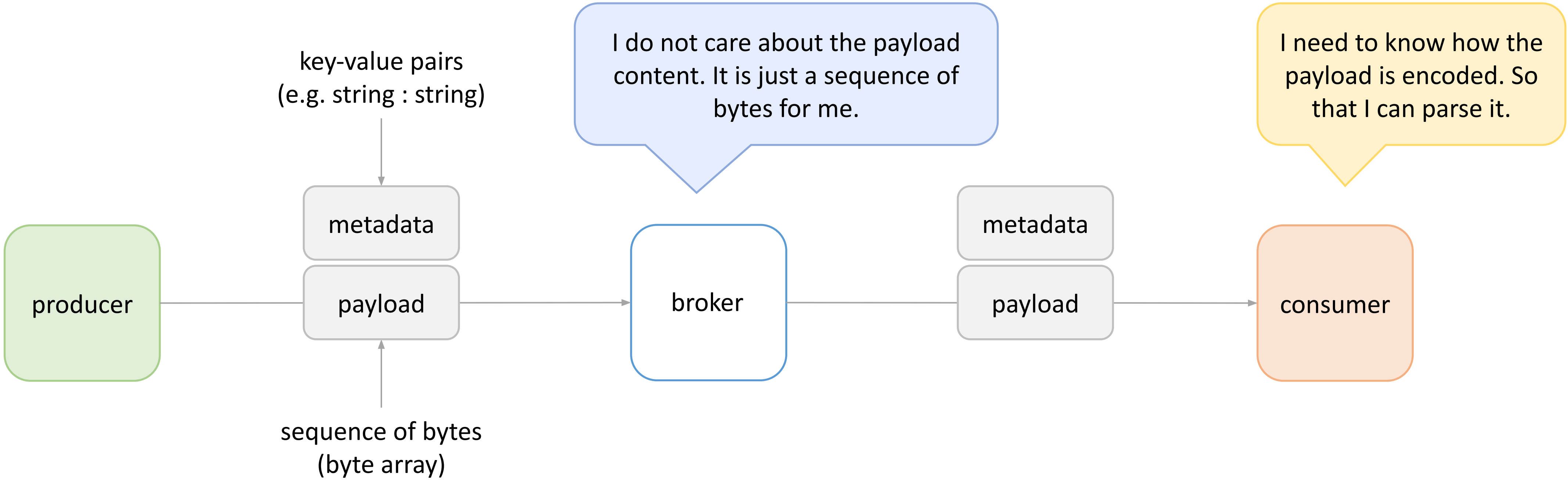
Data encoding formats



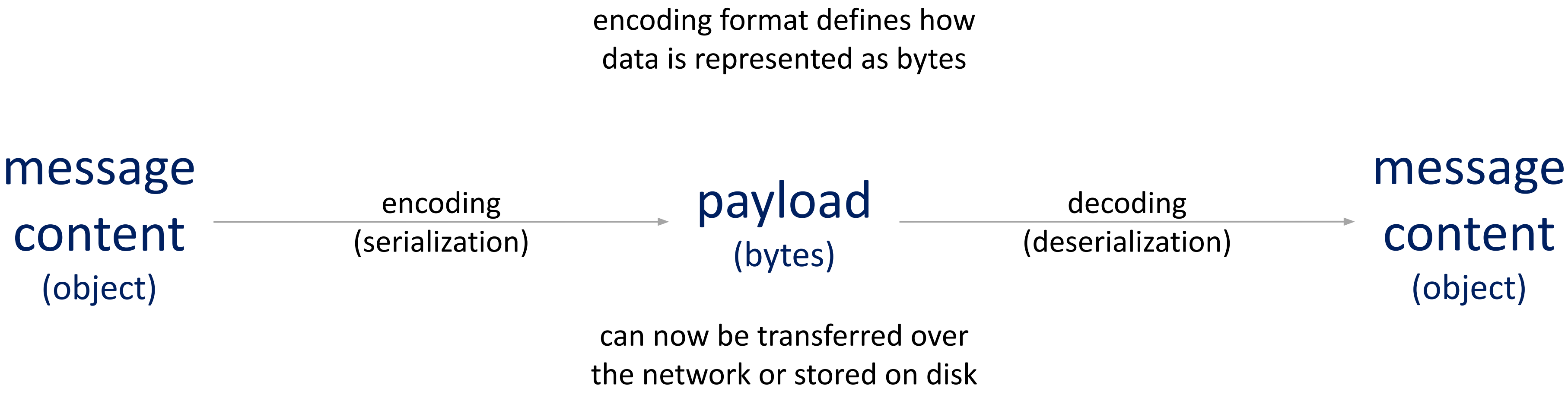
key-value pairs (e.g. string : string)

I do not care about the payload content. It is just a sequence of bytes for me.

I need to know how the payload is encoded. So that I can parse it.

sequence of bytes (byte array)

Data encoding formats



encoding format defines how
data is represented as bytes

message content (object)

encoding (serialization)

payload (bytes)

decoding (deserialization)

message content (object)

can now be transferred over
the network or stored on disk

Data encoding formats

textual formats

(JSON, XML, CSV,
…)

pros

- Human-readable (easier to debug and test).
- Widely supported by languages and tools.

cons

- Bigger messages (slower to transfer).
- Slower serialization and deserialization.

binary formats

(Thrift, Protobuf, Avro, …)

pros

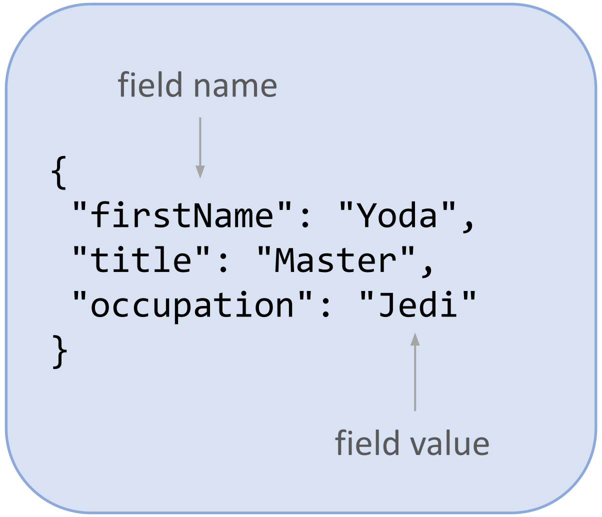
- Smaller messages
  (faster to transfer and less space needed to store).
- Faster to serialize/deserialize.

cons

- Not human-readable (harder to debug and test).

Data encoding formats

schema

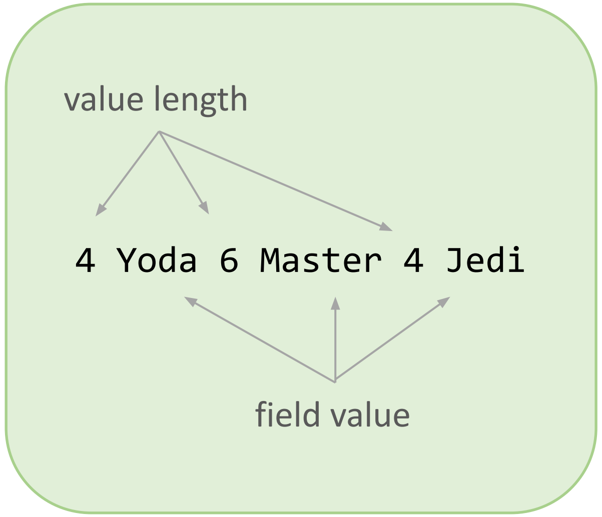


```
{
 "firstName": "Yoda",
 "title": "Master",
 "occupation": "Jedi"
}
```

payload textual representation

```
{
 "type": "record",
 "name": "StarWars",
 "fields": [
     {"name": "firstName",  "type": "string"},
     {"name": "title",      "type": "string"},
     {"name": "occupation", "type": "string"}
 ]
}
```

Avro schema



```
4 Yoda 6 Master 4 Jedi
```

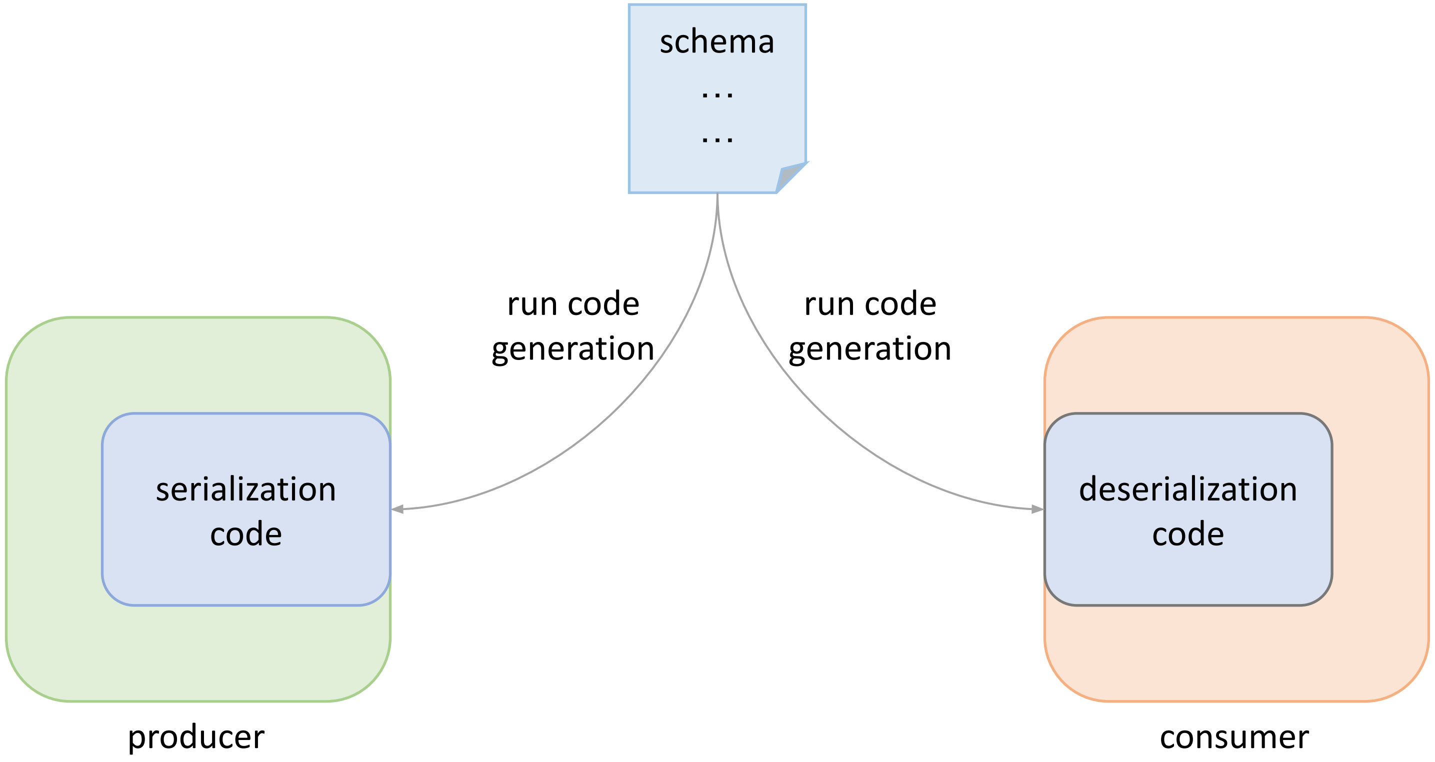
payload binary representation (as text 😀)

Data encoding formats

schema sharing options (1 of 3) - through code

examples

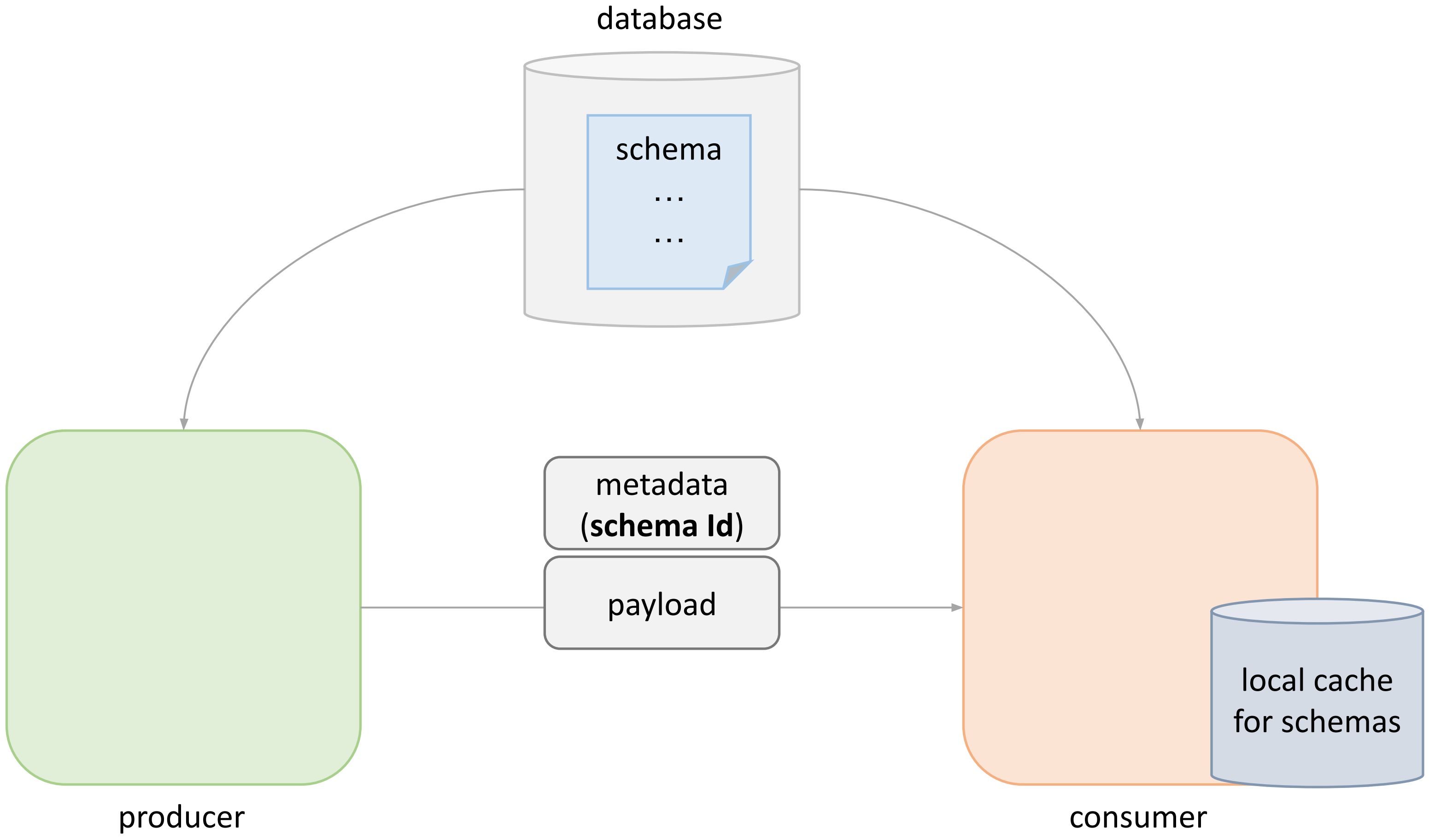
- Google Protobuf
- Facebook Thrift
- Amazon Ion



Data encoding formats

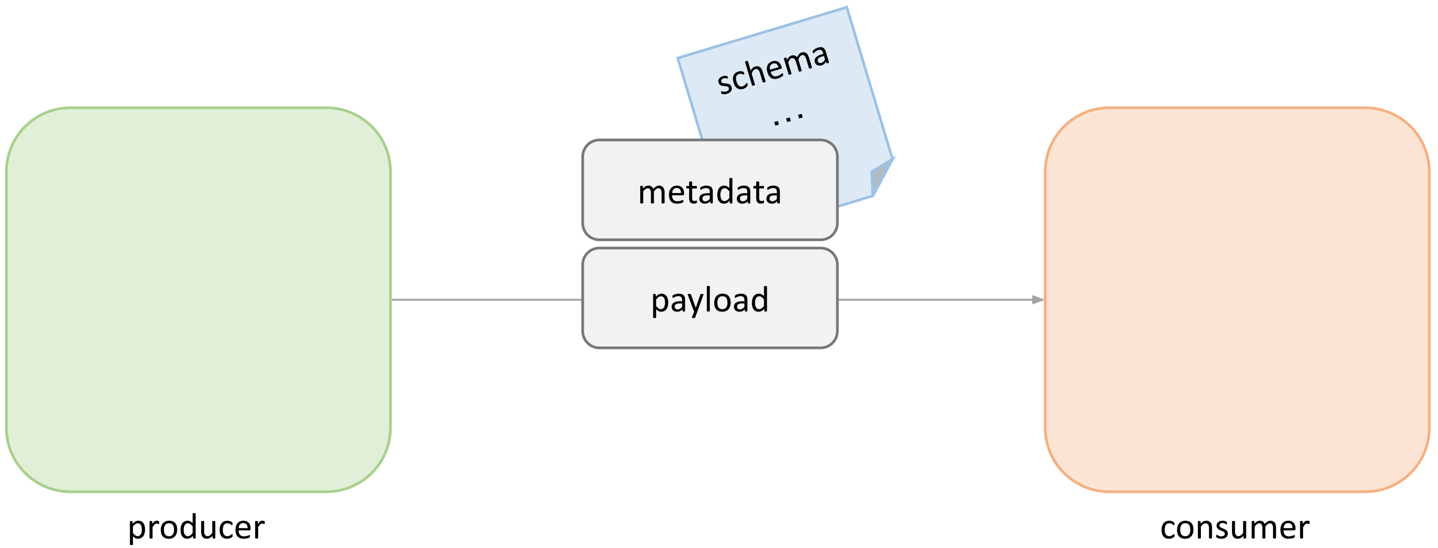
schema sharing options (2 of 3) – schema registry

popular option for
Avro schemas



Data encoding formats

schema sharing options (3 of 3) – <u>send along with the payload</u>



cons

- Increased message size
  (higher transmission latency and storage cost)

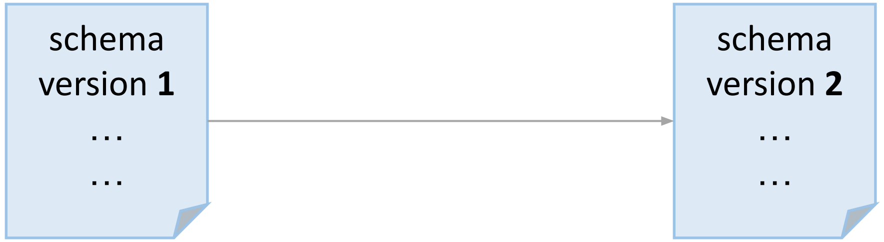
pros

- No need to build and maintain a schema registry.
- Easier to re-process messages afterwards.

Data encoding formats

schema evolution

schema will change over time

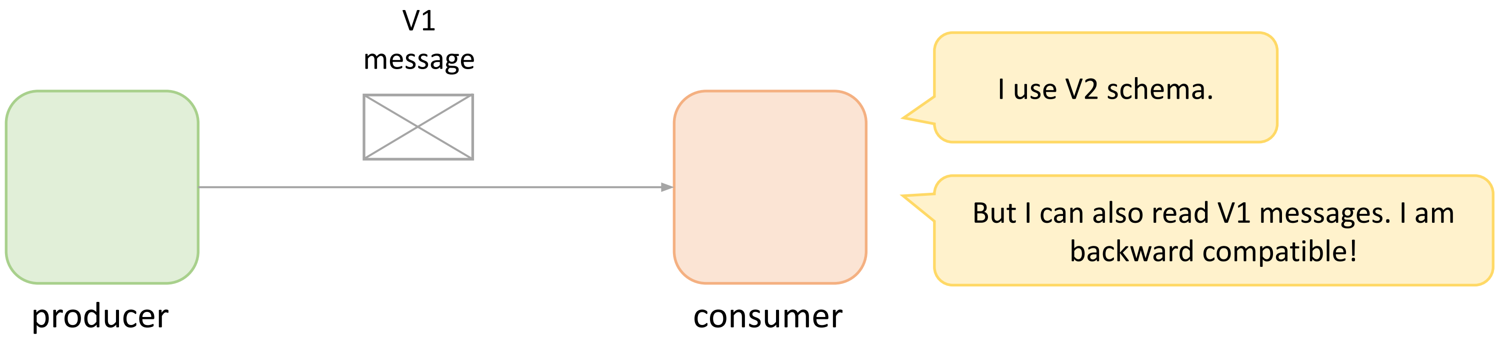


schema
version **1**
…
…

schema
version **2**
…
…

backward compatibility

consumers using the new schema can read data produced with the old schema



V1
message

producer

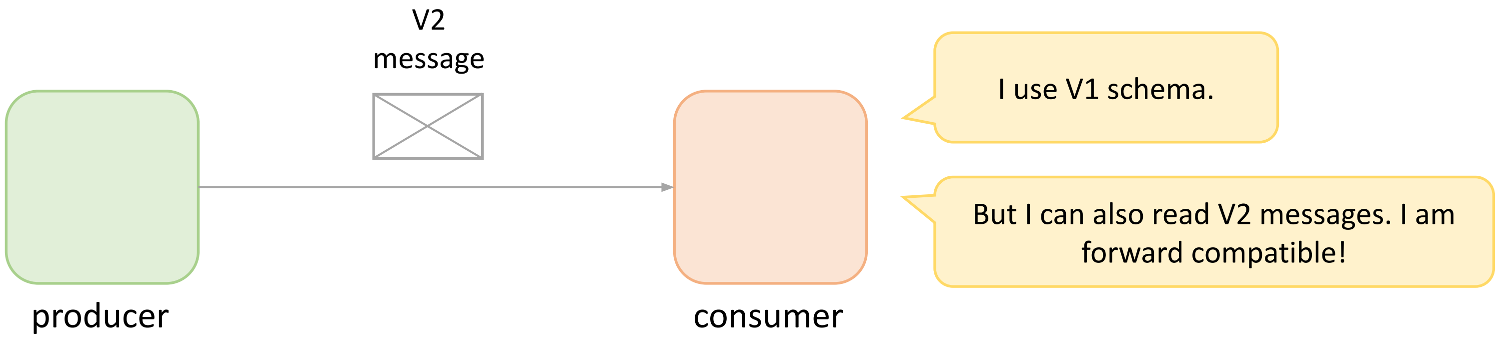
consumer

I use V2 schema.

But I can also read V1 messages. I am backward compatible!

forward compatibility

data produced with the new schema can be read by consumers using the old schema



V2
message

producer

consumer

I use V1 schema.

But I can also read V2 messages. I am forward compatible!