12 강

알고리즘과 자료구조

해시테이블

서울과학기술대학교 신일훈 교수

한국방송통신대학교 프리임칼리지

학습목표

1. 해시 테이블의 개념을 이해한다.
2. 충돌 및 충돌의 해결 방안을 이해한다.
3. 충돌 해결 방안 중 선형 조사를 구현할 수 있다.

01

해시 테이블 개념

1. 해시 테이블 개념

- 연결리스트, 큐, 스택의 특성 및 장점
  - 선형 자료구조
  - 구현 용이
  - 스택, 큐 등은 LIFO, FIFO 등의 정책을 구현하기에 효율적인 자료구조

1. 해시 테이블 개념

❑ 연결리스트, 큐, 스택의 단점 및 한계

- 탐색에는 적합하지 않음
- 선형 (순차) 탐색의 비효율성
  - 최악의 경우, N개의 노드를 모두 검사해야 함

- 예> 아파트의 우편물 배송
  - 큐로 구현한다면? 우리 집 우편물 찾기 매우 어려움.
  - 해결책?

1. 해시 테이블 개념

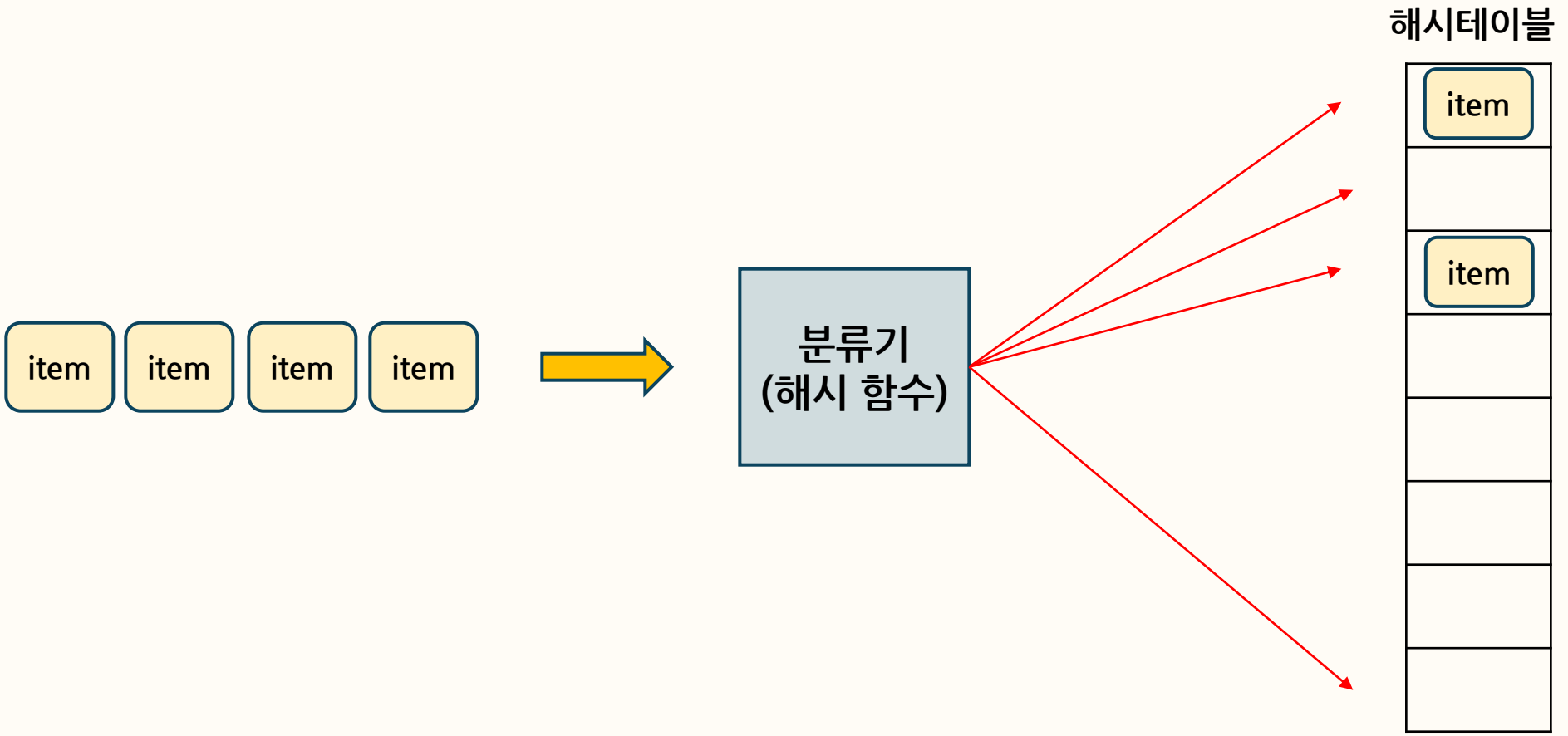
핵심 아이디어

- 탐색 시, 탐색할 아이템 (탐색 공간)의 개수를 줄이자.
- 예> 아파트 각 동의 우편함
  - 자기 집 우편함만 찾으면 됨

1. 해시 테이블 개념

❑ 핵심 아이디어

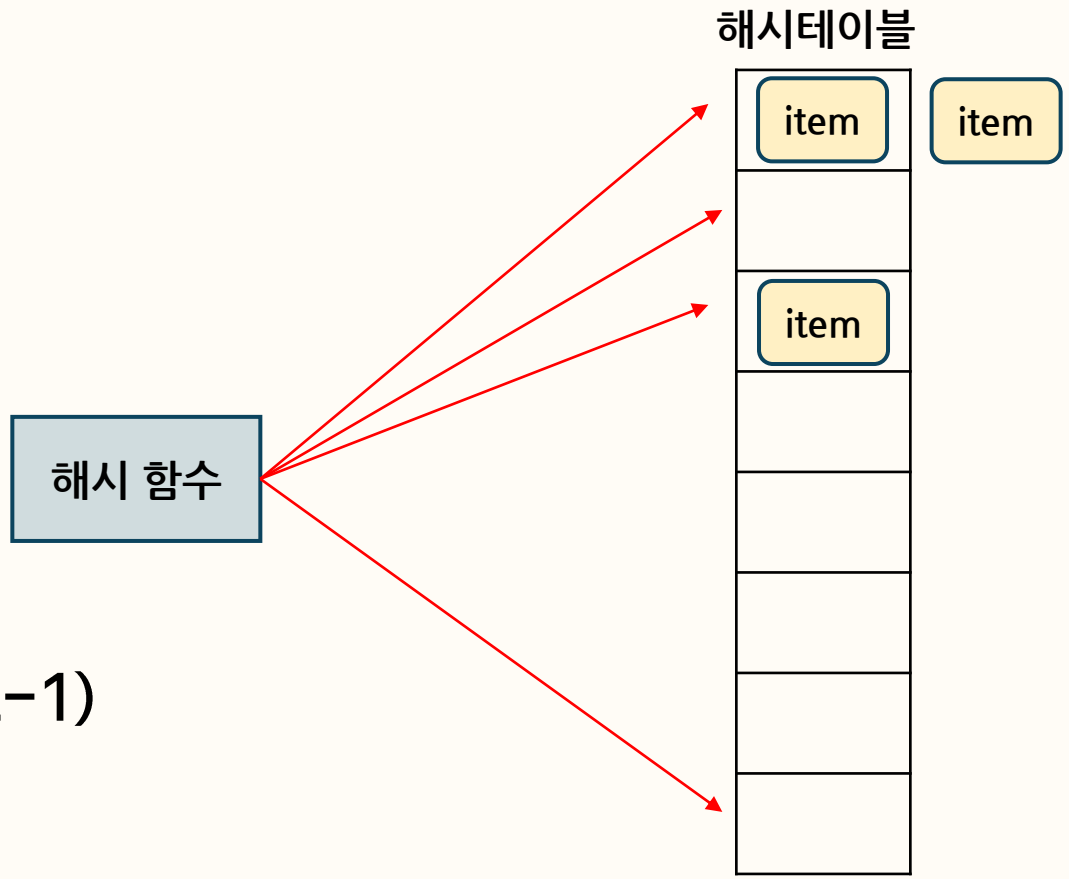
- 탐색 시, 탐색할 아이템 (탐색 공간)의 개수를 줄이자.



1. 해시 테이블 개념

❑ 해시 함수

- 이상적인 해시함수의 조건
  - 균등 분산 (충돌 방지, 탐색 공간 줄이기)
  - low 계산 오버헤드
- 해시함수의 예
  - % (나머지 연산)
    - key % 해시 테이블 크기 (L) => 0 ~ (L-1)
  - 기타
    - (key를 이용한 여러 연산) % L



02

충돌 및 해결 기법

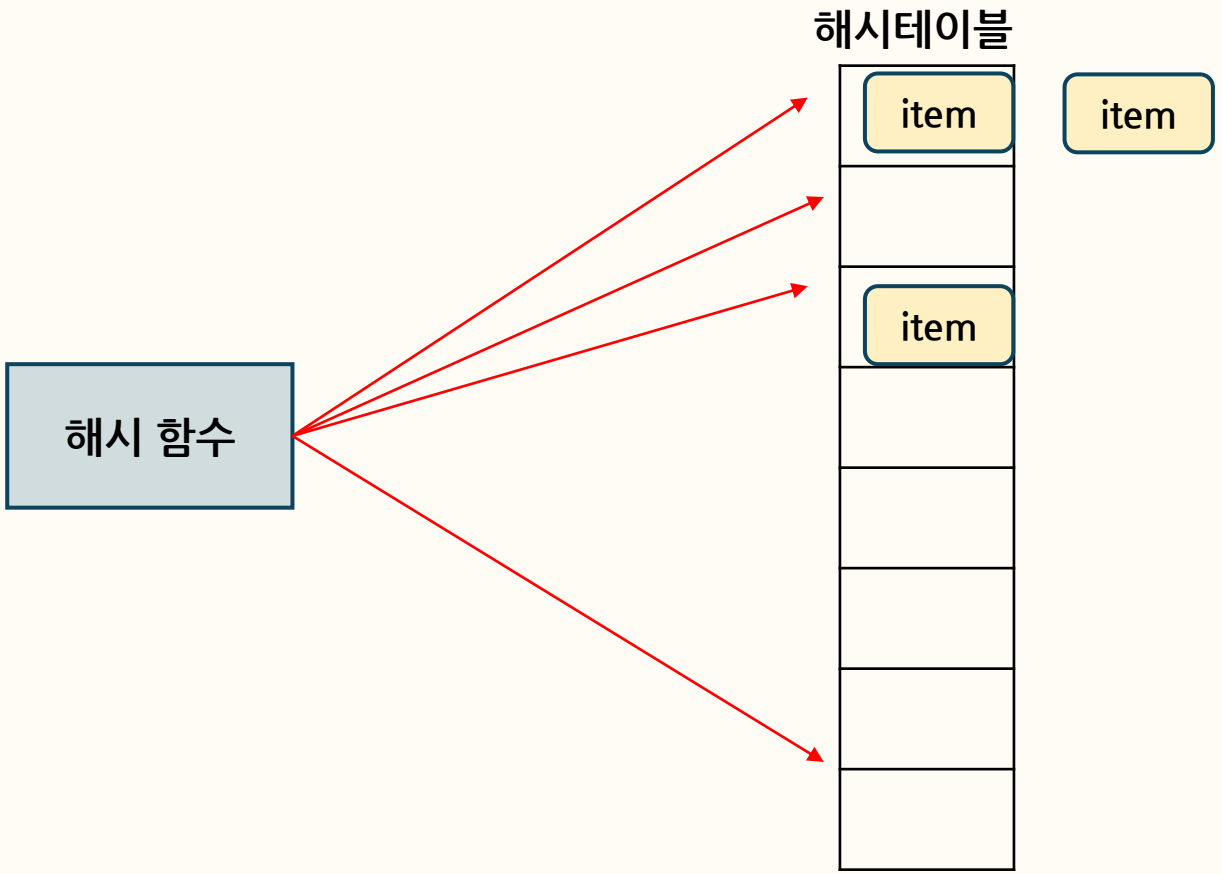
2. 충돌 및 해결 기법

❑ 충돌

- key가 다름에도 불구하고 해시 함수의 출력이 동일함.
- key가 다름에도 불구하고 동일한 인덱스로 매핑됨.
- 예> %를 해시 함수로 사용하고, 해시 테이블의 크기가 130이라면,
  - 7, 20, 33, … 등의 해시 함수 출력이 모두 동일함.

2. 충돌 및 해결 기법

❑ 충돌



2. 충돌 및 해결 기법

❑ 충돌 해결 기법

- 선형 조사
- 이중 해싱
- 체이닝

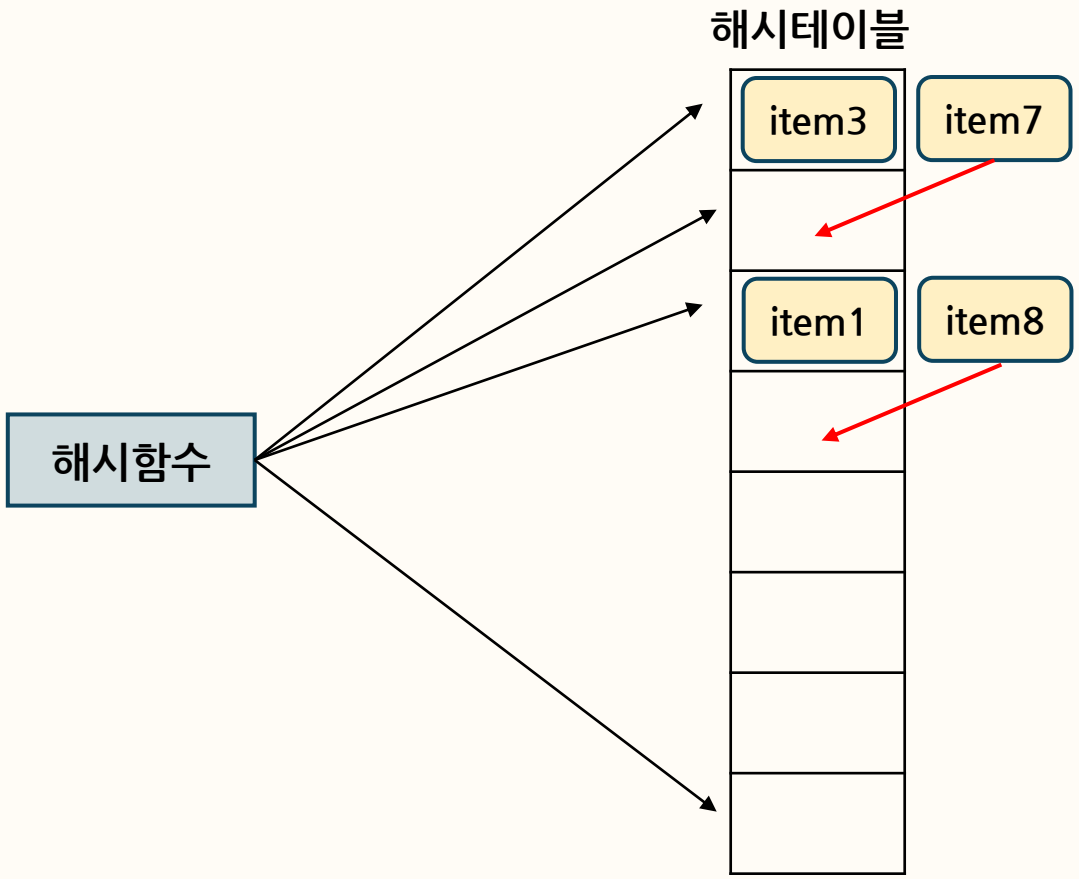
2. 충돌 및 해결 기법

❑ 선형 조사(linear probing)

- 충돌이 발생하면, 순차적으로 다음의 빈 자리를 탐색하여 저장한다.
- 검색 연산
  1. 해시함수를 통해 타겟 인덱스 발견
  2. 타겟 인덱스의 아이템이 찾는 아이템인지 체크
  3. 아니면 그 다음 인덱스로 진행하여, 찾는 아이템인지 체크
  4. 2-3 작업을 빈 자리를 만날 때까지 혹은 처음 인덱스에 이를 때까지 반복
- clustering (군집화) 발생 가능 & 성능 저하

2. 충돌 및 해결 기법

❑ 선형 조사(linear probing)



2. 충돌 및 해결 기법

❑ 이중 해싱 (double hashing)

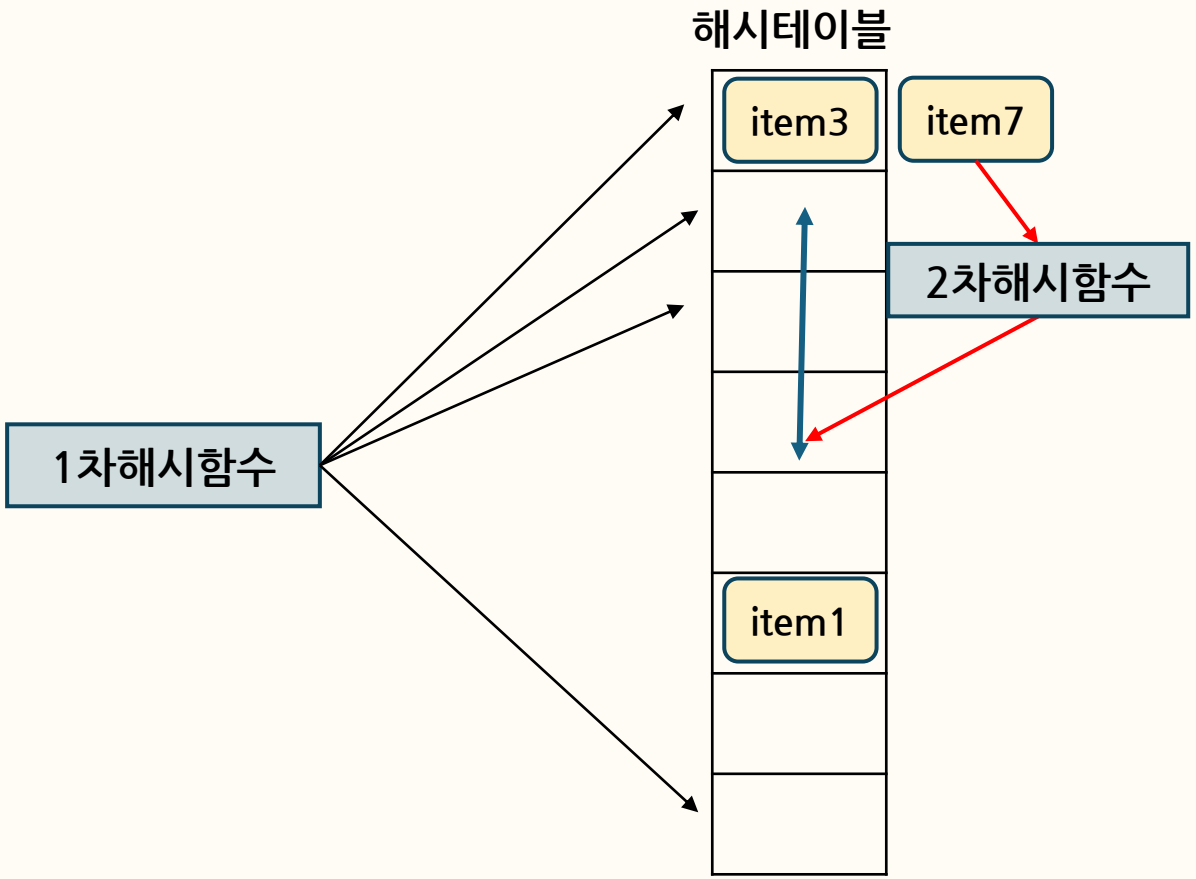
- 충돌이 발생하면, 2차 해시 함수를 적용하여 해당 distance 만큼 떨어진 위치를 다음 자리로 결정
- 저장 연산

1. 1차 해시 함수로 타겟 인덱스 결정 & 타겟 인덱스가 비어 있으면 저장
2. 충돌 발생하면 2차 해시함수 적용하여 거리 계산
3. 현재 타겟 인덱스에 거리를 더한 인덱스를 새로운 타겟 인덱스로 결정
4. 빈 인덱스를 발견할 때까지 (혹은 처음 위치에 이를 때까지) 2-3번 반복.

2. 충돌 및 해결 기법

❑ 이중해싱 (double hashing)

- 저장 연산



2. 충돌 및 해결 기법

❑ 이중해싱 (double hashing)

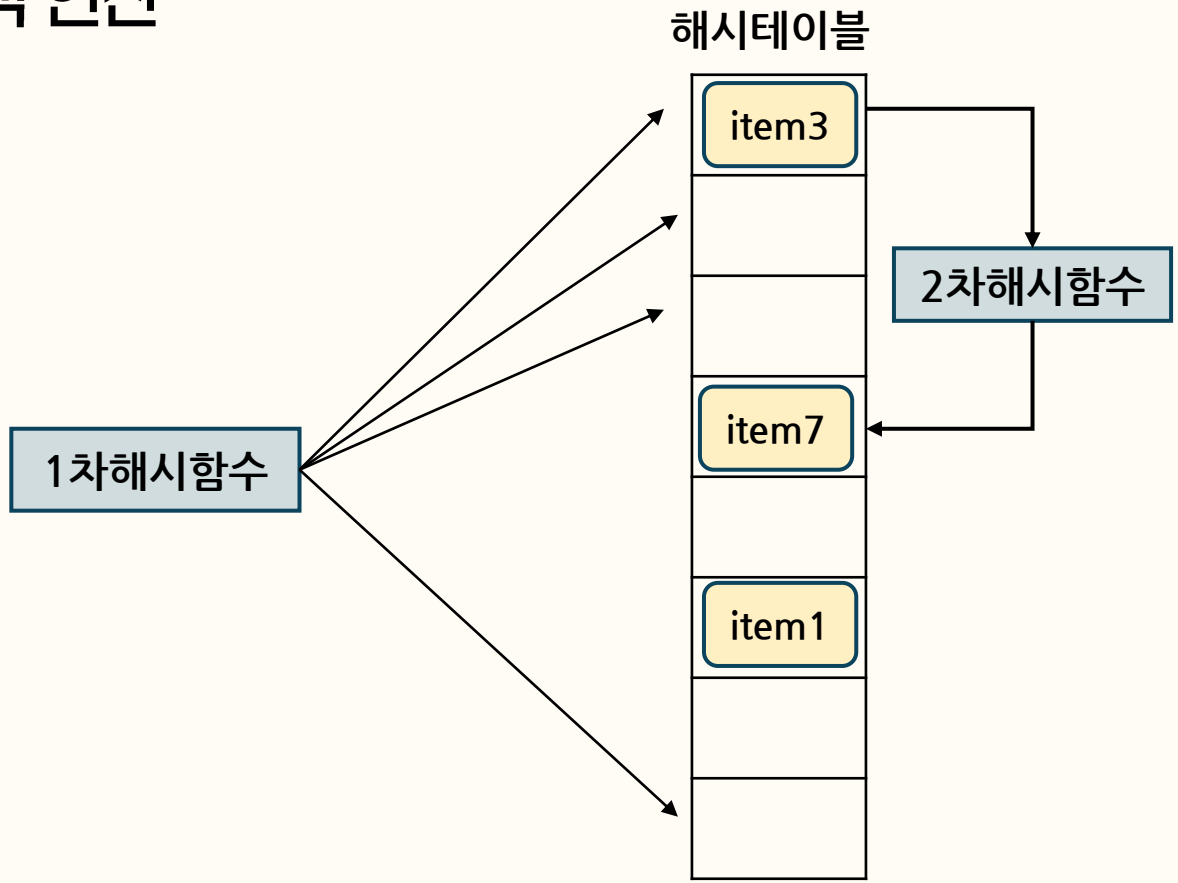
- 검색 연산

1. 1차 해시 함수를 통해 타겟 인덱스 발견
2. 타겟 인덱스의 원소가 찾는 아이템인지 체크
3. 아니면 2차 해시함수를 적용하여 다음 인덱스로 진행하여
   해당 인덱스의 원소가 찾는 아이템인지 체크
4. 2-3 작업을 빈 인덱스를 만날 때까지
   (혹은 처음 인덱스에 이를 때까지) 반복

2. 충돌 및 해결 기법

❑ 이중해싱 (double hashing)

- 검색 연산



2. 충돌 및 해결 기법

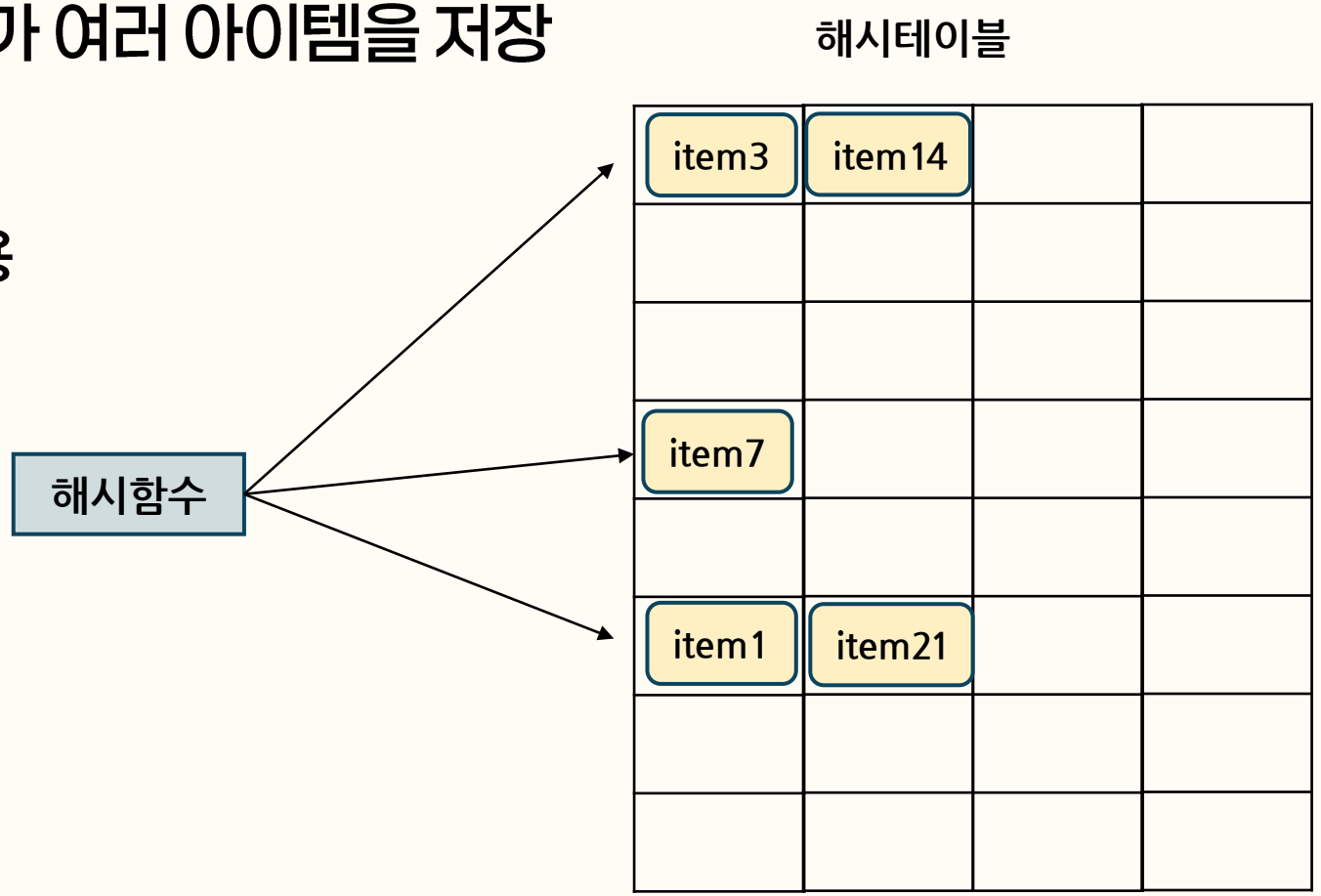
체이닝 (chaining)

- 해시 테이블의 각 원소가 여러 아이템을 저장
  - 방법1:
    - 2차원 배열을 활용하여 여러 아이템을 저장할 수 있음
    - 저장 가능한 아이템의 최대 개수가
      배열의 column 값에 의해 결정됨

2. 충돌 및 해결 기법

❑ 체이닝 (chaining)

- 해시테이블의 각 원소가 여러 아이템을 저장
  - 방법1:
    - 2차원 배열을 활용



2. 충돌 및 해결 기법

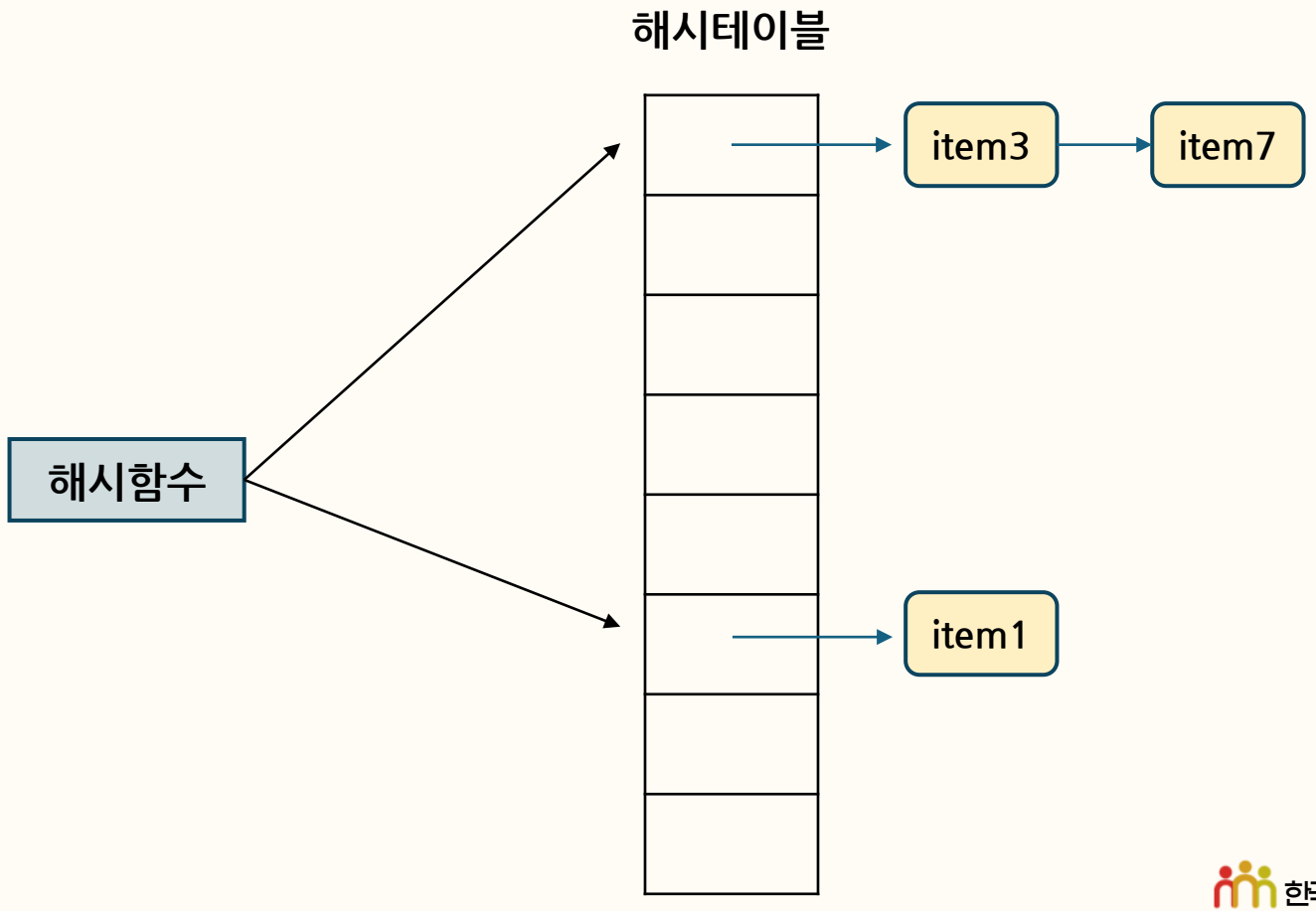
체이닝 (chaining)

- 해시테이블의 각 원소가 여러 아이템을 저장
  - 방법2:
    - 1차원 배열을 사용하되, 배열의 각 원소를 연결 리스트의 헤더로 사용.
    - 동일한 원소로 해시된 아이템들을 연결 리스트로 연결

2. 충돌 및 해결 기법

❑ 체이닝 (chaining)

- 해시테이블의 각 원소가 여러 아이템을 저장
  - 방법2:



03

선형 조사 구현

3. 선형 조사 구현

- 클래스 LinearProbing 정의
  - 선형조사 방식의 해시 테이블
  - 멤버 변수
    - tablesize : 해시 테이블의 크기
    - table: 해시 테이블
      리스트이며, 각 원소는 (key, value)의 튜플

3. 선형 조사 구현

- 클래스 LinearProbing 정의
  - 메서드
    - 생성자
    - hash()
    - add()
    - search()
    - remove()
    - print()

3. 선형 조사 구현

클래스 LinearProbing 정의 (생성자)

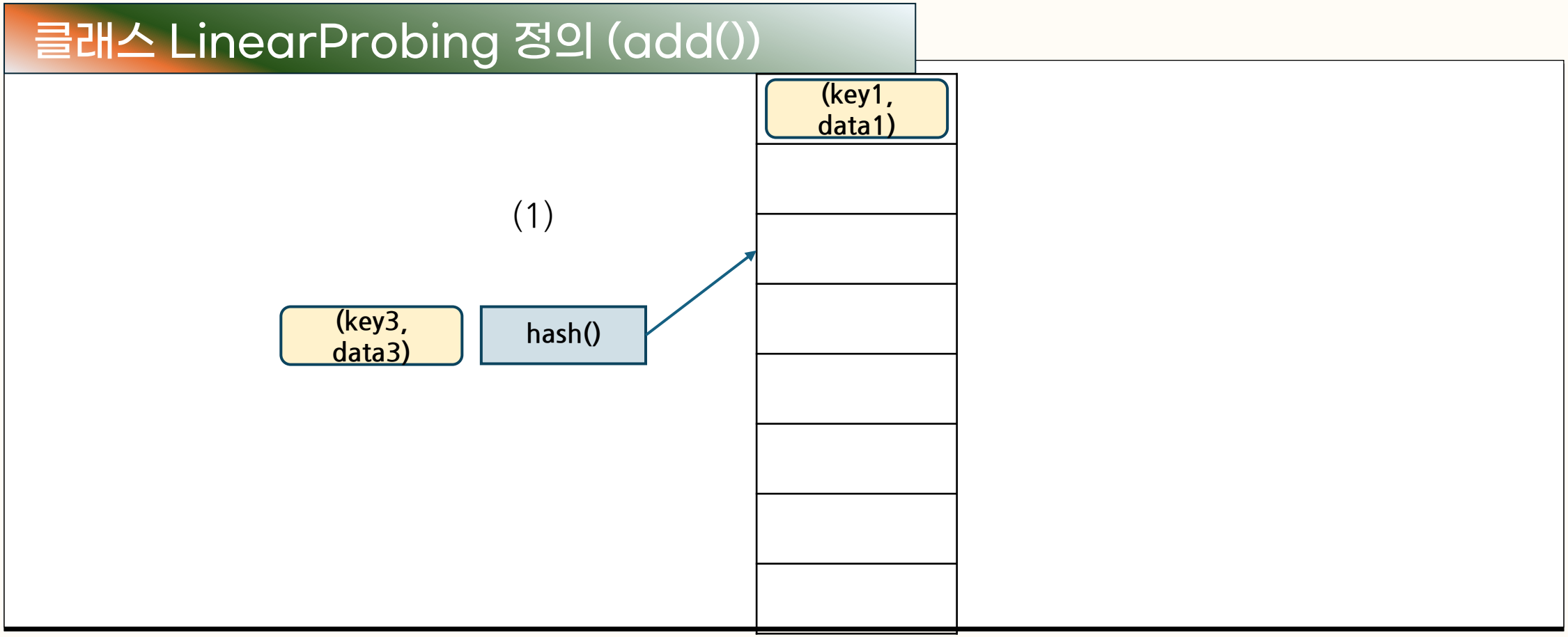
```
class LinearProbing:
    def __init__(self, size):        # 해시 테이블을 빈 상태로 초기화해야 함
        self.tablesize = size
        self.table = [(None, None)] * size
```

3. 선형 조사 구현

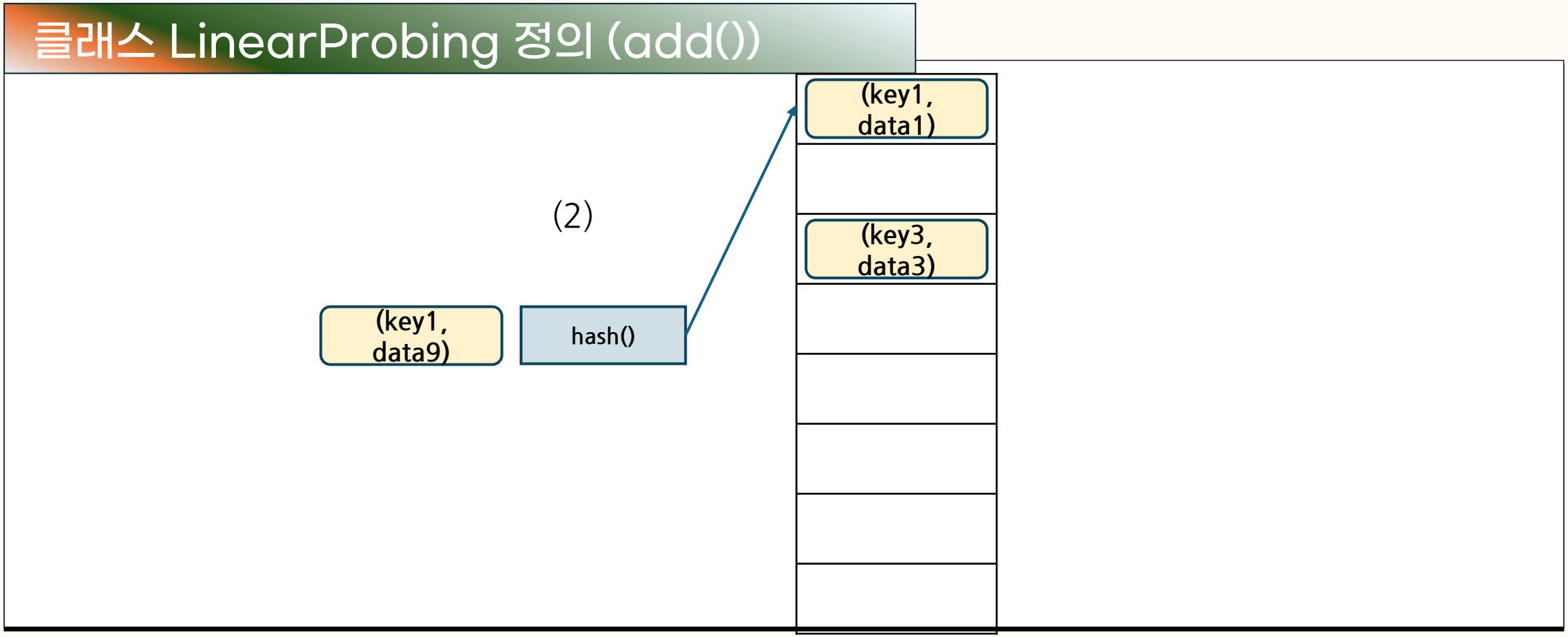
클래스 LinearProbing 정의 (hash())

```
class LinearProbing:
    def hash(self, key):
        return key % self.tablesize
```

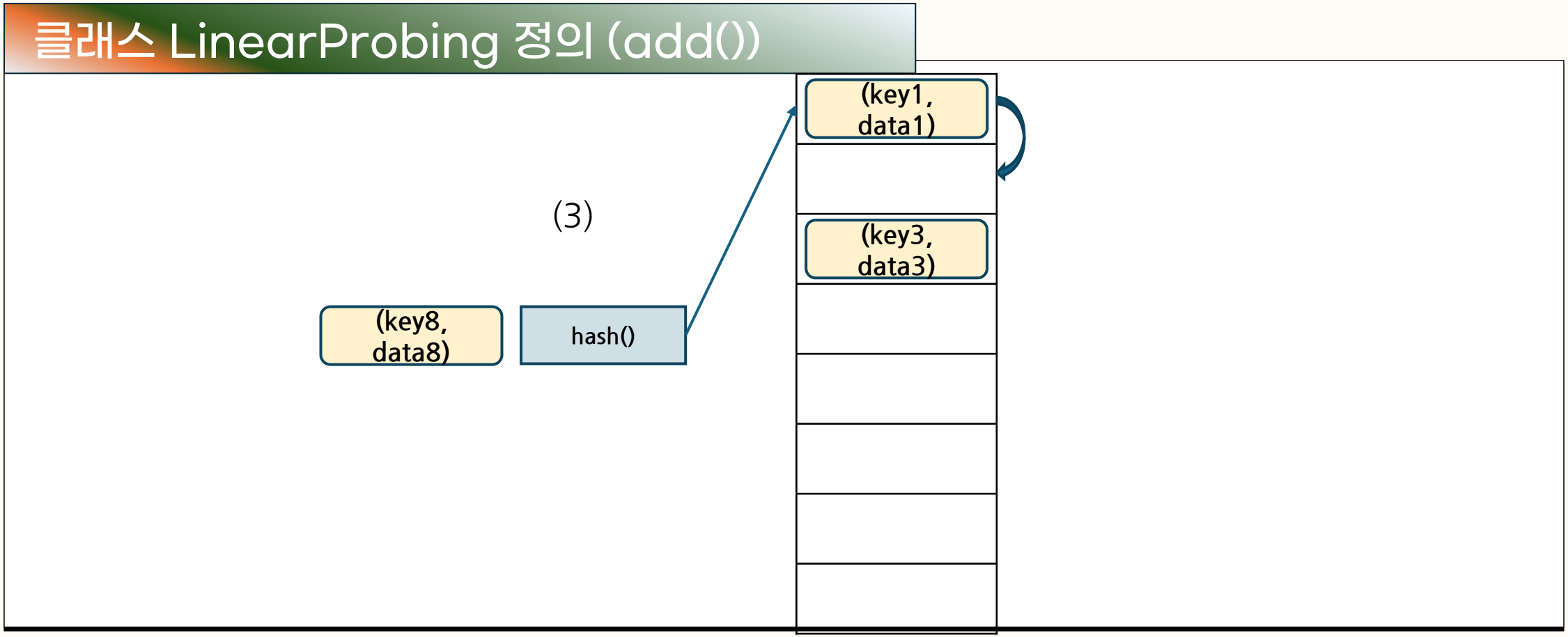
3. 선형 조사 구현



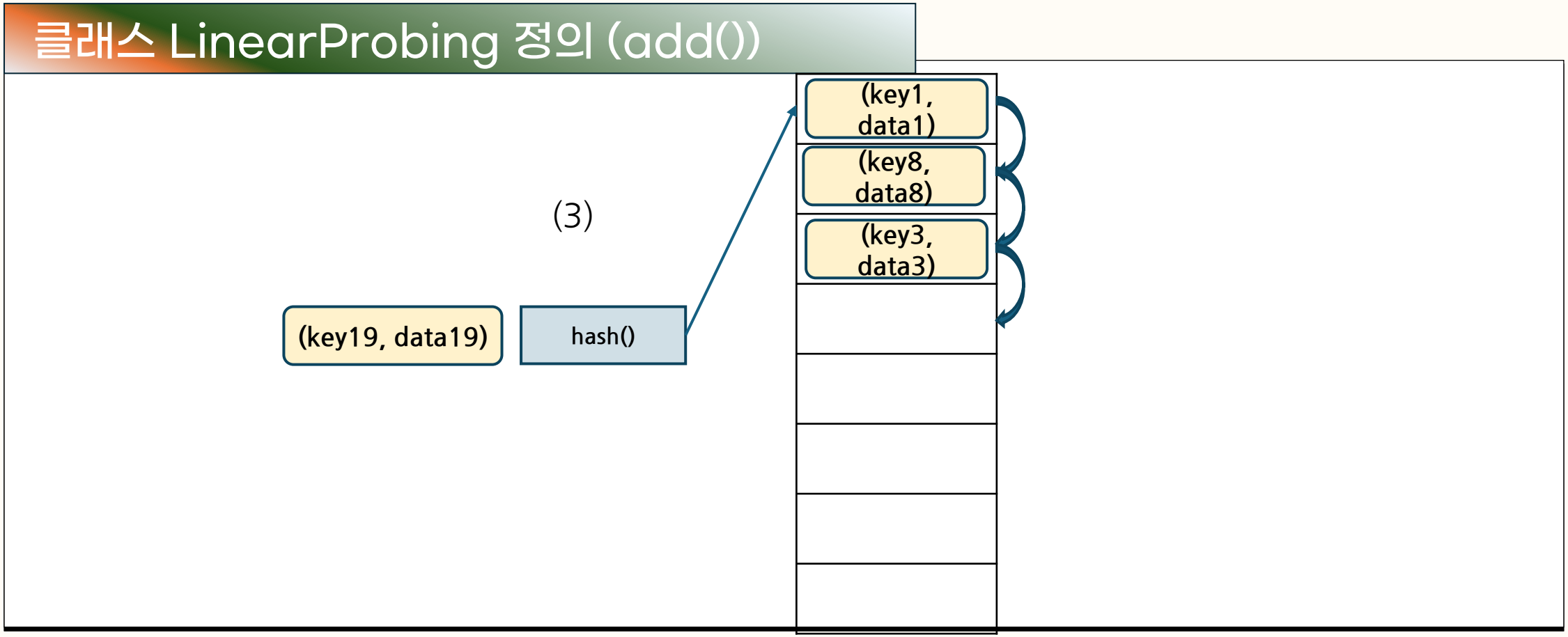
3. 선형 조사 구현


클래스 LinearProbing 정의 (add())
(key1, data1)
(2)
(key3, data3)
(key1, data9)
hash()

3. 선형 조사 구현

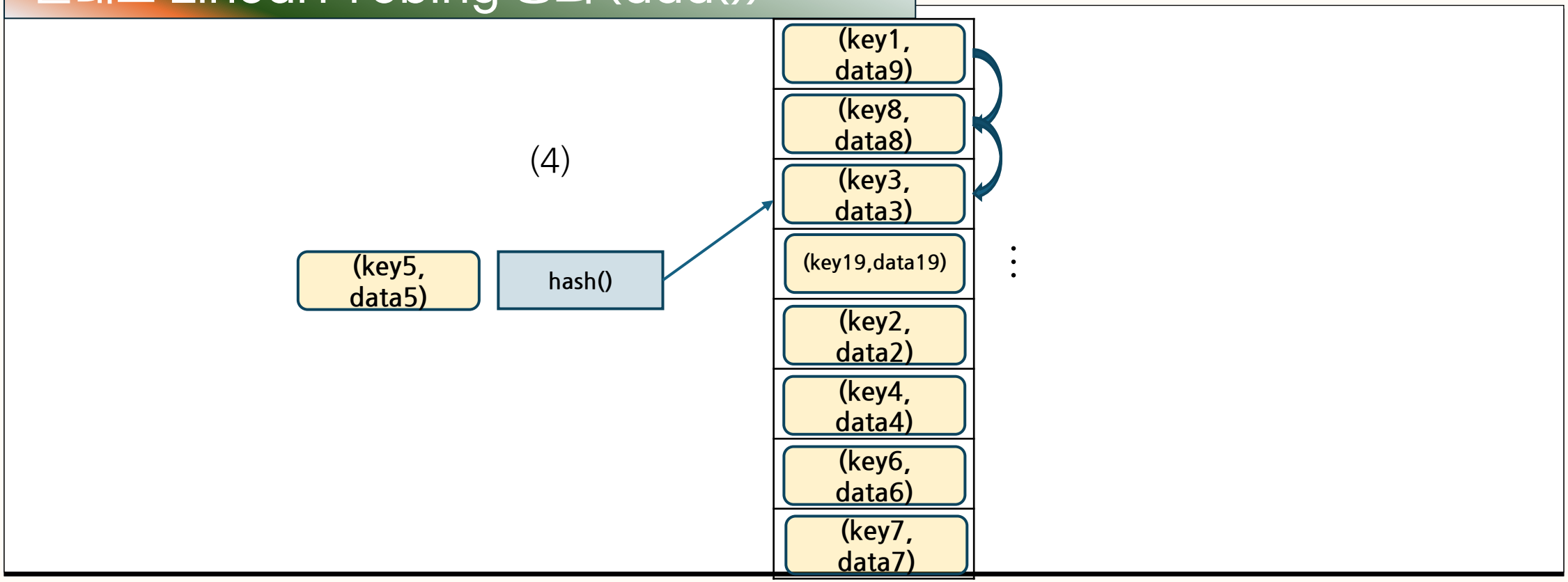

클래스 LinearProbing 정의 (add())
(key1, data1)
(3)
(key3, data3)
(key8, data8)
hash()

3. 선형 조사 구현


클래스 LinearProbing 정의 (add())
(key1, data1)
(key8, data8)
(key3, data3)
(3)
(key19, data19)
hash()

3. 선형 조사 구현

클래스 LinearProbing 정의 (add())



3. 선형 조사 구현

클래스 LinearProbing 정의 (add())

```
class LinearProbing:
    def add(self, key, value):
        initial_position = self.hash(key)
        position = initial_position
```

3. 선형 조사 구현

클래스 LinearProbing 정의 (add())

```
class LinearProbing:
    def add(self, key, value):
        ...
        while True:
            (fkey, fvalue) = self.table[position]
            if fvalue == None:          # (1) 빈 버킷 발견 & 추가
                self.table[position] = (key, value)
                return  True
            elif fkey == key:           # (2) 동일 아이템 발견 & 값 수정
                self.table[position] = (key, value)
                return  True
```

3. 선형 조사 구현

클래스 LinearProbing 정의 (add())

```
class LinearProbing:
    def add(self, key, value):
        ...
        while True:
            ...
            position = (position + 1) % self.tablesize    # 충돌 & 다음 버킷 이동
            if position == initial_position:              # 추가할 빈 버킷 없음
                return False
```

3. 선형 조사 구현

클래스 LinearProbing 정의 (print())

```
class LinearProbing:
    def print(self):
        i = 0
        for tuple in self.table :
            print(i, tuple)
            i += 1
        print()
```

3. 선형 조사 구현

LinearProbing 1차 테스트

```
if __name__ == '__main__':
    t = LinearProbing(7)
    t.add(7, 'grape')
    t.add(1, 'apple')
    t.add(2, 'banana')
    t.print()
    t.add(15, 'orange')
    t.print()
```

3. 선형 조사 구현

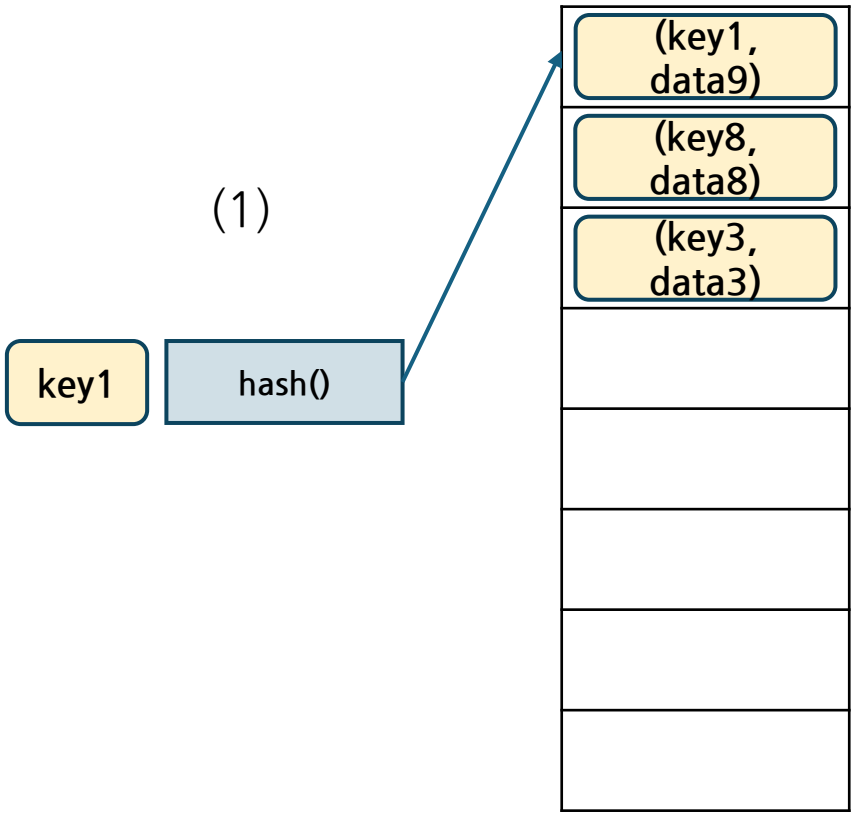
LinearProbing 1차 테스트 실행

```
0 (7, 'grape')
1 (1, 'apple')
2 (2, 'banana')
3 (None, None)
4 (None, None)
5 (None, None)
6 (None, None)

0 (7, 'grape')
1 (1, 'apple')
2 (2, 'banana')
3 (15, 'orange')
4 (None, None)
5 (None, None)
6 (None, None)
```

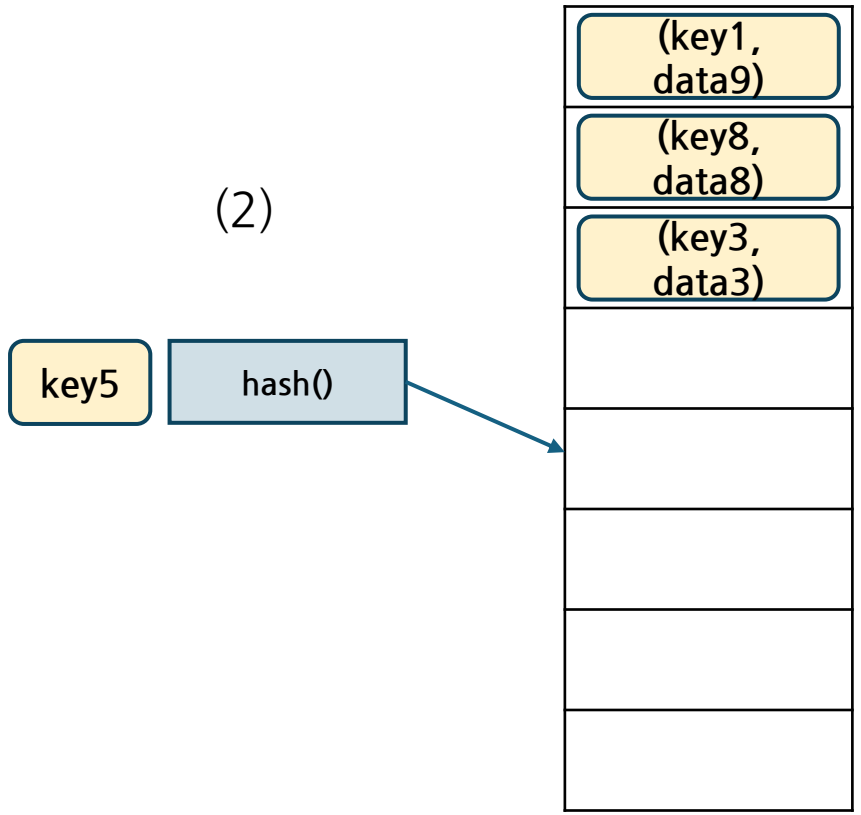
3. 선형 조사 구현

클래스 LinearProbing 정의 (search())



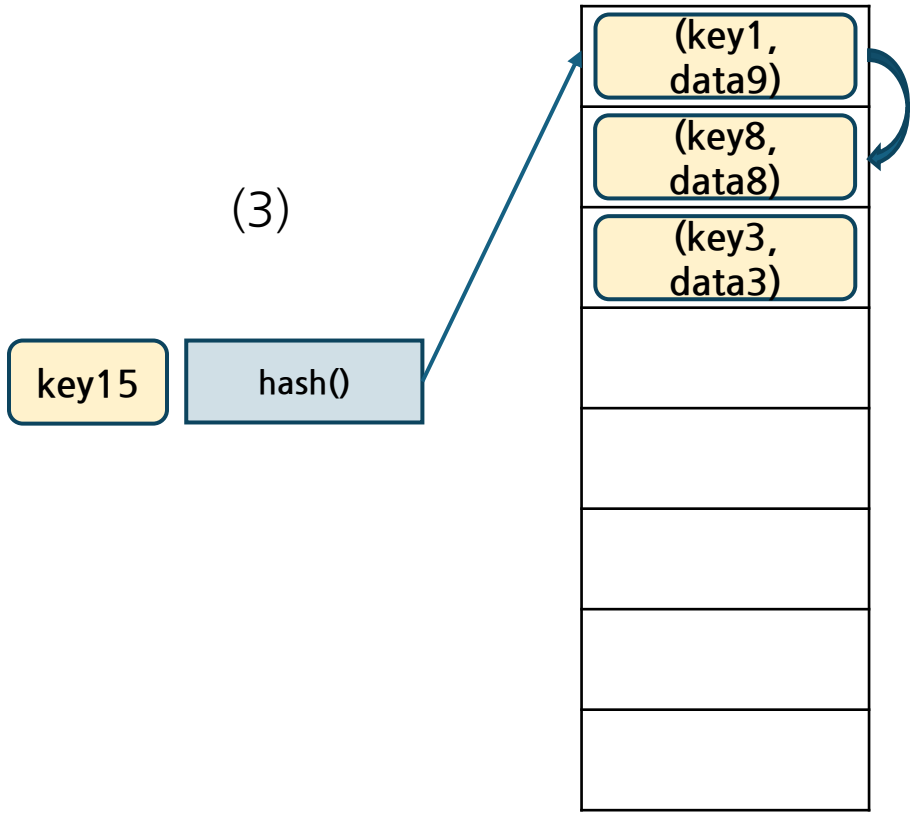
3. 선형 조사 구현

클래스 LinearProbing 정의 (search())



3. 선형 조사 구현

클래스 LinearProbing 정의 (search())



3. 선형 조사 구현

클래스 LinearProbing 정의 (search())

```
class LinearProbing:
    def search(self, key):
        initial_position = self.hash(key)
        position = initial_position
```

3. 선형 조사 구현

클래스 LinearProbing 정의 (search())

```
class LinearProbing:
    def search(self, key):
        ...
        while True:
            (fkey, fvalue) = self.table[position]
            if fkey == key:
                return fvalue
            elif fkey == None:
                return None
```

3. 선형 조사 구현

클래스 LinearProbing 정의 (search())

```
class LinearProbing:
    def search(self, key):
        ...
        while True:
            ...
            position = (position + 1) % self.tablesize
            if position == initial_position:
                return None
```

3. 선형 조사 구현

LinearProbing 2차 테스트

```
if __name__ == '__main__':
    t = LinearProbing(7)
    t.add(7, 'grape')
    t.add(1, 'apple')
    t.add(2, 'banana')
    t.add(15, 'orange')
    print('1의 data = ', t.search(1))
    print('15의 data = ', t.search(15))
    print('11의 data = ', t.search(11))
```

3. 선형 조사 구현

LinearProbing 2차 테스트 실행

```
1의 data =  apple
15의 data =  orange
11의 data =  None
```

정리하기

- 해시 테이블의 개념
- 충돌 및 해결 방안
- 선형 조사 구현

13 강

다음 시간 안내 ▶▶▶

트리 1