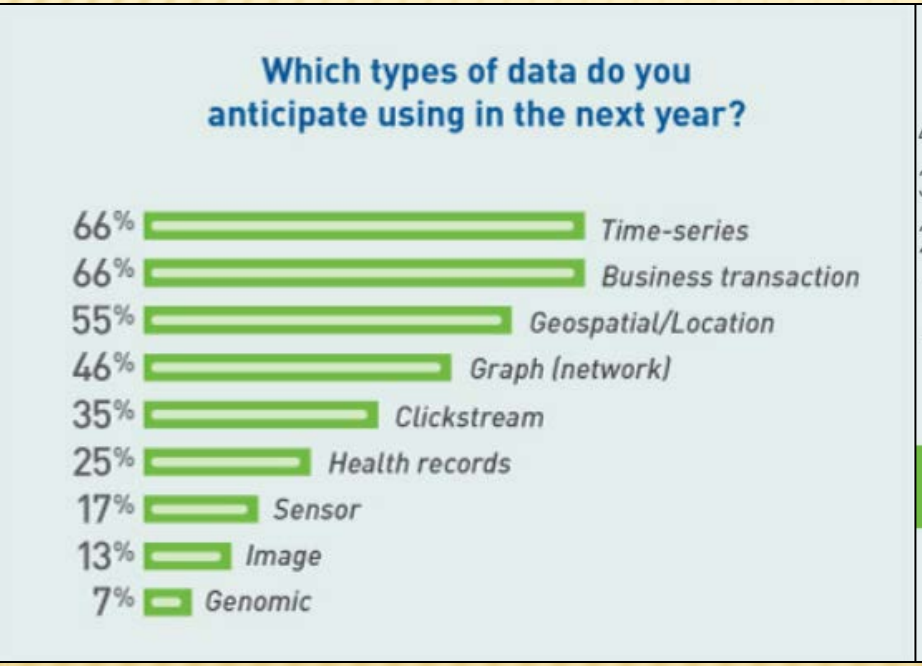
SDSC Summer Institute 2014, HPC Meets Big Data

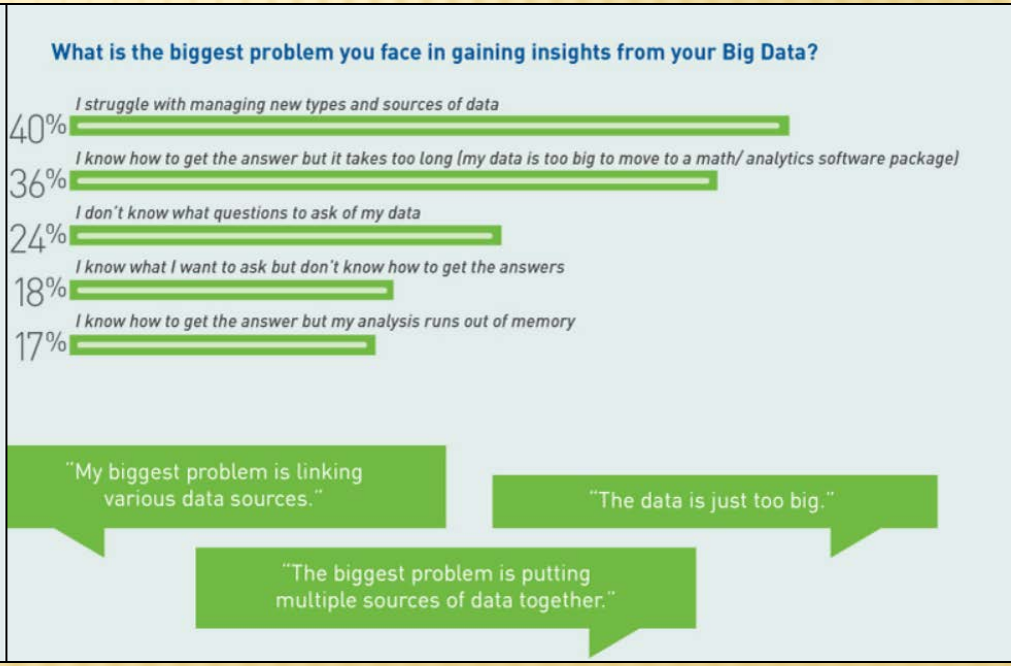
Amarnath Gupta

HOW DO I HANDLE MY BIG DATA PROBLEM?

WHAT IS A "BIG DATA" PROBLEM?

- 111 data scientists were surveyed about their current and future work and their problems





The "big data" problem in data science is *not always* about storage of exabytes of data and computational grids with petaflop performance

WHAT IS MY BIG DATA PROBLEM? (TAKE-1)

- Some examples (sampled from your applications)
  - Have very large geospatial data that I need to manipulate and run analytics on
  - Analyze activity logs (text, timestamps…) of many millions of users to find patterns
  - Computation on large graphs (e.g., social networks) is complex
  - Need to scale Natural Language Processing algorithms to millions of documents
  - Analyzing a large number of multi-modal streaming data is difficult

THE KNOWLEDGE-GAP WE ADDRESS TODAY

- We will not try to define “big data”
  - But we discuss several *data situations* and technologies
- We show
  - Data scientists can encounter the “bigness barrier” much before they approach huge numbers
- We contend that the knowledge gap is mostly related to:
  - Providing the scientists and information analysts with too much of too complex information
  - Asking traditional data engineers to provide more analysis than they are used to or integrating data they are not used to
  - Managing the economics of information
    - Should we go buy a bigger, faster machine, newer software, or manage the problem ourselves with more programming?
  - Figuring out what technology to align with for “my specific problem space”
- When and how do HPC techniques meet “big data”?

BIG DATA AND ANALYTICS

Angle 1: A key problem with a lot of real world datasets is **sparsity**. The higher the dimensionality of the data, it is harder to build statistical models on a small dataset.

Angle 2: Some problems are just **naturally large**, and most current algorithms are inefficient, and many serious computations need a cluster of machines.

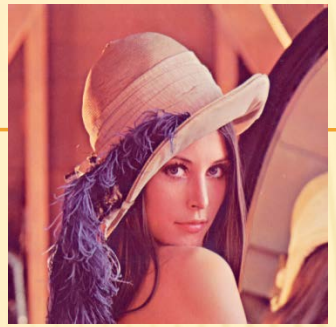
Angle 3: It is often the case that **algorithms** that work on a relatively small amount of data do very well with larger data. So one can research on a small sample, and then on larger data on analytical software (*assuming it can support a ton of data*) to construct models.

Angle 4: There are whole **classes of problems do not work on small datasets**, and need extremely large unlabeled sets of data to draw statistical conclusions on their own by minimizing energy functions under constraints.

Angle 5: A lot of data are collected about user behavior. To create recommendation engines and behavior prediction methods, we need a sufficient data set to **construct features that are adequate** to predict the behavior of every user.

Dimitri Belenco, Google

WHAT HAPPENS WHEN



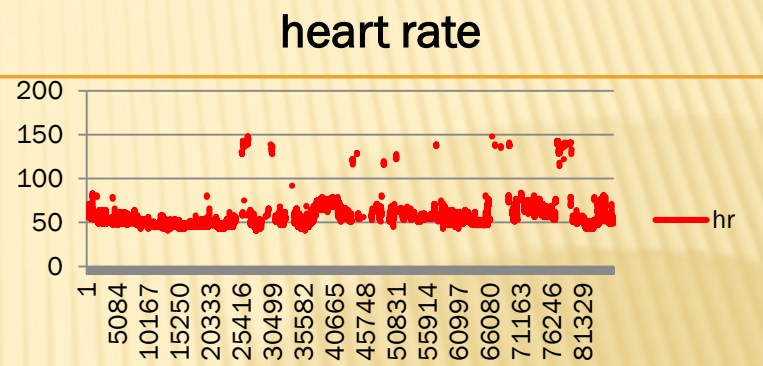
- A standard analysis program receives more data than it can handle?
- Simple example
  - Edge Detection in images with Canny's algorithm
- What will happen if this standard algorithm is given an image that is 1 TB in size?

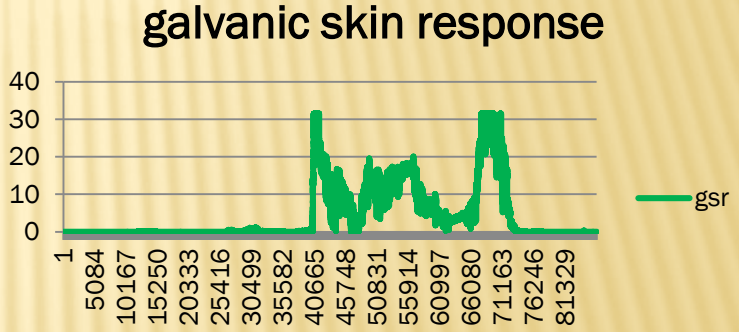
```
sigma = 1.4
f = 'lena_std.jpg.jpg'
img = Image.open(f).convert('L') #grayscale
imgdata = numpy.array(img, dtype = float)
G = ndi.filters.gaussian_filter(imgdata, sigma)
```

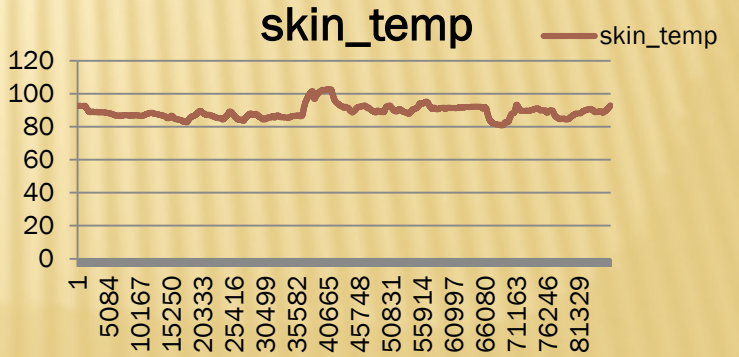
```
sobelout = Image.new('L', img.size)
gradx = numpy.array(sobelout, dtype = float)
grady = numpy.array(sobelout, dtype = float)

sobel_x = [[-1,0,1],
           [-2,0,2],
           [-1,0,1]]
sobel_y = [[-1,-2,-1],
           [0,0,0],
           [1,2,1]]

width = img.size[1]
height = img.size[0]

for x in range(1, width-1):
    for y in range(1, height-1):
        px = (sobel_x[0][0] * G[x-1][y-1]) + (sobel_x[0][1] * G[x][y-1])
             (sobel_x[0][2] * G[x+1][y-1]) + (sobel_x[1][0] * G[x-1][y])
             (sobel_x[1][1] * G[x][y]) + (sobel_x[1][2] * G[x+1][y]) + \
             (sobel_x[2][0] * G[x-1][y+1]) + (sobel_x[2][1] * G[x][y+1])
             (sobel_x[2][2] * G[x+1][y+1])

        py = (sobel_y[0][0] * G[x-1][y-1]) + (sobel_y[0][1] * G[x][y-1])
             (sobel_y[0][2] * G[x+1][y-1]) + (sobel_y[1][0] * G[x-1][y])
             (sobel_y[1][1] * G[x][y]) + (sobel_y[1][2] * G[x+1][y]) + \
             (sobel_y[2][0] * G[x-1][y+1]) + (sobel_y[2][1] * G[x][y+1])
             (sobel_y[2][2] * G[x+1][y+1])
        gradx[x][y] = px
        grady[x][y] = py
sobeloutmag = scipy.hypot(gradx, grady)
sobeloutdir = scipy.arctan2(grady, gradx)
```

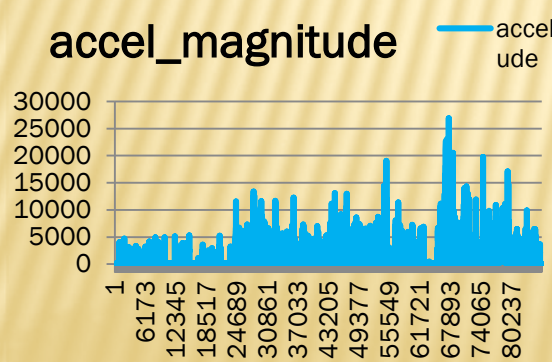
WHAT HAPPENS WHEN

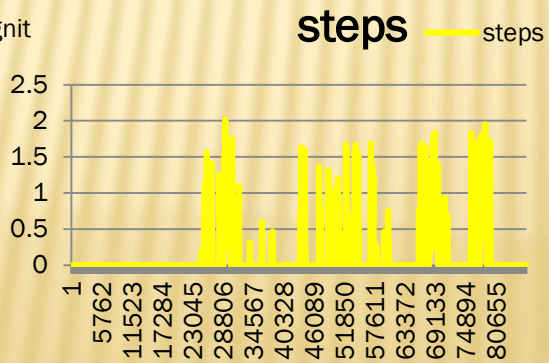
- A database engineer who knows SQL very well is asked to perform potentially complex temporal analysis on data:
  - Find time-partitions in data signifying episodes
  - relationship between HR variability and GSR





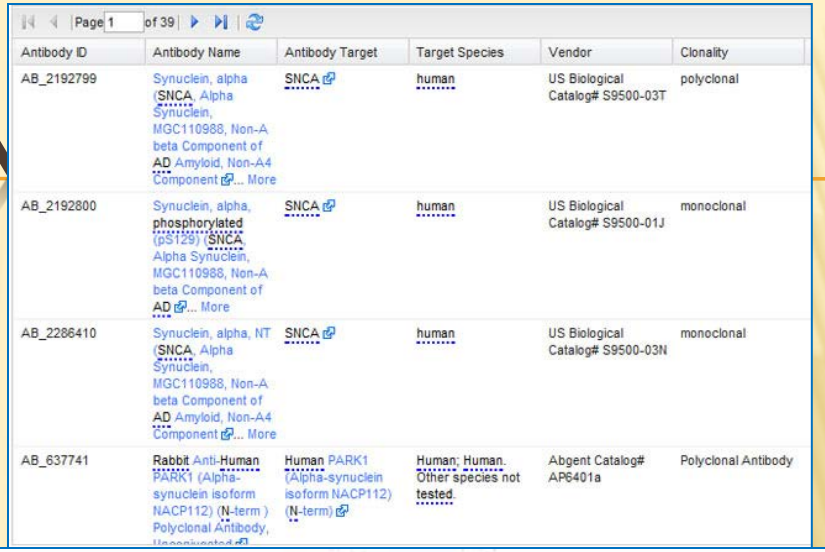






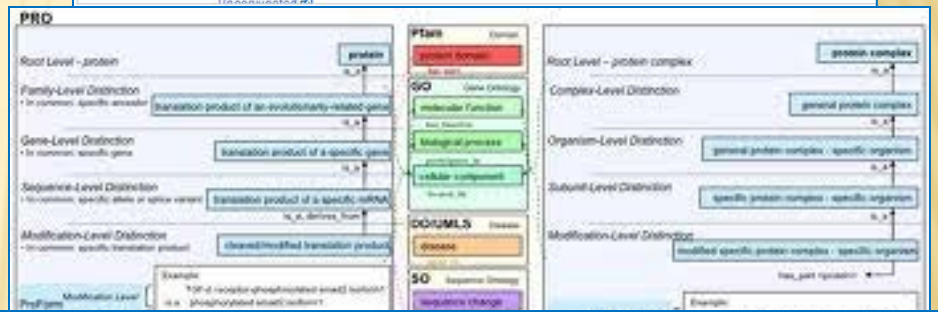
WHAT HAPPENS WHEN

- A biologist in a company wants to know which antibodies (from a database) have actually worked (as per publications) in experiments related to their effects on proteins of a specific family (as per a protein classification scheme)
  - Integration between relational, textual and hierarchical data
- Direct analogs in commercial world



Page 1 of 39

Antibody ID	Antibody Name	Antibody Target	Target Species	Vendor	Clonality
AB_2192799	Synuclein, alpha (SNCA, Alpha Synuclein, MGC110988, Non-A beta Component of AD Amyloid, Non-A4 Component ... More	SNCA	human	US Biological Catalog# S9500-03T	polyclonal
AB_2192800	Synuclein, alpha, phosphorylated (pS129) (SNCA, Alpha Synuclein, MGC110988, Non-A beta Component of AD ... More	SNCA	human	US Biological Catalog# S9500-01J	monoclonal
AB_2286410	Synuclein, alpha, NT (SNCA, Alpha Synuclein, MGC110988, Non-A beta Component of AD Amyloid, Non-A4 Component ... More	SNCA	human	US Biological Catalog# S9500-03N	monoclonal
AB_637741	Rabbit Anti-Human PARK1 (Alpha-synuclein isoform NACP112) (N-term) Polyclonal Antibody,	Human PARK1 (Alpha-synuclein isoform NACP112) (N-term)	Human; Human. Other species not tested.	Abgent Catalog# AP6401a	Polyclonal Antibody



Both the monoclonal anti-Pdx1 antibodies F6 and F109 were shown to recognize the GST-Pdx1 antigen b western blotting (Figure 1A, lane 3, and data not shown). In addition, the antibodies stained a band of 39 kDa, corresponding to the Pdx1 protein, in nuclear extracts from the SV40 large T-antigen transformed mouse insulinoma βTC6 cell line (Figure 1B, lane 2 and 4), but not from the αTC1.9 glucagonoma mouse cell line (Figure 1B, lane 1 and 3). This again reflects the expected expression of Pdx1 in these two islet tumor lines. This Western-blot staining specificity of F6 and F109 was confirmed by pre-absorption studies, where only pre-incubation with GST-Pdx1 fusion protein and not GST-Nkx6.1 was able to abolish the appearance of the 39 kDa band (*data not shown*).

The purified monoclonal mouse anti-Pdx1 antibodies F6 and F109 were further characterized by staining of frozen sections of adult mouse pancreas and fetal pancreas. Both antibodies were demonstrated to specifically stain the nuclei of a sub-population of cells in the endocrine islets of Langerhans, as expected for a Pdx1 reactive antibody (Figure 1, C and D). Specificity was demonstrated as the staining of the cell nuclei was eliminated by pre-absorption of the antibodies with the GST-Pdx1 fusion protein (Figure 1, E and F), but not by pre-absorption with an irrelevant GST fusion protein (Figure 1, G and H). Furthermore, double staining of adult pancreas with a rabbit anti-Pdx1 antiserum (1858.5)[29] demonstrated a near complete overlap of the nuclei recognized by the two antibodies (Figure 2A–C and D–F). On close

THE V-BASED CHARACTERIZATION OF "BIG DATA"

- **Volume**
  - The amount of data
- **Velocity**
  - The speed of data going in and out, and hence the amount of time that is available to process one unit of information
- **Variety**
  - The range of data sources, types, operations within and across types
- **Valence**
  - The complex inter-relatedness of data
- **Veracity**
  - The degree of uncertainty and trustworthiness in the data
- **Variability**
  - Many options or variable interpretations confound analysis

THE VOLUME PROBLEM

- Why is the data volume “big”?
  - A single data item to be processed at a time is inherently large
  - A single data item is not large but the collection to be processed is large
  - The data volume of the collection is not excessively large per se, but the computing operations access the data too many times, perhaps through too many IO operations

THE VELOCITY PROBLEM

- Why is “velocity” a problem?
  - The data rate is faster than the processing rate
  - The data arrives fast and continuously, so only a limited amount of time can be spent on a data element
  - The results must be produced within a bounded (usually small) time
  - Additional complications
    - A computation on the streaming data must be processed by using disk-resident data which is IO limited

THE VARIETY PROBLEM – DATA GENRES

- Structured Data
  - Relations – tables – DBMS: Oracle, PostgreSQL
  - Nested relations – HTML tables – DBMS: Oracle Object Relational
- Unstructured Data
  - Text – the content needs to be retrieved/analyzed independent of its structure – Apache Solr
  - Images and Videos
- Semi-structured Data
  - XML – direct XML, PowerPoint, Word files, … DBMS: MongoDB
  - JSON -- DBMS: MongoDB
  - Graph-structured data – social networks, citation networks, …
    - DBMS: Neo4j, OrientDB
  - Array-structured data – images, matrix operations, simulation outputs
    - RasDaMan, SciDB
- Domain specific Structures
  - HDF, NetCDF – for scientific data
  - Time series
  - …

Stationary or Streaming

There are data management products for each of these genres. Many of them now have a scalable solution

What happens when you have a data integration problem involving all these data genres?

THE VERACITY PROBLEM

- Veracity is not necessarily a “big data” problem
  - “It was raining in San Diego around 9 am today”
    - Pick an arbitrary place in San Diego at 9:02 am
      - Can we say that it was raining there at that time? – perhaps only probabilistically
  - Veracity and “big data”
    - Not every data item or collection comes with an uncertainty model
    - If there a k data sets, each having its own uncertainty model, determining of their “joint” uncertainty model becomes a huge challenge
    - How to produce the “most certain” result of any data query operation?
      - Increases time and space complexity

THE VALENCE PROBLEM

- Definition
  - Degree of interdependency among data items of big data apps that are not embarrassingly parallel
    - Structural dependency
      - nested relations, trees, graphs
    - State-based dependency
      - correlated values in a computation
- Issues
  - A computation step needs all related objects of a given object
    - Possibly filtered by predicates on the properties of and relationships between these objects
  - As data gets larger
    - Connection points get larger and fetching the web of correlated data becomes more expensive

SOME "STANDARD" SOLUTIONS

- For geospatial data
  - ArcGIS, which has a raster management module
    - For large data
      - ArcSDE is used to interface with a relational back-end
      - ArcSDE handles data exchange and transformation
    - The solution for large arrays is through relational database operations
- For image data
  - MATLab, a common software for scientific image analysis
    - Parallel Computing Toolbox provides a parallel for-loop for multicore computers.
    - Distributed Computing Server allows parallel execution in parallel on clusters of machines
    - The blockproc function in Image Processing Toolbox operates on big images by processing them efficiently a block at a time. These computations run in parallel on multiple cores and GPUs when used with Parallel Computing Toolbox

```
in_file = 'myLargeImage.tif';
h = fspecial('gaussian',5,2); % some function
myFun = @(block_struct) imfilter(block_struct.data,h);
block_size = [64 64];
border_size = [2 2];
out_file = 'output.tif'; blockproc(in_file,block_size,myFun, 'BorderSize',border_size,'Destination',out_file);
```

RASDAMAN

http://www.rasdaman.org

- Native array management system
- Basic structure is derived from C/C++
  - Declare a type T

```
typedef struct {
  double rainfall;
  double temperature;
  double windspeedX, windspeedY;
} WeatherData;
```

  - Declare an array type A of type T

```
typedef marray<
  WeatherData, [ 0:664, 0:656, 0:39, 0:77 ]
> WeatherCube;
```

  - Declare a collection (set) of the array type A

```
typedef set< WeatherCube > WeatherSet;
```

  - Query
    - for all simulations conducted where temperature difference exceeds 10

```
select netcdf(
  sqrt(   sq( w.windspeedX )
        + sq( w.windspeedY )
      )
)
from  WeatherSet AS w
where   max_cells( w.temperature )
      - min_cells( w.temperature )
      > 10
```

A short tutorial

RASDAMAN

SOME FUNDAMENTAL STRATEGIES FOR BIG DATA

- Parallelism
  - Pipelined parallelism
  - Partitioned parallelism
  - Inter-query parallelism
  - Intra-query parallelism
- Analytical computation performed close to data
  - Reduce data movement and communication cost
  - Allow analytical engine to inter-operate with data engine
    - Interleave data and analysis operations
- Elasticity
  - Automatically scaling out resources on demand
    - Storage and computational elasticity
- Multiple models of computation within same platform
  - Many modern DBMSs Map-reduce and relational operations

PIPELINED PARALLELISM

- The output of operation A is consumed by another operation B, before A has produced the entire output
- Many machines, each doing one step in a multi-step process

✓ Does not scale up well when:

- the computation does not provide sufficiently long chain to provide a high degree of parallelism:
- relational operators do not produce output until all inputs have been accessed – block, or
- A's computation cost is much higher than that of B

DATA PARTITIONED PARALLELISM

- Many machines performing the same operation on different pieces of data
  - Intraquery,
  - interquery,
  - intraoperation,
  - interoperation

The parallelism behind MapReduce

PARTITIONING

- Partition a data object into segments and distribute them to different processors
- Maximize processing at each individual processor
- Minimize data shipping across boundaries

- Query types:
  - scan a data set,
  - point queries (A = v),
  - range queries (v < A and A < v')

PARTITIONING STRATEGIES

N disks, a relation R

- **Round-robin**: send the *j*-th tuple of R to the disk number *j mod N*
  - Even distribution: good for scanning
  - Not good for equal joins (point queries) and range queries (all disks have to be involved for the search)
- **Range partitioning: partitioning attribute** A, vector $[v_1, ..., v_{n-1}]$
  - send tuple t to disk *j* if t[A] in $[v_{j-1}, v_j]$
  - good for point and range queries on partitioning attributes (using only a few disks, while leaving the others free)
  - Execution skew: distribution may not be even, and all operations occur in one or few partitions (scanning)
- **Hash partitioning**: hash function f(t) in the range of [0, n-1]
  - Send tuple t to disk f(t)
  - good for point queries on partitioning attributes, and sequential scanning if the hash function is even
  - Not good for point queries on non-partitioning attributes and range queries

INTERQUERY VS. INTRAQUERY PARALLELISM

- Inter-query: different queries or transactions execute in parallel
- Intra-query: a single query in parallel on multiple processors
  - Inter-operation: operator tree
  - Intra-operation: parallelize the same operation on different sets of the same data objects
    - Parallel data loading
    - Parallel sorting
    - Parallel join
    - Selection, projection, aggregation

WHAT IS MY BIG DATA PROBLEM? (TAKE-2)

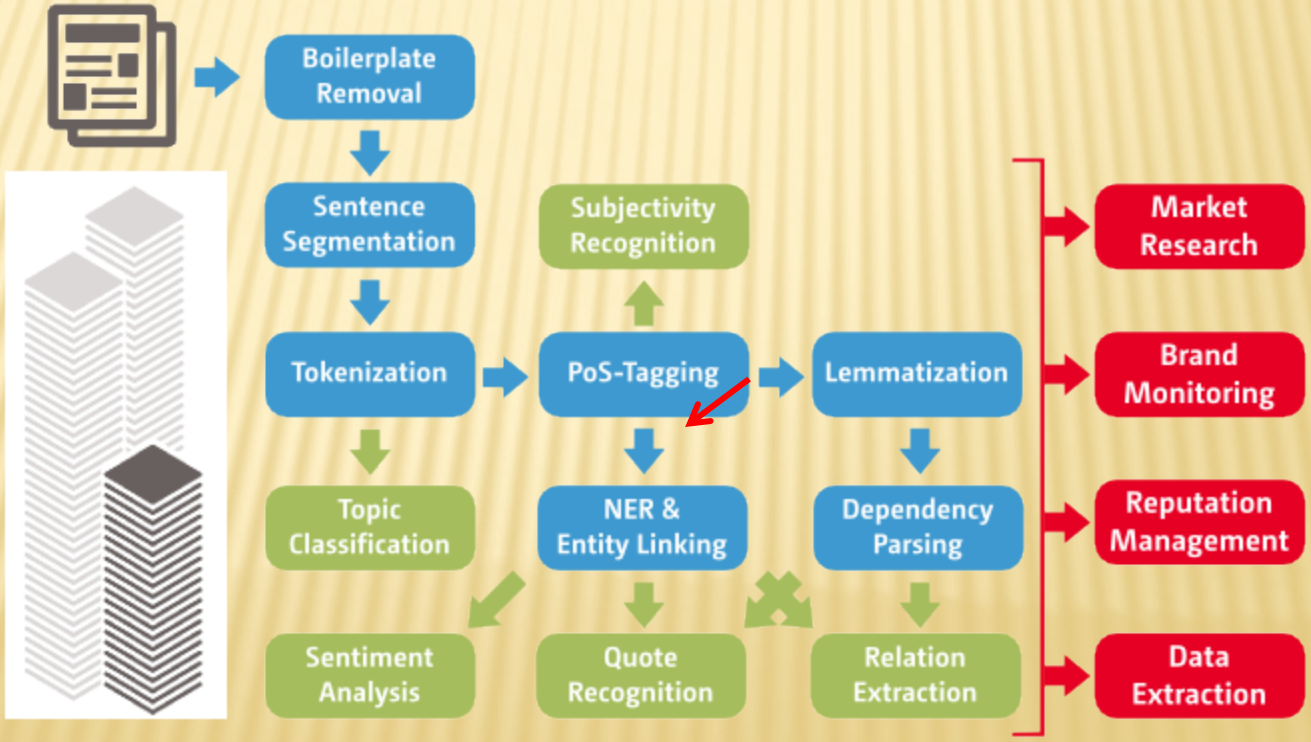
- Some examples (a data-centric view)
  - Large geospatial data
    - Integration of **arrays** with geo-semantics
  - Analyze large activity logs
    - **Temporal text** with **named entities**
  - Retrieval and computation on **graphs**
    - Partitioning and clustering on selected connected subgraphs
  - Text analytics and NLP
    - **Topic modeling** with background knowledge
  - Streaming data analytics
    - **In-stream processing** and **in-memory systems**

ARRAY AS A NATURAL DATA ABSTRACTION

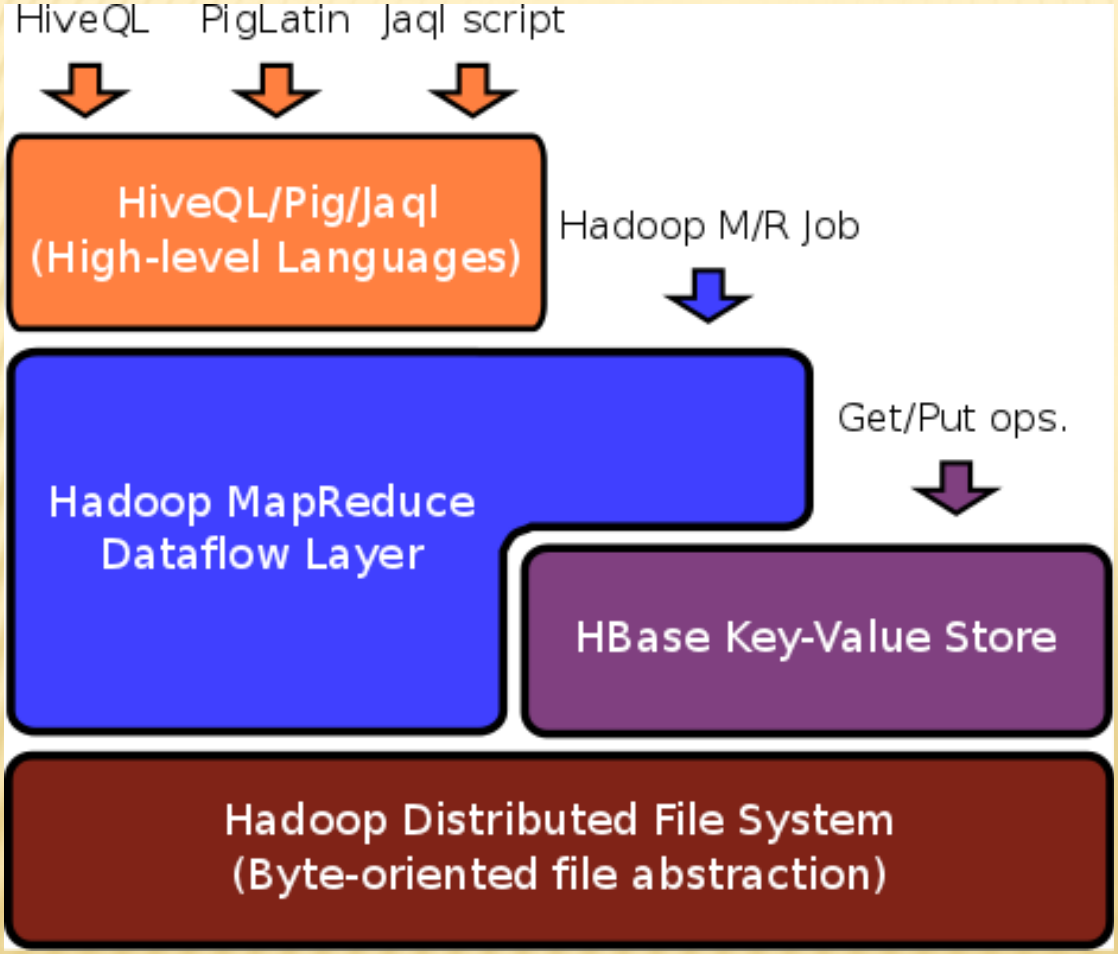
- Many different problem areas and problem solving techniques operate on arrays
  - Spatial processing and simulations
  - Image/video processing and computer vision
    - Multi-spectral images
  - Biological sequence analysis
  - Linear algebra problems
  - Some graph problems
  - Time-series data analysis
    - Financial analysis

"BIG DATA" IN NATURAL LANGUAGE PROCESSING

- Unstructured data management
  - Pig in Natural Language Processing (courtesy neofone.de)



THE OPEN SOURCE BIG DATA STACK



Courtesy: AsterixDB group

NAMED ENTITY RECOGNITION

- "Four years after becoming part of Disney, Marvel has emerged as a creative juggernaut for the company as we continue to introduce its rich universe of characters into a variety of businesses including movies, television, theme parks, and consumer products" – *Disney Annual Report*, 2013
- Which of the several possible entities does a surface term refer to?
  - Computation of 4 probabilities
    - P(entity), P(surface term), P(surface term | entity), P(entity | surface term)
    - P(entity | surface form) = count(entity, surface form) / count(surface form)
    - Estimate using the number of Wikipedia links

FROM NER TO PIG LATIN

```
parsed = LOAD 'enwiki-20111207-pages-articles.xml',
         USING pignlproc.storage.ParsingWikipediaLoader('en')
         AS (title, id, pageUrl, text, redirect, links, headers, paragraphs);
noredirect = FILTER parsed BY redirect  IS NOT NULL;
projected = FOREACH noredirect GENERATE title, text, links, paragraphs;
sentences = FOREACH projected GENERATE title,
flatten(pignlproc.evaluation.SentencesWithLink( text, links, paragraphs));
stored = FOREACH sentences GENERATE title, sentenceOrder, linkTarget, linkBegin, linkEnd,
sentence; ordered = ORDER stored BY linkTarget ASC, title ASC, sentenceOrder ASC
STORE ordered INTO '$OUTPUT/$LANG/sentences_with_links'
```

```
sentences = LOAD '$INPUT/$LANG/sentences_with_links'
         AS (title: chararray, sentenceOrder: int, linkTarget: chararray, linkBegin: int, linkEnd: i
wikiuri_types = LOAD '$INPUT/$LANG/wikiuri_to_types' AS (wikiuri: chararray, typeuri: chararra
type_names = LOAD '$TYPE_NAMES' AS (typeuri: chararray, typename: chararray);
-- Perform successive joins to find the OpenNLP typename of the linkTarget
joined =
joined_projected = FOREACH joined GENERATE wikiuri, typename;
joined2 = JOIN joined_projected BY wikiuri, sentences BY linkTarget;
result = FOREACH joined2 GENERATE title, sentenceOrder, typename, linkBegin, linkEnd, sente
```

RELATIONAL DATA – ROW AND COLUMN STORES

- A modern relational database system
  - Designed for scale-out scalability
  - Massively parallel SQL engine
  - Distributed storage – takes advantage of different storage layers
  - Inter-operates with Hadoop
    - Allows users to perform some computations on Hadoop
  - Designed as a column store or offers a hybrid of row and column stores

ROW STORES

- Row based tables have advantages in the following circumstances:
  - The application needs to only process a single record at one time (many selects and/or updates of single records).
  - The application typically needs to access a complete record (or row).
  - The columns contain mainly distinct values so that the compression rate would be low.
  - Neither aggregations nor fast searching are required.
  - The table has a small number of rows (e. g. configuration tables).

COLUMN STORES

- Column-based tables have advantages in the following circumstances:
  - Calculations are typically executed on single or a few columns only.
  - The table is searched based on values of a few columns.
  - The table has a large number of columns, and more can be added.
  - The table has a large number of rows and columnar operations are required (aggregate, scan etc.)
  - High compression rates can be achieved because the majority of the columns contain only few distinct values (compared to number of rows).

HANDLING GENOMIC DATA WITH VERTICA

BACKUP SLIDES

MIKE STONEBRAKER'S VIEWPOINT – 1

- The "big data" problem can be thought of as different combinations of data characteristics and analytical needs problems.
- **Big Volume – Little Analytics**
  - Well addressed by database and data warehouse crowd who are pretty good at SQL analytics on
    - Hundreds of nodes and Petabytes of data
- **Big Data – Big Analytics**
  - Complex math operations (machine learning, clustering, trend detection, ….)
    - the world of the "quants" and data miners
    - Mostly specified as linear algebra on matrix data
  - A dozen or so common 'inner loops'
    - Matrix multiplication, QR decomposition, SVD decomposition, Linear regression, …

MIKE STONEBRAKER'S VIEWPOINT – 2

- Big Velocity Data
  - Big pattern - little state (electronic trading)
    - Find the occurrence of 'IBM down' followed within 100 msec by a 'MSFT down'
    - Complex event processing (CEP) is focused on this problem
      - Patterns in a fire hose
  - Big state - little pattern
    - For every security, assemble my real-time global position
    - And alert me if my exposure is greater than X
    - Looks like high performance OLTP where we want to update a database at very high speed
  - Big state – big pattern?

MIKE STONEBRAKER'S VIEWPOINT – 3

- Big Variety
  - A typical enterprise has many (sometimes thousands of) operational systems
    - Only a few get into a structured, managed database
    - What about the rest?
  - For the remaining data
    - One has to look at and be scalable to 1000s of sites
    - Deal with incomplete, conflicting, and incorrect data
  - Be incremental because the task is never done