Streaming with Structure

Kate Stanley

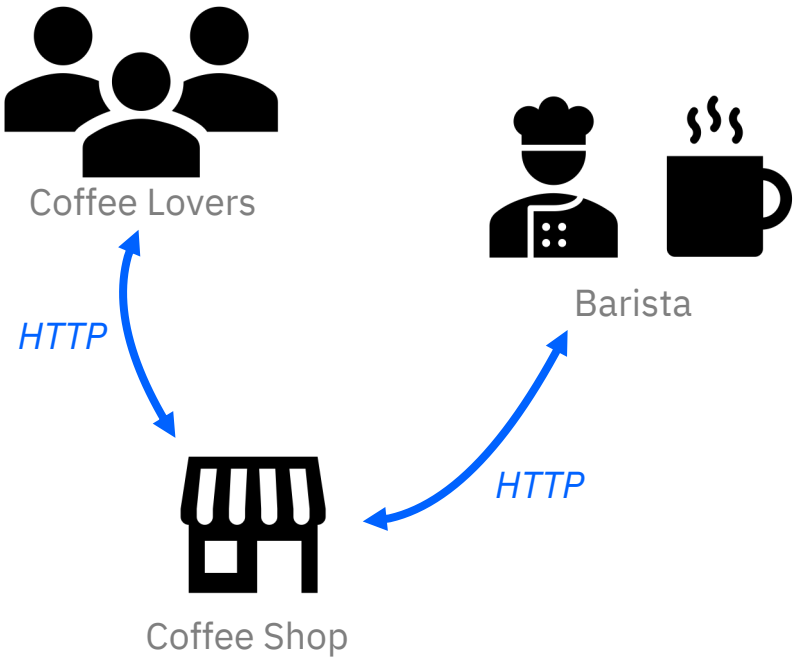
Apache Kafka

IBM

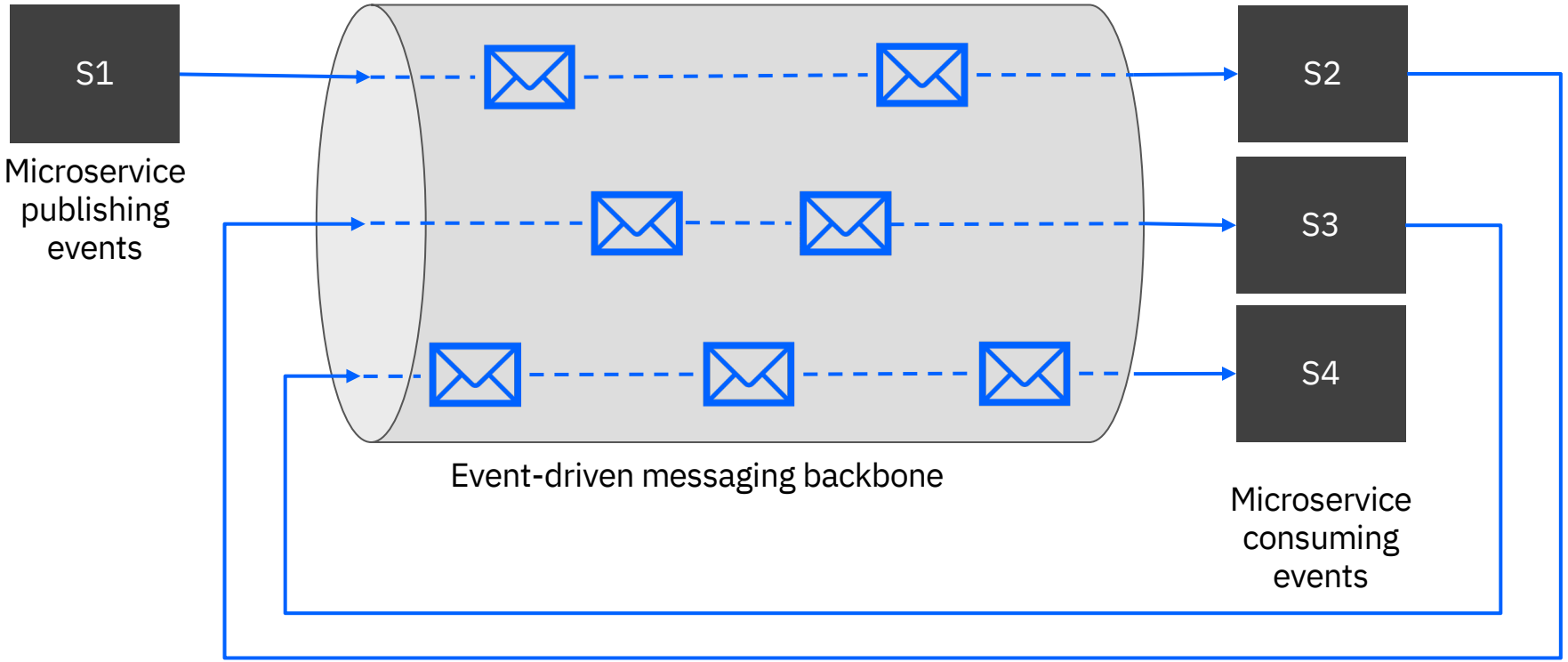
Let’s get some coffee…

Barista Example:

https://github.com/cescoffier/quarkus-coffeeshop-demo



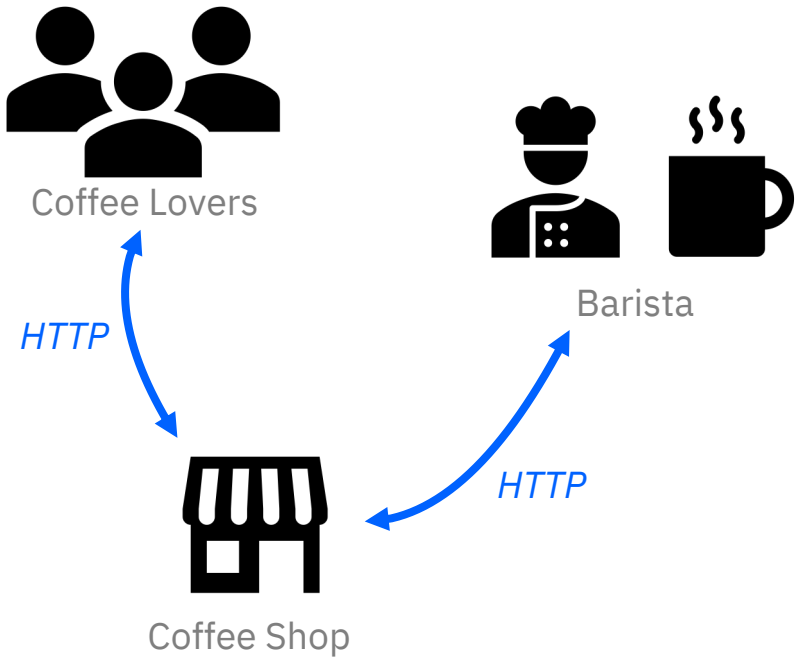
Event Driven Architecture

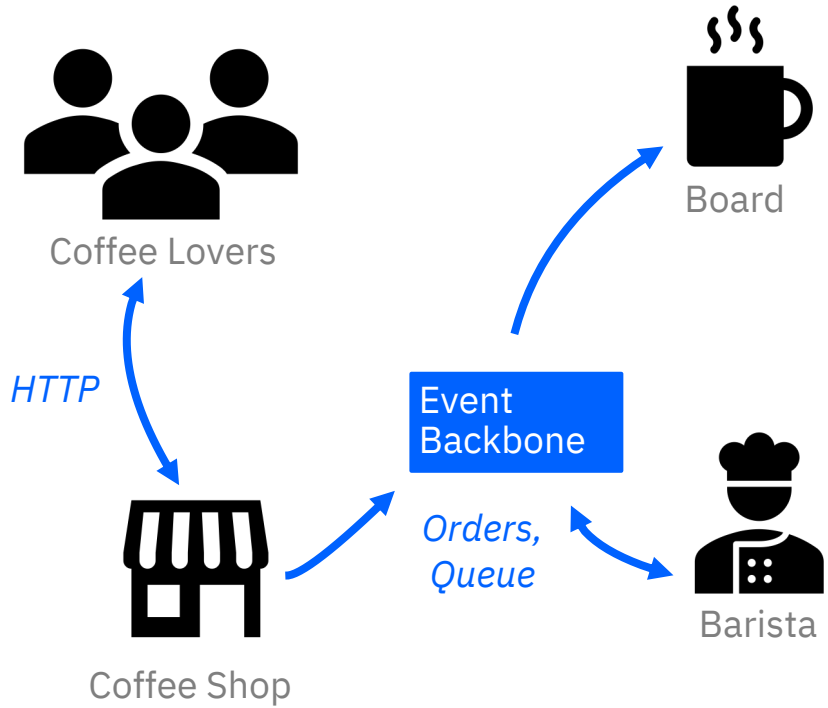


http://ibm.biz/AdvantagesOfEDA

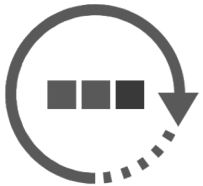
Barista Example:

https://github.com/cescoffier/quarkus-coffeeshop-demo



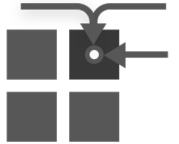


Apache Kafka is an **open source, distributed streaming platform**

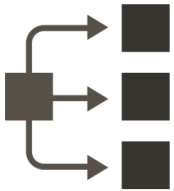


Stream history

Immutable data



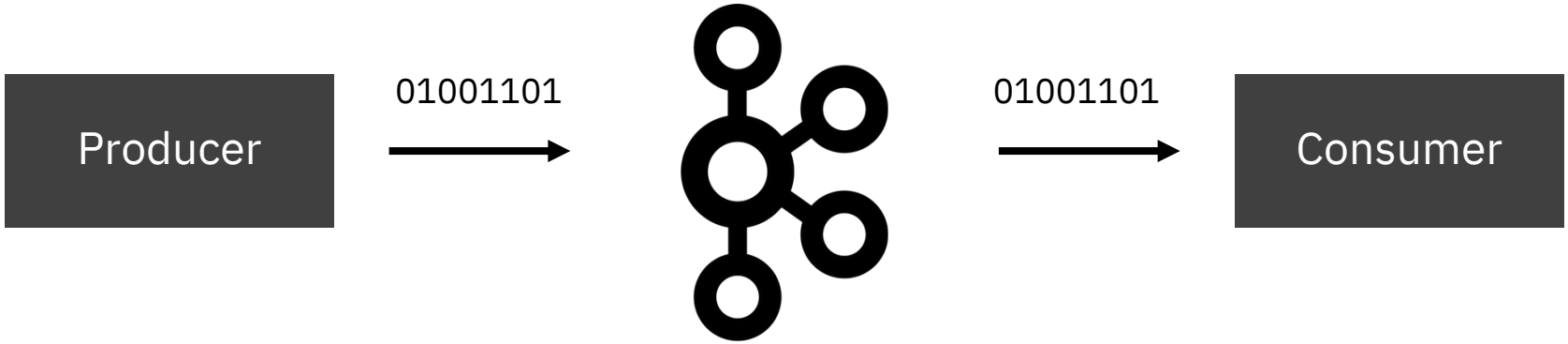
Highly available



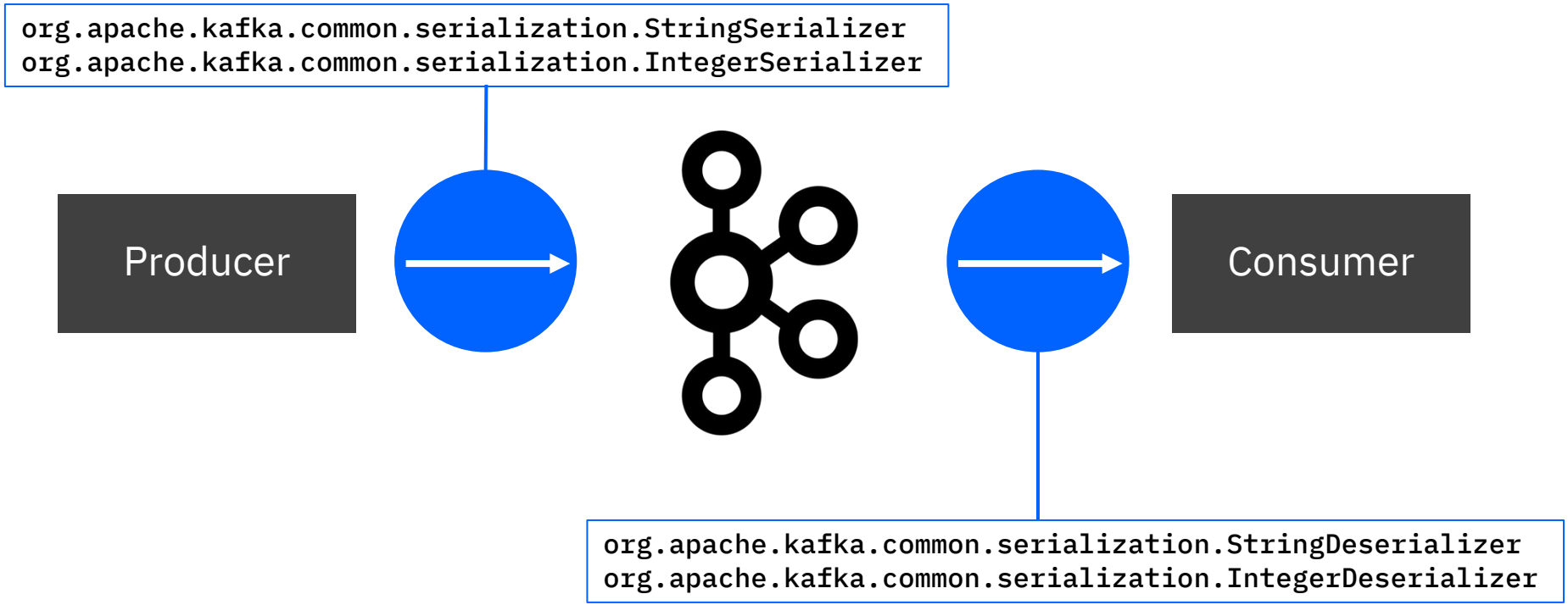
Scalable consumption

Scalable

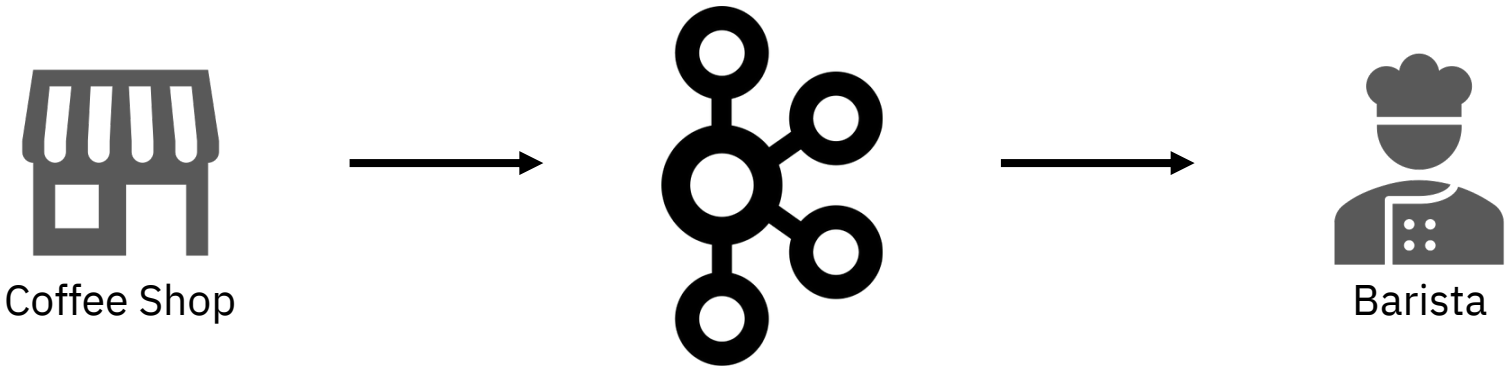
Data is stored in bytes



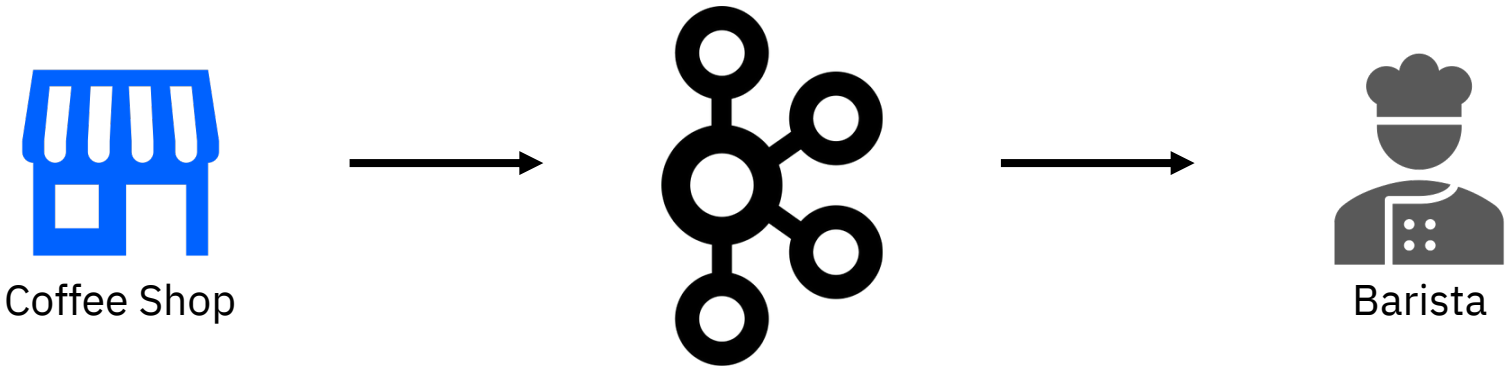
Data is stored in bytes



StringSerializer

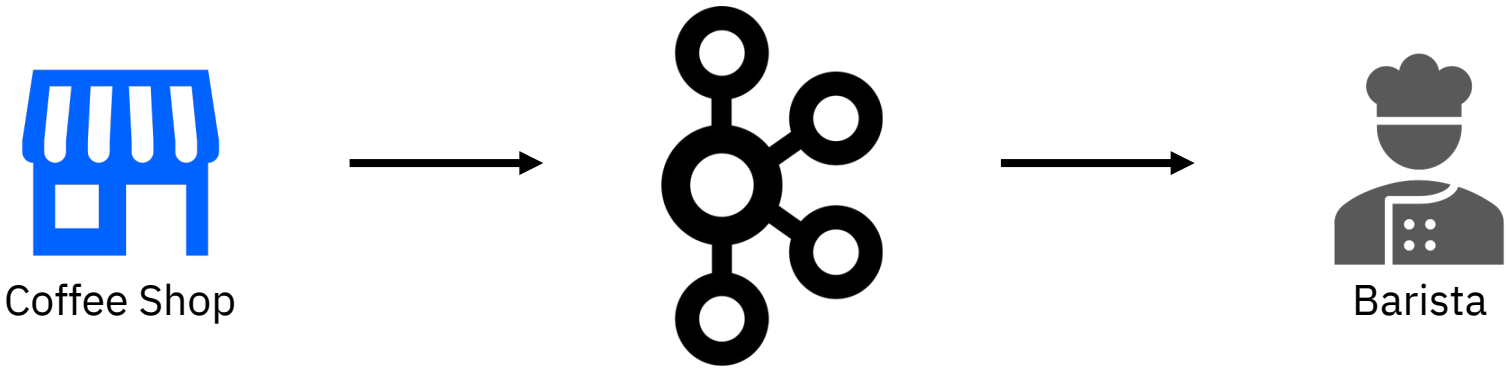


StringSerializer



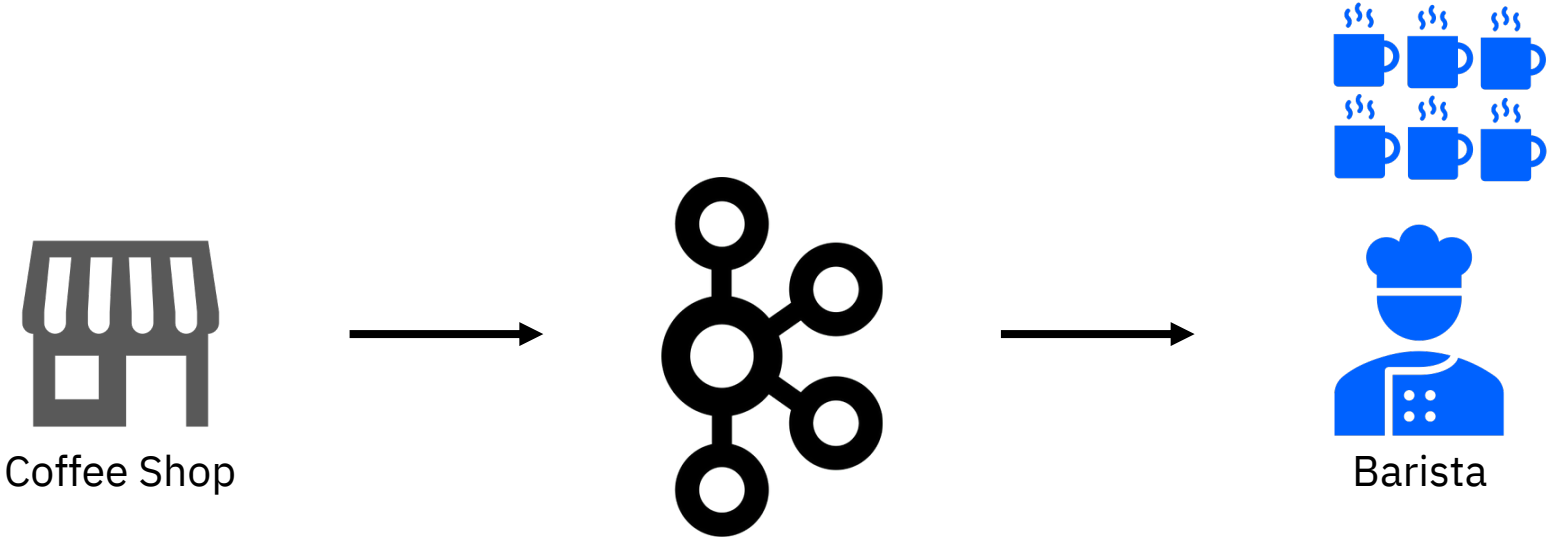
“id,product,customer,size”

StringSerializer



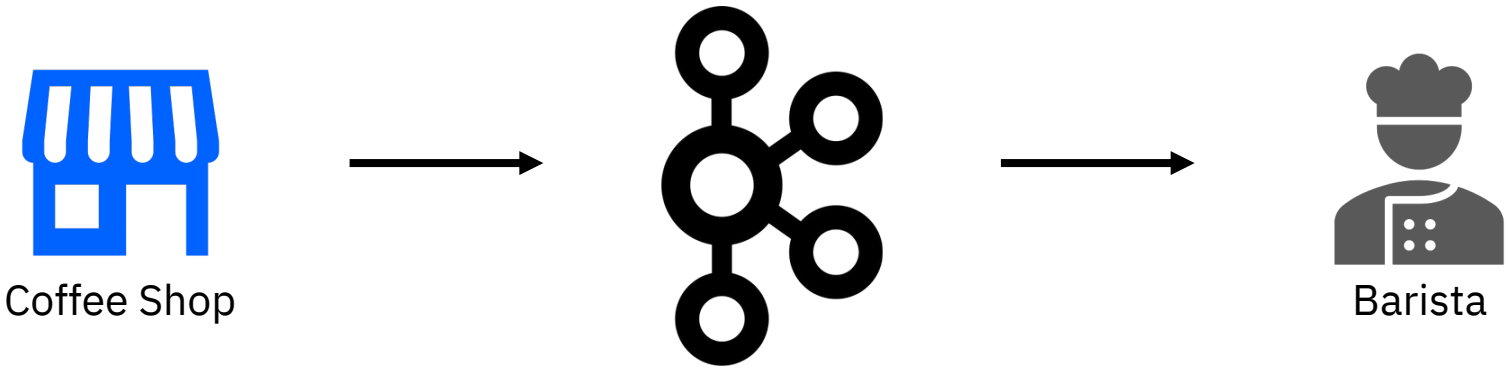
“1234,hot chocolate,Kate,small”

StringSerializer



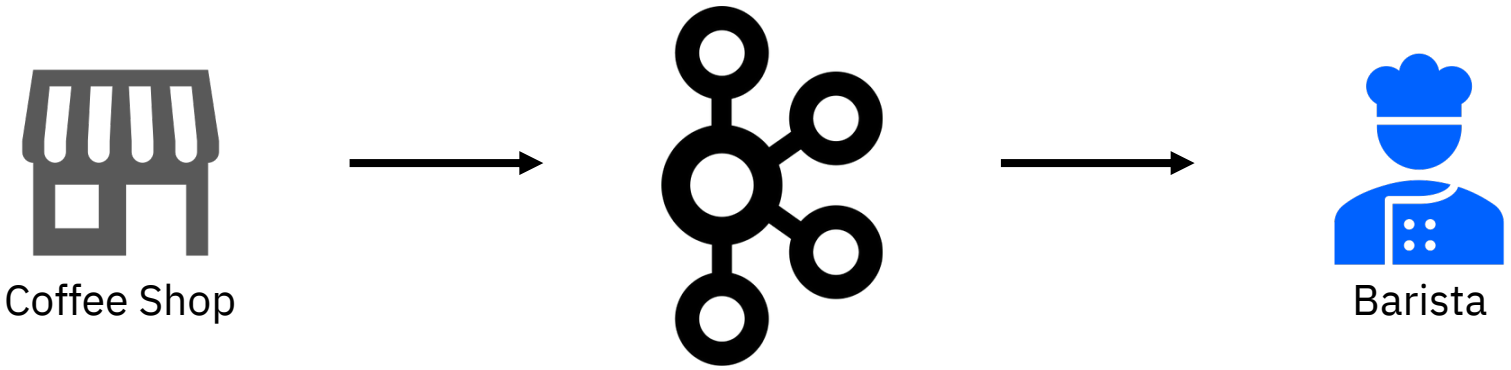
“1234,hot chocolate,Kate,small”

StringSerializer



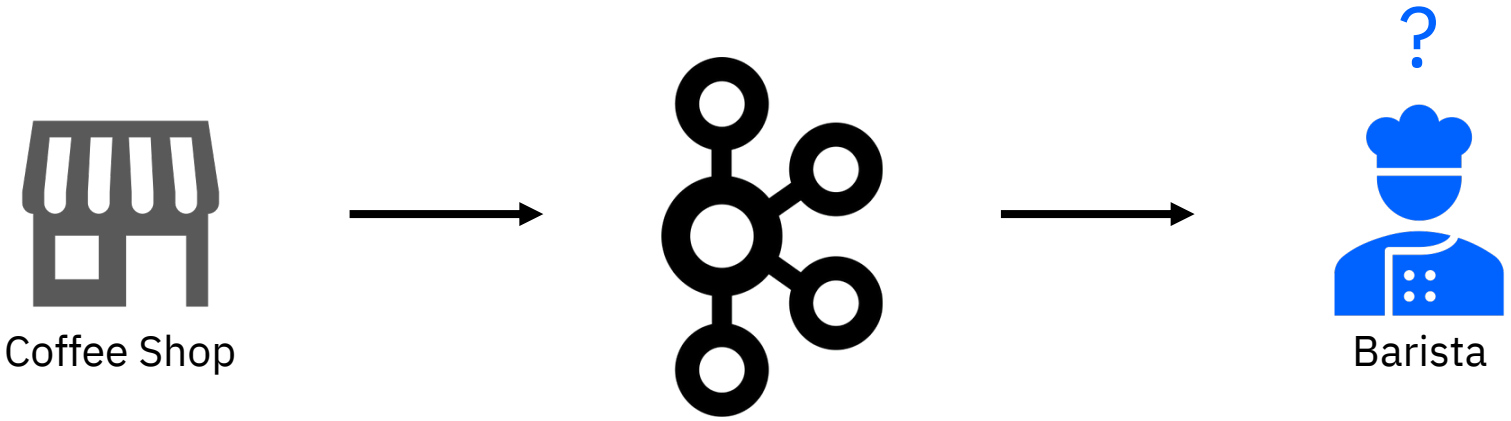
“id,product,marshmallows,customer,size”

StringSerializer



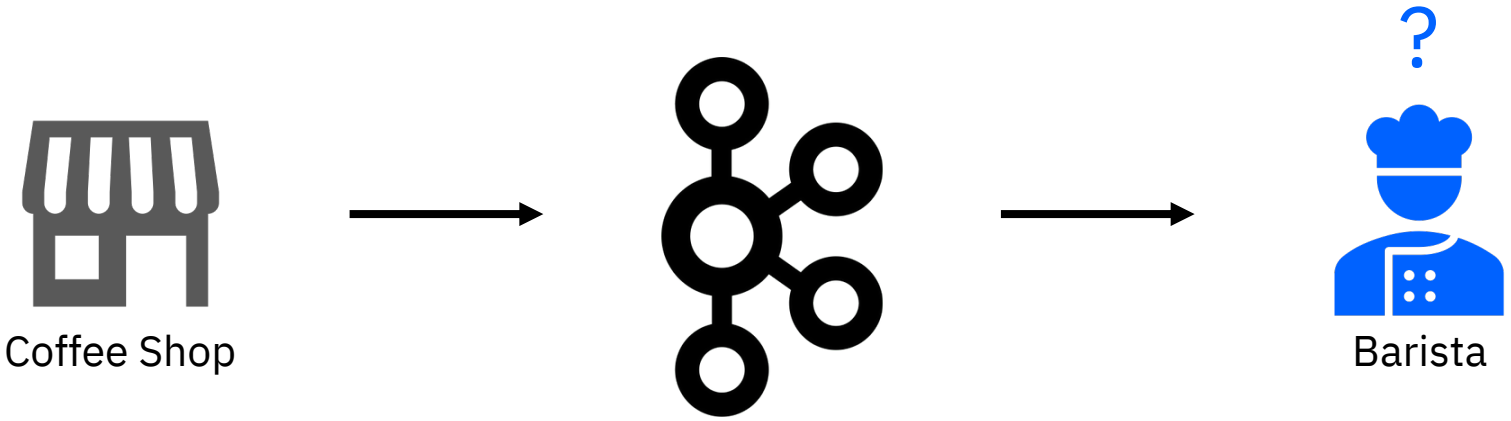
“1234,hot chocolate,true,Kate,small”

StringSerializer



“1234,hot chocolate,true,Kate,small”

StringSerializer



Using JSON

JSON gives structure

```
{
    “orderId”: “1234”,
    “product”: “hot chocolate”,
    “customer”: ”Kate”,
    “size”: small
}
```

JSON gives structure

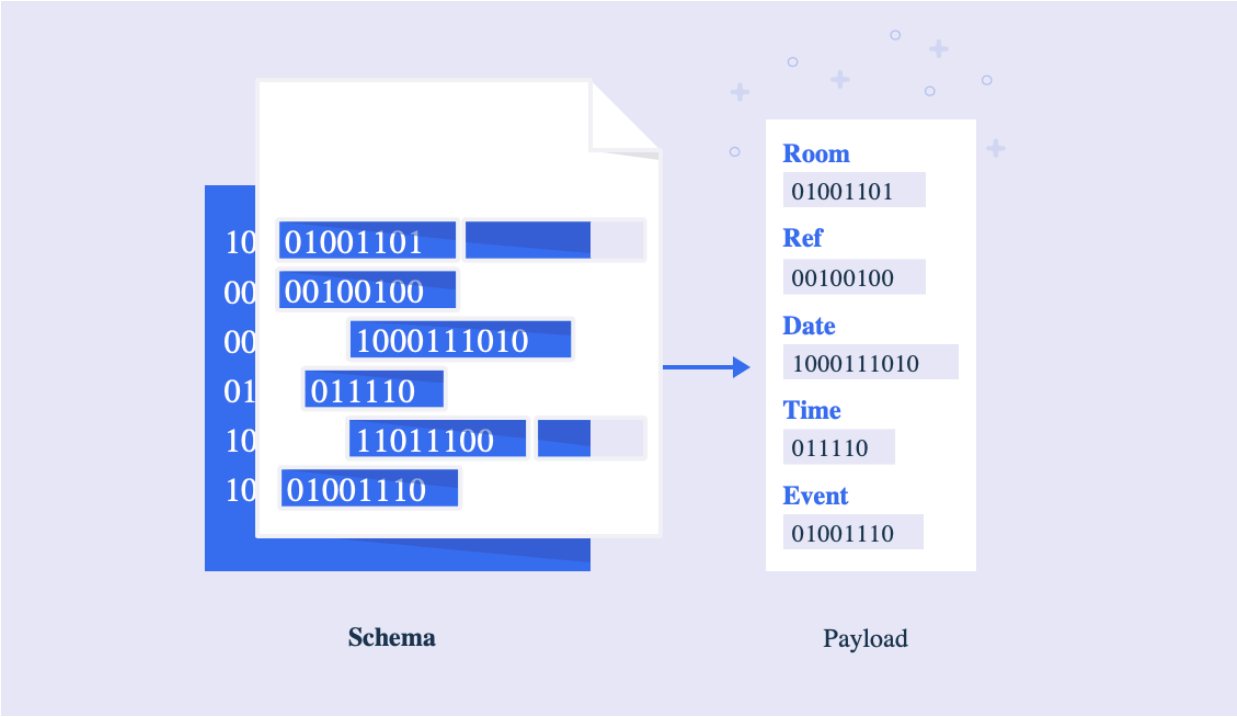
```
{
    “orderId”: “1234”,
    “product”: “hot chocolate”,
    “customer”: ”Kate”,
    “size”: small,
    ”marshmallows”: true
}
```

JSON lacks metadata

```
{
    “orderId”: “1234”,
    “product”: “hot chocolate”,
    “customer”: ”Kate”,
    “size”: small,
    ”marshmallows”: “yes”
}
```

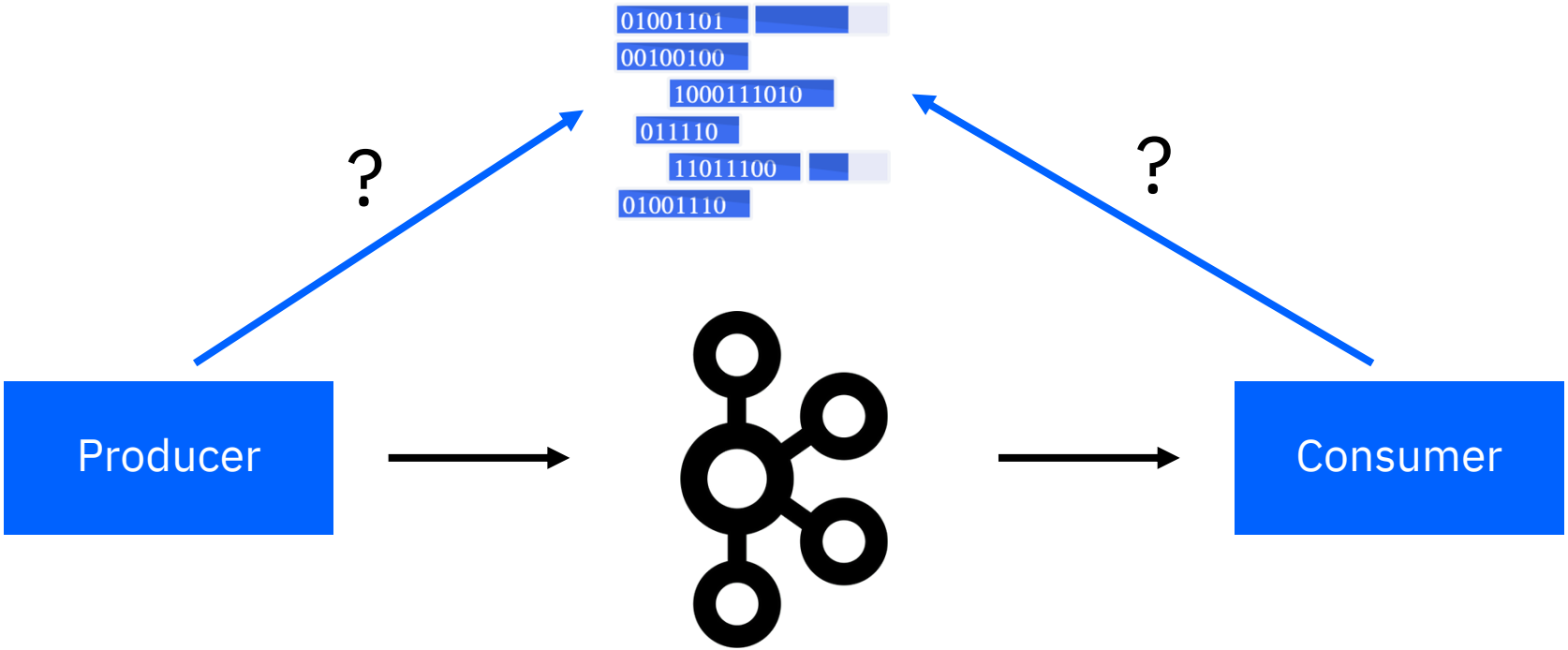
Adding structure with schemas

Adding structure with Schemas

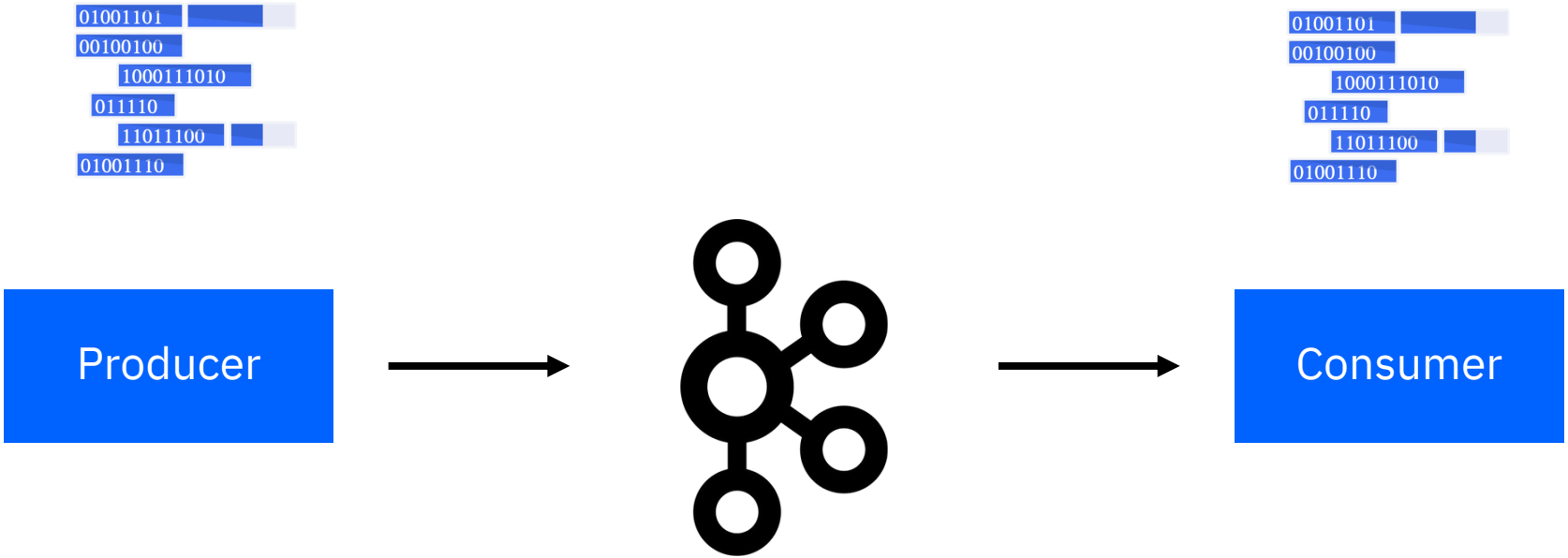


```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://c
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testin

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

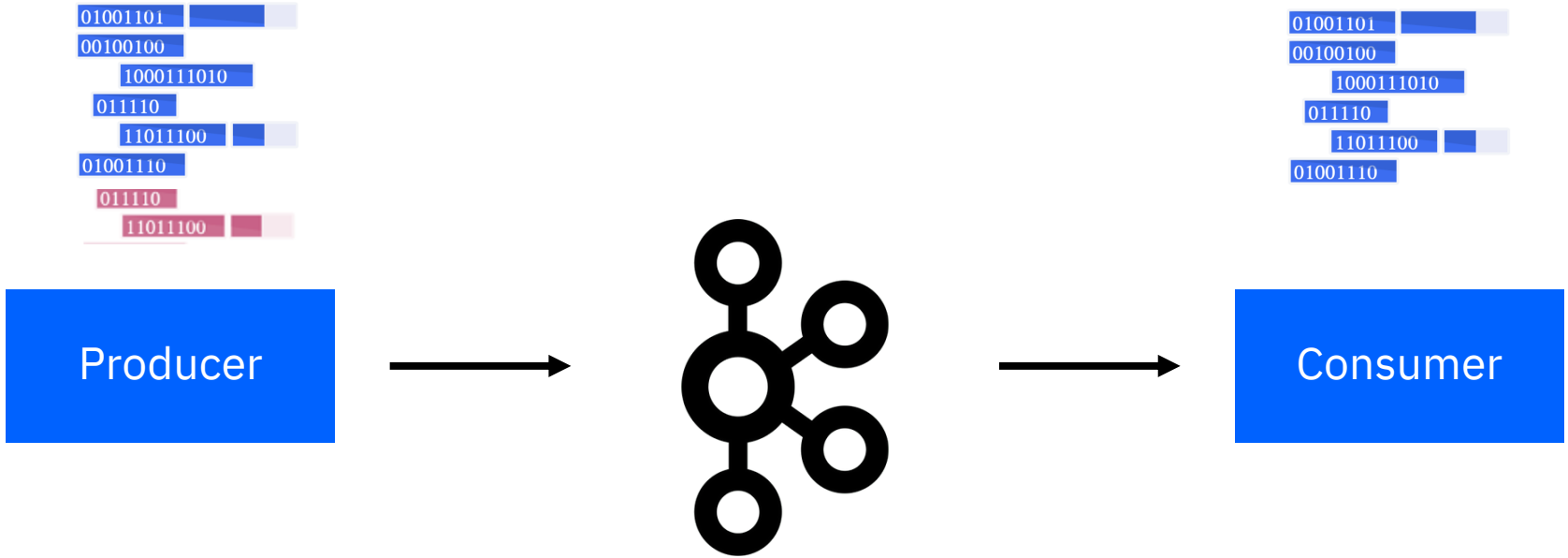
Where to store the schema



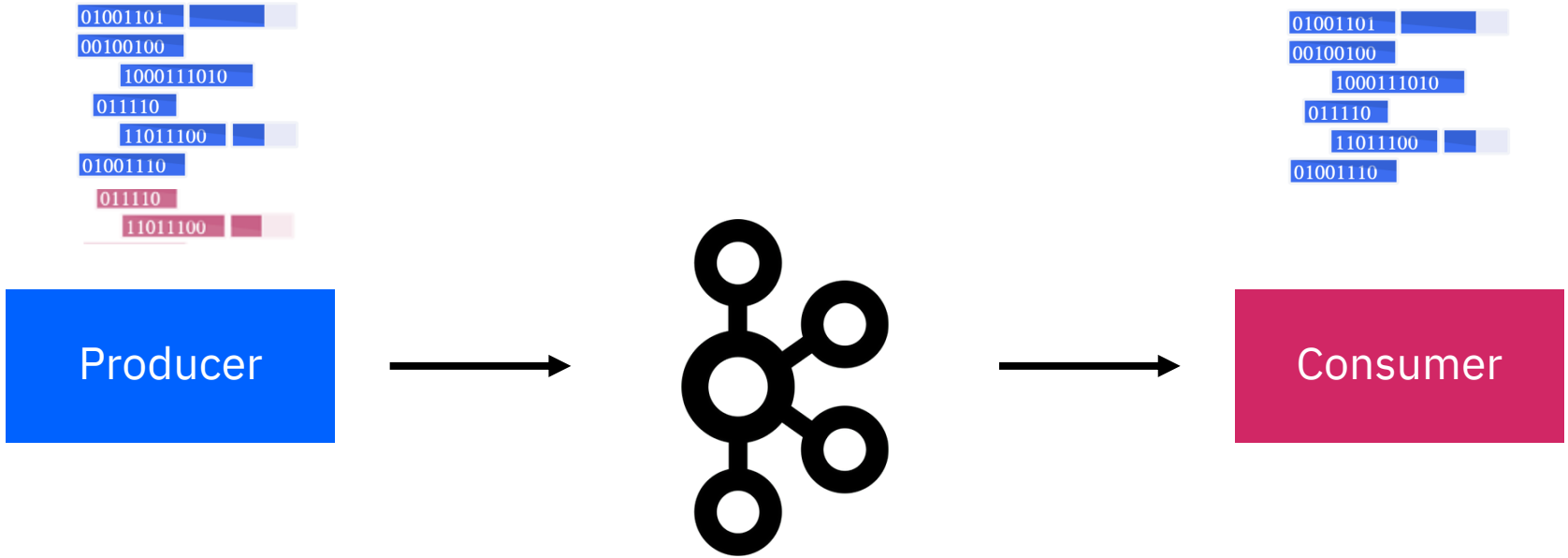
Where to store the schema



Where to store the schema

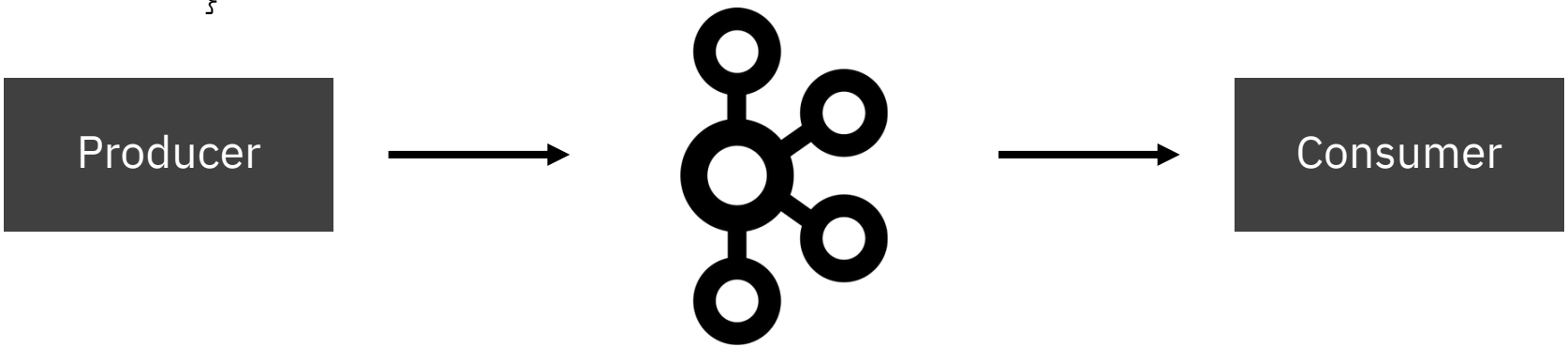


Where to store the schema



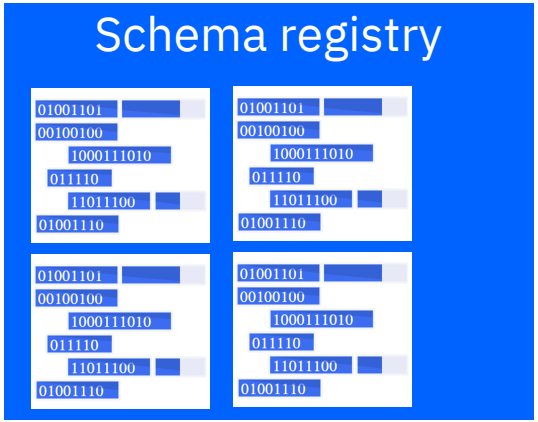
Schema in the message

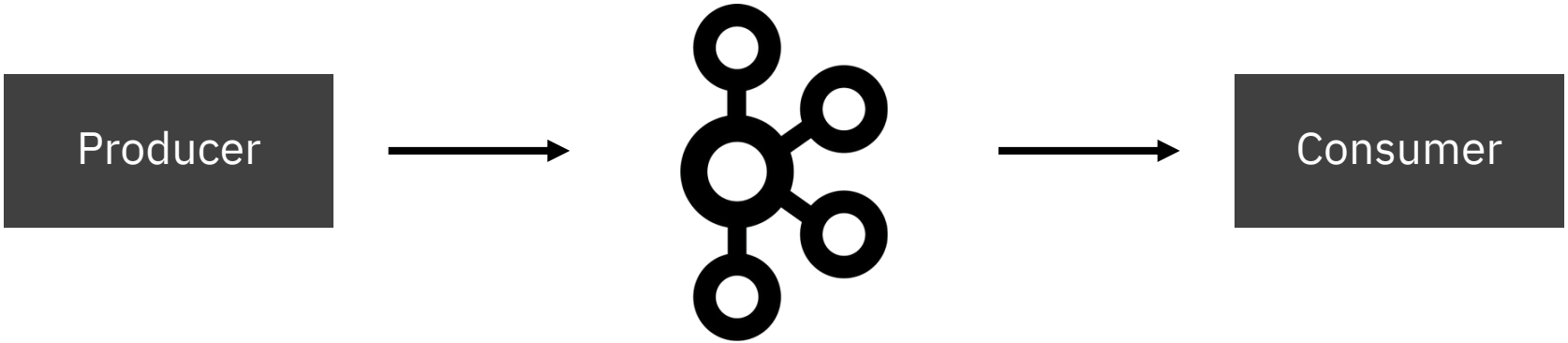
```
{
    “schema”: {
        “properties”: {
            “orderId”: {“type”: “string”},
            “size”: {“type”: ”bool”},
            …
    },
    “orderId”: “1234”,
    “product”: “hot chocolate”,
    “customer”: ”Kate”,
    “size”: small
}
```



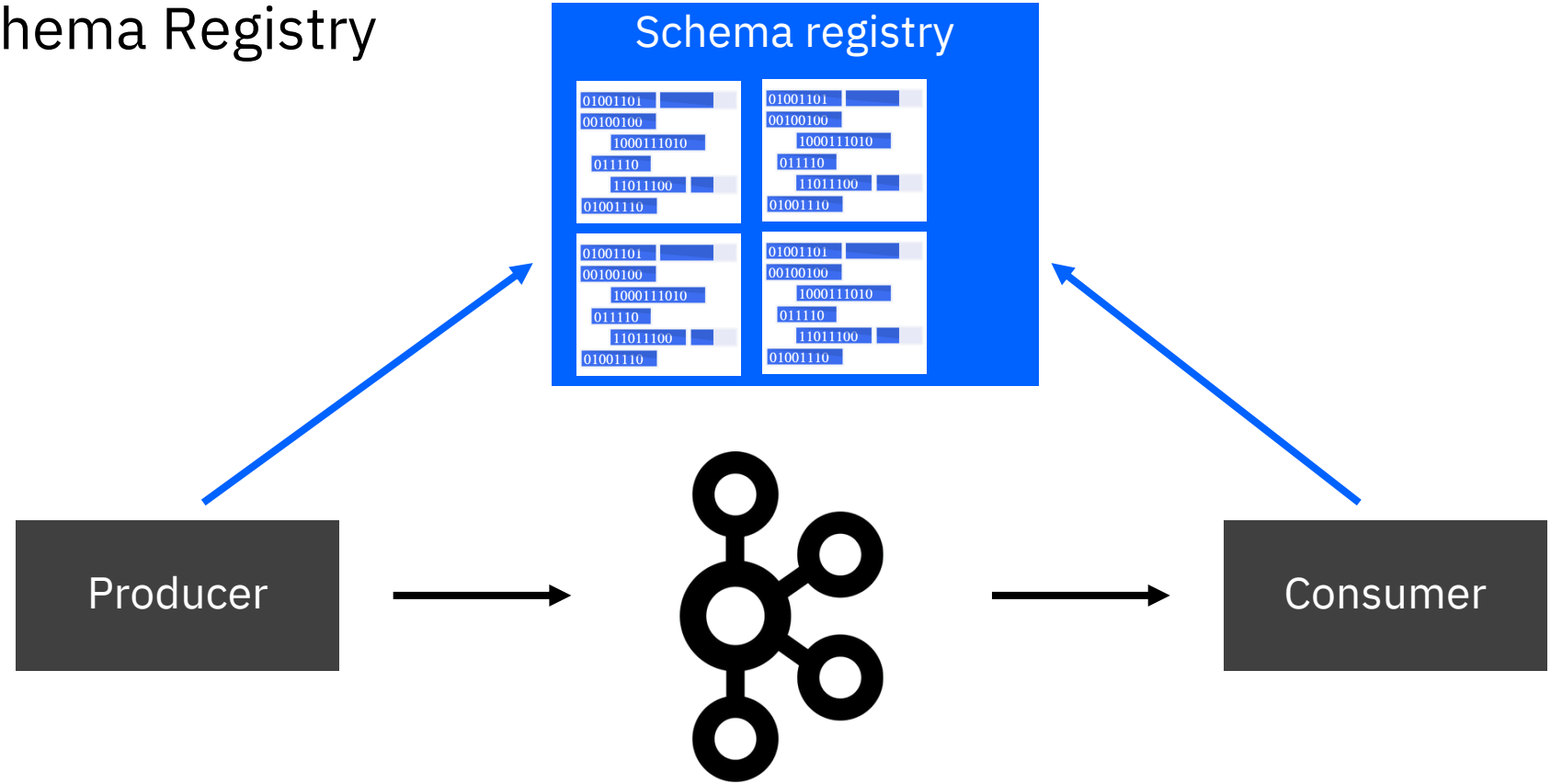
Schema Registry

Schema Registry





Schema Registry



What format to use?

Which schema to choose?

Schema formats

Schema Formats

JSON Schema

Google Protocol Buffer

Apache Avro

JSON Schema

A vocabulary that allows you
to annotate and validate JSON documents

- Describes your existing data format(s)
- Provides clear human- and machine- readable documentation

https://json-schema.org

```
{
    "type": "object",
    "properties": {
        "product": {"type": "string"},
        "customer": {"type": "string"},
        "size": {
            "type": "string",
            "enum": ["small", "medium"]
        }
    },
    "required": ["product", "size"]
}
```

Protobuf

Mechanism for serializing structured data

- Created by Google
- Language neutral
- Supports generated code in Java, Python, Objective-C and C++

https://developers.google.com/protocol-buffers/

.proto

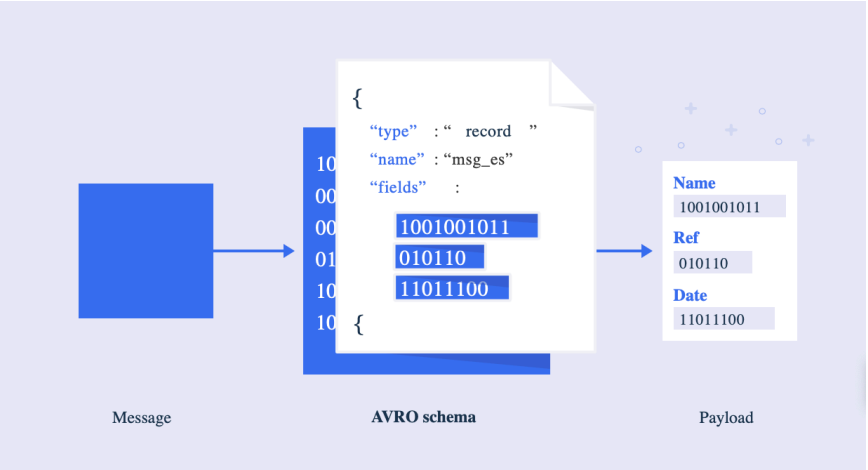
```
message Order {
    required int32 id = 1;
    required string product = 2;
    optional bool marshmallows = 3;
}
```

```
Order hotChocolate = Order.newBuilder()
    .setOrderId(1234)
    .setProduct("hot chocolate")
    .setMarshmallows(true)
    .build();
```

Apache Avro

A data serialization system that provides:

- Rich data structures
- A compact, fast, binary data format
- Simple integration with dynamic languages



Apache Avro

.avsc

```
{
    "type": "record",
    "name": "Order",
    "fields": [
        {"name": "product", "type": "string"},
        {"name": "customer",  "type": "string"},
        {"name": "orderId", "type": ”string", "logicalType": "uuid"},
        {"name": "marshmallows", "type": "boolean"},
        {"name": "size", "type": "enum", "symbols" : ["small", "medium"]}
    ]
}
```

Apache Avro

.avsc

```
{
    "type": "record",
    "name": "Order",
    "fields": [
        {"name": "product", "type": "string"},
        {"name": "customer",  "type": "string"},
        {"name": "orderId", "type": "string", "logicalType": "uuid"},
        {"name": "marshmallows", "type": "boolean"},
        {"name": "size", "type": "enum", "symbols" : ["small", "medium"]}
    ]
}
```

Apache Avro

.avsc

```
{
    "type": "record",
    "name": "Order",
    "fields": [
        {"name": "product", "type": "string"},
        {"name": "customer",  "type": "string"},
        {"name": "orderId", "type": "string", "logicalType": "uuid"},
        {"name": "marshmallows", "type": "boolean"},
        {"name": "size", "type": "enum", "symbols" : ["small", "medium"]}
    ]
}
```

Apache Avro

.avsc

```
{
    "type": "record",
    "name": "Order",
    "fields": [
        {"name": "product", "type": "string"},
        {"name": "customer",  "type": "string"},
        {"name": "orderId", "type": "string", "logicalType": "uuid"},
        {"name": "marshmallows", "type": "boolean"},
        {"name": "size", "type": "enum", "symbols" : ["small", "medium"]}
    ]
}
```

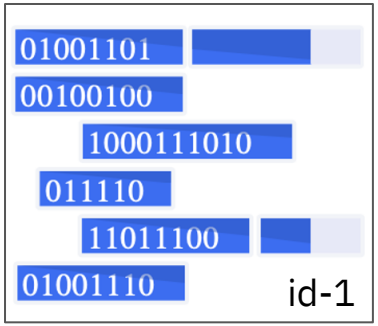
Apache Avro

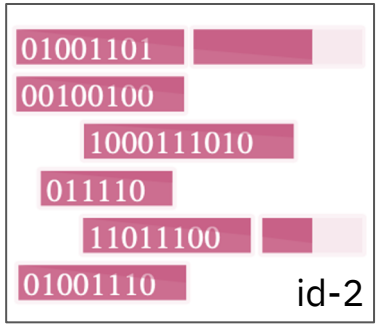
```
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
```

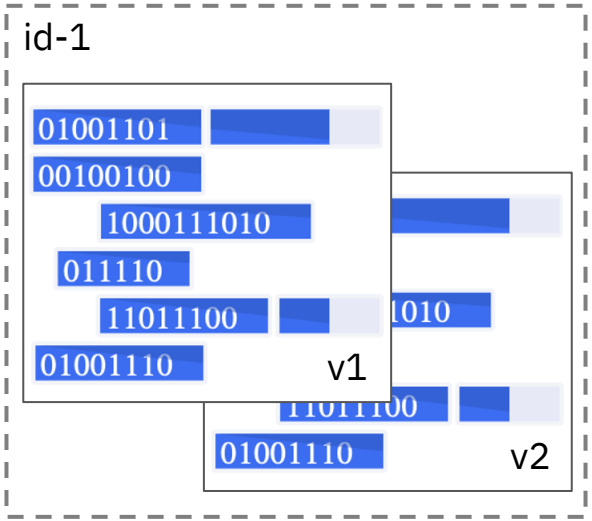
```
Schema schema = new Schema.Parser().parse(ORDER_SCHEMA);

GenericData.Record record = new GenericData.Record(schema);
record.put("product", "hot chocolate");
record.put("customer", "Kate");
record.put("marmshallows", true);
```

Which schema to choose

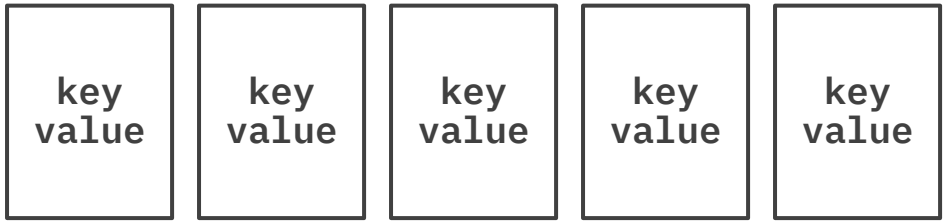
Identifying schemas







Kafka Records

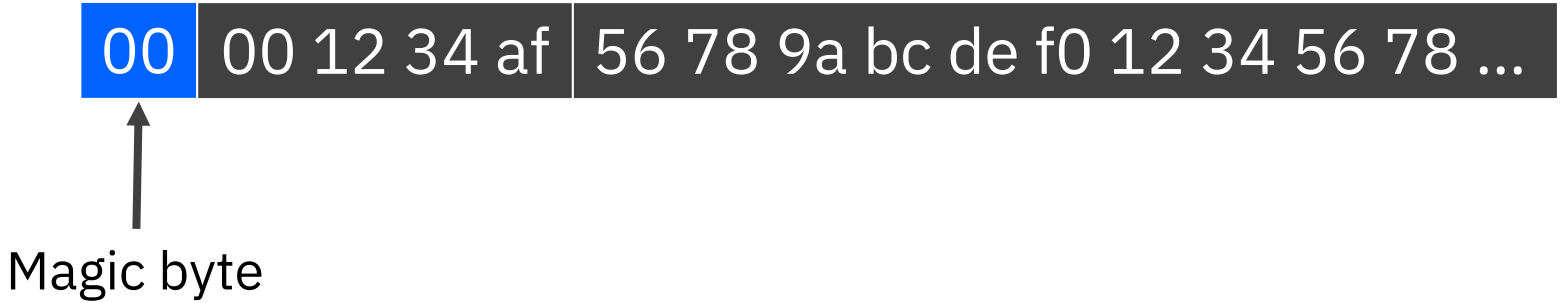


Magic bytes

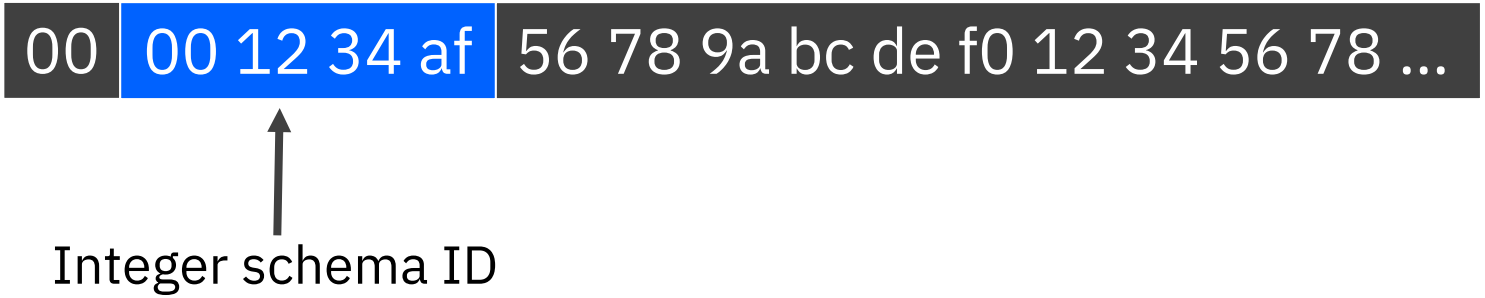
```
00   00 12 34 af   56 78 9a bc de f0 12 34 56 78 ...
```

Magic bytes

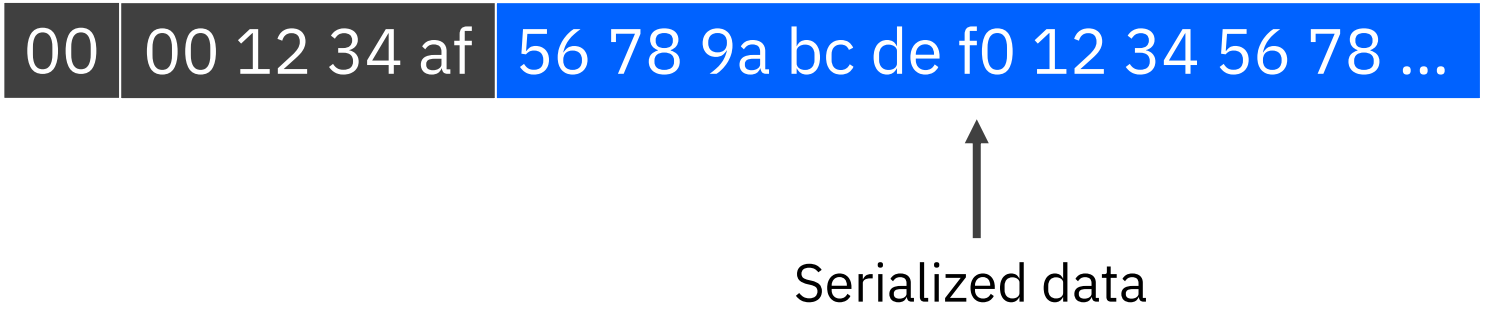
Magic bytes



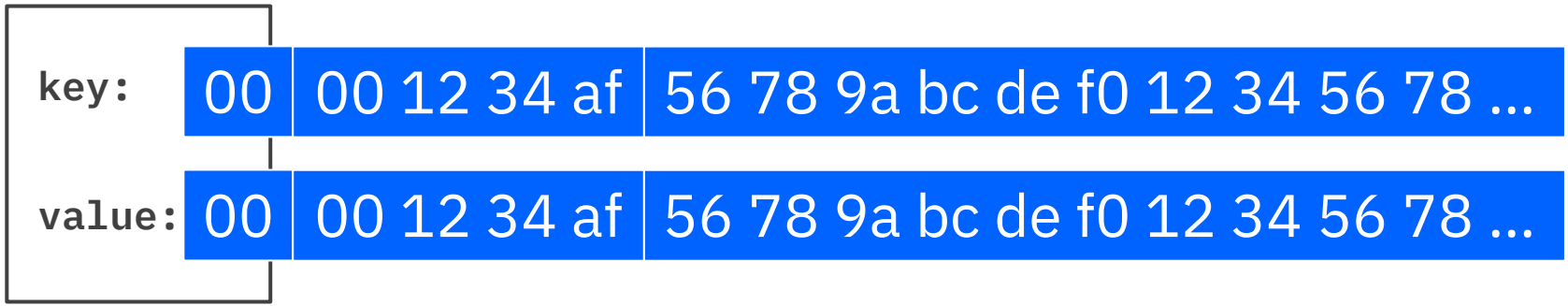
Magic bytes



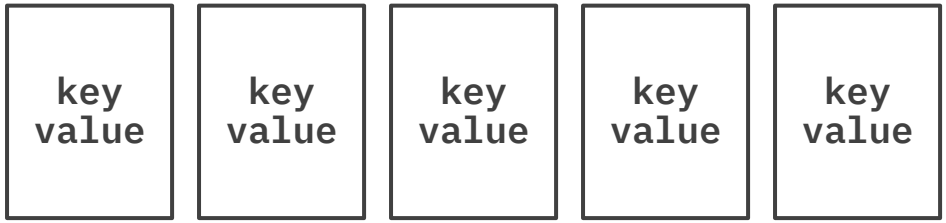
Magic bytes



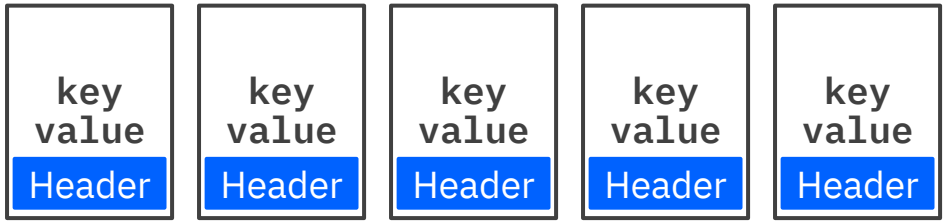
Magic bytes



Record Headers



Record Headers



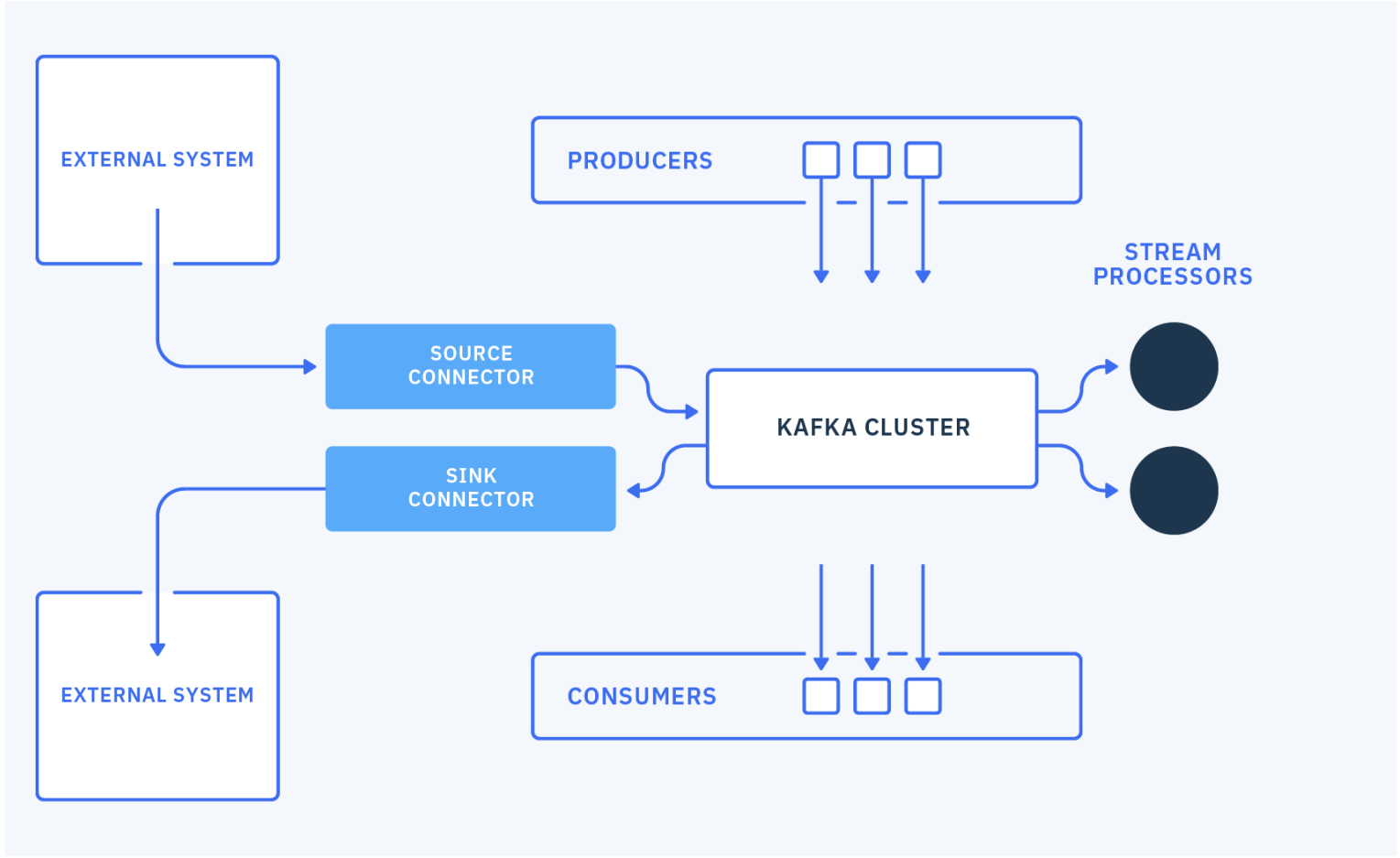
Headers

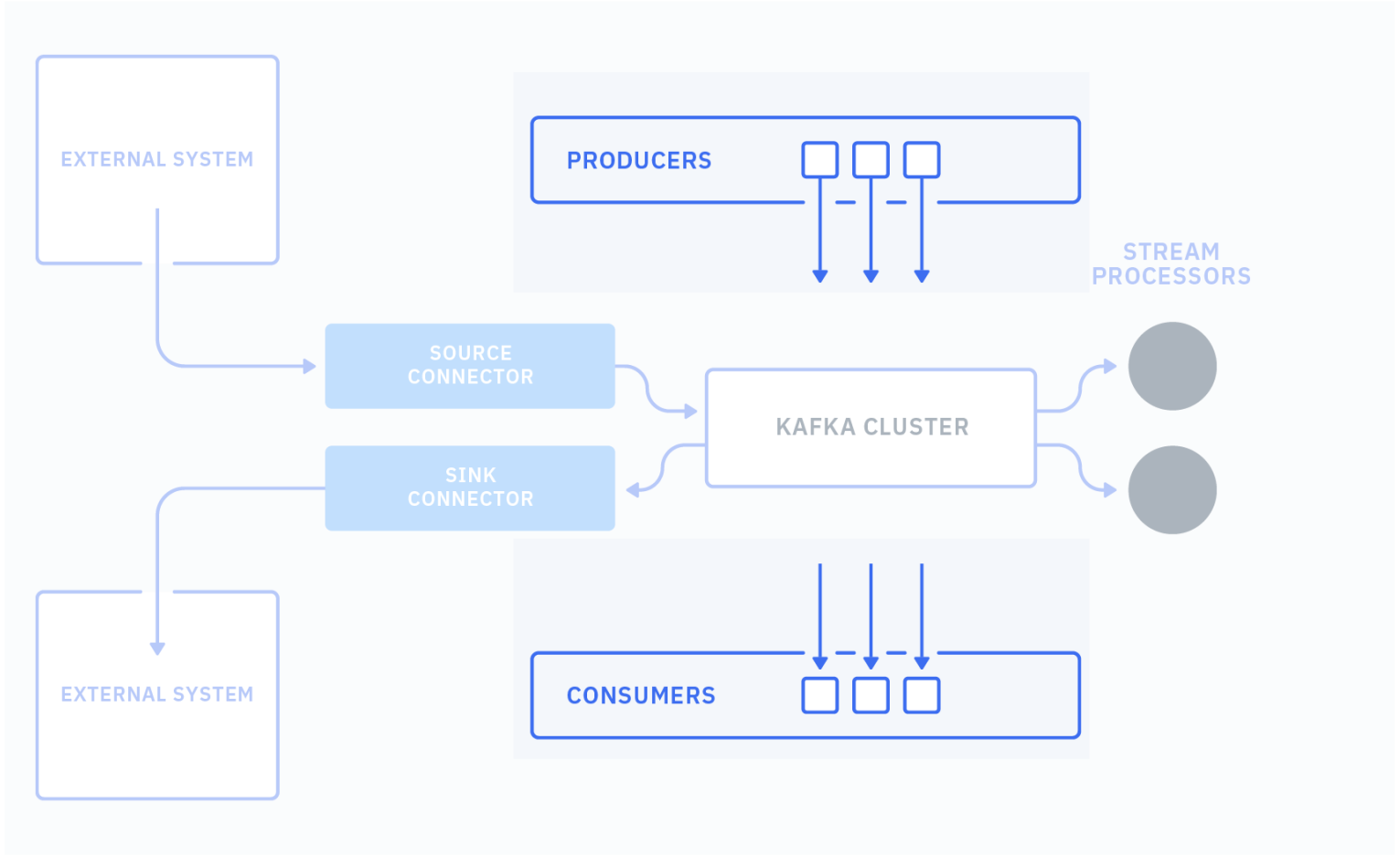
```
ProducerRecord<String, MySchema> producerRecord =
    new ProducerRecord<String, MySchema>(“key”, specificRecord);


producerRecord.headers().add(“value-schema-id”, schema.getIdAsBytes());
producerRecord.headers().add(“value-schema-version”, schema.getVersionAsBytes());


producer.send(producerRecord);
```

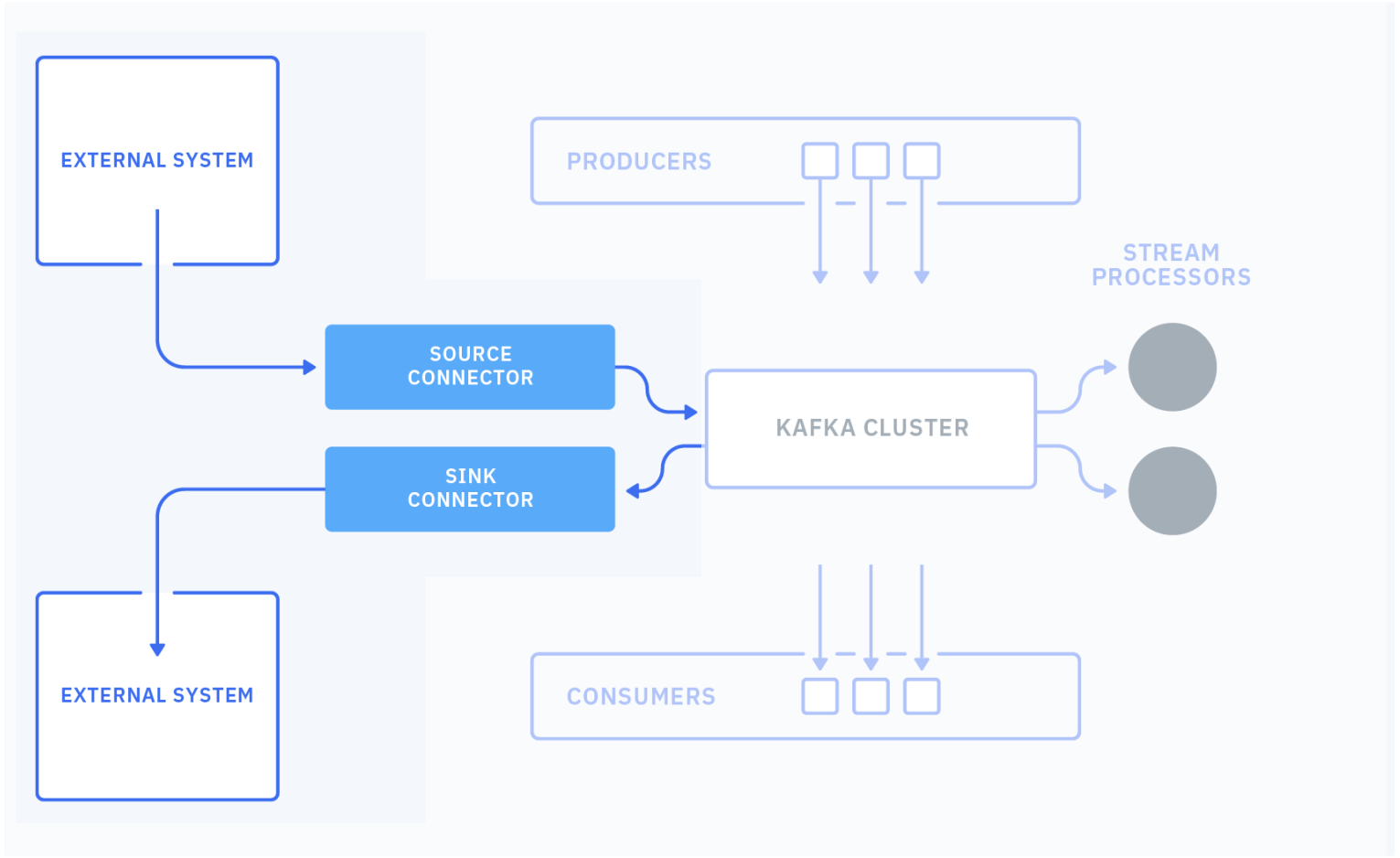
Magic Byte

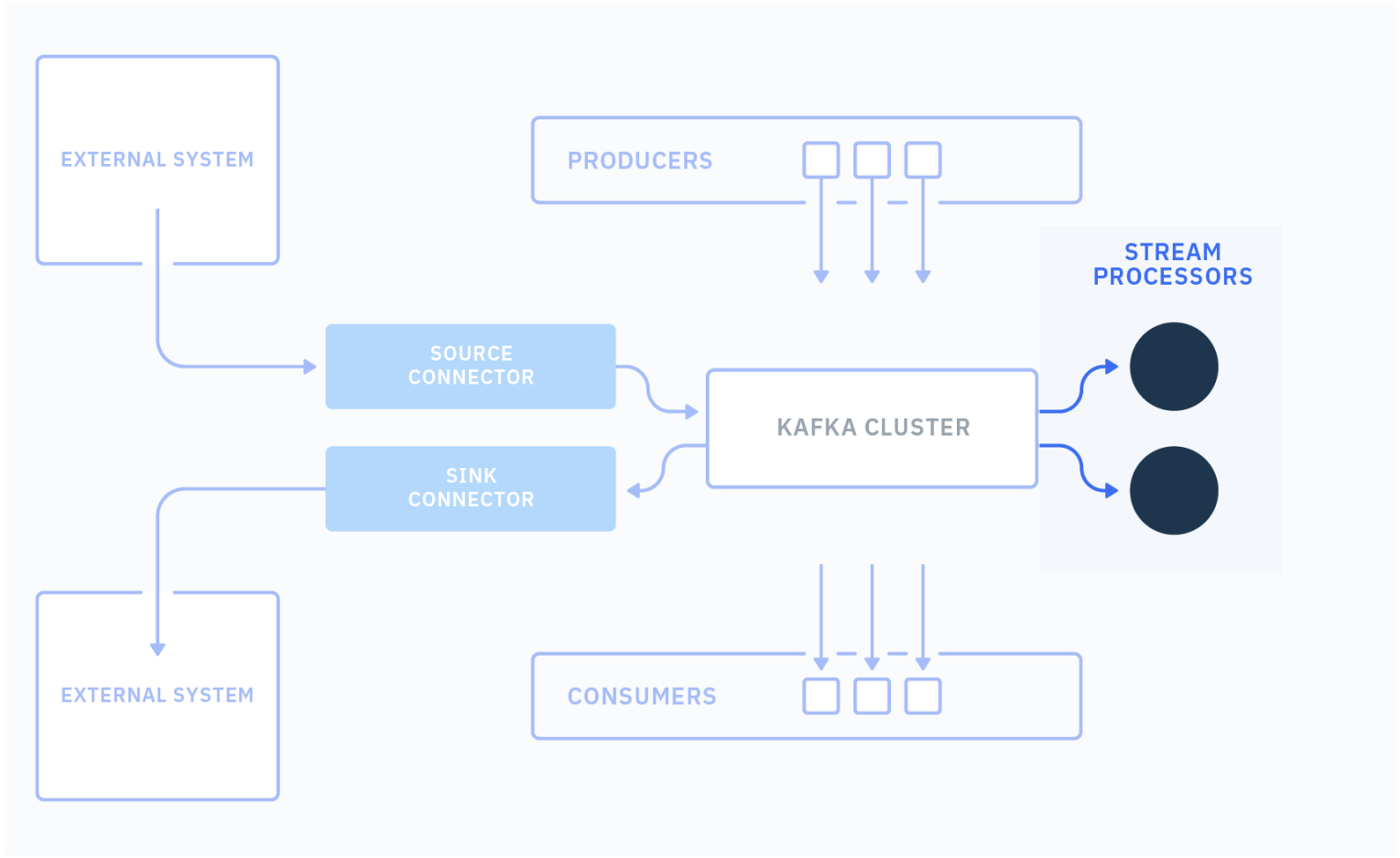
Record Header

Serdes and Converters









Producer/Consumer

Producer

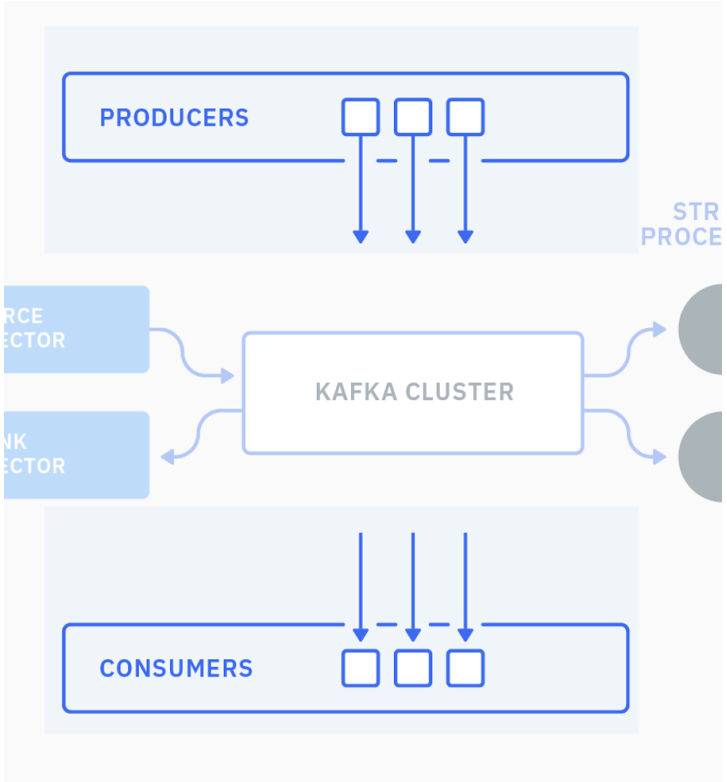
key.serializer

value.serializer

Consumer

key.deserializer

value.deserializer



Kafka Connect

Converter

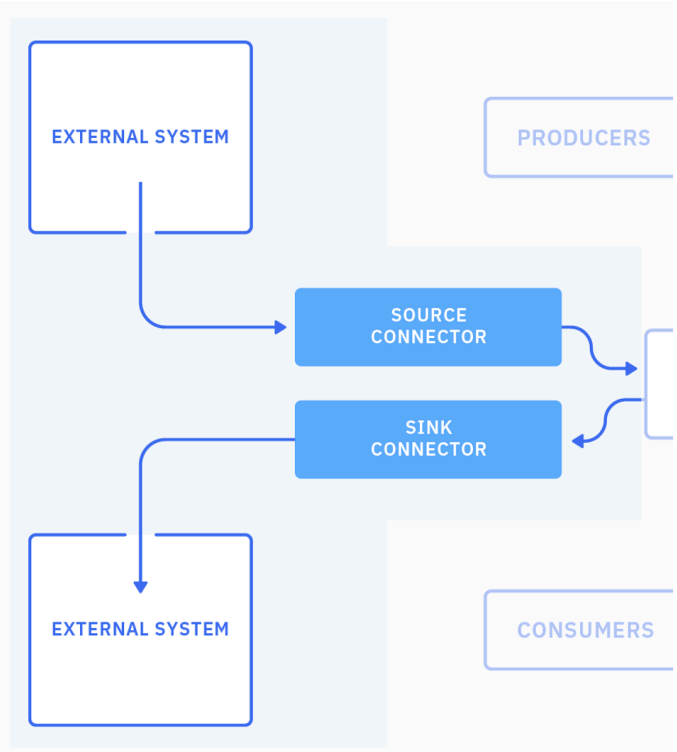
key.converter

value.converter

org.apache.kafka.connect.json.JsonConverter

org.apache.kafka.connect.storage.StringConverter

org.apache.kafka.connect.converters.ByteArrayConverter

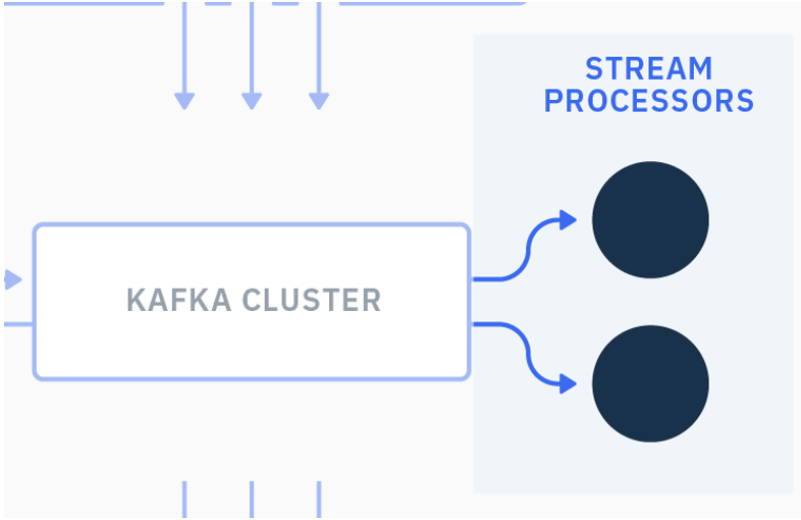


Kafka Streams

SerDes

key.serde

value.serde

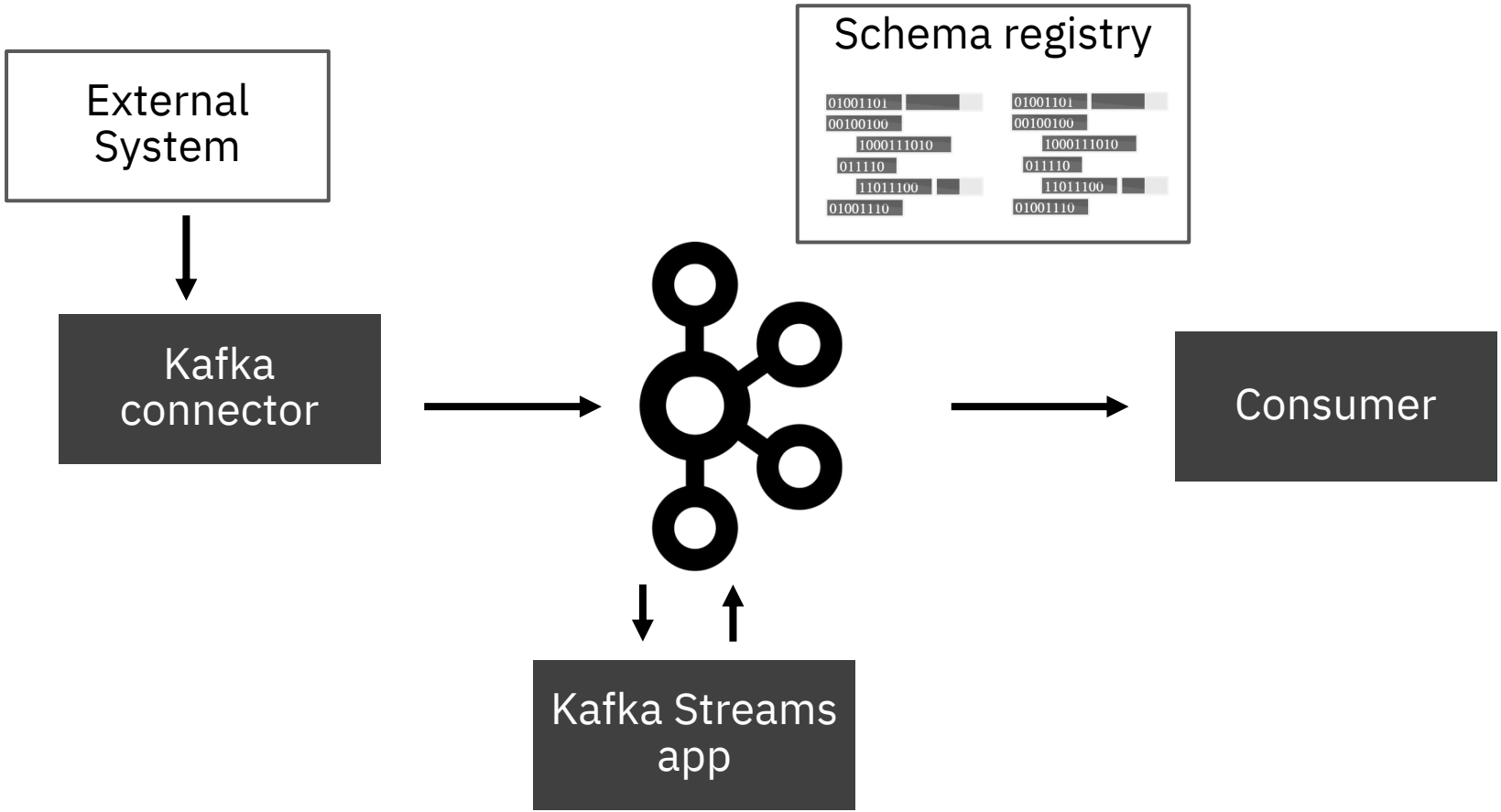


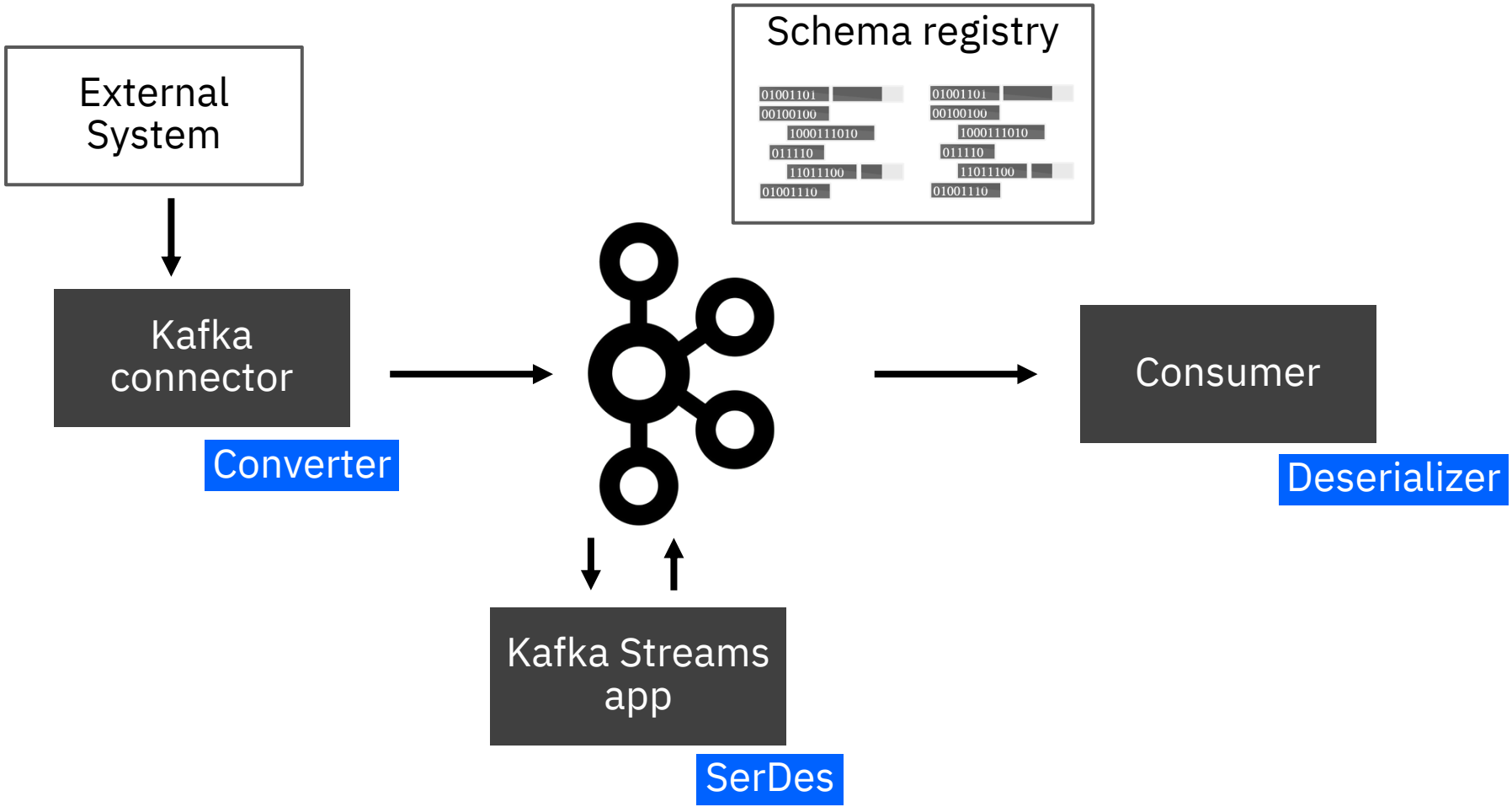
org.apache.kafka.common.serialization.Serdes

Serdes.ByteArray()

Serdes.String()

Serdes.Integer()





Apicurio Registry

Apicurio Registry

Open Source schema registry

Format options:

- Apache Avro, Google protocol buffers, JSON Schema

Storage options:

- Kafka, PostgreSQL

IBM and Confluent compatibility APIs

https://github.com/Apicurio/apicurio-registry

Apicurio Registry

Producer: io.apicurio.registry.utils.serde.AvroKafkaSerializer

Consumer: io.apicurio.registry.utils.serde.AvroKafkaDeserializer

Kafka Connect: io.apicurio.registry.utils.converter.AvroConverter

Kafka Streams: io.apicurio.registry.utils.serde.AvroSerde

Confluent Schema Registry

Confluent Schema Registry

Confluent community license

Format options:

- Apache Avro, Google protocol buffers, JSON Schema

Storage options:

- Kafka

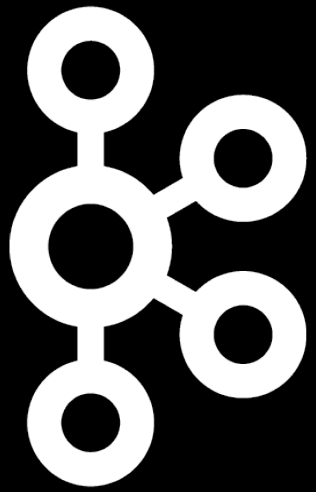
https://docs.confluent.io/platform/current/schema-registry/index.html#

Summary

Use schemas to get structure

Schema registry provides a centralized store

Choose your favourite format and identification mechanism



Thank you

Kate Stanley | @katestanley91

https://ibm.github.io/event-streams/schemas/overview/

https://www.apicur.io/registry/docs/apicurio-registry/index.html
https://github.com/Apicurio/apicurio-registry

https://docs.confluent.io/platform/current/schema-registry/index.html#

Getting started with Kafka:
https://kafka.apache.org/quickstart
https://strimzi.io
https://github.com/ibm-messaging/kafka-java-vertx-starter