AVSystem

Achieving prod-ready streaming pipelines with Kafka Streams

KSUG meetup • 28.05.19

AVSystem

Agenda

- Kafka introduction
- Kafka Streams API
- Live coding

Disclaimer

!=

AVSYSTEM

In case of emergency

AVSYSTEM

Let’s talk about

AVSYSTEM

Apache Kafka

- Initially conceived as a messaging queue
- Distributed commit log
- Distributed event-streaming platform

AVSystem

So... what is it?

- **A messaging system** (queueing, pub-sub)
- **A storage system** (high-perf, low-latency, replication)
- **A streaming platform**

Kafka basics

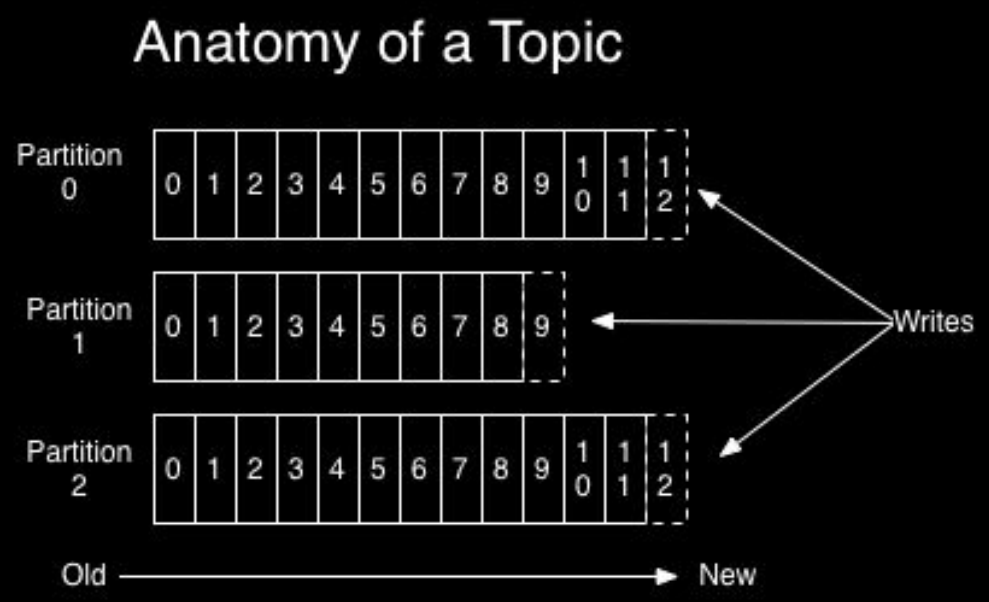
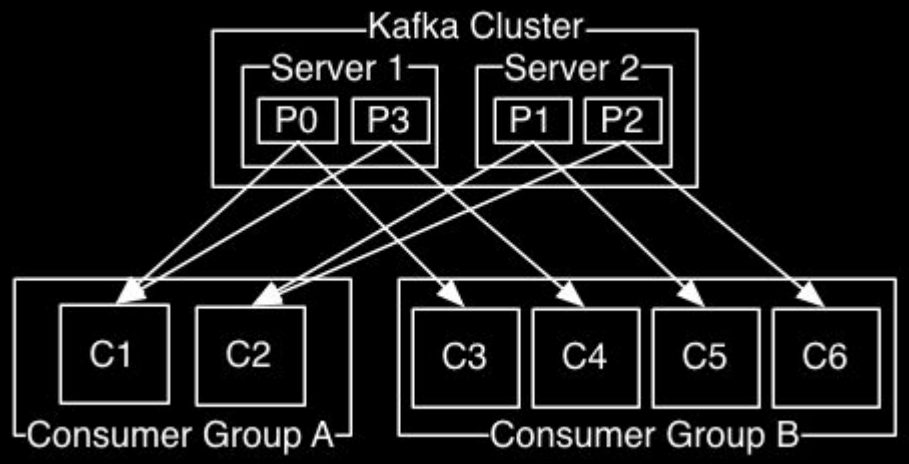


Image sources:
https://kafka.apache.org/



- Any partition is consumed by one and only one member of the same consumer group
- Any partition is consumed by all of the present consumer groups

Image source:
https://kafka.apache.org/

AVSYSTEM

Implications

- Every record published to a topic is delivered to all of the subscribed consumer groups.
- Each consumer **in a single** consumer group => effective **load balancing** over all consumers.
- Each consumer **in its own** consumer group => effective **broadcast** over all consumers.

AVSYSTEM

Let's talk about

Kafka Streams API

AVSYSTEM

Kafka Streams

- Lightweight Java library
- Enables transforming and enriching data from the Kafka topics in a streaming fashion
- No separate cluster required

AVSYSTEM

- Code-centred (Scala API)
- Elastic, highly scalable, fault-tolerant
- Deploy to containers, VMs, bare metal, cloud
- Equally viable for small, medium, & large use cases
- Comes with all the Kafka goodies (partition-scalability , Kafka Security, exactly-once semantics).

Enter the DSL

Transformations:

- Stateless (`filter`, `map`, `flatMap`, `foreach`, `branch`, etc...)
- Stateful (`count`, `reduce`, `aggregate`, `join`, `windowed` ops)

AVSystem

Stateless operators

Transform 1 message into 0 or more messages:

- 1:1 - `map`, `branch`
- 1:[0,1] - `filter`
- 1:[0,...,n] - `flatMap`

AVSYSTEM

An example (Word Count)

```
builder.stream[String, String]("streams-file-input")
  .flatMapValues { value => value.toLowerCase().split(" ") }
  .map { (_, value) => (value, value) }
  .groupByKey
  .count("Counts")
  .to("streams-wordcount-output")

val streams = new KafkaStreams(builder, props)
streams.start()
```

AVSYSTEM

Repartitioning

- Some of the operators may change the key of the transformed message(s). That may cause the creation of an intermediate repartition-topic.
- Operators indicating no key-change:
  - `filter`, `mapValues`, `flatMapValues`

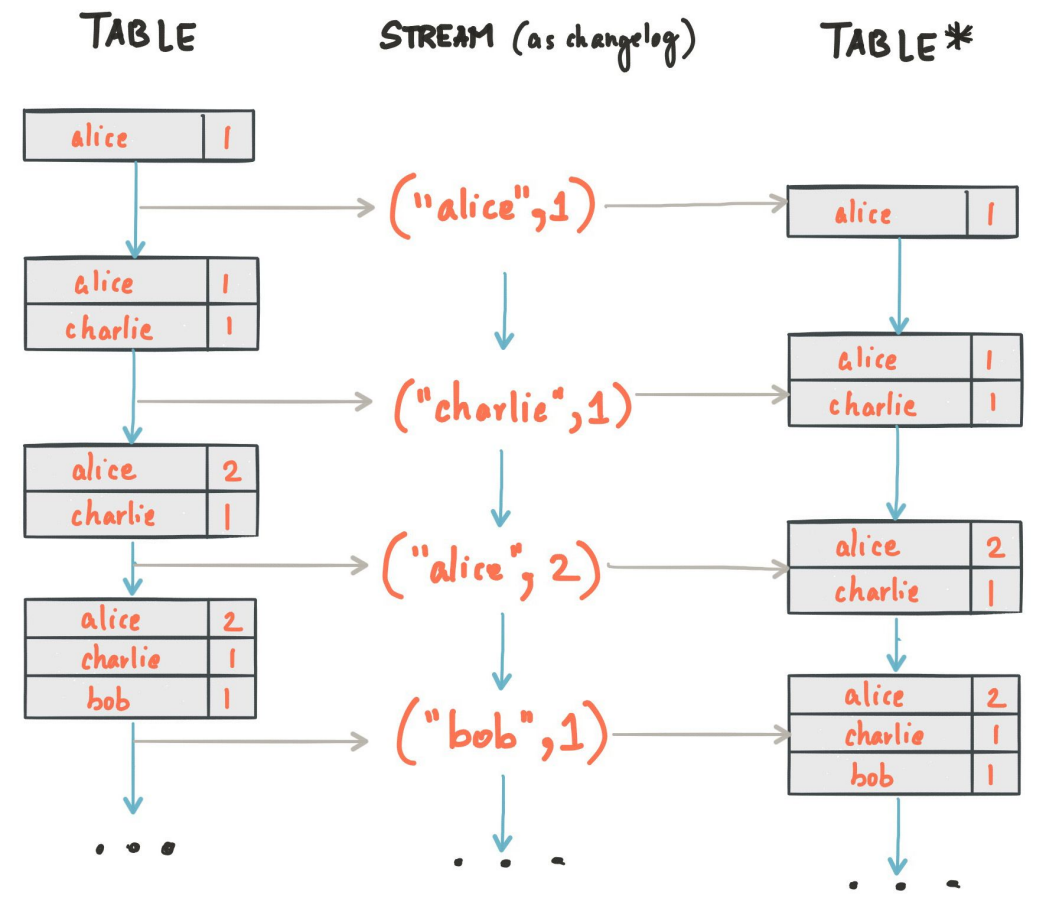
AVSYSTEM

Duality of Streams and Tables

- Stream processing usually needs both streams and databases
- A stream may be seen as a changelog of a table
- Table may be seen as a snapshot of the latest value for each key in a stream.

AVSYSTEM

Duality of Streams and Tables



avsystem

KStreams and KTables

- **KStream** is an abstraction of a **record** stream
  - log compaction may remove important data
- **KTable** is an abstraction of a **changelog** stream
  - only latest value is significant
  - log compaction highly encouraged
  - table-lookup via joins

AVSYSTEM

Joins

Problem:

Process records only with the specific key.

What if the key is dynamically defined?

What if the key is defined on another topic?

Example:

Stream (main) record structure:
(key: *PersonID*, value: *ImportantData*)

Other (configuration) topic:
(key: *PersonID*, value: *Boolean*)

AVSYSTEM

Solution:

- KStream-to-KTable join (inner join or left join)

Different types of joins:

- KTable-to-KTable (inner join, left join, outer join)

- KStream-to-KStream (same, but only windowed)

AVSystem

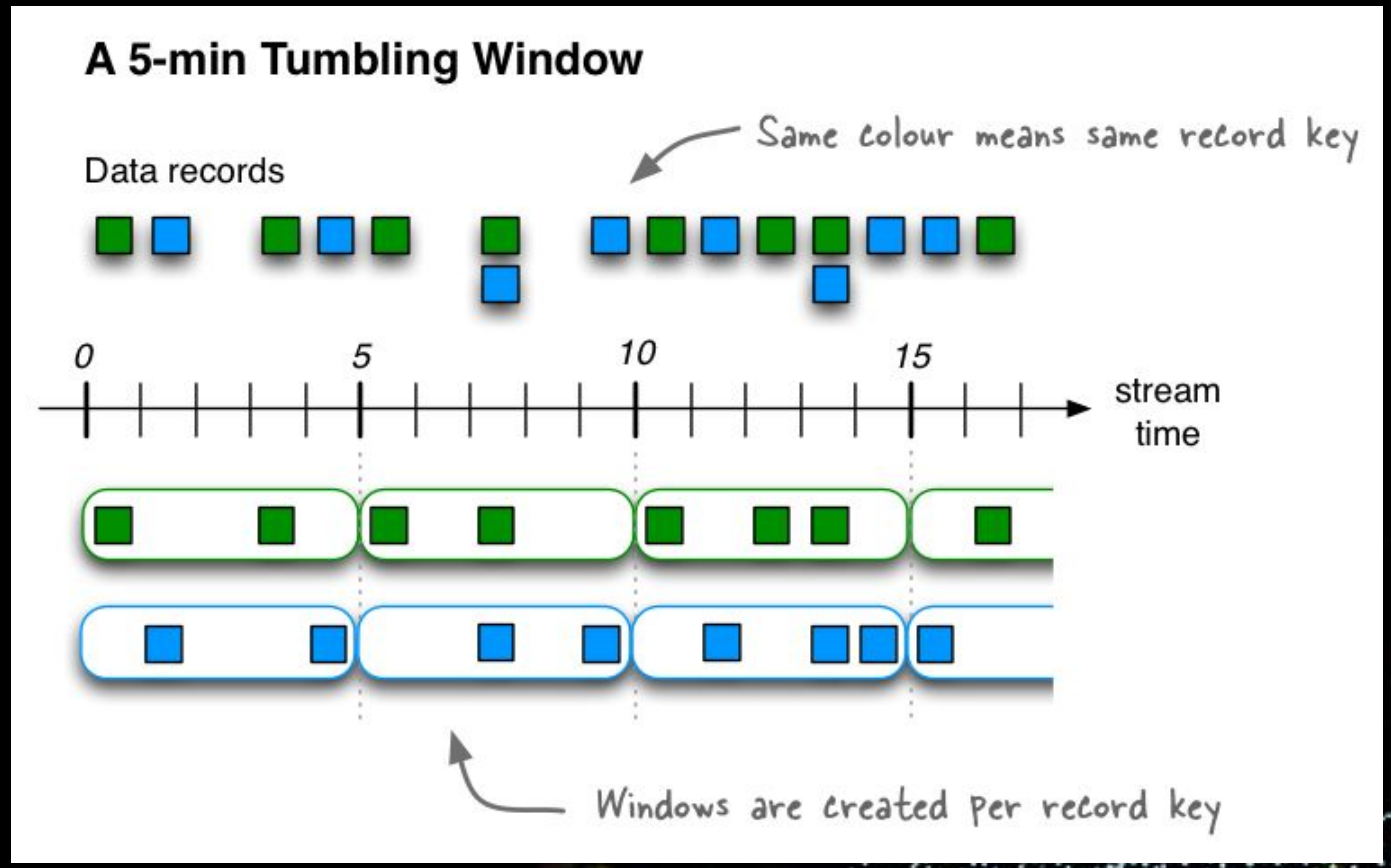
Aggregation

- Transforms N (grouped) messages into 1 message
- Key-based operation, records need to be grouped
  - only grouping by key causes no repartitioning
- May be both windowed and non-windowed
- 3 types of aggregate functions:
  - count, reduce, aggregate

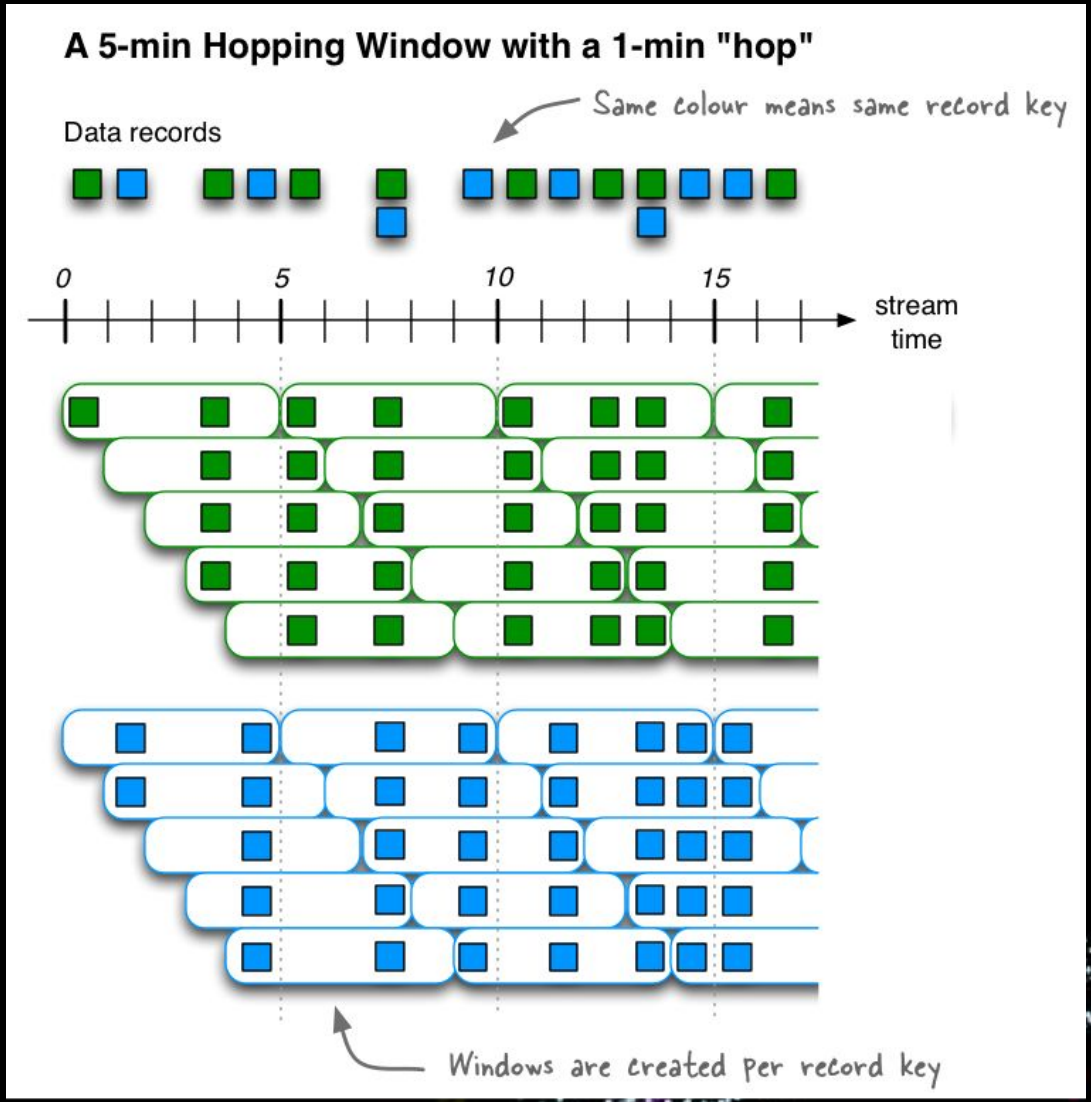
AVSYSTEM

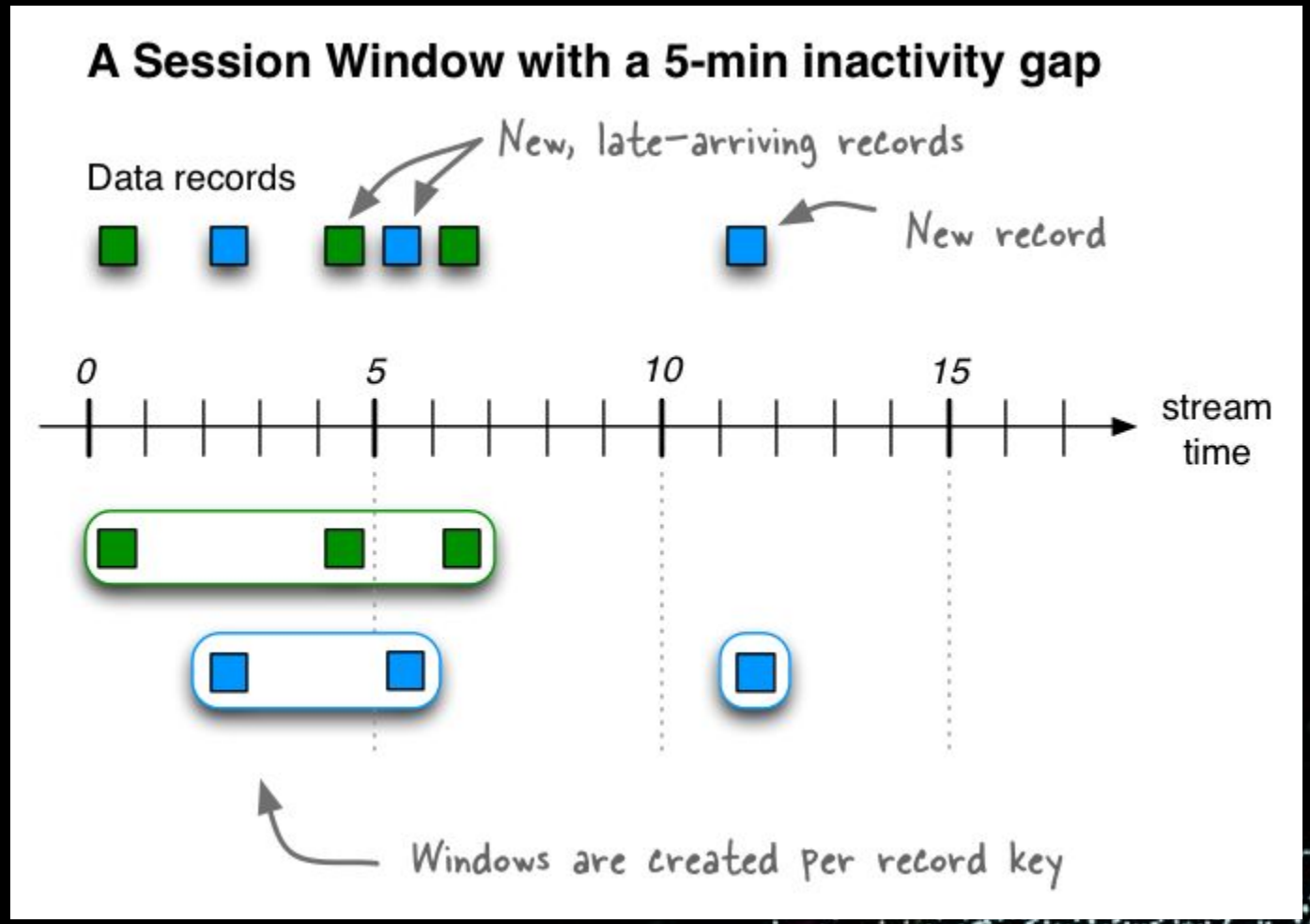
Windowing

- Different types of windows:
  - *tumbling window*, *hopping window* (windowed aggregates)
  - *sliding window* (windowed joins)
  - *session window* (session aggregates, e.g. user activities)


A 5-min Tumbling Window
Same colour means same record key
Data records
0
5
10
15
stream time
Windows are created per record key

AVSYSTEM


A 5-min Hopping Window with a 1-min "hop"
Same colour means same record key
Data records
0
5
10
15
stream time
Windows are created per record key


A Session Window with a 5-min inactivity gap
Data records
New, late-arriving records
New record
0
5
10
15
stream time
Windows are created per record key

AVSYSTEM

Time semantics

3 types of time in terms of stream processing:

- **Event time** (real event timestamp at “source”)
- **Ingestion-time** (when event was produced to Kafka)
- **Processing-time** (when event was processed)

AVSystem

Windowed time semantics

3 types of window-time configuration:

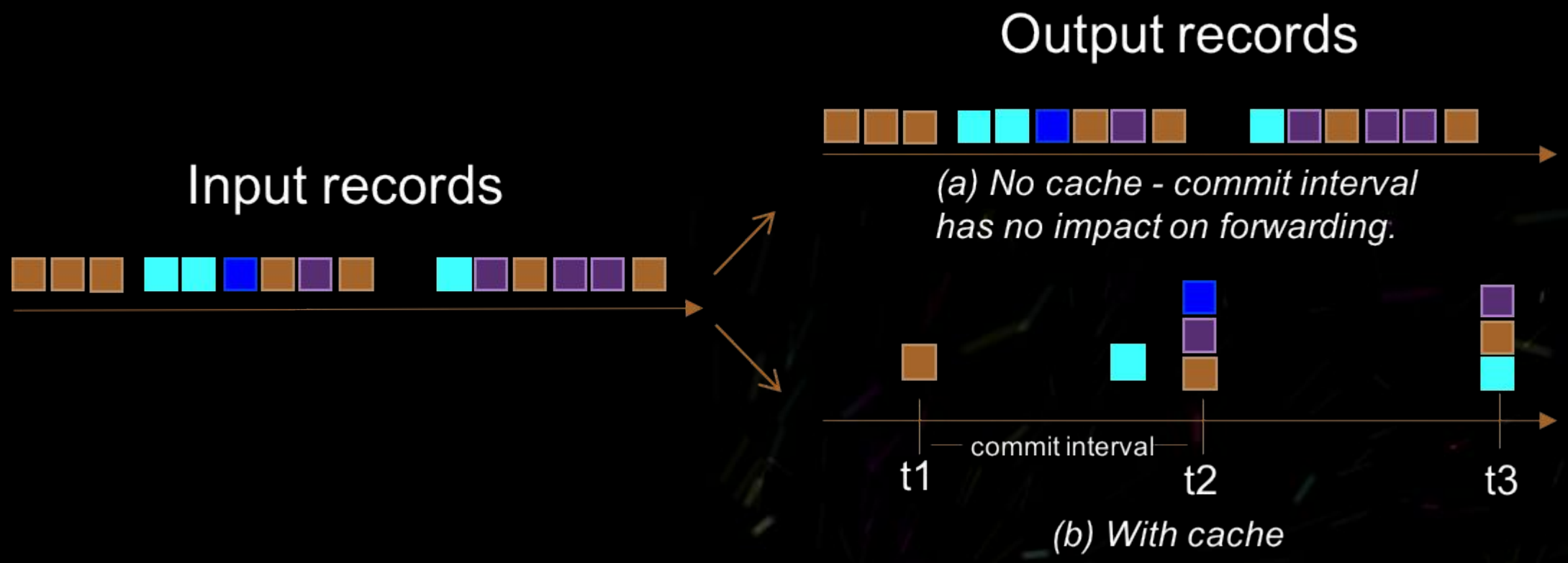
- **Window duration** (size of the window)
- **Window interval** (frequency of subsequent windows)
- **Grace period** (actual size of the window, when late records are still accepted)

AVSystem

When exactly are the windowing results produced?

- After every update? (yes, when the cache is disabled)
- Whenever the earliest of *commit.interval.ms* or *cache.max.bytes.buffering* hits.

AVSYSTEM


Input records
Output records
(a) No cache - commit interval has no impact on forwarding.
commit interval
t1
t2
t3
(b) With cache


gifbin.com

AVSYSTEM

Suppression (Kafka 2.1.0)

- Final windowing results produced after the grace period (finally).

https://www.confluent.io/blog/kafka-streams-take-on-watermarks-and-triggers

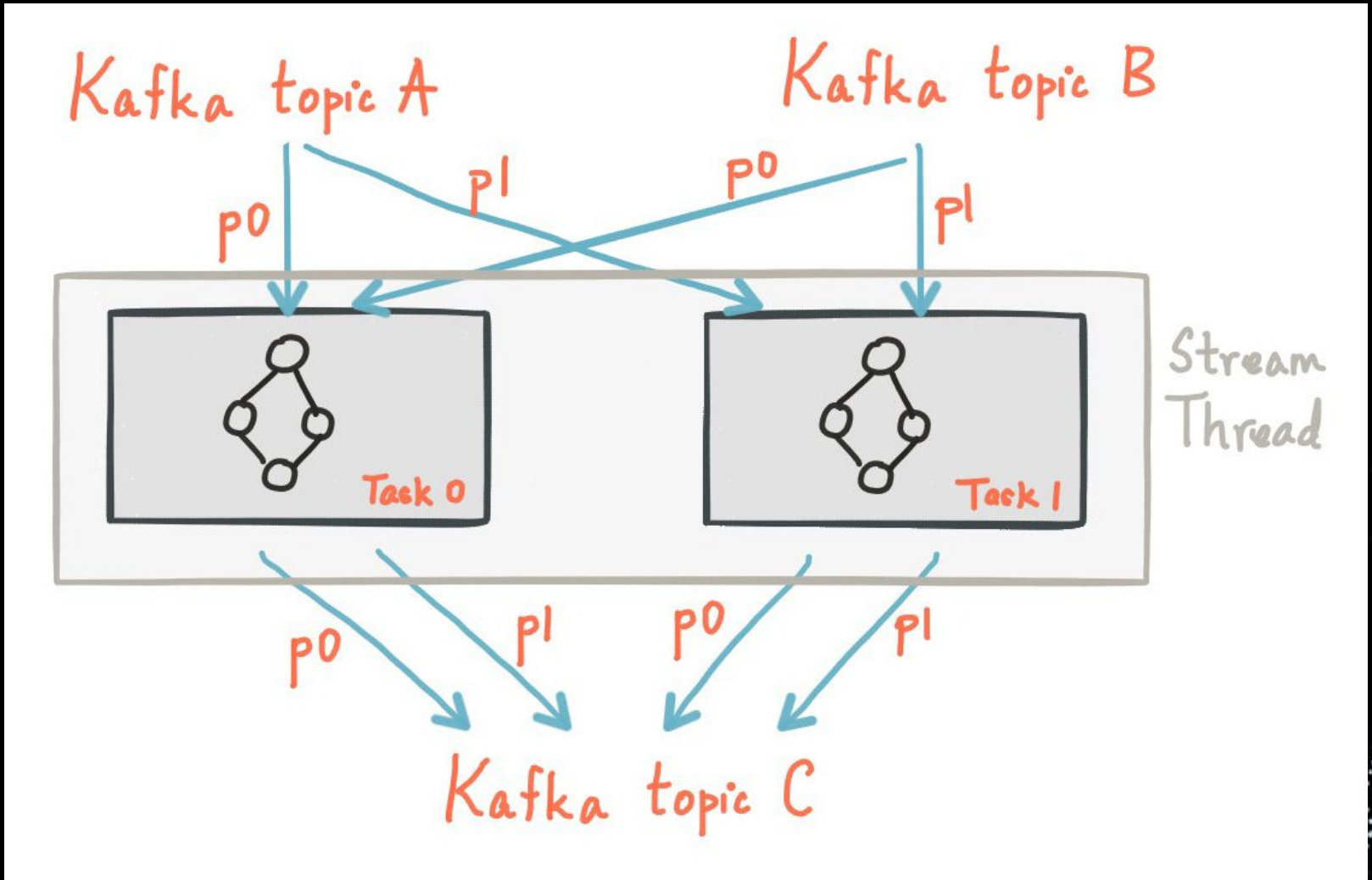
```
events
  .groupByKey()
  .windowedBy(
    TimeWindows.of(Duration.ofMinutes(2).withGrace(Duration.ofMinutes(2))
  )
  .count(Materialized.as("count-metric"))
  .suppress(Suppressed.untilWindowClose(BufferConfig.unbounded()))
```

https://www.confluent.io/blog/kafka-streams-take-on-watermarks-and-triggers

AVSYSTEM

Threading model

- **Stream partition** is a totally ordered sequence of data records and maps to a Kafka topic partition.
- **Stream task** is a logical unit of parallelism model - an application's topology is scaled by breaking it into multiple stream tasks.
- **Streaming thread** is just a thread and may execute many stream tasks.


Kafka topic A
Kafka topic B
p0
p1
p0
p1
Task 0
Task 1
Stream Thread
p0
p1
p0
p1
Kafka topic C

AVSYSTEM

Stores

- Used to store and query data (for stateful operations)
- Automatically recoverable from changelog topics
- Implementations to choose:
  - RocksDB (by default), in-memory hashmap,

    or anything you implement

AVSYSTEM

Testing

Two options:

- TopologyTestDriver.java (comes with Kafka dependencies, constantly updated, rich API, Java :<)
- MockedStreams.scala (really lightweight Scala wrapper for the TopologyTestDriver)

```
import com.madewithtea.mockedstreams.MockedStreams

val input = Seq(("x", "v1"), ("y", "v2"))
val exp = Seq(("x", "V1"), ("y", "V2"))
val strings = Serdes.String()

MockedStreams()
  .topology { builder => builder.stream(...) [...] } // Scala DSL
  .input("topic-in", strings, strings, input)
  .output("topic-out", strings, strings, exp.size) shouldEqual exp
```

AVSystem

Live coding / demo time

More?

Repo: https://github.com/mowczare/kafka-streams-scala

Blog: https://www.avsystem.com/blog/large-scale-data-monitoring-with-kafka-streams/

Careers: https://www.workwiththebest.pl

Thanks for the attention!