Tecnologías de datos masivos

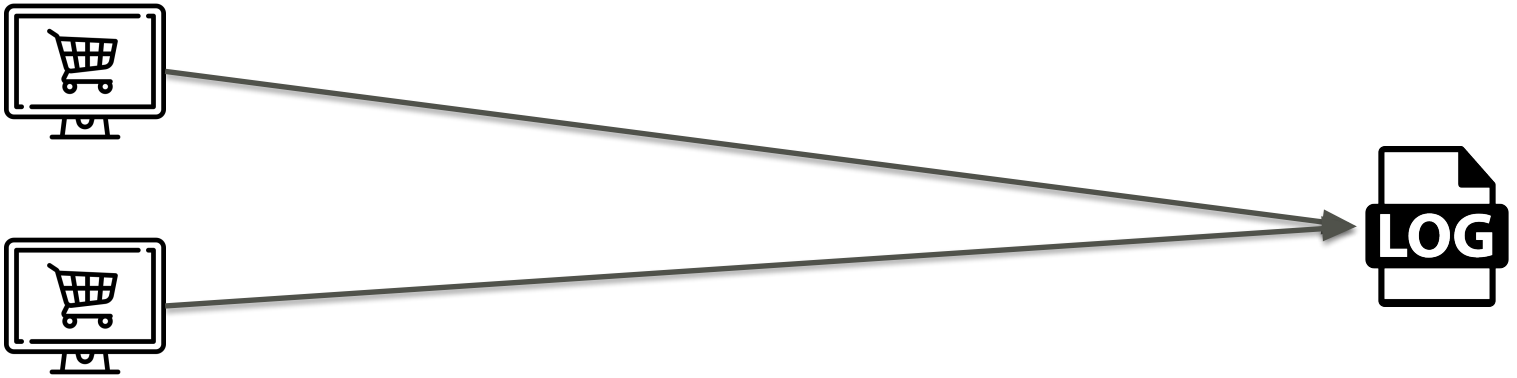
Doble Grado en Ingeniería en Tecnologías de Telecomunicación y Business Analytics

Apache Kafka

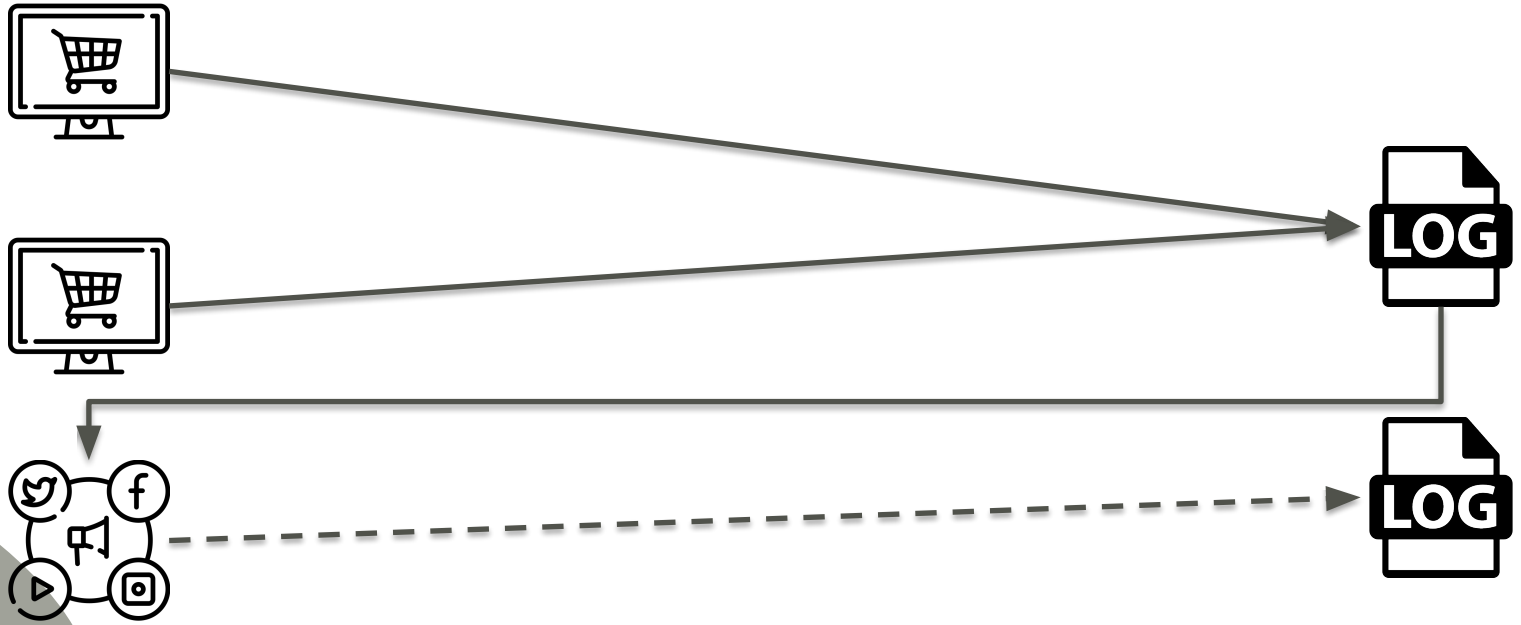
Apache Kafka - ¿por qué nace?

- Todas las aplicaciones informáticas registran su actividad en archivos o base de datos de “log”
- En un principio cada parte de una arquitectura de Software se encargaba de conectarse a sus propios sistemas de log y lo gestiona sin tener en cuenta los demás sistemas
- Con el paso del tiempo otro equipo se da cuenta que podría utilizar dichas métricas para mejorar su parte del sistema
- Conectar distintas parte de una arquitectura hacía que se tuvieran que hacer distintas modificaciones para que ese nuevo sistema pudiera utilizar estas métricas
- Cuando un nuevo equipo requería estas métricas se tenía que adaptar otra vez a este sistema y hacía que los desarrollos fueran complejos y difíciles

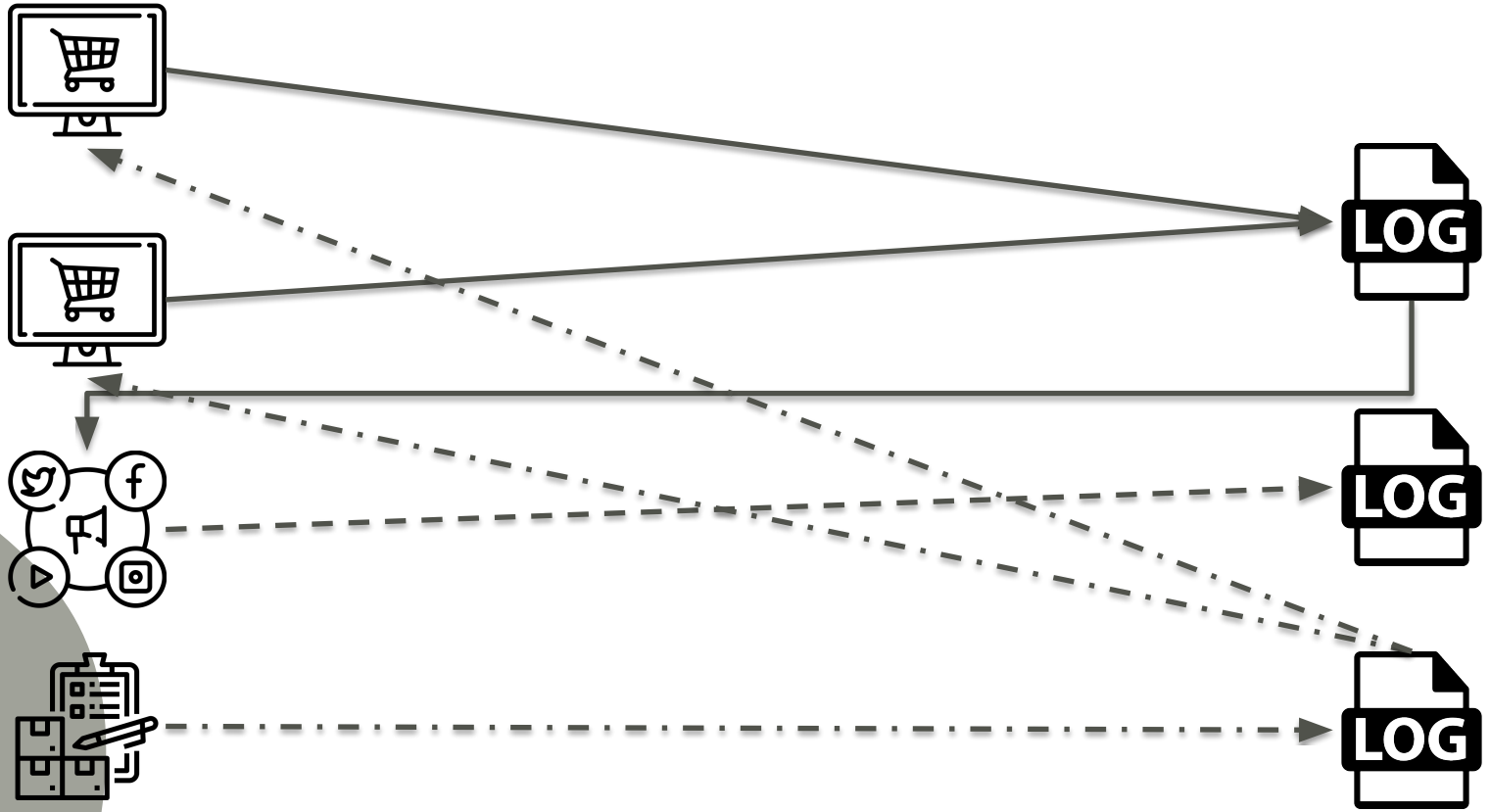
Apache Kafka - ¿por qué nace?



Apache Kafka - ¿por qué nace?



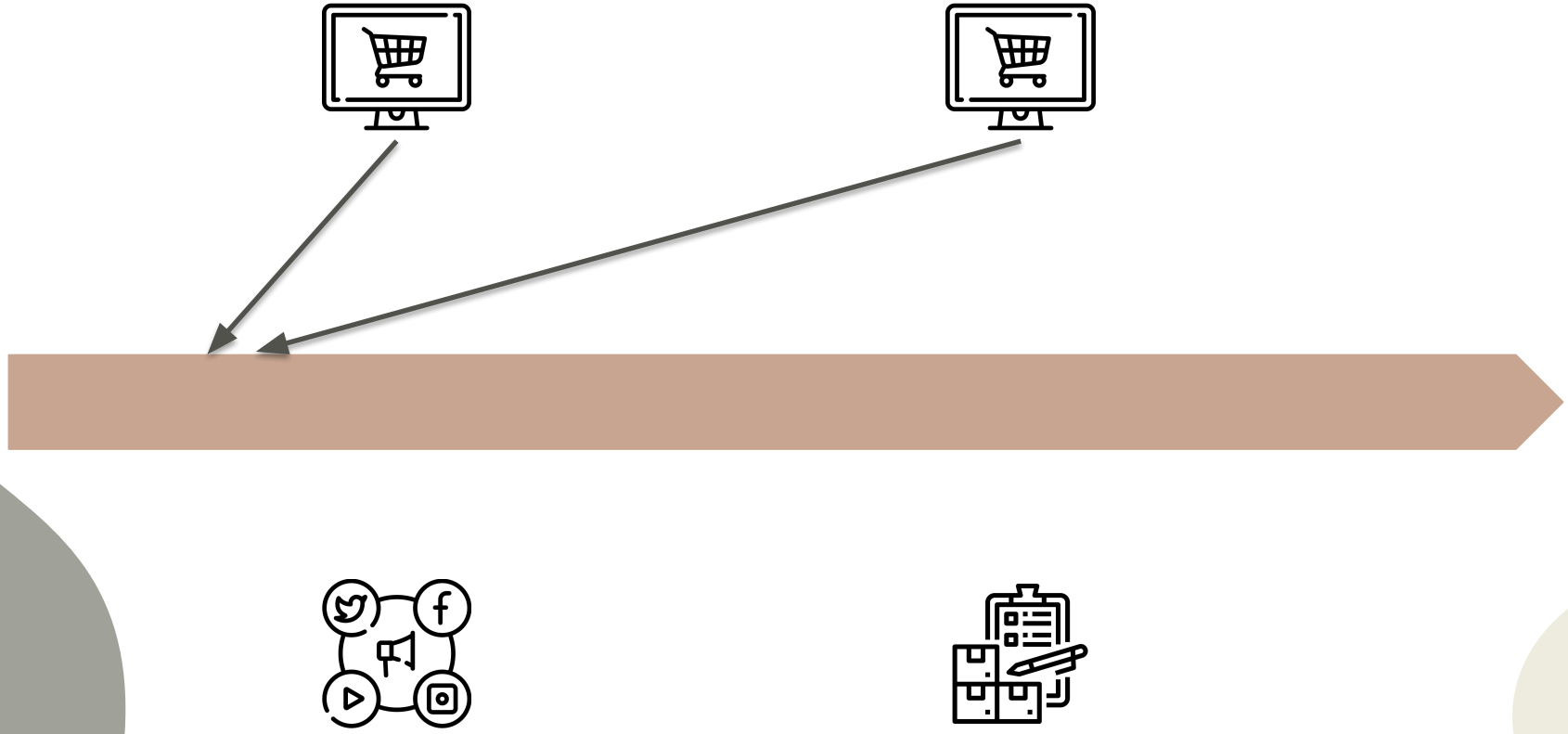
Apache Kafka - ¿por qué nace?



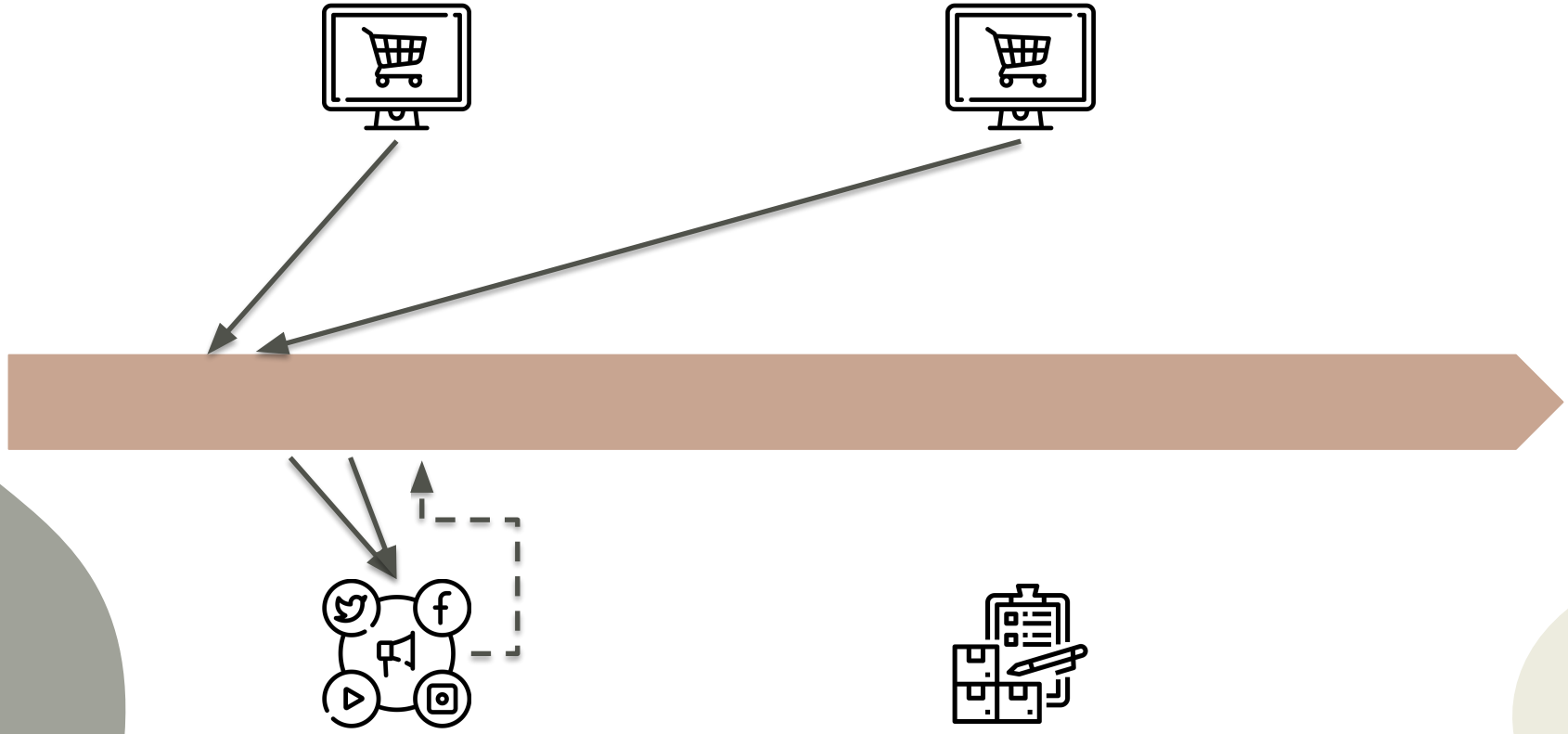
Apache Kafka - ¿por qué nace?

- Los “logs” de los distintos sistemas son mensajes en los que el sistema guarda información de su estado
- Distintos sistemas podrían usar la información de otros sistemas si estos mensajes son legibles para todos.
- Para ello bastaría con que todos los sistemas pudieran escribir en un lugar común y un formato que sea interpretable por todos los servicios
- Los “log” serán escritos en orden de llegada y además podrían ser leídos en el mismo orden
- Al estar almacenados estos “logs” son consumidos pero siguen disponibles para que otro proceso los consuma

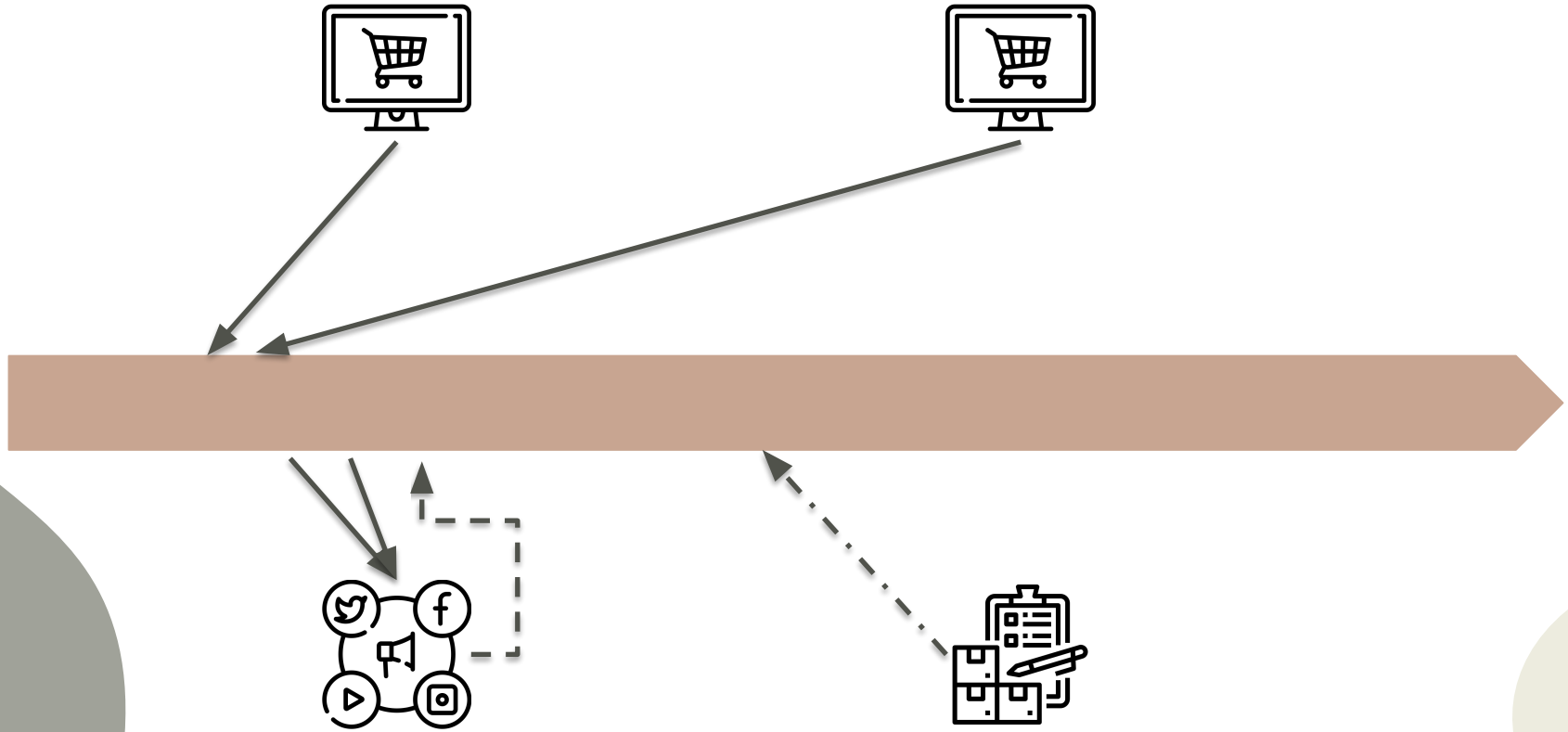
Apache Kafka - ¿por qué nace?



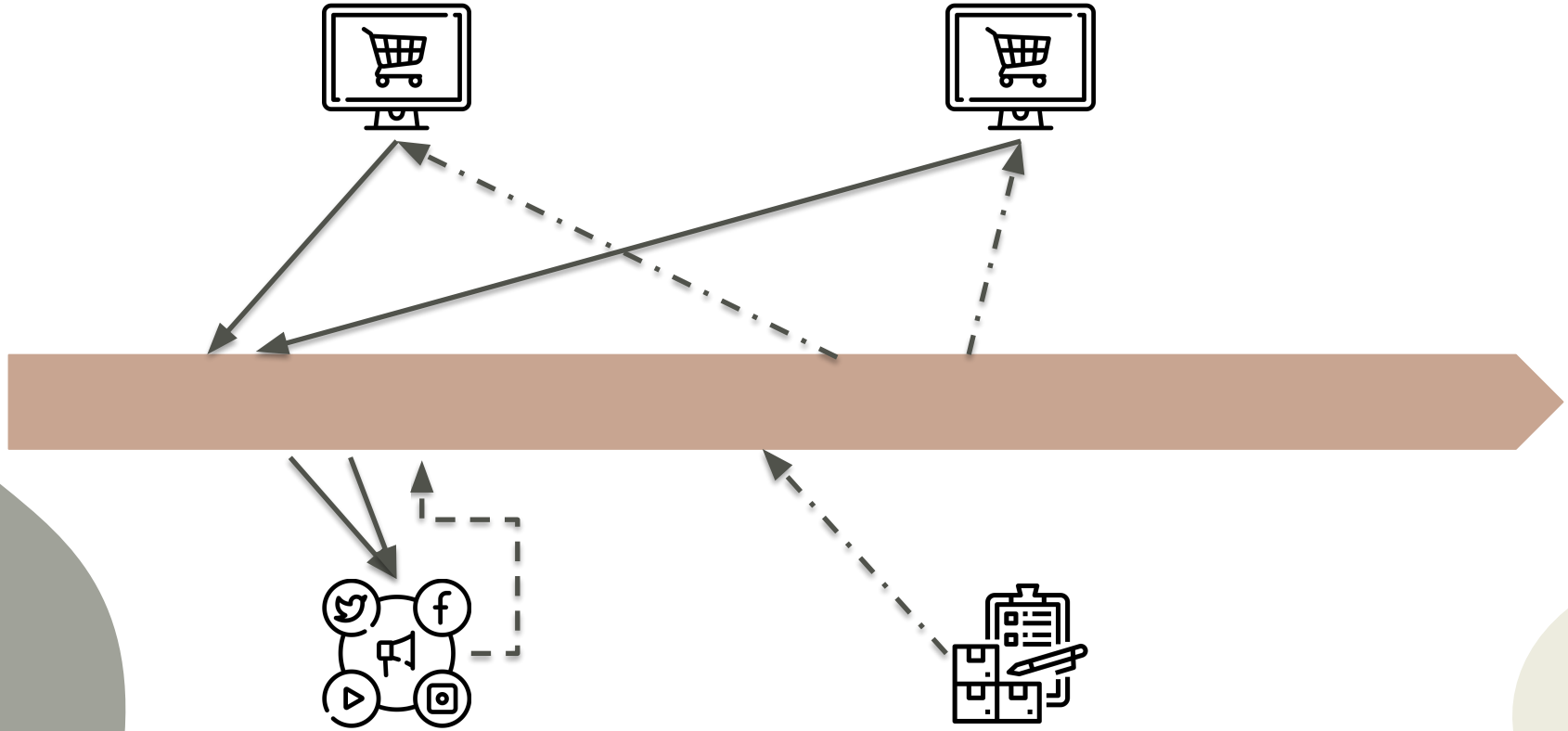
Apache Kafka - ¿por qué nace?



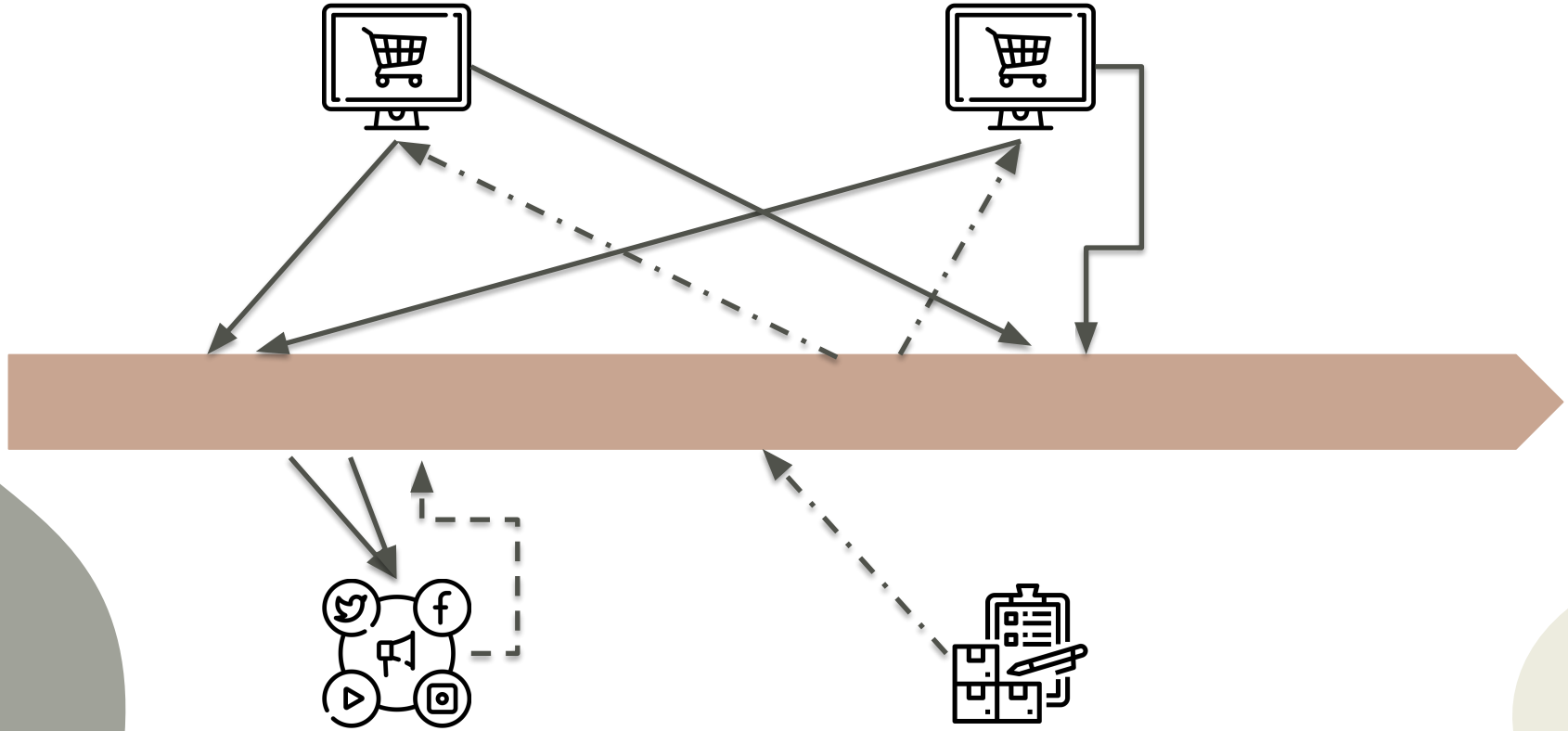
Apache Kafka - ¿por qué nace?



Apache Kafka - ¿por qué nace?



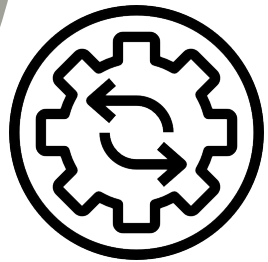
Apache Kafka - ¿por qué nace?



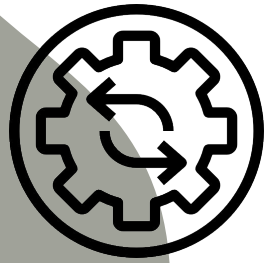
Apache Kafka - Arquitectura

- Apache Kafka sigue una arquitectura distribuida horizontalmente escalable del tipo peer to peer
- Los componentes de una arquitectura de Kafka se llaman Brokers
- Además de los brokers Kafka necesita de **Productores** y **Consumidores** para emitir y recibir mensajes
- En las primeras versiones de Kafka se apoyaba en Zookeeper para almacenar los mensajes de los topics
- A partir de su versión 3.0 ya elimina su dependencia de Zookeeper sustituyéndolo por su protocolo propio

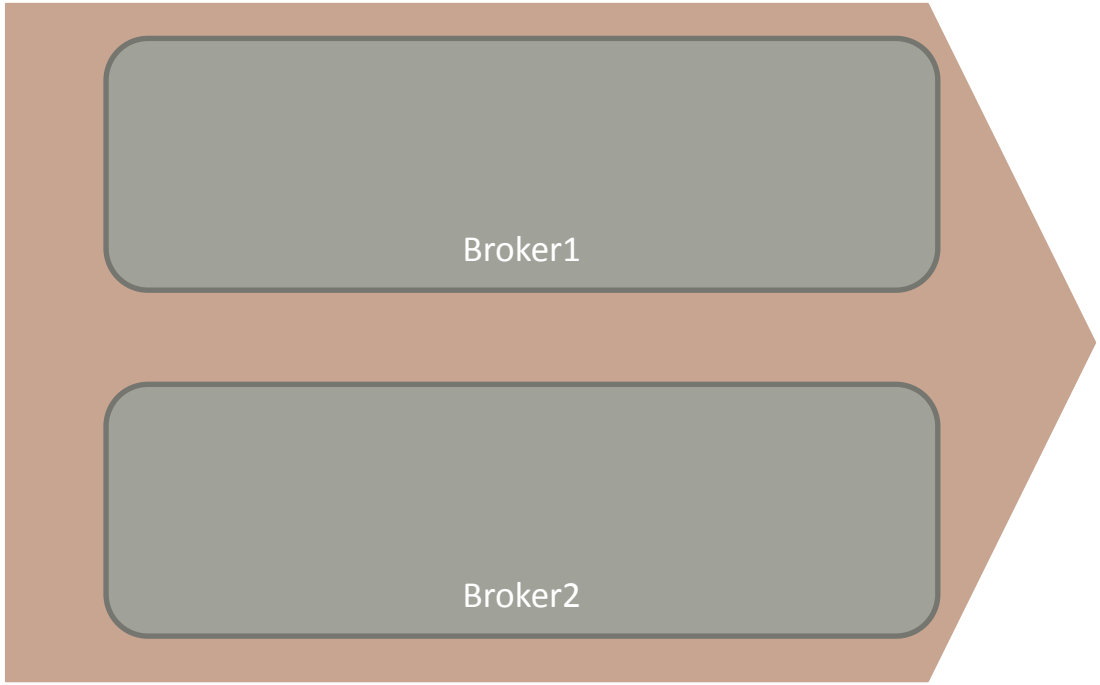
Apache Kafka - Arquitectura

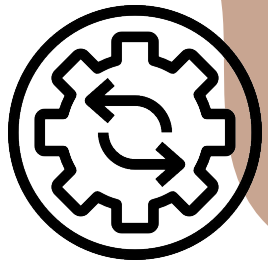


Producer-1

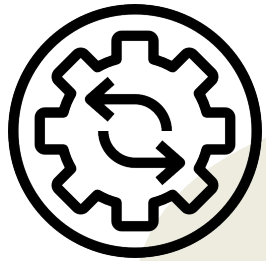


Producer-2





Consumer-1

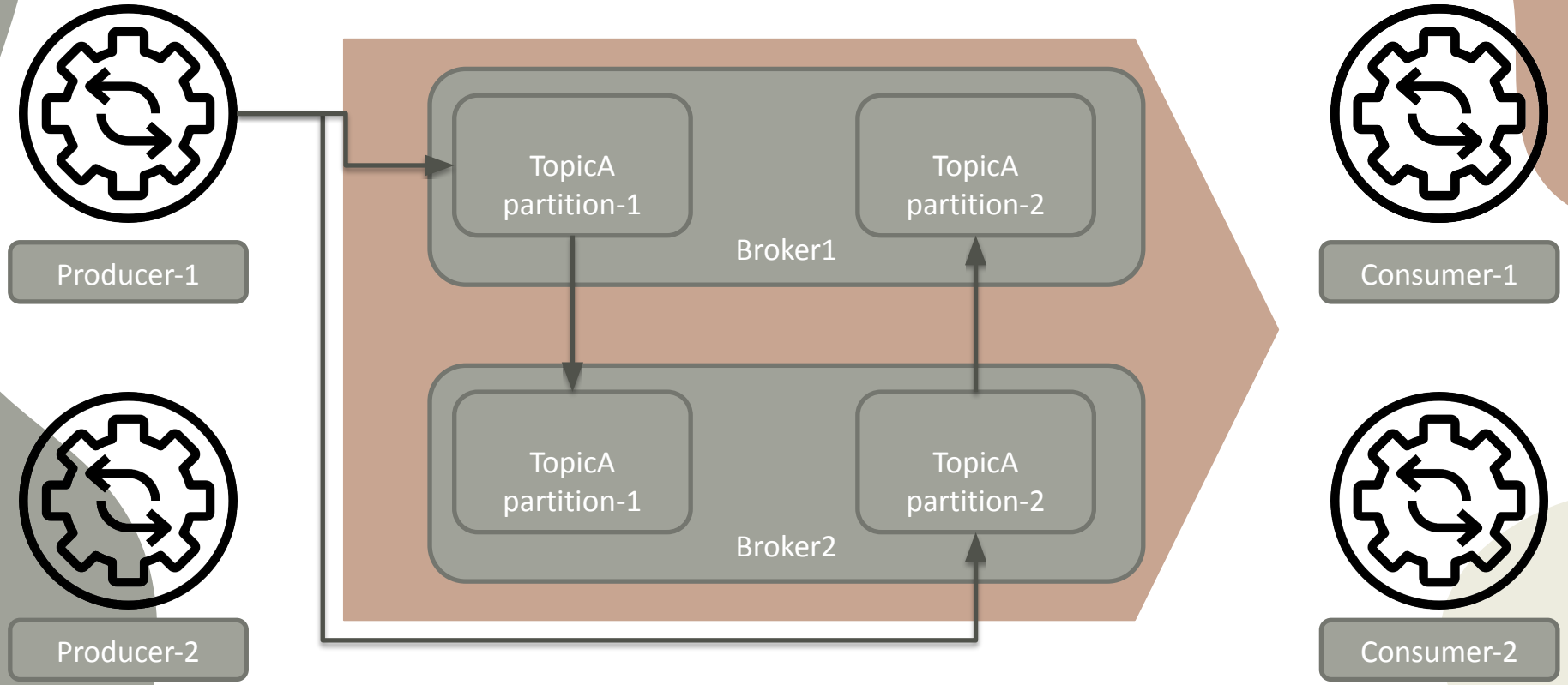


Consumer-2

Apache Kafka - Arquitectura

- Cada topic está particionado en distintas particiones
- Una partición está asociada a un broker
- Kafka es una solución horizontalmente escalable
    - Si una partición está asociada a un único broker y este se cae la información de su partición se pierde
    - Para evitar esto cada partición de Kafka es replicada en otros broker
    - Cada particion de un topic tiene un broker lider y el resto de brokers donde se replica se les conoce como brokers réplicas

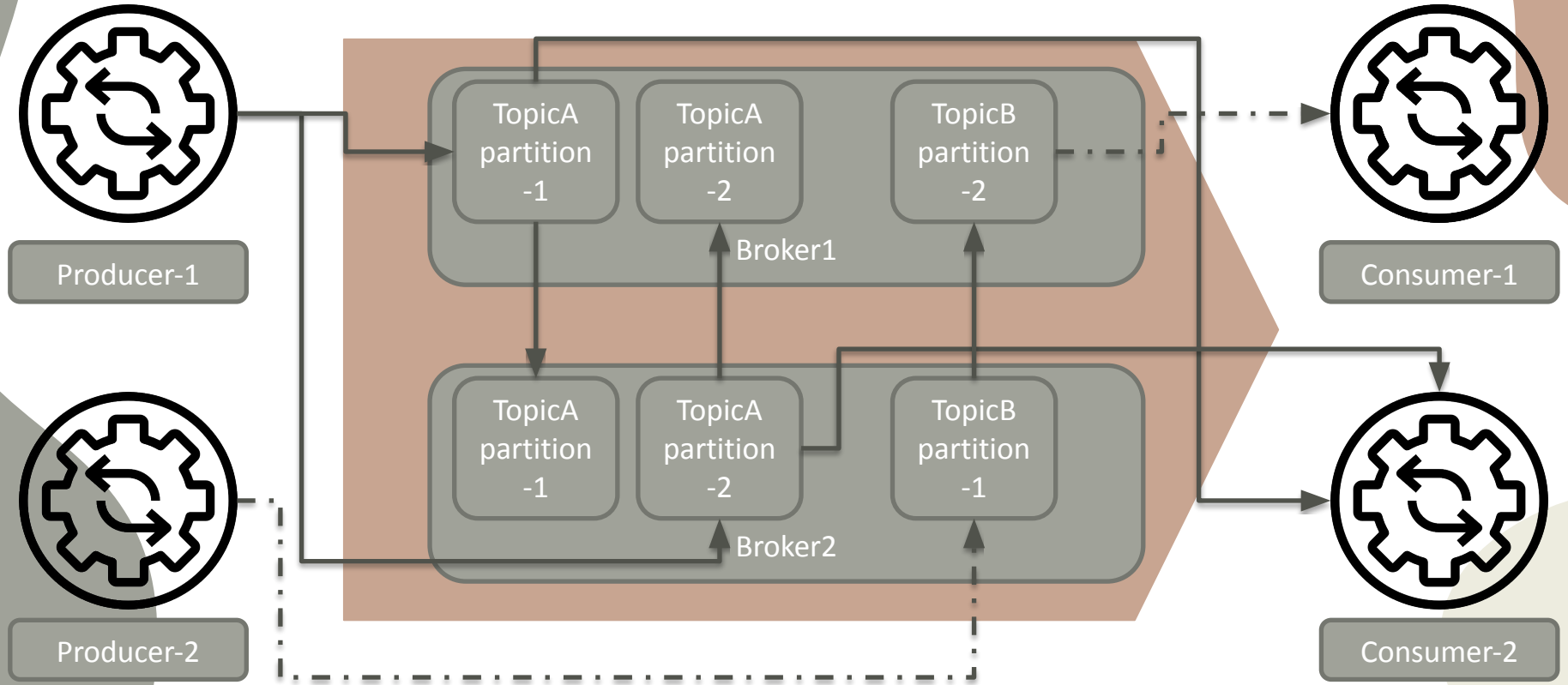
Apache Kafka - Arquitectura



Apache Kafka - Arquitectura

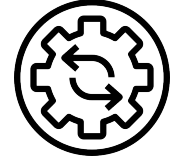
- Su unidad mínima de comunicación es el mensaje.
    - Un mensaje es un array de bytes, por lo que se puede enviar cualquier cualquier objeto que sea **serializable**
    - Los dos tipos más habituales son los JSON y Apache Avro
    - Se asocia una clave a cada mensaje mediante la cual se enviará dicho mensaje a una partición
    - Un mensaje es enviado por un **Producer** *en tiempo real* asociandole un Offset
    - Los mensajes son leídos por uno o varios **Consumer** tanto en *streaming* como leyendo a partir de un Offset concreto
    - Los mensajes nunca pueden superar un tamaño máximo (por defecto 1Mb)

Apache Kafka - Arquitectura

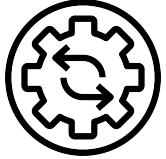


Apache Kafka - Escrituras y lecturas

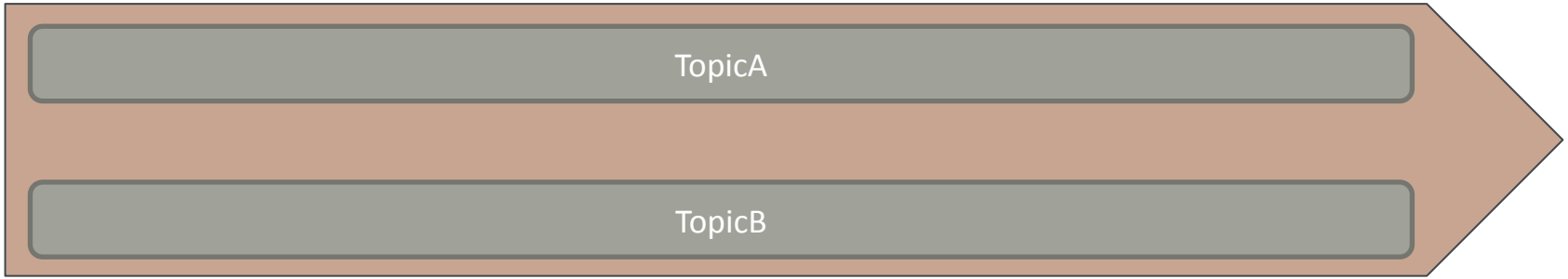
- Un mensaje es producido por un único Producer y consumido por uno o varios Consumers
- El consumer consume **varios mensajes** de uno o varios topics
- Los consumers se agrupan en consumers_group
- Se define un máximo de tiempo entre dos peticiones de recolección de eventos
- A diferencia de otras colas de mensajería Kafka nos permite reprocesar los mensajes en orden a pasado



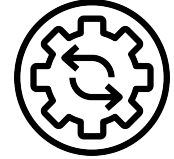
P1



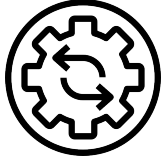
P2



Creamos dos productores que escribirán cada uno en un topic distinto.



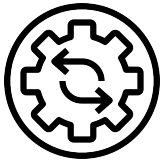
P1



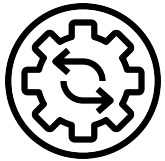
P2

TopicA

TopicB

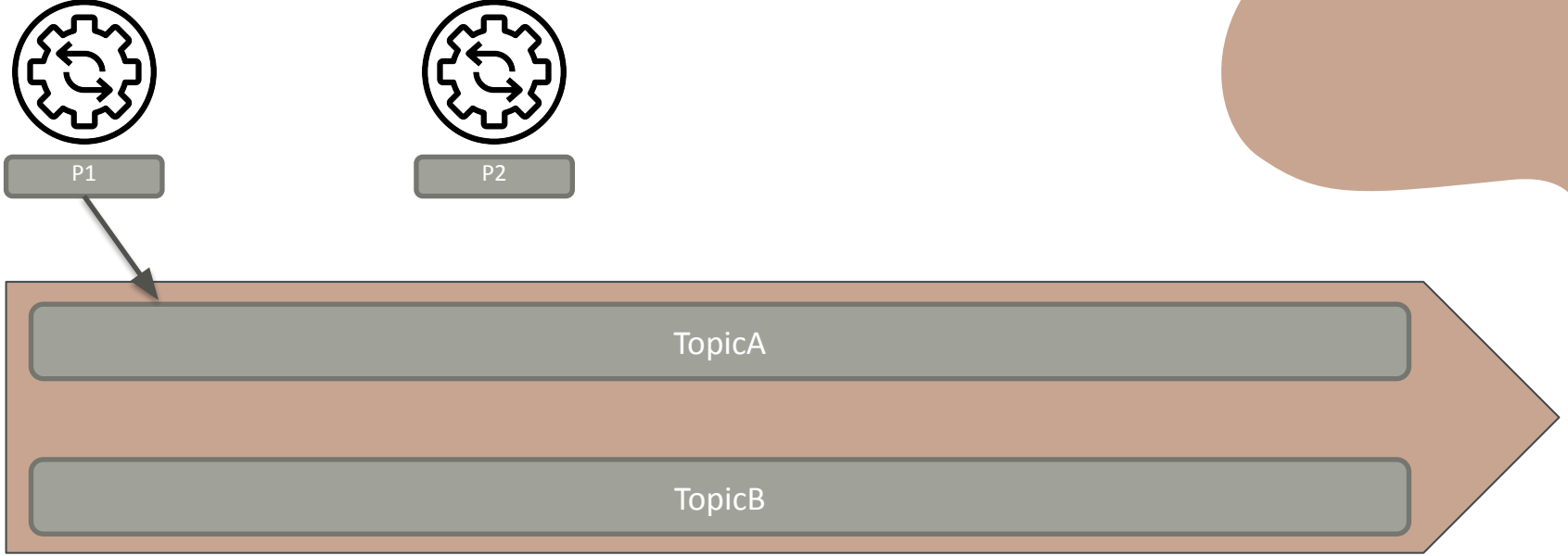


C1

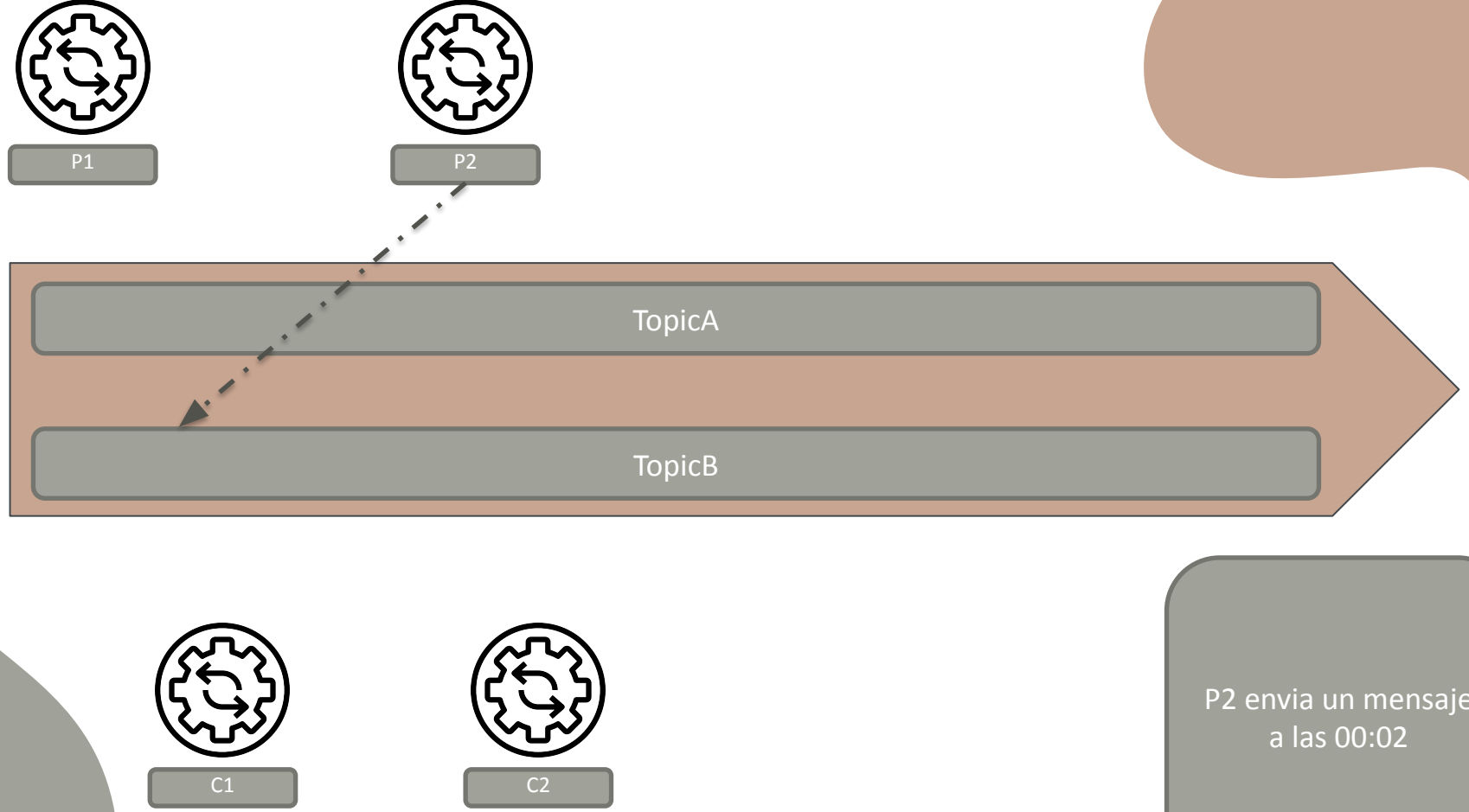


C2

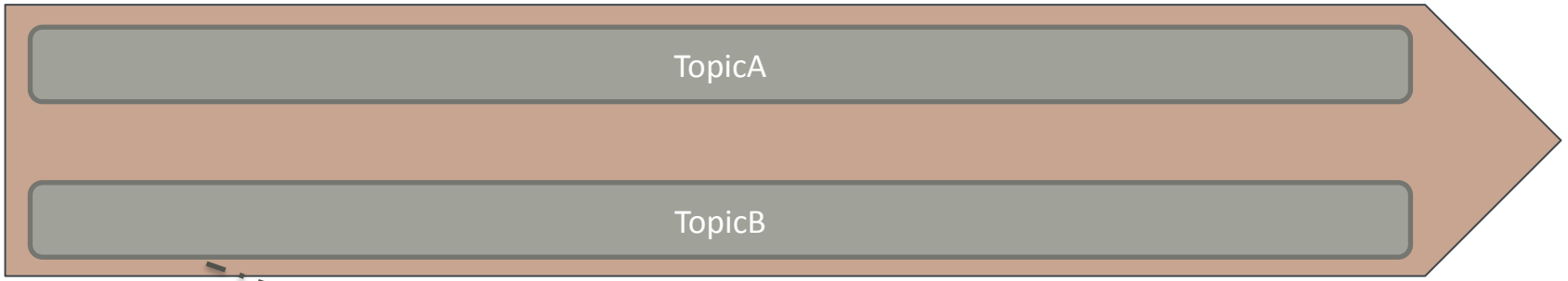
Creamos dos consumidores que consumirán cada uno en un topic distinto.

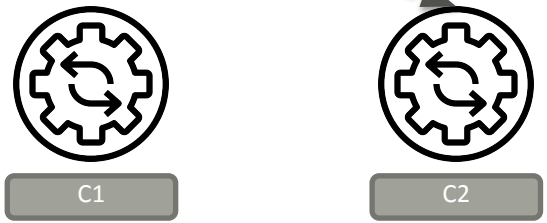


P1 envia un mensaje
a las 00:00

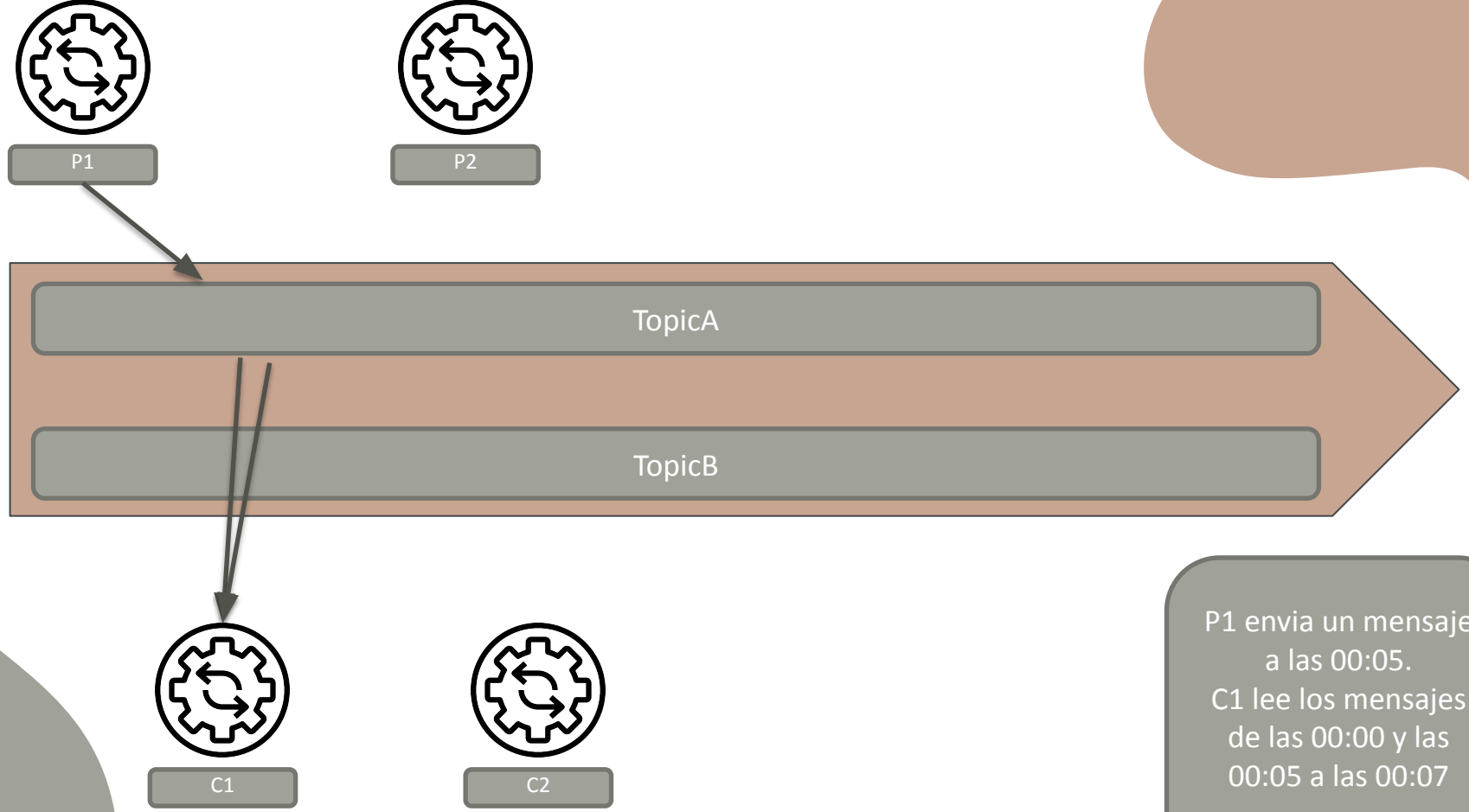


P2 envia un mensaje
a las 00:02

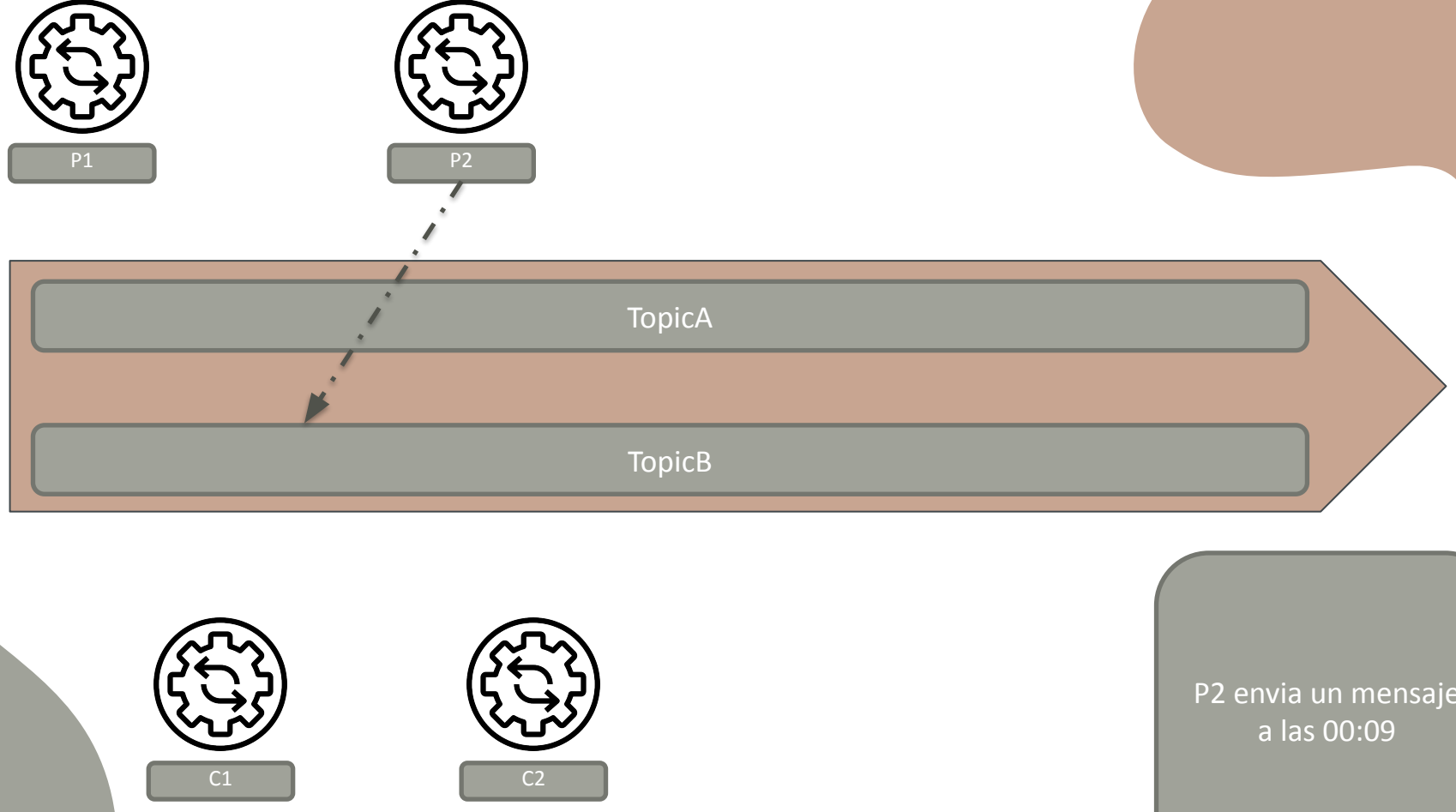




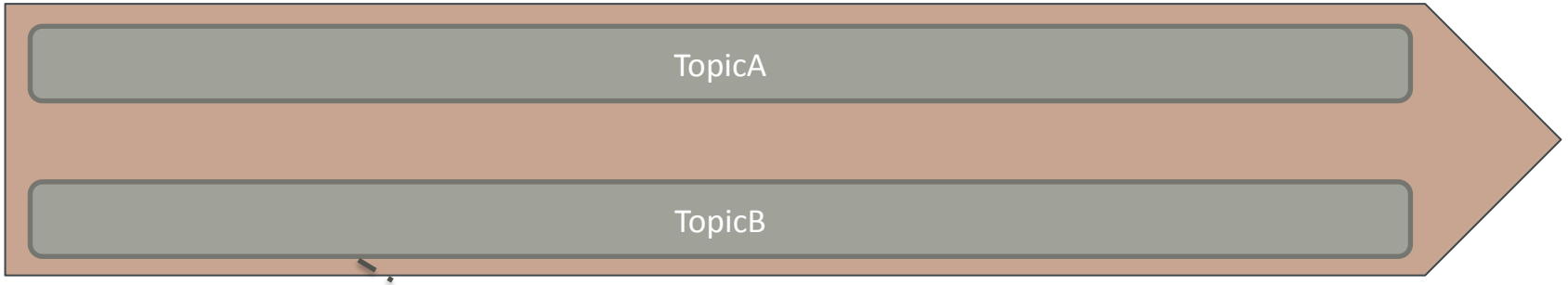
C2 lee el mensaje de las 00:02 a las 00:03

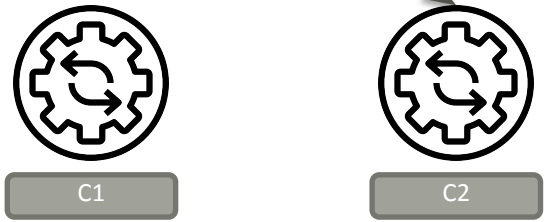


P1 envia un mensaje a las 00:05.
C1 lee los mensajes de las 00:00 y las 00:05 a las 00:07

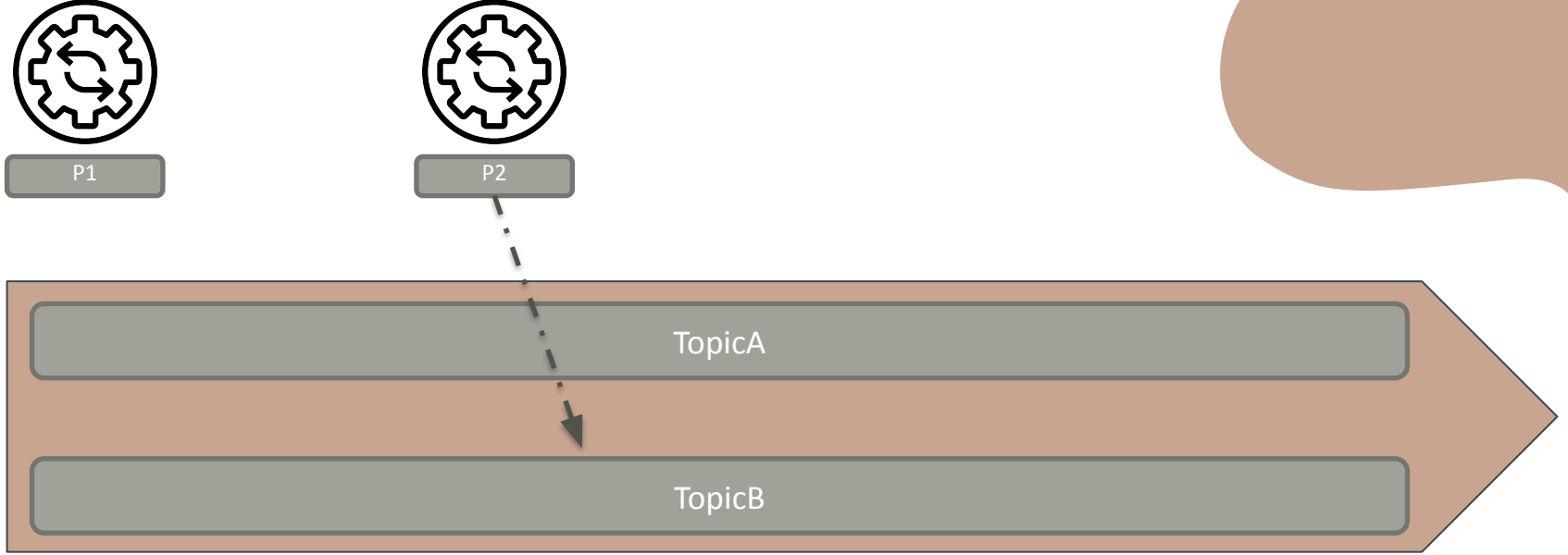


P2 envia un mensaje
a las 00:09

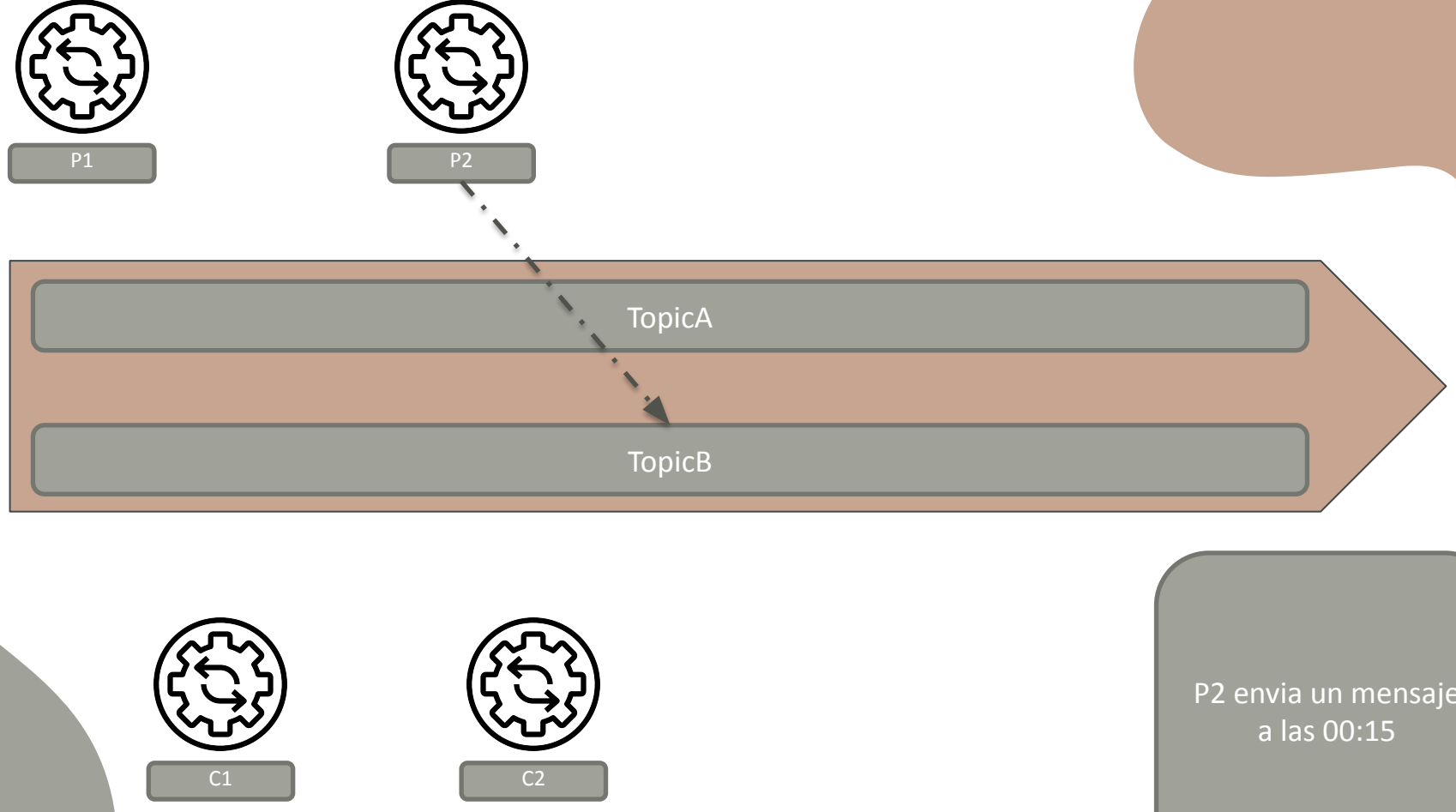


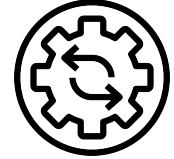


C2 lee el mensaje de las 00:09 a las 00:10

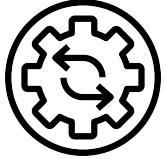


P2 envia un mensaje
a las 00:12

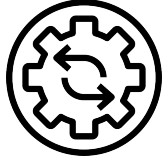

P1
P2
TopicA
TopicB
C1
C2
P2 envia un mensaje
a las 00:15



P1



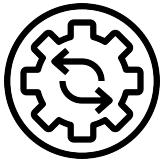
P2



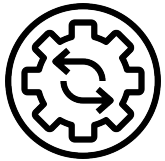
P3

TopicA

TopicB

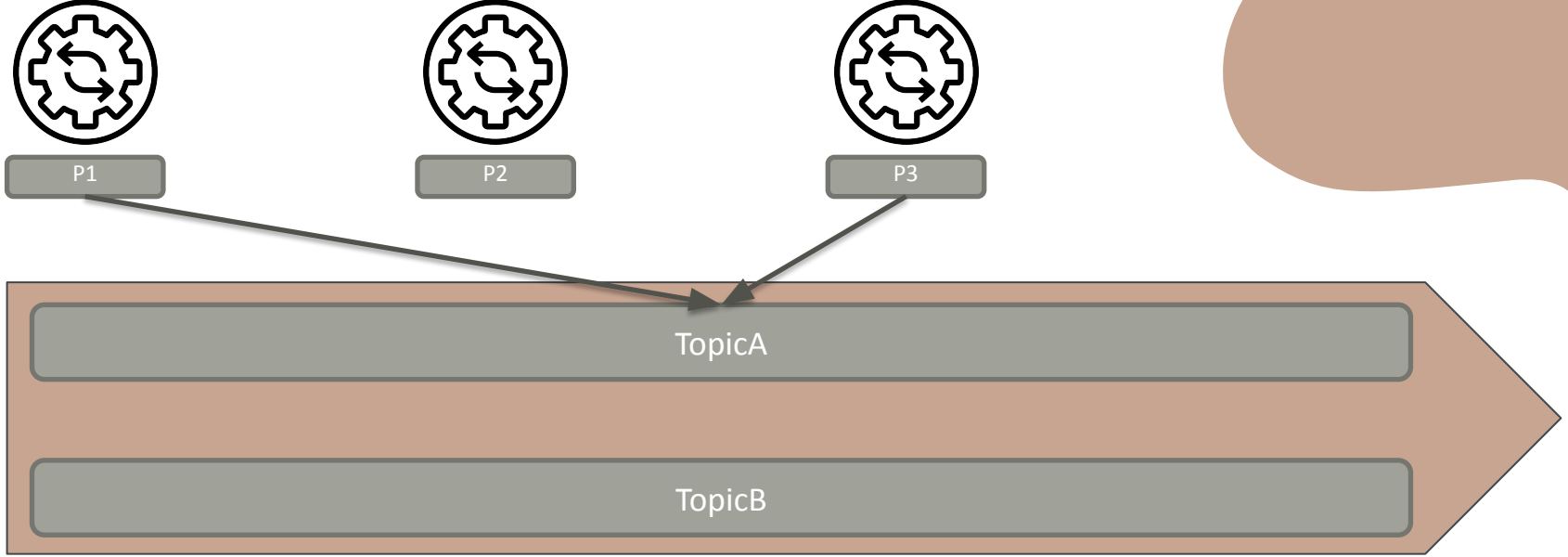


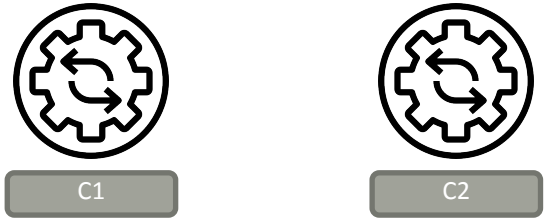
C1



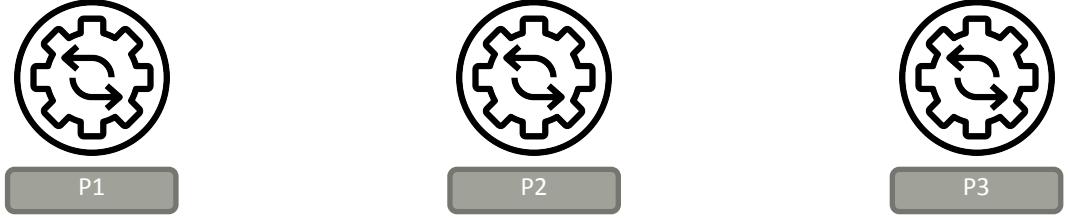
C2

Se añade un nuevo productor P3 que producirá mensajes en el topic A

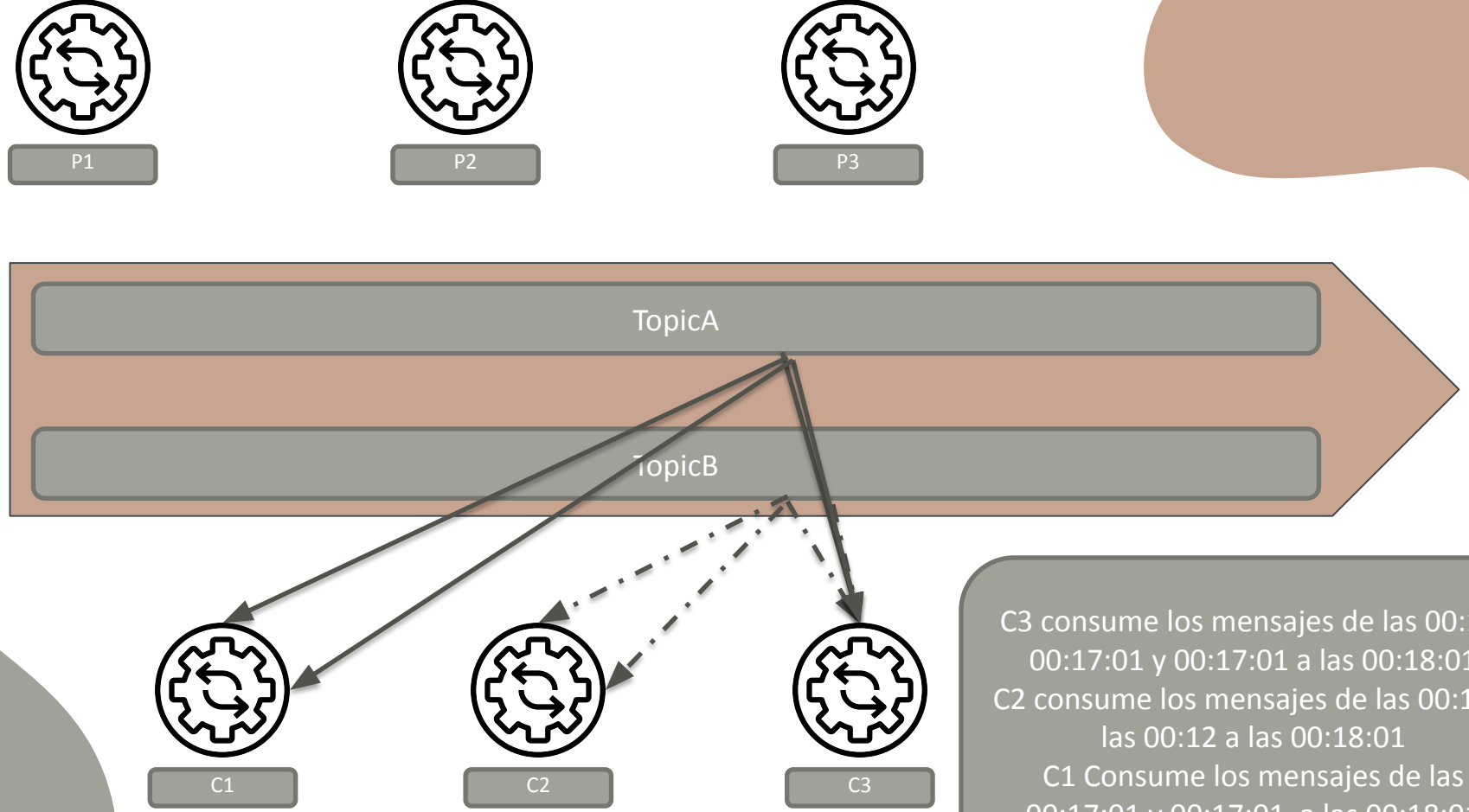




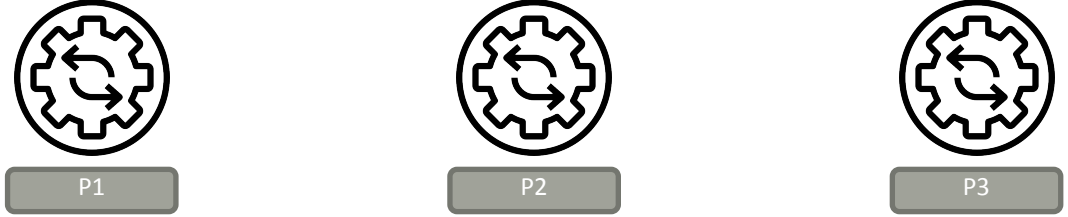
P3 envia un mensaje a las 00:17:01 y P1 envia un mensaje a las 00:17:02



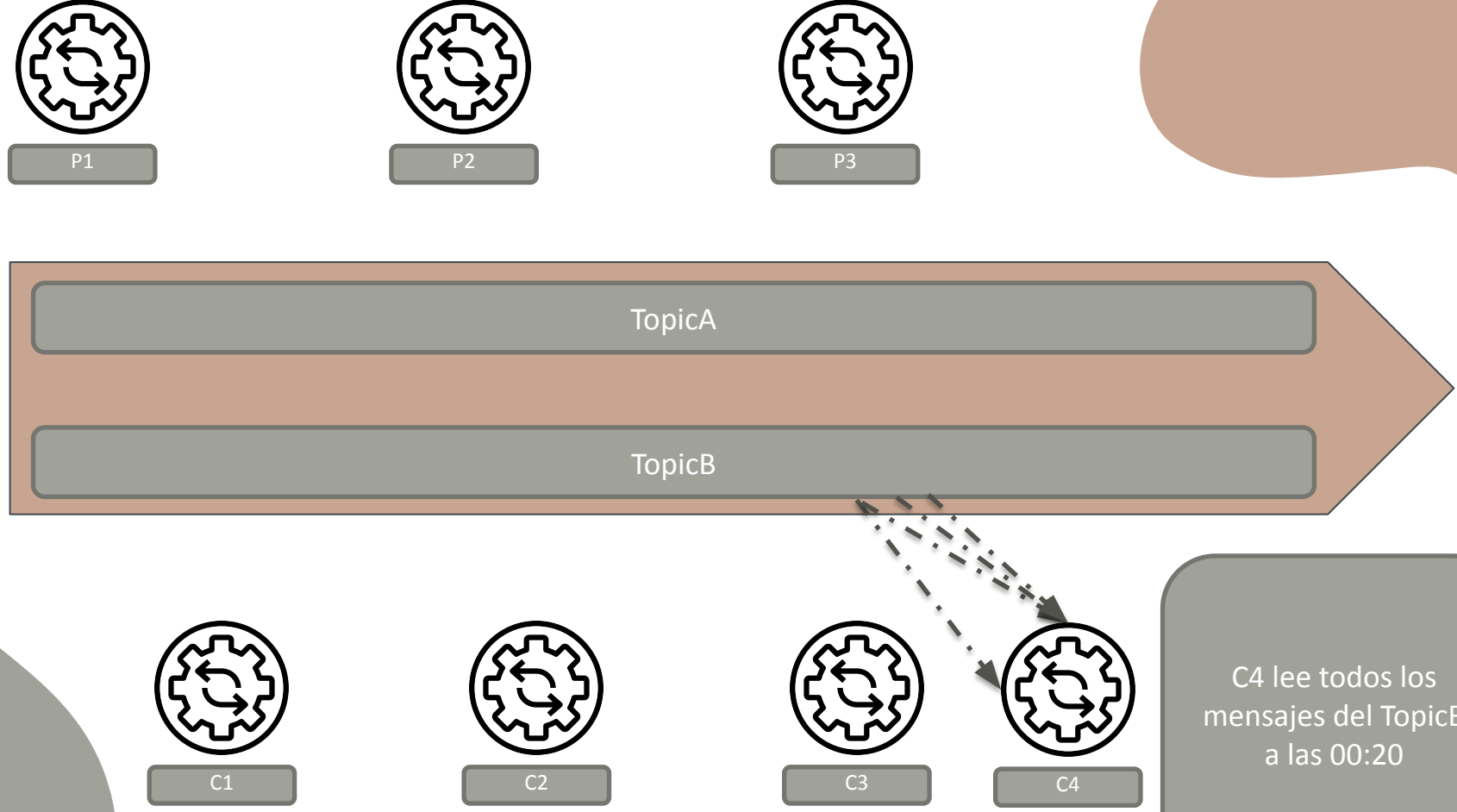
Se añade un nuevo consumidor C3 que consumirá los mensajes de los topics A y B desde las 00:14 a las 00:18



C3 consume los mensajes de las 00:15, 00:17:01 y 00:17:01 a las 00:18:01
C2 consume los mensajes de las 00:15 y las 00:12 a las 00:18:01
C1 Consume los mensajes de las 00:17:01 y 00:17:01 a las 00:18:01



Se añade un nuevo consumidor C4 que consumirá los mensajes del topic A desde el principio a las 00:20


P1
P2
P3
TopicA
TopicB
C1
C2
C3
C4
C4 lee todos los mensajes del TopicB a las 00:20

Apache Kafka - KafkaStreams

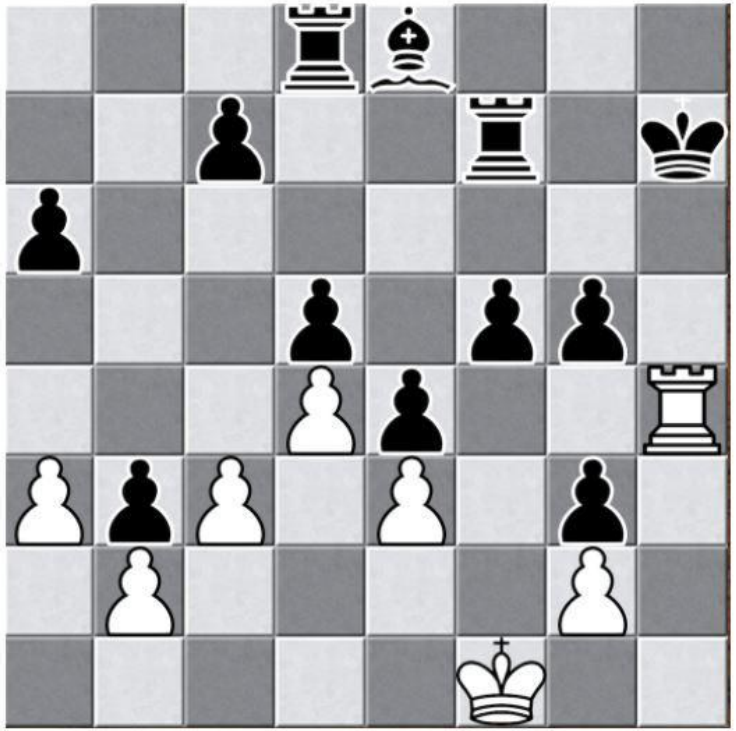
- Al igual que Spark Kafka cada vez está más extendido
- Programar Consumers y Producers no es muy amigable
- Confluent crea una librería para agilizar los desarrollos en Kafka llamada KafkaStreams o KStreams
- Consta de una series de operaciones llamadas Transformaciones que nos ayudan a procesar los datos de un topic
- Dichas transformaciones se separan en dos, con estado o sin estado
- Necesita de un cluster de Kafka activo para funcionar
- Considera que los datos llegan en un flujo constante de información por lo que no hay batchs

Apache Kafka - KafkaStreams vs KafkaTable

- Además de KafkaStream tambien podemos usar KTables para el almacenamientos de datos de un KafkaStream
- Mientras que un KafkaStream es un flujo continuo de datos una KTable es una imagen de los datos sin conocimiento del pasado o del futuro de los datos
- Es un concepto similara a una partida de ajedrez y un tablero siendo KafkaStream el flujo de las jugadas y un KTable el tablero de la partida en un instante de tiempo

Apache Kafka - KafkaStreams vs KafkaTable

♔: e2..f1
♖: f6..f7
♖: f4..h4
♚: g8..h7
♘: c5..e6
♖: f7..e7
♘: e6..f4
♟: a7..a6
♖: h4..h5
♖: e7..f7
♘: f4..h3
♝: c6..e8
♖: h5..h4
♖: b8..d8
♘: h3..g5
♟: h6..g5 {♘}



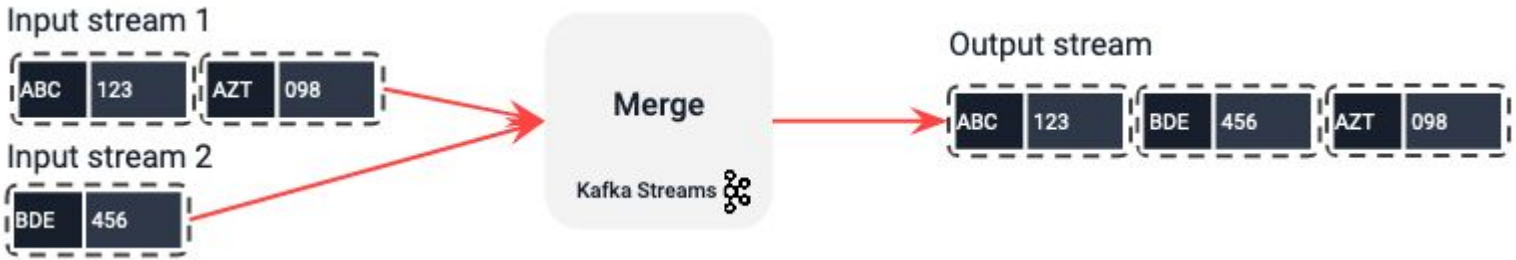
Apache Kafka - Transformaciones sin estado

Apache Kafka - Transformaciones sin estado

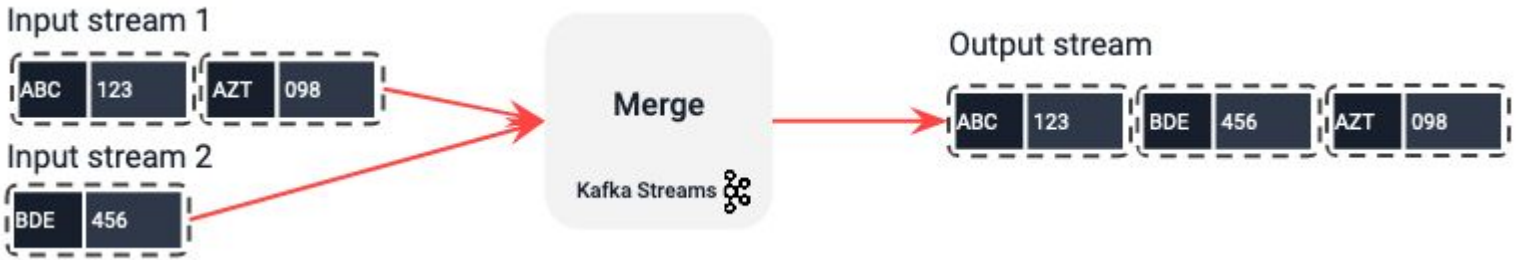
Apache Kafka - Transformaciones sin estado

Apache Kafka - Transformaciones sin estado

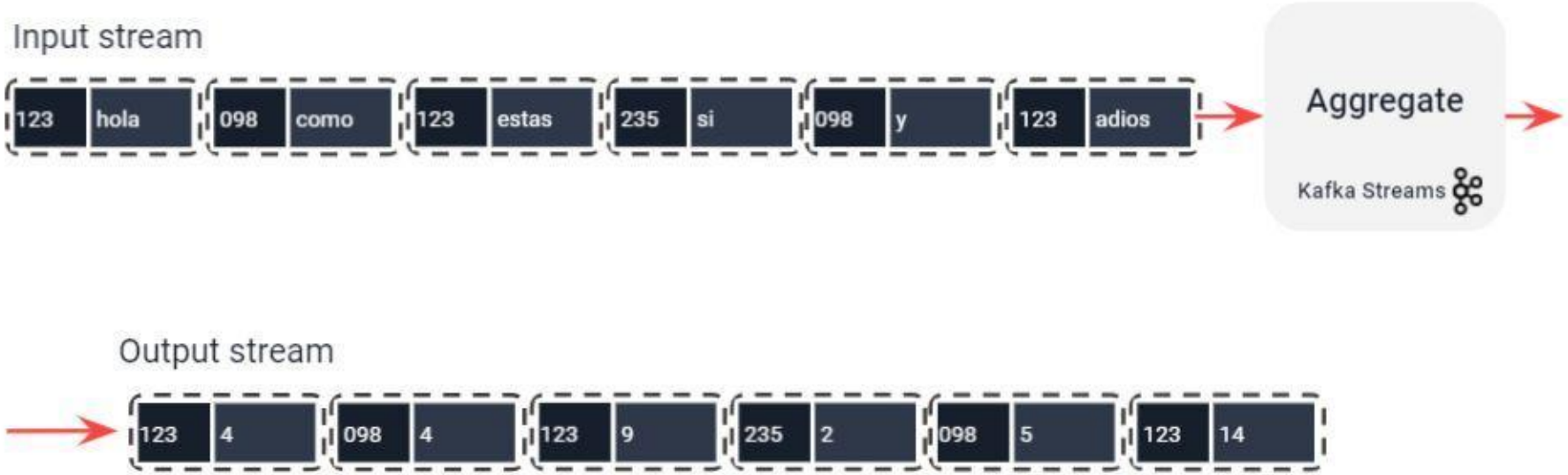
Apache Kafka - Transformaciones sin estado



Apache Kafka - Transformaciones sin estado



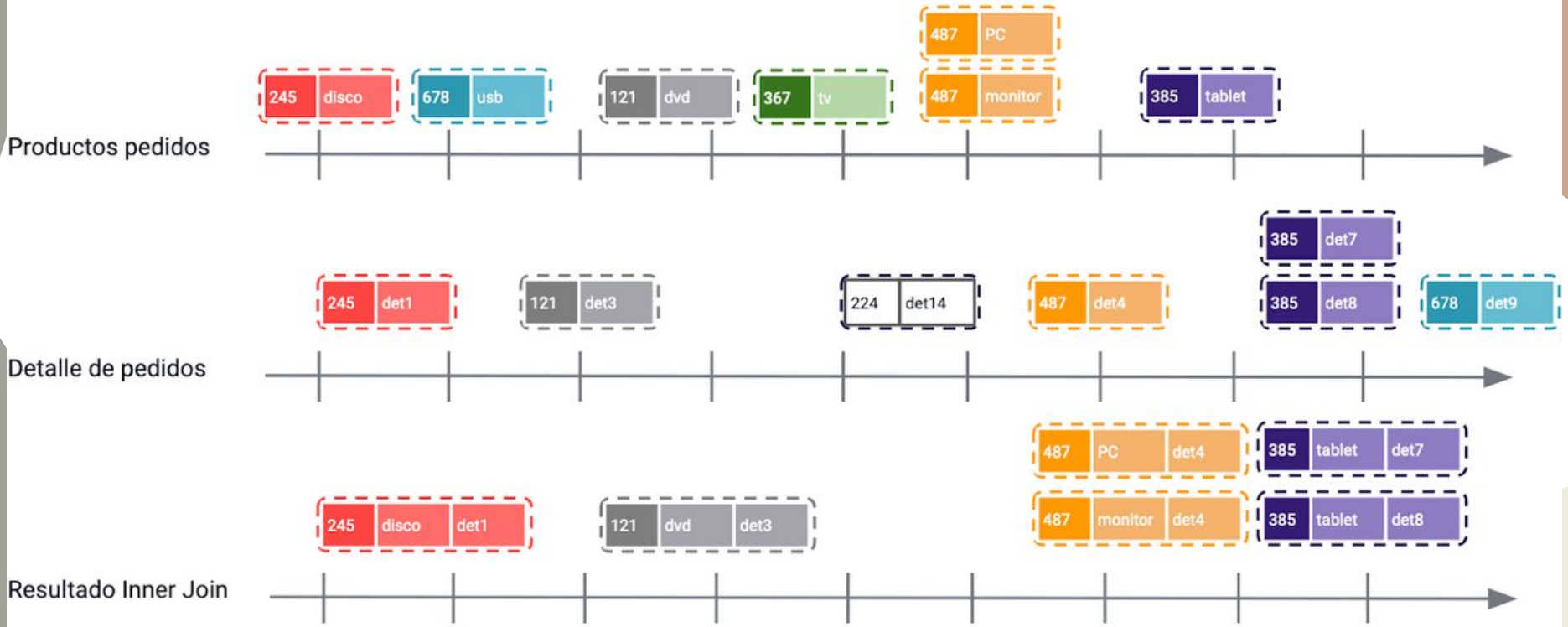
Apache Kafka - Transformaciones con estado



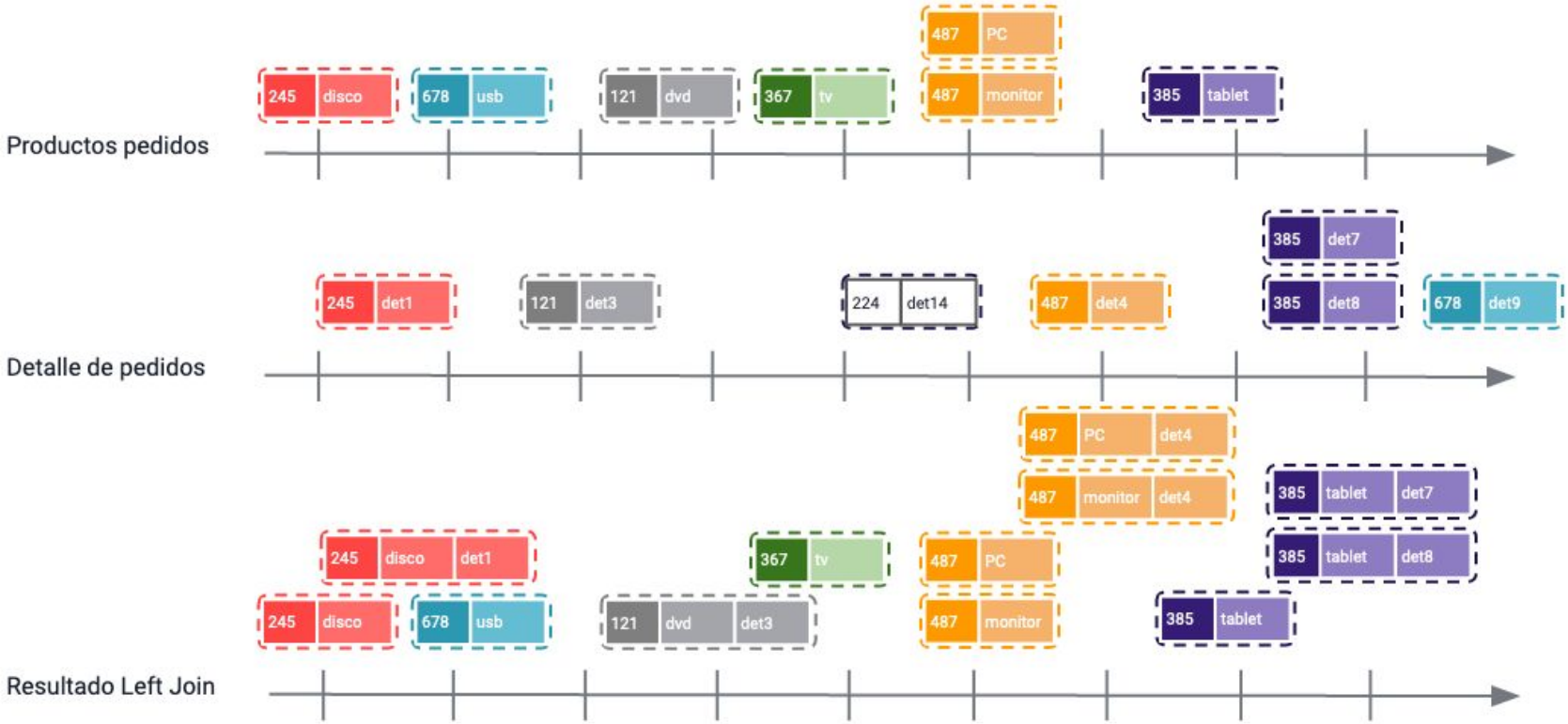
Apache Kafka - Transformaciones con estado

Apache Kafka - Transformaciones con estado

Apache Kafka - Transformaciones con estado(join)



Apache Kafka - Transformaciones con estado(join)



Apache Kafka vs Apache Spark

- Tanto Apache Spark como Apache Kafka son herramientas de procesamiento horizontalmente escalables con capacidad de tratar datos en Streaming

- Aun siendo capaces de resolver la casuística de streaming cada tecnología la resuelve de forma distinta

- Apache Spark
  - Micro Batch
  - Proporciona una interfaz sencilla para procesamiento de datos
  - Permite mezclar procesamiento en batch de datos masivos con procesamiento en Streaming*
  - Permite mezclar procesamiento de Machine Learning de datos masivos con procesamiento en Streaming*

Apache Kafka vs Apache Spark

- Tanto Apache Spark como Apache Kafka son herramientas de procesamiento horizontalmente escalables con capacidad de tratar datos en Streaming

- Aun siendo capaces de resolver la casuística de streaming cada tecnología la resuelve de forma distinta

- Apache Kafka
  - Streaming
  - Proporciona una interfaz sencilla para procesamiento de datos
  - Permite mezclar procesamiento en batch con procesamiento en Streaming*
  - Permite mezclar procesamiento de Machine Learning con procesamiento en Streaming*

Apache Kafka vs Apache Spark

- ¿Cuándo hay que usarlos?
    - Nuestro problema requiere cantidades masivas de datos
    - Siempre que tengamos un caso sólo de Batch será recomendable usar Apache Spark
    - Siempre que nuestro caso de uso impida que nuestros eventos sean procesados en orden se tendrá usar Kafka

SparkStreaming