YOW DATA 2019

AUTO FEATURE ENGINEERING

RAPID FEATURE HARVESTING USING DFS AND DATA ENGINEERING TECHNIQUES

Ananth Gundabattula

Senior Architect @CBA

FEATURE ENGINEERING – TWO PARADIGMS

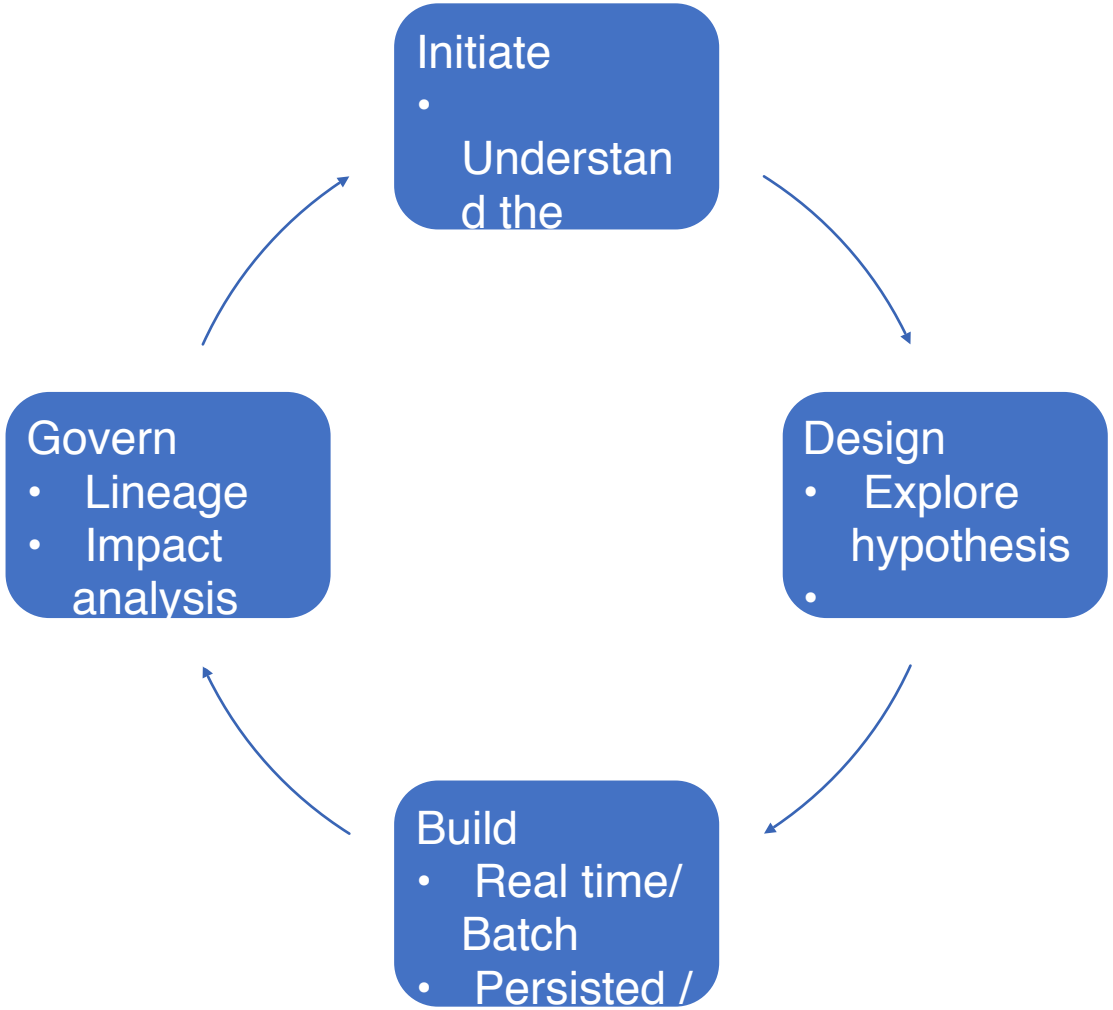
Classical machine learning

- Feature engineering : The art of creating features from raw data using domain knowledge
- Explainable
- Relational and human behavioral use cases. (Ex: Reporting)
- Expensive - Easily one of the top two cost contributors in building a model as it is human intuition driven.

Deep learning

- Feature engineering is intrinsic to the model.
- Not explainable (yet!)
- Problem/cost shifts towards building the right parameters of the neural net model

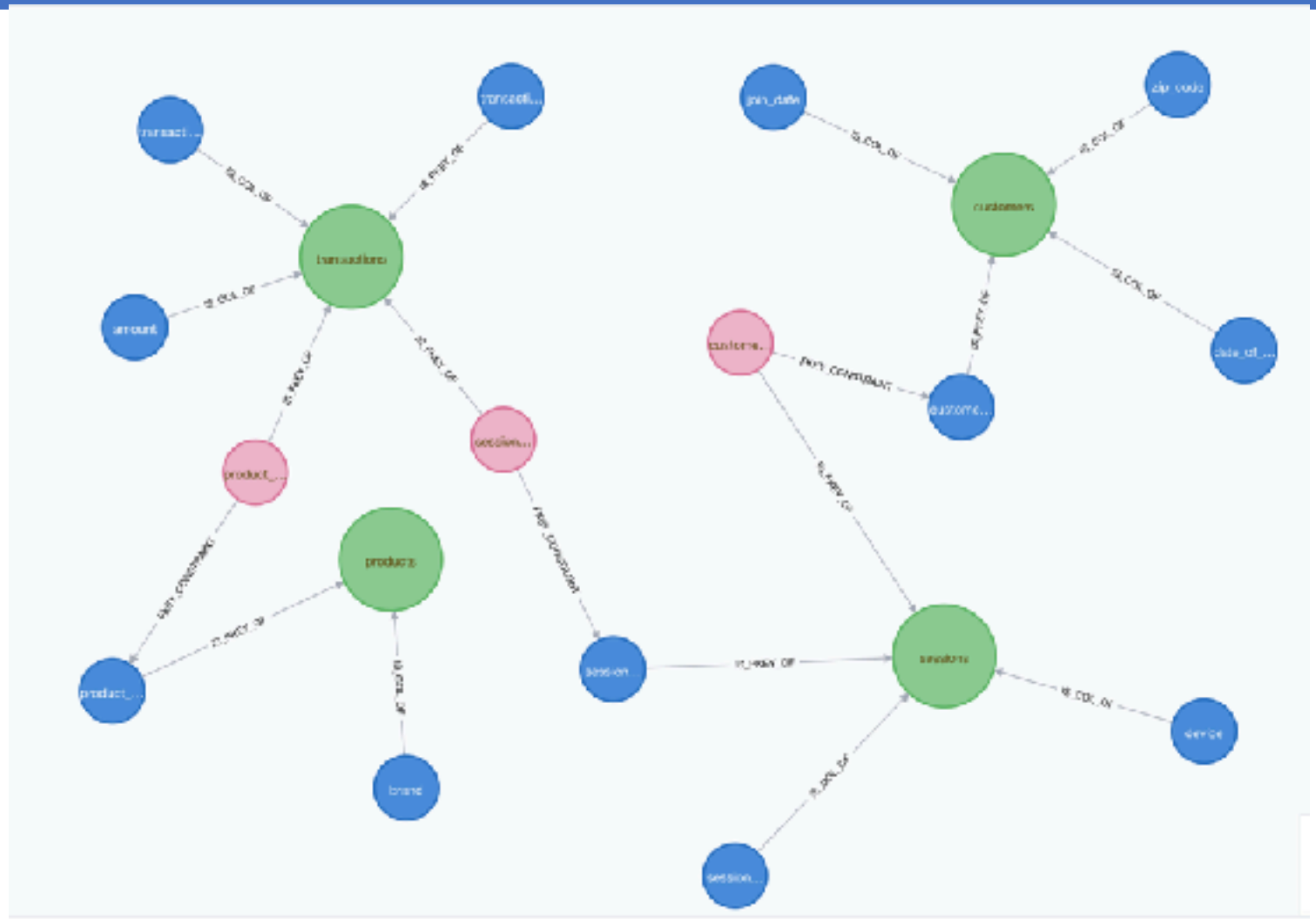
LIFETIME OF A FEATURE



CAN WE AUTOMATE PARTS OF THIS LIFECYCLE ?

- Reduce cost
- Possibly govern better
- Risk assess better
- Maintain it better

TOY EXAMPLE SCHEMA



TOY EXAMPLE DEMO – DFS ALGORITHM

GENERATE FEATURES FOR THE CUSTOMER ENTITY

SAMPLE FEATURE ANALYSIS

Customer- MAX(sessions.MEAN(transactions.amount))

The maximum of the average spend across each session of a customer

Customer - MIN(sessions.NUM_UNIQUE(transactions.products.MODE(transactions.session_id)))

The minimum of the number of unique products that were purchased in sessions with most frequent number of transactions.

Transactions - sessions.customers.NUM_UNIQUE(sessions.WEEKDAY(session_start))

The number of unique weekdays that a customer typically shops in – Is he/she a weekend customer ?

Transactions - sessions.customers.SUM(sessions.MIN(transactions.amount))

Sum of all minimum transaction amount per session until now

DFS – DEEP FEATURE SYNTHESIS ALGORITHM

DFS ALGORITHM – BUILDING BLOCKS

Direct Feature (JOIN and INHERIT)

product_id	product_price

JOIN

order_id	product_id	order_date

As it is a forward relationship

order_id	product_id	order_date	product_price

Aggregation Feature (JOIN + GROUP BY)

order_id	cust_id	order_date	product_price

As it is a backward relationship

GROUP BY (Product price) + **Aggregation Primitive (ex: SUM)**

order_id	cust_id	order_date	SUM(product_price)

DFS ALGORITHM – BUILDING BLOCKS CONTD….

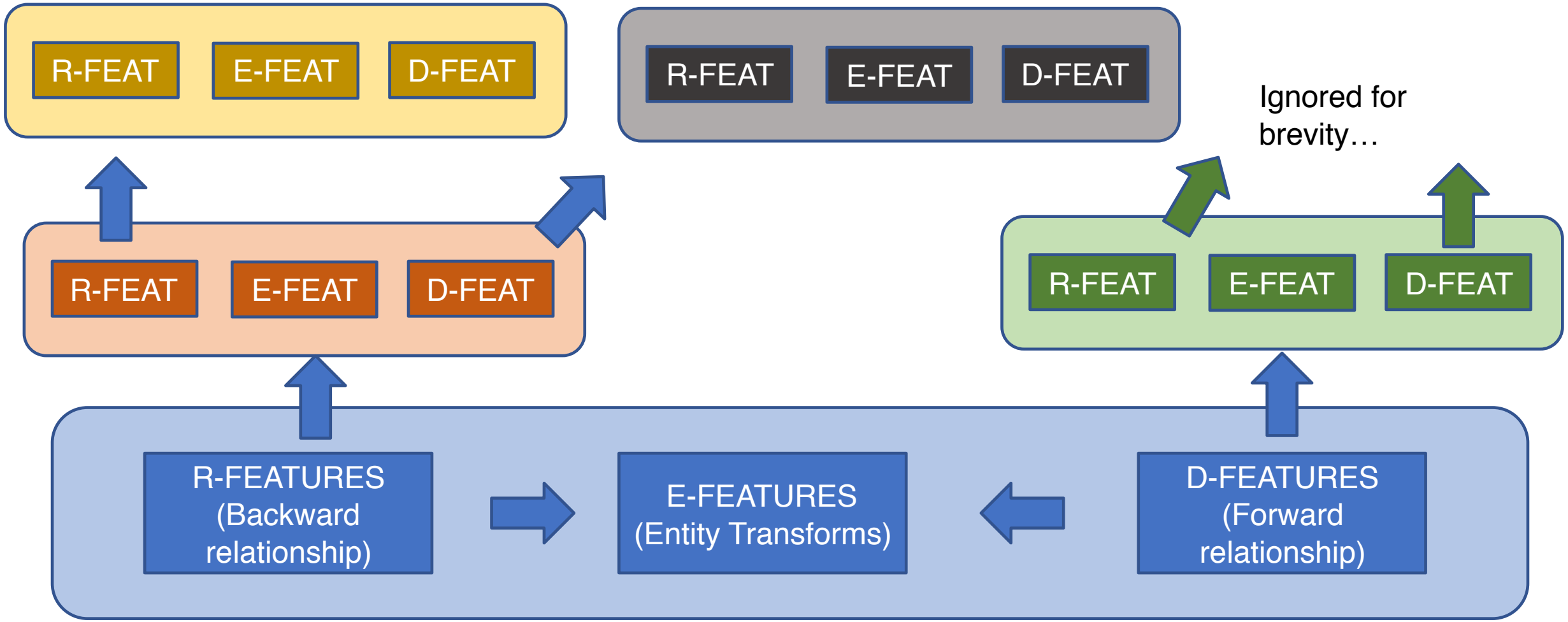
Transform Feature (Primitive on same row)

order_id	order_dt	product_id

MONTH function on datetime type

order_id	order_dt	month(order_date)	product_id

DFS ALGORITHM – STACK TO HARVEST



DESIGN PHASE CONSTRUCTS

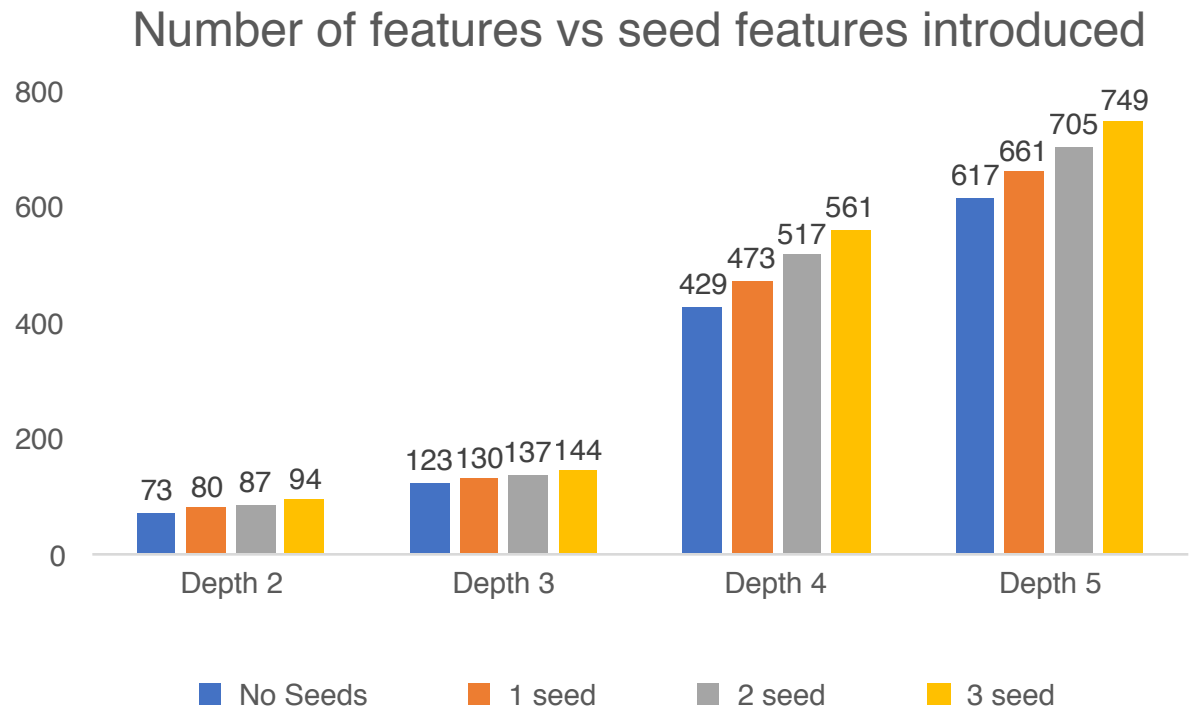
DOMAIN - UNDERSTANDING OF THE BASE DATA TYPES

Not a permutation and combination library. Data types taken into account when fed as metadata

- Datetime – Apply on time functions.
- Categorical types – ex: NUM_UNIQUE on categorical but not on amounts
- Time Deltas
- IDs,
- Lat/Long
- ZIPCode, CountryCode
- IPAddress, Phone Number
- Email Addresses
- FilePath

DOMAIN FEATURES – SEEDING BASE FEATURES

SKEW(sessions.**PERCENT_TRUE**(
transactions.amount > 125))



Expensive purchase – Amount > 125
Inexpensive purchase – Amount < 25
Rounded 10 purchase – Amount %10 == 0

transaction_id	transaction_amount	expensive_purchase	inexpensive_purchase	multiple_of_10s

DOMAIN – INTERESTING VALUES/ FOCUS ON SUBSET OF ROWS

session_id	customer_id	device
1	1	desktop
2	3	iphone
3	1	iphone
4	2	galaxy
5	4	desktop
6	4	desktop

COUNT(sessions WHERE device = desktop)

COUNT(sessions WHERE device = iphone)

BRING YOUR OWN DOMAIN FUNCTIONS

Features based on SQL function primitives

case_id	customer_id	case type

SQL Aggregation catalogue

SQL Transform catalogue

case_id	comments

Features based on NLP function primitives

case_id	customer_id	case type

NLTK Aggregation catalogue

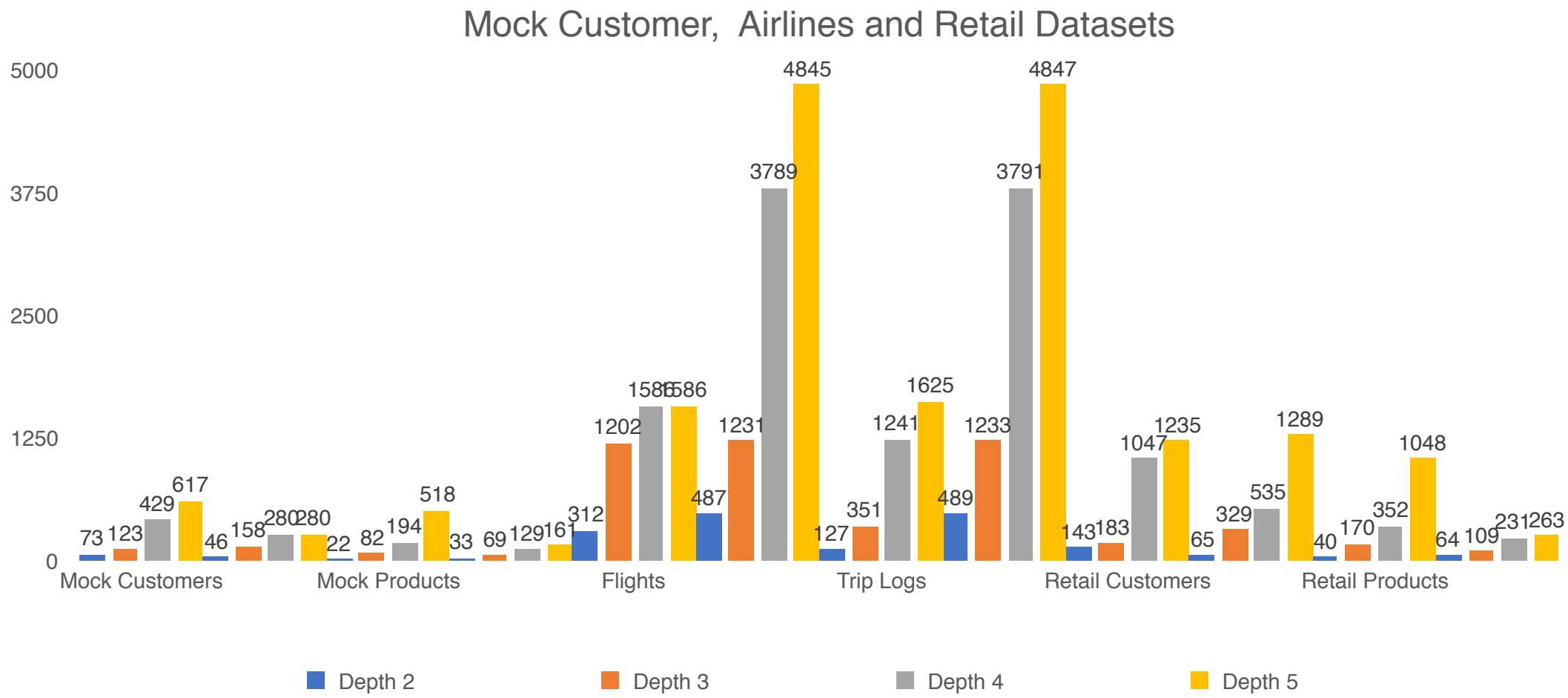
NLTK Transform catalogue

case_id	comments

FEATURE EXPLOSION

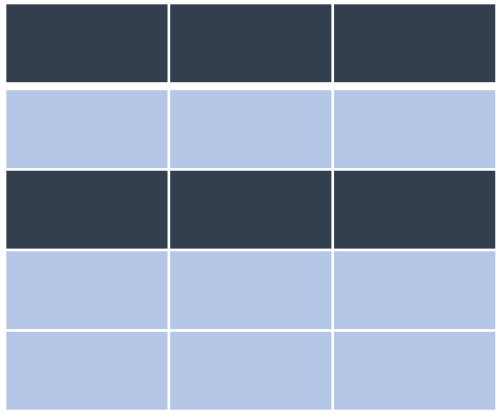
Number of features is a function of:

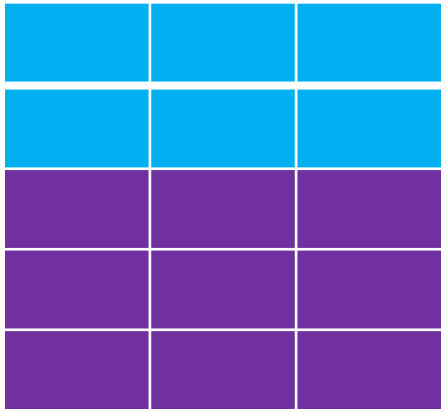
- Entities
- Rfeat
- Dfeat
- Depth
- EFeat

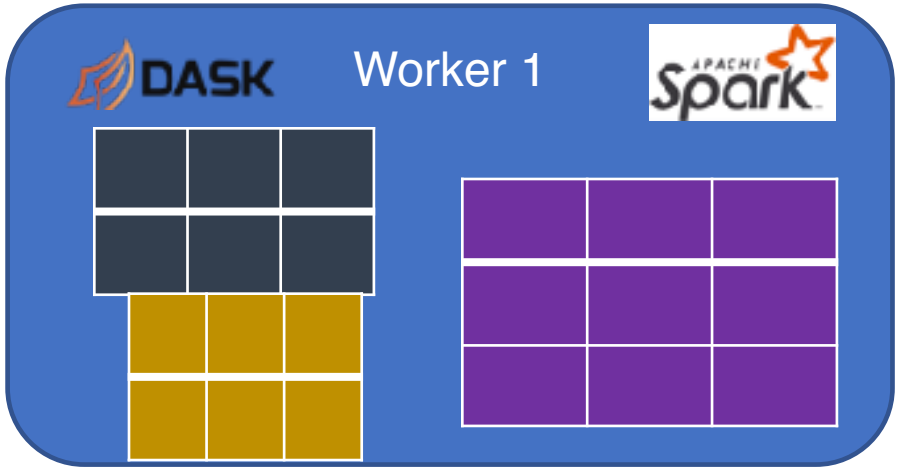


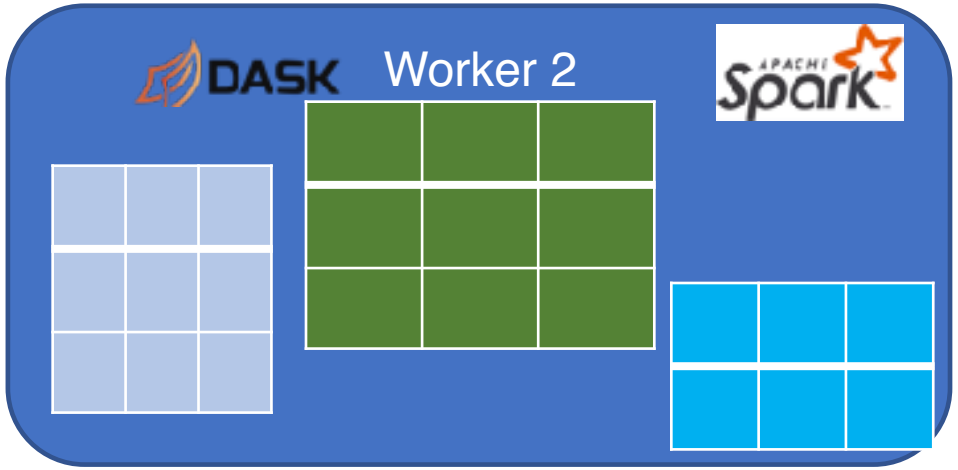
DISTRIBUTED FEATURE GENERATION

Ensure each distributed partition contains all the forward and backward related rows of the primary entity in the same partition









FEATURE SELECTION – A COMPUTE INTENSIVE CHALLENGE

- Cost model **shifts** from human cost to compute cost
- **Compute costs much cheaper** than human cost and hence the overall reduced cost

Compute is spent in materializing large number of feature columns so that feature selection can select the best fit features

- Filter
- Wrapper
- Embedded

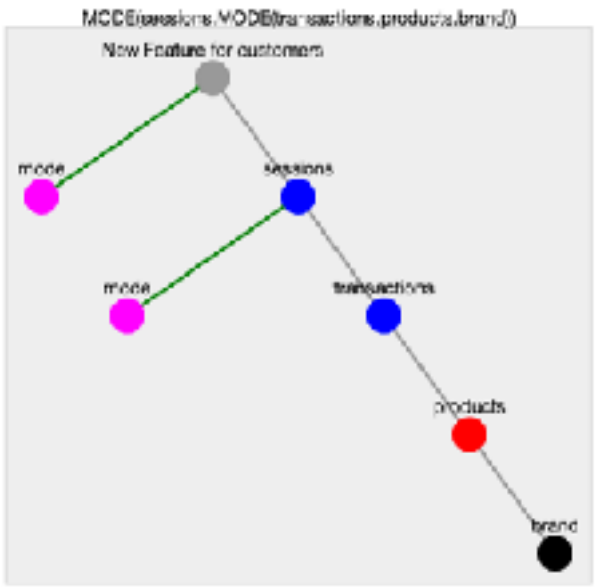
AutoML could be a logical extension for an end to end automation approach. AutoML is itself a very compute intensive approach

Ready for the build pipelines

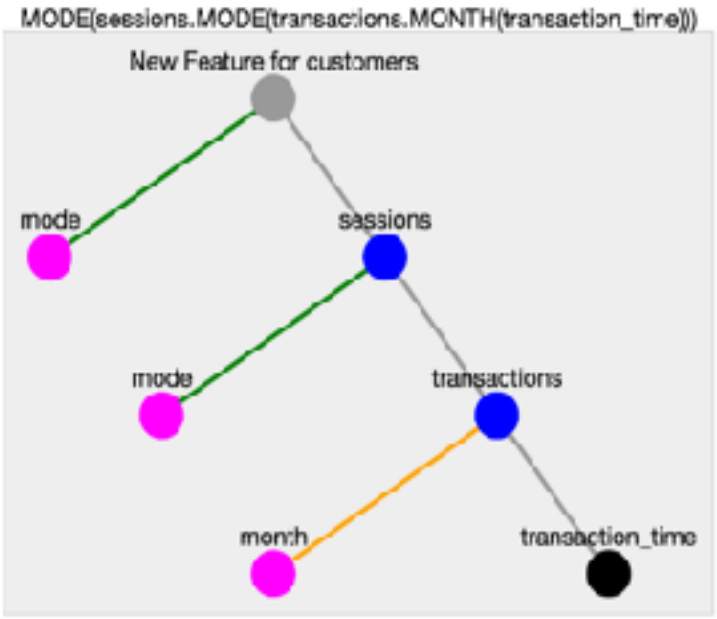
SEMANTIC INTERPRETATION

SEMANTIC TREE STRUCTURES

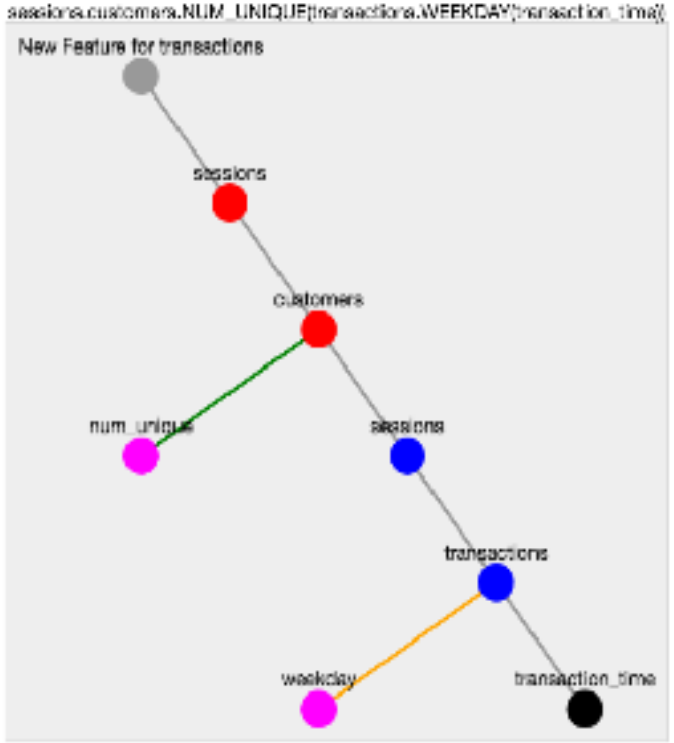
Customer Entity –
Aggregate features only



Customer Entity -
Mix of aggregate and transform featu



Transactions Entity -
Direct feature



BUILD PHASE CONSTRUCTS

RISK & SECURITY - GOVERNANCE

Path suppressions

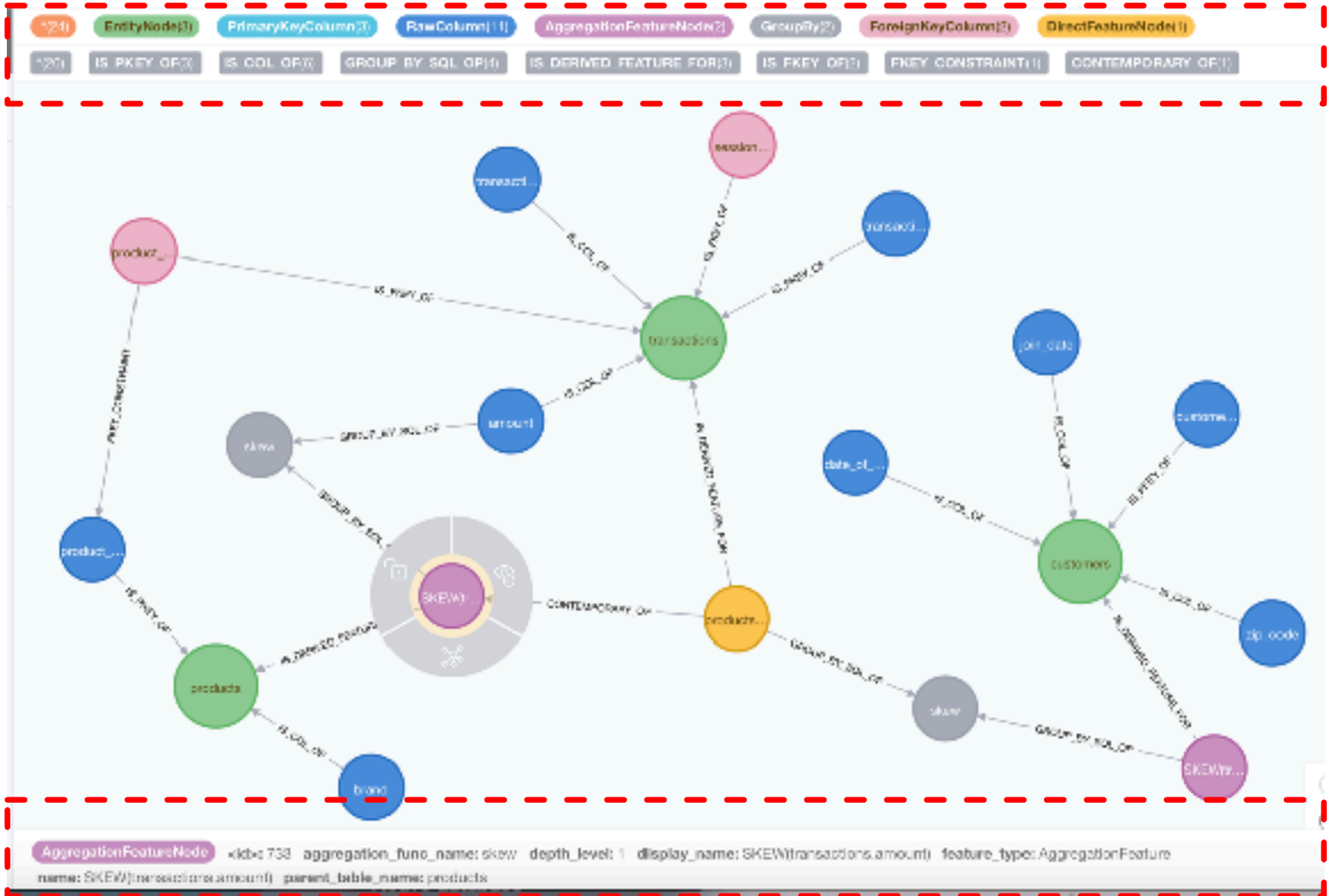
- **Disallow paths** to prevent features using certain stacking paths.
- Can be used to align with **data office regulations** of the enterprise.

Provenance and controls

- **Codified** feature engineering paths are a better model for provenance.
- Simplified **data access security** is a by-product of this approach.
- Better **CI/CD** controls
  - Break a build if a GDPR violation is part of a feature definition

LINEAGE ANALYSIS AND METADATA SYSTEMS INTEGRATION

- Metadata
  - Data types
  - Relational model
- Lineage
  - Feature types (How was a column derived)
  - SQL Operations



FEATURE ENGINEERING – TWO PATHS TO PRODUCTION

SQL driven pipelines

- **SQL** is the major pattern for data processing and transformation logic
- Supported by both **streaming** as well as **batch processing** engines.

Low latency pipelines

- Absolute low latency engines might opt for **NO-SQL** styled driven pipelines
- Feature tools is easier to port into this world as it supports **Ser-De for feature definitions**.

BATCH USE CASES - REPRESENTATIVE SQL COMPLEXITY

Depth 1 – COUNT(transactions)

```
SELECT
  COUNT(t.transaction_id),
  c.customer_id
FROM
  customers c
  JOIN sessions s ON s.customer_id = c.customer_id
  JOIN transactions t ON t.session_id = s.session_id
GROUP BY
  c.customer_id
```

BATCH USE CASES - REPRESENTATIVE SQL COMPLEXITY

Depth 2 – MAX(sessions.MEAN(transactions.amount))

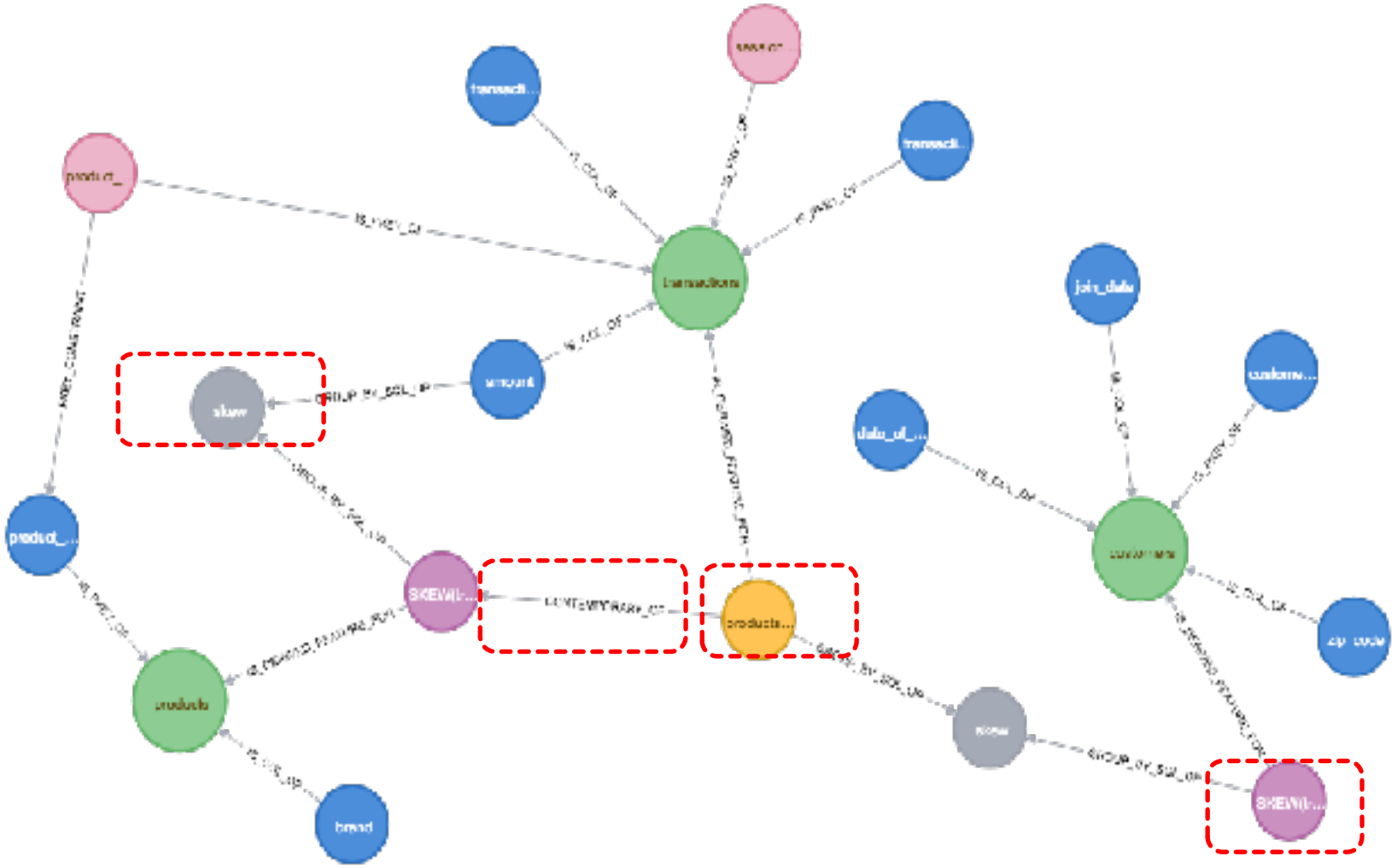
```
SELECT MAX(g.AmountMean), c.customer_id FROM customers c
    JOIN (
      SELECT s.customer_id as CustomerID, b.AmountMean from sessions s
        JOIN (
          SELECT s.session_id, MEAN(t.amount) as AmountMean FROM sessions s
              JOIN transactions t
              ON s.session_id = t.session_id
          GROUP BY s.session_id
          ) b
          ON s.session_id = b.session_id
       ) g
       ON c.customer_id = g.CustomerID
GROUP BY c.customer_id
```

BATCH USE CASES - REPRESENTATIVE SQL COMPLEXITY

Depth 3 - **STD(transactions.products.STD(transactions.amount))**

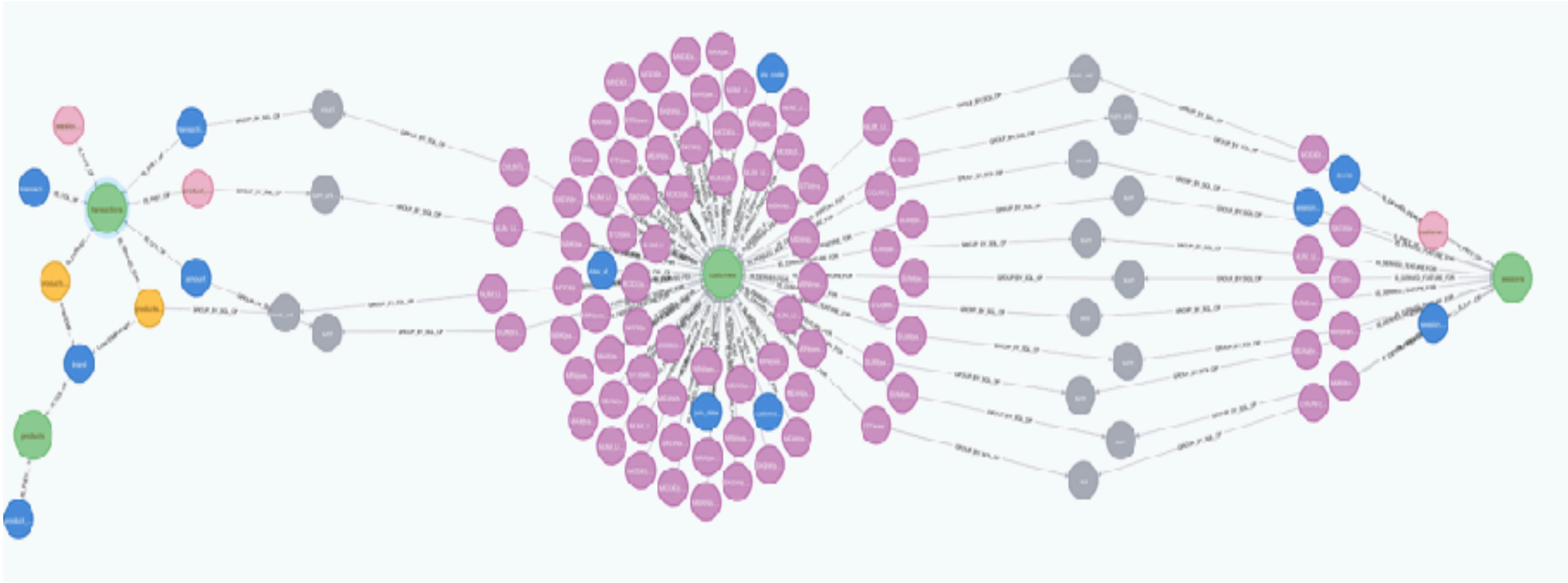
```
SELECT STD(y.AmountSTD), c.customer_id FROM customers c
       JOIN (
              SELECT s.customer_id, x.AmountSTD from sessions s
                     JOIN (
                            SELECT t.product_id, t.transaction_id, t.session_id, w.AmountSTD from transactions t
                                   JOIN (
                                          SELECT p.product_id, STD(t.amount) as AmountSTD FROM products p
                                                 JOIN transactions t
                                                 ON p.product_id = t.product_id
                                          GROUP BY p.product_id
                                          ) w
                                          ON t.product_id = w.product_id
                            ) x
                            ON s.session_id = x.session_id
             ) y
             ON c.customer_id = y.customer_id
GROUP BY c.customer_id
```

SQL GENERATION

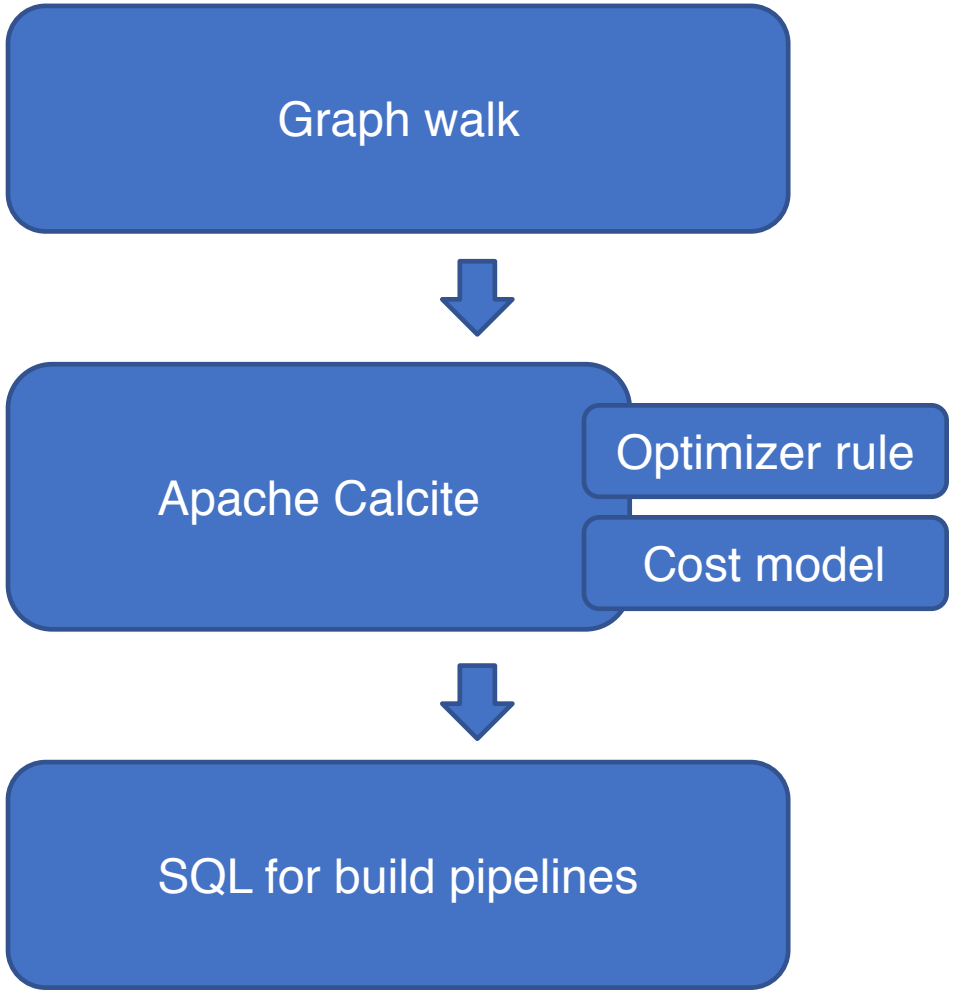


Customer feature - SKEW(transactions.products.SKEW(transactions.amount))

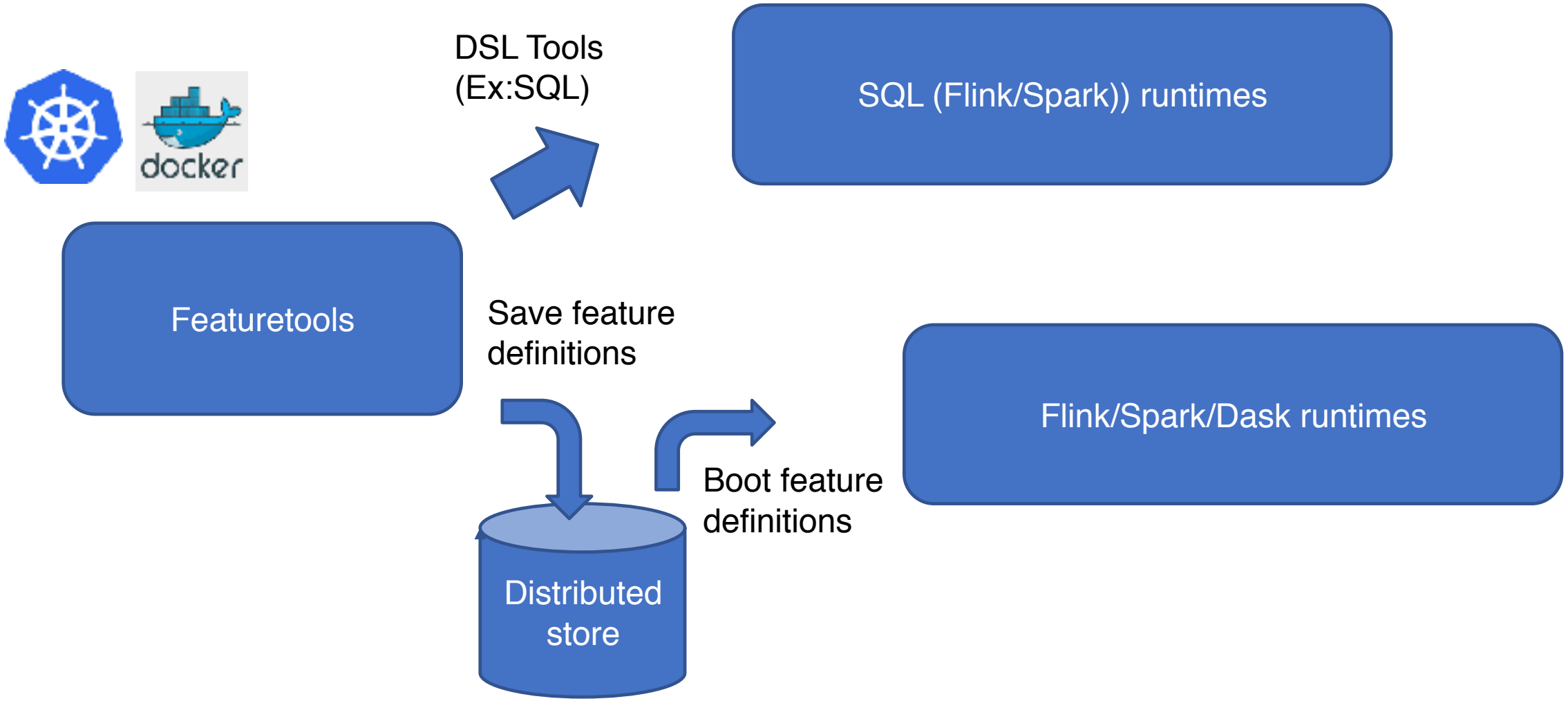
SEMANTIC FOREST STRUCTURES – PARTIAL/SUBSET VIEW



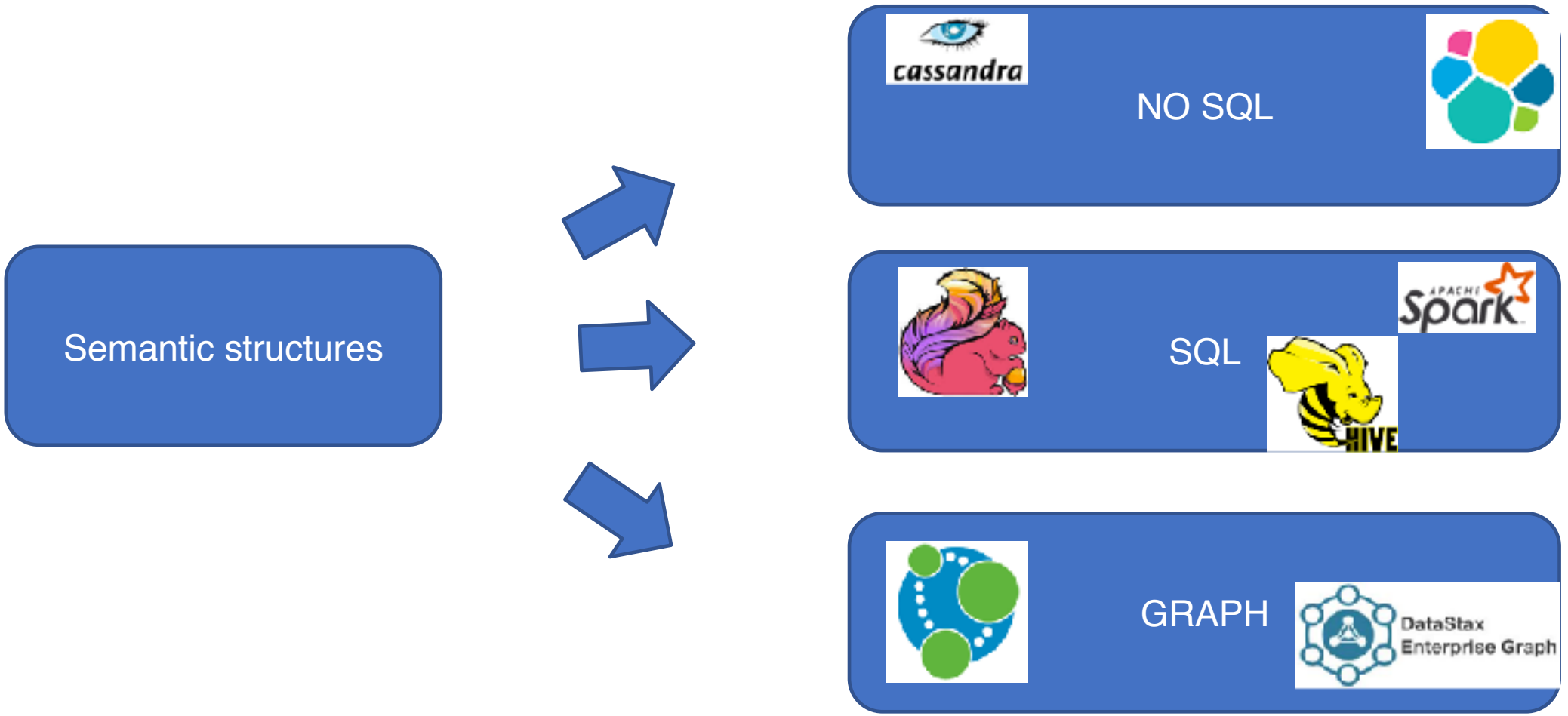
SQL OPTIMIZATION



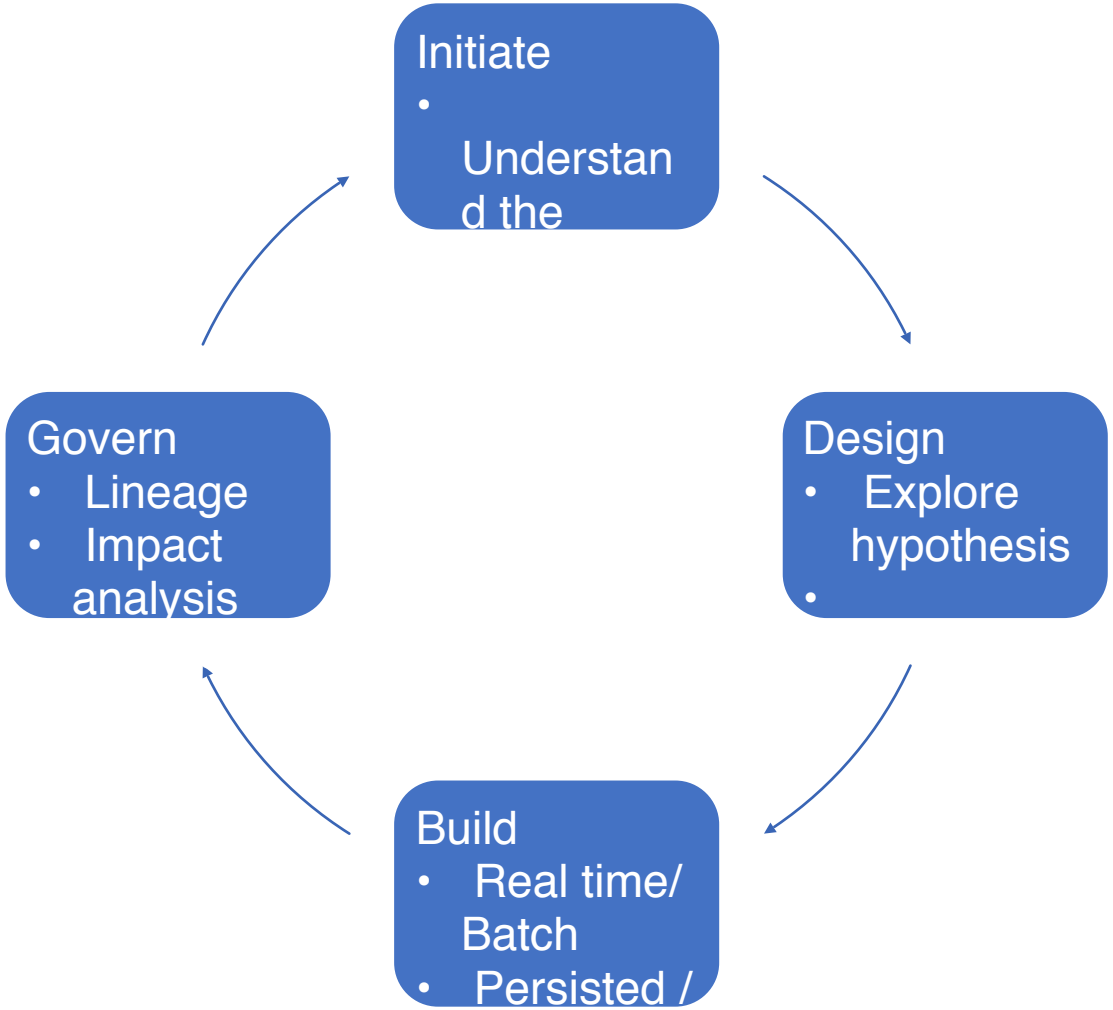
PIPELINE DEPLOYMENT PATTERNS



SEMANTIC STRUCTURES TO VARIED DATA SINKS



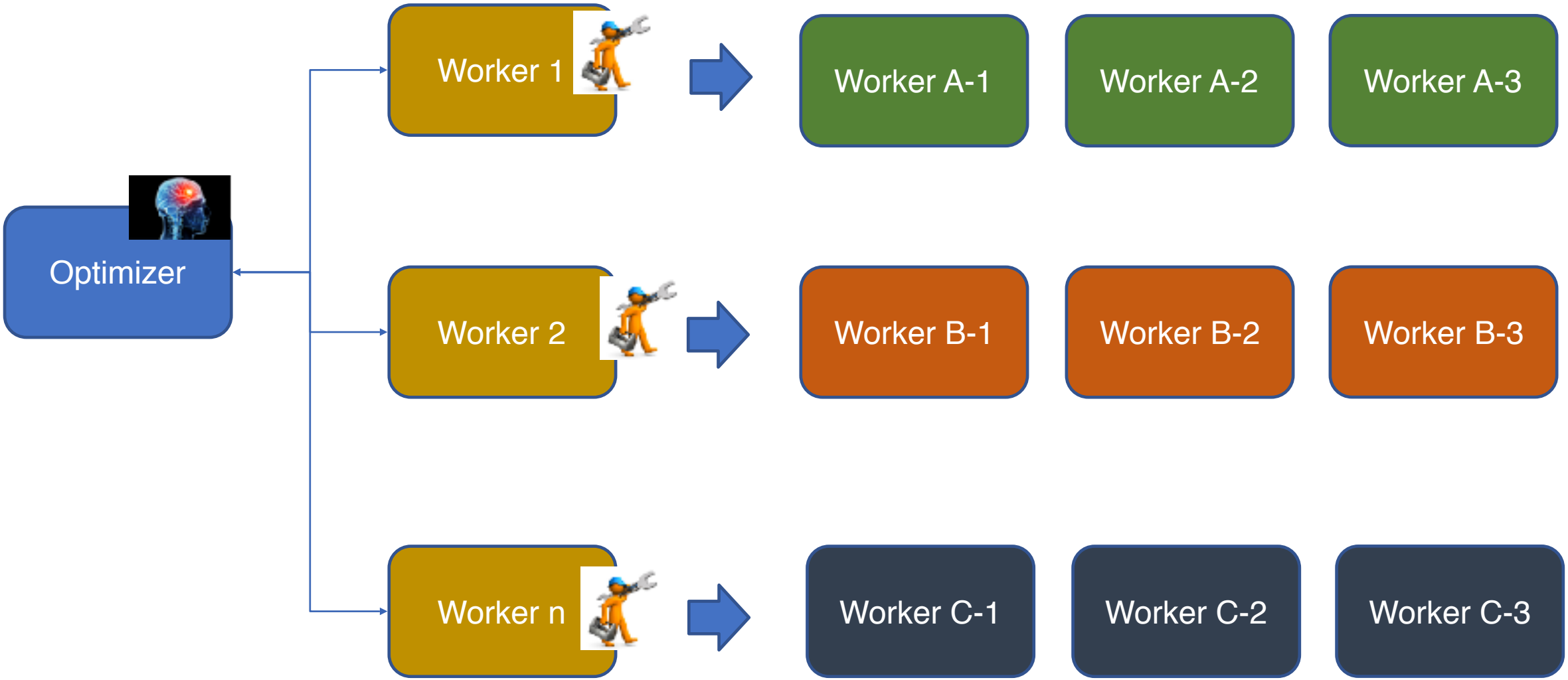
REVISIT LIFECYCLE – AUTOMATION CAN HELP



Q&A

THANK YOU

AUTOML


Optimizer
Worker 1
Worker 2
Worker n
Worker A-1
Worker A-2
Worker A-3
Worker B-1
Worker B-2
Worker B-3
Worker C-1
Worker C-2
Worker C-3