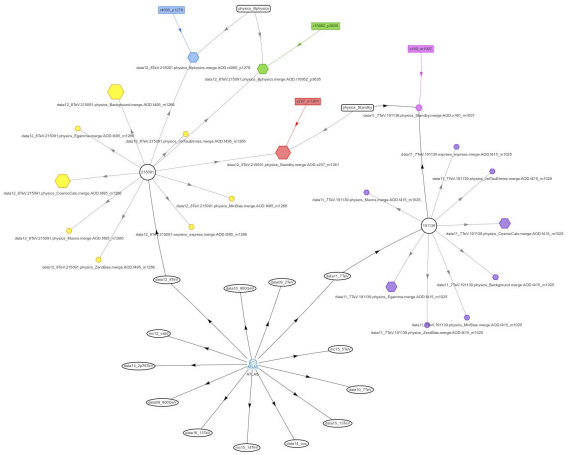
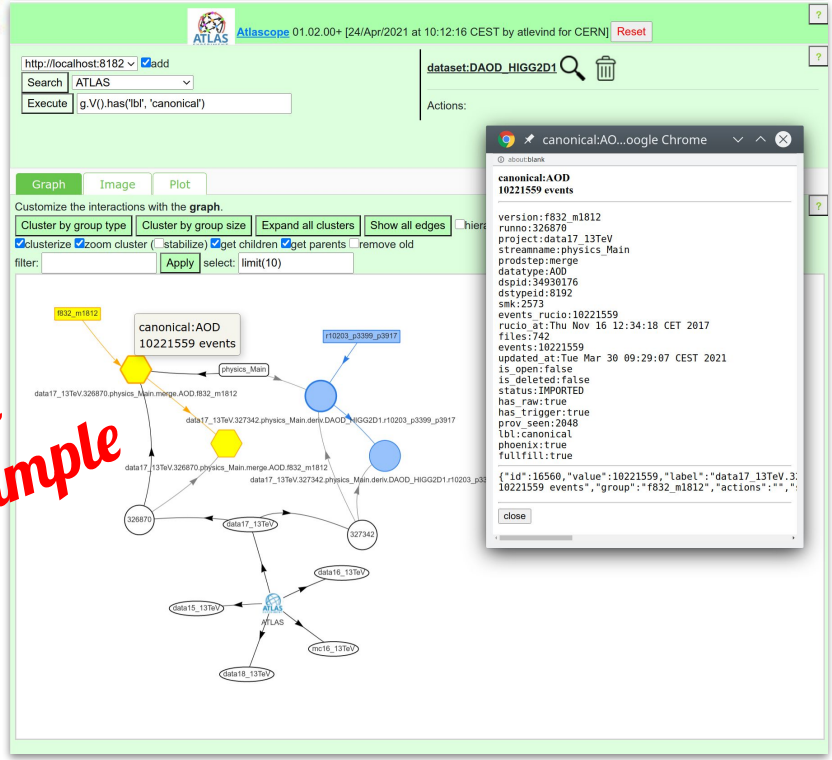
Atlascope





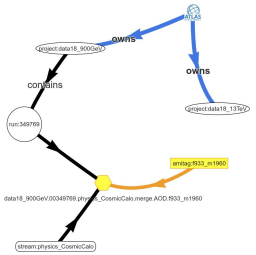
Real data from Small IFIC sample @CERN

- Status
- Vertex Types
- Shadowing
- Strategy Proposal
- Pure Python Client
- Event Lookup with Graphs
- Virtual Collections

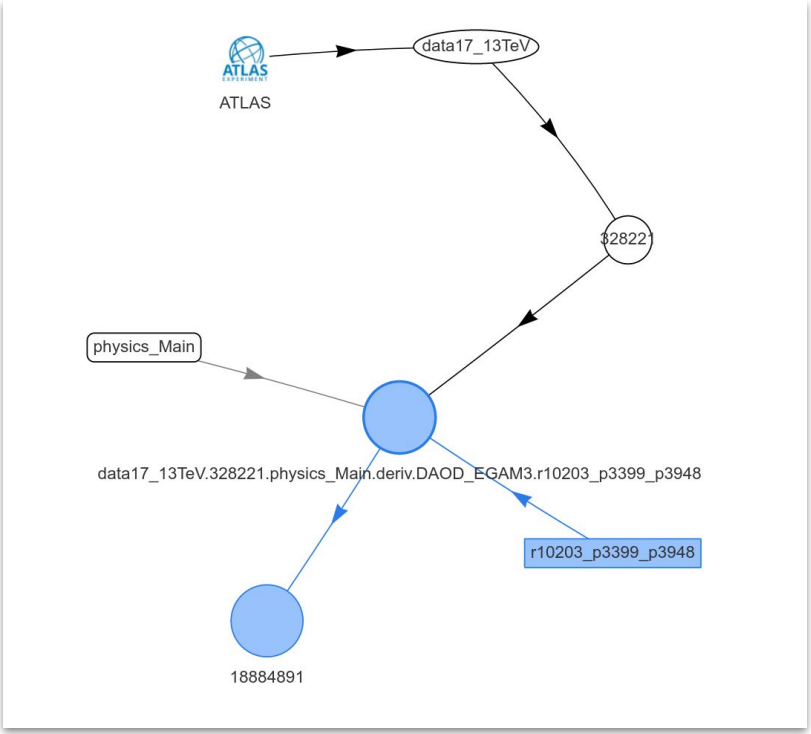
Julius Hrivnac, IJCLab

EI, 3/May/2021

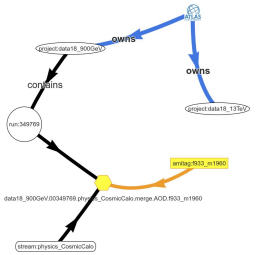
Status



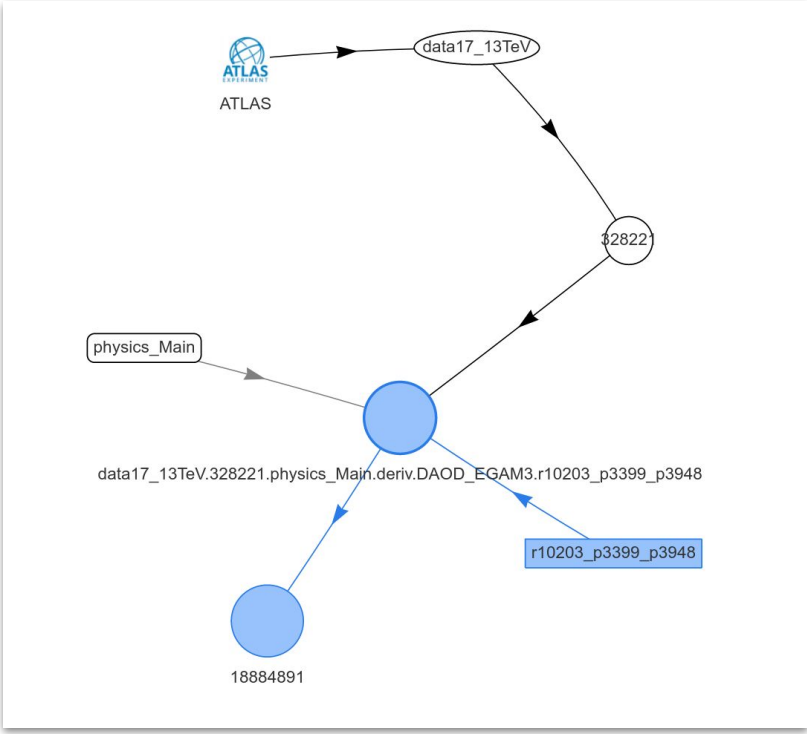
- Importing data from IFIC Phoenix database (@CERN) works fine
  - @ 50Hz
    - Mostly due to Phoenix overhead and connection over Socket
    - Direct importing from files gives about 1kHz
  - Importing **datasets** and **canonical**
  - While importing, creating full graph structure
  - So enabling *searching by navigation*
- So far, filling all attributes
  - Even when redundant
    - no reason to replicate everything in **dataset** which has a relation/Edge to **canonical**
    - dspid, dstypeid,... are only needed to connect to Phoenix, Graph uses relations/Edges
    - project, streamname, version are replaced with relations/Edges to their Vertices
      - for example version property is replaced with an Edge to Amitag Vertex
  - Production Graph will be much smaller
    - So import will be much faster



Vertex Types (labels)

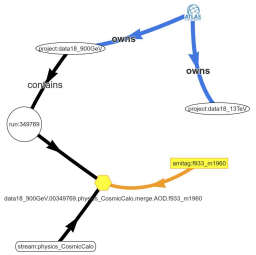


- **Canonical**: bulk imported from Phoenix
- **Dataset**: bulk imported from Phoenix, childrens of Canonical
- **Stream**: created
- **Amitag**: created
- **Run**: created
- **Project**: created
- **ATLAS**: the top level Vertex
- **Ecollection**: Event Collection, to be created by users
- **Event**: To be filled lazily



Shadowing

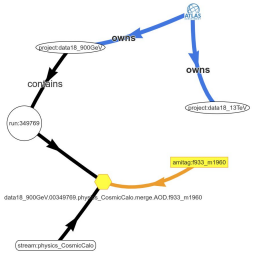
for Events



- Besides complete bulk import, elements can be imported *lazily*, on demand, when requested
  - Keeping link to their Phoenix origin
  - Either filling all attributes or requesting them when used (slower)
- Those elements cannot be searched (till they are completely imported)
- So far, it works via a socket-connected proxy-server isolating Phoenix JDBC driver
  - Due to incompatibilities between HBase of Phoenix and HBase of JanusGraph
  - Phoenix JDBC driver (a client !) includes the whole universe inside (almost 40000 classes !)

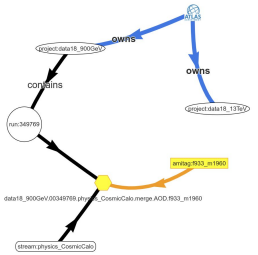
```
// Get or create a Event (which is_a Vertex) and connect to its Phoenix source
d = Event.getOrCreate(...rowkey..., g, false);
// If committed, it will be stored in Graph database


// Get a Event vertex from Graph (or create it in a similar way)
V = g.V().has('lbl', ‘event’).has(...rowkey...).next();
// Create a connection to the Phoenix database (if possible)
//   Wertex is_a Vertex
h = Wertex.enhance(v);
```

Strategy Proposal



- Fully import all **dataset** and higher elements as they are imported into Phoenix
  - And create the graph
  - So that they <u>can be searched</u>
  - How to assure consistency ?
- Import events when needed
  - For example when creating *Virtual Datasets*

Pure Python Client

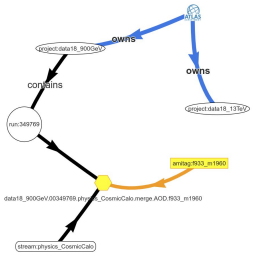



```
#pip install gremlinpython

from gremlin_python import statics
from gremlin_python.process.anonymous_traversal import traversal
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
from gremlin_python.process.traversal import T
from gremlin_python.process.traversal import Order
from gremlin_python.process.traversal import Cardinality
from gremlin_python.process.traversal import Column
from gremlin_python.process.traversal import Direction
from gremlin_python.process.traversal import Operator
from gremlin_python.process.traversal import P
from gremlin_python.process.traversal import Pop
from gremlin_python.process.traversal import Scope
from gremlin_python.process.traversal import Barrier
from gremlin_python.process.traversal import Bindings
from gremlin_python.process.traversal import WithOptions

statics.load_statics(globals())

g = traversal().withRemote(DriverRemoteConnection('ws://aiatlas073.cern.ch:8182/gremlin','g'))

x = g.V().has('lbl', 'dataset').has(...).valueMap().next()
```

Easy integration in Atlas Framework

Event Lookup with Graphs

Naive Way

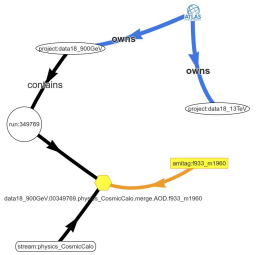


- Suppose Dataset Vertex is just a mirror of its Phoenix extry, with all properties
- Proceeding by ***search*** (may be helped by indexes & ElasticSearch)
- Can be loaded in the Server as a UDF

```
dataset = g.V().has('lbl',      'dataset')          // all Datasets
               .has('project',  'data17_13TeV')     // of a project name
               .has('runno',     328374)            // of a run number
               .has('prodstep', 'merge')            // of a prodstep
               .has('datatype', 'AOD')              // of a datatype
               .next();                             // get the first one (otherwise, get their stream)
events = Event.getOrCreate(dataset, g, 22222, true) // get all Events of that dataset with eventno==22222
                                                    //   and fill their properties from Phoenix (true)
```

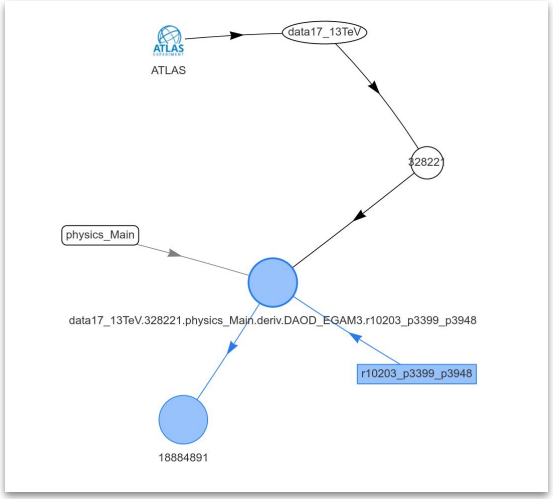
Event Lookup with Graphs

Graph Way



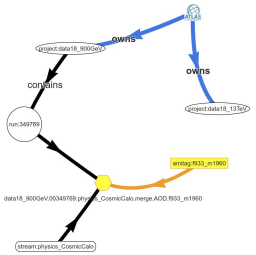
- All Vertices contain only their proper properties
  - Other properties are available as relations/Edges and related Vertices
- Proceeding by ***navigation***
- Much faster
- Need much less disk space

```
dataset = g.V().has('lbl', 'ATLAS')          // top level Vertex
               .out('owns')                  // it owns projects
               .has('name', 'data17_13TeV') // select project name
               .out('gets')                  // it gets runs
               .has('runno', 328374)         // select run number
               .out('fills')                 // it fills canonical
               .has('prodstep', 'merge')     // select prodstep
               .has('datatype', 'AOD')       // select datatype
               .out('contains')              // it contains datasets
               .next();                      // take the first dataset
  events = Event.getOrCreate(dataset, g, 22222, true)
```



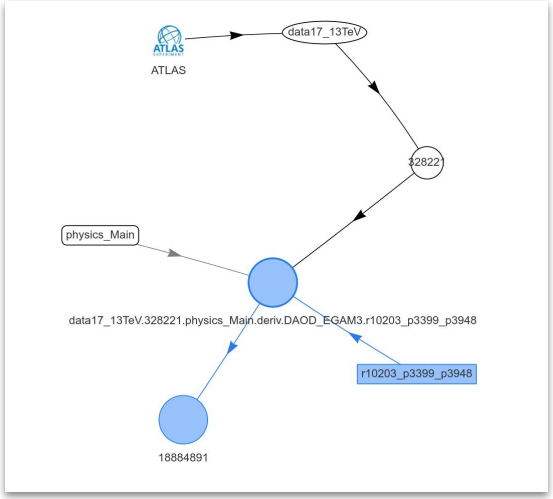
Event Lookup with Graphs

Simple Graph Way

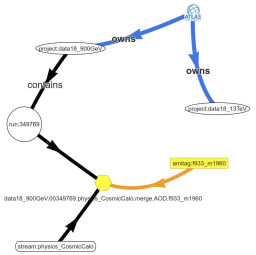


- Simplified
  - Thanks to simple graph structure

```
dataset = g.V().has('lbl', 'ATLAS')
               .out().has('name',     'data17_13TeV')
               .out().has('runno',    328374)
               .out().has('prodstep', 'merge')
                     .has('datatype', 'AOD')
               .out()
               .next();
events = Event.getOrCreate(dataset, g, 22222, true)
```

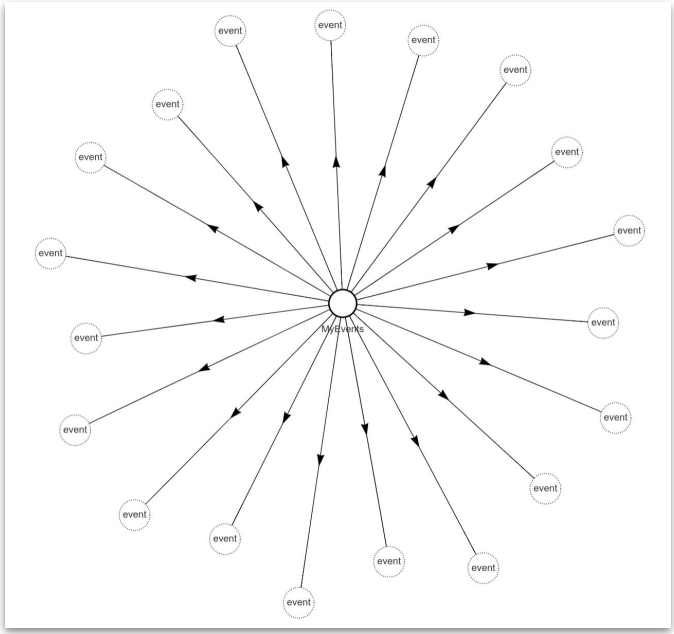


Virtual Collections

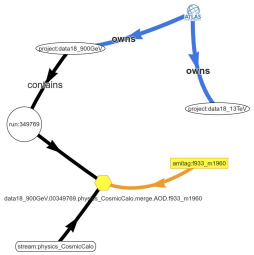


Virtual Collection = Collection Vertex + Edges to contained Elements

```
// 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().has('lbl', 'event')
     .has(...some selection...)
     .collect {
       eventsCollection.addEdge('contains', it)
       };

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



Links



It can work with any SQL database

- **Home page:**
  - https://cern.ch/hrivnac/Activities/Packages/Lomikel
  - https://cern.ch/hrivnac/Activities/Packages/Atlascope
- **Code:**
  - https://github.com/hrivnac/Lomikel
  - https://github.com/hrivnac/Atlascope
- **Gremlin Server:**
  - aiatlas073.cern.ch:8182
- (Prototype) **Web Service:**
  - https://atlas-event-index.cern.ch/Atlascope/?profile=CERN

Thanks to Miltiadis Gialousis for help with connecting JanusGraph to HBase @CERN