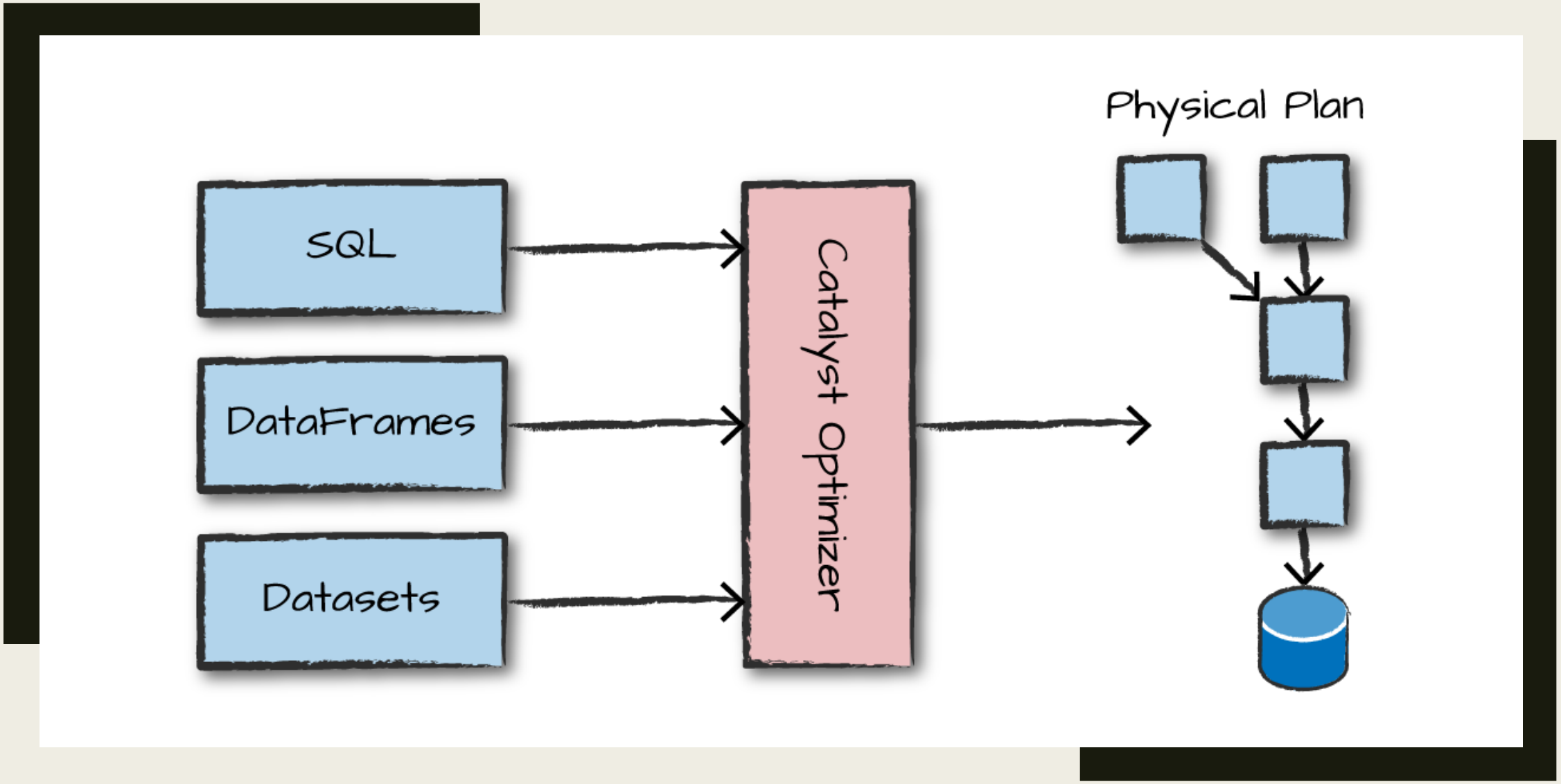
Big Data

PLANOS DE EXECUÇÃO DO SPARK

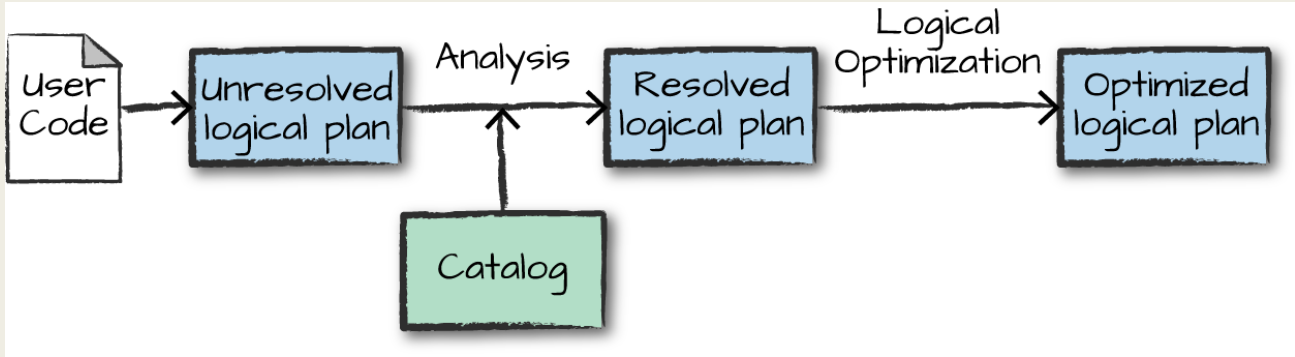
Planos de execução

- As transformações do Spark são declarativas
  - Expectativa do usuário vs Processamento
- *Driver* recebe sequência de operações em código para DataFrame/SQL.
- Spark converte em um *Logical Plan*.
- Spark transforma o *Logical Plan* em um *Physical Plan*, aplicando as otimizações ao longo da transformação.
- Spark então executa o *Physical Plan* (operações com RDD) no cluster.

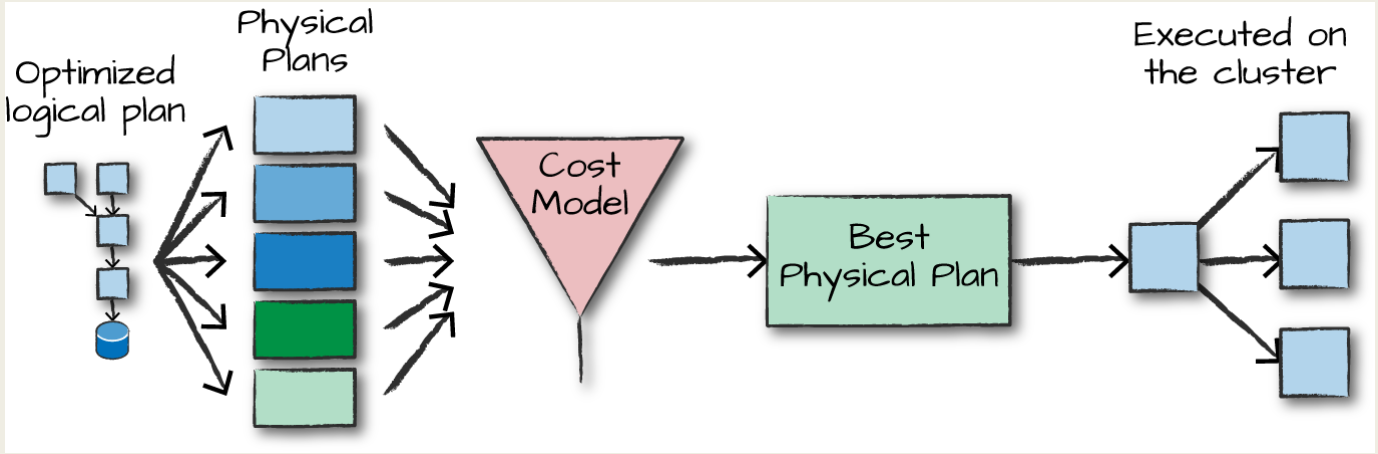

SQL
DataFrames
Datasets
Catalyst Optimizer
Physical Plan

Processamento do plano lógico

- Plano lógico não possui relação com *executors* e *driver*
- Código do usuário é o plano lógico não resolvido
- Catálogo: repositório de tabelas e informações de Data Frames
- Plano lógico resolvido assegura existência das tabelas e colunas referenciadas (e seus tipos adequados às operações)
- Plano lógico otimizado reorganiza a ordem das operações



Processamento do plano físico



- Determina de que forma o plano lógico será executado no “ambiente físico”
- Avalia diferentes planos candidatos e seleciona o melhor de acordo com critérios de desempenho no ambiente
- Plano final é transformado em operações RDD

Jobs, Actions e Stages

- *Stages* representam grupos de tarefas (*tasks*) que podem ser executadas em conjunto em um grupo de *workers*.
- O *Spark* tentará agrupar em um mesmo *Stage* o máximo de transformações quanto possíveis, mas criará um novo *Stage* a cada operação de *shuffle*.
- *Shuffle* são operações que reparticionam um DataFrame, como ordenações, group by, e alguns tipos de Join.
  - *Operações que requerem o envio de registros de mesma chave para um mesmo nodo.*
- O Spark gerencia a criação e sequenciamento dos *Stages*.

Jobs, Actions e Stages

- Cada action dispara 1 job
- Job composto por estágios (Stages)
- Arquivo 00-exemplo_stages.py
  - *Stage 1: x Tasks* *# Criação de Data Frame (tamanho default)*
  - *Stage 2: x Tasks* *# Criação de Data Frame (tamanho default)*
  - *Stage 3: 6 Tasks* *# Repartition step12*
  - *Stage 4: 5 Tasks* *# Repartition step1*
  - *Stage 5: 200 Tasks* *# Saída do Join com default 200*
  - *Stage 6: 1 Task* *# sum resulta em um único valor*

Jobs, Actions e Stages

- spark.sql.shuffle.partitions: default 200
- spark.conf.set("spark.sql.shuffle.partitions", 50)
  - *Número de partições deve ser maior que o número de executors no cluster.*
    Conforme a carga de trabalho, o ideal é criar 2x, 3x, ou mais para reduzir o tamanho de cada task/executor.

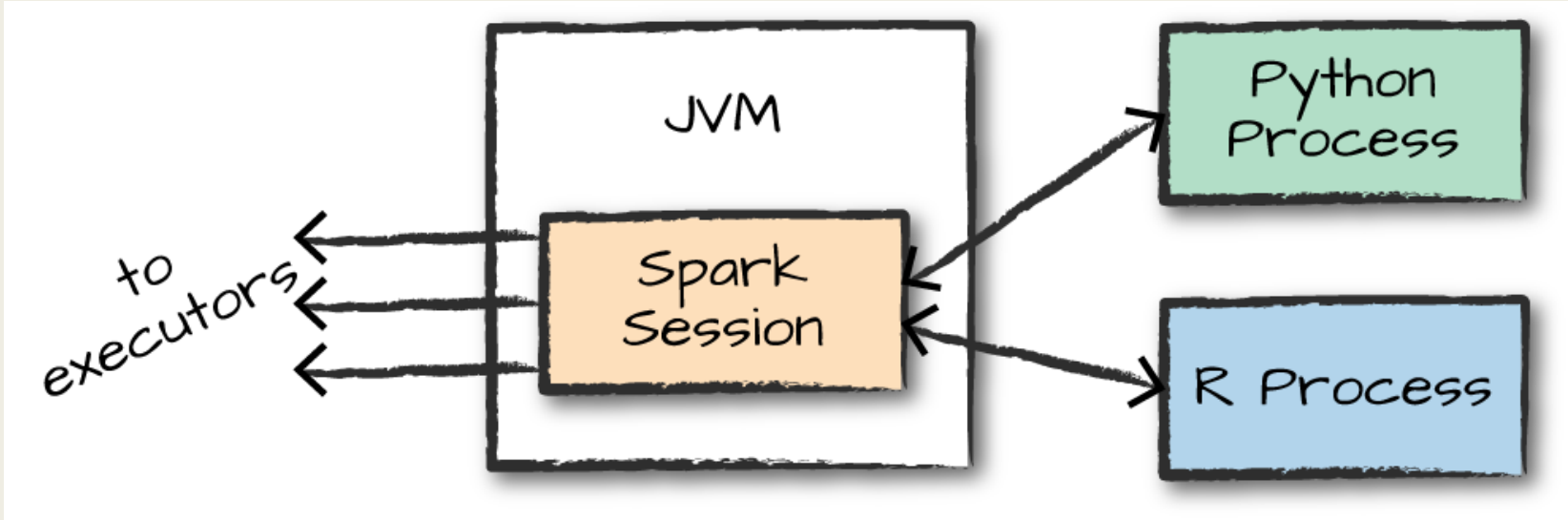
Tasks

- *Stages* são divididos em *tasks*.
- Cada *task* corresponde a uma partição que será processada em um único executor.
  - *Se existir somente uma partição, ela será processada em uma única task.*
  - *Se existirem 1000 partições, 1000 tasks poderão ser processadas em paralelo.*
- *Task* é uma unidade de computação (*executor*) aplicada a uma unidade de dados (*partição*).

Materiais extras sobre o Catalyst

- https://www.youtube.com/watch?v=GDeePbbCz2g
  - A Deep Dive into the Catalyst Optimizer
- https://www.youtube.com/watch?v=5ajs8EIPWGI
  - Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal
- https://www.youtube.com/watch?v=qSaS99TiCM
  - Cost Based Optimizer in Apache Spark 2.2

PYSPARK



LEMBRANDO

Integração Pandas & PySpark

```
import pandas as pd

df = pd.DataFrame({"first":range(200), "second":range(50,250)})

sparkDF = spark.createDataFrame(df)

newPDF = sparkDF.toPandas()

newPDF.head()

sparkDF.show(5)
```

Documentação

- http://spark.apache.org/docs/2.3.0/api/python/pyspark.sql.html
  - *Funções disponíveis para uso com Data Frames e SQL*
- http://spark.apache.org/docs/latest/sql-programming-guide.html
  - *APIs estruturadas*