

Designing Scalable and Extendable Data Pipeline for Call of Duty Games

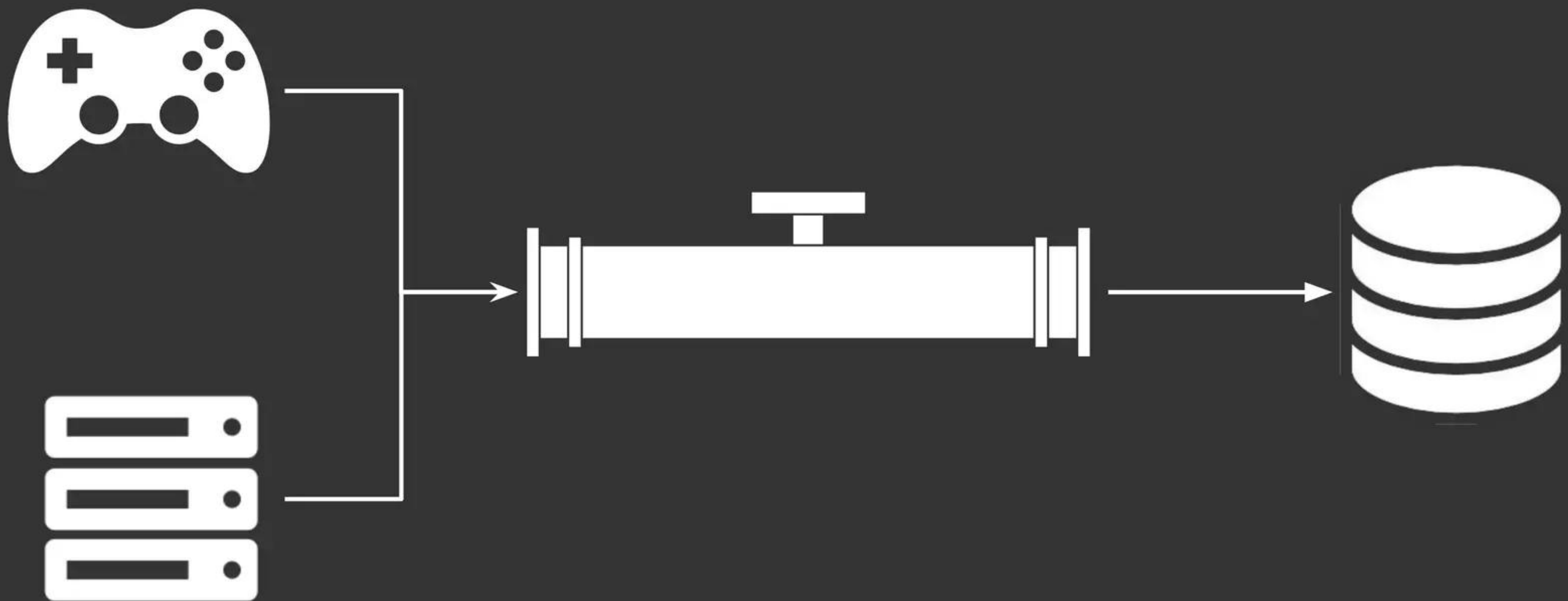


Yaroslav Tkachenko
Senior Data Engineer at Activision



WWII

CALL OF DUTY
WWII



1+
PB

Data lake size
(AWS S3)

Number of topics in the
biggest cluster
(Apache Kafka)

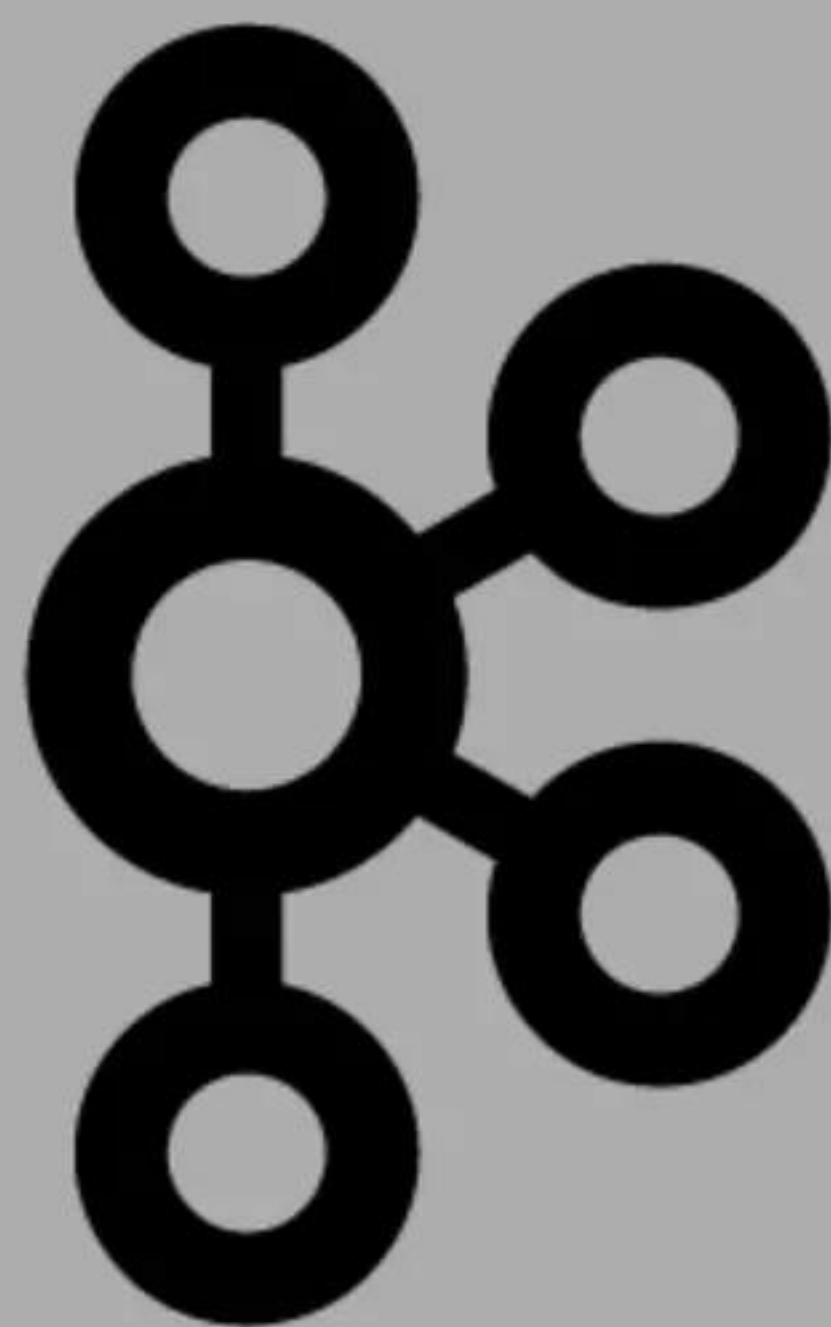
6000+

10k+

Messages per second
(Apache Kafka)

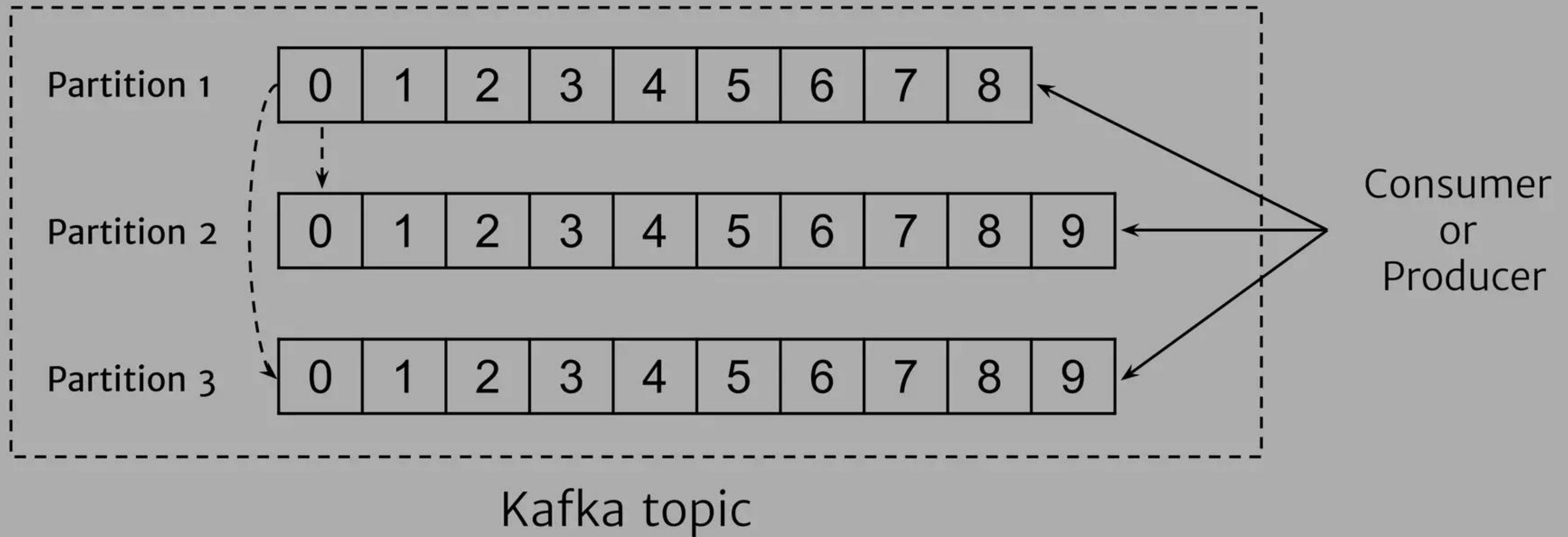
Scaling the data pipeline even further





kafka

Kafka topics are partitioned and replicated



We need to keep the number of topics and partitions low

More topics means more operational burden.

Number of partitions in a fixed cluster is not infinite.

Autoscaling Kafka is impossible, scaling is hard.

Topic naming convention

Producer

Unique game id
"CoD WW2 on PSN"

\$env.\$source.\$title.\$category-\$version

The diagram illustrates the topic naming convention with two annotations. An arrow points from the word 'Producer' to the '\$env' variable in the template. Another arrow points from the text 'Unique game id' and '"CoD WW2 on PSN"' to the '\$title' variable in the template.

prod.glutton.1234.telemetry_match_event-v1

A proper solution has
been invented decades
ago.

Think about databases.

Messaging system IS a form of a database

Data topic = Database + Table.

= Namespace + Data type.

Compare this

prod.glutton.1234.telemetry_match_event-v1

dev.user_login_records.4321.all-v1

prod.marketplace.5678.purchase_event-v1

telemetry.matches

user.logins

marketplace.purchases

Each approach has pros and cons

- Topics that use metadata for their names are obviously easier to track and monitor (and even consume).
- As a consumer, I can consume exactly what I want, instead of consuming a single large topic and extracting required values.
- These dynamic fields can and *will* change. Producers (sources) and consumers *will* change.
- Very efficient utilization of topics and partitions.
- Finally, it's impossible to enforce any constraints with a topic name. And you can always end up with dev data in prod topic and vice versa.

After removing
necessary metadata
from the topic names
stream processing
becomes mandatory.

Stream processing becomes mandatory

Measuring → Validating → Enriching → Filtering & routing

Refinery

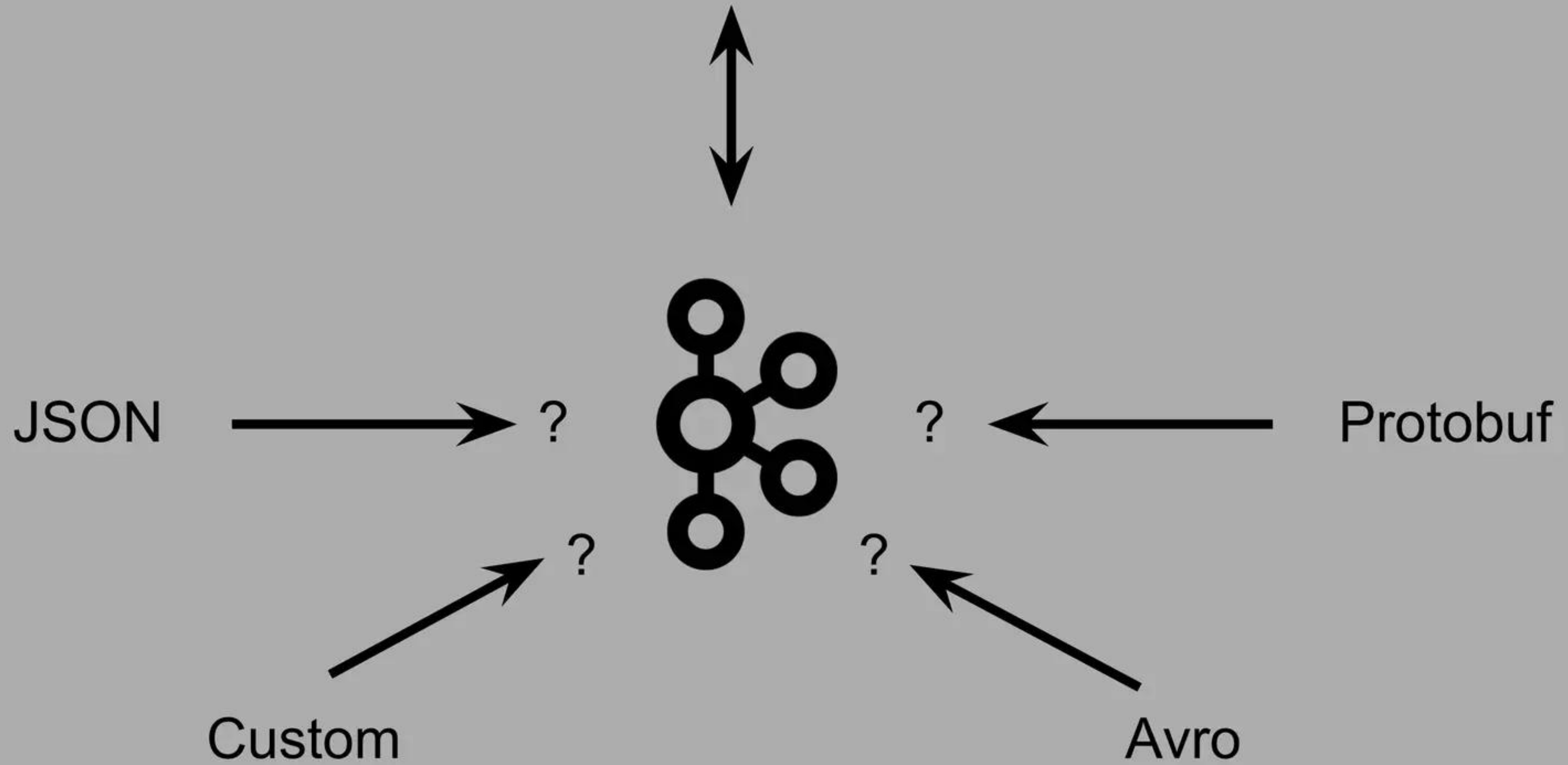


Having a single
message schema for a
topic is more than
just a nice-to-have.

Number of supported
message formats

8

Stream processor



Custom deserialization

```
// Application.java  
props.put("value.deserializer", "com.example.CustomDeserializer");
```

```
// CustomDeserializer.java  
public class CustomDeserializer implements Deserializer<???> {  
    @Override  
    public ??? deserialize(String topic, byte[] data) {  
        ???  
    }  
}
```

Message envelope anatomy



Unified message envelope

```
syntax = "proto2";
```

```
message MessageEnvelope {  
    optional bytes message_id = 1;  
    optional uint64 created_at = 2;  
    optional uint64 ingested_at = 3;  
    optional string source = 4;  
    optional uint64 title_id = 5;  
    optional string env = 6;  
    optional UserInfo resource_owner = 7;  
    optional SchemaInfo schema_info = 8;  
    optional string message_name = 9;  
    optional bytes message = 100;  
}
```

Schema Registry

- API to manage message schemas
- **Single** source of truth for all producers and consumers
- It should be **impossible** to send a message to the pipeline without registering its schema in the Schema Registry!
- Good Schema Registry supports immutability, versioning and basic validation
- Activision uses custom Schema Registry implemented with Python and Cassandra

Summary: scaling and extending the data pipeline

Games

- By using unified message envelope and topic names adding a new game becomes almost effortless
- “Operational” stream processing makes it possible
- Still flexible enough: each game can use its own message payload format via Schema Registry

Use-cases

- Topic names express data types, not producers or consumers
- Stream filtering & routing allows low-cost experiments
- Data catalog built on top of Schema Registry promotes data discovery

Thanks !

@sap1ens