



# Big Data Pipelines with Scala

Gleb Kanterov  
@kanterov

[github.com/kanterov/  
lambdacnf-2017-bigdata](https://github.com/kanterov/lambdacnf-2017-bigdata)

# Who am I?

- DATA @ Spotify
- ♥ Functional Programming and Cofree Coffee
- In Scala since 2012

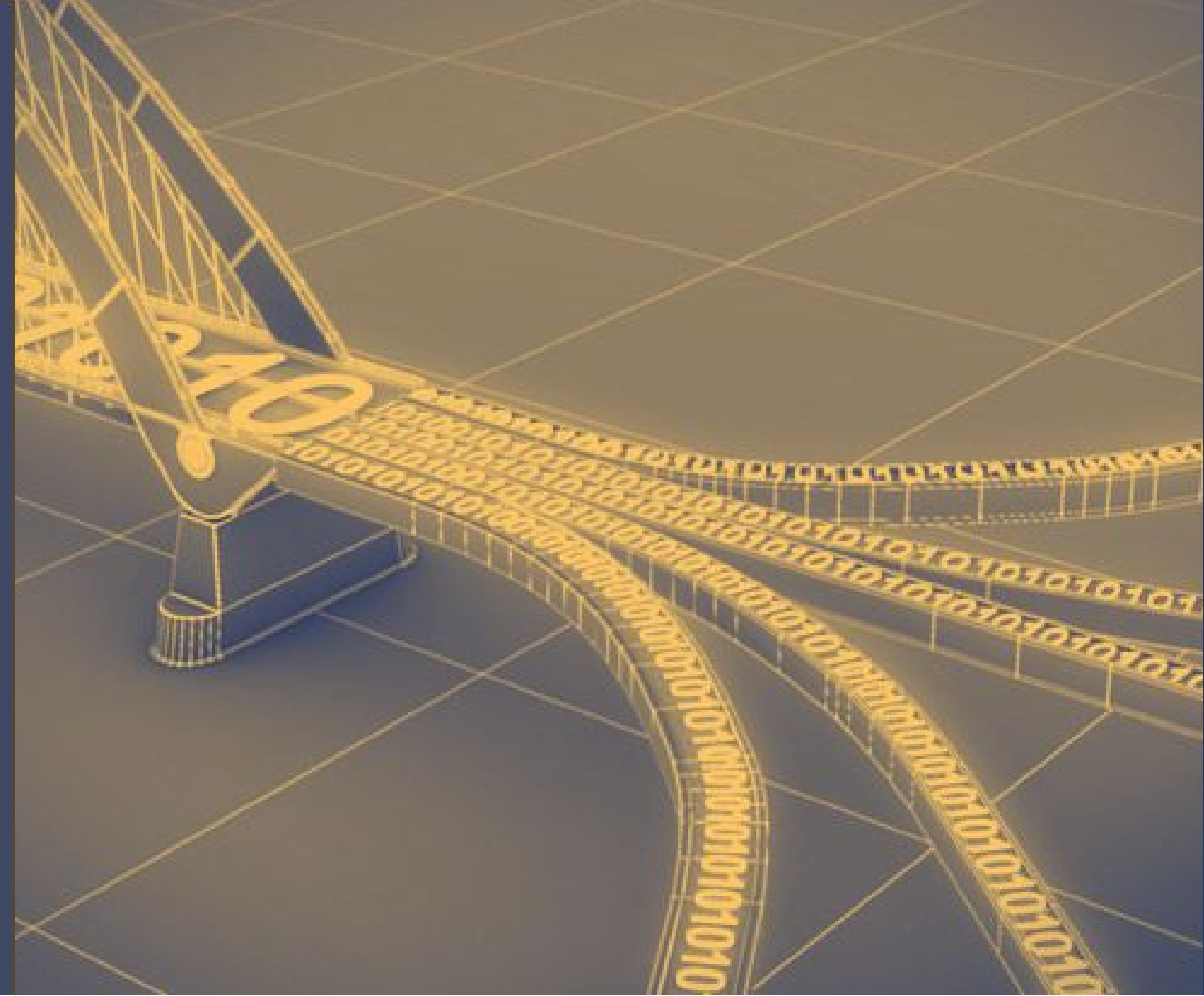


**We are hiring!**

**DATA** 



```
9 function addNumbers(a, b) {
10   return a + b;
11 };
12
13 // Takes the elements of an array and returns the total. Demonstrates simple
14 // recursion.
15 function totalForArray(arr, currentTotal) {
16   currentTotal = addNumbers(currentTotal + arr.shift());
17
18   if (arr.length > 0) {
19     return totalForArray(currentTotal, arr);
20   }
21   else {
22     return currentTotal;
23   }
24 }
25
26 // Or you could just use reduce
27 function totalForArray(arr) {
28   return arr.reduce(addNumbers);
29 }
30
31 // Should really be called divideTwoNumbers
32 function average(arr, count) {
33   return count / total;
34 }
35
36 function averageForArray(arr) {
37   return average(arr.length, totalForArray(arr));
38 }
39
40 // Gets the value associated with the property of an object. Needed for
41 // use with a collection method like map, hence the generator.
42 function getItem(propertyName) {
43   return function(item) {
44     return item[propertyName];
45   };
46 }
```



# Unit 01

**Batch Pipelines and Big Data**

# Unit 02

**Functional Programming and Big Data**

# Unit 03

**From Batch to Streaming**





# Fast Facts

- 100M+ Active Users
- 50M+ Subscribers
- 30M+ Songs
- 2B+ Playlists



# Hadoop@Spotify





# Hadoop@Spotify

- On-Premise
- 2,500 nodes
- 100 PB Disk
- 100 TB RAM
- 100B+ events per day
- 20K+ jobs per day







**Moving to  
Google Cloud**

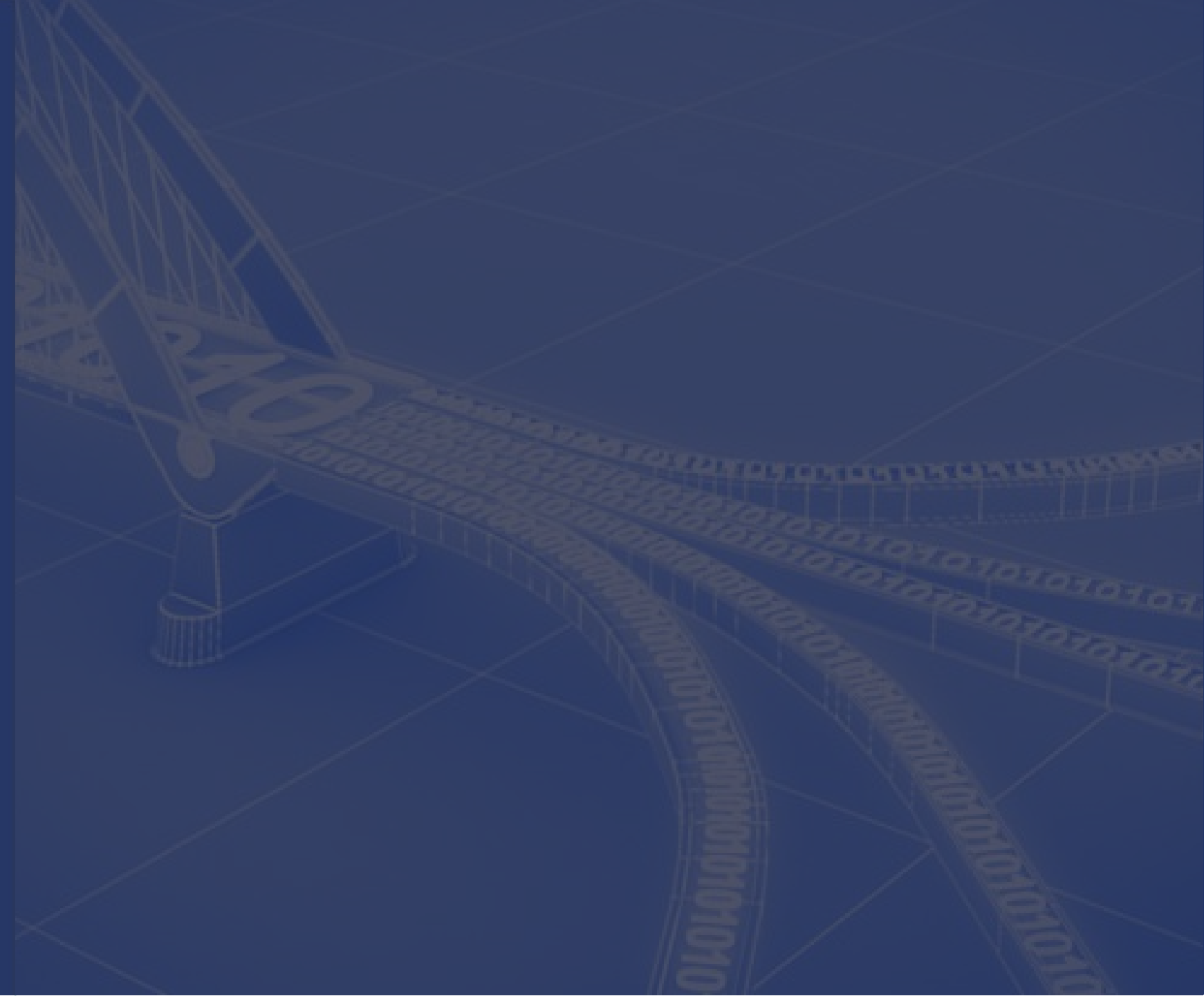
# The Evolution of Big Data Ecosystem







```
8 function addNumbers(a, b) {
9   return a + b;
10 };
11
12 // Takes the array and returns the total. Demonstrates simple
13 // recursion.
14 function totalForArray(arr, currentTotal) {
15   currentTotal = addNumbers(currentTotal, arr.shift());
16
17   if (arr.length > 0) {
18     return totalForArray(currentTotal, arr);
19   }
20   else {
21     return currentTotal;
22   }
23 }
24
25 // Or you could just use reduce
26 function totalForArray(arr) {
27   return arr.reduce(addNumbers);
28 }
29
30 // Should really be called findTotalNumber
31 function average(arr, count) {
32   return count / total;
33 }
34
35 function averageForArr(arr) {
36   return average(arr.length, totalForArray(arr));
37 }
38
39 // Gets the value associated with the property of an object. Needed for
40 // using the reduce function method like map, hence the totalForArray
41 function getProp(propertyName) {
42   return function(item) {
43     return item[propertyName];
44   }
45 }
```



# Unit 01

**Batch Pipelines and  
Big Data**

# Unit 02

**Functional Programming and  
Big Data**

# Unit 03

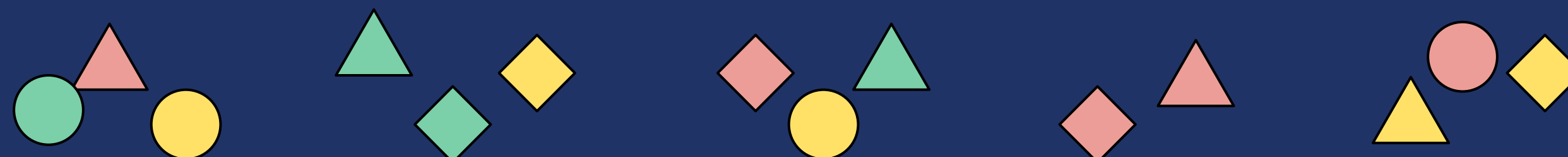
**From Batch to  
Streaming**

# MapReduce: Batch Processing

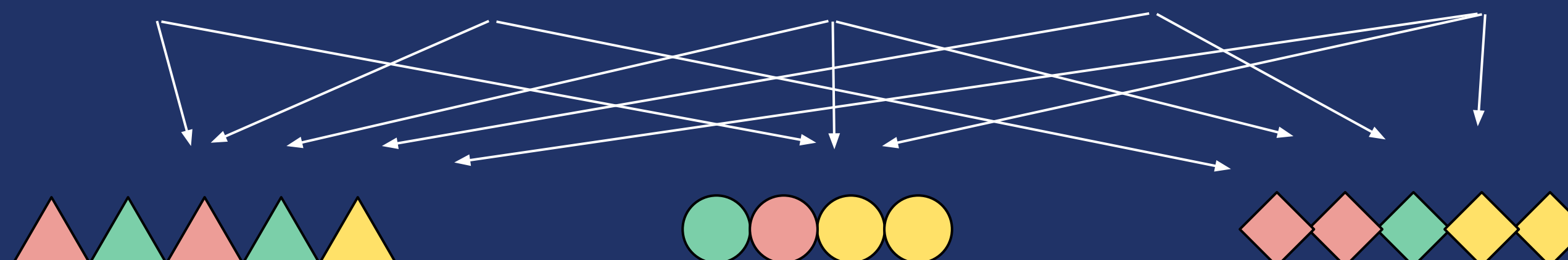
(Prepare)



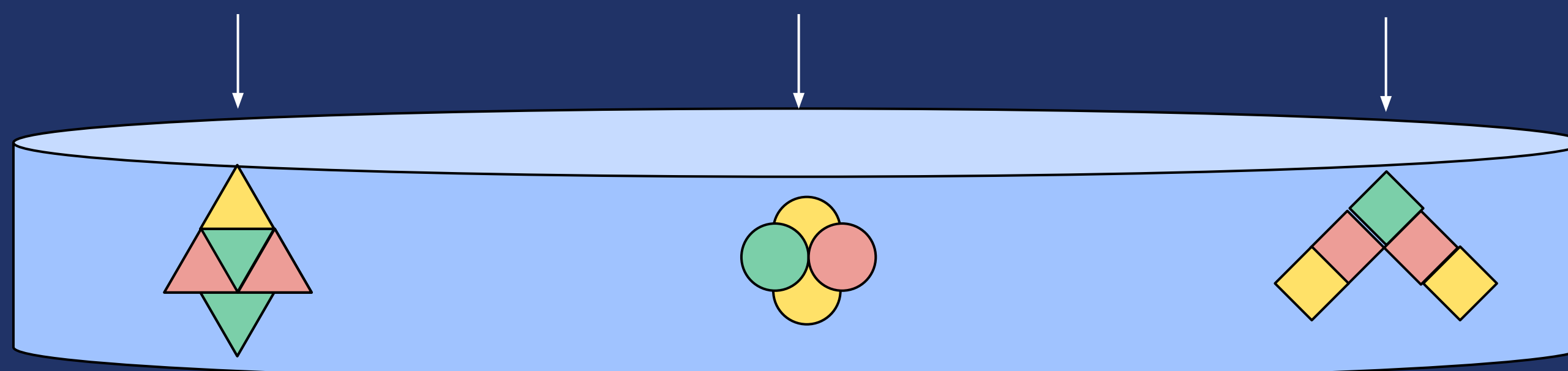
Map



(Shuffle)



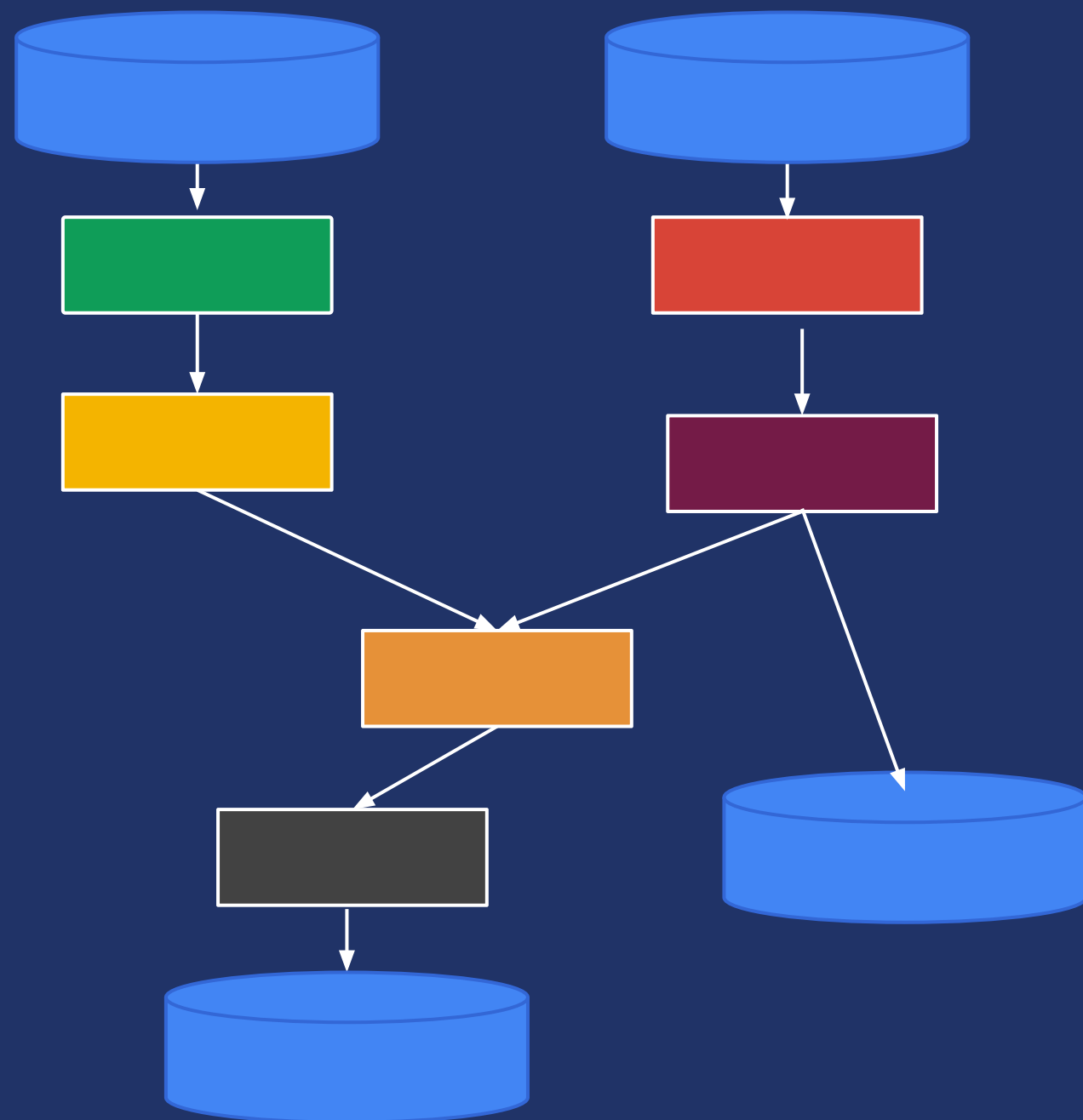
Reduce



(Produce)



# Apache Beam



- High Level API
- User provides DAG of computations to be performed
- SDK available for Java and Python
- Runs on
  - Google Dataflow
  - Apache Spark
  - Apache Apex
  - Apache Flink

***Scio***

**Latin /'ʃi.o/, ['ʃi:.o]**

**Verb: I can, know, understand,  
have knowledge.**



```
case class TrackCount(trackId: TrackId, plays: Long)
```

```
case class PlayTrack(userId: UserId, trackId: TrackId)
```

```
playTracks
```

```
.map(track => (track.trackId, 1L))
```

```
.reduceByKey(_ + _)
```

```
.map { case (trackId, plays) => TrackCount(trackId, plays) }
```

```
trait SCollection[A] {  
  def map[B](f: A => B): SCollection[B]  
  def flatMap[B](f: A => Iterable[B]): SCollection[B]  
  def filter(f: A => Boolean): SCollection[A]  
}  
  
implicit class PairSCollectionOps[K, V](  
  self: SCollection[(K, V)]) {  
  
  def reduceByKey(f: (V, V) => V): SCollection[(K, V)]  
  def sum(implicit V: Semigroup[V]): SCollection[(K, V)]  
}
```



# Unit 01.

# 1. open sbt and run pipeline

```
$ ./sbt shell
```

```
> unit-1
```

```
> test-unit-1
```

```
> avro-read in/play_track/
```

```
> avro-read out/play_count/
```



**2. output total content hours per user**

**3. output top n users by content  
hours**

```
9 function addNumbers(a, b) {
10   return a + b;
11 };
12
13 // Takes the elements of an array and returns the total. Demonstrates simple
14 // recursion.
15 function totalForArray(arr, currentTotal) {
16   currentTotal = addNumbers(currentTotal + arr.shift());
17
18   if (arr.length > 0) {
19     return totalForArray(currentTotal, arr);
20   }
21   else {
22     return currentTotal;
23   }
24 }
25
26 // Or you could just use reduce
27 function totalForArray(arr) {
28   return arr.reduce(addNumbers);
29 }
30
31 // Should really be called provideTwoNumbers
32 function average(total, count) {
33   return count / total;
34 }
35
36 function averageForArray(arr) {
37   return average(arr.length, totalForArray(arr));
38 }
39
40 // Gets the value associated with the property of an object. Intended for
41 // use with a collection method like map, hence the generator.
42 function getItem(propertyName) {
43   return function(item) {
44     return item[propertyName];
45   };
46 }
```

# Unit

# 01

Batch Pipelines and  
Big Data

# Unit

# 02

Functional Programming and  
Big Data

# Unit

# 03

From Batch to  
Streaming

# Immutable Data

```
case class TrackCount(trackId: TrackId, plays: Long)
```

```
case class PlayTrack(userId: UserId, trackId: TrackId)
```

```
playTracks  
  .map(track => (track.trackId, 1L))  
  .reduceByKey(_ + _)  
  .map { case (trackId, plays) => TrackCount(trackId, plays) }
```



# Second-Order Functions

```
case class TrackCount(trackId: TrackId, plays: Long)
```

```
case class PlayTrack(userId: UserId, trackId: TrackId)
```

```
playTracks
```

```
.map(track => (track.trackId, 1L))
```

```
.reduceByKey(_ + _)
```

```
.map { case (trackId, plays) => TrackCount(trackId, plays) }
```

# First-Class Functions and Lambdas

```
case class TrackCount(trackId: TrackId, plays: Long)
```

```
case class PlayTrack(userId: UserId, trackId: TrackId)
```

```
playTracks
```

```
.map(track => (track.trackId, 1L))
```

```
.reduceByKey(_ + _)
```

```
.map { case (trackId, plays) => TrackCount(trackId, plays) }
```

# Pattern-Matching

```
case class TrackCount(trackId: TrackId, plays: Long)
```

```
case class PlayTrack(userId: UserId, trackId: TrackId)
```

```
playTracks  
  .map(track => (track.trackId, 1L))  
  .reduceByKey(_ + _)  
  .map { case (trackId, plays) => TrackCount(trackId, plays) }
```



# Composition

```
def contentHours(playTracks: SCollection[PlayTrack]):  
    SCollection[ContentHours] = ...  
  
def topUsers(n: Int, playTracks: SCollection[PlayTrack]):  
    SCollection[ContentHours] = {  
        contentHours(playTracks)  
        .top(n)(Ordering.by(_.msPlayedSum))  
        .flatMap(x => x)  
    }
```

# Unit 02.

# **What Can Go Wrong**



# Problem: Skewed Data

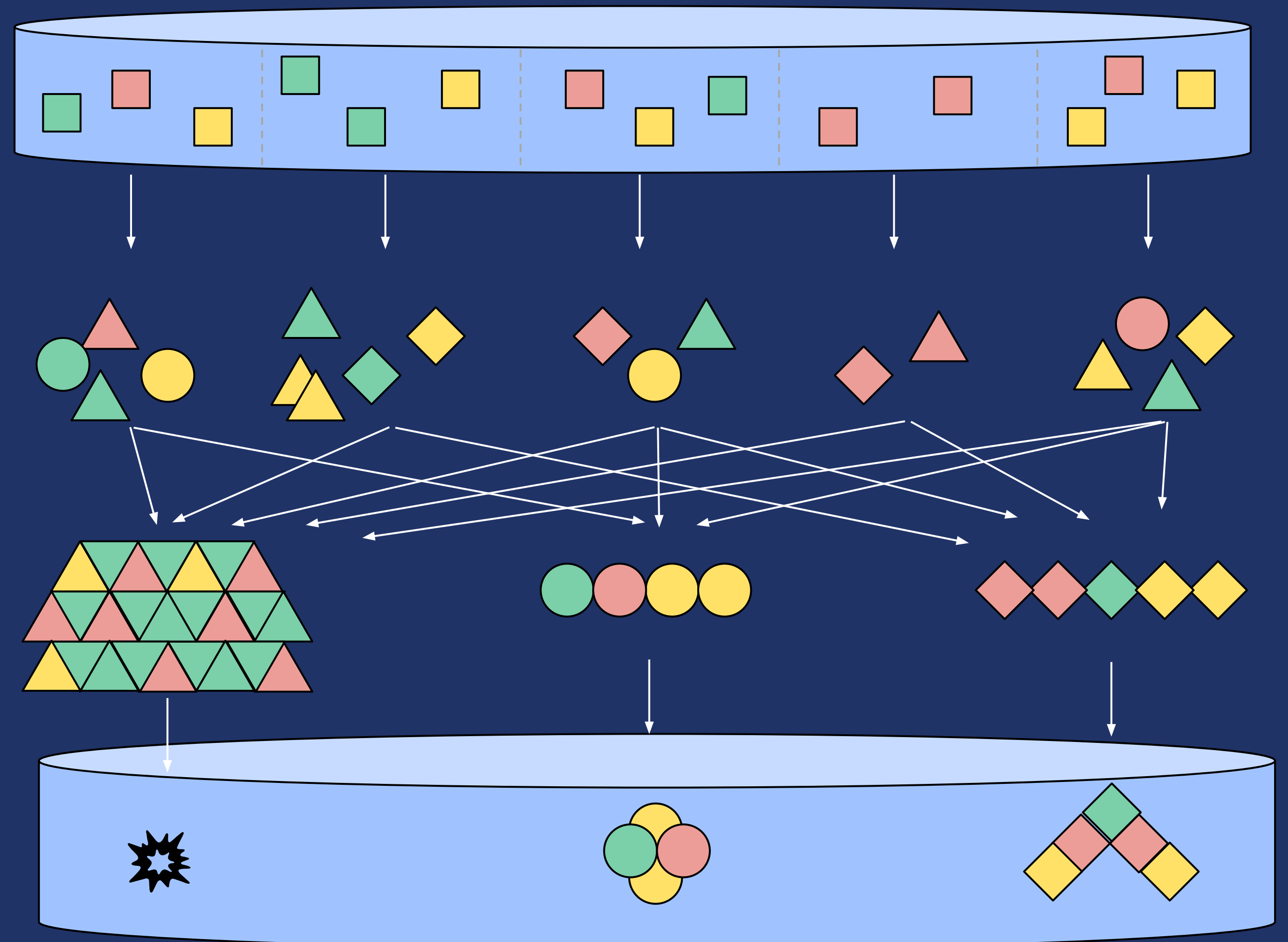
(Prepare)

Map

(Shuffle)

Reduce

(Produce)



# Hot Key Fanout

```
def withHotKeyFanout(hotKeyFanout: K => Int):  
  SCollectionWithHotKeyFanout[K, V]
```

- adds intermediate step to reduce skewed keys
- **hotKeyFanout** determines number of nodes

# Hash Join

```
def hashJoin[W](that: SCollection[(K, W)]):  
    SCollection[(K, (V, W))]
```

- replicates that to all workers
- right side should be tiny and fit into memory

# Skewed Join

```
def skewedJoin[W](that: SCollection[(K, W)]):  
    SCollection[(K, (V, W))]
```

- adds intermediate step with **hashJoin** of hot keys
- uses Count-Min-Sketch to find hot keys

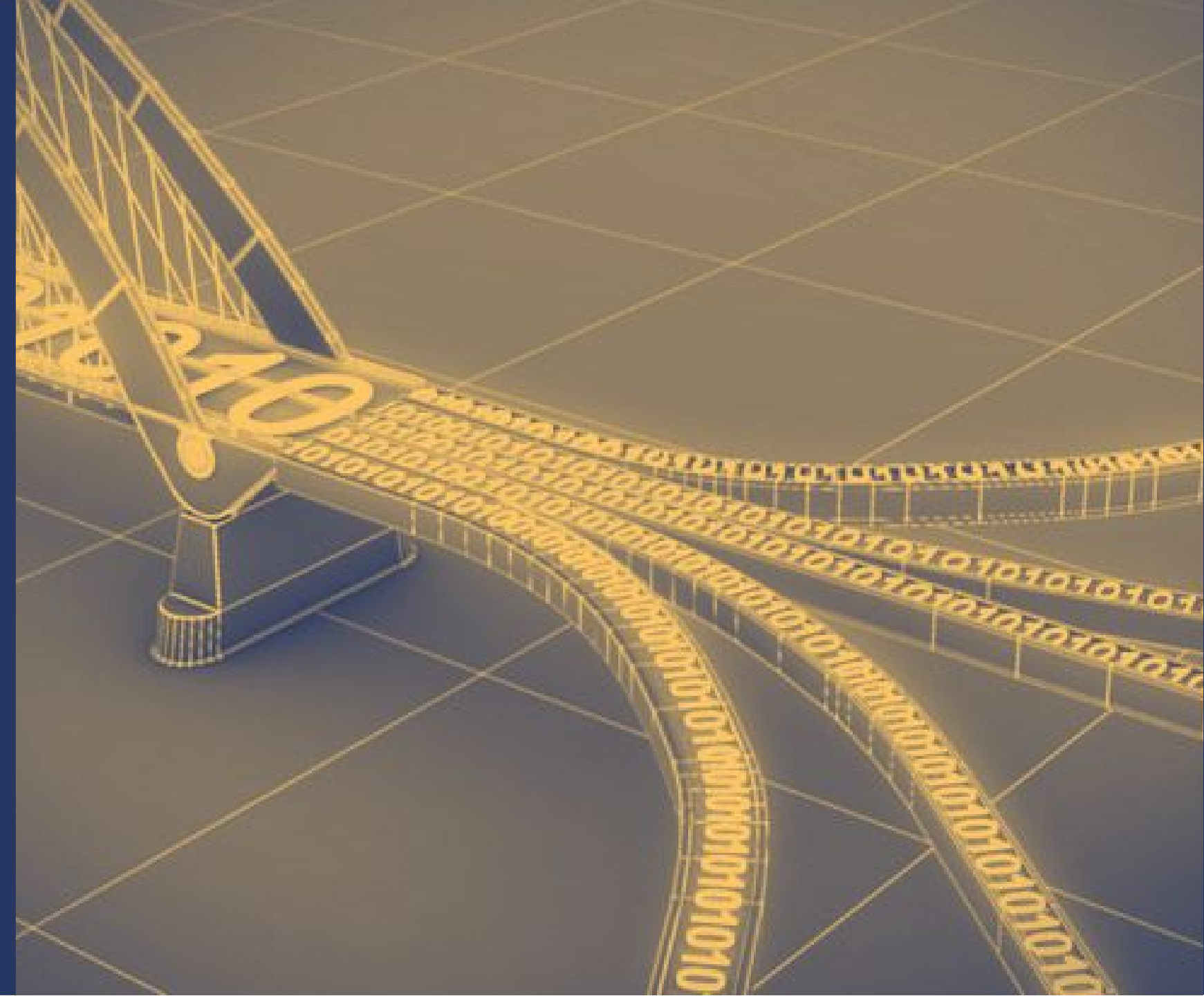


# Testing your Code

See `./src/test/scala/us.lambdaconf/  
4_tests.scala`



```
9 function addNumbers(a, b) {
10   return a + b;
11 };
12
13 // Takes the array and returns the total. Demonstrates simple
14 // recursion.
15 function totalForArray(arr, total) {
16   currentTotal = addNumbers(currentTotal, arr.shift());
17
18   if (arr.length < 0) {
19     return totalForArray(currentTotal, arr);
20   }
21   else {
22     return currentTotal;
23   }
24 }
25
26 // Or you could just use reduce
27 function totalForArray(arr) {
28   return arr.reduce(addNumbers);
29 }
30
31 // Should really be called findTwoNumbers
32 function average(count, total) {
33   return count / total;
34 }
35
36 function averageForArr(arr) {
37   return average(arr.length, totalForArray(arr));
38 }
39
40 // Gets the value associated with the property of an object. Needed for
41 // using the object method like map, hence the property name is needed for
42 function getProp(propertyName) {
43   return function(item) {
44     return item[propertyName];
45   };
46 }
```



# Unit 01

Batch Pipelines and  
Big Data

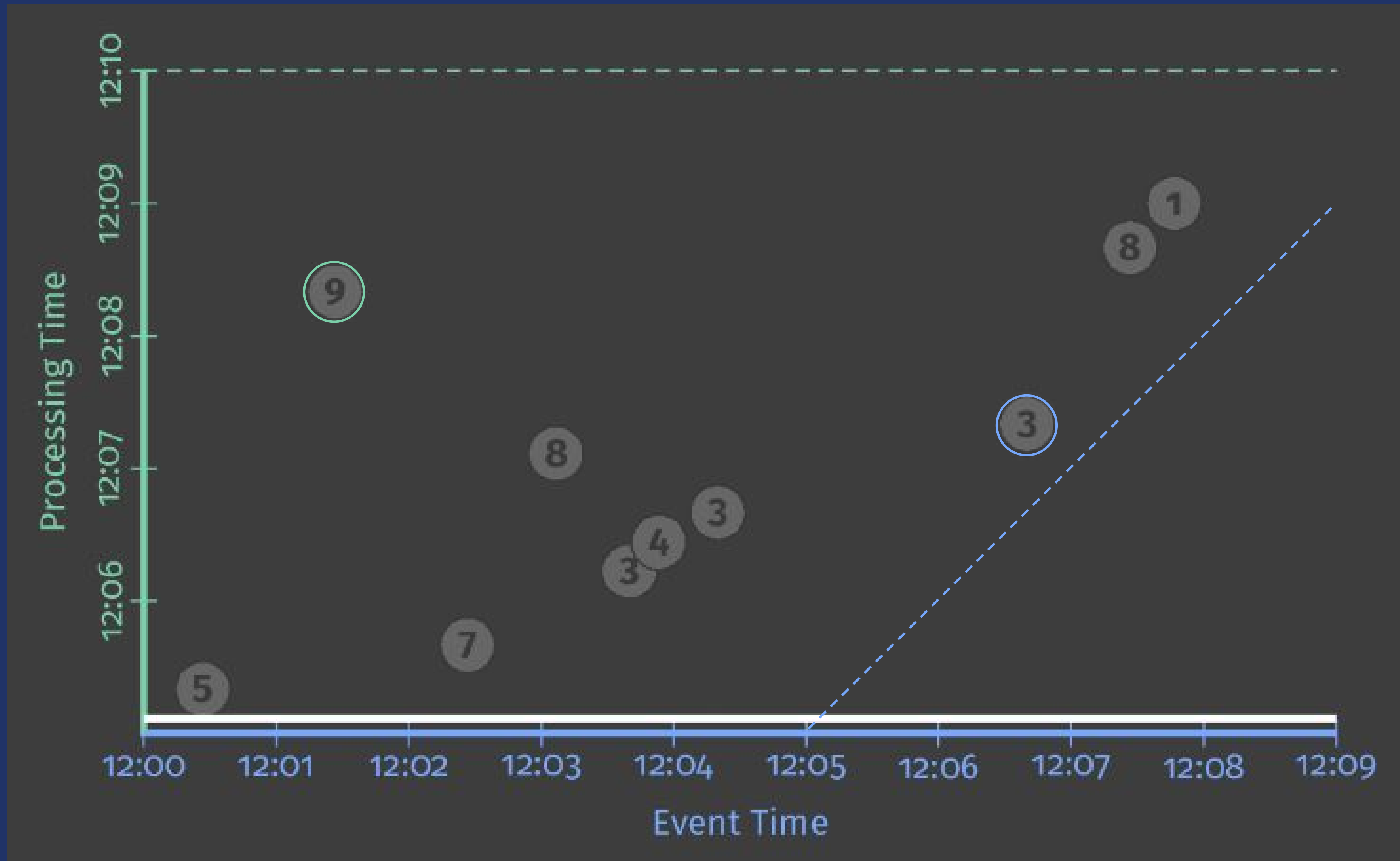
# Unit 02

Functional Programming and  
Big Data

# Unit 03

From Batch to  
Streaming

# Event Time vs. Processing Time





**What** are you computing?

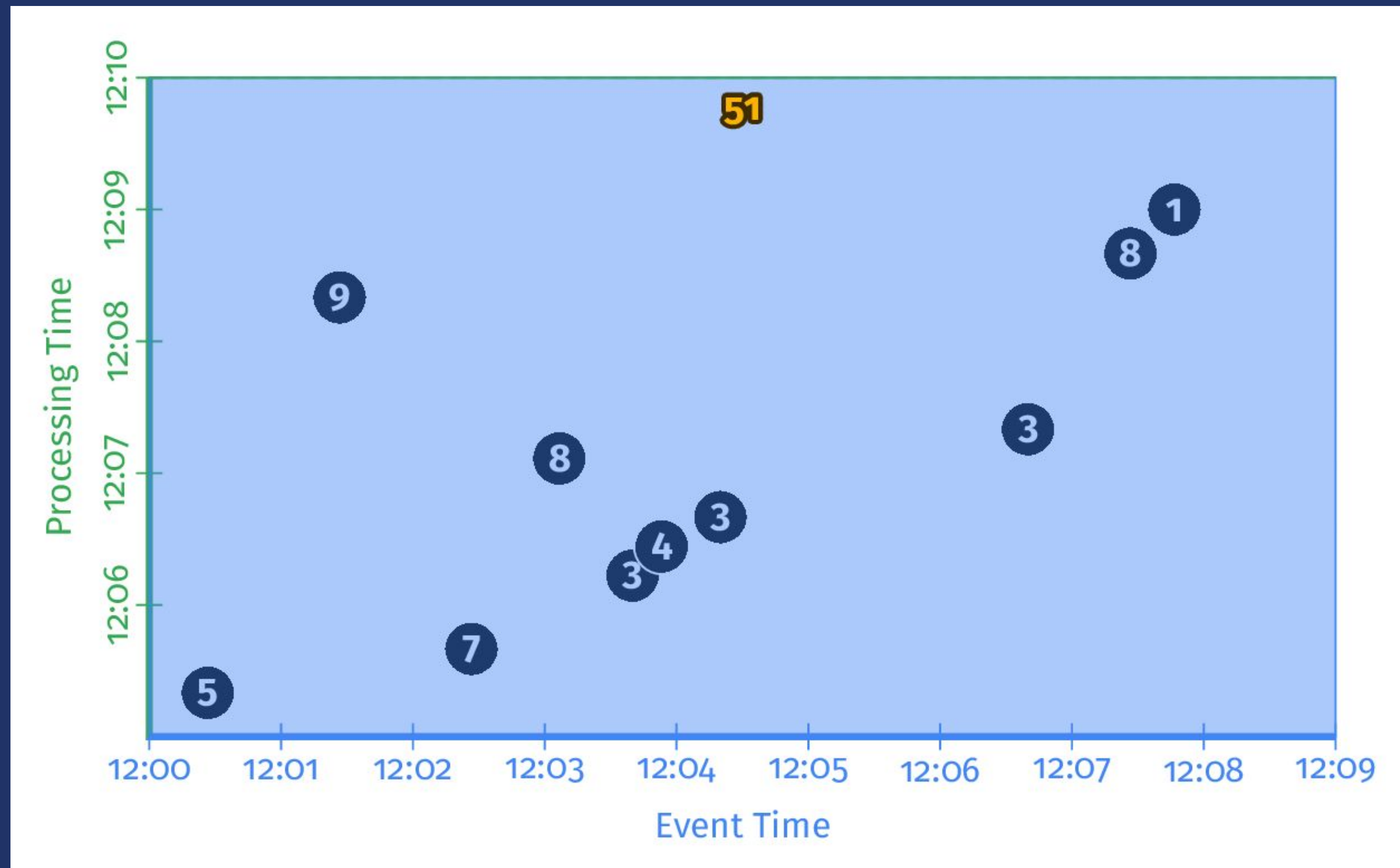
**Where** in event time?

**When** in processing time?

**How** do refinements relate?

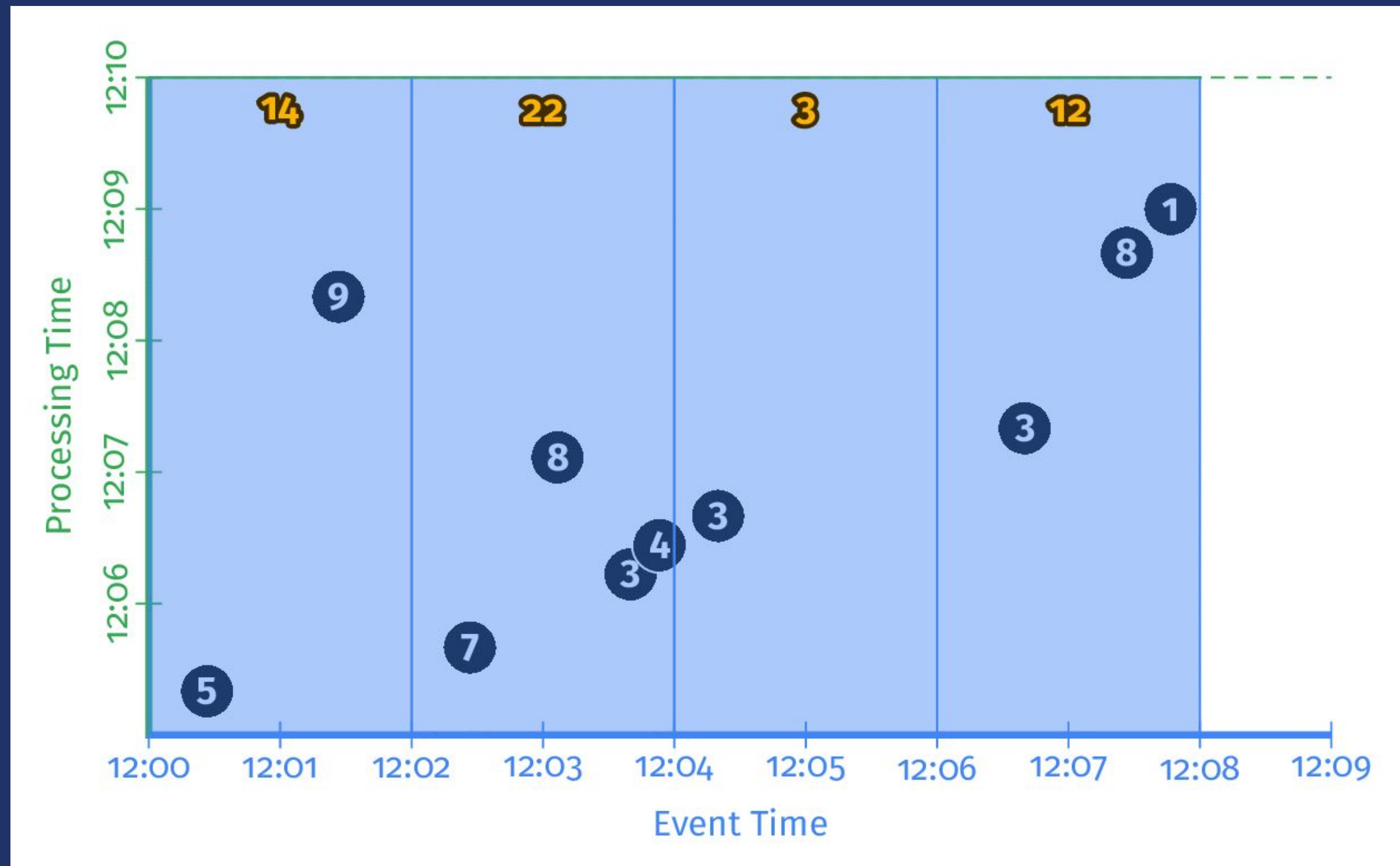
# The Beam Model: **What** is Being Computed?

```
playTracks  
  .map(_.msPlayed)  
  .sum
```



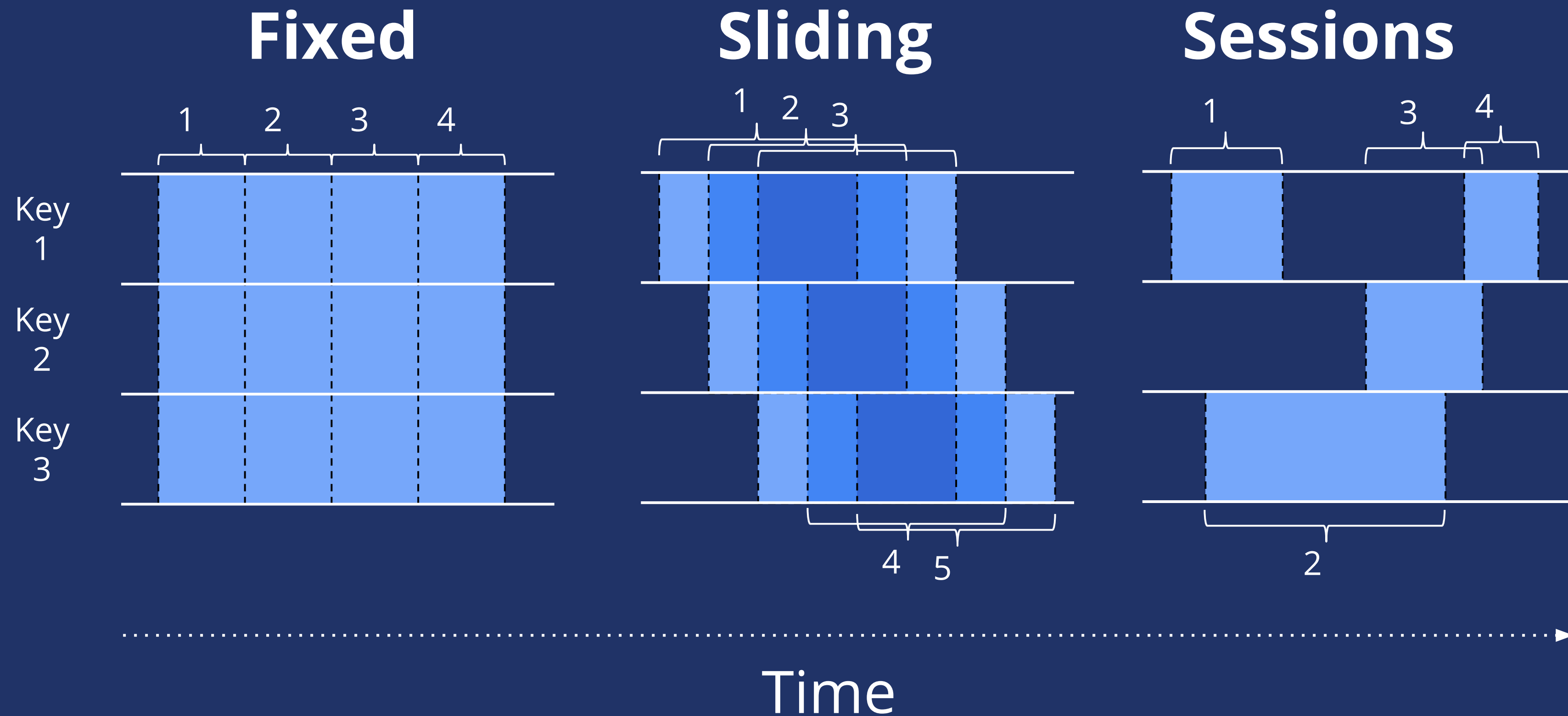
# The Beam Model: **Where** in Event Time?

```
playTracks  
  .map(_.msPlayed)  
  .withFixedWindows(minutes(2))  
  .sum
```



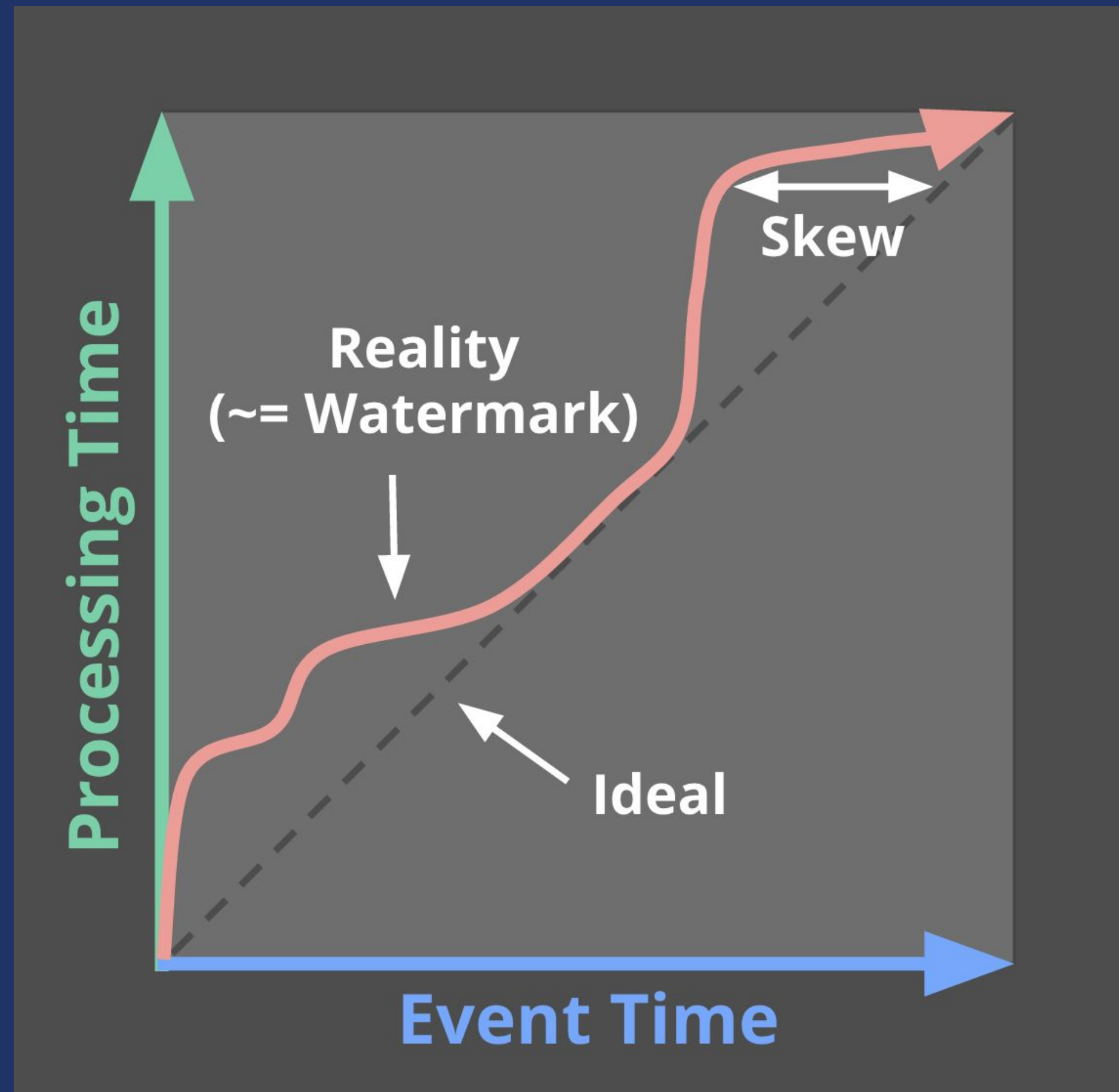
# Where in event time?

Windowing divides data into event-time-based finite chunks.



Often required when doing aggregations over unbounded data.

# When in processing time?

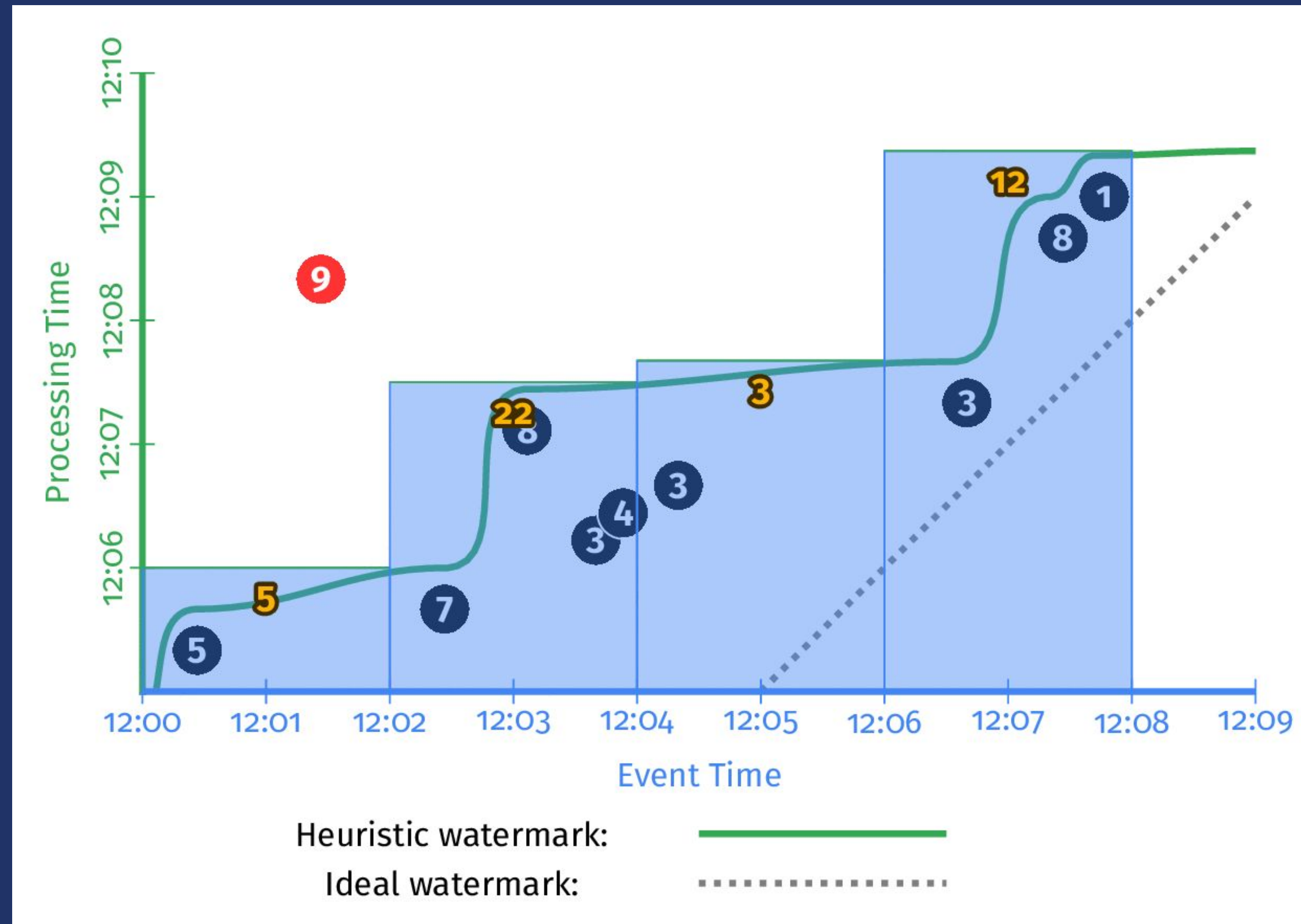


- **Triggers** control when results are emitted.
- Triggers are often relative to the watermark.



# The Beam Model: **When** in Processing Time?

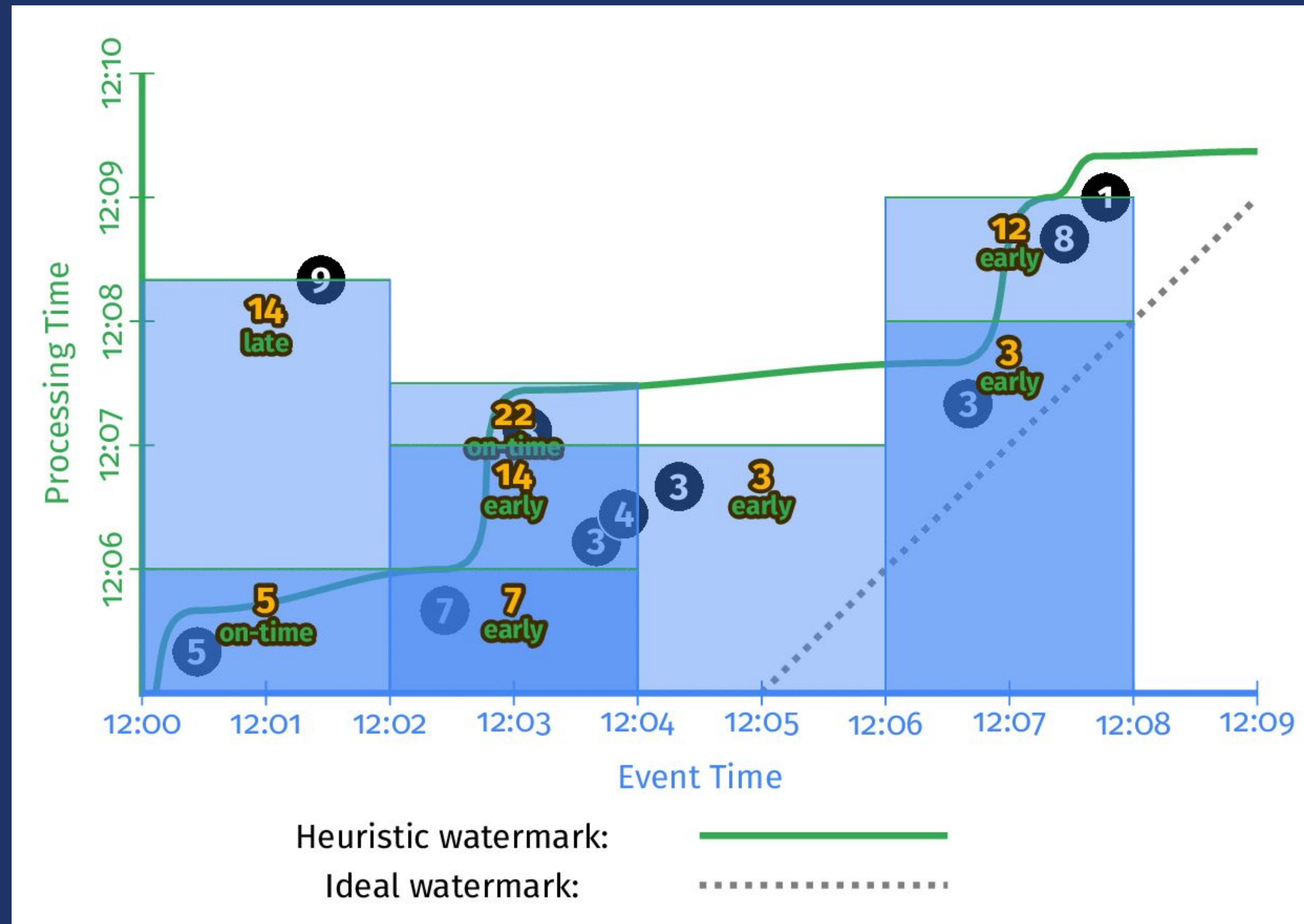
```
val trigger =  
  AfterWatermark.pastEndOfWindow()  
  
val options = WindowOptions(trigger)  
  
playTracks  
  .map(_.msPlayed)  
  .withFixedWindows(  
    duration = minutes(2),  
    options = options)  
  .sum
```



# The Beam Model: **How** do Refinements Relate?

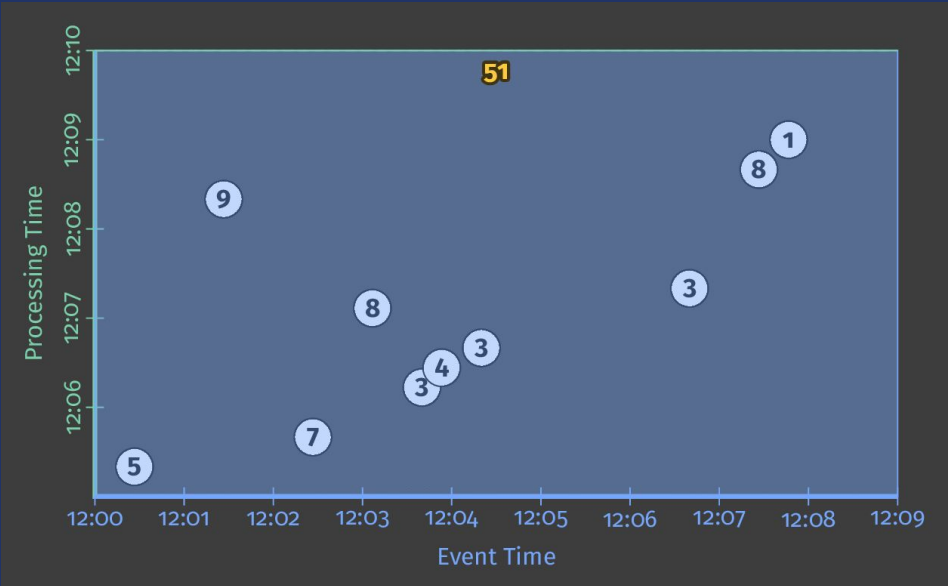
```
val trigger = AfterEach.inOrder(  
  Repeatedly.forever(  
    AfterProcessingTime  
      .pastFirstElementInPane()  
      .plusDelayOf(minutes(1))  
  ).orFinally(  
    AfterWatermark.pastEndOfWindow()),  
  Repeatedly.forever(  
    AfterPane  
      .elementCountAtLeast(1)))
```

```
val options = WindowOptions(  
  trigger = trigger,  
  allowedLateness = days(1),  
  accumulationMode =  
    ACCUMULATING_FIRED_PANES)
```

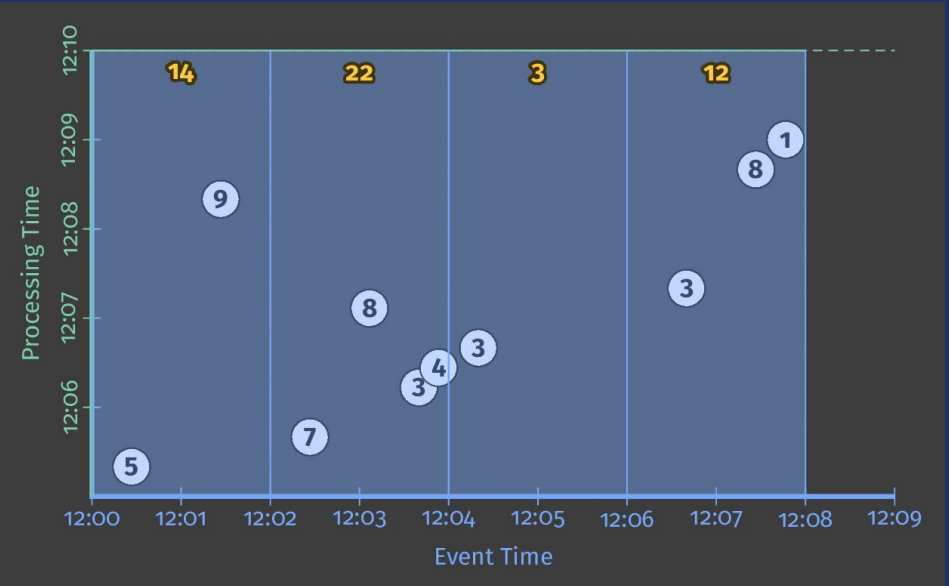




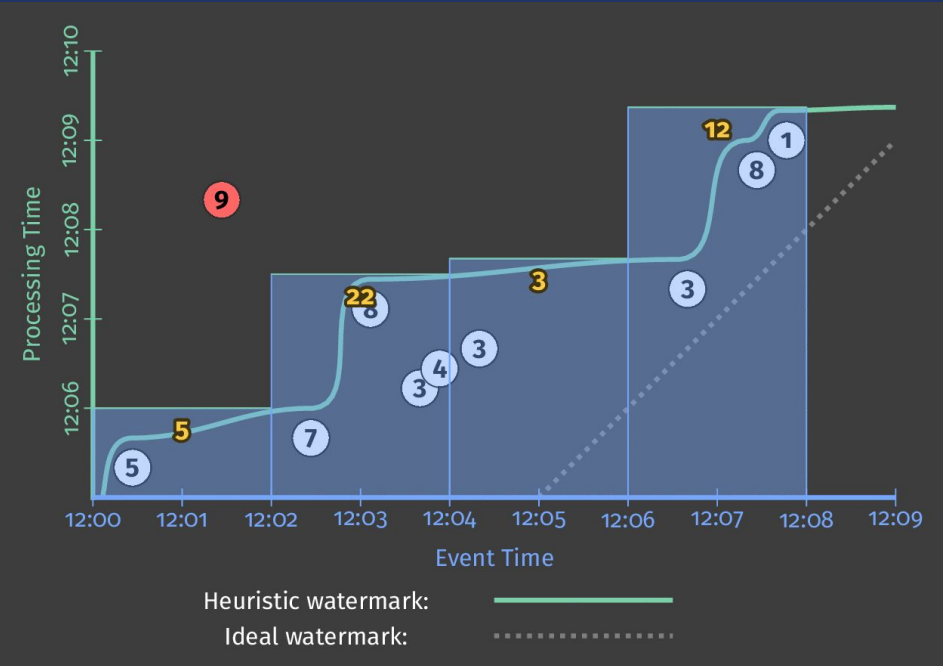
# Customizing **What** **When** **Where** **How**



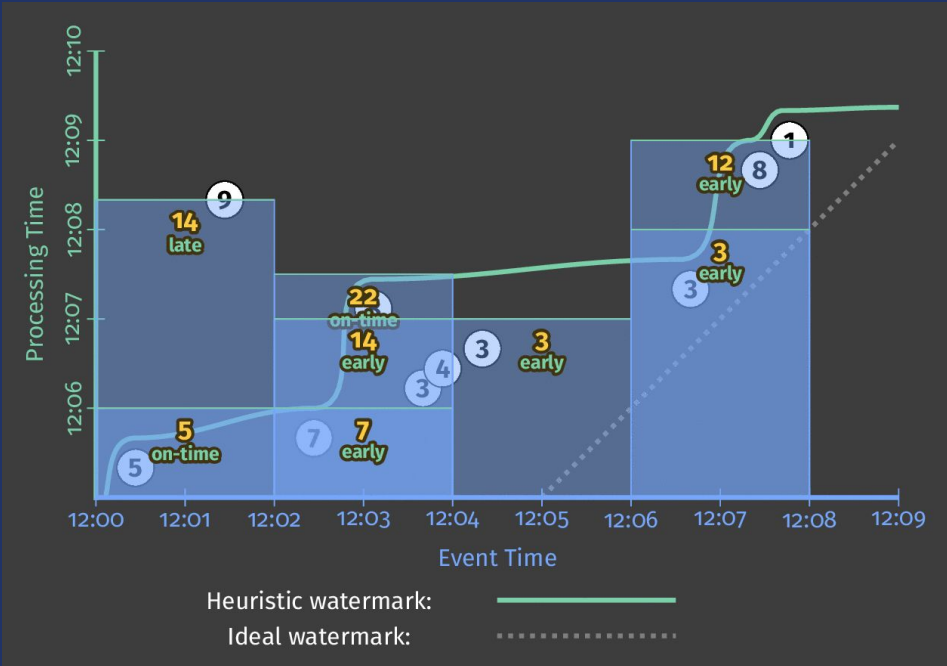
1. Classic Batch



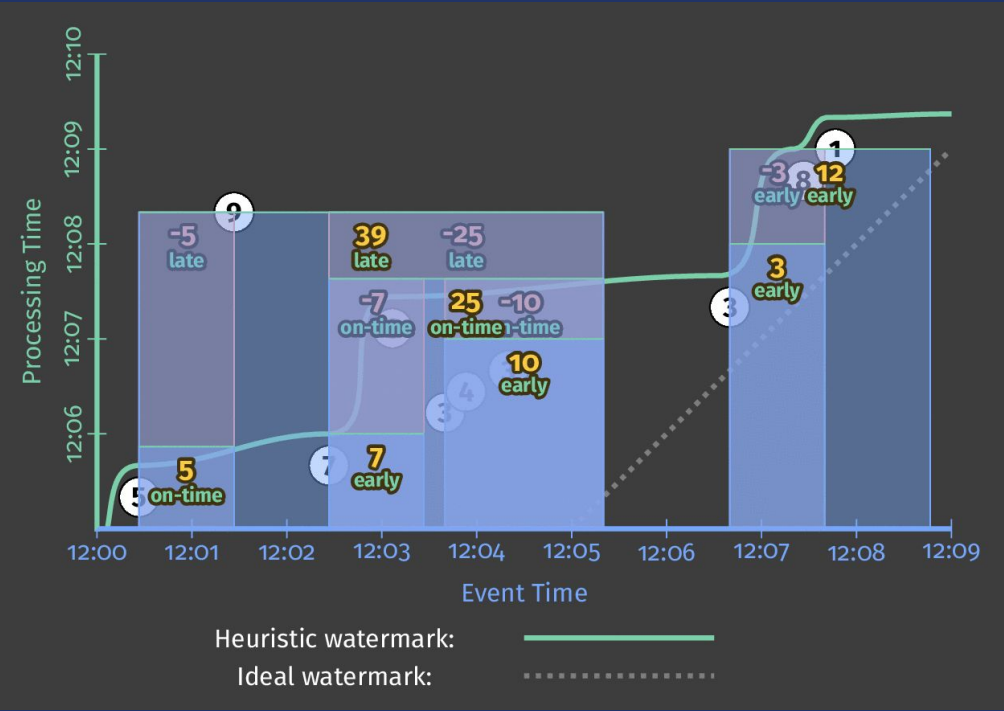
2. Batch with Fixed Windows



3. Streaming



4. Streaming with Speculative + Late Data



5. Streaming With Retractions

# Unit 03.

- 1. override windowing options, set accumulation strategy for late events**
- 2. drop events late more than 1 day**
- 3. accumulate late events iff batch has more than 3 elements**
- 4. wait for one hour for last event before closing the window**



# Learn More!

- [github.com/spotify/scio](https://github.com/spotify/scio)
- [beam.apache.org](https://beam.apache.org)

