PIVOTAL NY – BIG DATA & ANALYTICS MEETUP

Prasad Radhakrishnan
Amey Banarse
Josh Plotkin

April 7th, 2015
Pivotal Labs, New York

SQL and Machine Learning on HDFS using HAWQ

Hadoop-HDFS

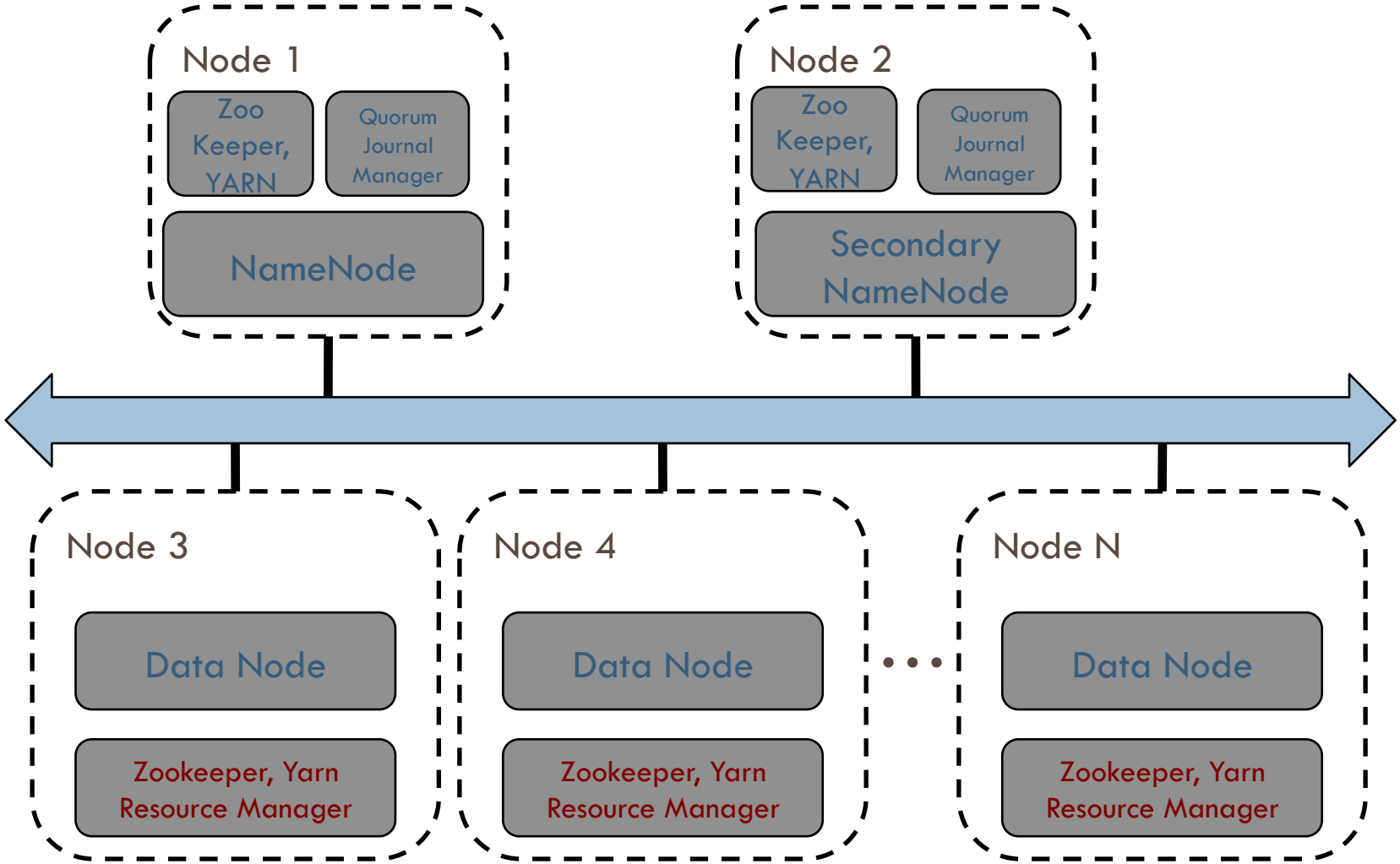
vs

MPP Databases

Quick overview of Hadoop-HDFS

- Hadoop is a Framework to store and process large volumes of data.
- HDFS is a fault-tolerant file system:
  - Assumes that Hardware will fail
  - Combines cluster's local storage into a single namespace.
  - All data is replicated to multiple machines and Infinitely (read highly) Scalable.
- A classic Master-Slave Architecture.

Hadoop & HDFS Architecture – Simplistic View



Pivotal

Hadoop – HDFS: 2 sides of the same coin

[……. Side Bar: This is the first US Penny, 223 years old. Sold for $1.2 mil……]

- How is Data stored in HDFS?
- How is Data accessed, how does the framework support processing / computations?

Hadoop – HDFS: Storage side

- Everything is a File.
- A File is broken into small blocks.
- Stores multiple copies of each block.
- If a node goes down, there are other nodes that can provide the missing blocks.
- Schema-less, it's a distributed FS that will accept any kind of data (Structured, un/semi-Structured).

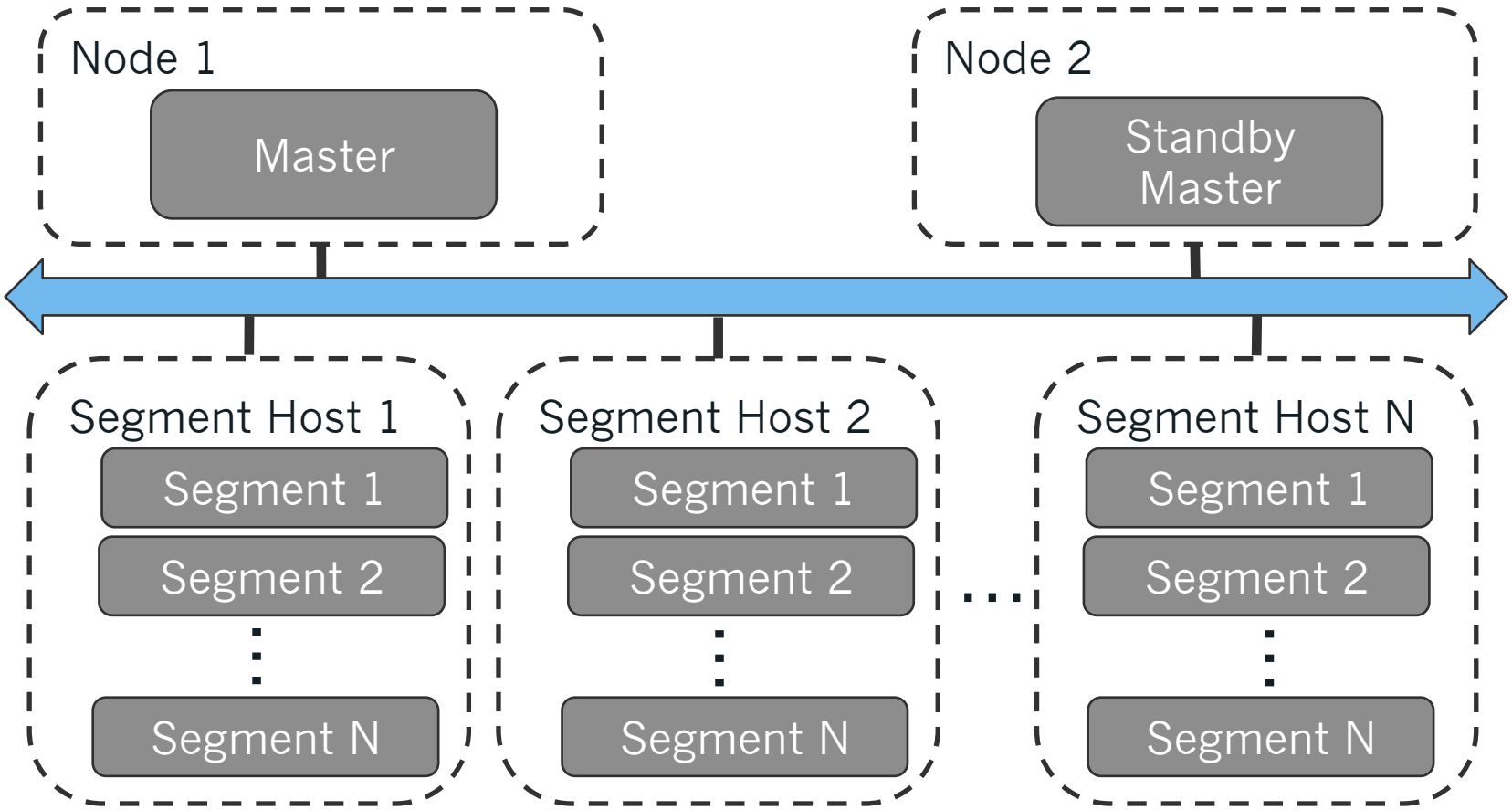
Hadoop – HDFS: Processing side

- Map-Reduce is the most popular processing framework for access and computational needs.
- MR is slow. Suited for batch work loads and to process un/semi-structured data.
- SQL support is minimal and not matured.

Quick overview of MPP Databases

- MPPs predate Hadoop-HDFS.
- Most MPPs have a Master-Slave design and a shared-nothing Architecture
- A very few software based MPPs, most run on custom h/w.
- They are known for their sheer horse power to process and compute big data work loads.

Simplistic view of Greenplum MPP DB Architecture



MPP Databases: 2 sides of the same coin

- How is Data stored in MPPs?
- How is Data accessed, how does the framework support processing / computations?

Pivotal

MPP Databases: Storage side of things

- MPPs too, have a distributed FS but only for a table's data. Tables are database objects.
- A File needs to be loaded into a table, pre-knowledge of it's schema is required.
- MPPs are fault-tolerant as well. They store redundant copies of data.
- Handling un/semi-structured data is not it's sweet spot and could get very challenging.

MPP Databases: Processing side

- MPPs are SQL engines.
- Known for their horse power in being able to handle & process big data work loads.
- Extensive/Full ANSI SQL support.
- Rich ETL and BI tools for easy dev & consumption.
- Some MPPs like Greenplum support Machine Learning use cases extensively.

The Winning Combination

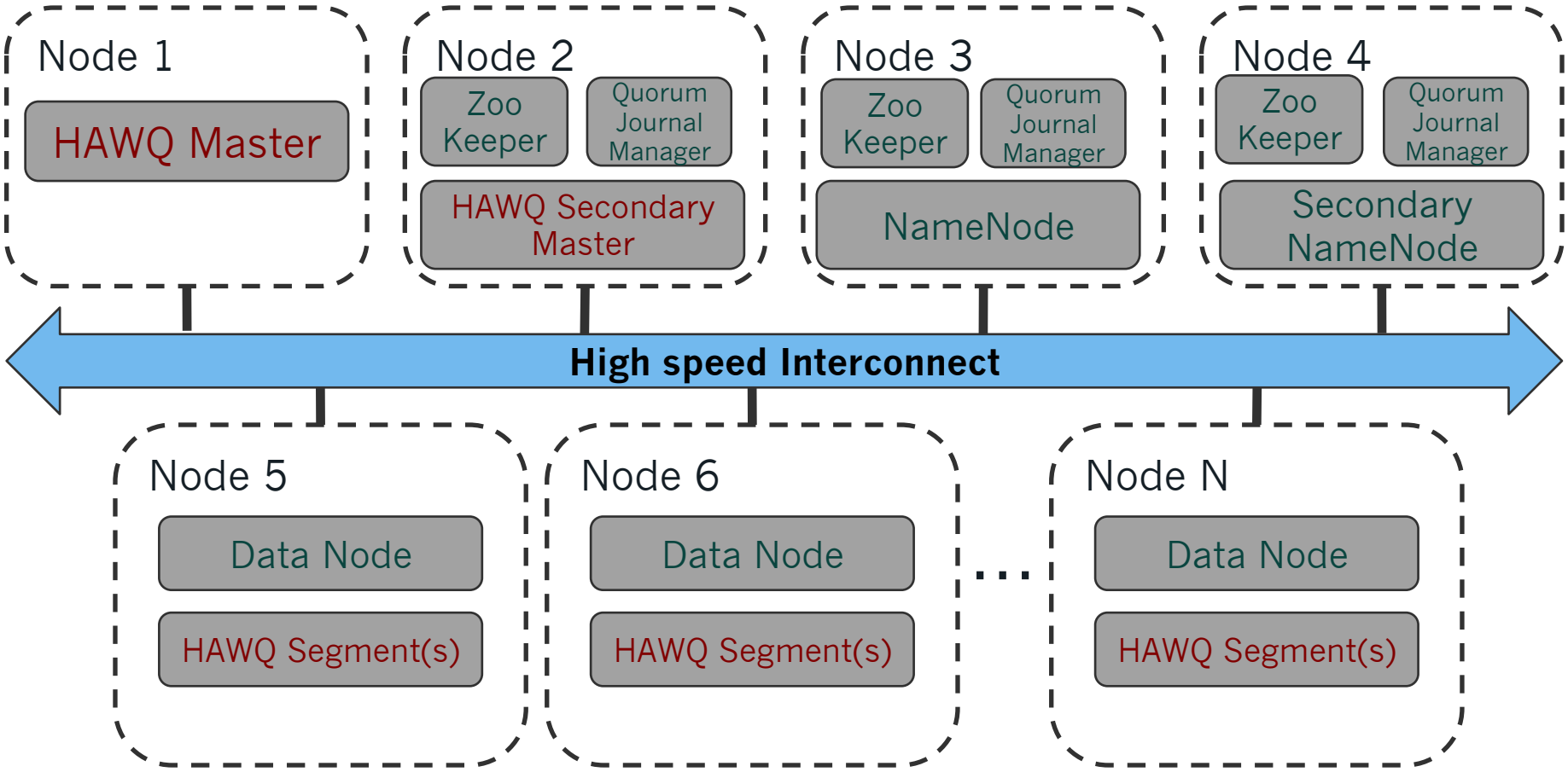
- HDFS as the distributed FS.
- Schema-less, fault-tolerant.
- Huge Apache eco-system of tools/products.

- MPP SQL Engine for Data access, processing and ML work loads.
- Blend with and leverage BI and ETL tools
- Blend with Hadoop File formats and integrate with Apache HDFS tools/products.

HAWQ, the MPP SQL Engine for HDFS

- Derived from Greenplum MPP, shortening many years engineering efforts that will go into building an MPP from scratch.
- Stores data in HDFS using Parquet file format.
- Familiar SQL interface, 100% ANSI compliant.
- All kinds of joins, analytical functions etc.
- Doesn't use MapReduce, no dependencies on Hive or Hive metastore.
- Way faster than MR, not even a valid comparison.
- Enables In-Database Machine Learning.

Pivotal HAWQ Architecture



Pivotal

Pivotal HAWQ Architecture

Master

- Master and Standby Master Host
- Master coordinates work with Segment Hosts

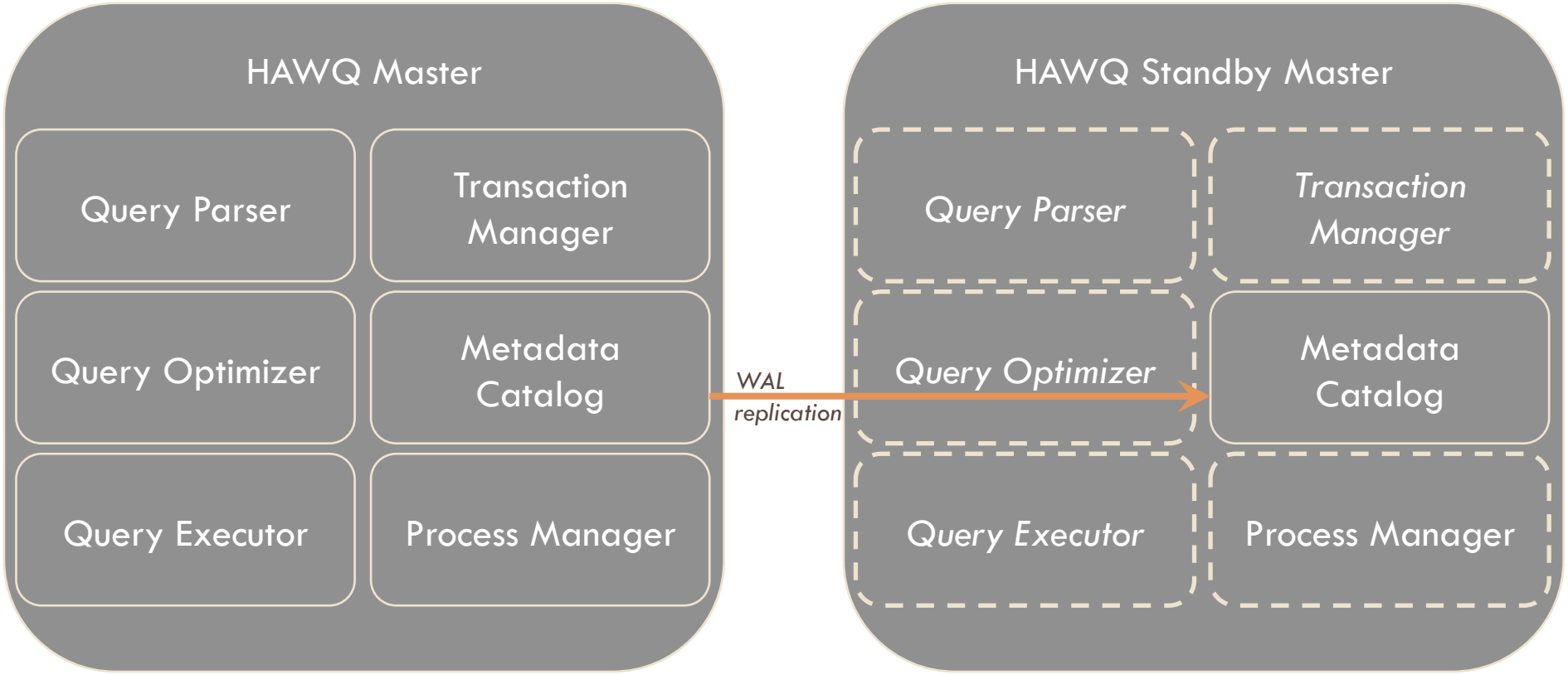
Segment

- One or more Segment Instances
- Process queries in parallel
- Dedicated CPU, Memory and Disk

Interconnect

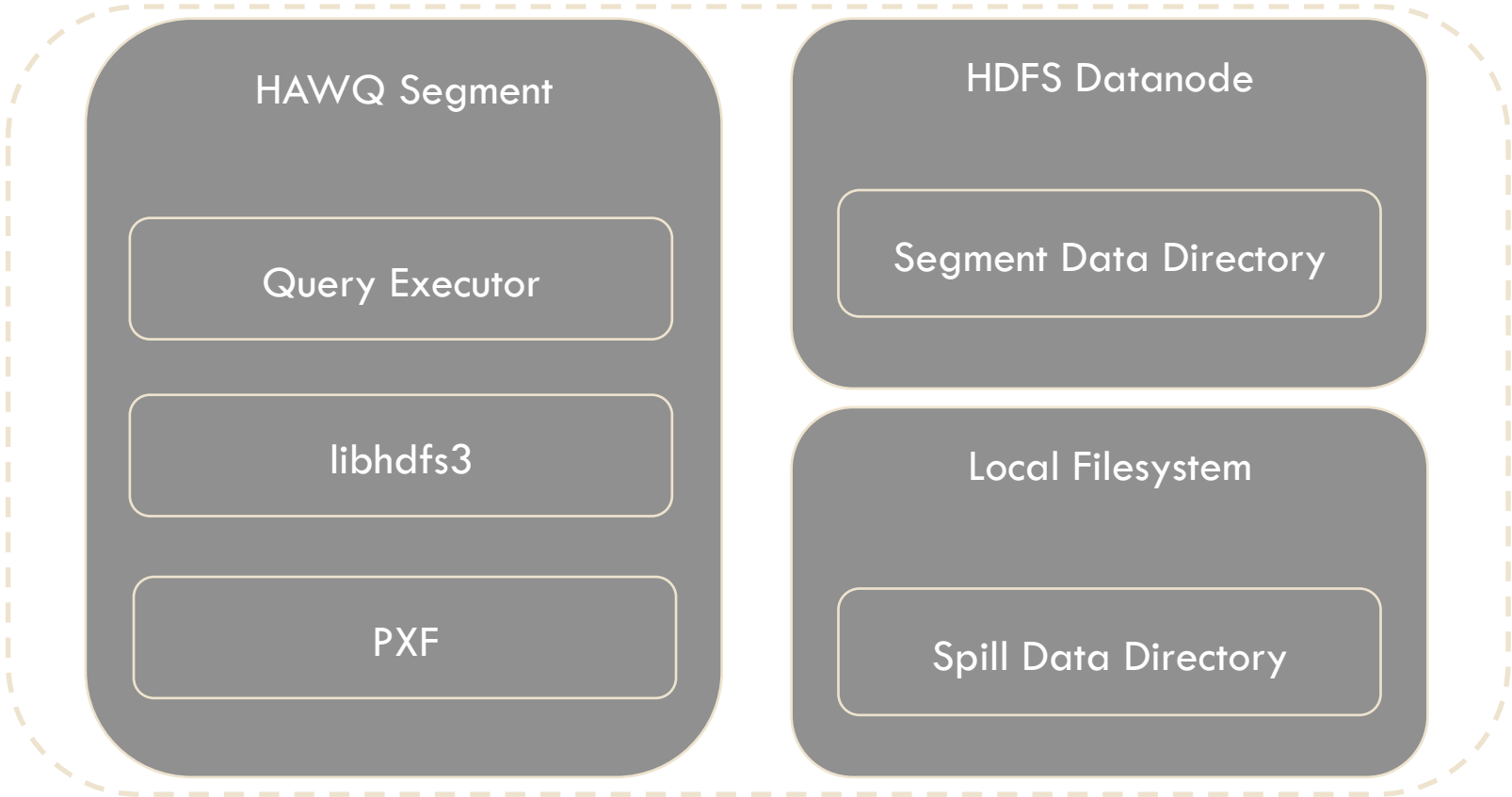
- High Speed Interconnect for continuous pipelining of data processing

Key components of HAWQ Master



Pivotal

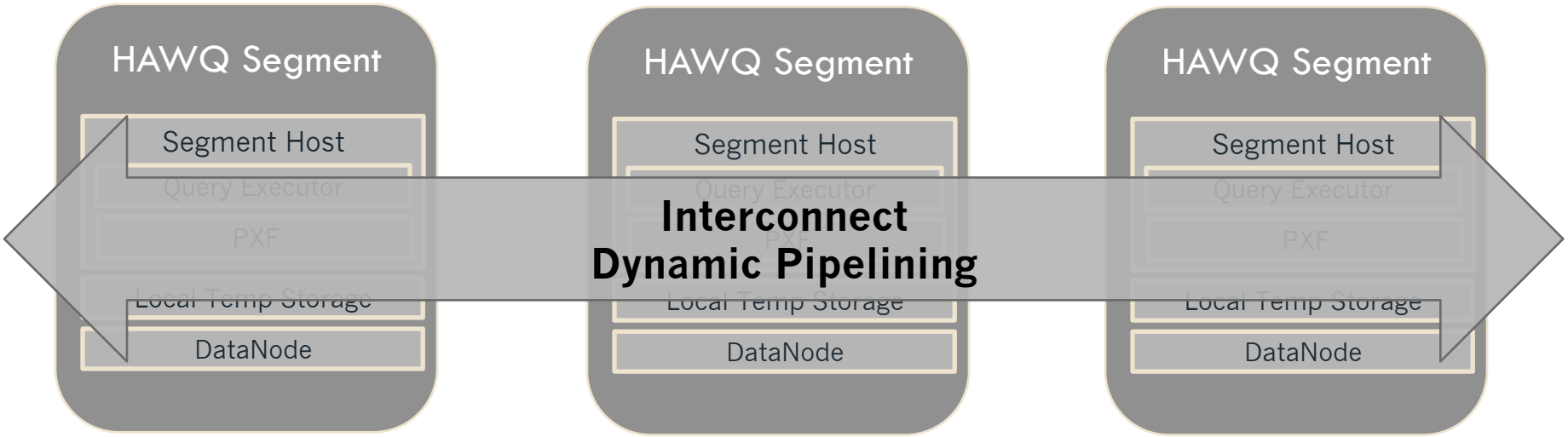
Key components of HAWQ segments



What differentiates HAWQ?

What differentiates HAWQ: Interconnect & Dynamic Pipelining

- Parallel data flow using the high speed UDP interconnect
- No materialization of intermediate data
  - Map Reduce will always "write" to disk, HAWQ will "spill" to disk only when insufficient memory
- Guarantees that queries will always complete



Pivotal

What differentiates HAWQ: libhdfs3

- Pivotal rewrote libhdfs in C++ resulting in libhdfs3
  - C based library
  - Leverages protocol buffers to achieve greater performance
- libhdfs3 is used to access HDFS from HAWQ
- Could benefit from short circuit reads
- Open sourced libhdfs3
  https://github.com/PivotalRD/libhdfs3

What differentiates HAWQ: HDFS Truncate

- HDFS is designed as append-only, POSIX-like FS
- ACID compliant –
  - HDFS guarantees the atomicity of Truncate operations i.e. either it succeeds or fails
- Added HDFS Truncate(HDFS-3107) to support rollbacks

What differentiates HAWQ: ORCA

- Multi-core query optimizer(codename ORCA)
- Cost based optimizer
- Window and Analytic Functions
- 1.2 Billion options evaluated in 250ms as an ex. for TPC-DS Query #21
  - Partition Elimination
  - Subquery Unnesting
- Presented at ACM SIGMOD 2014 Conference

Pivotal Xtensions Framework(PXF)

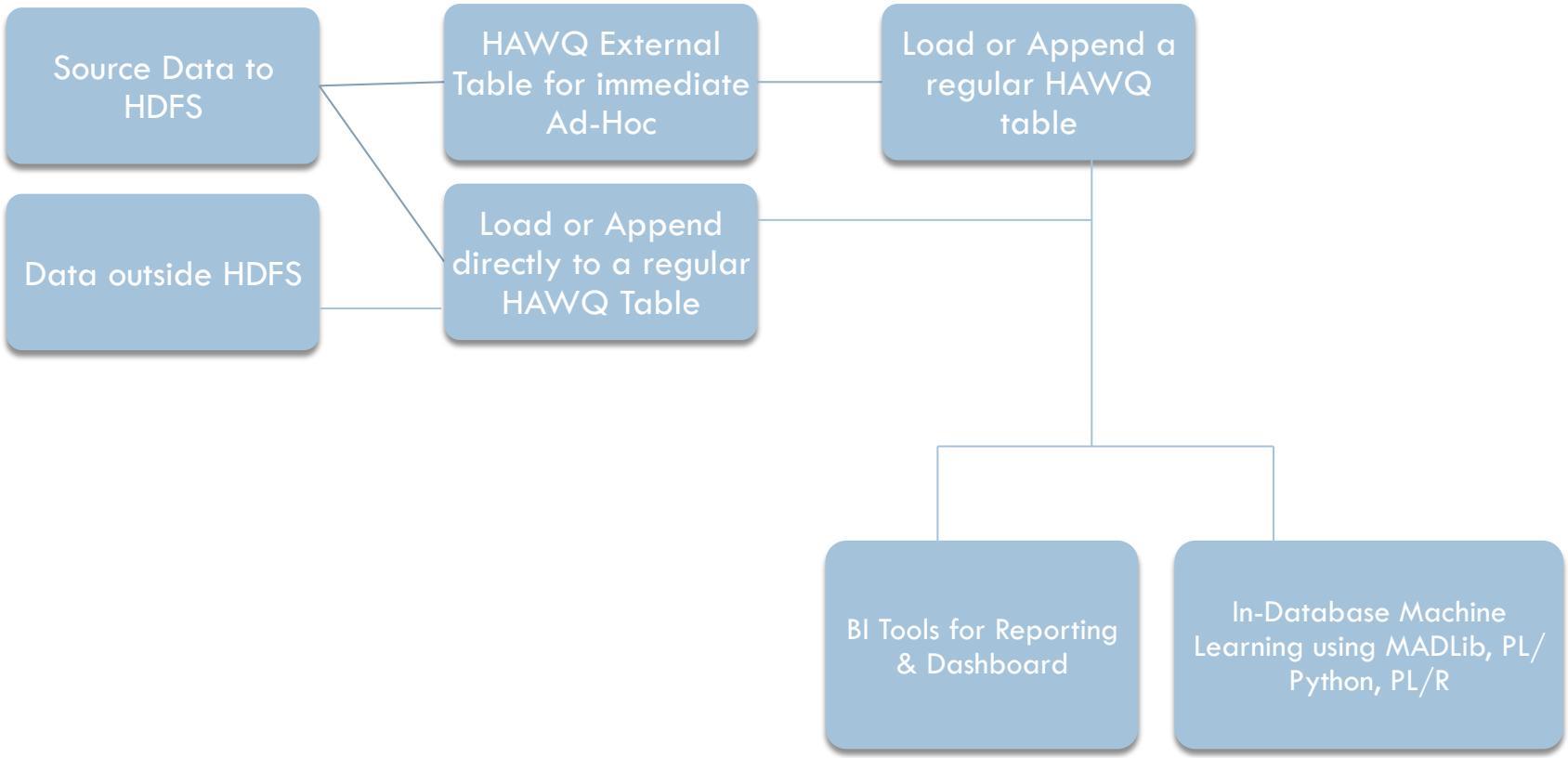
- External table interface in HAWQ to read data stored in Hadoop ecosystem
- External tables can be used to
  - Load data into HAWQ from Hadoop
  - Query Hadoop data without materializing it into HAWQ
- Enables loading and querying of data stored in
  - HDFS Text
  - Avro
  - HBase
  - Hive
    - Text, Sequence and RCFile formats

PXF example to query HBase

- Get data from an HBase table called 'sales'. In this example we are only interested in the rowkey, the qualifier 'saleid' inside column family 'cf1', and the qualifier 'comments' inside column family 'cf8'

```
CREATE EXTERNAL TABLE hbase_sales (
        recordkey bytea,
        "cf1:saleid" int,
        "cf8:comments" varchar
)
LOCATION('pxf://10.76.72.26:50070/sales?
          Fragmenter=HBaseDataFragmenter&
          Accessor=HBaseAccessor&
          Resolver=HBaseResolver')
FORMAT 'custom' (formatter='gpxfwritable_import');
```

Example Data Flow in HAWQ



HAWQ Demo

- HAWQ Cluster Topology
- HAWQ Ingest
- HAWQ PXF
- Dashboards

HAWQ Demo DataSet

- 23 Million rows of fake data, Customer Credit Card spend.
- Time Dimension
- Aggregate Tables

MACHINE LEARNING ON HDFS USING HAWQ

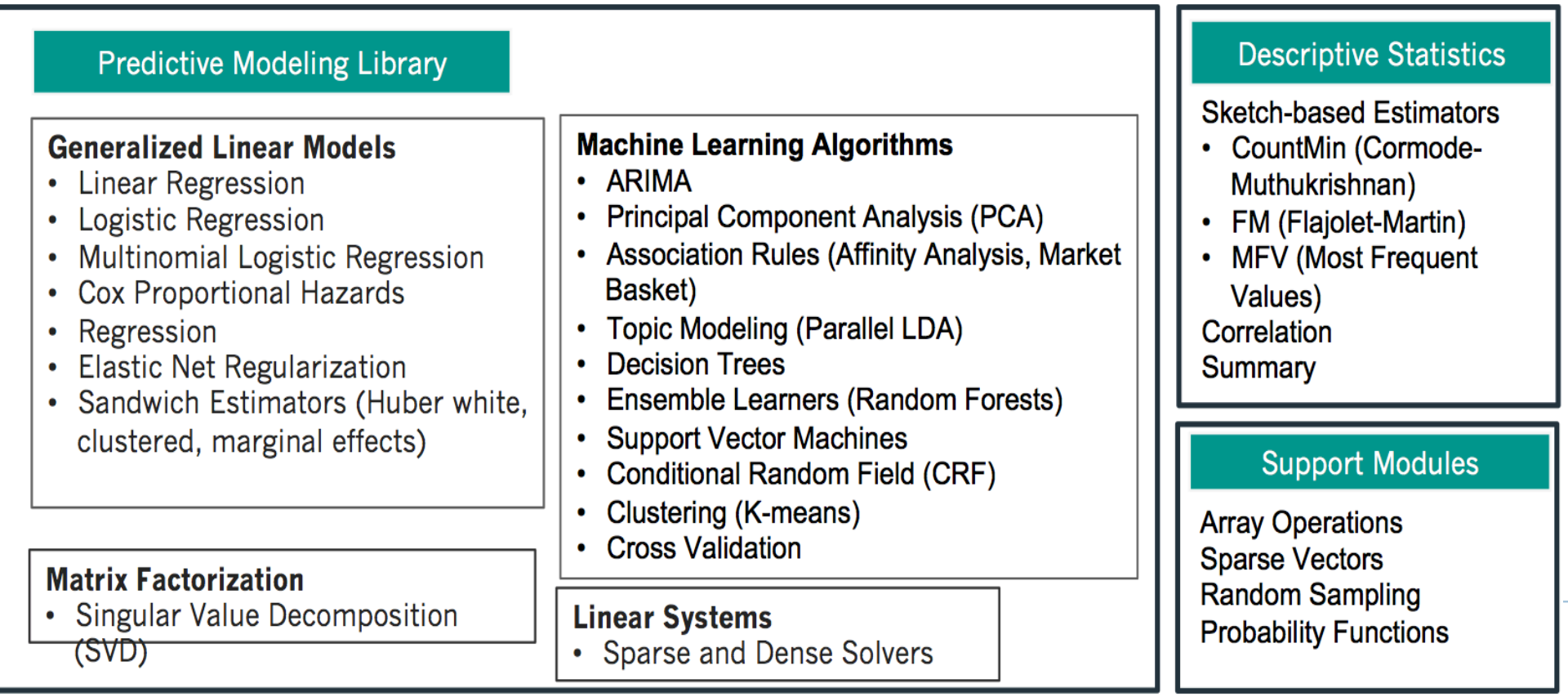
SQL and Machine Learning on HDFS using HAWQ

Analytics on HDFS, Popular Notions

- Store it all in HDFS and Query it all using HAWQ
  - Scalable Storage and Scalable Compute
- "Gravity" of Data changes everything
  - Move **code**, not data
  - Parallelize **everything**
  - Look at entire data set and not just a **sample**
  - More data could mean **simpler** models

MADLib for Machine Learning

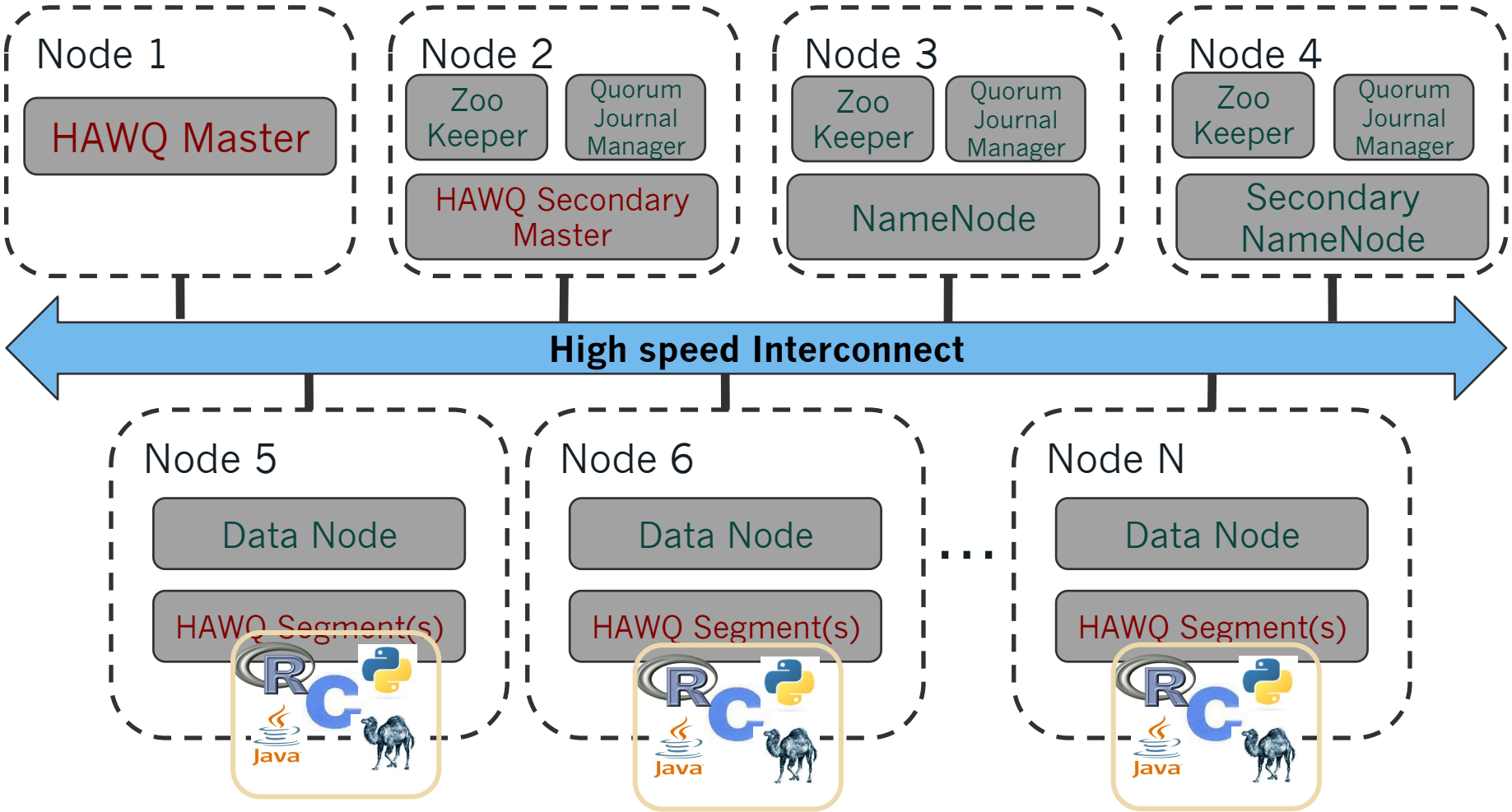
- MADlib is an open-source library for scalable in-database analytics. It provides data-parallel implementations of mathematical, statistical and machine learning methods for structured and unstructured data.



UDF - PL/X : X in {pgsql, R, Python, Java, Perl, C etc.}

- Allows users to write functions in the R/Python/Java, Perl, pgsql or C languages
- The interpreter/VM of the language 'X' is installed on each node of the Greenplum Database Cluster
- Can install Python extensions like Numpy, NLTK, Scikit-learn, Scipy.
- Data Parallelism: PL/X piggybacks on MPP architecture

UDF - PL/X : X in {pgsql, R, Python, Java, Perl, C etc.}

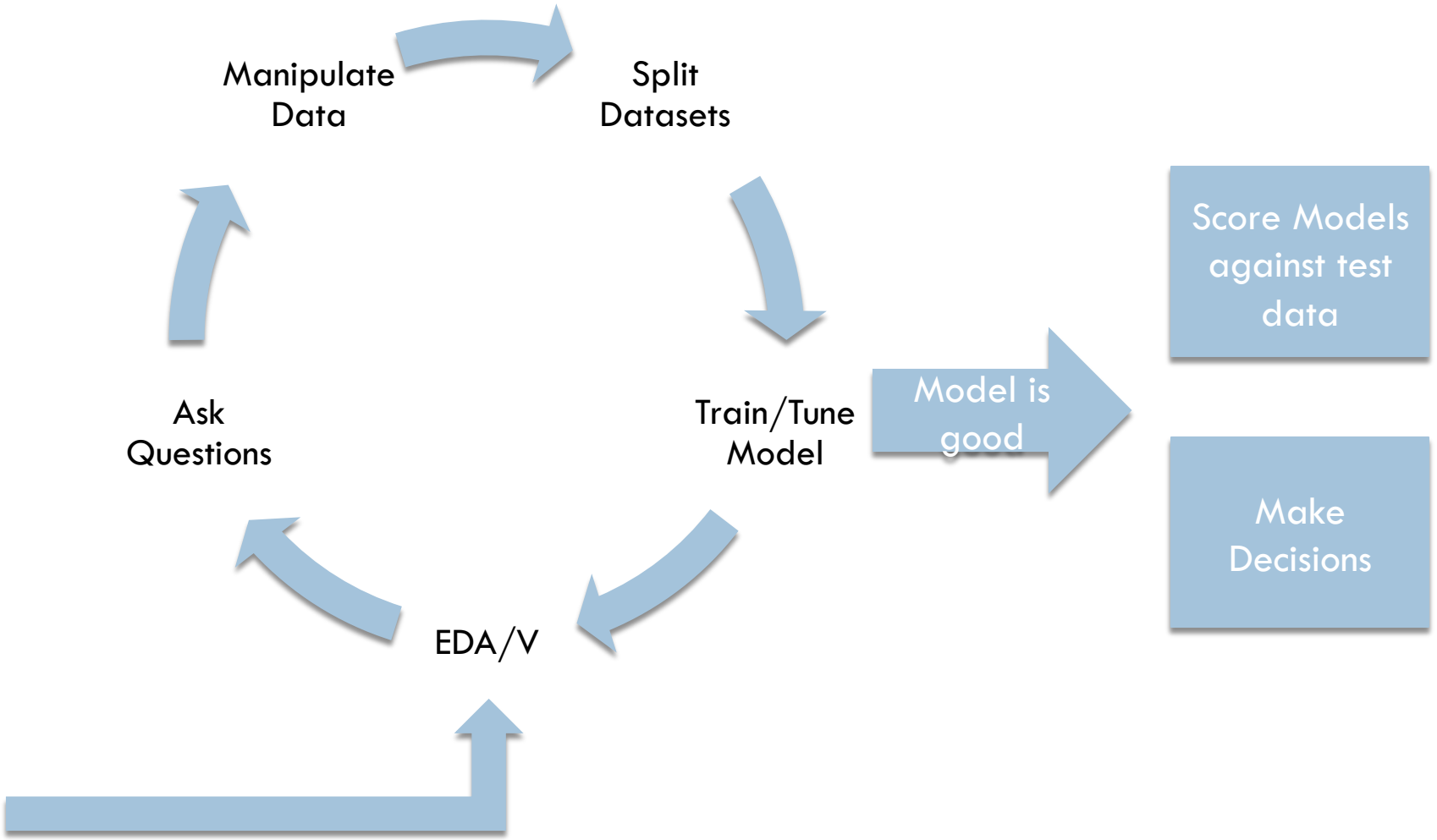


Pivotal

Data Analytics in SQL w/ HAWQ

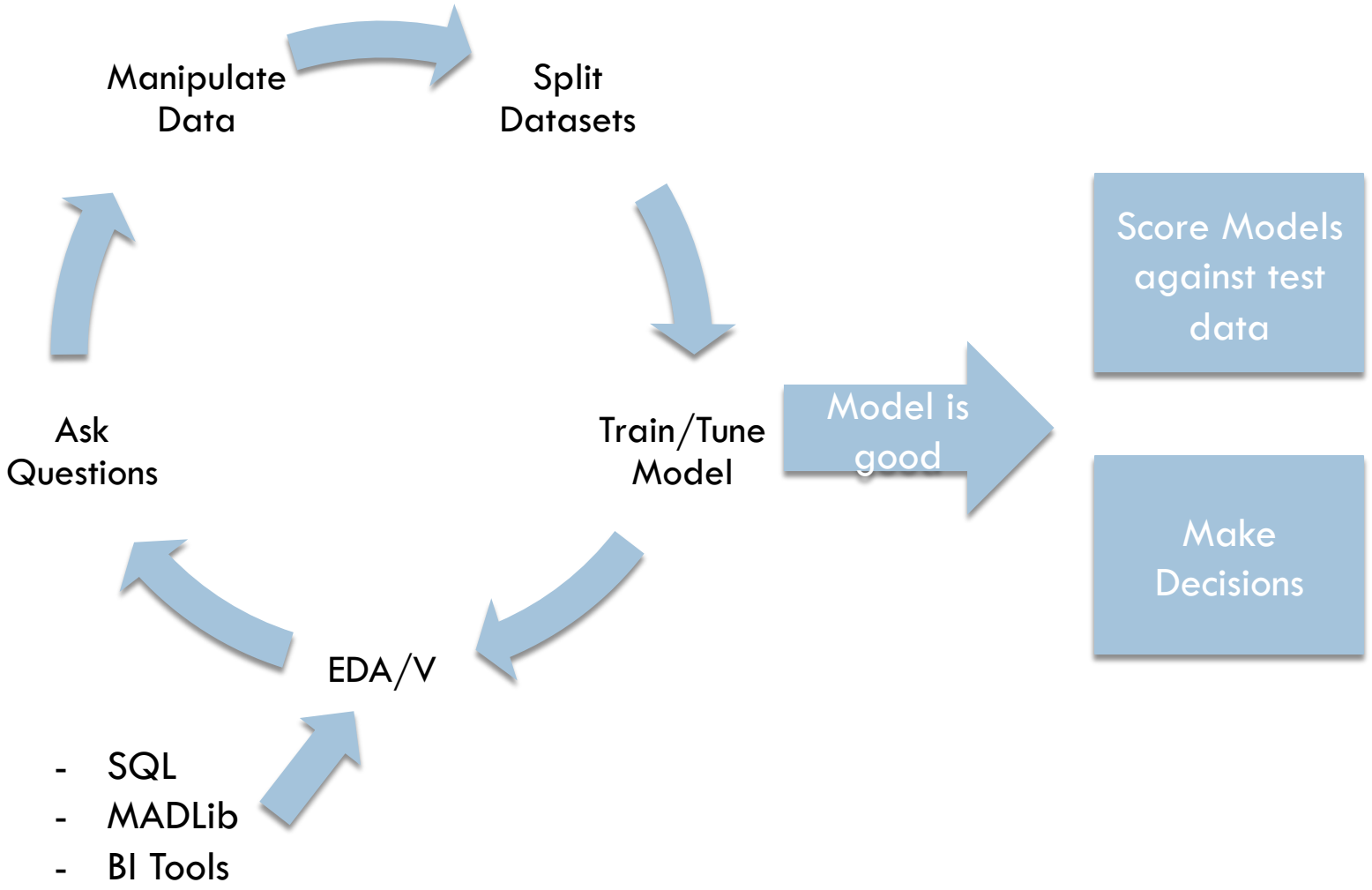
- A Data Science workflow in SQL
- Re-training is often necessary to take the step from traditional BI to DS
- Many parts in parallel
  - Data manipulation
  - Training (e.g. Linear regression, elastic net, SVM)
  - Model evaluation
- MADLib for native SQL interface
  - Simple syntax
- Extensible with PL/Python and PL/R
  - All your favorite packages

Workflow



Pivotal

Exploratory Data Analysis/Viz



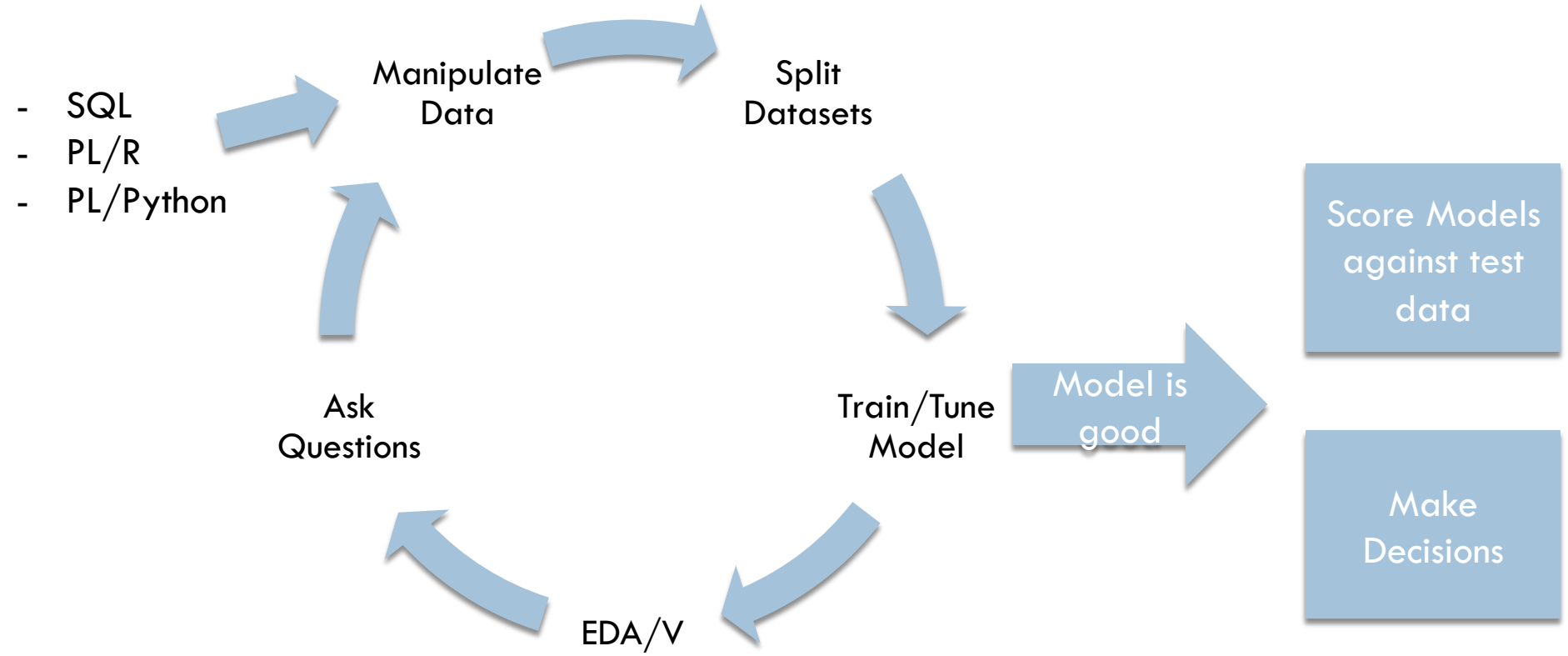
Explore Data with MADLib

- Select a row
- Group by customer (+ age and gender) to see total spend
  - Call it ‘summary_stats’
  - Select a row
- Use MADLib Summary Stats function
  - select madlib.summary('summary_stats', 'summary_stats_out’, ‘total_spent');
  - Creates a table ‘summary_stats_out’ we can query
  - Can group by: select madlib.summary('summary_stats', 'summary_stats_out_grouped', ‘total_spent’, ‘gender, age’);

Ask Questions

- Start with simple questions and simple (linear) models.
- Imagine we get new data with masked demographic data…
- What is the simplest possible question we can ask of this dataset?
  - Can we predict age by total amount spent?
- What needs to be done first?

Feature Engineering



Shaping the Dataset

- We want the data to look something like:

```
        ssn       |     sum        |  age
------------------+----------------------
621-95-5488 | 19435.72 |  30
```

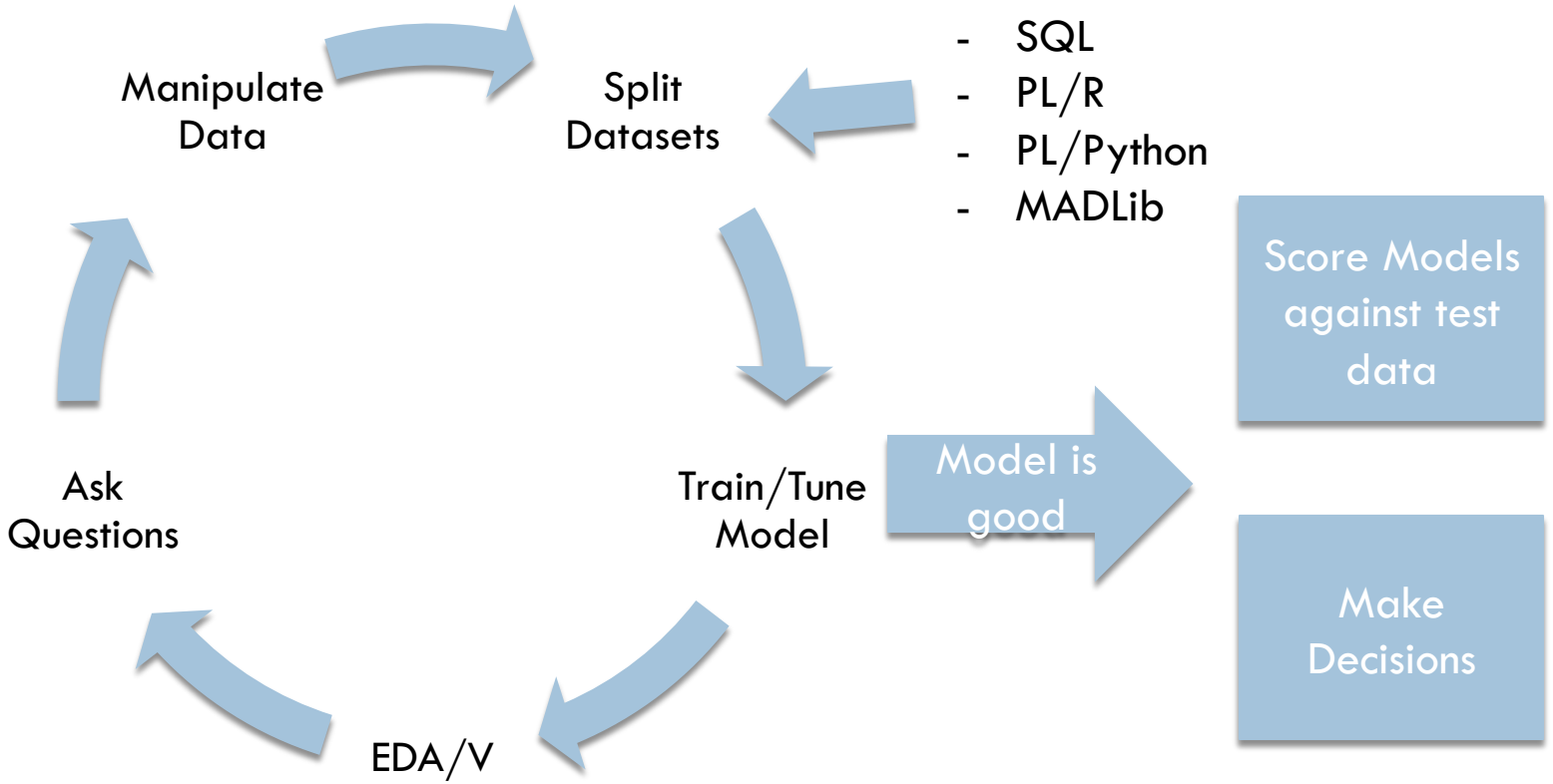
Shaping the Dataset

- We want the data to look something like:

```
        ssn       |     sum        |  age
------------------+----------------------
 621-95-5488 | 19435.72 |  30
```

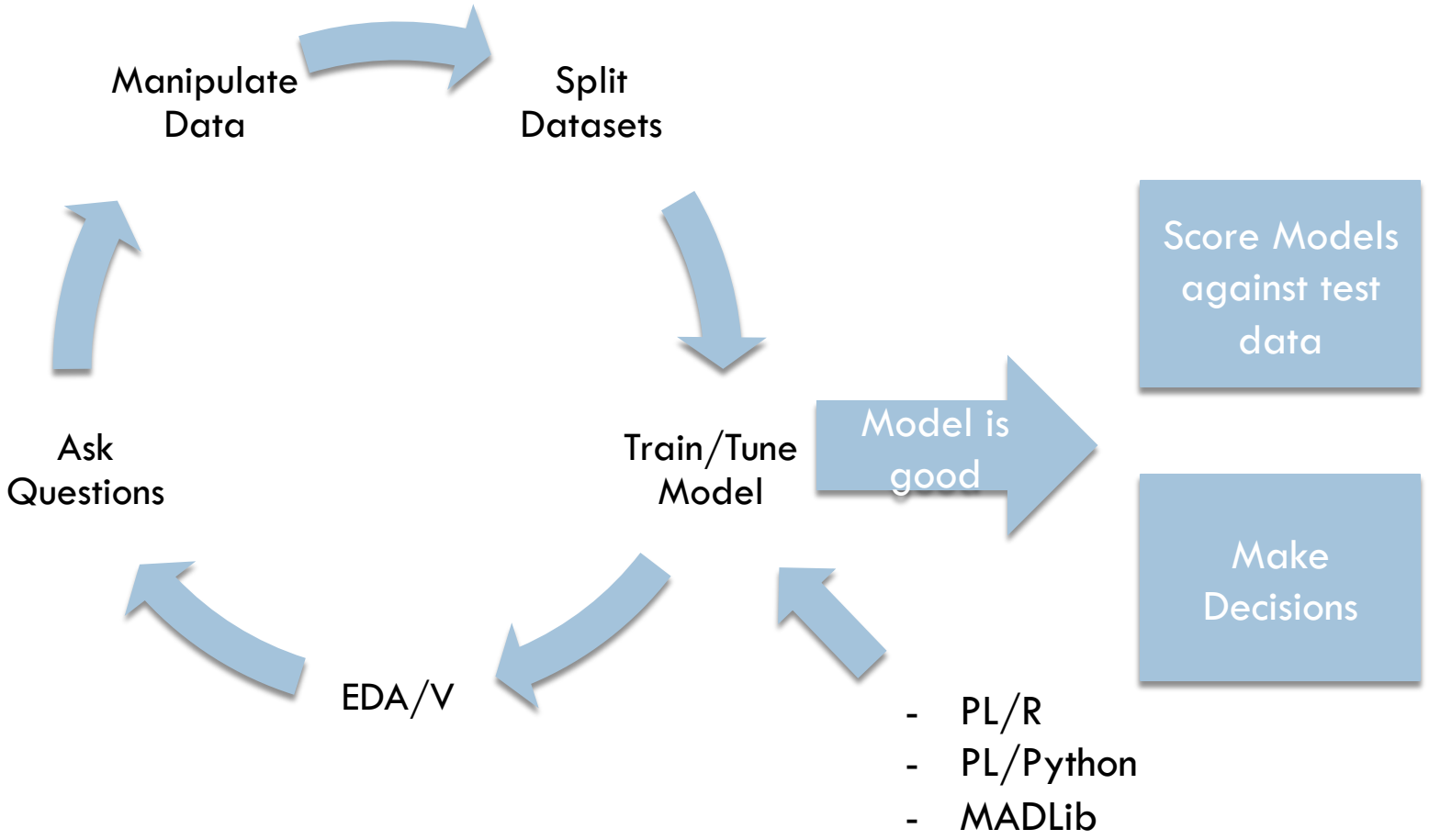
```
CREATE OR REPLACE VIEW cust_age_totalspend AS
SELECT ssn, sum(amt), extract(years from age(NOW(),dob)) as age
FROM mart.trans_fact
GROUP BY ssn, extract(years from age(NOW(),dob));
```

Skipping for now…



Pivotal

Building Models



Pivotal

MADLib's Interface

- Inputs: tables
  - VIEW: cust_age_totalspend
- Outputs: tables that can be queried
  - MADLib generates this automatically
  - This can be piped into PyMADLib, pulled via Tableau connector, or queriable from an API
- Labels
- Features
- Parameters
- Parallel when possible

MADLib's Interface

- Inputs: tables
  - VIEW: cust_age_totalspend
- Outputs: tables that can be queried
  - MADLib generates this automatically
  - This can be piped into PyMADLib, pulled via Tableau connector, or queriable from an API
- Labels
- Features
- Parameters
- Parallel when possible

```
SELECT madlib.linregr_train( 'cust_age_totalspend',
                             'linear_model',
                             'age',
                             'ARRAY[1, sum]');
```

MADLib's Interface

- Inputs: tables
  - VIEW: cust_age_totalspend
- Outputs: tables that can be queried
  - MADLib generates this automatically
  - This can be piped into PyMADLib, pulled via Tableau connector, or queriable from an API
- Labels
- Features
- Parameters
- Parallel when possible

```
SELECT madlib.linregr_train( 'cust_age_totalspend',
                             'linear_model',
                             'age',
                             'ARRAY[1, sum]');
```

MADLib's Interface

- Inputs: tables
  - VIEW: cust_age_totalspend
- Outputs: tables that can be queried
  - MADLib generates this automatically
  - This can be piped into PyMADLib, pulled via Tableau connector, or queriable from an API
- Labels
- Features
- Parameters
- Parallel when possible

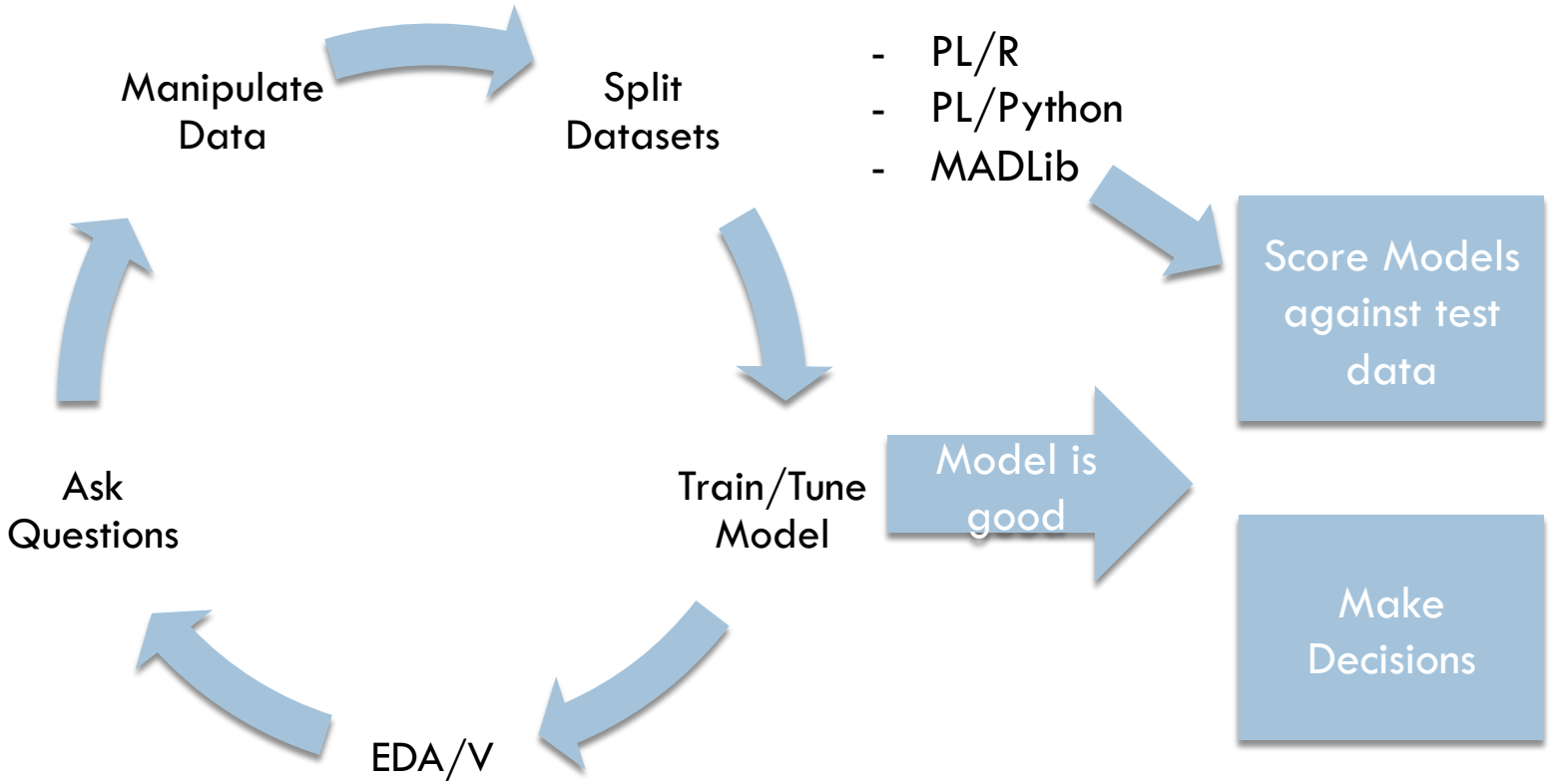
```
SELECT madlib.linre    train( 'cust_age_totalspend',
                              ear_model',
                              'age',
                              'ARRAY[1, sum]');
```

MADLib's Interface

- Inputs: tables
  - VIEW: cust_age_totalspend
- Outputs: tables that can be queried
  - MADLib generates this automatically
  - This can be piped into PyMADLib, pulled via Tableau connector, or queriable from an API
- Labels
- Features
- Parameters
- Parallel when possible

```
SELECT madlib.linregr_train( 'cust_age_totalspend',
                             'linear_model',
                             'age',
                             'ARRAY[1, sum]');
```

Scoring Chosen Model



Pivotal

Viewing Coefficients

- As simple as querying a table:

```
SELECT * FROM linear_model;
```

- Unnest arrays:

```
SELECT unnest(ARRAY['intercept', 'sum']) as attribute,
       unnest(coef) as coefficient,
       unnest(std_err) as standard_error,
       unnest(t_stats) as t_stat,
       unnest(p_values) as pvalue
FROM linear_model;
```

Making Predictions

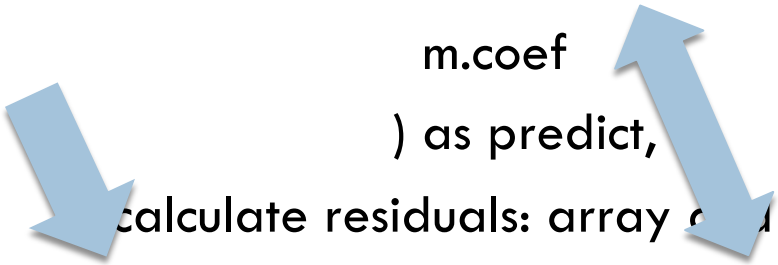
```
SELECT ssn, sum, age,
        -- calculate predictions: array and m.coef are vectors
        madlib.linregr_predict( ARRAY[1, sum],
                                m.coef
                              ) as predict,
        -- calculate residuals: array and m.coef are vectors
          age - madlib.linregr_predict( ARRAY[1, sum],
                                m.coef
                              ) as residual
FROM cust_age_totalspend, linear_model m;
```

Making Predictions

```
SELECT ssn, sum, age,
        -- calculate predictions: ar    and m.coef are vectors
        madlib.linregr_predict( ARRAY[1, sum],
                                m.coef
                              ) as predic
        -- calculate residuals: array and m.coef are vectors
          age - madlib.linregr_predict( ARRAY[1, sum],
                                        m.coef
                                      ) as residual
FROM cust_age_totalspend, linear_model m;
```

Making/Evaluating Predictions

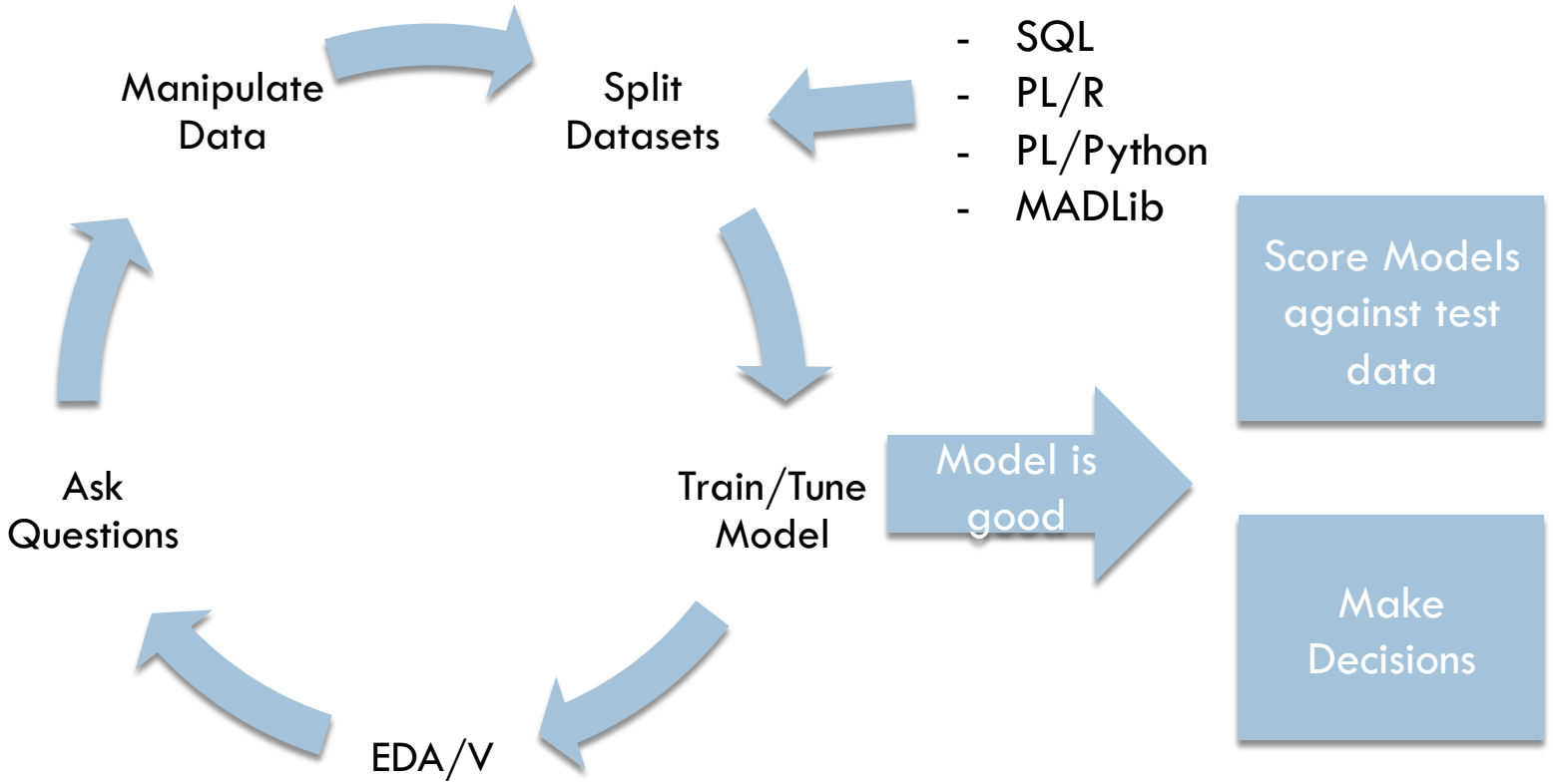
```
SELECT ssn, sum, age,
        -- calculate predictions: array and m.coef are vectors
        madlib.linregr_predict( ARRAY[1, sum],
                                m.coef
                              ) as predict,
        -- calculate residuals: array and m.coef are vectors
          age - madlib.linregr_predict( ARRAY[1, sum],
                                        m.coef
                                      ) as residual
FROM cust_age_totalspend, linear_model m;
```



Scoring (Mean Squared Error)

```
SELECT avg(residual^2) AS mse
FROM (SELECT age - madlib.linregr_predict( ARRAY[1, sum],
                                           m.coef
                                         ) as residual
FROM cust_age_totalspend, linear_model m) PREDICTIONS;
```

Training, Testing, Cross-Validation Sets



Pivotal

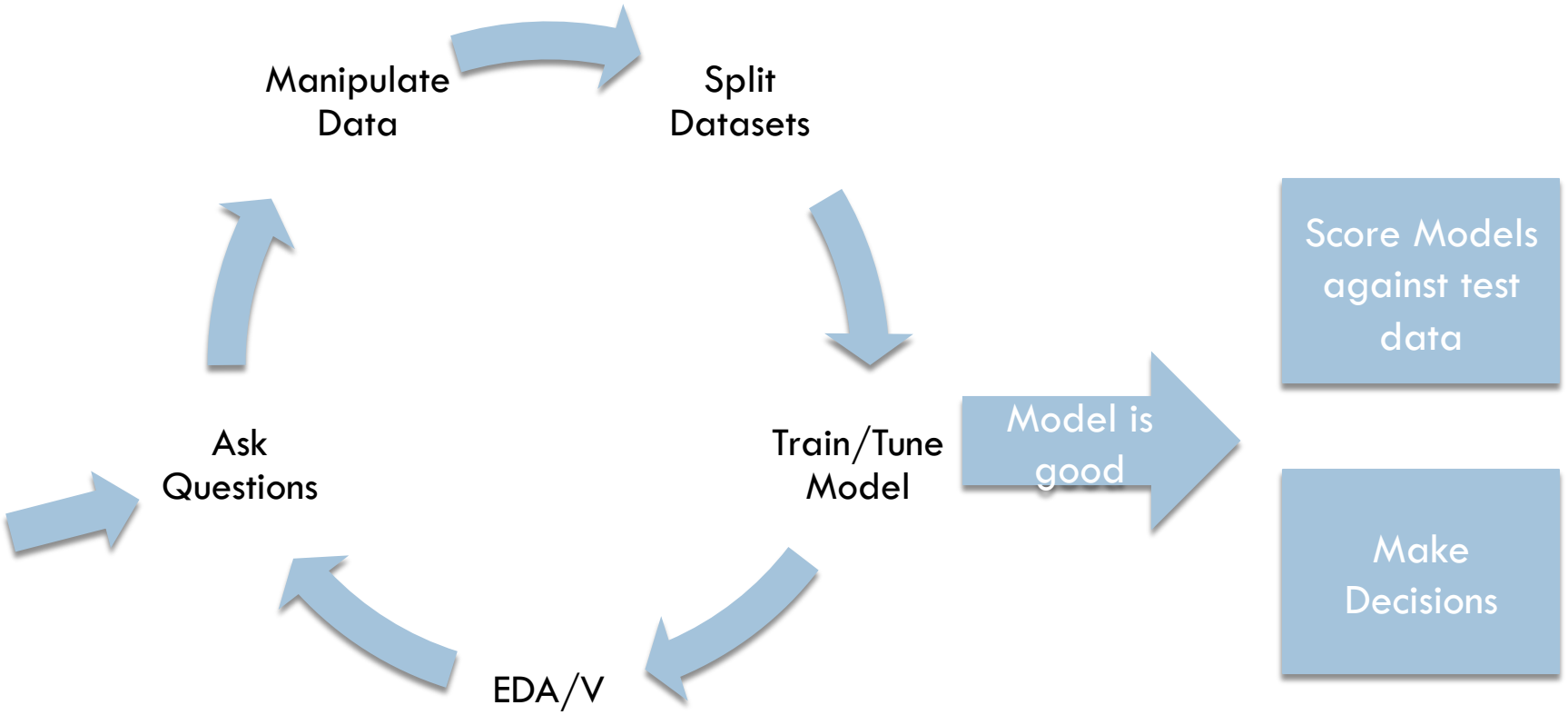
SQL To Create 80/20 Test/Train

- MADLib has a built-in cross validation function
- Stratified Sampling with pure SQL
  - Random seed (-1, 1):
    - SELECT setseed(0.444);
    - Partition the data by group
    - Sort by the random #
    - Assign a row count
    - Use a correlated subquery to select the lowest 80% of row counts by partition into the training set
  - The code…

SQL To Create 80/20 Test/Train

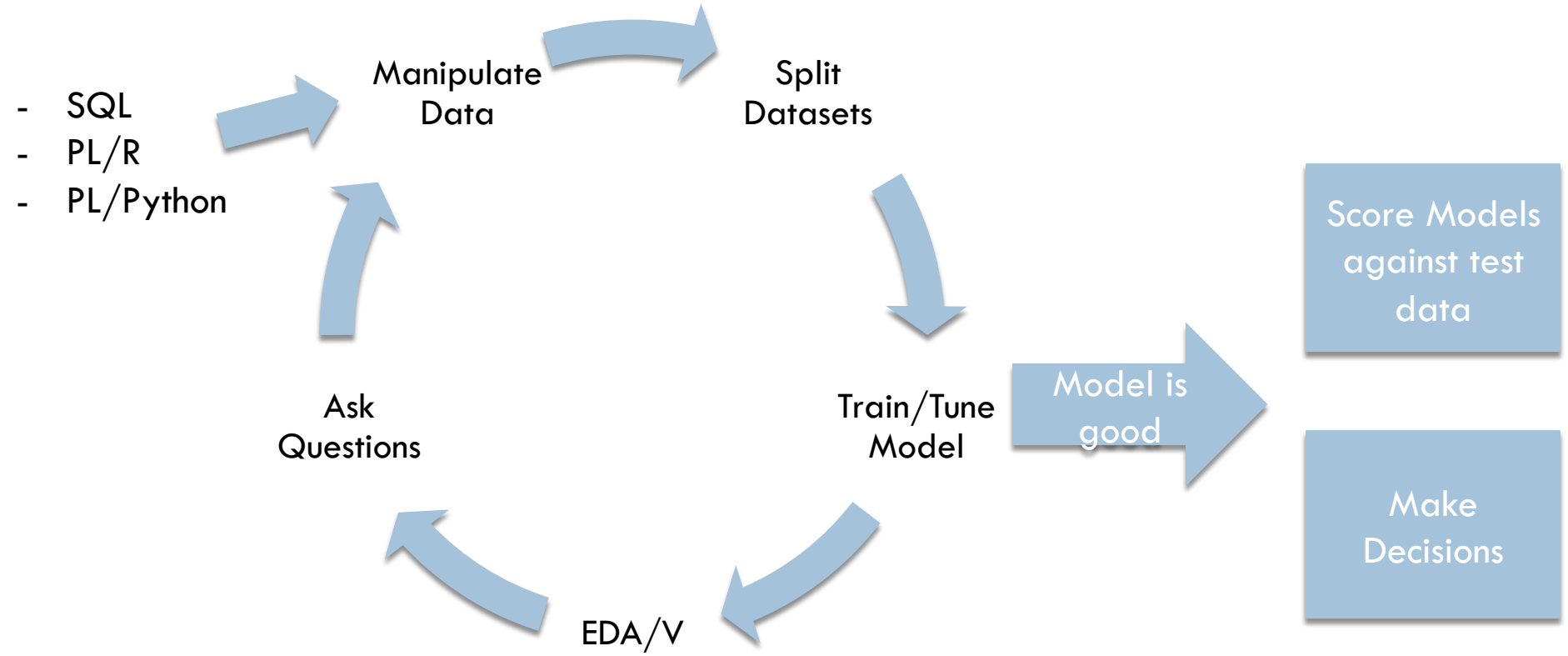
```
-- 1. perform the same aggregation step as before in creating the view
WITH assign_random AS (
    SELECT ssn, extract(years from age(NOW(),dob)) as age, sum(amt) as total_spent, random() as random
    FROM mart.trans_fact
    GROUP BY ssn, extract(years from age(NOW(),dob))
),
-- 2. add a different row count for group type (in this case, just age)
training_rownums AS (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY age ORDER BY random) AS rownum FROM assign_random
)
-- 3. sample the lowest 80% of rownums as the training set by group using
----- a correlated subquery (iterate over groups) because HAWQ is ANSI complient
SELECT ssn, age, total_spent
    FROM training_rownums tr1
    WHERE rownum <= (
        (SELECT MAX(rownum) * 0.8 FROM training_rownums tr2 WHERE tr1.age = tr2.age
    )));
```

Another question



Pivotal

Back Here Again…



Shaping the Dataset

- Remember the initial dataset:
  - select ssn, gender, trans_date, category, round(amt, 2) as amount from mart.trans_fact limit 5;

```
    ssn   | gender | trans_date |    category    | amount
------------+--------+------------+----------------+-------
 587-27-8957 | M      | 2013-12-21 | health_fitness | 10.59
 587-27-8957 | M      | 2014-10-19 | shopping_pos   |  0.80
 587-27-8957 | M      | 2013-05-04 | entertainment  |  5.17
 587-27-8957 | M      | 2014-03-27 | utilities      |  9.59
 587-27-8957 | M      | 2013-03-30 | home           | 10.64
```

- How can we shape the data?
  - Transpose using case statements
  - Aggregate and sum
  - The query…

Shaping the Dataset

- Remember the initial dataset:
  - select ssn, gender, trans_date, category, round(amt, 2) as amount from mart.trans_fact limit 5;

```
    ssn   | gender | trans_date |    category    | amount
------------+--------+------------+----------------+-------
 587-27-8957 | M      | 2013-12-21 | health_fitness | 10.59
 587-27-8957 | M      | 2014-10-19 | shopping_pos   |  0.80
 587-27-8957 | M      | 2013-05-04 | entertainment  |  5.17
 587-27-8957 | M      | 2014-03-27 | utilities      |  9.59
 587-27-8957 | M      | 2013-03-30 | home           | 10.64
```

- How can we shape the data?
  - Transpose using case statements
  - Aggregate and sum
  - The query…

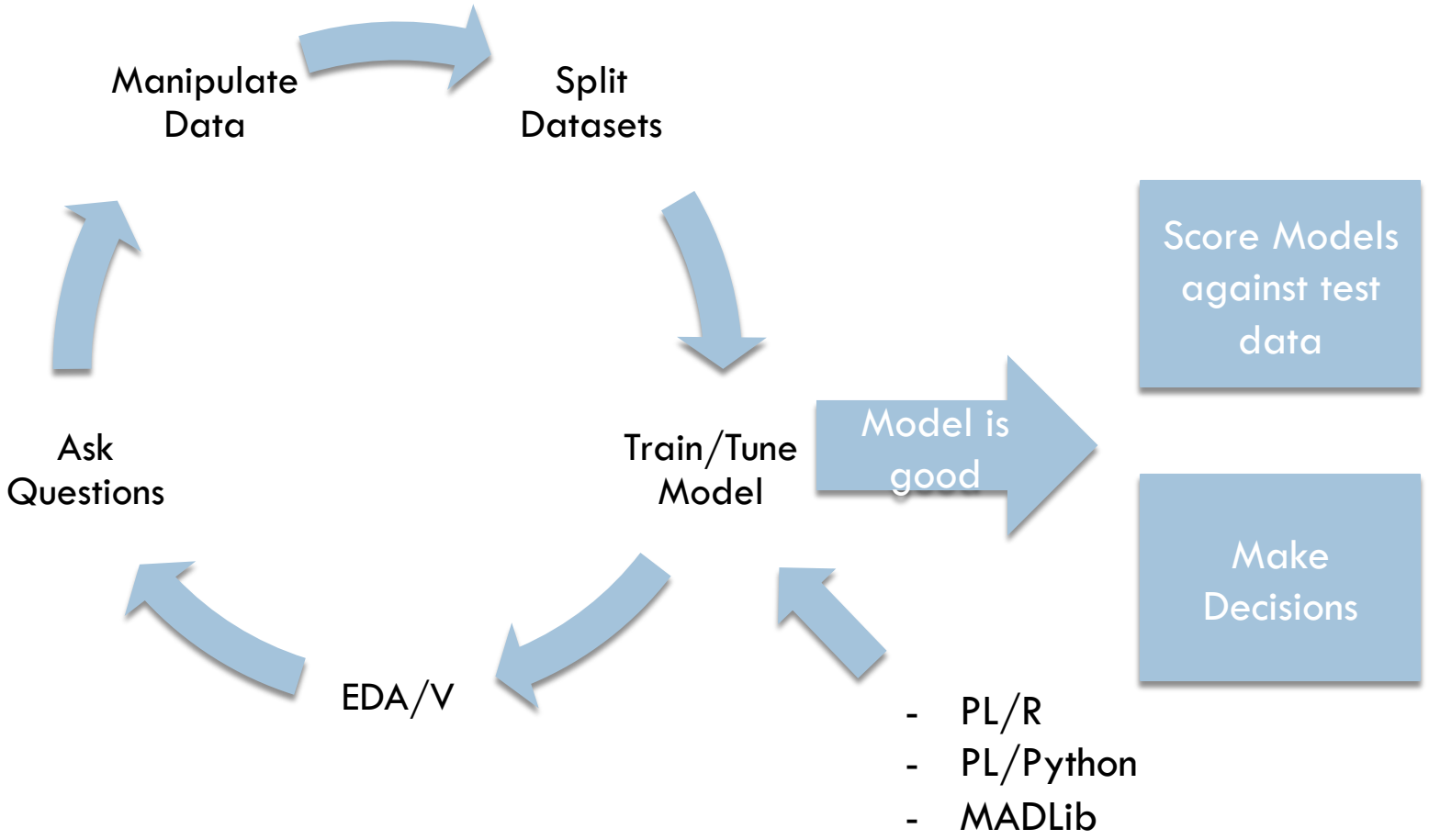
Shaping the Dataset

- Remember the initial dataset:
  - select ssn, gender, trans_date, category, round(amt, 2) as amount from mart.trans_fact limit 5;

```
    ssn   | gender | trans_date |    category    | amount
------------+--------+------------+----------------+-------
 587-27-8957 | M      | 2013-12-21 | health_fitness | 10.59
 587-27-8957 | M      | 2014-10-19 | shopping_pos   |  0.80
 587-27-8957 | M      | 2013-05-04 | entertainment  |  5.17
 587-27-8957 | M      | 2014-03-27 | utilities      |  9.59
 587-27-8957 | M      | 2013-03-30 | home           | 10.64
```

- How can we shape the data?
  - Transpose using case statements
  - Aggregate and sum
  - The query…

Building Models

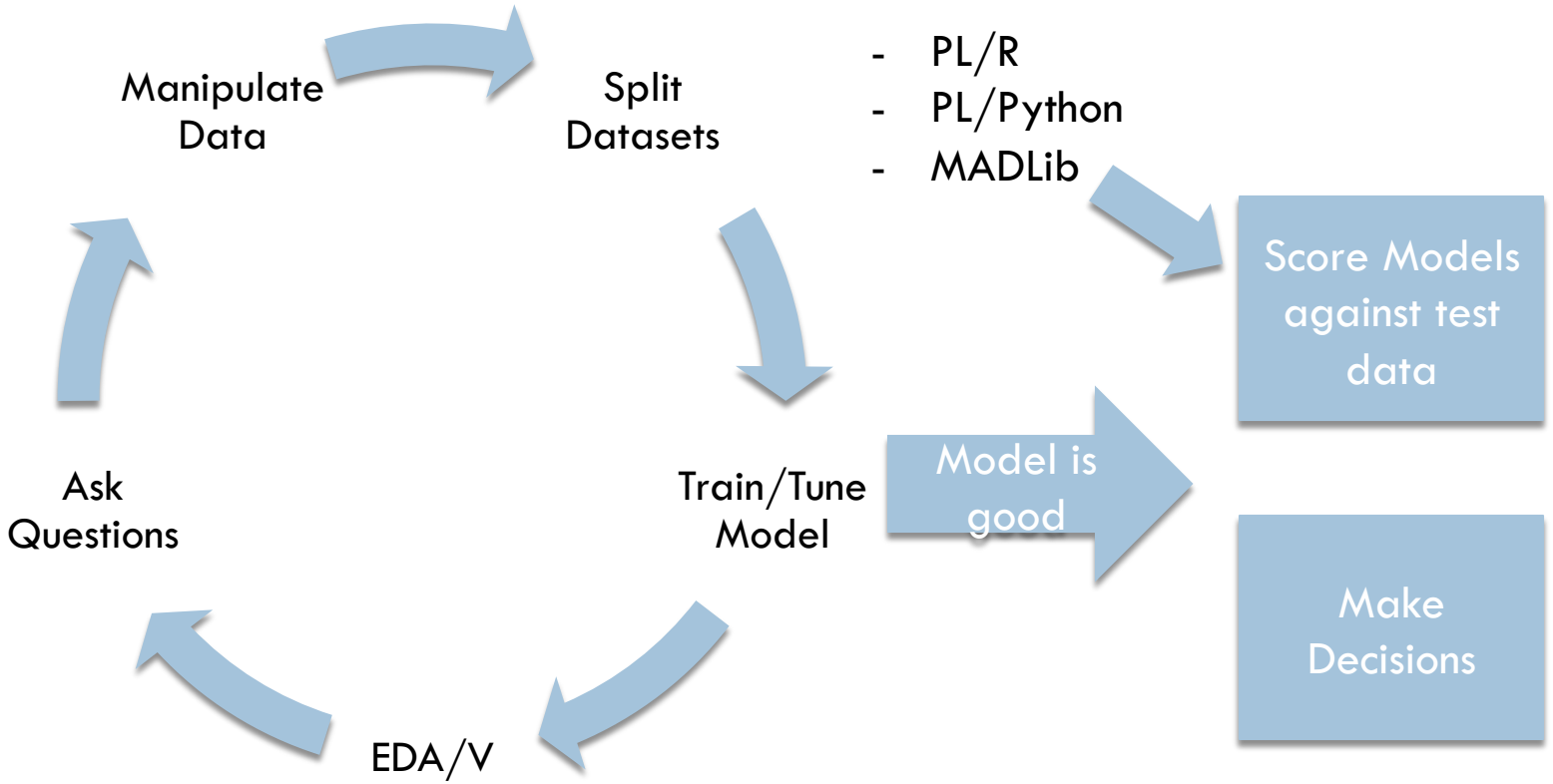


Pivotal

Learn Age Using Category Spending

```
DROP TABLE IF EXISTS elastic_output;
SELECT madlib.elastic_net_train( 'training_set',
                                 'elastic_output',
                                 'age',
                                 'array[food_dining, utilities, grocery_net, home,
pharmacy, shopping_pos, kids_pets, personal_care, misc_pos, gas_transport,
misc_net, health_fitness, shopping_net, travel]',
                                <other parameters>);
```

Scoring Chosen Model



Pivotal

Predictions on Test Set

```
SELECT ssn, predict, age - predict AS residual
FROM (
    SELECT
        test_set.*,
        madlib.elastic_net_gaussian_predict(
            m.coef_all,
            m.intercept,
            ARRAY[food_dining, utilities, grocery_net, home, pharmacy,
shopping_pos, kids_pets, personal_care, misc_pos, gas_transport, misc_net,
health_fitness, shopping_net, travel]
            ) AS predict
    FROM test_set, elastic_output m) s
ORDER BY ssn;
```

Predictions on Test Set

```
SELECT ssn, predict, age - predict AS residual
FROM (
    SELECT
        test_set.*,
        madlib.elastic_net_gaussian_predict(
            m.coef_all,
            m.intercept,
            ARRAY[food_dining, utilities, grocery_net, home, pharmacy,
shopping_pos, kids_pets, personal_care, misc_pos, gas_transport, misc_net,
health_fitness, shopping_net, travel]
            ) AS predict
    FROM test_set, elastic_output m) s
ORDER BY ssn;
```

Score Results

```
SELECT avg(residual^2)
FROM (<previous slide>) RESIDUALS;
```

PL/Python

- PL/Python (or PL/R) for data manipulation and training models
- Less intuitive than MADLib

Shaping the Dataset - UDFs

```
DROP TYPE IF EXISTS cust_address CASCADE;
CREATE TYPE cust_address AS
(
        ssn text,
        address text
);
CREATE OR REPLACE FUNCTION
        mart.test(ssn text, street text, city text, state text, zip
numeric)
RETURNS cust_address
AS $$
        return [ssn, street + ', ' + city + ', ' + state + ' ' + str(zip)]
$$ LANGUAGE plpythonu;

---------------------------
SELECT mart.test(ssn, street, city, state, zip)
from mart.trans_fact limit 5;
```

Shaping the Dataset - UDFs

```
CREATE TYPE cust_address AS
(
        ssn te  ,
        addr ss text
);
CREATE OR REPLACE FUNCTION
        mart.test(ssn text, street text, city text, state text, zip
numeric)
RETURNS cust_address
AS $$
        eturn [ssn, street + ', ' + city + ', ' + state + ' ' + str(zip)]
$$ LANGUAGE plpythonu;

--------------------------
SELECT mart.test(ssn, street, city, state, zip)
from mart.trans_fact limit 5;
```

Shaping the Dataset - UDFs

```
DROP TYPE IF EXISTS cust_address CASCADE;
CREATE TYPE cust_address AS
(
        ssn text,
        address text
);
CREATE OR REPLACE FUNCTION
        mart.test(ssn text, street text, city text, state text, zip
numeric)
RETURNS cust_address
AS $$
        return [ssn, street + ', ' + city + ', ' + state + ' ' + str(zip)]
$$ LANGUAGE plpythonu;

---------------------------
SELECT mart.test(ssn, street, city, state, zip)
from mart.trans_fact limit 5;
```

Shaping the Dataset - UDFs

```
DROP TYPE IF EXISTS cust_address CASCADE;
CREATE TYPE cust_address AS
(
        ssn text,
        address text
);
CREATE OR REPLACE FUNCTION
        mart.test(ssn text, street text, city text, state text, zip
numeric)
RETURNS cust_address
AS $$
        return [ssn, street + ', ' + city + ', ' + state + ' ' + str(zip)]
$$ LANGUAGE plpythonu;

--------------------------
SELECT mart.test(ssn, street, city, state, zip)
from mart.trans_fact limit 5;
```

Shaping the Dataset - UDFs

```
DROP TYPE IF EXISTS cust_address CASCADE;
CREATE TYPE cust_address AS
(
        ssn text,
        address text
);
CREATE OR REPLACE FUNCTION
        mart.test(ssn text, street text, city text, state text, zip
numeric)
RETURNS cust_address
AS $$
        return [ssn, street + ', ' + city + ', ' + state + ' ' + str(zip)]
$$ LANGUAGE plpythonu;

---------------------------
SELECT mart.test(ssn, street, city, state, zip)
from mart.trans_fact limit 5;
```

Shaping the Dataset - UDFs

```
INSERT INTO addresses
SELECT distinct (mart.test(ssn, street, city, state, zip)).ssn,
                (mart.test(ssn, street, city, state, zip)).address
FROM mart.trans_fact
LIMIT 5;

SELECT * FROM addresses;
```