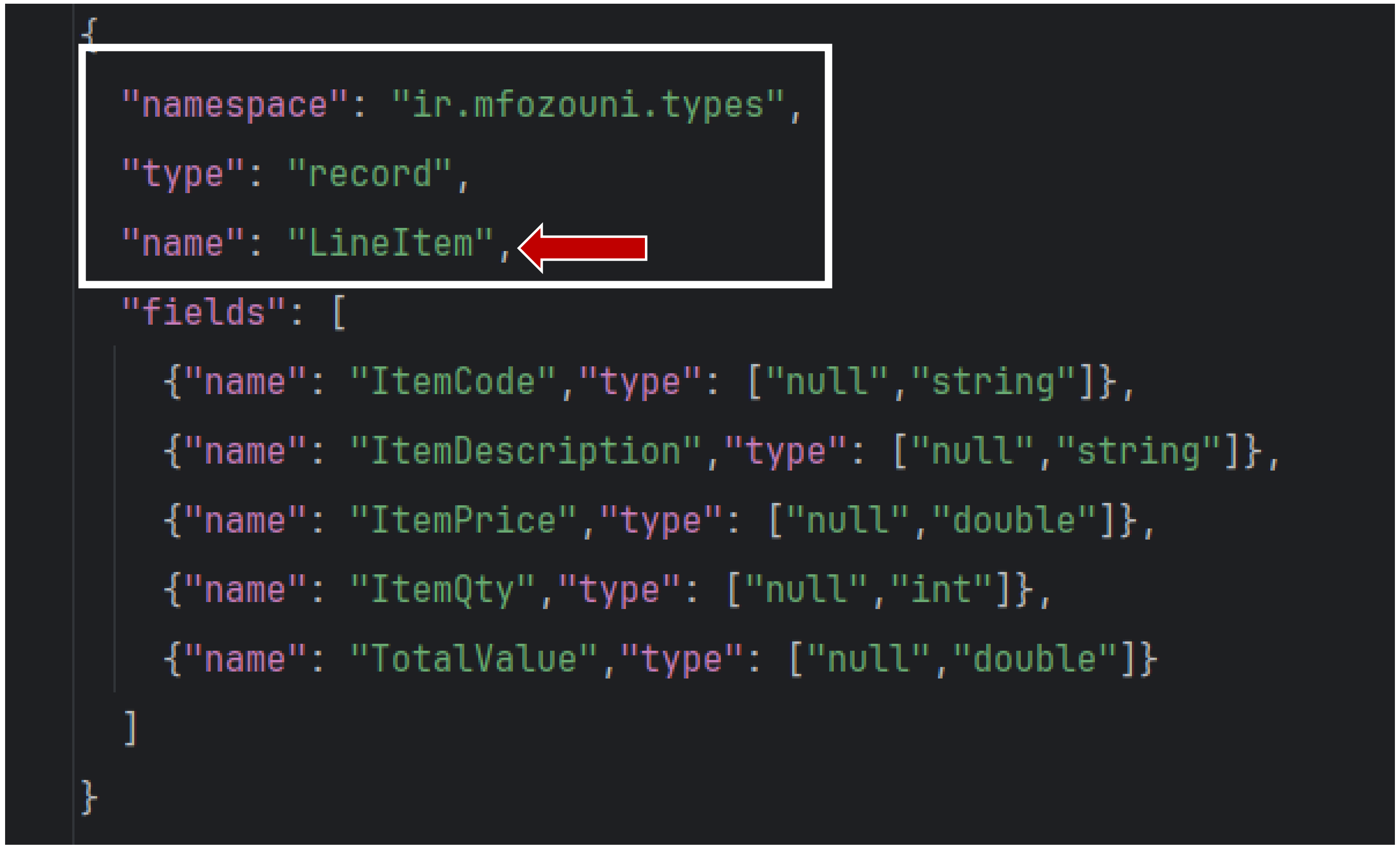
AVRO format and Schema Registry

What is Apache AVRO format?

Avro

Avro stores the data definition in JSON format making it easy to read and interpret; the data itself is stored in binary format making it compact and efficient.

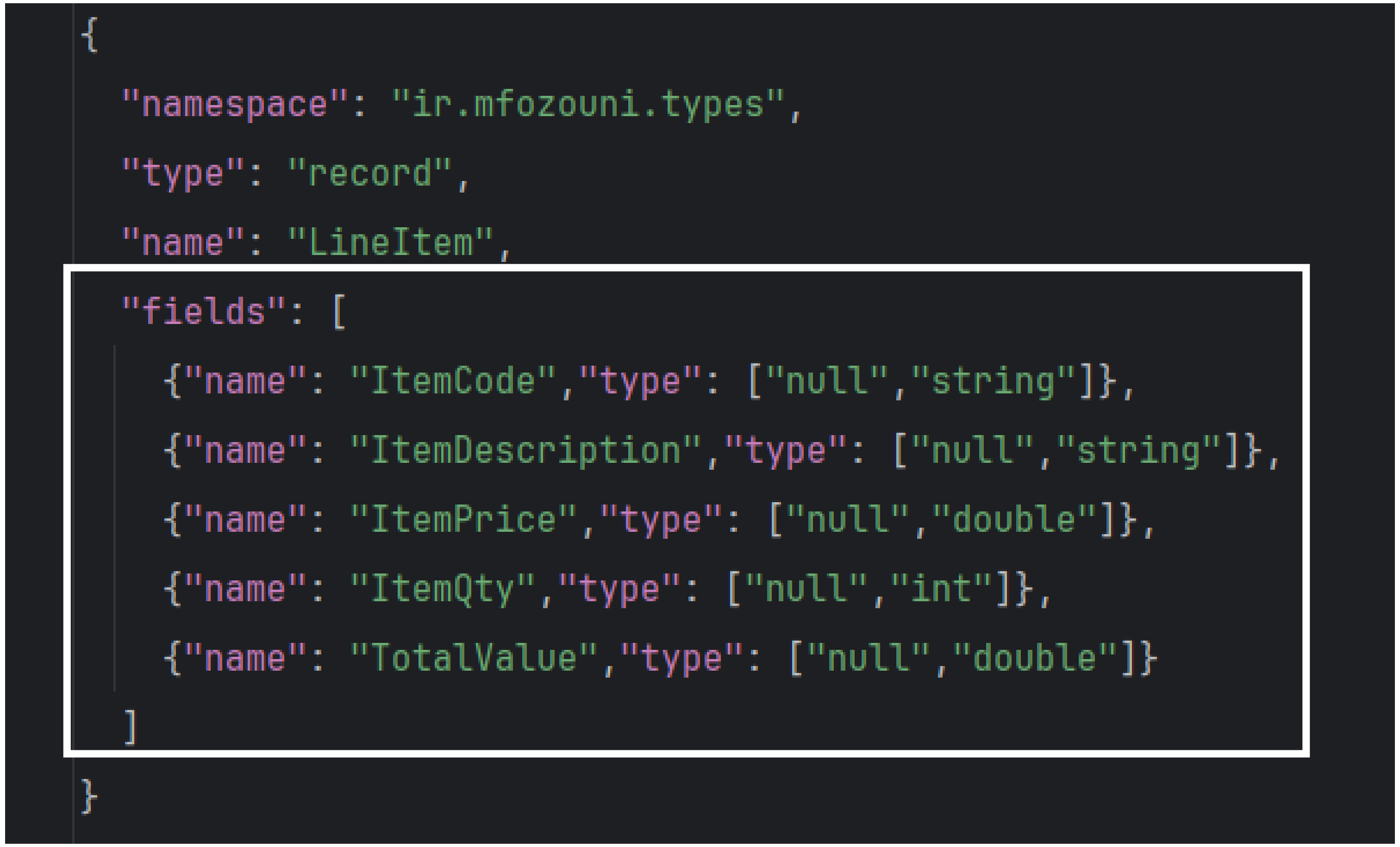
So, Avro is suitable for big data.



```
{
  "namespace": "ir.mfozouni.types",
  "type": "record",
  "name": "LineItem",
  "fields": [
    {"name": "ItemCode","type": ["null","string"]},
    {"name": "ItemDescription","type": ["null","string"]},
    {"name": "ItemPrice","type": ["null","double"]},
    {"name": "ItemQty","type": ["null","int"]},
    {"name": "TotalValue","type": ["null","double"]}
  ]
}
```

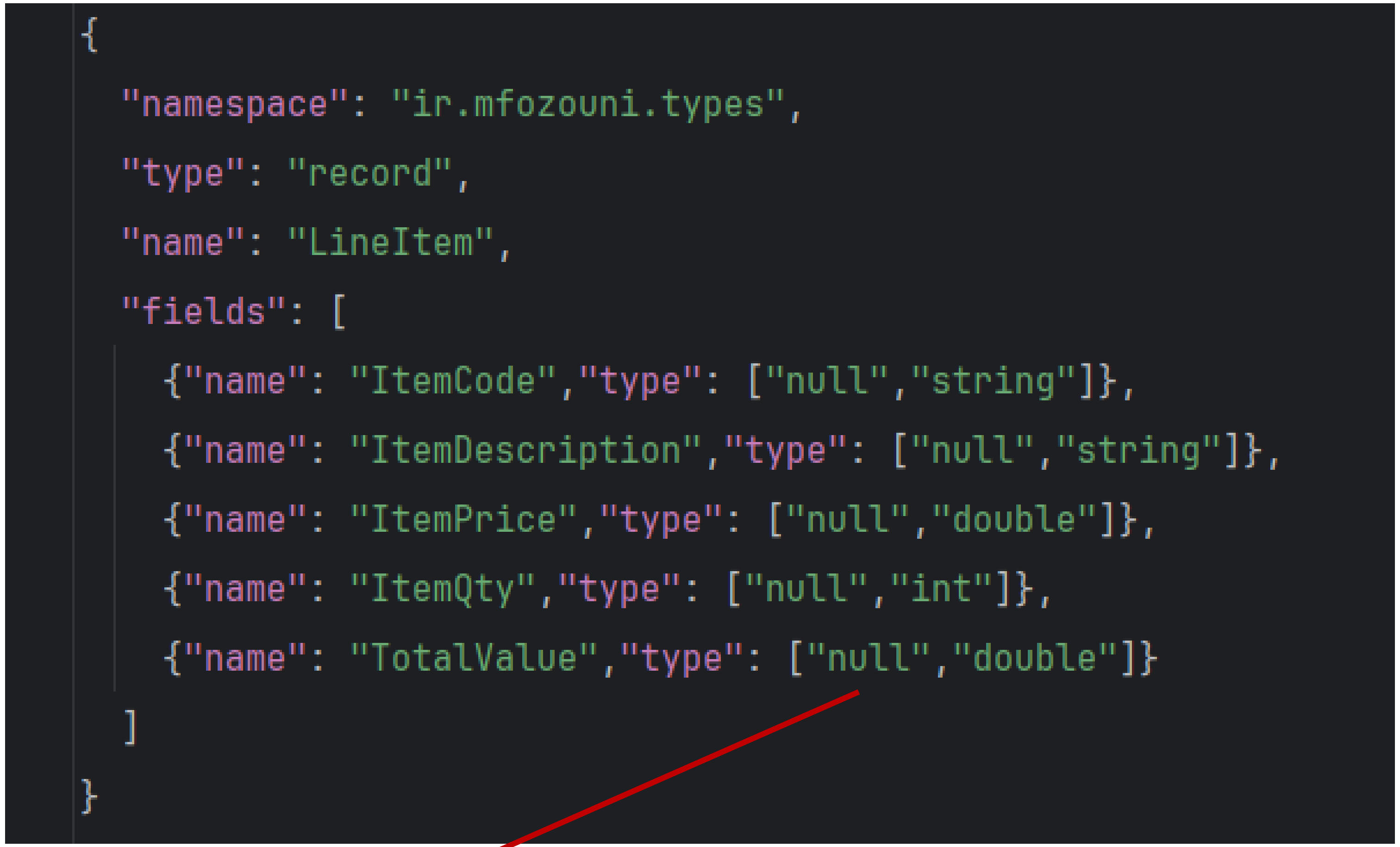
LineItem.avsc

Avro stores the data definition in JSON format making it easy to read and interpret; the data itself is stored in binary format making it compact and efficient.



```
{
  "namespace": "ir.mfozouni.types",
  "type": "record",
  "name": "LineItem",
  "fields": [
    {"name": "ItemCode","type": ["null","string"]},
    {"name": "ItemDescription","type": ["null","string"]},
    {"name": "ItemPrice","type": ["null","double"]},
    {"name": "ItemQty","type": ["null","int"]},
    {"name": "TotalValue","type": ["null","double"]}
  ]
}
```

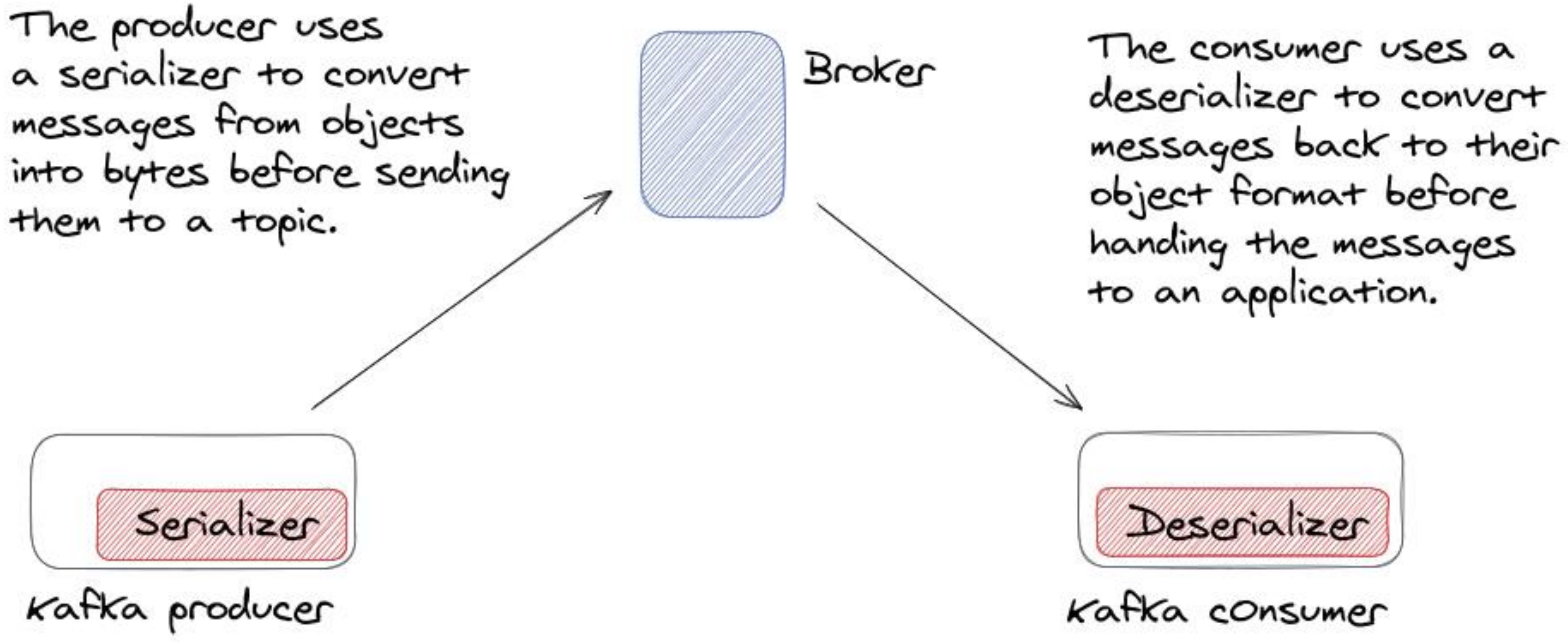
Now we should specify our fields and its corresponding types.



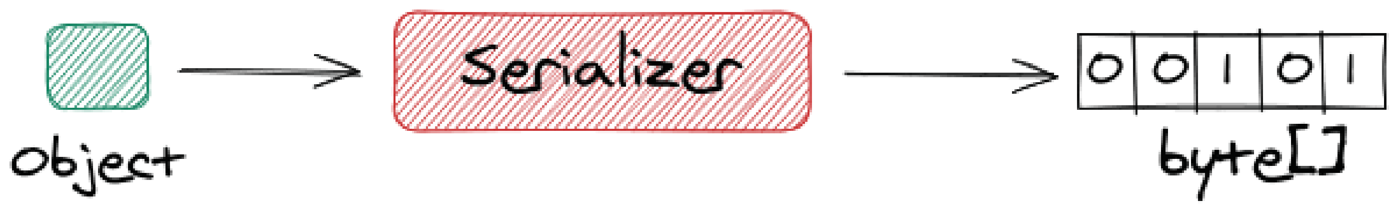
```
{
  "namespace": "ir.mfozouni.types",
  "type": "record",
  "name": "LineItem",
  "fields": [
    {"name": "ItemCode","type": ["null","string"]},
    {"name": "ItemDescription","type": ["null","string"]},
    {"name": "ItemPrice","type": ["null","double"]},
    {"name": "ItemQty","type": ["null","int"]},
    {"name": "TotalValue","type": ["null","double"]}
  ]
}
```

The field ***TotalValue*** is defined as type ["null", "double"]. It indicates that the field can either have a null value or a double value.

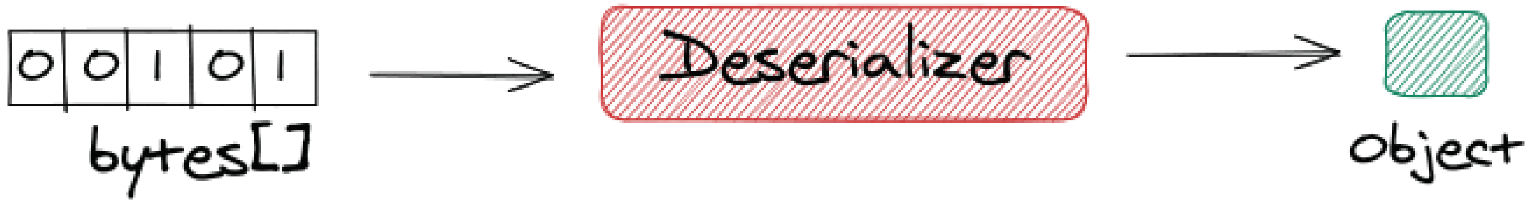
Schema Registry

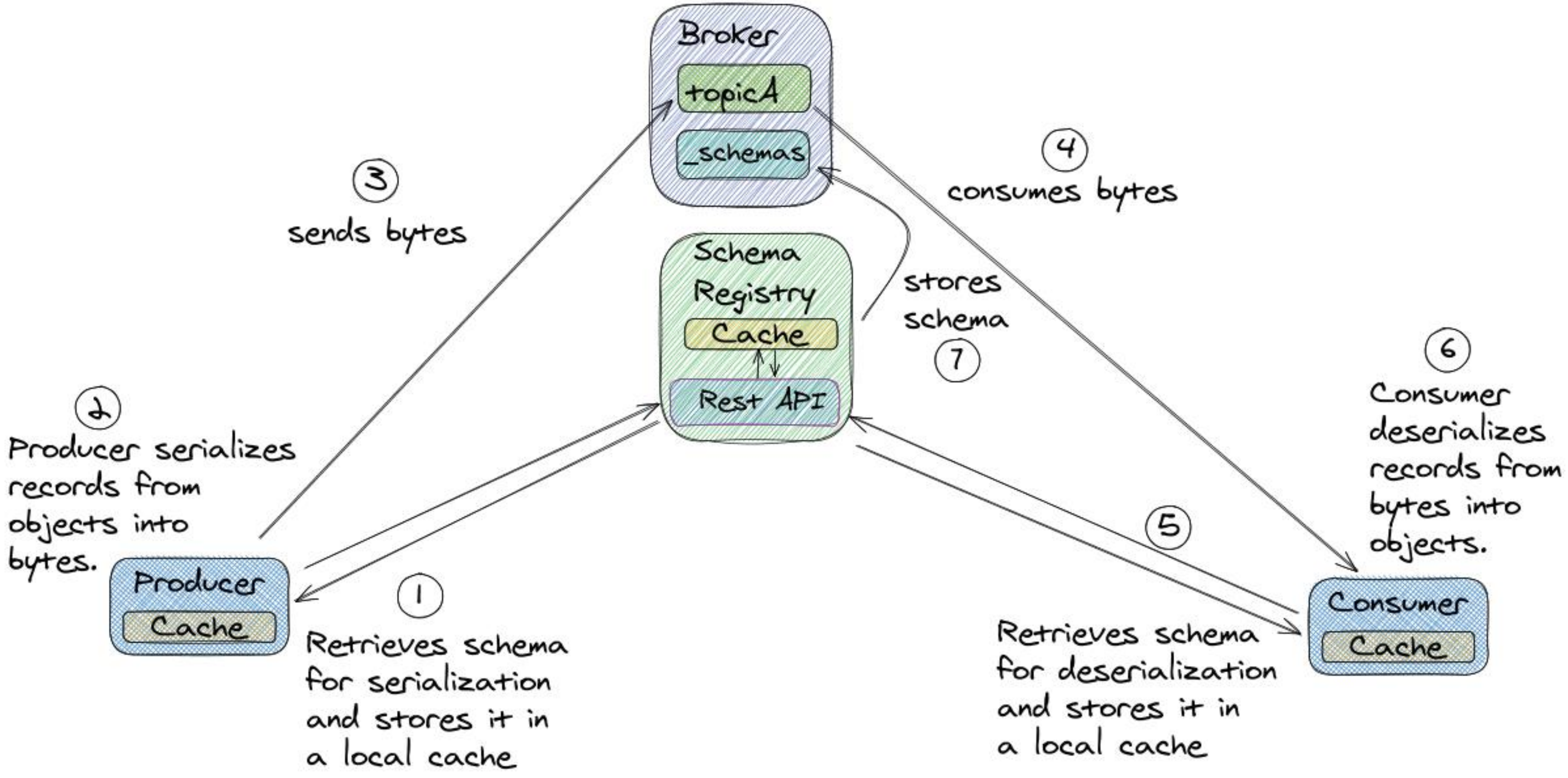

The producer uses a serializer to convert messages from objects into bytes before sending them to a topic.
Broker
The consumer uses a deserializer to convert messages back to their object format before handing the messages to an application.
Serializer
Kafka producer
Deserializer
Kafka consumer

Kafka Producers execute -> Serializer.serialize(T message)

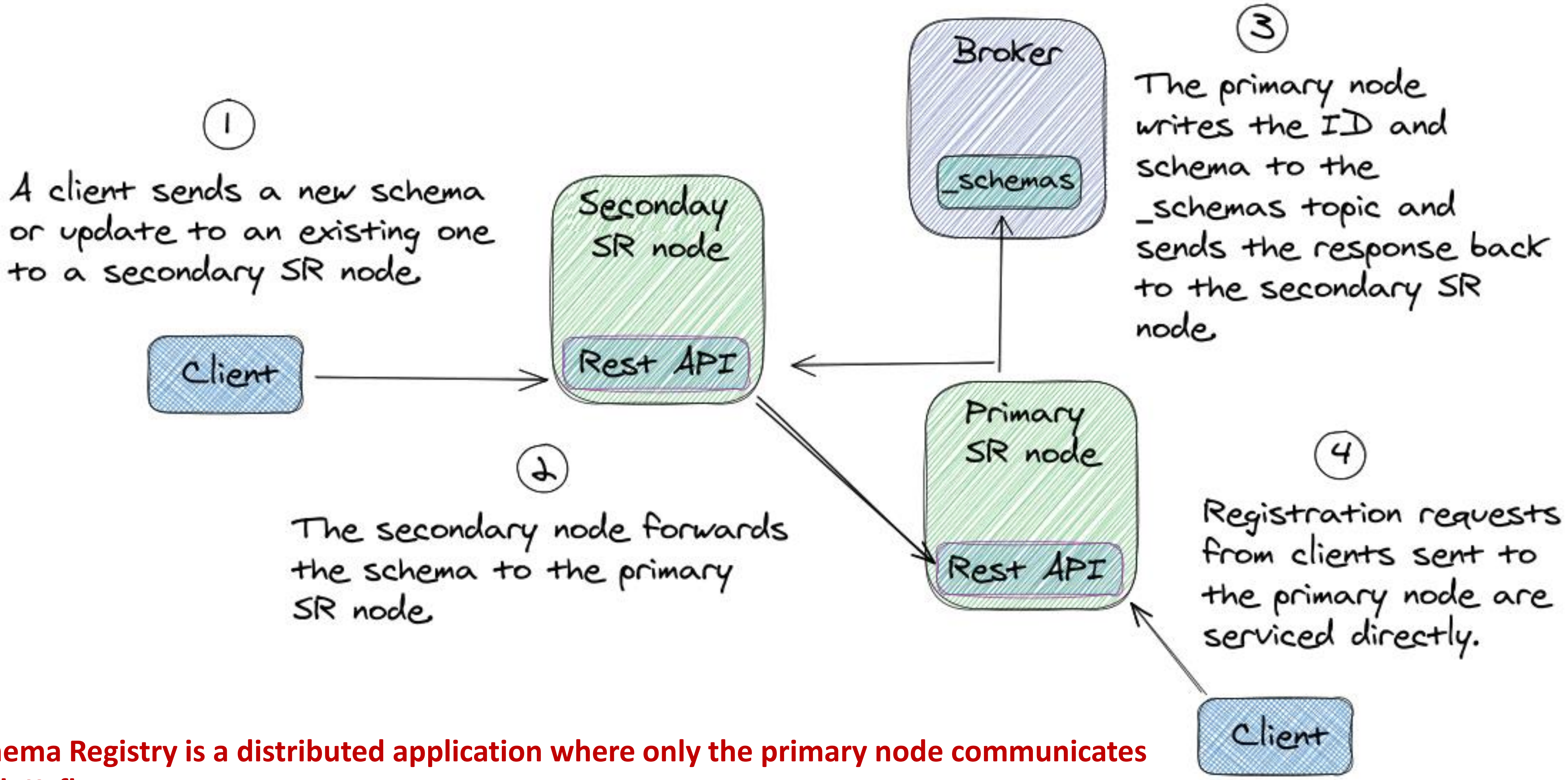


Kafka Consumers execute -> Deserializer.deserialize(byte[] bytes)

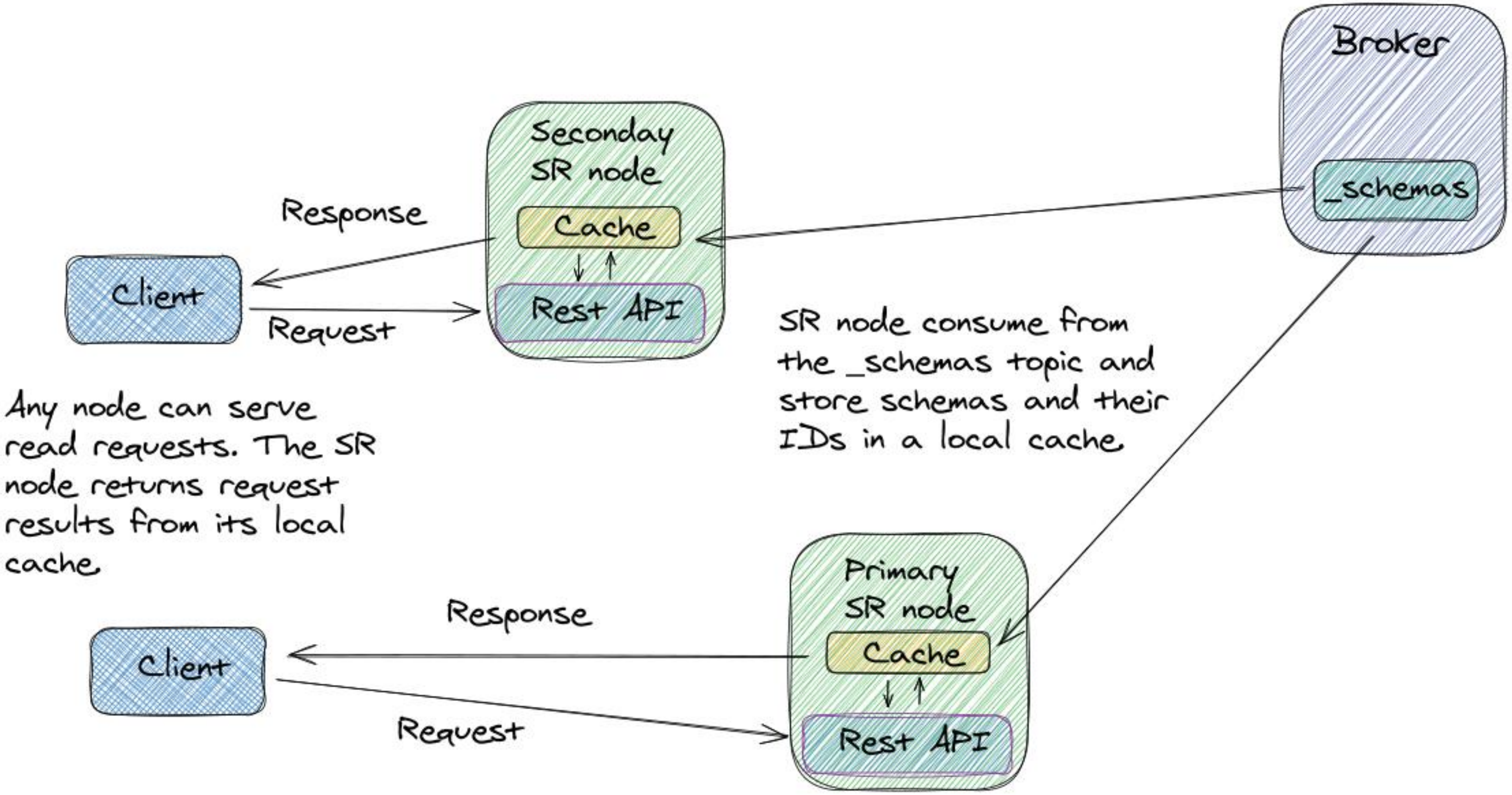



Broker
topicA
_schemas
③
sends bytes
④
consumes bytes
Schema Registry
Cache
Rest API
stores schema
⑦
⑥
Consumer deserializes records from bytes into objects.
②
Producer serializes records from objects into bytes.
Producer
Cache
①
Retrieves schema for serialization and stores it in a local cache
⑤
Retrieves schema for deserialization and stores it in a local cache
Consumer
Cache

Schema Registry ensures consistent data format between producers and consumers.


1
A client sends a new schema or update to an existing one to a secondary SR node
Client
Seconday SR node
Rest API
2
The secondary node forwards the schema to the primary SR node
Broker
_schemas
3
The primary node writes the ID and schema to the _schemas topic and sends the response back to the secondary SR node
Primary SR node
Rest API
4
Registration requests from clients sent to the primary node are serviced directly.
Client

Schema Registry is a distributed application where only the primary node communicates with Kafka.



Broker
_schemas
Seconday SR node
Cache
Rest API
Response
Client
Request
SR node consume from the _schemas topic and store schemas and their IDs in a local cache.
Any node can serve read requests. The SR node returns request results from its local cache.
Primary SR node
Cache
Rest API
Response
Client
Request

All Schema Registry nodes can serve read requests.

Schema Registry supports:

1. Avro
2. Protobuf (Protocol Buffers)
3. JSON Schema schemas