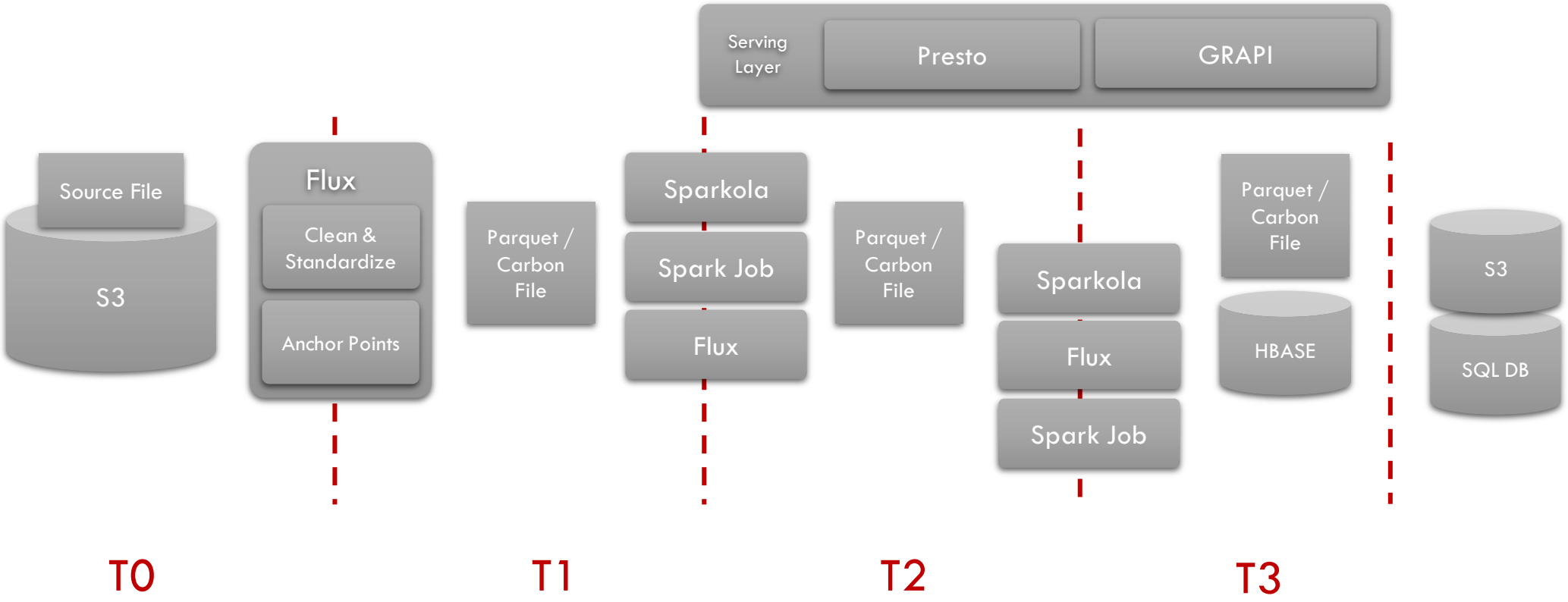
AEROSPIKE NEXTGEN NOW SUMMIT '20

Building a Real-time Ingestion Platform using Kafka and Aerospike

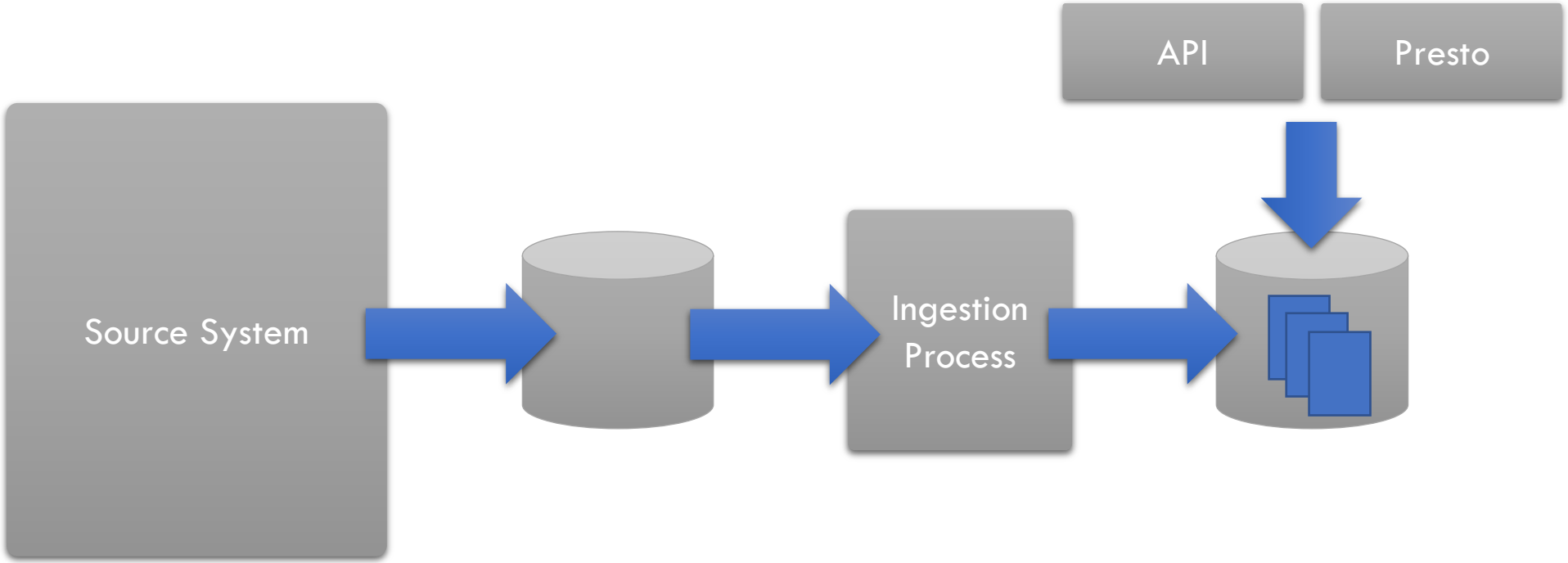
DBS

Matteo Pelati
EXECUTIVE DIRECTOR, TECHNOLOGY
HEAD OF DATA PLATFORM
DBS BANK

Batch-oriented Platform



Batch Ingestion



Source System produces daily files and uploads them into an S3 bucket

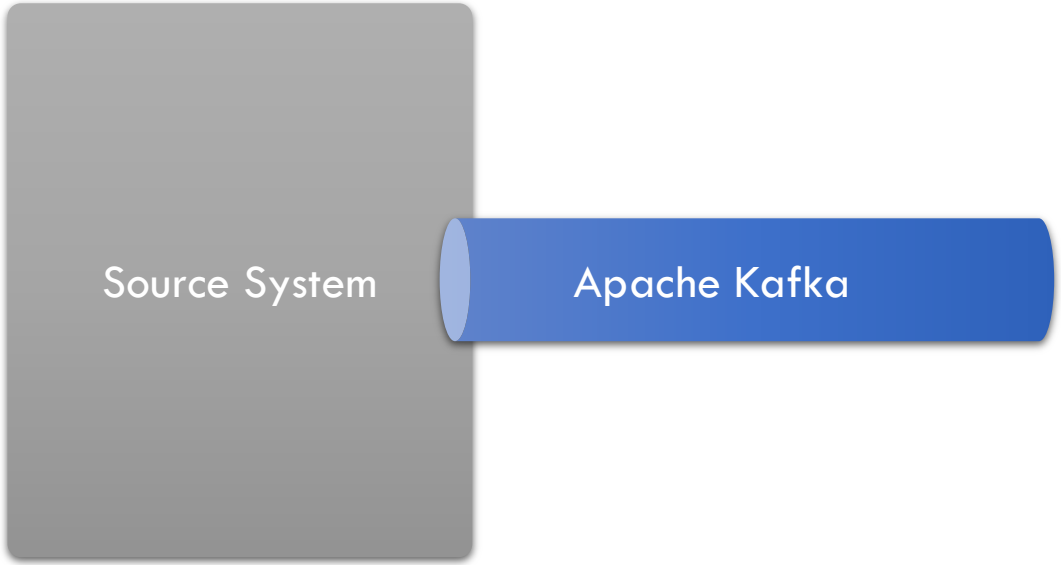
Files are standardized and converted to Parquet/Carbon format

We need streaming support

Challenges

- Typical storage formats used in data platforms are not suitable for streaming
- Kafka historical storage does not fit the purpose as we do not simply need historical records. Records need to be “projected” to represent the current state
- A distributed database for the entire data platform would be too costly as we keep most of the historical records in S3

Partial Events

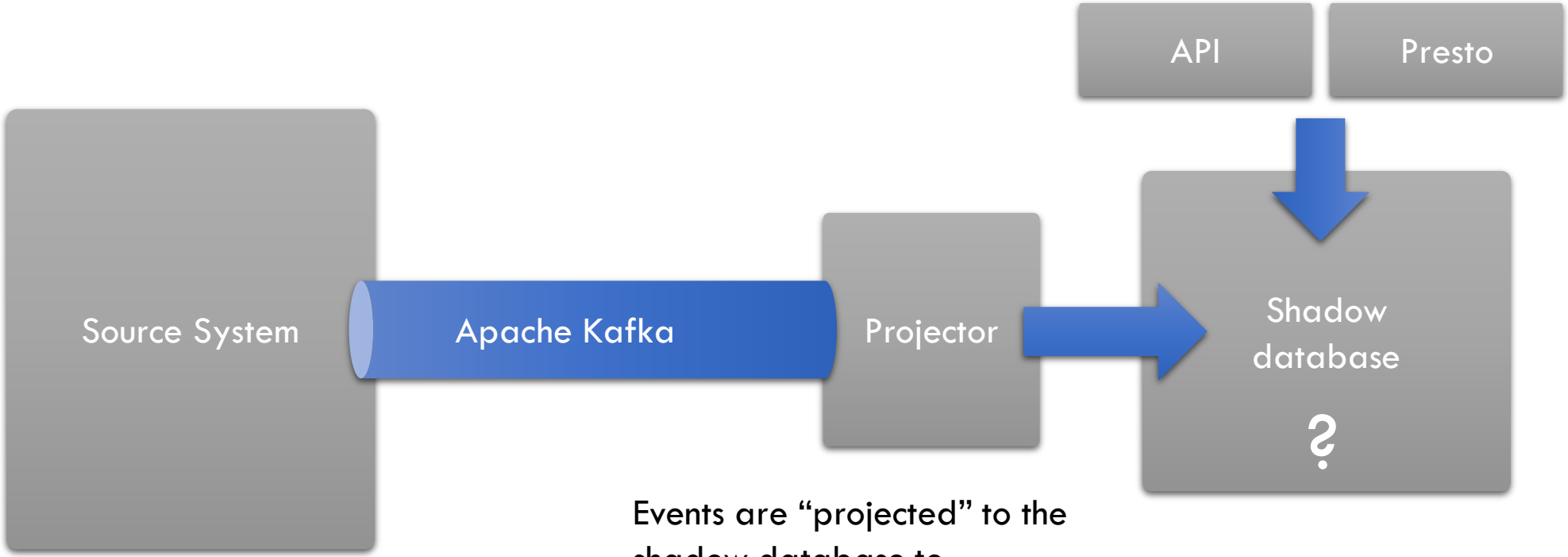


Source System generates events notifying any state change (CICS, CDC) and push them to Kafka

```
{
  fields:{
     "phone_number": "91147890"
  },
  op: "UPDATE"
  pk: 3434833838
}
```

```
{
  fields:{
     "home address": "91 Kovan Road"
  },
  op: "UPDATE"
  pk: 3434833838
}
```

Eventing + shadow database



Source System generates events notifying any state change (CICS, CDC) and push them to Kafka

Events are "projected" to the shadow database to replicate the data

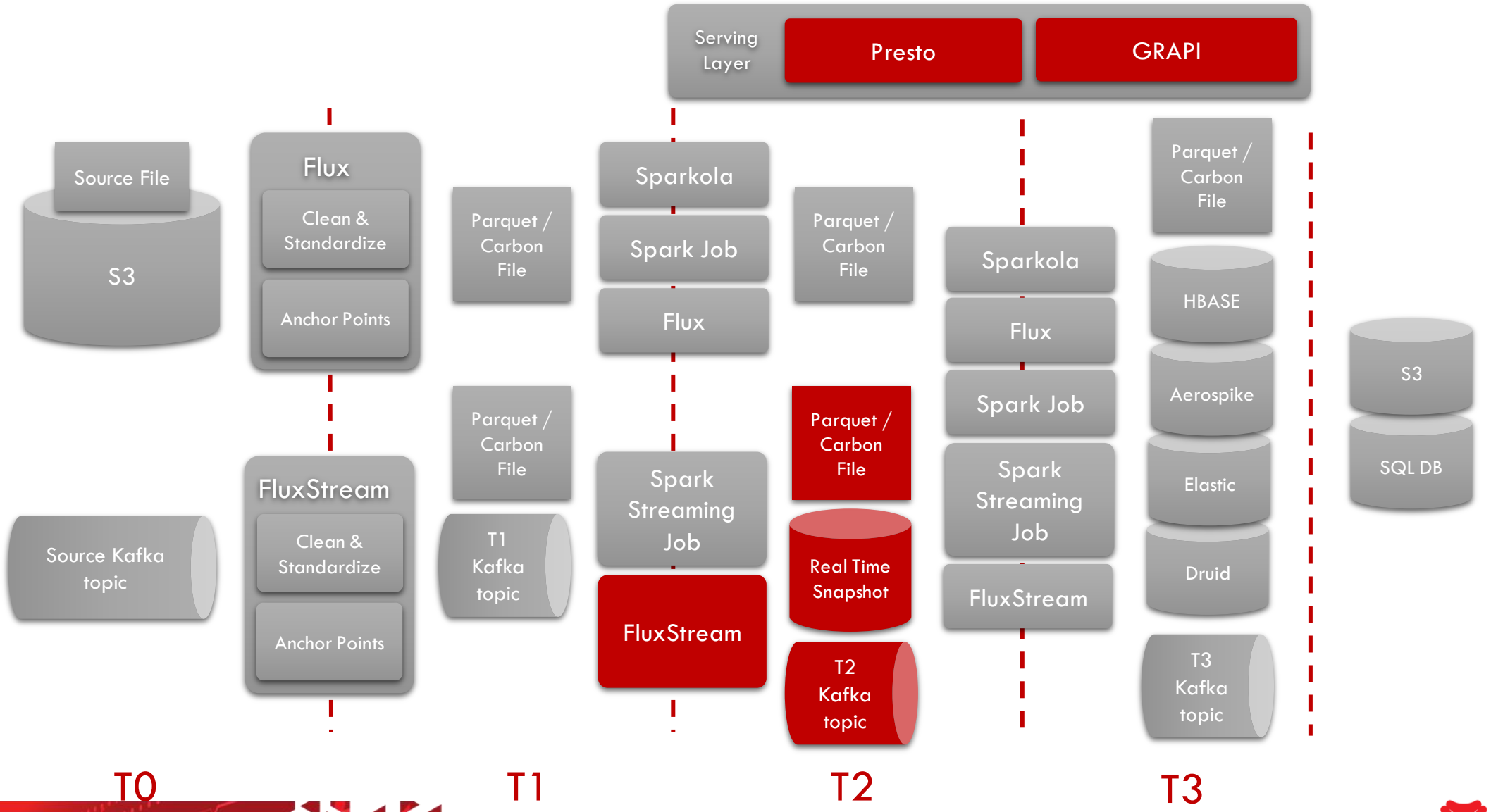
Shadow database contains an eventually consistent replica of the source system

Database choice

	Aerospike	MongoDB	Redis	FDB
Fast key/value lookup	●	●	●	●
Open Source	●	●	●	●
Memory + flash storage	●	●	●	●
Shared-nothing architecture	●	●	●	●
Complex data structures	●	●	●	●
Secondary Indexes	●	●	●	●
Commercial Support	●	●	●	●

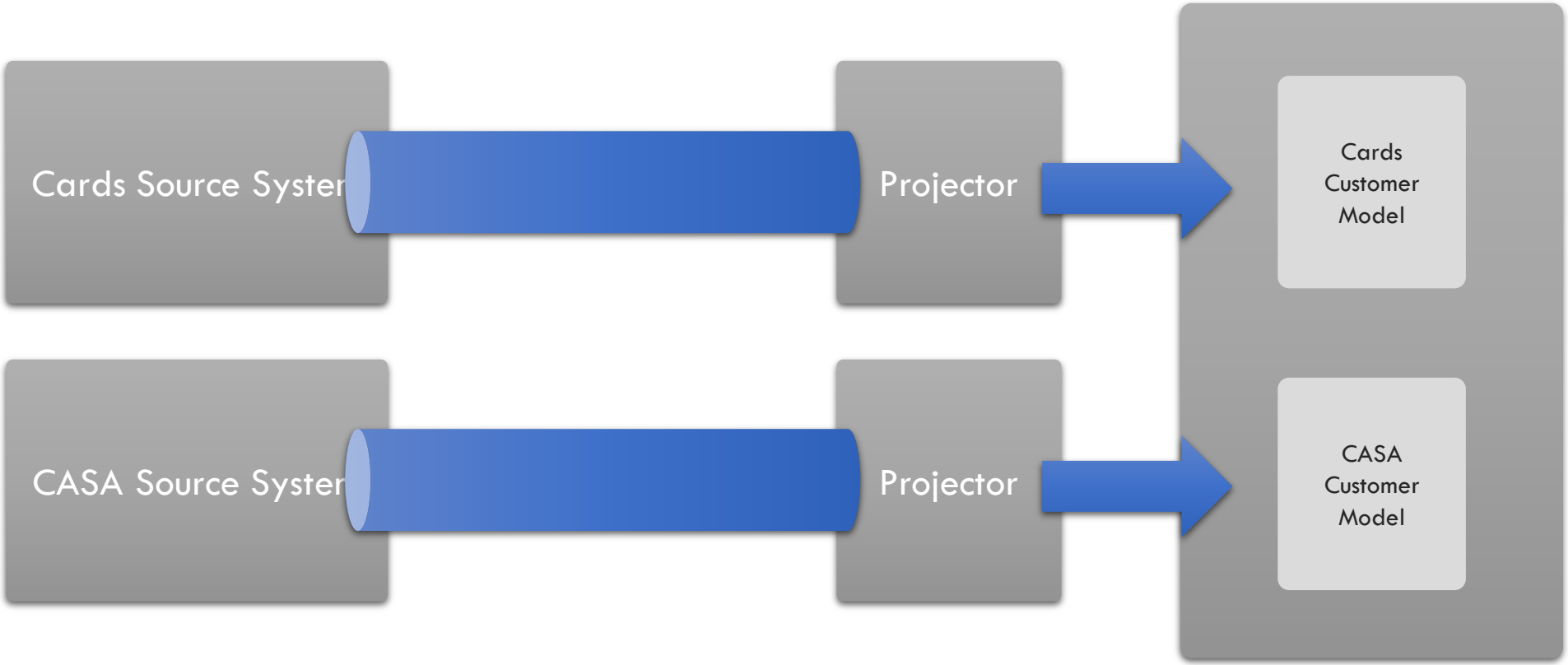
Based on the table above, we decided to adopt Aerospike as the database of choice for all services

With Real Time ingestion



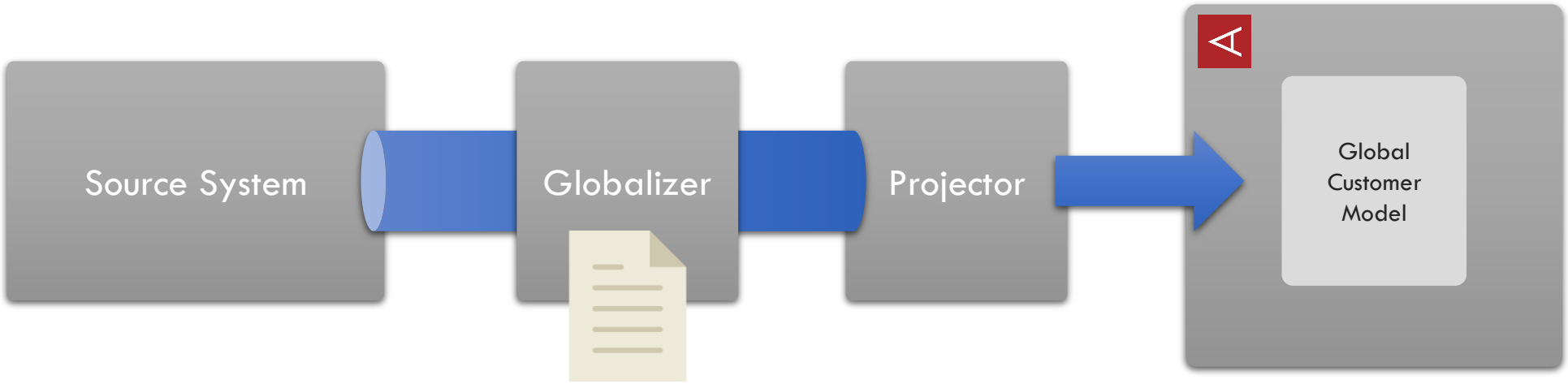
We needed a Global Model

The same business entity is very different from source system to source system. We need an efficient way to "globalize" the model stored in the database.



Introducing ”the Globalizer”

Every model gets translated into a Global model by a Globalizer component, which is fully configurable using a custom DSL. This way, domain developers can easily plug-in new source systems without requiring to write any code.

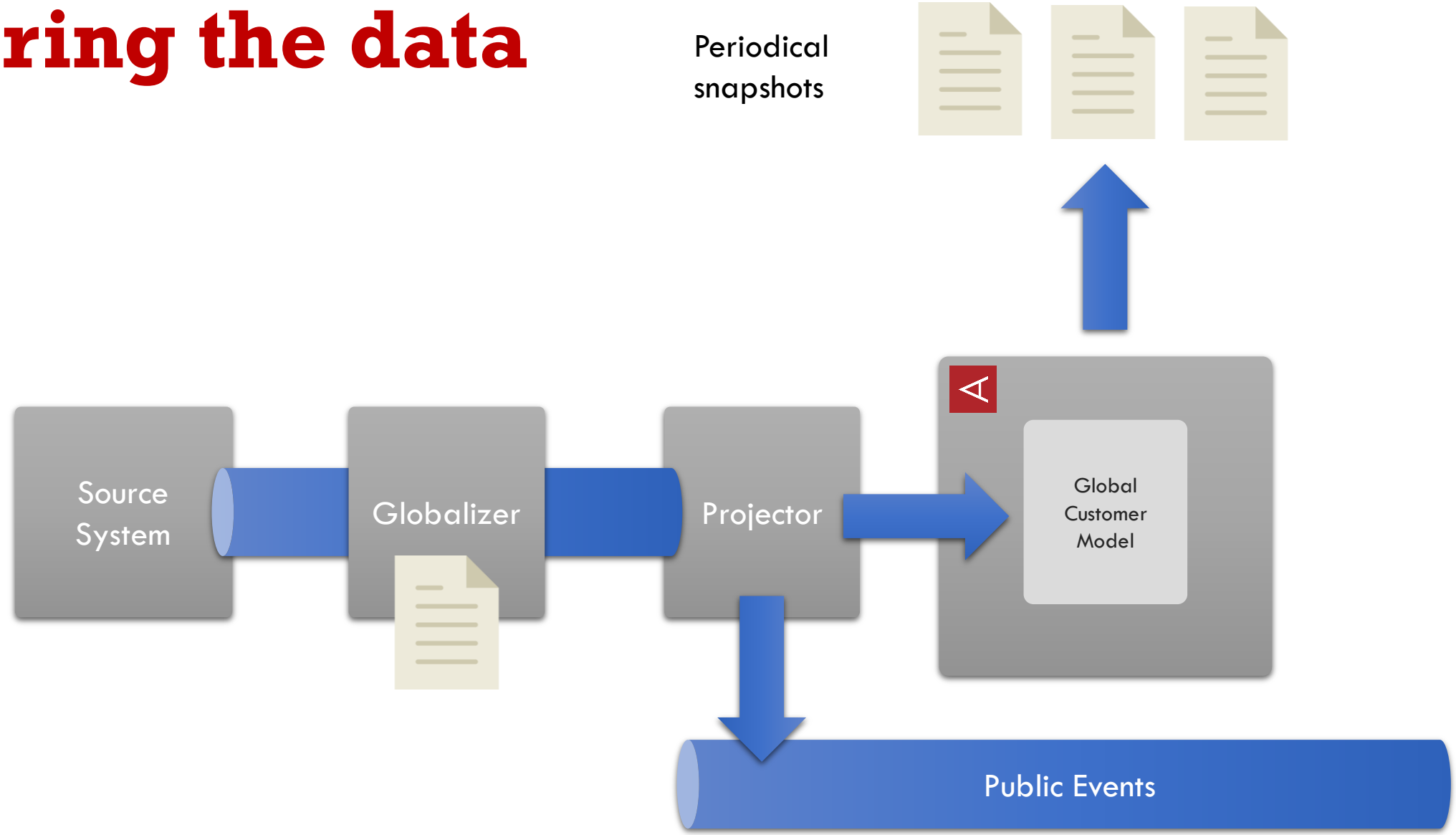


Data Format

- **De facto standard for streaming**: Avro is the format of choice of Apache Kafka
- **RPC Support**: Avro can seamlessly be used for implementing RPC services
- **Compact**: it's binary, thus it's fast
- **Evolutionary**: its supports very well schema evolution

Storing the data

Periodical snapshots



The Avro Record Format

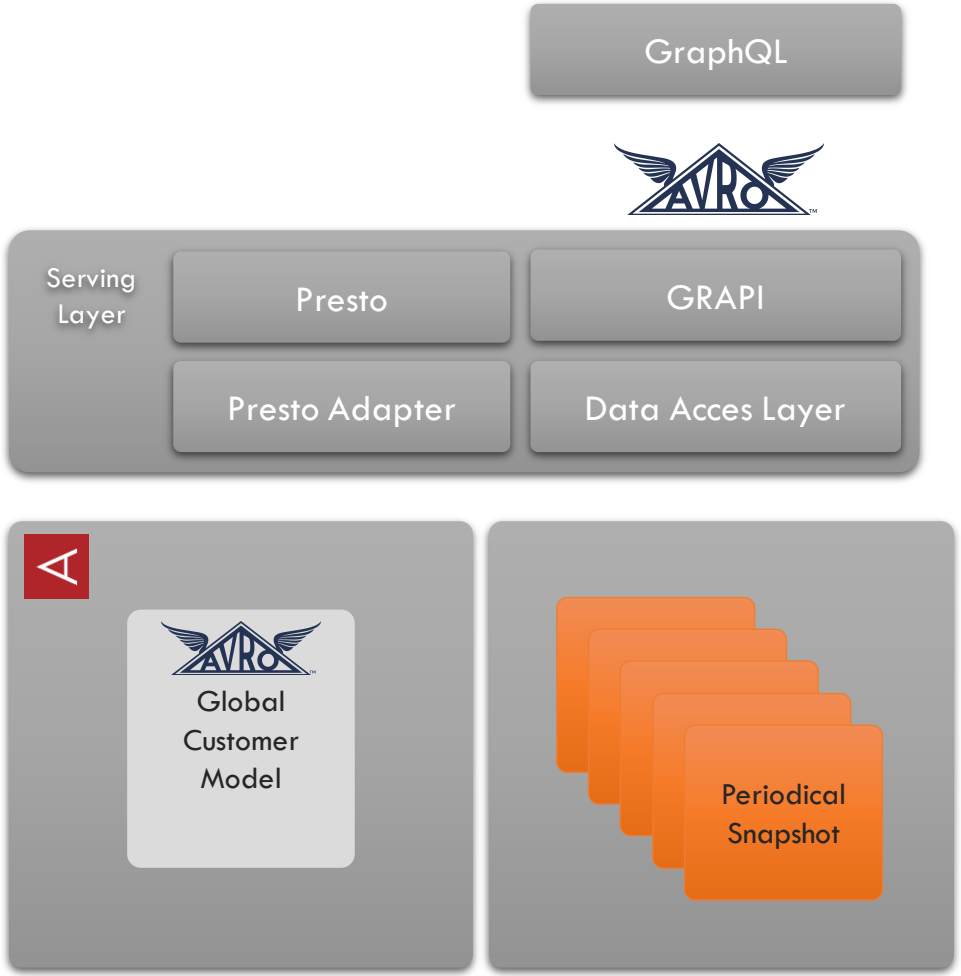
How can we speed-up even further the responses? Just dump the AVRO payload into the database

```
{
   "type" : "record",
   "namespace" : ”Customers",
   "name" : ”Customer",
   "fields" : [
      { "name" : ”CustomerId" , "type" : "string”, “pk”: true },
      { "name" : ”Country" , "type" : ”string”, “sk”: true }
   ]
}
```

Payload:	a002ef10cc76eb21964abbf3489
CustomerId	366635326
Country	SG

The Payload field is used to return the raw data to the client during an API call, while other fields are solely used for creating secondary indexes. This design was inspired by the Record Layer of FDB from Apple

We still have APIs



We still have challenges

- **Eventual consistency sometimes is a problem**: applications are not designed based on eventual consistency principle. They expect after an action, data is updated immediately. That is not the case for an eventual consistent system.
- **The batch nature of core systems sometimes require the cache to be completely refreshed**: not always source systems can generate events in real time. Sometimes, after a batch operation, the cache needs to be refreshed in bulk

Thank You