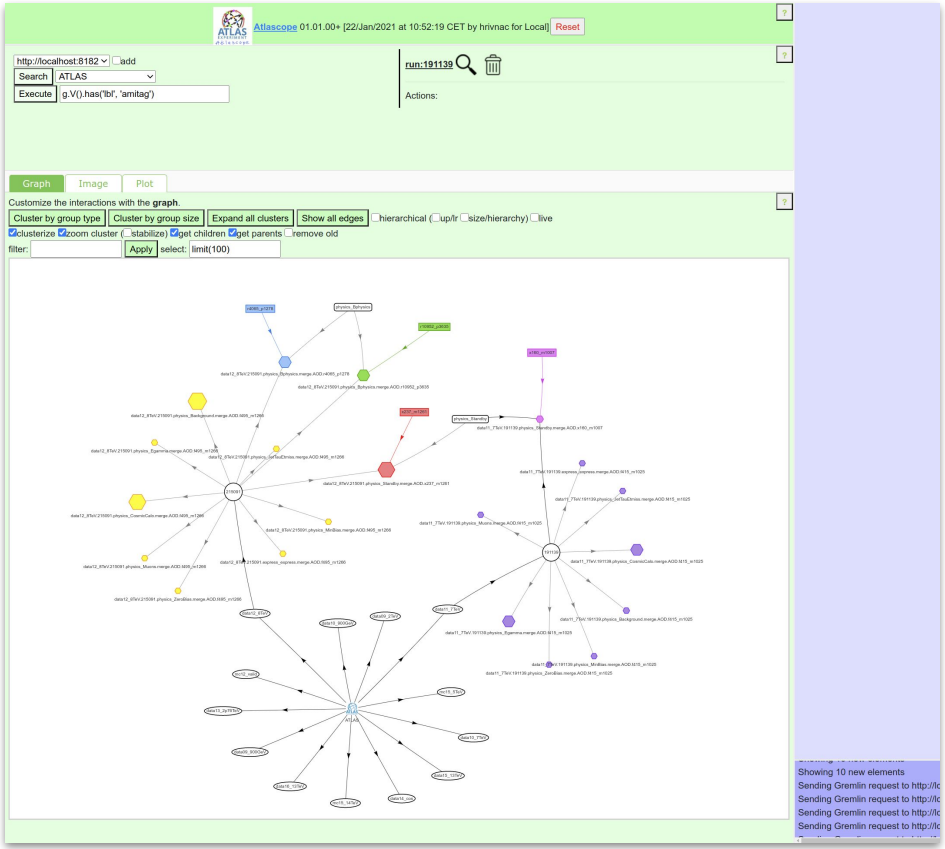
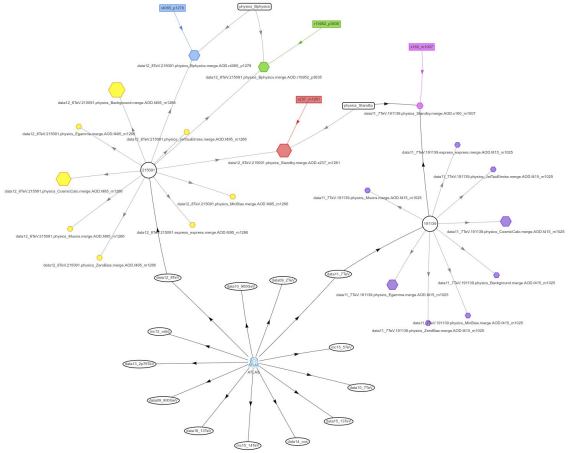
Atlascope



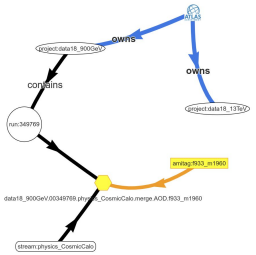


- Architecture
- Views
- Group Population
- Virtual Collections
- Composition

Julius Hrivnac, IJCLab

EI WS, 25/Jan/2021

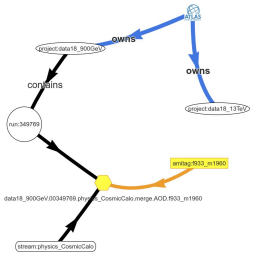
Architecture



Graph Vertices linked with Table rows

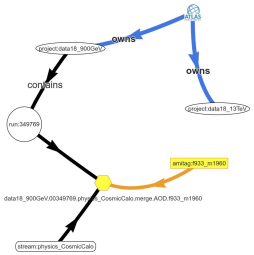
- **Table** has more information, but only contains static information
- **Graph** has less information, but contains structural elements
  - Relations between elements
  - Group of elements
  - Additional elements properties
- Table is read-only, Graph can be extended (by users, groups,...)
- Table rows and Graph Vertices are linked (user can navigate between them)

Architecture



- The architecture is simple:
  - A **Graph layer** on top of an **SQL or NoSQL (HBase) database**
  - A **table** corresponds to a **Vertex type** (label)
  - A **row** corresponds to an individual **Vertex**
  - Graph layer is transparent, Vertexes are created when first requested, then stored in GraphDB (**lazy creation**)
  - SQL table **relations** are automatically represented by graph **Edges**
  - New Vertices and Edges can be freely created in the Graph layer (independent on the SQL/NoSQL storage)
  - **Collections** are represented by Vertices with Edges to included Vertices or by embedded commands to get them from the associated storage
  - All Graph **tools** are then available for access, navigation, analyses and visualisation
- Very little of code
  - Using a lot of (mostly Apache) projects
  - Standard APIs, replaceable components

Architecture



TinkerPop

Gremlin Client

Gremlin
(JSON over REST)

Gremlin
Proxy

TinkerPop

Gremlin Server

Phoenix Connector
UDF

Gremlin DB (Janus Graph)

SQL/JDBC
over Socket

Tomcat

Atlascope WS

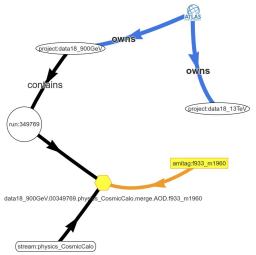
SQL (Phoenix,...)

Happy User

our code (in red)

HBase

Architecture - Evolution



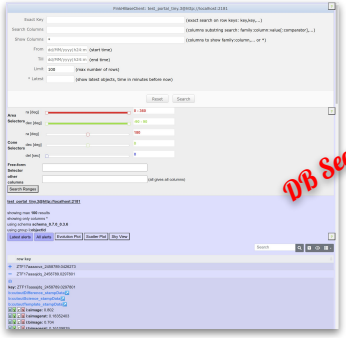
- Original idea: Full JavaScript client accessing both SQL and Graph servers
- Problem: CORS prevents co-loading pages from different servers
- Solution: **Proxy Service (@Tomcat) using the standard Gremlin protocol in-out**

- Original idea: Graph vertices created lazily once requested
- New possibilities:
  - **Bulk loading**
  - **Vertex collections with embedded query**

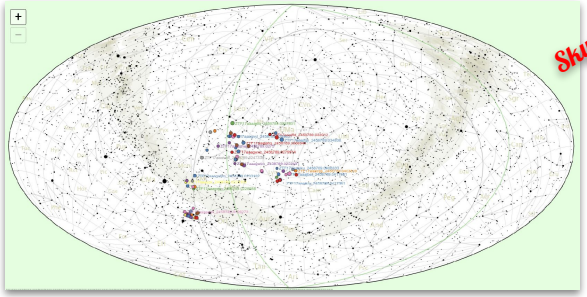
- Original idea: supporting any SQL database
- New possibility: **Supporting directly also NoSQL (HBase) storage**

- Original idea: Support Graph View + embedded external Views
- New possibilities: **Embedded exploration of SQL/NoSQL data via specific Applets (Views)**

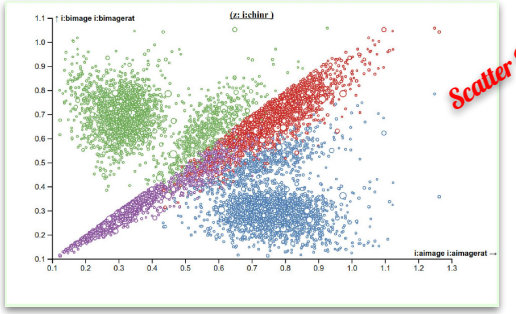
Specific Views



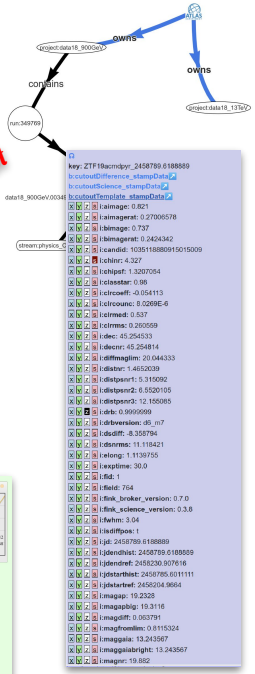
DB Search



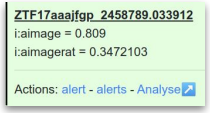
Sky View



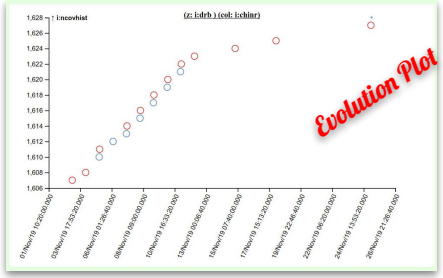
Scatter Plot



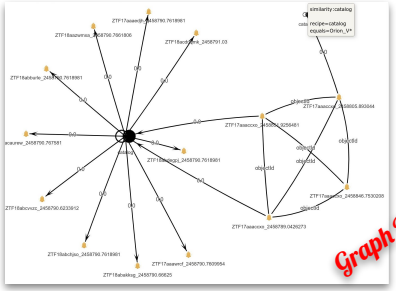
DB View



Data Introspection



Evolution Plot



Graph View

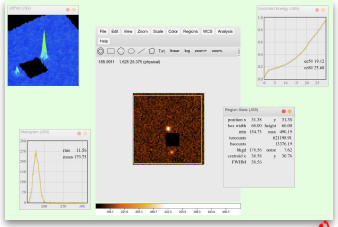


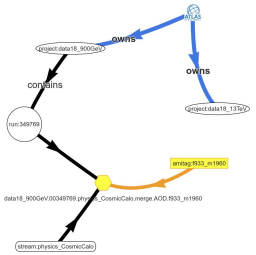
Image View

SQL

HBase

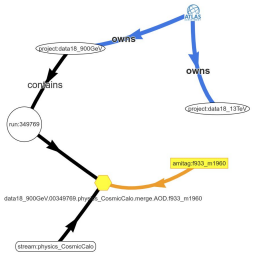
Graph

Views



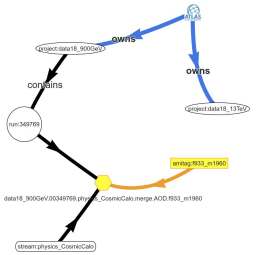
- Views are interconnected:
  - For example
    - When you click on the element (Vertex or Edge) in the Graph View,
    - you will get its properties in a popup
    - and all possible ‘context sensitive action’ available for that element,
    - which allows you to view the element in another view
    - or to use it as a base for the subsequent database search
- There are two ‘hub views’:
  - The **Graph View**, which gives the hierarchical structure of all elements with basic properties and the possibility to show them on other views
  - The **Database View**, which allows searching for elements, their interrogation and use in other views
- Views can be embedded in other web applications

Graph Population



- Initial Graph can be
  - Populated at the same time as the SQL/HBase database
    - small: just row ids + higher-level elements (datasets,...) + relations
  - Populated lazily at need
    - the default behaviour so far
  - Augmented with properties, elements and relations not available from the bare SQL/HBase database

Virtual Collections

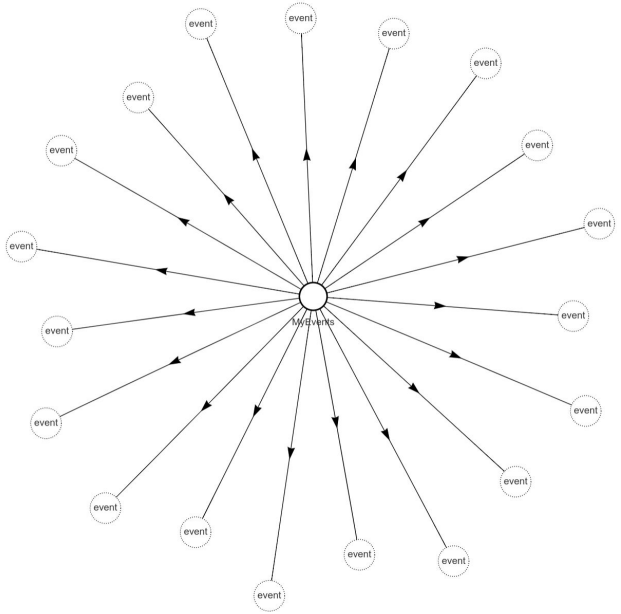


Virtual Collection = Collection Vertex + Edges to contained Elements

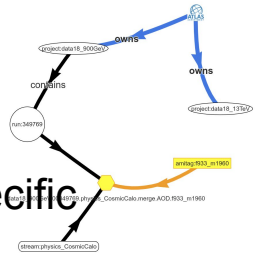
```
// Create new collection of events
eventsCollection = g.addV('ecollection')
                    .property('name','MyEvents');

// Find all events satisfying certain conditions
// and connect them to the event collection
g.V().hasLabel('event')
     .has(...some selection...)
     .collect {
       eventsCollection.addEdge('contains', it)
       };

graph.tx().commit();
```

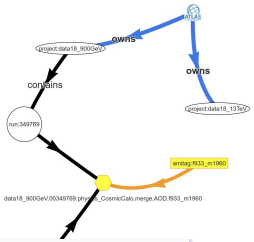


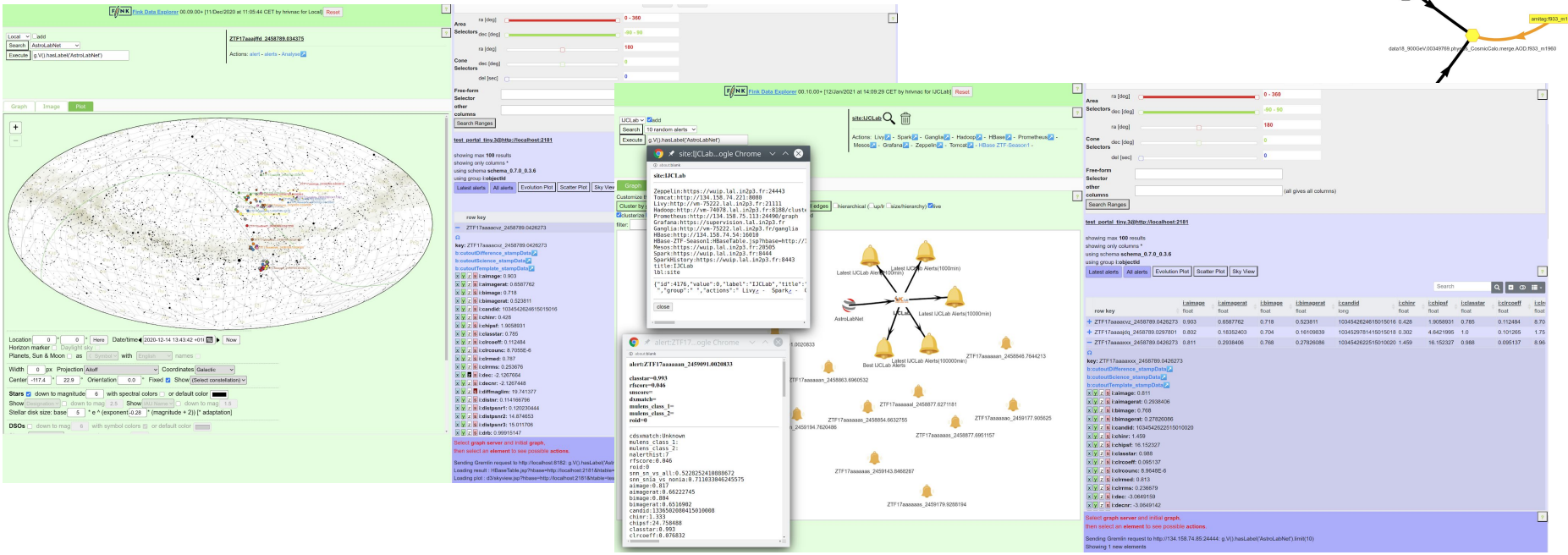
Composition



- ***Lomikel*** contains most of the functionality, which is used from experiment-specific applications vie set of stylesheets and configurations:
  - ***Fink Explorer*** to access Rubin Lab. - LSST alerts managed by the Fink Broker
  - ***Atlascope*** to access ATLAS metadata

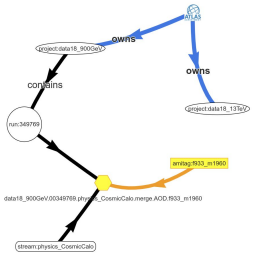
Using in LSST/Rubin Lab (JanusGraph + HBase)





- **LSST Alerts database** contains alerts from the Rubin Lab telescope
- Each alert consists of basic alert properties + references to detailed telescope data
- Alerts are broadcasted world-wide via network of Brokers
- 10 millions alerts = 20 TB of data per night
- **Very similar structure to ATLAS Event Index database**

Links



- **Home page:**
  - https://cern.ch/hrivnac/Activities/Packages/Lomikel
  - https://cern.ch/hrivnac/Activities/Packages/Atlascope
  - https://cern.ch/hrivnac/Activities/Packages/FinkBrowser
- **Code:**
  - https://github.com/hrivnac/Lomikel
  - https://github.com/hrivnac/Atlascope
  - https://github.com/hrivnac/FinkBrowser

Using (copy of) the old Zbyszek setup @CERN

Need SQL schema & JDBC URL to test with new database (GIT?)