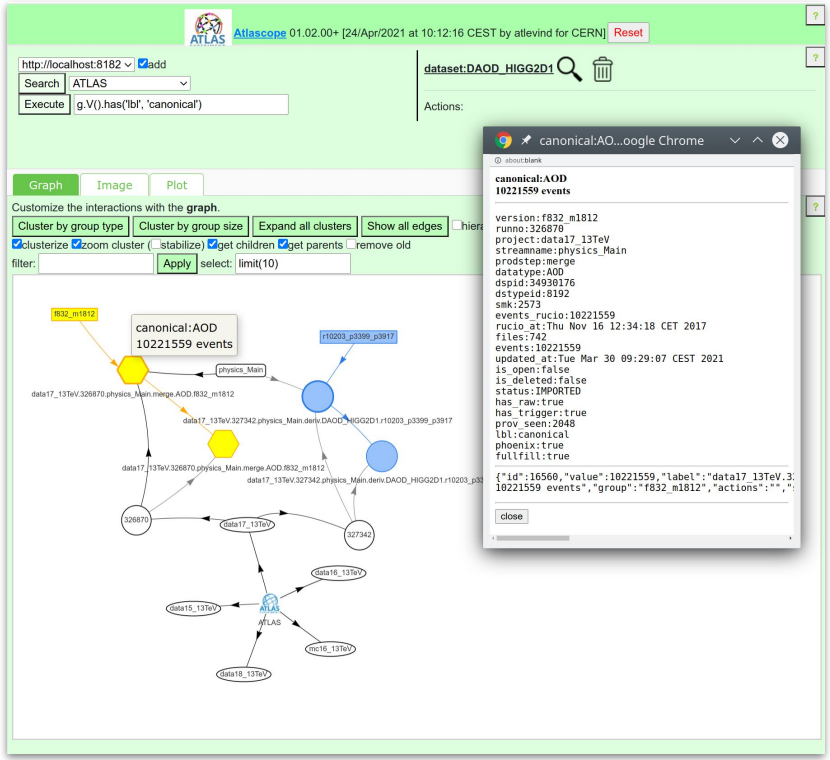
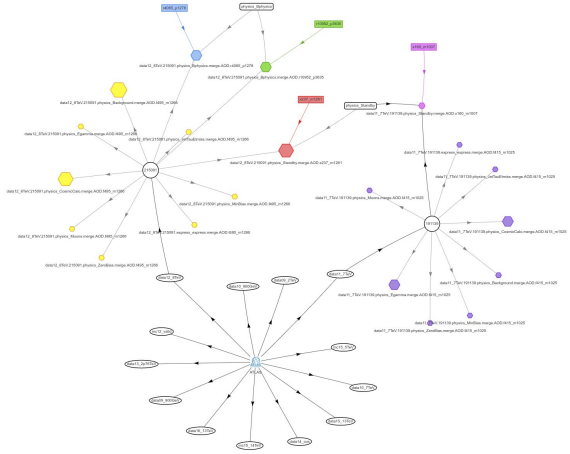
New Event Index





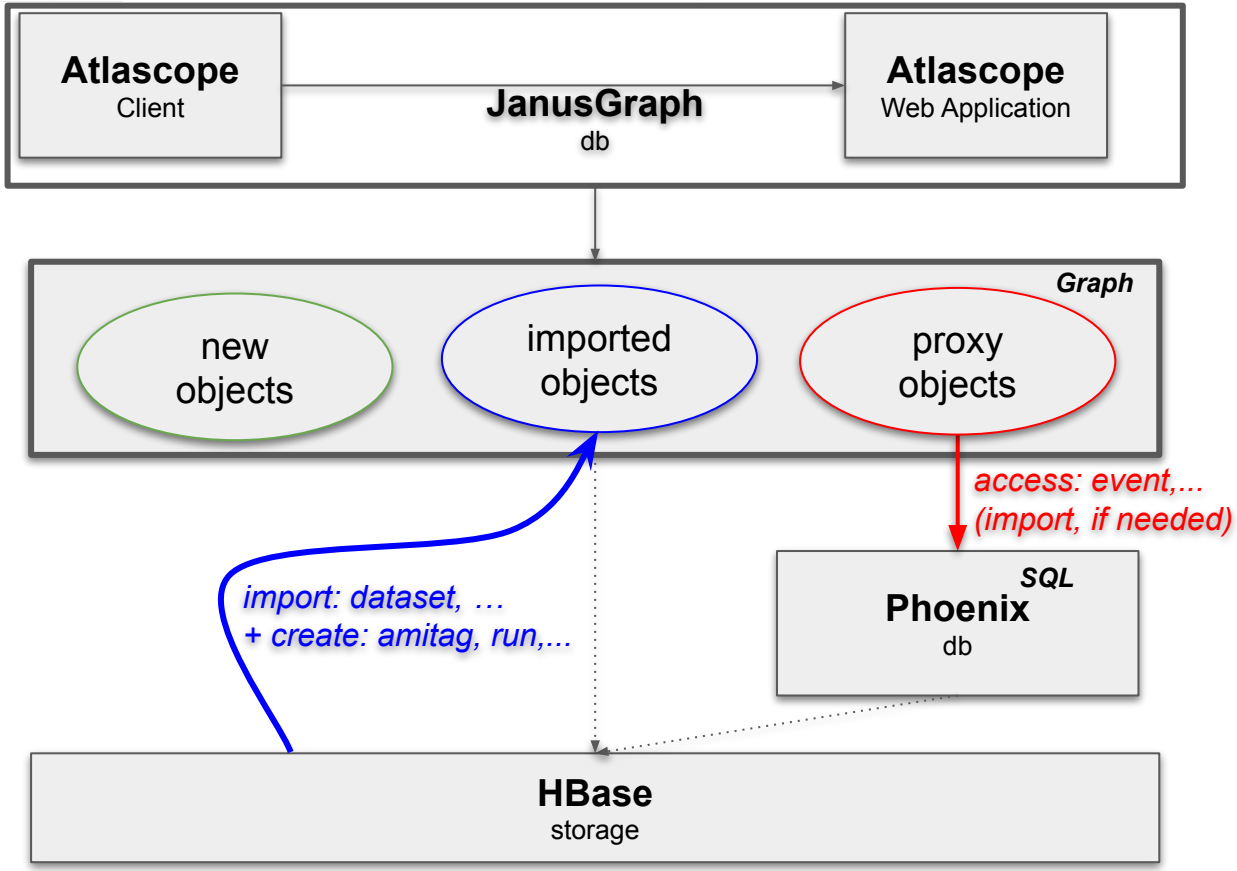
- Event Index
- Graph Databases
- Atlascope

Julius Hrivnac, IJCLab
ATLAS, 6/Dec/2021

Event Index

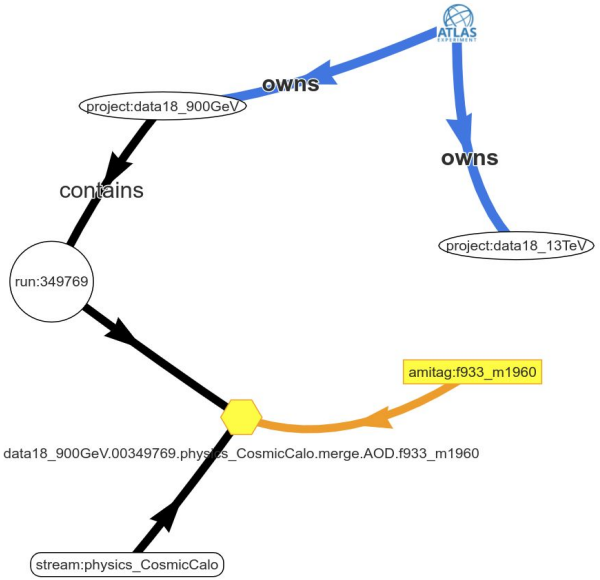
- Catalog of all ATLAS events (data and MC) in all their versions (RAW -> … -> DAOD)
  - With basic characteristics, triggers and references to physical datasets
  - > 360 * 10^9 entries
- Main Use cases:
  - Event Lookup: fast mapping of (set of) eventNumber+runNumber -> physical dataset
    - Grigori in Jan’22
  - Searching for events satisfying certain triggers
  - Overlaps between datasets (AOD, DAODs)
  - Overlaps between triggers within a dataset, triggers statistics
  - Finding errors in data processing or simulation (multiple copies of events,...)
- Incarnations:
  - Tag Database (retired): Included also volatile (physical) information, based on Oracle SQL
  - Current EI (in production): Based on Hadoop HBase (NoSQL database)
  - Run 3 EI (under development): Based on Phoenix SQL database + JanusGraph Graph database over HBase NoSQL database
- Event Index + AMI + Rucio form global metadata service allowing to locate all ATLAS data

Architecture



- Phoenix SQL API provides interface compatible with other ATLAS SQL databases (mostly Oracle)
- Graph API provides flexible and intuitive view of data
- Some Phoenix objects are imported (replicated) into Graph, others are proxied (and can be replicated on request), and others can be created native to Graph

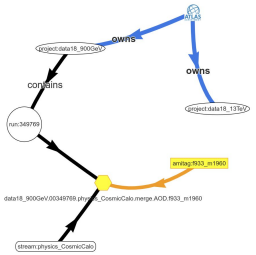
Schema



- In Phoenix, mirrored to **Graph**:
  - Dataset: runno, project, streamname, prodstep, datatype, version, tid, dspid, dstypeid, smk, events_rucio, rucio_at, files, events, events_uniq, events_dup, files_dup, state, updated_at, dups_at, trigger_at, is_open, has_raw, has_trigger, prov_seen, sr_cnt, sr_clid, sr_tech
- In Phoenix, proxied to **Graph**:
  - Event: dspid, dstypeid, eventno, seq, tid, sr, mcc, mcw, pv, lb, bcid, lpsk, etime, phid, tbp, tap, tav, lb1, bcid1, hpsk, lph, ph
- In **Graph**:
  - Project (data18_13TeV,...)
  - Stream (physics_Main,...)
  - Run (348894,...)
  - Amitag (n0002_r13084_p4397,...)

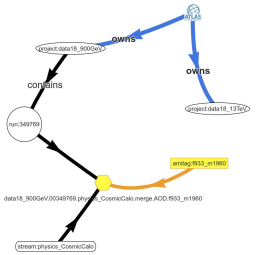
More General Remarks about Storage & Graphs

HEP Storage

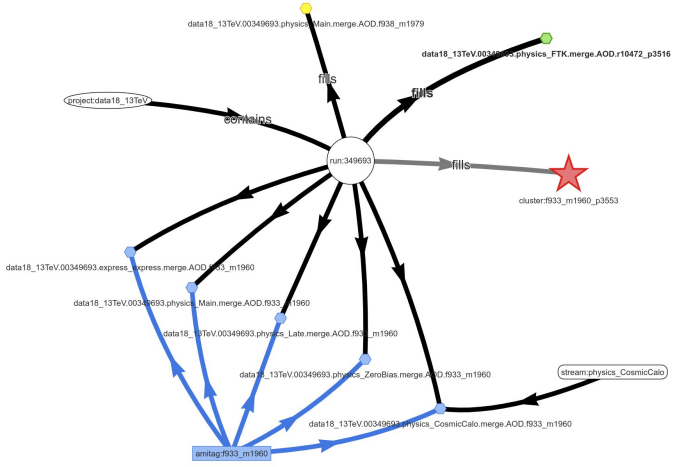


- Traditional data structures in HEP:
  - tuples (tables)
  - trees
  - nested tuples (trees of tuples)
  - relational (SQL-like)
- Schema-based or schema-less
- **But many of HEP data are graph-like** & schema-less
  - **Entities with relations**
- Not handled by standard tree-ntuple storage
  - Relations should be added and interpreted outside storage
- Not well covered by relational (SQL) databases
  - We need to add new relations, not covered by schema
- Difficult to manage by Object Oriented (OO) databases or serialisation
  - Problem to distinguish essential relations from volatile ones

Graph Databases

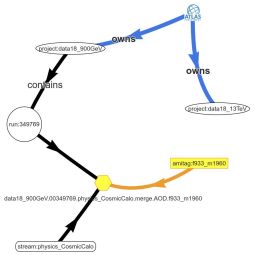


- Storing Graphs in a database
- **Graph = (Vertexes, Edges)**, G = (V, E)
- Vertices and Edges have properties



- Graph databases have existed for a long time
  - Matured only recently thanks to Big Data & AI (Graph NN)
  - Very good implementations & (de-facto) standards available
  - Rapid evolution
- **Moving essential structure from code to data**
  - Together with migration from imperative to declarative semantics
  - Things don't **happen**, but **exist**
  - Structured data with relations facilitates **Declarative Analyses**
- Data elements appear in a **Context**
  - Which simplifies understanding, analyses and processing
- The difference between SQL and Graph database is similar as between Fortran and C++/Java
  - On one side, a rigid system, which can be very optimized
  - On the other side, a flexible dynamical system, which allows expressing of complex structures
- Graph database is a synthesis of OO and SQL databases
  - Expressing web of objects without fragility of OO world
  - Capturing only essential relations, not an object dump

Graph DB: Languages



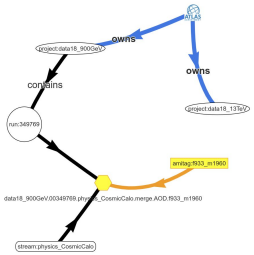
- Direct manipulation of Vertices and Edges
  - Always available from all languages
  - Doesn't use full graph expression power
- *Cypher (and GQL)*
  - Pure declarative
  - Inspired by SQL and OQL
    - But applied to schema-less database
  - Available to all languages via JDBC-like API
    - Semantic mismatch, passed as String
    - There is a wall between coder and database, with a thin tunnel, only Strings can pass
  - Coming from Neo4J
    - Accepted as a standard
    - Neo4J can be also used with Gremlin

```
MATCH (a:run)-[:has]->(b:dataset)
  WHERE a.rnumber = 98765
  RETURN b.name
```

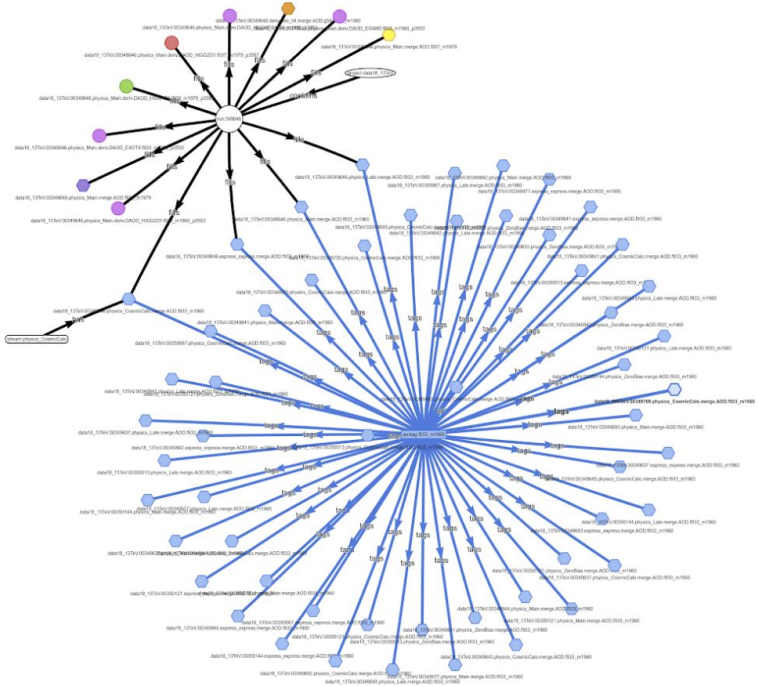
- *Gremlin*
  - Functional syntax
  - Originated from *Groovy*, but available to all languages supporting functional programming
  - Integrated in the language

```
g.V().has('run', 'rnumber', 98765)
     .out('has')
     .values('name')
```

Graph Databases for HEP

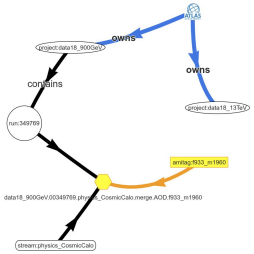


- A lot of ongoing HEP effort to make execution more structured and parallel
  - Parallel programming
  - Functional programming
- Less effort (so far) to structure the data
  - More structured data => simpler and faster access
- <u>Graphical Database advantages</u>
  - More transparent code
    - Stable data structure is handled in the storage layer
  - Suitable for ***Functional Style*** and ***Parallelism***
  - Suitable for ***Deep Learning***
  - Suitable for ***Declarative Analyses***
  - Can help with ***Analysis Preservation***
  - Language & Framework neutral
- How to proceed
  - Store (all) data in a real Graph database
  - Build a Graph layer on top of the existing storage
    - Close to DB layer
    - In the application layer

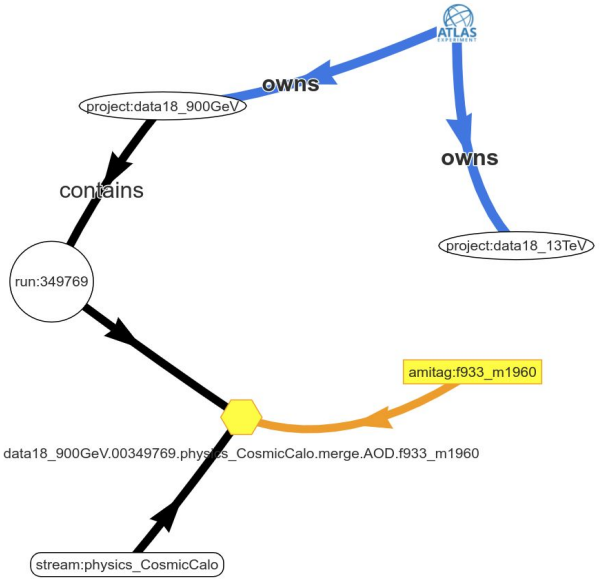


Part of Event Index is stored in a Graph Database

Schema

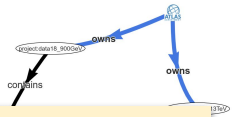


Back to Event Index



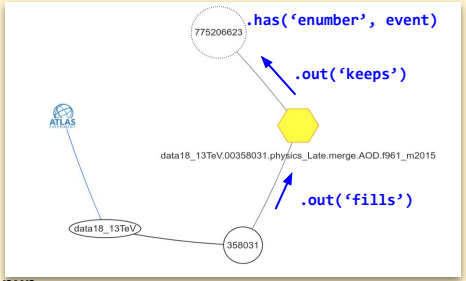
- In **Phoenix**, mirrored to **Graph**:
  - Dataset: runno, project, streamname, prodstep, datatype, version, tid, dspid, dstypeid, smk, events_rucio, rucio_at, files, events, events_uniq, events_dup, files_dup, state, updated_at, dups_at, trigger_at, is_open, has_raw, has_trigger, prov_seen, sr_cnt, sr_clid, sr_tech
- In **Phoenix**, proxied to **Graph**:
  - Event: dspid, dstypeid, eventno, seq, tid, sr, mcc, mcw, pv, lb, bcid, lpsk, etime, phid, tbp, tap, tav, lb1, bcid1, hpsk, lph, ph
- In **Graph**:
  - Project (data18_13TeV,...)
  - Stream (physics_Main,...)
  - Run (348894,...)
  - Amitag (n0002_r13084_p4397,...)

Gremlin Client



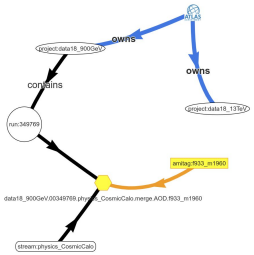
- Functional syntax
  - On top of other languages
- **Functional** & **navigational** semantics
- Very intuitive, **no special syntax needed** (using existing functional syntax), easy integration.
- Database just accessed as objects with structure and relations.
  - Nested collections with links.
- Can use functional API (streams) and Lambda.
- No **semantic mismatch**.
  - Using one language.
- Came from **Groovy**
  - (Almost) identical for other supported languages (Python, Scala, Go,...).
- Both **search** and **traversal** steps.
- Search steps can be boosted by indexes.
- Functions can be loaded on server for faster execution.

```
# add a vertex 'experiment' with the name 'ATLAS'
g.addV('experiment').property('ename', 'ATLAS')
# add edges 'owns' from all vertices 'project' to vertex 'experiment' 'ATLAS'
g.V().hasLabel('project')
     .addE('owns')
     .from(g.V()
     .hasLabel('experiment')
     .has('ename', 'ATLAS'))
# show datasets with more events or number of events in an interval
g.V().has("run", "number", 358031)
     .out()
     .has("nevents", gt(7180136))
     .values("name", "nevents")
g.V().has("run", "number", 358031)
     .out()
     .has("nevents", inside(7180136, 90026772))
     .values("name, "nevents")
```

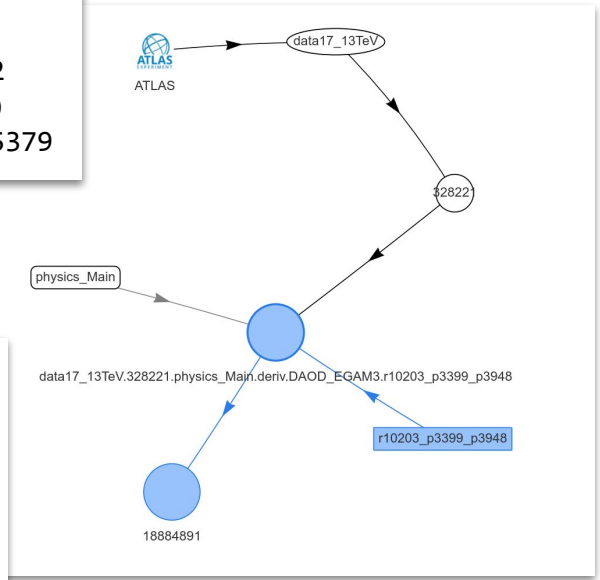



```
# Event-Lookup function (server side UDF)
def el(run, event, g) {
  e = g.V().hasLabel('run')          # all runs
           .has('rnumber', run)      # selected run
           .out('fills')             # all datasets filling that run
           .out('keeps')             # all events kept in that dataset
           .has('enumber', event)    # selected event
           .values('guid')           # its guid
  }

# CLI command
curl -XPOST -d '{"gremlin":"el(run, event)"}' http://ei-gremlin-server.cern.ch:8182
# or using standard gremlin client
gremlin << EOF
:remote connect tinkerpop.server $janusgraph_home/conf/remote.yaml
el(run, event)
EOF
```

Gremlin Client Examples

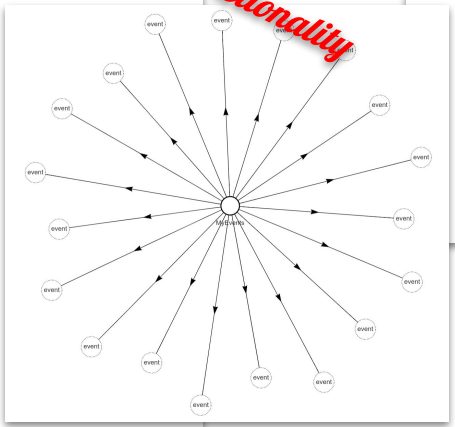



```
// Event Lookup using Graphs
g.V().has('lbl', 'dataset'). // all datasets
      has('runno', 801122).  // datasets with runno == 801122
      out('has').            // all 'has' children (= events)
      has('eventno', 15379)  // all events with eventno == 15379
```

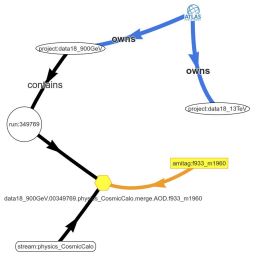


Whiteboard functionality

```
// 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)
       };
```



(C)Python Client

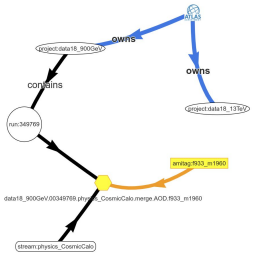


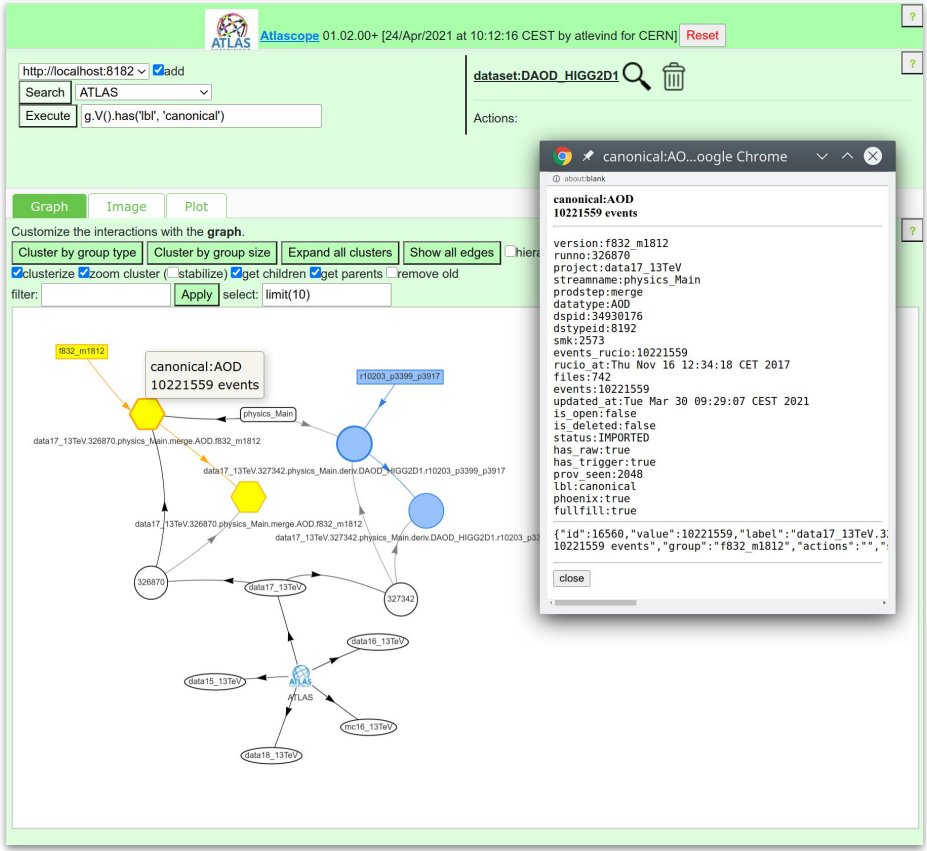

```
#pip install gremlinpython

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

Clients exist in most languages

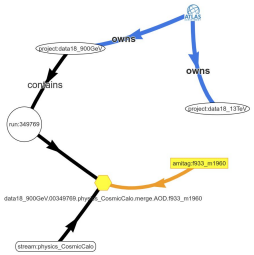
Web Client





- Built on top of Gremlin Client
- Generic Graph Browser
  - Customised for ATLAS Event Index
  - Another customisation exists for LSST

Web Client - Initial Top Panels



Database to use
('Proxy' tunnels requests to the database
through the Web Service to by-pass firewalls)

Initial Gremlin request

Atlascope 02.00.00+ [02/Oct/2021 at 23:13:57 CEST by atlevind for CERN] Reset

CERN-Proxy add

Search ATLAS

Execute g.V().has('lbl', 'dataset')

Connect to the **graph server**
and request the initial **graph**

Customize the interactions with the **graph**.

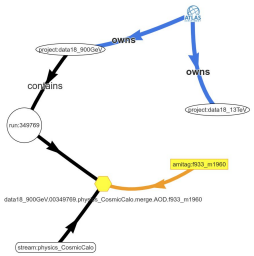
Cluster by group type | Cluster by group size | Expand all clusters | Show all edges | hierarchical (up/lr size/hierarchy) live

clusterize zoom cluster (stabilize) get children get parents remove old

filter: Apply select: limit(4)

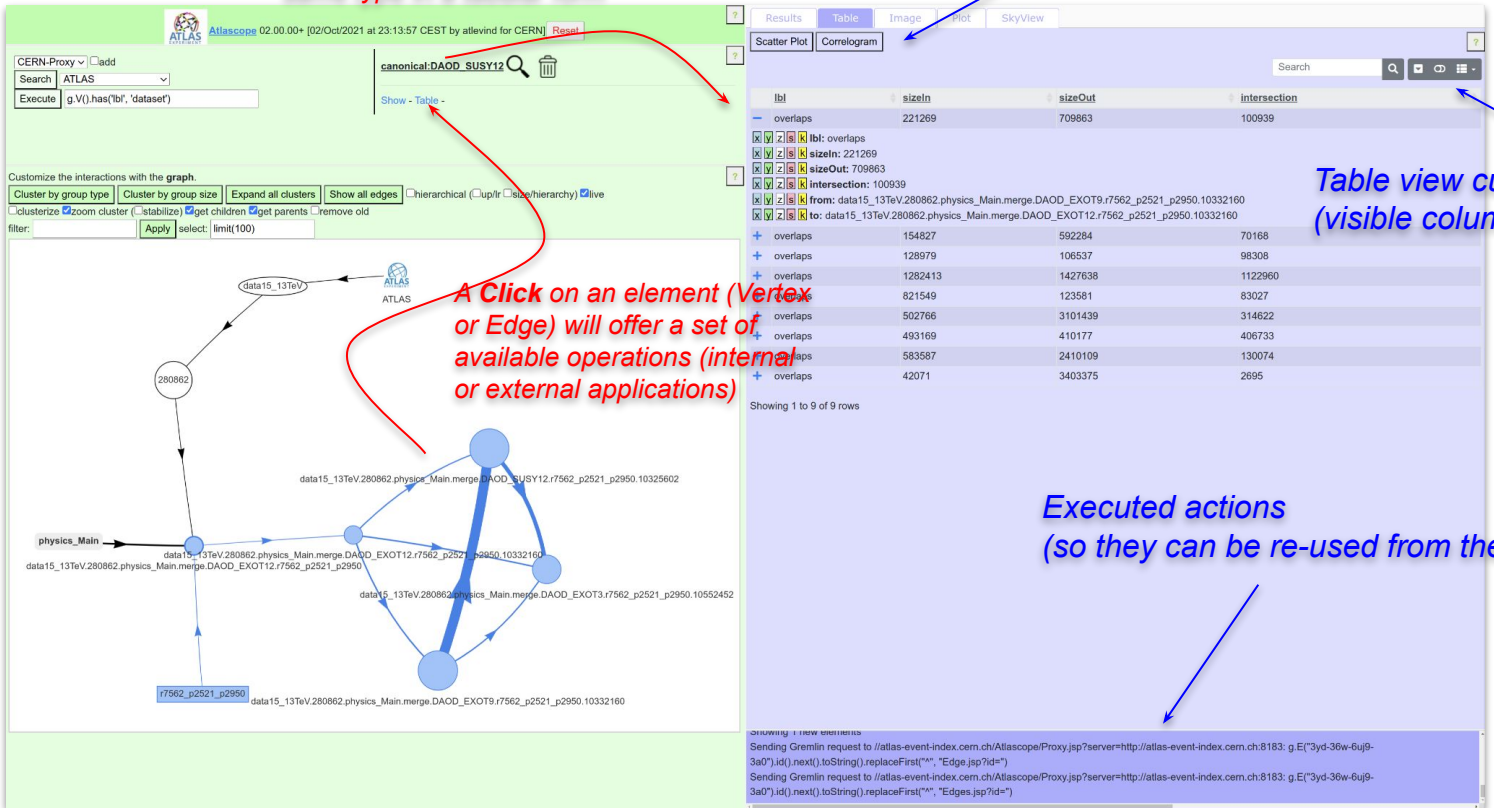
Options for interactive Graph manipulation

Web Client - Graph with Table View



A **Table** operation will show all visible elements of the same type in a tabular form

Table data can be plotted in various ways

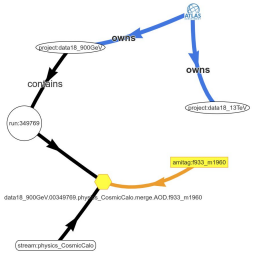


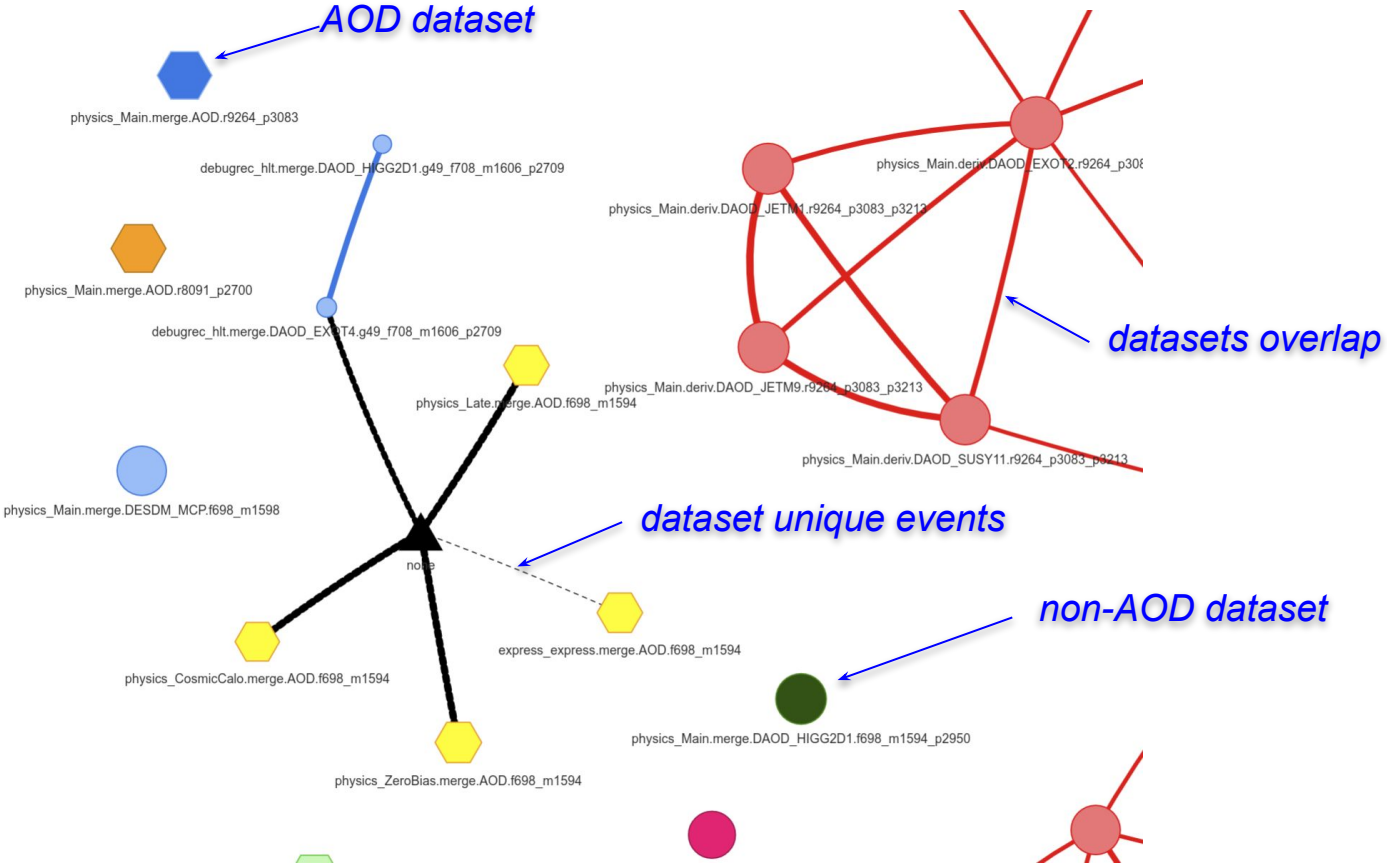
A **Click** on an element (Vertex or Edge) will offer a set of available operations (internal or external applications)

Table view customisation (visible columns, searches,...)

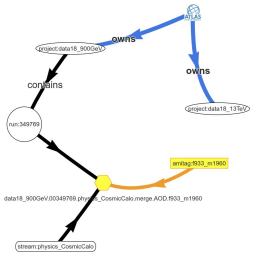
Executed actions (so they can be re-used from the command line)

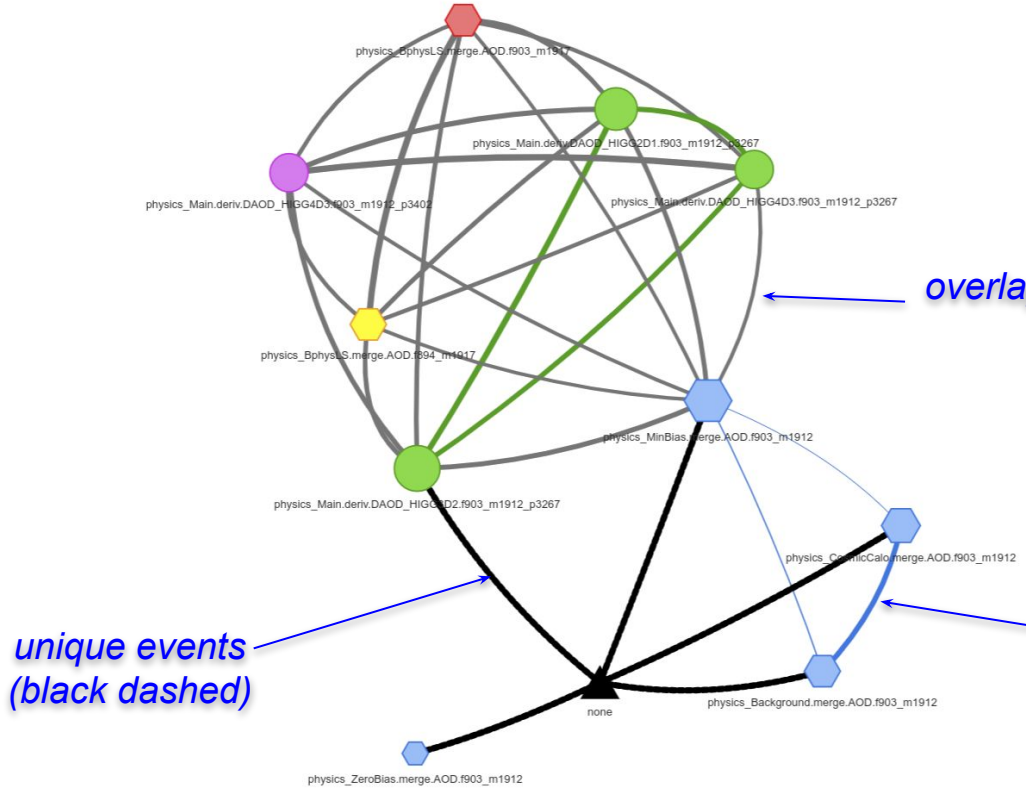
Datasets Graphical View





Overlaps between Tags, Unique Event



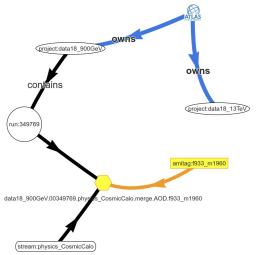


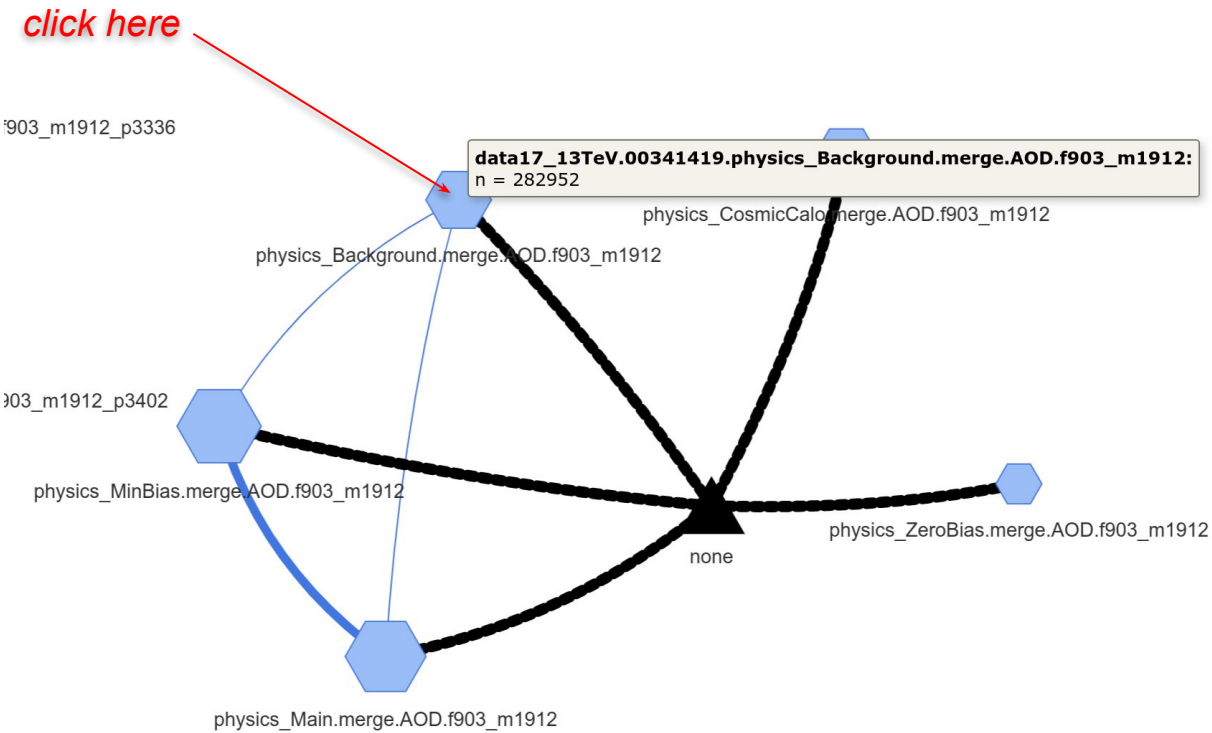
overlap between tags (grey)

unique events (black dashed)

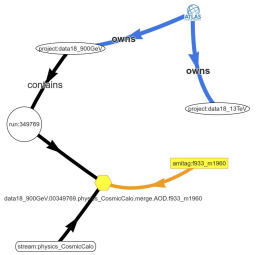
overlap within tag (with color of the tag)

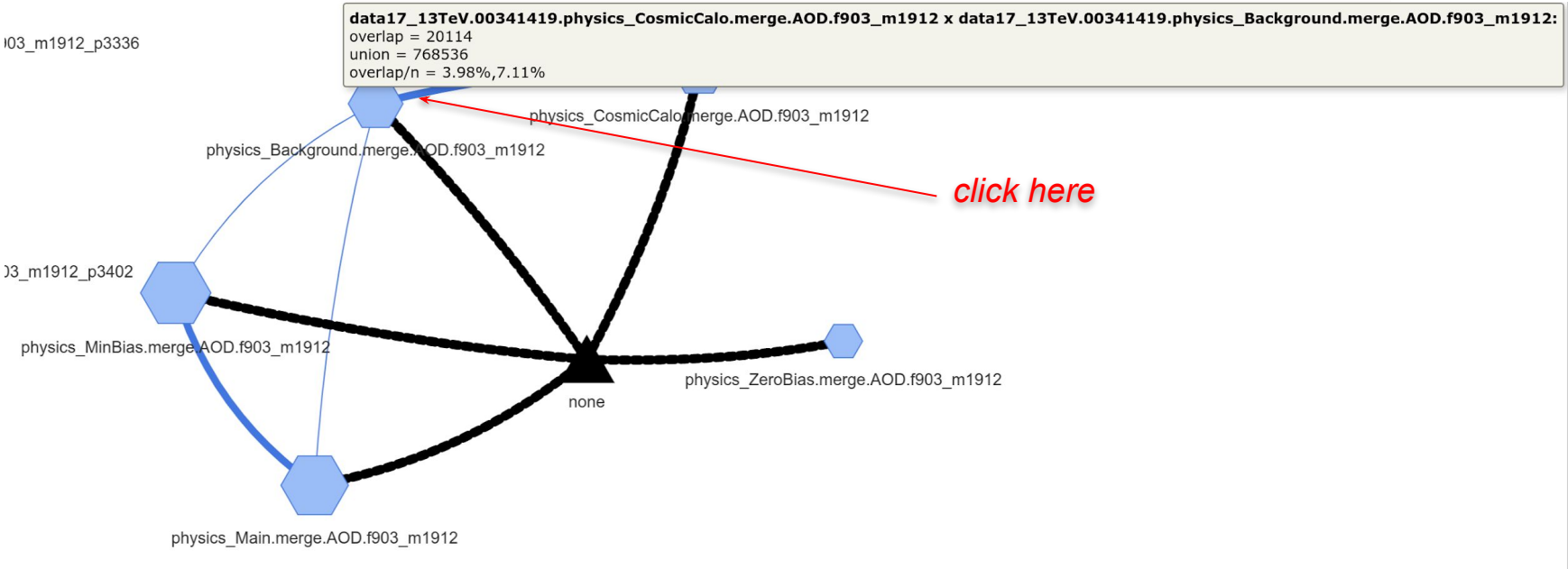
Dataset Details



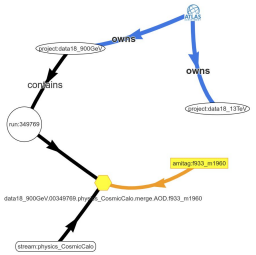


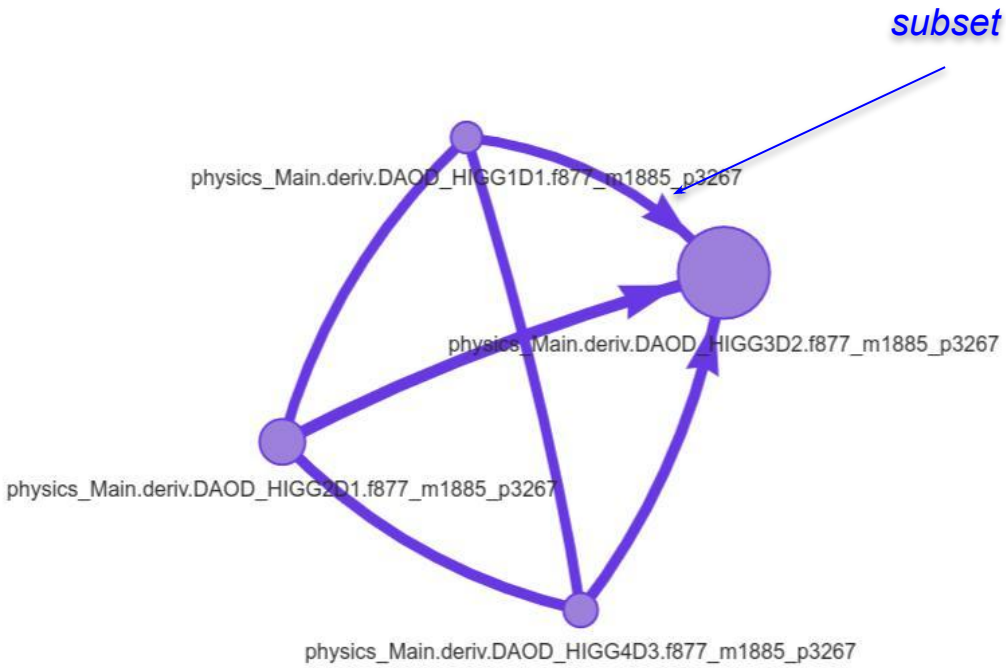
Overlap Details

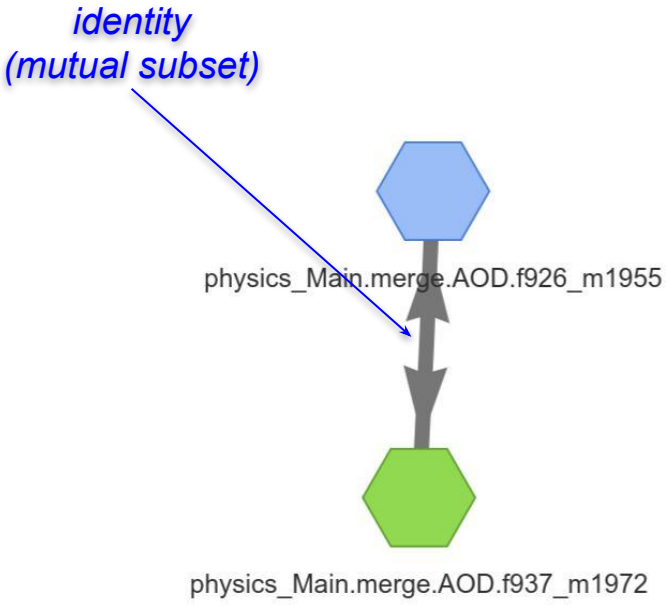




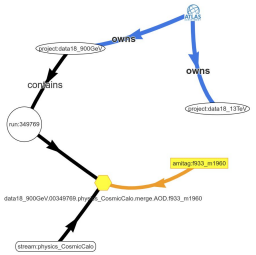
Subsets


owns
owns
contains


subset
physics_Main.deriv.DAOD_HIGG1D1.f877_m1885_p3267
physics_Main.deriv.DAOD_HIGG3D2.f877_m1885_p3267
physics_Main.deriv.DAOD_HIGG2D1.f877_m1885_p3267
physics_Main.deriv.DAOD_HIGG4D3.f877_m1885_p3267


identity
(mutual subset)
physics_Main.merge.AOD.f926_m1955
physics_Main.merge.AOD.f937_m1972

Unique Events Details



click here

03_m1912_p3336

physics_CosmicCalo.merge.AOD.f903_m1912

physics_Background.merge.AOD.f903

data17_13TeV.00341419.physics_Background.merge.AOD.f903_m1912:
unique = 262834
non-unique = 20118
uniqueness = 92.89%

3_m1912_p3402

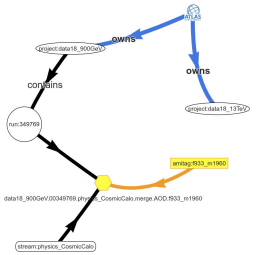
physics_MinBias.merge.AOD.f903_m1912

physics_ZeroBias.merge.AOD.f903_m1912

none

physics_Main.merge.AOD.f903_m1912

All Overlaps of AODs



show all overlaps within a tag

show all overlaps between tags

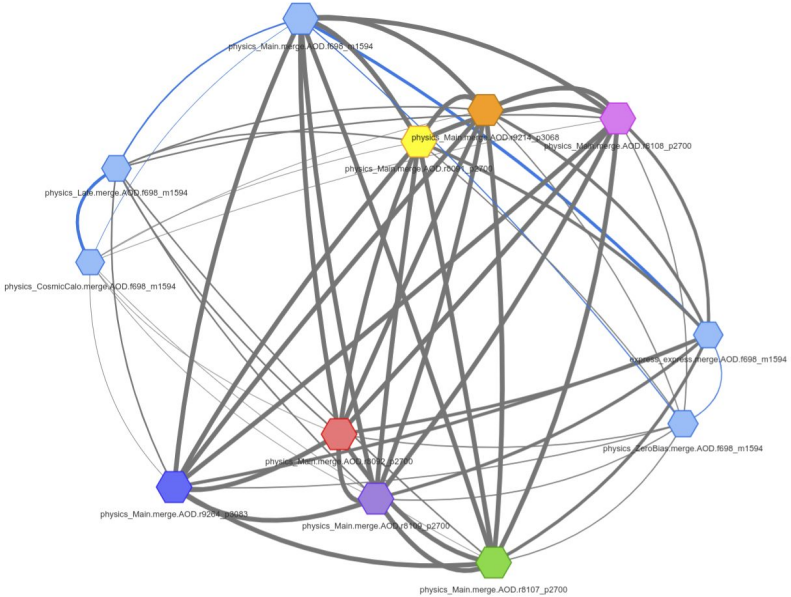
EI16.1/00299184: live Help

Cluster by AMI Tag | Cluster by group size | Expand all clusters

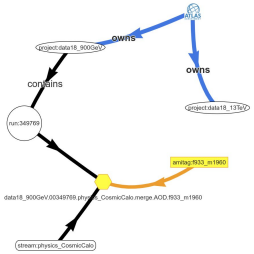
overlap thresholds: 0% 0%

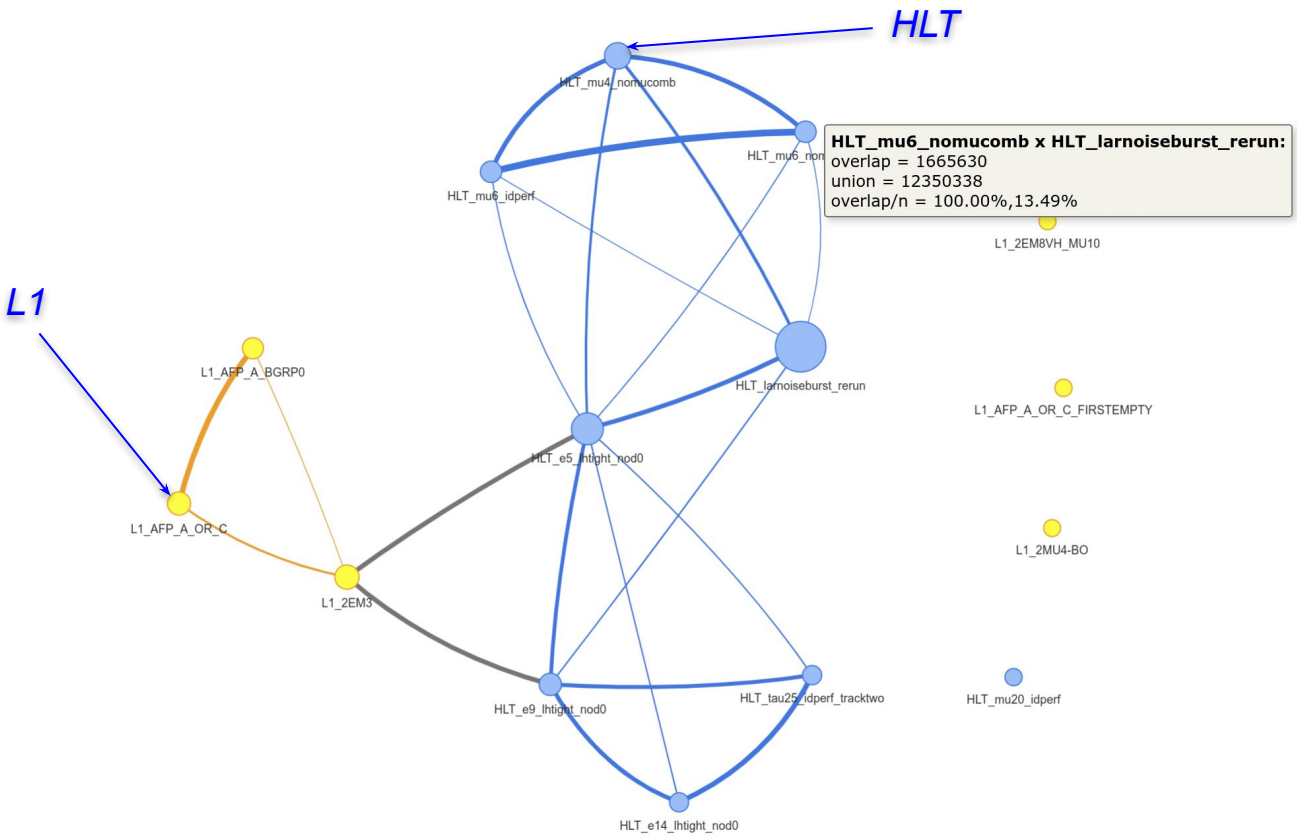
tag level: 1 target: filter: .AOD. Recreate

Context-sensitive menu will be here.

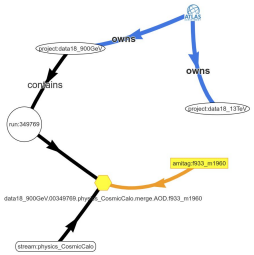


Trigger Overlaps





Clusters



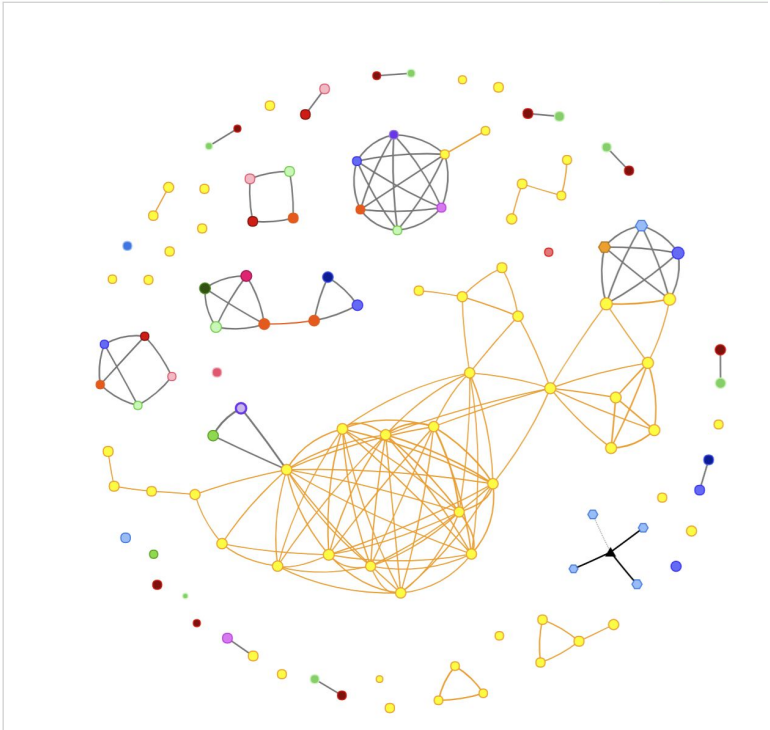
EI16.1/00298690: live Help

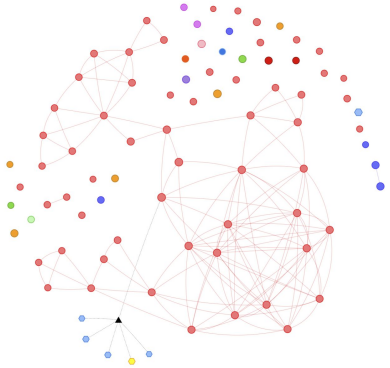
Cluster by AMI Tag | Cluster by group size | Expand all clusters

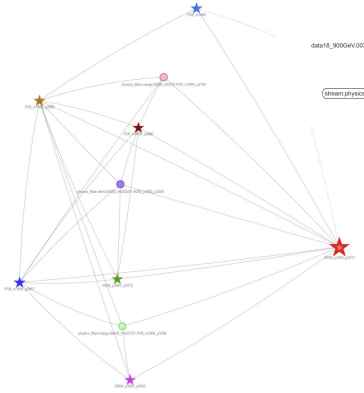
overlap thresholds: 20% 80%

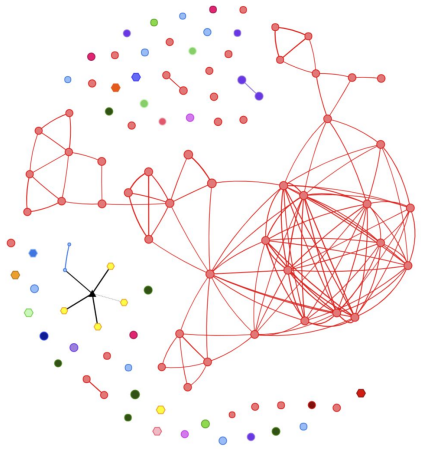
tag level: 99 target: null filter: null Recreate

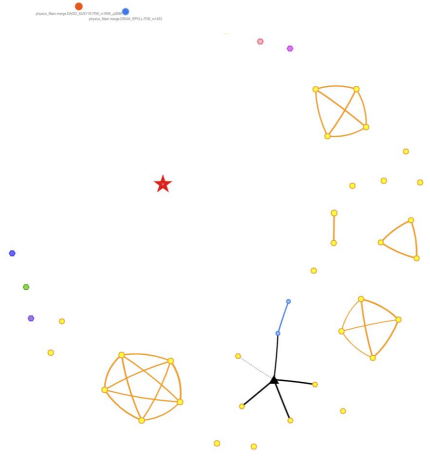
Context-sensi



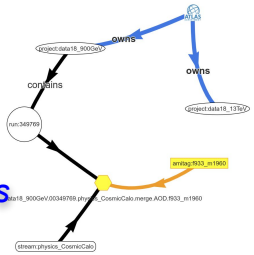








Web Client - Correlogram



Various types of data views

Atlascope 02.00.00+ [02/Oct/2021 at 23:13:57 CEST by atlevind for CERN] Reset

CERN-Proxy add

Search ATLAS

Execute g.V().has('lbl', 'dataset')

canonical:DAOD_SUSY12

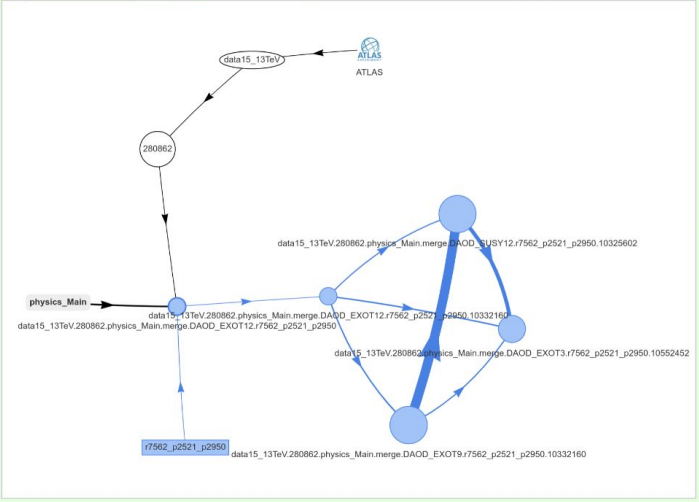
Show - Table -

Customize the interactions with the **graph**.

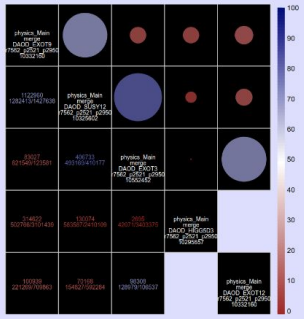
Cluster by group type | Cluster by group size | Expand all clusters | Show all edges | hierarchical (up/lr) size/hierarchy) live

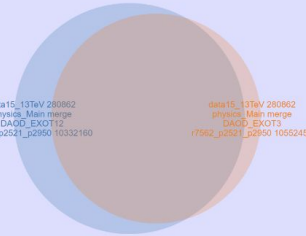
clusterize zoom cluster (stabilize) get children get parents remove old

filter: Apply select: limit(100)



Results | Table | Image | Plot | SkyView





A = data15_13TeV 280862 physics_Main merge DAOD_EXOT12 r7562_p2521_p2950 10332160
B = data15_13TeV 280862 physics_Main merge DAOD_EXOT3 r7562_p2521_p2950 10552452

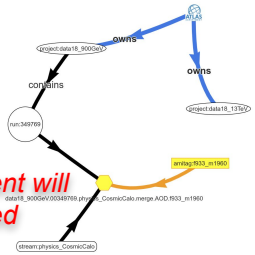
A	128979 (94%)
B	106537 (78%)
A^B	98308 (72%)
A-A^B	30671 (22%)
B-A^B	30671 (6%)
AvB	137208 (100%)

Showing 1 new elements

Sending Gremlin request to //atlas-event-index.cern.ch/Atlascope/Proxy.jsp?server=http://atlas-event-index.cern.ch:8183: g.E("3yd-36w-6uj9-3a0").id().next().toString().replaceFirst("^", "Edge.jsp?id=")

Sending Gremlin request to //atlas-event-index.cern.ch/Atlascope/Proxy.jsp?server=http://atlas-event-index.cern.ch:8183: g.E("3yd-36w-6uj9-3a0").id().next().toString().replaceFirst("^", "Edges.jsp?id=")

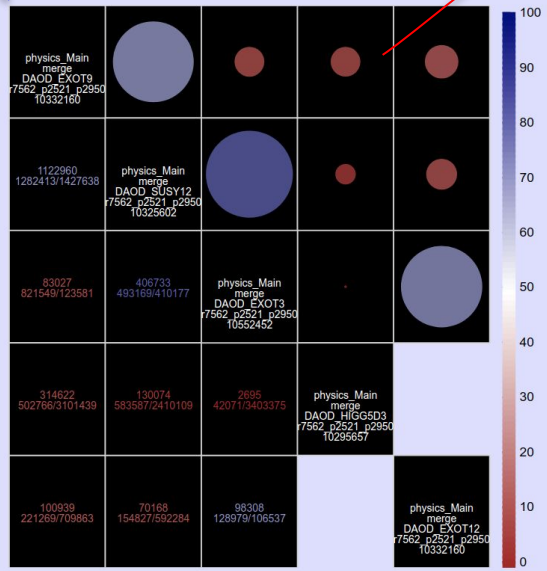
Web Client - Overlaps between Datasets



Dataset names

Overlaps as circles

***Hovering** over an overlap element will give a **Venn** diagram with detailed analyses*



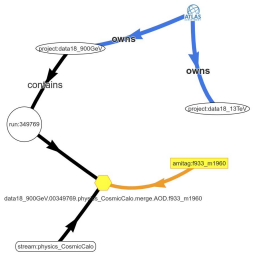
A = data15_13TeV 280862 physics_Main merge DAOD_EXOT12 r7562_p2521_p2950 10332160
B = data15_13TeV 280862 physics_Main merge DAOD_EXOT3 r7562_p2521_p2950 10552452

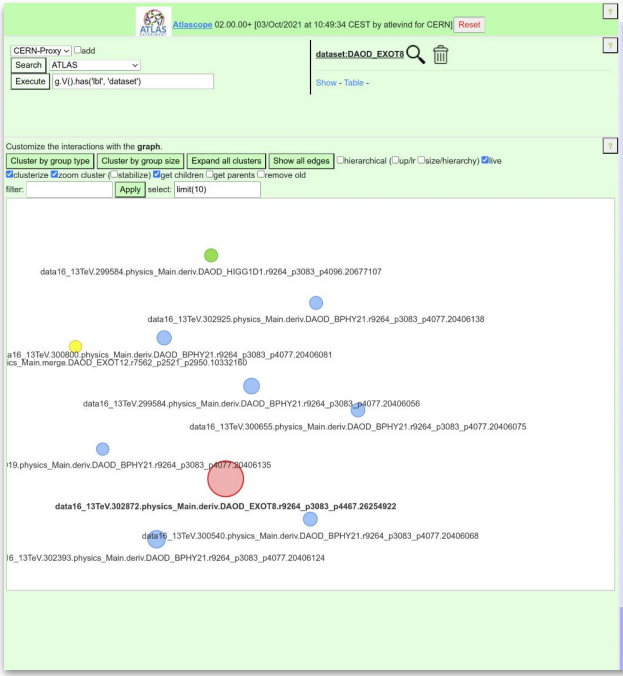
A	128979	(94%)
B	106537	(78%)
A^B	98308	(72%)
A-A^B	30671	(22%)
B-A^B	30671	(6%)
AvB	137208	(100%)

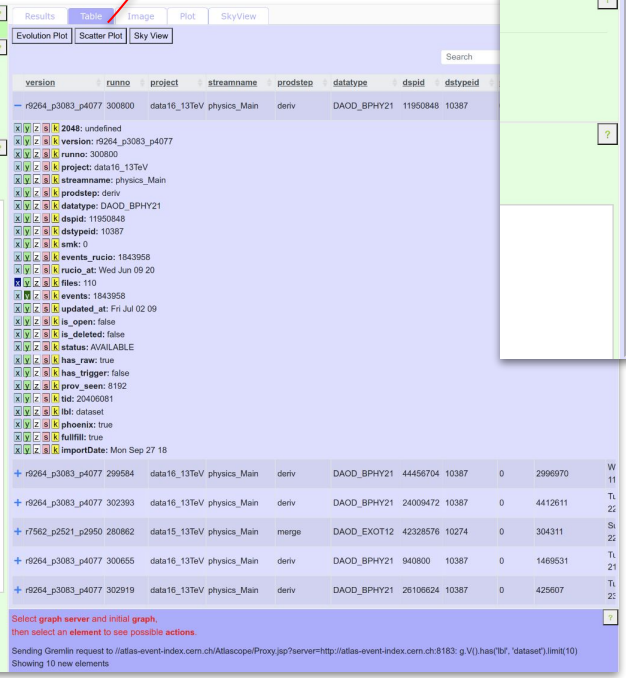
Overlaps as numbers

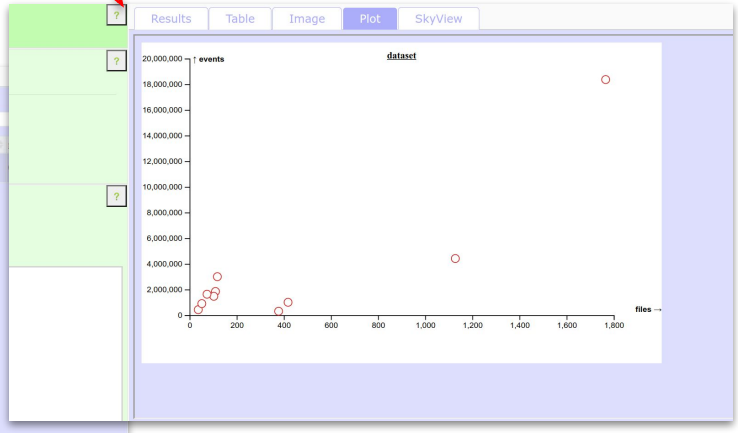
Web Client - Plots

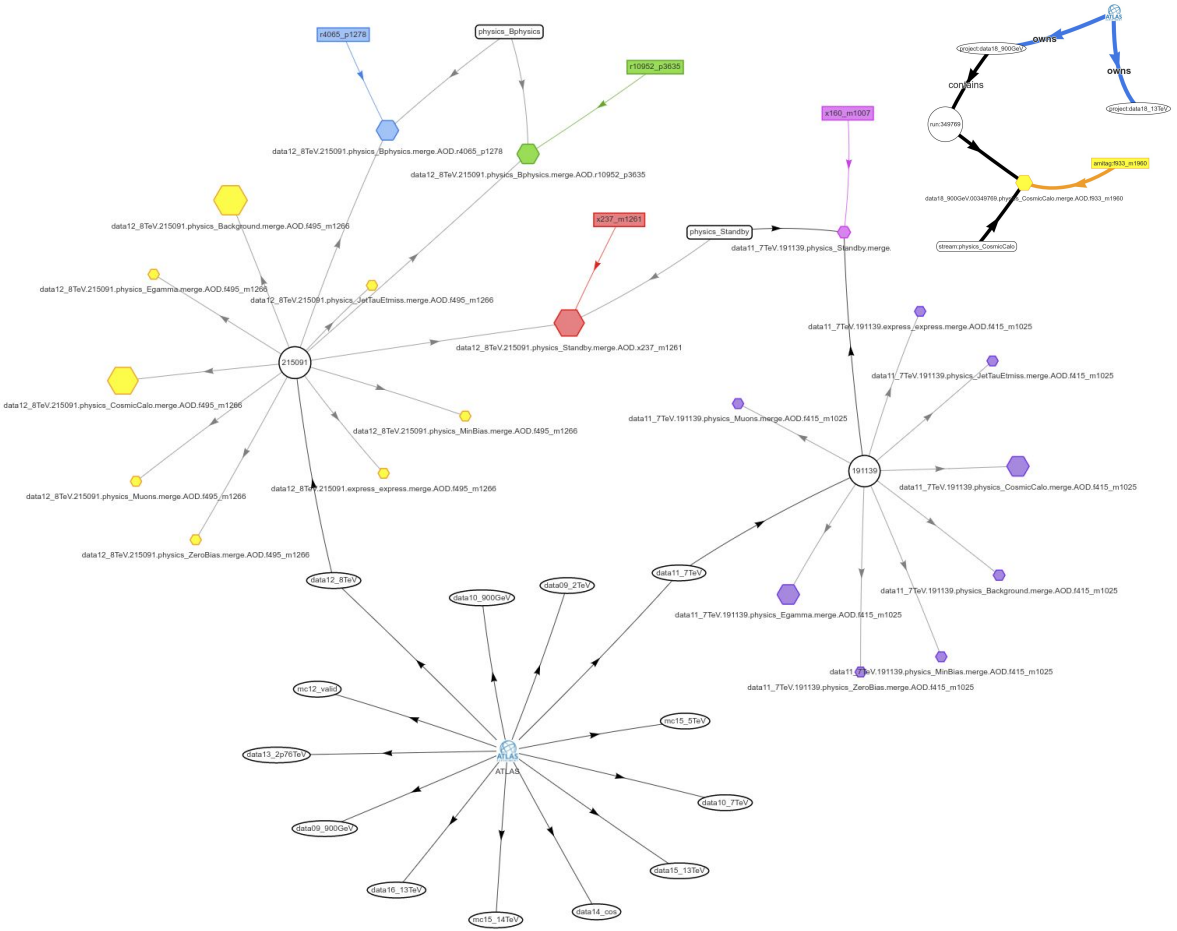
Table columns can be presented in various plot types











- **Event Index Home:**
  https://atlas-event-index.cern.ch/
- **Presentation to CHEP'2019:**
  https://docs.google.com/presentation/d/12UnR3iDWmYYKUQuZ5Hu0RdKbZAXC5uc3-LFKvgBbehs/edit?usp=sharing