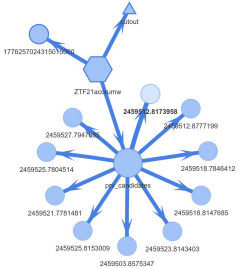
Using Graphs to Organize Fink Alerts



Storage, organization and analyses of the Alerts
stored in the **Fink** Broker environment
using Hybrid **Graph** database and Graph theory algorithms.

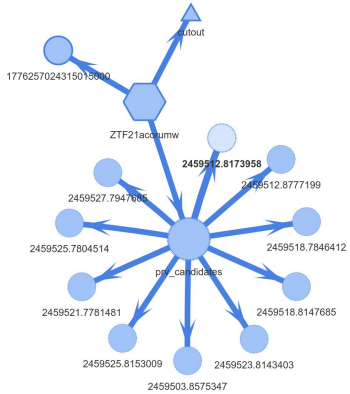
1. Hybrid Graph Alert storage
2. Where it can help in deep analyses

Julius Hrivnac, Julien Peloton for Fink @ IJCLab
LAPP, 07/Jun/2023

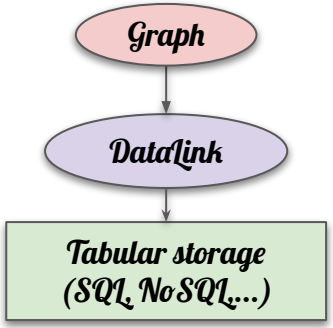
Fink Data Storage

- The original implementation = one big HBase table + a set of small HBase index tables for fast search

- The **Graph** implementation:
- The HBase table is converted into Graph
  - Rows become Vertices
  - Relations become Edges between Vertices
    - Which are now explicite, directly stored in the database
- Structure and relations are moved from the code to the storage
- Both Vertices and Edges have properties
  - Some are defined in a Schema, others can be freely added
  - Also new Vertices and Edges can be added and modified
- Indexes may be attached to any property for faster search
- Graph DB is slower on injection, similar on search, very fast on navigation, very slow on deletion



- The **Hybrid** architecture = all coming alert data are stored in HBase tables
- The alert data structure is created in the JanusGraph
  - Contains also the most important attributes
  - Has datalinks to HBase data

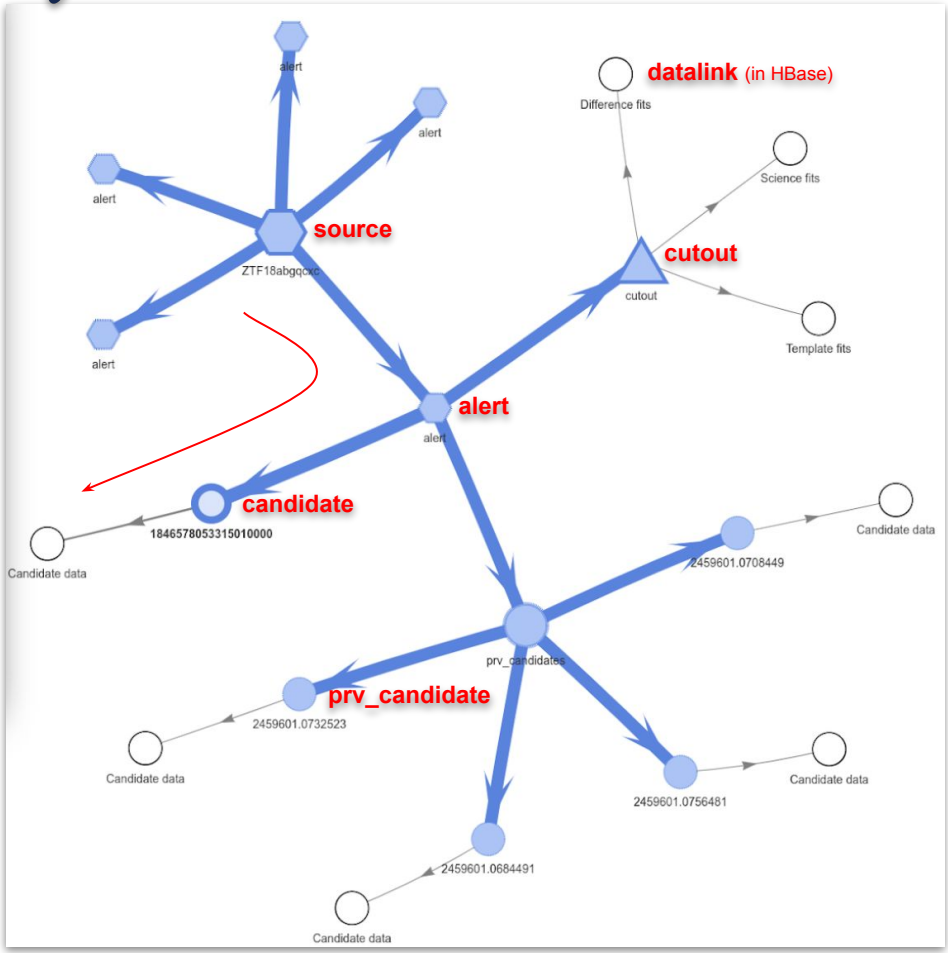


Gremlin Query Language

- Data can be accessed via **Gremlin query language**
  - Based on Groovy language
  - **Functional-style** syntax, where function = relation and function execution = relation navigation
  - Available for almost all major programming languages
    - But best suited for languages, which are already naturally functional
  - Simple searches are very intuitive, but sophisticated operation are possible
  - Allows complex graph navigation, mathematical and statistical operations and full functional processing of graphs

```
g.V().has('objectId', 'ZTF18acgkldj').out().has('lbl', 'candidate').valueMap('classtar', 'jd', 'direction')
```

Data Architecture




```
// Get data from Graph
g.V().has('lbl',      'source'      ).
      has('objectId', 'ZTF18abmfxvf').out().
      has('lbl',      'alert'       ).out().
      has('lbl',      'candidate'   ).
      valueMap('classtar', 'jd', 'direction')
{classtar=[1.0],   direction=[POINT (-92.808682 41.394857)], jd=[2459484.926331] }
{classtar=[1.0],   direction=[POINT (-92.808593 41.394903)], jd=[2459496.94
{classtar=[0.991], direction=[POINT (-92.808662 41.394983)], jd=[2459530.90
...
```

valueMap()

candidate:...gle Chrome

about:blank

candidate:1735390904215015000

```
aimage:0.814
aimagerat:0.24154302
bimage:0.733
bimagerat:0.21750742
candid:1735390904215015001
chinr:1.125
chipsf:395.6072
classtar:1.0
clrcoeff:-0.058636
clrcounc:7.9711E-6
clrmed:0.58
clrrms:0.141181
decnr:41.3949221
diffmaglim:20.63922
direction:POINT (-92.808583 41.394918)
distnr:0.1684893
distpsnr1:0.15476647
distpsnr2:0.90726954
distpsnr3:2.7971349
drb:0.6651443
drbversion:d6_m7
```

```
// Get data (from HBase) attached to 'candidate's
g.V().has('lbl',      'source'      ).
      has('objectId', 'ZTF18abmfxvf').out().
      has('lbl',      'alert'       ).out().
      has('lbl',      'candidate'   ).out().
      has('lbl',      'datalink'    ).
      each {
        println(FinkBrowser.getDataLink(it))
        }
```

Gremlin for Fink

- Frequently used and typical queries are implemented as server-side function to be available to all clients
- Typical user request:
  - Server-side selection function
  - + Further refining selection
  - + Set of values to return
  - + Further math or graphics
- Any Gremlin code is possible
  - With some kind of user authentication and authorisation

```
g.V().has('lbl',      'source'      ).
      has('objectId', 'ZTF18abmfxvf').out().
      has('lbl',      'alert'       ).out().
      has('lbl',      'candidate'   ).
      valueMap('classtar', 'jd', 'direction')
{classtar=[1.0],   direction=[POINT (-92.808682 41.394857)], jd=[2459484.926331] }
{classtar=[1.0],   direction=[POINT (-92.808593 41.394903)], jd=[2459496.9461458]}
{classtar=[0.991], direction=[POINT (-92.808662 41.394983)], jd=[2459530.9093403]}
...
```

server-side selection functions

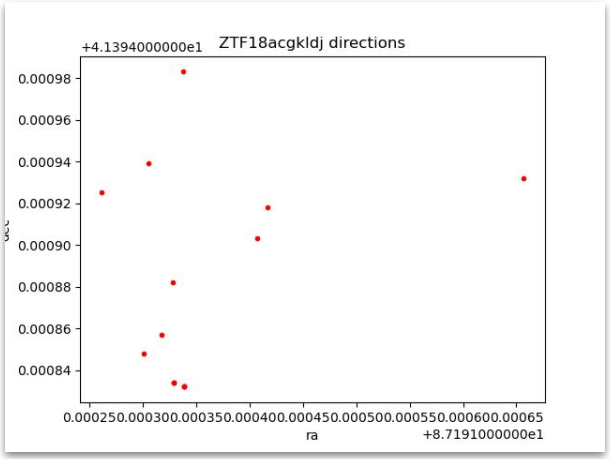
```
candidates('ZTF18acgkldj').valueMap('classtar', 'jd', 'direction')
```

```
// gives 10 first candidates 0.1 degree around direction 57.5 x -1.97 between two jd times
// implemented as a server-side function
geosearch(57.5, -1.97, 0.1, 2359300.7629977, 2559317.7015982, 10).has('lbl', 'candidate').valueMap(...)
// internally contains (with protection against overuse and optimisation code):
g.V().has('direction', geoWithin(Geoshape.circle(dec, 180 - ra, dist))).has('jd', inside(jdmin, jdmax))
```

Python API

```
import sys

import jpype
import jpype.imports
from jpype import JImplements, JOverride, JImplementationFor

import matplotlib.pyplot as plt

# ../dist/FinkBrowser.exe.jar
jpype.startJVM(jpype.getDefaultJVMPath(), "-ea", "-Djava.class.path=" + sys.argv[1], convertStrings=False)

from com.Lomikel.Januser                  import StringGremlinClient
from com.astrolabsoftware.FinkBrowser.Utils import Init

Init.init()

client = StringGremlinClient("graph-server", 24444);

results = client.interpret("candidates('ZTF18acgkldj').elementMap('direction')");

ra  = []
dec = []

for element in results:
  dec += [element.getObject().get("direction").getPoint().getLatitude()]
  ra  += [element.getObject().get("direction").getPoint().getLongitude() + 180]

plt.plot(ra, dec, 'r.')
plt.title('ZTF18acgkldj directions')
plt.xlabel('ra')
plt.ylabel('dec')
plt.show()
client.close()

jpype.shutdownJVM()
```

simple Python Example



Direct|String API

```
# instead of
client = StringGremlinClient("graph-server", 24444);
results = client.interpret("candidates('ZTF18acgkldj').elementMap('direction')");

# we can do
client = DirectGremlinClient("graph-server", 24444);
g = client.g();
query = g.V().has('lbl', 'alert').limit(4).values(objectId);
results = client.submit(query);
# advantage: results is an actual object,
#            while above it was just a string with JSON content
# problem: cannot use server-side functions and objects,
#          which are unknown to client
```

End User Access

- All queries can be issued using standard Gremlin clients (in all popular languages)
- Requests can be send directly to Gremlin server
- **A special client** also available in several incarnation, providing some pre/post-processing, overuse protection and connection handling
  - Java executable
  - Linux native executable
  - GUI (platform independent)
  - REST Web Service
  - Python, Java, Scala, Groovy,... API
  - Jupyter API
- The same answer in CLI and from REST WS

```
# Direct connection to Graph server (gives very verbose JSON answer, not all queries supported)
curl 'http://graph-server:24444/gremlin'
  -XPOST -d '{"gremlin":"candidates(\"ZTF18acgkldj\").valueMap(\"classtar\", \"jd\", \"direction\")"}'
# Connection to Fink server
curl 'http://fink-server:8080/FinkBrowser/Fremlin.jsp'
  -get --data-urlencode 'gremlin=candidates("ZTF18acgkldj").valueMap("classtar", "jd", "direction")'
# Java client
java -jar FinkBrowser.exe.jar --gremlin 'candidates("ZTF18acgkldj").valueMap("classtar", "jd", "direction")'
# Native Linux client
FinkBrowser.exe --gremlin 'candidates("ZTF18acgkldj").valueMap("classtar", "jd", "direction")'
```

GQL (Cypher)

- SQL-like (declarative) graph query languages developed by Neo4J
- GQL can be run on top of Gremlin
  - Not the other way around

Gremlin

```
g.V().has('objectId', 'ZTF18acgkldj').out().has('lbl', 'candidate').valueMap('classtar', 'jd', 'direction')
```

GQL

If you prefer SQL

```
(a) - [:contains:] - (b:candidate)
WHERE a.objectId = 'ZTF18acgkldj'
RETURN b.classtar, b.jd, b.direction
```

Graph Analyses

What to do with Graphs ?

- Accumulated Graph can be enhanced with additional (calculated) relations
- Those relation can be then analysed using Graph theory algorithms
  - And exported to external **Graph analyses toolkits**
  - Or used for **Graph Neural Network** searches

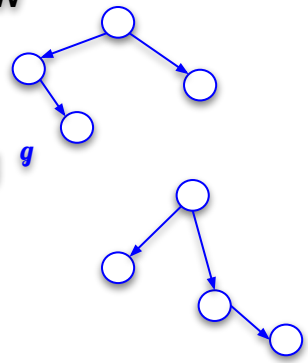
Graph Analyses of PCAs

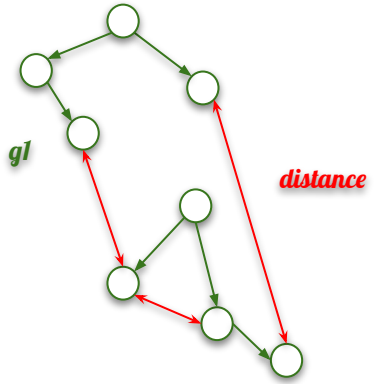
The First example of Graph use in Fink

- As alerts are only a limited view of each object (30 days maximum), they are first aggregated by ID to form longer light-curves
- From these light-curves, we extract a large set of features (statistical features, machine learning scores, external catalog labels), and compress this information by using a **principal component** analysis *(Julien)*
  - With the current set-up, the first 10 PCAs are generally enough to capture most of the relevant information, but a work is still ongoing regarding the definition of the input set of features.
- PCAs are then **included in the Graph** (as new Vertices) and connected to their sources

PCAs Clustering

1. Add derived information as new Vertices
   (here: PCAs)
2. Create a '**metric**' of interesting relations and store them as Edges
   (here: differences between PCAs)
3. Find features/patterns
   (here: clusters of 'close' PCAs)



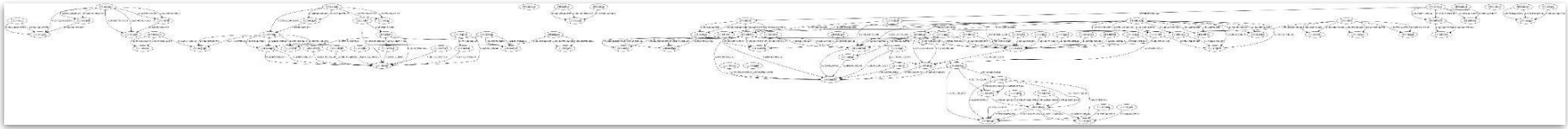


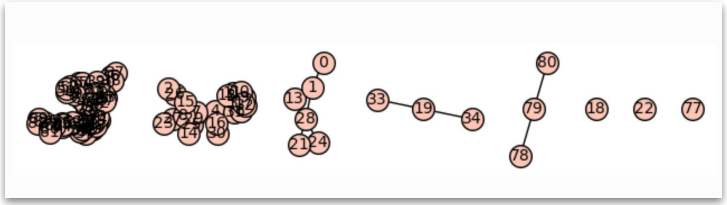

```
// Find all pairs of PCA Vertexes, which are close enough.
// Connect them with the Edge 'distance' having a 'difference' property equal to
// the calculated difference.
variables = 'pca00 pca01 … pca24'
difference = 'qdistane(variables,...) // quadratic distance
gr.structurise(g.V().has('lbl', 'PCA'), difference, variables, 'distance', 'difference', …)
// Get some statistics about newly created Edges.
g.E().hasLabel('distance').values('difference').union(min(), max(), sum(), mean(), count())
// Find clusters of 'close' PCAs.
FinkBrowser.findPCAClusters(g, 'distance', 'difference', 2, 10.5)
```

Exporting Graph

```
// Create a new personal Graph.
graph1 = Lomikel.myGraph()
// Get the entry point to the Graph traversal.
g1 = graph1.traversal()
// GremlinRecipies is a class with various useful Gremlin methods.
gr = new GremlinRecipies(g)
// Get 'source' Vertexes from the main Graph (automatically available as 'g') and
// clone them in the private Graph 'g1'.
g.V().has('lbl', 'source').each {source ->
  gr.gimme(source, g1, -1, -1)
  }
// Export the graph into GraphML file, to be read into graph tools.
graph1.io(IoCore.graphml()).writeGraph('exported.graphml')
```

- Create a ‘private’ subgraph
- Analyse it with Gremlin
- Or export it and analyse it in external graph toolkit





in SageMath

in GraphViz

Links

- Main ideas:
  - **Use Graph DB to provide flexibility**
  - **Expose directly Gremlin query language**
    - **In API, CLI, REST**
  - **Provide server-side functions with requested functionality**
  - **Enhance Graphs with calculated relations**
  - **Analyse using Graph Theory Algorithms**
- More info about Graph databases:
  - Using Graph Databases (CHEP 2019 talk)
  - Multidatabase (CHEP 2023 talk)
  - Gremlin Query language