Building a Business Logic Translation Engine with Spark Streaming for Communicating Between Legacy Code and Microservices

Patrick Bamba, Attestation Légale (ALG)

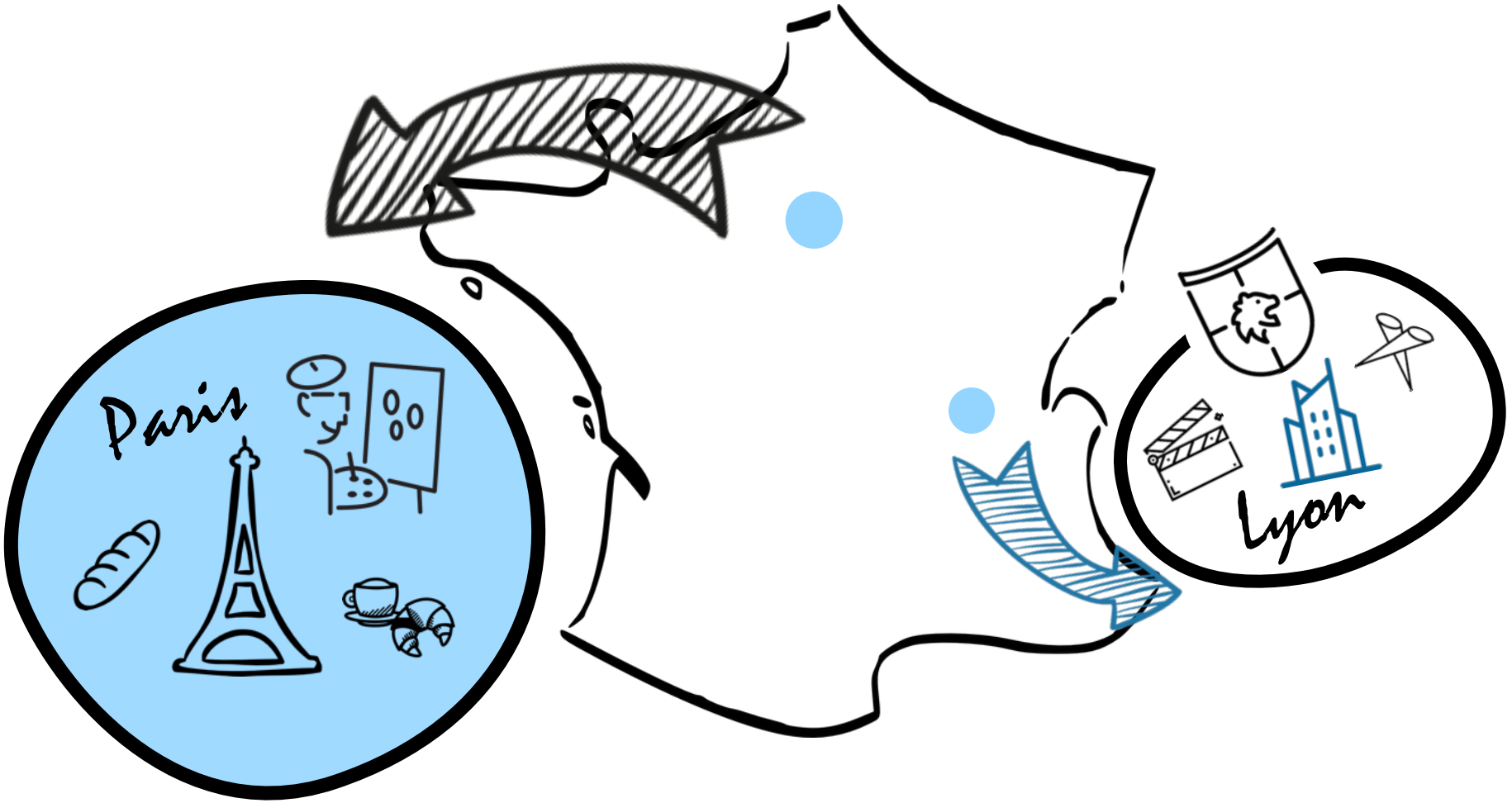
#EUstr6

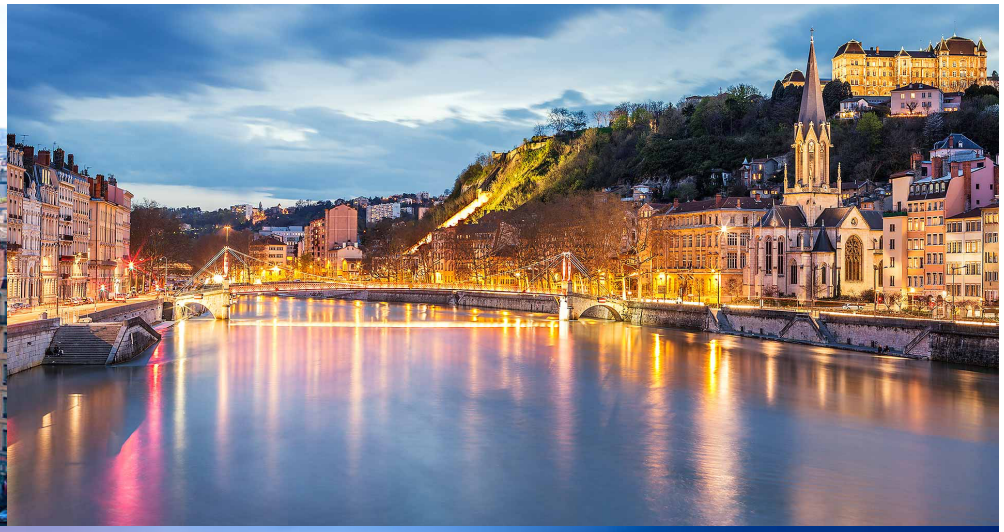
Agenda

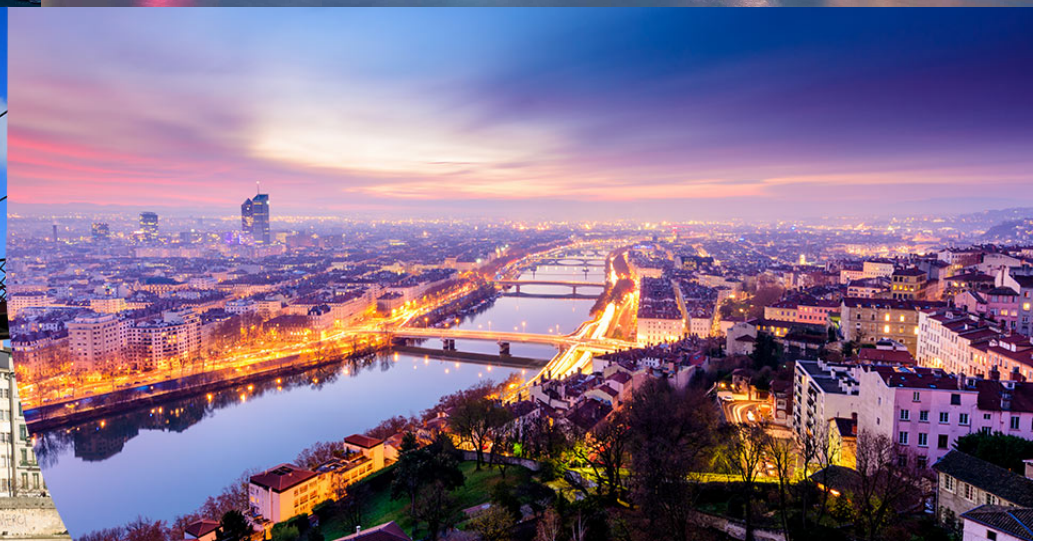
- Who we are
- What we do
- Our context
- The translation engine
- Why Spark ?
- Implementation example
- Takeaways

Who we are

- B2B social network
- Customers in Construction, Temporary work, Transport & other industries
- 3 French Tech Pass (+100% growth in turnover per year)
- From Lyon, France

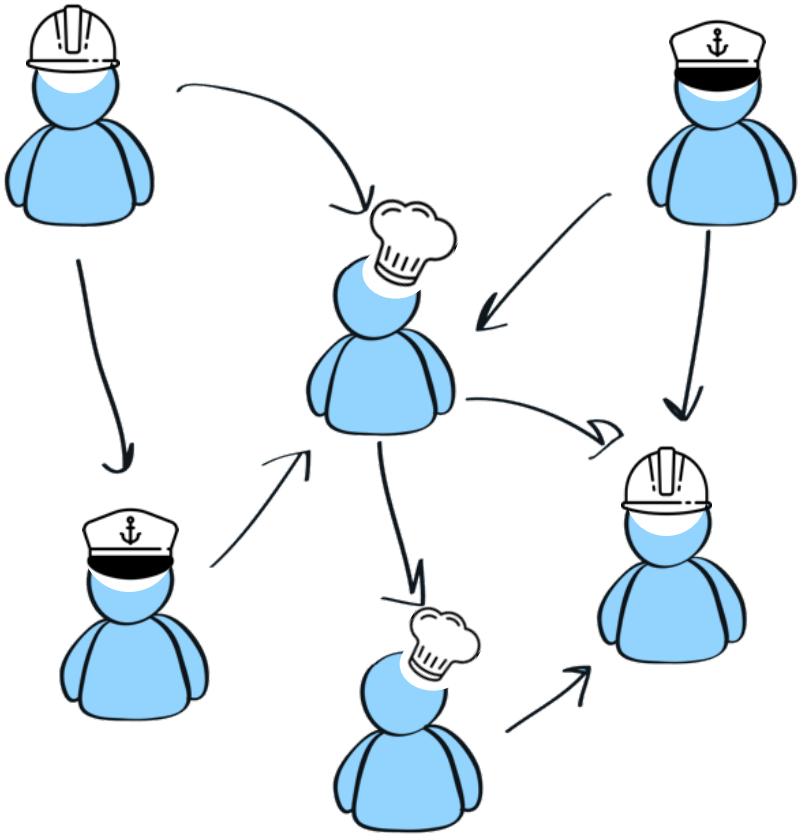

Paris
Lyon

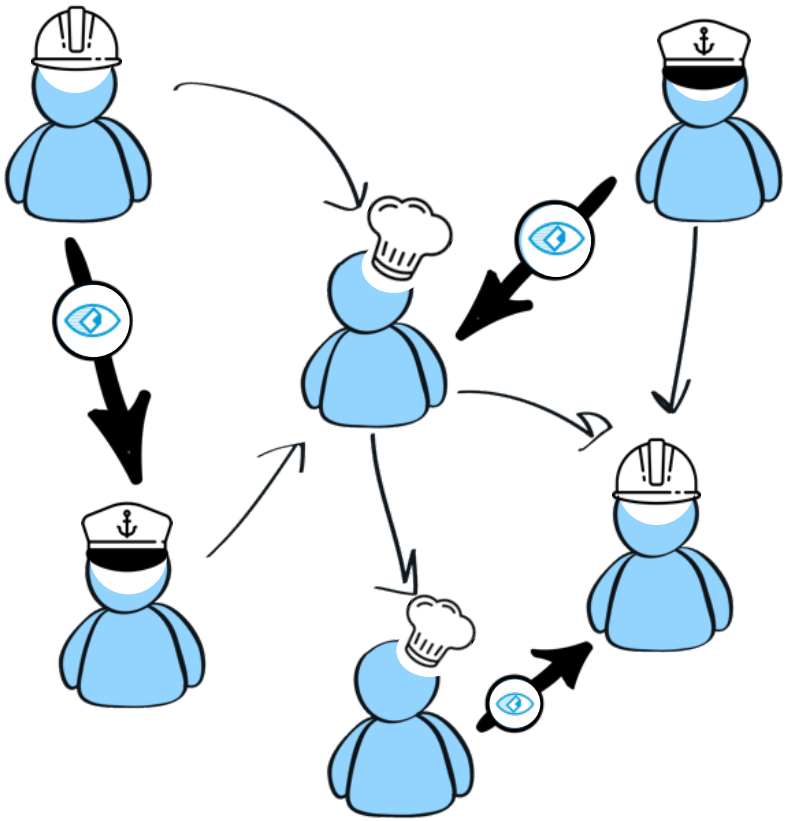



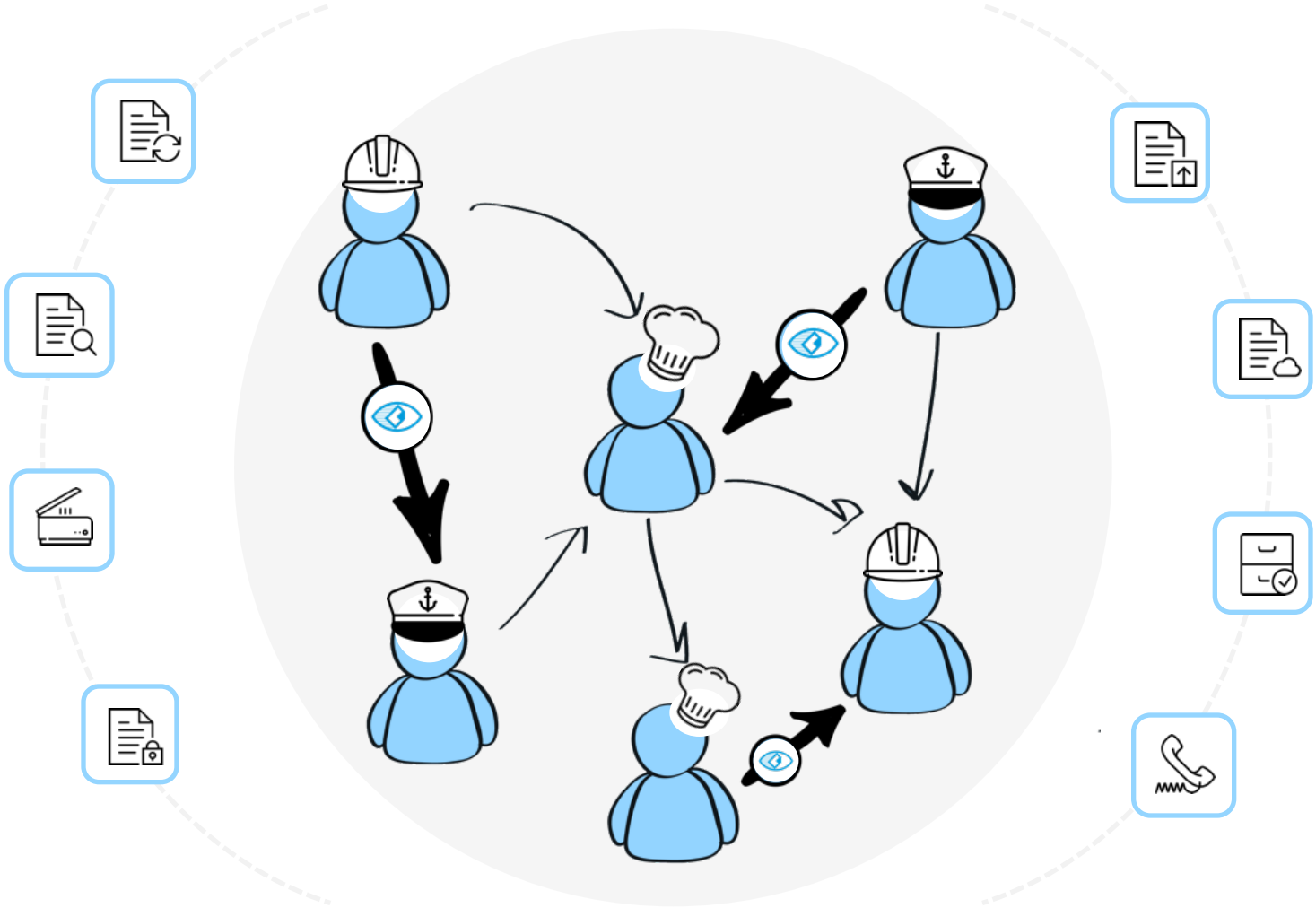


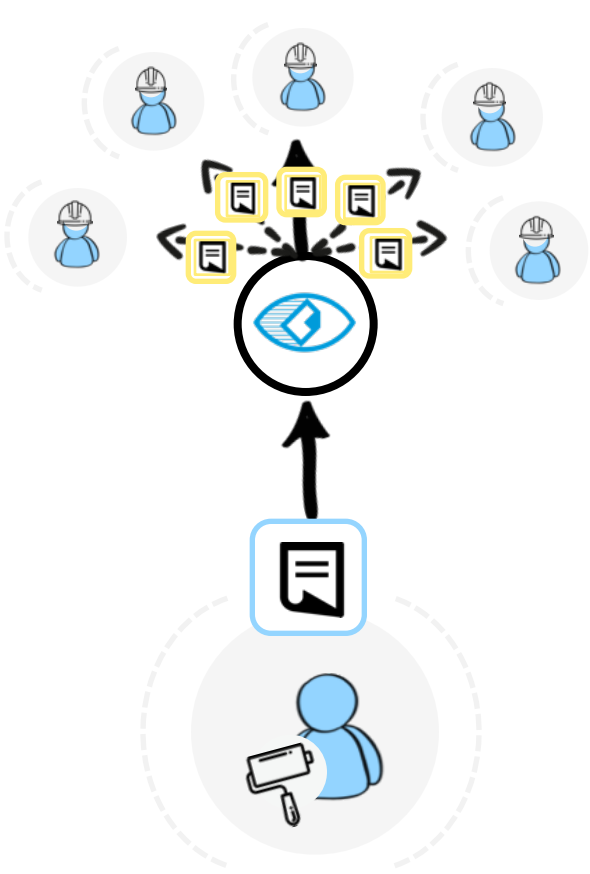
What we do

- Reducing risk in business relationships
- Reducing administrative burden
- Enabling supplier/client relationship management and discovery

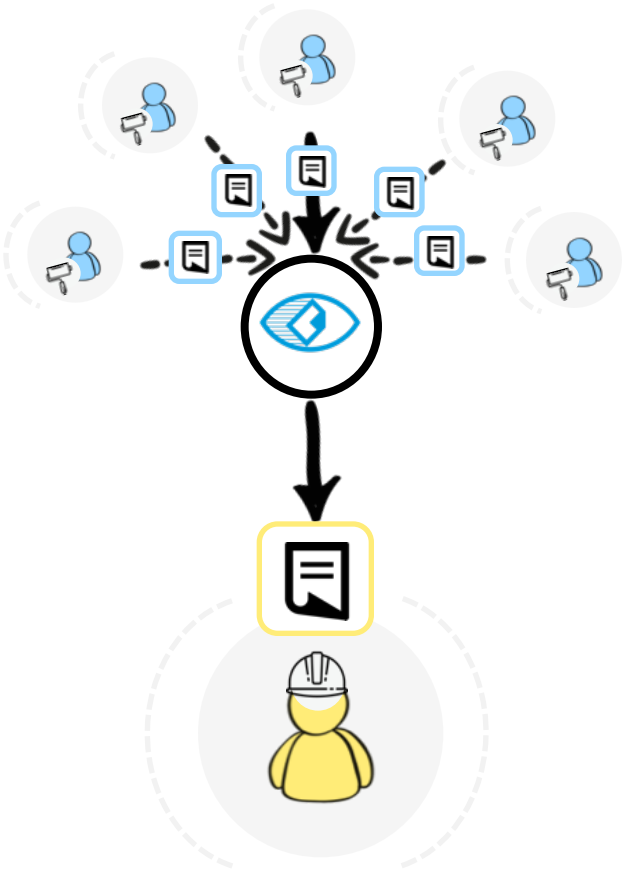








SUPPLIER

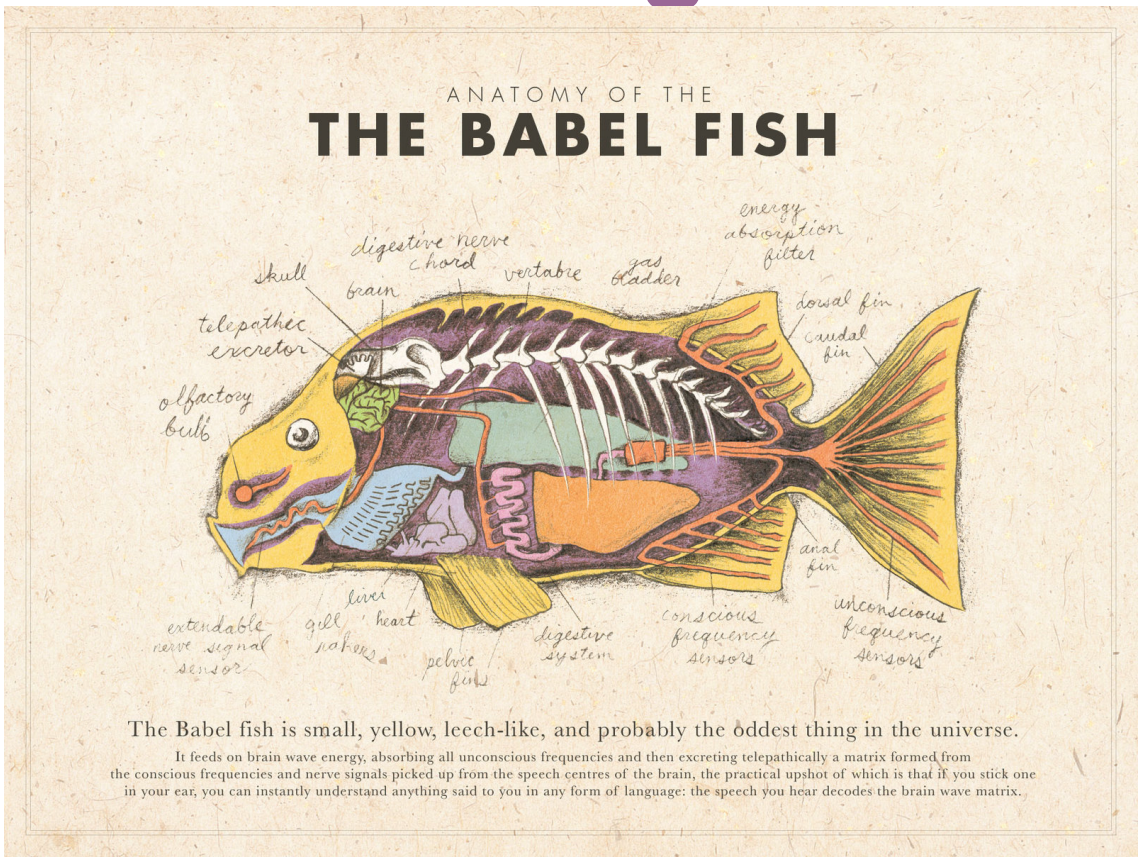


CLIENT

Our context

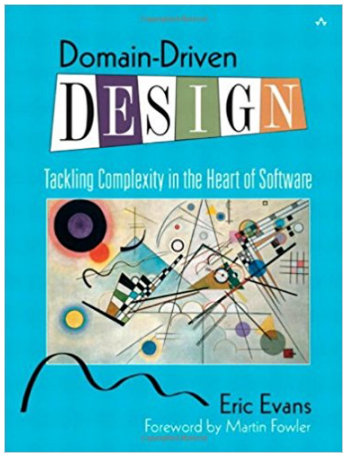
- Context
  - Monolith-first strategy
  - Technical debt
- Problem
  - Heterogeneous customer base (clients / suppliers)
  - Striving to become the market leader
- Strategy
  - Incremental refactoring
  - Strangler applications

The translation engine



The translation engine

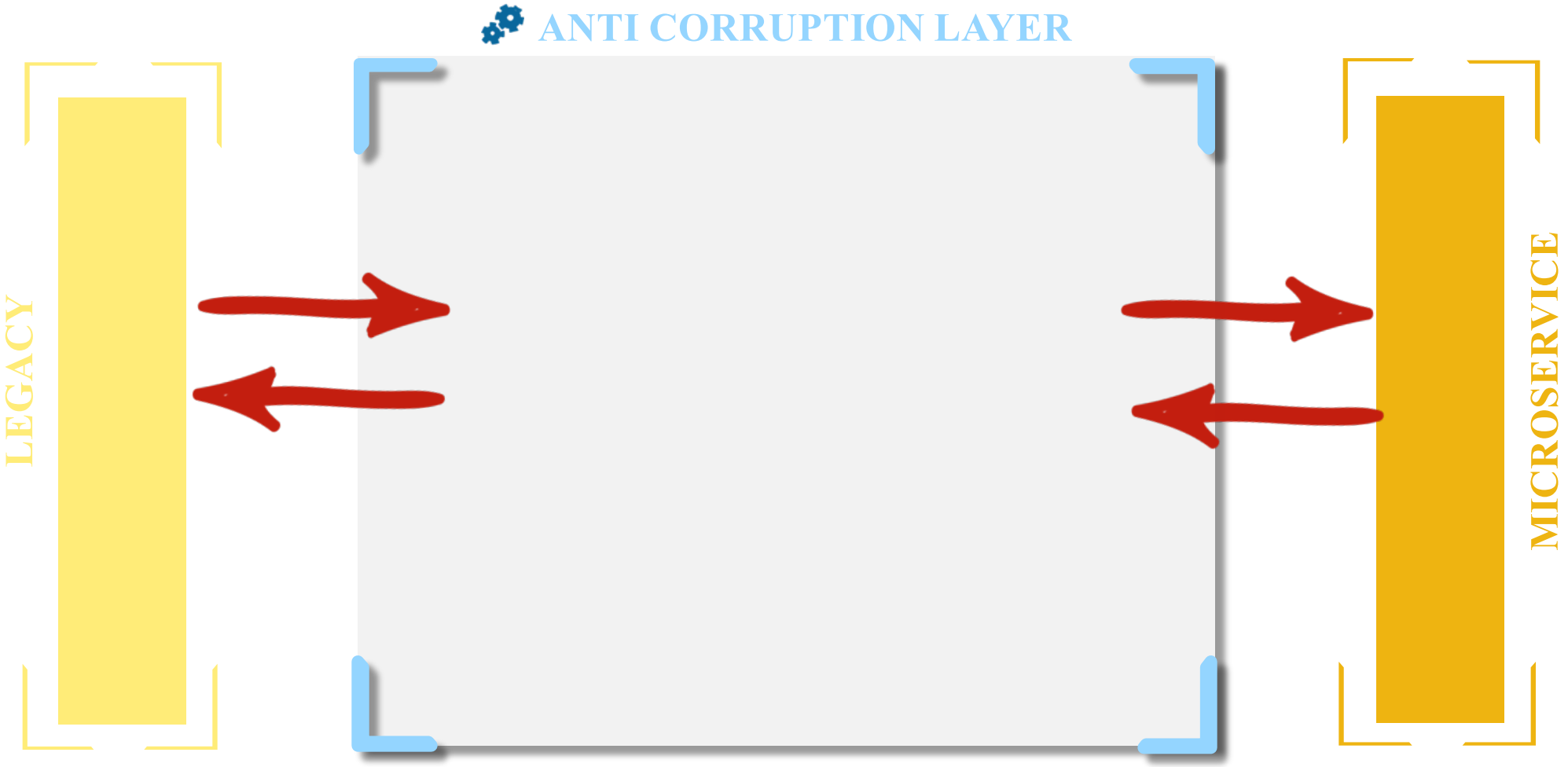
- ***Anti-corruption layer:*** *" It is a mechanism that translates conceptual objects and actions from one model and protocol to another " – Eric Evans, Domain Driven Design*



The translation engine

- Translation layer between two bounded contexts with a large interface
- Usually between modern applications and legacy system
- Removes the dependency on legacy system


ANTI CORRUPTION LAYER
LEGACY
MICROSERVICE 1
MICROSERVICE 2
MICROSERVICE 3


ANTI CORRUPTION LAYER
LEGACY
MICROSERVICE

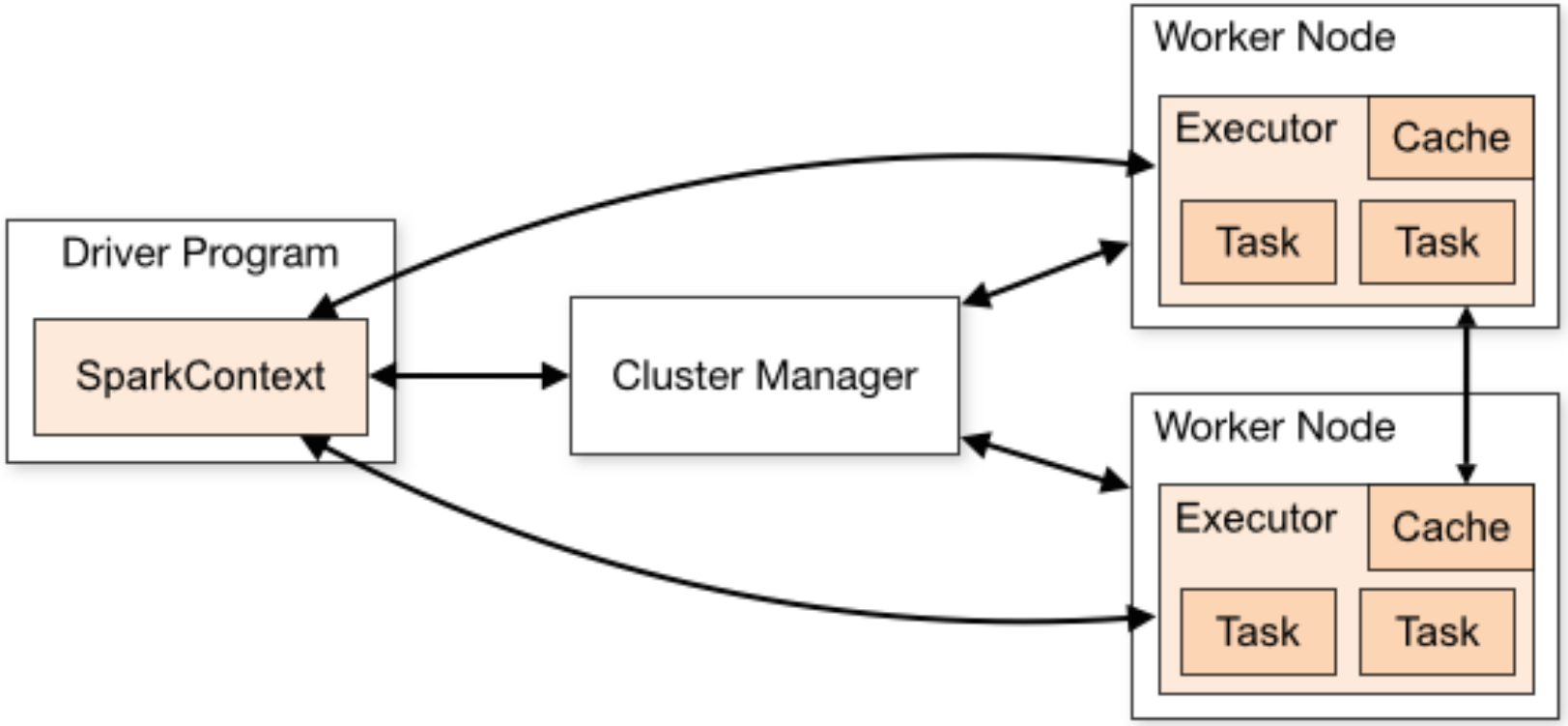
Why Spark ?

- *" There are many hurdles in interfacing with an external system."*
  - *" the **infrastructure** layer must provide the means to communicate with another system that might be on a different platform or use different **protocols** "*
  - *" The **data types** of the other system must be translated into those of your system "*

Why Spark ?

- Types of hurdle
  - **Architecture**
  - **Infrastructure**
  - **Data types & protocols**
  - Model

Why Spark ? Architecture



Why Spark ? Infrastructure

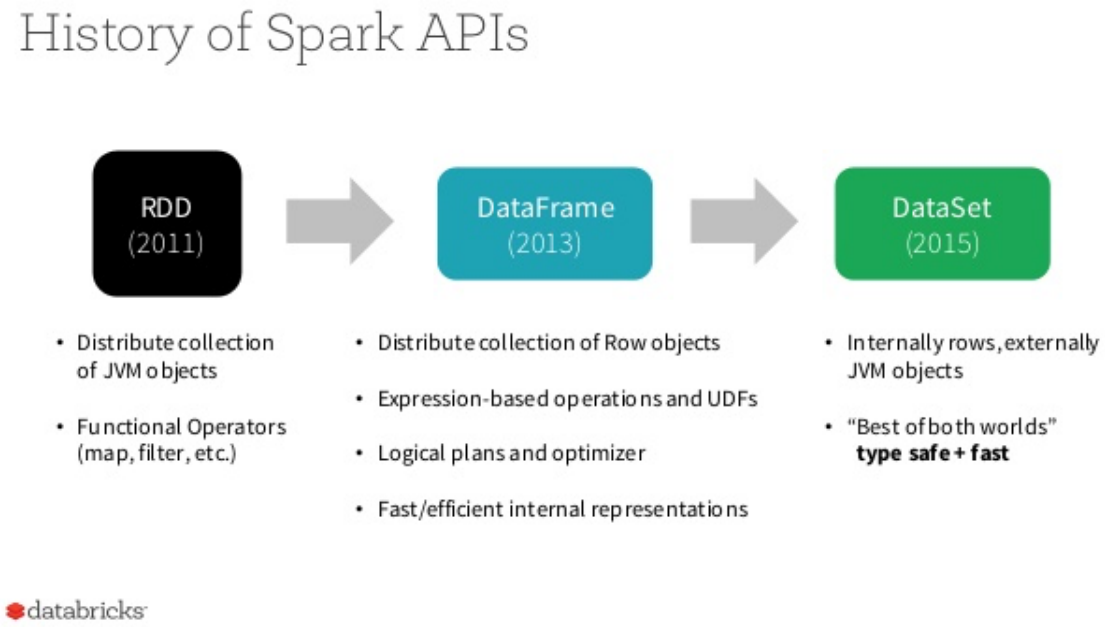
- Cluster managers
  - Standalone
  - Apache Mesos
  - Hadoop YARN
- Services
  - Databricks
  - AWS EMR
  - Google Cloud Dataproc

Why Spark ? Data types & protocols

- Built-in data sources
  - Parquet files
  - JSON datasets
  - Hive tables
  - JDBC data sources
- Open source connectors
  - Elasticsearch
  - Cassandra
  - Neo4j
  - Redis
  - Redshift
  - ….

Why Spark ? Model

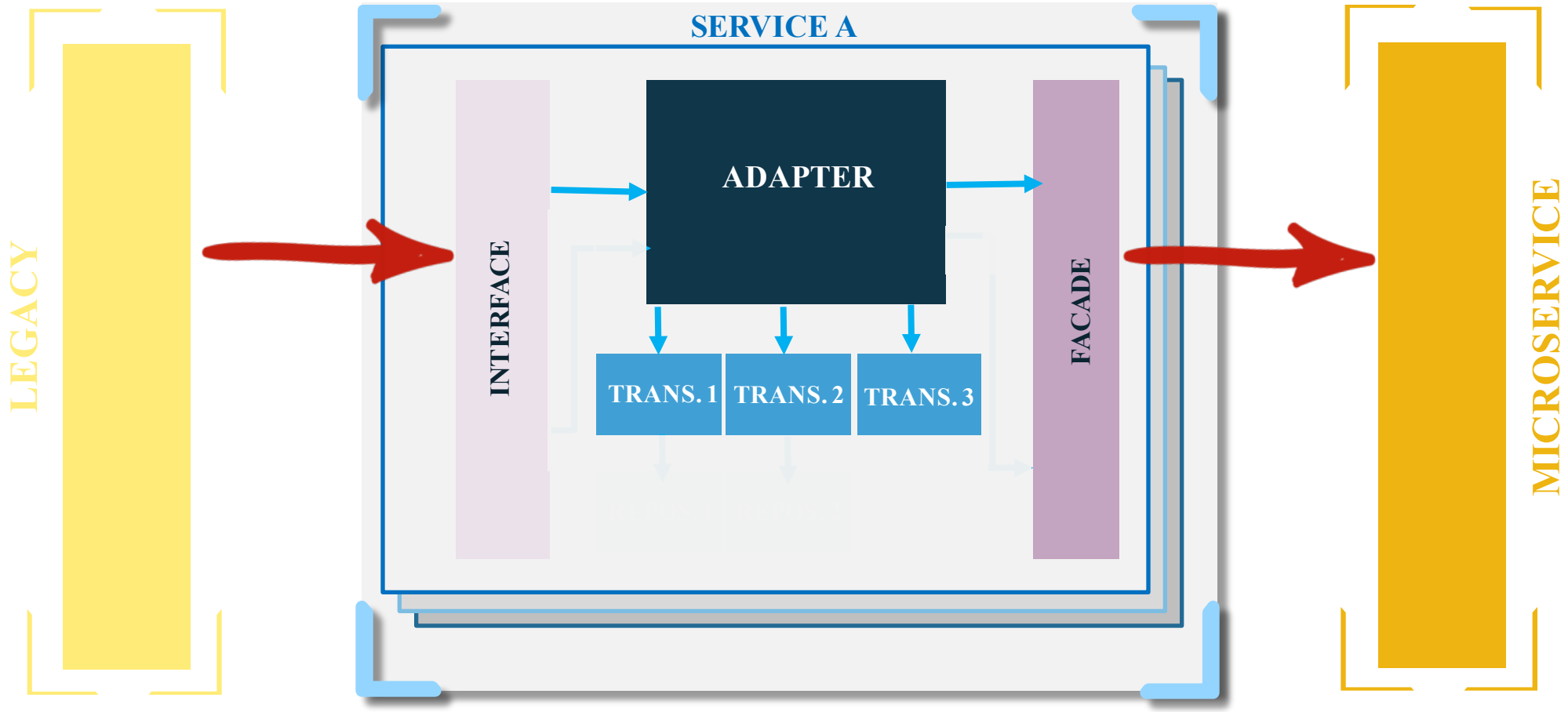
- APIs
  - RDD
  - Dataframe
  - Datasets



Why Spark ? Streaming

- Event-Driven Architecture
- Unification of streaming and batch processing
- Low latency
- End to end reliability and correctness guarantees

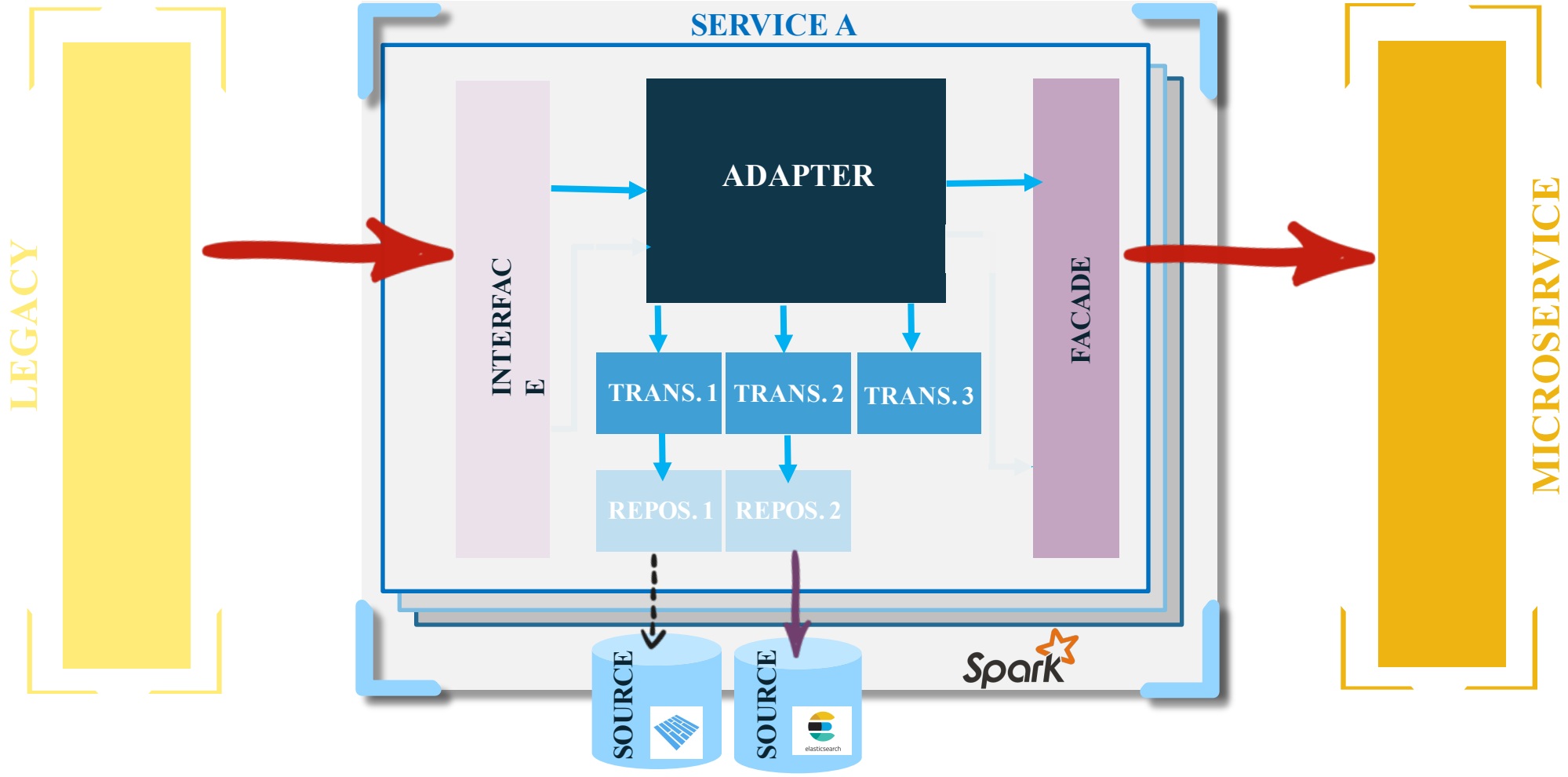
ANTI CORRUPTION LAYER



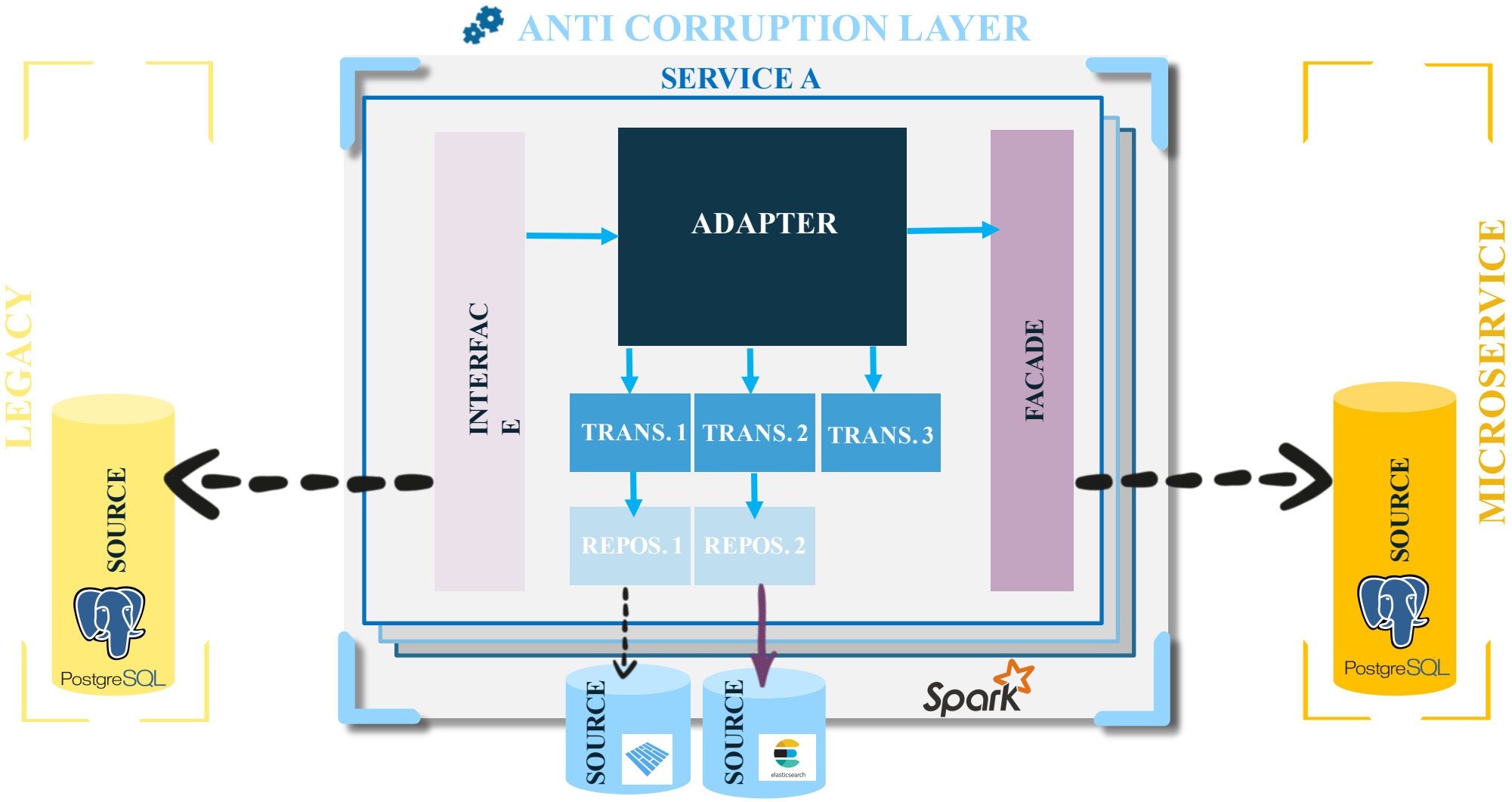
```scala
case class LegacyAdministrativeDocument(id: Int, branchCode: String, insuranceId: Int)

case class AdministrativeDocument(id: Int, branch: Branch, insurance: Insurance)

case class Branch(code: String, legalName: String, town: String, country: String)

case class Insurance(id: Int, companyName: String)

case class TranslationStep(
                           legacyAdministrativeDocument: LegacyAdministrativeDocument,
                           branch: Option[Branch],
                           insurance: Option[Insurance],
                           translationErrors: ListBuffer[TranslationError])

trait Translator {
  val description: String
  val translate: Dataset[TranslationStep] => Dataset[TranslationStep]
}

class InsuranceResolver(sparkSession: SparkSession) extends Translator with Serializable {

  override val description: String = "Translator responsible for adding insurance information"

  override val translate: Dataset[TranslationStep] => Dataset[TranslationStep] = {
    (translationStepDs) =>
        ...
  }
}
```

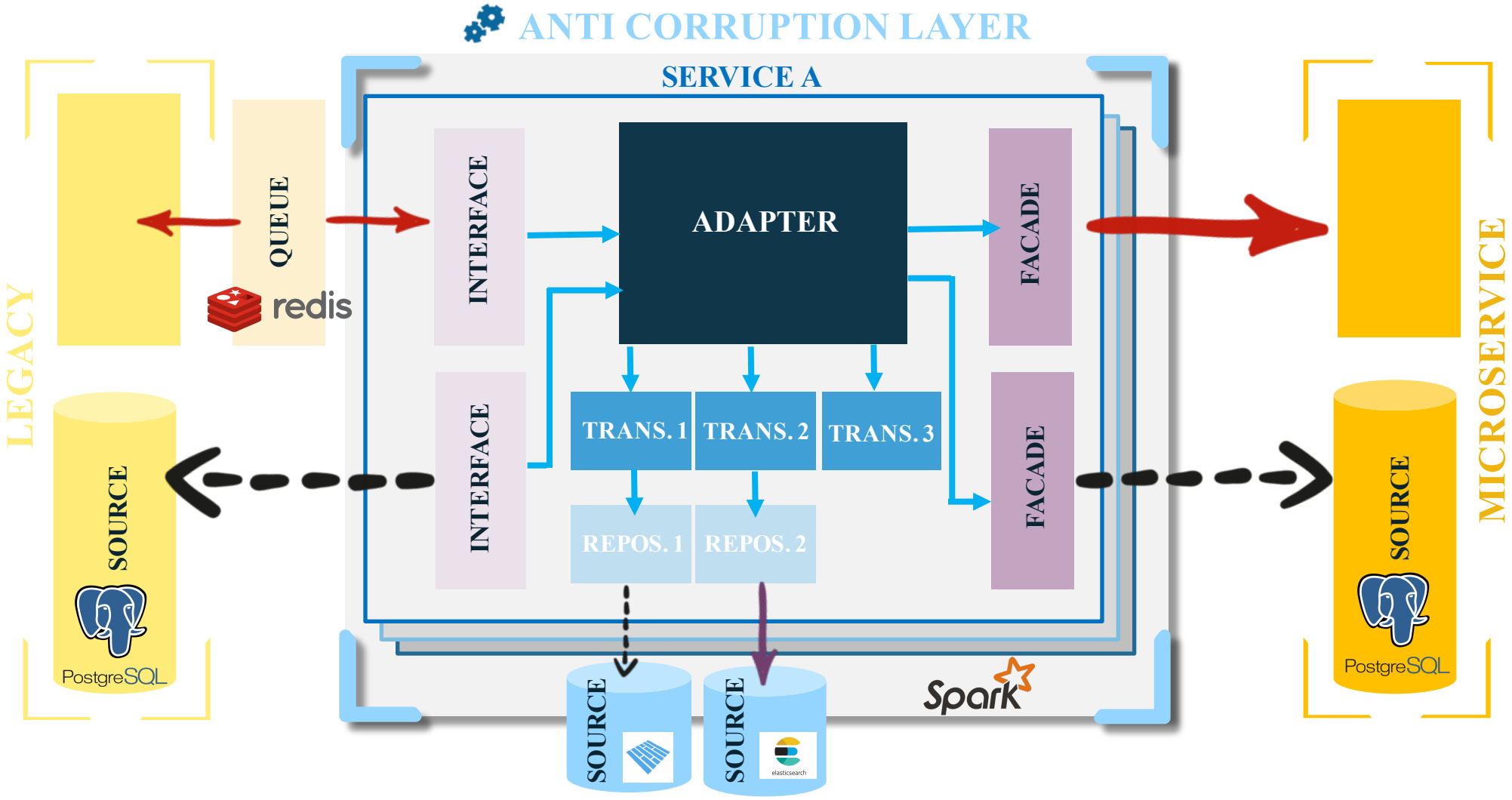
```scala
abstract class LegacyDomainAdapter(sparkSession: SparkSession) {

  import sparkSession.implicits._

  def initialize(inputDF: DataFrame): Dataset[TranslationStep] = {
    inputDF.as[LegacyAdministrativeDocument].map(TranslationStep(_))
  }

  def adapt(translationStepDs: Dataset[TranslationStep],
            translators: Seq[Translator]): Dataset[TranslationStep] = {

    val translations: Seq[Dataset[TranslationStep] => Dataset[TranslationStep]] =
      translators.map(_.translate)

    val pipeline: Dataset[TranslationStep] => Dataset[TranslationStep] =
      translations.reduceLeft(_ andThen _)

    translationStepDs.transform(pipeline)
  }
}
```

ANTI CORRUPTION LAYER



```scala
class InsuranceRepository(sparkSession: SparkSession) extends Serializable {
  def load(): Dataset[Insurance] = {
    import sparkSession.implicits._
    sparkSession.read.parquet("...").as[Insurance]
  }
}

class InsuranceResolver(sparkSession: SparkSession, insuranceRepository: InsuranceRepository)
  extends Translator with Serializable {

  override val description: String = "Translator responsible for adding insurance information"

  override val translate: Dataset[TranslationStep] => Dataset[TranslationStep] = {
    (stepDs) =>
      import sparkSession.implicits._
      val insuranceDs = insuranceRepository.load()
      stepDs.joinWith(insuranceDs, $"legacyAdministrativeDocument.insuranceId" === $"id", "left").map {
        case (step, null) =>
          step.copy(translationErrors = step.translationErrors :+ TranslationError(description))
        case (step, insurance) =>  step.copy(insurance = Option(insurance))
      }
  }
}
```


ANTI CORRUPTION LAYER
SERVICE A
LEGACY
MICROSERVICE
INTERFACE
ADAPTER
FACADE
TRANS. 1
TRANS. 2
TRANS. 3
REPOS. 1
REPOS. 2
SOURCE
PostgreSQL
SOURCE
SOURCE
elasticsearch
Spark
SOURCE
PostgreSQL


ANTI CORRUPTION LAYER
SERVICE A
LEGACY
QUEUE
redis
INTERFACE
ADAPTER
FACADE
INTERFACE
TRANS. 1
TRANS. 2
TRANS. 3
FACADE
REPOS. 1
REPOS. 2
SOURCE
PostgreSQL
SOURCE
SOURCE
elasticsearch
Spark
MICROSERVICE
SOURCE
PostgreSQL

Structured streaming interface

- Sources
  - File
  - Kafka
  - Kinesis
  - Socket (testing)
- Custom sources with *Source* interface
  - Must be able to replay arbitrary sequence of data
  - Designed to provide end-to-end exactly once guarantees
- Simpler approach than Dstream receivers

```scala
class RedisSource(sparkSession: SparkSession,
                  override val schema: StructType) extends Source {

  protected var lastOffsetCommitted: LongOffset = new LongOffset(0)

  override def getOffset: Option[Offset] = {
    val sortedScoreRDD = sparkSession.sparkContext
      .fromRedisZRangeByScoreWithScore("*", lastOffsetCommitted.offset.toDouble, Double.MaxValue)
      .map(_._2).sortBy(-_)

    if(sortedScoreRDD.count() == 0){
      None
    }  else {
      Some(LongOffset(sortedScoreRDD.first().toLong))
    }
  }

  override def getBatch(start: Option[Offset], end: Offset): DataFrame = {
    val startPos = start.flatMap(LongOffset.convert).getOrElse(LongOffset(0)).offset.toInt
    val endPos = LongOffset.convert(end).getOrElse(LongOffset(0)).offset.toInt

    import sparkSession.implicits._

    val jsonDataset: Dataset[String] = sparkSession
      .sparkContext.fromRedisZRangeByScore("*", startPos, endPos).toDS()

    sparkSession.sqlContext.read.schema(schema).json(jsonDataset)
  }

  override def commit(end: Offset) : Unit = {
    lastOffsetCommitted = LongOffset.convert(end).getOrElse(LongOffset(0))
  }

  override def stop() {}
}
```

```scala
class RedisSourceProvider extends StreamSourceProvider with DataSourceRegister {
  override def sourceSchema(
                             sqlContext: SQLContext,
                             schema: Option[StructType],
                             providerName: String,
                             parameters: Map[String, String]): (String, StructType) =
    (shortName(), schema.getOrElse(StructType(Nil)))

  override def createSource(
                             sqlContext: SQLContext,
                             metadataPath: String,
                             schema: Option[StructType],
                             providerName: String,
                             parameters: Map[String, String]): Source = {
    new RedisSource(sqlContext.sparkSession, schema.getOrElse(StructType(Nil)))
  }

  override def shortName(): String = "redis"
}
```

Structured streaming facade

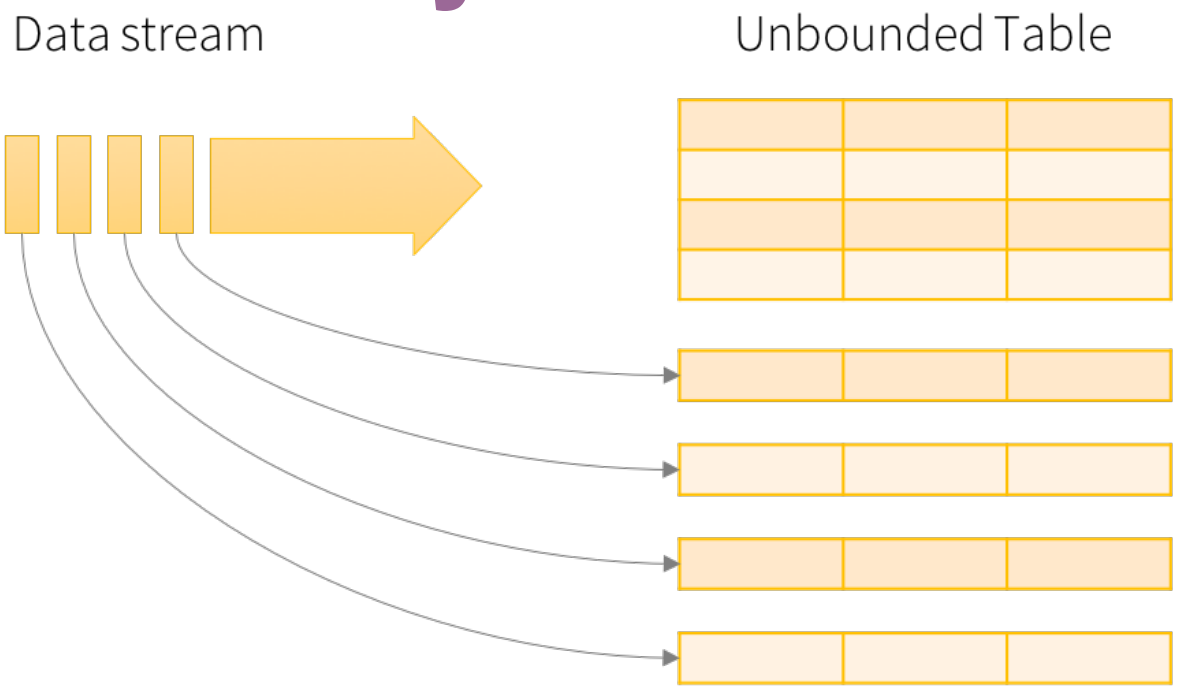
- Sinks
  - File
  - Console (testing)
  - Memory (testing)
- Custom sink with *ForeachWriter* interface
- Output modes
  - Append
  - Complete
  - Update

```scala
adminDocumentDs.writeStream.foreach(new ForeachWriter[AdministrativeDocument] {
  var httpRequest: HttpRequest = _
  var partitionId: Long  = _

  def open(partitionId: Long, version: Long): Boolean = {
    httpRequest = Http("http://httpbin.org/post")
    true
  }

  def process(record: AdministrativeDocument): Unit = {
    httpRequest.postData(record.toJson).asString
  }

  def close(errorOrNull: Throwable): Unit = {}
})
```

Takeaways

- Streaming represents unbounded DataFrames / Datasets
- Exactly same API but with some unsupported operations
- Different behavior for similar operations (*writeStream.format("json") vs write.json*)
- Simplifies testing

Takeaways



new data in the
data stream
=
new rows appended
to a unbounded table

Data stream as an unbounded table

```
val inputDF = sparkSession.read
  .format("jdbc")
  .option("url", "jdbc:protocol://host:port/database?user=user&password=password")
  .option("dbtable", "tableName")
  .option("driver" , "com.mysql.cj.jdbc.Driver")
  .load()

val insuranceResolver = new InsuranceResolver(sparkSession, new InsuranceRepository(sparkSession))
val branchDecoder = new BranchDecoder(sparkSession, new BranchRepository(sparkSession))

val adapter = new LegacyDomainBatchAdapter(sparkSession)
val translationStepDS = adapter.initialize(inputDF)
val adminDocDs: Dataset[AdministrativeDocument] = adapter
  .adapt(translationStepDS, Seq(insuranceResolver, branchDecoder))

adapter.write(adminDocDs)
```

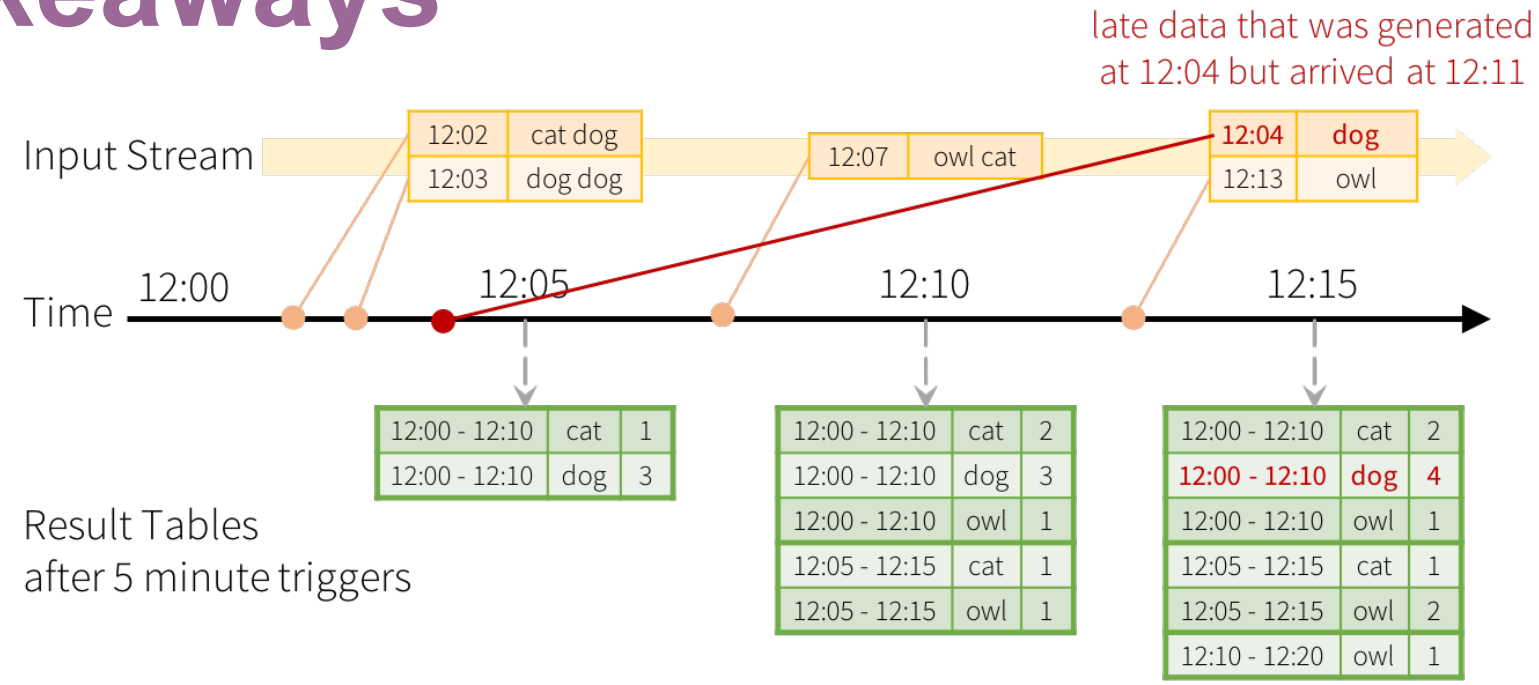
```
val inputDF = sparkSession.readStream
  .format("fr.alg.acl.source.RedisSourceProvider")
  .schema(StructType(StructField("branchCode", StringType) ::
    StructField("insuranceId", IntegerType) :: Nil))
  .load()

val insuranceResolver = new InsuranceResolver(sparkSession, new InsuranceRepository(sparkSession))
val branchDecoder = new BranchDecoder(sparkSession, new BranchRepository(sparkSession))

val adapter = new LegacyDomainBatchAdapter(sparkSession)
val translationStepDS = adapter.initialize(inputDF)
val adminDocDs: Dataset[AdministrativeDocument] = adapter
  .adapt(translationStepDS, Seq(insuranceResolver, branchDecoder))

adapter.write(adminDocDs)
```

```
val legacyDomainObject: LegacyAdministrativeDocument = LegacyAdministrativeDocument("1234", 4321)
val branch: Branch = Branch("1234", "branchName", "townName", "countryName")
val insurance: Insurance = Insurance(4321, "insuranceName")

val mockBranchRepository = mock[BranchRepository]
when(mockBranchRepository.load()).thenReturn(Seq(branch).toDS())
val mockInsuranceRepository = mock[InsuranceRepository]
when(mockInsuranceRepository.load()).thenReturn(Seq(insurance).toDS())

val inputDF = Seq(legacyDomainObject).toDF("branchCode", "insuranceId")

val adapter = new LegacyDomainBatchAdapter(spark)
val insuranceResolver = new InsuranceResolver(spark, mockInsuranceRepository)
val branchDecoder = new BranchDecoder(spark, mockBranchRepository)
val resultDs = adapter.adapt(adapter.initialize(inputDF), Seq(insuranceResolver, branchDecoder))
val expectedDomainDS = Seq(AdministrativeDocument(branch, insurance)).toDS()

assertDatasetEquals(expectedDomainDS, resultDs)
assert(inputDF.count() == resultDs.count())
```

Takeaways

- Recover from failure with checkpointing
- Use StreamingQueryListener to monitor queries
- Use .trigger and maxOffsets…
- Use watermarking to drop old records

Takeaways



Late data handling in
Windowed Grouped Aggregation

Conclusion

- Great framework for implementing ACLs
- Structured streaming is still young
  - Sources
  - Sinks
  - Monitoring
  - End to end guarantees
- Thinning line between batch and streaming

Resources

- Spark documentation and code
- Jacek Laskowski's *"Spark Structured Streaming"*
- Databricks blog
- Holden Karau's spark-testing-base


We’re hiring

attestation légale
ONCE FOR ALL

Thank you !

github.com/pbamba/spark-summit-acl

linkedin/in/pbamba

#EUstr6