Big Data Pipelines with Scala

Gleb Kanterov
@kanterov

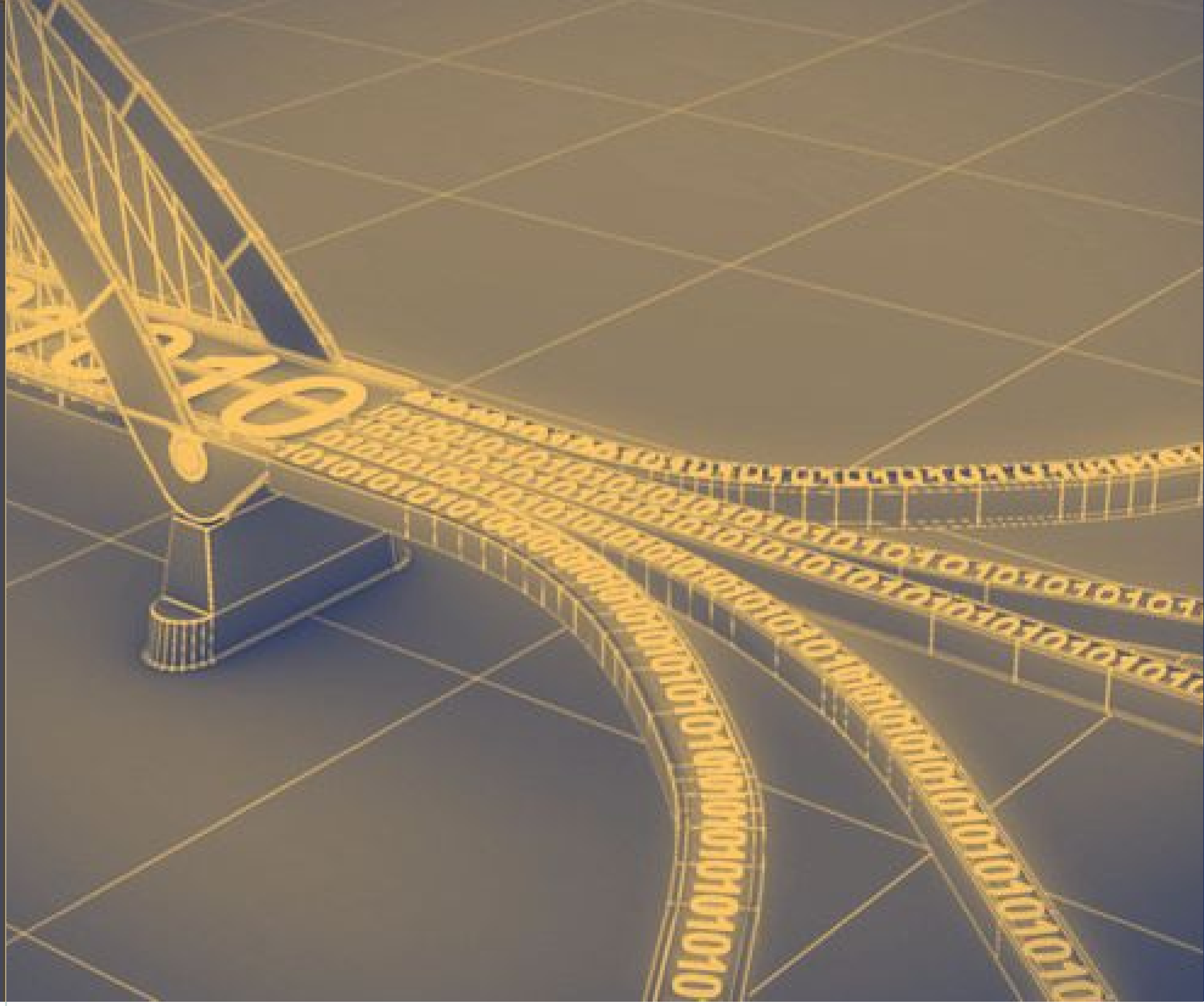
github.com/kanterov/
lambdaconf-2017-bigdata

Who am I?

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

We are hiring!

DATA



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

[1] https://press.spotify.com/us/about/

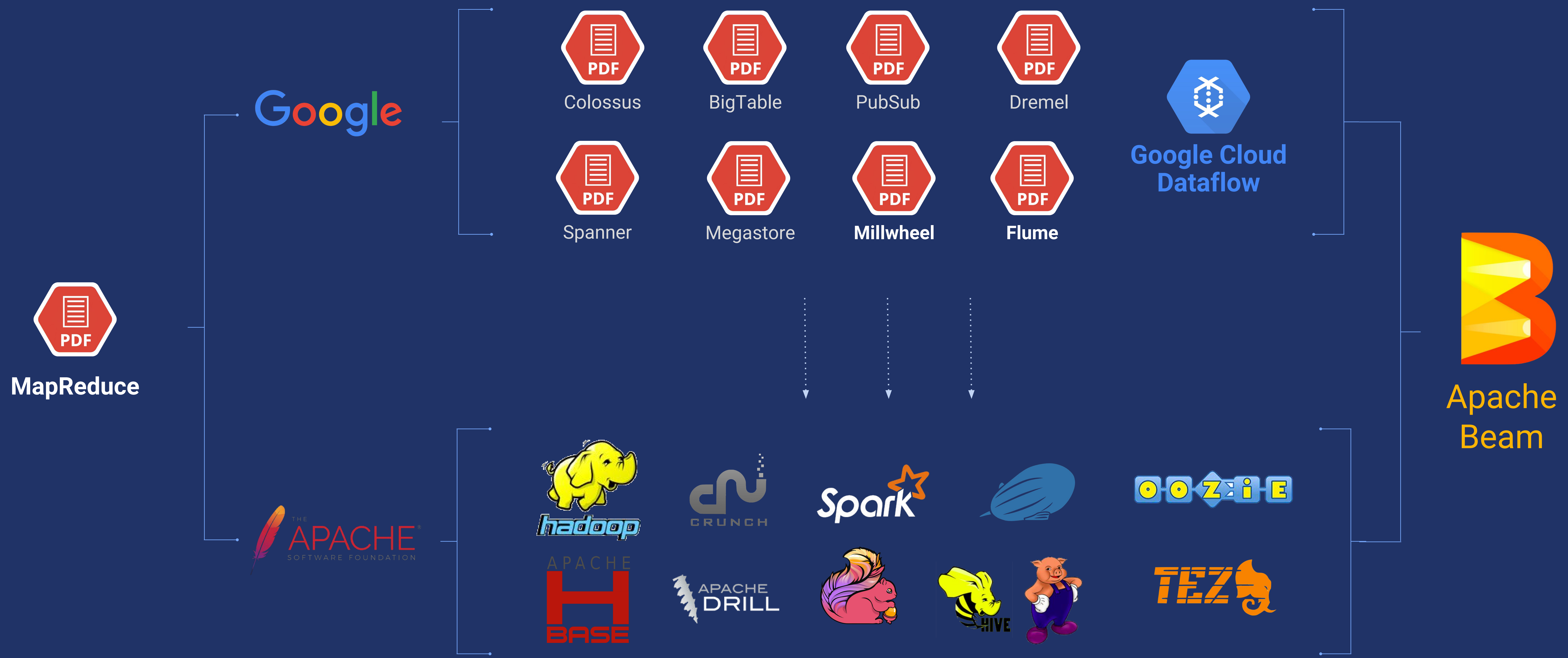
Hadoop@Spotify

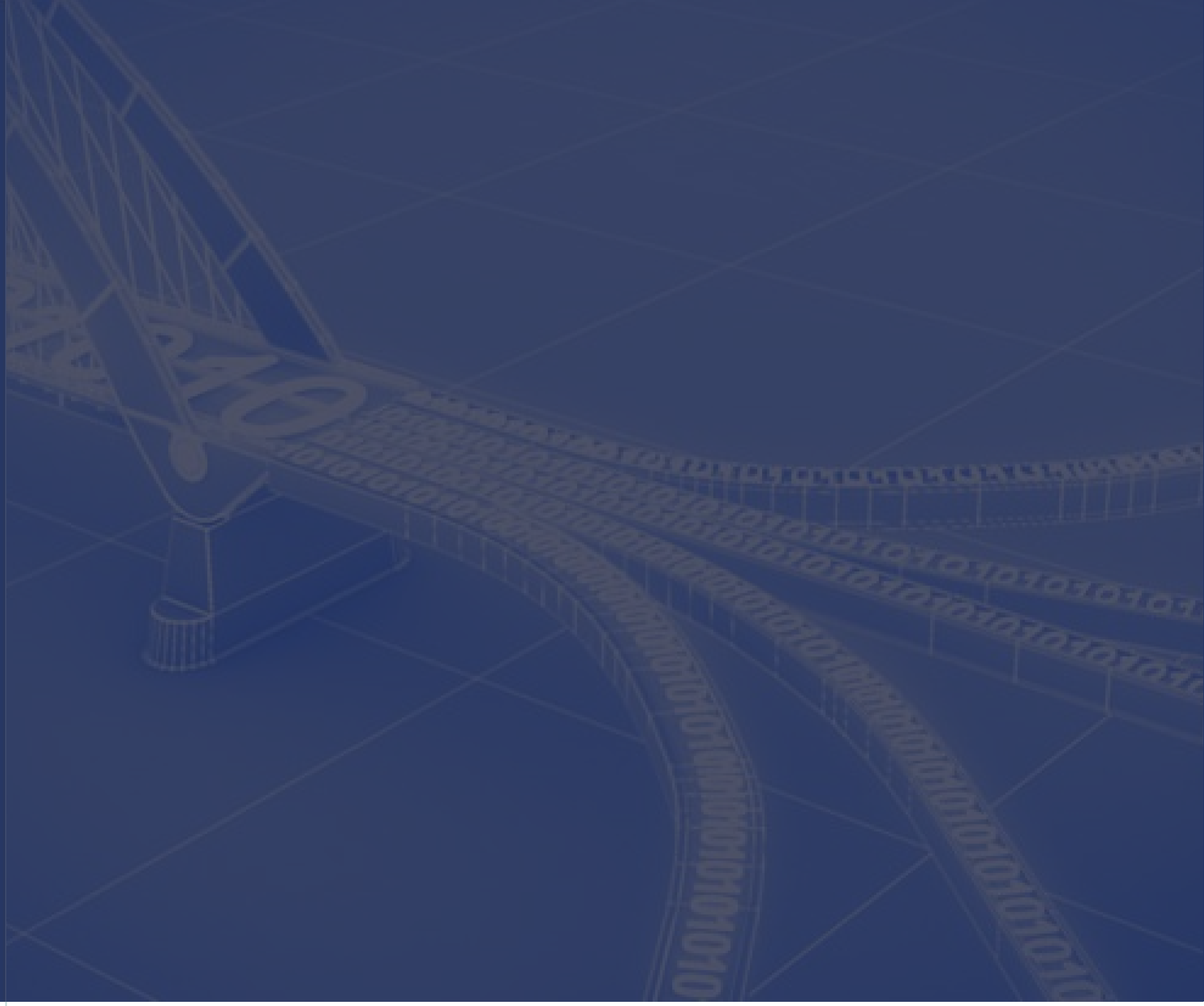
Hadoop@Spotify

- 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





Unit 01

Batch Pipelines and Big Data

Unit 02

Functional Programming and Big Data

Unit 03

From Batch to Streaming

MapReduce: Batch Processing

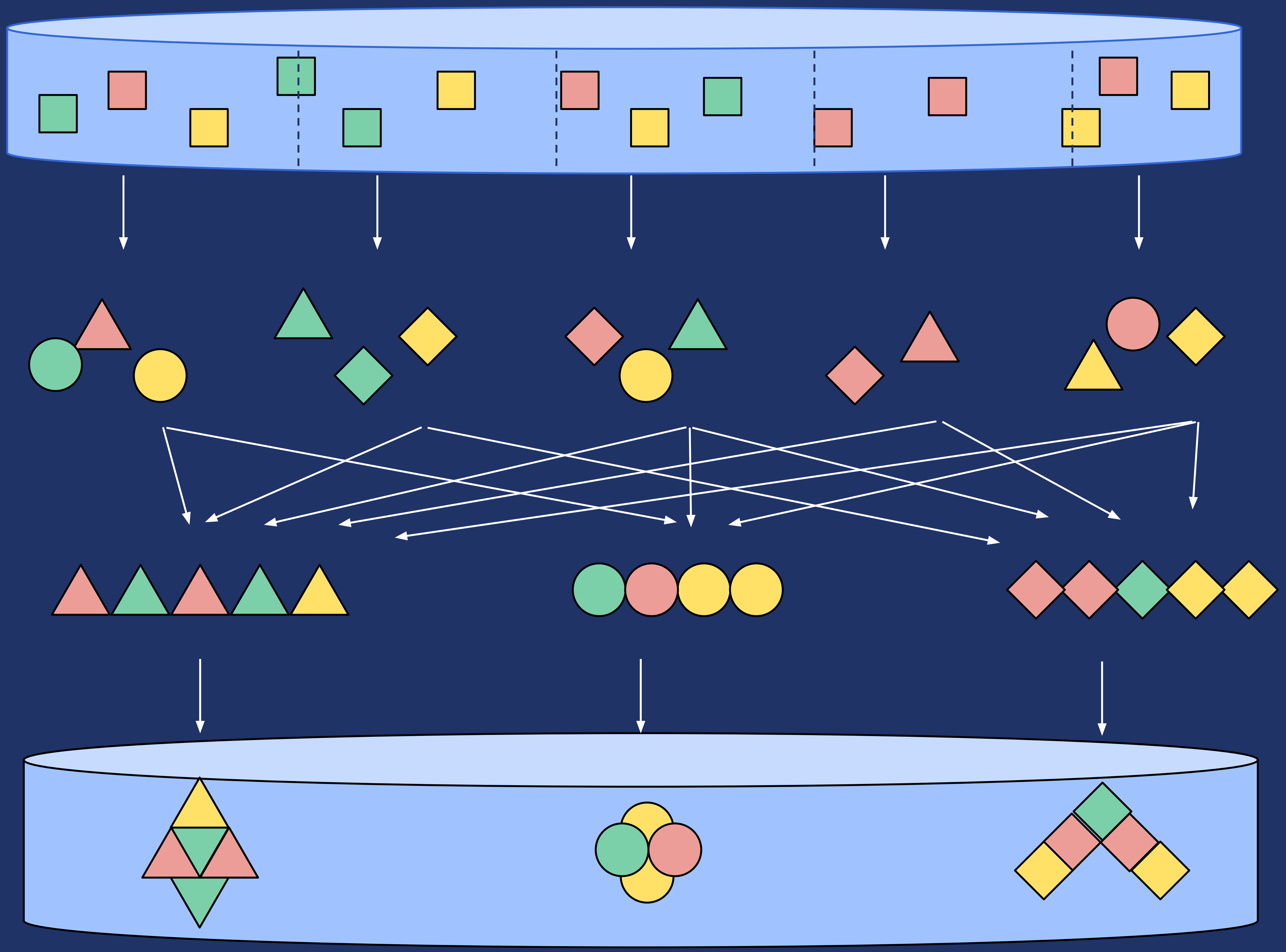
(Prepare)

Map

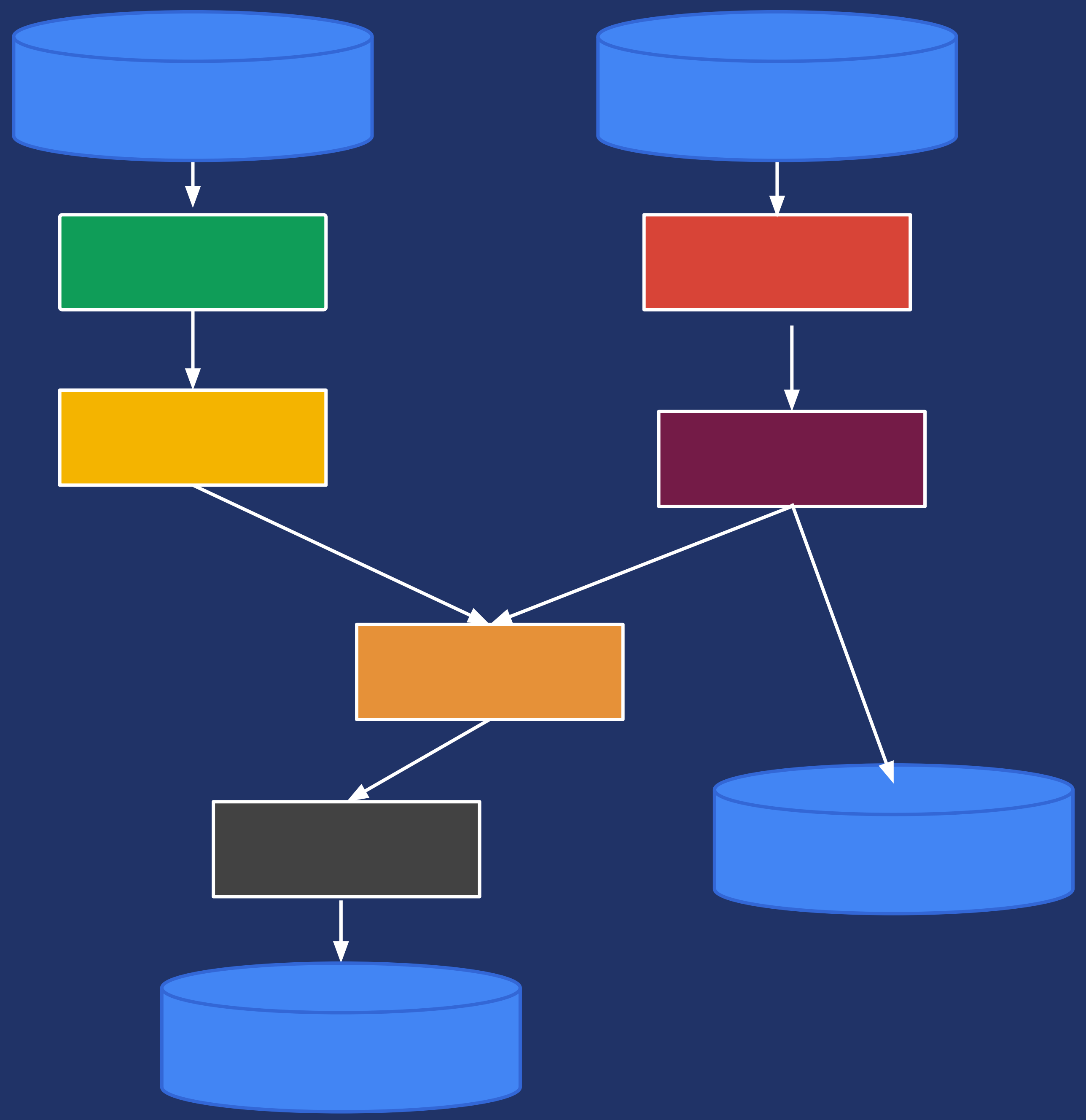
(Shuffle)

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 /ˈʃi.o/, [ˈʃiː.o]

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

```scala
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) }
```

```scala
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)]
}
```

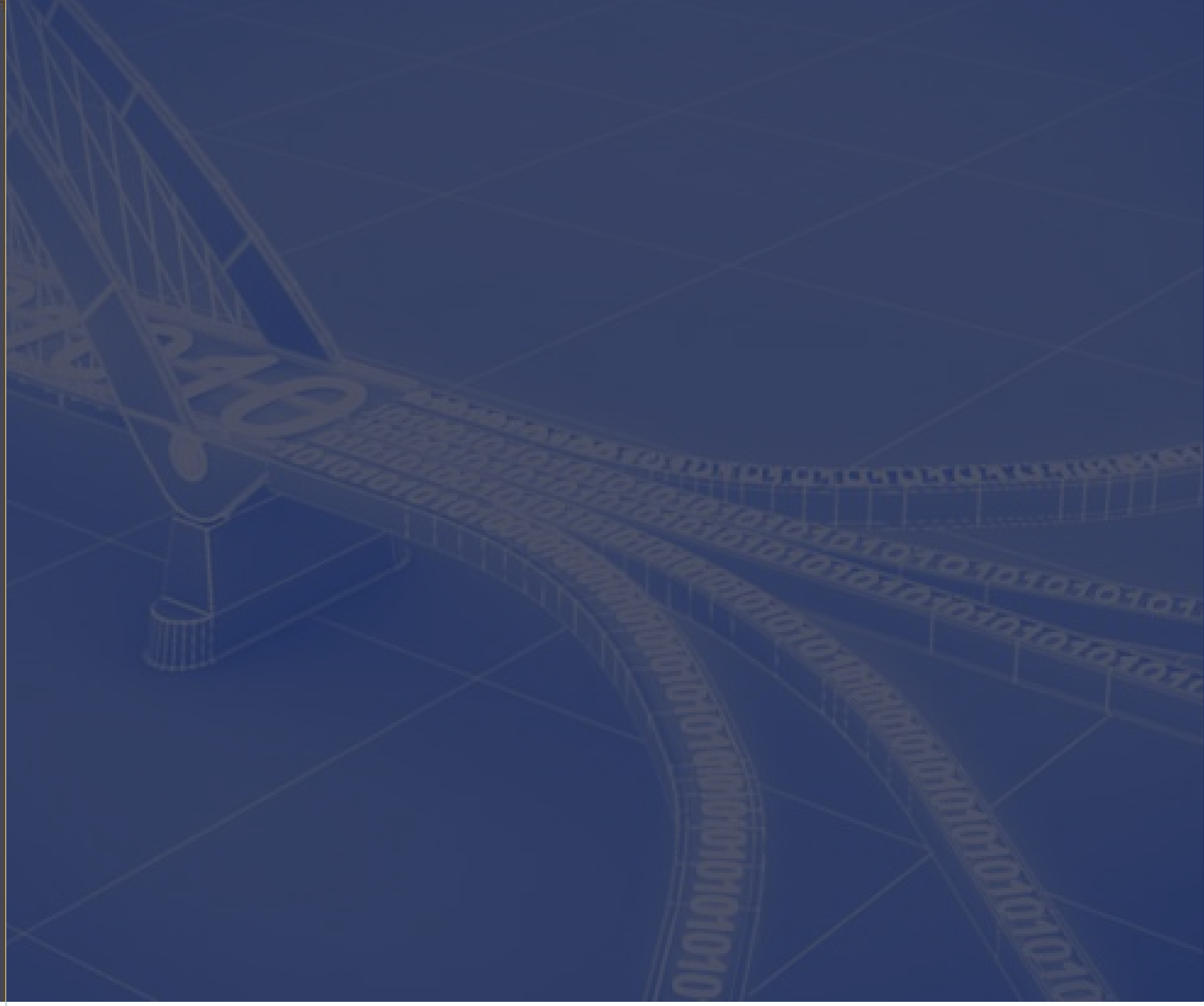
Unit 01.

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



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

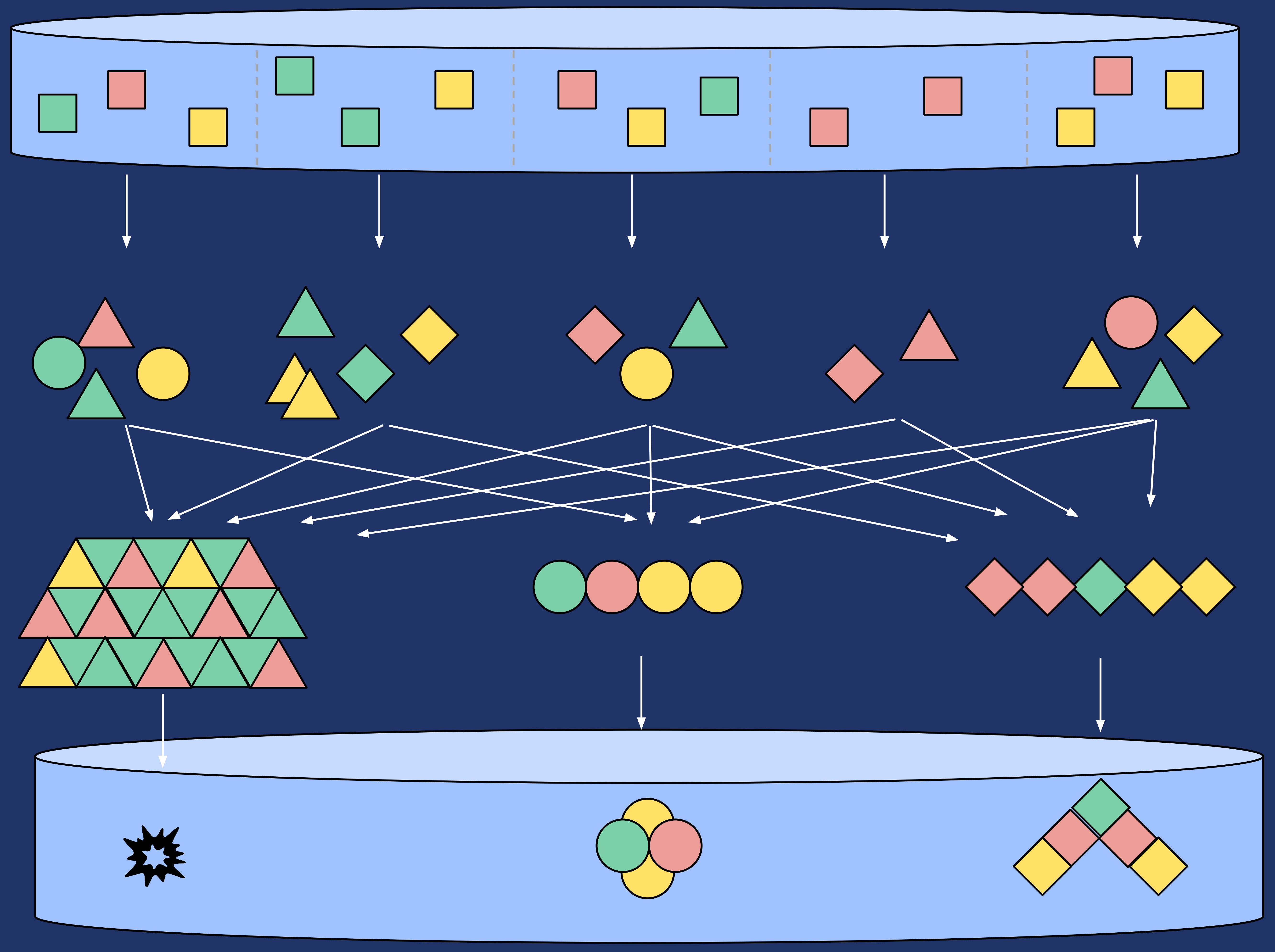
(Prepare)

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

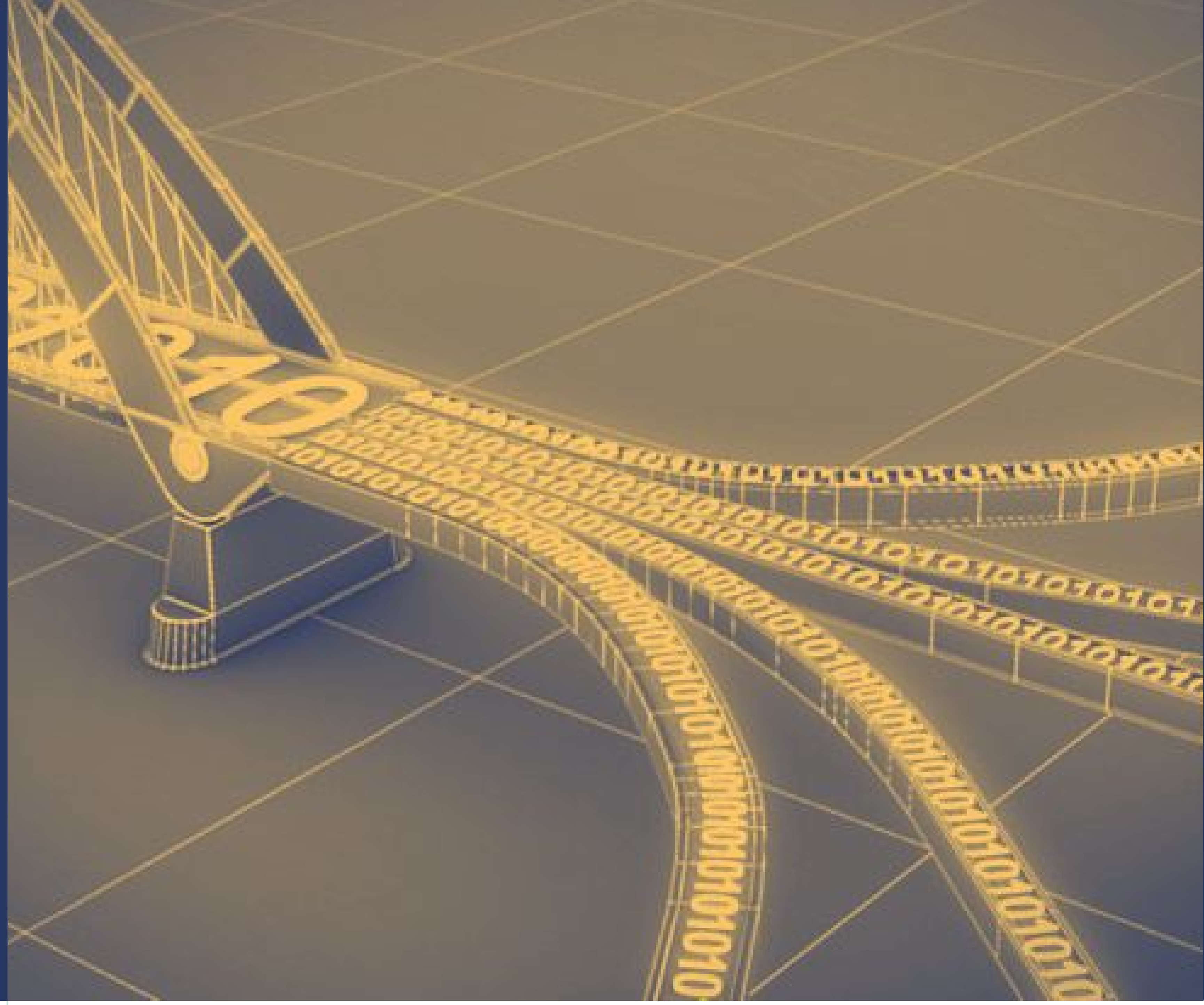
Skewed Join

```
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



Unit 01

Batch Pipelines and Big Data

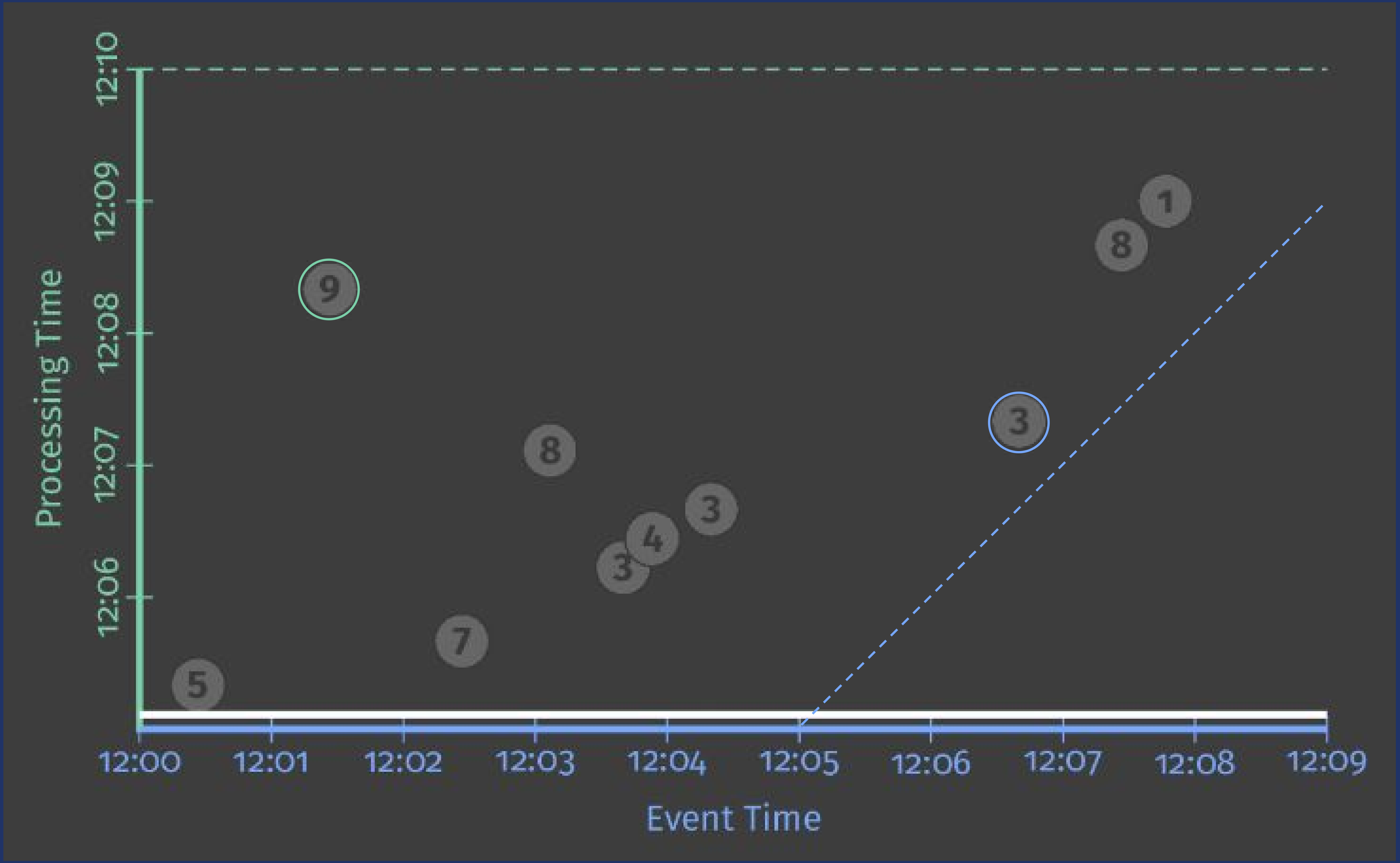
Unit 02

Functional Programming and Big Data

Unit 03

From Batch to Streaming

Event Time vs. Processing Time



What are you computing?

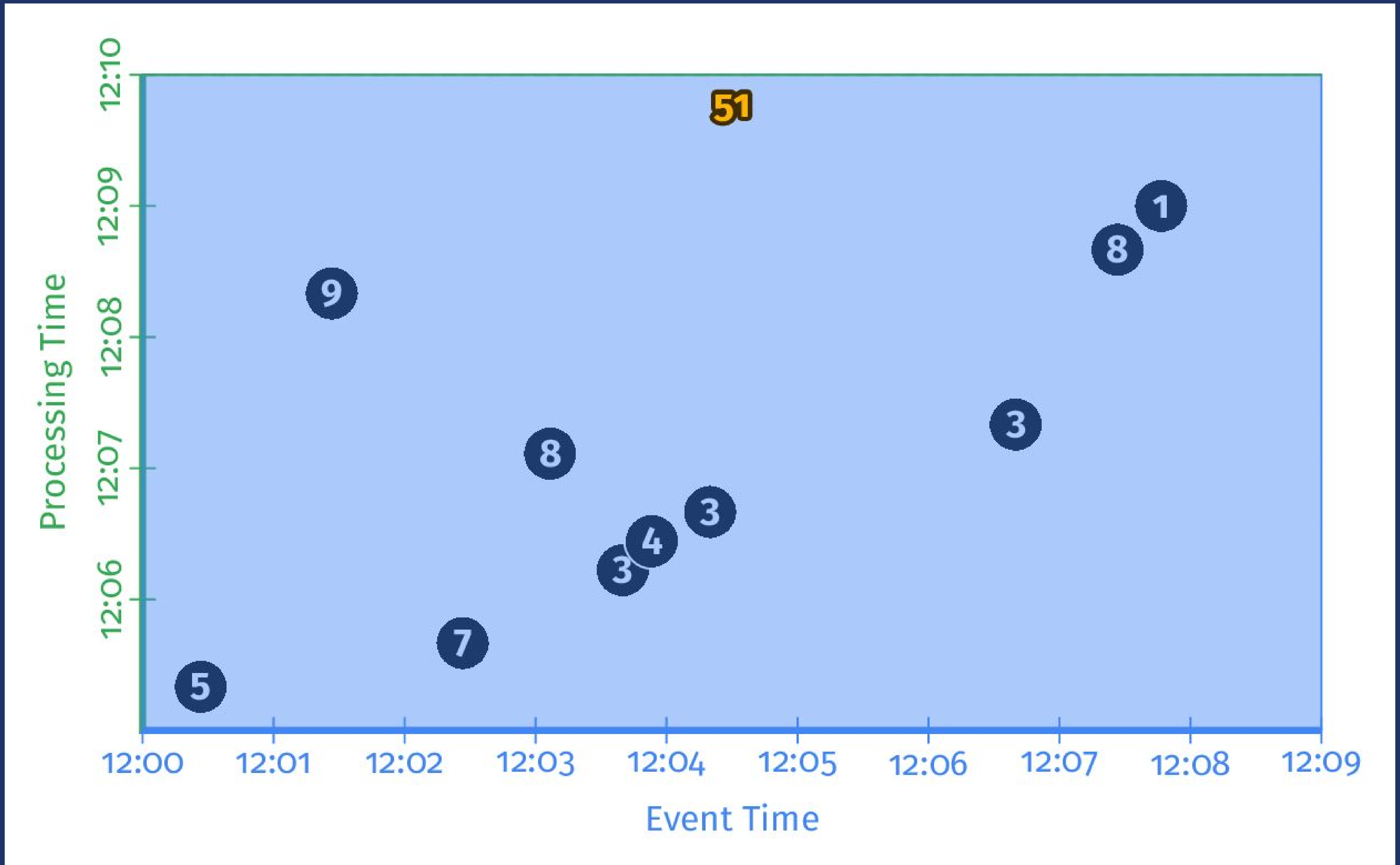
Where in event time?

When in processing time?

How do refinements relate?

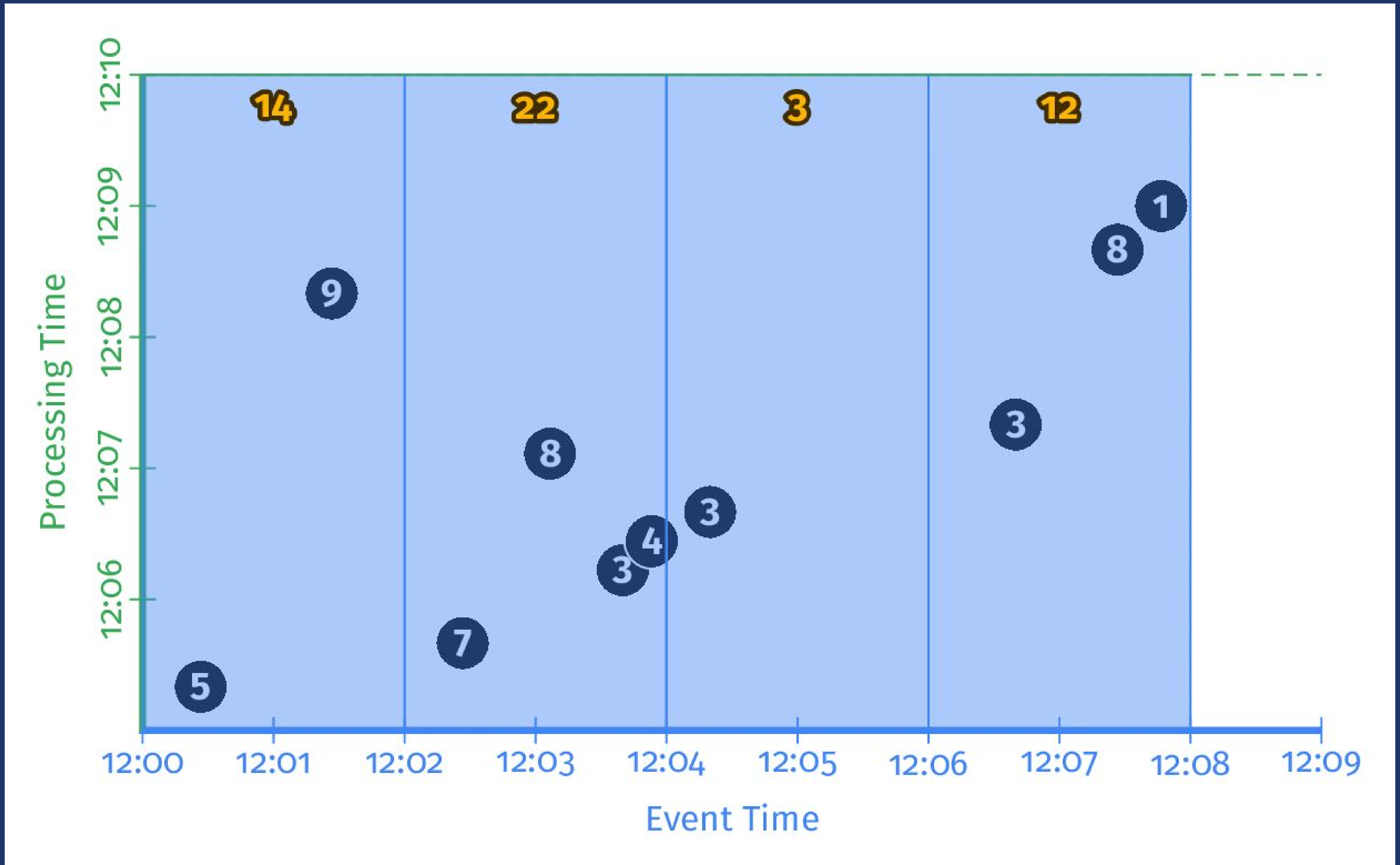
The Beam Model: What is Being Computed?

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



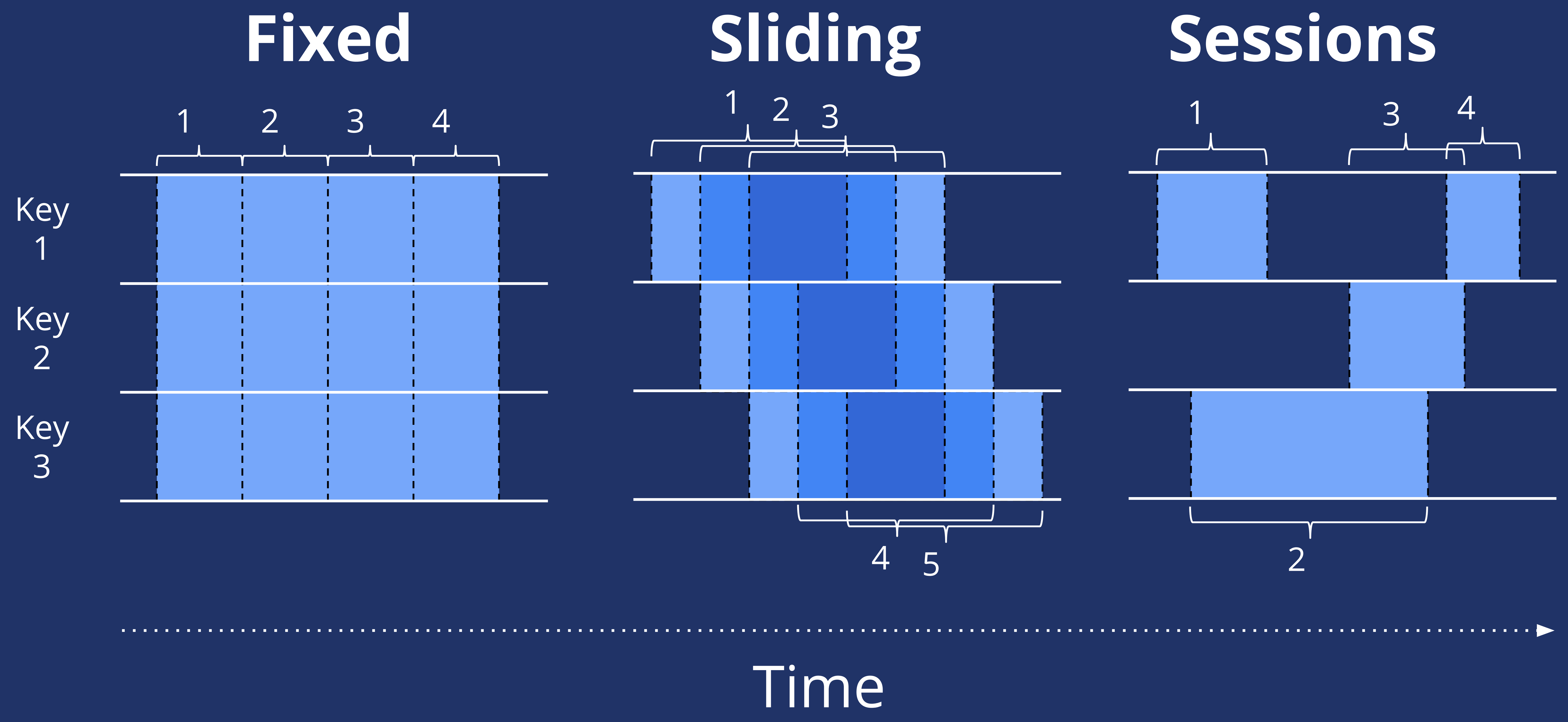
The Beam Model: Where in Event Time?

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



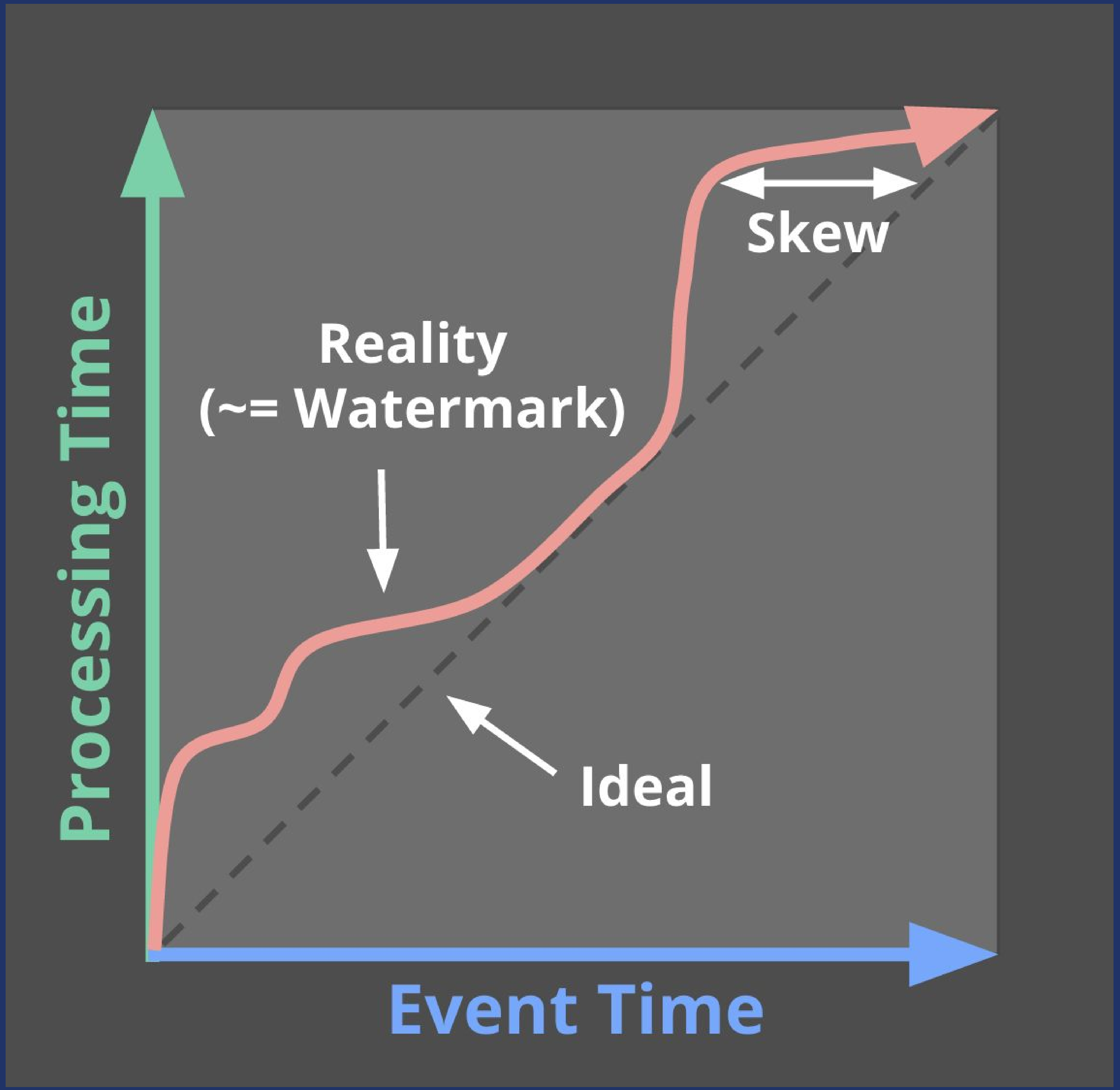
Where in event time?

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



Often required when doing aggregations over unbounded data.

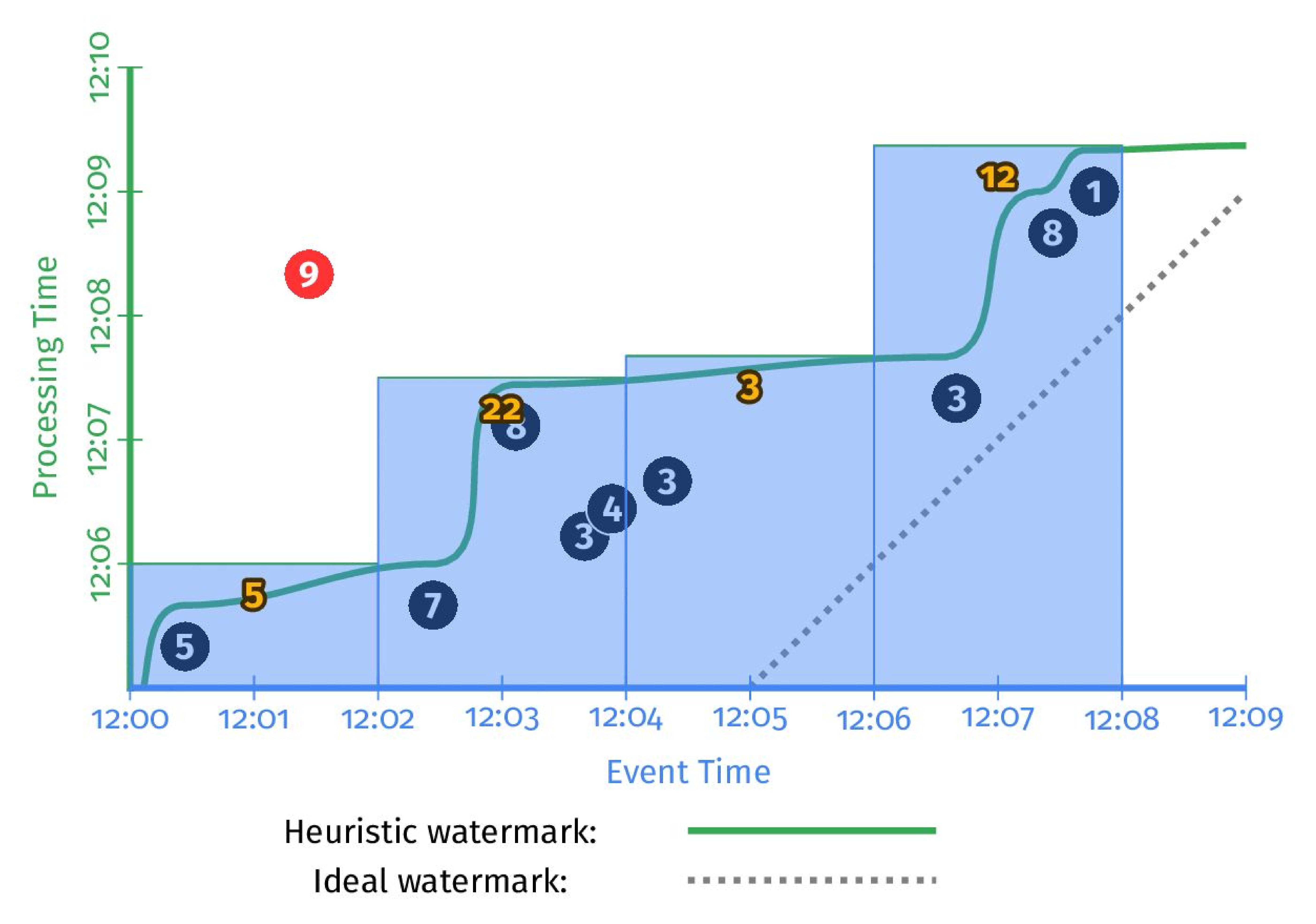
When in processing time?



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

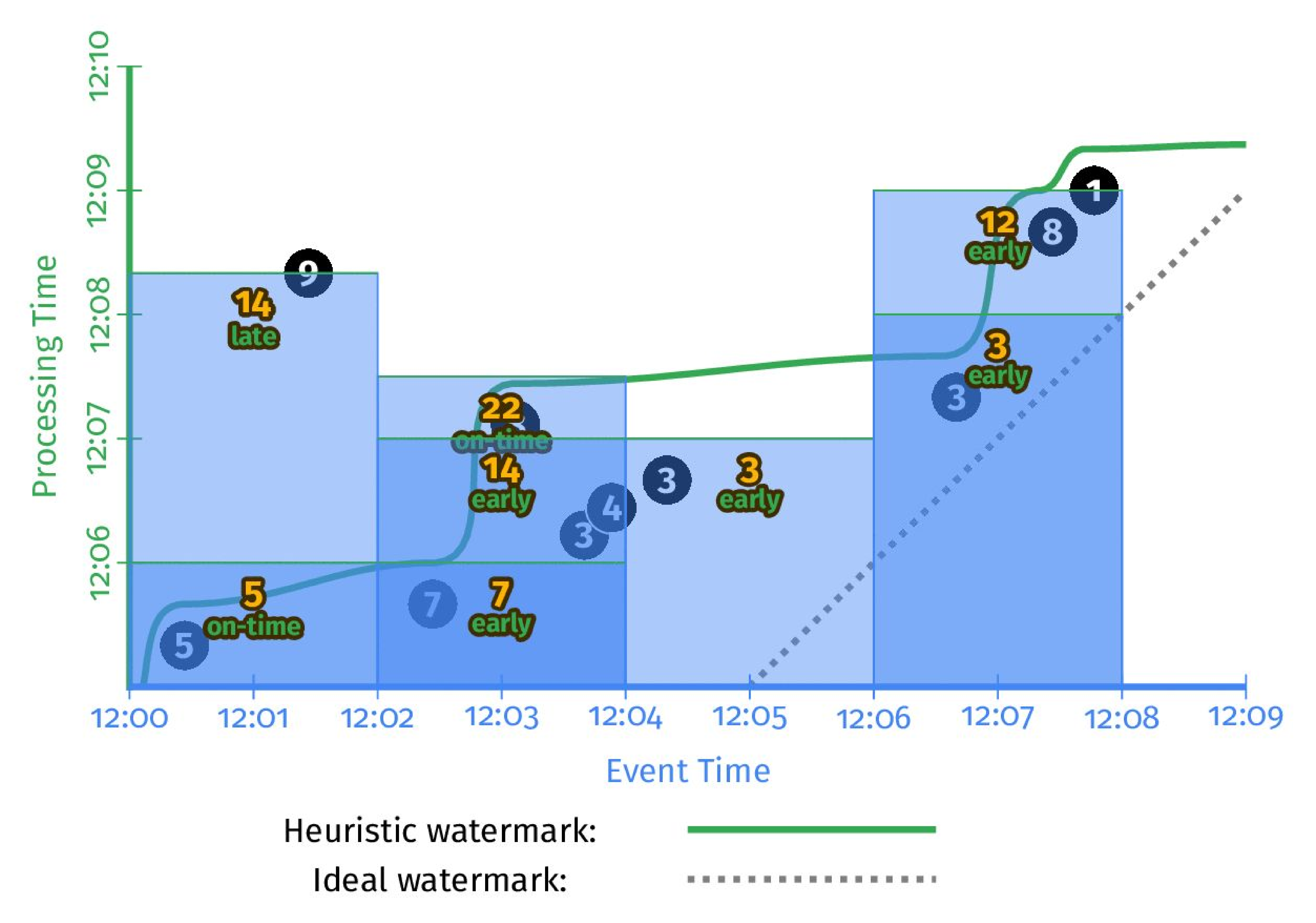
The Beam Model: When in Processing Time?

```
val trigger =
 AfterWatermark.pastEndOfWindow()

val options = WindowOptions(trigger)

playTracks
 .map(_.msPlayed)
 .withFixedWindows(
   duration = minutes(2),
   options = options)
 .sum
```

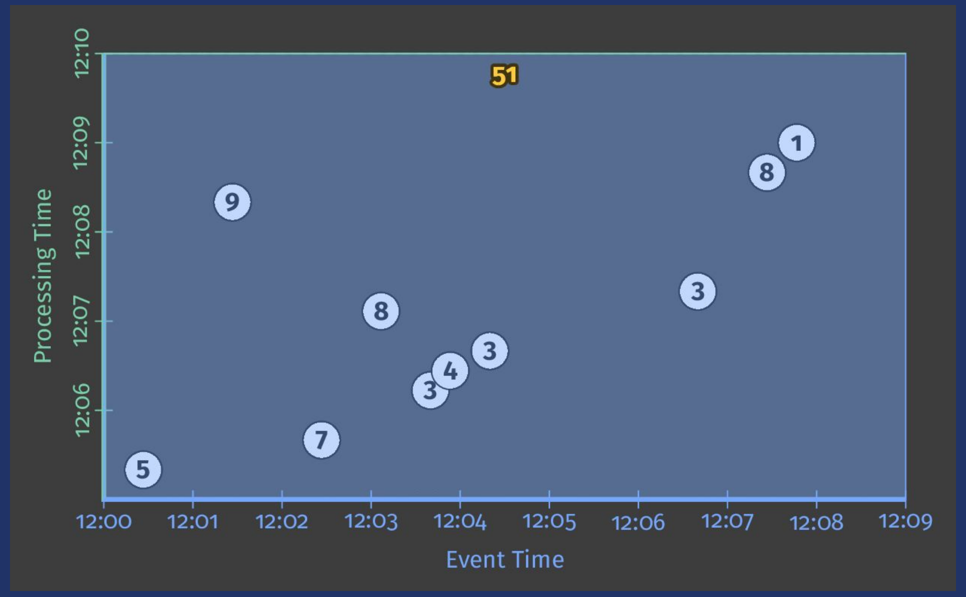


The Beam Model: How do Refinements Relate?

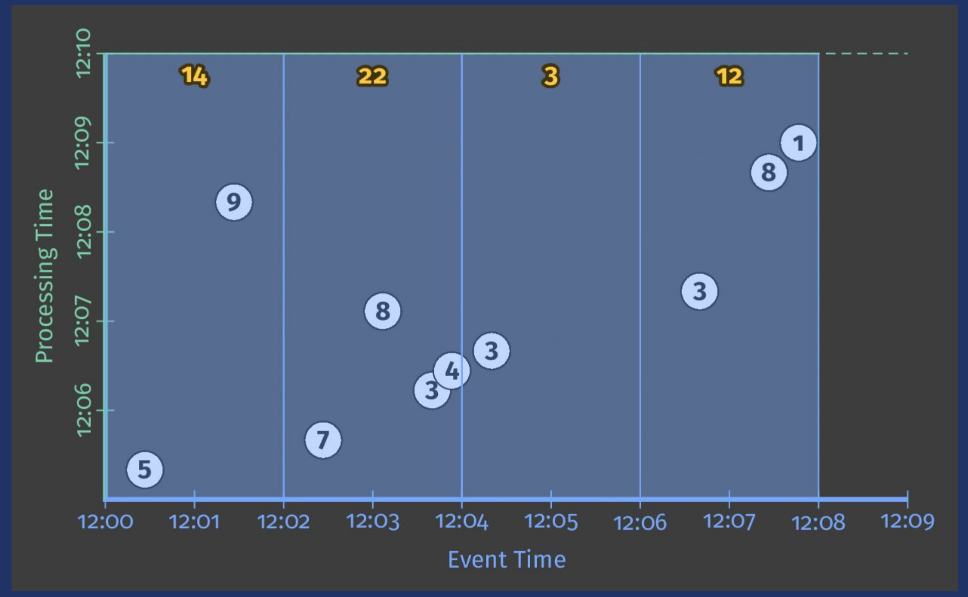
```
val trigger = AfterEach.inOrder(
 Repeatedly.forever(
   AfterProcessingTime
     .pastFirstElementInPane()
     .plusDelayOf(minutes(1))
 ).orFinally(
   AfterWatermark.pastEndOfWindow()),
 Repeatedly.forever(
   AfterPane
     .elementCountAtLeast(1)))

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



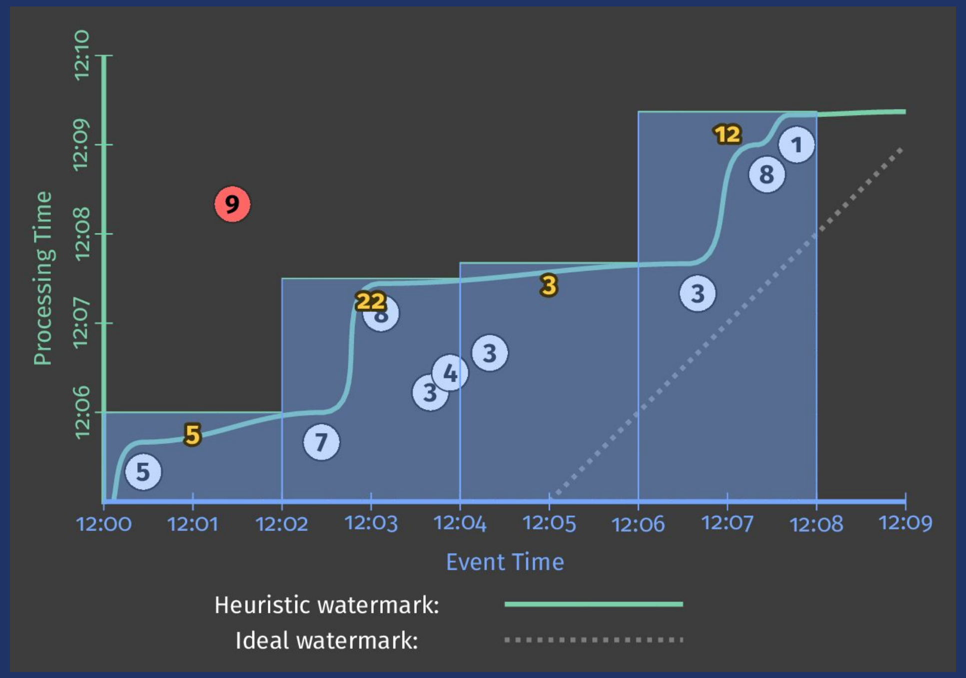
Customizing What When Where How



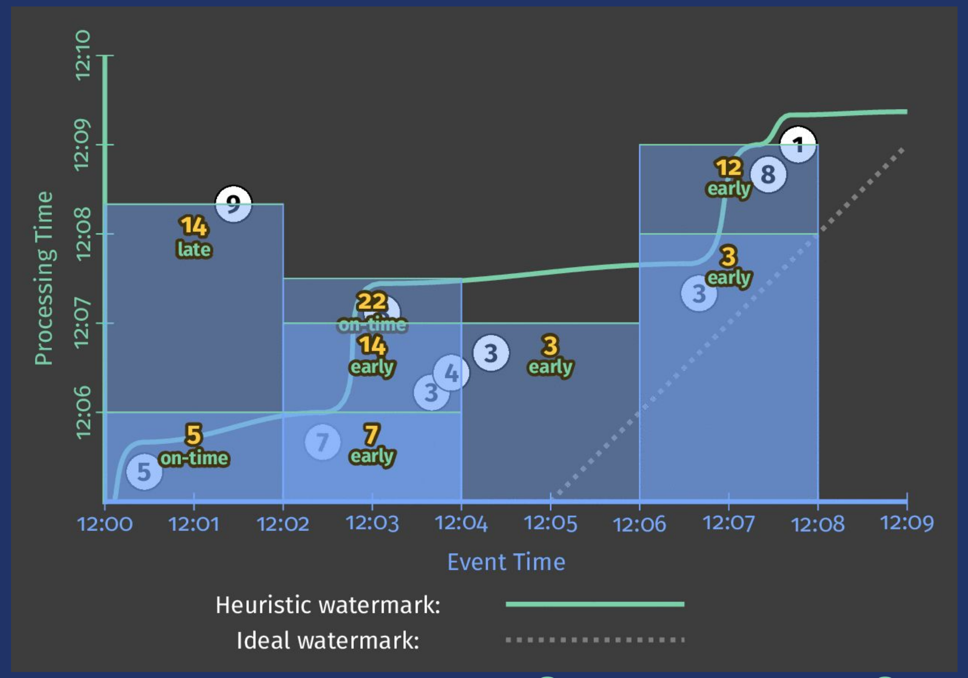
1.Classic Batch



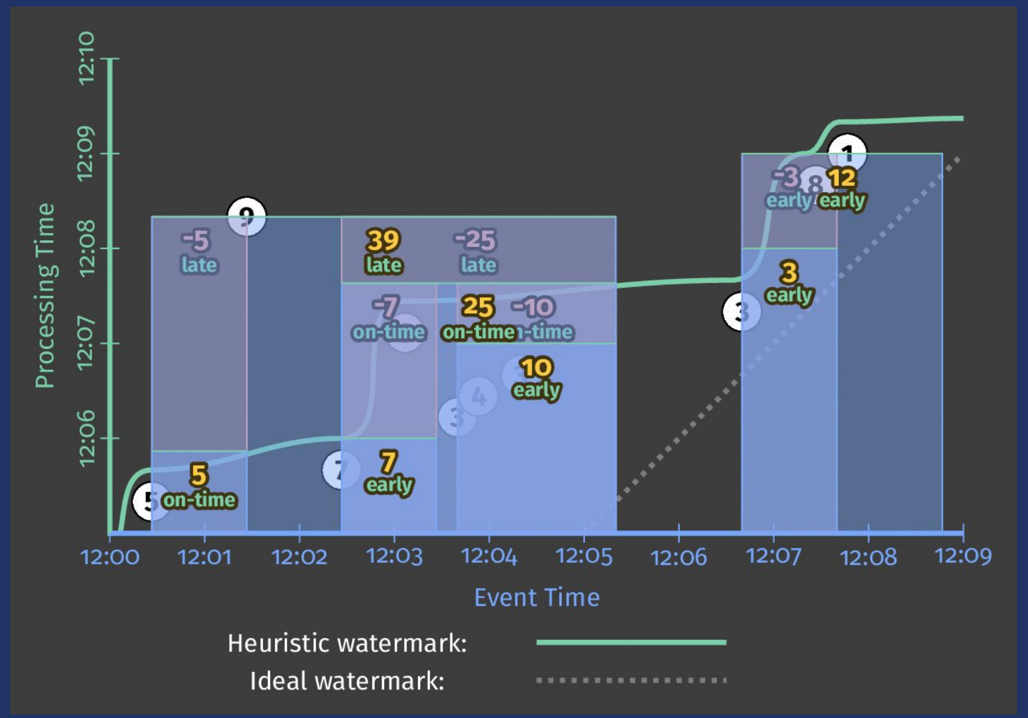
2. Batch with Fixed Windows



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
- beam.apache.org

Spotify®