# Big Data Analytics

## Parallel Data Processing Beyond Map and Reduce

**Volker Markl**

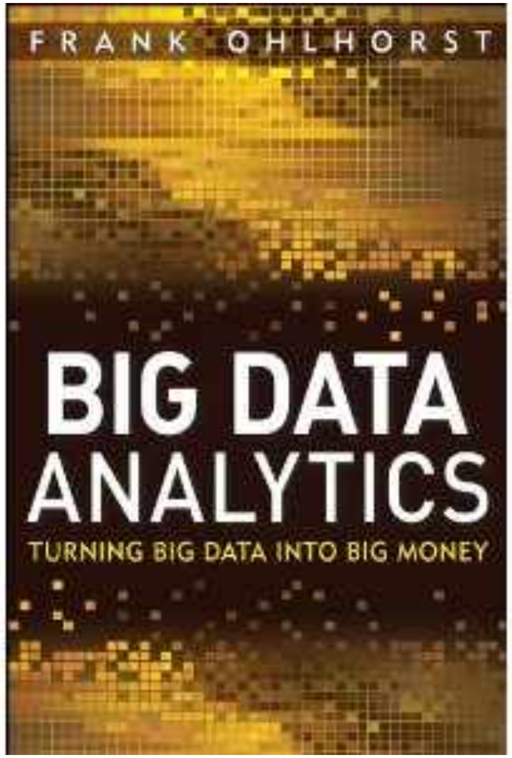
Database Systems and Information Management Lab
Technische Universität Berlin

www.dima.tu-berlin.de

# About Big Data Analytics


Source: McKinsey


Source: Information Week

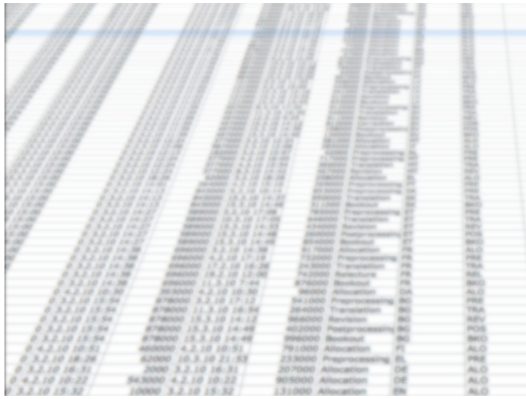



Source: Amazon

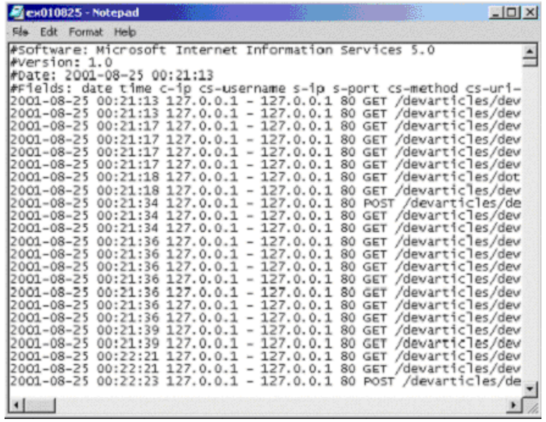

Source: NCSU

# Agenda

- Big Data Analytics
  - Motivating Big Data Analytics
  - Examples
  - Chances and Applications
  - Challenges
  - Technologies
- Parallel Data Processing beyond map and reduce
  - Limits of Parallel Data Procesing
  - Map/Reduce
  - Stratosphere
  - Nephele
  - PACTS



Transactional Data
at „Webscale“

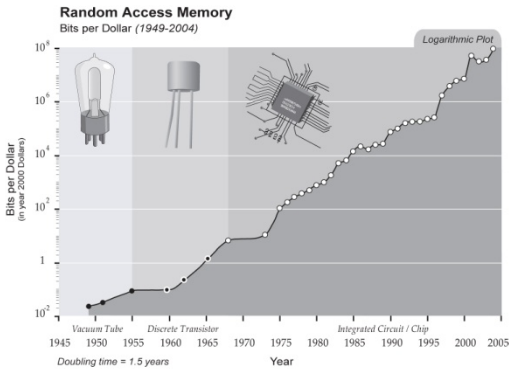


```
ex010825 - Notepad
File Edit Format Help
#Software: Microsoft Internet Information Services 5.0
#Version: 1.0
#Date: 2001-08-25 00:21:13
#Fields: date time c-ip cs-username s-ip s-port cs-method cs-uri-
2001-08-25 00:21:13 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:13 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:17 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:17 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:17 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:17 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:18 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dot
2001-08-25 00:21:18 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:34 127.0.0.1 - 127.0.0.1 80 POST /devarticles/de
2001-08-25 00:21:34 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:34 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:36 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:39 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:21:39 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:22:21 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:22:21 127.0.0.1 - 127.0.0.1 80 GET /devarticles/dev
2001-08-25 00:22:23 127.0.0.1 - 127.0.0.1 80 POST /devarticles/de
```

Web Logfiles

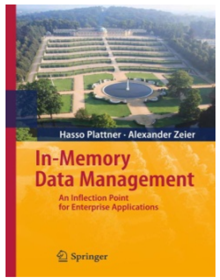
Linked Open Data

**These data sets will fit into main memory soon!**

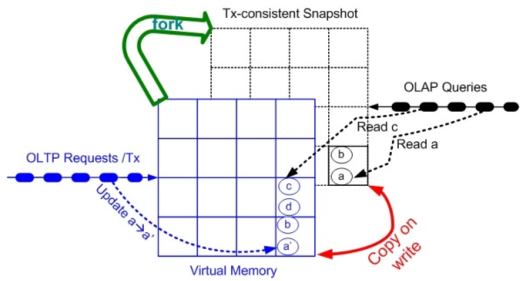




IBM ISAO



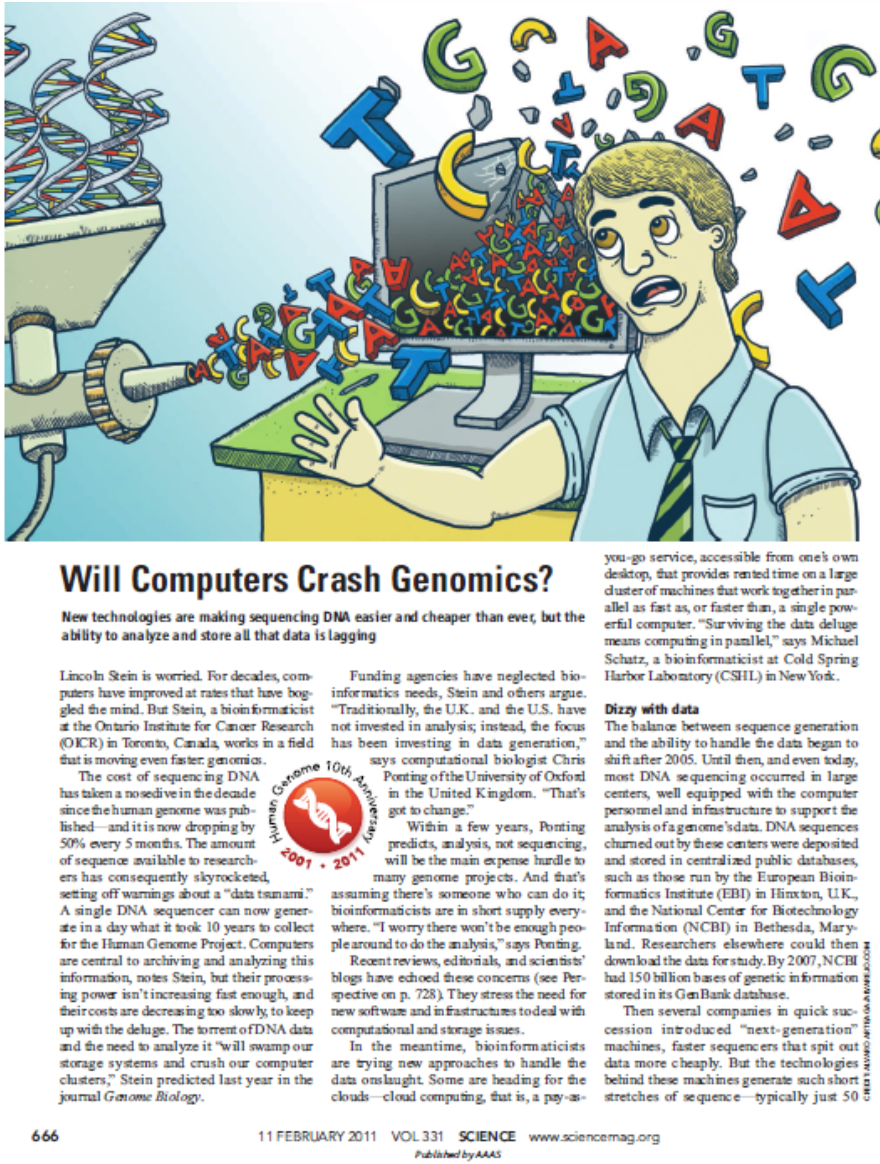

SAP HANA




TUM Hyper

**Who needs Hadoop or NoSQL for these?**

# Big Data for Real



## Will Computers Crash Genomics?

**New technologies are making sequencing DNA easier and cheaper than ever, but the ability to analyze and store all that data is lagging**

Lincoln Stein is worried. For decades, computers have improved at rates that have boggled the mind. But Stein, a bioinformaticist at the Ontario Institute for Cancer Research (OICR) in Toronto, Canada, works in a field that is moving even faster: genomics.

Human Genome 10th Anniversary 2001 • 2011

The cost of sequencing DNA has taken a nosedive in the decade since the human genome was published—and it is now dropping by 50% every 5 months. The amount of sequence available to researchers has consequently skyrocketed, setting off warnings about a "data tsunami." A single DNA sequencer can now generate in a day what it took 10 years to collect for the Human Genome Project. Computers are central to archiving and analyzing this information, notes Stein, but their processing power isn't increasing fast enough, and their costs are decreasing too slowly, to keep up with the deluge. The torrent of DNA data and the need to analyze it "will swamp our storage systems and crush our computer clusters," Stein predicted last year in the journal *Genome Biology*.

Funding agencies have neglected bioinformatics needs, Stein and others argue. "Traditionally, the U.K. and the U.S. have not invested in analysis; instead, the focus has been investing in data generation," says computational biologist Chris Ponting of the University of Oxford in the United Kingdom. "That's got to change."

Within a few years, Ponting predicts, analysis, not sequencing, will be the main expense hurdle to many genome projects. And that's assuming there's someone who can do it; bioinformaticists are in short supply everywhere. "I worry there won't be enough people around to do the analysis," says Ponting.

Recent reviews, editorials, and scientists' blogs have echoed these concerns (see Perspective on p. 728). They stress the need for new software and infrastructures to deal with computational and storage issues.

In the meantime, bioinformaticists are trying new approaches to handle the data onslaught. Some are heading for the clouds—cloud computing, that is, a pay-as-you-go service, accessible from one's own desktop, that provides rented time on a large cluster of machines that work together in parallel as fast as, or faster than, a single powerful computer. "Surviving the data deluge means computing in parallel," says Michael Schatz, a bioinformaticist at Cold Spring Harbor Laboratory (CSHL) in New York.

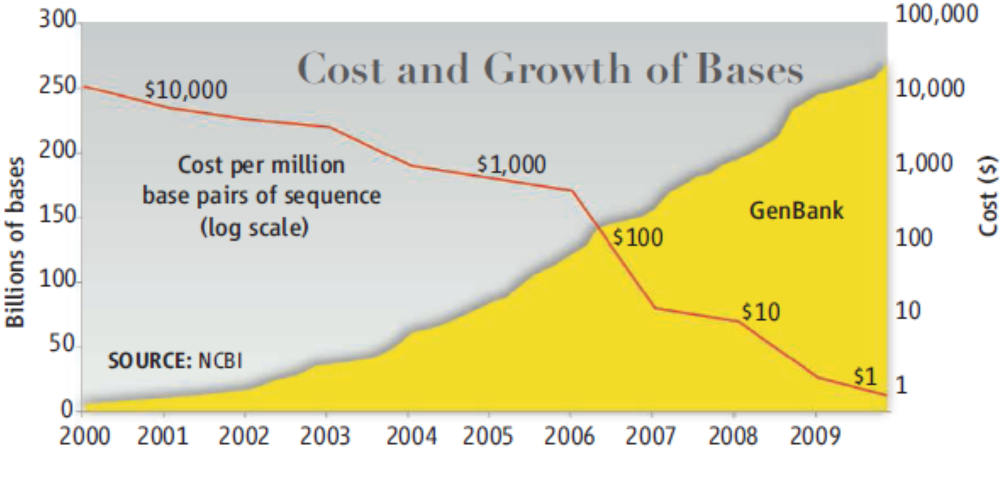
### Dizzy with data

The balance between sequence generation and the ability to handle the data began to shift after 2005. Until then, and even today, most DNA sequencing occurred in large centers, well equipped with the computer personnel and infrastructure to support the analysis of a genome's data. DNA sequences churned out by these centers were deposited and stored in centralized public databases, such as those run by the European Bioinformatics Institute (EBI) in Hinxton, U.K., and the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Researchers elsewhere could then download the data for study. By 2007, NCBI had 150 billion bases of genetic information stored in its GenBank database.

Then several companies in quick succession introduced "next-generation" machines, faster sequencers that spit out data more cheaply. But the technologies behind these machines generate such short stretches of sequence—typically just 50

666 11 FEBRUARY 2011 VOL 331 SCIENCE www.sciencemag.org
Published by AAAS

**It is cheaper to resequence than to store genome data!**




SCIENCE 11 FEBRUARY 2011
VOL 331, ISSUE 6018, PAGES 639-806

# Other Examples of Really Big Data Sets



Smart Grids

Video Streams

**EXABYTE**
(1,152,921,504,606,846,976 BYTES; $2^{60}$)
approx. 1,000,000,000,000,000,000 or $10^{18}$

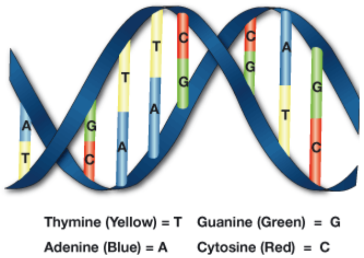
Web Archive

5 EXABYTES: ALL WORDS EVER SPOKEN BY HUMAN BEINGS

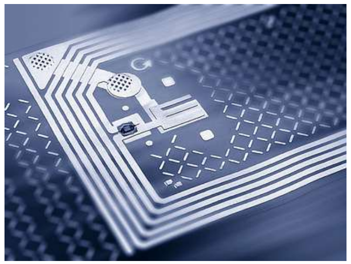
Sensor Data

**ZETTABYTE**
(1,180,591,620,717,411,303,424 BYTES; $2^{70}$)
approx. 1,000,000,000,000,000,000,000 or $10^{21}$

Audio Streams




Genome Data



RFID Data

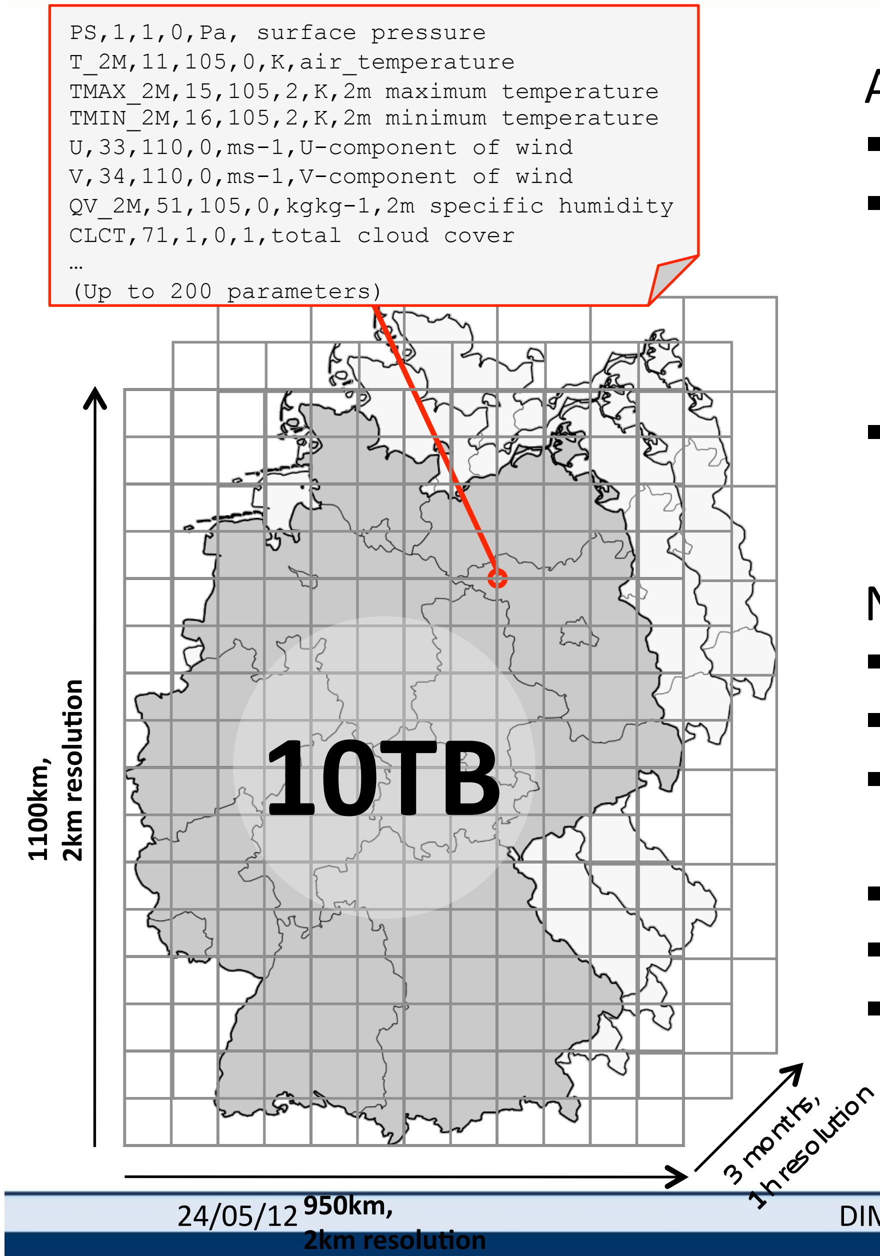
Size
Structure/Representation
Uncertainty
„Cleanliness“
Generating Process
etc.

**Data**

Selection/Grouping
Relational Operators
Information Extraction & Integration
Data Mining
Predictive Models
etc.

**Query**

# Big Data Example: Climate Data Analysis




## Analysis Tasks on Climate Data Sets

- Validate climate models
- Locate „hot-spots“ in climate models
  - Monsoon
  - Drought
  - Flooding
- Compare climate models
  - Based on different parameter settings

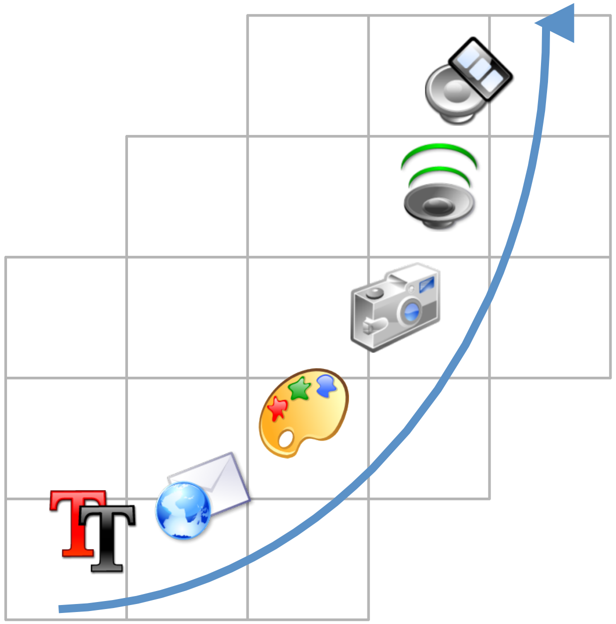
## Necessary Data Processing Operations

- Filter
- Aggregation (sliding window)
- Join

- Multi-dimensional sliding-window operations
- Geospatial/Temporal joins
- Uncertainty

# Motivating Data Parallel Programming Models

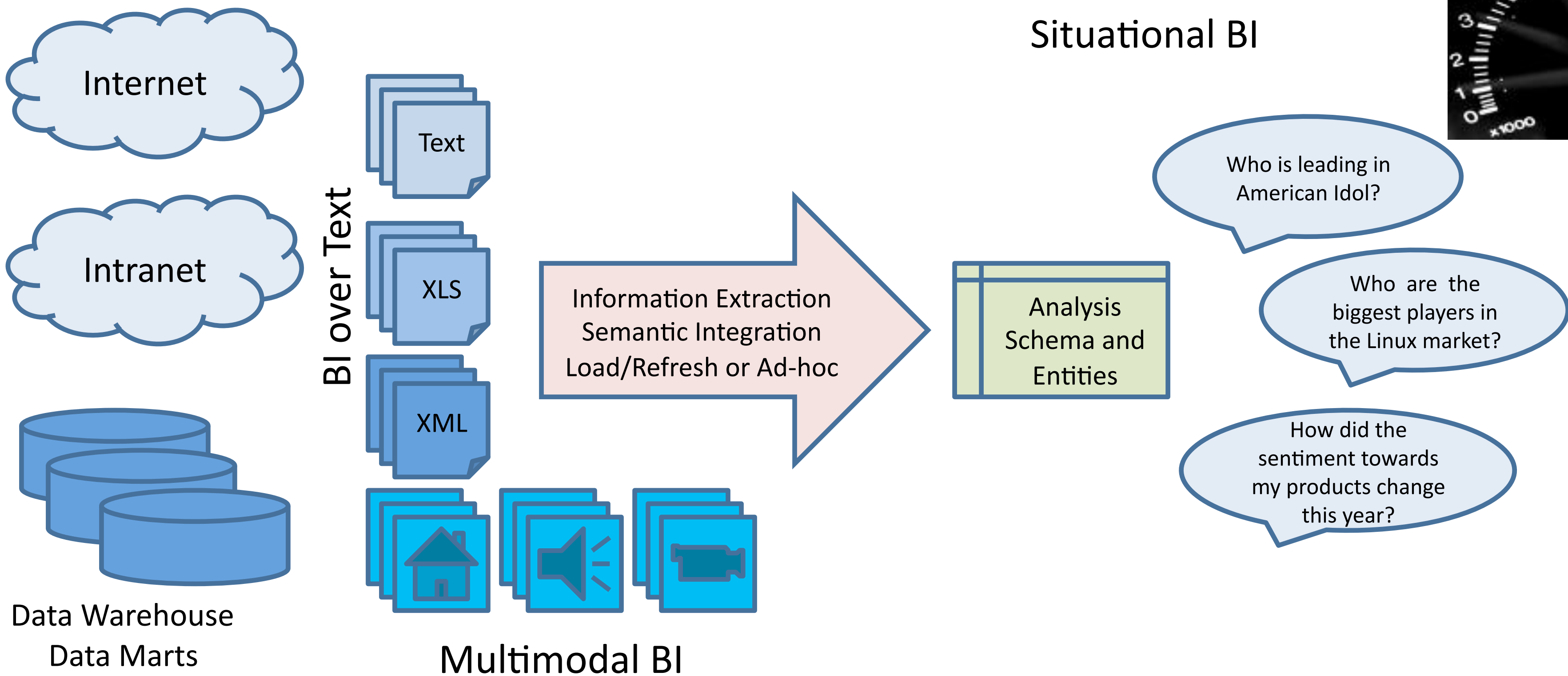
- Need for data parallel programming
  - Increase of data complexity
  - Increase of query complexity
  - Moore's Law: ManyCore and Cluster Computing
  - Scale-up no longer possible
  - ➔ **Scale-out is the name of the game**



- Parallel programming is not easy
  - (Network) Communication
  - Concurrent programming: Divide & Conquer
  - Synchronization as bottleneck
  - Fault tolerance

- Programming models ease development of parallel tasks
  - Abstractions hide the gory details
  - Automatic adaption to hardware
    - Parallelization and Optimization
  - Beware: Data flow and control flow dependencies!

# Vision for a Next Generation of Business Intelligence





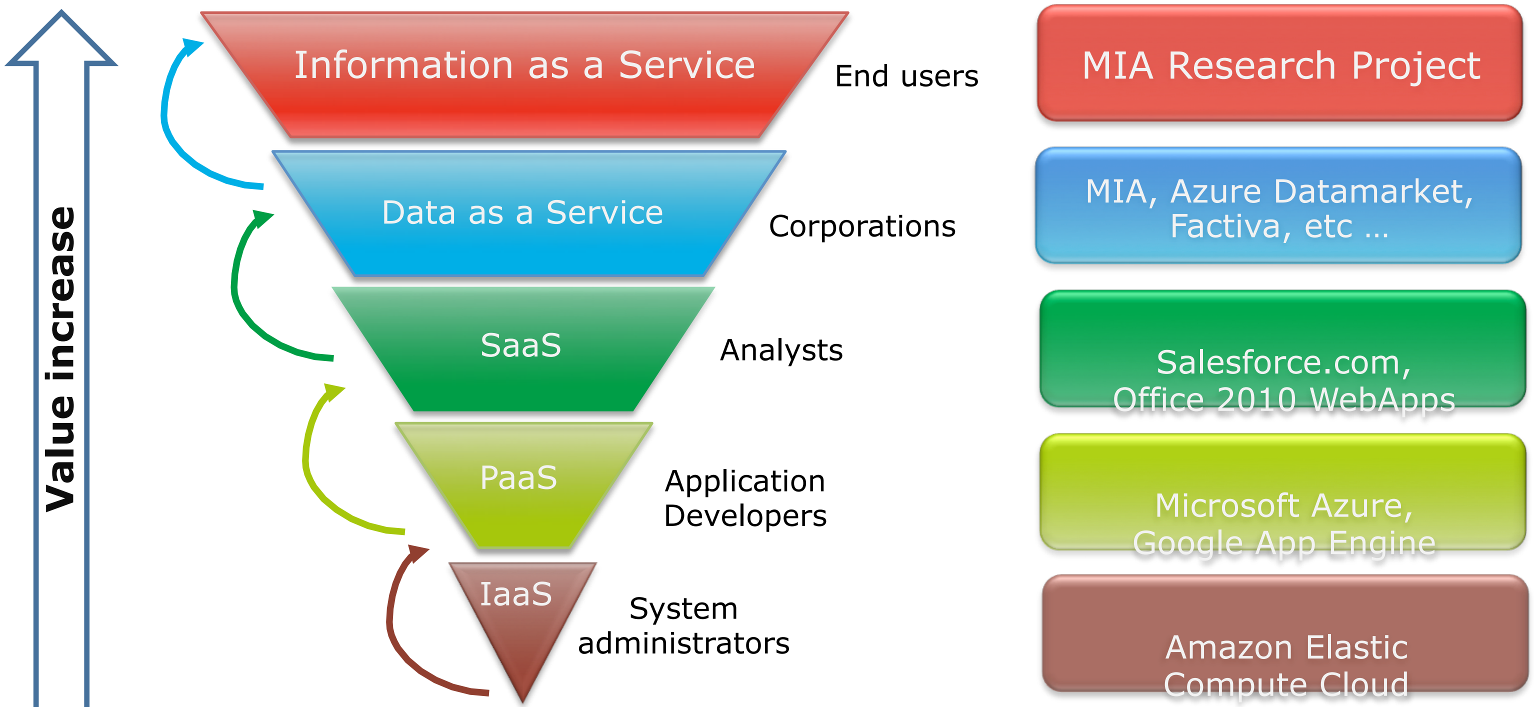
The *next generation of Business Intelligence (NGBI)* will correlate data warehouses with text and other modalities from web services of information providers, corporate Intranets and the Internet

# A New Chance: Information Marketplaces

**A major new trend in information processing will be the trading of original and enriched data, effectively creating an information economy.**

„When hardware became commoditized, software was valuable.
Now that software is being commoditized, data is valuable.“ (TIM O‘REILLY)

„The important question isn't who owns the data. Ultimately, we all do.
A better question is, who owns the means of analysis?“ (A. CROLL, MASHABLE, 2011)

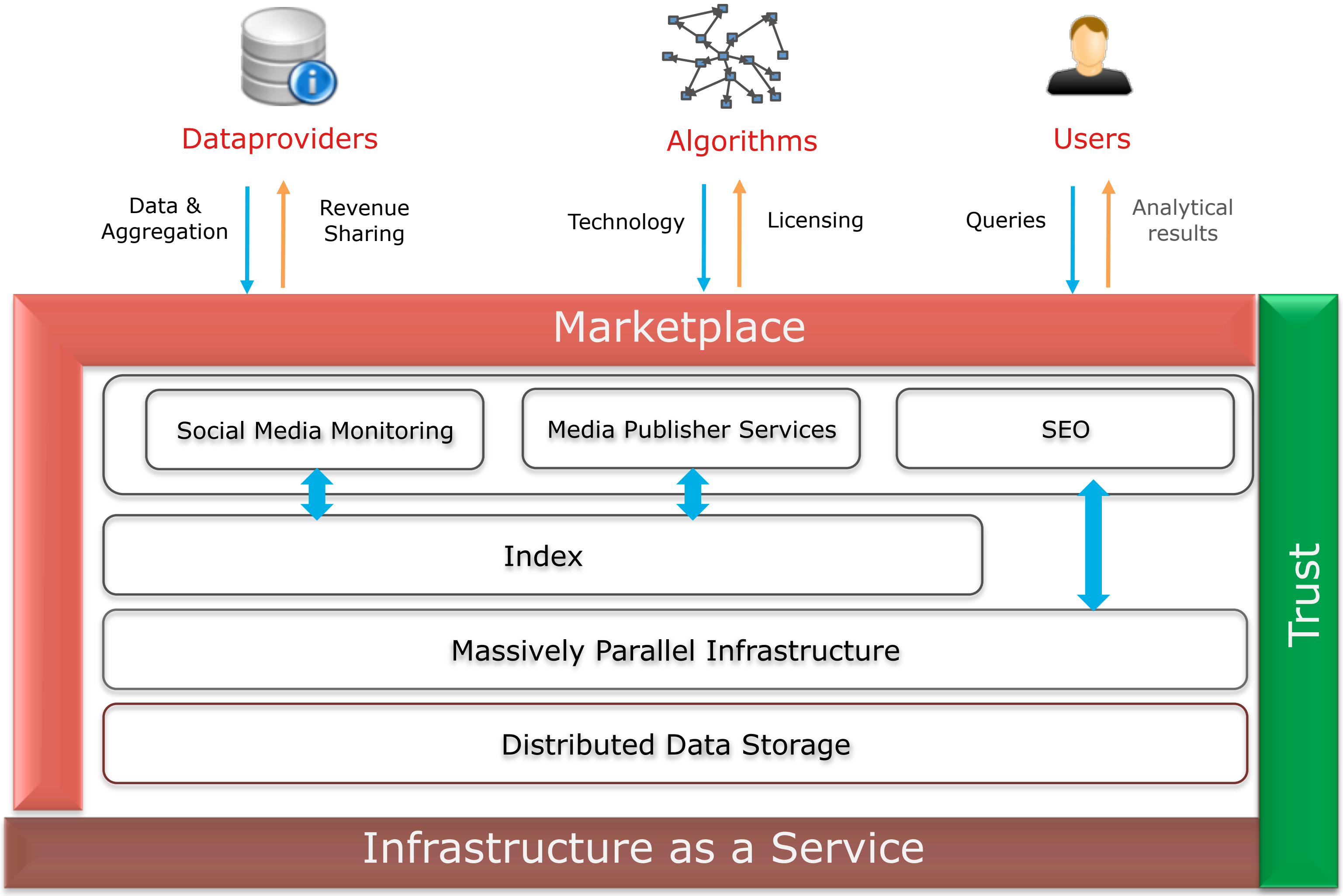



Cloud-Computing Stack

The Market situation in Cloud-Computing

# A possible Architecture for an Information Marketplace





# Further Applications

Healthcare

Home Automation

Water Management

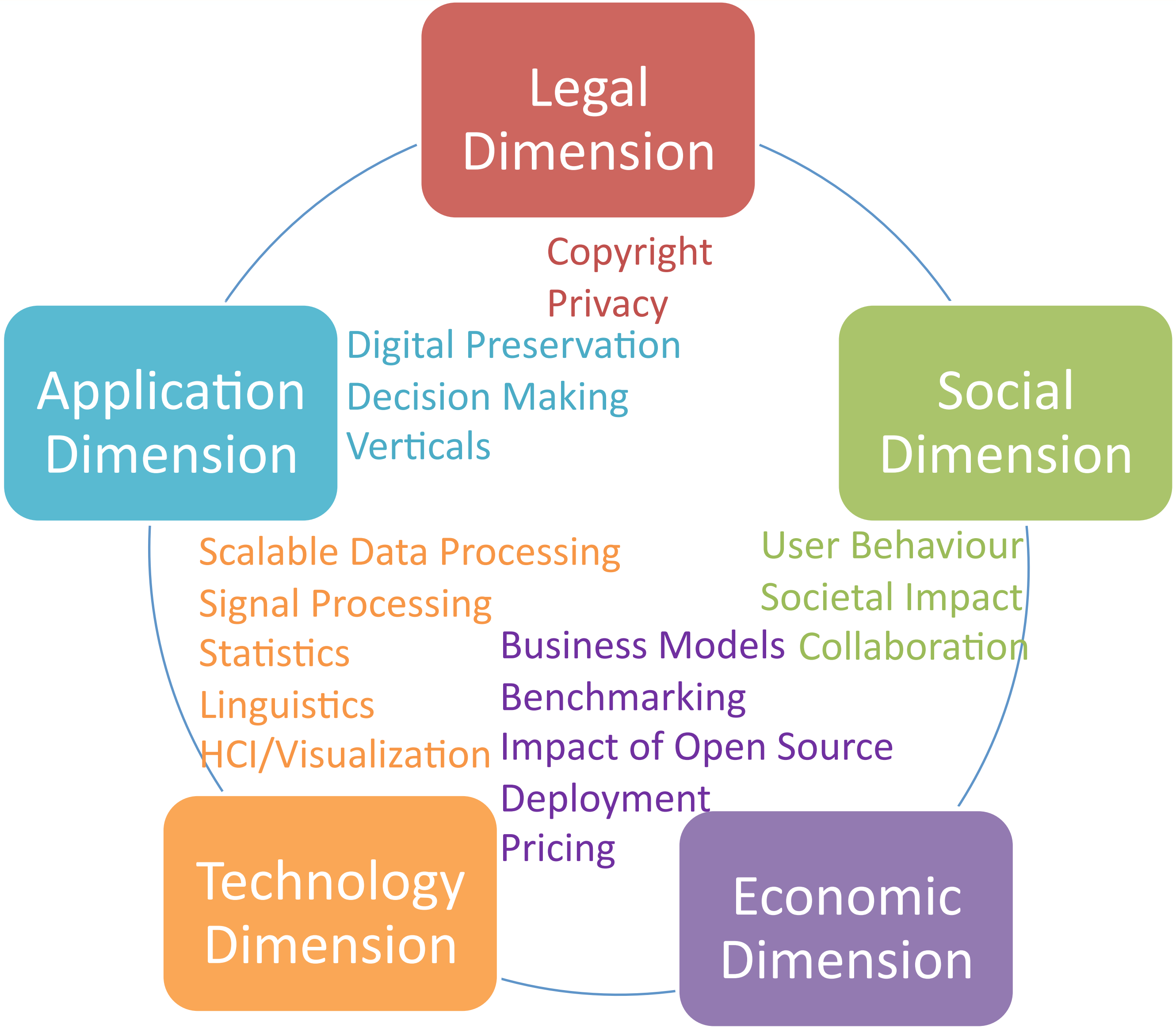
Lifecycle Management

Energy Management


Sales/Marketing

Traffic Management

# Some Challenges



# Some Solutions

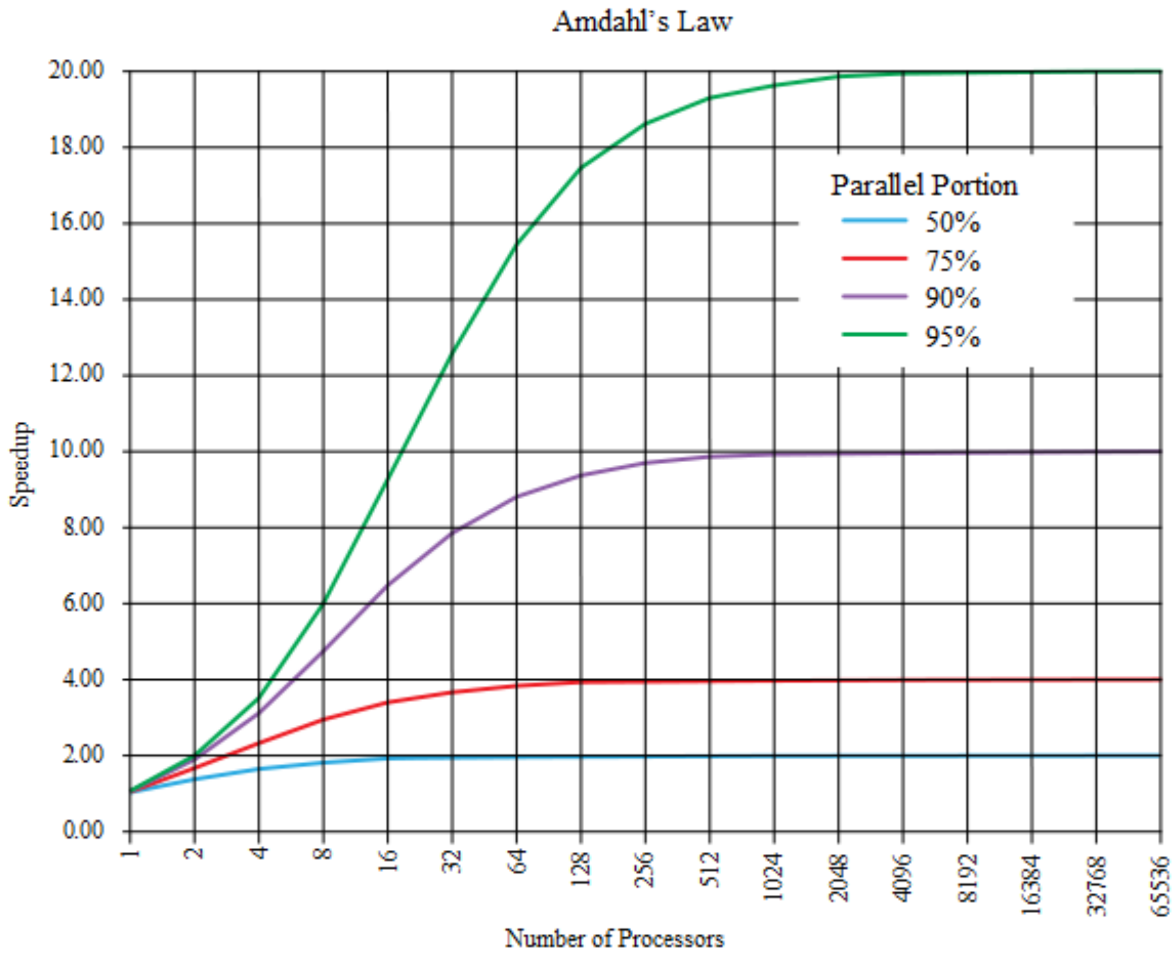
- Parallel Data Processing
- Parallel Data Management: ParStream et al.
- Map/Reduce: Hadoop, Stratosphere et al.

# Parallel Speedup

- The Speedup is defined as: $S_p = {}^{T_1}/_{T_p}$
  - $T_1$: runtime of the sequential program
  - $T_p$: runtime of the parallel program on p processors

- Ahmdal's Law: „The maximal speedup is determined by the non-parallelizable part of a program" :
  - $S_{max} = \frac{1}{(1-f)+{}^{f}/_{p}}$ f: fraction of the program that can be parallelized
  - → Ideal speedup: S=p for f=1.0 (linear speedup)
  - However - since usually f<1.0 -, S is bound by a constant ! (e.g. ~10 for f=0.9)
  - → Fixed problems can only be parallelized to a certain degree!

# Parallel Speedup



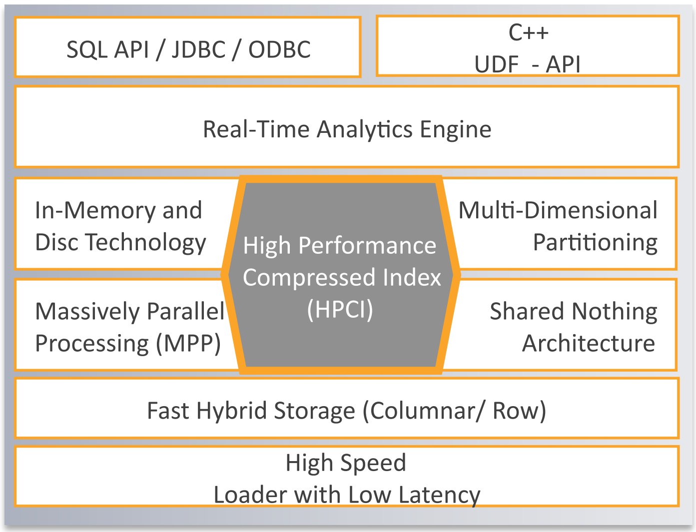
# ParStream Big Data Analytics Platform

## ParStream provides the unique combination of REAL-TIME – LOW-LATENCY – HIGH THROUGHPUT

ParStream's key advantages:

- Unique Highly compressed Bitmap Index which can be analyzed in compressed form (patent application filed)
- Real-time analytics through Massively Parallel Processing (MPP)
- Very high throughput due to massively reduced workload (no decompression, small index, efficient algorithms)
- Low-Latency through continuous import of new data without slowing down analytics
- Columnar data / index allows very flexible analytics (multi-column, multi-value)
- Specialized data / index types and algorithms
- Shared Nothing Architecture

PARSTREAM PARALLEL ARCHITECTURE




PARSTREAM INDEX ARCHITECTURE

Parallel Search within Compressed Index

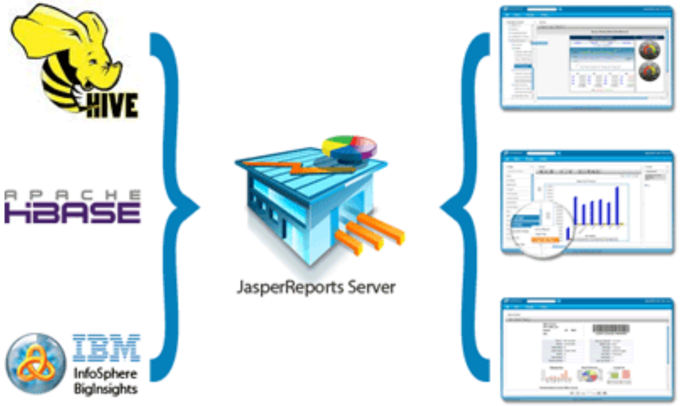
Big Data Analytics is a game changer in every industry
and is a huge market opportunity

**All Industries**

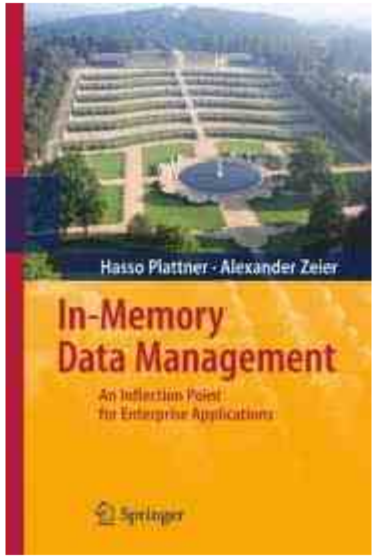
**Many Applications**

| eCommerce Services | Social Networks | Telco | Finance | Energy Oil and Gas | Many More |
|---|---|---|---|---|---|
| ▪ Facetted Search<br>▪ Web analytics<br>▪ SEO-analytics<br>▪ Online-Advertising | ▪ Ad serving<br>▪ Profiling<br>▪ Targeting | ▪ Customer attrition prevention<br>▪ Network monitoring<br>▪ Targeting<br>▪ Prepaid account mgmt | ▪ Trend analysis<br>▪ Fraud detection<br>▪ Automatic trading<br>▪ Risk analysis | ▪ Smart metering<br>▪ Smart grids<br>▪ Wind parks<br>▪ Mining<br>▪ Solar Panels | ▪ Production<br>▪ Mining<br>▪ M2M<br>▪ Sensors<br>▪ Genetics<br>▪ Intelligence<br>▪ Weather |

# Further Players




Sybase IQ™

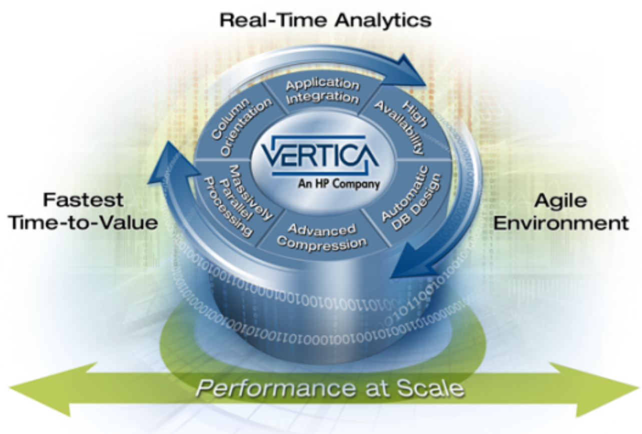



sas ANALYTICS

SPSS STATISTICAL DATA ANALYSIS


Greenplum

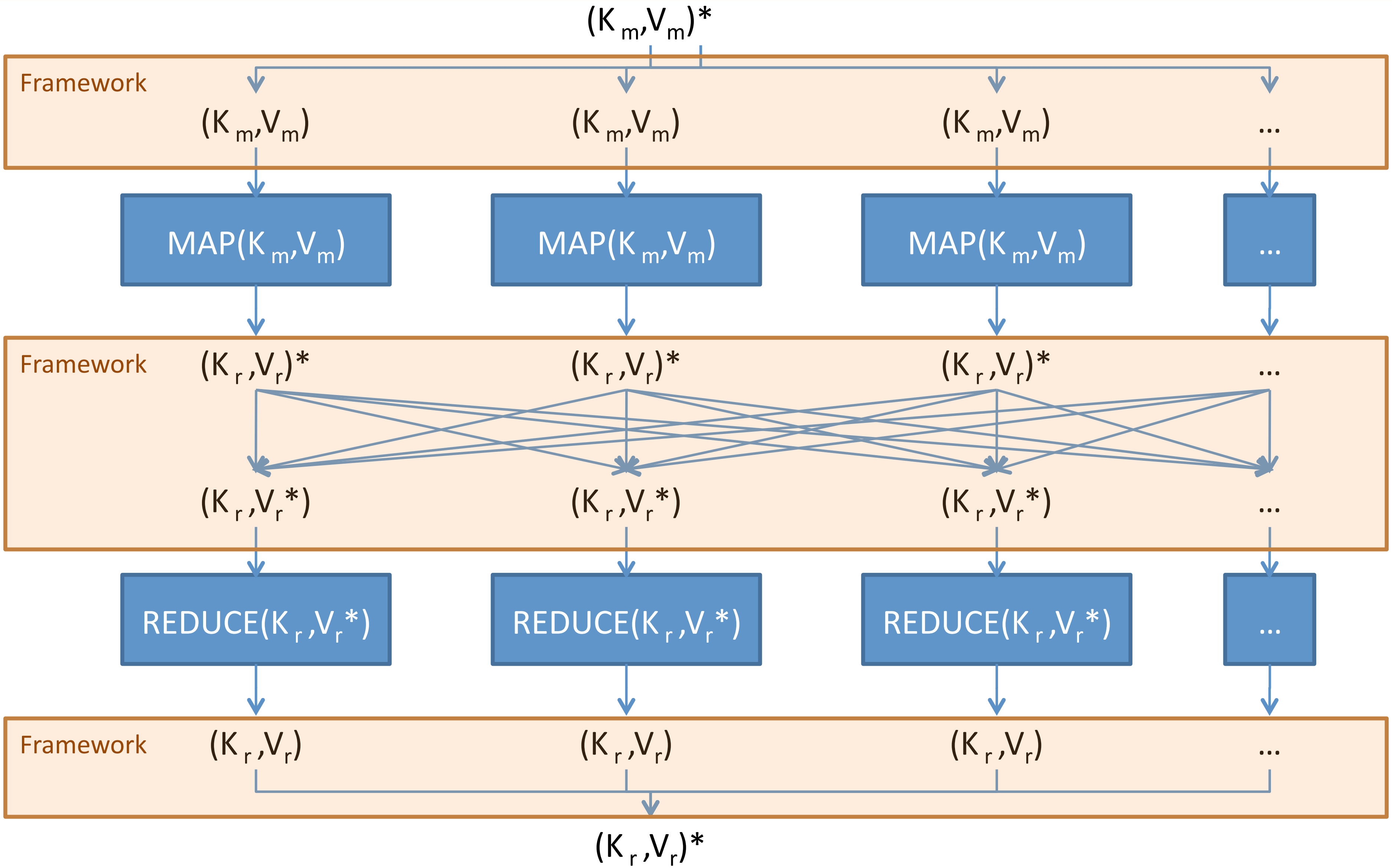
aster data
big data. fast insights.



# What is Map/Reduce?

- Programming model for data-intensive programming
  - well-suited for large scale parallel execution
  - automatic parallelization & distribution of data and computational logic
  - easy to extend with fault tolerance schemes
  - clean abstraction for programmers
- Based on functional programming
  - treats computation as the evaluation of mathematical functions and avoids state and mutable data
  - no changes of states (no side effects)
  - output value of a function depends only on its arguments
- `Map` and `Reduce` are higher-order functions
  - take user-defined functions as argument
  - return a function as result
  - to define a map/reduce job, the user implements the two functions m and r

# Data Flow in Map/Reduce



# Map Reduce Illustrated (1)

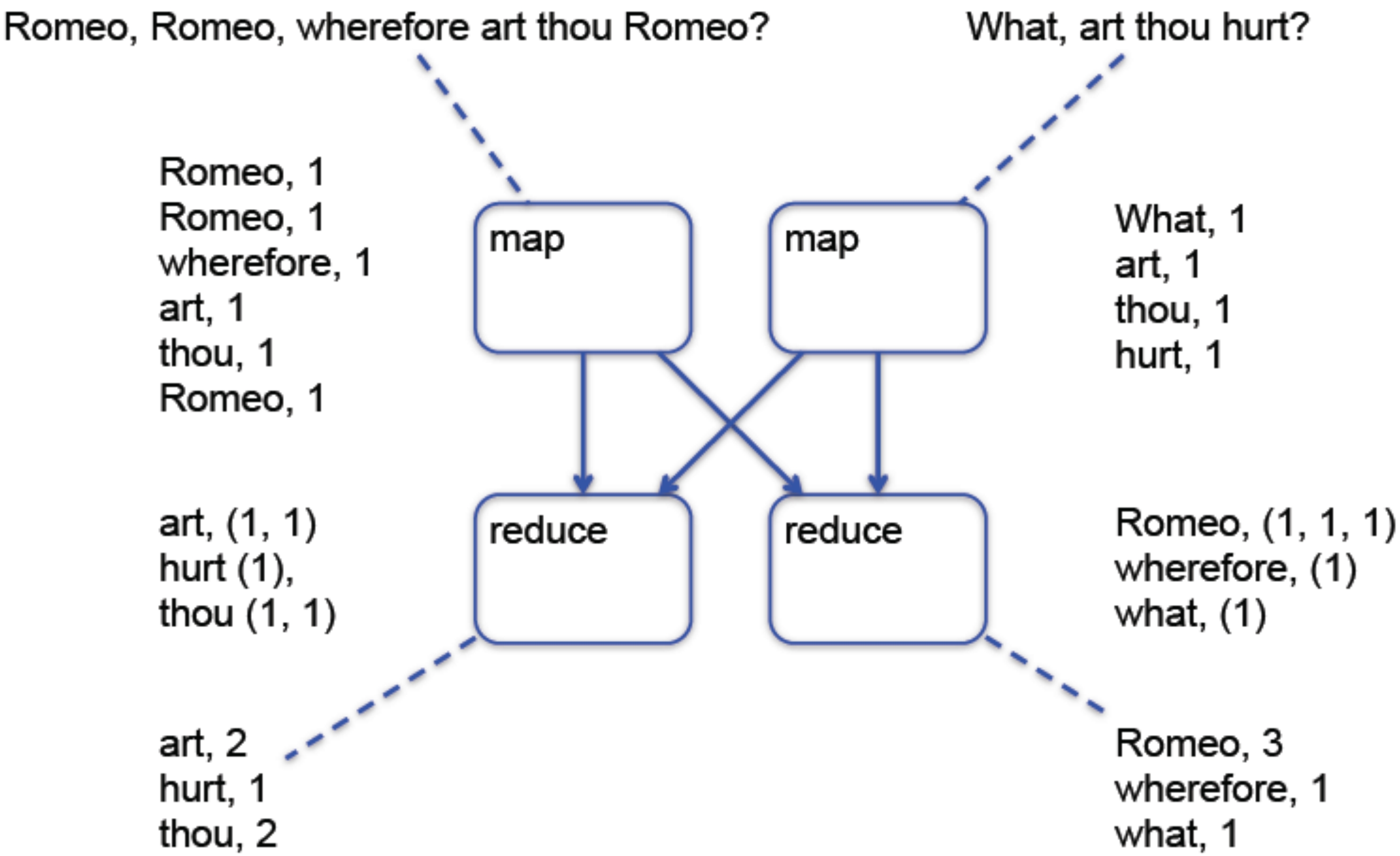
- *Problem*: Counting words in a parallel fashion
  - How many times different words appear in a set of files
  - **juliet.txt**: Romeo, Romeo, wherefore art thou Romeo?
  - **benvolio.txt:** What, art thou hurt?
  - Expected output: Romeo (3), art (2), thou (2), art (2), hurt (1), wherefore (1), what (1)

- *Solution*: Map-Reduce Job

```
m (filename, line) {
  foreach (word in line)
     emit(word, 1);
 }

r (word, numbers) {
  int sum = 0;
  foreach (value in numbers) {
    sum += value;
  }
  emit(word, sum);
}
```

# Map Reduce Illustrated (2)



# Relational Operators as Map/Reduce jobs

- Selection / projection / aggregation
  - SQL Query:

```
SELECT year, SUM(price)
FROM sales
WHERE area_code = "US"
GROUP BY year
```

  - Map/Reduce job:

```
map(key, tuple) {
  int year = YEAR(tuple.date);
  if (tuple.area_code = "US")
    emit(year, {'year' => year, 'price' => tuple.price });
}

reduce(key, tuples) {
  double sum_price = 0;
  foreach (tuple in tuples) {
    sum_price += tuple.price;
  }
  emit(key, sum_price);
}
```

- Sorting
  - SQL Query:

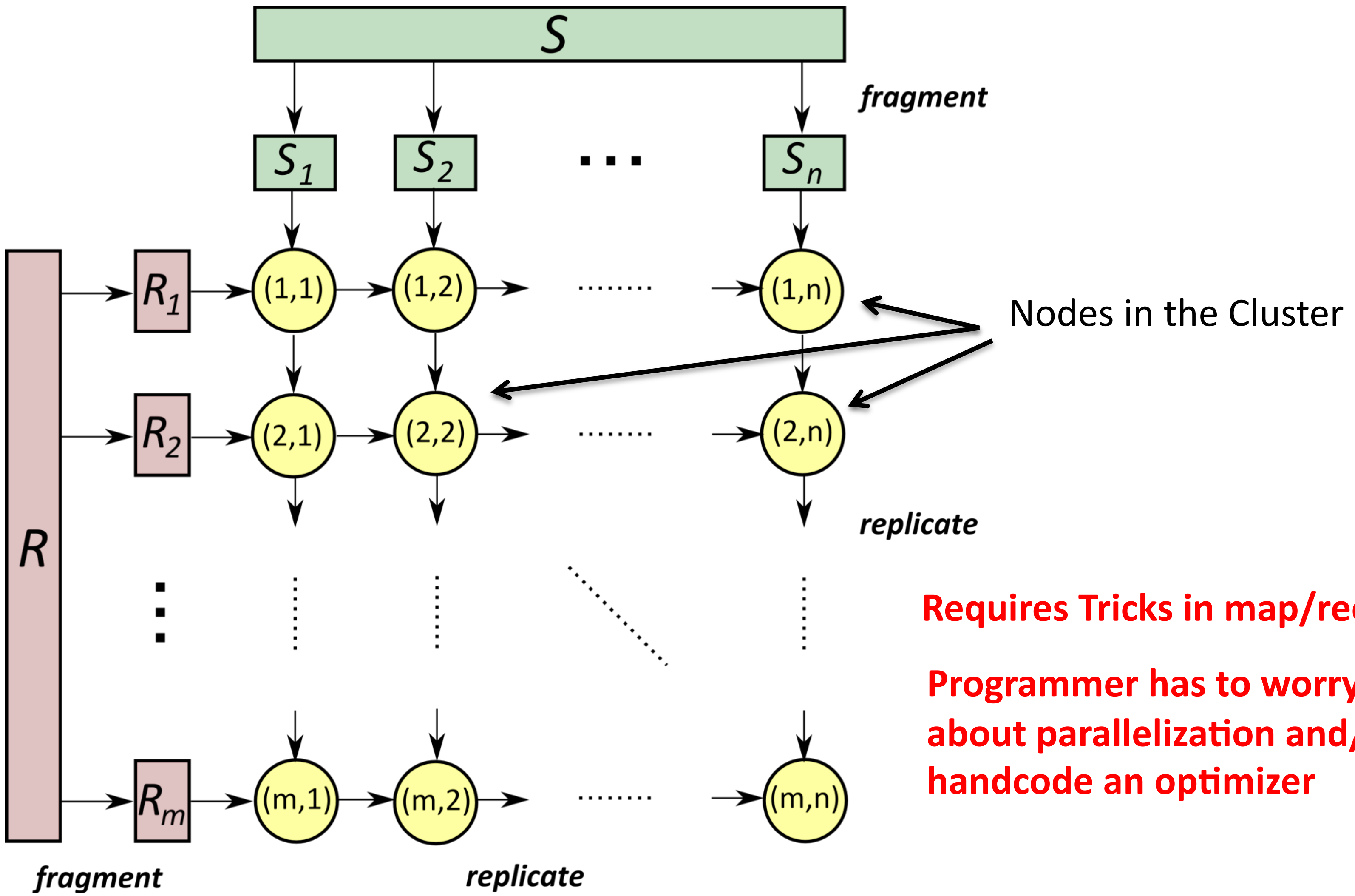
```
SELECT *
FROM sales
ORDER BY year
```

  - Map/Reduce job:

```
map(key, tuple) {
  emit(YEAR(tuple.date) DIV 10, tuple);
}

reduce(key, tuples) {
  emit(key, sort(tuples));
}
```

# Symmetric Fragment-and-Replicate Join




**Requires Tricks in map/reduce**

**Programmer has to worry about parallelization and/or handcode an optimizer**

Important generic example. Specialized parallel joins may exploit data locality

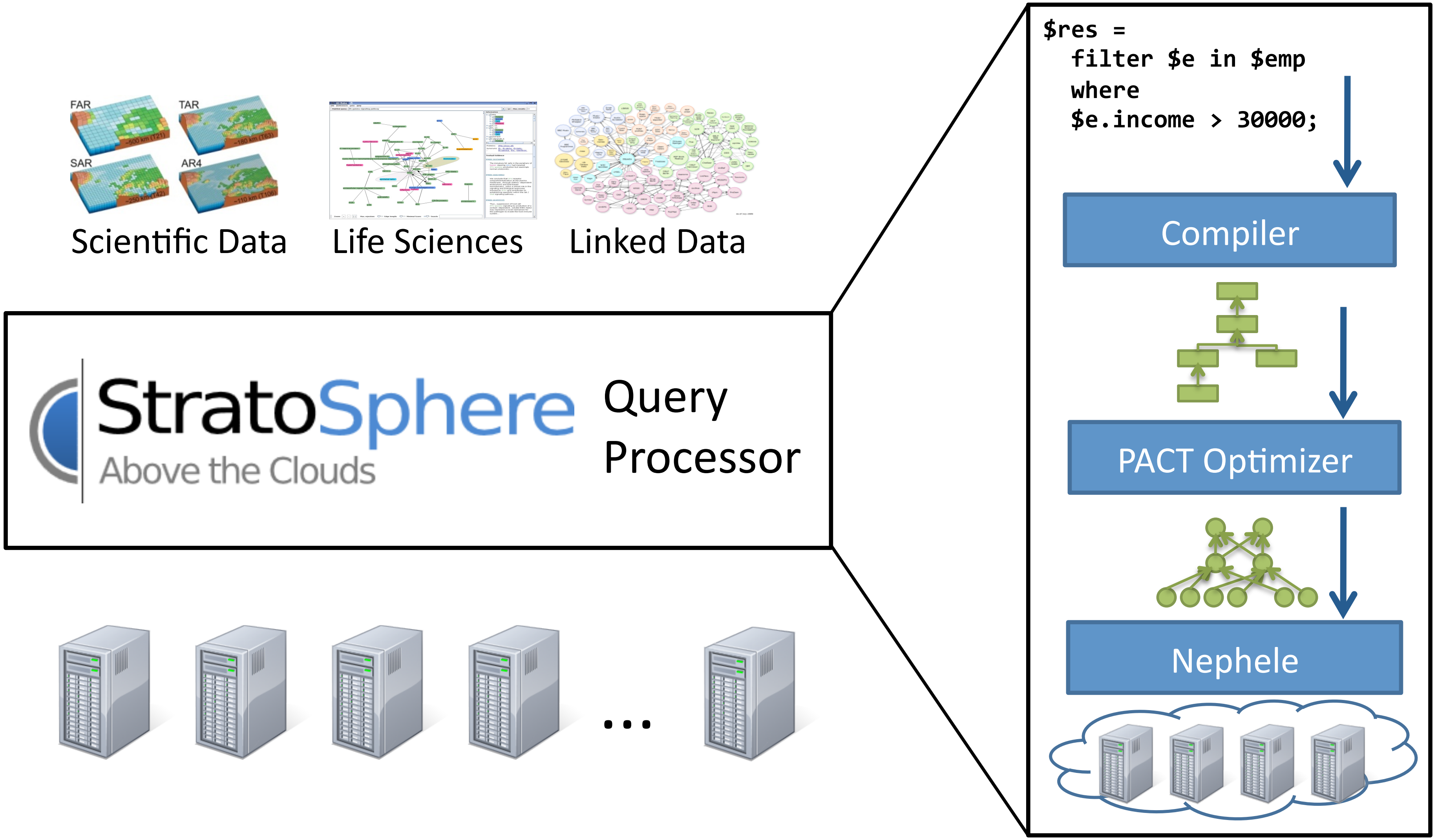
# Stratosphere

- Infrastructure for Big Data Analytics
- Collaborative research group
  - 3 Universities in greater Berlin area (TUB, HUB, HPI)
  - 5 research groups (DIMA, CIT, DBIS, WBI, IS)
  - 5 professors, 2 postdocs, 9+ PhD students, 11+ MSc students
- Flagship project at DIMA, TU Berlin
- Open-source platform
  - www.stratosphere.eu
  - Used for teaching and research by (among others) UCSD, RWTH, INRIA, KTH, UCI

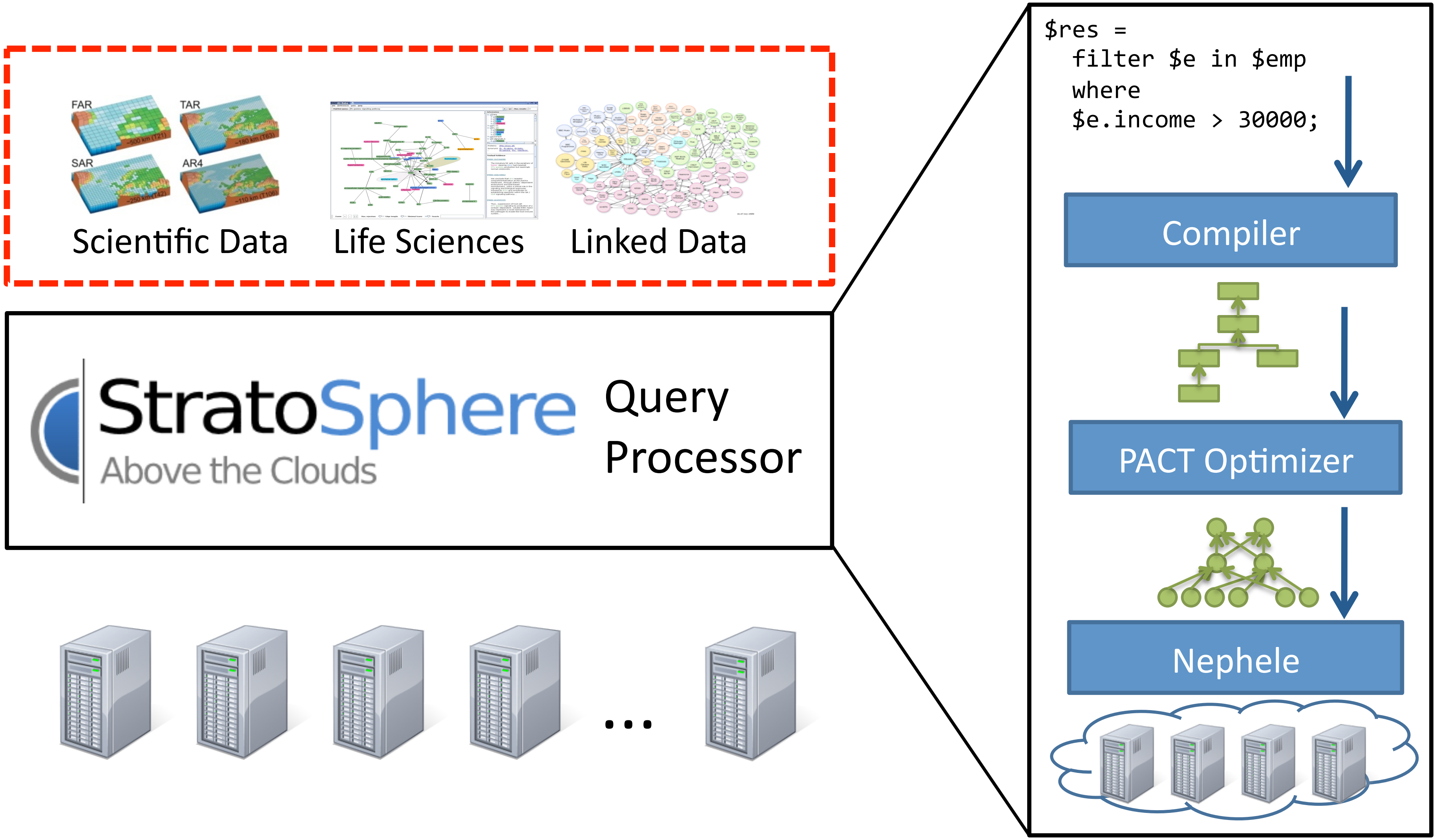
# Stratosphere Research Agenda

- Programming models for Big Data
  - Programming model based on second-order functions
  - Query languages for complex data
  - Intersection with functional programming
- Re-architecting data management systems in massively parallel scale
  - Robust and adaptive query optimization
  - Parallel execution
  - Fault tolerance
  - Resource management
- Use cases on scientific data management, data cleansing and text mining

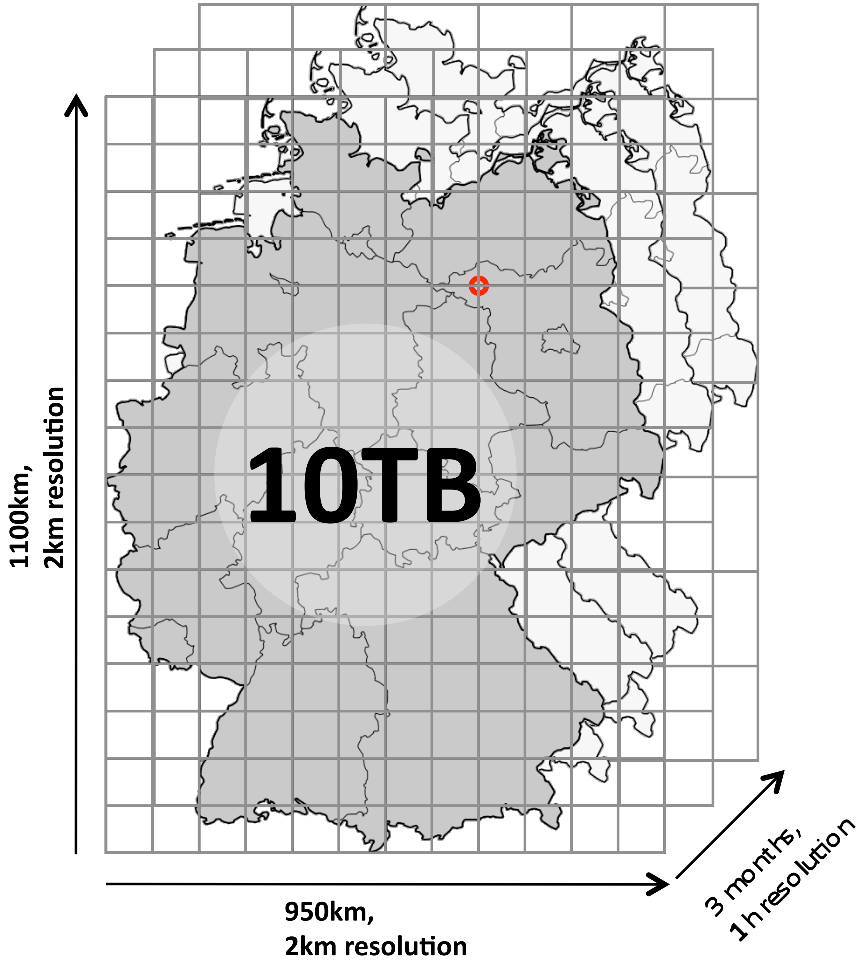
# The Stratosphere Stack

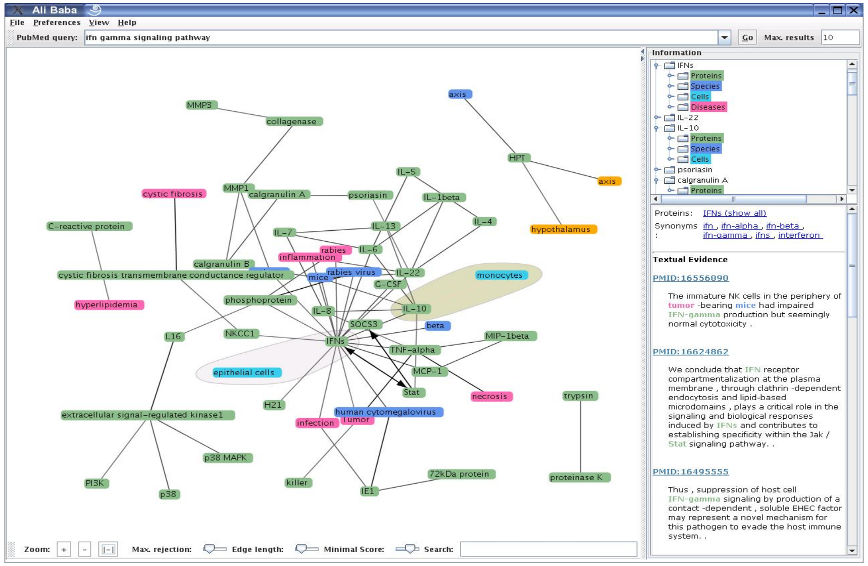


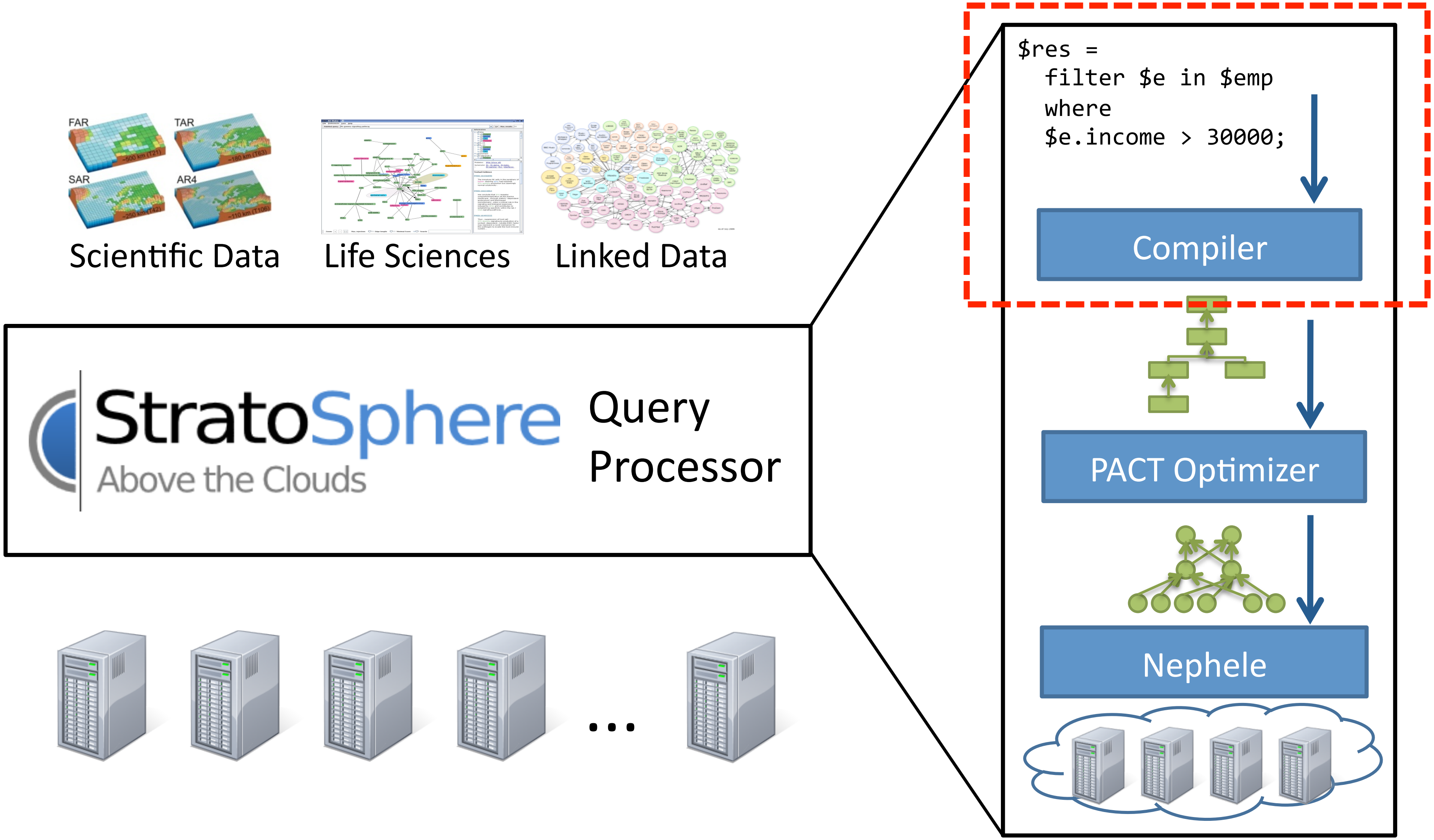
# The Stratosphere Stack



# Stratosphere Use Cases



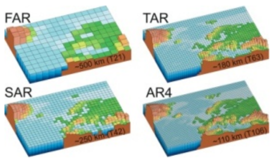



Scientific Data
Life Sciences
Linked Data
StratoSphere
Above the Clouds
Query Processor
...
$res =
filter $e in $emp
where
$e.income > 30000;
Compiler
PACT Optimizer
Nephele

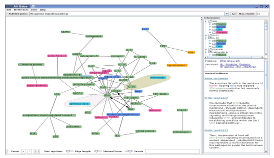
# Stratosphere Query Language

- Currently a few high-level language efforts
  - Inspired by Jaql, XQuery (@INRIA), Pig (@KTH), SQL (@TUB)
  - "Official": Jaql-inspired SIMPLE language (HPI)
- JSON data model, efficient packing to records
- Set of common operators
  - Read, write, filter, transform, intersect, replace, group, join, …
  - Support for Java UDFs
  - Extensible: libraries for domain-specific operators, e.g., data cleansing, text mining, …
- Compiler translates query to *PACT program,* applies logical optimizations

# The Stratosphere Stack




Scientific Data

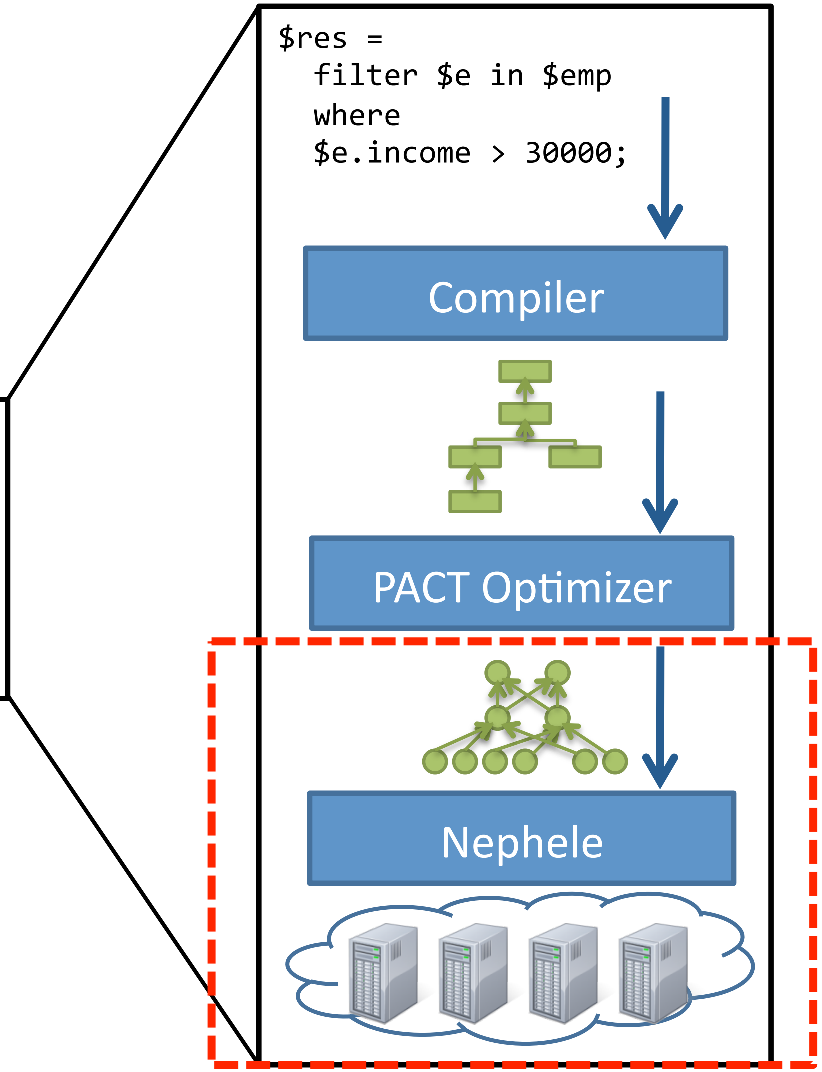


Life Sciences

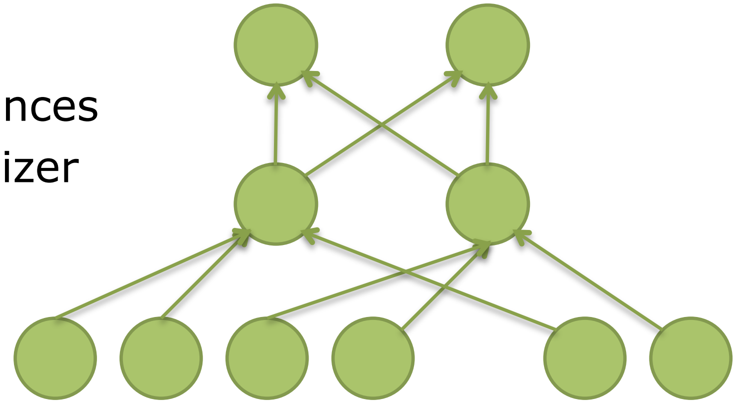
Linked Data

StratoSphere
Above the Clouds

Query Processor

# The Nephele Execution Engine

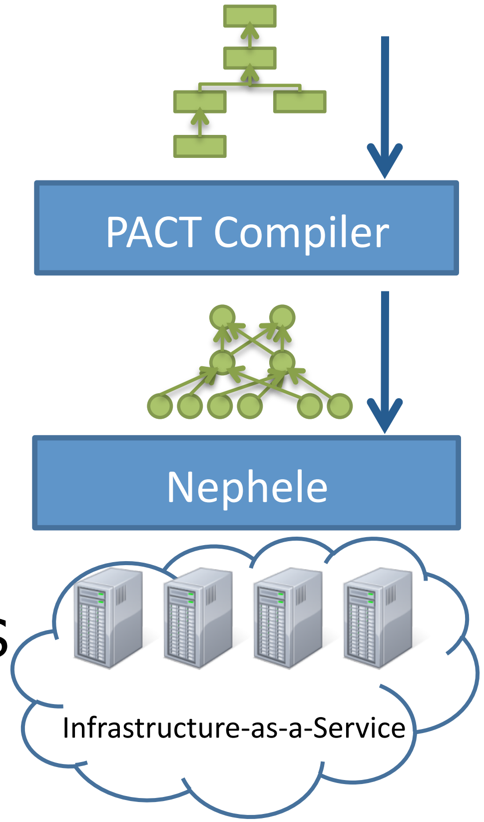


- Executes Nephele schedules
  - DAGs of already parallelized operator instances
  - Parallelization already done by PACT optimizer
- Design decisions
  - Designed to run on top of an IaaS cloud
  - Predictable performance
  - Scalability to 1000+ nodes with flexible fault-tolerance
- Permits network, in-memory (both pipelined), file (materialization) channels

*D. Warneke, O. Kao: Nephele: Efficient Parallel Data Processing in the Cloud. SC-MTAGS 2009*

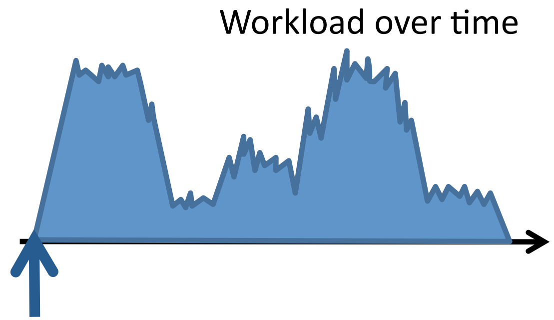
# Nephele Execution Engine

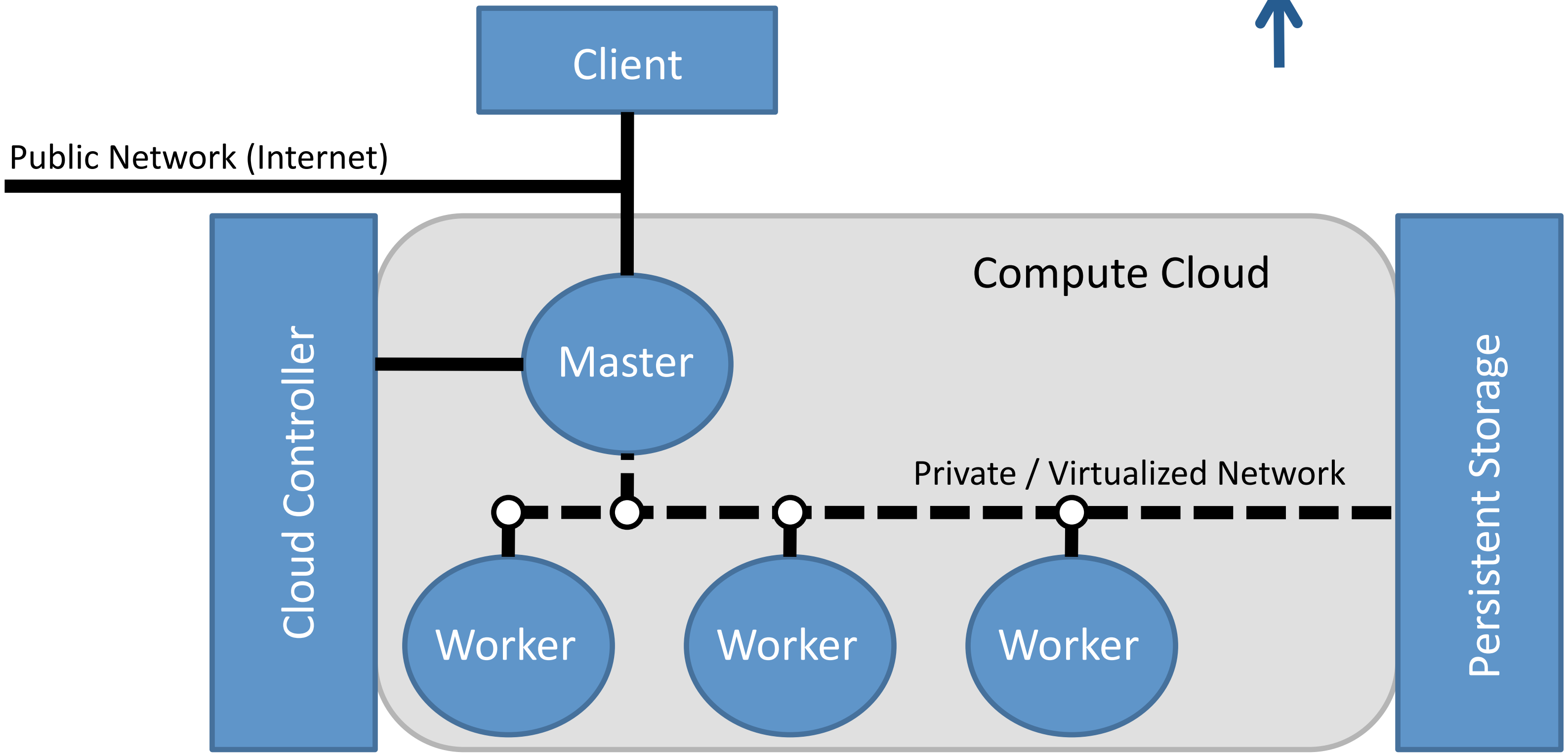
- Executes Nephele schedules
  - compiled from PACT programs
- Design goals
  - Exploit scalability/flexibility of clouds
  - Provide predictable performance
  - Efficient execution on 1000+ nodes
  - Introduce flexible fault tolerance mechanisms
- Inherently designed to run on top of an IaaS Cloud
  - Can exploit on-demand resource allocation
  - Heterogeneity through different types of VMs possible
  - Knows Cloud's pricing model



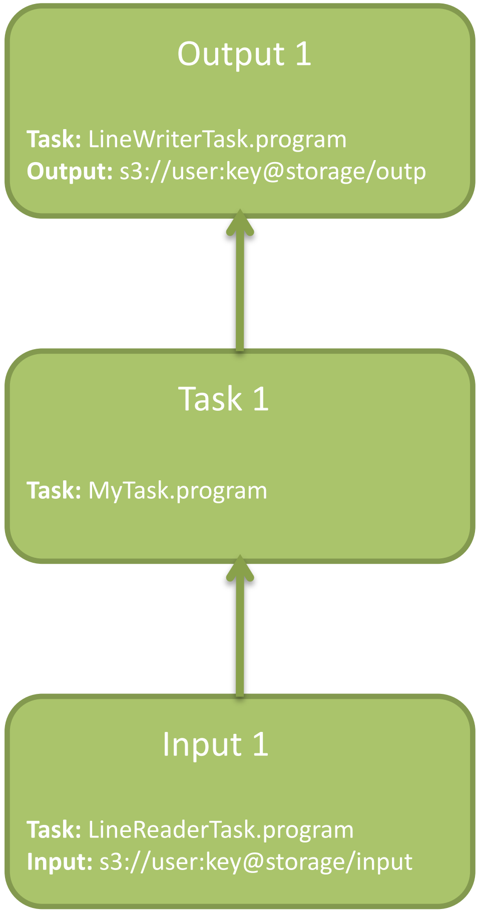
# Nephele Architecture

- Standard master worker pattern
- Workers can be allocated on demand



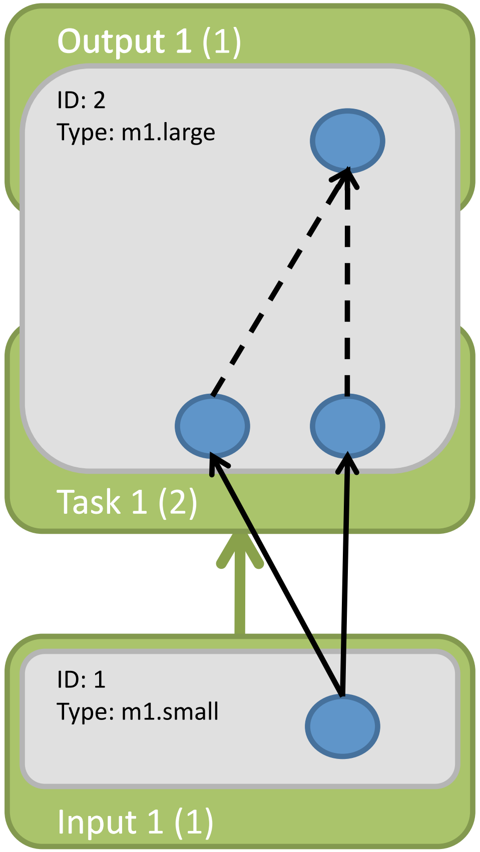
# Structure of a Nephele Schedule



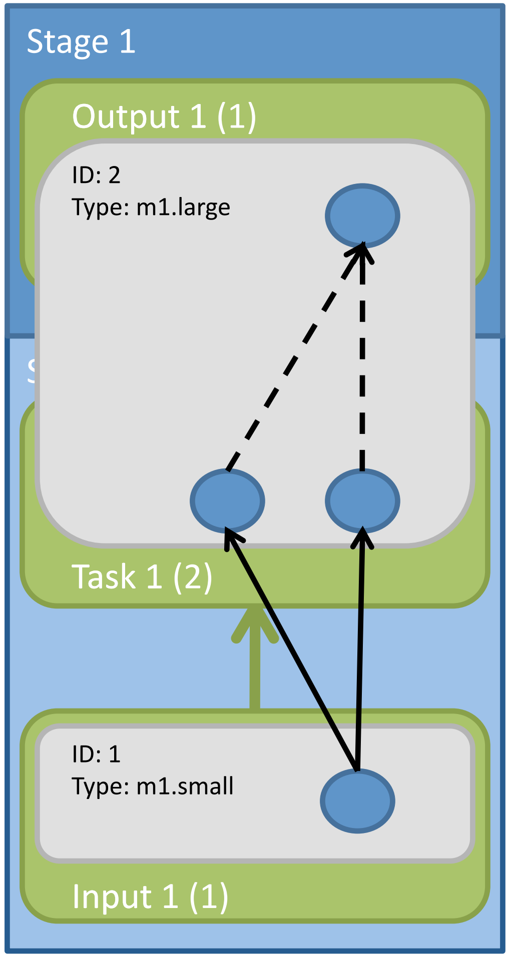

- Nephele Schedule is represented as DAG
  - Vertices represent tasks
  - Edges denote communication channels
- Mandatory information for each vertex
  - Task program
  - Input/output data location (I/O vertices only)
- Optional information for each vertex
  - Number of subtasks (degree of parallelism)
  - Number of subtasks per virtual machine
  - Type of virtual machine (#CPU cores, RAM…)
  - Channel types
  - Sharing virtual machines among tasks

# Internal Schedule Representation





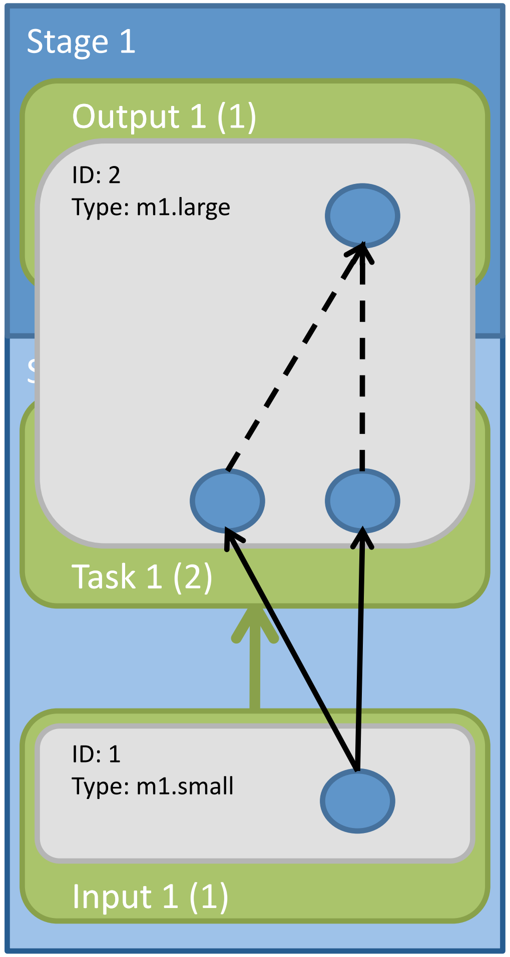

- Nephele schedule is converted into internal representation
- Explicit parallelization
  - Parallelization range (mpl) derived from PACT
  - Wiring of subtasks derived from PACT
- Explicit assignment to virtual machines
  - Specified by ID and type
  - Type refers to hardware profile




- Issues with on-demand allocation:
  - When to allocate virtual machines?
  - When to deallocate virtual machines?
  - No guarantee of resource availability!

- Stages ensure three properties:
  - VMs of upcoming stage are available
  - All workers are set up and ready
  - Data of previous stages is stored in persistent manner

# Channel Types




- Network channels (pipeline)
  - Vertices must be in ***same*** stage
- In-memory channels (pipeline)
  - Vertices must run on same VM
  - Vertices must be in ***same*** stage
- File channels
  - Vertices must run on same VM
  - Vertices must be in ***different*** stages

# Nephele Evaluation (1/2)

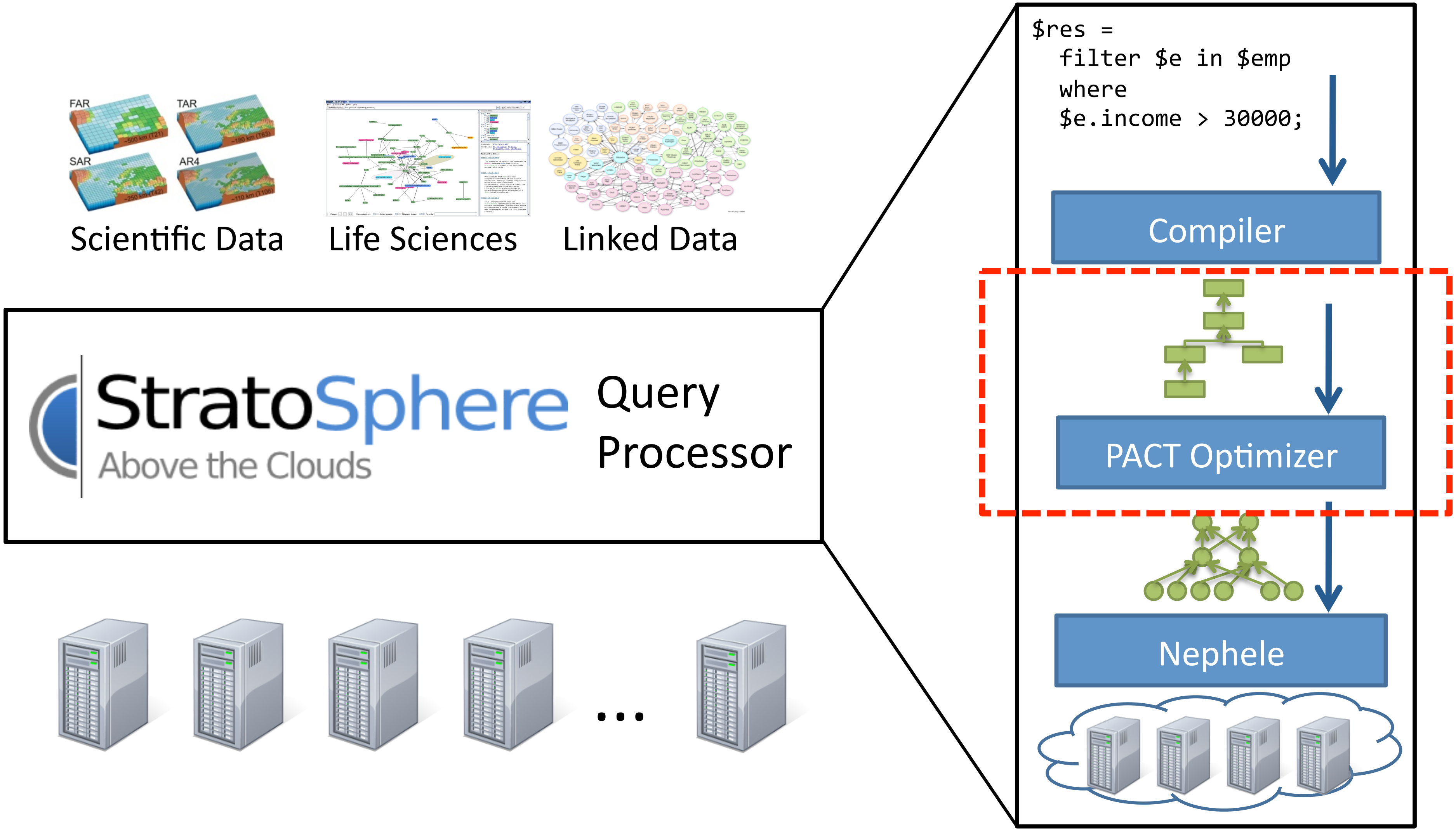
- Demonstrates benefits of dynamic resource allocation

- Challenge: Sort and Aggregate
  - Sort 100 GB of integer numbers (from GraySort benchmark)
  - Aggregate TOP 20% of these numbers (exact result!)

- First execution as map/reduce jobs with Hadoop
  - Three map/reduce jobs on 6 VMs (each with 8 CPU cores, 24 GB RAM)
  - TeraSort code used for sorting
  - Custom code for aggregation

- Second execution as map/reduce jobs with Nephele
  - Map/reduce compatilibilty layer allows to run Hadoop M/R programs
  - Nephele controls resource allocation
  - Idea: Adapt allocated resources to required processing power

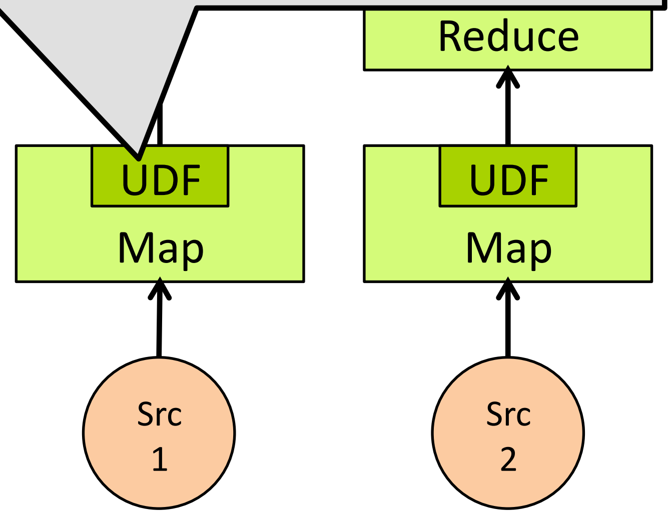
# The Stratosphere Stack



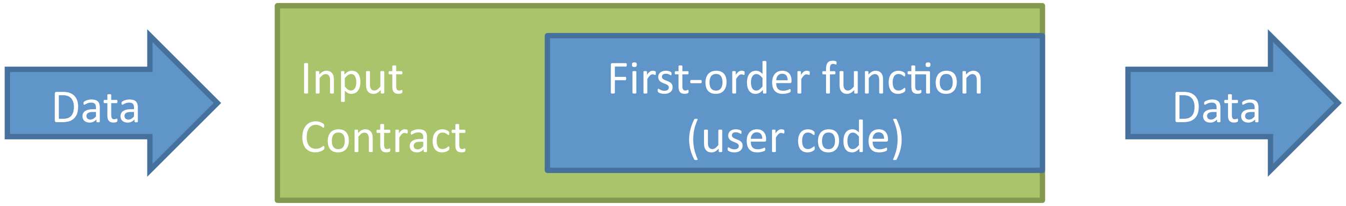
# The PACT Programming Model

- *PACT program* compiler from higher-level language, or written directly by the programmer
- Program is a hard-wired data flow D/ PACT (Parallelization ConTracts) oper
- PACT operator consists of
  - A signature taken from a fixed set of seco order functions
  - A user-defined first-order function written

```
Record mapStub (Record i)
{
  Record o = i.copy();
  if (i.getField(0) < 3.0) {
     o.setField(2,"aab");
     return o;
  }
}
```

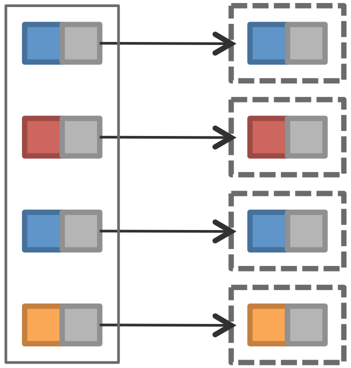


# Parallelization Contracts

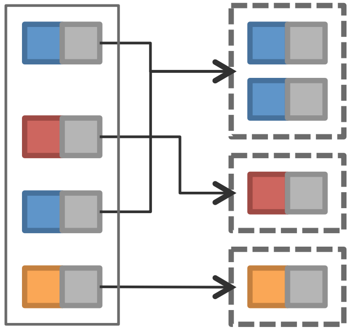



- Describe how input is partitioned in groups
  - “What is processed together”
- First-order UDF called once per input group
- Map PACT
  - Each input record forms a group,
  - Each record is independently processed by UDF
- Reduce PACT
  - One attribute is the designated key
  - All records with same key value form a group

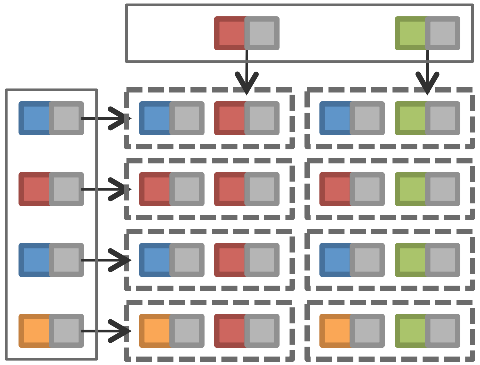
Map PACT



Reduce PACT



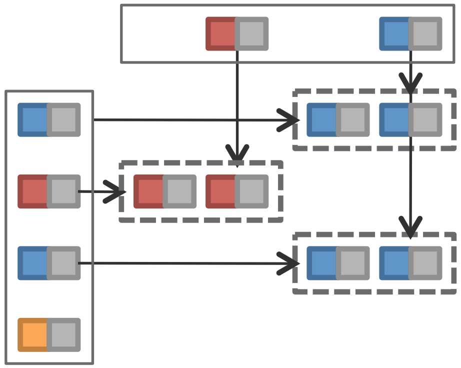
# More Parallelization Contracts



Cross PACT

Each pair of input records forms a group

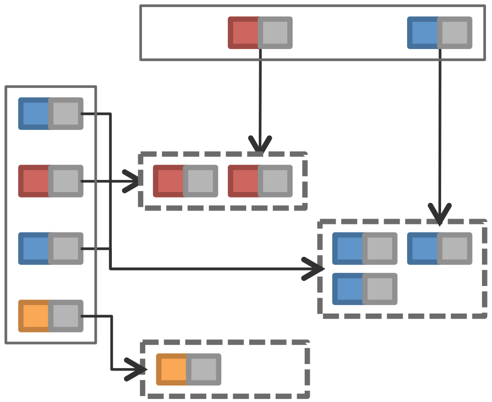
Distributed Cartesian product



Match PACT

Each pair with equal key values forms a group

Distributed equi-join



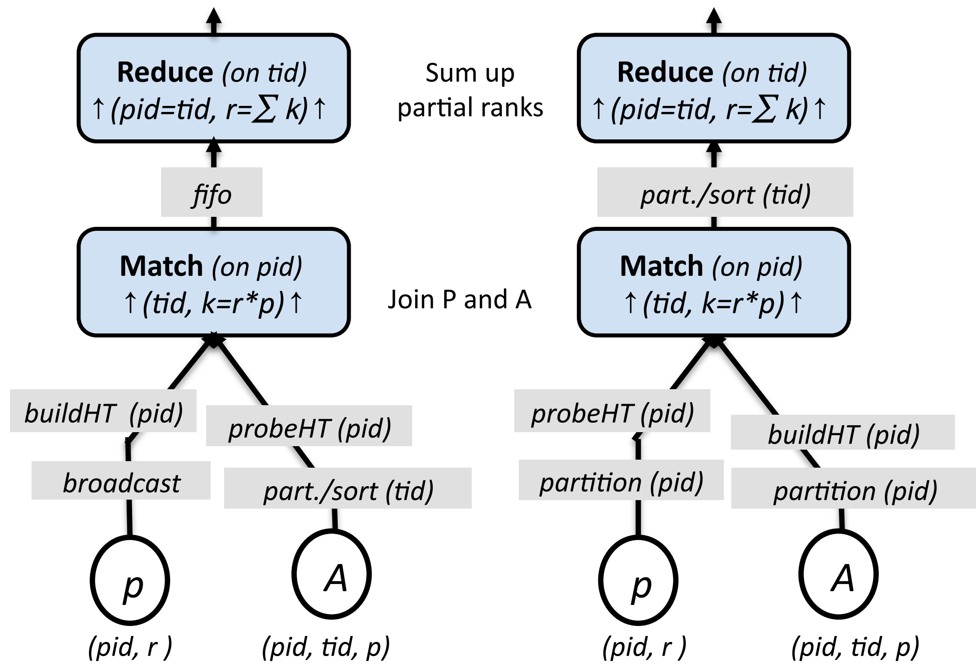
CoGroup PACT

All pairs with equal key values form a group

2D Reduce

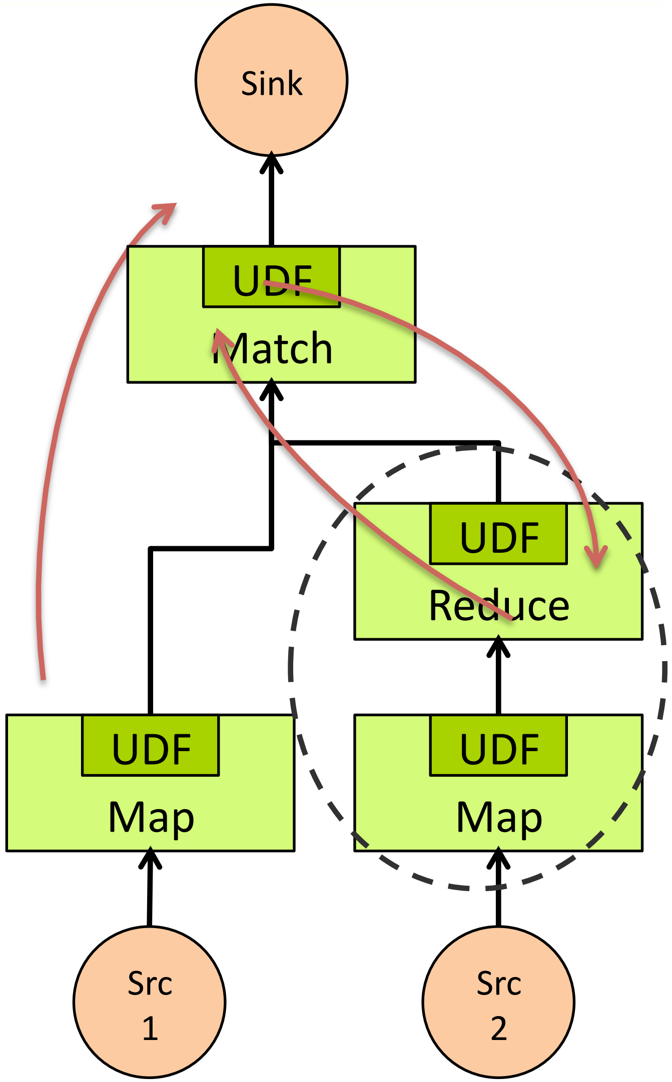
- More PACTs currently under development
  - For similarity operators, stream processing, etc

- Knowledge of PACT signature permits *automatic parallelization*
- Parallelizing and executing Match
  - Broadcast or partition
  - Same as parallel joins
  - Sort-merge or hash
- Volcano-style optimizer
  - Interesting properties
  - Need output contracts



# Reordering of PACT Programs

- Algebraic reordering as in relational DBMSs
- Problem: UDFs are black-box Java functions – semantics unknown
- Derive conflicts on data flow attributes using static code analysis of UDFs
- Prove and apply reordering rules
- Can emulate most relational optimizations
  - Selection and join reordering
  - Limited aggregation push-down



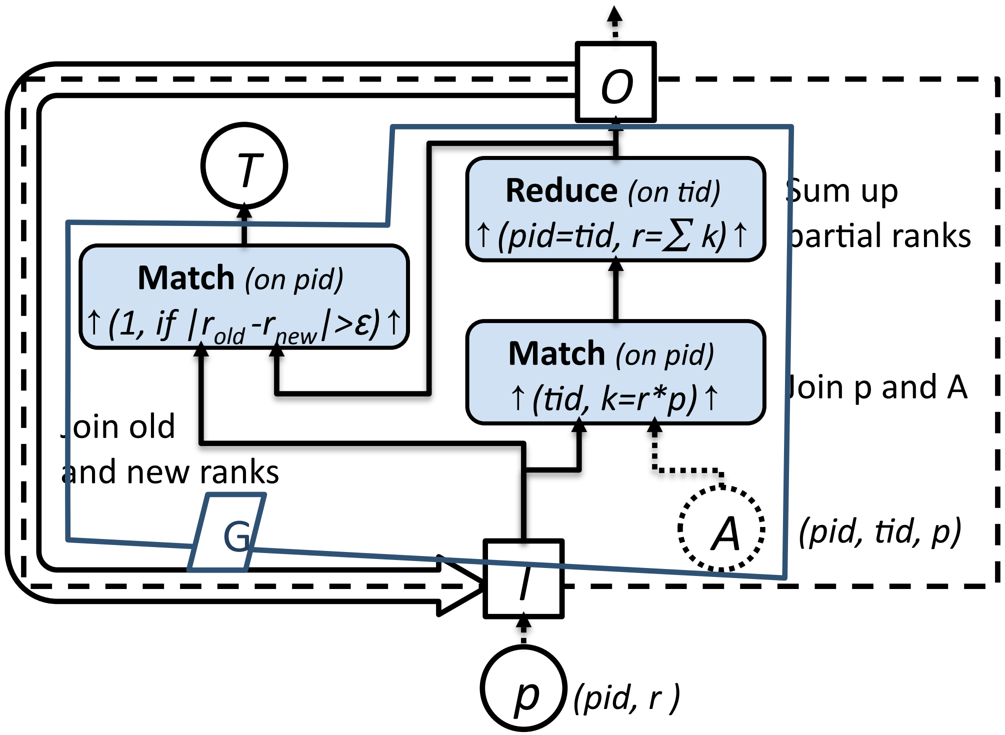
# Reordering of PACT Programs

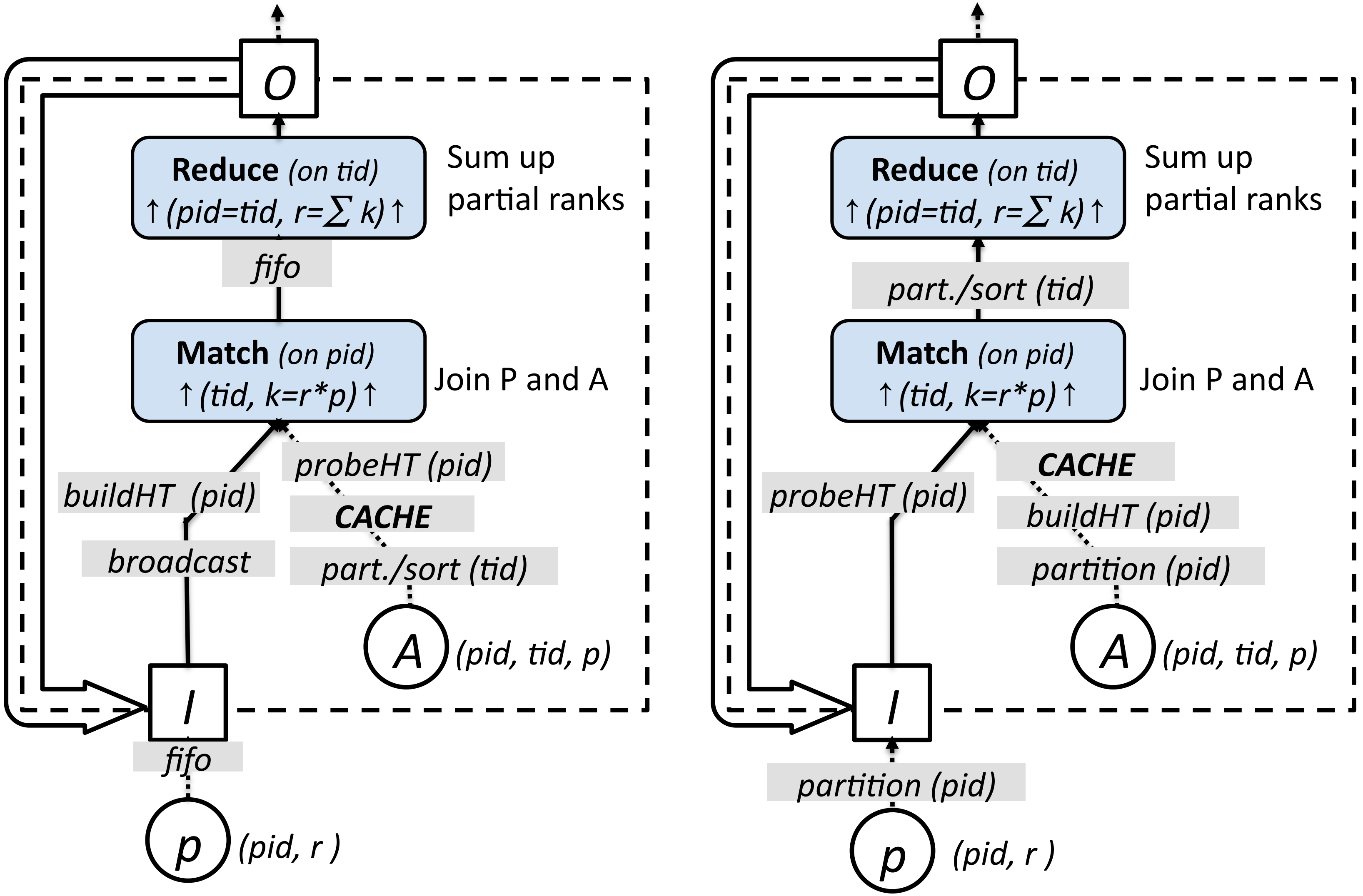
- Novel notion of Read and Write Sets
  - Read set: Attributes that may affect output of operator
  - Write set: Attributes with different values in output
- Given R/W sets, formally prove reordering conditions
  - Map: Only read conflicts
  - Reduce: In addition, key groups are preserved
  - Cross/Match/CoGroup are extensions of Map/Reduce
- Automatically derive R/W sets by analyzing operator code
  - Shallow static code analysis using Use-Def and Def-Use chains
  - Difficulty comes from different code paths – guarantee safety

# Iteration and Recursion

- Many non-trivial data analytics applications involve some form of iteration or recursion
  - E.g., ML and graph analysis
- Iterations realized as fixpoints over PACT programs

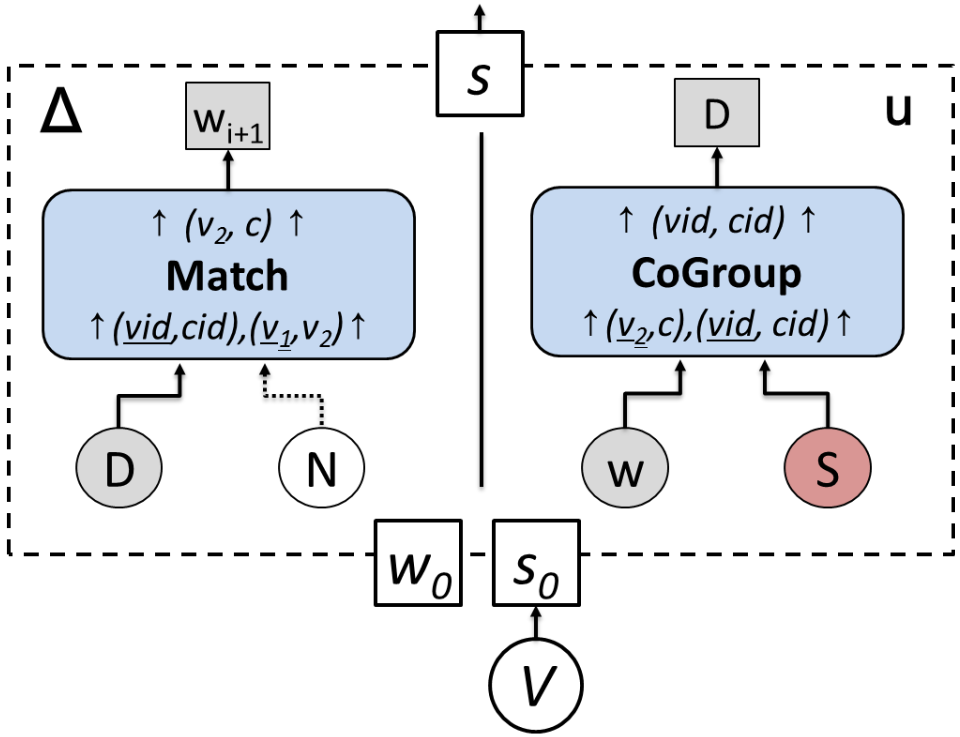
- Several implications
  - Execution, optimization, fault tolerance, etc
- Differential iterations
  - Ala datalog semi-naïve evaluation
  - Permit asynchronous execution




O
Reduce (on tid)
↑(pid=tid, r=∑ k)↑
Sum up
partial ranks
fifo
Match (on pid)
↑(tid, k=r*p)↑
Join P and A
buildHT (pid)
probeHT (pid)
CACHE
broadcast
part./sort (tid)
A (pid, tid, p)
I
fifo
p (pid, r )
O
Reduce (on tid)
↑(pid=tid, r=∑ k)↑
Sum up
partial ranks
part./sort (tid)
Match (on pid)
↑(tid, k=r*p)↑
Join P and A
probeHT (pid)
CACHE
buildHT (pid)
partition (pid)
A (pid, tid, p)
I
partition (pid)
p (pid, r )

- Each iteration changes only few elements
- E.g., the state change of a graph vertex triggered by state changes at the neighborhood

- Encapsulate in an operator with two functions
  - **Δ** updates the workset: elements that may update
  - **u** updates current solution
  - Solution indexed and implicitly combined with changed elements
- Can emulate Pregel



# Summary

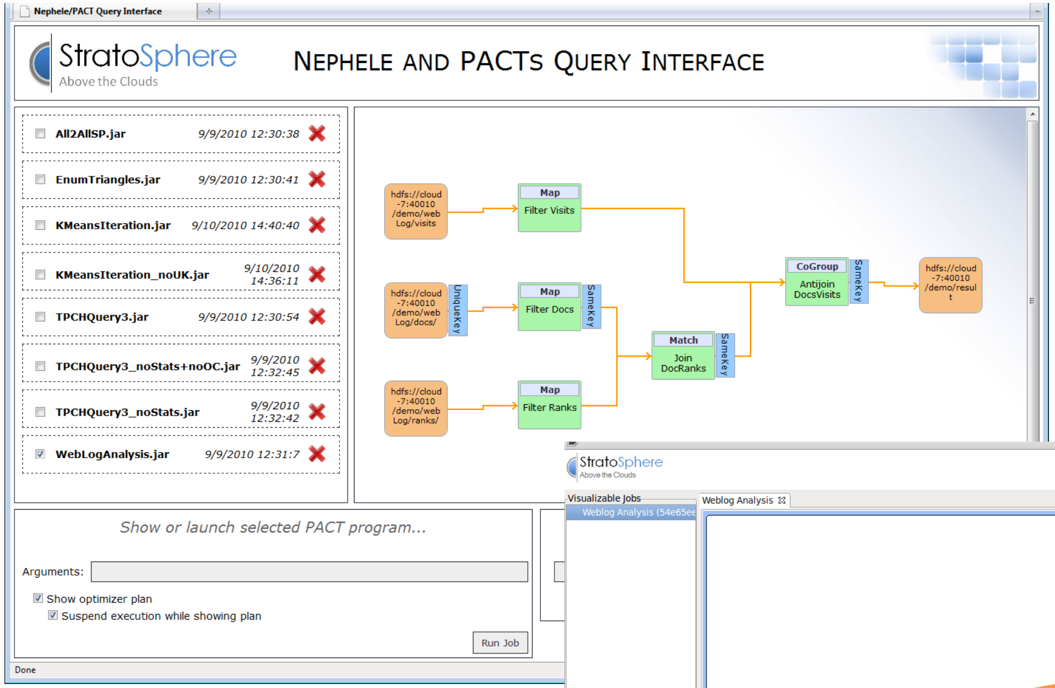
- Stratosphere
  - A modern, open-source platform for Big Data Analytics
- Research themes in Stratosphere
  - Automatic optimization of UDFs
  - Robust query optimization
  - Iteration and recursion
  - Flexible fault-tolerance mechanisms
  - Adaptive optimization and execution
  - Concrete applications: text mining, data cleansing
- Open-source system, used externally for teaching and research
  - Version 0.2 coming soon

# Outlook

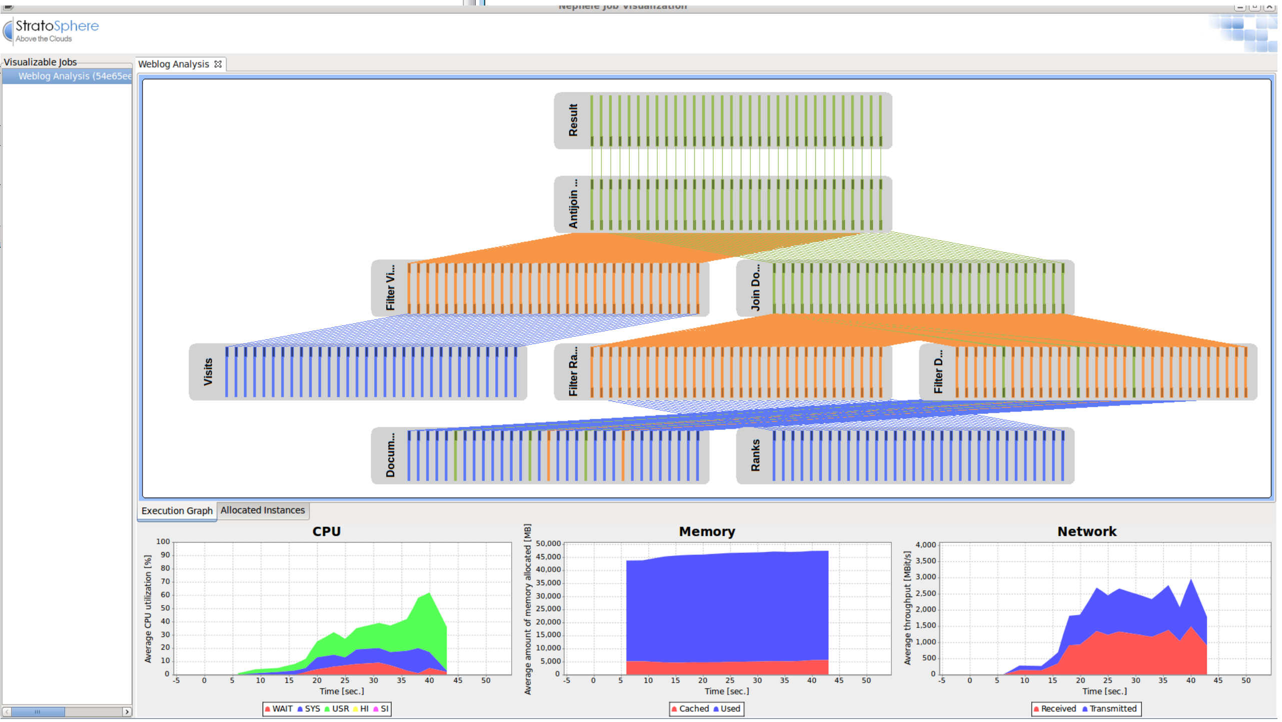
- The data management landscape is changing
  - Hadoop made open-source popular, embraced by big players, startups. New systems are emerging
- Retain key attractive aspects of MapReduce
  - Arbitrary Java UDFs
  - Massive parallelism, fault tolerance
  - Analysis of “in situ” data, “no knobs”
- Stratosphere pushes paradigm forward
  - Data independence aspects from RDBMs
  - Query optimization, indexing
  - Iterative/recursive algorithms
  - Benchmarking (with IBM CAS)

# Download at http://www.stratosphere.eu






Screenshot of
PACT Data Flow Visualizer




Screenshot of
Nephele Parallel
Execution Visualizer