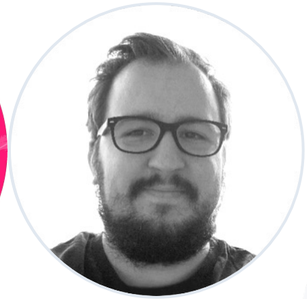
Using Specialized Collections



Mateo Prigl
Software Developer

Handling Missing Dictionary Keys

`try-except` Block

Catch errors gracefully

`in` Membership Check

Pre-check existence

`get()` Method

Fetch with a fallback

`setdefault()` Method

Set and fetch in one step

Initializing `defaultdict` Dictionary

The `defaultdict` is a dictionary subclass.

```
from collections import defaultdict


dd = defaultdict(default_factory)
```

```
>
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(default_factory)
```

```
>
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)
```

```
>
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {})
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': })
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': })
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': list()})
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': []})
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': [1]})
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': [1]})
```

Initializing `defaultdict` Dictionary

The `default_factory` is a callable that returns the default value for any missing key.

```
from collections import defaultdict


dd = defaultdict(list)

dd["key1"].append(1)

print(dd["key2"])
```

```
> defaultdict(<class 'list'>, {'key1': [1], 'key2': []})
```

Initializing `defaultdict` Dictionary

You can use a `lambda` function with `defaultdict` to return any constant or computed default value when a missing key is accessed.

```
from collections import defaultdict


dd = defaultdict(default_factory)

print(dd["key3"])
```

```
> defaultdict(<class 'list'>, {})
```

Initializing `defaultdict` Dictionary

You can use a `lambda` function with `defaultdict` to return any constant or computed default value when a missing key is accessed..

```
from collections import defaultdict


dd = defaultdict(lambda: "Default value")

print(dd["key3"])
```

```
> defaultdict(<class 'list'>, {})
```

Initializing `defaultdict` Dictionary

You can use a `lambda` function with `defaultdict` to return any constant or computed default value when a missing key is accessed..

```
from collections import defaultdict


dd = defaultdict(lambda: "Default value")

print(dd["key3"])
```

```
> defaultdict(<class 'list'>, {})
```

Initializing `defaultdict` Dictionary

You can use a `lambda` function with `defaultdict` to return any constant or computed default value when a missing key is accessed..

```
from collections import defaultdict


dd = defaultdict(lambda: "Default value")

print(dd["key3"])
```

```
> defaultdict(<class 'list'>, {'key3': })
```

Initializing `defaultdict` Dictionary

You can use a `lambda` function with `defaultdict` to return any constant or computed default value when a missing key is accessed..

```
from collections import defaultdict


dd = defaultdict(lambda: "Default value")

print(dd["key3"])
```

```
> defaultdict(<class 'list'>, {'key3': })
```

Initializing `defaultdict` Dictionary

You can use a `lambda` function with `defaultdict` to return any constant or computed default value when a missing key is accessed..

```
from collections import defaultdict


dd = defaultdict(lambda: "Default value")

print(dd["key3"])
```

```
> defaultdict(<class 'list'>, {'key3': 'Default value'})
```

`Counter` Dictionaries

A `Counter` is a specialized dictionary that maps elements to their counts.

```
from collections import Counter


letters = ["a", "c", "c", "a", "a", "b"]
c = Counter(letters)
```

```
> Counter({})
```

Counter Dictionaries

A `Counter` is a specialized dictionary that maps elements to their counts.

```
from collections import Counter


letters = []
c = Counter(letters)
```

```
a a a
b
c c
```

```
> Counter({})
```

Counter Dictionaries

A `Counter` is a specialized dictionary that maps elements to their counts.

```
from collections import Counter


letters = []
c = Counter(letters)
```

```
a a a   3
b       1
c c     2
```

```
> Counter({})
```

`Counter` Dictionaries

A `Counter` is a specialized dictionary that maps elements to their counts.

```
from collections import Counter


letters = []
c = Counter(letters)
```

```
> Counter({'a': 3, 'c': 2, 'b': 1})
```

Counter Dictionaries

You can initialize a counter with an existing group of objects.

```
from collections import Counter


c = Counter({"a": 10, "b": 12, "c": 11})

# You can also use keyword arguments
c = Counter(a=10, b=12, c=11)
```

```
> Counter({"a": 10, "b": 12, "c": 11})
```

Counter Dictionaries

You can initialize a counter with an existing group of objects.

```
from collections import Counter


c = Counter({"a": 10, "b": 12, "c": 11})

# You can also use keyword arguments
c = Counter(a=10, b=12, c=11)
```

```
> Counter({"a": 10, "b": 12, "c": 11})
```

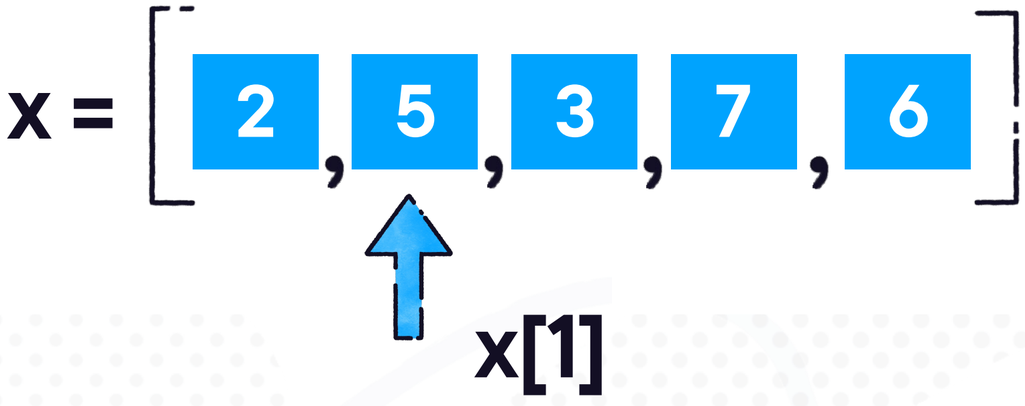
Big O Notation

Big O notation is a way to describe how fast or slow something (like a function or algorithm) gets as the amount of data it works on grows.

It doesn't give exact time, but it shows how the time increases as input size increases.

Big O Efficiency of Python List Operations

O(1)	vs.	
Accessing an element		



Big O Efficiency of Python List Operations

O(1)	vs.
Accessing an element	
Appending an element to the end	
Popping an element from the end	

$$x = [2, 5, 3, 7, 6]$$

Big O Efficiency of Python List Operations

O(1)	vs.
Accessing an element	
Appending an element to the end	
Popping an element from the end	

x = [2 , 5 , 3 , 7 , 6]

x.append(4)

Big O Efficiency of Python List Operations

O(1)	vs.
Accessing an element Appending an element to the end Popping an element from the end	

x = [2 , 5 , 3 , 7 , 6 , 4]

x.append(4)

Big O Efficiency of Python List Operations

O(1)	vs.
Accessing an element	
Appending an element to the end	
Popping an element from the end	

x = [2 , 5 , 3 , 7 , 6 , 4]

x.pop()

Big O Efficiency of Python List Operations

O(1)	vs.
Accessing an element	
Appending an element to the end	
Popping an element from the end	

x = [2 , 5 , 3 , 7 , 6]

x.pop()

Big O Efficiency of Python List Operations

O(1)	vs.	O(n)
Accessing an element Appending an element to the end Popping an element from the end		Removing an element from an arbitrary position Inserting an element to an arbitrary position

$$x = [\,2, 5, 3, 7, 6\,]$$

Big O Efficiency of Python List Operations

O(1)	vs.	O(n)
Accessing an element Appending an element to the end Popping an element from the end		Removing an element from an arbitrary position Inserting an element to an arbitrary position

x = [2 , 5 , 3 , 7 , 6]

x.insert(1, 9)

Big O Efficiency of Python List Operations

O(1)	vs.	O(n)
Accessing an element Appending an element to the end Popping an element from the end		Removing an element from an arbitrary position Inserting an element to an arbitrary position

x = [2 , 9 , 5 , 3 , 7 , 6]

x.insert(1, 9)

Big O Efficiency of Python List Operations

O(1)	vs.	O(n)
Accessing an element Appending an element to the end Popping an element from the end		Removing an element from an arbitrary position Inserting an element to an arbitrary position

x = [2 , 9 , 5 , 3 , 7 , 6]

x.pop(0)

Big O Efficiency of Python List Operations

O(1)	vs.	O(n)
Accessing an element Appending an element to the end Popping an element from the end		Removing an element from an arbitrary position Inserting an element to an arbitrary position

x = [9 , 5 , 3 , 7 , 6]

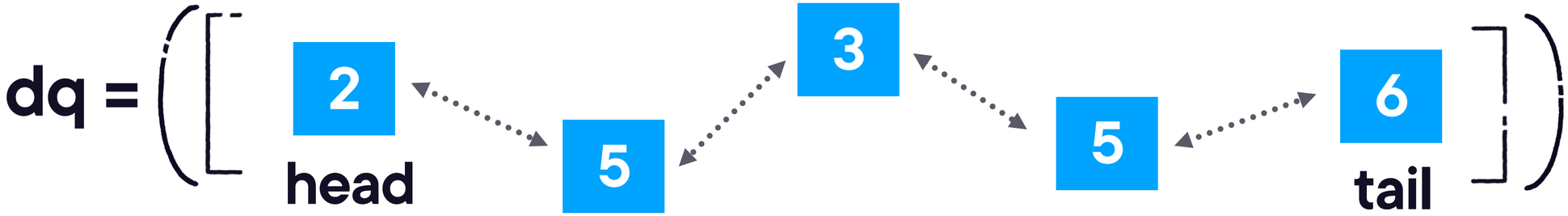
x.pop(0)

Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)

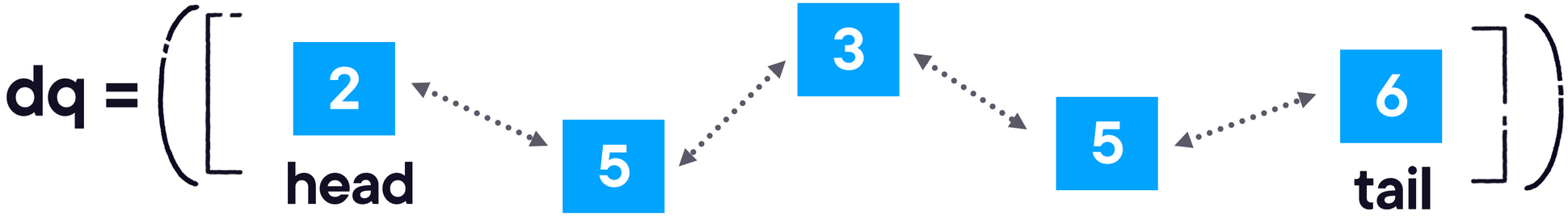
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)



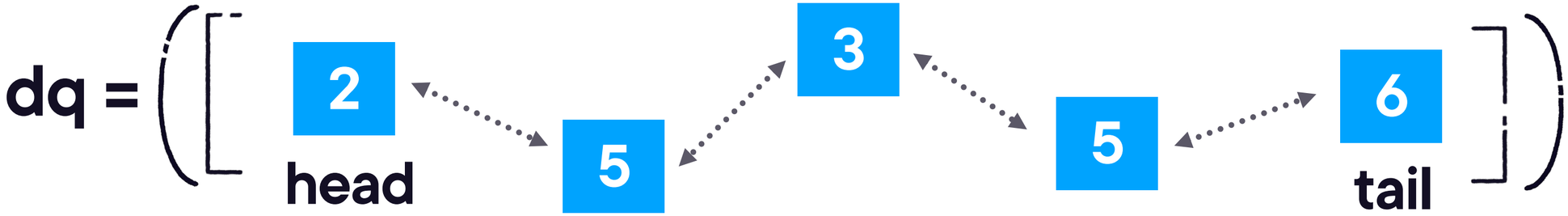
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Accessing the first or last element		



Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Accessing the first or last element		Accessing an element from an arbitrary position



Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Accessing the first or last element		Accessing an element from an arbitrary position

Big O Efficiency of Python deque Operations

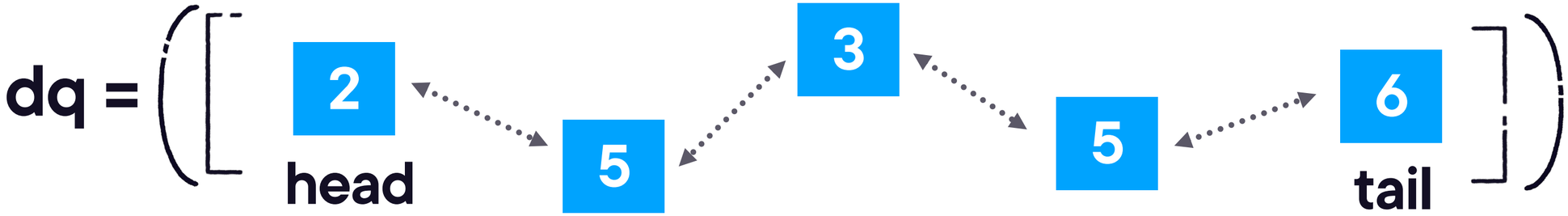
O(1)	vs.	O(n)
Accessing the first or last element		Accessing an element from an arbitrary position

Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Accessing the first or last element		Accessing an element from an arbitrary position

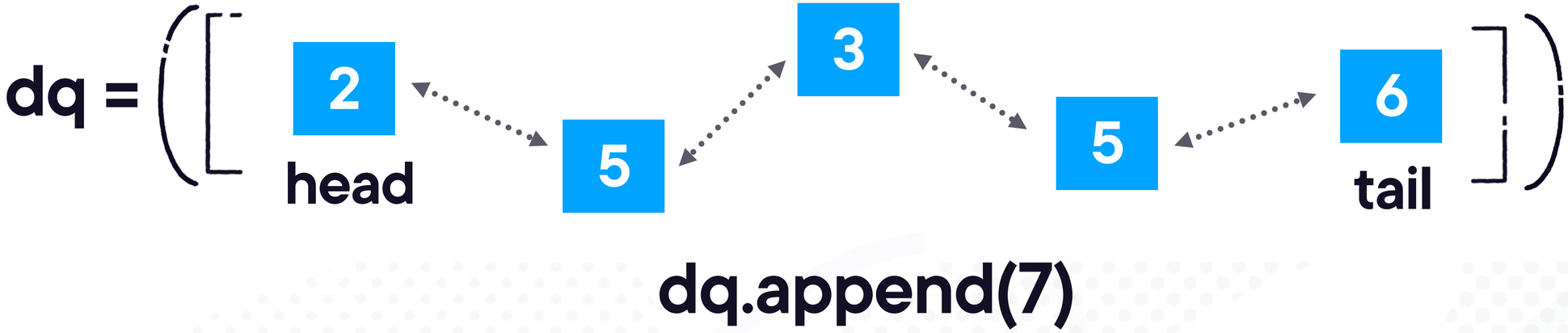
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)



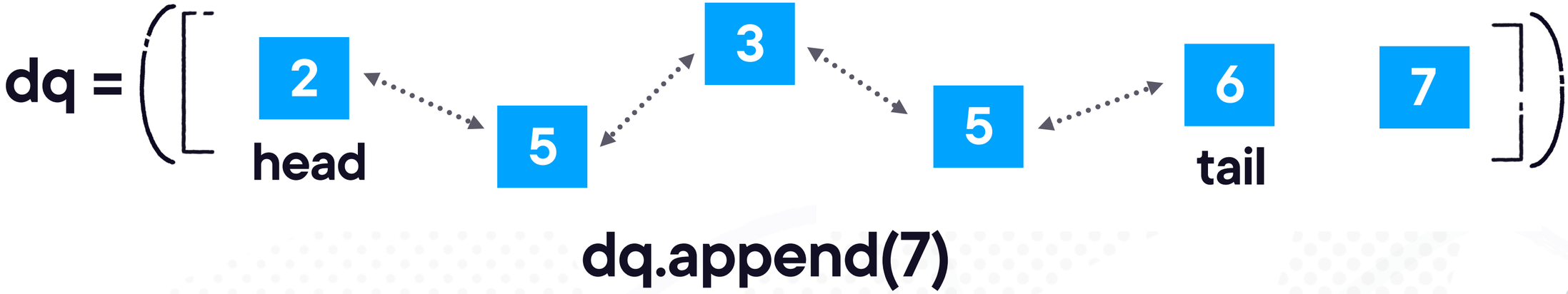
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



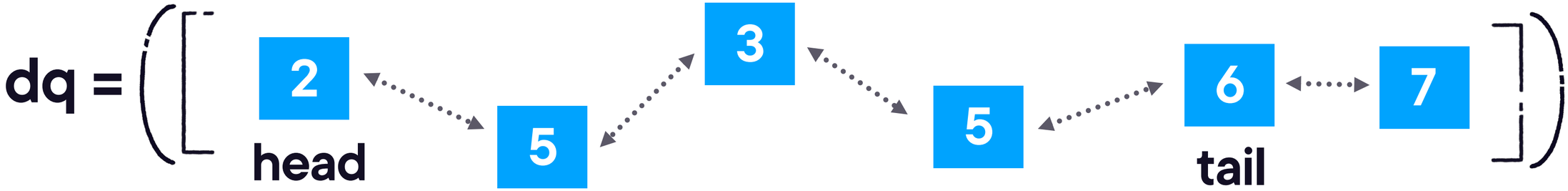
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



Big O Efficiency of Python deque Operations

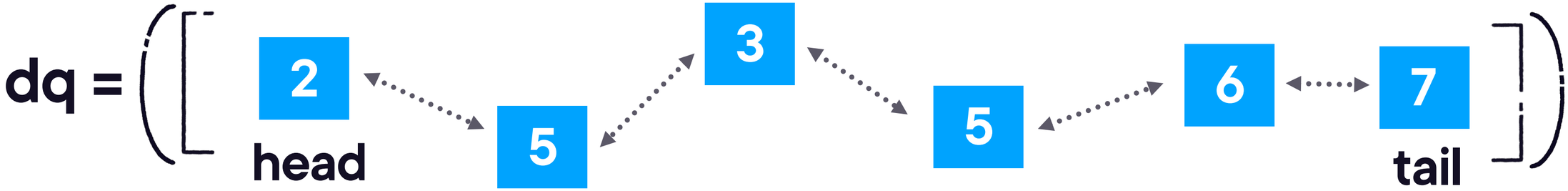
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



dq.append(7)

Big O Efficiency of Python deque Operations

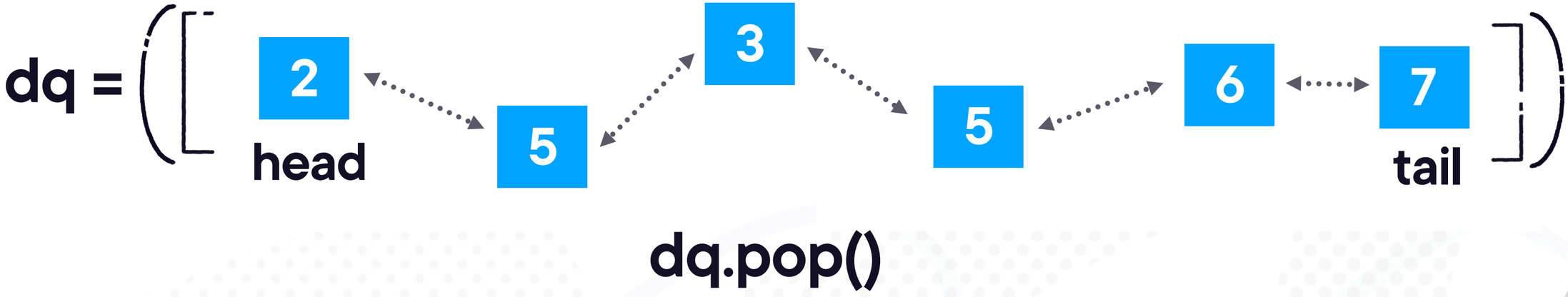
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



dq.append(7)

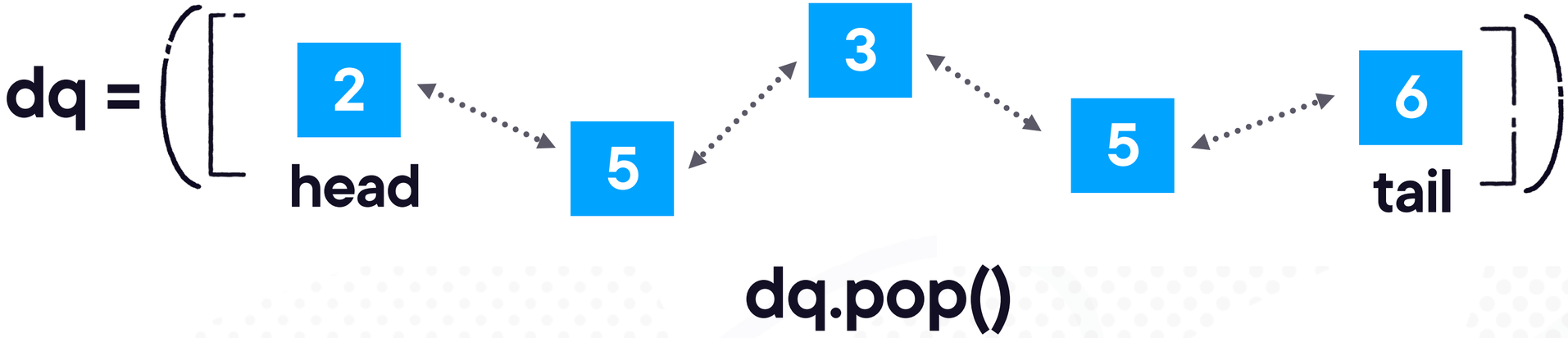
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



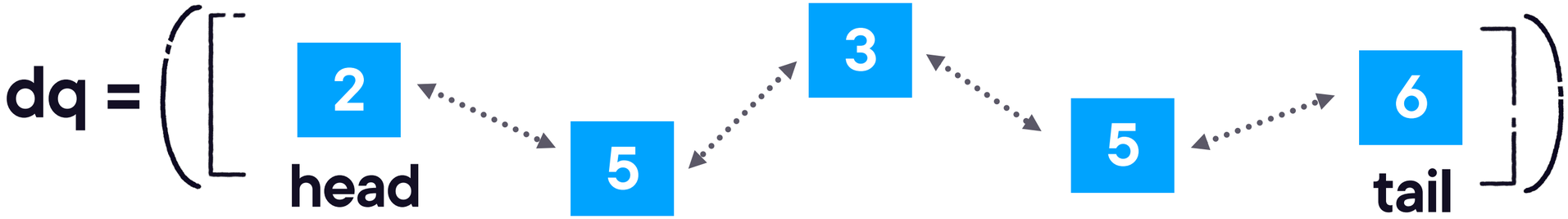
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



Big O Efficiency of Python deque Operations

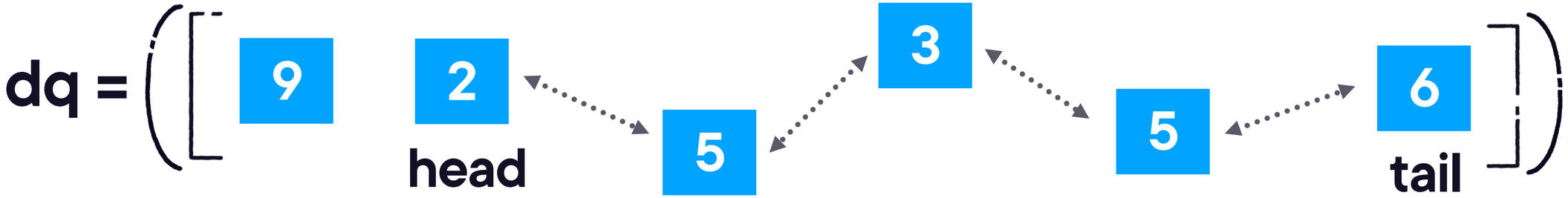
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



dq.appendleft(9)

Big O Efficiency of Python deque Operations

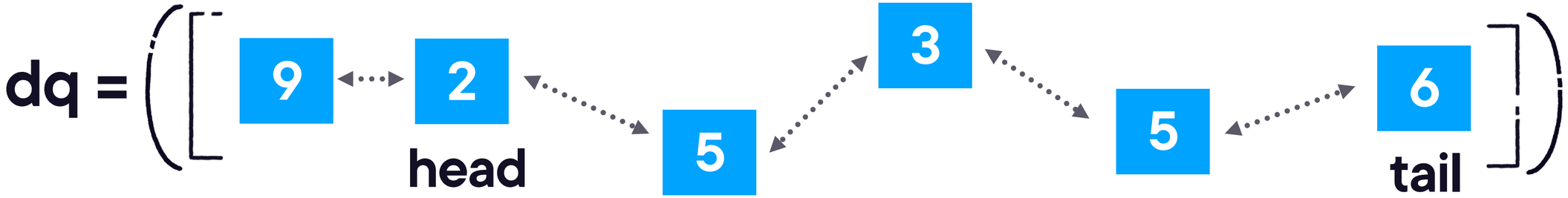
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



dq.appendleft(9)

Big O Efficiency of Python deque Operations

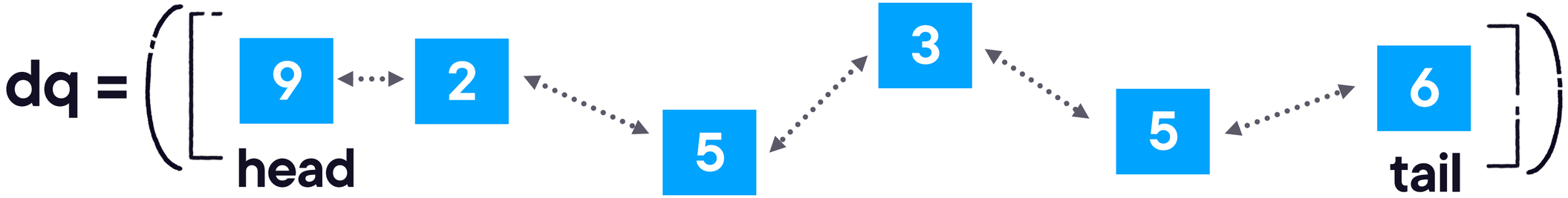
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



dq.appendleft(9)

Big O Efficiency of Python deque Operations

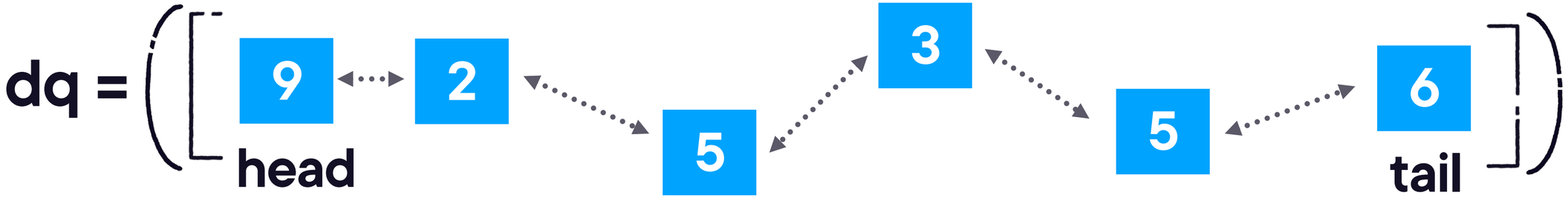
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		



dq.appendleft(9)

Big O Efficiency of Python deque Operations

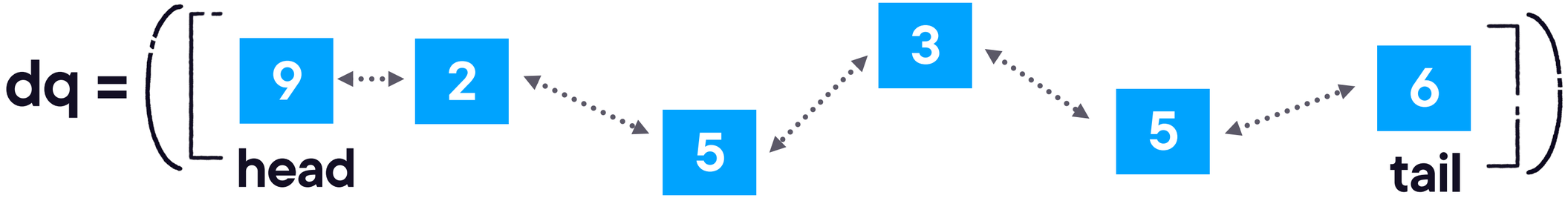
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



dq.appendleft(9)

Big O Efficiency of Python deque Operations

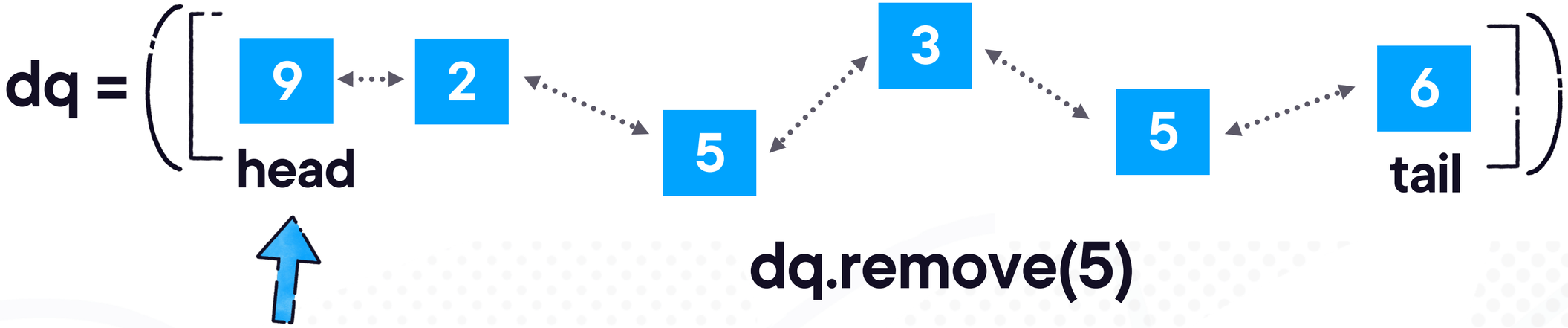
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



dq.remove(5)

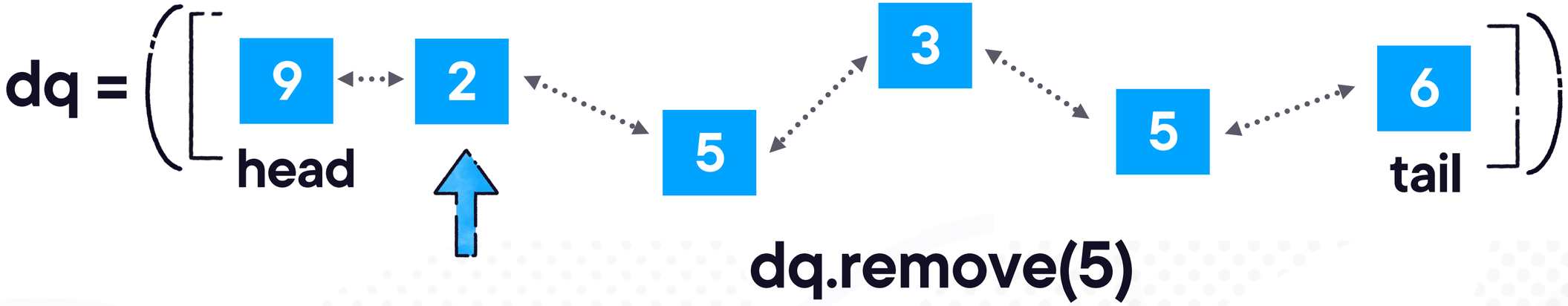
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



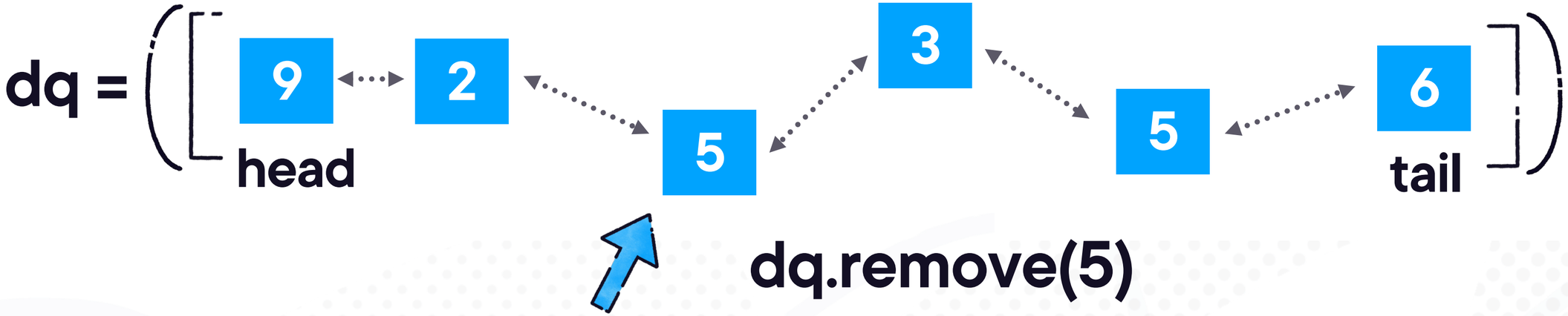
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



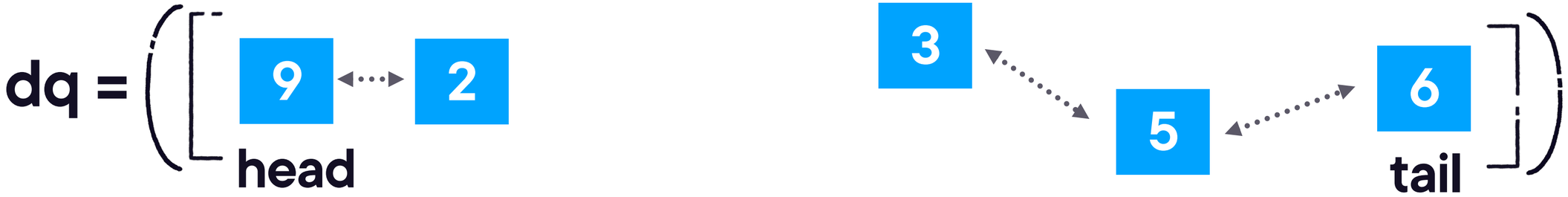
Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



Big O Efficiency of Python deque Operations

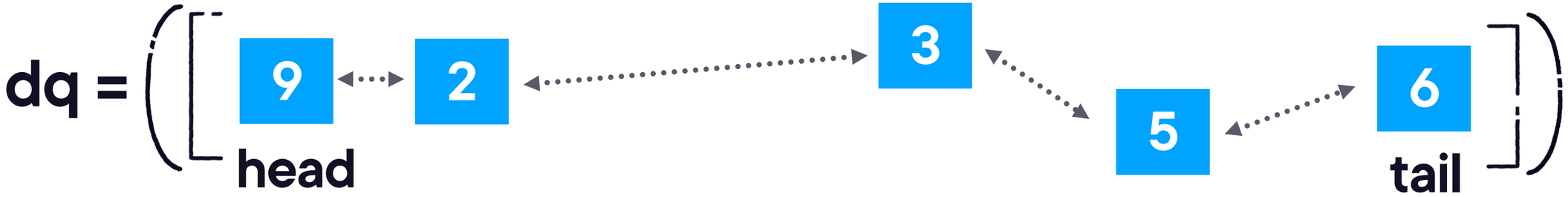
O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



dq.remove(5)

Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Appending an element to either side Popping an element from either side		Removing an element from an arbitrary position Inserting an element to an arbitrary position



dq.remove(5)

Big O Efficiency of Python deque Operations

O(1)	vs.	O(n)
Accessing the first or last element		Accessing an element from an arbitrary position
Appending an element to either side		Removing an element from an arbitrary position
Popping an element from either side		Inserting an element to an arbitrary position

Thank you for your time!