Streaming Algorithms: Data without a disk

H. Andrew Schwartz

CSE545
Spring 2023

Big Data Analytics, The Class

Goal: Generalizations
A *model* or *summarization* of the data.

Data Workflow Frameworks

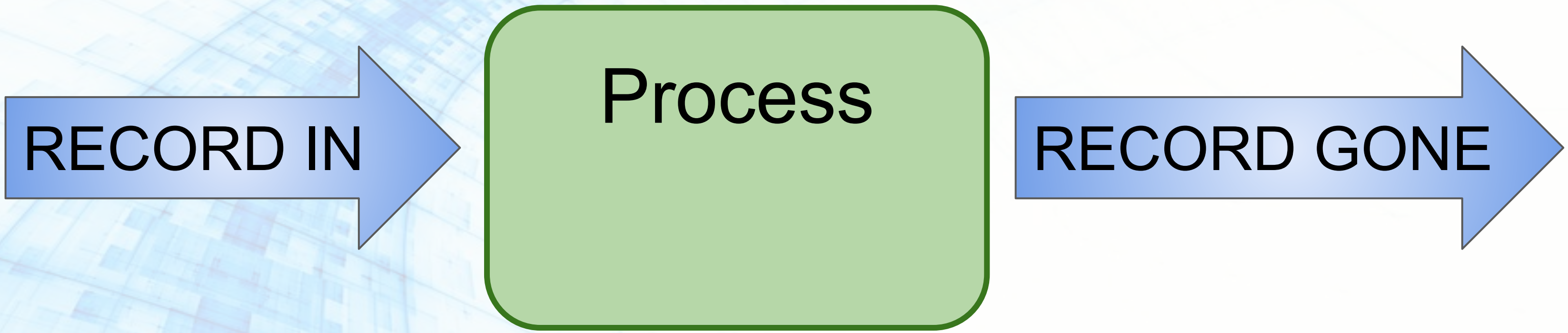
- Hadoop File System
- Spark
- **Streaming**
- MapReduce
- Deep Learning Frameworks

Analytics and Algorithms

- Similarity Search
- Hypothesis Testing
- Transformers/Self-Supervision
- Recommendation Systems
- Link Analysis

What is Streaming?

Broadly:



Why Streaming?

(1) **Direct:** Often, data …

- … cannot be stored (too big, privacy concerns)
- … are not practical to access repeatedly (reading is too long)
- … are rapidly arriving (need rapidly updated "results")

Why Streaming?

(1) **Direct:** Often, data …

- … cannot be stored (too big, privacy concerns)
- … are not practical to access repeatedly (reading is too long)
- … are rapidly arriving (need rapidly updated "results")

Examples: *Google search queries*

Satellite imagery data

Text Messages, Status updates

Click Streams

Why Streaming?

(1) **Direct:** Often, data …

- … cannot be stored (too big, privacy concerns)
- … are not practical to access repeatedly (reading is too long)
- … are rapidly arriving (need rapidly updated "results")

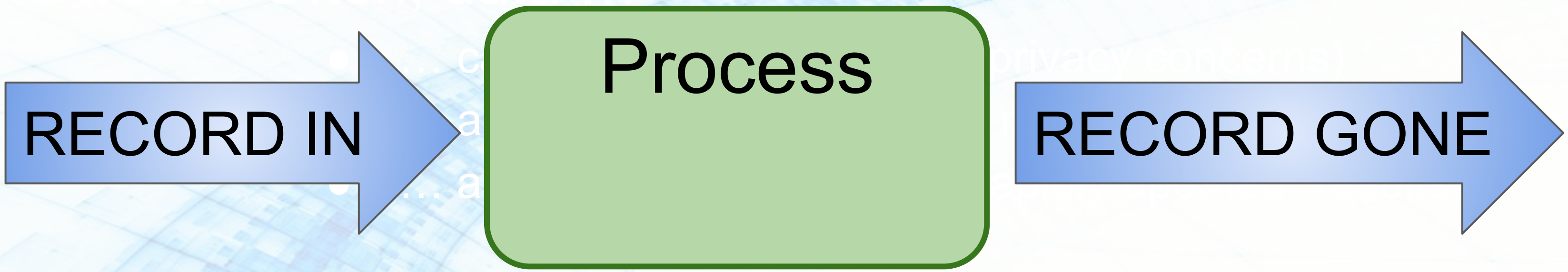
(2) **Indirect:** The constraints for streaming data force one to solutions that are often efficient even when storing data.

Streaming Approx Random Sample

Distributed IO (MapReduce, Spark)

Why Streaming?

Often translates into *O(N)* or strictly *N* algorithms.



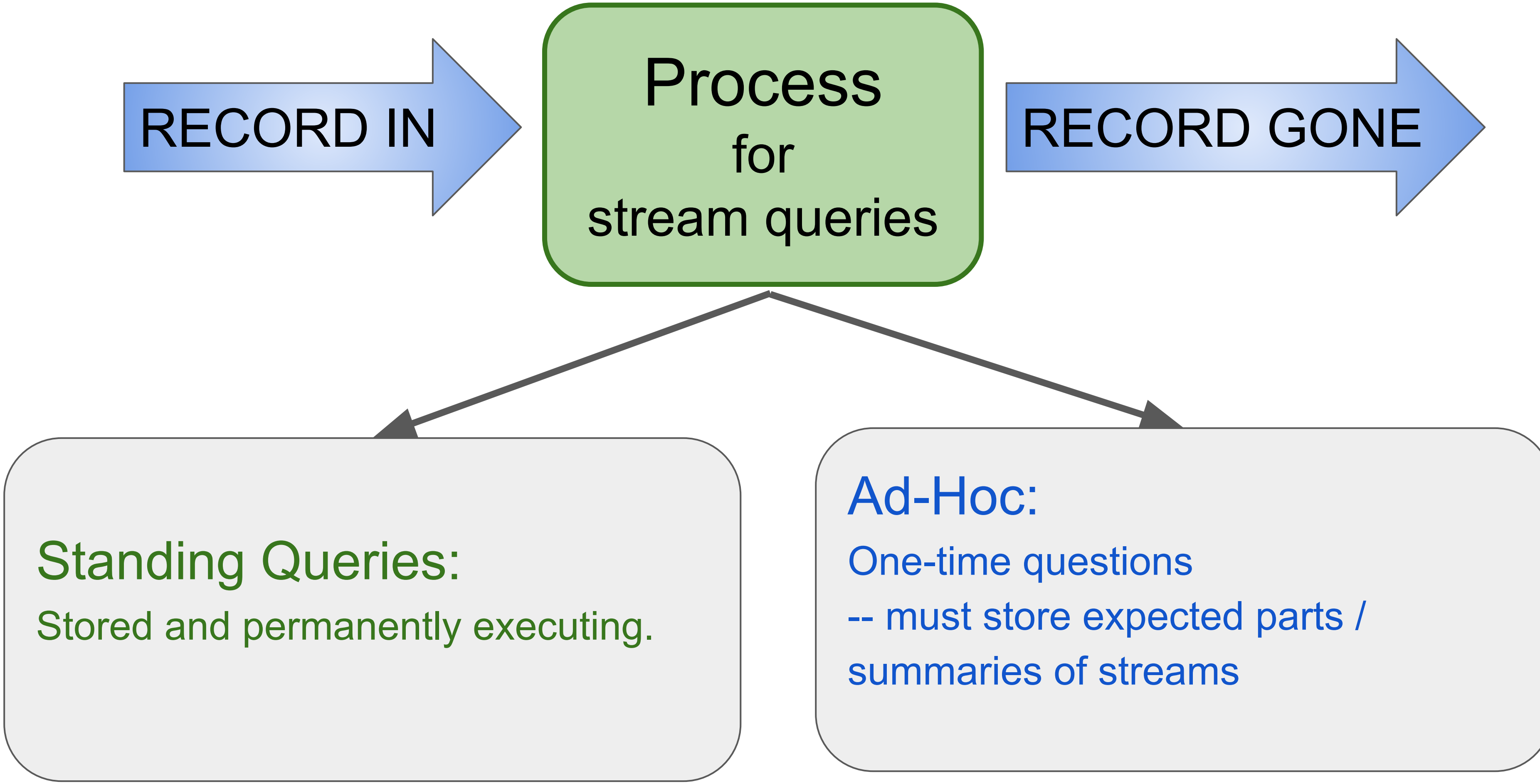
(2) **Indirect:** The constraints for streaming data force one to solutions that are often efficient even when storing data.

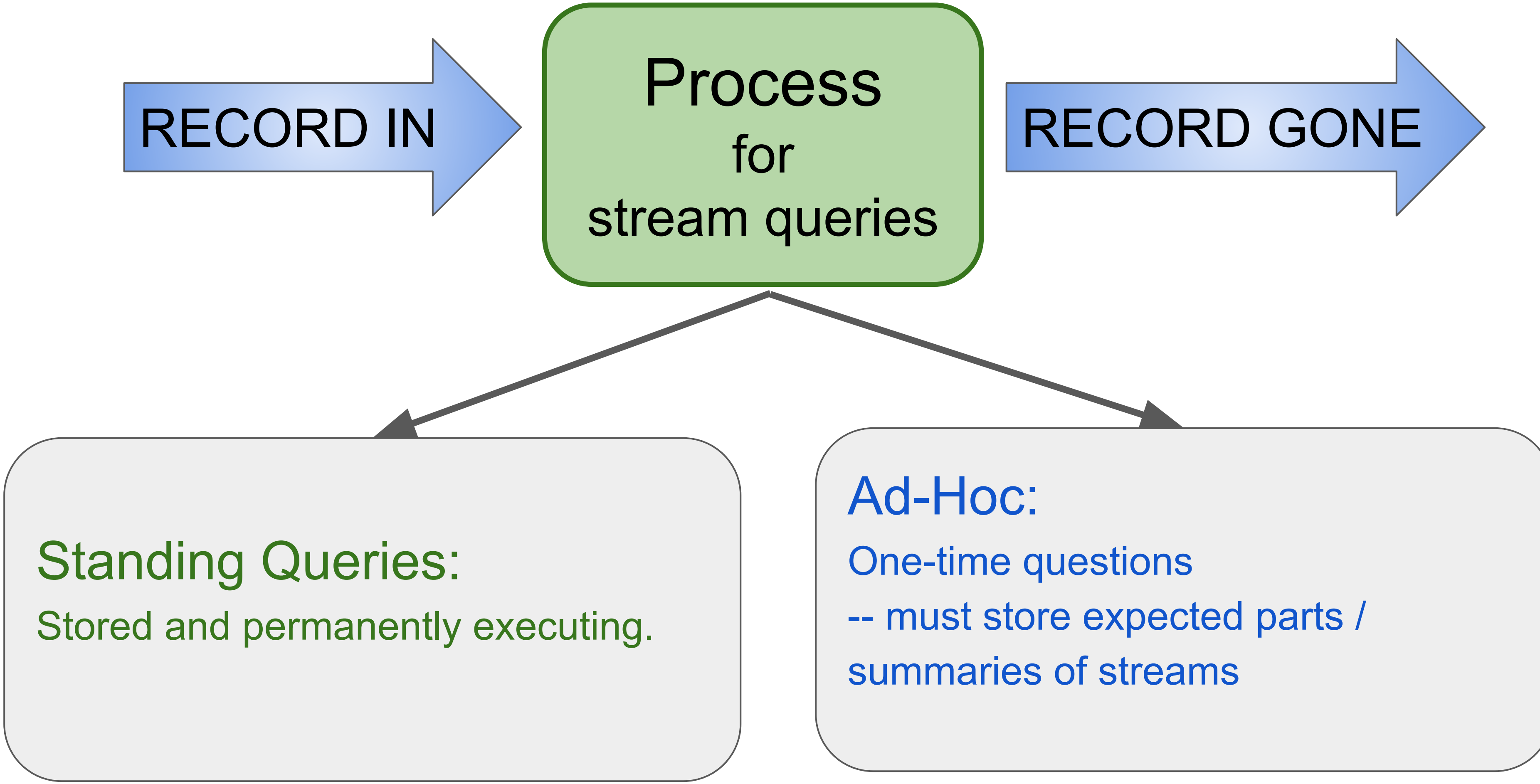
Streaming Approx Random Sample

Distributed IO (MapReduce, Spark)

Streaming Topics

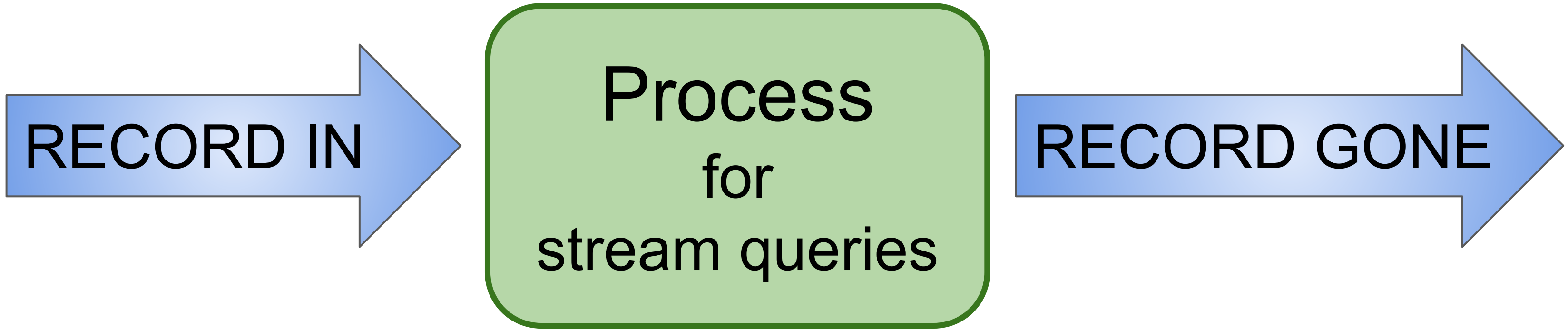
- General Stream Processing Model
- Sampling
- Counting Distinct Elements
- Filtering data according to a criteria


RECORD IN
Process
for
stream queries
RECORD GONE
Standing Queries:
Stored and permanently executing.
Ad-Hoc:
One-time questions
-- must store expected parts /
summaries of streams



E.g. How would you handle:

What is the mean of values seen so far?

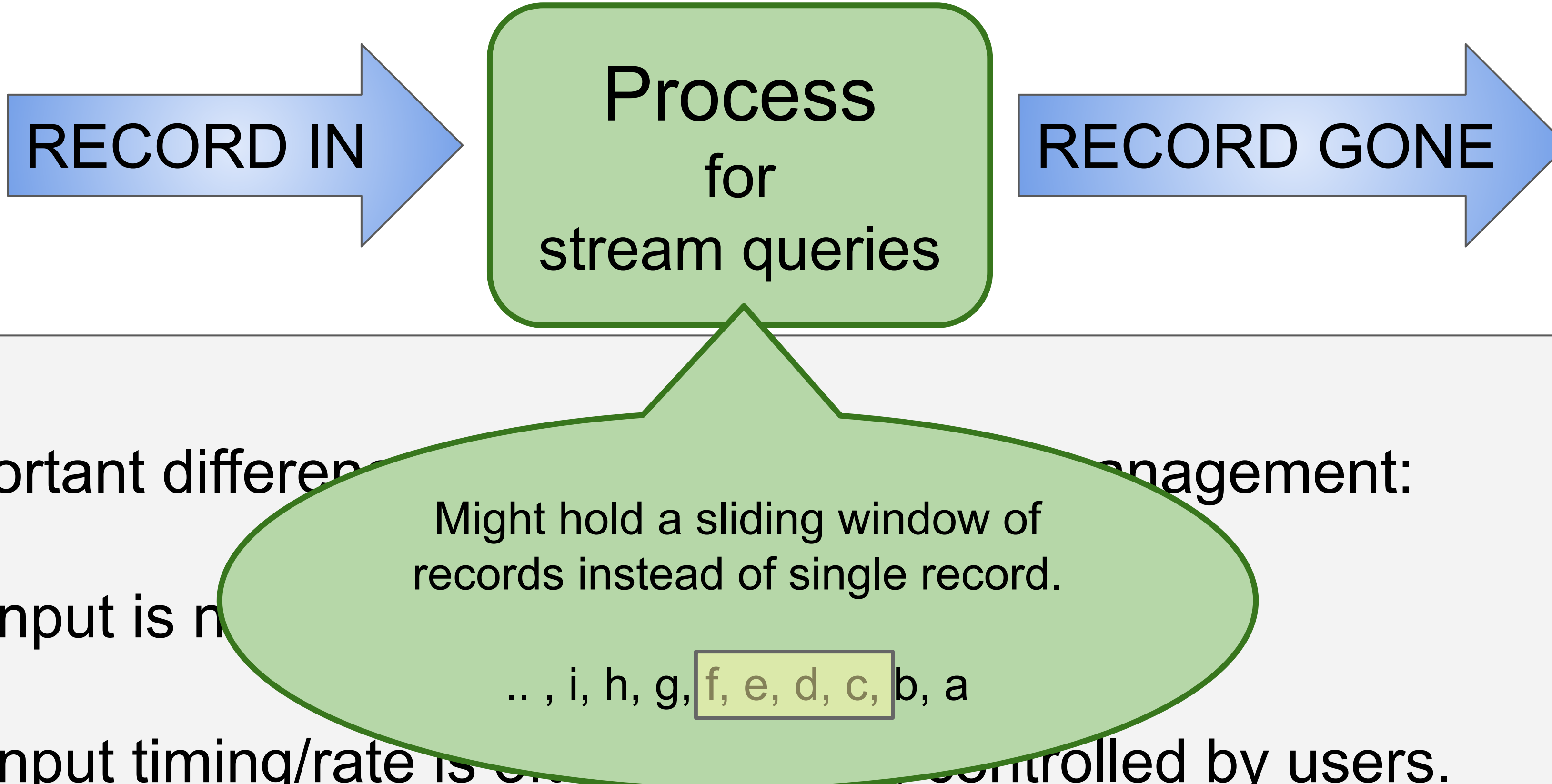


Important difference from typical database management:

- Input is not controlled by system staff.
- Input timing/rate is often unknown, controlled by users.

E.g. How would you handle:

What is the mean of values seen so far?



Important differen ... anagement:

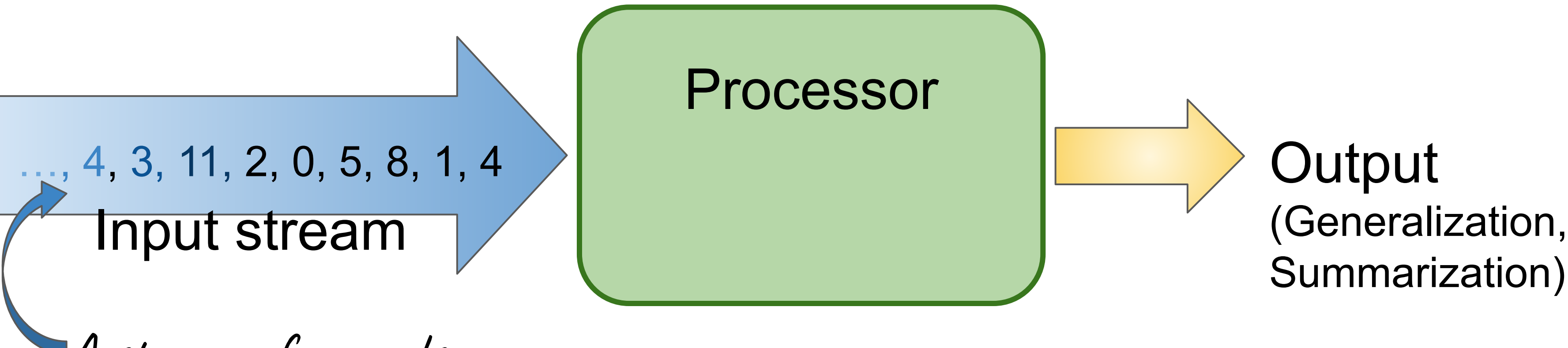
- Input is n
- Input timing/rate is ... controlled by users.

E.g. How would you handle:

What is the mean of values seen so far?

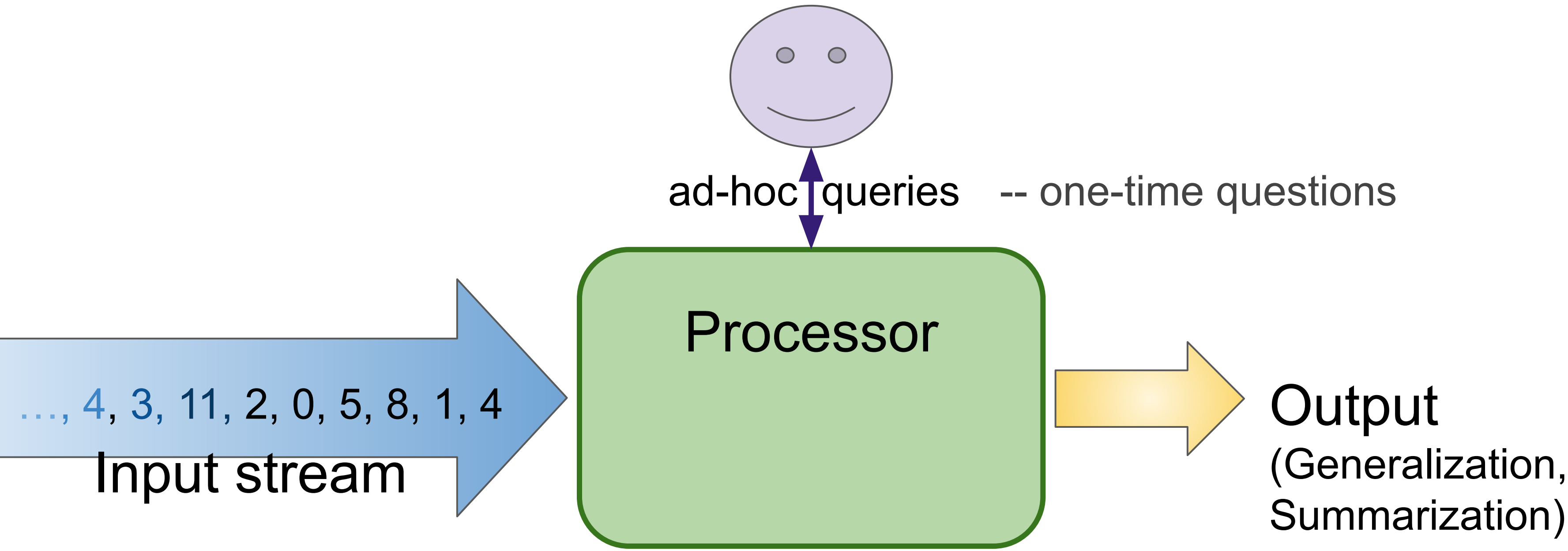
General Stream Processing Model

(Leskovec et al., 2014)

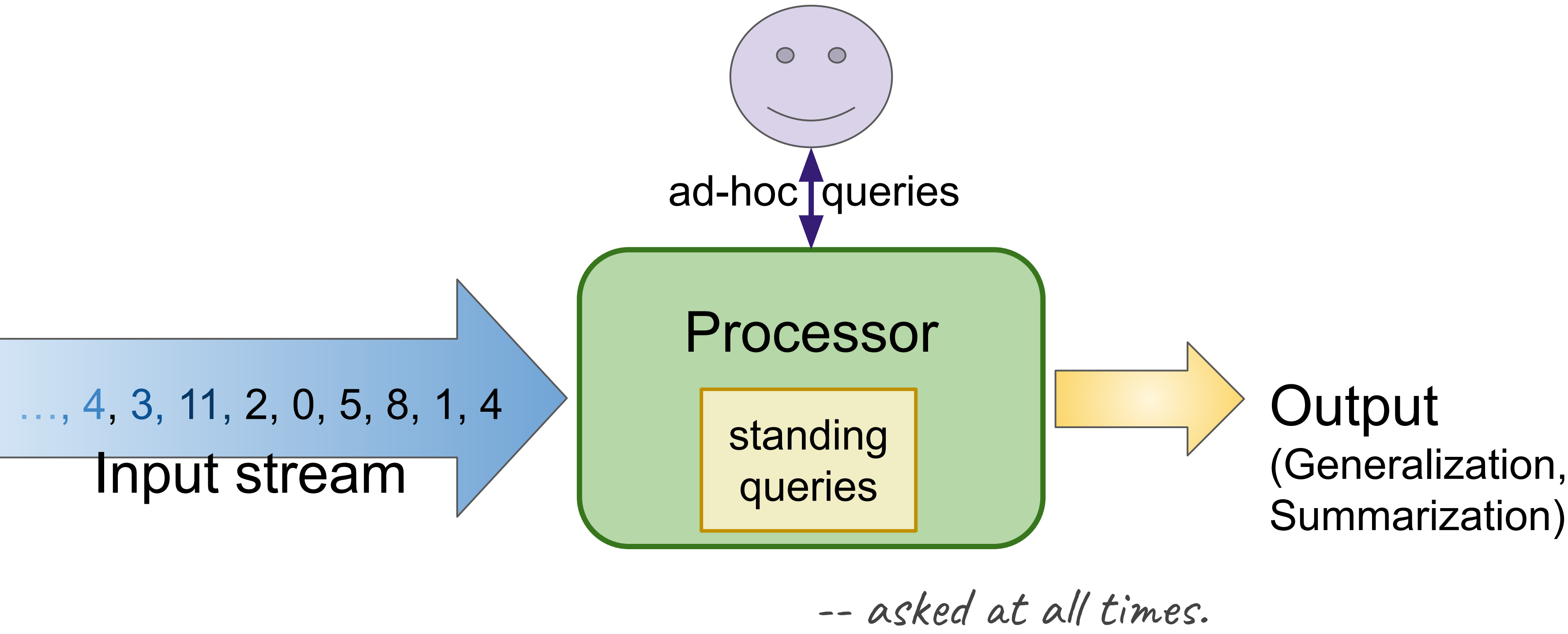


A stream of records
(also often referred to as "elements", "tuples", "lines", or "rows")
Theoretically, could be anything! search queries, numbers, bits, image files, ...

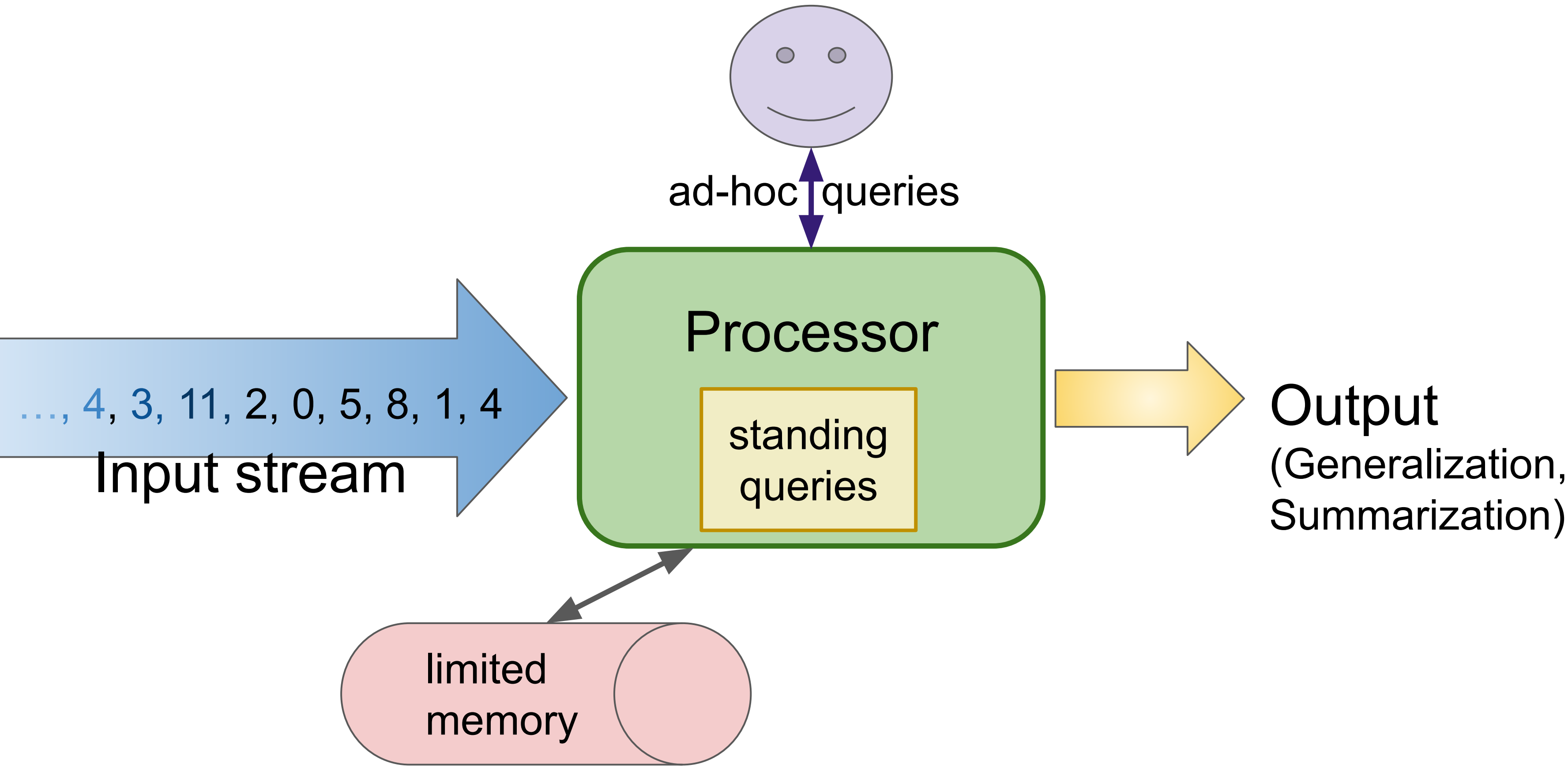
General Stream Processing Model



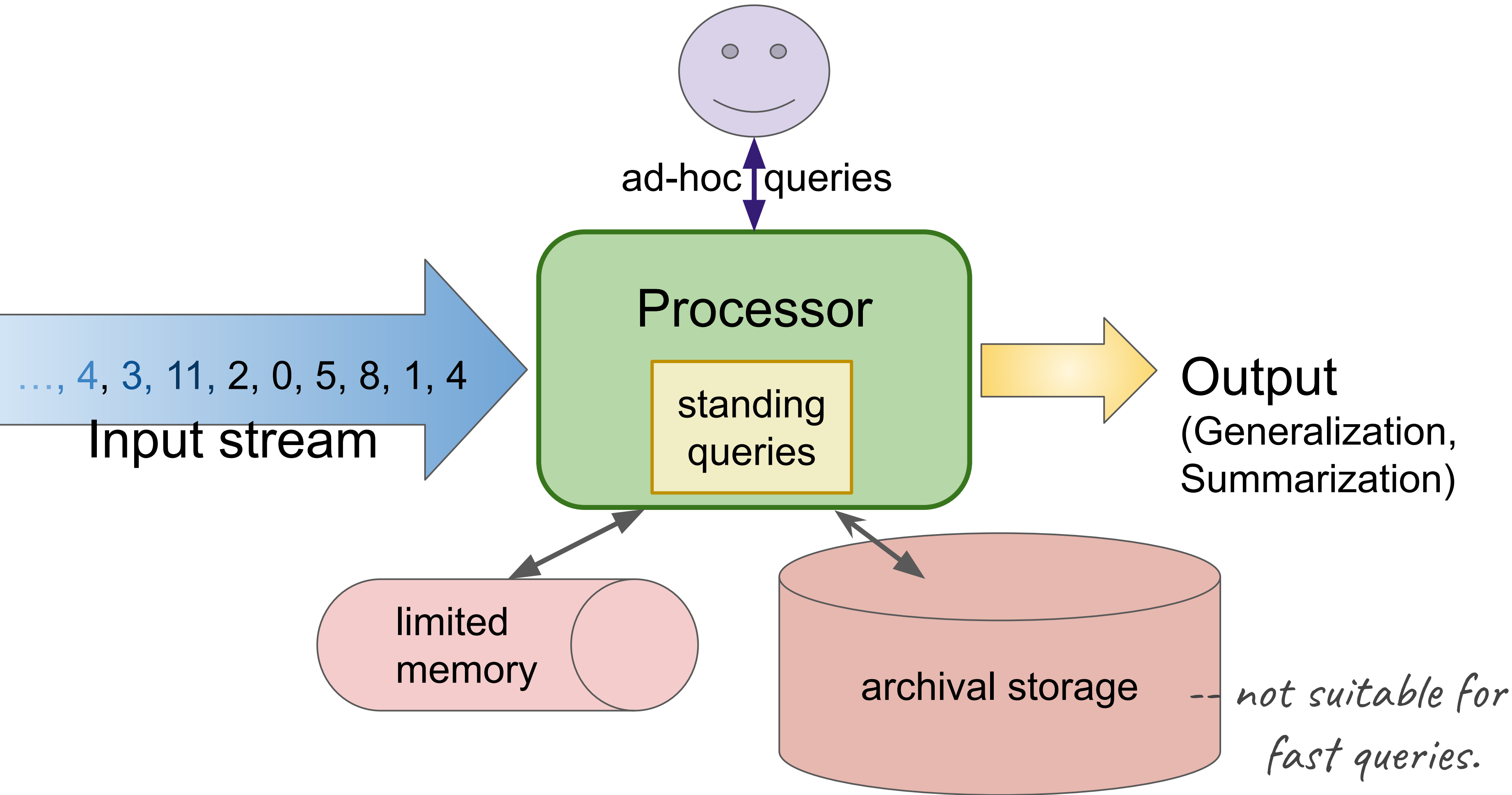
General Stream Processing Model



General Stream Processing Model



General Stream Processing Model

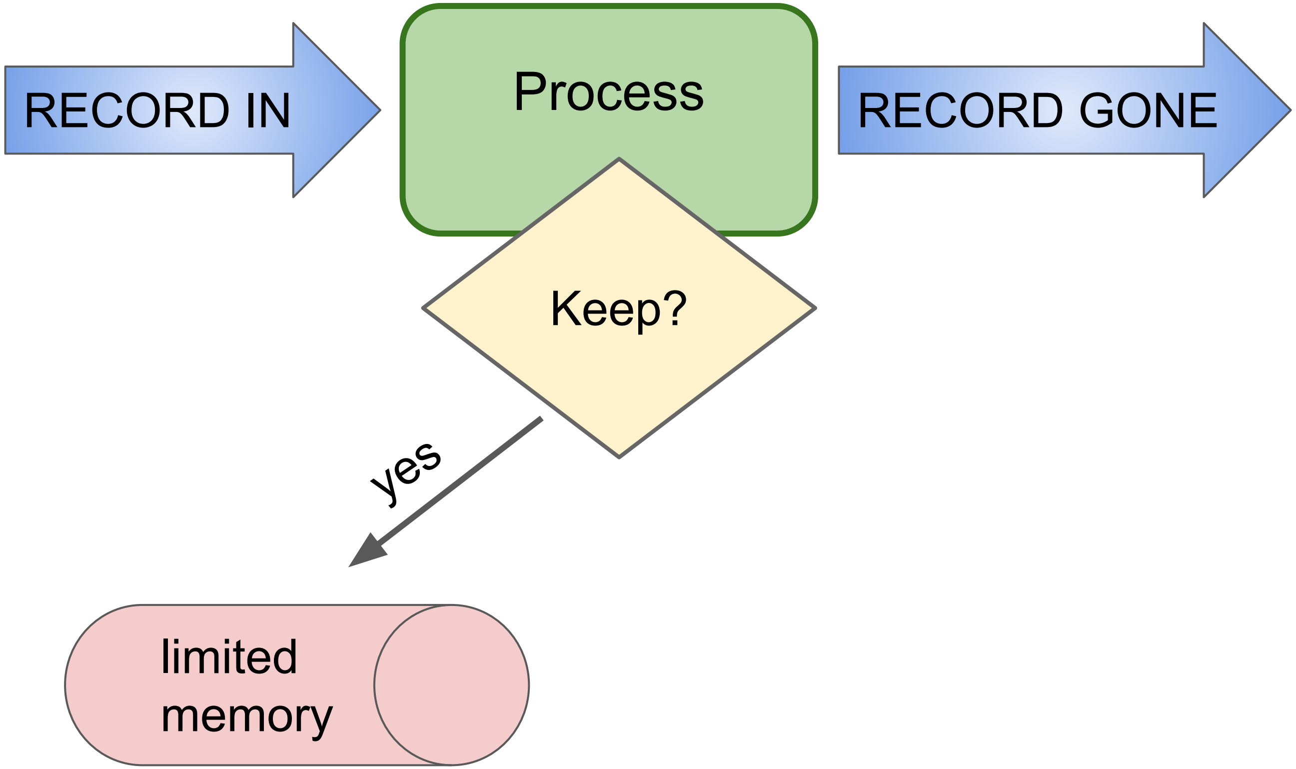


Sampling

Create a random sample for statistical analysis.

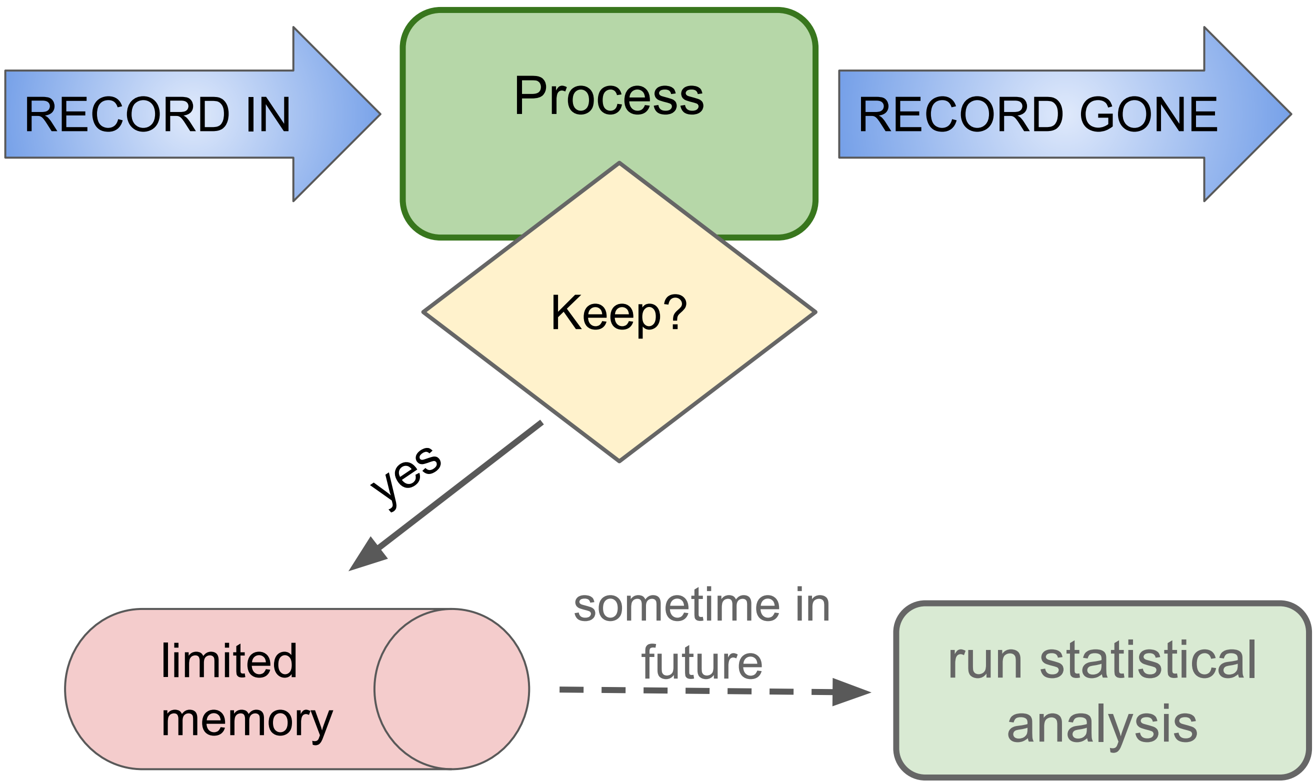
Sampling

Create a random sample for statistical analysis.



Sampling

Create a random sample for statistical analysis.



Sampling: 2 Versions

Create a random sample for statistical analysis.

1. **Simple Sampling:** Individual records are what you wish to sample.

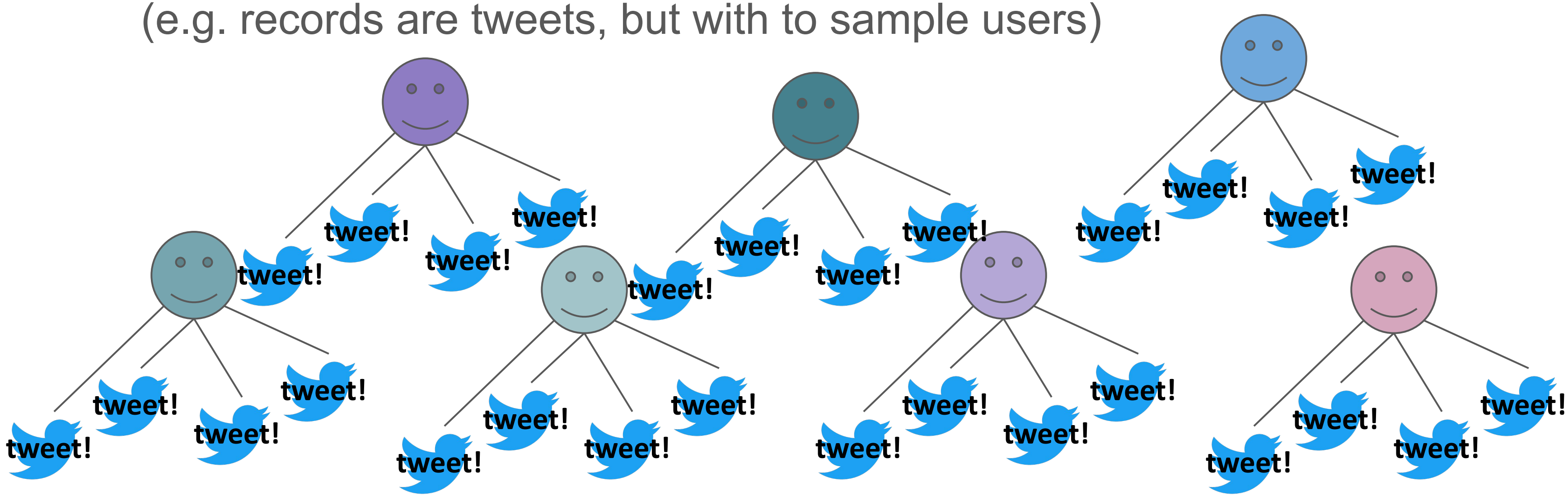
Sampling: 2 Versions

Create a random sample for statistical analysis.

1. **Simple Sampling:** Individual records are what you wish to sample.

2. **Hierarchical Sampling:** Sample an attribute of a record.

   (e.g. records are tweets, but with to sample users)



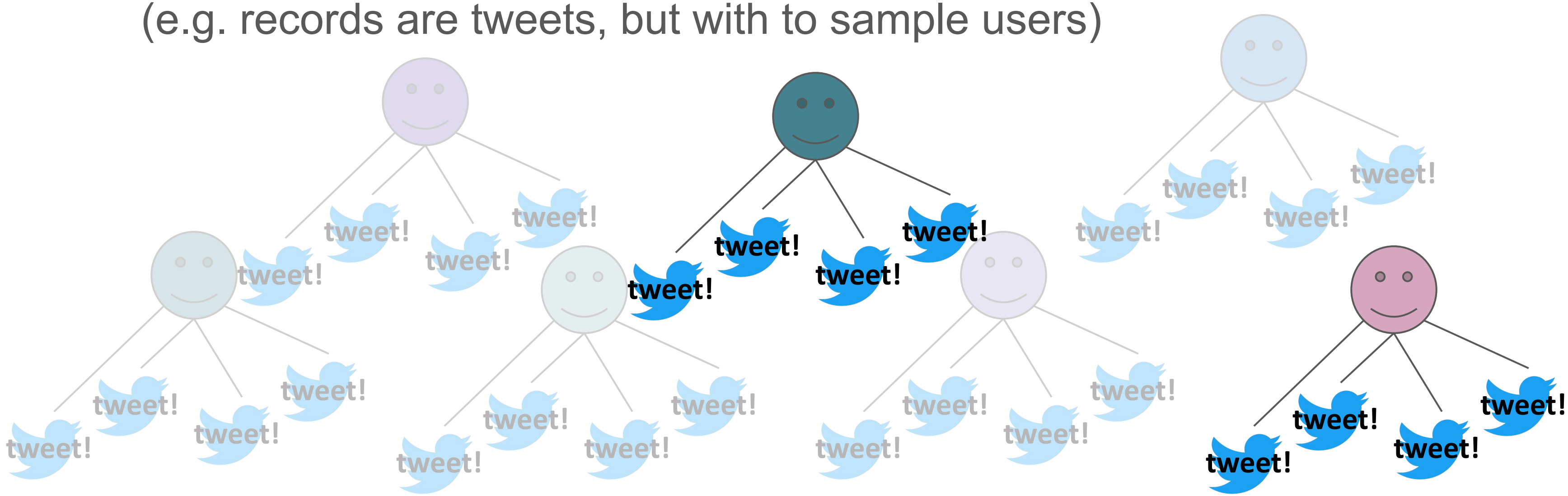
Sampling: 2 Versions

Create a random sample for statistical analysis.

1. **Simple Sampling:** Individual records are what you wish to sample.

2. **Hierarchical Sampling:** Sample an attribute of a record.

   (e.g. records are tweets, but with to sample users)



Sampling

Create a random sample for statistical analysis.

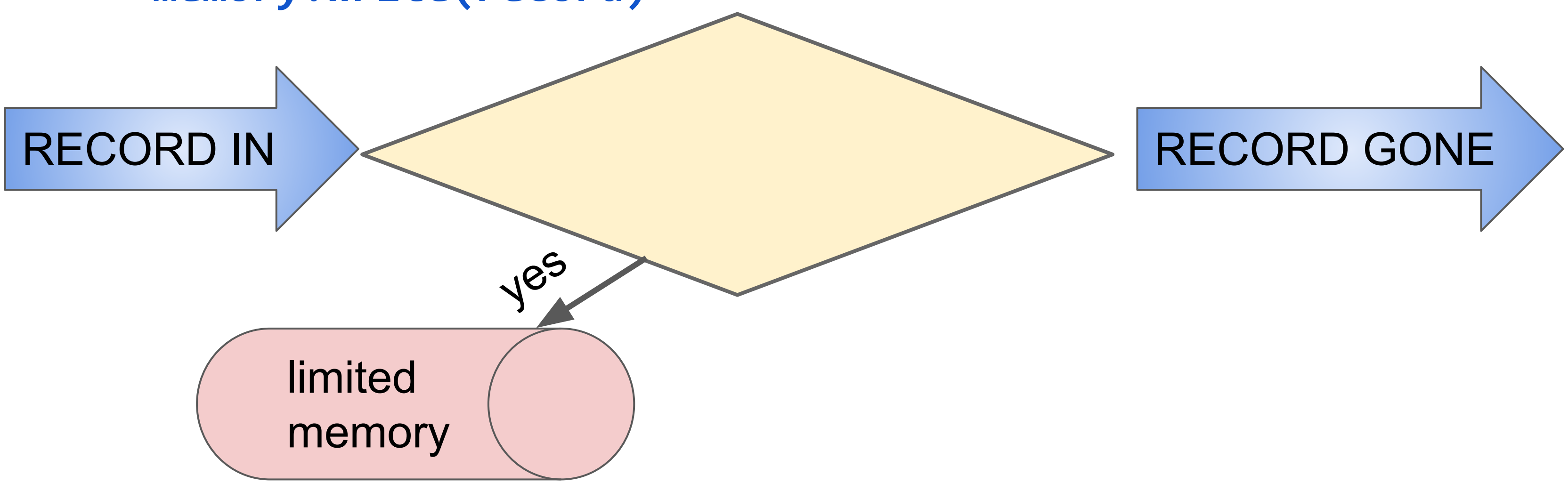
1. **Simple Sampling:** Individual records are what you wish to sample.

Sampling

Create a random sample for statistical analysis.

1. **Simple Sampling:** Individual records are what you wish to sample.

```
record = stream.next()
if ?: #keep: e.g., true 5% of the time
    memory.write(record)
```

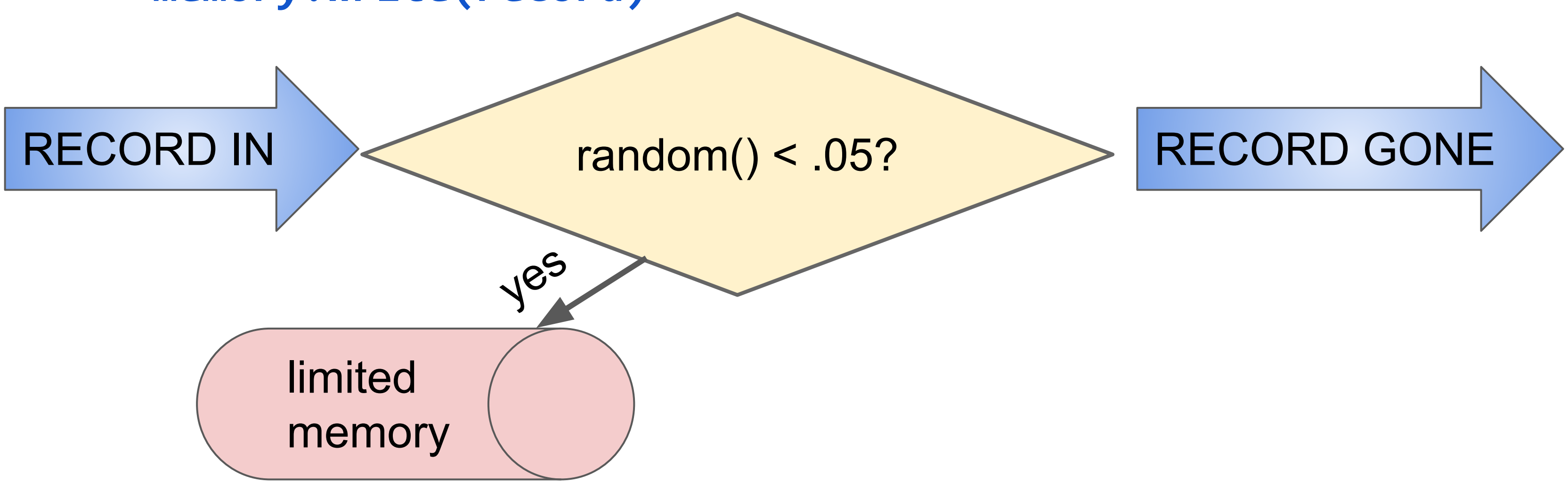


Sampling

Create a random sample for statistical analysis.

1. **Simple Sampling:** Individual records are what you wish to sample.

```
record = stream.next()
if random() <= .05: #keep: true 5% of the time
    memory.write(record)
```



Sampling

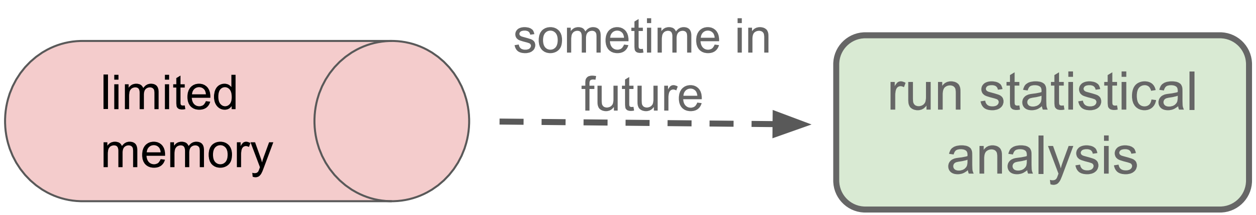
Create a random sample for statistical analysis.

1. **Simple Sampling:** Individual records are what you wish to sample.

```
record = stream.next()
if random() <= .05: #keep: true 5% of the time
    memory.write(record)
```

Problem: records/rows often are not units-of-analysis for statistical analyses

E.g. user_ids for searches, tweets; location_ids for satellite images



Sampling

2. **Hierarchical Sampling:** Sample an attribute of a record.

   (e.g. records are tweets, but with to sample users)

```
record = stream.next()
if random() <= .05: #keep: true 5% of the time
    memory.write(record)
```

   Solution: ?

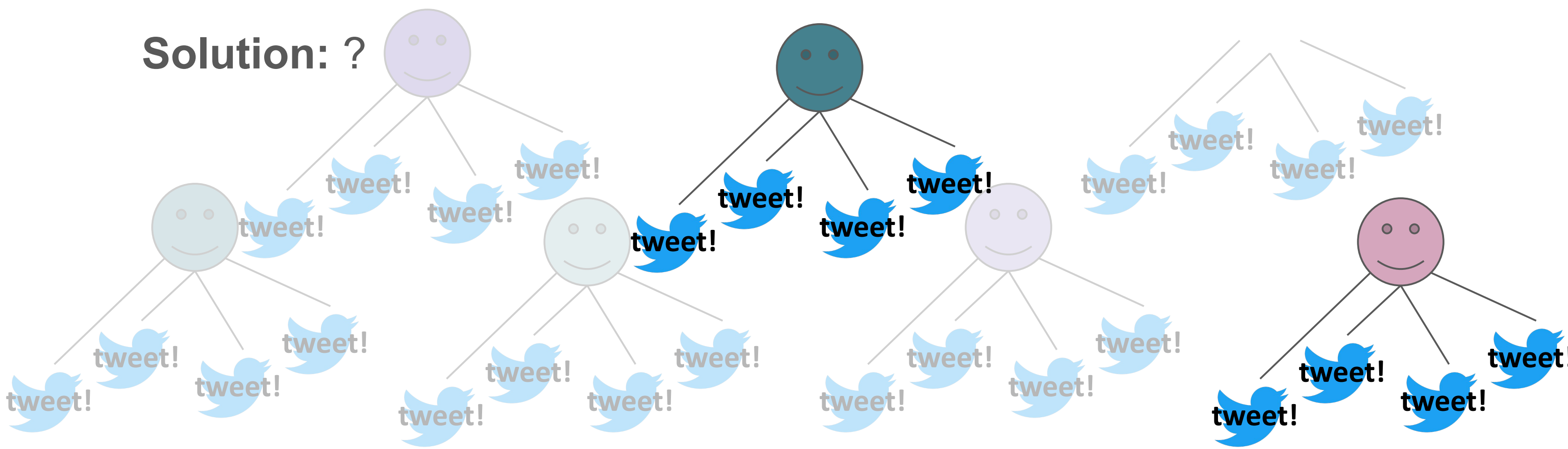
Sampling

2. **Hierarchical Sampling:** Sample an attribute of a record.

   (e.g. records are tweets, but with to sample users)

```
record = stream.next()
if ??: #keep
    memory.write(record)
```

Solution: ?



Sampling

2. **Hierarchical Sampling:** Sample an attribute of a record.

   (e.g. records are tweets, but with to sample users)

```
record = stream.next()
if ??: #keep:
    memory.write(record)
```

   Solution: instead of checking random digit; hash the attribute being sampled.

   – streaming: only need to store hash functions; may be part of standing query

Sampling

2. **Hierarchical Sampling:** Sample an attribute of a record.

   (e.g. records are tweets, but with to sample users)

```
record = stream.next()
if hash(record['user_id']) == 1: #keep
    memory.write(record)
```

Solution: instead of checking random digit; hash the attribute being sampled.

– streaming: only need to store hash functions; may be part of standing query

How many buckets to hash into?

Streaming Topics

- General Stream Processing Model
- Sampling
- Counting Distinct Elements
- Filtering data according to a criteria

Counting Moments

Moments:

- Suppose m_i is the count of distinct element i in the data
- The kth moment of the stream is $\sum_{i \in \mathrm{Set}} m_i^k$

Counting Moments

Moments:

- Suppose m_i is the count of distinct element i in the data
- The kth moment of the stream is $\sum_{i \in \mathrm{Set}} m_i^k$

- 0th moment: count of distinct elements
- 1st moment: length of stream
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

Moments:

- Suppose m_i is the count of distinct element i in the data
- The kth m… m is $\sum_{i \in \text{Set}} m_i^k$

Trivial: just increment a counter

- 0th momen… unt of distinct elements
- **1st moment: length of stream**
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

Applications
Counting…
distinct words in large document.
distinct websites (URLs).
users that visit a site without storing.
unique queries to Alexa.

0th moment

- **0th moment: count of distinct elements**
- 1st moment: length of stream
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

Applications
Counting…
distinct words in large document.
distinct websites (URLs).
users that visit a site without storing.
unique queries to Alexa.

0th moment
One Solution: Just keep a set (hashmap, dictionary, heap)

Problem: Can't maintain that many in memory; disk storage is too slow

- **0th moment: count of distinct elements**
- 1st moment: length of stream
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements observed

pick a hash, *h,* to map each element to $\log_2 n$ bits (buckets)

- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected overestimate of total number of elements observed

pick a hash, *h,* to map each element to $\log_2 n$ bits (buckets)

```
----------------------------------
R = 0 #current max number of zeros at tail
for each stream element, e:
    r(e) = trailZeros(h(e)) #num of trailing 0s from h(e)
    R = r(e) if r[e] > R

estimated_distinct_elements = 2^R
```

- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

Mathematical Intuition

P(trailZeros(*h(e)*) >= *i*) = 2^{-i}

 # P(h(e) == __0) = .5; P(h(e) == __00) = .25; …

P(trailZeros(*h(e)*) < *i*) = $1 - 2^{-i}$

for m elements: $= (1 - 2^{-i})^m$

P(one *e* has trailZeros > i) = $1 - (1 - 2^{-i})^m$

$\approx 1 - e^{-m2^{\wedge}-i}$

If $2^R >> m$, then $1 - (1 - 2^{-i})^m \approx 0$

If $2^R << m$, then $1 - (1 - 2^{-i})^m \approx 1$

0th moment

Streaming Solution: Flajolet-Martin

General idea:

n -- suspected total number of e

pick a hash, *h,* to map each element t (buckets)

```
---------------------------------
R = 0 #current max number of ze      tail
for each stream element, e:
    r(e) = trailZeros(h(e)) #nu    f trailing 0s from h(e)
    R = r(e) if r[e] > R

estimated_distinct_elements = 2^R # m
```

- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Momen

Mathematical Intuition

P(`trailZeros(h(e))` >= *i*) = 2^{-i}

 # P(h(e) == __0) = .5; P(h(e) == __00) = .25; …

P(`trailZeros(h(e))` < *i*) = $1 - 2^{-i}$

for m elements: $= (1 - 2^{-i})^m$

P(one *e* has trailZeros > i) $= 1 - (1 - 2^{-i})^m$

$\approx 1 - e^{-m2^\wedge-i}$

If $2^R \gg m$, then $1 - (1 - 2^{-i})^m \approx 0$

If $2^R \ll m$, then $1 - (1 - 2^{-i})^m \approx 1$

0th moment

Streaming Solution: Flajolet-Martin

General idea:

n -- suspected total number of e

pick a hash, *h,* to map each element tc (buckets)

```
-------------------------------------
R = 0 #current max number of ze
for each stream element, e:
    r(e) = trailZeros(h(e)) #nu
    R = r(e) if r[e] > R

estimated_distinct_elements = 2^R =
```

Problem:

Unstable in practice.

Solution:

Multiple hash functions

but how to combine?

- 2nd moment: sum of squares (measures *uneveness;* related to variance)

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements

pick a hash, *h,* to map each element to l

```
Rs = list()
for h in hashes:
    R = 0 #potential max number of zeros at tail
    for each stream element, e:
        r(e) = trailZeros(h(e)) #num of trailing 0s from h(e)
        R = r(e) if r[e] > R
    Rs.append(2^R)

groupRs = [Rs[i:i+log n] for i in range(0, len(Rs), log n)]

estimated_distinct_elements = median(map(mean, groupRs))
```

Problem:

Unstable in practice.

Solution: Multiple hash functions

1. Partition into groups of size log n
2. Take mean in groups
3. Take median of group means

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements

pick a hash, *h,* to map each element to l

```
Rs = list()
for h in hashes:
    R = 0 #                              eros at tail
    fo
                                    iling 0s from h(e)

    Rs.appe

groupRs = [Rs[i:i+log n] for i in range(0, len(Rs), log n)]

estimated_distinct_elements = median(map(mean, groupRs))
```

Problem:

Unstable in practice.

Solution: Multiple hash functions

1. Partition into groups of size log n
2. Take mean in groups
3. Take median of group means

A good approach anytime one has many “low resolution” estimates of a true value.

Counting Moments

2nd moment
Streaming Solution: Alon-Matias-Szegedy Algorithm

(Exercise; Out of Scope; see in MMDS)

- 0th moment: count of distinct elements
- 1st moment: length of stream
- **2nd moment: sum of squares (measures *uneveness* related to variance)**

Counting Moments

standard deviation
(square-root of variance for numeric data)

$$s = \frac{1}{N}\sqrt{\sum_{1}^{N}(x_i - \bar{x})^2}$$

Counting Moments

standard deviation

(square-root of variance for numeric data)

$$s = \frac{1}{N}\sqrt{\sum_{1}^{N}(x_i - \bar{x})^2} = \sqrt{(\overline{x^2}) - \bar{x}^2} = \sqrt{\frac{\sum x^2}{N} - \left(\frac{\sum x}{N}\right)^2}$$

Counting Moments

standard deviation
(square-root of variance for numeric data)

$$s = \frac{1}{N}\sqrt{\sum_{1}^{N}(x_i - \bar{x})^2} = \sqrt{(\overline{x^2}) - \bar{x}^2} = \sqrt{\frac{\sum x^2}{N} - \left(\frac{\sum x}{N}\right)^2}$$

For streaming, just need to store (1) number of elements, (2) sum of elements, and (3) sum of squares.

Counting Moments

standard deviation
(square-root of variance for numeric data)

$$s = \frac{1}{N}\sqrt{\sum_{1}^{N}(x_i - \bar{x})^2} = \sqrt{(\bar{x^2}) - \bar{x}^2} = \sqrt{\frac{\sum x^2}{N} - \left(\frac{\sum x}{N}\right)^2}$$

However, challenge:
Sum of squares can blow up!

For streaming, just need to store (1) number of elements, (2) sum of elements, and (3) sum of squares.

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam detector

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

Given:

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

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

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

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  #B is a bit vector
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1 #all bits resulting from
```

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam detector

The Bloom Filter (approximates; allows *false positives but not false negatives*)

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  #B is a bit vector
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1 #all bits resulting from
  ... #usually embedded in other code
while key x arrives next in stream #filter:
  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
```

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

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  #B is a bit vector
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1 #all bits resulting from
  ... #usually embedded in other code
while key x arrives next in stream #filter:
  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
```

Setup filter

Apply Filter

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

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 #filter:
  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
```

What is the probability of a *false positive (FP)*?

Q: What fraction of |B| are 1s?

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

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 #filter:
  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
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * k darts at n targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

$= e^{-d/n}$ are **0s**

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

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 #filter:
  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
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * k darts at n targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

= $e^{-d/n}$ are **0s**

= e^{-1}

for large n

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

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 #filter:
  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
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * k darts at n targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

= $e^{-d/n}$ are **0s**

thus, $(1 - e^{-d/n})$ are **1s**

probability all k being 1?

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

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 #filter:
  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
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:
Throw |S| * *k* darts at *n* targets.
1 dart: $1/n$
d darts: $(1 - 1/n)^d$ = prob of 0
$= e^{-d/n}$ are **0s**

thus, $(1 - e^{-d/n})$ are **1s**

probability all *k* being 1?
$$(1 - e^{-(|S|*k)/n})^k$$

|S| size of set
k: number of hash functions
n: number of buckets

Note: Can expand S as stream continues as long as |B| has room (e.g. adding verified email addresses)

(Leskovec et al., 2014)

Side Note on Generating Hash Functions:

What hash functions to use?

Start with 2 decent hash functions

e.g. $h_a(x)$ = ascii(string) % large_prime_number
$h_b(x) = (3*$ascii(string) + $16)$ % large_prime_number

Add together multiplying the second times i:

$h_i(x) = h_a(x) + i*h_b(x)$ % $|BUCKETS|$
$e.g.\ h_5(x) = h_a(x) + 5*h_b(x)$ % 100

https://www.eecs.harvard.edu/~michaelm/postscripts/rsa2008.pdf

Popular choices: md5 (fast, predistable); mmh3 (easy to seed; fast)

Streaming Topics

- General Stream Processing Model
- Sampling
  - approx. random
  - hierarchical approx. random
- Counting Elements
  - distinct elements
  - mean, standard deviation
- Filtering data according to a criteria
  - bloom filter setup + application
  - calculating false positives