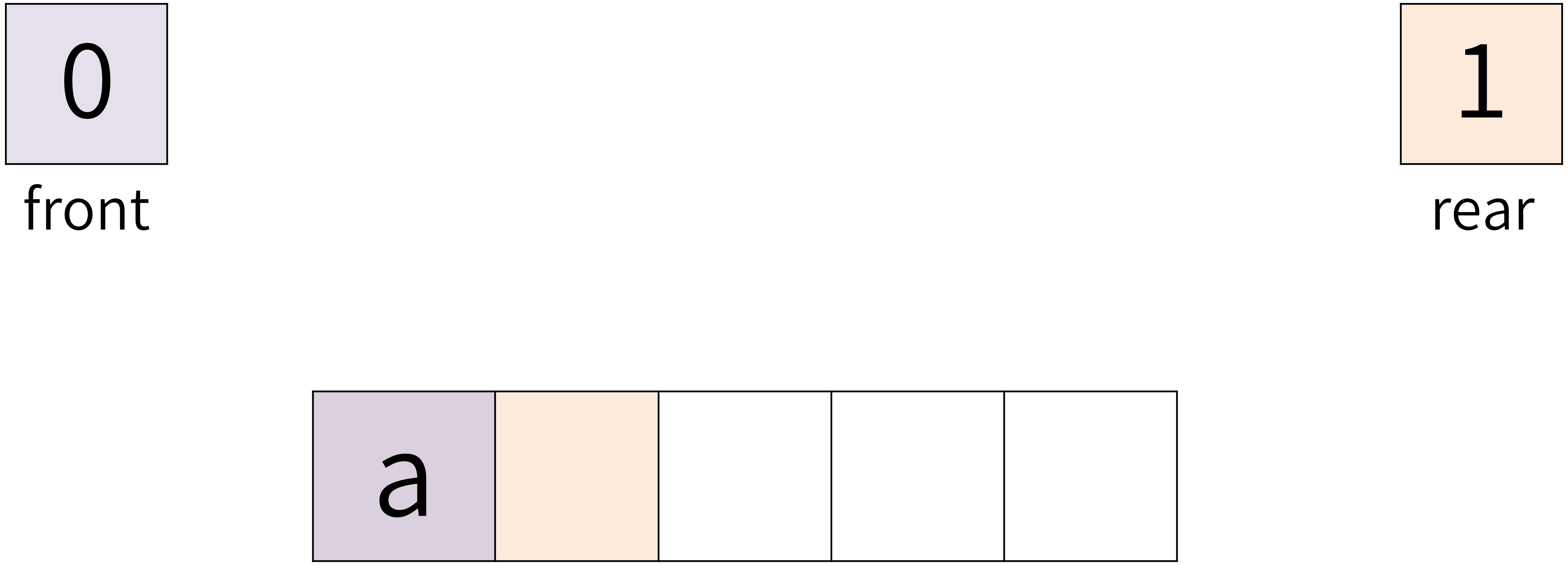
1

rear

a				

```
Enqueue(a)
```

Queue Implementation with Arrays



0

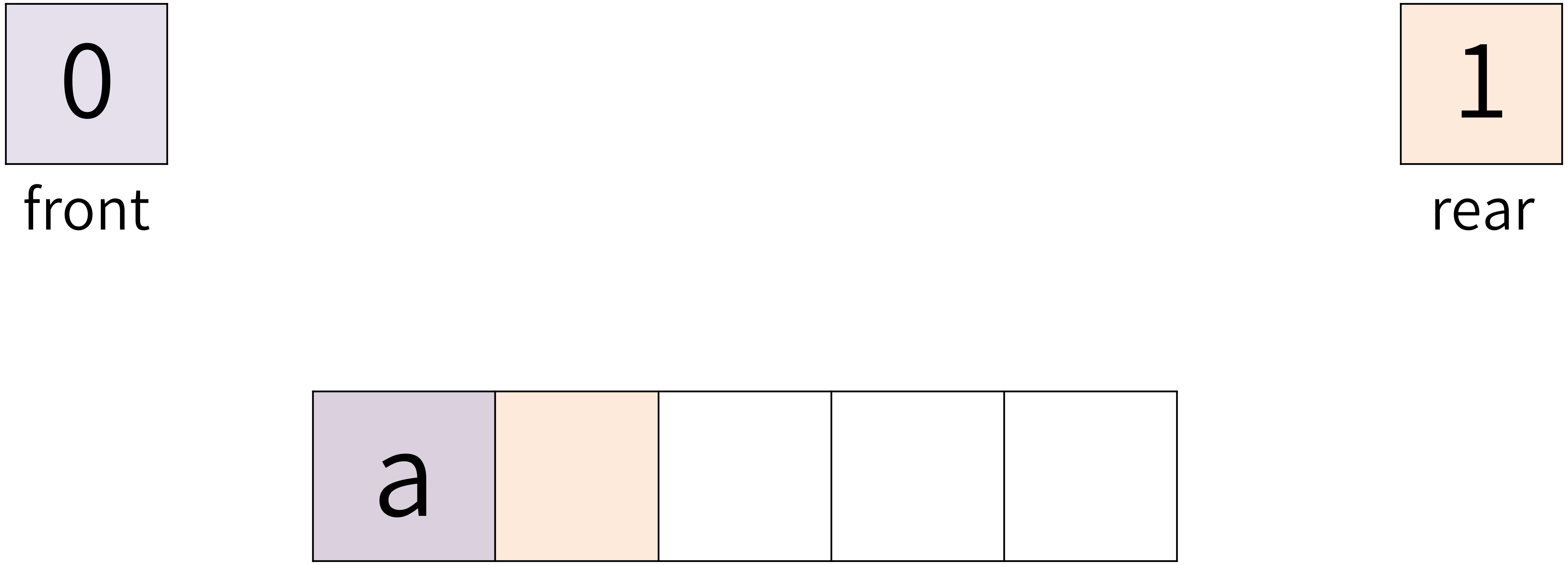
front

1

rear

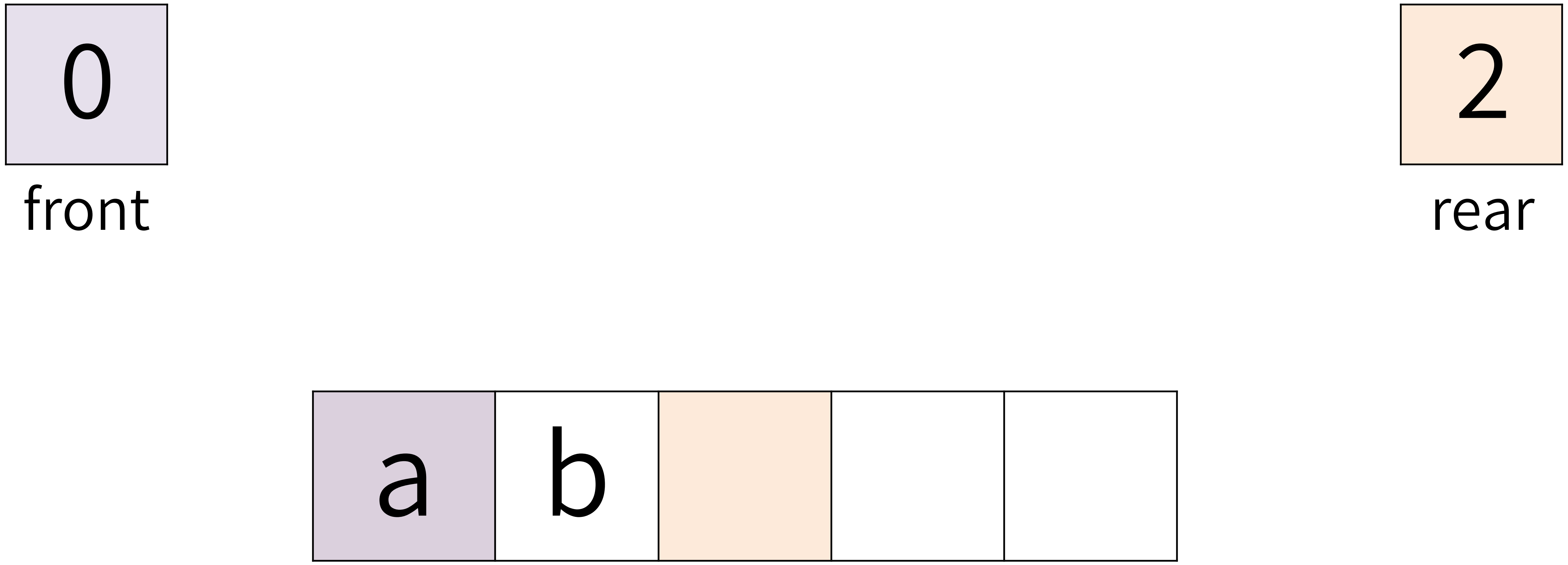
a

Queue Implementation with Arrays



`Enqueue(b)`

Queue Implementation with Arrays



0

front

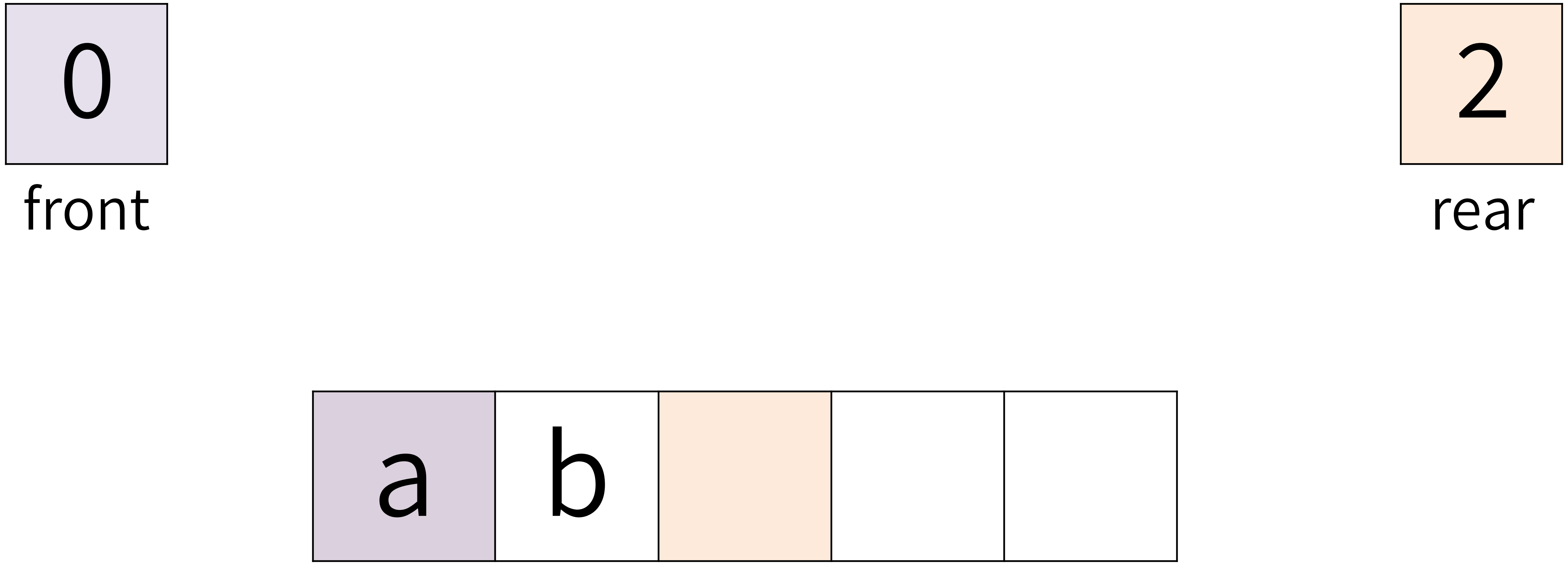
2

rear

a	b			

`Enqueue(b)`

Queue Implementation with Arrays



0

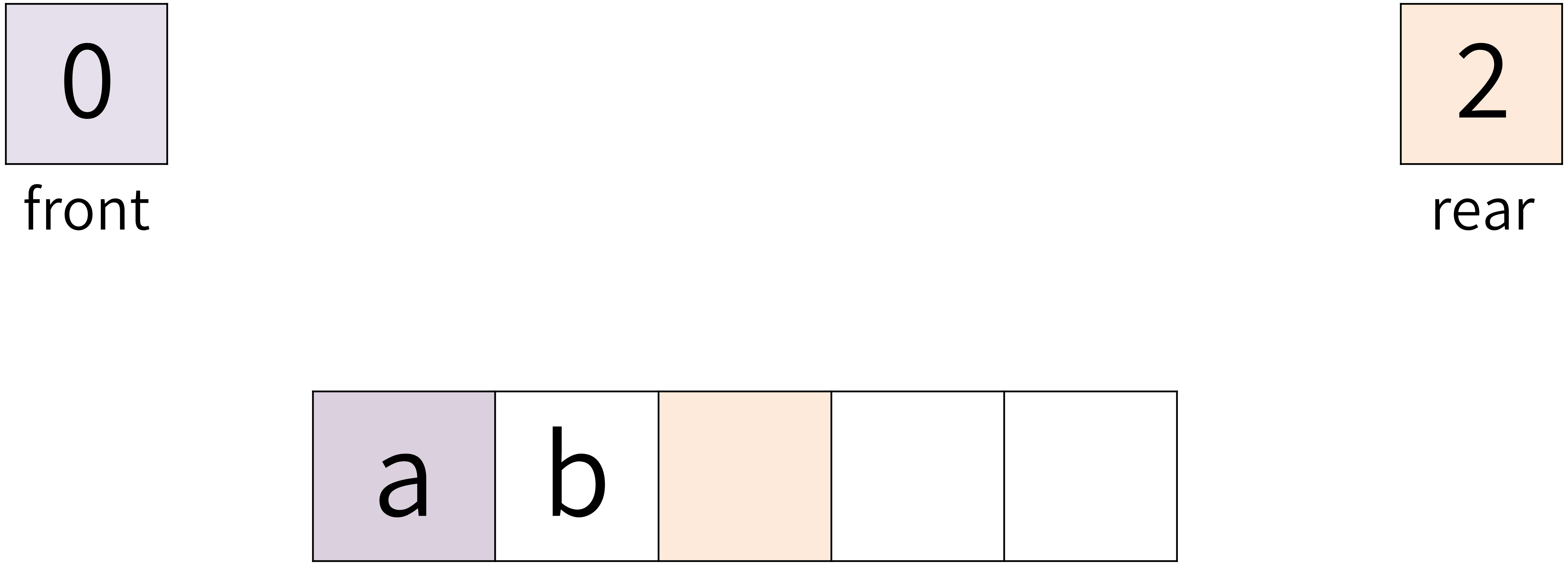
front

2

rear

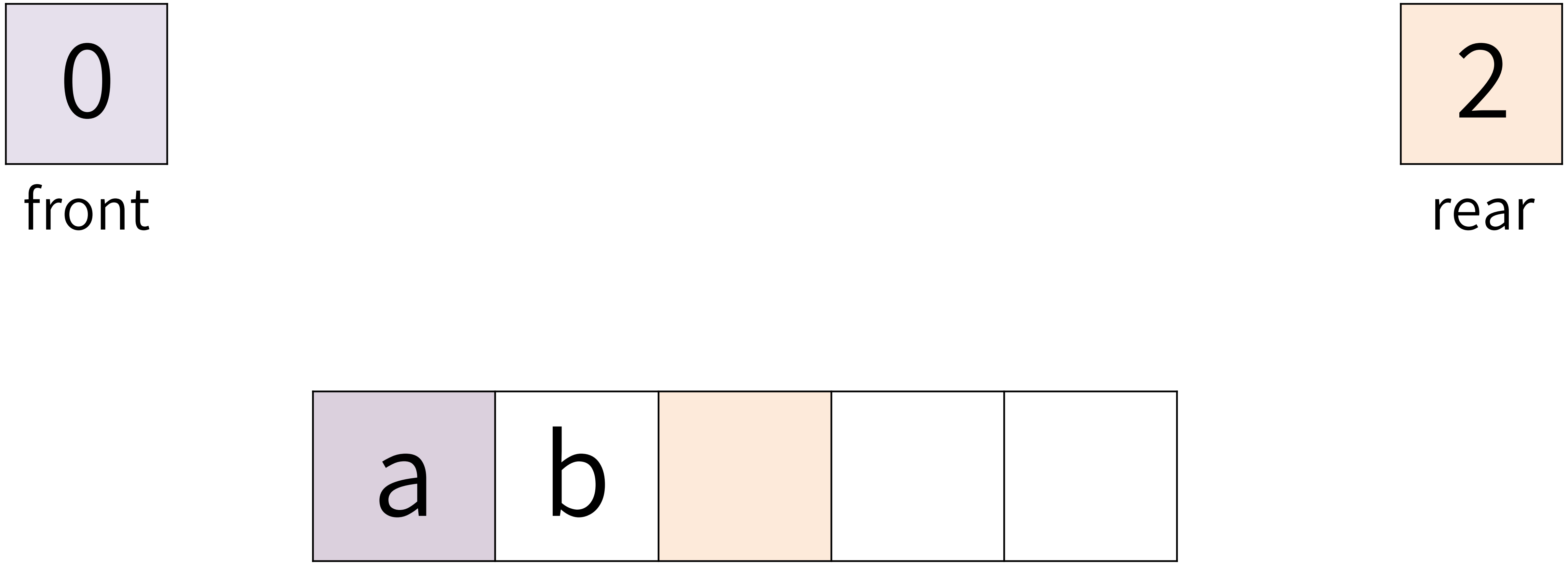
a	b			

Queue Implementation with Arrays



`Empty()`

Queue Implementation with Arrays



0

front

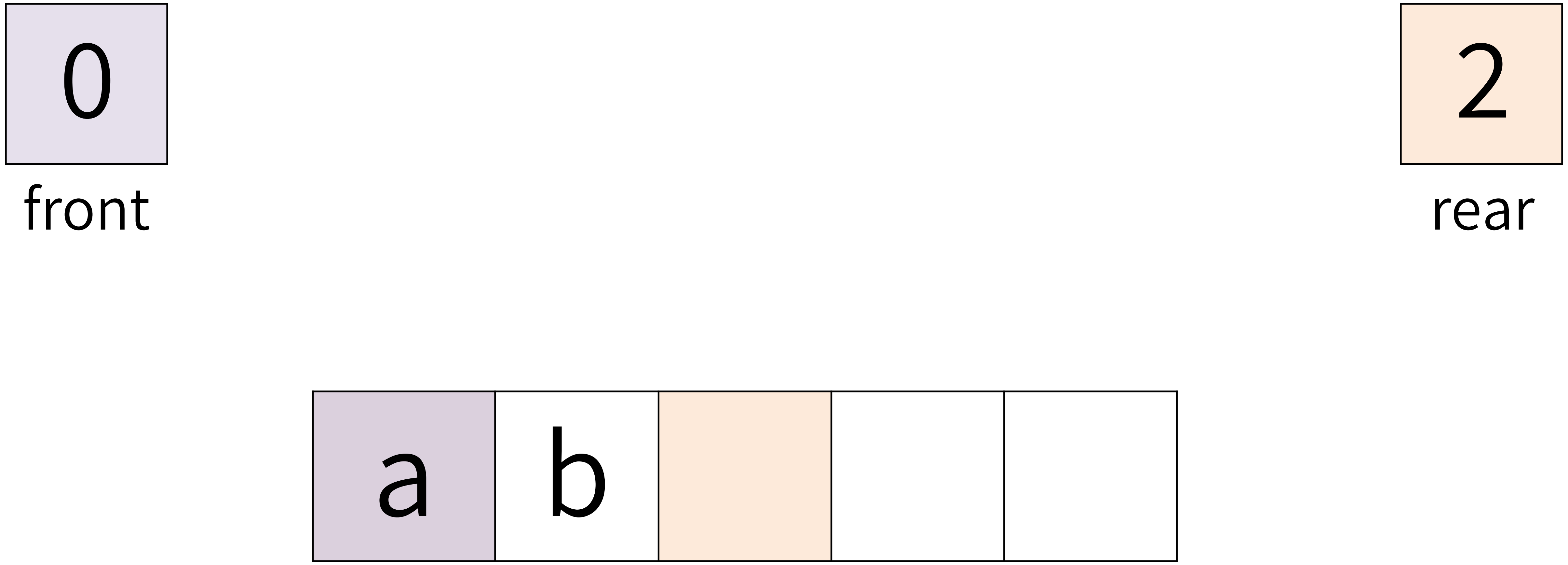
2

rear

a	b			

`Empty()→ False`

Queue Implementation with Arrays



0

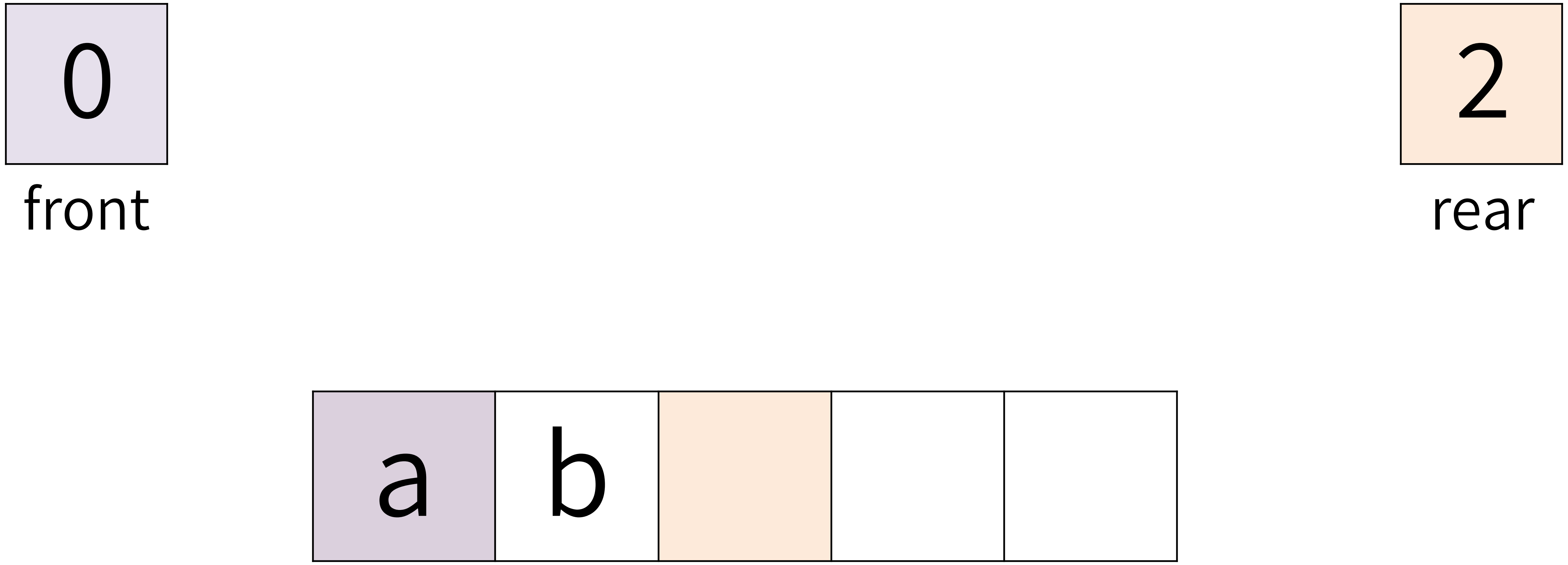
front

2

rear

a	b			

Queue Implementation with Arrays



0

front

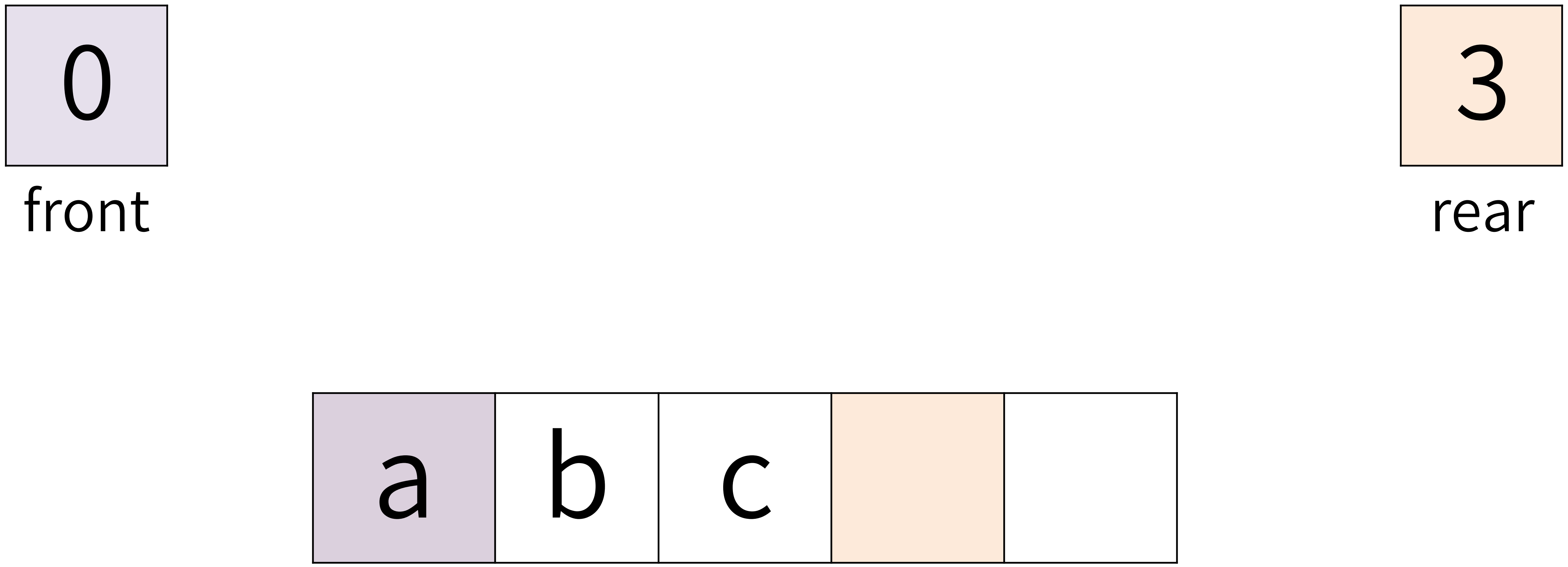
2

rear

a	b			

`Enqueue(c)`

Queue Implementation with Arrays



0

front

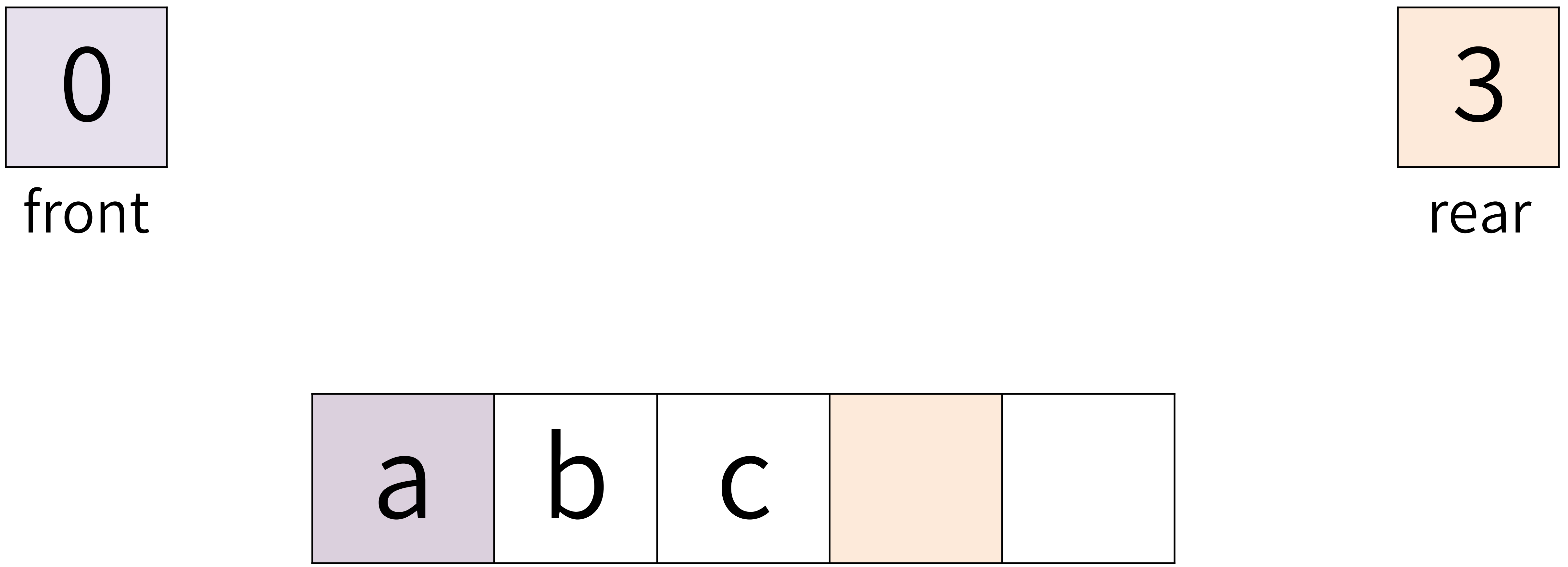
3

rear

a	b	c		

`Enqueue(c)`

Queue Implementation with Arrays



0

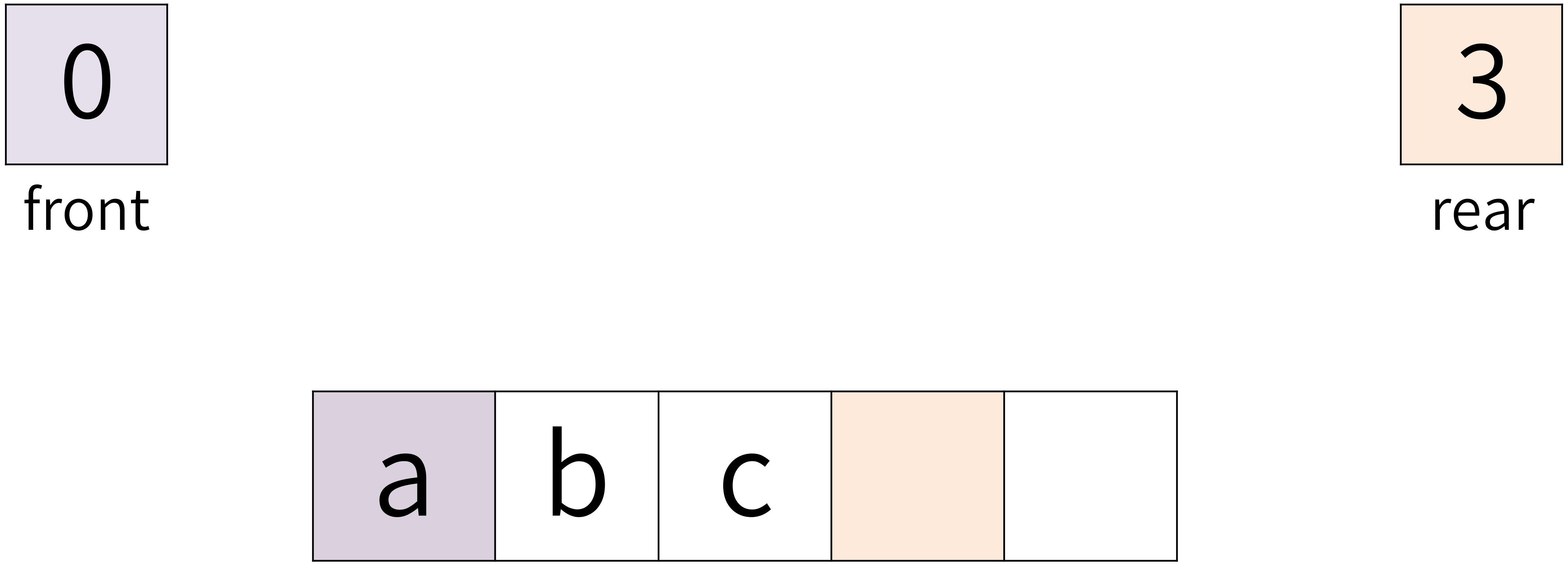
front

3

rear

a	b	c		

Queue Implementation with Arrays



0

front

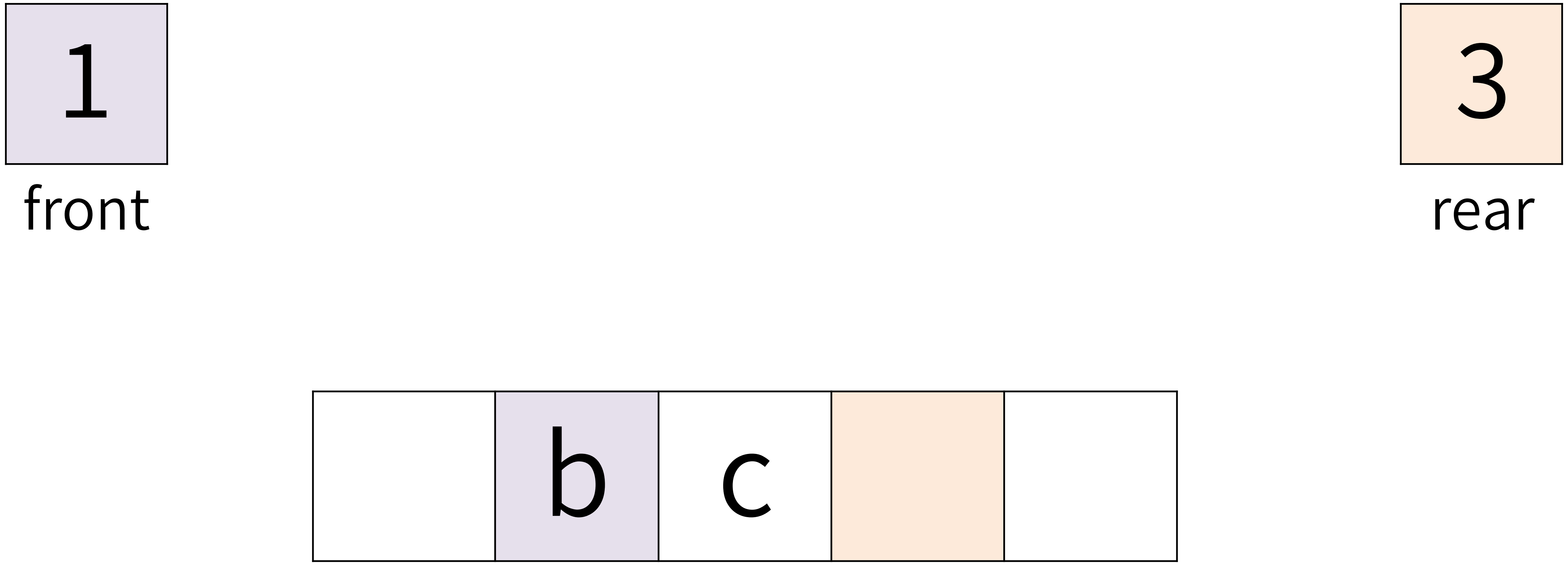
3

rear

a	b	c		

```
Dequeue()
```

Queue Implementation with Arrays



1

front

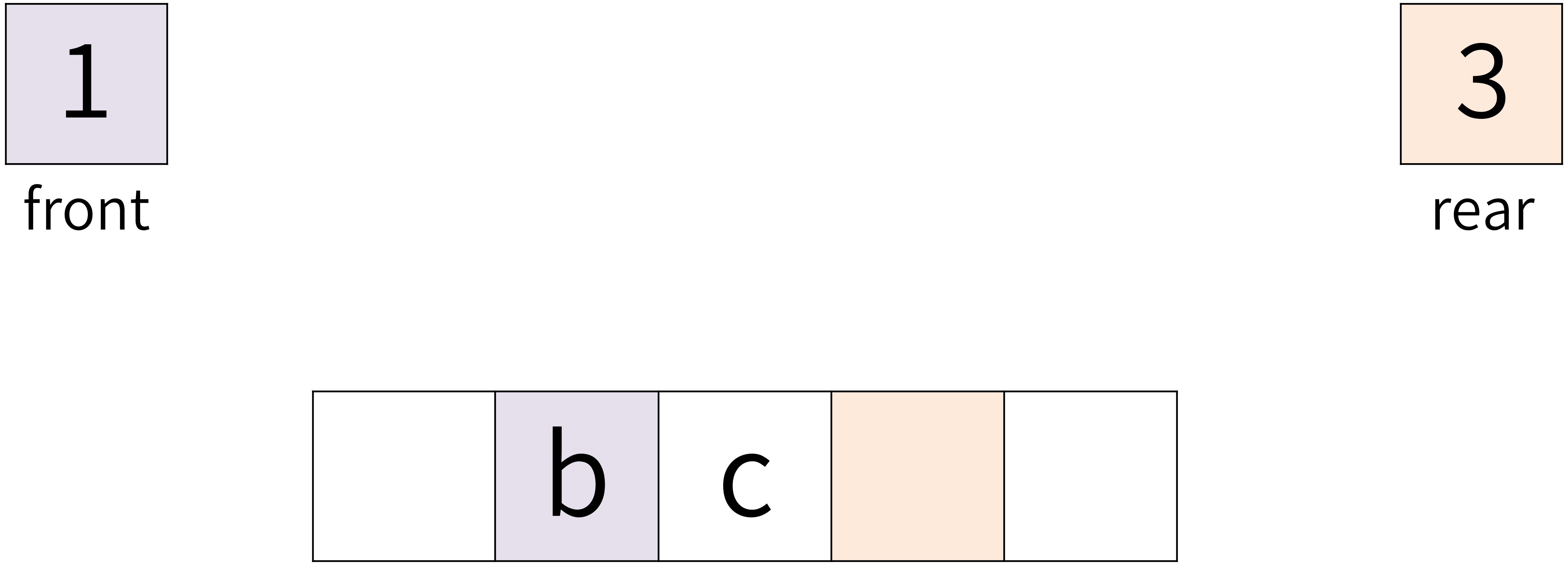
3

rear

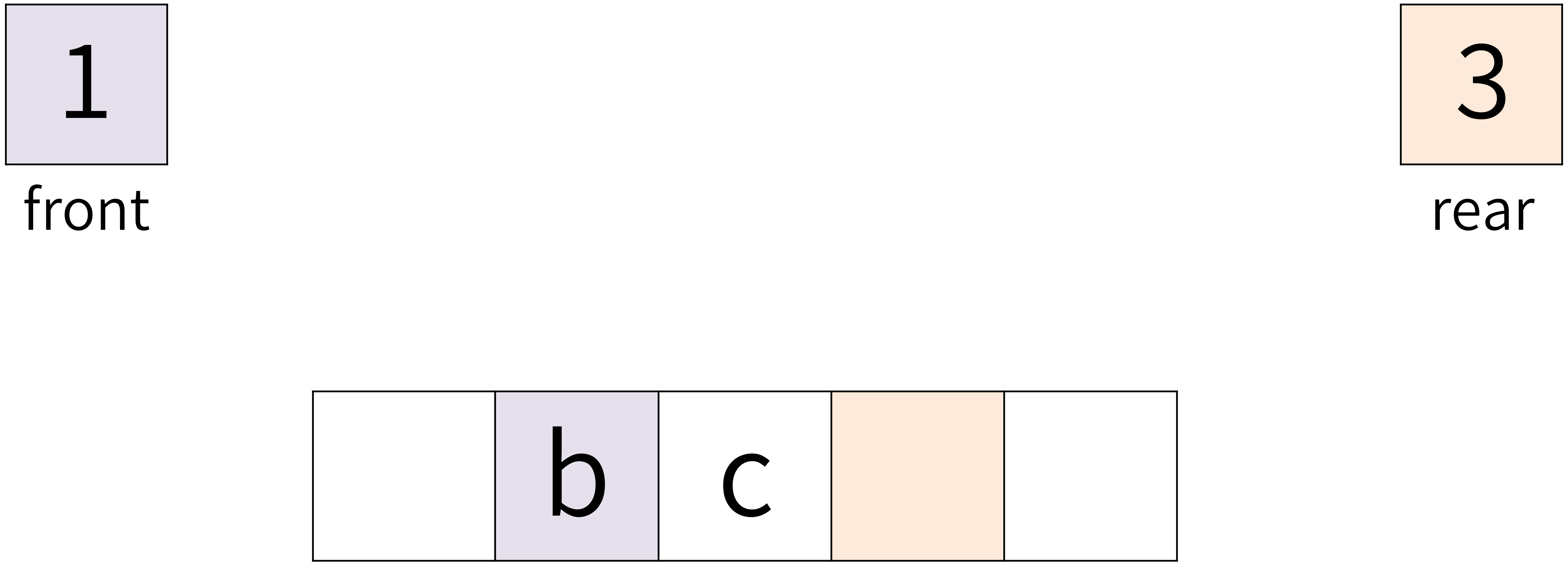
	b	c		

Dequeue() →a

Queue Implementation with Arrays



Queue Implementation with Arrays



1

front

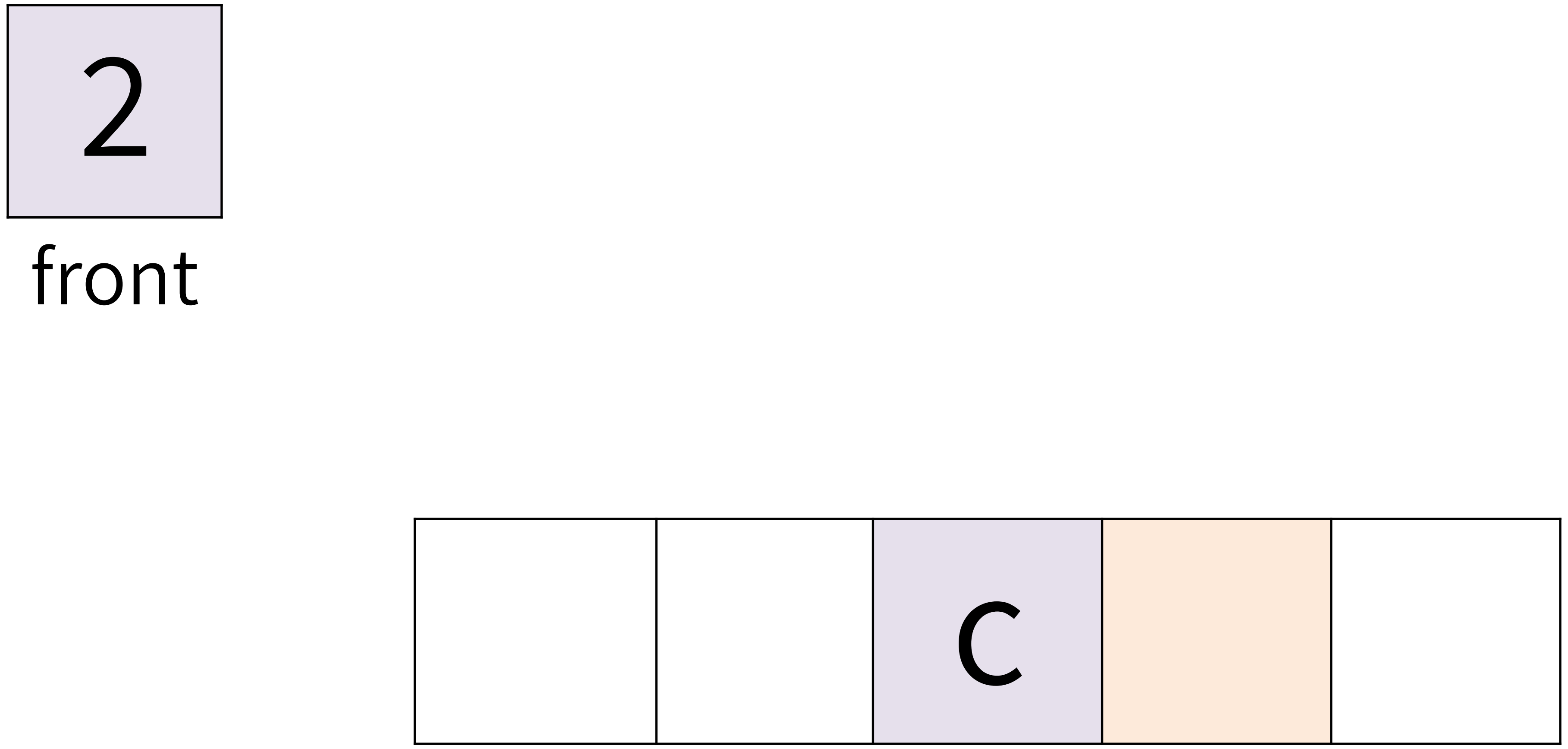
3

rear

	b	c		

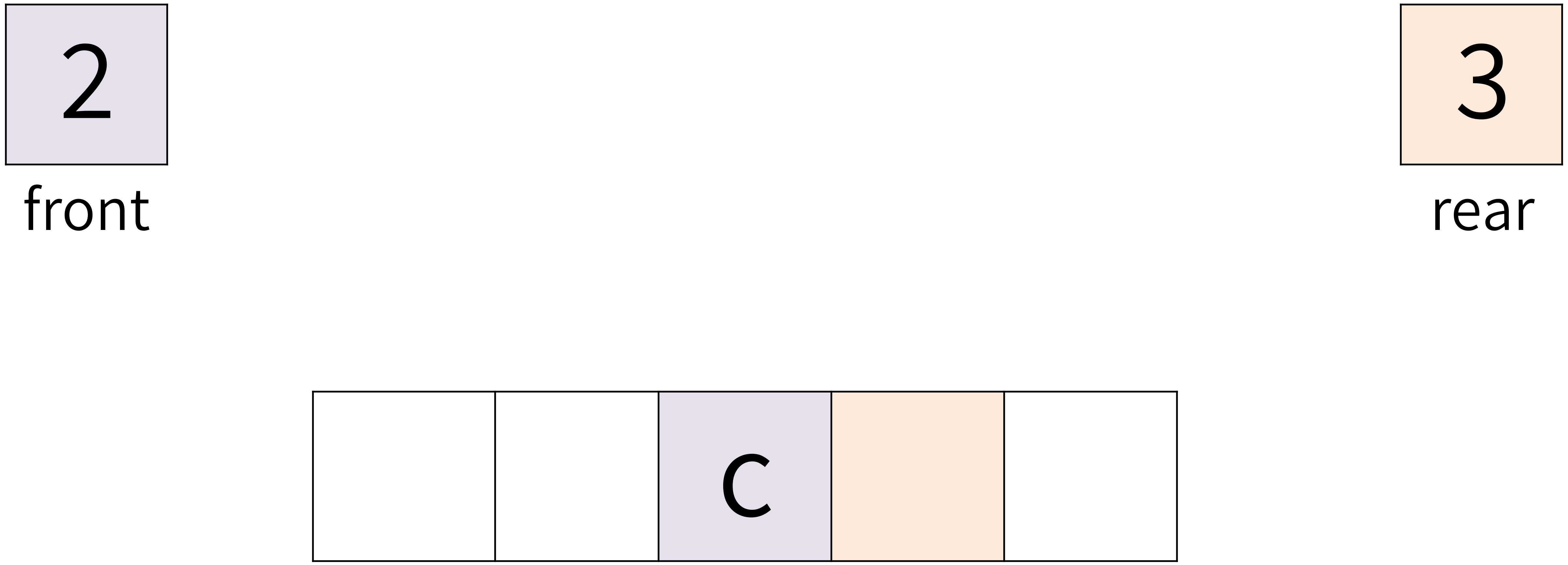
`Dequeue()`

Queue Implementation with Arrays



`Dequeue()→b`

Queue Implementation with Arrays



2

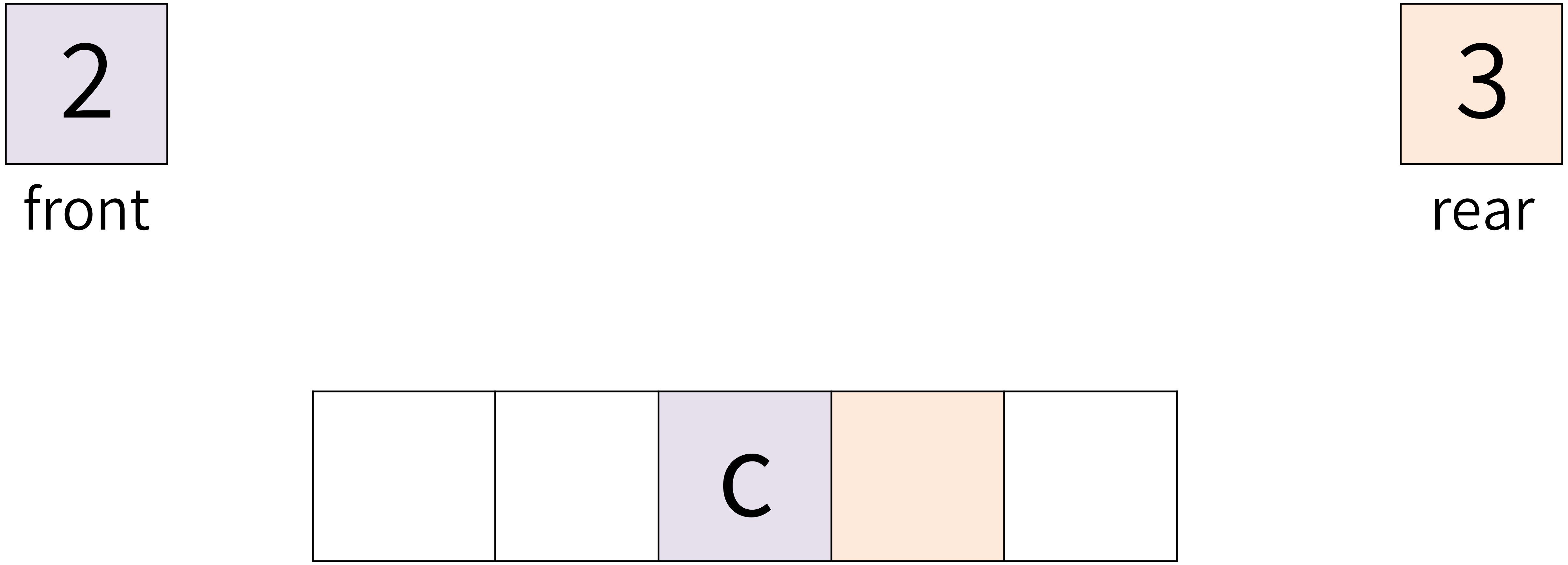
front

3

rear

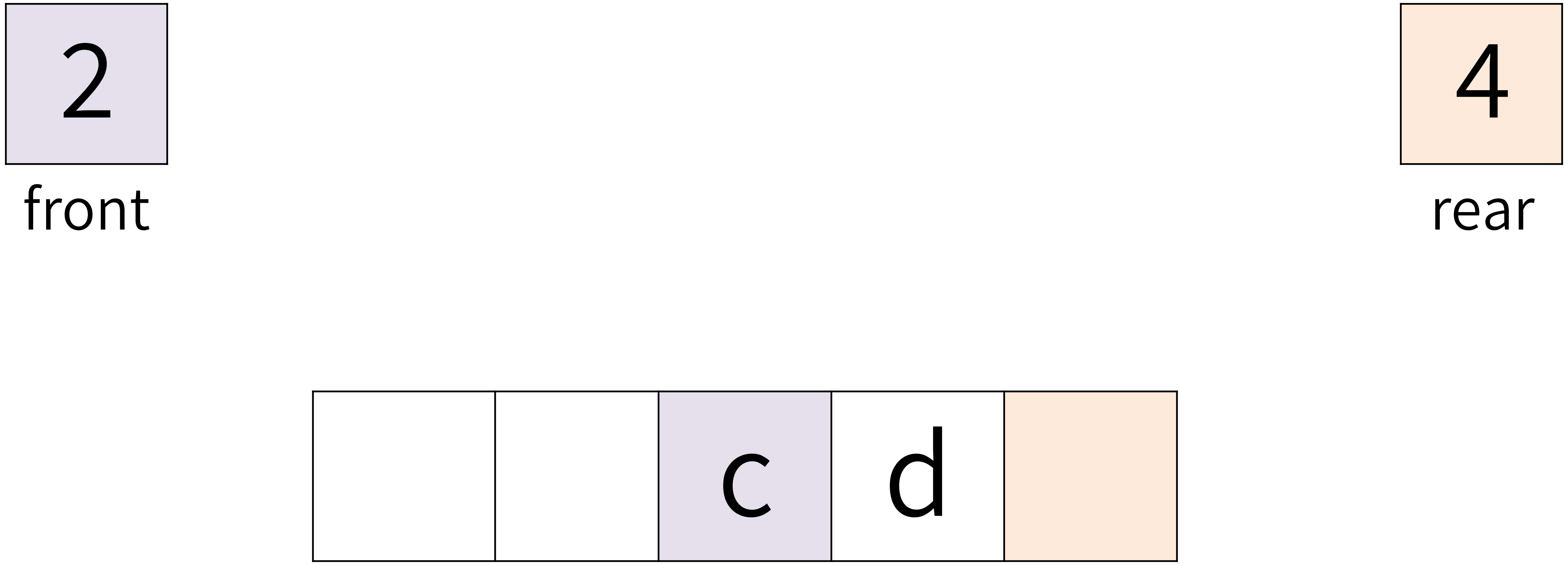
C

Queue Implementation with Arrays



```
Enqueue(d)
```

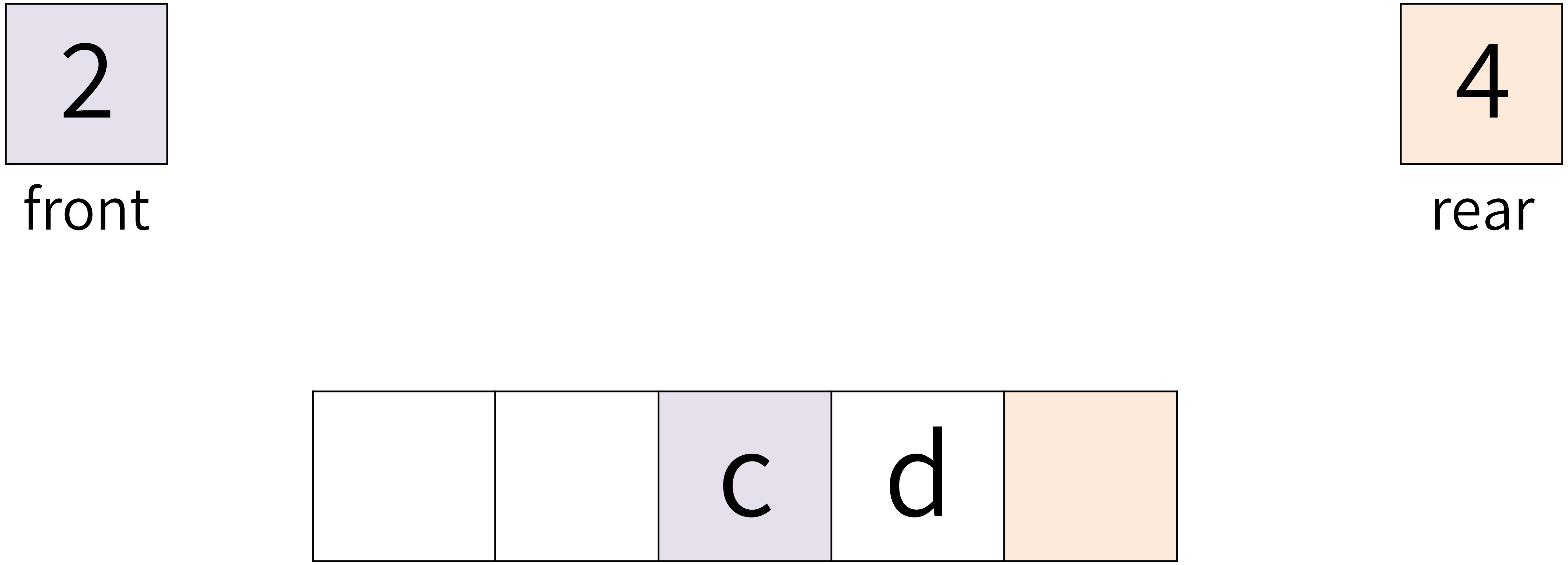
Queue Implementation with Arrays



		c	d	

```
Enqueue(d)
```

Queue Implementation with Arrays



2

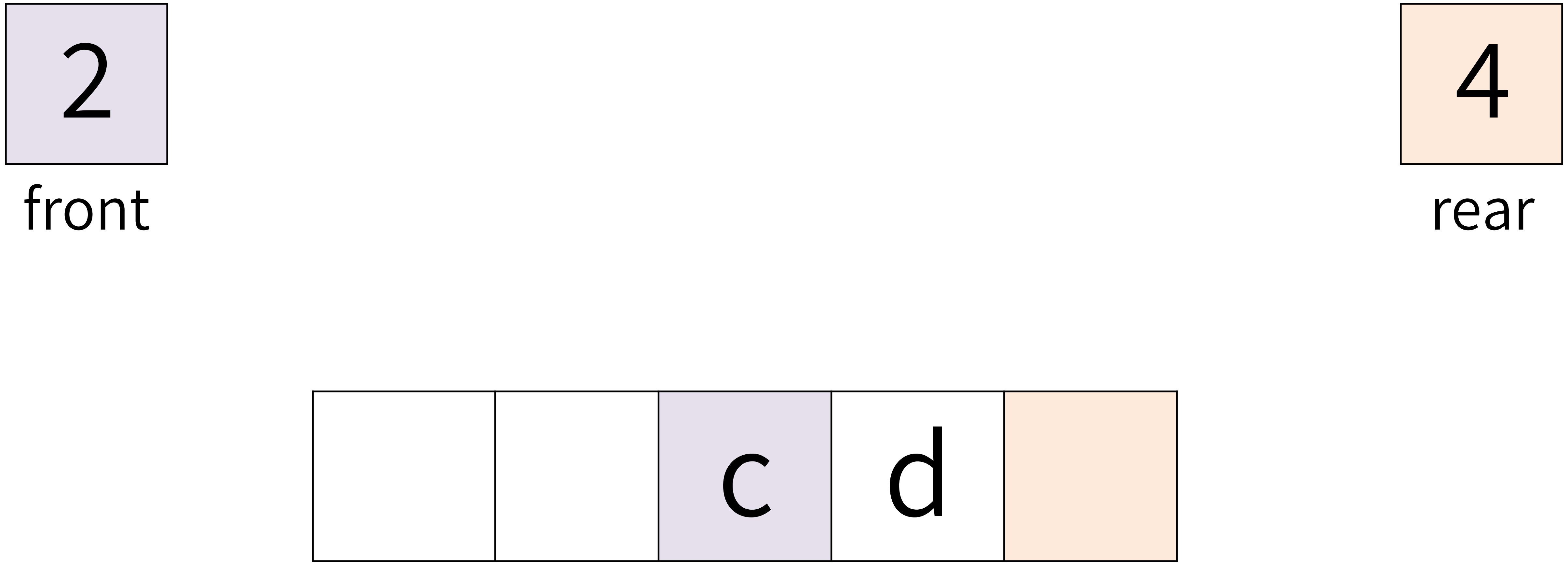
front

4

rear

		c	d	

Queue Implementation with Arrays



2

front

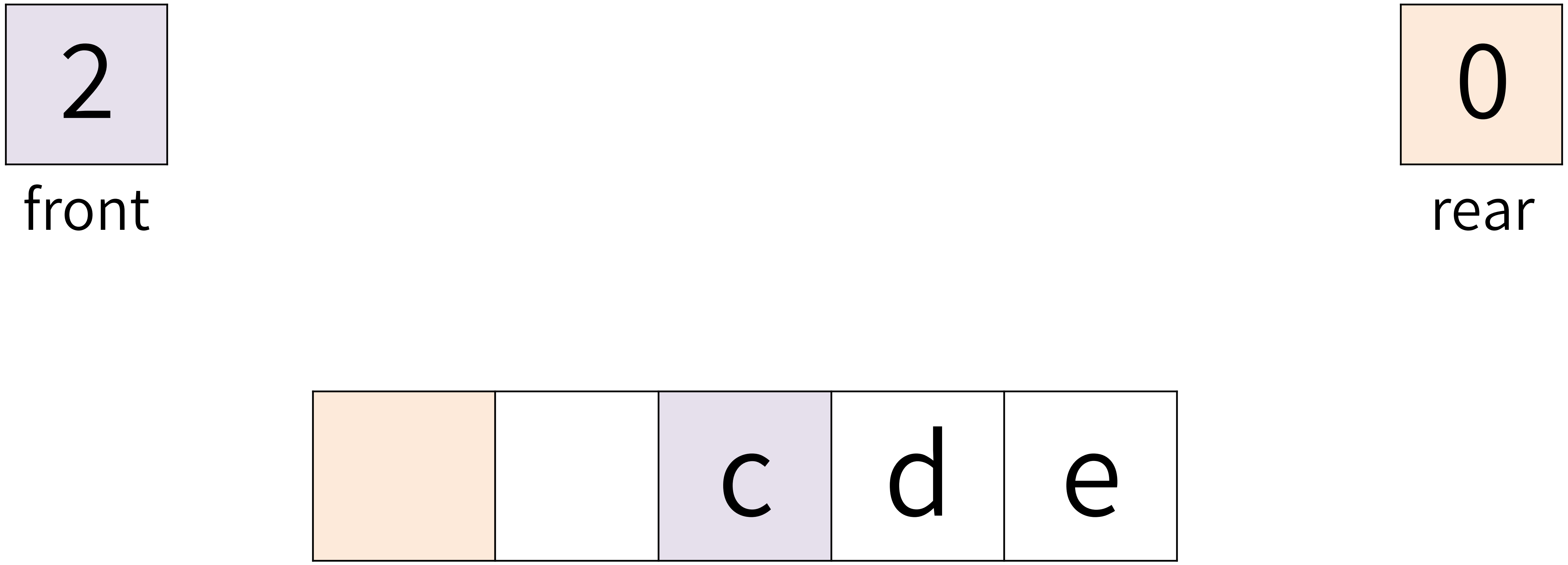
4

rear

		c	d	

`Enqueue(e)`

Queue Implementation with Arrays



2

front

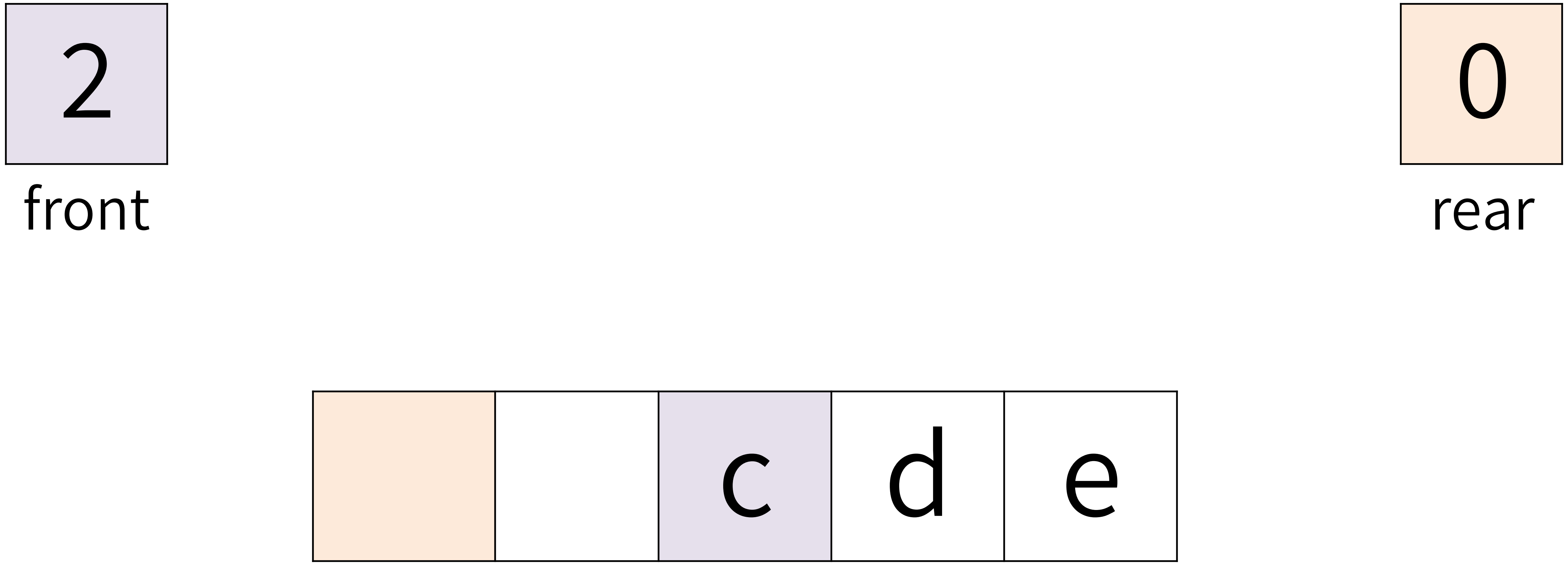
0

rear

		c	d	e

`Enqueue(e)`

Queue Implementation with Arrays



2

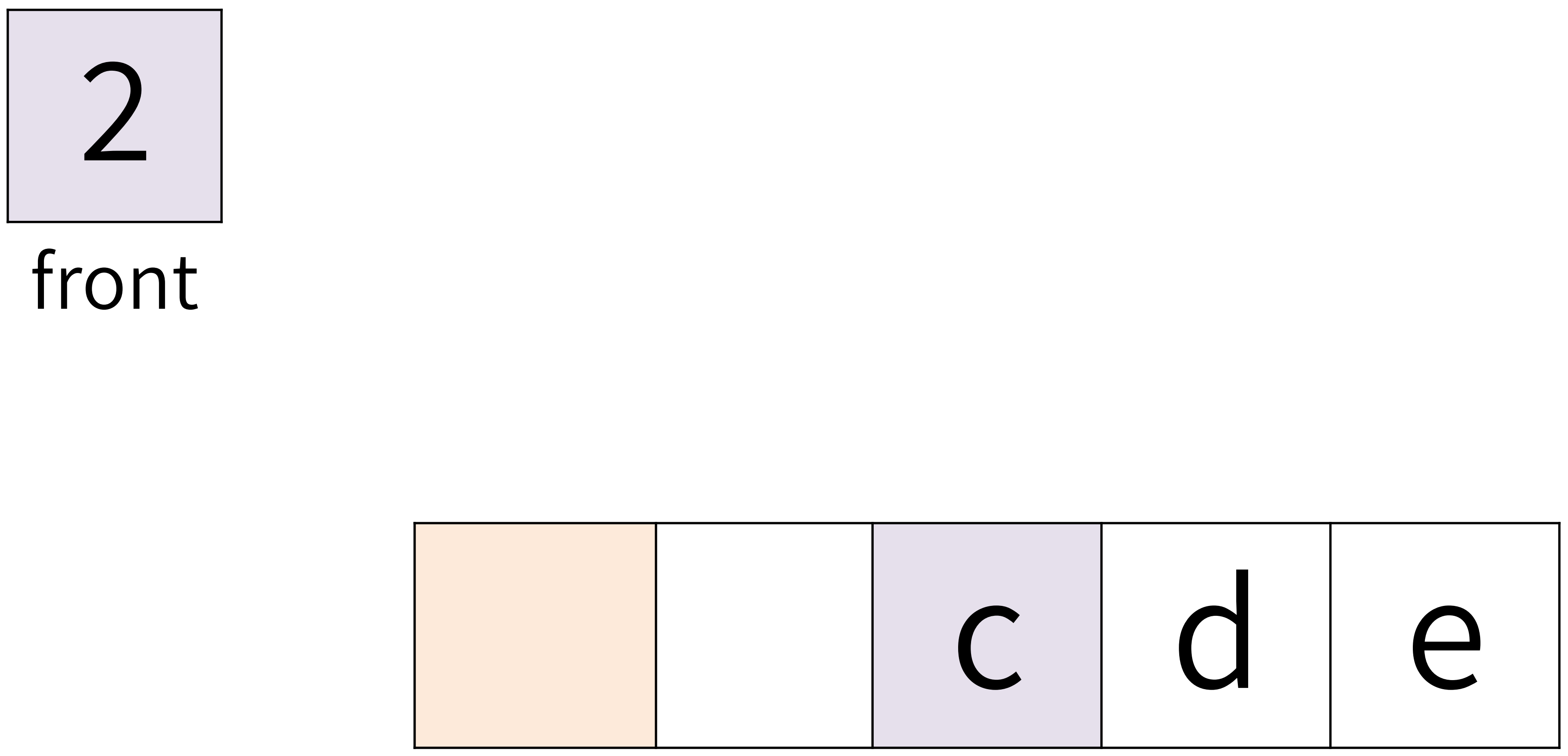
front

0

rear

		c	d	e

Queue Implementation with Arrays



2

front

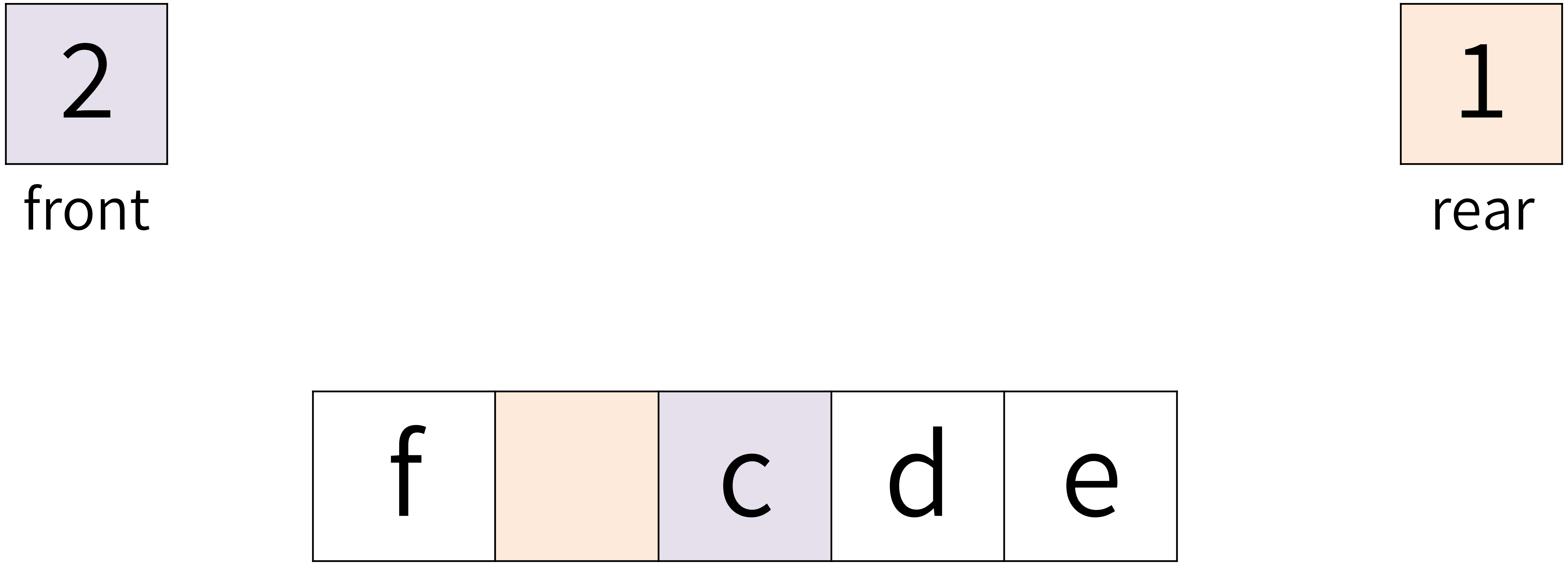
0

rear

		c	d	e

```
Enqueue(f)
```

Queue Implementation with Arrays



2

front

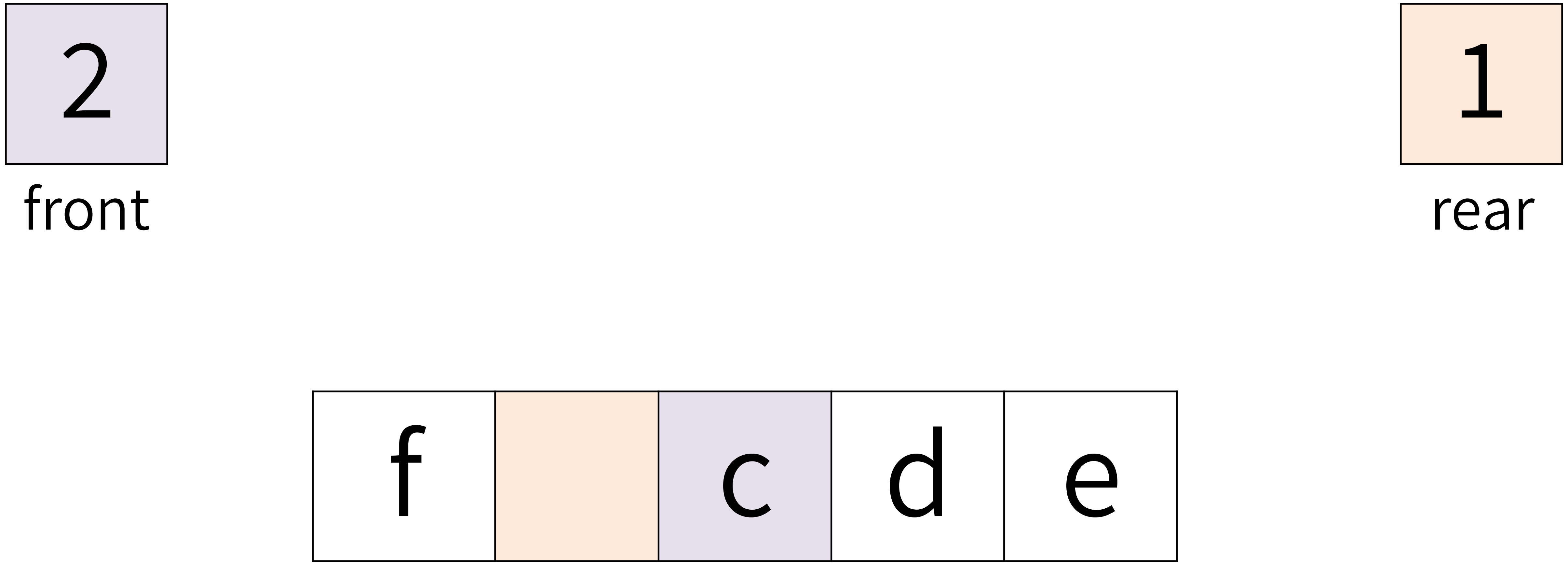
1

rear

f		c	d	e

`Enqueue(f)`

Queue Implementation with Arrays



2

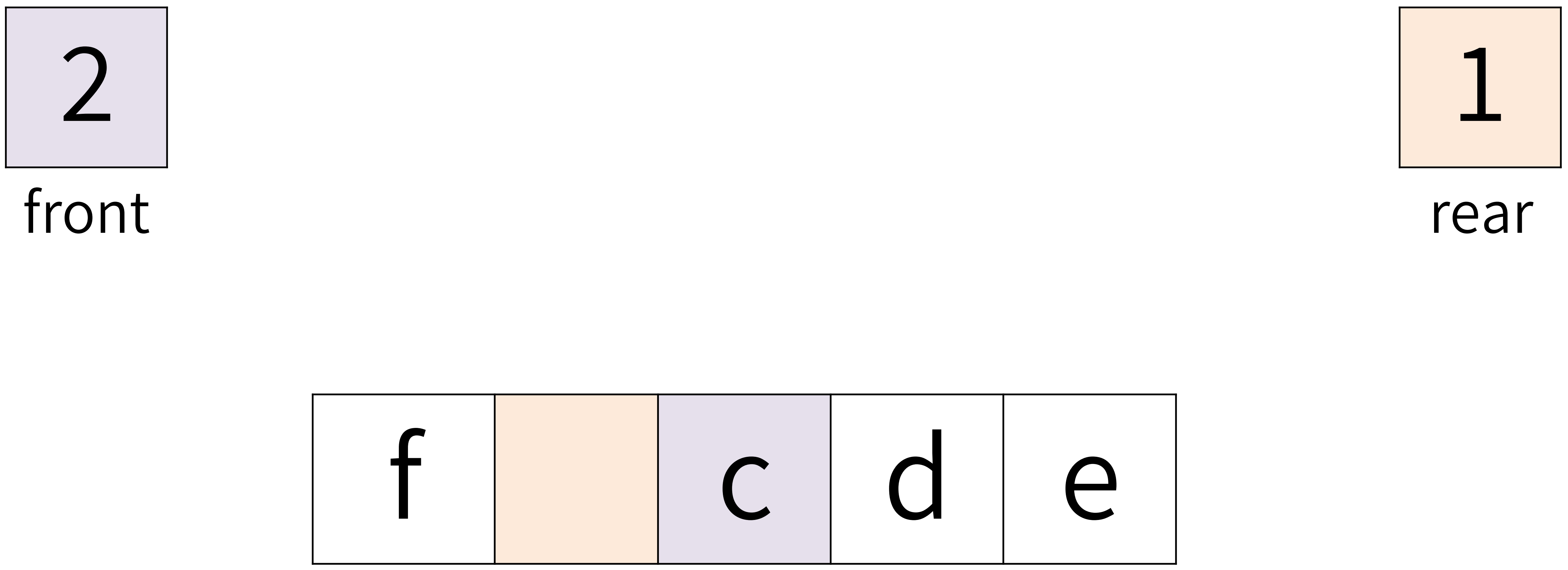
front

1

rear

f		c	d	e

Queue Implementation with Arrays



2

front

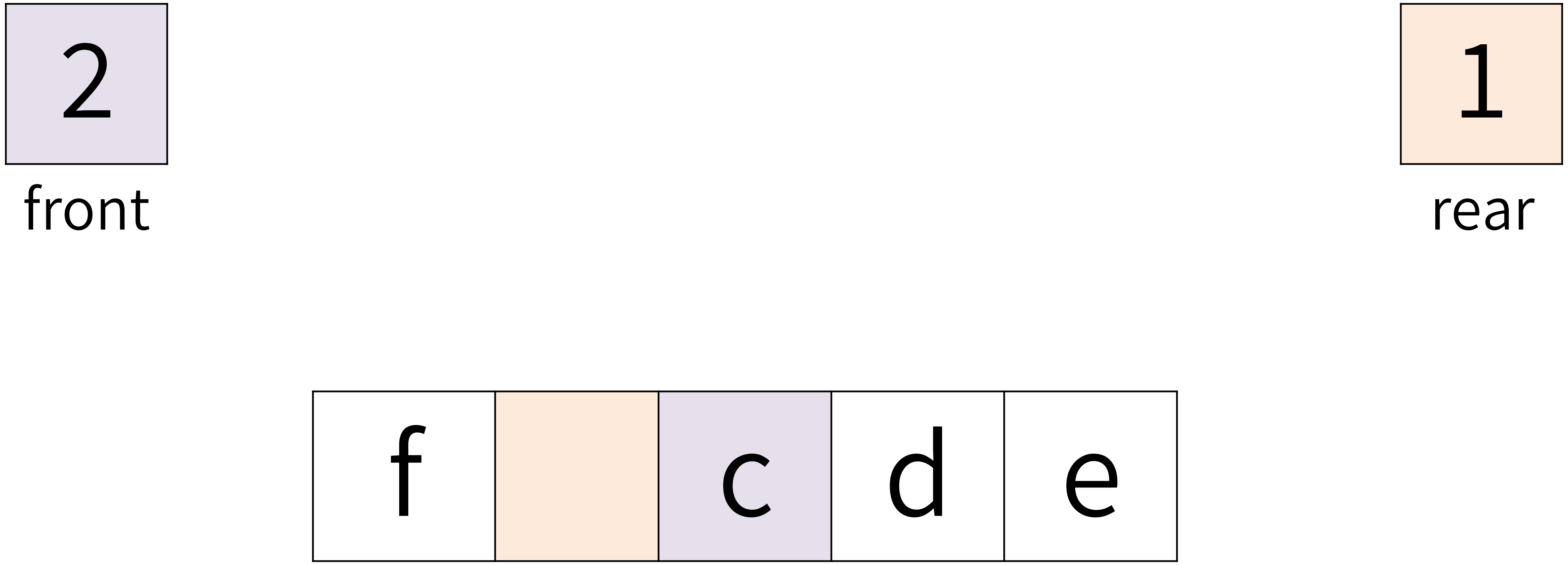
1

rear

f		c	d	e

```
Enqueue(g)
```

Queue Implementation with Arrays



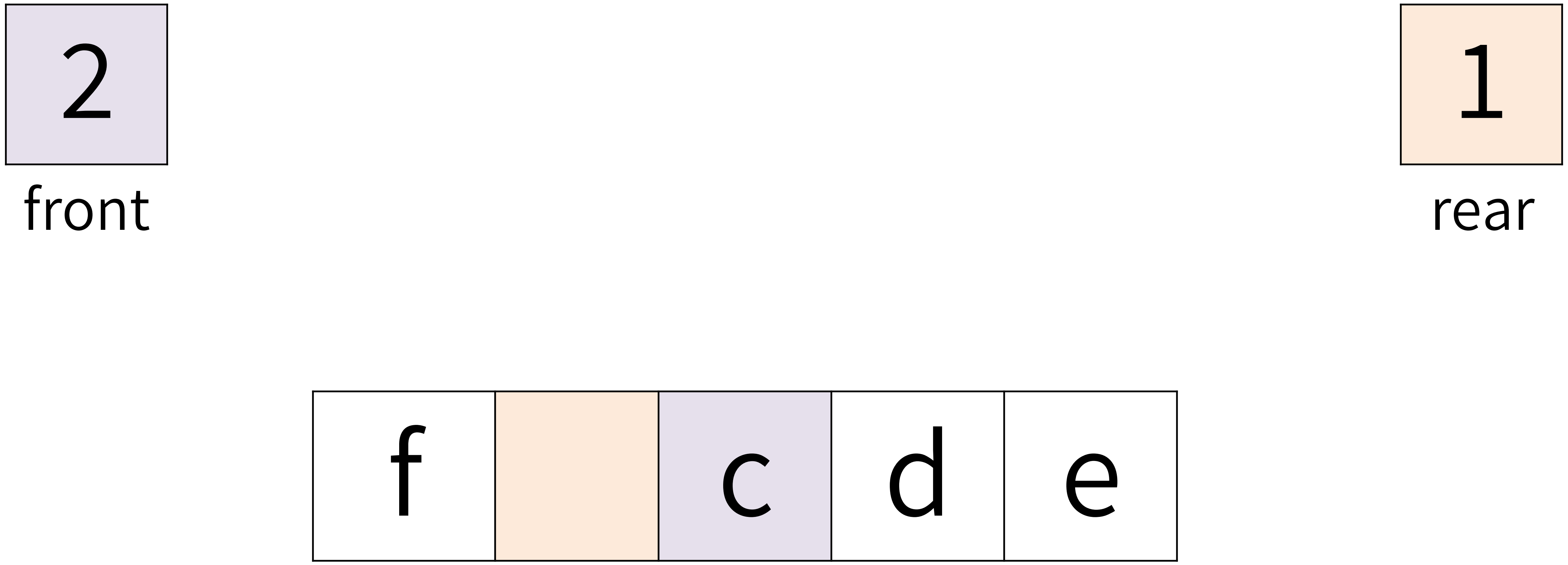
2
front

1
rear

f		c	d	e

`Enqueue(g)→ERROR`

Queue Implementation with Arrays

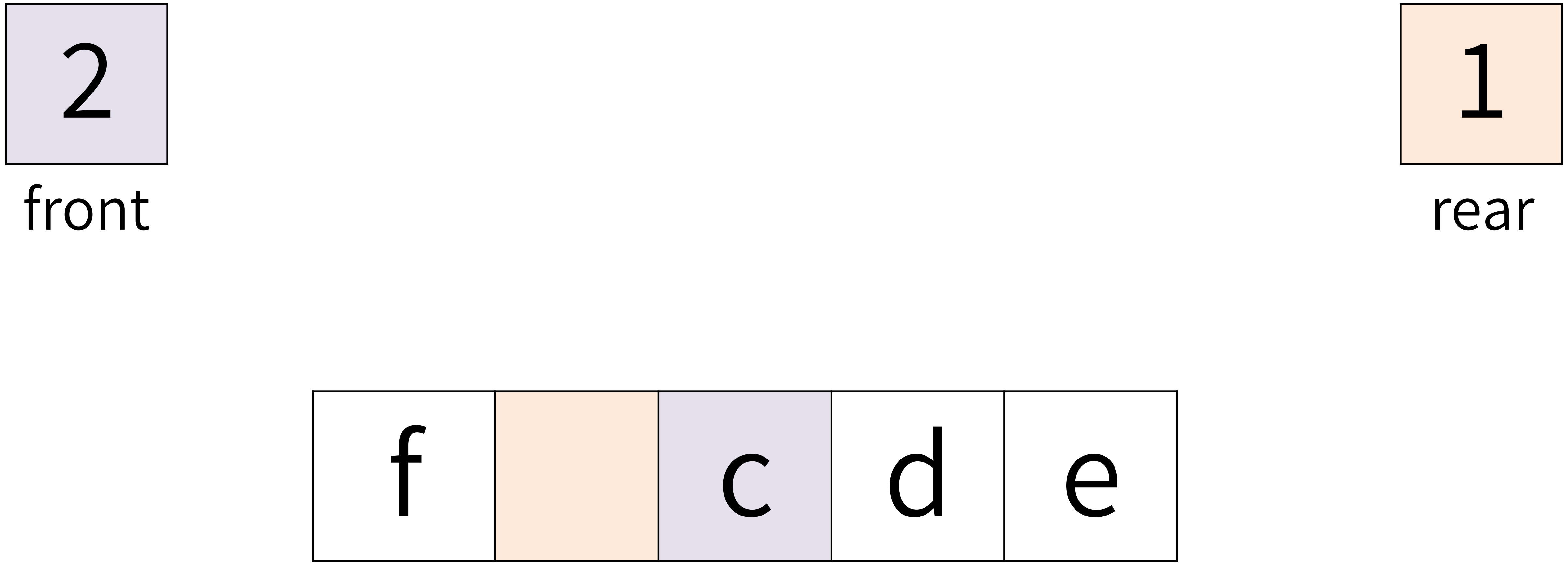


2
front

1
rear

f		c	d	e

Queue Implementation with Arrays



2

front

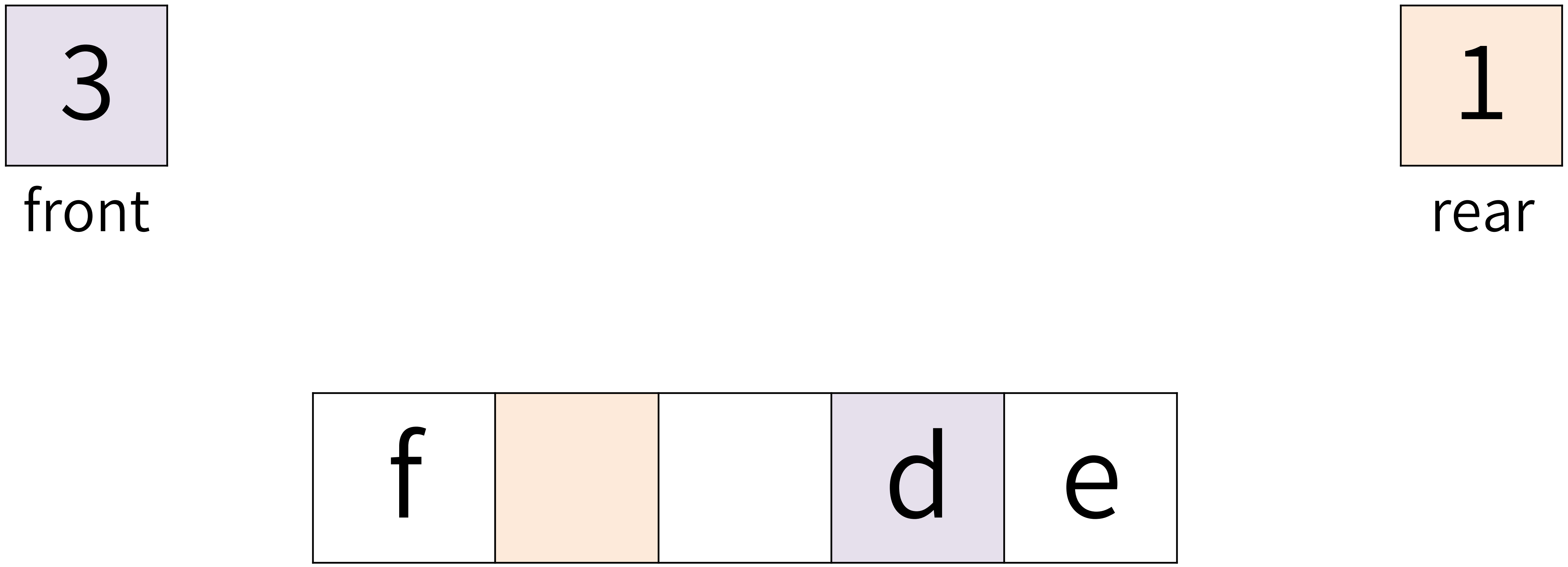
1

rear

f		c	d	e

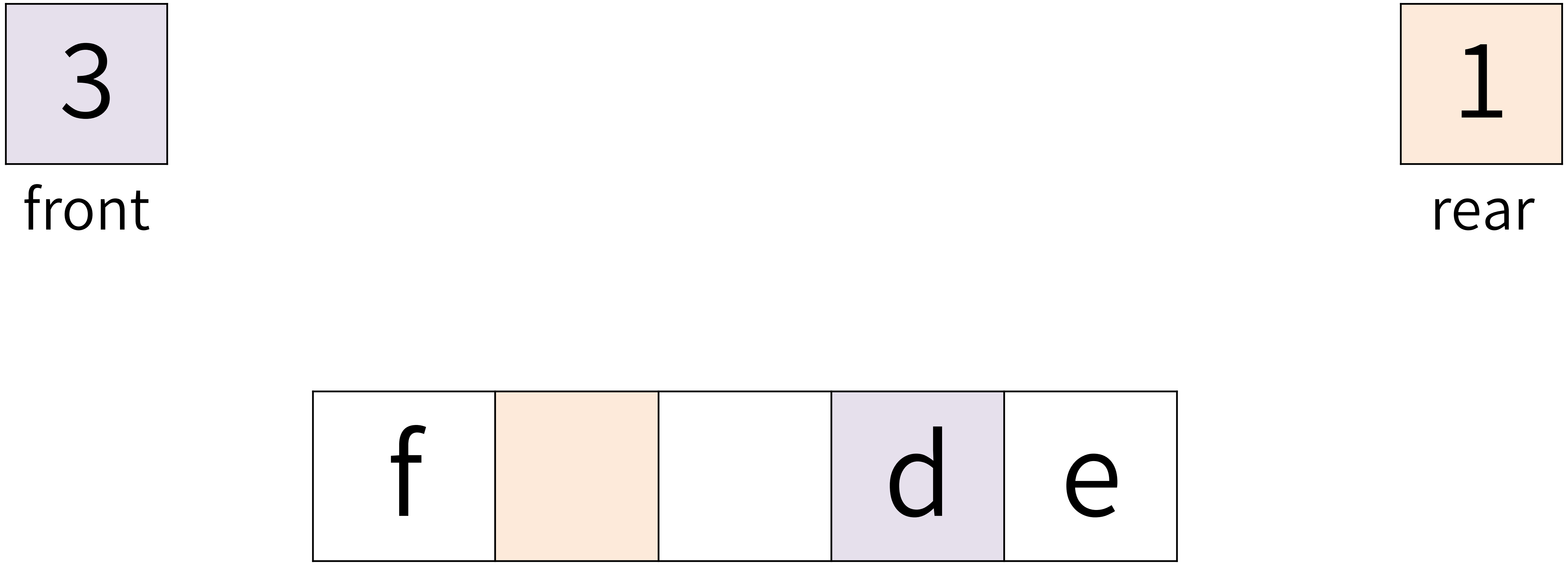
`Dequeue()`

Queue Implementation with Arrays



`Dequeue()→c`

Queue Implementation with Arrays



3

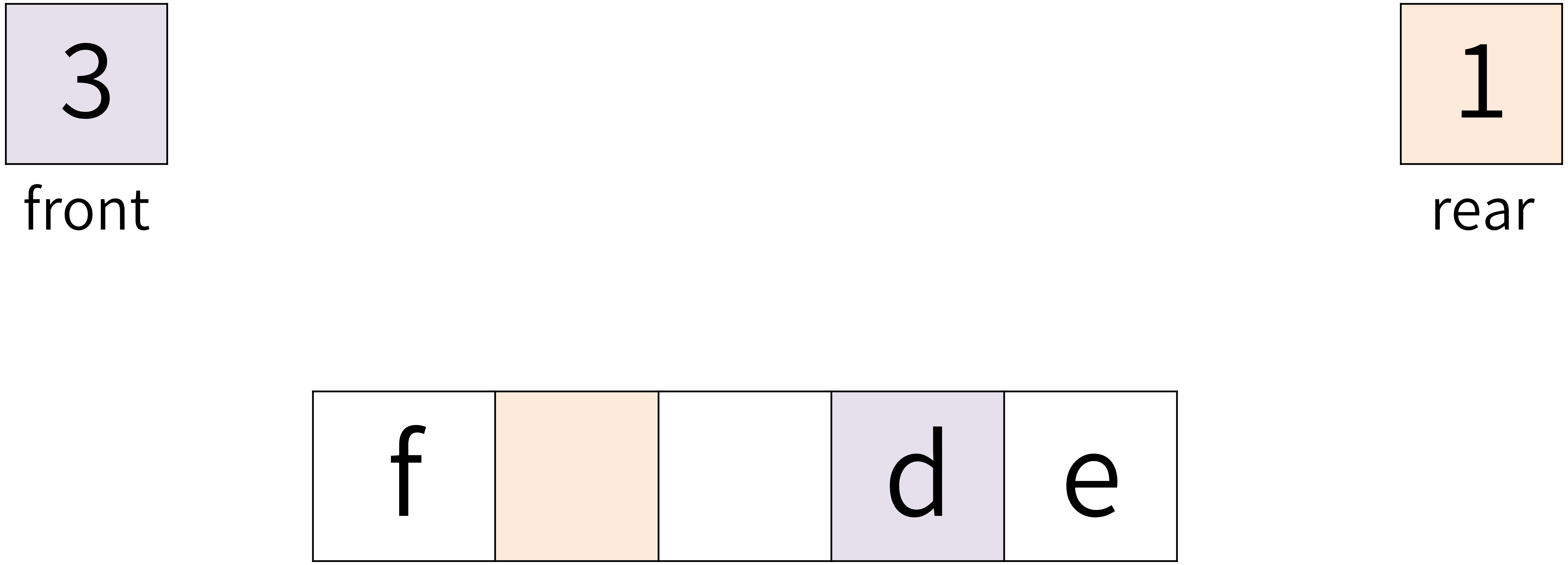
front

1

rear

f			d	e

Queue Implementation with Arrays



3

front

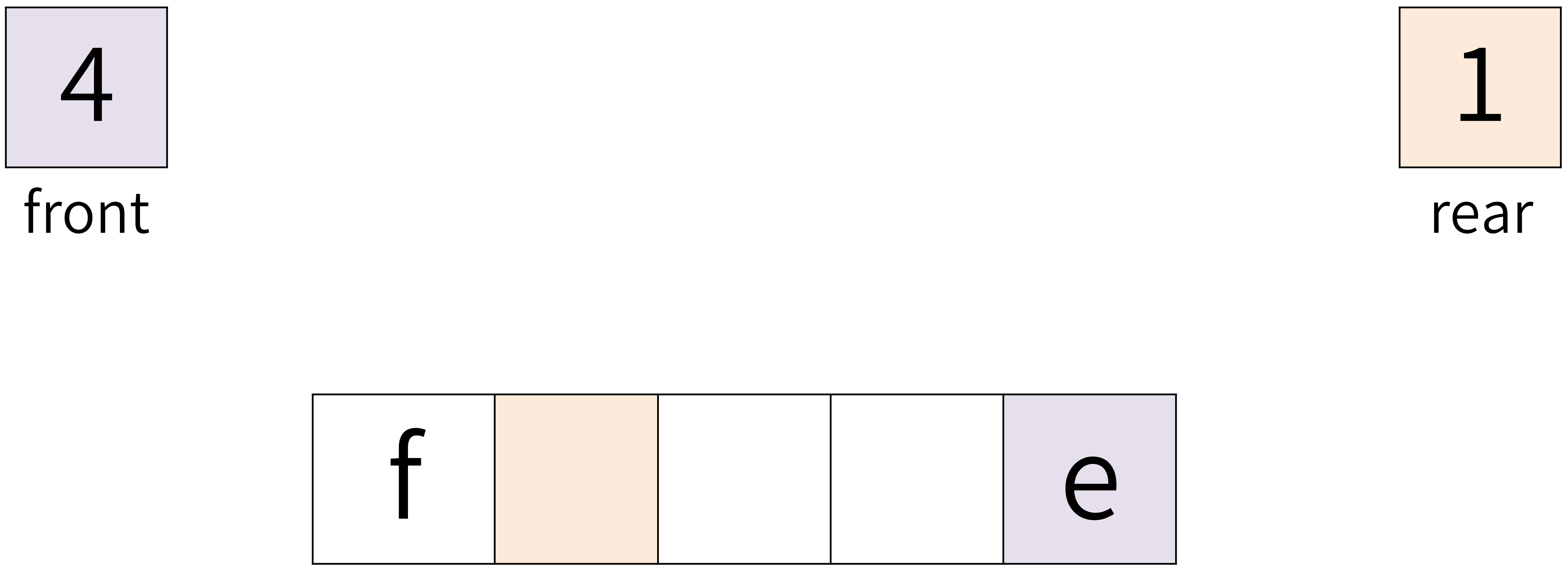
1

rear

f			d	e

`Dequeue()`

Queue Implementation with Arrays



4

front

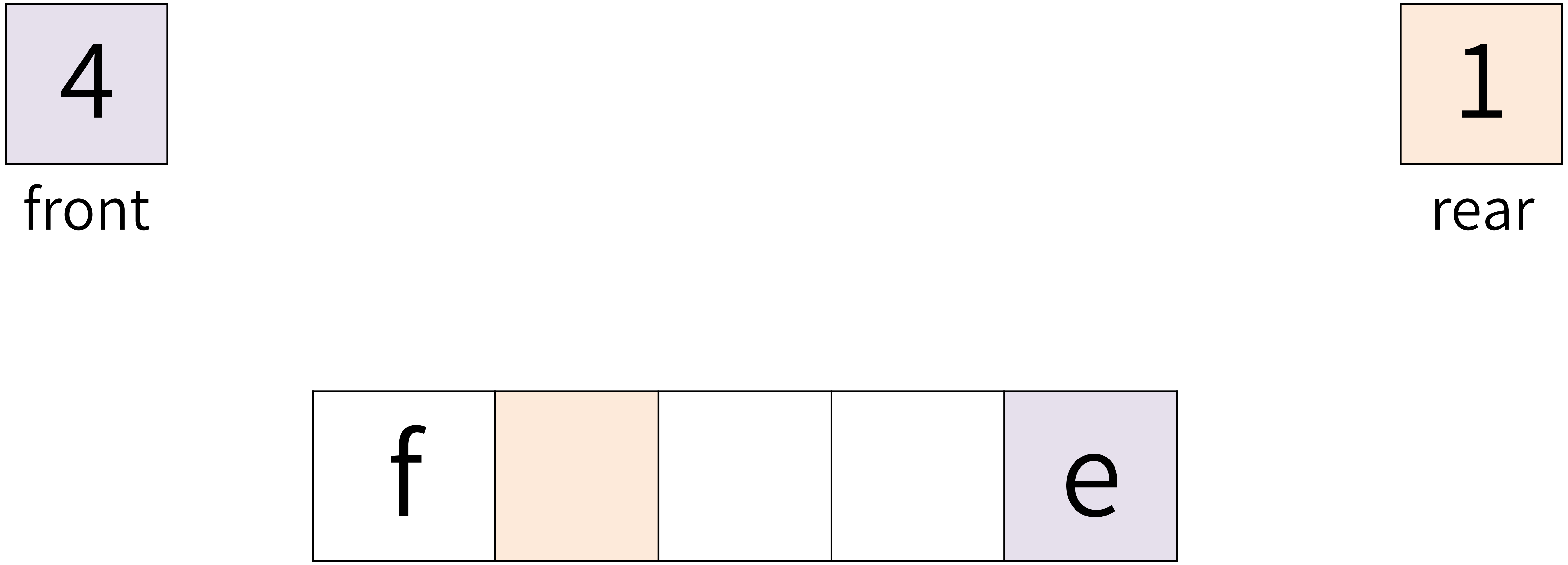
1

rear

f				e

`Dequeue()→d`

Queue Implementation with Arrays



4

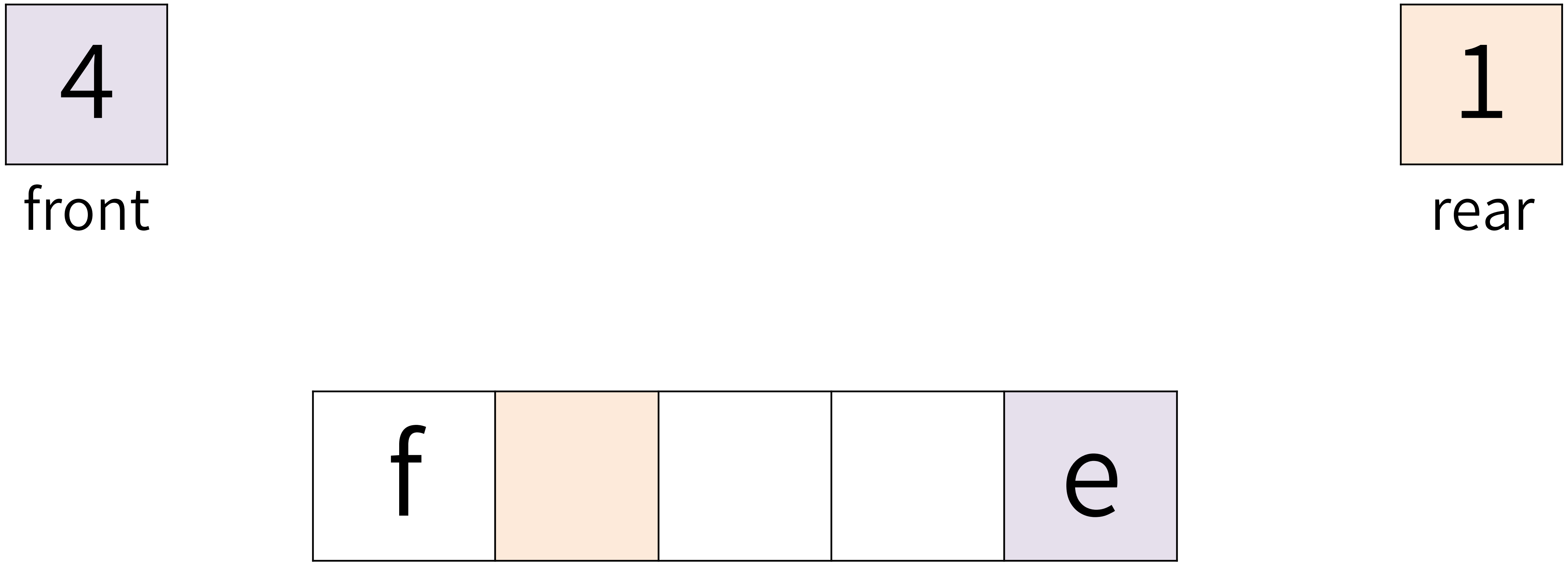
front

1

rear

f				e

Queue Implementation with Arrays



4

front

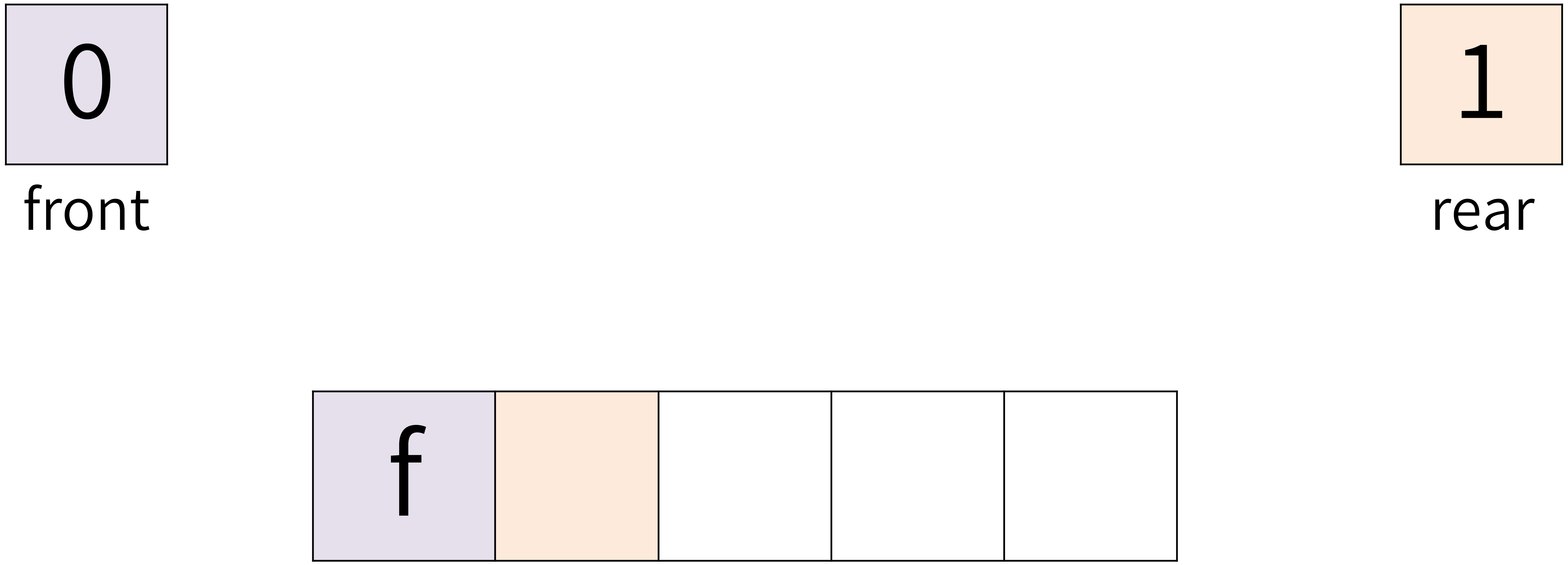
1

rear

f				e

`Dequeue()`

Queue Implementation with Arrays



0

front

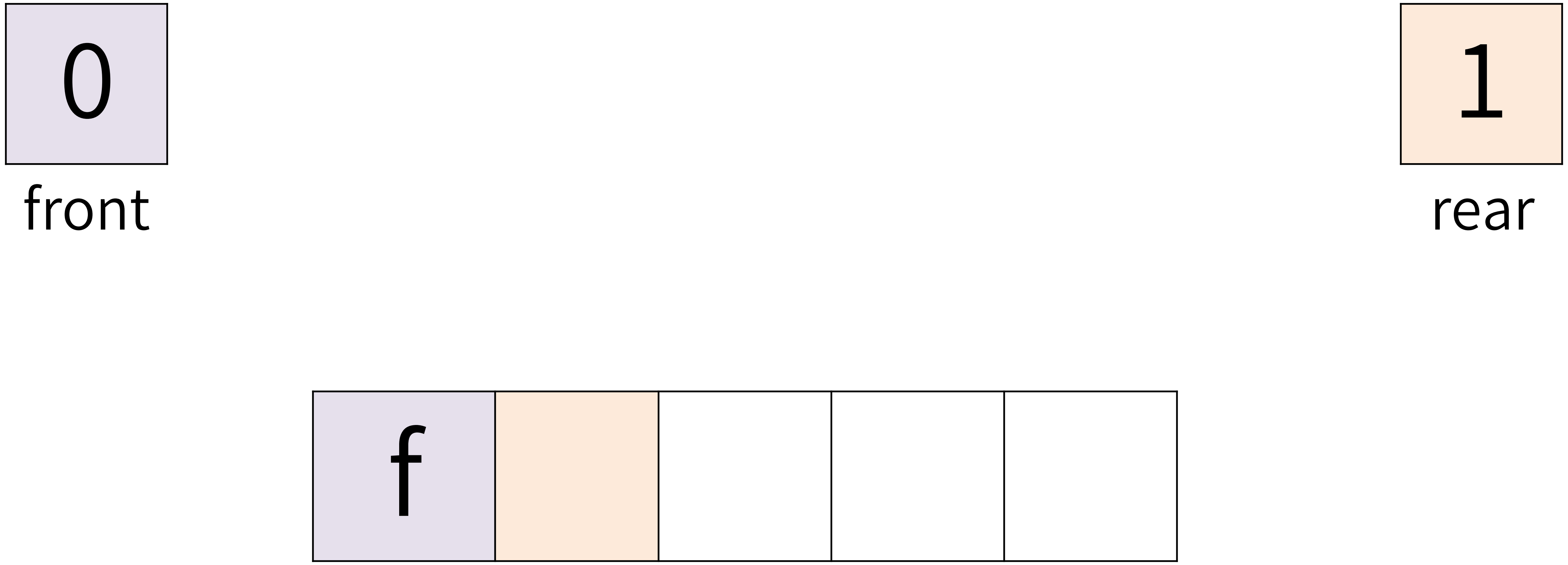
1

rear

f				

`Dequeue()→e`

Queue Implementation with Arrays



0

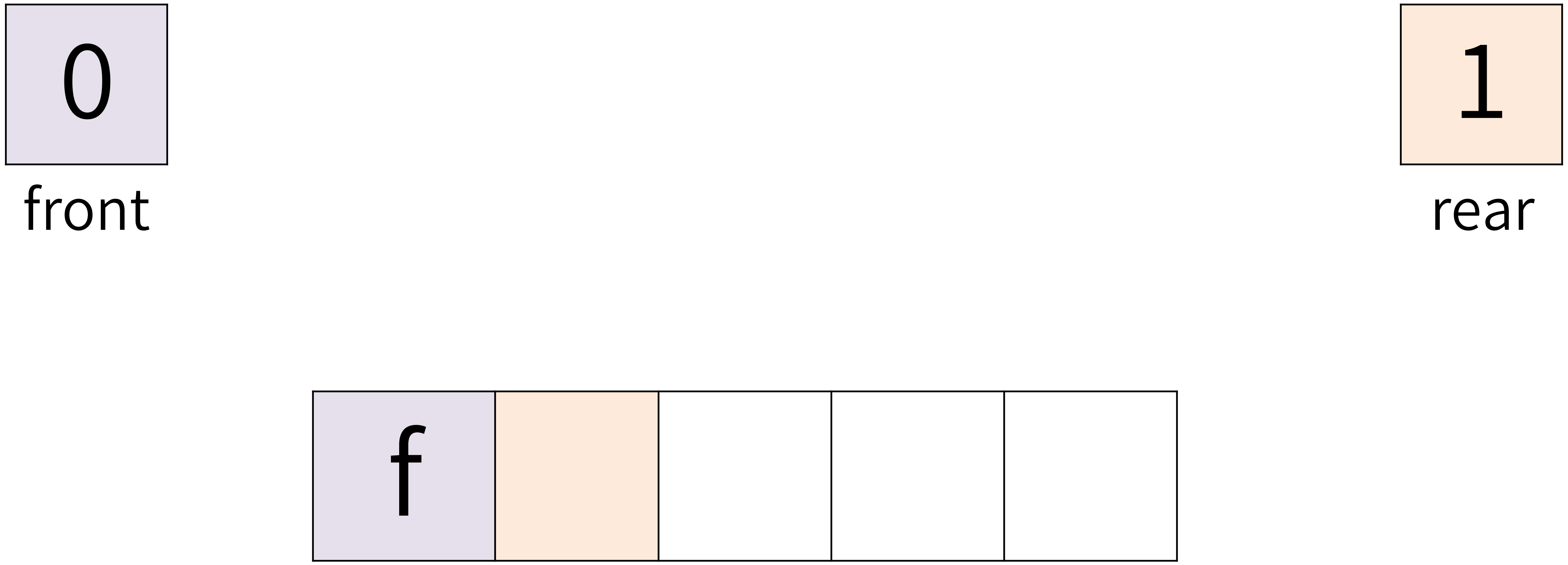
front

1

rear

f				

Queue Implementation with Arrays



0

front

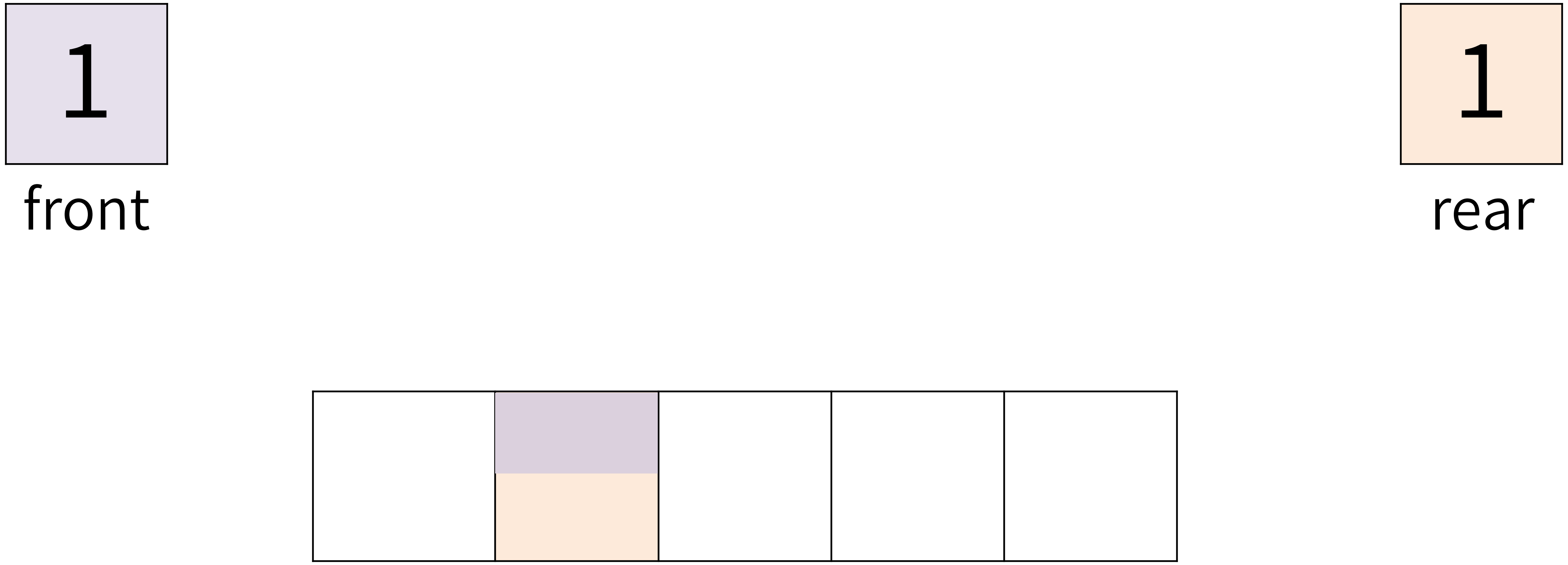
1

rear

f				

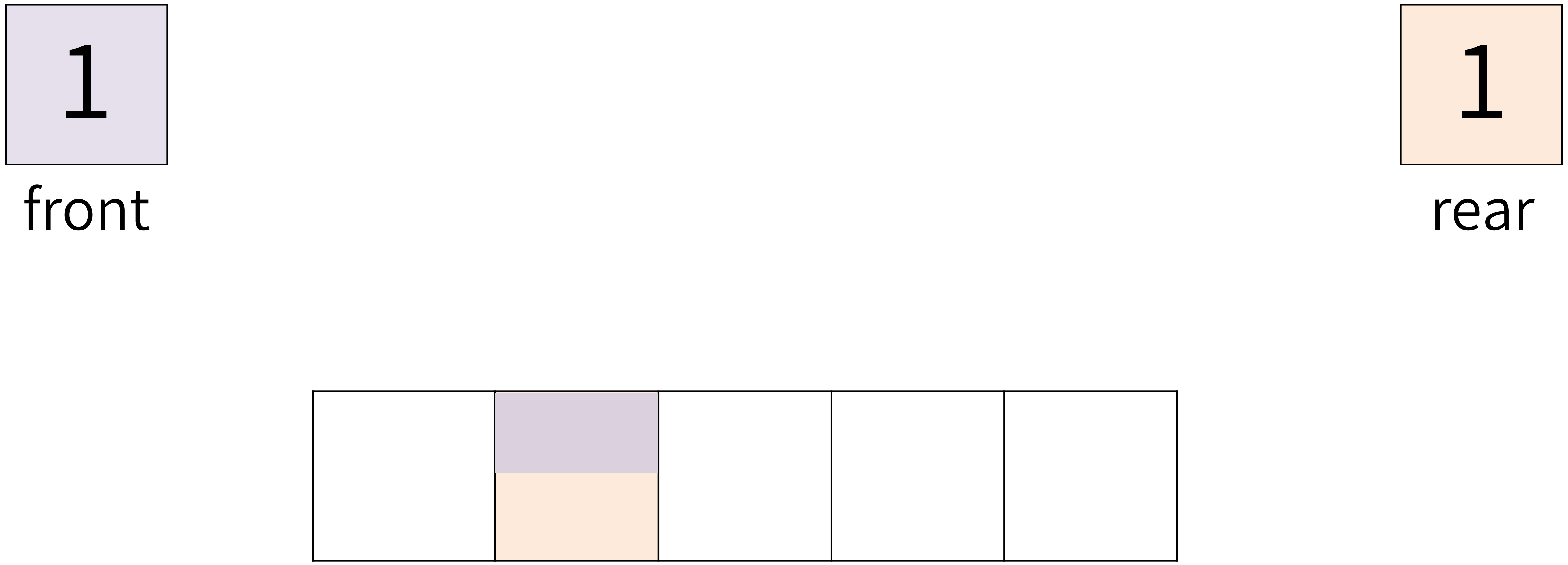
Dequeue()

Queue Implementation with Arrays



`Dequeue()→f`

Queue Implementation with Arrays



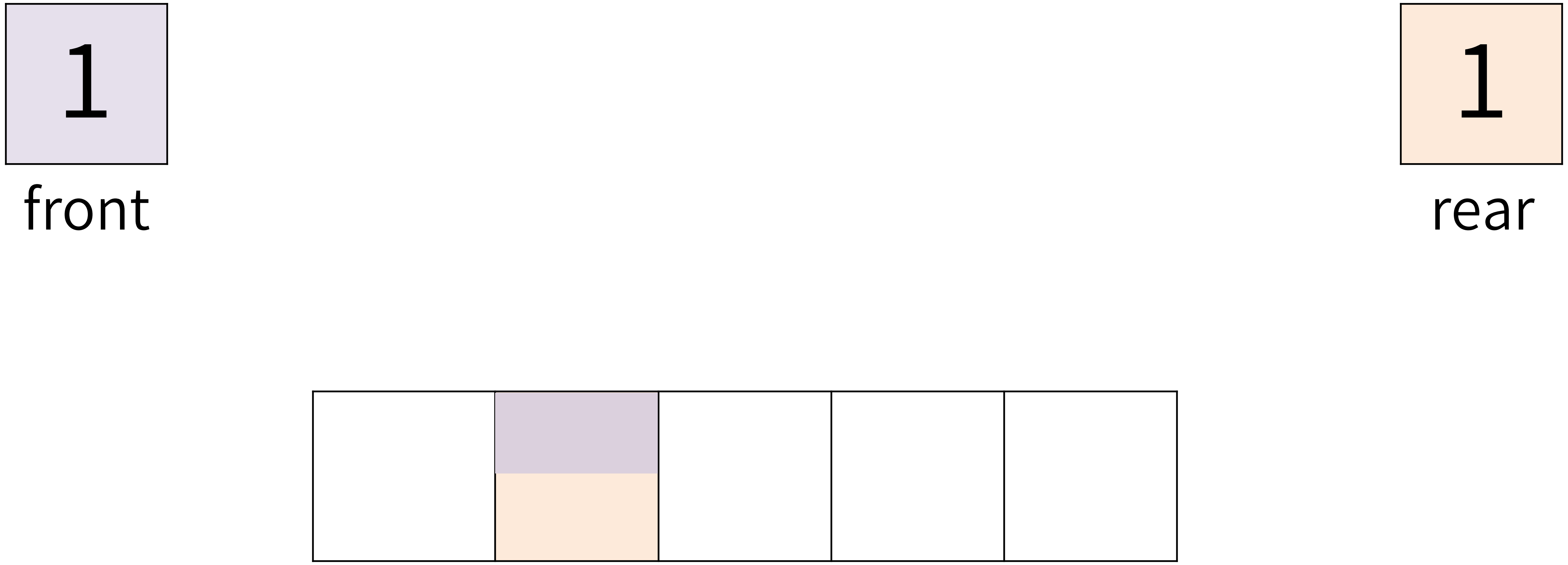
1

front

1

rear

Queue Implementation with Arrays



1

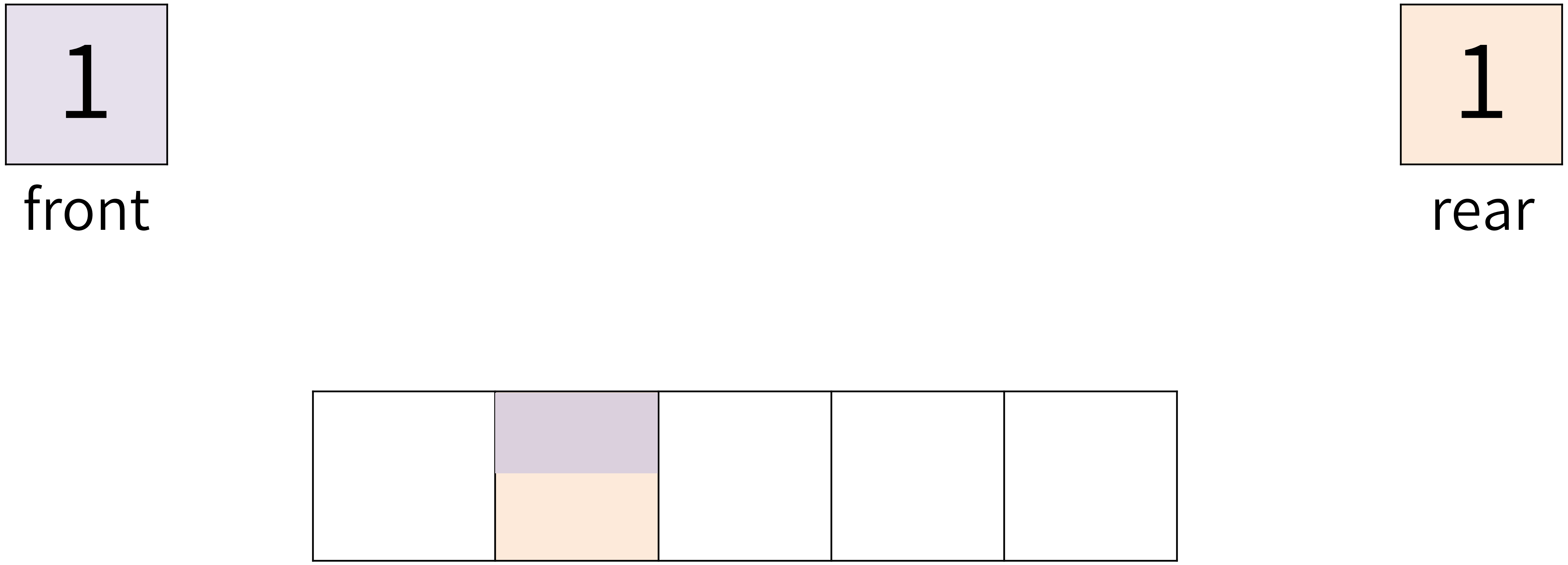
front

1

rear

Empty()

Queue Implementation with Arrays



1

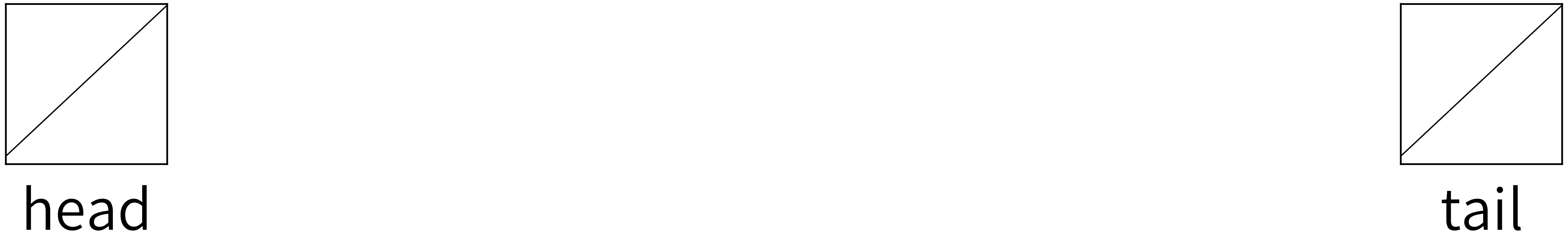
front

1

rear

Empty()→True

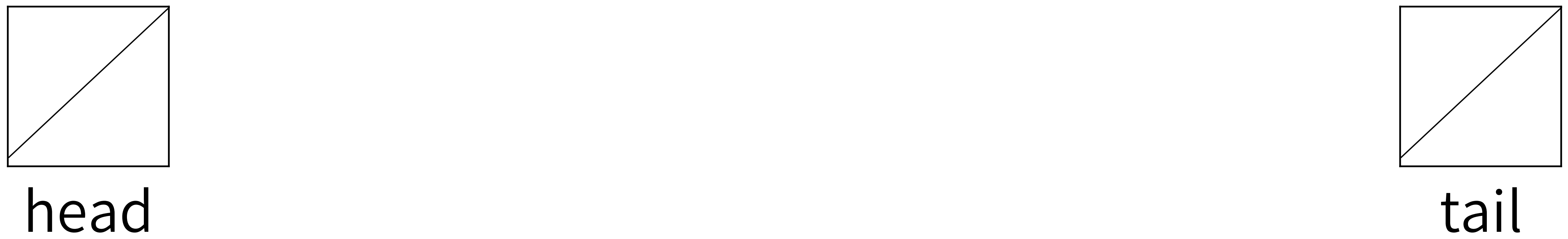
Queue Implementation with Linked List



head

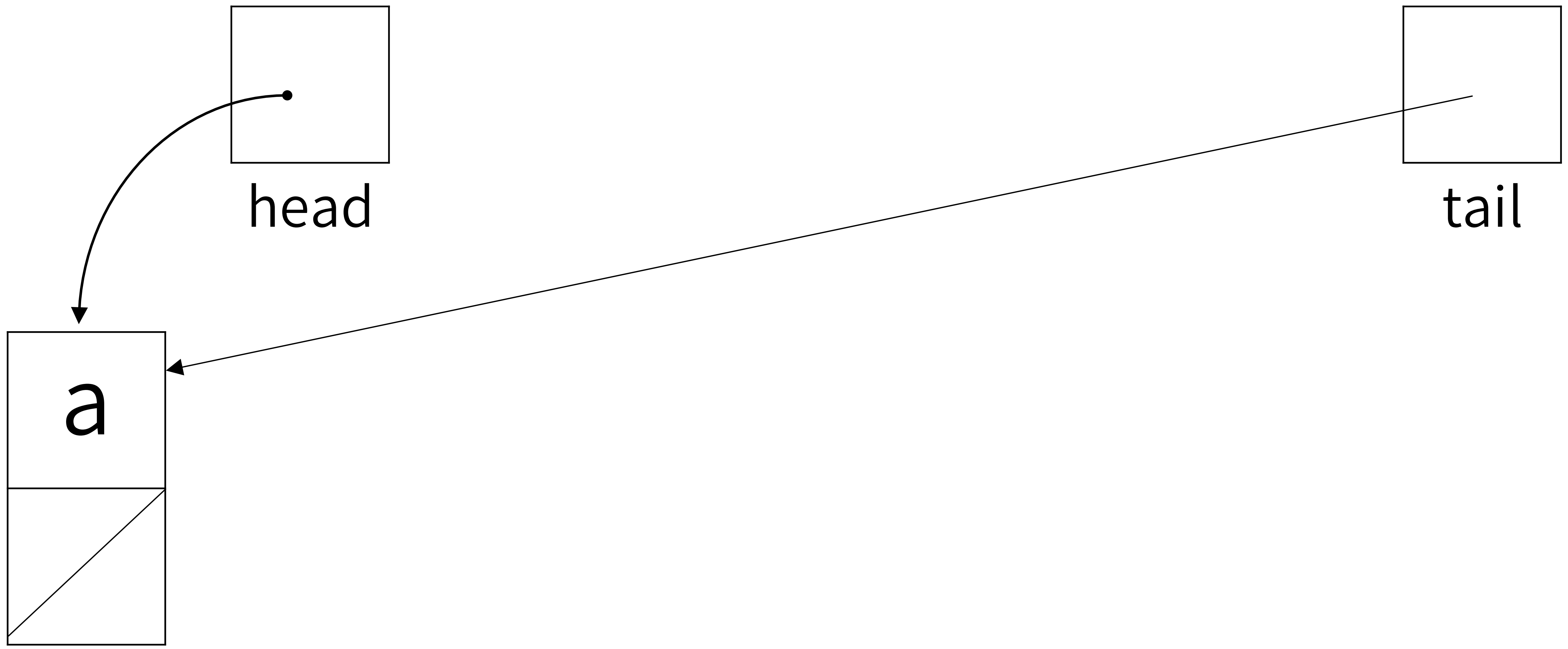
tail

Queue Implementation with Linked List



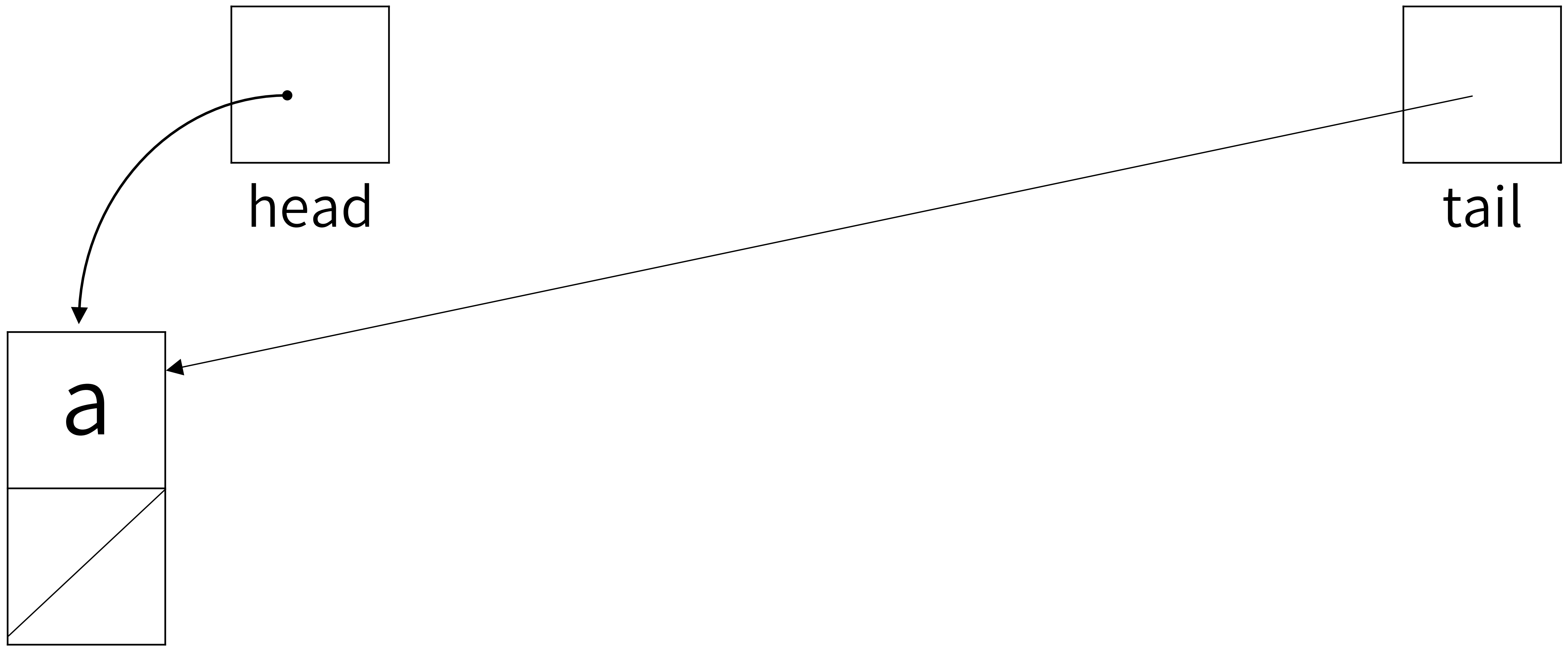
`Enqueue(a)`

Queue Implementation with Linked List

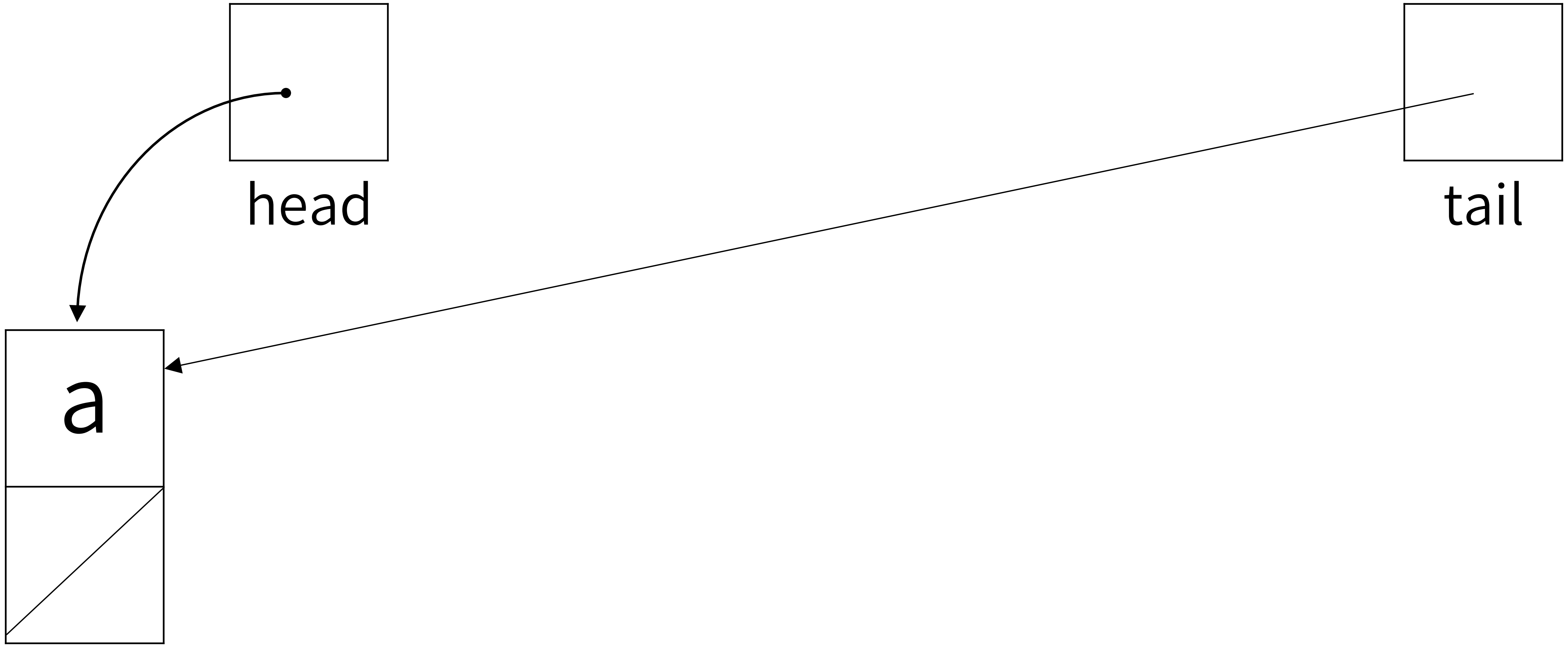


`Enqueue(a)`

Queue Implementation with Linked List

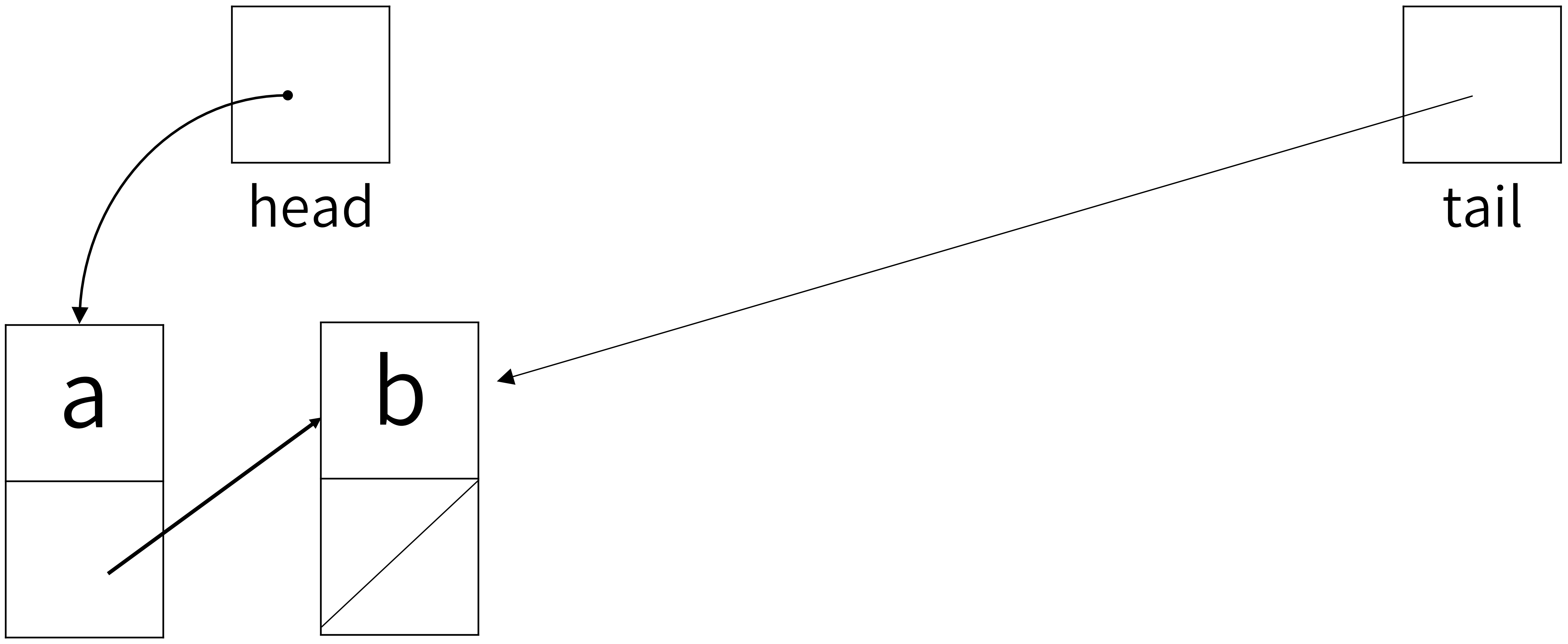


Queue Implementation with Linked List



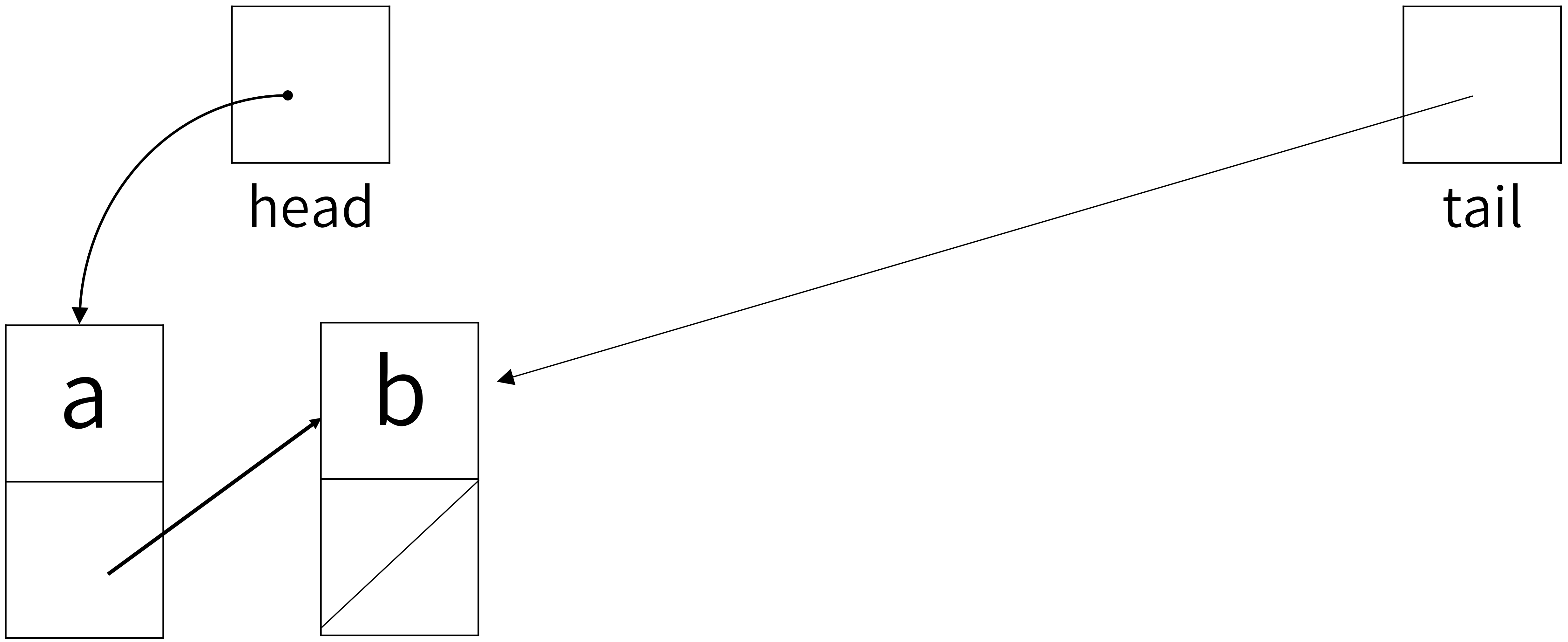
`Enqueue(b)`

Queue Implementation with Linked List

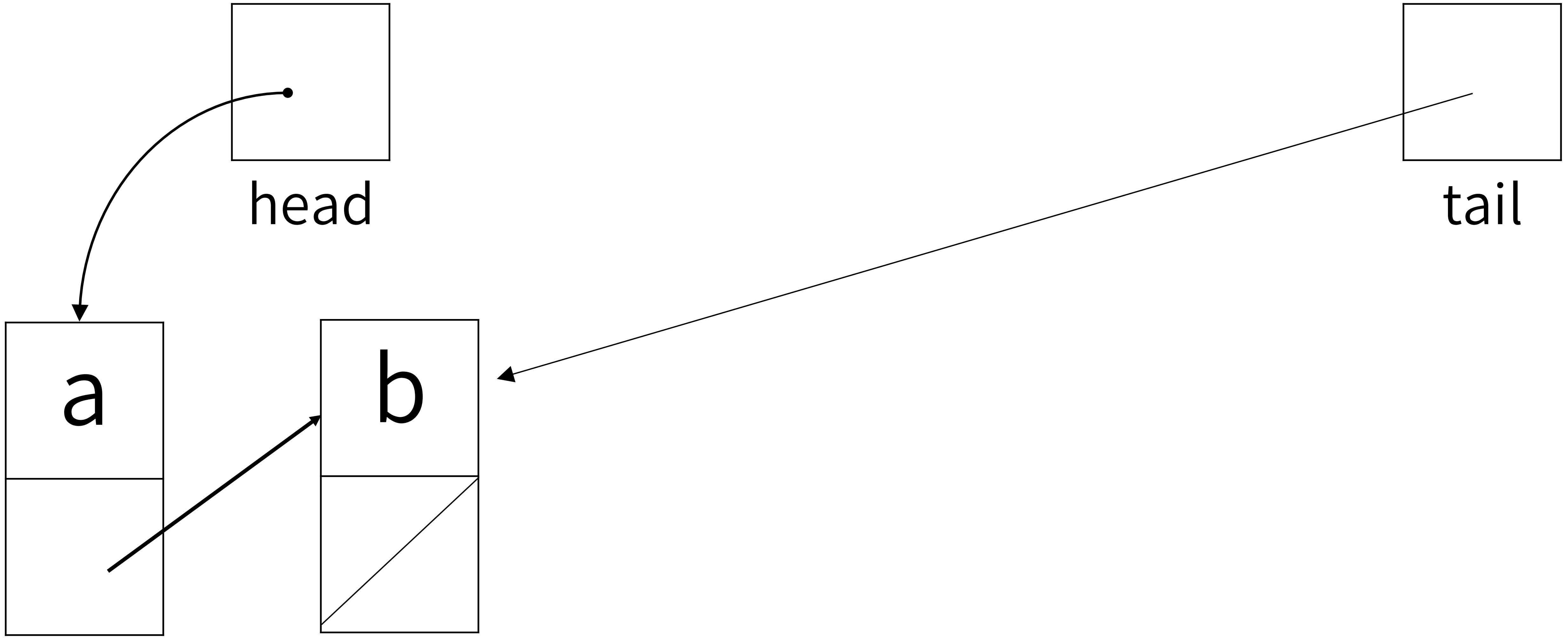


Enqueue(b)

Queue Implementation with Linked List

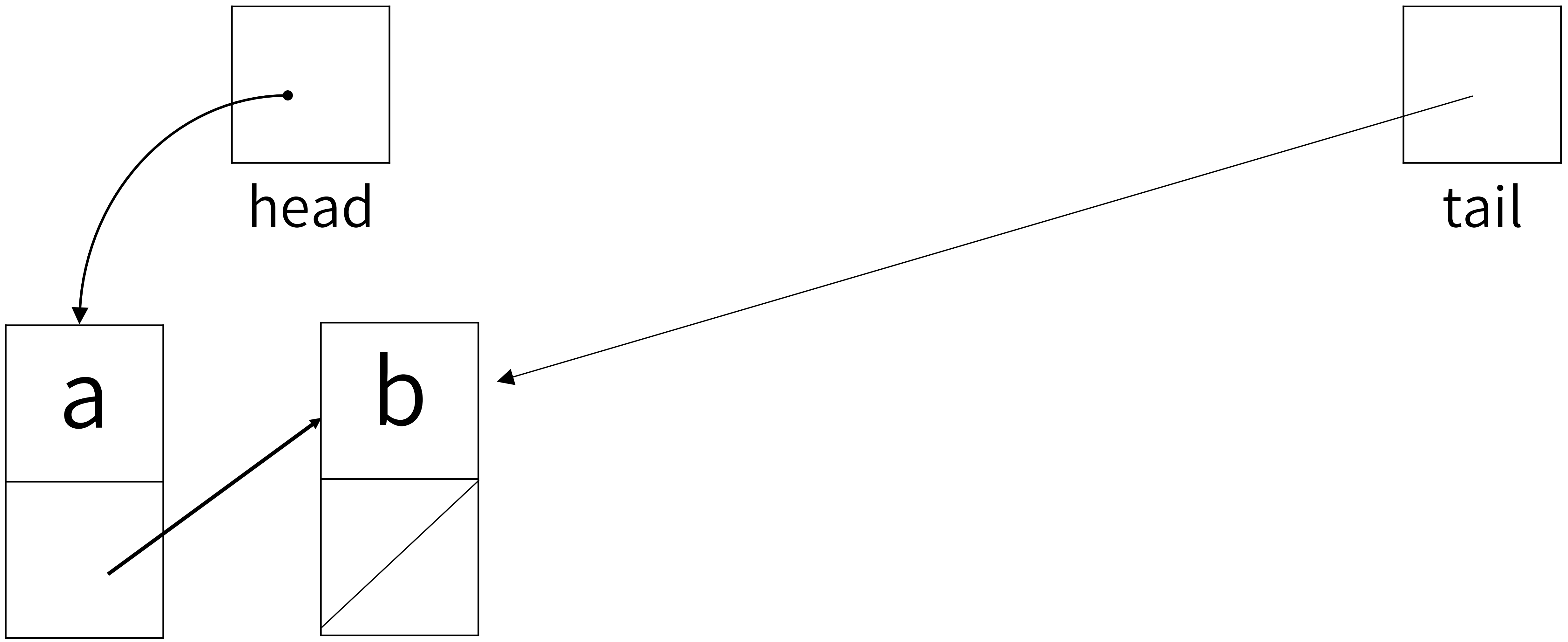


Queue Implementation with Linked List



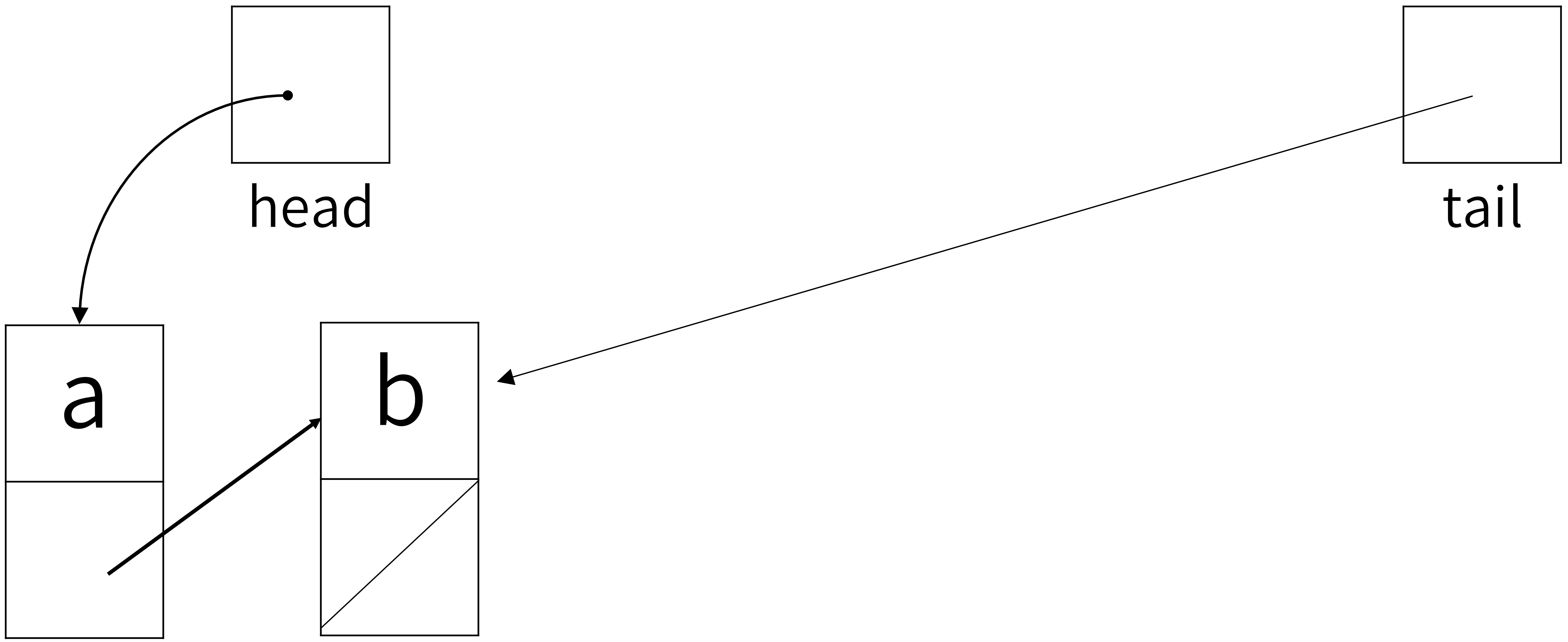
`Empty()`

Queue Implementation with Linked List

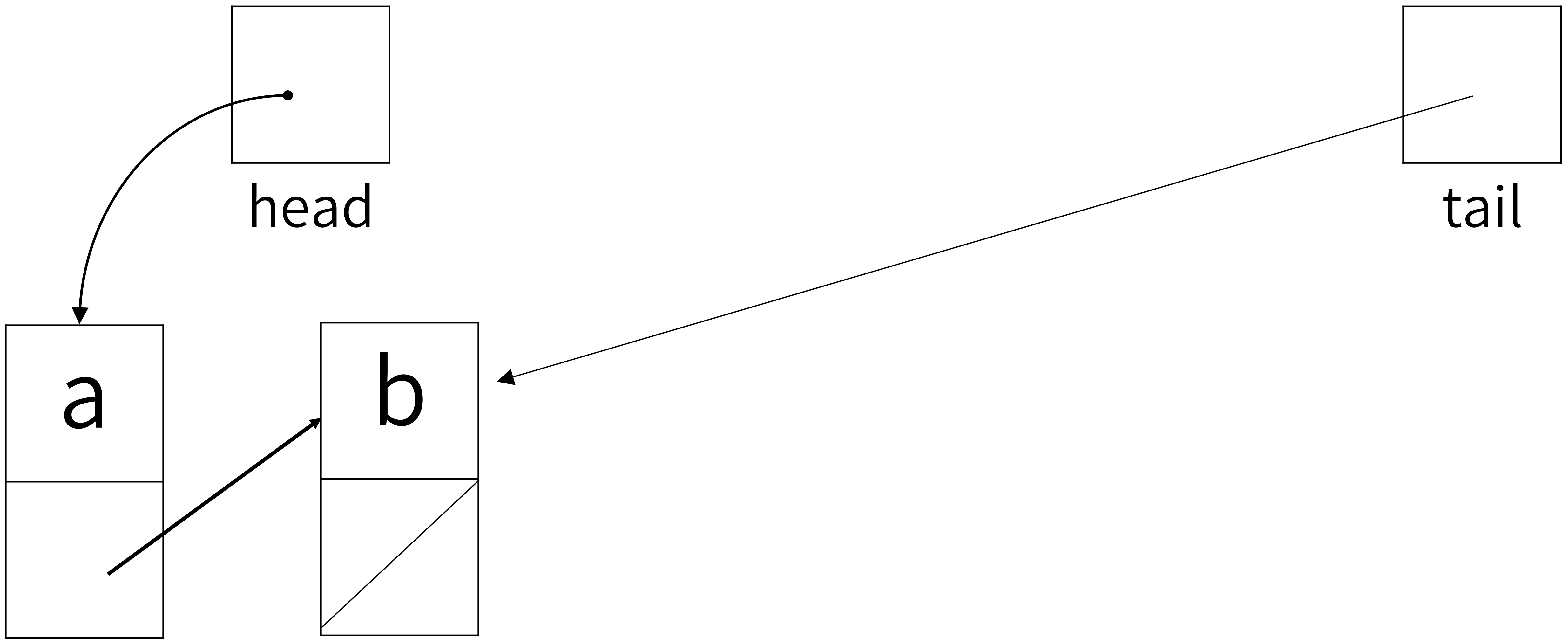


`Empty()→False`

Queue Implementation with Linked List

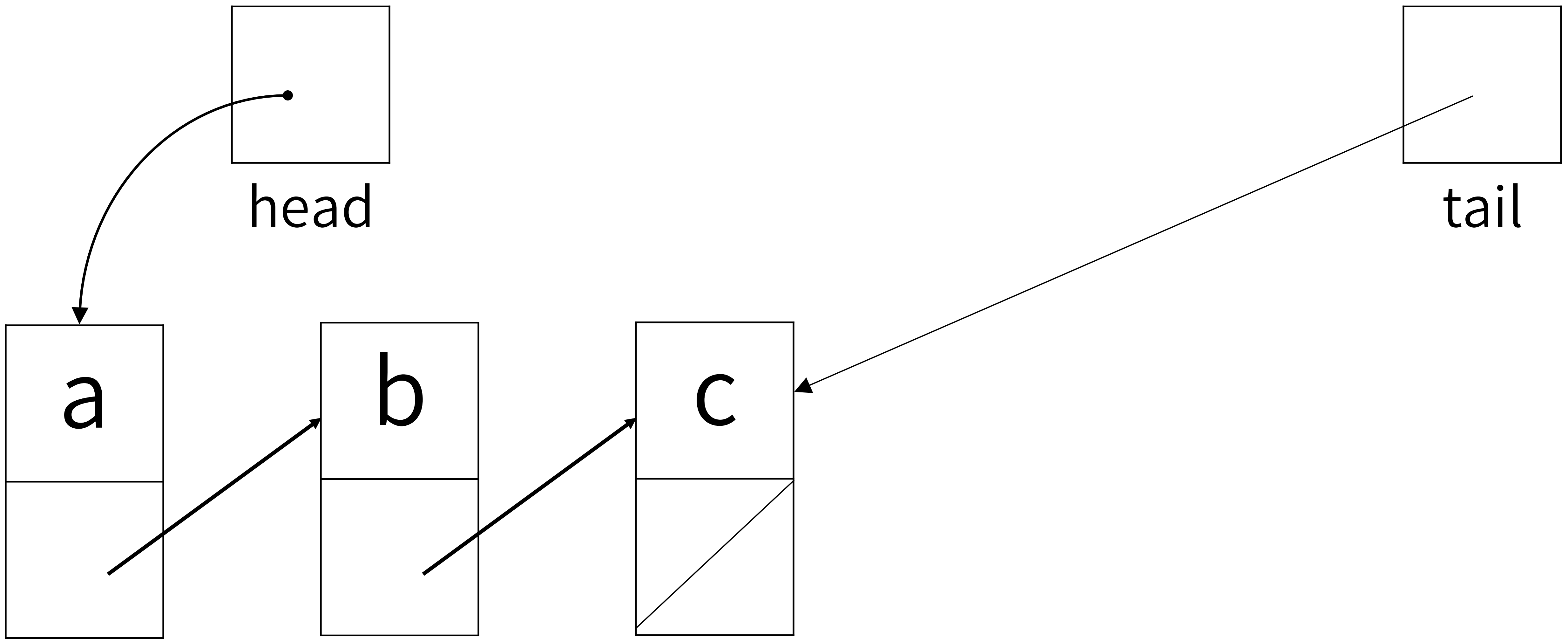


Queue Implementation with Linked List



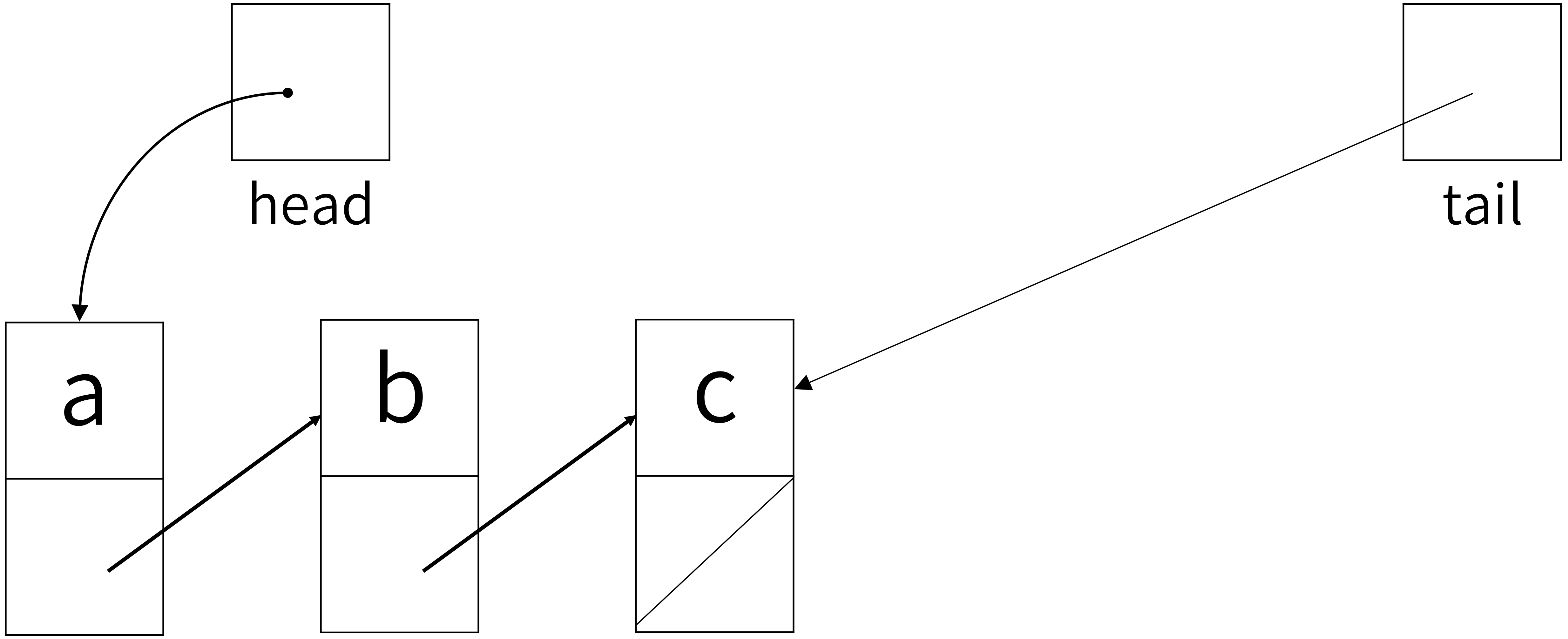
`Enqueue(c)`

Queue Implementation with Linked List

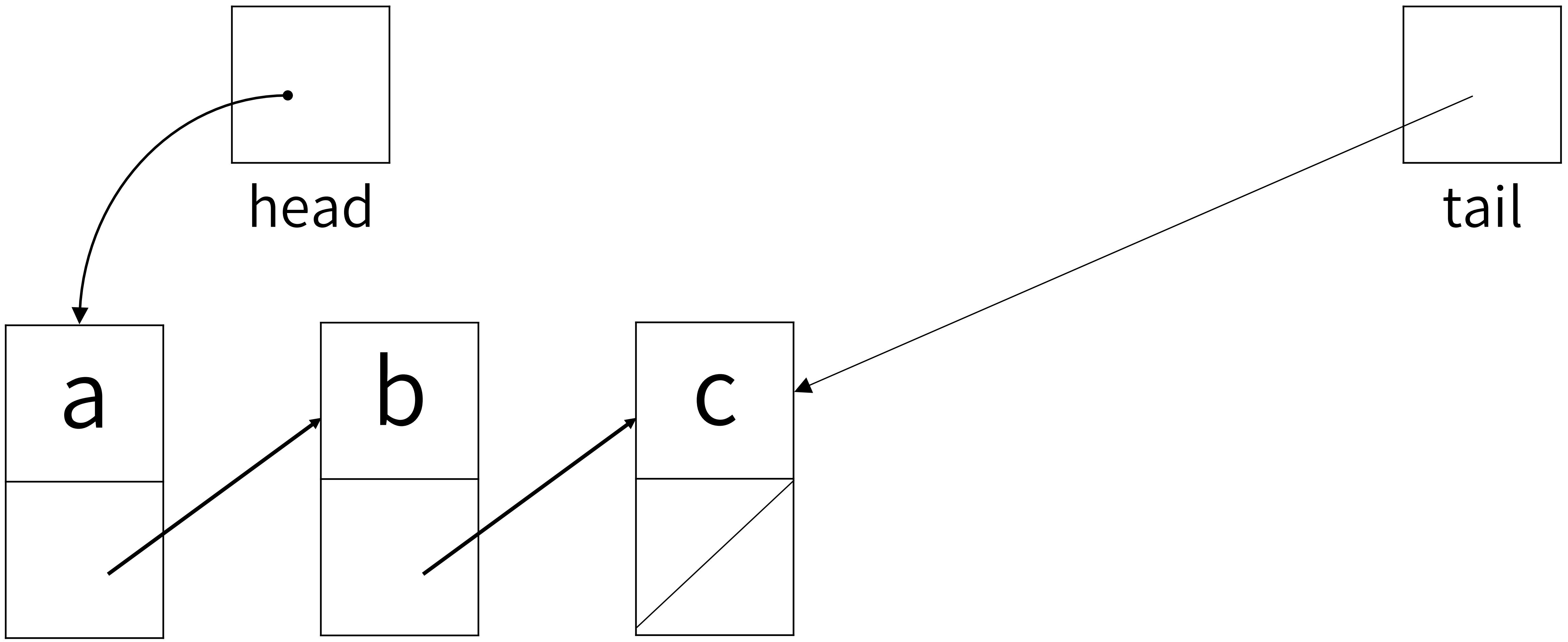


`Enqueue(c)`

Queue Implementation with Linked List

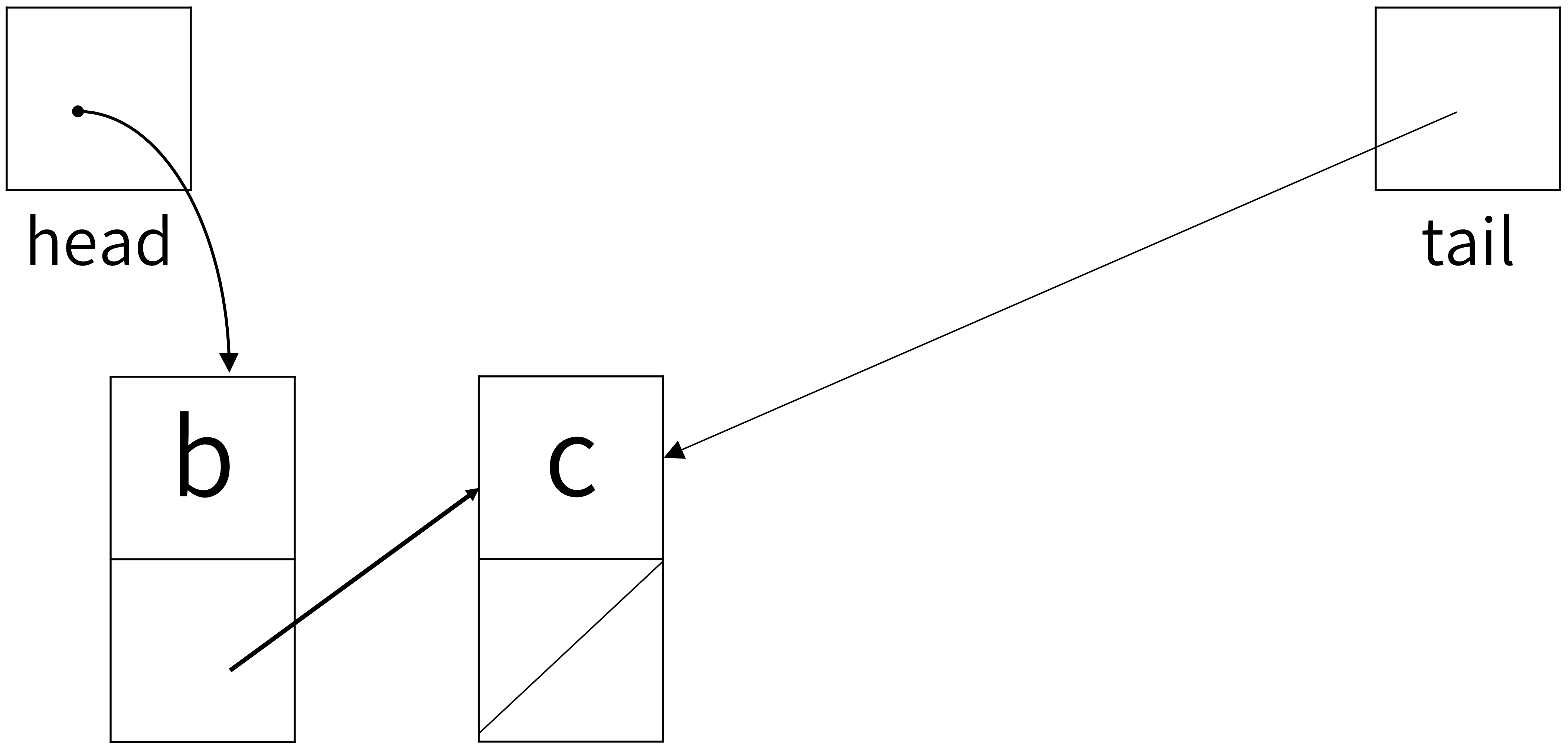


Queue Implementation with Linked List



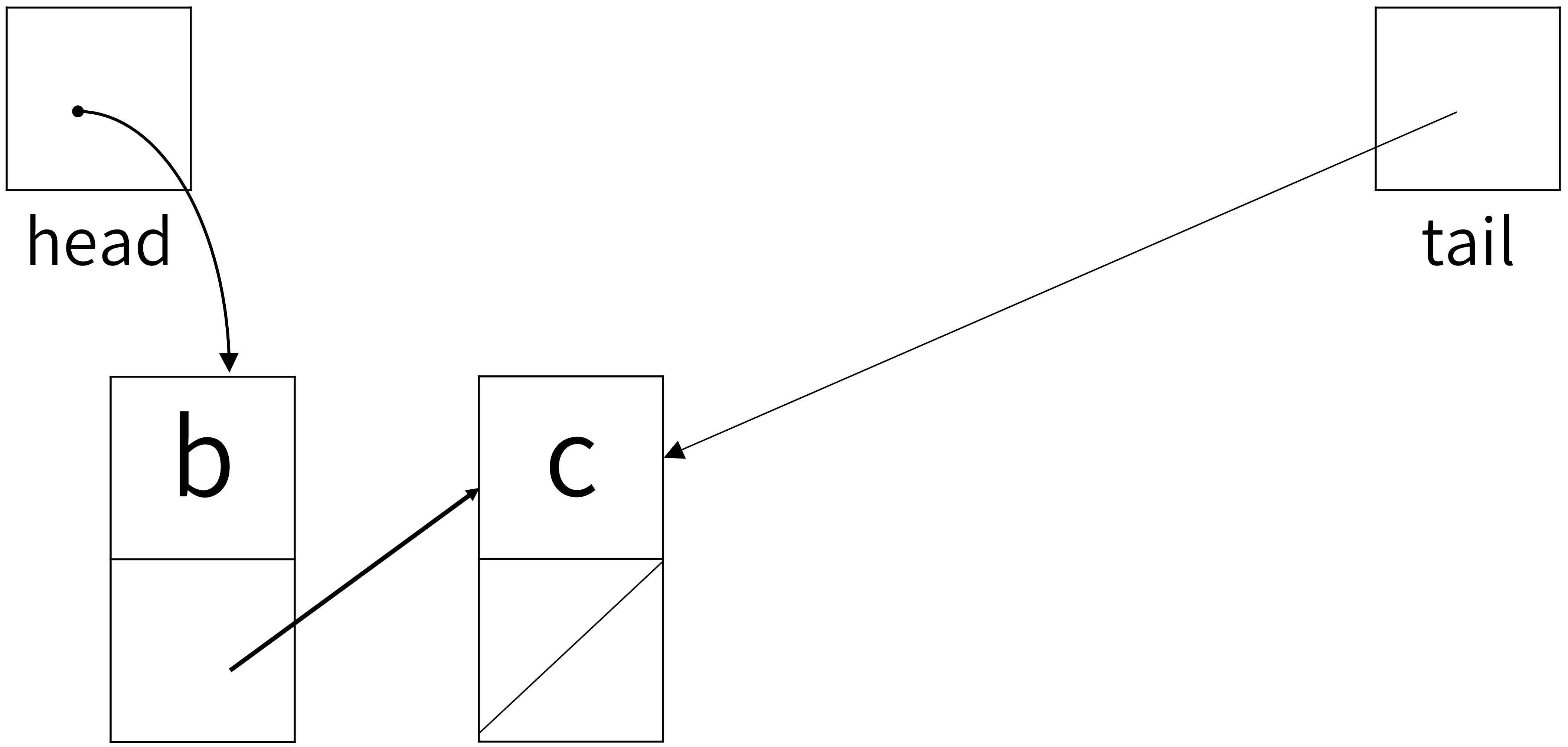
`Dequeue()`

Queue Implementation with Linked List

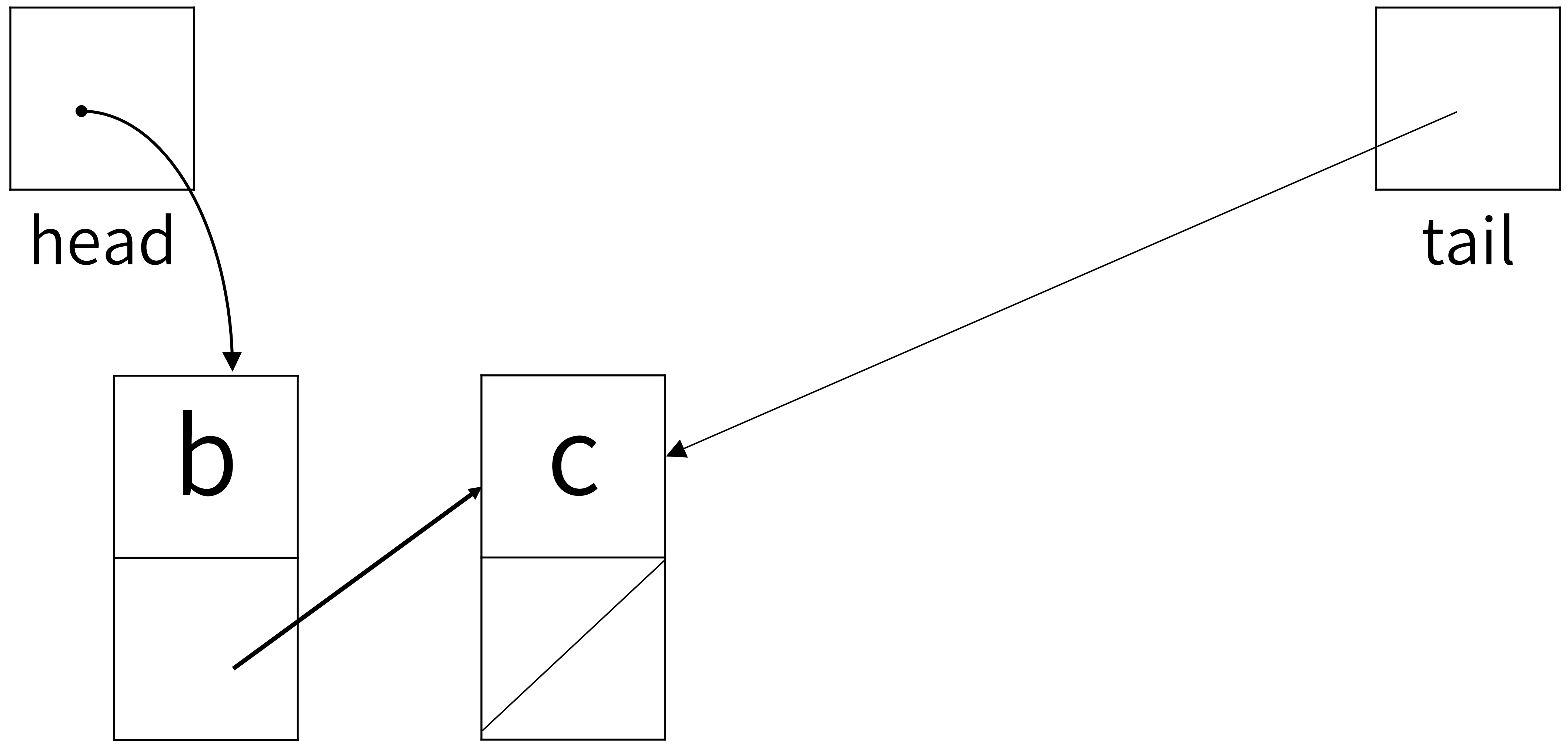


`Dequeue()→a`

Queue Implementation with Linked List

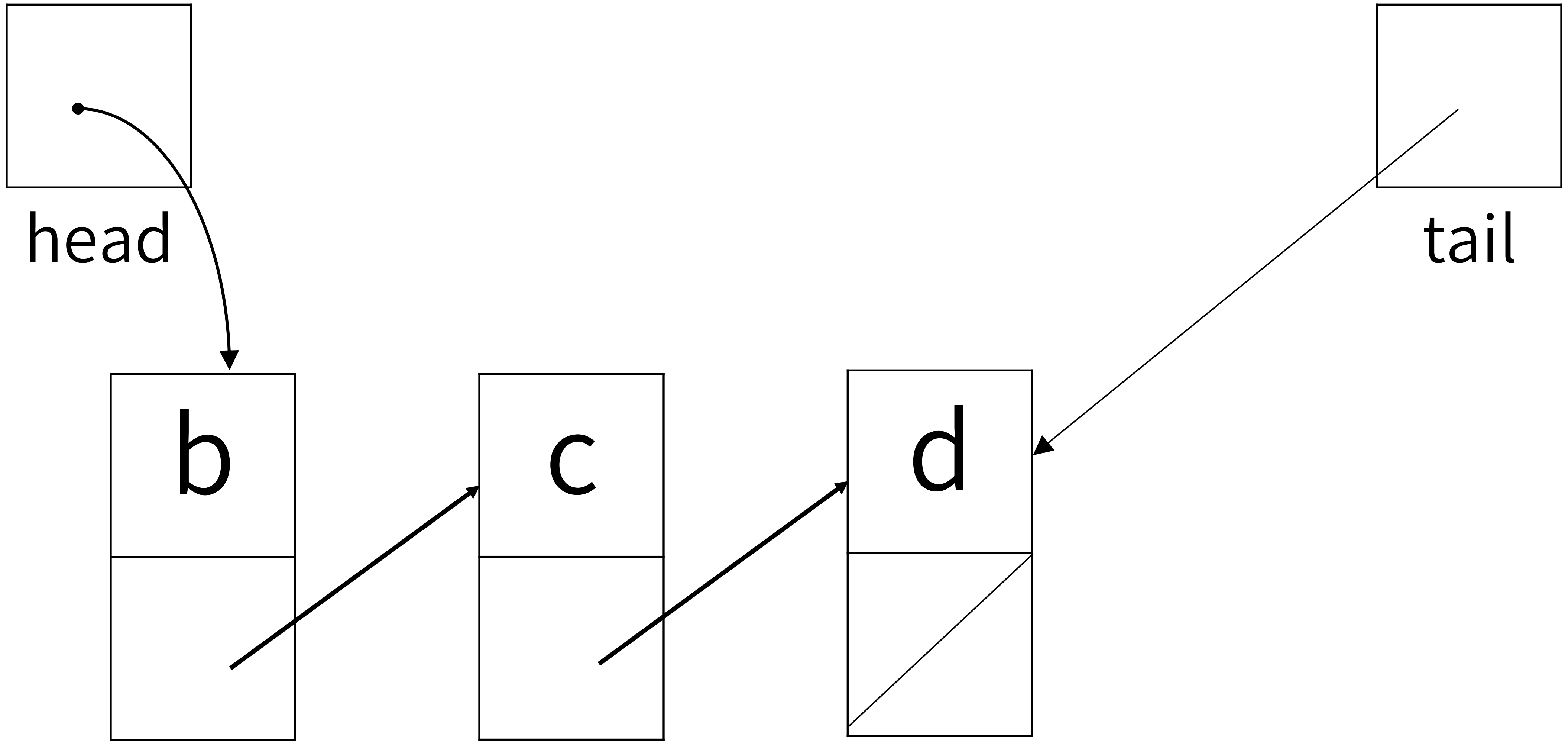


Queue Implementation with Linked List



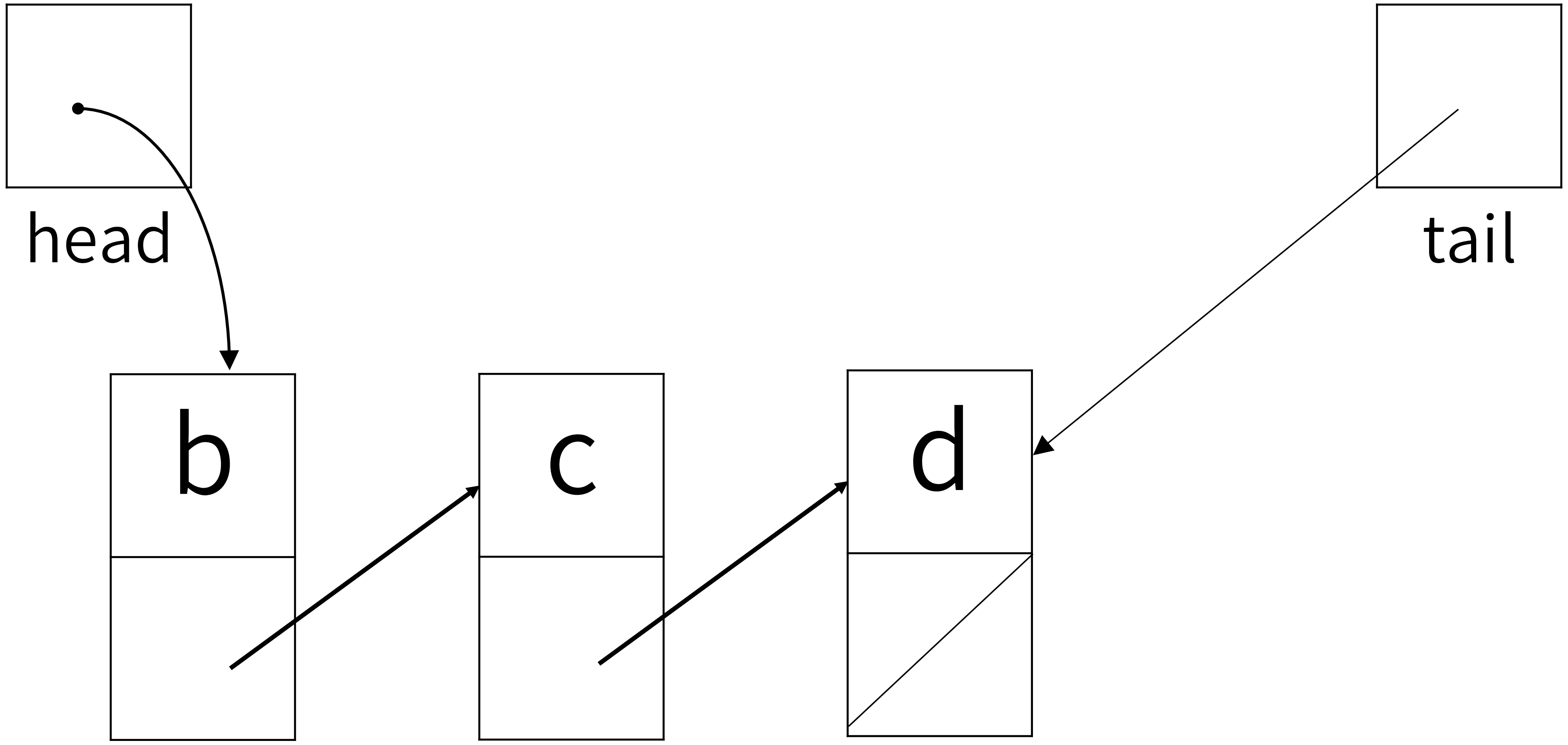
`Enqueue(d)`

Queue Implementation with Linked List

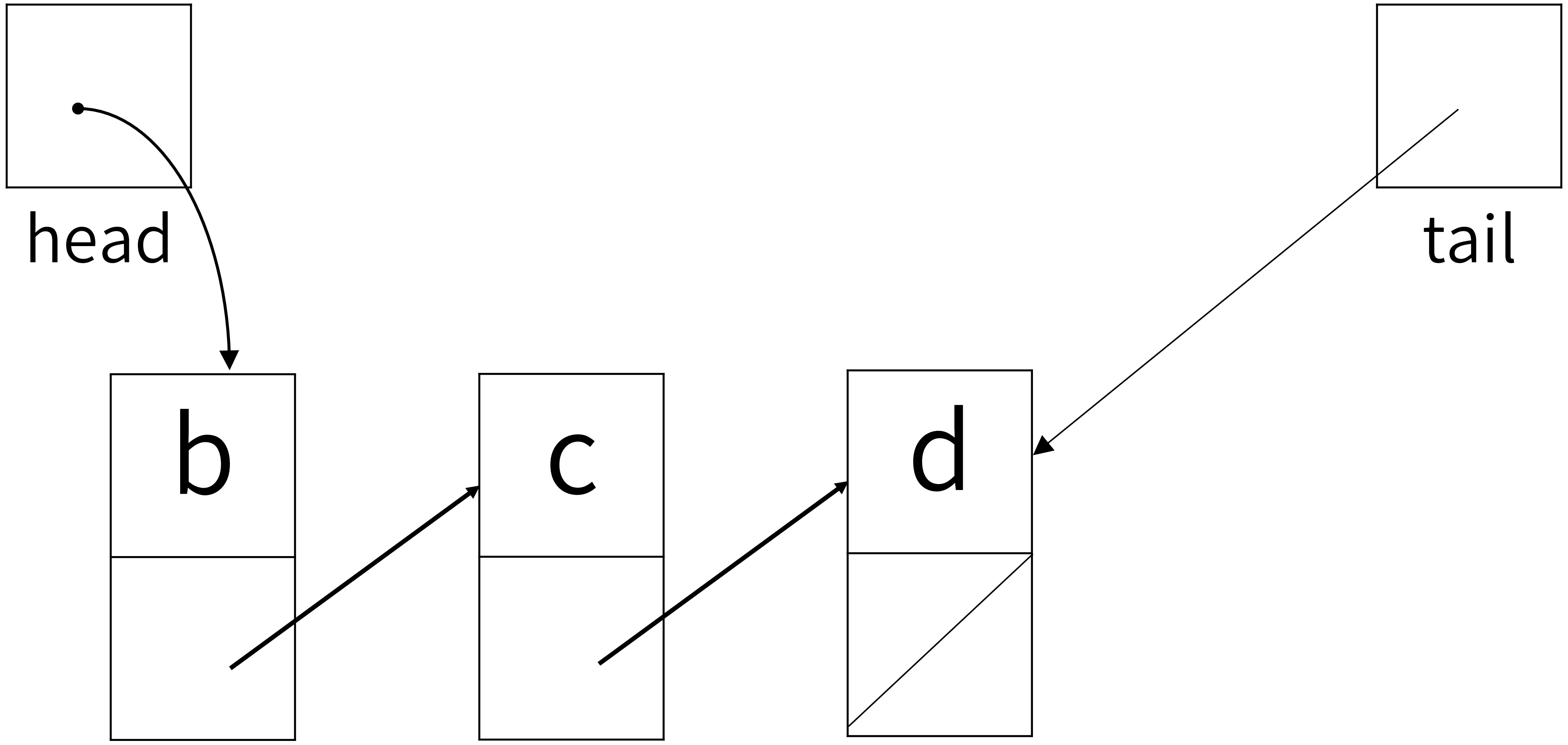


Enqueue(d)

Queue Implementation with Linked List

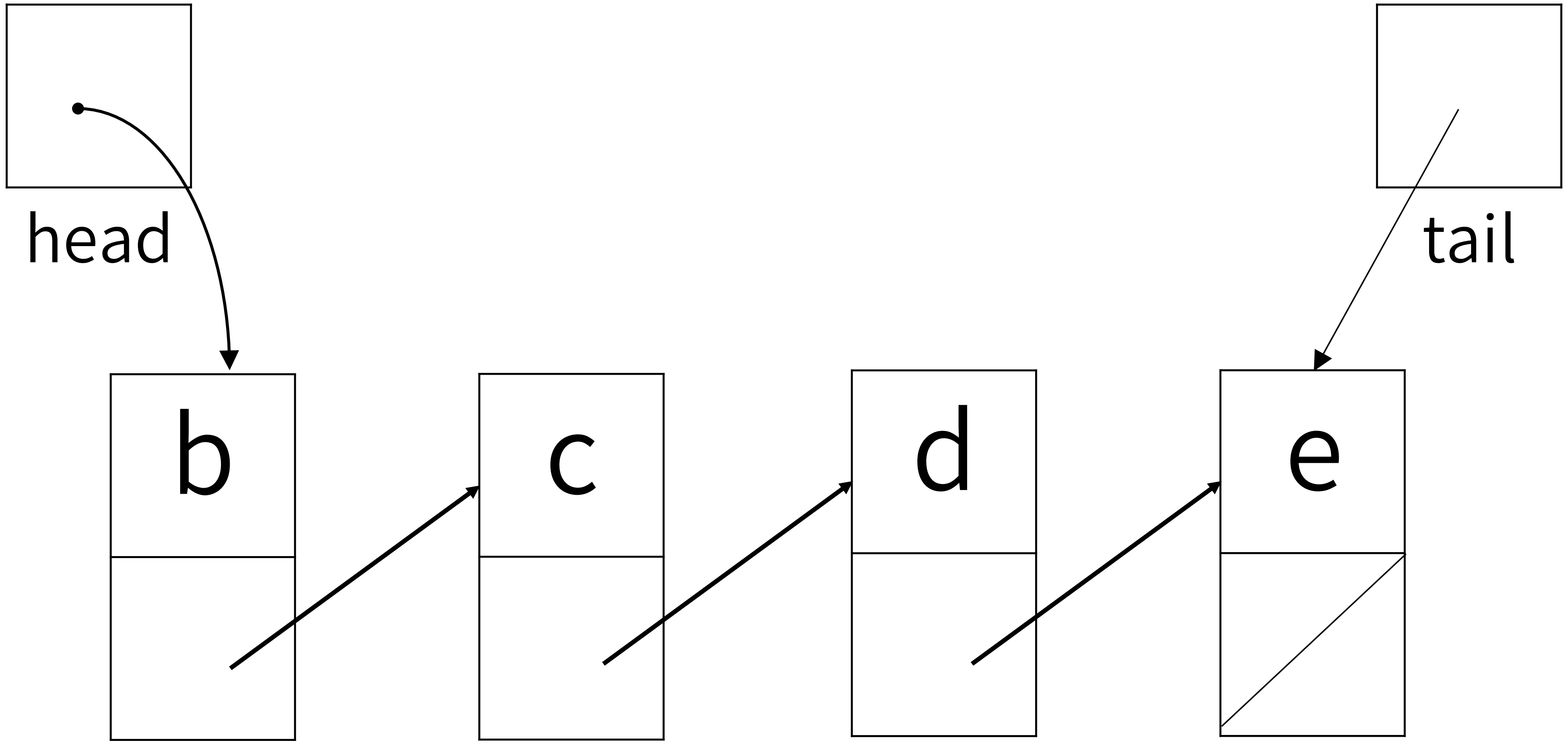


Queue Implementation with Linked List



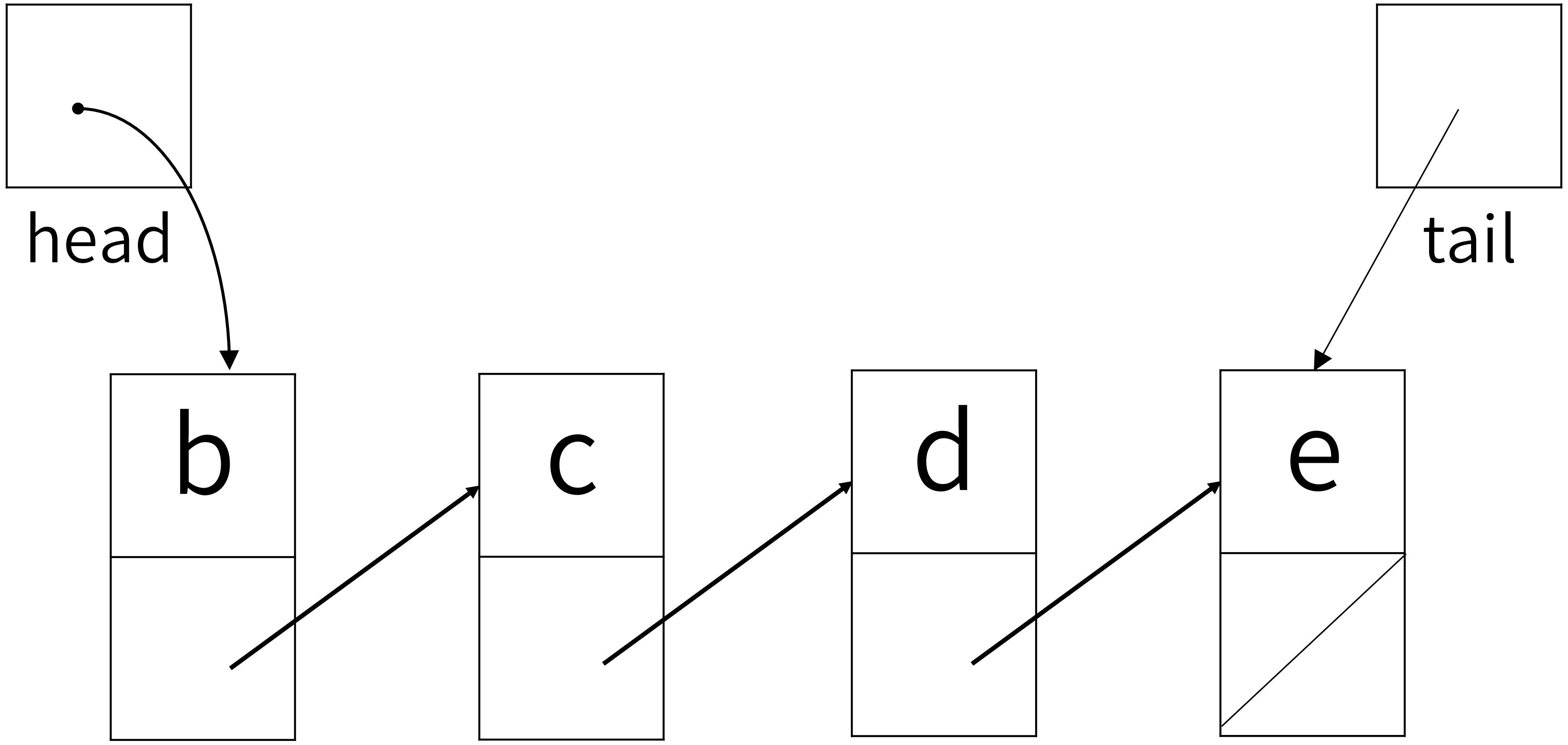
Enqueue(e)

Queue Implementation with Linked List

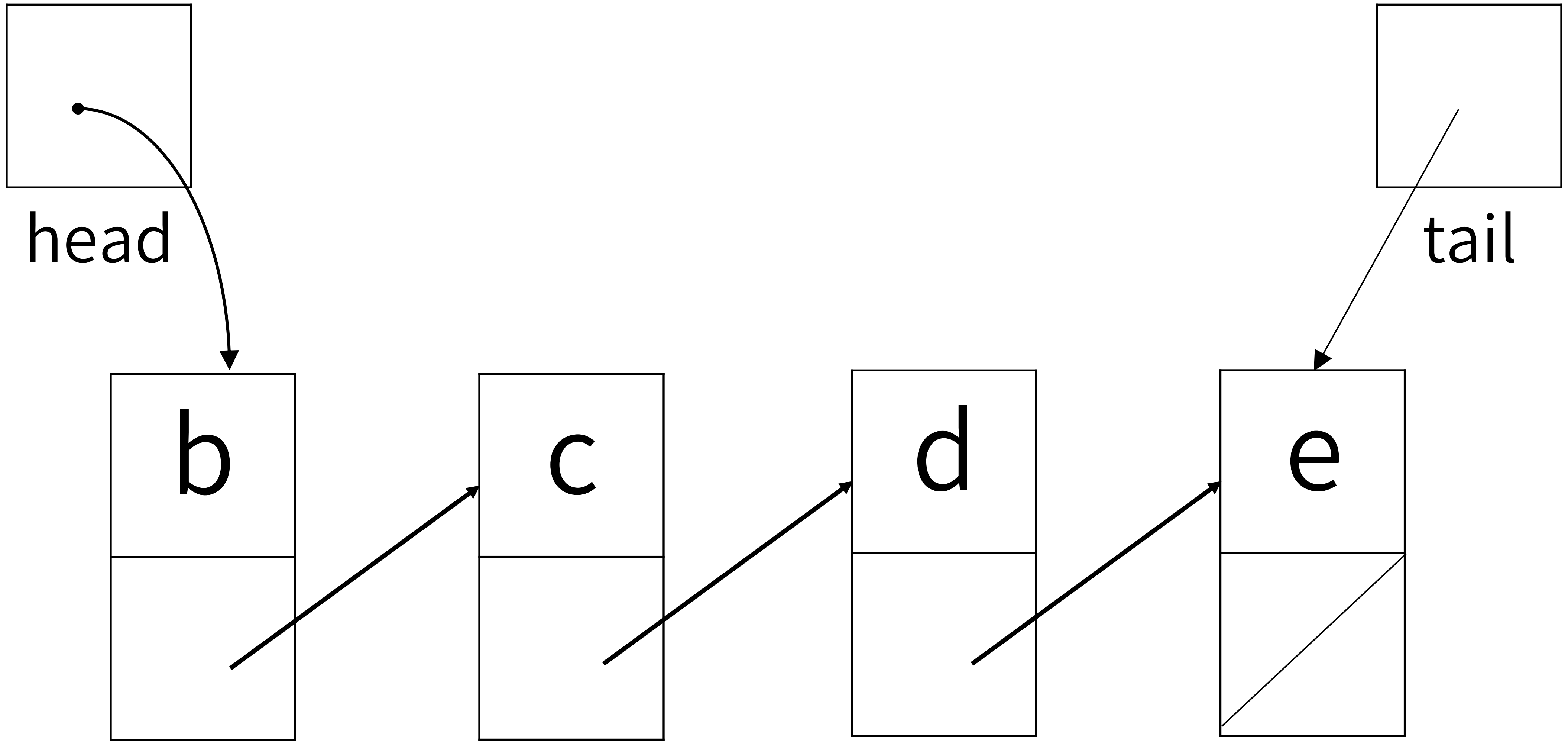


Enqueue(e)

Queue Implementation with Linked List

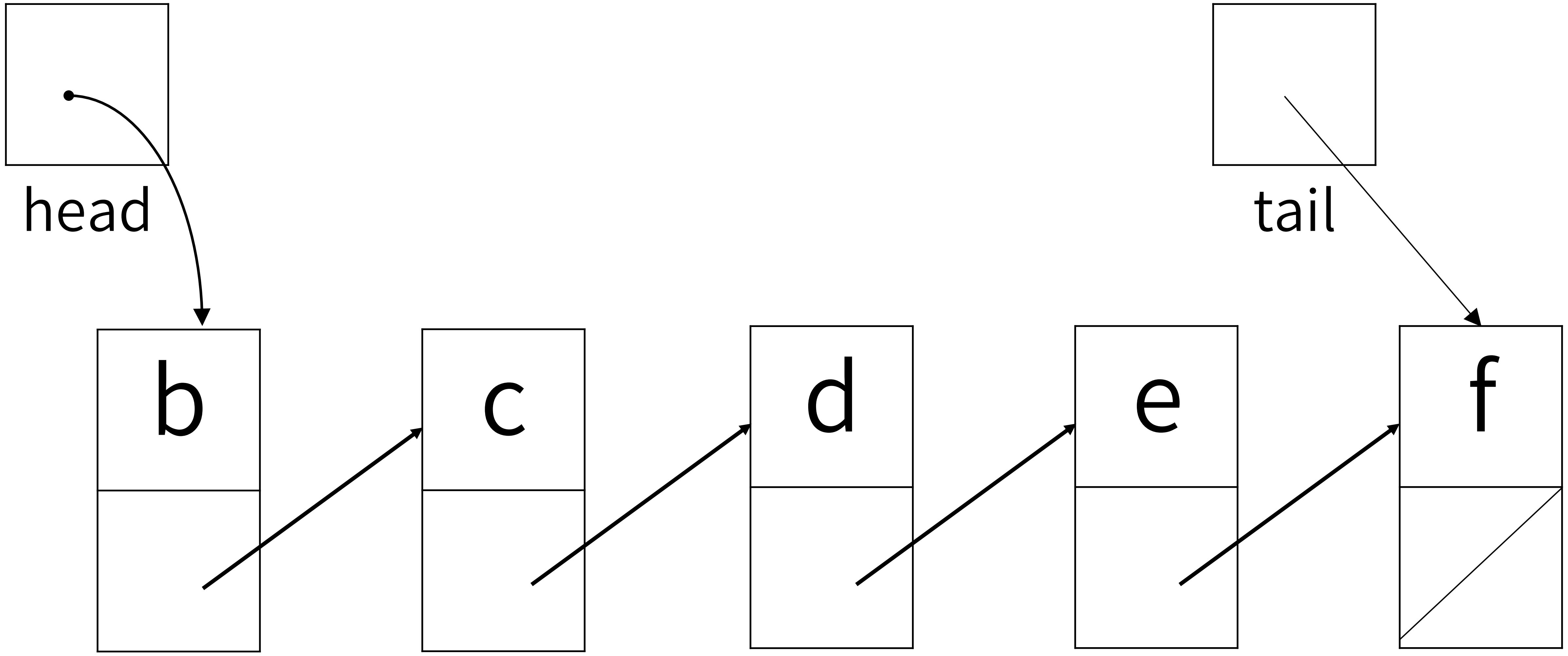


Queue Implementation with Linked List



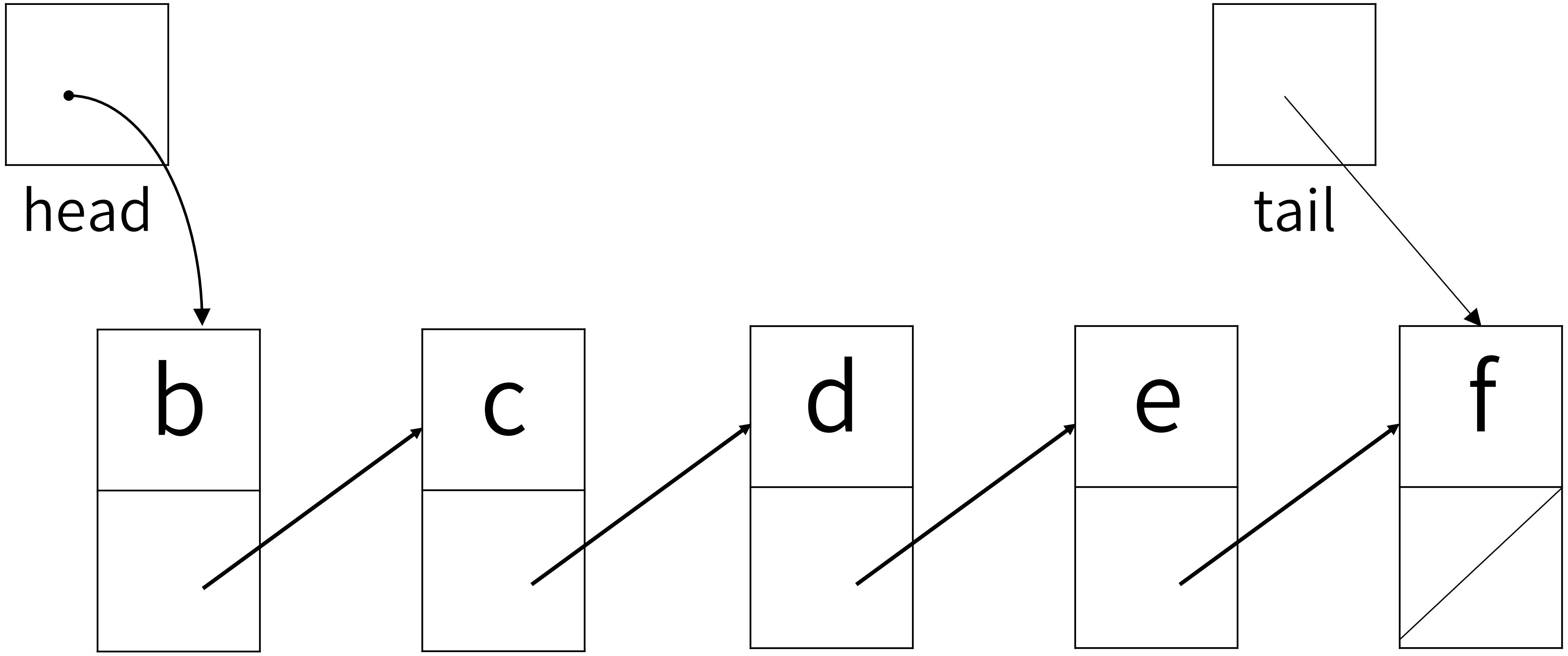
Enqueue(f)

Queue Implementation with Linked List

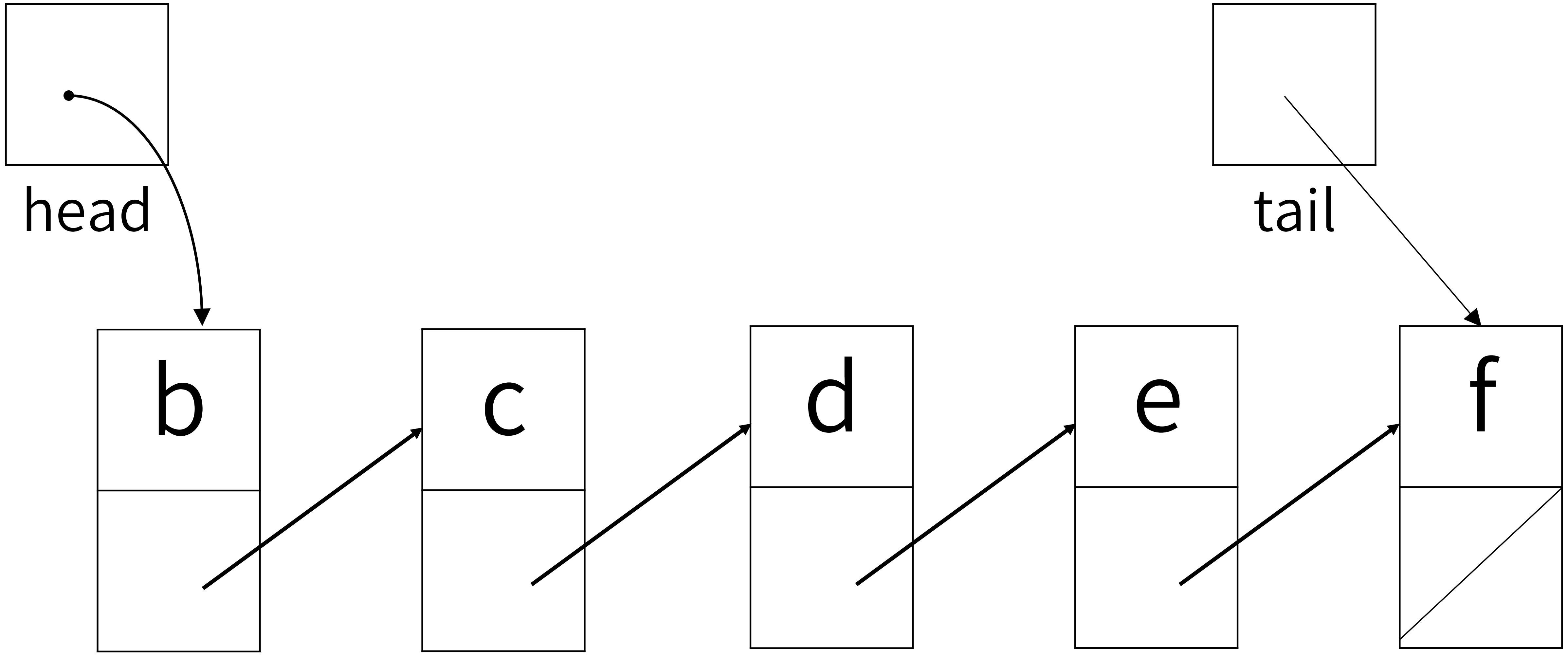


Enqueue(f)

Queue Implementation with Linked List

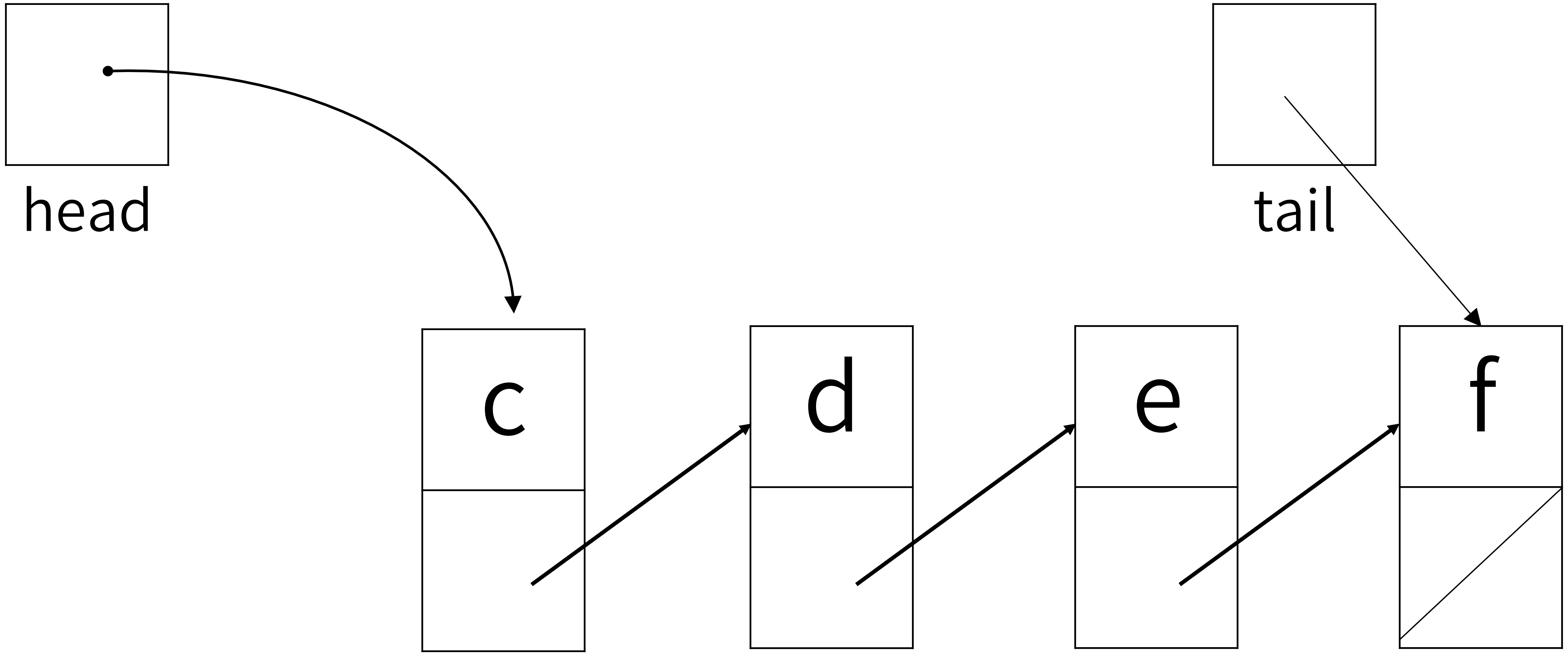


Queue Implementation with Linked List



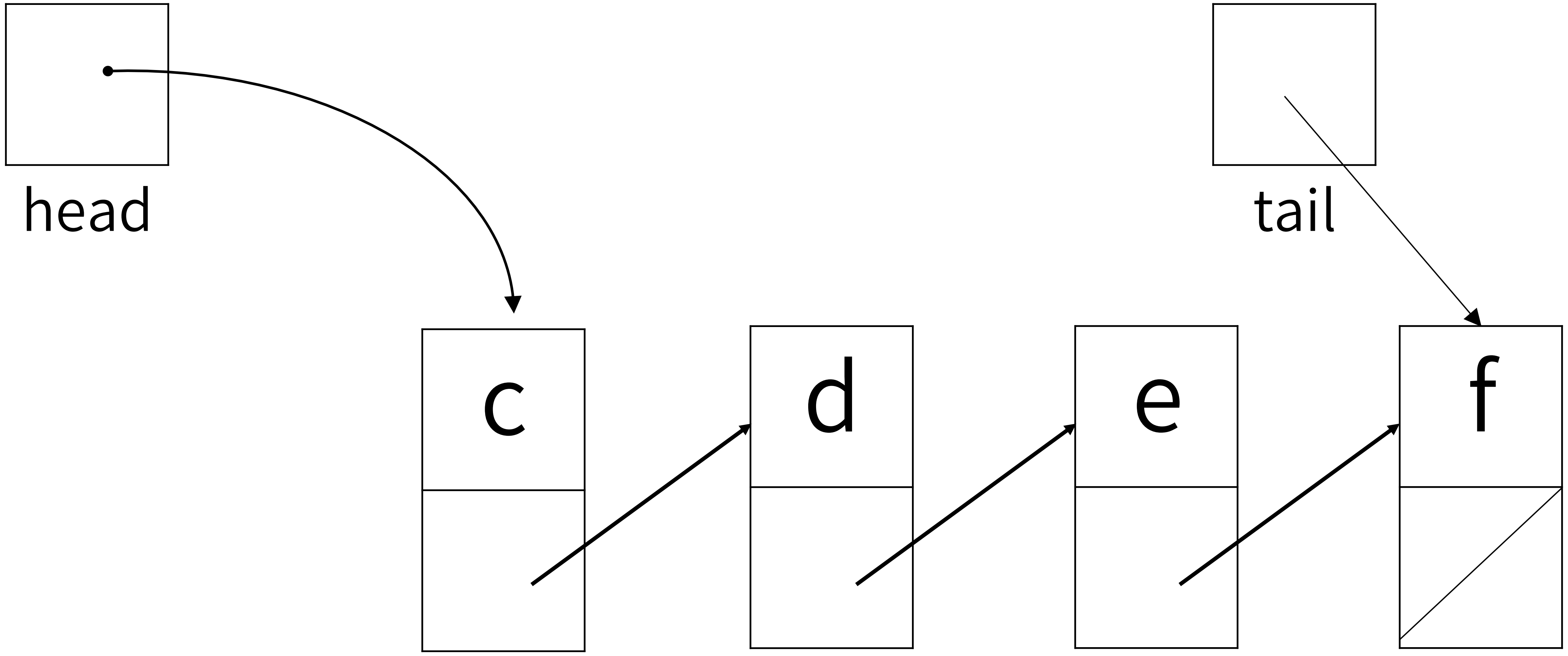
`Dequeue()`

Queue Implementation with Linked List



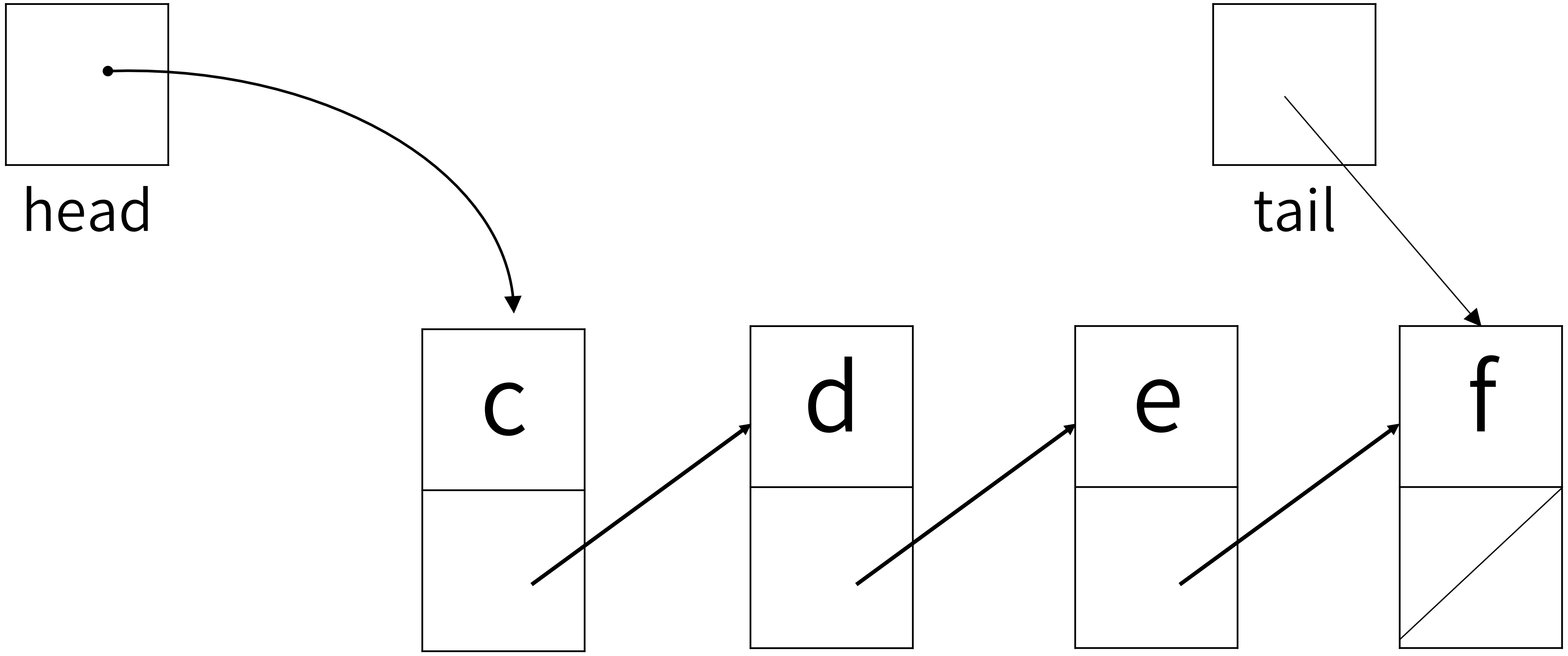
`Dequeue()`→b

Queue Implementation with Linked List



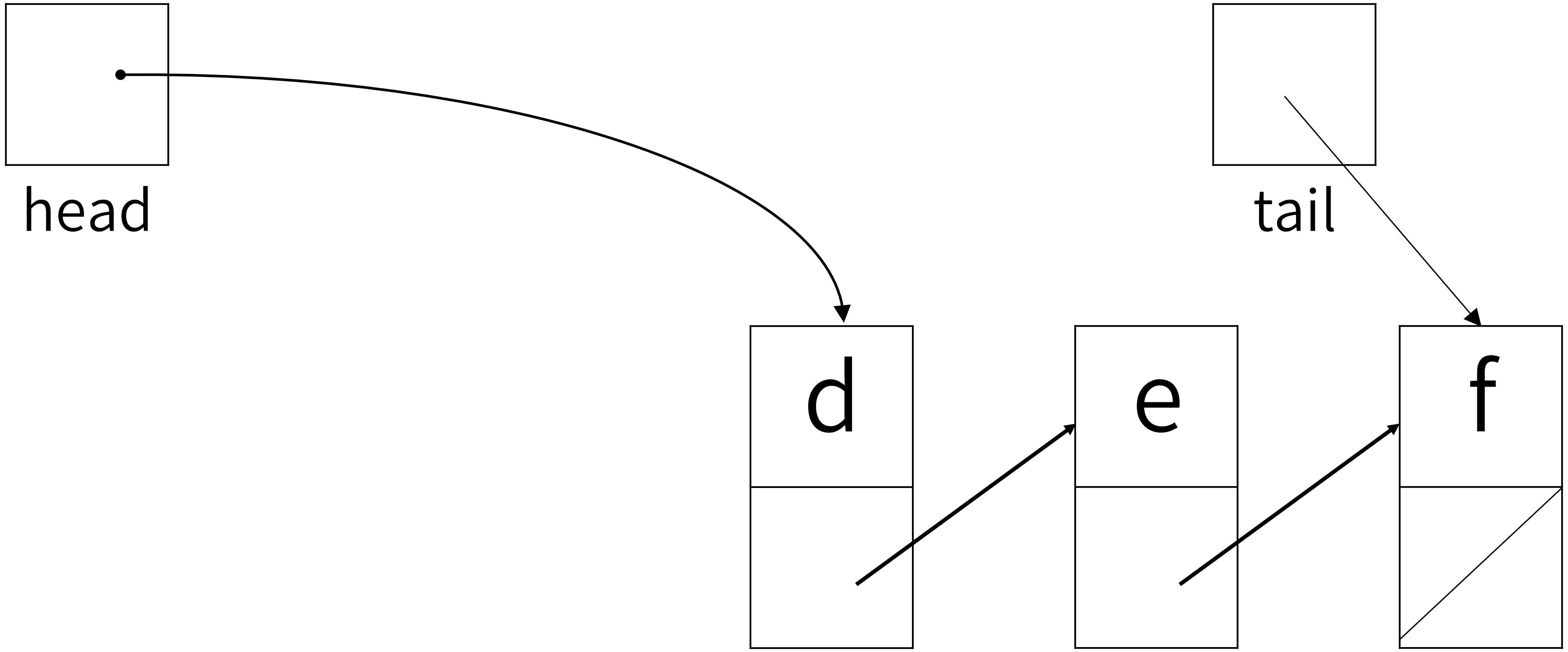
Dequeue()→b

Queue Implementation with Linked List



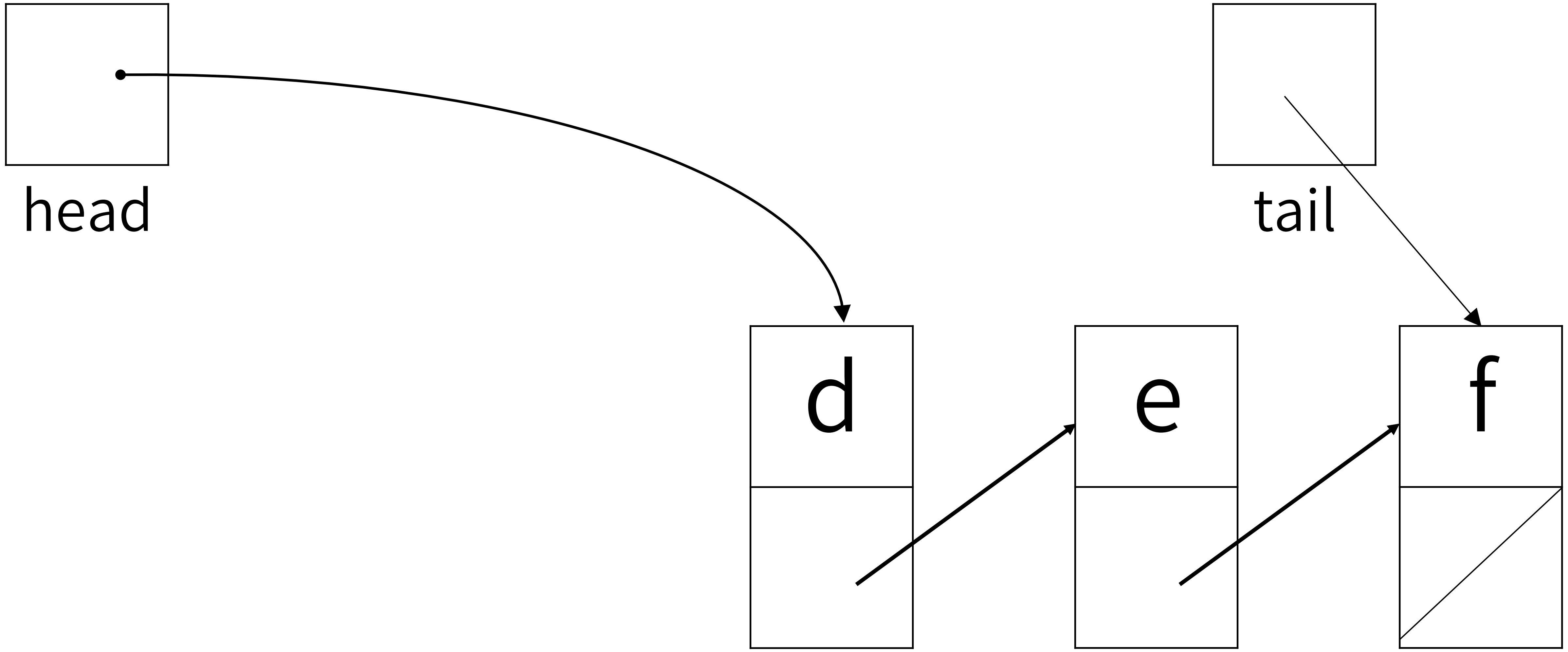
Dequeue()

Queue Implementation with Linked List

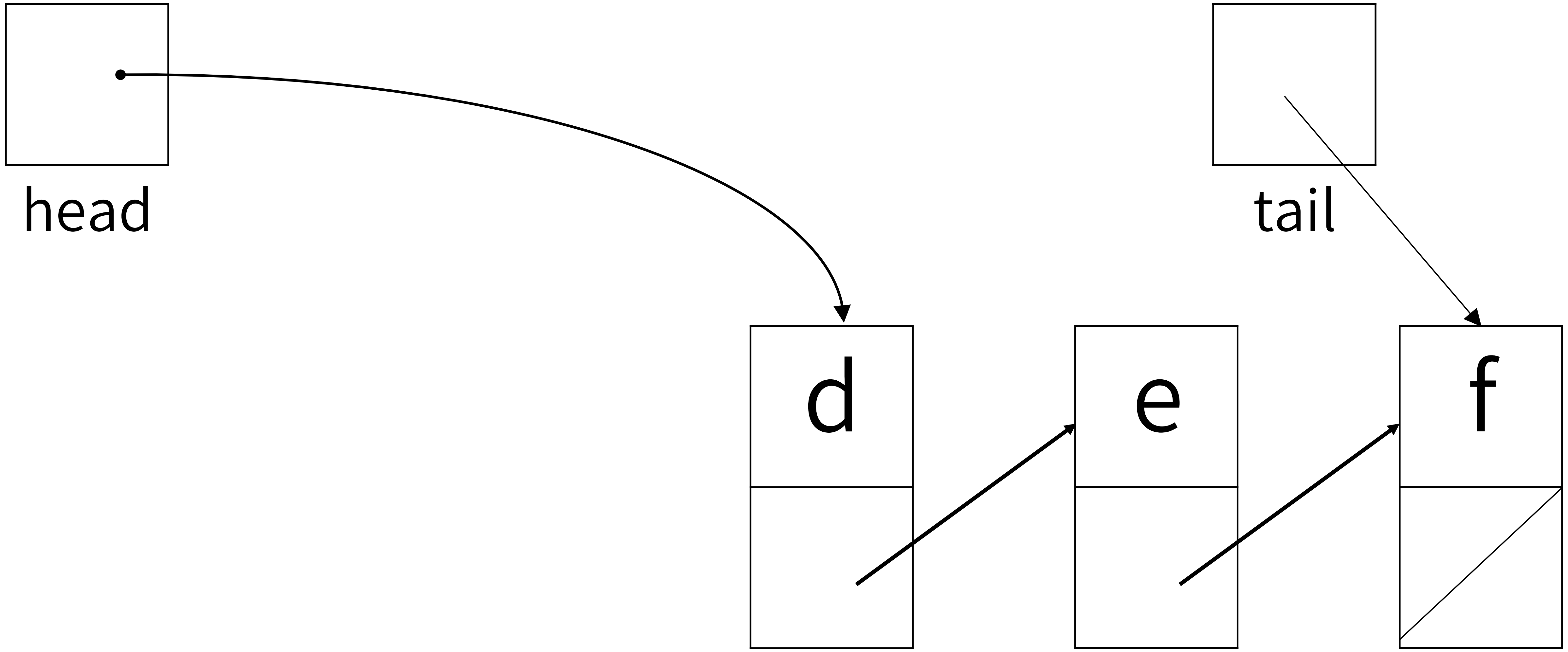


`Dequeue()→c`

Queue Implementation with Linked List

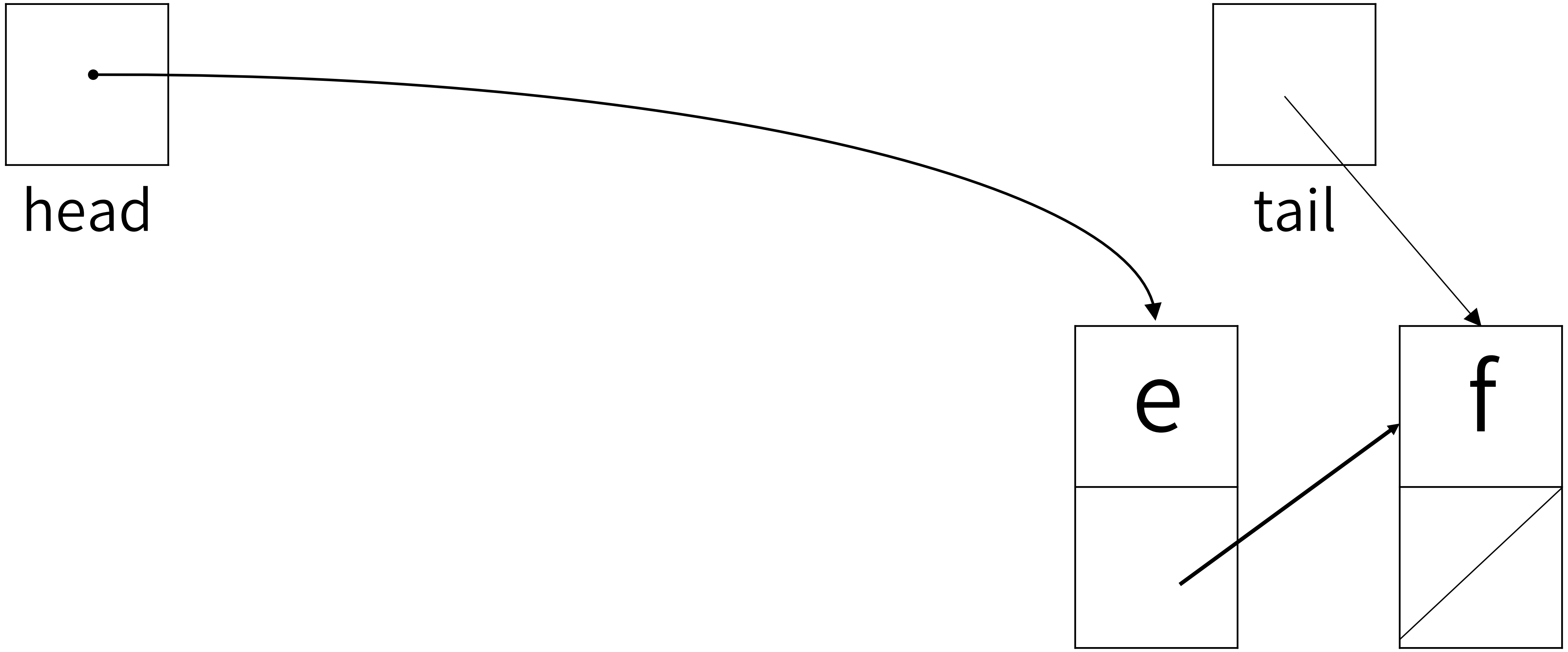


Queue Implementation with Linked List



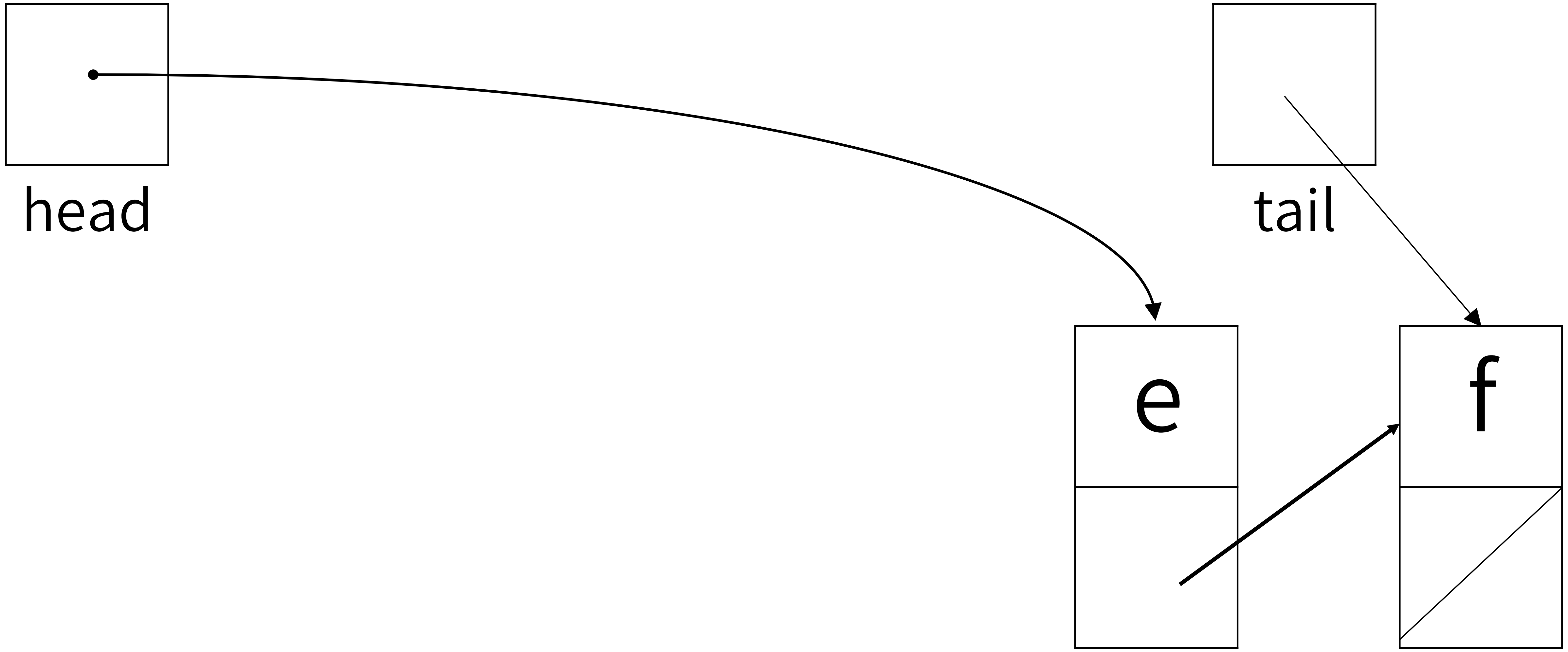
`Dequeue()`

Queue Implementation with Linked List

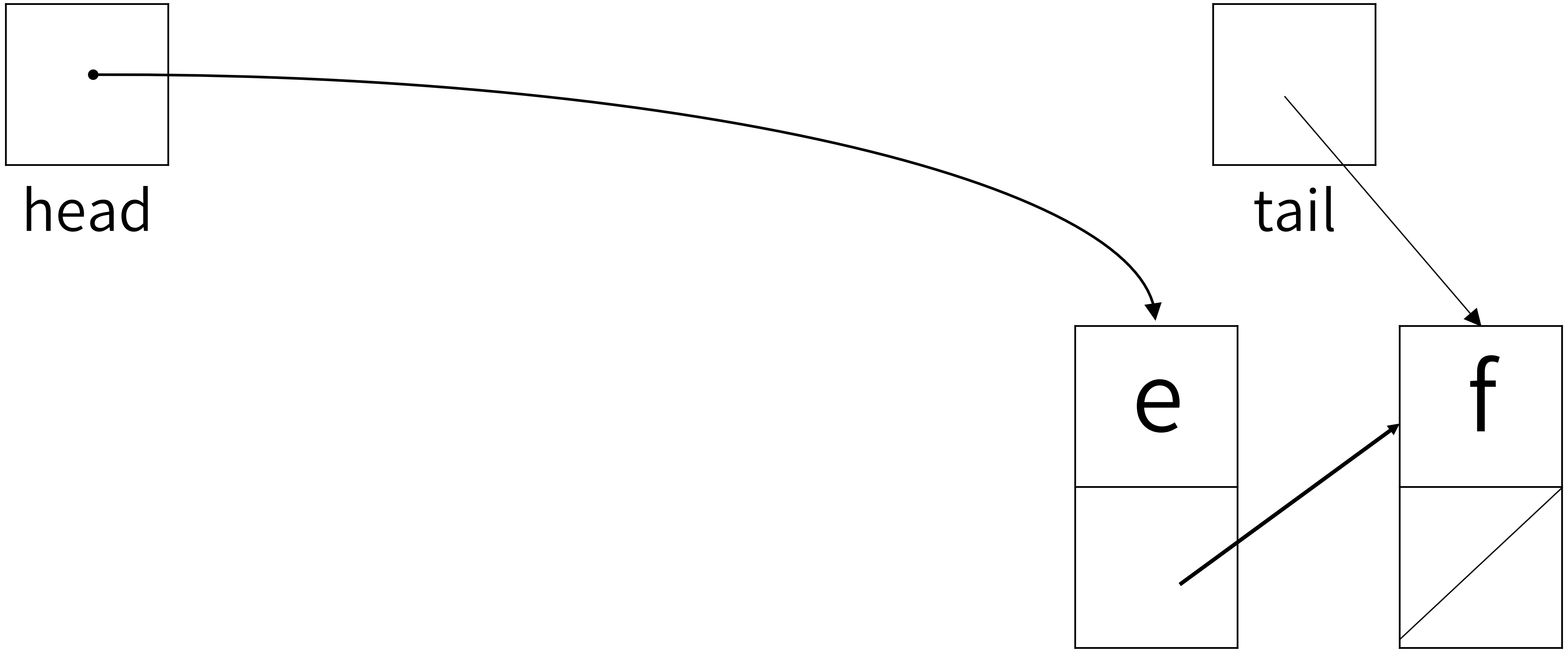


`Dequeue()→d`

Queue Implementation with Linked List

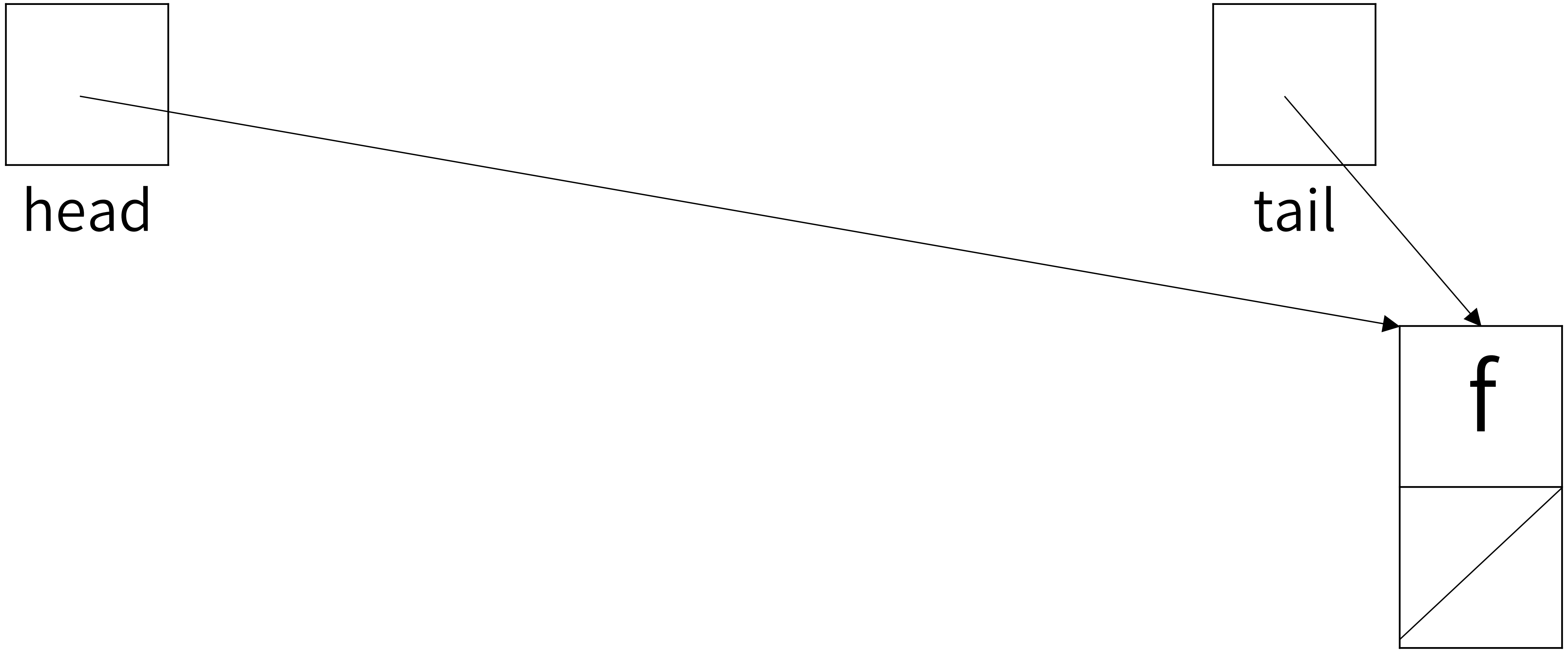


Queue Implementation with Linked List



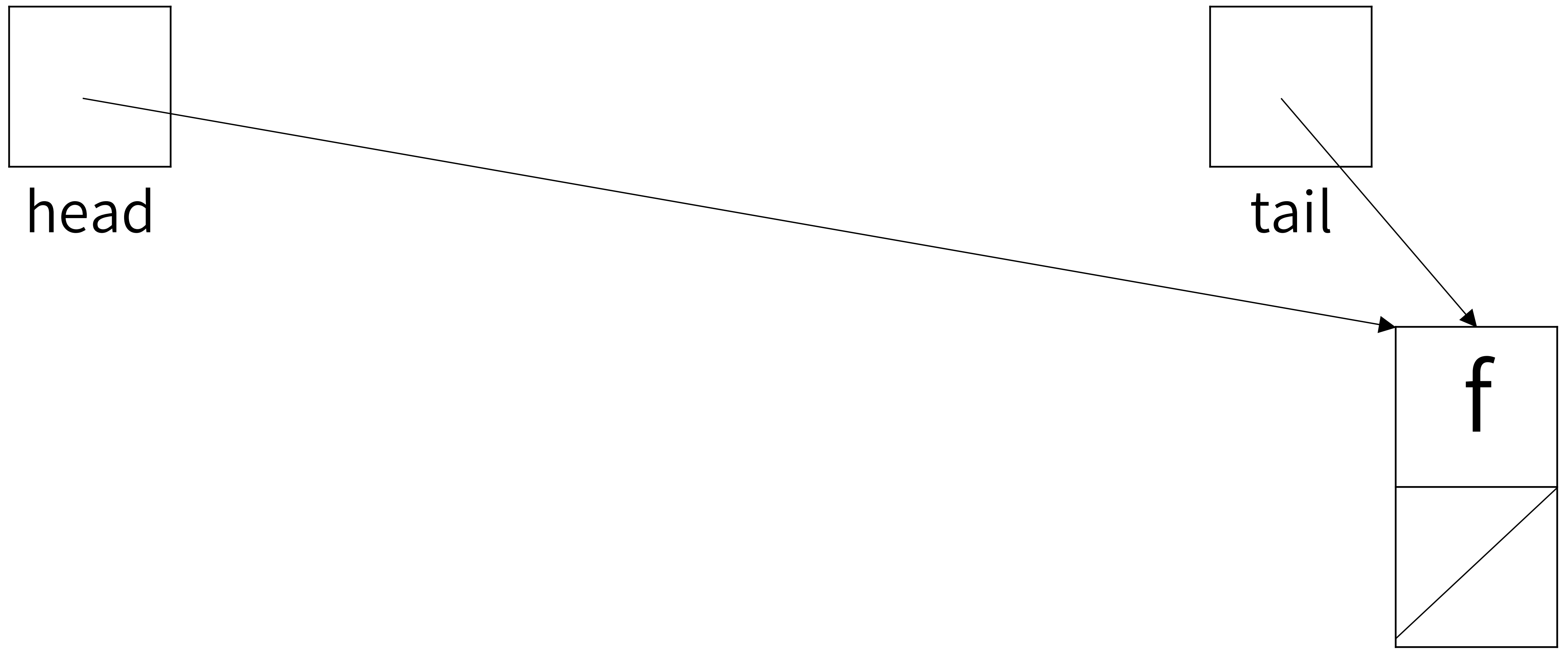
`Dequeue()`

Queue Implementation with Linked List

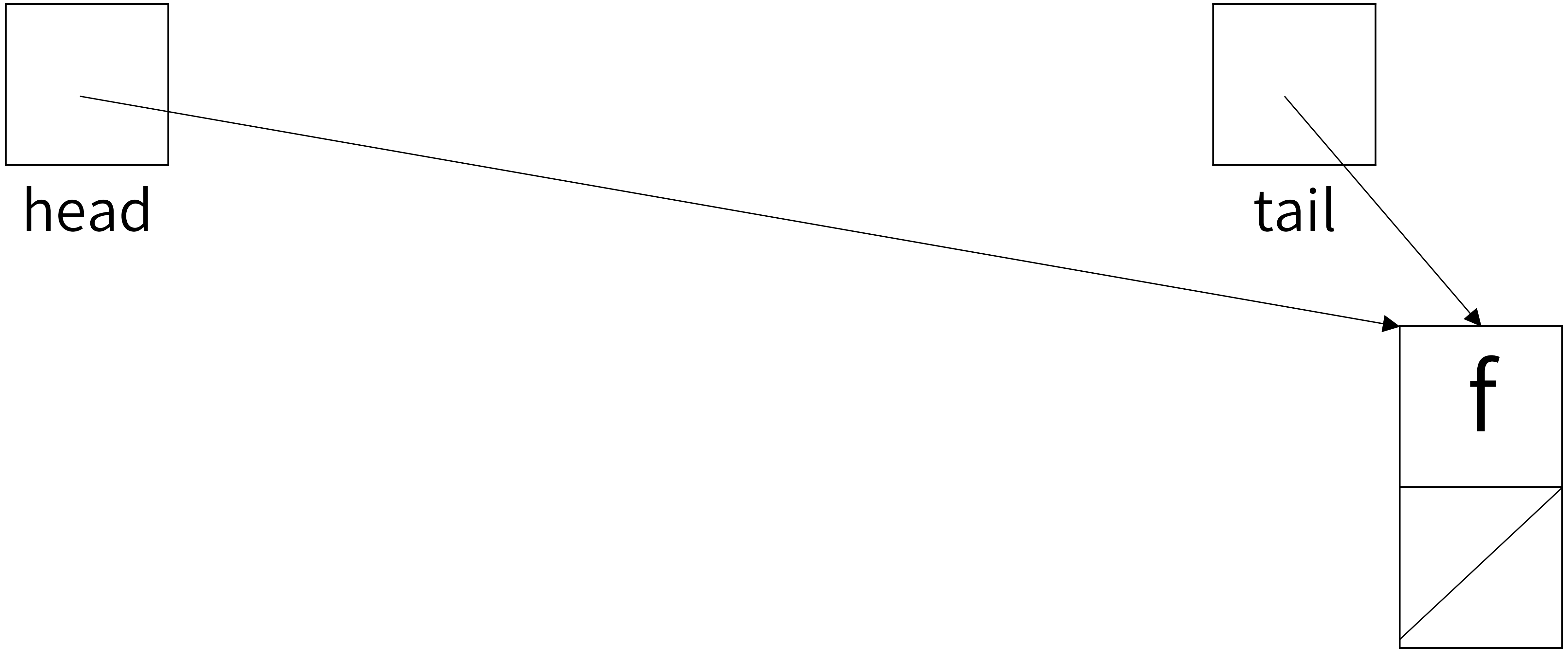


`Dequeue()` →`e`

Queue Implementation with Linked List

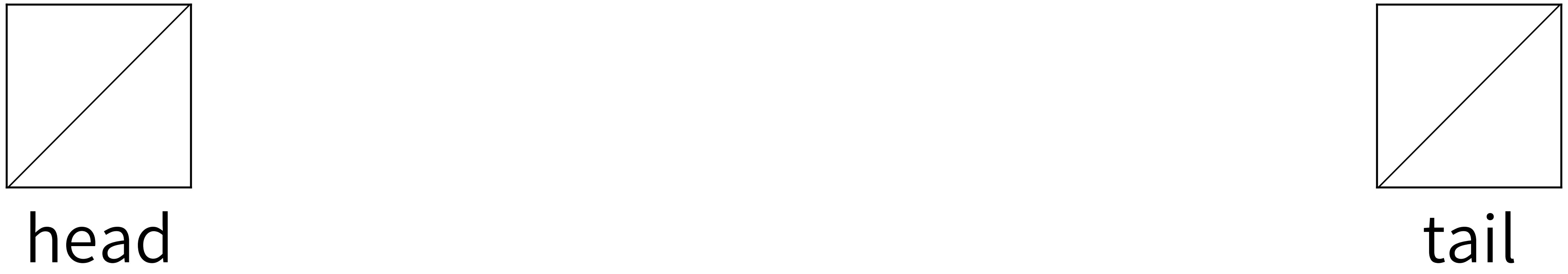


Queue Implementation with Linked List



Dequeue()

Queue Implementation with Linked List

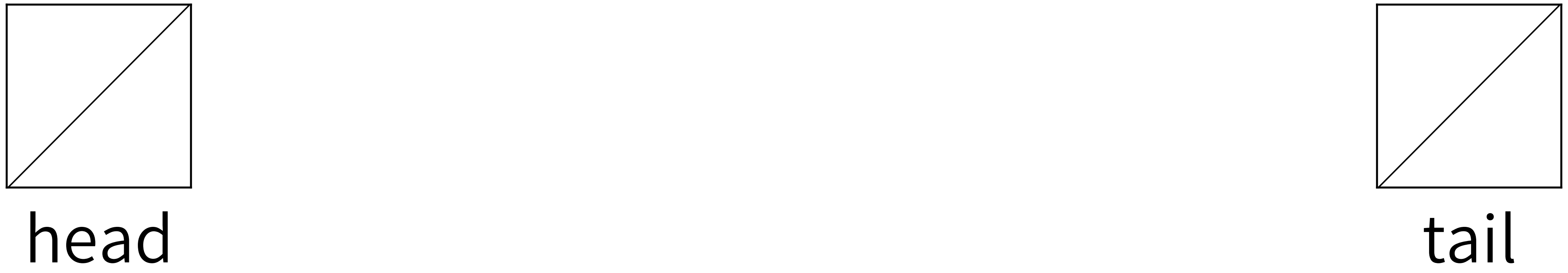


`Dequeue()→f`

Queue Implementation with Linked List

`Empty()`

Queue Implementation with Linked List



head

tail

`Empty() → True`

Summary

- Queues can be implemented with either an array or a linked list (with tail pointer).
- Each stack operation is O(1):Enqueue, dequeue, empty.
- Implementation with linked list:
  - `Enqueue : PushBack()`
  - `Dequeue : TopFront() + PopFront()`
- Stacks are occasionally known as FIFO queues.