From DBMS to BDMS

After this lesson you will be able to:

- Explain at least 5 desirable characteristics of a Big Data Management System
- Explain the difference between ACID and BASE
- Describe what the CAP Theorem states
- List examples of BDMSs and describe some of their similarities and differences

FROM DBMS TO BDMS

(capabilities / features)

ideal

Desired Characteristics of BDMS

big data management system

- **A flexible, semistructured data model** — flexible: can take many forms (not only xml)
  - "schema first" to "schema never"
- **Support for today's common "Big Data data types"** (text, docs, social media /time components, spacial data (geo))
  - Textual, temporal, and spatial data values
- **A full query language** — effectively manage large volumes of data (more convenient) → query automatically determines optimal ways to recieve data
  - Expectedly at least the power of SQL — most DBMS vendors offer own extension of SQL
- **An efficient parallel query runtime**

that runs in multiple machines ↖ connected to shared nothing — shared memory — architectures shared cluster

* critical requirement

Desired Characteristics of BDMS

- **Wide range of query sizes**
- **Continuous data ingestion**
  - Stream ingestion
- **Scale gracefully to manage and query large volumes of data**
  - Use large clusters
- **Full data management capability**
  - Ease of operational simplicity

— not emphasized as it should be
— apps (generated by tools or machine learning) have many conditions return many objects

streaming data (volume) + large data

— design and operate over cluster
knows how to handle a failure
handle new machines join & / or leaving existing

↳ data management capabilities — operational management as simple as possible

ACID and BASE

too much data
too many updates
from too many users

- **ACID properties hard to maintain in a BDMS** — may lead to a significant slow down of the system
- **BASE relaxes ACID**
  - BA: Basic Availability : system guarantee the availability of data
    request → response
  - S: Soft State : state changes (always soft)
  - E: Eventual Consistency — becomes consistent when stops receiving input
    ↳ does not check consistency of everything all the time

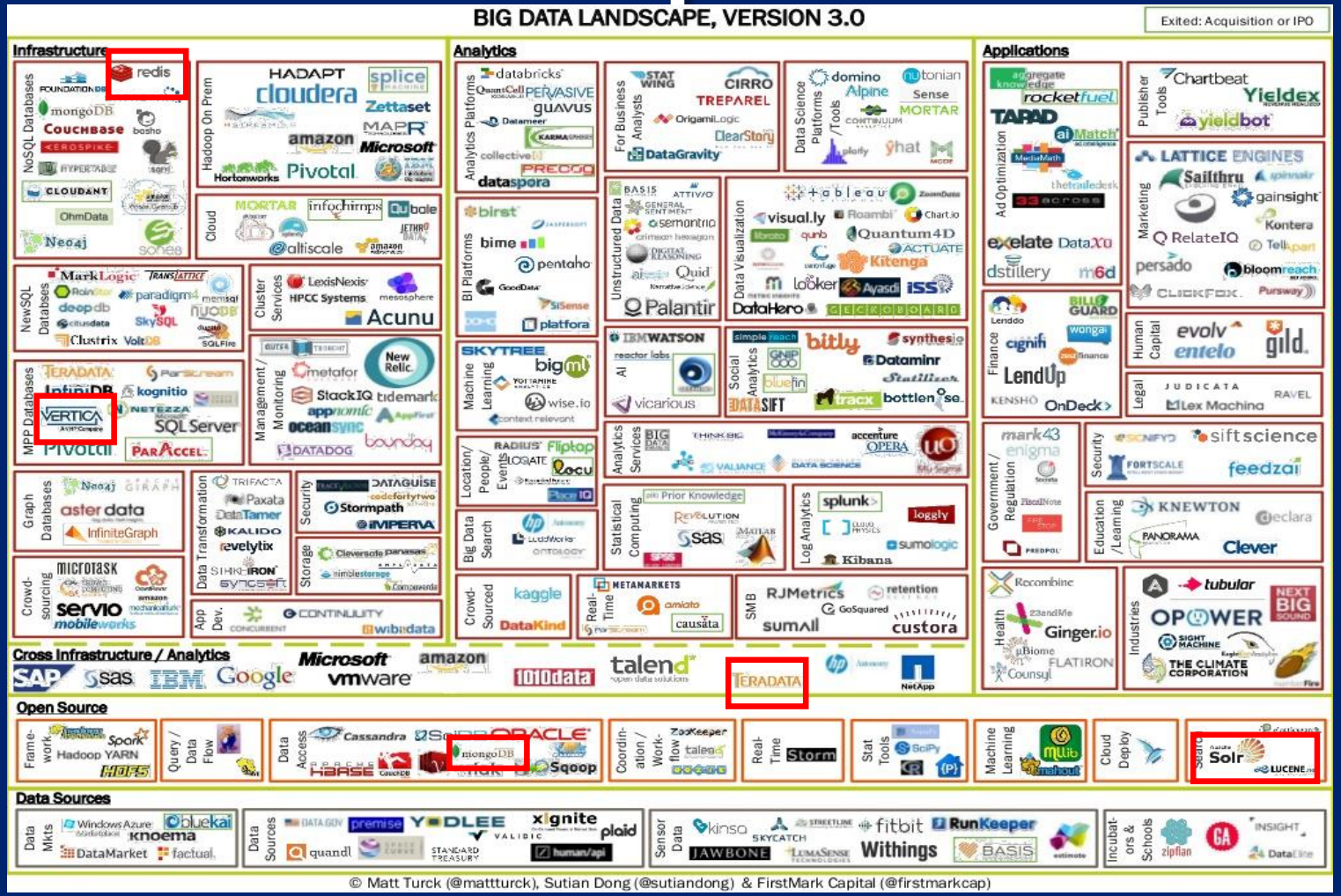
    (because there are transactions to process)

CAP Theorem (Brewer's theorem)

- A distributed computer system (impossible to) cannot simultaneously achieve...
  - **Consistency** : all nodes see the same data at any time
  - **Availability** : every request recieves a response (< success fail >)
  - **Partition Tolerance** : system continues to operate despite partitioning (due to network failures)

The Marketplace

Matt Turk's depiction of big data products



categories:
- no SQL
- massively parallel databases
- analytical systems
- ... (etc)

quick tour through products

- assess aspects of ideal BDMS covered
- wether they have obvious limitations
- highligh aspects relevant to BDMS discussion

pause

REDIS

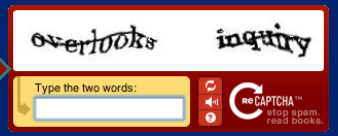
Redis – An Enhanced Key-Value Store

- not a full blown DBMS
  - persists data on disks
  - saves its state
  - BUT intended use is optimally use memory and methods to make data structures fast for users

- **In-memory data structure store**
  - data structures it supports:
    - strings, hashes, lists, sets, sorted sets
- **Look-up Problem**
  - **Case 1: (key:string, value:string)**

. REDIS string can be binary

Can this be a key?



key: ID of image (string) → image itself instead as key
Value: "overlooks inquiry" (also string) desired text
  - content as binary string
* key size up to 512 mB

  - Keys may have internal structure and expiry
    - comment:1234:reply.to
    - Hierarchical keys: user.commercial, user.commercial.entertainment, user.commercial.entertainment.movie-industry

. REDIS can generate new keys

how long to keep it around?

eg: product codes

. benefits to structured keys: it can encode a hierarchy to the structure

prod family + manufacturing batch + product ID =

one ID
↳ represent increasingly finer subgroups

* data values have limited utility beyond certain period

Redis and Data Look-up

- **Case 2: (key:string, value: list)**
  - userID: [tweetID1, tweetID2, …]
  - Ziplists compress lists – reduction in memory use
  - Twitter innovation: list of ziplists
    - http://www.infoq.com/presentations/Real-Time-Delivery-Twitter
    - https://www.youtube.com/watch?v=rP9EKvWtozo

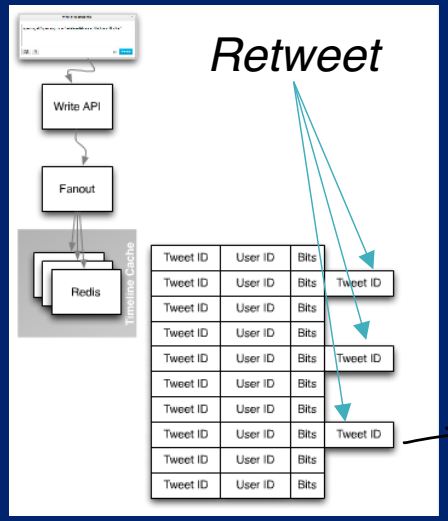
not atomic, a collection set

twitter stores timelines

compacts size of listing memory

insertion and deletion can be performed in milliseconds (constant time)

- long lists → space saving
  - ziplists * open source
    - efficient for retrieval
    - complex for insertion and deletion



RT: ID added to list of first tweet

Redis and Data Look-up

- **Case 3: (key:string, value: attribute-value pairs)**
  - REDIS Hashes — name and their values containers of unique fields
    - std:101 name:“John Smith" dob:01-01-2000 gender:M active:0 cgpa:2.9

→ retrieval is efficient

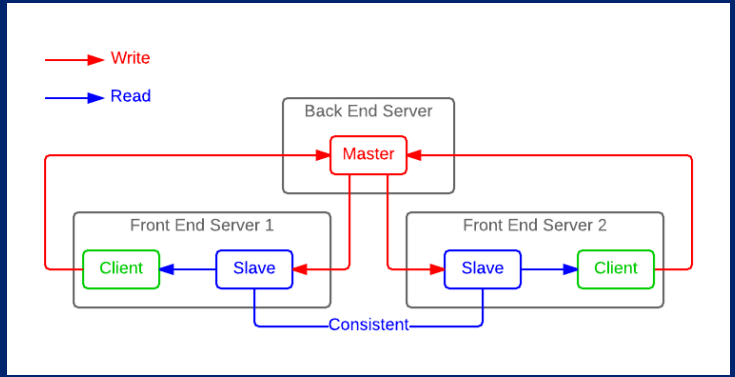
Redis and Scalability

scale out (horizontal scalability): ability to achieve scalability when the number of machines increases

- **Partitioning and Replication**
  - Range partitioning — takes numeric key an breaks up the range of keys into bins
    - Example: User record number 1-10000 goes to machine 1, 10001-20000 goes to machine 2, … → bins of 10,000 → each bin assigned a machine
  - Hash partitioning
    - Pick a key of a record, e.g., "abcde"
    - Using a hash function, turn it into a number, e.g., 152
    - 152 mod 10 is 2, so the record goes to machine 2

Redis and Scalability

- **Partitioning and Replication**
  - Master-Slave mode replication
    - Clients write to master, master replicates to slaves — slaves have copy of master node
    - Clients read from slaves to scale up read performance — slaves can serve read queries
    - Slaves are mostly consistent — replication process ensures that they are consistent with each other



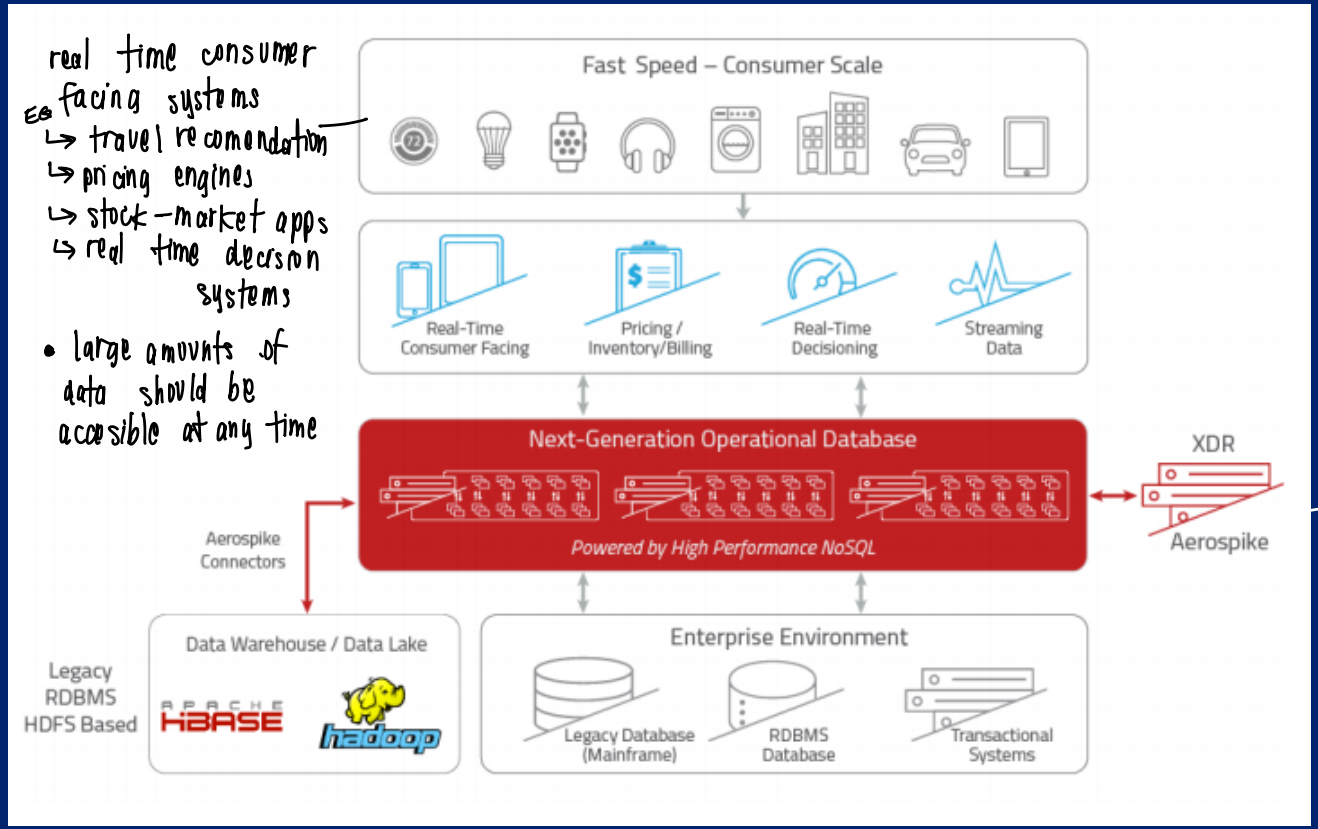
pause

AEROSPIKE

Aerospike – a New Generation KV Store

real time consumer facing systems
Eg
↳ travel recomendation
↳ pricing engines
↳ stock-market apps
↳ real time decision systems

- large amounts of data should be accesible at any time



Aerospike: distributed NoSQL database and Key value store

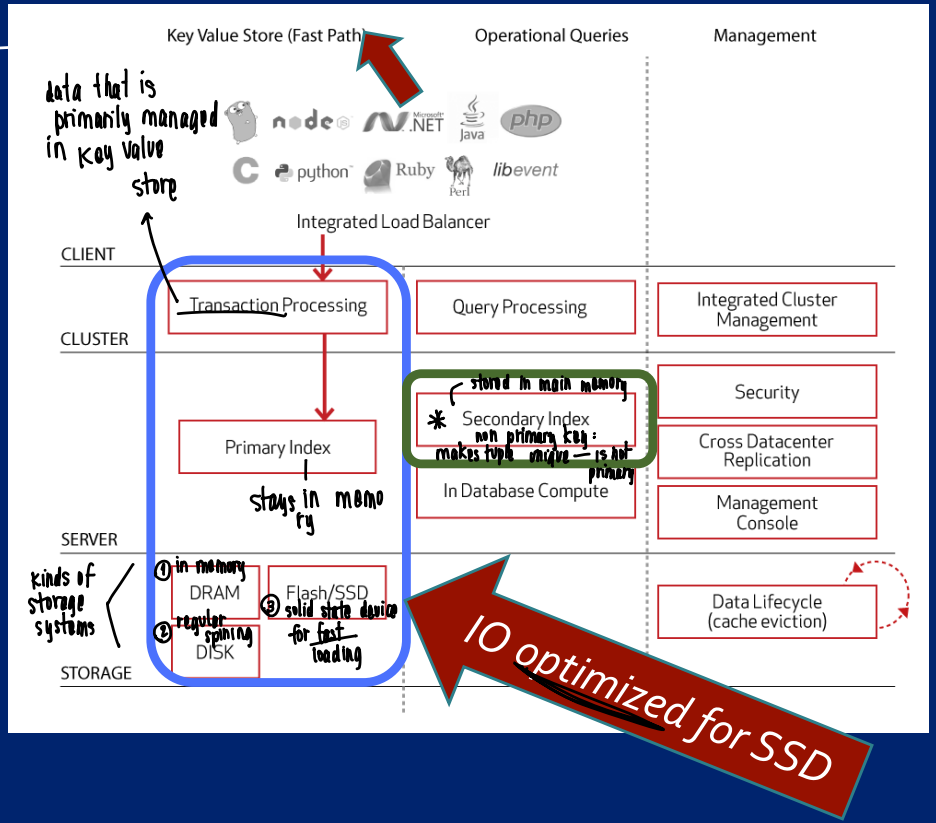
- architected for performance needs of "today" web scale apps

how aerospike relates to the ecosystem for which it is designed

high level

Aerospike Architecture

fast path
(left side of architecture)



Querying Aerospike

* focused on real time web apps

- **Data types**
  - Standard scalar, lists, maps, geospatial, large objects

integer, string

places with latitude and longitude values
regency polygons

- **KV store operations**
  - Geospatial queries like point-in-polygon — is the location of this in La Jolla?

standard query language

distance query : find hotels within three miles of my location

- **AQL: an SQL-like language**

more declarative language

  - SELECT name, age FROM users.profiles

namespace
record, belongs to

- advocate functions like = user defined, sum, average — may be evaluated through map reduce style operation

values

Transactions in Aerospike

* offers ACID guarantees ; accomplished using techniques

mechanisms that many systems use to balance between
and
- large scale data management
- transaction management in a cluster where nodes can join or leave

- **Aerospike ensures ACID**
  - Consistency – all copies of a data item are in sync
    - ensure constraints are satisfied
    - ensure copies are in sync
    - Uses synchronous write to replicas within the cluster — write process is considered successfull only if the replica is updated
    - Mechanisms to relax immediate consistency — by bypassing all consistency checks
      * eventual consistency will still be enforced
  - Durability –
    - Flash storage on every node — direct reads from it
    - Replication management — we have multiple copies of data → if one node fails, the latest copy of the last data is available from one or more replica nodes in same clusters as well as in nodes residing in remote clusters
  - Network partitioning reduced
    - Tighter cluster control → nodes in different parts of network have different data content

contradict CAP theorem?

* master nodes know where nodes are & proper replication

pause

ASTERIX DB

AsterixDB – a DBMS for Semistructured Data

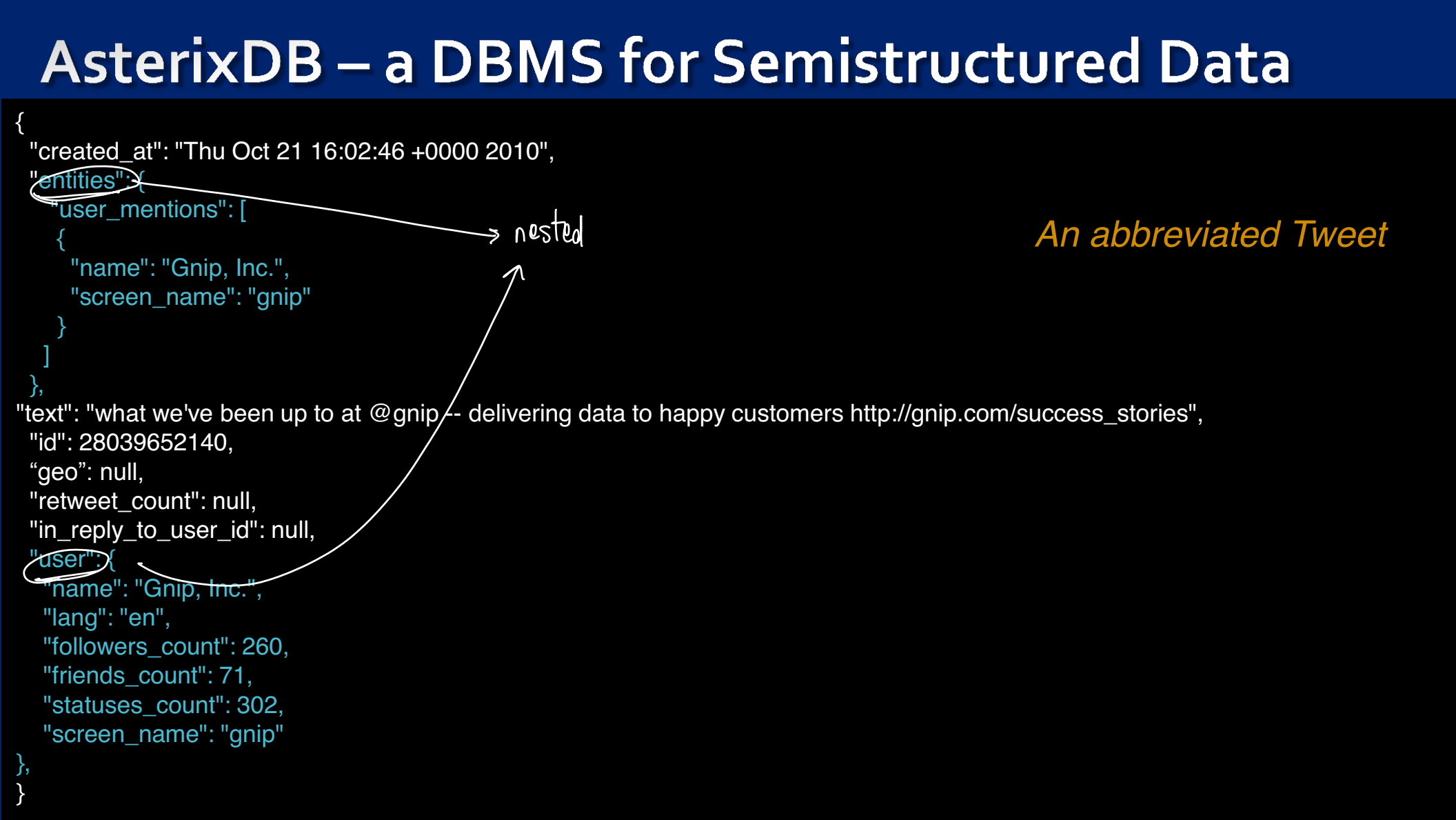
└ new, incubated by apache

* Mongo DB for json style

• asset guarantees

AsterixDB – a DBMS for Semistructured Data


```
{
  "created_at": "Thu Oct 21 16:02:46 +0000 2010",
  "entities":{
     "user_mentions": [
     {
       "name": "Gnip, Inc.",
       "screen_name": "gnip"
     }
    ]
  },
"text": "what we've been up to at @gnip -- delivering data to happy customers http://gnip.com/success_stories",
  "id": 28039652140,
  “geo”: null,
  "retweet_count": null,
  "in_reply_to_user_id": null,
  "user":{
     "name": "Gnip, Inc.",
     "lang": "en",
     "followers_count": 260,
     "friends_count": 71,
     "statuses_count": 302,
     "screen_name": "gnip"
},
}
```

An abbreviated Tweet

Semistructured Schema

```
create dataverse LittleTwitterDemo;
create type TwitterUserType as open {
    screen-name: string,
    lang: string,
    friends_count: int32,
    statuses_count: int32,
    id: int32,
    followers_count: int32
}
create type TweetMessageType as closed {
    tweetid: string,
    user: TwitterUserType,
    geo: point?,
    created_at: datetime,
    referred-topics: {{ string }},
    text: string
}
create dataset TweetMessages(TweetMessageType)
primary key tweetid;
```

namespace for data

declared in terms of data types

user portion

represents message

can handle * spatial data

* ? question mark says its optional

instead of nesting
└ captures hierarchical structure of JSON

actual data can have more attributes than specified there

data instance must have the same attributes

Options for Querying in AsterixDB

- AQL is a natively-supported query language

```
for $user in dataset TwitterUsers order by
$user.followers_count desc, $user.lang asc return $user
```

- Hive queries

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON
(a.key=b.key)
```

- Xquery
- Hadoop MR jobs
- SQL++ (coming up)

has its own language

* provides options for query support

what it looks like

all users in TwitterUsers

followers number

preferred language

hiveQL

* engine can process queries in multiple languages

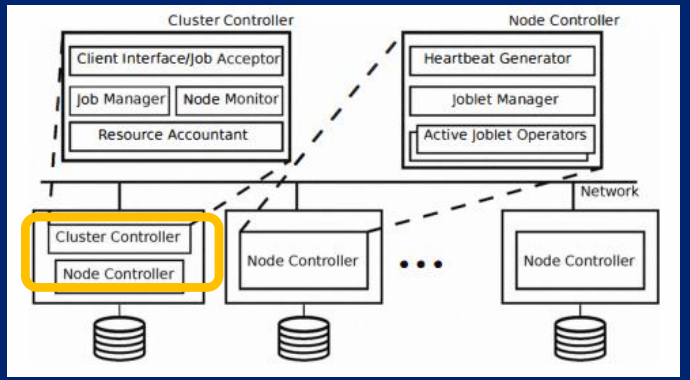
extends SQL for json

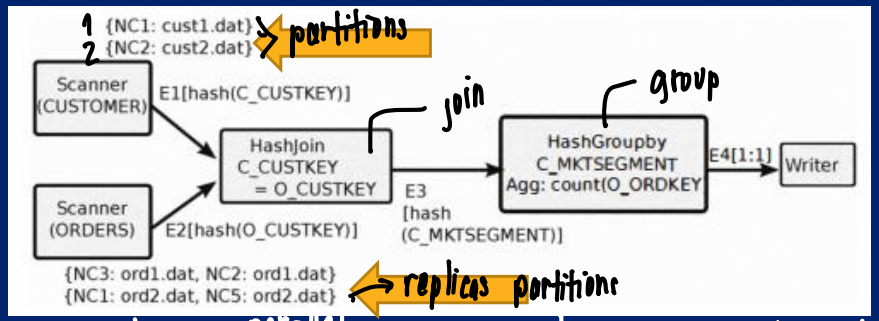
Operating Over a Cluster

- **Hyracks** — used for partitioned parallel execution of query plans
  - Query execution engine for partitioned parallel execution of queries
    (data parallelism)
- **Example**
  - CUSTOMER (C_CUSTKEY, C_MKTSEGMENT, …)
  - ORDERS (O_ORDERKEY, O_CUSTKEY, …)

(relations)

Hyracks Job Management





jobs < parallel stage by stage

managed by cluster controller

QUERY:
find number of orders for every market segment that the customers belong to

datasets — divided into instances of various types

instances

can be decomposed to different machines by
< range partitioning
hash "

Accessing External Data

- Real-time data *from files in a directory path*

external data sources

```
1 create dataset Tweets (Tweet)
    primary key id;

2 create feed TestFileFeed using localfs
  (("path"="127.0.01:///Users/adc/text/"),
  ("format"="adm"), ("type-name"="Tweet"),
  ("expression"=".*\\.adm"));

3 connect feed TestFileFeed to dataset Tweets;
```

local file system

location of directory

every 5min into JSON

1. create empty data set (Tweets)
2. create feed (external data source)
3. feed connected to dataset
   ↓
   system starts reading unread files from directory

Accessing External Data

- Real-time data *from an external API*

```
use dataverse feeds;
create dataset Tweets (Tweet)
    primary key id;
create feed TwitterFeed if not exists using "push_twitter"
    (("type-name"="Tweet"),
     ("consumer.key"="some-key"),
     ("consumer.secret"="some-secret"),
     ("access.token"="some-token"),
     ("access.token.secret"="some-token-secret"));
connect feed TwitterFeed to dataset Tweets;
```

1

2 this method invokes the twitter client with the four authentication parameters required by the API

3

pause

SOL R

Solr – Managing Text

— text indexing engine
└ for search problems (search engines)

- **Basic challenges with text** ← storing, indexing, matching data
  - Defining a match — text vary, find good match
    - Analyze ≈ Analyse ≈ ANALYZE? **Lexical difference, capitalization** — spelling variations
    - abc:def-230-39 ≈ abcdef23039? **Structural punctuations** — different parts of a string represent different kinds of info
      - like: file paths, URLs, product ID
      - └ the searcher may not always know the structure
    - "Barak Hussein Obama" ≈ "Barak Obama" ≈ "Barak H. Obama" ≈ "B. H. Obama"? **Nominal Variations** — common in proper nouns
    - Mom ≈ mother? **Synonyms**
    - Dr. ≈ Doctor **Abbreviation**
    - USA ≈ "United States of America" **Initialism** (new) special case of abbrev
      - ★ initialism that can be said as real word is acronym (NATO)
    - "The tradition is completely American. Students should …"
      - Should this match the query "American students"? NO
    - "Mrs. Clinton also said …"
      - Should this match the query "mrs Clinton"? YES

Inverted Index

Lucene : the engine on which solr is built — not a database

- **Vocabulary**
  - All terms in a collection of documents
    - Multi-word terms, synonym sets
- **Occurrence**
  - For each term in the collection
    - List of doc ID
    - List of doc ID, [position of occurrence]
    - Other statistics like tf, idf, ...

Inverted index

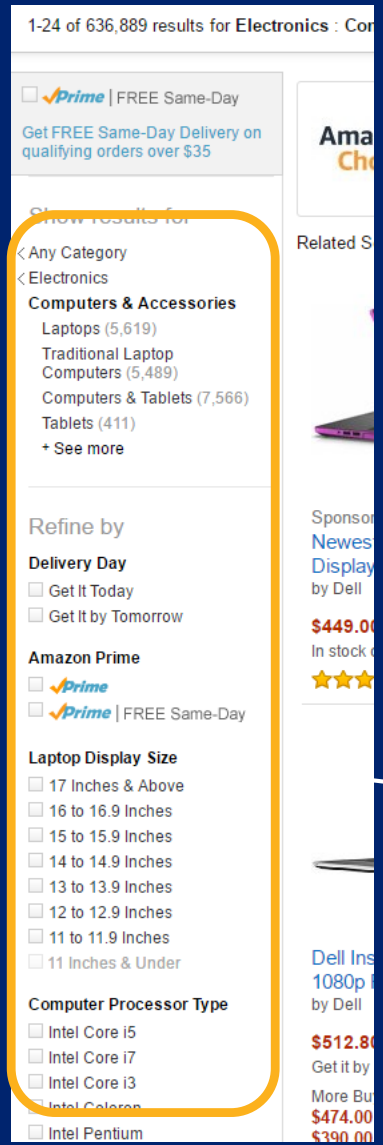
tf: term frequency

idf: inverse document frequency

Solr Functionality

- **Enterprise Search Platform**
- **Inverted index**
  - For every field in a structured text document
    - indexes text, numbers, geographic information, ...
- **Faceted Search**
- **Term highlighting**

(input)



faceted:
extracts individual values of fields with count
↓
called facets

Solr Functionality

how to parse
- how to tokenize: the process of breaking down the characters read
- how to filter

- **Index-time Analyzers**
  - Tokenizers
  - filters

```
<fieldType name="text" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StandardFilterFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymFilterFactory"
synonyms="synonyms.txt" ignoreCase="true" expand="true" />
    <filter class="solr.EnglishPorterFilterFactory"/>
  </analyzer>
</fieldType>
```

- tokenizer class="solr.StandardTokenizerFactory"/> — gets words with immediate punctuation
- filter class="solr.StandardFilterFactory"/> — removes punctuation
- filter class="solr.LowerCaseFilterFactory"/> — turns everything into lowercase
- filter class="solr.SynonymFilterFactory" — uses synonym file
- filter class="solr.EnglishPorterFilterFactory"/> — removes common english words

Solr Functionality

- **Query-time Analyzers** — get query string

```
<analyzer type="query">
   <tokenizer class="solr.PatternTokenizerFactory"
pattern="[\s,\.;]+"/>
     <filter class="solr.StandardFilterFactory"/>
     <filter class="solr.LowerCaseFilterFactory"/>
     <filter class="solr.CommonGramsFilterFactory"
words="stopwords.txt" ignoreCase="true"/>
     <filter class="solr.StopFilterFactory"
ignoreCase="true" words="stopwords.txt"/>
     <charFilter
class="solr.HTMLStripCharFilterFactory"/>
     <filter class="solr.PorterStemFilterFactory"/>
 </analyzer>
```

executed in order

— removes whitespaces, periods, semi-colon

— creates tokens out of pairs of terms "the cat"

— removes stopwords in the file

— HTML character stripped off

— remaining words are stemmed

Solr Queries — searches

Consider the CSV file

```
id,cat,pubyear_i,title,author,series_s,sequence_i
book1,fantasy,2001,American Gods,Neil Gaiman,American Gods,
book2,fantasy,1996,A Game of Thrones,George R.R. Martin,A Song of Ice and Fire,1
book3,fantasy,1999,A Clash of Kings,George R.R. Martin,A Song of Ice and Fire,2
book4,sci-fi,1951,Foundation,Isaac Asimov,Foundation Series,1
book5,sci-fi,1952,Foundation and Empire,Isaac Asimov,Foundation Series,2
book6,sci-fi,1992,Snow Crash,Neal Stephenson,Snow Crash,
book7,sci-fi,1984,Neuromancer,William Gibson,Sprawl trilogy,1
book8,fantasy,1985,The Black Company,Glen Cook,The Black Company,1
book9,fantasy,1965,The Black Cauldron,Lloyd Alexander,The Chronicles of Prydain,2
```

Solr Queries — queries to post

- All books
  - http://localhost:8983/solr/query? q=*.*
- All books with "black" in the title field, return author, title
  - http://localhost:8983/solr/query? q=title:black &fl=author,title
- All books sort by pubyear_i in descending order
  - http://localhost:8983/solr/query? q=*.*&sort=pubyear_i desc
- Above query but facet by all values in the cat field
  - http://localhost:8983/solr/query? q=*.*&sort=pubyear_i desc &facet=true&facet.field=cat

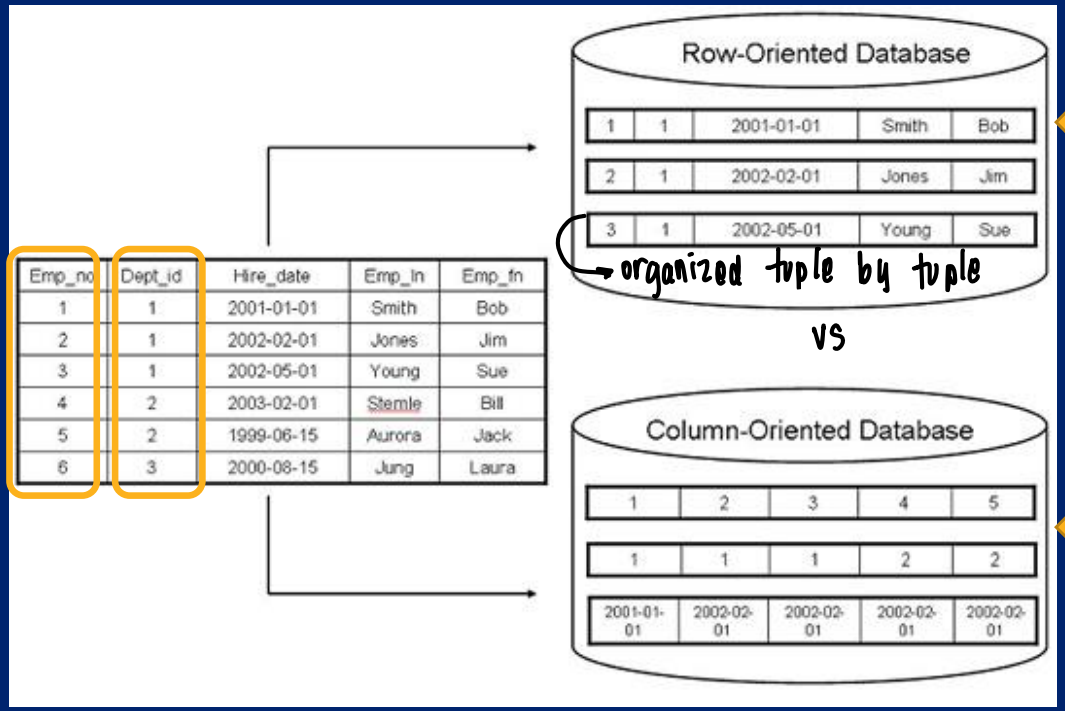
q= — query

fl — what you want back

pause

relational data
VERTICA

Vertica – a Columnar DBMS



Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

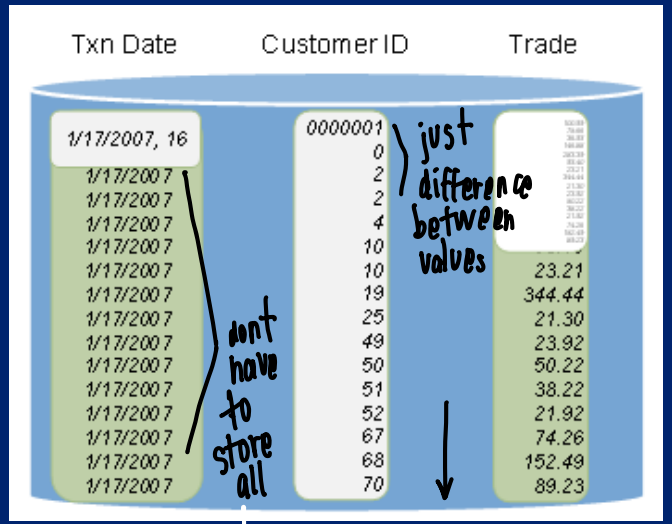
- **Column Store**
  - Store data ~~column-wise~~ organized
  - A query only uses the columns needed * full record is not used most of the time
  - Usually must faster for queries even for large data

relational DBMS designed to operate on top of HTFS

Space Efficiency

- Column stores keep columns in sorted order
- Values in columns can be compressed
  - Run-length encoding *
    - 1/1/2007 – 16 records
  - Frame-of-reference encoding **
    - Fix a number and only record the difference
- Compression saves storage space

this form of shortened representation



BDMs should use all tricks to improve space efficiency

Working with Vertica

- **Column-Groups** — mini table
  - Frequently co-accessed columns behave as mini row-stores within the column store

improves performance

- **Update performance slower** (writing slower)
  - Internal conversion from row-representation to column-representation
- **Enhanced suite of analytical operations**
  - Window statistics
    - Ticks (ts TIMESTAMP, Stock varchar(10), Bid float

1 — more statistic functions than classical DBMS

2 —

Vertica and Analytic Functions

table

```
'2011-07-12 10:23:54', 'abc', 10.12
'2011-07-12 10:23:58', 'abc', 10.34
'2011-07-12 10:23:59', 'abc', 10.75
'2011-07-12 10:25:15', 'abc', 11.98
'2011-07-12 10:25:16', 'abc'
'2011-07-12 10:25:22', 'xyz', 45.16
'2011-07-12 10:25:27', 'xyz', 49.33
'2011-07-12 10:31:12', 'xyz', 65.25
'2011-07-12 10:31:15', 'xyz'
```

1min 26secs after previous

```
SELECT ts, bid, AVG(bid)
  OVER(ORDER BY ts
    RANGE BETWEEN INTERVAL '40
seconds' PRECEDING AND CURRENT ROW)
FROM ticks
WHERE stock = 'abc'
GROUP BY bid, ts
ORDER BY ts;
```

average

range

on last column

query

```
 ts                  | bid   |    ?column?
---------------------+-------+-----------------
 2011-07-12 10:23:54 | 10.12 |           10.12
 2011-07-12 10:23:58 | 10.34 |           10.23
 2011-07-12 10:23:59 | 10.75 | 10.4033333333333
 2011-07-12 10:25:15 | 11.98 |           11.98
 2011-07-12 10:25:16 |       |           11.98
(5 rows)
```

→ doesn't consider previous

result of query

moving average of bid every 40 seconds

* happening inside the database

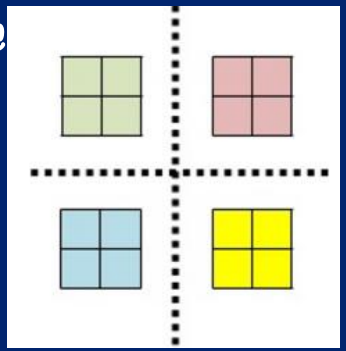
row by row, for each computation only considers timestamps within 40 seconds

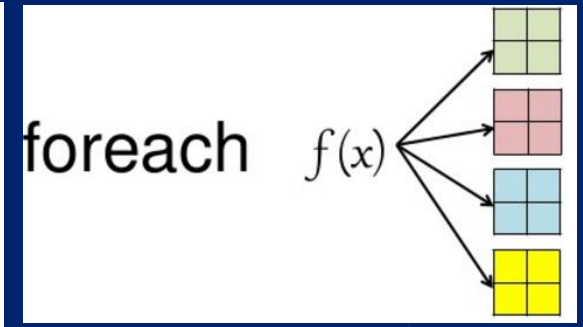
Vertica and Distributed R

second feature: statistics package

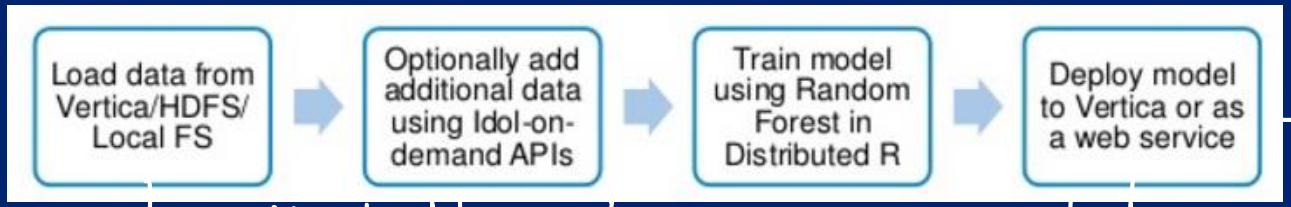
- can read data from files

- **Distributed R** — high performance platform for R
  - High-performance statistical analysis
  - Master node: schedules tasks and sends code (from SQL database)
  - Worker nodes: maintain data partitions and compute
  - Uses a data structure called dArray or distributed array — partitioned like that





compute function foreach mini-array



workflow of constructing and deploying predictive model

vertica is data supplier + model consumer

data to be analyzed is output of vertica query

"vertica fast transfer"