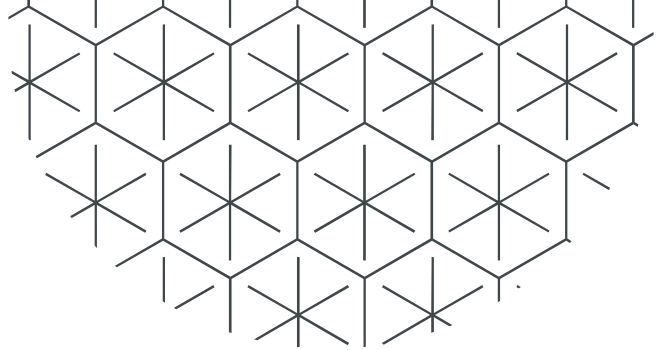
Tecnologías de datos masivos

Doble Grado en Ingeniería en Tecnologías de Telecomunicación y Business Analytics



SparkSQL

SparkSQL - Introducción

- Conforme se fue popularizando el uso de Spark se planteó como reemplazo de Map&Reduce para los proyectos en los que usaba como solución
- Hive fue uno de los primeros proyectos en reemplazar Map&Reduce por Spark
- El primer intento de reemplazo fue Shark
- Se vió que intentar reemplazar Map&Reduce requería mucho tiempo y cambios en Spark
- Se empieza a trabajar en una solución interna de Spark llamada SparkSQL

SparkSQL - Introducción

- SparkSQL buscaría en un principio únicamente crear RDD con la misma estructura en todos los elementos del RDD
- Hasta SparkSQL las Task eran cajas negras
    - Sólo se sabía que entra y que sale
    - No sabemos qué hace ninguna de las Task de aplicación
- Los RDD sólo tenían tipos de variables
- Al ser cajas negras no se puede sacar estadísticas de lo que está sucediendo dentro

SparkSQL - Introducción

- Al igual que Hive al saber exactamente que hace cada uno de los **Stage** y los **Task** de un **application Spark** optimiza el procesamiento
- Al basarse en Spark para la lectura de los datos puede optimizar la misma
- Al igual que Hive SparkSQL tiene un metastore que se puede compartir entre distintos **application**
- Al igual que Hive SparkSQL permite a herramientas de Bussiness Intelligence procesen cantidades masivas de datos de manera eficinete
- En las últimas versiones de Spark SparkSQL se convierte en el estandar de procesamiento y en la pieza central de Spark

SparkSQL - SparkSession

- En sus primeras versiones Spark tenía tres puntos de acceso para sus distintas problemáticas:
    - SparkContext: Punto de entrada de Batch en el que, además, se establecen todas las propiedades de comunicación con el gestor de recursos
    - SQLContext: Se creaba a partir de un SparkContext se usa para crear Dataframes y la interacción con datastores externos de una manera sencilla y amigable.
    - StreamingContext: En él se gestionan todos los procesos de Streaming y se definen las ventanas de tiempo de Batch. Necesita también un Contexto de Spark para gestionar su RDDs internos

SparkSQL - SparkSession

- En sus versiones más actuales Spark ha sustituido el punto de acceso de SQL del **SQLContext** al **SparkSession**
- **SparkSession** nos permite tener una sesión por usuario que utilice un SparkContext
- Crear múltiples **SparkSession** en un mismo **SparkContext** habilita que distintos usuarios utilicen el mismo contexto de Spark
- En la práctica se usa únicamente un **SparkSession** por **SparkContext** para evitar la compartición de recursos
- A diferencia del **SparkContext SparkSession** nos permite cambiar alguna de sus configuraciones en caliente

SparkSQL - SparkSession

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import SparkSession
import os
username = os.environ['JUPYTERHUB_USER']


spark = SparkSession.builder.appName(f"sql-{username}")\
.master('yarn-client')\
.config("spark.executor.instances", "3")\
.config("spark.executor.cores", "2")\
.config("spark.executor.memory", "2g")\
.getOrCreate()
```

SparkSQL - Catálogo

- SparkSession tiene un catálogo similar al metastore de Apache Hive
- Como en Hive el Catalogo de Spark guarda la metainformación de las tablas que se pueden usar en una sesión y además nos permite hacer sentencias SQL tradicionales
- Registra también funciones definidas por el usuario llamadas **UDF** y **UDFA**
- A diferencia del metastore de Hive no es necesario, aunque es posible, guardarlo en una base datos tradicional
- Podemos almacenar **RDD** con estructura (**Dataframe**) para almacenar tablas

SparkSQL - Catálogo

Listar las tablas y columnas de un catalogo de Spark

```
>>> spark.catalog.listTables()
[Table(name='a', database='default', description=None,
tableType='MANAGED', isTemporary=False)]

>>> spark.catalog.listColumns(tableName="A")
[Column(name='salary', description=None, dataType='int', nullable=True,
isPartition=False, isBucket=False), Column(name='firstname',
description=None, dataType='string', nullable=True, isPartition=False,
isBucket=False)]
```

SparkSQL - Catálogo

Mostrar los registros de una tabla registrada en el catalogo de Spark

```
>>>persons = spark.sql("SELECT * FROM persons")

>>>persons.show()
+------+-----------+
|salary|  firstname|
+------+-----------+
|   100|Jorge Lopez|
|   200| Mario Ruiz|
+------+-----------+
```

SparkSQL - Datastore

- Tanto SparkSQL como spark Core sólo pueden crear que cuya informacion se ubique en memoria o en sistema de ficheros distribuido
- Tanto SparkSQL como spark Core sólo pueden o bien devolver la información de RDD por pantalla o guardarlo en un sistema de ficheros distribuido
- Tanto en SparkCore como en SparkSQL se añade compatibilidad con múltiples tecnologías en las que se puede guardar y leer información
- Para Spark Core leer datos de otra tecnologia de almacenamiento requería crear un nuevo tipo de RDD e incluso a veces un nuevo tipo de Contexto y cada tecnología tenía el suyo propio con sus propias características
- Con la llegada de SparkSQL se unifica la forma de crear RDD que leea de otra tecnología de datos

SparkSQL - Datastore

- Las tecnologias de almacenamiento compatibles con SparkSQL se conocen como Datastores
- Para añadirlas a nuestro application Spark ha desarrollado una forma simple de traernos las dependencias
- Todos los **Dataframes** proceden de un datastore y se crean usando el **SparkSession**
- A diferencia de los sistemas de ficheros distribuidos, aumentar la versión de Spark pude hacer que nuestro código deje de ser compatible
- Codificiar un nuevo Datastore es un proceso complejo que requiere conocimiento experto de la tecnología a leer, por lo que se suele delegar el desarrollo a las compañías responsables de la tecnologia

SparkSQL - Datastore

- Las tecnologias de almacenamiento compatibles con SparkSQL se conocen como Datastores
- Para añadirlas a nuestro application Spark ha desarrollado una forma simple de traernos las dependencias
- Todos los **Dataframes** proceden de un datastore y se crean usando el **SparkSession**
- A diferencia de los sistemas de ficheros distribuidos, aumentar la versión de Spark pude hacer que nuestro código deje de ser compatible
- Codificiar un nuevo Datastore es un proceso complejo que requiere conocimiento experto de la tecnología a leer, por lo que se suele delegar el desarrollo a las compañías responsables de la tecnologia

SparkSQL - Catálogo

Crear un Dataframe con los ficheros de personal en formato CSV

```
>>>persons = spark.read.option("header", "true").csv("/data/personal")
```

SparkSQL - Catálogo

Crear un Dataframe con los ficheros de personal en formato CSV infiriendo su esquema

```
>>>persons = spark.read.option("header", "true").csv("/data/personal")

>>>persons_with_types = spark.read.option("header", "true")\
.option("inferSchema", "true")\
.csv("/data/personal")

>>>persons.printSchema()
root
 |-- age: string (nullable = true)
 |-- name: string (nullable = true)
 |-- surname: string (nullable = true)
 |-- salary: string (nullable = true)
```

SparkSQL - Catálogo

Crear un Dataframe con los ficheros de personal en formato CSV infiriendo su esquema

```
>>>persons = spark.read.option("header", "true").csv("/data/personal")

>>>persons_with_types = spark.read.option("header", "true")\
.option("inferSchema", "true")\
.csv("/data/personal")

>>>persons_with_types.printSchema()
root
 |-- age: integer (nullable = true)
 |-- name: integer (nullable = true)
 |-- surname: integer (nullable = true)
 |-- salary: integer (nullable = true)
```

SparkSQL - Dataframe

- SparkSQL utiliza un nuevo tipo de **RDD**, los **Dataframe**
- Los **Dataframe** cumplen todas las propiedades de los **RDD**
- Los **Dataframe** son RDD con estructura
- Cada elemento del **Dataframe** tiene que tener la misma estructura y cada elemento es un **Row**
- Un **Row** es un Array en el que cada elemento puede tener un tipo distinto de datos

SparkSQL - Dataframe

- El hecho de conocer todas las operaciones que se realizan dentro de un application permite optimizar las operaciones de manera similar a lo que hace **Apache Hive**
- Los **Dataframe** pueden recorrer los **RDD para saber la estructura**
- Facilita el acceso a columnas específicas
- Al igual que **Apache Hive** el conocer la estructura de los datos y las operaciones exactas que vamos a realizar con ellos **SparkSQL** optimiza nuestros **Job**s

SparkSQL - Dataframe

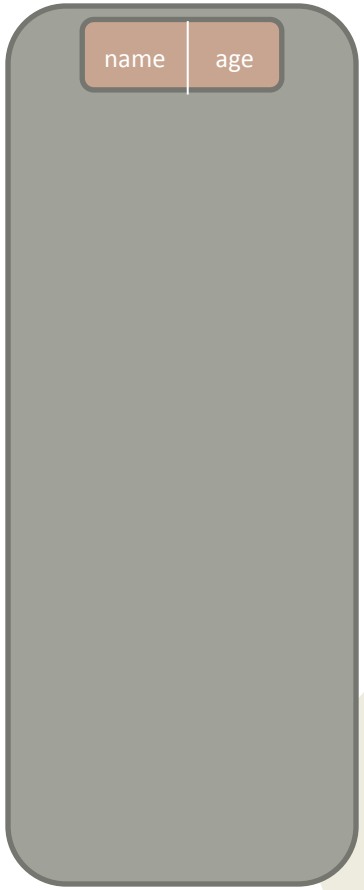
- El hecho de conocer todas las operaciones que se realizan dentro de un application permite optimizar las operaciones de manera similar a lo que hace **Apache Hive**
- Los **Dataframe** pueden recorrer los **RDD para saber la estructura**
- Facilita el acceso a columnas específicas
- Al igual que **Apache Hive** el conocer la estructura de los datos y las operaciones exactas que vamos a realizar con ellos **SparkSQL** optimiza nuestros **Job**s

SparkSQL - Select

- **Select** selecciona un conjunto de columnas de un **Dataframe**
- Nos permite seleccionar columnas enteras de datos
- Se corresponde a una operación map en la que seleccionamos sólo un conjuto de columnas de nuestros datos
- No nos permite transformar ningun dato
- Es una transformación del tipo Narrow y por ende **no** llama a Shuffle ni termina un **Stage**

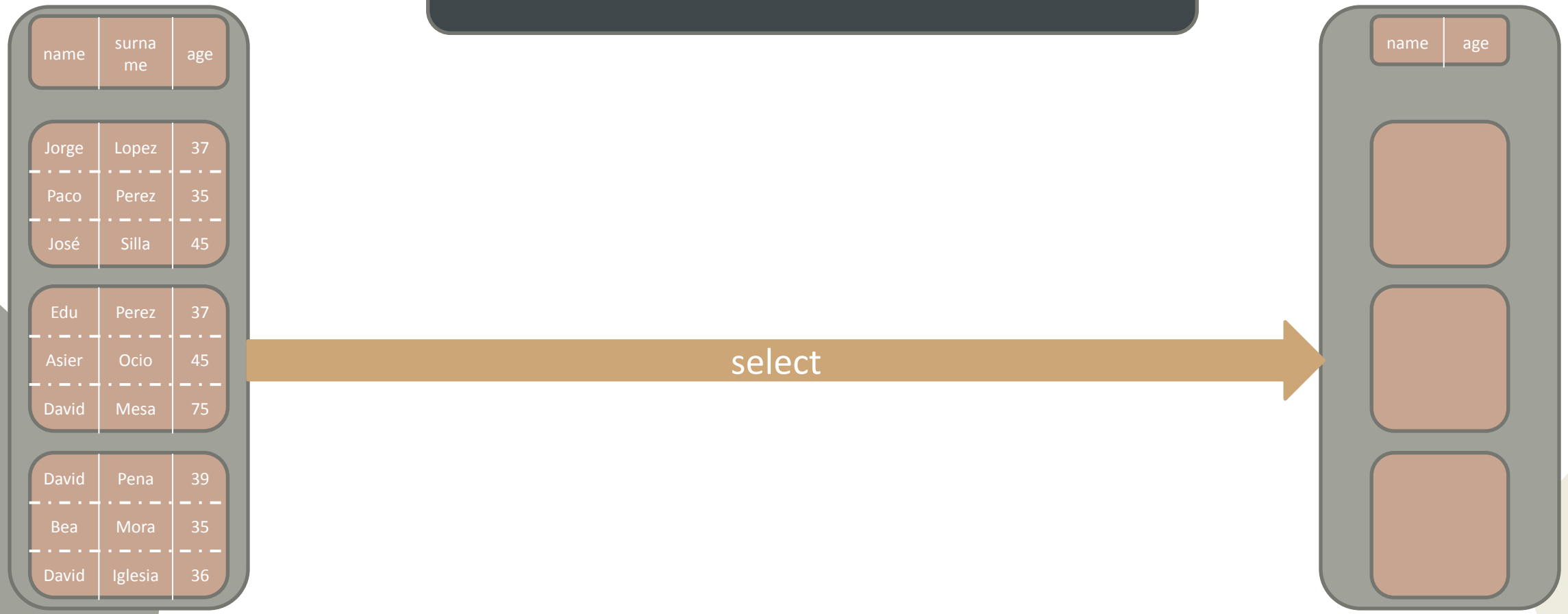
SparkSQL - Select

```
>>>name_age = persons.select("name", "age")
```



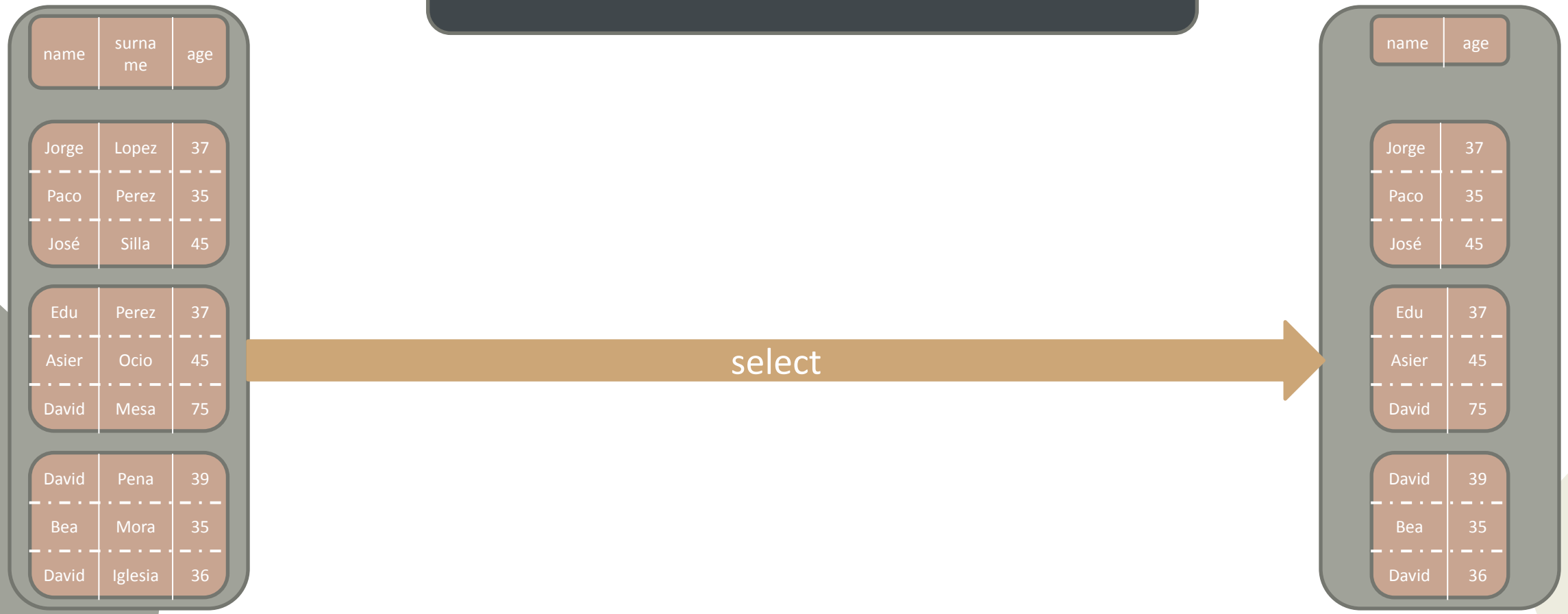
SparkSQL - Select

```
>>>name_age = persons.select("name", "age")
```



SparkSQL - Select

```
>>>name_age = persons.select("name", "age")
```



SparkSQL - withColumn

- **withColumn** añade una columna al conjunto de columnas de un **Dataframe**
- Podemos transformar una columna en otro tipo de datos, pero seguiremos teniendo las dos
- Se puede asignar el resultado a una columna ya existente del **Dataframe** lo que hará que se sustituya su valor
- Se pueden usar funciones que ya han sido creadas por otros o usar una udf generada por el usuario
- Es una transformación del tipo Narrow y por ende **no** llama a Shuffle ni termina un **Stage**

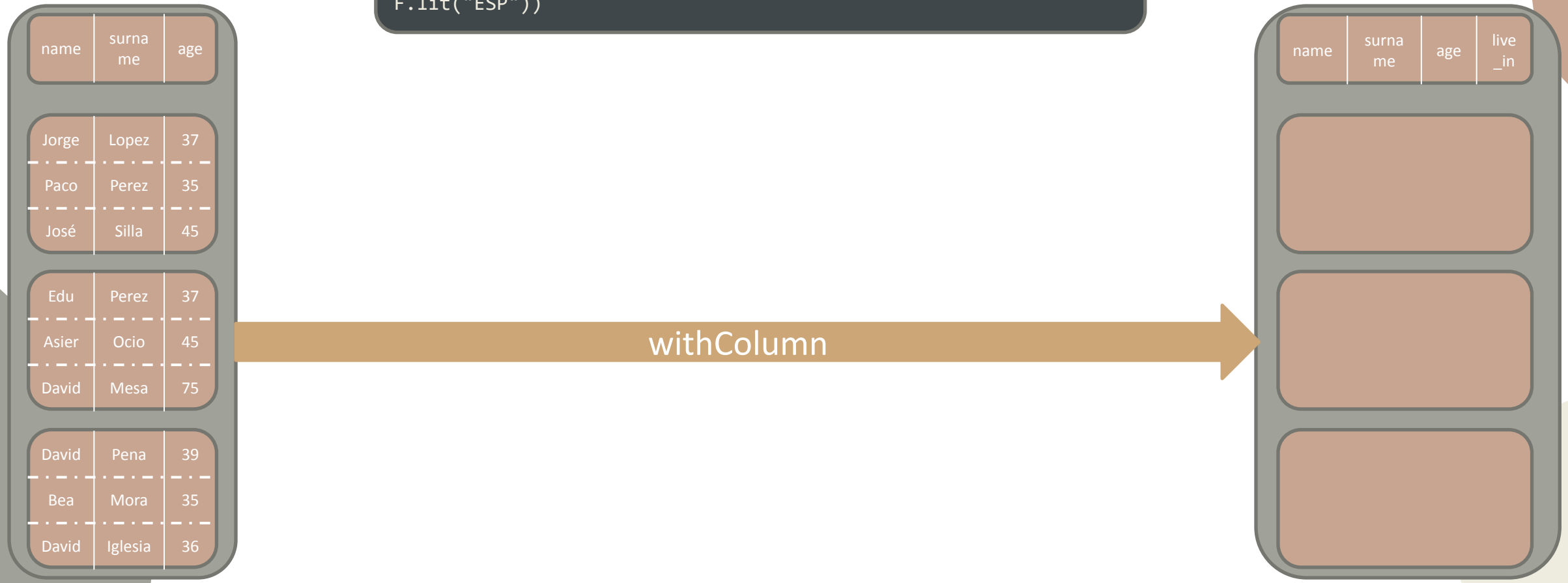
SparkSQL - withColumn

```
import pyspark.sql.functions as F

>>>spanish_name_age = persons.withColumn("live_in",
F.lit("ESP"))
```

name	surname	age
Jorge	Lopez	37
Paco	Perez	35
José	Silla	45
Edu	Perez	37
Asier	Ocio	45
David	Mesa	75
David	Pena	39
Bea	Mora	35
David	Iglesia	36

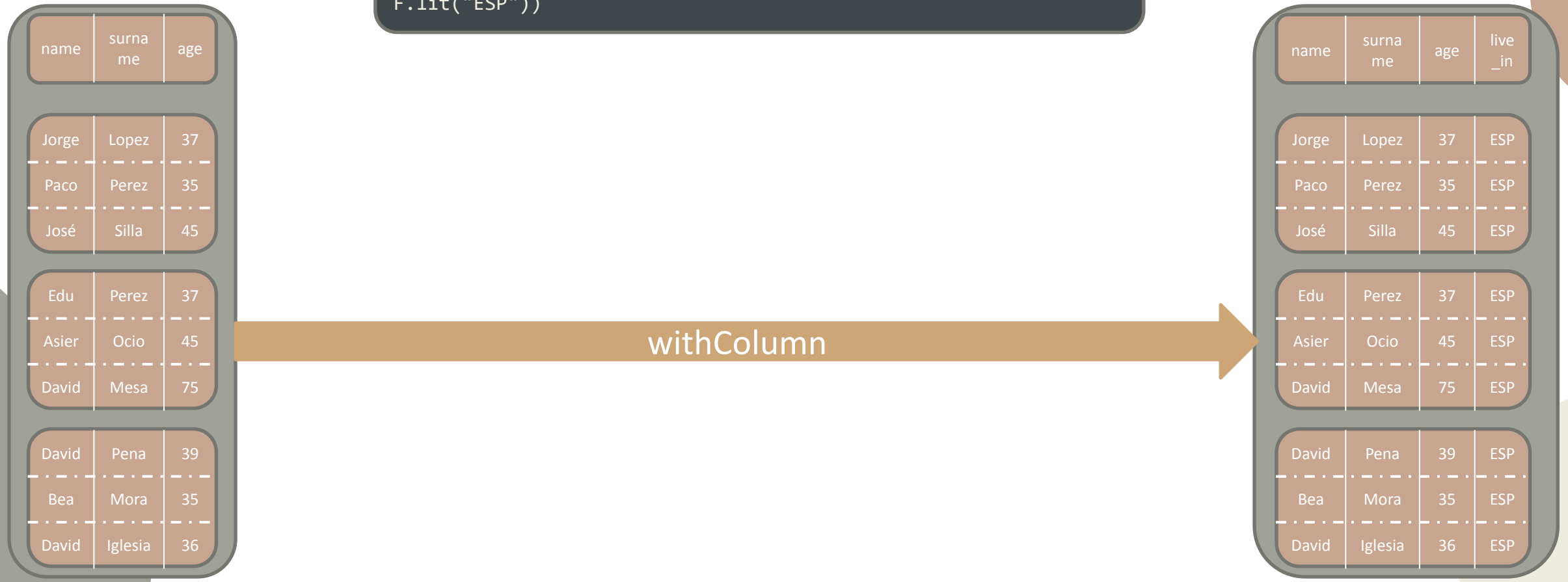
SparkSQL - withColumn

```
import pyspark.sql.functions as F

>>>spanish_name_age = persons.withColumn("live_in",
F.lit("ESP"))
```



SparkSQL - withColumn

```
import pyspark.sql.functions as F

>>>spanish_name_age = persons.withColumn("live_in",
F.lit("ESP"))
```



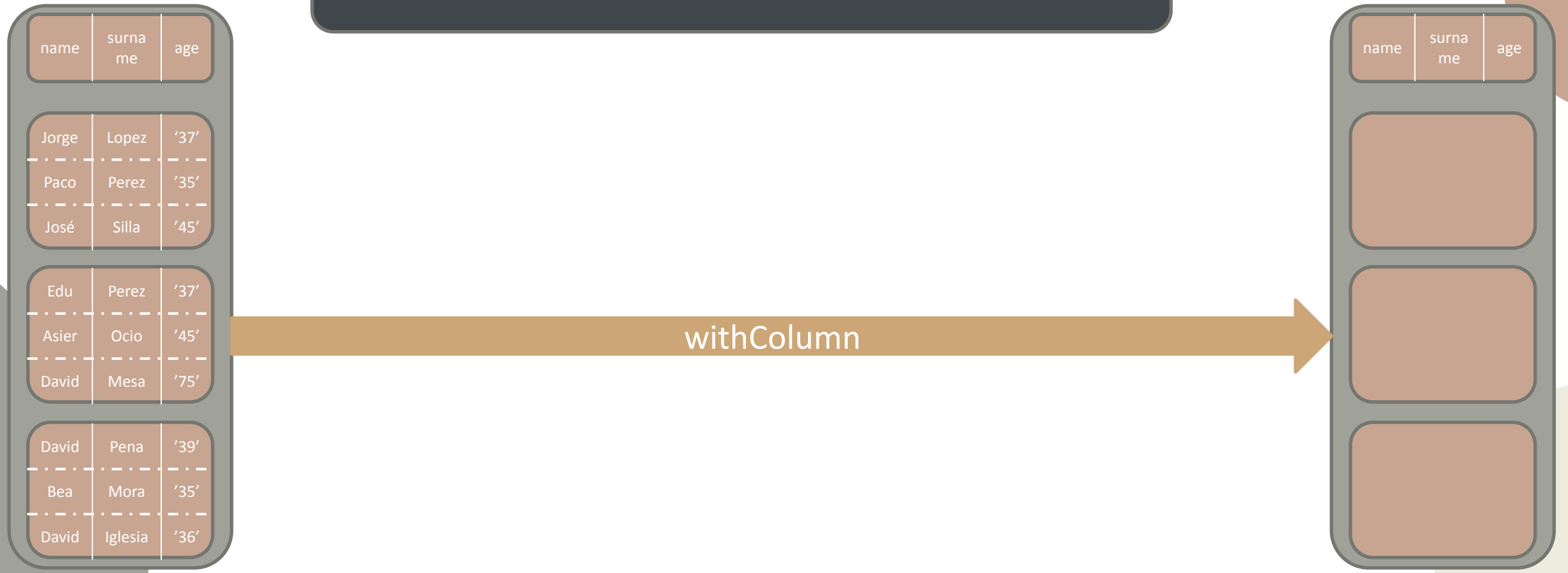
SparkSQL - withColumn

```
import pyspark.sql.functions as F

>>>name_age = persons.withColumn("age", persons["age"].cast("Integer"))
```

name	surname	age
Jorge	Lopez	‘37’
Paco	Perez	’35’
José	Silla	’45’
Edu	Perez	’37’
Asier	Ocio	’45’
David	Mesa	’75’
David	Pena	’39’
Bea	Mora	’35’
David	Iglesia	’36’

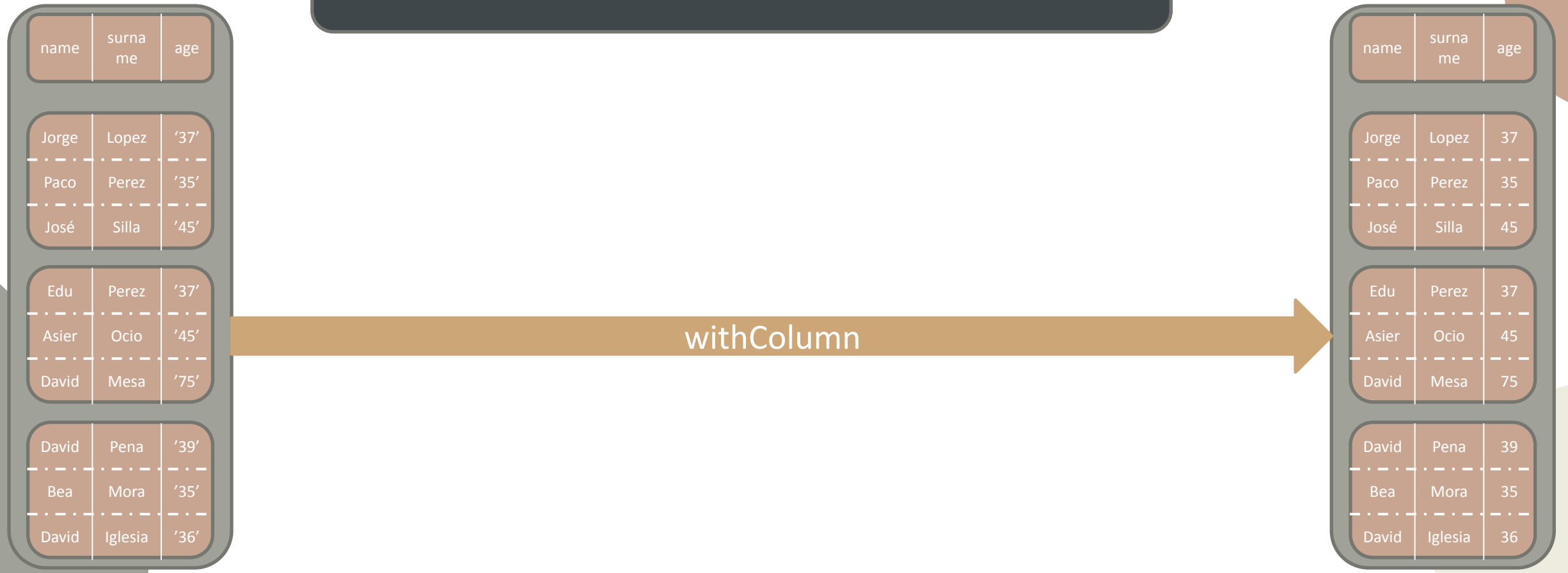
SparkSQL - withColumn

```
import pyspark.sql.functions as F

>>>name_age = persons.withColumn("age", persons["age"].cast("Integer"))
```



SparkSQL - withColumn

```
import pyspark.sql.functions as F

>>>name_age = persons.withColumn("age", persons["age"].cast("Integer"))
```



SparkSQL - withColumnRenamed

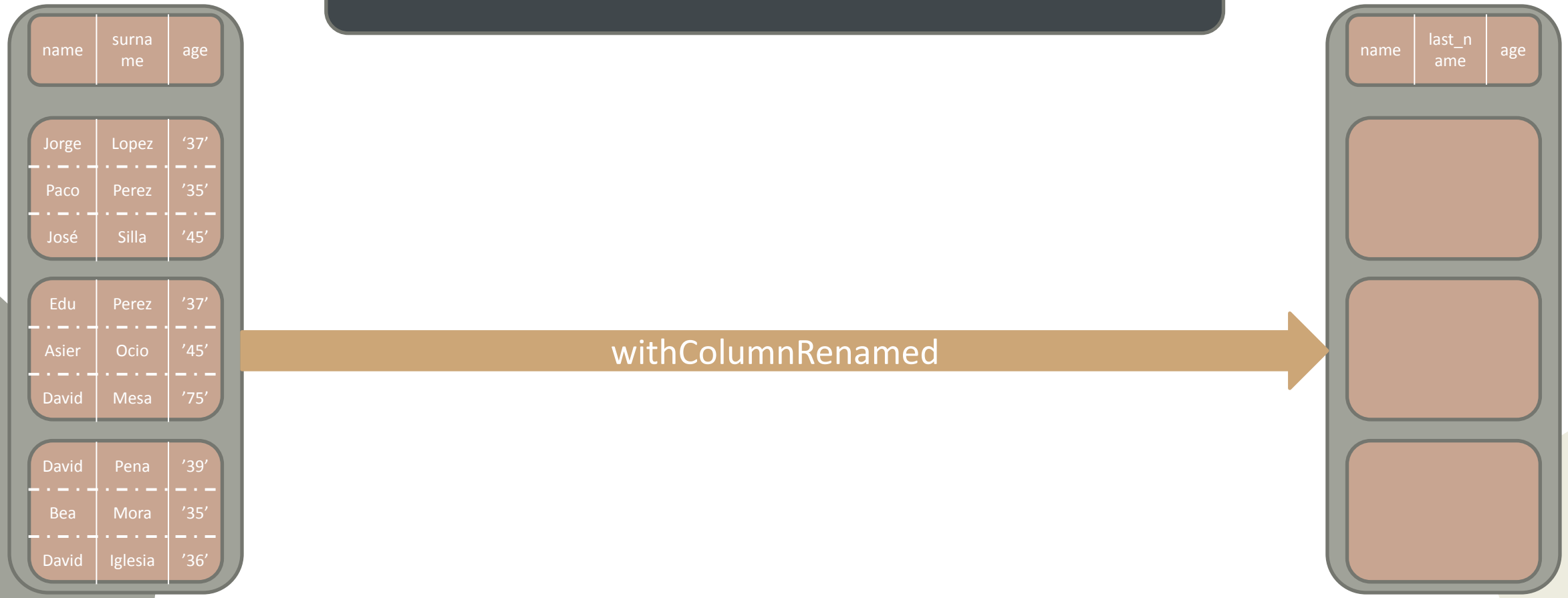
- **withColumnRenamed** renombra una columna de un **Dataframe**
- Los datos serán exactamente iguales pero cambia el nombre de la columna
- Util para evitar la ambigüedad de columnas cuando se hace un join
- Es una transformación del tipo Narrow y por ende **no** llama a Shuffle ni termina un **Stage**

SparkSQL - withColumnRenamed

```
import pyspark.sql.functions as F

>>>name_age = persons.withColumnRenamed("surname", "last_name")
```

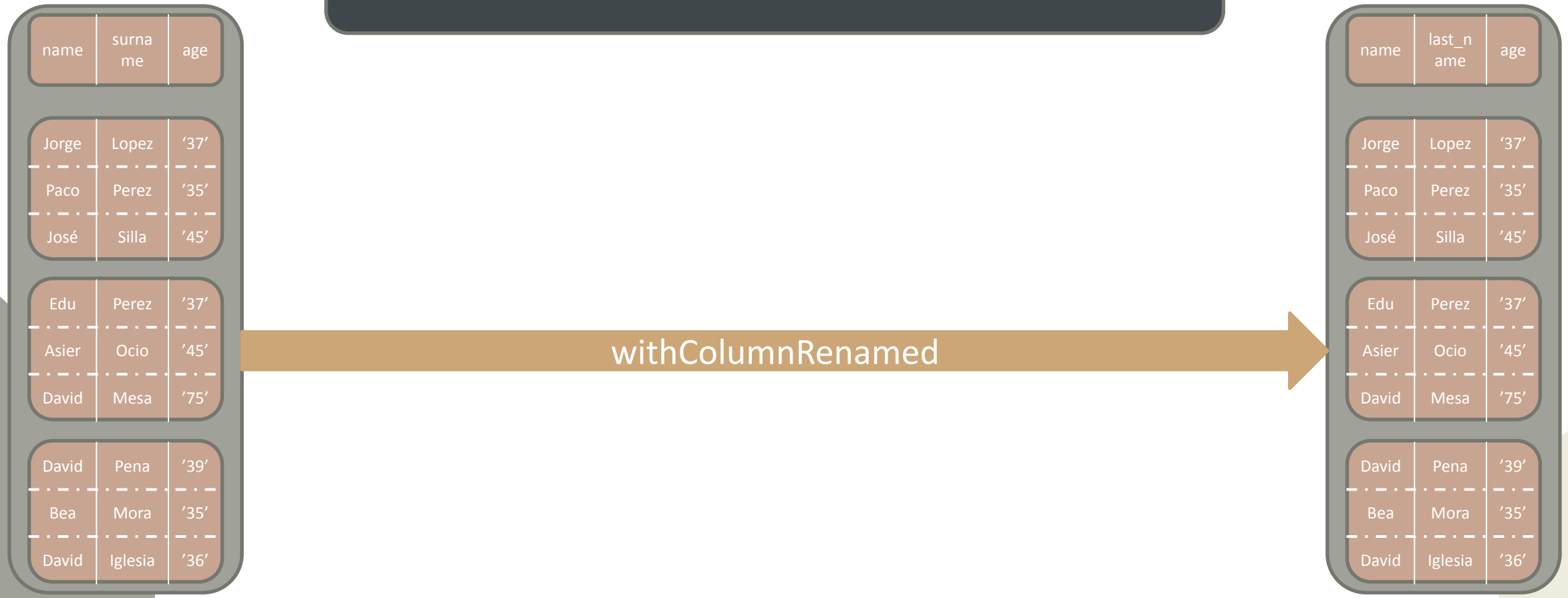
SparkSQL - withColumnRenamed

```
import pyspark.sql.functions as F

>>>name_age = persons.withColumnRenamed("surname", "last_name")
```



SparkSQL - withColumnRenamed

```
import pyspark.sql.functions as F

>>>name_age = persons.withColumnRenamed("surname", "last_name")
```



SparkSQL - where

- **where** flitra los registros de un **Dataframe** quedandose únicamente con aquellos que cumplan la condición
- No cambia el tipo de datos de ninguna columna
- A difrencia de otras igualdades en python los operadores bianarios **and** y **or** se sistuyen por **& y |** respectivamente
- Es una transformación del tipo Narrow y por ende **no** llama a Shuffle ni termina un **Stage**

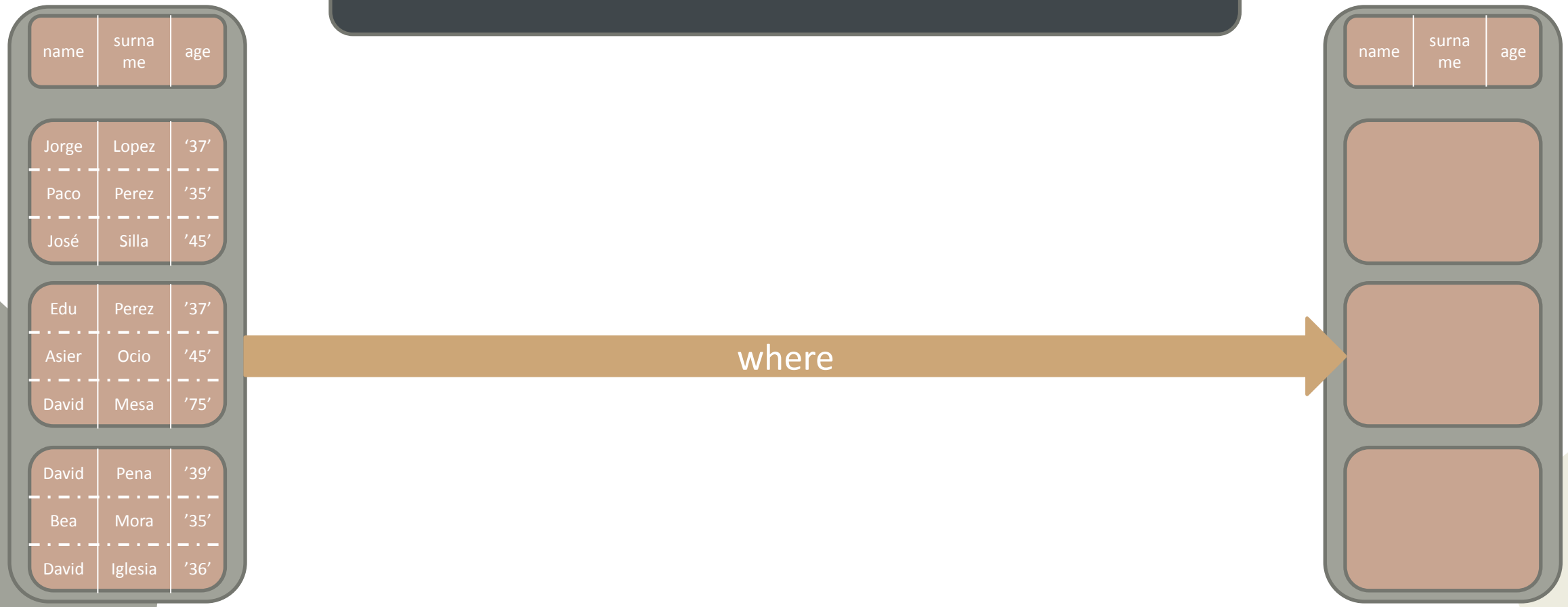
SparkSQL - where

```
>>>older_than37_ez = persons.where((personal["age"] >= 37) &
(personal["surname"].endswith('ez')))
```

name	surname	age
Jorge	Lopez	‘37’
Paco	Perez	’35’
José	Silla	’45’
Edu	Perez	’37’
Asier	Ocio	’45’
David	Mesa	’75’
David	Pena	’39’
Bea	Mora	’35’
David	Iglesia	’36’

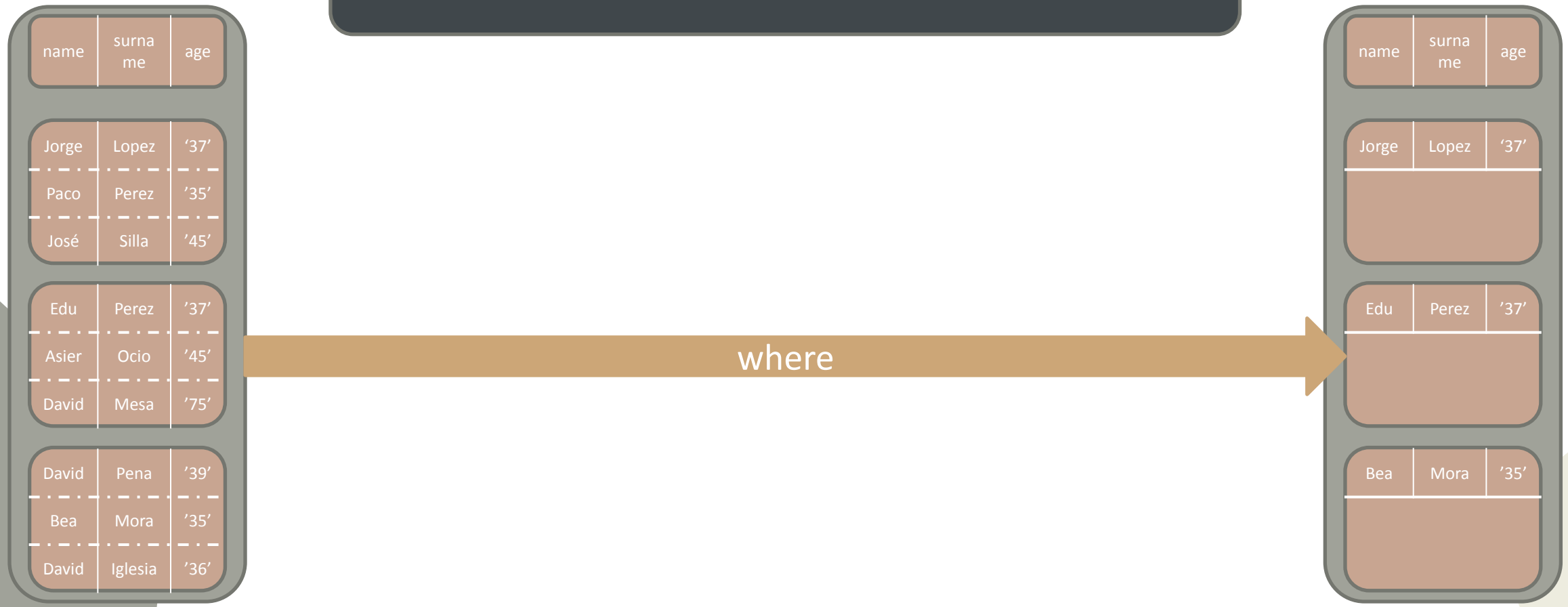
SparkSQL - where

```
>>>older_than37_ez = persons.where((personal["age"] >= 37) &
(personal["surname"].endswith('ez')))
```



SparkSQL - where

```
>>>older_than37_ez = persons.where((personal["age"] >= 37) &
(personal["surname"].endswith('ez')))
```



SparkSQL - groupBy

- **groupBy** agrupa los registros de un **Dataframe** con el mismo valor en las columnas que se le pasan por parametro
- Es la única transformación de cuyo resultado no se puede realizar una acción
- Tras un group by hay que aplicar alguna operacion de reducción al resto de columnas del **Dataframe**
- Es una transformación del tipo Wide y por ende llama a Shuffle y termina un **Stage**

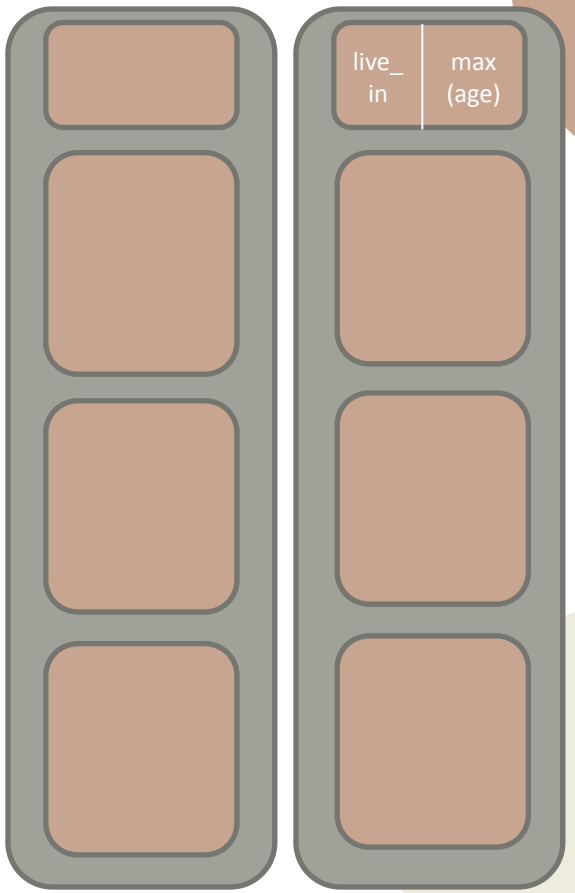
SparkSQL - groupBy

```
>>>max_age_per_live_in =
spanish_name_age.groupBy(persons["live_in"]).max("age")
```

name	surname	age	live_in
Jorge	Lopez	37	ESP
Paco	Perez	35	ESP
José	Silla	45	FRA
Edu	Perez	37	ESP
Asier	Ocio	45	GBR
David	Mesa	75	ESP
David	Pena	39	FRA
Bea	Mora	35	GER
David	Iglesia	36	GBR

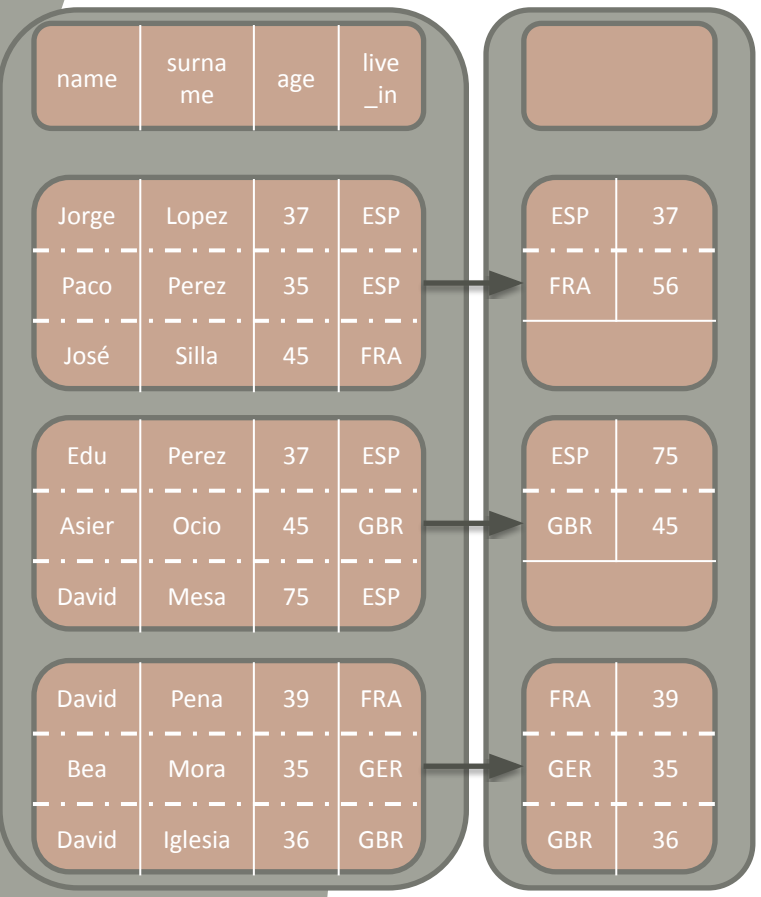
SparkSQL - groupBy

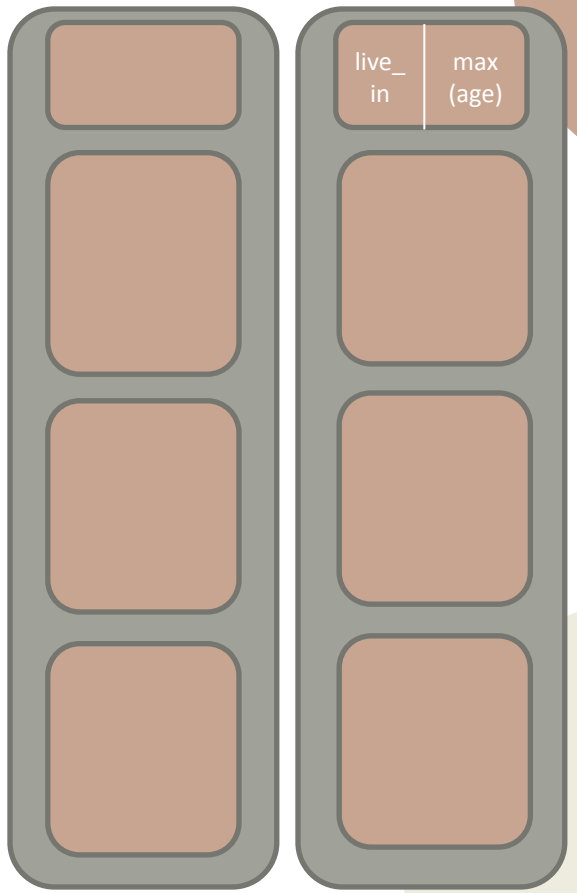
```
>>>max_age_per_live_in =
spanish_name_age.groupBy(persons["live_in"]).max("age")
```



SparkSQL - groupBy

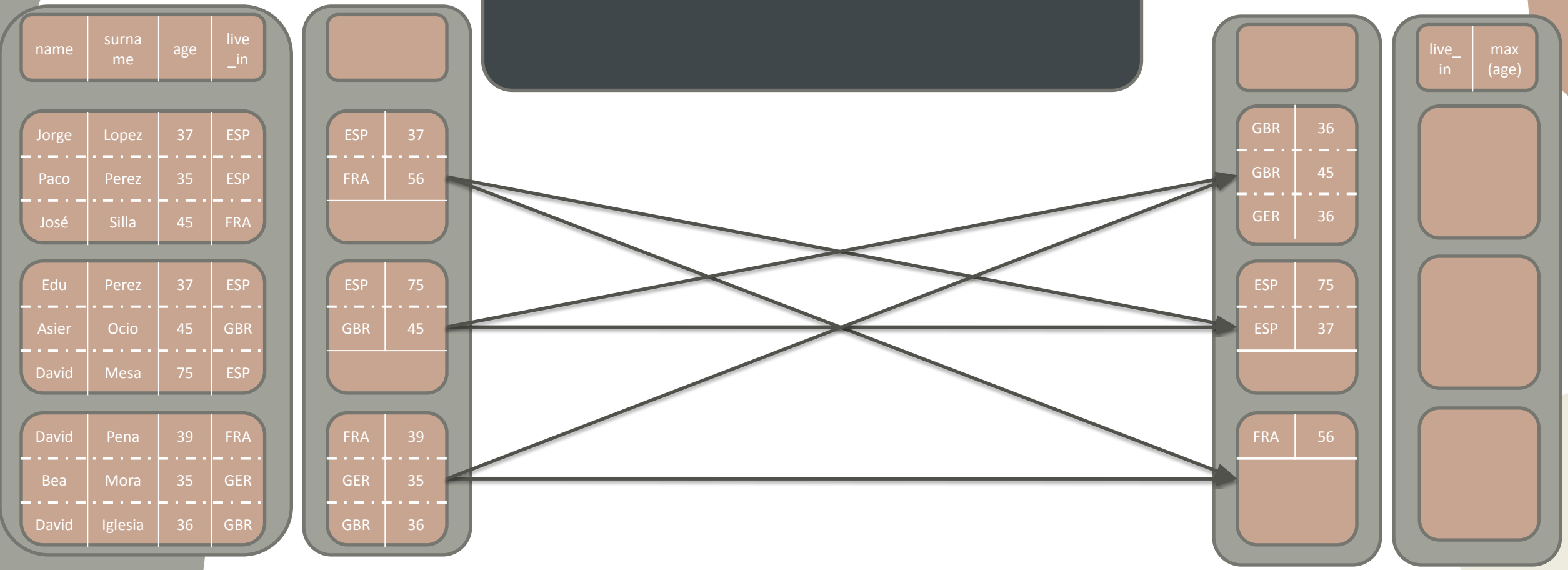
```
>>>max_age_per_live_in =
spanish_name_age.groupBy(persons["live_in"]).max("age")
```





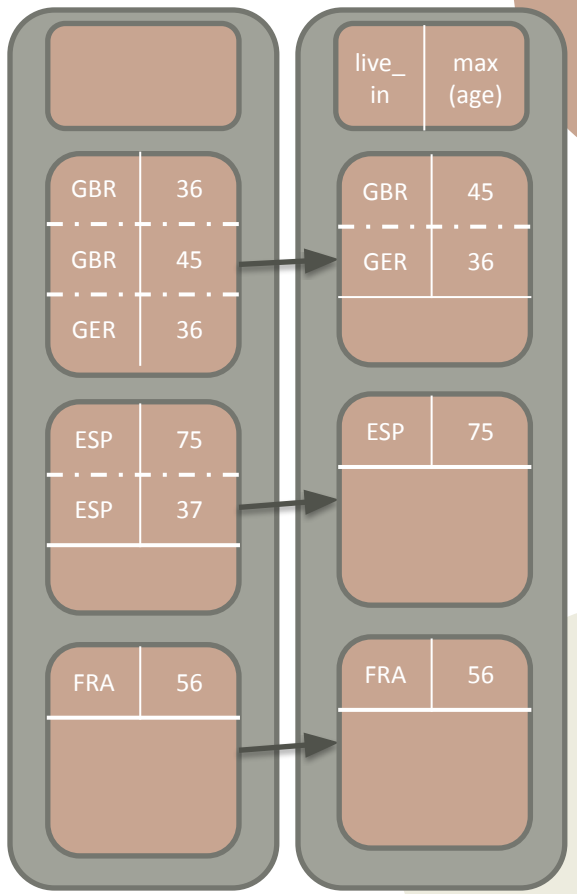
SparkSQL - groupBy

```
>>>max_age_per_live_in =
spanish_name_age.groupBy(persons["live_in"]).max("age")
```



SparkSQL - groupBy

```
>>>max_age_per_live_in =
spanish_name_age.groupBy(persons["live_in"]).max("age")
```



SparkSQL - join

- **join** junta los registros de dos **Dataframes**
- Hay varios tipos de joins disponibles
- El join se hace mediante una condición booleana de columnas de los dos **Dataframes**
- El resultado final tendrá todas las columnas de los dos Dataframes incluidas las columnas
- Es una transformación del tipo Wide y por ende llama a Shuffle y termina un **Stage**

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>teachers_id_with_students =
teachers.join(students,teacher['course'] ==
students['course']).select('id')
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>teachers_id_with_students =
teachers.join(students,teacher['course'] ==
students['course']).select('id')

pyspark.sql.utils.AnalysisException: "Reference
'id' is ambiguous, could be: id, id.;"
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>teachers_with_students =
teachers.join(students_renamed,teacher['course'] ==
students['course'])
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>teachers_with_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'])
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student

id_student	course_student	id	course

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>teachers_with_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'])
```

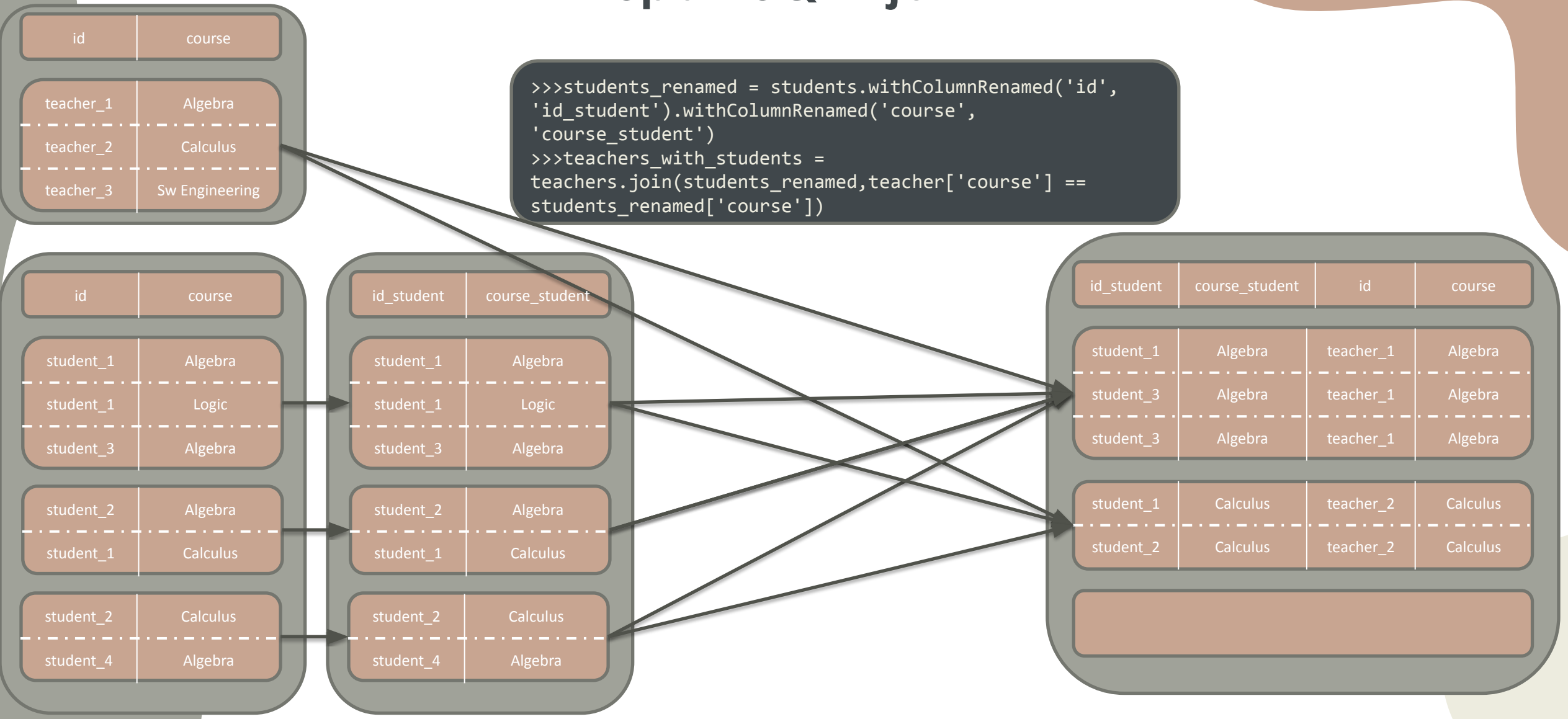
id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student	id	course

SparkSQL - join

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>teachers_with_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'])
```



SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_teachers_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'left')
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_teachers_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'left')
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student

id_student	course_student	id	course

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_teachers_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'left')
```

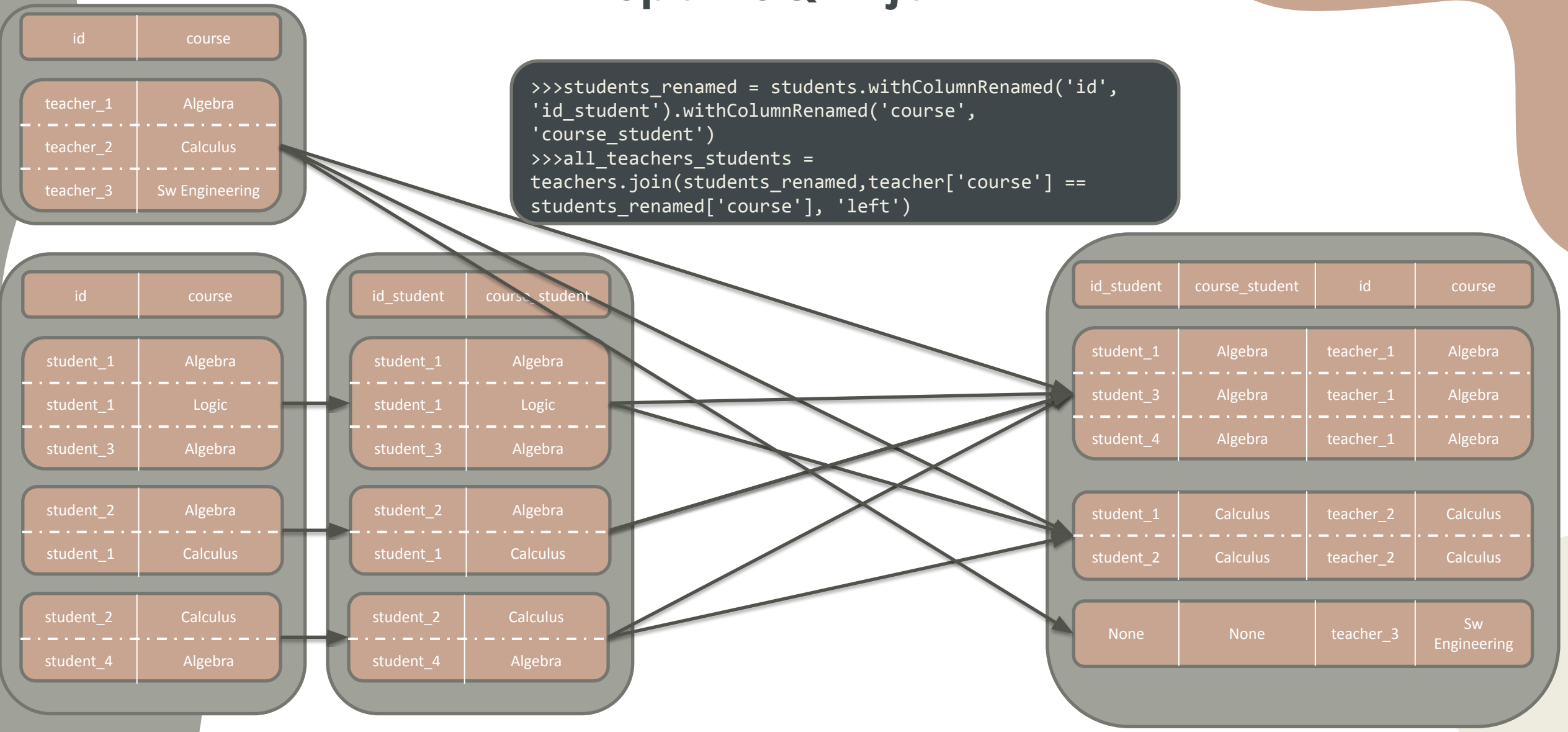
id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student	id	course

SparkSQL - join

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_teachers_students =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'left')
```



SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'right')
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student

id_student	course_student	id	course

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'right')
```

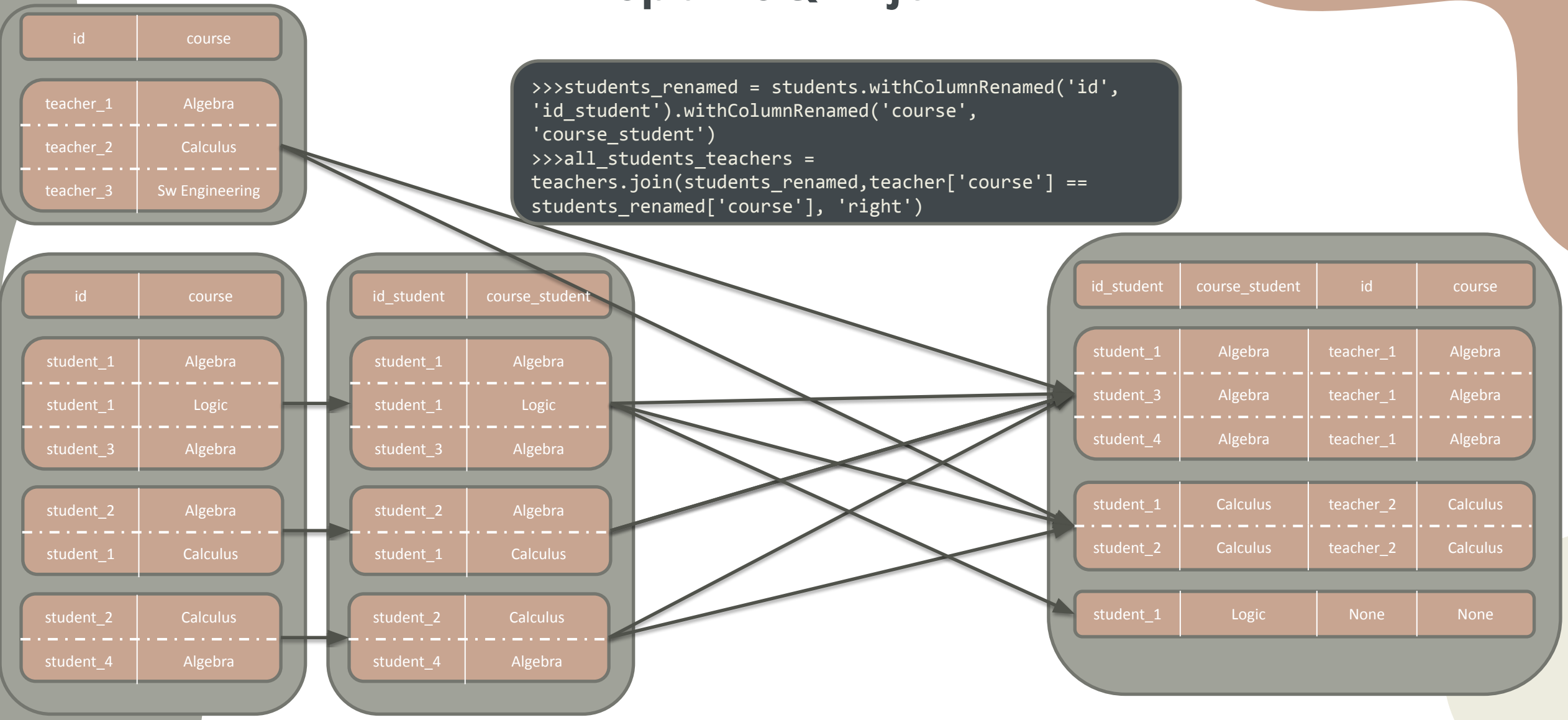
id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student	id	course

SparkSQL - join

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'right')
```



SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_all_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'full')
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_all_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'full')
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student

id_student	course_student	id	course

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_all_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'full')
```

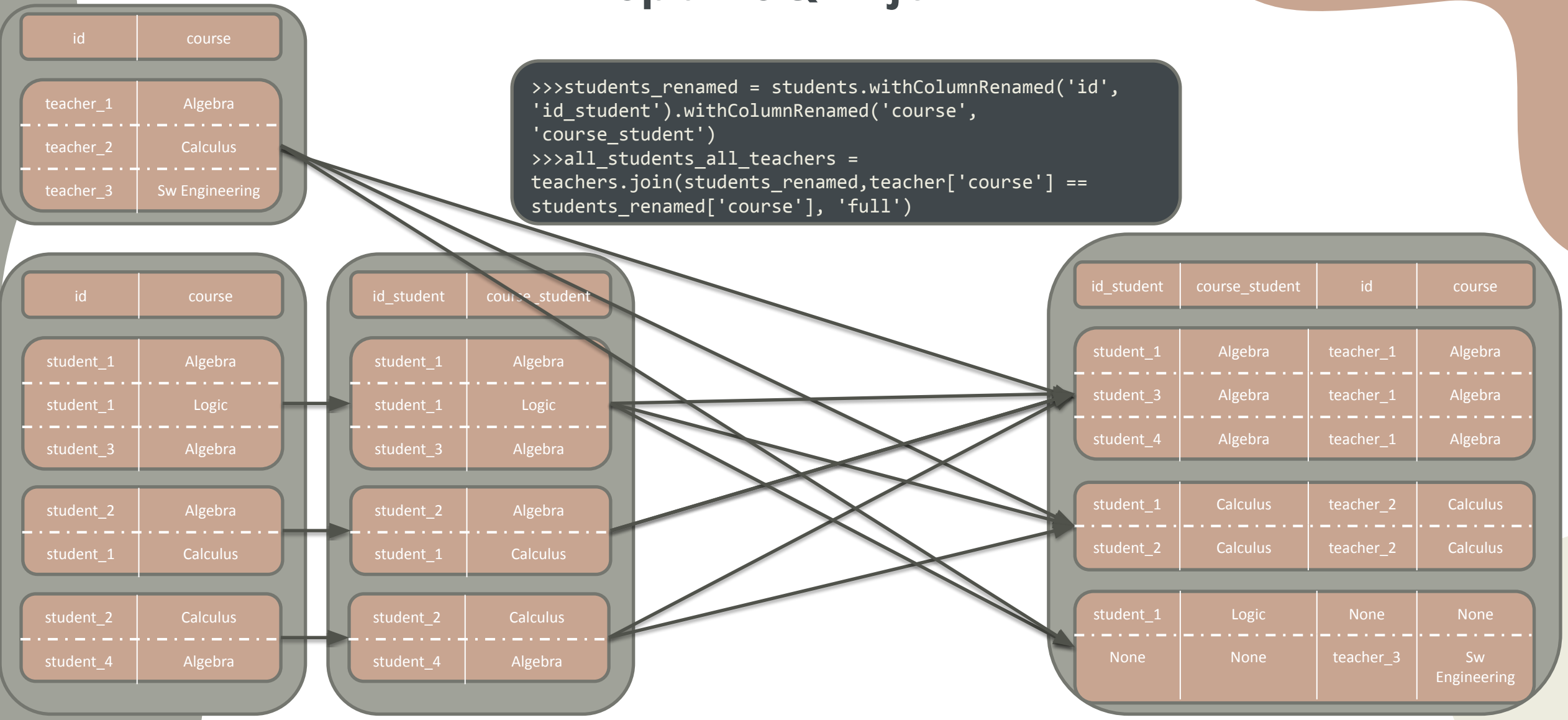
id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id_student	course_student	id	course

SparkSQL - join

```
>>>students_renamed = students.withColumnRenamed('id',
'id_student').withColumnRenamed('course',
'course_student')
>>>all_students_all_teachers =
teachers.join(students_renamed,teacher['course'] ==
students_renamed['course'], 'full')
```



SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>teachers_with_students =
teachers.join(students,teacher['course'] ==
students_renamed['course'], 'left_semi')
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

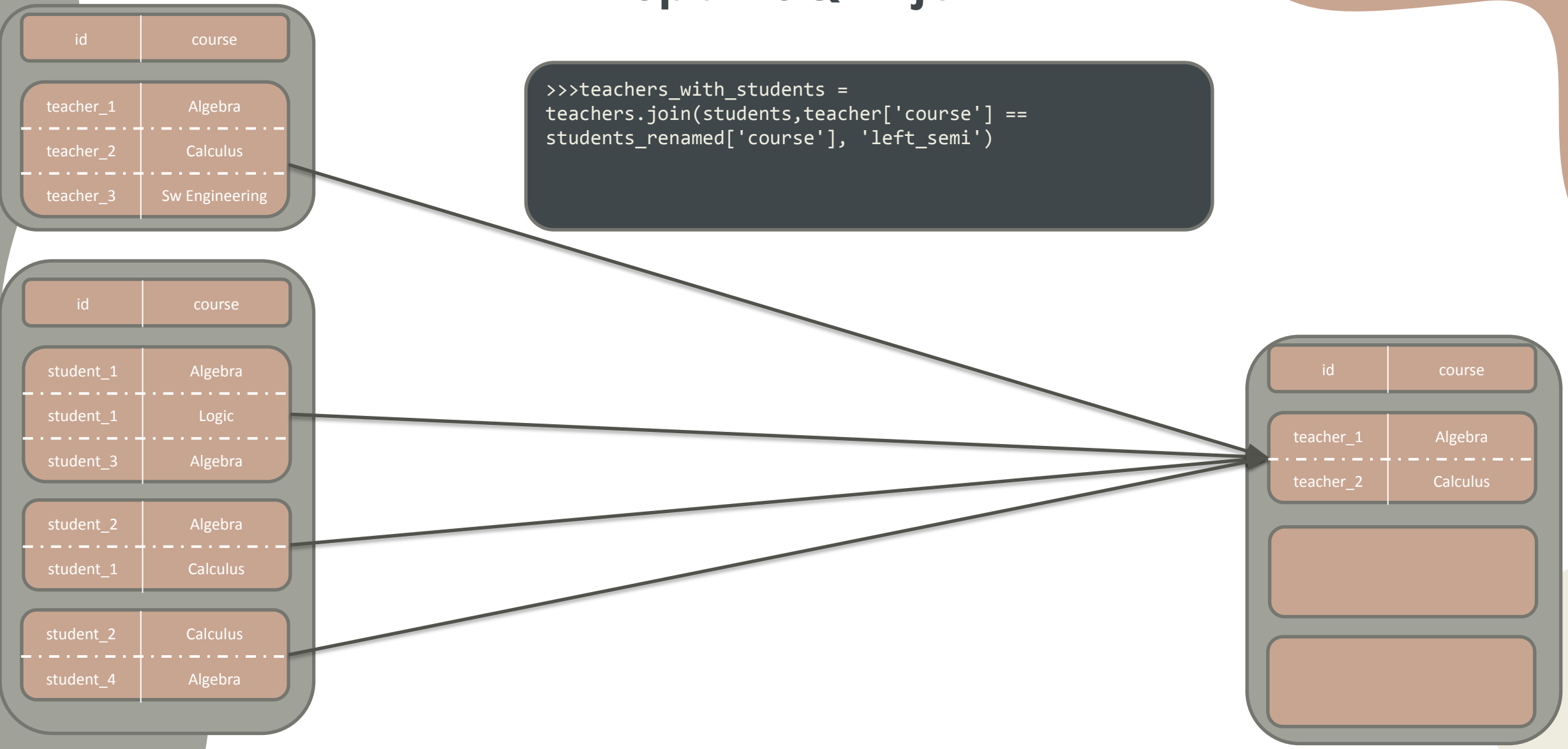
```
>>>teachers_with_students =
teachers.join(students,teacher['course'] ==
students_renamed['course'], 'left_semi')
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id	course

SparkSQL - join

```
>>>teachers_with_students =
teachers.join(students,teacher['course'] ==
students_renamed['course'], 'left_semi')
```



SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

```
>>>teachers_without_students =
teachers.join(students,teacher['course'] ==
students_renamed['course'], 'left_anti')
```

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>teachers_without_students =
teachers.join(students,teacher['course'] ==
students_renamed['course'], 'left_anti')
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id	course

SparkSQL - join

id	course
teacher_1	Algebra
teacher_2	Calculus
teacher_3	Sw Engineering

```
>>>teachers_without_students =
teachers.join(students,teacher['course'] ==
students_renamed['course'], 'left_anti')
```

id	course
student_1	Algebra
student_1	Logic
student_3	Algebra
student_2	Algebra
student_1	Calculus
student_2	Calculus
student_4	Algebra

id	course
teacher_3	Sw Engineering

SparkSQL - funciones de orden superior

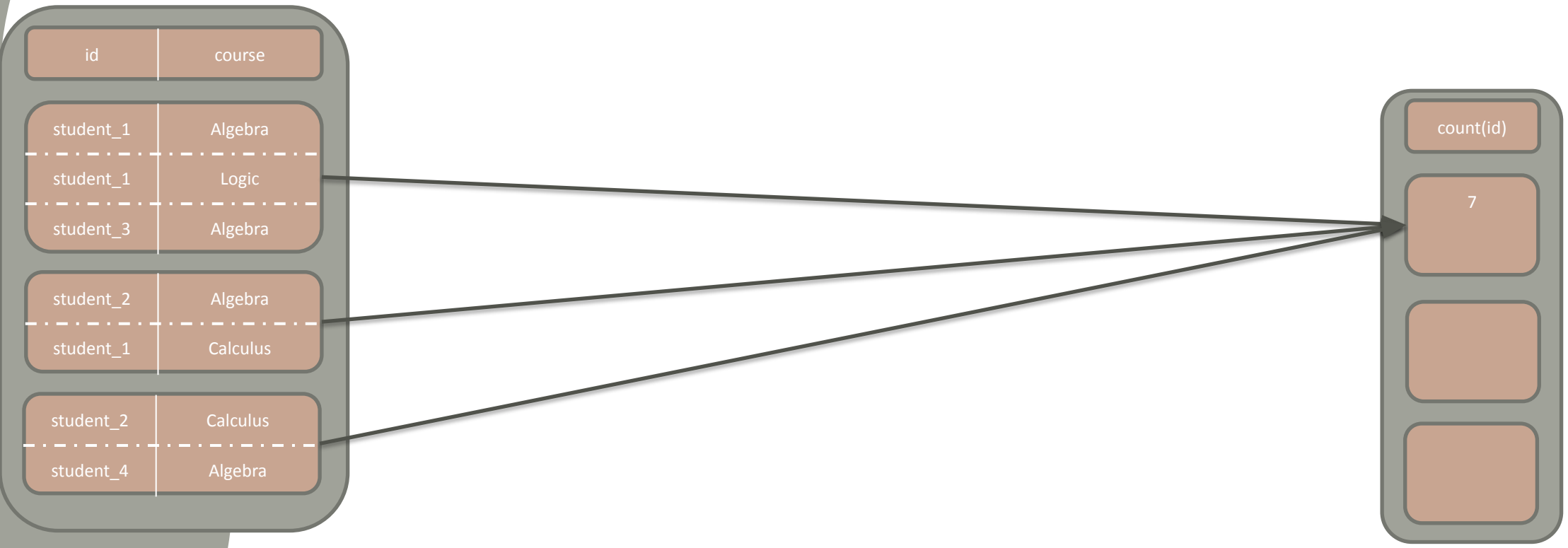
- **SparkSQL** no resuelve todas las casuísticas con las operaciones estandar
- El usuario se puede definir operaciones que interacciones sobre una o varias columnas y cuya lógica se almacene en una columna
- Estas operaciones no serán optimizadas dato que **SparkSQL** no sabe que está ocurriendo dentro de la operación
- Aplicar una UDF es una transformación del tipo Narrow y por ende **no** llama a Shuffle ni termina un **Stage**

SparkSQL - funciones de orden superior

- Conforme aumentan los proyectos que usan **SparkSQL** las **UDF** se van convirtiendo cada vez en más necesarias
- Al tratarse de cajas negras no se pueden optimizar y hacen que el código sea cada vez más lento
- Los desarrolladores de Spark incorporan cada vez más **UDF** a las distribuciones de **Spark**
- Estas funciones son conocidas como funciones de orden superior, en este enlace están las disponibles para la versión del cluster

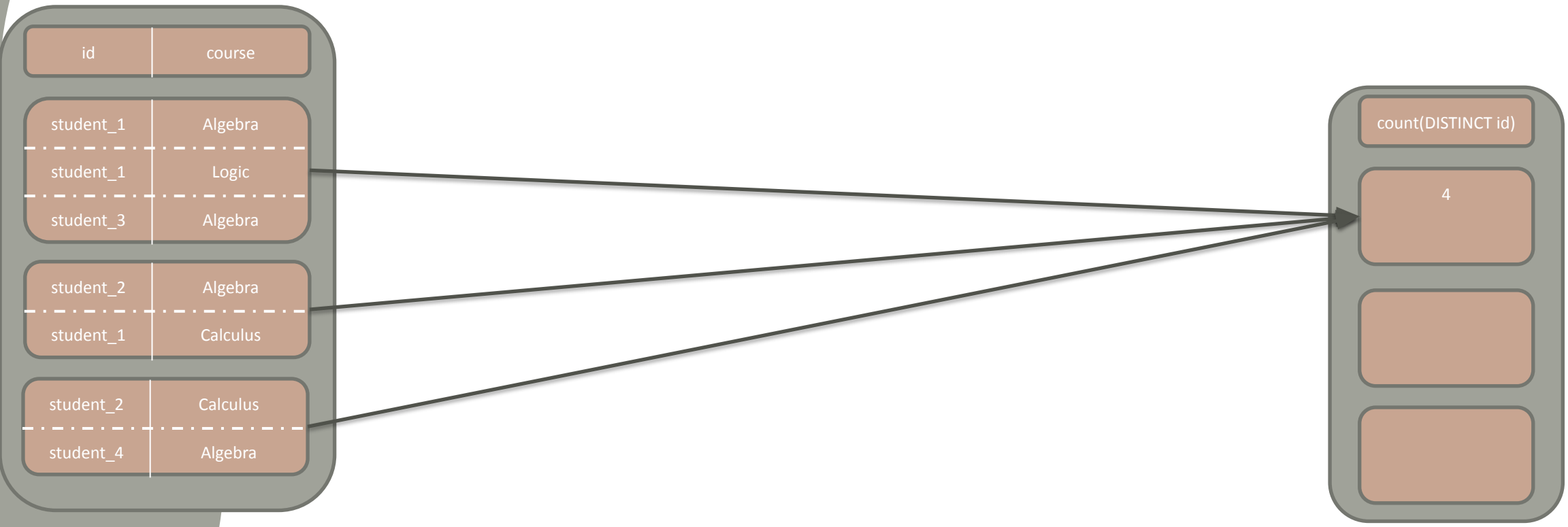
SparkSQL - funciones de orden superior(aggregacion)

```
from pyspark.sql.functions import count

>>>number_of_id = students.agg(count('id'))
```



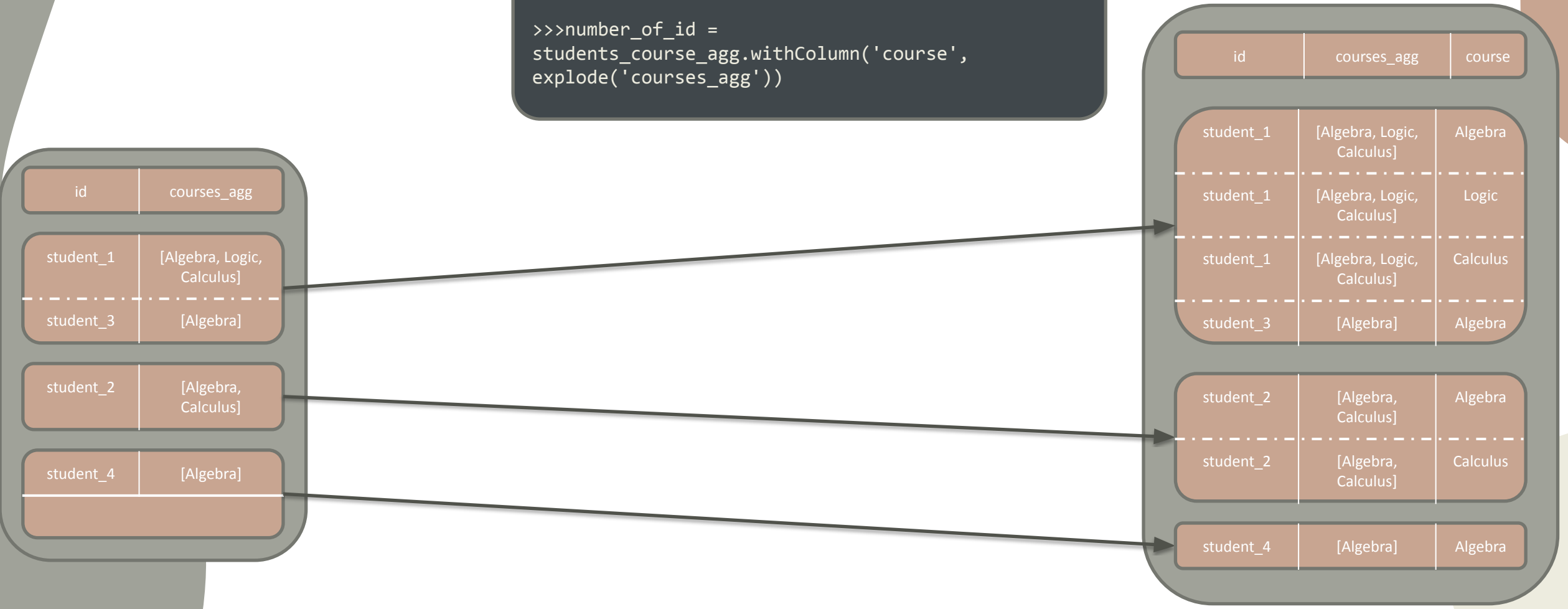
SparkSQL - funciones de orden superior(aggregacion)

```
from pyspark.sql.functions import countDistinct

>>>number_of_distinct_id =
students.agg(countDistinct('id'))
```



SparkSQL - funciones de orden superior(colecciones)

```
from pyspark.sql.functions import explode

>>>number_of_id =
students_course_agg.withColumn('course',
explode('courses_agg'))
```



SparkSQL - funciones de orden superior

- Dentro las limitaciones de **SparkSQL** estaban las funciones que requerían un orden en los valores de una clave
- Para mitigar, en parte, esta carencia se desarrollaron las operaciones de ventana
- Nos permiten asignar a un valor el registro siguiente/anterior a cada fila de un **Dataframe**
- Estas operaciones son transformaciones del tipo **Wide**, llaman a **Shuffle** y terminan un **stage**

SparkSQL - funciones de orden superior(ventana)

```
>>> numbers_bucket = spark.range(9).withColumn("bucket", col('id') % 3)


>>> numbers_bucket.show()
+---+------+
| id|bucket|
+---+------+
|  0|     0|
|  1|     1|
|  2|     2|
|  3|     0|
|  4|     1|
|  5|     2|
|  6|     0|
|  7|     1|
|  8|     2|
+---+------+
```

SparkSQL - funciones de orden superior(ventana)

```
>>> from pyspark.sql.window import Window
>>> from pyspark.sql.functions import rank

>>> byBucket = Window.partitionBy(col('bucket')).orderBy(col('id'))

>>>numbers_bucket_raked_by_Bucket = numbers_bucket.withColumn("rank", rank().over(byBucket))
>>>numbers_bucket_raked_by_Bucket.show()
+---+------+----+
| id|bucket|rank|
+---+------+----+
|  0|     0|   1|
|  3|     0|   2|
|  6|     0|   3|
|  1|     1|   1|
|  4|     1|   2|
|  7|     1|   3|
|  2|     2|   1|
|  5|     2|   2|
|  8|     2|   3|
+---+------+----+
```

SparkSQL - funciones de orden superior(ventana)

```
>>> window_data = [('A', 20211101, 11), ('A', 20210101, 1), ('A', 20220101, 100), ('B',
20211101, 11), ('C', 20211101, 22), ('C', 20210101, 2)]

>>> window_data_column_names = ('id', 'date', 'value')

>>> window_df = spark.createDataFrame(winow_data, window_data_column_names)

>>> window_df.show()
+---+--------+-----+
| id|    date|value|
+---+--------+-----+
|  A|20211101|   11|
|  A|20210101|    1|
|  A|20220101|  100|
|  B|20211101|   11|
|  C|20211101|   22|
|  C|20210101|    2|
+---+--------+-----+
```

SparkSQL - funciones de orden superior(ventana)

```
>>> from pyspark.sql.window import Window
>>> from pyspark.sql.functions import lead

>>> byIdSortByDate = Window.partitionBy(col('id')).orderBy(col('date'))

>>> window_df.withColumn('next', lead(col("value")).over(byIdSortByDate)).show()
+---+--------+-----+----+
| id|    date|value|next|
+---+--------+-----+----+
|  B|20211101|   11|null|
|  C|20210101|    2|  22|
|  C|20211101|   22|null|
|  A|20210101|    1|  11|
|  A|20211101|   11| 100|
|  A|20220101|  100|null|
+---+--------+-----+----+
```

SparkSQL - funciones de orden superior(ventana)

```
>>> from pyspark.sql.window import Window
>>> from pyspark.sql.functions import lag

>>> byIdSortByDate = Window.partitionBy(col('id')).orderBy(col('date'))

>>> window_df.withColumn('prev', lag(col("value")).over(byIdSortByDate)).show()
+---+--------+-----+----+
| id|    date|value|prev|
+---+--------+-----+----+
|  B|20211101|   11|null|
|  C|20210101|    2|null|
|  C|20211101|   22|   2|
|  A|20210101|    1|null|
|  A|20211101|   11|   1|
|  A|20220101|  100|  11|
+---+--------+-----+----+
```

Apache SparkSQL