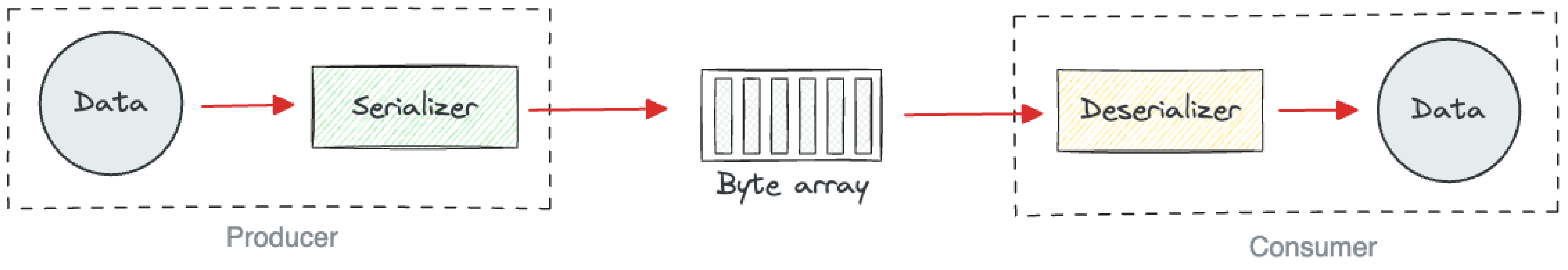
# Kafka Schema Registry - An Overview

Data serialization is the process of converting complex data structures, such as objects or data arrays, into a format that can be easily stored, transmitted, or reconstructed later. Typically, this format is a stream of bytes that can be written to a file or transmitted over a network.

The serialized data needs to be deserialized, or reconstructed back into its original form before it can be used again. Deserialization involves the reconstruction of the original data structure from the byte stream.

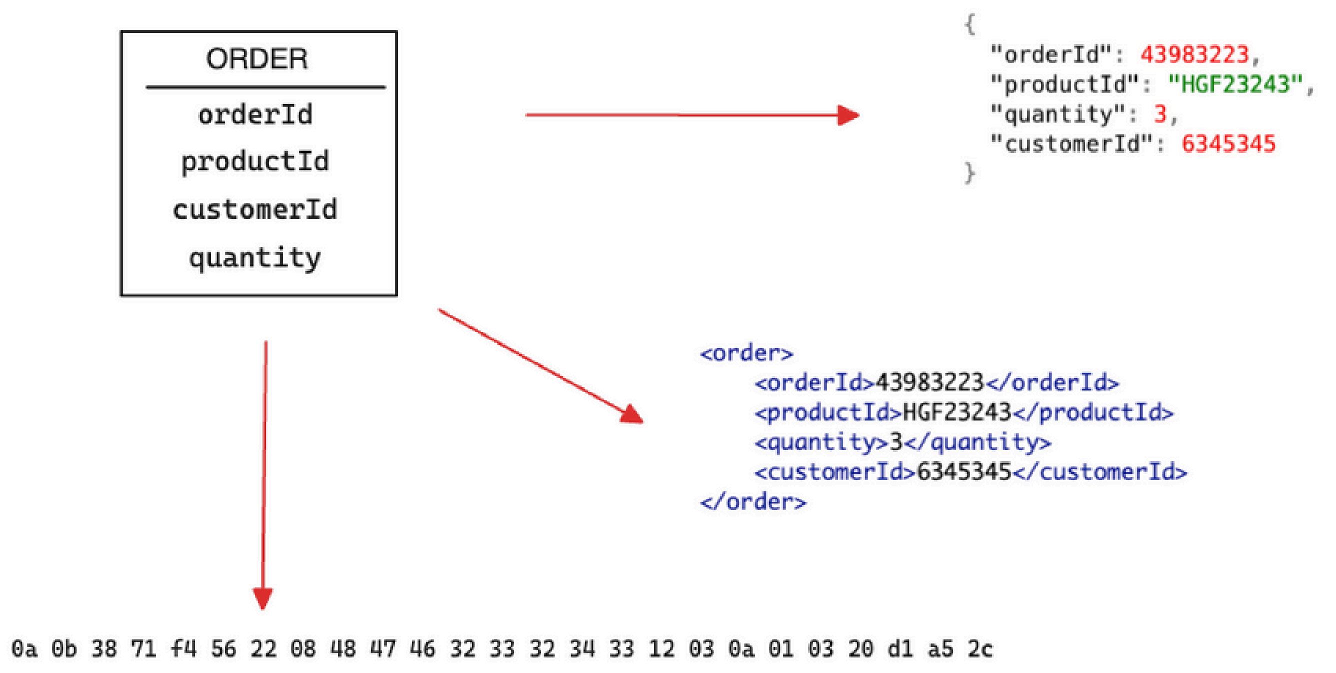
A **serializer** is a software component responsible for serialization, while a **deserializer** handles deserialization.



## Data formats

We utilize various **data formats** to represent data, such as JSON, XML, Avro, Google Protocol Buffers (ProtoBuff), and YAML.

Having a well-defined format is like having a common language that all developers can understand and use, facilitating collaboration and communication.



```
{
  "orderId": 43983223,
  "productId": "HGF23243",
  "quantity": 3,
  "customerId": 6345345
}
```

```
<order>
    <orderId>43983223</orderId>
    <productId>HGF23243</productId>
    <quantity>3</quantity>
    <customerId>6345345</customerId>
</order>
```

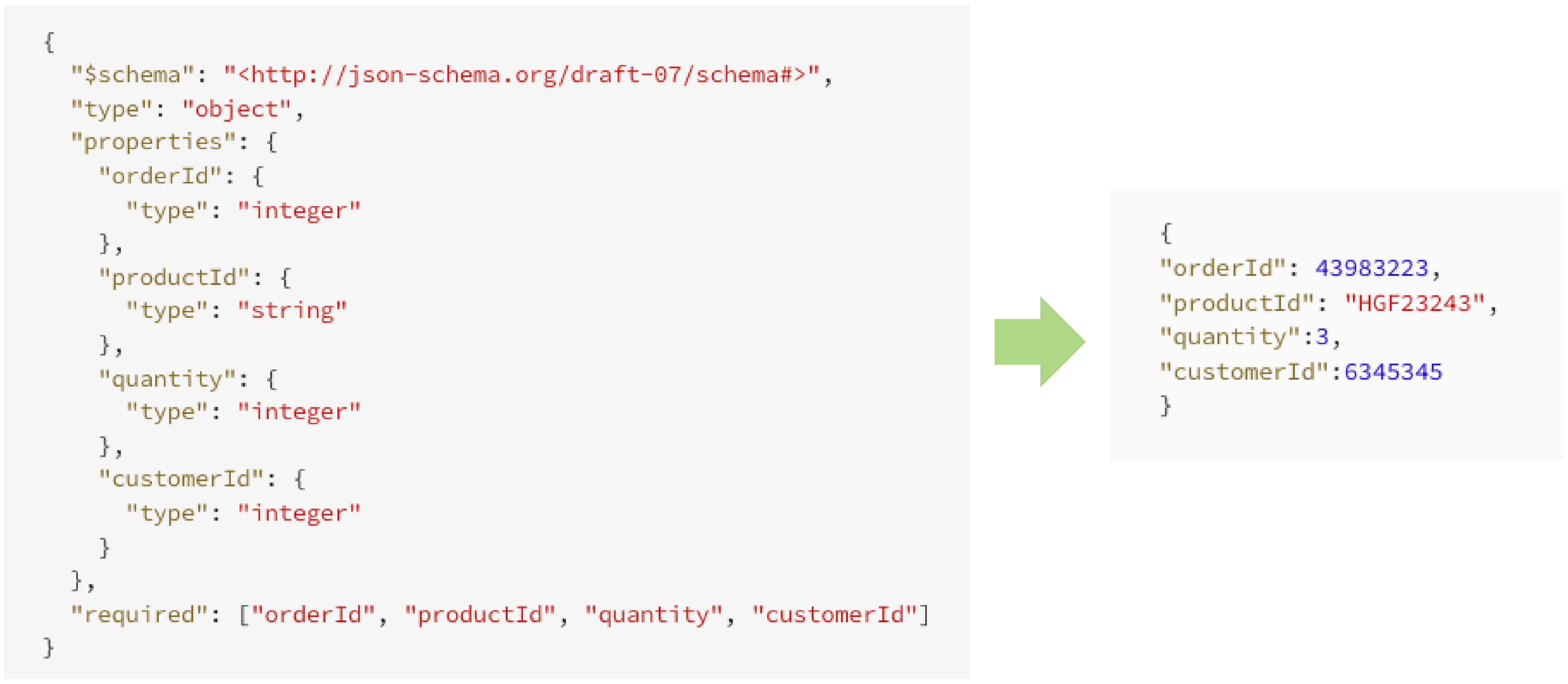
```
0a 0b 38 71 f4 56 22 08 48 47 46 32 33 32 34 33 12 03 0a 01 03 20 d1 a5 2c
```

## Schemas

Formats only define how data should be structured, but not what the data should be. This is where schemas come into play.

A schema is like a blueprint of how the data should be constructed. It defines the type of data (e.g., integer, string, date), the order of data, and whether the data is mandatory or optional. By defining these aspects, schemas provide a much more robust and detailed definition of the data than formats alone.

For example, the following is the JSON schema for a JSON object representing an order.



```
{
  "$schema": "<http://json-schema.org/draft-07/schema#>",
  "type": "object",
  "properties": {
    "orderId": {
      "type": "integer"
    },
    "productId": {
      "type": "string"
    },
    "quantity": {
      "type": "integer"
    },
    "customerId": {
      "type": "integer"
    }
  },
  "required": ["orderId", "productId", "quantity", "customerId"]
}
```

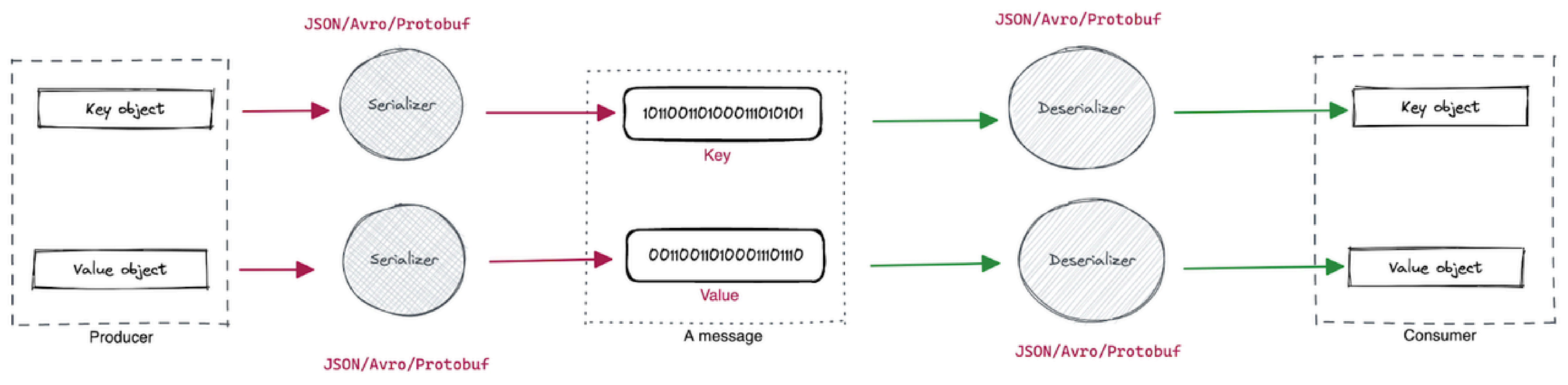
```
{
"orderId": 43983223,
"productId": "HGF23243",
"quantity":3,
"customerId":6345345
}
```

In Kafka, a message is composed of a key and a value. Different serializers and deserializers (SerDe) can be specified for these keys and values. These Serdes are a part of the language-specific SDK, and support data formats including Apache Avro™, JSON Schema, and Google's Protobuf.

For Kafka, Avro is by far the most popular choice for data serialization.

## Kafka Schema Registry - An Overview

This figure illustrates how serialization and deserialization works for keys and values.



### Writer and reader schemas

When there's a message, the serializer does its job based on the schema specified. This schema is often called the **writer's schema**. When the message gets deserialized, the deserializer always needs a schema to continue, and it’s called the **reader's schema**.

Now, the question is how the serializer shares the writer's schema with the deserializer.

There are two options.

- **Embedding the schema into each message** - Formats like Avro and Protobuf simply package schema information along with data in each message, allowing the deserializer to read the schema from the message itself. However, this would consume a lot of network bandwidth.

- **Sharing the schema manually** - Another less popular option would be sharing the writer's schema with the deserializer offline or embedding it in the producer's source code. Although this might work, it’ll soon become chaotic as the number of consumers increases and the schema evolves.

But what if we keep the schemas in a central location, independent of producers and consumers, allowing them to register, discover and retrieve schemas during the serialization and deserialization?

This is why the **schema registry** exists.

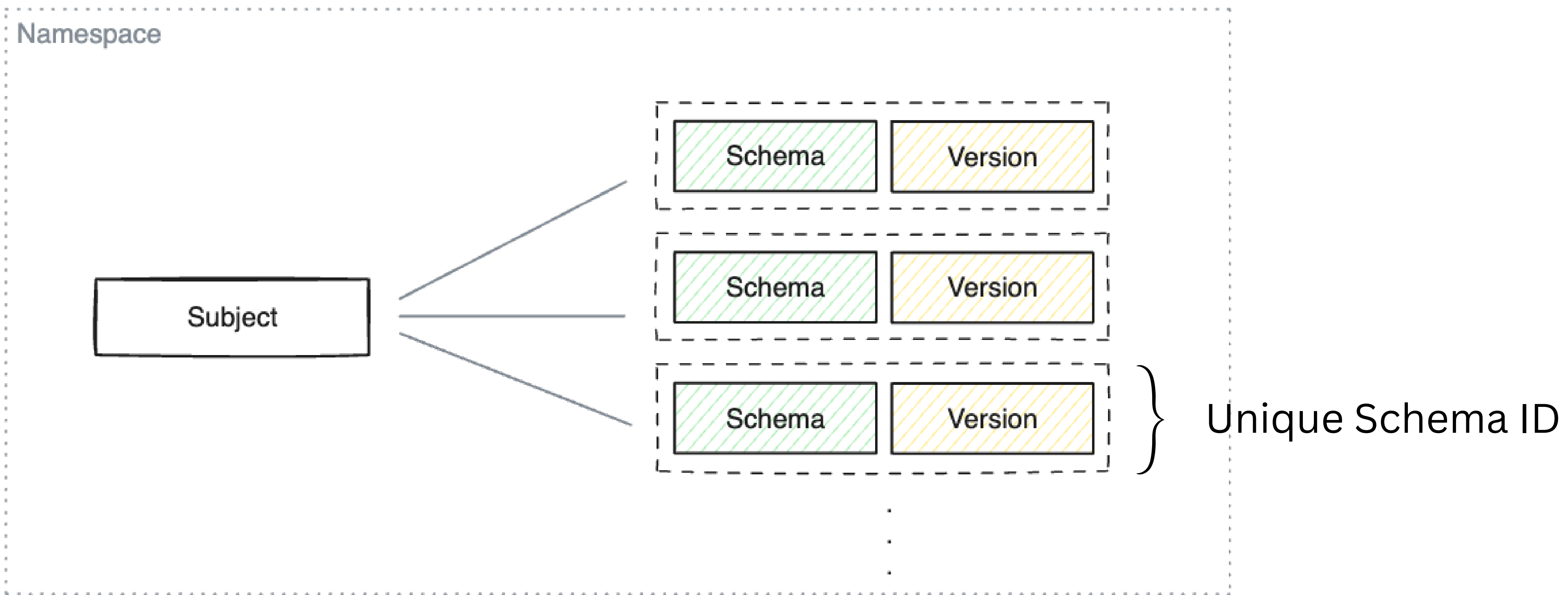
# Kafka Schema Registry - An Overview

## Schema registry

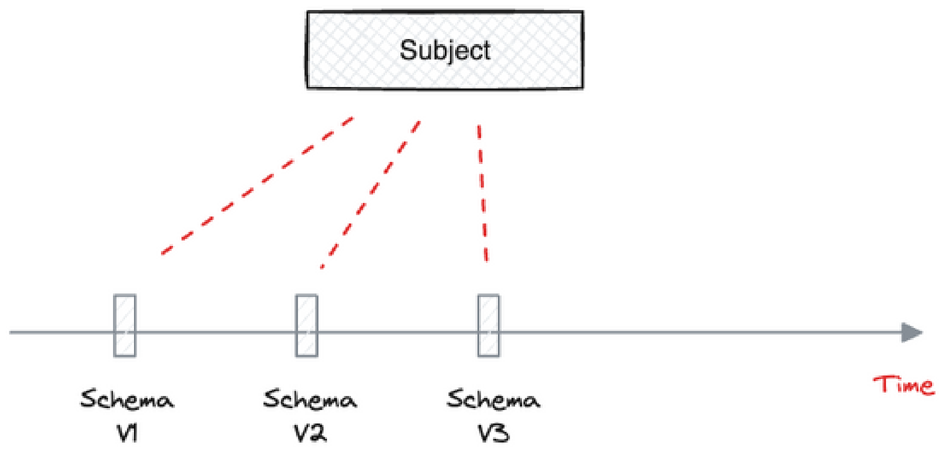
A schema registry is a central repository where schemas are stored. It provides Kafka producers and consumers with APIs to register, discover, and retrieve schemas during data serialization and deserialization. Having a central repository for schemas eliminates the need to embed schemas in each message or share schemas manually, both of which can be inefficient or chaotic.

A schema registry maintains a hierarchy of information for keeping track of **subjects**, **schemas**, and their **versions**.

When a new schema is registered with the schema registry, it's always associated with a subject, representing a unique namespace within the registry. Multiple versions of the same schema can be registered under the same subject, and a unique schema ID identifies each version. The subject name is used to organize schemas and ensure a unique identifier for each schema within the registry.



A Subject can have many schemas with different versions



## Message serialization with schema registry

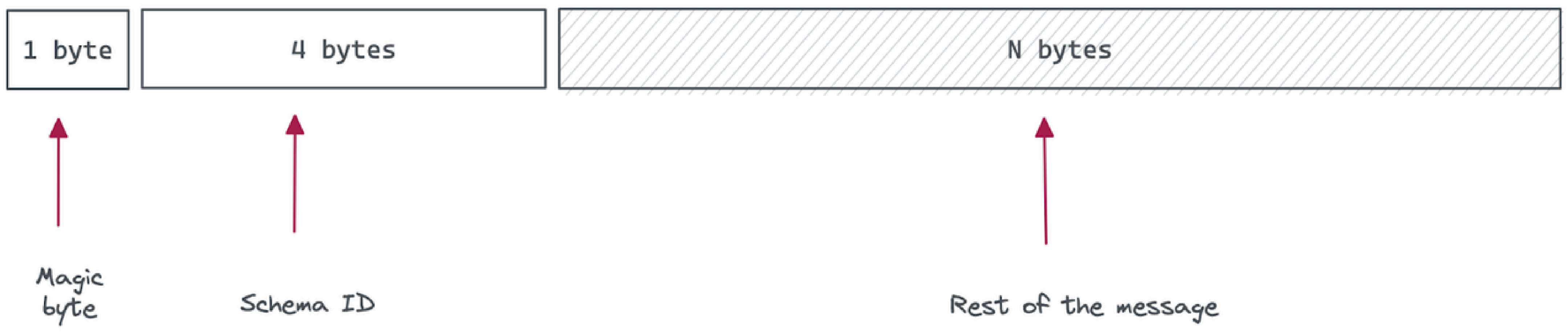
Initially, the producer passes the message to the appropriate key/value serializer. The serializer then determines which schema version to use for serialization.

To do this, the serializer first verifies if the schemaID for the given subject is present in the local schema cache. If the schemaID isn't in the cache, the serializer registers the schema in the schema registry and collects the resulting schemaID in the response. Typically, the serializer performs this task automatically.

In either case, the serializer should have the schemaID by now and proceeds with adding padding to the beginning of the message, containing:

- **The magic byte** - always contains the value of 0
- **SchemaID** - 4 bytes long integer containing the schemaID

This is illustrated as follows.



Finally, the serializer serializes the message and returns the byte sequence to the producer.

# Kafka Schema Registry - An Overview

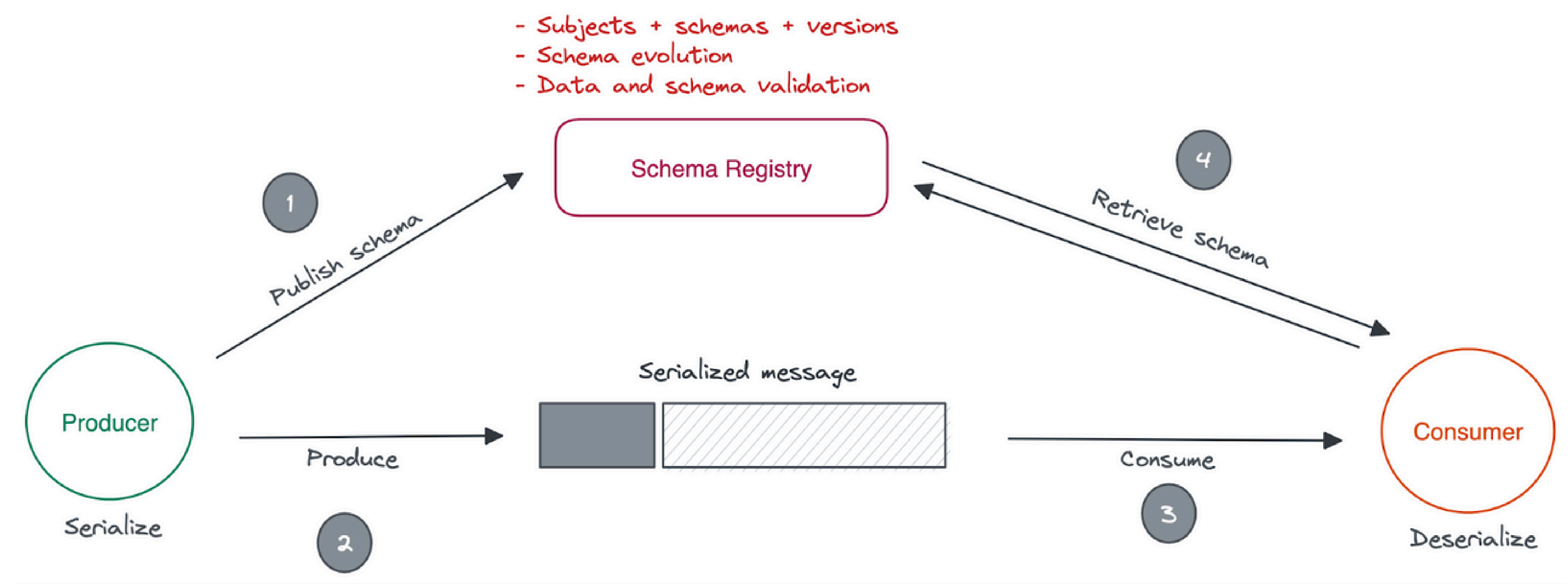
## Message deserialization with schema registry

The consumer fetches messages from Kafka and hands them over to the deserializer. The deserializer first checks the existence of the magic byte and rejects the message if it doesn't.

The deserializer then reads the schemaID and checks whether the related schema exists in its local cache. If that exists, deserialization happens with that schema. Otherwise, the deserializer retrieves the schema from the registry based on the schemaID.

Once the schema is in place, the deserializer proceeds with the deserialization.

The following figure illustrates the serialization and deserialization in one place.


- Subjects + schemas + versions
- Schema evolution
- Data and schema validation
Schema Registry
1
Publish schema
4
Retrieve schema
Producer
Serialize
Serialized message
Produce
2
Consume
3
Consumer
Deserialize