Developing PulsarIO Connector

By Marco Robles

BEAM SUMMIT Austin, 2022

Agenda

- Introduction
- What is Pulsar?
- Initial approach
- Current implementation
- Example
- Next steps
- Q&A

Who am I?

Software Engineer
@ Wizeline

Who we are

Wizeline, a **global technology services provider**, builds high-quality digital products and platforms that accelerate time-to-market.

- We focus on **measurable outcomes**, partnering with our customers to modernize core technologies, mature data-driven capabilities, and improve user experience.
- Our **adaptive teams** provide the right combination of solutions, capabilities, and methodologies to deliver results, while partnering with our customers' teams to foster innovation through continuous learning.
- We are invested in **doing well while doing good**, striving to make a positive impact where we live and work. Our diverse culture of innovation, ownership, and community, combined with our **Academy**, creates an inspiring environment for talent to build long-term careers.

OTHERS PROMISE, WE DELIVER

Wizeline delivers seamless, scalable digital solutions, embedding the right technology, methodology, and mindsets within our customers' organizations. Our technology expertise and focus on AI & continuous learning, combined with our diverse and inclusive teams, allow us to deliver what you need right now, while also building a roadmap to your future.

20+

nationalities represented at Wizeline globally

2000+

Wizeline employees

Wizeline Team - Beam's Contributors

Benjamin Gonzalez

Mike Hernandez

Fernando Morales

Daniela Martin

Andoni Guzman

Elias Segundo

Lupita Amezcua

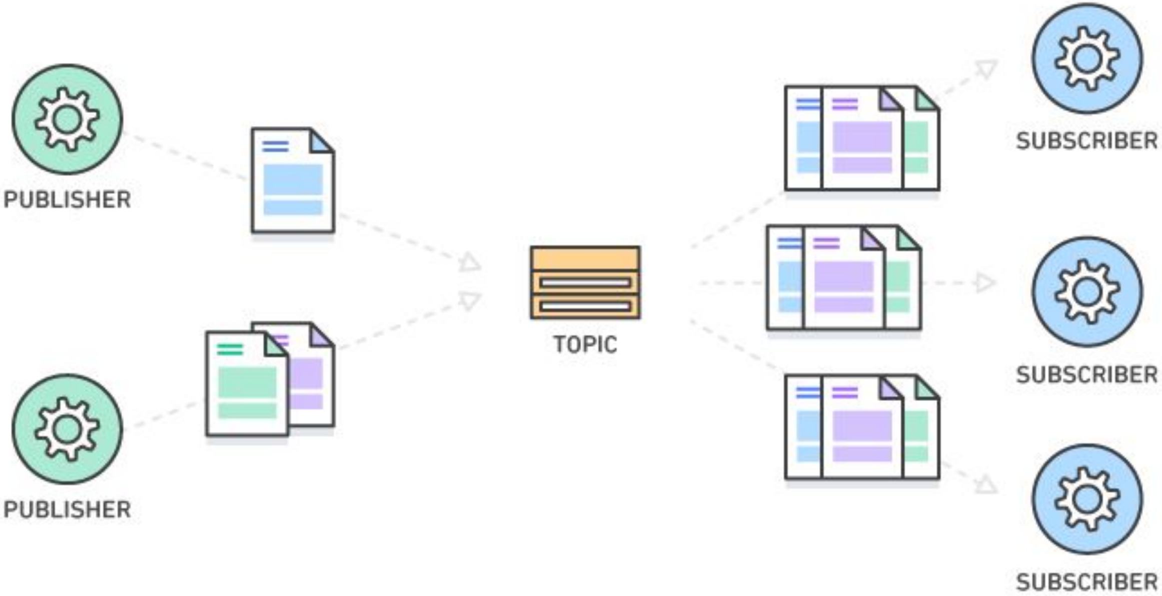
What is Pulsar?

- Introduction
- What is Pulsar?
- Initial approach
- Current implementation
- Example
- Next steps
- Q&A

BEAM SUMMIT

The basics

A pub/sub messaging system originally catered towards queuing use cases



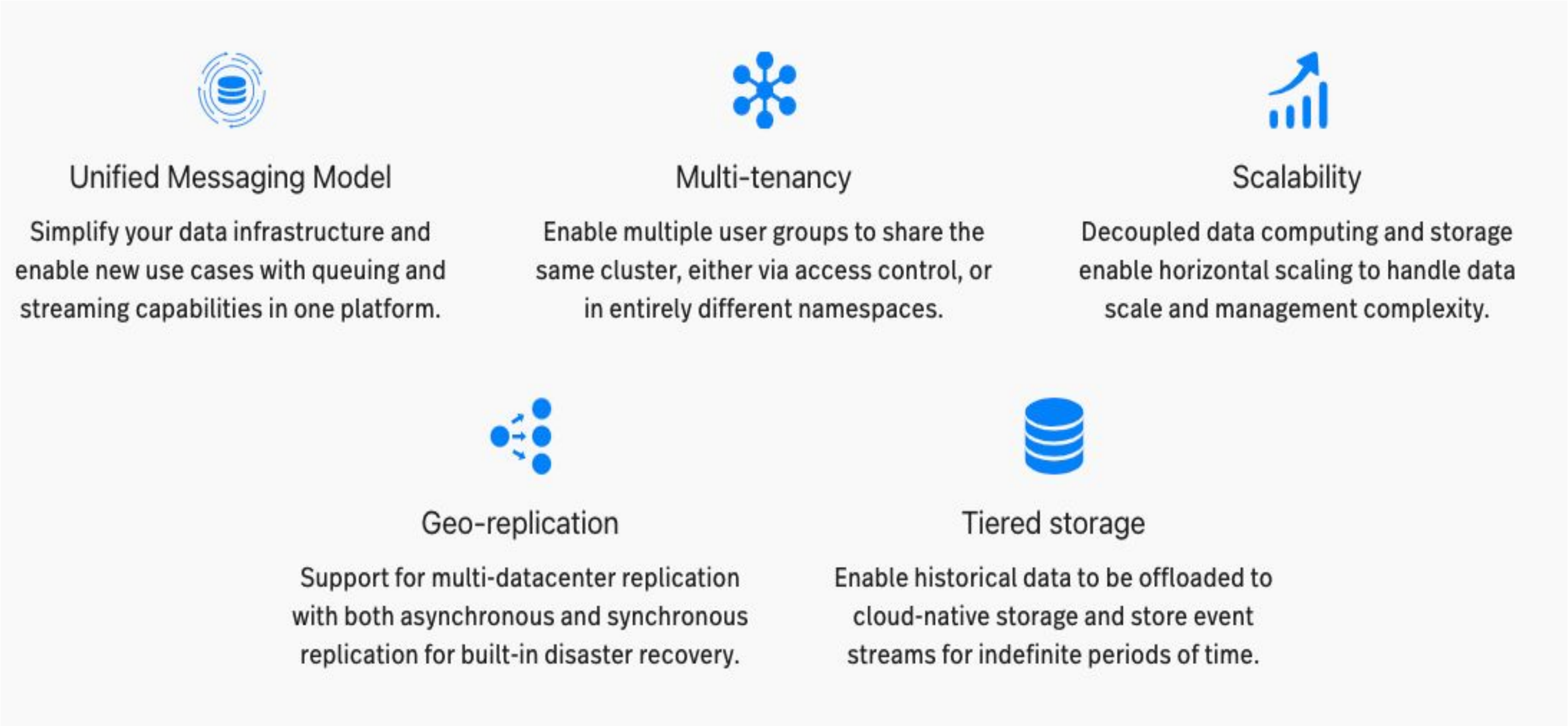
What is Apache Pulsar?

A distributed messaging and streaming platform originally created at Yahoo.

Pulsar is a multi-tenant, high-performance solution for server-to-server messaging.

PULSAR

Why Pulsar?



Unified Messaging Model

Simplify your data infrastructure and enable new use cases with queuing and streaming capabilities in one platform.

Multi-tenancy

Enable multiple user groups to share the same cluster, either via access control, or in entirely different namespaces.

Scalability

Decoupled data computing and storage enable horizontal scaling to handle data scale and management complexity.

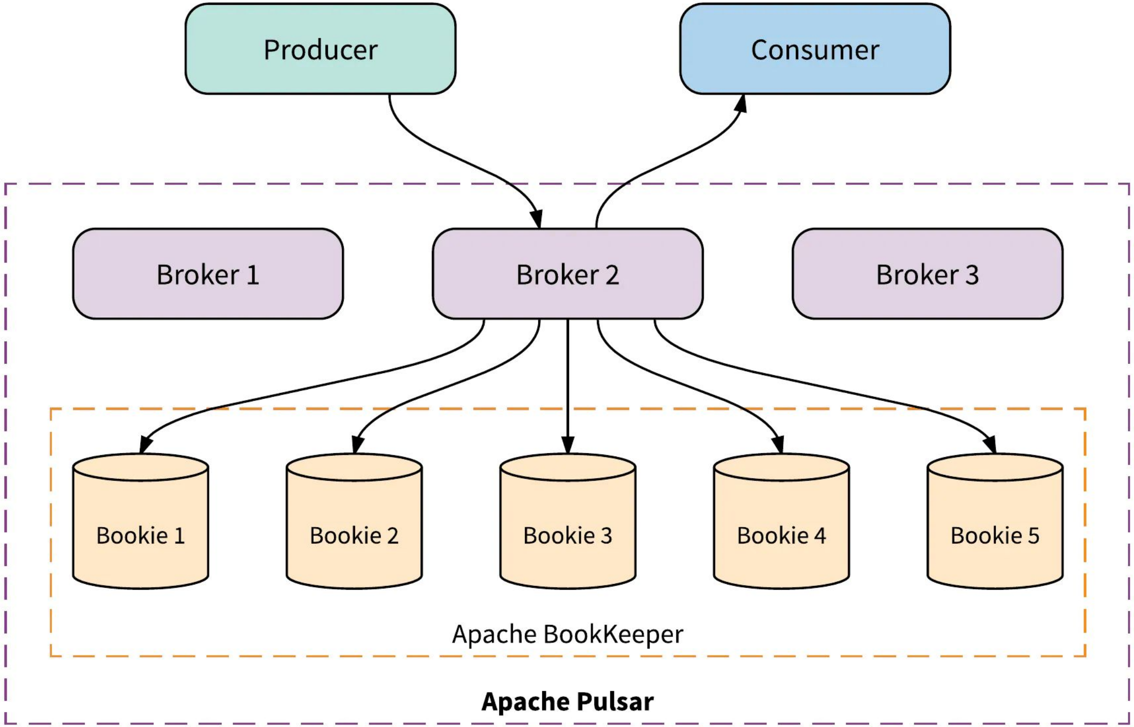
Geo-replication

Support for multi-datacenter replication with both asynchronous and synchronous replication for built-in disaster recovery.

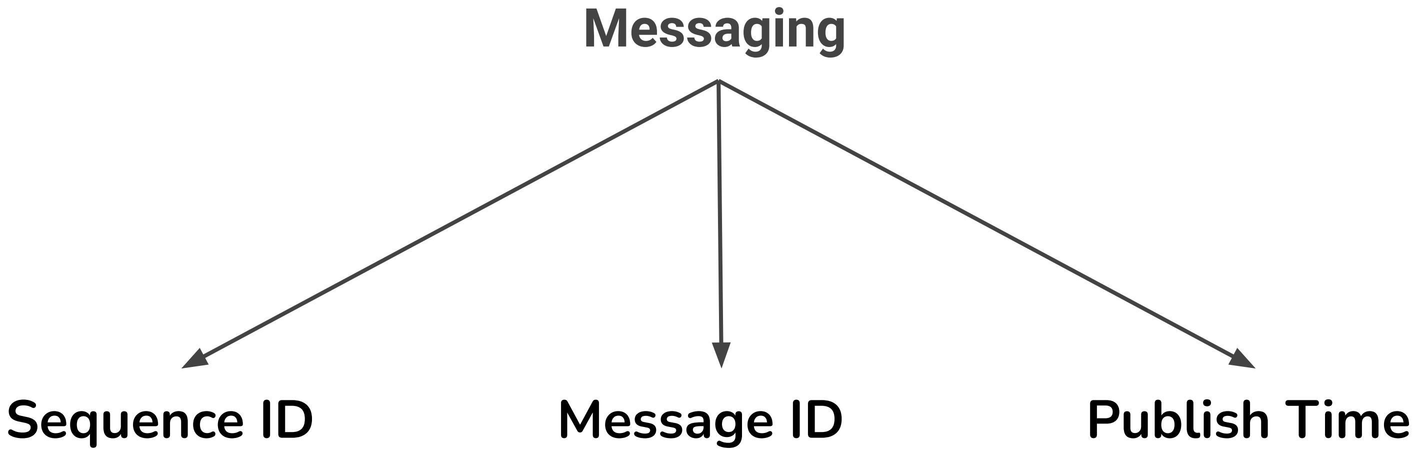
Tiered storage

Enable historical data to be offloaded to cloud-native storage and store event streams for indefinite periods of time.

Pulsar architecture



Pulsar messaging



Messaging

Sequence ID

Message ID

Publish Time

https://pulsar.apache.org/docs/concepts-messaging

Sequence ID

Each Pulsar message belongs to an ordered sequence on its topic.
Assigned by the producer (optional)

Constraints:

- **sequenceID >= 0**
- **sequenceID(N+1) > sequenceID(N)**
- It's **not necessary** for sequence IDs to be **consecutive.** There can be **holes** between messages.

Message ID

Indicates a message's specific position in a **ledger** and is unique within Pulsar cluster.

Constraints:

- It is **not** a **numeric value.**
- It has its **own value type** (Message ID class).

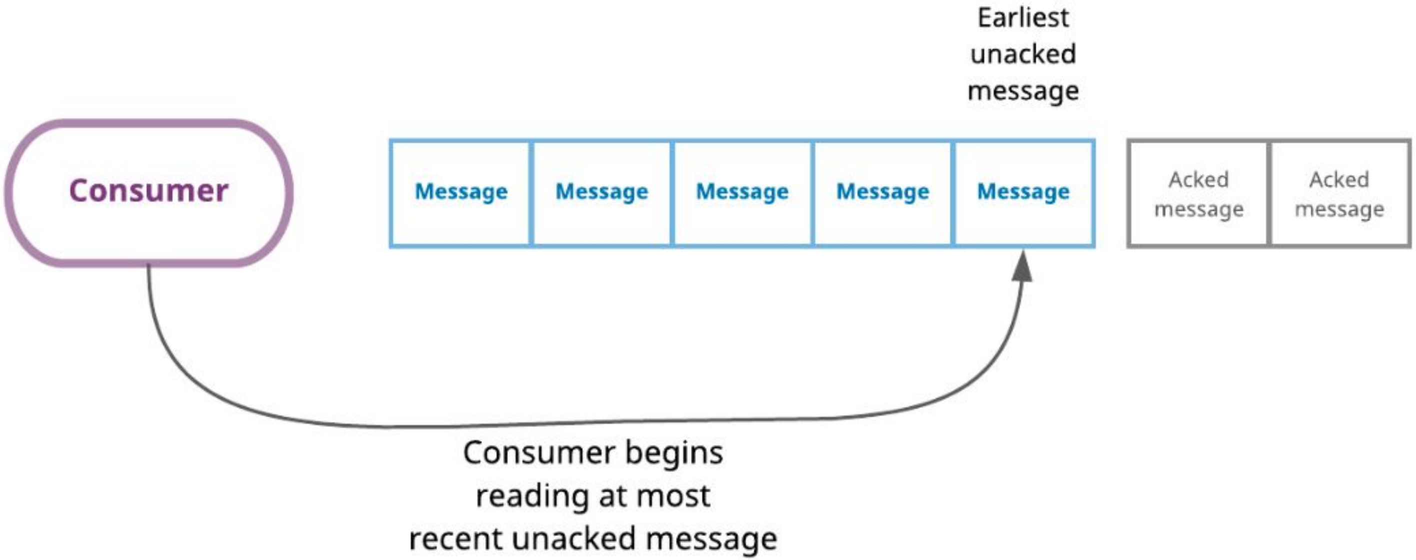
Publish time

The **timestamp** of when the message is **published**.

Automatically applied by the producer.

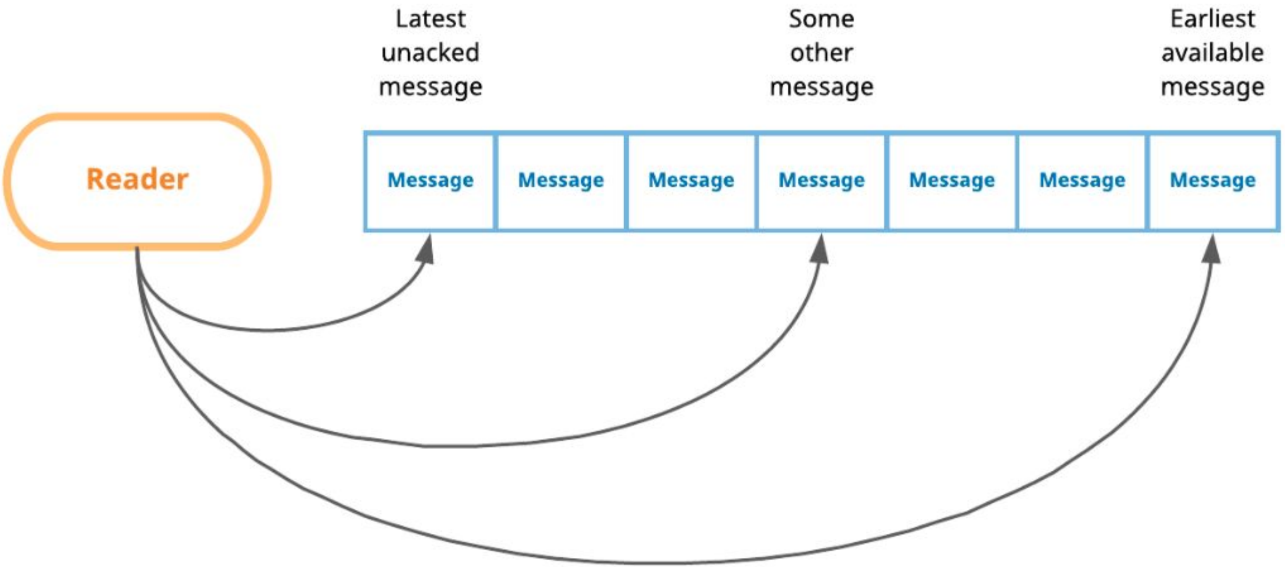
Consumer interface

Pulsar automatically manages topic cursors



Reader interface

Applications manually control topic cursors



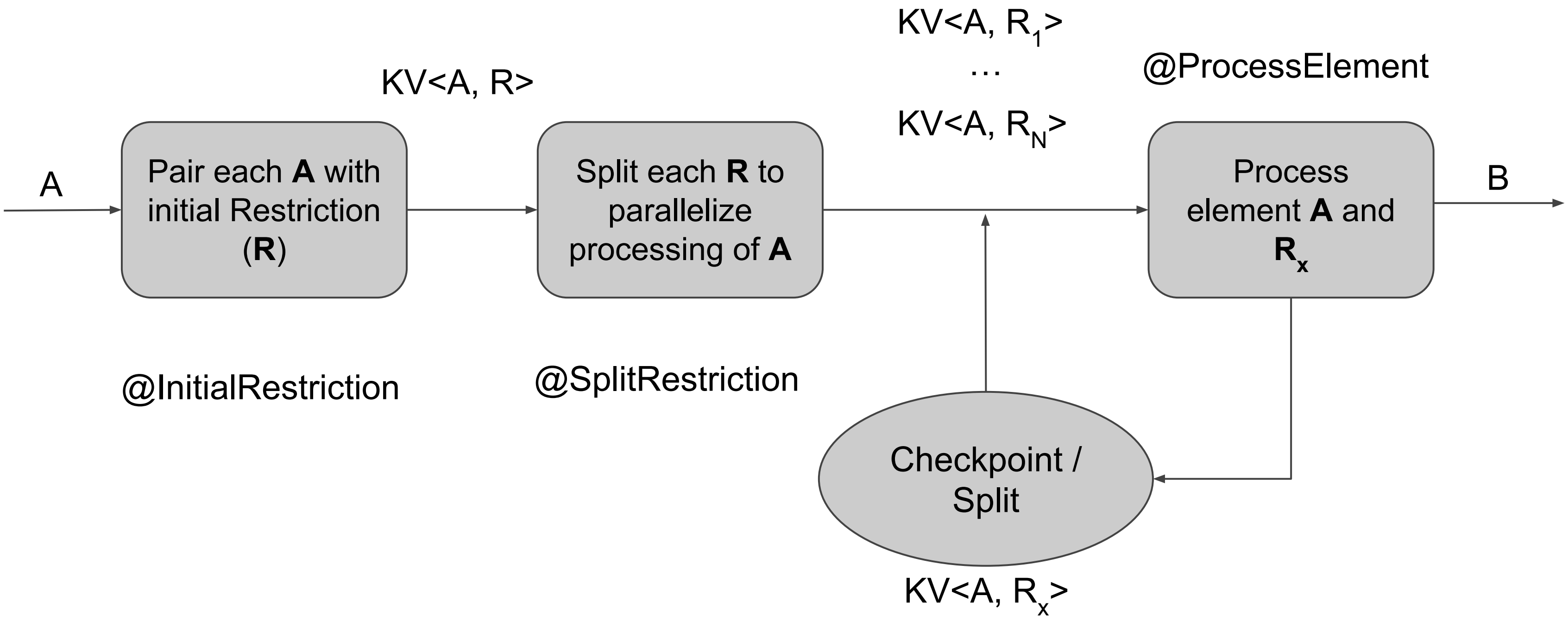
Reader begins reading at an application-specified message

Initial approach

- Presentation
- What is Pulsar?
- **Initial approach**
- Current implementation
- Example
- Next steps
- Q&A

Approach

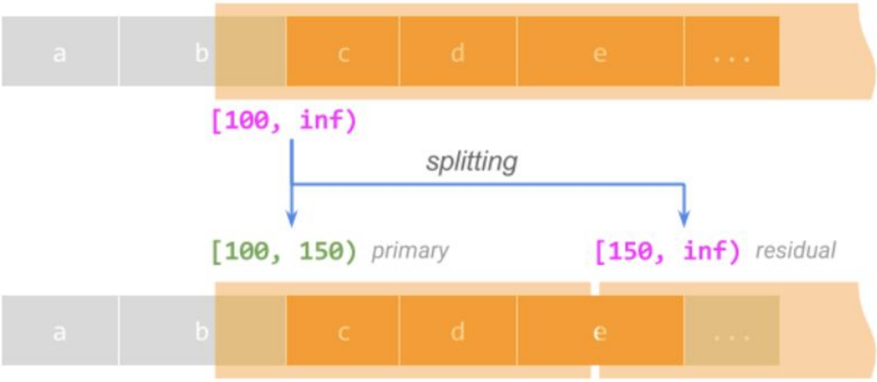
A basic **splittable DoFn (SDF)** implementation:


A
Pair each **A** with initial Restriction (**R**)
@InitialRestriction
KV<A, R>
Split each **R** to parallelize processing of **A**
@SplitRestriction
KV<A, R_1>
…
KV<A, R_N>
@ProcessElement
Process element **A** and **R_x**
B
Checkpoint / Split
KV<A, R_x>

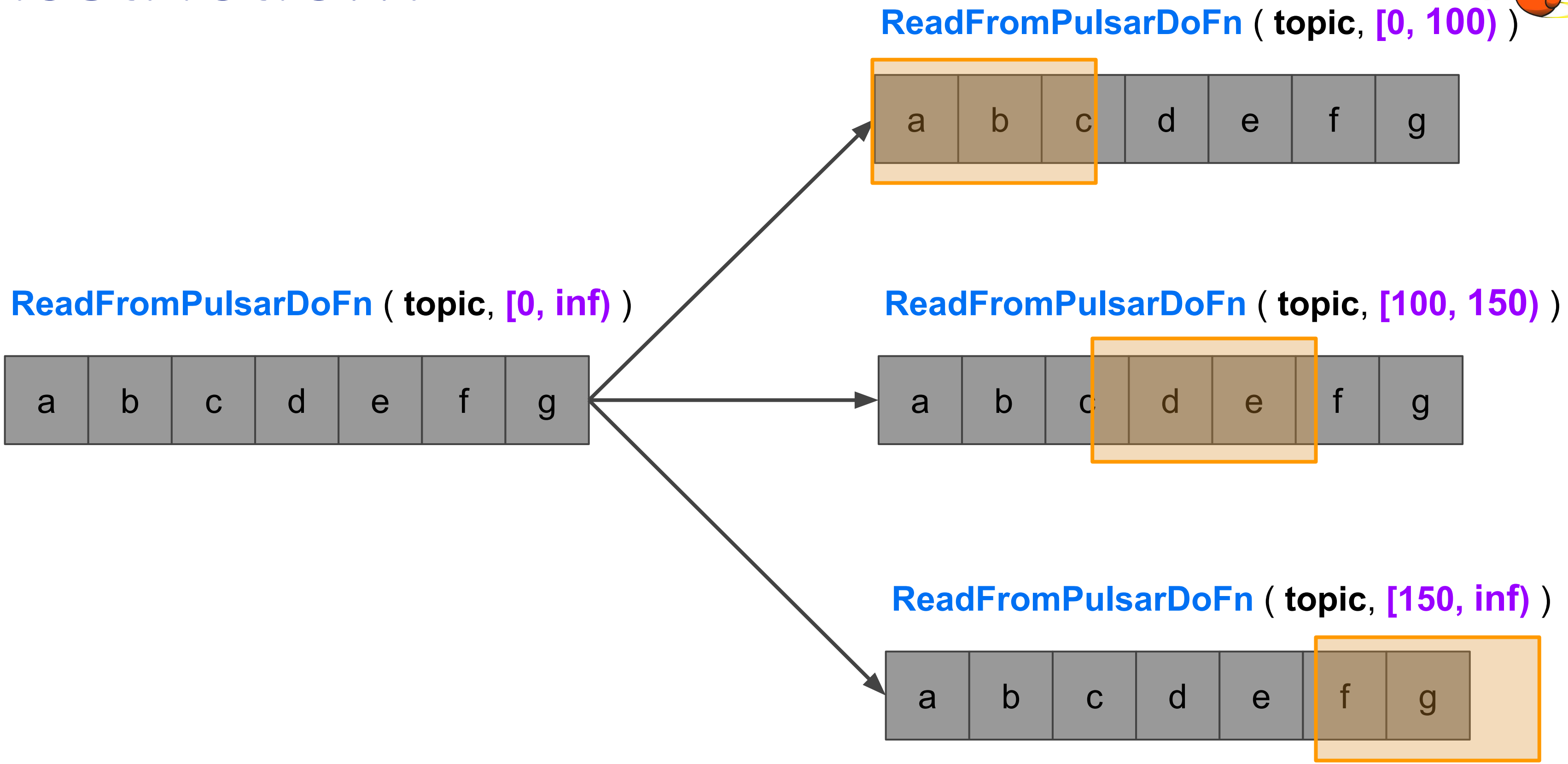
Which restriction can we use?

(element, restriction) -> (element, restriction$_1$) + (element, restriction$_2$)

```
ReadKafkaFn( some-topic, [100, inf)  )
```



Restriction?



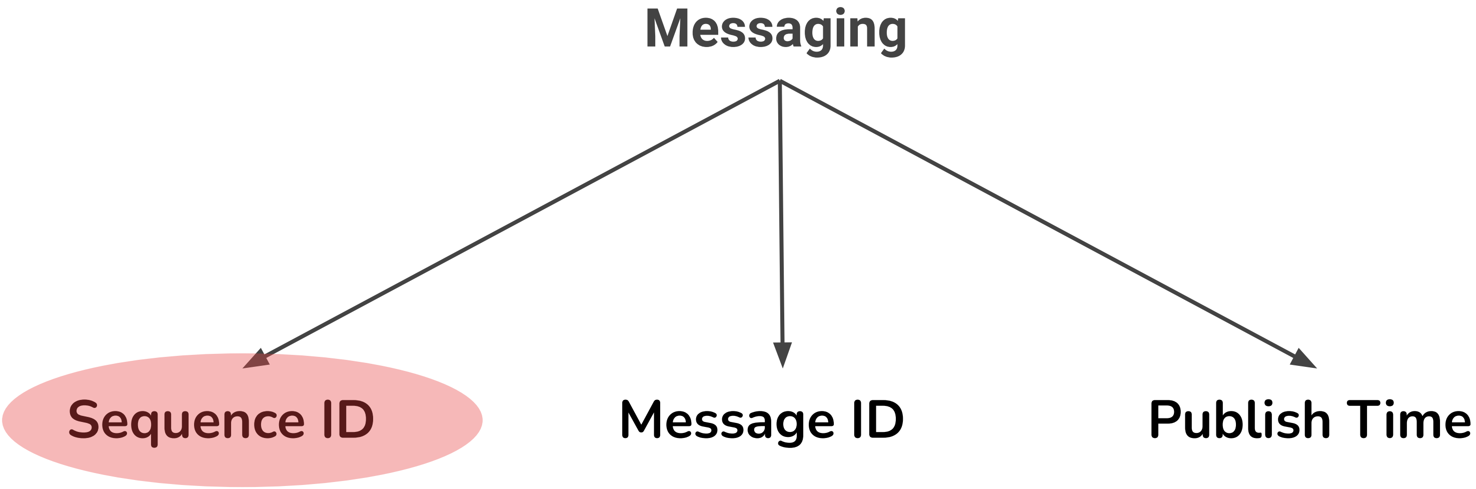
ReadFromPulsarDoFn (**topic**, [0, inf))

ReadFromPulsarDoFn (**topic**, [0, 100))

ReadFromPulsarDoFn (**topic**, [100, 150))

ReadFromPulsarDoFn (**topic**, [150, inf))

Pulsar messaging



Messaging

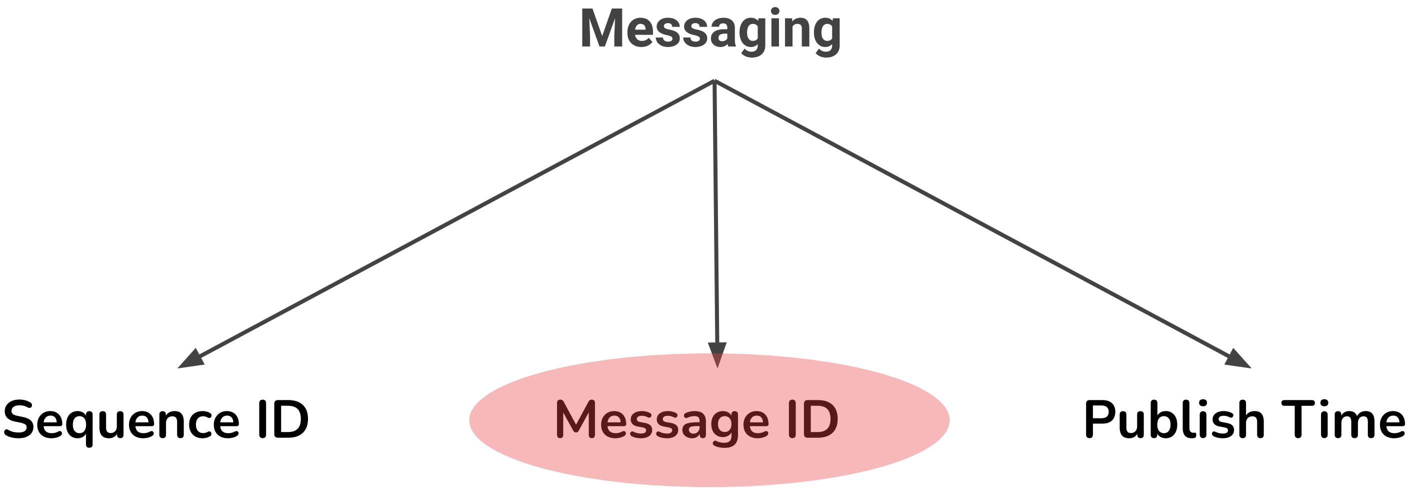
Sequence ID

Message ID

Publish Time

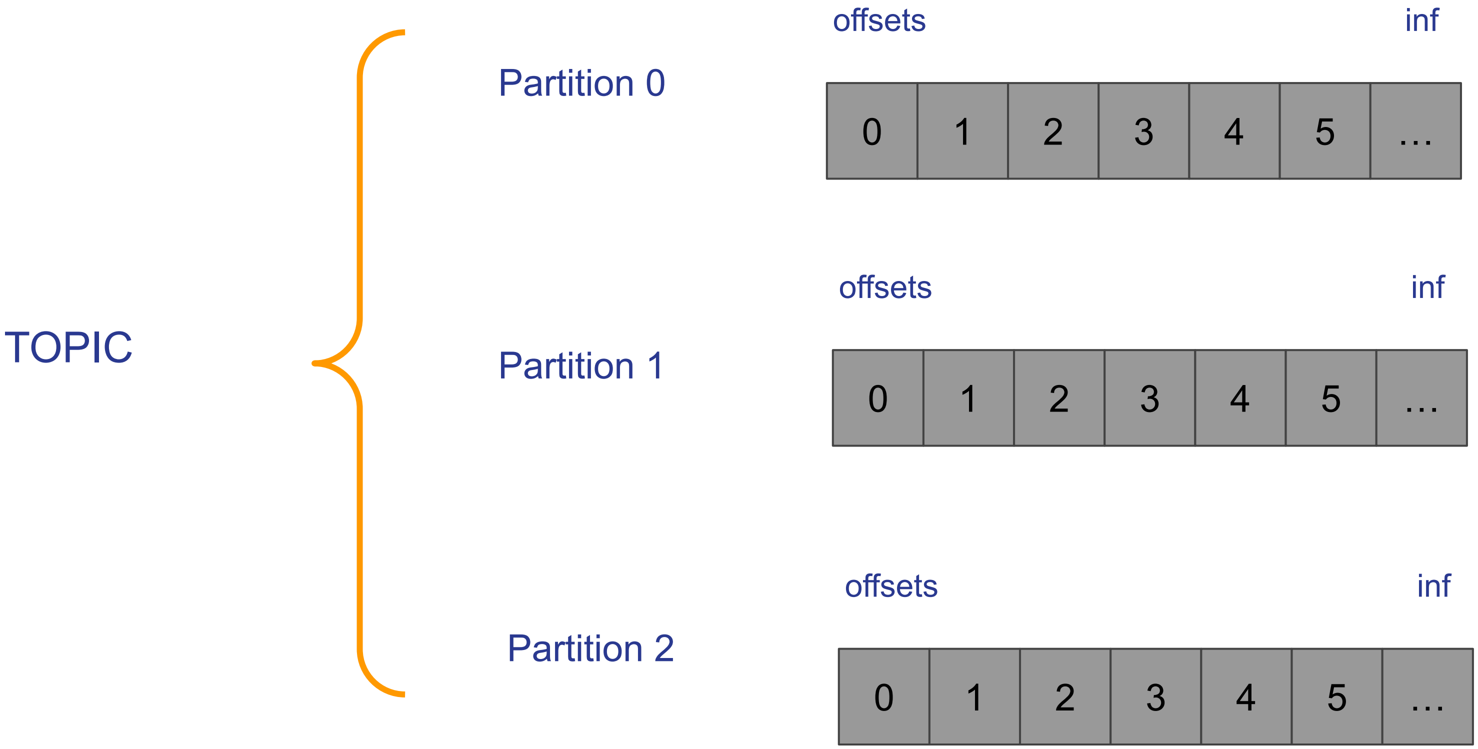
https://pulsar.apache.org/docs/concepts-messaging

Pulsar messaging



https://pulsar.apache.org/docs/concepts-messaging

In Kafka



TOPIC

Partition 0

offsets inf

0	1	2	3	4	5	...

Partition 1

offsets inf

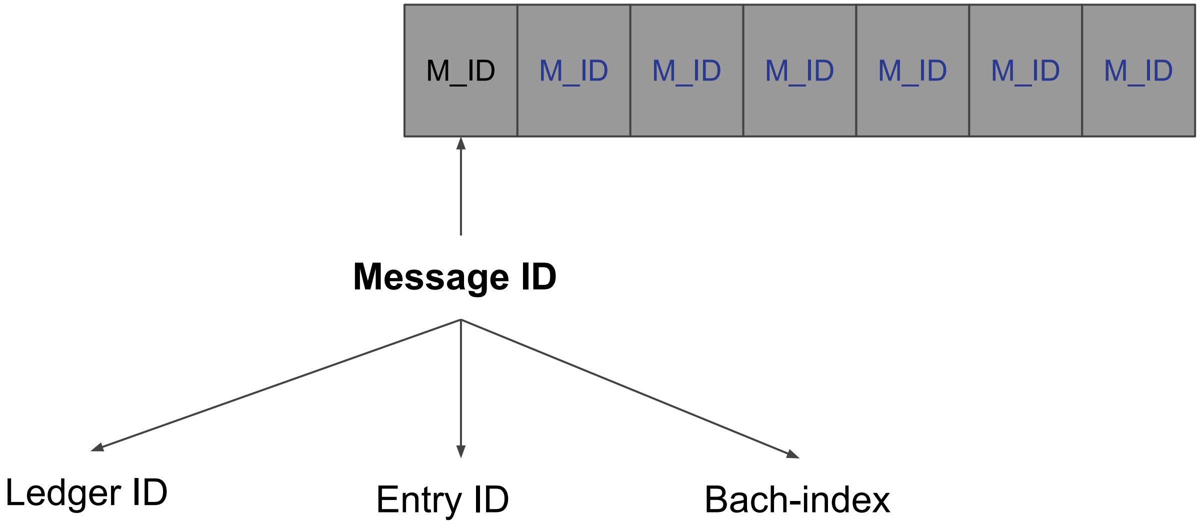
0	1	2	3	4	5	...

Partition 2

offsets inf

0	1	2	3	4	5	...

In Pulsar



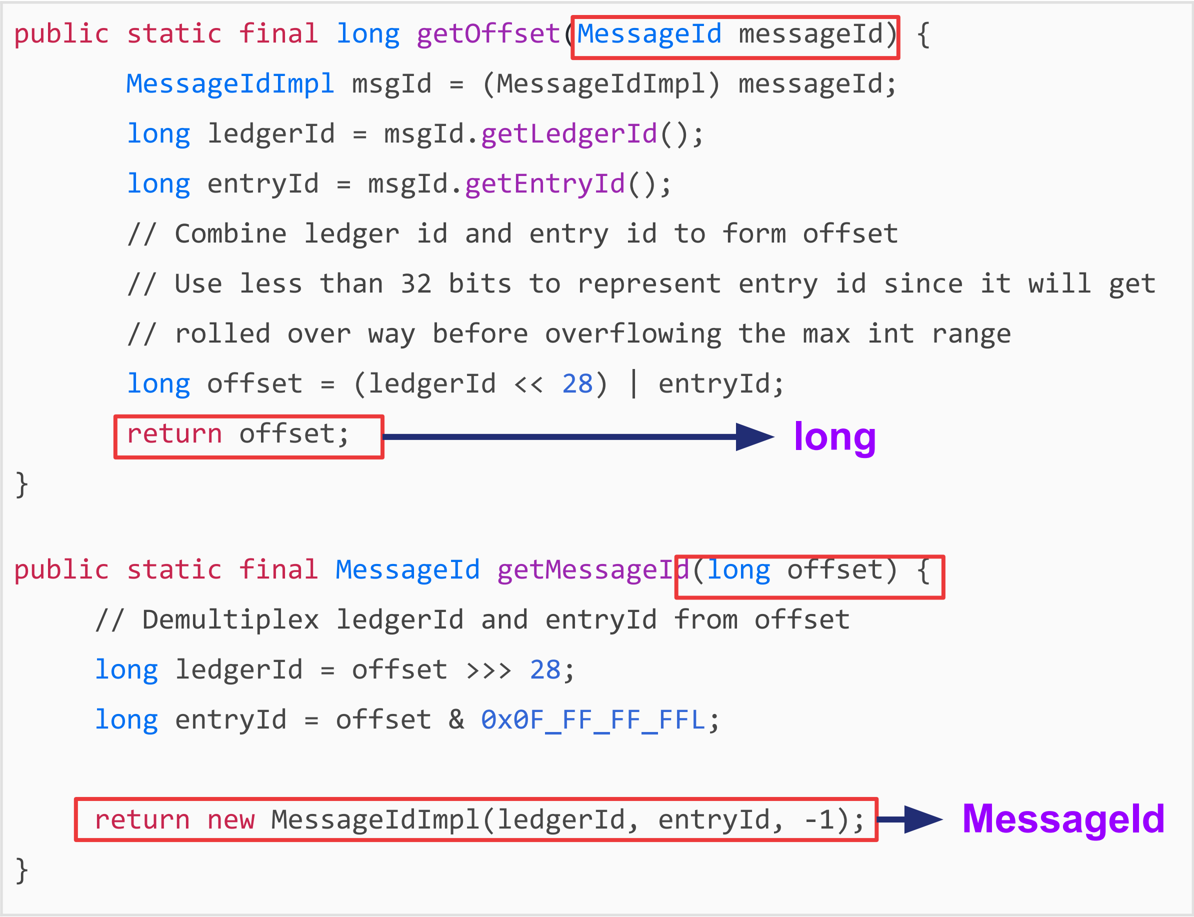
M_ID	M_ID	M_ID	M_ID	M_ID	M_ID	M_ID

Message ID

Ledger ID

Entry ID

Bach-index

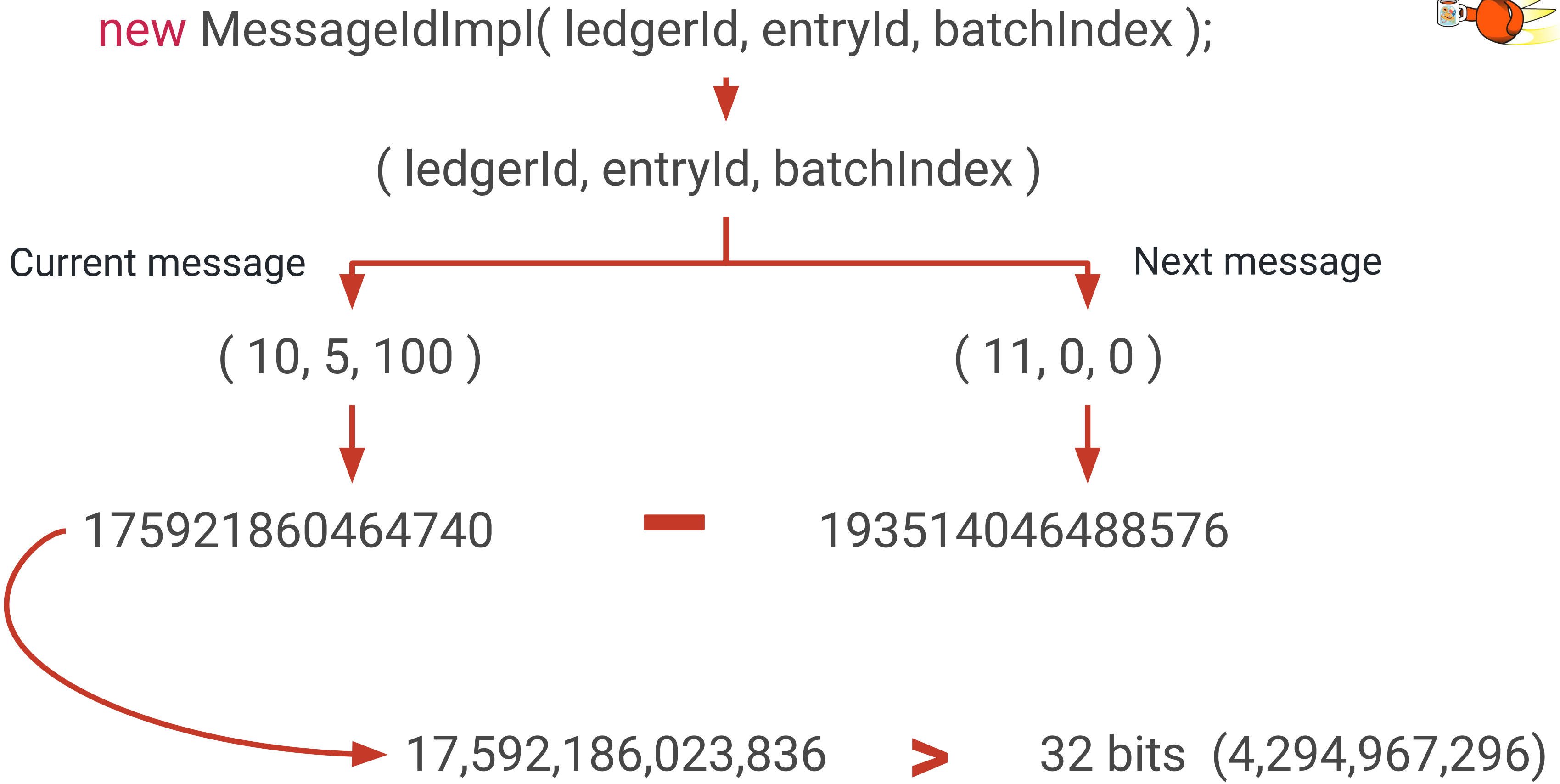


```
public static final long getOffset(MessageId messageId) {
       MessageIdImpl msgId = (MessageIdImpl) messageId;
       long ledgerId = msgId.getLedgerId();
       long entryId = msgId.getEntryId();
       // Combine ledger id and entry id to form offset
       // Use less than 32 bits to represent entry id since it will get
       // rolled over way before overflowing the max int range
       long offset = (ledgerId << 28) | entryId;
       return offset;
}

public static final MessageId getMessageId(long offset) {
     // Demultiplex ledgerId and entryId from offset
     long ledgerId = offset >>> 28;
     long entryId = offset & 0x0F_FF_FF_FFL;

     return new MessageIdImpl(ledgerId, entryId, -1);
}
```

long

MessageId

https://github.com/apache/pulsar/blob/master/pulsar-client/src/main/java/org/apache/pulsar/client/util/MessageIdUtils.java



```
new MessageIdImpl( ledgerId, entryId, batchIndex );
```

(ledgerId, entryId, batchIndex)

Current message

Next message

(10, 5, 100)

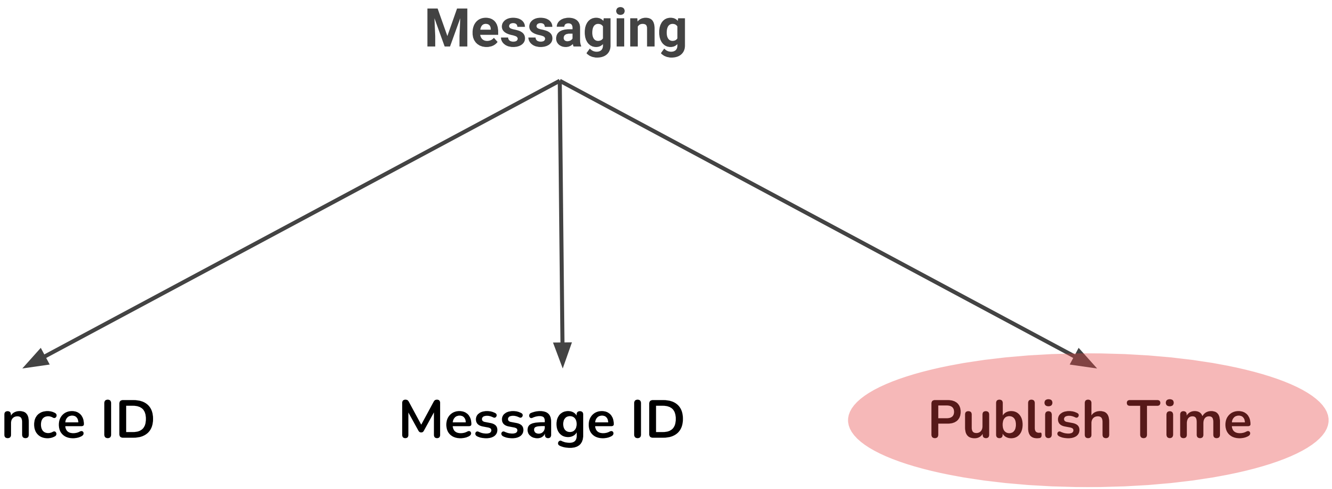
(11, 0, 0)

175921860464740 − 193514046488576

17,592,186,023,836 > 32 bits (4,294,967,296)

What can we do?

Publish time



Messaging

Sequence ID

Message ID

Publish Time

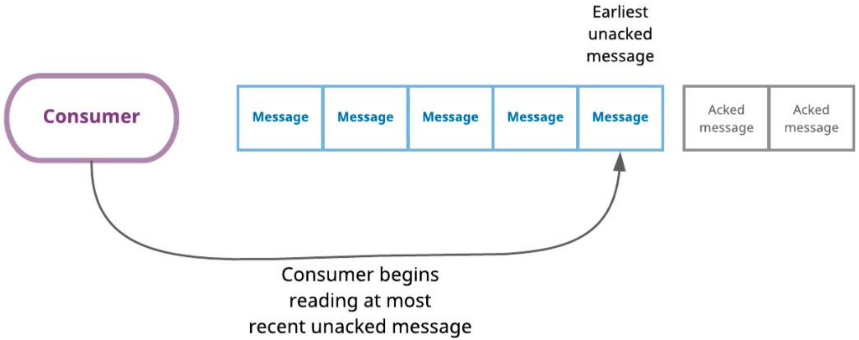
https://pulsar.apache.org/docs/concepts-messaging

Which client interface use?

Client interface

Consumer interface

Pulsar automatically manages topic cursors



Earliest unacked message

Consumer

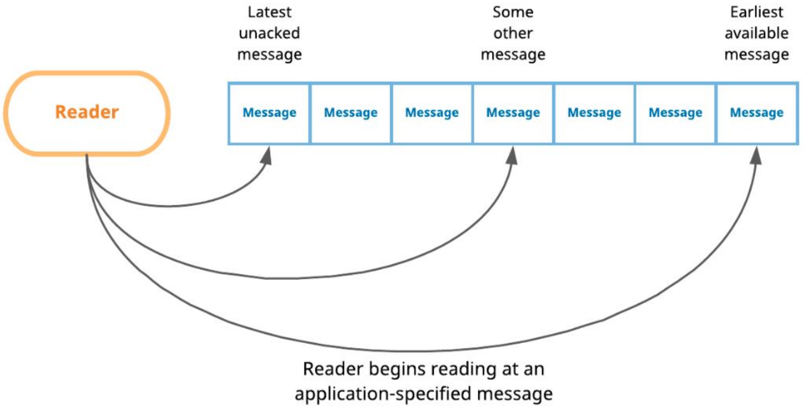
Message Message Message Message Message

Acked message Acked message

Consumer begins reading at most recent unacked message

Reader interface

Applications manually control topic cursors



Latest unacked message

Some other message

Earliest available message

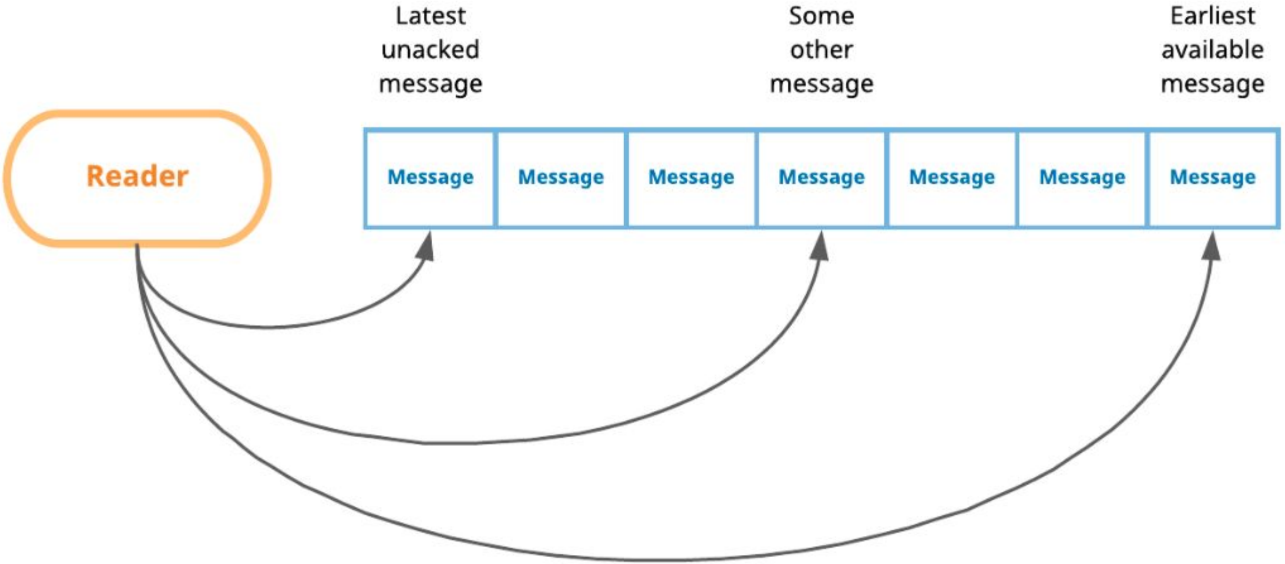
Reader

Message Message Message Message Message Message Message

Reader begins reading at an application-specified message

Reader interface

Applications manually control topic cursors

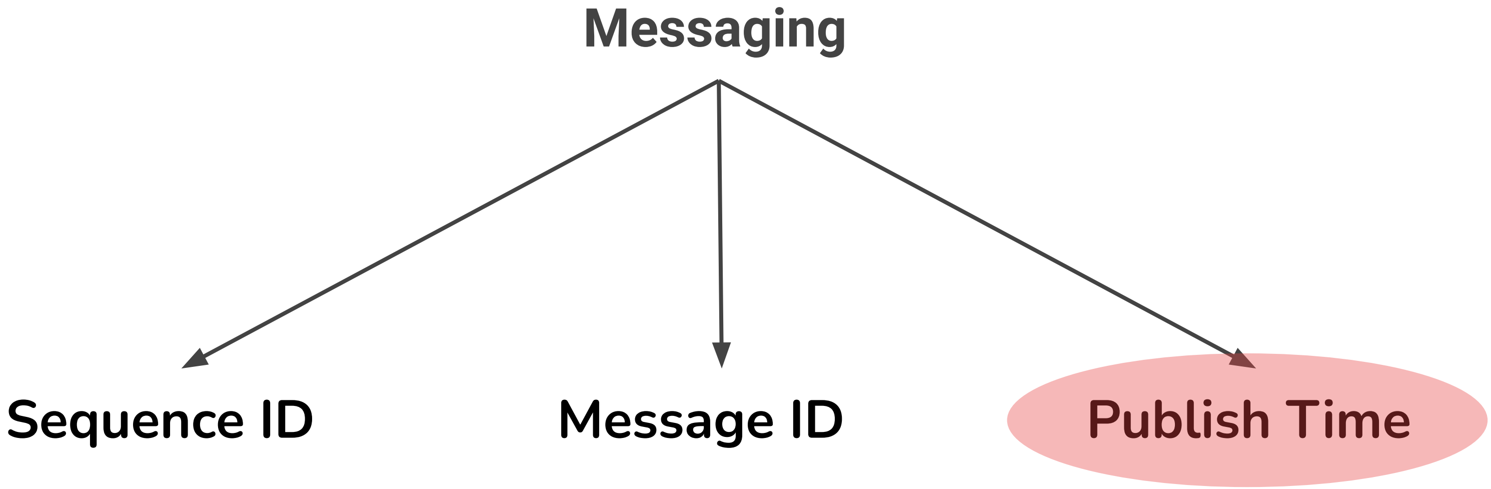


Reader begins reading at an application-specified message

Current implementation

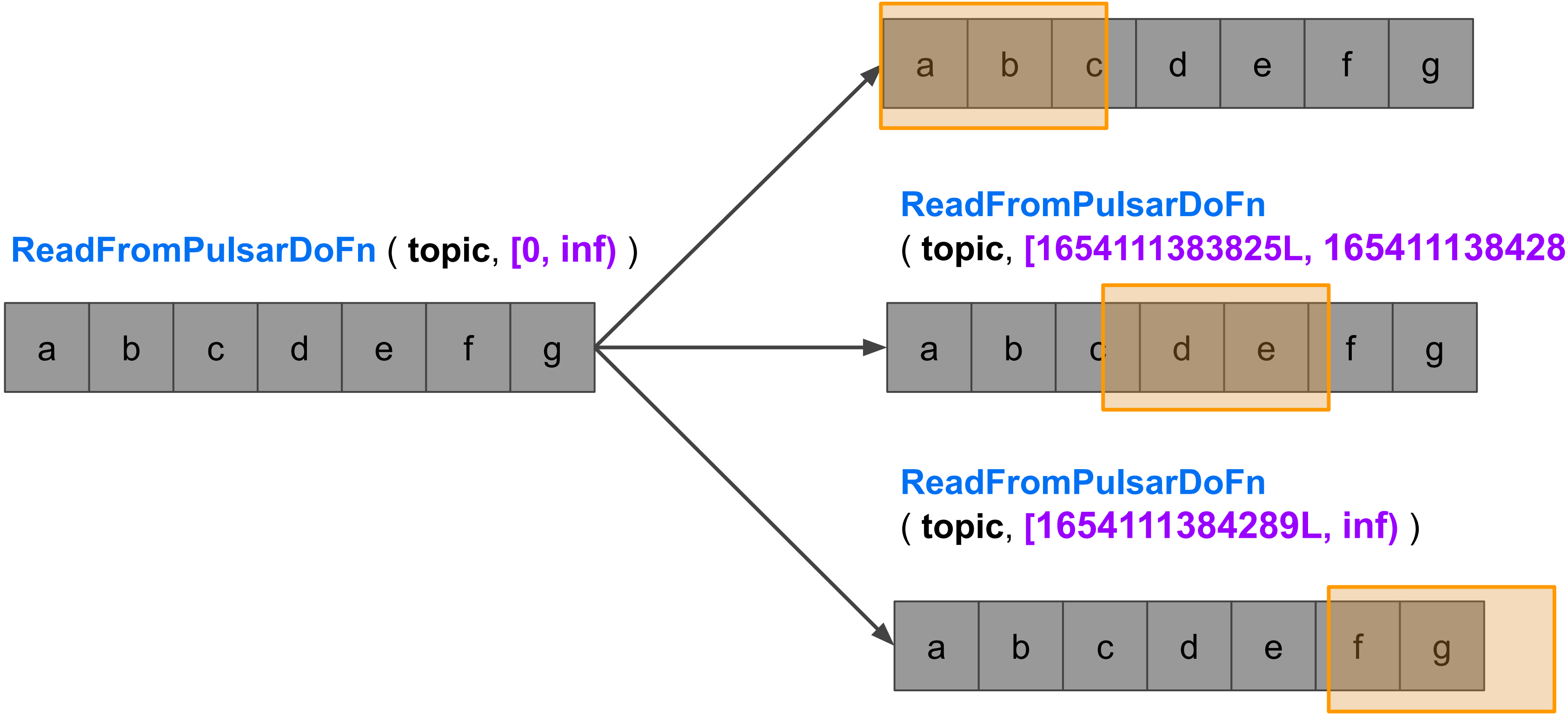
- Presentation
- What is Pulsar?
- Initial approach
- **Current implementation**
- Example
- Next steps
- Q&A

Publish time



https://pulsar.apache.org/docs/concepts-messaging

Restriction



ReadFromPulsarDoFn (topic, [0, inf))

ReadFromPulsarDoFn
(topic, [0, 1654111383825L))

ReadFromPulsarDoFn
(topic, [1654111383825L, 1654111384289L))

ReadFromPulsarDoFn
(topic, [1654111384289L, inf))

ReadFromPulsarDoFn

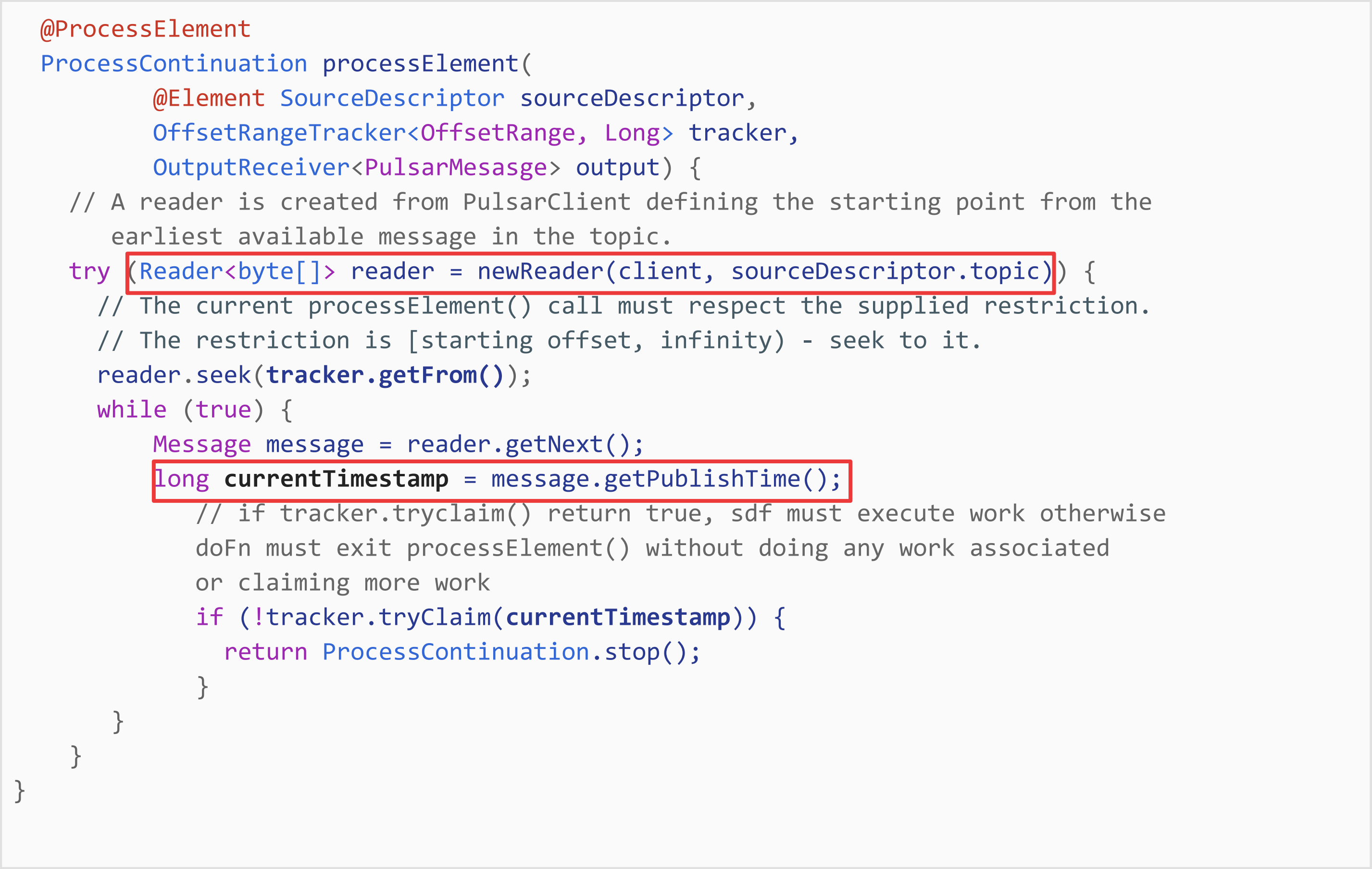
Splittable DoFn

@InitialRestriction

```
class SourceDescriptor { String topic; long startOffset; Message messageRecord }

@GetInitialRestriction
OffsetRange initialRestriction(sourceDescriptor) {
    long startTime = 0;
    long endTime = Long.MAX_VALUE;
    if ( sourceDescriptor.startOffset != null ) {
        startTime = sourceDescriptor.startOffset;
    }
    if ( sourceDescriptor.endOffset != null ) {
        endTime = sourceDescriptor.endOffset;
    }
    new OffsetRange(startTime, endTime);
}
```

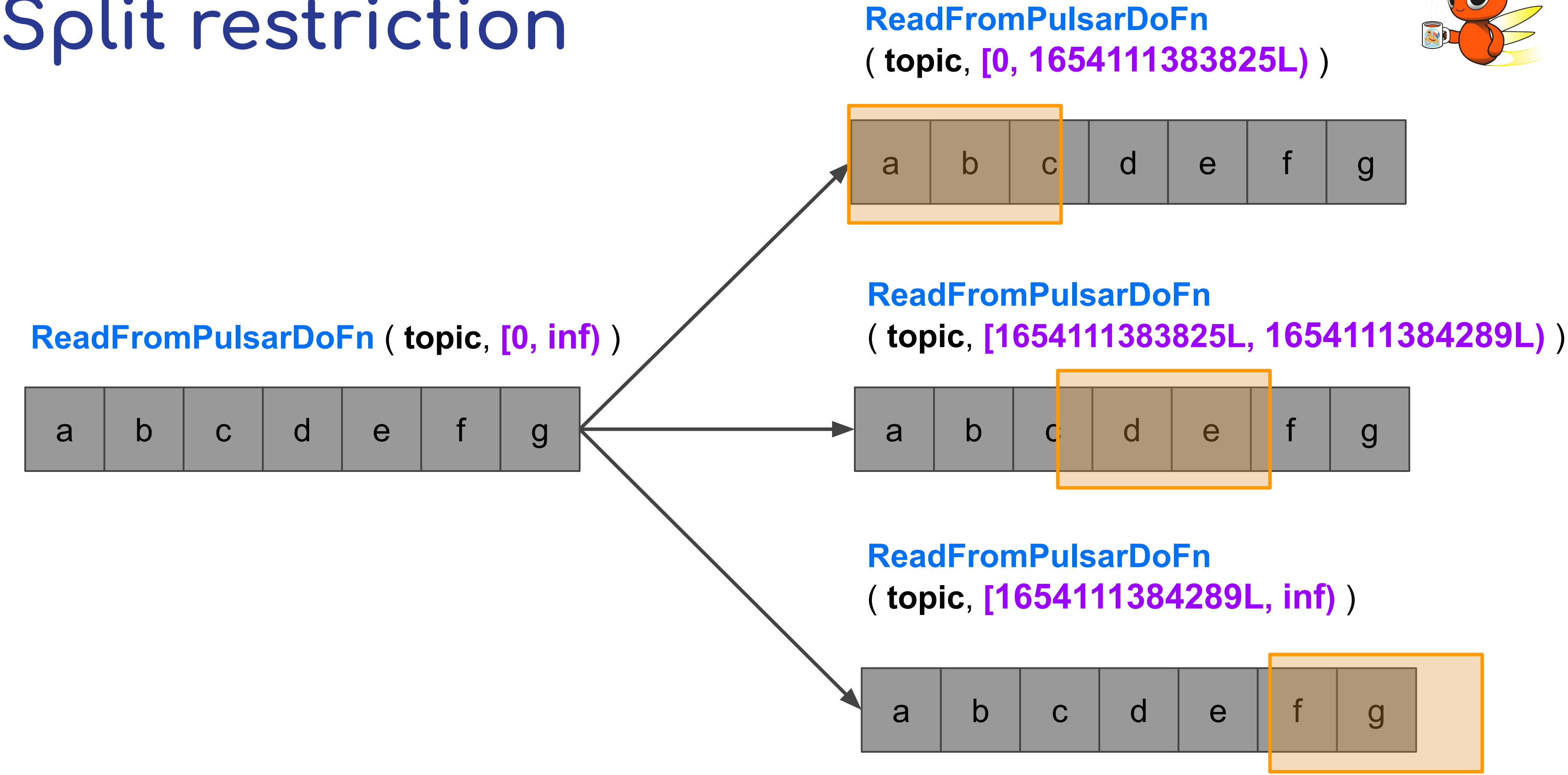
[0, inf)

@ProcessElement



```
  @ProcessElement
  ProcessContinuation processElement(
          @Element SourceDescriptor sourceDescriptor,
          OffsetRangeTracker<OffsetRange, Long> tracker,
          OutputReceiver<PulsarMesasge> output) {
    // A reader is created from PulsarClient defining the starting point from the
       earliest available message in the topic.
    try (Reader<byte[]> reader = newReader(client, sourceDescriptor.topic)) {
      // The current processElement() call must respect the supplied restriction.
      // The restriction is [starting offset, infinity) - seek to it.
      reader.seek(tracker.getFrom());
      while (true) {
          Message message = reader.getNext();
          long currentTimestamp = message.getPublishTime();
             // if tracker.tryclaim() return true, sdf must execute work otherwise
             doFn must exit processElement() without doing any work associated
             or claiming more work
             if (!tracker.tryClaim(currentTimestamp)) {
               return ProcessContinuation.stop();
             }
       }
    }
}
```

Split restriction



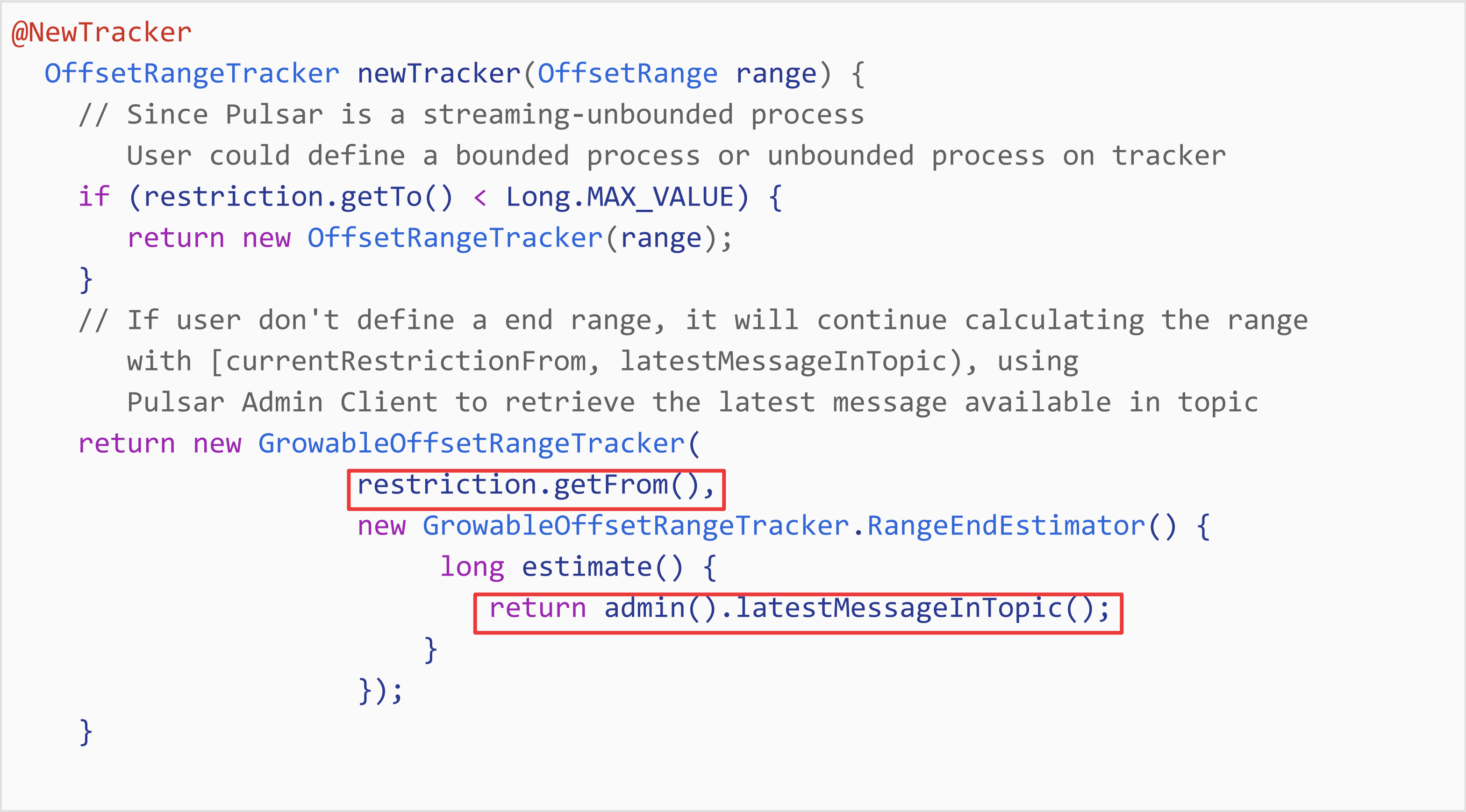
ReadFromPulsarDoFn (topic, [0, inf))

ReadFromPulsarDoFn
(topic, [0, 1654111383825L))

ReadFromPulsarDoFn
(topic, [1654111383825L, 1654111384289L))

ReadFromPulsarDoFn
(topic, [1654111384289L, inf))

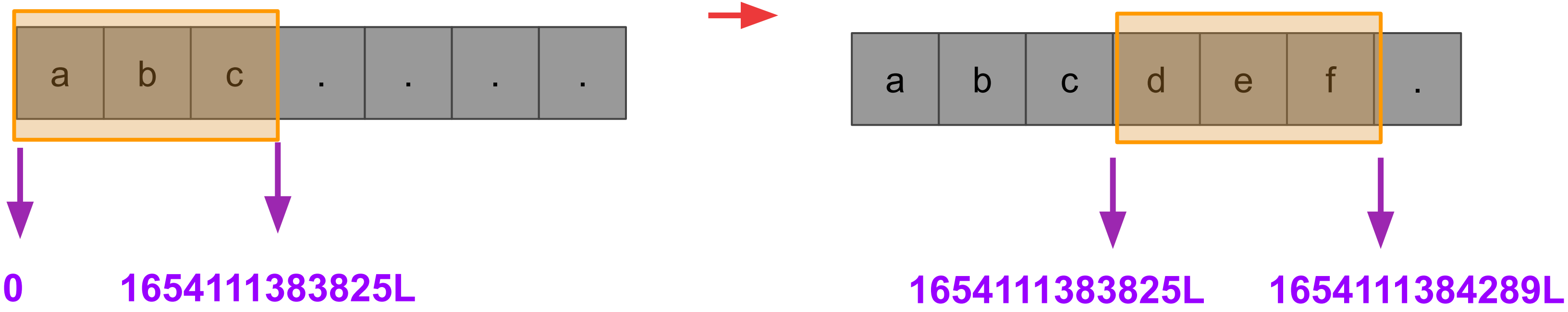
@NewTracker



```
@NewTracker
  OffsetRangeTracker newTracker(OffsetRange range) {
    // Since Pulsar is a streaming-unbounded process
       User could define a bounded process or unbounded process on tracker
    if (restriction.getTo() < Long.MAX_VALUE) {
       return new OffsetRangeTracker(range);
    }
    // If user don't define a end range, it will continue calculating the range
       with [currentRestrictionFrom, latestMessageInTopic), using
       Pulsar Admin Client to retrieve the latest message available in topic
    return new GrowableOffsetRangeTracker(
                     restriction.getFrom(),
                     new GrowableOffsetRangeTracker.RangeEndEstimator() {
                          long estimate() {
                              return admin().latestMessageInTopic();
                          }
                     });
    }
```

ReadFromPulsarDoFn
(**topic**, [0, 1654111383825L))

ReadFromPulsarDoFn
(**topic**,
[1654111383825L, 1654111384289L))



Watermark

Timestamp observing

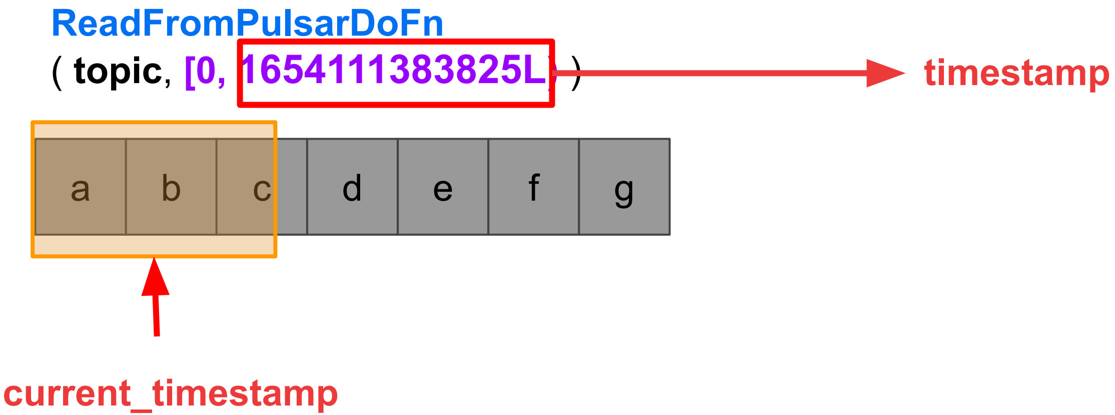
↓

Timestamp of each record

External clock observing

↓

Timestamp not associated



ReadFromPulsarDoFn

(topic, [0, 1654111383825L))

timestamp

a b c d e f g

current_timestamp

Watermark estimator

There are some build-on watermark estimator implementations in Java:

1. Manual
2. Monotonically increasing
3. Wall time

ReadFromPulsarIO has two types of timers:

Publish time

Processing time

```
@ProcessElement
ProcessContinuation processElement(
        @Element SourceDescriptor sourceDescriptor,
        OffsetRangeTracker<OffsetRange, Long> tracker,
        OutputReceiver<PulsarMesasge> output) {
      …
      PulsarMessage pulsarMessage =
          new PulsarMessage(message.getTopicName(),
                            message.getPublishTime(),
                            message);
      Instant outputTimestamp = extractOutputTimestampFn.apply(message);
      output.outputWithTimestamp(pulsarMessage, outputTimestamp);

}
```

```
static class ExtractOutputTimestampFn {
    public static SerializableFunction<Message<byte[]>, Instant>
           useProcessingTime() {
      return record -> Instant.now();
    }

    public static SerializableFunction<Message<byte[]>, Instant>
           usePublishTime() {
      return record -> new Instant(record.getPublishTime());
    }
}
```

Example

- Presentation
- What is Pulsar?
- Initial approach
- Current implementation
- **Example**
- Next steps
- Q&A

PulsarIO Reader

```
PulsarIO.Read reader = PulsarIO.read()
              .withClientUrl("pulsar_client_url")
              .withPulsarClient(SerializableFunction…)
              .withAdminUrl("pulsar_admin_url")
              .withTopic("topic")
              .withStartTimestamp(startTime)
              .withEndTimestamp(endExpectedTime)
              .withPublishTime();

pipeline.apply(reader);
```

PulsarIO Reader

```java
PulsarIO.Read reader = PulsarIO.read()
              .withClientUrl("pulsar_client_url")
              .withPulsarClient(SerializableFunction…)
              .withAdminUrl("pulsar_admin_url")
              .withTopic("topic")
              .withStartTimestamp(startTime)
              .withEndTimestamp(endExpectedTime)
              .withPublishTime()
              .withProcessingTime();

pipeline.apply(reader);
```

PulsarIO Writer

```
PulsarIO.Write writer = PulsarIO.write()
              .withClientUrl("pulsar_client_url")
              .withTopic("topic");

List<byte[]> messages = new ArrayList<>();
messages.add("MESSAGE_1".getBytes());
messages.add("MESSAGE_2".getBytes());

pipeline.apply(Create.of(messages))
        .apply(writer);
```

Next steps

- Presentation
- What is Pulsar?
- Initial approach
- Current implementation
- Example
- **Next steps**
- Q&A

BEAM SUMMIT

A lot work to do...

- Acknowledge messages
- Multi-topic partition
- Set a dynamic stop limit for reader and writer
- Allow subscription types
- ...

Thanks

Questions?

marco.robles@wizeline.com | dev@beam.apache.org
linkedin.com/in/marcoantoniorob
github.com/MarcoRob