



UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

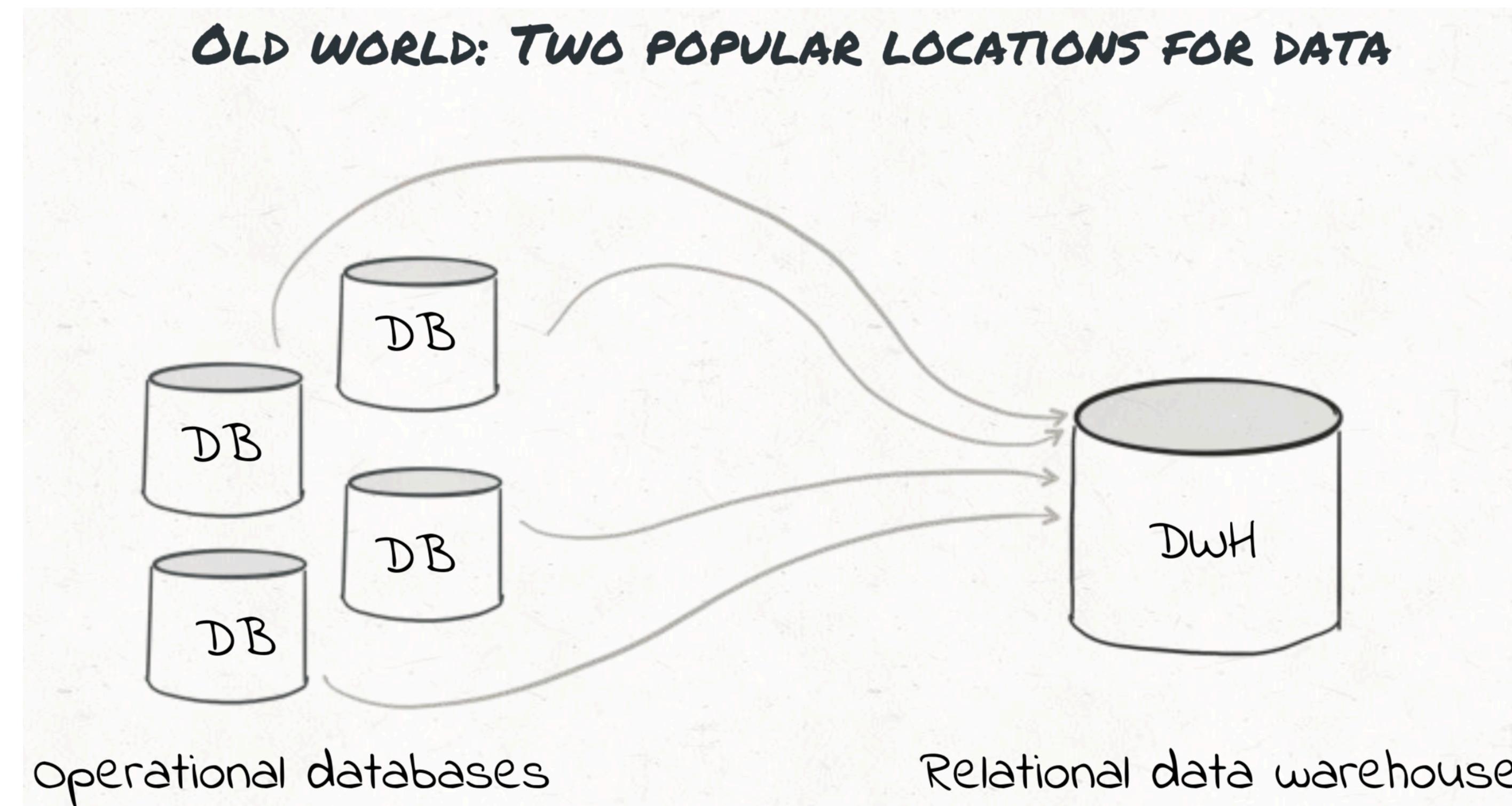
Kafka: Streaming data

Candidato: Carmignola Stefano

Relatore: Prof. Momigliano Alberto
Correlatore: Dr. Porcu Edmondo

Motivazioni per l'evoluzione delle architetture ETL

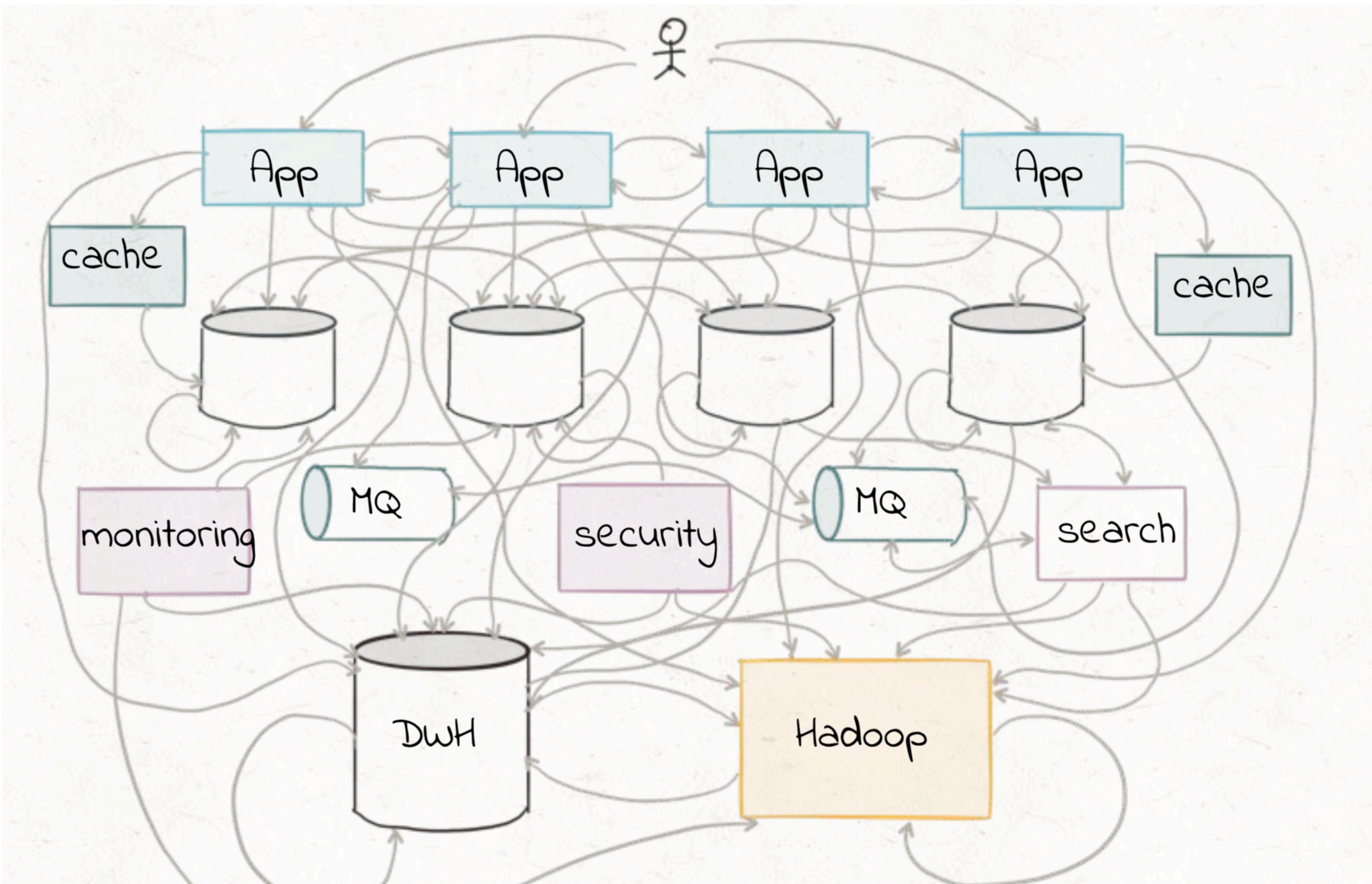
- Architetture con un solo database sono abbandonate sempre più frequentemente in favore di soluzioni *distribuite* basate su molteplici database
- Le tipologie e sorgenti dati utilizzate **non sono più solo RDBMS**. La necessità di gestire e registrare dati presenti anche su database non relazionali (MongoDB, Cassandra) oppure provenienti da sensori o logs è sempre più comune.
- La frequenza di trasferimento di dati è sempre più elevata con una spinta tecnologica mirata verso gli **streams** piuttosto che i processi **batch**.



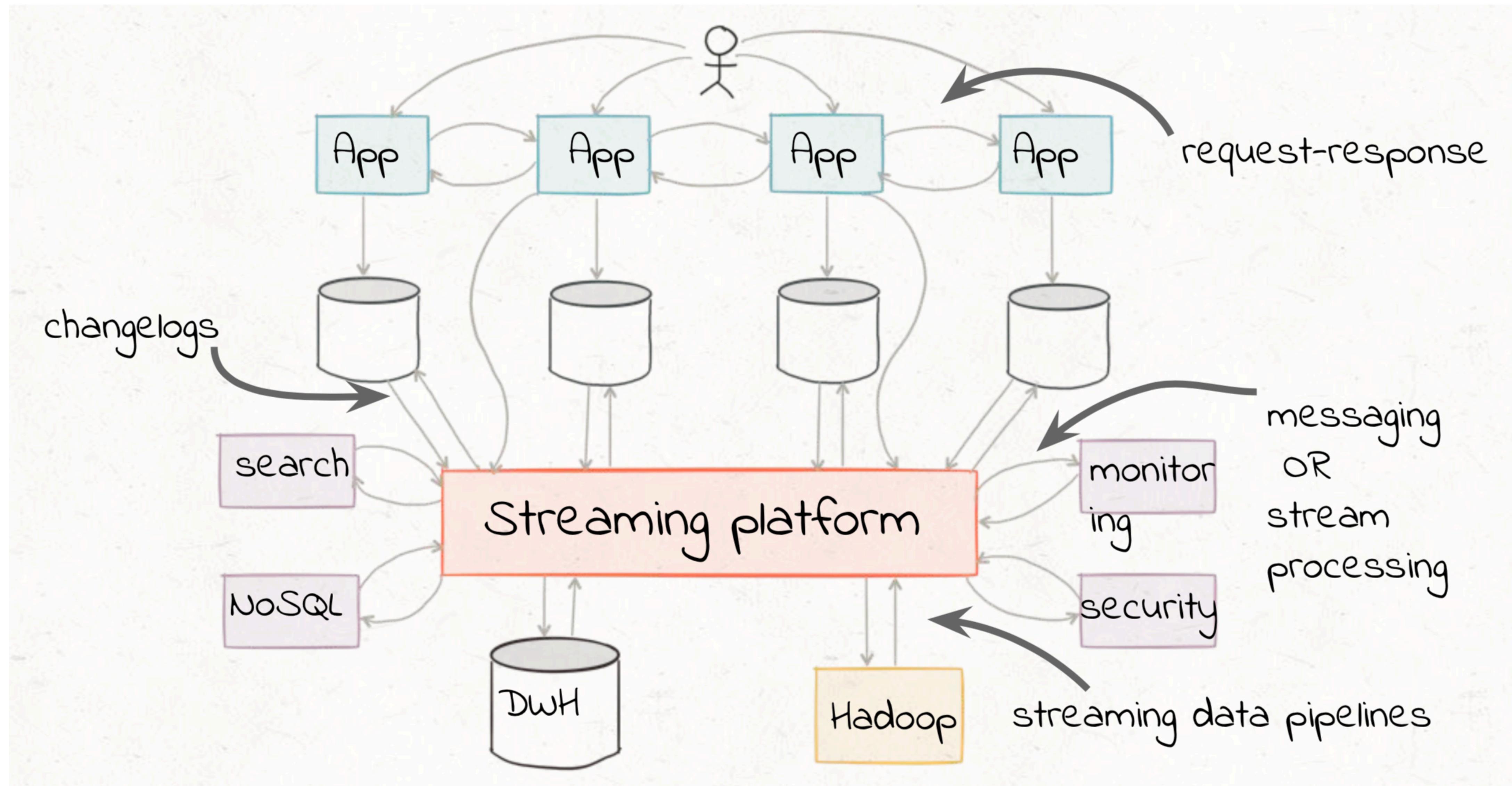
I problemi delle architetture ETL

- E' richiesto un ***schema globale*** del modello dei dati gestito dal processo di ETL
- Le trasformazioni dei dati sono ***inclini agli errori*** e a volte manuali
- I processi di ETL sono generalmente ***lenti*** e richiedono un numero elevato di risorse
- I moderni strumenti di ETL sono ***nati e pensati per gestire dati in modalità batch***; scarsa compatibilità con gli streams di dati
- Trasformazione decentralizzata: inserire la medesima trasformazione di una sorgente in due diversi database richiede la ripetizione della trasformazione

Soluzione ad-hoc



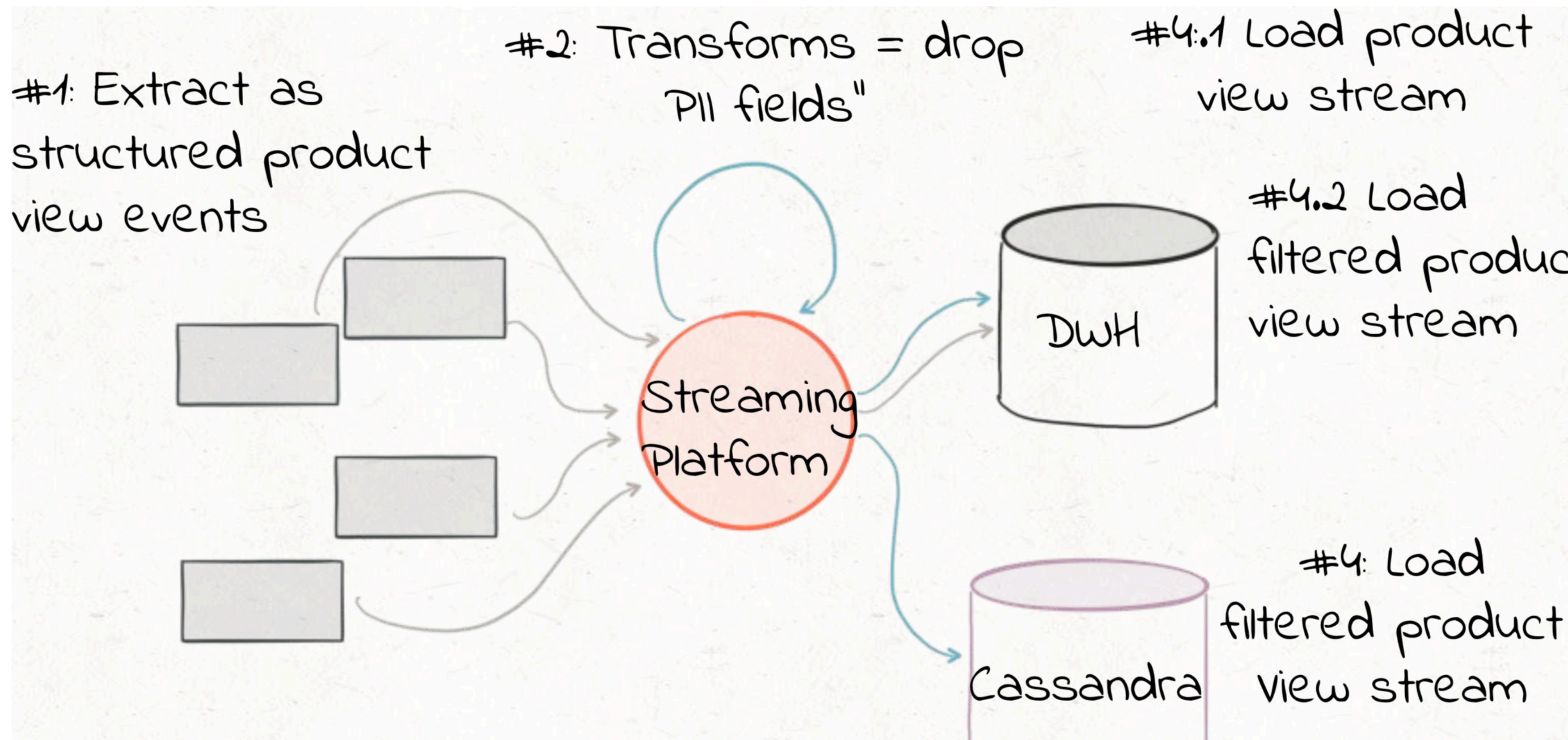
Soluzione basate su streaming platform



Soluzione basate su streaming platform

Una streaming platform è utilizzata come **source-of-truth** e **bus** dei dati dell'intera architettura.

- Capace di gestire **grandi volumi** di **tipologie di dati differenti**
- Creata per la **gestione di eventi** piuttosto che oggetti
- Architettura **forward-compatible**: i dati una volta estratti e trasformati sono trasferibili a qualsiasi destinazione (quando richiesti)
- Le trasformazioni dei dati vengono mantenute dalla streaming platform



Apache Kafka

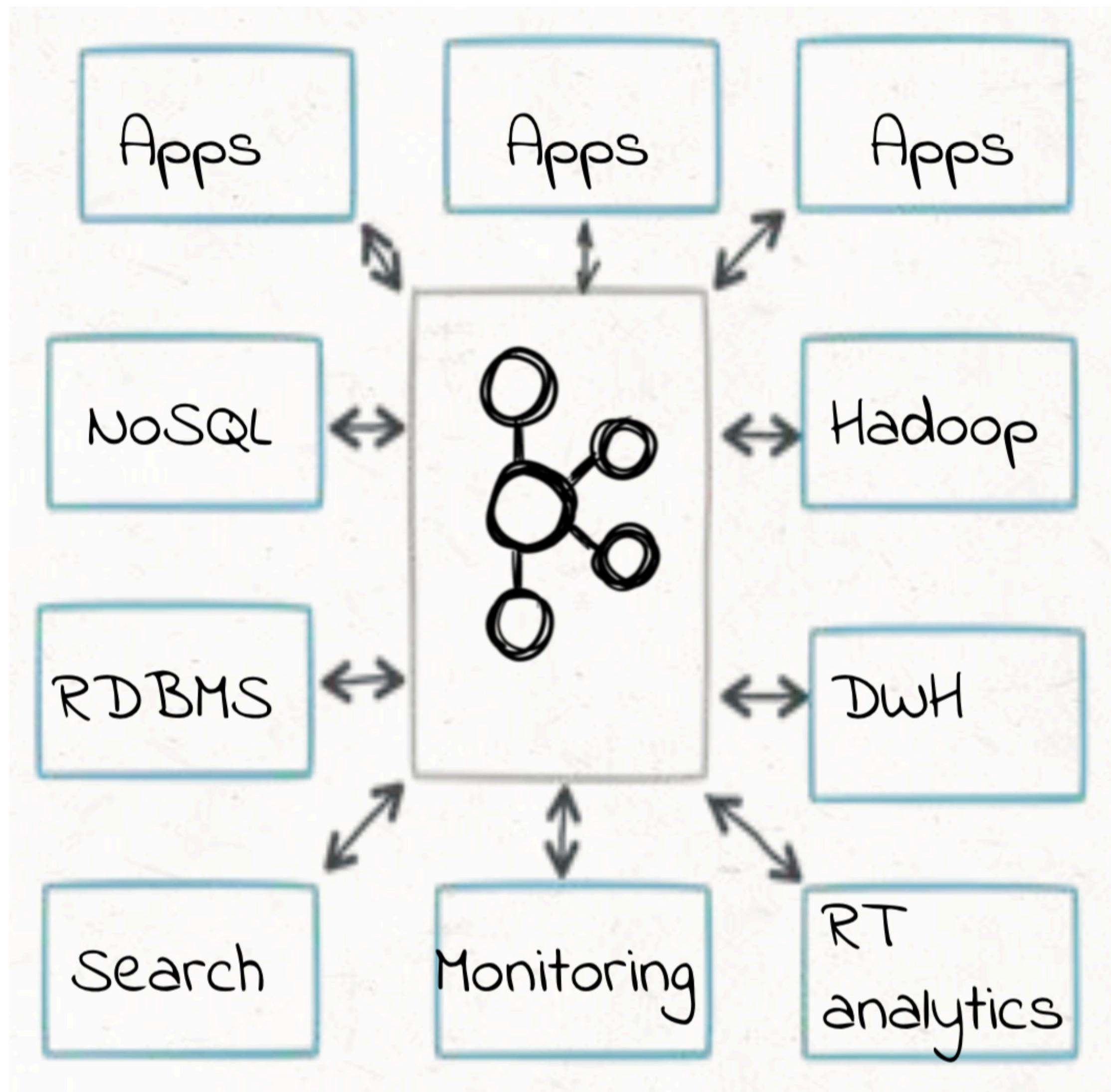
E' una streaming platform creata da **LinkedIn** nel 2011. Open-source, sviluppata in Scala e Java con un vasto ecosistema di librerie e tools dedicati.

Utilizzata da molte delle più importanti aziende nell'ambito tech (Netflix, Paypal, Airbnb, eBay, Apple, Uber per citarne alcune).

Nasce con l'intento di creare un log transazionale distribuito, scalabile ad un grosso numero di messaggi e basato sul pattern pub/sub.

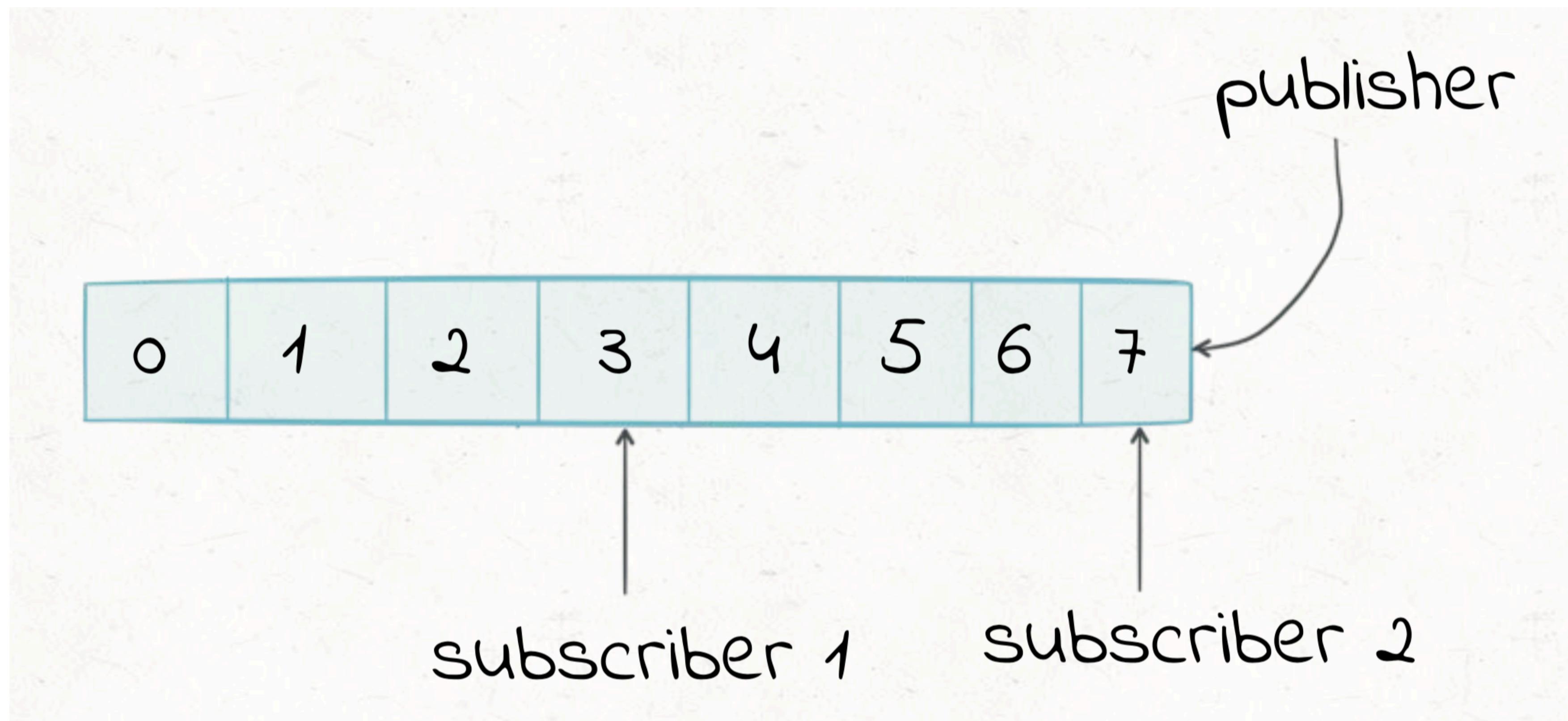
Casi d'uso

1. Database per streams di messaggi
2. Piattaforma per la gestione ed integrazione di dati e applicazioni



Caso d'uso #1: Log

- E' la struttura dati fondamentale alla base di Kafka
- E' alla base del pattern publish/subscribe utilizzato dall'architettura
- E' una coda FIFO dove ogni messaggio è definito da una coppia <chiave, valore>
- I messaggi sono pubblicati da *producers* e consumati da *consumers* in totale autonomia
- Un *publisher* può solo pubblicare messaggi alla fine di una coda mentre un *consumer* mantiene autonomamente il suo offset di lettura
- I dati sono organizzati in *topic* a loro volta suddivisi in più *partition*



Caso d'uso #2: Kafka Messaging API

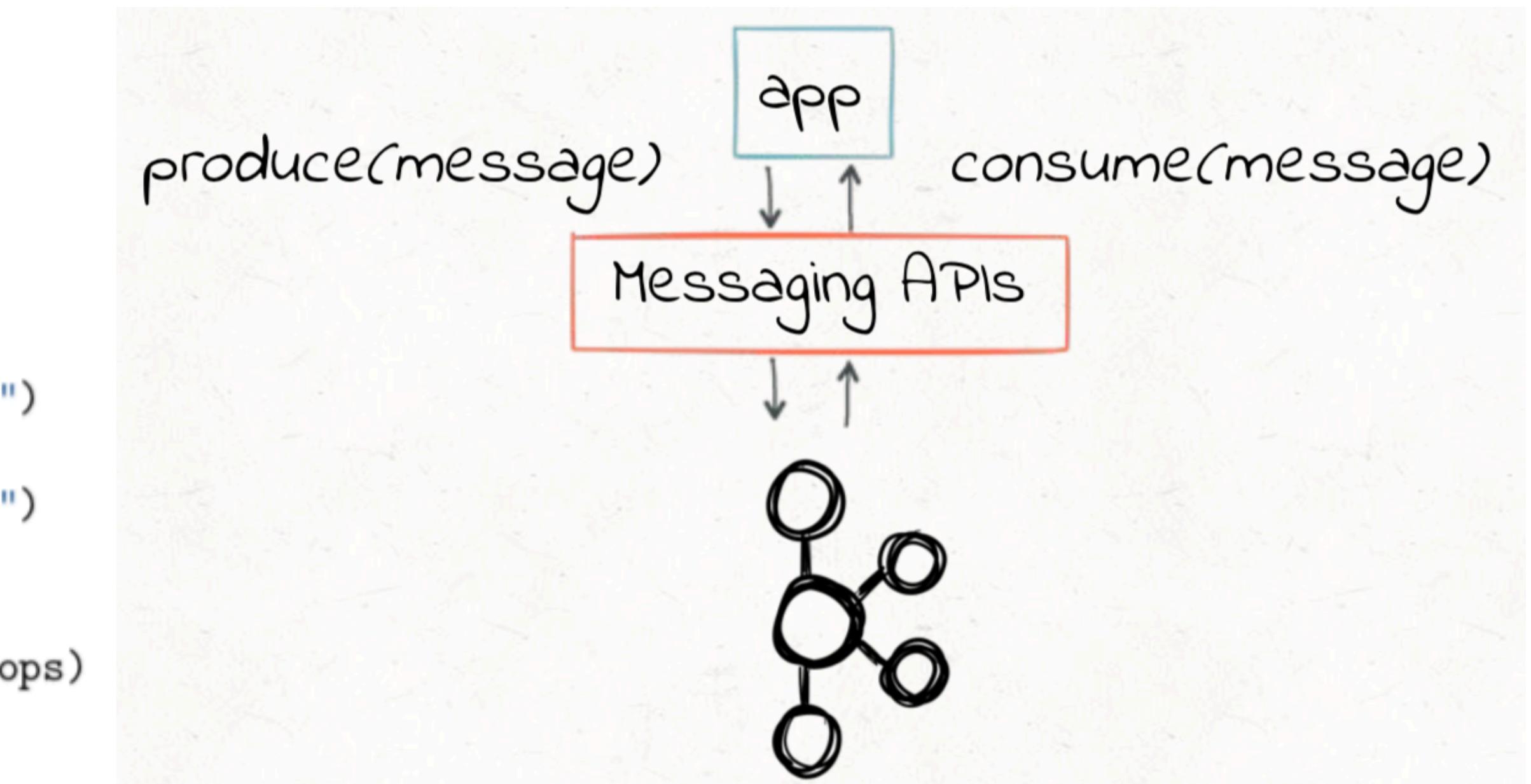
- Librerie che permettono ad una applicazione di produrre e consumare messaggi da/a Kafka
- Ampio supporto per molti linguaggi (Java, Scala, Python, Javascript, ecc.)

```
import java.util.Properties
import org.apache.kafka.clients.consumer.KafkaConsumer

case class Consumer(topic: String){
    val props = new Properties()
    props.put("bootstrap.servers", "localhost:9092")
    props.put("key.deserializer",
        "org.apache.kafka.common.serialization.StringSerializer")
    props.put("value.deserializer",
        "org.apache.kafka.common.serialization.StringSerializer")
    props.put("group.id", "example")

    private val consumer = new KafkaConsumer[String, String](props)
    consumer.subscribe(util.Collections.singletonList(topic))

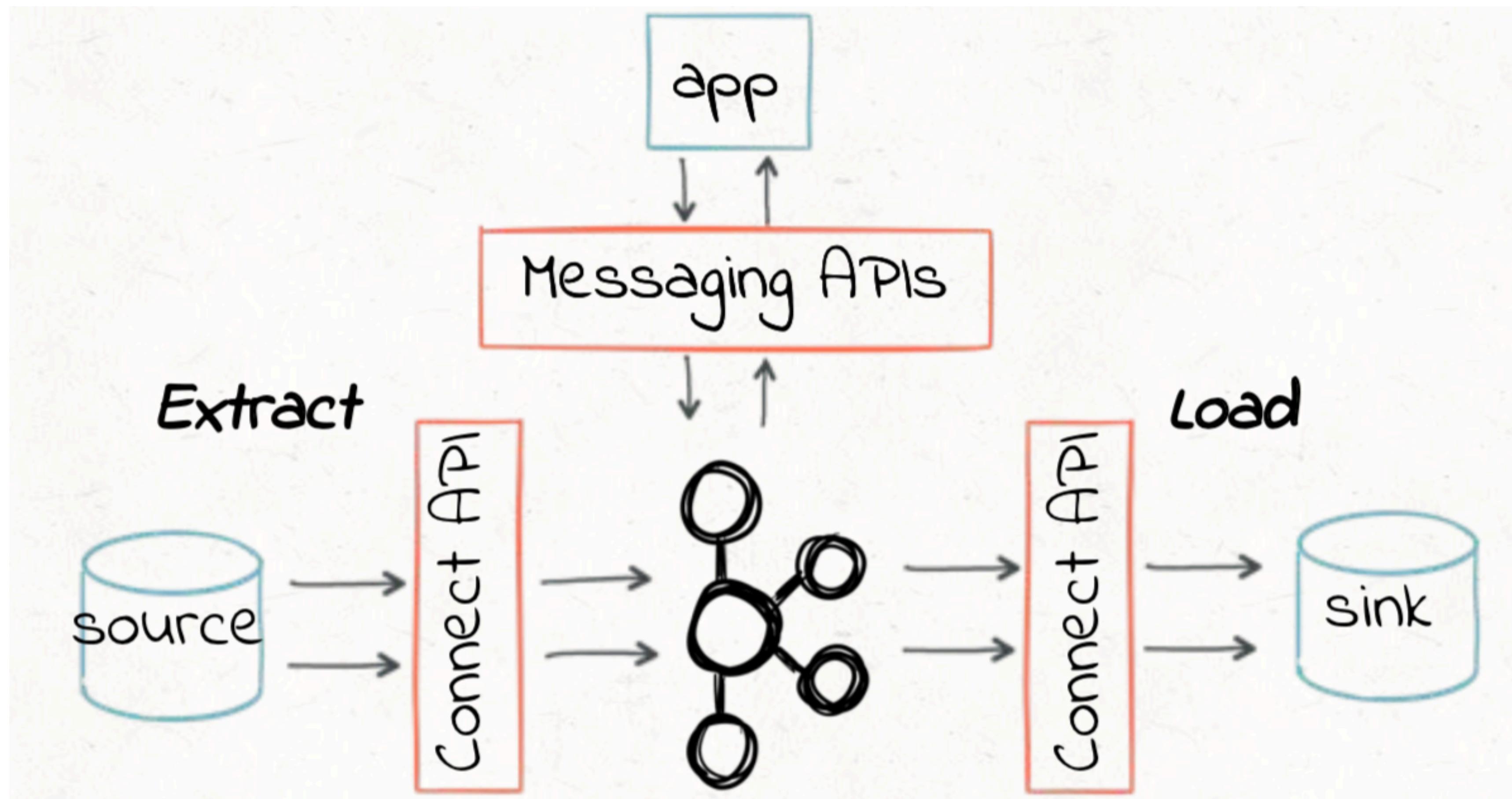
    def readTopic(){
        while(true){
            val records = consumer.poll(100)
            for (r <- records.asScala){
                println(s"${r.offset} ${r.key} ${r.value}")
            }
        }
    }
}
```



Esempio di consumer in Scala

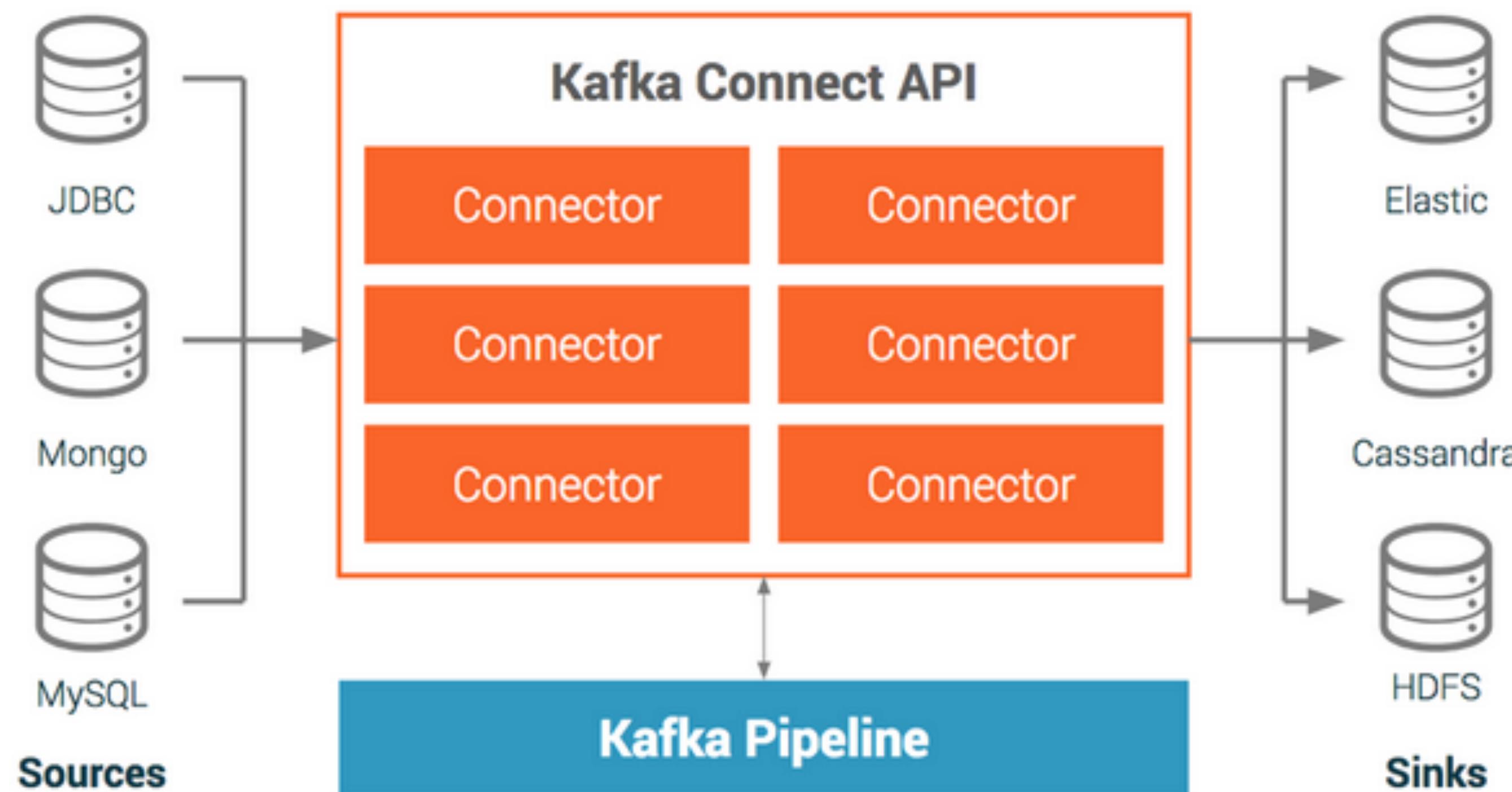
Caso d'uso #2: Kafka Connect API

- Framework per la creazione *Source* e *Sink connectors*
- Utilizzati per creare delle data pipeline tra Kafka e servizi esterni
- Molteplici soluzioni OSS per collegare Kafka ai DB più comuni (MySQL, PostgreSQL, Cassandra, MongoDB, AWS) oppure applicazioni esterne (Salesforce, Twitter, applicazioni IoT)
- Supporto per la gestione/conversione automatica del formato dei dati tramite *Schema Registry*



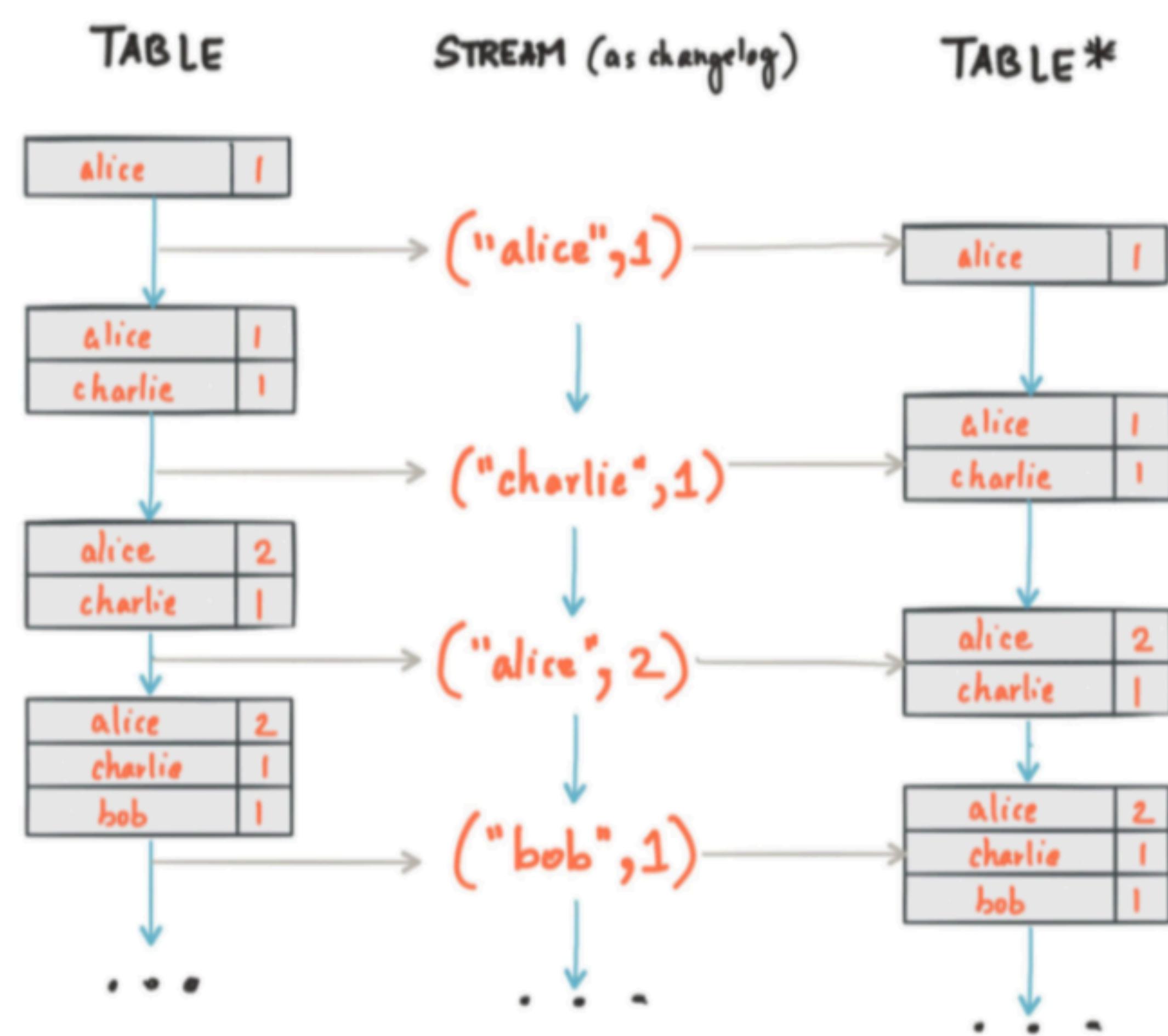
Source e Sink connectors

- Corrispondono alle fasi di extract e load in un processo di ETL
- Basati su meccanismi di push/pull
- Source connectors sono utilizzati per permettere a servizi esterni di scrivere verso topic Kafka (push)
- Sink connectors permettono a servizi esterni di leggere dati da Kafka (pull)



Dualità tra database e stream di dati

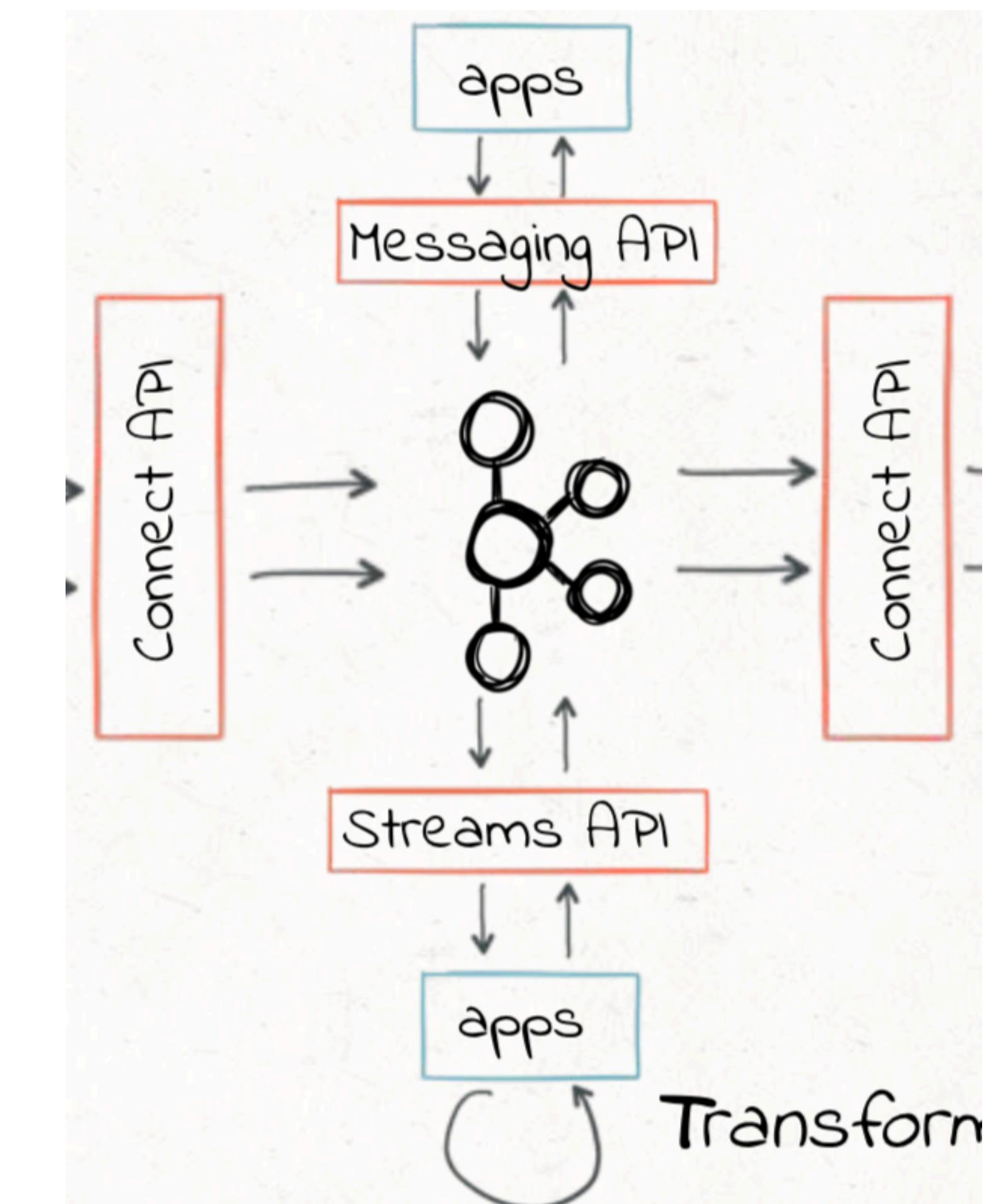
E' possibile vedere una tabella di un database come l'aggregazione in un particolare momento di eventi passati e viceversa. Questa dualità permette a Kafka di sfruttare i *changelog* di un database per creare stream di dati partendo da una tabella del db.



Caso d'uso #2: Kafka Streams API

```
import java.util.Properties  
  
import org.apache.kafka.common.serialization._  
import org.apache.kafka.streams._  
import org.apache.kafka.streams.KStream, KStreamBuilder  
  
object MapFunctionScalaExample {  
  def main(args: Array[String]) {  
    val builder = new KStreamBuilder  
  
    val streamingConfig = {  
      val settings = new Properties()  
      settings.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")  
      settings.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,  
                  Serdes.ByteArray.getClass.getName)  
      settings.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,  
                  Serdes.String.getClass.getName)  
      settings  
    }  
  
    val text: KStream[Array[Byte], String] = builder.stream("input-topic")  
    val upperCasedText: KStream[Array[Byte], String] =  
      textLines.mapValues(_.toUpperCase())  
    upperCasedText.to("output-topic")  
  }  
}
```

- La fase di trasform in un processo ETL
- Libreria per *stream processing*
- Utilizzata per manipolare facilmente i dati provenienti da un topic Kafka
- Astrazione di un event store



Grazie per l'attenzione

Citazioni

- **Neha, N. (2017).** *ETL Is Dead, Long Live Streams: Real-Time Streams with Apache Kafka.*
www.youtube.com/watch?v=I32hmY4diFY
- **Neha Narkhede, G. S. (2017).** *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale.*
O'Reilly Media, Inc.