Kafka for Developers using Schema Registry

Dilip Sundarraj

About Me

- Dilip
- Building Software’s since 2008
- Teaching in **UDEMY** Since 2016

What’s Covered ?

- Fundamentals of Data Serialization and different Serialization formats available
- Introduction to AVRO and its importance
- Build a real time Kafka Producer and Consumer application that exchanges data using AVRO Serialization format
- Introduction of Schema Registry and its vital role for the evolution of the data
- Integrate Schema Registry into the Producer and Consumer application
- Different Techniques to evolve the data using Schema Registry
- Build a Kafka Producer and Consumer Spring Boot app that integrates with Schema Registry
- Gradle and Maven

Targeted Audience

- Experience Java Developers
- Any developer who is interested in AVRO and its benefits
- Any developer who is Interested in using **AVRO** for sharing data using Kafka
- Any developer who is interested in learning about **Schema Registry** and how to build Kafka Producer and consumer applications that interacts with Schema Registry

Source Code

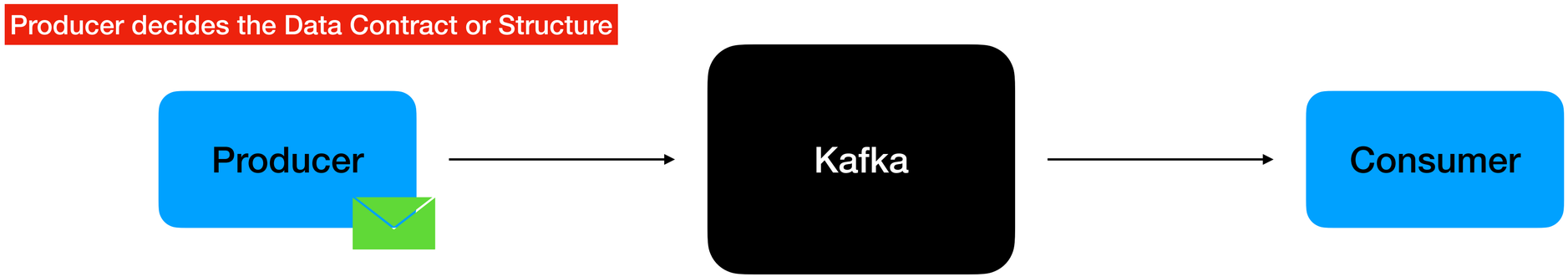
Thank You!

Prerequisites

- Java 17 (Java 11 or Higher is needed)
- Docker
- Prior Java Experience is a must
- Prior Kafka Experience is a must
- Prior Spring Framework/SpringBoot is a nice to have
- **Intellij** or any other IDE

Data Contract
&
Serialization
in Kafka

Data Contracts



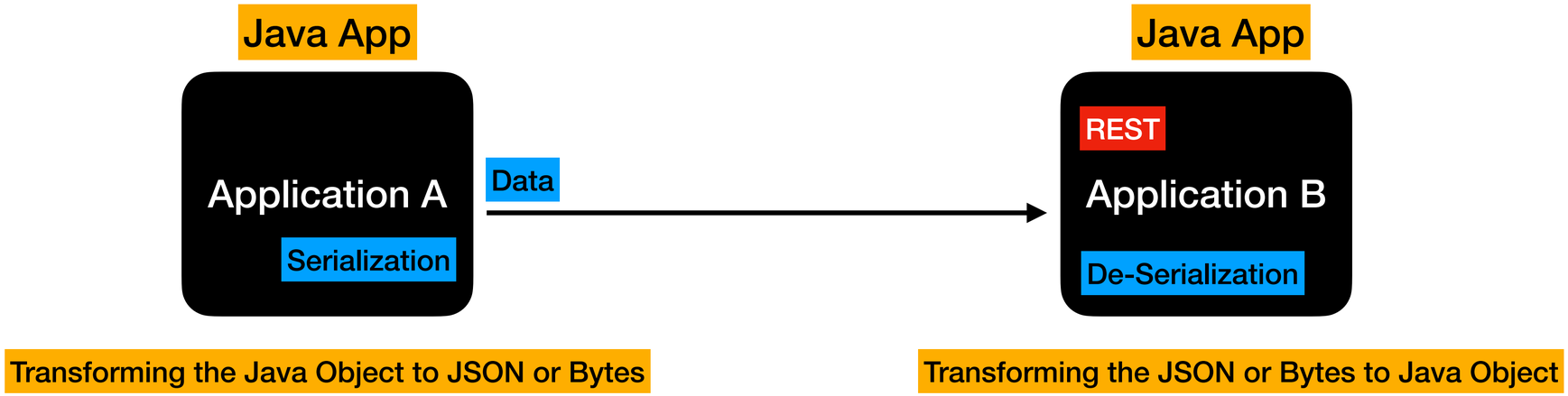
- Producers and Consumers are decoupled and independent of each other
- Consumer is indirectly coupled with the data format that’s been sent by the producer

Serialization

Wikipedia:

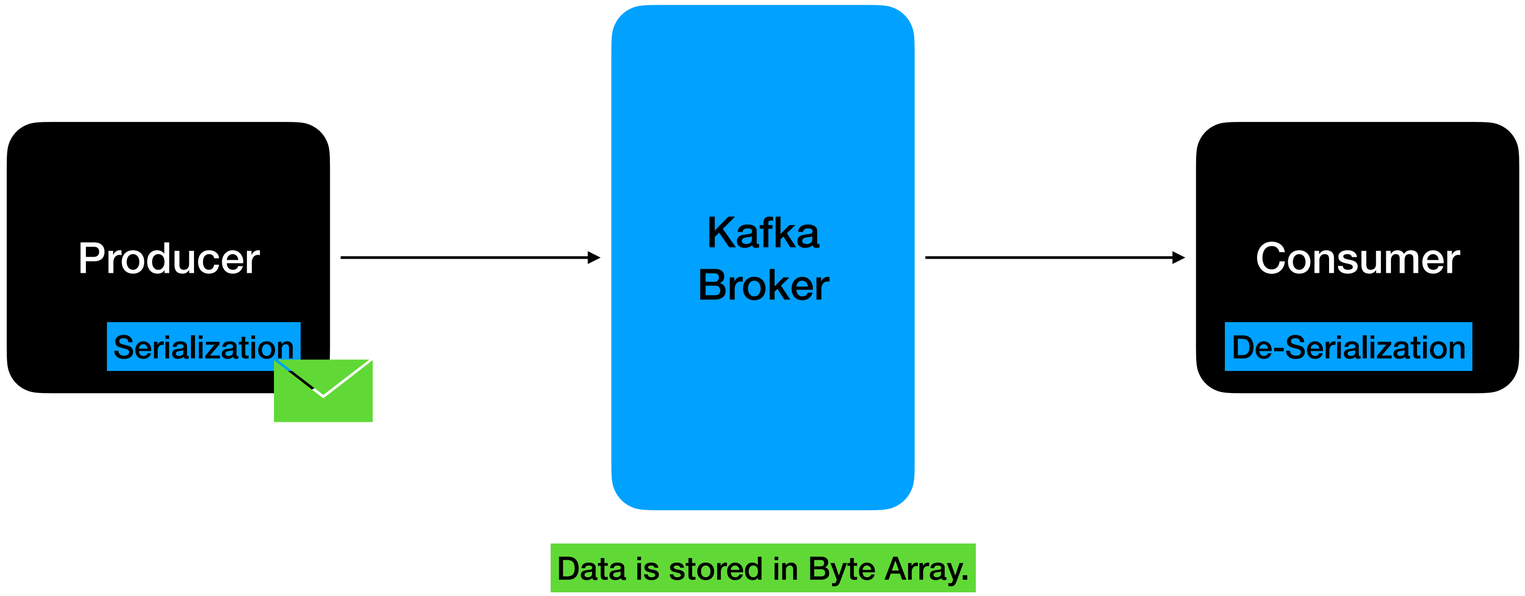
In computing, **serialization** is the process of translating a data structure or object state into a format that can be stored (in a file or memory data buffer) or transmitted (over a computer network) and reconstructed later (possibly in a different computer environment)

Serialization - Explained



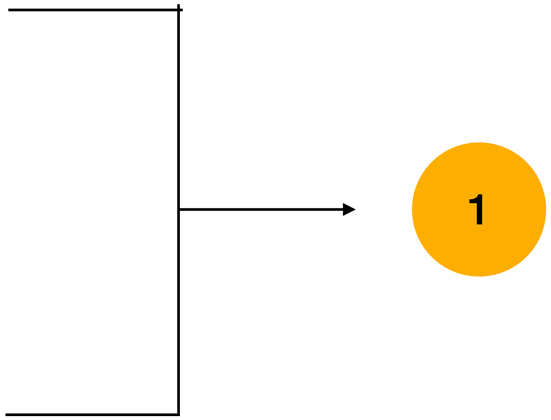
How Serialization is related to Kafka?

Serialization in Kafka



Next Steps ?

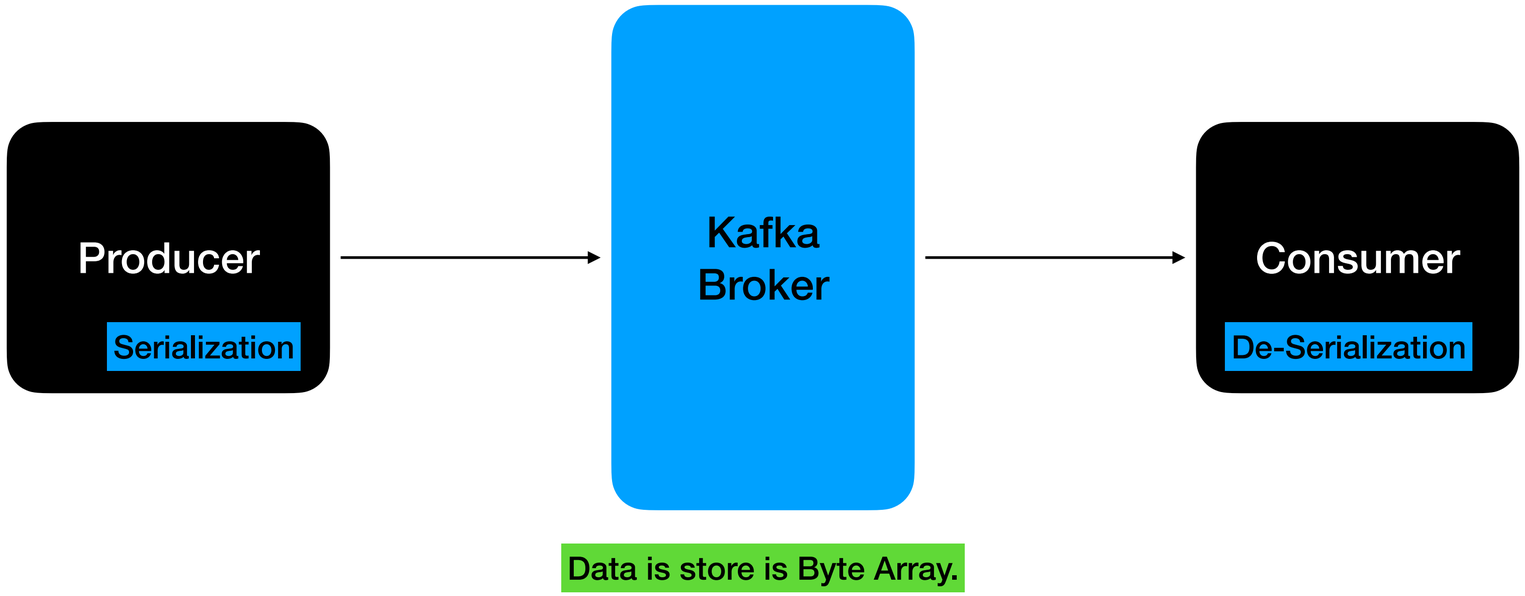
- Different Serialization Formats:
  - Faster data transfer
  - Compact data storage



- How to remove the Consumer Dependency on the data format from the Producer using **Schema Registry**? 2

Serialization Formats

Serialization in Kafka



Different Serialization Formats

- Binary Serialization Format

- Plaintext Serialization Format

Binary Serialization

- This serializes the data to byte array
- Not Human readable
- This is more efficient because the data is compact and less memory overhead
- Serialization in generally faster

Plaintext Serialization

- This serializes the data into an ecoded text.
- Human Readable
- Data is verbose and it can be inefficient
- Serialization in slower compared to Binary Serialization

Binary Serialization

- AVRO
- ProtocolBuf
- Thrift

→ Schema IDL

Define a Schema for the data structure and make sure the data is always aligned to that schema.

Plaintext Serialization

- JSON → JSON Schema
- XML → Schema IDL

Serialization Formats popular in Kafka

- JSON - Human readable ,and its widely in spaces outside Kafka.
- AVRO - Binary Serialization System and natively supported in Schema Registry.
- ProtocolBuf - Built by google and really popular in gRPC space.
- Thrift - Built by Facebook and open sourced in 2020

Why AVRO over other formats?

- **AVRO** Schema can still defined in JSON
    - **Protocolbuf** and **Thrift** have their own language which requires a steep learning curve.
- Compared with JSON, Avro data is compact and processing records are really faster compared to JSON.

https://www.confluent.io/blog/avro-kafka-data/

CONFLUENT PLATFORM

Why Avro for Kafka Data?

JAY KREPS

FEB 25, 2015

If you are getting started with Kafka one thing you'll need to do is pick a data format. The most important thing to do is be consistent across your usage. Any format, be it XML, JSON, or ASN.1, provided it is used consistently across the board, is better than a mishmash of ad hoc choices.

But if you are starting fresh with Kafka, you'll have the format of your choice. So which is best? There are many criteria here: efficiency, ease of use, support in different programming languages, and so on. In our own use we have found Apache Avro to be one of the better choices for stream data.

What is Avro?

Avro is an open source data serialization system that helps with data exchange between systems, programming languages, and processing frameworks. Avro helps define a binary format for your data, as well as map it to the programming language of your choice.

Why Use Avro with Kafka?

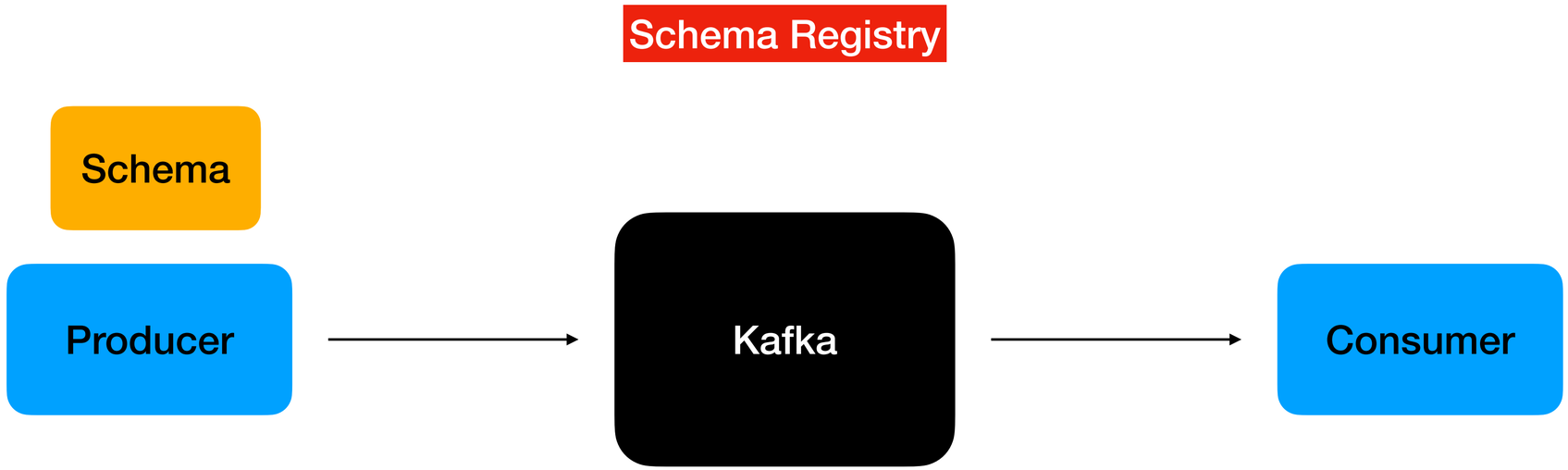
AVRO

What is AVRO ?

- AVRO is a data serialization system and it helps to exchange data between two systems using the Binary Serialization format.
- AVRO is a compact and fast binary data format
  - It takes up less space
  - It has a direct impact on memory and speed of transfer of data
- It has the support for most of popular programming languages like Java, C++, Go , Python and more.
- AVRO also has the capability to make Remote procedure call.

Why AVRO ?

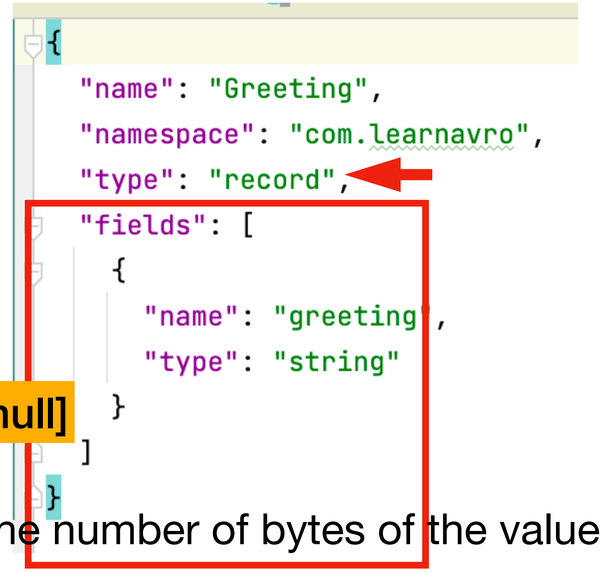
- AVRO has the support for Schema IDL
  - Data Owner defines a Schema in JSON format for the data structure that they would want to communicate to the other system



  - This architecture also makes Data Changes and Schema Evolution fairly less complicated
  - We can also communicate AVRO Records without a Schema Registry

Why AVRO ?

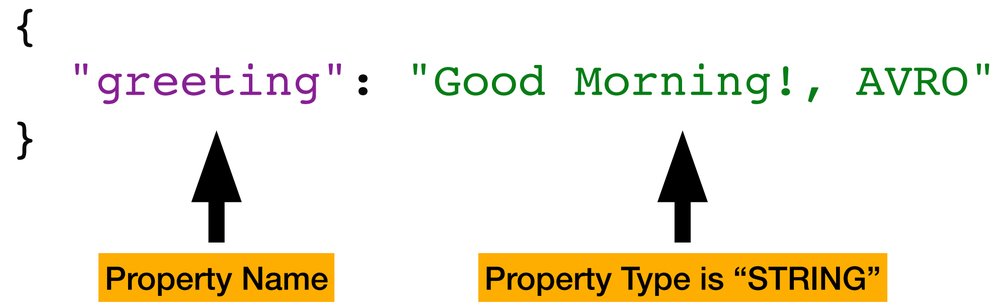
- AVRO has the support for rich data structures
- Primitive Types
  - String, bytes, int ,long, float, double, boolean and null
- Complex Types
  - enum
  - arrays
  - maps
  - record - This type is normally used to hold multiple complex types.
  - union - This is used to represent a field can hold multiple types. [String , null]
  - fixed - This is used to represent a field can be of a fixed size, specifying the number of bytes of the value



Let’s build a simple AVRO Schema

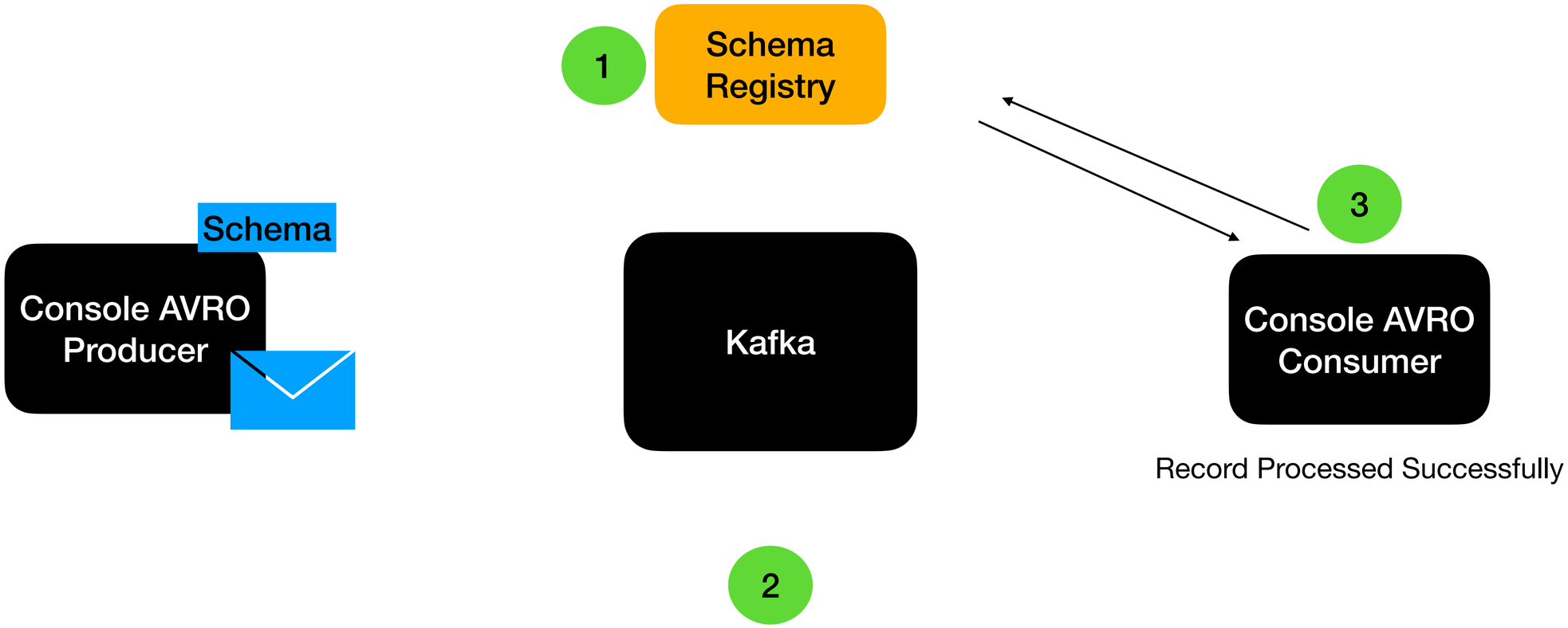
Build an AVRO Schema - UseCase

- Sample AVRO Message



Avro Messages using Console AVRO Producer/Consumer

Produce/Consume AVRO Messages using CLI



How to use AVRO in a Java Project?

Working with AVRO

- Place the AVRO schema file in a the **src** directory of the Java Project
  - src/avro [Default Path]
- Generate Java Classes from the .avsc file
- Use the Generated Java Classes in the Producer & Consumer

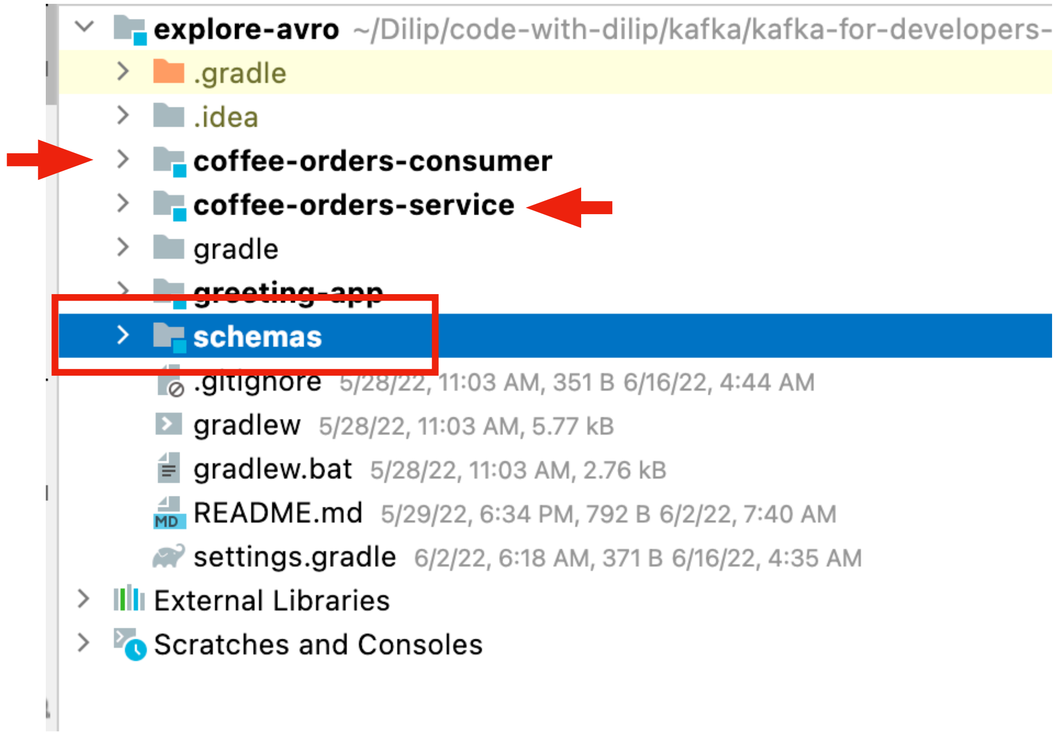
Coffee Order Service

Coffee Order Service

Sample Coffee Order Message

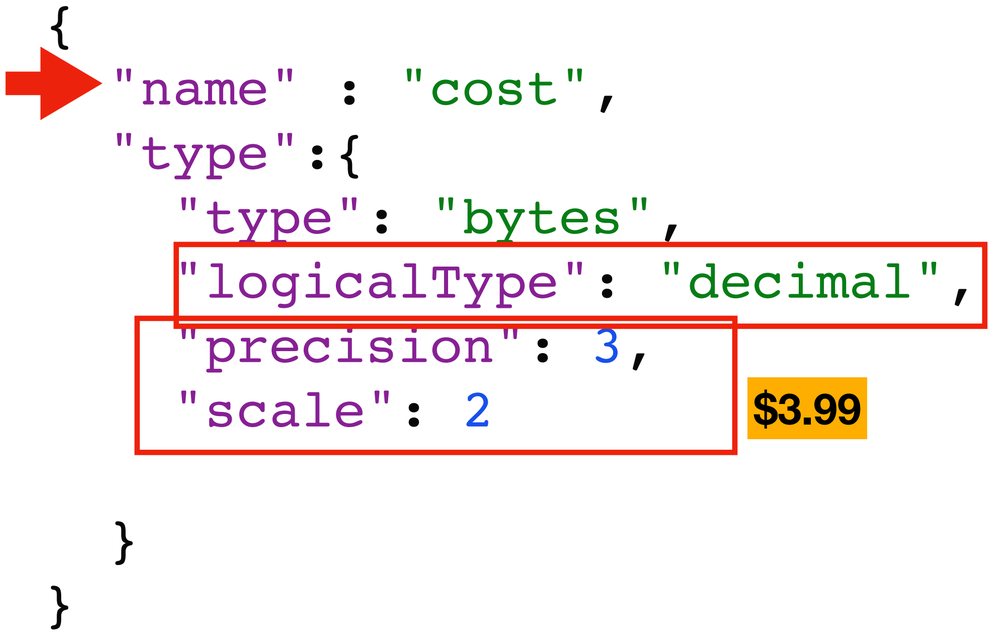
Project SetUp

- The is a multi module project
- Avoids Code duplication



Logical Types in AVRO

- Logical Types are used to represent Decimals, UUID, Date, Timestamp, Time.
- AVRO uses some additional attributes to represent a particular logical type

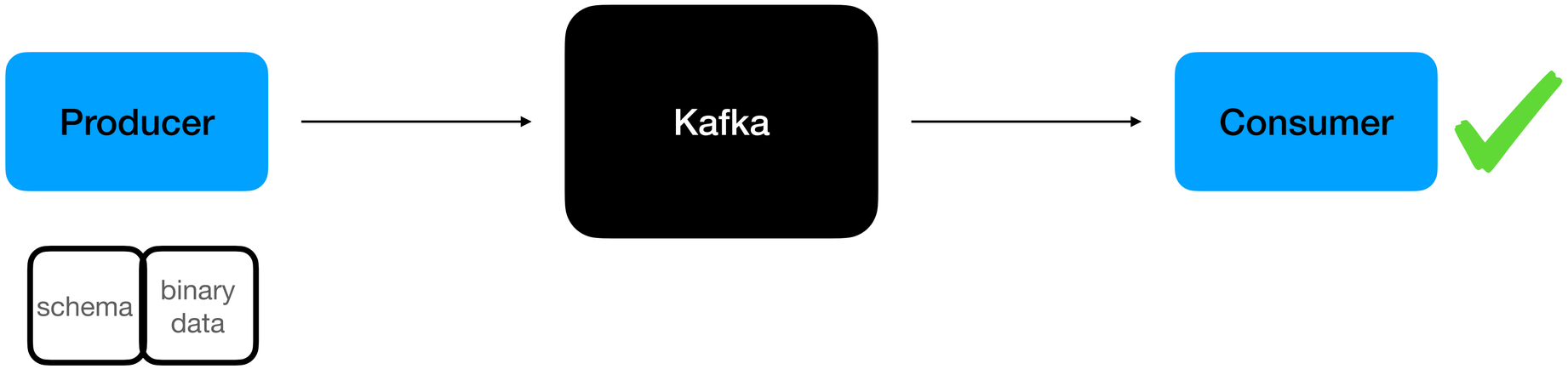


What’s inside an AVRO Record ?

What’s inside an AVRO record ?

- This pattern makes the message bulky
- Providing the schema with every event in Kafka leads to inefficient network bandwidth and adds storage overhead.

Evolving a Schema



- Update the consumer first and then produce the message
- Consumer processes the record without any issues

Lets not update the Consumer at all and publish the new AVRO record

Schema Registry

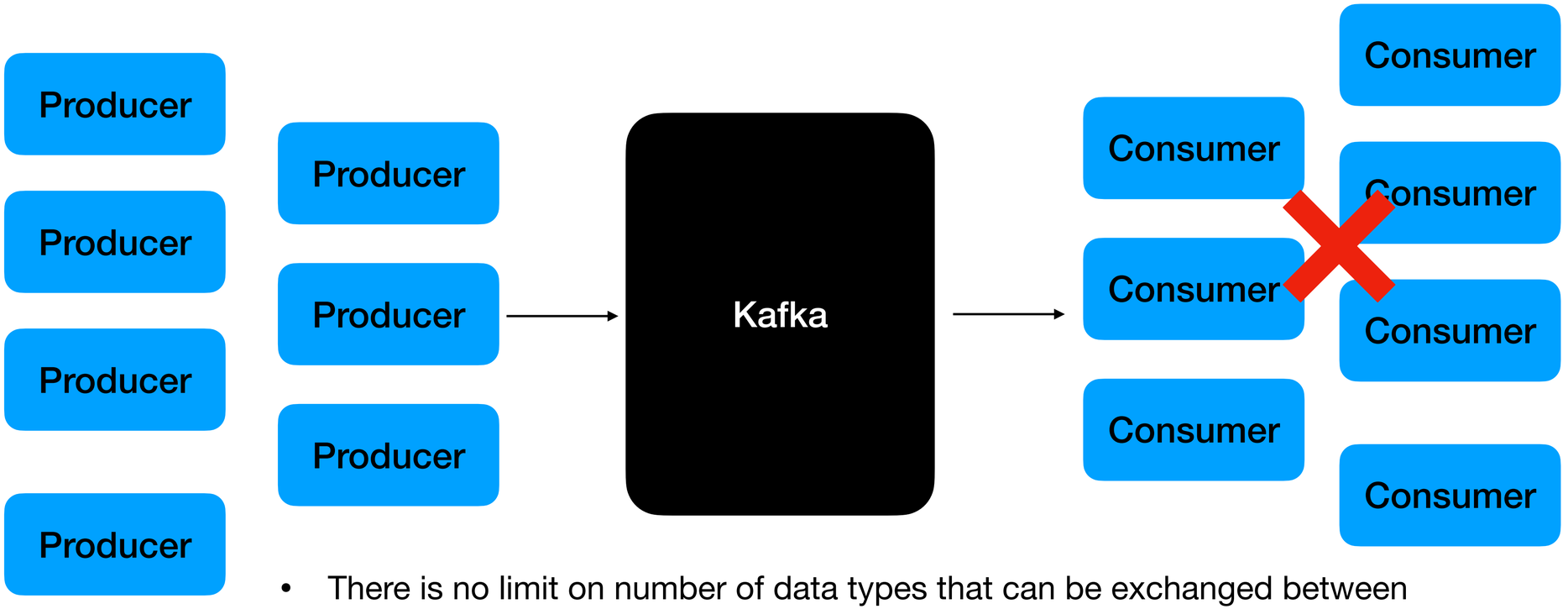
Why Schema Registry ?

- Producers and consumers are decoupled and they exchange data using the Kafka Broker
- But the consumer is indirectly coupled to producer to understand the data format
- Anytime business requirements change data may be needed to change.

Handling Data Changes

- Business needed to make change and the producer modified the data

Kafka Architecture in Prod



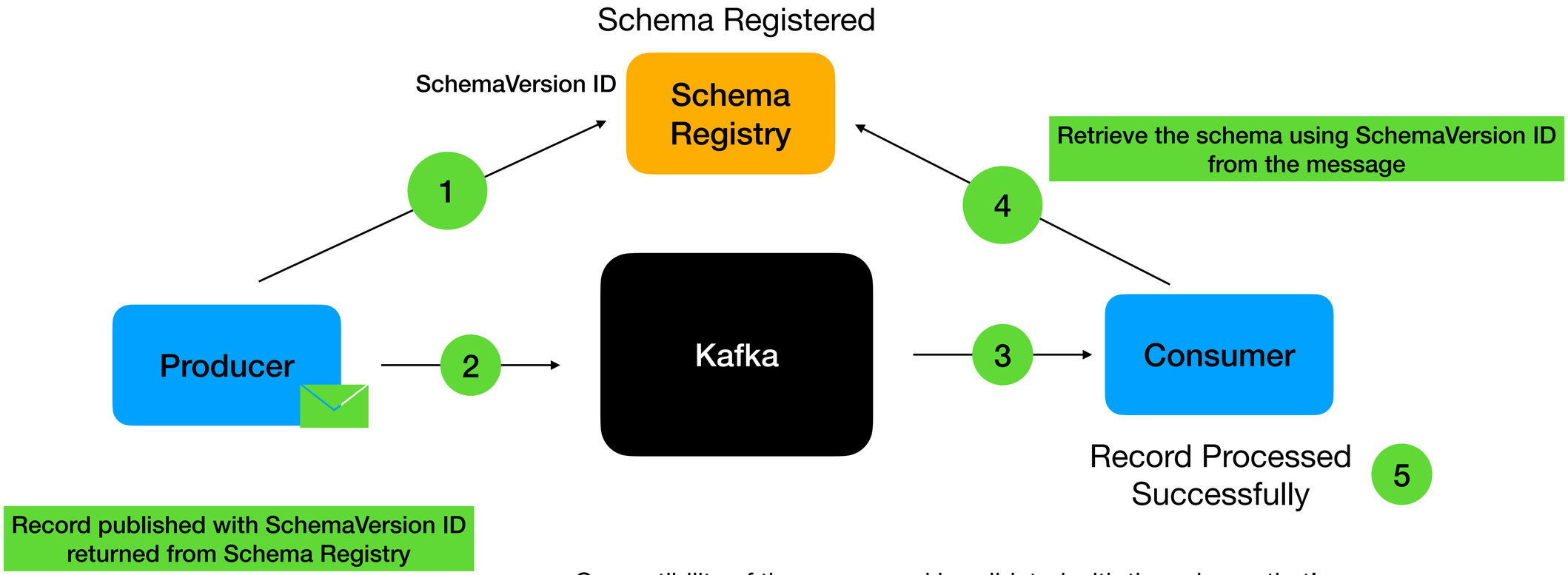
- There is no limit on number of data types that can be exchanged between the apps
- With this kind of architecture the failure is very likely in the consumer side
- Publishing any new changes requires approval from all the consuming apps

Schema Registry to the rescue.

Schema Registry

- Schema Registry is a product from **Confluent**
- It serves two purposes:
  - Enforcing Data Contracts
  - Handling Schema Evolution
- This one is under a Confluent Community License
  - You are allowed to use in production
  - But not allowed to build competing products using Schema Registry

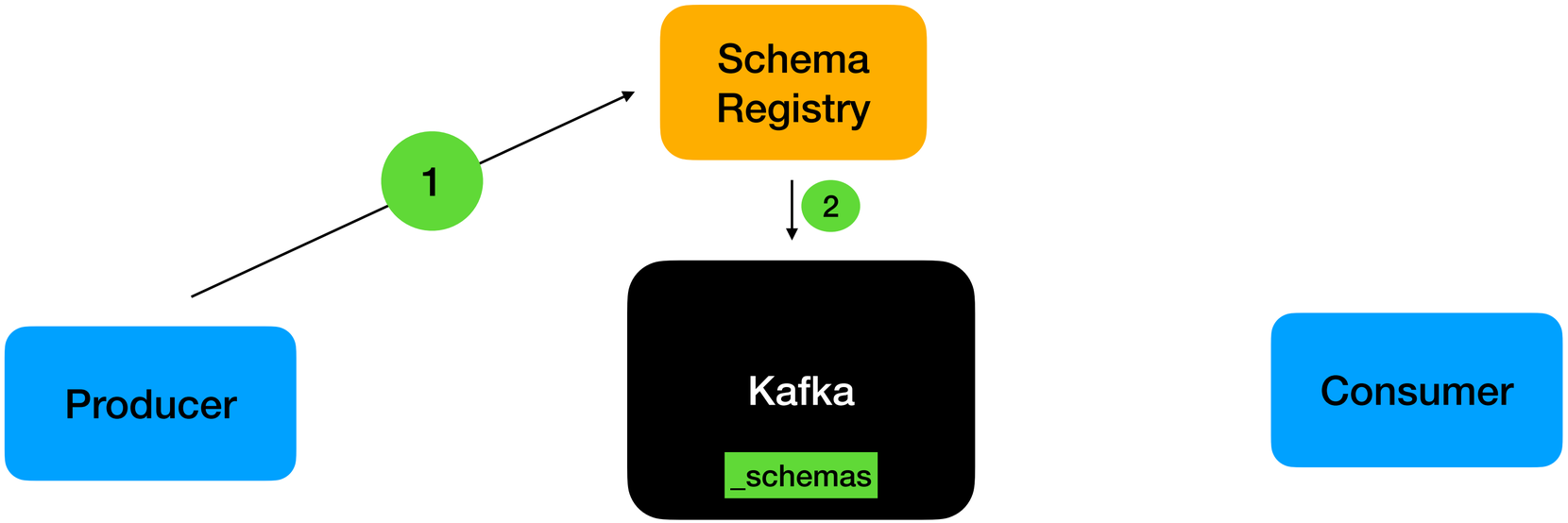
Kafka Architecture with Schema Registry



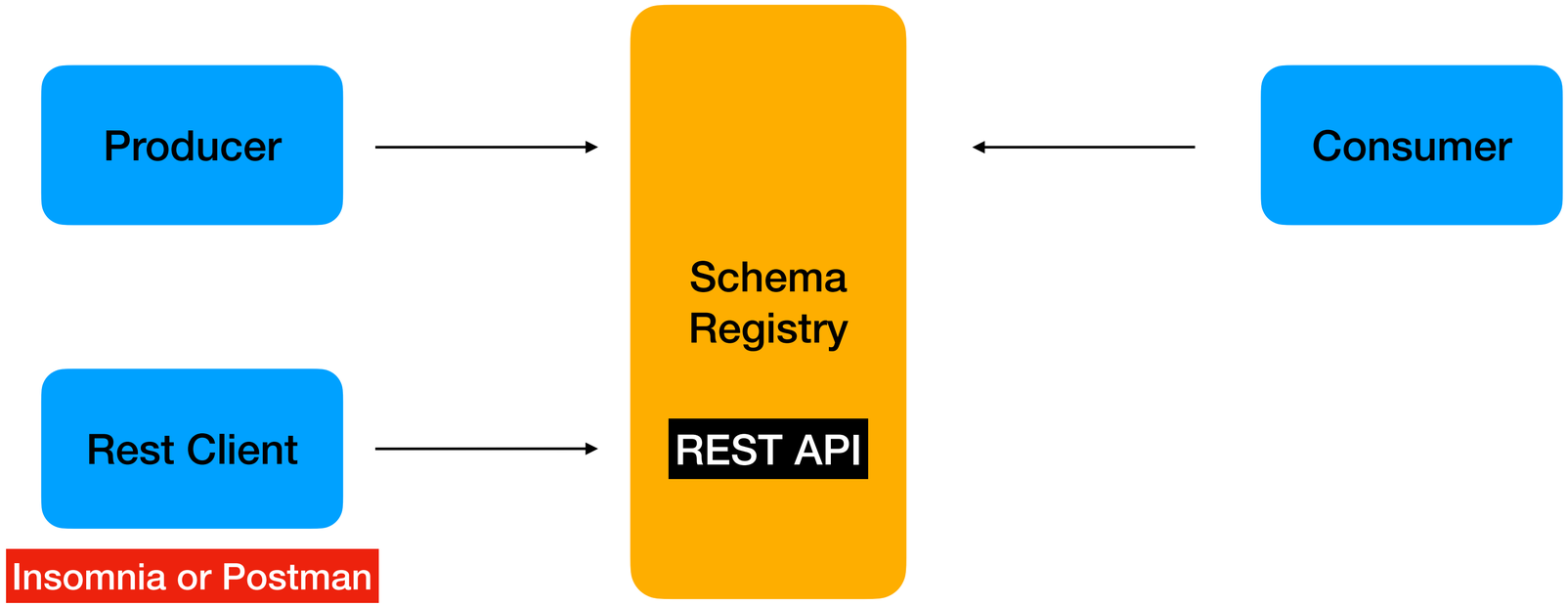
- Compatibility of the new record is validated with the schema that's returned
- AVRO record is lighter because there is no schema present in every record

Schema Registry Internals
&
Interacting with Schema Registry

Where does the Schema gets Stored ?



Different ways to Interact with Schema Registry?



Schema Registry REST API

- Schema Registry REST Resources
  - Subjects
  - Schemas
  - Config
  - Compatibility

- Fundamentally a scope in which the Schema's evolve.

- This resource is used to represent a Schema.

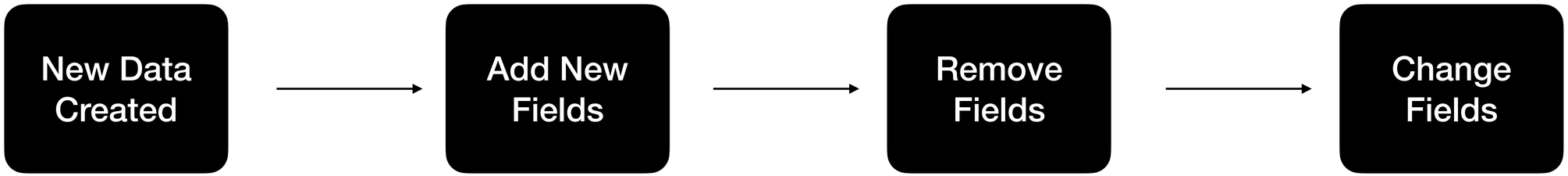
- This resource is used to update cluster level config for the Schema Registry

- This resource is used to check the compatibility between Schemas

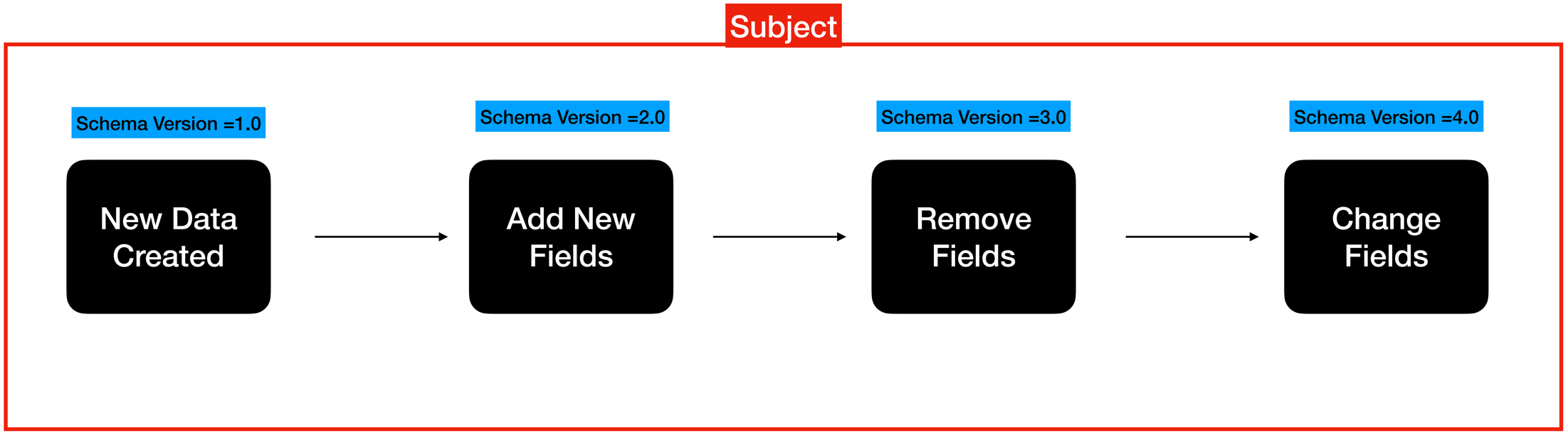
Data Evolution using Schema Registry

Data Life Cycle

Data Evolution



Data/Schema Evolution



Data/Schema Evolution

Anytime the data/schema evolves, the downstream consumers should be

able to handle the old and new schema seamlessly

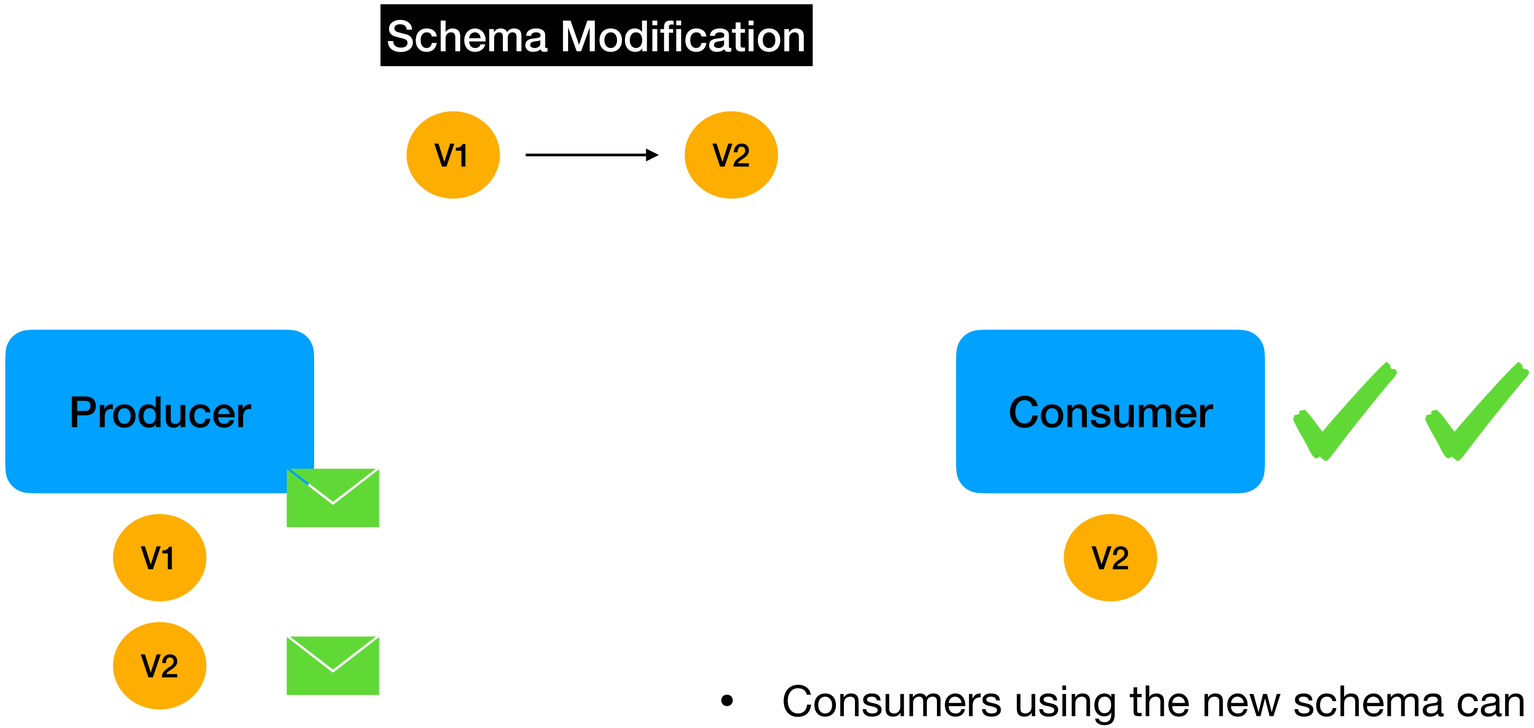
Compatibility Types in Schema Registry

Compatibility is configured through “/config” endpoint.

Backward Compatibility

- **BACKWARD (Default)**

- This means the consumers using the new Schema can read the data produced with the old schema



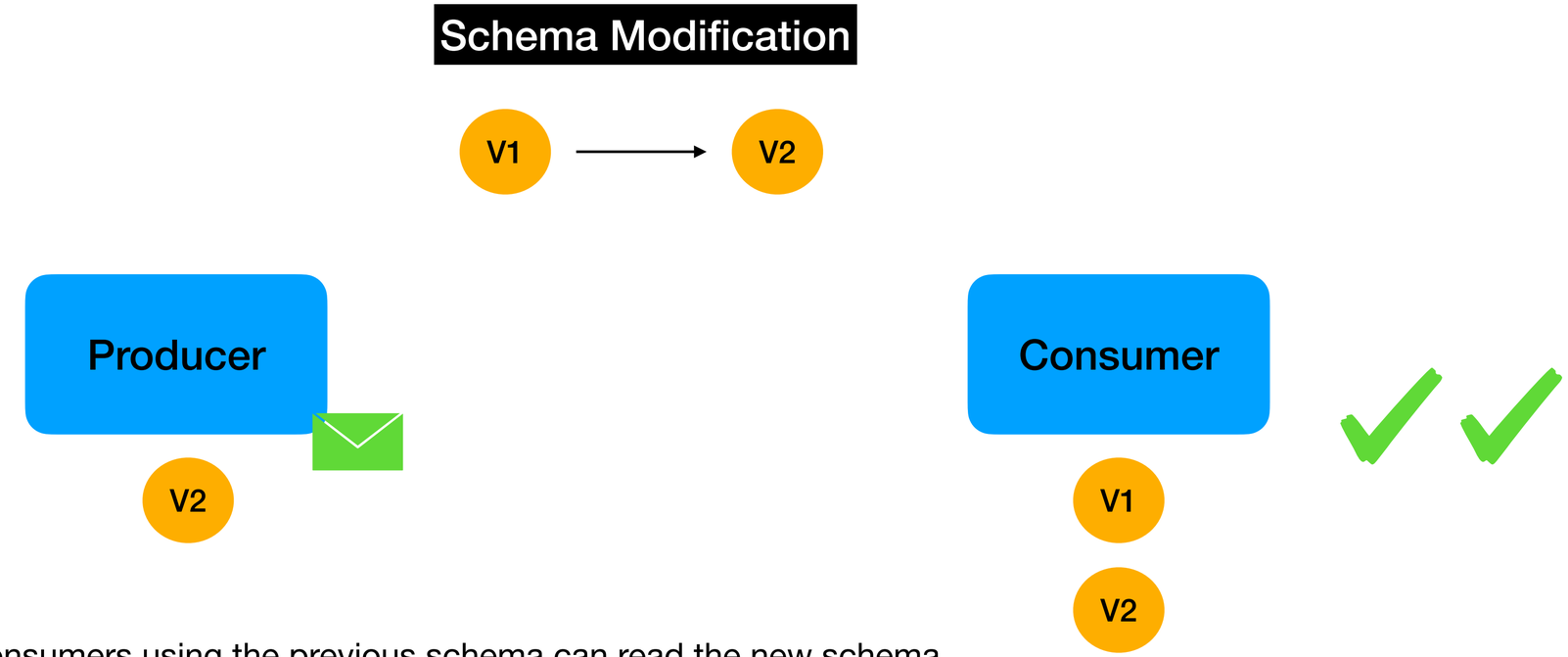
- Consumers using the new schema can read the old and new schema seamlessly.

Backward Compatibility

- Rules:
  - Delete Fields
  - Add Optional Fields
- Upgrade First
  - Consumers
- If the consumer using the new schema needs to be able to process data written by all registered schemas, not just the last two schemas, then use **BACKWARD_TRANSITIVE**

Forward Compatibility

- **FORWARD** compatibility means the data produced with a new schema can be read by consumers using the last schema.



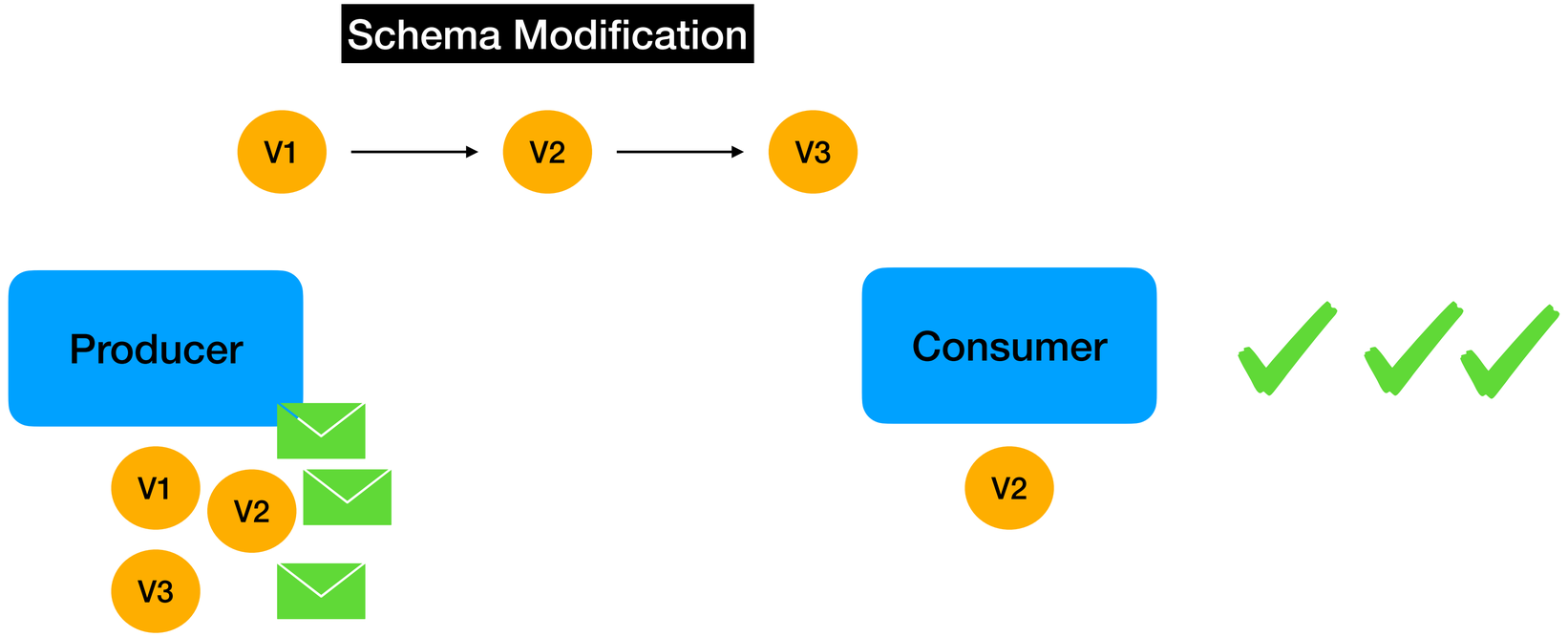
- Consumers using the previous schema can read the new schema without any issues
- Compatibility check happens between the last schema and the current schema

Forward Compatibility

- Rules:
  - Add Fields
  - Delete Optional Fields
- Upgrade First
  - Producers
- **FORWARD_TRANSITIVE**
  - If data produced with a new schema needs to be read by consumers using all registered schemas, not just the last two schemas, then use **FORWARD_TRANSITIVE**

FULL Compatibility

- **FULL** compatibility means its both forward and backward compatible
- It means the **Old data** can be read with the **New** schema or **New data** can also be read with the **Old schema**.



FULL Compatibility

- Rules:
  - Restrictive compared to the Backward and Forward compatibility
  - Add Optional Fields
  - Delete Optional Fields
- Upgrade First
  - Producers or Consumer(Any order)
- **FULL_TRANSITIVE**
  - If data produced with a new or old schema needs to be read by consumers using all the registered schemas, then use **FULL_TRANSITIVE.**

None Compatibility

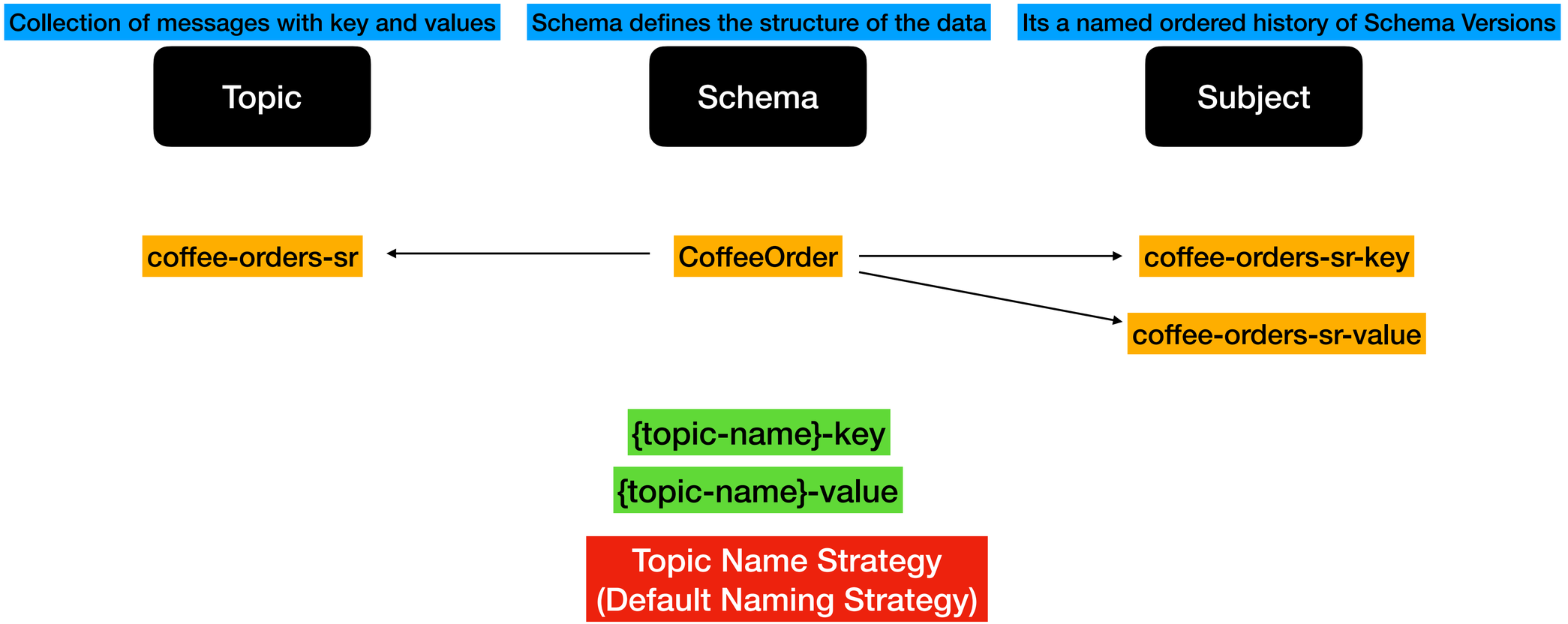
- **NONE**
- This means the schema compatibility checks are disabled
- When to use this NONE compatibility check?
  - Changing the type of the field
    - Example : String to Integer
  - Changing the name of the field from one to another
    - Example : name -> fullname

None Compatibility

- Upgrade First
  - Producers and Consumers needs to upgrade at the same time.
- Recommended Option
  - Use case like these will require us to create a new schema and new topic.
  - Introduce a new field and delete the old field

Schema Naming Strategies

Schema Naming Strategies



Different Subject Naming Strategies

- TopicNameStrategy
- RecordNameStrategy
- TopicRecordNameStrategy

Different Subject Naming Strategies

TopicNameStrategy

- Use it when single type of message should be published in a topic
- This means that the schemas of all messages in the topic must be compatible with each other.
- Subject name is derived using this technique:
  - {topic-name}-key
  - {topic-name}-value

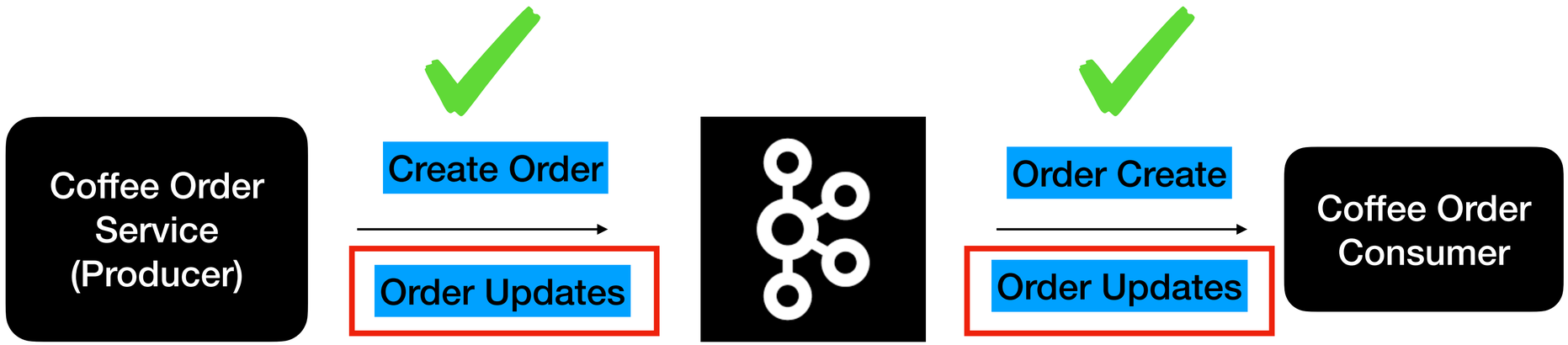
RecordNameStrategy

- Use it when you have use case where multiple types of related events can be published into the topic and the ordering of events needs to be maintained.
- Thus, the schema registry checks the compatibility for a particular record type, regardless of topic.
- Subject name is derived from the fully qualified record name.
  - **com.learnavro.domain.generated.CoffeeOrder**
- **Usecase**:
  - CoffeeOrder and the related update events
  - For Banking , related transactions needs to be ordered.

TopicRecordNameStrategy

- Use it when you have use case where multiple types of related events can be published into the topic.
- Schema registry checks compatibility to the **current topic** only
- Subject name is derived from the topic and fully qualified record name.
  - {topic-name}-{RecordName}
  - **coffee-orders-sr-com.learnavro.domain.generated.CoffeeOrder**

Coffee Order Service

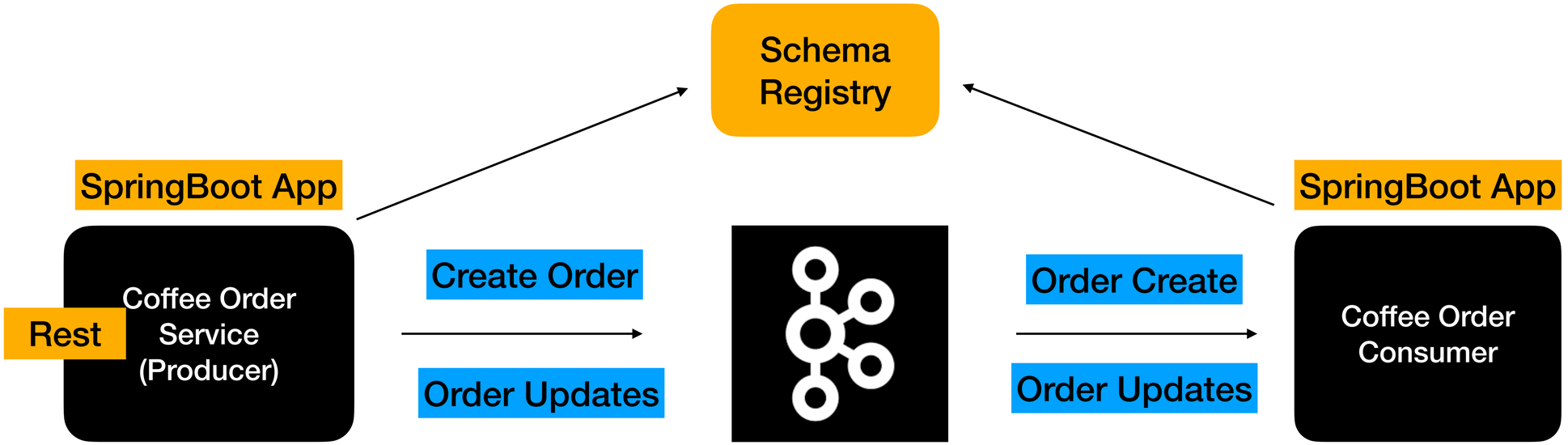


Sample CoffeeOrder Update JSON

```
{
  "id": 358,
  "status": "READY_FOR_PICK_UP"
}
```

Possible Values are : `PROCESSING, READY_FOR_PICK_UP`

Coffee Order Service



- Build a **RESTFUL** Service
- **SpringBoot** is a popular JVM framework