PiterPy 2019

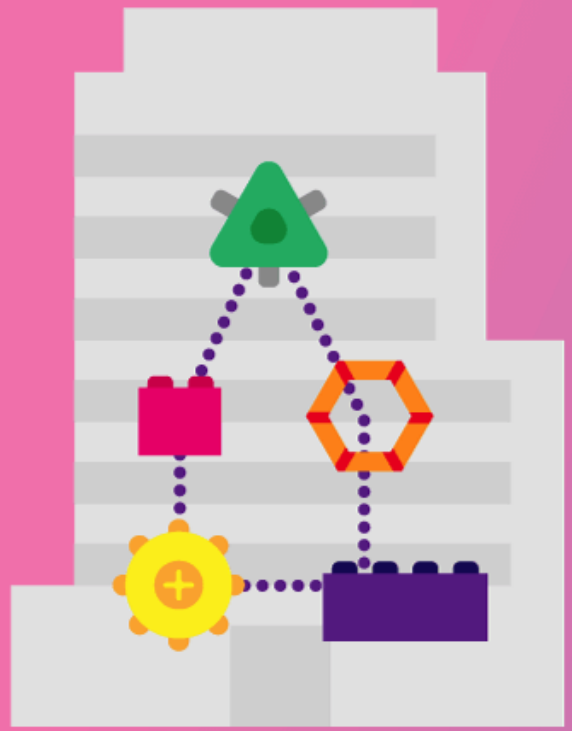
Building a microservices-based application using Kafka and Django

Mikalai Saskavets

art by René Aigner

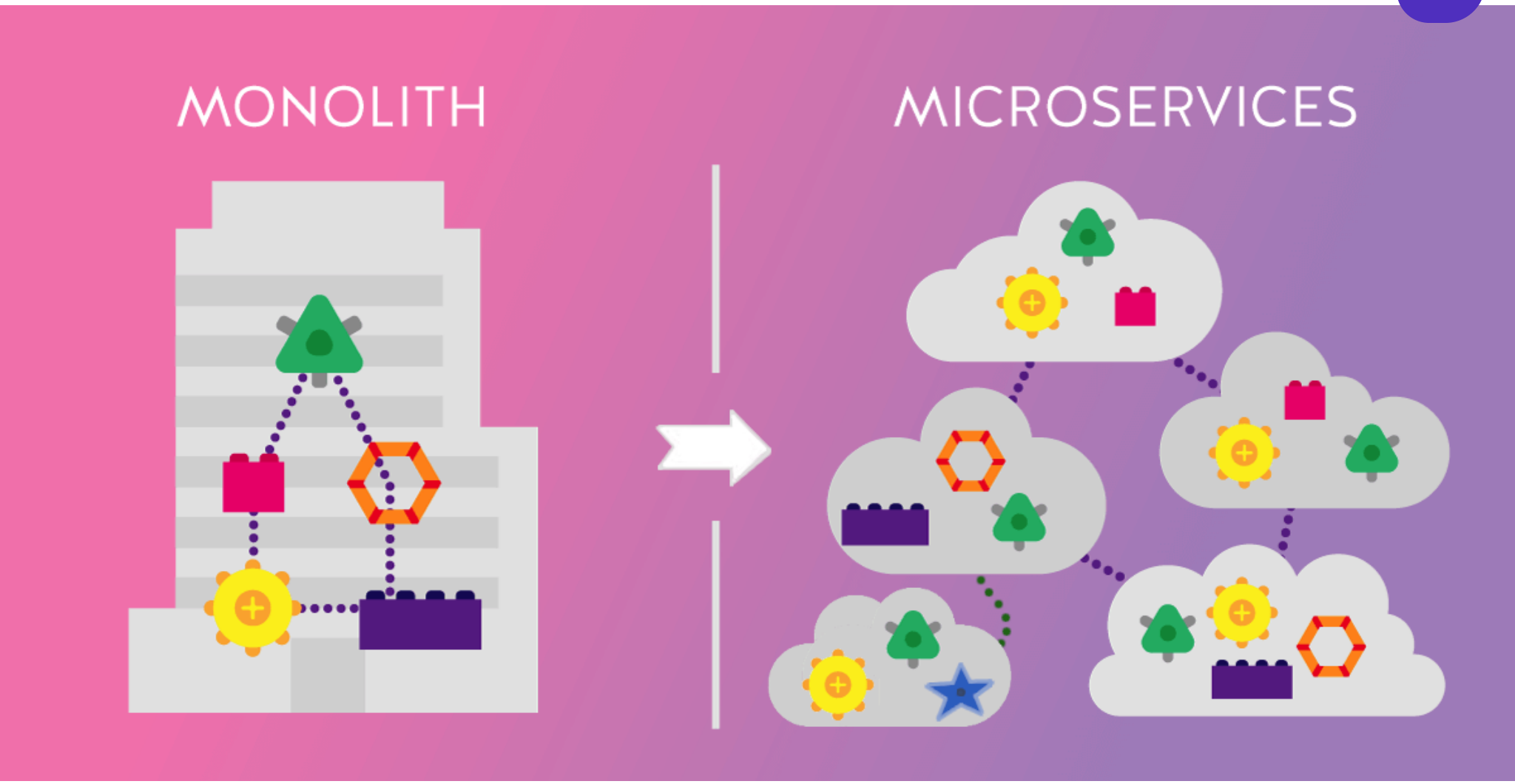

MICROSERVICES
MICROSERVICES EVERYWHERE

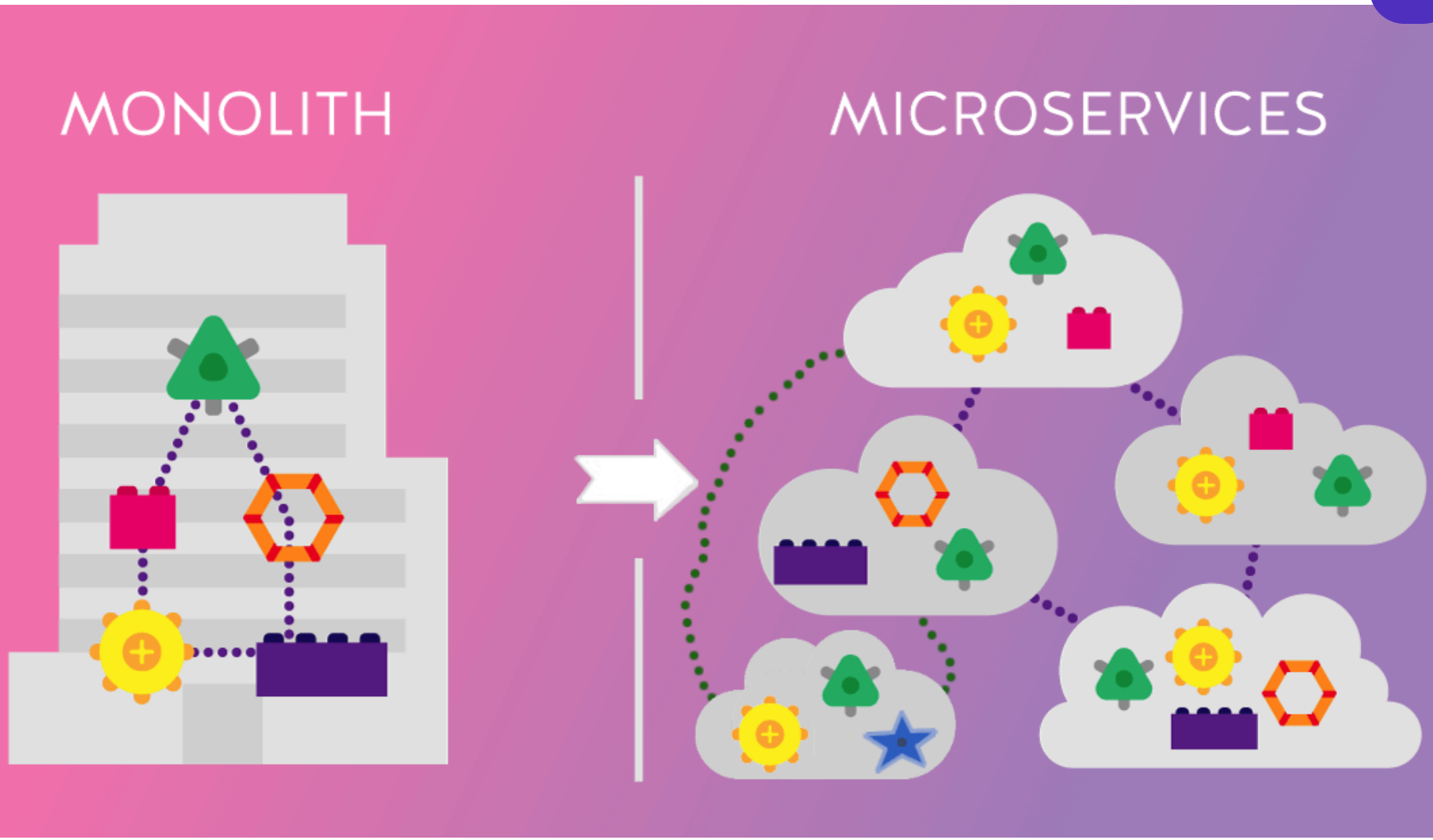
MONOLITH

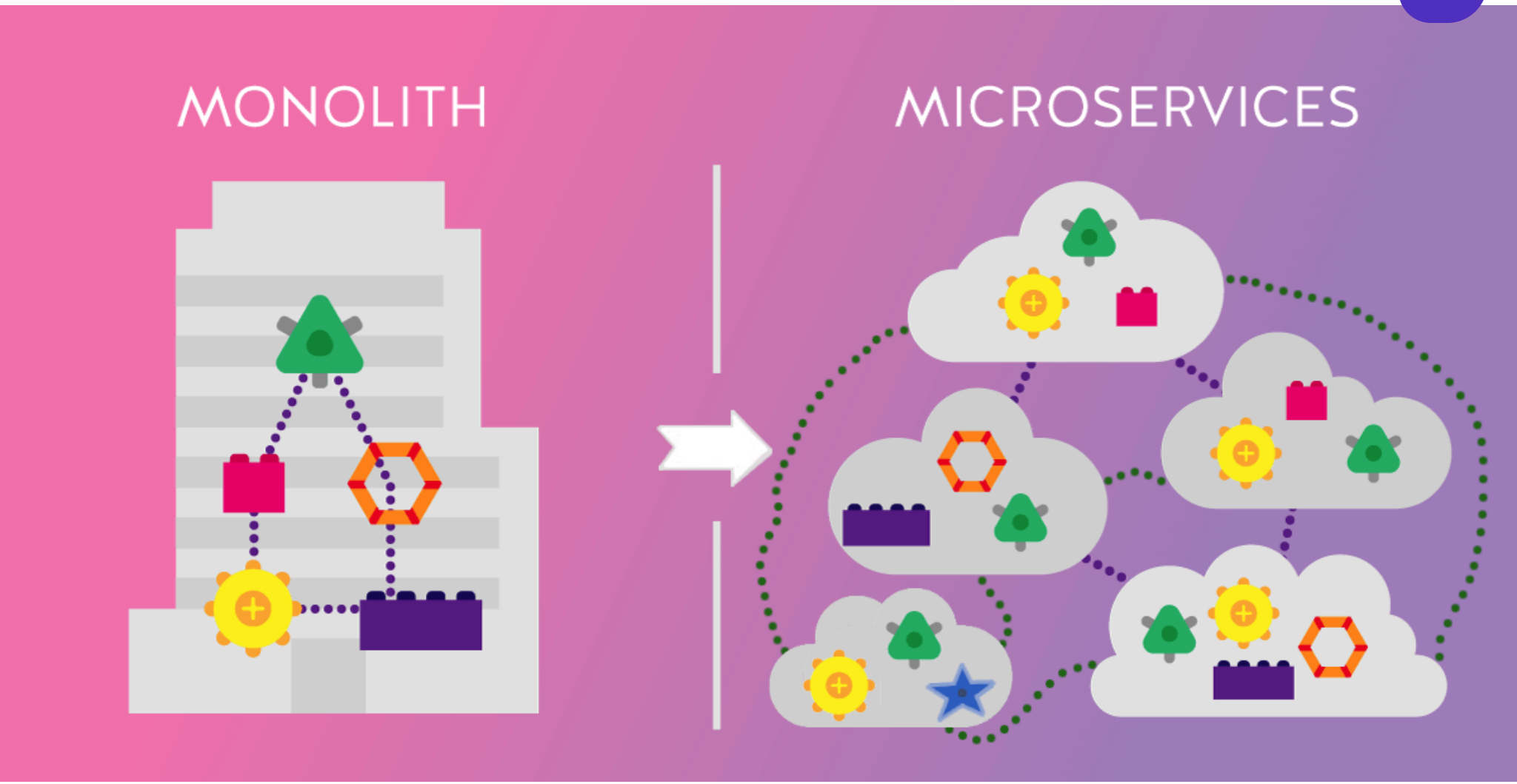


VS

MICROSERVICES

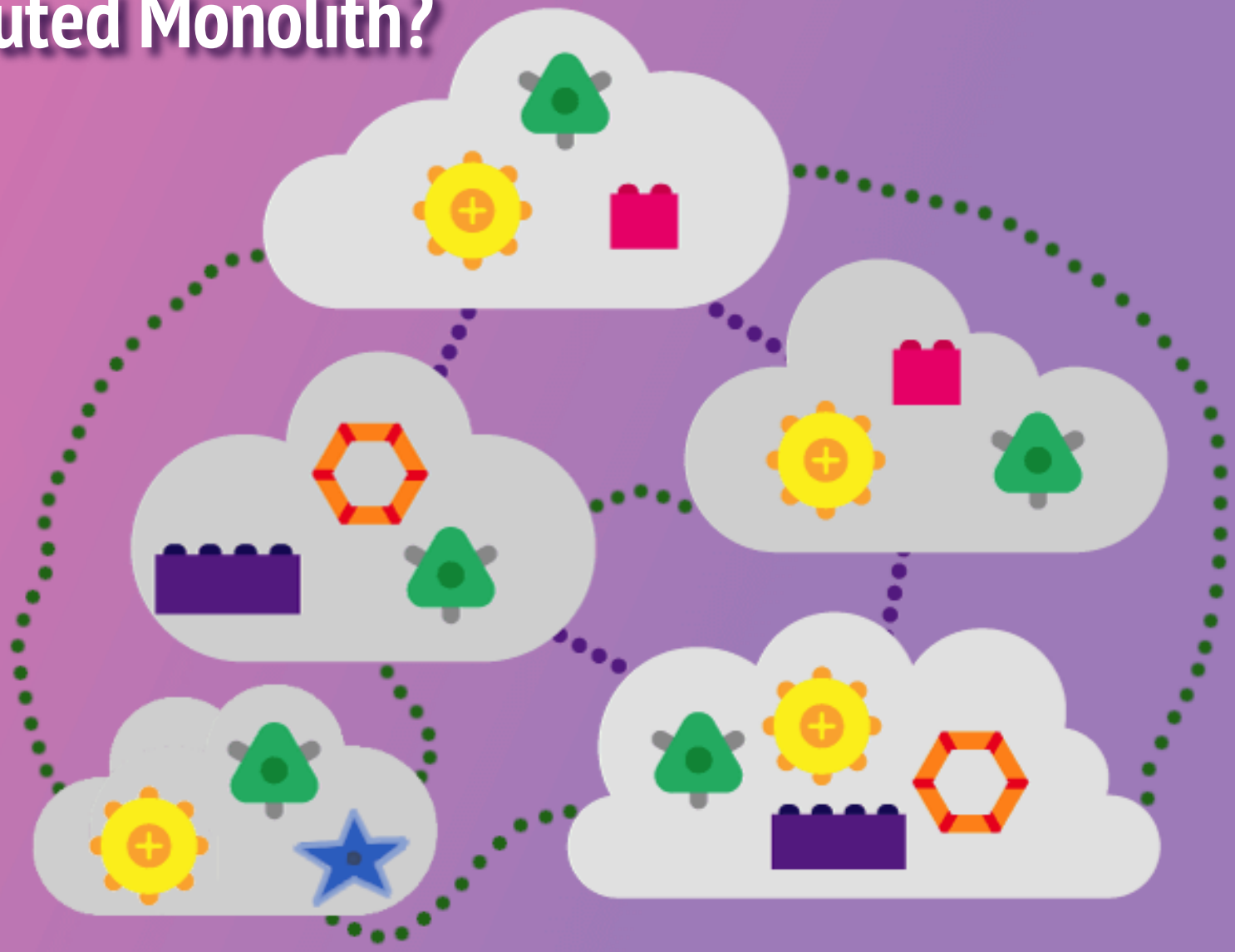

MONOLITH
MICROSERVICES


MONOLITH
MICROSERVICES


MONOLITH
MICROSERVICES

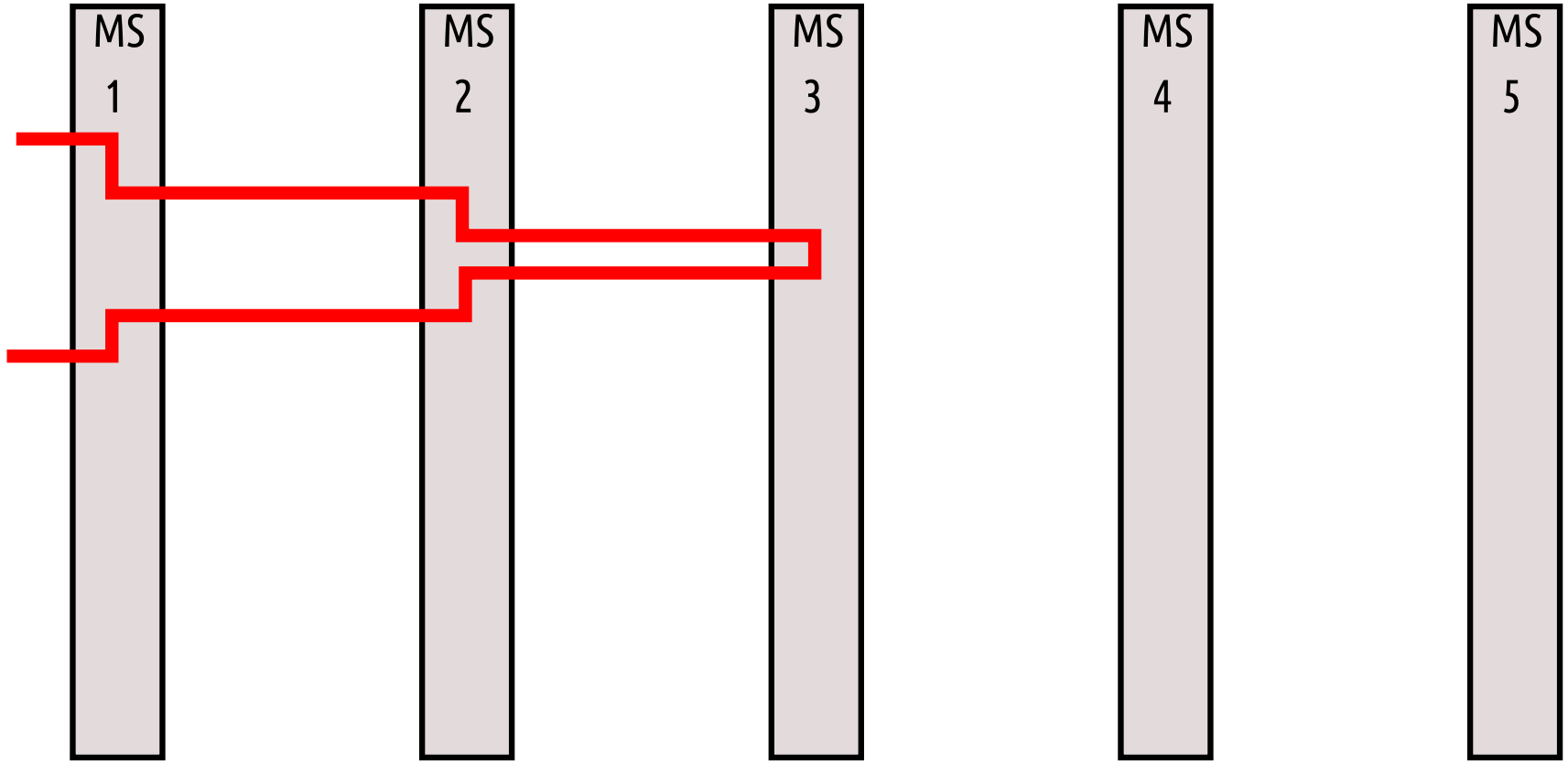
Microservices: Coupling and Cohesion?

Distributed Monolith?



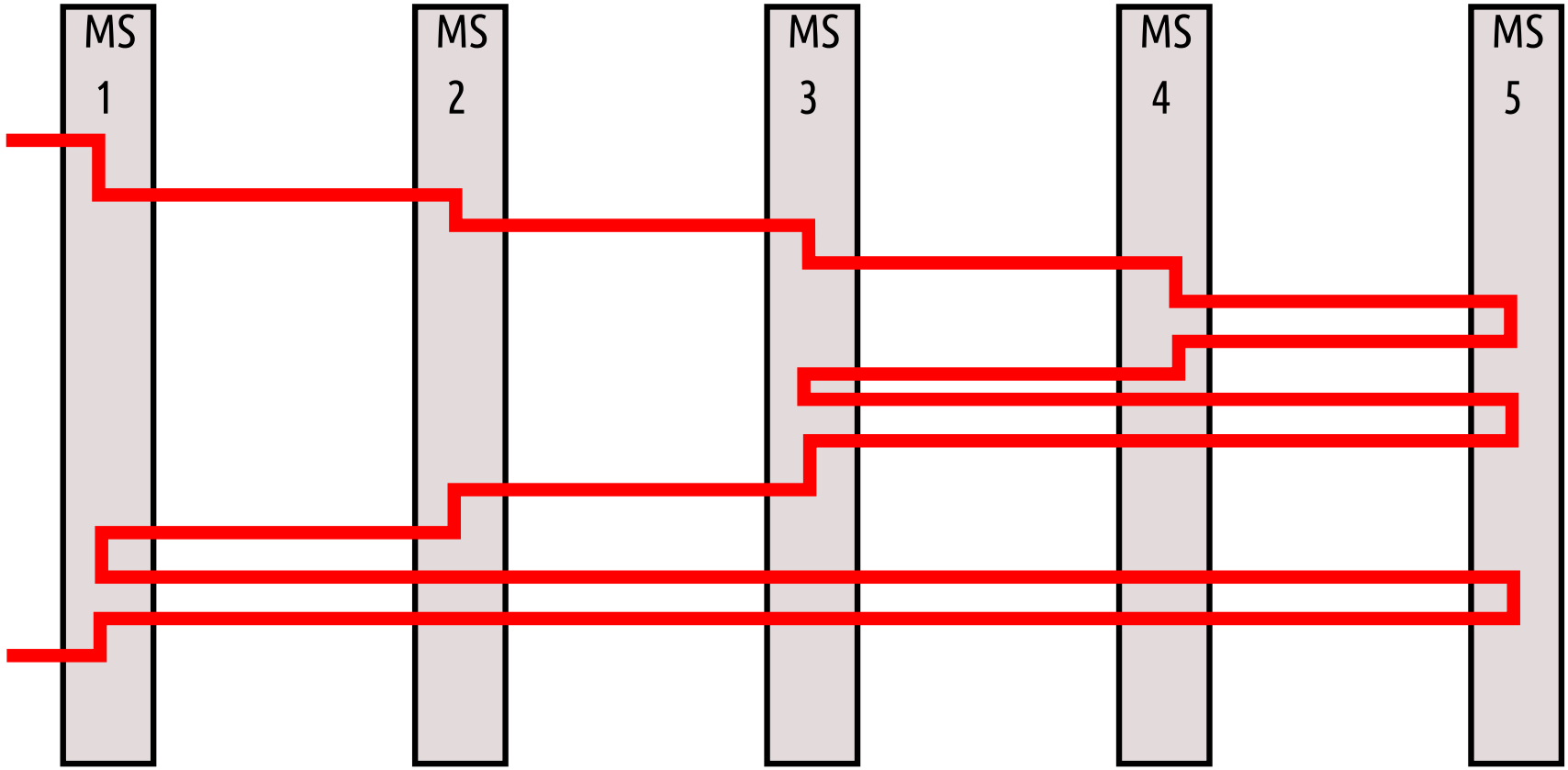
Microservices: Synchronous Communication

(REST API)

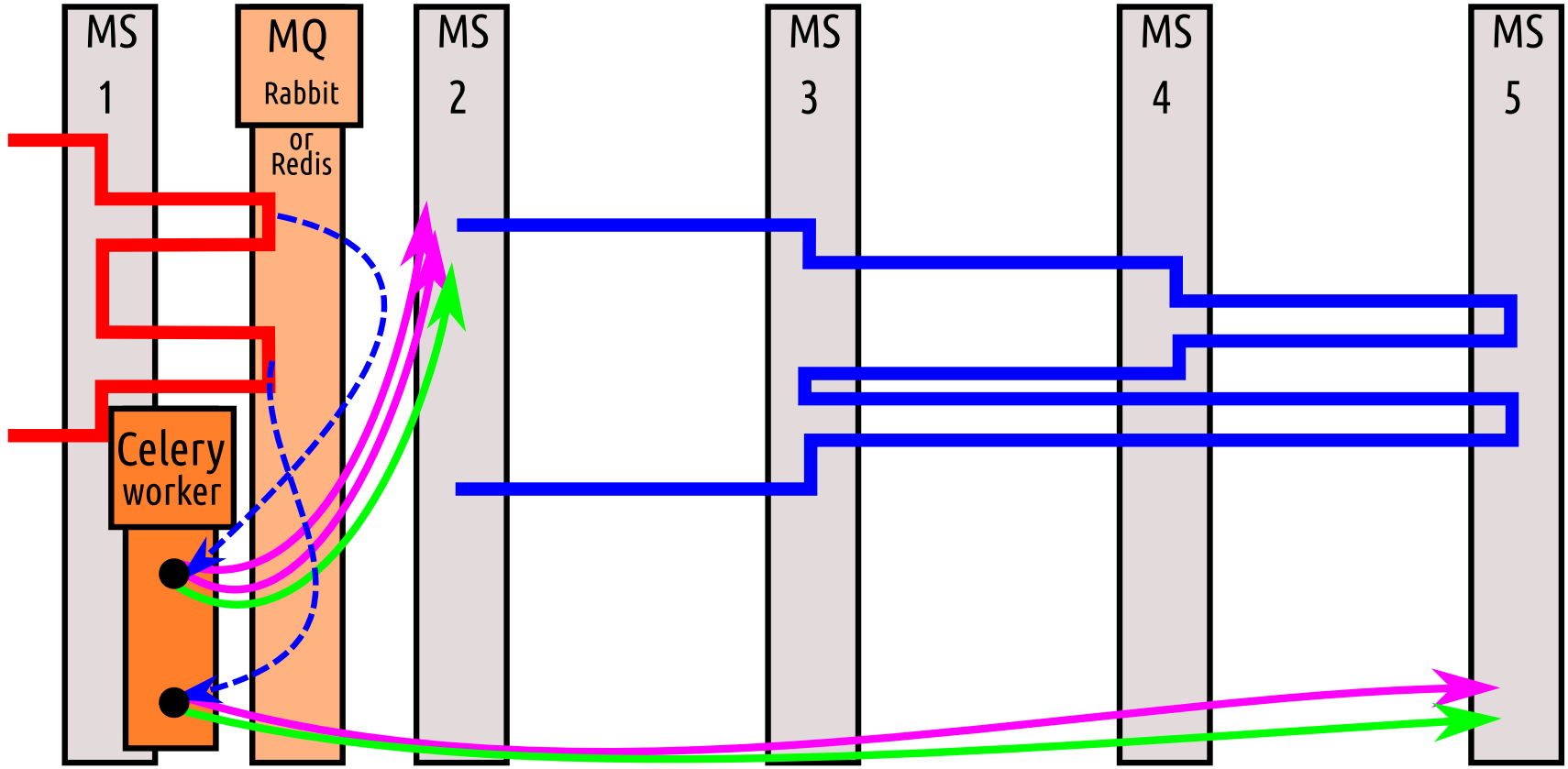


Microservices: Synchronous Communication

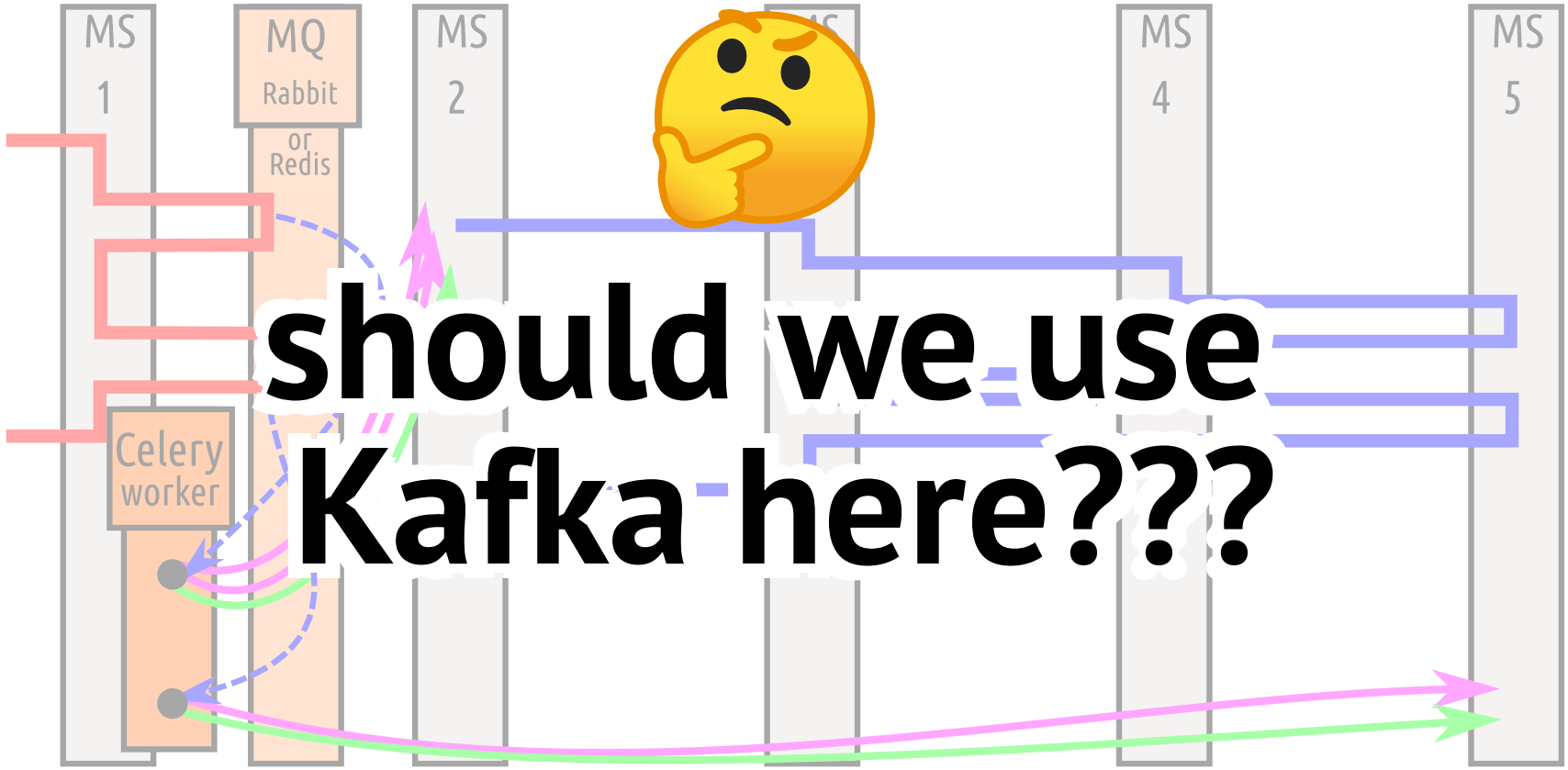
(REST API)



Microservices: Async Communication (Celery)



Microservices: Async Communication (Celery)



What is Kafka?

What is Kafka?

Distributed Streaming Platform

Initially conceived as a messaging queue... Now Apache Kafka is a full-fledged distributed event streaming platform capable of handling trillions of events a day. [↪]

Kafka for what??

Messaging

Metrics

Website Activity Tracking

Log Aggregation

Stream Processing

Event Sourcing

Commit Log

Queue

Why Kafka???

up to 2,000,000 messages per sec

or even more... [↪]

Alternatives for Kafka

RabbitMQ

ActiveMQ

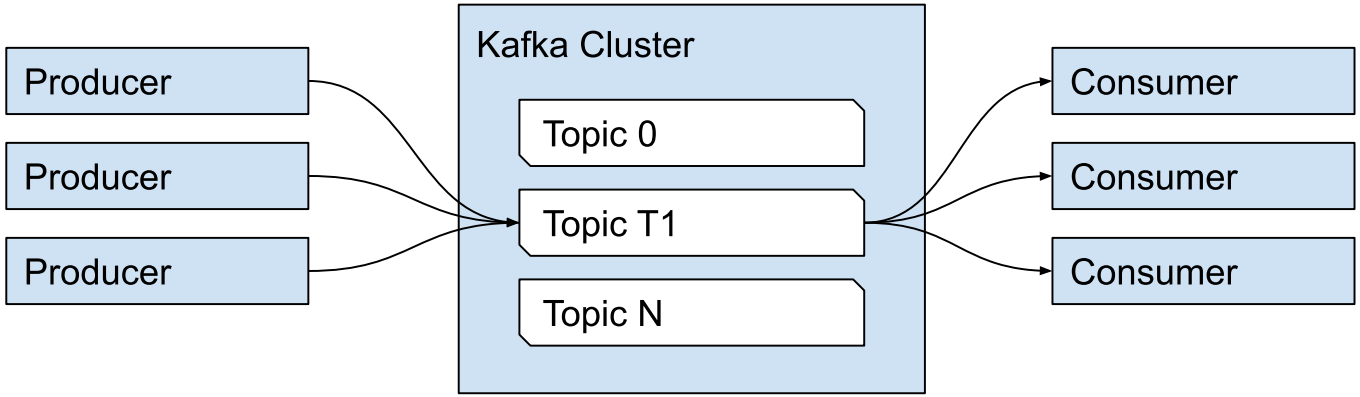
NATS

Apache Pulsar

NSQ

AWS Kinesis

Kafka: Overview



see some details on dzone.com

Kafka: Partitions for Topics

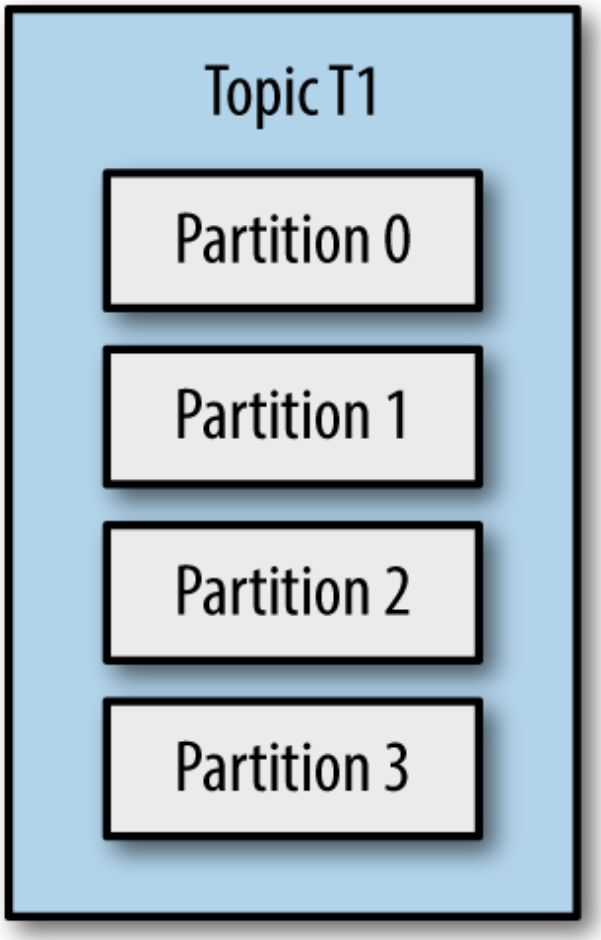
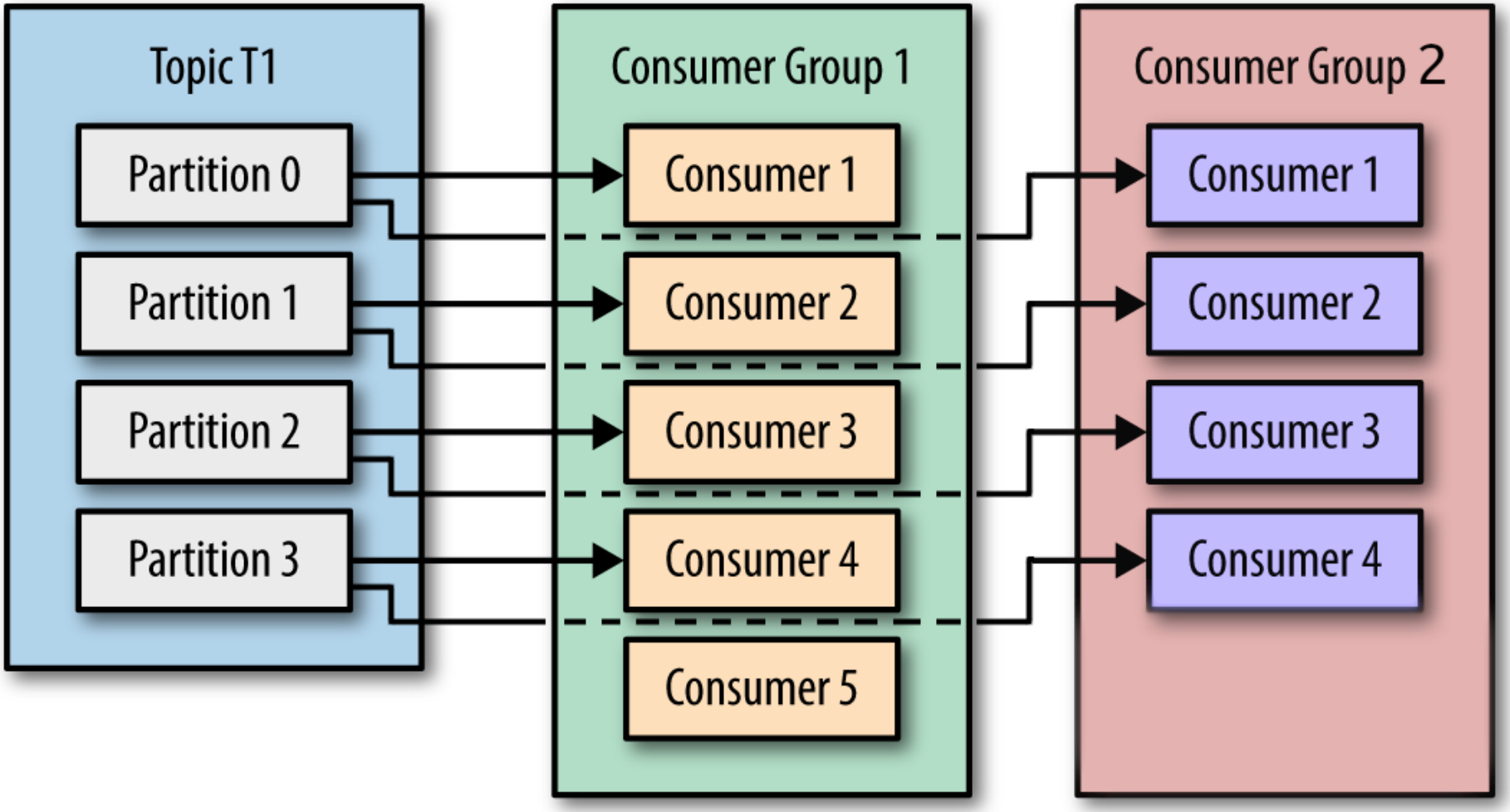


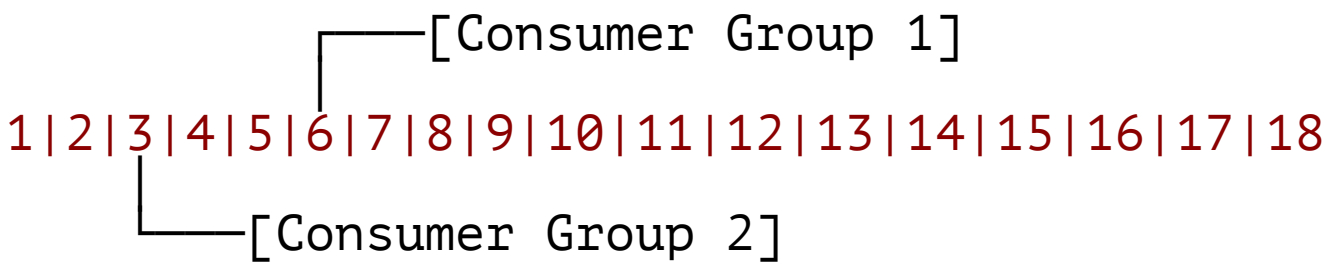
Fig. from mrsrinivas

Kafka: Partitions: few consumer groups

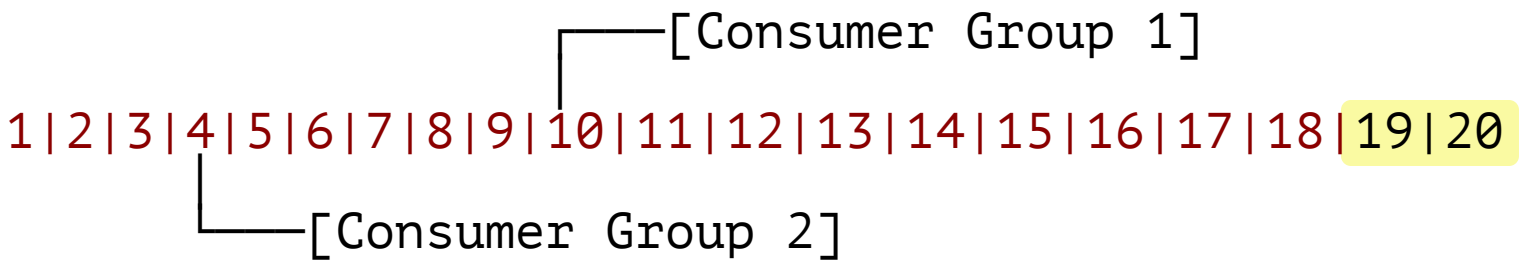


derivative of fig. from mrsrinivas

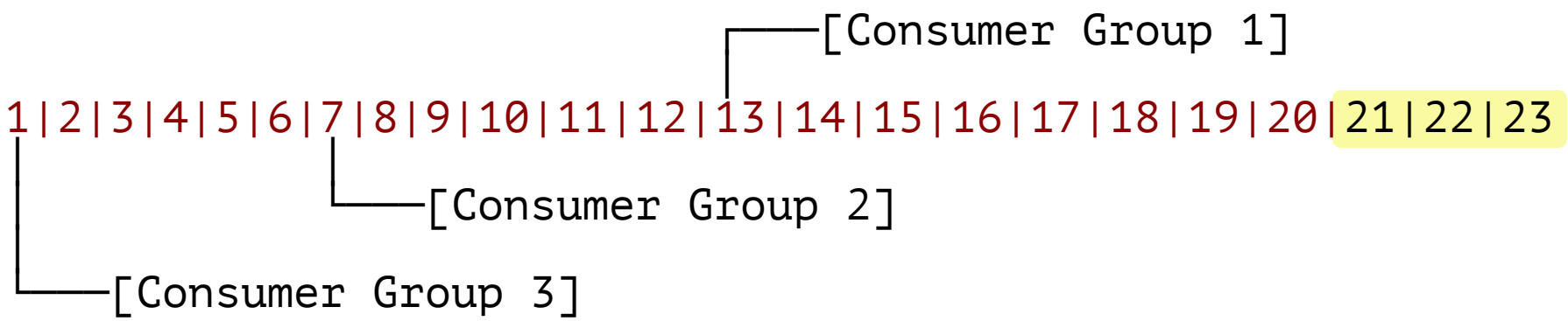
Kafka: Consumers Offsets Example for 1 Partition



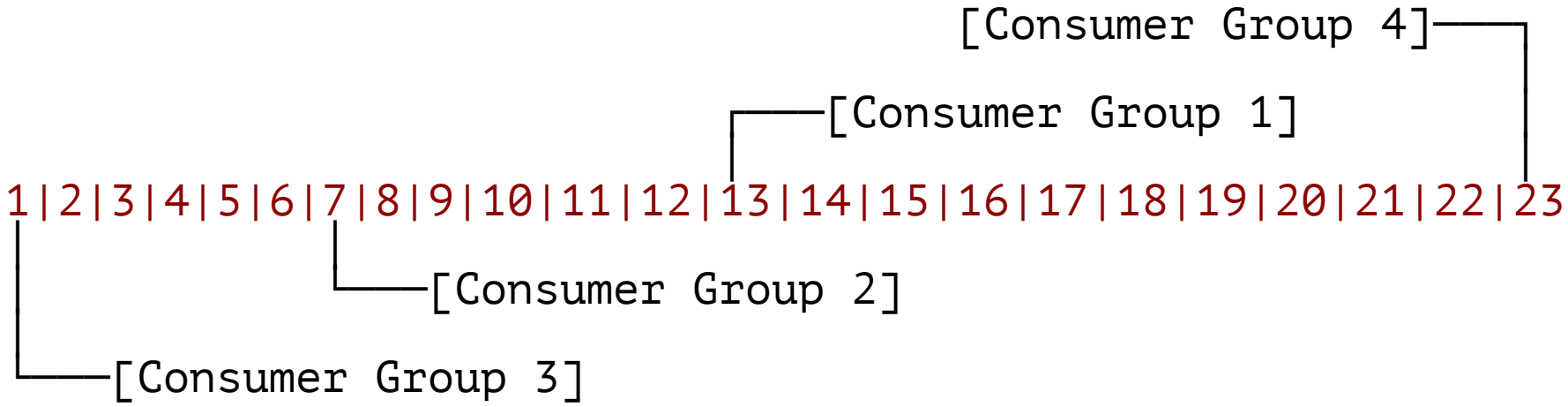
Kafka: Consumers Offsets Example for 1 Partition



Kafka: Consumers Offsets Example for 1 Partition



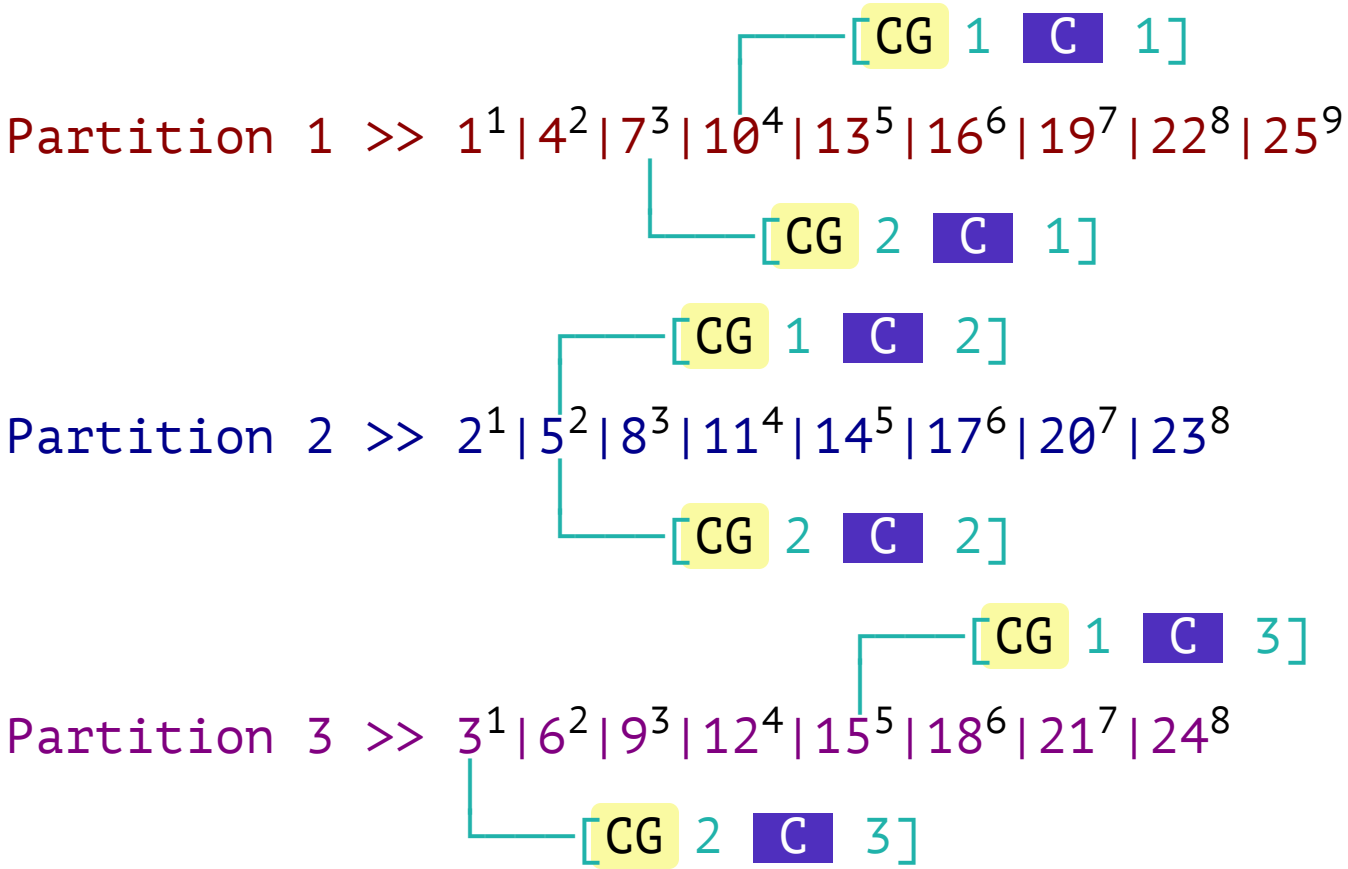
Kafka: Consumers Offsets Example for 1 Partition



Kafka: Reset Consumer Group Offsets tooling


```
Reset to Datetime

Reset from Duration

Reset to Earliest

Reset to Latest

Reset to Current Time

Reset to Offset

Shift Offset by 'n'

Reset from file
```

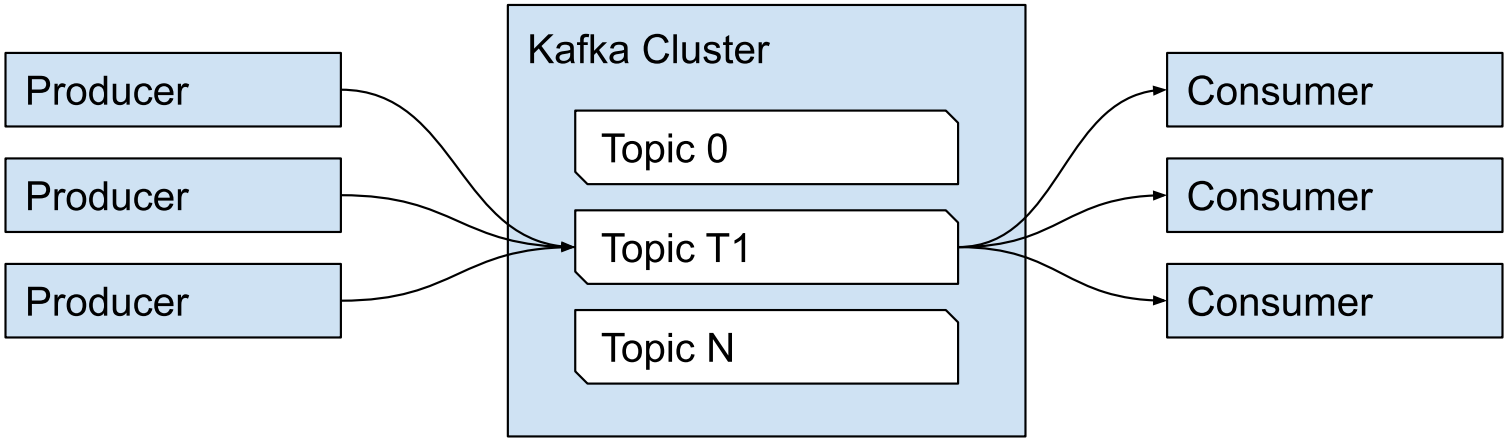
Kafka: Consumers Offsets Example for 3 Partitions



legend:

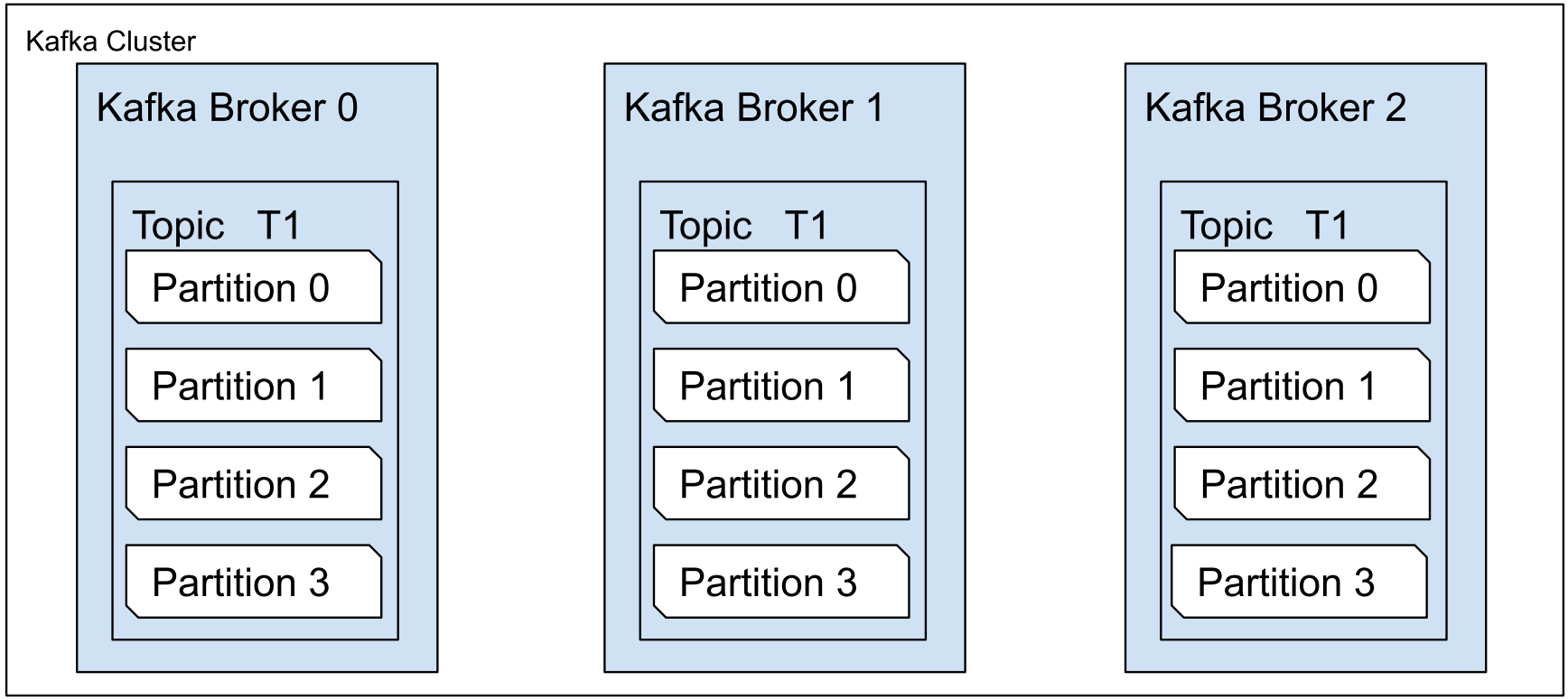
CG – Consumers Group
C – Consumer (in group)

Kafka: Overview, again :-)

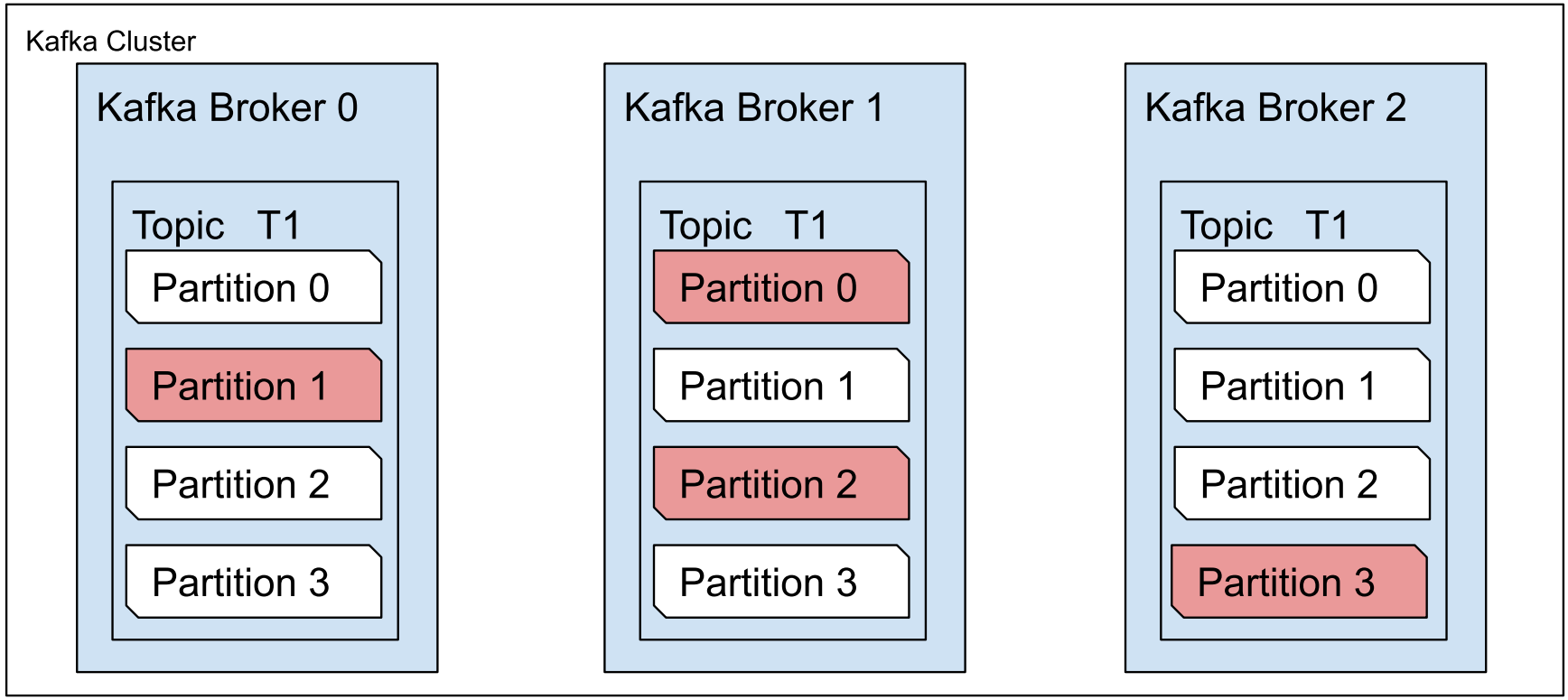


see some details on dzone.com

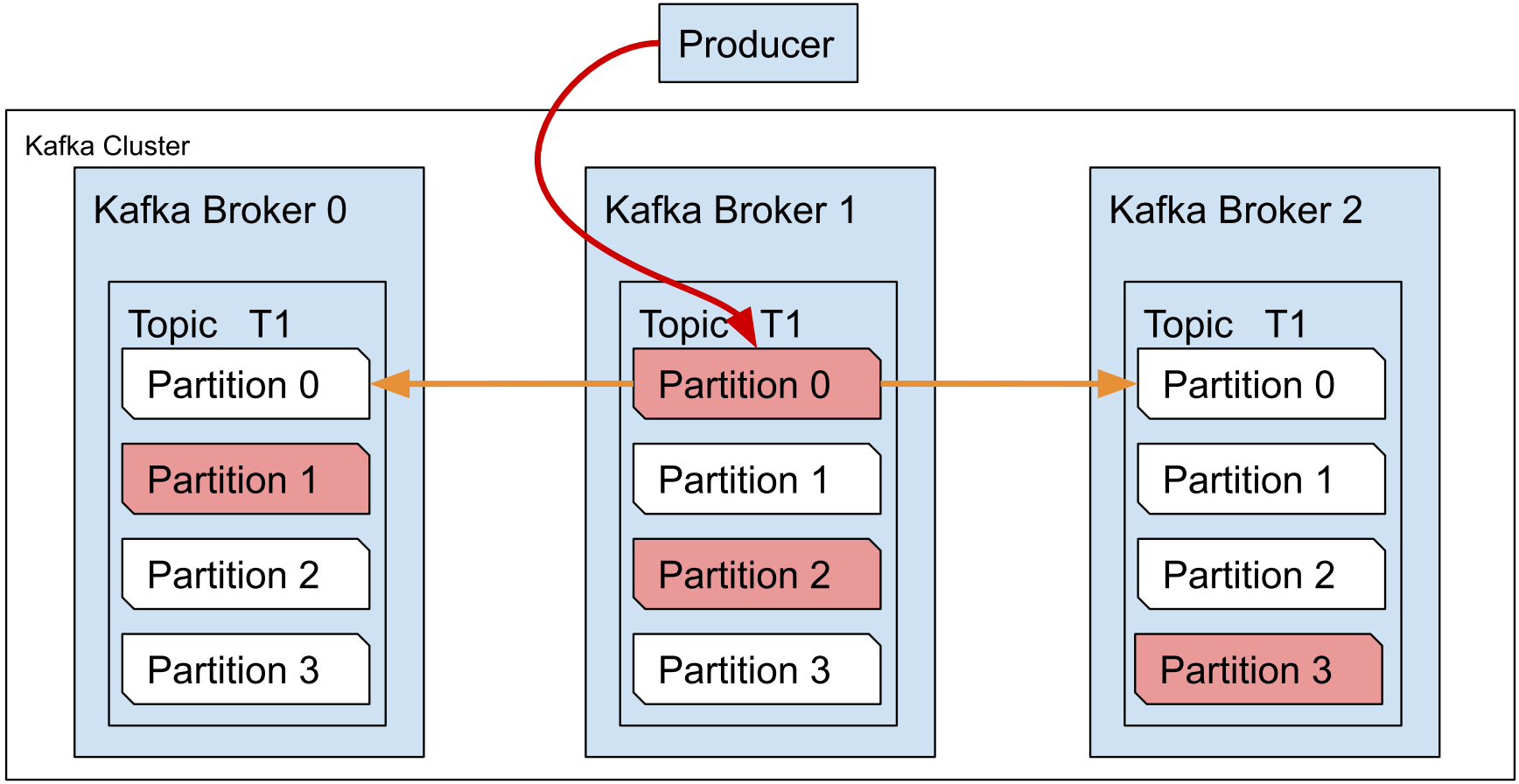
Kafka: Cluster



Kafka: Partition Leaders



Kafka: Replication



some details on dzone.com

Kafka: Message structure

key, value
byte[], byte[]

Kafka: Serializers

JSON, Protobuf, Thrift, Avro

Kafka: Schema overhead

JSON (Schemaless)

```
{user_id: 53,
timestamp: 1497842472,
address: "2 Elm St. Chattanooga, TN"}
```

74 bytes

Schema + Avro Payload

```
{   "type": "record",
    "name": "Person",
    "fields": [
        {"name": "user_id",   "type": "long"},
        {"name": "timestamp", "type": "long"},
        {"name": "address",   "type": "string"}]}
```

204 bytes

35	59474328	19
3220456c6d2053 742e2043686174 74616e6f6f6761 2c20544e		

34 bytes

Schema ID + Avro Payload

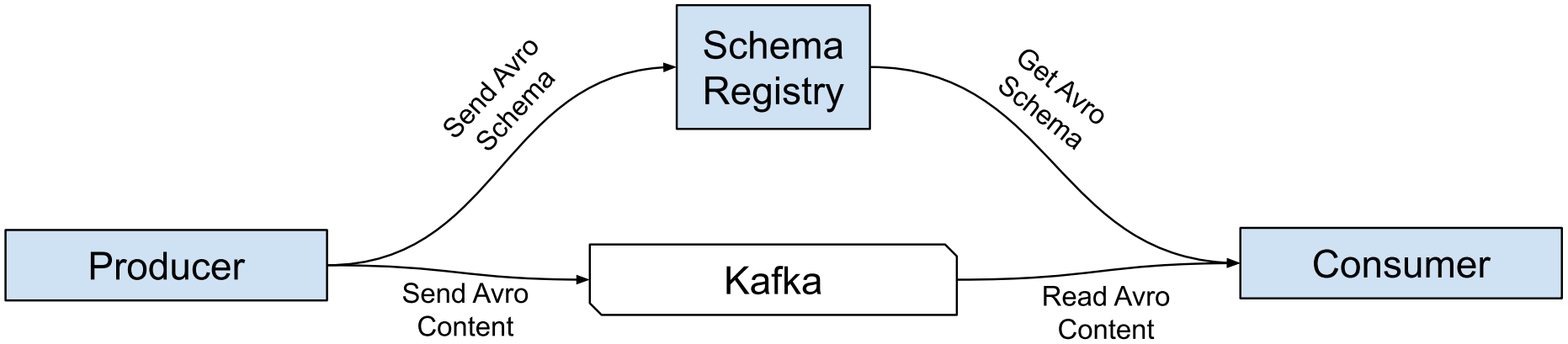
21	35	59474328	19
	3220456c6d2053 742e2043686174 74616e6f6f6761 2c20544e		

4 bytes

34 bytes

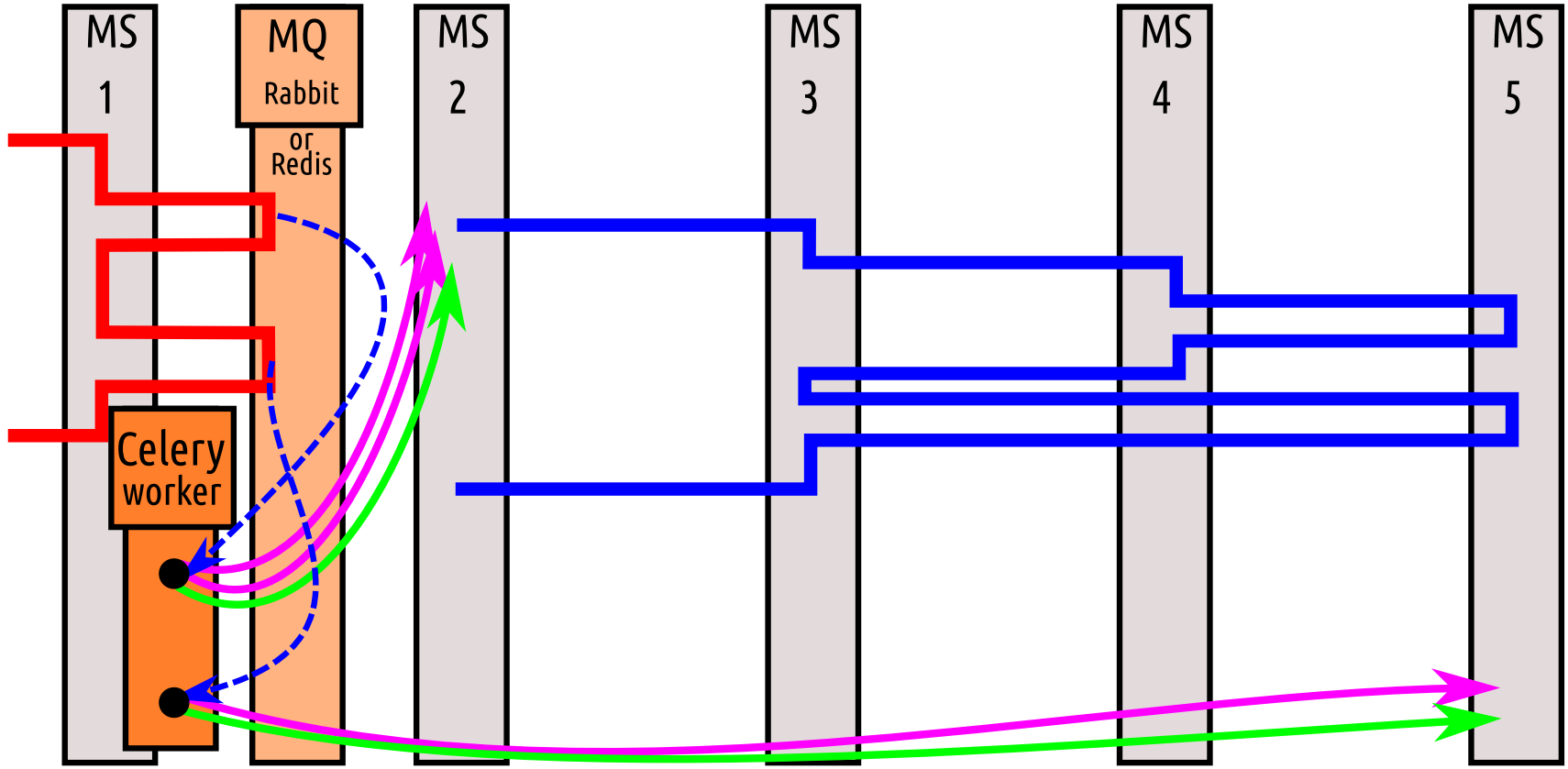
Fig. from confluent.io

Kafka: Schema Registry Concept

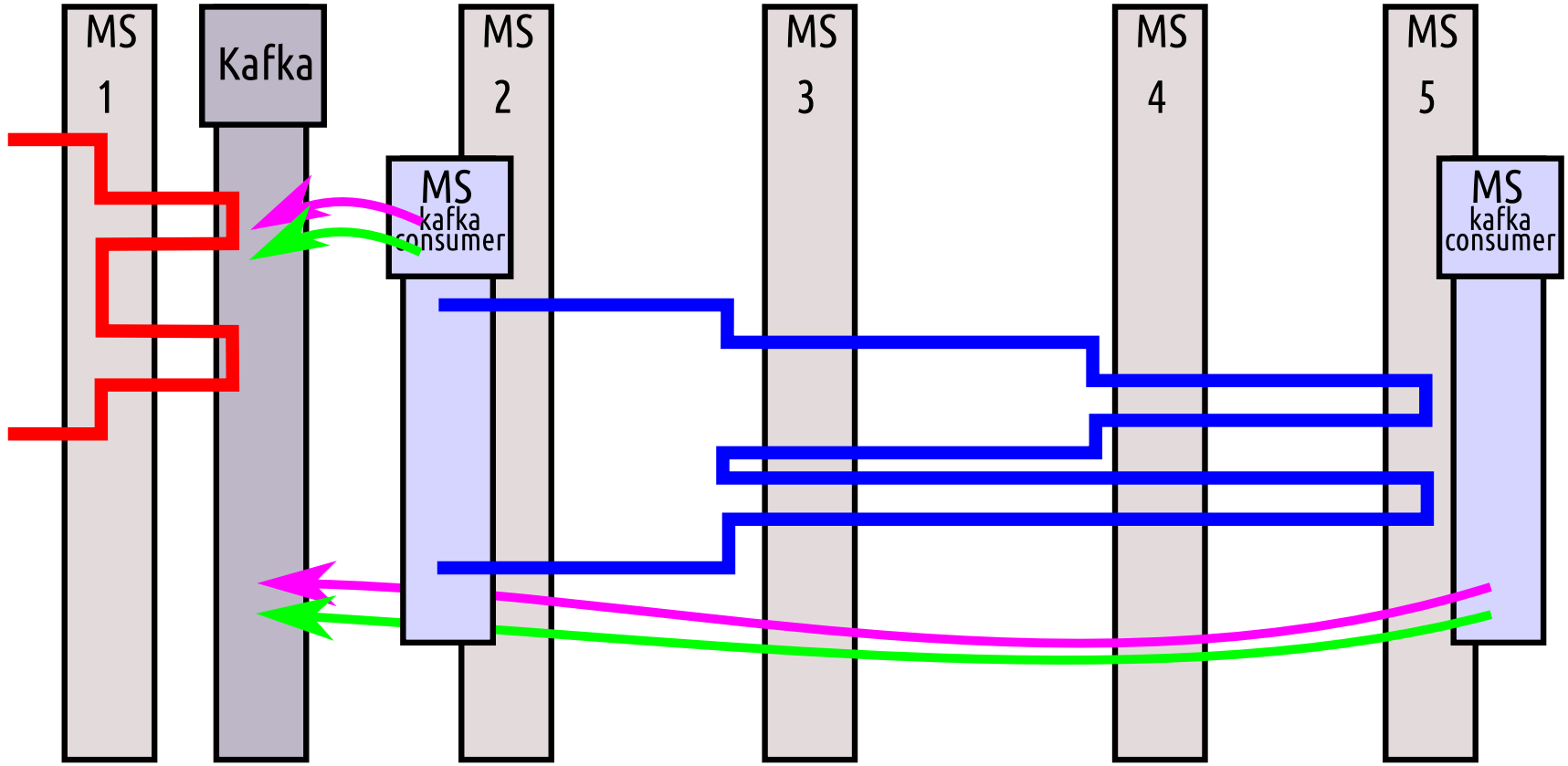


Microservices?

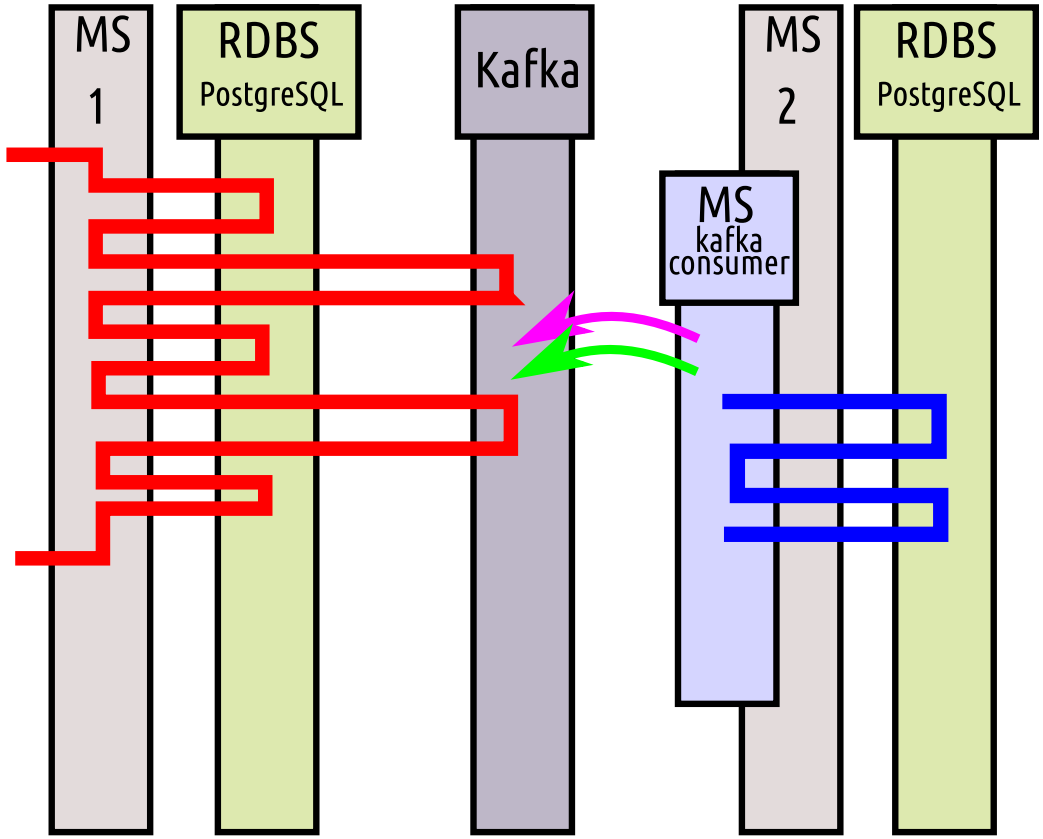
Microservices: Async Communication (Celery)



Microservices: Async Communication (Kafka)



Microservices: Async. What about databases?



Relational Database ⟶ Kafka

debezium

Stream data from RDBS to Kafka

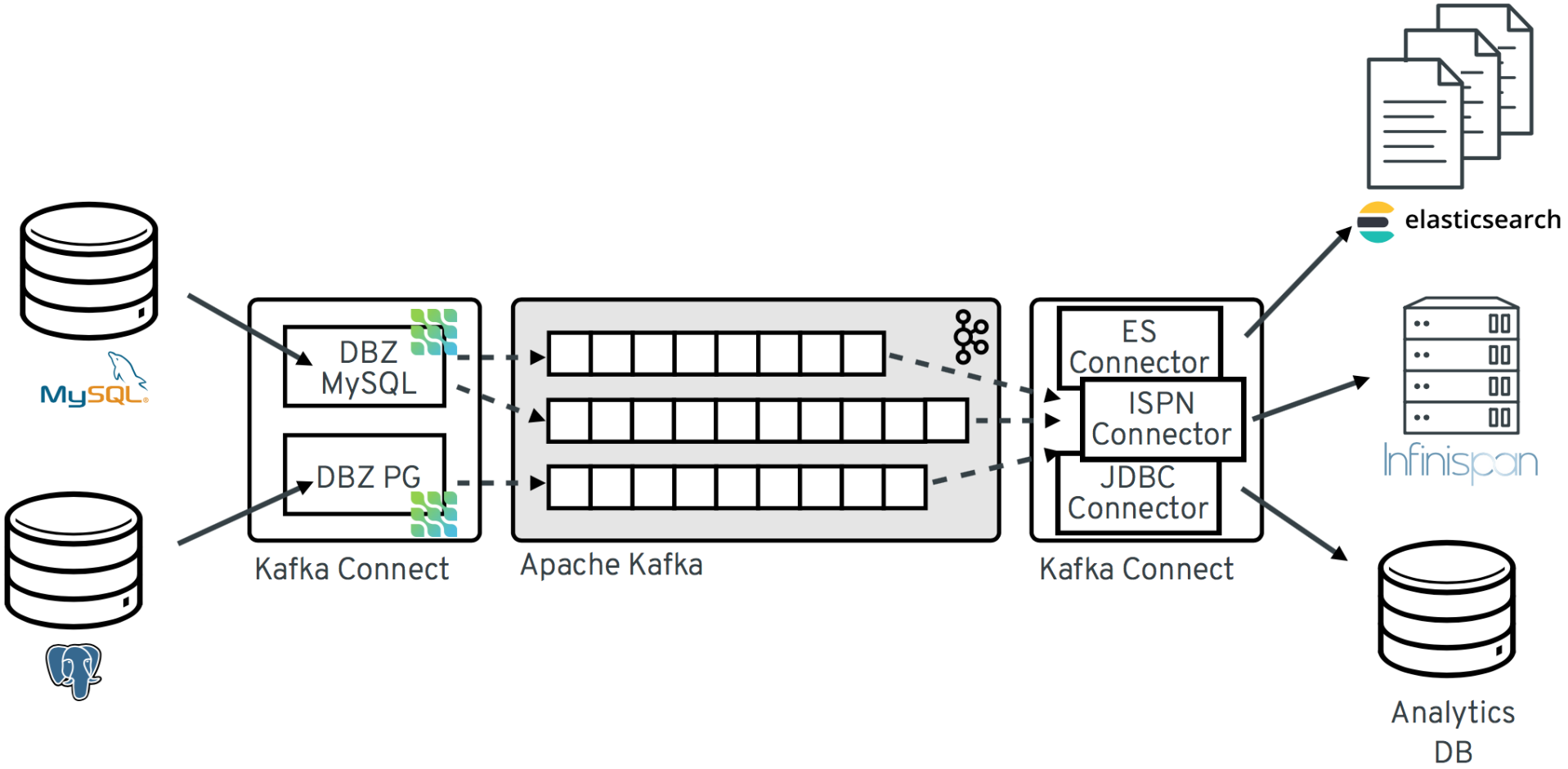


Fig. from debezium.io

Stream data from PostgreSQL to Kafka

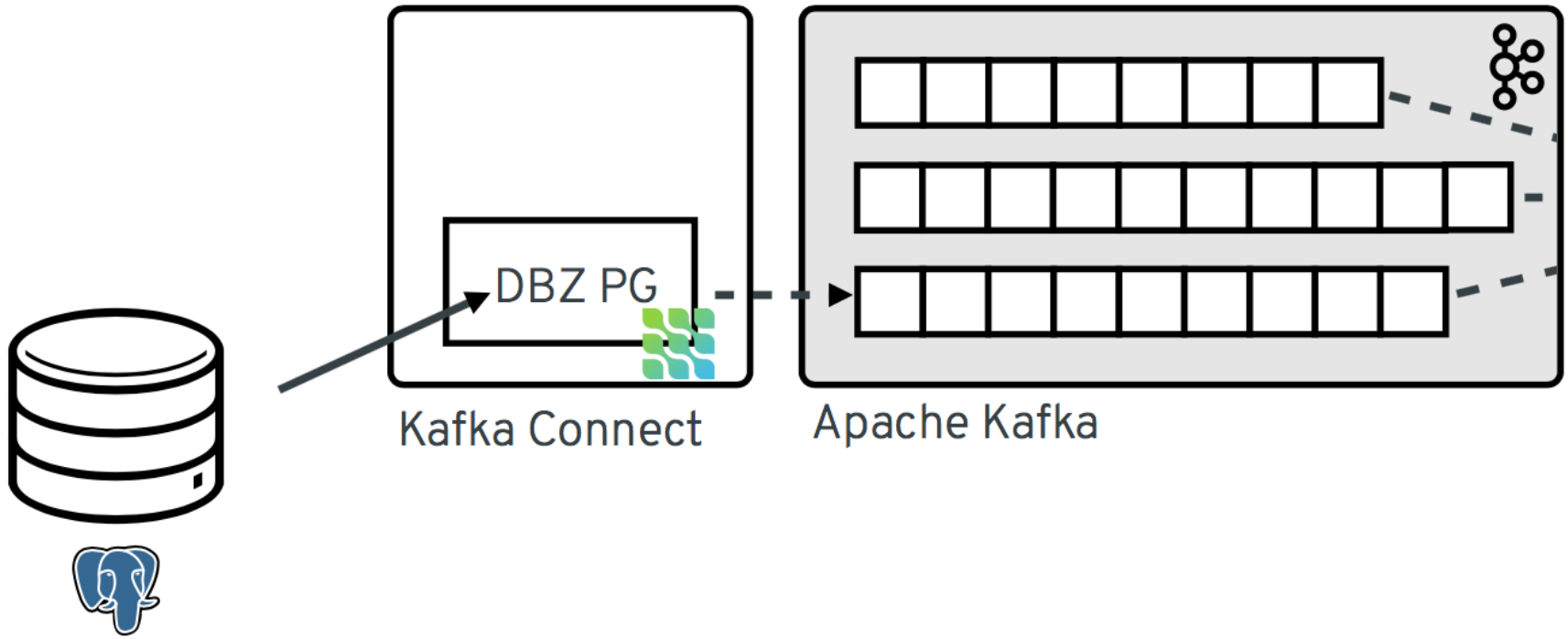
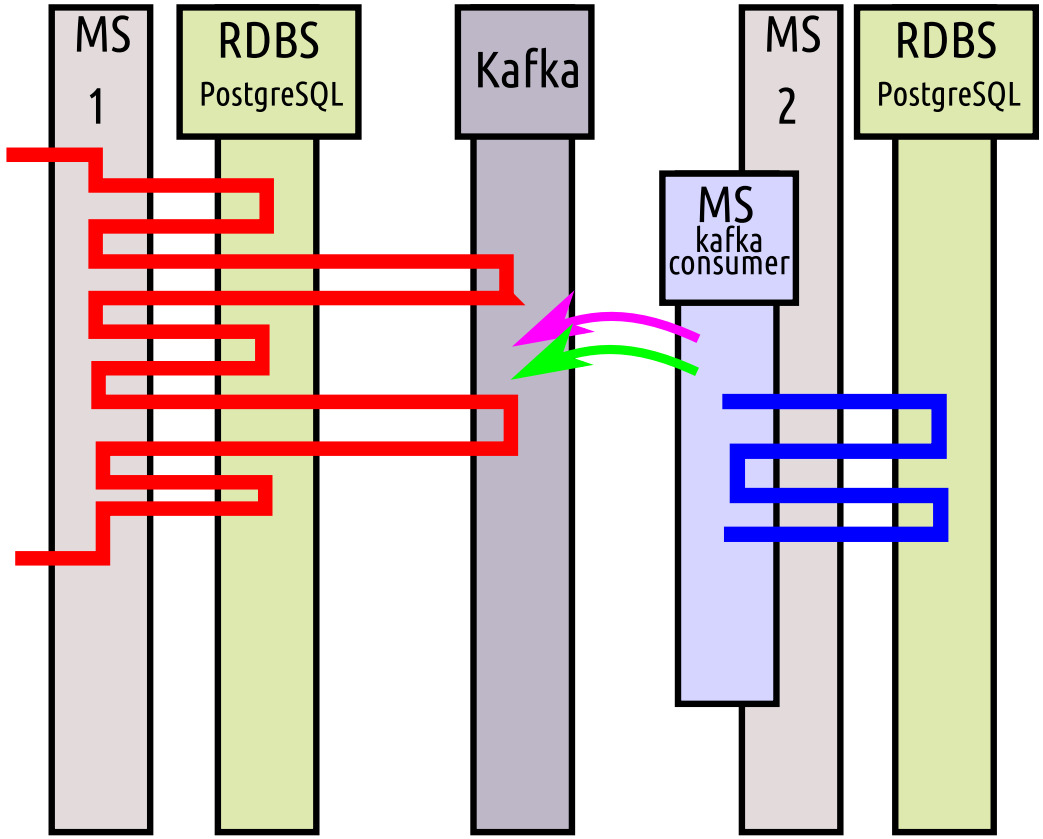
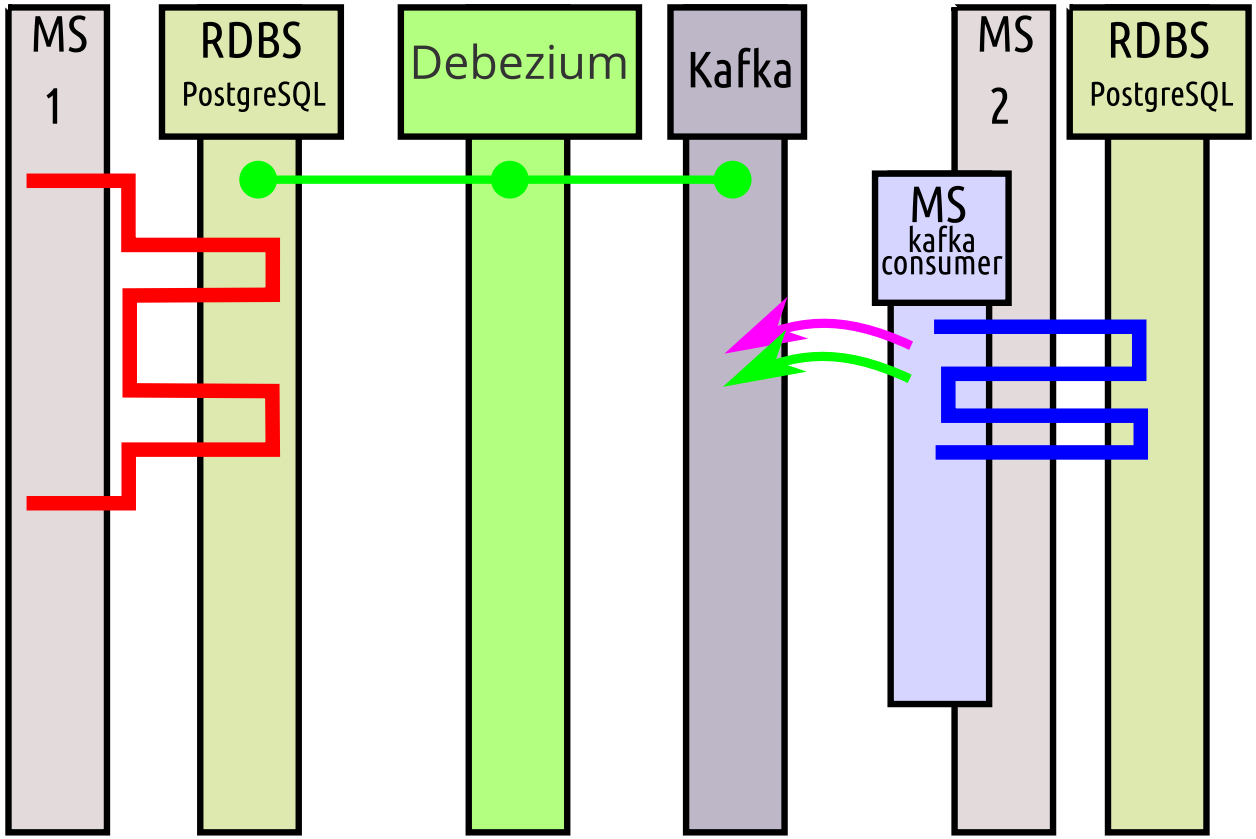


Fig. from debezium.io

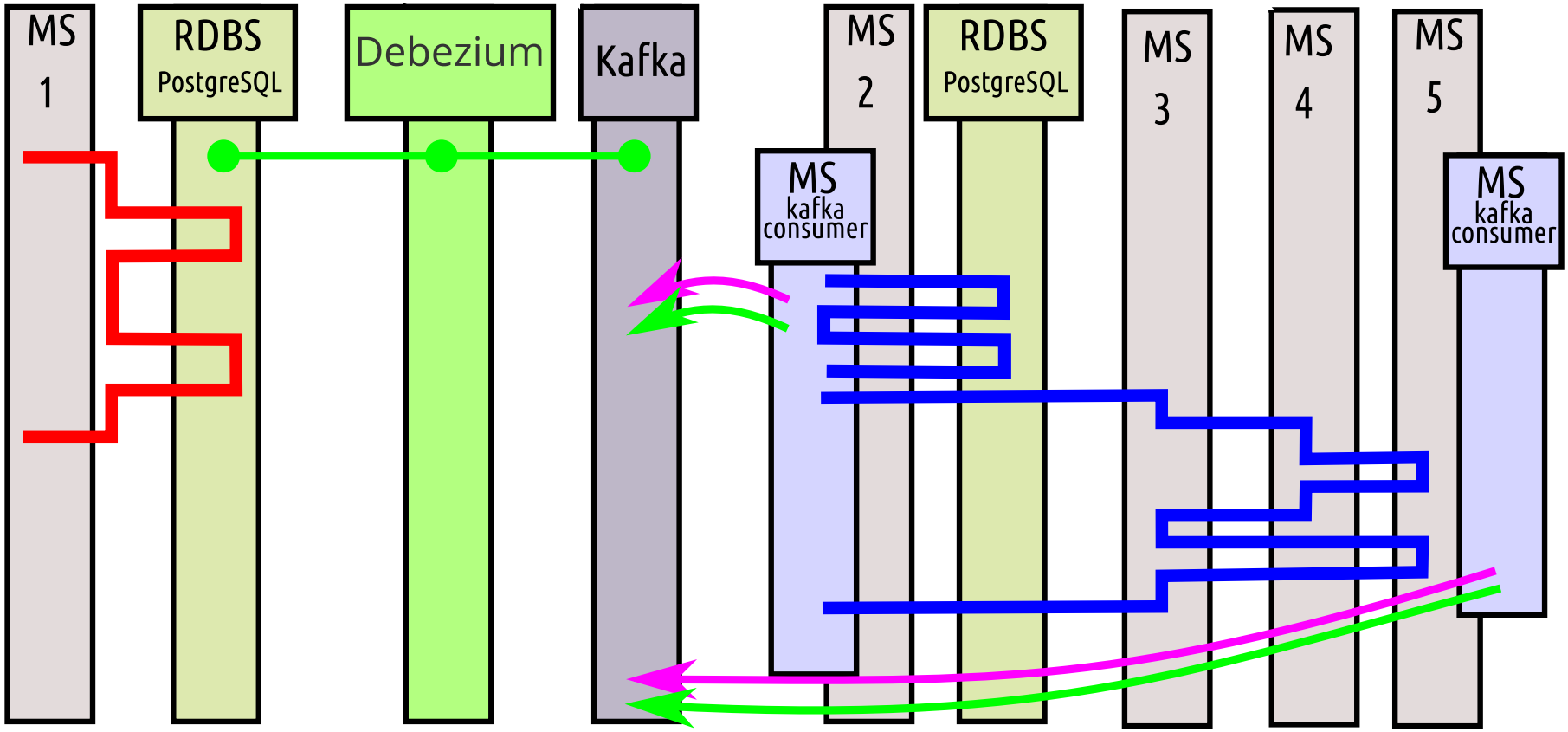
Microservices: Async. What about databases?



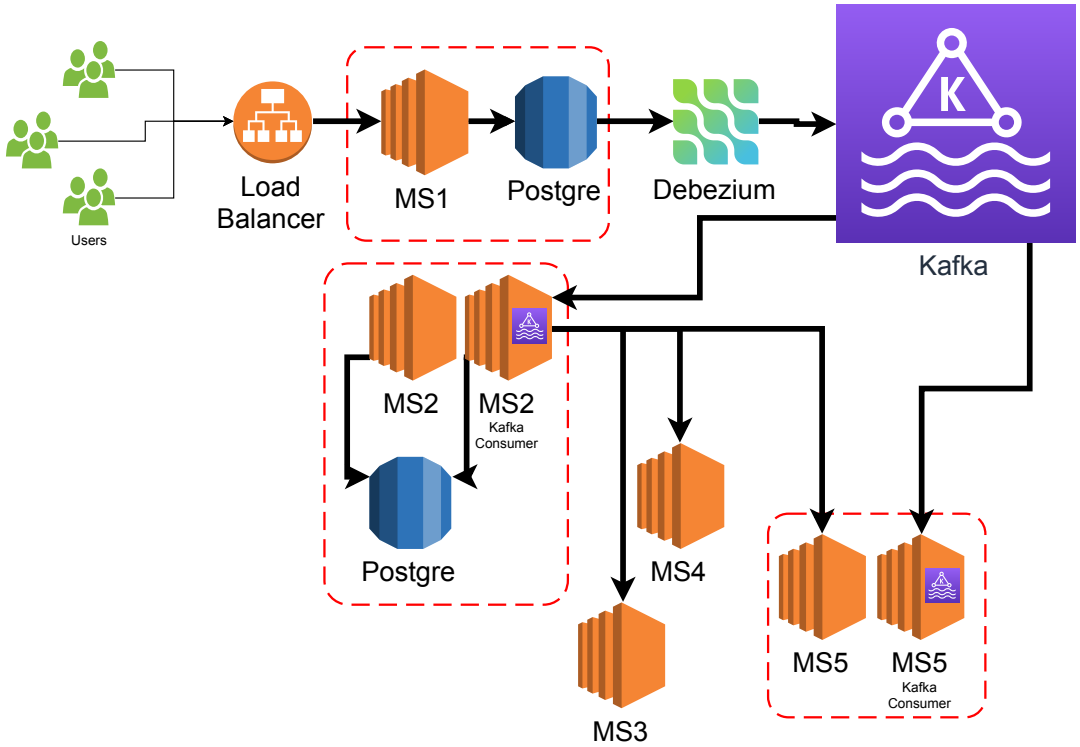
Microservices: Async. Debezium



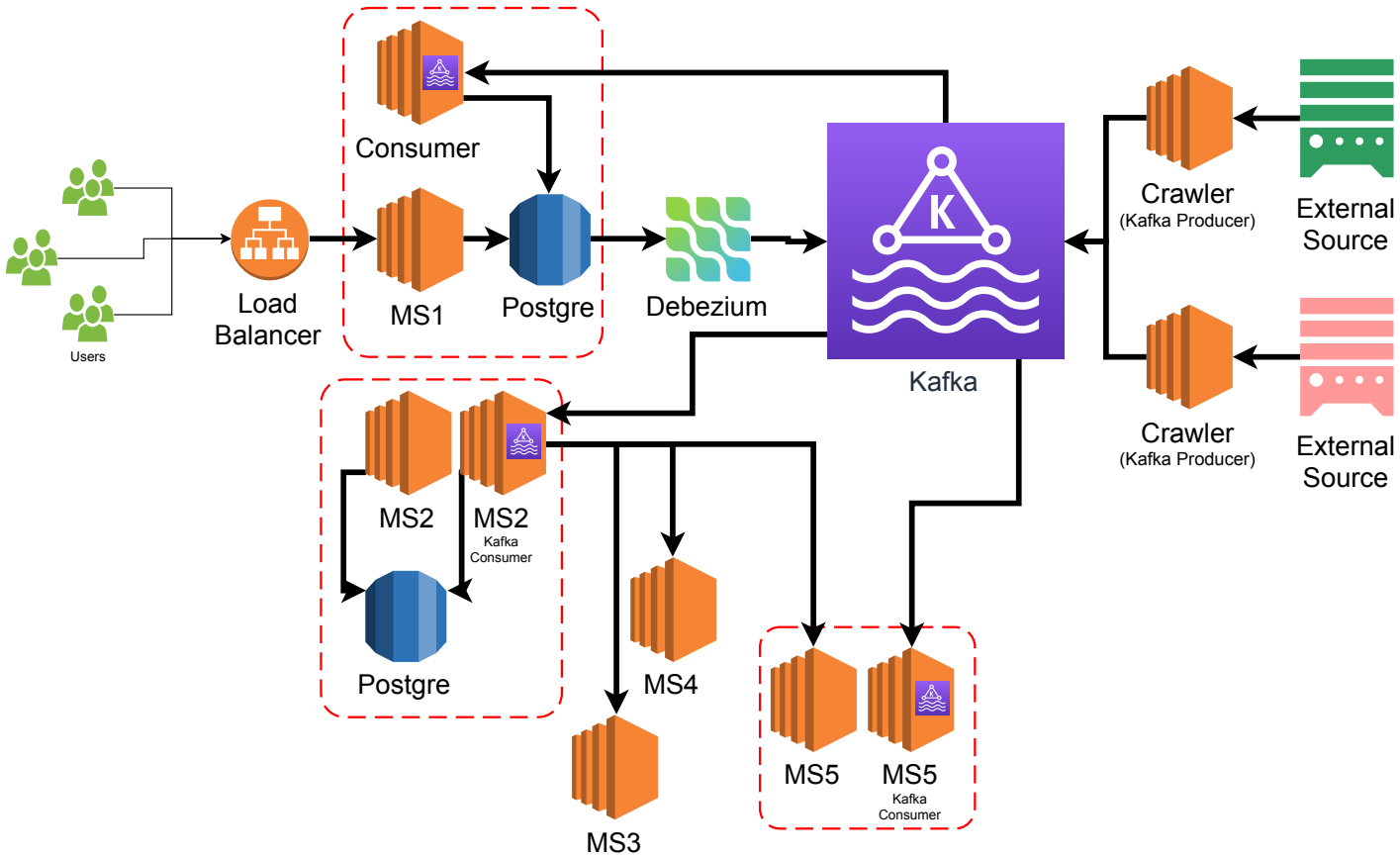
Microservices: Async. Debezium. Full



Application Diagram



Application Diagram. External Data Handling



What about Python?

kafka-python

pykafka

confluent-kafka-python

aiokafka

What to choose?

	kafka-python	pykafka	confluent-kafka
Stars	~ 3 400	~ 1 000	~ 1 500
Contributors*	0 / 4 / 13 / 186	1 / 5 / 10 / 77	0 / 1 / 6 / 57
Releases track	fine	hm... (2018???)	fine
Development track	fine	hm...	fine

```
*
a / b / c / d
a — more 1k commits
100 ⩽ b < 1000
10 ⩽ c < 100
d — all commits
```

What about Throughput?

		time (seconds)	MBs/s	Msgs/s
producer	confluent_kafka_producer	5.4	17	183 000
producer	pykafka_producer_rdkafka	16	6.1	64 000
producer	pykafka_producer	57	1.7	17 000
producer	python_kafka_producer	68	1.4	15 000
consumer	confluent_kafka_producer	5.4	17	183 000
consumer	pykafka_producer_rdkafka	16	6.1	64 000
consumer	pykafka_producer	57	1.7	17 000
consumer	python_kafka_producer	68	1.4	15 000

benchmarking by Matthew Rocklin

Let's code!


```
from confluent_kafka import Producer
import sys
if __name__ == '__main__':
    broker = sys.argv[1]
    topic = sys.argv[2]

    producer = Producer(**{'bootstrap.servers': broker})
    delivery_callback = lambda err, msg: print(err or msg)

    for something in range(1000):
        try:
            producer.produce(topic, str(something),
                             callback=delivery_callback)
        except BufferError:
            print("Local producer queue is full, try again")
        # Serve delivery callback queue.
        producer.poll(0)
        # Wait until all messages have been delivered
        producer.flush()
```

Let's code!


```
from confluent_kafka import Consumer, KafkaException
import sys
if __name__ == '__main__':
    broker, group, topics = sys.argv[1], sys.argv[2], sys.argv[3:]
    conf = {'bootstrap.servers': broker, 'group.id': group,
            'auto.offset.reset': 'earliest'}
    clb = lambda consumer, p: print('Assigned partition:', p)

    consumer = Consumer(conf)
    consumer.subscribe(topics, on_assign=clb)
    try:
        while True:
            msg = c.poll(timeout=1.0)
            if msg is None:
                continue
            if msg.error():
                raise KafkaException(msg.error())
            else:
                print(msg.topic(), msg.partition(), msg.offset(),
                      msg.key(), msg.value())
    finally:
        consumer.close() # ... to commit final offsets.
```

Conclusion?

Thanks!

Questions?