

Data-Intensive Distributed Computing

CS 431/631 451/651 (Winter 2019)

Part 9: Real-Time Data Analytics (2/2)

April 2, 2019

Adam Roegiest

Kira Systems

These slides are available at <http://roegiest.com/bigdata-2019w/>

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details



Since last time...

Storm/Heron

Gives you pipes, but you gotta connect everything up yourself

Spark Streaming

Gives you RDDs, transformations and windowing –
but no event/processing time distinction

Beam

Gives you transformations and windowing, event/processing time distinction –
but too complex

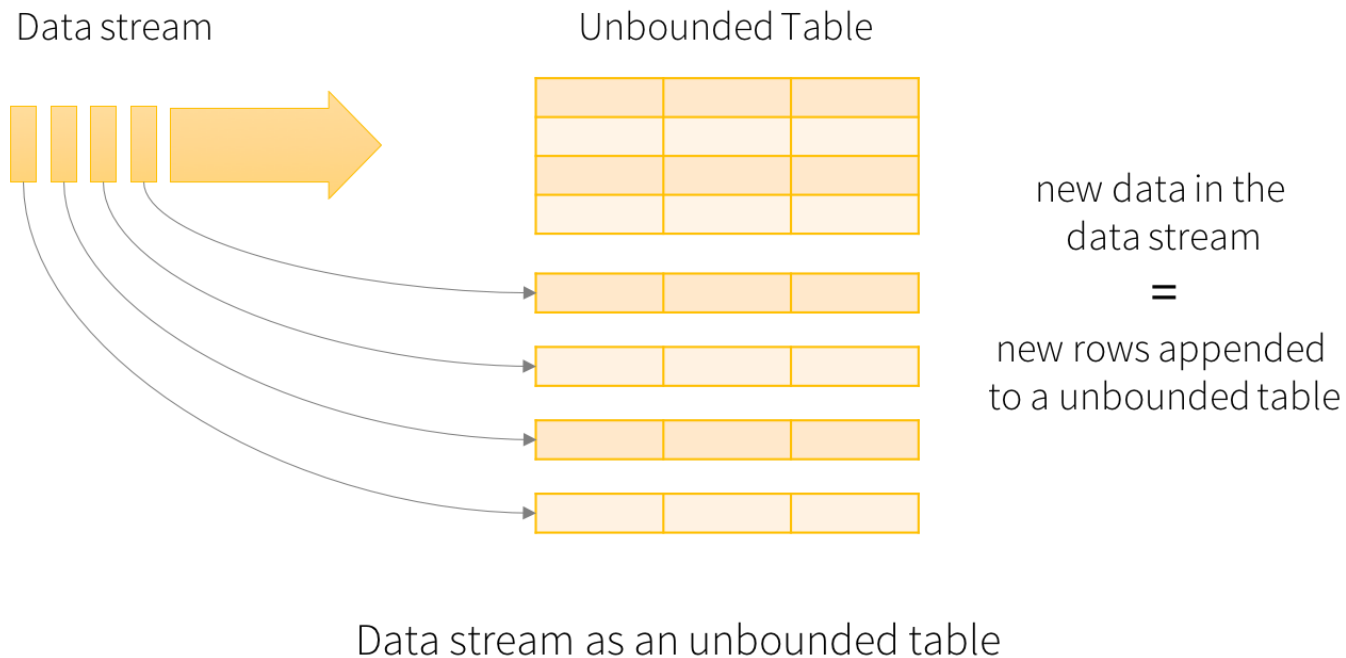
A photograph of a traditional watermill situated in a narrow stone mill race. The mill is constructed from dark wood and is positioned between two stone buildings. Water flows over a series of stone steps, creating a small waterfall. The scene is set in a lush, green environment with trees and foliage in the background. The text "Spark Structured Streaming" is overlaid in white, italicized font at the top center of the image.

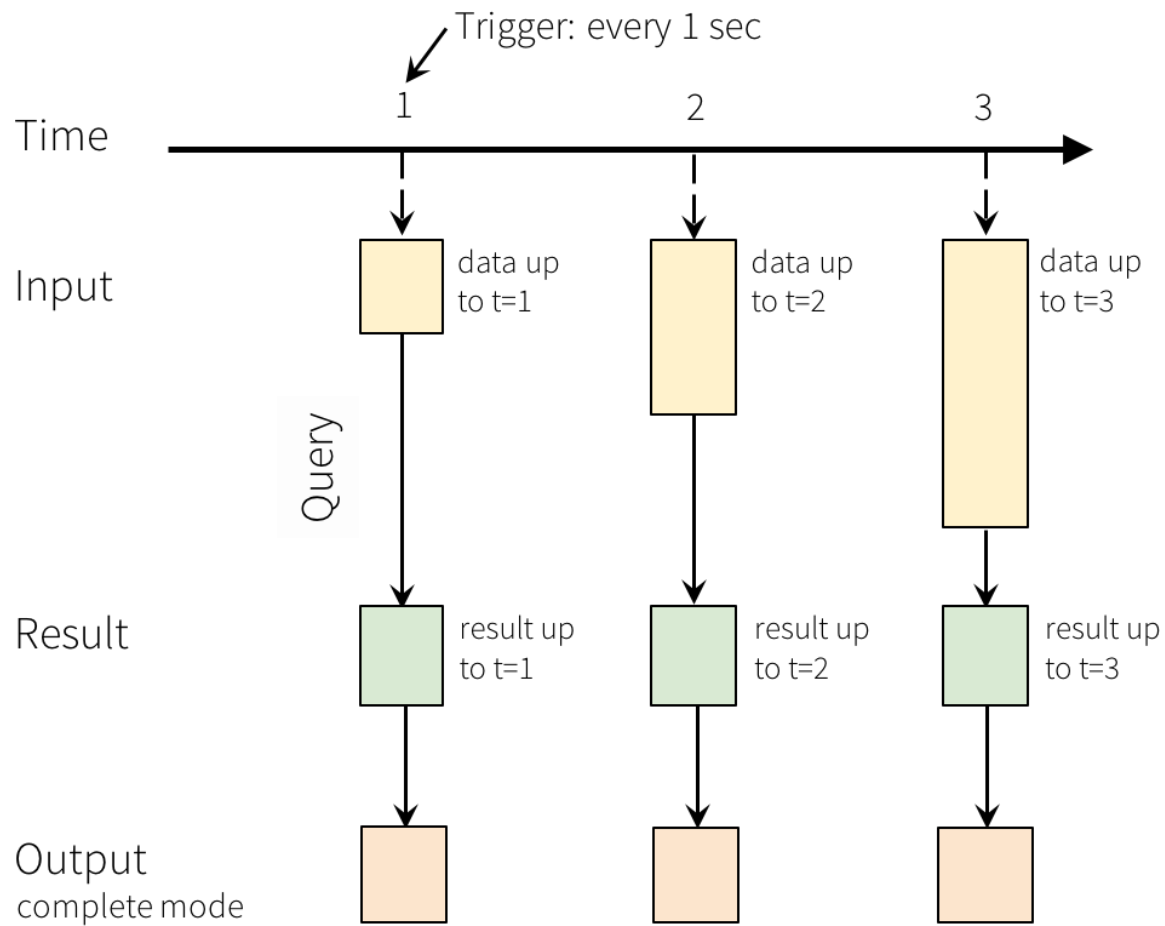
Spark Structured Streaming

Stream Processing Frameworks

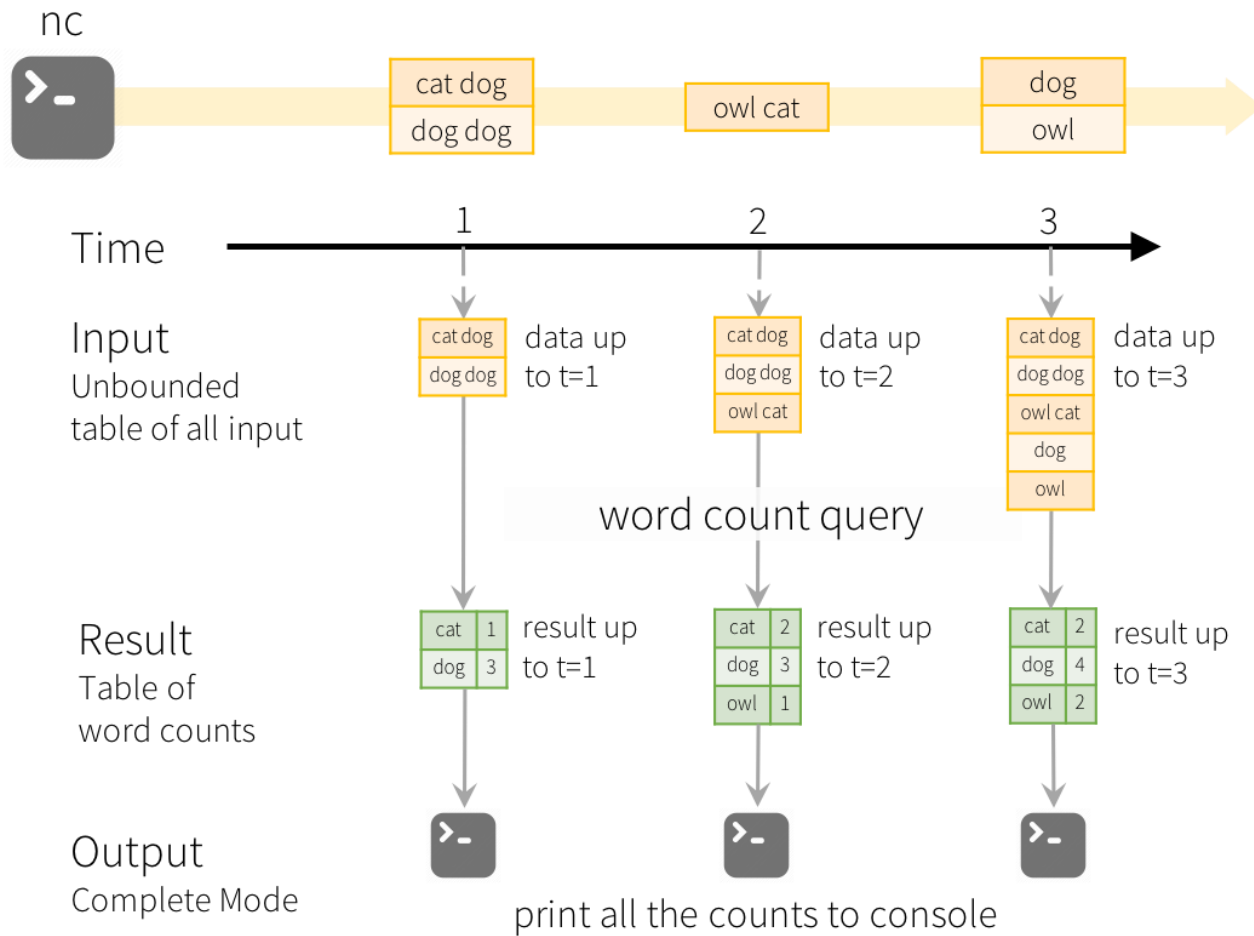
Step 1: From RDDs to DataFrames

Step 2: From bounded to unbounded tables

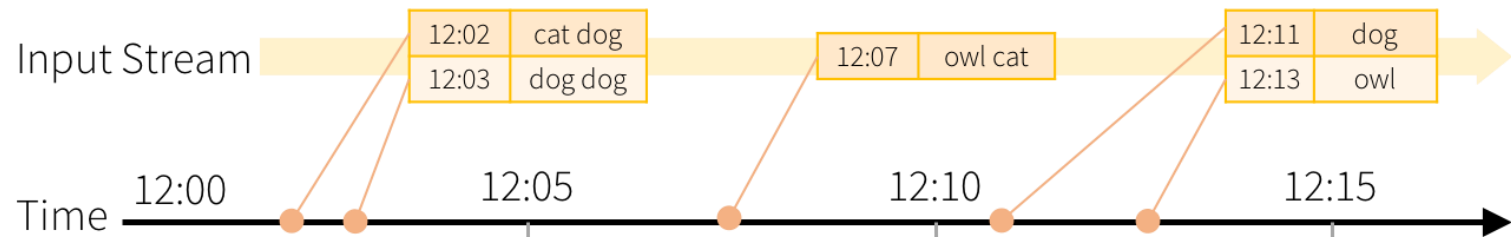




Programming Model for Structured Streaming



Model of the Quick Example



Result Tables
after 5 minute triggers

12:00 - 12:10	cat	1
12:00 - 12:10	dog	3

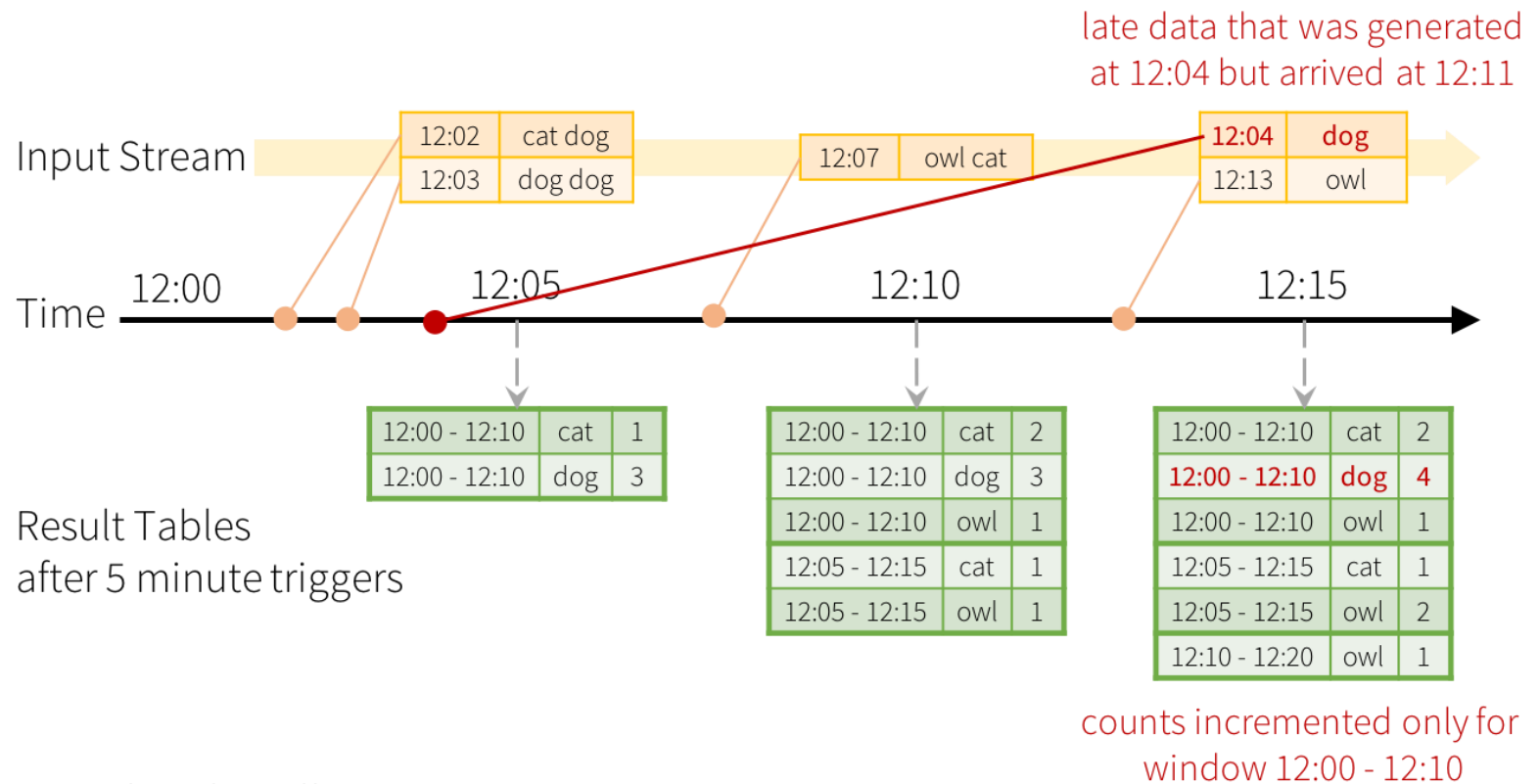
12:00 - 12:10	cat	2
12:00 - 12:10	dog	3
12:00 - 12:10	owl	1
12:05 - 12:15	cat	1
12:05 - 12:15	owl	1

counts incremented for windows
12:00 - 12:10 and 12:05 - 12:15

12:00 - 12:10	cat	2
12:00 - 12:10	dog	3
12:00 - 12:10	owl	1
12:05 - 12:15	cat	1
12:05 - 12:15	owl	2
12:05 - 12:15	dog	1
12:10 - 12:20	dog	1
12:10 - 12:20	owl	1

counts incremented for windows
12:05 - 12:15 and 12:10 - 12:20

Windowed Grouped Aggregation
with 10 min windows, sliding every 5 mins



Late data handling in
Windowed Grouped Aggregation



Interlude

Streams Processing Challenges

Inherent challenges

Latency requirements

Space bounds

System challenges

Bursty behavior and load balancing

Out-of-order message delivery and non-determinism

Consistency semantics (at most once, exactly once, at least once)

Algorithmic Solutions

Throw away data

Sampling

Accepting some approximations

Hashing

Reservoir Sampling

Task: select s elements from a stream of size N with uniform probability

N can be very very large

We might not even know what N is! (infinite stream)

Solution: Reservoir sampling

Store first s elements

For the k -th element thereafter, keep with probability s/k
(randomly discard an existing element)

Example: $s = 10$

Keep first 10 elements

11th element: keep with $10/11$

12th element: keep with $10/12$

...

Reservoir Sampling: How does it work?

Example: $s = 10$

Keep first 10 elements

11th element: keep with $10/11$

If we decide to keep it: sampled uniformly by definition

probability existing item is discarded: $10/11 \times 1/10 = 1/11$

probability existing item survives: $10/11$

General case: at the $(k + 1)$ th element

Probability of selecting each item up until now is s/k

Probability existing item is discarded: $s/(k+1) \times 1/s = 1/(k + 1)$

Probability existing item survives: $k/(k + 1)$

Probability each item survives to $(k + 1)$ th round:

$$(s/k) \times k/(k + 1) = s/(k + 1)$$

Hashing for Three Common Tasks

Cardinality estimation

What's the cardinality of set S ?

How many unique visitors to this page?

HashSet HLL counter

Set membership

Is x a member of set S ?

Has this user seen this ad before?

HashSet Bloom Filter

Frequency estimation

How many times have we observed x ?

How many queries has this user issued?

HashMap CMS

HyperLogLog Counter

Task: cardinality estimation of set
size() → number of unique elements in the set

Observation: hash each item and examine the hash code

On expectation, $1/2$ of the hash codes will start with 0

On expectation, $1/4$ of the hash codes will start with 00

On expectation, $1/8$ of the hash codes will start with 000

On expectation, $1/16$ of the hash codes will start with 0000

...

How do we take advantage of this observation?

Bloom Filters

Task: keep track of set membership

$\text{put}(x) \rightarrow$ insert x into the set

$\text{contains}(x) \rightarrow$ yes if x is a member of the set

Components

m -bit bit vector

k hash functions: $h_1 \dots h_k$



Bloom Filters: put

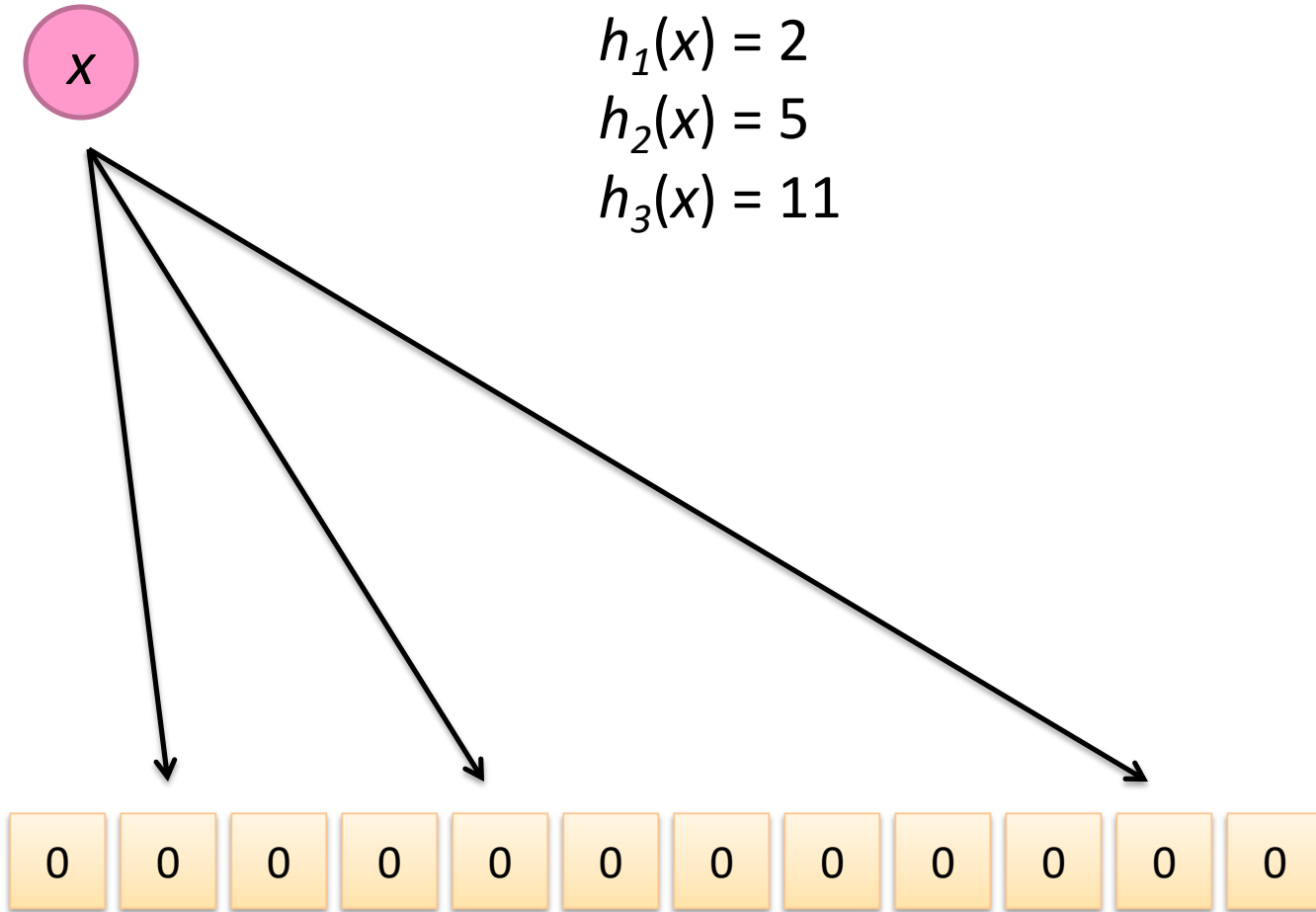
put

X

$$h_1(x) = 2$$

$$h_2(x) = 5$$

$$h_3(x) = 11$$

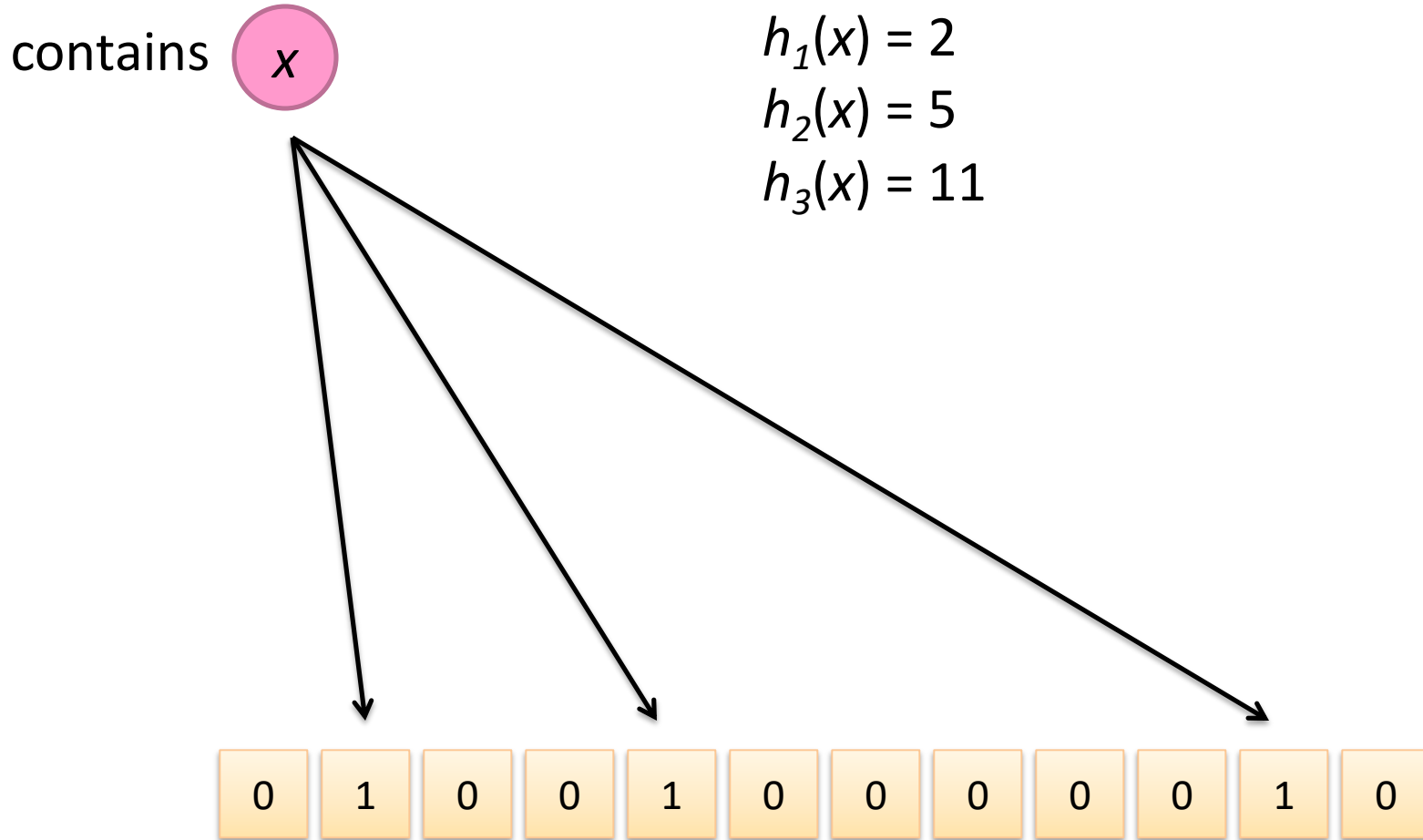


Bloom Filters: put

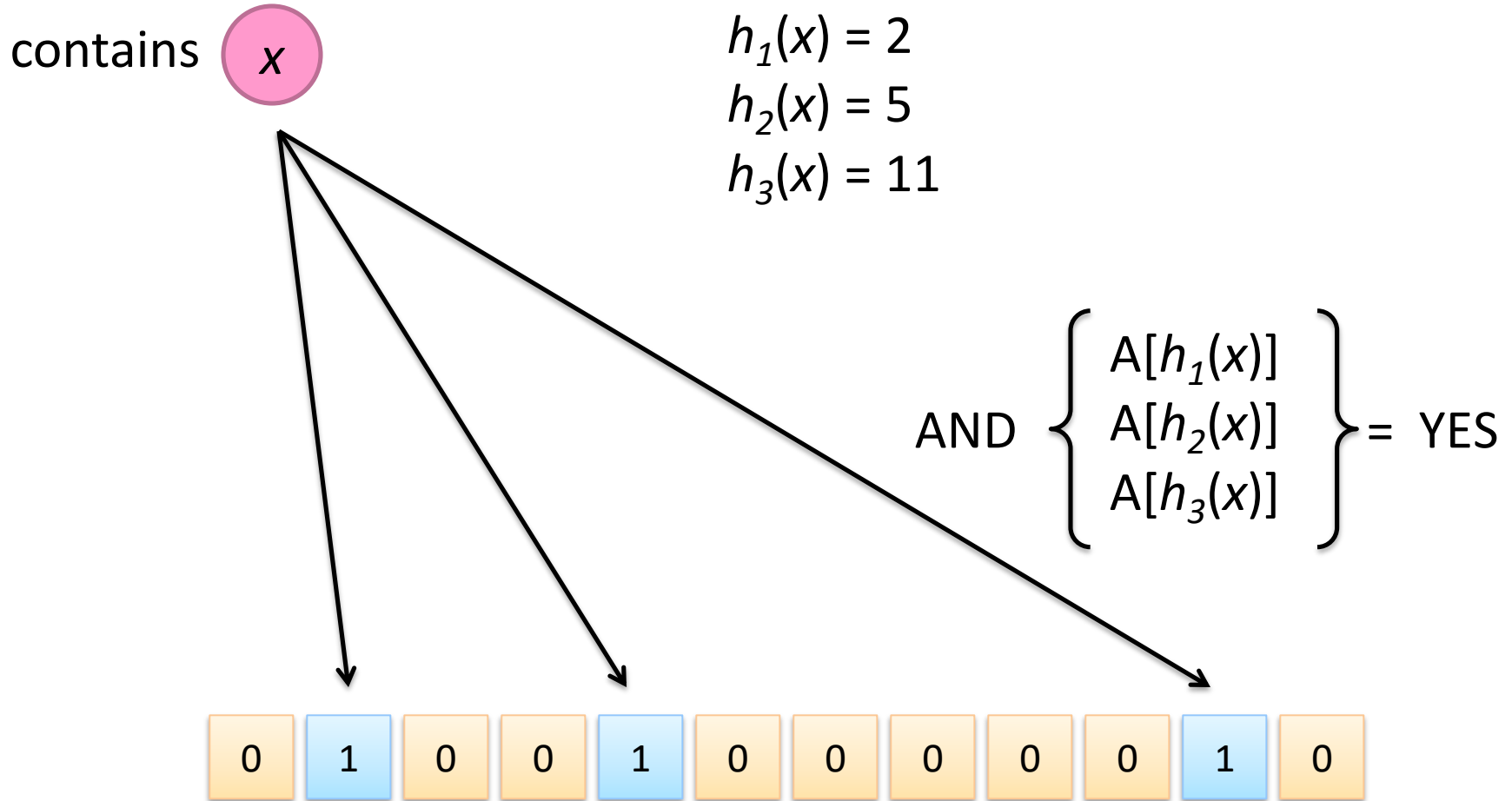
put



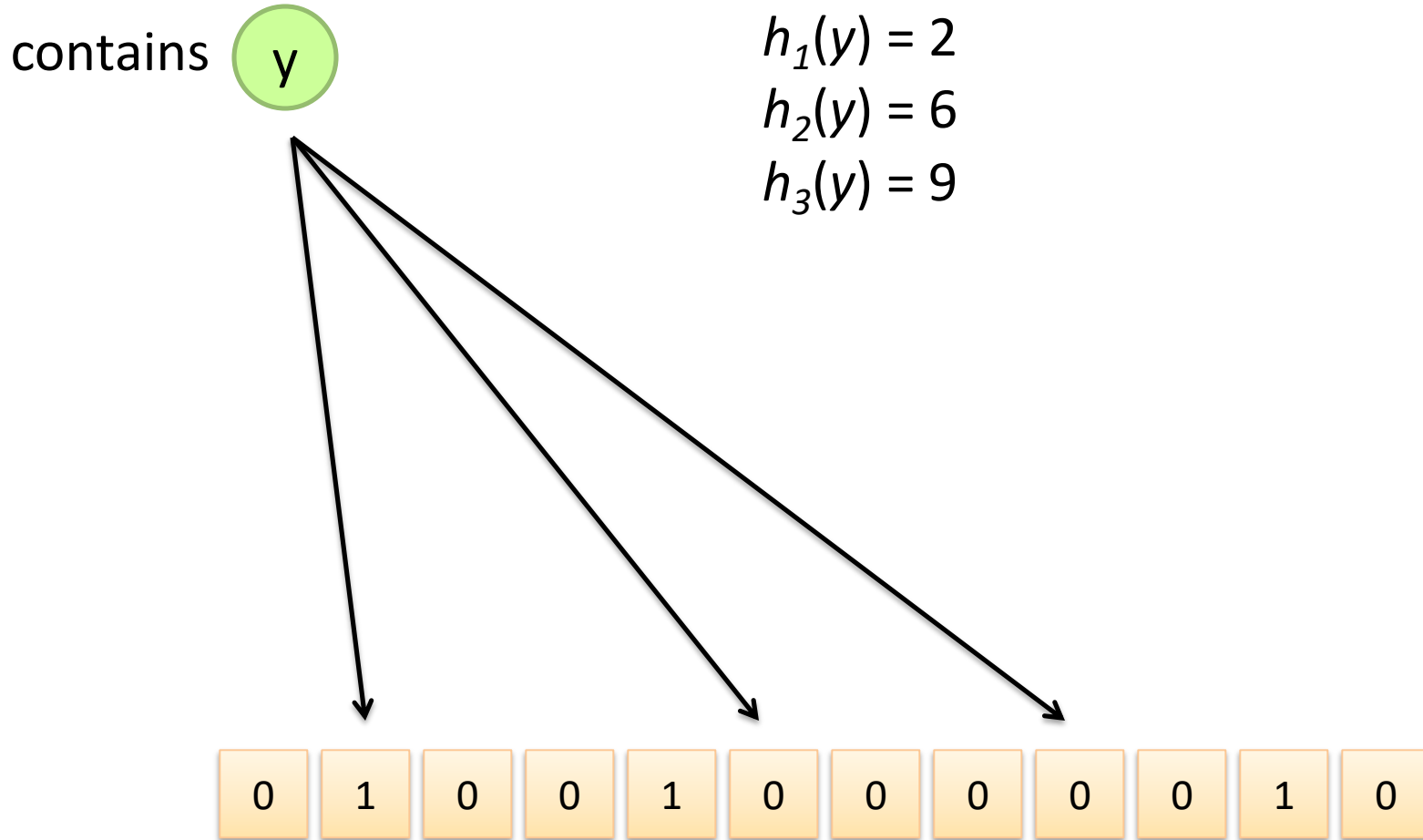
Bloom Filters: contains



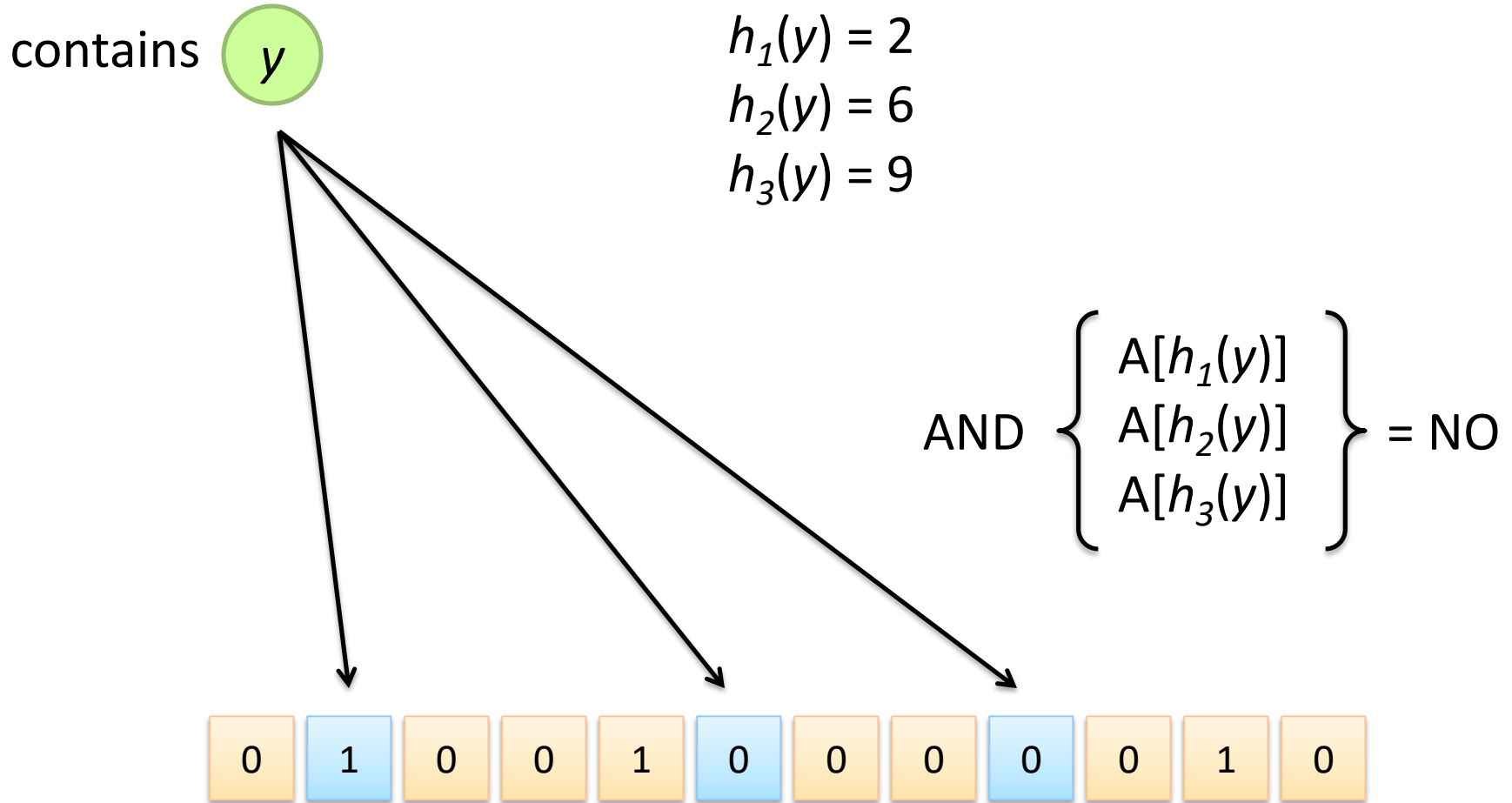
Bloom Filters: contains



Bloom Filters: contains



Bloom Filters: contains



What's going on here?

Bloom Filters

Error properties: contains(x)

False positives possible

No false negatives

Usage

Constraints: capacity, error probability

Tunable parameters: size of bit vector m , number of hash functions k

Count-Min Sketches

Task: frequency estimation

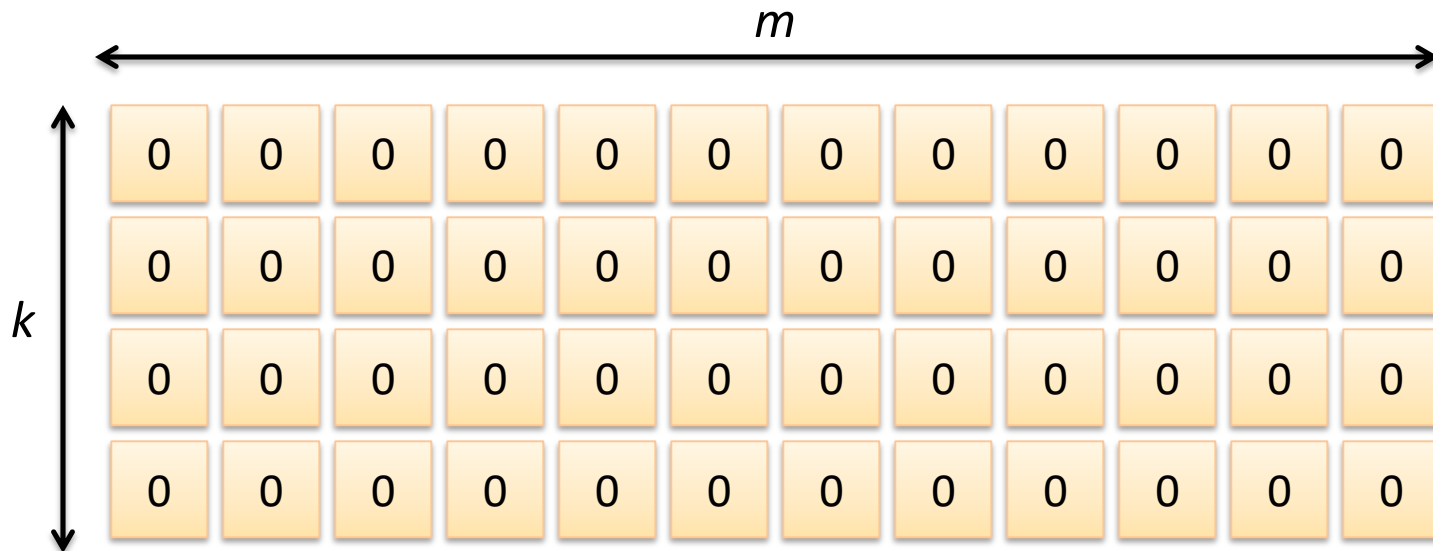
$\text{put}(x) \rightarrow$ increment count of x by one

$\text{get}(x) \rightarrow$ returns the frequency of x

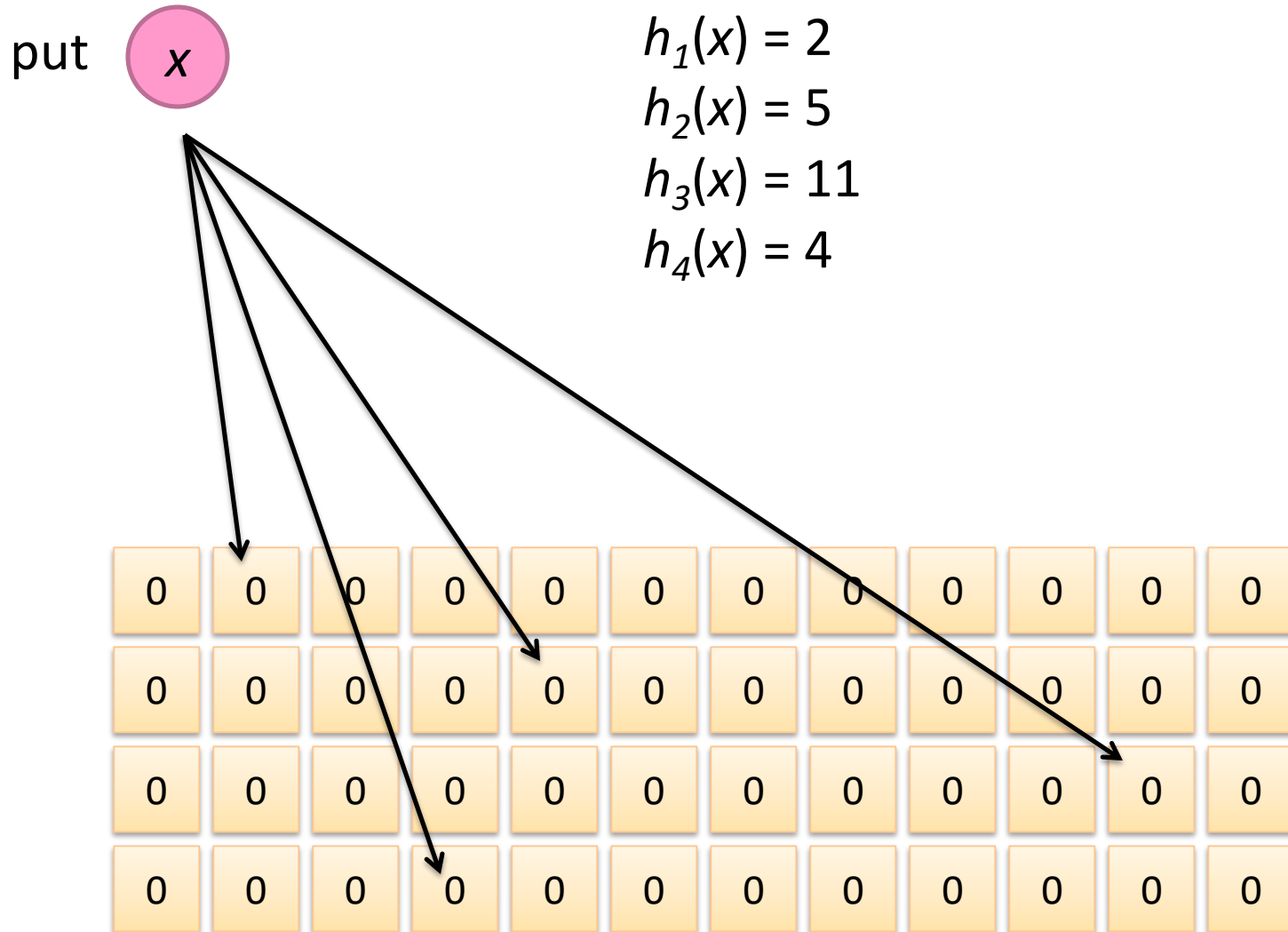
Components

m by k array of counters

k hash functions: $h_1 \dots h_k$

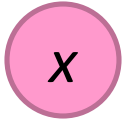


Count-Min Sketches: put



Count-Min Sketches: put

put




0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0

Count-Min Sketches: put

put 

0	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	2	0
0	0	0	2	0	0	0	0	0	0	0	0

Count-Min Sketches: put

put 

0	2	0	0	0	1	0	0	0	0	0	0
0	0	0	0	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	2	1
0	1	0	2	0	0	0	0	0	0	0	0

Count-Min Sketches: get

get

X

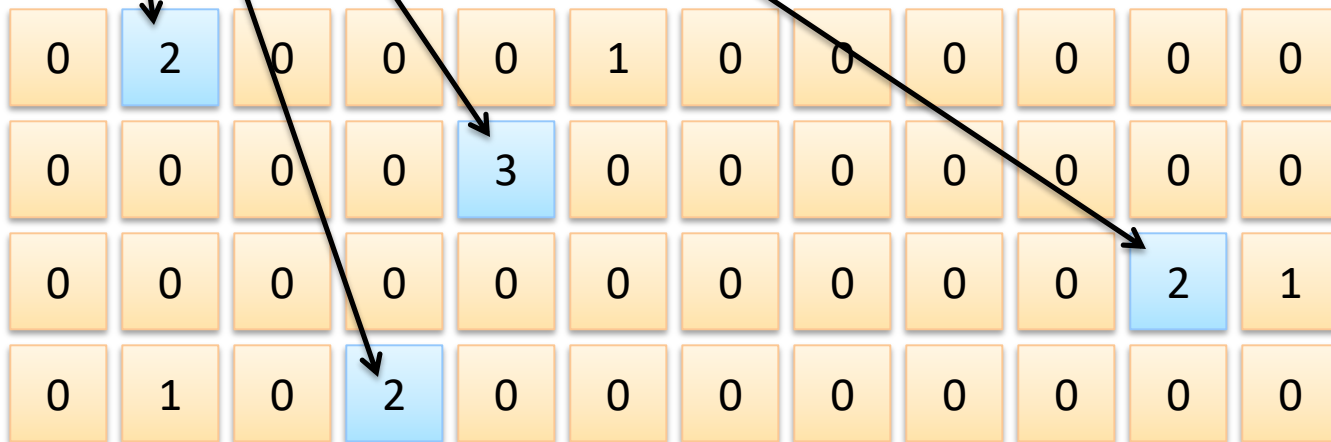
$$h_1(x) = 2$$

$$h_2(x) = 5$$

$$h_3(x) = 11$$

$$h_4(x) = 4$$

$$\text{MIN} \left\{ \begin{array}{c} A[h_1(x)] \\ A[h_2(x)] \\ A[h_3(x)] \\ A[h_4(x)] \end{array} \right\} = 2$$



Count-Min Sketches: get

get

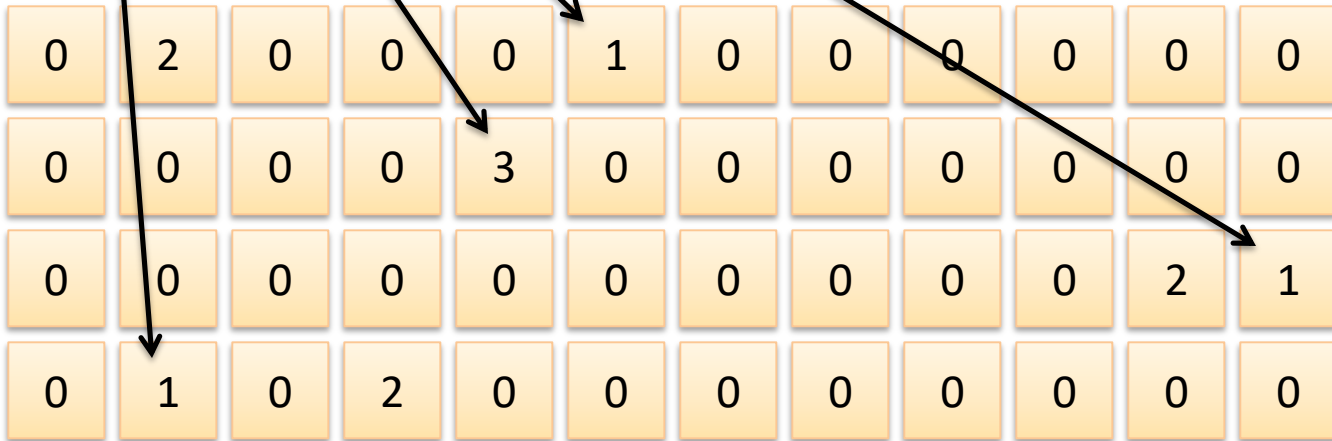


$$h_1(y) = 6$$

$$h_2(y) = 5$$

$$h_3(y) = 12$$

$$h_4(y) = 2$$



Count-Min Sketches

Error properties: $\text{get}(x)$

Reasonable estimation of heavy-hitters

Frequent over-estimation of tail

Usage

Constraints: number of distinct events, distribution of events, error bounds

Tunable parameters: number of counters m and hash functions k , size of counters

Hashing for Three Common Tasks

Cardinality estimation

What's the cardinality of set S ?

How many unique visitors to this page?

HashSet HLL counter

Set membership

Is x a member of set S ?

Has this user seen this ad before?

HashSet Bloom Filter

Frequency estimation

How many times have we observed x ?

How many queries has this user issued?

HashMap CMS



Stream Processing Frameworks



users

Frontend

Backend

Kafka, Heron, Spark
Streaming, Spark
Structured
Streaming, ...

OLTP
database

ETL

(Extract, Transform, and Load)

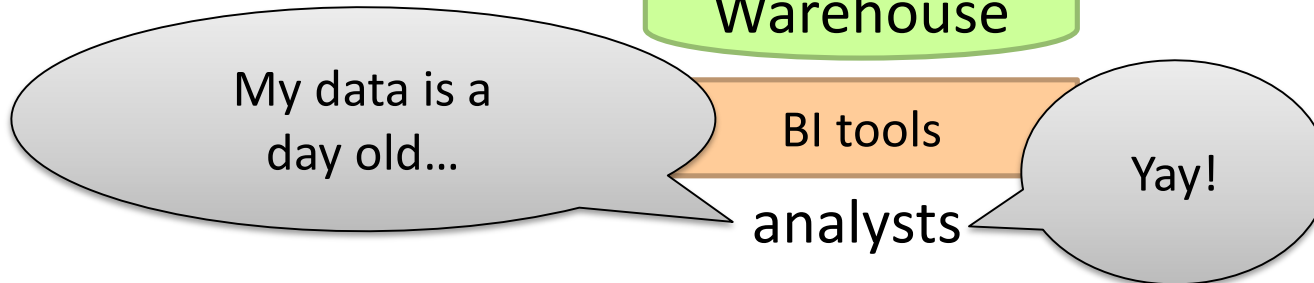
Data
Warehouse

My data is a
day old...

BI tools

analysts

Yay!

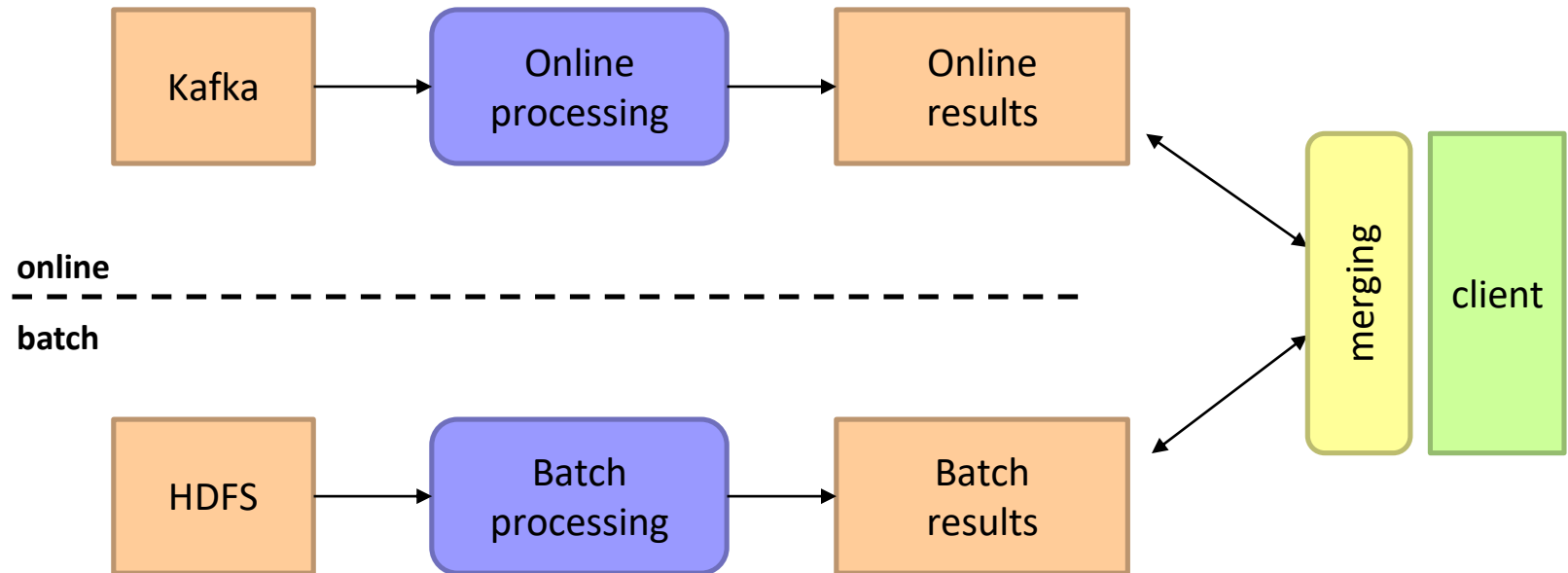


What about our cake?



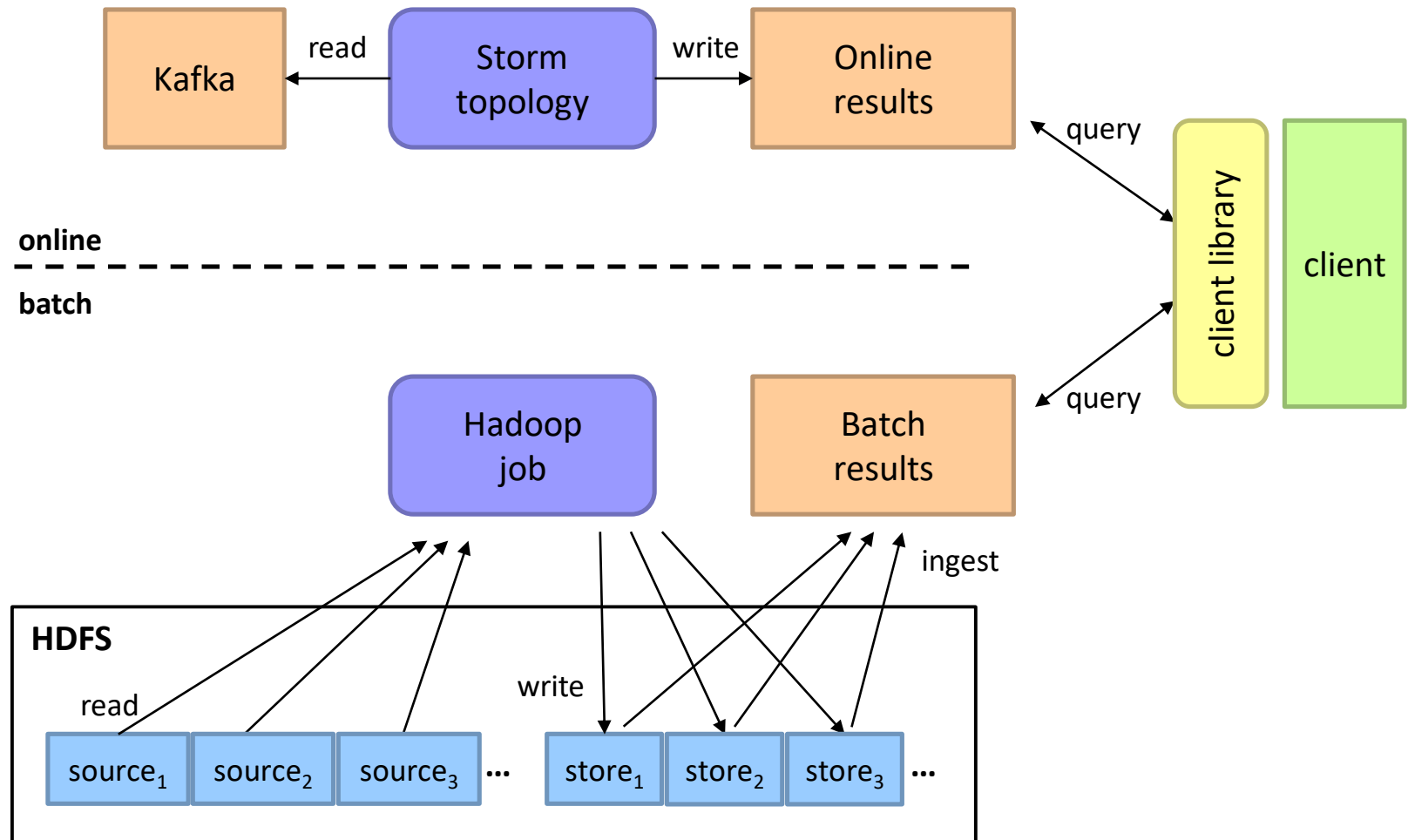
Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time



Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time

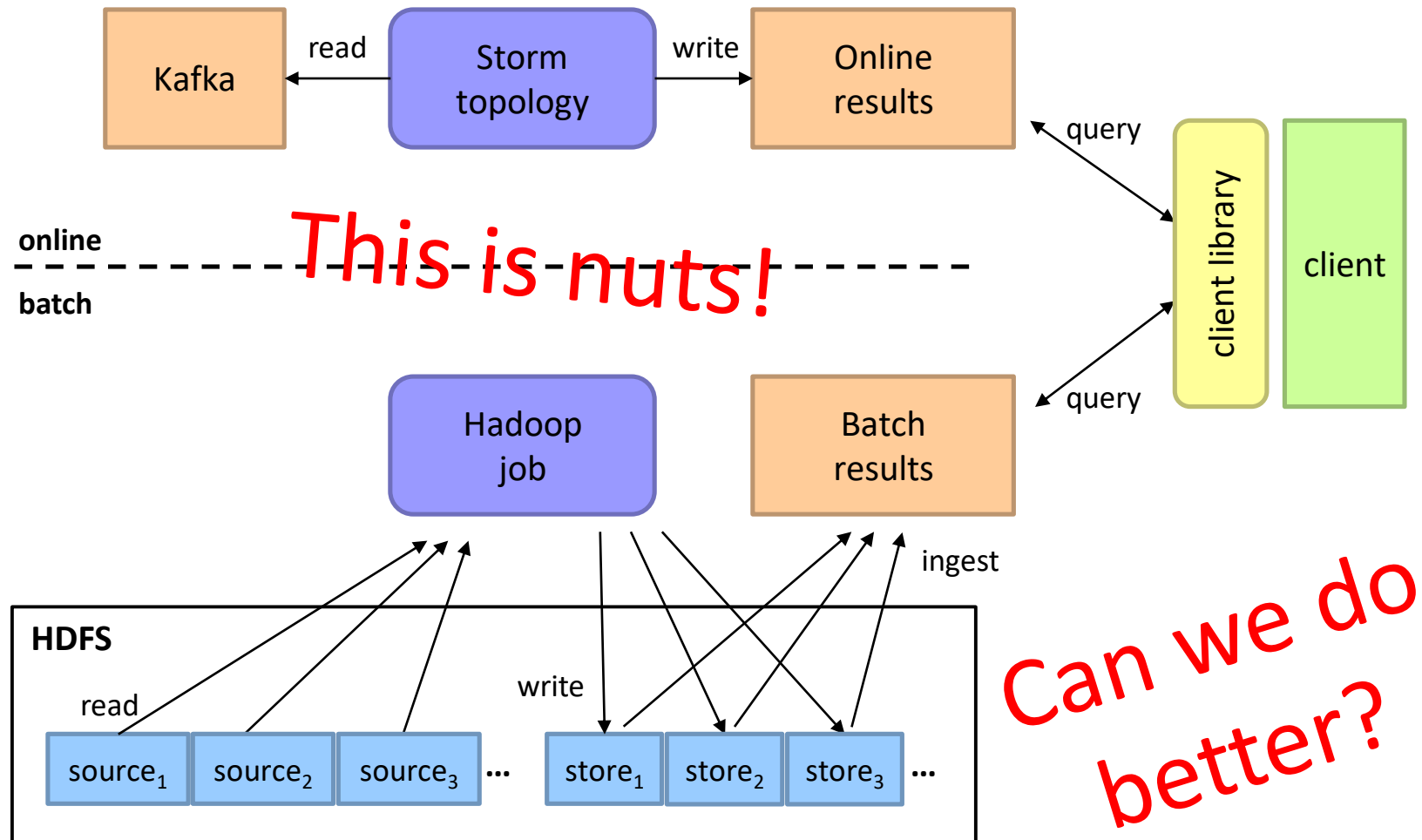


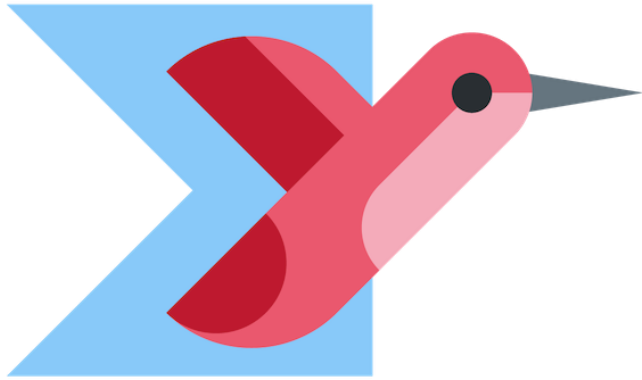
λ

(I hate this.)

Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time





Summingbird

A domain-specific language (in Scala) designed to integrate batch and online MapReduce computations

Idea #1: Algebraic structures provide the basis for seamless integration of batch and online processing

Idea #2: For many tasks, close enough is good enough
Probabilistic data structures as monoids

Batch and Online MapReduce

“map”

`flatMap[T, U](fn: T => List[U]): List[U]`

`map[T, U](fn: T => U): List[U]`

`filter[T](fn: T => Boolean): List[T]`

“reduce”

`sumByKey`

Idea #1: Algebraic structures provide the basis for seamless integration of batch and online processing

Semigroup = (M, \oplus)

$$\oplus : M \times M \rightarrow M, \text{ s.t., } \forall m_1, m_2, m_3 \in M$$

$$(m_1 \oplus m_2) \oplus m_3 = m_1 \oplus (m_2 \oplus m_3)$$

Monoid = Semigroup + identity

$$\varepsilon \text{ s.t., } \varepsilon \oplus m = m \oplus \varepsilon = m, \forall m \in M$$

Commutative Monoid = Monoid + commutativity

$$\forall m_1, m_2 \in M, m_1 \oplus m_2 = m_2 \oplus m_1$$

Simplest example: integers with + (addition)

Idea #1: Algebraic structures provide the basis for seamless integration of batch and online processing

Summingbird values must be at least semigroups
(most are commutative monoids in practice)

Power of associativity =
You can put the parentheses anywhere!

$(a \oplus b \oplus c \oplus d \oplus e \oplus f)$

Batch = Hadoop

$(((((a \oplus b) \oplus c) \oplus d) \oplus e) \oplus f)$

Online = Storm

$((a \oplus b \oplus c) \oplus (d \oplus e \oplus f))$

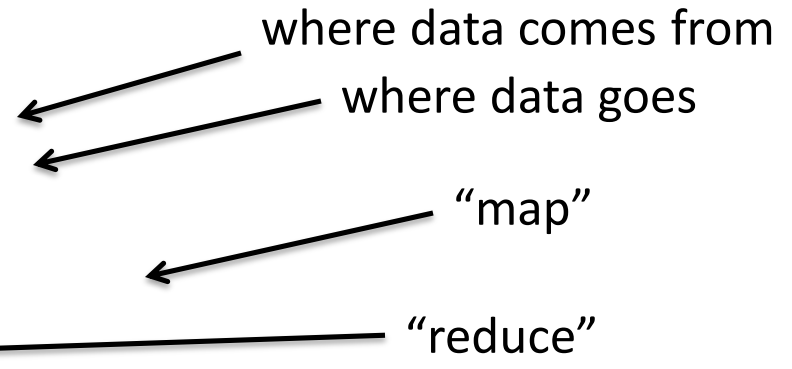
Mini-batches

Results are exactly the same!

Summingbird Word Count

```
def wordCount[P <: Platform[P]]  
  (source: Producer[P, String],  
   store: P#Store[String, Long]) =  
    source.flatMap { sentence =>  
      toWords(sentence).map(_ -> 1L)  
    }.sumByKey(store)
```

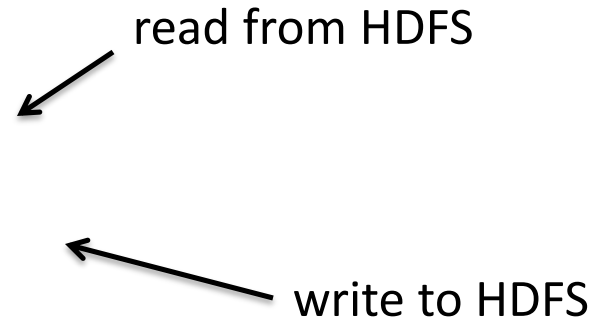
where data comes from
where data goes
"map"
"reduce"



Run on Scalding (Cascading/Hadoop)

```
Scalding.run {  
  wordCount[Scalding](  
    Scalding.source[Tweet]("source_data"),  
    Scalding.store[String, Long]("count_out")  
  )  
}
```

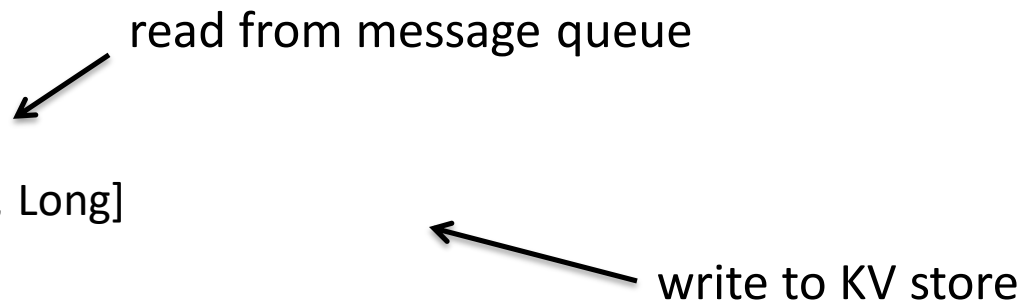
read from HDFS
write to HDFS

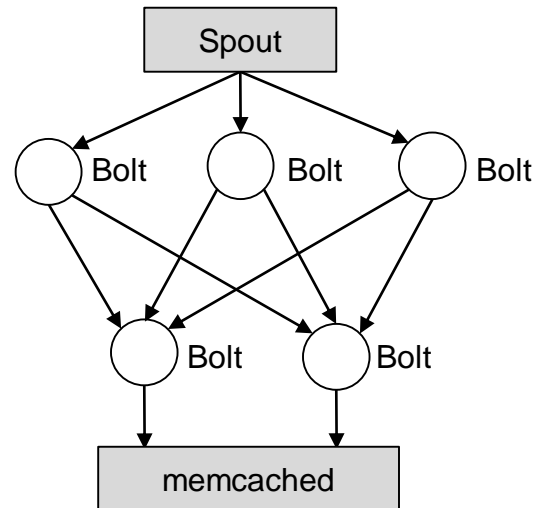
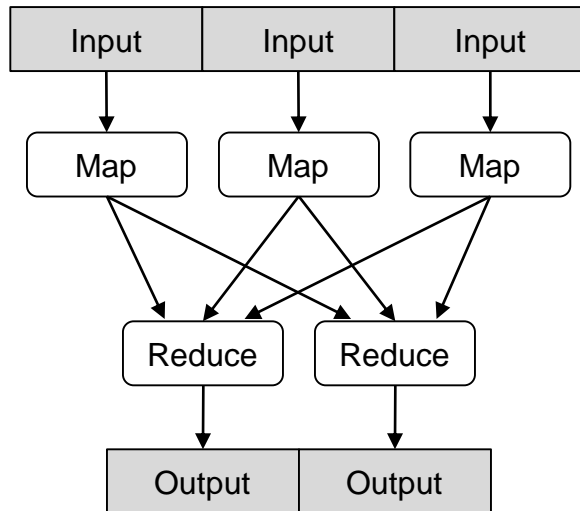
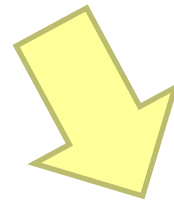
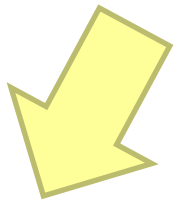
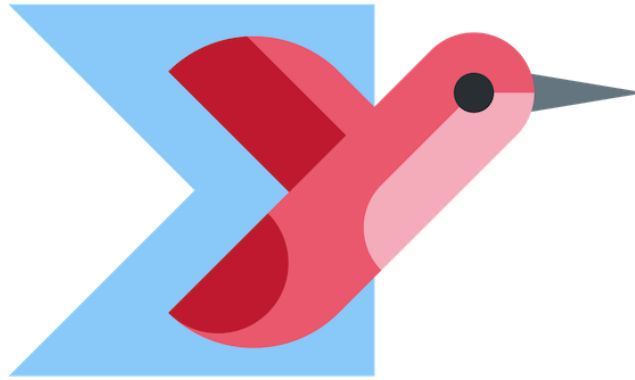


Run on Storm

```
Storm.run {  
  wordCount[Storm](  
    new TweetSpout(),  
    new MemcacheStore[String, Long]  
  )  
}
```

read from message queue
write to KV store





“Boring” monoids

addition, multiplication, max, min

moments (mean, variance, etc.)

sets

tuples of monoids

hashmaps with monoid values

More interesting monoids?

“Interesting” monoids

Bloom filters (set membership)

HyperLogLog counters (cardinality estimation)

Count-min sketches (event counts)

Idea #2: For many tasks, close enough is good enough!

Cheat Sheet

	Exact	Approximate
Set membership	set	Bloom filter
Set cardinality	set	hyperloglog counter
Frequency count	hashmap	count-min sketches

Example: Count queries by hour

Exact with hashmaps

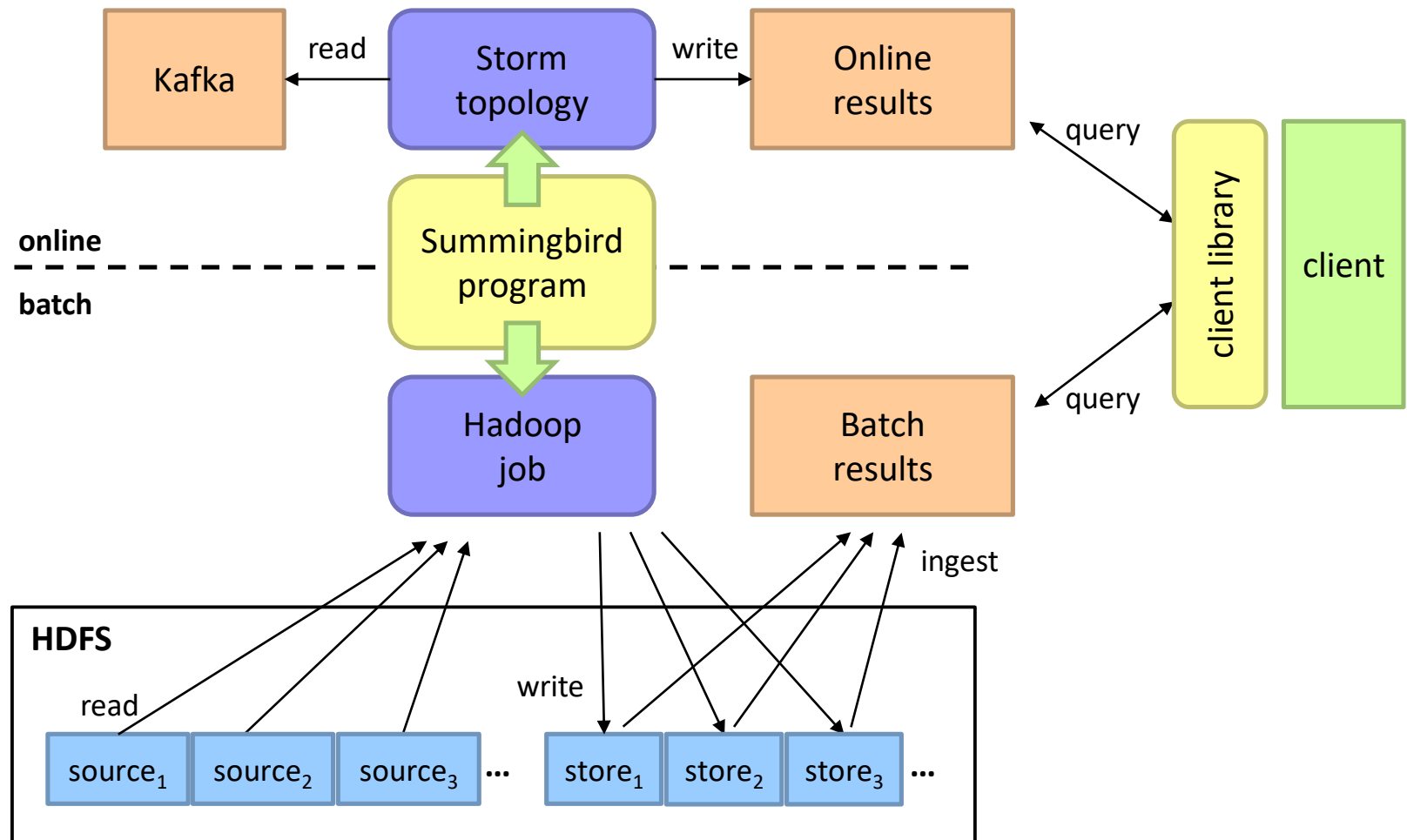
```
def wordCount[P <: Platform[P]]  
  (source: Producer[P, Query],  
   store: P#Store[Long, Map[String, Long]]) =  
  source.flatMap { query =>  
    (query.getHour, Map(query.getQuery -> 1L))  
  }.sumByKey(store)
```

Approximate with CMS

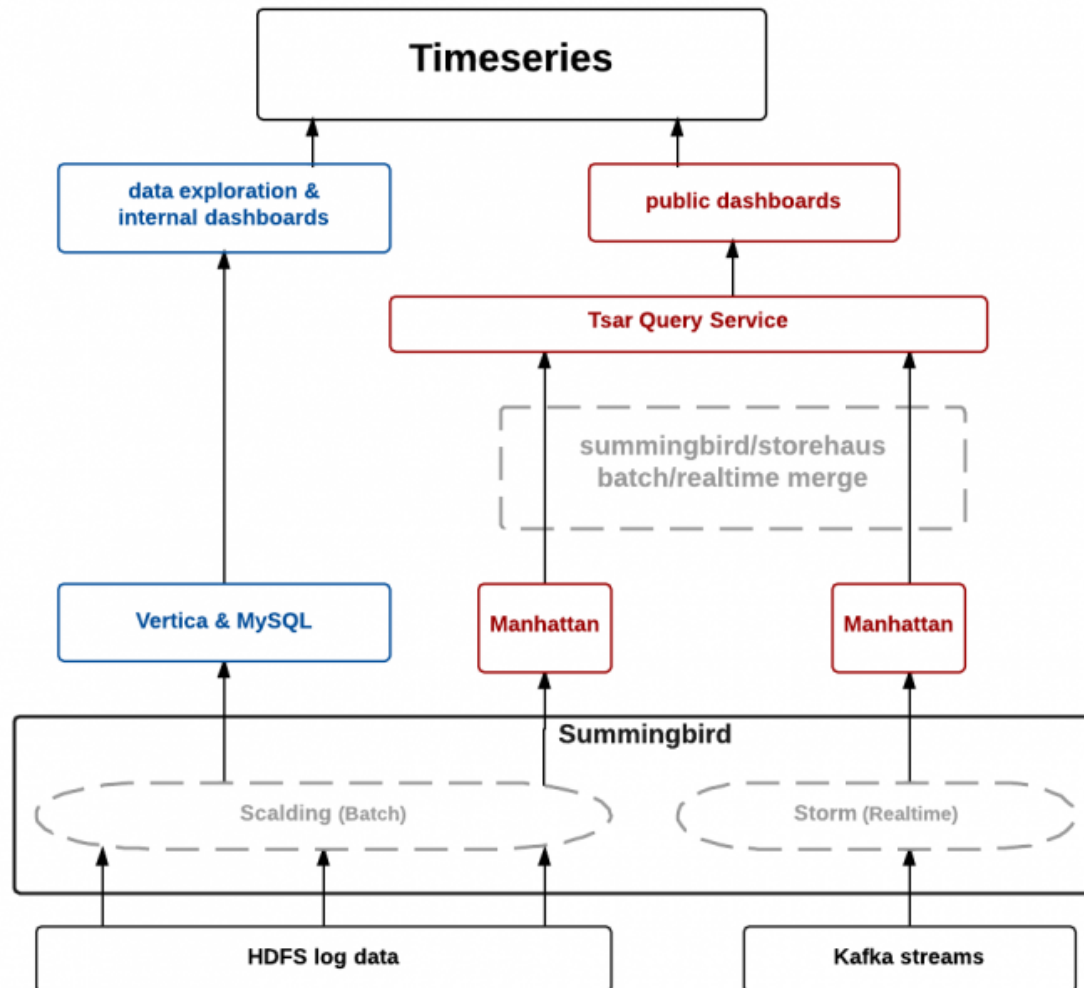
```
def wordCount[P <: Platform[P]]  
  (source: Producer[P, Query],  
   store: P#Store[Long, SketchMap[String, Long]])  
  (implicit countMonoid: SketchMapMonoid[String, Long]) =  
  source.flatMap { query =>  
    (query.getHour,  
     countMonoid.create((query.getQuery, 1L)))  
  }.sumByKey(store)
```

Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time

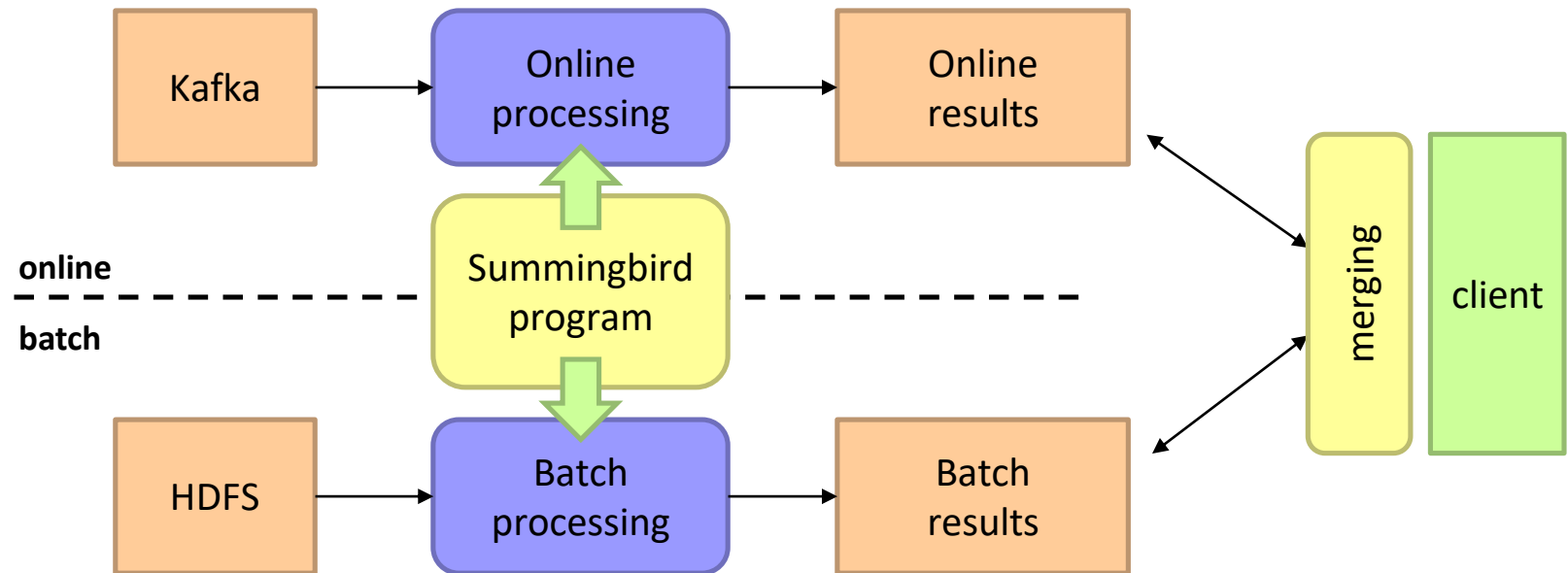


TSAR, a TimeSeries AggregatoR!



Hybrid Online/Batch Processing

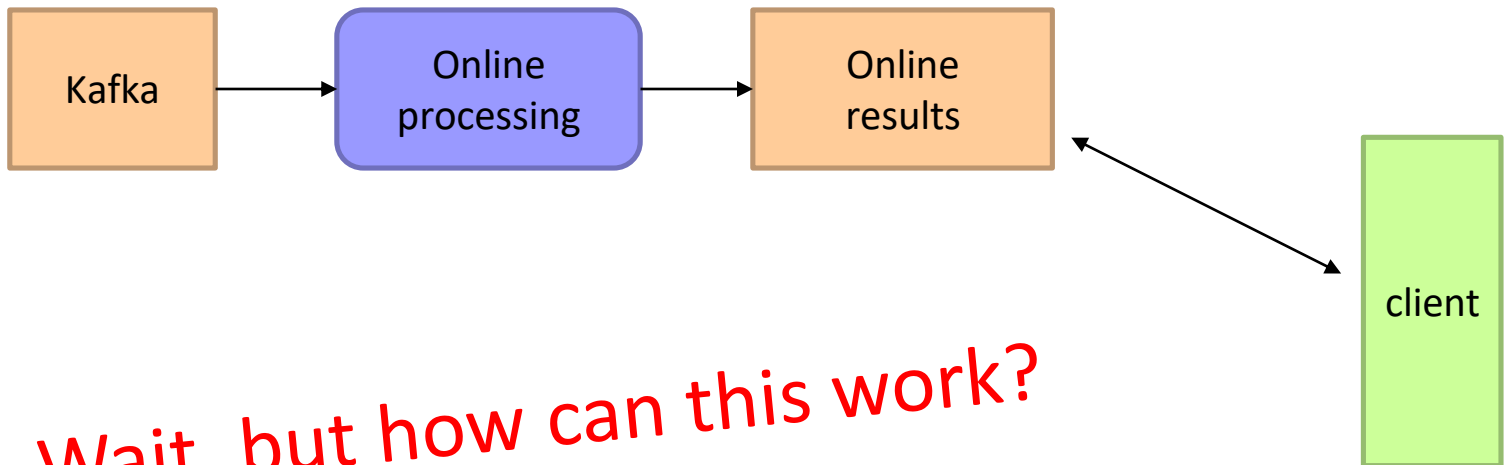
Example: count historical clicks and clicks in real time



But this is still too painful...

Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time

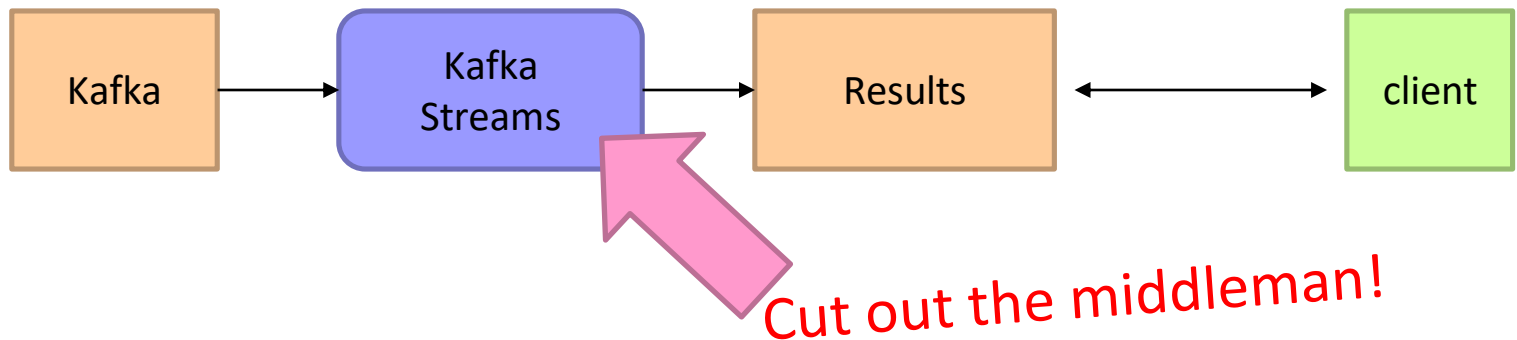


Wait, but how can this work?

Idea: everything is streaming

Batch processing is just streaming through a historic dataset!

Everything is Streaming!



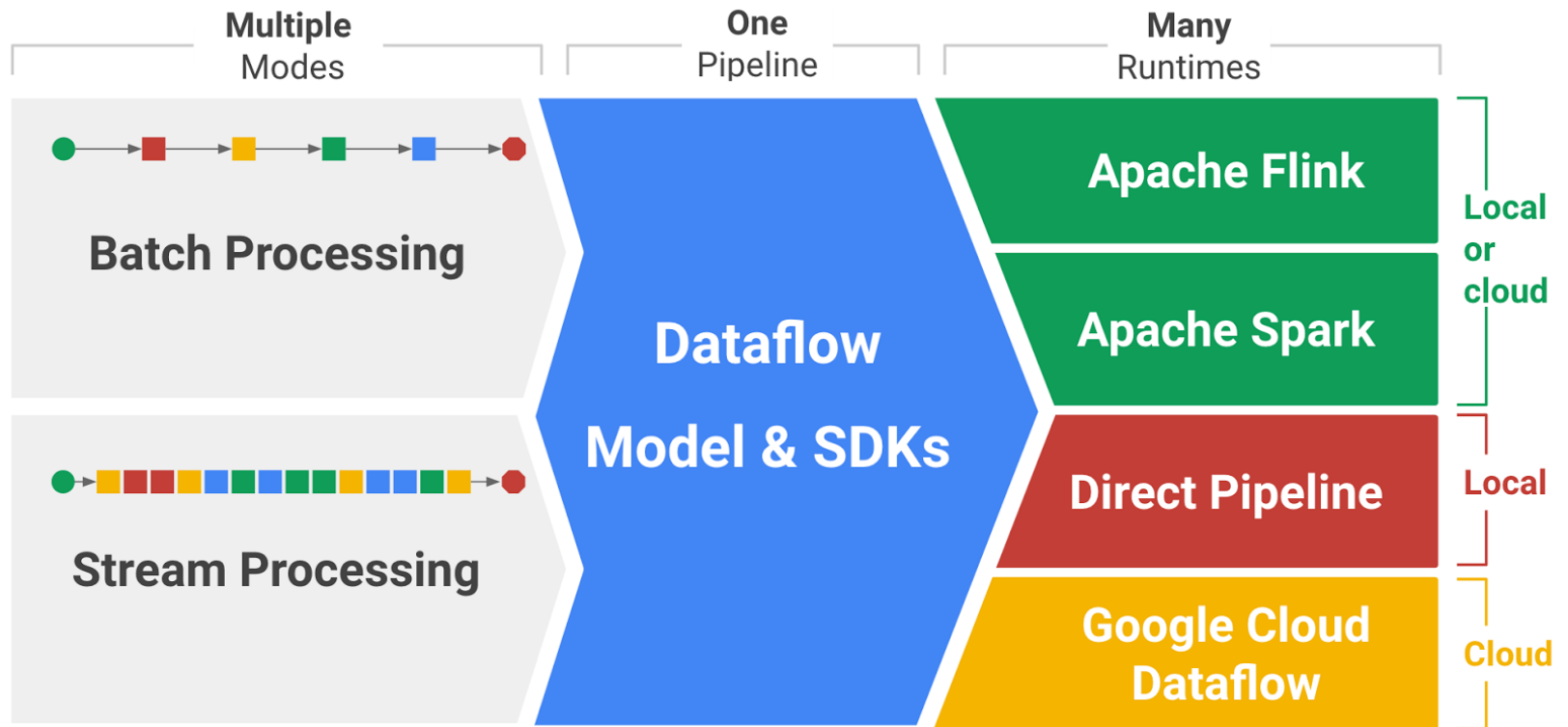
```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> textLines = builder.stream("TextLinesTopic");
KTable<String, Long> wordCounts = textLines
    .flatMapValues(textLine ->
        Arrays.asList(textLine.toLowerCase().split("\\W+")))
    .groupBy((key, word) -> word)
    .count(Materialized.<String, Long,
        KeyValueStore<Bytes, byte[]>>as("counts-store"));
wordCounts.toStream().to("WordsWithCountsTopic",
    Produced.with(Serdes.String(), Serdes.Long()));

KafkaStreams streams = new KafkaStreams(builder.build(), config);
streams.start();
```

K

(I hate this too.)

The Vision



Processing Bounded Datasets

```
Pipeline p = Pipeline.create(options);
```

```
p.apply(TextIO.Read.from("gs://your/input/"))
```

```
.apply(FlatMapElements.via((String word) ->  
  Arrays.asList(word.split("[^a-zA-Z']+"))))  
.apply(Filter.by((String word) -> !word.isEmpty()))  
.apply(Count.perElement())  
.apply(MapElements.via((KV<String, Long> wordCount) ->  
  wordCount.getKey() + ": " + wordCount.getValue()))  
.apply(TextIO.Write.to("gs://your/output/"));
```

Processing Unbounded Datasets

```
Pipeline p = Pipeline.create(options);
```

```
p.apply(KafkaIO.read("tweets")
  .withTimestampFn(new TweetTimestampFunction())
  .withWatermarkFn(kv ->
    Instant.now().minus(Duration.standardMinutes(2))))
.apply(Window.into(FixedWindows.of(Duration.standardMinutes(2)))
  .triggering(AtWatermark())
  .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
  .withLateFirings(AtCount(1)))
  .accumulatingAndRetractingFiredPanels())
.apply(FlatMapElements.via((String word) ->
  Arrays.asList(word.split("[^a-zA-Z]+"))))
.apply(Filter.by((String word) -> !word.isEmpty()))
.apply(Count.perElement())
.apply(KafkaIO.write("counts"))
```

Where in event time?

When in processing time?

How do refines relate?

