# Archiving System for FAIR

## Concept

Vitaliy Rapp

# Scope

*Collect and store pertinent accelerator data centrally to permit analysis of the accelerator performance and its proper function*

# Agenda

- Scope
- Requirements
- Technical Overview
- Roadmap

# Requirements

# Requirements

- **Long term archive of data over the life-cycle of the accelerator facility**
- **Free and reusable configuration of data, which needs to be archived**
  - default configuration profile
  - adjustable and selectable predefined configuration profiles
  - custom user configuration profiles
- **Supporting functions to query and correlate archived data**
- **Automatic and configurable reduction or deletion of archived data**
- **Configurations for reduction and deletion must be storable for later re-use**
- **The Archiving System must offer an API**
  - to write data into the archive database
  - to read data from the archive database
  - to help correlate data read from the archive database
- **GUI to configure the system, but also display and find correlated data from the archive**

# Requirements (cont.)

- **Export of (correlated) archived data in a generic format to support further post-processing using third-party applications and scripts**

- **Storing of meta-data that can be used to index, preselect and filter the data to query from the database. Some of the meta-data include:**
  - targeted beam parameter
  - machine parameter
  - accelerator & beam modes
  - whether PM trigger have been issued
  - operational ranges (OP day, week, time between technical stop)
  - important operational markers (start/end operation, start/end technical stop, …),

- **Load management and monitoring of the Archiving System infrastructure**

- **User access management (kicking-out, limiting of piggy clients, bandwidth shaping/scheduling, ...**

# Requirements (open)

- **Acting as Data Proxy**
  - Archiving System shall proxy and relay the already gathered online information to be stored into the database to other users and online applications. This would alleviate the load on the front-end controller that may not serve as many users as the Archiving System.

- **Archiving System data exposure to non-accelerator network users**

# Some Numbers*

* Copy from here: 20150603_FCWG_Archiving_and_PM

| | | SIS18 | SIS100 | CR | HESR | HEBT | Super-FRS | Total |
|---|---|---|---|---|---|---|---|---|
| Total variables: | | 166 | 436 | 129 | 129 | 912 | 260 | |
| proposed sampling | [Hz] | 1000 | 100 | 10 | 10 | 1 | 1 | |
| Data rate | [MB/s] | 6.33 | 1.66 | 0.05 | 0.05 | 0.03 | 0.01 | |
| Data amount (weekly) | [TB/week] | 3.65 | 0.96 | 0.03 | 0.03 | 0.02 | 0.01 | **4.69** |
| Data amount (year) | [TB/year] | 190.45 | 50.02 | 1.48 | 1.48 | 1.05 | 0.30 | **244.77** |

- Instantaneous load appears to be less of an issue
- Total bandwidth: < 10 MB/s (N.B. DVD: 1-2 MB/s, Blu-ray: 5-6 MB/s)
- Required buffer: ~ 5 TB/week
- Should aim at a total storage amount of less than 5-10 TB/year

# Some use cases*

* Taken from D. Ondreka

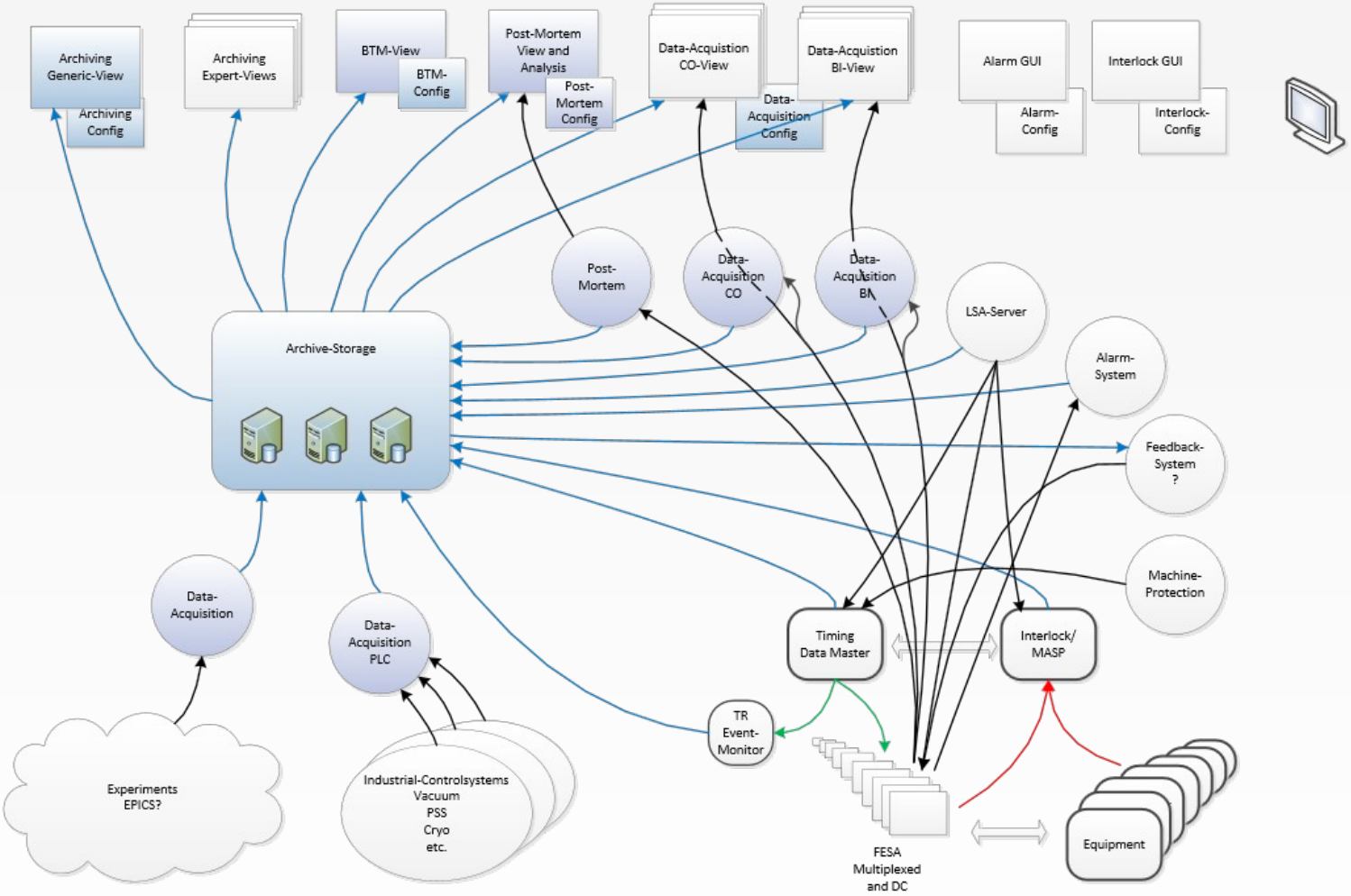
- FAIR accelerators are partly aliens
  - We have strong expectations about their behavior
  - Will (hopefully) largely turn out true
  - Be prepared to reveal the hidden 'features':
    Log data as much as you can
- Examples from GSI operation
  - Mysterious reduction of SIS18 current:
    - No transmission change in UNILAC
    - By accident UNILAC profile grids had been printed
    - Vertical beam position changed
    - Traced to change of beam request timing
  - Sudden pressure rise in SIS18 extraction sector
    - All vacuum valves closed
      (logged, but not order nor source of vac. interlock)
    - FRS suspected guilty, but no hard evidence
  - Unexpected activation of H=2 cavity in SIS18
    - Comparison of beam loss patterns might help chasing down the source, but no data available
  - Dynamic vacuum questions
    - Topic often comes up in analysis of MD studies
    - Of course, nobody thought about recording…

Why we should log data as much as we can:

- We don't know in advance which data might be interesting/useful later
- Performance evaluation
  - Why was beam XY better/worse than before?
  - Search for long-term drifts and their causes
- Collect data routinely
  - No risk of forgetting to record data
  - Accumulation of data not only during MDs
  - Large data sets for all kinds of analysis
  - Somebody's noise is somebody else's signal
- New machines/operation modes
  - Backup data in case of unexpected problems
- MD studies often reveal unexpected effects
  - Relevant data might not have been recorded
  - Repetition of study would waste beam time
- Analysis of rare events
  - Typically not easily reproducible
  - Might have a history of 'near misses'
  - Might not be detectable by post-mortem

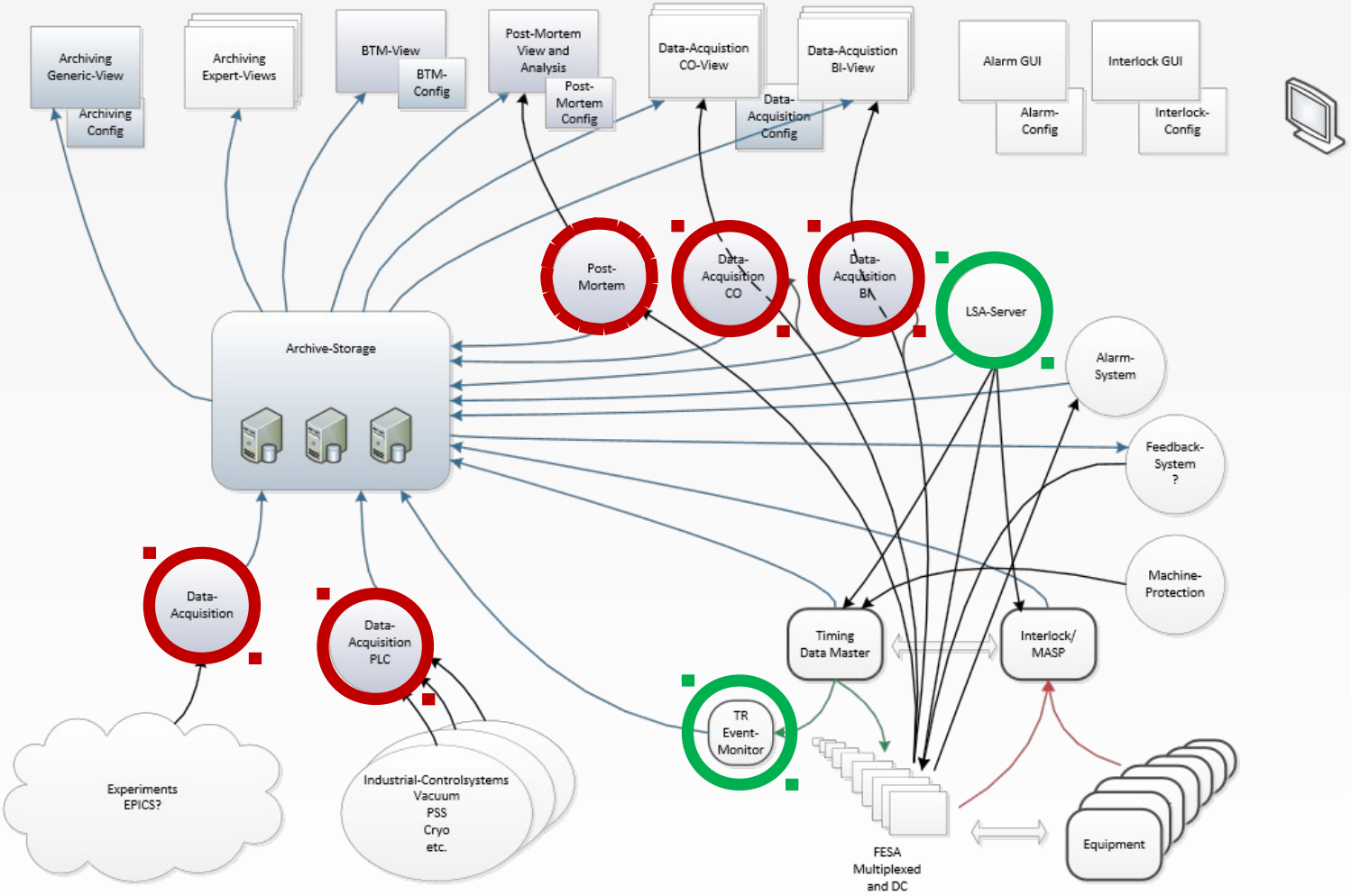
# Technical Overview

# Archiving Environment



# Data Acquisition

# Data Acquisition



# Data Acquisition

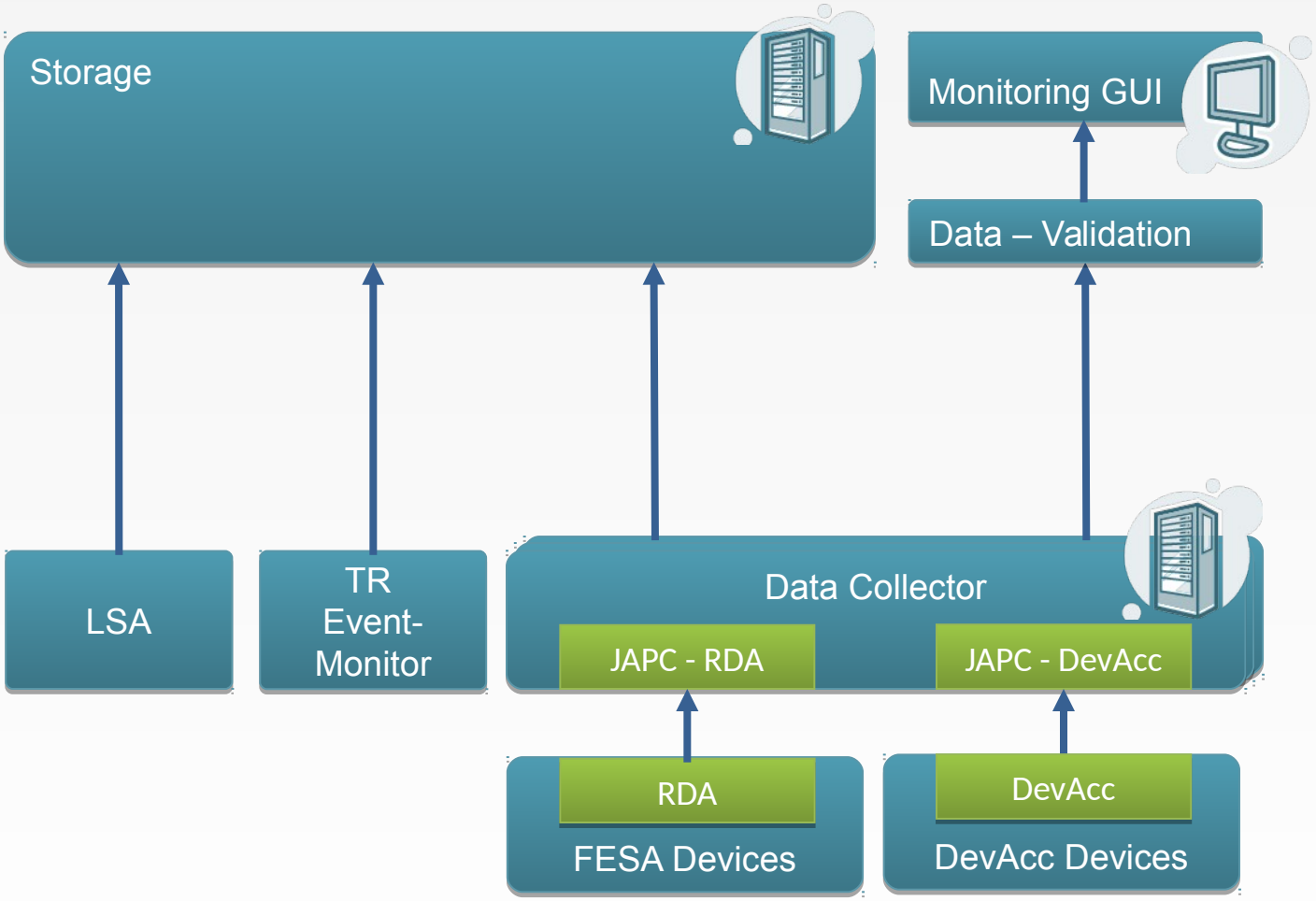
- Main data sources:
  - FESA Devices
  - UNICOS
- Additional required data:
  - Timing information (Events with timestamps)
  - LSA information (Meta-data about particular beam chains)
- Post Mortem System should be able to write data to the AS
  - Use of generic API
- Validation of data is required
  - May be sync- and asynchronous
  - Synchronous validation can be costly in terms of performance

# Data Acquisition – Data Collector

- A component called Data Collector is responsible for acquisition of data from FESA devices

- Needs to be highly scalable and performant
  - Multiple instances and shared load
  - Load distribution: monitor the amount of incoming load and distribute it at runtime

# Data Acquisition (overview)



# FESA Devices

- JAPC Subscriptions are to be use to acquire the data from particular devices
  - Received data contains a timestamp and selector if the device in connected to the timing network, otherwise only a NTP timestamp is available
  - JAPC Subscriptions can be used to collect data from DeviceAccess Devices

- One particular archiving value may be a single field in an acquisition parameter
  - It is common at GSI to use only one “Acquisition” parameter containing all get-fields

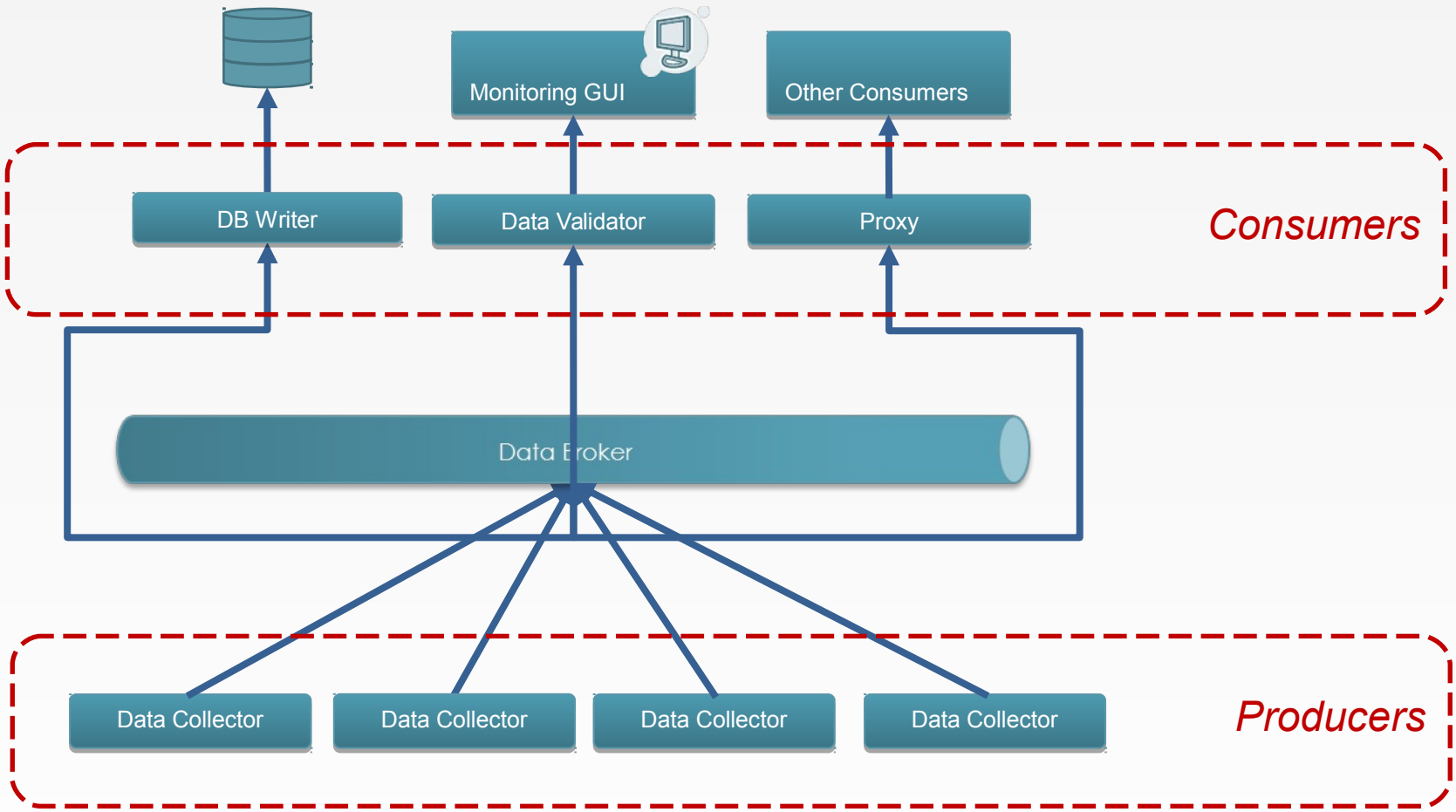
# UNICOS Devices

- UNICOS is using an own database (Oracle or RAIMA?) to store data coming from devices
- This data should be transferred via Database to Database transfer

# Data delivery and propagation

- A typical messaging broker pattern
- Usage of some high performance messaging broker to concentrate the data coming from Data Collectors and then distribute it to different consumers
- Possible broker consumers
  - Data Validator
  - Actual Data Writer
  - Some Proxy-like components
- Possible broker technologies
  - Apache Kafka
  - Chronicle queue
  - Real-Logic Aeron

# Messaging


Monitoring GUI
Other Consumers
DB Writer
Data Validator
Proxy
Consumers
Data Broker
Data Collector
Data Collector
Data Collector
Data Collector
Producers

# Having a Broker

- Advantages :
  - Easy message distribution
  - Extendibility : easy to add new consumers
  - Additional level of fault tolerance if the broker caches messages

- Disadvantages
  - Additional component to maintain and support
  - Additional bottleneck

# Choosing a Broker Technology

- Apache Kafka (current favorite)
  - Should deliver enough performance for all FAIR devices even in 1-node configuration
  - Good spread in the industry (Cisco, Netflix, Ueber etc.)
  - Good documentation cover
  - Diverse tools and plugins


- Chronicle Queue
  - Outperform Kafka in tests (http://bit.ly/1TiTKyo)
  - Commercial Product - not all features available in open source version
- Aeron
  - New product (late 2014) – not widely spread
  - Fast, but lacking of some features (queue persistence)

# Kafka vs Kafka

# Data Storage

# Which Database technology to choose?

- Fundamental question

- Oracle DB would be a native choice at GSI but:
  - Does it cover the model well enough?
  - Does it preforms good enough for future needs ?
  - Does it scale well for an affordable price?
  - Does it provides flexibility for changes in the future?

- Need to consider alternatives as well:
  - Apache Cassandra
  - Apache Hadoop
  - Elasticsearch
  - MondoDB

# Storage - Configurable parameters

- Technical configuration (available for tech. users)
  - Data Reduction:
    - Reduction strategy and rate (if possible)
  - DB Writer:
    - Data writing parameters (e.g. amount of data inserts per write commit)
  - TR Event Monitor
    - Which Events should be written and logged

# Database

- Current Favorite: Apache Cassandra
  - Suits most for our needs with focus on fast writes of small data
  - Good spread in the industry
  - Commercial support available (Datastax)
  - Good documentation, large community
  - Scalable, Reliable etc...

# Data Storage

- Two storage types
  - Short-Term storage (STS)
    - Full range of information, available for a short period of time (couple of weeks)
    - Main focus: fast data writing
  - Long-Term storage (LTS)
    - Reduced information amount (factor 10 if possible), available for longer time period (many years)
    - Main focus: fast search and read of data

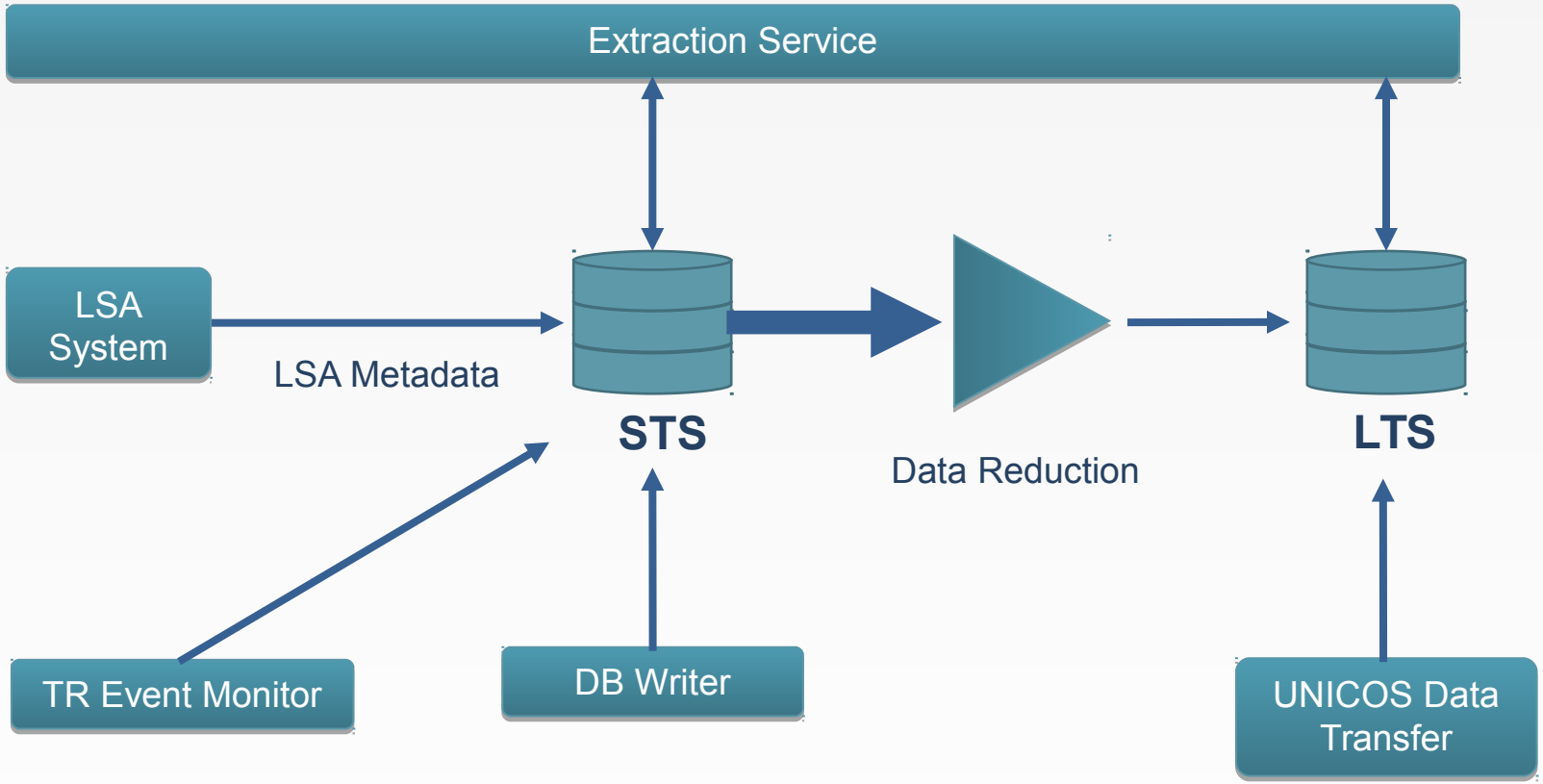
# Data Reduction

- Possibilities of data Reduction:
  - Reduction of sampling frequency
    - Can be adoptable depending on beam operation mode
  - Write down only values changes bigger then certain threshold
    - No lose of information if threshold = 0
  - Cumulate and compress data for a particular signal of multiple cycles for given periods
    - Store <min/mean/stdev/median/max> values
    - Store transients - values outside of n*<stdev> band
    - See Dr. R.Steinhagen slides (20150603_FCWG_Archiving_and_PM) for more details
    - Same principle is used in UNICOS

# Data Storage

- Additional information to store
  - Timing
    - Events including timestamps in order to map the data from devices to particular beam production chains (if no selector was provided)
  - LSA
    - Meta information about particular beam production chains in order to find data corresponding to chains with selected properties

# Data Storage (Overview)



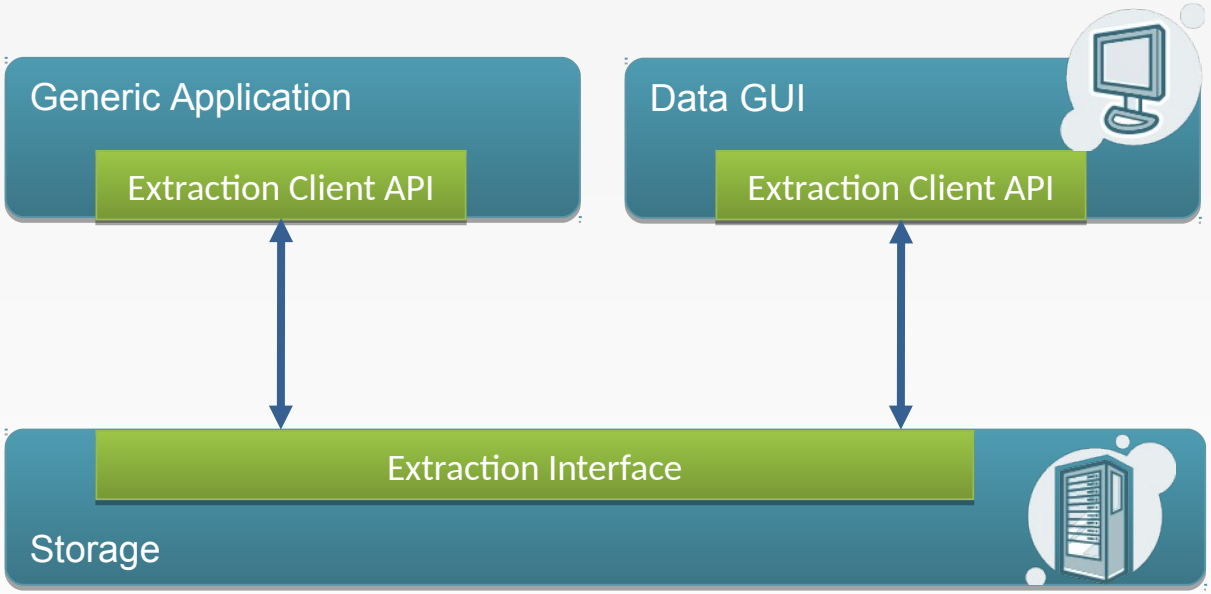
# Data Extraction

# Data Extraction

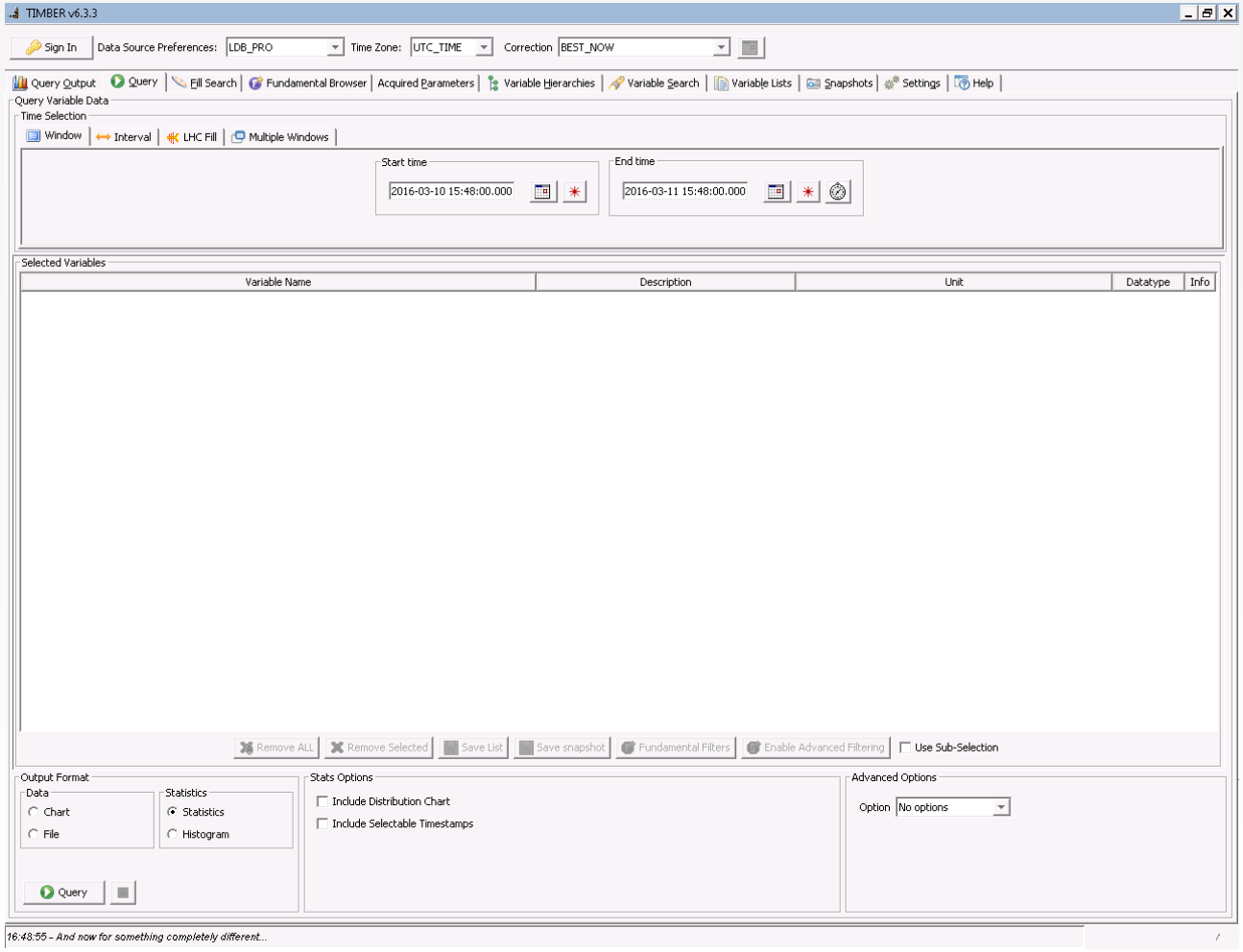
- A generic API to extract data from the Storage
  - Main focus here is Java, but a Phyton and C++ interfaces are also desirable (e.g. for fast data analytics)

- The AS should also provide a GUI to query and display the data
  - CERN Timber can be used as a typical example of such GUI:
    - http://lhc-logging.web.cern.ch/lhc-logging/timber/ (accessible only from CERN)

# Data Extraction (overview)



# Archiving GUI (CERN Timber example)

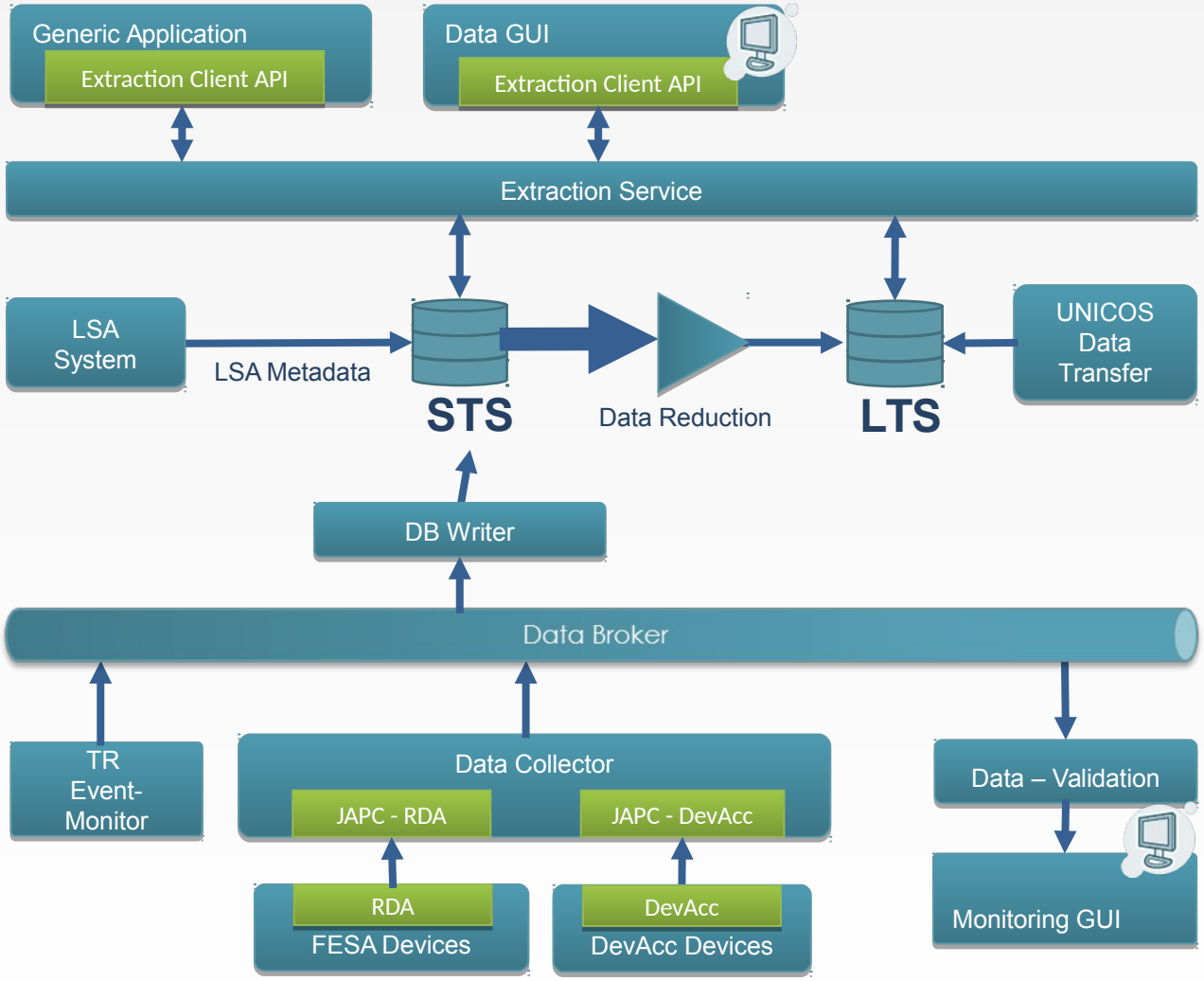


# Some Example Use-Cases

- Select a and view a particular variable for a given period of time (as graph)
- Compare the value of one parameter with another for the same time period in the same graph
- Data selection criteria:
  - Given context (e.g. BPC)
  - Only beam processes with beam
  - Between particular events
  - Context with specific beam type
  - Beam parameter
  - Machine parameter
  - BPC with PM Events
  - Operational ranges

# All together in one picture

# Archiving – Big Picture

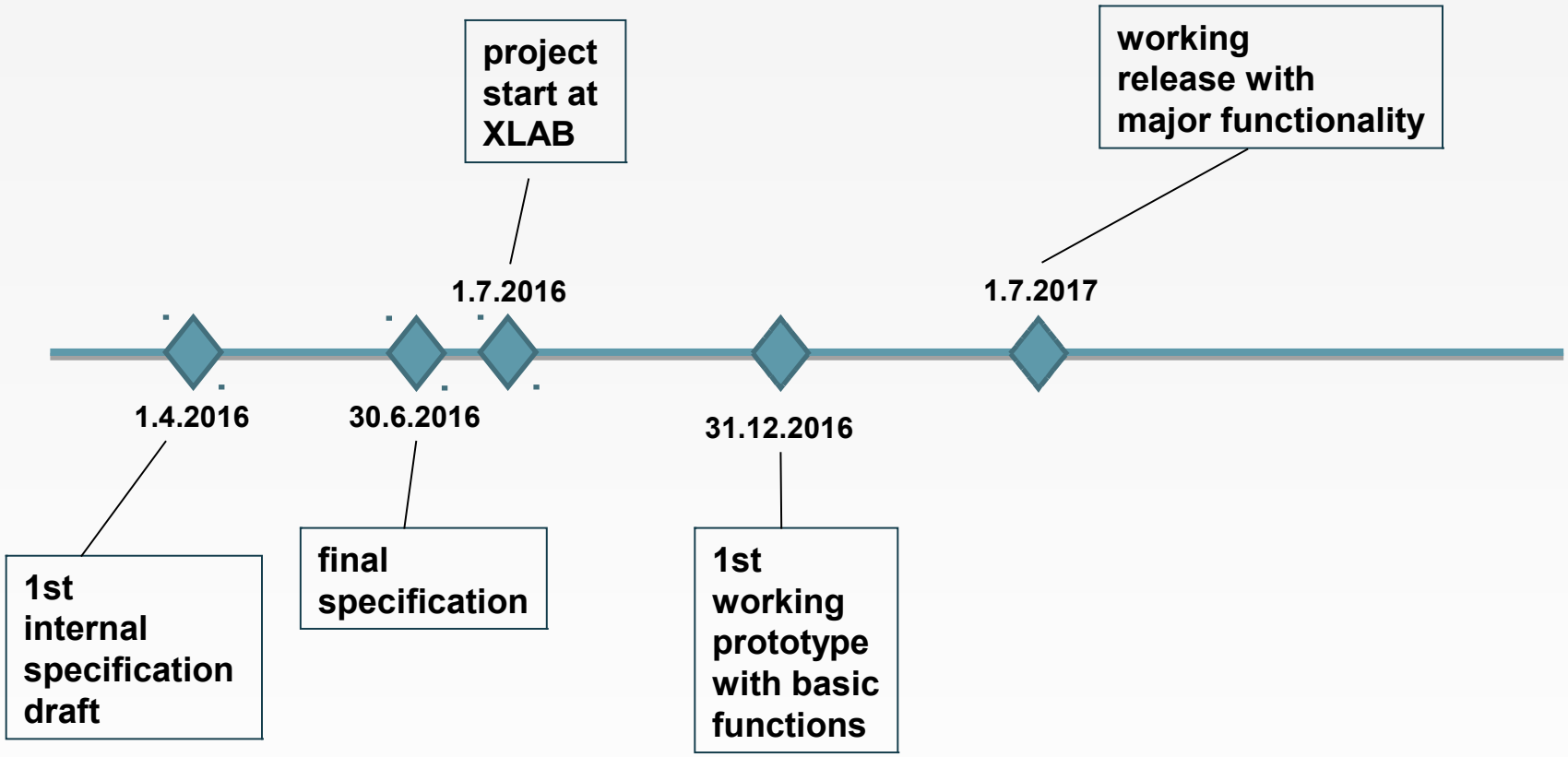


# Roadmap and Next Steps

# Next Steps

- Evaluate the usage of external extraction products e.g.
  - Spark
  - Solr
- Implement a small and simple prototype (PoC) with mentioned products
- Finalize the specification
- Prepare an release plan with XLAB
  - Iterations and their scope

# Roadmap


project start at XLAB
working release with major functionality
1.7.2016
1.7.2017
1.4.2016
30.6.2016
31.12.2016
1st internal specification draft
final specification
1st working prototype with basic functions

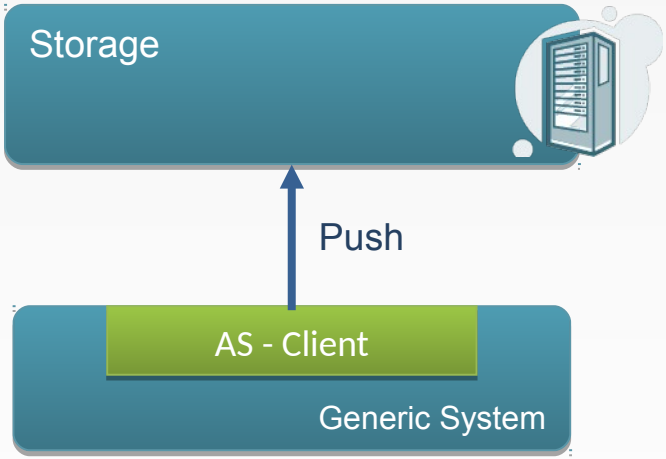
# That's it ??

**Thank you for your attention**

# Outdated or Backup

# Generic API

- A generic API client (Java/C++) should be developed to **push** data to AS



# Generic API (implementation)


| Generic API | |
|---|---|
| **RDA - based** | **Non-RDA based** |
| + Ready to use messaging framework<br><br>- Requires use of device/property model<br>- Client and server will need to be in the same network<br>- Depends on DirectoryServer<br>- No generic protocol -> data delivery only using provided client | + Flexible, since can be designed as we want<br><br>- Development from scratch<br>- High implementation overhead |
| **RDA - SET as push method** | **Request for Subscription** |
| + lightweight<br>+ easy to build<br>+ easy to understand<br><br>- Hard to adapt for device/property model | + can be combined with other subscriptions<br><br>- Client API should act as a RDA Server |

- Some more complex decision looking (incl. prototyping) is required

# Data Acquisition - Configurable parameters

- General configuration
  - Mapping between Device/Parameter/Field value and physical signals if required
  - Monitoring/Subscription parameters (what to collect and archive)
    - Device/Property/Field
    - Selector
    - Filters
  - Validation parameters (if required)

- Technical configuration
  - Amount and distribution of single subscriptions/load per Data Collector instance
  - RDA properties of particular Data Collector instances

# Data Extraction

- Making the Extraction as 2-tier will allow to control and to monitor the accessing to the AS
  - Communication technology needs to be defined

- Technology alternatives:
  - Java Spring RMI:
    - (+) Easy to understand
    - (+) Existing examples at GSI
    - (+) Small implementation overhead
    - (+/-) Can be combined with REST access
    - (-) Requires an App-Server to run
    - (-) Available natively only for Java
  - REST:
    - (+) Generic – client can be written in any language
    - (-) Could be difficult to combine with access right management
  - Also to consider:
    - Apache Thrift