CSCI 443: LECTURE 23 EXPLORATORY DATA ANALYSIS (EDA) AND DATA CLEANING

Professor David Harrison

DATES OF INTEREST

April 25	HW5 handed out (last night)
May 2	HW5 due (Thursday)

worth 6% of your grade which is under the 10% limit for dead week.

May 6-10	Finals week (M-F)
May 7	Final (Tuesday, 4:00pm)

OFFICE HOURS

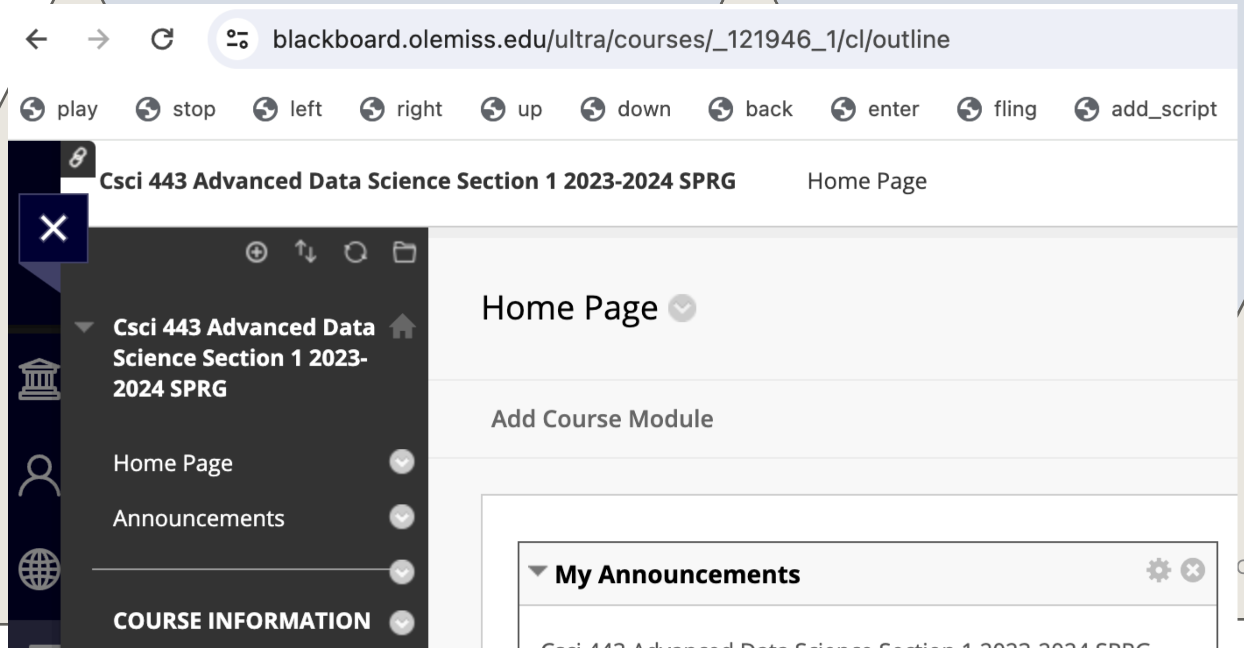
Tuesday	4:00–5:00 PM
Wednesday	12:30–2:30 PM

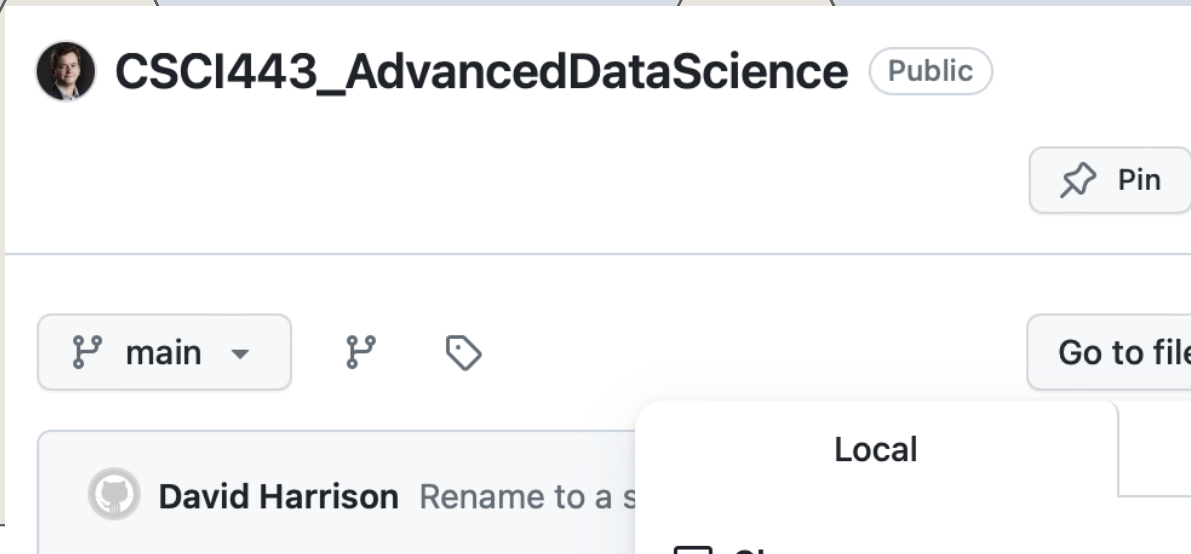
BLACKBOARD & GITHUB

Slides and a jupyter notebook for lectures 20 and 21 are on blackboard and in GitHub.

The project is at

https://github.com/dosirrah/CSCI443_AdvancedDataScience





TODAY

- Exploratory Data Analysis
- Data Cleaning

PREVIOUSLY: SAID THIS WAS DUMB

For Pandas and Pandas on Spark DataFrame

```
states = customers_df["customer_state"][:10]
```

```
Command took 0.21 seconds -- by harrison@cs.olemis u at
```

Cmd 12

```
print(states)
```

PREVIOUSLY: SAID THIS WAS DUMB

.

For Pandas I ran this many times and I saw no significant difference between

```
states = customers_df["customer_state"][:10]
```

and

```
states = customers_df["customer_state"].head(10)
```

PREVIOUSLY: SAID THIS WAS

.

For Pandas I ran this many times and I saw no significant difference between

```
states = customers_df["customer_state"][:10]
```

and

```
states = customers_df["customer_state"].head(10)
```

PREVIOUSLY: SAID THIS WAS

Pandas uses numpy underneath.

Slicing does not allocate a new array.

Maintains reference to part of existing numpy array.

Thus creating a slice takes negligible time.

```
>>> import numpy as np
>>> np.array([4,6,2,6,2,1,1,9])
array([4, 6, 2, 6, 2, 1, 1, 9])
>>> arr = np.array([4,6,2,6,2,1,1,9])
>>> slice = arr[:5]
>>> type(slice)
<class 'numpy.ndarray'>
>>> arr[0]
4
>>> slice[0] = 5
>>> arr[0]
5
>>> 
```

PREVIOUSLY: SAID THIS WAS

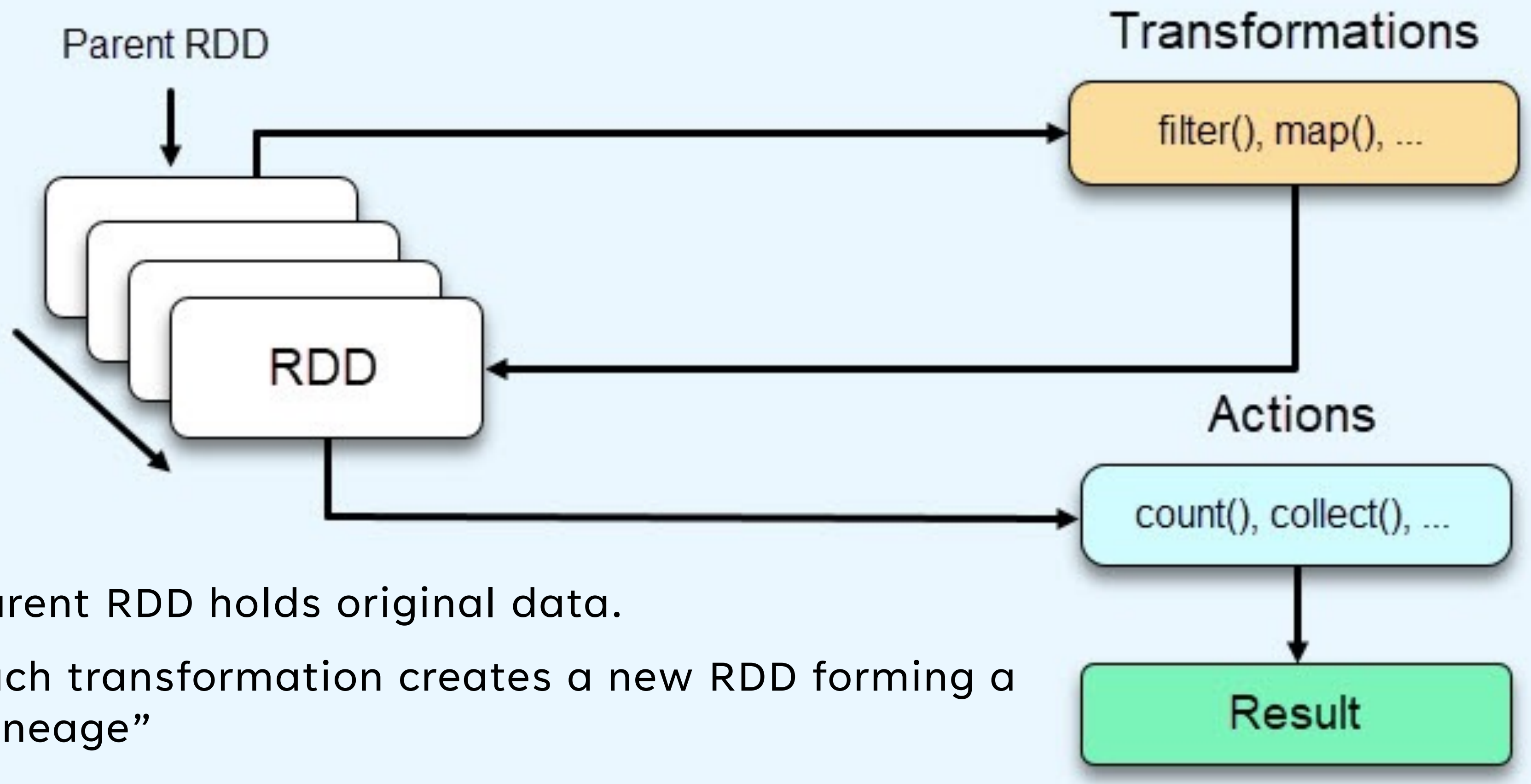
With Pands-on-Spark

```
states = customers_df["customer_state"][:10]
```

Catalyst Optimizer will likely change it to

```
states = customers_df["customer_state"].head(10)
```

PREVIOUSLY: SEQUENCE OF TRANSFORMATIONS

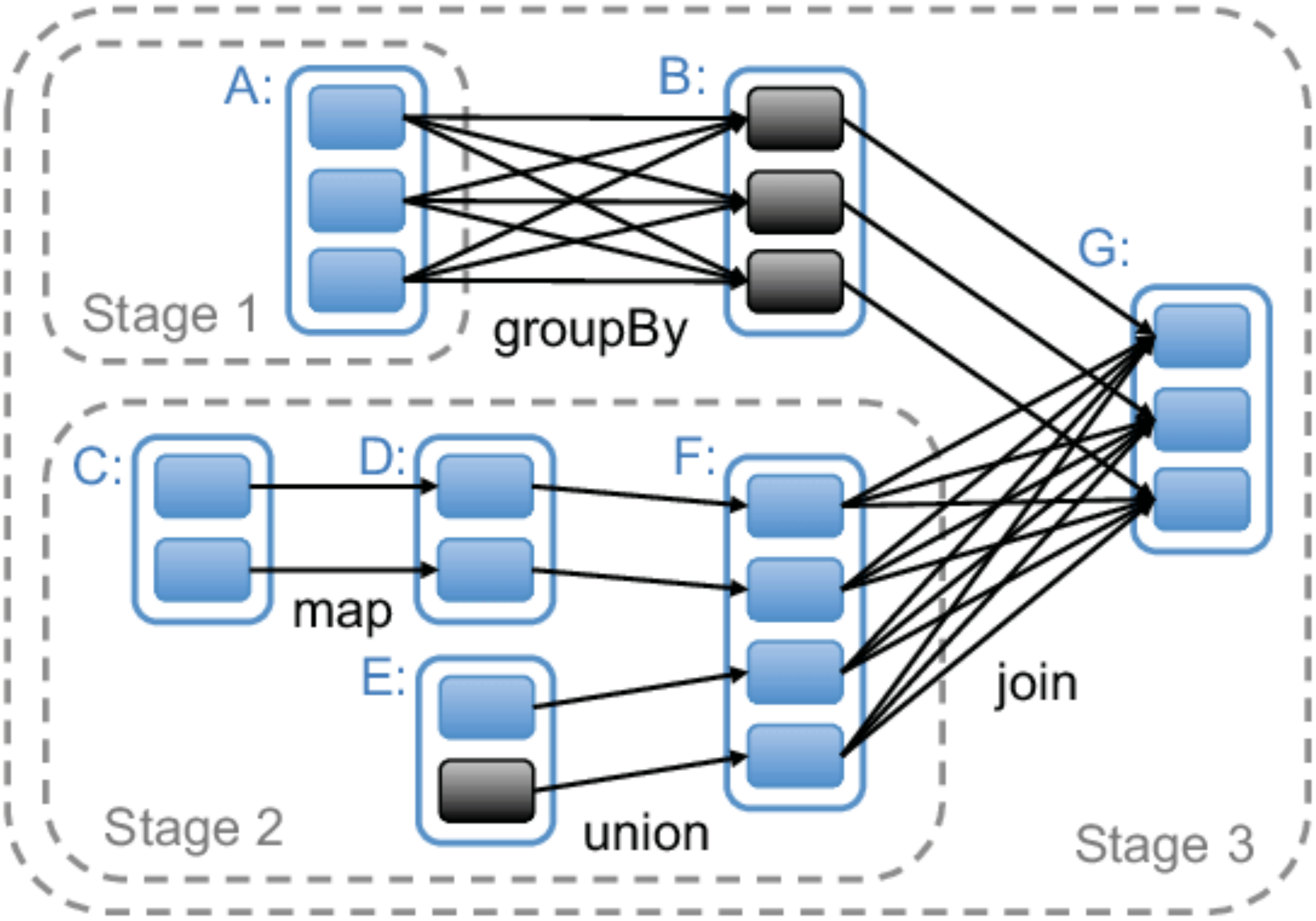


Parent RDD holds original data.

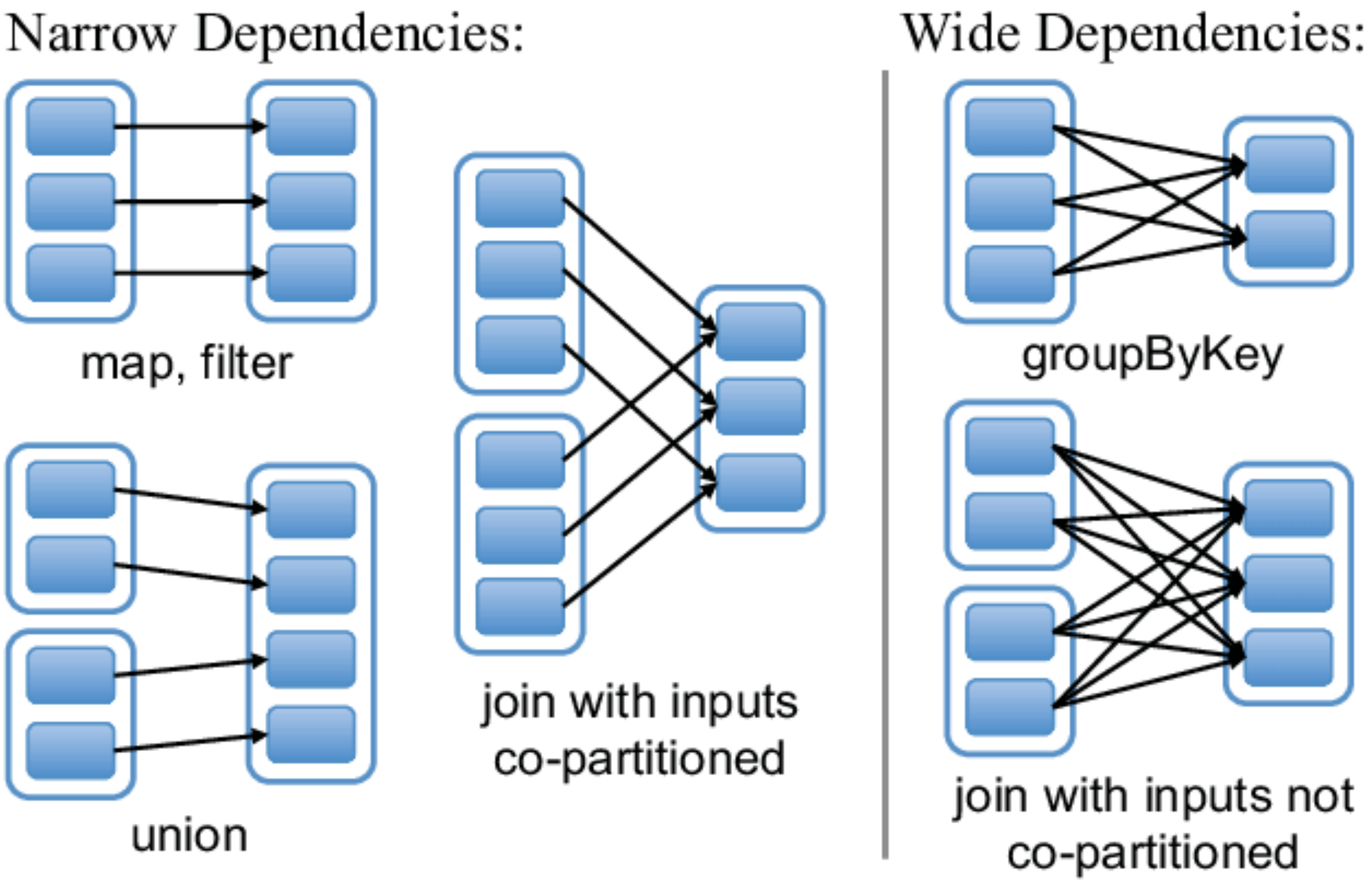
Each transformation creates a new RDD forming a "lineage"

Actions trigger computation.

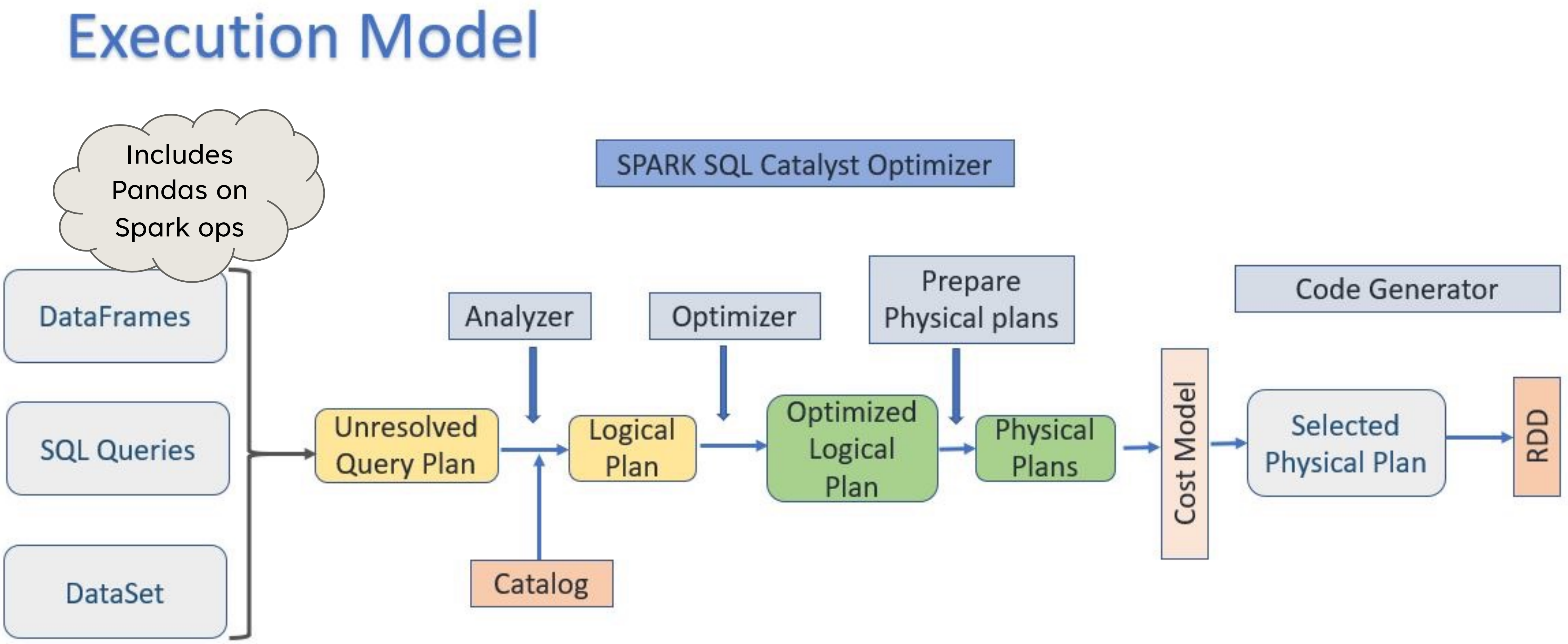
PREVIOUSLY: TRANSFORMATIONS FORM A DAG



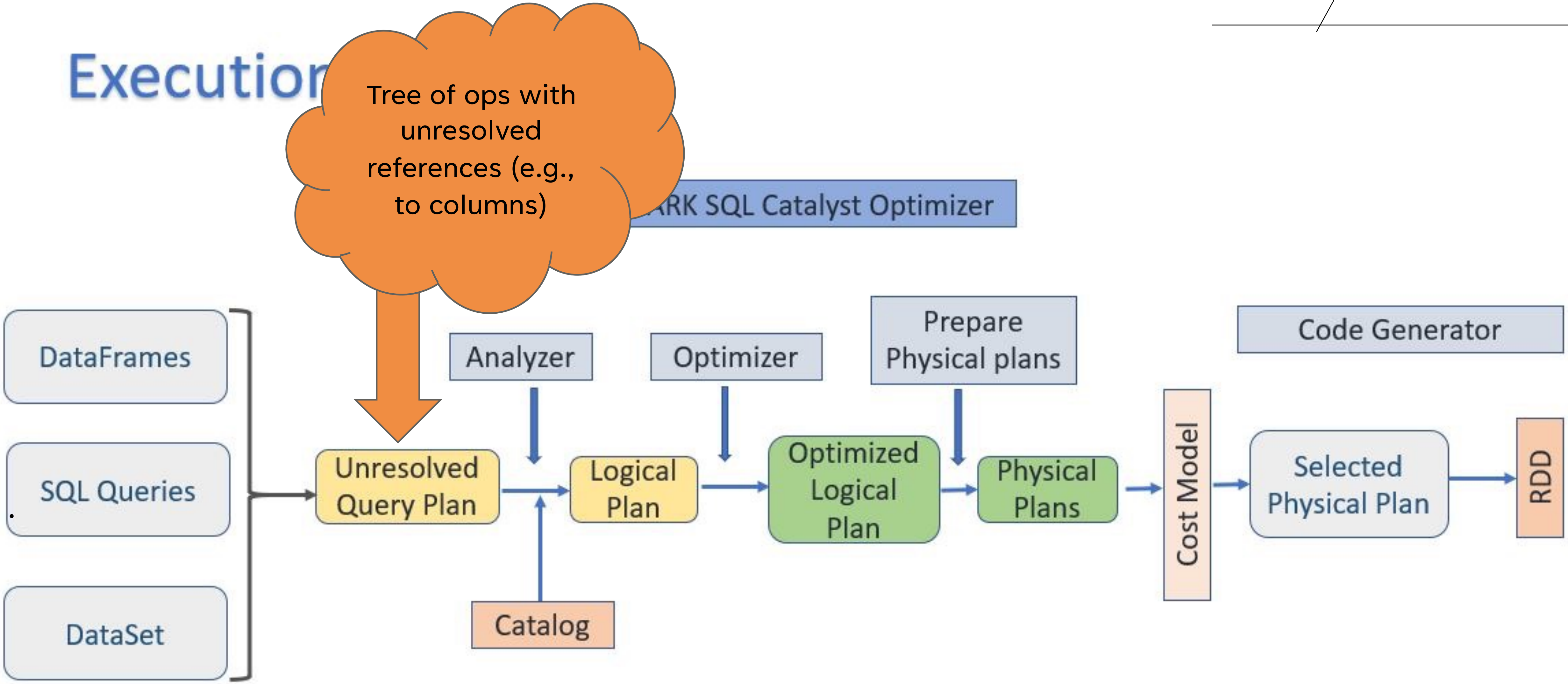
PREVIOUSLY: DIFFERENT TRANSFORMATIONS, DIFFERENT DEPENDENCIES



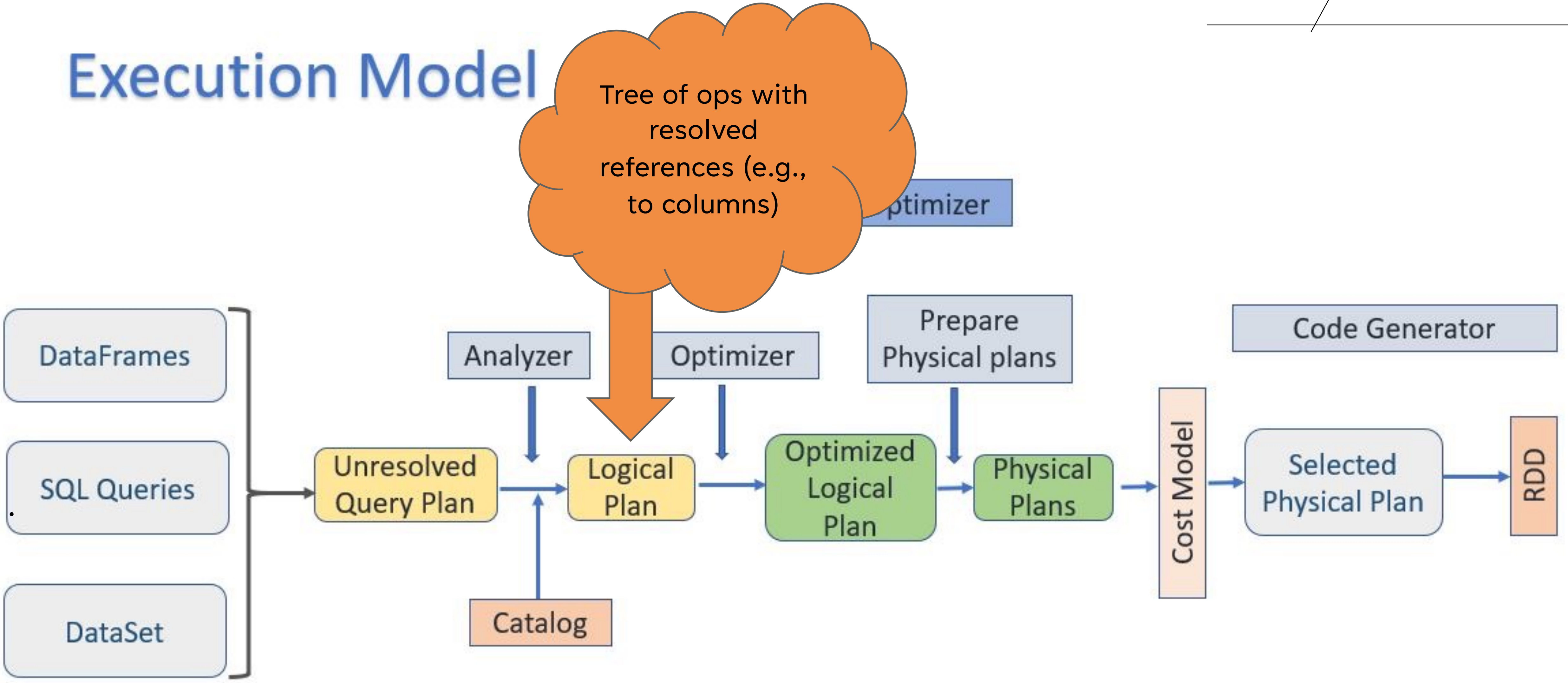
PREVIOUSLY: CATALYST OPTIMIZER



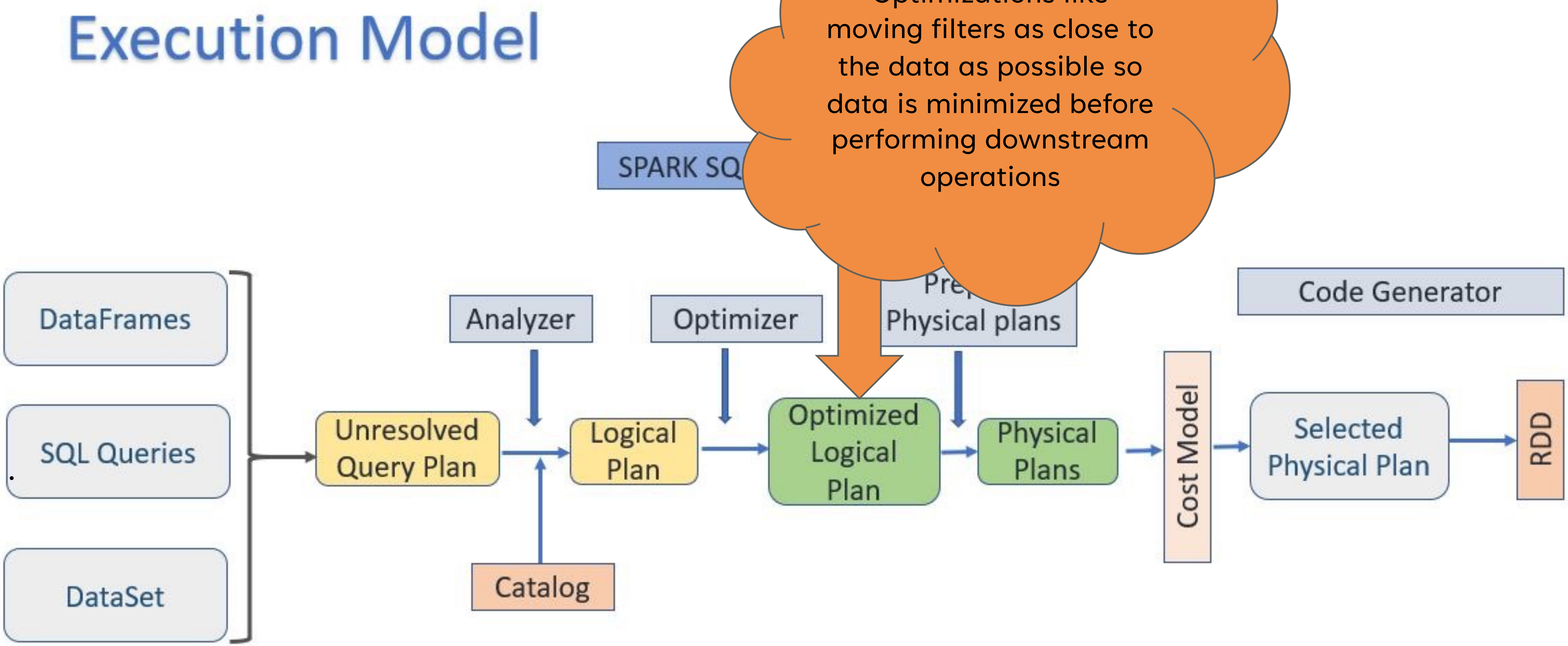
PREVIOUSLY: CATALYST OPTIMIZER



CATALYST OPTIMIZER (PREVIOUSLY MISSING SLIDE)

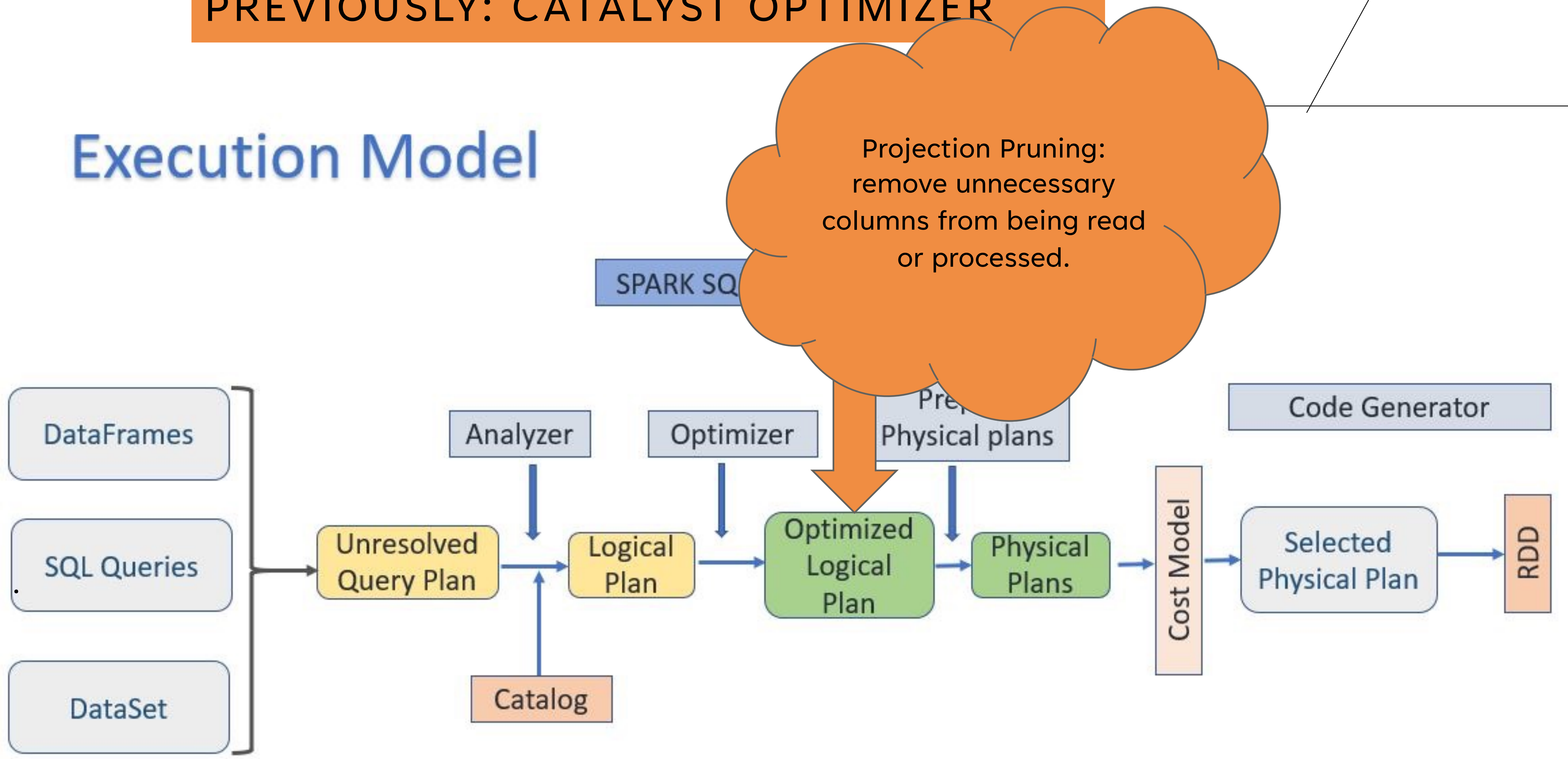


PREVIOUSLY: CATALYST OPTIMIZER

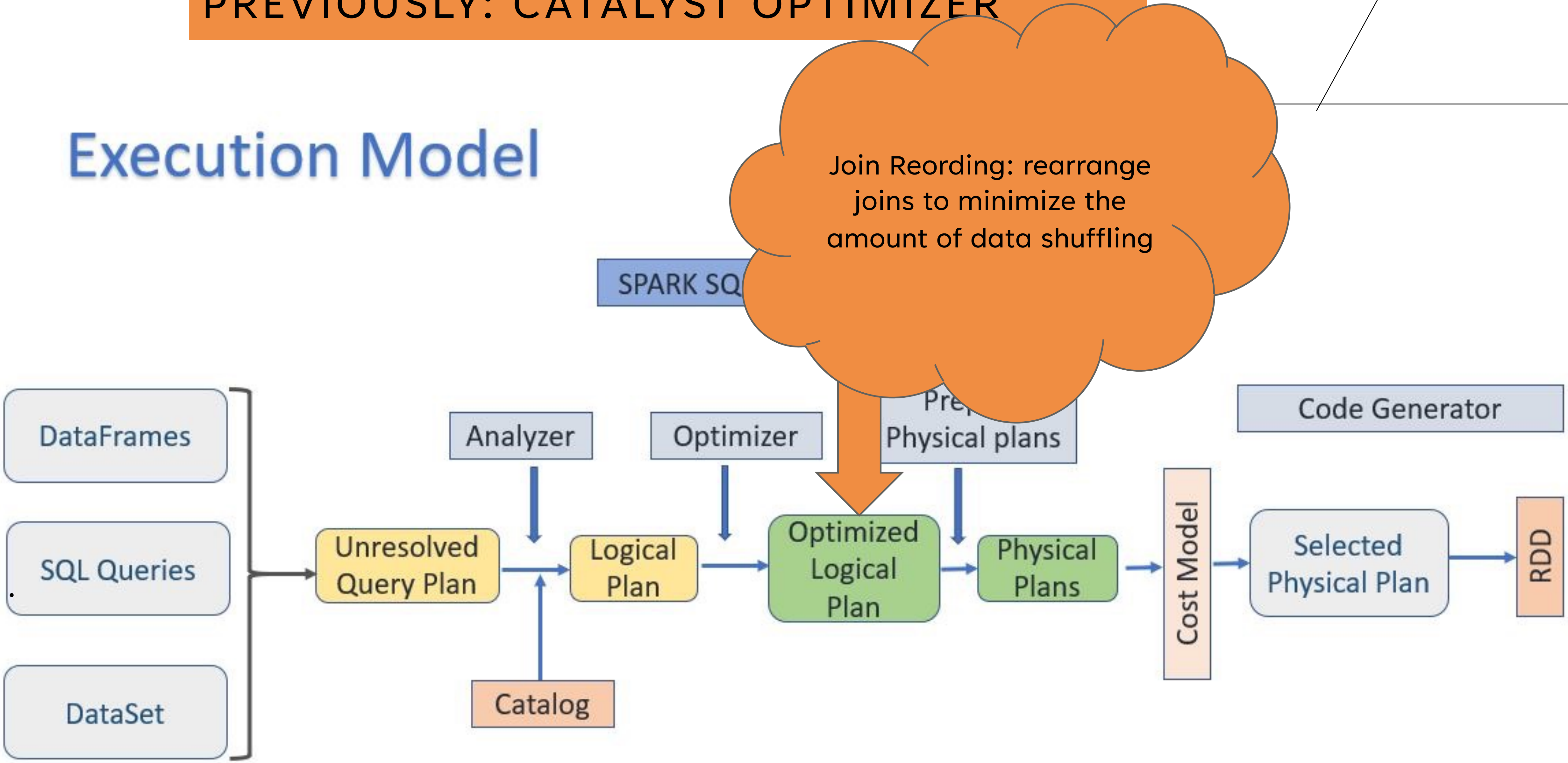


Optimizations like moving filters as close to the data as possible so data is minimized before performing downstream operations

PREVIOUSLY: CATALYST OPTIMIZER

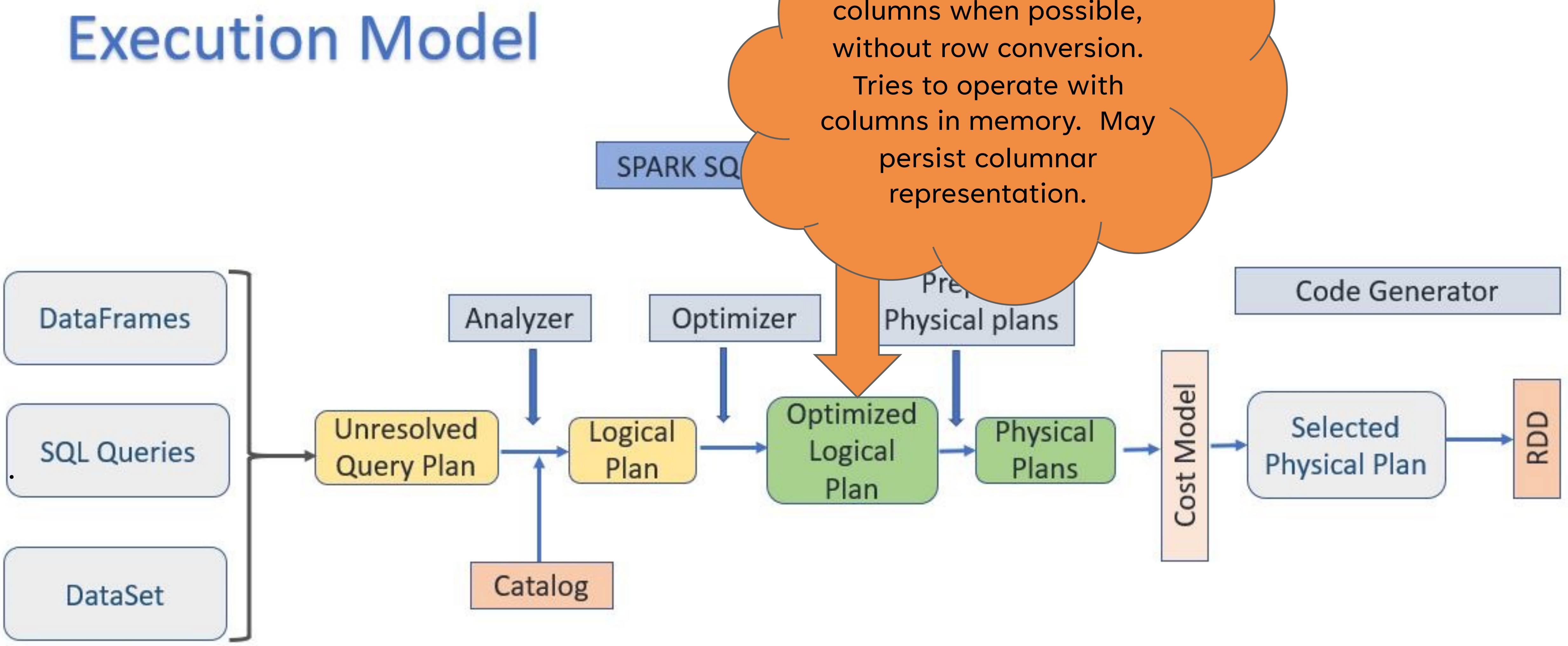


PREVIOUSLY: CATALYST OPTIMIZER

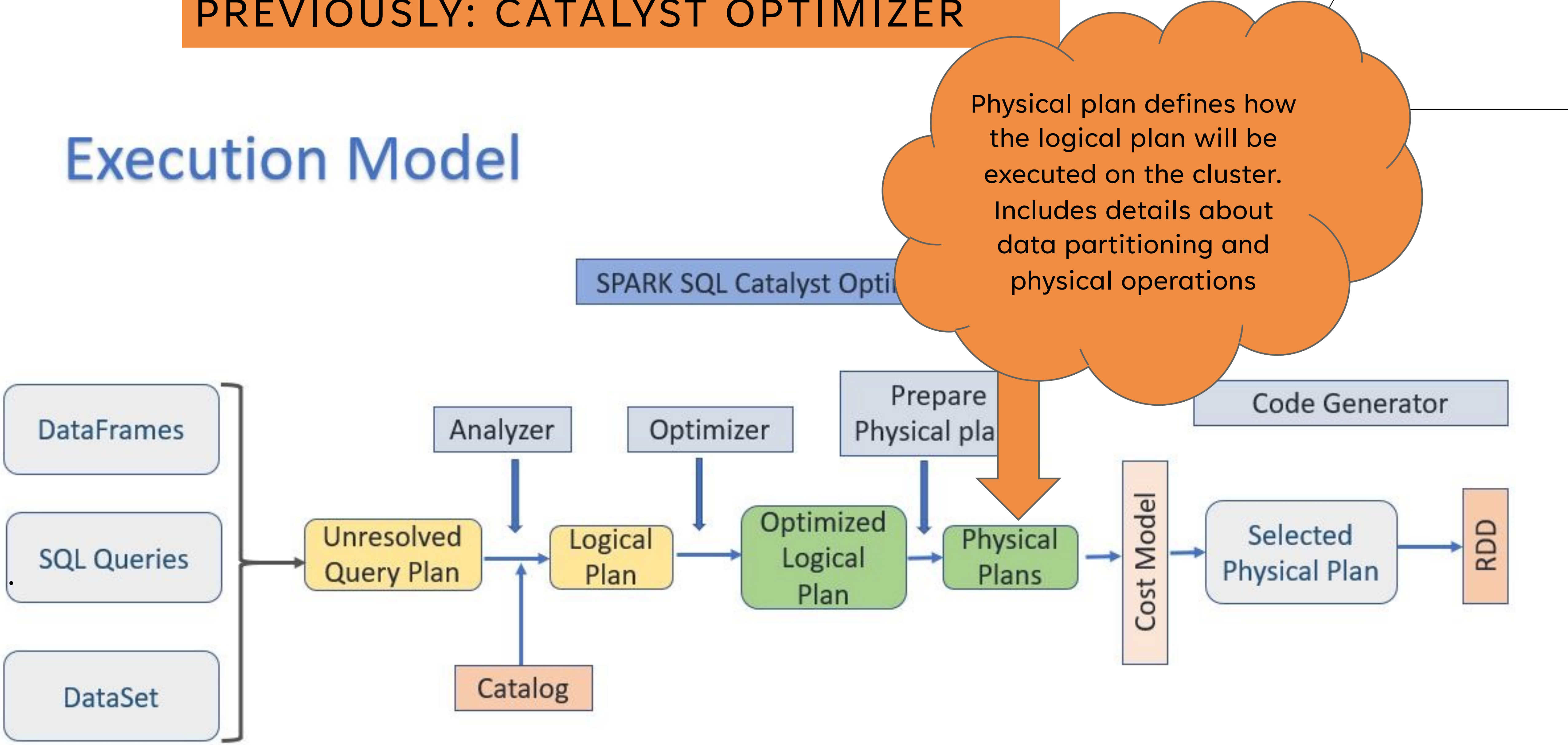


PREVIOUSLY: CATALYST OPTIMIZER

Columnar processing: use columns when possible, without row conversion. Tries to operate with columns in memory. May persist columnar representation.

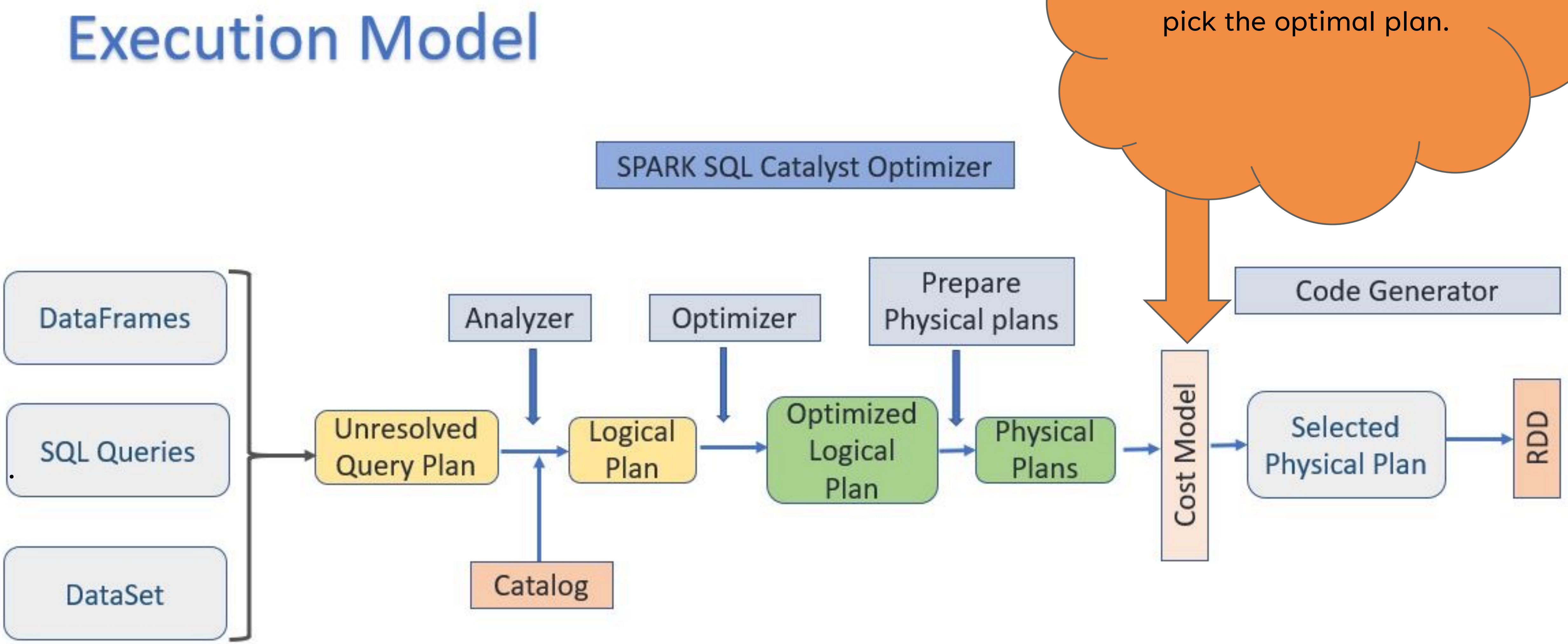


PREVIOUSLY: CATALYST OPTIMIZER



PREVIOUSLY: CATALYST OPTIMIZER

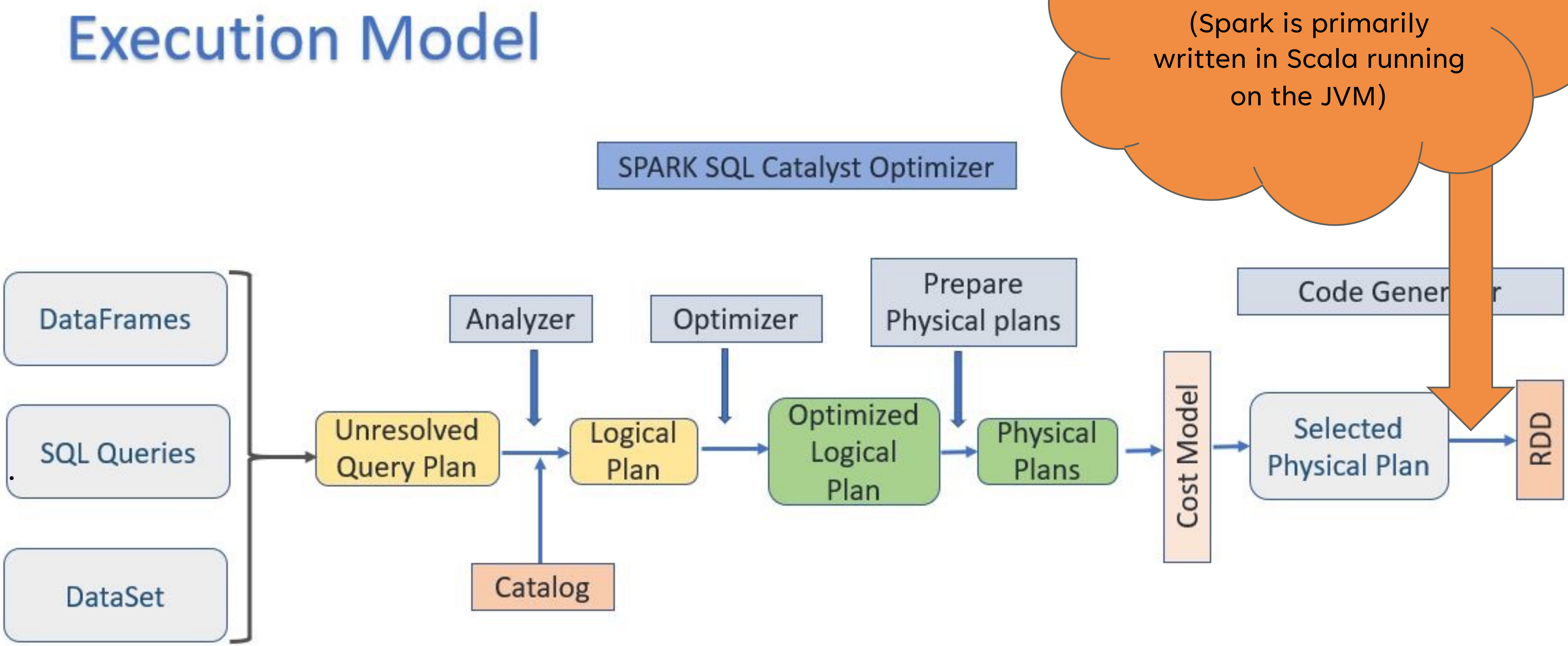
Cost model is used to pick the optimal plan.



CATALYST OPTIMIZER

Code generator outputs optimized Java bytecode

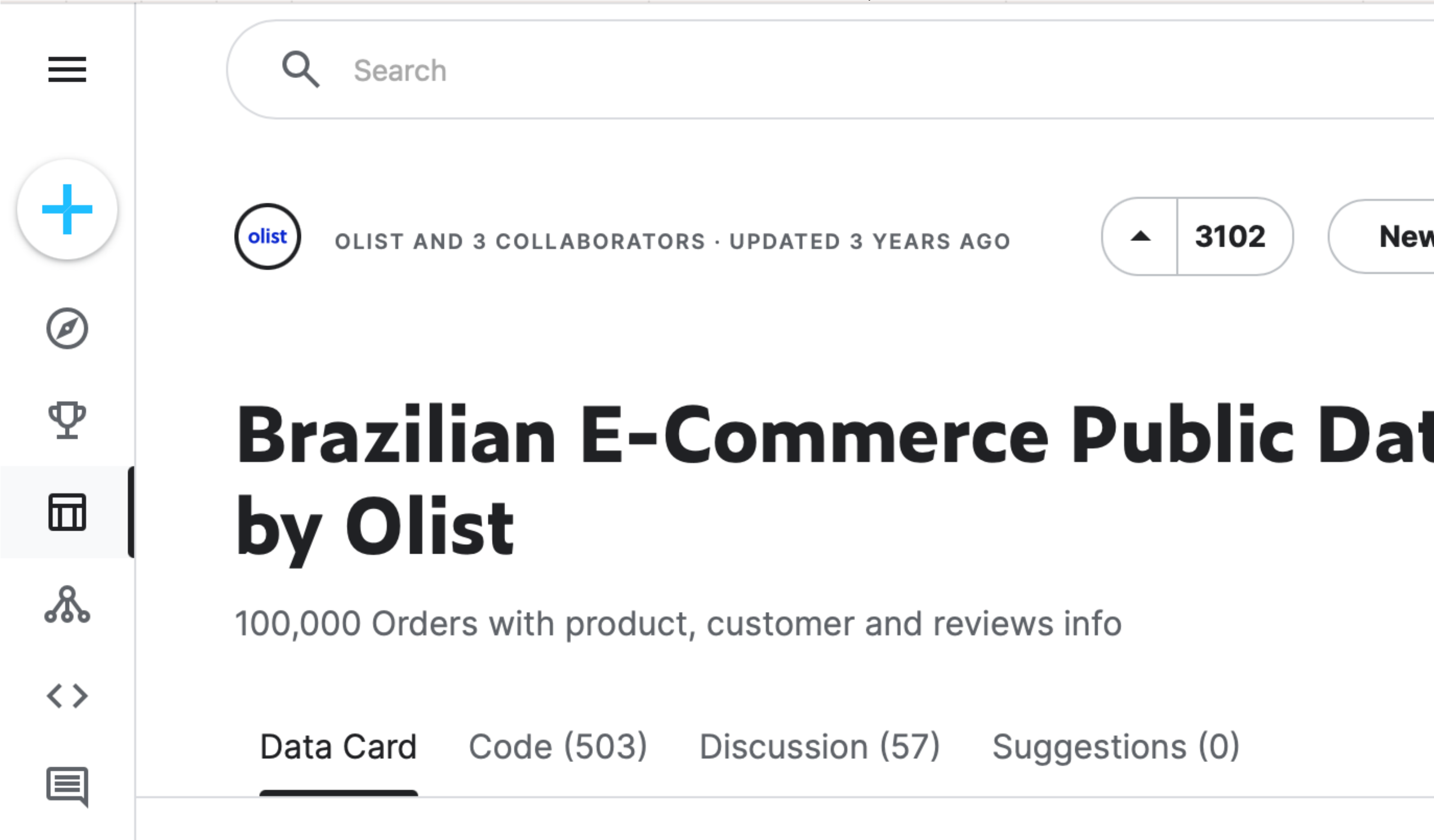
(Spark is primarily written in Scala running on the JVM)



OLIST: BRAZILIAN E-COMMERCE PLATFORM

OLIST

- Dataset for homework 5.



OLIST

- The dataset has been committed to github using git's large-file-storage (git-lfs).
- If you pull the class repository it will download a copy of the dataset to your local system. (see hw5/archive)

  OR

- Or you can download the dataset from Kaggle.

```
dave@FogelmauashsMBP archive % ls -1
olist_customers_dataset.csv
olist_geolocation_dataset.csv
olist_order_items_dataset.csv
olist_order_payments_dataset.csv
olist_order_reviews_dataset.csv
olist_orders_dataset.csv
olist_products_dataset.csv
olist_sellers_dataset.csv
product_category_name_translation.csv
dave@FogelmauashsMBP archive %
```

EXPLORATORY DATA ANALYSIS

- Purpose is to better understand the data.
  - Understand its structure
  - Understand its semantics
  - Understand the relationships between data.

DATA CLEANING

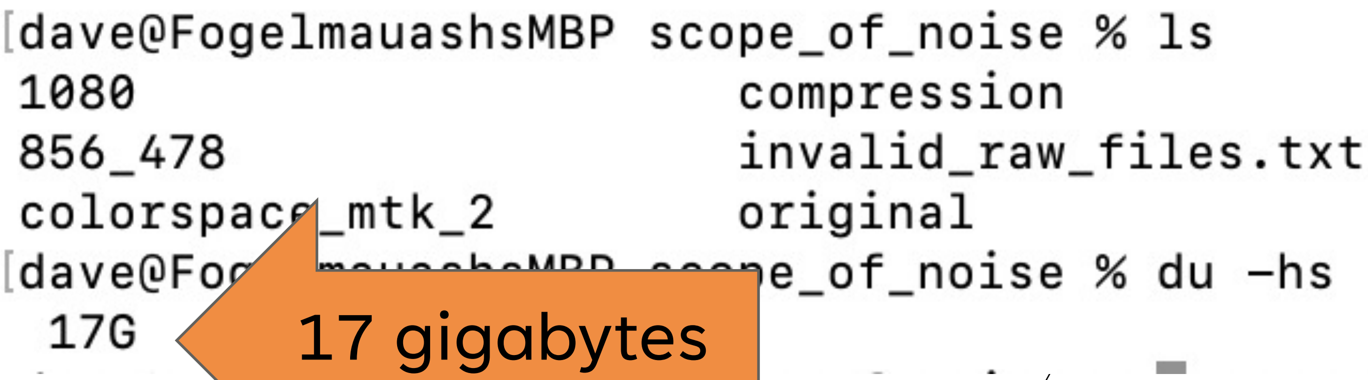
- Usually viewed as part of Exploratory Data Analysis.
- How do we deal with the messiness of real-world data?
- Identifying missing values.
    - Imputation (filling in missing values)
    - Exclusion
- Detecting outliers
- Detecting errors
    - Sometimes errors are obvious and correctable.

EXPLORATORY DATA ANALYSIS: UNDERSTANDING THE DATA

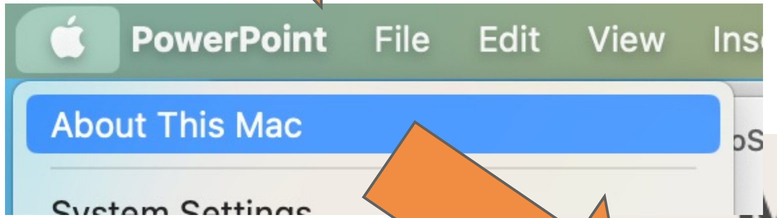
Understand size of data.

Is data bigger than physical memory?

If so, maybe Pandas isn't sufficient. Need cluster.



17 gigabytes



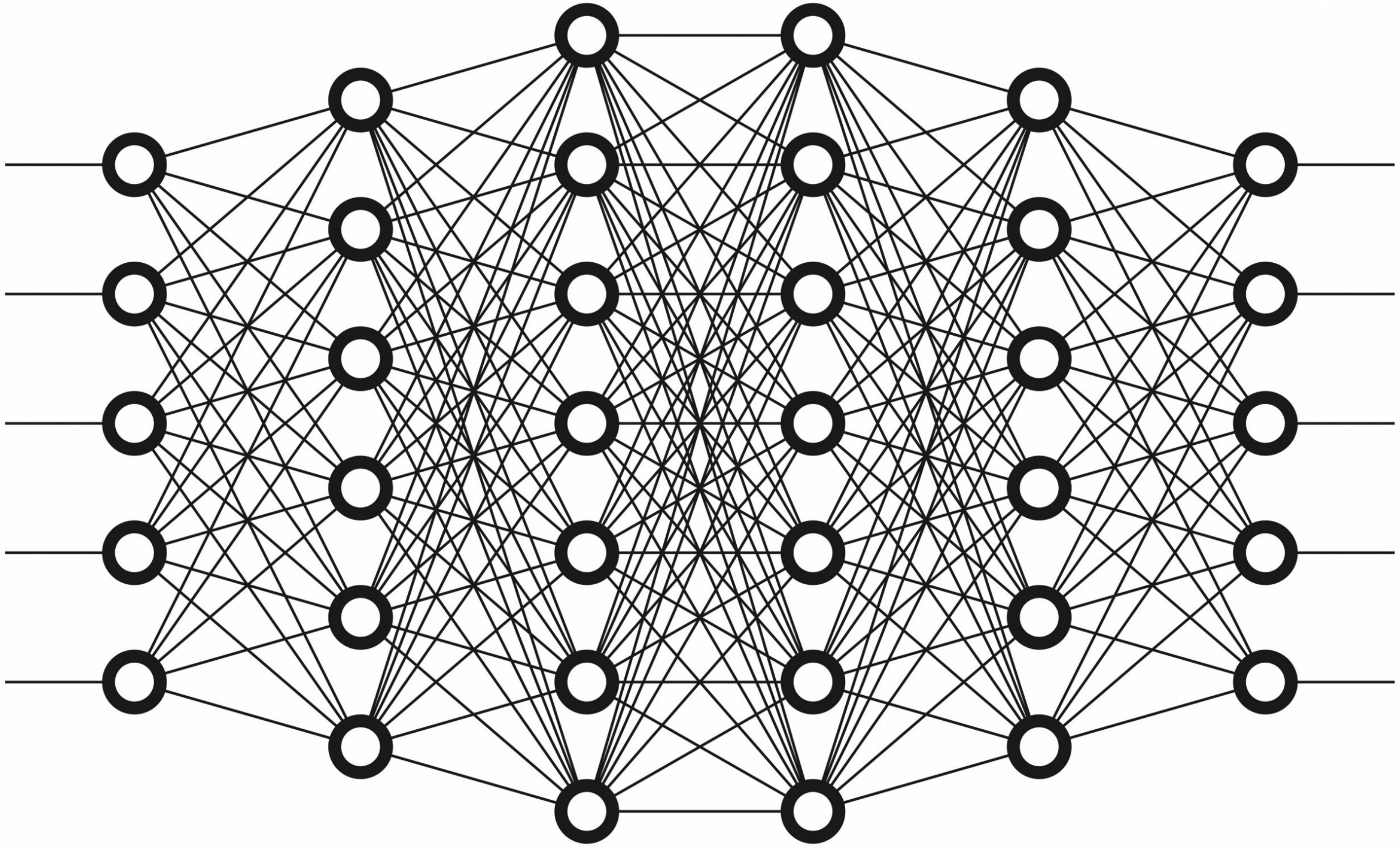
32 gigabytes

EXPLORATORY DATA ANALYSIS: UNDERSTANDING THE DATA

Understand computational requirements.

Do I intend to do machine learning?

Perhaps a machine with many GPUs is appropriate or Google Colab.



EXPLORATORY DATA ANALYSIS: UNDERSTANDING THE DATA

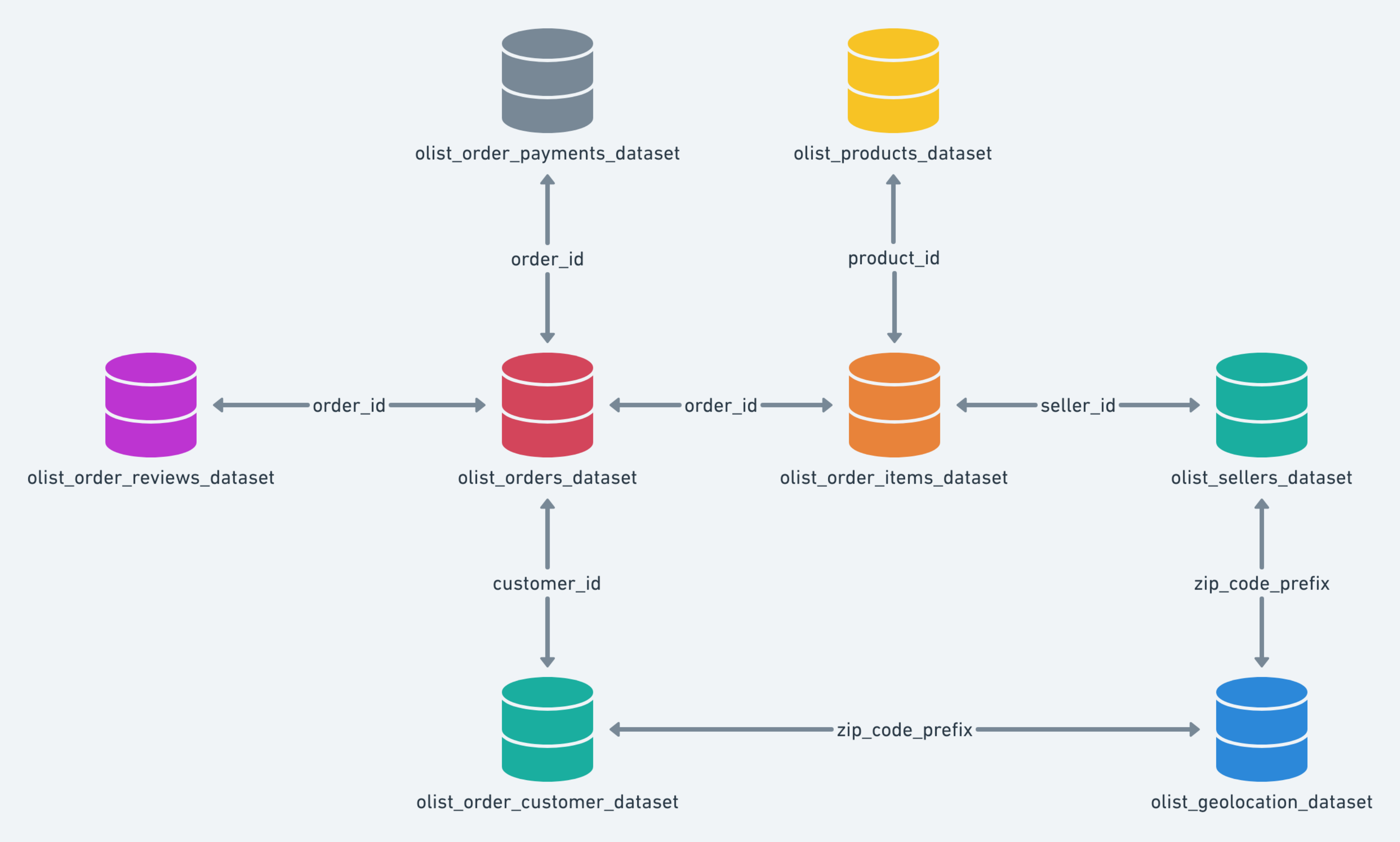
Understand computational requirements.

Am I planning to do many transformations, map-reduce operations or SQL operations.

Maybe Spark.

EXPLORATORY DATA ANALYSIS: UNDERSTANDING THE DATA

Diagram the relationships between datasets.



EXPLORATORY DATA ANALYSIS: UNDERSTANDING THE DATA

Summarize

- types of data
  - Numeric, categorical
  - Ordinal

the meaning (semantics) of each field,

customer_id	customer_unique...	# customer_zip_co...	customer_city
key to the orders dataset. Each order has a unique customer_id.	unique identifier of a customer.	first five digits of customer zip code	customer city name
99441 unique values	**96096** unique values	 1003 100.0k	sao paulo 16% rio de janeiro 7% Other (77019) 77%
06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	14409	franca
18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	09790	sao bernardo do campo
4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	01151	sao paulo

EXPLORATORY DATA ANALYSIS: UNDERSTANDING THE DATA

Look at some of the raw data

[134]:
```
customers_df = pd.read_csv(os.path.join(DS, 'olist_customers_dataset.csv'))
display(customers_df.head(10))
```

	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	1151	sao paulo	SP
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	8775	mogi das cruzes	SP
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	13056	campinas	SP
5	879864dab9bc3047522c92c82e1212b8	4c93744516667ad3b8f1fb645a3116a4	89254	jaragua do sul	SC
6	fd826e7cf63160e536e0908c76c3f441	addec96d2e059c80c30fe6871d30d177	4534	sao paulo	SP
7	5e274e7a0c3809e14aba7ad5aae0d407	57b2a98a409812fe9618067b6b8ebe4f	35182	timoteo	MG
8	5adf08e34b2e993982a47070956c5c65	1175e95fb47ddff9de6b2b06188f7e0d	81560	curitiba	PR
9	4b7139f34592b3a31687243a302fa75b	9afe194fb833f79e300e37e580171f22	30575	belo horizonte	MG

DATA CLEANING:MISSING DATA

Is there missing data?

- If yes, we need to either using imputation to fill-in missing data

OR

- We remove the records missing data.

NO missing data in this case.

```
customers_df.isnull().sum()
```

(3) Spark Jobs

```
Out[10]: customer_id                0
customer_unique_id        0
customer_zip_code_prefix  0
customer_city             0
customer_state            0
dtype: int64
```

DATA CLEANING: MISSING DATA

If we only look for nulls, we may not catch some of the missing data.
Other indicators of missing data:

- Empty strings
- “None”
- “N/A”
- 0
  - 0 can be tricky since it may be valid for some valid for numerical data.
- -1
  - -1 can also be tricky. Maybe used for positive or nonnegative integer numeric data to denote missing.

DATA CLEANING : MISSING DATA

Pandas and Pandas-on-Spark can efficiently check for conditions across entire data sets:

```
import pandas as pd

# Data
data = {
    'Value': [0, 2, 3, 5, 0, 7, 0, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

zeroes = (df.select_dtypes(include=[np.number]) == 0)
display(zeroes)
```

DATA CLEANING: MISSING DATA

Pandas and Pandas-on-Spark can efficiently check for conditions across entire data sets:

```python
import pandas as pd

# Data
data = {
    'Value': [0, 2, 3, 5, 0, 7, 0, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

zeroes = (df.select_dtypes(include=[np.number]) == 0)
display(zeroes)
```

	Value	Name
0	0	Alice
1	2	Bob
2	3	Charlie
3	5	David
4	0	Eve
5	7	Frank
6	0	Grace
7	9	Helen

	Value
0	True
1	False
2	False
3	False
4	True
5	False
6	True
7	False

DATA CLEANING: MISSING DATA

Pandas and Pandas-on-Spark can efficiently check for cond
across entire data sets:

```python
import pandas as pd

# Data
data = {
    'Value': [0, 2, 3, 5, 0, 7, 0, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

zeroes = (df.select_dtypes(include=[np.number]) == 0)
display(zeroes)
```

df

	Value	Name
0	0	Alice
1	2	Bob
2	3	Charlie
3	5	David
4	0	Eve
5	7	Frank
6	0	Grace
7	9	Helen

zeroes

	Value
0	True
1	False
2	False
3	False
4	True
5	False
6	True
7	False

DATA CLEANING: MISSING DATA

Selecting fields with NaN (Not-a-Number).

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

nans = df.select_dtypes(include=[np.number]).isna()
display(nans)
```

DATA CLEANING: MISSING DATA

Selecting fields with NaN (Not-a-Number).

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

nans = df.select_dtypes(include=[np.number]).isna()
display(nans)
```

df

	Value	Name
0	0.0	Alice
1	2.0	Bob
2	NaN	Charlie
3	5.0	David
4	0.0	Eve
5	7.0	Frank
6	NaN	Grace
7	9.0	Helen

nans

	Value
0	False
1	False
2	True
3	False
4	False
5	False
6	True
7	False

DATA CLEANING: MISSING DATA

Combine selections of zeroes and NaNs.

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)
```

Logical OR

DATA CLEANING: MISSING DATA

Combine selections of zeroes and NaNs.

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)
```

df

	Value	Name
0	0.0	Alice
1	2.0	Bob
2	NaN	Charlie
3	5.0	David
4	0.0	Eve
5	7.0	Frank
6	NaN	Grace
7	9.0	Helen

num conds

	Value
0	True
1	False
2	True
3	False
4	True
5	False
6	True
7	False

DATA CLEANING: MISSING DATA

Let's assume we choose to discard the rows with missing data.

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)

# axis=1 means create column that is true if any of the
# values in a row are true.
rows_to_drop = numconds.any(axis=1)
display(rows_to_drop)

filtered_df = df[~rows_to_drop]
display(filtered_df)
```

DATA CLEANING: MISSING DATA

Let's assume we choose to discard the rows with missing data.

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)

# axis=1 means create column that is true if any of the
# values in a row are true.
rows_to_drop = numconds.any(axis=1)
display(rows_to_drop)

filtered_df = df[~rows_to_drop]
display(filtered_df)
```

df

	Value	Name
0	0.0	Alice
1	2.0	Bob
2	NaN	Charlie
3	5.0	David
4	0.0	Eve
5	7.0	Frank
6	NaN	Grace
7	9.0	Helen

DATA CLEANING: MISSING DATA

Let's assume we choose to discard the rows with missing data.

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)

# axis=1 means create column that is true if any of the
# values in a row are true.
rows_to_drop = numconds.any(axis=1)
display(rows_to_drop)

filtered_df = df[~rows_to_drop]
display(filtered_df)
```

df

	Value	Name
0	0.0	Alice
1	2.0	Bob
2	NaN	Charlie
3	5.0	David
4	0.0	Eve
5	7.0	Frank
6	NaN	Grace
7	9.0	Helen

num conds

	Value
0	True
1	False
2	True
3	False
4	True
5	False
6	True
7	False

DATA CLEANING: MISSING DATA

Let’s assume we choose to discard the rows with missing data.

```
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)

# axis=1 means create column that is true if any of the
# values in a row are true.
rows_to_drop = numconds.any(axis=1)
display(rows_to_drop)

filtered_df = df[~rows_to_drop]
display(filtered_df)
```

df

	Value	Name
0	0.0	Alice
1	2.0	Bob
2	NaN	Charlie
3	5.0	David
4	0.0	Eve
5	7.0	Frank
6	NaN	Grace
7	9.0	Helen

num conds

	Value
0	True
1	False
2	True
3	False
4	True
5	False
6	True
7	False

rows to drop

```
0     True
1    False
2     True
3    False
4     True
5    False
6     True
7    False
dtype: bool
```

rows_to_drop is now a Series.

DATA CLEANING: MISSING DATA

Let's assume we choose to discard the rows with missing data.

```python
# Data with NaN values
data = {
    'Value': [0, 2, np.nan, 5, 0, 7, np.nan, 9],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
             'Frank', 'Grace', 'Helen']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display DataFrame
display(df)

numconds = ((df.select_dtypes(include=[np.number]) == 0) |
    df.select_dtypes(include=[np.number]).isna())

display(numconds)

# axis=1 means create column that is true if any of the
# values in a row are true.
rows_to_drop = numconds.any(axis=1)
display(rows_to_drop)

filtered_df = df[~rows_to_drop]
display(filtered_df)
```

df

	Value	Name
0	0.0	Alice
1	2.0	Bob
2	NaN	Charlie
3	5.0	David
4	0.0	Eve
5	7.0	Frank
6	NaN	Grace
7	9.0	Helen

rows to drop

```
0     True
1    False
2     True
3    False
4     True
5    False
6     True
7    False
dtype: bool
```

filtered _df

	Value	Name
1	2.0	Bob
3	5.0	David
5	7.0	Frank
7	9.0	Helen

DATA CLEANING: CATEGORICAL DATA

Only showed cleaning based on numeric types.

Should also check for empty strings, NaN.

Should also check for non-sensical values.

For categorical data, make sure all values represent defined categories.

DATA CLEANING: IMPUTATION

What if we don’t have enough data to drop rows with one or two missing fields?

What if we think removing the rows with missing data will introduce bias?

- Ex: social desirability bias may cause someone to refuse to answer a question on a survey.

Answer: IMPUTE

Imputation = substituting values for missing data.

- For numeric data the values are often based on the other values in a column

OR

- based on rows that have other correlating features like demographics.

DATA CLEANING: NUMERICAL IMPUTATION

Mean imputation: substitute mean of the feature column

Median imputation: substitute median of the feature column.

Mode imputation: substitute mode of the feature column

Linear interpolation: where there are surrounding numerical data, e.g., on a surface or in a time series, you may linear interpolate values for missing data points.

Polynomial/Spline interpolation: enables better estimates for non-linear surfaces or curves.

Linear/Polynomial regression: fit a curve that minimizes an error function (e.g., sum squared error).

DATA CLEANING: NUMERICAL IMPUTATION

Clustering: assign to the nearest cluster.

Random Forest Regression: use machine learning to infer the missing values.

Deep Learning: Generative algorithms. Go wild.

DATA CLEANING: CATEGORICAL IMPUTATION

Mode imputation: substitute the most common category.

Logistic Regression Classifier: useful for binary classification. Use other fields to infer the missing field.

Random Forest Classifier: another ML tool for inferring the value based on existing values.

Deep Learning (again).

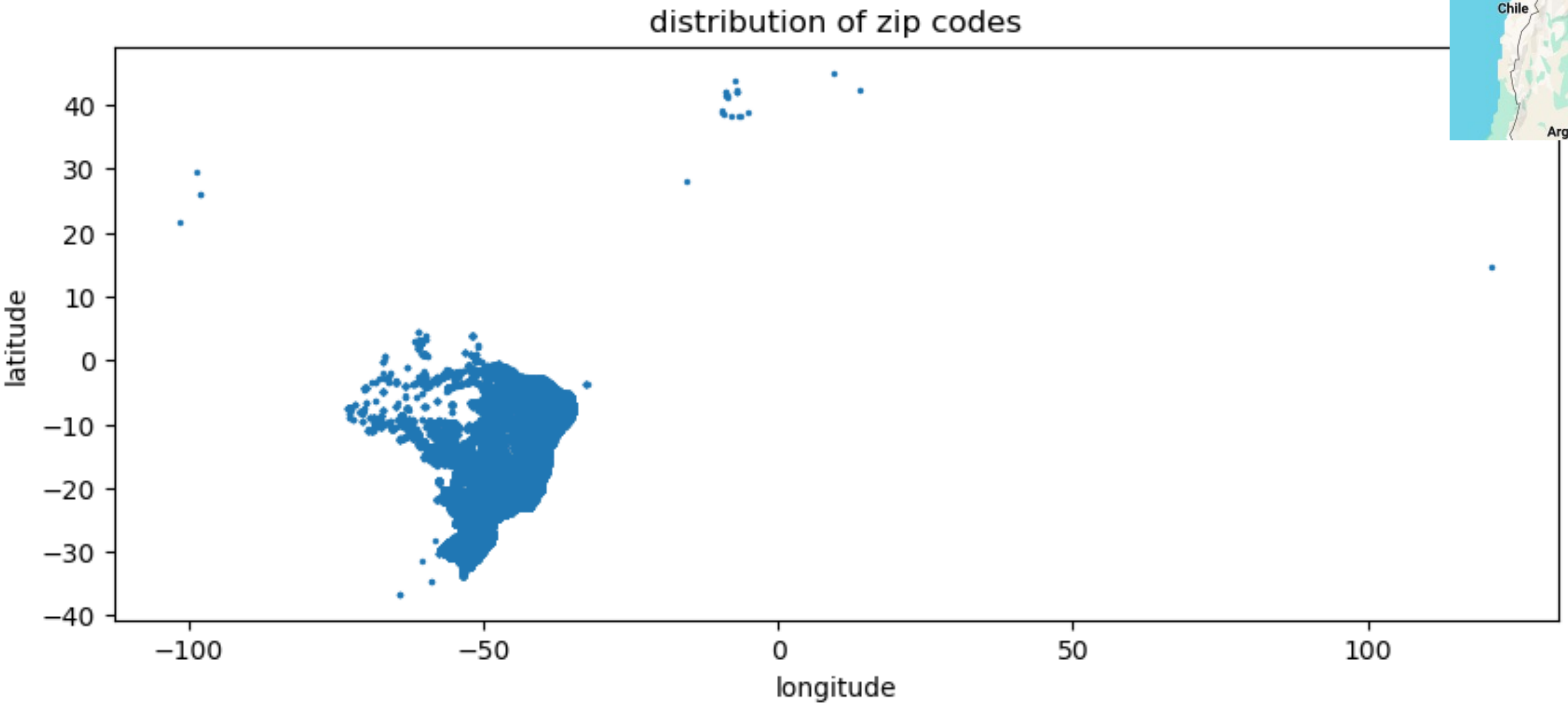
DATA CLEANING:GEOLOCATION

Do geolocations make sense?

Let's consider a case of fixing erroneous rather than missing data.

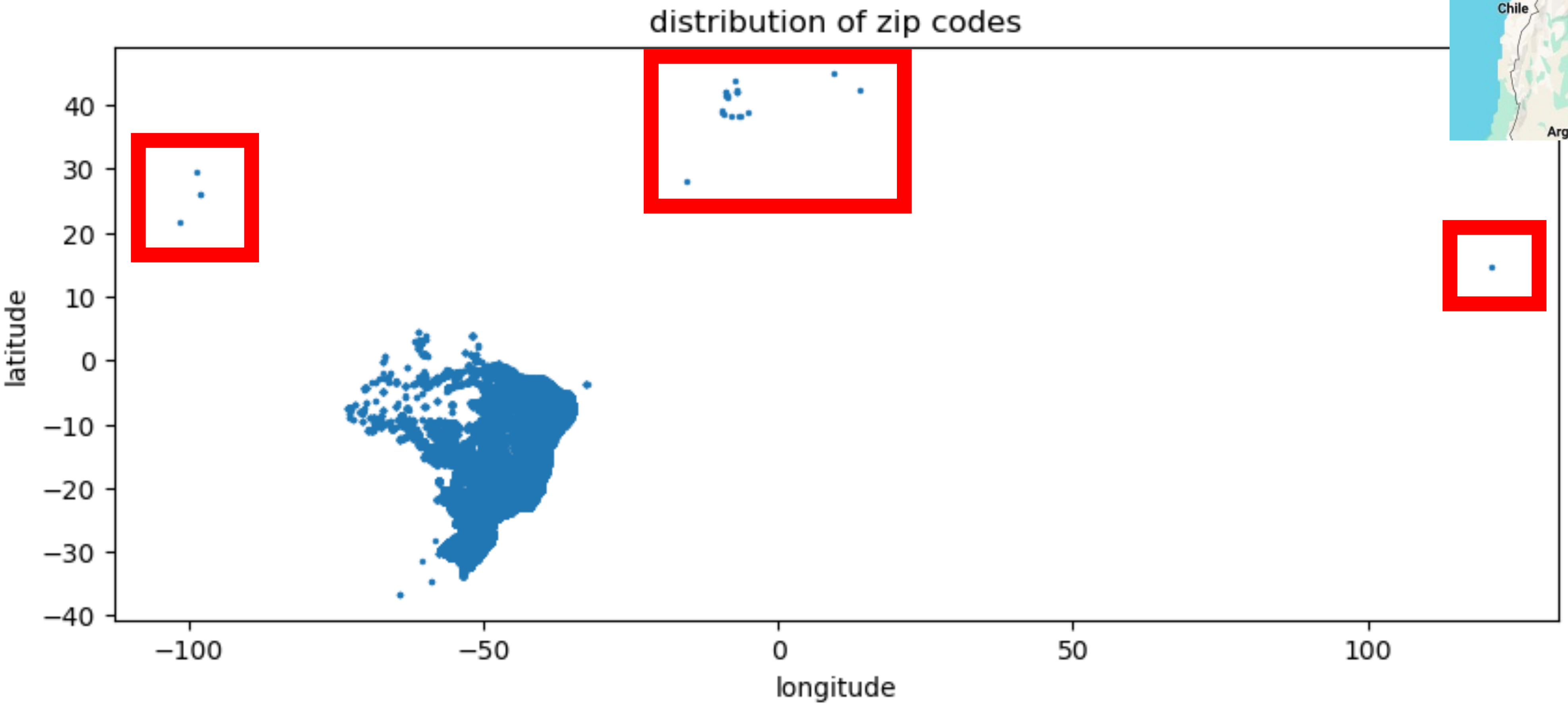
DATA CLEANING:GEOLOCATION

•



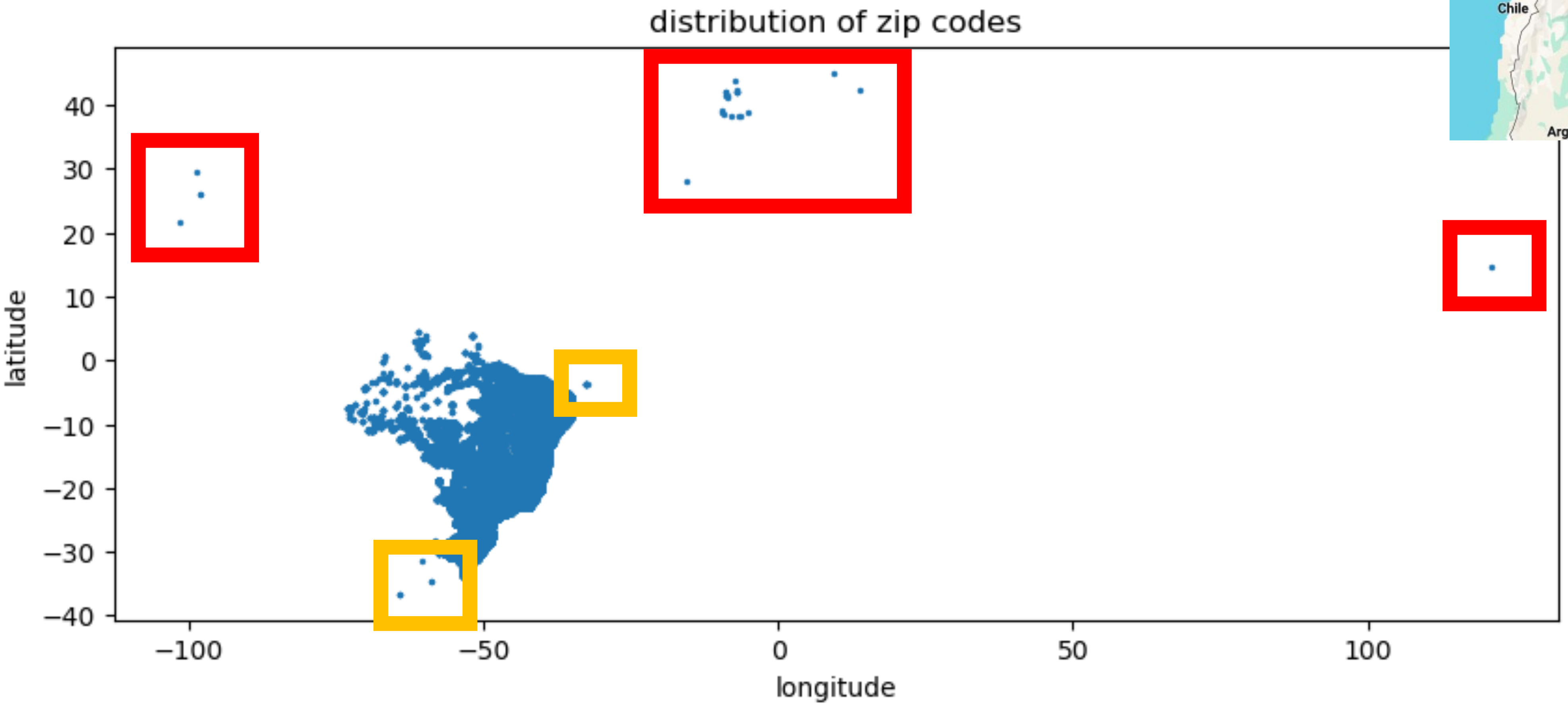
DATA CLEANING:GEOLOCATION

•



DATA CLEANING:GEOLOCATION

•



DATA CLEANING:GEOLOCATION

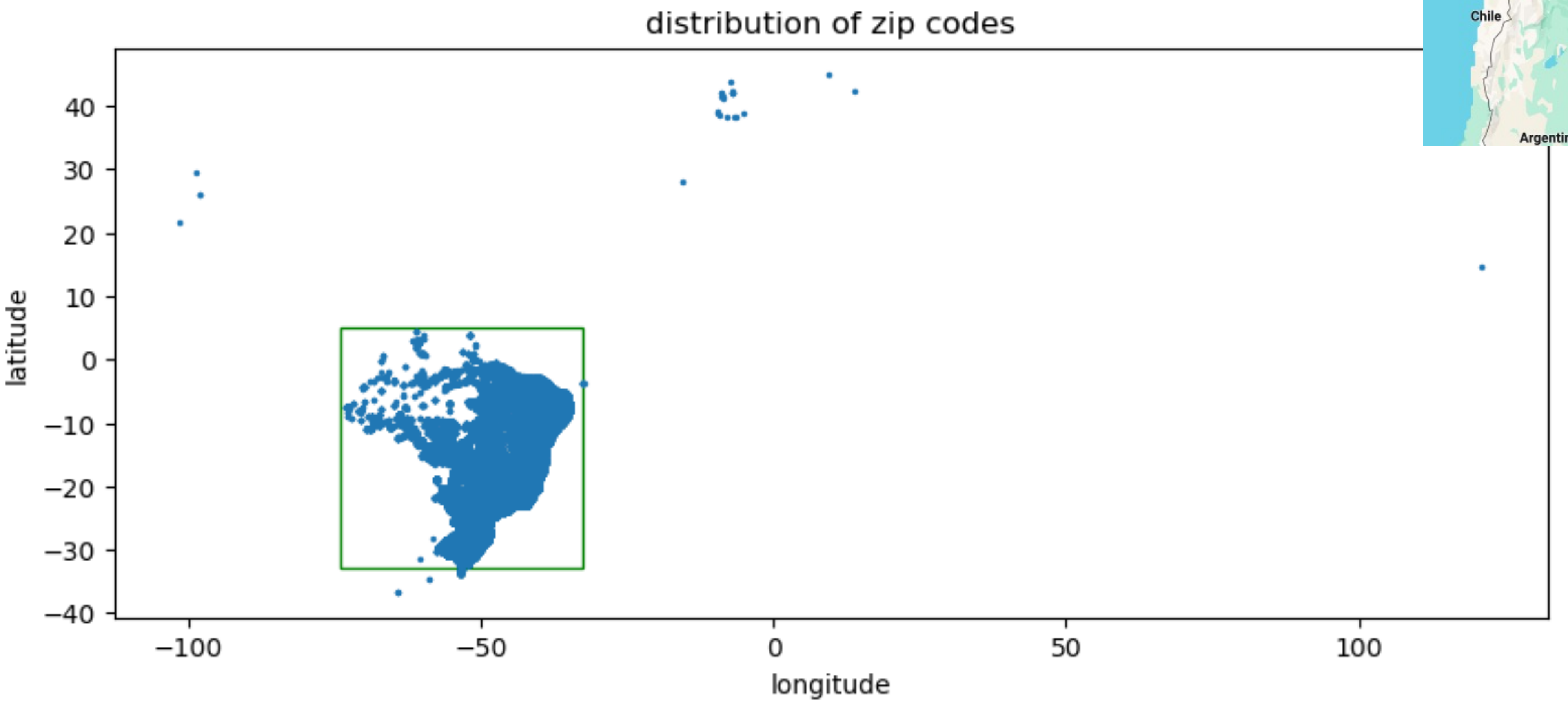

```
indices_outside = geo_df[geo_df['geolocation_lng'] < w_long].index
indices_outside = indices_outside.append(o > e_long].index)
indices_outside = indices_outside.append(geo_df[geo_df['geolocation_lat'] > n_lat].index)
indices_outside = indices_outside.append(geo_df[geo_df['geolocation_lat'] < s_lat].index)

indices_inside = geo_df.index.difference(indices_outside)
filtered_df = geo_df.drop(indices_inside)
display(filtered_df)
```

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state
387565	18243	28.008978	-15.536867	bom retiro da esperanca	SP
513631	28165	41.614052	-8.411675	vila nova de campos	RJ
513643	28155	-34.586422	-58.732101	santa maria	RJ
513754	28155	42.439286	13.820214	santa maria	RJ
514429	28333	38.381672	-6.328200	raposo	RJ
516682	28595	43.684961	-7.411080	portela	RJ
538512	29654	29.409252	-98.484121	santo antônio do canaã	ES
538557	29654	21.657547	-101.466766	santo antonio do canaa	ES
585242	35179	25.995203	-98.078544	santana do paraíso	MG
585260	35179	25.995245	-98.078533	santana do paraiso	MG

Same city in different hemispheres?

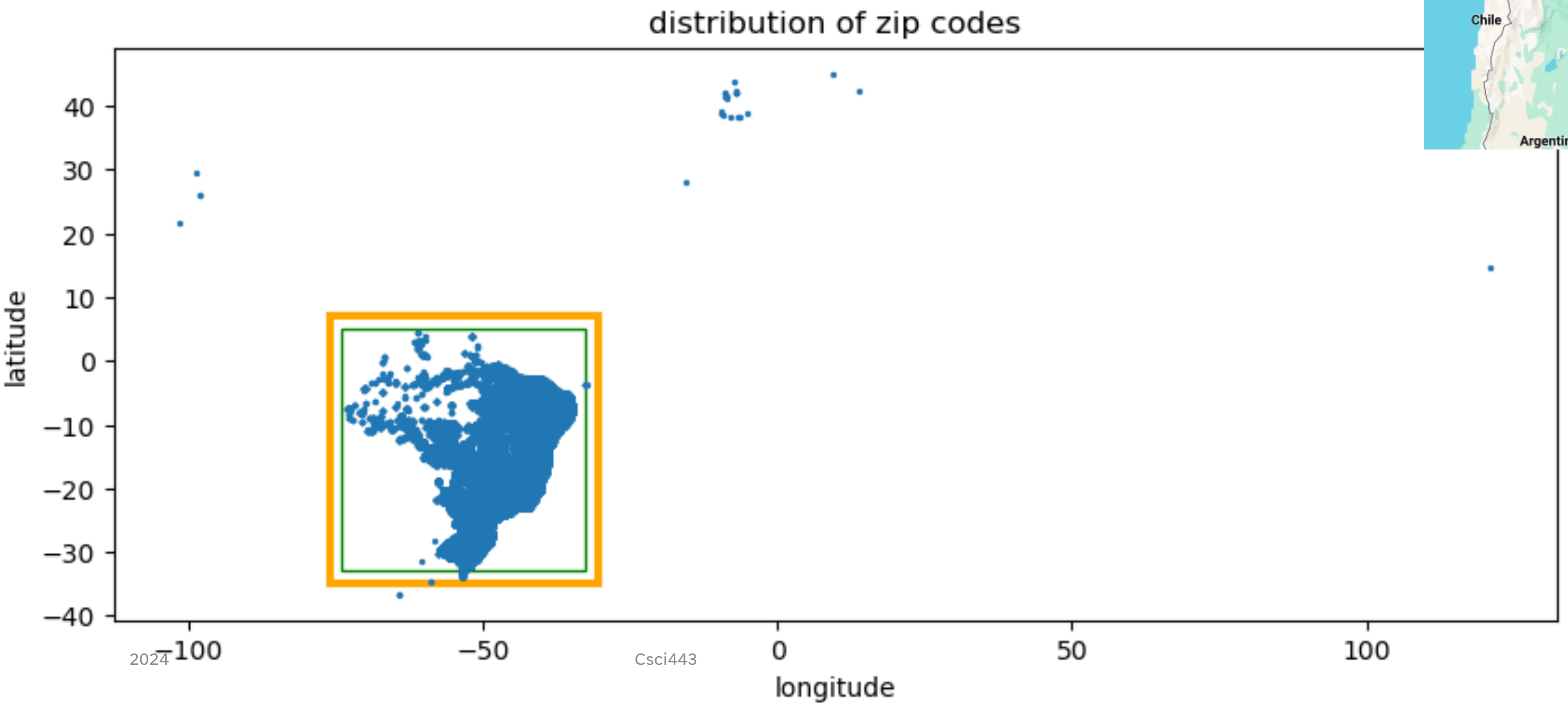
DATA CLEANING:GEOLOCATION

Add bounding box based on Google Maps (lat, long).

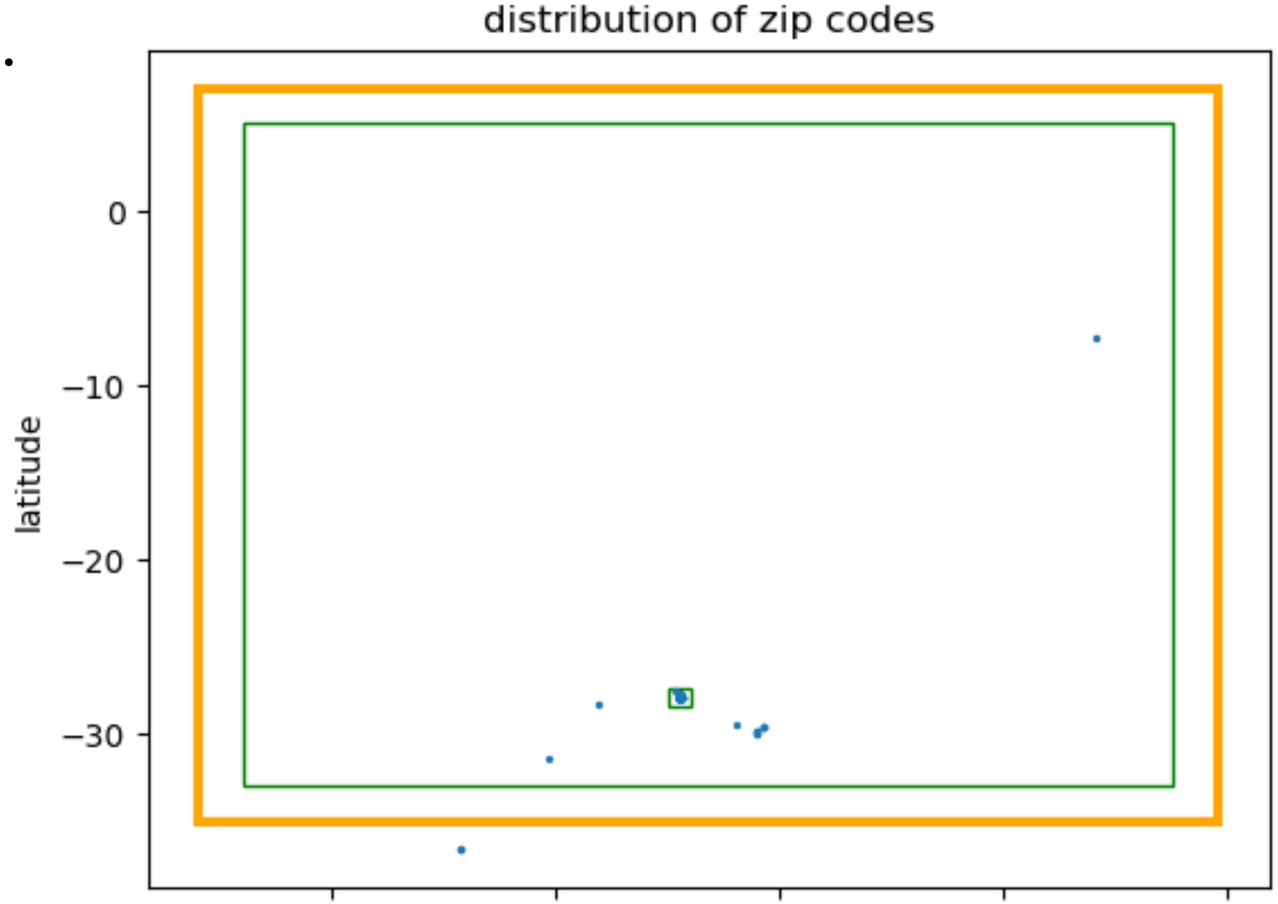


DATA CLEANING:GEOLOCATION

Add margin of 2 degrees in each direction.

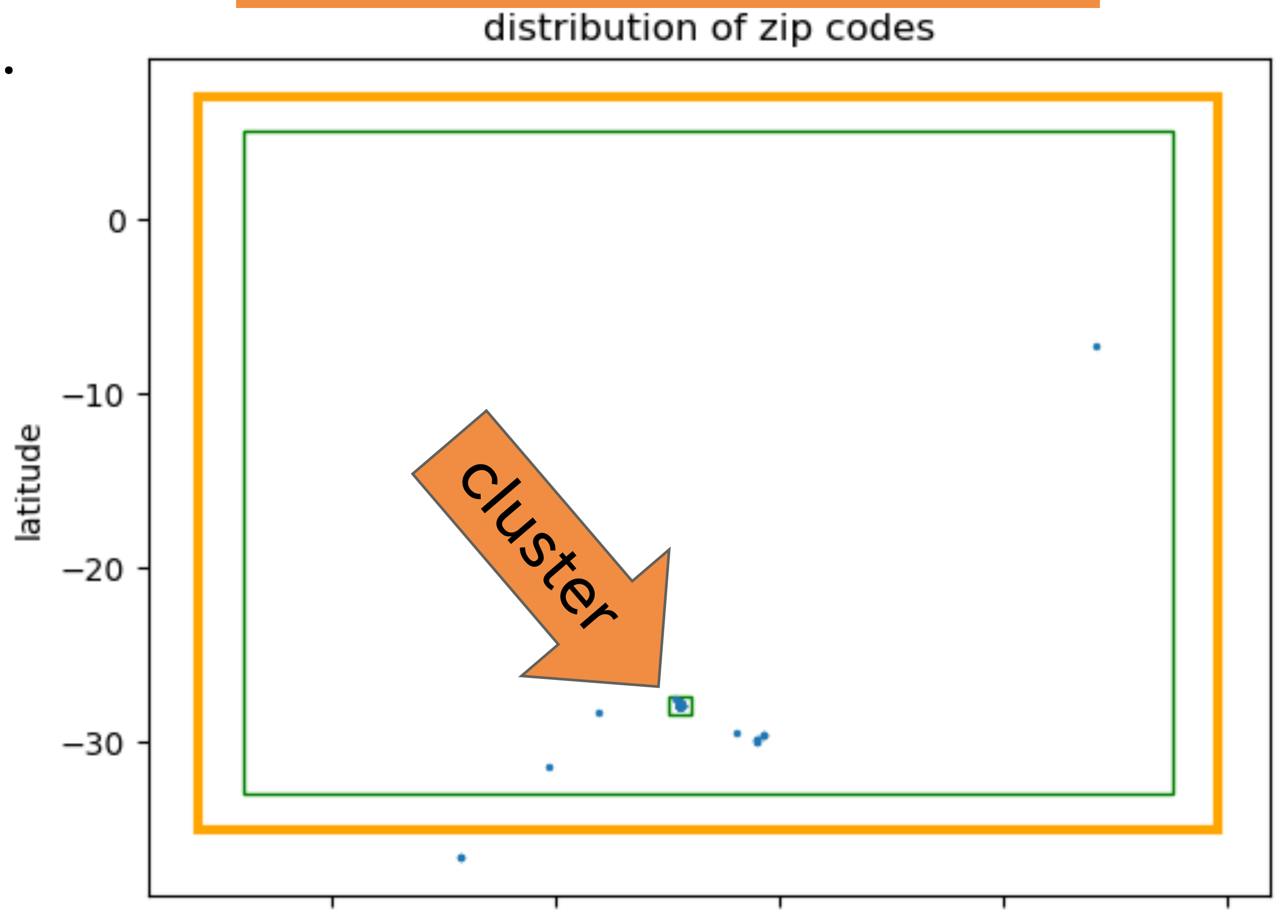


DATA CLEANING:GEOLOCATION



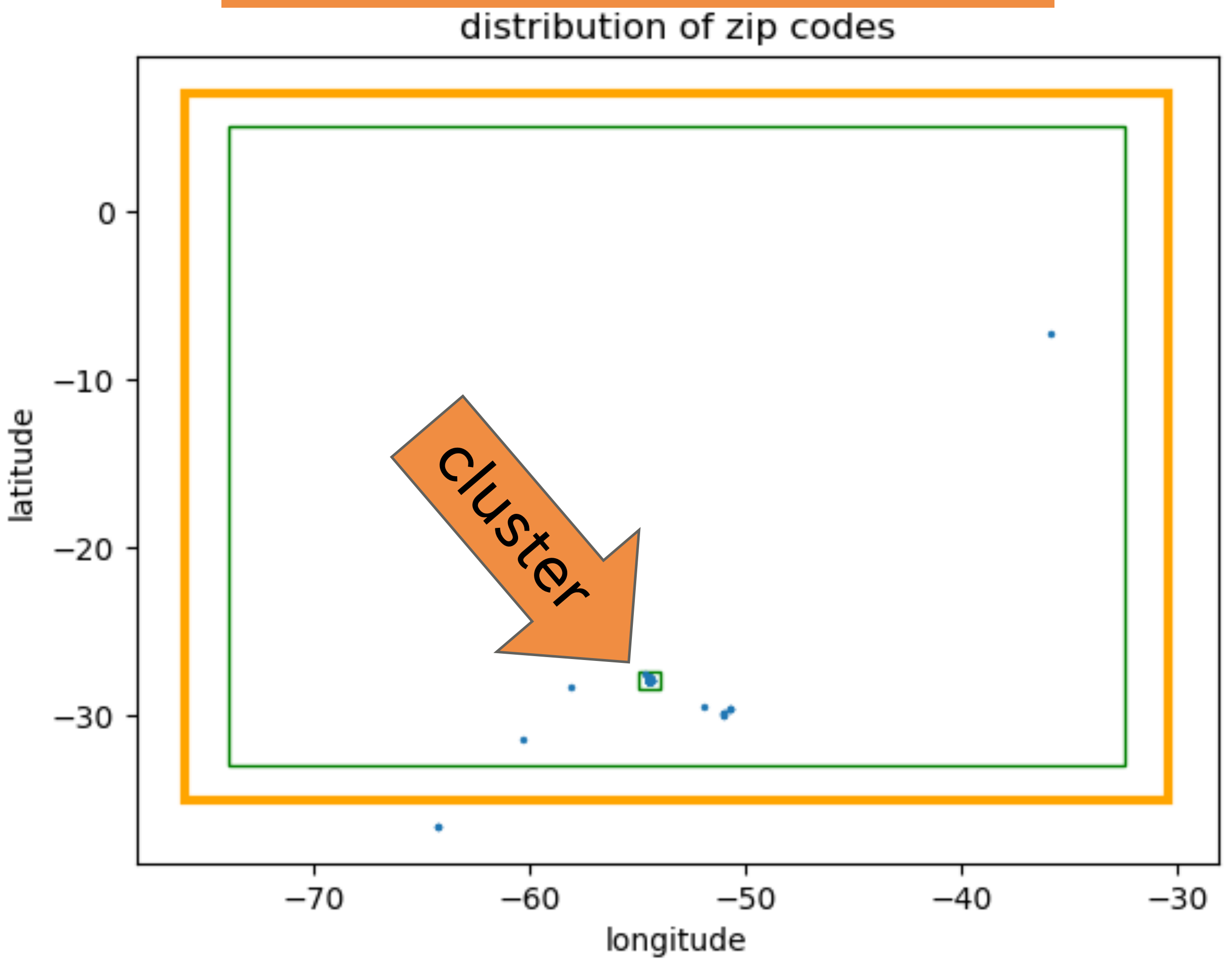

```
correct_santa_rosa = Rectangle((-54.457-0.5, -27.875-0.5), 1, 1, linewidth=1, edgecolor='g', facecolor='none')
plt.gca().add_patch(correct_santa_rosa)

santa_rosa = geo_df[(geo_df["geolocation_city"] == "santa rosa") & (geo_df["geolocation_state"] == "RS")]
plt.scatter(santa_rosa["geolocation_lng"], santa_rosa["geolocation_lat"], s=2)
plt.show()
```

DATA CLEANING:GEOLOCATION




```
correct_santa_rosa = Rectangle((-54.457-0.5, -27.875-0.5), 1, 1, linewidth=1, edgecolor='g', facecolor='none')
plt.gca().add_patch(correct_santa_rosa)

santa_rosa = geo_df[(geo_df["geolocation_city"] == "santa rosa") & (geo_df["geolocation_state"] == "RS")]
plt.scatter(santa_rosa["geolocation_lng"], santa_rosa["geolocation_lat"], s=2)
plt.show()
```

DATA CLEANING:GEOLOCATION



Use clustering and take the biggest cluster?

Use cluster centroid for zip codes > certain distance from the cluster centroid?

THANK YOU

David Harrison

Harrison@cs.olemiss.edu