# 자료구조

## L10: Hashing

2022년 1학기

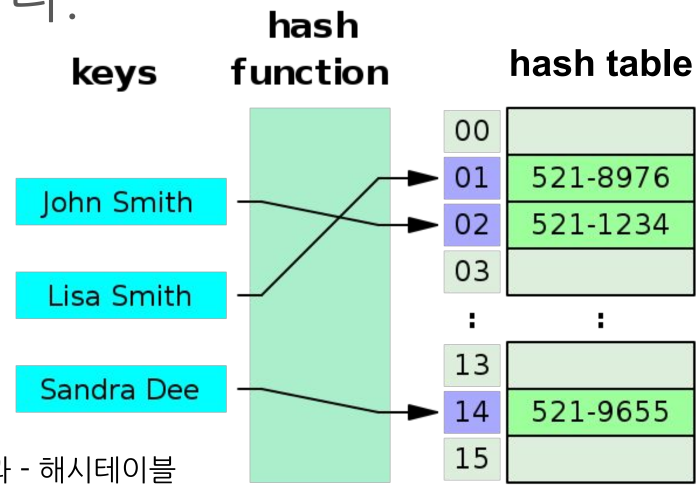
국민대학교 소프트웨어학부

# Overview

- **Hashing**
- Hash Functions
- Collision Resolution
- Analysis of Closed Hashing

# Hashing

- 키가 K인 아이템을 상수 시간에 찾을 수 있을까?
- Hashing
  - Hash Table (HT): 아이템을 담는 배열. 크기는 M.
  - Hash Function (h): 키 K를 0 ~ M-1의 값으로 매핑하는 함수
    - 예) h(K) = K % M
  - Hashing은 키가 K인 아이템을 HT[h(K)]에 넣는다.
    - 찾을 때도 HT[h(K)]에서 찾는다.




그림출처: 위키백과 - 해시테이블

# Hashing

- Hashing은 키의 개수에 비해서 키의 범위가 클 때 유용
- M개의 키가 0부터 M-1까지 순차적으로 있다면?
- M개의 키가 X부터 X+M-1 까지 순차적으로 있다면?
- M개의 키가 0부터 N (M << N)사이에 듬성듬성 있다면?
- 키가 문자열이라면?

# Collision

- Collision: 두 키 K1과 K2의 해시값이 같은 경우
  - 즉, $h(K1) = h(K2)$ 인 상황
    - 예) 키가 K1인 아이템이 이미 HT에 있는데,
      K1과 같은 해시값 K2를 갖는 아이템을 넣으려고 한다면?
- Collision Resolution
  - Collision이 발생 했을 때 어떻게 해결할 것인가?
    - 예) h(K2)에서 collision이 발생하면, HT[h(K2) + 1]에 값을
      저장한다. 찾을 때에도 HT[h(K2)+1]을 살펴본다.

# Overview

- Hashing
- **Hash Functions**
- Collision Resolution
- Analysis of Closed Hashing

# Hash Functions

- 어떤 해시 함수가 좋은 hash function일까?
  - Collision이 최소로 발생하는 함수가 좋은 hash function
  - 즉, 들어오는 데이터에 상관 없이 모든 slot에 고르게 할당하는 함수가 좋은 hash function
- M=16이고, h(K) = K % 16이라면, h(K)는 좋은 hash function일까?
  - 만일 키가 모두 짝수라면?
  - 만일 키의 하위 4개 비트가 모두 동일하다면?

# Hash Functions

- 문제점 1) 만일 키가 모두 짝수라면, Hash table의 절반은 사용되지 않는다.
  - M값을 소수(prime number)로 정하면 Hash table의 slot들을 모두 활용하는데 도움이 된다.
  - Why?
    - 모든 키가 어떤 정수 c의 배수라고 하자 (즉, c * i),
    - 만일 M과 c가 서로소이면 c * i (i=0,1,…,M-1)는 모두 다른 slot에 할당된다!
    - 증명 (모순증명법)
      - Assume c * i = c * j (mod M) where i != j in [0, M-1].
      - Since c is relatively prime to M, this implies i-j is a multiple of M; but it cannot happen since i != j in [0, M-1], thus contradiction.

# Hash Functions

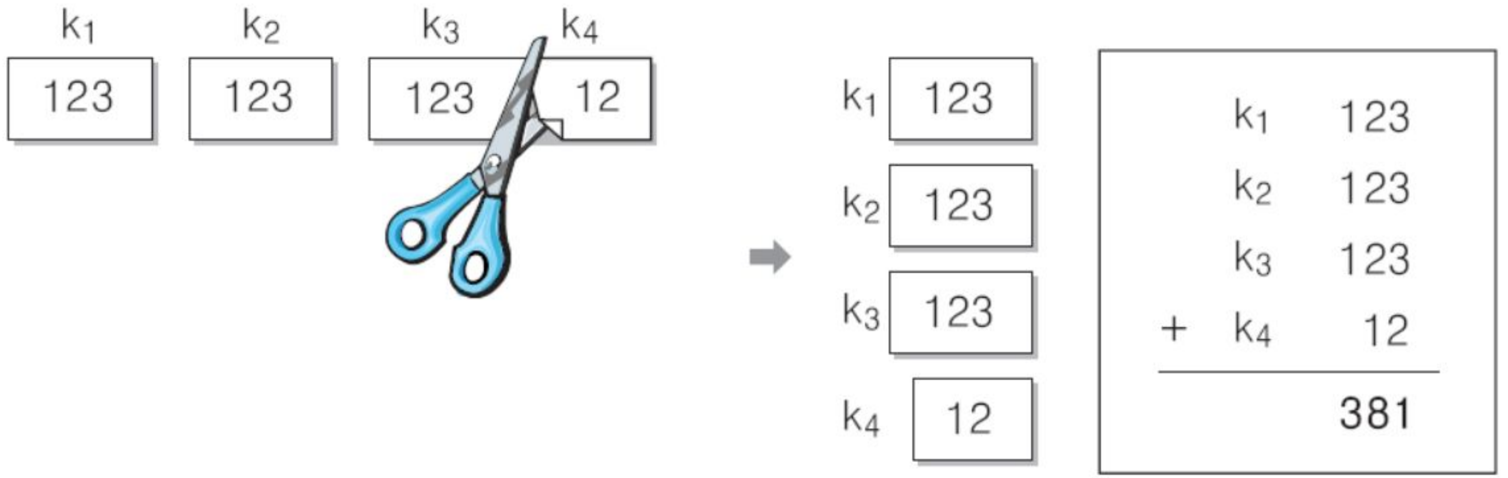
- 문제점 2) 키의 마지막 **4개 비트만 사용**하기 때문에, 입력 데이터에 따라서 **h(K)의 결과가 치우쳐져**있을 수 있다.
- 해결방법: 키의 모든 비트를 사용한다
  - 예 1) Mid-square method:
    key를 **제곱하여 가운데 r개의 비트**를 M으로 나눈 나머지 구하기

```
                 00110101 10100111
X                00110101 10100111
----------------------------------
00001011001111101001001011110001
```

(가운데 8비트 11101001 강조)

# Hash Functions

- 문제점 2) 키의 마지막 **4개 비트만 사용**하기 때문에, 입력 데이터에 따라서 **h(K)의 결과가 치우쳐져**있을 수 있다.
- 해결방법: 키의 모든 비트를 사용한다
  - 예 2) folding approach: **t개씩 비트를 잘라서 합**한 후 M으로 나눈 나머지 구하기. 짝수번째 조각을 **뒤집어서 합**할 수도 있음



# Overview

- Hashing
- Hash Functions
- **Collision Resolution**
- Analysis of Closed Hashing

# Collision Resolution

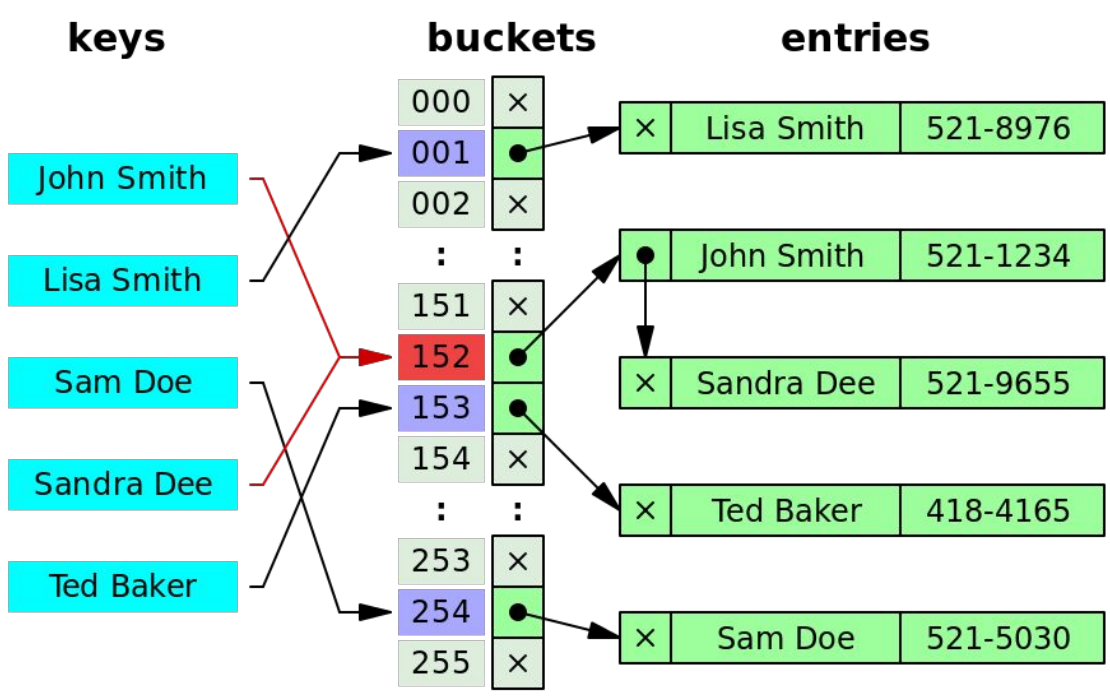
- Collision은 피하기 힘들다
  - 보통 키의 범위가 slot의 수보다 많기 때문에 (a.k.a. **비둘기 집의 원리 (Pigeonhole principle)**)
  - **생일 문제 (Birthday paradox):** 34명의 자료구조 수강생 중에 생일이 같은 두 사람이 존재할 확률은?

# Collision Resolution

- 아이템을 넣으려는데 Collision이 발생한 경우, 어떻게 하면 좋을까?
- Collision resolution techniques
  - Open hashing (a.k.a. separate chaining)
  - Closed hashing (a.k.a. open addressing)

# Open hashing

- Open hashing (a.k.a. separate chaining)
  - Hash Table의 각각의 slot에 Linked List를 연결
  - 삽입시 Collision이 발생하면 해당 slot의 Linked List에 추가



# Closed Hashing

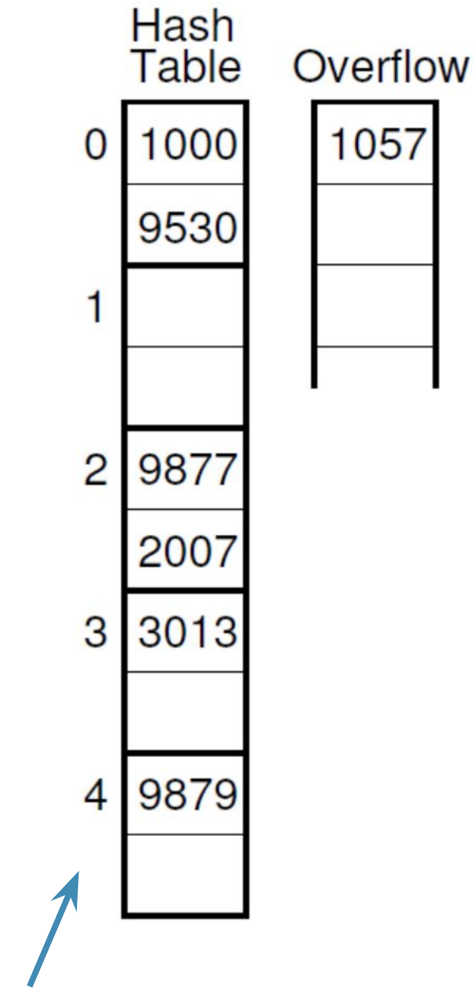
- Closed hashing (a.k.a. open addressing)
  - 삽입시 Collision이 발생하면 hash table의 다른 slot에 아이템을 저장

- 예: Bucket Hashing, Linear Probing, …

# Bucket Hashing

M = 10, B = 5
h(K) = K mod 5

- hash table의 slot들을 여러 bucket으로 그룹화
  - M slot들이 B개의 bucket으로 나눔
    - 각 buket은 M/B개의 slot을 가짐
  - Hash function (key → bucket 수)은 아이템을 bucket의 첫번째 slot에 넣음.
  - 만약 차있으면? bucket의 다음 slot에 넣음
  - bucket이 가득차면? 별도의 bucket (overflow bucket)에 넣음

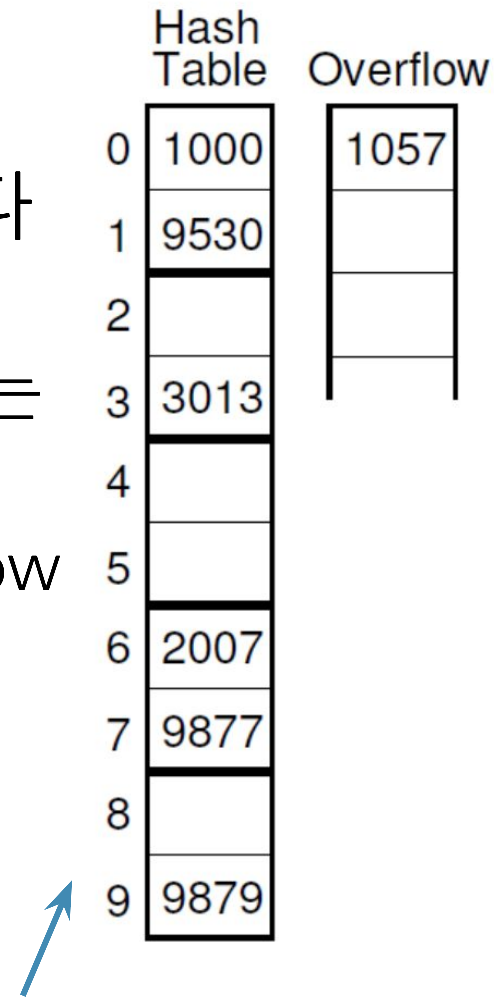



넣은 순서: 9877, 2007, 1000, 9530, 3013, 9879, 1057

# Bucket Hashing

M = 10, B = 5
h(K) = K mod 10

- Bucket Hashing의 변형: 마치 bucket이 없는 것처럼 hashing을 한다
  - slot이 비어있으면? 넣는다
  - slot이 차있으면? 해당 bucket의 비어있는 slot을 찾아 넣는다
  - bucket의 모든 slot이 차있으면? overflow bucket에 넣는다




넣은 순서: 9877, 2007, 1000, 9530, 3013, 9879, 1057

# Bucket Hashing

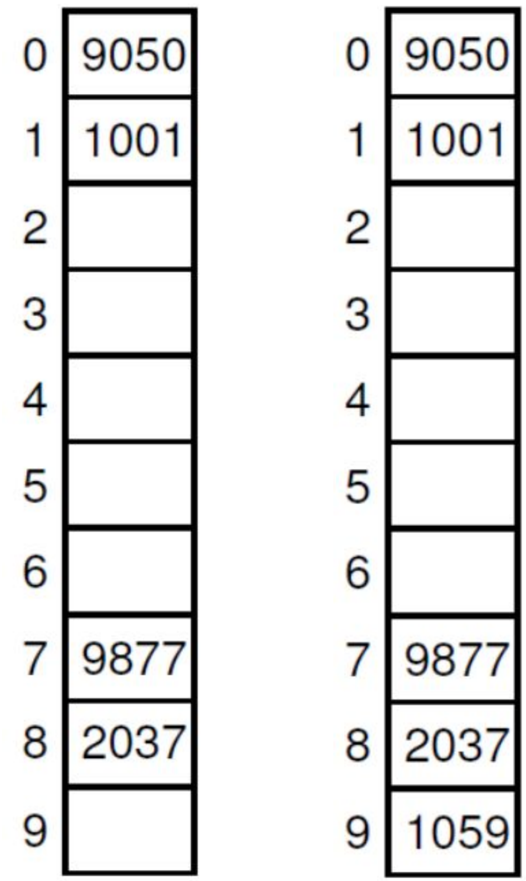
- Bucket hashing vs open hashing?
  - Bucket hashing에서 collision이 더 많이 발생
    $\Rightarrow$ 아이템 탐색 시간이 더 길어질 수 있음
  - Bucket hashing은 추가 공간을 덜 차지함

- Bucket Hashing의 한계
  - hash table에 빈 공간이 많이 남아있는 경우에도 bucket이 가득차면 overflow bucket에 값을 넣음

# Linear Probing

- Bucket 없이 closed hashing하기: 삽입시 collision이 발생하면 hash table의 모든 slot중에 빈 곳을 찾아 넣음
  - 빈 slot 찾기: 비어있는 slot이 나올 때까지 다음 slot으로 이동
  - h(K) + p(K, i)
  - p(K, i) = i
  - i는 충돌 발생 횟수

M = 10
h(K) = K mod 10




(a) (b)

1059 추가 삽입

넣은 순서: 1001, 9050, 9877, 2037

# Linear Probing

- Linear probing의 문제점
  - Primary clustering: 채워져있는 slot들이 뭉쳐짐
    → 남은 slot들에 할당될 확률이 서로 달라짐
  - 예) 오른쪽 그림에서, 삽입 연산시 각 slot에 값이 할당될 확률은?

| | |
|---|---|
| 0 | 9050 |
| 1 | 1001 |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | 9877 |
| 8 | 2037 |
| 9 | 1059 |

(b)

# Improving Linear Probing

- 개선 방법 1) 상수배 건너뛰기
  - P(K, i) = ci
  - c는 M과 서로소여야함 (why?)
  - 한계: 여전히 primary clustering 존재
- 개선 방법 2) Pseudo-random probing
  - P(K, i) = Perm[i-1]
  - Perm: 1부터 M-1사이의 값이 섞여있는 길이 M-1의 배열
- 개선 방법 3) Quadratic probing
  - $P(K, i) = c_1 i^2 + c_2 i^2 + c_3$

# Improving Linear Probing

- Pseudo-random probing과 Quadratic probing은 모두 secondary clustering 문제가 있음
  - Secondary clustering: 해시함수가 특정 slot에 대하여 cluster를 형성함
    - 키의 해시값이 같은 아이템들은 모두 같은 probe sequence를 가지기 때문에 발생
  - Secondary clustering 발생 원인? P(K, i)가 i만의 함수라서 → K도 사용하면 해결
    - 예) Double hashing: $P(K, i) = i * h_2(K)$
    - Pseudo-random/quadratic probing에 double hashing을 적용 가능

# Overview

- Hashing
- Hash Functions
- Collision Resolution
- **Analysis of Closed Hashing**

# Analysis of Closed Hashing

- Load factor α = N/M (N: 아이템 수, M: Hash table 크기)
- 넣는데 걸리는 평균 탐색 횟수
  - 첫번째 탐색만에 넣을 확률 P(1) = $1 - \frac{N}{M}$
  - 두번째 탐색만에 넣을 확률 P(2) = $\frac{N}{M}\left(1 - \frac{N-1}{M-1}\right)$
  - i번째 탐색만에 넣을 확률 P(i) = $\frac{N(N-1)\cdots(N-i+2)}{M(M-1)\cdots(M-i+2)}\left(1 - \frac{N-i+1}{M-i+1}\right)$
  - 평균 탐색 횟수 = $\sum_{i=1}^{N+1} i \cdot P(i)$

$$= 1 + \frac{N}{M} + \frac{N(N-1)}{M(M-1)} + \cdots \approx 1 + \sum_{i=1}^{\infty}(N/M)^i = \frac{1}{1-\alpha}$$

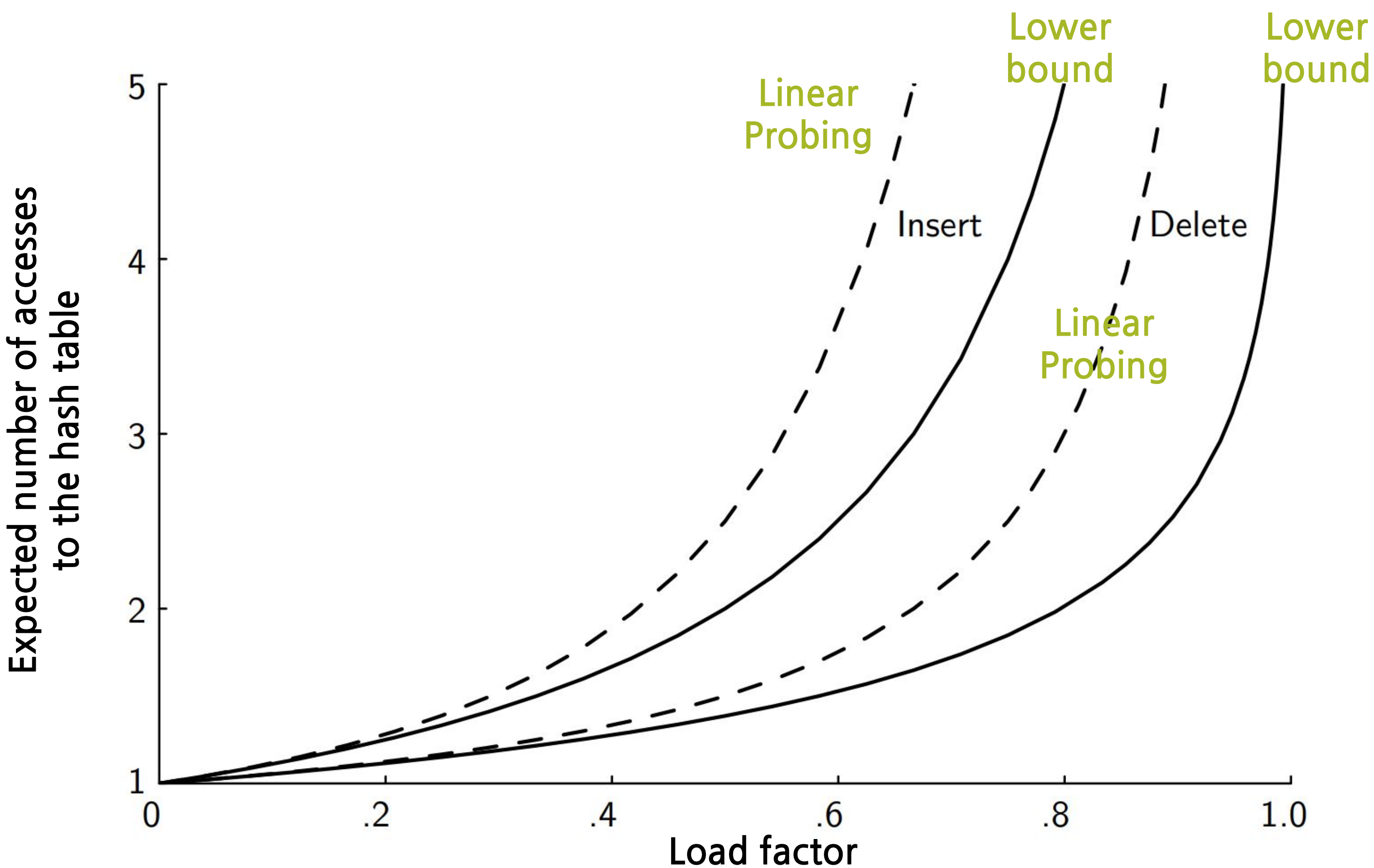
# Analysis of Closed Hashing

- 찾는데(지우는데) 걸리는 평균 탐색 횟수
  - 어떤 아이템을 찾을 때 걸리는 탐색 횟수는 그 아이템을 넣는 시점에 걸렸던 탐색 횟수와 같음

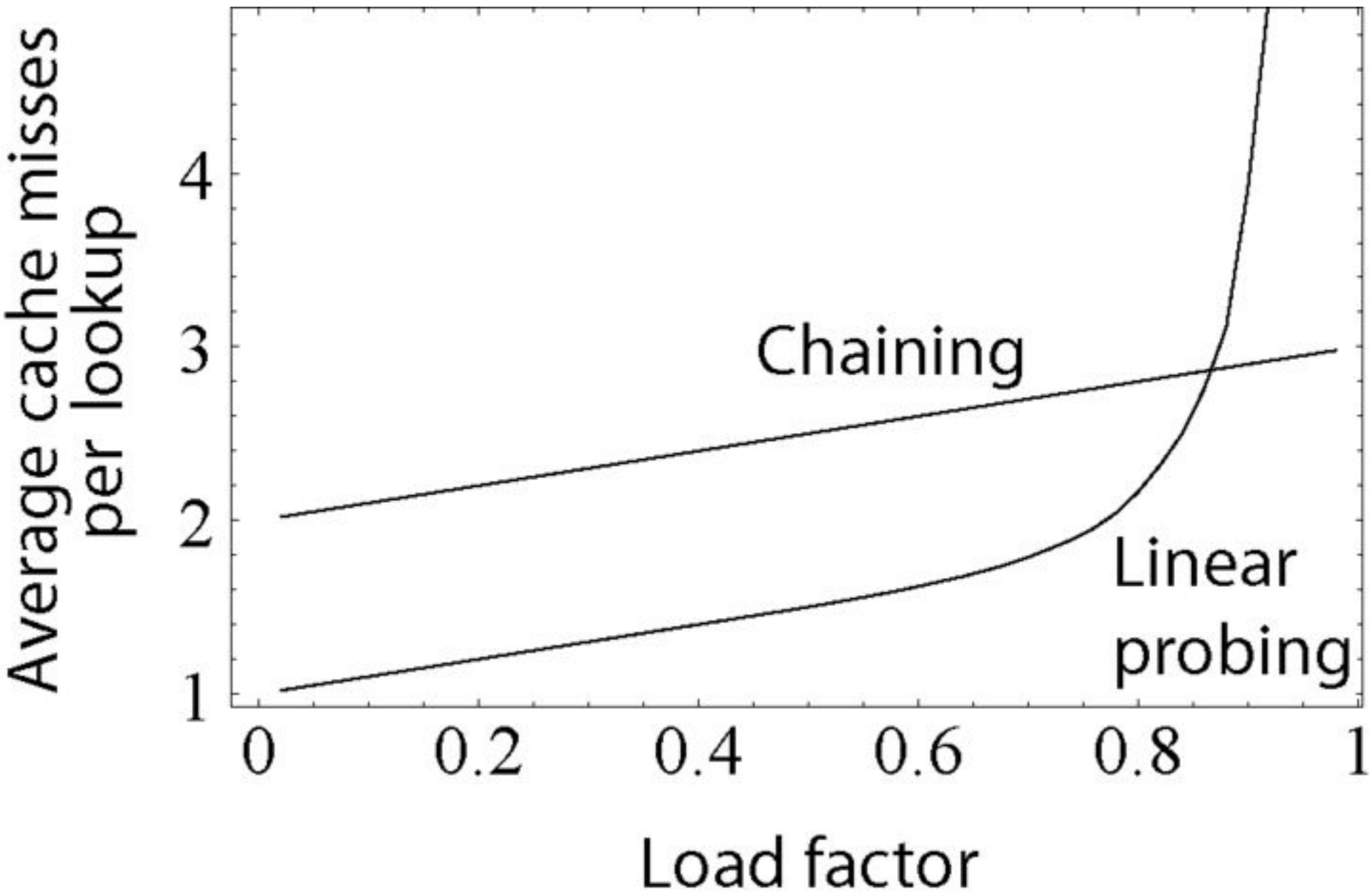
$$\frac{1}{\alpha}\int_0^{\alpha}\frac{1}{1-x}dx = \frac{1}{\alpha}\log_e\frac{1}{1-\alpha}.$$

- Linear Probing의 경우
  - 삽입: $\frac{1}{2}(1+1/(1-\alpha)^2)$
  - 삭제: $\frac{1}{2}(1+1/(1-\alpha))$

# Performance of Closed Hashing



# Performance of Closed Hashing



# Overview

- Hashing
- Hash Functions
- Collision Resolution
- Analysis of Closed Hashing

# Questions?