UNIVERSITY OF WATERLOO

Data-Intensive Distributed Computing

CS 451/651 (Fall 2018)

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

November 27, 2018

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2018f/

Since last time…

Storm/Heron

Gives you pipes, but you gotta connect everything up yourself

Spark Streaming

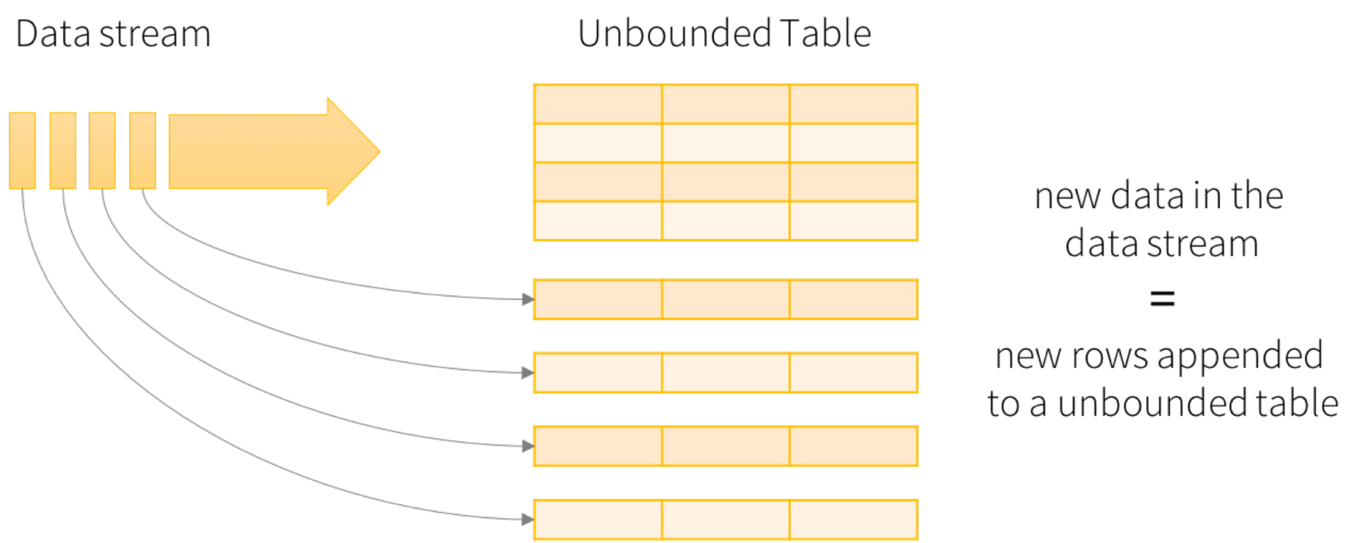
Gives you RDDs, transformations and windowing –
but no event/processing time distinction

Beam

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

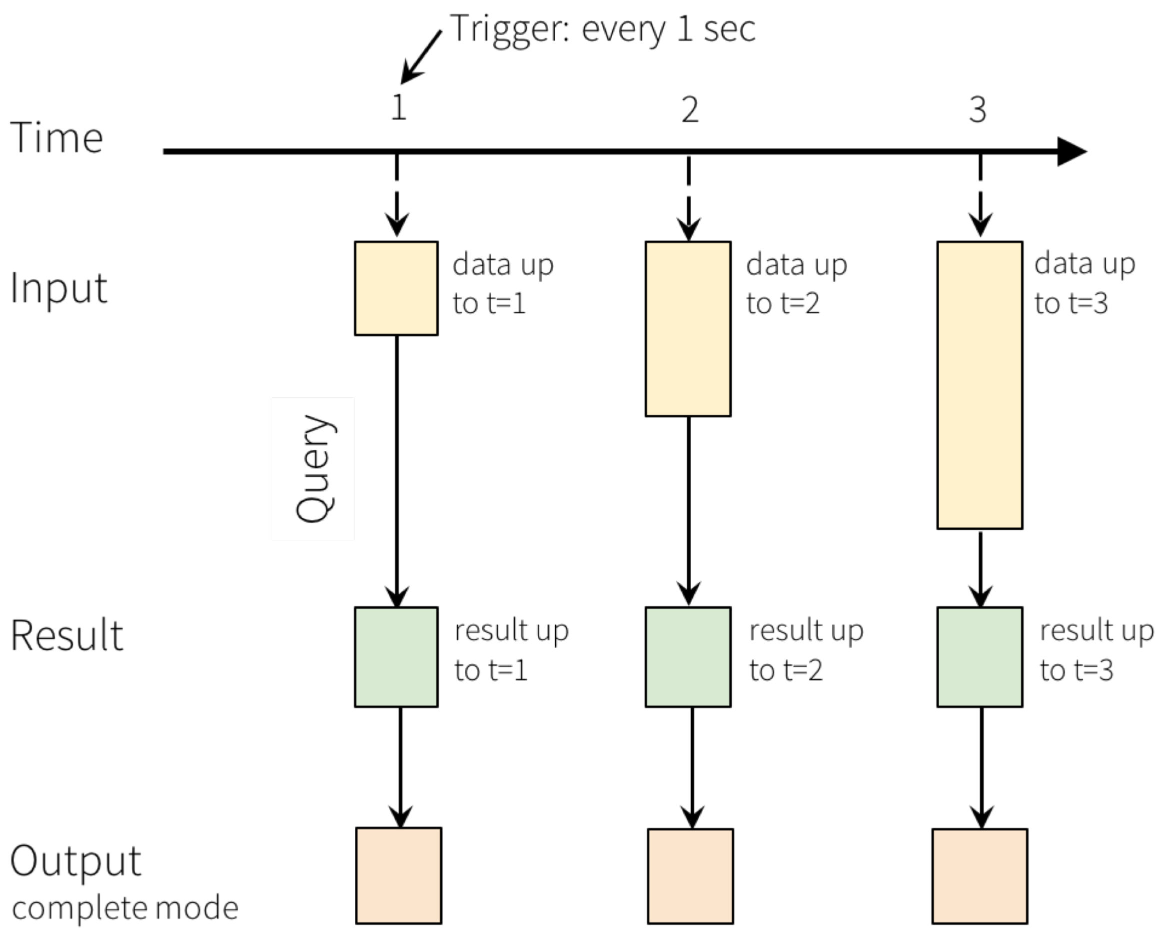

Spark *Structured* Streaming
Stream Processing Frameworks
Source: Wikipedia (River)

Step 1: From RDDs to DataFrames
Step 2: From bounded to unbounded tables



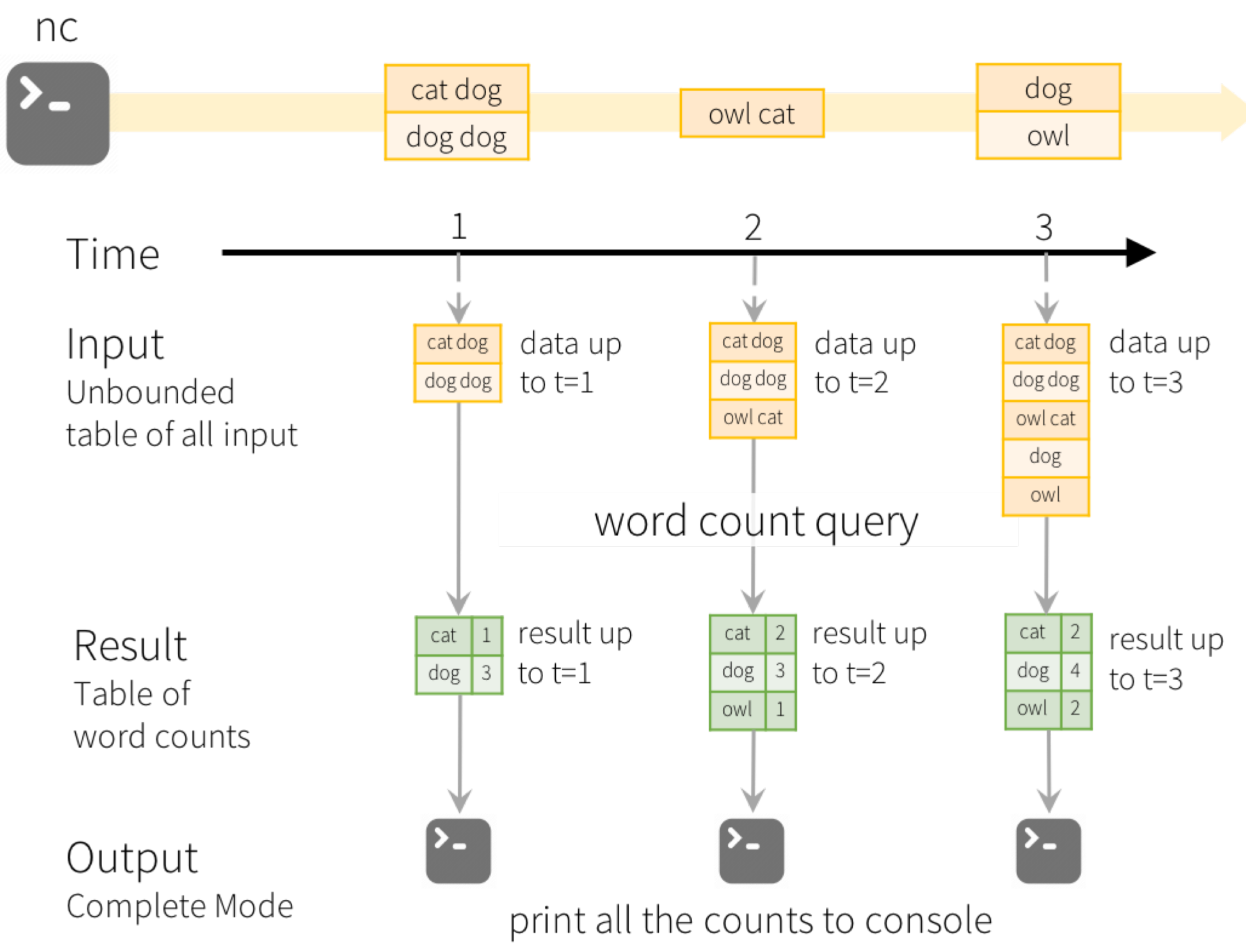
Data stream as an unbounded table

Source: Spark Structured Streaming Documentation



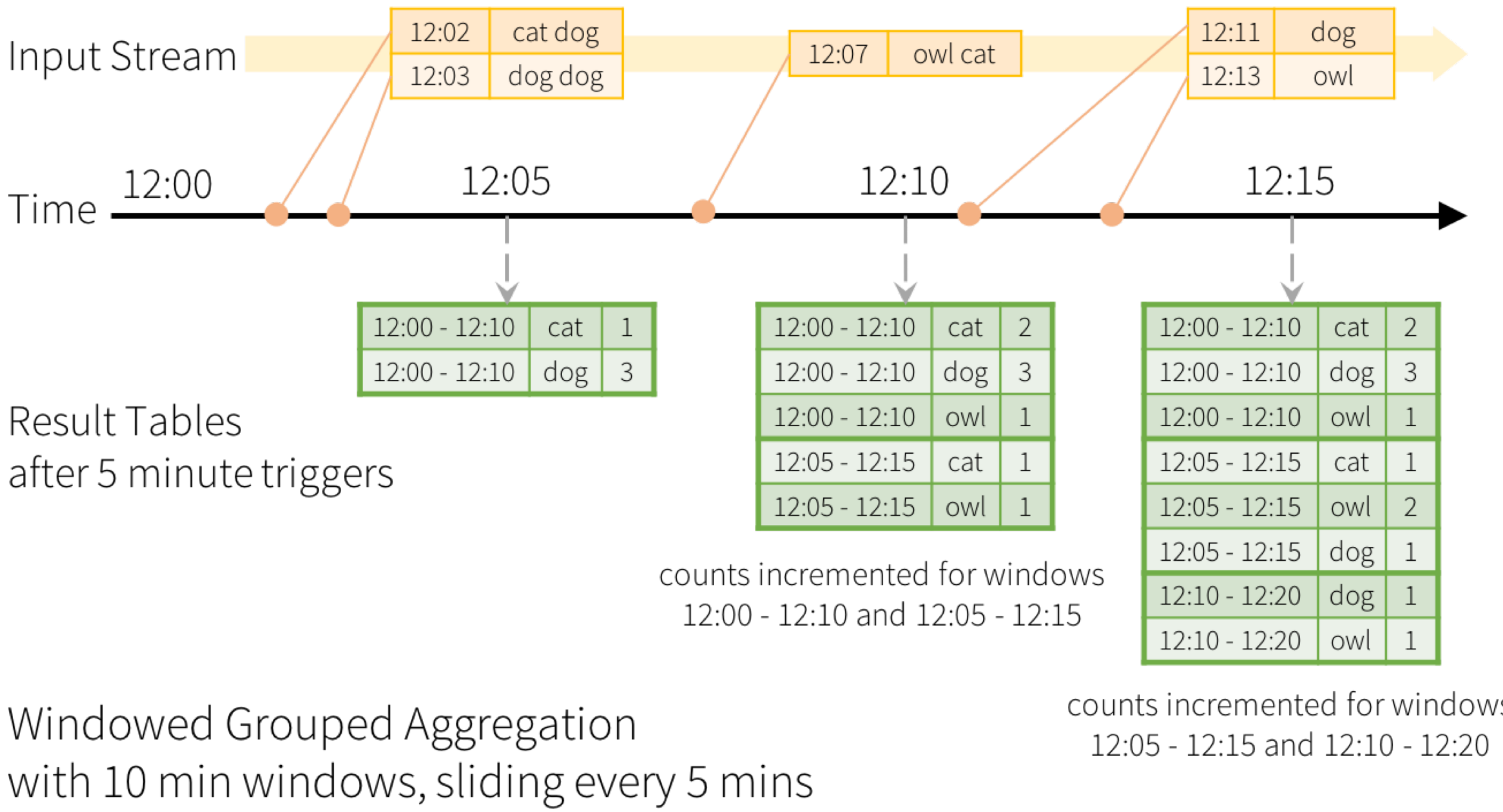
Programming Model for Structured Streaming

Source: Spark Structured Streaming Documentation

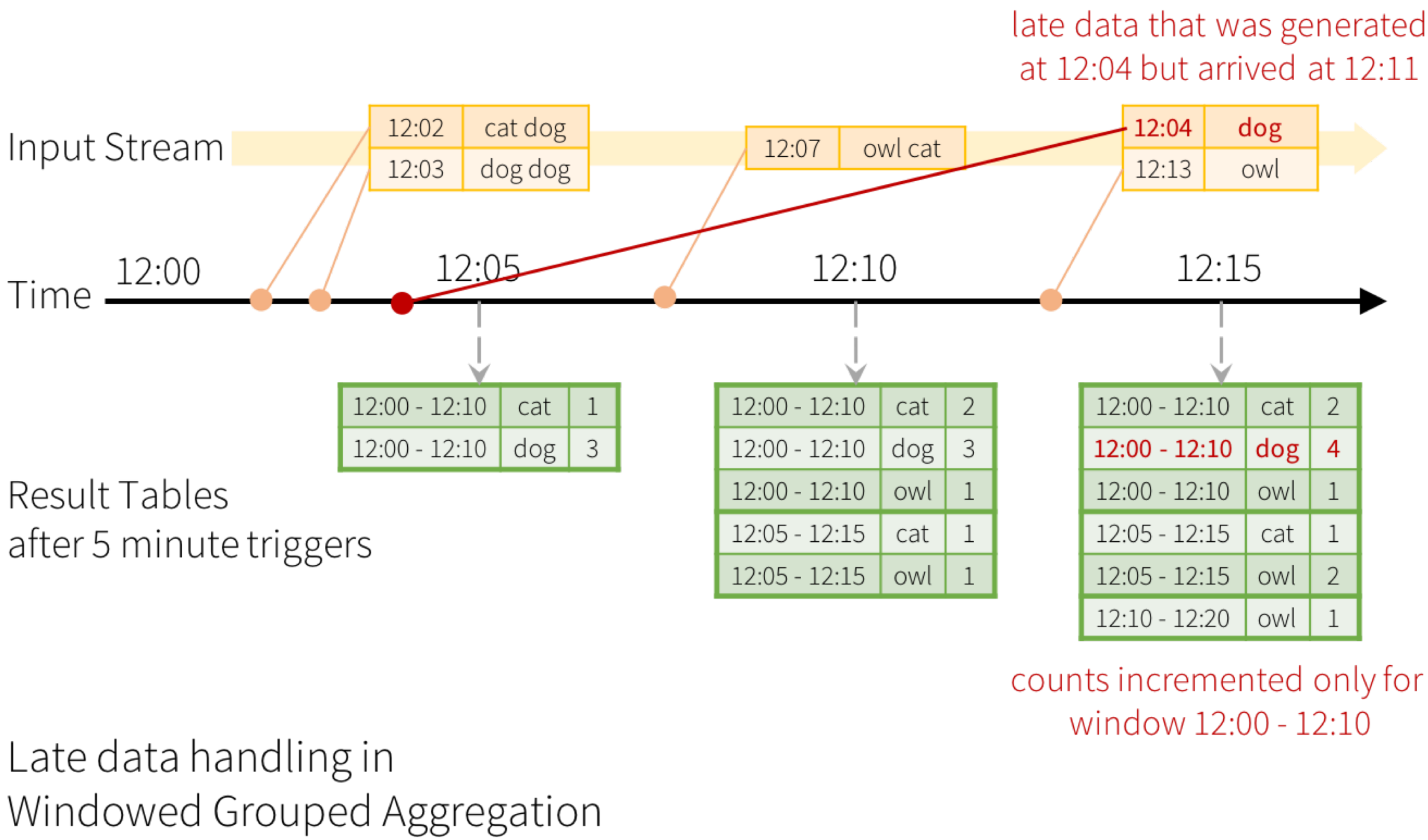


Model of the Quick Example

Source: Spark Structured Streaming Documentation



Source: Spark Structured Streaming Documentation



Source: Spark Structured Streaming Documentation


Interlude
Source: Wikipedia (River)

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

put(x) $\longrightarrow$ insert x into the set

contains(x) $\longrightarrow$ yes if x is a member of the set

Components

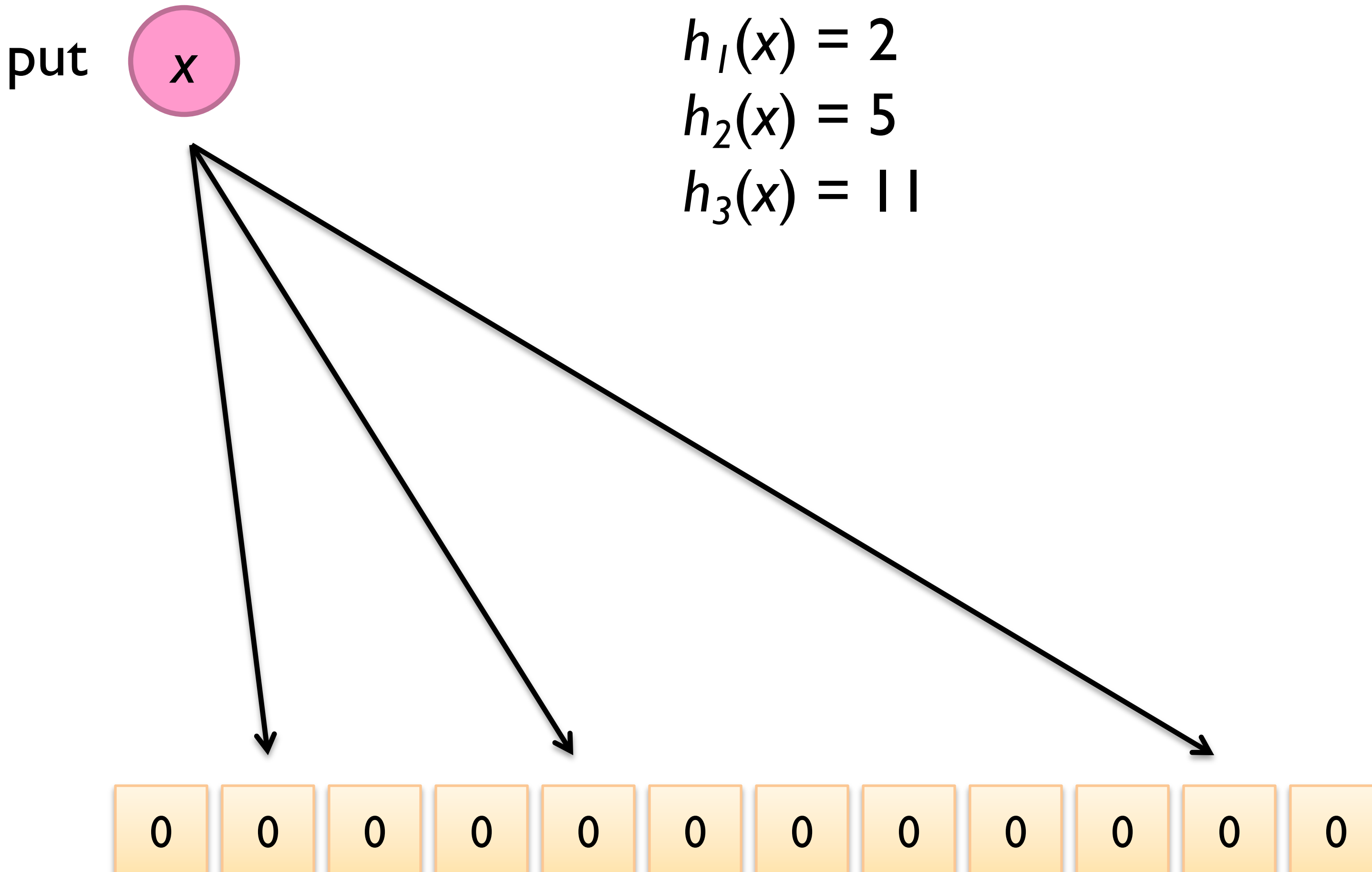
m-bit bit vector

k hash functions: h_1 … h_k

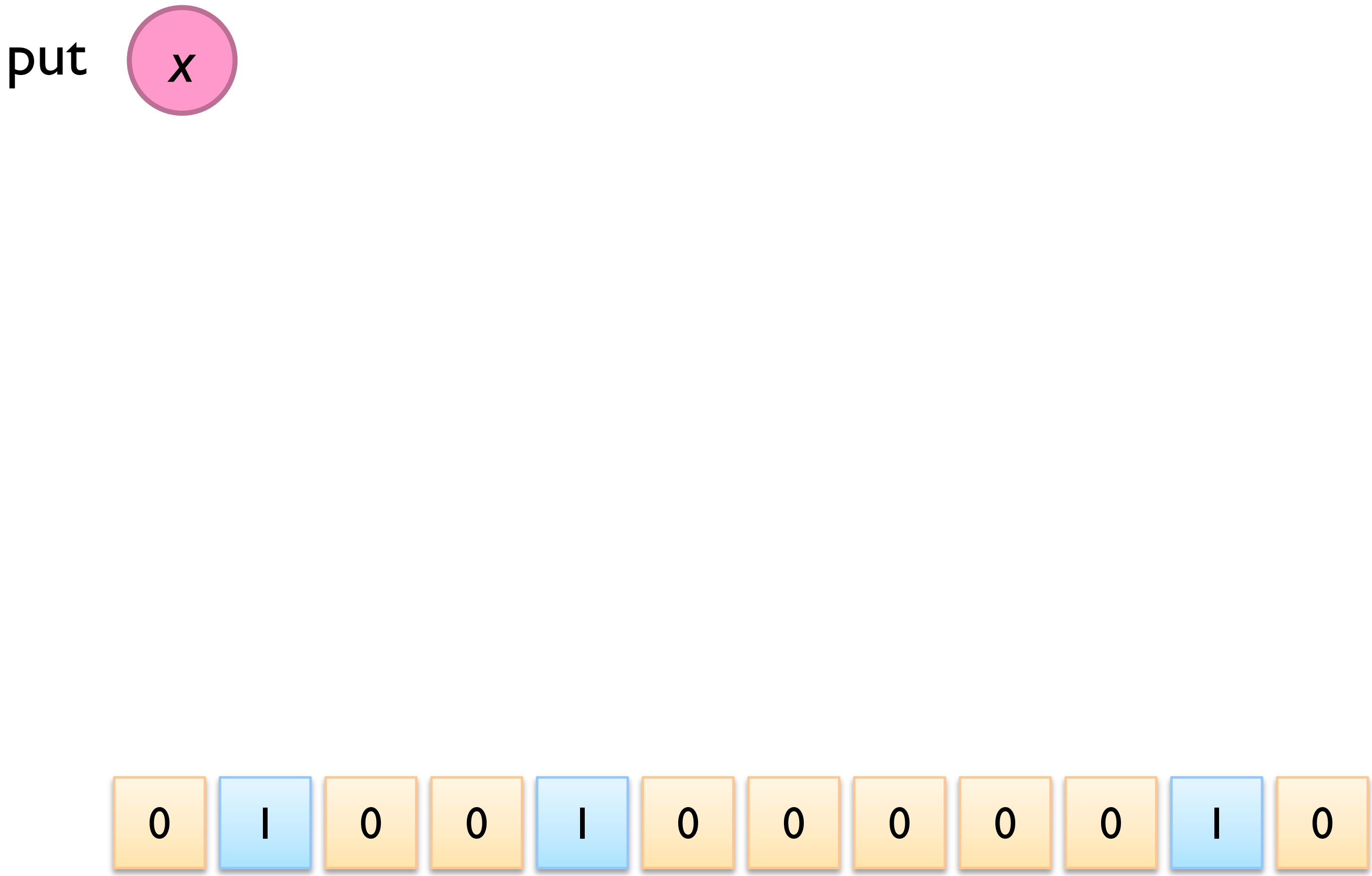
0	0	0	0	0	0	0	0	0	0	0	0

Bloom Filters: put

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$

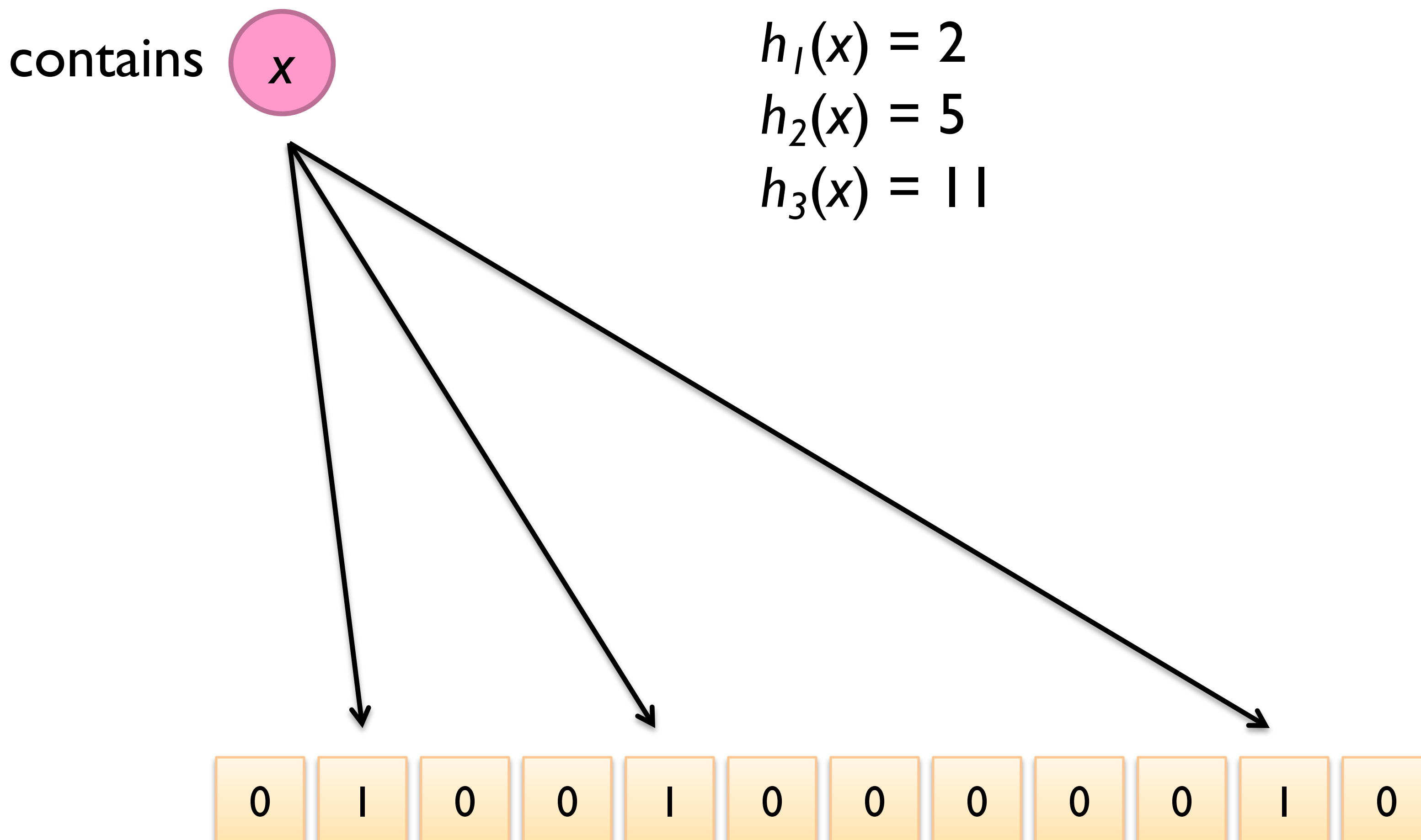


Bloom Filters: put



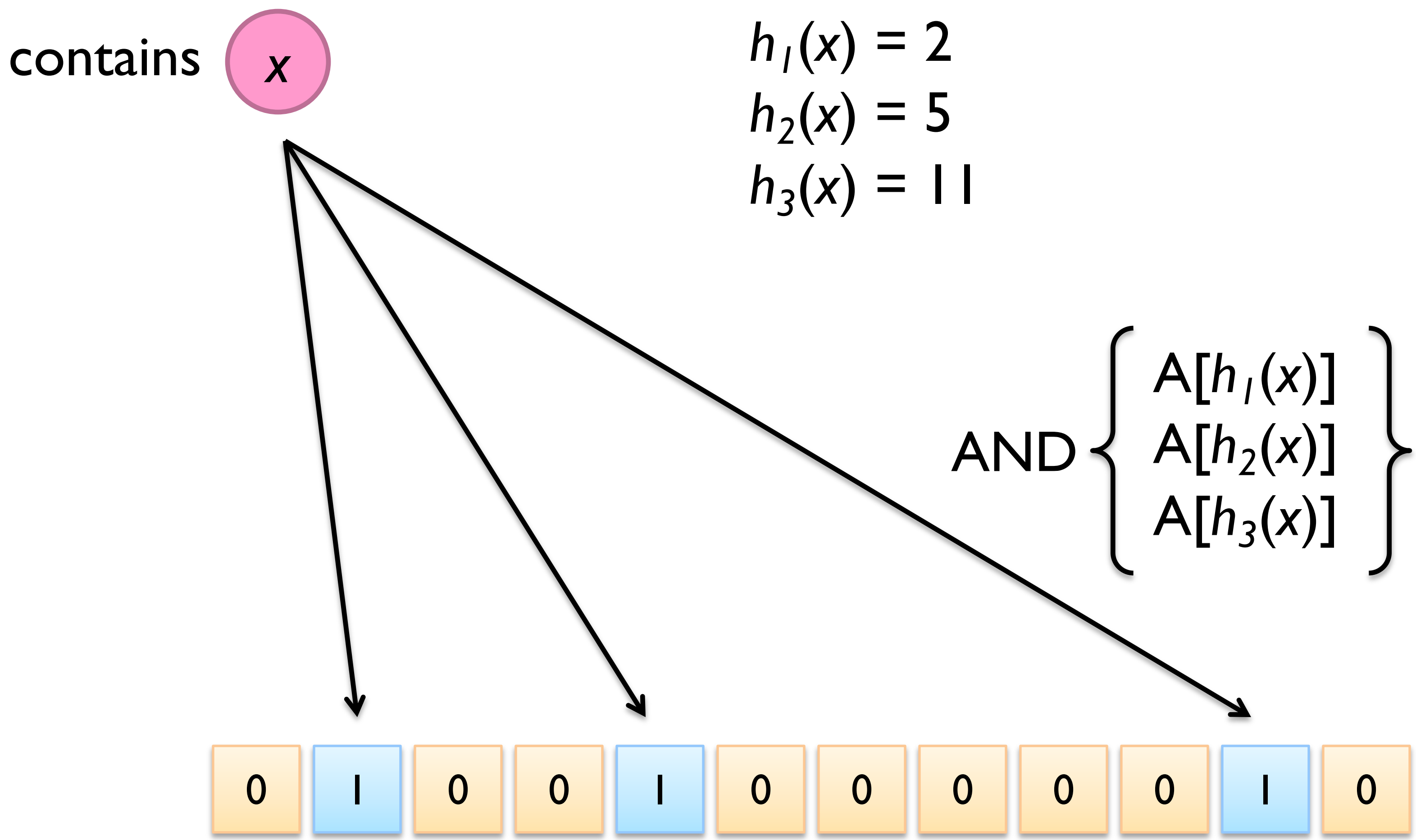
Bloom Filters: contains

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$



Bloom Filters: contains

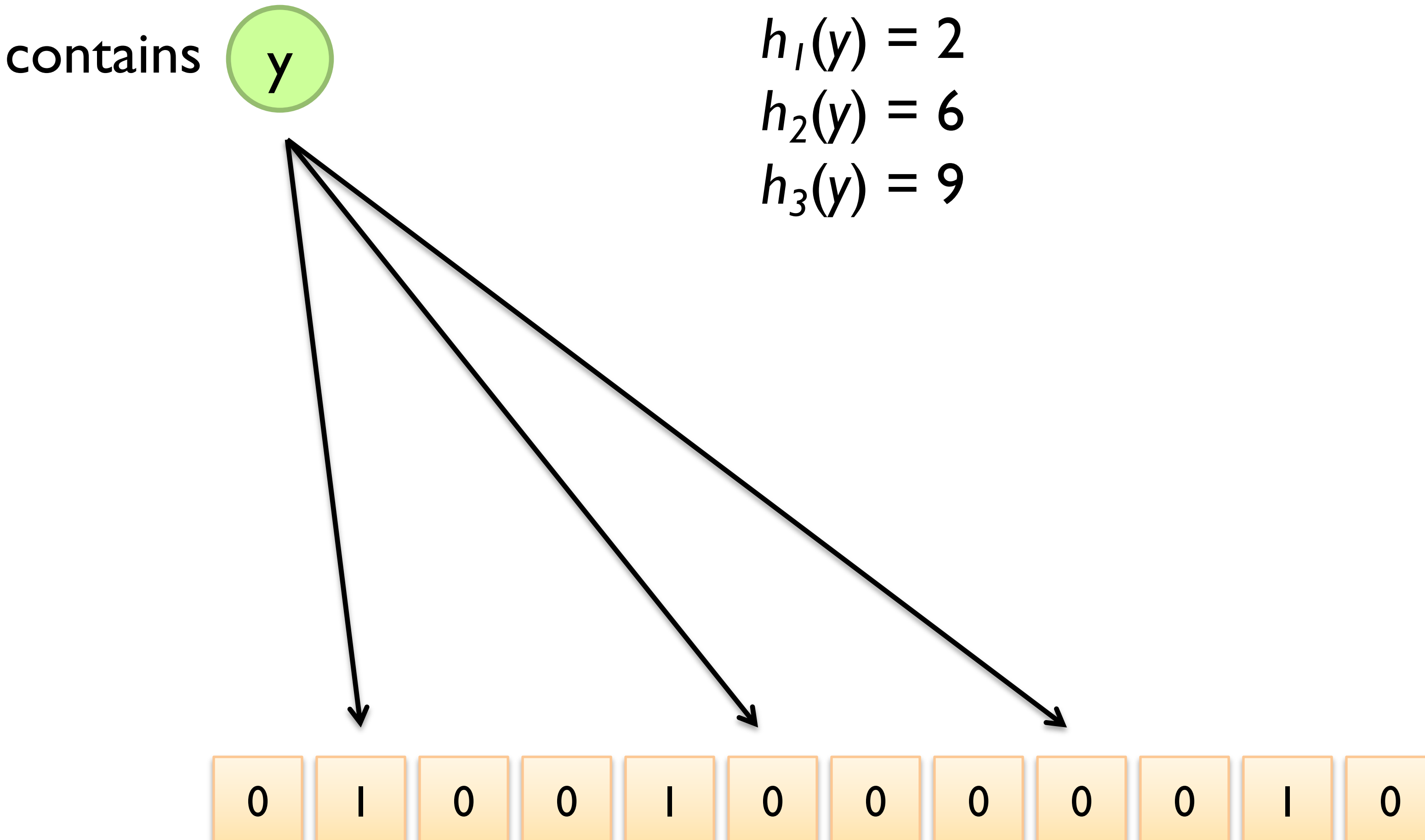
$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$



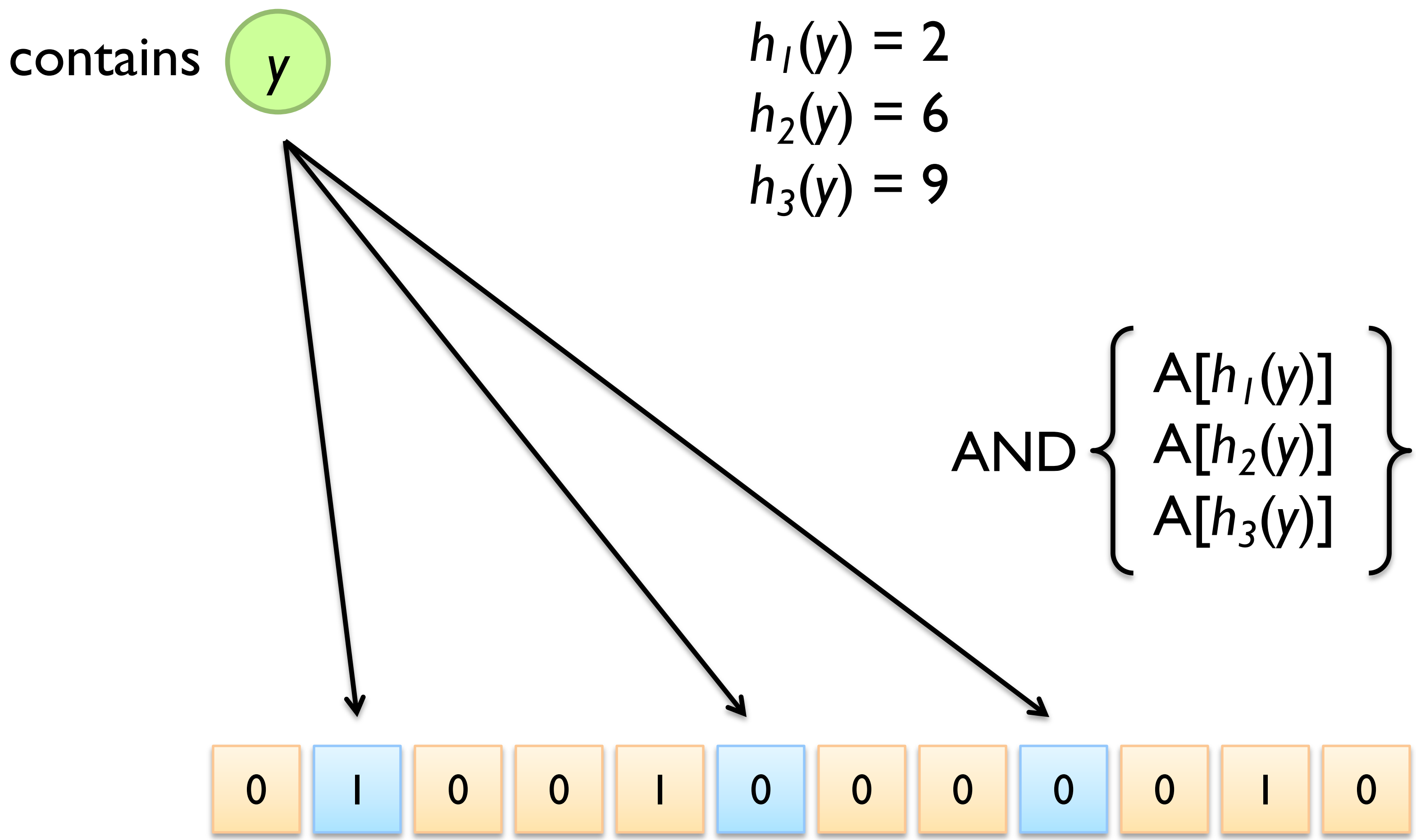
$$\text{AND}\left\{\begin{array}{l} A[h_1(x)] \\ A[h_2(x)] \\ A[h_3(x)] \end{array}\right\} = \text{YES}$$

Bloom Filters: contains

$h_1(y) = 2$
$h_2(y) = 6$
$h_3(y) = 9$



Bloom Filters: contains



$h_1(y) = 2$
$h_2(y) = 6$
$h_3(y) = 9$

$$\text{AND}\left\{\begin{array}{l} A[h_1(y)] \\ A[h_2(y)] \\ A[h_3(y)] \end{array}\right\} = \text{NO}$$

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

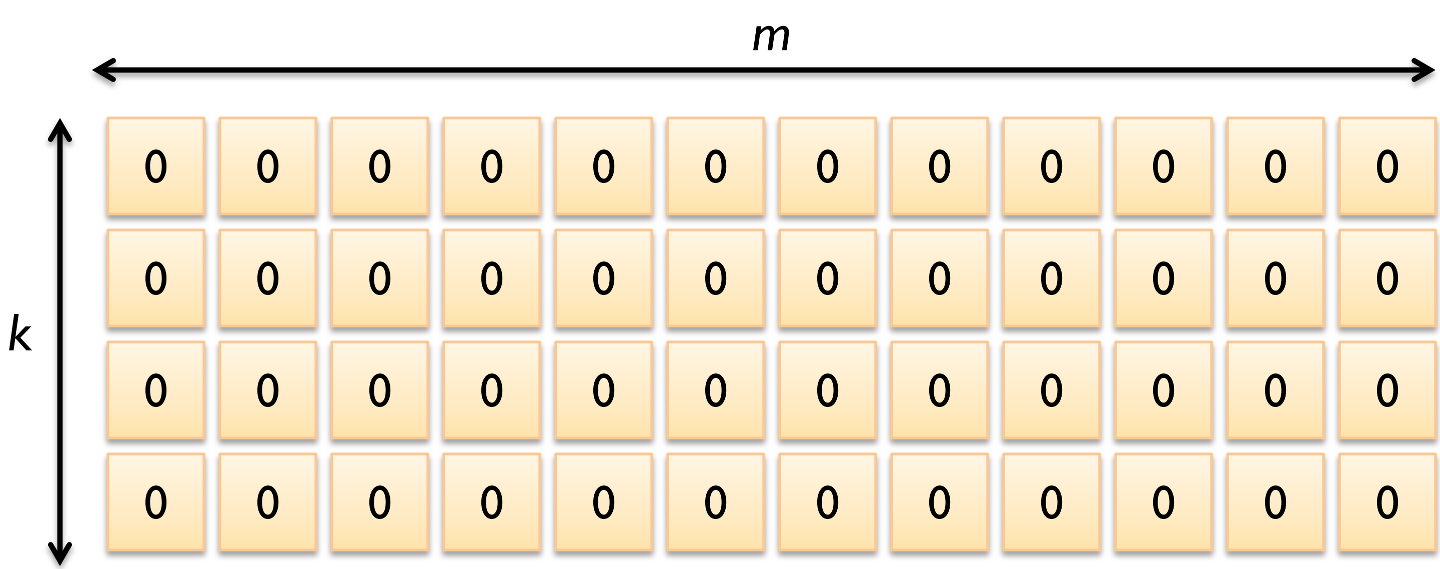
put(x) → increment count of x by one

get(x) → returns the frequency of x

Components

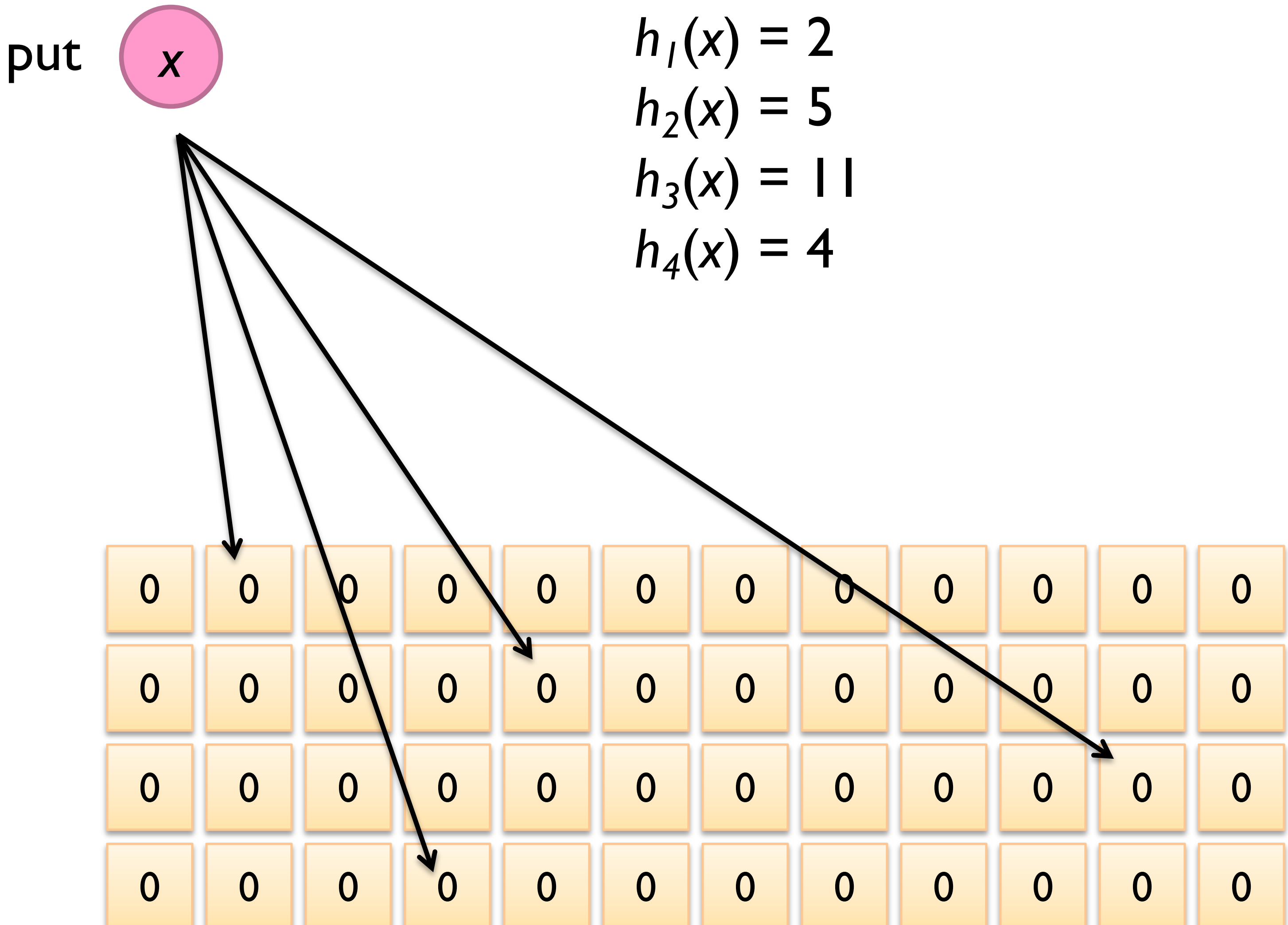
m by k array of counters

k hash functions: $h_1 \ldots h_k$



Count-Min Sketches: put

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$
$h_4(x) = 4$



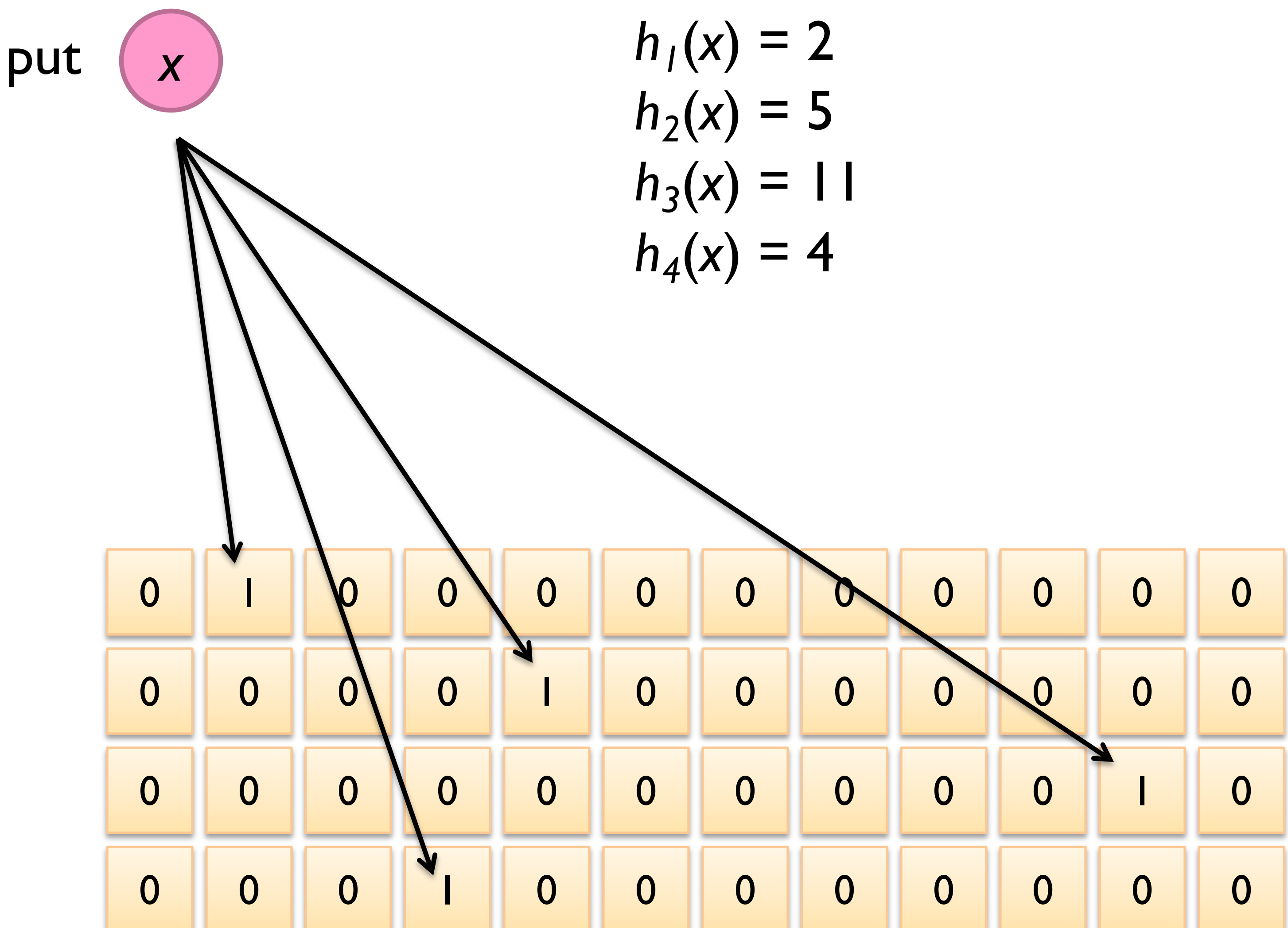
Count-Min Sketches: put

put x

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

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$
$h_4(x) = 4$



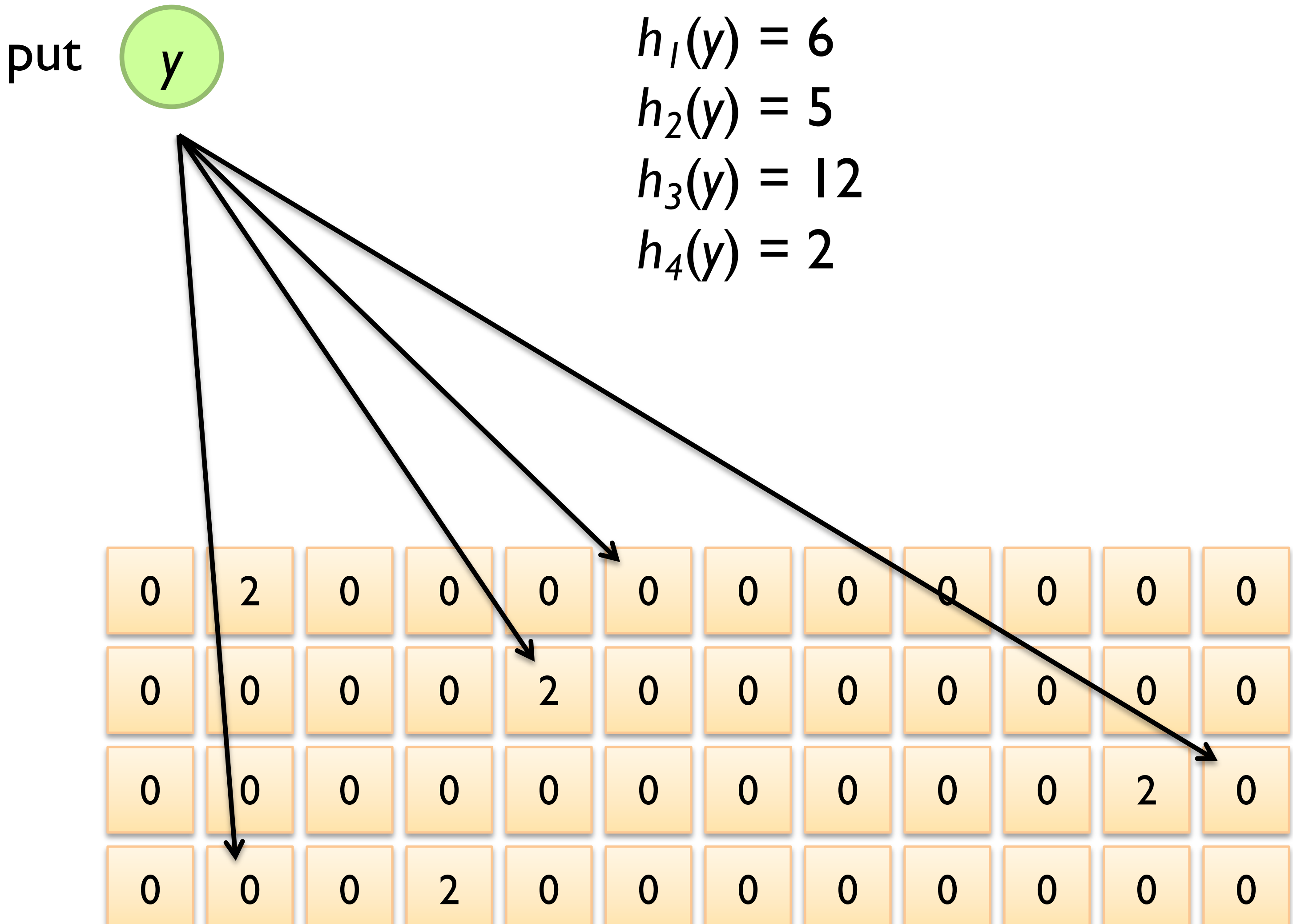
Count-Min Sketches: put

put x

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

$h_1(y) = 6$
$h_2(y) = 5$
$h_3(y) = 12$
$h_4(y) = 2$



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 y

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

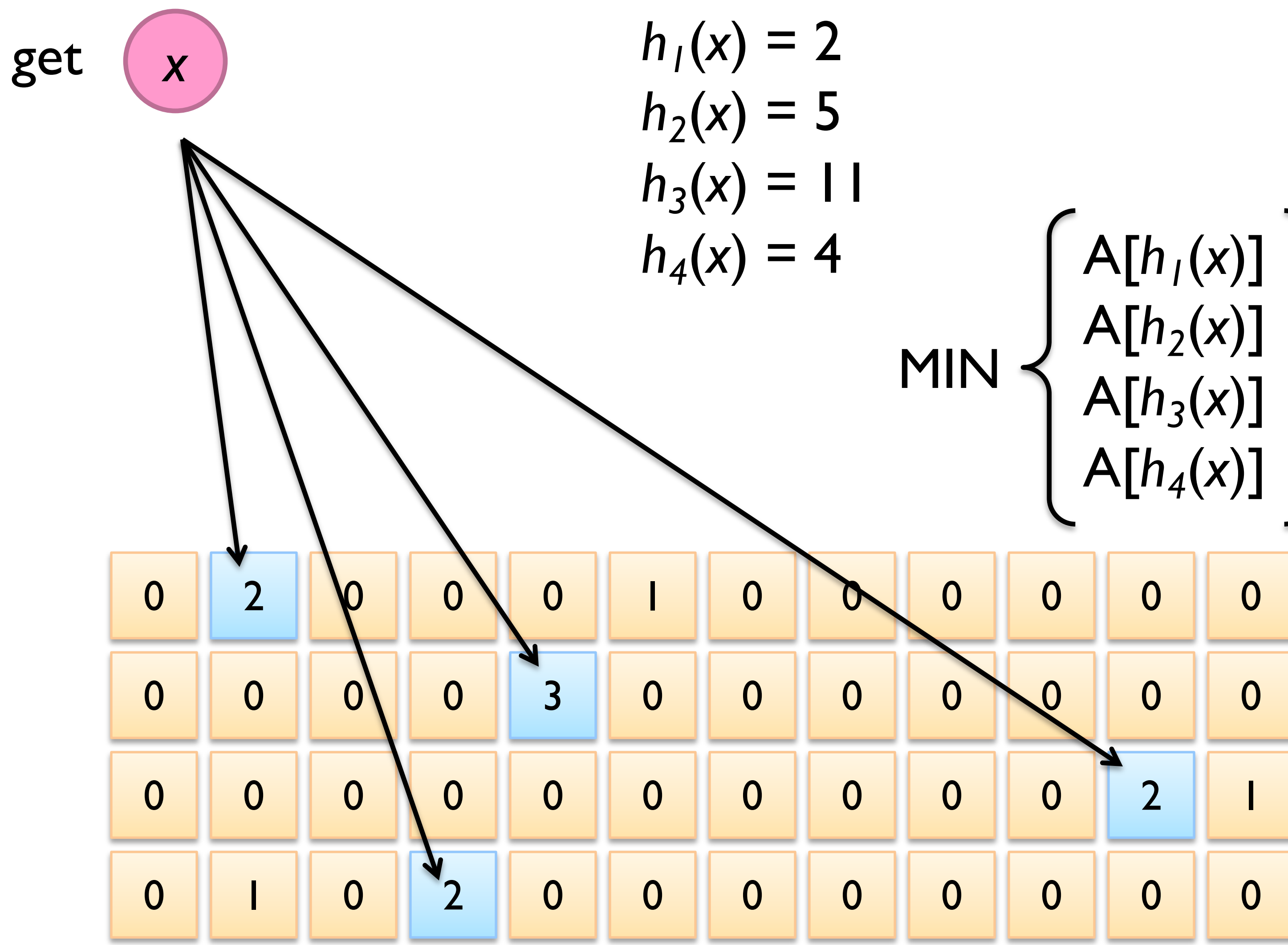
$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$
$h_4(x) = 4$

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

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$
$h_4(x) = 4$

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



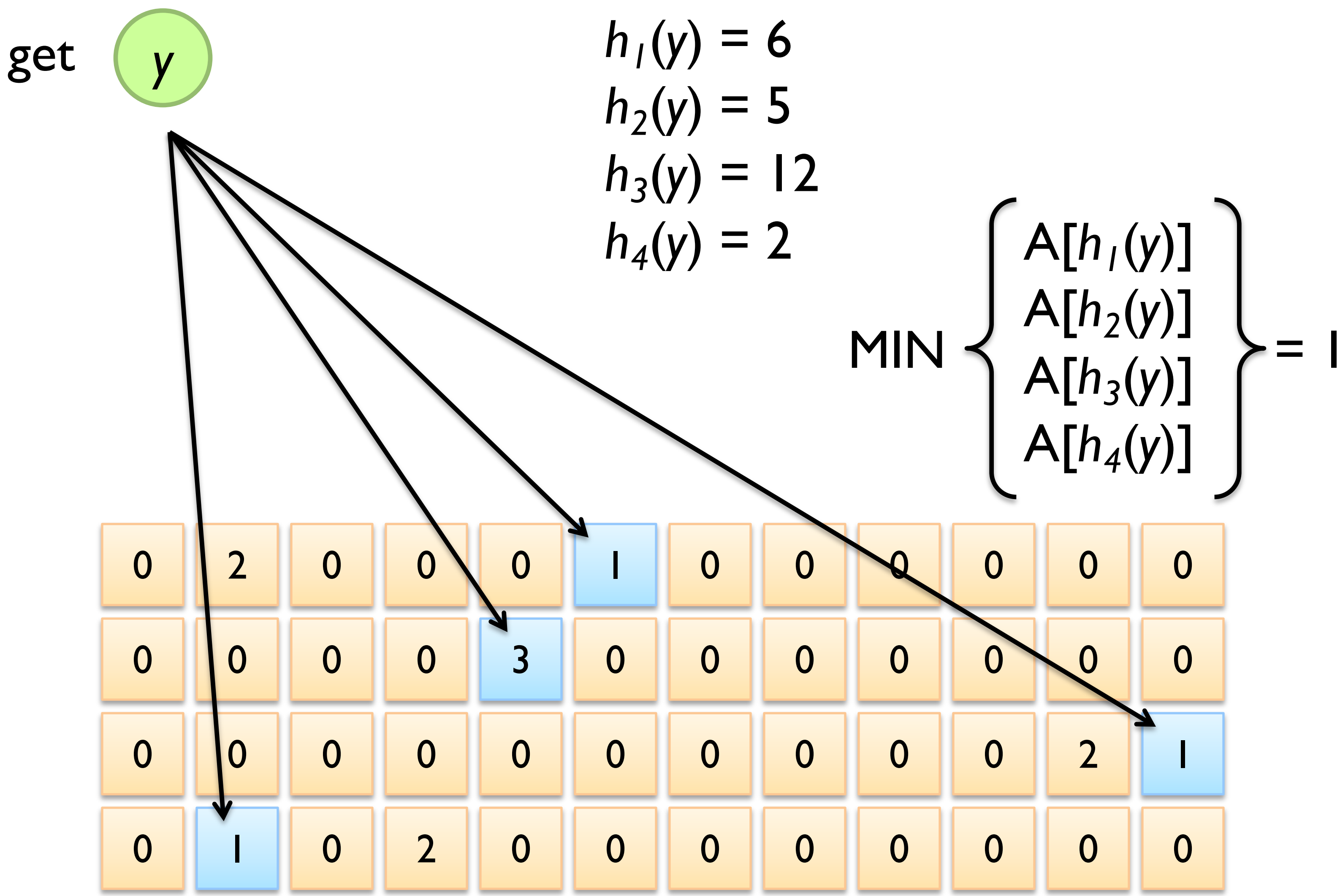
Count-Min Sketches: get

get y

$h_1(y) = 6$
$h_2(y) = 5$
$h_3(y) = 12$
$h_4(y) = 2$

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



Count-Min Sketches

Error properties: 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

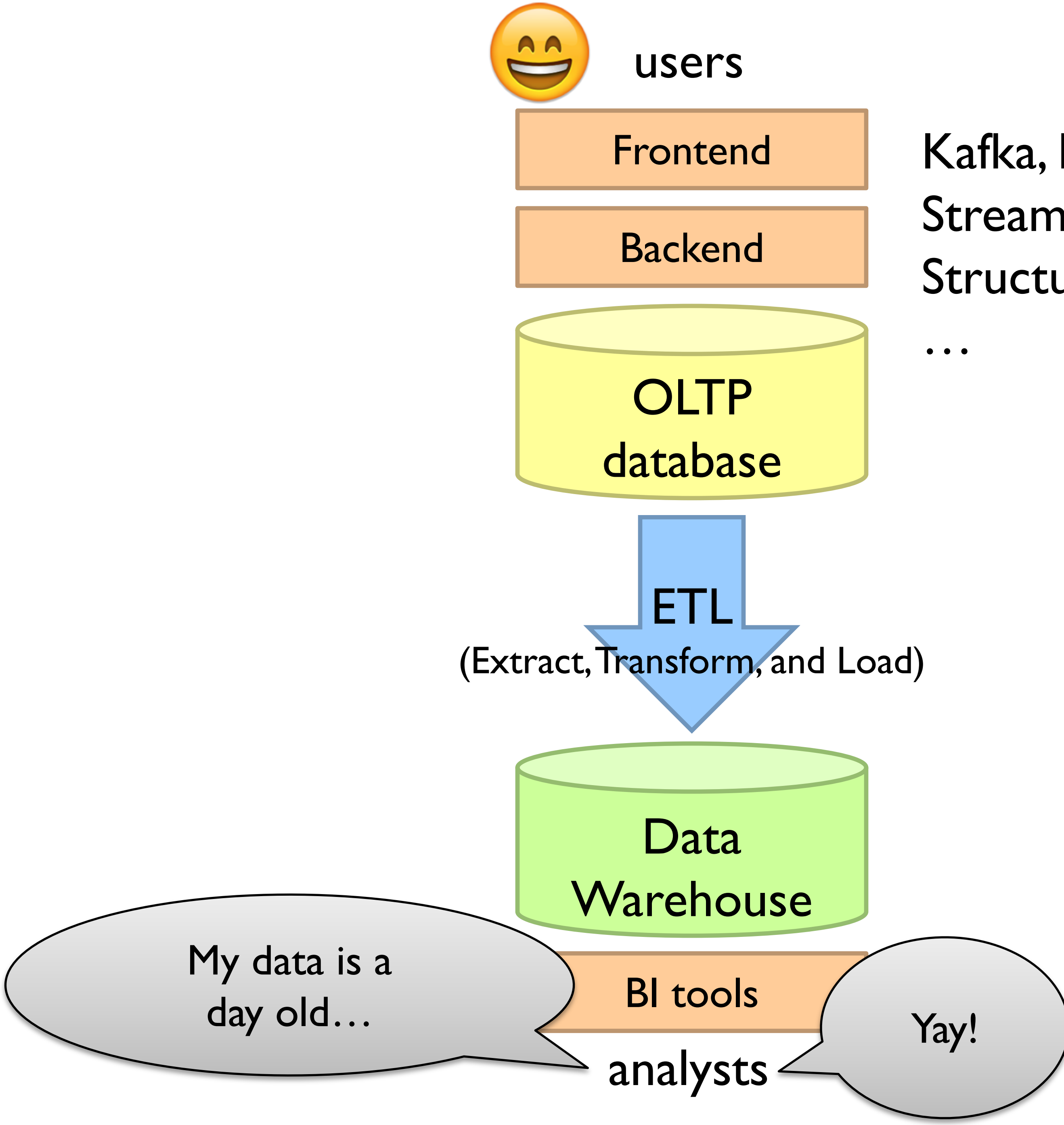
Frequency estimation

How many times have we observed *x*?

How many queries has this user issued?

HashMap CMS


Stream Processing Frameworks
Source: Wikipedia (River)



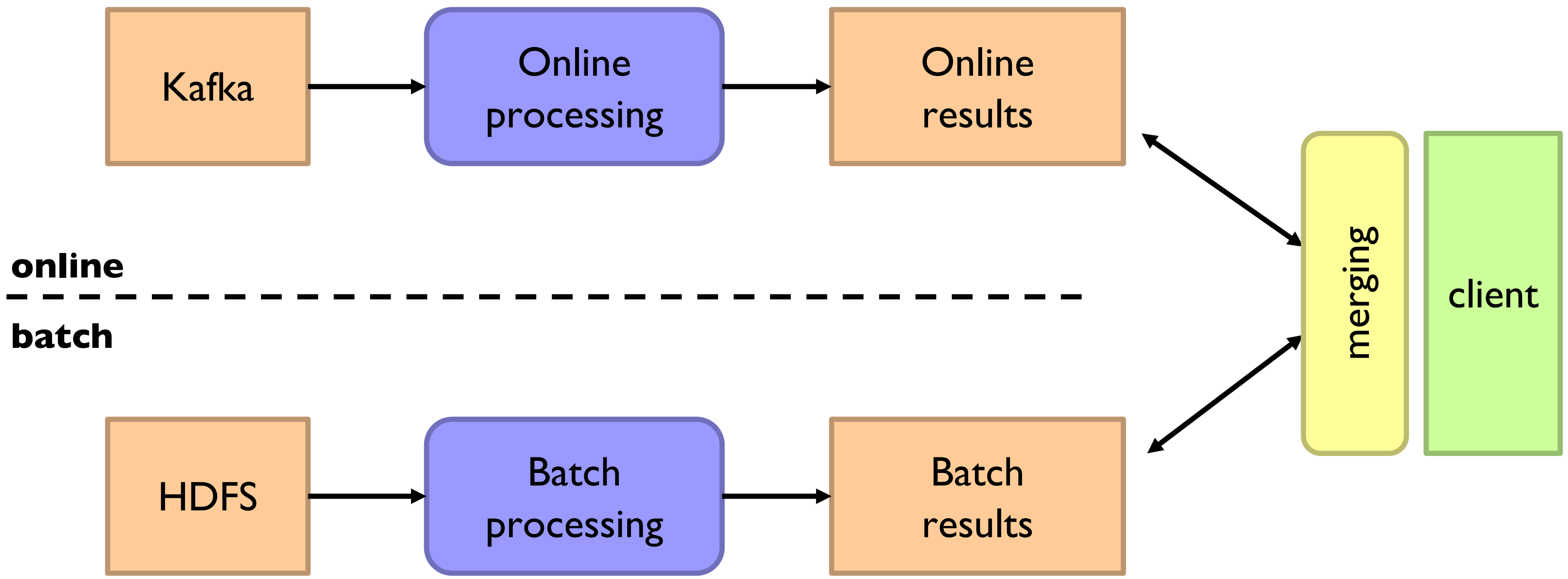
Kafka, Heron, Spark Streaming, Spark Structured Streaming, …

What about our cake?

Source: Wikipedia (Cake)

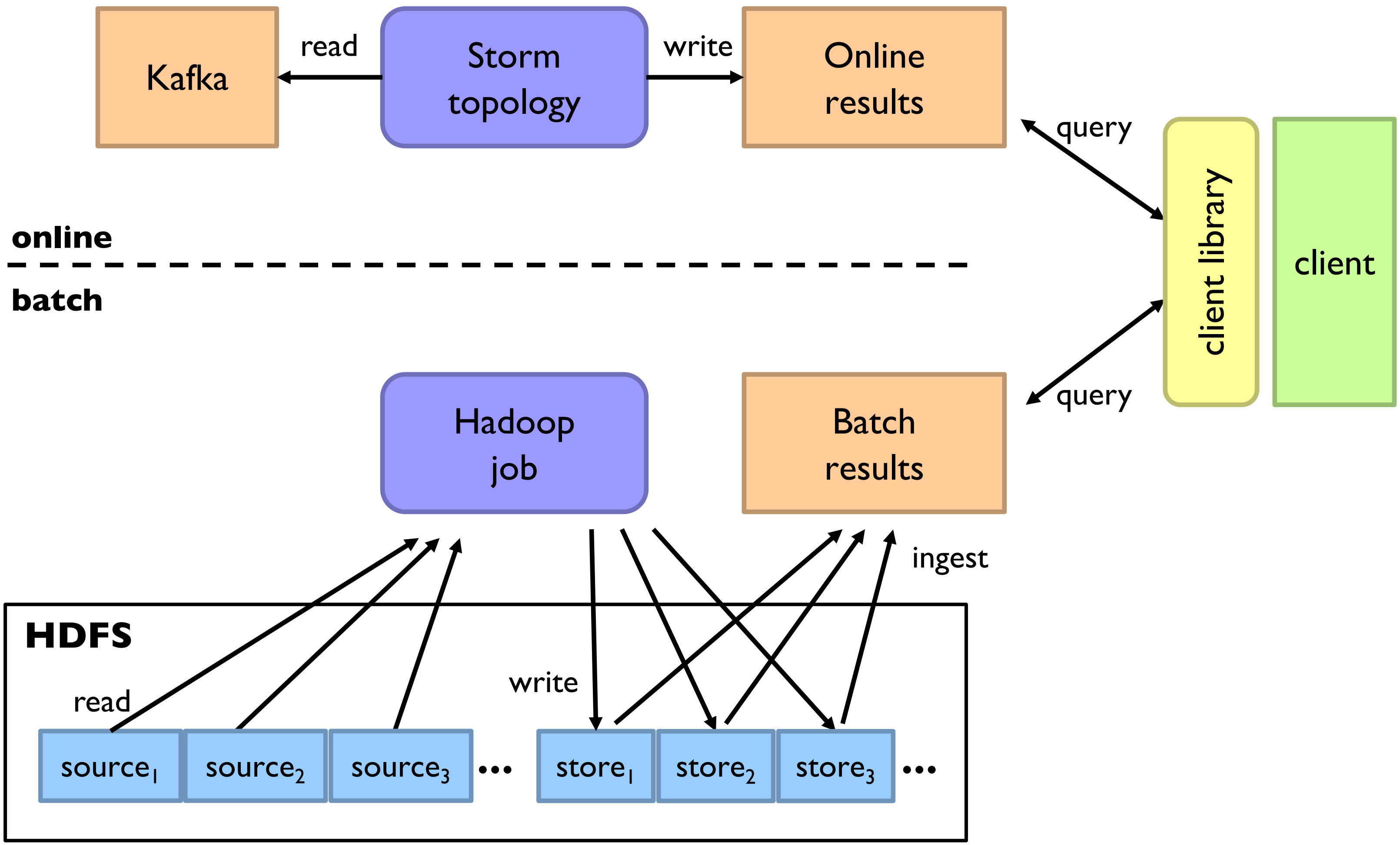
Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time



Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time

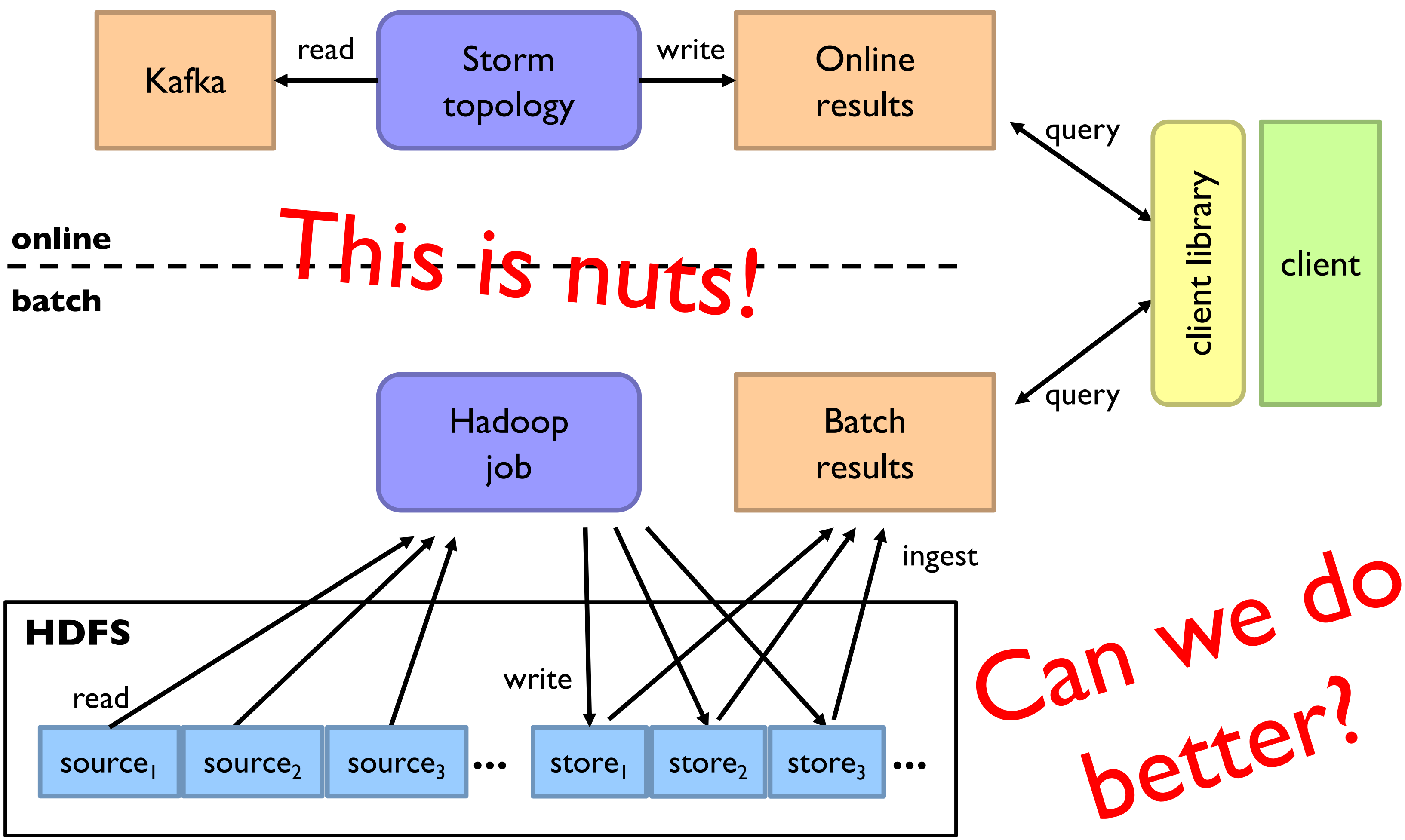


λ

(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

Boykin, Ritchie, O'Connell, and Lin. Summingbird: A Framework for Integrating
Batch and Online MapReduce Computations. PVLDB 7(13):1441-1451, 2014.

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 , $\oplus$)

$\oplus : M \times M \rightarrow M$, s.t., $\forall m_1, m_2, m_3 \ni M$

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

Monoid = Semigroup + identity

ε s.t., $\varepsilon \oplus m = m \oplus \varepsilon = m, \forall m \ni M$

Commutative Monoid = Monoid + commutativity

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

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

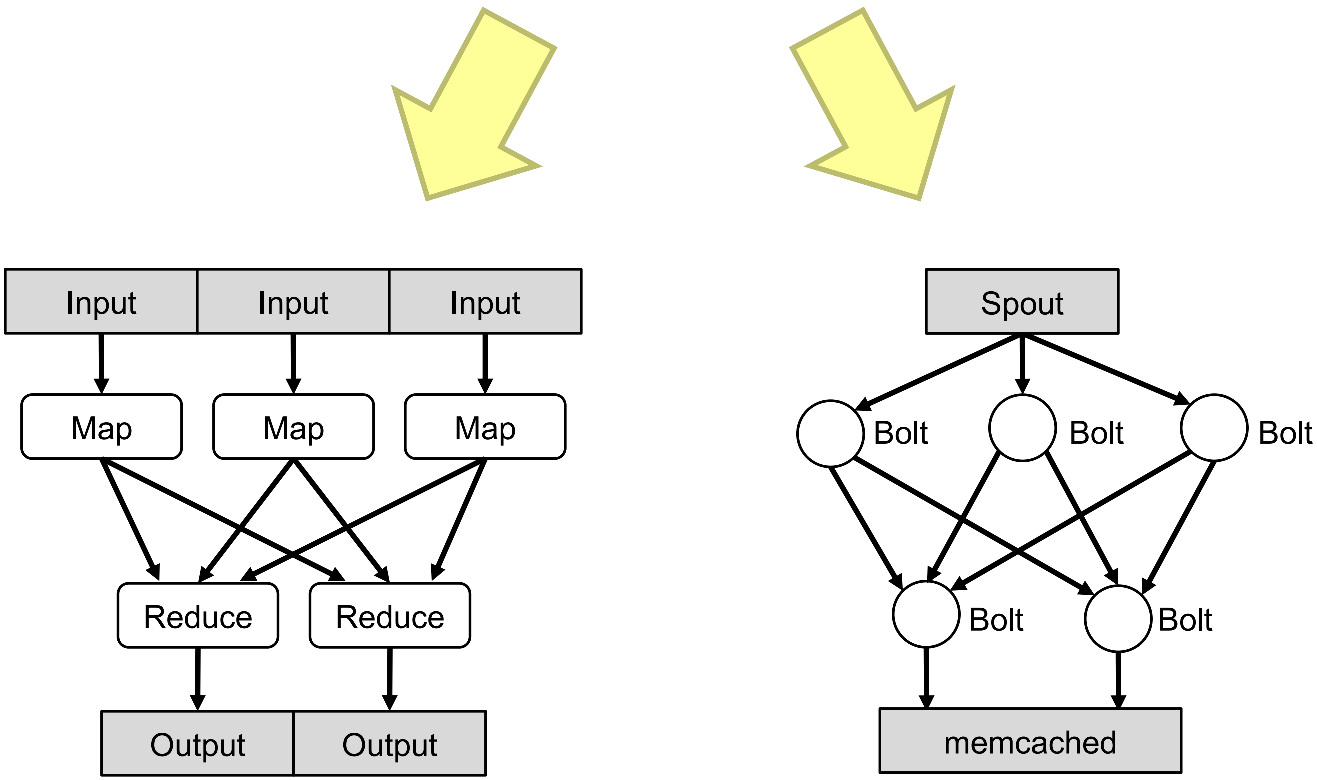
write to HDFS

Run on Storm

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

read from message queue

write to KV store


Input
Input
Input
Map
Map
Map
Reduce
Reduce
Output
Output
Spout
Bolt
Bolt
Bolt
Bolt
Bolt
memcached

“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

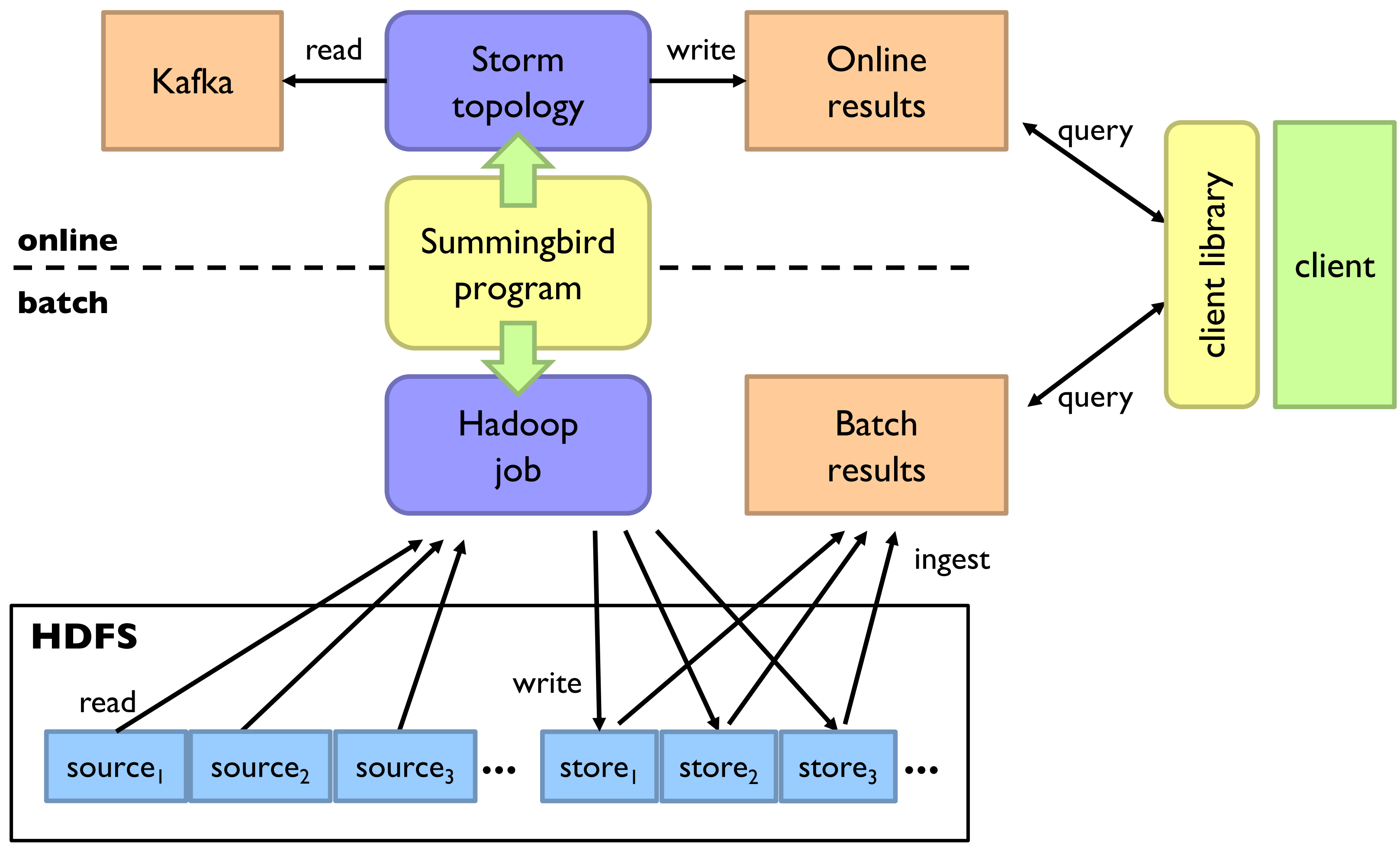
```
def wordCount[P <: Platform[P]]
  (source: Producer[P, Query],
   store: P#Store[Long, Map[String, Long]]) =
   source.flatMap { query =>
      (query.getHour, Map(query.getQuery -> 1L))
    }.sumByKey(store)
```

Approximate with CMS

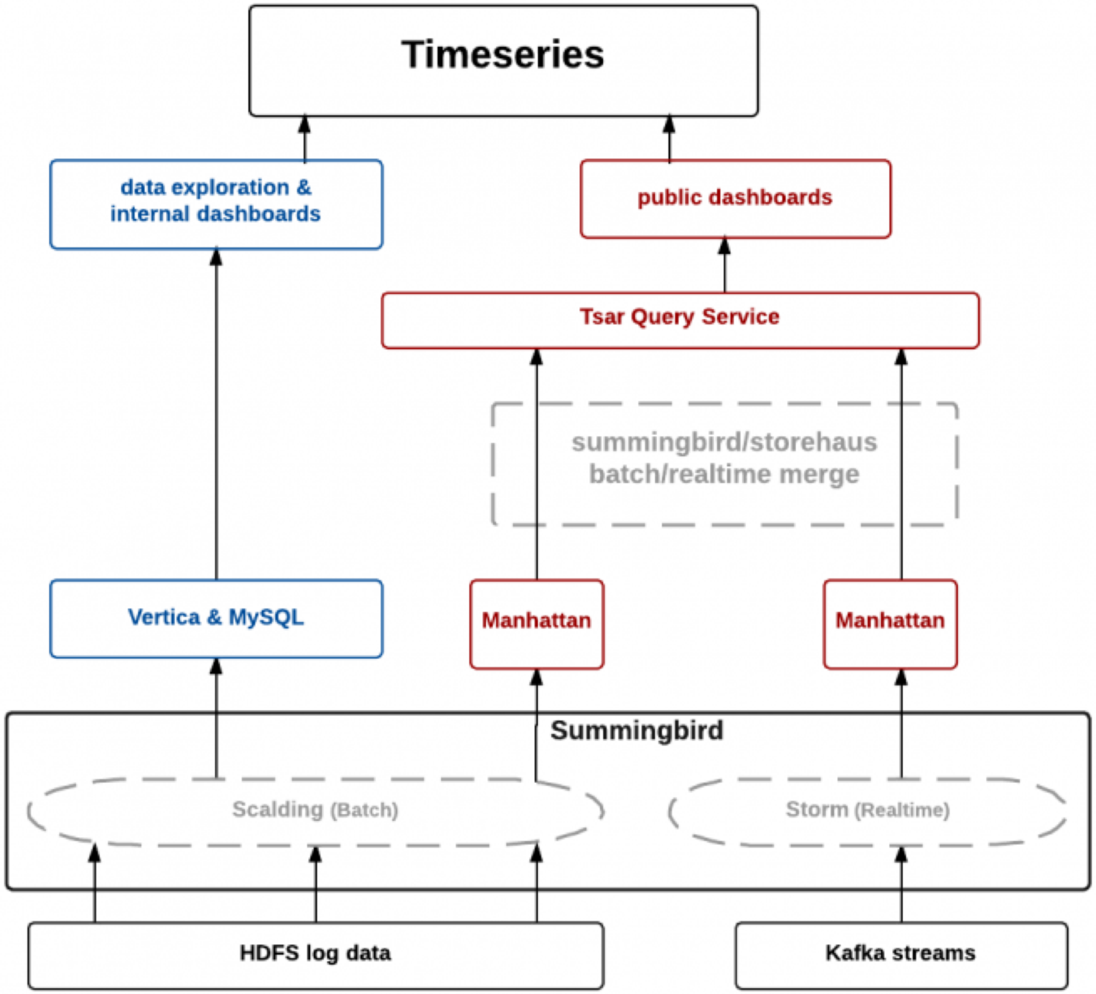
```
def wordCount[P <: Platform[P]]
  (source: Producer[P, Query],
   store: P#Store[Long, SketchMap[String, Long]])
  (implicit countMonoid: SketchMapMonoid[String, Long]) =
   source.flatMap { query =>
      (query.getHour,
       countMonoid.create((query.getQuery, 1L)))
    }.sumByKey(store)
```

Hybrid Online/Batch Processing

Example: count historical clicks and clicks in real time



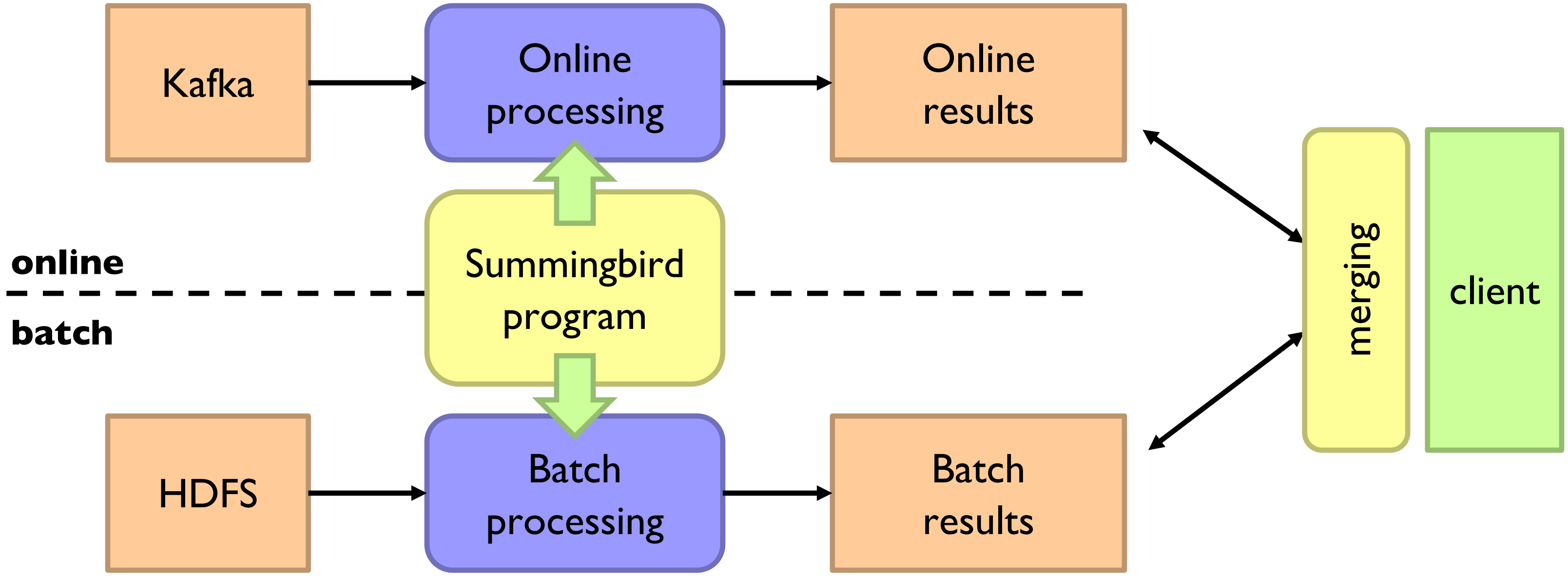
TSAR, a TimeSeries AggregatoR!



Source: https://blog.twitter.com/2014/tsar-a-timeseries-aggregator

Hybrid Online/Batch Processing

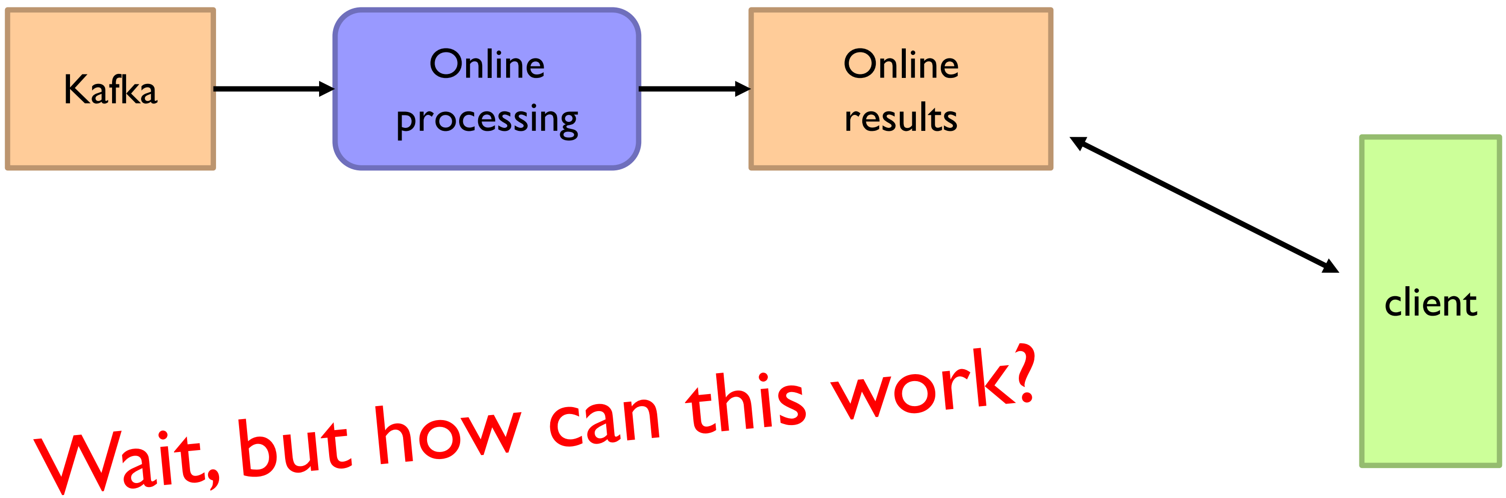
Example: count historical clicks and clicks in real time



But this is still too painful…

Hybrid Online/Batch Processing

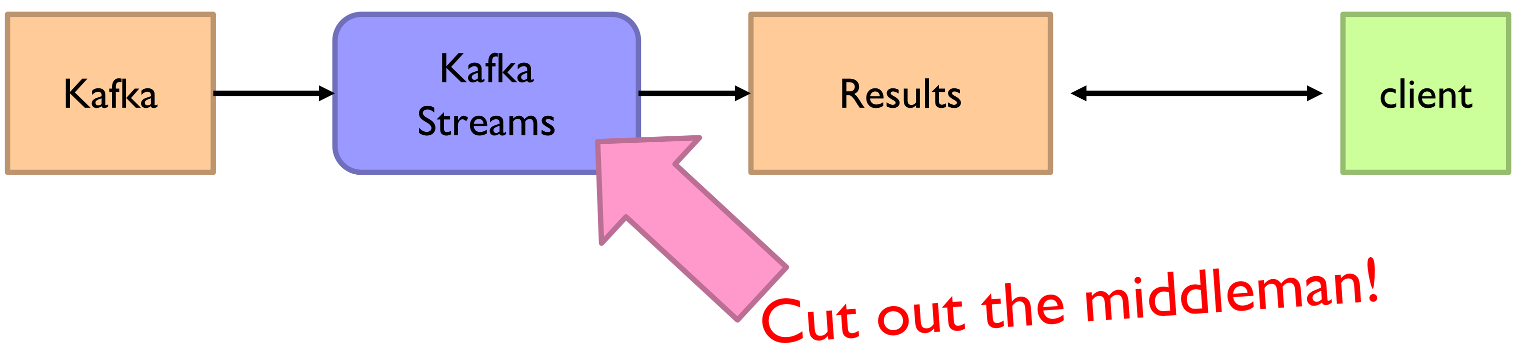
Example: count historical clicks and clicks in real time



Idea: everything is streaming

Batch processing is just streaming through a historic dataset!

Everything is Streaming!



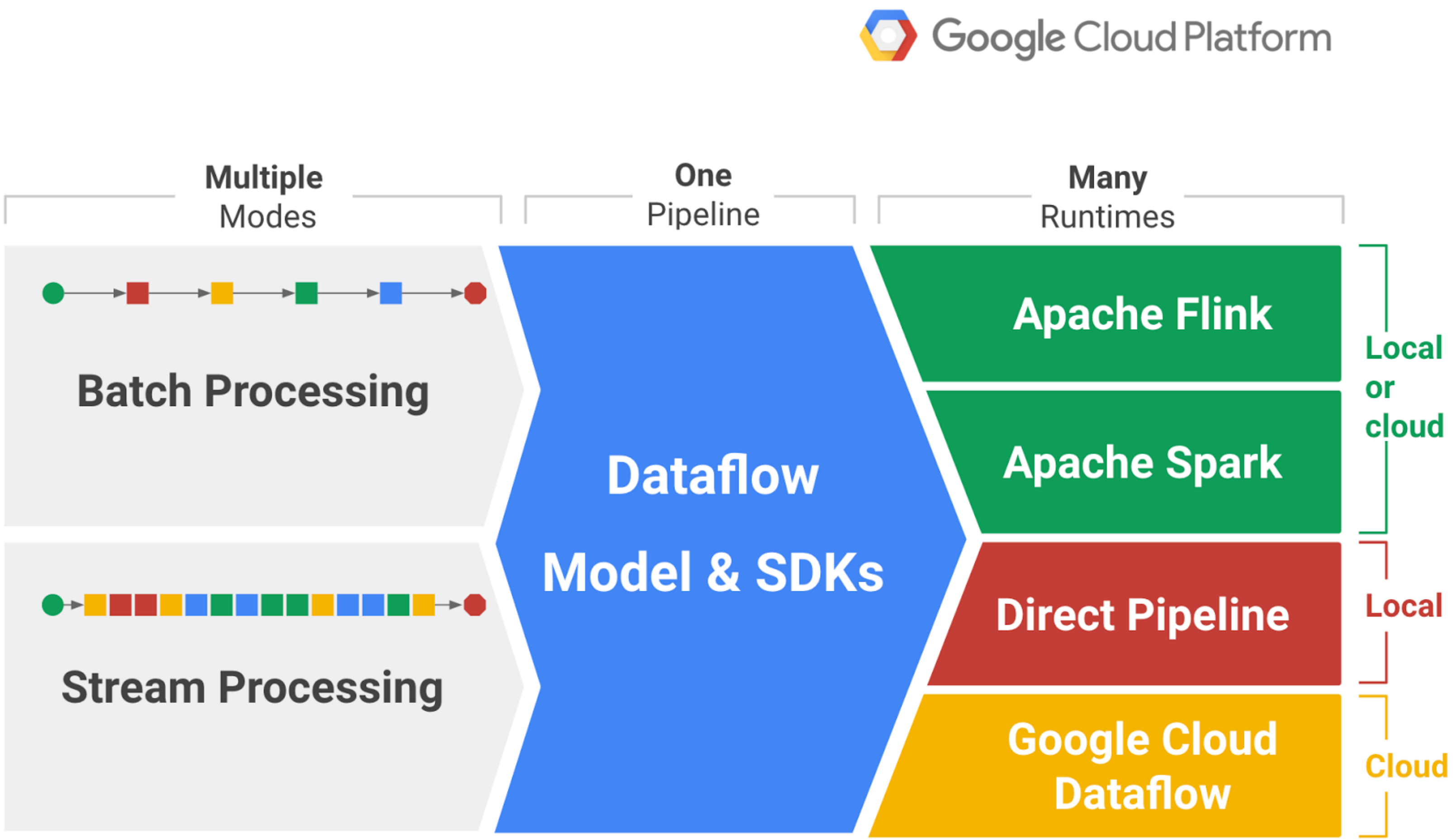

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> textLines = builder.stream("TextLinesTopic");
KTable<String, Long> wordCounts = textLines
    .flatMapValues(textLine ->
        Arrays.asList(textLine.toLowerCase().split("\\W+")))
    .groupBy((key, word) -> word)
    .count(Materialized.<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



Source: https://cloudplatform.googleblog.com/2016/01/Dataflow-and-open-source-proposal-to-join-the-Apache-Incubator.html

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)))
    .accumulatingAndRetractingFiredPanes())
 .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?

Source: flickr (https://www.flickr.com/photos/39414578@N03/16042029002)