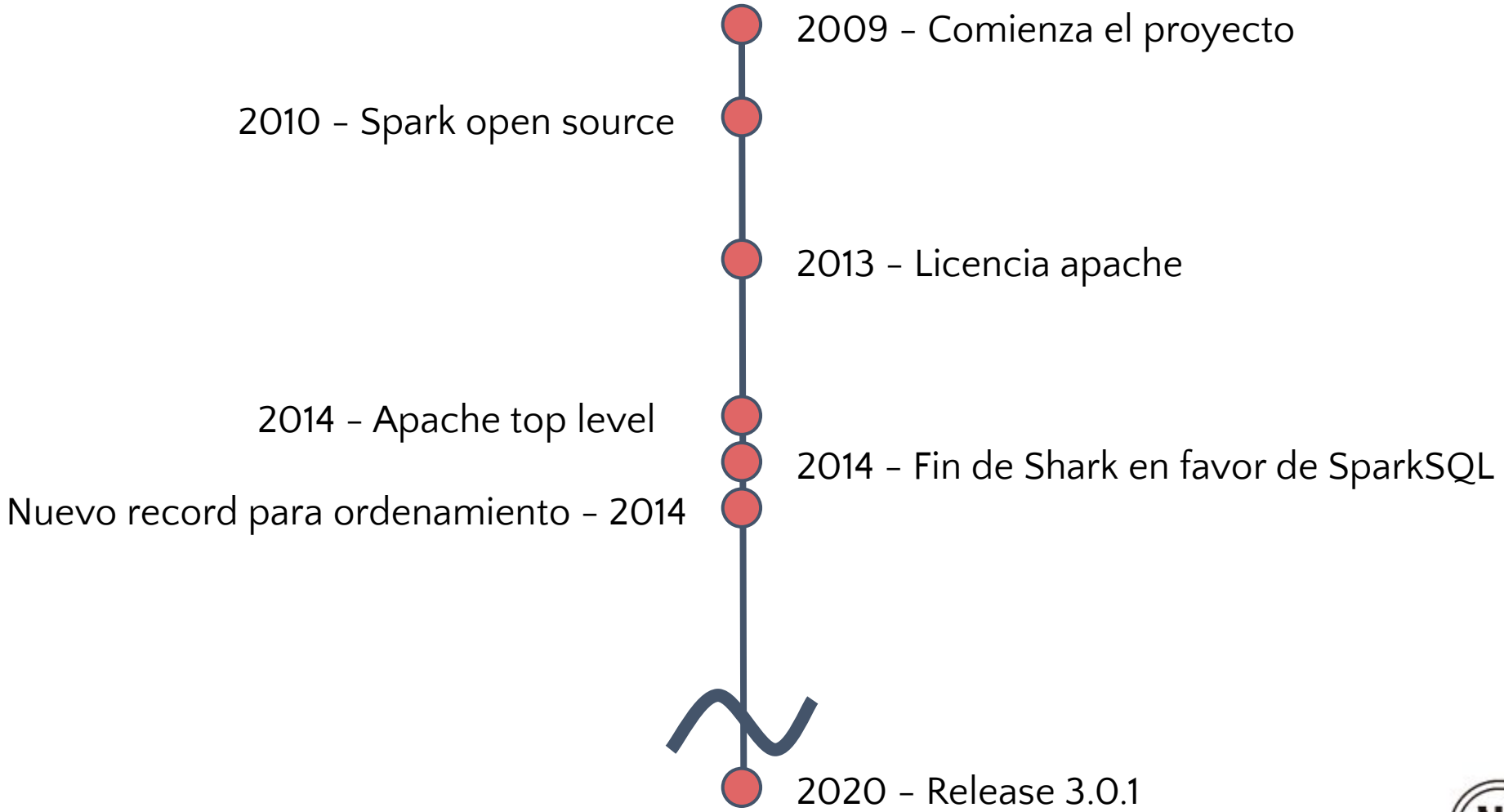
Spark - SQL

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Historia

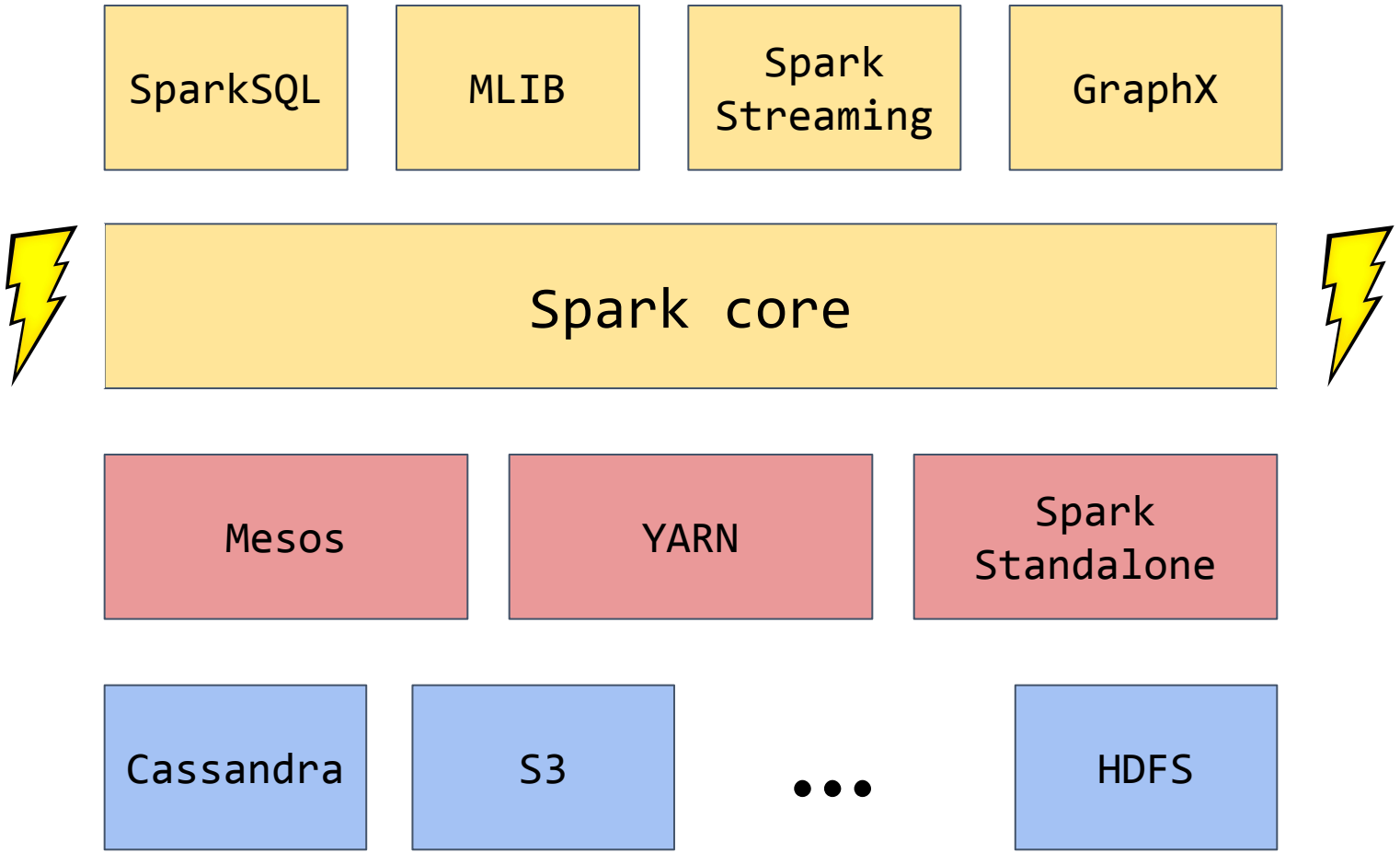


¿Qué es SparkSQL?

- Componente para trabajar con información **estructurada** o **semiestructurada**
- Relativamente nuevo (desarrollado a principios de 2014)
- Derivado de **Shark**. Un proyecto independiente que traducía sql a funciones de sparkcore con RDDs
- Ampliamente utilizado en conjunto con Hive para conseguir consultar tablas en HDFS y acceder a UDFs y UDAFs

Desde spark 2.0+ **sparkSQL** es la interfaz de facto para interactuar con spark. Cada vez más se tiene a querer ocultar los detalles del RDD arriba de una API más rica y amigable

Spark Stack



Spark – Ejemplo aplicación

```
$SPARK_HOME/bin/pyspark --master local[1]
```

```
rdd = spark.sparkContext.textFile("/titanic.csv")

valuesRDD = rdd.map(lambda x: x.split(",")).filter(lambda x: "Pclass" not in x)

byclassRDD = valuesRDD.map(lambda value: (value[1], 1))

countRDD = byclassRDD.reduceByKey(lambda old, new: old + new)

countRDD.collect()
```

Spark – Ejemplo aplicación

```
$SPARK_HOME/bin/pyspark --master local[1]
```

```
df = spark.read.option("header", "true").csv("/titanic.csv")

df.groupBy("Pclass").count().show()
```

Spark – Ejemplo aplicación

```
$SPARK_HOME/bin/pyspark --master local[1]
```

```
df = spark.read.option("header", "true").csv("/titanic.csv")

df.groupBy("Pclass").count().show()
```

```
from pyspark.sql import functions as F

df = spark.read.option("header", "true").csv("/titanic.csv")

df.groupBy("Pclass").agg(F.count(F.col("Pclass")).alias("cuenta")).show()
```

Spark – Ejemplo aplicación

```
$SPARK_HOME/bin/pyspark --master local[1]
```

```
df = spark.read.option("header", "true").csv("/titanic.csv")

df.createOrReplaceTempView("titanic")

spark.sql("""
SELECT Pclass, COUNT(*)
FROM titanic
GROUP BY Pclass
""").show()
```

SparkSQL

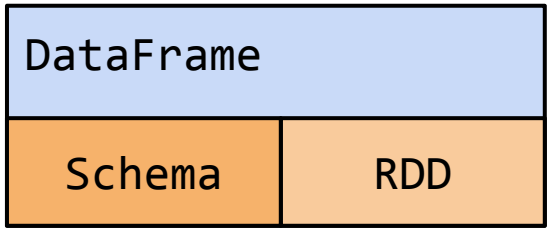
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

SparkSQL – DataFrame y Dataset

- Tanto el DataFrame como el Dataset son las estructuras base de sparkSQL.
- Pueden pensarse como una tabla distribuida.
- Poseen una referencia a la ubicación de la información, así como un **schema** de los datos que contienen.

Spark - DataFrame

DataFrame



- Estructura de datos para manejo de información estructurada
- Posee una API similar al lenguaje SQL
- Internamente contiene un RDD y un **schema** asociado a los datos
- Permite optimizar transformaciones y acciones al conocer esquemas de **origen** y **destino** de los datos

Spark - DataFrame

DataFrame

- Estructura de datos con para manejo de información estructurada
- Posee una API similar al lenguaje SQL
- Internamente contiene un RDD y un **schema** asociado a los datos
- Permite optimizar transformaciones y acciones al conocer esquemas de **origen** y **destino** de los datos

Spark - DataFrame

DataFrame

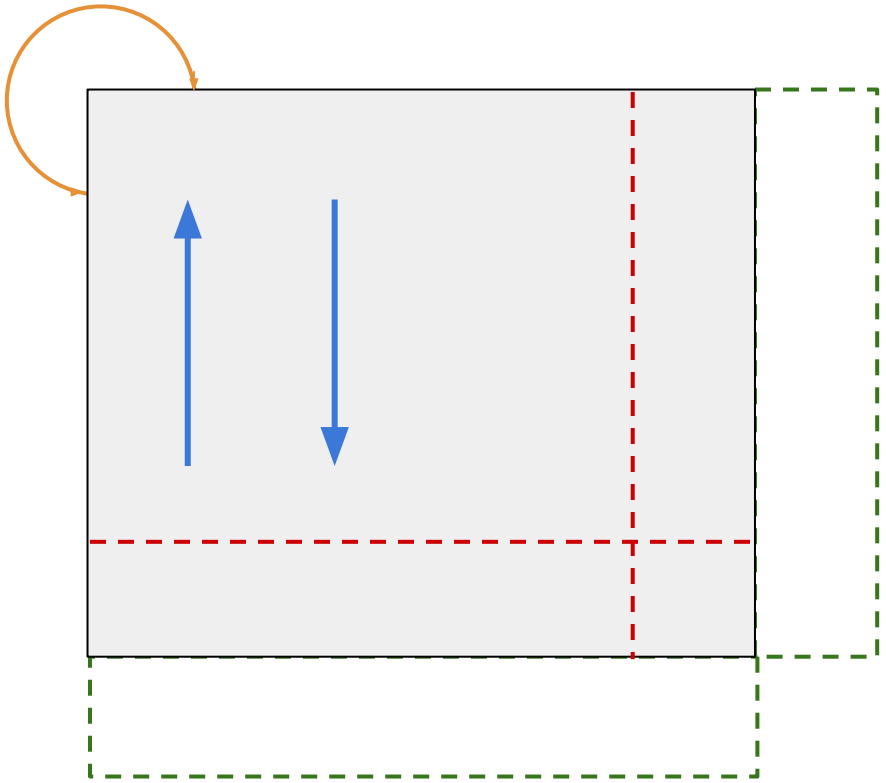
- Estructura de datos con para manejo de información estructurada
- Posee una API similar al lenguaje SQL
- Internamente contiene un RDD y un **schema** asociado a los datos
- Permite optimizar transformaciones y acciones al conocer esquemas de **origen** y **destino** de los datos

Las acciones más comunes son **`show()`**, **`count()`** y **`escribir en un storage externo`**

Spark - DataFrame

DataFrame

Que se puede hacer?



- Eliminar filas o columnas
- Convertir fila en columna o viceversa
- Añadir filas o columnas
- Ordenar filas

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark - DataFrame

Crear un DataFrame

Existen dos maneras de crear un DataFrame

- Leyendo de una fuente de datos estructurada o semiestructurada
  - Base de datos
  - Archivos (CSV, Parquet, ORC, Json)

- Paralelizando una colleción de **Rows**

Spark - DataFrame

Leyendo de una fuente de datos estructurada o semiestructurada:

Diferentes formatos de archivos

```
json_df = spark.read.json("/example.json")
csv_df = spark.read.option("header","true").csv("/titanic.csv")
parquet_df = spark.read.parquet("/example.parquet")
```

MySQL *($SPARK_HOME/pyspark --packages mysql:mysql-connector-java:<version>)*

```
mysql_df = spark.read\
              .format("jdbc")\
              .option("url", "jdbc:mysql://localhost:3306")\
              .option("dbtable", "<schema>.<table>")\
              .load()
```

Spark - DataFrame

Leyendo de una fuente de datos estructurada o semiestructurada:

Cassandra

```
$SPARK_HOME/pyspark \
    --conf "connection.host=<contacthostlist>" \
    --packages com.datastax.spark:spark-cassandra-connector_2.11:<version>
```

```
cassandra_df = spark.read\
               .format("org.apache.spark.sql.cassandra")\
               .options(table="<table>", keyspace="<keyspace>")
               .load()
```

Spark - DataFrame

Leyendo de una fuente de datos estructurada o semiestructurada:

MongoDB

```
$SPARK_HOME/pyspark \
    --packages com.mongodb.spark:mongo-spark-connector_2.11:<version>
```

```
mongo_df = spark.read\
               .format("com.mongodb.spark.sql.DefaultSource")\
               .options(uri="mongodb://<mongohost>:<port>/<db>.<collection>")
               .load()
```

Spark - DataFrame

Leer columnas *df = spark.read.option("header","true").csv("/titanic.csv")*

`select` & `selectExpr`

La operación `select` permite obtener las columnas que se pidan y operar sobre ellas, mientras que `selectExpr` permite utilizar una expresión para devolver una columna (permitiendo renombrar, utilizar udfs, expresiones lógicas, etc)

```
df.select("Pclass").show()
df.selectExpr("Pclass").show()
```

Spark - DataFrame

Leer columnas *df = spark.read.option("header","true").csv("/titanic.csv")*

`select` & `selectExpr`

La operación `select` permite obtener las columnas que se pidan y operar sobre ellas, mientras que `selectExpr` permite utilizar una expresión para devolver una columna (permitiendo renombrar, utilizar udfs, expresiones lógicas, etc)

```
df.select("Pclass").show()
df.selectExpr("Pclass").show()
```

```
from pyspark.sql import functions as F

df.select((F.col("Pclass") > '1').alias("has_money")).show()
df.selectExpr("Pclass > 1 as has_money").show()
```

Spark - DataFrame

Filtrar columnas *df = spark.read.option("header","true").csv("/titanic.csv")*

`filter` & `where`

De la misma manera que **`select`** y **`selectExpr`**, **`filter`** permite filtrar filas mediante **manipulación de columnas**, mientras que **`where`** recibe un string y evalúa los filtros usando SQL y el contexto de funciones definidas

```
from pyspark.sql import functions as F

df.filter((F.col("Pclass") != '1').show()
df.where("Pclass != 1").show()
```

Spark - DataFrame

Añadir filas y columnas *df = spark.read.option("header","true").csv("/titanic.csv")*

`withColumn`

La **transformación** `withColumn` permite añadir una nueva columna a un DataFrame, que puede ser una constante, o una combinación de otras columnas

```
from pyspark.sql import functions as F

df.withColumn("joven", F.expr("Age IS NOT NULL AND < '15'")).show()
```

`union`

```
df.union(df2).show()
```

Spark - DataFrame

```
df = spark.read.option("header","true").csv("/titanic.csv")
```

drop

```
df.drop("Pclass").show()
```

limit

```
df.limit(10).show()
```

orderBy

```
df.orderBy(F.desc("Fare"))
```

distinct

```
df.select("Pclass").distinct().show()
```

Spark - DataFrame - Aggregations

Las agregaciones permiten operar sobre un conjunto de filas. Dejando de lado los casos que son derivados de SQL (`COUNT, SUM, MAX, MIN, etc`), las agregaciones en spark permiten definir funciones custom, llamadas **UDAF** aunque solo son soportadas en scala por el momento.

Las agrupaciones posibles son:

- Agrupación total
- "**Group by**": especificando una o más columnas para agrupar y operar por ellas (uso clásico de SQL)
- "**roll up**": Permite brindar una o más columnas y alguna función de agregación para agrupar y sumarizar de manera jerárquica
- "**cube**": Realiza la combinación de todas las columnas
- "**window**": Define una ventana que espcifica qué filas se le van a pasar a la función de agregación

Spark - DataFrame - Aggregations

```
from pyspark.sql import functions as F

df = spark.read.option("header", "true").csv("/titanic.csv")

df.groupBy("Pclass").agg(F.count(F.col("Pclass")).alias("cuenta")).show()
```

Spark - DataFrame - Aggregations

```
from pyspark.sql import functions as F

df = spark.read.option("header", "true").csv("/titanic.csv")

df.groupBy("Pclass").agg(F.count(F.col("Pclass")).alias("cuenta")).show()
```

```
df = spark.read.option("header", "true").csv("/titanic.csv")

df.createOrReplaceTempView("titanic")

spark.sql("""
    SELECT Pclass, COUNT(*)
    FROM titanic
    GROUP BY Pclass
""").show()
```

Spark - DataFrame - UDF

De las siglas **User Defined Function**

Permite extender las funciones disponibles en sparkSQL con comportamiento arbitrario:

```
def add_random_number(number):
    random_number = 4 # Elegido con una tirada de dados. Garatizadamente aleatorio
    return number + random_number

from pyspark.sql.types import IntegerType

spark.udf.register("add_random_number", add_random_number, IntegerType())
```

Spark - DataFrame - UDF

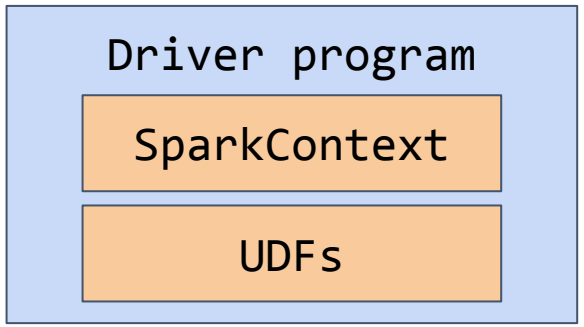
De las siglas **User Defined Function**

Permite extender las funciones disponibles en sparkSQL con comportamiento arbitrario:

```
def add_random_number(number):
    random_number = 4 # Elegido con una tirada de dados. Garatizadamente aleatorio
    return number + random_number

from pyspark.sql.types import IntegerType

spark.udf.register("add_random_number", add_random_number, IntegerType())
```



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark - DataFrame - UDF

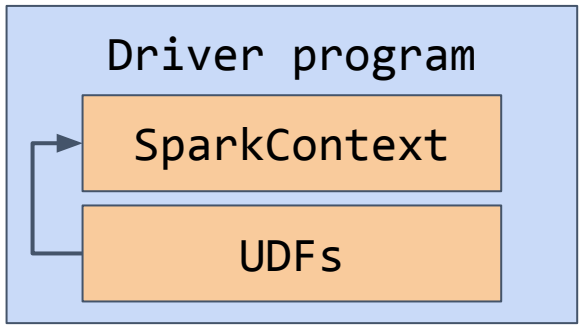
De las siglas **User Defined Function**

Permite extender las funciones disponibles en sparkSQL con comportamiento arbitrario:

```
def add_random_number(number):
    random_number = 4 # Elegido con una tirada de dados. Garatizadamente aleatorio
    return number + random_number

from pyspark.sql.types import IntegerType

spark.udf.register("add_random_number", add_random_number, IntegerType())
```



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark - DataFrame - UDF

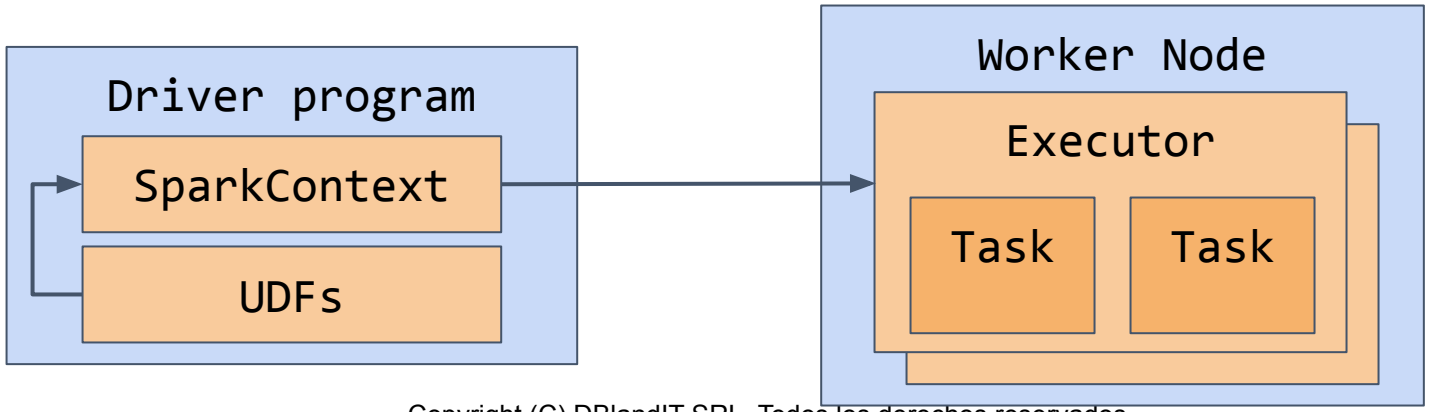
De las siglas **User Defined Function**

Permite extender las funciones disponibles en sparkSQL con comportamiento arbitrario:

```
def add_random_number(number):
    random_number = 4 # Elegido con una tirada de dados. Garatizadamente aleatorio
    return number + random_number

from pyspark.sql.types import IntegerType

spark.udf.register("add_random_number", add_random_number, IntegerType())
```



Spark - DataFrame - UDF

De las siglas **User Defined Function**

```
df = spark.read.option("header", "true").csv("/titanic.csv")

df.createOrReplaceTempView("titanic")

spark.sql("""
    SELECT add_random_number(Fare)
    FROM titanic
    GROUP BY Pclass
""").show()
```

Spark - DataFrame - UDF

De las siglas **User Defined Function**

```
df = spark.read.option("header", "true").csv("/titanic.csv")

df.createOrReplaceTempView("titanic")

spark.sql("""
    SELECT add_random_number(Fare)
    FROM titanic
""").show()
```

```
df.selectExpr("add_random_number(Fare)").show()
```

SparkSQL – Ejecución

SparkSQL

Unos de los principales motores de la popularidad y el desarrollo de sparkSQL consiste en las optimizaciones que realiza para realizar procesamiento.

Los pasos de ejecución de un job de sparkSQL son:

- Escribir código mediante DataFrames/Datasets/SQL
- Conversión del código a un *plan lógico*
- Transformación *plan lógico* a *plan físico*
- Deploy del plan físico de ejecución al cluster spark y ejecutar

SparkSQL

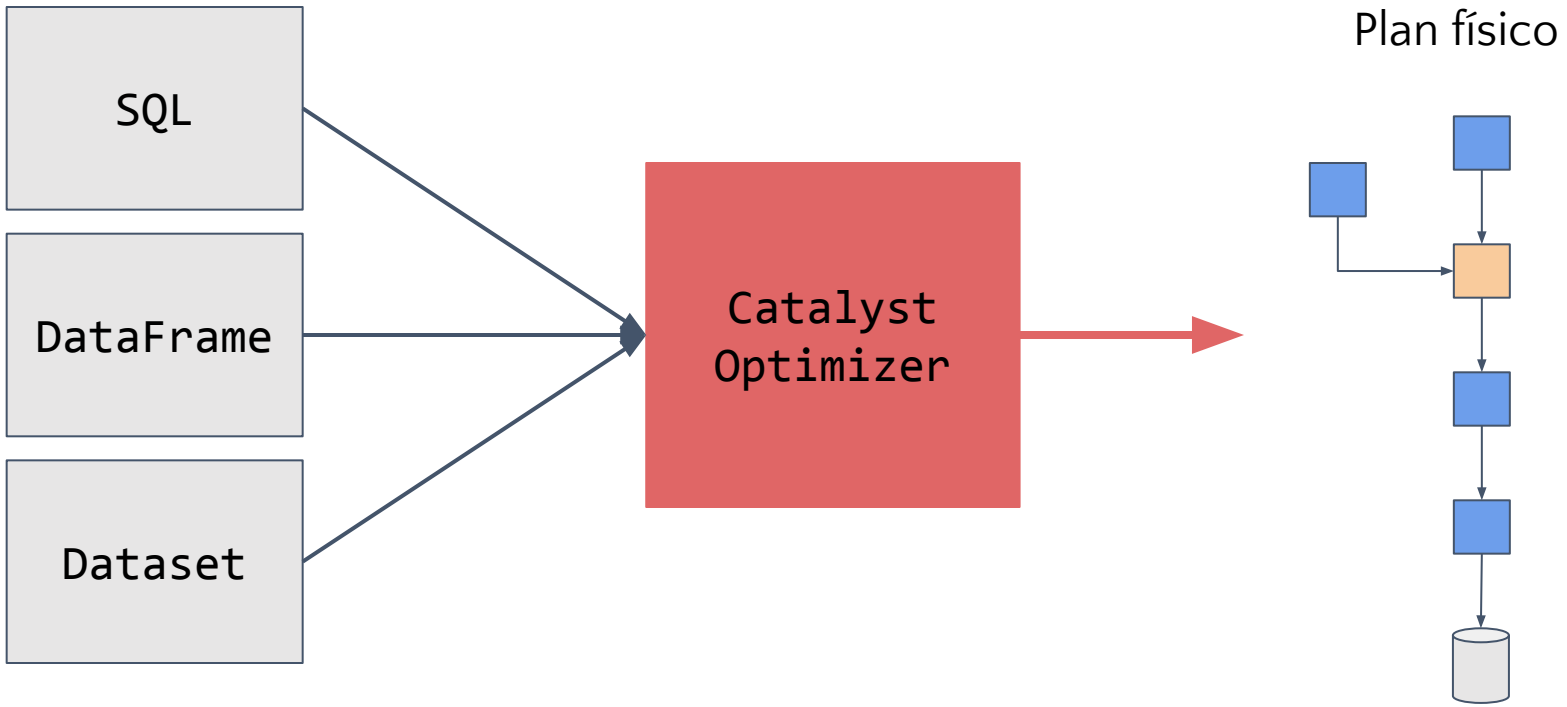
Unos de los principales motores de la popularidad y el desarrollo de sparkSQL consiste en las optimizaciones que realiza para realizar procesamiento.

Los pasos de ejecución de un job de sparkSQL son:

- Escribir código mediante DataFrames/Datasets/SQL
- Conversión del código a un *plan lógico*
- Transformación *plan lógico* a *plan físico*
- Deploy del plan físico de ejecución al cluster spark y ejecutar

Este conjunto de operaciones es realizada por un **Catalyst Optimizer**

SparkSQL



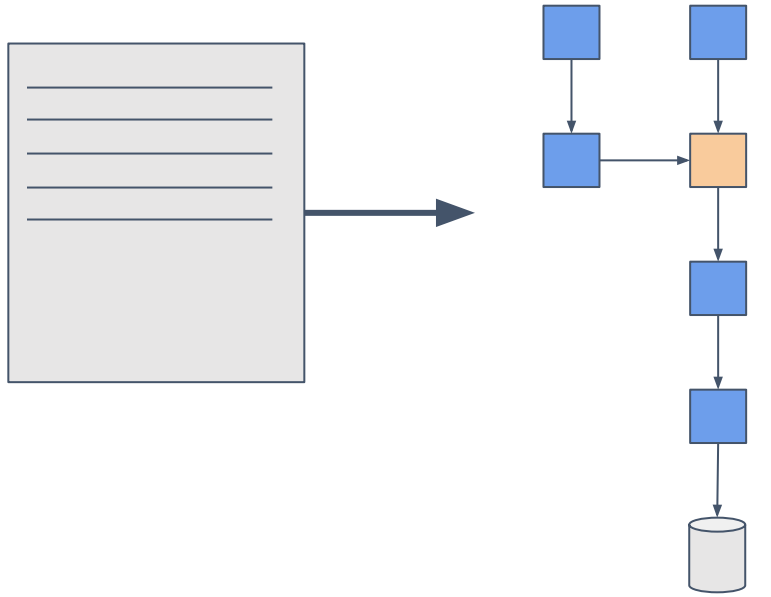
SparkSQL – Logical plan

User code

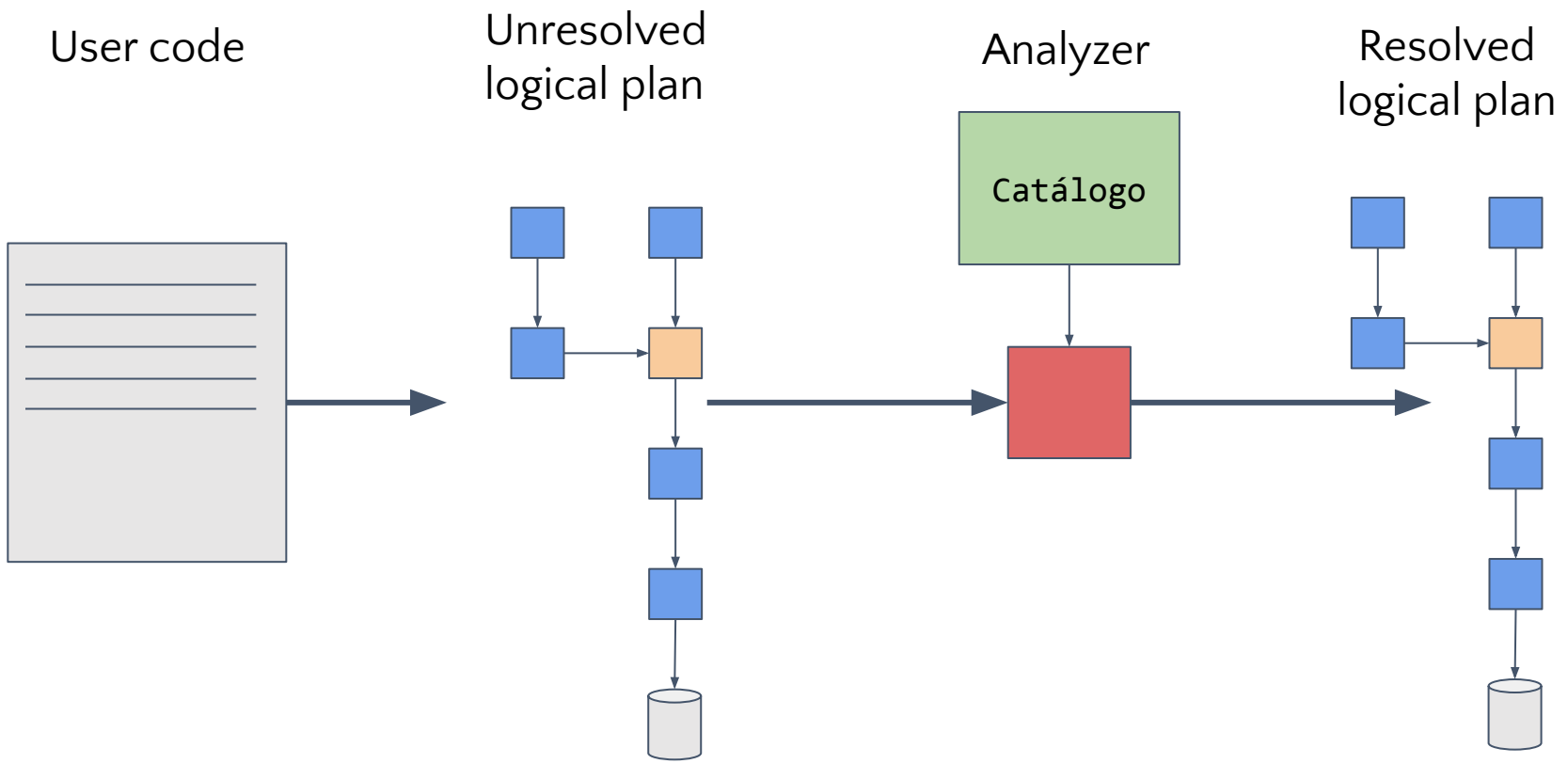
SparkSQL – Logical plan

User code

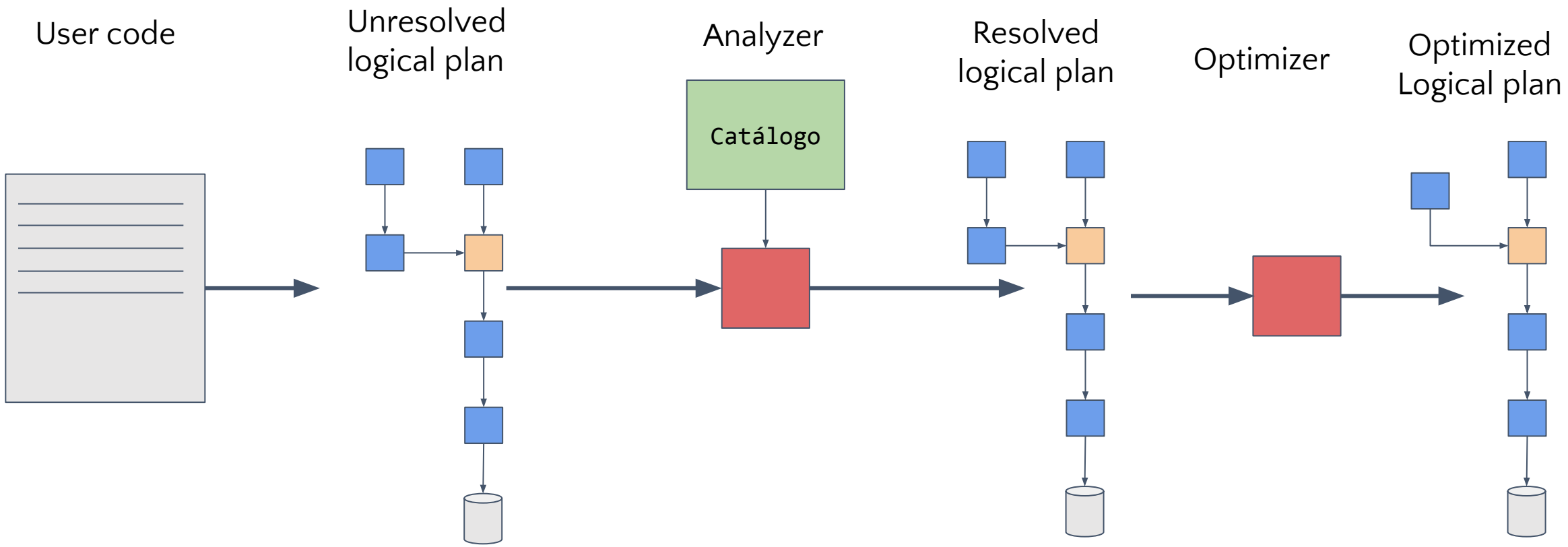
Unresolved logical plan



SparkSQL – Logical plan



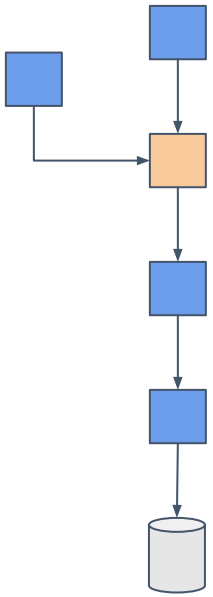
SparkSQL – Logical plan



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

SparkSQL – Logical plan

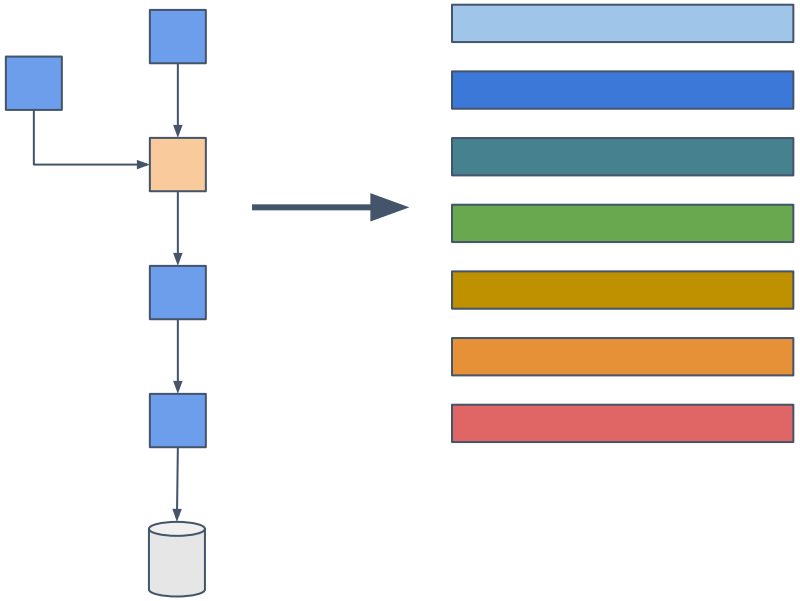
Optimized
Logical plan



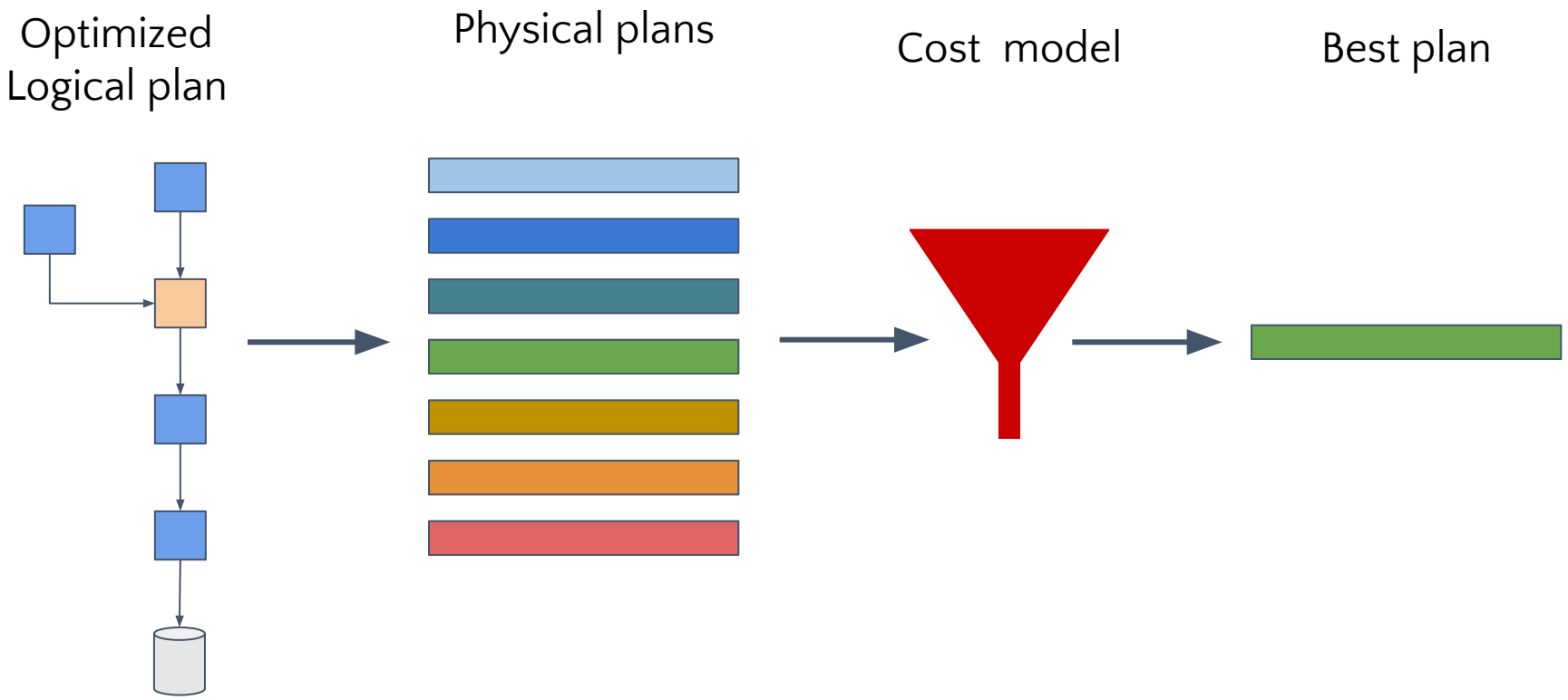
SparkSQL – Logical plan

Optimized
Logical plan

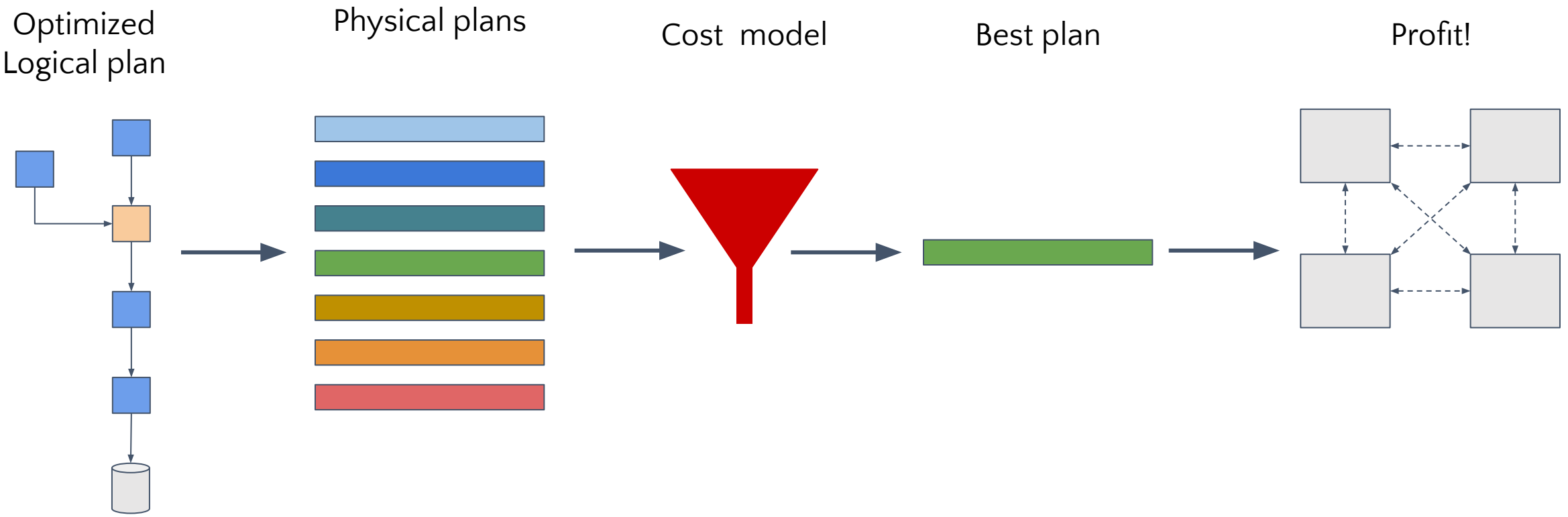
Physical plans



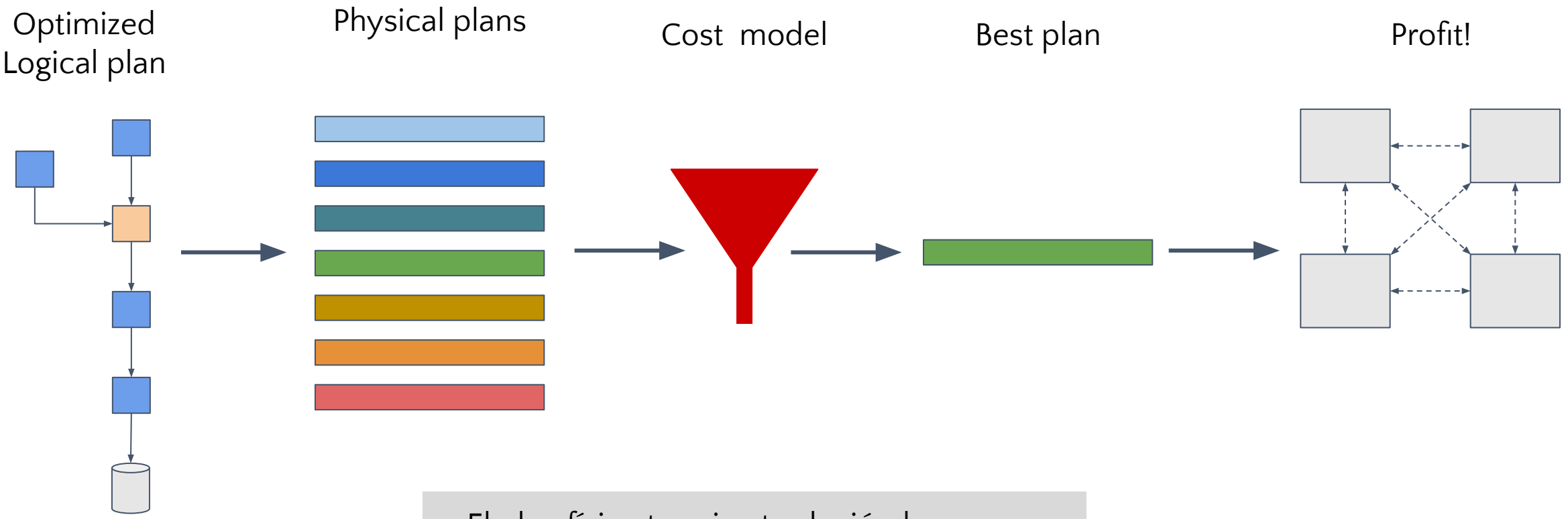
SparkSQL – Logical plan



SparkSQL – Logical plan

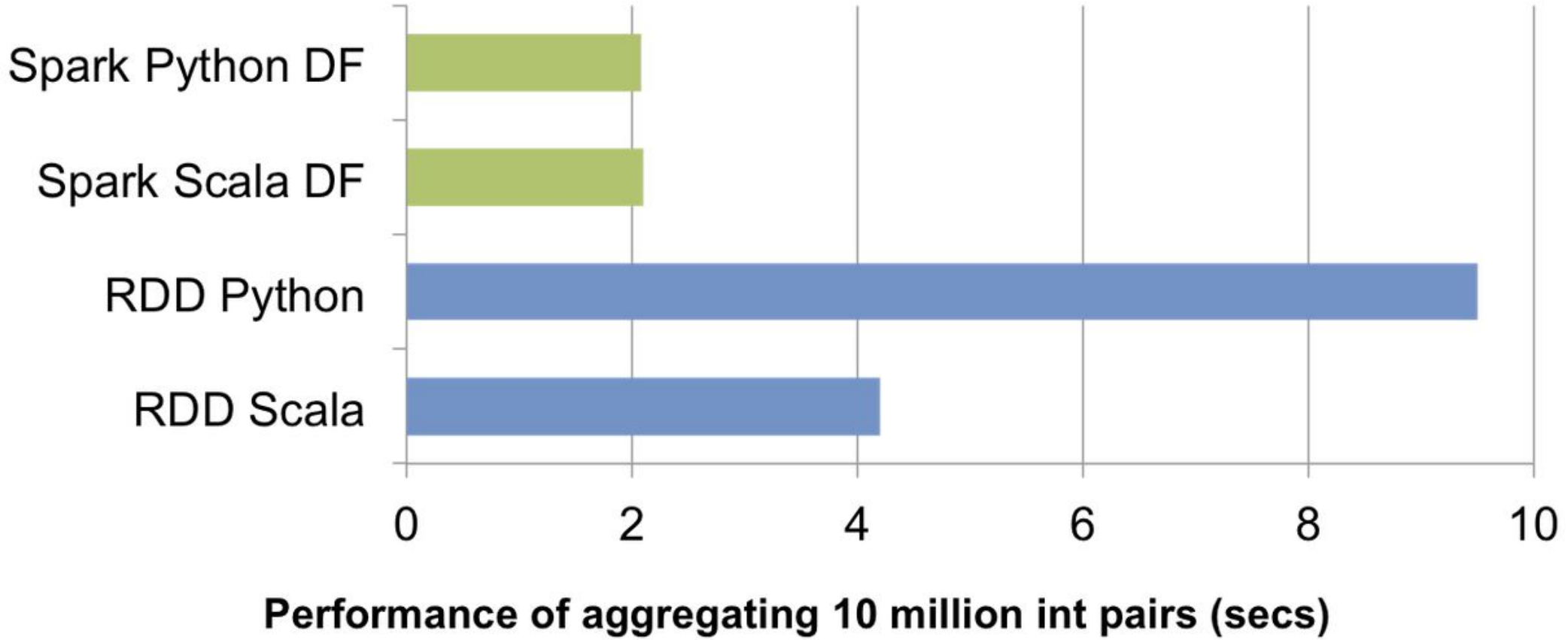


SparkSQL – Logical plan

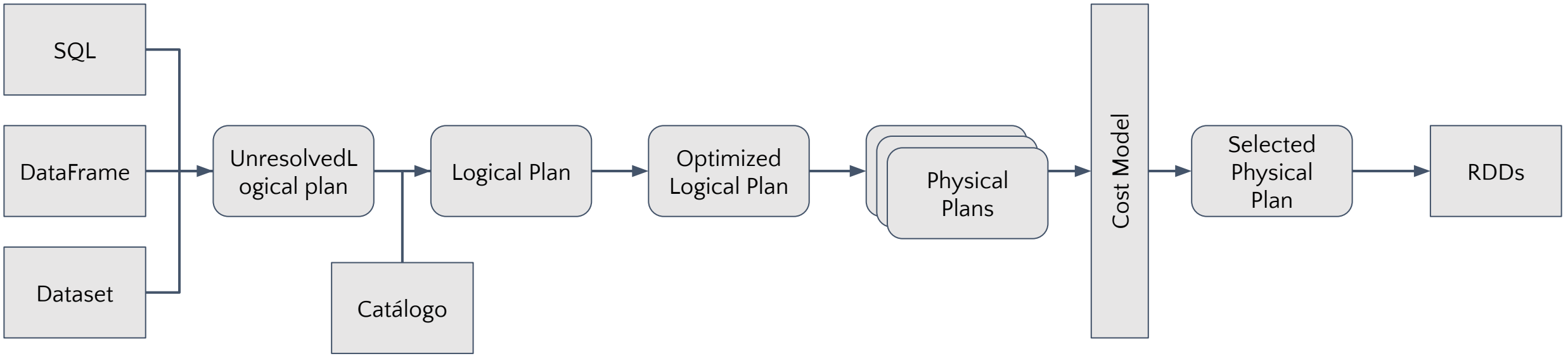


El plan físico termina traduciéndose a una serie de operaciones y transformaciones sobre **RDDs** por eso es que se dice que spark funciona como un **compilador**

SparkSQL – Logical plan



SparkSQL



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

SparkSQL – Joins

El optimizador por costos en principio solo funcionaba para las operaciones de Join.
Existen tres casos diferentes al momento de joinar:

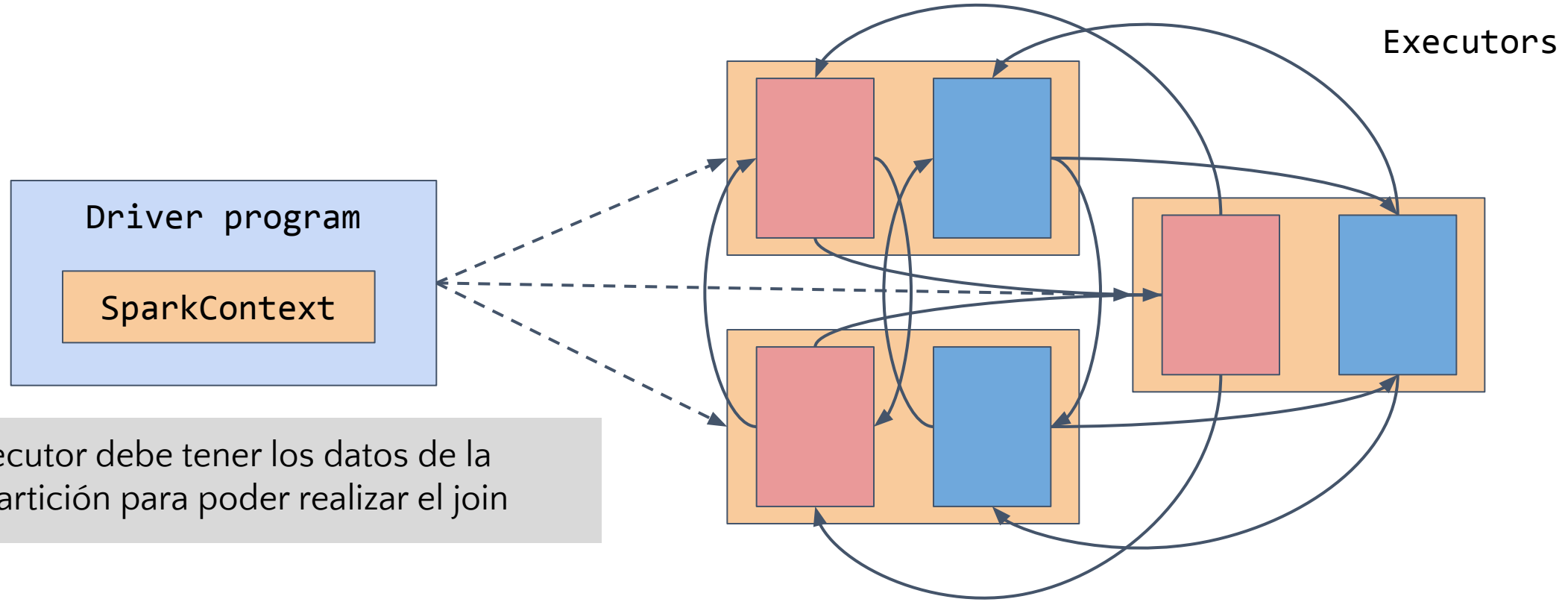
- Tabla grande vs Tabla grande
- Tabla grande vs Tabla chica
- Tabla chica vs Tabla chica

A grandes rasgos existen dos estrategias para realizar joins de tablas en spark:

- **Shuffle join**
- **Broadcast Join**

SparkSQL – Tabla grande vs Tabla grande

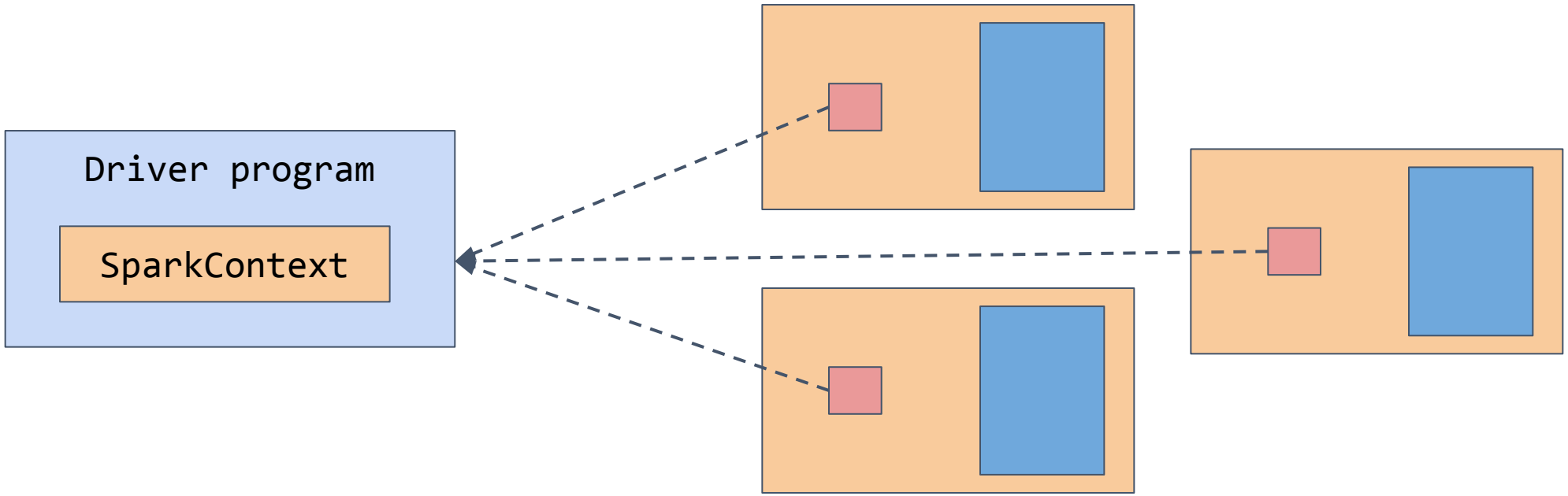
En el caso de tabla grande vs tabla grande, es necesario realizar un **shuffle join** *(en el 99% de los casos)*



Cada executor debe tener los datos de la misma partición para poder realizar el join

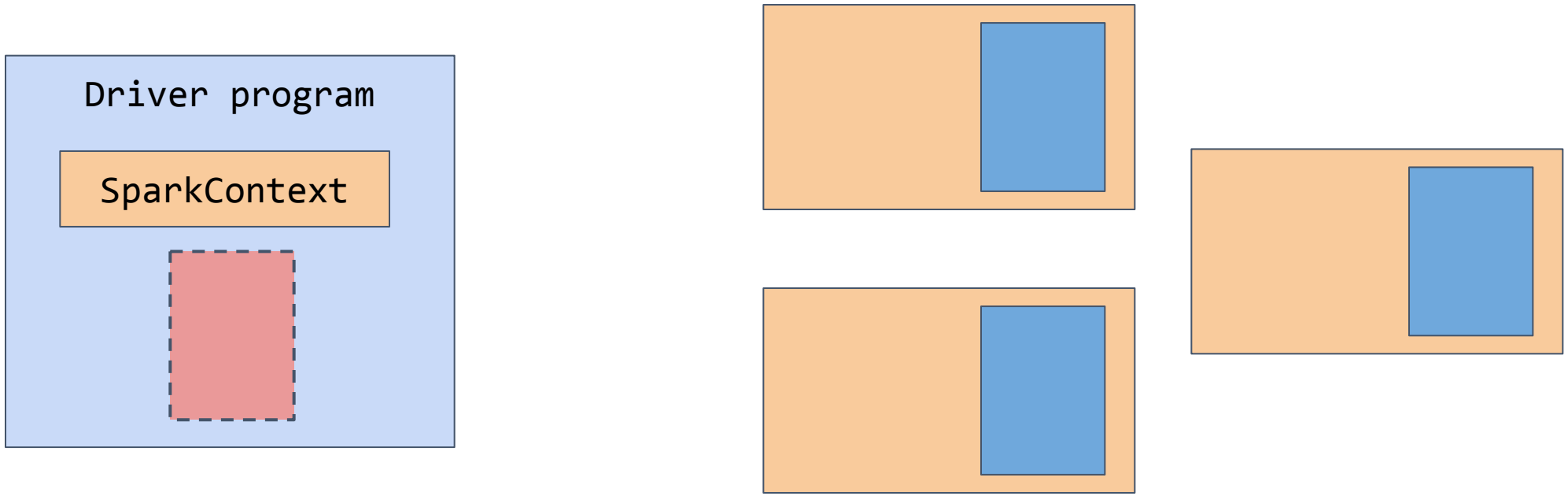
SparkSQL – Tabla grande vs Tabla chica

En el caso de tabla grande vs tabla chica, se realizará un **broadcast join**



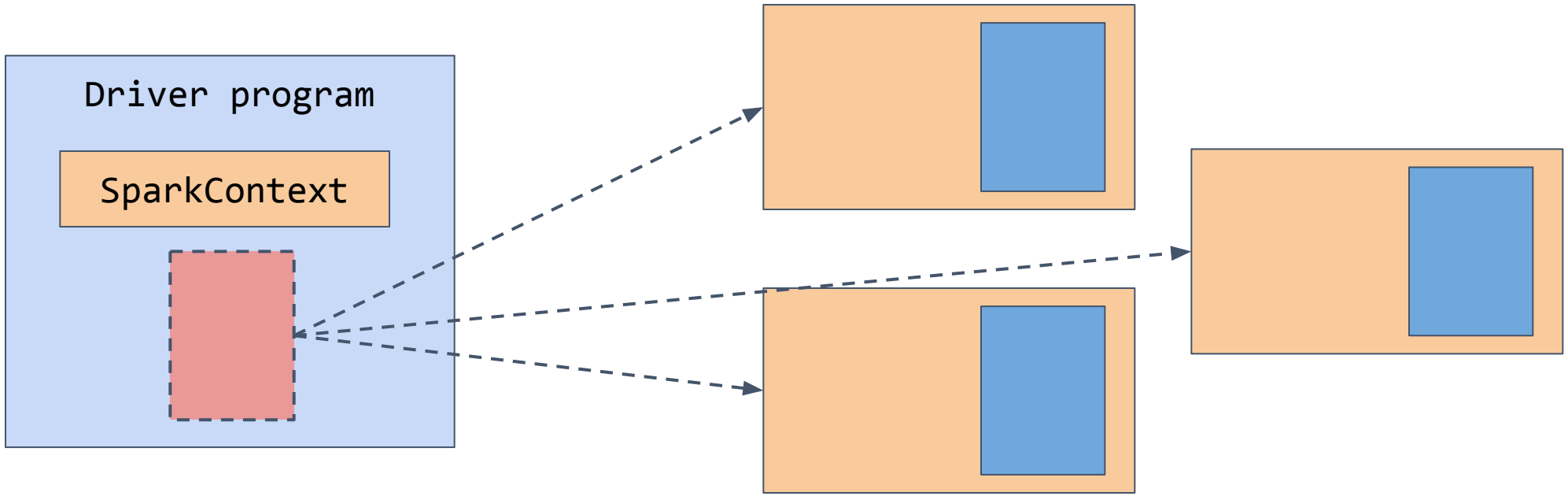
SparkSQL – Tabla grande vs Tabla chica

En el caso de tabla grande vs tabla chica, se realizará un **broadcast join**



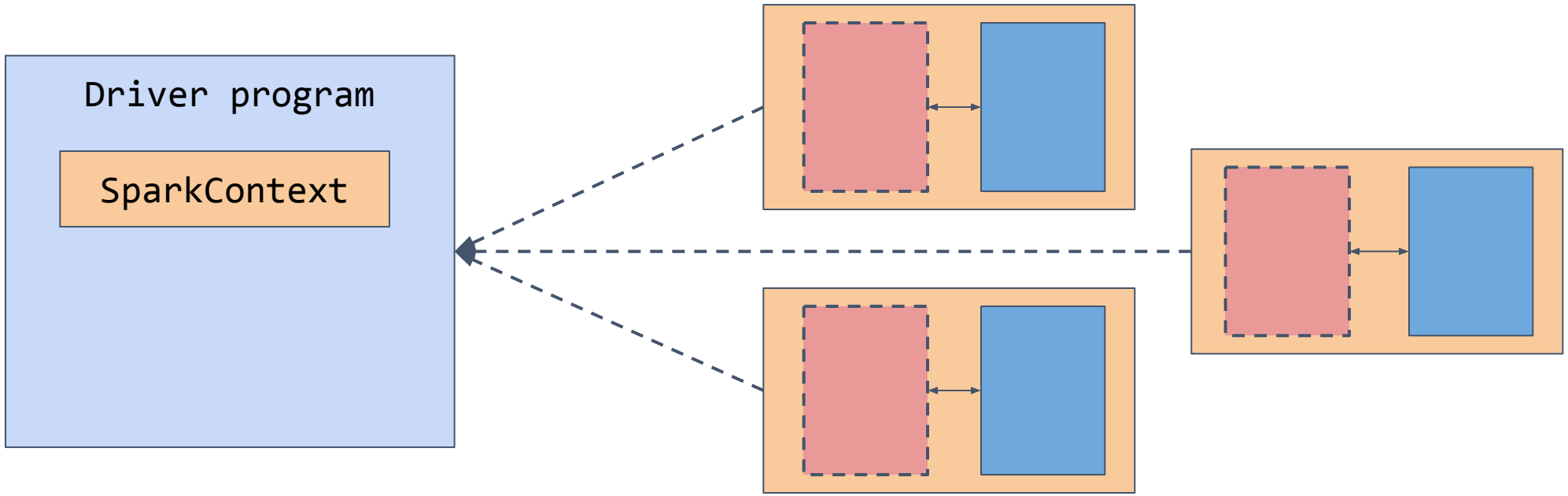
SparkSQL – Tabla grande vs Tabla chica

En el caso de tabla grande vs tabla chica, se realizará un **broadcast join**



SparkSQL – Tabla grande vs Tabla chica

En el caso de tabla grande vs tabla chica, se realizará un **broadcast join**



Demostración

ETL

Extract Transform & Load

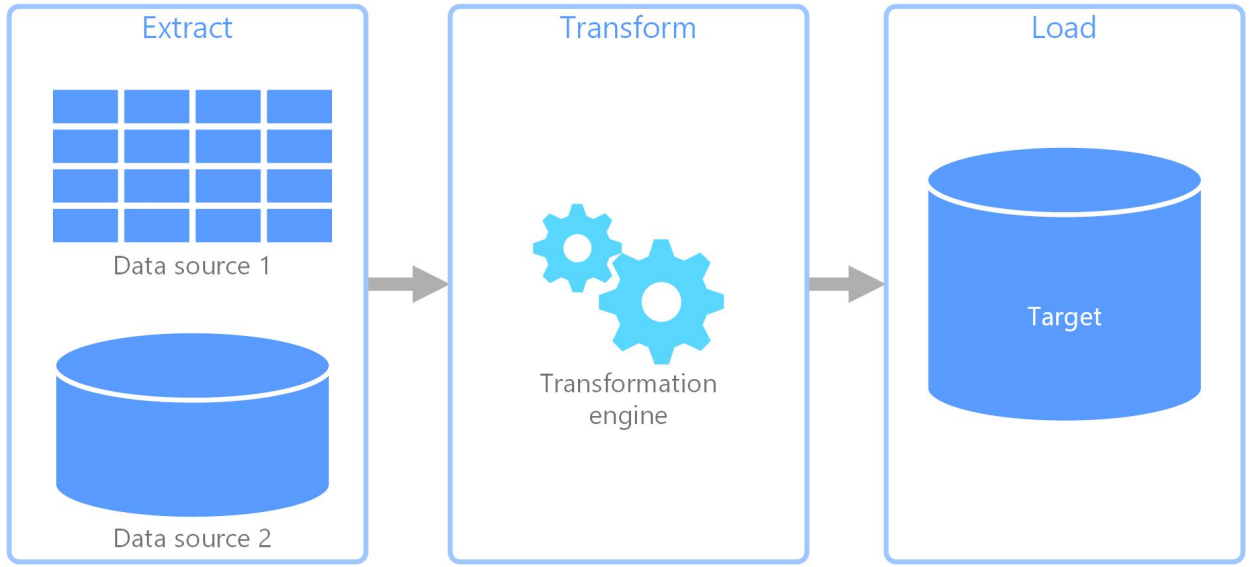
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Definición

Extract: Extracción y validación de datos de diversas fuentes homo y heterogéneas.

Transform: Transformación de datos con el objetivo de adecuar el esquema y la calidad de datos a los del destino

Load: Almacenamiento de datos en el destino cumpliendo con los requerimientos de este



Objetivo: Periodicamente mover, filtrar y organizar datos para que puedan ser analizados en un futuro.

E - Extracción

Pasos necesarios para la extracción de datos:

- Leer una fuente de datos:
  - *spark.read.format('**xxx**').load(**filepath**)*
- *Definir un esquema de datos*


```
from pyspark.sql.types import StructType, StructField, IntegerType
schema = StructType([StructField('name', StringType(), nullable=False),
                     StructField('lastname', StringType()),
                     StructField('age', IntegerType())
                    ])
rdd.toDF(schema).show()
```


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

T - Transformación

Pasos necesarios para la transformación de datos:

- Data cleansing
  - Validación de datos:
    - *df.filter(func)*
    - *when(cond, result).otherwise(result)*
    - *expr("SQL expression")*
    - *df.drop_duplicates(cols)*
  - Limpieza de nulos:
    - *df.fillna({ col : value })*
  -

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

T - Transformación

Pasos necesarios para la transformación de datos:

- Transformación de columnas
  - Selección de columnas
    - *df[cols] df.col sql.select(cols)*
  - Codificación/decodificación
    - df.withColumn(col, F.when(cond, value).otherwise(value))
  - Nuevas columnas
    - df.withColumn("total", df.cant * df.precio)
  - Ordenamientos
    - df.orderBy(cols, ascending=True) df.sort(cols, ascending)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

T - Transformación

Pasos necesarios para la transformación de datos:

- Transformación de columnas
  - Union de multiples fuentes (join)
    - *df.join(other, on, how)*
  - *Agregaciones*
    - *df.groupBy(col).agg(func)* *df.agg({'col': 'func'})*
    - *df.cube()*
    - *df.rollup()*
    - *df.over(window) window = Window.partitionBy(cols).orderBy(cols)*

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

T - Transformación

Pasos necesarios para la transformación de datos:

- Transformación de columnas
  - Clave sustituta (surrogate key)
    - Timestamp
      - *F.current_timestamp()*
      - *F.unix_timestamp(time_string, format)*
    - Counter
      - *F.monotonically_increasing_id()*
      - *F.row_number() F.rank(), F.dense_rank()*

T - Transformación

Pasos necesarios para la transformación de datos:

- Transformación de columnas
  - Transposicion / Pivot
    - *df.groupBy(cols).pivot(col).agg(func)*


```
df.groupby(df.Survived, df.Pclass).pivot("sex").avg("Age", "Fare").show()
```

```
+--------+------+------------------+------------------+------------------+------------------+
|Survived|Pclass|   female_avg(Age)|  female_avg(Fare)|     male_avg(Age)|    male_avg(Fare)|
+--------+------+------------------+------------------+------------------+------------------+
|       0|     3|              21.5|15.367366666666664| 18.03846153846154|          16.31625|
|       1|     1|              43.6| 75.69721666666668|              28.0|              35.5|
|       1|     3|20.428571428571427|11.354554545454546|              null|11.237499999999999|
|       0|     1|              null|              null|41.857142857142854| 81.24113750000001|
|       1|     2|22.285714285714285|              23.2|              34.0|              13.0|
|       0|     2|              27.0|              21.0|              50.5|             18.25|
+--------+------+------------------+------------------+------------------+------------------+
```


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

T - Transformación

Optimización - Performance Tuning

- Cacheo de tablas
  - *spark.catalog.cacheTable("tabla")* *df.cache()*
  - *spark.catalog.uncacheTable("tabla")*
- *Configuración de variables*
  - *spark.sql.inMemoryColumnarStorage.compressed*
  - *spark.sql.inMemoryColumnarStorage.batchSize*


```
SparkSession.builder.appName("Aplicacion")\
                .config("spark.sql.inMemoryColumnarStorage.compressed",
"true")\
                .config("Spark.sql.inMemoryColumnarStorage.batchSize",
          "10000")\
S       .getOrCreate()
```


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

T - Transformación

Optimización - Performance Tuning

- *Configuración de variables*
  - *spark.sql.files.maxPartitionBytes*
  - *spark.sql.autoBroadcastJoinThreshold*
  - *Spark.sql.shuffle.partitions*
- *Broadcast hint*


```
df1.join(broadcast(df2), "id")

df1.createOrReplaceTempView("df1")
df2.createOrReplaceTempView("df2")
spark.sql( "SELECT /*+ MAPJOIN(df2) */ * FROM df1 JOIN df2 ON df1.id = df2.id" )
```

L - Cargar (Load)

Almacenamiento de datos:

- *Funcion generica*
  - *df.write.format(format).save(path,)*
  - *df.saveAsTable(tableName)*
  - *df.write.jdbc(url, table, mode, properties)*
- *Modificadores*
  - *df.write.partitionBy(cols)*
  - *df.write.bucketBy(numBuckets, cols)*
  - *df.write.sortBy(cols)*

L – Cargar (Load)

Almacenamiento de datos:

- *Conexión con Hive*
  - *config("spark.sql.warehouse.dir", warehouse_location)*

```
spark = SparkSession \
    .builder \
    .appName("HiveTest") \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()

spark.sql("CREATE TABLE IF NOT EXISTS t1 (key INT, value STRING) USING hive")
spark.sql("LOAD DATA LOCAL INPATH 'examples/kv1.txt' INTO TABLE t1")
sqlDF = spark.sql("SELECT key, value FROM t1")
```

L – Cargar (Load)

Almacenamiento de datos:

- *Spark como SQL engine*

```
./sbin/start-thriftserver.sh \
  --hiveconf hive.server2.thrift.port=<listening-port> \
  --hiveconf hive.server2.thrift.bind.host=<listening-host> \
  --master <master-uri>
```

- *Ejemplo de conexión*

```
./bin/beeline

>> !connect jdbc:hive2://localhost:10000
```

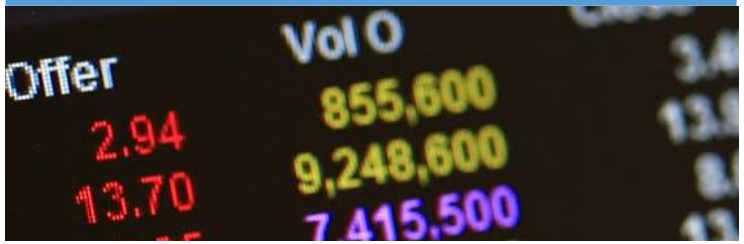
Business Intelligence

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Tipos de Sistemas

SI
Sistemas de Información

OLTP
On-Line Transaction Processing



OLAP
On-Line Analysis Processing

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

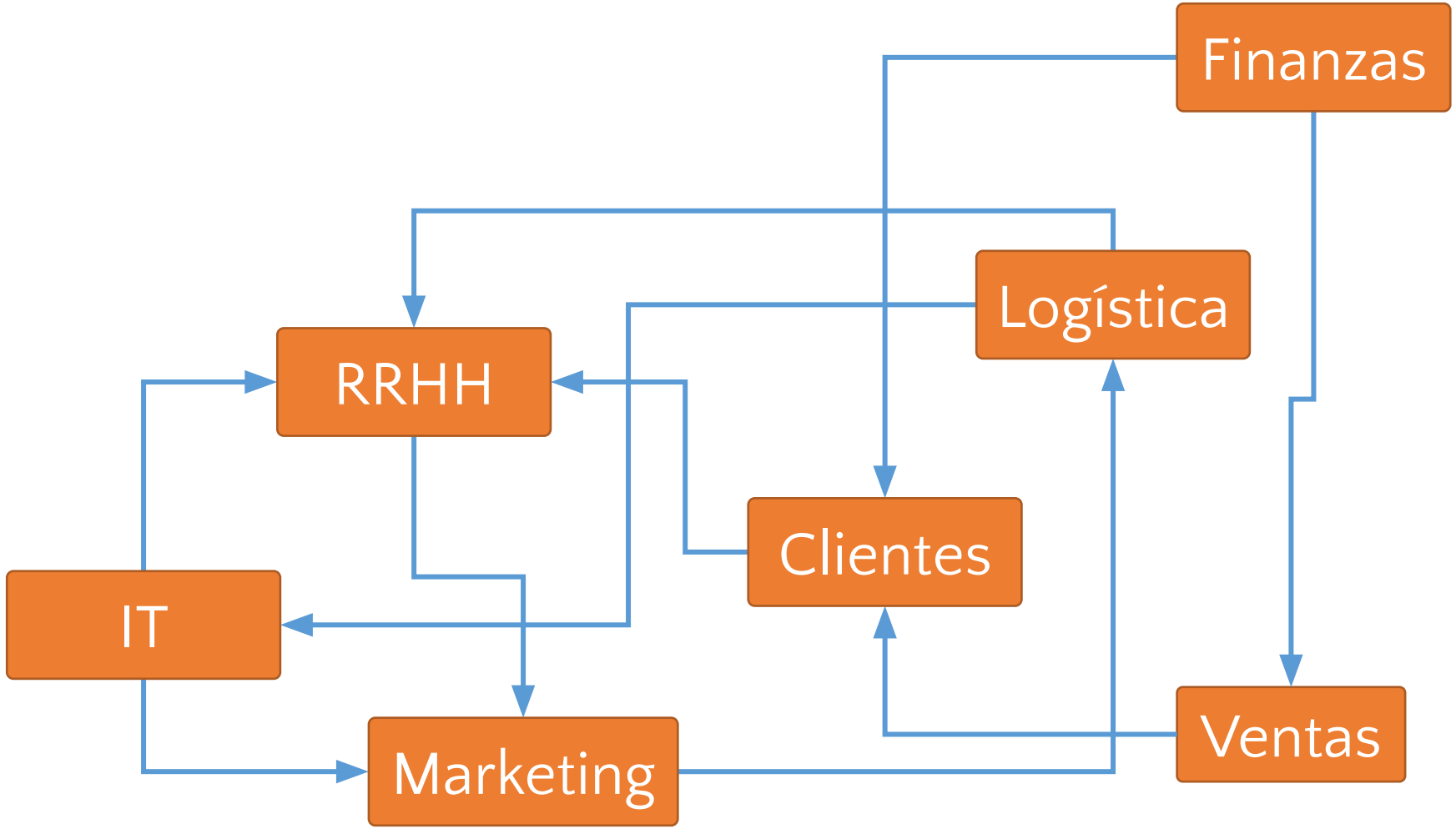
Sistemas OLTP vs OLAP

OLTP	OLAP
Alineados por aplicación o funcionalidad. Poca integración, fronteras tecnológicas definidas.	Integrados. Abarcan varios procesos de negocio, u cruzan información entre si.
Actualización **online.**	Actualización **batch.**
Acceso transaccional *(créate, read, update, delete).*	Acceso de solo lectura (*read*).
Datos recientes o de periodos de tiempos cortos.	Datos Históricos.
Información detallada y **no** redundante.	Información agregada y redundante.
Favorecer la **Operación Transaccional del día a día.**	Favorecer el **Análisis.**

Datos Vs. Información Vs. Conocimiento

DATO	Es un **valor**... Por ejemplo, “500 unidades”.
INFORMACION	Tiene un **contexto**... “Las ventas del mayo fueron de 500 unidades”
CONOCIMIENTO	Se obtiene mediante el **análisis** de la información... “Mayo es el mes más alto en ventas”

Como obtener la información y conocimientos necesarios?



Business Intelligence

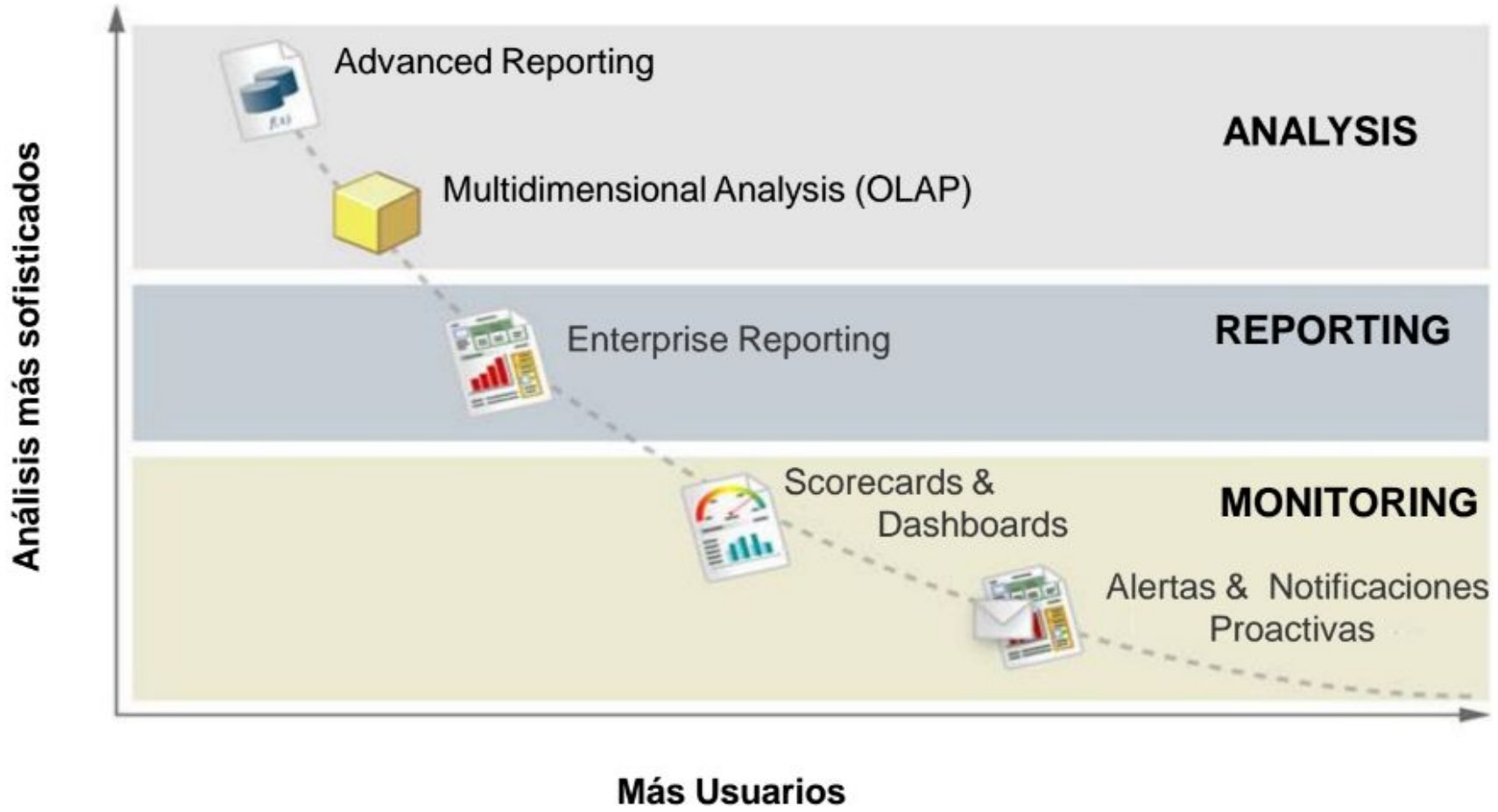
“Procesos, tecnologías y herramientas necesarias para transformar datos en información, información en conocimiento, y conocimientos en planes que nos lleven a **tomar una acción** de negocio rentable.” –

”Conjunto de procesos y herramientas orientadas al **análisis de información** con el objetivo de hacer **uso de datos reales** en el proceso de **toma de decisiones**.” –

Donde se toman las decisiones?

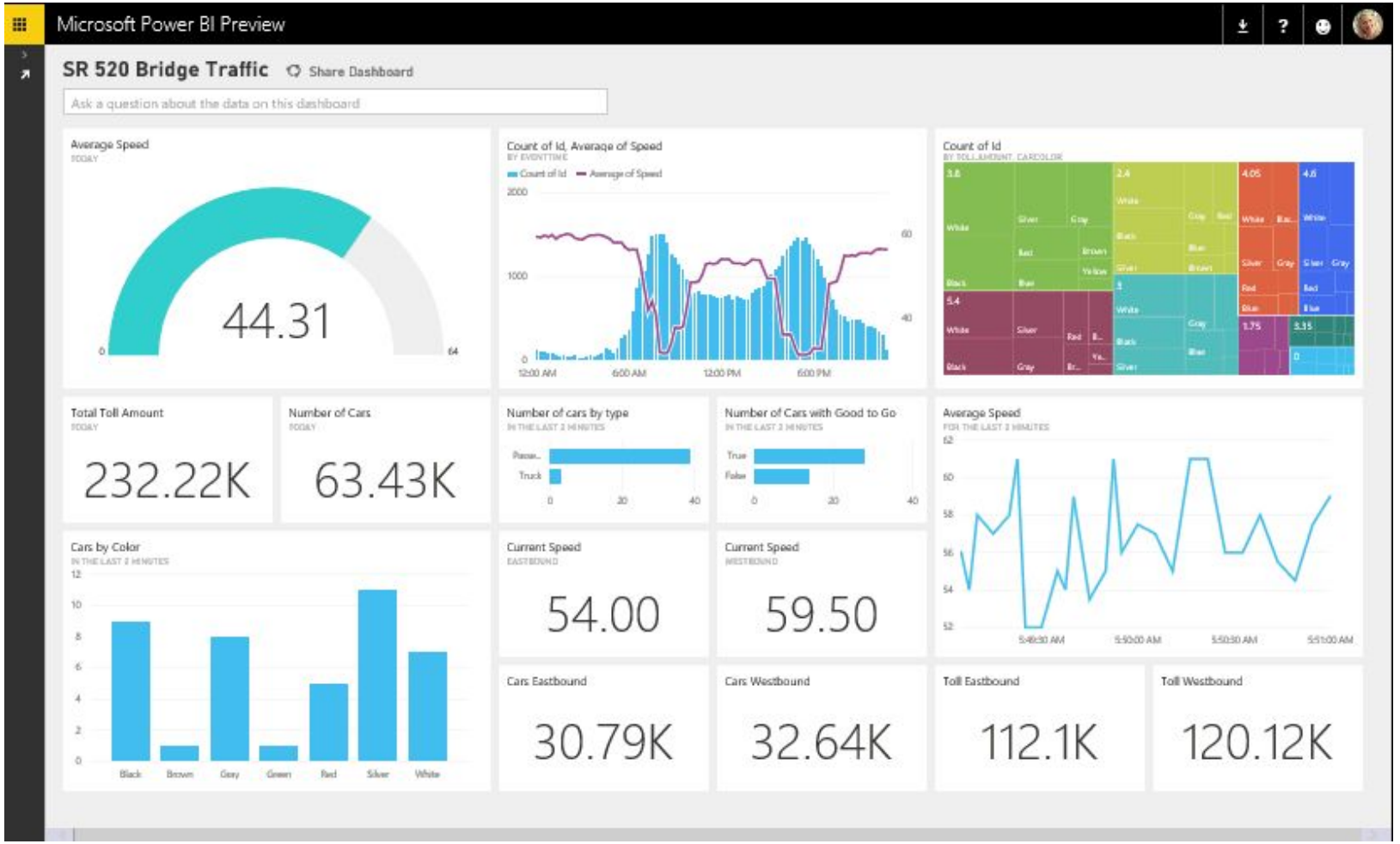
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Niveles de realización de BI



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Monitoreo:
Dashborads (Tableros) & Scorecards

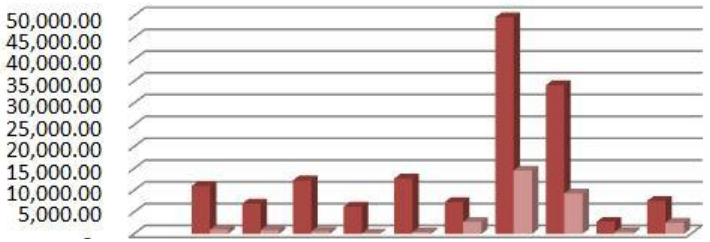


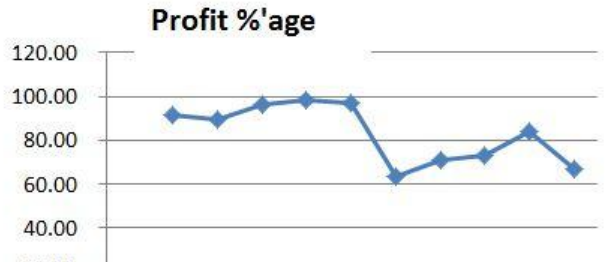
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Reportes

Actual Attendance	Prior YTD					YTD					Yearly Growth
	2010 Q 1	2010 Q 2	2010 Q 3	2010 Q 4	Total	2011 Q 1	2011 Q 2	2011 Q 3	2011 Q 4	Total	
Child Admission	5,849	44,698	50,269	905	**101,721**	5,747	40,791	47,956	774	**95,268**	-6.34%
Adult Admission	12,188	78,431	85,399	1,749	**177,767**	12,893	75,973	83,290	1,690	**173,846**	-2.21%
Membership Admission	54,829	238,444	178,764	5,086	**477,123**	50,070	227,687	170,624	5,265	**453,646**	-4.92%
Group Sales Tickets	2,678	22,732	24,182	353	**49,945**	2,961	19,776	22,348	576	**45,661**	-8.58%
Group Sales - Events	744	10,484	10,367	830	**22,425**	1,398	9,103	12,698	465	**23,664**	5.53%
Comp Admission	956	17,985	9,518	185	**28,644**	1,727	21,088	9,177	238	**32,230**	12.52%
School Admissions	3,605	78,409	7,888	995	**90,897**	3,381	68,582	5,979	14	**77,956**	-14.24%
Event Admission	337	7,298	5,338		**12,973**	1,268	8,679	7,400	15	**17,362**	33.83%
Education Program Admissions	3,944	12,515	10,267	1,007	**27,733**	2,722	10,614	8,398	255	**21,989**	-20.71%
Online Tickets	1	1	125	3	**130**	80	2,030	427	1	**2,538**	1,852.31%
Total	**85,131**	**510,997**	**382,117**	**11,113**	**989,358**	**82,247**	**484,323**	**368,297**	**9,293**	**944,160**	**-4.57%**

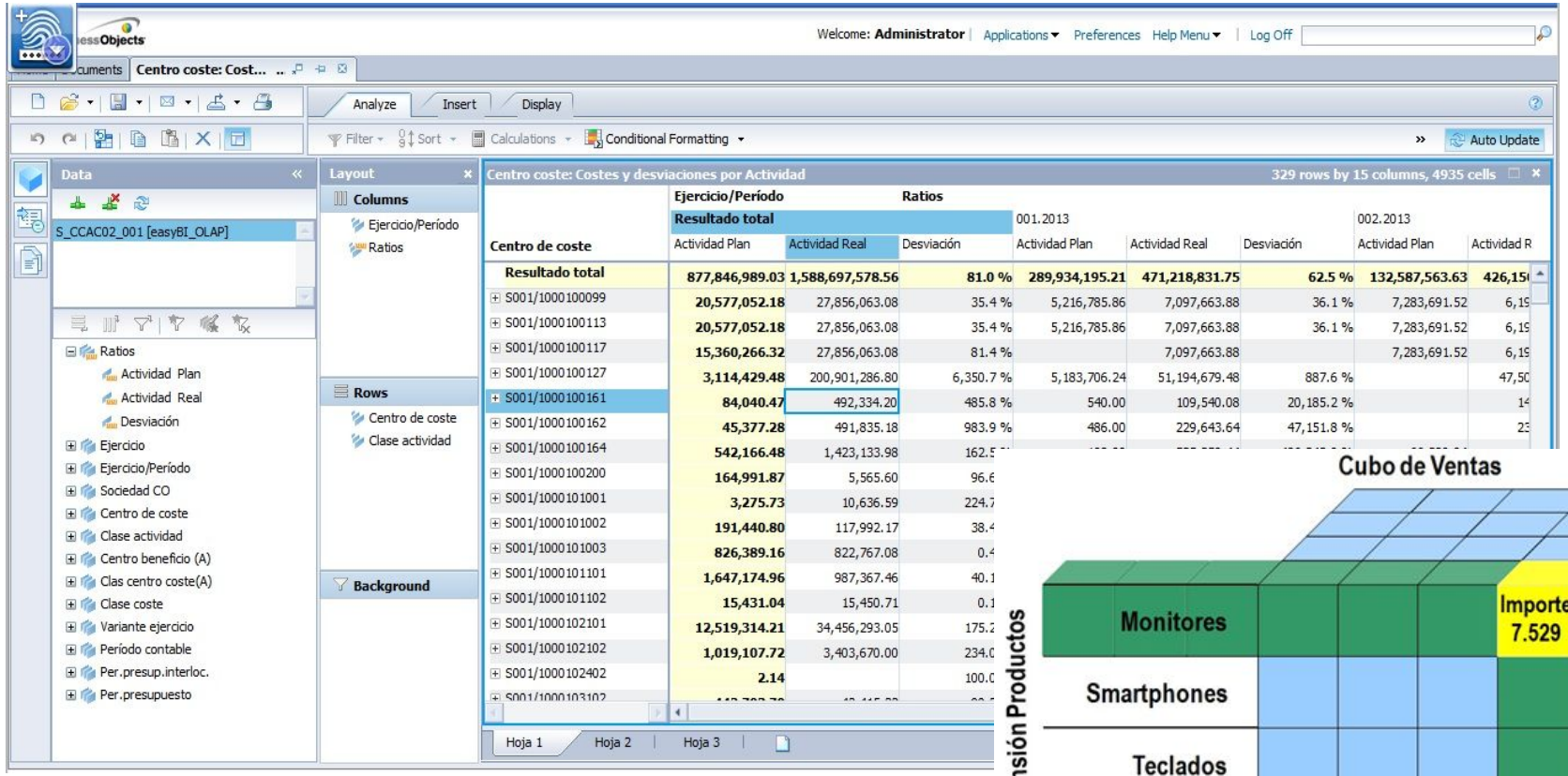
All Client Projects 150,433.35 32,233.35 118,739.16 78.93

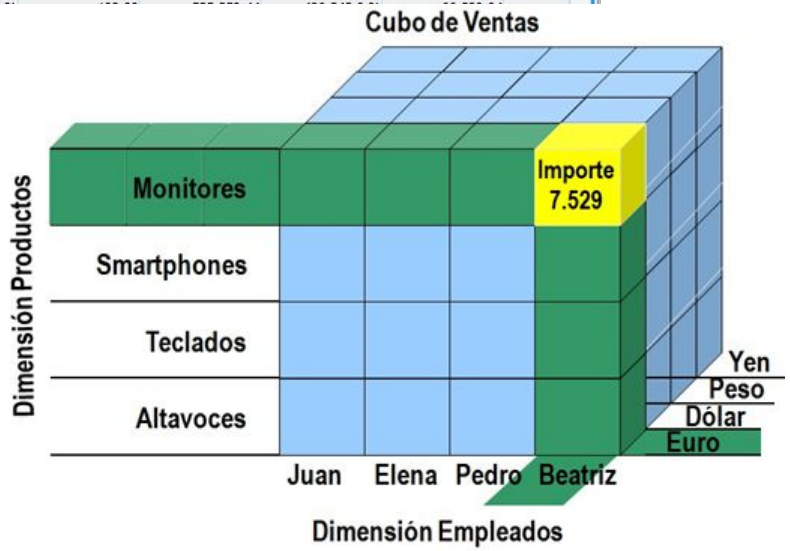




UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

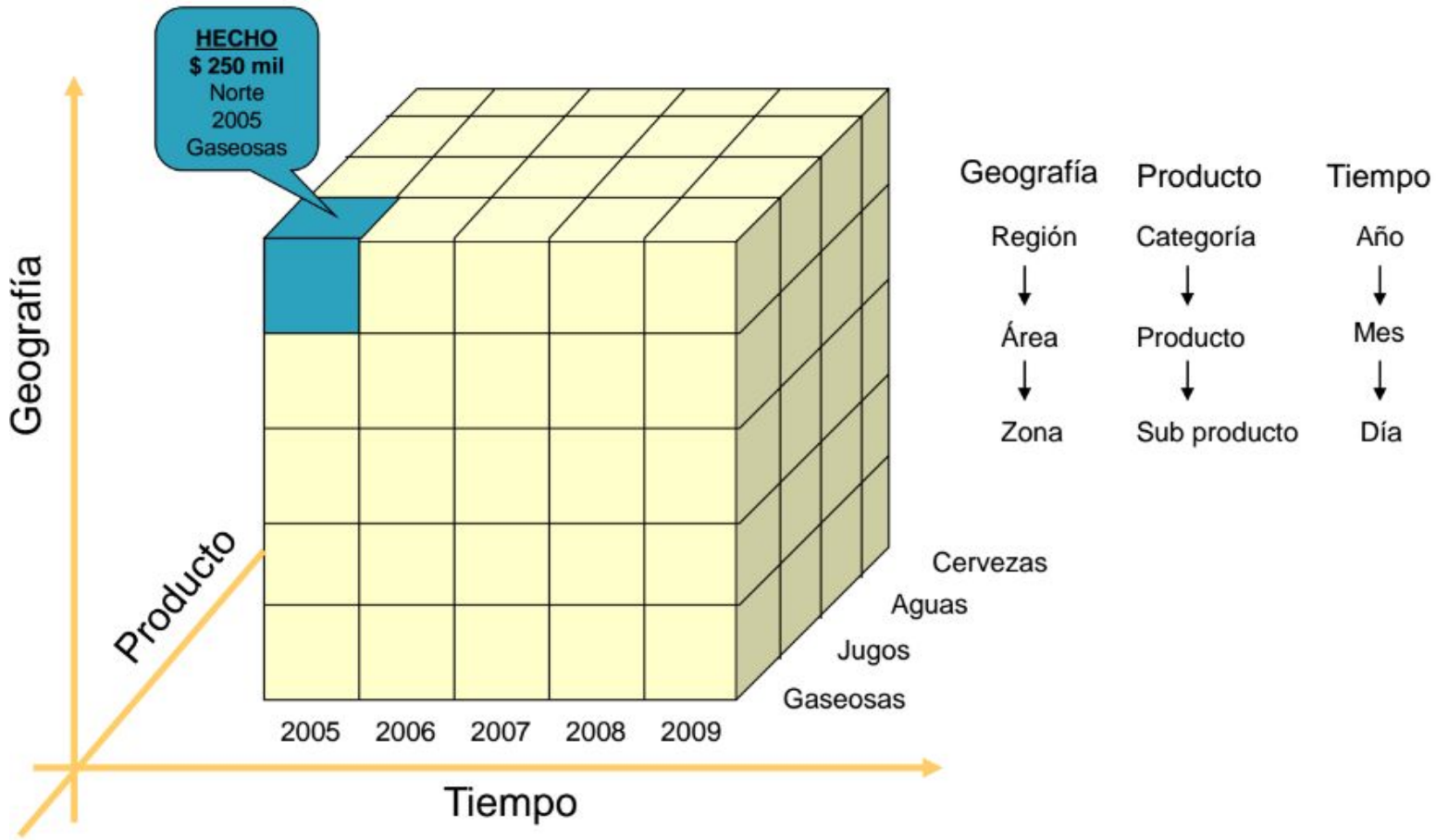
Analysis OLAP/Mulidimensional



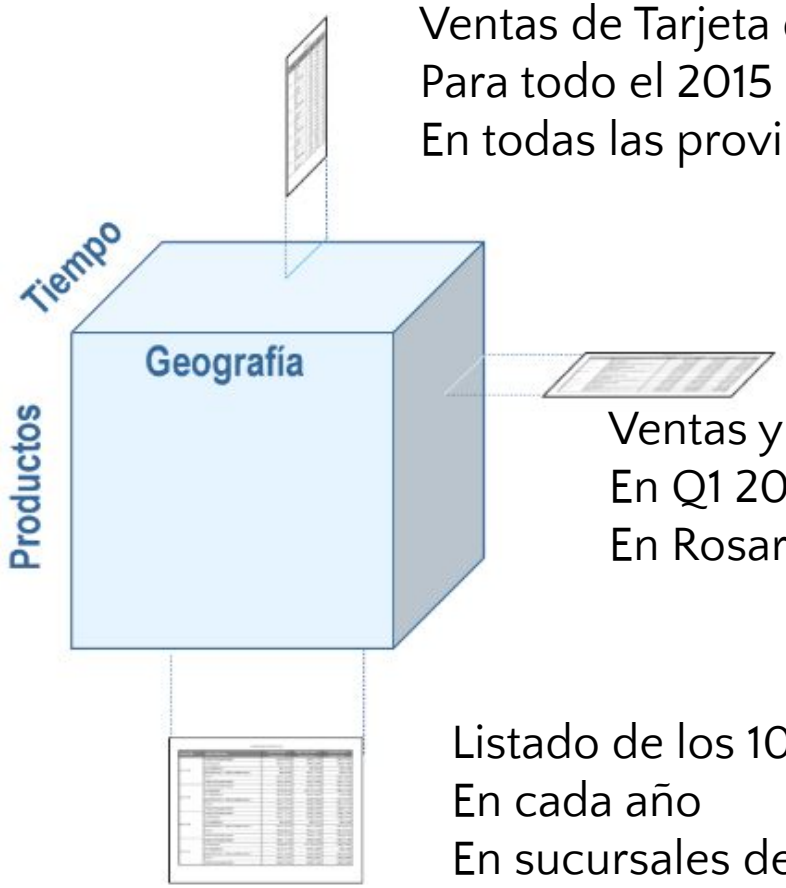


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Analysis OLAP/Mulidimensional



Analysis OLAP/Mulidimensional



Ventas de Tarjeta de Créditos
Para todo el 2015
En todas las provincias

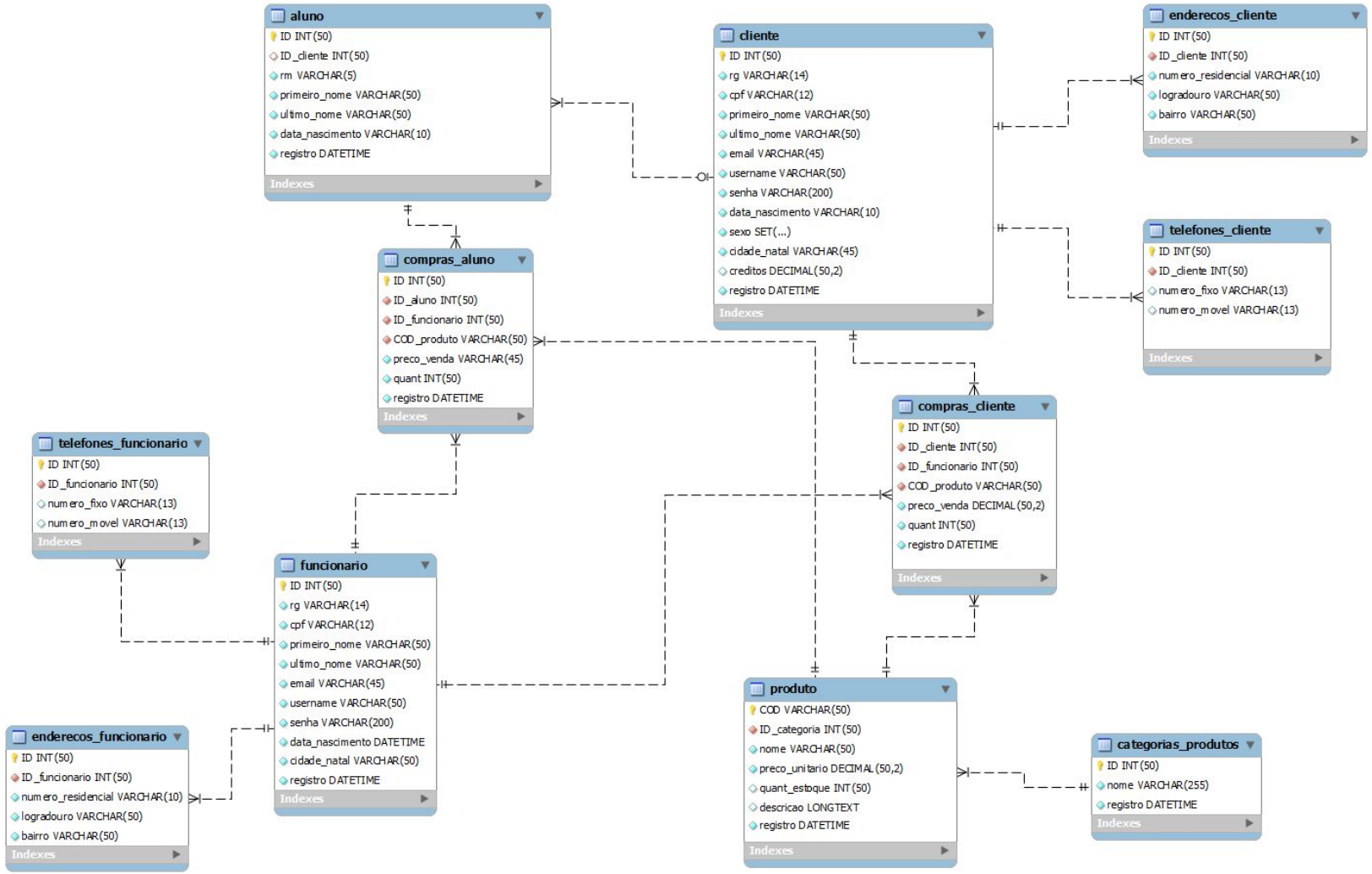
Ventas y costo totales
En Q1 2016
En Rosario (Santa Fe)

Listado de los 10 productos mas vendidos
En cada año
En sucursales de zona rurales

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Modelado

DER



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Modelado Dimensional

"El modelado dimensional es una técnica de diseño que busca presentar los datos en un framework estándar, intuitivo y escalable, que permite un acceso a los datos altamente performante, basándose en el modelado relacional pero con algunas restricciones de diseño importantes"

Objetivos Modelo Dimensional

Modelos predecibles y estándar: ayudando a quienes tienen que hacer reportes, herramientas, dba´s.

No presenta cambios inesperados. Todas las dimensiones son equivalentes, pueden ser pensadas como puntos simétricos de acceso a las fact.

- Interfaz de usuario simétrica.
- Estrategias de queries simétricas.
- SQL generado simétrico.

Existen varios prototipos para los modelos de negocios ya existentes.

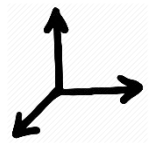
Fácilmente escalable

Conceptos Modelado Dimensional...

Hechos (Facts)

Métricas

Elementos



Dimensiones

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Hecho

Evento concretos y especifico del proceso de negocio, y de interés para la organización.

"An observation in the Marketplace" – *Kimball*

Valores Cuantitativos de tipo numérico (*Especialmente con decimales*)

No se conoce de antemano

Atributos

Generalmente del tipo texto *(o pueden tratarse como tal)*.

Valores cualitativos.

Describe características de una entidad.

Proveen contexto a los hechos.

Proveen nivel de detalle a las métricas.

Elemento

Instancia o valor que puede tomar un atributo.

Estado Civil *(Atributo)*
- Soltero *(Elemento)*
- Casado *(Elemento)*
- Viudo *(Elemento)*
- Divorciado *(Elemento)*

Nacionalidad *(Atributo)*
- Argentina *(Elemento)*
- Chileno *(Elemento)*

Fecha de Nacimiento *(Atributo)*
- 2/3/85 *(Elemento)*
- 3/3/85 *(Elemento)*

...

Nombre de Producto *(Atributo)*
- Televisor *(Elemento)*
- Buzo *(Elemento)*
- Campera *(Elemento)*
- Notebook *(Elemento)*

Categoria *(Atributo)*
- Microcentro *(Elemento)*
- San Telmo *(Elemento)*
- Recoleta *(Elemento)*
- Pilar *(Elemento)*
- Córdoba *(Elemento)*

Dimensiones

Agrupaciones de atributos que estén altamente correlacionados entre si.

Cliente *(Dimensión)*

- Estado Civil *(Atributo)*: Soltero *(Elemento)*
- Sexo *(Atributo)*: Masculino *(Elemento)*
- Nacimiento *(Atributo)*: 3/7/85 *(Elemento)*

Dimensiones Conformadas

- Es una dimensión que tiene el mismo significado para todos los datamarts que se generan.
- La mayor responsabilidad del equipo que diseña un DW es establecer, publicar, mantener las dimensiones conformadas.
- Sin una estricta adherencia de las dimensiones conformadas, el DW no podrá funcionar como un todo integrado.
- Claves para poder obtener tener visión integral de la organización