

Streaming Algorithms

CSE 545 - Spring 2019

Motivation

One often does not know when a set of data will end.

- Can not store
- Not practical to access repeatedly
- Rapidly arriving
- Does not make sense to ever “insert” into a database

Can not fit on disk but would like to generalize / summarize the data?

Motivation

One often does not know when a set of data will end.

- Can not store
- Not practical to access repeatedly
- Rapidly arriving
- Does not make sense to ever “insert” into a database

Can not fit on disk but would like to generalize / summarize the data?

Examples: Google search queries

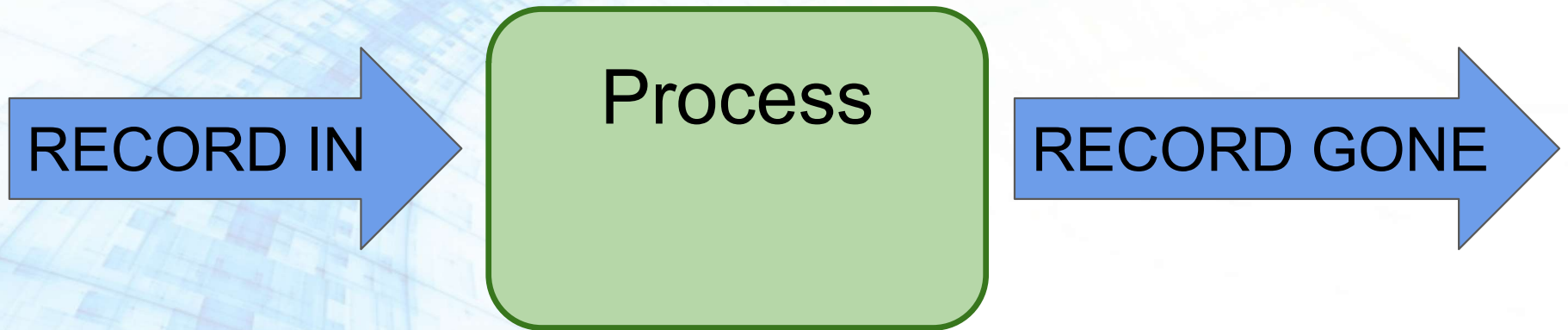
Satellite imagery data

Text Messages, Status updates

Click Streams

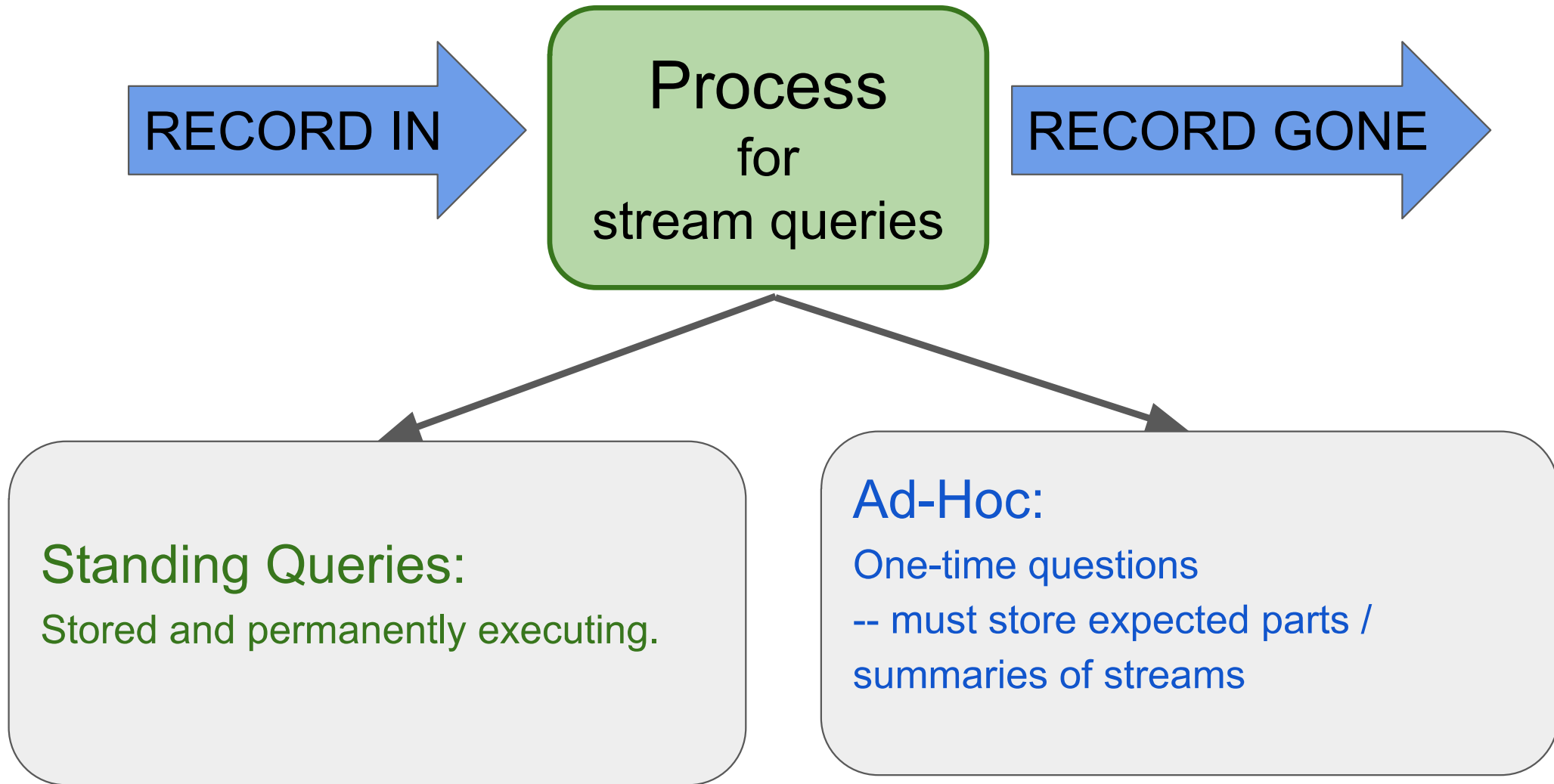
Motivation

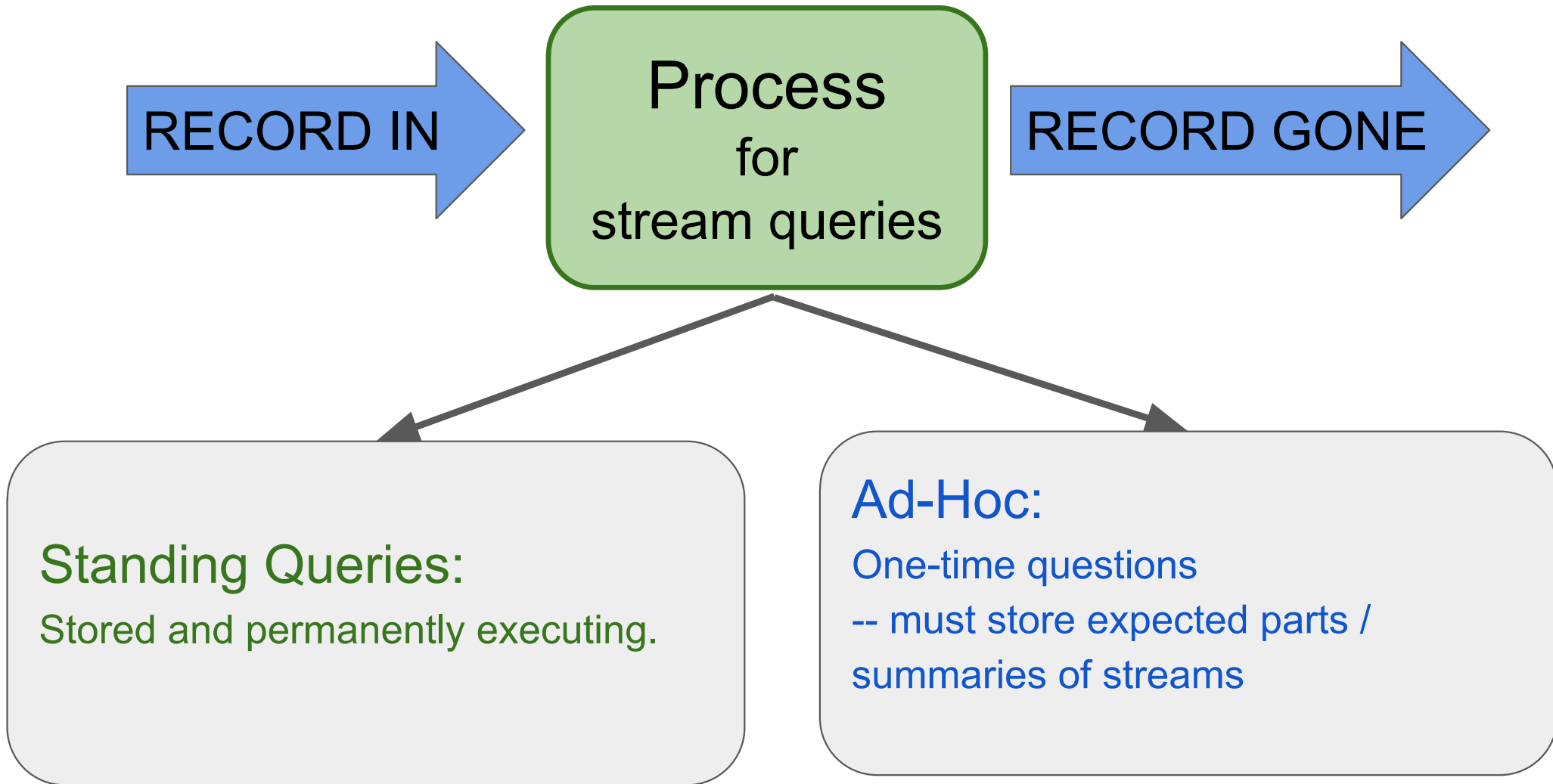
Often translate into $O(N)$ algorithms.



We will cover the following algorithms:

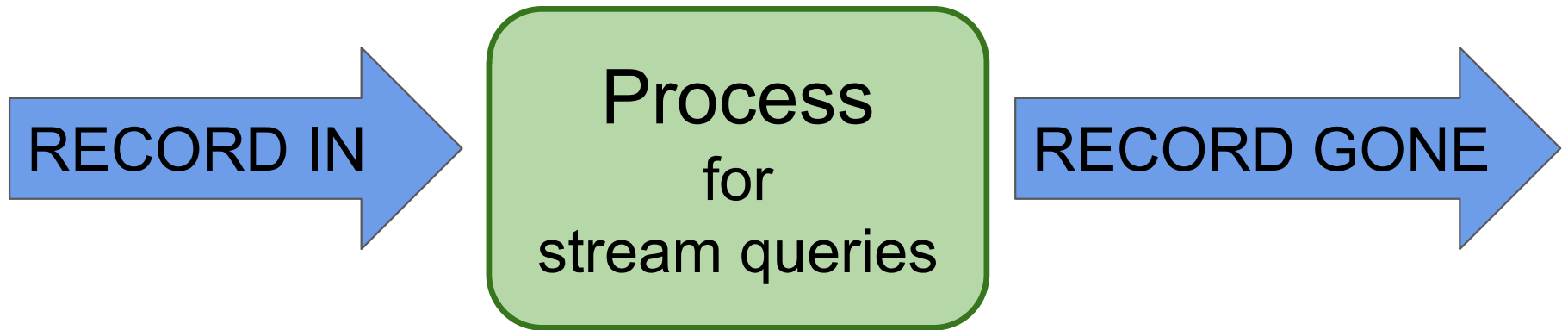
- General Stream Processing Model
- Sampling
- Filtering data according to a criteria
- Counting Distinct Elements





E.g. How would you handle:

What is the mean of values seen so far?

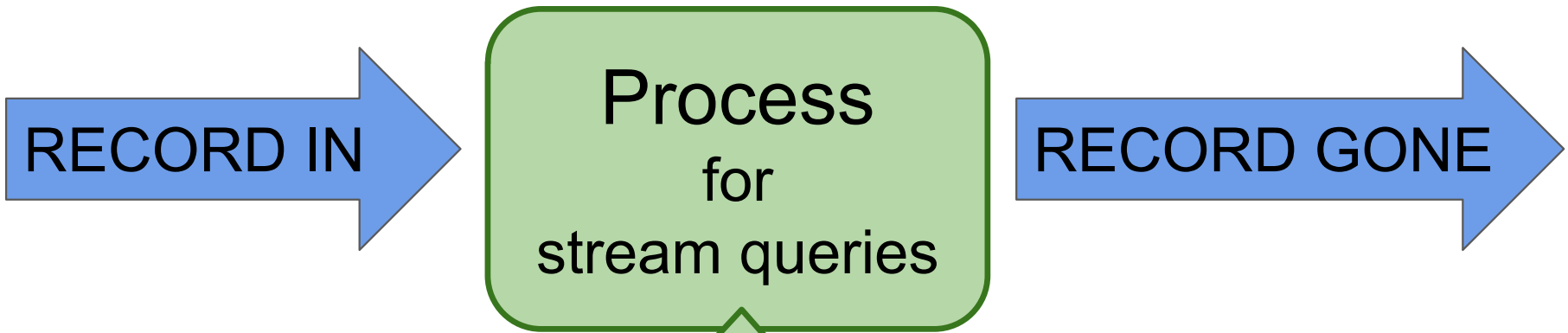


Important difference from typical database management:

- 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 differences:

Management:

Might hold a sliding window of records instead of single record.

- Input is not

.. , i, h, g, f, e, d, c, b, a

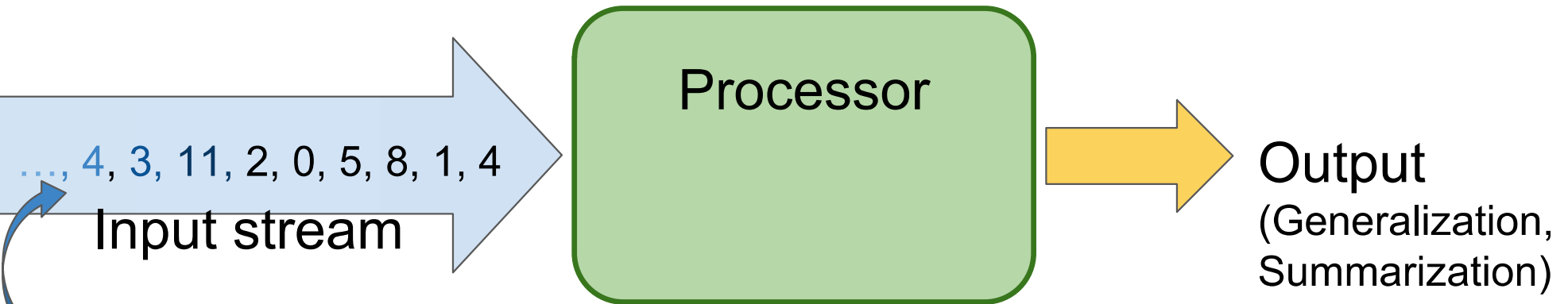
- Input timing/rate is not controlled by users.

E.g. How would you handle:

What is the mean of values seen so far?

General Stream Processing Model

(Leskovec et al., 2014)

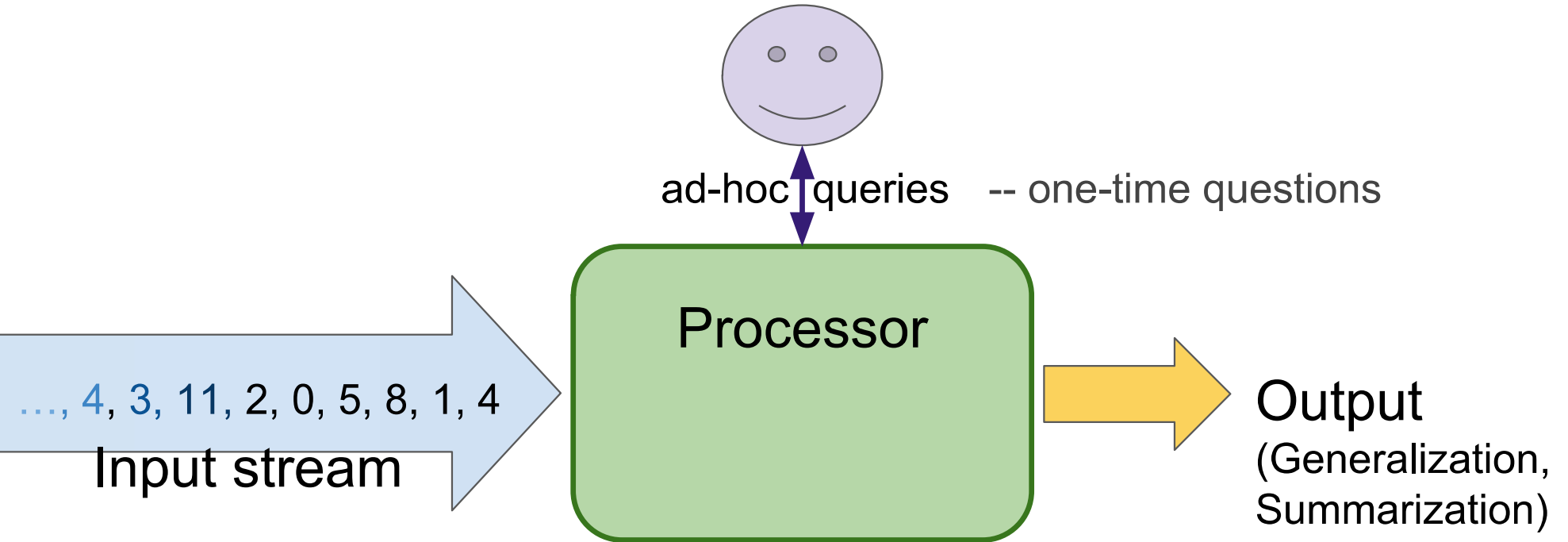


A stream of records

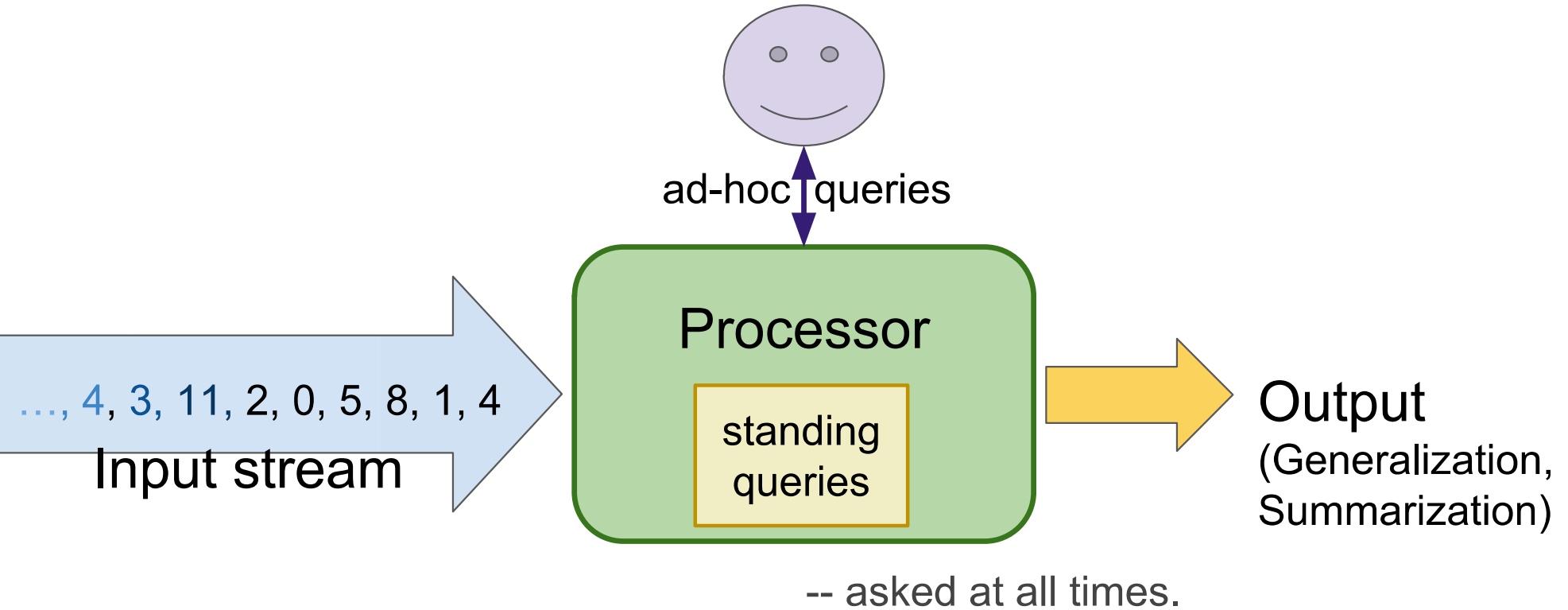
(also often referred to as “elements” or “tuples”)

Theoretically, could be anything! *search queries, numbers, bits, image files, ...*

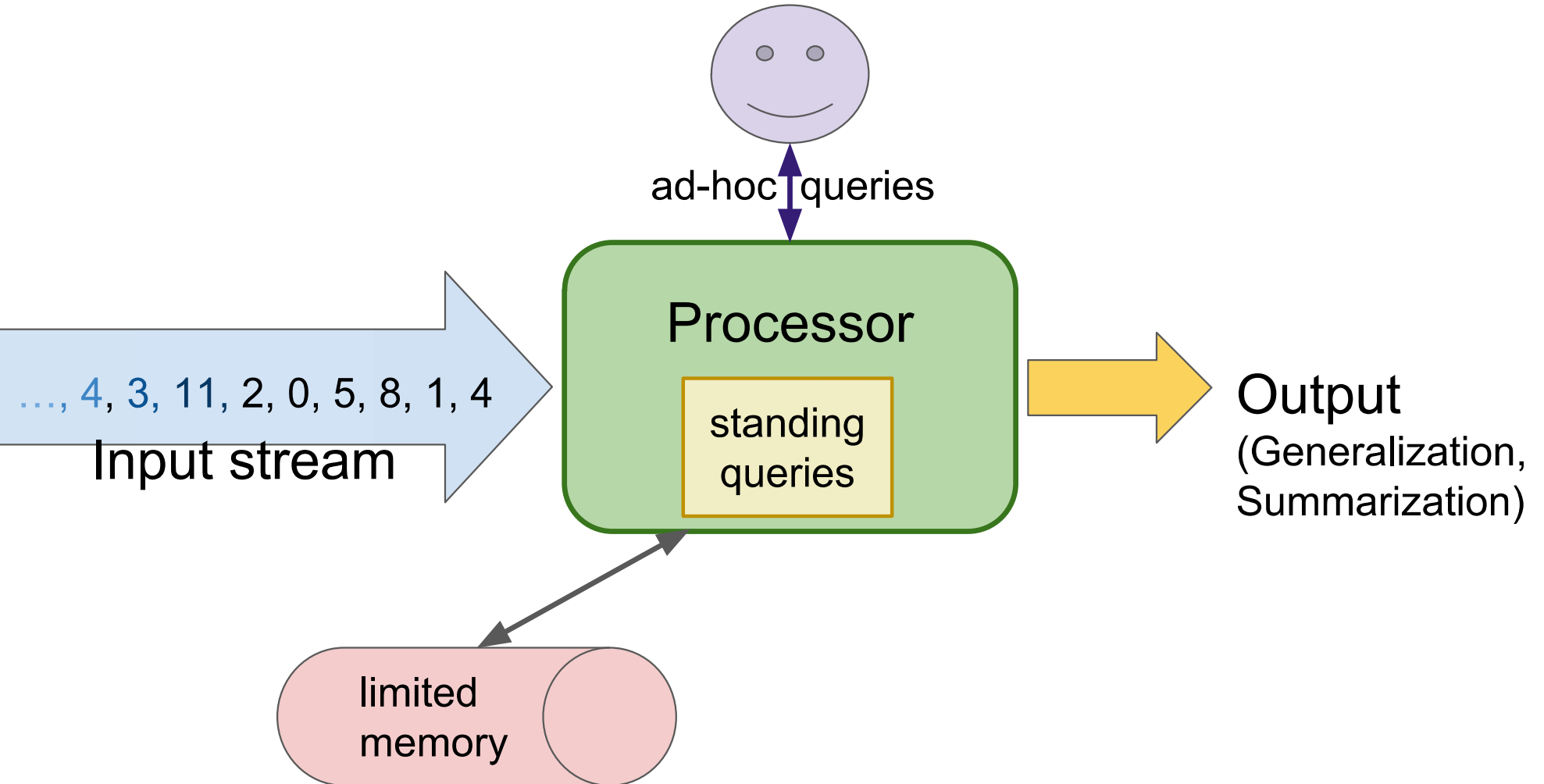
General Stream Processing Model



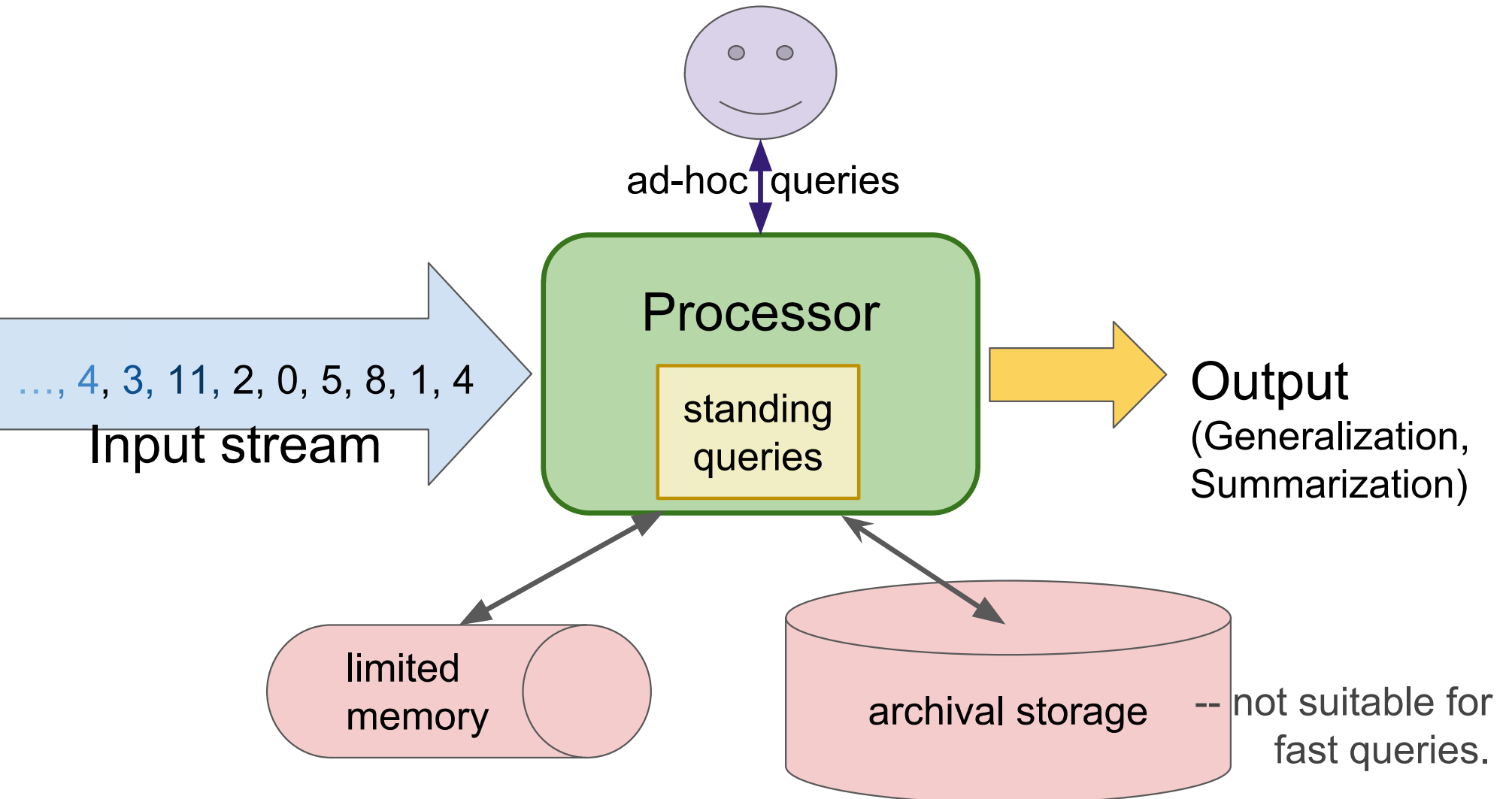
General Stream Processing Model



General Stream Processing Model

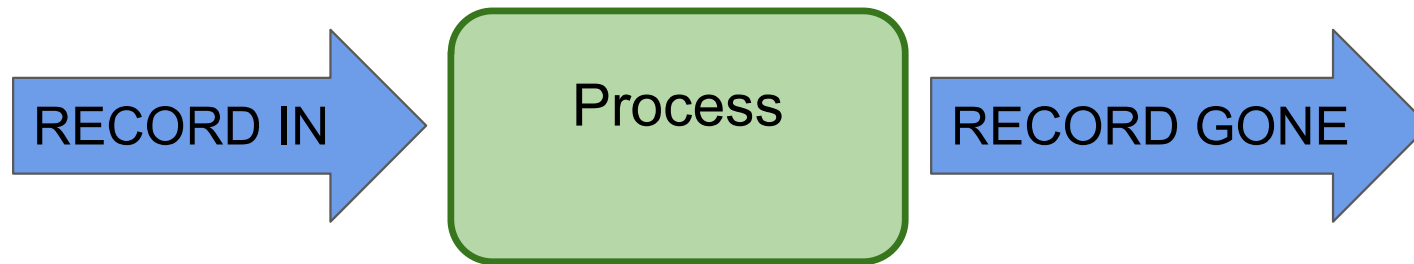


General Stream Processing Model



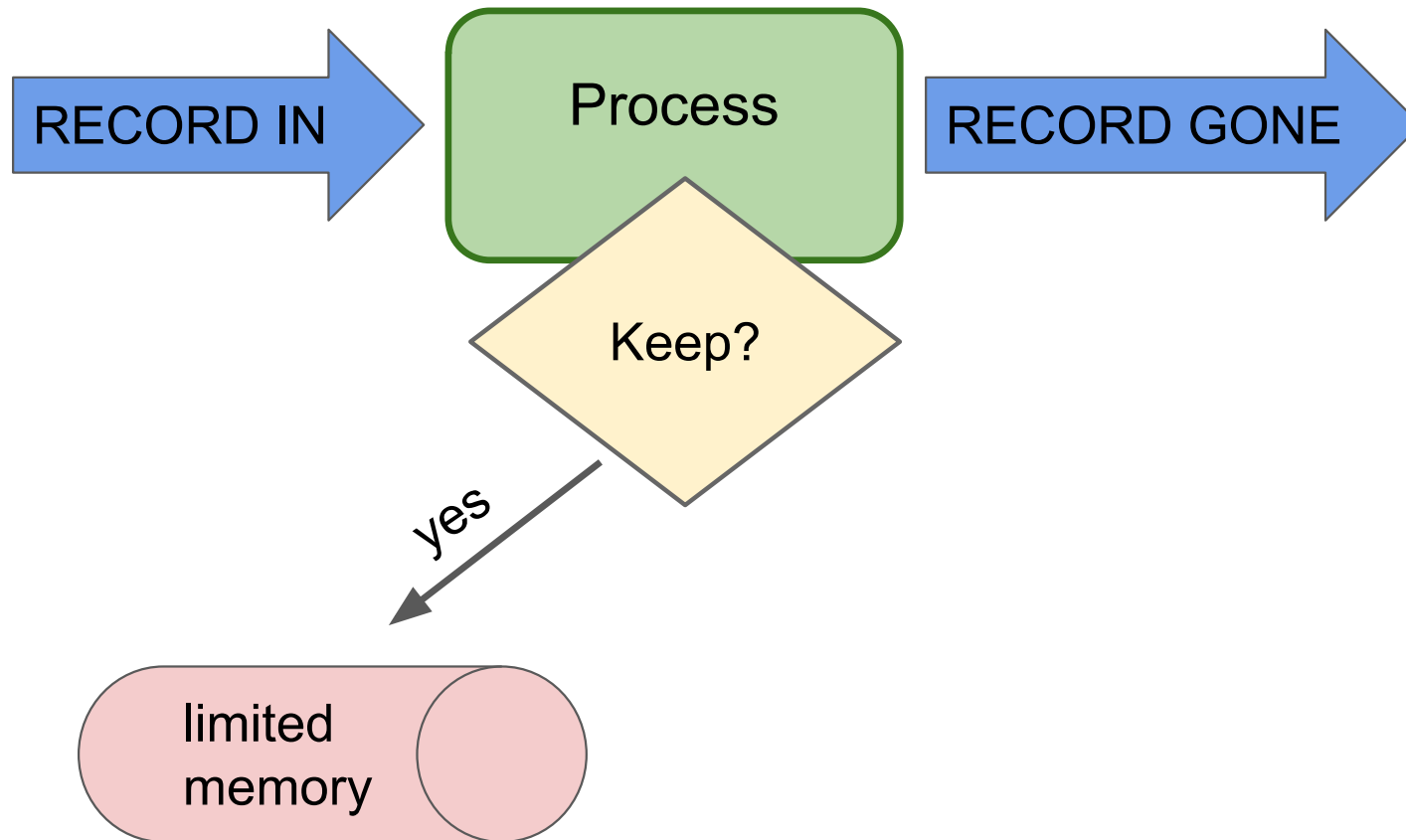
Sampling

Create a random sample for statistical analysis.



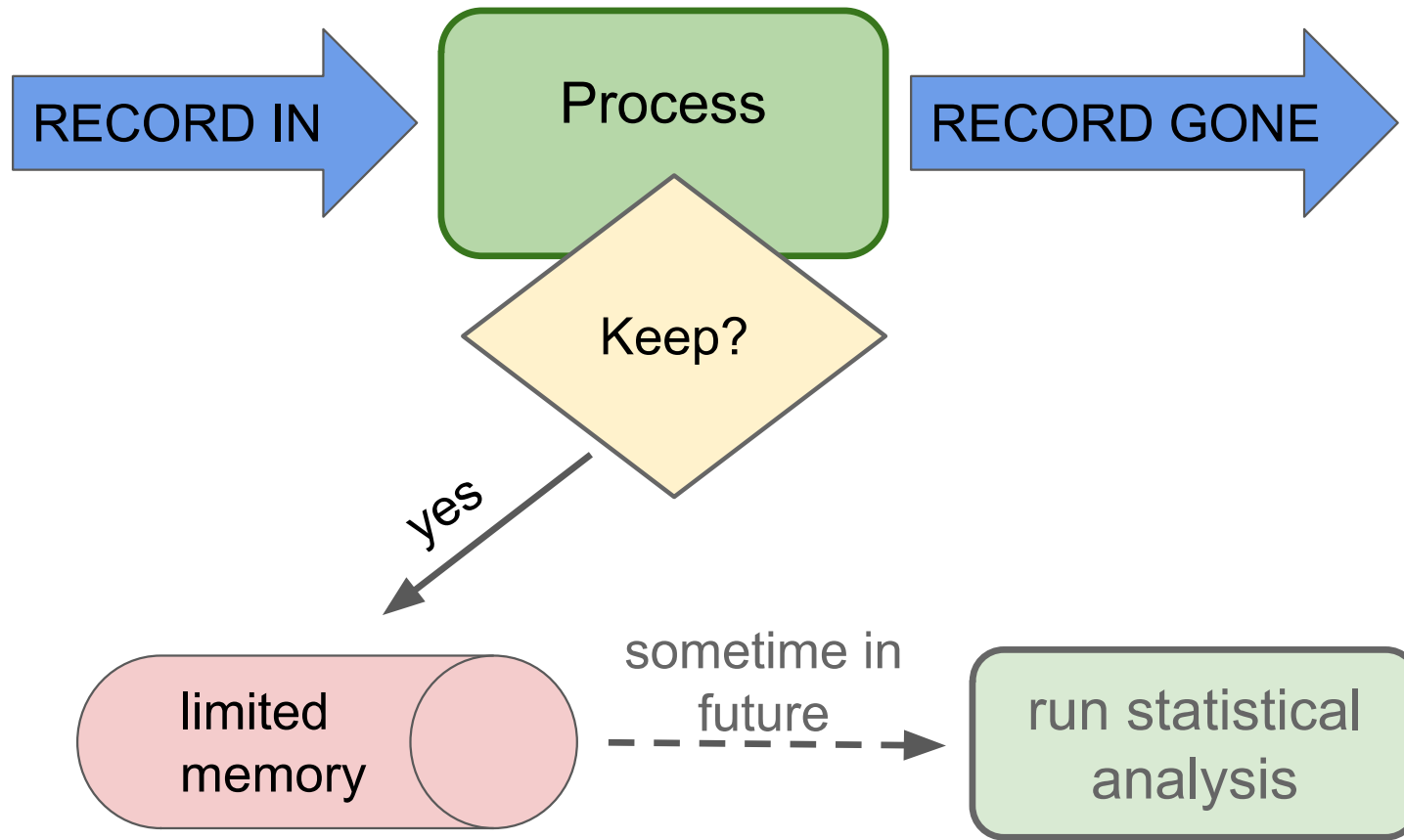
Sampling

Create a random sample for statistical analysis.



Sampling

Create a random sample for statistical analysis.



Sampling

Create a random sample for statistical analysis.

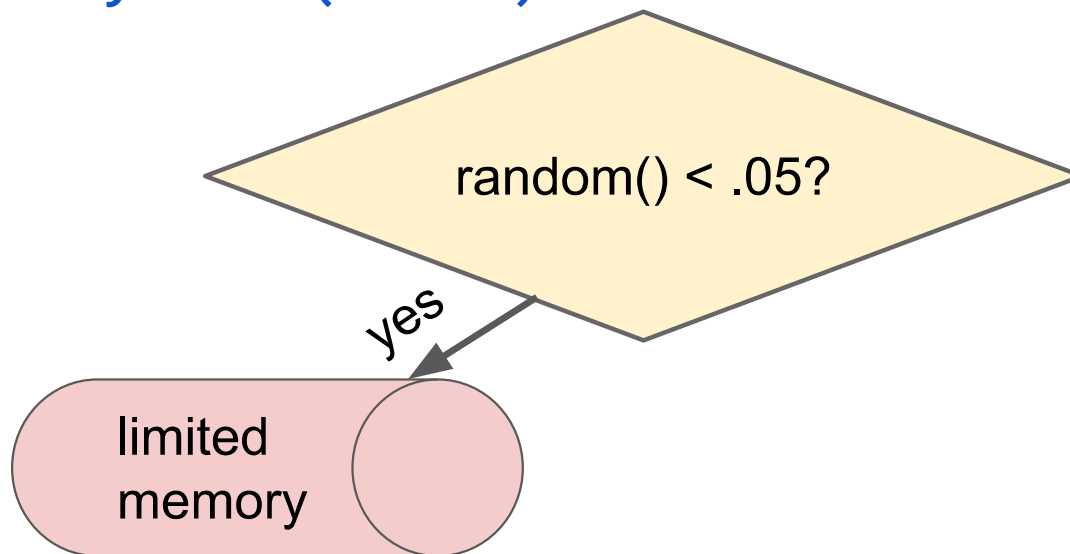
Simple Solution: generate a random number for each arriving record

Sampling

Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

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



Sampling

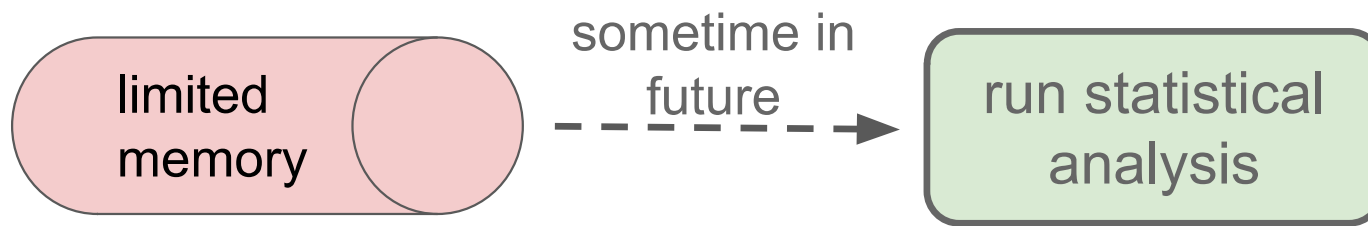
Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
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

Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
record = stream.next()
if random() <= perc: #keep: true perc% 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

Solution: hash into $N = 1/perc$ buckets; designate 1 bucket as “keep”.

```
if hash(record[‘user_id’]) == 1: #keep
```

Sampling

Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
record = stream.next()
if random() <= perc: #keep: true perc% 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

Solution: hash into $N = 1/perc$ buckets; designate 1 bucket as “keep”.

```
if hash(record[‘user_id’]) == 1: #keep
```

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

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

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, \dots, 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, \dots, 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 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, \dots, 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, \dots, h_k independent hash functions

Algorithm:

set all B to \emptyset

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

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, \dots, h_k independent hash functions

Algorithm:

set all B to \emptyset

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

What is the probability of a *false positive*?

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 *F*Ps)

Given:

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

hashes = h_1, h_2, \dots, 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

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: $(1 - 1/n)^d = \text{prob of } 0$
 $= e^{-d/n}$ 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 *F*Ps)

Given:

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

hashes = h_1, h_2, \dots, 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

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: $(1 - 1/n)^d = \text{prob of 0}$
 $= e^{-d/n}$ 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 *F*Ps)

Given:

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

hashes = h_1, h_2, \dots, h_k independent hash functions

Algorithm:

set all B to \emptyset

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

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: $(1 - 1/n)^d = \text{prob of } 0$
 $= e^{-d/n}$ 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 *F*Ps)

Given:

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

hashes = h_1, h_2, \dots, h_k independent hash functions

Algorithm:

set all B to \emptyset

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

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: $(1 - 1/n)^d = \text{prob of } 0$
 $= e^{-d/n}$ are **0s**

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

probability all k being 1?

$(1 - e^{-(|S|*k)/n})^k$

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

(Leskovec et al., 2014)

Counting Moments

Moments:

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

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

Counting Moments

Moments:

- Suppose m_i is the count of distinct element i in the data

- The k th moment $m^{(k)}$ is $\sum_{i \in \text{Set}} m_i^k$

Trivial: just increment
a counter

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

Counting Moments

Applications

Counting...

distinct words in large document.
distinct websites (URLs).
users that visit a site.
unique queries to Alexa.

0th moment

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

Counting Moments

Applications

Counting...

distinct words in large document.
distinct websites (URLs).
users that visit a site.
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 *unevenness*; 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 *unevenness*; 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)

$R = 0$ #potential max number of zeros at tail

for each stream element, e :

$r(e) = \text{trailZeros}(h(e))$ #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 *unevenness*; related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements
pick a hash, h , to map each element to $\log_2 n$ bits (buckets)

 $R = 0$ #potential max number of zeros at tail

for each stream element, e :

$r(e) = \text{trailZeros}(h(e))$ #num of trailing 0s from $h(e)$

$R = r(e)$ if $r[e] > R$

estimated_distinct_elements = 2^R # m

Mathematical Intuition

$$P(\text{trailZeros}(h(e)) \geq i) = 2^{-i}$$

$P(h(e) == _0) = .5$; $P(h(e) == _00) = .25$; ...

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

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

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

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

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

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

- 2nd moment. Sum of squares
(measures *unevenness*; related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements
pick a hash, h , to map each element to $\log_2 n$ bits (buckets)

 $R = 0$ #potential max number of zeros
for each stream element, e :
 $r(e) = \text{trailZeros}(h(e))$ #num of zeros
 $R = r(e)$ if $r[e] > R$

estimated_distinct_elements = 2^R

Mathematical Intuition

$$P(\text{trailZeros}(h(e)) \geq i) = 2^{-i}$$

$P(h(e) == _0) = .5$; $P(h(e) == _00) = .25$; ...

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

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

$$P(\text{one } e \text{ has trailZeros} > i) = 1 - (1 - 2^{-i})^m$$
$$\approx 1 - e^{-m2^{-i}}$$

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

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

Problem:

Unstable in practice.

Solution:

Multiple hash functions
but how to combine?

- 2nd moment. Sum of squares
(measures *unevenness*; 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 k

```
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(2R)
```

```
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 k

```
Rs = list()
```

```
for h in hashes:
```

```
    R = 0
```

```
    for
```

```
Rs.append
```

```
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 *unevenness* related to variance)**