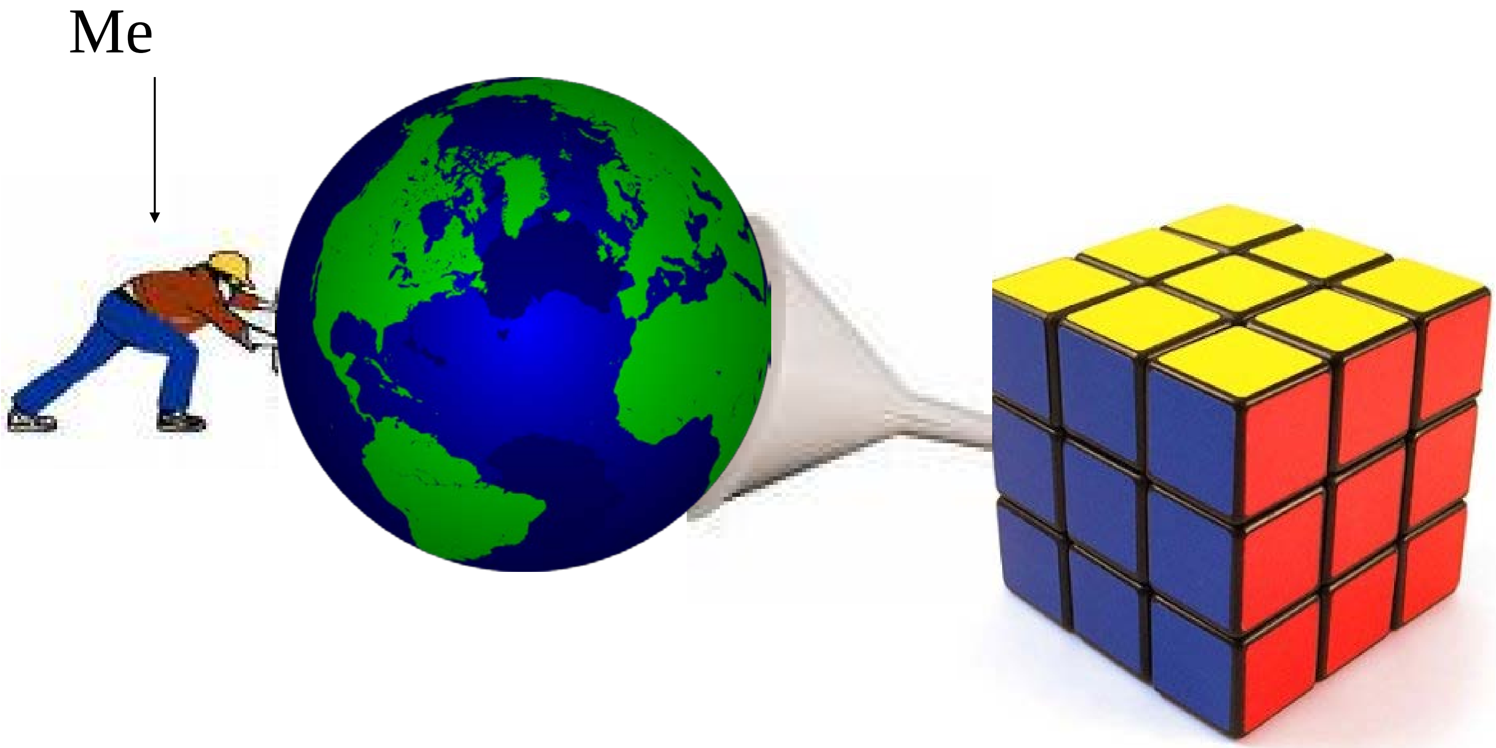
Managing
Big Multidimensional Data

Torben Bach Pedersen
Daisy@CS@Aalborg University

Center for Data-intensive Systems

Speaker Presentation

- I try to squeeze the world into cubes…

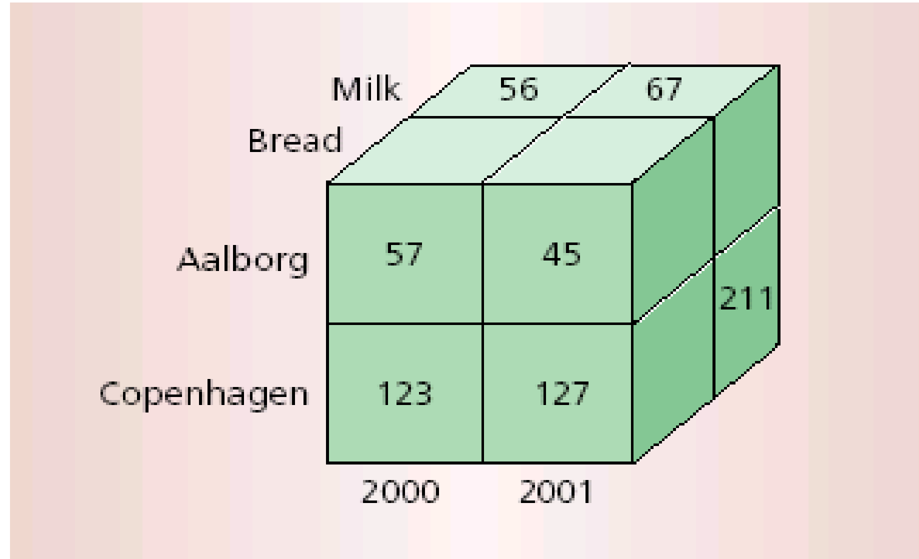


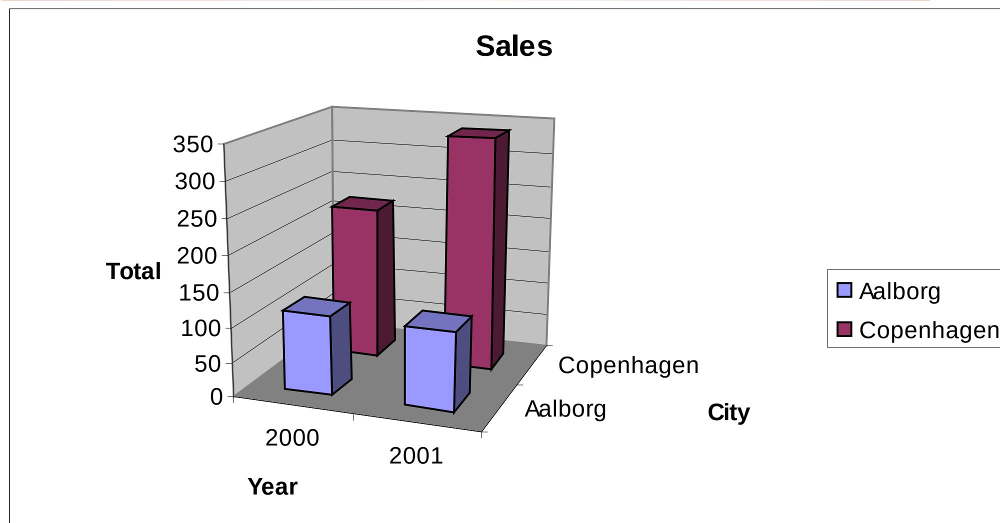
Agenda

- Managing Big Multidimensional Data
- What is Multidimensional Data and what is Big Data?
- What is then Big Multidimensional Data?
  - And what is really new about it?
- Where is it used?
  - Energy, transport, logistics, health, science…
  - Enables new cross-sector optimizations
    - Smart cities/societies,…
- Challenges
  - Volume, velocity, variety, …
  - More iron is not enough…

Multidimensional Data

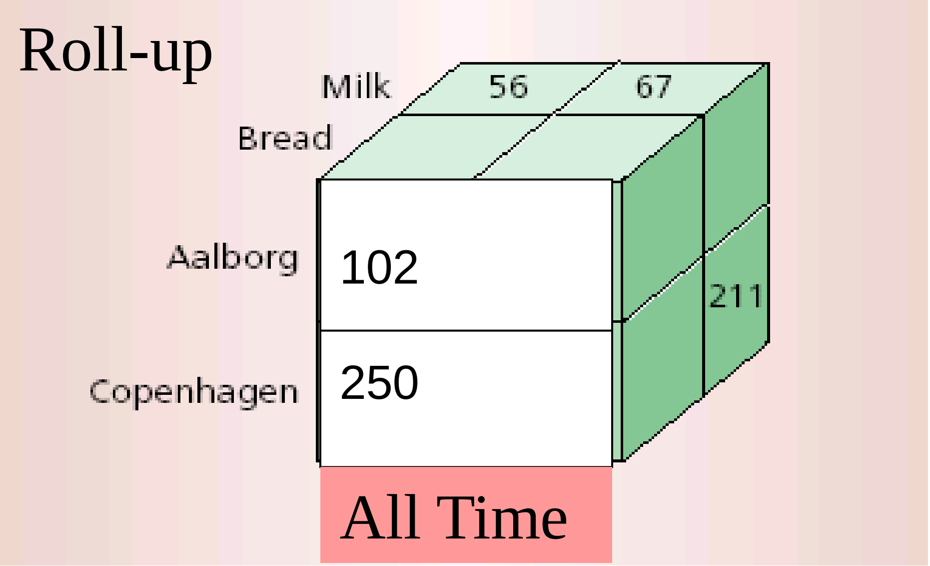
- MD characteristics
  - *Facts* (Sale)
  - *Dimensions* (Time, Product)
  - Facts form *cells* in MD *cubes*
  - Aggregatable *measures* (Price)
  - *Hierarchies* (Prod.,Type,Categ.)
- On-Line Analytical Processing (OLAP)
  - Fast, interactive analysis of large amounts of data
  - Spreadsheets on stereoids
- Iterative queries of two types:
  - Navigate/explore dimensions
  - Aggregate/disaggregate along dimensions (rollup/drilldown)
- Traditionally used for *business intelligence (BI)*

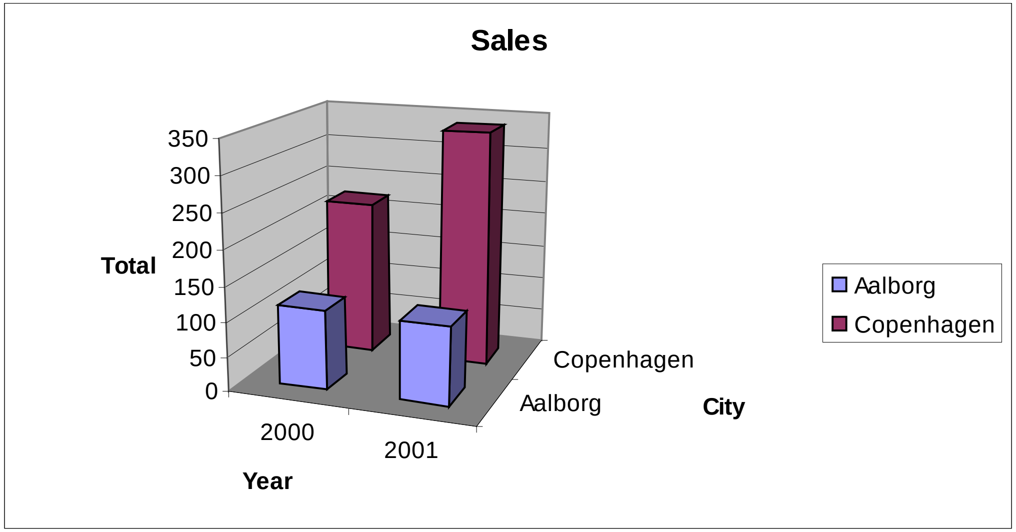




Multidimensional Data

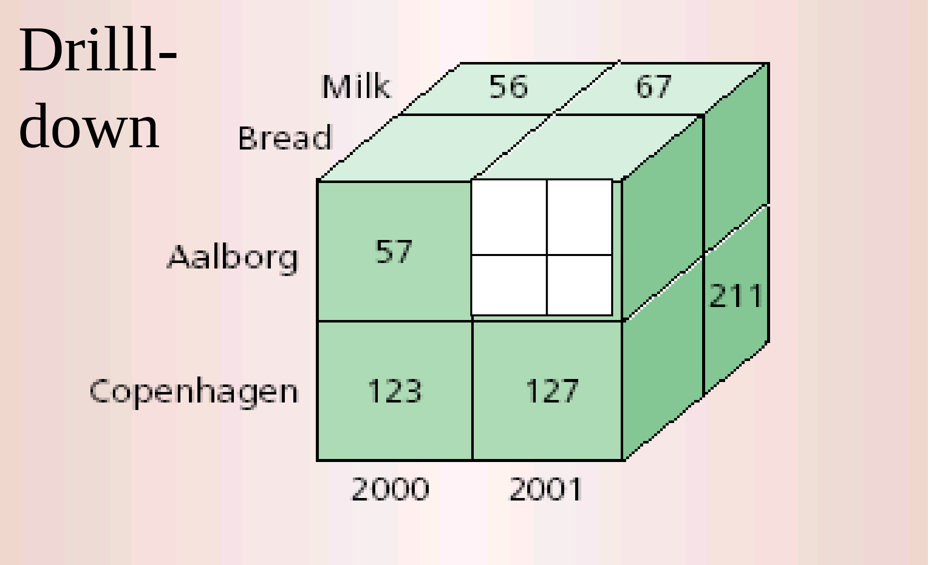
- MD characteristics
  - *Facts* (Sale)
  - *Dimensions* (Time, Product)
  - Facts form *cells* in MD *cubes*
  - Aggregatable *measures* (Price)
  - *Hierarchies* (Prod.,Type,Categ.)
- On-Line Analytical Processing (OLAP)
  - Fast, interactive analysis of large amounts of data
  - Spreadsheets on stereoids
- Iterative queries of two types:
  - Navigate/explore dimensions
  - Aggregate/disaggregate along dimensions (rollup/drilldown)
- Traditionally used for *business intelligence (BI)*

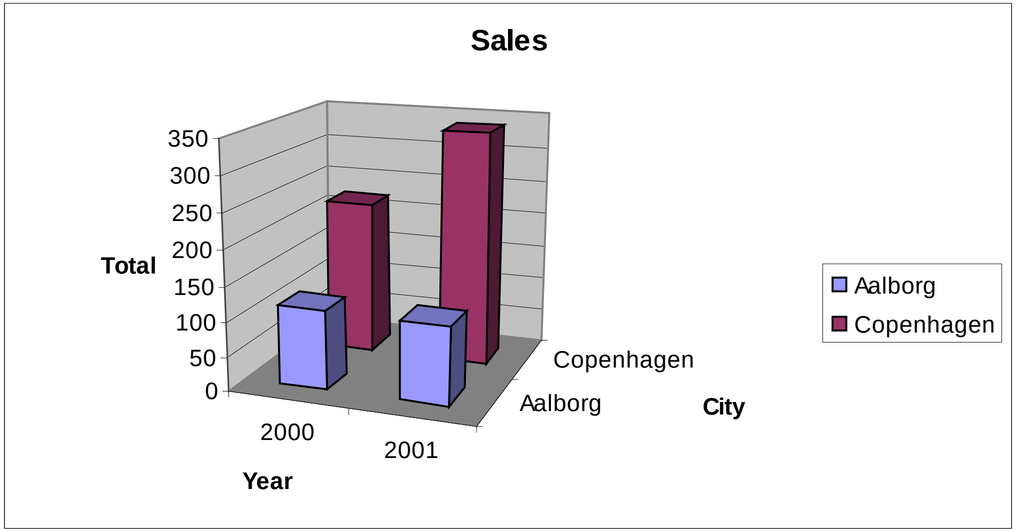




Multidimensional Data

- MD characteristics
  - *Facts* (Sale)
  - *Dimensions* (Time, Product)
  - Facts form *cells* in MD *cubes*
  - Aggregatable *measures* (Price)
  - *Hierarchies* (Prod.,Type,Categ.)
- On-Line Analytical Processing (OLAP)
  - Fast, interactive analysis of large amounts of data
  - Spreadsheets on stereoids
- Iterative queries of two types:
  - Navigate/explore dimensions
  - Aggregate/disaggregate along dimensions (rollup/drilldown)
- Traditionally used for *business intelligence (BI)*





What is Business Intelligence?

- Business intelligence is *“the ability to apprehend the interrelationships of presented facts in such a way as to guide action towards a desired goal”*
  - *H. P. Luhn, A Business Intelligence System, IBM Journal of Research and Development. Vol. 2(4), 1958*
- *Business intelligence is “an umbrella term that includes the applications, infrastructure and tools, and best practices that enable access to and analysis of information to improve and optimize decisions and performance”*
  - Gartner Reports, IT Glossary, 2013

- So, it’s about optimizing your business using data…
- For example:
  - ***Show the total sales by product category***
  - ***What is the trend over time (drill-down by month)?***
  - ***How do sales correlate with location (drill-down by store loc)?***

What is Big Data, then?

- ”Big data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.”
  - http://en.wikipedia.org/wiki/Big_data

- So, it should be so ”big” that it becomes ”difficult” to do it the traditional way…

Big Data Characteristics

- ”The 3 V’s” (but 1-2 V’s is “enough”)
- Volume
  - **Very** large data volumes
- Velocity
  - Data arrives **very** fast (data streams)
- Variety
  - Data has **varied/complex** formats/types/meanings

More V’s:

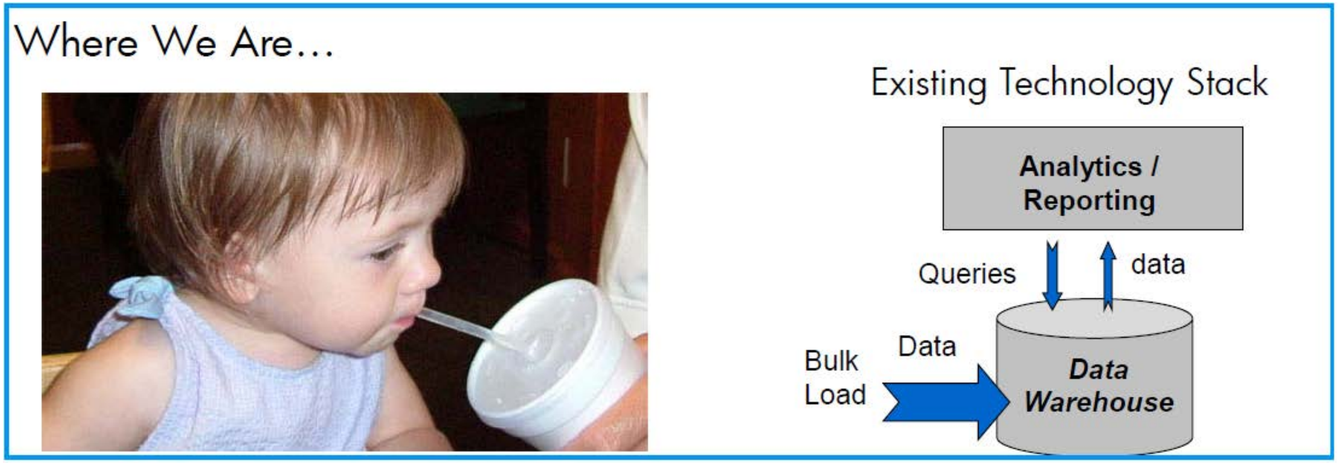
  - Veracity – how much can we trust data?
  - Viability – can our data be used for anything useful?
  - Visibility – data must be visible to the Big Data processes
  - Variability – the meaning of data changes over time/place/context
  - Visualization – complex visualization needed to fully understand
  - Value – what *real value* can this data add to our business?

BI Versus Big Data

- Similarities (what is not so new?)
  - Collecting, integrating, and analyzing data to gain knowledge
  - Large data volumes
  - Data (often) arrives at a fast pace
- Differences (what is really new?)

	BI	**Big Data**
Data types	Structured (mostly)	Unstructured (also)
Data sources	Mostly internal	Mostly external
History	Essential	(Often) less relevant
Users	Manager/controller	Data scientist
Precision	Exact results	Approximate results
Privacy	Not critical	Critical
Control over data	Almost full control	Little or no control

Illustrating The Change



Malú Castellanos, HP Vertica

(Typical) Types of Big Data

- Search data
  - Web pages, searches, rankings, etc.
  - Google's data…the first type of Big Data
- Social network data
  - Updates from Twitter, Facebook, LinkedIn, user fora,….
  - Text, images, user info, Likes, location, friends-graph,…
- Linked/Open Data
  - Data shared/published on WWW, e.g., using Semantic Web techn.
- But it is not just from WWW…
- Big Sensor Data
  - Big Science Data (CERN Large Hadron Collider, etc.)
  - Big GPS/Location Data
  - Big RFID Data
  - Big **Energy** Data – the basis of the Smart Grid

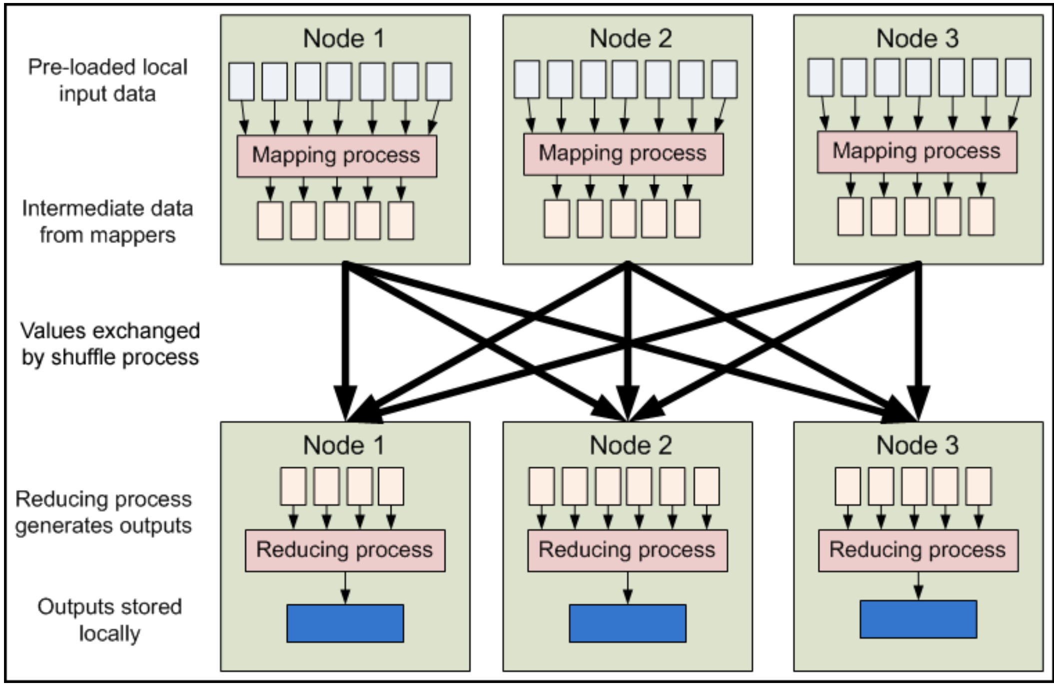
How to do BI on Big Data?

- How to handle…

- Volume
  - …really **big** data volumes
- Velocity
  - …that arrive very **fast**
- Variety
  - …and has very different **types**/**meanings**?

Volume – Typical Approach

- Data parallelism
  - Split data, compute in parallel, coordinate, redundancy
  - MapReduce/Hadoop
  - Lucene/Solr for text
- Pros:
  - Scalability, cheap HW, fault tolerant, (often) intuitive model
- Cons:
  - Load balancing, latency, (often) inefficient, low productivity
  - Work harder, not smarter ☺



[Hadoop Tutorial, Yahoo developer network]

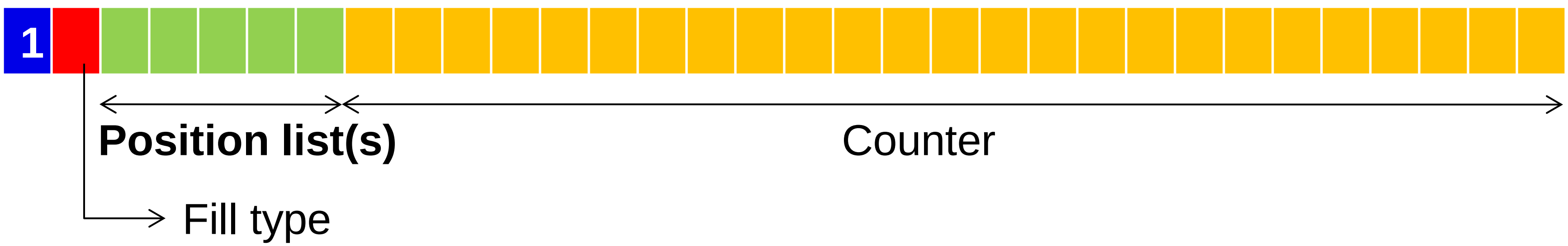
Volume: Efficiency

- Pure parallelization is not enough
- Efficient algorithms and data structures (still) necessary
- A particularly efficient data structure for multidimensional searches is **(compressed) bitmap indices**
  - So, what is that?
- Idea: make a ”position bitmap” for every possible value
  - #Danmark: 01110010101010… (row 2,3,4,7… has #Danmark)
  - #BigData: 10001101010101… (row 1,5,6,8… has #BigData)
  - Only takes (no. values)*(no. rows)*1 bit space
  - **Very** efficient ”index intersection” (CPU AND/OR) on bitmaps
- Problem: space usage
  - With *m* possible values and *n* rows: n*m bits needed
  - But the probability of a 1 is only 1/*m* => very few 1's

PLWAH Bitmap Compression

- Position List Word Aligned Hybrid
  - Literal word

  - Fill word



  - Four intuitive steps (integrated in practice):
    1. Split bitmap into chunks of w-1 bits (word length w)
    2. Make fill words or literal words
    3. Merge fill words (adapt the counter)
    4. Merge fill words with literal words (if possible)

PLWAH Example (Step 1)

- Original uncompressed bitmap (w = 32)

  0000000000 0000000000 0000000000 00

  0000000000 0000000000 0000000000 00

  0000000000 0000000000 0000000000 00

  0000001000 00

- Form groups of w-1 bits

  0000000000 0000000000 0000000000 0

  0000000000 0000000000 0000000000 0

  0000000000 0000000000 0000000000 0

  0000000001 0000000000 0000000000 0

PLWAH Example (Step 2)

0000000000 0000000000 0000000000 0

0000000000 0000000000 0000000000 0

0000000000 0000000000 0000000000 0

0000000001 0000000000 0000000000 0

- Generate Fill and Literal words

  1|0|00000|0000000000 0000000000 00001

  1|0|00000|0000000000 0000000000 00001

  1|0|00000|0000000000 0000000000 00001

  0|0000000001 0000000000 0000000000 0

PLWAH Example (Step 3)

1|0|00000|0000000000 0000000000 00001

1|0|00000|0000000000 0000000000 00001

1|0|00000|0000000000 0000000000 00001

0|0000000001 0000000000 0000000000 0

- Merge Fill words

  1|0|00000|0000000000 0000000000 00011

  0|0000000001 0000000000 0000000000 0

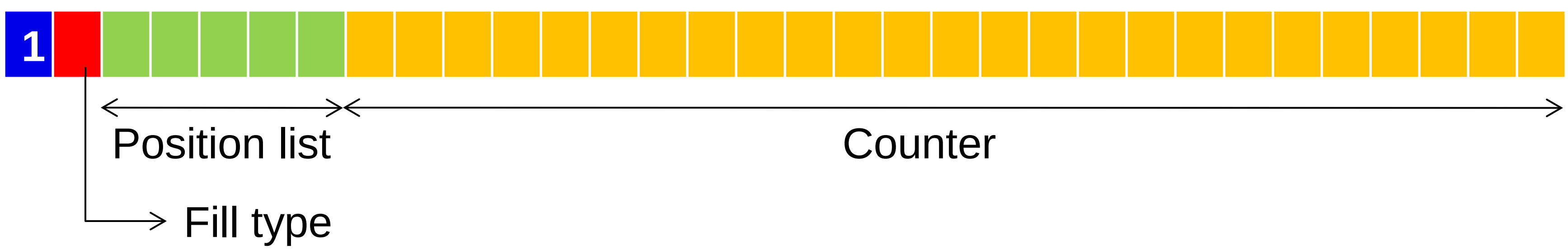
PLWAH Example (Step 4)

1|0|00000|0000000000 0000000000 00011

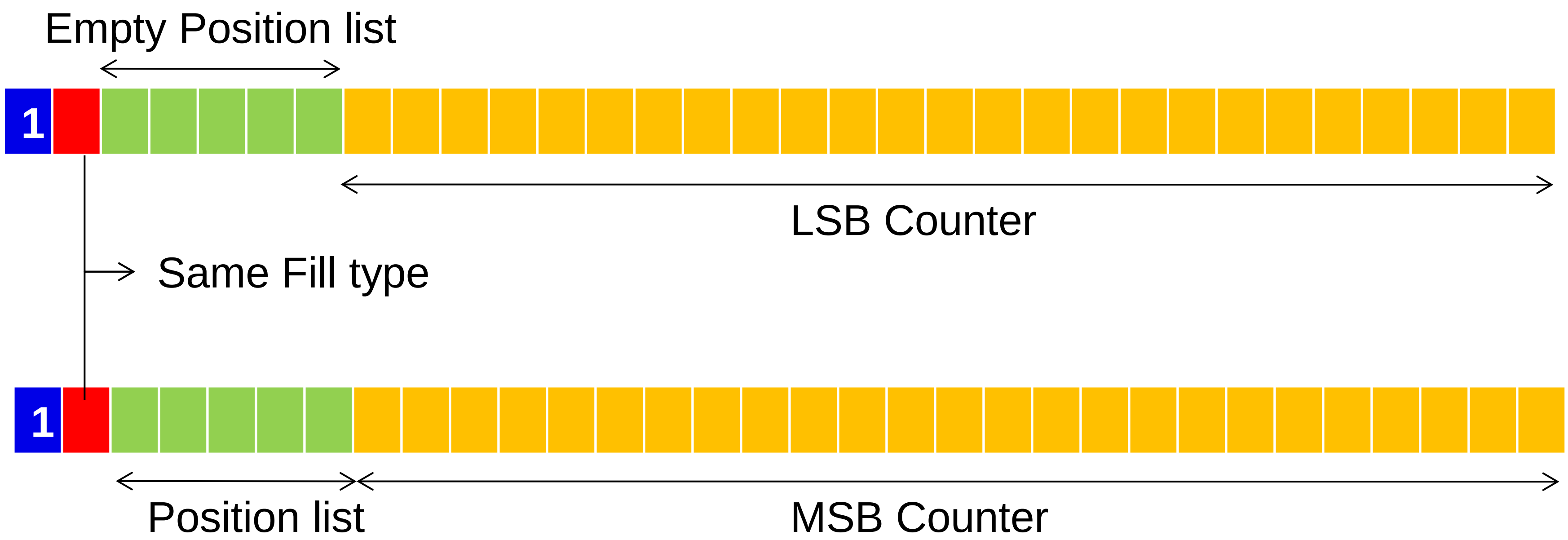
0|0000000001 0000000000 0000000000 0

- Merge Fill words with Literal words

1|0|01010|00000000000 000000000 00011



Adaptive Counter



- A second Fill Word is used if the counter is too small
  - Two fill words of the same type
  - First fill word has an empty position list

PLWAH Storage Estimates

- High-cardinality attribute uniformly distributed
  → most bitmaps are sparse

0000…0000100000…0000

Fill word of 0s with a non-empty position list → one word

→ c bitmaps, each bitmap has n / c set bits

→ total size = **n** words (versus m*n for uncompressed)

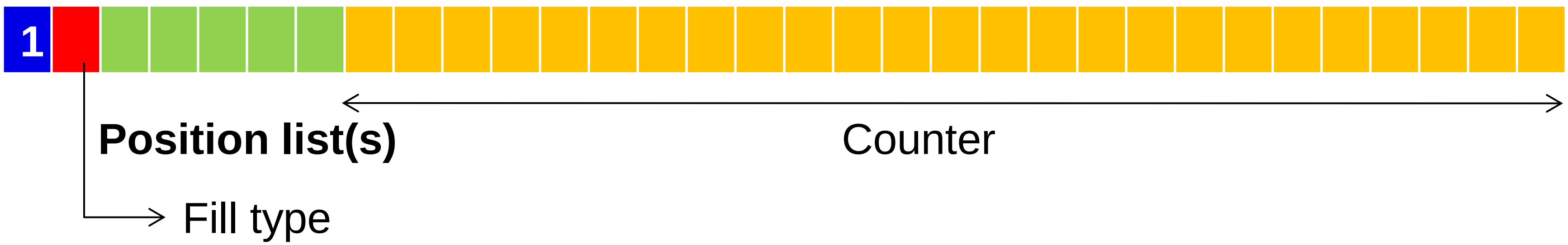
→ Independent from the cardinality (for c >> w)

→ PLWAH compressed bitmaps are half the size of the classical WAH compressed bitmaps
(within the compression limits)

PLWAH Summary

- Literal+fill words; split bitmaps into w-1 bit chunks
- 1 or more chunks with all 0's/1's = fill, otherwise literal

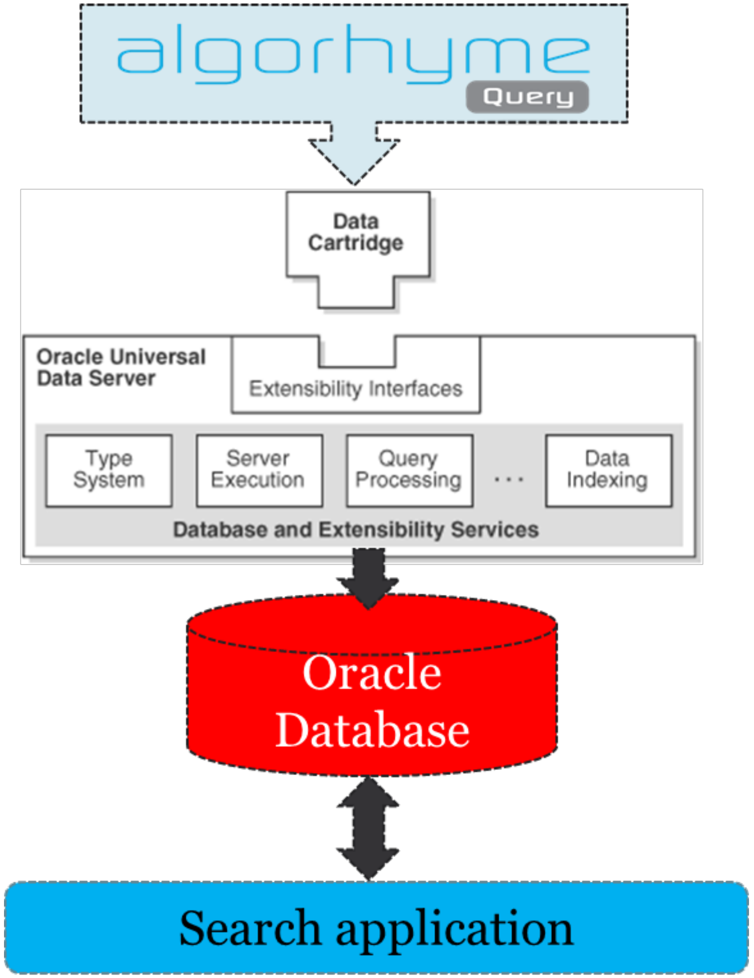
- Finally, merge fill words with "few bit" literals at the end



- Employs novel CPU instruction sets (POPCnt, etc)
- Storage: comparable to BBC (Oracle), half of WAH
- Speed: 40% faster than WAH, 15 times BBC (Oracle)
- Patent pending, Algorhyme spin-out

Algorhyme Query

- Oracle Data Cartridge
  - “DB Chip tuning set”
- AQ vs. Oracle Bitmaps
  - 10-15 times faster
- AQ vs. Oracle Text
  - 10-50 times faster
- AQ vs. Apache Lucene
  - 20-30 times faster
- Combined text and structured metadata
  - Up to 100 times faster

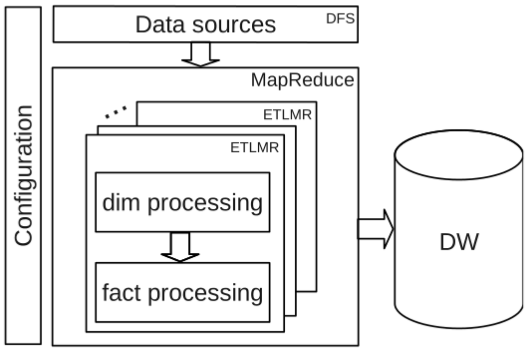


Volume: Productivity?

- Doing ETL in Hadoop is **very** cumbersome
  - And even Pig and Hive are not suited for **dimensional ETL**
- Solution**:** Programmable ETL (instead of ETL GUIs)
  - Powerful libraries for dimensional concepts (dimension, fact, measure, SCD,…) allows powerful yet compact ETL code
- Several versions
  - PygramETL: single+multicore
  - ETLMR: Python-based MR, productivity focus, speed via scale-out
  - CloudETL: Hadoop (Java), more efficient, less productivity…

ETLMR

- Define sources/targets/dimensions/facts
- Process dimensions
  - In parallel, 4 schemes
- Process facts+load
  - In parallel
- Evaluation
  - Linear speed-up (20 tasks)
  - 14 statements **with** SCDs
  - Pig/Hive: 23/40 statements **without** SCDs




```
# Defined in config.py
# Define the data sources:
fileurls = ['dfs://localhost/TestResults0.csv',
            'dfs://localhost/TestResults1.csv',
            'dfs://localhost/TestResults2.csv',
            'dfs://localhost/TestResults3.csv']
# Declare dimension tables (only pagedim is shown here):
pagedim = SlowlyChangingDimension(name='page',
             Key='pageid', lookupatts=['url'], attributes=['url',
             'size', 'validfrom', 'validto', 'version', 'domainid',
             'serverversionid'], versionatt='version',
             srcdateatt='lastmoddate', fromatt='validfrom',
             toatt='validto', srcdateatt='lastmoddate')
```

```
# In config.py
# Declare the fact table (here we support bulk loading):
testresultsfact = BulkFactTable(name='testresultsfact',
                    keyrefs = ['pageid','testid', 'dateid'], measures=['errors'],
                    bulkloader=UDF_pgcopy, bulksize=5000000)
# Set the referenced dimensions and the transformations applied to facts:
facts = {testresultsfact : {'refdims':(pagedim, datedim, testdim),
          'rowhandlers' : (UDF_convertStrToInt,)}}
```

CloudETL

- Process dimensions
  - Pre-update in mappers for more efficient SCDs
  - Big Dimension scheme
- Process facts+load
  - Dimension look-up indices: multiway+big dimensions
- Evaluation
  - SCD: CloudETL 4 stmts/708 chars, Hive 112 stmts/4192 chars
  - SCD: Hive 4 times slower
  - Linear speed-up (32 cores)

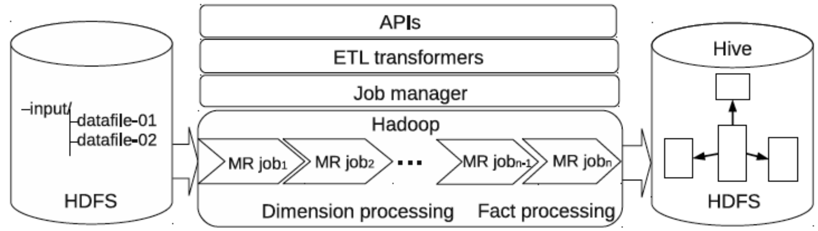


Figure 1: CloudETL Architecture

```
/* 1) Define the data source of page dimension */
Reader  pagesReader = new CSVFileReader("/user/cloudetl/input/pages")
                .setField("url", DataType.STRING, FieldType.BKEY)
                .setField("size", DataType.INT)
                .setField("moddate", DataType.DATE, FieldType.SCD_DATE);

/* 2) Create the transform pipe, and add the  transformation
      operators for cleansing data. */
TransformingReader pipe = new TransformingReader(pagesReader)
               .add(new ExcludeFields("size"))
                .add(new AddField("pageid", new Seq("pageid"), DataType.INT))
                .add(new RenameField("moddate", "validfrom"));

/* 3) Define the target  dimension table */
Writer pagedim = new SlowlyChangingDimensionWriter("/user/cloudetl/output",
                                                  "pagedim")
                .setField("pageid", DataType.INT, FieldType.PKEY)
                .setField("url", DataType.STRING, FieldType.LOOKUP)
                .setField("version", DataType.INT, FieldType.SCD_VERSION)
                .setField("validfrom", DataType.DATE, FieldType.SCD_VALIDFROM)
                .setField("validto", DataType.DATE, FieldType.SCD_VALIDTO);

/* 4) Add transformer and start ETL */
JobPlanner.addTransformer(pipe, pagedim).start();
```

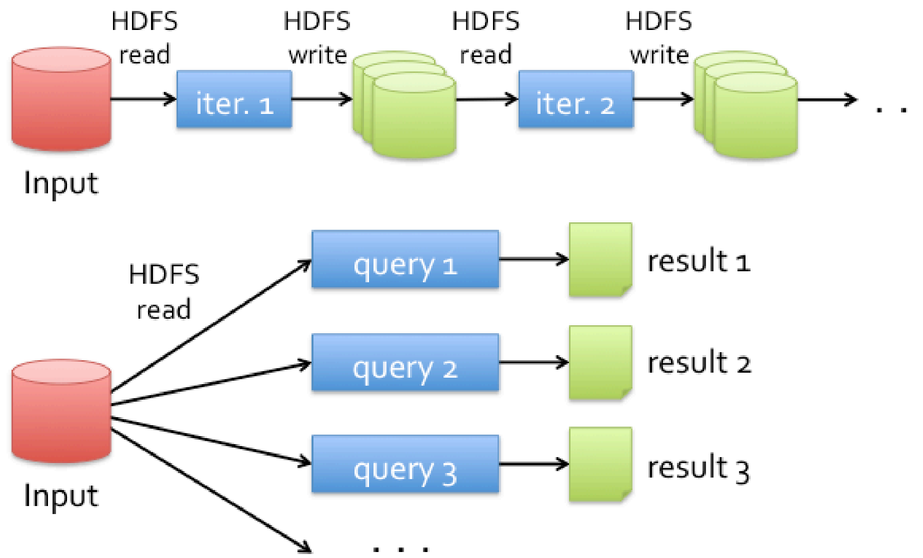
Figure 10: The ETL code for the SCD `pagedim`

```
/* 1) Define the fact data source */
DataReader  testResultsReader = new CSVFileReader("/user/cloudetl/input/testresults")
                        .setField("localfile", DataType.STRING)
                        .setField("url", DataType.STRING)
                        .setField("lastmoddate", DataType.DATE)
                        .setField("downloaddate", DataType.DATE)
                        .setField("test", DataType.STRING)
                        .setField("errors", DataType.INT);

/* 2) Do the necessary data transformation and look up dimension key values */
TransformingReader testresultsfactPipe = new TransformingReader(testResultsReader)
  .add(new ExcludeFields("localfile"))
  .add(new LookupTransformer("pageid", new SCDLookup(pagedim, "url", lastmoddate", -1)))
  .add(new LookupTransformer("dateid", new Lookup(datedim, "downloaddate", -1)))
  .add(new LookupTransformer("testid", new Lookup(testdim, "test", -1)));

/* 3) Define the target fact table */
 DataWriter testresultsfact = new FactTableWriter("/user/cloudetl/fact", "testresultsfact")
                        .setField("pageid", DataType.INT)
                        .setField("dateid", DataType.INT)
                        .setField("testid", DataType.INT)
                        .setField("errors", DataType.INT);

/* 4) Add transformer and start ETL */
JobPlanner.addTransfer(testresultsfactPipe, testresultsfact).start();
```

Figure 11: The ETL code for fact processing

Velocity: Typical Approach

- Everything in RAM
  - Avoid redundancy + disk intermediaries, recompute if necessary
- Apache Spark
  - Resilient Distributed Datasets (RDD's)
  - Operators on RDD's
- Pros
  - 10-100*faster
  - More productivity
- Cons
  - RAM expensive and limited
  - Standalone scenario
  - Misses some optimization potentials

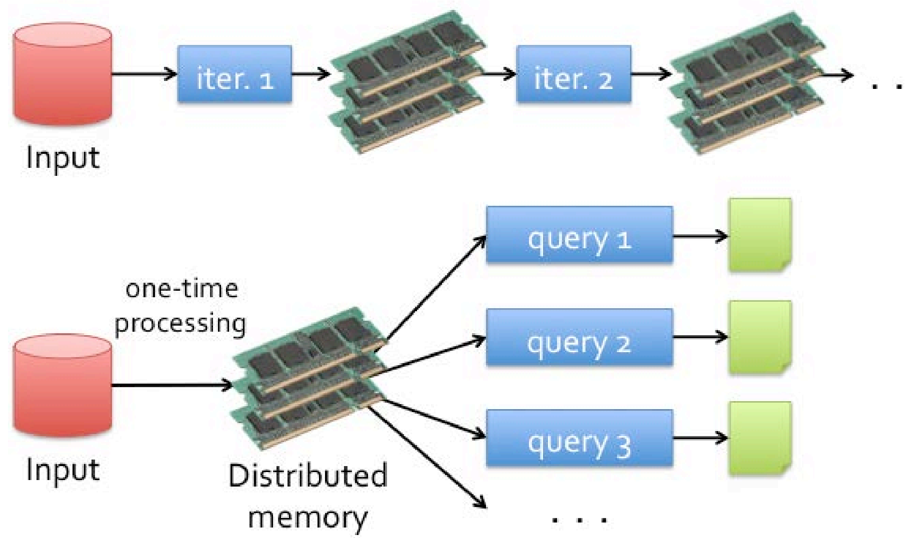


[amplab]

Hadoop/MapReduce data sharing

Velocity: Typical Approach

- Everything in RAM
  - Avoid redundancy + disk intermediaries, recompute if necessary
- Apache Spark
  - Resilient Distributed Datasets (RDD’s)
  - Operators on RDD’s
- Pros
  - 10-100*faster
  - More productivity
- Cons
  - RAM expensive and limited
  - Standalone scenario
  - Misses some optimization potentials



[amplab]

Spark data sharing

Vel: Other Scenarios/Optimizations

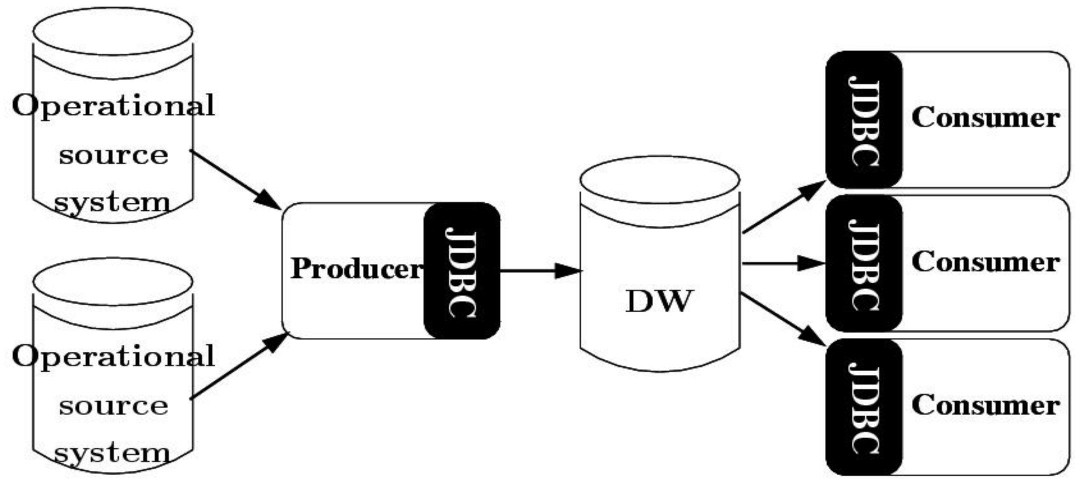
- Data should end up in in standard DBMS quickly
  - Where all the other enterprise data is
  - Allows combining high velocity data with existing enterprise data

- Integration of historical and predicted data
  - So fast it hasn’t even happened yet…

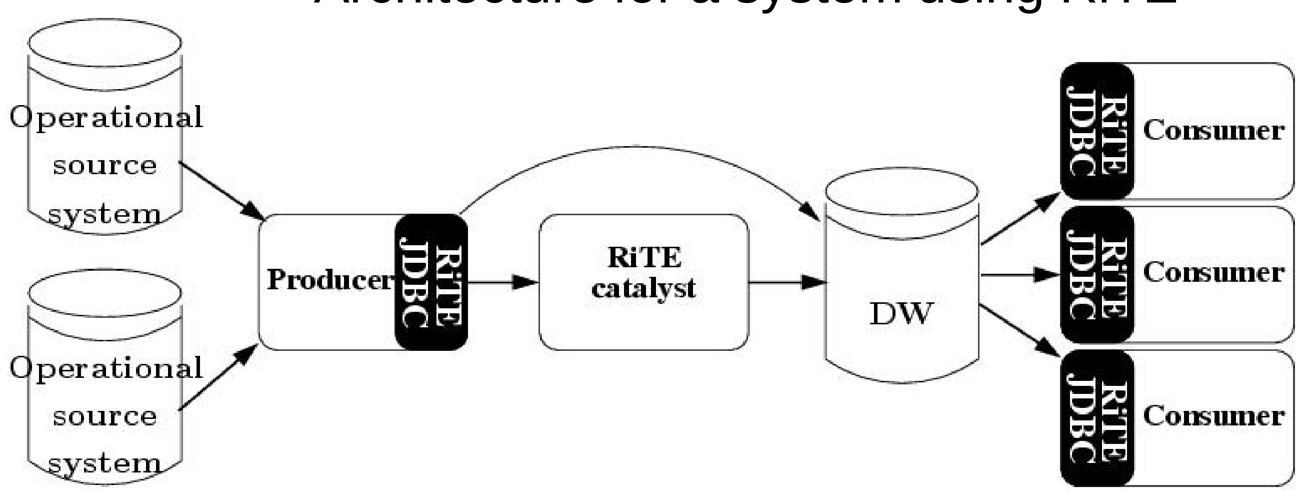
RiTE: Right-Time ETL

- INSERT/UPDATE like data availability with (88% of) bulk load speed
- PostgreSQL/JDBC prototype

Classical architecture



Architecture for a system using RiTE

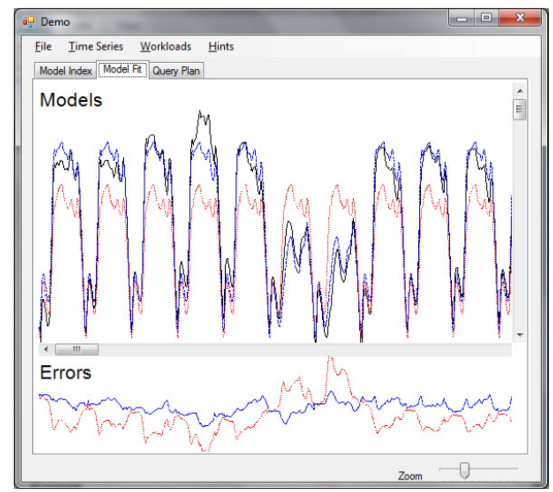


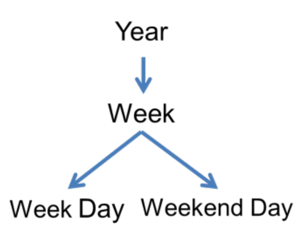
Velocity: Fast Energy Data

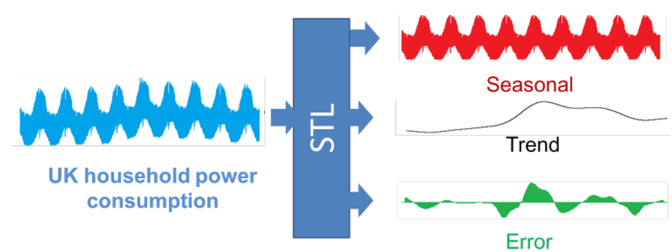
- Many time series (supply, demand, flexibility,…)
- Data start out in the **future**
  - Long term forecast, (more accurate) medium term forecast, (even more accurate) short term forecast, more and more accurate
- And finally make it to the **present**
  - Read actual data value from sensor and store it (*inaccuracy/delay*)
- …and into the **past**
  - Keep for long term analytics and as basis for re-forecasting
- Key observation:
  - **Only** difference btw. forecasted and ”real” data is level of accuracy
- Idea
  - Use (better and better) *models* to represent **all** data
  - *Model adaption* instead of loading (perhaps free ☺)

The TimeTravel System

- Past, future and combined (timetravel) queries
  - ”Show average consumption for today and tomorrow”
- Exact queries
  - ”Show average consumption for today and tomorrow” (using detailed time series values)
  - Future values are (of course) not ”exact” since they are forecasted
- Approximate queries (absolute or relative error w.r.t. detailed time series values)
  - ”Show average consumption for today and tomorrow with up to 5% error”
  - Potential for huge performance gains
- Hierarchical model index
  - Progressively lower error
- Time series: Seasonal,Trend,Error components
  - Period *hints* for seasonality, e.g., 1 or 2 seasonalities per week
- PostgreSQL based prototype
- Up to 2 orders of magnitude smaller/faster
- Query past+future seamlessly with SQL!

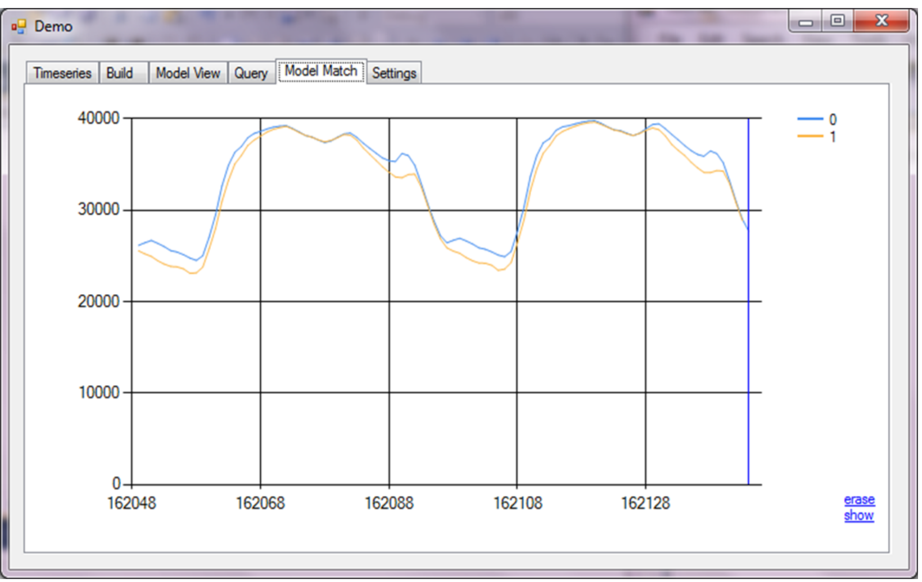


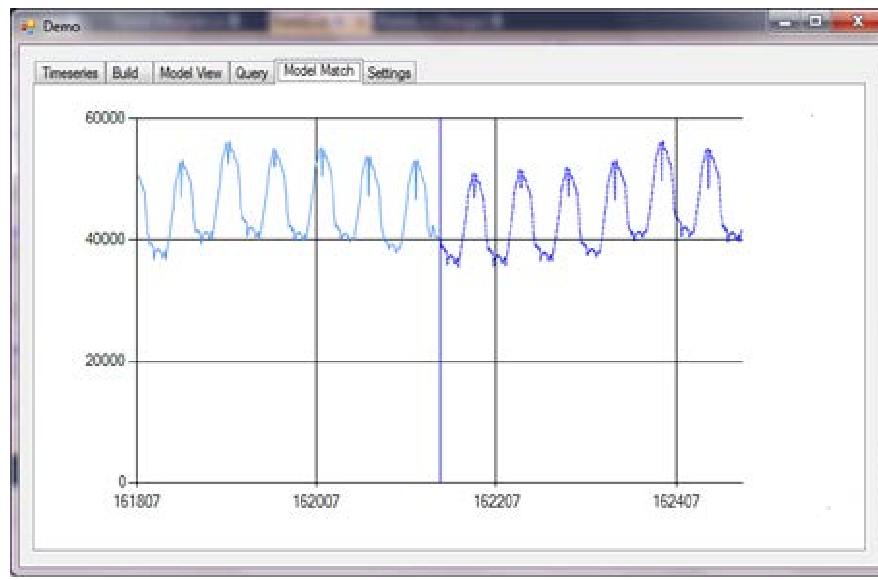




TimeTravel Queries

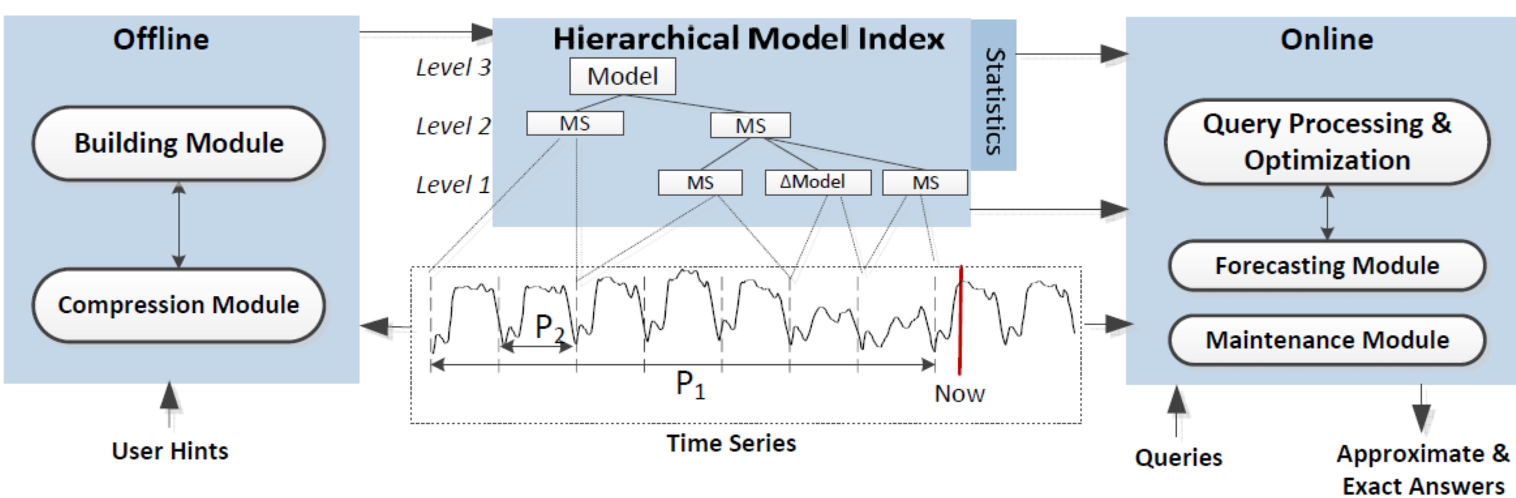
- ***(Past Query, Approximate) Show the 15-minutes power consumption for yesterday with an absolute max error of 100***
- SELECT TIME, POWER_CONSUMPTION FROM M_UK [-96,0] PINTERVAL=15-MIUNUTE ERROR=100;
- ***(TimeTravel Query) Find daily maximum power consumption over the last and next week:***
- SELECT MAX(POWER_CONSUMPTION) FROM M_UK [-336,+336];





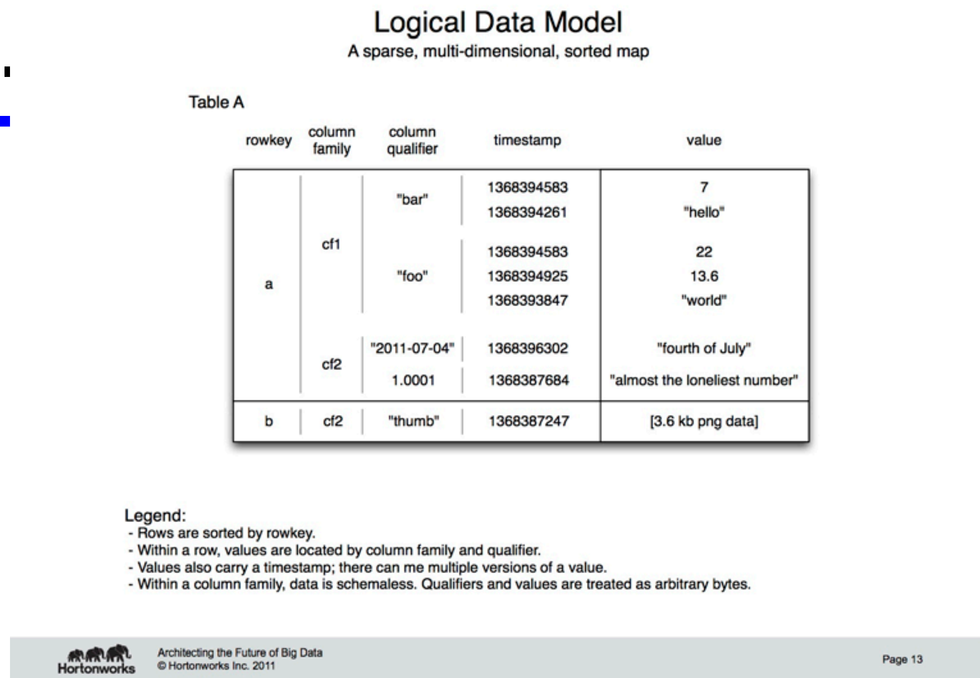
TimeTravel Architecture

- ***Building Module***. Hints+timeseries->hierarchical model index
- ***Compression Module.*** Reduce model storage by combining similar models
- ***Query Processing Module.*** Extends PostgreSQL processor/optimizer
  - Support approximate point range, aggregate and join queries
  - Traverses down the model index until required accuracy is reached.
- ***Forecasting Module***. Predicts future time series values, estimates error and confidence, re-estimates forecast method parameters.
- ***Maintenance Module***. Maintains hierarchical model with new time series values, adds new models to HMI or updates model parameters



Variety: Typical App.

- NoSQL
  - Get rid of schemas+SQL
- Key-value stores
  - BigTable, Hbase,…
- Pros:
  - Scalable
  - Fault tolerant (redundancy)
  - Fleksibelt
- Cons:
  - Consistency
  - Produktivity (no SQL)
  - Only for **some** scenarios



Logical Data Model

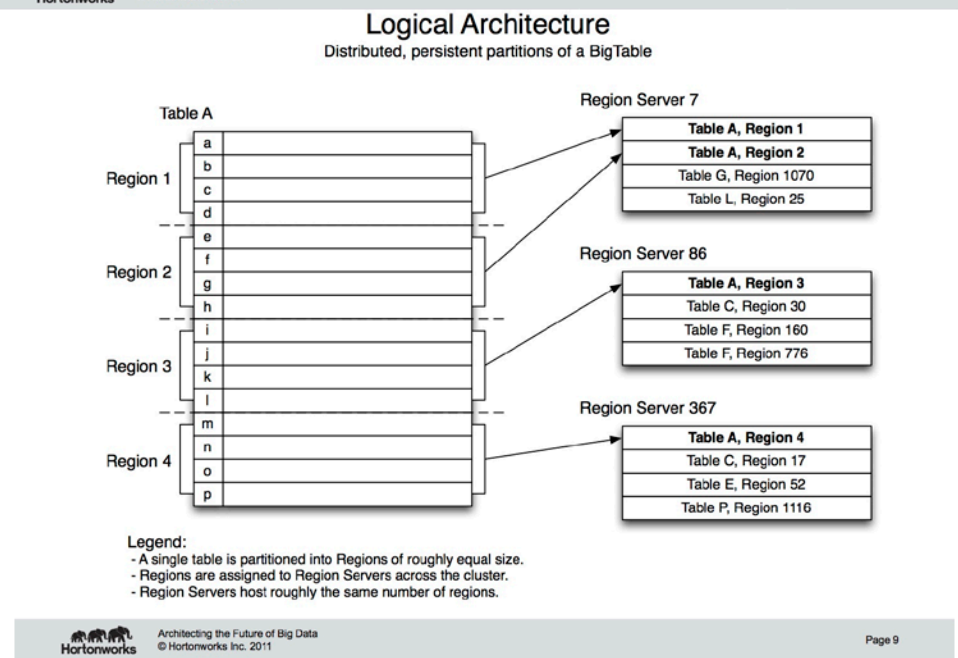
A sparse, multi-dimensional, sorted map

Table A

rowkey	column family	column qualifier	timestamp	value
a	cf1	"bar"	1368394583	7
			1368394261	"hello"
		"foo"	1368394583	22
			1368394925	13.6
			1368393847	"world"
	cf2	"2011-07-04"	1368396302	"fourth of July"
		1.0001	1368387684	"almost the loneliest number"
b	cf2	"thumb"	1368387247	[3.6 kb png data]

Legend:
- Rows are sorted by rowkey.
- Within a row, values are located by column family and qualifier.
- Values also carry a timestamp; there can me multiple versions of a value.
- Within a column family, data is schemaless. Qualifiers and values are treated as arbitrary bytes.

Hortonworks Architecting the Future of Big Data © Hortonworks Inc. 2011 Page 13



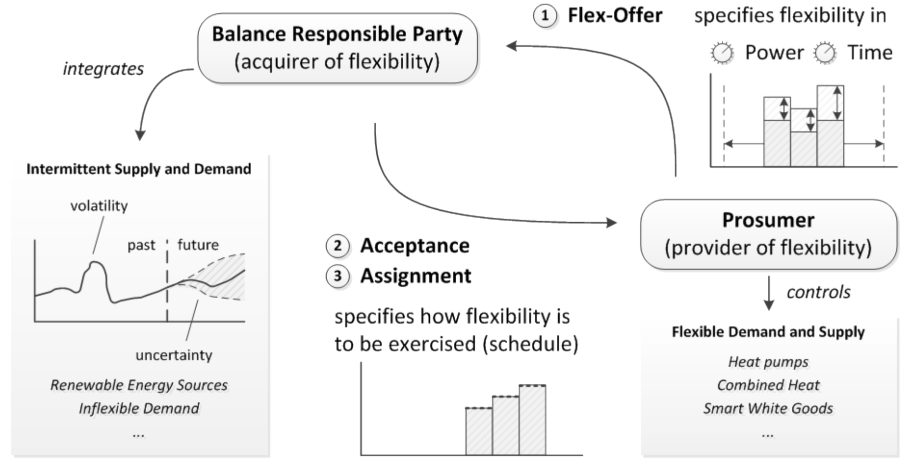
Variety: What?

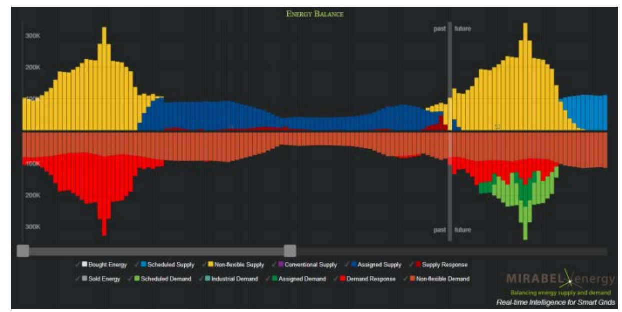
- A large part of the challenge is given by the application domain
  - Special types of data
  - Special queries
  - Complex data flows

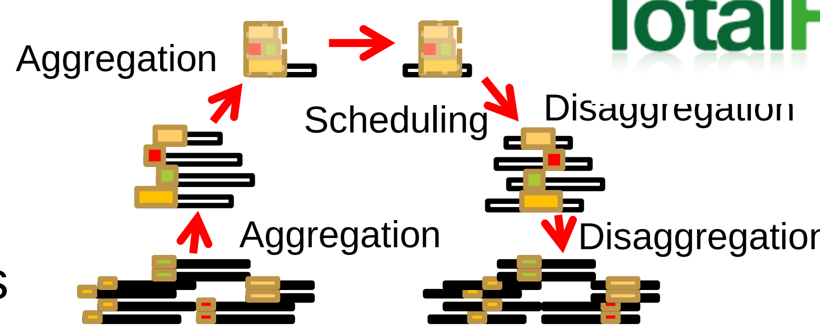
- Let us look at some examples

Variety: Big Energy Data

- Complex time series to be forecasted
- Collect/manage explicit flexibilities (flex-offers)
- Balance supply and demand in real time
- Predict production, consumption, flexibility down to device level
- Results so far:
  - Peak load 13-50% smaller
  - 15% flexibility neutralizes 70% of renewables impact
  - 10-20% cheaper, 50% less CO_2 – towards 100%!

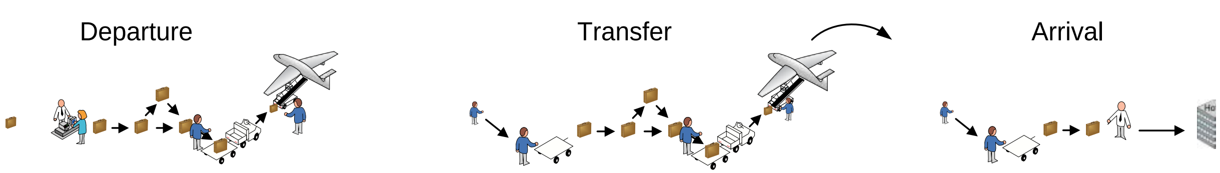






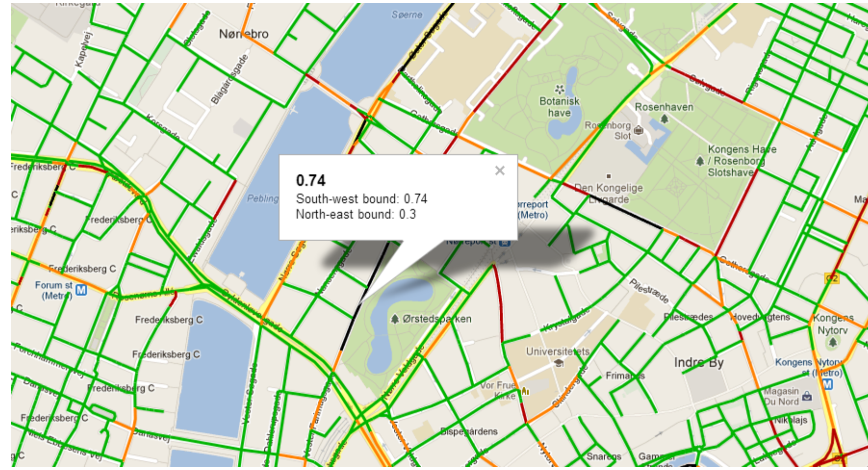
Variety: Big RFID Data

- ”BagTrack – styr på bagagen”
  - Daisy, Lyngsoe, SAS (**Arlanda**!), IATA, AAL - **app**
  - Bag tags w. RFID – remote reading
  - License plate (ID), route, date
  - Vision: real-time world-wide baggage info in 2020: 50% less baggage problems, save 1.2 bio. US$/year
- Daisy Big Data research
  - Data cleansing – get true meaning from RFID read
  - Real-time data and queries
  - OLAP/DW – analyze processes and measurement
  - Data mining: problems/causes in event sequences
  - Big/complex data, 1000+ airports

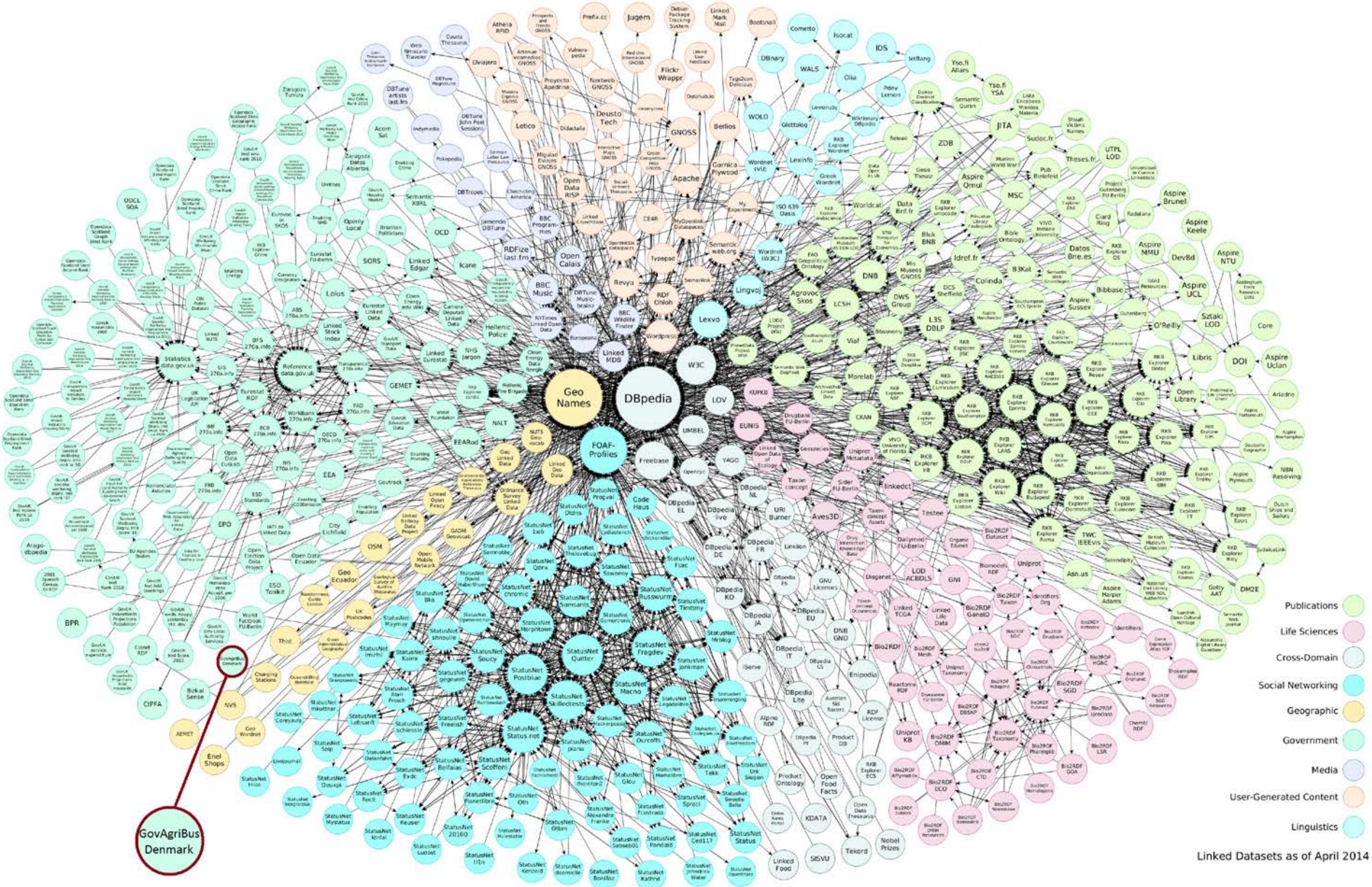


Variety: Big Transport/GPS Data

- Mostly Daisy colleagues
- Understand and measure environmental impact
- Assign time-varying eco-weights to road segments
- Eco-routing: fuel-saving route planning – used by Danish flex taxies today
- BI/analytics: find and predict locations, routes, rideshares,…

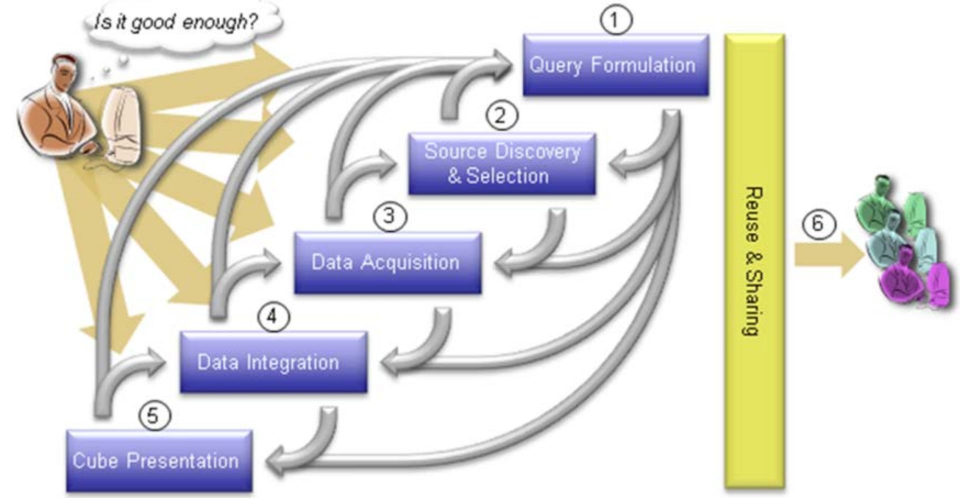


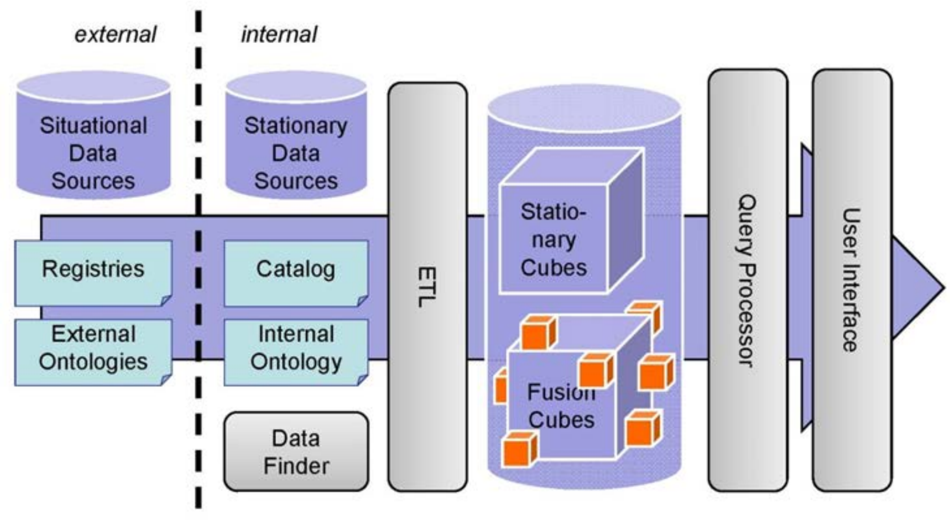
Variety: Open/Linked Data



Fusion Cubes: BI on Linked Data

- Need for **external data**
  - Format/meaning/queries?
- Solution: Semantic Web
  - Formal **ontologies**
  - **Link** to other ontologies/concepts (Linked Data)
  - SPARQL queries
- Self-service BI
- BI solution ”grown” gradually (not built)
- Share dimensions, transformations, results
- OLAP+ETL on distributed/ spatiotemp. Linked Data

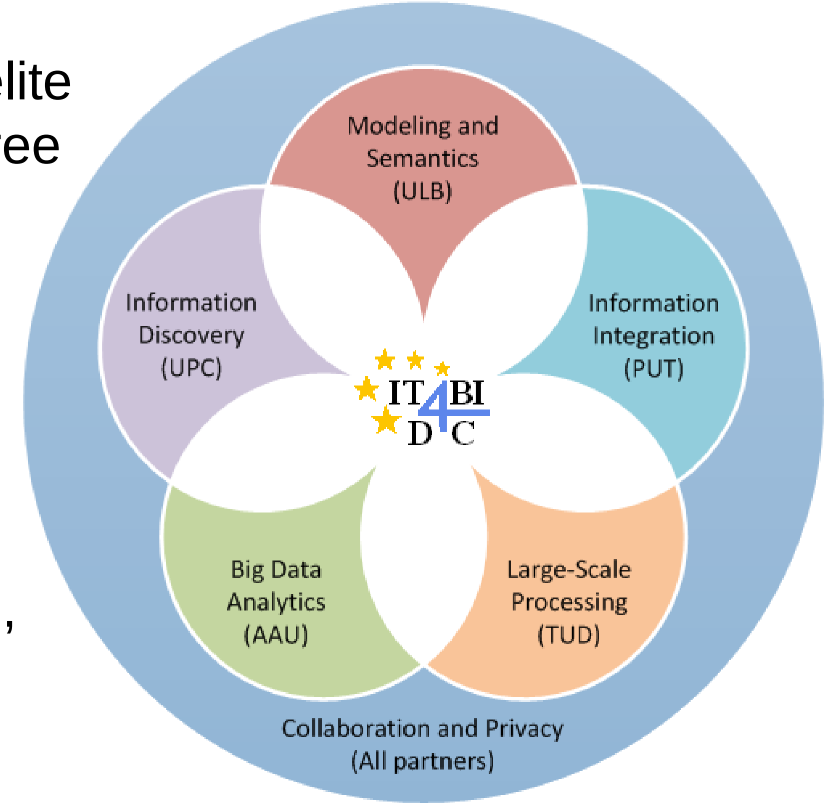




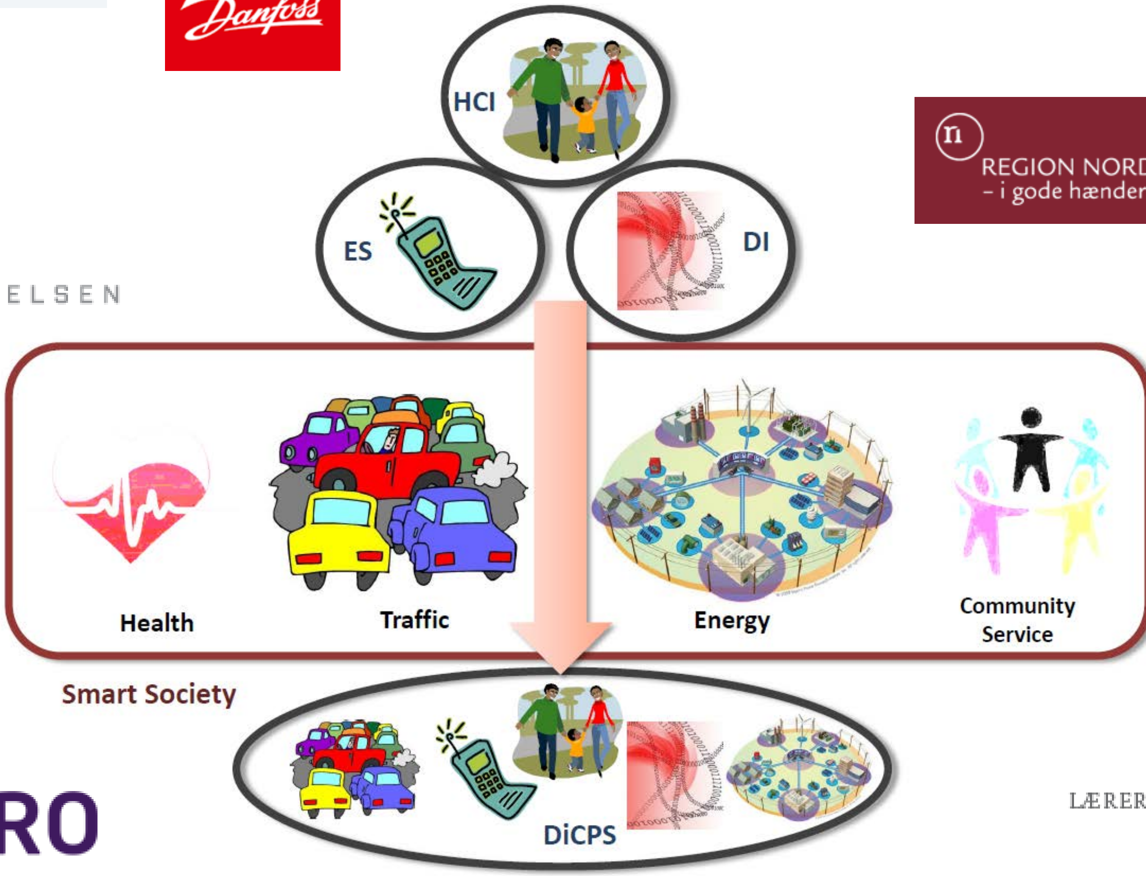
A. Abello et al. *Fusion Cubes: Towards Self-Service Business Intelligence*. IJDWM 9(2), 2013.

IT4BI-DC: The Big MD Data Ph.D.

- IT Technologies for Business Intelligence – Doctoral College
  - Erasmus Mundus Joint Doctorate elite doctoral program – joint Ph.D. degree
  - Up to 100 Ph.D.s in BI/Big Data
  - Daisy leads Big Data Analytics
- Partners
  - ULB Bruxelles, AAU, TU Dresden, UPC Barcelona, Poznan UT
  - SAP, Microsoft Research, HP Labs, Oracle Labs, LBNL, HKU, UQ, …
- See https://it4bi-dc.ulb.ac.be



Next Step: Data-intensive CPS



Integrate models of processes, data, and users in energy, transport,…

Summary

- Big (Multidimensional) Data
  - Volume, Velocity, Variety…and more V's
  - (Partly) novel
  - Useful for explaining/characterizing data management research
- Volume
  - More iron helps, but is not enough
  - Efficiency (PLWAH), Productivity: ETLMR, CloudETL
- Velocity
  - RAM helps, but is not enough
  - DBMS integration (RiTE), Forecast integration (TimeTravel)
- Variety
  - noSQL is not enough
  - Energy, transport, RFID, Linked Data…: domain knowledge needed
- Bottom line: good job security for data geeks☺

Key References

- http://people.cs.aau.dk/~tbp
- Pedersen, *Managing Complex Multidimensional Data*, eBISS 2012, Springer LNBIB
- Deliège et al. *Position list word aligned hybrid: optimizing space and performance for compressed bitmaps*. EDBT 2010
- Algorhyme: www.algorhyme.com
- Liu et al. *MapReduce-based Dimensional ETL Made Easy*. PVLDB 5(12), 2012
- Liu et al. *ETLMR: A Highly Scalable Dimensional ETL Framework Based on MapReduce*. TLDKS 8, 2013
- Liu et al. *CloudETL: scalable dimensional ETL for Hive*. IDEAS 2014: 195-206
- Thomsen et al. *RiTE: Providing On-Demand Data for Right-Time Data Warehousing*. ICDE 2008.
- Khalefa et al. *Model-based Integration of Past & Future in TimeTravel*. PVLDB 5(12), 2012
- Böhm et al. *Data management in the MIRABEL smart grid system*. EDBT/ICDT WS 2012
- Pedersen: Energy Data Management: Where Are We Headed? EDBT/ICDT Workshops'14
- Abello et al. Fusion Cubes: Towards Self-Service Business Intelligence. IJDWM 9(2), 2013
- Ahmed et al. *A Data Warehouse Solution for Analyzing RFID-Based Baggage Tracking Data*. MDM 2013
- IT4BI-DC: http://it4bi-dc.ulb.ac.be

Acknowledgements

- Some slides borrowed from colleagues and collaborators