QAWARE
SOFTWARE ENGINEERING

Apache Solr as a compressed, scalable, and high performance time series database

FOSDEM 2015

Florian Lautenschlager
31. January 2015
FOSDEM 2015, Brussels

68.000.000.000* time correlated data objects.

How to store such amount of data on your laptop computer and retrieve any point within a few milliseconds?

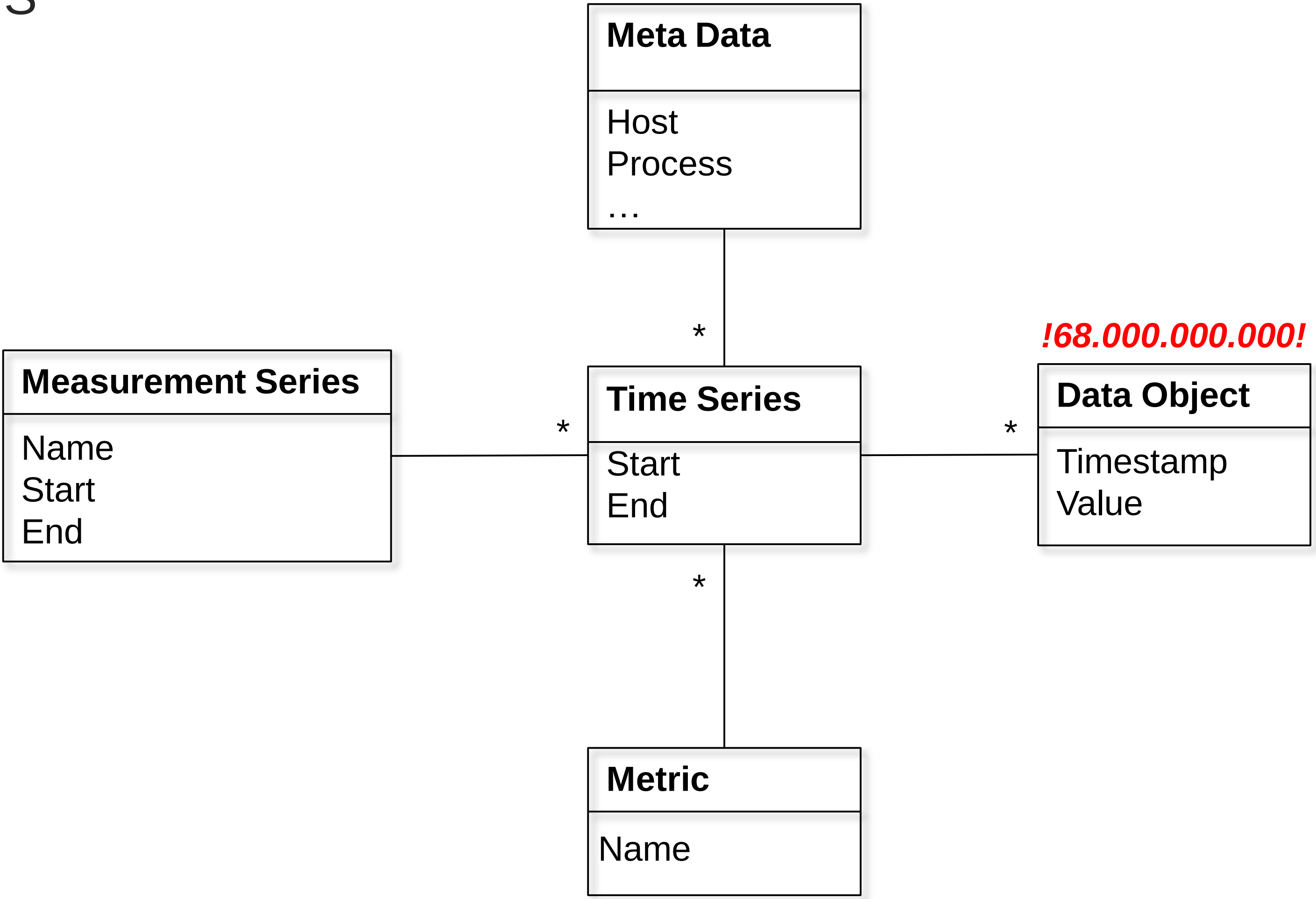
* or collect and store 680 metrics x 500 processes x 200 hosts over 3 years

This approach does not work well.

Scales up to 1 million data objects.

- Store data objects in a classical RDBMS

- Reasons for us:
  - Slow import of data objects
  - Hugh amount of hard drive space
  - Slow retrieval of time series
  - Limited scalability due to RDBMS



Approach felt like …

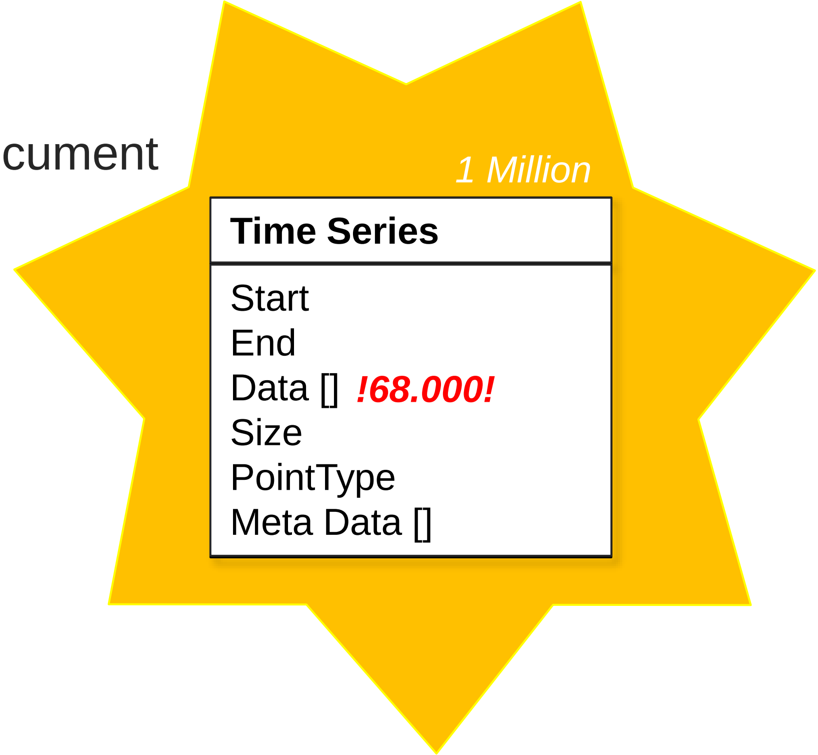
Not sure whether bad driver or wrong car!?

Nathan Wong,http://upload.wikimedia.org/wikipedia/commons/e/e7/Rowan_Atkinson_on_a_Mini_at_Goodwood_Circuit_in_2009.jpg

Changed the car and the driver… and it works!

Scales up to X billion data objects.

- **The key ideas to enable the efficient storage of billion data objects:**
  - Split data objects into chunks of the same size
  - Compress these chunks to reduce the data volume
  - Store the compressed chunks and the metadata in one Solr document
- Reason for success:
  - 37 GB disk usage for 68 billion data objects
  - Fast retrieval of data objects within a few milliseconds
  - Searching on metadata
  - Everything runs on a laptop computer
  - … **and many more!**



1 Million

Time Series

Start
End
Data [] ***!68.000!***
Size
PointType
Meta Data []

That‘s all.
No secrets, nothing special and nothing more to say ;-)

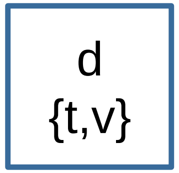
Hard stuff - Time for beer!

The agenda for the rest of the talk.

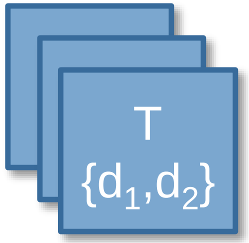
- Time Series Database - What’s that? Definitions and typical features.
- Why did we choose Apache Solr and are there alternatives?
- How to use Apache Solr to store billions of time series data objects.

Time Series Database: What’s that?

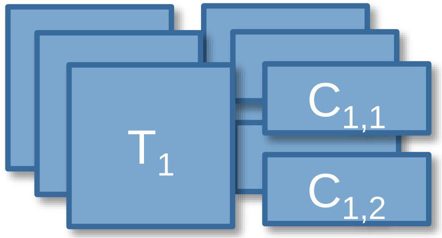
- *Definition 1*: “A data object *d* is a 2-tuple of *{timestamp, value}*, where the value **could be any kind of object**.”



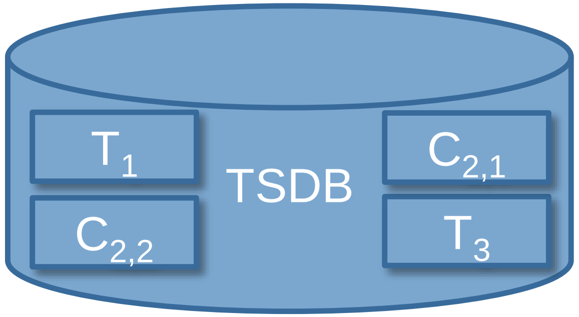
- *Definition 2*: “A time series *T* is an arbitrary list of **chronological ordered** data objects of **one value type**”



- Definition 3: “A **chunk** *C* **is a** chronological ordered **part** of a time series.”



- *Definition 3*: “A time series database *TSDB* is a specialized database for **storing and retrieving** time series in an **efficient and optimized** way”.

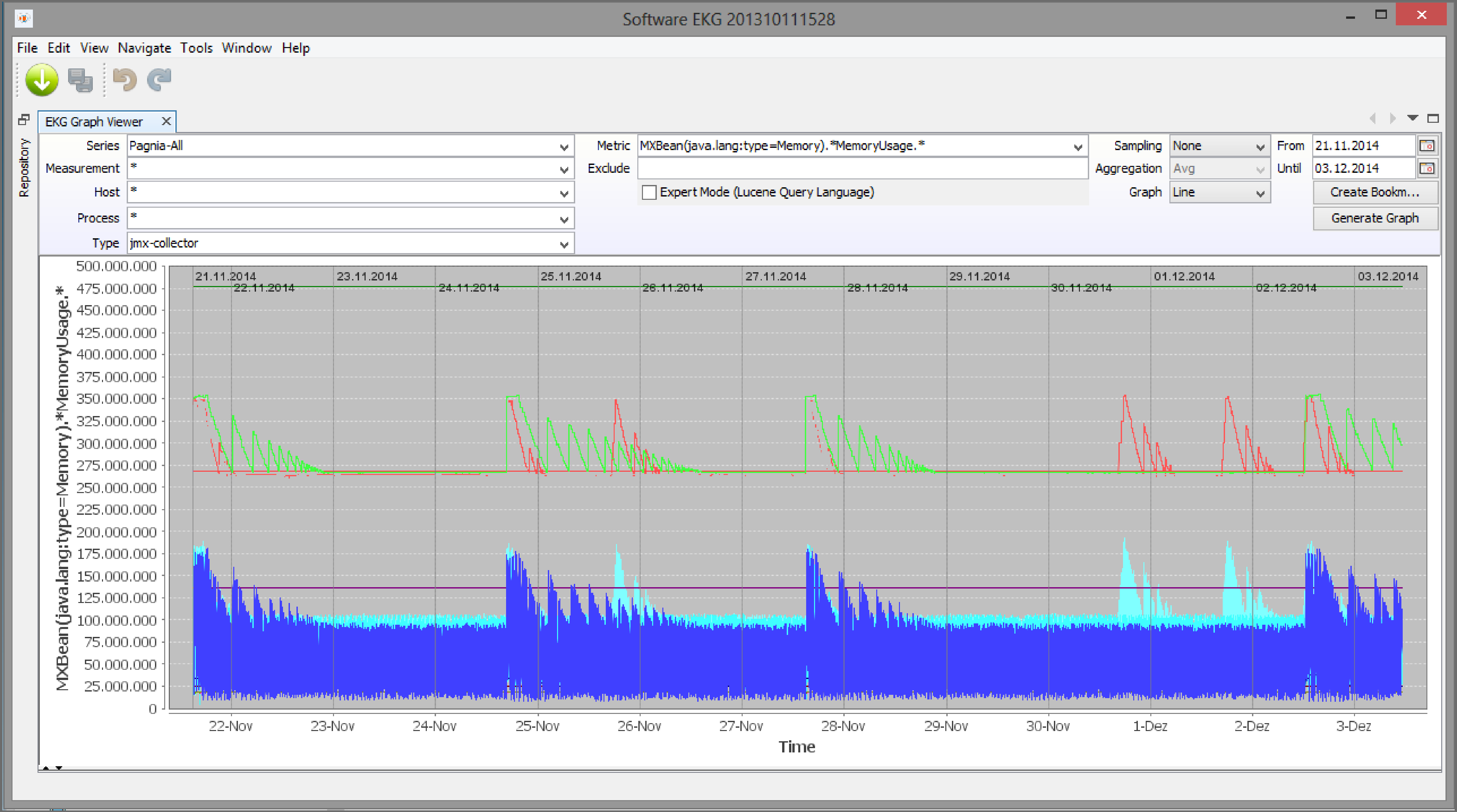


A few typical features of a time series database

- Data management
  - Round Robin Storages
  - Down-sample old time series
  - Compression
- Arbitrary amount of Metadata
  - For time series (Country, Host, Customer, …)
  - For data object (Scale, Unit, Type)
- Performance and Operational
  - Rare updates, Inserts are additive
  - Fast inserts and retrievals
  - Distributed and efficient per node
  - No need of ACID, but consistency
- Time series language and API
  - Statistics: Aggregation (min, max, median), …
  - Transformations: Time windows, time shifting, resampling, ..

Check out: A good post about the requirements of a time series: http://www.xaprb.com/blog/2014/06/08/time-series-database-requirements/

That’s what we need the time series database for.



Series: Pagnia-All

Measurement: *

Host: *

Process: *

Type: jmx-collector

Metric: MXBean(java.lang:type=Memory).*MemoryUsage.*

Exclude:

☐ Expert Mode (Lucene Query Language)

Sampling: None

Aggregation: Avg

Graph: Line

From: 21.11.2014

Until: 03.12.2014

Some time series databases out there.

- RRDTool - http://oss.oetiker.ch/rrdtool/
  - Mainly used in traditional monitoring systems
- InfluxDB - http://influxdb.com/
  - The new kid on the block. Based on LevelDB
- OpenTSDB - http://opentsdb.net/
  - Is a scalable time series database and runs on Hadoop and Hbase
- SciDB - http://www.scidb.org/
  - Is computational DBMS and is programmable from R & Python
- … many more

“Ey, there are so many time series databases out there? Why did you create a new solution? Too much time?”

“Our tool has been around for a ere was no time series database that complies our r

Alternatives?
In our opinion the best alternative is **ElasticSearch**.
Solr and ElasticSearch are both based on Lucene.

Our Requirements	**he Solr**	
A fast write and query performance	ased on Lucene which is really fast	✓
Run the database on a laptop computer	Runs embedded or as standalone server	✓
Minimal data volume for stored data objects	Lucene has a build in compression	✓
Storing arbitrary metadata	Schema or schemaless	✓
A Query API for searching on all information	Solr Query Language	✓
Large community and an active development	Lucidworks and an Apache project	✓

Apache Solr

Many more!

Solr has a powerful query language that enriches the Lucene query language.

- An example for a complex query:

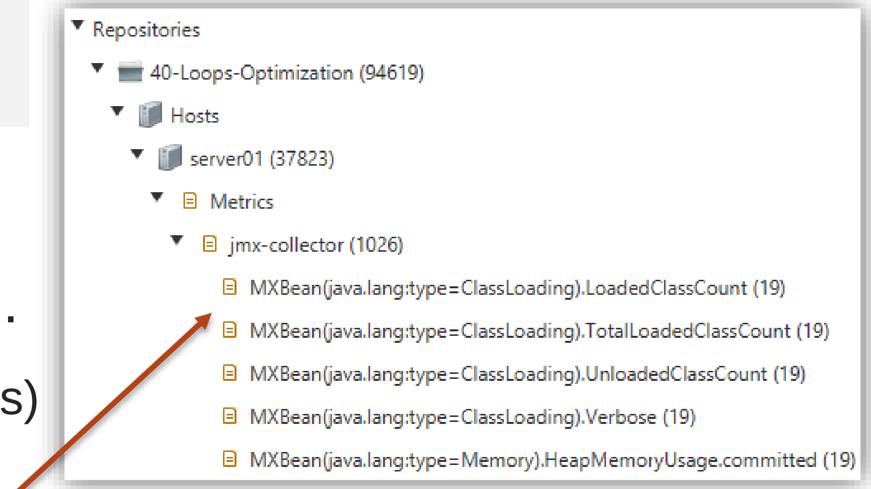
```
host:h* AND metric:*memory*used AND -start:[NOW - 3 DAYS] OR -end:[NOW + 3 DAYS]
```

- A few powerful Solr query language features
  - Wildcards: *host*:server**?**1 (single) and *host*:server***** (multiple characters)
  - Boolean operators: *conference*:FOSDEM **AND** *year*:(2015 **||** 2016) **NOT** *talk*:"Time series in RDBMS"
  - Range queries: *zipCode*: **[123 TO *]**
  - Date-Math: *conferenceDate*:**[* TO NOW],** *conferenceDate:***[NOW-1YEAR/DAY TO NOW/DAY+1DAY]**
  - Boosting of terms: "I am a four times boosted search term"**^4**, "I am just normal search term"
  - … -> https://cwiki.apache.org/confluence/display/solr/Query+Syntax+and+Parsing

Fast navigation over time series metadata is a must-have when dealing with billions of data objects.

- Solr has a powerful query language which allows complex wildcard expressions

```
series:40-Loops-Optimzation AND host:server01
AND process:* AND type:jmx-collector
```



Repositories
40-Loops-Optimization (94619)
Hosts
server01 (37823)
Metrics
jmx-collector (1026)
MXBean(java.lang:type=ClassLoading).LoadedClassCount (19)
MXBean(java.lang:type=ClassLoading).TotalLoadedClassCount (19)
MXBean(java.lang:type=ClassLoading).UnloadedClassCount (19)
MXBean(java.lang:type=ClassLoading).Verbose (19)
MXBean(java.lang:type=Memory).HeapMemoryUsage.committed (19)

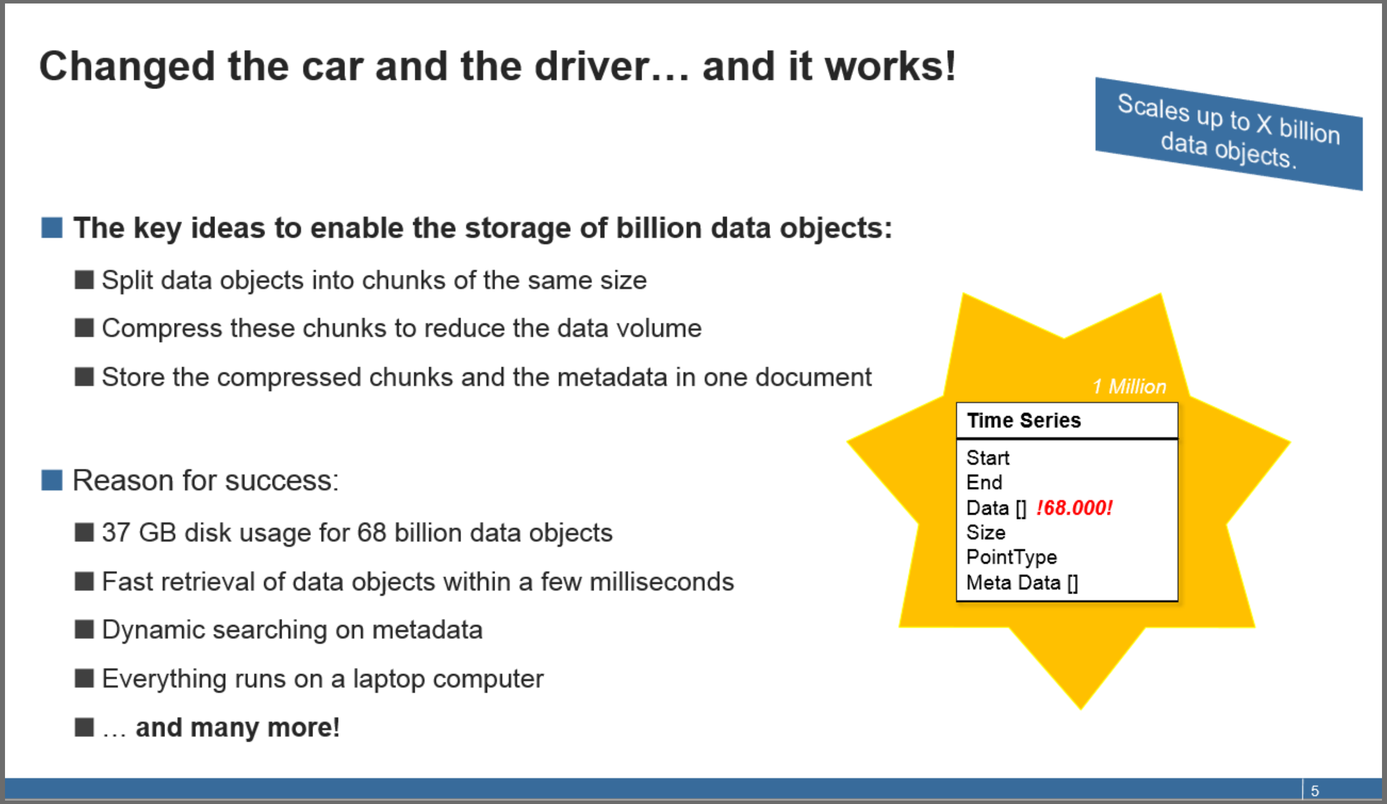
- The faceting functionality allows a dynamic drilldown navigation.
  - Faceting is the arrangement of search results into categories (Facets) based on indexed terms

```
QueryResponse response = solr.query(query);
FacetField field = response.getFacetField(SolrSchema.IDX_METRIC);
List<FacetField.Count> count = field.getValues();

if (count == null) {return Stream.empty();}
return count.stream().filter(c ->
    c.getCount() != 0).map(c -> new Metric(c.getName().substring(1),c.getCount()));
```

Many slides later…

…we are continuing from slide five.



Changed the car and the driver… and it works!

Scales up to X billion data objects.

- **The key ideas to enable the storage of billion data objects:**
  - Split data objects into chunks of the same size
  - Compress these chunks to reduce the data volume
  - Store the compressed chunks and the metadata in one document
- Reason for success:
  - 37 GB disk usage for 68 billion data objects
  - Fast retrieval of data objects within a few milliseconds
  - Dynamic searching on metadata
  - Everything runs on a laptop computer
  - … **and many more!**

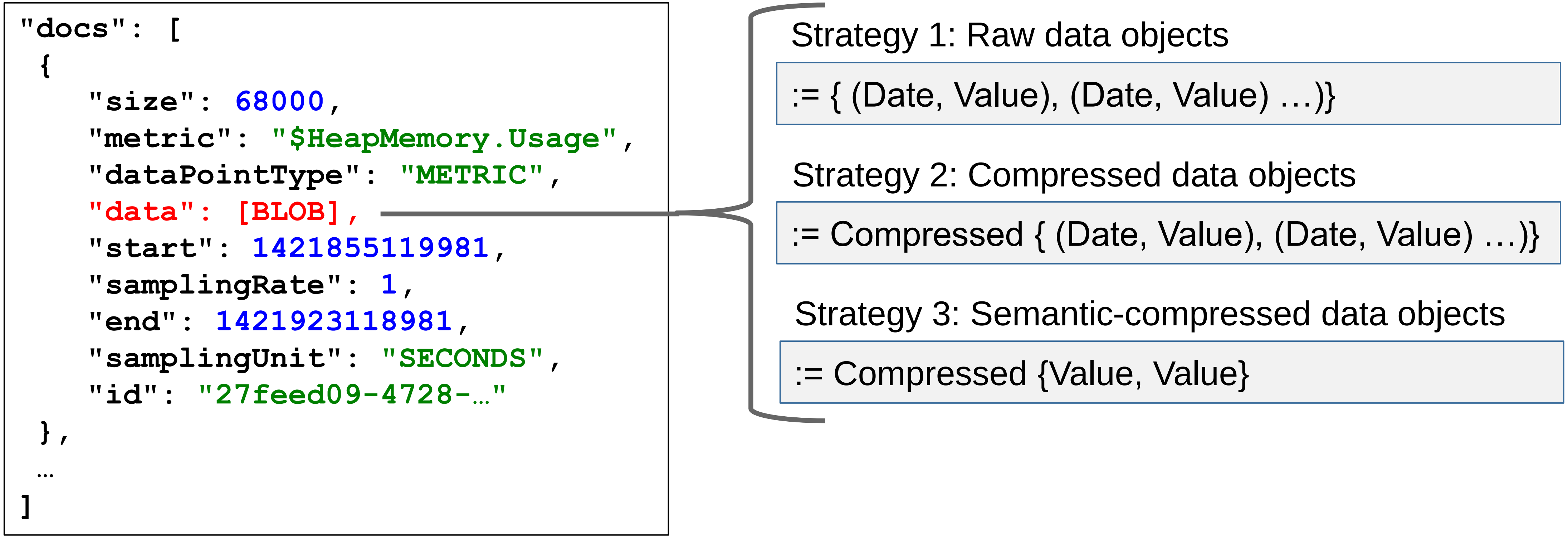
1 Million

Time Series

Start
End
Data [] ***!68.000!***
Size
PointType
Meta Data []

First: Do not store data object by data object by data object by...

- **Do not** store 68 billion single documents. Do **instead** store **1.000.000** documents each containing **68000** data objects as BLOB.



```
"docs": [
 {
    "size": 68000,
    "metric": "$HeapMemory.Usage",
    "dataPointType": "METRIC",
    "data": [BLOB],
    "start": 1421855119981,
    "samplingRate": 1,
    "end": 1421923118981,
    "samplingUnit": "SECONDS",
    "id": "27feed09-4728-…"
 },
 …
]
```

Strategy 1: Raw data objects

:= { (Date, Value), (Date, Value) …)}

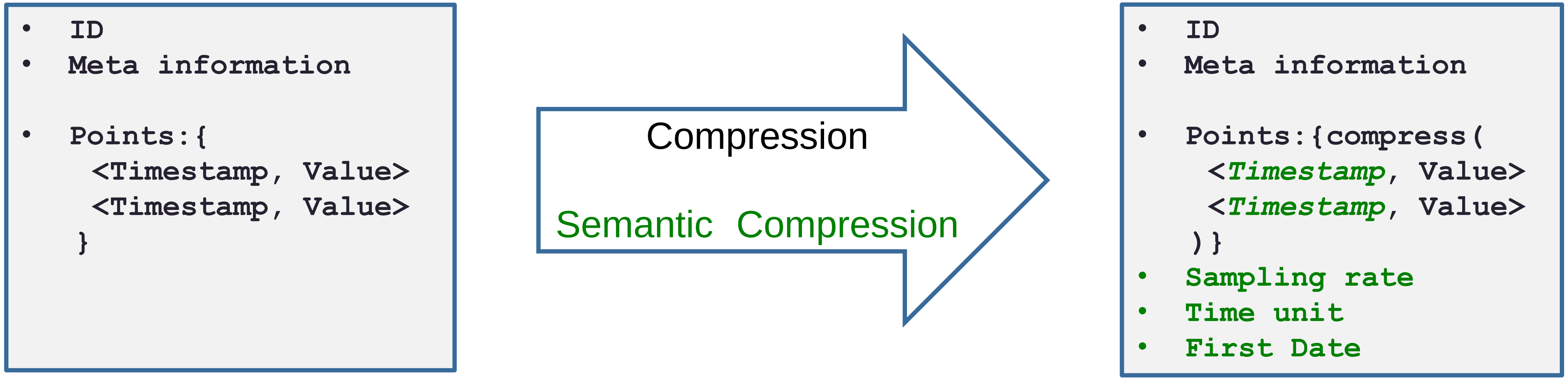
Strategy 2: Compressed data objects

:= Compressed { (Date, Value), (Date, Value) …)}

Strategy 3: Semantic-compressed data objects

:= Compressed {Value, Value}

Don’t store needless things. Two compression approaches.



```
•  ID
•  Meta information

•  Points:{
    <Timestamp, Value>
    <Timestamp, Value>
   }
```

Compression

Semantic Compression

```
•  ID
•  Meta information

•  Points:{compress(
    <Timestamp, Value>
    <Timestamp, Value>
   )}
•  Sampling rate
•  Time unit
•  First Date
```

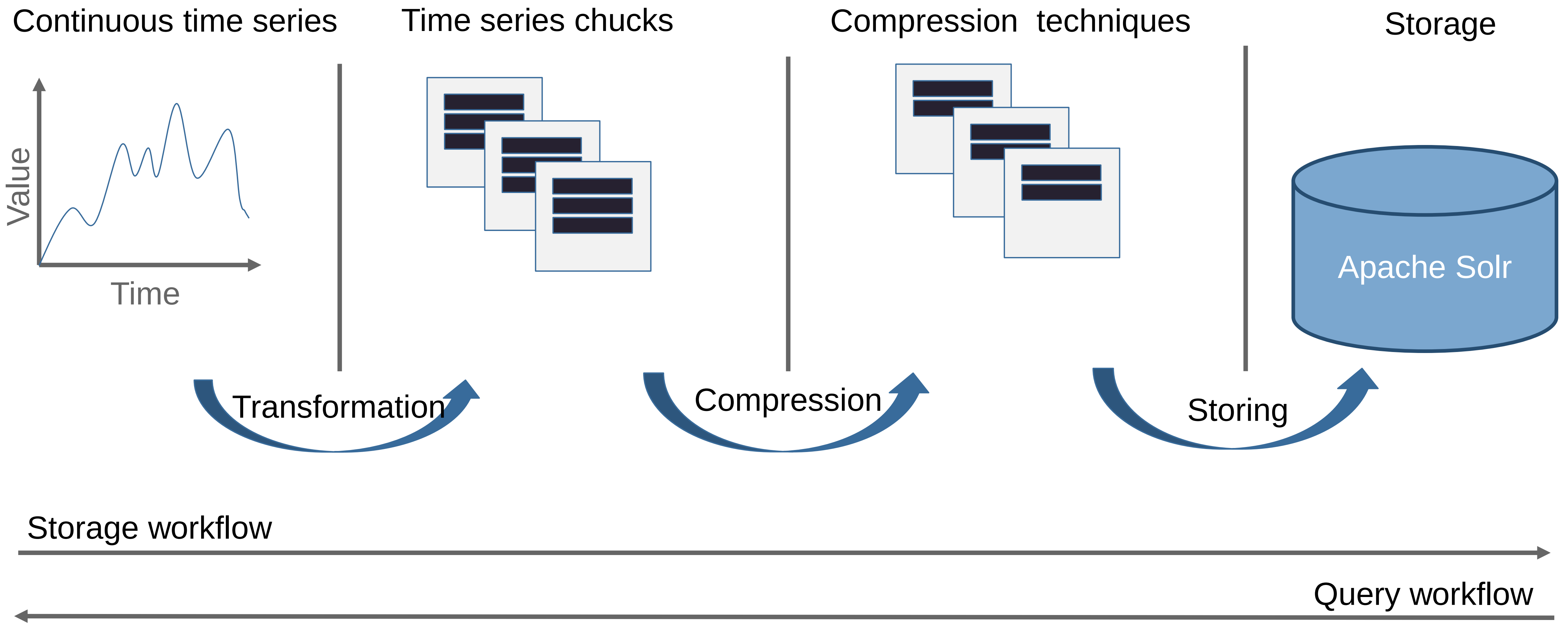
- Strategy 2: Basic compression with GZIP, lz4, …
  - Works for every data object and the compression rate is higher, if the document has more data objects

    := Compressed { (Date, Value), (Date, Value) …)}

- Strategy 3: Semantic compression by only storing the algorithm to create the timestamp
  - Works only on time series with a fixed time interval between the data objects (Sampling, …)

    := Compressed {Value, Value} + First Date + Sampling Rate + Time Unit

Second: Correct handling of continuous time series in a document oriented storage.



Continuous time series

Time series chucks

Compression techniques

Storage

Apache Solr

Transformation

Compression

Storing

Storage workflow

Query workflow

Solr allows server-side decompression and aggregation by implementing custom function queries.

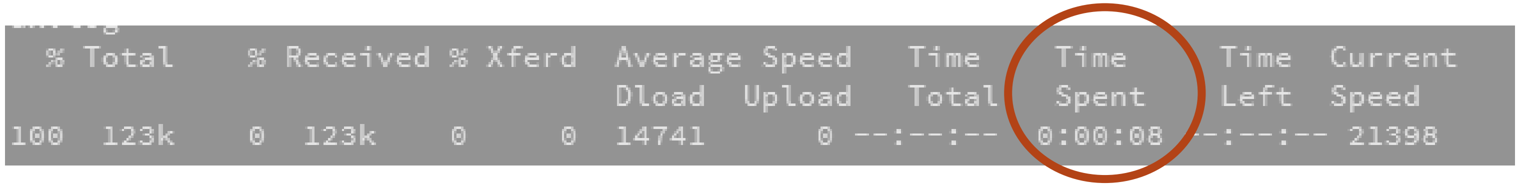
- Why should we do that? **Send the query to the data!**
  - Aggregation should be done close to the data to avoid unnecessary overhead for serialization, transportation and so on.
  - A *function query* enables you to create server-side dynamic query-depending results and use it in the query itself, sort expressions, as a result field, …

Our ValueSourceParser

- Imagine you want to check the maximum of all time series in our storage

```
http://localhost:8983/core/select?q=*:*&fl=max(decompress(data))
```

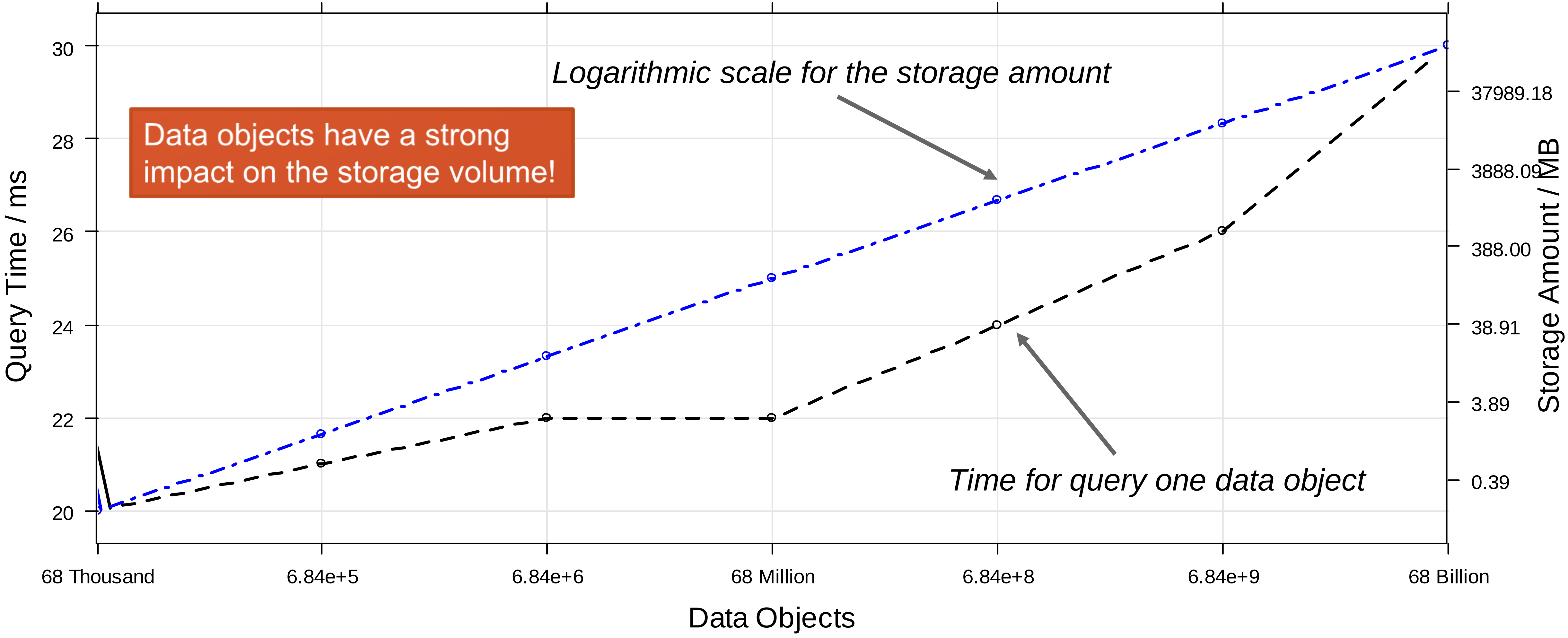
- And now get your own impression.



```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  123k    0  123k    0     0  14741      0 --:--:--  0:00:08 --:--:-- 21398
```

68.400.000 data objects in 1000 documents and each has 86400 Points.

Third: Enjoy the outstanding query and storage results on your laptop computer.



Data objects have a strong impact on the storage volume!

Logarithmic scale for the storage amount

Time for query one data object

Our present for the community:
The storage component including the Query-API

(currently nameless, work in progress)

We are done!

- We are planning to publish the Query-API and its storage component on GitHub.
  - Interested? Give me a ping: **florian.lautenschlager@qaware.de**

- Excessive use of Java 8 Stream API
- Time Shift, Fourier Transformation, Time Windows and many more
- Groovy DSL based on the fluent API (concept)
- Optional R-Integration for higher statistics

```
QueryMetricContext query = new QueryMetricContext.Builder()
     .connection(connection)
     .metric("*fosdem*visitor*statistics*delighted.rate")
     .build();

Stream<TimeSeries> fosdemDelightedStats = new AnalysisSolrImpl(query)
    .filter(0.5, FilterStrategy.LOWER_EQUALS)//Delighted visitors
    .timeFrame(1, ChronoUnit.DAYS)//on each day
    .timeShift(1, ChronoUnit.YEARS)//and next year
    .result();
```

Questions?