MADlib

An open source library for in-database analytics

Hitoshi Harada

PGCon 2012, May 17th

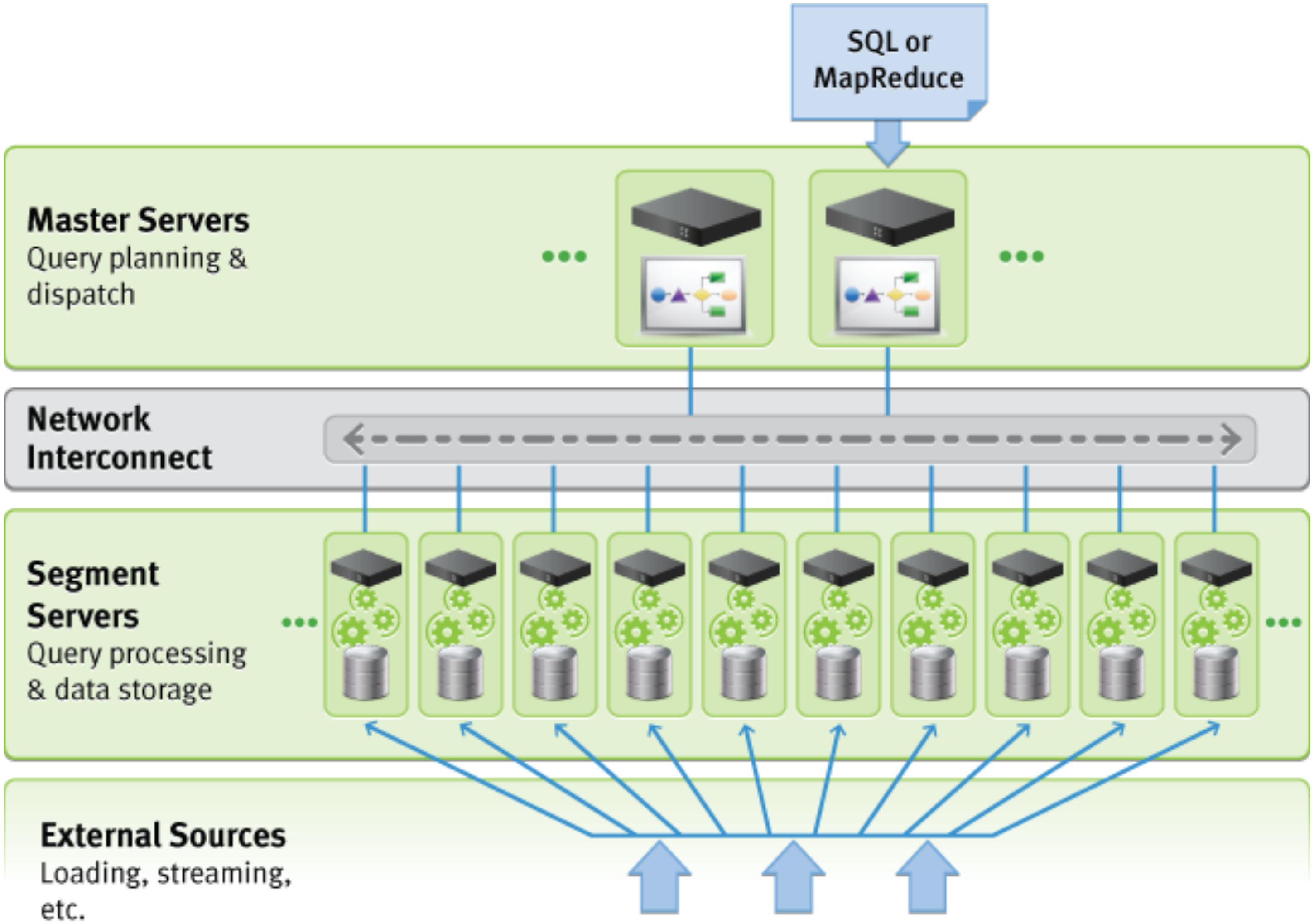
GREENPLUM

EMC²

Myself

- Window functions in 8.4 and 9.0
- Help wCTE work in 9.1
- PL/v8
- Other modules like twitter_fdw, tinyint
- Working at Greenplum

Greenplum Database: Massively Parallel Processing Database



"**Big Data** Is Less About Size, And More About Freedom"

—Techcrunch

"Findings: 'Big Data' Is More Extreme Than Volume"

— Gartner

"Big Data! It's Real, It's Real-time, and It's Already Changing Your World"

—IDC

"Total data: 'bigger' than big data"

— 451 Group

!!!

GREENPLUM

EMC²

THE ERA OF BIG DATA IS HERE

GREENPLUM®

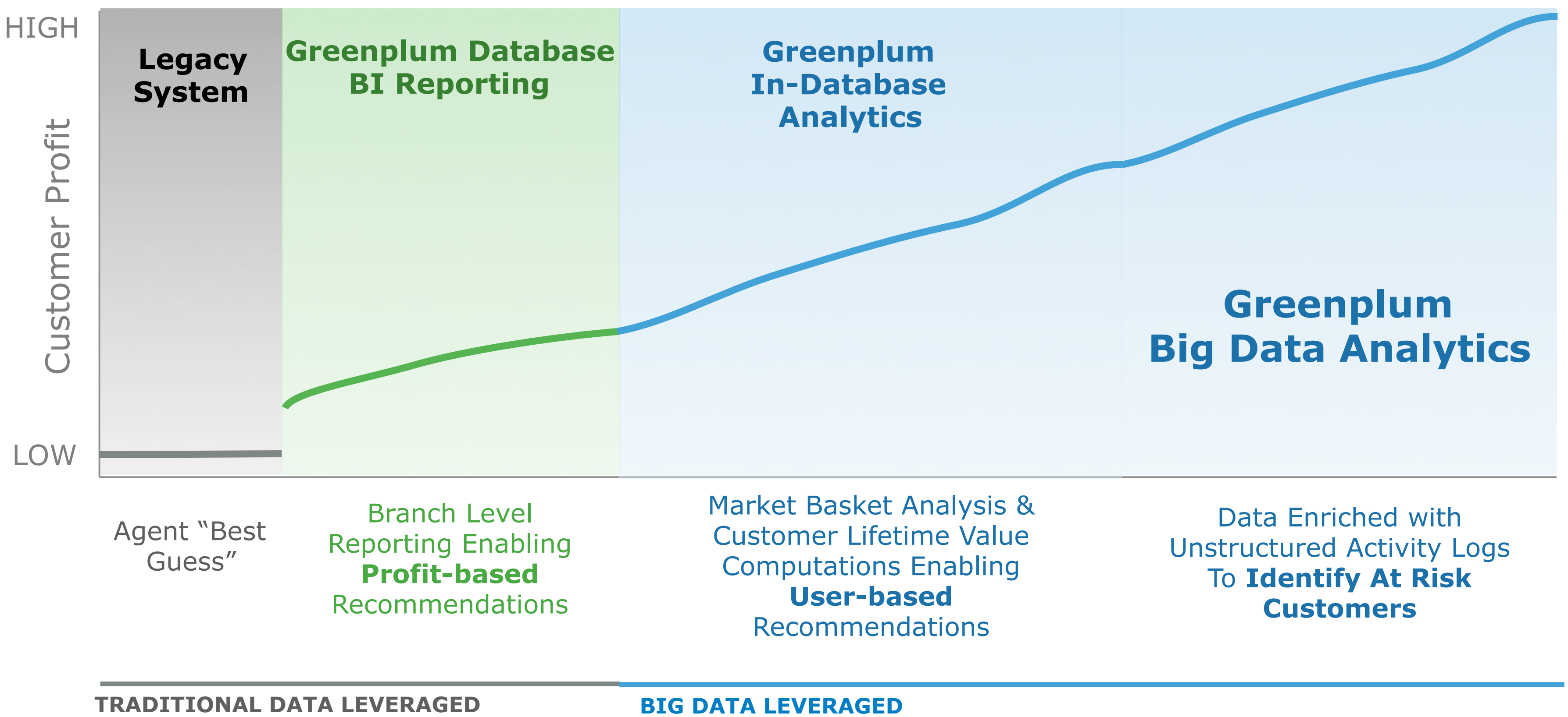
EMC* makes no representation and undertakes no obligations with regard to product planning information, anticipated product characteristics, performance specifications, or anticipated release dates (collectively, “Roadmap Information”). Roadmap Information is provided by EMC as an accommodation to the recipient solely for purposes of discussion and without intending to be bound thereby.

EMC²

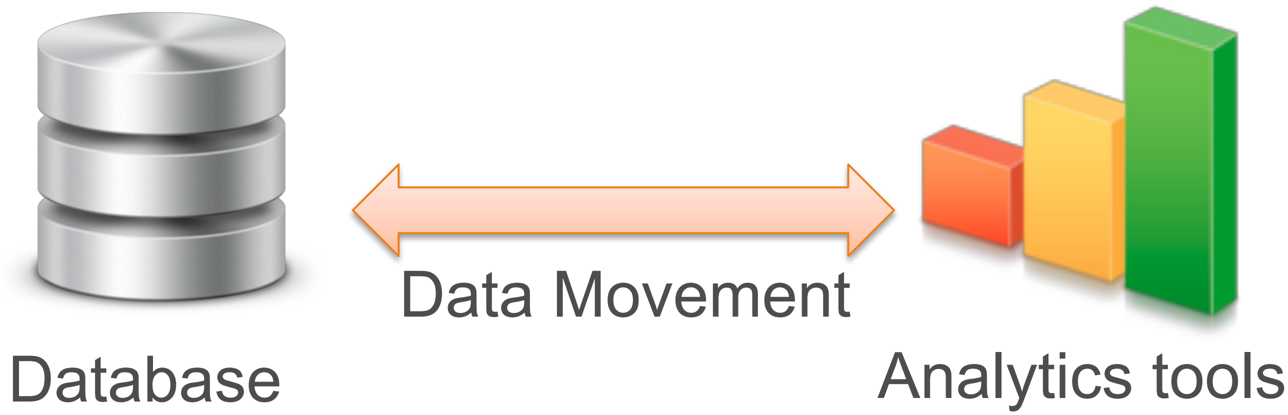
USE CASE

Predict Buyer Behavior to Increase Revenue

Big Data Analytics Enables Increased Per-Customer-Profit

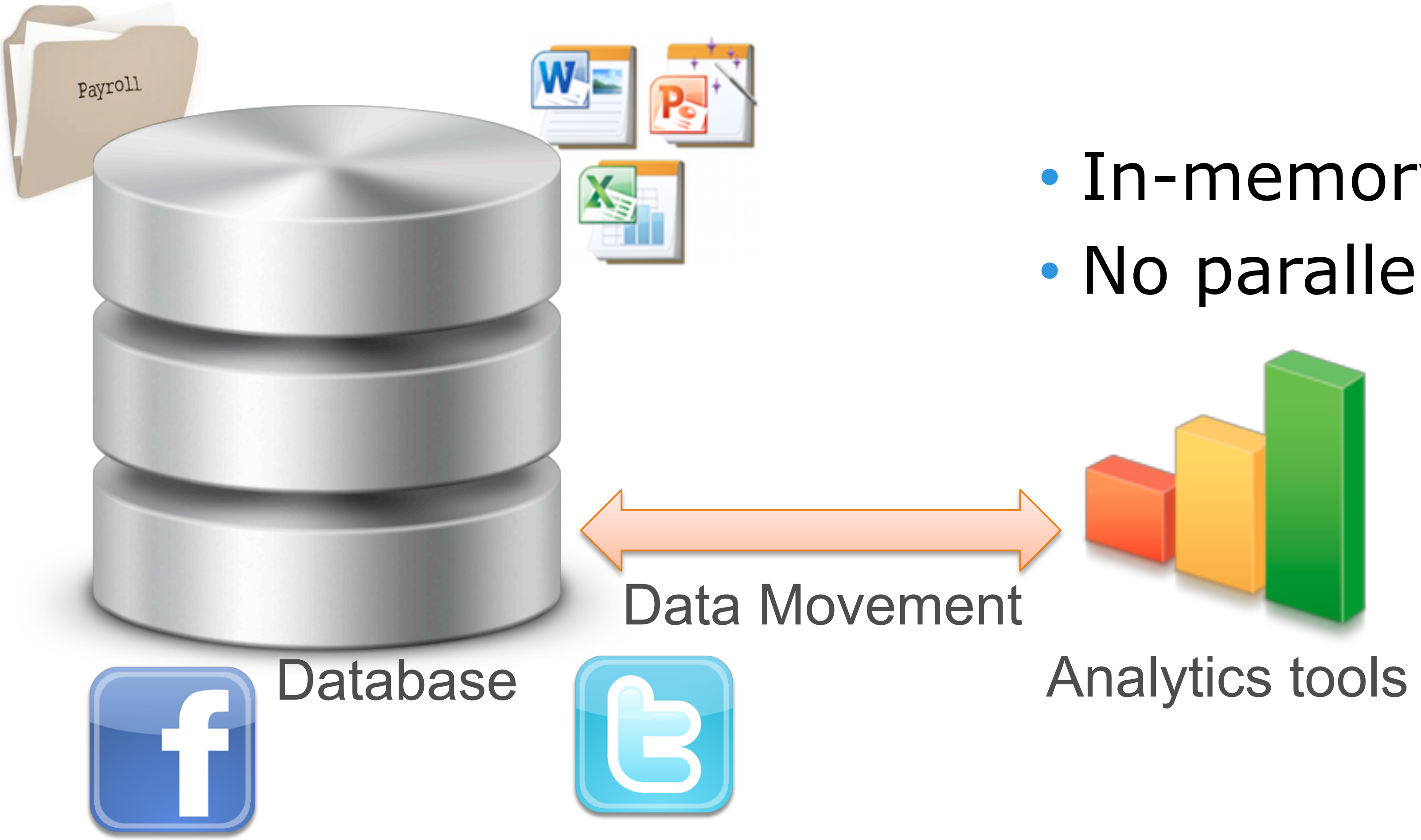


Traditional BI/Analytics



Big Data Arrives

- In-memory
- No parallelism



Analytics into Database

Analytics into Database

- Magnetic
  - Structured/Unstructured
- Agile
  - More Iterations
- Deep
  - More Accurate Methods

MADlib Introduction

MADlib: Introduction

- http://db.cs.berkeley.edu/papers/vldb09-madskills.pdf
  - MAD Skills: New Analysis Practices for Big Data
  - Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, Caleb Welton

- http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-38.pdf
  - The MADlib Analytics Library or MAD Skills, the SQL
  - Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Zhe Daisy Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, Arun Kumar

MADlib: Definition

- **MAD** stands for: 

- **lib** stands for **library** of:
  - advanced (mathematical, statistical, machine learning)
  - parallel & scalable
  - in-database functions

- **Mission:** to foster widespread development of scalable analytical skills, by harnessing efforts from commercial practice, academic research, and open-source development.

GREENPLUM®

EMC²

MADlib: A Community Project

Open Source: BSD License

- Developed as a partnership with multiple universities
  - University of California-Berkeley
  - University of Wisconsin-Madison
  - University of Florida
- Compatibility with Postgres and Greenplum Database.
- Designed for Data Scientists to provide Scalable, Robust Analytics capabilities for their business problems.
- Homepage: http://madlib.net
- Source: https://github.com/madlib
- Forum: http://groups.google.com/group/madlib-user-forum

MADlib: Sane Answer to Big Data

- Better Performance and Scalability
  - Run inside your database
  - Leverage parallelism
- Easy to Use
  - No additional tools. SQL is your friend.
- Open Source
  - Hackable

MADlib: Sane Answer to Big Data

- Better Performance and Scalability
  - Run inside y
  - Leverage pa
- Easy to Use
  - No additiona ur friend.
- Open Sourc
  - Hackable

Current | Targeted

Greenplum MADlib Roadmap

Q3 '11 | Q1 '12 | Q2 '12

Software

MADlib v 0.2

Supervised Learning
Linear Regression
Logistic Regression
Support Vector Machines
Decision Trees
Naïve Bayes Classification
Unsupervised Learning
Association Rule Mining
K-Means Clustering
SVD Matrix Factorization
Latent Dirchlet Allocation
Descriptive Statistics
Profile
Quantile

MADlib v 0.3

Better Scalability
Better Performance
Better Predictive Accuracy

Linear Regression

Logistic Regression
Gradient descent Algorithm

K-Means Clustering
New Distance metrics
New Seeding options

Decision Trees
Tree scoring
Tree pruning
Tree output display
New Split Criteria

MADlib v 0.4

Statistical Relevance Tests
T-tests, F-tests

Distribution Functions
T-distribution, F-distribution

Ensemble Modeling
Random Forest

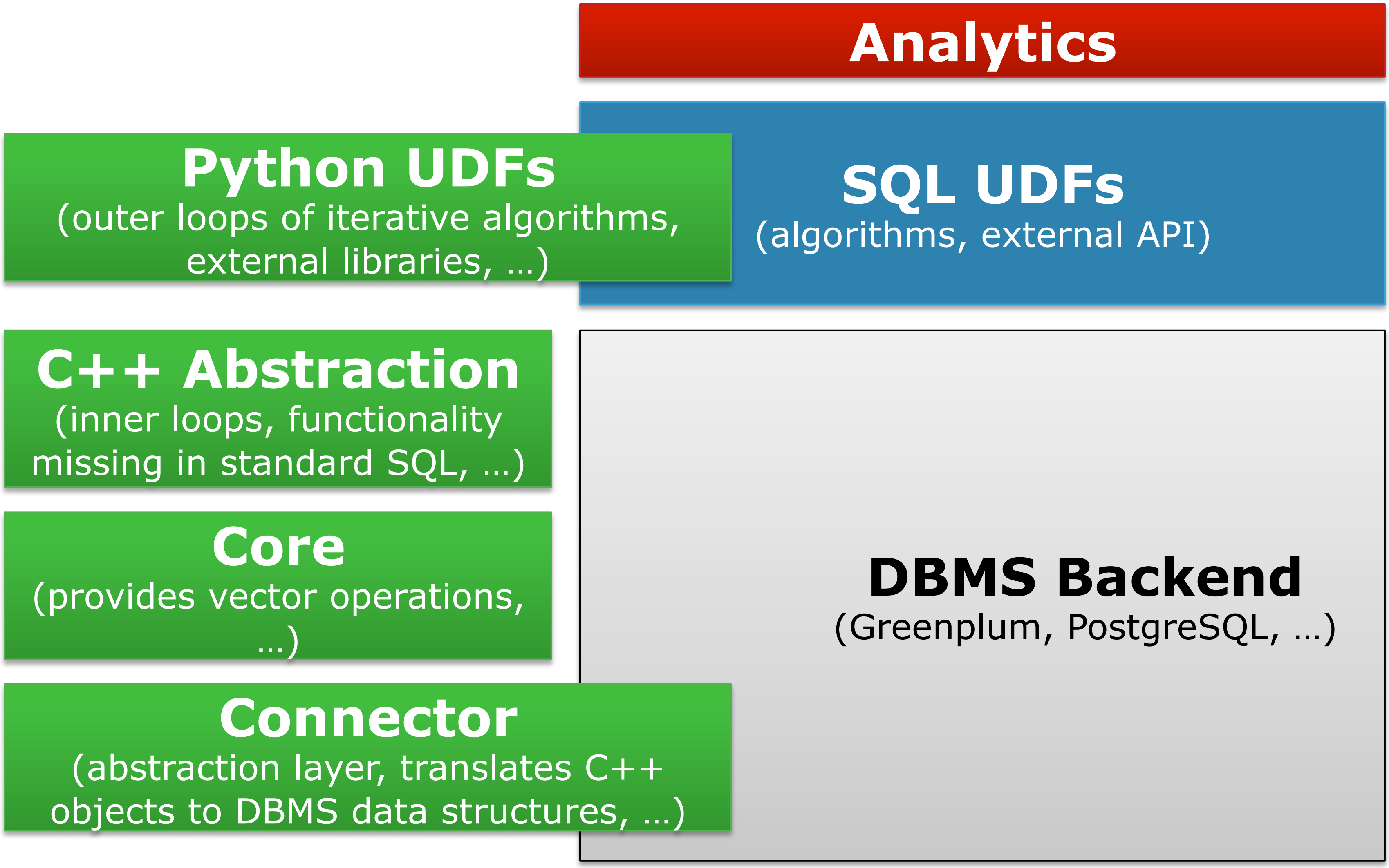
Text Processing
Viterbi Feature Extraction

GREENPLUM

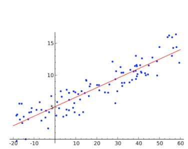
EMC²

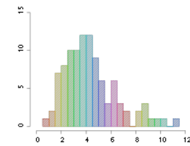
MADlib Architecture

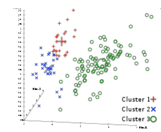
Architecture

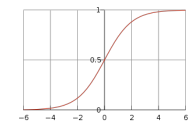


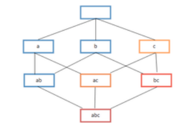
MADlib: Contents











Data Modeling

Supervised Learning

- Naive Bayes Classification
- Linear Regression
- Logistic Regression
- Decision Tree
- Support Vector Machines

Unsupervised Learning

- Association Rules
- k-Means Clustering
- SVD Matrix Factorization
- Parallel Latent Dirichlet Allocation

Descriptive Statistics

Sketch-based Estimators

- CountMin (Cormode-Muthukrishnan)
- FM (Flajolet-Martin)
- MFV (Most Frequent Values)

Profile

Quantile

Support Modules

Array Operations

Conjugate Gradient

Sparse Vectors

MADlib 0.3

User Documentation

Main Page | Modules | Files | Search

k-Means Clustering

Unsupervised Learning

Collaboration diagram for k-Means Clustering:

About:

Clustering refers to the problem of partitioning a set of objects according to some problem-dependent measure of *similarity*. In the k-means variant, one is given n points $x_1, \dots, x_n \in \mathbf{R}^d$, and the goal is to position k centroids $c_1, \dots, c_k \in \mathbf{R}^d$ so that the sum of squared distances between each point and its closest centroid is minimized. (A cluster is identified by its centroid and consists of all points for which this centroid is closest.) Formally, we wish to minimize the following objective function:

$$(c_1, \dots, c_k) \mapsto \sum_{i=1}^{n} \min_{j=1}^{k} \operatorname{dist}(x_i, c_j)^2$$

This problem is computationally difficult (NP-hard), yet the local-search heuristic proposed by Lloyd [4] performs reasonably well in practice. In fact, it is so ubiquitous today that it is often referred to as the *standard algorithm* or even just the *k-means algorithm* [1]. It works as follows:

1. Seed the k centroids (see below)
2. Repeat until convergence:
   a. Assign each point to its closest centroid
   b. Move each centroid to the barycenter (mean) of all points currently assigned to it
3. Convergence is achieved when no points change their assignments during step 2a.

MADlib Use Cases

Supervised vs. Unsupervised Learning

Machine learning

- Unsupervised is a learning from raw data (no labels)

  Example: A consumer market segmentation study
  Methods: K-means Clustering

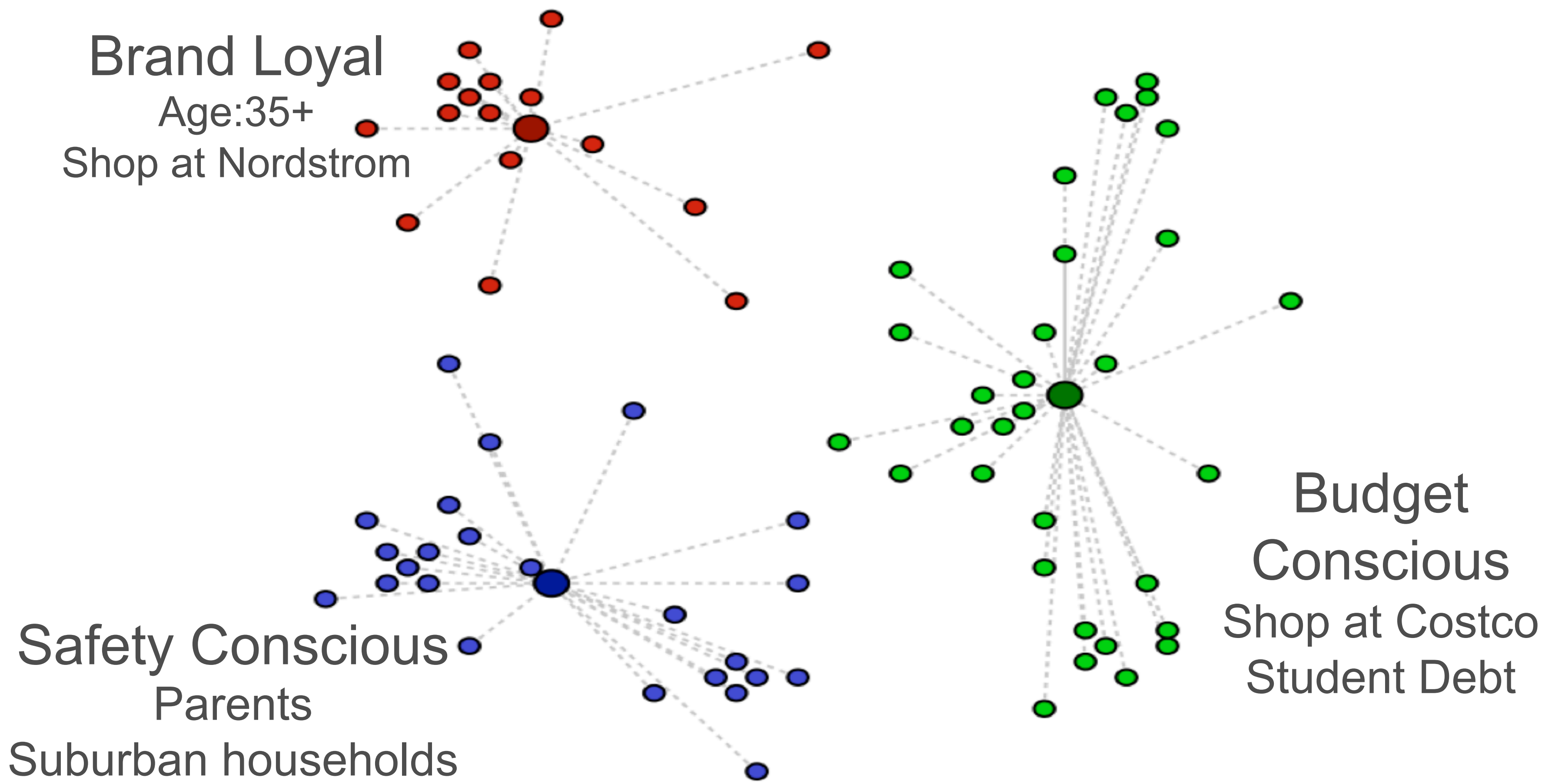
- Supervised is a learning from data where data is classified into different categories (data has labels)

  Example: Classify email as spam and non-spam
  Methods: Logistic Regression

[Unsupervised Learning]

Market Segmentation

Customer Segmentation Study



K-Means Clustering

Preparing the Data

- Vectorize the input attributes (into float8[]):

```
CREATE TABLE input_points AS
   SELECT row_id, array[x,y]::float8[] as points
   FROM source_table;

SELECT * FROM input_points LIMIT 5;
 row_id | array
--------+--------
      2 | {2,-1}
      4 | {2,1}
      6 | {2,2}
      8 | {2,2}
     10 | {3,-5}
```

K-Means Clustering

Centroid Initialization

- MADlib supports several different ways of initializing the centroids to use for clustering:
  - Kmeans_random(…) – Random
    - Chooses some random points from the input
    - May take longer to converge on a solution
  - kmeans_plusplus(…) – Kmeans++
    - Chooses some random points that are "distant" from each other.
  - Kmeans_cset(…) – Centroid Set
    - The user supplies the initial set of points
    - Centroids must be stored in a separate relation

K-Means Clustering

Distance Metrics

- MADlib supports several different ways of measuring "distance" between points:
  - L1norm
  - L2norm
    - aka the Euclidian distance
    - Good for spatial data, or data with natural geometric distances
  - Cosine
    - measure of the angle between two vectors
    - Often used for sparse high dimensional spaces, including text
  - Tanimoto
    - Generally used to compare similarity and diversity of sample sets.

K-Means Clustering

Invoking k-means

```
SELECT *
FROM madlib.kmeans_plusplus(
        ’input_points',     -- name of the table of input data
        ’points',           -- name of the column containing the feature vector
        ‘row_id’,           -- name of the id column, or NULL if no such column
        'km_p',             -- output table name: points
        'km_c',             -- output table name: centroids
        'l2norm',           -- distance metric to use
        10,                 -- maximum number of iterations
        0.001,              -- convergence threshold
        False,              -- evaluation goodness of fit?
        False,              -- verbose output?
        10,                 -- k: number of clusters
        0.01);              -- sample fraction to use for generating
                                initial centroids
```

K-Means

Making Sense of the Results

- K-means will produce two output tables:
  - Output points

```
sql> select * from km_p;
 pid |      coords       | cid
-----+-----------------+-----
   1 | {1,1}:{2,-1}     |   1
   3 | {1,1}:{2,-1}     |   1
   5 | {1,1}:{2,1}      |   1
   7 | {2}:{2}          |   1
   9 | {1,1}:{2,10}     |   2
```

  - Output centroids

```
select * from km_c;
 cid |          coords
-----+---------------------------
   1 | {1,1}:{2.1111111111111,0}
   2 | {1,1}:{2,10}
```

[Supervised Learning]

Heart Attack Risk

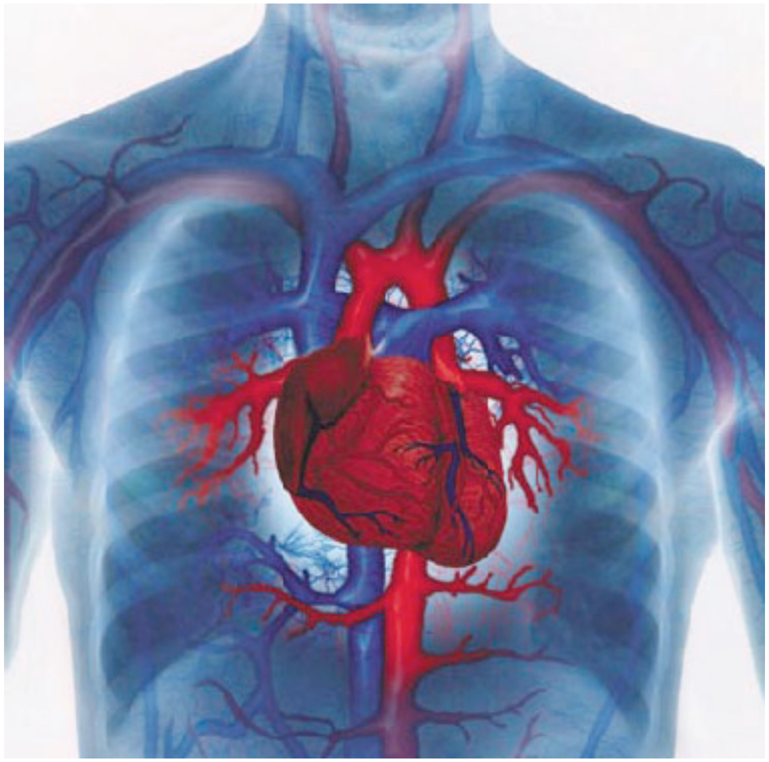
Classification Analysis

- Classification: identify which category a new observation belongs to with known observations.
- This generally involves:
  - Training: which builds the model based on labeled data
  - Classification: which labels new data based on the model.
- Examples:
  - Logistic Regression
  - Decision Trees
  - Naïve Bayes

Heart Attack prediction using Logistic Regression

- Calculate the potential risk of heart attack based on the historical data with a number of attributes.

- What affects?
  - Age
  - Cholesterol
  - Height
  - Weight
  - etc.?



Logistic Regression

Preparing the Data

- Prepare the labeled (training) data.

```
CREATE TABLE coronary(
    age               integer,
    blood_pressure    float8,
    cholesterol       float8,
    height            float8,
    weight            float8,
    heart_attack      boolean
);
```

- Transform into an array.

```
CREATE TABLE coronary_prepared AS
   SELECT heart_attack,
              array[1, age, blood_pressure,
                  cholesterol, height, weight] as features
   FROM coronary;
```

Logistic Regression

Training the Model

- Build the model.

```
CREATE TABLE coronary_model AS
   SELECT * FROM madlib.logregr(
      ‘coronary_prepared’,-- Input table name
      ‘heart_attack’,    -- name of the label column
      ‘features’   – name of the feature vector column
   );
```

Logistic Regression

Training the Model

- Examine the model.

```
    SELECT unnest(array['intercept', 'age',
'blood_pressure', 'cholesterol',
                    'height', 'weight'])   as feature_name
           unnest(coef) as coefficient,
           unnest(std_err) as std_err
    FROM coronary_model;
```

feature_name	coefficient	std_err
intercept	-0.05	2.97761374227056e+63
age	-9.15	1.48880687113528e+65
blood_pressure	18.125	4.76418198763289e+65
cholesterol	-13.05	7.77157186732615e+65
height	-3.3	1.96522506989857e+65
weight	10.325	4.31753992629231e+65

Logistic Regression

Classifying new data

- To predict outcomes for new data based on a trained logistic model you must:
  - Calculate the dot product of the new feature vectors vs the calculated model coefficients.
  - Call the logistic function over the dot product

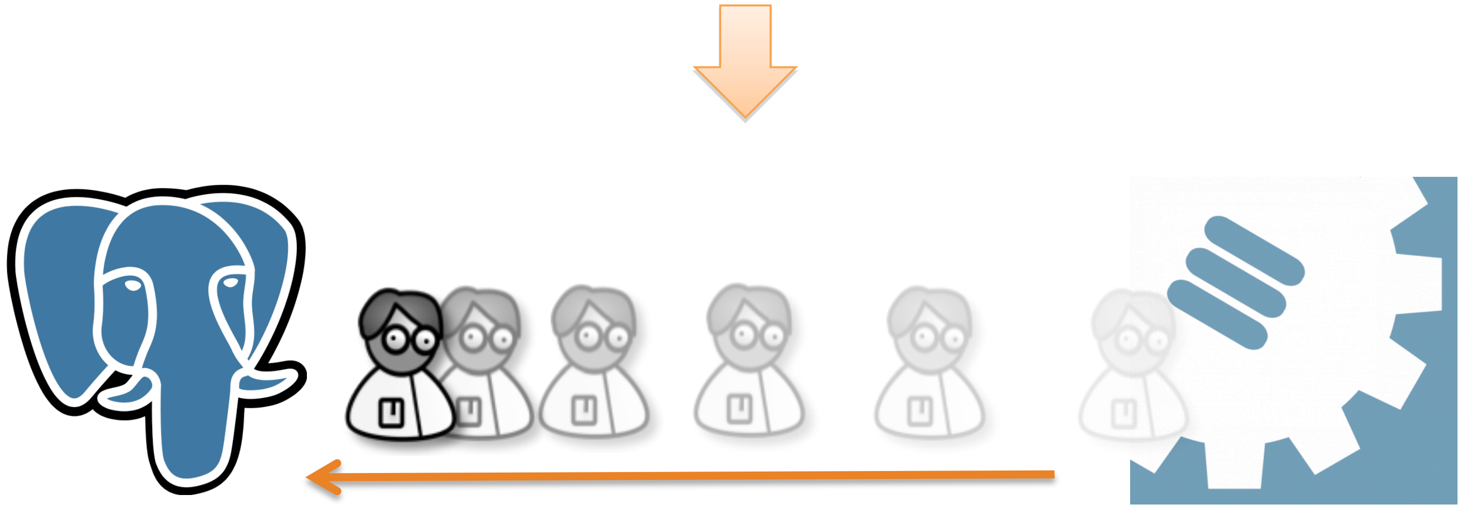
```
SELECT new_data.*,
       madlib.logistic(madlib.arr_dot(features,coef))
  FROM new_data, coronary_model;

 id |        features        |       logistic
----+-----------------------+----------------------
  4 | {1,34,140,230,44,88} | 7.88926258624503e-06
…
```

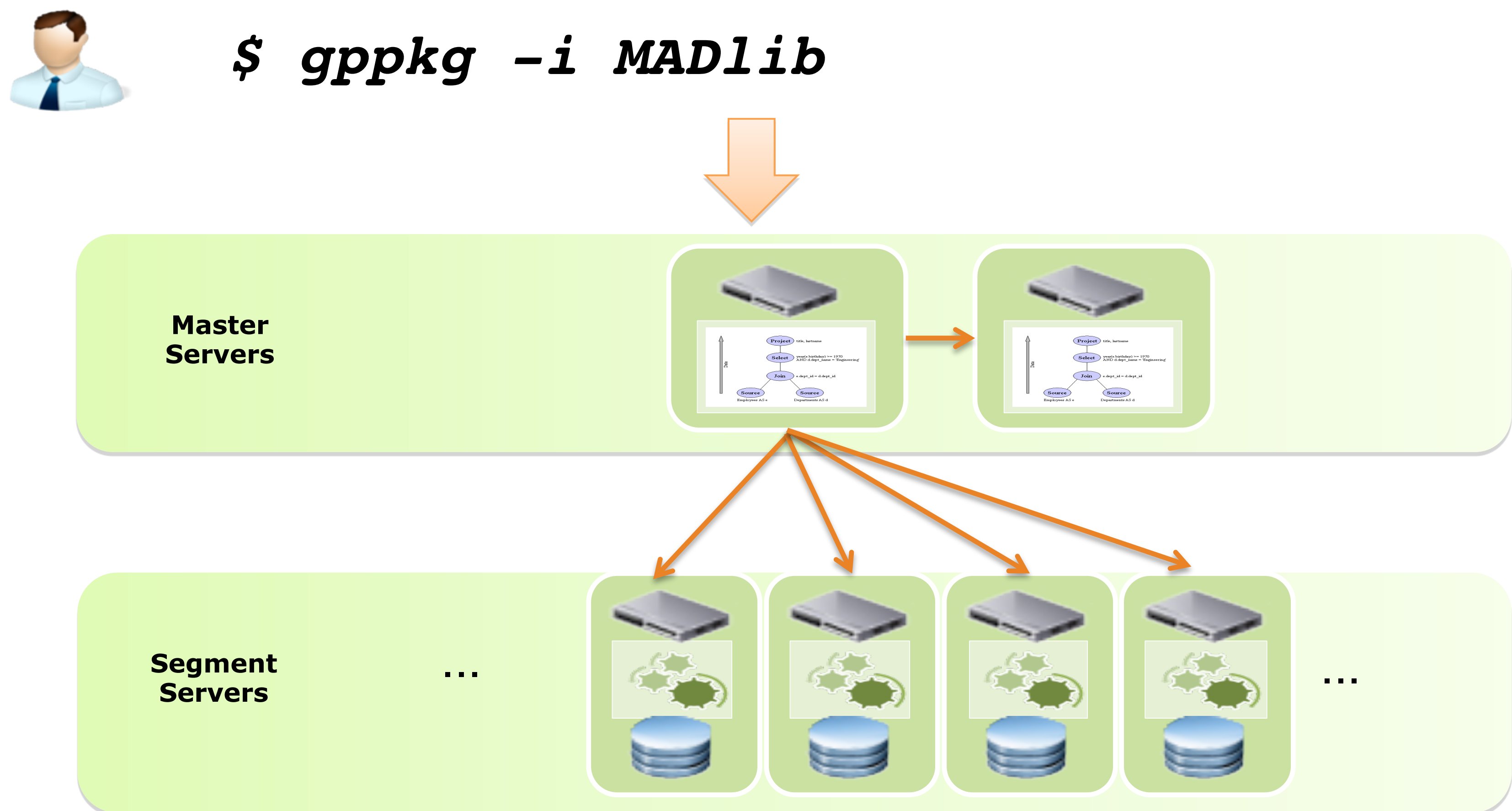
Deploying MADlib

Automatic Install of Analytic Extensions

```
$ pgxn install madlib
```



Automatic Install of Analytic Extensions



I WANT YOU
TO OPEN SOURCE!
WWW.MIL-OSS.ORG

Thank You!