programmers

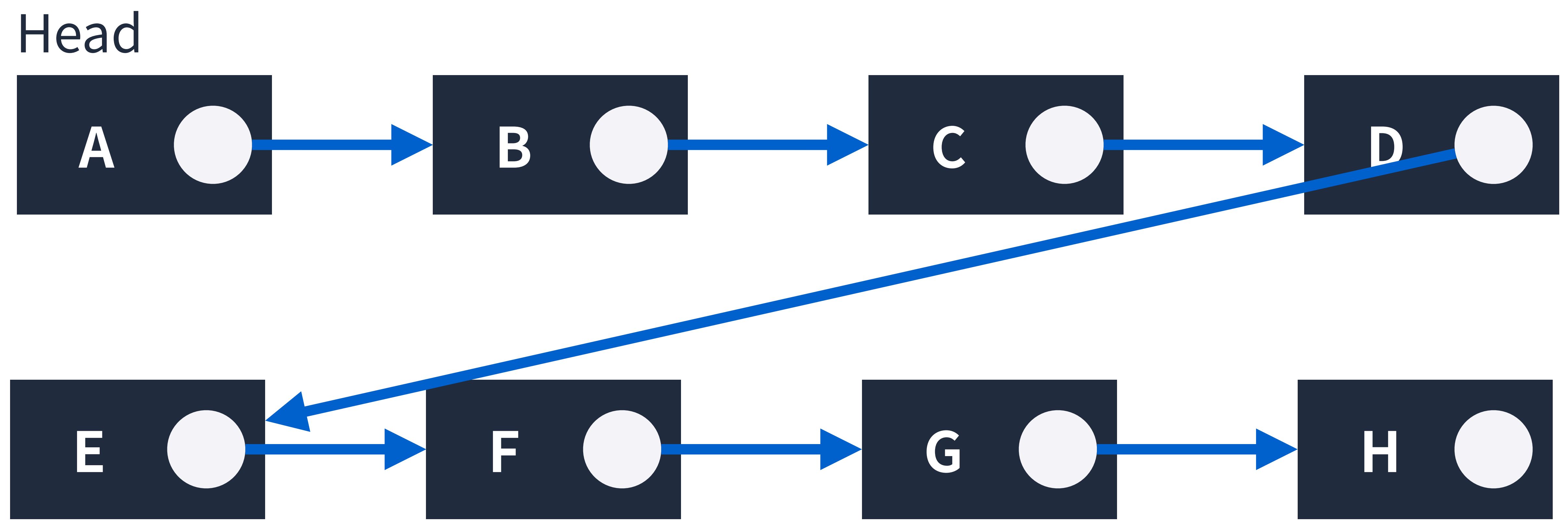
연결 리스트

코딩테스트 광탈방지 A to Z : JavaScript - 이선협 @kciter

추가와 **삭제**가 반복되는 로직이라면
어떻게 해야할까?

연결 리스트

연결 리스트는 각 요소를 포인터로 연결하여 관리하는 선형 자료구조다.
각 요소는 노드라고 부르며 데이터 영역과 포인터 영역으로 구성된다.



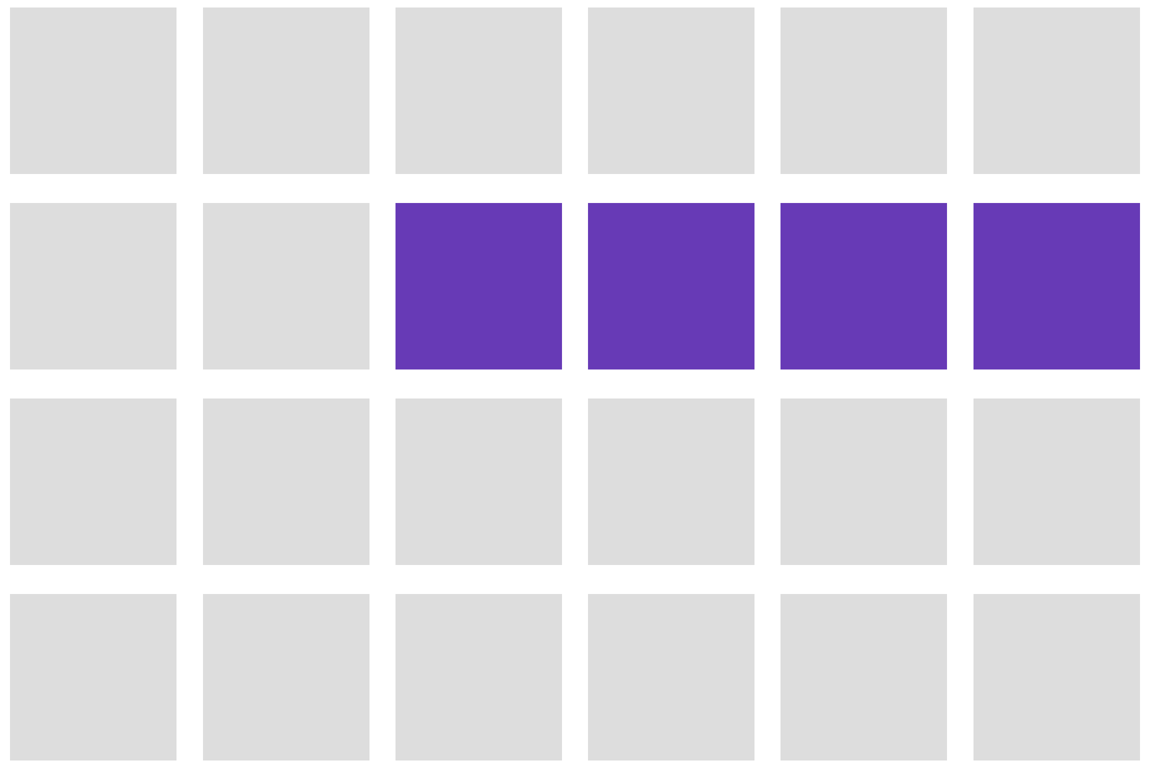
연결 리스트의 특징

- 메모리가 허용하는한 요소를 제한없이 추가할 수 있다.
- 탐색은 O(n)이 소요된다.
- 요소를 추가하거나 제거할 때는 O(1)이 소요된다.
- Singly Linked List, Doubly Linked List, Circular Linked List가 존재한다.

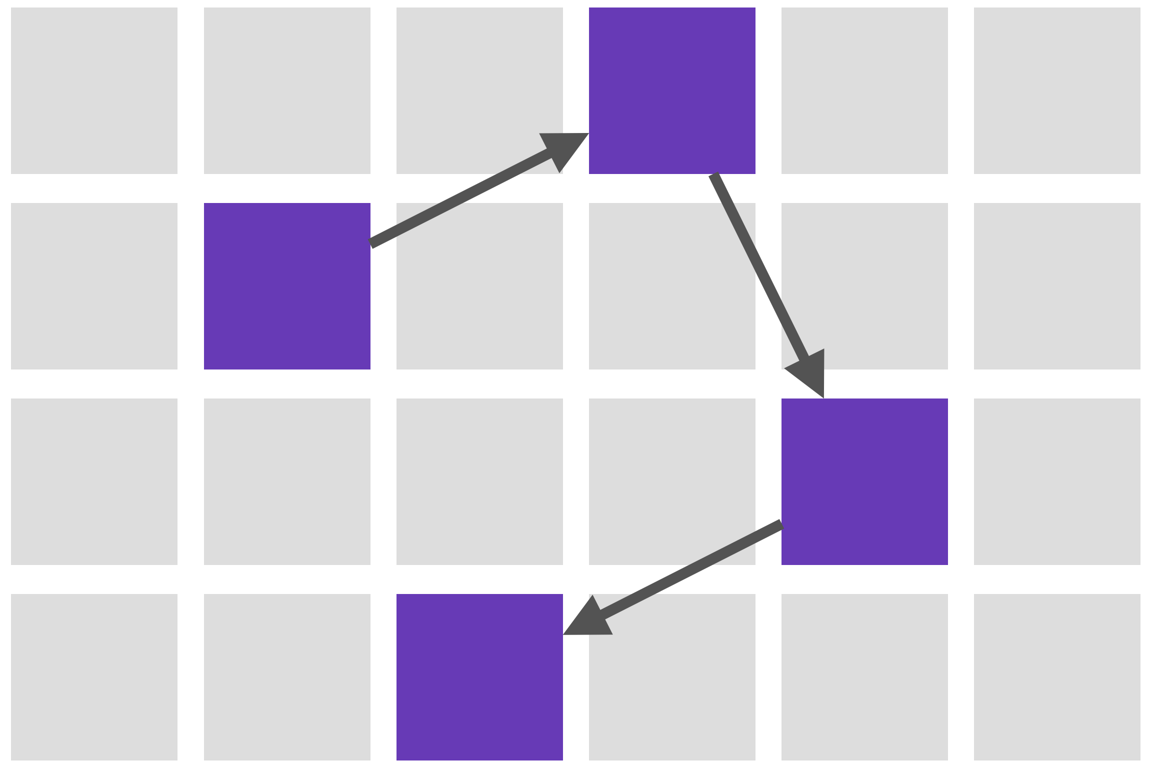
배열과 차이점

메모리 차이

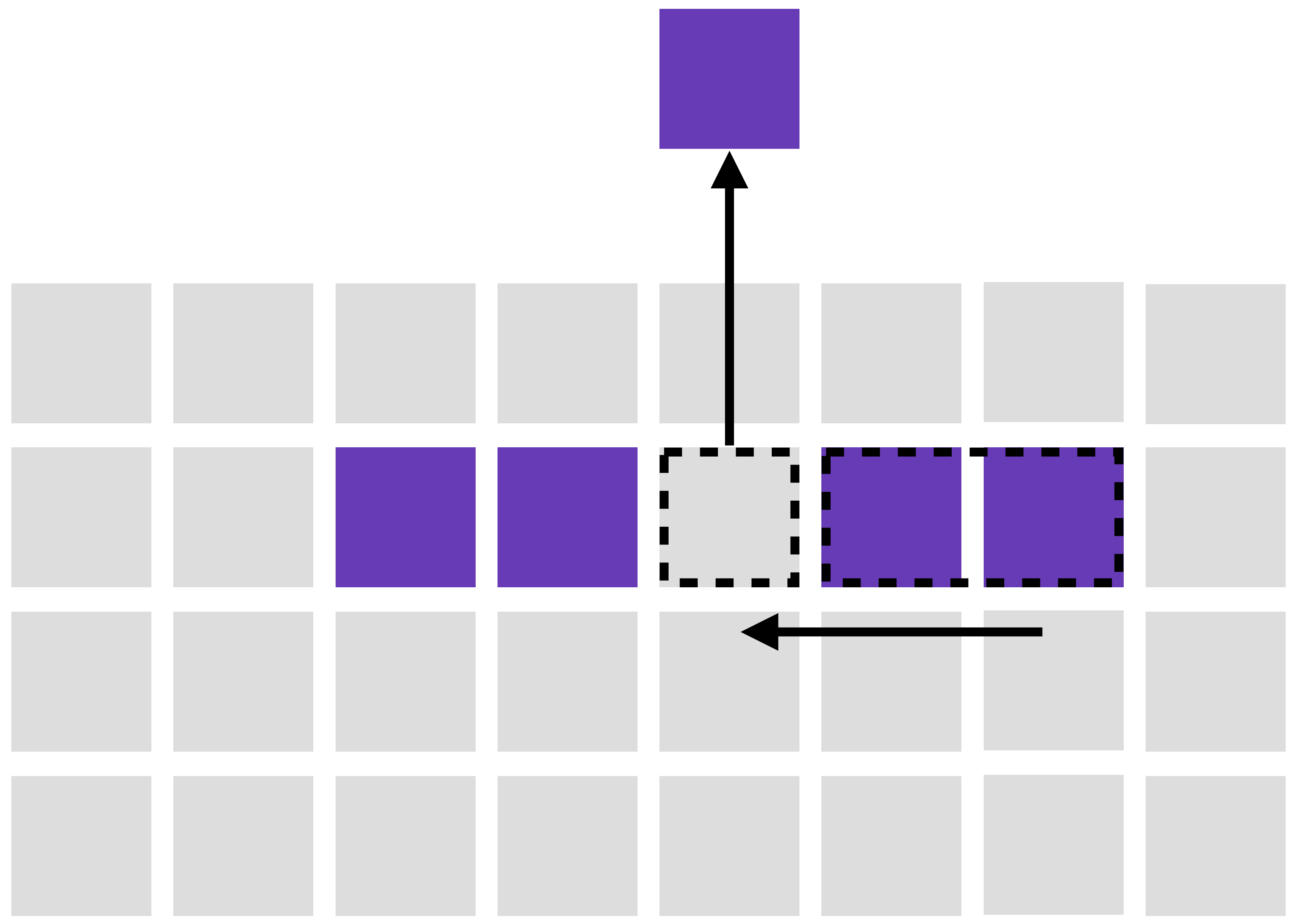
Array



Linked List

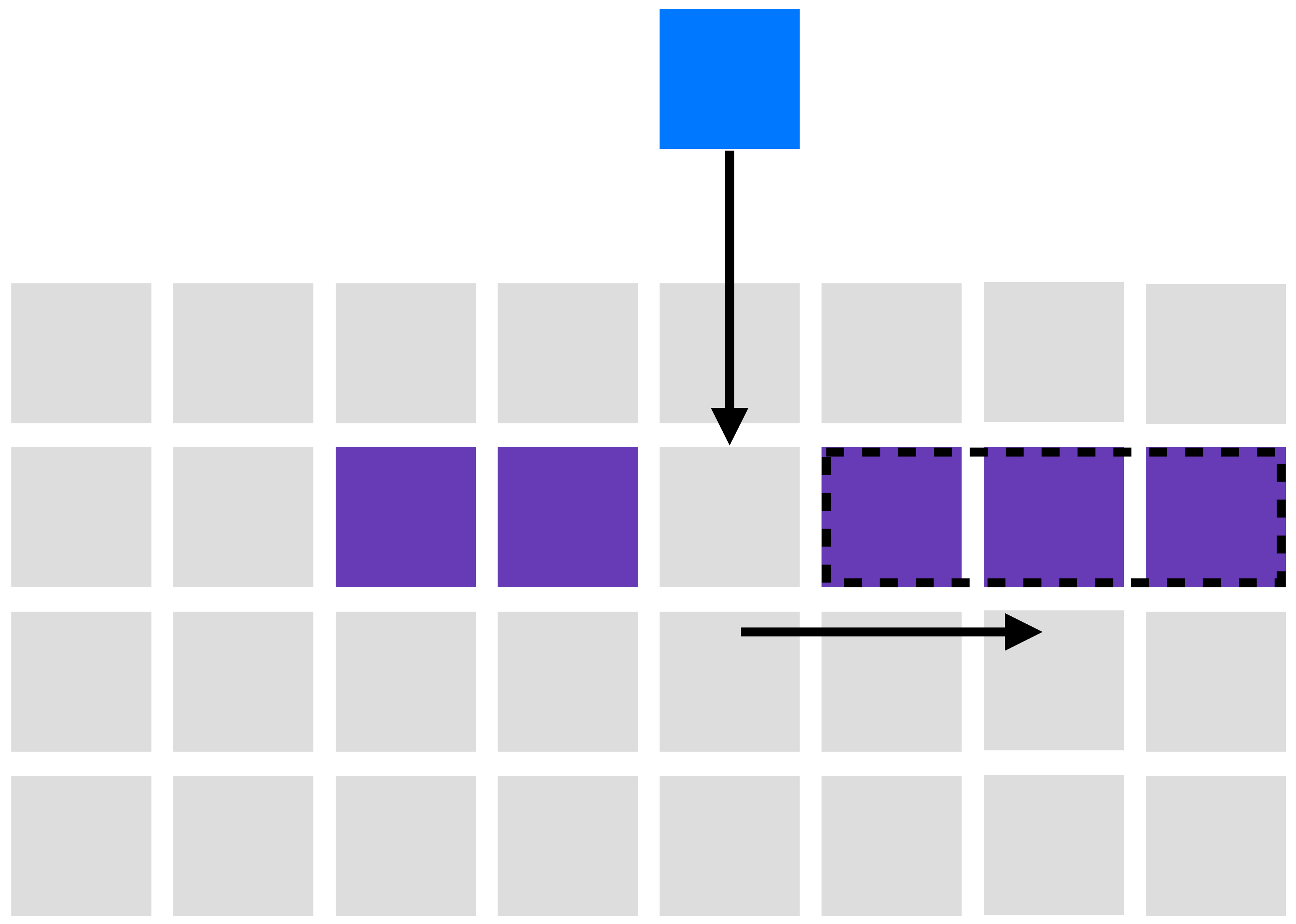


배열 요소 삭제



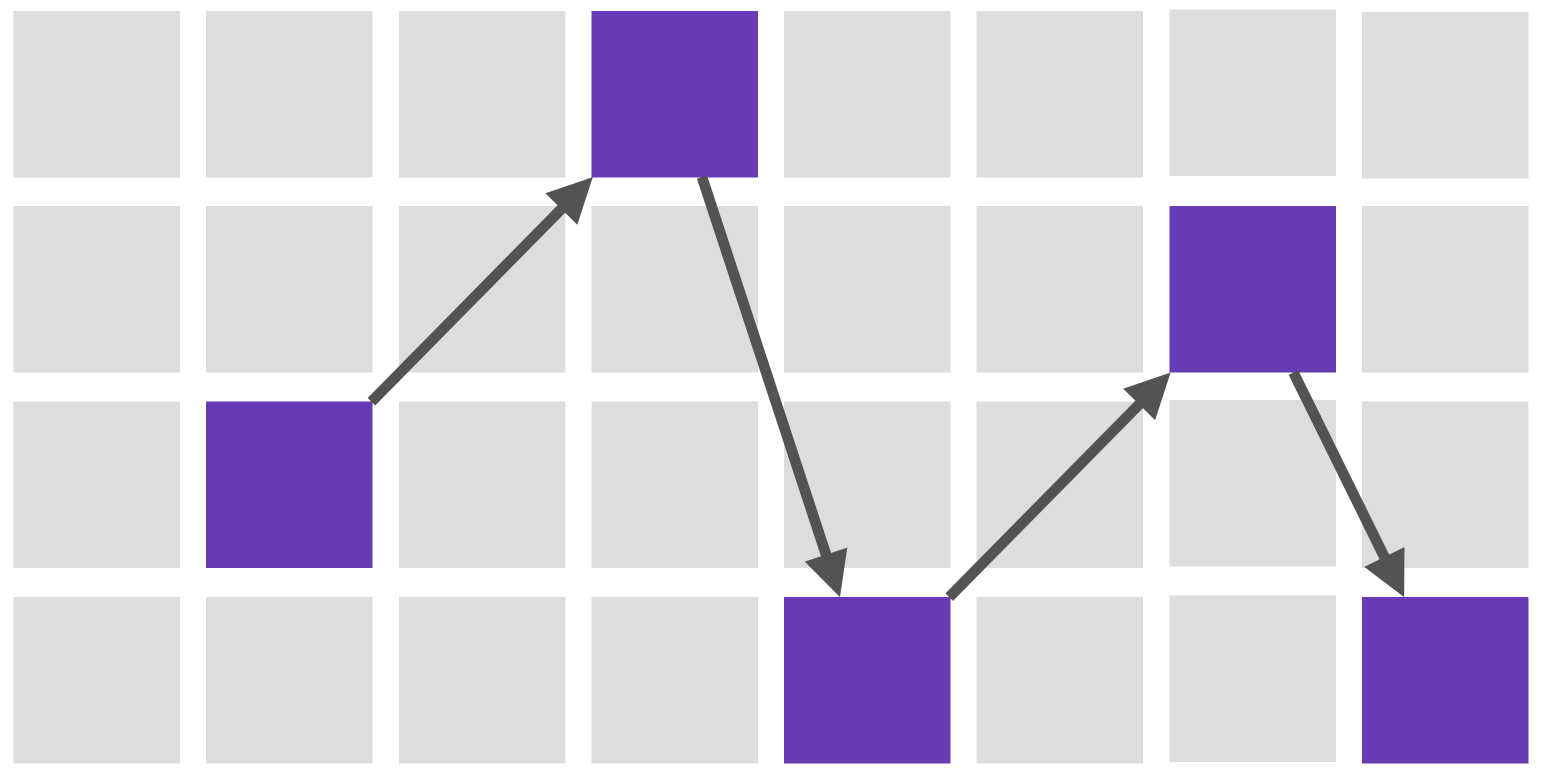
$O(n)$

배열 요소 추가

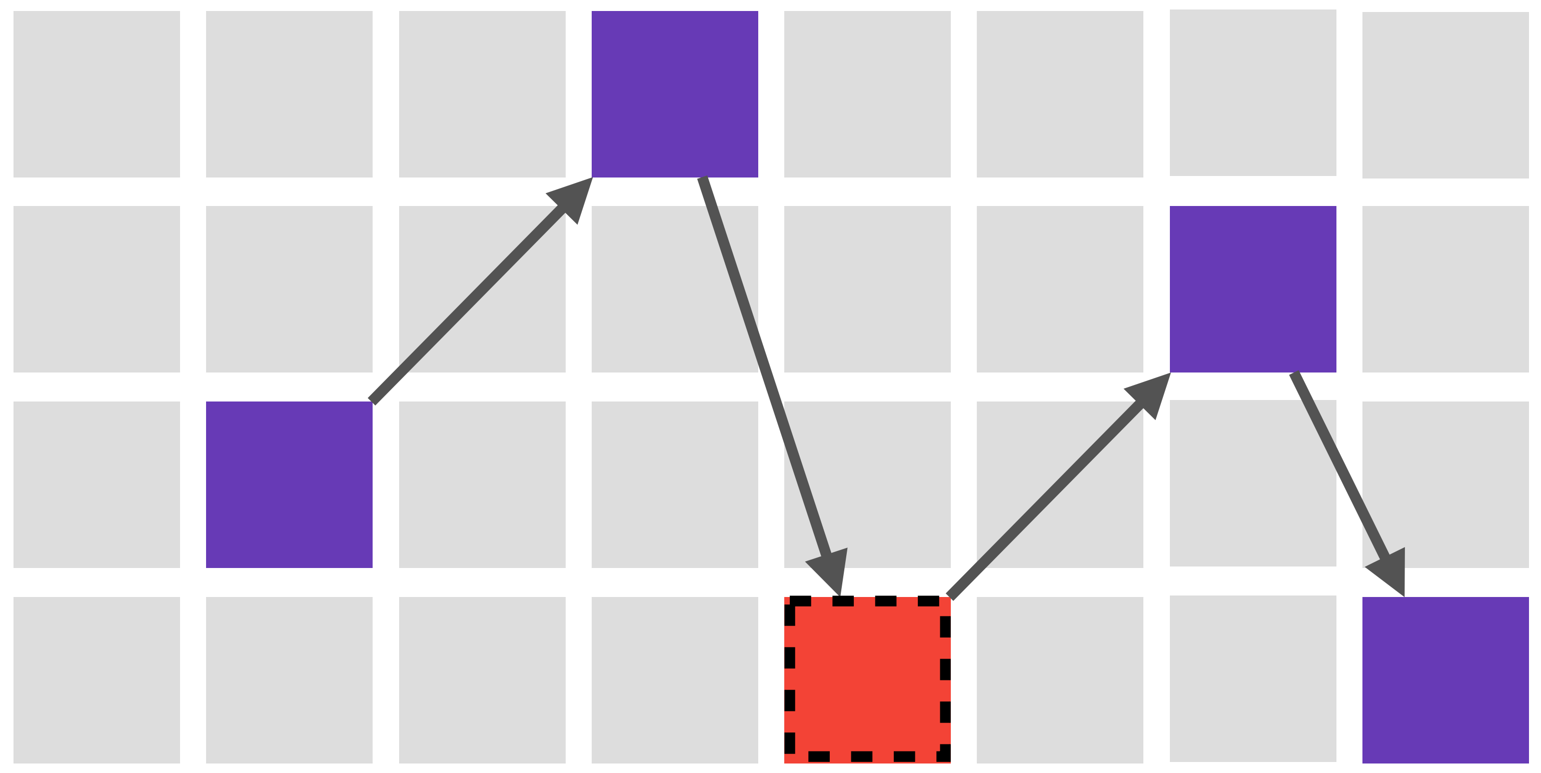


$O(n)$

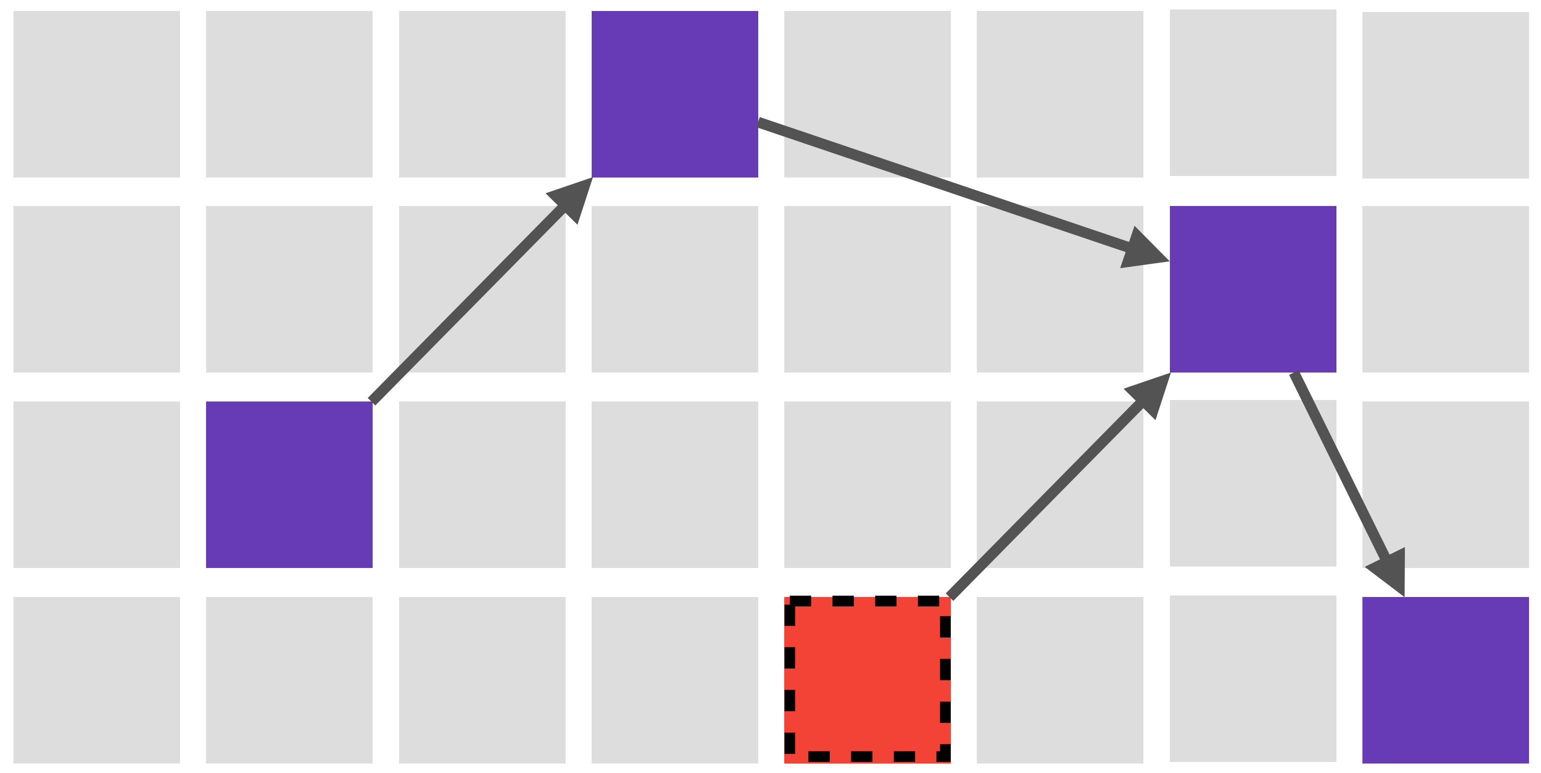
연결 리스트 요소 삭제



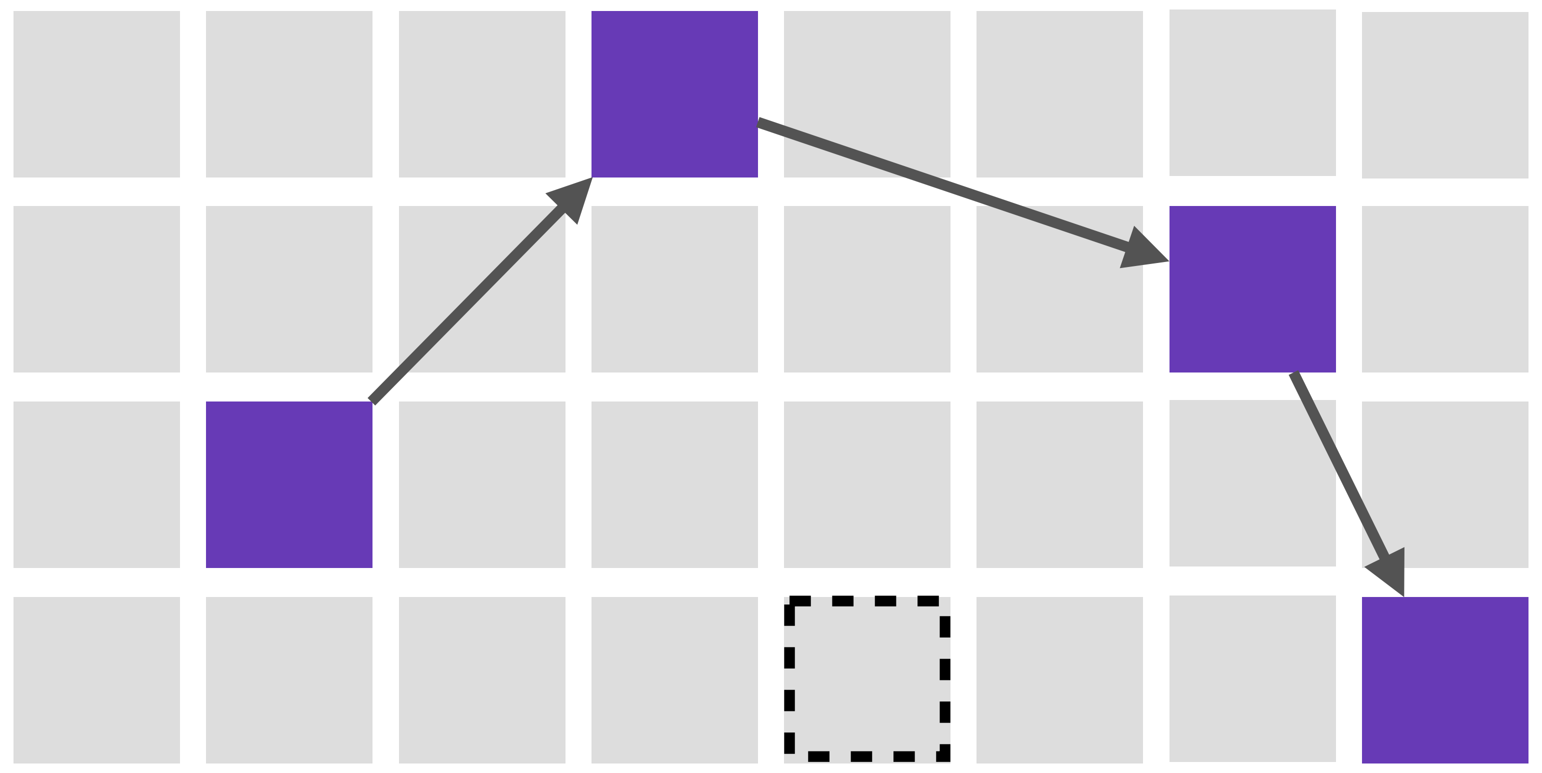
연결 리스트 요소 삭제



연결 리스트 요소 삭제

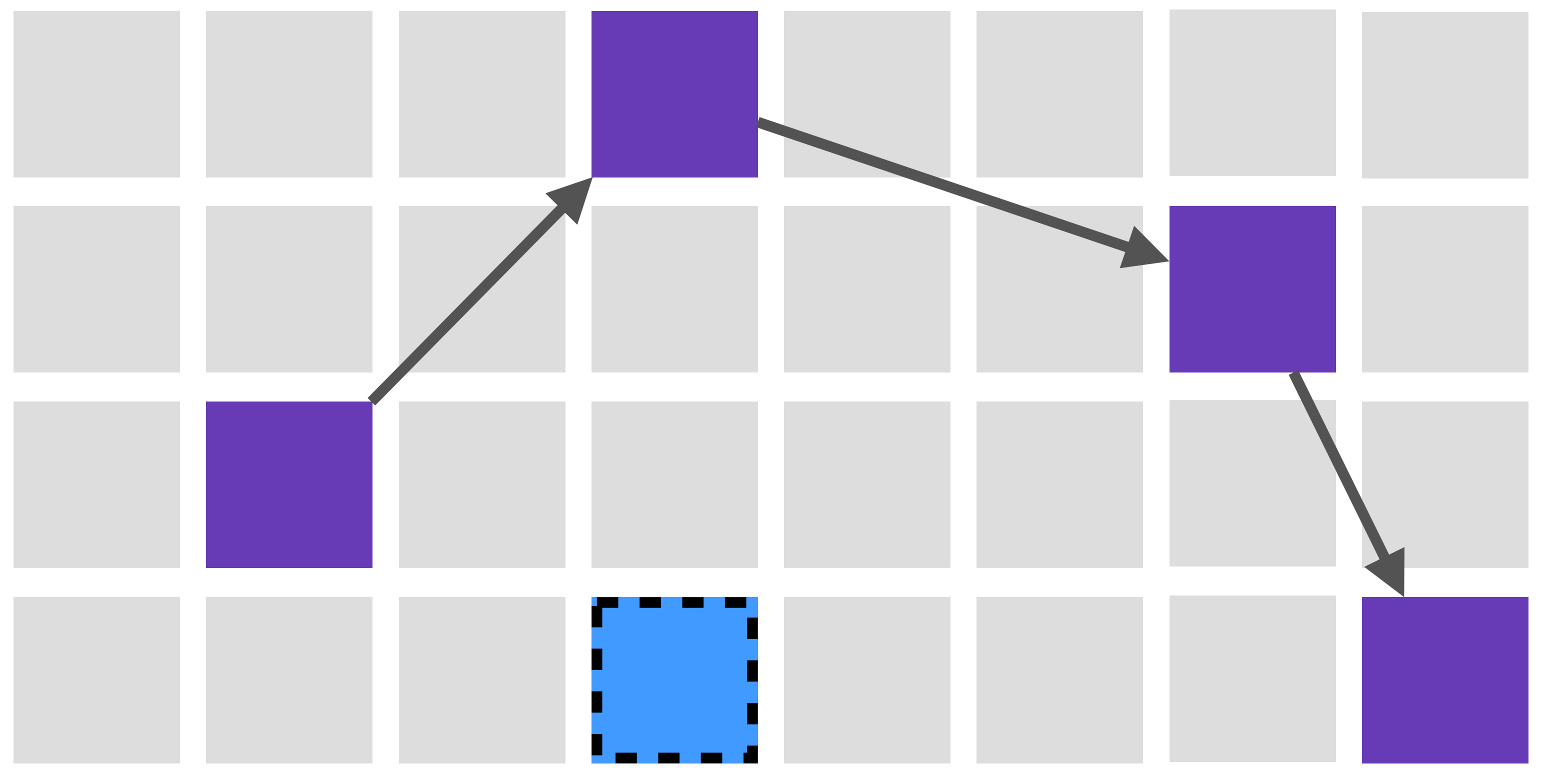


연결 리스트 요소 삭제

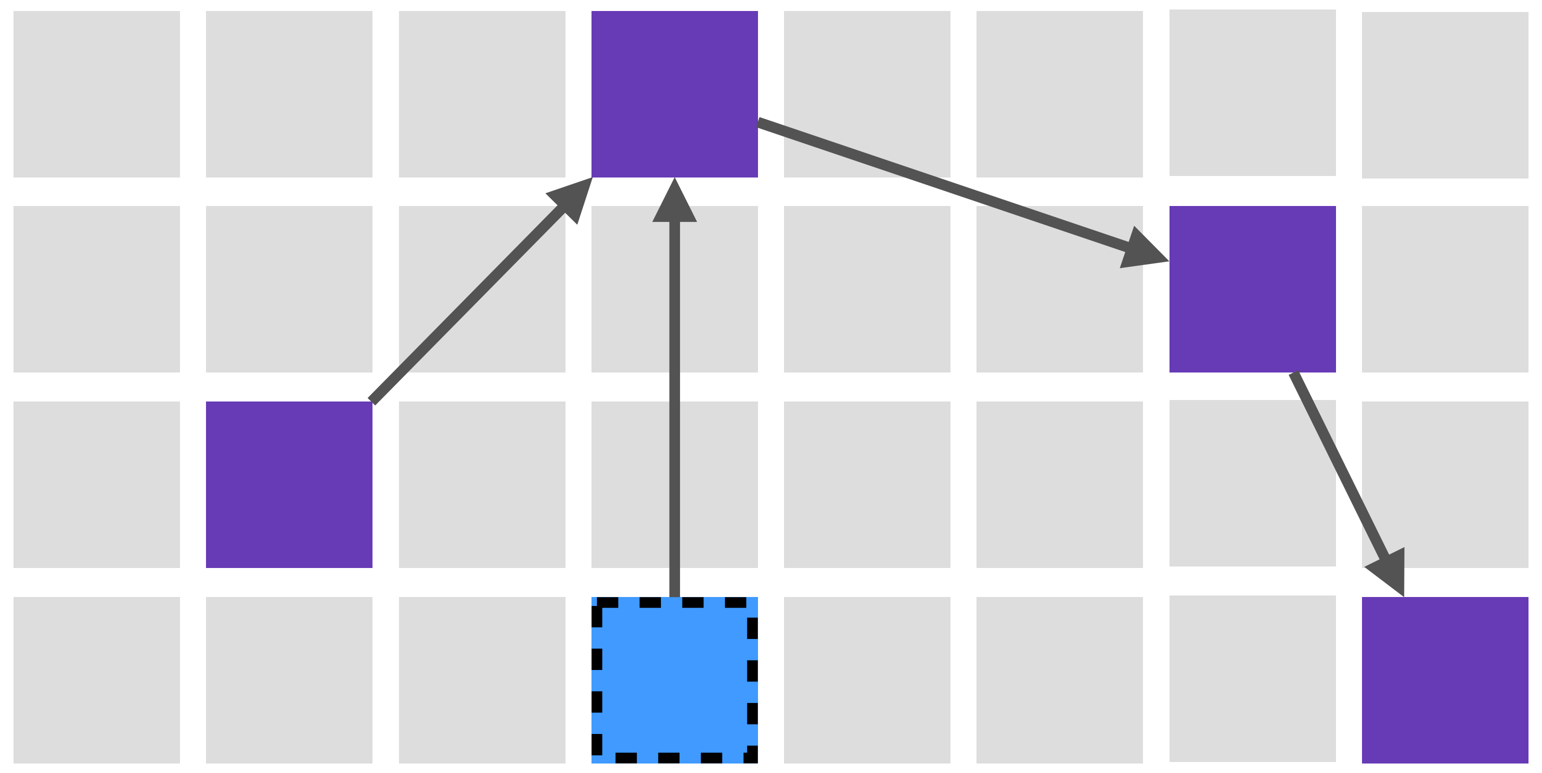


$O(1)$

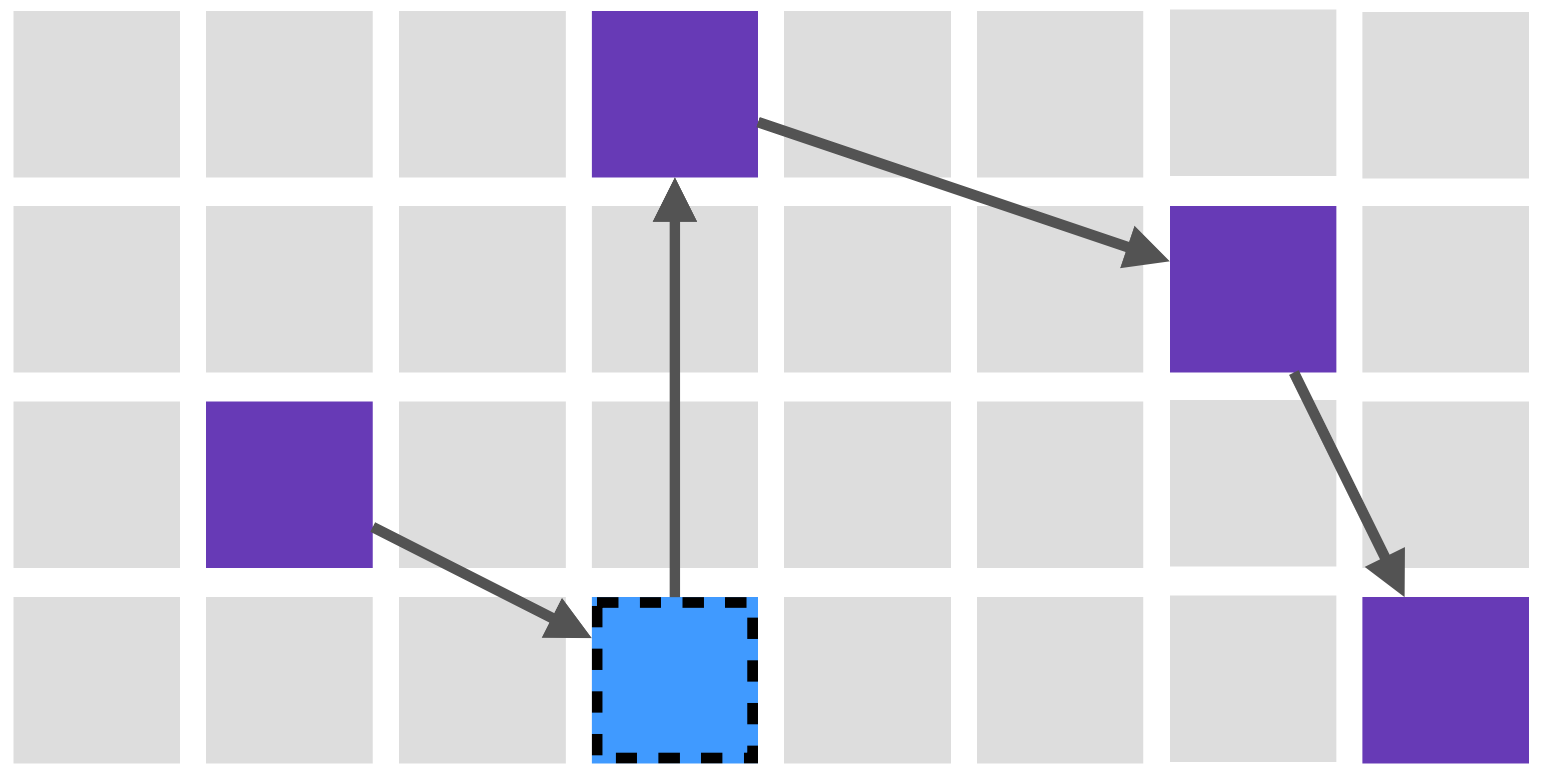
연결 리스트 요소 추가



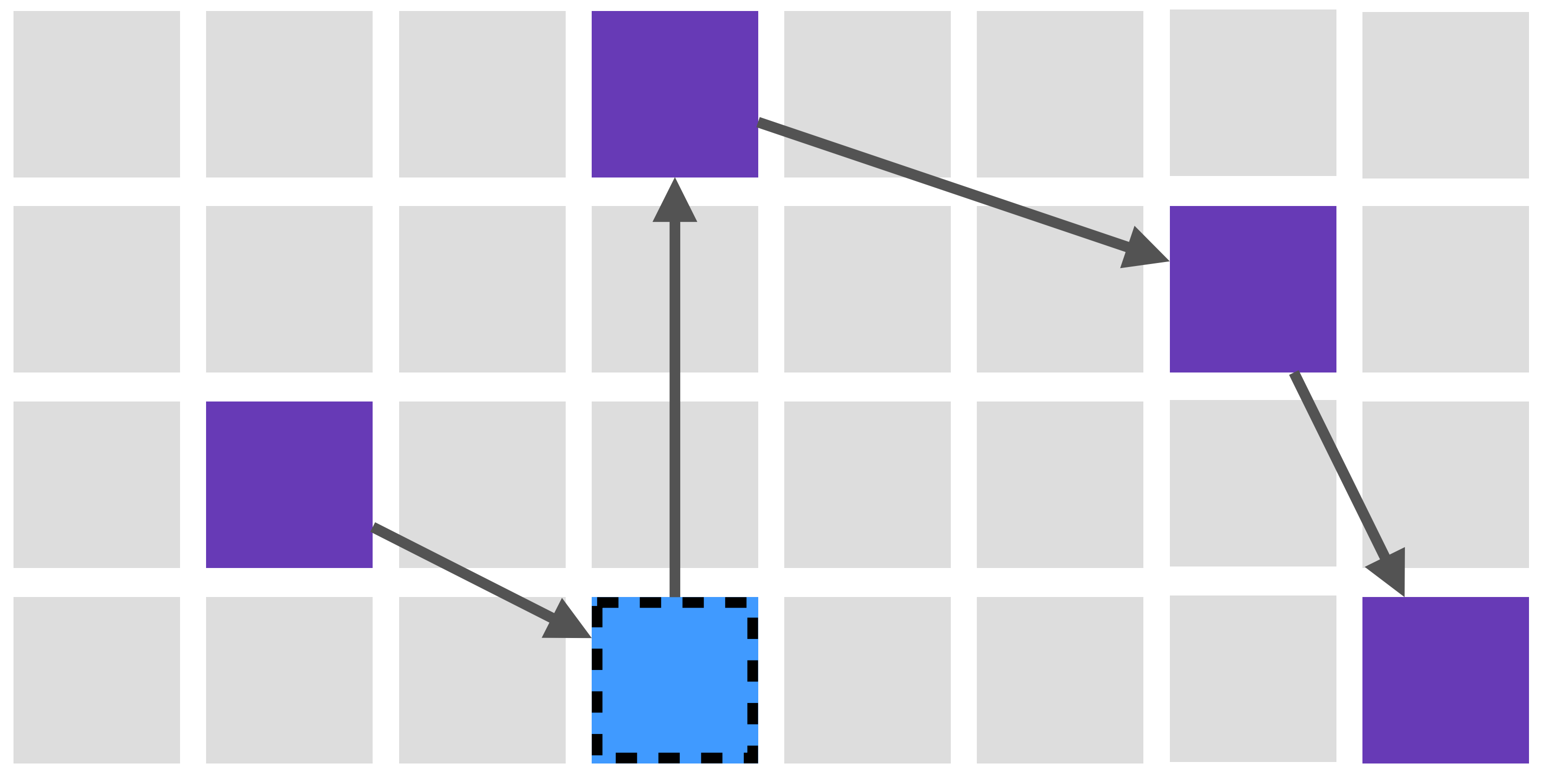
연결 리스트 요소 추가



연결 리스트 요소 추가



연결 리스트 요소 추가

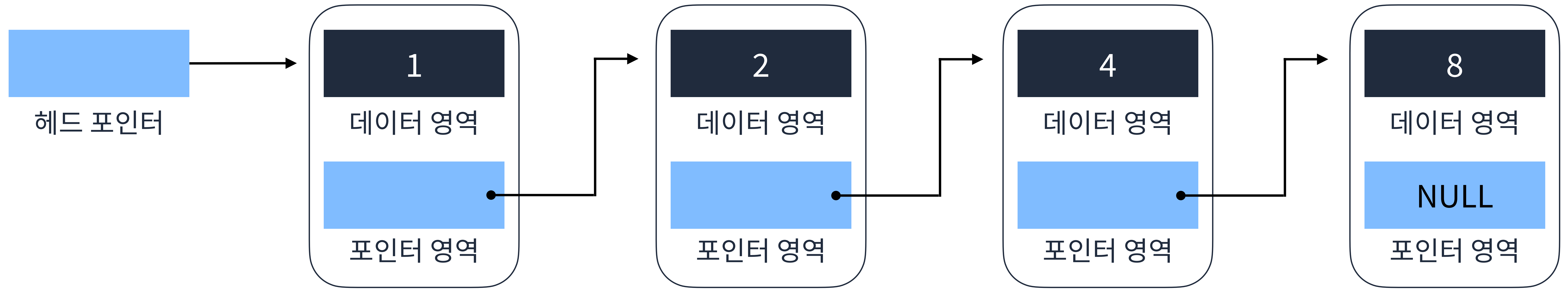


$O(1)$

Singly Linked List

Singly Linked List

Head에서 Tail까지 단방향으로 이어지는 연결 리스트
가장 단순한 형태인 연결 리스트다.



핵심 로직

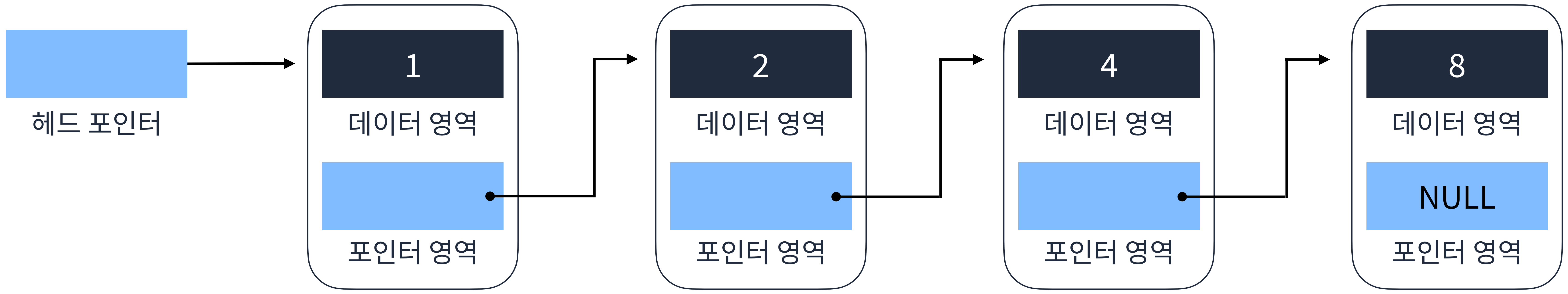
요소 찾기

요소 추가

요소 삭제

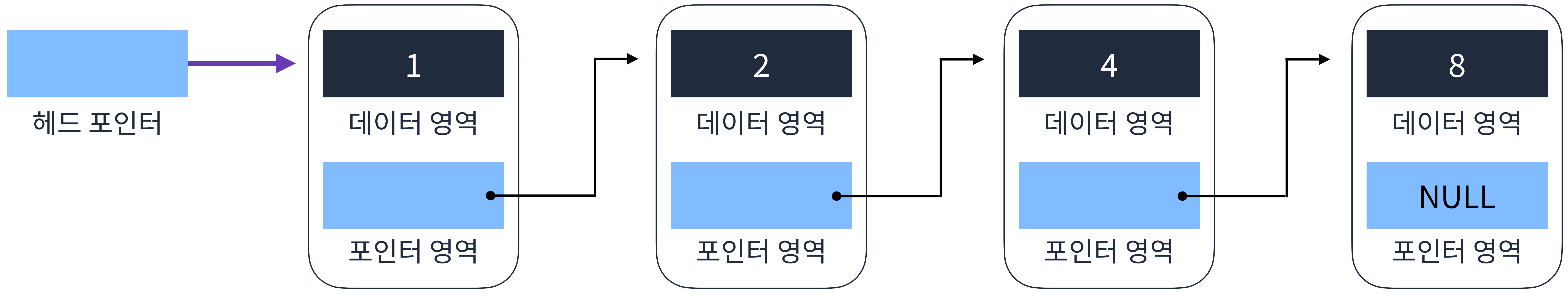
요소 찾기

'4'를 찾는다면?


헤드 포인터
1
데이터 영역
포인터 영역
2
데이터 영역
포인터 영역
4
데이터 영역
포인터 영역
8
데이터 영역
NULL
포인터 영역

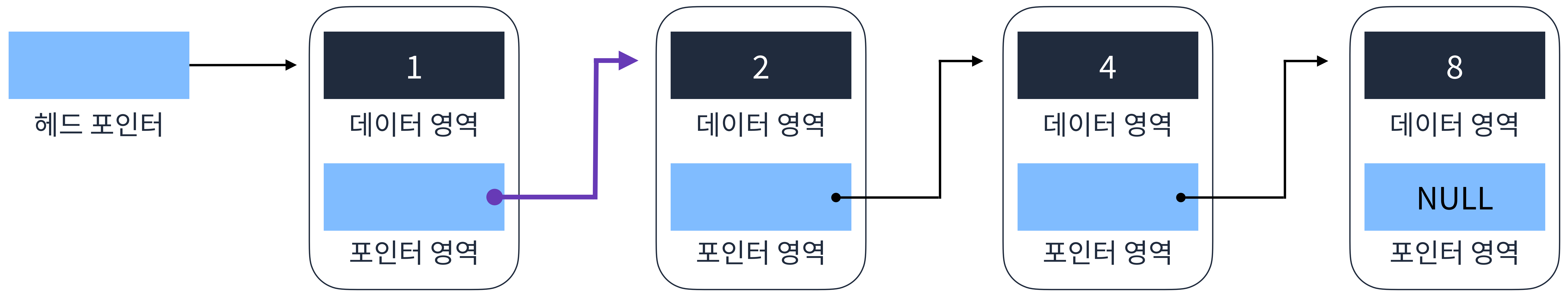
요소 찾기

'4'를 찾는다면?


헤드 포인터
1
데이터 영역
포인터 영역
2
데이터 영역
포인터 영역
4
데이터 영역
포인터 영역
8
데이터 영역
NULL
포인터 영역

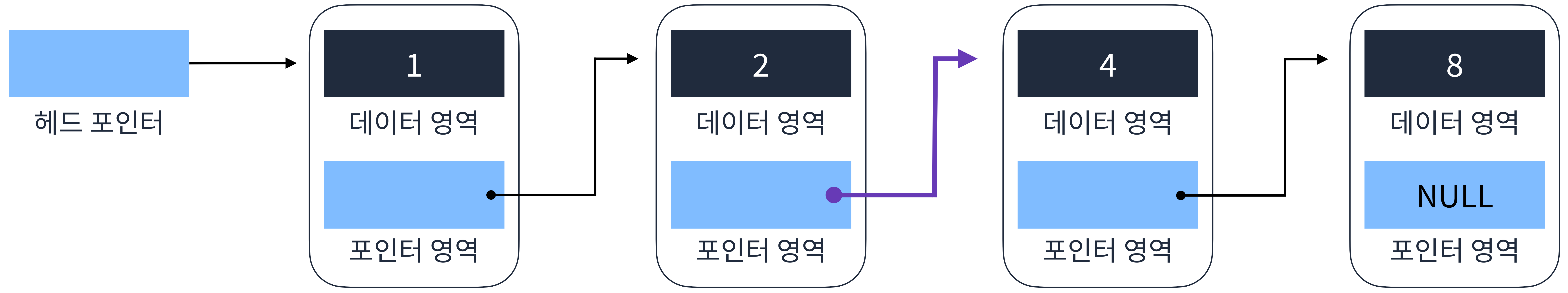
요소 찾기

'4'를 찾는다면?


헤드 포인터
1
데이터 영역
포인터 영역
2
데이터 영역
포인터 영역
4
데이터 영역
포인터 영역
8
데이터 영역
NULL
포인터 영역

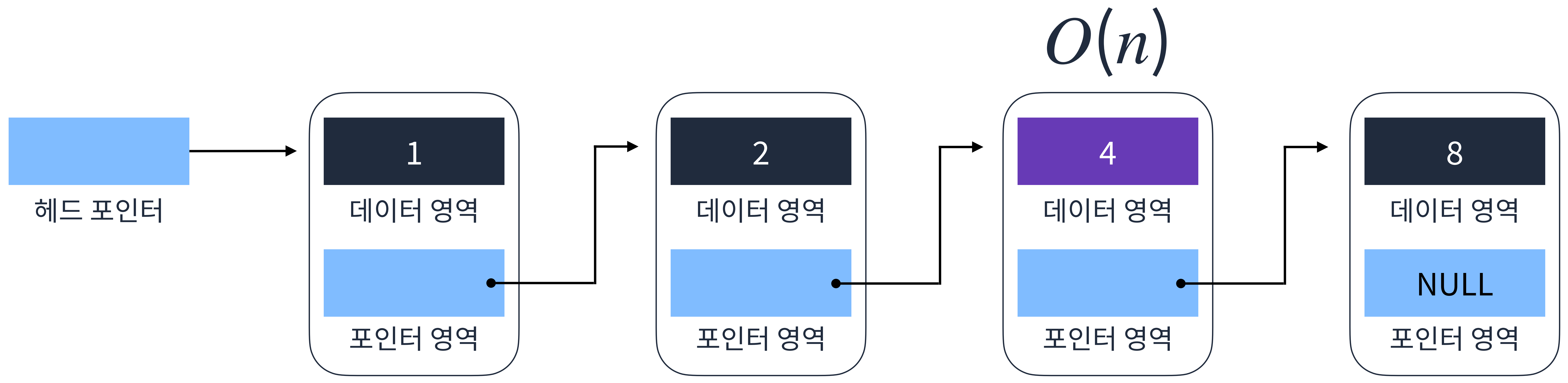
요소 찾기

'4'를 찾는다면?


헤드 포인터
1
데이터 영역
포인터 영역
2
데이터 영역
포인터 영역
4
데이터 영역
포인터 영역
8
데이터 영역
NULL
포인터 영역

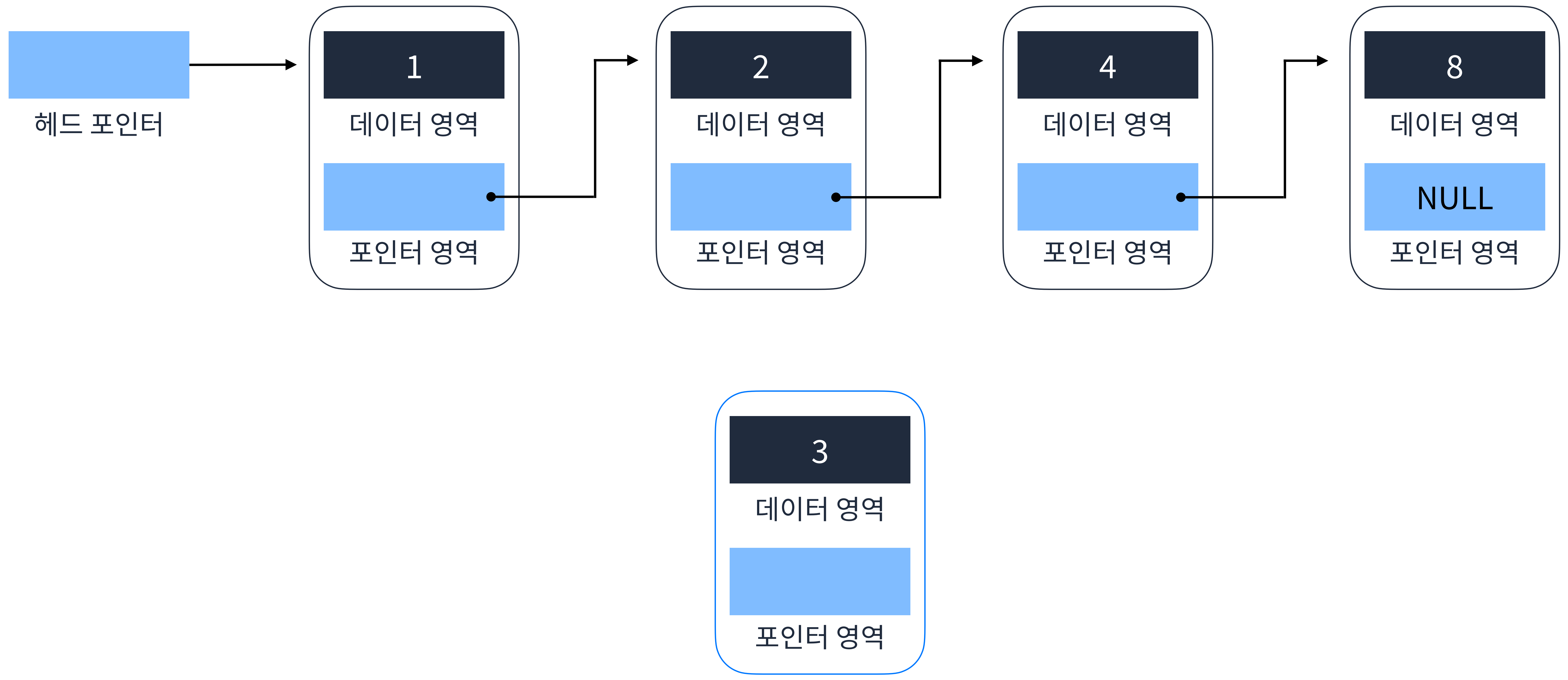
요소 찾기

'4'를 찾는다면?


$O(n)$
헤드 포인터
1
데이터 영역
포인터 영역
2
데이터 영역
포인터 영역
4
데이터 영역
포인터 영역
8
데이터 영역
NULL
포인터 영역

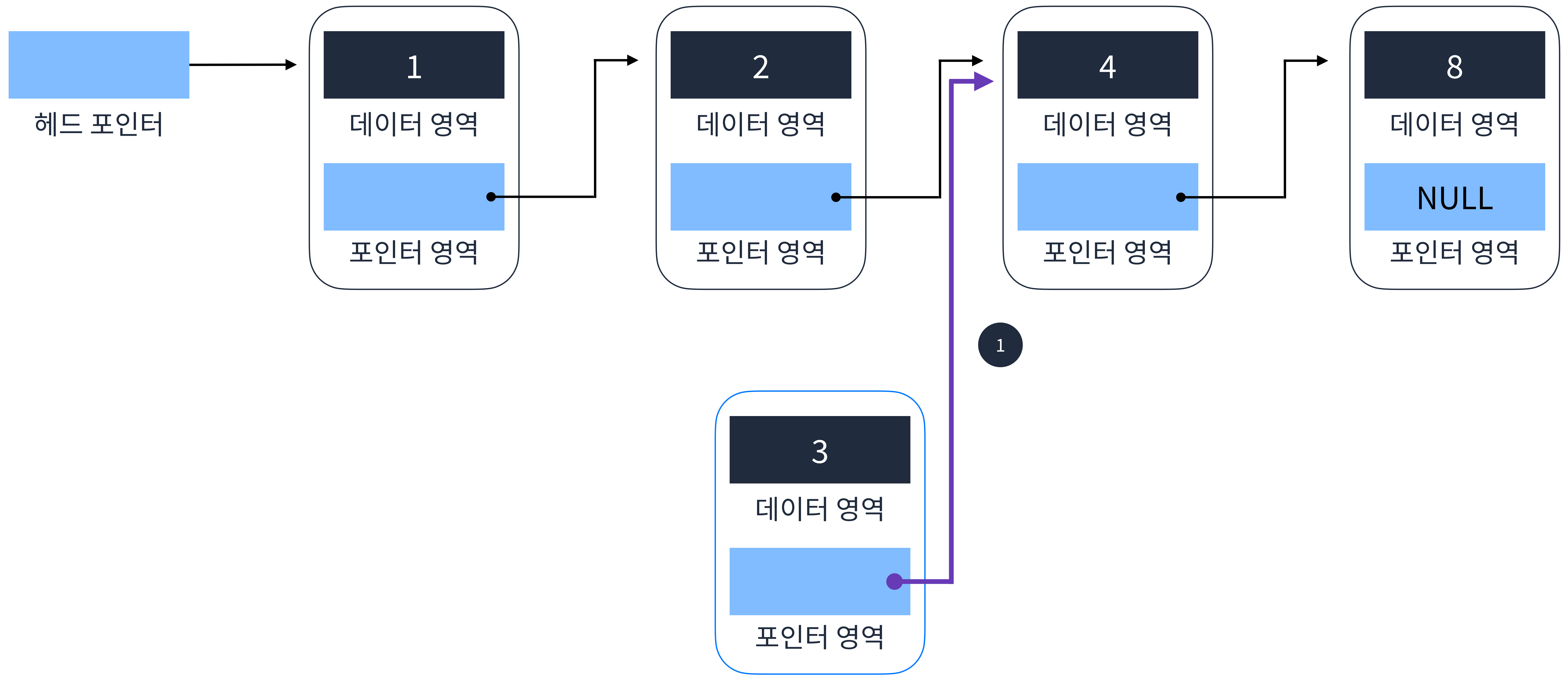
요소 추가

'3'을 중간에 추가한다면?



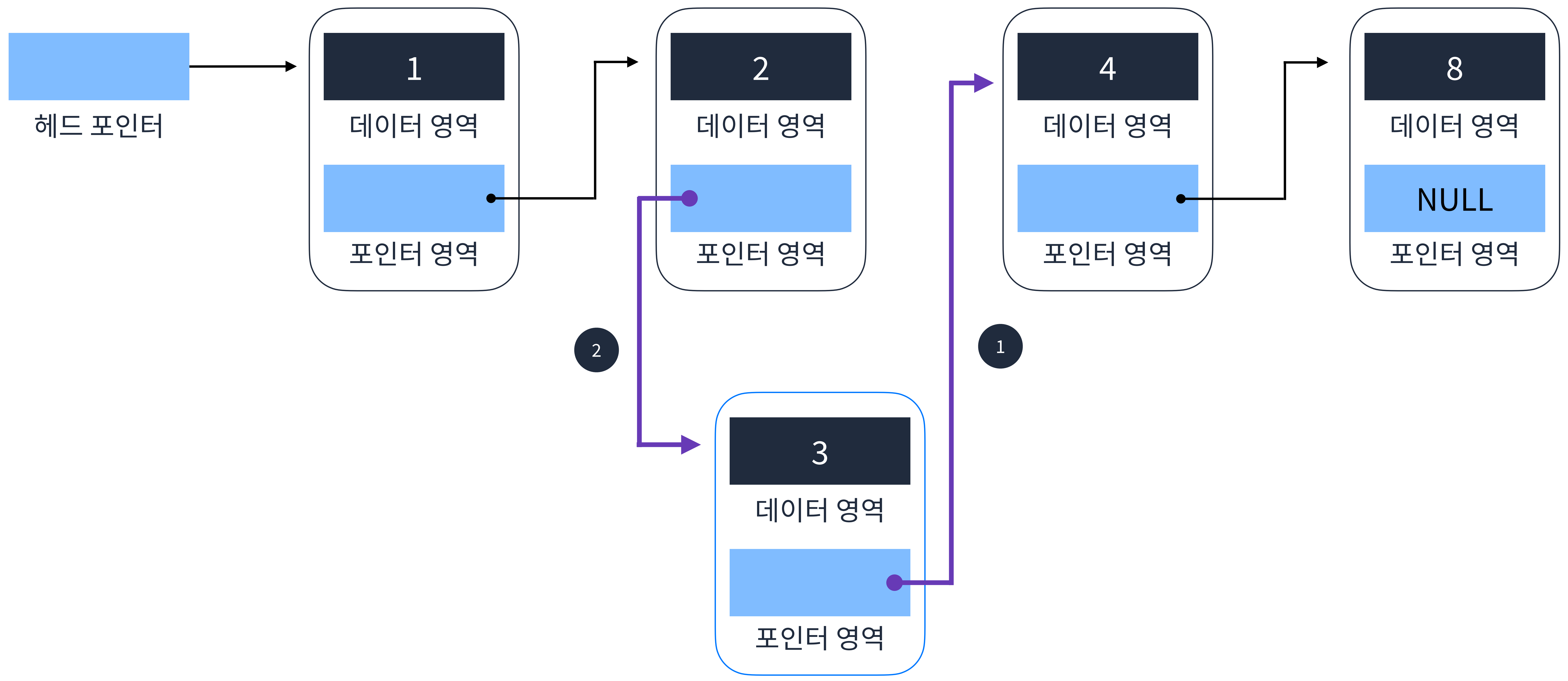
요소 추가

'3'을 중간에 추가한다면?



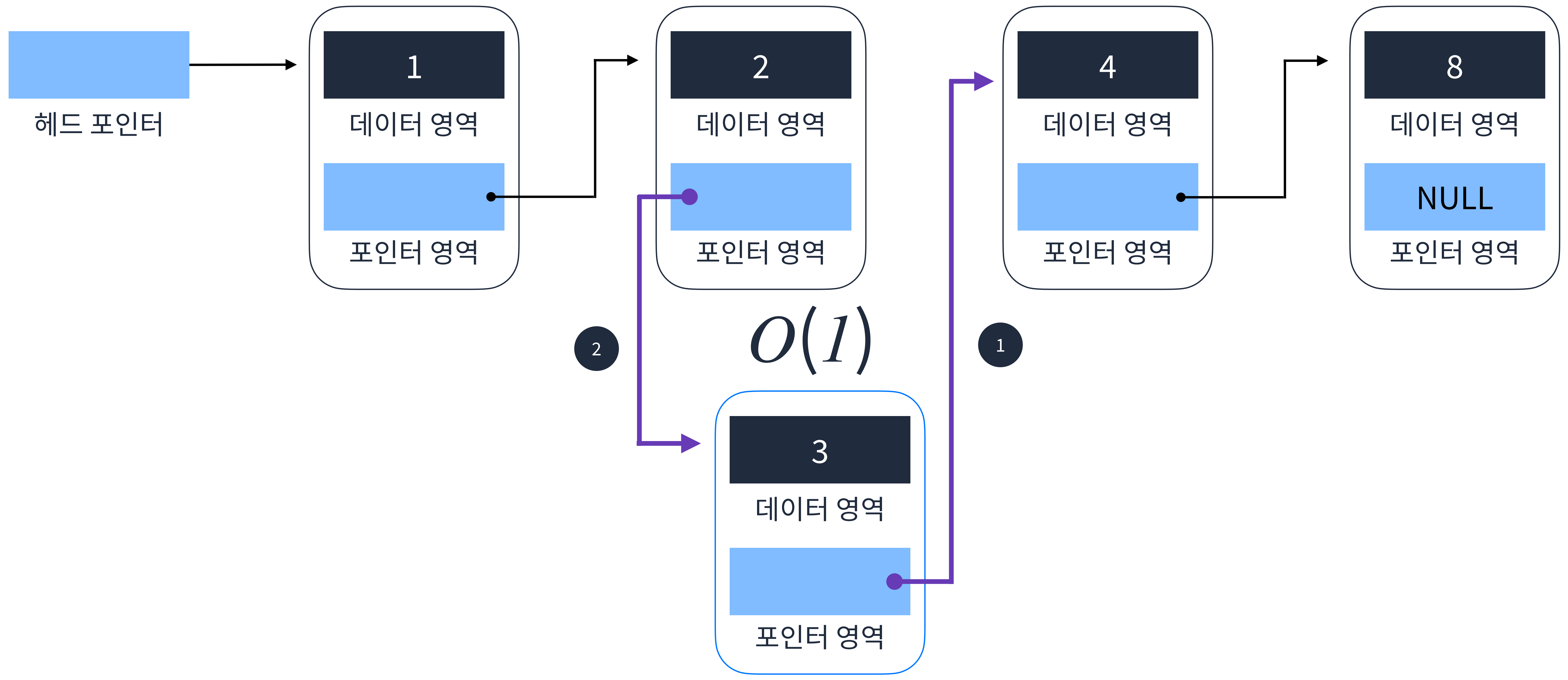
요소 추가

'3'을 중간에 추가한다면?



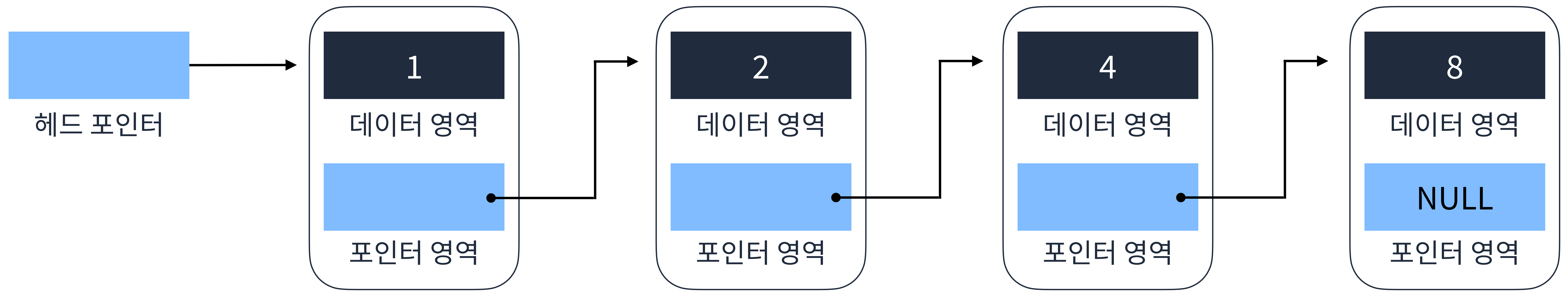
요소 추가

'3'을 중간에 추가한다면?



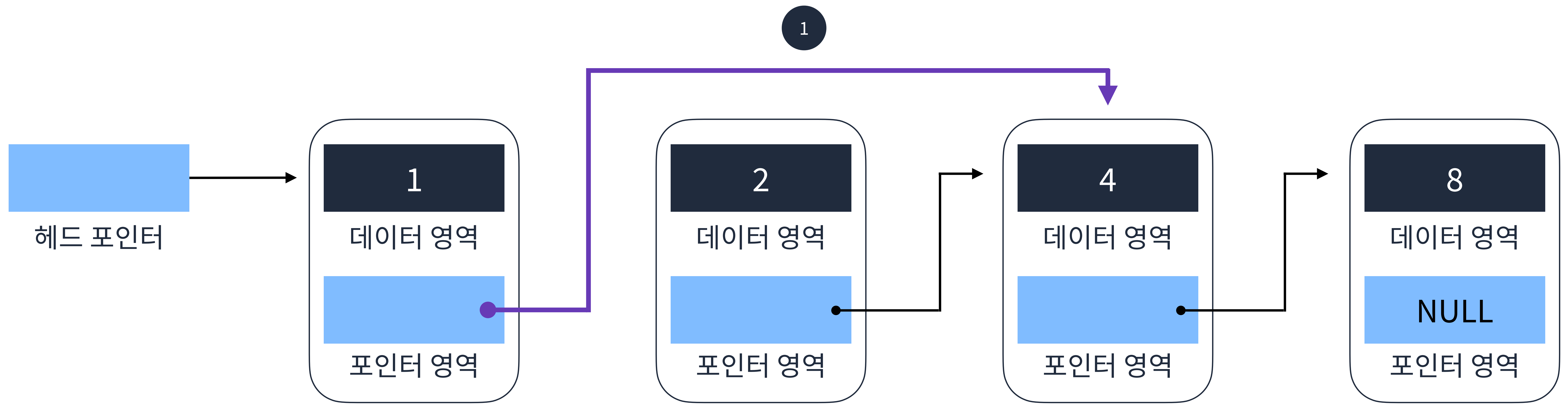
요소 삭제

'2'를 삭제한다면?



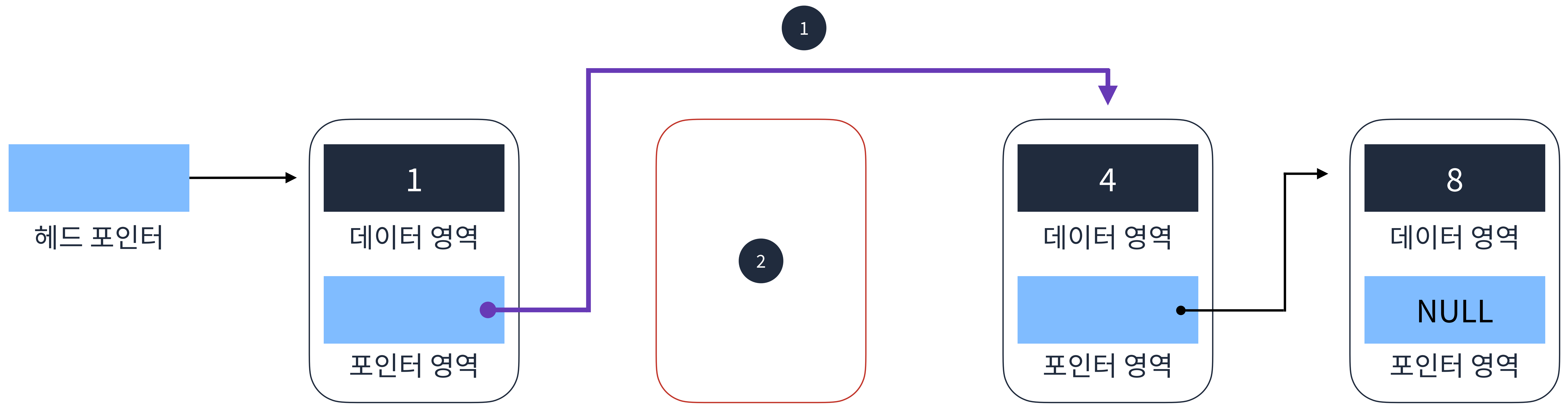
요소 삭제

'2'를 삭제한다면?



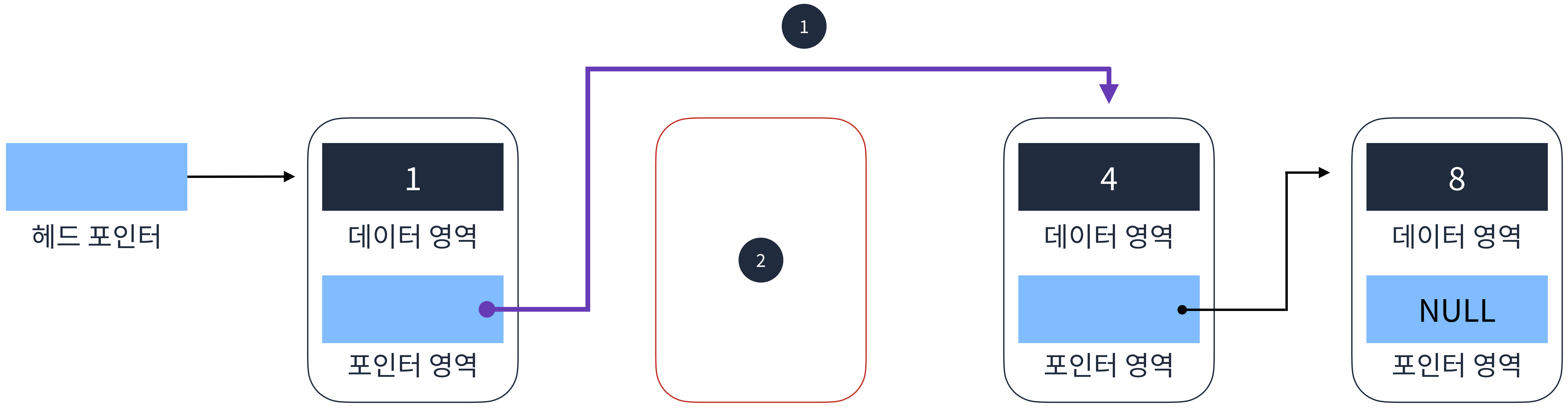
요소 삭제

'2'를 삭제한다면?



요소 삭제

'2'를 삭제한다면?

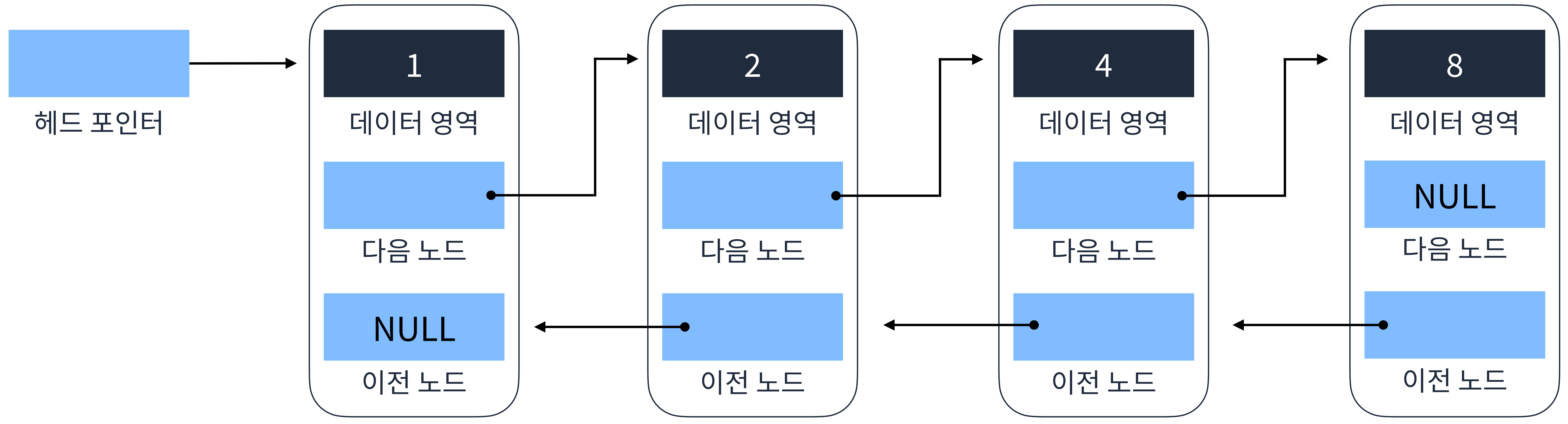


$O(1)$

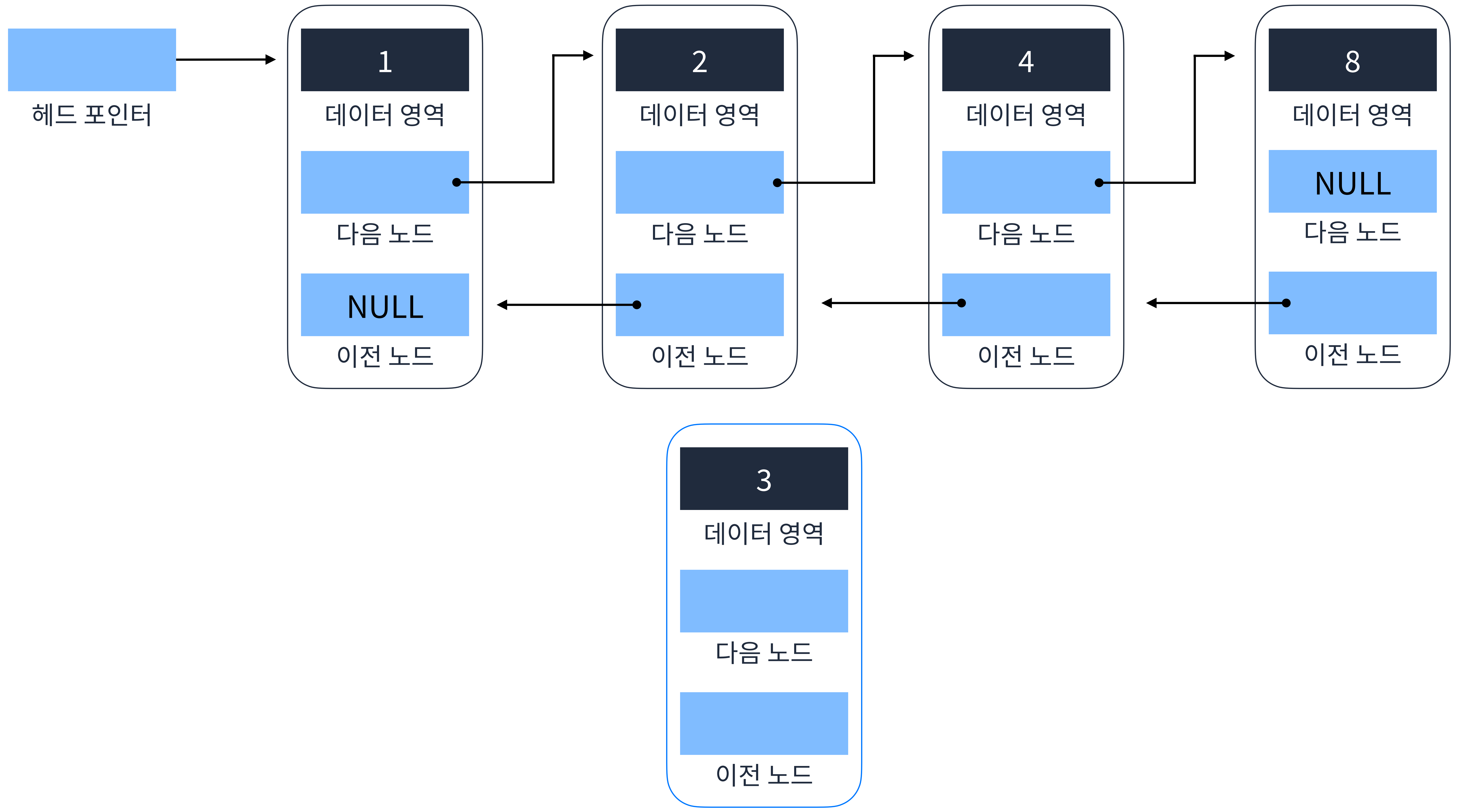
Doubly Linked List

Doubly Linked List

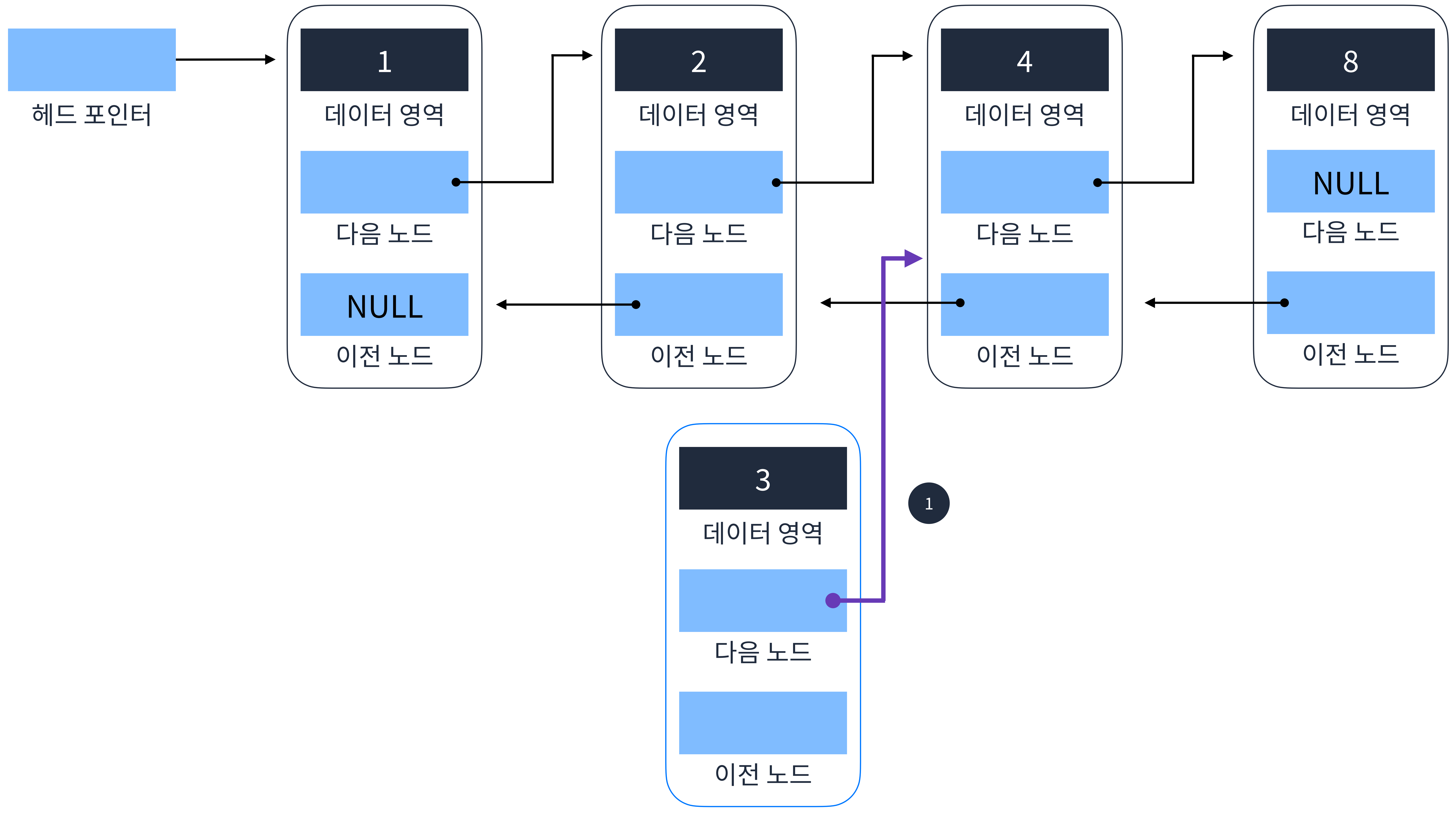
양방향으로 이어지는 연결 리스트
Singly Linked List보다 자료구조의 크기가 조금 더 크다.



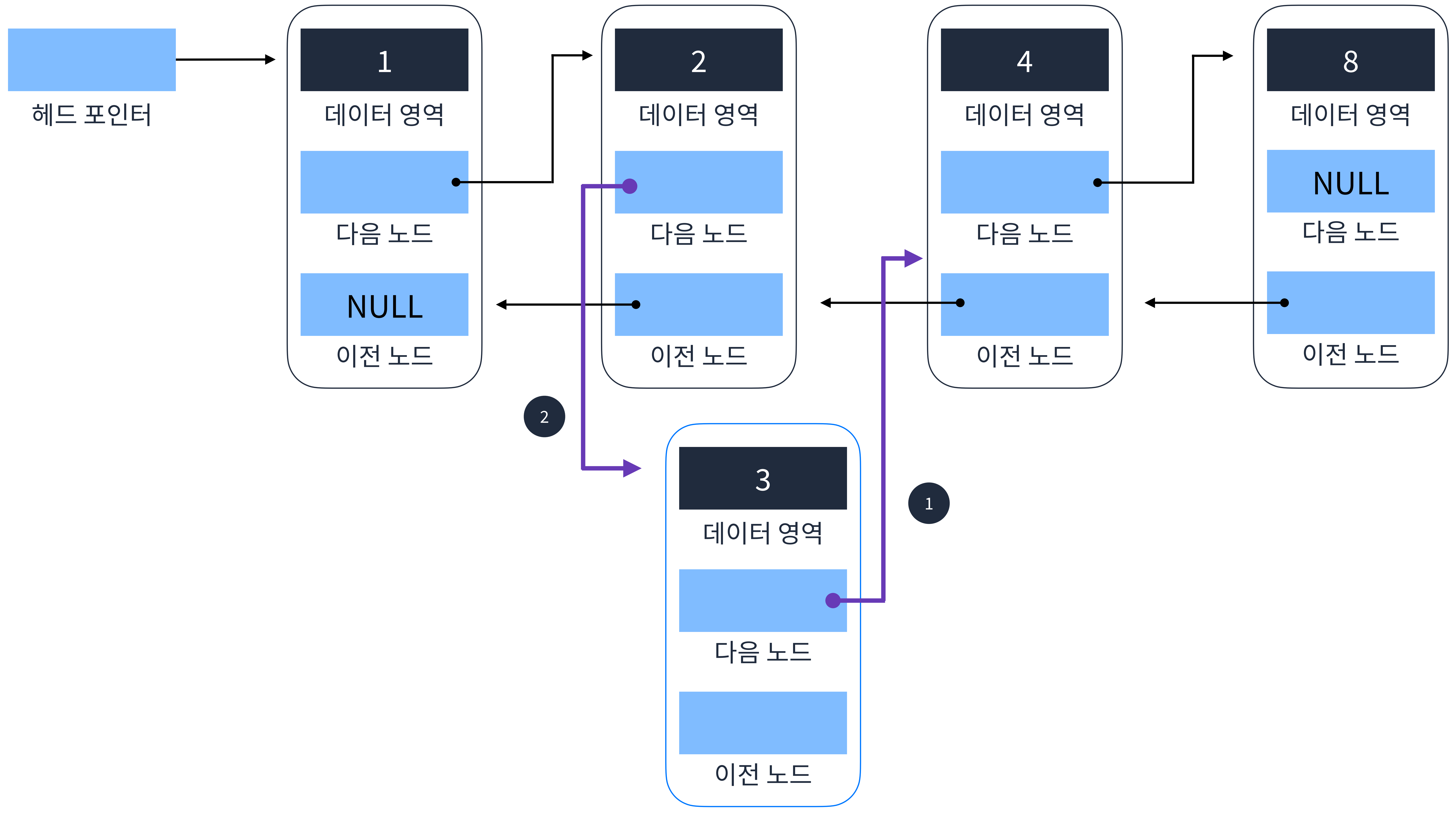
요소 추가


헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드
3
데이터 영역
다음 노드
이전 노드

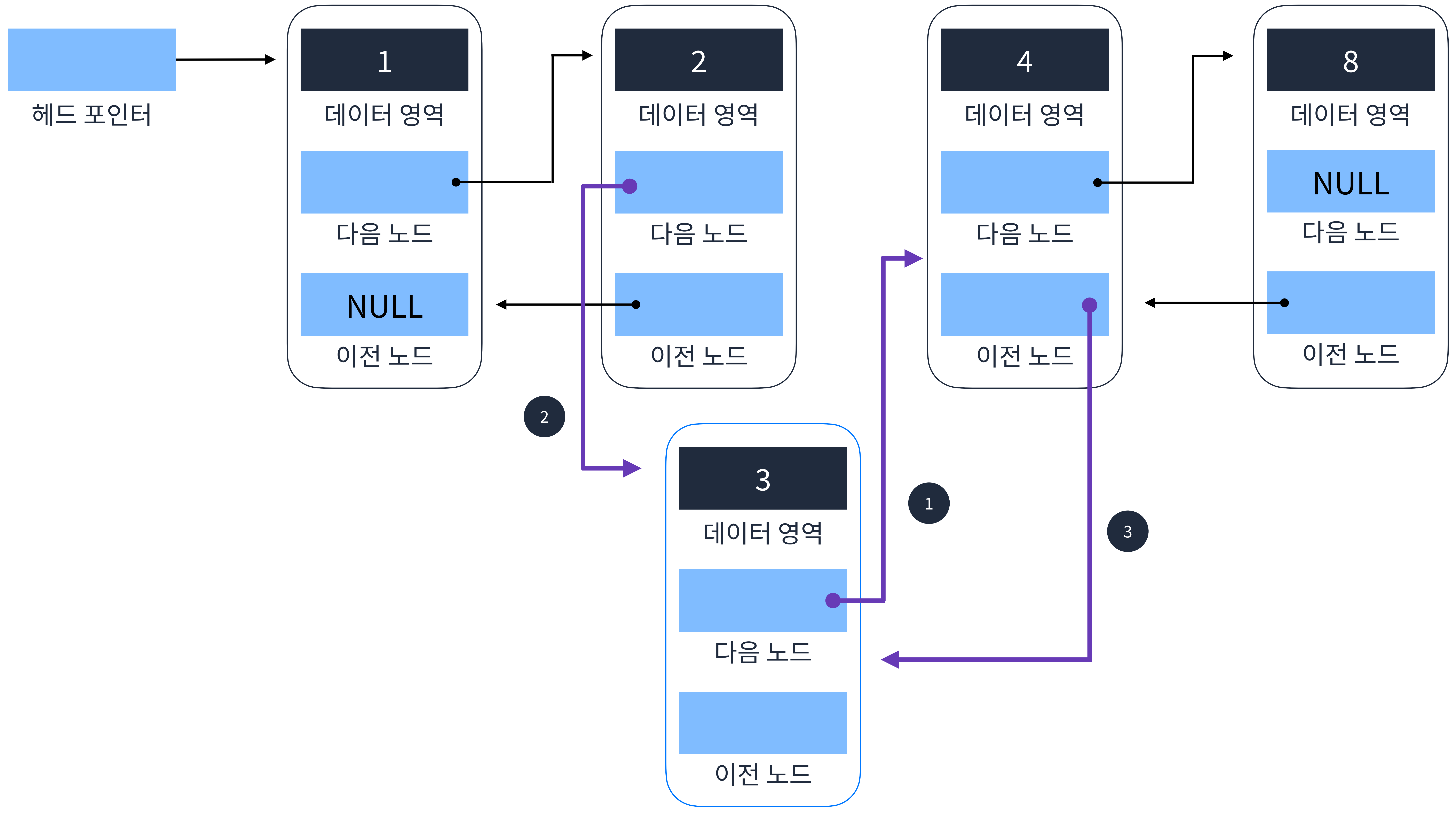
요소 추가


헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드
3
데이터 영역
다음 노드
이전 노드
1

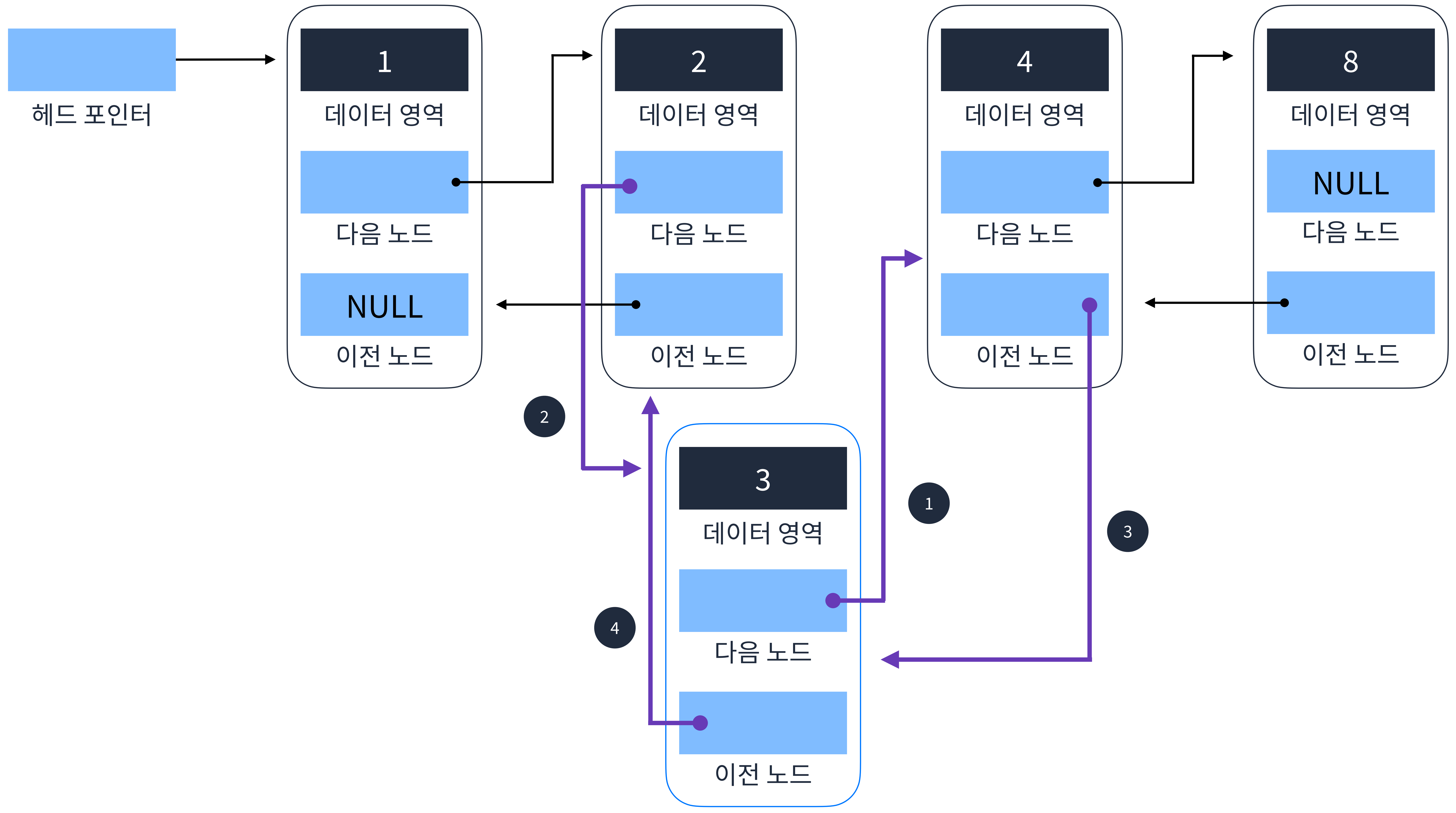
요소 추가


헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드
3
데이터 영역
다음 노드
이전 노드
2
1

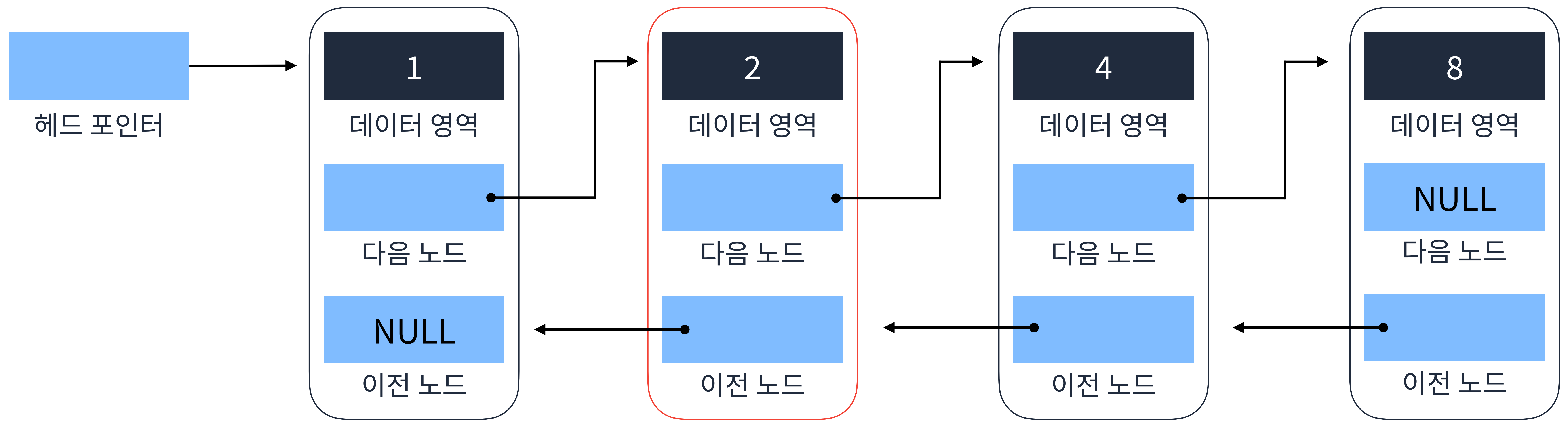
요소 추가


헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드
3
데이터 영역
다음 노드
이전 노드
1
2
3

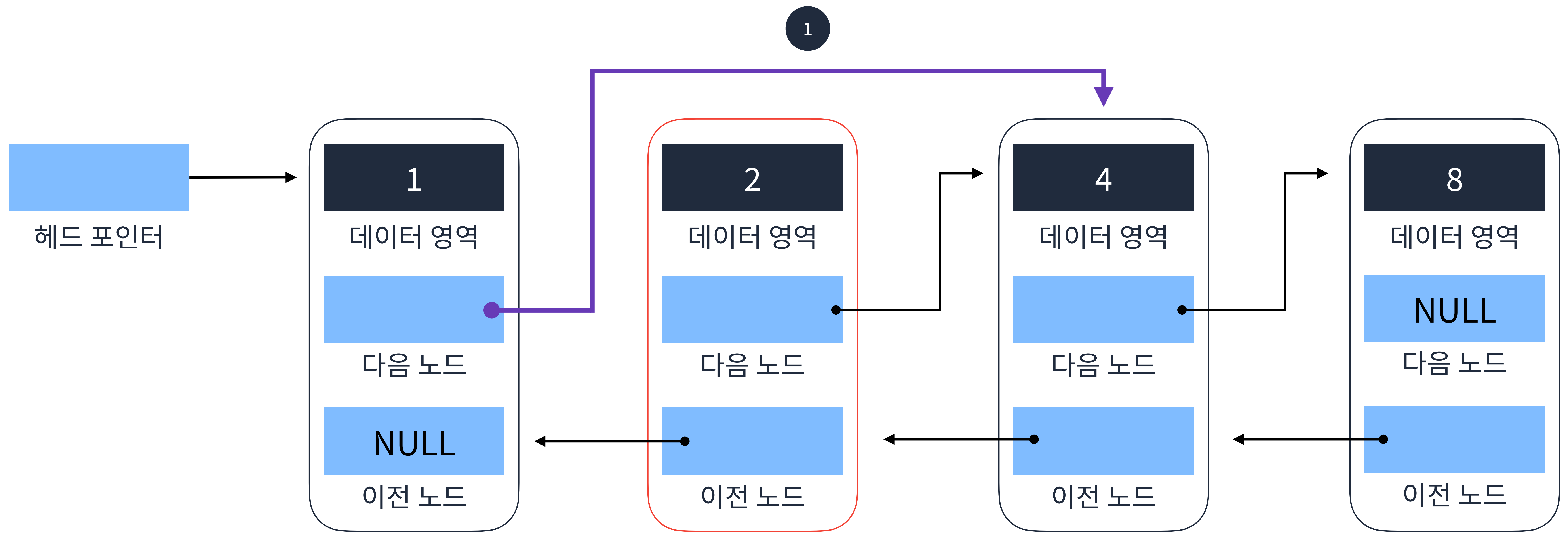
요소 추가


헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드
3
데이터 영역
다음 노드
이전 노드
1
2
3
4

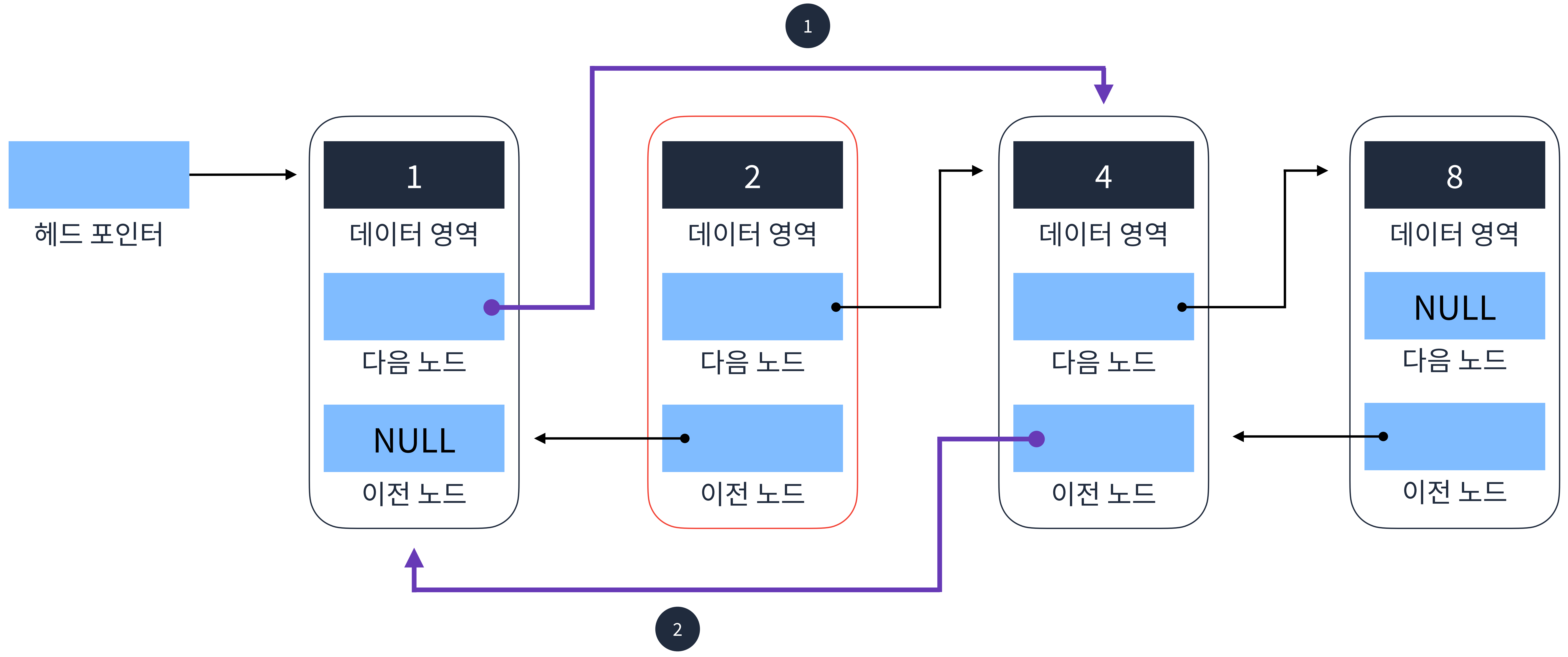
요소 삭제


헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드

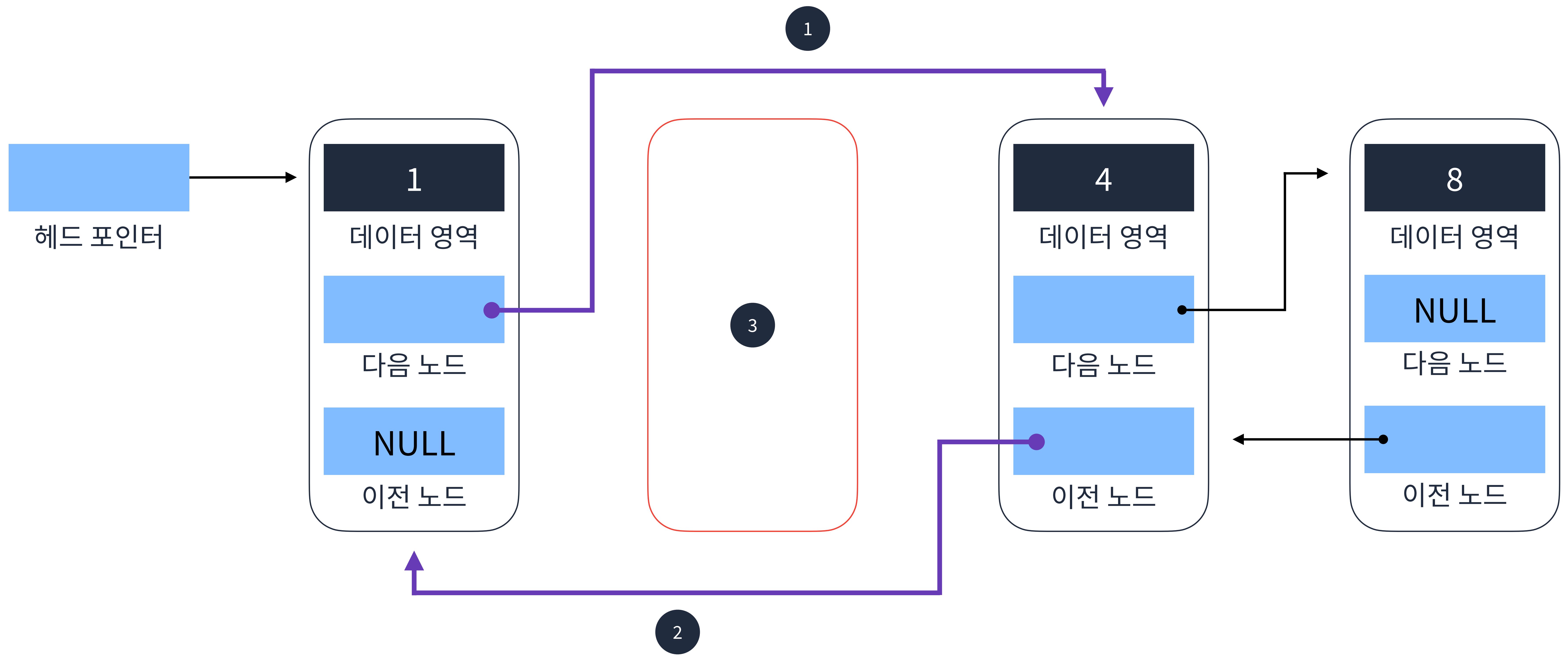
요소 삭제


1
헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드

요소 삭제


1
헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
2
데이터 영역
다음 노드
이전 노드
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드
2

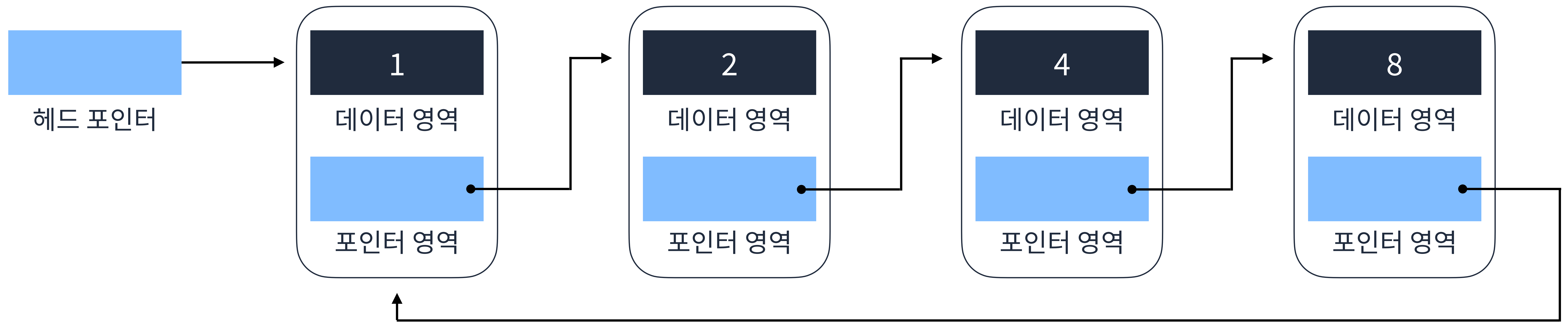
요소 삭제


1
헤드 포인터
1
데이터 영역
다음 노드
NULL
이전 노드
3
2
4
데이터 영역
다음 노드
이전 노드
8
데이터 영역
NULL
다음 노드
이전 노드

Circular Linked List

Circular Linked List

Singly 혹은 Doubly Linked List에서 Tail이 Head로 연결되는 연결 리스트
메모리를 아껴쓸 수 있다. 원형 큐 등을 만들때도 사용된다.



JavaScript 코드

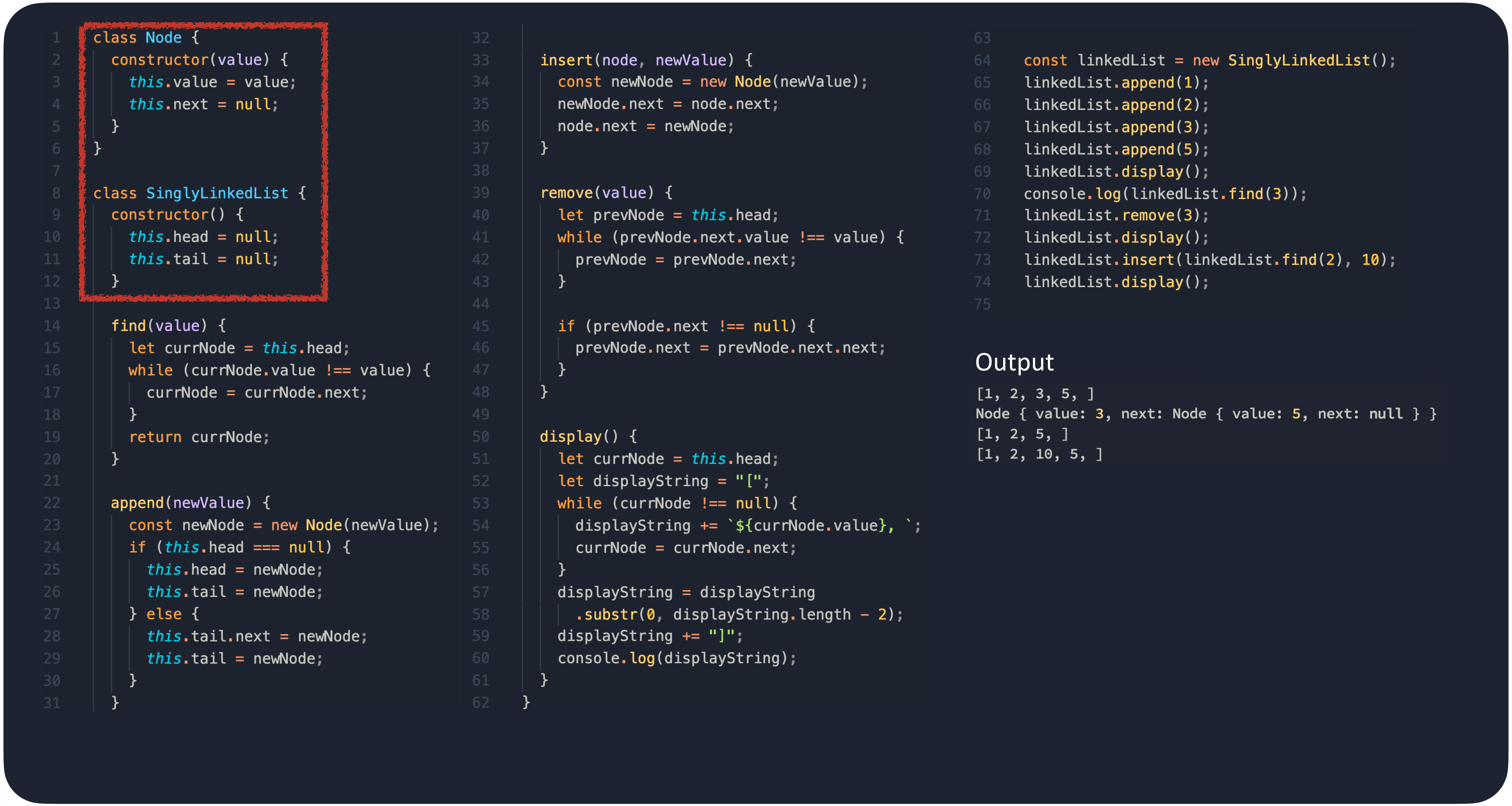
연결 리스트

```javascript
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

연결 리스트

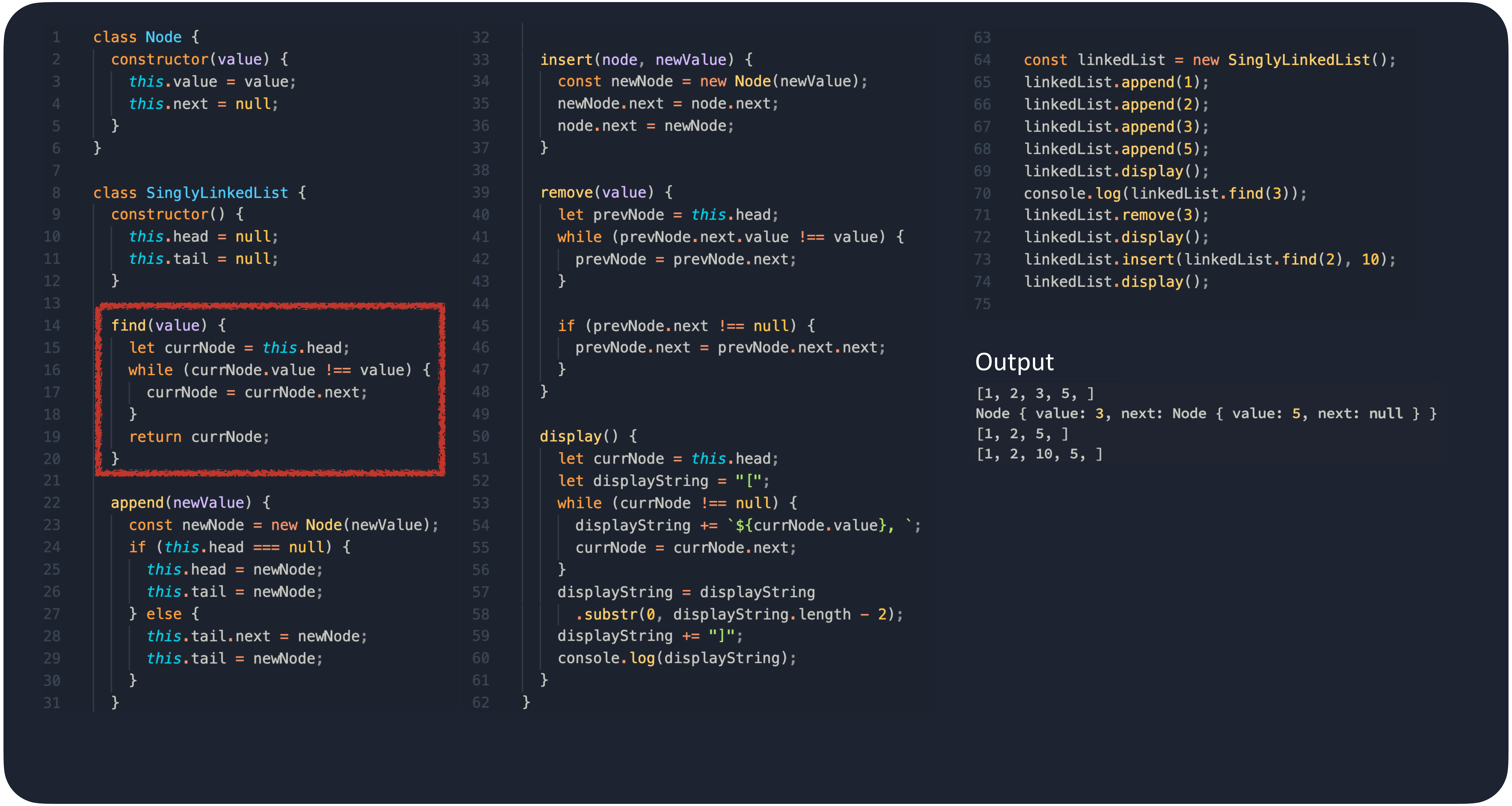


```javascript
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

연결 리스트

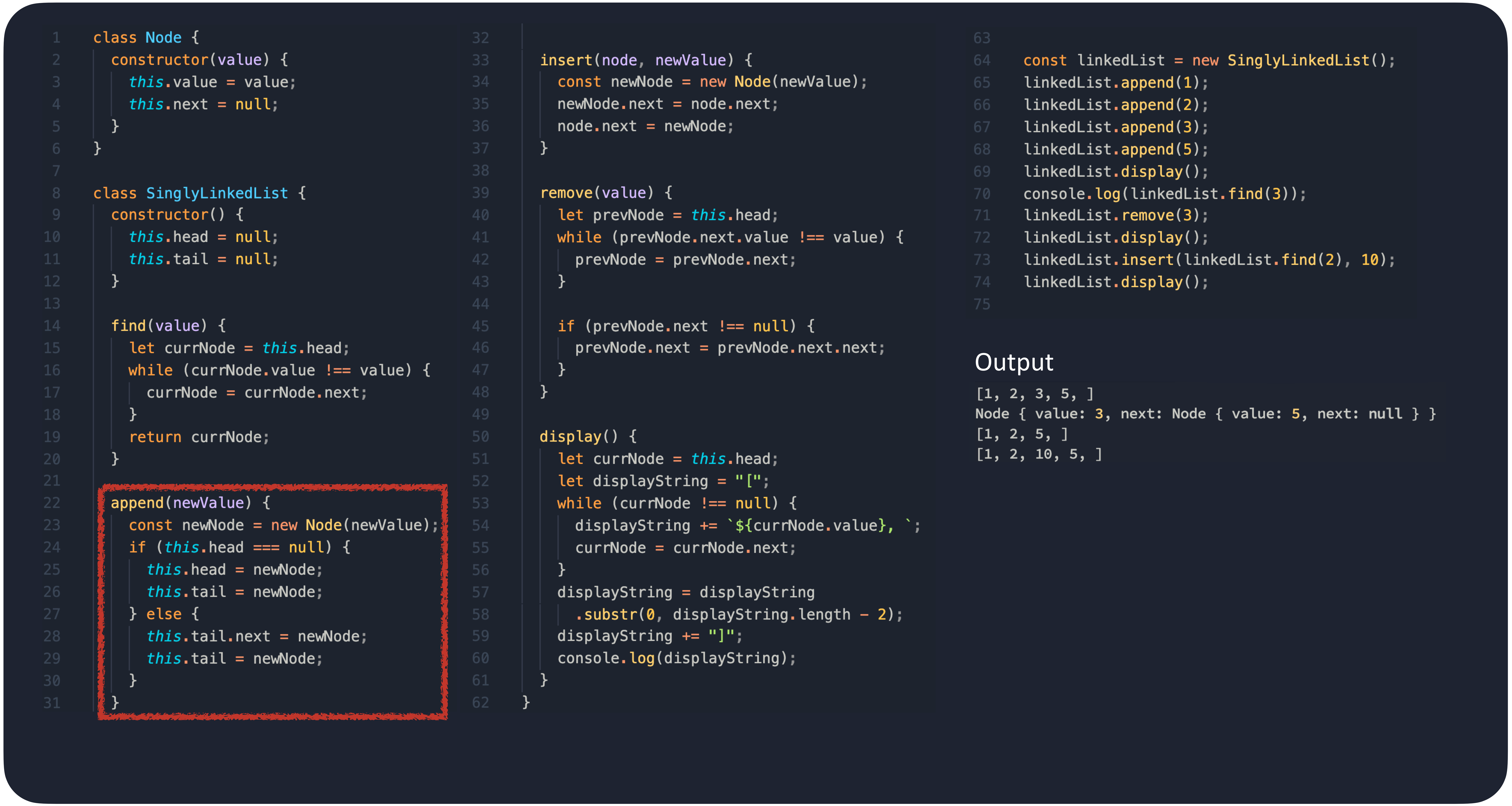


```javascript
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

연결 리스트

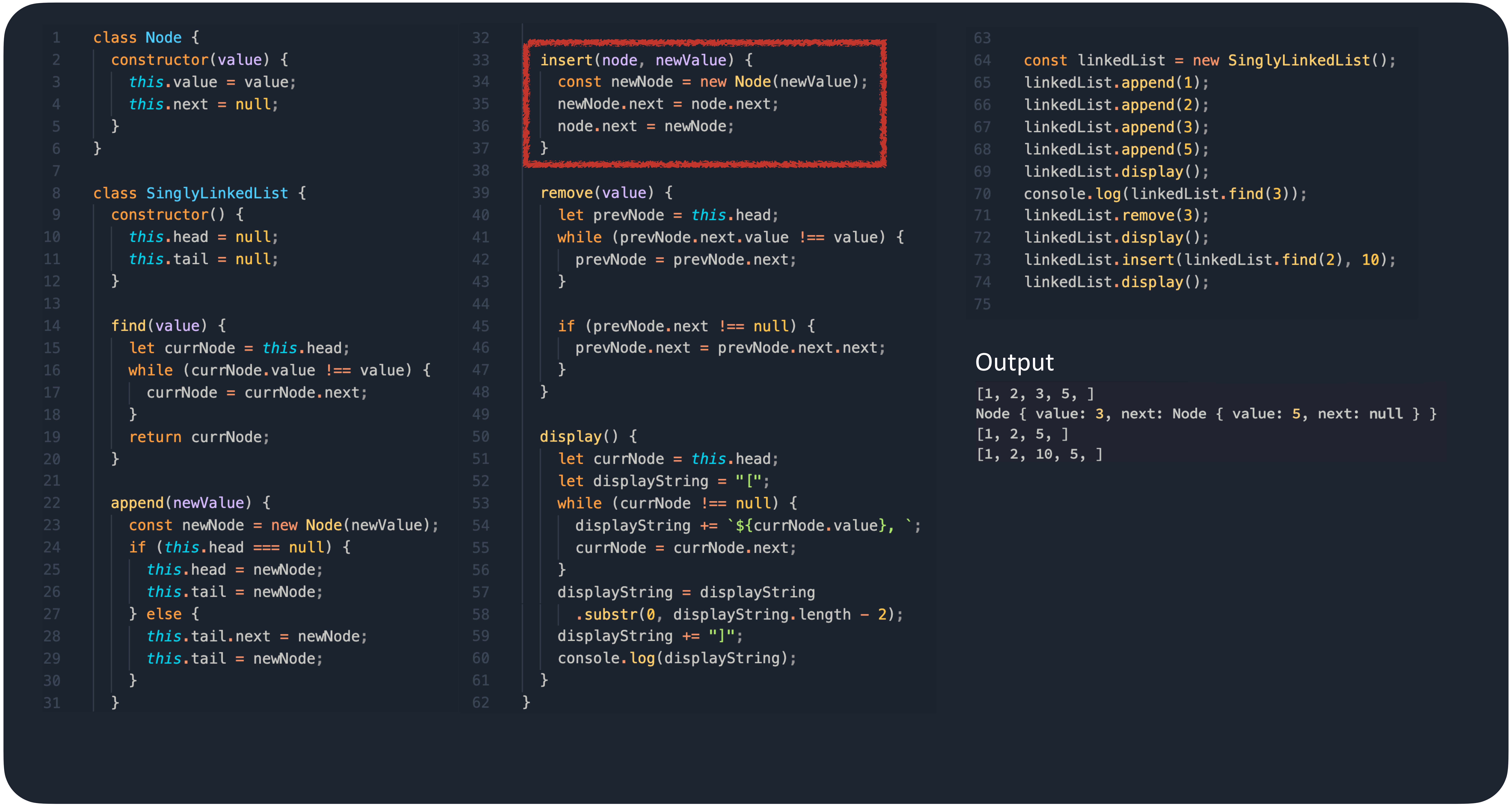


```javascript
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

연결 리스트

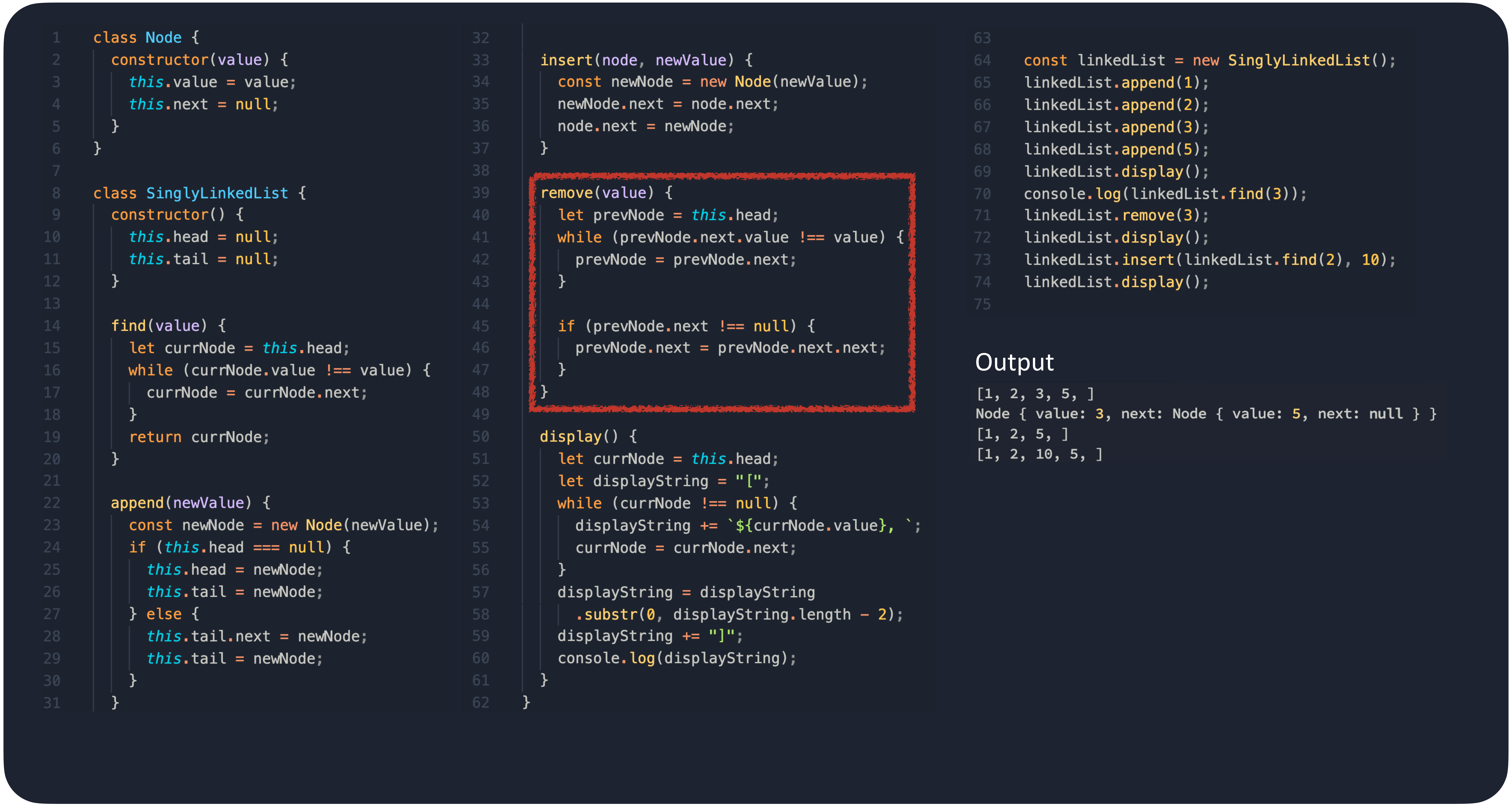


```javascript
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

연결 리스트

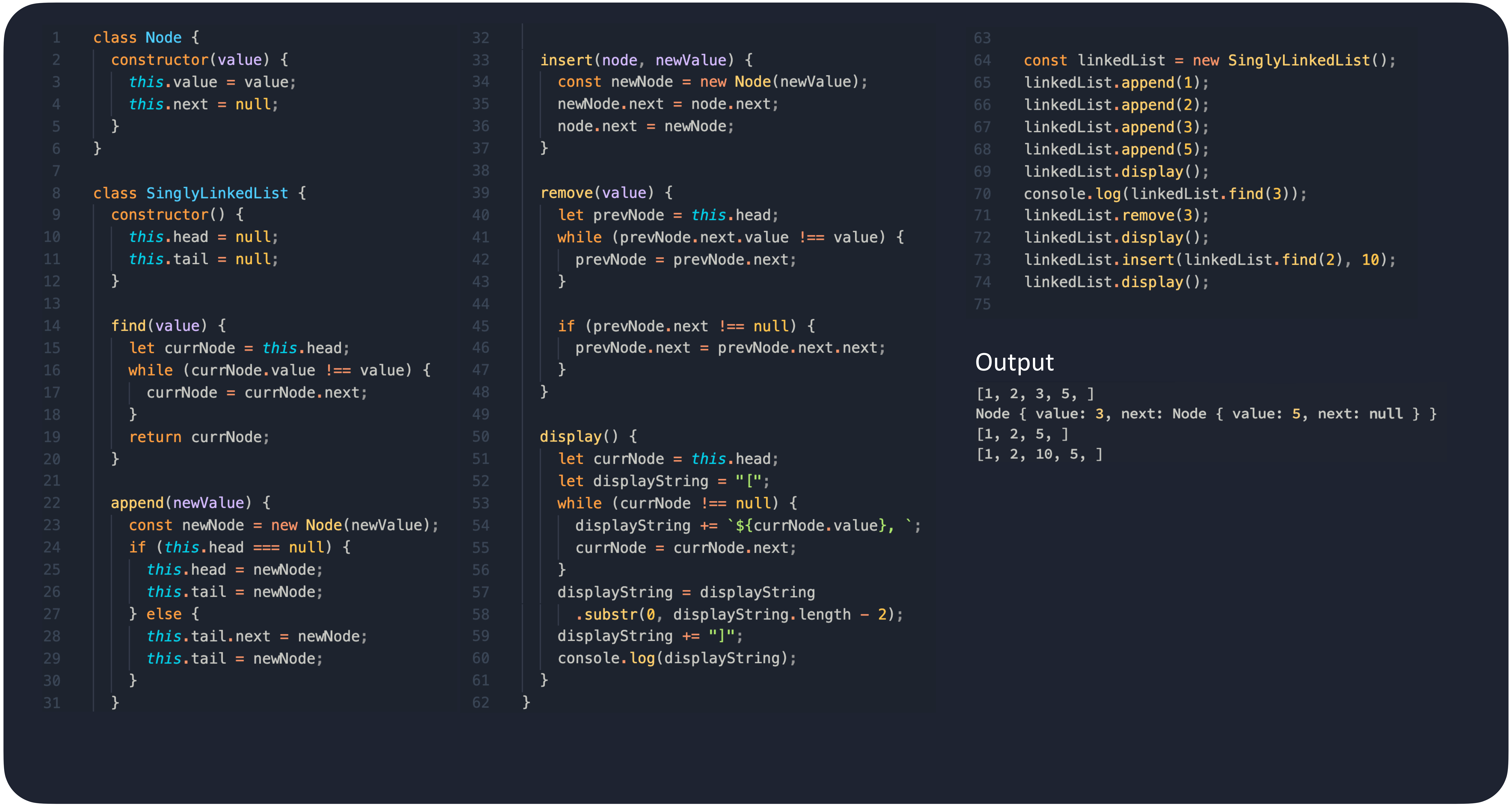


```javascript
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

연결 리스트



```
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class SinglyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  find(value) {
    let currNode = this.head;
    while (currNode.value !== value) {
      currNode = currNode.next;
    }
    return currNode;
  }

  append(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  insert(node, newValue) {
    const newNode = new Node(newValue);
    newNode.next = node.next;
    node.next = newNode;
  }

  remove(value) {
    let prevNode = this.head;
    while (prevNode.next.value !== value) {
      prevNode = prevNode.next;
    }

    if (prevNode.next !== null) {
      prevNode.next = prevNode.next.next;
    }
  }

  display() {
    let currNode = this.head;
    let displayString = "[";
    while (currNode !== null) {
      displayString += `${currNode.value}, `;
      currNode = currNode.next;
    }
    displayString = displayString
      .substr(0, displayString.length - 2);
    displayString += "]";
    console.log(displayString);
  }
}

const linkedList = new SinglyLinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.append(5);
linkedList.display();
console.log(linkedList.find(3));
linkedList.remove(3);
linkedList.display();
linkedList.insert(linkedList.find(2), 10);
linkedList.display();
```

Output

```
[1, 2, 3, 5, ]
Node { value: 3, next: Node { value: 5, next: null } }
[1, 2, 5, ]
[1, 2, 10, 5, ]
```

programmers

연결 리스트

코딩테스트 광탈방지 A to Z : JavaScript - 이선협 @kciter