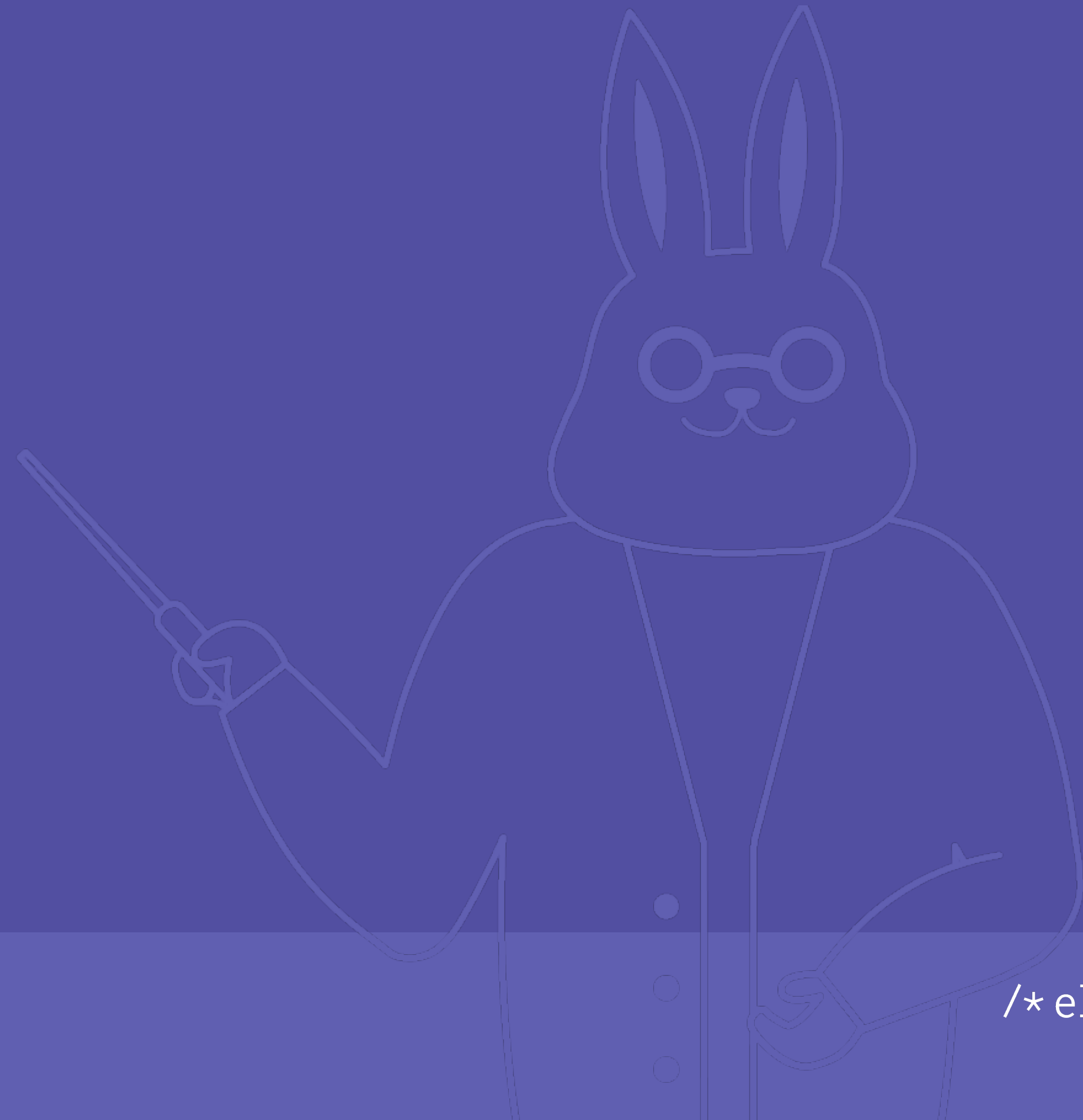


자료구조

1장 자료구조란?

김경민 선생님



Contents

- 01. 자료구조의 의미
- 02. 추상적 자료형
- 03. 배열과 연결 리스트
- 04. 자료구조의 구현 방법
- 05. 구슬 넣기 문제 해결하기
- 06. 주문 관리 시스템 문제 해결하기
- 07. 마무리 및 부록

01

자료구조의 의미



01 자료구조의 의미

✓ 자료구조란?

자료를 저장하는 구조

여러 가지 종류가 있으며
저장된 자료에 대해 접근하는 방법 등의 차이가 존재한다.

01 자료구조의 의미

✓ 자료구조란?

그렇다면, 자료구조는 왜 배울까?

01 자료구조의 의미

✓ 자료구조란?



똑같은 음식을 같은 양만큼 담고 있는 두 그릇이 있다.

여우는 넓은 그릇이 편리하고 두루미는 길쭉한 그릇이 편리하다.

01 자료구조의 의미

✓ 자료구조란?

자료구조 또한 형태에 따라 **장단점**이 존재하며
구현하고자 하는 프로그램의 **성능**을 고려하여 알맞은 자료구조를 선택해야 한다.

01 자료구조의 의미

✓ 자료구조란?

밥상에는 **음식**이 필요한 것처럼, 프로그램에도 **자료**가 빠져서는 안 된다.

프로그램에 필요한 자료를 **효율적**으로 담기 위해 자료구조를 배운다.

01 자료구조의 의미

✓ 자료구조란?

여우는 **접시**를, 두루미는 **호리병**을 써야
행복하게 음식을 먹을 수 있다.

음식을 담는 그릇도 **먹는 사람, 먹을 음식**을 고려하여
적절한 것을 선택해야 한다.

01 자료구조의 의미

✓ 자료구조란?

마찬가지로 **프로그램**에서 특정 **알고리즘**을 구현하기 위해
적절한 **자료구조**를 사용해야 좋은 **성능**을 낼 수 있다.

02

추상적 자료형



02 추상적 자료형

✓ 추상적 자료형이란?

어떤 자료와 그 자료에 대한 연산(동작)들의 수학적인 **정의**를 의미한다.

그리고 그 정의를 **구현**하는 방법은 **명시되어 있지 않다.**

02 추상적 자료형

✓ 추상적 자료형이란?

"추상적"이라는 단어의 의미도 뭔가 추상적이다.

"추상적 자료형"이란 용어를 파악하기 위해서,

자료형이 무엇인지부터 알아보자.

02 추상적 자료형

✓ 자료형

자료형은 어떤 자료가 **식별**될 수 있는 방법과,
그 자료에 대한 여러 가지 **연산(동작)**을 제공한다.

02 추상적 자료형

✓ 자료형의 중요성 - 값의 해석 방법 명시

65

예를 들어, 65라는 자료가 있다.

컴퓨터는 이 자료가 **수**를 나타내는 65인지, 아니면 **알파벳** 'A' 를 나타내는 값인지
자료형을 모르는 경우에는 해석할 수 없다.

(컴퓨터는 내부적으로 unicode라고 불리는 숫자를
문자에 대입하는 방식으로 각 문자를 식별하고, 65는 'A'에 해당한다.)

`/* elice */`

02 추상적 자료형

✓ 자료형의 중요성 - 연산 방법 제공

65

또, 수를 나타내는 자료형은
덧셈, 뺄셈 등을 비롯한 연산이 가능하다.

02 추상적 자료형

✓ 자료형

이처럼, 자료형은 어떤 자료가 **식별**되는 방법을 정의하고
자료에 적용할 수 있는 **연산**을 결정한다.

자료를 특정 분류에 따라 올바르게 **표현**하기 위한 정의와,
그 **구현**이 바로 자료형이라고 할 수 있다.

02 추상적 자료형

✓ 추상적 자료형

추상적 자료형은 **구현 방법**을 명시하고 있지 않다는 점이 특징이다.

02 추상적 자료형

✓ 추상적 자료형

그릇을 추상적 자료형이라고 한다면,
그릇이 보관하는 **자료**와, 그 자료에 해당하는 **연산**의 정의를 포함해야 한다.

02 추상적 자료형

✓ 추상적 자료형

자료 연산
그릇은 **음식**을 보관하고, 그릇에 담긴 음식을 **먹을 수 있다**.

02 추상적 자료형

✓ 추상적 자료형

이때 음식을 어떻게 먹을 수 있는지,

즉 어떻게 자료에 접근할 수 있는지는 주어지지 않는다.

`/* elice */`

02 추상적 자료형

✓ 추상적 자료형과 자료구조

'그릇'이라는 추상적 자료형이 있고,

그릇에 음식을 어떻게 담을 것인지,
그릇을 이용하여 어떻게 음식을 먹을 수 있는지
명확히 구현된 것을 자료구조라고 할 수 있다.

02 추상적 자료형

✓ 추상적 자료형과 자료구조

그릇(추상적 자료형)

음식을 담을 수 있고, 그릇에 담긴 음식을 먹을 수 있다.

구현

접시(자료구조)

평평한 면에 음식을 얹을 수 있고, 넓은 면적을 활용하여
수저로 음식을 먹기 쉽다.

호리병(자료구조)

길쭉한 입구를 이용하여 음식을 저장할 수 있으며
액체의 저장에 유리하다.
병을 들고 마시거나, 빨대를 사용할 수 있다.

02 추상적 자료형

✓ 정리

추상적 자료형 : 자료들과, 그 자료들에 대한 연산들을 개념적으로 정의만 한 것

자료구조 : 자료를 저장하는 방법과 자료에 적용할 수 있는 연산을 **구체적으로** 제공한 것

추상적 자료형을 **구체적으로 구현**한 결과가 자료구조

03

배열과 연결 리스트



03 배열과 연결 리스트

✓ 시작하기 앞서

3챕터 '배열과 연결 리스트'에서는 **리스트**라는 개념이 등장합니다.

파이썬(Python)의 **리스트**와는 다른 개념으로, 용어의 혼선을 없애기 위해 기존에 알고 있던 **리스트**는 **배열(Array)**이라는 이름으로 표기하고 있습니다.

`/* elice */`

03 배열과 연결 리스트

✓ 리스트(List)

리스트라는 이름의 추상적 자료형이 있다.

`/* elice */`

03 배열과 연결 리스트

✓ 리스트가 담는 자료

순서가 존재하며, **일렬**로 나열된 값들이 들어있다.

03 배열과 연결 리스트

✓ 리스트가 담는 자료에 적용되는 연산

조회 : 임의의 위치의 자료가 무엇인지 **알아본다**.

삽입 : 임의의 위치에 자료를 **추가**한다.

삭제 : 임의의 위치의 자료를 **제거**한다.

이외에도 다양한 연산이 있다.

03 배열과 연결 리스트

✓ 배열(Array)

리스트라는 추상적 자료형을 구현한 대표적인 예시가 바로 '**배열**'이다.

(Python에서는 배열을 리스트라는 이름으로 제공한다.)

(본 과목에서 말하는 추상적 자료형을 뜻하는 "리스트"와 다른 개념이다.)

03 배열과 연결 리스트

✓ 배열

배열에 저장되는 값들은 순서를 나타내는 번호를 가진다.
이 번호를 **인덱스**라고 부른다.

	0	1	2	3	4	5	6	7
myList	3	4	2	5	1	2	6	3

03 배열과 연결 리스트

✓ 배열 - 조회

따라서 배열 내의 특정 순서의 값을 조회하고자 할 때
단번에 찾아낼 수 있다.

	0	1	2	3	4	5	6	7
myList	3	4	2	5	1	2	6	3

5번 인덱스의 값은?

`/* elice */`

03 배열과 연결 리스트

✓ 배열 - 조회

따라서 배열 내의 특정 순서의 값을 조회하고자 할 때
단번에 찾아낼 수 있다.

	0	1	2	3	4	5	6	7
myList	3	4	2	5	1	2	6	3

2!

/* elice */

03 배열과 연결 리스트

✓ 배열 - 삽입

반면에 자료를 **추가**할 땐 조금 번거롭다.

새로운 자료가 추가되면서, 자료들의 **순서**를 **변경**해주어야 하기 때문이다.

myList	0	1	2	3	4	5	6	7
	3	4	2	5	1	2	6	3

3번 인덱스에 10을 추가하자.

`/* elice */`

03 배열과 연결 리스트

✓ 배열 - 삽입

우선, 새로운 자료가 들어갈 공간을 마련해주어야 한다.

배열에 들어갈 자료가 하나 늘었으므로 공간도 하나 더 만들어준다.

myList	0	1	2	3	4	5	6	7	8
	3	4	2	5	1	2	6	3	

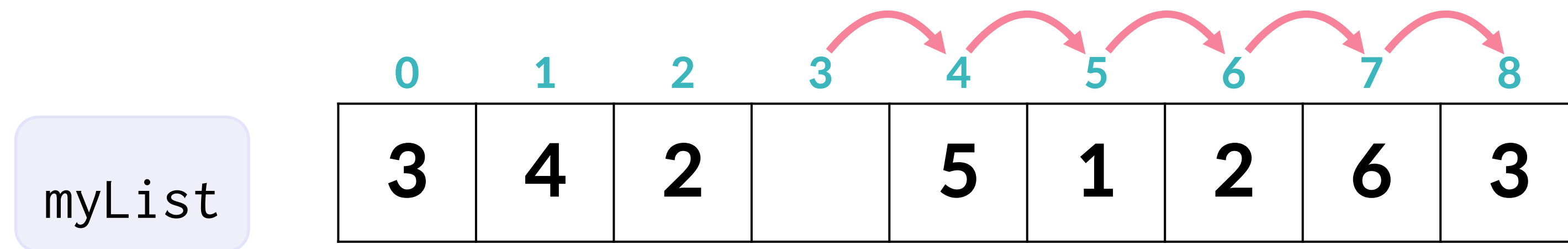
3번 인덱스에 10을 추가하자.

`/* elice */`

03 배열과 연결 리스트

✓ 배열 - 삽입

추가될 자료가 들어갈 공간을 비워주기 위해
기존 자료들은 한 칸씩 밀려나게 된다.



3번 인덱스에 10을 추가하자.

`/* elice */`

03 배열과 연결 리스트

✓ 배열 - 삽입

마련된 빈 자리에 추가될 값을 넣어준다.

myList

0	1	2	3	4	5	6	7	8
3	4	2	10	5	1	2	6	3

추가 완료!

03 배열과 연결 리스트

✓ 배열 - 삭제

자료를 **제거**할 때는 이와 반대로 진행하면 된다.

	0	1	2	3	4	5	6	7	8
myList	3	4	2	10	5	1	2	6	3

5번 인덱스의 자료를 제거하자.

`/* elice */`

03 배열과 연결 리스트

✓ 배열 - 삭제

제거할 인덱스의 자료를 비운다.

myList

0	1	2	3	4	5	6	7	8
3	4	2	10	5		2	6	3

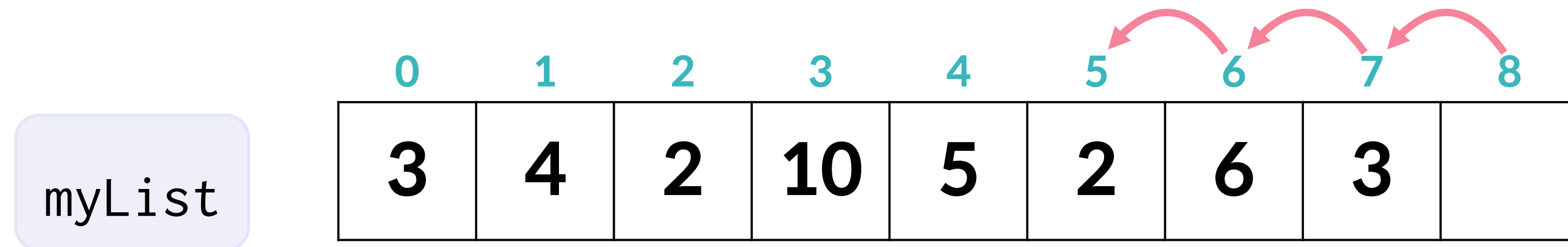
5번 인덱스의 자료를 제거하자.

`/* elice */`

03 배열과 연결 리스트

✓ 배열 - 삭제

자료가 줄어들었으므로,
빈 자리를 메꾸기 위해 자료들이 한 칸씩 당겨진다.



5번 인덱스의 자료를 제거하자.

```
/* elice */
```


03 배열과 연결 리스트

✓ 배열 - 삭제

배열이 담고 있는 자료가 하나 줄었으므로
남는 공간도 제거한다.

myList	0	1	2	3	4	5	6	7
	3	4	2	10	5	2	6	3

제거 완료!

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트(Linked List)

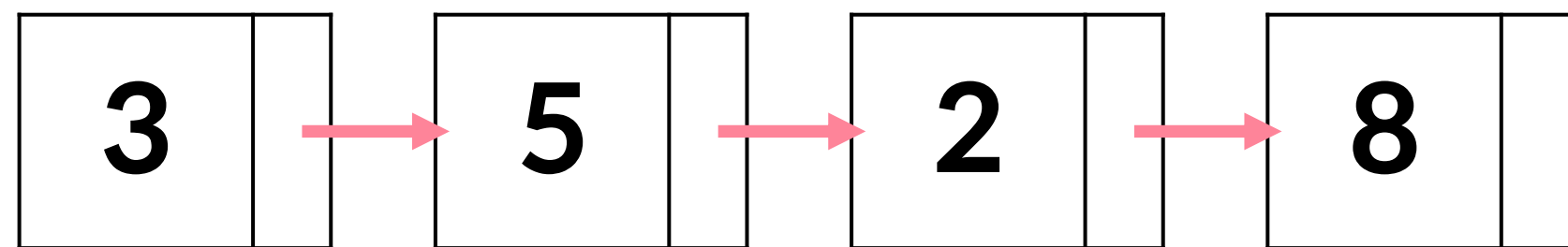
리스트를 구현한 자료구조 중
대표적인 다른 예시는 **연결 리스트**이다.

03 배열과 연결 리스트

✓ 연결 리스트(Linked List)

배열은 각 자료에 인덱스를 붙여 저장하는 반면,
연결 리스트는 여러 개의 '**노드**'를 저장하는 방식으로 구현한다.

myLinkedList

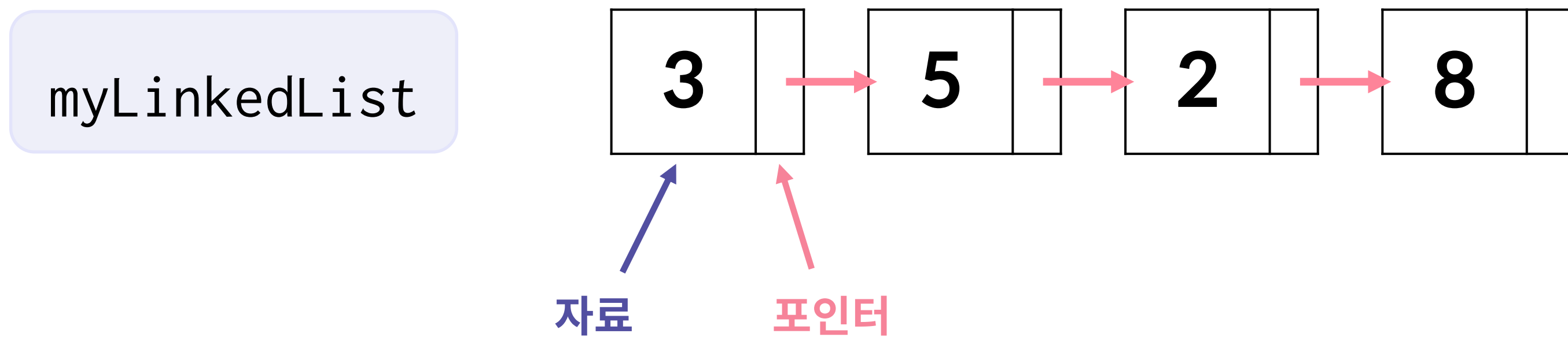


`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트(Linked List)

노드는 **자료**와 **포인터**를 가진다.

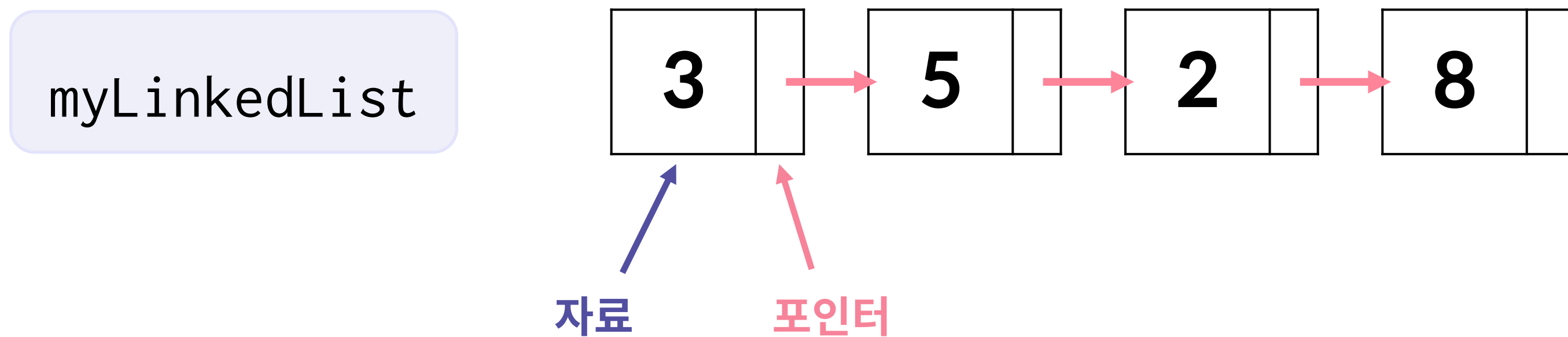


`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트(Linked List)

자료는 저장하는 **값** 자체를 의미하고,
포인터는 자신의 다음 순서의 **노드**를 가리킨다.



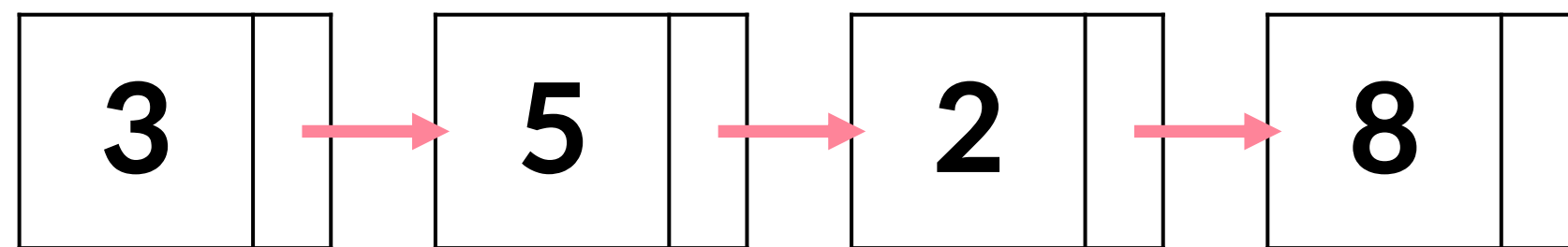
`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 조회

이러한 특성 때문에, 연결 리스트에서
특정 위치의 자료를 찾는 것이 번거롭다.

myLinkedList



3번째 자료의 값은?

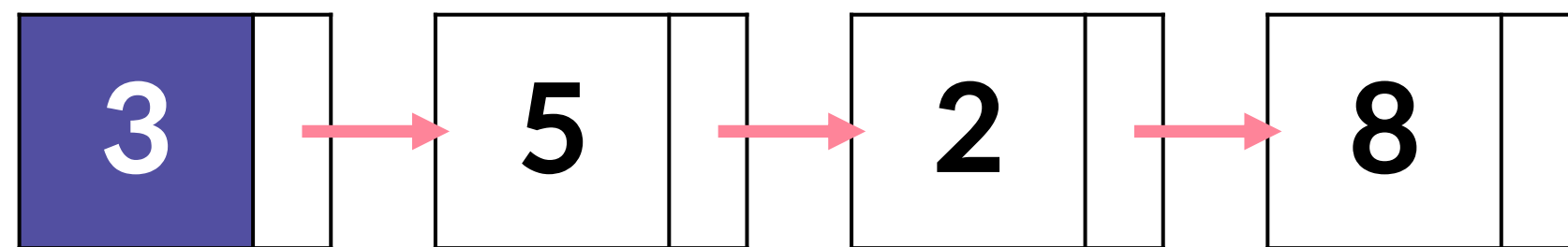
`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 조회

이러한 특성 때문에, 연결 리스트에서
특정 위치의 자료를 찾는 것이 번거롭다.

myLinkedList



3번째 자료의 값은?

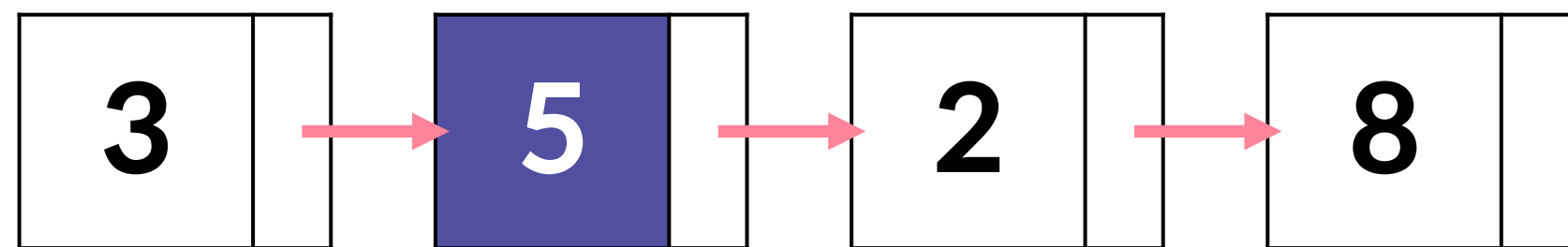
`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 조회

이러한 특성 때문에, 연결 리스트에서
특정 위치의 자료를 찾는 것이 번거롭다.

myLinkedList



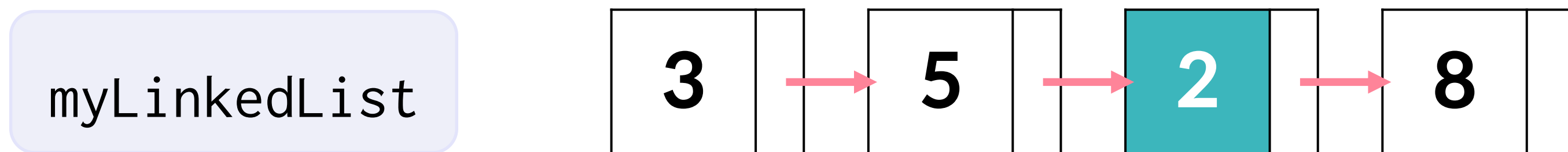
3번째 자료의 값은?

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 조회

이러한 특성 때문에, 연결 리스트에서
특정 위치의 자료를 찾는 것이 번거롭다.



2!

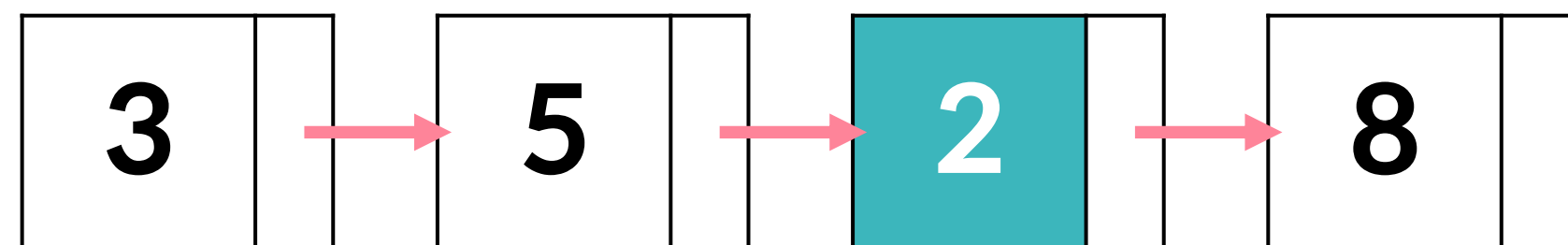
/* elice */

03 배열과 연결 리스트

✓ 연결 리스트 - 조회

배열은 찾는 자료의 위치와 관계없이 **항상 단번에** 값을 찾을 수 있지만,
연결 리스트는 찾는 자료의 위치가 **시작점으로부터 멀수록** 연산 횟수가 많아진다.

myLinkedList



2!

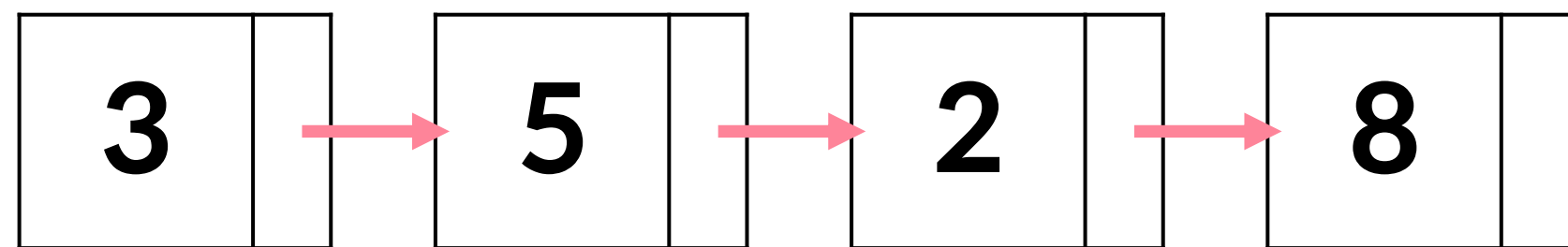
`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트

연결 리스트는 **자료의 추가, 삭제**에서 그 진가를 발휘한다.

myLinkedList



`/* elice */`

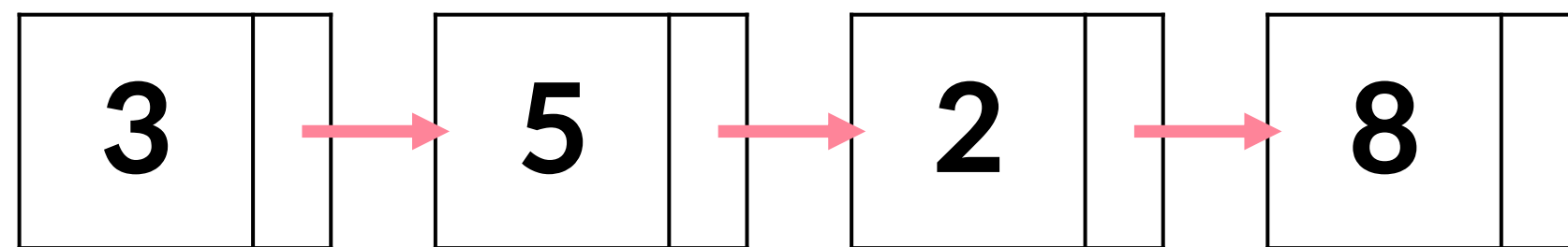
03 배열과 연결 리스트

✓ 연결 리스트 - 삽입

조회 연산과 마찬가지로

연결 리스트 상에서 n번째 위치를 찾아야 하는 과정은 동일하다.

myLinkedList

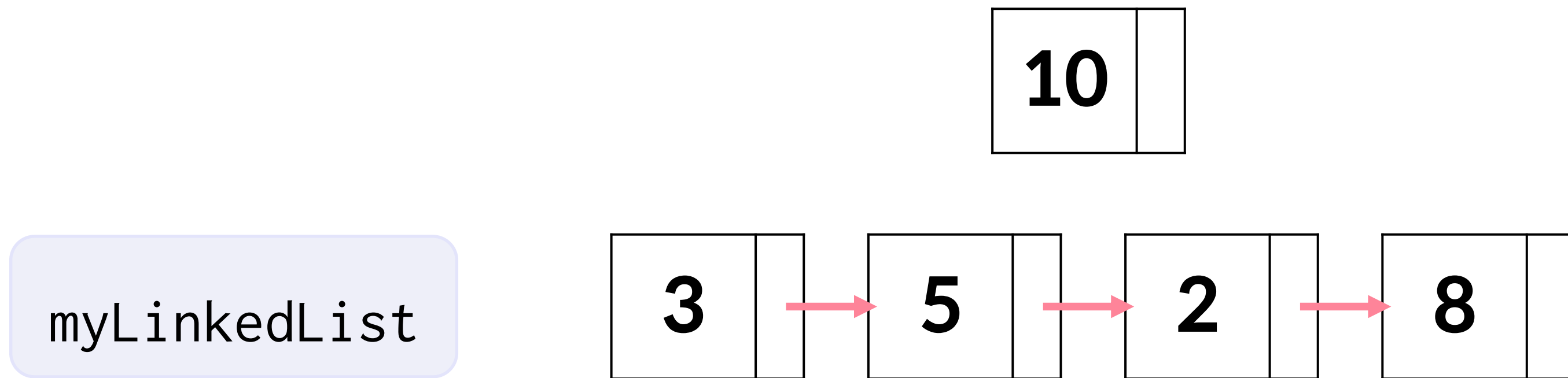


3번째 위치에 10을 추가하자

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삽입



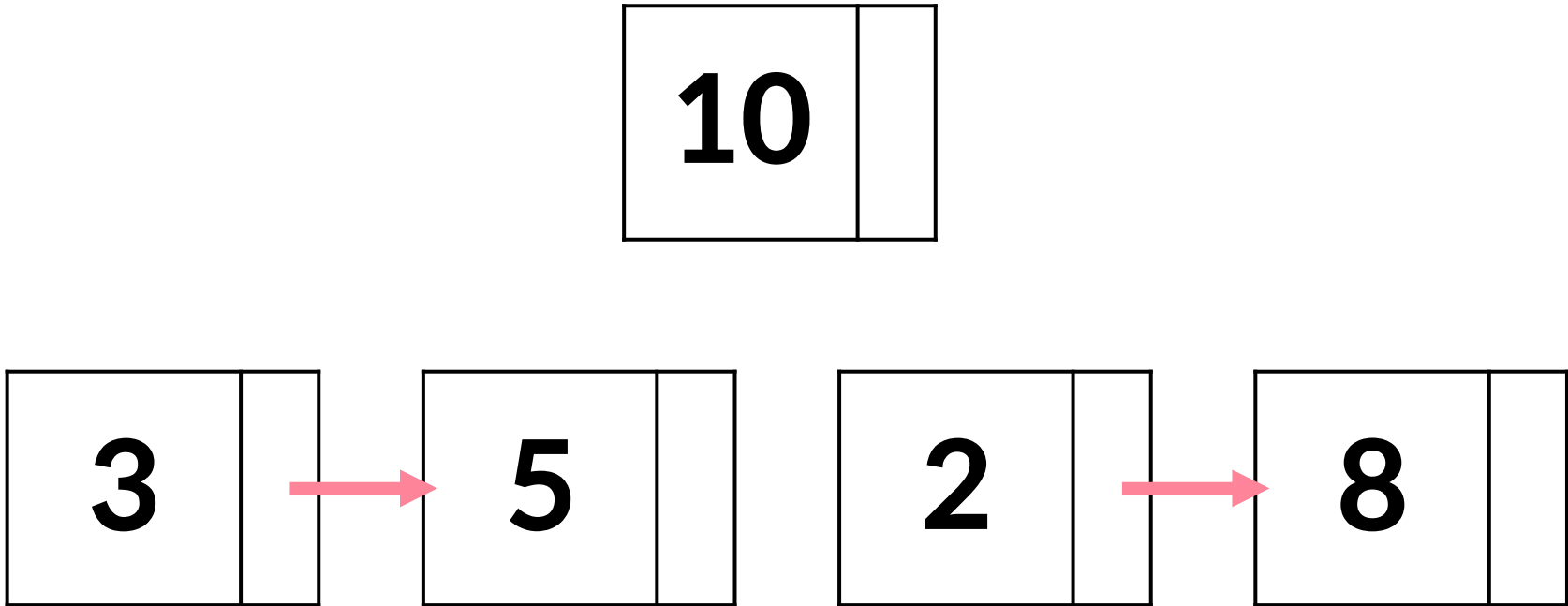
3번째 위치에 10을 추가하자

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삽입

myLinkedList



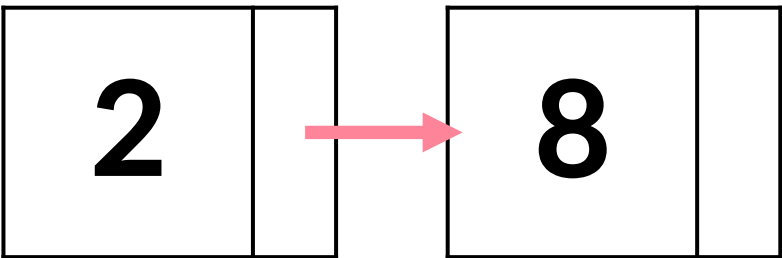
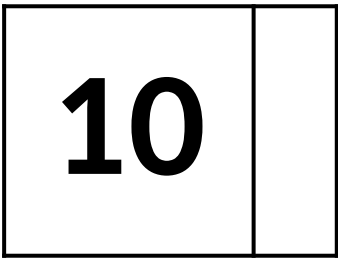
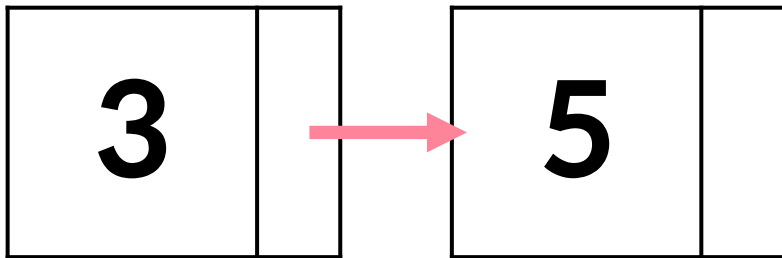
3번째 위치에 10을 추가하자

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삽입

myLinkedList



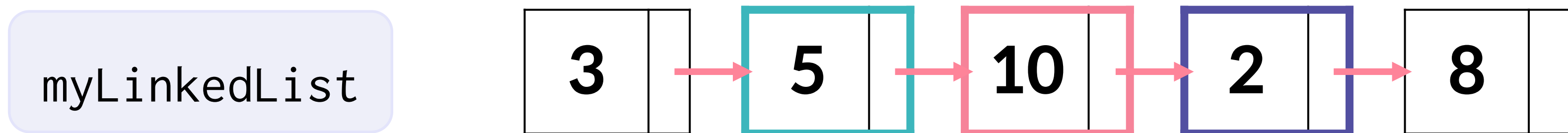
3번째 위치에 10을 추가하자

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삽입

추가할 위치의 **이전 노드**의 포인터를 **새로운 노드**로,
새로운 노드의 포인터를 **이전 노드가 가리키던 노드**로 설정한다.



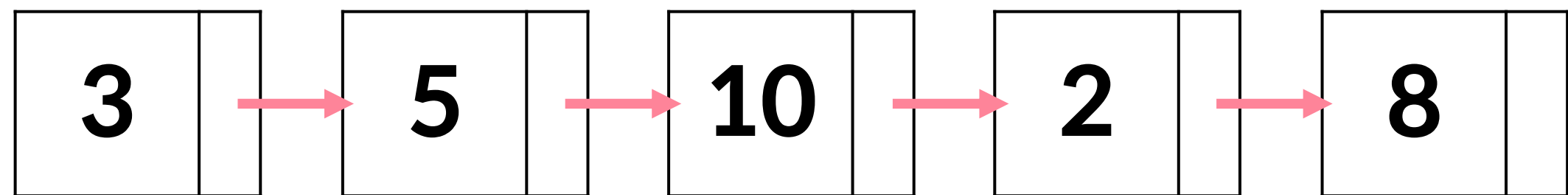
추가 완료!

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삭제

myLinkedList



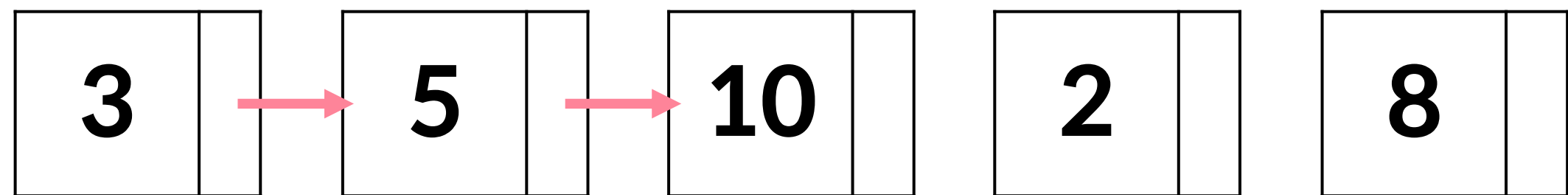
4번째 위치의 자료를 제거하자

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삭제

myLinkedList



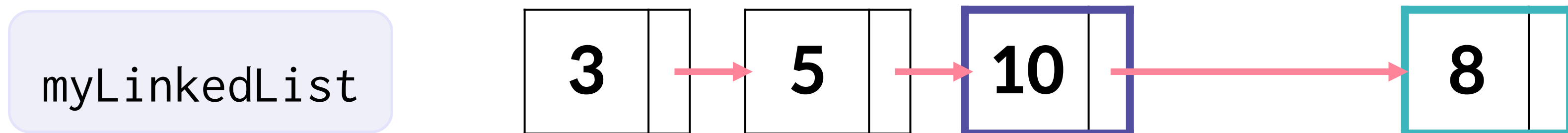
4번째 위치의 자료를 제거하자

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트 - 삭제

제거할 노드의 이전 노드의 포인터가
제거할 노드가 가리키던 노드를 가리키도록 설정한다.



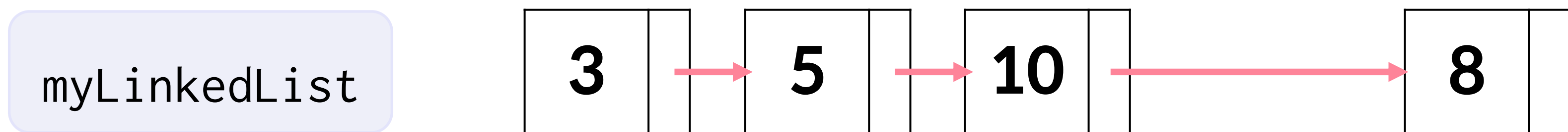
제거 완료!

`/* elice */`

03 배열과 연결 리스트

✓ 연결 리스트

인덱스를 이용하여 **절대적**인 순서를 표현하는 배열과는 달리,
연결 리스트는 자신의 다음 노드를 가리키는 **상대적**인 순서를 표현한다.



03 배열과 연결 리스트

✓ 배열 vs 연결 리스트

	배열	연결 리스트
장점	특정 위치의 자료 탐색	자료의 삽입과 삭제
단점	자료의 삽입과 삭제	특정 위치의 자료 탐색

03 배열과 연결 리스트

✓ 배열 vs 연결 리스트

리스트(추상적 자료형)

값들이 일렬로 저장되어 있으며 리스트 연산을 제공한다.

구현

배열

일렬로 저장된 값들이 인덱스라는 번호를 가진다.
특정 위치의 자료를 탐색하는 데 유리하다.

연결 리스트

일렬로 저장된 값들이 노드의 형태로 저장되어 있다.
각 노드는 자신의 다음 순서의 노드를 가리키며
자료의 삽입, 삭제에 유리하다.

04

자료구조의 구현 방법



04 자료구조의 구현 방법

✓ 자료구조의 구현 방법

자료구조는 **추상적 자료형**에 명시된 표현 및 연산 방법을 구현한다.

04 자료구조의 구현 방법

✓ 자료구조의 구현 방법

객체지향 프로그래밍에서

추상적 자료형은 인터페이스

자료구조는 클래스

로 생각할 수 있다.

04 자료구조의 구현 방법

✓ 인터페이스란?

객체지향 구조에서 **추상 메서드**만으로 이루어진 설계용 클래스

구현 부분이 비어있는 메서드를 추상 메서드라고 하며
상속받는 클래스에서 이를 구현하여 사용한다.

`/* elice */`

04 자료구조의 구현 방법

✓ 인터페이스란?

즉, '리스트'라는 인터페이스에는 "삽입과 삭제를 지원해야 한다"라는 명세만 주어지고

실제 동작 부분은 리스트를 상속받은 배열 클래스, 연결 리스트 클래스에서 구현해야 한다.

04 자료구조의 구현 방법

✓ 인터페이스란?

Example

```
class MyInterface(metaclass=ABCMeta) :    추상 클래스로 만들기 위한 메타클래스 정의(Abstract Class Meta)
    @abstractmethod ← 추상 메서드임을 나타내는 데코레이션
    def func() :
        pass
```

Java 등 다른 언어와는 달리 Python에서는 인터페이스 기능을 직접 지원하지는 않으므로
위와 같은 방식으로 표현된다.

(본 강의에서는 인터페이스를 만들지 않습니다.)

/* elice */

04 자료구조의 구현 방법

✓ 자료구조의 구현 방법

자료구조를 만드는 데에는 **클래스**가 탁월하다.

클래스가 갖고 있는 **'필드'**가 자료에 해당하고,
'메서드'가 자료에 적용할 수 있는 연산이다.

04 자료구조의 구현 방법

✓ 자료구조의 구현 예시

Example

```
import queue  
q = queue.Queue()
```

파이썬 기본 라이브러리 중, '큐'라는 자료구조를 구현한 **Queue 클래스**도 있다.
큐 자료구조의 자료 저장 및 연산 방법을 갖추고 있다.

`/* elice */`

04 자료구조의 구현 방법

✓ 자료구조 구현해보기

클래스를 이용한 첫 자료구조를 만들어보자.

`/* elice */`

04 자료구조의 구현 방법

✓ 최댓값 기계

여러분이 구현해야 하는 자료구조의 추상적 자료형은 다음과 같습니다.

`/* elice */`

04 자료구조의 구현 방법

✓ 최댓값 기계 - 자료

최댓값 기계는 **여러 정수**를 담을 수 있는 **컨테이너**를 가져야 합니다.

(컨테이너란? 리스트, 튜플, 딕셔너리 등을 비롯한 하나 이상의 요소를 담을 수 있는 것.

내부적으로 Container 클래스를 상속받은 sub class에 해당함.)

`/* elice */`

04 자료구조의 구현 방법

✓ 최댓값 기계 - 연산

그리고, 해당 컨테이너에 아래의 연산을 수행하는 메서드를 가져야 합니다.

1. 정수를 컨테이너에 **추가**
2. 정수를 컨테이너에서 **제거**
3. 컨테이너 내의 정수 중 **최댓값 반환**

`/* elice */`

04 자료구조의 구현 방법

✓ [실습1] 최댓값 기계

주어진 자료와 연산을 갖는 클래스를 구현하여
자료구조를 만들어봅시다.

`/* elice */`

05

구슬 넣기 문제 해결하기



05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 문제

양쪽이 열려있는 파이프에 구슬을 넣고 결과를 출력해보자.
왼쪽 또는 오른쪽으로 구슬을 넣을 수 있다.

입력 예시

```
3
1 0 # 왼쪽으로 1 삽입
2 1 # 오른쪽으로 2 삽입
3 0 # 왼쪽으로 3 삽입
```

출력 예시

```
3 1 2
```

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 문제

파이프를 갖는 클래스를 구현하여 자료구조를 만들려고 한다.

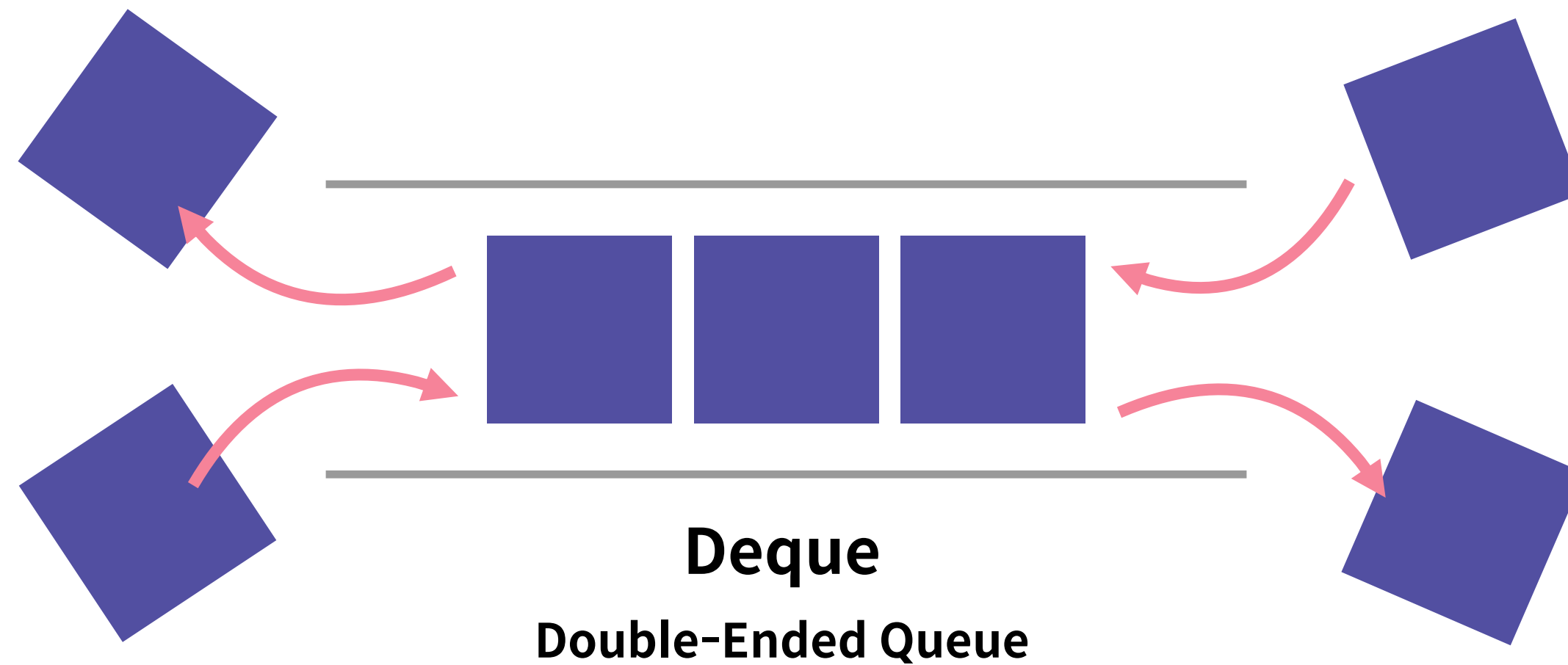
추상적 자료형은 구현 방법을 지정하지 않으므로
파이프를 **어떻게 구현하든** 상관없다.

우리는 파이프를 **배열**로 한 번, **연결 리스트**로 한 번 구현해보고자 한다.

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 문제



(2장 참조)

이 문제를 가장 잘 해결할 수 있는 '**덱**'이라는 자료구조가 있지만,
배열과 연결 리스트로 구현하여 두 자료구조의 장단점을 확인해봅시다.

05 구슬 넣기 문제 해결하기

✓ [실습2] 구슬 넣기 - 배열

명령

왼쪽으로 1 삽입

오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습2] 구슬 넣기 - 배열

0
1

명령

왼쪽으로 1 삽입

오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습2] 구슬 넣기 - 배열

0	1
1	2

명령

왼쪽으로 1 삽입

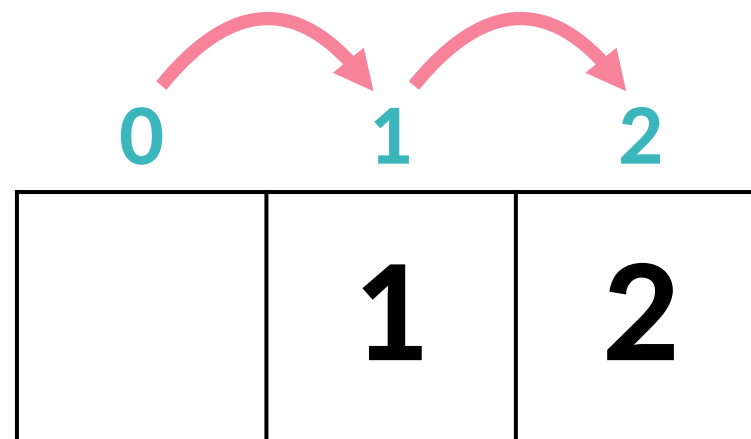
오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습2] 구슬 넣기 - 배열



명령

왼쪽으로 1 삽입

오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습2] 구슬 넣기 - 배열

0	1	2
3	1	2

명령

왼쪽으로 1 삽입

오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습2] 구슬 넣기 - 배열

배열의 특성에 의해

왼쪽으로 구슬을 삽입하는 경우

파이프 내의 모든 구슬을 한 칸씩
움겨야 하는 연산이 필요하다.

05 구슬 넣기 문제 해결하기

✓ 좋은 해법인지 생각해보기

좋은 해법인지 판단하는 기준은 여러 가지가 있다.

(코드가 간결한가?

얼마나 빠른가?

리소스를 얼마나 차지하는가?

구현 시간이 짧은가?…)

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ 좋은 해법인지 생각해보기

수행하는 명령의 수가 적을수록 시간이 덜 걸린다

Example

```
sum = 0
for i in range(30) :
    sum = sum + 1
```

Example

```
sum = 0
for i in range(300000) :
    sum = sum + 1
```

/* elice */

05 구슬 넣기 문제 해결하기

✓ 시간 복잡도

알고리즘이 문제를 해결하는 데 걸리는 시간을
정량화하여 나타낼 수 있는 방법

일반적으로, 문제에서 주어지는 **최악의 경우**에 대한
소요 시간을 나타내는 데 사용한다.

05 구슬 넣기 문제 해결하기

✓ 대략 몇 개의 명령이 수행되는가? - 배열

우리 알고리즘이 무슨 일을 하는가?

1. 숫자 하나를 왼쪽으로 삽입
2. 숫자 하나를 오른쪽으로 삽입

05 구슬 넣기 문제 해결하기

✓ 대략 몇 개의 명령이 수행되는가? - 배열

시간 복잡도는 일반적으로 **최악**의 경우를 고려해야 한다.

구슬을 **왼쪽**으로 삽입하면,
배열 내에서 한 칸씩 이동해야 하므로 느리다.

그렇다면, 최악의 경우의 연산 횟수는?

05 구슬 넣기 문제 해결하기

✓ 대략 몇 개의 명령이 수행되는가? - 배열

구슬 n개를 모두 **왼쪽**으로만 삽입

구슬이 3개라면?

$$1 + 2 + 3 = 6$$

구슬이 5개라면?

$$1 + 2 + 3 + 4 + 5 = 15$$

구슬이 n개라면?

$$\frac{n(n+1)}{2}$$

05 구슬 넣기 문제 해결하기

✓ 대략 몇 개의 명령이 수행되는가? - 배열

즉, 이 문제를 배열로 해결하였을 때의 시간 복잡도는

$$O\left(\frac{n(n+1)}{2}\right)$$

$O()$ 는 최악 조건의 시간 복잡도를 의미하는 기호로,
Big-O Notation이라고 합니다.

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트

명령

왼쪽으로 1 삽입

오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트

1	
---	--

명령

- 왼쪽으로 1 삽입
- 오른쪽으로 2 삽입
- 왼쪽으로 3 삽입

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트

명령

1	
---	--

왼쪽으로 1 삽입

오른쪽으로 2 삽입

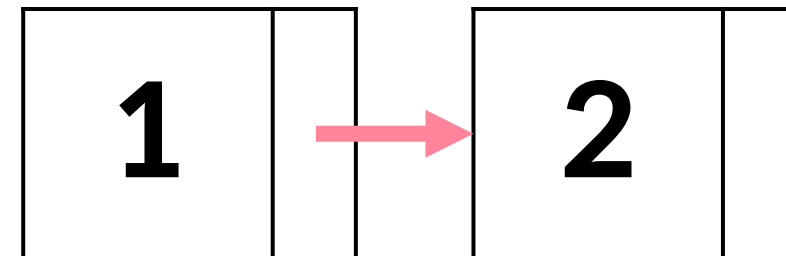
왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트

명령



왼쪽으로 1 삽입

오른쪽으로 2 삽입

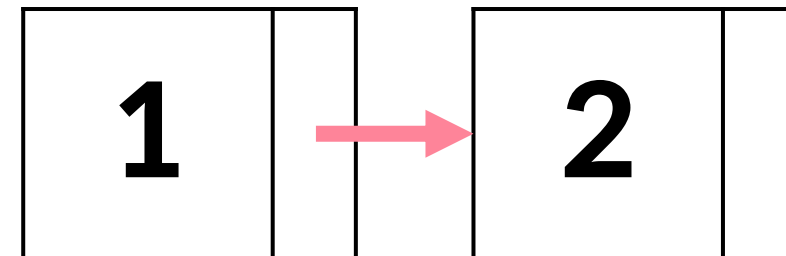
왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트

명령



왼쪽으로 1 삽입

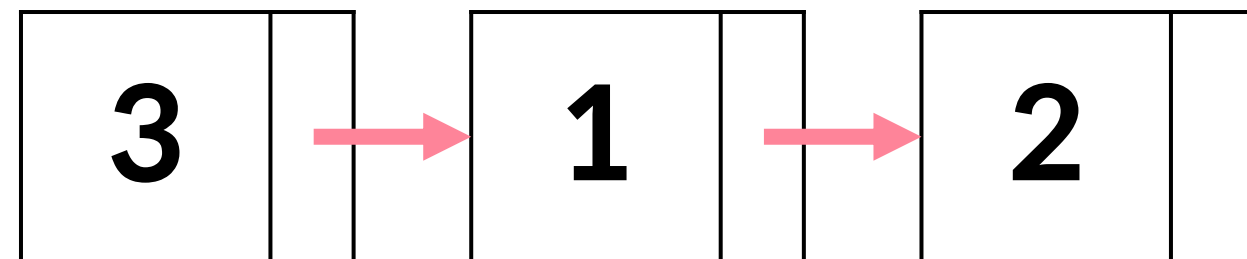
오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트



명령

왼쪽으로 1 삽입

오른쪽으로 2 삽입

왼쪽으로 3 삽입

`/* elice */`

05 구슬 넣기 문제 해결하기

✓ [실습3] 구슬 넣기 - 연결 리스트

연결 리스트는 구슬을 어디로 넣든 **한 번의 연산**으로 수행한다.

05 구슬 넣기 문제 해결하기

✓ 대략 몇 개의 명령이 수행되는가? - 연결 리스트

구슬 n개를 모두 **왼쪽**으로만 삽입

구슬이 3개라면?

$$1 + 1 + 1 = 3$$

구슬이 5개라면?

$$1 + 1 + 1 + 1 + 1 = 5$$

구슬이 n개라면?

$$n$$

05 구슬 넣기 문제 해결하기

✓ 대략 몇 개의 명령이 수행되는가? - 연결 리스트

즉, 이 문제를 **연결 리스트**로 해결하였을 때의 시간 복잡도는

$$O(n)$$

05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 - 정리

연결 리스트로 구현한 것이
배열로 구현한 것보다 빠르다.

이 문제에서는 연결 리스트가 더 좋다.

05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 - 정리

자료구조를 공부해야 하는 이유?

작성하고자 하는 프로그램의 **목적**을
가장 **효율적**으로 달성할 수 있는 **자료구조**를 사용하기 위해서

05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 - 정리

각 자료구조의 **특성**과
동작 방법을 제대로 이해하고 있어야 한다.

05 구슬 넣기 문제 해결하기

✓ 구슬 넣기 - 정리

1. 문제를 파악한다.

(= 어떤 자료를 담을지, 자료에 어떤 의미가 있는지 파악)

2. 자료구조에 **필요한 기능**을 파악한다.

(= 자료를 어떻게 사용하는지 파악)

3. 문제를 **효율적**으로 해결하는 자료구조를 설계 및 사용한다.

(= 목적에 맞는 자료구조로 문제를 해결)

06

주문 관리 시스템 문제 해결하기



06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

주문 **생성**, 주문 **제거**, 주문 **조회**의 기능을 가진
주문 관리 시스템을 구현해야 한다.

입력 예시

```
5
1 1 # 1번 주문 생성
1 2 # 2번 주문 생성
3 2 # 2번 주문이 몇 번째인지 조회
2 1 # 1번 주문 제거
3 2 # 2번 주문이 몇 번째인지 조회
```

출력 예시

```
2
1
```

/* elice */

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

`orderManager` 클래스가 기본으로 주어지고,

주문 생성, 주문 제거, 주문 조회를 각각

`addOrder`, `removeOrder`, `getOrder` 함수를 구현하면 된다.

`/* elice */`

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

배열

연결 리스트

addOrder

`myList.append(n)`

끝에 하나 추가

removeOrder

`myList.remove(n)`

따라가면서 찾은 후 삭제

getOrder

몇 번째인지 반환

몇 번째인지 반환

`/* elice */`

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 성능 확인

배열

```
--- 주문 조회가 매우 많을 경우 테스트 ---  
Testcase 11: accept (5 points, 461.355 ms),  
Testcase 12: accept (5 points, 606.895 ms),  
Testcase 13: accept (5 points, 760.886 ms),  
Testcase 14: accept (5 points, 938.456 ms),  
Testcase 15: accept (5 points, 1136.838 ms),  
--- 주문 조회가 별로 없을 경우 테스트 ---  
Testcase 16: accept (5 points, 843.022 ms),  
Testcase 17: accept (5 points, 1874.939 ms),  
Testcase 18: accept (5 points, 3265.385 ms),  
Testcase 19: accept (5 points, 5156.427 ms),  
Testcase 20: accept (5 points, 7444.110 ms),
```

연결 리스트

```
--- 주문 조회가 매우 많을 경우 테스트 ---  
Testcase 11: accept (5 points, 2040.262 ms),  
Testcase 12: accept (5 points, 2806.707 ms),  
Testcase 13: accept (5 points, 3505.856 ms),  
Testcase 14: accept (5 points, 4563.550 ms),  
Testcase 15: accept (5 points, 5342.908 ms),  
--- 주문 조회가 별로 없을 경우 테스트 ---  
Testcase 16: accept (5 points, 11956.102 ms),  
Testcase 17: accept (5 points, 26132.382 ms),  
Testcase 18: accept (5 points, 47280.964 ms),  
Testcase 19: accept (5 points, 73159.718 ms),  
Testcase 20: accept (5 points, 106797.463 ms),
```

/* elice */

06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 성능 확인

연결 리스트로 구현하였을 때
처리 속도가 **너무 느린 것**을 알 수 있다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

배열

연결 리스트

addOrder

myList.append(n)

끝에 하나 추가

removeOrder

myList.remove(n)

따라가면서 찾은 후 삭제

getOrder

몇 번째인지 반환

몇 번째인지 반환

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

	배열	연결 리스트
addOrder	$O(1)$	$O(1)$
removeOrder	$O(n)$	$O(n)$
getOrder	$O(n)$	$O(n)$

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 연결 리스트

연결 리스트의 **특정 노드**를 삭제하기 위해서
그 **특정 노드**에 접근하는 과정이 필요하다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 연결 리스트

연결 리스트의 특성에 의해
특정 원소에 접근하기 위해서는
시작 원소부터 하나씩 따라가야 한다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 연결 리스트

연결 리스트는 어떤 노드를 삭제하기 위해서

그 노드의 이전 노드와 다음 노드가 무엇인지 알고 있어야 하기 때문이다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 딕셔너리 활용

이 단점을 개선하기 위해
연결 리스트 내에 **딕셔너리**를 두고,
모든 노드들의 정보를 저장하고 있도록 한다.

/* elice */

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 딕셔너리 활용

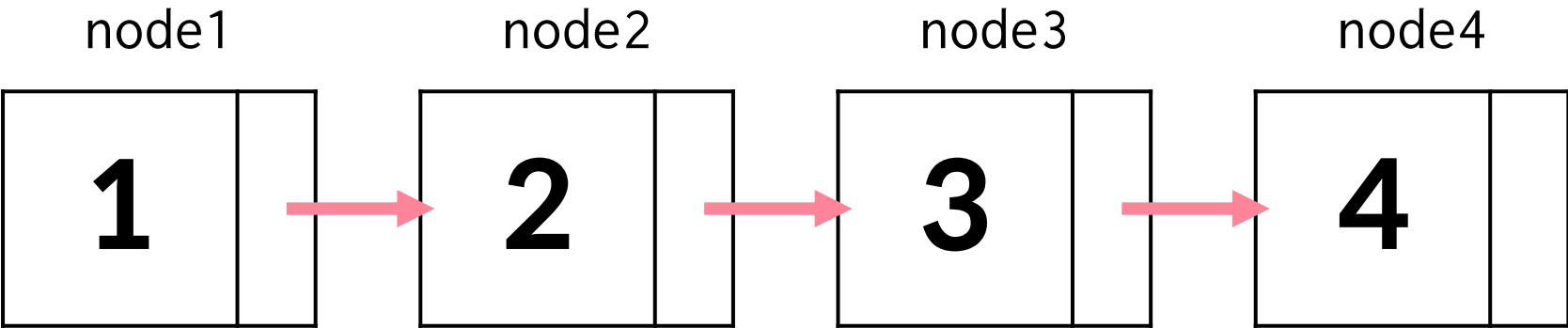
딕셔너리는 내부적으로 **해시 테이블**이라는 자료구조로 동작하며
어떠한 key에 대한 value를 $O(1)$ 의 시간 복잡도로 접근할 수 있다.

(단, 충돌이 일어나지 않았을 때를 전제로 한다. 자세한 내용은 144 페이지 부록 참고)

06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 딕셔너리 활용

myLinkedList



myDict

Example

```
{
    1 : node1,
    2 : node2,
    3 : node3,
    4 : node4
}
```

/* elice */

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 딕셔너리 활용

주문번호에 대해 각 노드를 대응시키는 **딕셔너리**를 하나 만들어두면

`removeOrder` 함수를 수행할 때

삭제할 노드를 **$O(1)$** 만에 찾아낼 수 있다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

삭제할 노드를 알아냈다고 해서 끝이 아니다.

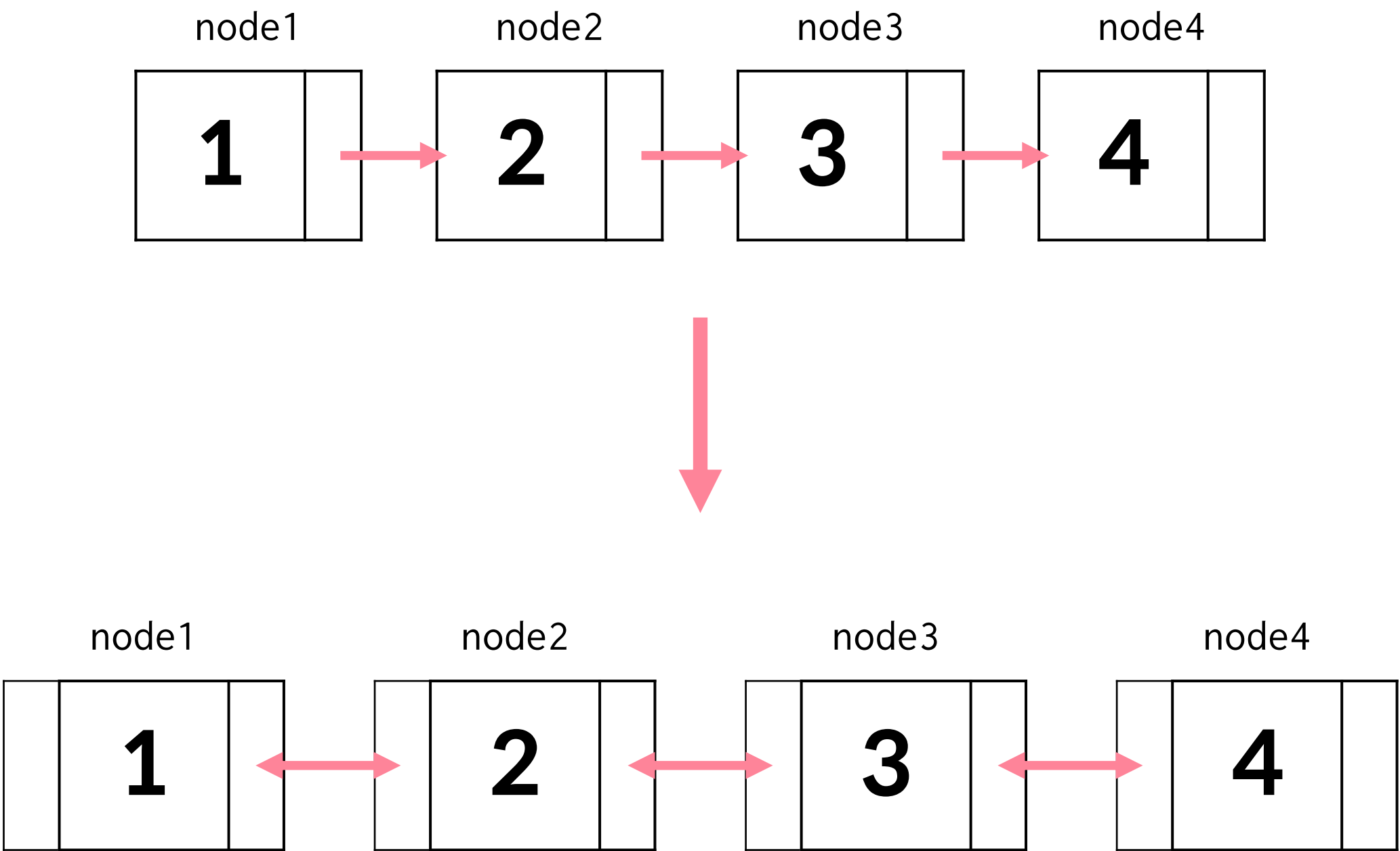
삭제할 노드의 다음 노드와 이전 노드를 알아야 하는데,

현재로서는 이전 노드에 접근할 방법이 없다.

/* elice */

06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 이중 연결 리스트 활용



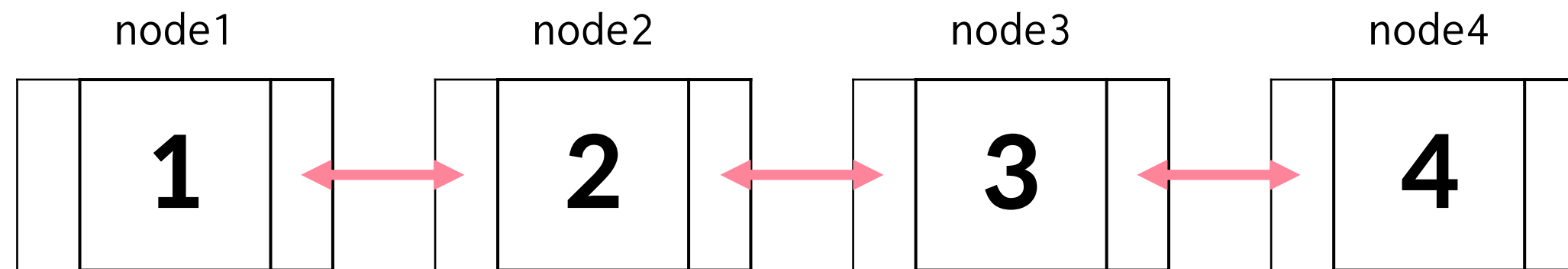
06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 이중 연결 리스트 활용

노드에 포인터가 **두 개**가 있고
각각 **이전 노드**와 **다음 노드**를 가리키는 형태의
연결 리스트를 **이중 연결 리스트**라고 한다.

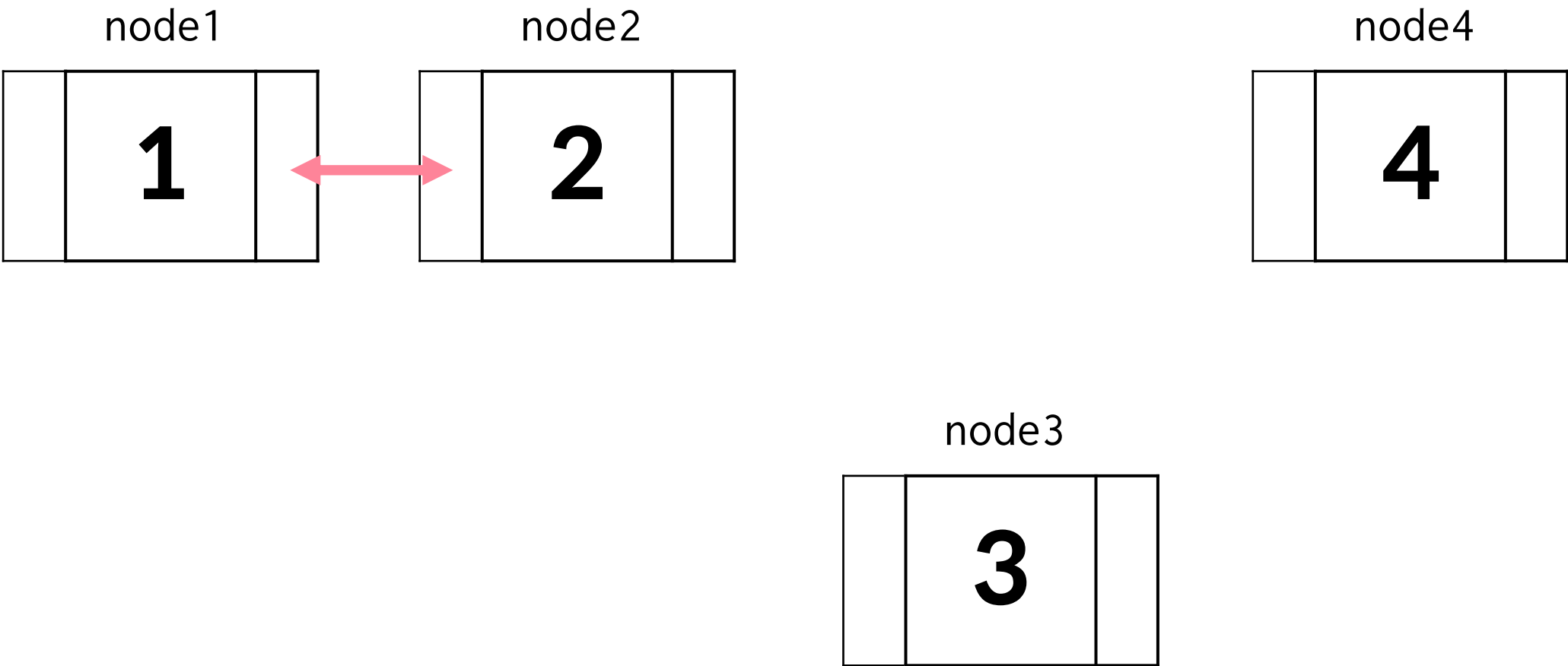
06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 이중 연결 리스트 활용



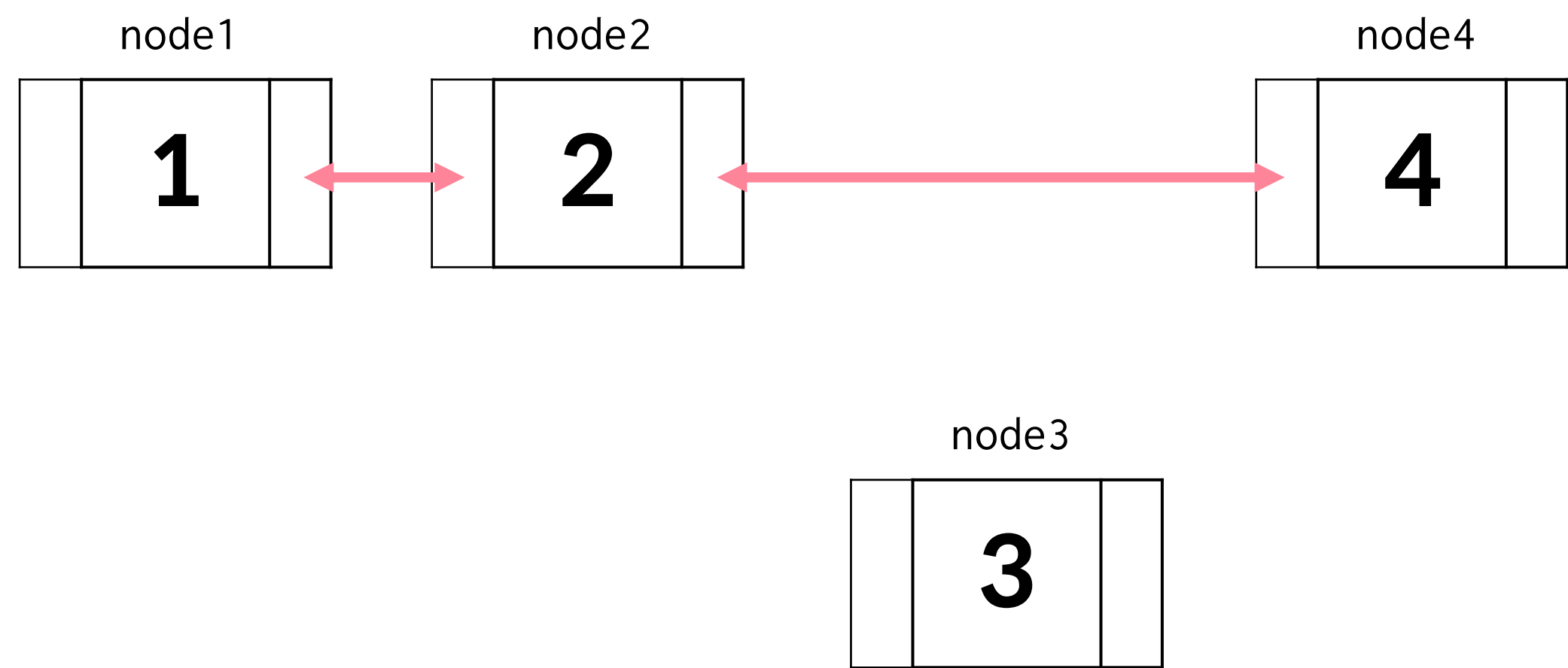
06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 이중 연결 리스트 활용



06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 이중 연결 리스트 활용



06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 이중 연결 리스트 활용

이중 연결 리스트를 이용하여
노드를 간편하게 삭제할 수 있다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 성능 확인

```
--- 주문 조회가 매우 많을 경우 테스트 ---  
Testcase 11: accept (5 points, 699.858 ms),  
Testcase 12: accept (5 points, 909.207 ms),  
Testcase 13: accept (5 points, 1193.801 ms),  
Testcase 14: accept (5 points, 1488.627 ms),  
Testcase 15: accept (5 points, 1805.969 ms),  
--- 주문 조회가 별로 없을 경우 테스트 ---  
Testcase 16: accept (5 points, 157.588 ms),  
Testcase 17: accept (5 points, 271.329 ms),  
Testcase 18: accept (5 points, 408.856 ms),  
Testcase 19: accept (5 points, 620.997 ms),  
Testcase 20: accept (5 points, 881.348 ms),
```

/* elice */

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 주문 조회

배열이든 연결 리스트든,
주문 번호가 주어졌을 때 해당 주문이 몇 번째인지 알기 위해서는
맨 첫 번째 주문부터 하나씩 확인해봐야 한다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템 - 주문 조회

두 자료구조 모두 비슷한 방식으로 주문 조회를 수행하는데 왜 배열이 더 빠를까?

배열

```
--- 주문 조회가 매우 많을 경우 테스트 ---  
Testcase 11: accept (5 points, 461.355 ms),  
Testcase 12: accept (5 points, 606.895 ms),  
Testcase 13: accept (5 points, 760.886 ms),  
Testcase 14: accept (5 points, 938.456 ms),  
Testcase 15: accept (5 points, 1136.838 ms),  
--- 주문 조회가 별로 없을 경우 테스트 ---  
Testcase 16: accept (5 points, 843.022 ms),  
Testcase 17: accept (5 points, 1874.939 ms),  
Testcase 18: accept (5 points, 3265.385 ms),  
Testcase 19: accept (5 points, 5156.427 ms),  
Testcase 20: accept (5 points, 7444.110 ms),
```

연결 리스트

```
--- 주문 조회가 매우 많을 경우 테스트 ---  
Testcase 11: accept (5 points, 699.858 ms),  
Testcase 12: accept (5 points, 909.207 ms),  
Testcase 13: accept (5 points, 1193.801 ms),  
Testcase 14: accept (5 points, 1488.627 ms),  
Testcase 15: accept (5 points, 1805.969 ms),  
--- 주문 조회가 별로 없을 경우 테스트 ---  
Testcase 16: accept (5 points, 157.588 ms),  
Testcase 17: accept (5 points, 271.329 ms),  
Testcase 18: accept (5 points, 408.856 ms),  
Testcase 19: accept (5 points, 620.997 ms),  
Testcase 20: accept (5 points, 881.348 ms),
```

/* elice */

06 주문 관리 시스템 문제 해결하기

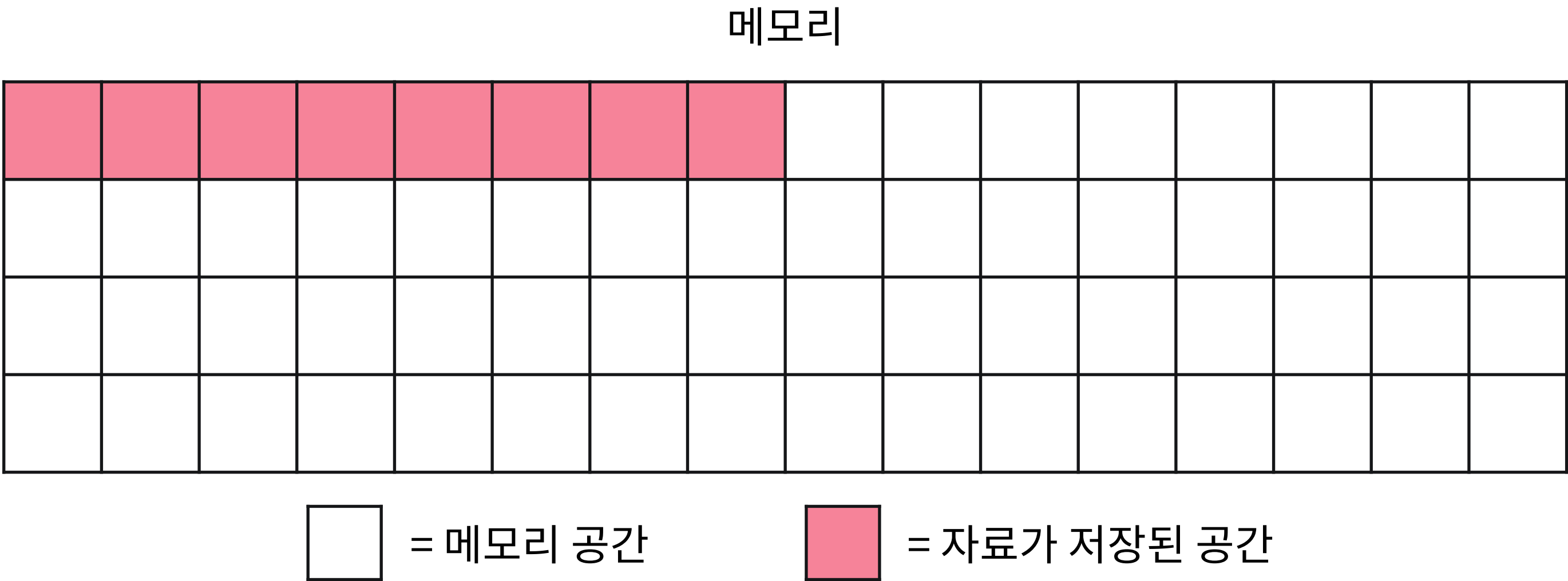
✓ 주문 관리 시스템 - 주문 조회

배열의 요소들은 컴퓨터에서 **물리적**으로 가깝기 때문이다.

연결 리스트는 **각각 별도의 객체**들을 임의로 연결하는 방식으로 구현되지만
배열의 경우, 각 요소들이 컴퓨터 내부에서 **매우 가깝게 위치**하고 있다.

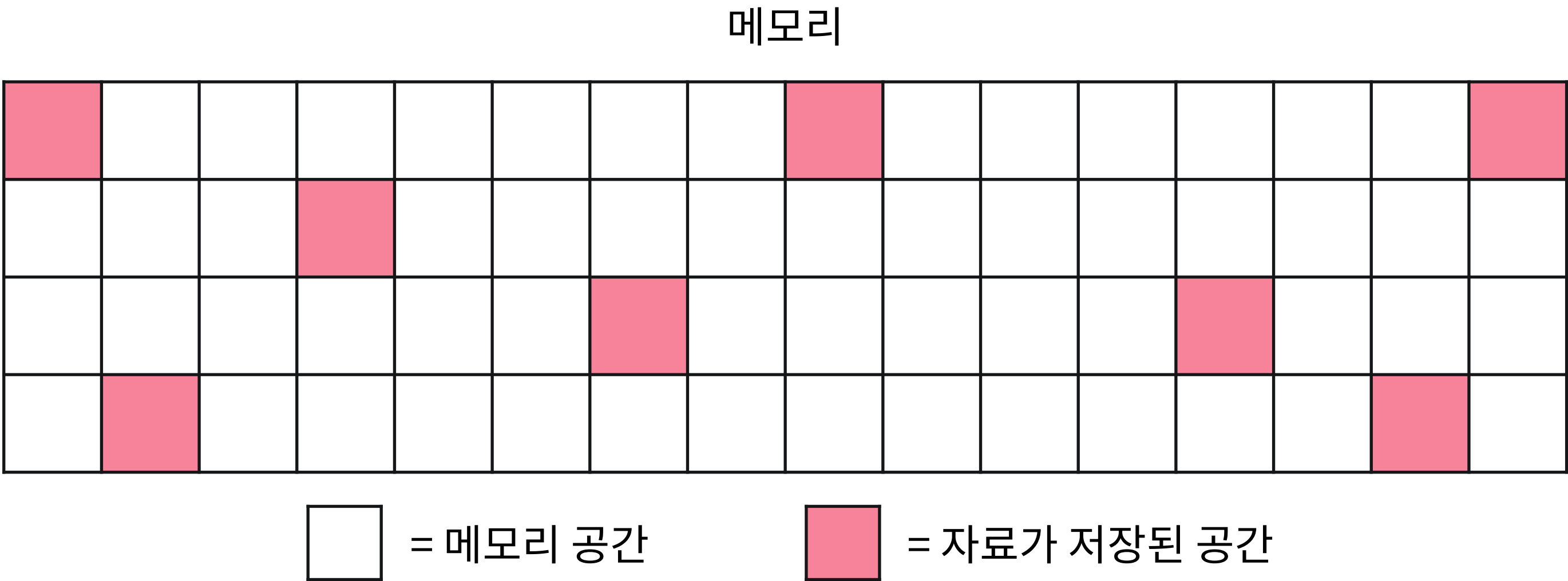
06 주문 관리 시스템 문제 해결하기

✔ 자료를 메모리에 저장하는 방법 - 배열



06 주문 관리 시스템 문제 해결하기

✔ 자료를 메모리에 저장하는 방법 - 연결 리스트



06 주문 관리 시스템 문제 해결하기

✔ 주문 관리 시스템 - 주문 조회

따라서 전체 요소를 **순회**하는 연산 또한 배열이 더 유리하다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

주문 관리 시스템이 처리하는 주문의 양이 같더라도,

주문 조회가 많을 때, 적을 때 유리한 자료구조가 각각 있다.

06 주문 관리 시스템 문제 해결하기

✓ 주문 관리 시스템

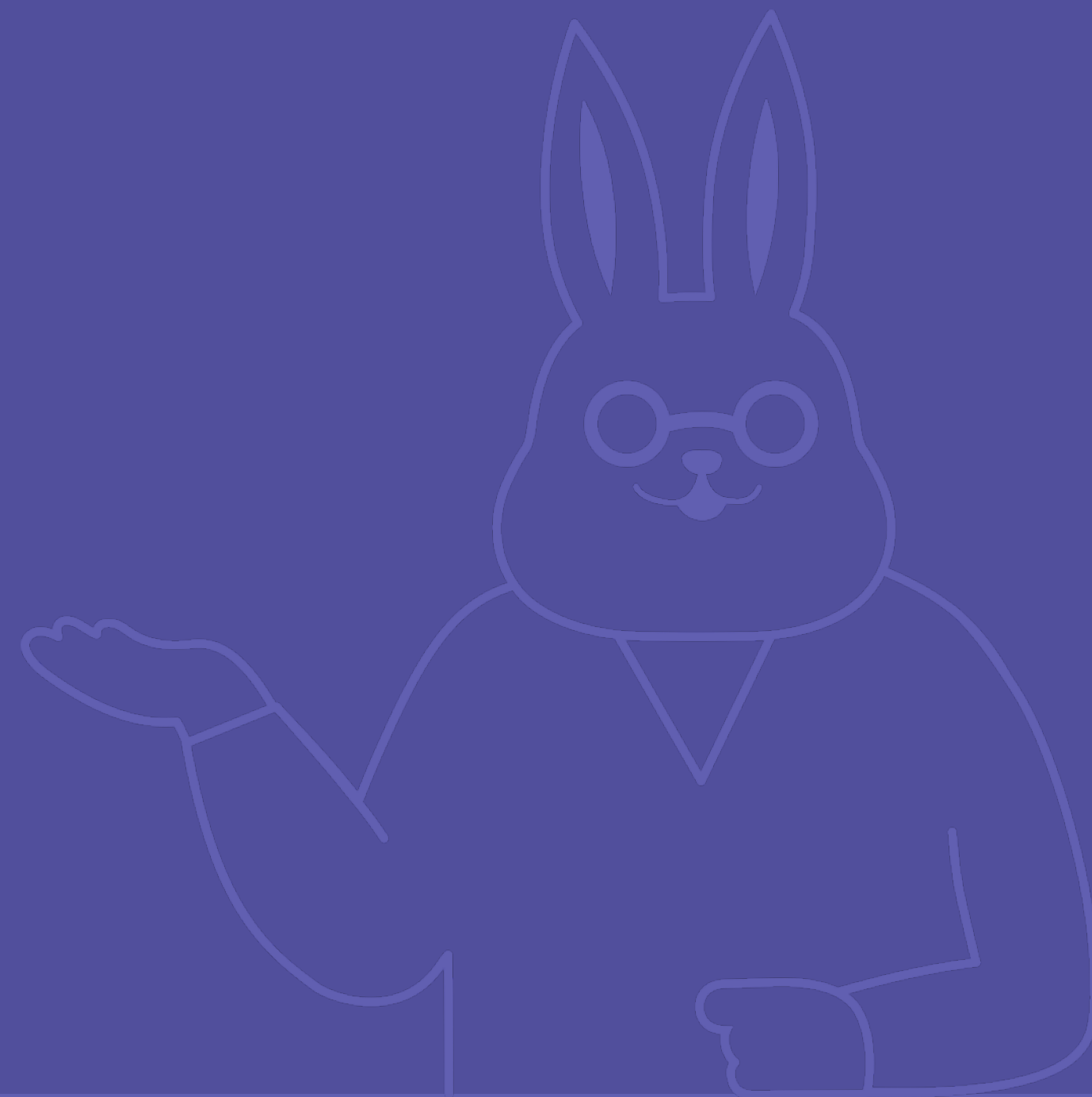
알고리즘이 같더라도 데이터의 처리에 따라 성능이 다르다.

따라서 **데이터**의 입출력과 **문제 상황**을 잘 파악하여

적절한 자료구조를 선택해야 한다.

07

마무리 및 부록



✓ 마무리

자료구조란?

자료를 저장하는 구조

자료구조를 공부하는 이유?

자료를 잘 저장하기 위해서

✓ 마무리

'자료를 잘 저장한다'는 말의 의미

구현하고자 하는 프로그램의 **의도**에 맞는 자료구조를 사용하여
효율적인 성능으로 구현한 경우

✓ 마무리

추상적 자료형이란?

자료를 담는 방법과, 자료에 대한 연산에 대한 정의가 담긴 것
구현 방법은 정의되어 있지 않다.

07 마무리 및 부록

✓ 마무리

추상적 자료형에서의 자료 저장 방법과, 연산 방법을 자세히 구현한 것



자료구조

07 마무리 및 부록

✓ 마무리

리스트라는 **추상적 자료형**을 구현한 대표적인 두 가지 자료구조는?

배열, 연결 리스트

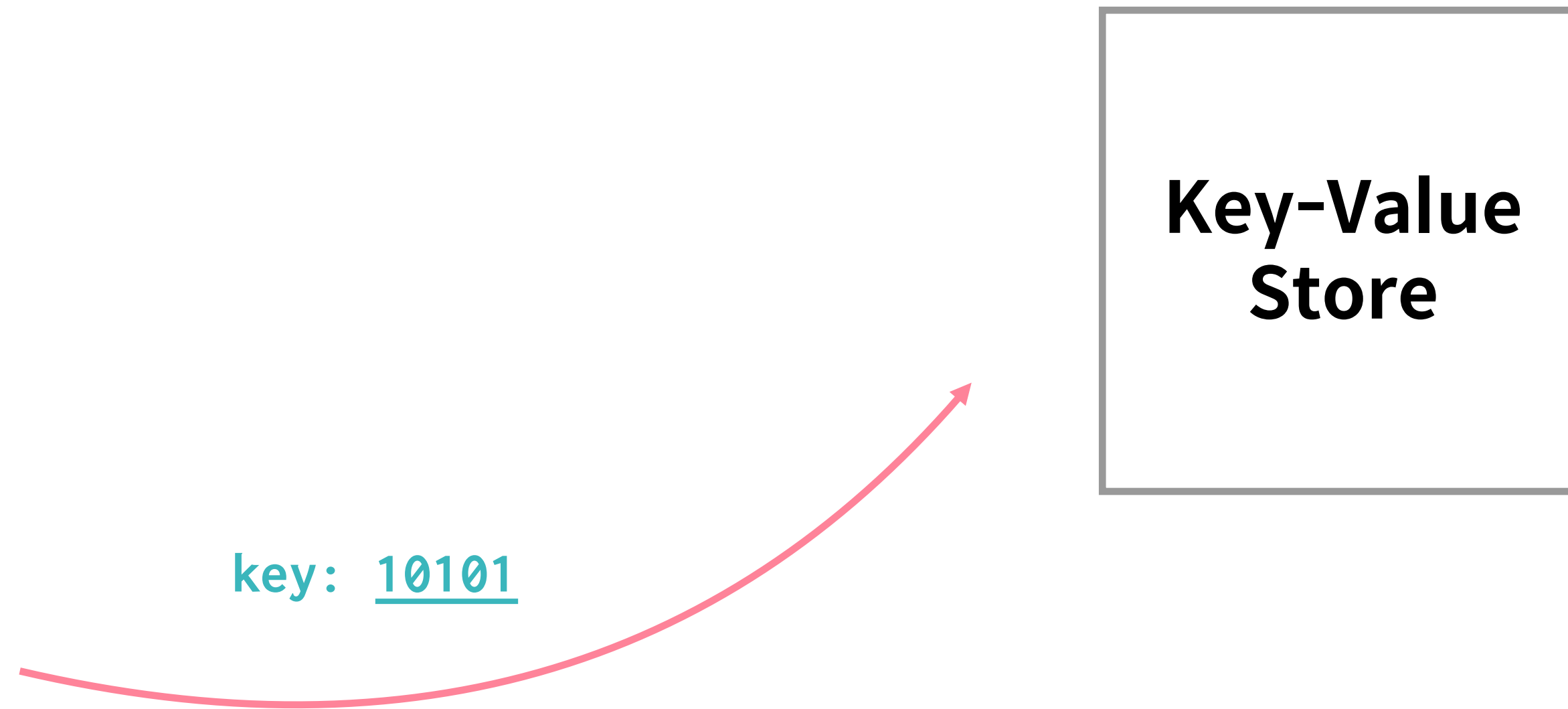
✔ 마무리

	배열	연결 리스트
장점	특정 위치의 자료 탐색	자료의 삽입과 삭제
단점	자료의 삽입과 삭제	특정 위치의 자료 탐색

07 마무리 및 부록

✓ 부록 - 해시 테이블

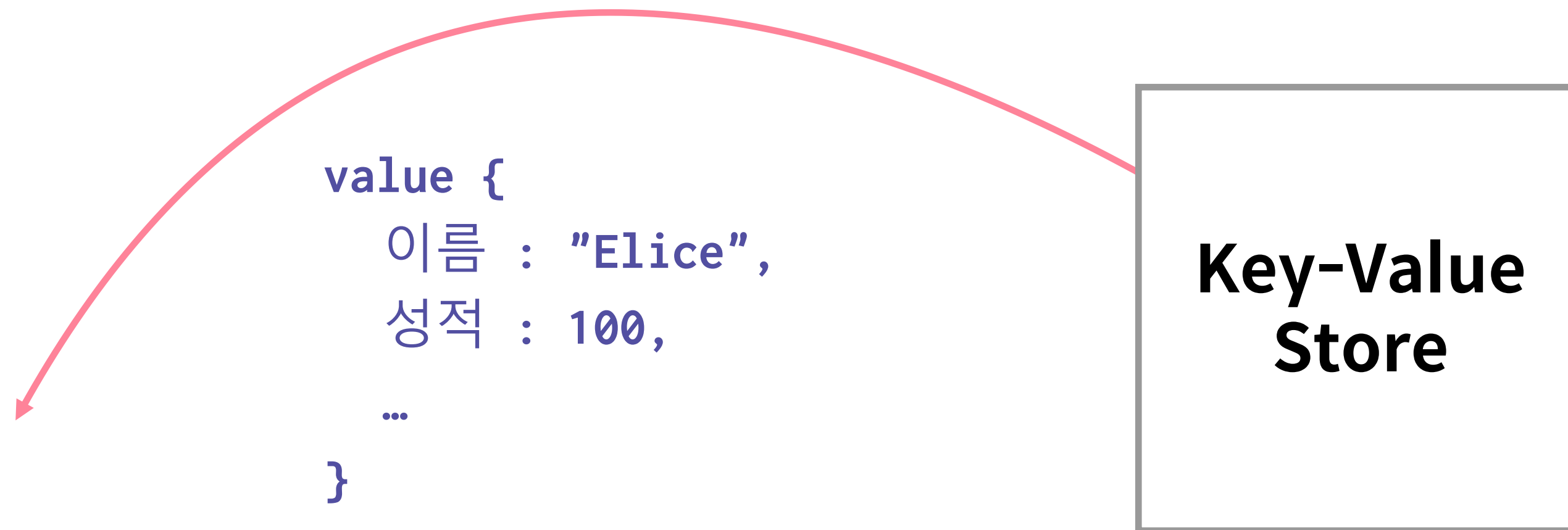
각 데이터(value)를 고유한 **key**에 대응하도록 저장하는 **개념**



07 마무리 및 부록

✓ 부록 - 해시 테이블

각 데이터(value)를 고유한 **key**에 대응하도록 저장하는 **개념**



`/* elice */`

07 마무리 및 부록

✓ 부록 - 해시 테이블

key가 정수이고, **value** 역시 정수인
Key-Value Store를 구현해보자.

Key-Value 쌍을 **입력**하는 연산은 **put**,
특정 Key의 Value를 **조회**하는 연산은 **get**이라고 정의한다.

`/* elice */`

07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

배열에 **value**를 저장하고,
배열의 **인덱스**를 **key**로 이용하는 방식으로 구현해보자.

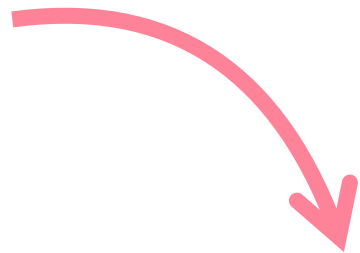
`/* elice */`

07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

`db.put(2, 7)`

`db = kvStore()`



	0	1	2	3	4	5	6
myValue	3				6		

`/* elice */`

07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

db = kvStore()

	0	1	2	3	4	5	6
myValue	3		7		6		

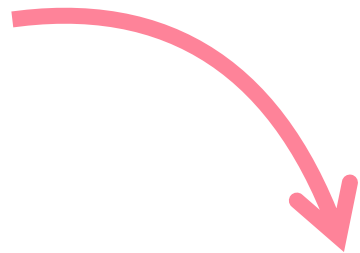
/* elice */

07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

`db.get(4)`

`db = kvStore()`



	0	1	2	3	4	5	6
myValue	3		7		6		

`/* elice */`

07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

db = kvStore()

6

myValue

0 1 2 3 4 5 6

3

7

6

/* elice */

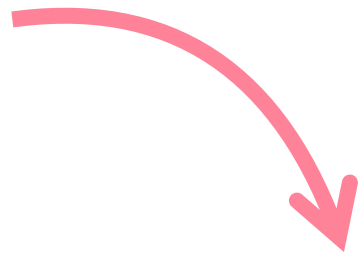
07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

문제점 : 배열의 **크기**를 벗어난 인덱스에 **추가**해야 하는 경우는?

`db.put(100, 2)`

`db = kvStore()`



	0	1	2	3	4	5	6
myValue	3		7		6		

`/* elice */`

07 마무리 및 부록

✓ 배열의 index를 key로 이용하기

문제점 : **value가 없는 인덱스**에 get을 하는 경우는?

`db.get(3)`

`db = kvStore()`

???

	0	1	2	3	4	5	6
myValue	3		7		6		

`/* elice */`

✓ 배열의 index를 key로 이용하기

장점

(운 좋으면) 자료의 **쓰기** 연산이 빠르다.
자료의 **읽기** 연산이 빠르다.

단점

“**자료가 없다**”를 표현하는 것이 쉽지 않다.
공간이 지나치게 낭비될 수 있다.

07 마무리 및 부록

✓ key와 value를 각각 배열에 저장하기

key를 저장하는 배열과 **value**를 저장하는 배열,
두 개의 배열을 사용하여 저장해보자.

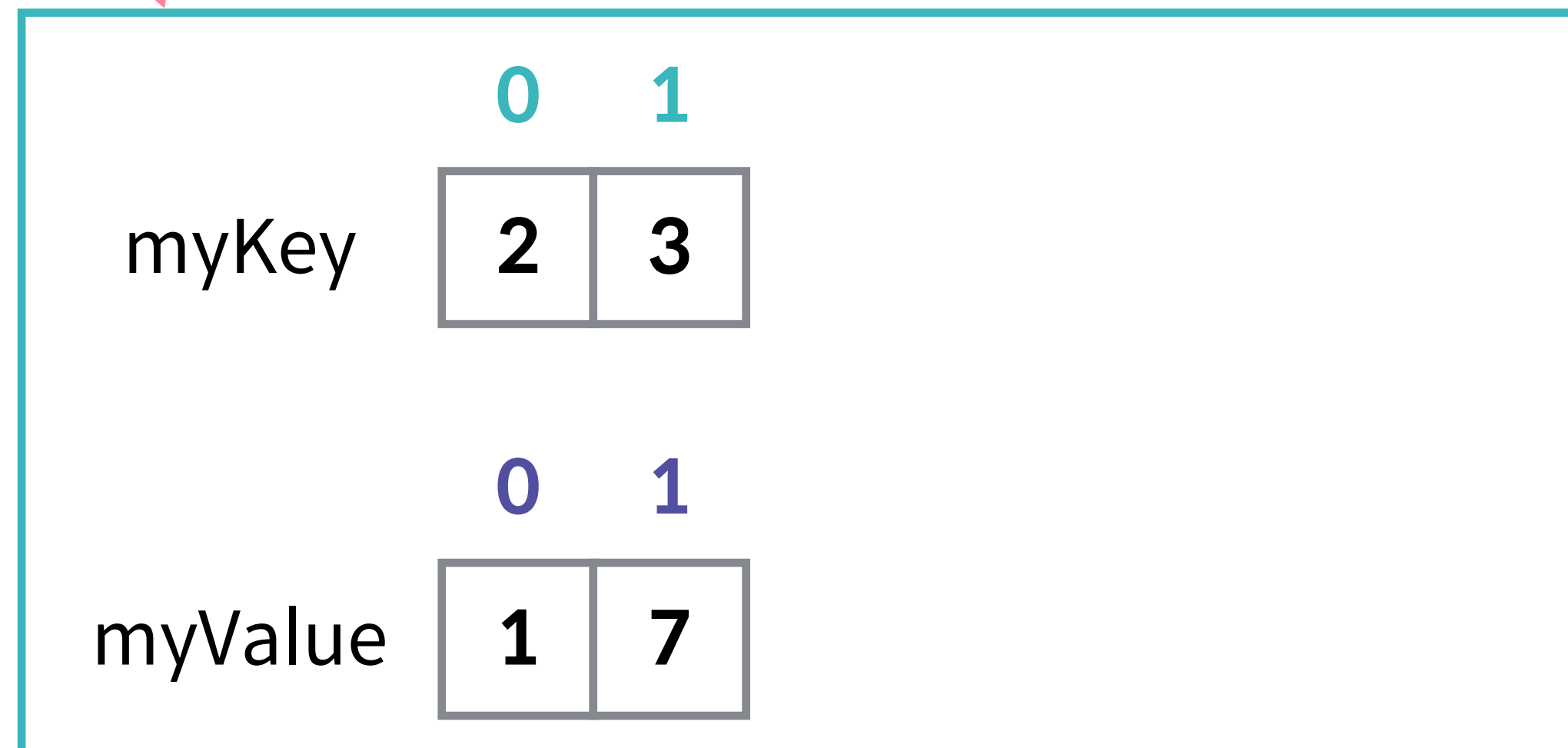
`/* elice */`

07 마무리 및 부록

- ✓ key와 value를 각각 배열에 저장하기

`db.put(4, 8)`

`db = kvStore()`



`/* elice */`

07 마무리 및 부록

✓ key와 value를 각각 배열에 저장하기

db = kvStore()

	0	1	2
myKey	2	3	4
	0	1	2
myValue	1	7	8

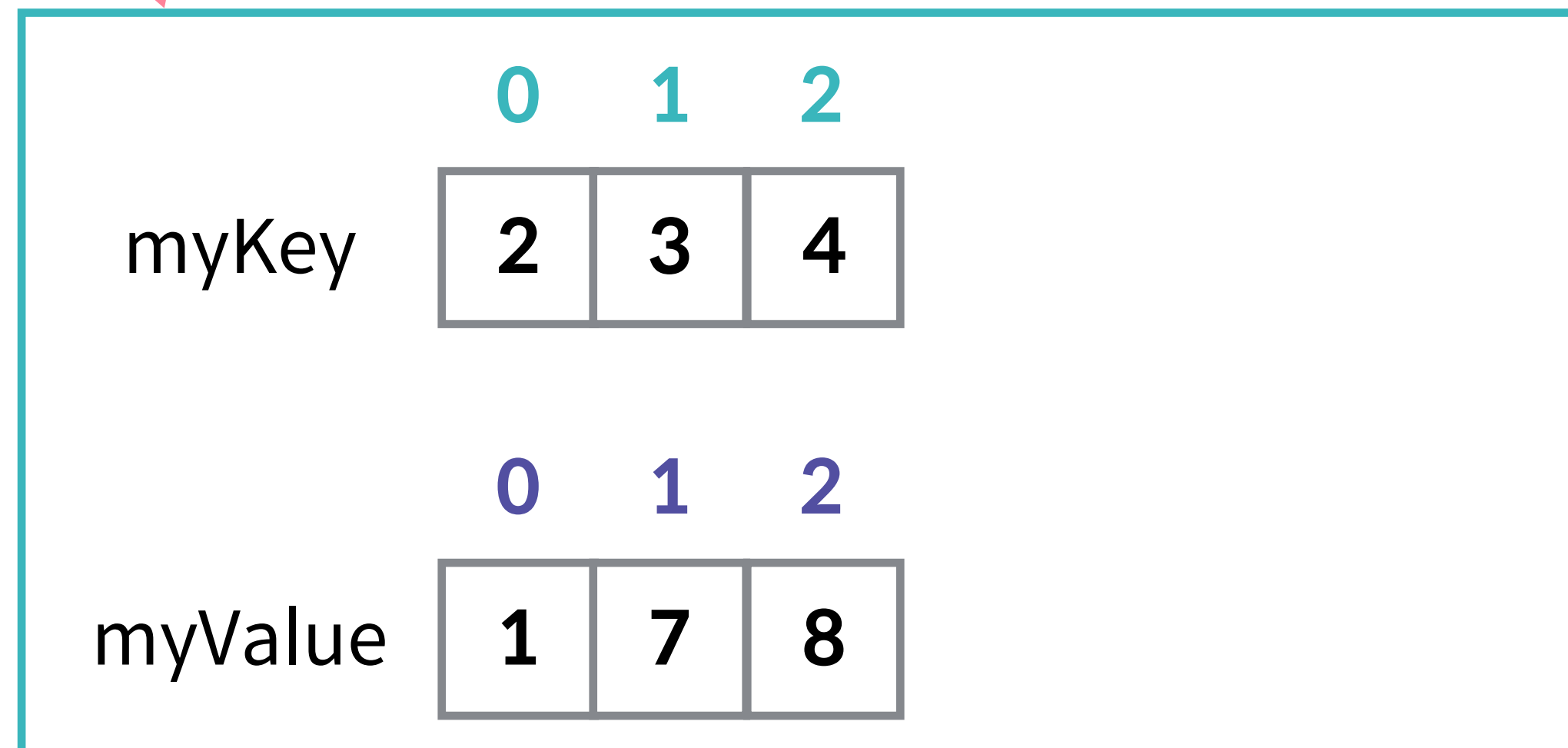
/* elice */

07 마무리 및 부록

✓ key와 value를 각각 배열에 저장하기

`db.put(100, 2)`

`db = kvStore()`



`/* elice */`

07 마무리 및 부록

✓ key와 value를 각각 배열에 저장하기

db = kvStore()

	0	1	2	3
myKey	2	3	4	100
	0	1	2	3
myValue	1	7	8	2

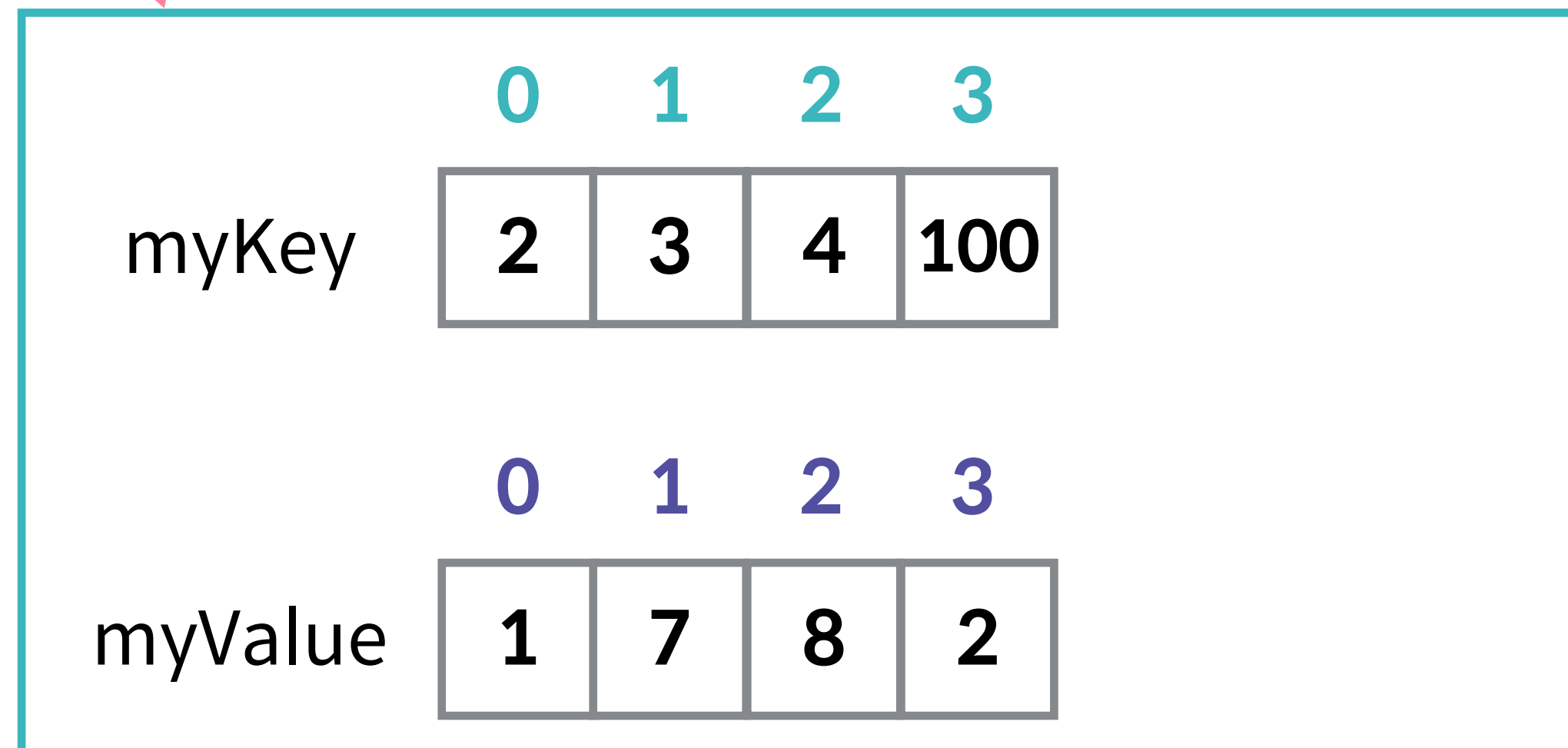
/* elice */

07 마무리 및 부록

- ✓ key와 value를 각각 배열에 저장하기

`db.get(4)`

`db = kvStore()`



	0	1	2	3
myKey	2	3	4	100
	0	1	2	3
myValue	1	7	8	2

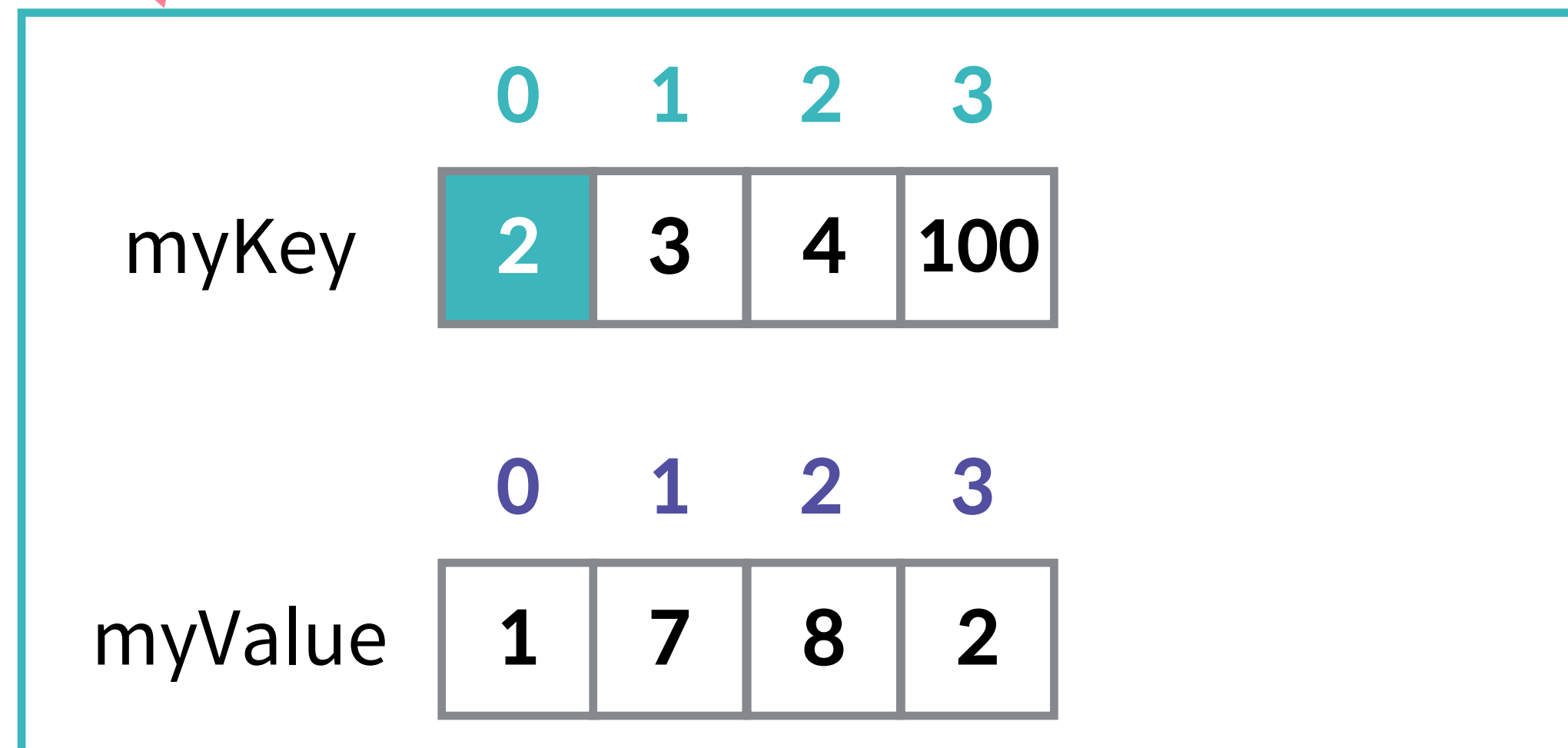
`/* elice */`

07 마무리 및 부록

- ✓ key와 value를 각각 배열에 저장하기

db.get(4)

db = kvStore()



	0	1	2	3
myKey	2	3	4	100
	0	1	2	3
myValue	1	7	8	2

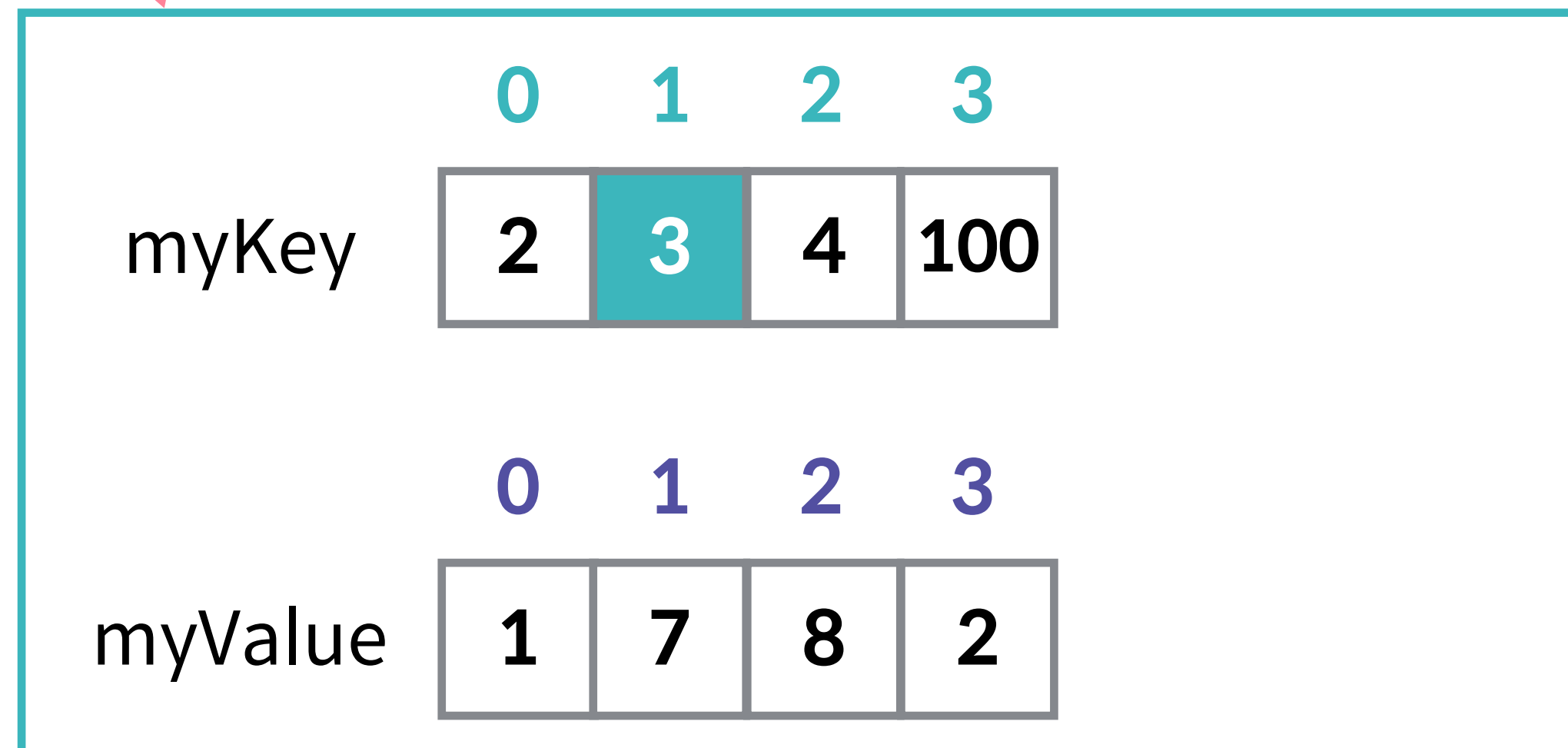
/* elice */

07 마무리 및 부록

- ✓ key와 value를 각각 배열에 저장하기

db.get(4)

db = kvStore()



	0	1	2	3
myKey	2	3	4	100
myValue	1	7	8	2

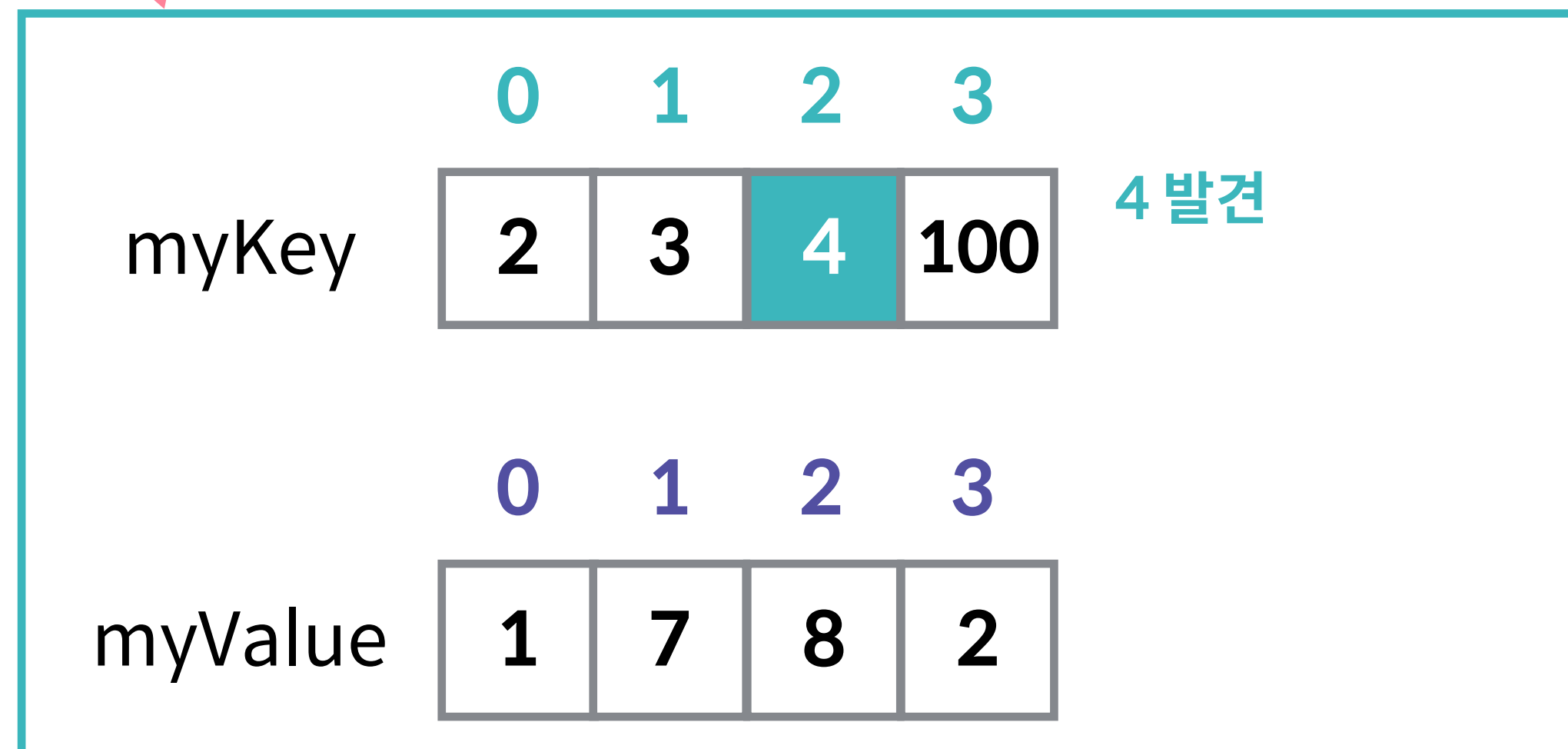
/* elice */

07 마무리 및 부록

- ✓ key와 value를 각각 배열에 저장하기

db.get(4)

db = kvStore()



	0	1	2	3	
myKey	2	3	4	100	4 발견
	0	1	2	3	
myValue	1	7	8	2	

/* elice */

07 마무리 및 부록

✓ key와 value를 각각 배열에 저장하기

db = kvStore()

8

	0	1	2	3
myKey	2	3	4	100
			↓	
	0	1	2	3
myValue	1	7	8	2

/* elice */

07 마무리 및 부록

✓ key와 value를 각각 배열에 저장하기

장점

공간의 낭비가 없다.
자료가 없는 경우를 표현할 수 있다.

단점

자료의 읽기(조회) 연산이 느리다.
자료의 쓰기(입력 등) 연산도 느리다.

✓ 해시(hash)

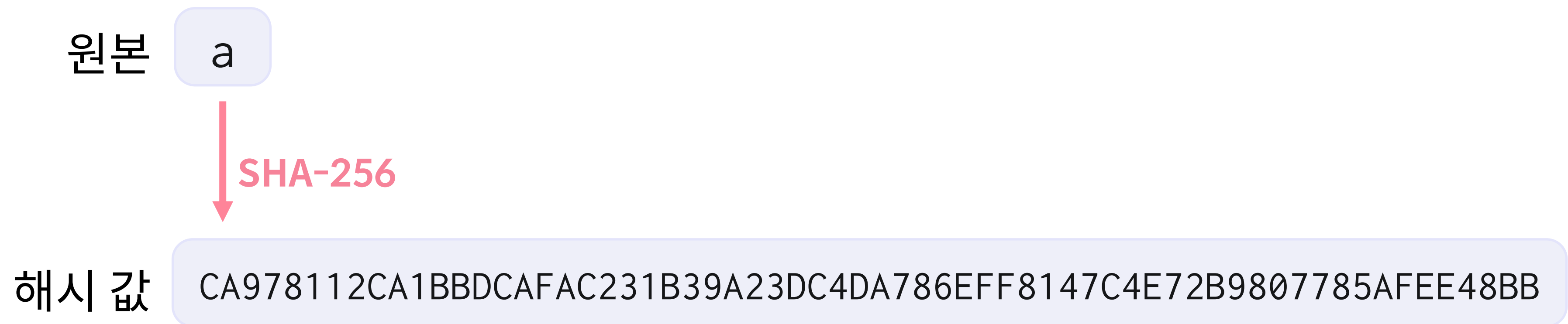
해시란 임의의 데이터에 **해시 함수**를 이용하여
고정된 길이의 데이터(문자열 등)으로 **변환**하는 것을 의미한다.

변환하는 과정 자체를 **해싱(hashing)**,
변환된 값을 **해시 값(hash value)** 라고도 부른다.

07 마무리 및 부록

✓ 해싱 예시 (SHA-256 알고리즘)

해시 알고리즘으로 알려진 대표적인 예시는 **SHA-256**이 있다.



07 마무리 및 부록

✓ 해싱 예시 (SHA-256 알고리즘)

해시 알고리즘으로 알려진 대표적인 예시는 **SHA-256**이 있다.

원본

elice



SHA-256

해시 값

5C15623E912523F2C1585DC812A553F214C5EF1EFAA82A336BF3FA9457150E79

`/* elice */`

07 마무리 및 부록

✓ 해싱 예시 (Python 내장 함수)

Example

```
for i in range(10) :  
    print(hash("a"))
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
2563686919053509095
```

```
코드 실행이 완료되었습니다.
```

파이썬 내장 함수 `hash()`를 이용하여 임의의 값의 **해시 값(정수)**을 알 수 있다.

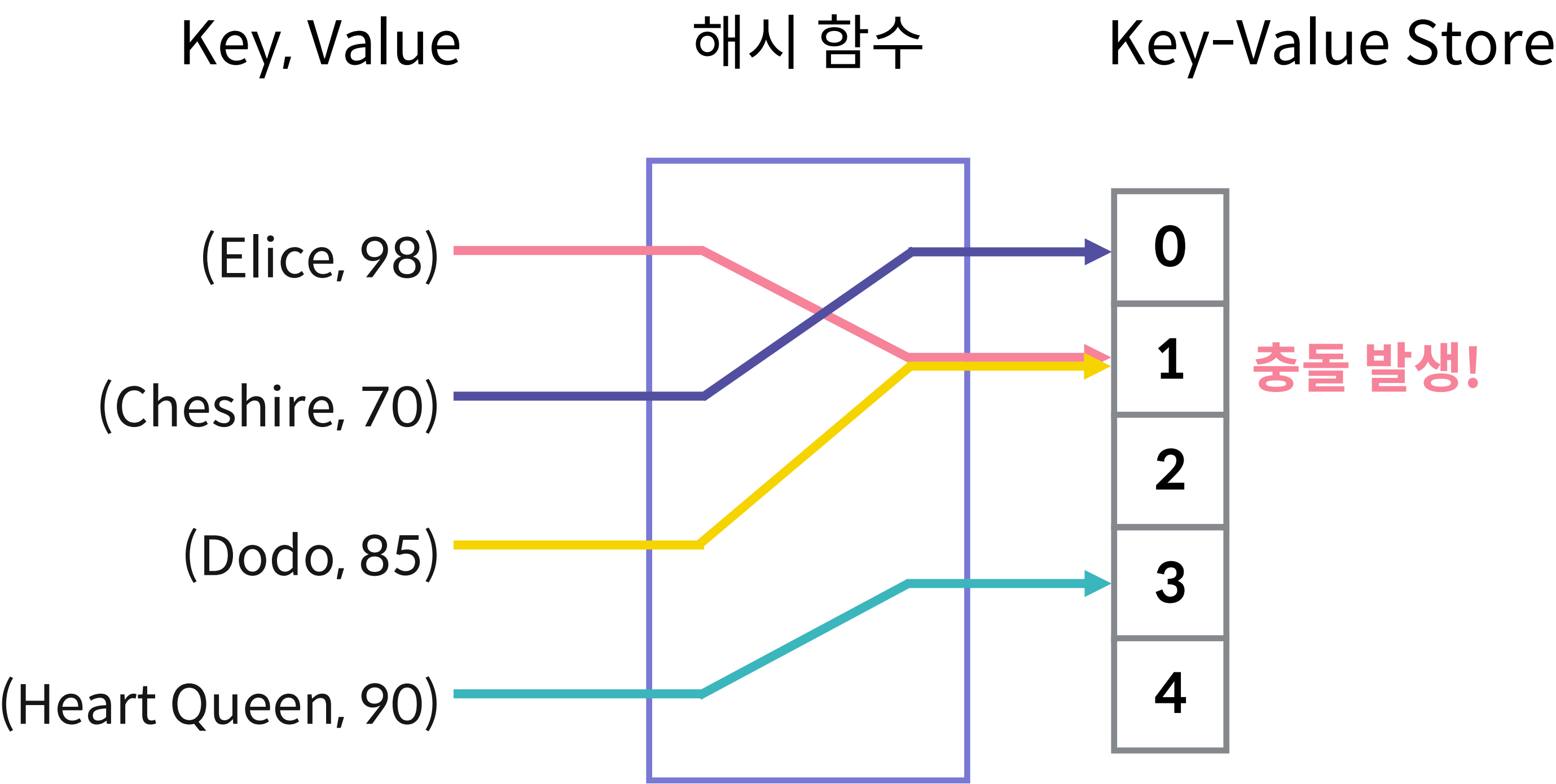
파이썬의 `hash()` 함수는 **보안**을 이유로 프로그램을 실행할 때마다 반환값이 달라진다.

✓ 해시(hash)

해시를 이용하여 Key-Value Store를 구현할 수 있다.

해시를 이용하여 구현된 Key-Value Store를 **해시 테이블(Hash Table)**,
또는 **딕셔너리(Dictionary)**라고 한다.

✓ 해시(hash)



`/* elice */`

✓ 해시 함수의 충돌 : 비둘기집의 원리

좋은 해시 함수는 중복되는 해시 값이 최대한 없도록 하여 되도록 충돌이 발생하지 않는 함수이다.

그러나 해시 함수의 반환값의 경우의 수는 **유한**하고,
입력값의 경우의 수는 **무한**하기 때문에
비둘기집의 원리에 의해 충돌은 **어쩔 수 없이 발생**한다.

✓ 해시 함수의 충돌 : 비둘기집의 원리

n 개의 비둘기집에 $(n + 1)$ 마리의 비둘기를
한 집에 한 마리씩 넣는 것은 불가능하다는 원리를 말한다.

비둘기가 **최대 1마리**씩 들어있는 비둘기집이 **n 개** 있다면
전체 비둘기집에는 **비둘기가 많아야 n 마리** 존재한다.

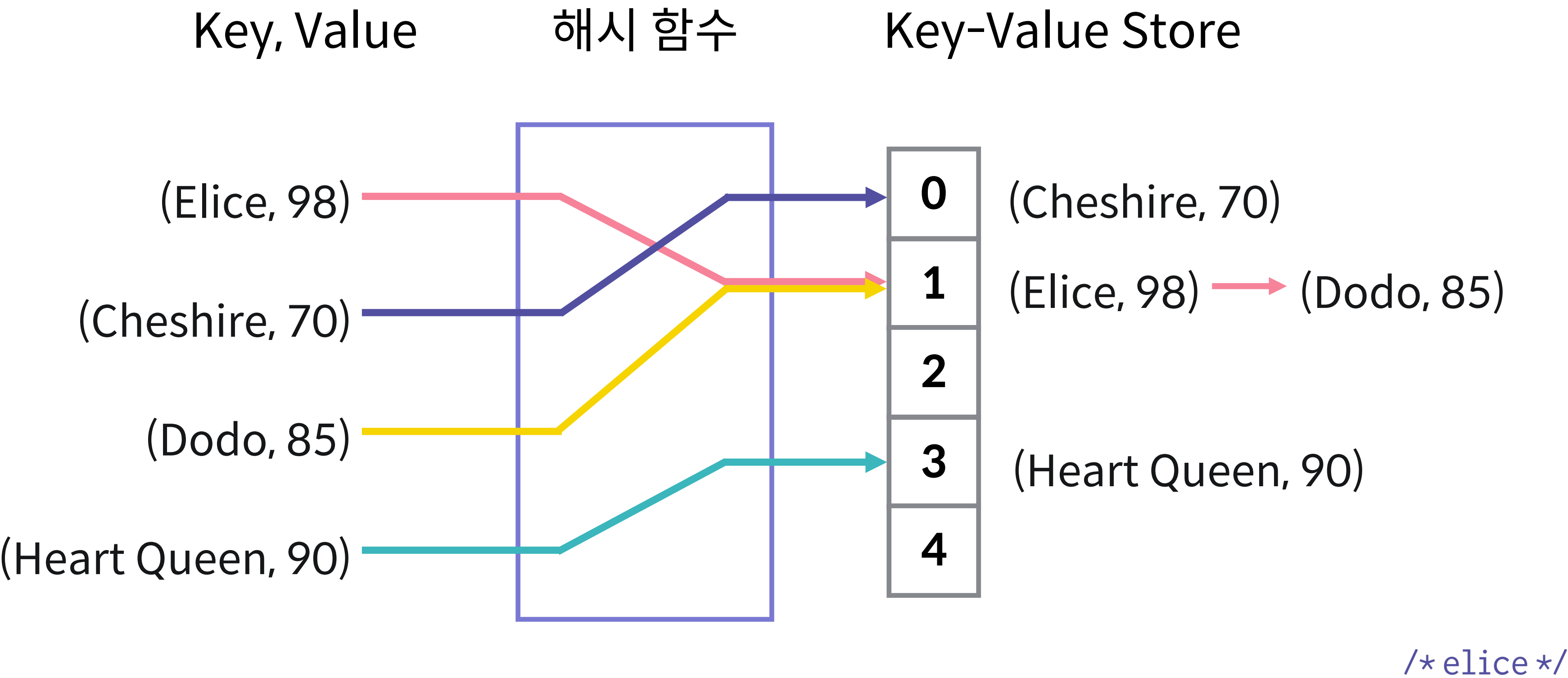
따라서 $(n + 1)$ 마리의 비둘기를 n 개의 비둘기집에 한 마리씩 넣는 것은 불가능하다.

07 마무리 및 부록

✓ 해시 함수의 충돌 : 비둘기집의 원리

연락처에 있는 사람 366명의 생일을 올해 달력에 적으려 한다.
만약 올해가 윤년이 아니고 2월 29일이 생일인 사람이 없다면
한 날짜에 두 사람의 이름을 적어야 하는 경우가 반드시 생긴다.

✓ 해시 - 충돌 해결 방법 : 개별 체이닝

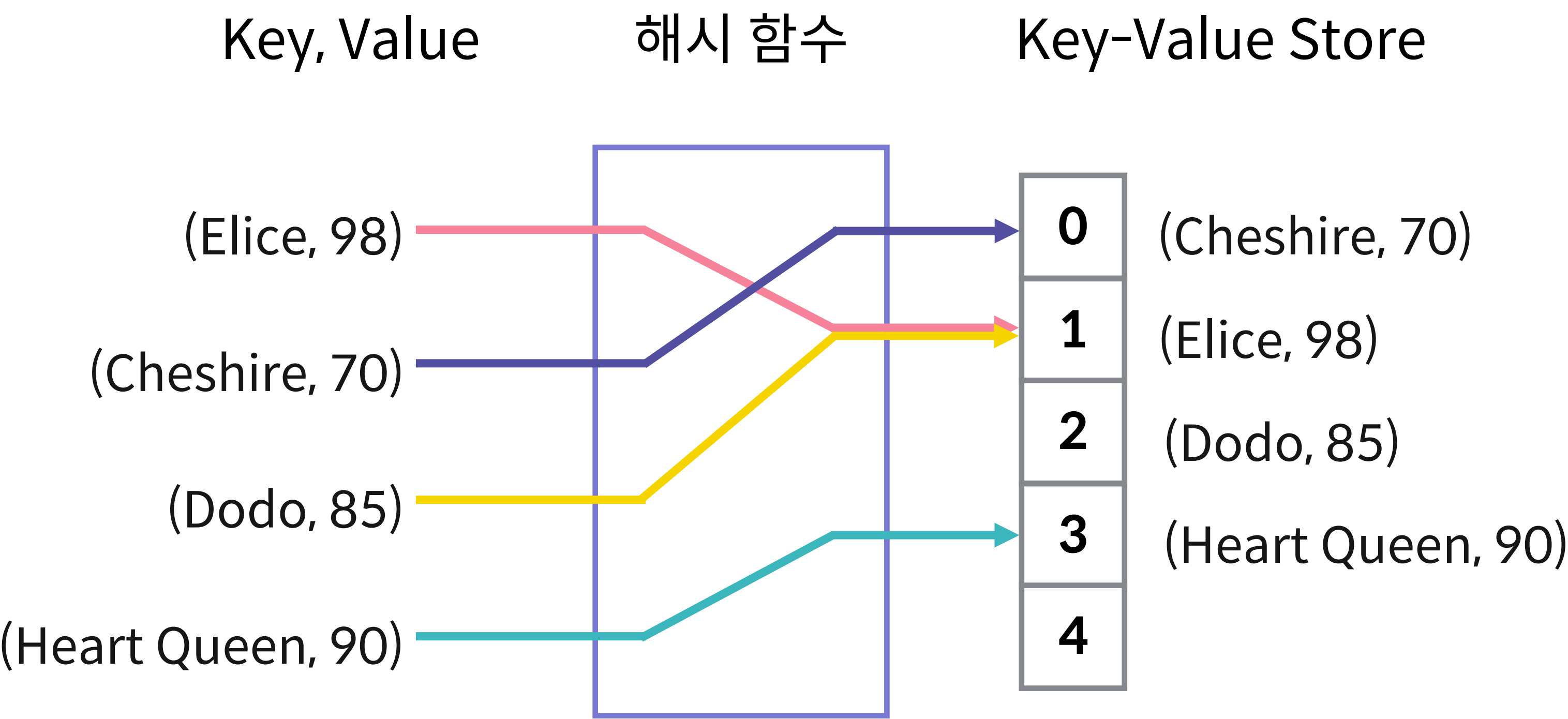


✓ 해시 - 충돌 해결 방법 : 개별 체이닝

Key-Value Store의 각 인덱스를 '**연결 리스트**'로 만들어서
동일한 인덱스의 값들을 연결하는 방법이다.

위 예에서는 동일한 인덱스(1)를 갖는 Elice와 Dodo 자료를
서로 연결하였다.

✓ 해시 - 충돌 해결 방법 : 오픈 어드레싱



`/* elice */`

07 마무리 및 부록

✓ 해시 - 충돌 해결 방법 : 오픈 어드레싱

충돌이 발생했을 때 자료를 저장하기 위해 **빈 공간을 탐색**하는 방식이다.

따라서 모든 원소가 자신의 **해시 값과 일치하는 인덱스에 저장된다는 보장은 없다.**

위 예시에서는 Dodo 자료를 저장하기 위해 가장 가까운 빈 공간을 탐색하였고,
그 결과 인덱스 2에 저장하였다.

✓ 해시 - 충돌 해결 방법 : 오픈 어드레싱

오픈 어드레싱에서 빈 공간을 찾는 방법은 여러 가지가 있지만
가장 간단하고 대표적인 방법은 '**선형 탐사 방식**'이다.
앞서 살펴 본 것 처럼, 원래 인덱스의 **다음 인덱스**부터 탐색하여
가장 가까운 빈 공간을 찾는 방법이다.

07 마무리 및 부록

✓ 해시 - 충돌 해결 방법 : 오픈 어드레싱

Python에서는 딕셔너리의 해시 값 충돌이 발생하였을 때
오픈 어드레싱 방식으로 해결하도록 구현되어 있다.