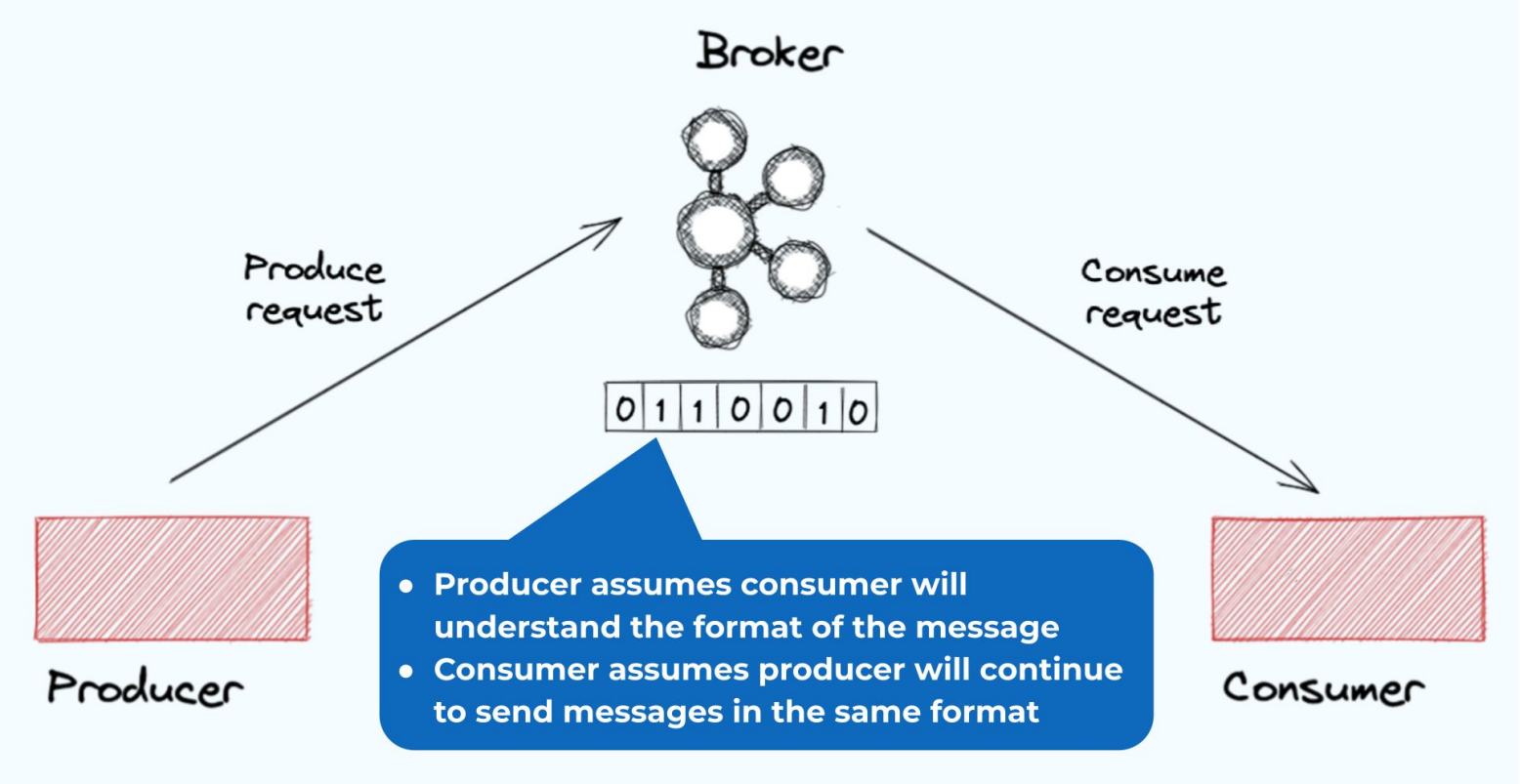
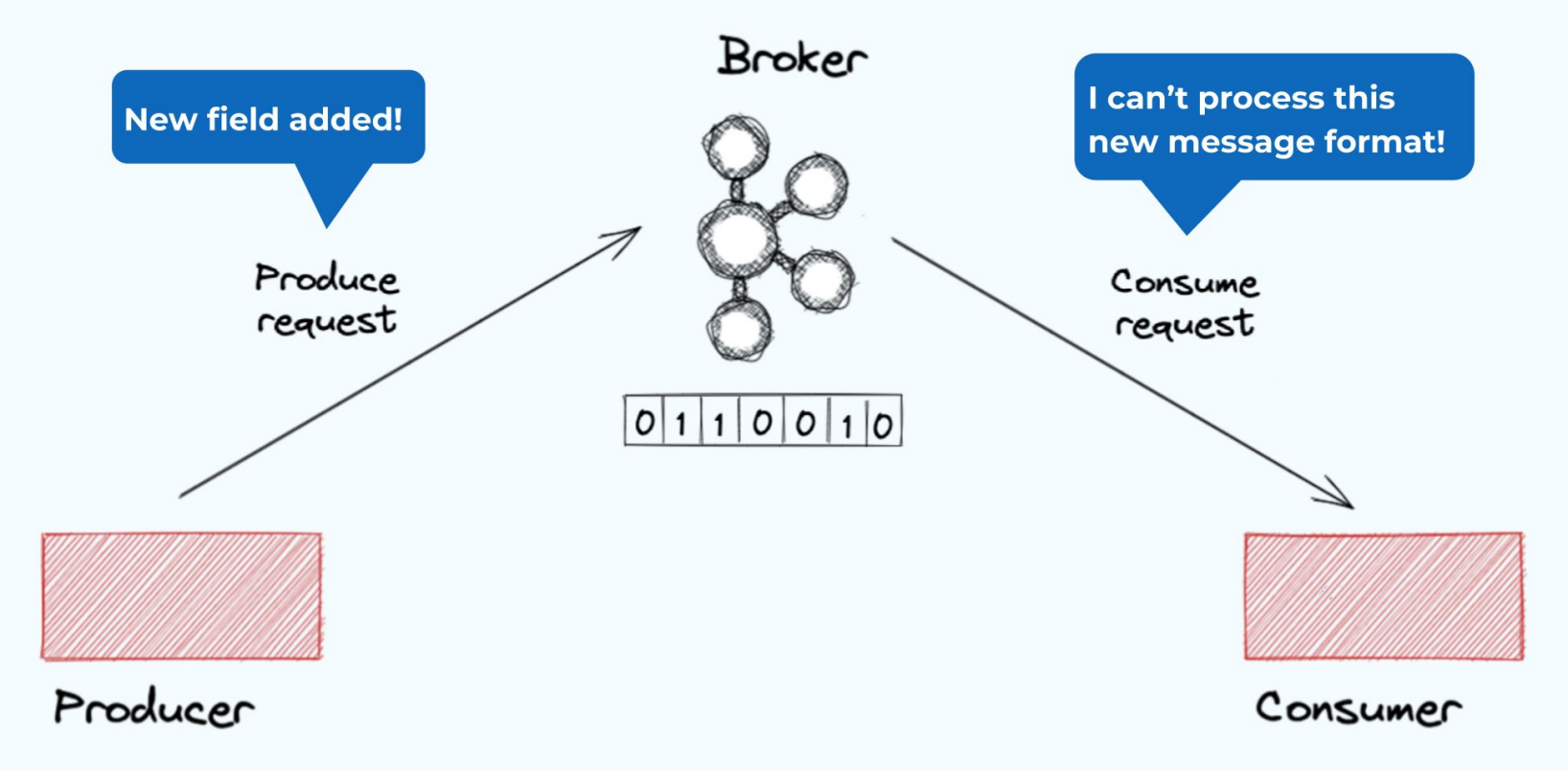
Schema Registry

Key Concepts of a Schema Registry

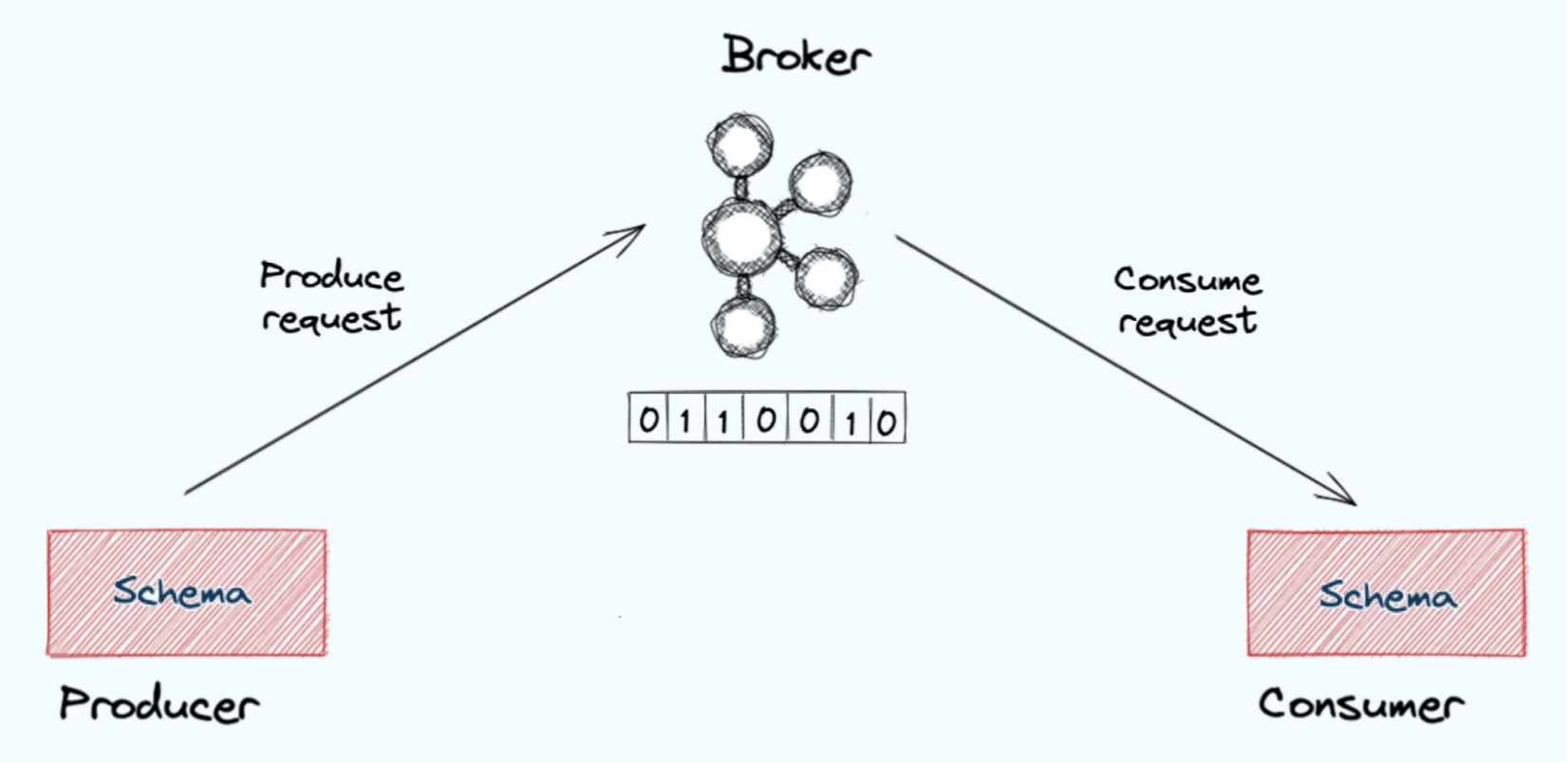
Implied Contract Between Kafka Applications



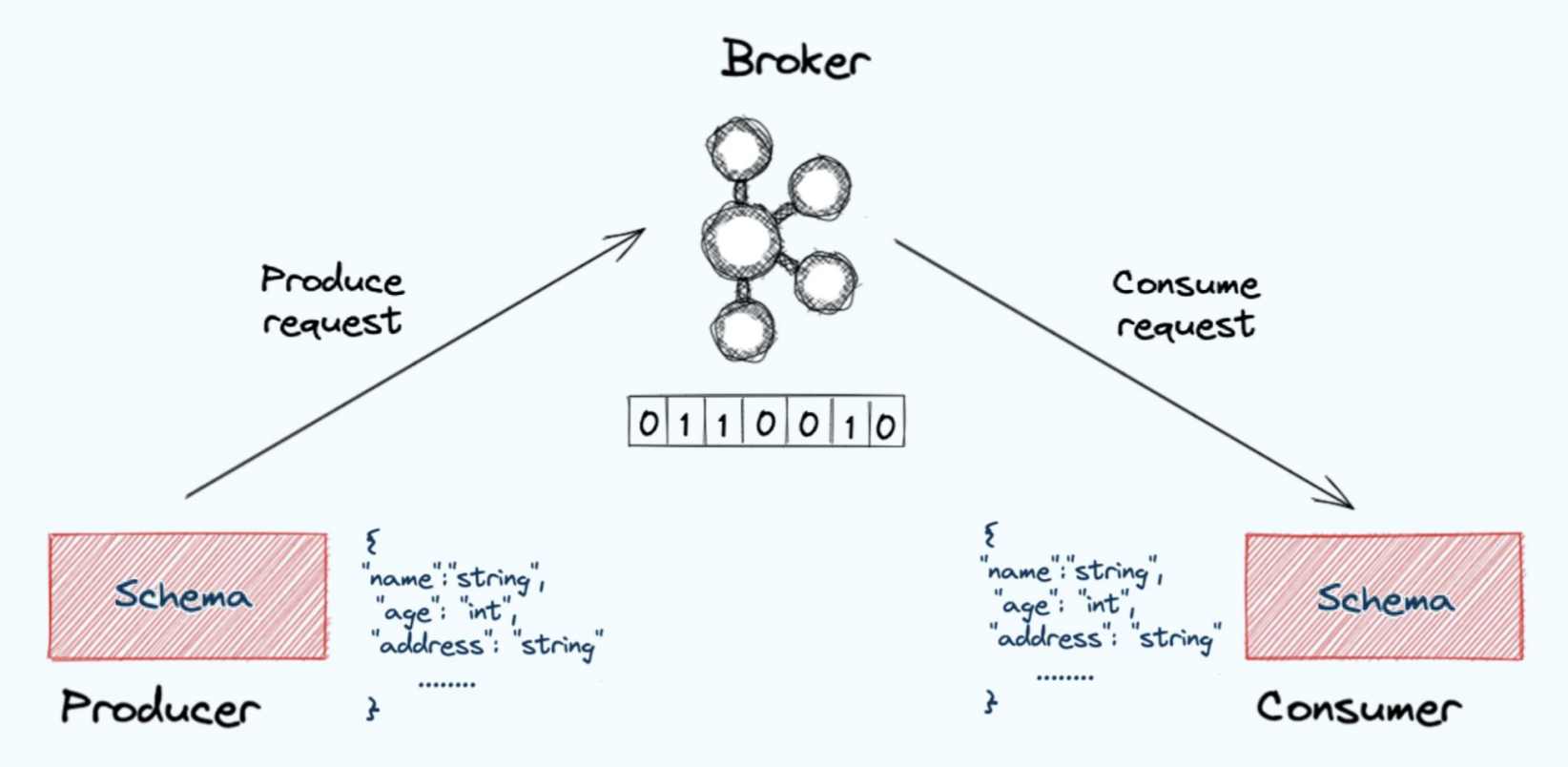
Change Always Happens



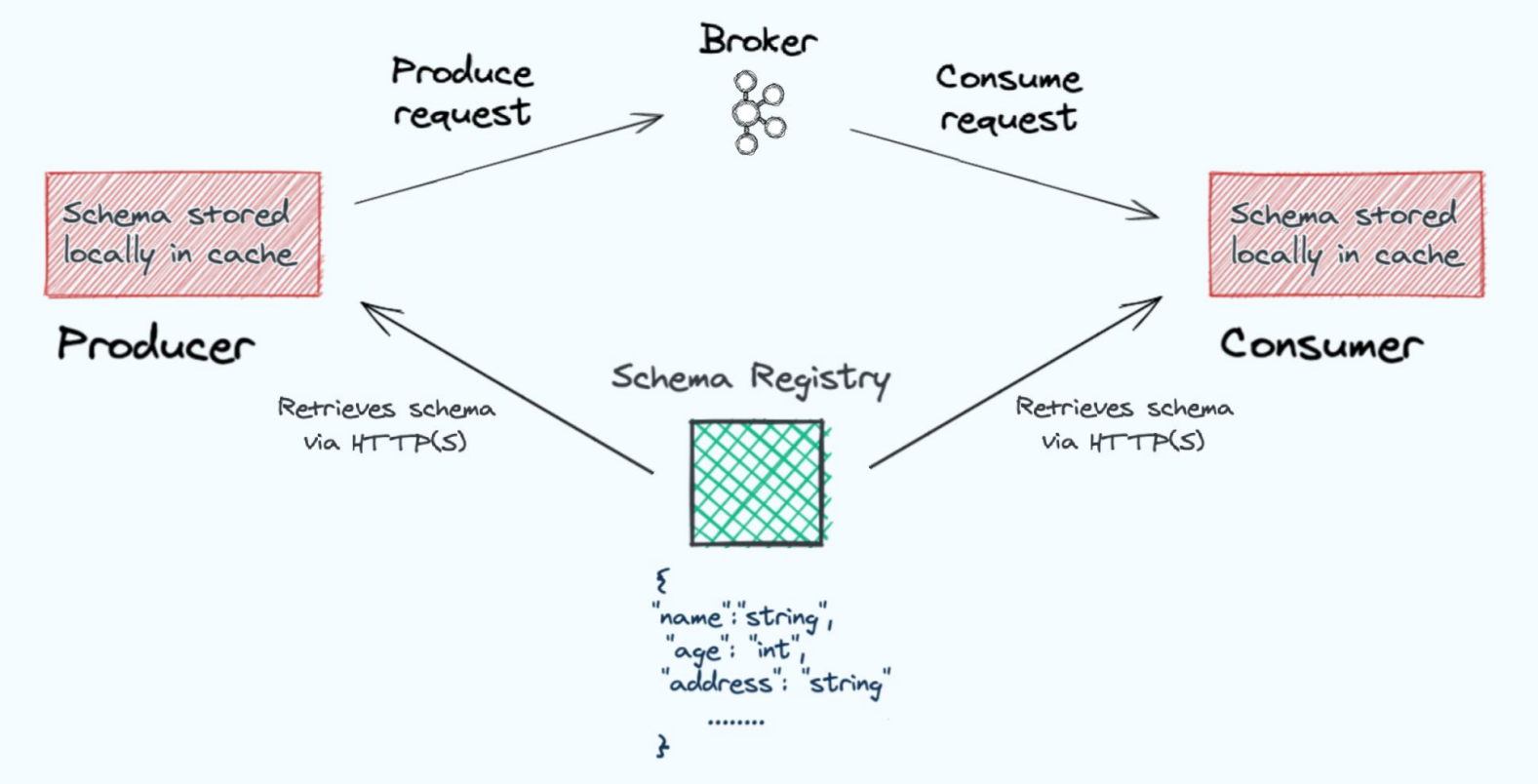
Schema is the Contract



Schema Establish the Format of the Message

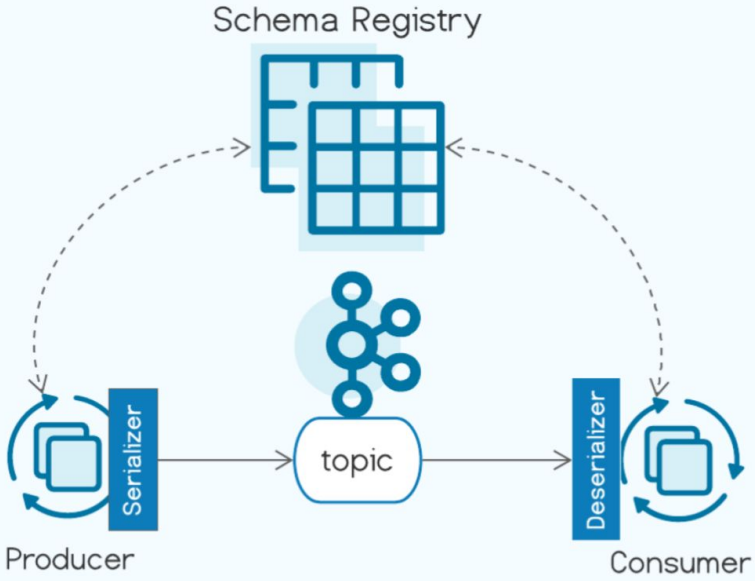


Schema Registry Is the Contract Arbitrator



Confluent Schema Registry

- **Versioned** schema repository
- **Safe** schema evolution
- **Resilient** data pipelines
- **Enhanced** data integrity
- **Reduce** storage and computation
- **Discover** your data
- **Cost-efficient** ecosystem



AVRO protobuf Protocol Buffers {JSON}

The Schema Workflow

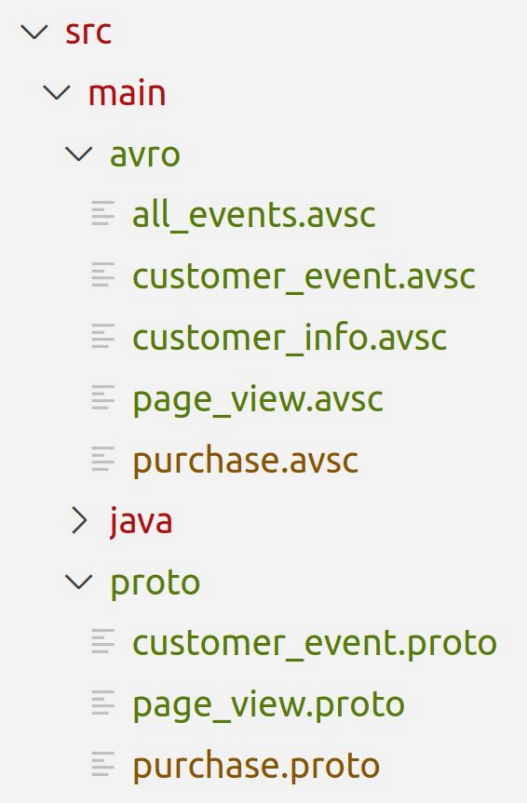
Writing a Schema File - Protobuf

```
syntax = "proto3";
package io.confluent.developer.proto;
option java_outer_classname = "PurchaseProto";

message Purchase {
  string item = 1;
  double total_cost = 2;
  string customer_id = 3;
}
```

Writing a Schema File - Avro

```
{
  "type":"record",
  "namespace": "io.confluent.developer.avro",
  "name":"Purchase",
  "fields": [
    {"name": "item", "type":"string"},
    {"name": "total_cost", "type": "double"},
    {"name": "customer_id", "type": "string"}
  ]
}
```

Place Schema Files in a Project



Plugin Configuration

```
plugins {
  ... other plugin definitions

  id "com.google.protobuf" version "0.8.15"
  id "com.github.imflog.kafka-schema-registry-gradle-plugin" version "1.1.1"
  id "com.github.davidmc24.gradle.plugin.avro" version "1.0.0"
}
```

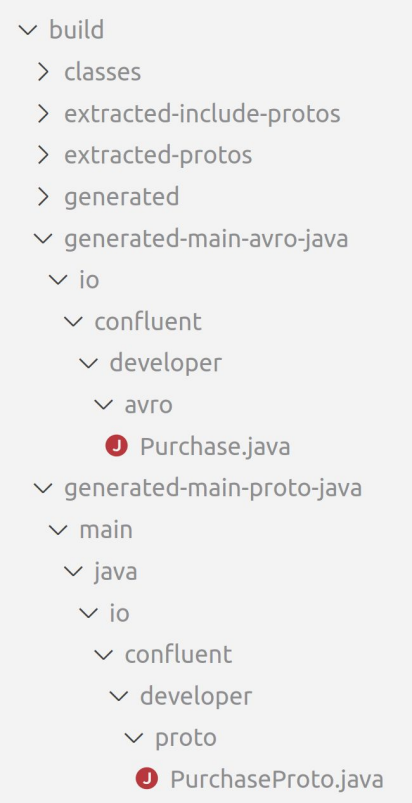
Protobuf Plugin Configuration

```
protobuf {

  generatedFilesBaseDir = "${project.buildDir}/generated-main-proto-java"

  protoc {
    artifact = 'com.google.protobuf:protoc:3.15.3'
  }
}
```

Schema Registry Configuration

```
schemaRegistry {
    url = Confluent Cloud SR endpoint
    credentials {
        //characters up to the ':' in the basic.auth.user.info property
        username = <username>
        // password is everything after ':' in the basic.auth.user.info property
        password = <password>
    }
    // Possible types are ["JSON", "PROTOBUF", "AVRO"]
    register {
        subject('avro-purchase', 'src/main/avro/purchase.avsc', 'AVRO')
        subject('proto-purchase', 'src/main/proto/purchase.proto', 'PROTOBUF')
    }
}
```

Generated Files in a Project



Working with Schema Formats

Protobuf

```
syntax = "proto3";
package io.confluent.developer.proto;
option java_outer_classname = "PurchaseProto";

message Purchase {
  string item = 1;
  double total_cost = 2;
  string customer_id = 3;
}
```

Protobuf Collections

```
message Purchase {
  string item = 1;
  double total_cost = 2;
  string customer_id = 3;
  repeated string coupon_codes = 4;
  map<string, string> item_options = 5;
}
```

Protobuf Enumerations

```
message Purchase {
  string item = 1;
  double total_cost = 2;
  string customer_id = 3;
  repeated string coupon_codes = 4;
  map<string, double> item_options = 5;

  enum PurchaseType {
    RETAIL = 0;
    WHOLESALE = 1;
  }
  PurchaseType type = 6;
}
```

Importing Protobuf

```
import "purchase.proto";
import "page_view.proto";

message CustomerEvent {

  Purchase purchase = 1;
  PageView page_view = 2;
}
```

Alternate Values for a Field

```
import "purchase.proto";
import "page_view.proto";

message CustomerEvent {

  oneof customer_action {
    Purchase purchase = 1;
    PageView page_view = 2;
  }
}
```

Default Values

```
message Purchase {
  string item = 1;  -> default is ""
  double total_cost = 2; -> default is 0.0
  string customer_id = 3;
  repeated string coupon_codes = 4; -> default is []
  map<string, string> item_options = 5; -> default is {}
}
```

Avro

```
{
  "type":"record",
  "namespace": "io.confluent.developer.avro",
  "name":"Purchase",
  "fields": [
    {"name": "item", "type":"string"},
    {"name": "total_cost", "type": "double"},
    {"name": "customer_id", "type": "string"}
  ]
}
```

Avro Collections - Arrays

```
{
  "type":"record",
  "namespace": "io.confluent.developer.avro",
  "name":"Purchase",
  "fields": [
    {"name": "item", "type":"string"},
    {"name": "total_cost", "type": "double"},
    {"name": "customer_id", "type": "string"},
    {"name": "coupon_codes",
      "type": {
        "type": "array",
        "items": "string"
      }
    }
  ]
}
```

Avro Collections - Map

```
{
  "type":"record",
  "namespace": "io.confluent.developer.avro",
  "name":"Purchase",
  "fields": [
    {"name": "item", "type":"string"},
    {"name": "total_cost", "type": "double"},
    {"name": "customer_id", "type": "string"},
    {"name": "item_options",
      "type": {
        "type": "map",
        "values": "double"
      }
    }
  ]
}
```

Avro Enumerations

```
{
  "type":"record",
  "namespace": "io.confluent.developer.avro",
  "name":"Purchase",
  "fields": [
    {"name": "item", "type":"string"},
    {"name": "total_cost", "type": "double"},
    {"name": "customer_id", "type": "string"},
    {"name": "category",
      "type": {
        "type": "enum",
        "name": "purchase_type",
        "symbols": ["RETAIL", "WHOLESALE"]
      }
    }
  ]
}
```

Avro Records in a Schema

```
{
  "type": "record",
  "namespace": "io.confluent.developer.avro",
  "name": "CustomerEvent",
  "fields" : [
    {"name": "purchase", "type": "io.confluent.developer.avro.Purchase" },
    {"name": "page_view", "type": "io.confluent.developer.avro.PageView" },
    {"name": "id", "type": "string"}
  ]
}
```

Avro Unions

```
{
  "type": "record",
  "namespace": "io.confluent.developer.avro",
  "name": "CustomerEvent",
  "fields" : [
    {"name": "action", "type": [
      "io.confluent.developer.avro.Purchase",
      "io.confluent.developer.avro.PageView"
    ]},
    {"name": "id", "type": "string"}
  ]
}
```

Avro Default Values

```
"fields": [
  {"name": "item", "type":"string"},
  {"name": "total_cost", "type": "double", default = 0.0},
  {"name": "customer_id", "type":"string"},
  {"name": "item_options",
    "type": {
      "type": "map",
      "values": "string",
      "default": {}
    }
  }
]
```

Working with Generated Objects

```
Purchase.newBuilder()
  .setCustomerId("vandelay1234")
  .setTotalCost(437.83)
  .setItem("flux-capacitor")
  .build();

purchase.getTotalCost()
```

Changing State of Generated Objects

Avro only

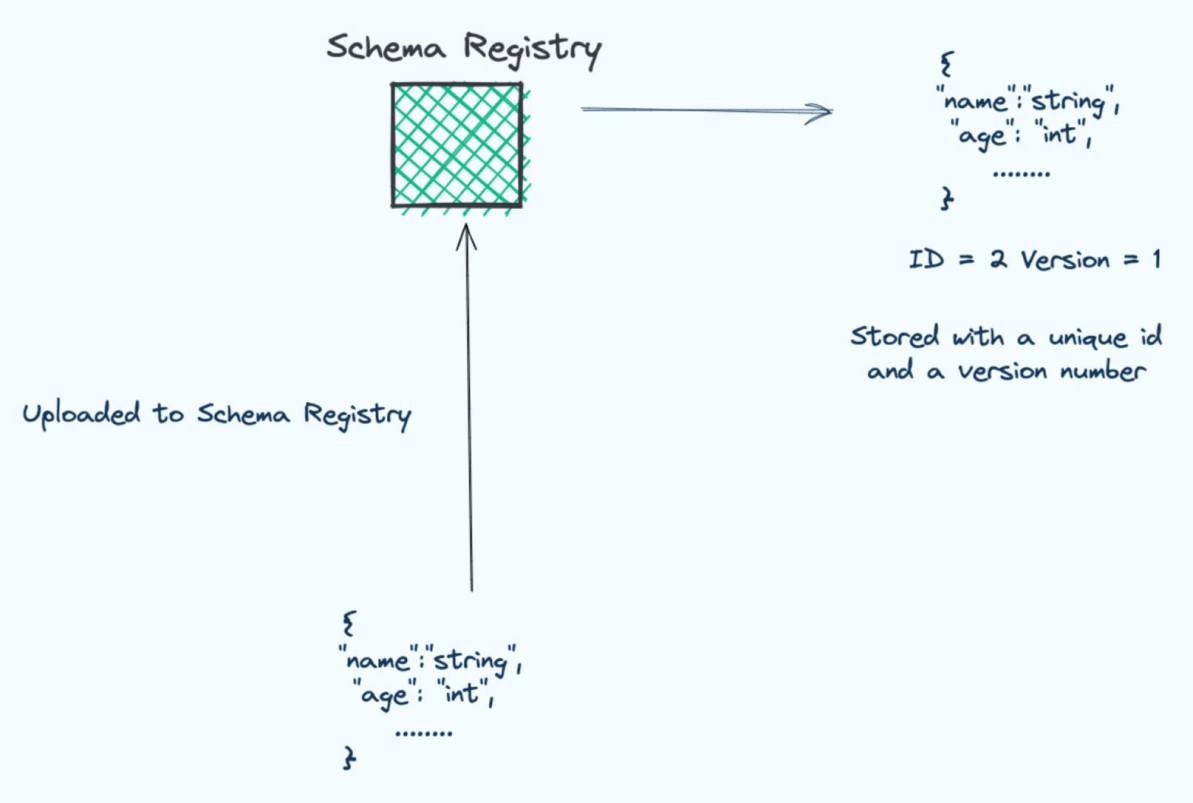
```
purchase.setTotalCost(100.00)
```

Protobuf and Avro

```
Purchase.newBuilder(purchase)
        .setTotalCost(100.00)
        .build()
```

Schema Management

Schema ID and Version



Register Schema - Confluent CLI

```
confluent schema-registry schema create \
  --subject purchases-value \
  --schema src/main/proto/purchase.proto \
  --type PROTOBUF \
  --api-key YYY \
  --api-secret ZZZ
```

Register an Avro Schema - REST API

```
jq '. | {schema: tojson}' src/main/avro/purchase.avsc | \
curl -u API_KEY:API_SECRET \
  -X POST https://CLUSTER_IP/subjects/purchases-value/versions \
  -H "Content-Type:application/json" \
  -d @-
```

Register a Protobuf Schema - REST API

Protobuf schema definition passed as input to script

```
./protoJsonFmt.sh src/main/proto/purchase.proto | \
curl -u API_KEY:API_SECRET \
  -X POST https://CLUSTER_IP/subjects/purchases-value/versions \
  -H "Content-Type:application/json" \
  -d @-

PROTO_JSON=$(awk '{gsub(/\n/,"\\\n"; gsub(/"/, "\\\"");print}' $1) \
  && SCHEMA="{\"schemaType\":\"PROTOBUF\",\"schema\":\"${PROTO_JSON}\"\n}" \
  && echo ${SCHEMA}
```

JSON formatted schema definition outputted from script used as input by `curl` command

Register a Schema - Gradle

```
register {
  subject('purchases-value', 'src/main/avro/purchase.avsc', 'AVRO')
  subject('purchases-value', 'src/main/proto/purchase.proto', 'PROTOBUF')
}
```

```
./gradlew registerSchemasTask
```

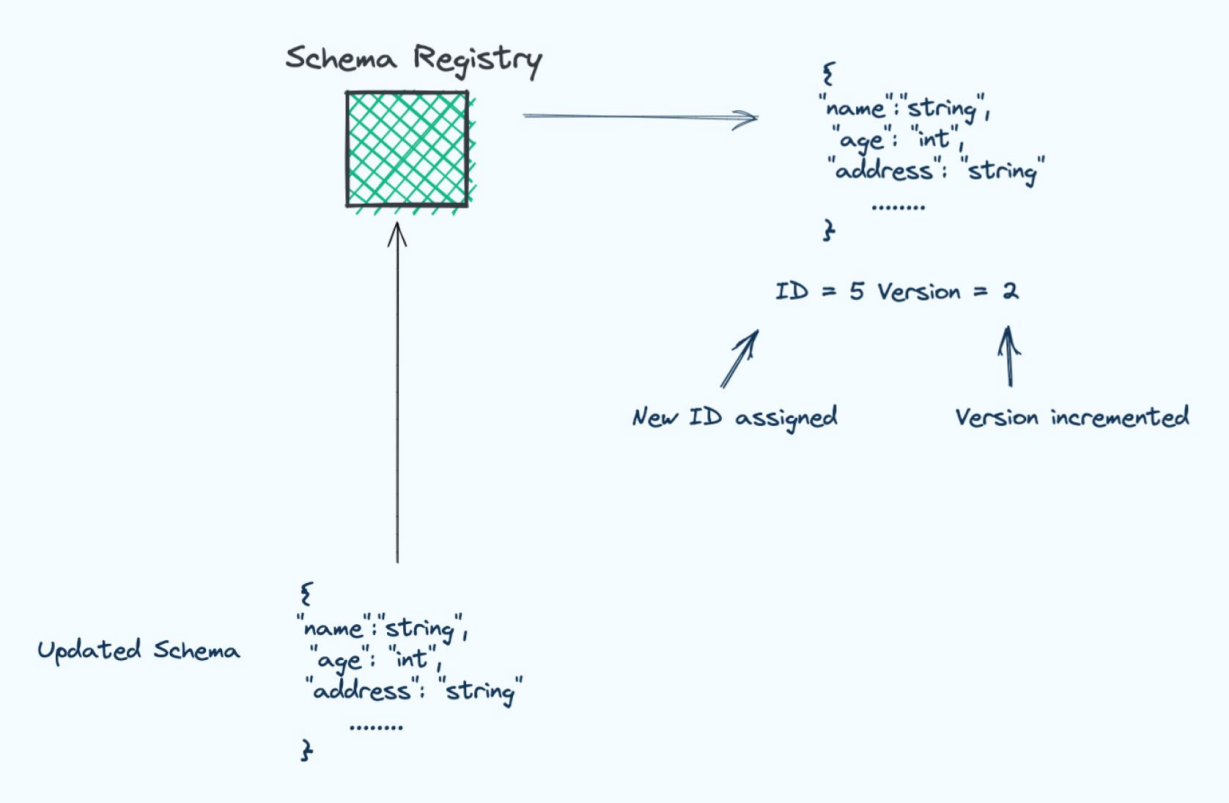
Auto-Register a Schema

Producer clients can “auto-register”

```
producerConfigs.put(“auto.register.schemas”, true)
```

Not for production!!!

Updating a Schema



View or Download a Schema

```
confluent schema-registry schema describe \
  --subject purchases-value \
  --version [version number] | "latest" \
  --api-key YYY \
  --api-secret ZZZ

curl -u API_KEY:API_SECRET \
  -s "https://CLUSTER_IP/subjects/purchases-value/versions/latest" jq '.'

curl -u API_KEY:API_SECRET \
  -s "https://CLUSTER_IP/subjects/purchases-value/versions/N" jq '.'
```

Integrating Schema Registry with Client Applications

Console Producer

```
./bin/kafka-protobuf-console-producer \
   --topic purchases \
   --bootstrap-server localhost:9092 \
   --property schema.registry.url = http://... \
   --property value.schema ="$(<src/main/proto/purchase.proto)"
> {"item":"pizza", "total_cost":17.99, "customer_id":”lombardi"}
```

Console Consumer

```
./bin/kafka-protobuf-console-consumer \
    --from-beginning \
    --topic purchases \
    --bootstrap-server localhost:9092 \
    --property schema.registry.url = http://...
```

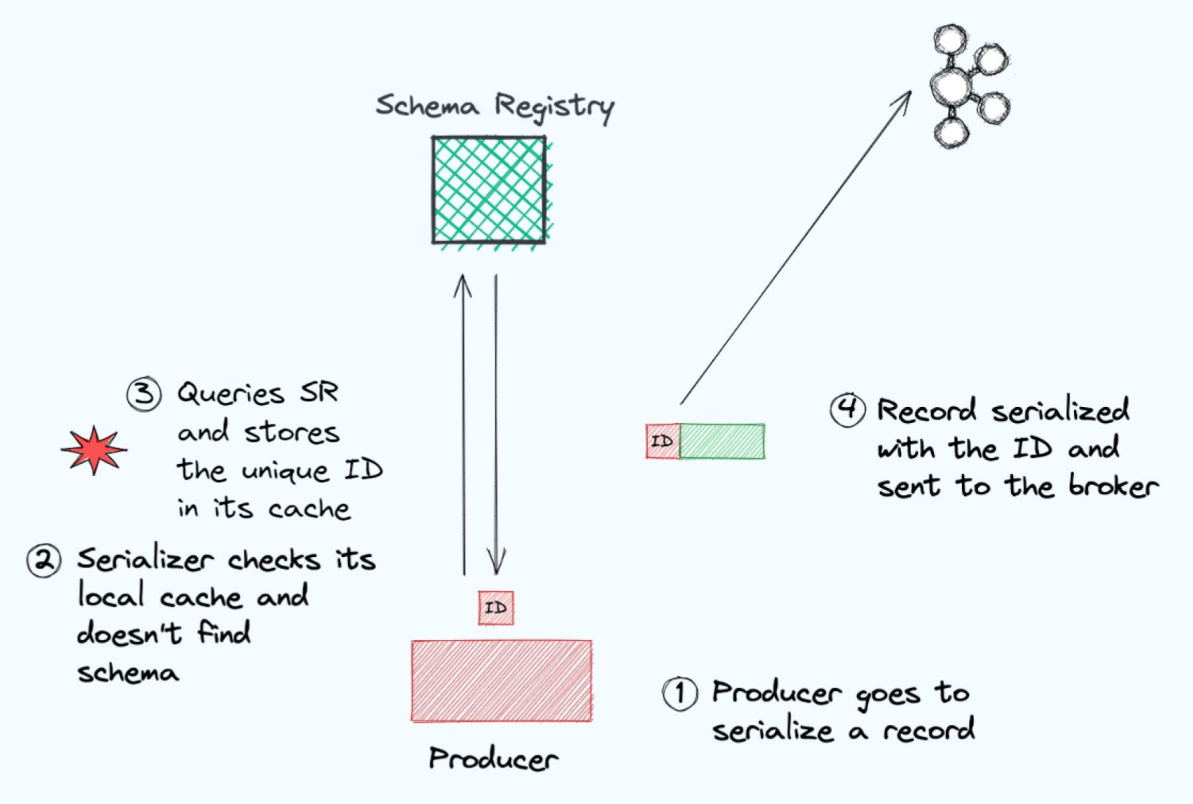
Common Client Configuration Settings

```
schema.registry.url=https://<CLUSTER>.us-east-2.aws.confluent.cloud

basic.auth.credentials.source=USER_INFO
basic.auth.user.info=API_KEY:API_SECRET
```

Kafka Producer

```
// Possible serializer classes are:
//   KafkaAvroSerializer.class
//   KafkaProtobufSerializer.class
//   KafkaJsonSchemaSerializer.class

configs.put("value.serializer", <SERIALIZER_CLASS>);

Producer<Purchase> producer = new KafkaProducer<>(configs);
```

Schema Lifecycle



Kafka Consumer

```
// Possible deserializer classes are:
//   KafkaAvroDeserializer.class
//   KafkaProtobufDeserializer.class
//   KafkaJsonSchemaDeserializer.class

configs.put("value.deserializer", <DESERIALIZER_CLASS>);

Consumer<Purchase> consumer = new KafkaConsumer<>(configs);
```

Kafka Consumer

```
specific.avro.reader = true|false

specific.protobuf.value.type = proto class name

json.value.type = class name
```

Kafka Consumer

Avro

```
SpecificRecord - myObj.getFoo()
GenericRecord -  generic.get("foo")
```

Protobuf

```
Specific
Dynamic
```

JSON Schema

```
JsonNode
```

ksqlDB

```
CREATE STREAM add_item_to_cart (
  cart_id BIGINT,
  item_id BIGINT
 WITH (
   KAFKA_TOPIC = 'add_item_to_cart',
   VALUE_FORMAT = 'AVRO',
   PARTITIONS = 6
  )
);
```

Understanding Schema Subjects

Schema Subject - What is It?

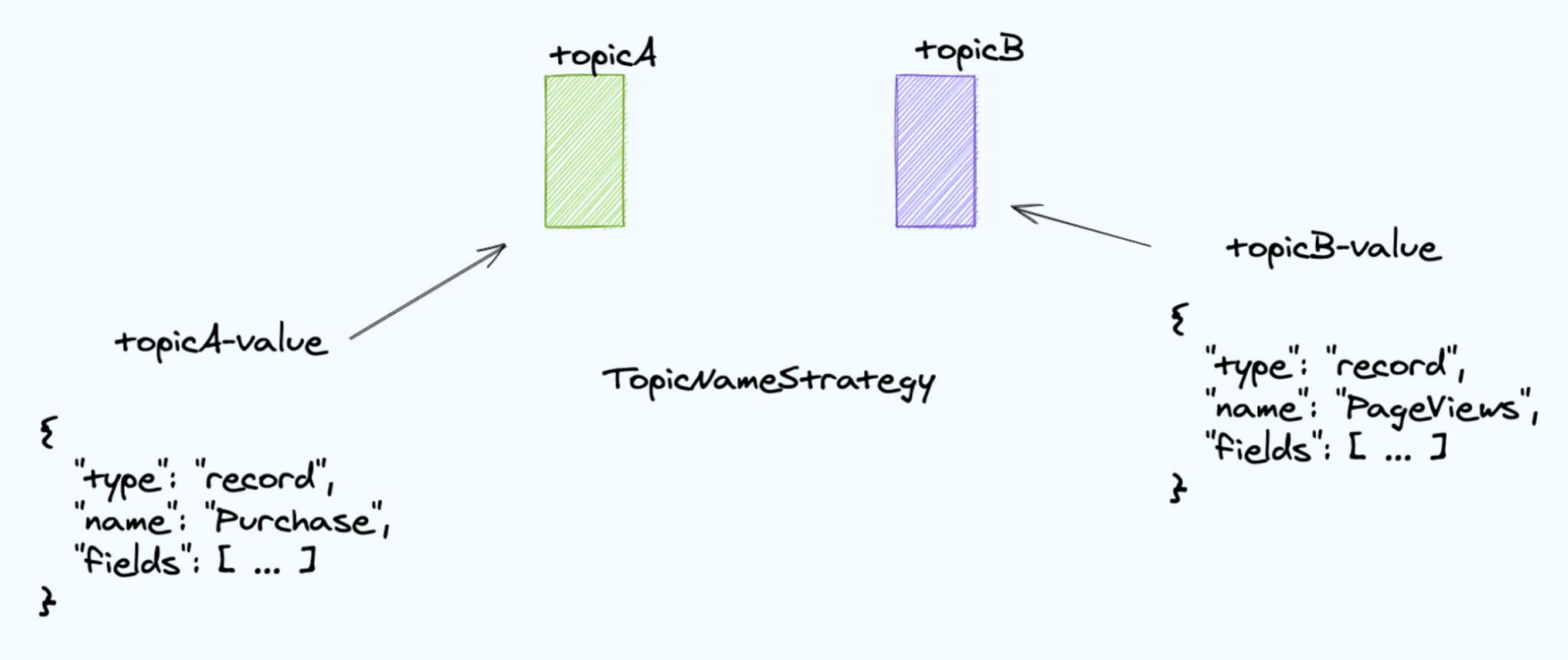
```
confluent schema-registry schema create --subject purchases-value \...

curl -u API_KEY:API_SECRET \
     -X POST https://CLUSTER_IP/subjects/purchases-value/versions \....

register {
       subject('purchases-value', 'src/main/avro/purchase.proto', 'PROTOBUF')
```

Subject Name Strategies

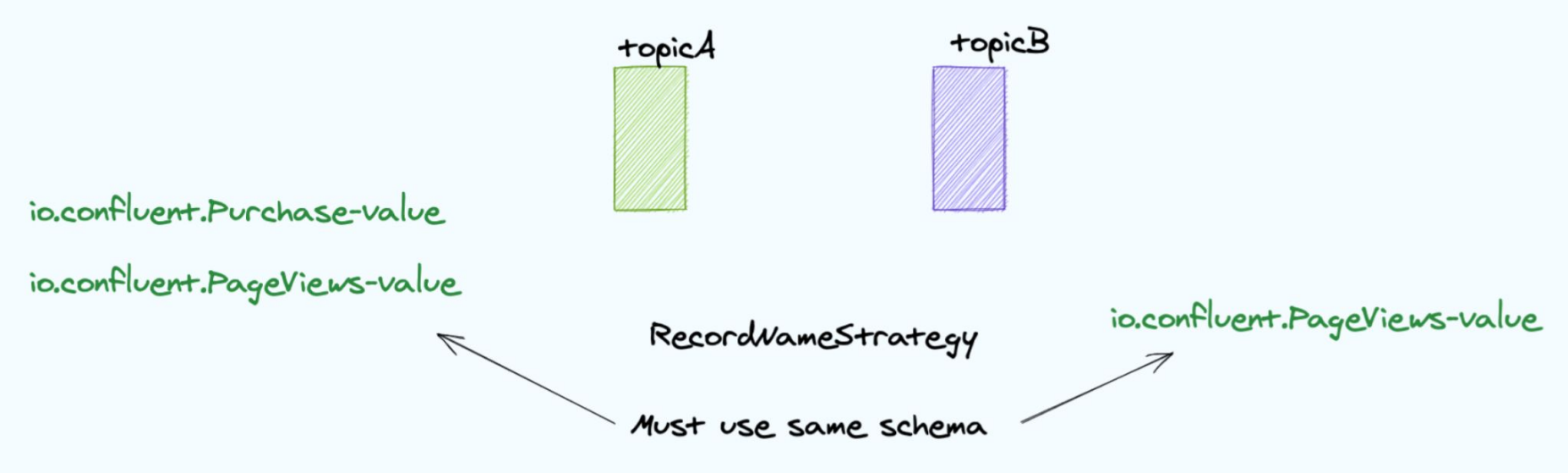
When providing a subject name, there are three strategies or ways to provide it:

- TopicNameStrategy (default setting)
- RecordNameStrategy
- TopicRecordNameStrategy

TopicNameStrategy



RecordNameStrategy



TopicRecordNameStrategy

topicA-io.confluent.Purchase-value

topicA

topicB

topicB-io.confluent.PageViews-value

version 2

TopicRecordNameStrategy

topicA-io.confluent.PageViews-value

version 1

Schemas scoped to the topic level, so now they can evolve individually

Using Different Subject Name Strategies

Choosing and using a particular strategy involves two steps:

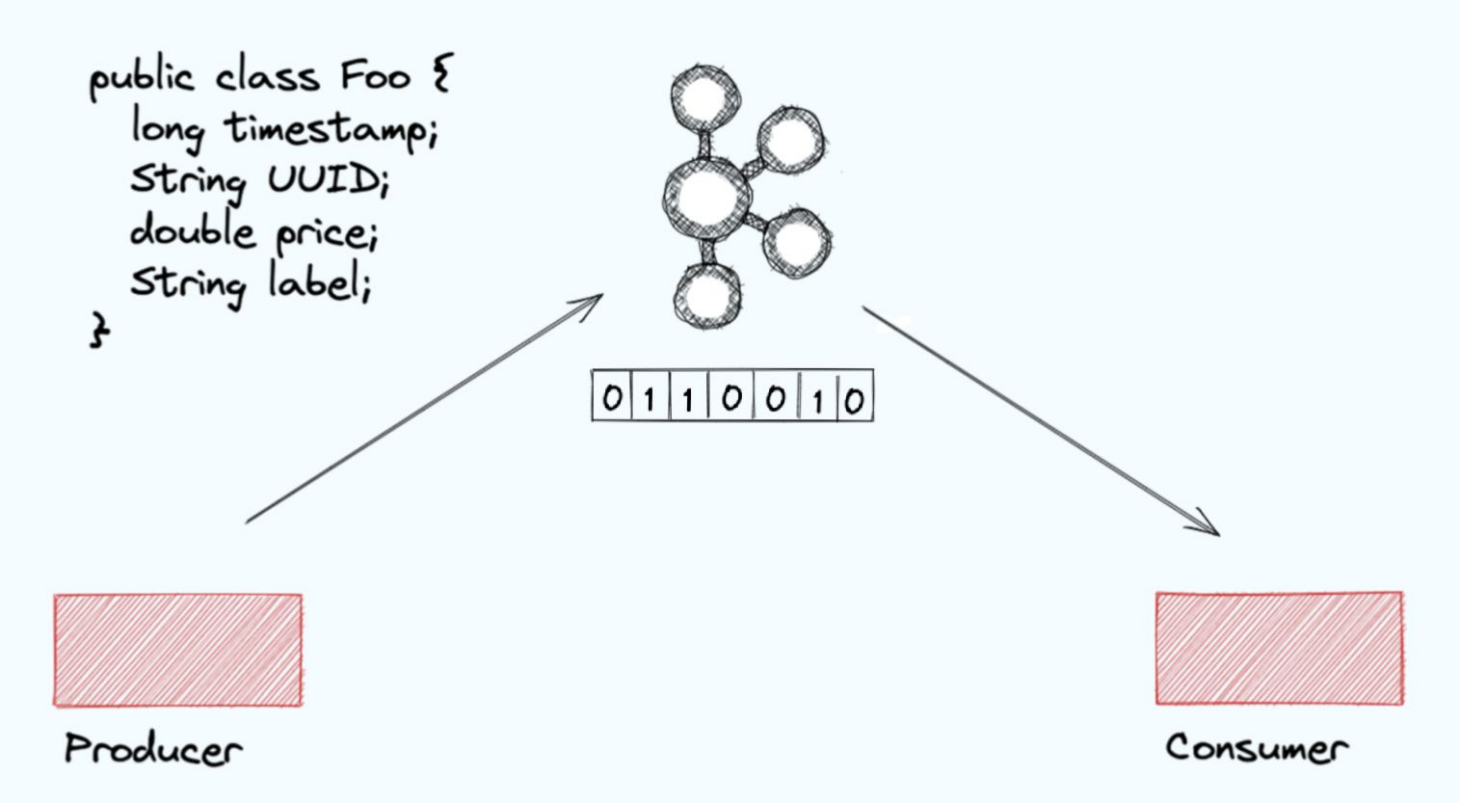
1. Registering the schema with the correct subject name format.
2. Setting the subject strategy on the clients.

Since TopicNameStrategy is the default, clients are already set to use it. To use a strategy other than the default, set key.subject.name.strategy or value.subject.name.strategy on the client as needed.

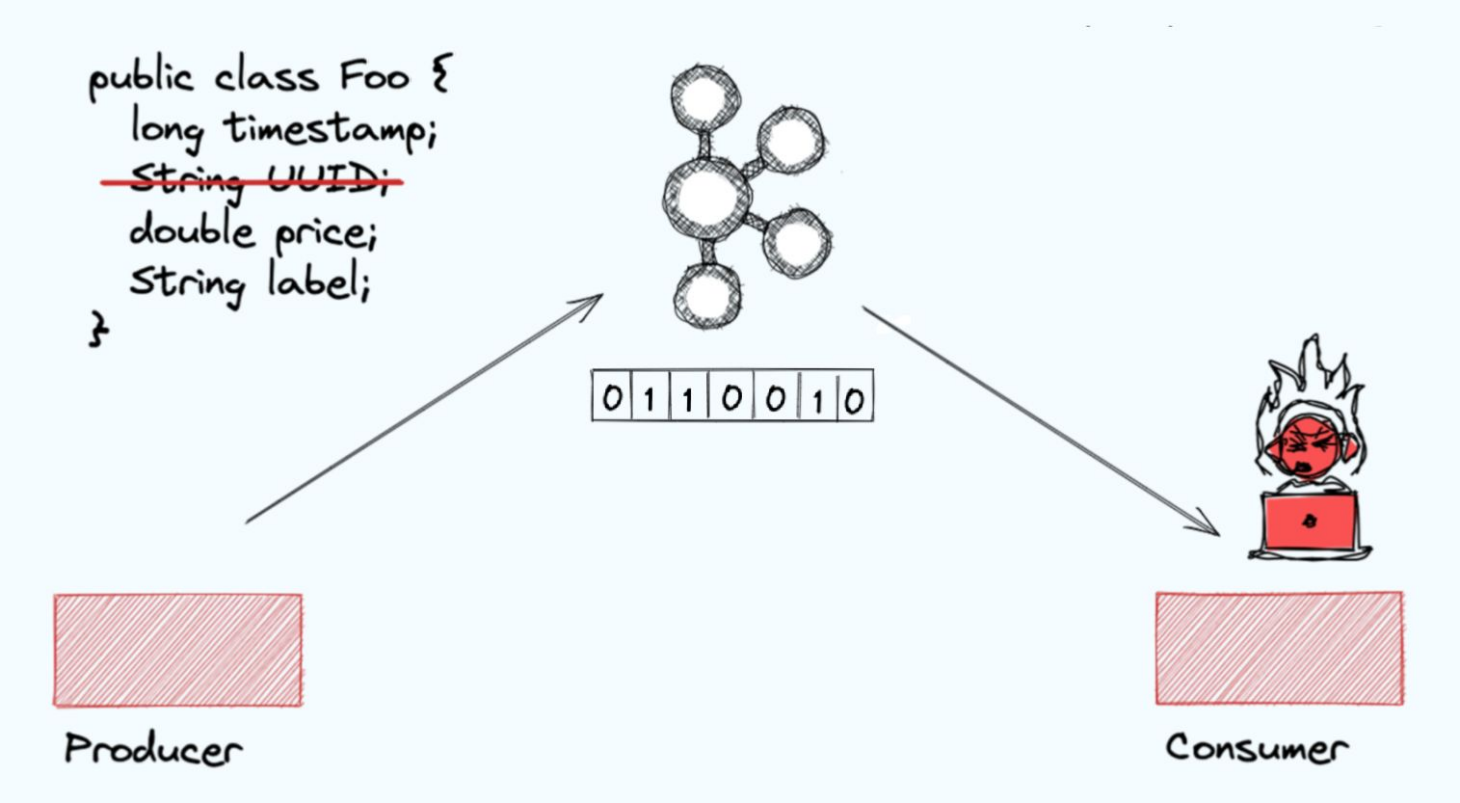
- Record Name Strategy - io.confluent.kafka.serializers.subject.RecordNameStrategy
- Topic Record Name Strategy - io.confluent.kafka.serializers.subject.TopicRecordNameStrategy

Schema Compatibility

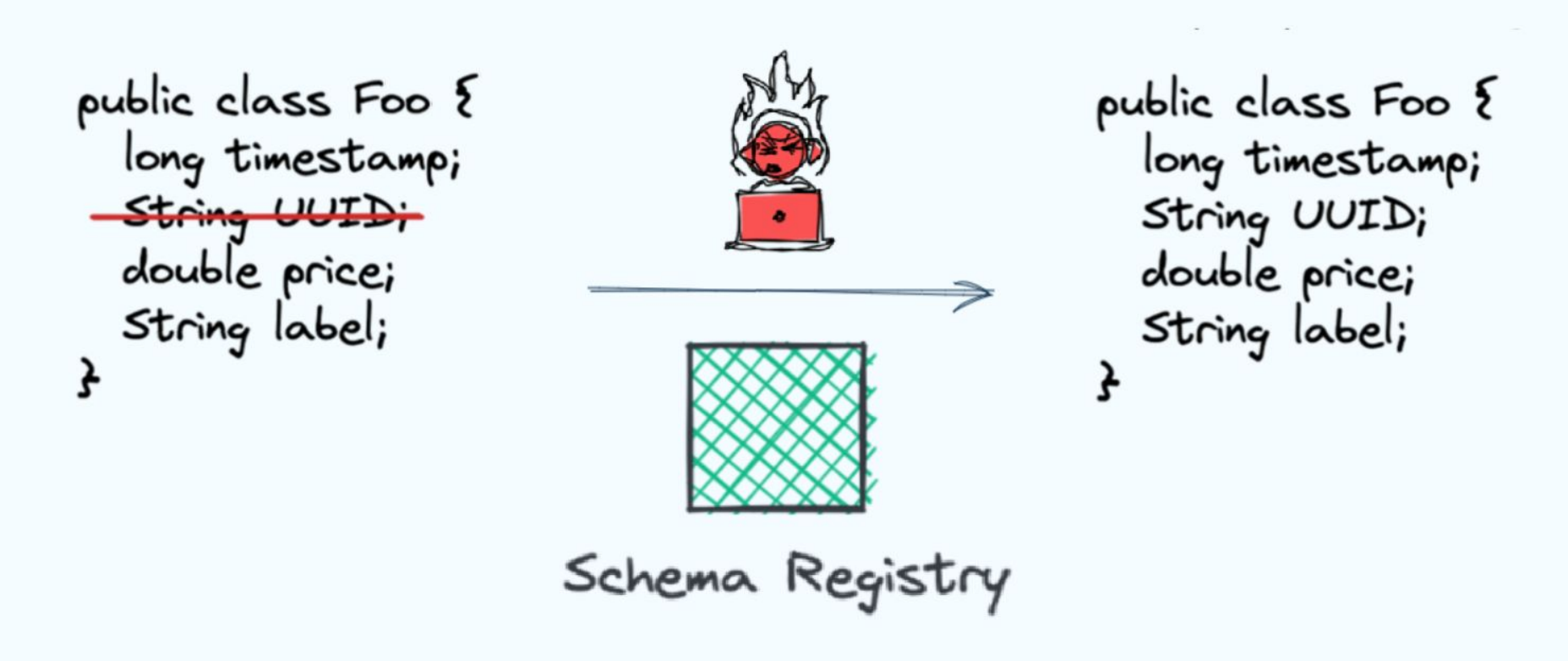
Expectations of Object Structure



Unexpected Changes May Cause Exceptions



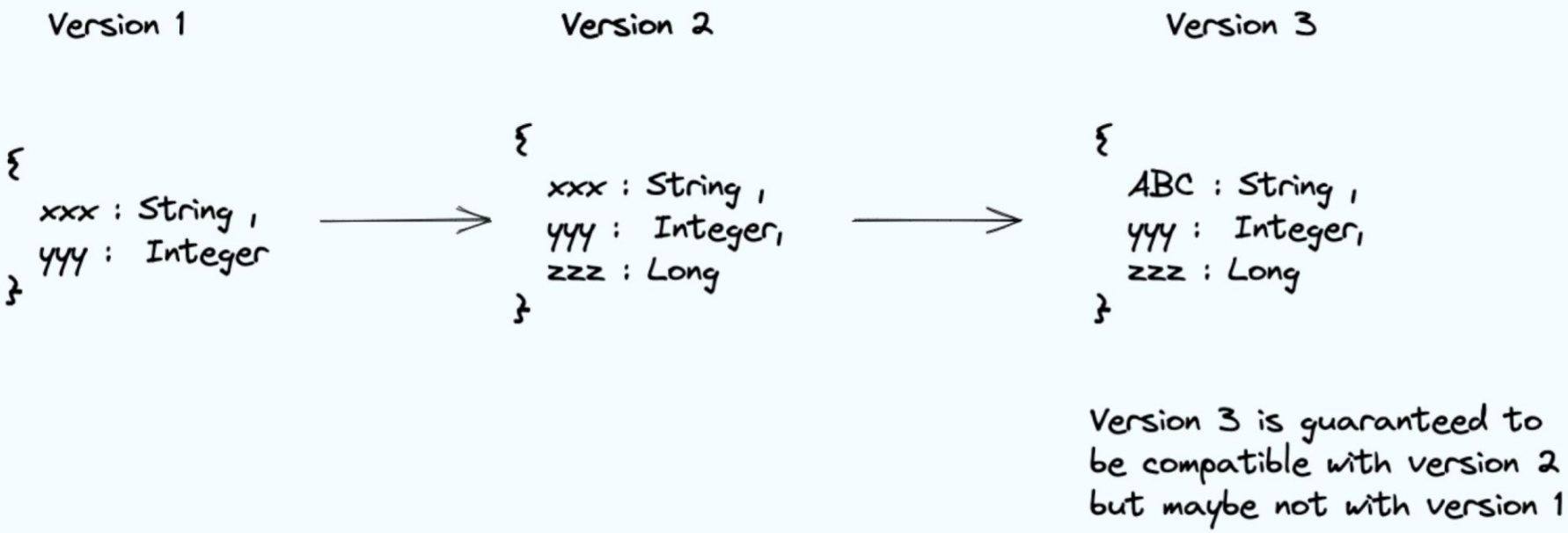
Schema Compatibility



Schema Compatibility Modes

Compatibility Type	Schema Resolution
BACKWARD	Consumers using the new schema version can read data produced with the previous version
FORWARD	Consumers using the previous schema version can read data produced with the new version
FULL	The combination of BACKWARD and FORWARD compatibility
NONE	Compatibility checking disabled

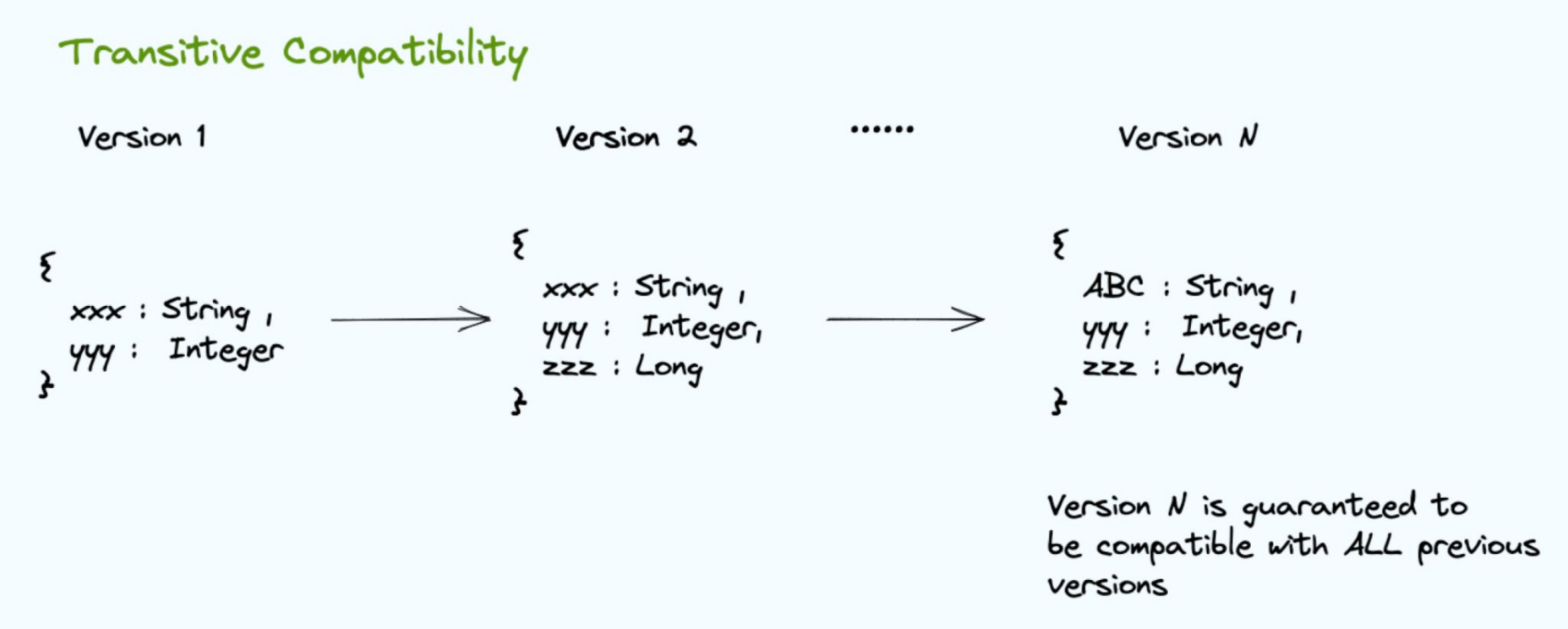
Schema Compatibility Example



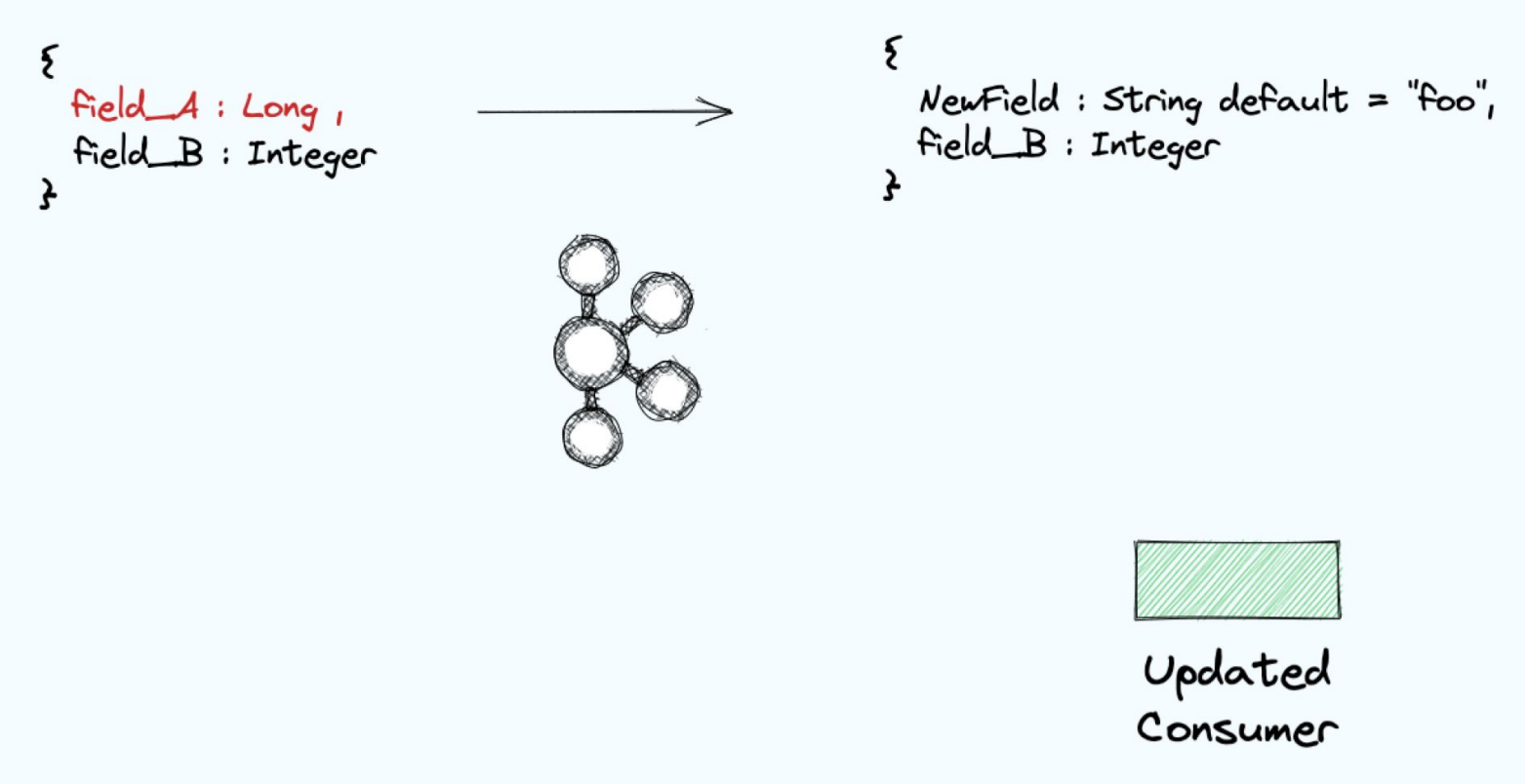
Transitive Schema Compatibility Modes

Compatibility Type	Schema Resolution
BACKWARD_TRANSITIVE	Consumers using the new schema version can read data produced with all previous versions
FORWARD_TRANSITIVE	Consumers using the previous schema version can read data produced with all later versions
FULL_TRANSITIVE	The combination of BACKWARD and FORWARD compatibility

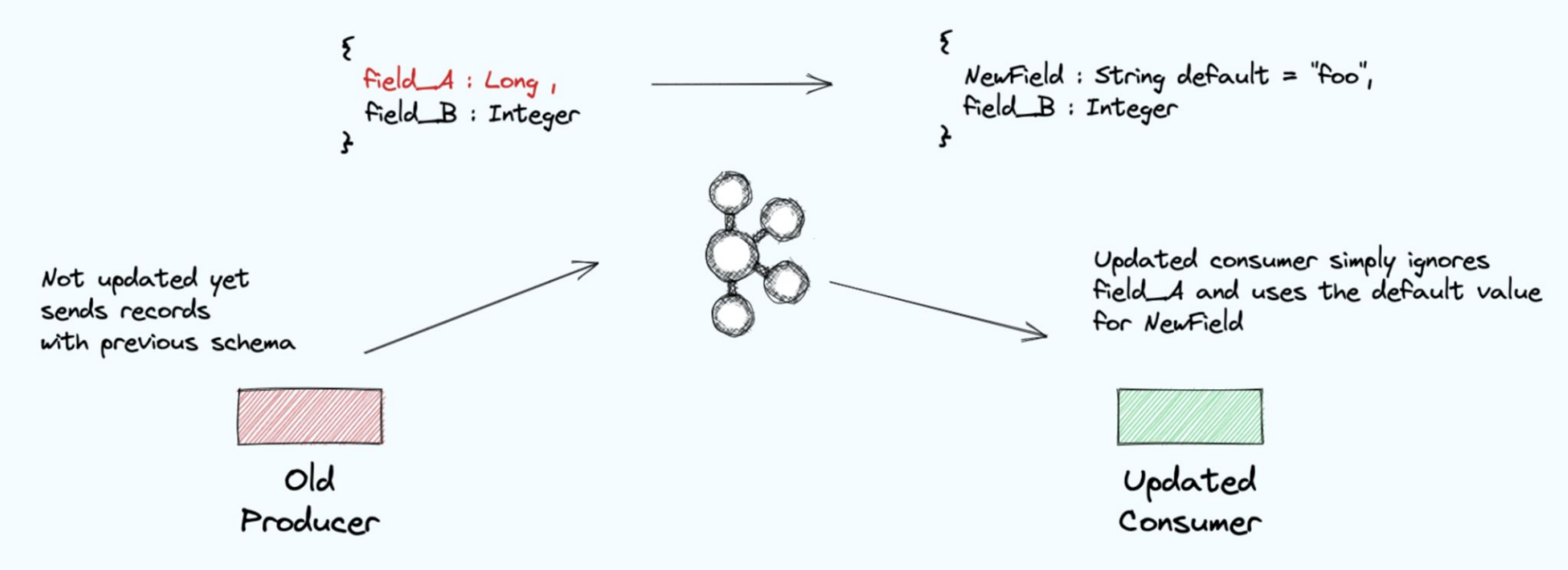
Transitive Schema Compatibility Example



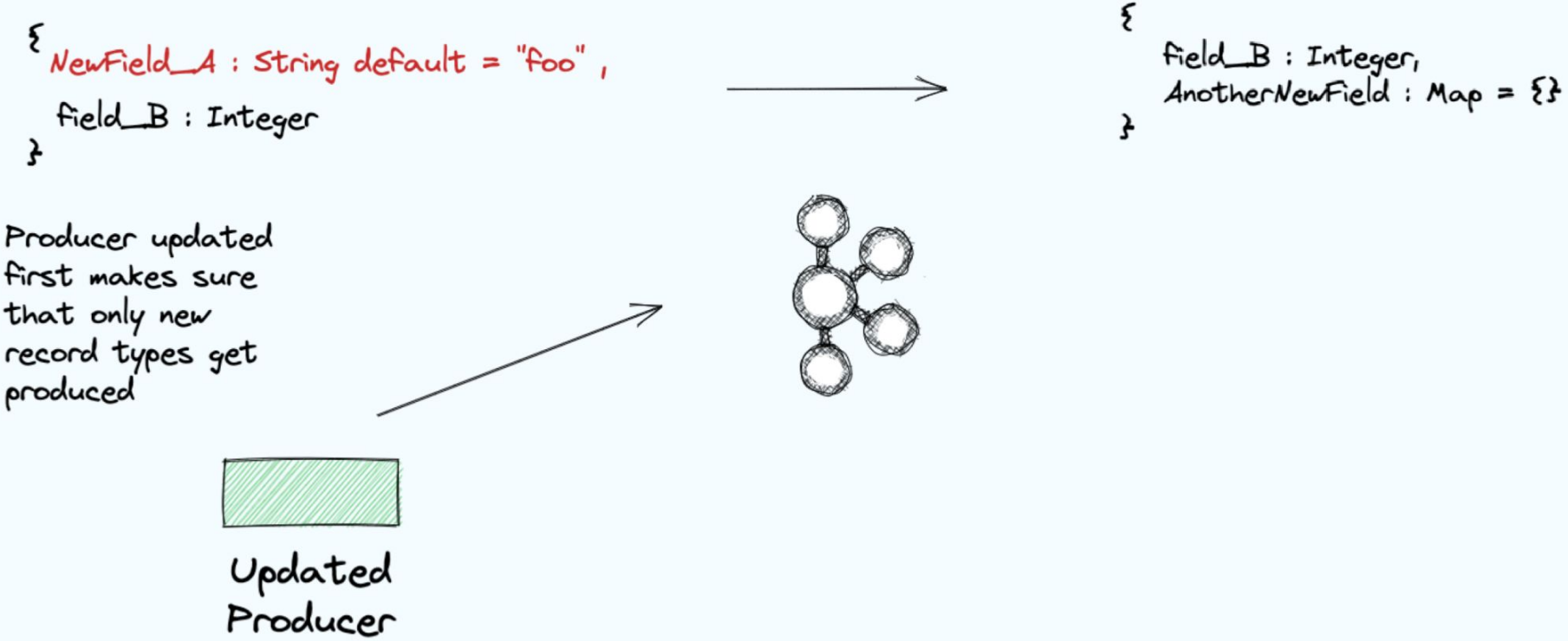
Backward Compatibility Illustrated



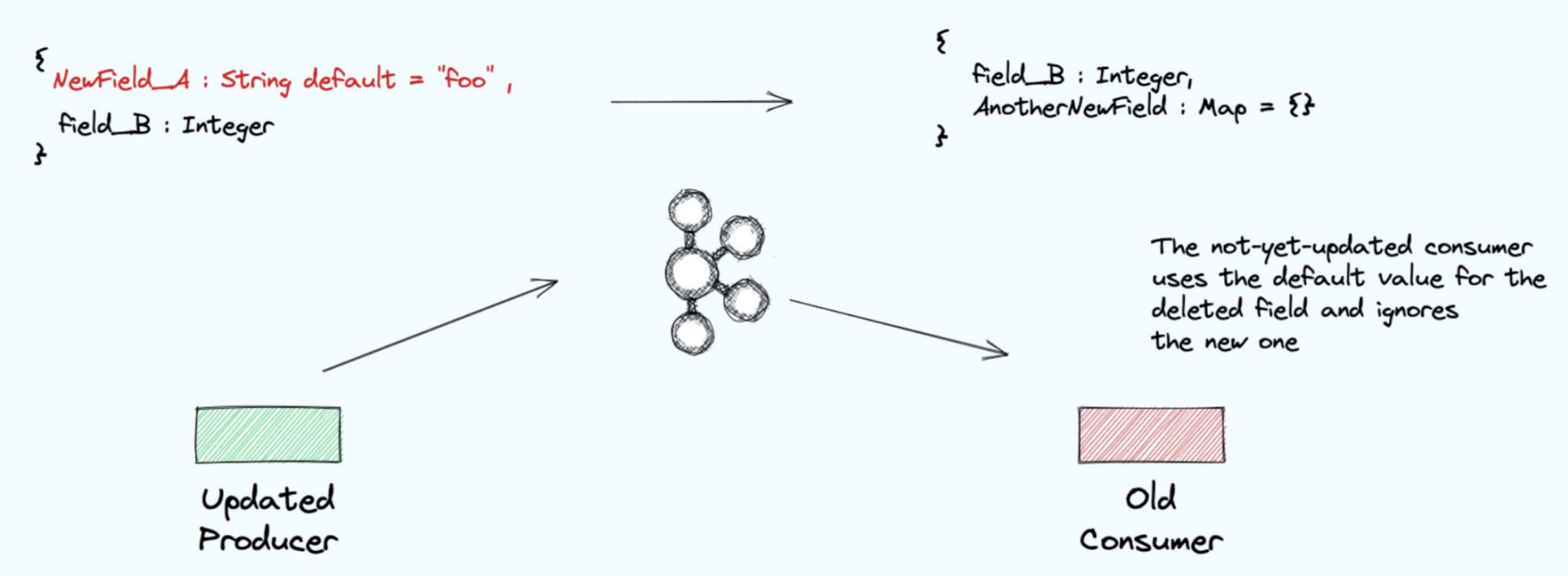
Backward Compatibility Illustrated



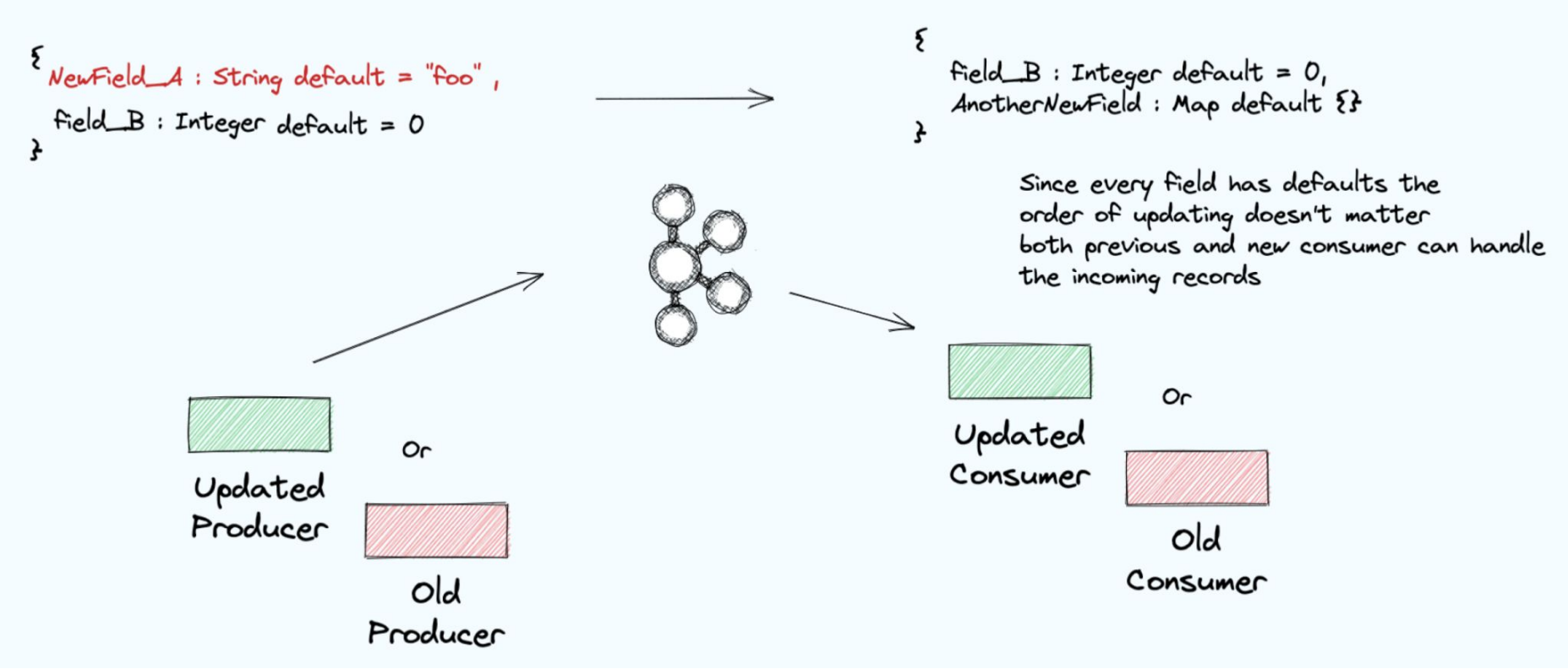
Forward Compatibility Illustrated



Forward Compatibility Illustrated



Full Compatibility Illustrated



Schema Compatibility Consideration

Backward	Forward	Full
Delete fields Add fields with default values	Delete fields with default values Add fields	Delete fields with default values Add fields with default values
Update consumer clients first	Update producer clients first	Order of update doesn’t matter

Checking a Schema for Compatibility

```
jq '. | {schema: tojson}' src/main/avro/purchase.avsc | \
curl -u API_KEY:API_SECRET \
     -X POST https://CLUSTER_IP/compatibility/subjects/purchases-value/version/latest \
     -H "Content-Type:application/json" \
     -d @-
```