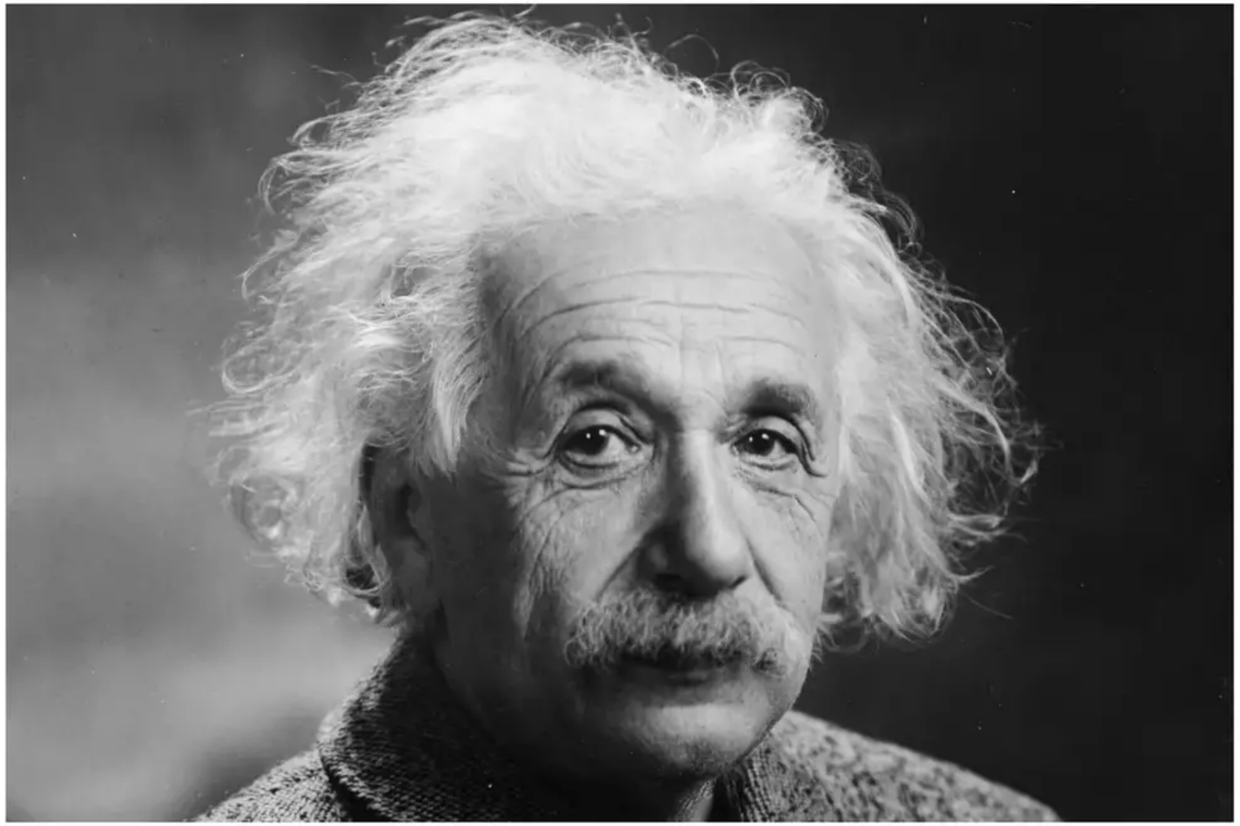
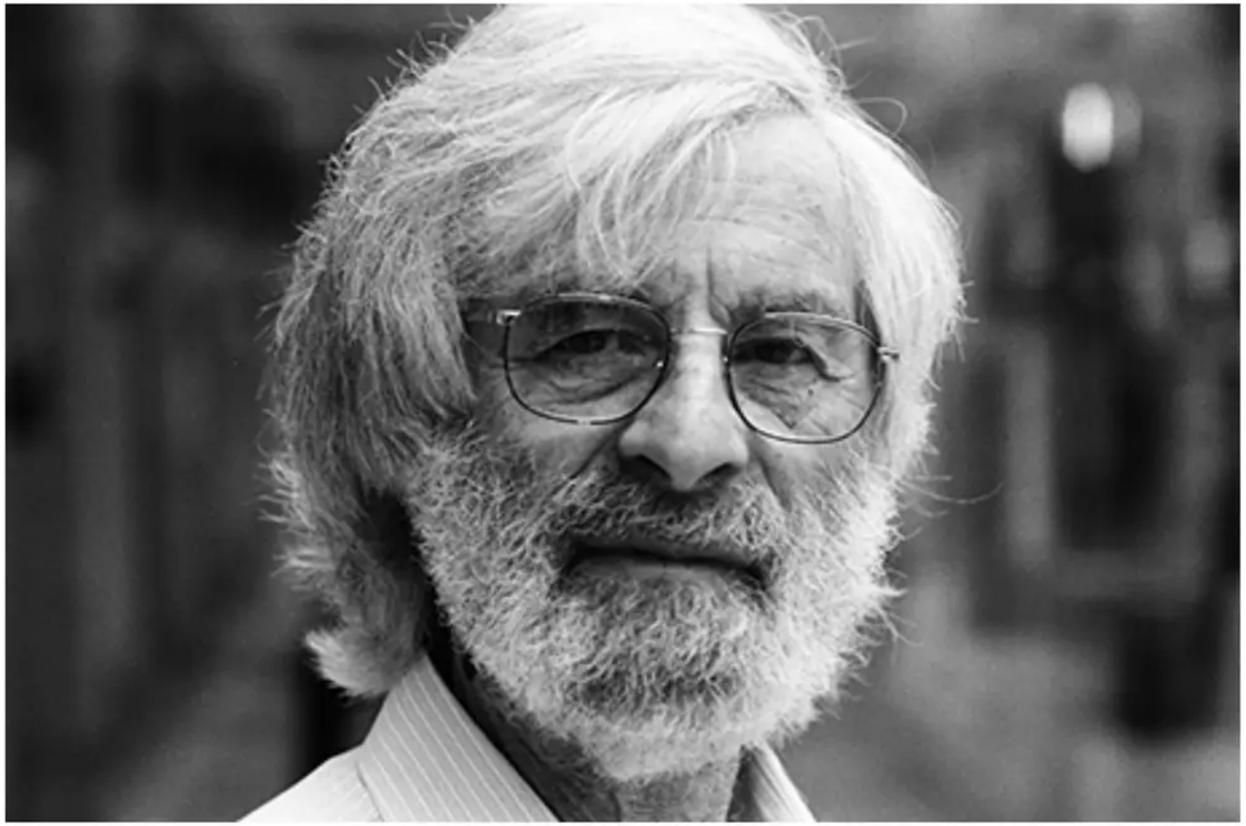
confluent

What's the Time? …and Why?

Matthias J. Sax | Software Engineer | @MatthiasJSax





The Notion of Time

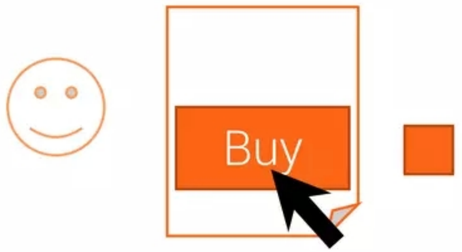
09:34:23.473

10:24:13.001 10:12:46.937

Application

09:34:23.784 09:34:23.534

topic



09:34:23.146

10:24:12.899 10:24:12.538

Application

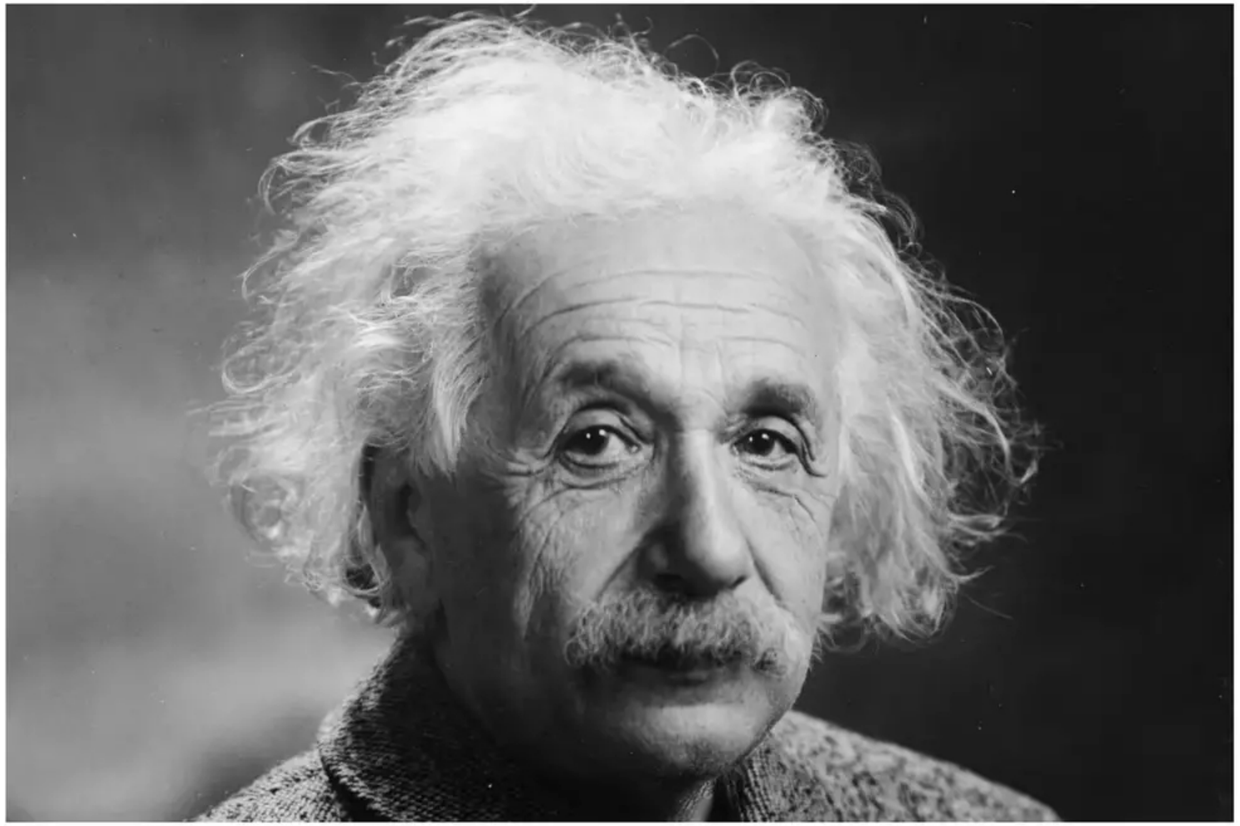
Storing Data Streams

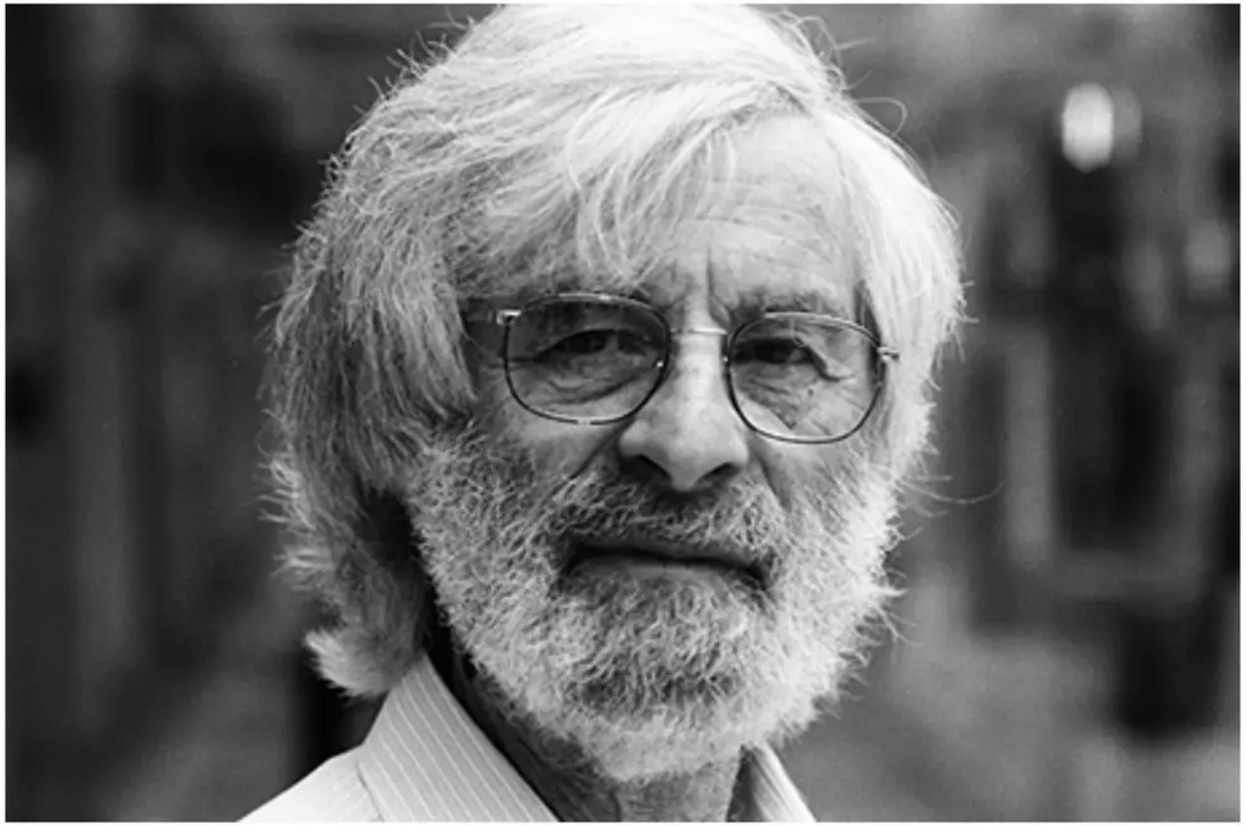
offsets, timestamps, order

confluent

What's the Time? …and Why?

Matthias J. Sax | Software Engineer | @MatthiasJSax





The Notion of Time

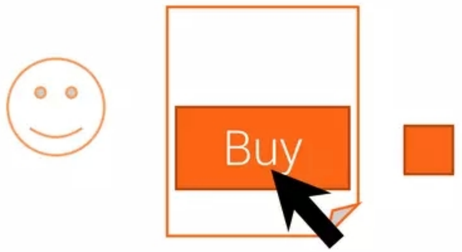
09:34:23.473

10:24:13.001 10:12:46.937

Application

09:34:23.784 09:34:23.534

topic



09:34:23.146

10:24:12.899 10:24:12.538

Application

Storing Data Streams

offsets, timestamps, order

Producing Data

```
public class ProducerRecord {
    public ProducerRecord(…);
    public ProducerRecord(…, long timestamp, …); // as of v0.10.0
}
```

- If no timestamp is set by the user, `KafkaProducer` sets current system timestamp in `#send()`

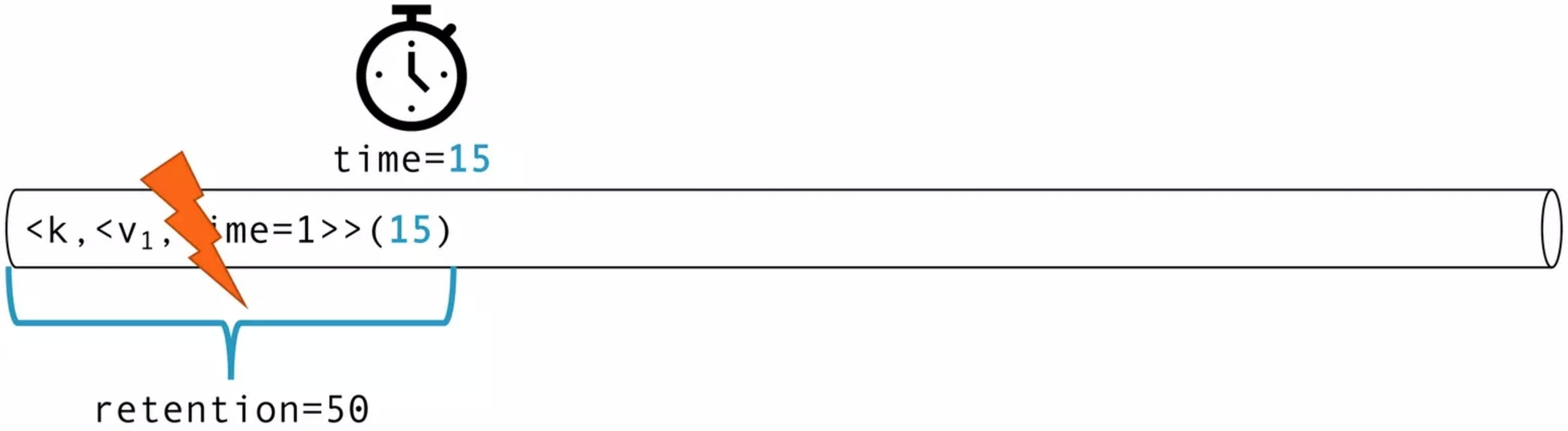
- Requires message format 0.10 (note that *broker version != message format*)
  - Producers can still write into topics with 0.9 message format; timestamp is dropped (down conversion)
    -> embed timestamp in payload, i.e., key or value (as of `v0.11.0`, maybe headers)
  - If older producers send messages, brokers sets timestamp to `-1` (i.e., unknown) (up conversion)*

- Timestamps cannot be negative: KIP-228

*) depending on broker/topic config

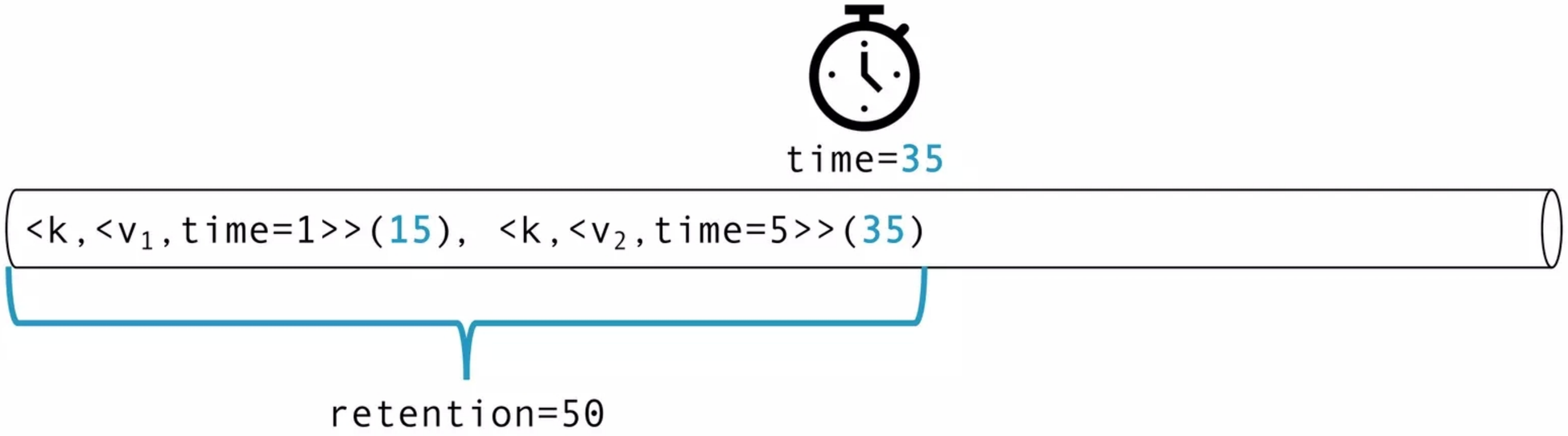
Data Retention and Writing “old” Data

- Retention semantics in `v0.9.0` and older:
  - Purely system time based
  - No message timestamp:
    - Event timestamp must be embedded in key or value
    - Event timestamp irrelevant for data retention



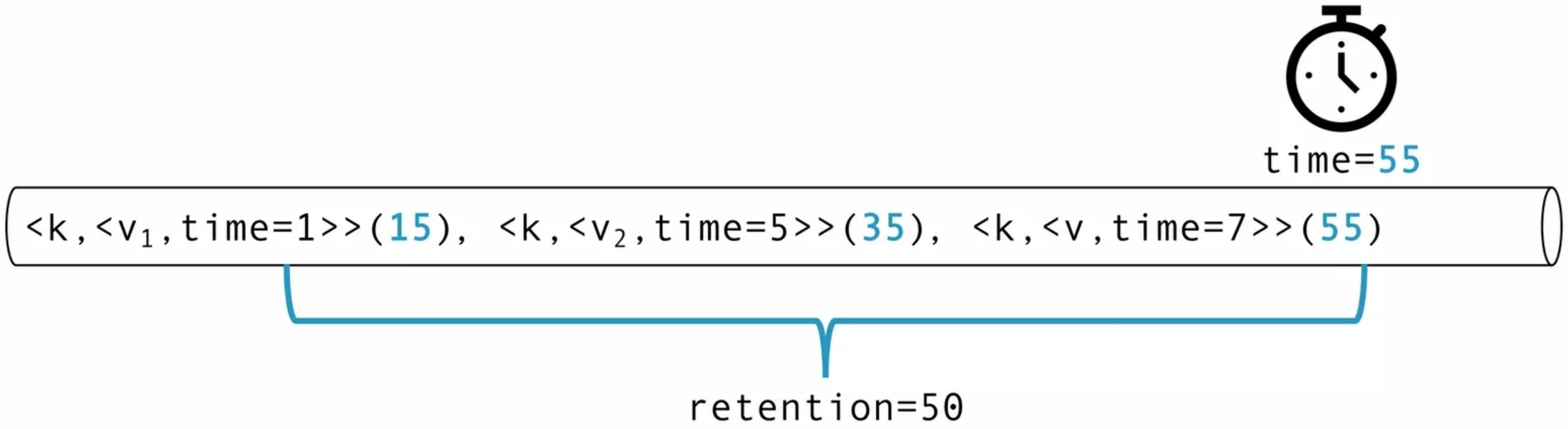
Data Retention and Writing “old” Data

- Retention semantics in `v0.9.0` and older:
  - Purely system time based
  - No message timestamp:
    - Event timestamp must be embedded in key or value
    - Event timestamp irrelevant for data retention



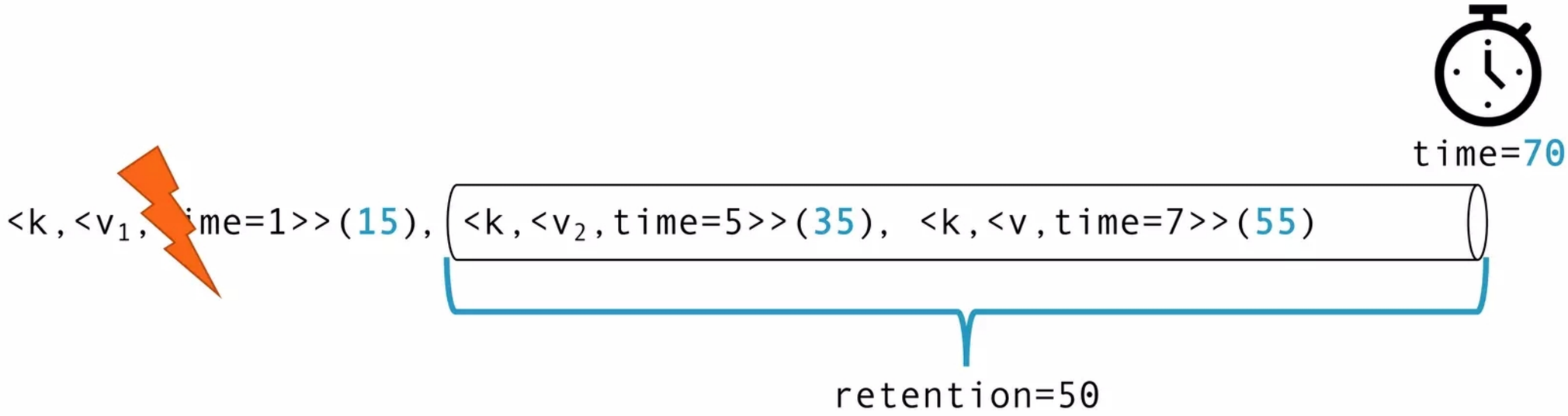
Data Retention and Writing “old” Data

- Retention semantics in `v0.9.0` and older:
  - Purely system time based
  - No message timestamp:
    - Event timestamp must be embedded in key or value
    - Event timestamp irrelevant for data retention



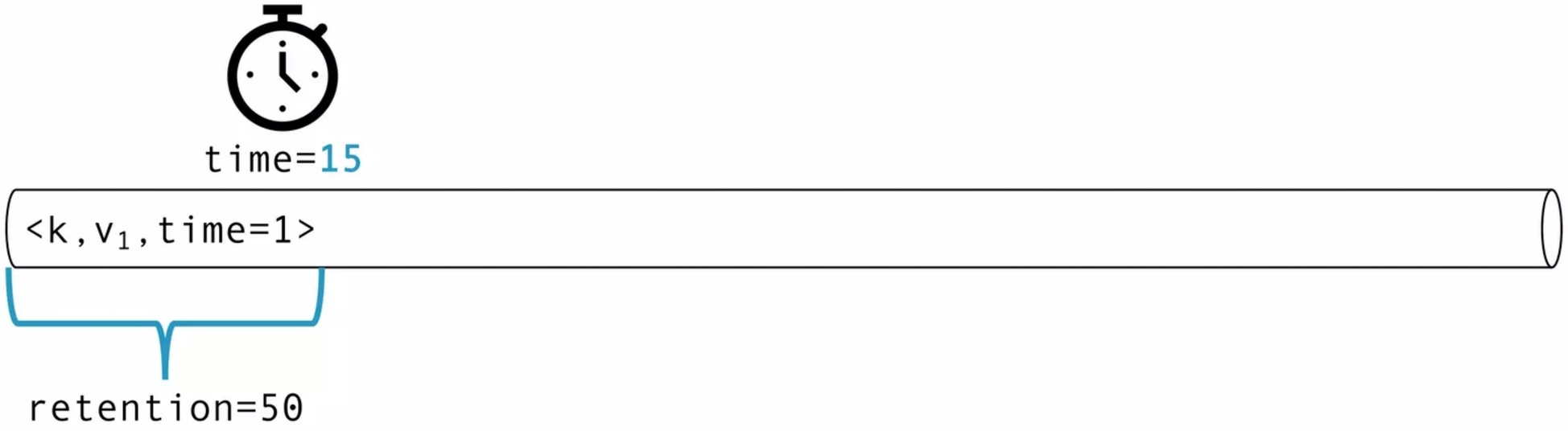
Data Retention and Writing “old” Data

- Retention semantics in `v0.9.0` and older:
  - Purely system time based
  - No message timestamp:
    - Event timestamp must be embedded in key or value
    - Event timestamp irrelevant for data retention



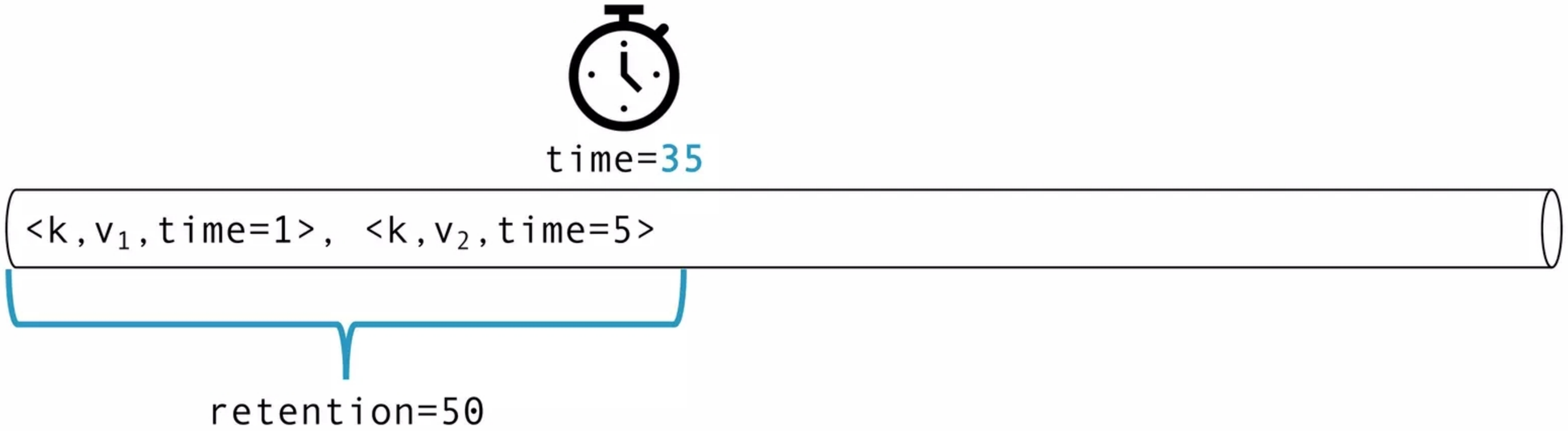
Data Retention and Writing “old” Data (cont.)

- Retention semantics in `v0.10.0` and newer:
  - Message timestamp dictates “age” of a message
    - If producer sets ”old” timestamps, segments will be rolled quickly
    - Writing “old” data requires larger retention time



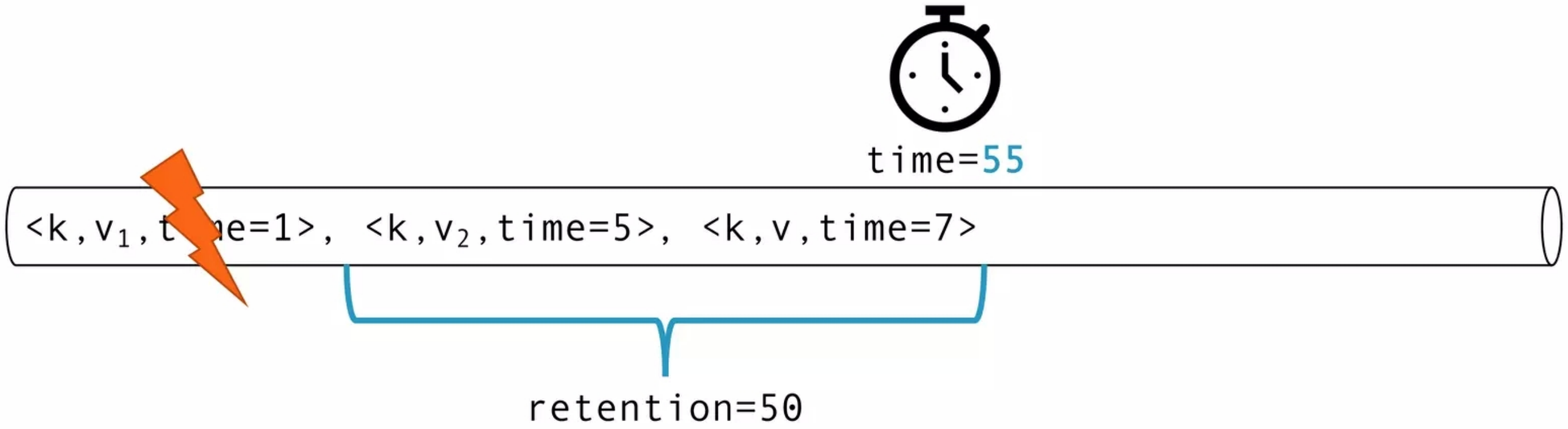
Data Retention and Writing “old” Data (cont.)

- Retention semantics in `v0.10.0` and newer:
  - Message timestamp dictates “age” of a message
    - If producer sets ”old” timestamps, segments will be rolled quickly
    - Writing “old” data requires larger retention time



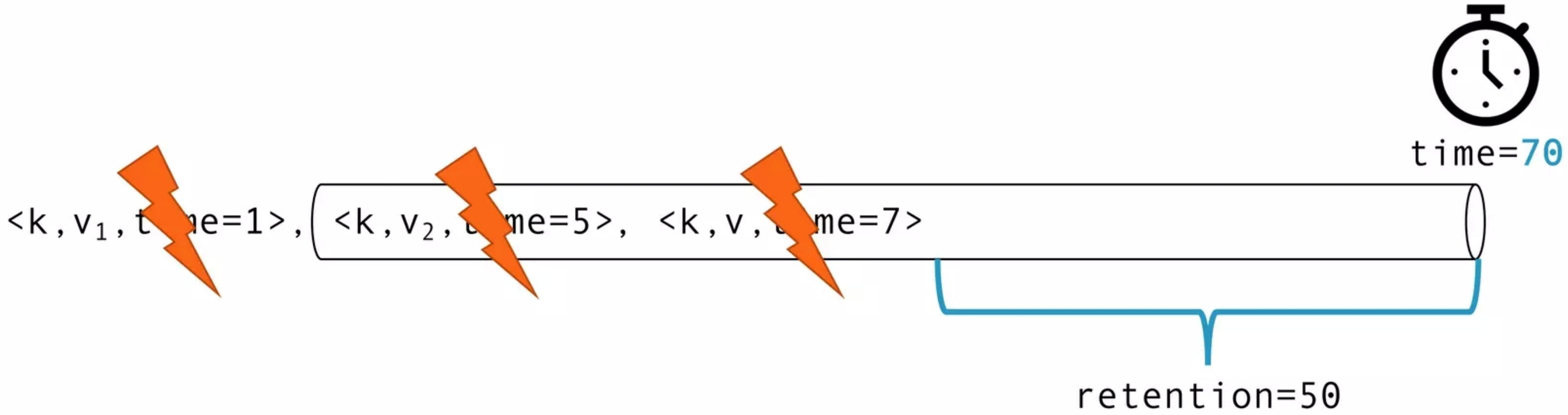
Data Retention and Writing “old” Data (cont.)

- Retention semantics in `v0.10.0` and newer:
  - Message timestamp dictates “age” of a message
    - If producer sets ”old” timestamps, segments will be rolled quickly
    - Writing “old” data requires larger retention time



Data Retention and Writing “old” Data (cont.)

- Retention semantics in `v0.10.0` and newer:
  - Message timestamp dictates “age” of a message
    - If producer sets ”old” timestamps, segments will be rolled quickly
    - Writing “old” data requires larger retention time



No Clock – Old Data – What Now?

I cannot guarantee that the producer system clock is synchronized.

I ingest sensor data, and there *is* no local clock.

I use an older producer and cannot set a timestamp at all.

I want to insert “old” data but keep a short retention time.

Log Append Time for Rescue

- Configuration parameters:
  - `log.message.timestamp.type` (broker)
  - `message.timestamp.type` (topic level)
  - Accepted values: “`CreateTime`” / “`LogAppendTime`”

- CreateTime (default):
  - Broker preserves producer specified timestamp

- LogAppendTime:
  - Broker overwrites message timestamp with its current local system timestamp

  - Log append time can be an approximation* of create time
  - Log append time provides ”old” log retention time semantics
  - Event time can be embedded in payload, i.e., key, value, or header

*) assuming write happen quickly after a message was created–i.e., an event happened–and broker clocks are accurate

Timestamp Summary

- Message timestamps
  (as of v0.10.0)
- CreateTime vs LogAppendTime
- Retention
- Embed event time in payload

Kafka has strong ordering guarantees*!

Kafka Streams support out-of-order data processing!

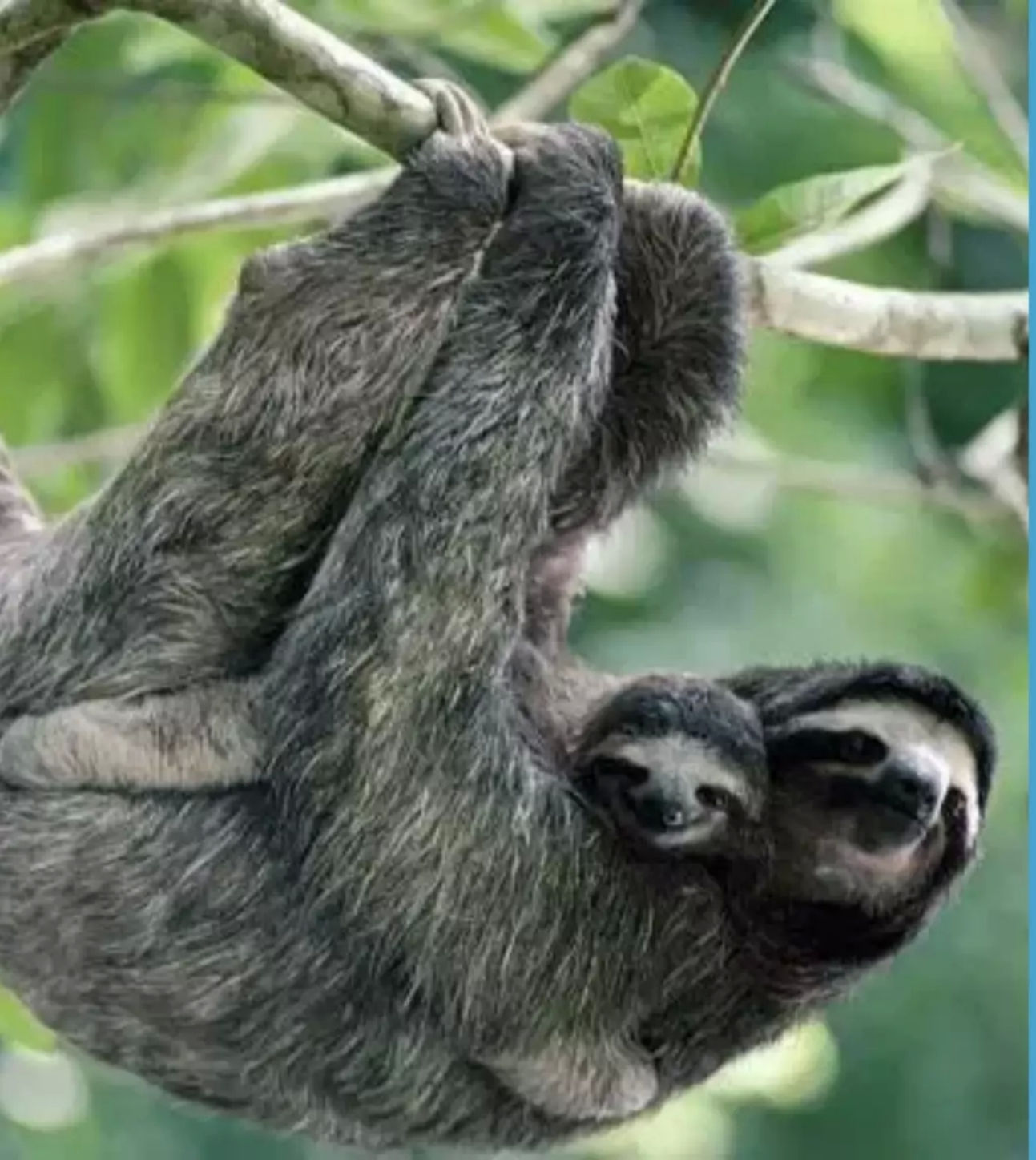
*) per partition

How can data be out-of-order if Kafka guarantees ordering*?

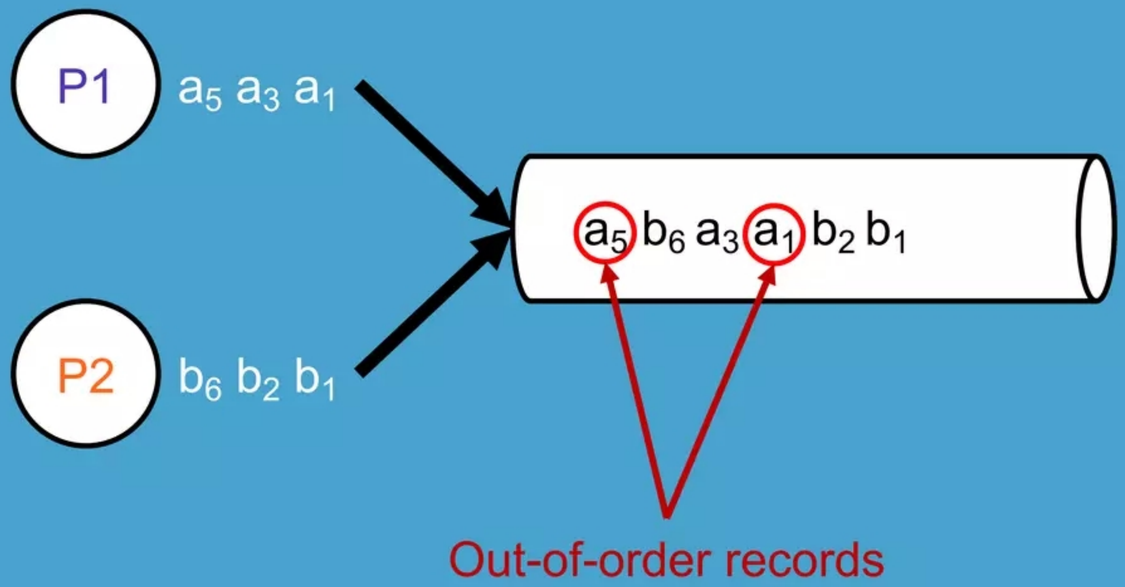
*) per partition

Offset vs. Timestamp Order

- Kafka guarantees that all consumers see messages in the same *offset* order (per partition)
- There is no guarantee that messages are appended in timestamp order though



Interleaved Writes



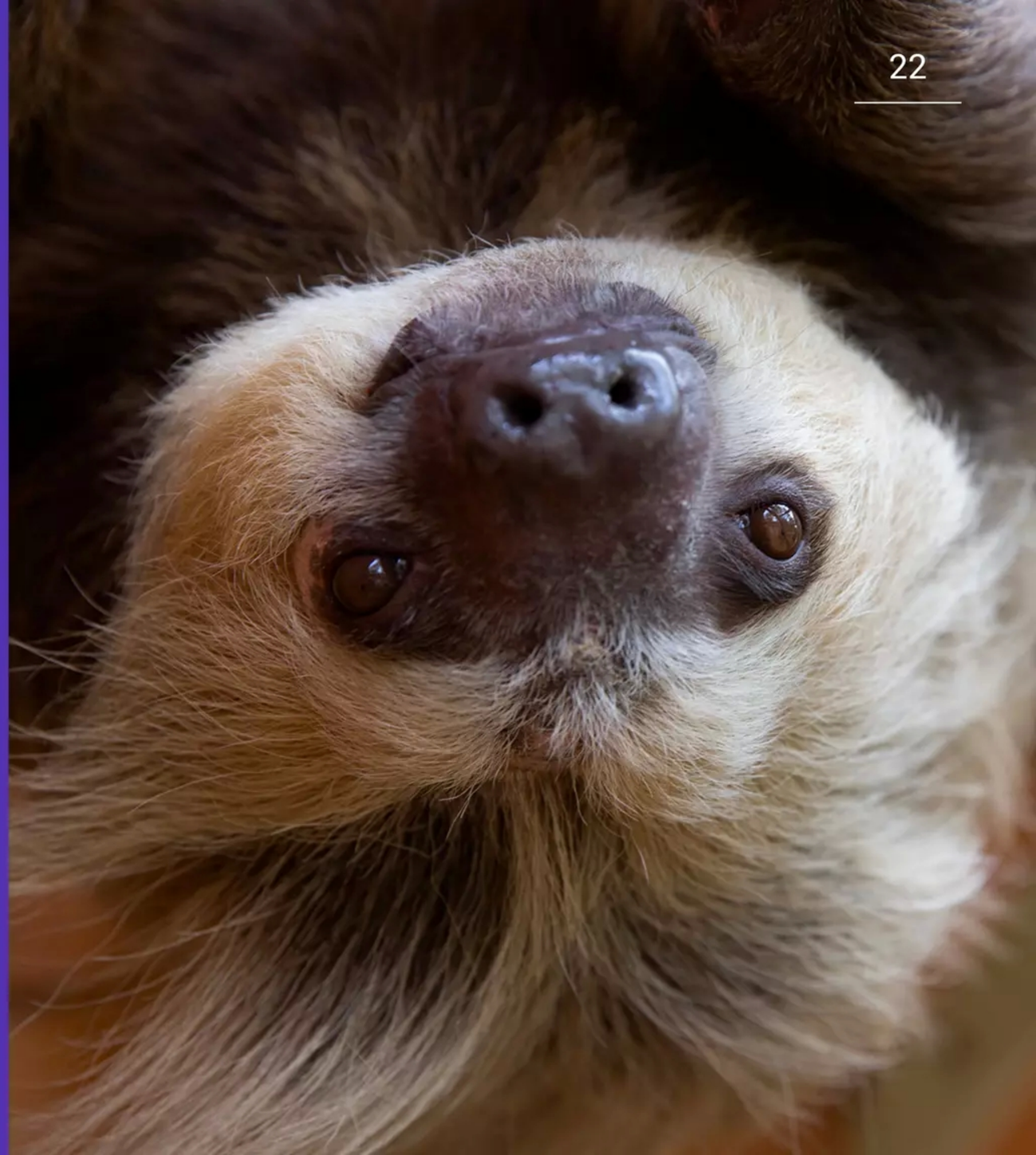
Single Writer Principle (strict)

- There is exactly one producer per partition
- Guarantees timestamp order
- Watch your config!
  - `max.in.flight.requests.per.connection = 1`
  - Or: `enable.idempotence = true`

@MatthiasJSax

Single Writer Principle (relaxed)

- There is exactly one producer *per key*
- Out-of-order data possible across different keys



"Enforced" Timestamp Order

- Use *LogAppendTime*
- Timestamp order "guaranteed"; remember: best, it's an approximation of event time

Consuming Data Streams

Processing Order, Position, and Time

- Initial startup behavior *(no committed offsets yet)*
  - Configuration: `auto.offset.reset`
  - `KafkaConsumer`: “`earliest`” / “`latest`” (default)
  - `KafkaStreams`: “`earliest`” (default) / “`latest`”

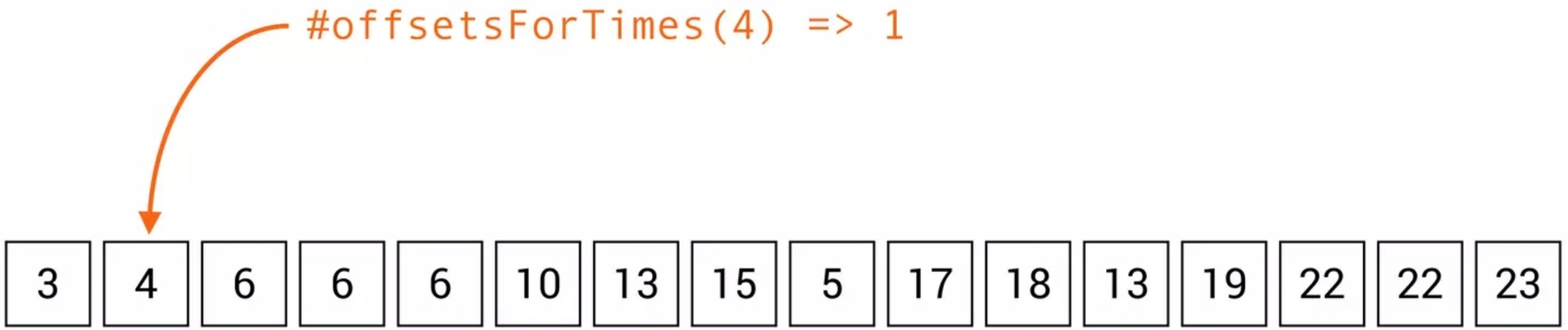
- Seek to offset (`KafkaConsumer` only)
  - `#seek()`, `#seekToBeginning()`, `#seekToEnd()`

- Seek by timestamp (`KafkaConsumer` only) (as of `v0.10.1`)
  - `#offsetsForTimes()` + `#seek()`
  - Uses message timestamp; seeking to timestamp contained in payload is ***not*** supported

- Command line tools:
  - Commit offset based on offset or timestamp (each absolute or relative) before startup
  - `bin/kafka-consumers-group.sh` (as of v0.11.0)
  - `bin/kafka-streams-application-reset.sh` (as of v1.1)

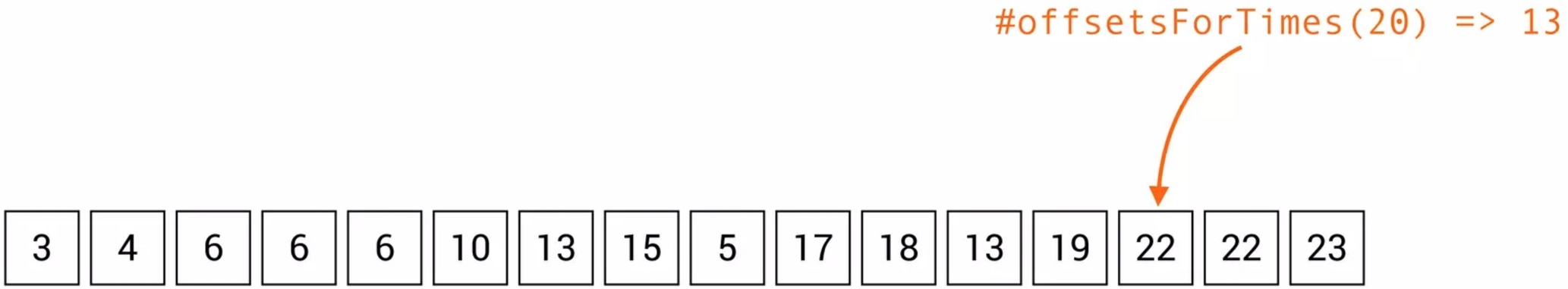
Timestamp Index (as of v0.10.1)

- KafkaConsumer#offsetsForTimes(…)



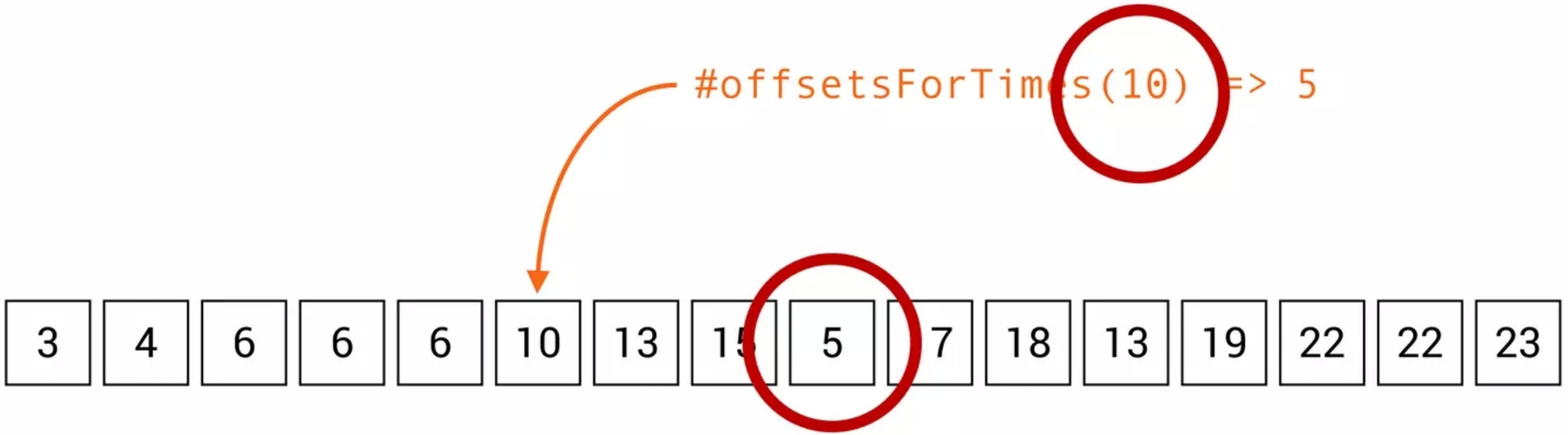
Timestamp Index (as of v0.10.1)

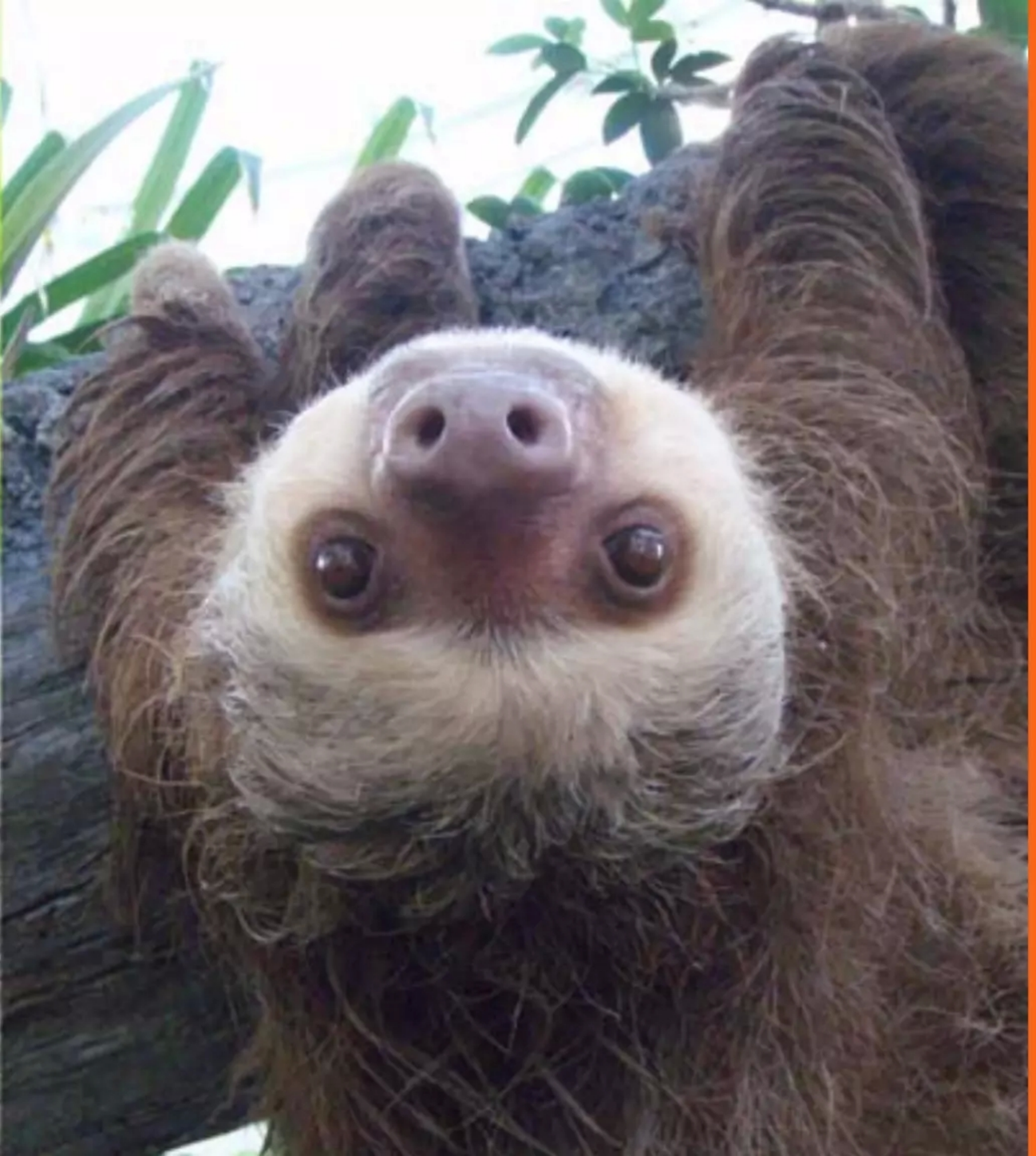
- KafkaConsumer#offsetsForTimes(…)



Timestamp Index (as of v0.10.1)

- KafkaConsumer#offsetsForTimes(…)





Processing Order

- Consumer returns messages in ***offset order*** per partition
- Multiple partitions:
  - Your code decides
  - Consider using `pause()/resume()`

Tracking Time in Kafka Streams

Event time is, when something happens in the real world.

Processing time is, when an event gets processed.

```
public interface TimestampExtractor {
    long extract(ConsumerRecord record, long partitionTime);
}
```

Record will be skipped/dropped if invalid (i.e., negative) timestamp is returned

Configuration parameter: `timestamp.extractor`

Available:

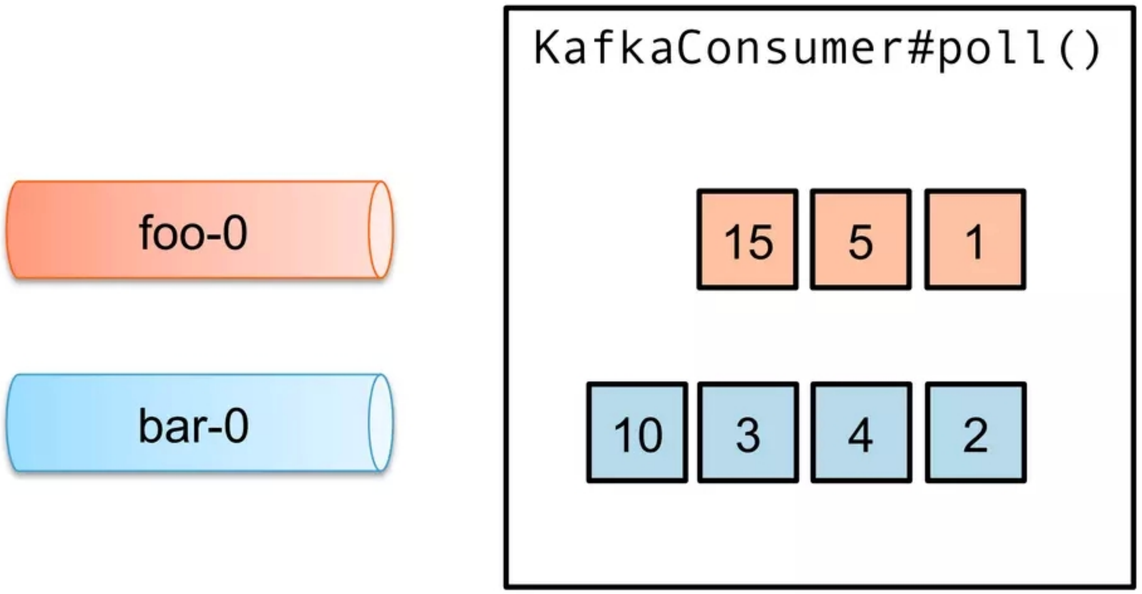
- `FailOnInvalidTimestamp` **(default)**
  - return record metadata timestamp (i.e., message timestamp)
- `WallclockTimestampExtractor`: returns current client system time
- *user defined* (e.g., extract a timestamp from the payload—key, value, or headers)

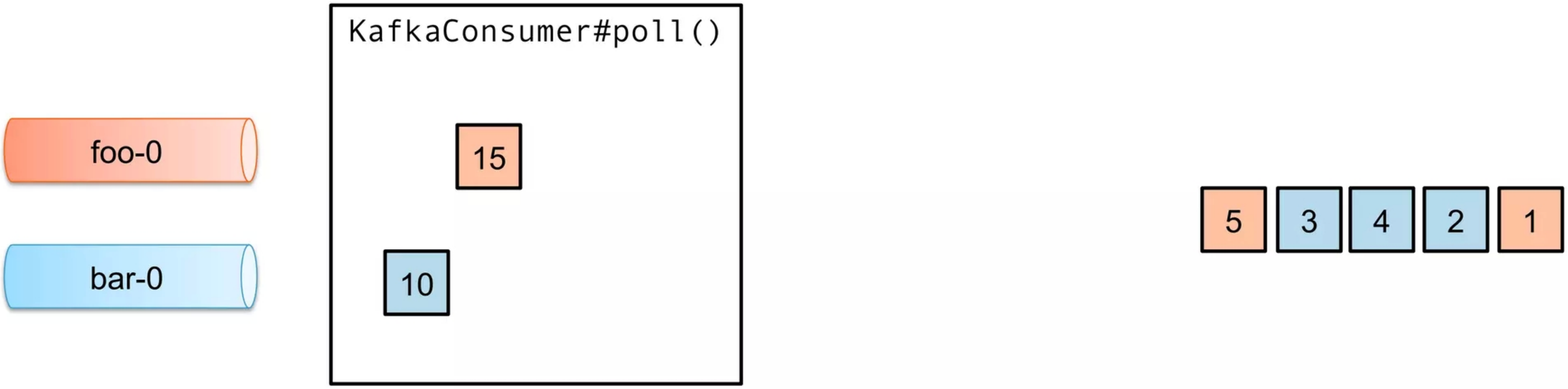
Partition Time (v2.1)

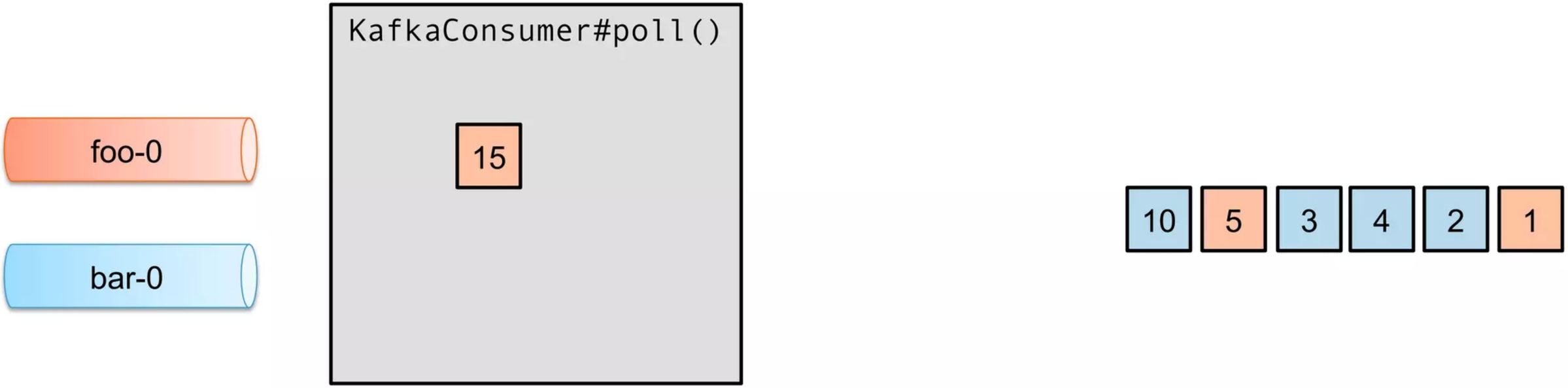
- maximum timestamp observed
- never goes backwards
- initially unknown
- preserved over restarts/rebalances (as of v2.4)

Processing Order

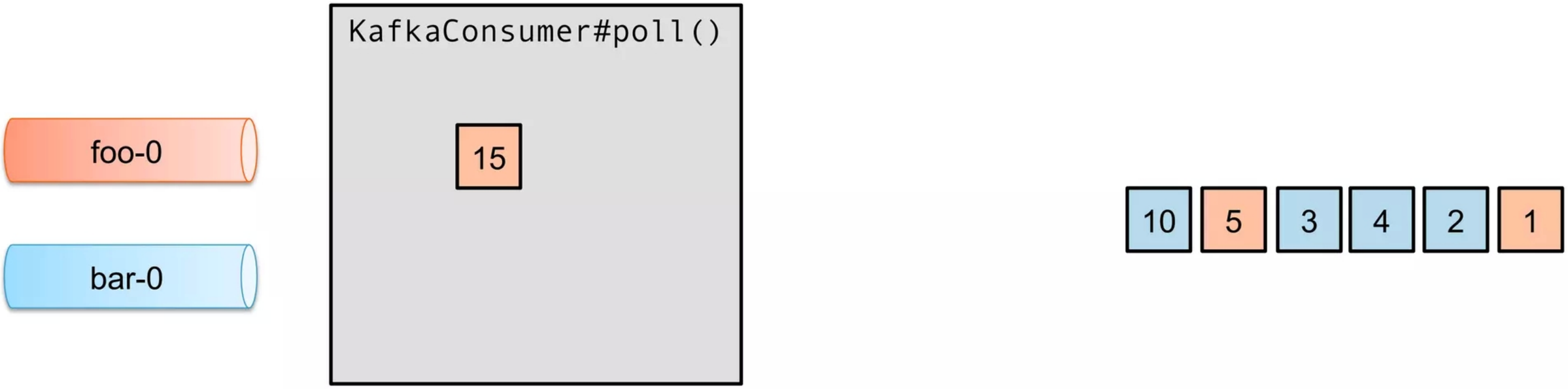
- Records are processed in ***offset order*** per partition (even if there is out-of-order records)
- Multiple partitions:
  - Smallest timestamp first
  - Only consider “head record” per partition
  - Interleaved


foo-0
bar-0
KafkaConsumer#poll()
15
5
1
10
3
4
2


KafkaConsumer#poll()
foo-0
bar-0
15
10
5
3
4
2
1



Don’t know next timestamp from bar-0.
Need to `poll()` before we can continue.



What happens of `poll()` does not return data?
Block until `max.task.idle.ms` pass.

```
public interface ProcessorContext {
    // other methods omitted
    long timestamp();
    void forward(K key, V value);
    void forward(K key, V value, To to);
    Cancelable schedule(Duration interval, PunctuationType t, Punctuator callback);
}
```

- `timestamp()` returns the timestamp of the currently processed record
- Output record(s) inherit timestamp of input record by default.
- Output record timestamp can be explicitly set via

```
context.forward(…, To.all().withTimestamp(…));
```

- Schedule regular callbacks, based on ***stream time*** or system time progress

Stream Time (v2.1)

- Tracked per task
- Maximum over all partition times
- Same properties as partition time

Wrapping Up

- Strict offset ordering
- Rich timestamps semantics
  - producer / broker, topic / consumer
- Watch your configs!

We are hiring!

@MatthiasJSax

matthias@confluent.io | mjsax@apache.org