알아도 도움안되는 얕은 개발 지식들 (이건 어떻게 구현되었을까?)

charsyam@naver.com

Who am I?

- (현) 레몬트리 잡부(?)
- (전) 위버스 데이터 엔지니어
- (전) Udemy 데이터 엔지니어
- (전) 카카오 백엔드 엔지니어
- (전) 네이버 백엔드 엔지니어

오늘의 목차

- 왜 expire 는 제 시간에 지워지지 않을까?
- 없다고 하면 없는데, 있다고 하면 없을 수도 있는 자료구조는 언제 쓰는거지?

왜 Expire 는
제 시간에 지워지지 않을까?

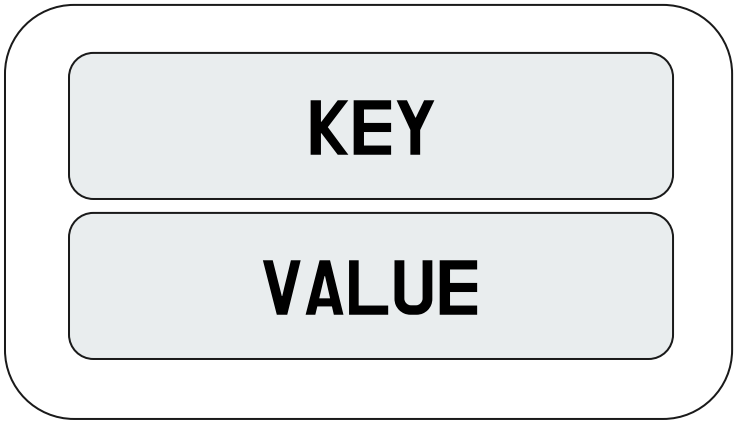
Expire

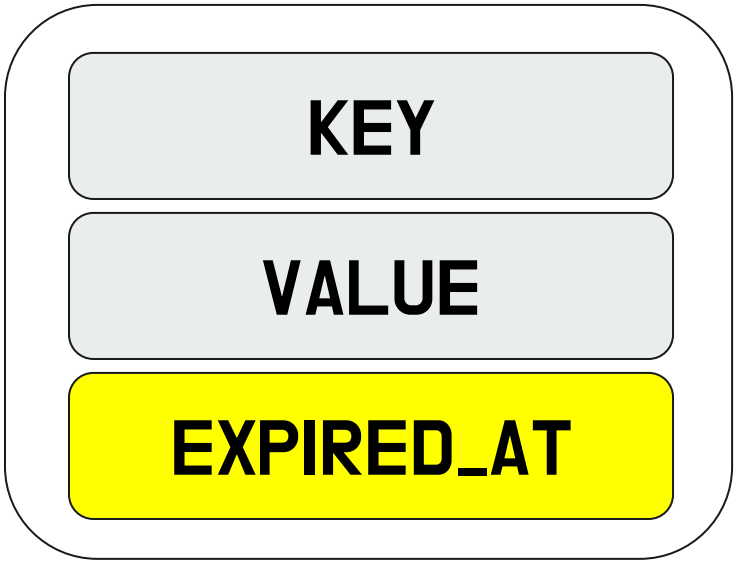
Expire 는
특정 시간에 데이터가 지워지게
하는 것.

Expire

Redis, Memcached 같은
캐시 솔루션에서 많이 사용됨.

한번 만들어볼까요?


KEY
VALUE


KEY
VALUE
EXPIRED_AT

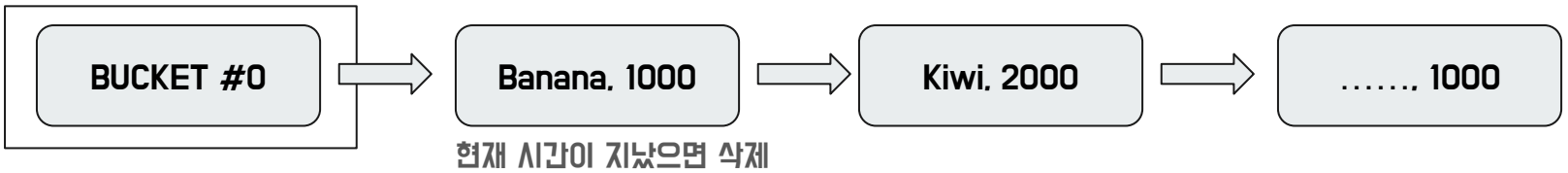
어떻게 구현하실껀가요?

간단한 생각

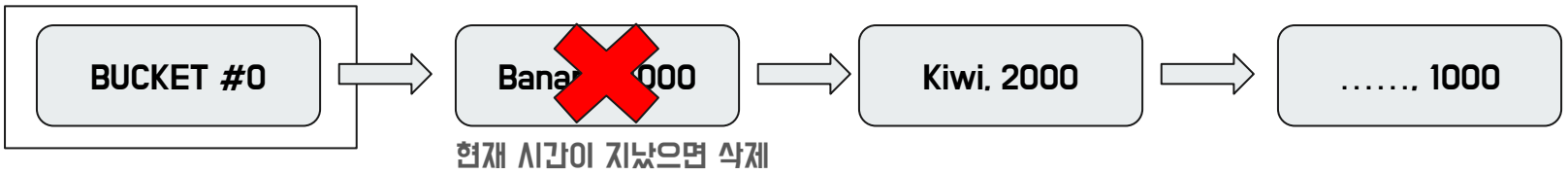
간단한 생각

일정 시간마다 체크

일정시간 마다 체크



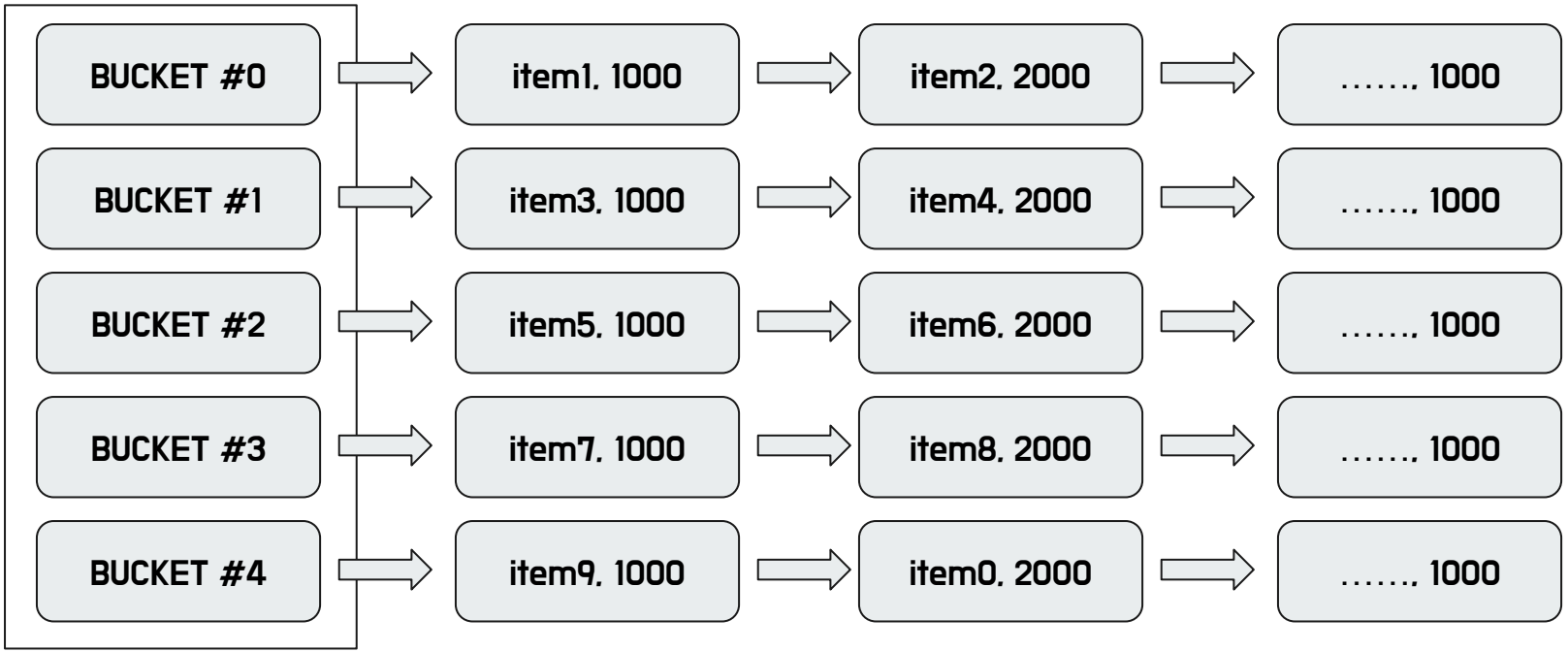
일정시간 마다 체크



한계

체크해야 하는 데이터가 얼마나 될까요?

데이터가 엄청 많으면?


BUCKET #0
item1. 1000
item2. 2000
……. 1000
BUCKET #1
item3. 1000
item4. 2000
……. 1000
BUCKET #2
item5. 1000
item6. 2000
……. 1000
BUCKET #3
item7. 1000
item8. 2000
……. 1000
BUCKET #4
item9. 1000
item0. 2000
……. 1000

한계

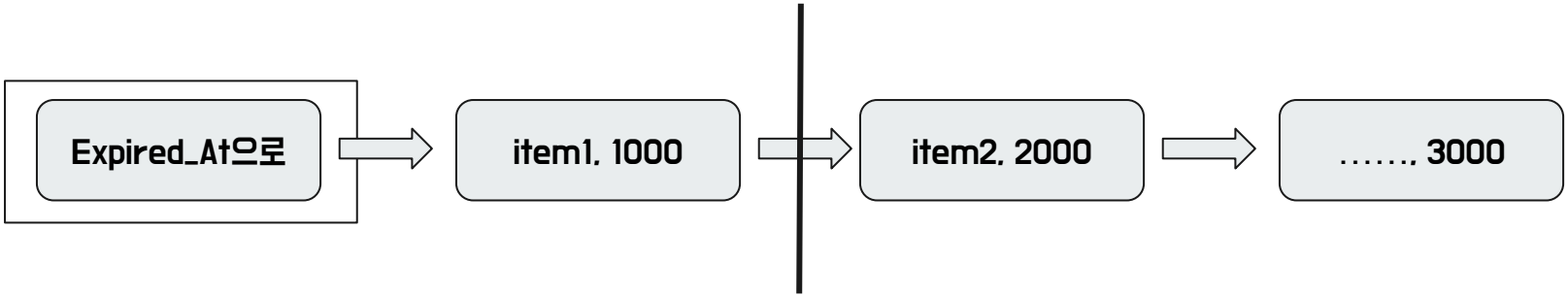
어떤 주기로 체크를 해야 할까?

조금 복잡한 생각

조금 복잡한 생각

Expired_At 으로
정렬해두면?

Expired_At 으로 정렬



특정 노드 이상으로는 더 볼 필요가 없다.
뒤는 Expired_At 해당 노드 보다 많이 남았기
때문에

조금 복잡한 생각

그런데 이러면...
검색이 느려지지 않을까?

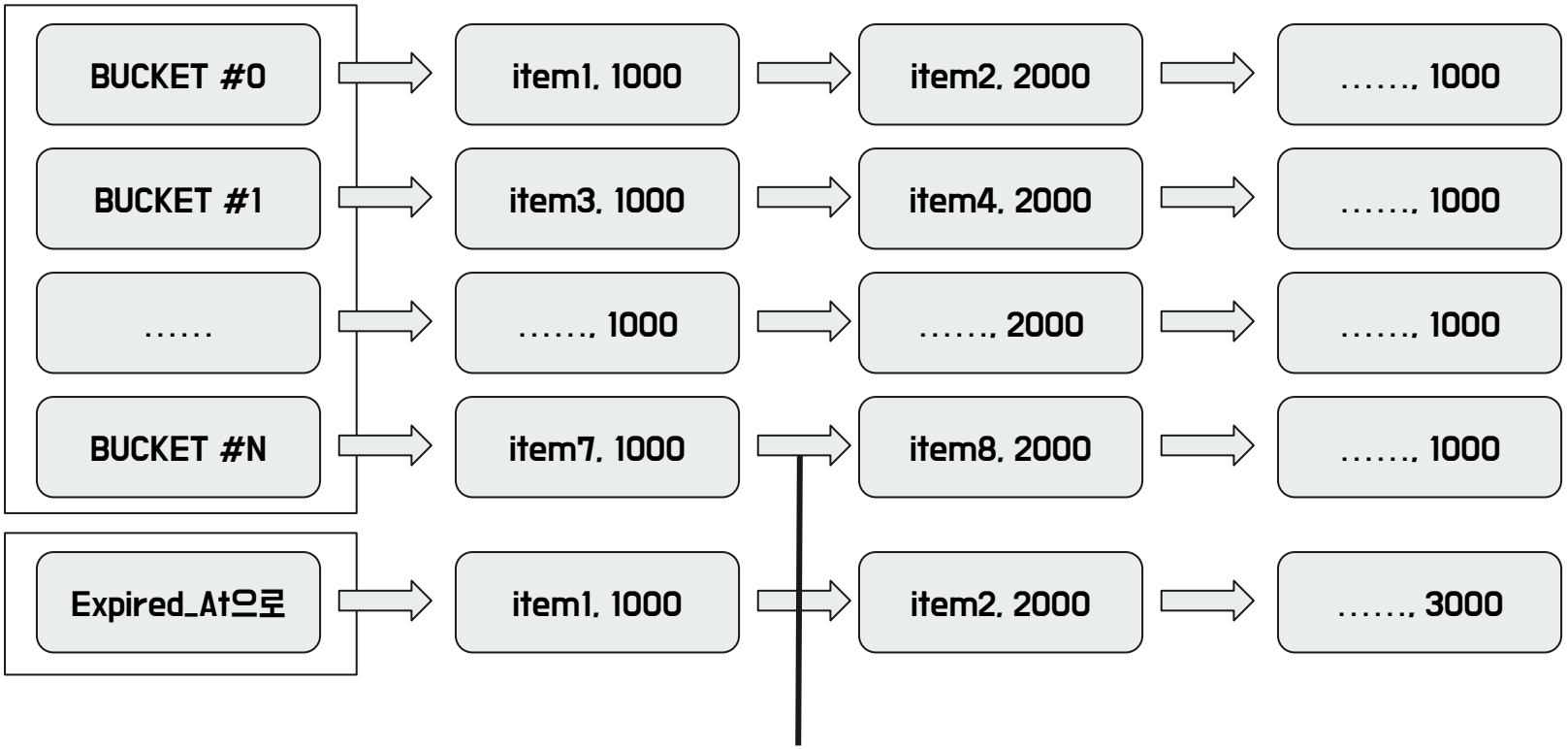
조금 더 복잡한 생각

두 개를 다 유지하자.

두 개를 다 유지하자.

1. Expired_At 으로 정렬된 노드
2. KEY를 쉽게 찾기 위한 해시 노드

데이터가 엄청 많으면?


BUCKET #0
item1. 1000
item2. 2000
……. 1000
BUCKET #1
item3. 1000
item4. 2000
……. 1000
……
……. 1000
……. 2000
……. 1000
BUCKET #N
item7. 1000
item8. 2000
……. 1000
Expired_At으로
item1. 1000
item2. 2000
……. 3000

메모리는 더 많이 사용하게 된다.

두 군데를 관리해야 함.

그런데... 그래도 Expired가 비슷한 데이터가 엄청 많으면?

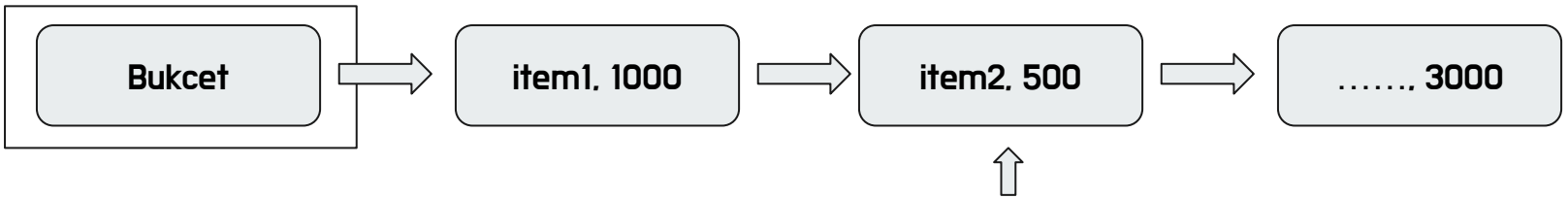
체크 속도는 결국 O(N)

조금 더어~ 복잡한 생각

안 지우면 안될까?

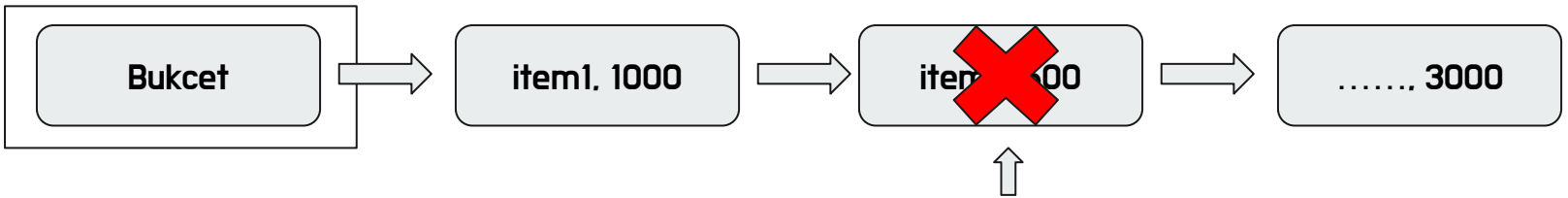
지우지 않더라도, 접근할 일이 생기면, 그때 지워버리자?

Expired_At 으로 정렬



item2에 접근했을 때 현재 시간이 Expired_At 을
넘었다면. 이 때. 지우고 없다고 리턴한다.

Expired_At 으로 정렬



item2에 접근했을 때 현재 시간이 Expired_At 을
넘었다면. 이 때. 지우고 없다고 리턴한다.

이거면 될까요?

한계

Expired_At 이 지난 Key 가
많아도 메모리는 해제되지 않는다.

마지막 생각

한계

세 가지를 다 쓰자.

마지막...

1. 기본적으로 일정 시간마다,
   Expired_At 이 지난 KEY를
   지운다.(모두 지우지 않고 일부만)

마지막...

2. 속도를 위해서 Expired_At 으로 정렬된 리스트를 하나 더 유지한다.

마지막...

3. 키에 접근했을 때 이미 Expired_At 이 지워지면 이 때 삭제한다.

여러분의 아이디어를
추가해 보세요.

없다고 하면 진짜 없는데
있다고 하면 없을 수 있는
자료구조

이런 자료구조가
왜 필요할까?

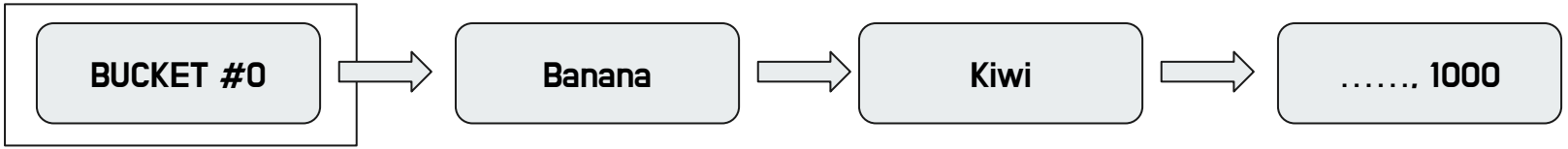
먼저, 있다고 하면 항상 있고
없다고 하면 항상 없는
자료구조를 만들어 봅시다.

어떻게 만들어야 할까요?

간단한 생각

모든 데이터를 순회하면 됩니다.

모든 데이터를 순회

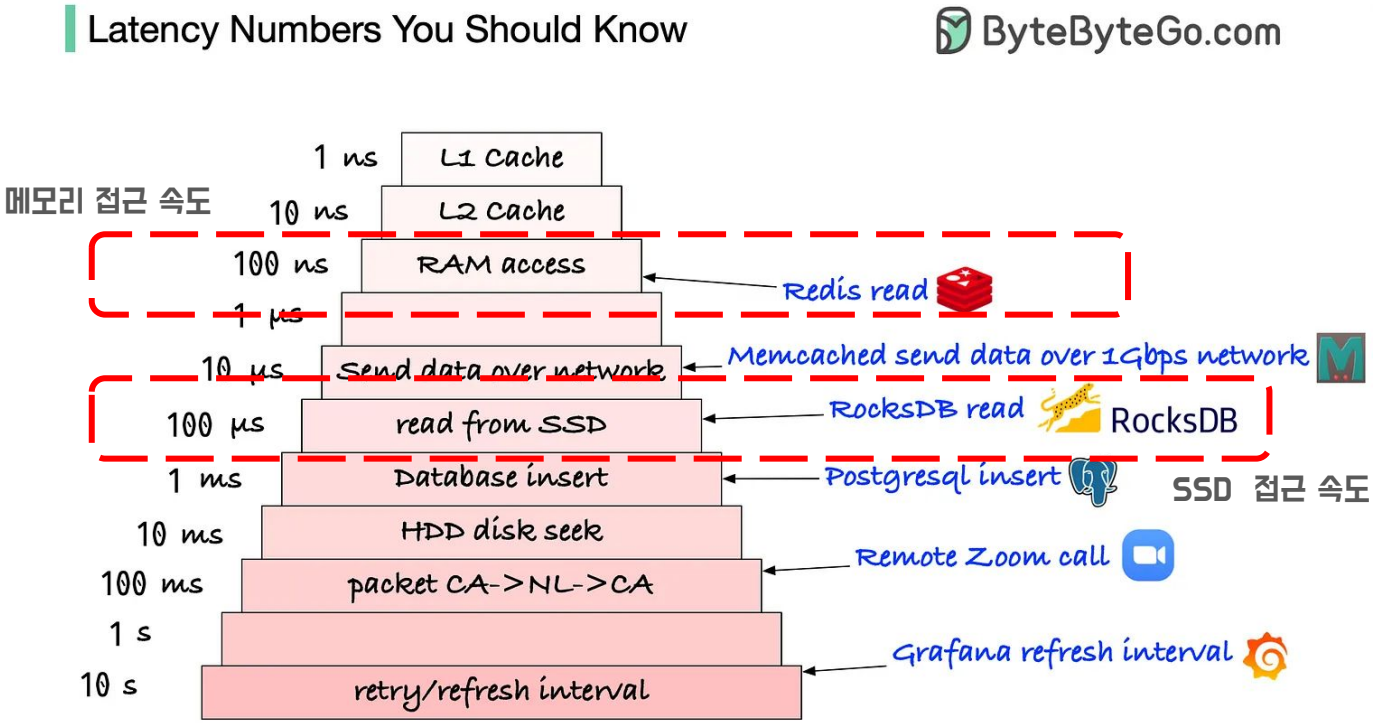


끝까지 다 찾아보면, 데이터가 있는지 없는지
정확하게 알 수 있다.

그 전에...
이걸 어디에 쓸까요?

Which latency numbers should you know?

Reference: https://blog.bytebytego.com/p/ep22-latency-numbers-you-should-know



메모리 접근 속도가
디스크(SSD) 접근속도 보다
1000배 정도 빠릅니다.

위의 자료구조는
디스크에 접근하는 것을
줄이는게 목표

아이템이 한 100만개 정도 있으면 메모리가 얼마나 필요할까?

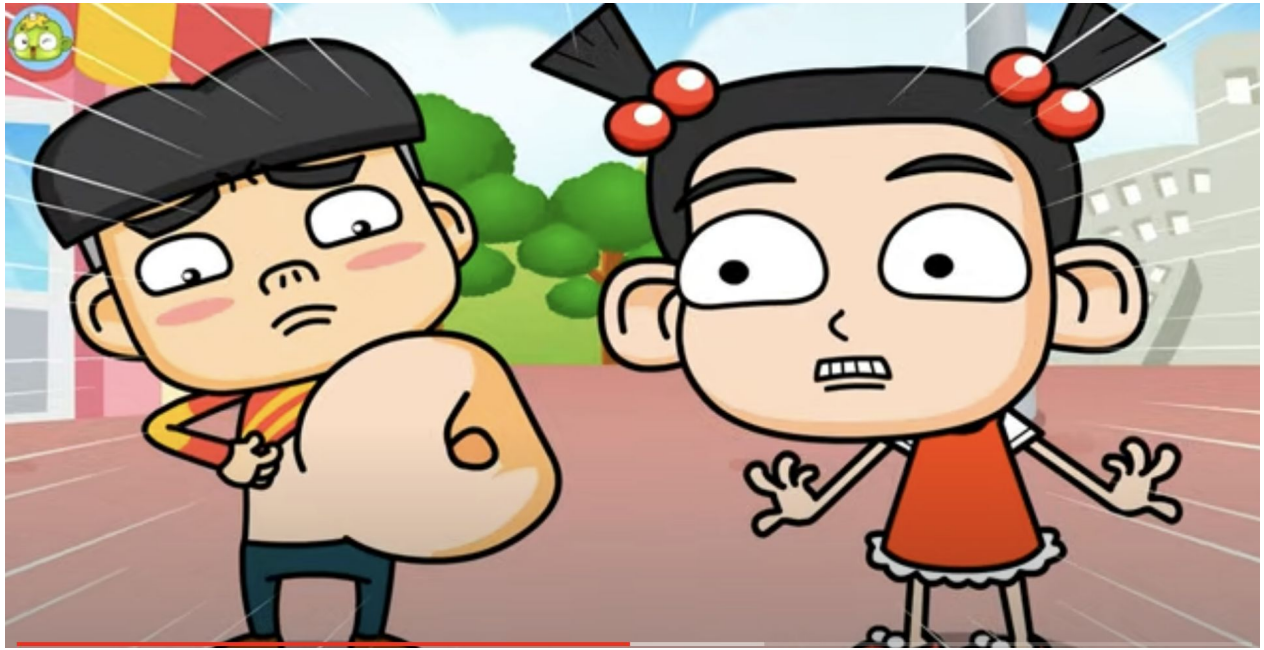
단순계산

4K * 100만 = 4GB

아이템이 1000만개? (40G)
아이템이 1억개? (400G)

그런데 디스크에는
훨어얼얼얼 많은 아이템이 존재

메모리를 너무 많이 쓸 수가 없다.



메모리는 아끼면서
이런걸 체크할 수 있는
방법이 있을까?


MEMORY
ACCURACY

조금 복잡한 생각

hash 를 써보자.
crc16 이나 crc32?

crc16을 쓰면
65,535 의 배열이 필요하다.

BYTE 말고 BIT를 쓰면
공간은 1/8
8192 BYTE

65,535는 너무 작으니
CRC32는
4,294,967,295(42억)

BYTE 말고 BIT를 쓰면
공간은 1/8
536,870,912 BYTE (512MB)

그런데 hash가 충돌이 일어나면?

hash(x) = hash(y)

x != y 일때

우리는 충돌이라고 한다.

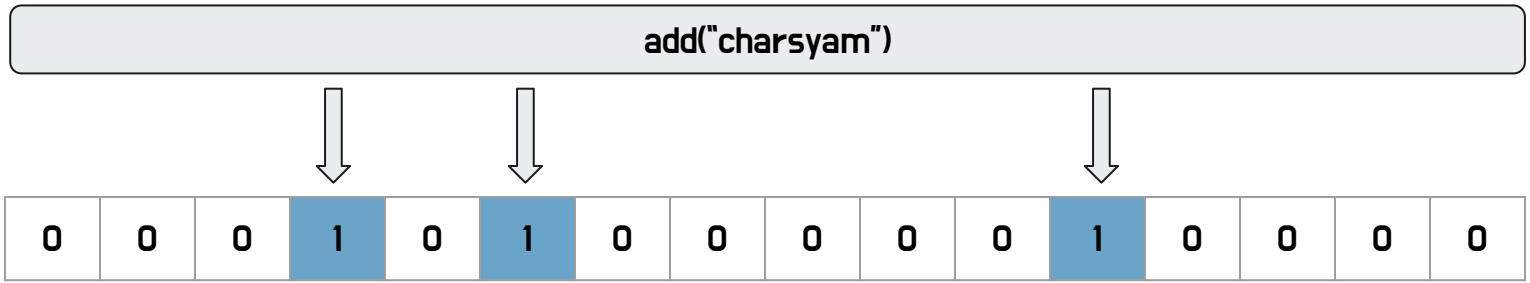
없다고 하면 진짜 없는데
있다고 하면 없을 수 있는
자료구조
= BloomFilter

BloomFilter 는 어떻게 동작하는가?

- BloomFilter 는 bitarray
- N개의 hash를 선택합니다.
- 예를 들어, crc, murmur, md5 를 쓴다면
  - hash(key, 0)는 crc(key)
  - hash(key, 1)은 murmur(key)
  - hash(key, 2)는 md5(key)

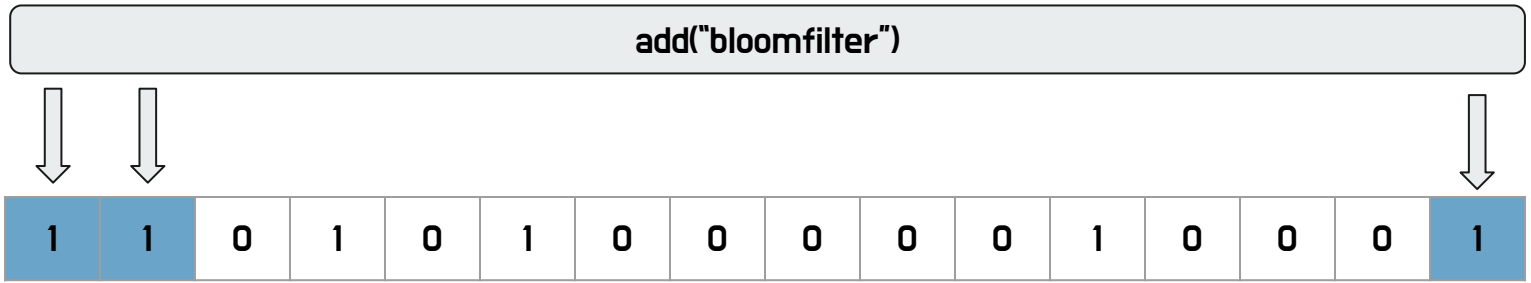
BloomFilter 의 동작 #1 - Add

- 16bit의 array를 사용한다고 가정합니다.
- hash("charsyam", 0) % 16 = 3
- hash("charsyam", 1) % 16 = 5
- hash("charsyam", 2) % 16 = 11



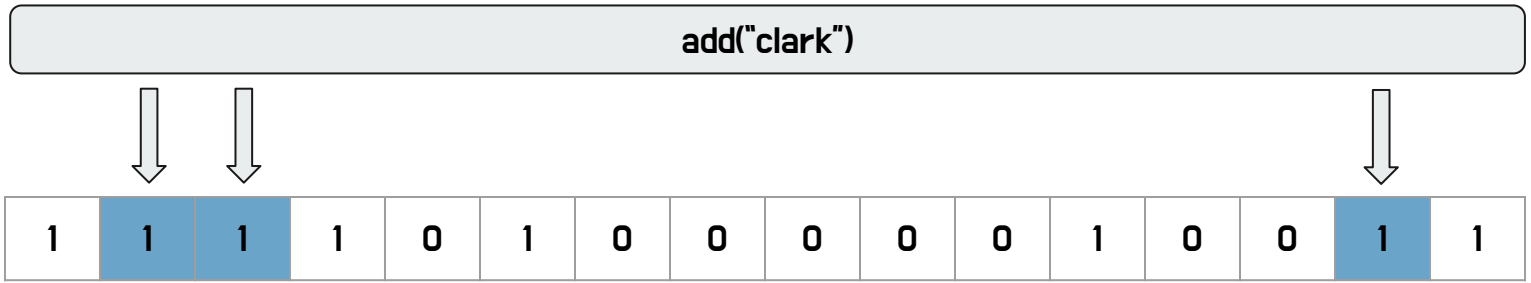
BloomFilter 의 동작 #2 - Add

- hash("bloomfilter", 0) % 16 = 0
- hash("bloomfilter", 1) % 16 = 1
- hash("bloomfilter", 2) % 16 = 15



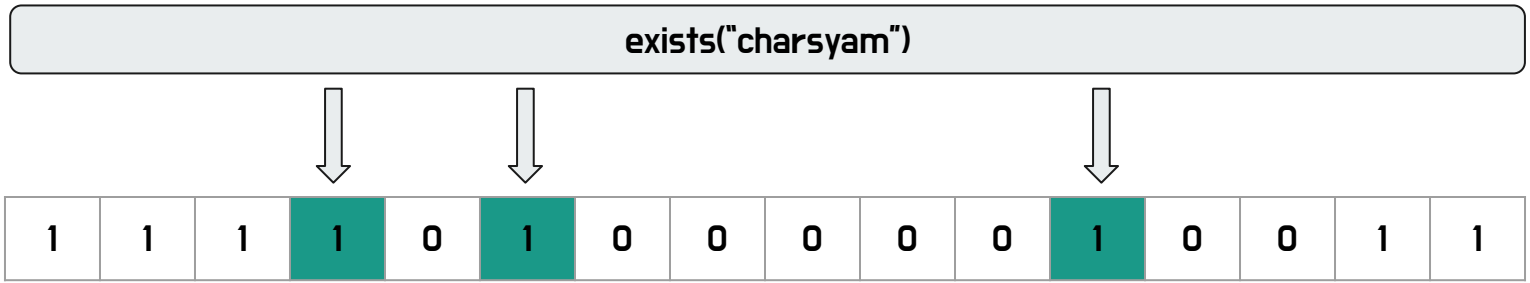
BloomFilter 의 동작 #3 - Add

- hash("clark", 0) % 16 = 1
- hash("clark", 1) % 16 = 2
- hash("clark", 2) % 16 = 14



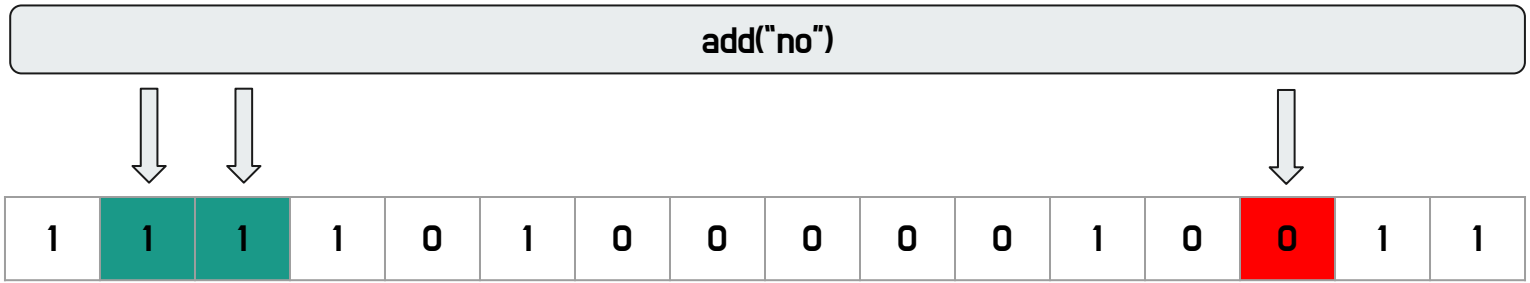
BloomFilter 의 동작 #4 - Exist

- exists 의 동작. 확인할 3개의 비트가 모두 1로 채워져 있으면 존재한다고 알림.
- hash("charsyam", 0) % 16 = 3
- hash("charsyam", 1) % 16 = 5
- hash("charsyam", 2) % 16 = 11



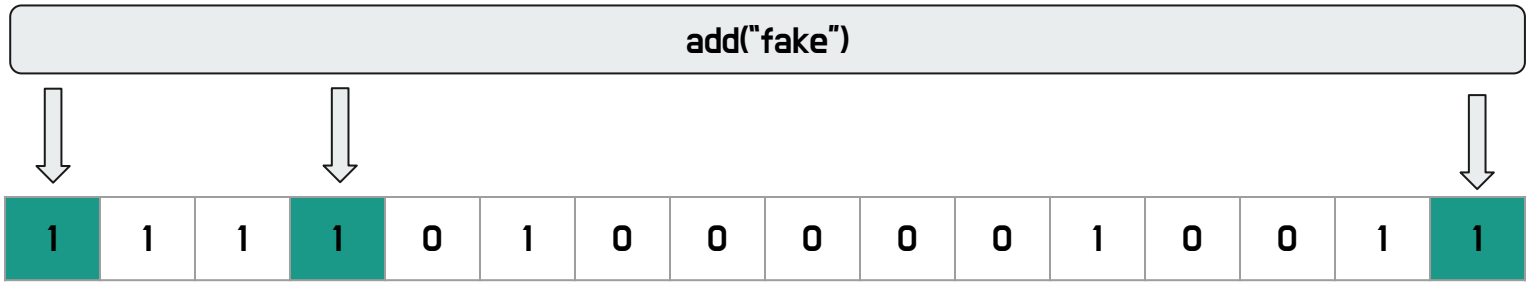
BloomFilter 의 동작 #5 - Exists(없는 경우)

- no는 3개중에 2개만 1이므로 실패로 리턴한다.(모두 1이어야 한다.)
- hash("no", 0) % 16 = 1
- hash("no", 1) % 16 = 2
- hash("no", 2) % 16 = 13



BloomFilter 의 동작 #6 - Exists(False Positive)

- fake는 추가하지 않았지만 모든 bit가 1임. 있다고 알려주지만. 실제로 존재하지 않는다.
- hash("fake", 0) % 16 = 0
- hash("fake", 1) % 16 = 3
- hash("fake", 2) % 16 = 15



왜 N개의 hash 를 사용하는가?
충돌이 일어나도, 여러개가 동시에 일어나기는 어렵다.

N개의 hash가 모두 Set 되면
데이터가 존재한다고 생각한다.

없다고 하면 정확하게 없는 이유

N 개의 hash 의 결과가 모두 1이 여야만 존재
하나만 0이라도 존재하지 않다는게 보장.
False Negative 는 존재하지 않는다.

다른 아이템의 Hash 결과로
비트가 채워질 수 있다.

있다고 해도 없을 수 있는 이유.

N개의 hash 결과가 모두 1이라도
존재한다는 것은 보장하지는 못한다.
다른 item이 채운거일 수 있어서...
False Positive 가 존재하는 이유.

어디서 많이 쓸까?

크롬에서 스팸 IP를 확인하는 형태로 사용한다고 합니다.

HBase, Cassandra 같은
NoSQL에서도 많이 씁니다.

고민 #1
그냥 CRC32 쓰면 안되나요?
BIT로 계산하면 메모리 사용량은
더 적은데?

CRC32를 써도 된다.
똑같이 CRC32도 해당 BIT가 비어있으면... 없다.
그런데... 무조건 512MB가 필요함...
42억개를 항상 처리할 필요가 있을까?

BloomFilter 는 False Positive 를 조정하는 것으로
사이즈를 좀 더 조절이 가능하다.

고민 #2
그런데 왜 MySQL 이나 Postgresql 같은
일반 DBMS는 사용한다는 얘기를 못들어봤을까?

BloomFilter 에서 아이템 삭제가 가능할까요?

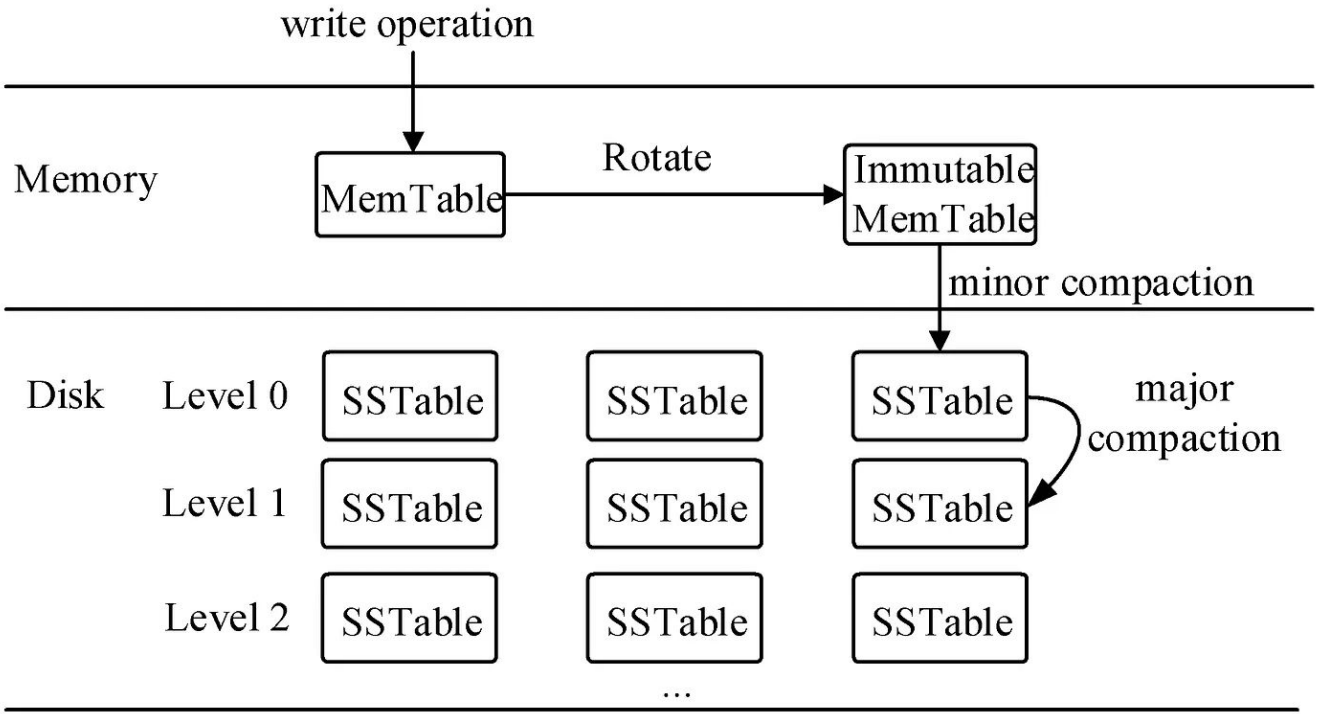
그런데 그럼 왜 HBase나 Cassandra 같은 NoSQL은 사용 가능할까요?

LSM(Log Structured Merge) Tree
라는 Append 형태의 자료구조를 쓰는 곳에서
사용하기에 좋습니다.

이렇게 존재하냐 아니냐를
Membership Query 라고 합니다.

BloomFilter 는 확률적 자료구조라고 합니다.
(Probabilistic Data Structures)

LSM Tree 는 다음 번에 ...



결론

- 알아도 당장 쓸 일이 없지만...
- 호기심을 키웁시다.
- 혹시아나요? 담에 이걸 직접 만들어야 할지?