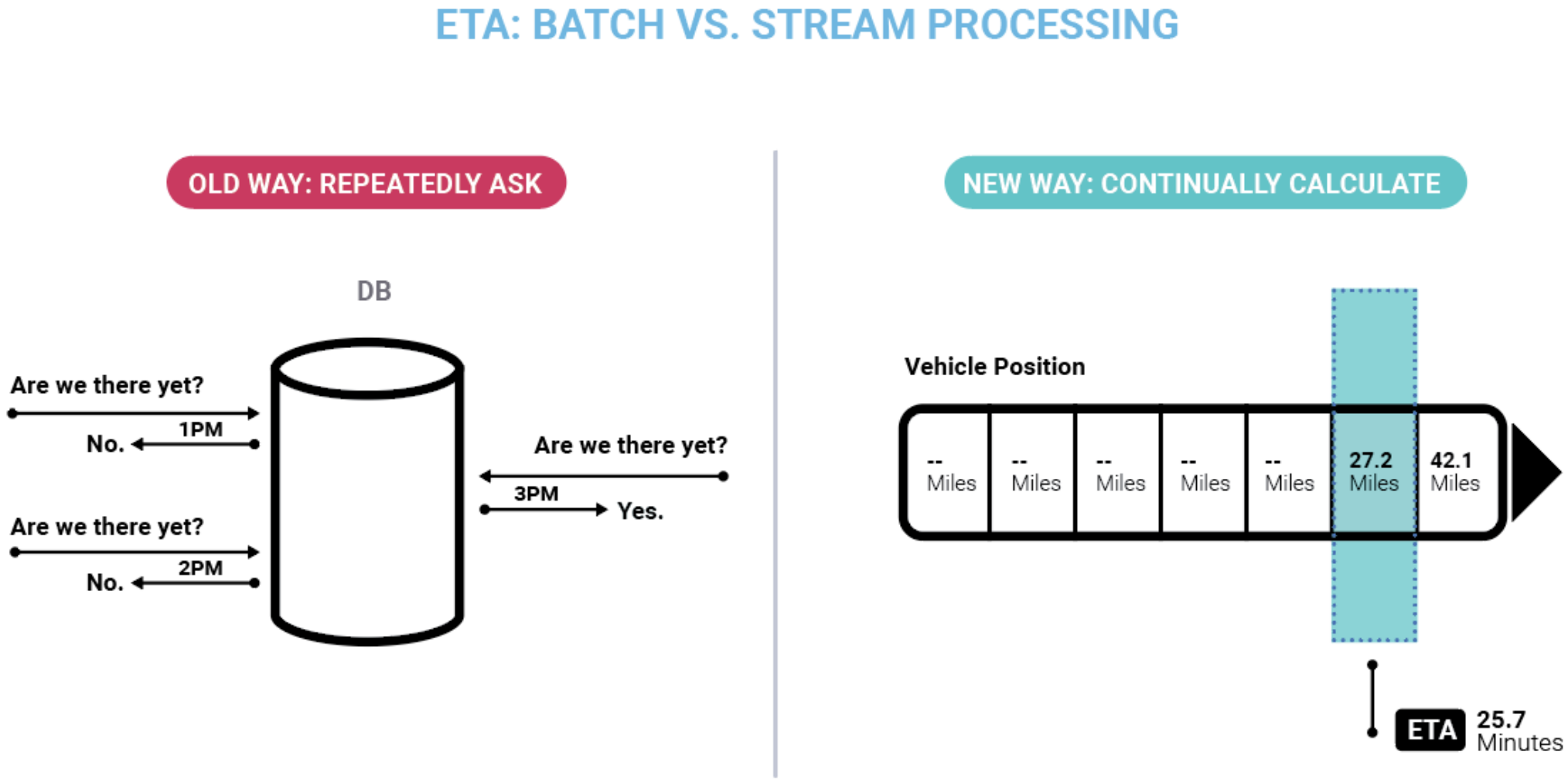
스트리밍 알고리즘(Streaming Algorithms): Data without a disk

빅데이터분석
컴퓨터AI공학부
천세진

강의목표

- 우리는 언제 데이터 집합이 언제 종료될지 모른다
  - 저장할 수가 없다
  - 반복적으로 접근하는 것은 비실용적이다
  - 데이터는 빠르게 시스템에 도착한다
  - 데이터베이스에 추가하는 것이 올바르지 않을 수도 있다
- 데이터가 디스크 저장에 적합하지 않으면,
- 일반화(Generalize)나 요약(Summarize)하는 것은 어떠한 가?

배치와 스트림 처리와의 차이



https://www.confluent.io/blog/every-company-is-becoming-software/

강의목표

- 우리는 언제 데이터 집합이 언제 종료될지 모른다
  - 저장할 수가 없다
  - 반복적으로 접근하는 것은 비실용적이다
  - 데이터는 빠르게 시스템에 도착한다
  - 데이터베이스에 추가하는 것이 올바르지 않을 수도 있다
- 데이터가 디스크 저장에 적합하지 않으면,
- 일반화(Generalize)나 요약(Summarize)하는 것은 어떠한 가?

구글 검색 쿼리

위성 이미지 데이터

텍스트 메세지

클릭 스트림

140Hz 빈도의 데이터 수집

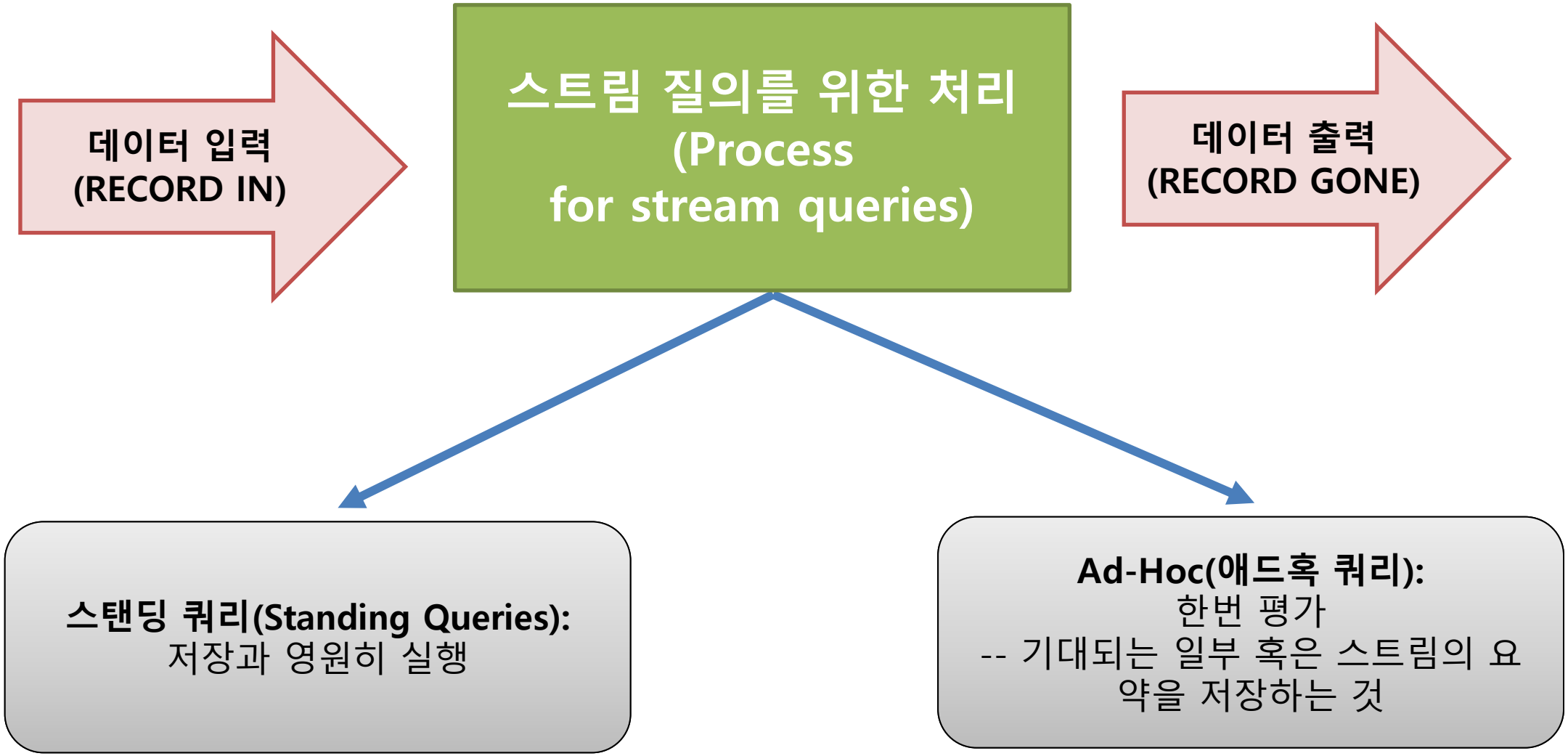
강의목표

- O(N)으로 데이터를 처리하고 싶다
  - 데이터 한번만 패싱(single pass)하면서 보는 수준

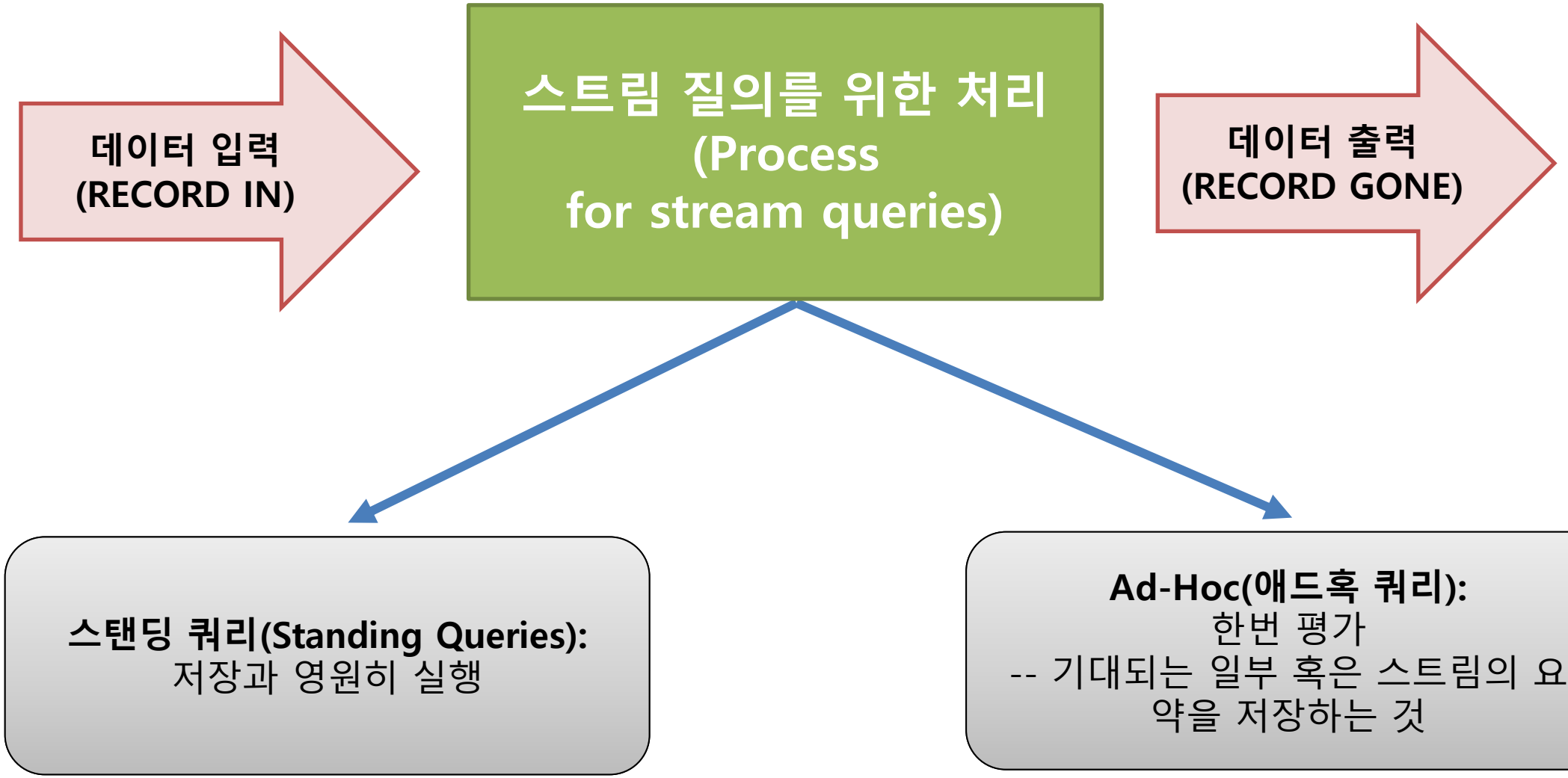
Streaming Topics

- 일반화된 스트림 처리 모델(General Stream Processing Model)
- 샘플링(Sampling)
- 조건에 따른 필터링(Filtering data according to a criteria)
- 구분 요소를 카운팅(Counting Distinct Elements)

스트림 처리 구조

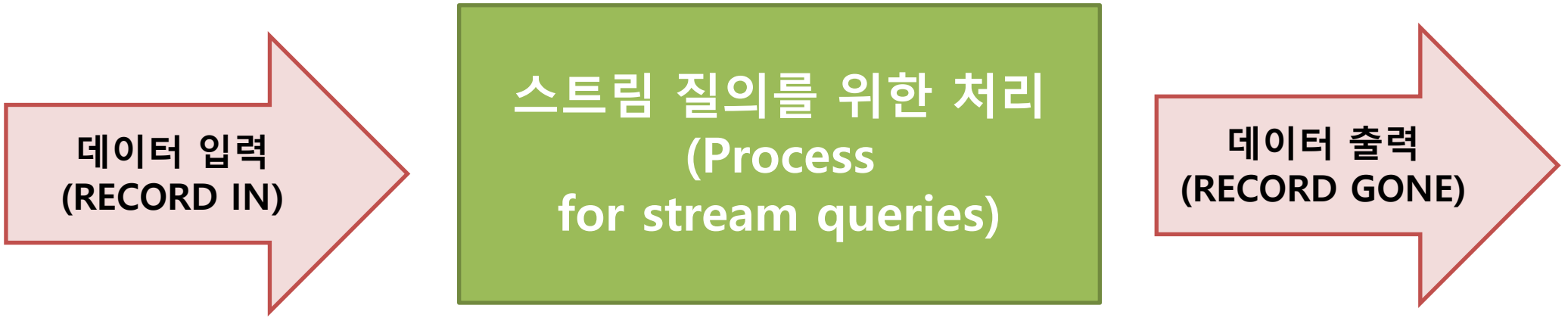

데이터 입력
(RECORD IN)
스트림 질의를 위한 처리
(Process
for stream queries)
데이터 출력
(RECORD GONE)
스탠딩 쿼리(Standing Queries):
저장과 영원히 실행
Ad-Hoc(애드혹 쿼리):
한번 평가
-- 기대되는 일부 혹은 스트림의 요
약을 저장하는 것

스트림 처리 구조



어떻게 우리는 이것을 다룰 수 있을까?
예로, 지금까지 본 값들의 평균은 무엇인가?

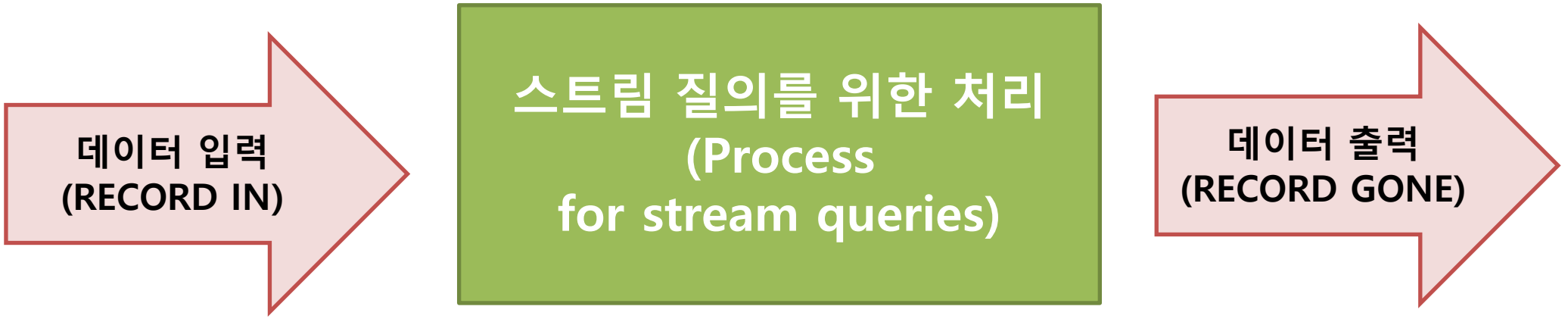
스트림 처리 구조



전형적인 데이터베이스 관리와 중요한 차이

- 입력은 시스템 스태프(staff)에 의해 제어되지 않습니다.
- 입력의 타이밍/비율은 대부분 알 수 없습니다

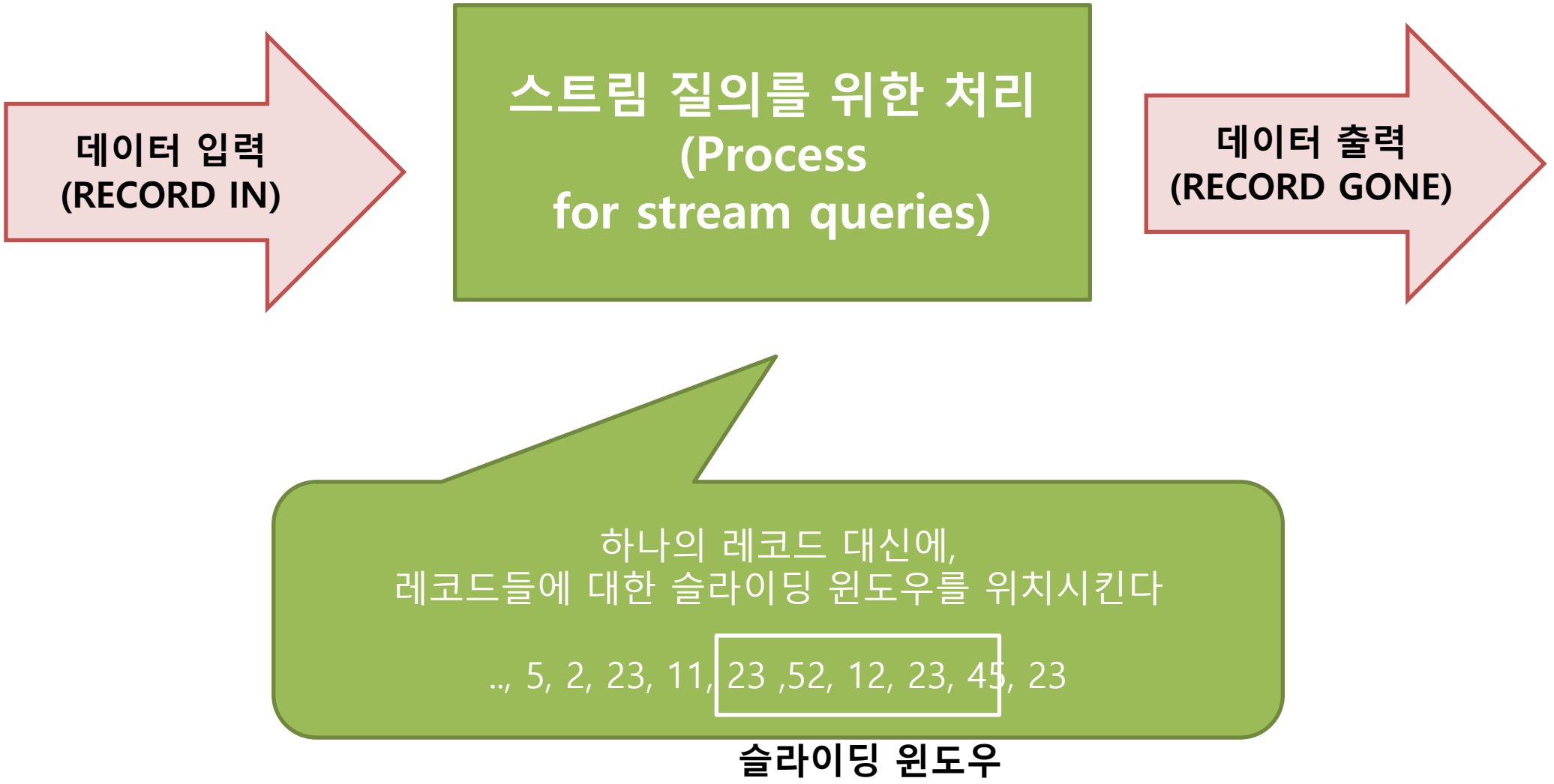
스트림 처리 구조



전형적인 데이터베이스 관리와 중요한 차이

- 입력은 시스템 스태프(staff)에 의해 제어되지 않습니다.
- 입력의 타이밍/비율은 대부분 알 수 없습니다

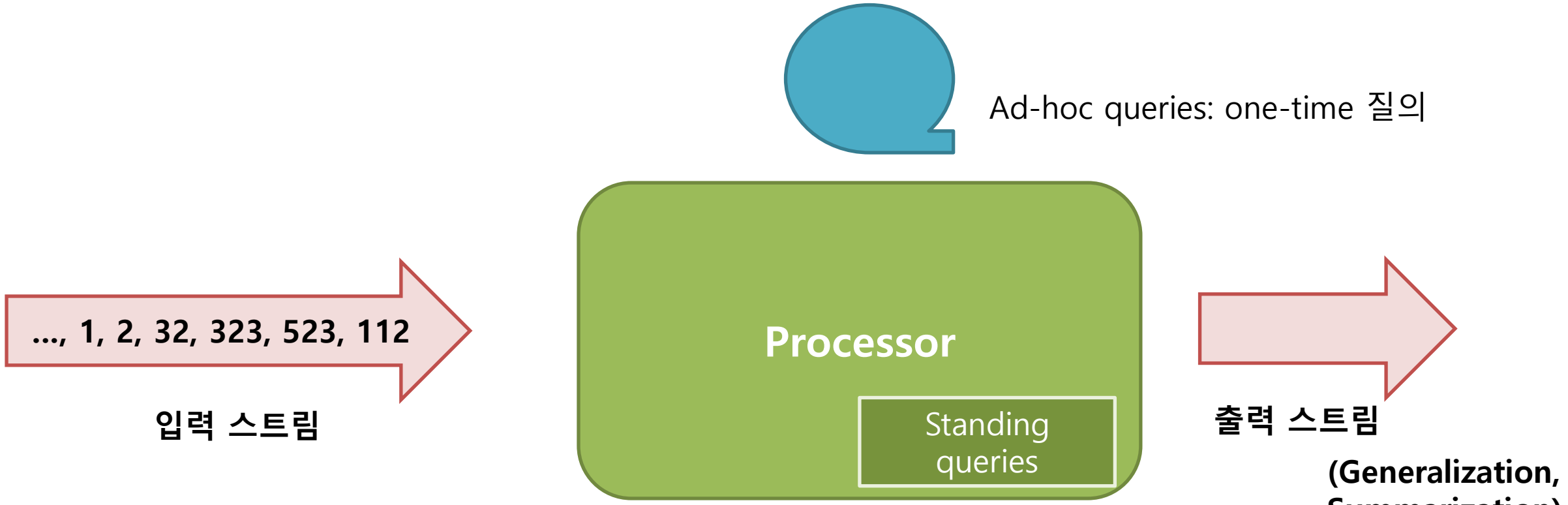
스트림 처리 구조


데이터 입력
(RECORD IN)
스트림 질의를 위한 처리
(Process
for stream queries)
데이터 출력
(RECORD GONE)
하나의 레코드 대신에,
레코드들에 대한 슬라이딩 윈도우를 위치시킨다
.., 5, 2, 23, 11, 23 ,52, 12, 23, 45, 23
슬라이딩 윈도우

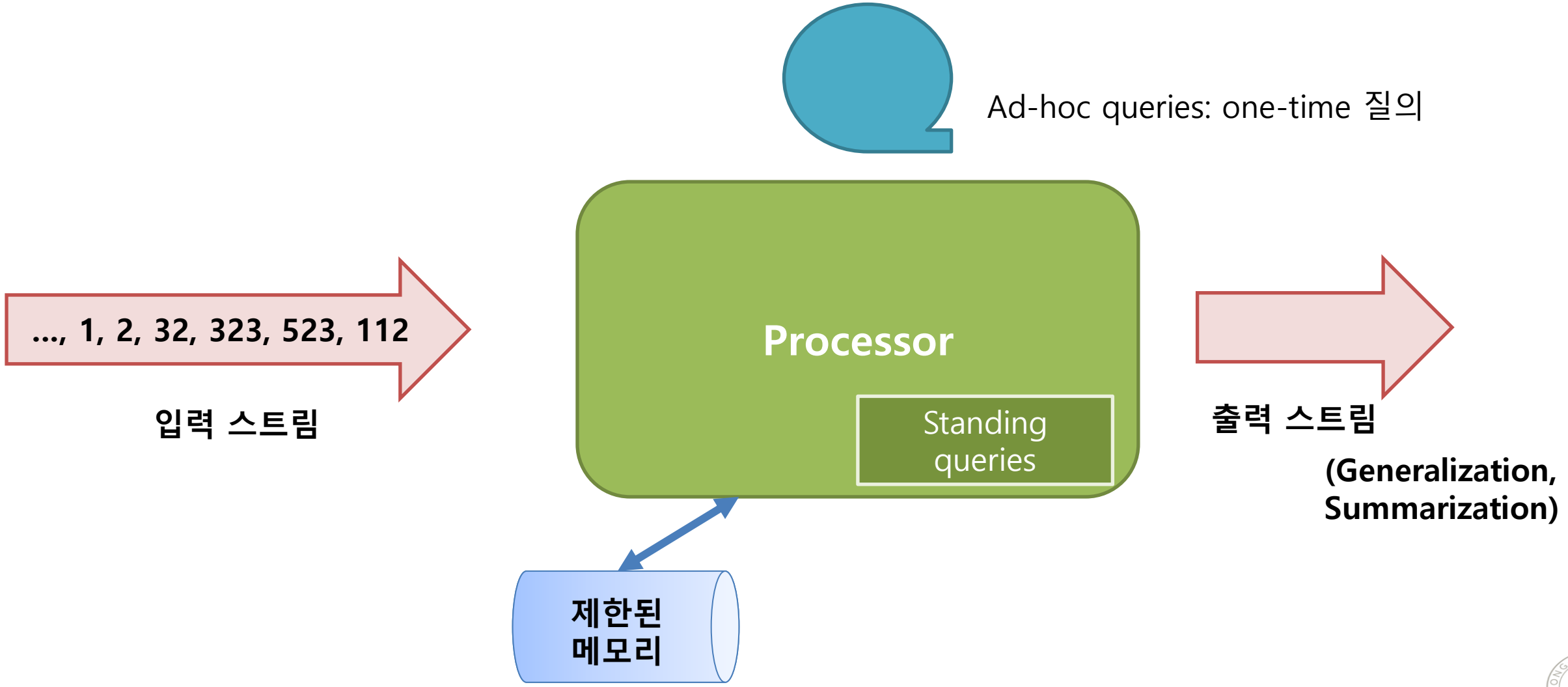
General Stream Processing Model

레코드의 스트림(a stream of records)
레코드 대신 튜플(tuples) 혹은 요소(elements)라고도 함
이론적으로, 어떠한 것도 될수 있음. 검색 질의, 숫자, 비트, 이미지 파일 등

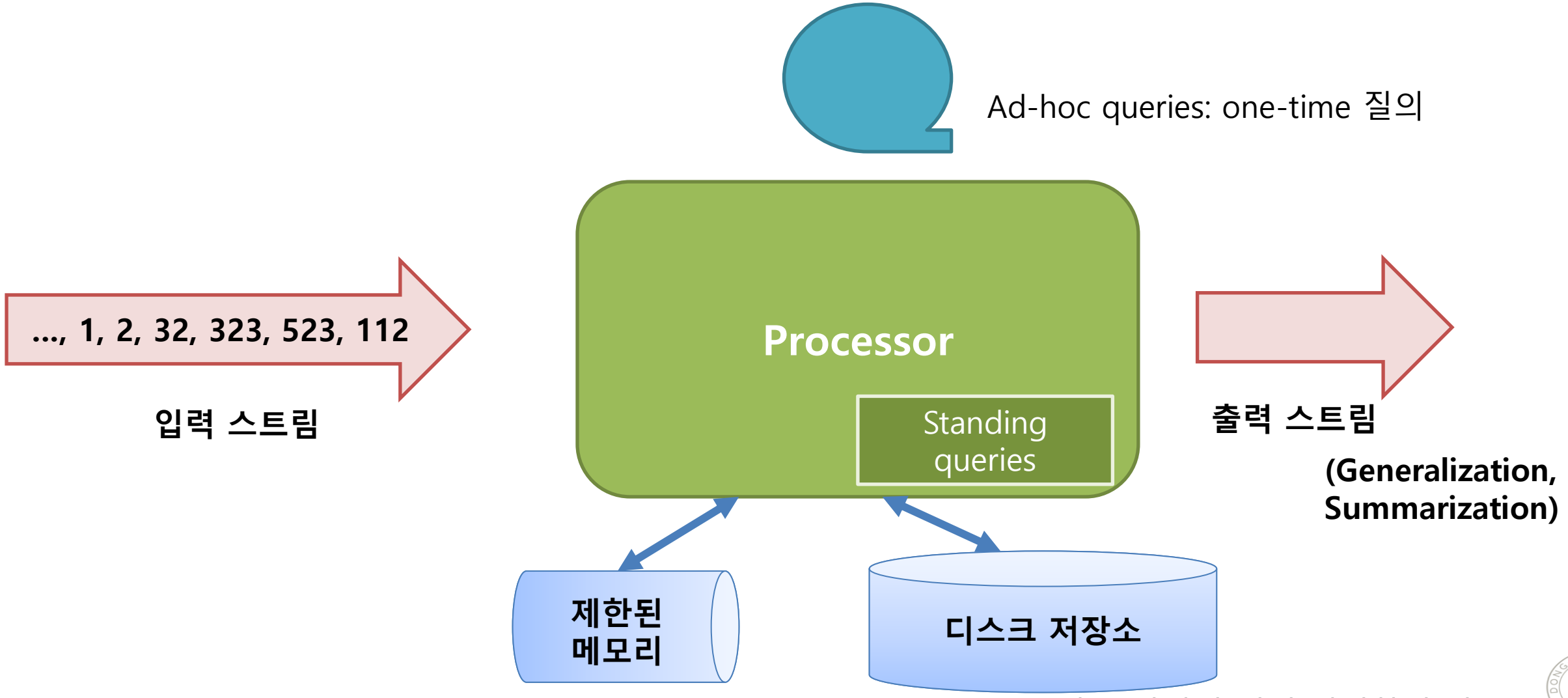
General Stream Processing Model



General Stream Processing Model



General Stream Processing Model

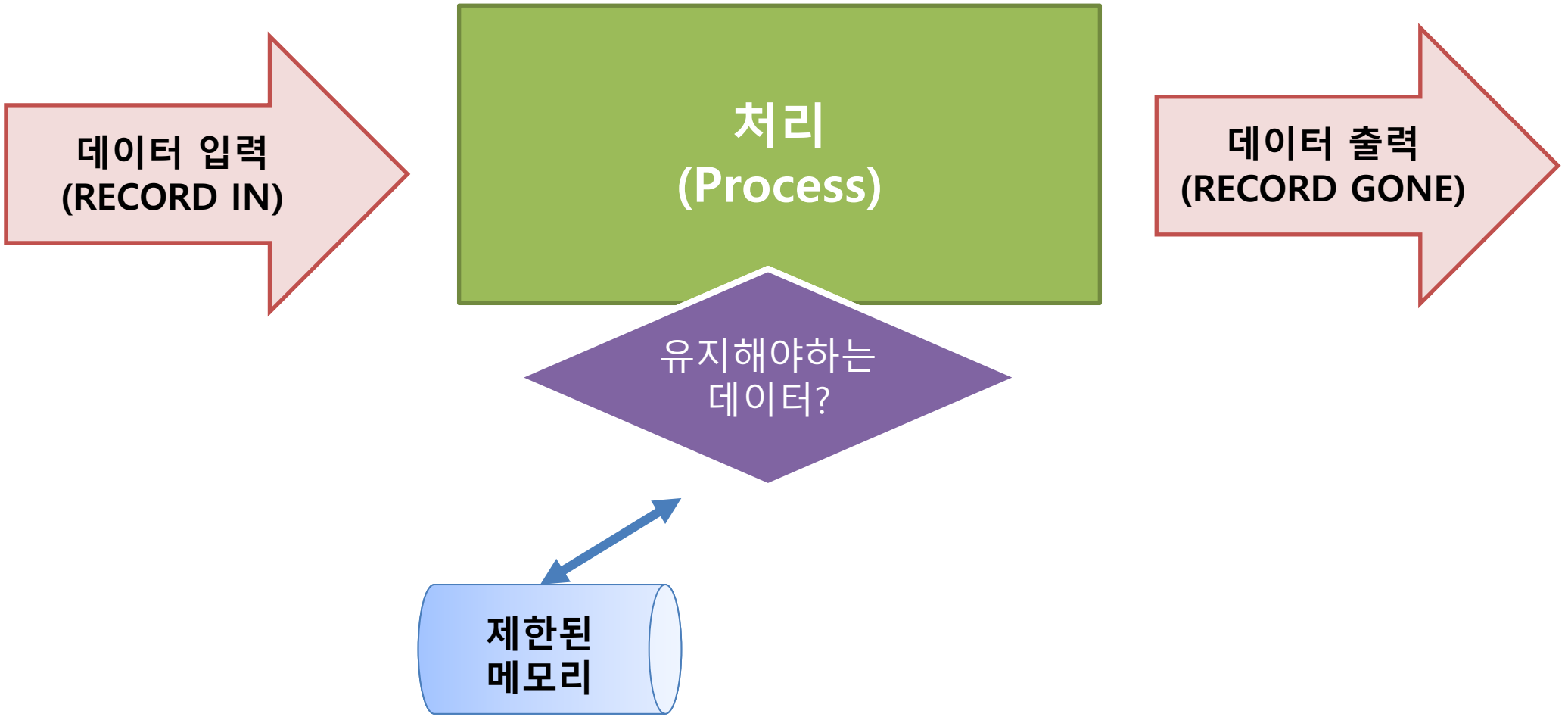


샘플링(Sampling)

- 통계적인 분석을 위한 랜덤 샘플링을 생성

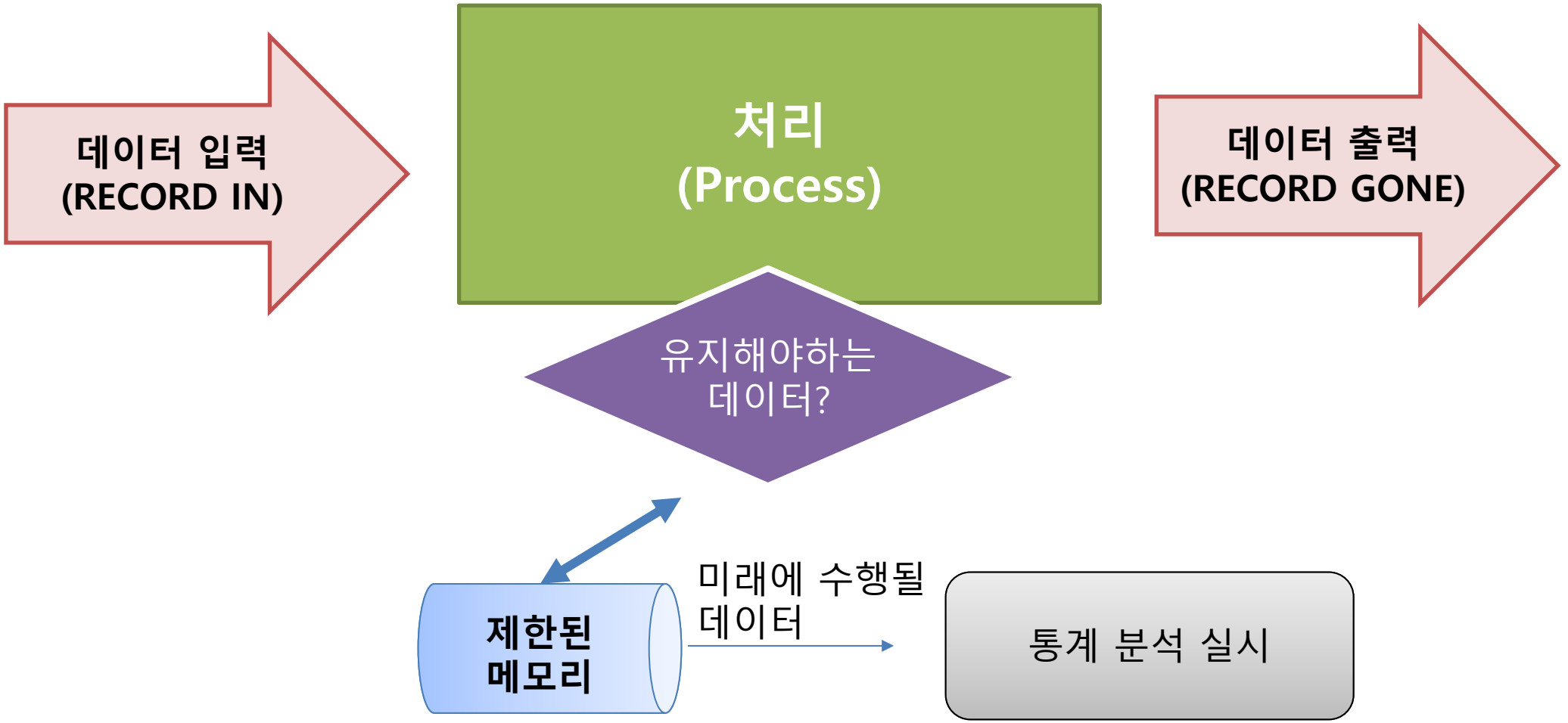
샘플링(Sampling)

- 통계적인 분석을 위한 랜덤 샘플링을 생성



샘플링(Sampling)

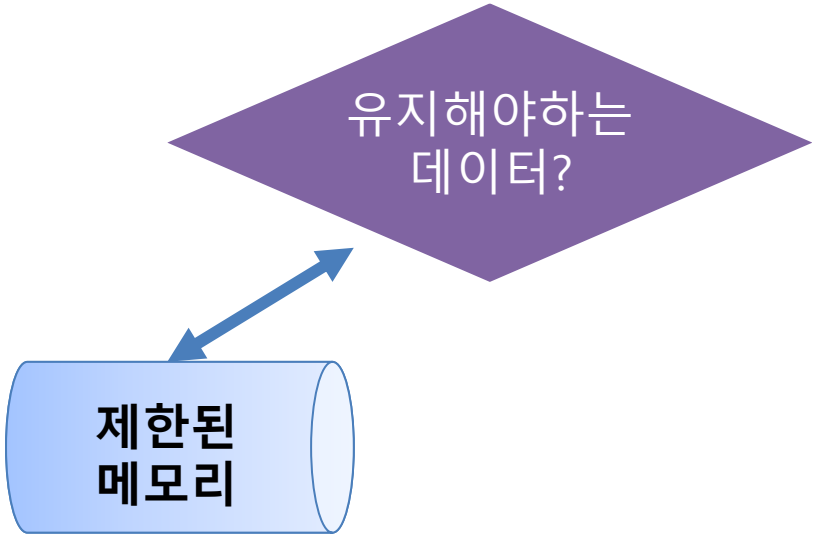
- 통계적인 분석을 위한 랜덤 샘플링을 생성



샘플링(Sampling)

- 통계적인 분석을 위한 랜덤 샘플링을 생성
- 간단한 해결책: 도착하는 레코드에 대해 랜덤 숫자를 생성

```
record = stream.next()
if random() <= .05: # 5% 확률로 유지
    memory.write(record)
```

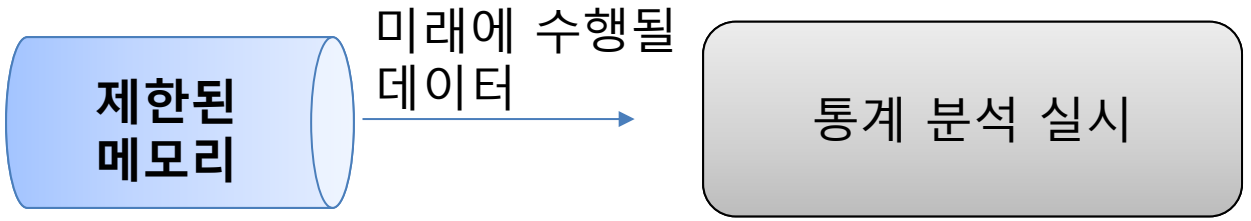


샘플링(Sampling)

- 통계적인 분석을 위한 랜덤 샘플링을 생성
- 간단한 해결책: 도착하는 레코드에 대해 랜덤 숫자를 생성

```
record = stream.next()
if random() <= .05: # 5% 확률로 유지
    memory.write(record)
```

- 문제점: 레코드/튜플은 통계적인 분석을 위한 **분석 단위**가 될 수 없음
  - users_ids for searches, tweets; locations_ids for satellite images



샘플링(Sampling)

- 통계적인 분석을 위한 랜덤 샘플링을 생성
- 간단한 해결책: 도착하는 레코드에 대해 랜덤 숫자를 생성

```
record = stream.next()
if random() <= perc: # perc% 확률로 유지
    memory.write(record)
```

- 문제점: 레코드/튜플은 통계적인 분석을 위한 **분석 단위**가 될 수 없음
  - users_ids for searches, tweets; locations_ids for satellite images
- 해결책: hash into N= 1/perc buckets; 1 버킷을 유지한다

```
If hash(record['user_id']) == 1: # 유지
```

샘플링(Sampling)

- 통계적인 분석을 위한 랜덤 샘플링을 생성
- 간단한 해결책: 도착하는 레코드에 대해 랜덤 숫자를 생성

```
record = stream.next()
if random() <= perc: # perc% 확률로 유지
    memory.write(record)
```

- 문제점: 레코드/튜플은 통계적인 분석을 위한 **분석 단위**가 될 수 없음
  - users_ids for searches, tweets; locations_ids for satellite images
- 해결책: hash into N= 1/perc buckets; 1 버킷을 유지한다

```
If hash(record['user_id']) == 1: # 유지
```

  - Hash 함수를 저장소만 필요하고; 이것은 standing query가 될 것임

Sampling의 Generalization

- Tuples(K,V)
  - K는 key이고 V는 value
- Key에 중심을 두어 sample을 수행해야함.
  - (K,V)의 쌍으로 중심을 두면 안됨.

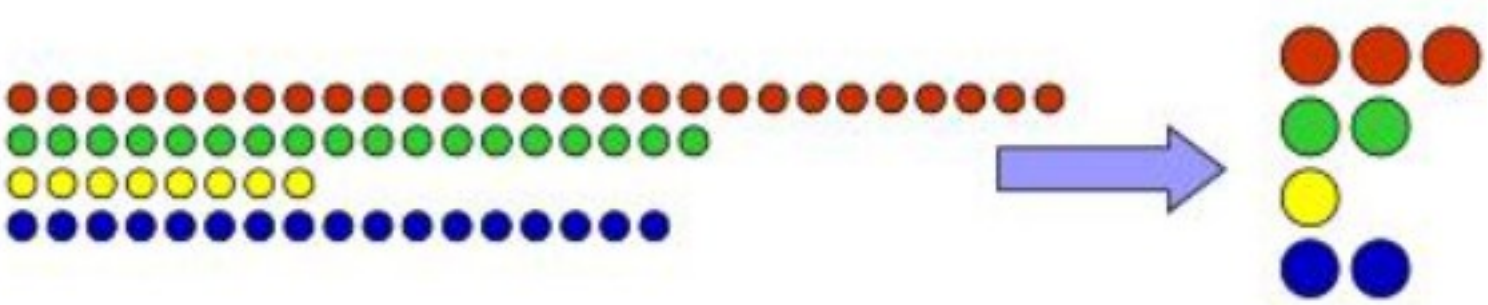
Example: Salary Ranges

- Data = Tuples of the form (EmpID, Dept, Salary)
- Query: 부서 내 salary의 average range는 무엇인가?

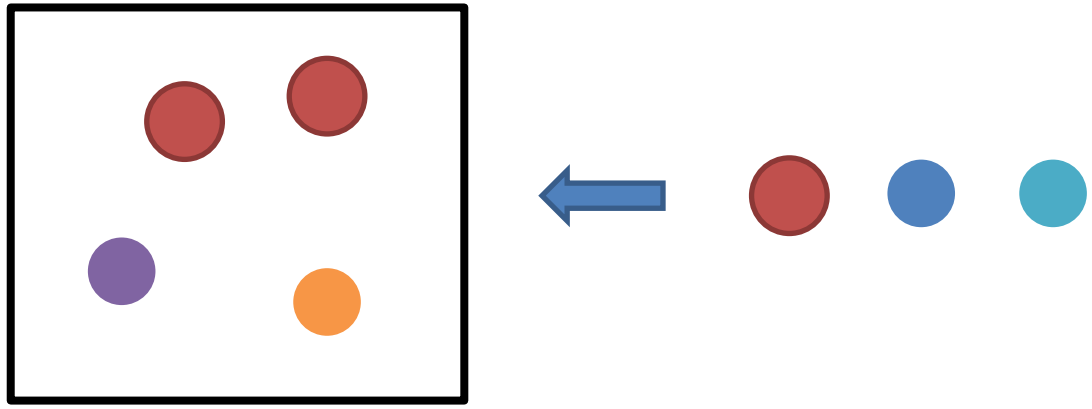
EmpID	Dept	Salary
111	AI	50000
222	CE	40000
442	CE	50000
333	AI	40000

샘플링(Sampling)

- Sample **m** items uniformly from stream
  - Approximate costly computation on small sample
- Challenge: don’t know how long stream is
- Two solution
  - Reservoir sampling
  - Min-wise sampling



Reservoir sampling



- Sample first **m** items
- N 번째 아이템을 확률 **m / n 과 함께** Sample
- If sampled, 랜덤으로 이전의 sampled 아이템과 변경
- 스트림 길이가 k일 때의 확률: 1/k
- 단점: 병렬처리가 어려움

Code: Reservoir_sampling

- https://github.com/vikotse/Reservoir-Sampling/blob/master/reservoir_sampling_py36.py

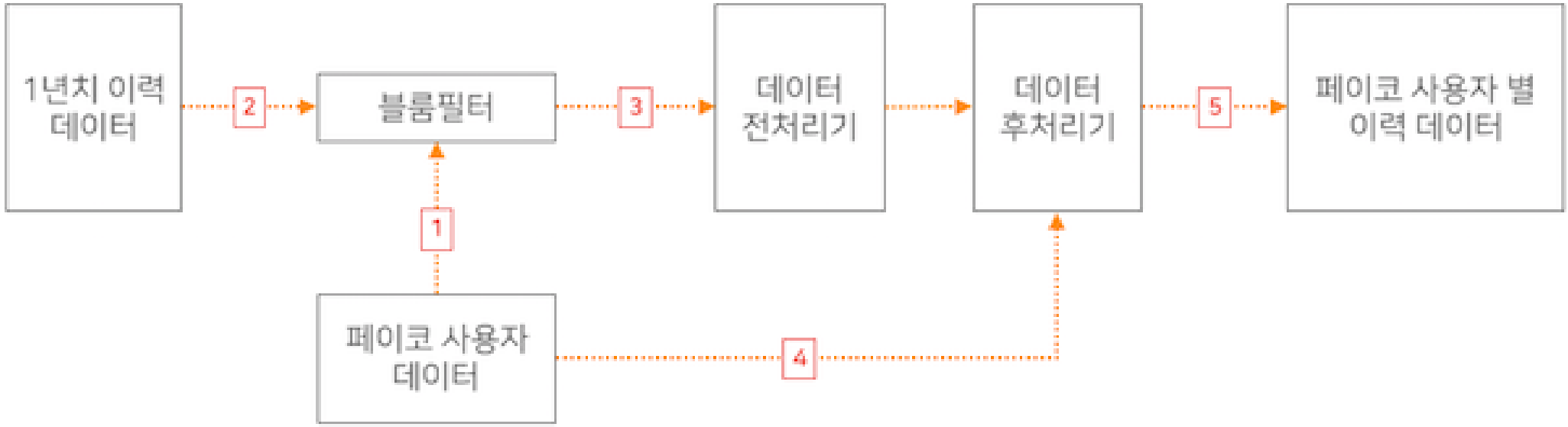
```
4   def reservoir_sampling(sampled_num, total_num):
5       pool = []
6       for i in range(0, total_num):
7           if i < sampled_num:
8               pool.append(i)
9           else:
10              r = random.randint(0, i)
11              if r < sampled_num:
12                  pool[r] = i
13      return pool
```

Filtering Data

- 속성 x값을 갖는 데이터를 선택하기

- 예제: 스팸필터를 위한 40B 안전한 이메일 주소가 주어졌을 때
  - The Bloom Filter (대략적으로, false positives 수락하나 false negatives 은 허용하지 않음)
    - 어떤 값이 집합에 속해 있는가?를 검사하는 필터 및 이를 구성하는 자료형

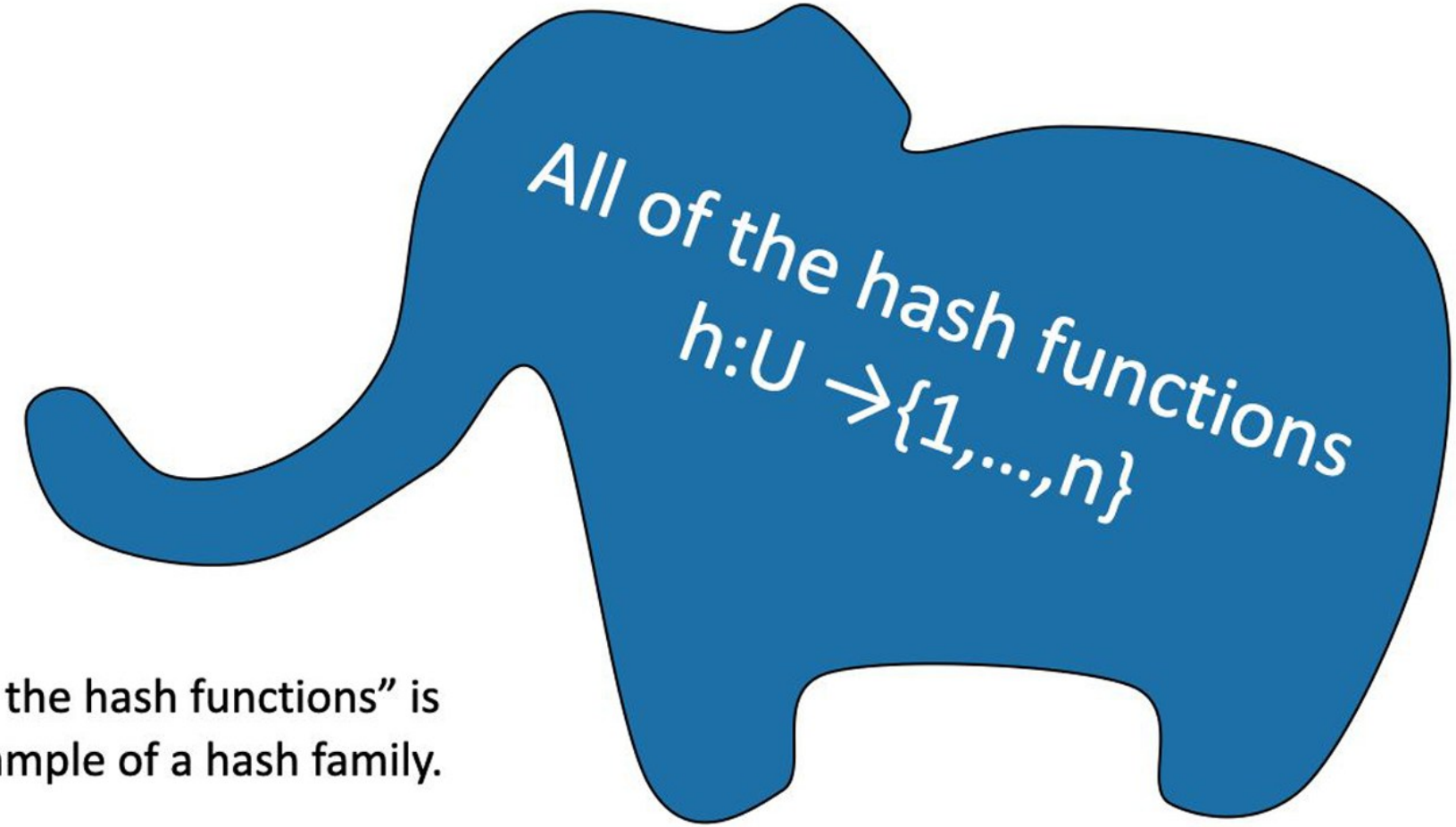
Bloom Filter를 쓰는 예



블룸필터를 이용한 데이터처리 <출처: 내가또그림>

Hash families

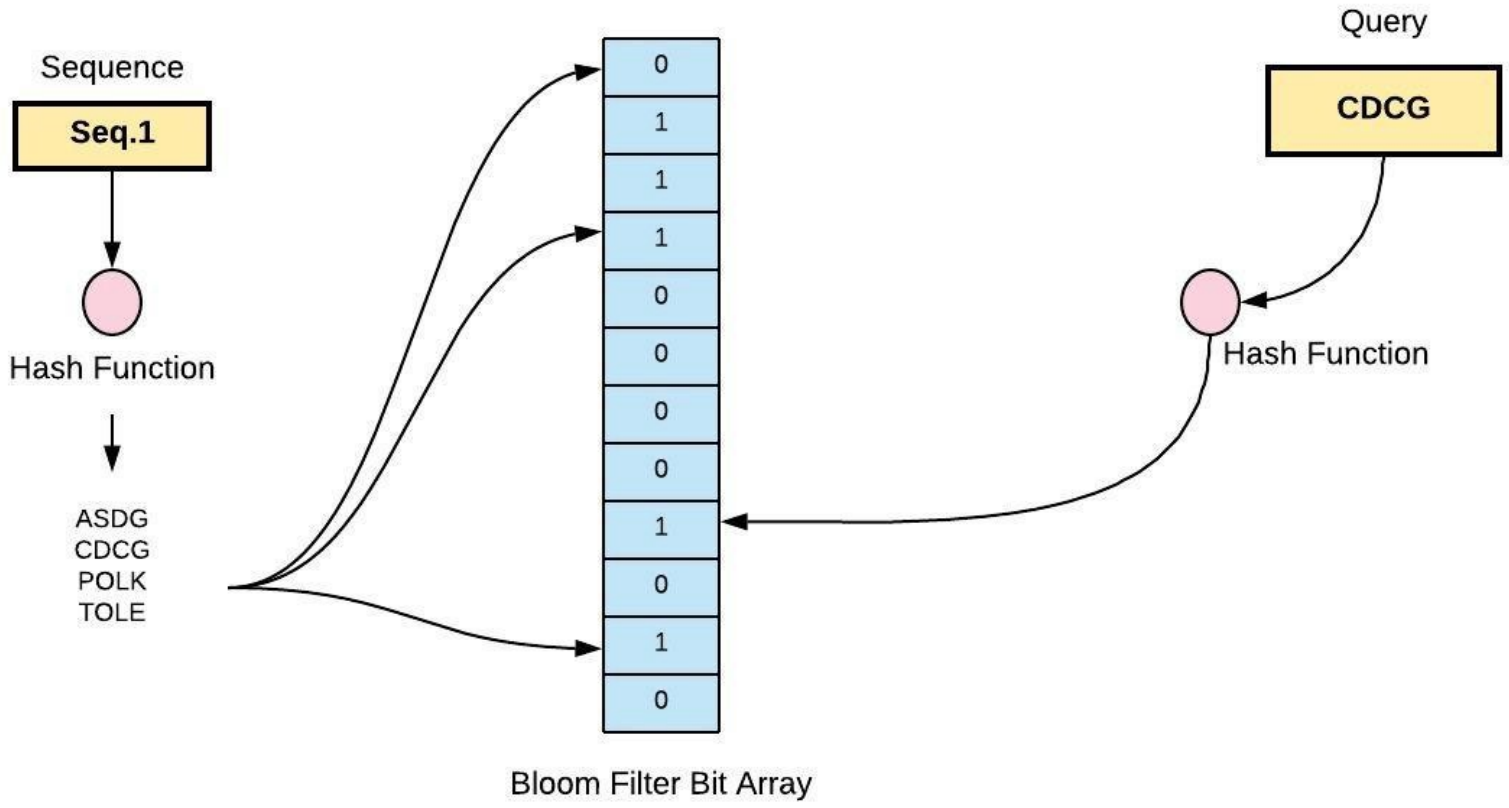
- A collection of hash functions



"All of the hash functions" is an example of a hash family.

Filtering Data

- Bloom filter 알고리즘



Filtering Data: Bloom filter

- 데이터 스트림에 요소 S가 있다고 가정하고,
  우리는 S가 이전에 보았던 것인지를 알고 싶어함
- 각 해쉬 함수와 함께 h_k(S)를 계산
  - 모든 계산된 비트 위치가 1인 경우, 우리는 S를 이전에 보았다고 할 수 있음
  - 물론, 우리는 잘못 인지할 수 있음
    - 다른 입력이 동일한 비트를 만들어 낼 수 있음
- 적어도 하나의 비트가 0이라면, 우리는 S는 이전에 보았던 것이
  아님을 이야기할 수 있음

Example

필터의 크기는 11 bits
H1는 홀수 비트를 선택
H2는 짝수 비트를 선택

Stream element	H1	H2	Filter contents
25			
59			
85			
64			

Bloom Filter

- FP(False Positives)가 존재한다는 점만 유의할 것

Given:

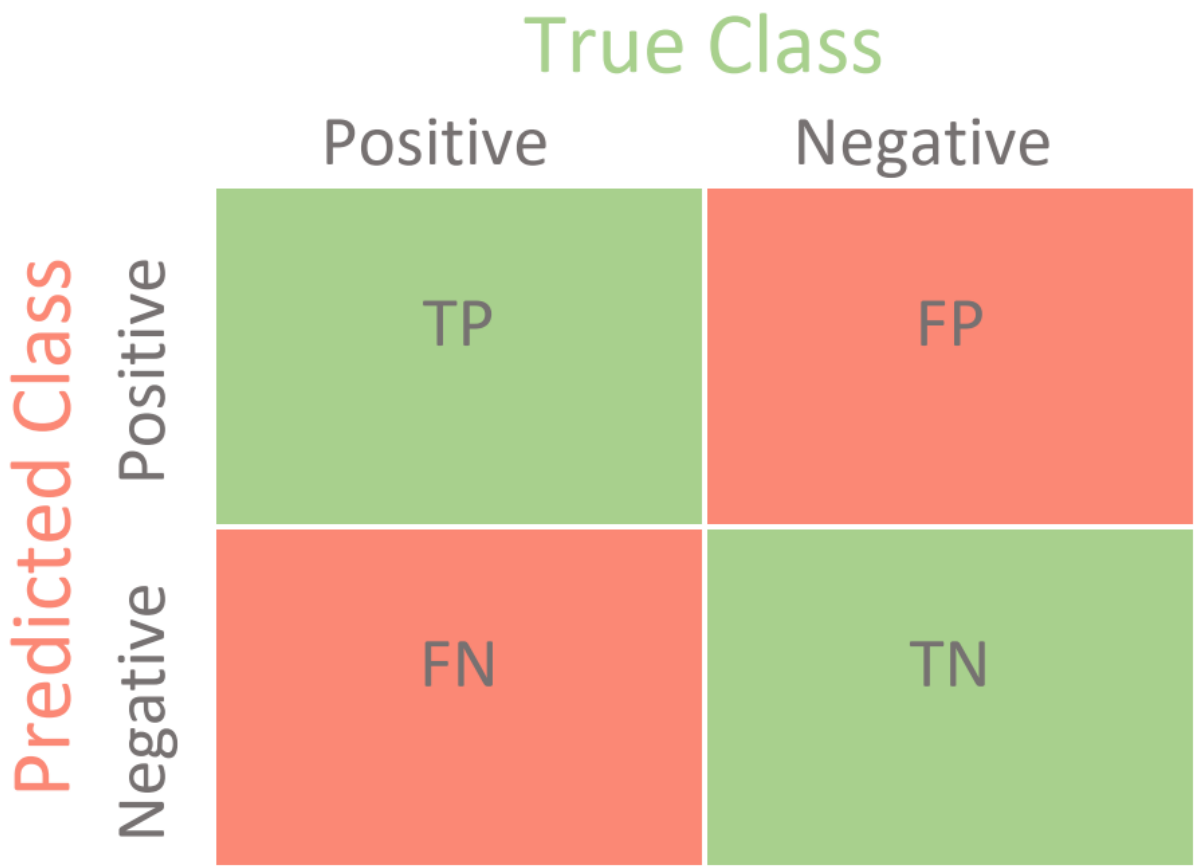
|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  ... #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

Confusion Matrix(오차 행렬)



$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Throwing Darts

- False positives에 대한 심화분석

- **가정**: m개의 다트를 동등한 확률의 n개 타겟에 던진다고 하자
- 타겟이 적어도 하나의 다트를 얻을 확률은?

- 타겟 (n) = 비트수
- 던진 다트 (km) = 아이템의 해쉬 값
  - km = m개 원소 x k 개 해시

Throwing Darts

- m개의 다트와 n개의 타겟이 존재
- 타겟이 적어도 하나의 다트를 얻을 확률은?

$$(1 - 1/n)$$

처음 다트를 던질때, 어떤 타겟이 다트에 의해 히트되지 않을 확률
= 해시 1회 시 특정 비트 안 맞을 확률

Throwing Darts

- m개의 다트와 n개의 타겟이 존재
- 타겟이 적어도 하나의 다트를 얻을 확률은?

$$\left(1 - 1/n\right)^{k}$$

K개의 해시일때,

$$\left(1 - 1/n\right)^{km}$$

M개 원소 일때,

M개 원소 삽입 후 특정 비트가 0일 확률

Throwing Darts

- m개의 다트와 n개의 타겟이 존재
- 타겟이 적어도 하나의 다트를 얻을 확률은?

$$1 - (1 - 1/n)^{km}$$

적어도 하나의 다트가 타켓에 맞출 확률
= 특정 비트가 1이 될 확률

Throwing Darts

- m개의 다트와 n개의 타겟이 존재
- False Positive 조건
  - 새 원소 x가 들어왔을때, 해시 k개 위치가 모두 1이면
- FP 확률

$$\left(1 - \left(1 - 1/n\right)^{km}\right)^{k}$$

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^{n} = 1/e \qquad \left(1-\frac{1}{n}\right)^{km} \approx e^{-km/n}$$

Throwing Darts

- 주어진 배열의 1의 비율 =

  FN의 확률 비율 $=1-e^{-\left(\frac{km}{n}\right)}$

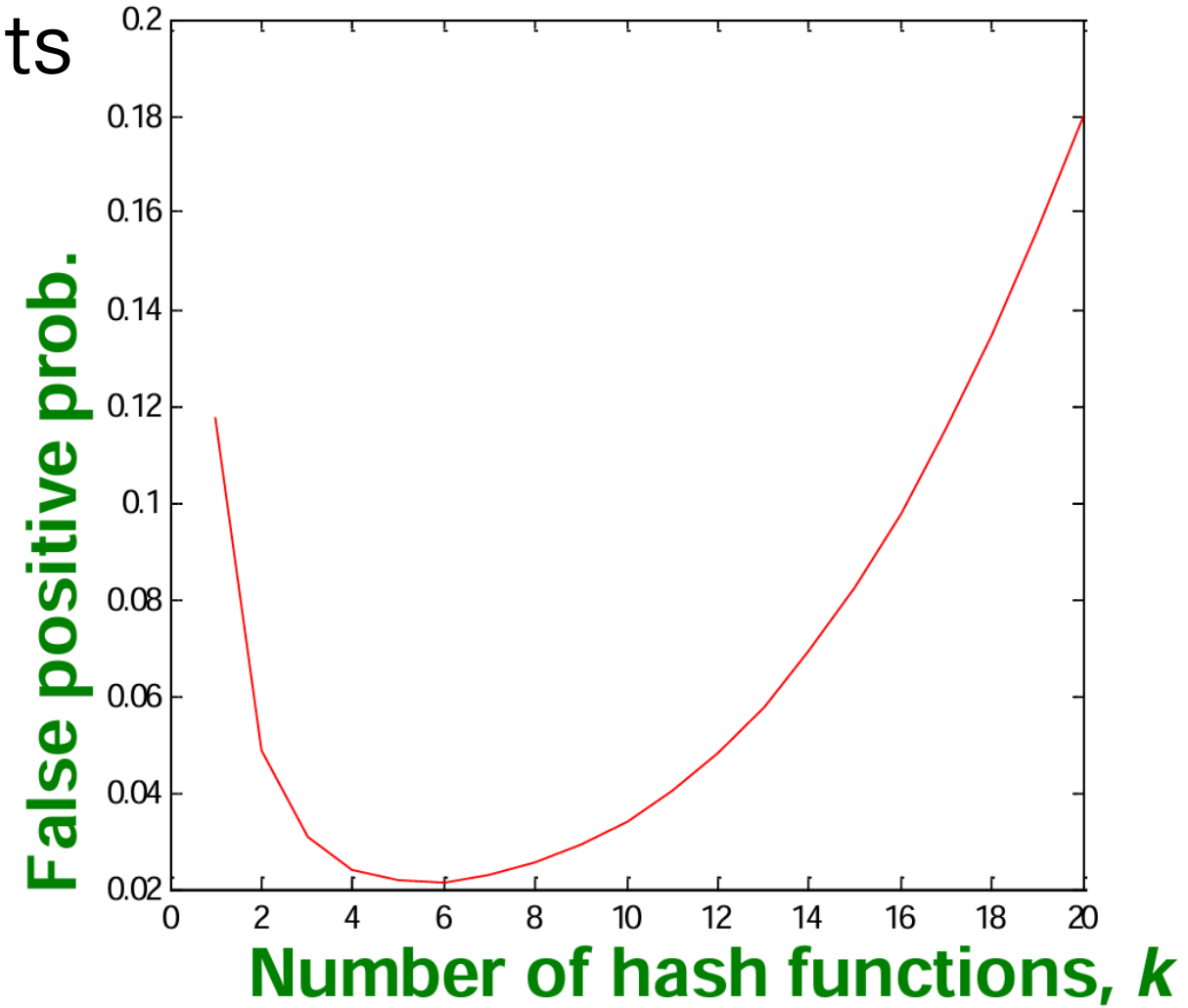
  - n의 타겟, km은 다트의 수(x원소의 수)

- 예제: 10^9 다트, $8*10^9$ 타겟
  - 1의 비율 = 0.1175

Optimal number of hash functions

- m = 1 billion, n = 8 billions bits
  - k = 1: 0.1175
  - k = 2: 0.0489

- Optimal value of k:
  - n / m * ln(2) = 5.54 ~ 6
    - ln(2) = 0.69
  - Error at k = 6
    - $\left(1 - e^{-\frac{3}{4}}\right)^6 = 0.0216$



(Again) Bloom Filter

- FP(False Positives)가 존재한다는 점만 유의할 것

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  ... #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

Bloom Filters (Medium level)

- Better than Linear search and Binary search

- 주어진 오류(False Positives)값을 만족하는 BF의 크기 생성
  - False positive의 확률(p)= $1 - e^{-\left(\frac{m}{n}\right)}$
  - $n = -(m * \ln(p))/(\ln(2)^2)$
- **적절한 Hash function의 개수도 결정**
  - n / m * ln(2)

https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/

Bloom Filter: Wrap-up

- No false negative를 보장하고, 제한된 메모리로 동작
  - 비싼 비용의 Checking 이전에 전처리 시 훌륭한 방법임
  - 병렬 처리가 가능
  - False negative
    - SPAM ID인데 스팸이 아니라고 하는 경우
- False positives가 존재하다 (SPAM ID가 아닌데 스팸이라고 하는 경우)
  - Hash function의 수에 따라 오류율이 달라질 수 있으며, 최적화된 개수를 찾는 것도 매우 중요

Counting Moments

- Moments(순간):
  - 가정: m_i 는 데이터 내 구분 요소 i의 개수
  - 스트림의 K번째 순간은 $\sum_{i \in \text{Set}} m_i^k$
  - 0번째 순간: **구분된 요소의 개수**
  - 1번째 순간: 스트림의 길이
  - 2번째 순간: 제곱의 합

Applications

- 다른 단어들이 웹사이트에서 얼마나 많이 발견되었는가?
  - 웹페이지를 크롤링할때
- 각 고객마다 요청한 다른 웹페이지가 얼마나 있는가?
- 지난 주에 판매한 고유한 상품의 개수는 몇 개인가?

Flajolet-Martin Approach

- N개의 요소를 최소 $\log_2 N$ 비트에 매핑하는 해쉬함수를 선택

- r(e): The number of trailing 0s
  - h(a) = 12, 1100 (binary) r(a) = 2

  - R = $\max_a r(a)$

- 구분된 요소의 추정 개수 = 2^R

Counting Moments

- 0번째 순간

응용:
Counting…
문서 내 구분된 언어
구분된 웹사이트
사이트에 방문한 유저
알렉사/시리 에 요청한 고유한 질의

One solution: Set을 유지(hashmap, dictionary, heap)

문제점: 메모리 내 많은 것을 유지할 수 없음; 디스크 저장은 매우 느림

Counting Moments

0번째 순간

응용:
Counting...
문서 내 구분된 언어
구분된 웹사이트
사이트에 방문한 유저
알렉사/시리 에 요청한 고유한 질의

One solution: Set을 유지(hashmap, dictionary, heap)

문제점: 메모리 내 많은 것을 유지할 수 없음; 디스크 저장은 매우 느림
스트리밍 솔루션: Flajolet-Martin(FM) 알고리즘 O(n)
일반적인 아이디어:
n - 관찰된 요소들의 (예상되는) 전체 개수
해쉬 함수 h를 사용하여 각 요소를 매핑하고 log2 n 비트로 매핑
R # 마지막부분의 0의 최대 수
스트림 데이터가 들어올때, 각 스트림 데이터 e:
r(e) = trailZero(h(e)) # h(e)를 이용하여 trailing 0의 수를 계산
R = r(e) if r[e] > R
추정되는 구분된 요소의 수는 = 2^R

Counting Moments

0번째 순간

응용:
Counting...
문서 내 구분된 언어
구분된 웹사이트
사이트에 방문한 유저
알렉사/시리 에 요청한 고유한 질의

One solution: Set을 유지(hashmap, dictionary, heap)

문제점: 메모리 내 많은 것을 유지할 수 없음; 디스크 저장은 매우 느림
스트리밍 솔루션: Flajolet-Martin(FM) 알고리즘 O(n)
일반적인 아이디어:
n - 관찰된 요소들의 (예상되는) 전체 개수
해쉬 함수 h를 사용하여 각 요소를 매핑하고 log2 n 비트로 매핑
R # 마지막부분의 0의 최대 수
스트림 데이터가 들어올때, 각 스트림 데이터 e:
r(e) = trailZero(h(e)) # h(e)를 이용하여 trail
R = r(e) if r[e] > R
추정되는 구분된 요소의 수는 = 2^R

실용적인 부분은 문제가 있음
Multiple hash functions을 이용하여 결합

1. log n 크기의 그룹으로 파티션
2. 그룹의 평균(mean)을 취하고
3. . 그룹의 평균의 중간값(median)을 취함

Example: Counting Moments (not Scale)

- Input stream = [1,3,2,1,2,3,4,3,1,2,3]
- Hash function, h(x) = 6x + 1 mod 5

h(1) = 2
h(3) = 4
h(2) = 3
h(1) = 2
h(2) = 3
h(3) = 4

h(4) = 0
h(3) = 4
h(1) = 2
h(2) = 3
h(3) = 3
h(1) = 2

Example: Counting Moments (not Scale)

- Input stream = [1,3,2,1,2,3,4,3,1,2,3]
- Hash function, h(x) = 6x + 1 mod 5

h(1) = 2 = 010
h(3) = 4 = 100
h(2) = 3 = 011
h(1) = 2 = 010
h(2) = 3 = 011
h(3) = 4 = 100

h(4) = 0 = 000
h(3) = 4 = 100
h(1) = 2 = 010
h(2) = 3 = 011
h(3) = 3 = 100
h(1) = 2 = 010

Example: Counting Moments (not Scale)

- Input stream = [1,3,2,1,2,3,4,3,1,2,3]
- Hash function, h(x) = 6x + 1 mod 5

h(1) = 2 = 010
h(3) = 4 = 100
h(2) = 3 = 011
h(1) = 2 = 010
h(2) = 3 = 011
h(3) = 4 = 100

1

h(4) = 0 = 000
h(3) = 4 = 100
h(1) = 2 = 010
h(2) = 3 = 011
h(3) = 3 = 100
h(1) = 2 = 010

2

Example: Counting Moments (not Scale)

- Input stream = [1,3,2,1,2,3,4,3,1,2,3]
- Hash function, h(x) = 6x + 1 mod 5

h(1) = 2 = 010 =1
h(3) = 4 = 100 =2
h(2) = 3 = 011 =0
h(1) = 2 = 010 =1
h(2) = 3 = 011 =0
h(3) = 4 = 100 =2

h(4) = 0 = 000 =0
h(3) = 4 = 100 =2
h(1) = 2 = 010 =1
h(2) = 3 = 011 =0
h(3) = 3 = 100 =2
h(1) = 2 = 010 =1

Example: Counting Moments (not Scale)

- Input stream = [1,3,2,1,2,3,4,3,1,2,3]
- Hash function, h(x) = 6x + 1 mod 5

h(1) = 2 = 010 =1
h(3) = 4 = 100 =2
h(2) = 3 = 011 =0
h(1) = 2 = 010 =1
h(2) = 3 = 011 =0
h(3) = 4 = 100 =2

R= 2

h(4) = 0 = 000 =0
h(3) = 4 = 100 =2
h(1) = 2 = 010 =1
h(2) = 3 = 011 =0
h(3) = 3 = 100 =2
h(1) = 2 = 010 =1

2^R = 2^2 = 4 [1,3,2,4]

Why it works

- h(a)가 적어도 r의 0으로 끝날 확률은 2^{-r}

- 따라서, m개의 서로 다른 원소들 중에서
  tail 길이가 r이 아닐 확률은 $1- 2^{-r}$

- M개의 요소들 가운데 r의 tail을 가진 찾지 못할 확률은
  - $(1 - 2^{-r})^{m}$

요약

- 스트림 처리 구조
  - Standing queries, Single pass

- 스트림 처리에 대한 주제
  - 일반적인 모델
  - 샘플링
  - 데이터 필터링
  - 구분된 요소 카운팅