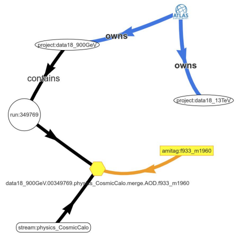
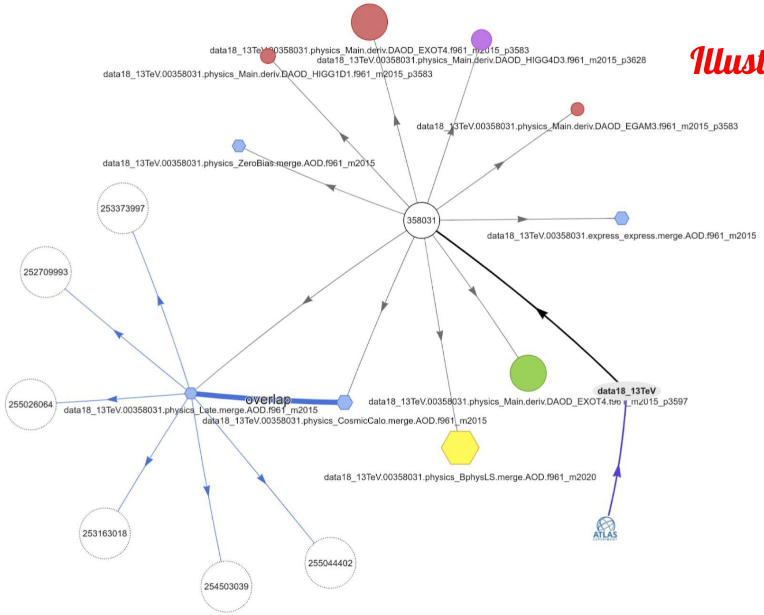


Using Graph Databases

Julius Hrivnac (LAL Orsay) on behalf of the ATLAS Collaboration

CHEP 2019

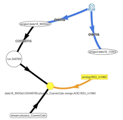


Illustrating Graph Databases potential on new ATLAS Event Index prototypes.

- HEP storage
- Graph Databases
- Atlas Event Index
  - Graph Database Prototypes
- Graph Databases for HEP
  - What they can do for us

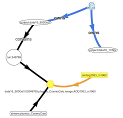
See also Poster 211

ATLAS EXPERIMENT · LABORATOIRE DE L'ACCÉLÉRATEUR LINÉAIRE · UNIVERSITÉ PARIS SUD · université PARIS-SACLAY · cnrs



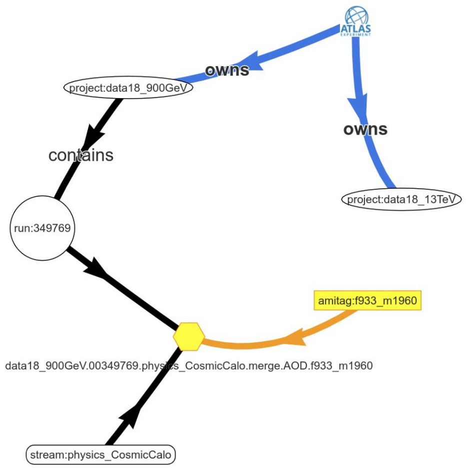
HEP Storage

- ➢ 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

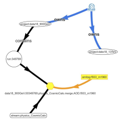


Graph Databases

- ➢ 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

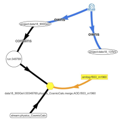
All three methods can be used to access the same database
(e.g. using Cypher for query and Gremlin for traversal).

- 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

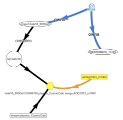
```
g.V().has('run', 'rnumber', 98765)
      .out(‘has’)
      .values(‘name’)
```



Atlas Event Index

- ATLAS Event Index Service keeps references to all real and simulated ATLAS events. Hadoop Map files and HBase tables are used to store the Event Index data, a subset of data is also stored in the Oracle database.
    - Contains information about **events, datasets, streams, runs,...** - and their relations.
- Several user interfaces are currently used to access and search the data. From the simple command line interface, through programmatical API to sophisticated Graphical Web Services.
- History
    - Original Event Index (EI) in Oracle
        - Too rigid (can't easily add columns, relations), other problems
    - Migrated to Hadoop
        - Map files in HDFS
        - Flexible
        - Too slow for searching (ok for processing)
        - Typeless
    - Partially migrated to HBase
        - Two tables: Catalog + Events
        - Tables contain a lot of ad-hoc relations (references to other entries)
            - <u>**We have in fact implemented a poor-man's GraphDB on top of HBase**</u>
    - Several prototypes developed to study next generation Event Index (to be fully deployed for Run-3, end 2020), using Graph Database in different ways:
        - Prototype storing all EI data directly in JanusGraph database over HBase storage
        - Prototype storing data in a HBase table with Phoenix SQL interface, Graph structure added via another auxiliary HBase table

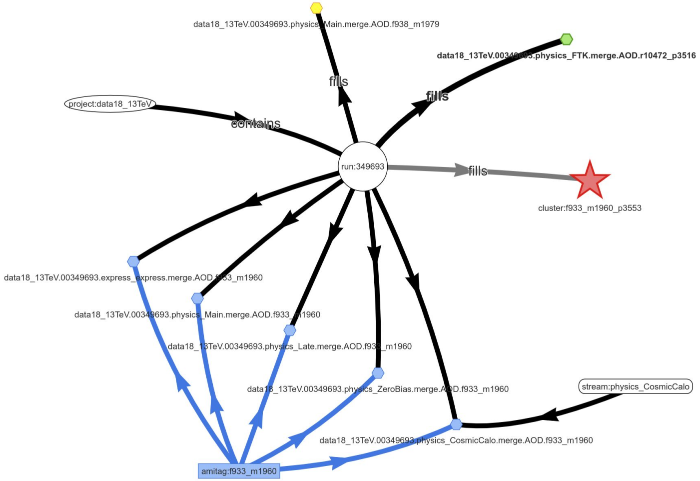
The ATLAS EventIndex for LHC Run 3 - Track 4 on Th at 14:45

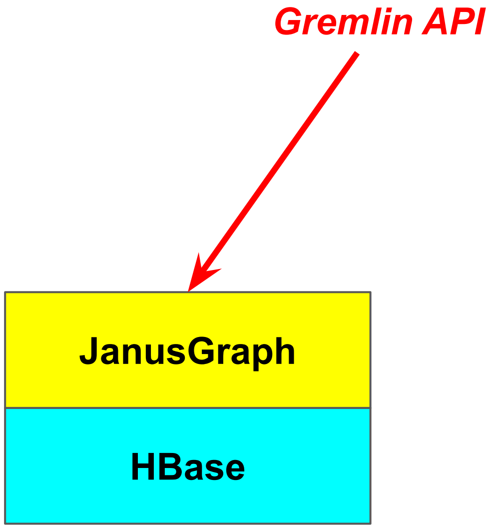


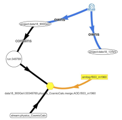
Prototype 1

Prototype 1:
JanusGraph on top of HBase

- ➢ Subset of data imported into full **JanusGraph** Database storing data in an **HBase** table
- ➢ Part of existing functionality implemented
  - ○ Large part of code (handling relations and properties) moved to structure of the graph
- ➢ Most of the graphical part implemented
  - ○ By standalone JS implementation







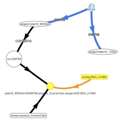
Implementations Choices

Prototype 1

Functional syntax with additional navigational semantics !

- De-facto standard language/api: **Gremlin**
  - <u>Gremlin is a functional, data-flow language to **traverse a property graph**.</u> Every Gremlin traversal is composed of a sequence of (potentially nested) steps. A step performs an atomic operation on the data stream. Every step is either a *map*-step (transforming the objects in the stream), a *filter*-step (removing objects from the stream), or a *sideEffect*-step (computing statistics about the stream).
  - Gremlin supports **transactional** & **non-transactional** processing in **declarative** or **imperative** manner.
  - Gremlin can be expressed in all languages supporting function composition & nesting.
  - Supported languages: **Java**, **Groovy**, **Scala**, **Python**, Ruby, Go, …

- Commonly used framework: **TinkerPop**
- Leading implementation: **JanusGraph**
  - Supported storage backends: Cassandra, **HBase**, Google Cloud, Oracle BerkeleyDB
  - Supported graph data analytics: Spark, Giraph, Hadoop
  - Supported searches: Elastic Search, Solr, Lucene
  - Growing popularity of Neo4J
- Chosen visualisation: **visj.org**
  - Many others exist



Gremlin Syntax

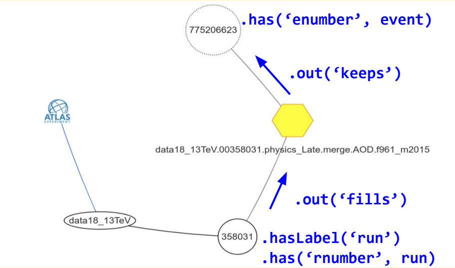
Prototype 1

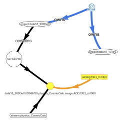
- Functional syntax
- **Functional** & **navigational** semantics
- Very intuitive, **no special syntax needed** (using existing functional syntax), easy integration.
- Database just accessed as objects with structure and relations.
  - Nested collections with links.
- Can use functional API (streams) 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
```

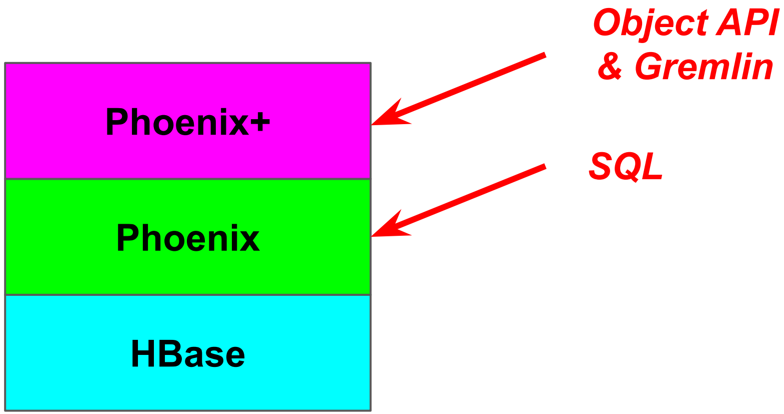




Prototype 2:
Phoenix Project

Prototype 2

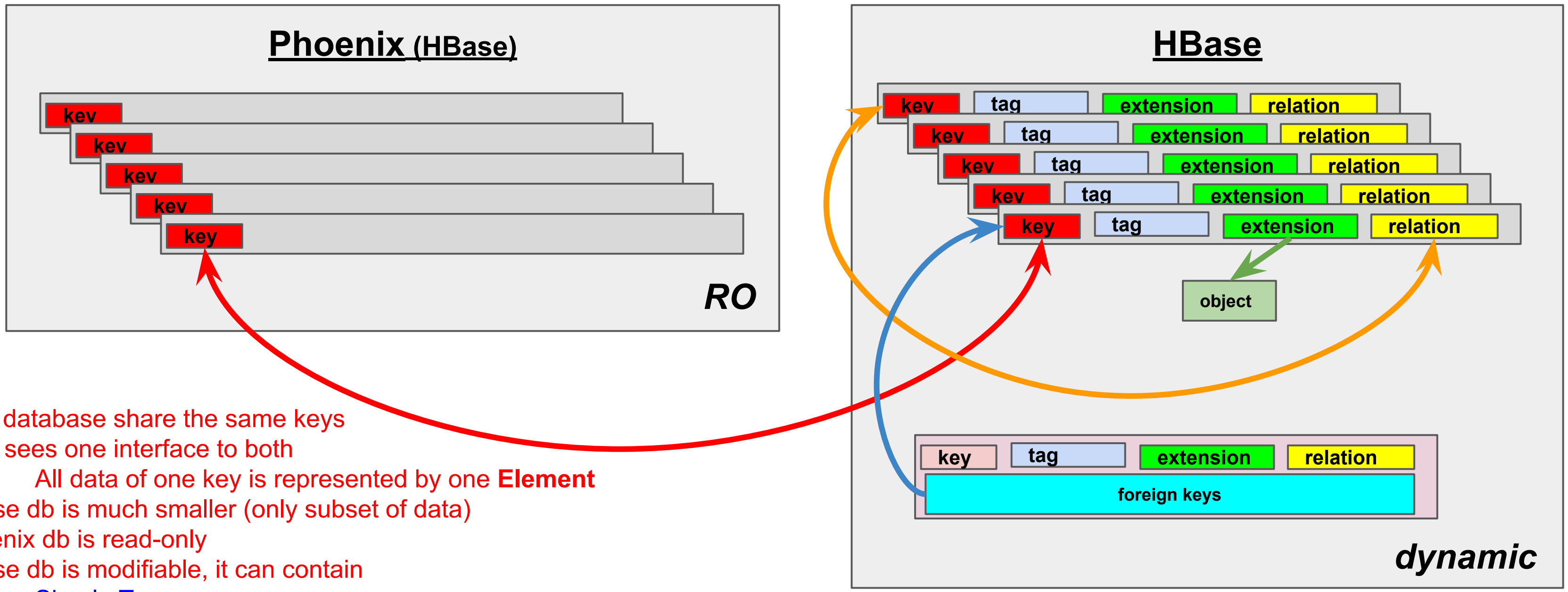
- ➢ Aim
  - **Simple and flexible NoSQL storage (HBase)**
    - Good experience
  - **Compatibility with other SQL database (Phoenix API)**
    - SQL used in other ATLAS components
  - **Advantages of Graph Database**
- ➢ Solution
  - Extend Phoenix/HBase storage with pure HBase storage
    - Sharing the same keys
  - So keeping Phoenix advantages (speed, SQL interface) for RO data
  - While opening for new possibilities, adaptability to changing environment
  - Phoenix/HBase for static data, HBase for dynamic data
- ➢ Data stored in pure HBase table with Phoenix SQL API
- ➢ Graph structure added via aux HBase table
  - ‘Lazy Graphs’: only created when needed
- ➢ Home-grown ‘object API’
  - With Gremlin subset API



Architecture

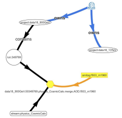
Prototype 2

Creating Graph structure on top of an SQL database - a user sees <u>one</u> database.

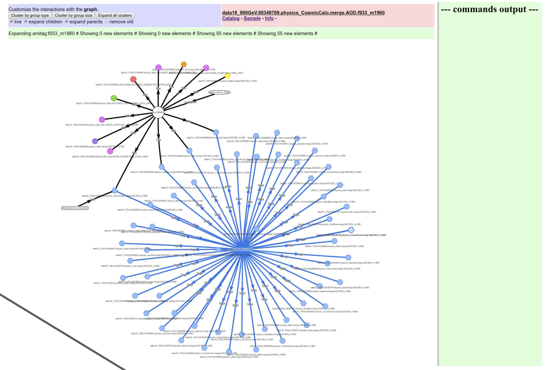


- ➢ Both database share the same keys
- ➢ User sees one interface to both
  - All data of one key is represented by one **Element**
- ➢ HBase db is much smaller (only subset of data)
- ➢ Phoenix db is read-only
- ➢ HBase db is modifiable, it can contain
  - Simple **Tags**
    - They can be also used in search filter
  - **Extensions** with any object
    - E.g. Trigger statistics and overlap, duplicated events list,...
  - **Relations** to other elements (like Graph DB)
    - E.g. overlaps between datasets

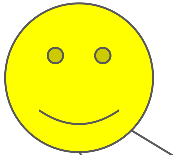
- ➢ HBase can also contain Elements without Phoenix partner: **Hubs**
  - They represent pre-defined **virtual collections** of Elements
    - E.g. Amitag, Stream, Run, Project,...
  - They can be extended and searched in the same way as other Elements
- ➢ Ad-hoc virtual collections can be build also using Tags

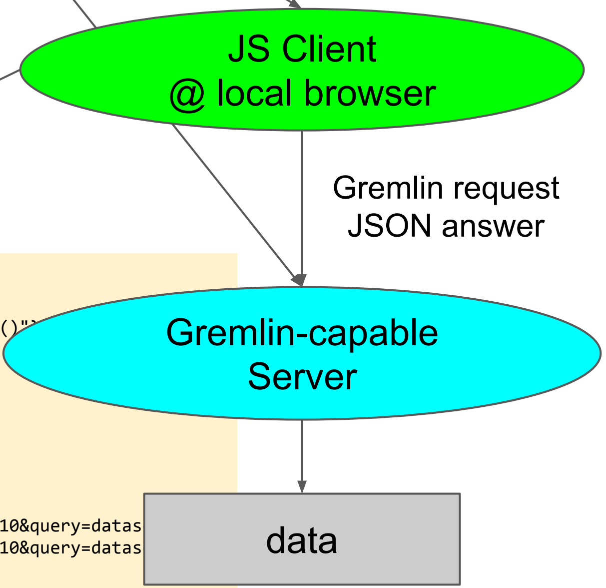


GUI WS



- **Generic Web Service graphical visualisation**
  - Can display **any Gremlin**-compatible database
    - Visualisation can be customised via Stylesheet (JSON)
  - Implemented **completely in JavaScript**
    - So doesn't need server-side application
    - Connects to standard Gremlin server to get JSON view of data






```
"dataset": {
      graphics: {
        label:{gremlin:"sideEffect(values('prodStep').store('4')).sideEffect(values('dataType')......values().join().toString()"
        title:"dataType",
        subtitle:{gremlin:"values('nevents').join().toString().concat(' events')"},
        group:{gremlin:"in().hasLabel('amitag').values('version')"},
        shape:{js:"if(title=='dataset:AOD') {shape = 'hexagon';} else {shape = 'dot';}"},
        value:{gremlin:"values('nevents').join().toString()"}
        },
      actions: [
        {name:"Catalog", url:"https://atlas-event-index.cern.ch/EIHadoop/CatalogView.jsp?query=dataset:"},
        {name:"Sample" , url:"https://atlas-event-index.cern.ch/EIHadoop/InspectView.jsp?view=txt&action=dump&climit=1&limit=10&query=datas
        {name:"Info" ,   url:"https://atlas-event-index.cern.ch/EIHadoop/InspectView.jsp?view=txt&action=info&climit=1&limit=10&query=datas
        ]
      },
```

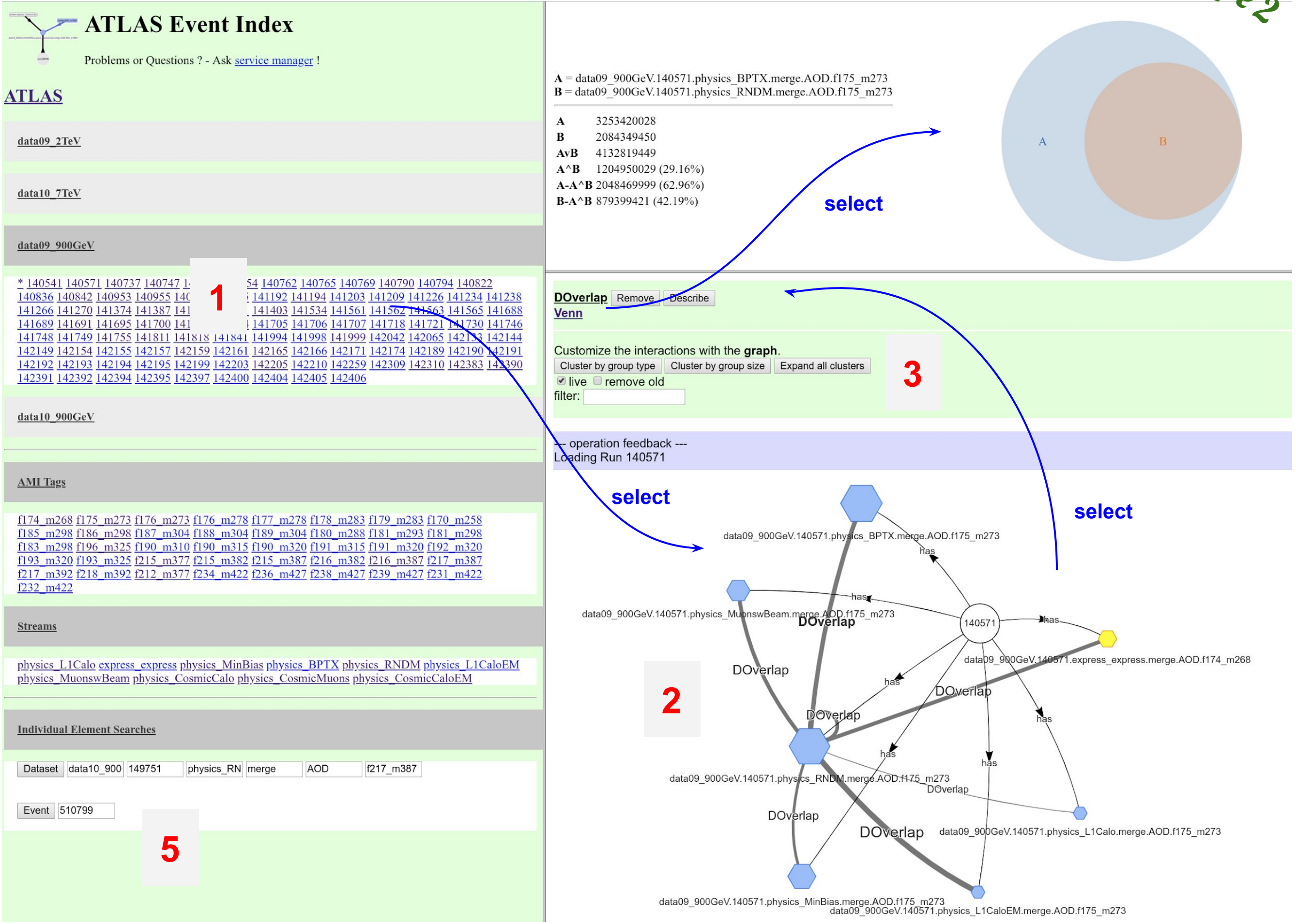
WS GUI

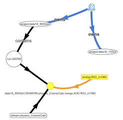
Prototype 2

Direct API and REST Web Service also available.

- ➢ **Very generic**
- ➢ **Very thin layer on top of data**
  - ○ **Structure is already there**
- ➢ **Hierarchical navigation**
  - ○ **A'la Google Earth**
- ➢ **Show all available data, their relations and properties**
  - ○ **And all available actions for them**

1. Select period, run, ami tag, stream, dataset or event
   a. Or any other Hub (virtual collection)
2. Get it, together with all related entities and information stored in EI
3. Each entity has a set of possible actions to perform
4. Each entity can be annotated
5. Detailed search is possible too

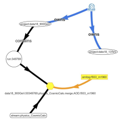




Added Values & New Possibilities

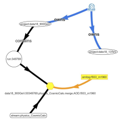
for Event Index

- <u>Big part of the application code absorbed in Graph Database structure</u>
  - Delegate implementation/optimisation details to suitable database framework
  - Database carries information about data structure which would otherwise have to be handled by the application code
- Simple graphical access
  - JavaScript client connects directly do Gremlin server
- Using standards
  - So components can be replaced
    - JanusGraph with Neo4J,...
    - Hadoop with Cassandra,...
- Possibility of Creation of Virtual Entities
  - Virtual Collections
    - ‘Whiteboard’ functionality
    - Can create API allowing users creating their own (or group-wise) Collections
  - Persistent Requests results
    - Results can be stored as new objects with relations (cache functionality)
  - Query Spaces
    - Abstract space on top of data to allow faster searching and navigation
- Finding Higher level correlations
  - Multiple trigger overlaps between events, trigger pattern in data derivations,...



Performance

- Requests in general in three phases
  - First search of the initial entry point (event, dataset, run,...)
    - Could be optimised
      - Natural order
      - Indexes
      - Elastic Search
      - Spark
      - More hierarchical navigation
  - Then navigation on the graph
    - Very fast
  - And finally accumulation of results
- Data can still be accessed directly, without Graph Database API
  - So with the same performance as non-GraphDB
  - Navigational step (instead of sub-search) can only speed it up
- In general:
  - Very fast retrieval
  - Slower import
    - Because import creates structures
    - Which are used in retrieval (simpler & faster)

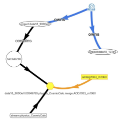


General Comments

Graph Databases for <u>Functional Programming</u>

- Relations (edges) can be considered as functions
    - Navigation as a function execution
    - **From the user point of view, there is no difference in creating new object or navigating to it**
        - Both operation can be ‘lazy’
- Functional processing and graph navigation (“Graph Oriented Programming”) can work very well together
    - Using the same functional syntax
    - Both are realisation of Categories
        - Vertex == object, Edge == morphism
        - Functional program can be modeled as a Graphs
        - Graph data can be navigated using functions
    - Data **ready for parallel access**
- Very well implemented by Gremlin

Extending parallel-ready functional model from code to data !

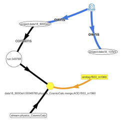


General Comments

Graph Databases for <u>Deep Learning</u>

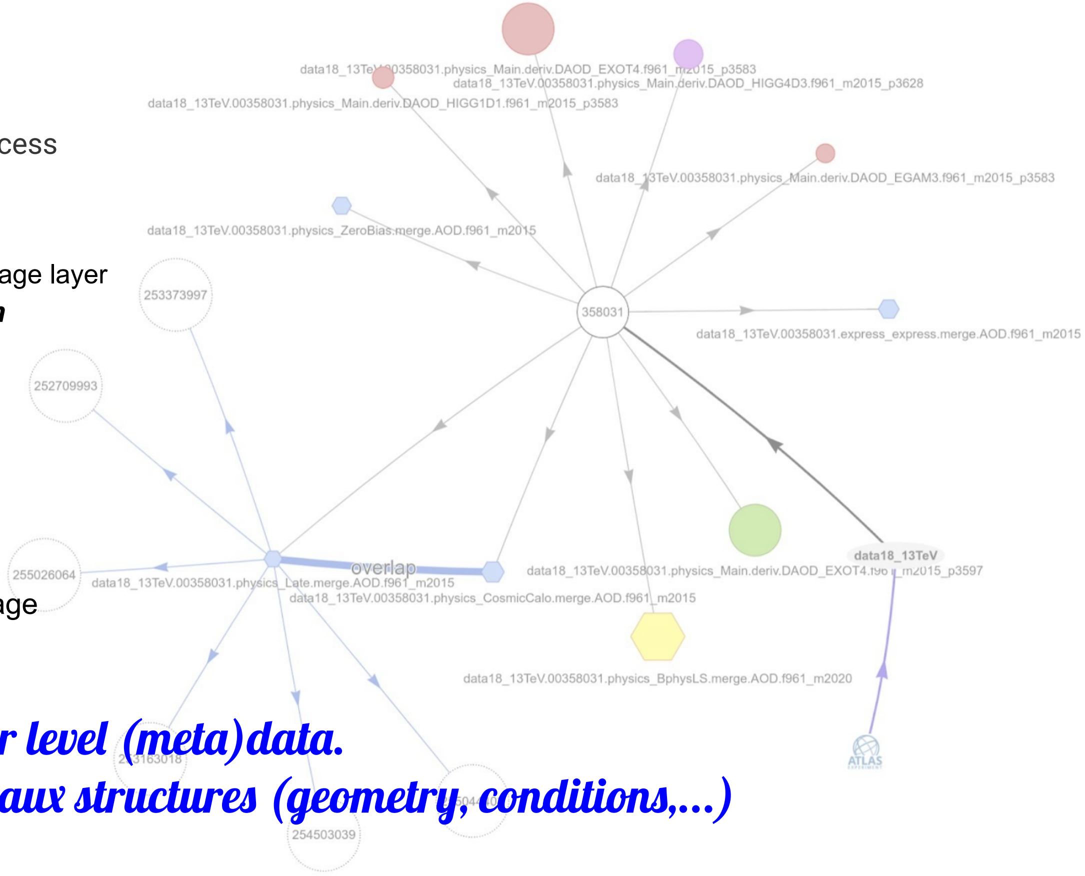
- Neural Network itself is a Graph
  - Using Graph Database to describe NN itself
- In many cases, Neural Network handles Graph data (objects with relations)
  - They can operate either on individual nodes (Node-focused tasks)
  - Or on the whole graph (Graph-focused tasks)
- GraphNN can be seen as a generalisation of ConvolutionalNN
  - Non-geometric
- Possibility to impose constraints/knowledge to NN
  - Inductive Bias
  - Semantic Induction

[Graph Neural Networks](#) create a Natural environment for Deep Learning !

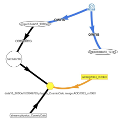


Graph Databases for HEP

- ➢ 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 data in a real Graph database
  - ○ Build a Graph layer on top of the existing storage
    - ■ Close to DB layer
    - ■ In the application layer

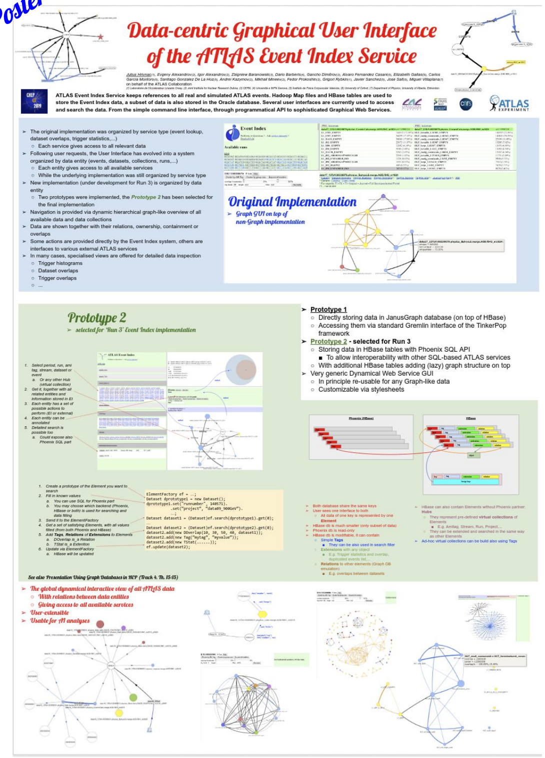


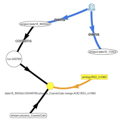
Described project uses Graphs to store higher level (meta)data.
Graphs can be also used to store Events and aux structures (geometry, conditions,...) to give Graph functionality to all data.



Additional Information

See also Poster 211

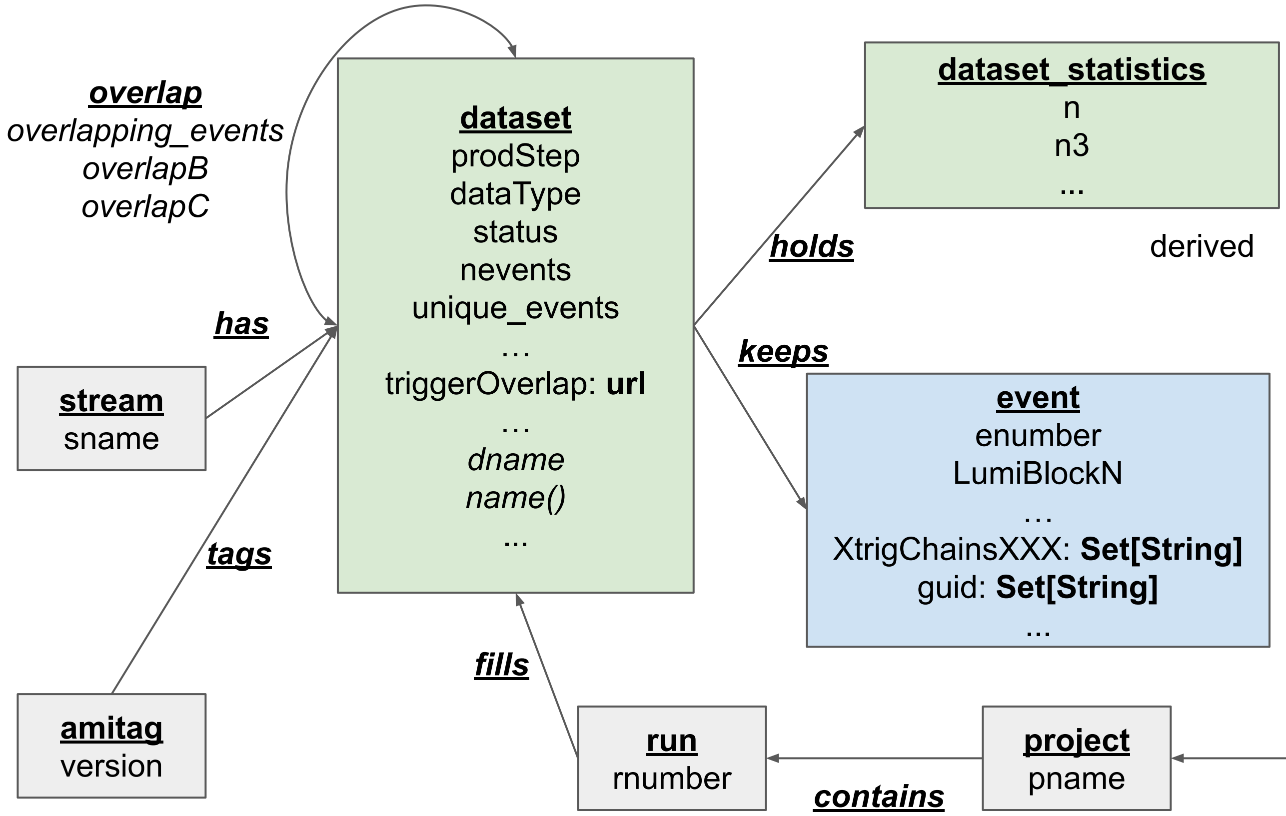




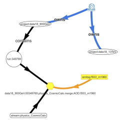
Schema may be specified to speed up searching and navigation.
Other nodes & vertexes can be added.

Schema

Prototype 1



- Should decide what is a feature and what is a relation
- Arrows have only logical meaning, they can be navigated equally from both sides
- Relations and features have defined multiplicities (not shown here) and types (int, string, date,..., Set[...],...)
- Defined entities (and combinations of them) can be indexed for fast search
- Other features and relations can be freely added to any entity (vertex or edge)



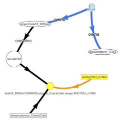
Object API

Prototype 2

- ➢ *REST and GUI WS are build on top of this API*
- ➢ *Subset of Gremlin API also available*

1. Create a prototype of the Element you want to search
2. Fill in known values
   a. You can use SQL for Phoenix part
   b. You may choose which backend (Phoenix, HBase or both) is used for searching and data filling
3. Send it to the ElementFactory
4. Get a set of satisfying Elements, with all values filled (from both Phoenix and HBase)
5. Add Tags, Relations of Extensions to Elements
   a. DOverlap is_a Relation
   b. TStat is_a Extention
6. Update via ElementFactory
   a. HBase will be updated

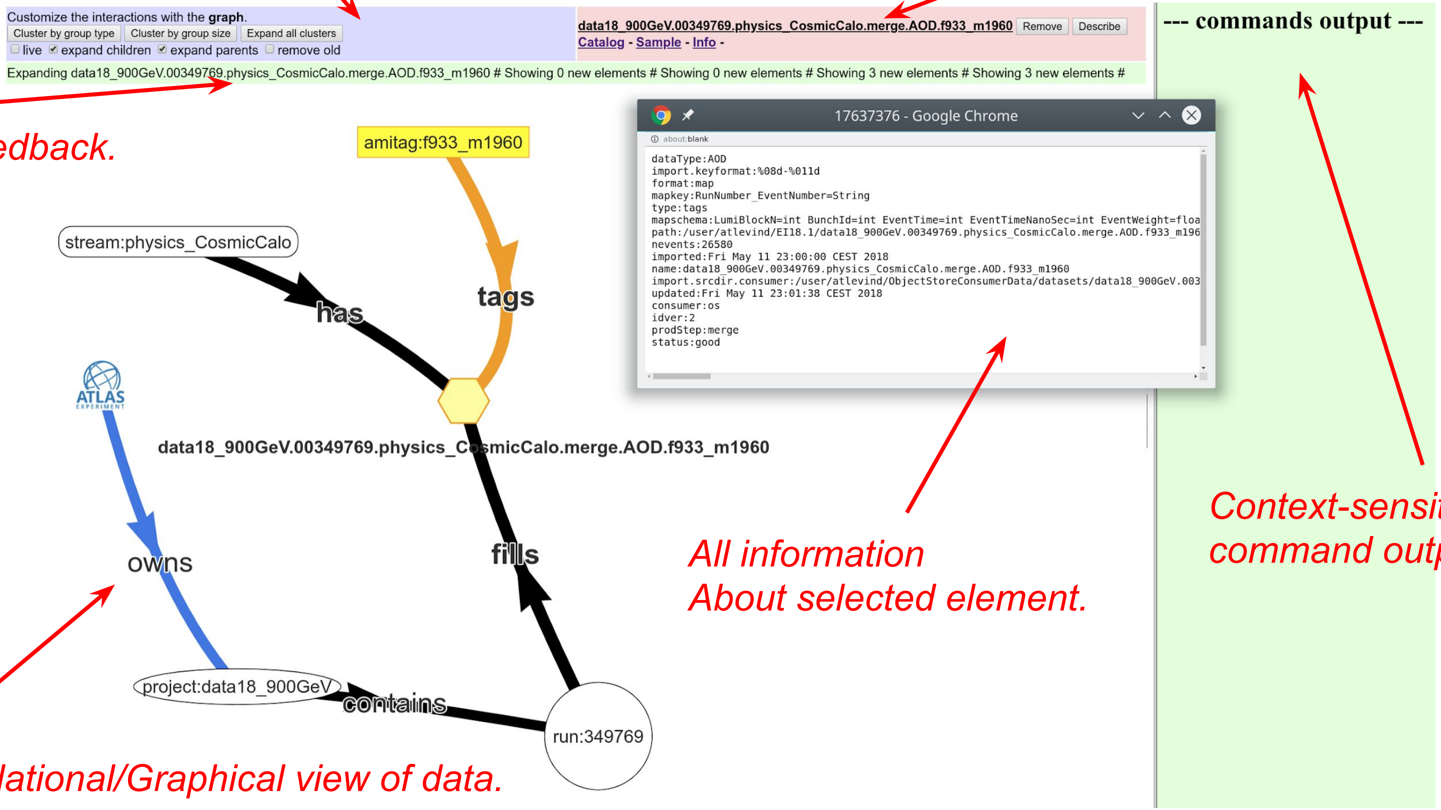
```
ElementFactory ef = …;
Dataset dprototype1 = new Dataset();
dprototype1.set("runnumber", 140571).
           .set("project", "data09_900GeV").
           …;
Dataset dataset1 = (Dataset)ef.search(dprototype1).get(0);
…
Dataset dataset2 = (Dataset)ef.search(dprototype2).get(0);
dataset2.add(new DOverlap(10, 30, 50, 40, dataset1));
dataset2.add(new Tag("mytag", "myvalue"));
dataset2.add(new TStat(......));
ef.update(dataset2);
```



GUI WS

Operations on View.

Context-sensitive action on selected element(s).

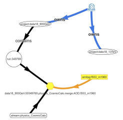


Operation feedback.

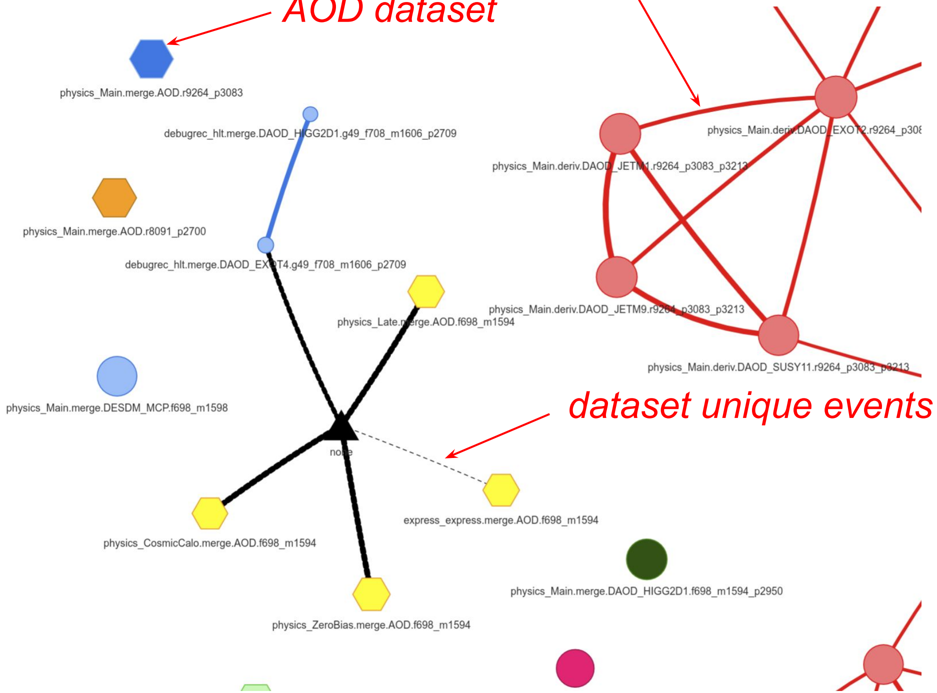
All information About selected element.

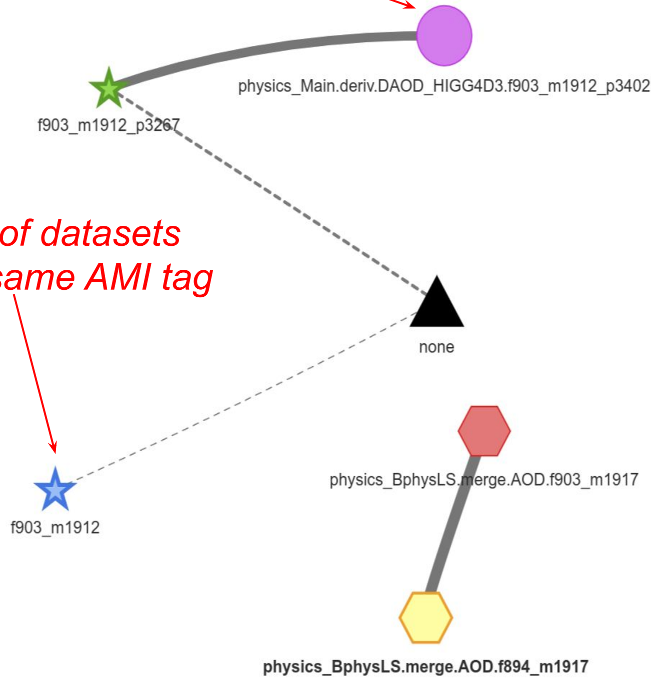
Context-sensitive action command output.

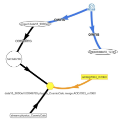
InterActive Relational/Graphical view of data.



Datasets Graphical View







Overlaps, Unique Events

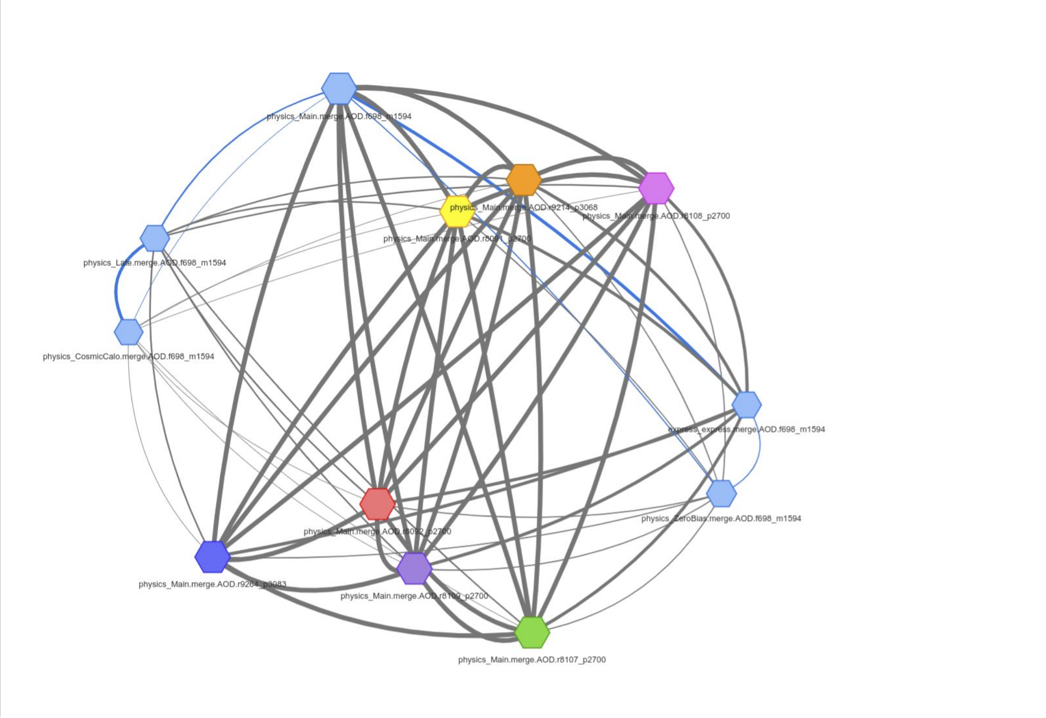
EI16.1/00299184: ☑ live Help

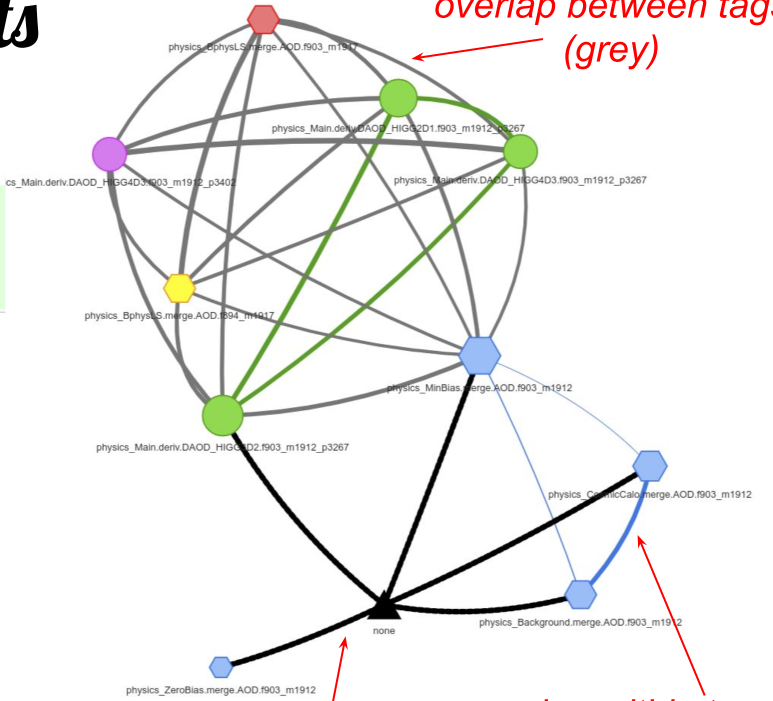
Cluster by AMI Tag | Cluster by group size | Expand all clusters

overlap thresholds: 0% 0%

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

Context-sensitive menu will be here.



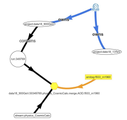


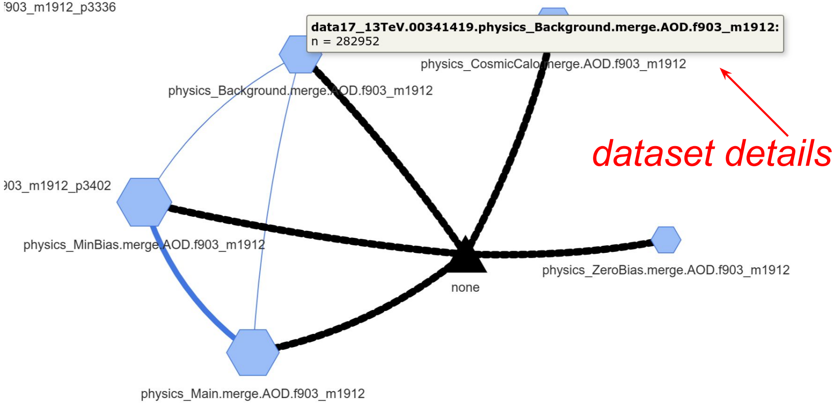
overlap between tags (grey)

overlap within tag (with color of the tag)

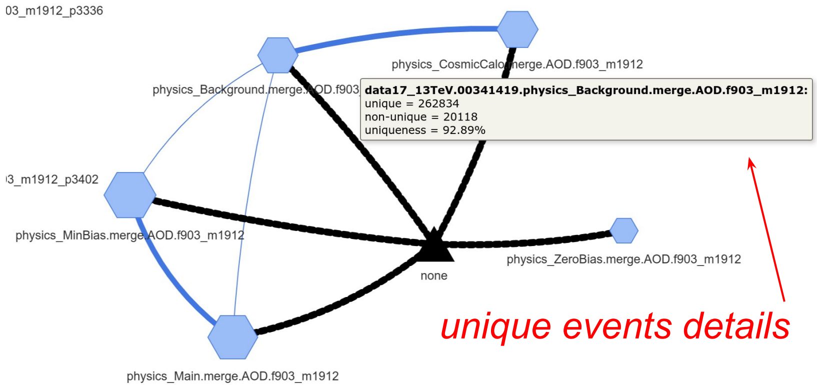
unique events (black dashed)

Details



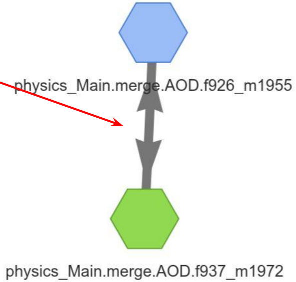


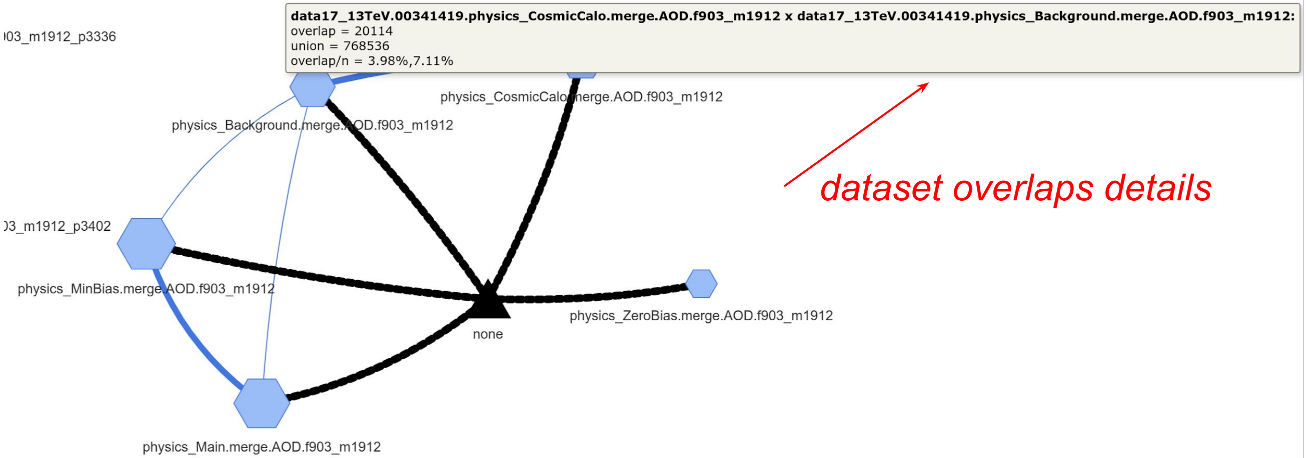
dataset details



unique events details

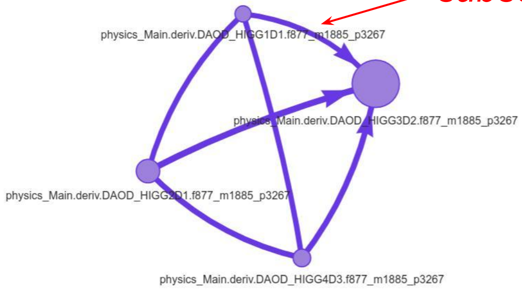
identity
(mutual subset)

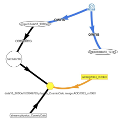




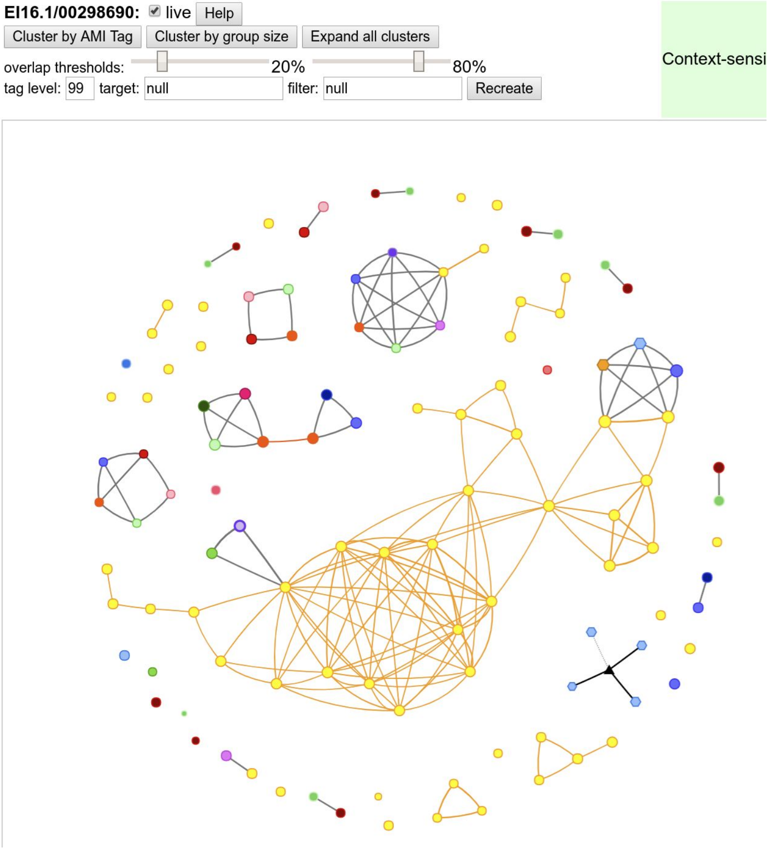
dataset overlaps details

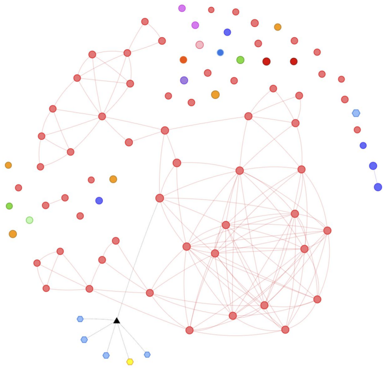
subset

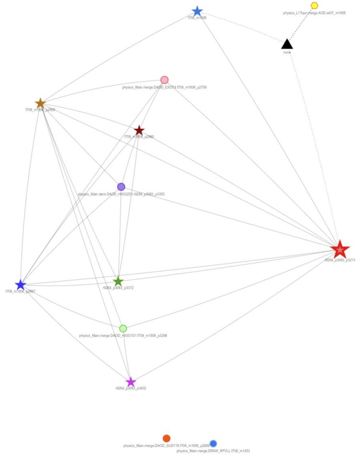


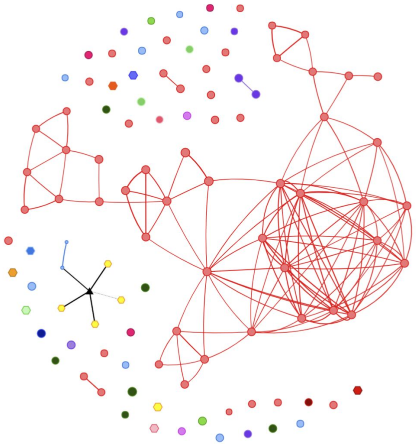


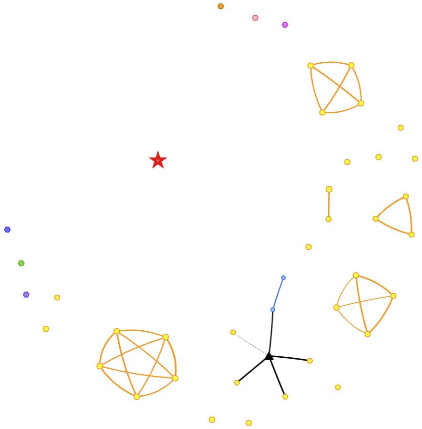
Clusters

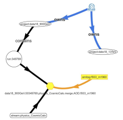

EI16.1/00298690:
live
Help
Cluster by AMI Tag
Cluster by group size
Expand all clusters
Context-sensi
overlap thresholds:
20%
80%
tag level: 99
target: null
filter: null
Recreate



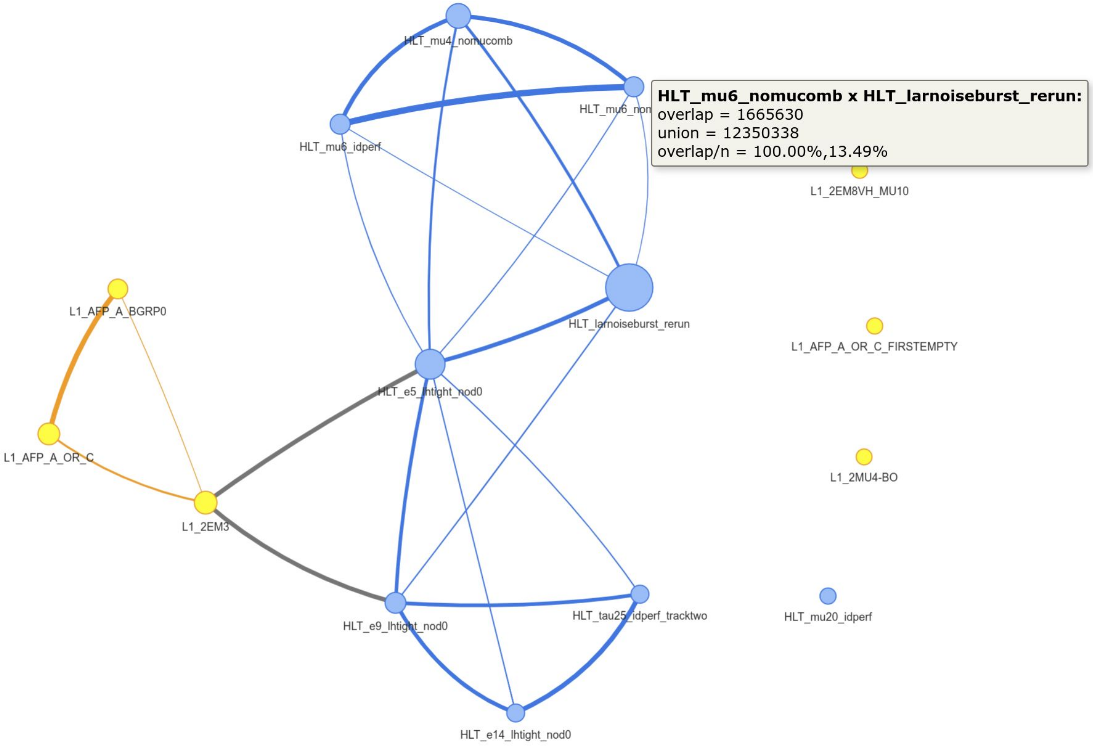


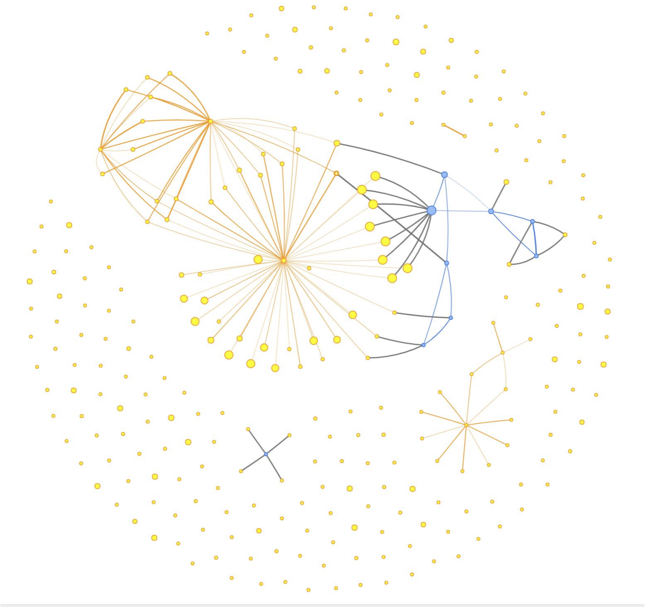




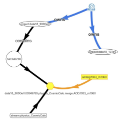


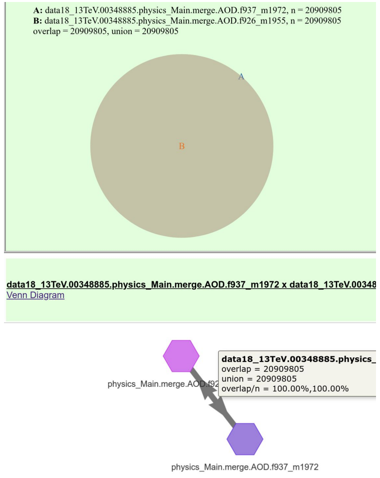
Trigger Overlaps

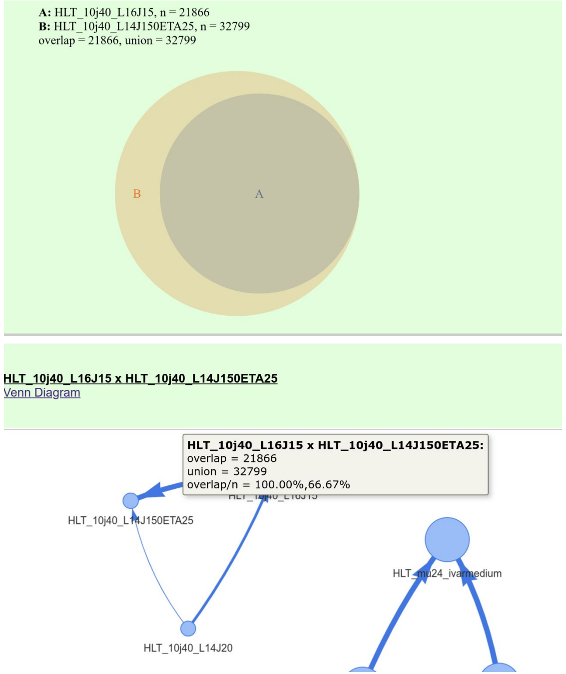


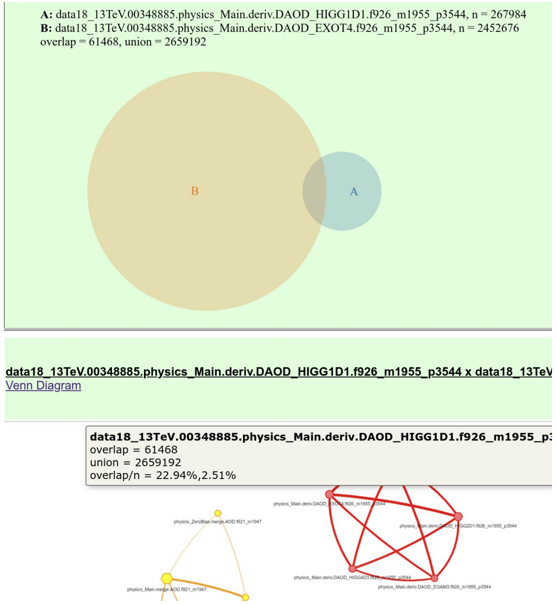


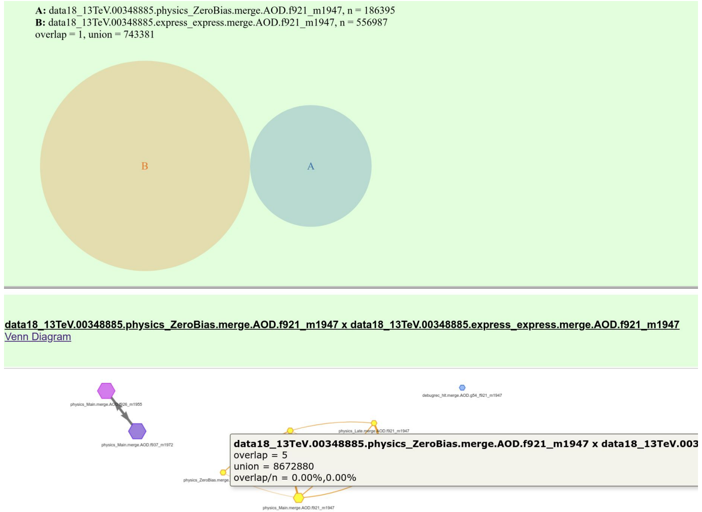
Overlaps Venn Diagrams

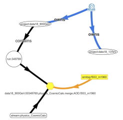




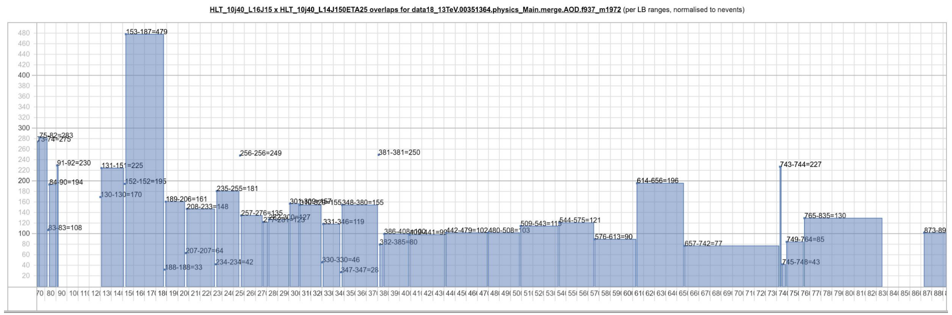


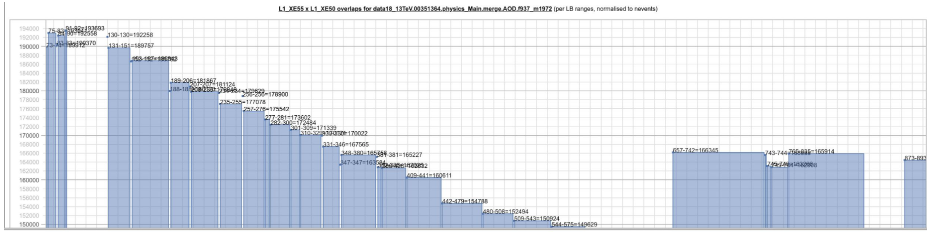




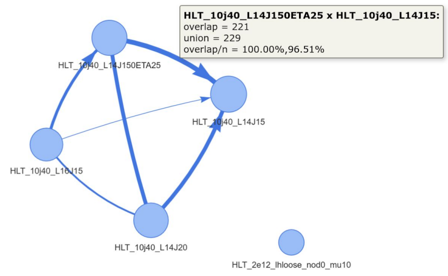


Trigger Overlaps for LB Ranges

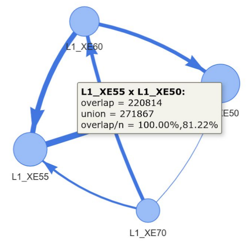


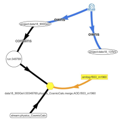


HLT_10j40_L16J15 x HLT_10j40_L14J150ETA25
Venn Diagram - LB Graph



L1_XE55 x L1_XE50
Venn Diagram - LB Graph(*)





Performance Example

Event Lookup

75% of the time is spend by the entry point search, following graph traversal is very fast.

This was the first request, the second one will be cca 10x faster (even on different event).

```
gremlin> el(358031, 775206623, g).profile()
==>Traversal Metrics
Step                                                               Count  Traversers       Time (ms)    % Dur
=============================================================================================================
JanusGraphStep([],[~label.eq(event), enumber.eq...                     1           1         204.805    75.74
    \_condition=(~label = event AND enumber = 775206623)
    \_isFitted=true
    \_query=multiKSQ[1]@2147483647
    \_index=event:enumber:u
    \_orders=[]
    \_isOrdered=true
  optimization                                                                                 4.614
  optimization                                                                               130.444
  backend-query                                                        1                       7.742
    \_query=event:enumber:u:multiKSQ[1]@2147483647
JanusGraphVertexStep(IN,[keeps],vertex)                                1           1          25.560     9.45
    \_condition=type[keeps]
    \_isFitted=true
    \_vertices=1
    \_query=org.janusgraph.diskstorage.keycolumnvalue.SliceQuery@b3a55b7f
    \_orders=[]
    \_isOrdered=true
  optimization                                                                                11.927
  backend-query                                                        1                       3.103
    \_query=org.janusgraph.diskstorage.keycolumnvalue.SliceQuery@b3a55b7f
JanusGraphVertexStep(IN,[fills],vertex)                                1           1          10.388     3.84
    \_condition=type[fills]
    \_isFitted=true
    \_vertices=1
    \_query=org.janusgraph.diskstorage.keycolumnvalue.SliceQuery@b3a605c1
    \_orders=[]
    \_isOrdered=true
  optimization                                                                                 7.661
  backend-query                                                        1                       1.442
    \_query=org.janusgraph.diskstorage.keycolumnvalue.SliceQuery@b3a605c1
HasStep([rnumber.eq(358031)])                                          1           1          13.129     4.86
SelectOneStep(last,e)                                                  1           1           0.993     0.37
NoOpBarrierStep(2500)                                                  1           1           0.159     0.06
JanusGraphPropertiesStep([guid],value)                                 2           2          14.800     5.47
    \_condition=type[guid]
    \_isFitted=true
    \_vertices=1
    \_query=org.janusgraph.diskstorage.keycolumnvalue.SliceQuery@b11f98a7
    \_orders=[]
    \_isOrdered=true
  optimization                                                                                 7.478
NoOpBarrierStep(2500)                                                  2           2           0.568     0.21
                                              >TOTAL                   -           -         270.406        -
```