# Greenplum:
# MAD Analytics in Practice

MMDS

June 16th, 2010

# Warehousing Today

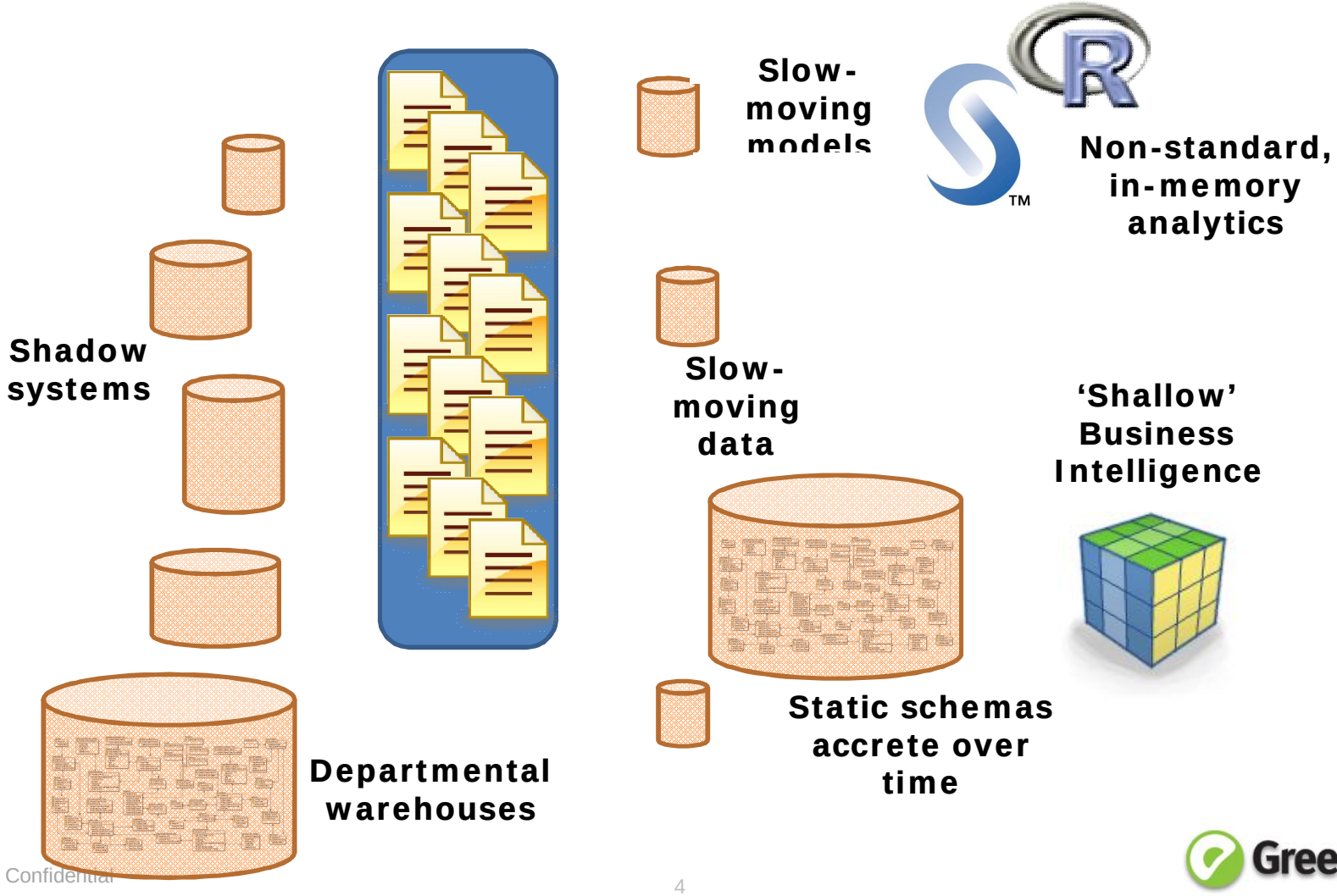
# In the Days of Kings and Priests

- Computers and Data: Crown Jewels
- Executives depend on computers
  - But cannot work with them directly
- The DBA "Priesthood"
  - And their Acronymia
    - EDW, BI, OLAP, 3NF
  - Secret functions and techniques, expensive tools



Greenplum

# The Architected EDW



# New Realities

"Welcome to the Petabyte Age"

- TB disks < $100
- Everything is data
- Rise of data-driven culture
  - Very publicly espoused by Google, Netflix, etc.
  - Terraserver, USAspending.gov

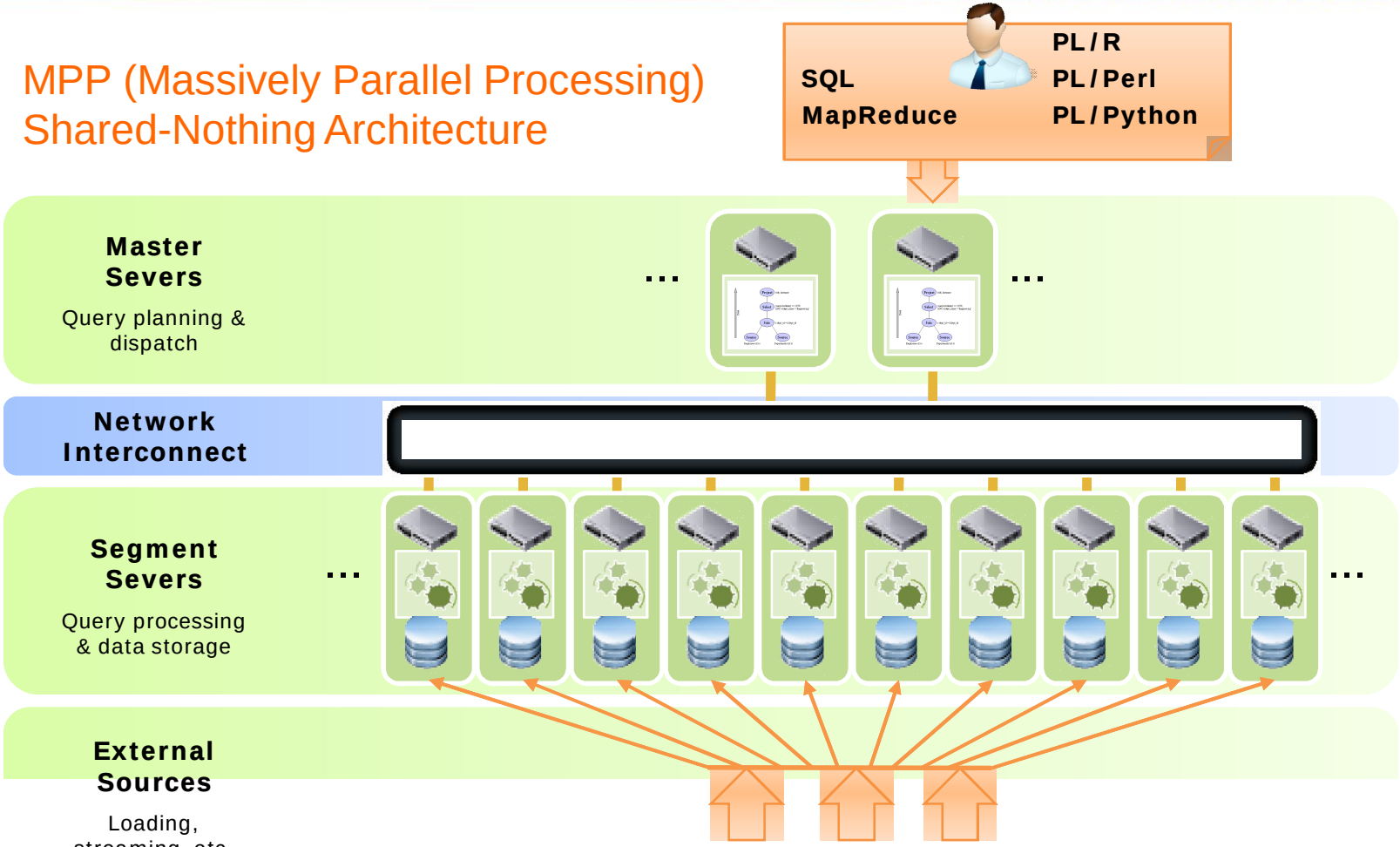

## The New Practitioners

- Aggressively Datavorous
- Statistically savvy
- Diverse in training, tools

# Greenplum Overview

# Greenplum Database Architecture

MPP (Massively Parallel Processing)
Shared-Nothing Architecture



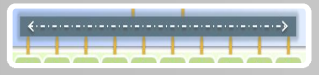
# Key Technical Innovations

## Scatter-Gather Data Streaming

- Industry leading data loading capabilities

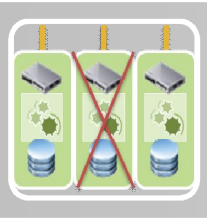
## Online Expansion

- Dynamically provision new servers with no downtime



## Map-Reduce Support

- Parallel programming on data for advanced analytics



## Polymorphic Storage

- Support for both row and column-oriented storage

# Benefits of the Greenplum Database Architecture

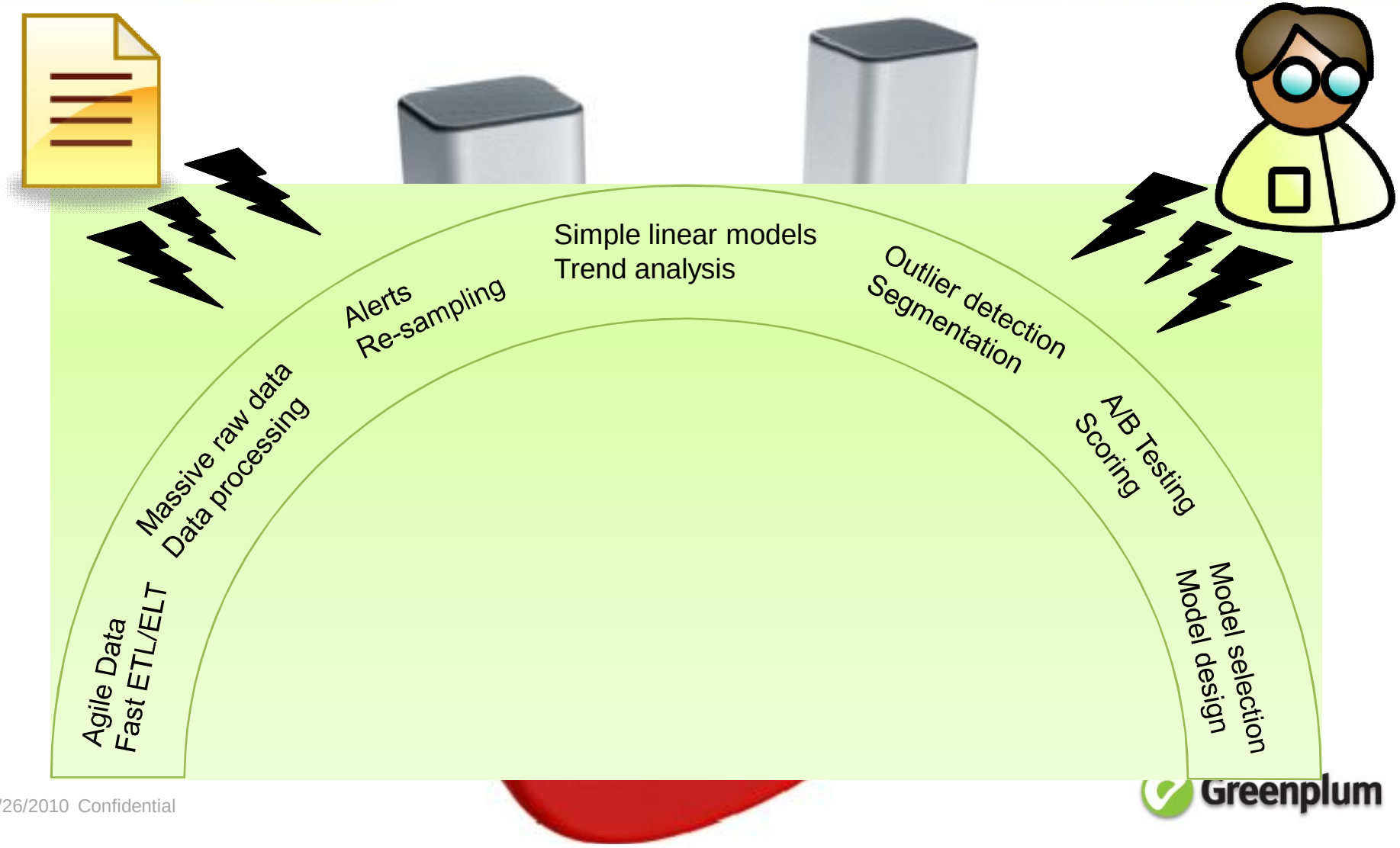
- **Simplicity**
  - Parallelism is automatic – no manual partitioning required
  - No complex tuning required – just load and query
- **Scalability**
  - Linear scalability to 1000s of cores, on commodity hardware
  - Each node adds storage, query performance, loading performance
  - Example: 6.5 petabytes on 96 nodes, with 17 trillion records
- **Flexibility**
  - Fully parallelism for SQL92, SQL99, SQL2003 OLAP, MapReduce
  - Any schema (star, snowflake, 3NF, hybrid, etc)
  - Rich extensibility and language support (Perl, Python, R, C, etc)

# Customer Example: eBay Petabyte-Scale

- **Business Problem**
  - Fraud detection and click-stream analytics
- **Data Size**
  - 6.5 Petabytes of user data
  - Loading 18 TB every day (130 Billion rows)
  - Every click on eBay's website, 365 days a year
  - 20 Trillion row fact table
- **Hardware**
  - 96-node Sun Data Warehouse Appliance
  - Expansion to go to 192 nodes
- **Benefit**
  - Scalability and price/performance
  - Cost effective complement to Teradata

# MAD Analytics

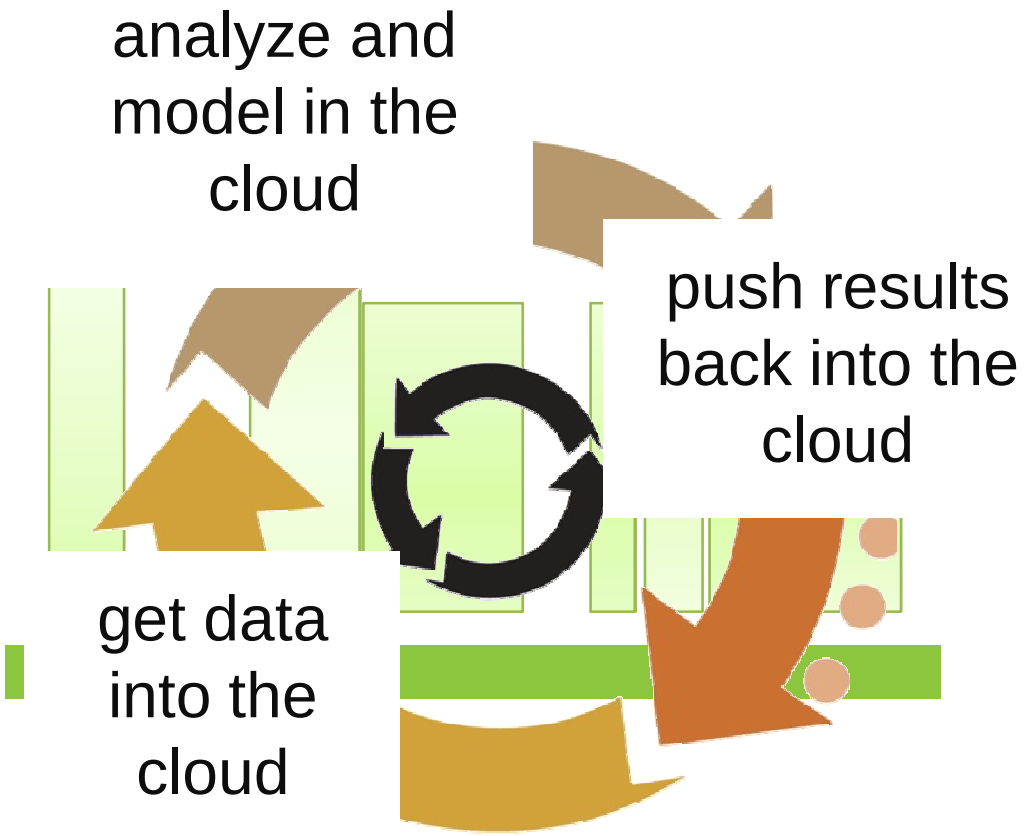
# Magnetic






Greenplum

# Agile


analyze and model in the cloud
push results back into the cloud
get data into the cloud

# Deep


| | Facts | Interpretation |
|---|---|---|
| Future | What will happen? | How can we do better? |
| Past | What happened where and when? | How and why did it happen? |

# Dolan's Vocabulary of Statistics

- Data Mining focused on individuals
  - Statistical analysis needs more
  - Focus on *density*methods
- Need to be able to utter statistical sentences
  - And run massively parallel, on Big Data!

1. (Scalar) Arithmetic
2. Vector Arithmetic
   - I.e. Linear Algebra
- Functions
  - E.g. probability *densities*
- Functionals
  - i.e. functions on functions
  - E.g., A/B testing: a *functional over densities*
- Misc Statistical methods
  - E.g. resampling

# MAD Skills Whitepaper

- Paper includes parallelizable, stat-like SQL for
  - Linear algebra (vectors/matrices)
  - Ordinary Least Squares (multiple linear regression)
  - Conjugate Gradiant (iterative optimization, e.g. for SVM classifiers)
  - Functionals including Mann-Whitney U test, Log-likelihood ratios
  - Resampling techniques, e.g. bootstrapping
- Encapsulated as stored procedures or UDFs
  - Significantly enhance the vocabulary of the DBMS!
- These are examples.
  - Related stuff in NIPS '06, using MapReduce syntax
- Plenty of research to do here!!

# MAD Analytics Examples

# Example

***What's the right price for my products?***

# What's the right price for my products?

| Date | BasePrice | Display Price | Feature Price | Feature/Display Price | TPR | Volume |
|---|---|---|---|---|---|---|
| 2009-02-24 | 7.33 | 6.67 | 7.33 | 7.33 | 7.20 | 20484.52 |
| 2009-03-10 | 7.47 | 5.94 | 5.72 | 7.00 | 5.72 | 34313.94 |
| 2009-03-24 | 7.75 | 6.74 | 5.74 | 7.26 | 5.82 | 25477.33 |
| 2009-04-07 | 7.40 | 7.19 | 7.40 | 7.40 | 7.23 | 18772.57 |
| 2009-04-21 | 7.75 | 7.36 | 6.74 | 7.75 | 6.22 | 20743.68 |
| 2009-05-05 | 7.43 | 6.56 | 6.98 | 7.43 | 5.70 | 28244.82 |
| 2009-05-19 | 7.70 | 6.57 | 7.70 | 7.70 | 6.23 | 20234.74 |
| 2009-06-02 | 6.87 | 6.67 | 6.87 | 6.87 | 6.64 | 23262.60 |
| 2009-06-16 | 7.36 | 7.00 | 7.36 | 7.36 | 7.44 | 19290.87 |
| 2009-06-30 | 6.92 | 6.72 | 6.92 | 6.92 | 6.73 | 23617.61 |
| 2009-07-14 | 7.49 | 7.32 | 7.49 | 7.49 | 7.58 | 18017.58 |
| 2009-07-28 | 7.69 | 7.44 | 5.69 | 7.69 | 5.70 | 29193.44 |
| 2009-08-11 | 7.19 | 6.24 | 7.19 | 7.19 | 6.72 | 23863.13 |
| 2009-08-25 | 7.72 | 6.74 | 7.72 | 7.72 | 5.72 | 25138.34 |

## Get the raw data…

```
DROP TABLE IF EXISTS misc.price_promo;

CREATE TABLE misc.price_promo
(
dt date
 ,base_price numeric
 ,display_price numeric
 ,feature_price numeric
 ,feature_display_price numeric
 ,tpr numeric
 ,volume numeric
) DISTRIBUTED BY(dt);


\copy misc.price_promo from data.csv with delimiter
      ','
```

Greenplum

# What's the right price for my products?

| intercept_beta | base_price_beta | display_price_beta | feature_display_price_beta | tpr_beta | r2 |
|---|---|---|---|---|---|
| 72804.48332 | 5049.03841 | -1388.842417 | -6203.882026 | -4801.114351 | 0.883172235 |

## Train the model…

```
CREATE TABLE misc.price_promo_coefs AS
SELECT
 coefs[1] AS intercept_beta                    ,coefs[2] AS base_price_beta
 ,coefs[3] AS display_price_beta               ,coefs[4] AS feature_display_price_beta
 ,coefs[5] AS tpr_beta                         ,r2
FROM (
 SELECT
mregr_coef(volume, array[1::int, base_price_per_unit, display_price,        feature_display_price,
    temporary_price_reduction]) AS coefs
  ,mregr_r2(volume, array[1::int, base_price_per_unit, display_price,  feature_display_price, temporary_price_reduction])
    AS r2
 FROM
misc.price_promo
 ) AS a
DISTRIBUTED RANDOMLY;
```

# What's the right price for my products?

| volume | volume_fitted | ape |
|---|---|---|
| 20484.52 | 20507.88 | 0.1140 |
| 34313.94 | 31381.52 | 8.5458 |
| 25477.33 | 29591.06 | 16.1466 |
| 18772.57 | 19560.80 | 4.1988 |
| 20743.68 | 23769.63 | 14.5873 |
| 28244.82 | 27746.83 | 1.7630 |
| 20234.74 | 24876.55 | 22.9398 |
| 23262.60 | 23727.72 | 1.9994 |
| 19290.87 | 18862.64 | 2.2198 |
| 23617.61 | 23168.44 | 1.9018 |
| 18017.58 | 17595.93 | 2.3402 |
| 29193.44 | 26224.39 | 10.1702 |
| 23863.13 | 23571.29 | 1.2229 |
| 25138.34 | 27065.91 | 7.6678 |
| 24307.88 | 23945.45 | 1.4909 |
| ... | ... | ... |

## Evaluate the model…

```
CREATE OR REPLACE VIEW misc.v_price_promo_fitted AS
SELECT
 volume
 ,volume_fitted
 ,100 * abs(volume - volume_fitted)::numeric / volume AS ape
FROM (
 SELECT
p.volume
  ,c.intercept_beta
   + p.base_price * c.base_price_beta
      + p.display_price * c.display_price_beta
      + p.feature_display_price * c.feature_display_price_beta
      + p.tpr * c.tpr_beta
     AS volume_fitted
 FROM
misc.price_promo_coefsc
  ,misc.price_promop
) AS a
```



Greenplum

# Example

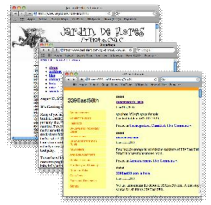
***What are our customers saying about us?***

# What are our customers saying about us?


- How do you discern trends and categories within thousands of on-line conversations?
  - Search for relevant blogs
  - Construct a ‘fingerprint’ for each document based on word frequencies
  - Use this to define what it means for documents to be similar, or ‘close’
  - Identify ‘clusters’ of documents

# Accessing the data

- Build the directory list into a set of files that we will access:



```
-INPUT:
    NAME: filelist
    FILE:
      - maple:/Users/demo/blogsplog/filelist1
      - maple:/Users/demo/blogsplog/filelist2
    COLUMNS:
      - path text
```

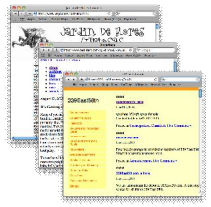
- For each record in the list "open()" the file and read it in its entirety

```
-MAP:
    NAME:      read_data
    PARAMETERS: [path text]
    RETURNS:    [id int, path text, body text]
    LANGUAGE:   python
    FUNCTION: |
      (_, fname) = path.rsplit('/', 1)
      (id, _)    = fname.split('.')
      body       = f.open(path).read()
                                                    …
```

```
 id   |                path                |                body
------+------------------------------------+-----------------------------------
 2482 | /Users/demo/blogsplog/model/2482.html | <!DOCTYPE html PUBLIC "...
    1 | /Users/demo/blogsplog/model/1.html    | <!DOCTYPE html PUBLIC "...
   10 | /Users/demo/blogsplog/model/1000.html | <!DOCTYPE html PUBLIC "...
 2484 | /Users/demo/blogsplog/model/2484.html | <!DOCTYPE html PUBLIC "...
...
```

# Parse the documents into word lists

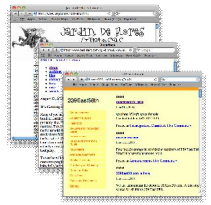
Convert HTML documents into parsed, tokenized, stemmed, term lists with stop-word removal:



```
-MAP:
    NAME:      extract_terms
   PARAMETERS: [id integer, body text]
    RETURNS:    [id int, title text, doc _text]
    FUNCTION: |

                if 'parser' not in SD:
                    import ...
                    class MyHTMLParser(HTMLParser):
                      ...
                    SD['parser'] = MyHTMLParser()

                parser = SD['parser']
                parser.reset()
                parser.feed(body)

                yield (id, parser.title, '{"' + '","'.join(parser.doc) + '"}')
```

# Parse the documents into word lists

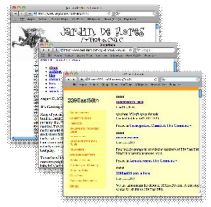
Use the HTMLParser library to parse the html documents and extract titles and body contents:



```
if 'parser' not in SD:
        from HTMLParser import HTMLparser
        ...
        class MyHTMLParser(HTMLParser):
          def __init(self):
            HTMLParser.__init__(self)
            ...
          def handle_data(self, data):
            data = data.strip()
            if self.inhead:
              if self.tag == 'title':
                 self.title = data
              if self.inbody:
        ...
parser = SD['parser']
parser.reset()
...
```

# Parse the documents into word lists

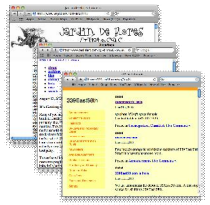
Use nltk to tokenize, stem, and remove common terms:



```
if 'parser' not in SD:
        from nltk import WordTokenizer, PorterStemmer, corpus
        ...
        class MyHTMLParser(HTMLParser):
          def __init(self):
            ...
            self.tokenizer = WordTokenizer()
            self.stemmer   = PorterStemmer()
            self.stopwords = dict(map(lambdax: (x, True),
        corpus.stopwords.words()))
          ...
          def handle_data(self, data):
            ...
            if self.inbody:
              tokens = self.tokenizer.tokenize(data)
              stems = map(self.stemmer.stem, tokens)
              for x in stems:
                if len(x) < 4: continue
                x = x.lower()
                if x in self.stopwords: continue
                self.doc.append(x)
 ...
parser = SD['parser']
parser.reset()
...
```

# Parse the documents into word lists

Use nltk to tokenize, stem, and remove common terms:



```
                    if 'parser' not in SD:
                        from nltk import WordTokenizer, PorterStemmer, corpus
                        ...
                        class MyHTMLParser(HTMLParser):
                            ...
                            if x in self.stopwords: continue
                            self.doc.append(x)
                        ...
                    parser = SD['parser']
                    parser.reset()
                    ...
```

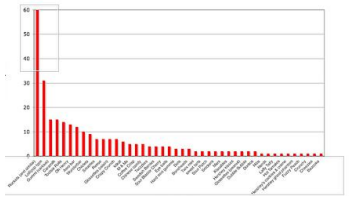
```
shell$  gpmapreduce -f blog-terms.yml
mapreduce_75643_run_1
DONE

sql#    SELECT id, title, doc FROM blog_terms LIMIT 5;

 id   |      title       |                                 doc
------+------------------+-------------------------------------------------------------------
 2482 | noodlepie        | {noodlepi,from,gutter,grub,gourmet,tabl,noodlepi,blog,scoff,...
    1 | Bhootakannadi    | {bhootakannadi,2005,unifi,feed,gener,comment,final,integr,...
   10 | Tea Set          | {novelti,dish,goldilock,bear,bowl,lide,contain,august,...
...
```



Greenplum

# Create histograms of word frequencies

Extract a term-dictionary of terms that show up in at least ten blogs



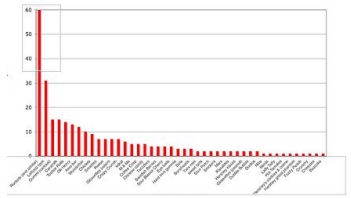
```
sql#SELECT term, sum(c) AS freq, count(*) AS num_blogs
    FROM (
      SELECT id, term, count(*) AS c
      FROM (
        SELECT id, unnest(doc) AS term
        FROM blog_terms
      ) term_unnest
      GROUP BY id, term
    ) doc_terms
    WHERE term IS NOT NULL
    GROUP BY term
    HAVING count(*) > 10;

 term    | freq | num_blogs
----------+------+-----------
 sturdi   |   19 |        13
 canon    |   97 |        40
 group    |   48 |        17
 skin     |  510 |       152
 linger   |   19 |        17
 blunt    |   20 |        17
```



Greenplum

# Create histograms of word frequencies

Use the term frequencies to construct the term dictionary…



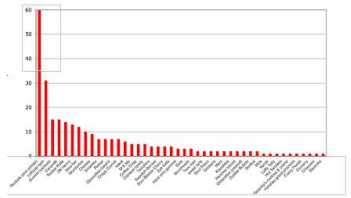
```
sql#    SELECT array(SELECT term FROM blog_term_freq) dictionary;

                          dictionary
-------------------------------------------------------------------
{sturdi,canon,group,skin,linger,blunt,detect,giver,annoy,telephon,...
```

…then use the term dictionary to construct feature vectors for every document, mapping document terms to the features in the dictionary:

```
sql#    SELECT id, gp_extract_feature_histogram(dictionary, doc)
        FROM blog_terms, blog_features;

 id  |                        term_count
-----+-----------------------------------------------------------
2482 | {3,1,37,1,18,1,29,1,45,1,...}:{0,2,0,4,0,1,0,1,0,1,...}
   1 | {41,1,34,1,22,1,125,1,387,...}:{0,9,0,1,0,1,0,1,0,3,0,2,...}
  10 | {3,1,4,1,30,1,18,1,13,1,4,...}:{0,2,0,6,0,12,0,3,0,1,0,1,...}
...
```

# Create histograms of word frequencies

Format of a sparse vector



```
 id  |                    term_count
-----+-----------------------------------------------------------------
...
  10 | {3,1,40,...}:{0,2,0,...}
...
```

Dense representation of the vector

```
 id  |                    term_count
-----+-----------------------------------------------------------------
...
  10 | {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...}
...
                        dictionary
-----------------------------------------------------------------------
{sturdi,canon,group,skin,linger,blunt,detect,giver,...}
```

Representing the document

```
{skin, skin, ...}
```

# Transform the blog terms into statistically useful measures

Use the feature vectors to construct **TFxIDF** vectors:

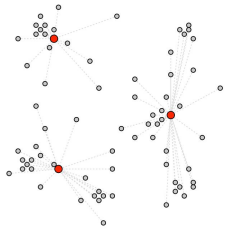
These are a measure of the importance of terms.

```
sql#  SELECT id, (term_count*logidf) tfxidf
      FROM blog_histogram, (
        SELECT log(count(*)/count_vec(term_count)) logidf
        FROM blog_histogram
      ) blog_logidf;

 id  |                          tfxidf
-----+-------------------------------------------------------------------
2482 | {3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.34311110...}
   1 | {41,1,34,1,22,1,125,1,387,...}:{0,0.771999985977529,0,1.999427...}
  10 | {3,1,4,1,30,1,18,1,13,1,4,...}:{0,2.95439664949608,0,3.2006935...}
...
```

# Create document clusters around iteratively defined centroids

Now that we have TFxIDFs we have something that is a statistically significant metric, which enables all sorts of real analytics.

The current example is k-means clustering which requires two operations.

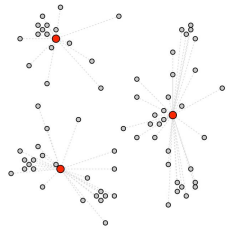


First, we compute a distance metric between the documents and a random selection of centroids, for instance cosine similarity:

```
sql#  SELECT id, tfxidf, cid,
        ACOS((tfxidf %*% centroid) /
          (svec_l2norm(tfxidf) * svec_l2norm(centroid))
        ) AS distance
      FROM blog_tfxidf, blog_centroids;

 id  |                        tfxidf                         | cid |  distance
-----+-------------------------------------------------------+-----+------------
2482 | {3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.3431111...} |   1 | 1.53672977
2482 | {3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.3431111...} |   2 | 1.55720354
2482 | {3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.3431111...} |   3 | 1.55040145
```

# Create document clusters around iteratively defined centroids

Next, use an averaging metric to re-center the mean of a cluster:



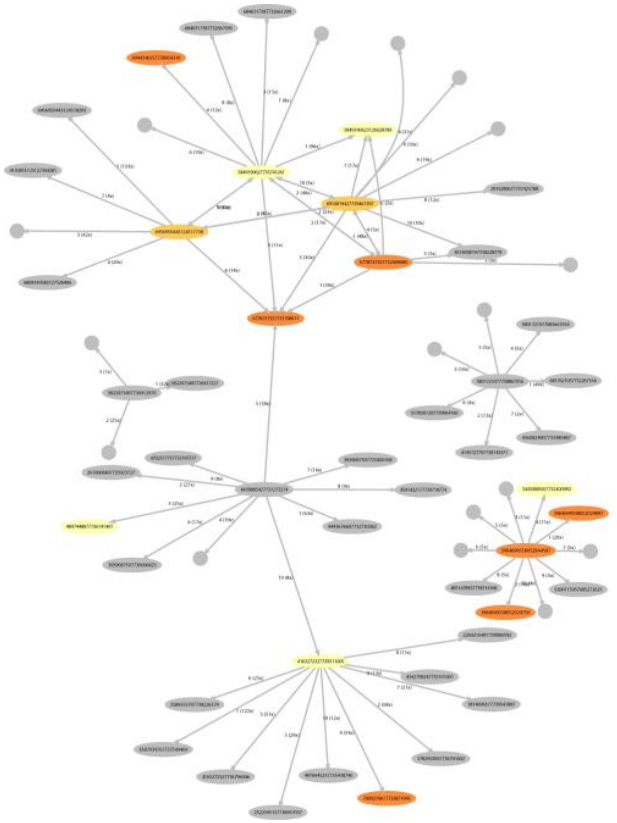
```
sql#  SELECT cid, sum(tfxidf)/count(*) AS centroid
      FROM (
        SELECT id, tfxidf, cid,
          row_number() OVER (PARTITION BY id ORDER BY distance, cid) rank
        FROM blog_distance
      ) blog_rank
      WHERE rank = 1
      GROUP BY cid;

 cid |                              centroid
-----+-------------------------------------------------------------------------
   3 | {1,1,1,1,1,1,1,1,1,...}:{0.157556041103536,0.0635233900749665,0.050...}
   2 | {1,1,1,1,1,1,3,1,1,...}:{0.0671131209568817,0.332220028552986,0,0.0...}
   1 | {1,1,1,1,1,1,1,1,1,...}:{0.103874521481016,0.158213686890834,0.0540...}
```

**Repeat the previous two operations until the centroids converge, and you have k-means clustering.**

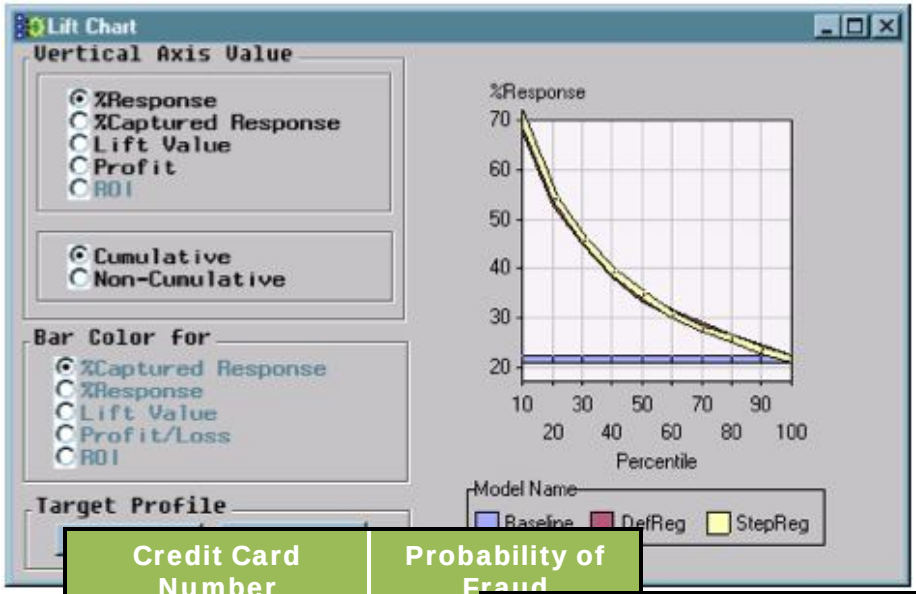
# MAD Analytics in Practice

# MAD Skills in practice

- Extracted data from EDW and other source systems into new analytic sandbox
- Generated a social graph from call detail records and subscriber data
- Within 2 weeks uncovered behavior where “connected” subscribers were seven times more likely to churn than average user



# Retention models




| Credit Card Number | Probability of Fraud |
|---|---|
| 7125 6289 5972 9510 | |
| 3955 8125 1327 7120 | |
| 2190 6379 9218 9290 | |
| 2760 1924 2864 0950 | |
| 4915 1908 8302 9940 | |
| 3534 7203 6200 4010 | |

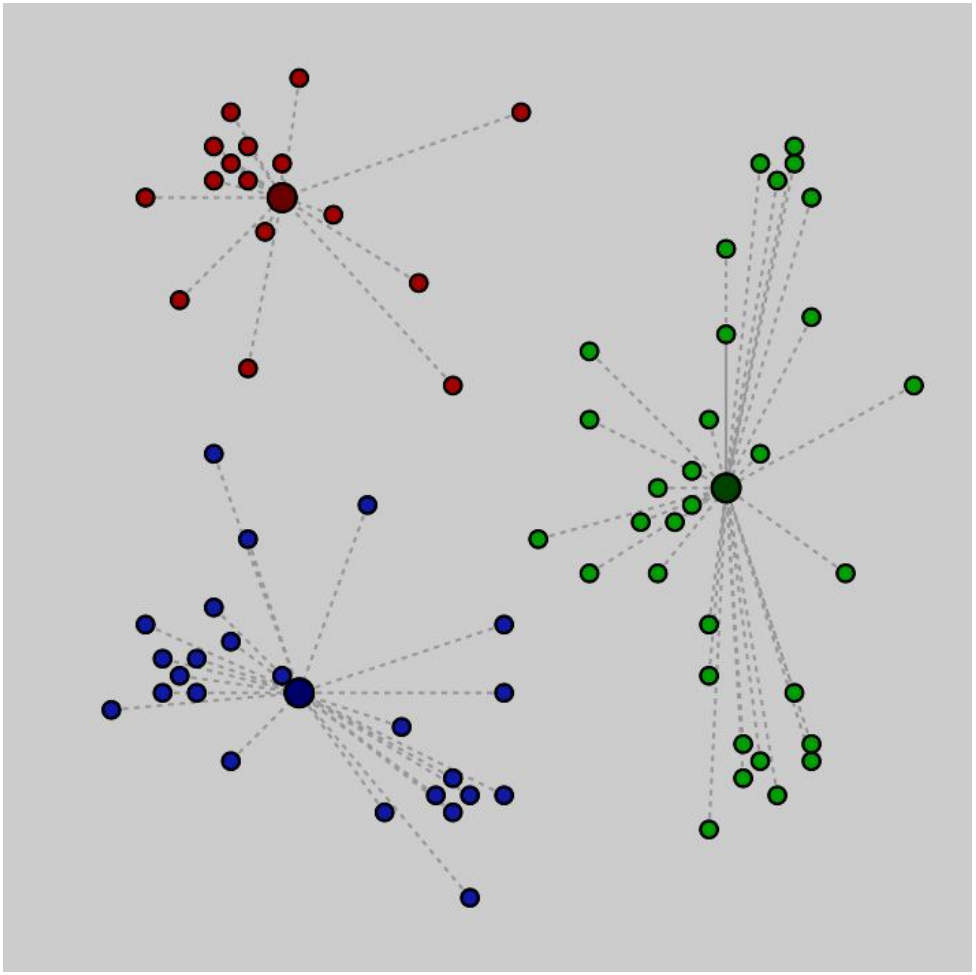
| SSN | Probability of Churn |
|---|---|
| 611-43-2435 | 15% |
| 812-35-1035 | 22% |
| 253-23-2943 | 7% |
| 732-62-1435 | 47% |
| 483-32-5298 | 12% |
| 821-90-8574 | 3% |

- Customer Retention
  - Identify those at risk of abandoning their accounts.
  - Use logistic regression models, or SAS scoring models.
- Also used to predict…
  - fraud in on-line and financial transactions
  - hospital return visits
  - etc.



Greenplum

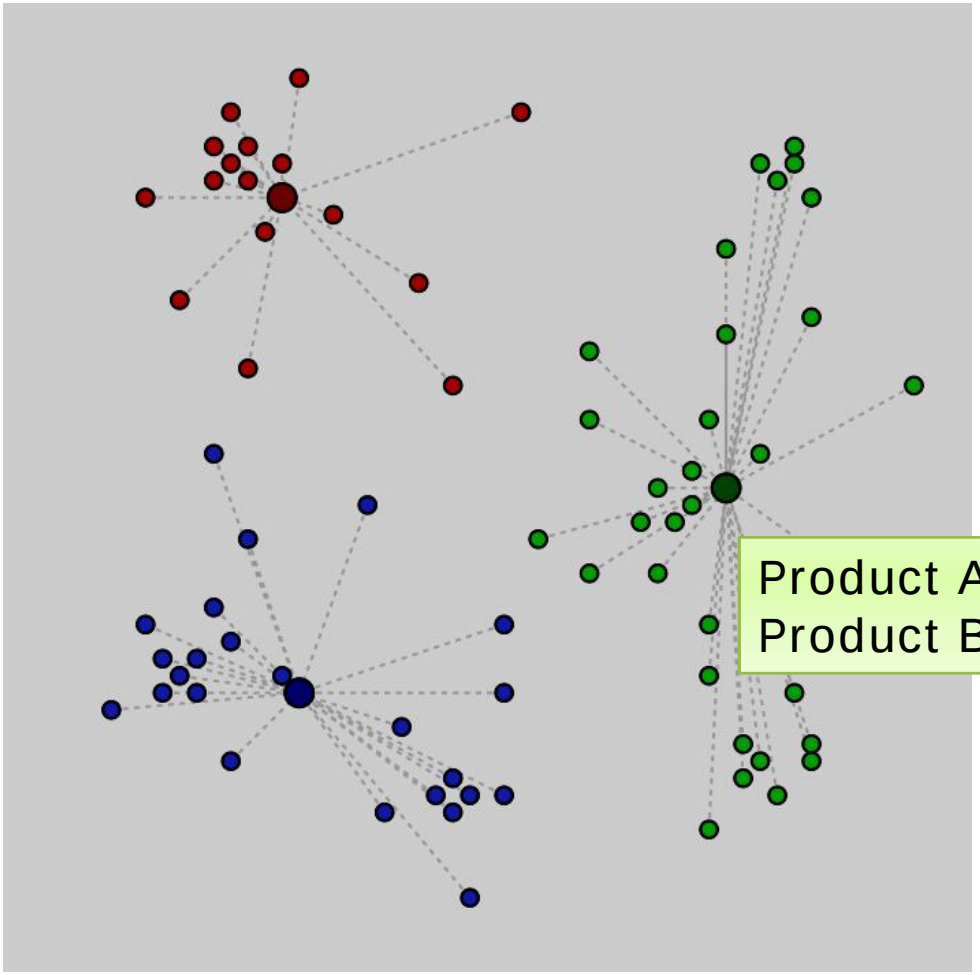
# Segmentation



## Using segments

– Create clusters of customers based on profiles, product usage, etc.

# Association Rules



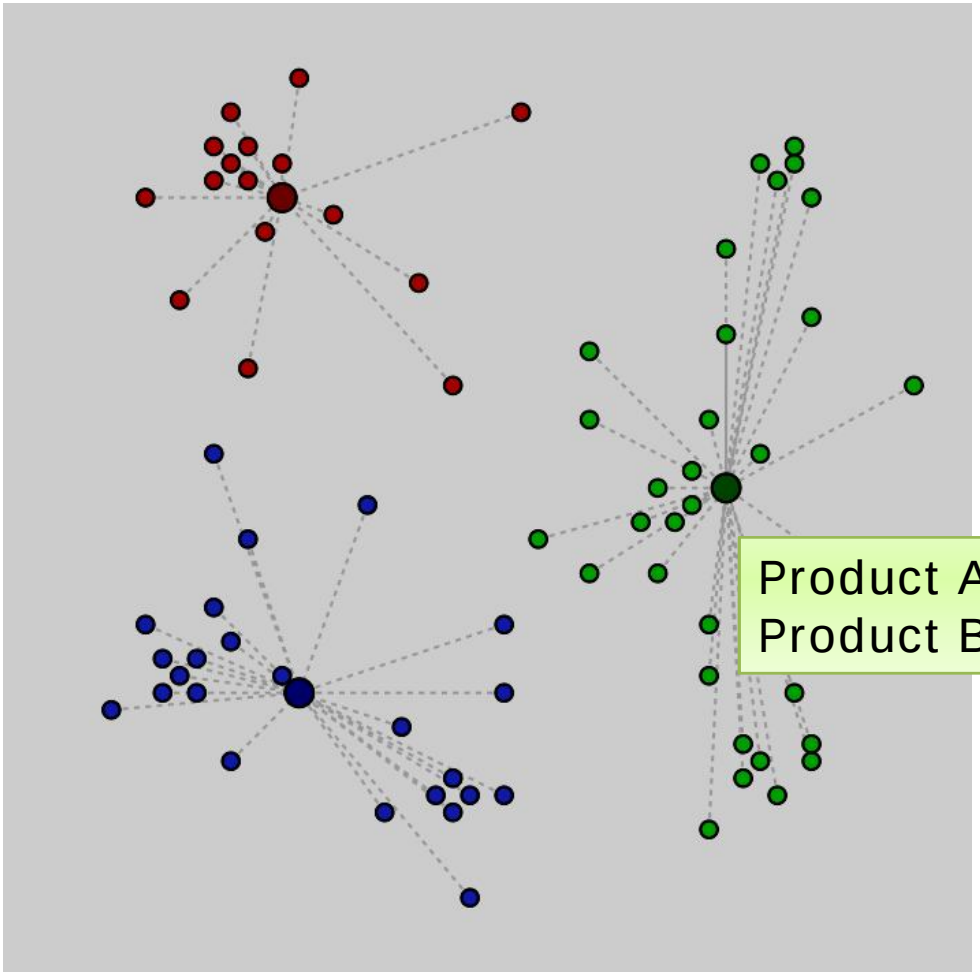

## Using segments

– For low or medium-value customers, compute possible new products using association rules



Greenplum

# Segmentation and Association Rules




## Using segments

– Filter down to products associated with high-value customers in the same segment.



Greenplum

# Questions