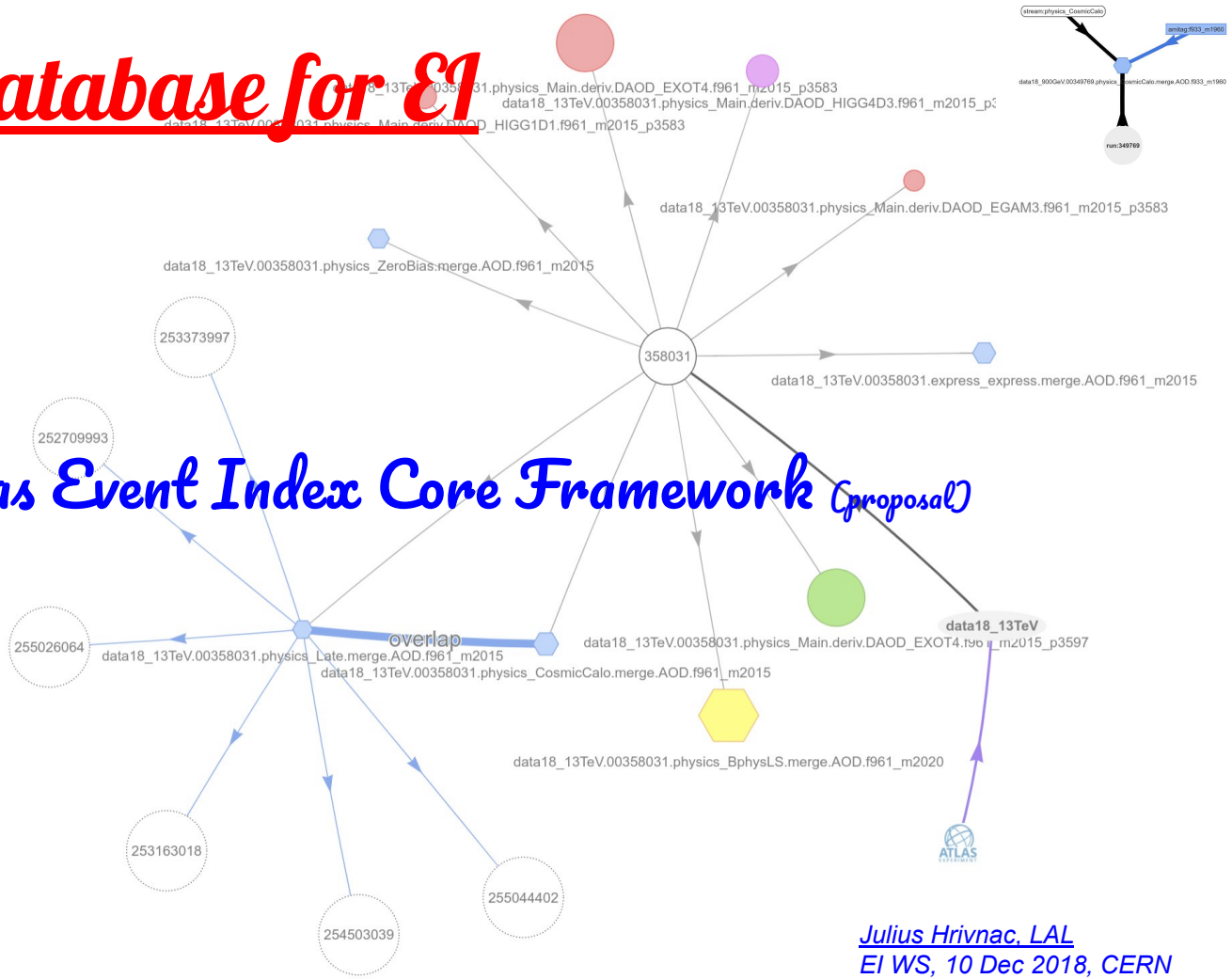
Graph Database for EI



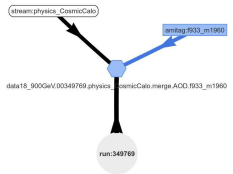
Evolution of the Atlas Event Index Core Framework (proposal)

Work in Progress

Julius Hrivnac, LAL

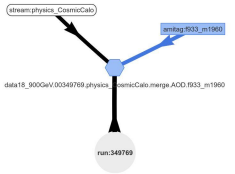
EI WS, 10 Dec 2018, CERN

Graph Databases for HEP



- Traditional data structures in HEP:
  - tuples (tables)
  - trees
  - nested tuples (trees of tuples)
  - relational (SQL-like)
- Schemafull or schemaless
- **But many of our data are graph-like** & schemaless
  - **Entities with relations**
  - G = (V, E) # *graph = (vectors, edges)*
- Not well covered by relational (SQL) databases
  - We don't need only a possibility to add new data with pre-defined relations
  - We need to add new relations
- Graph databases exist since a long time
  - Matured only recently thanks to Big Data & AI (adaptive NN)
  - Very good implementations & (de-facto) standards available
- The difference between SQL and Graph DB 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
- GraphDB is a synthesis of OODB and SQLDB
  - Expressing web of objects without fragility of OO world
  - Capturing only essential relations, not an object dump
- Moving structure from data to code
  - Together with migration from imperative to declarative semantics
  - Things don't **happen**, but **exist**

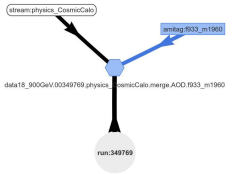
Graph Databases for Event Index ✔



- Original 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)
    - We have in fact implemented a poor-man's GraphDB on top of HBase
- Graphical WS presenting data as graphs

Graph Databases ✔

Standards & Choices



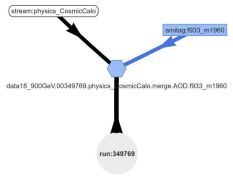
Functional syntax with additional navigational semantics !

- De-facto standard language/api: **Gremlin**
  - Gremlin is a functional, data-flow language to **traverse a property graph.** 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
  - Other candidate: Neo4j, the same Gremlin interface, used by Atlas Geometry DB
- Chosen visualisation: **visj.org**

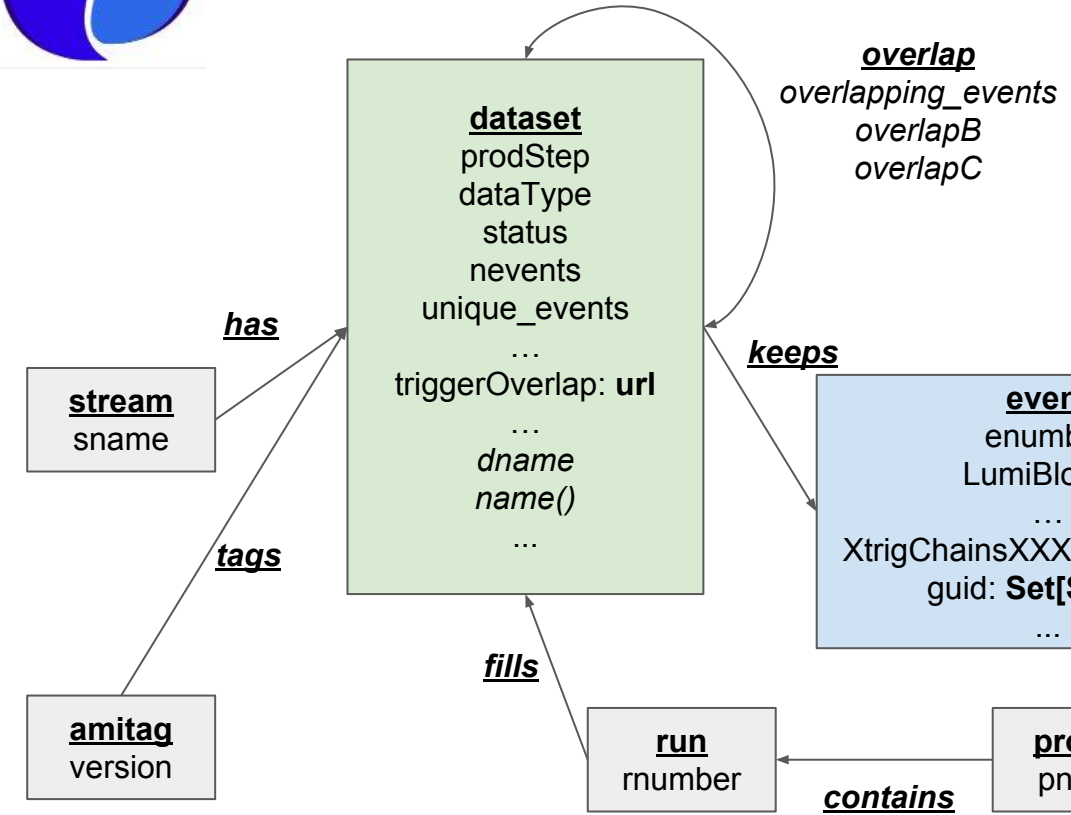
JanusGraph

Schema

Current status



Just some structure on top of the current schema, no big schema change (yet).

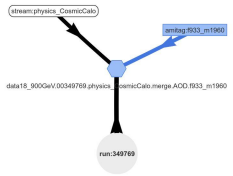


- Should decide what is a feature and what is a relation
- Arrows have only logical meaning, they can be navigated equally from both sides
- Relations and features have defined multiplicities (not shown here) and types (int, string, date,..., Set[...],...)
- Defined entities (and combinations of them)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

https://atlas-event-index.cern.ch/GraphDB-doc/Gremlin/schema.gremlin

Schema

Realisation



Operations on View.

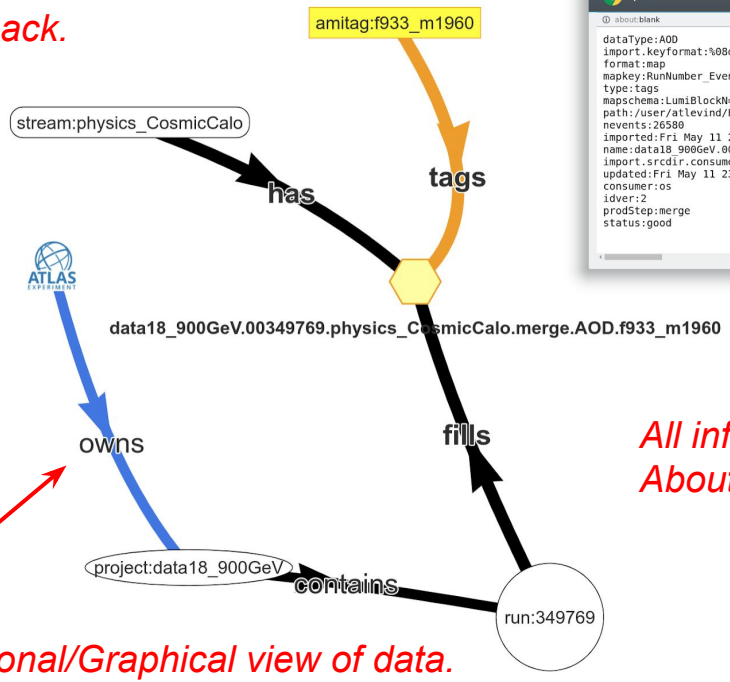
Context-sensitive action on selected element(s).

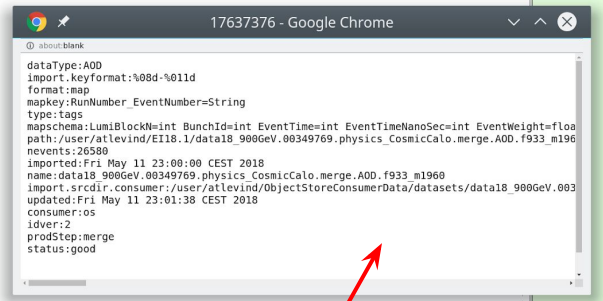
Customize the interactions with the **graph**.
Cluster by group type | Cluster by group size | Expand all clusters
live | expand children | expand parents | remove old

data18_900GeV.00349769.physics_CosmicCalo.merge.AOD.f933_m1960 Remove Describe
Catalog - Sample - Info -

Expanding data18_900GeV.00349769.physics_CosmicCalo.merge.AOD.f933_m1960 # Showing 0 new elements # Showing 0 new elements # Showing 3 new elements #

Operation feedback.





All information
About selected element.

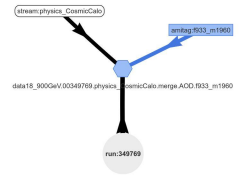
--- commands output ---

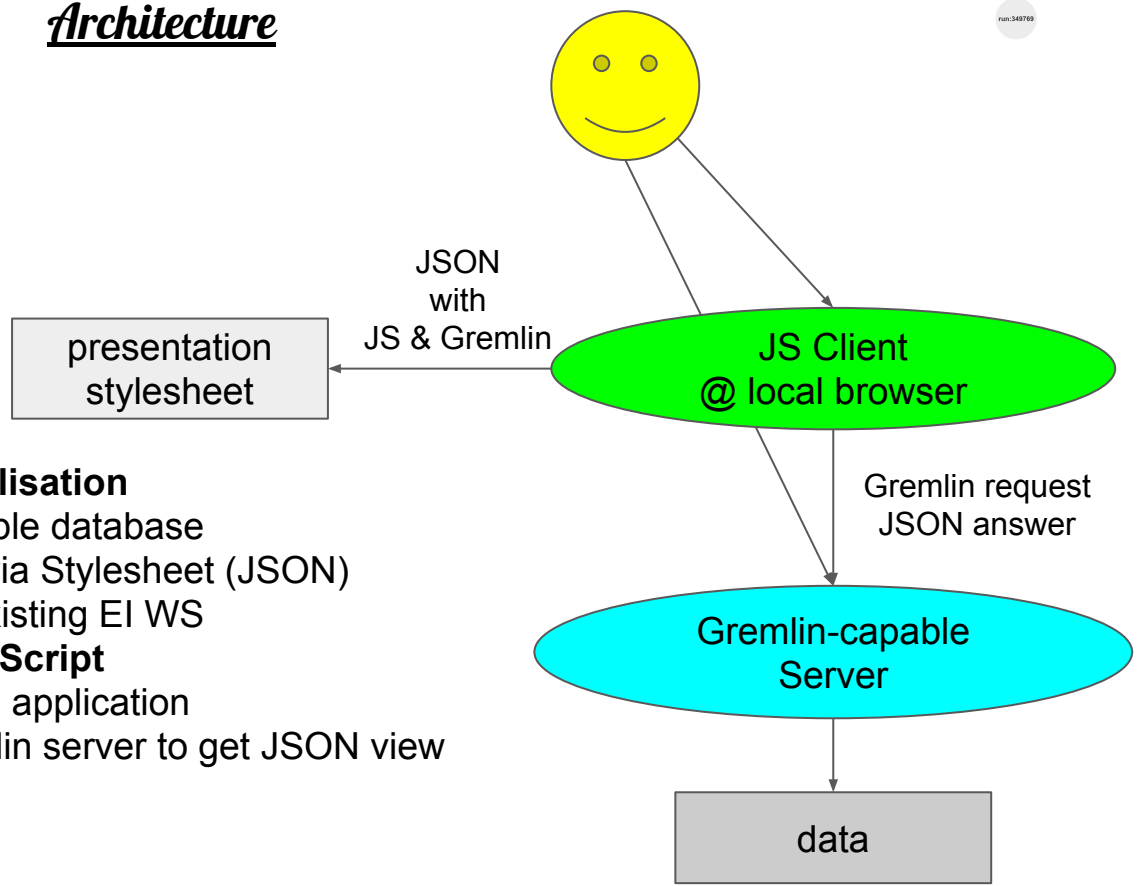
Context-sensitive action command output.

InterActive Relational/Graphical view of data.

GUI WS

Architecture

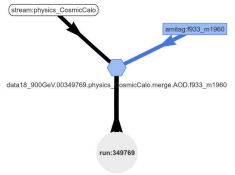




- Using standard Graph DB
  - Importing data from EI
- **Generic Web Service graphical visualisation**
  - Can display any Gremlin-compatible database
  - Visualisation can be customised via Stylesheet (JSON)
    - To give the same L&F as existing EI WS
  - Implemented **completely in JavaScript**
    - So doesn't need server-side application
    - Connects to standard Gremlin server to get JSON view of data

GUI WS

Stylesheet

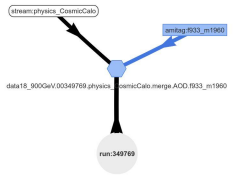


how to present "dataset" vertex

how to show it
(can contain Gremlin or JS code)

```
"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=dataset:"},
        {name:"Info" ,   url:"https://atlas-event-index.cern.ch/EIHadoop/InspectView.jsp?view=txt&action=info&climit=1&limit=10&query=dataset:"}
        ]
      },
```

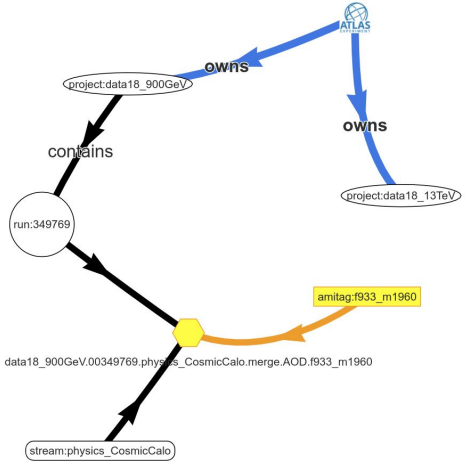
external actions

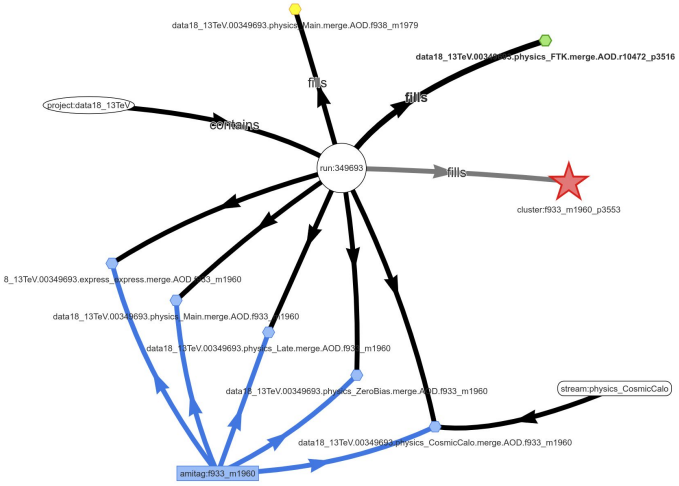


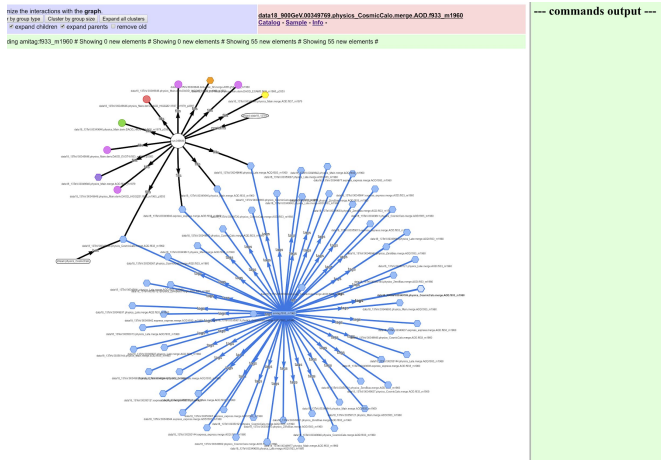
Implementation Status

- Using HBase backend @aiatlas016 (thanks to Zbyshek)
- Subset of data imported (full Catalog, some datasets and overlaps)
- Part of existing EI functionality implemented
- Most of the graphical part implemented
  - By standalone JS implementation

Using already IT-supported infrastructure

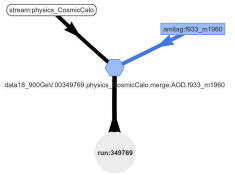






Gremlin Syntax

Groovy style

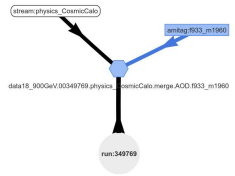


- *Functional syntax*
- *Functional & **navigational** semantics*

```
# 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'))
# a function deriving a dataset name (which is not stored as such)
# from existing dataset relations by traversing the graph
def datasetName(d) {
  return d.sideEffect(values("prodStep").store("4"))
          .sideEffect(values("dataType").store("5")).in()
          .sideEffect(hasLabel("run").values("rname").store("2"))
          .sideEffect(hasLabel("amitag").values("version").store("6"))
          .sideEffect(hasLabel("stream").values("sname").store("3"))
          .sideEffect(hasLabel("run").in().hasLabel("project").values("pname").store("1"))
          .cap("1", "2", "3", "4", "5", "6").next().values().join().toString()
  }
```

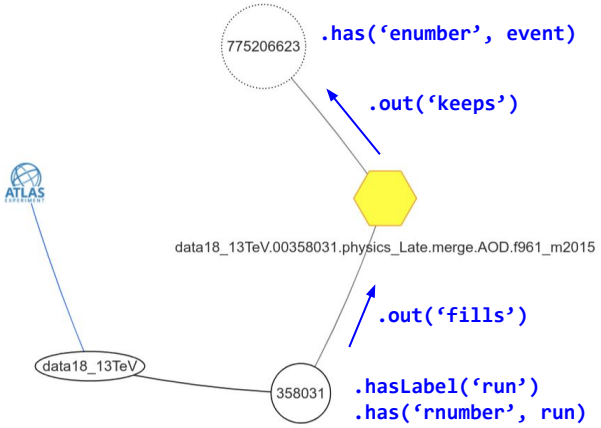
Event Lookup

Groovy style



- Both search and traversal steps
- Search steps can be boosted by indexes
- Functions loaded @ Janus server

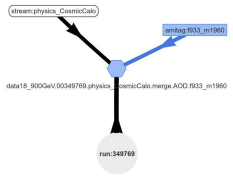
```
# Event-Lookup function (loaded inside the server)
def el(run, event, g) {
  e = g.V().hasLabel('run')          # all runs
           .has('rnumber', run)      # selected run
           .out('fills')             # all datasets filling that run
           .out('keeps')             # all events kept 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
```

Trigger Statistics

Groovy style



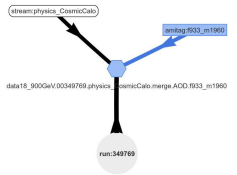

```
# Trigger Statistics function (loaded inside the server)
def tstat(dataset, chains, g) {
  return g.V().hasLabel('dataset')   # all datasets
              .has('dname', dataset) # selected dataset
              .out('keeps')          # all events kept in the dataset
              .values(chains)        # selected trigger chain
              .groupCount()          # accumulate counts
              .unfold()              # present result
  }

gremlin> tstat('data18_13TeV.00358031.physics_Late.merge.AOD.f961_m2015', 'L1trigChainsTAP', g)
==>L1_J30_FIRSTEMPTY=13882
==>L1_RD2_BGRP12=27631
==>L1_RD0_BGRP11=3589
==>L1_RD0_BGRP10=13
==>L1_J30_EMPTY=12709
...
```

The command is fast (data are in HBase & use tipical JanusGraph task)
=> no need to store TStat tables
=> can be combined with additional constraints (LB,...),...

Other Examples

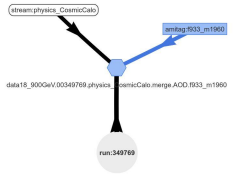
Java API




```
// 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”)
```

- *Very intuitive, **no special syntax needed**, easy integration.*
- *Database just accessed as a Java object with a structure.*
  - *Nested collections with links.*
- *Can use Java functional API (streams) and Lambda.*
- *No **semantic mismatch**.*
- *Similar for other supported languages (Python, Scala, Go,...).*

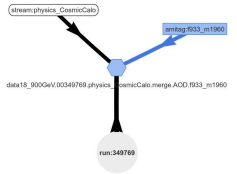
Performance



- 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
- In general:
  - **Very fast retrieval & Slower import**
  - Because import creates structures
    - Which are used in retrieval (simpler & faster)

Performance

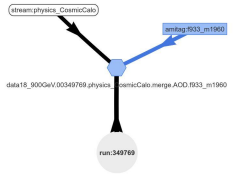
Event Lookup - 1




```
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
```

Performance

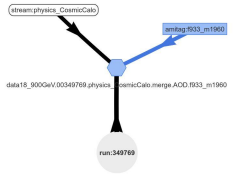
Event Lookup - 2



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	-

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 probably 10x faster (even on different event)

EI Evolution



Client

Apache/Tomcat

EI Core

Hadoop/HBase

➢ Data (almost) without structure

➢ **Complex code** (error-prone)

Client

EI Core (GraphDB)

JanusGraph,...

HBase

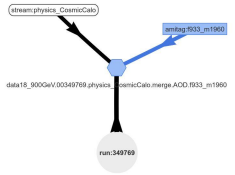
.....

➢ **Structured data**

➢ Interpreted by JanusGraph

Secondary Structures

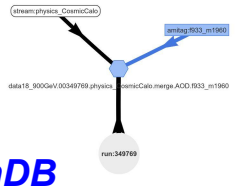
Stats, overlaps,...



- Trigger statistics:
  - Can be quickly calculated
  - That gives new possibilities
- Trigger overlaps:
  - The same as today: should be pre-calculated and stored
- Dataset overlaps:
  - Calculated during import
  - Stored within graph structure (as relations)
- Multiple events:
  - Found automatically during import
    - Constraint violation

Added Values

wrt current implementation

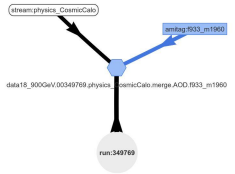


Once data are in GraphDB

- ***Access code is really simple***
- ***And fast***

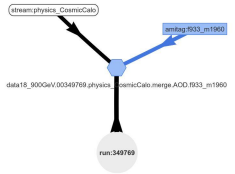
- **Big part of the current Core absorbed in GraphDB structure**
  - Delegate implementation/optimisation details to suitable framework
  - In past, we had in fact implemented our own GraphDB
    - Common development pattern in HEP :-)
- Apache/Tomcat service not needed
  - JavaScript client connects directly do Gremlin server
- No special CLI API needed
  - Standard Gremlin functions used
    - Can provide command wrappers for backward compatibility
- Using standards
  - So components can be replaced
    - JanusGraph with Neo4J
    - Hadoop with Cassandra
- New interface with the same (and enhanced) functionality
  - Also for WS
- Same or better performance
  - As the internal DB structure is very similar + better code

New Possibilities



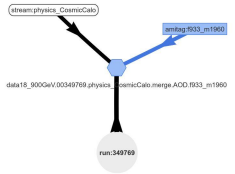
- Creation of Virtual Entities
  - Virtual Datasets
    - White Board functionality
    - Can create API allowing users creating their own (or group-wise) Virtual Datasets
  - Requests results
    - As in the current architecture, where any results is a new TagFile
  - Query Spaces
    - Part of original project
- Using GraphDB to store actual data
  - Today limited to Trees
  - Graphs are more appropriate
- Using together with Spark for more sophisticated analyses

Problems
(not serious)



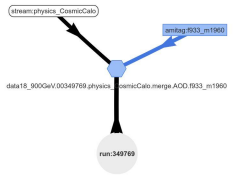
- Import is slower (x100 Hz)
  - Because it creates structure, which then makes searching faster
    - E.g. TrigStat generation
  - Simple import (no structure) cca 10x faster
  - There are ways to speed it up
    - Bulk-load without transaction
    - Bypass correctness checks
      - But they are important, e.g. to find multiple events
    - Load via Hadoop MapReduce
    - Load in parallel
    - Indexing after import
    - Optimise import (avoid repetitive searches for connected elements,...)
- Tricky coexistence with Hadoop in one application
  - Many overlapping libs with different versions

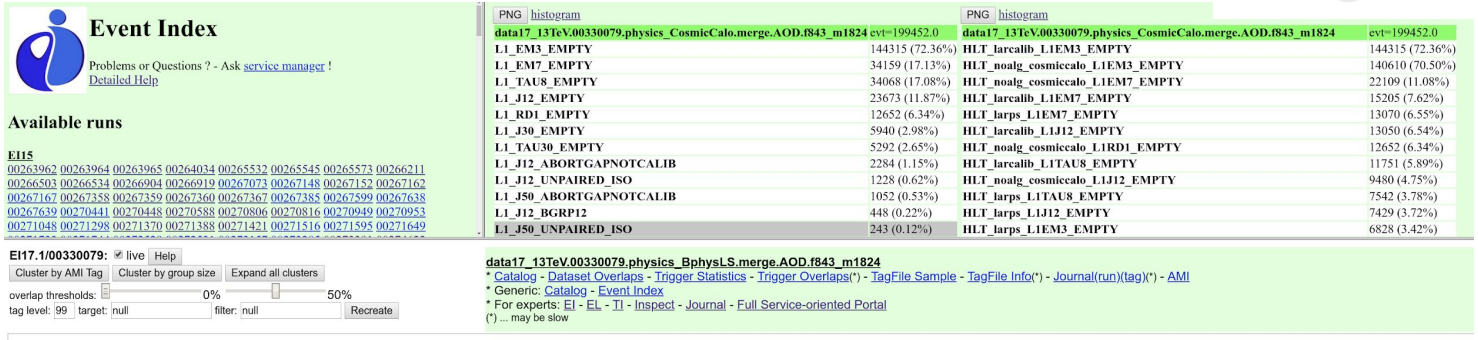
Plan



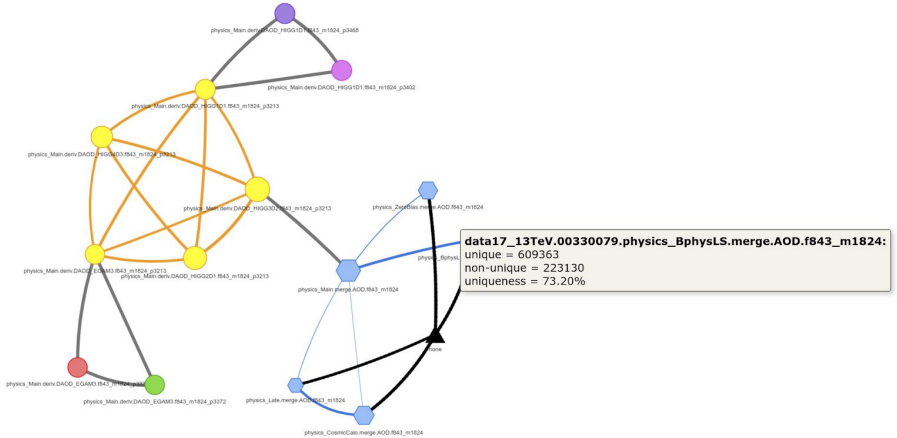
- Reimplement existing & requested functionality in GraphDB
  - Event Lookup, Overlaps & Statistics (¾ done)
  - + logging, journalling, monitoring, testing, ...
  - Easier than thought at the beginning
- Further performance study & profiling
  - For schema optimisation
  - Trigger ?
- Develop new functionality
  - User-created entities
  - Result caching
  - WB
- Develop Java code for migration of existing data
  - Currently in Gremlin/Groovy
  - Reimplement the Import in Java (90% done)
- Improve Trigger handling/decodingt
- Implement multiple events as related Virtual Dataset
- Migrate old data
- Expose Gremlin Server
- Make a Switch

Existing Relational WS



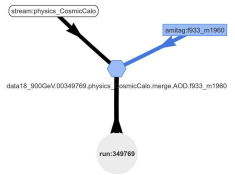


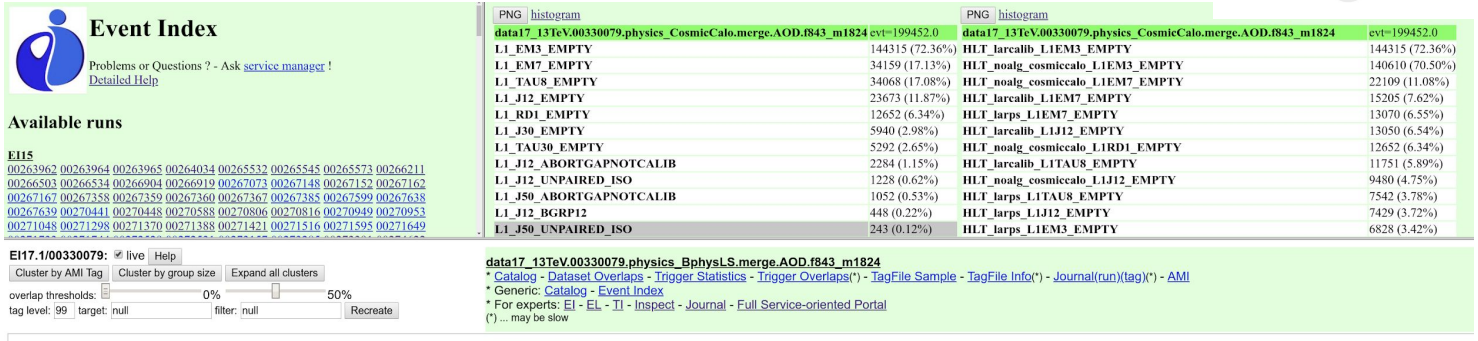
- *In production*
- *Simple data interpreted as a Graph by code*



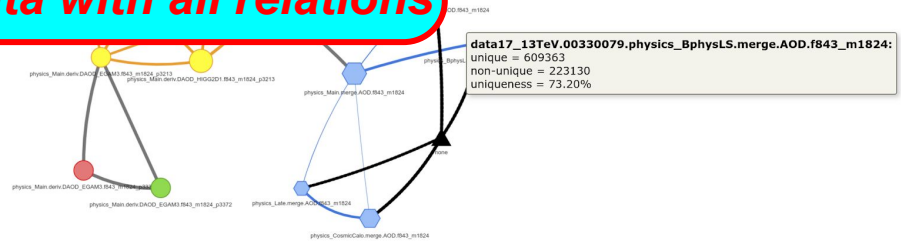
[Data-oriented Event Index WS](https://atlas-event-index.cern.ch/EIHadoop)

Existing Relational WS



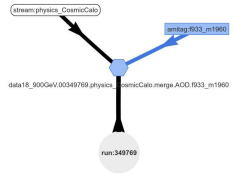


- *In production*
- *Simple data interpreted as a Graph by code*
- ***To be replaced with GraphDB & generic browser***
- ***Ultimate Aim: Global View of Atlas data with all relations***



Data-oriented Event Index WS

https://atlas-event-index.cern.ch/EIHadoop



Try It !

- Growing documentation: https://atlas-event-index.cern.ch/GraphDB-doc
- Web GUI (should work inside CERN): http://aiatlas016.cern.ch/GraphDB

A 'playground' has been installed @aiatlas016.
It contains the full Catalog, several datasets and DOverlap tables.
To play with it: connect to aiatlas016 as atlevind
(or checkout https://:@gitlab.cern.ch:8443/atlas-event-index/GraphDB.git).

```
$ cd work/DB/GraphDB/ant
$ source setup.sh
$ gremlin_local
```

And try some examples from
https://atlas-event-index.cern.ch/GraphDB-doc/Gremlin/examples.gremlin
https://atlas-event-index.cern.ch/GraphDB-doc/Gremlin/functions.gremlin
https://atlas-event-index.cern.ch/GraphDB-doc/Gremlin/tests.gremlin